

City Research Online

City St George's, University of London

Citation: Ibrahim, D. (1980). A bit-slice user microprogrammable microcomputer: Design and applications to real-time digital filtering. (Unpublished Doctoral thesis, The City University)

This is the accepted version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

Permanent repository link: https://openaccess.city.ac.uk/id/eprint/37217/

Copyright and Reuse: Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to City Research Online policy.

City Research Online: http://openaccess.city.ac.uk/ publications@citystgeorges.ac.uk

A BIT-SLICE USER MICROPROGRAMMABLE MICROCOMPUTER : DESIGN AND APPLICATIONS TO REAL-TIME DIGITAL FILTERING

Volume 1

-by-

Dogan Ibrahim, B.Sc., M.Sc

Thesis presented for the Degree of Doctor of Philosophy

The City University
Department of Electrical and Electronic Engineering
September 1980

CONTENTS

		Page
Acknowledgements		10
Declaration		11
Abstract		12
List of Symbols and Abbreviations		13
Chapter 1.	INTRODUCTION	17
Chapter 2.	PRINCIPLES OF MICROPROGRAMMING	24
2.1	Introduction	24
2.2	Wilkes' Model of a Microprogrammed Control Unit	27
2.3	Evolution of Microprogramming	30
2.4	Example on Microprogramming	32
Chapter 3.	THE BIT-SLICE DEVICES	36
3.1	Introduction	36
3.2	The Basic Elements of a Bit-Slice Processor	36
3.3	Commercially Available RALU Elements	37
3.3.1	Intel 3002	39
3.3.2	Fairchild 9405	42
3.3.3	Advanced Micro Devices Am2901	42
3.3.4	Monolithic Memories 6701	43
3.3.5	Texas Instruments SBP0400	43
3.3.6	Texas Instruments 74S481	44
3.3.7	Motorola MC10800	44
3.3.8	Advanced Micro Devices Am2903	45
3.3.9	Fairchild F100220	45
3.4	Commercially Available Sequencers	46
3.4.1	Texas Instruments SN74S482	48
3.4.2	Advanced Micro Devices Am2909/11	48
3.4.3	Motorola MC10801	48
3.4.4	Signetics 8X02	49
3.4.5	Monolithic Memories 67110	49
3.4.6	Intel 3001	50
3.4.7	Fairchild 9408	50
3.4.8	Advanced Micro Devices Am2910	51
3.5	Advantages of the Bit-Slice Processors	51

Page

3.6 Disadvantages of the Bit-Slice Processors 53
3.7 Conclusions 54

Chapter 4. DESIGN OF THE BIT-SLICE MICROCOMPUTER 55
4.1 Introduction 55
4.2 The Block Diagram of the Microcomputer 55

Chapter 5. DESIGN OF THE ARITHMETIC PROCESSING UNIT (APU) 60
5.1 Design of the Arithmetic and Logic Unit (ALU) 62
5.1.1 The Am2903 63
5.1.2 Cascading the Am2903s 70
5.2 Design of the Bit Manipulation Logic (BML) 81
5.3 Design of the Data Bus Buffers 88
5.3.1 Connecting the Y Port Lines of the Am2903s 89
5.3.2 Connecting the DB Port Lines of the Am2903s 92
5.4 Design of the Register Extension Logic (REL) 95
5.4.1 Examples on Microcoding the REL 97
5.5 Design of the Condition Code Select Logic (CCS) 99
5.5.1 Controlling the Condition Code Select Logic 102
5.5.2 Examples on Microcoding the CCS 108
5.6 Design of the Carry in Control Logic 109
5.7 Design of the Data Shifting Logic (DSL) 110
5.7.1 Examples on Microcoding the DSL 117
5.8 Conclusions 119

Chapter 6. DESIGN OF THE INPUT-OUTPUT UNIT 126
6.1 Introduction 126
6.2 The Design of the I/O Unit 127
6.3 The Number of I/O Ports 132
6.4 Design of the Data Acquisition Channel (Port 1) 134
6.4.1 Sample/Hold Amplifier 134
6.4.2 Analog to Digital Converter 137
6.4.3 Digital to Analog Converter 141
6.5 Design of the Serial Communication Channel (Port 6, Port 7) 143
6.5.1 Introduction 143

		Page
6.5.2	The Design	146
6.6	Conclusions	153
Chapter 7.	THE PROGRAM CONTROL UNIT (PCU)	154
7.1	Introduction	154
7.2	The Design of the PCU	157
7.2.1	Cascading and Control of the Am2911s	162
7.2.2	The Clock to the Am2911s	167
7.2.3	The Data Bus Registers	168
7.2.4	Selecting the 8/16 bits of Operation	168
7.3	Conclusions	170
Chapter 8.	THE MAIN MEMORY	173
8.1	Introduction	173
8.2	Design of the Main Memory Unit	175
8.3	Conclusions	186
Chapter 9.	THE INTERRUPT CONTROL UNIT	187
9.1	Introduction	187
9.2	Interrupt Structures	189
9.2.1	Single-Level, Multiple Poll Interrupt Structure	189
9.2.2	Single-Level, Daisy Chain Interrupt Structure	191
9.2.3	Multilevel Interrupt Structure	194
9.2.4	Multilevel Daisy Chain Interrupt Structure	196
9.3	Design of the Interrupt Control Unit (ICU)	196
9.3.1	Interrupt Latches	202
9.3.2	Mask Registers	204
9.3.3	Priority Encoder	206
9.3.4	Address Decoder	209
9.3.5	Timer Registers	212
9.3.6	Timer Counters	214
9.3.7	Interrupt Clear Logic	217
9.3.8	Interrupt Control Logic	220
9.4	Conclusions	222

		Page
Chapter 10.	THE TIMING UNIT	224
10.1	Introduction	224
10.2	Design of the Timing Unit	226
Chapter 11.	THE COMPUTER CONTROL UNIT (CCU)	231
11.1	Introduction	231
11.2	Typical Architecture of a 16 bit Microprogrammable CCU	232
11.3	An Improved CCU	235
11.4	Design of the CCU	238
11.4.1	The Instruction Register	238
11.4.2	The Operand Select Logic	241
11.4.3	The Microprogram Sequencer	243
11.4.4	The Mapping Memory	250
11.4.5	The Microprogram Memory Width	258
11.4.6	Choosing a Microprogram Format	259
11.4.7	Design of the Microprogram Memory	262
11.4.8	The Pipeline Register	267
Chapter 12.	MICROPROGRAMMING THE MICROCOMPUTER	270
12.1	Introduction	270
12.2	Instruction Formats	270
12.2.1	Single Word Instructions	273
12.2.2	Two Word Instructions	275
12.2.3	Three Word Instructions	275
12.3	The Microcode	276
12.4	The Mapping PROM Data	299
12.5	Mnemonics for the Macroinstructions	299
12.6	Conclusions	299
Chapter 13.	THE OPERATING SYSTEM (MONITOR)	305
13.1	Introduction	305
13.2	The UMM-16 Operating System	306
Chapter 14.	MINIMUM CYCLE TIME CALCULATIONS	311
14.1	Introduction	311
14.2	Speed Calculations in the Control Path	311
14.3	Speed Calculations in the Data Path	313
14.4	Speed Calculations in the Main Memory Path	315
14.5	Conclusions	317

		Page
Chapter 15.	A NOVEL TECHNIQUE FOR USER MICROPROGRAMMING (THE SUPPORT PROCESSOR)	319
15.1	Introduction	319
15.2	The Hardware of the Support Processor	321
15.2.1	The 3850 CPU	323
15.2.2	The 3853 SMI	323
15.2.3	The 3871 PIO	325
15.2.4	The RAM and the EPROM	326
15.2.5	Chip Select	328
15.2.6	Interfacing the Support Processor to the Bit-Slice Microcomputer	329
Chapter 16.	THE MONITOR OF THE SUPPORT PROCESSOR	333
16.1	Introduction	333
16.2	The Command 'NMIC'	335
16.3	The Command 'NMAP'	336
16.4	The Command 'LMIC'	338
16.5	The Command 'LMAP'	339
16.6	The Command 'M 1XX'	340
16.7	The Command 'T XX'	341
16.8	The Command 'E/mn/pq/'	342
16.9	The Command 'DC 1XX'	343
16.10	The Command 'DP XX'	343
16.11	The Command 'LD'	344
16.12	The Command 'H'	345
16.13	Other Features of the Monitor	346
16.14	Conclusions	346
Chapter 17.	THE THEORY OF DIGITAL FILTERS	349
17.1	Introduction	349
17.2	The z-Transform	351
17.3	The Difference Equation	355
17.4	Digital Filter Configurations	356
17.4.1	Nonrecursive Filter Structures	357
17.4.2	Recursive Filter Structures	357
17.5	Design of Infinite Impulse Response (IIR) Low Pass Digital Filters	363
17.5.1	Bilinear Transformation	365
17.5.2	A Design Technique for the Low Pass IIR Digital Filters	368

Page

17.6 Calculating the Digital Filter Coefficients Using a Programmable Pocket Calculator 377
17.6.1 The Butterworth Programme 377
17.6.2 The Chebyshev I Programme 380
17.7 Design of Finite Impulse Response (FIR) Digital Filters 382
17.7.1 Optimal FIR Digital Filter Design 386
17.8 Frequency Transformation of Low Pass Filters 389
17.9 Fractional Number Representation 393

Chapter 18. DESIGN OF REAL-TIME LOW PASS IIR DIGITAL FILTERS USING THE BIT-SLICE MICROCOMPUTER 396
18.1 Introduction 396
18.2 The Hardware Set-up 398
18.3 Computing the Filter Algorithm 398
18.3.1 A Second Order Section 400
18.3.2 A First Order Section 403
18.3.3 Cascading the Filter Sections 404
18.4 Choosing an Instruction Set 409
18.4.1 Instruction 'IN' 409
18.4.2 Instruction 'LPFL 2' 411
18.4.3 Instruction 'LPFG 2' 412
18.4.4 Instruction 'LPF 1' 413
18.4.5 Instruction 'OUT' 413
18.4.6 The Initialization Routine 414
18.5 Design Examples 416
18.5.1 First Order Butterworth Filter 416
18.5.2 First Order Chebyshev I Filter 418
18.5.3 Second Order Butterworth Filter 420
18.5.4 Second Order Chebyshev I Filter 424
18.5.5 Third Order Butterworth Filter 425
18.5.6 Third Order Chebyshev I Filter 428
18.5.7 Fourth Order Butterworth Filter 430
18.5.8 Fourth Order Chebyshev I Filter 433
18.5.9 Sixth Order Butterworth Filter 436
18.5.10 Eighth Order Butterworth Filter 440
18.5.11 Design of Higher Order Filters 441
18.6 Maximum Sampling Frequency for a Given Order 443

		Page
18.7	The Microcode of the Filter Instructions	446
18.8	Conclusions	454
Chapter 19.	DESIGN OF REAL-TIME HIGH PASS IIR DIGITAL FILTERS USING THE BIT-SLICE MICROCOMPUTER	458
19.1	Introduction	458
19.2	Computing the Filter Algorithm	458
19.3	Choosing an Instruction Set	461
19.3.1	Instruction 'HPFL 2'	461
19.3.2	Instruction 'HPFG 2'	462
19.3.3	Instruction 'HPF 1'	463
19.4	Design Examples	464
19.4.1	First Order Butterworth Filter	464
19.4.2	Second Order Butterworth Filter	466
19.4.3	Second Order Chebyshev I Filter	469
19.4.4	Third Order Chebyshev I Filter	473
19.4.5	Fourth Order Butterworth Filter	474
19.4.6	Sixth Order Butterworth Filter	477
19.4.7	Eighth Order Butterworth Filter	482
19.5	Microcode of the High Pass Filter Instructions	484
Chapter 20.	DESIGN OF REAL-TIME FIR DIGITAL FILTERS USING THE BIT-SLICE MICROCOMPUTER	489
20.1	Introduction	489
20.2	Computing the FIR Filter Algorithm	489
20.3	A Novel Technique for the Implementation of FIR Digital Filters	492
20.4	Choosing an Instruction Set	494
20.5	Design Examples	498
20.5.1	Low Pass Filter with N=15 (Hamming Windowed Design)	500
20.5.2	Low Pass Filter with N=10 (Optimum Design Algorithm)	501
20.5.3	High Pass Filter with N=15 (Optimum Design Algorithm)	504
20.5.4	Band Pass Filter with N=15 (Optimum Design Algorithm)	507
20.5.5	Band Stop Filter with N=15 (Optimum Design Algorithm)	513

		Page
20.6	Microcode of the Instruction 'FIR'	514
20.7	The Maximum Sampling Frequency	517
20.8	Conclusions	520
Chapter 21.	CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK	523
21.1	Conclusions	523
21.2	Suggestions for Future Work	529
22.	REFERENCES	531
23.	BIBLIOGRAPHY	535
24.	PUBLICATIONS	538



The Bit-Slice User Microprogrammable Microcomputer

Designed by the Author

ACKNOWLEDGEMENTS

The author wishes to express his sincere thanks to Dr. A.C. Davies for the helpful advice given throughout the duration of this research. The author is also grateful to the Science Research Council and the SIRA Institute Ltd for financial support.

DECLARATION

I grant powers of discretion to the University Librarian to allow this thesis to be copied in whole or in part without further reference to me. This permission covers only single copies made for study purposes, subject to normal conditions of acknowledgement.

ABSTRACT

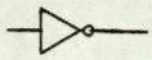
The subject of the research outlined in this thesis is to design a bit-slice microcomputer and to investigate its applications to real-time digital filtering, by inventing the necessary special purpose instructions. The microcomputer is designed such that operation with both 8 and 16 bits of data length are possible. The microprogram memory is designed from a combination of PROM and RAM type memories. Some of the PROM part is loaded with microinstructions which can execute a 16-bit fixed instruction set , with 62 general purpose instructions. The RAM part of the microprogram memory can be loaded with the user defined microinstructions. A low cost 8-bit microprocessor is used as a support processor. The support processor accepts the user microinstructions from the user terminal (TTY or VDU) and loads the RAM part of the microprogram memory. The bit slice microcomputer is supported by an operating system which is programmed by using the fixed instructions. The operating system allows the user to enter, modify, display, and execute their programs in the main memory, via a TTY or VDU. Thus the microcomputer can also be used as a general purpose 16-bit machine.

Special instructions are invented on the bit-slice microcomputer which greatly simplify the design of IIR type real-time digital filters. Both the low-pass and the high-pass filters are considered. The design of FIR type real-time digital filters are investigated by deriving a new algorithm. Special instructions are invented for the implementation of this algorithm on the bit-slice microcomputer.

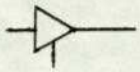
Two manuals are prepared by the author in support of the bit-slice microcomputer designed. These are included in Volume 2 of the thesis.

LIST OF SYMBOLS AND ABBREVIATIONS

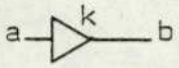
1. Symbols



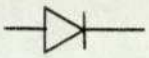
Inverter



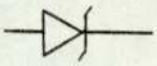
Three-state buffer



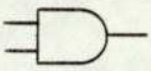
Multiplier by a constant (b = k.a)



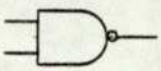
Diode



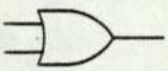
Zener diode



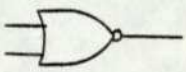
AND gate



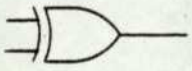
NAND gate



OR gate



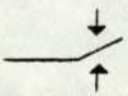
NOR gate



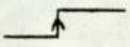
Exclusive-OR gate



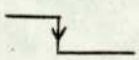
Socket



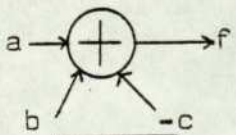
SPDT Analog switch



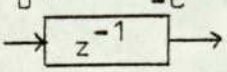
LOW-to-HIGH transition



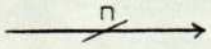
HIGH-to-LOW transition



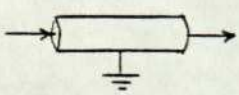
Adder (f = a+b-c)



Unit delay



n-bit connection



Screened cable

2. Abbreviations

A/D	Analog to digital converter
ALU	Arithmetic and logic unit
.AND.	Logical AND
APU	Arithmetic processing unit
BML	Bit manipulation logic
CCS	Condition code select logic
CCU	Computer control unit
C_n	Carry-in
C_{n+4} (or C)	Carry out
CP	Clock
CPU	Central processing unit
CS (or CE)	Chip select control input of memories
D/A	Digital to analog converter
DIL	Dual-in-line integrated circuit socket
D_n	n th bit of the scratchpad register
DSL	Data shifting logic
ECL	Emitter coupled logic
EPROM	Electrically reprogrammable, U.V. erasable semiconductor memory
.EX-OR.	Logical Exclusive-OR
FIR	Finite impulse response
f_d	Cutoff frequency of a low pass digital filter
f_h	Cutoff frequency of a high pass digital filter
f_r	Ripple cutoff frequency
f_s	Sampling frequency
HIGH	Logic '1' state
H'nnnn'	Hexadecimal data nnnn
ICU	Interrupt control unit
IIR	Infinite impulse response
Imm	Immediate data
I^2L	Integrated injection logic
I/O	Input-Output
LED	Light emitting diode
LOW	Logic '0' state
LSB	Least significant bit (bit 0)

LSI	Large scale integration
mI_{nnn}	Microinstruction at microprogram memory address nnn
MOS	Metal oxide semiconductor
μs	Microsecond
MSB	Most significant bit (bit 15 for a 16 bit data)
MSI	Medium scale integration
N	Sign status
ns	Nanosecond
.OR.	Logical OR
OVR	Overflow status
PC	Program counter register
PROM	Programmable read-only memory
Q	The 'Q' register of the Am2903
(R)	Contents of register R
$(\overline{R})$	Complement of the contents of register R
((R))	Contents of location addressed (pointed to) by register R
R_A	Scratchpad register addressed by the A field of the operand
R_A^1	Auxiliary register addressed by the A field of the operand
RALU	Register arithmetic logic unit
RAM	Random access memory (read/write memory)
R_B	Scratchpad register addressed by the B field of the operand
R_B^1	Auxiliary register addressed by the B field of the operand
R_D	Scratchpad register D
REL	Register extension logic
R_F	Scratchpad register F
ROM	Read-only memory
S	The status register
SP	The stack pointer
SSI	Small scale integration
TTL	Transistor-transistor logic
u	Discrete-time variable

UMM-16	Name of the bit-slice microcomputer designed (from: User Microprogrammable 16 bit Microcomputer)
VDU	Visual display unit
ω_a	Analog radian frequency
WE	Read/write control input of RAM type semiconductor memories
X	Don't care entry
z	z-transform operator
Z	Zero status

CHAPTER 1

1. INTRODUCTION

From the time medium scale integration was developed, it was obvious that larger and more complex circuits could and should be fabricated on a single silicon chip. Although this was technically feasible the semiconductor manufacturers were faced with the problem of economics. As the chip is made more complex, its application becomes more specialized and this reduces the number of potential buyers. Furthermore, the more complex the chip, the lower is the yield of good chips and this leads to higher manufacturing costs. The semiconductor manufacturers continually looked for customers which required identical complex circuits in the order of tens and hundreds of thousands. Only by this means could the unit costs be lowered to justify the economics of the production.

It was the MOS technology which first made the large scale integration (LSI) possible. The MOS transistors are smaller than the bipolar transistors and operate at smaller currents, thus allowing more complex circuits to be fabricated on a given chip size. The first major application of the MOS LSI technology was in the manufacture of calculators. A large number of functions were provided on the calculator chips which created a market large enough to justify the economics of the LSI fabrication.

A microprocessor is a central-processing-unit (CPU)

implemented mostly by the MOS LSI technology. It is capable of performing arithmetic and logical operations and making decisions under program control. The first microprocessor chip (Intel 4004) was manufactured in early 1970s. This was a 4-bit device and implemented with the p-channel MOS technology. Because of its low speed and limited memory and instruction capabilities, this device did not gain any popularity in computer applications and it was mainly used as a calculator chip.

As the advantages of the microprocessors became apparent, the semiconductor manufacturers invested large amounts of money in finding techniques to increase the speed and density of gates on a given chip size. During the 1970s, the early p-channel MOS microprocessors were joined by the faster n-channel MOS types. The change to the n-channel process was the most important of the advances in the MOS LSI technology. Most of the currently available microprocessors use some version of the n-channel MOS technology.

The present single chip microprocessors have fixed word lengths and relatively low speeds (determined by the technology employed) which makes them unsuitable in high-speed real-time applications such as high-speed disk controllers, real-time digital signal processors etc.

An alternative approach adopted by some semiconductor manufacturers was to provide bit-slice processors. Most of

the bit-slice processors are designed with the fast bipolar Schottky TTL technology (some are designed with the even faster ECL technology). The bit-slice processors are mostly provided with 4 bits of data length and these 4-bit slices can be cascaded to any width which is a multiple of four. Thus, a processor can be built that can handle non-conventional word lengths (e.g. 20, 24, 30 etc.). The bit-slice processors are by no means as complex as the conventional fixed instruction MOS microprocessors. In fact these devices are very primitive in their functional capabilities and they can only perform operations in the conventional microprocessor level when supported by a large number of MSI and LSI devices. This approach gives rise to higher cost, larger printed circuit board area and higher power consumption. The major advantages offered by the bit-slice approach are the increase in speed and flexibility of design. Bit-slice processors are microprogrammable and the user can invent his own instruction set for his immediate applications.

In a microprogrammed processor the operations of the system are controlled by an internal program stored in the microprogram memory. The signals output from the microprogram memory are distributed to all parts of the system. By means of microprogramming, the designer can implement any chosen instruction set. If the instruction set of the computer is to be modified then all that is required is to change the data stored in the microprogram memory.

The subject of the research outlined in this thesis is to design a bit-slice microcomputer and to investigate its applications to real-time digital filtering, by inventing the necessary special-purpose instructions. It was aimed to provide as much flexibility as possible in the design of the CPU so that a future user will require the minimum labour in expanding the system. The control section of the CPU is designed such that it offers 'user microprogramming', which enables the users to invent and implement their own instruction sets on the actual system.

The design of the microcomputer has been described in detail in many chapters of the thesis. This is necessary for the following reasons:-

i. Since one of the aims of this research work was to design a bit-slice microcomputer, it is necessary to describe in detail the steps involved in designing such a complex system.

ii. It is necessary to know the architecture of the system in detail in order to be able to follow the chapters describing the microprogramming process.

iii. There is very little published data on the design of such a complex system. The detailed design process described in this thesis can be an invaluable source of reference to those engaged in the design of a bit-slice microcomputer.

iv. A detailed description of the design is essential to those wishing to maintain the microcomputer.

The thesis can be divided into two major parts: Chapters 4 to 16 describe the design of the bit-slice microcomputer . The applications to digital filtering are investigated in Chapters 17 to 20.

Chapter 5 describes the design of the Arithmetic Processing Unit (APU) of the microcomputer. Here, a novel feature is developed such that although the data length of the APU is 16 bits wide, operation with 8 bits is also feasible. During 8 bits of operation only the higher byte of the Data Bus is active. This feature allows the 8 bit fixed instruction MOS microprocessors to be emulated on the bit-slice system.

Chapter 11 describes a novel method for the design of the control unit. The microprogram memory is designed from a combination of PROM and RAM type memories. The RAM part of the microprogram memory can be loaded with the user defined microinstructions. The PROM part is loaded with microinstructions which implement a fixed instruction set with 62 general purpose instructions. Thus, if the user does not wish to generate his own instructions, he can use the microcomputer as a general purpose machine. Also, a future user can develop his own microcode more easily, by analysing the ones already implemented in the PROM part of the microprogram memory.

Chapter 15 describes a novel technique for loading the RAM part of the microprogram memory. Here, a low cost 8-bit microprocessor is used as a Support Processor. The Support Processor accepts the user defined microinstructions from the user terminal (TTY or VDU) and loads the RAM part of the microprogram memory.

Chapter 17 gives a brief introduction to the theory of digital filters. In this Chapter, equations are derived for the design of Butterworth and Chebyshev I type digital filters.

Chapter 18 investigates the design of Low Pass IIR digital filters. A set of special instructions are invented in this Chapter which greatly simplified the design of real-time IIR digital filters, using the bit-slice microcomputer.

In Chapter 19, the design of real-time High Pass IIR digital filters are considered and a set of special instructions are invented for their implementation on the bit-slice microcomputer.

Chapter 20 investigates the design of FIR digital filters. Here, a novel FIR filter algorithm is derived and special instructions are invented for the implementation of this algorithm on the bit-slice microcomputer.

Volume 2 of the thesis contains the computer programs which are reffered to in the main text. It also includes

two manuals and other supplementary information prepared by the author in support of the bit-slice microcomputer. Note that UMM-16 (from:'User Microprogrammable 16 bit Microcomputer) is the name given to the bit-slice microcomputer designed.

CHAPTER 2

2. PRINCIPLES OF MICROPROGRAMMING

2.1 Introduction

Microprogramming was first proposed by M.V. Wilkes (1951) as a systematic technique for designing the control section of a digital computer. In order to understand the principles of microprogramming it is necessary to review the basic functional block diagram of a computer. Any digital computer can be partitioned into four (see Fig.2.1) distinct functional blocks: input - output, arithmetic and logic unit, storage, and control.

The control section controls the operations of the entire computer. It receives instructions from the storage section which tells it the operations to be performed. Upon receiving and decoding an instruction, the control section goes through a sequence of control steps issuing commands to open and close the specific gates for the execution of the required instruction. Once the arithmetic and logic unit has finished its task, the control section may issue commands to permit the result of the operation to be stored in an internal register or to be transferred to an output channel. At the end of every instruction, the control section forces the computer to fetch the next instruction from the storage unit. The basic tasks of the control section are then:-

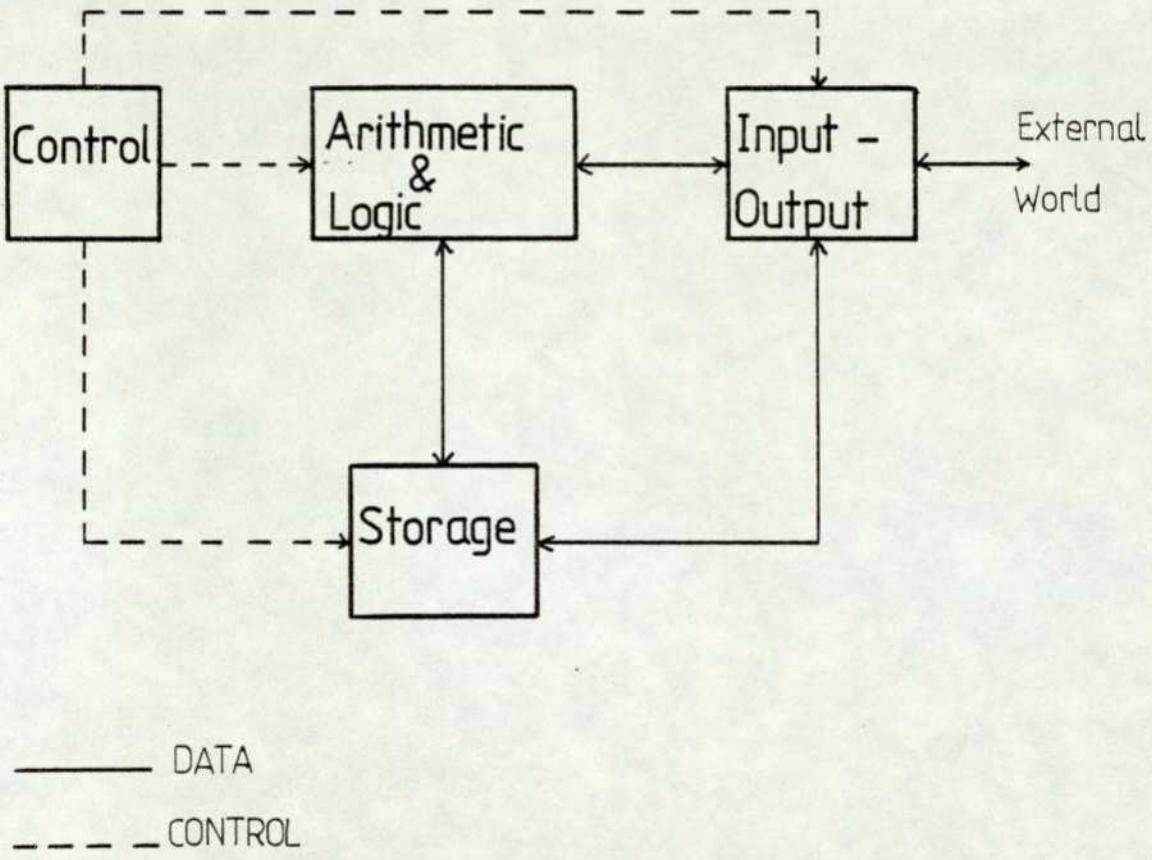


Figure 2.1 The functional blocks of a Digital Computer

i. Fetch an instruction from the storage section

ii. Decode the instruction (determine what sequence of control steps to be taken)

iii. Execute the instruction (send the appropriate commands to activate the data paths to perform the required operation).

iv. Fetch the next instruction from the storage section

Originally, the control section of a digital computer had been designed by connecting a very large array of logic elements (such as flip-flops, decoders, multiplexers etc.) in a somewhat random fashion. There were no systematic methods involved in the design. The control section design engineers had a very difficult task because of the complexity involved and the lack of modularity. If the instruction set of the target machine were to be modified slightly then the control section would have to be changed considerably. This operation was both costly and time consuming.

Microprogramming is a systematic design procedure as an alternative to the conventional hardware control section design. In a microprogrammed computer the control section is implemented by stored logic (microprogram memory) which is usually a fast semiconductor memory. The data output from the microprogram memory controls all parts of the computer. By means of microprogramming, the designer can implement any chosen instruction set. If the instruction set of the computer is to be modified then all that is

required is to change the data stored in the microprogram memory. Thus the architecture of a microprogrammable computer can easily be modified for a new application at a cost much less than that required to implement the same modification on a conventional hardware controlled computer.

Microprogramming is analogous to conventional programming. In conventional programming the user tells the computer what to do by placing a sequence of instructions (macroinstructions) in the main memory of the computer. In microprogramming, the user tells the system how to execute required macroinstructions by loading the appropriate data (microinstructions) in the microprogram memory. Although the two fields are analogous it is usually much more difficult to write microinstructions than writing the macroinstructions. A microprogrammer must have an understanding of every logic element in the computer and the way they are connected and operated. He must have knowledge about the system timing and the way each sequential piece of hardware operates. A conventional programmer on the other hand is only interested in the instruction set and the basic functional architecture of his computer.

2.2. Wilkes' Model of a Microprogrammed Control Unit

Wilkes originally proposed the microprogrammed

control unit by using diode matrices to implement the control functions as shown in Fig. 2.2. The basic system consists of the control matrix, a binary decoding tree and the sequencing matrix with the decision making logic.

The binary decoding tree has the function of decoding a pattern of n bits (from the address register R) and selecting 1 out of the 2^n possible outputs.

The control matrix consists of horizontal and vertical wires interconnected via diodes at predetermined points. The vertical wires constitute the control signals and are distributed to the rest of the computer to control the various gates and registers. The horizontal wires activate a particular set of vertical wires with diode connections to it. Thus the horizontal wires can be thought of as the addresses to the control matrix and the vertical wires as the contents of these addresses (microinstructions).

The signals from the binary decoding tree also activate the sequencing matrix. The data output from the sequencing matrix (vertical wires) load the register R with the addresses of the next microinstructions to be executed.

Thus a microinstruction is selected according to

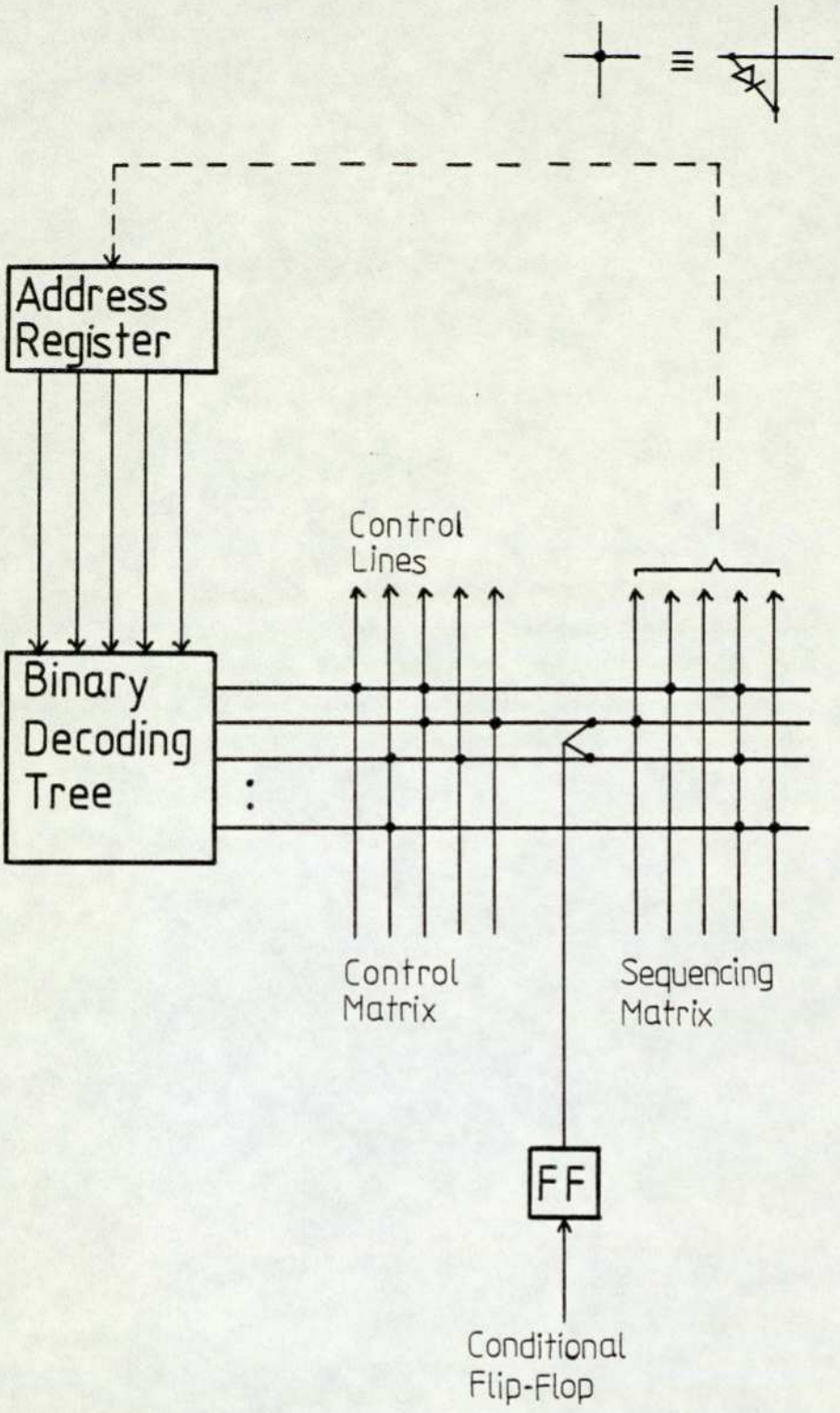


Figure 2.2 Wilkes' model of a microprogrammed Control Unit

the contents of address register R. The selected micro-instruction sends the appropriate control signals to operate the gates and registers in the system. It also selects the addresses of the next microinstructions to be executed.

The decision making logic is provided by a conditional flip-flop where depending upon the state of some external signals, it is capable of selecting one of two possible addresses in the sequencing matrix.

2.3 Evolution of Microprogramming

Microprogramming has evolved through three distinct phases according to the technology used to implement the microprogram memory. During the early 1950s diode matrices were dominantly used which gave rise to the implementation of simple control functions. Although the speed of the diode matrices were low (0.5 μs access time) this did not slow down the computer since the main memory was about 20 times slower than the diode matrices. During this phase, the microprogramming concept was not widely accepted by the computer design engineers and consequently there were no computer manufacturers offering a microprogrammed computer. Most of the work in this field was in the research phase based on the original concept of Wilkes'.

The second generation of microprogramming started with the introduction of magnetic core memories during the late 1950s. The speed of the main memory dropped to under 1 μs and the access time of a read-only microprogram memory ranged between 200 and 500 ns. By this time it was possible to execute several microinstructions per main memory cycle and yet maintain a high spped of operation. An important application of microprogramming during this phase was the emulation of several different machines on a single microprogrammed host computer (partly, the reason was the high investment in software, which users wanted to run on different machines). The major computer manufacturer IBM introduced their microprogrammed computer 'System/360' [3,4] during this phase.

The third generation of microprogramming started around the 1970s with the advent of fast bipolar memories and the bit-slice microprogrammable devices. With the advent of fast read-write memories it was possible to design writeable control stores. The development of the bipolar bit-slice devices made it possible to design user-microprogrammable processors (the subject of this thesis). The availability of bipolar memories and the bit-slice devices enhanced the importance of microprogramming in the design of problem oriented processors. The digital system designer can now build a special purpose micro-programmable system with considerably less effort and lower cost.

2.4 Example on Microprogramming

It is convenient at this point to consider an example to clarify the concept of microprogramming. Fig. 2.3 shows an example of a simple microprogrammable sequence controller, consisting of three logic elements: a multiplexer, a binary counter and a memory. Assume that the truth tables of the multiplexer and the counter are as shown below ('0' represent logic level LOW, ↑ represents LOW to HIGH transition of the clock input and 'X' is a don't care entry).

Mux. inputs		Mux. output
S_1	S_0	Y
0	0	B_0
0	1	B_1
1	0	B_2
1	1	B_3

Counter inputs			Counter outputs	
D	LOAD	CLOCK	Y	
0	1	↑	0	load
1	1	↑	1	load
X	0	↑	Y+1	count

Some of the data output from the microprogram memory controls the internal operations of the controller where others constitute the external control signals. The controller shown in this example is capable of performing 3 different instructions: CONTINUE, conditional JUMP and unconditional JUMP.

When the multiplexer control field of the microprogram

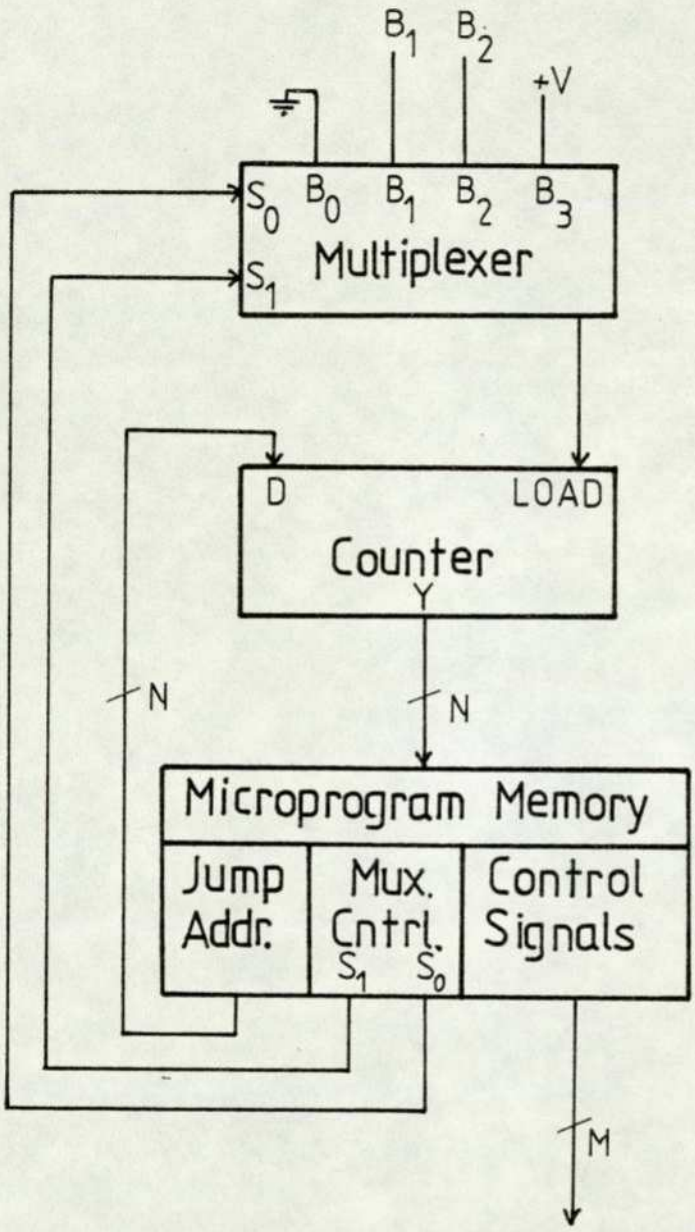


Figure 2.3 A simple microprogrammable sequence controller

memory contains the binary data 00 then B_0 will be selected at the output of the multiplexer which will force the LOAD imput of the counter LOW. Thus, the counter will be incremented by one (CONTINUE) to select the next sequential microinstruction as the next control data.

When the multiplexer control field of the microprogram memory contains the binary data 11 then B_3 will be selected at the output of the multiplexer which will force the LOAD input of the counter HIGH. Thus, on the next clock pulse the counter will be loaded with the data contained in the jump address field of the microprogram memory. Hence, an unconditional JUMP instruction will be performed.

When the multiplexer control field of the microprogram memory contains the binary data 01 then B_1 will be selected at the output of the multiplexer. If B_1 is true (logic HIGH), a JUMP operation will be performed on the next clock pulse. If on the other hand B_1 is false (logic LOW) then a CONTINUE operation will be performed on the next clock pulse. Thus, the controller will execute a conditional JUMP instruction depending upon the state of B_1.

Similarly when the multiplexer control field of the microprogram memory contains the binary data 10 then a conditional JUMP instruction will be executed depending upon the state of B_2.

Although the above mentioned controller is simple, it is powerful and represents the flexibility offered by microprogramming (the controller can be modified to suit many applications by simply re-programming the micro-program memory. A new elaborate hardware design is not required).

CHAPTER 3

3. THE BIT-SLICE DEVICES

3.1 Introduction

The bit-slice devices are bipolar microprogrammable LSI logic elements manufactured to ease the design of microprogrammable processors. They are bipolar because intended for high-speed applications (i.e. beyond the present capabilities of MOS technology). Because of the high power consumption (bipolar technology) and the functional complexity, the bit-slice devices are manufactured in 2 or 4 bit wide slices. Any number of these devices can be cascaded to increase the data handling capacity of the overall system.

It should be emphasized at this point that the bit-slice devices are by no means as complex as the conventional MOS microprocessors. In fact these devices are very primitive in their functional capabilities and they can only perform operations in the conventional microprocessor level when supported by a very large number of MSI and LSI devices.

3.2 The Basic Elements of a Bit-Slice Processor

A simplified block diagram of a bit-slice processor

is illustrated in Fig. 3.1, which shows a system divided into a sequencing (or control) section and a data processing (register arithmetic logic unit RALU) section. The two basic parts in any bit-slice processor are then the microprogram sequencer, which addresses the microinstructions and the RALU which performs the arithmetic and logic operations. Both of these two basic elements are commercially available and manufactured by several companies in different functional capabilities.

A bit-slice processor may also contain several additional sections that perform special functions, such as the interrupt controllers, memory, input and output peripheral devices, direct memory access channels etc.

Since one of the aims of this thesis is to describe the design of a bit-slice computer it is necessary to compare the merits of the various commercially available bit-slice RALU elements and the microprogram sequencers before a particular one is chosen.

3.3 Commercially Available RALU Elements [5]

This is the place where all of the arithmetic and logic operations are carried out. It consists basically of an arithmetic and logic unit and associated registers.

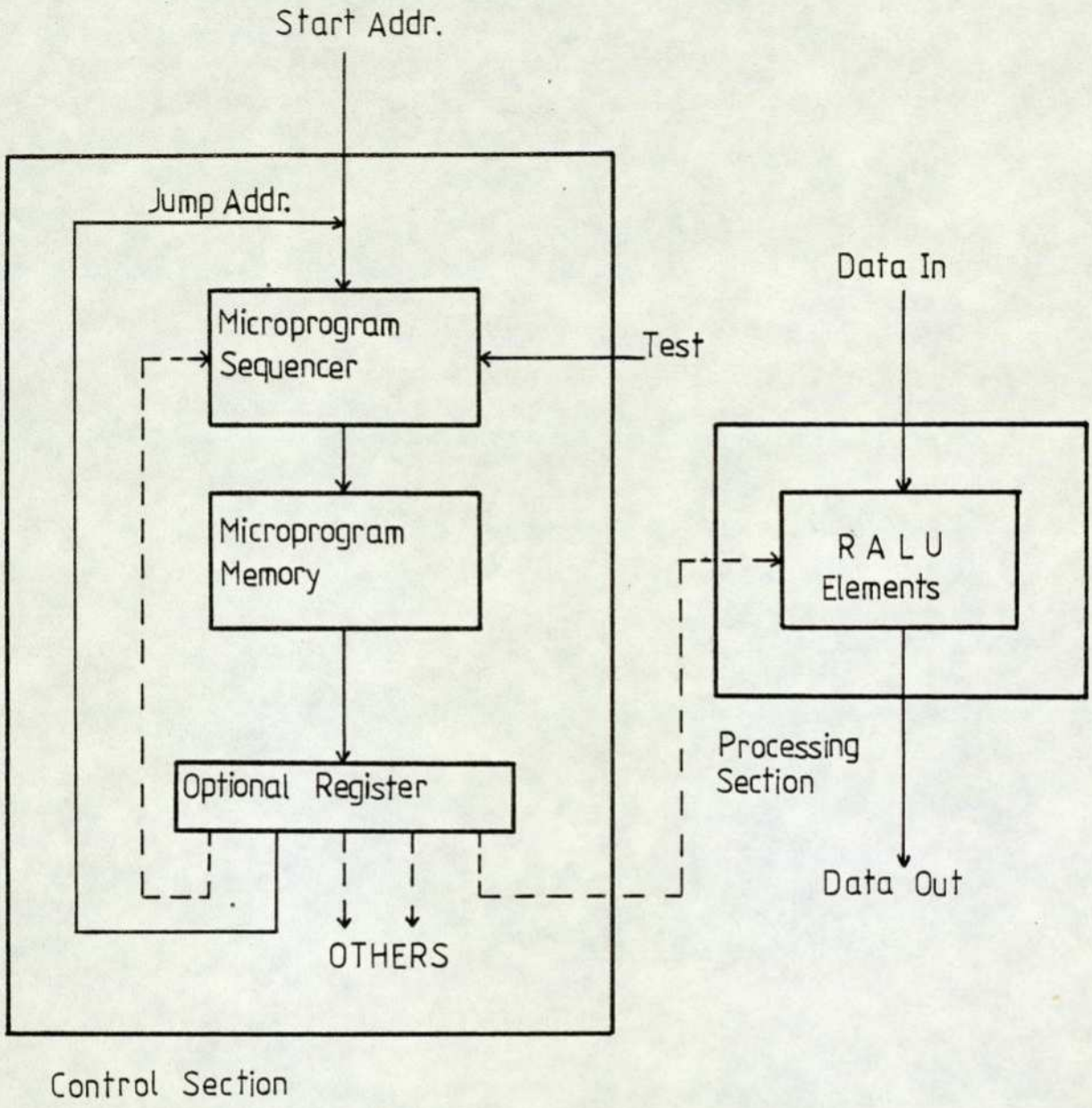


Figure 3.1 Basic elements of a Bit-Slice processor

It gives the bit-slice processor its name, because it generally handles data in 2 or 4 bits wide at a time. The 2 or 4 bit-slices operate in parallel and can be cascaded to any width that is a multiple of the basic slice. Thus a processor can be built that can handle non-conventional word lengths (e.g. 18,20,22 etc.) The arithmetic and logic functions performed by the bit-slice RALU elements are dictated by the microprogram memory and the microprogram sequencer. The sequencer provides an address to the microprogram memory corresponding to the macroinstruction to be executed. The microinstructions coming out of the microprogram memory send the appropriate control signals to the bit-slice RALU elements and to various other parts of the processor.

Table 3.1 lists the architectural characterstics of the commercially available RALU elements. The arithmetic and logic capabilities of the RALU elements are shown in Table 3.2.

The following sections will summarize the general characteristic of the commercially available RALU elements.

3.3.1 Intel 3002

This is a 2 bit wide slice. For a 16 bit processor,

Bit-Slice Device	Manufacturer	Technology	Slice Width (Bits)	No. of Pins	No. of Registers
3002	INTEL	Schottky TTL	2	28	13
9405	FAIRCHILD	Schottky TTL	4	24	9
Am2901	AMD	Schottky TTL	4	40	17
6701	MMI	Schottky TTL	4	40	17
SBP0400	TI	I^2L	4	40	10
SN74S481	TI	Schottky TTL	4	48	4
MC10800	MOTOROLA	ECL	4	48	2
Am2903	AMD	Schottky TTL	4	48	17
F100220	FAIRCHILD	ECL	8	68	1

Table 3.1 Commercially available Bit-Slice RALU elements

Bit-Slice Device	Arithmetic[1] Operations	Logical[2] Operations	Multiply	Divide	Normalize	BCD Add	Parity	2's comp/sign magnitude conv.
3002	Add,D1,2C	C,A,O,EN	-	-	-	-	-	-
9405	Add	C,A,O,EO	-	-	-	-	-	-
Am2901	Add,S,D1,2C	C,A,O,EO,EN	-	-	-	-	-	-
6701	Add,S,D1,2C	C,A,O,EO	-	-	-	-	-	-
SBP0400	Add,S,I1,2C	C,A,O,EO,EN	-	-	-	-	-	-
SN74S481	Add,S,I1,D1, 2C	C,A,O,ND,NR	YES	YES	-	-	-	-
MC10800	Add,S,D1,D2, I1,I2,2C	C,A,O,ND,NR, EO,EN	-	-	-	YES	YES	-
Am2903	Add,S,I1,I2, 2C	C,A,O,ND,NR EO,EN	YES	YES	YES	-	YES	YES
F100220	Add,S	C,A,O,EO	-	-	-	YES	YES	-

1.
Di:Decrement by i
2C:2's complement
S :Subtract
Ii:Increment by i

2.
C :Complement
A :AND
O :OR
EO:exclusive-OR
EN:exclusive-NOR
NR:NOR
ND:NAND

Table 3.2 Arithmetic and logical capabilities of the Bit-Slice RALU elements

eight 3002 devices must be connected in parallel. Intel 3002 has the advantage over the other RALU elements in that it offers a larger number of input/output ports and this makes it powerful in data manipulation applications than in arithmetic and logic operations. The device contains 13 internal working registers available to the user. Intel 3002 is manufactured in a 28 pin DIL package with the Schottky TTL technology.

3.3.2 Fairchild 9405

This is a very simple RALU element although it handles a 4 bit wide slice. 9405 can only perform one arithmetic operation ('Add') and is best suited for applications that are just beyond the capabilities of the MOS processors. The device contains only 9 internal working registers available to the user. Fairchild 9405 is manufactured in a 24 pin DIL package with the Schottky technology.

3.3.3 Advanced Micro Devices Am2901

This is a 4 bit wide RALU element containing 17 internal working registers available to the user. This device offers a wide range of arithmetic and logic capabilities. The device is controlled via 18 control

lines. The microinstruction flexibility of the Am2901 allows it to be used in high speed peripheral controllers and user microprogrammable microprocessors. This RALU element is manufactured in a 40 pin DIL package with the Schottky TTL technology.

3.3.4 Monolithic Memories 6701

This device is identical in many respects to the Am2901, the major difference being that the latter has more logical capabilities. 6701 is controlled via 17 control lines. This device is also manufactured in a 40 pin DIL package.

3.3.5 Texas Instruments SBP0400

This is a 4 bit wide RALU element containing 10 internal working registers available to the user. The device is controlled by 11 control lines. The logical capabilities of SBP0400 are equivalent to those of Am2901 but it has slightly different arithmetic capabilities. This RALU element has the disadvantage that it is the slowest RALU element of all. The device is manufactured in a 40 pin DIL package with the I^2L technology.

3.3.6 Texas Instruments 74S481

This is a 4 bit wide slice. It is the most complex RALU element available. The device contains built in multiply and divide instructions and is well suited to signal processing applications. It is controlled by 17 control lines. One of the disadvantages of 74S481 is that it does not contain any register file. Instead, 2 accumulator registers and 2 other registers (program counter register and memory counter register) are provided. This RALU element is manufactured in a 48 pin DIL package with the Schottky TTL technology.

3.3.7 Motorola MC10800

This is a 4 bit wide device. It is the fastest bit-slice RALU element currently available (built with the ECL technology). It is one of the RALU elements that provides binary-coded-decimal addition. Although the MC10800 contains only 2 internal registers, it can easily support an external register file. The major disadvantage of this device is that it is not TTL compatible and special hardware is required to interface with the standard TTL logic family.

3.3.8 Advanced Micro Devices Am2903

This RALU element is basically a modified version of the Am2901. It provides built in multiplication and division (both signed or non-signed). Additionally the Am2903 can perform special operations such as normalization and 2's complement/sign magnitude conversion. The device is controlled by 21 control lines and includes 17 internal working registers available to the user. The Am2901 is well suited to high speed applications such as the real-time signal processing. The device is manufactured with the Schottky TTL technology and housed in a 48 pin DIL package.

3.3.9 Fairchild F100220

This is the latest and largest bit-slice RALU element available. It is an 8 bit device which can be cascaded to any length. The device is controlled by four instruction bits and a multiplexed control line. A total of 27 instructions can be executed, including binary arithmetic, decimal arithmetic, single-bit shifts, logical data transfers etc. One major disadvantage of the F100220 is that it provides only one internal register (accumulator) available to the user (external register file can be connected to the device). The device is manufactured with the ECL technology and housed in a 68 pin package.

3.4 Commercially Available Microprogram Sequencers [5,6]

The main purpose of a microprogram sequencer is to present an address to the microprogram memory so that a macroinstruction can be fetched and executed. A microprogram sequencer basically contains a microprogram address counter /register and next address generation logic. The next address generation logic determines the source of the address to be loaded into the microprogram address counter/ register. All the operations of a microprogram sequencer are dictated by the microprogram memory. The typical operations performed by a microprogram sequencer are: increment (continue), conditional/unconditional jump, conditional/unconditional jump to a micro-subtoutine, perform microprogram loops and so on.

Table 3.3. gives a list of the commercially available LSI microprogram sequencers. The available sequencers may be divided into two categories: The first one is in which several such bit-slice devices are cascaded to address the microprogram memory. The second is not a bit-slice approach, but one in which a single package of sequencer is used to address the microprogram memory. While the bit-slice sequencers can be cascaded to any addressing range, the fixed address sequencers can not be cascaded, but instead additional cicuits and special paging techniques can be employed to extend their addressing capabilities.

Sequencer	Organisation	Addressing Range	Addressing Mode	Technology	Internal Looping Capability	Internal Subroutine Capability	Status Information	Manufacturer
SN74S482	Bit-Slice	See note	Relative	Schottky TTL	No	Yes	Not stored	Texas Inst.
Am2909	Bit-Slice	See note	Absolute	Schottky TTL	No	Yes	Not stored	AMD
Am 2911	Bit-Slice	See note	Absolute	Schottky TTL	No	Yes	Not stored	AMD
MC10801	Bit-Slice	See note	Absolute	ECL	Yes	Yes	stored	Motorola
8X02	Fixed-Address	1024	Absolute	Schottky TTL	No	Yes	Not stored	Signetics
67110	Fixed-Address	512	Absolute	Schottky TTL	Yes	Yes	Stored	Monolithic Mem.
3001	Fixed-Address	512	Absolute	Schottky TTL	No	No	Not stored	Intel
9408	Fixed-Address	1024	Absolute	I^2L	No	Yes	Stored	Fairchild
Am2910	Fixed-Address	4096	Absolute	Schottky TTL	Yes	Yes	Not stored	AMD

Note: Addressing range depends upon the number of slices cascaded (Basic slice is 4 bits wide)

Table 3.3 Commercially available microprogram sequencers.

3.4.1 Texas Instruments SN74S482

This is a 4 bit wide bit-slice microprogram sequencer. Unlike the other sequencers, the SN74S482 contains a full adder which permits relative addressing in the microprogram memory. A major disadvantage of this sequencer is that it does not have encoded inputs for instructions and this facility must be provided by the user. The device is housed in a 20 pin DIL package and designed with the Schottky TTL technology.

3.4.2 Advanced Micro Devices Am2909/11

Both of these devices are identical except the Am2909 is slightly more complex. Both devices are 4 bit wide, bit-slice microprogram sequencers. Like the SN74S482, they lack the encoded inputs for instructions. These sequencers contain a 4 word deep Last in First Out (LIFO) stack to save the return addresses during nested micro-subroutines. The Am2909 is housed in a 28 pin DIL package while the Am2911 is manufactured in a 20 pin package.

3.4.3 Motorola MC10801

Motorola's MC10801 emitter-coupled-logic sequencer

is the fastest of all. It is a 4 bit wide, bit-slice microprogram sequencer. The device provides repeated execution capability of a single microinstruction. Additionally, it contains a LIFO stack so that the micro-subroutine return addresses can be saved. The status information of the RALU elements can be stored internally by the MC10801, which may be used during the execution of conditional instructions. One disadvantage of this sequencer is that it is not TTL compatible.

3.4.4 Signetics 8X02

This is a fixed-address microprogram sequencer capable of addressing 1024 words of the microprogram memory. The device offers repeated looping capability in the micro-program memory. Housed in a 28 pin DIL package, the 8X02 is an excellent choice for medium performance applications.

3.4.5 Monolithic Memories 67110

This microprogram sequencer is capable of addressing up to 512 words of the microprogram memory. Like the MC10801, this device provides internal storage of the RALU array flags. The addressing scheme provided by the 67110 is rather awkward; the least significant bit of the address output is designed such that a conditional branch to an even

or odd address pair may be executed without the use of the next-address field. Thus care should be taken when writing and modifying the microinstructions to ensure that the even-odd branch addresses are stored in the even-odd addresses of the microprogram memory. The choice of the 67110 depends upon whether or not it is convenient to implement a pairwise branching scheme.

3.4.6 Intel 3001

This is another microprogram sequencer with 512 words of addressing capability. The device is controlled with 7 control lines. The addressing scheme of the 3001 is unique. A set of branch instructions allow jumps from a given address to a fixed subset of the address space that is a function of that address. Like the 67110, the 3001 provides storage of the flags of RALU elements. One disadvantage of this device is that it does not have any stacks to implement micro-subroutines. Also, no microprogram looping facilities are provided.

3.4.7 Fairchild 9408

This device is a fixed address microprogram sequencer capable of addressing 1024 words (1K) of the microprogram memory. Like the MC10801, the 9408 can save the status

information of the RALU elements. An unusual feature of this device is that it includes an 8-way branch instruction. Micro-subroutine return addresses can be saved in the internal 4 word by 10 bit stack file. Although slower than other microprogram sequencers, this device dissipates less power (I^2L technology).

3.4.8 Advanced Micro Devices Am2910

The Am2910 basically consists of three cascaded Am2911s and a programmed logic array for instruction decoding. It can address up to 4096 words of the micro-program memory. The device contains a 12 bit loop counter and a 5 word deep stack for micro-subroutines. The Am2910, being a high performance sequencer is a good choice for use in high-speed processor applications.

Before going into the details of the bit-slice microcomputer designed, it is worthwhile to look at the advantages and disadvantages of the bit-slice processors, compared to the conventional fixed instruction MOS processors.

3.5 Advantages of the Bit-Slice Processors

i. Bit-slice processors are mostly manufactured

using the bipolar Schottky technology that can operate at much higher speeds than the MOS processors. Some use the even faster ECL bipolar technology.

ii. Special macroinstructions can be implemented for specific applications. This not only increases the rate of operation but it also simplifies the assembly level of programming. By inventing special instructions, the programmer can select the optimum set of machine instructions for his immediate needs.

iii. The instruction sets of other processors can be emulated. This can be achieved by re-programming the microprogram memory. Thus the processor can be made to switch between a number of different system architectural characteristics.

iv. The designer of a bit-slice processor can choose and modify the word length according to the specific application by cascading several bit-slice devices (e.g. 4, 8, 12, 16 ... word lengths are all possible).

v. The hardware of the Central Processing Unit (CPU) can easily be modified and this makes the bit-slice processors very flexible and adaptable.

Since the designer of a bit-slice processor has to make all the architectural decisions himself, it follows that greater control is available over the instruction set of the target machine.

vi. The user of a bit-slice processor gains a better understanding towards the internal architecture of computers.

vii. Bit-slice design extends the useful life of the system. It has the ability to meet new demands by re-programming the microprogram memory.

3.6 Disadvantages of the Bit-Slice Processors

i. Microprogramming a bit-slice processor is not an easy task. The microprogrammer has to know the exact data flow in and out of every control gate and the timing points that control the transmission of data between the static units of the system. Although there are some commercially available micro-assemblers [6,7] to ease a microprogrammers job, these are usually designed for specific processors and are very primitive.

ii. Microprogram memory is usually read-only storage

which is difficult and expensive to modify. Where the microprogram memory is a read-write memory, then another processor may be required to transfer data easily to the microprogram memory.

iii. For systems where the architecture is simple and fixed and where the function of the system is relatively simple, use of bit-slice devices is too expensive.

iv. There is a lack of sufficient practical data published on the design of bit-slice processors. This discourages the designers from attempting to use the bit-slice devices.

3.7 Conclusions

This chapter has described the commercially available bit-slice devices. It was shown that there is a large variety of devices to choose from. The final choice of a particular device is greatly determined by the cost and the functional ability of that device.

The advantages and disadvantages of the bit-slice design were discussed. It is clear that for the high speed special purpose complex processor designs, the bit-slice approach is the only available tool for the designers (except that a large number of MSI chips could also be used).

CHAPTER 4

4. DESIGN OF THE BIT-SLICE MICROCOMPUTER

4.1 Introduction

The aims of the research outlined in this thesis include the design of a bit-slice microprogrammable microcomputer. Although during the initial design phase it was intended to design a general purpose bit-slice microcomputer, this basic requirement was expanded later with the thought that the microcomputer was to be used primarily in the digital signal processing applications. Thus the basic requirement of the design was that the microcomputer should be a general purpose bit-slice machine with the ability to process digital signals in real-time.

This and the following several chapters of this thesis will describe the design of the microcomputer.

4.2 The Block Diagram of the Microcomputer

The functional block diagram of the microcomputer is shown in Fig. 4.1. The functions of the various subsystems of the microcomputer are summarized below.

i. The Arithmetic Processing Unit (APU)

This is the section where all of the basic

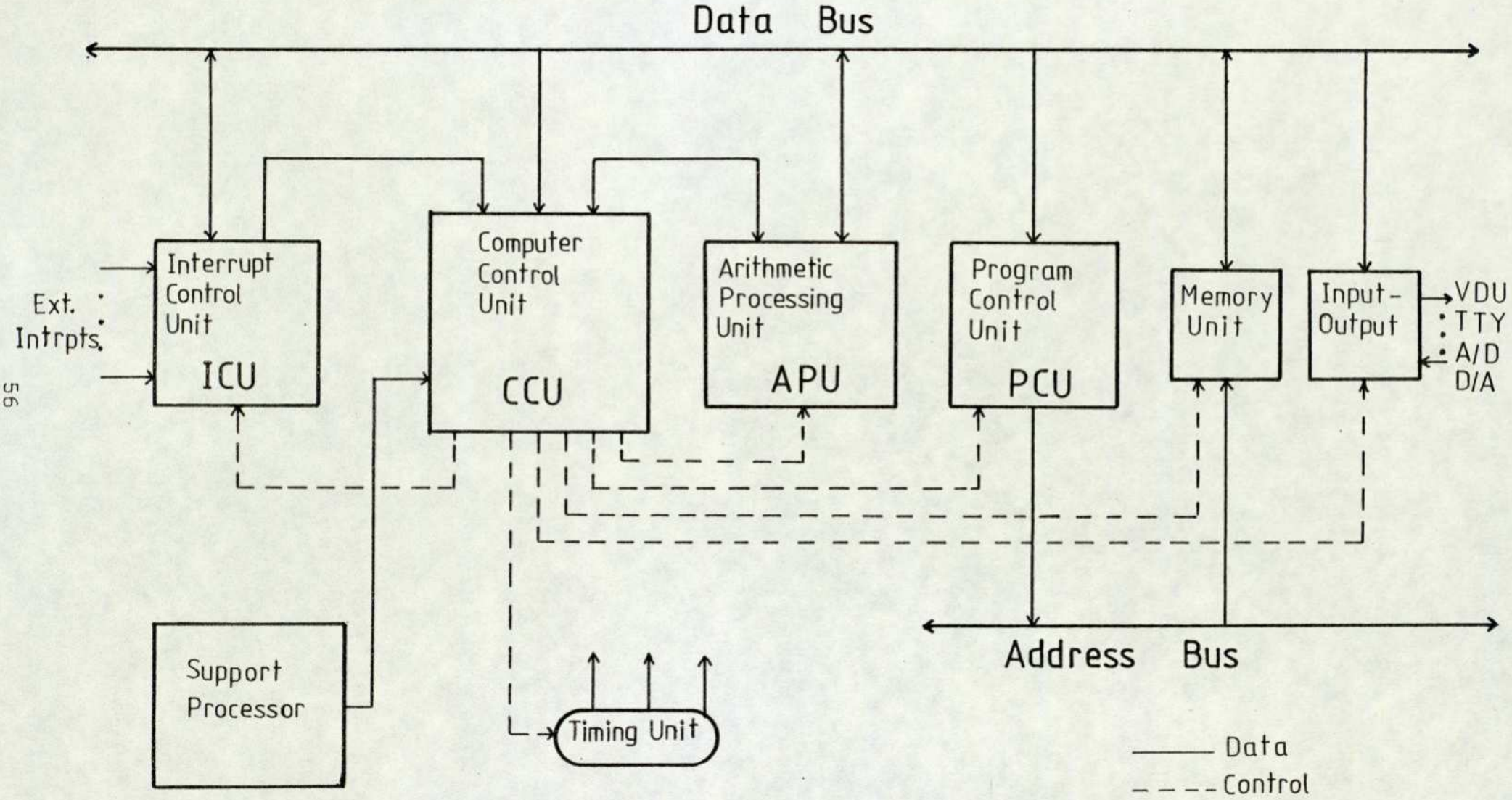


Figure 4.1 The functional block diagram of the microcomputer

arithmetic and logical operations are performed. Additionally, data shifting and rotating functions, generation and decoding of the status flags are all functions of the APU. The operations of the APU are controlled by the Computer Control Unit (CCU). The data width of the APU was chosen to be 16 bits, but facilities were provided to change it to 8 bits so that the architecture of the 8 bit MOS microprocessors could be emulated.

ii. The Computer Control Unit (CCU)

The Computer Control Unit is the most complicated subsystem of the microcomputer. It is this subsystem that controls the other subsystems, synchronizes the internal and external events, fetches and decodes the instructions and sends the appropriate commands to other subsystems of the microcomputer to execute the instructions. At the heart of the CCU is the microprogram memory which was designed from a combination of PROM and RAM type of memories.

iii. The Program Control Unit (PCU)

The function of the Program Control Unit is to address the next instruction to be fetched from the main memory. The PCU consists of basically

a program counter register, a stack, and the associated logic. The program counter register stores the current memory address and it is incremented by one after the execution of every macroinstruction.

iv. The Interrupt Control Unit (ICU)

Interrupts are external asynchronous events that change the normal program flow in a computer. By interrupting the processor, the external logic asks the processor to stop whatever it is currently doing in order to service the needs of the external logic. Upon servicing the interrupt, the processor carries on with its suspended operations. The microcomputer designed here accepts seven external and one internal (timer) interrupts. The ICU is controlled from the CCU.

v. Memory Unit

Two types of memories were used in the main memory section of the microcomputer: the PROM type of memories were used to store the system monitor (Operating System) and the RAM type of memories were used to store the user programs or data. The memory unit is controlled from the CCU.

vi. Input-Output Unit

The Input-Output subsystem provides the communication between the microcomputer and outside world. The microcomputer accepts data from the input devices (TTY, VDU, A/D converter etc.), processes them and the result is sent to the external devices (TTY, VDU, D/A converter etc.) via the output ports. The input-output unit is controlled from the CCU.

vii. The Timing Unit

The Timing Unit generates the system clocks, initializes the microcomputer and responds to external supervisory signals. The supervisory signals control the timing unit via external switches such as 'Run','Halt' and 'Single Clock'.

viii. The Support Processor

The Support Processor is an F8 microcomputer. Its purpose is to load the RAM part of the microprogram memory and the Mapping memory with the user-defined microinstructions.

CHAPTER 5

5. DESIGN OF THE ARITHMETIC PROCESSING UNIT (APU)

The functional block diagram of the APU is shown in Fig. 5.1. The functions of the various parts of the APU are as follows:-

i. The Arithmetic and Logic Unit (ALU) was designed from the bit-slice RALU elements and this unit performs the basic arithmetic (add, subtract, multiply, divide etc.) and logical (AND, OR, NAND etc.) operations either on 16 bits of data or 8 bits of data.

ii. In some real-time applications it may be required to perform high-speed bit operations (e.g. in Direct Digital Control). The Bit Manipulation Logic (BML)was designed to manipulate the individual bits of the internal APU working registers (e.g. bit-set, bit-reset, bit-test etc.).

iii. In high-speed signal processing applications, the number of registers provided by the RALU elements may not be adequate (e.g. for high order digital filtering). It was therefore decided to add external high-speed registers to the microcomputer. The Register Extension Logic (REL) was designed to provide 16 additional

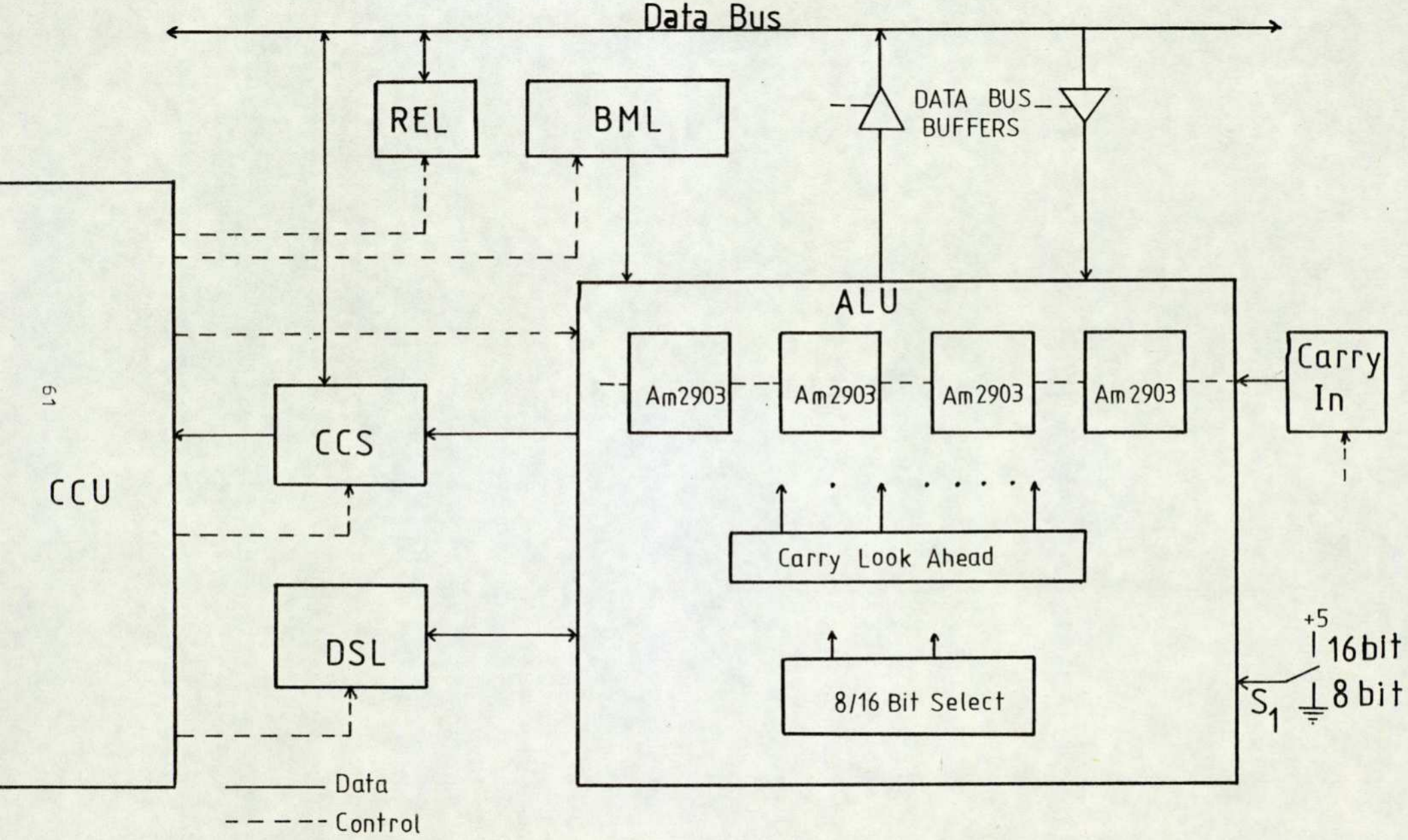


Figure 5.1 Block diagram of the Arithmetic Processing Unit

registers to the ALU.

iv. The Data Shifting Logic (DSL) was designed to shift (or rotate) data to either ends of the APU registers under the control of the CCU.

v. The Condition Code Select logic (CCS) manipulates the status bits of the ALU and sends the appropriate one to the computer control unit so that conditional operations can be executed (e.g. branch if zero, branch if positive etc.). Facilities were provided such that the source of data for the CCS can either be the ALU status bits or it can be derived from other sub-systems of the microcomputer (e.g. from the main memory).

vi. The buffers at the output of the ALU are used to send or receive data to the ALU and these lines constitute the system Data Bus.

vii. The Carry-in control logic controls the 'carry' input of the ALU.

5.1 Design of the Arithmetic and Logic Unit (ALU)

At the heart of the ALU are the bit-slice RALU elements. The RALU elements chosen here were the Am2903s.

The choice of these devices was mainly influenced by the fact that the Am2903s contain special instructions for the implementation of multiplication and division. Additionally, these devices are manufactured by Advanced Micro Devices (a leading company in the field of bit-slice processors) which provides probably the most comprehensive application data on bit-slice processors.

As the Am2903s are one of the basic building blocks of the microcomputer, it is essential to understand their operation in detail.

5.1.1 The Am2903 [8]

The Am2903 is a four bit cascadable bipolar RALU element that can perform seven arithmetic and nine logic operations on two four-bit operands. Additionally it provides nine special instructions for the implementation of multiplication, division,normalization etc.

The block diagram of the Am2903 is shown in Fig.5.2. The device contains a 16 word by 4 bit, two port RAM which serves as the internal working registers. The RAM words are addressed by the 'A' and 'B' address ports (4 bits each). Data is written to any RAM location selected by the 'B' address port, when the write enable input, $\overline{WE}$, the write output, $\overline{WRITE}$, and the clock CP are

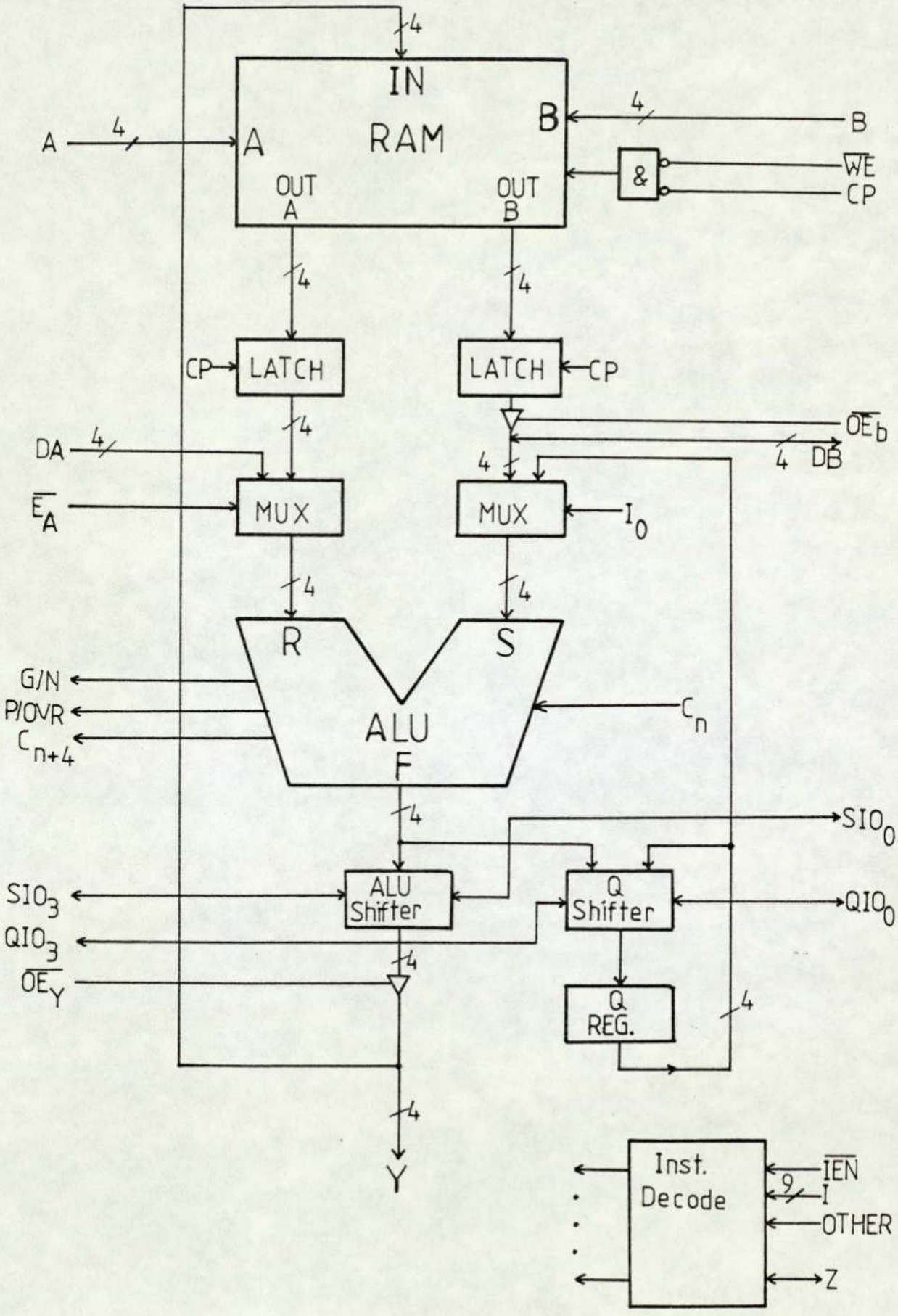


Figure 5.2 Block diagram of the Am2903

all LOW. In addition to the 16 words of RAM, the Am2903 also contains a 'Q' register which is 4 bits wide and it can be used as a general purpose register.

Multiplexers at the internal ALU inputs of the Am2903 provide the capability to select various pairs of ALU source operands. The $\overline{E}_A$ input selects either the 'DA' external data input or RAM output port A for use as one ALU operand. The $\overline{OE}_B$ and I_o inputs select RAM output port B, external data input 'DB' or the 'Q' register content for use as the other ALU operand. Table 5.1 shows the possible pairs of ALU source operands as a function of the $\overline{E}_A$, $\overline{OE}_B$ and I_o inputs.

When the instruction bits I_4, I_3, I_2, I_1 and I_o are all LOW then the Am2903 executes special functions. These special functions are defined in Table 5.4. When the Am2903 executes instructions other than the nine special instructions then the ALU operation is determined by the instruction bits I_4, I_3, I_2 and I_1 (see Table 5.2).

The instruction bits I_8, I_7, I_6 and I_5 define the destination of data after an ALU operation (see Table 5.3). The ALU output (F) can be routed to a destination as non-shifted or it can be shifted left one position (2F), or shifted right one position (F/2). Both arithmetic and logical shift operations are possible. In an arithmetic shift operation the most significant bit of the most significant slice is not shifted while in a logical



Table 5.1 Am2903 Source Operand

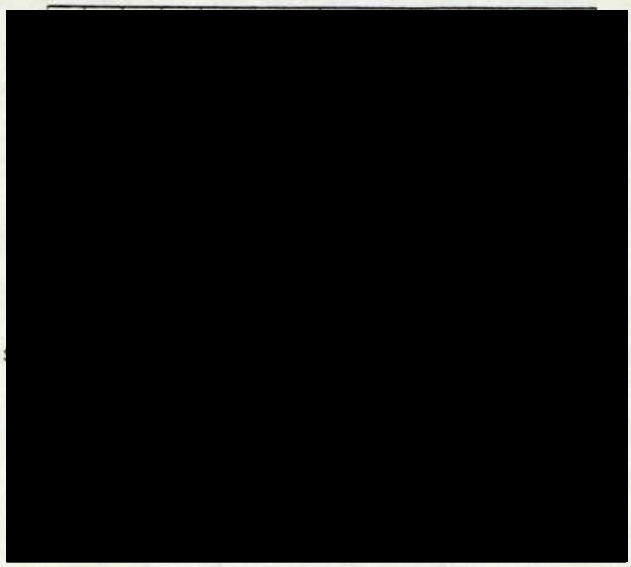


Table 5.2 Am2903 Function Control

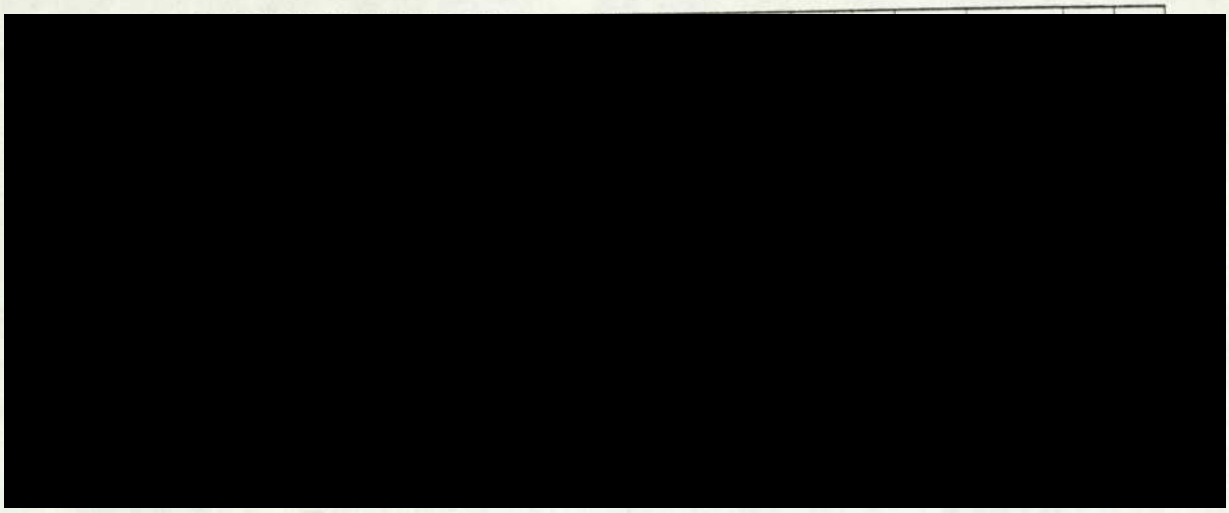


Table 5.3 Am2903 Destination Control



Table 5.4 Am2903 Special Function Control

shift operation all bit positions are shifted. The 'Q' register is always shifted in logical mode. SIO_0 and SIO_3 are bidirectional serial shift inputs/outputs for the two port RAM. Similarly, QIO_0 and QIO_3 are bidirectional shift inputs/outputs for the 'Q' register. During a shift-left operation, the SIO_0 (or QIO_0) is generally a serial shift input and SIO_3 (or QIO_3) is a serial shift output. During a shift-right operation, SIO_3 (or QIO_3) is generally a serial shift input and SIO_0 (or QIO_0) a serial shift output.

The 'DB' and 'Y' ports are four bit bidirectional ports and are driven by three-state buffers. When $\overline{OE}_y$ is LOW, the 'Y' outputs are enabled and are in a high impedence state when $\overline{OE}_y$ is HIGH. When $\overline{OE}_B$ is low, the 'DB' outputs are enabled and are in a high impedence state when $\overline{OE}_B$ is HIGH.

The control bit $\overline{IEN}$ is used to enable the $\overline{WRITE}$ ouput of the Am2903. When $\overline{IEN}$ is HIGH, the contents of the RAM and the 'Q' register are preserved. When $\overline{IEN}$ is LOW, new data can be written to the RAM or the 'Q' register dictated by the destination control bits I_5 through I_8.

The Am2903 generates four status signals after every arithmetic or logical operation. C_{n+4} is the

carry-out pin and in the active state it indicates that the operation exceeded the 4 bit number range. The OVR and N outputs indicate the overflow and the sign of the operation. The Z output is open collector and it can be wire-or'd between cascaded devices to indicate the zero status. To some extent, the meaning of the C_{n+4}, OVR and N signals vary with the ALU function being performed (see Table 5.5).

The Am2903 provides two special pins ($\overline{G}$ and $\overline{P}$) for commecting to a carry look ahead generator for high speed cascaded operations.

The slice position of an Am2903 is programmed via the $\overline{LSS}$ and the $\overline{WRITE/MSS}$ pins as shown in Table 5.6 below:-

Am2903 Position	$\overline{LSS}$	$\overline{WRITE/MSS}$
Least significant device	LOW	-
Intermediate devices	HIGH	HIGH
Most significant device	HIGH	LOW

When the $\overline{LSS}$ is tied LOW, the $\overline{WRITE/MSS}$ provides the $\overline{WRITE}$ output signal and it can be connected to the write enable pins, $\overline{WE}$, of all the devices so that writing to RAM or the 'Q' register will entirely be controlled by



Table 5.5 Am2903 Status Outputs

the $\overline{IEN}$ pin and the clock, CP, as shown below ('X' = don't care):-

$\overline{IEN}$	CP	RAM and 'Q' control
0	0	Write*
0	1	Inhibit writing
1	X	Inhibit writing

5.1.2 Cascading the Am2903s

Fig. 5.3 shows the conventional way of cascading [8] the Am2903s for a 16 bit data length, using a carry look ahead generator (Am2902). Here, the $\overline{G}$ and $\overline{P}$ outputs of the Am2903s are connected to the corresponding $\overline{G}$ and $\overline{P}$ inputs of the carry look ahead generator and the C_{n+x}, C_{n+y} and the C_{n+z} outputs of the carry look ahead generator are connected to the appropriate carry-in (C_n) inputs of the Am2903s.

Since our aim was to design the microcomputer with 8 and 16 bits of data lengths, then the Am2903s could not be cascaded by the conventional method. It was decided to use the two higher positioned Am2903s (devices 3 and 4

* It is necessary to enable the $\overline{OEY}$ control bit when writing to RAM (see Fig. 5.2).

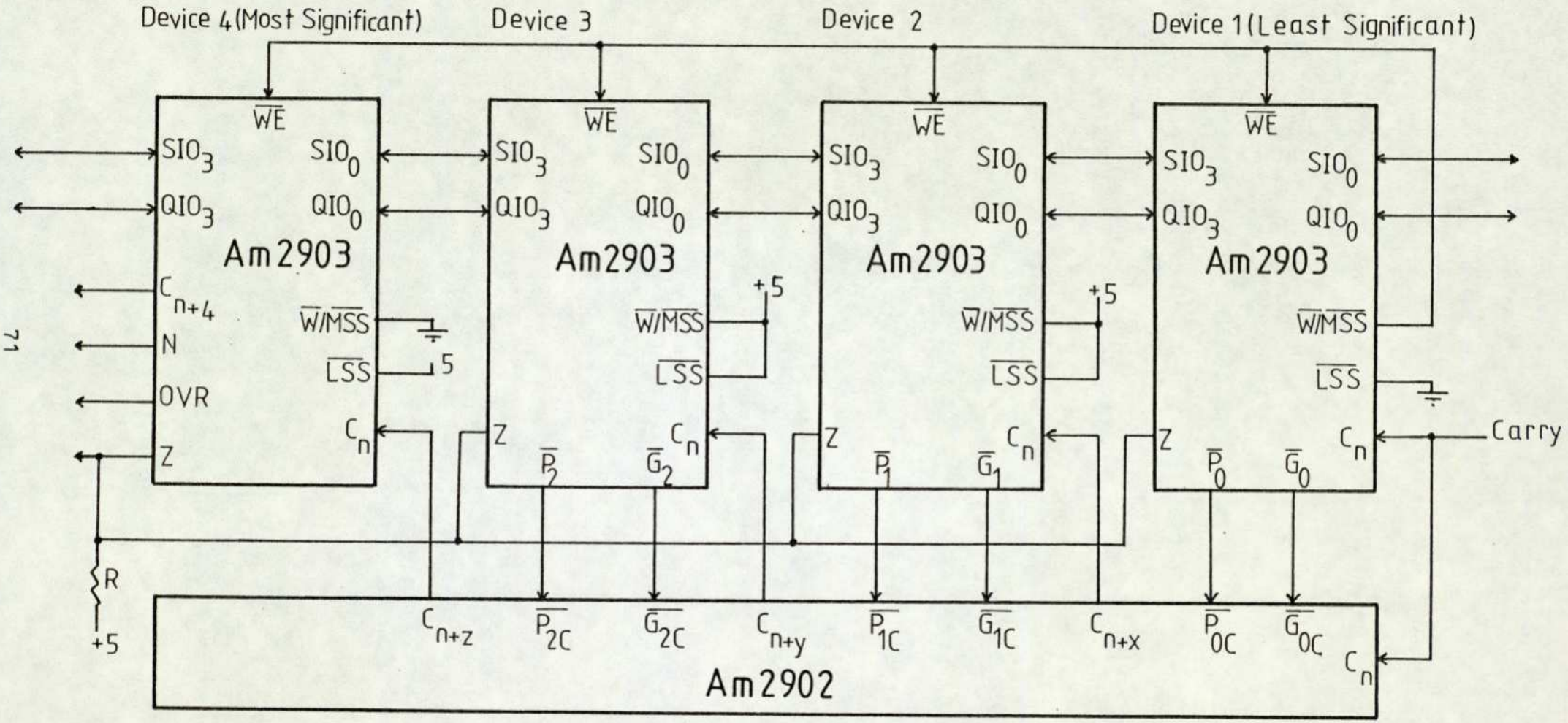


Figure 5.3 Conventional way of cascading the Am2903s for a 16 bit data length

in Fig. 5.3) during 8 bits of operation and to use all of the devices during 16 bits of operation. A switch S_1, was used to select the data length. When S_1 is HIGH, 16 bits of operation is selected. Then, the connections to the carry look ahead generator, the shift lines and the programming of the slice positions were controlled by the switch S_1 as described follows:-

i. Connections to the Carry Look Ahead Generator

During 8 bits of operation the C_{n+y} and the C_n pins of the carry look ahead generator must be at the same logic levels so that the two lower positioned Am2903s (devices 1 and 2) do not affect the operation of the circuit.

The Boolean operations describing the first two outputs of the carry look ahead generator (Am2902) are given by:

$$C_{n+x} = G_{oc} + P_{oc}C_n$$

$$C_{n+y} = G_{1c} + P_{1c}G_{oc} + P_{1c}P_{oc}C_n$$

now, C_{n+y} is equal to C_n if $G_{1c} = G_{oc} = 0$ and $P_{1c} = P_{oc} = 1$

The Am2902 is manufactured with the carry (G) and the propagate (P) signals inverted at their inputs. Hence, during 8 bits of operation the required carry look ahead function is:-

$$\overline{G_{oc}} = \overline{G_{1c}} = 1 \quad \text{and} \quad \overline{P_{oc}} = \overline{P_{1c}} = 0$$

Considering the data length select switch S_1, the following truth tables can be derived to define the connections between the Am2903s and the carry look ahead generator:-

S_1	$\overline{G}_o$	$\overline{G_{oc}}$	
0	0	1	8 bits of operation
0	1	1	
1	0	0	16 bits of operation
1	1	1	

S_1	$\overline{G}_1$	$\overline{G_{oc}}$	
0	0	1	8 bits of operation
0	1	1	
1	0	0	16 bits of operation
1	1	1	

S_1	$\overline{P}_o$	P_{oc}	
0	0	0	8 bits of operation
0	1	0	
1	0	0	16 bits of operation
1	1	1	

S_1	$\overline{P}_1$	$\overline{P_{1c}}$	
0	0	0	8 bits of operation
0	1	0	
1	0	0	16 bits of operation
1	1	1	

From the above truth tables it can be derived that,

$$\overline{G_{oc}} = \overline{S}_1 + \overline{G_o} \;, \quad \overline{G_{1c}} = \overline{S}_1 + \overline{G}_1$$

and $$\overline{P_{oc}} = S_1 \cdot \overline{P_o} \;, \quad \overline{P_{1c}} = S_1 \cdot \overline{P_1}$$

The above functions were used to connect the Am2903s to the carry look ahead generator as shown in Fig. 5.4.

ii. Programming the Slice Positions

The slice positions of the Am2903s are programmed as shown earlier in Table 5.6. During 8 bits of operation, Device 3 is the least significant device while during 16 bits of operation this is an intermediate device. Therefore, considering the data length select switch S_1, the following truth table can be derived for the $\overline{LSS}$ pin of device 3:

S_1	$\overline{LSS}$, device 3
0	0 , 8 bits of operation
1	1 , 16 bits of operation

Hence, the $\overline{LSS}$ pin of device 3 was directly connected to S_1.

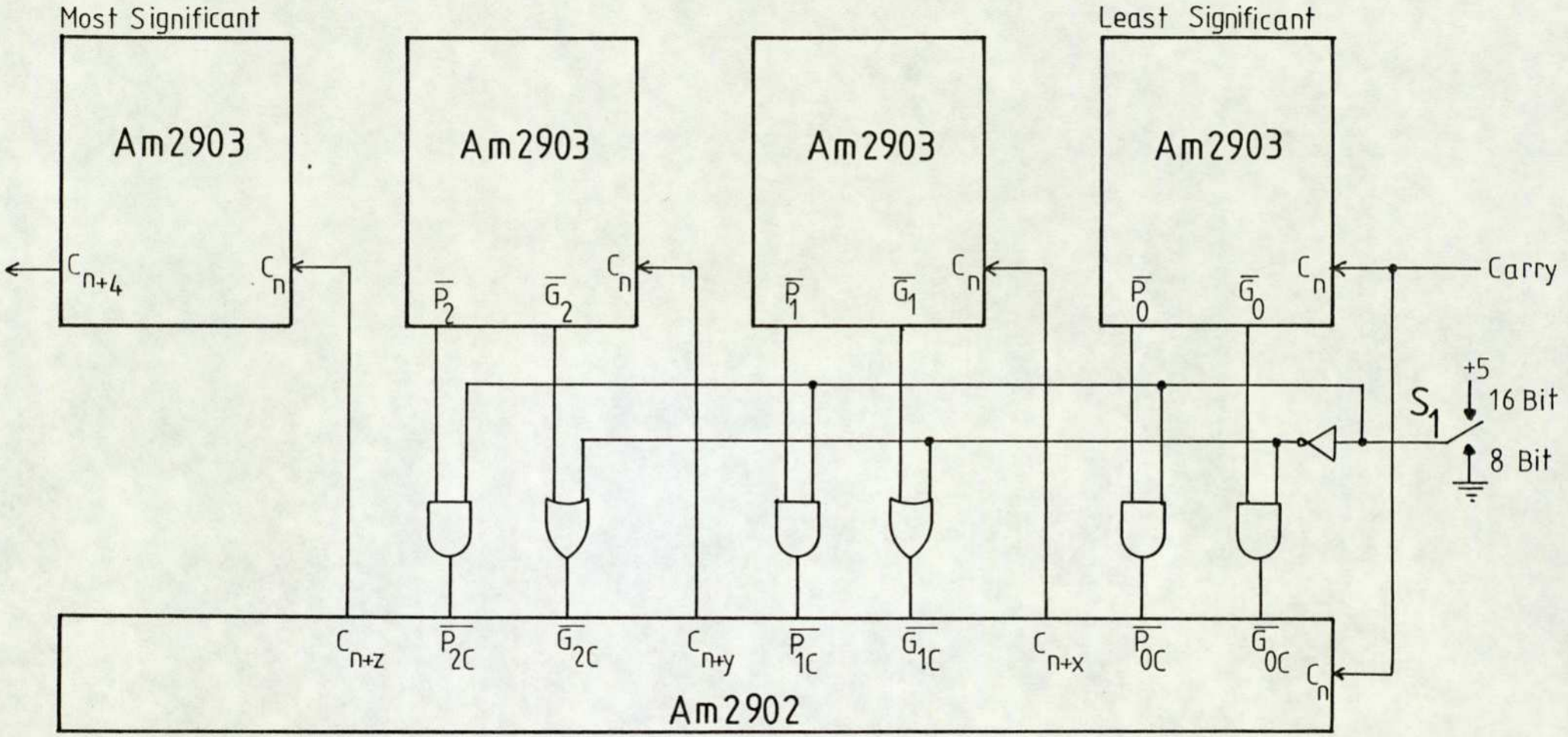


Figure 5.4 The Am2903s were cascaded by using the Am2902 carry look ahead generator and a few logic gates.

Similarly, during 16 bits of operation, the $\overline{WRITE/MSS}$ pin of device 3 must be at logic level HIGH while the same pin of device 1 must be connected to the $\overline{WE}$ pin of all the devices. Also, during 8 bits of operation, the $\overline{WRITE/MSS}$ pin of device 3 must be connected to the $\overline{WE}$ pins of all the devices. Considering the switch S_1, the following truth table can be derived for the $\overline{WE}$ inputs of all the devices.

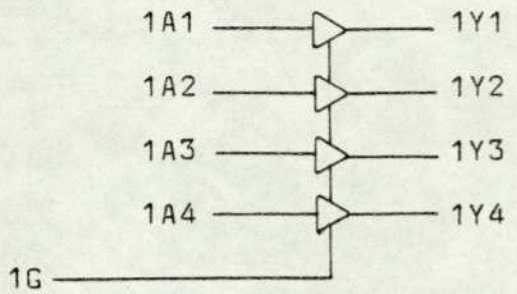
Switch S_1	Device 1 $\overline{WRITE/MSS}$ (W_1)	Device 3 $\overline{WRITE/MSS}$ (W_3)	$\overline{WE}$	
0	X	0	0	8 bits of operation
0	X	1	1	
1	0	X	0	16 bits of operation
1	1	X	1	

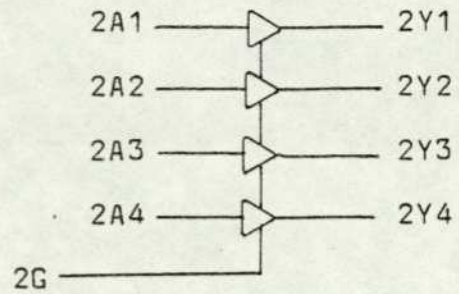
From the above truth table it can be derived that

$$\overline{WE} = \overline{S}_1 \cdot W_3 + W_1 \cdot W_3 + S_1 \cdot W_1$$

Since the $\overline{WRITE/MSS}$ pin of device 3 acts both as an input and output, a three-state buffer (Am74LS241) was used to control this pin.

The Am74LS241 is an octal three-state buffer organized as two groups of four non-inverting buffers. Each group is controlled separately by a control signal G as shown below:





1A	1G	1Y
X	1	Z, high impedence
0	0	0
1	0	1

2A	2G	2Y
X	0	Z, high impedence
0	1	0
1	1	1

Figure 5.5 shows the logic diagram of the ALU with the slice positions programmed for 8 or 16 bits of operation.

iii. Connecting the Shift Lines of the Am2903s

During 8 bits of operation, the QIO_0 and SIO_0 shift lines (see Fig. 5.3) of device 3 must be at the same logic levels as the QIO_0 and SIO_0 pins of device 1 respectively. Also, the QIO_3 and SIO_3 pins of device 2 must be disabled so that it does not affect the shift lines during 8 bits of operation. Remembering that the shift lines are bidirectional, they were controlled by using three-state buffers (Am74LS241) between device 2 and

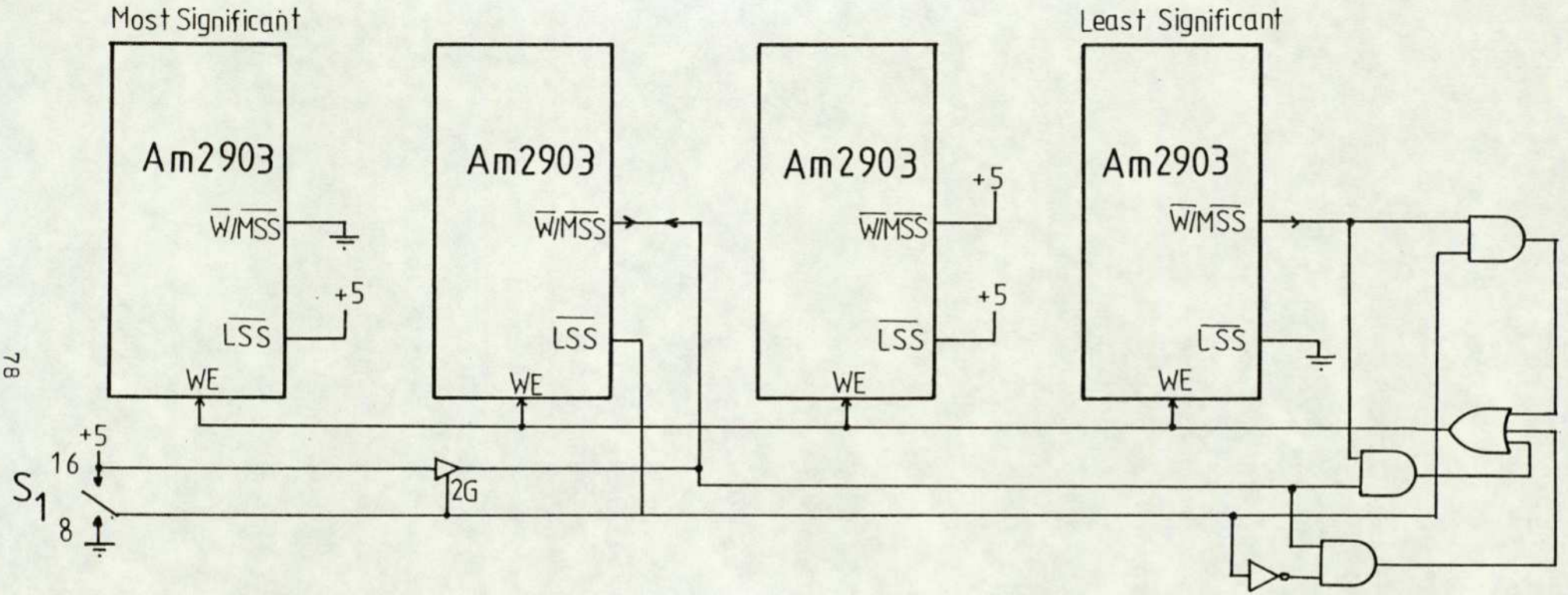
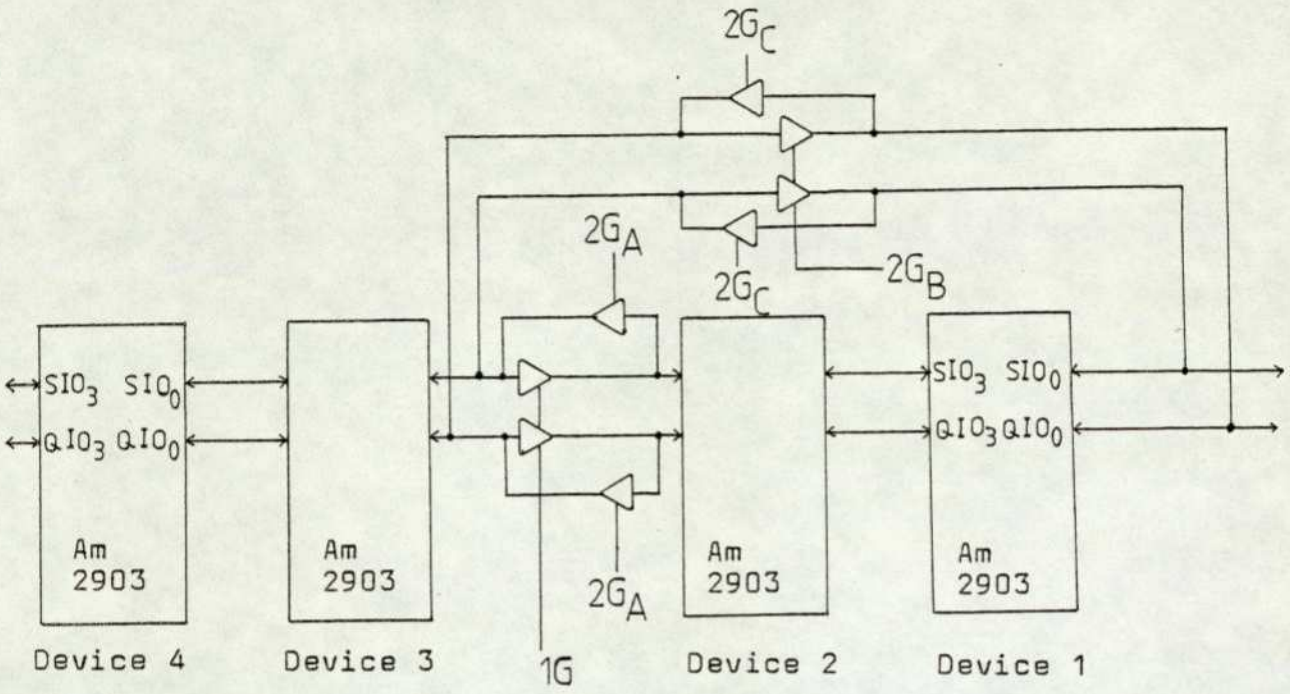


Figure 5.5 The slice positions of the Am2903s were programmed by using a three-state buffer and a few logic gates .

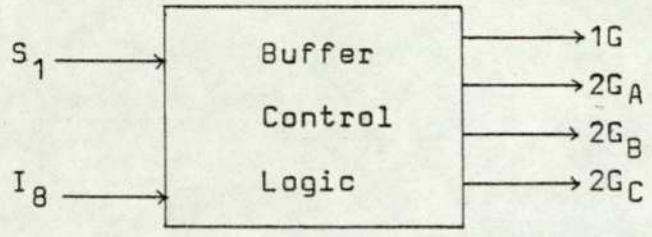
device 3 and also between device 1 and device 3 as shown below:-



During 8 bits of operation, the three-state buffers between device 2 and device 3 are disabled and the buffers between devices 1 and 3 are enabled. Also during 16 bits of operation, the buffers between device 1 and device 3 are disabled while the buffers between devices 2 and 3 are enabled.

Since the shift lines are bidirectional, it is necessary to enable the appropriate buffers during a shift-left or a shift-right operation. The Am2903 instruction bit I_8 controls the direction of shifting (see Table 5.3) such that when I_8 is LOW then a right-shift is performed and

when I_8 is HIGH a left-shift is performed. Therefore, the three-state buffers were controlled from the Am2903 instruction bit I_8 and the data length select switch S_1 via the Buffer Control Logic:



The Buffer Control Logic was implemented from the following truth table:-

S_1	I_8	1G	$2G_A$	$2G_B$	$2G_C$	
0	0	1	0	1	0	8 bits operation, right shift
0	1	1	0	0	1	8 bits operation, left shift
1	0	0	0	0	0	16 bits operation, right shift
1	1	1	1	0	0	16 bits operation, left shift

Hence, the three-state buffer control lines were derived from the following Boolean operations:-

$$1G = \overline{S}_1 + I_8$$

$$2G_A = S_1 \cdot I_8$$

$$2G_B = \overline{S}_1 \cdot \overline{I}_8$$

$$2G_C = \overline{S}_1 \cdot I_8$$

Fig. 5.6 shows the overall logic diagram of the airthmetic and logic unit of the microcomputer. In this figure the Am2903s are cascaded as described in this section. Also, the control and instruction bits common to the Am2903s (such as the 'A' and 'B' addresses, $\overline{IEN}$, $\overline{E}_A$ etc.) are connected together. Altogether 22 bits are required to control the ALU.

The following parts of this Chapter will describe the design of the remaining subsystems of the Arithmetic Processing Unit.

5.2 Design of the Bit Manipulation Logic (BML)

The function of the BML is to provide facilities to the arithmetic and logic unit so that the ALU can perform bit manipulation operations upon any of its internal working registers. The typical bit manipulation operations are:-

i. Set bit n : sets bit n of the specified register to logic level HIGH.

ii. Reset bit n : resets bit n of the specified register to logic level LOW.

iii. Test bit n : performs a conditional operation based upon the state of bit n of the specified register.

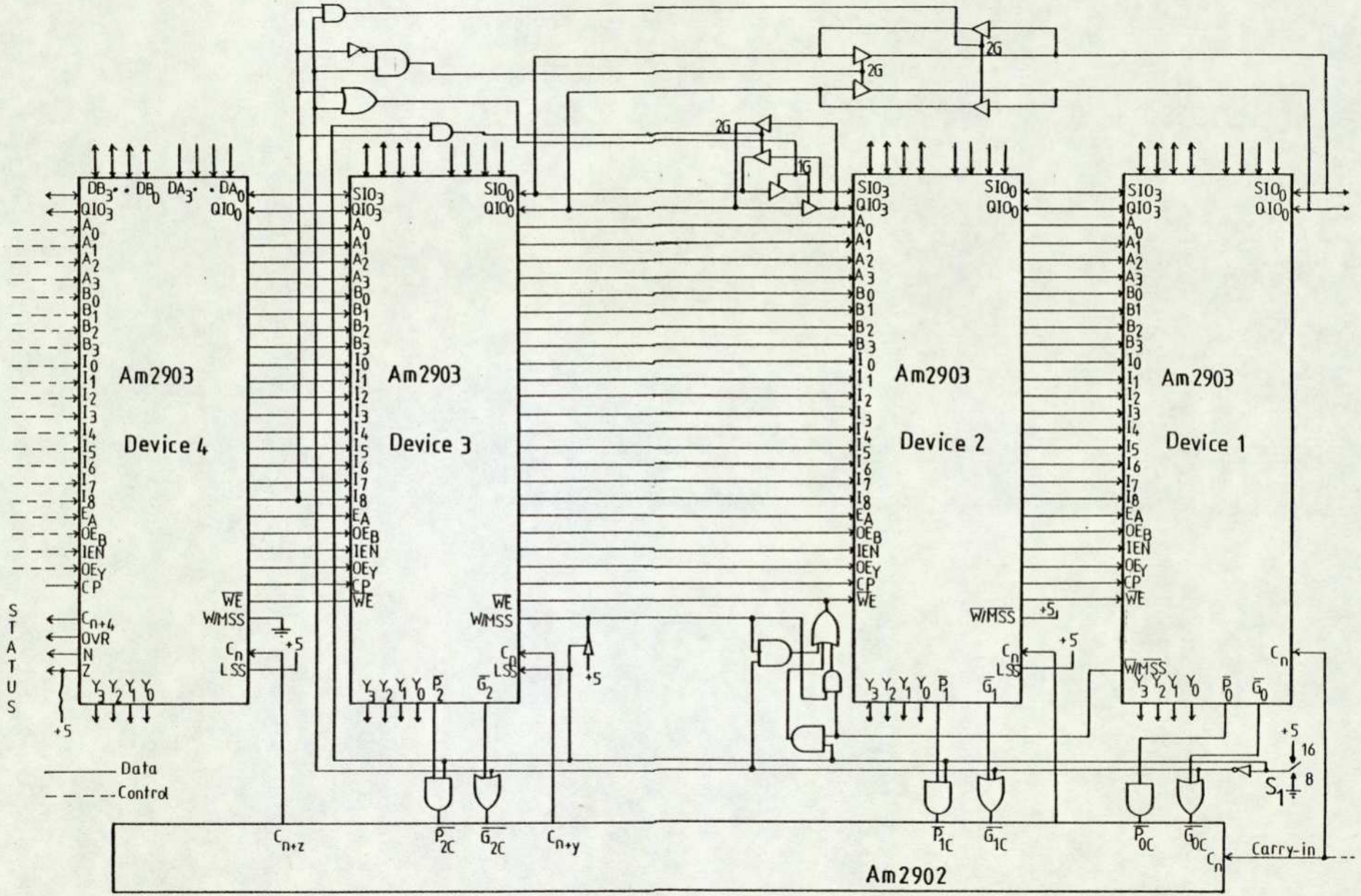


Figure 5.6 Logic diagram of the Arithmetic and Logic Unit

The bit manipulation logic was designed by using two Am25LS2538 type one-of-eight decoders. The operation of this decoder is shown in Table 5.7. The appropriate inputs of the two decoders were connected to the 'A' address field of the Am2903s and the 16 outputs from the decoders were connected to the 'DA' input ports of the Am2903s (see Fig. 5.7). Then, the two control bits 'E_4' and 'Pol' of the BML determine the nature of the bit operations as shown in Table 5.8.

The bit position of the register that is to be manipulated is determined by the 'A' address field of the Am2903s. Also, the register whose bits are to be manipulated is determined by the 'B' address field of the Am2903s.

During all bit manipulation operations one of the operands to the internal ALU of the Am2903s must be the register whose bits are to be manipulated (addressed by the 'B' port of RAM) and the other operand must be derived from the 'DA' inputs of the Am2903s. Then, for a bit-set or bit-reset operation, the following functions should be performed ('R_B'= Register addressed by the 'B' field of the Am2903s)

i. Bit-set

During a bit-set operation, the BML control bits E_4 and Pol must be at logic 1 and 0 respectively and the following operations should be performed internal to the Am2903s :-

INPUTS										OUTPUTS							
$\overline{OE}_1$	$\overline{OE}_2$	$\overline{E}_1$	$\overline{E}_2$	E_3	E_4	POL	C	B	A	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
1	X	X	X	X	X	X	X	X	X	Z	Z	Z	Z	Z	Z	Z	Z
X	1	X	X	X	X	X	X	X	X	Z	Z	Z	Z	Z	Z	Z	Z
0	0	1	X	X	X	0	X	X	X	0	0	0	0	0	0	0	0
0	0	1	X	X	X	1	X	X	X	1	1	1	1	1	1	1	1
0	0	X	1	X	X	0	X	X	X	0	0	0	0	0	0	0	0
0	0	X	1	X	X	1	X	X	X	1	1	1	1	1	1	1	1
0	0	X	X	0	X	0	X	X	X	0	0	0	0	0	0	0	0
0	0	X	X	0	X	1	X	X	X	1	1	1	1	1	1	1	1
0	0	X	X	X	0	0	X	X	X	0	0	0	0	0	0	0	0
0	0	X	X	X	0	1	X	X	X	1	1	1	1	1	1	1	1
0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	1	0	1	0	0	0	0	0	0
0	0	0	0	1	1	0	0	1	0	0	0	1	0	0	0	0	0
0	0	0	0	1	1	0	0	1	1	0	0	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0	0	1	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	1	0
0	0	0	0	1	1	0	1	1	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	1	0	0	0	0	1	1	1	1	1	1	1
0	0	0	0	1	1	1	0	0	1	1	0	1	1	1	1	1	1
0	0	0	0	1	1	1	0	1	0	1	1	0	1	1	1	1	1
0	0	0	0	1	1	1	0	1	1	1	1	1	0	1	1	1	1
0	0	0	0	1	1	1	1	0	0	1	1	1	1	0	1	1	1
0	0	0	0	1	1	1	1	0	1	1	1	1	1	1	0	1	1
0	0	0	0	1	1	1	1	1	0	1	1	1	1	1	1	0	1
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0

X = Don't care Z = High impedance

Table 5.7 Function Table of the Am25LS2538.

BML		Am2903 RAM A				Am2903 DA INPUTS															
E_4	Pol	A_3	A_2	A_1	A_0	DA_{15}															DA_0
0	0	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	X	X	X	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
1	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
1	1	0	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
1	1	1	0	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
1	1	1	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
1	1	1	0	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

X = don't care entry

Table 5.8 Output (DA) of the Bit Manipulation Logic as a function of the RAM address 'A' and the BML control bits E_4 and Pol .

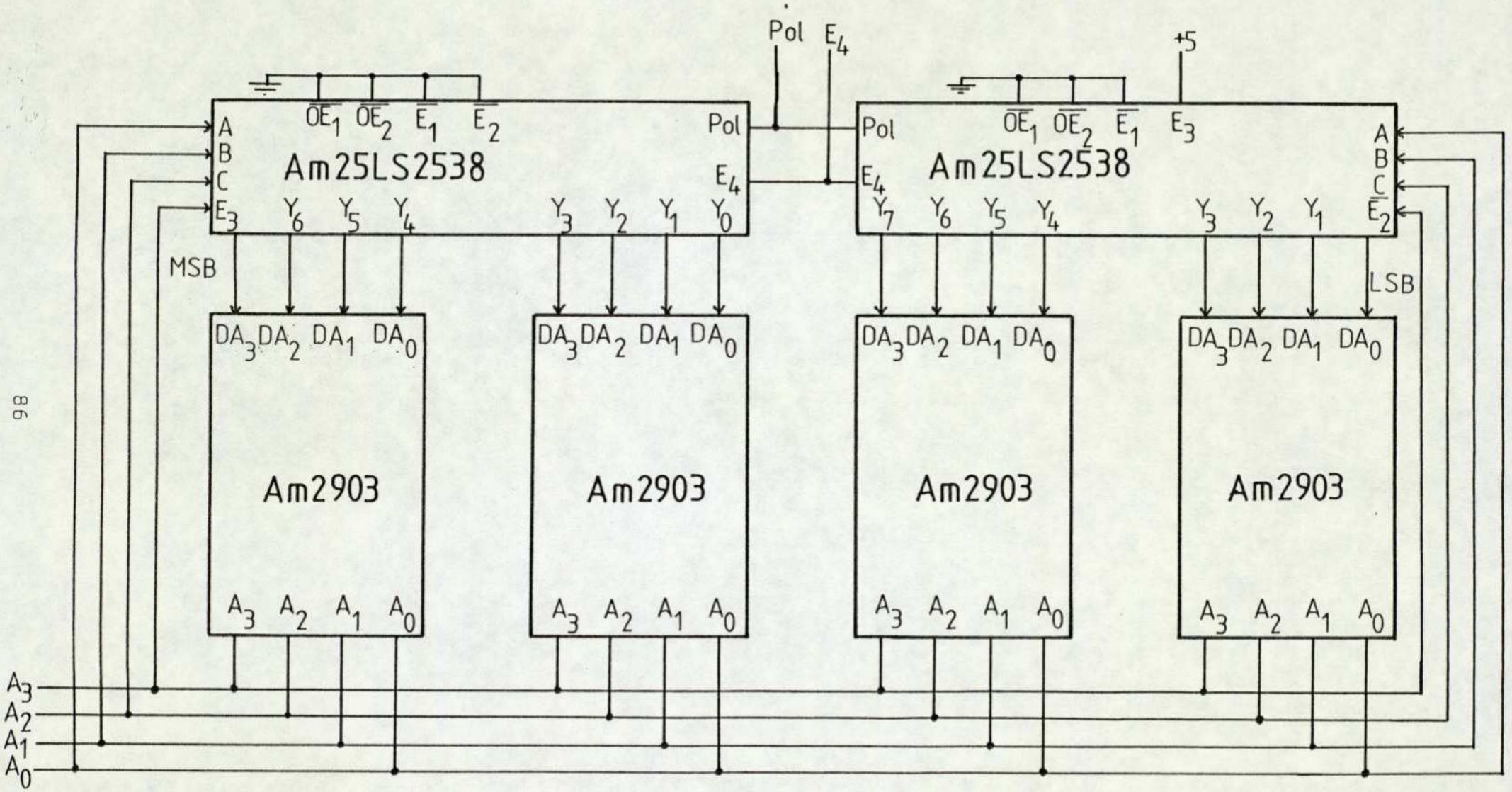


Figure 5.7 Logic diagram of the Bit Manipulation Logic (BML)

Source of ALU data : R_B and DA

ALU function : R_B.OR.DA

ALU destination : R_B

As an example, suppose that it is required to set bit 5 of register 1. Then, the 'A' and 'B' addresses of Am2903s must be 5 and 1 respectively and the Am2903s should be controlled to perform the above operations. The microcode for this example is shown below

BML CONTROL		ALU CONTROL															
		RAM		Source			Function				Destination				Other		
E_4	Pol	A	B	I_0	$\overline{E_A}$	$\overline{OE_B}$	I_4	I_3	I_2	I_1	I_8	I_7	I_6	I_5	$\overline{IEN}$	$\overline{OE_y}$	
1	0	0101	0001	0	1	0	1	1	1	1	0	1	0	0	0	0	

ii. Bit-reset

During a bit-reset operation, the BML control bits E_4 and Pol must both be at logic 1 and the following operations should be performed internal to the Am2903s:-

Source of ALU data : R_B and DA

ALU function : R_B.AND.DA

ALU destination : R_B

As an example, suppose that it is required to reset bit 3

of register 2. Then, the 'A' and 'B' addresses of Am2903s must be 3 and 2 respectively and the Am2903s should be controlled to perform the above operations. The microcode for this example is shown below:-

BML CONTROL		ALU CONTROL													
		RAM		Source			Function				Destination				Other	
E_4	Pol	A	B	I_0	$\overline{E_A}$	$\overline{OE_B}$	I_4	I_3	I_2	I_1	I_8	I_7	I_6	I_5	$\overline{IEN}$	$\overline{OE_Y}$
1	1	0011	0010	0	1	0	1	0	0	1	0	1	0	0	0	0

5.3 Design of the Data Bus Buffers

The Data Bus provides the communication of the Arithmetic Processing Unit with the rest of the computer. The instructions and data are transmitted along this 16 bit bidirectional bus (during 8 bits of operation only the upper byte is used). The data transmission along the Data Bus was controlled such that all the devices that send data to this bus have three-state buffers at their outputs and for proper operation of the Data Bus, only one of them must be enabled at any instant. Also to minimize the DC loading on the Data Bus, all the devices that receive data from this bus were chosen to be of low Power Schottky type wherever possible. The Data Bus was created from the bidirectional 'DB' and 'Y' ports

of the Am2903s as described follows.

5.3.1 Connecting the Y port lines of the Am2903s

The 16 Y port lines were connected as an output to the Data Bus so that the result of an arithmetic or logical operation could be transferred to the other subsystems of the microcomputer.

During the execution of an instruction which writes data to the RAM of the Am2903s, the $\overline{OE}_Y$ control bit of the ALU must be enabled so that the three-state buffers at the outputs of the Y ports become transparent (see Fig. 5.2). It then follows that the Y ports of the Am2903s could not be directly connected to the Data Bus since whenever $\overline{OE}_Y$ is enabled it may not be desirable to send the result of the ALU to the Data Bus, as this might conflict the data already existing on the bus. This situation was avoided by connecting the Y port lines to the Data Bus via the three-state buffers, Am74LS241 (see section 5.1.2 for the operation of these buffers).

As shown in Fig. 5.8, these buffers were controlled by the single control bit 'EDB'. The control of the Data Bus can then be described as in the following table.

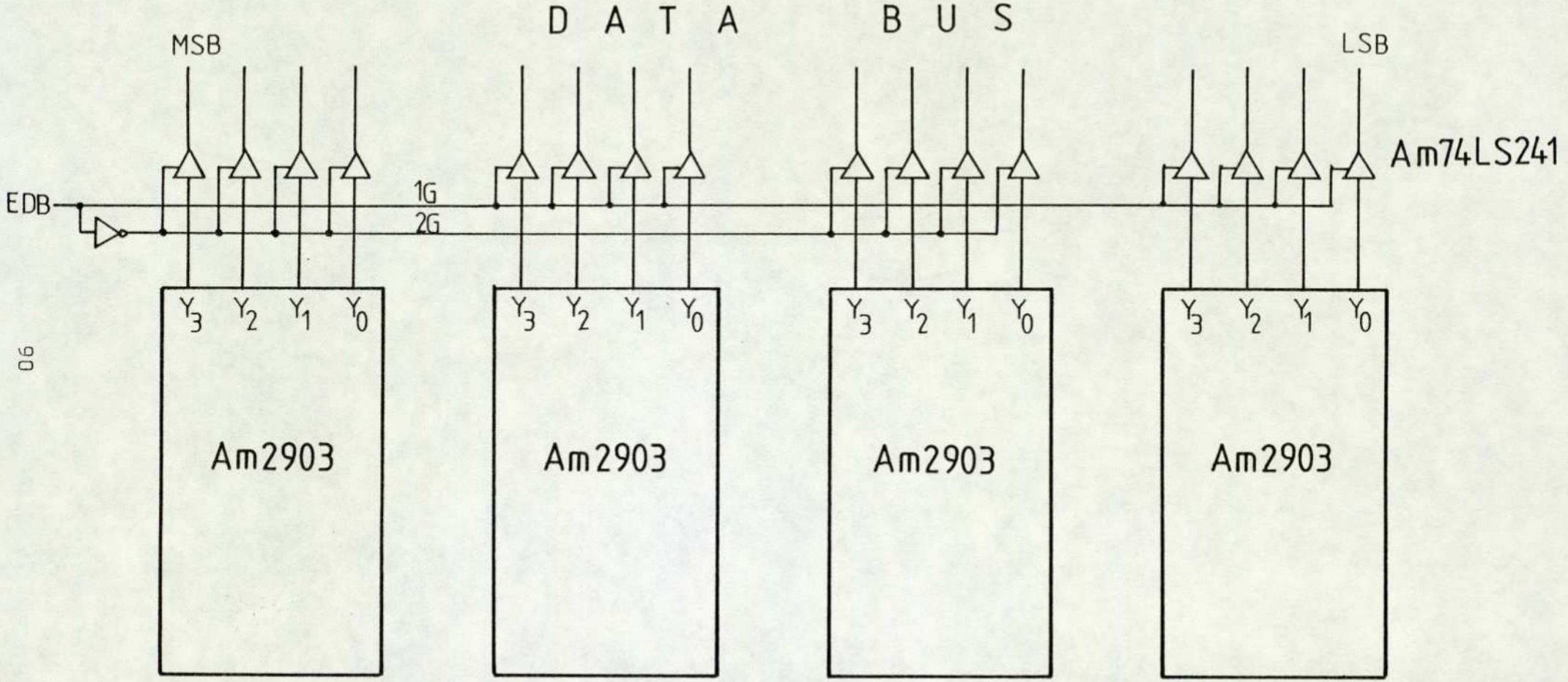


Figure 5.8 The Data Bus was created from the Y port lines of the Am2903s

EDB	Data Bus
0	Enable Y to Data Bus
1	Disable Y from Data Bus

In some cases it may be desirable to send the result of an ALU operation to the Data Bus without having to modify the contents of the RAM. Under these circumstances it will be necessary to disable the $\overline{IEN}$ (inhibit writing to RAM) and enable both $\overline{OE}_Y$ and EDB.

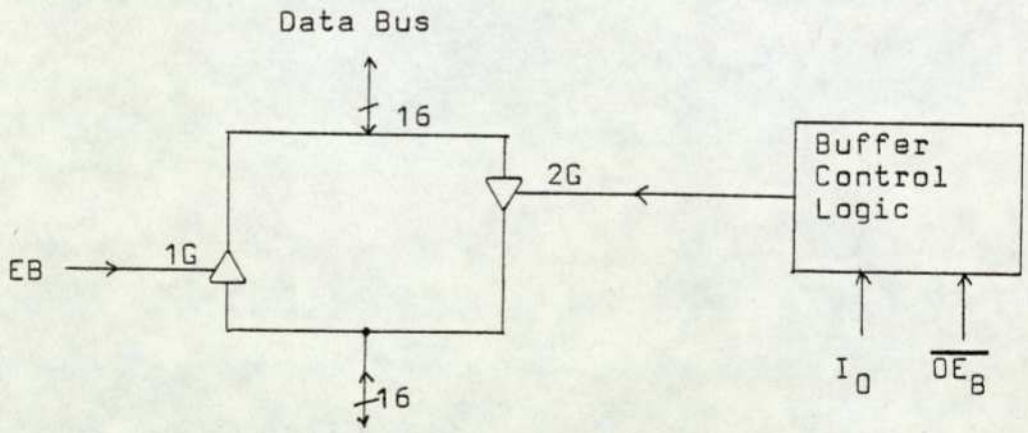
The following truth table describes the destination of an ALU operation in terms of the control bits $\overline{IEN}$, $\overline{OE}_Y$ and EDB.

$\overline{IEN}$	$\overline{OE}_Y$	$\overline{EDB}$	Destination of the ALU result
0	0	0	RAM and Data Bus
0	0	1	RAM only
0	1	0	Not allowed
0	1	1	Not allowed
1	0	0	Data Bus only
1	0	1	No change
1	1	0	Not allowed
1	1	1	No change

5.3.2 Connecting the DB Port Lines of the Am2903s

The 16 DB bidirectional port lines of the Am2903s were connected as an input and output to the Data Bus. As an input, these lines are used to receive external data from the Data Bus to the ALU and as an output they can be used to send the data in any RAM registers of the ALU to the Data Bus, without having to perform any arithmetic or logical operation (see Fig. 5.2).

The DB port lines were connected to the Data Bus via the three-state buffers, Am74LS241, organized in a bidirectional configuration and controlled by the three control bits EB, I_o and $\overline{OE}_B$ as follows:-



DB ports of Am2903s

The output from the DB port of the Am2903s were controlled by the control bit EB. When EB is LOW, the DB data appears at the Data Bus and when it is HIGH, these buffers are in a high impedance state.

The input to the DB port was controlled by the Am2903 control bits I_o and $\overline{OE}_B$ and the buffer control logic. Referring to the Am2903 internal ALU operand sources (Table 5.1), it is clear that the DB port is selected to the ALU whenever I_o is LOW and $\overline{OE}_B$ is HIGH. Remembering that the three-state buffers are enabled when 2G is HIGH, the function of the buffer control logic was derived from the following truth table:-

I_o	$\overline{OE}_B$	2G
0	0	0
0	1	1
1	0	0
1	1	0

Thus, $2G = \overline{I}_o.\overline{OE}_B$

The connection of the DB port lines to the Data Bus are shown in Fig.5.9 and the function of the three control bits EB, I_o and $\overline{OE}_B$ are summarized below:-

EB	I_o	$\overline{OE}_B$	Data transfer
0	0	0	DB to Data Bus
0	0	1	Not allowed
0	1	0	DB to Data Bus
0	1	1	DB to Data Bus
1	0	0	No change
1	0	1	Data Bus to DB
1	1	0	No change
1	1	1	No change

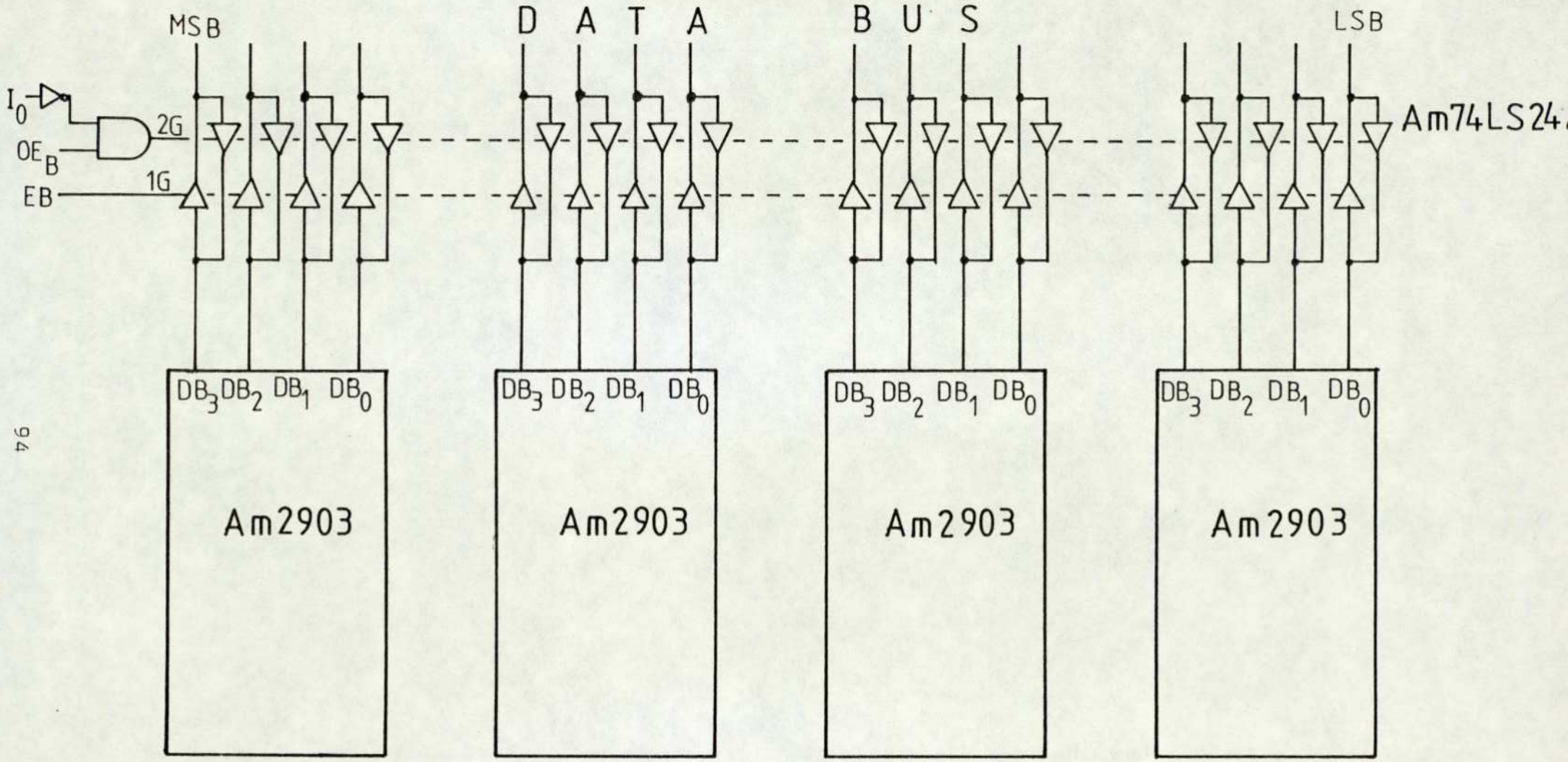


Figure 5.9 The DB port lines were connected to the Data Bus via three-state buffers

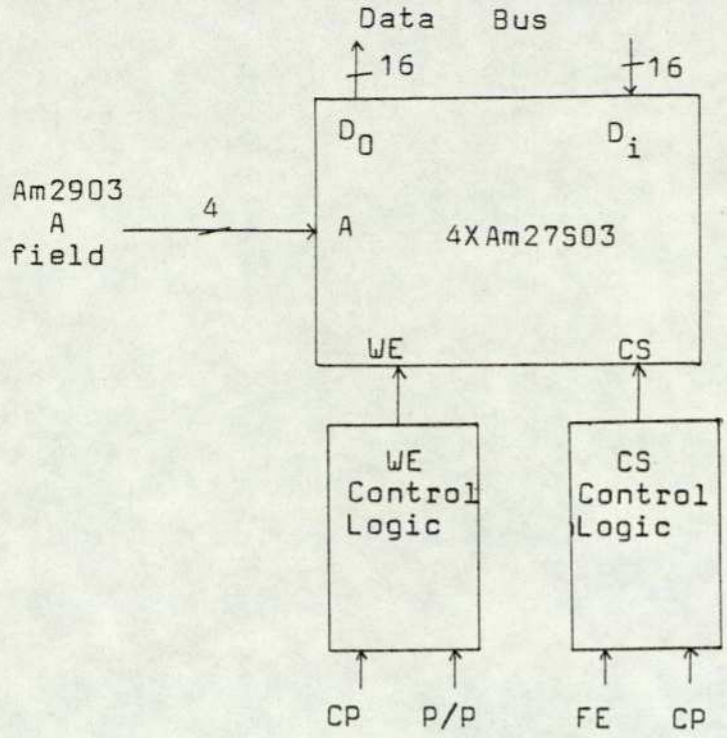
5.4 Design of the Register Extension Logic (REL)

The Am2903s contain 16 RAM registers and a Q register available to the user. The variables in an application can either be stored in these registers or in the main memory of the microcomputer. Because of the slow access time of the main memory, the register to register operations are faster than the memory to register or memory to memory operations. It was therefore decided to increase the number of registers of the system so that a greater number of variables could be stored in the registers to give a shorter processing time.

The REL was designed by using four 16 word by 4 bit bipolar read/write memories (Am27S03) connected to make a 16 words by 16 bits register file (this register file was named as the 'auxiliary registers').

The address inputs of this register file were connected to the 'A' address fields of the Am2903s so that the same addresses could be used to select the auxiliary registers and/or the 16 Am2903 RAM registers. The data input and output pins of the register file were connected to the system Data Bus.

The WE (read/write control) and the CS (chip select) pins of the REL were controlled by the control bits P/P (Push/Pop), Fe (file enable) and the system clock CP as follows:-



The WE control logic and the CS control logic were derived from the following truth table ('X' = don't care entry).

P/P	FE	CP	WE	CS	function
0	0	0	0	0	PUSH: write to register file
0	0	1	1	1	No register file operation
0	1	0	1	1	No register file operation
0	1	1	1	1	No register file operation
1	0	0	1	0	POP: read from register file
1	0	1	1	1	No register file operation
1	1	0	1	1	No register file operation
1	1	1	1	1	No register file operation

Data is written (PUSH) to the register file on the LOW to HIGH transition of the clock input CP, when P/P and FE are both LOW. Similarly Data is read out (POP) from

the register file when the clock input CP and the file enable FE are both LOW and the P/P control bit is HIGH.

The WE and the CS control logic were then implemented from the following Boolean operations as shown in Fig.5.10 (note that the data input to the register file is inverted to compensate the internal data inversion of the Am27S03).

$$WE = P/P + CP$$

and $$CS = FE + CP$$

5.4.1 Examples on Microcoding the REL

It is worthwhile to consider some examples to clarify the operation of the REL.

i. PUSH: Suppose that it is required to transfer the contents of register 2 of the ALU to the auxiliary register 7. The microcode for this example is then as follows:-

REL		BML		ALU CONTROL								
P/P	FE	E_4	Pol	RAM A	RAM B	Source $I_0\overline{E}_A\overline{OE}_B$	Func. $I_4I_3I_2I_1$	Dest. $I_8I_7I_6I_5$		$\overline{IEN}$	$\overline{OE}_Y$	EDB
0	0	0	0	0111	0010	0 1 0	1 1 1 1	0 1 0 0		0	0	0

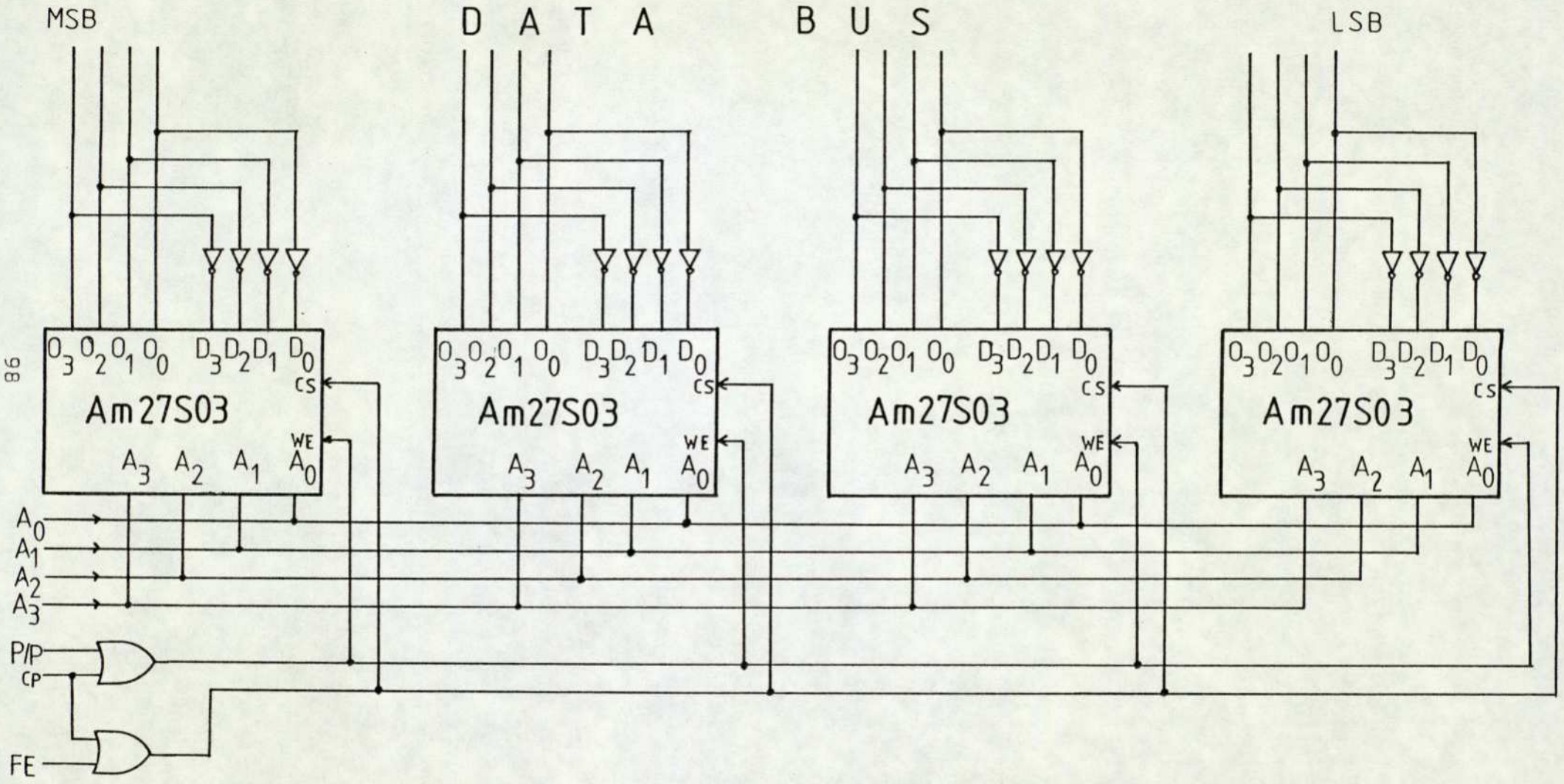


Figure 5.10 The Register Extension Logic (REL)

ii. Suppose that it is required to transfer the contents of auxiliary register 5 to the ALU register 3. The microcode is then as follows:-

REL	BML	ALU CONTROL					
P/P FE	E_4 Pol	RAM A B	Source $I_0\overline{E}_A\overline{OE}_B$	Func. $I_4I_3I_2I_1$	Dest. $I_8I_7I_6I_5$	$\overline{IEN}$ $\overline{OE}_Y$	EDB
1 0	0 0	0101 0011	0 1 1	1 1 1 1	0 1 0 0	0 0	1

5.5 Design of the Condition Code Select Logic (CCS)

The condition code select logic stores the system status signals (such as carry, overflow, sign etc.), selects the required status bit and presents it to the CCU for executing the conditional instructions (such as branch-on-zero, branch-on-positive etc.).

The block diagram of the CCS is shown in Fig. 5.11. The functions of the different parts of the CCS are as follows:-

i. Condition Multiplexer

The condition multiplexer receives 8 different test signals from several points of the ALU (such as the LSB or MSB of the Am2903s) and under the control of S_1, S_2 and S_3

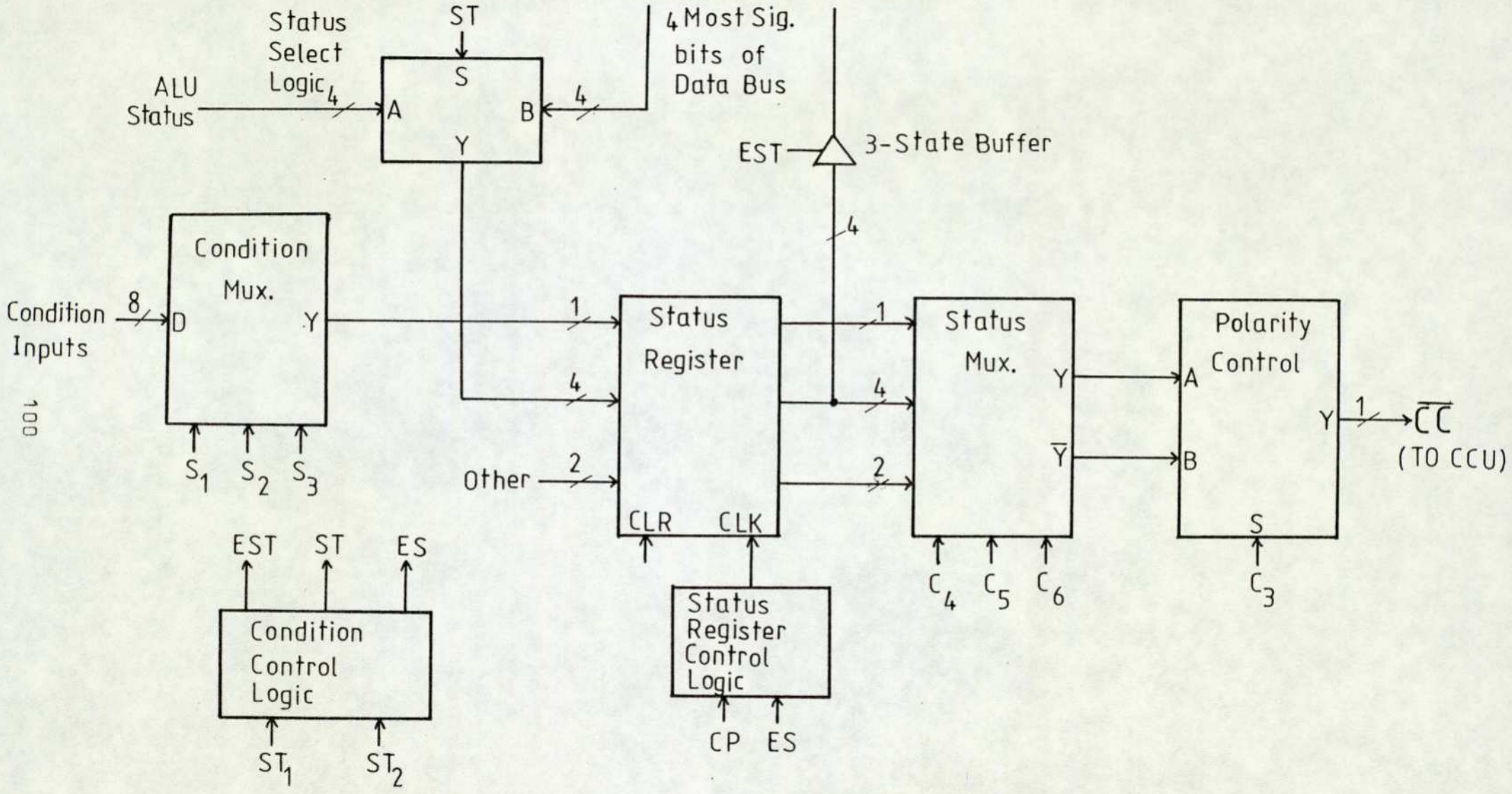


Figure 5.11 Block diagram of the Condition Code Select Logic (CCS)

control bits, it presents one of them to the status register. The condition multiplexer was designed from an Am74LS151 type 8 to 1 multiplexer.

ii. Status Select Logic

The status select logic is a four bit 2 line to 1 line data selector (Am74SI57). When ST is LOW, the A input data appears at Y and when ST is HIGH, the B input data is selected at Y. The status select logic thus selects either the four ALU status signals (carry, overflow, zero and sign) or the data at the four most significant bits of the Data Bus as the source for the status register.

iii. Status Register

This is an 8 bit clocked register (Am74LS273) which stores the status signals. The clock input to the status register was controlled from the status register control logic which enables or inhibits loading depending upon the system clock CP and the enable input ES. The status register can be loaded from the Data Bus whenever ST is at logic HIGH and the clock input is enabled. Similarly, the ALU status of the status register can be sent to the Data Bus by controlling the three-state buffers (Am74LS241) at the output of the status register (this facility is very useful whenever it is required to save the ALU status in a temporary register or in

the main memory e.g. during processor interrupts).

It is very important to realise that there is one clock period delay between an ALU operation and loading the status register with the ALU status. The status register contains the status of the ALU operation which was executed one clock period earlier. Therefore, a conditional operation can only be executed based upon the ALU status of the previous ALU operation.

iv. Status Multiplexer

The status multiplexer selects one of the status register data and presents it to the polarity control logic. It is controlled by C_4, C_5 and C_6. The status multiplexer was designed from Am74LS151 type 8 to 1 multiplexer.

v. Polarity Control Logic

The polarity control logic selects either the true or the inverted data from the status multiplexer and presents it to the CCU for executing the conditional operations.

5.5.1 Controlling the Condition Code Select Logic

The CCS requires 10 bits to be controlled from the

CCU for its operation. The control bits S_1, S_2, S_3, C_3, C_4, C_5 and C_6 were directly derived from the CCU, while the control bits ST, ES and EST were controlled from ST_1 and ST_2 of the condition control logic. By this approach, the total number of bits to control the CCS were reduced from 10 to 9 bits.

The status register control logic and the condition control logic were implemented as follows:-

i. Status Register Control Logic

The status register control logic controls the clock input (CLK) of the status register thus enabling or inhibiting loading. The status register is clocked on the LOW to HIGH transition of the system clock CP, whenever ES is LOW:-

CP	ES	CLK
0	0	0
0	1	1
1	0	1
1	1	1

Hence, CLK = CP + ES

ii. Condition Control Logic

The condition control logic controls the bits ST, ES and EST as described in the following truth table:-

ST_1	ST_2	ES	ST	EST	Status register function
0	0	0	0	1	Clock new status from ALU
0	1	0	1	1	Clock new status from Data Bus
1	0	1	0	0	Send the ALU status to Data Bus
1	1	1	X	1	Inhibit loading the status register

The control bits ES, ST and EST were therefore derived from the following Boolean operations:-

$$ES = ST_1$$

$$ST = ST_2$$

and $$EST = \overline{ST}_1 + ST_2$$

Fig. 5.12 shows the overall logic diagram of the condition code select logic. The source of data for the status register and the condition presented to the CCU are tabulated in Tables 5.9 and 5.10 respectively.

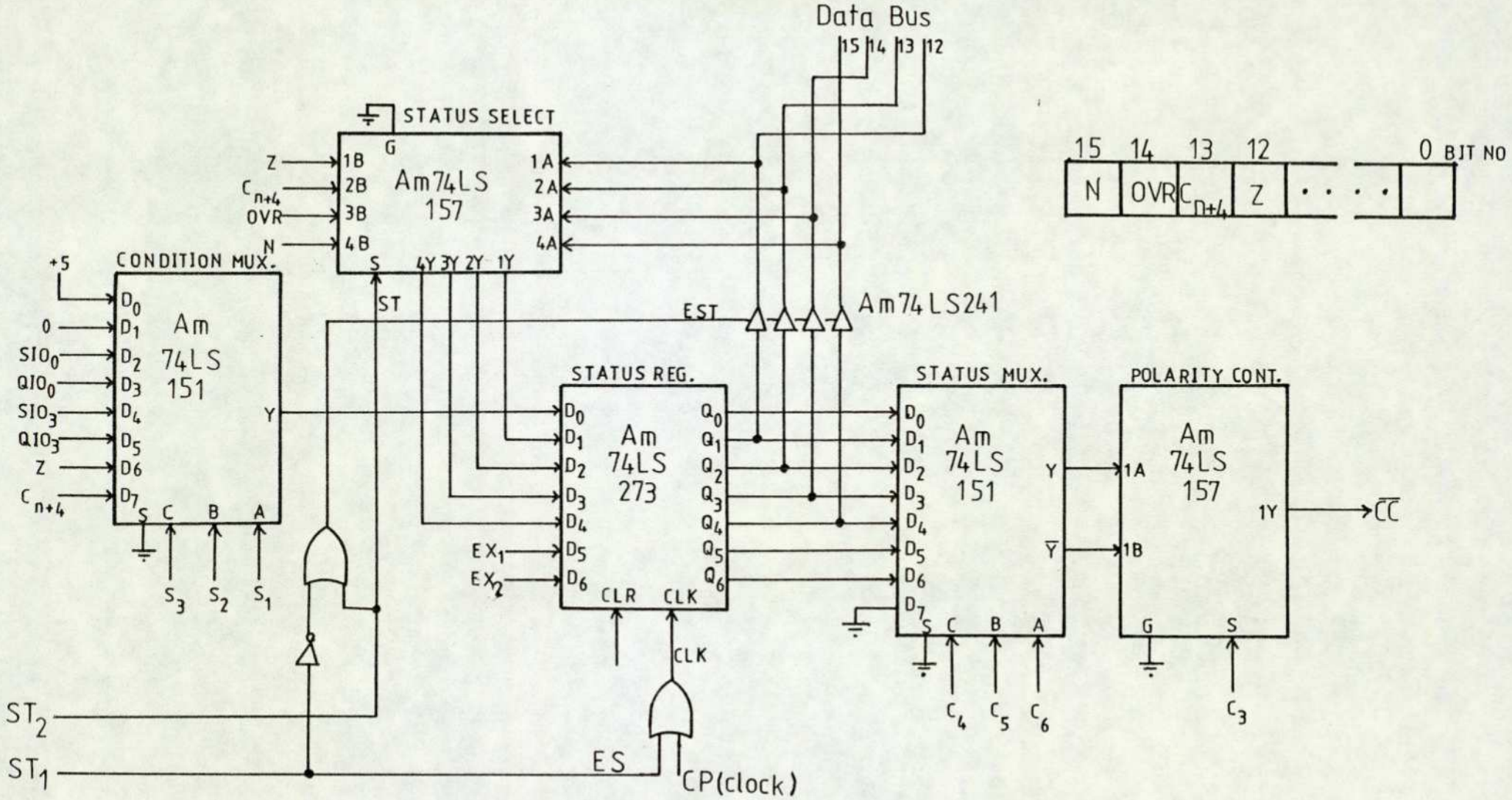


Figure 5.12 The Condition Code Select Logic is controlled by the nine bits $S_1, S_2, S_3, C_3, C_4, C_5, C_6, ST_1$ and ST_2 .

CONTROL					Status Register Contents (after the clock LOW to HIGH transition)						
ST_1	ST_2	S_3	S_2	S_1	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	EX_2	EX_1	N	OVR	C_{n+4}	Z	1
0	0	0	0	1	EX_2	EX_1	N	OVR	C_{n+4}	Z	0
0	0	0	1	0	EX_2	EX_1	N	OVR	C_{n+4}	Z	SIO_0
0	0	0	1	1	EX_2	EX_1	N	OVR	C_{n+4}	Z	QIO_0
0	0	1	0	0	EX_2	EX_1	N	OVR	C_{n+4}	Z	SIO_3
0	0	1	0	1	EX_2	EX_1	N	OVR	C_{n+4}	Z	QIO_3
0	0	1	1	0	EX_2	EX_1	N	OVR	C_{n+4}	Z	Z
0	0	1	1	1	EX_2	EX_1	N	OVR	C_{n+4}	Z	C_{n+4}
0	1	0	0	0	EX_2	EX_1	DB_{15}	DB_{14}	DB_{13}	DB_{12}	1
0	1	0	0	1	EX_2	EX_1	DB_{15}	DB_{14}	DB_{13}	DB_{12}	0
0	1	0	1	0	EX_2	EX_1	DB_{15}	DB_{14}	DB_{13}	DB_{12}	SIO_0
0	1	0	1	1	EX_2	EX_1	DB_{15}	DB_{14}	DB_{13}	DB_{12}	QIO_0
0	1	1	0	0	EX_2	EX_1	DB_{15}	DB_{14}	DB_{13}	DB_{12}	SIO_3
0	1	1	0	1	EX_2	EX_1	DB_{15}	DB_{14}	DB_{13}	DB_{12}	QIO_3
0	1	1	1	0	EX_2	EX_1	DB_{15}	DB_{14}	DB_{13}	DB_{12}	Z
0	1	1	1	1	EX_2	EX_1	DB_{15}	DB_{14}	DB_{13}	DB_{12}	C_{n+4}
1	X	X	X	X	NO CHANGE						

DB_i : i^{th} bit of the Data Bus.

Z = zero status, OVR = overflow status, C_{n+4} = carry status, N = sign status.

EX_1,EX_2 : Reserved for future expansion of the CCS .

SIO_0,SIO_3 = ALU shift input-output lines (The appropriate shift operations must be performed by the ALU to bring the LSB or MSB of RAM registers out of SIO_0 or SIO_3 during loading the status register. This is because SIO_0 and SIO_3 are not latched after an ALU operation) .

QIO_0,QIO_3 = ALU shift input-output lines (The appropriate shift operations must be performed by the ALU to bring the LSB or MSB of the Q register out of QIO_0 or QIO_3 during loading the status register. This is because QIO_0 and QIO_3 are not latched after an ALU operation)

Table 5.9 The source of data for the status register is controlled by S_1, S_2 , S_3 , ST_1, and ST_2 .

CONTROL				CCU Test input
C_4	C_5	C_6	C_3	$\overline{CC}$
0	0	0	0	D_0
0	0	0	1	$\overline{D_0}$
0	0	1	0	D_1
0	0	1	1	$\overline{D_1}$
0	1	0	0	D_2
0	1	0	1	$\overline{D_2}$
0	1	1	0	D_3
0	1	1	1	$\overline{D_3}$
1	0	0	0	D_4
1	0	0	1	$\overline{D_4}$
1	0	1	0	D_5
1	0	1	1	$\overline{D_5}$
1	1	0	0	D_6
1	1	0	1	$\overline{D_6}$
1	1	1	0	0*
1	1	1	1	1*

* These two conditions are not selected from the status reg.

D_{0-6} : Contents of the status register (see Table 5.9) .

Table 5.10 The condition presented to the CCU is selected from C_3, C_4, C_5, and C_6 control bits of the CCS .

5.5.2 Examples on Microcoding the CCS

It is worthwhile at this stage to consider some examples to clarify the operation of the condition code select logic.

Example 1

Suppose that it is required to add the contents of register 1 and register 3 of the ALU, load the status register with the resulting status and then send the carry status (C_{n+4}) to the CCU to perform conditional operations based upon the 'carry' of the addition (such as branch on carry, branch on non-carry etc.).

The microcode for this example is shown below ('X' = don't care entry).

CCS Control	ALU CONTROL					
$ST_1ST_2C_4C_5C_6S_3S_2S_1C_3$	RAM		Source	Function	Dest.	Other
	A	B	$I_0\overline{E}_A\overline{OE}_B$	$I_4I_3I_2I_1$	$I_8I_7I_6I_5$	$C_n\overline{IEN}\ \overline{OE}_Y$
0 0 X X X X X X X	0001	0011	0 0 0	0 0 1 1	0 1 0 0	0 0 0
1 1 0 1 0 X X X 0	XXXX	XXXX	X X X	X X X X	X X X X	X 1 X

Two microinstructions are required to execute the above operation; In the first microinstruction the addition

is performed and the status register is enabled ($ST_1 = ST_2 = 0$). On the next clock pulse the status register will be loaded with the status of the addition and the second microinstruction will be distributed to the APU. During this microinstruction the carry status will be selected (see Table 5.10) and sent to the CCU.

Example 2

Suppose that it is required to store the status bits of the ALU in register 2. The microcode is then as follows:-

CCS CONTROL										ALU CONTROL															
ST_1	ST_2	C_4	C_5	C_6	S_3	S_2	S_1	C_3		RAM A	RAM B	I_0	$\overline{E}_A$	$\overline{OE}_B$	I_4	I_3	I_2	I_1	I_8	I_7	I_6	I_5	$\overline{IEN}$	$\overline{OE}_y$	E_4	Pol
1	0	X	X	X	X	X	X	X		XXXX	0010	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0

5.6 Design of the Carry in Control Logic

The carry in control logic controls the carry input (C_n) of the arithmetic and logic unit. During some arithmetic or logical operations it is desirable to set or reset the carry input of the Am2903s (see Table 5.2). The carry input can be directly controlled from the CCU for simple operations such as add, increment by one, decrement by one, subtract,

2's complement etc. For more sophisticated carry dependent operations (such as 'add the carry status to the result', add the most significant bit of the Q register to a RAM register' etc.) it is necessary to derive the C_n from one of several sources.

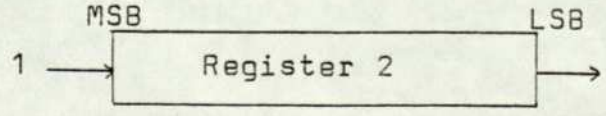
The carry in logic was designed by using a 4 bit multiplexer ($\frac{1}{2}$Am74LS153) whose inputs were derived from the condition multiplexer and the status multiplexer of the CCS, thus enabling the C_n to be derived from a considerably large number of sources.

The carry in control logic is controlled by the control bits S_1, S_2, S_3, C_3, C_4, C_5, C_6 and C_8 as shown in Fig. 5.13 and sources for the C_n input of the ALU are tabulated in Table 5.11.

5.7 Design of the Data Shifting Logic (DSL)

This is the final subsystem of the arithmetic processing unit. The functions of the DSL are:-

i. Shift data to either ends of the ALU registers. e.g. shift a 1 to the most significant bit position of register 2.



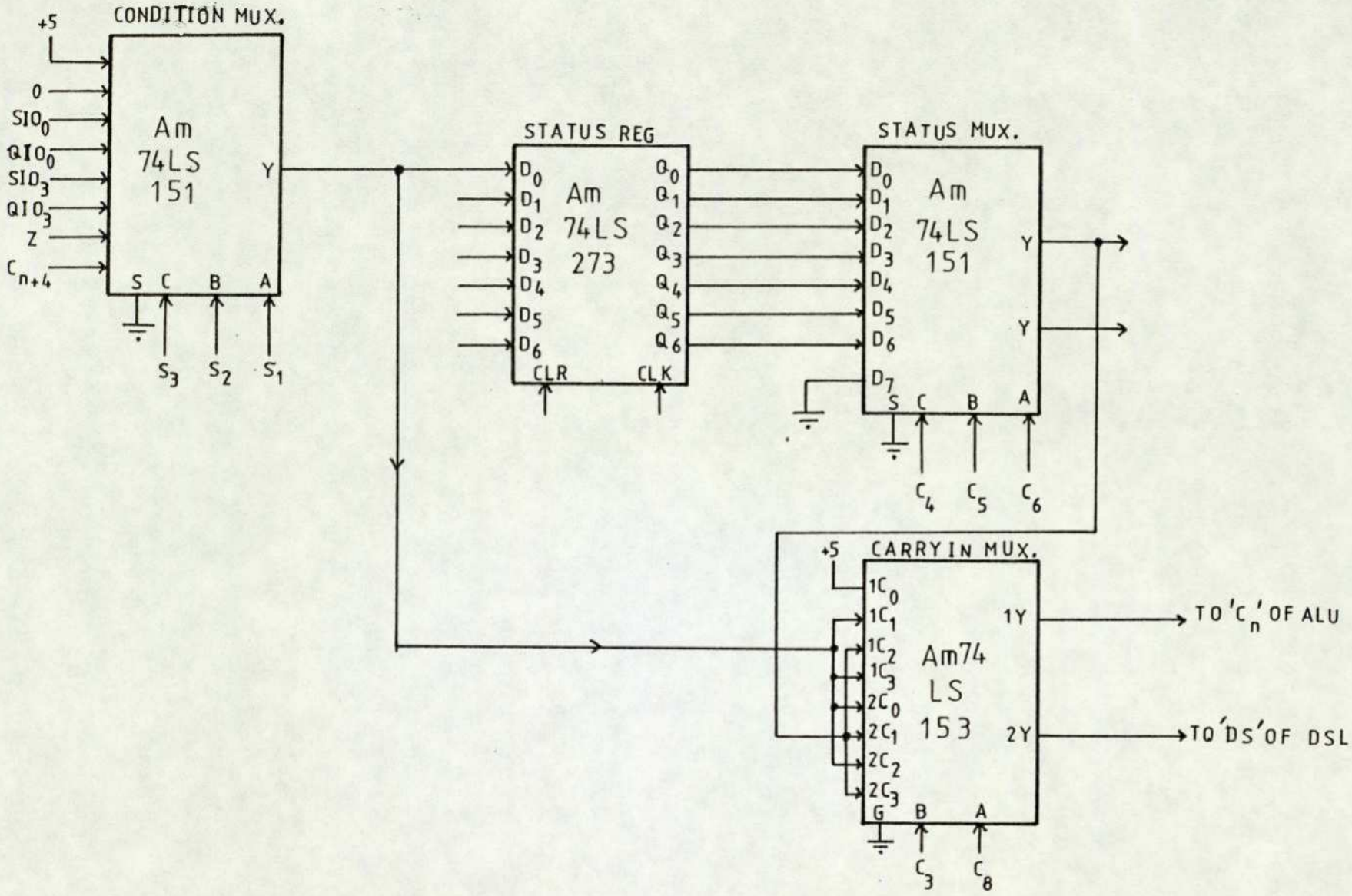


Figure 5.13 Controlling the Carry-in (C_n) of the ALU and the DS of the DSL

CONTROL					ALU Carry-in
C_8	C_3	S_3	S_2	S_1	C_n (not latched)
0	0	X	X	X	1
1	X	0	0	0	1
1	X	0	0	1	0
1	X	0	1	0	SIO_0 see note 1
1	X	0	1	1	QIO_0 see note 1
1	X	1	0	0	SIO_3 see note 1
1	X	1	0	1	QIO_3 see note 1
1	X	1	1	0	Z (zero status)
1	X	1	1	1	C_{n+4} (carry status)
0	1	X	X	X	*

* C_n is selected according to the contents of the status register (see Table 5.9) and the following Table

CONTROL			ALU Carry-in
C_4	C_5	C_6	C_n (latched)
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	0 see note 2

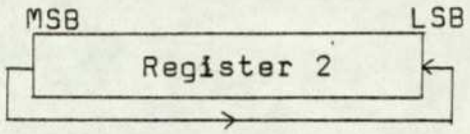
D_{0-6} : Contents of the status register (see Table 5.9)

note1: The appropriate shift operations must be performed by the ALU to shift LSB or MSB of RAM or the Q register out of the shift lines prior to sending to C_0 .

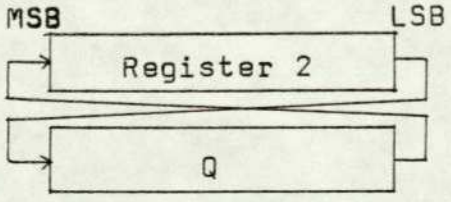
note2: This condition is not selected from the status register .

Table 5.11 Controlling the Carry-in (C_n) input of the ALU .

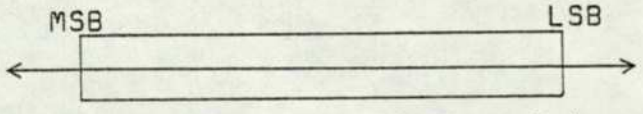
ii. Rotate the data (left or right) stored in the ALU registers. e.g. rotate the contents of register 2 by four positions to the left.



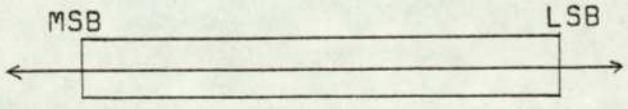
iii. Double shift or double rotate the data stored in an ALU register and the Q register. e.g. double rotate register 2 and the Q register by one position to the right.



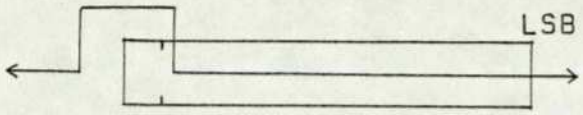
Shifting (or rotating) can either be in logical or in arithmetic mode. In logical shifting, the sign bit (most significant bit of the most significant Am2903) is also shifted left or right, where in the arithmetic shifting mode, the sign bit is preserved and the other bits are shifted. The Q register is always shifted in logical mode.



Logical shifting of Q register



Logical shifting of RAM



Arithmetic shifting of RAM

There is the possibility of shifting (or rotating) either the RAM and the Q register simultaneously or shifting either of them separately. Referrring to Table 5.3, the following tables can be deduced which explain the mode and the direction of shifting internal to the ALU.

ALU instruction bits I_7	I_6	I_5	ALU register shifted
0	0	0	Only RAM
0	1	X	Ram and Q
1	0	1	Only Q

ALU instruction bits I_8	I_5	Direction of Shifting
0	0	Right, arithmetic
0	1	Right, logical
1	0	Left, arithmetic
1	1	Left, logical

During a shift left operation, the ALU shift line SIO_0 (or QIO_0) is generally a serial shift input and SIO_3 (or QIO_3) is a serial shift output. Similarly, during a shift right operation, the ALU shift line SIO_3 (or QIO_3) is generally a serial shift input and SIO_0 (or QIO_0) a serial shift output.

The data shifting logic designed here uses two dual 4 line to 1 line data multiplexers (Am74LS253). Under the control of the two control bits S_4 and S_5, these multiplexers can perform four different shift operations as shown in the following table:-

DSL Control Bits S_4	S_5	Shift operation performed
0	0	Shift
1	0	Rotate
0	1	Double rotate
1	1	Double shift

The logic diagram of the DSL is shown in Fig. 5.14. The ALU instruction bit I_8 was used to control the direction of shifting such that during a right shift operation (I_8 = LOW) the output of the multiplexer M_2 is forced to be three-state while the multiplexer M_1 is enabled (1G, 2G = LOW). Similarly, during a left shift operation,

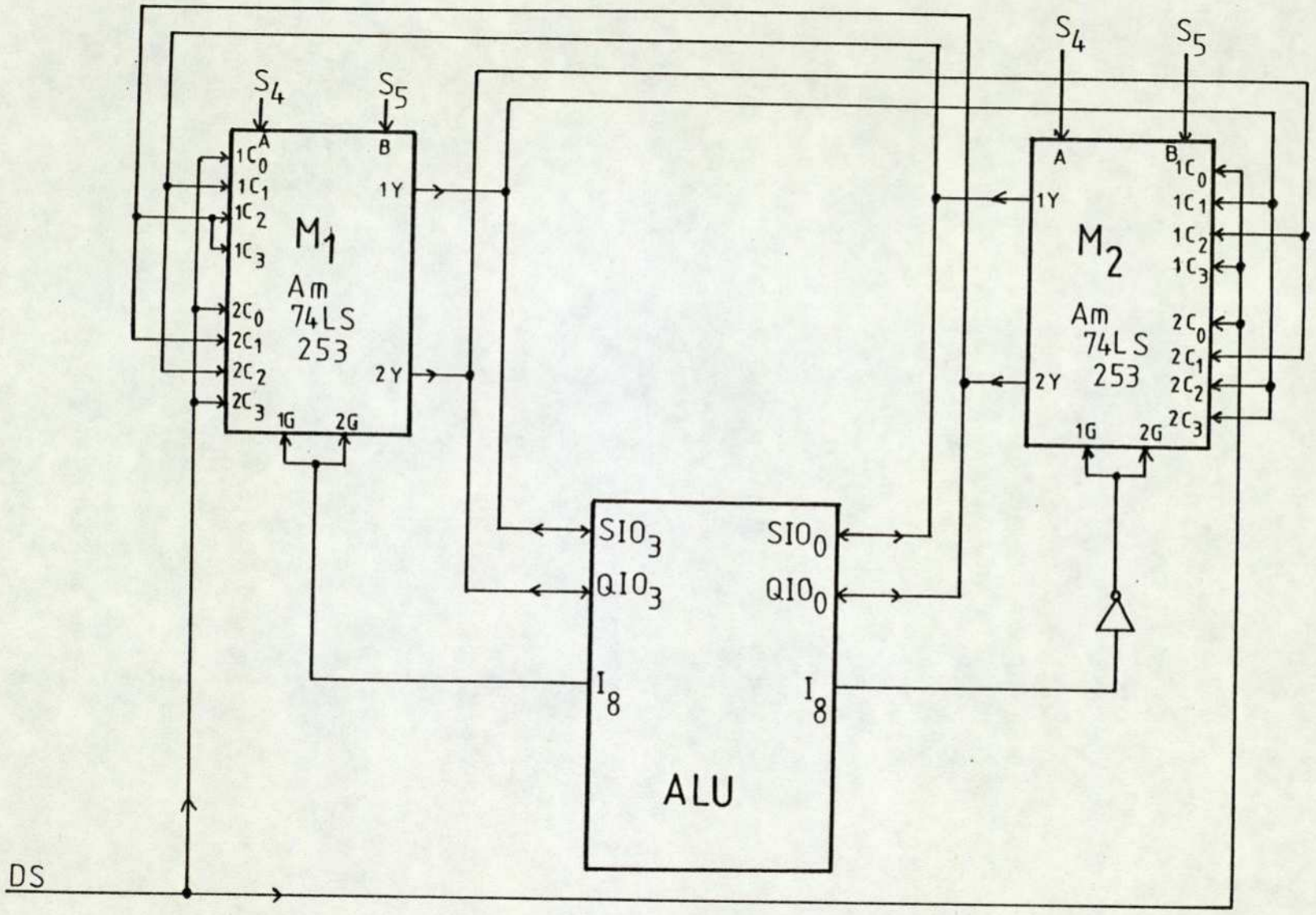


Figure 5.14 Logic diagram of the Data Shifting Logic (DSL) .

the multiplexer M_1 is forced to be three-state while M_2 is enabled.

The data to be shifted to the ALU was derived from the '2Y' output of the carry in multiplexer (see Fig. 5.13) and is tabulated in Table 5.12.

Tables 5.13 to 5.16 list the possible shift (or rotate) operations that can be performed by the data shifting logic.

5.7.1 Examples on Microcoding the DSL

During a data shifting (or rotating) instruction, the arithmetic and logic unit should perform one of the following operations:-

For the shifting of RAM:

$$R_B .\ \text{OR}.\ 0 \xrightarrow{\text{shift}} R_B$$

$$\text{or, } R_B .\ \text{AND}.\ 1 \xrightarrow{\text{shift}} R_B$$

$$\text{or, } R_B + 0 \xrightarrow{\text{shift}} R_B$$

(Where R_B is the register addressed by the 'B' field of the ALU) and for the shifting of the Q register:-

$$Q .\ \text{OR}.\ 0 \xrightarrow{\text{shift}} Q$$

$$\text{or, } Q .\ \text{AND}.\ 1 \xrightarrow{\text{shift}} Q$$

$$\text{or, } Q + 0 \xrightarrow{\text{shift}} Q$$

CONTROL				Data to be Shifted
C_8	S_3	S_2	S_1	DS (not latched)
0	0	0	0	1
0	0	0	1	0
0	0	1	0	SIO_0 see note 1
0	0	1	1	QIO_0 see note 1
0	1	0	0	SIO_3 see note 1
0	1	0	1	QIO_3 see note 1
0	1	1	0	Z (zero status)
0	1	1	1	C_{n+4} (carry status)
1	X	X	X	*

* DS is selected according to the contents of the status register (see Table 5.9) and the following Table .

CONTROL			Data to be Shifted
C_4	C_5	C_6	DS (latched)
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	0 see note 2

D_{0-6} : Contents of the status register (see Table 5.9)

note 1: The appropriate shift operations must be performed by the ALU prior to using these signals.

note 2: This condition is not selected from the status register.

Table 5.12 Selecting the data (DS) to be shifted to the ALU

If either of the first two logical operations are performed (OR, AND) then the status bits C_{n+4} and OVR will be reset (see Table 5.5) at the end of the shift. If on the other hand the arithmetic operation (add) is performed, all the four status bits of the ALU will change to reflect the result of the shifting.

The shift operation to be performed (left, right, logical or arithmetic) is determined from Tables 5.13 to 5.16 and the data to be shifted (DS) is determined from Table 5.12.

Table 5.17 shows examples of microcoding four different shift operations. The first microinstruction shifts 1 to the most significant bit position of register 3. The second microinstruction rotates (logical) the contents of register 3 and the Q register left by one bit. The third microinstruction double rotates (arithmetic) the contents of register 4 and the Q register, right by one bit. Finally, the last microinstruction double shifts (logical) register 5 and the Q register, left one bit, while shifting a '0' to the least significant bit position of the Q register. The status register is disabled in all of the four examples.

5.8 Conclusions

Chapter 5 has described the design of the arithmetic

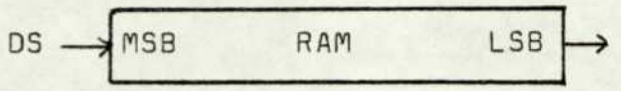
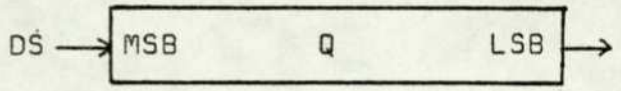
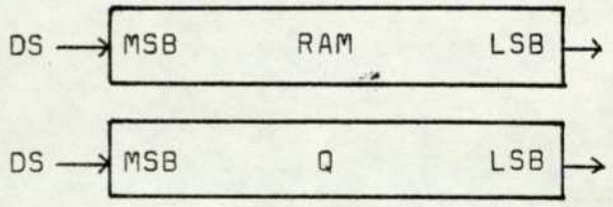
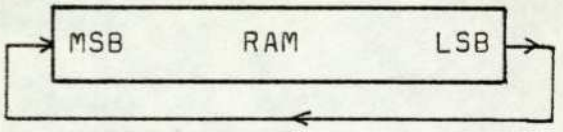
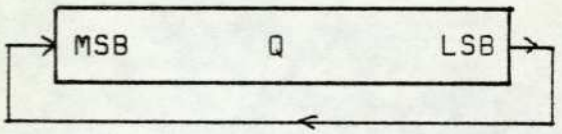
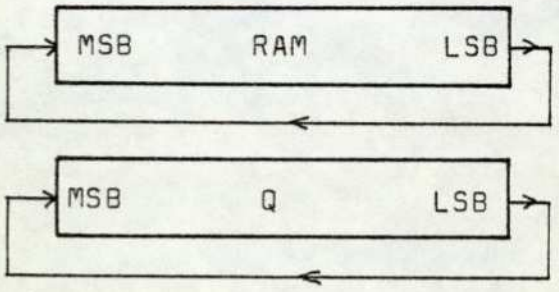
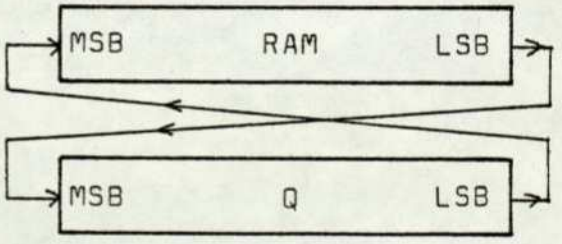
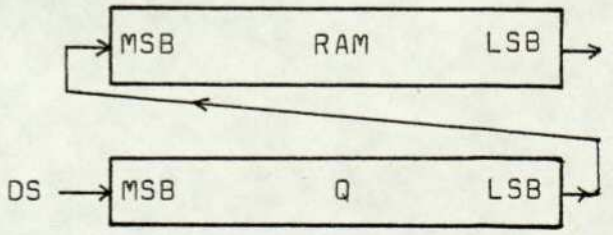
DSL Control S_5 S_4	ALU Control I_8 I_7 I_6 I_5	SHIFT OPERATION (see Table 5.12 for the selection of DS)
0 0	0 0 0 1	
0 0	0 1 0 1	
0 0	0 0 1 1	
0 1	0 0 0 1	
0 1	0 1 0 1	
0 1	0 0 1 1	
1 0	0 0 1 1	
1 1	0 0 1 1	

Table 5.13 Logical right shifting

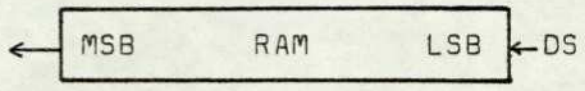
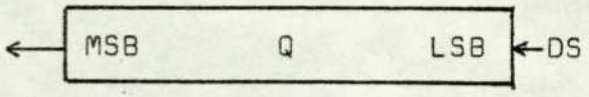
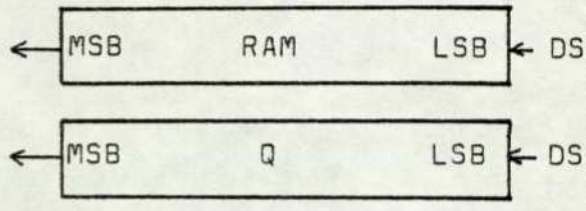
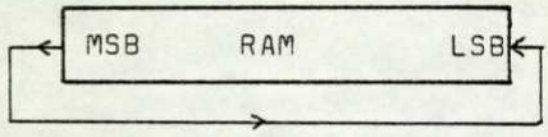
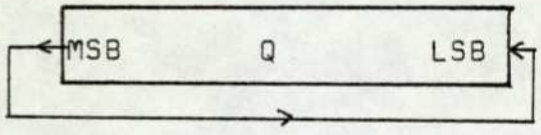
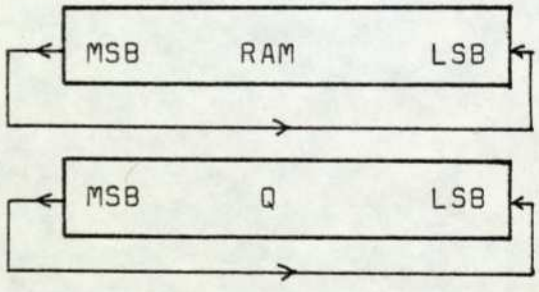
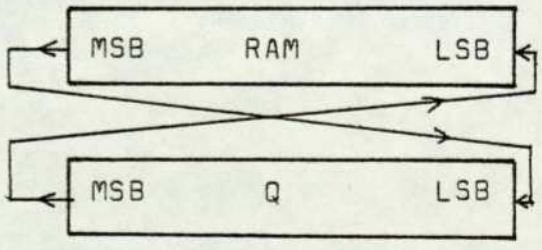
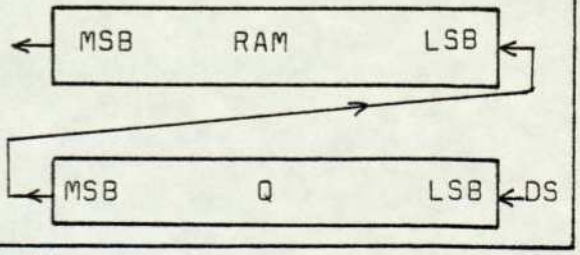
DSL Control S_5 S_4	ALU Control I_8 I_7 I_6 I_5	SHIFT OPERATION (see Table 5.12 for the selection of DS)
0 0	1 0 0 1	
0 0	1 1 0 1	
0 0	1 0 1 1	
0 1	1 0 0 1	
0 1	1 1 0 1	
0 1	1 0 1 1	
1 0	1 0 1 1	
1 1	1 0 1 1	

Table 5.14 Logical left shifting

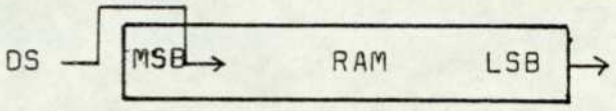
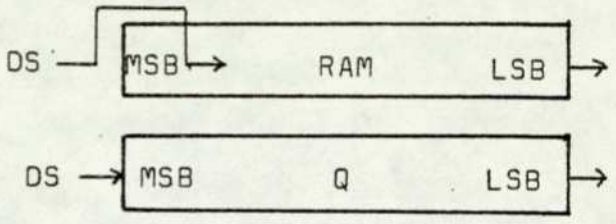
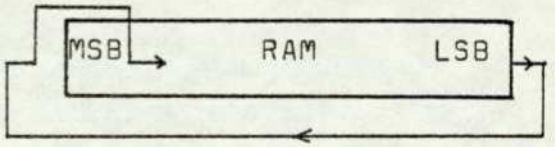
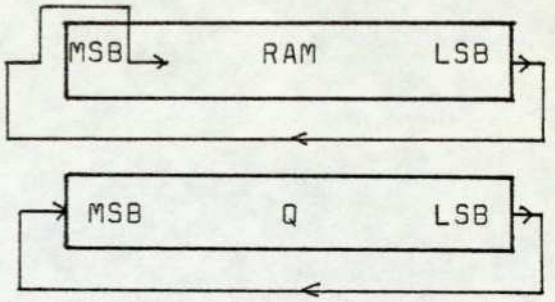
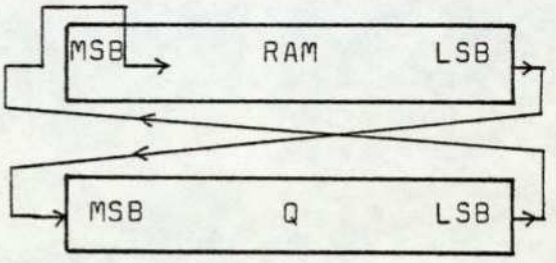
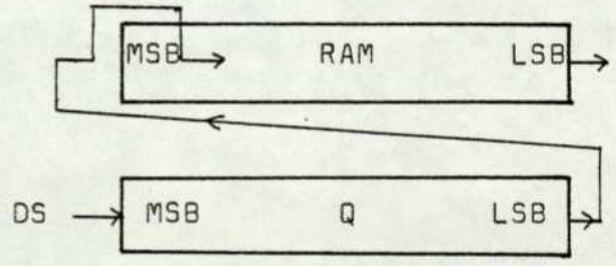
DSL Control S_5 S_4	ALU Control I_8 I_7 I_6 I_5	SHIFT OPERATION (see Table 5.12 for the selection of DS)
0 0	0 0 0 0	
0 0	0 0 1 0	
0 1	0 0 0 0	
0 1	0 0 1 0	
1 0	0 0 1 0	
1 1	0 0 1 0	

Table 5.15 Arithmetic right shifting

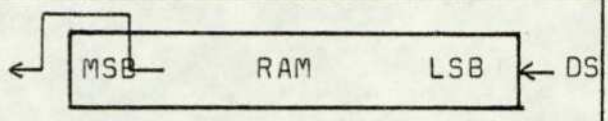
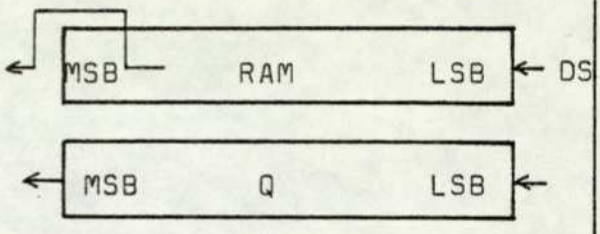
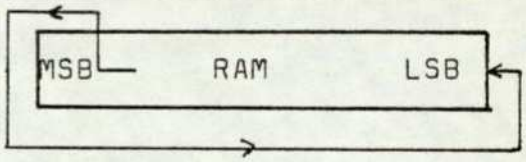
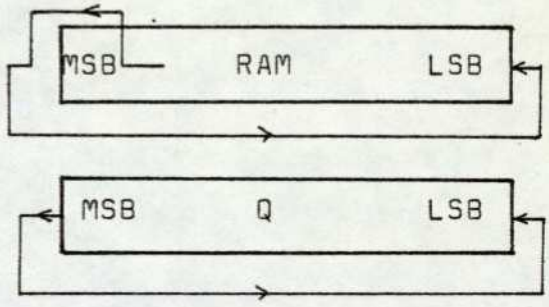
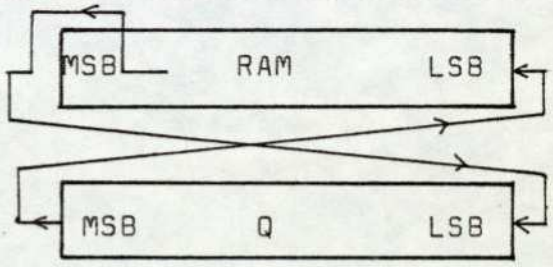
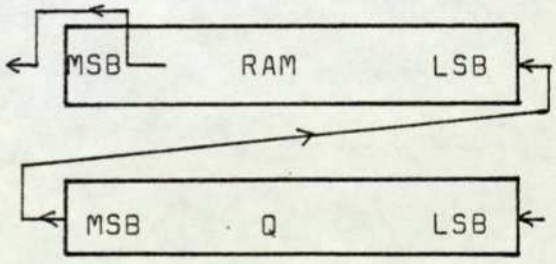
DSL Control S_5 S_4	ALU Control I_8 I_7 I_6 I_5	SHIFT OPERATION (see Table 5.12 for the selection of DS)
0 0	1 0 0 0	
0 0	1 0 1 0	
0 1	1 0 0 0	
0 1	1 0 1 0	
1 0	1 0 1 0	
1 1	1 0 1 0	

Table 5.16 Arithmetic left shifting .

CCS, DSL, BML CONTROL	ALU CONTROL					
	RAM		Source	Function	Dest.	Other
$S_5 S_4 ST_1 ST_2 C_4 C_5 C_6 C_8 S_3 S_2 S_1 C_3 E_4 Pol$	A	B	$I_0 \overline{E}_A \overline{OE}_B$	$I_4 I_3 I_2 I_1$	$I_8 I_7 I_6 I_5$	$\overline{IEN}$ $\overline{OE}_Y$
0 0 1 1 X X X 0 0 0 0 X 0 0	XXXX	0011	0 1 0	1 1 1 1	0 0 0 1	0 0
0 1 1 1 X X X X X X X X 0 0	XXXX	0011	0 1 0	1 1 1 1	1 0 1 1	0 0
1 0 1 1 X X X X X X X X 0 0	XXXX	0100	0 1 0	1 1 1 1	0 0 1 0	0 0
1 1 1 1 X X X 0 0 0 1 X 0 0	XXXX	0101	0 1 0	1 1 1 1	1 0 1 1	0 0

Table 5.17 Examples of microcoding shift operations .

processing unit of the microcomputer. A total of 39 lines are required to control the APU.

The next chapter of this thesis will describe the design of the input-output subsystem of the microcomputer.

CHAPTER 6

6. DESIGN OF THE INPUT-OUTPUT UNIT

6.1 Introduction

The Input-Output (I-O) Unit provides the means for a computer to interact with the external world. Information may be transferred between a computer and an external logic in either parallel or serial form. In parallel I/O, all bits of a data are transferred at the same time, while in serial I/O, the bits of a data are transferred one by one with an appropriate time delay between each bit. The serial communication techniques are widely used to interface a user terminal (TTY, VDU etc.)to a computer. The data transfer between a computer and an external device can either be via bidirectional I/O port lines or separate input and output port lines can be used to receive and send data.

There are three basic modes of I/O operation:-

i. Programmed I/O: In this case, all data transfers between a computer and the external logic are completely controlled by the program being executed by the computer.

ii. Interrupt-driven I/O: Here, the computer interacts with its environment only when a hardware interrupt

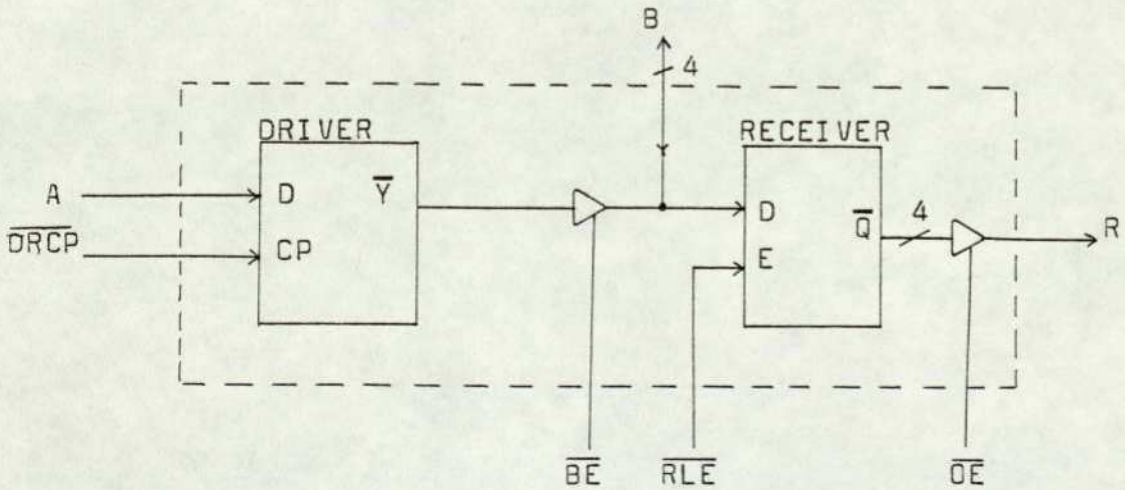
is sent to the computer from an external logic. The computer then suspends whatever it is currently doing in order to service the needs of the external logic (see Chapter 9 for the design of the Interrupt Control Unit, ICU).

iii. Direct memory access: In this kind of data transfer, data can be moved between the computer memory and the external devices without the participation of the Central Processing Unit.

The input-output unit of the microcomputer designed utilizes the programmed I/O concept and separate I/O port lines were used to communicate with the external devices.

6.2 The Design of the I/O Unit

The I/O Unit was designed by using Am2917 type Bipolar 4 bit latched bus transceivers. The simplified logic diagram of the Am2917 is as follows:-



When $\overline{BE}$ is LOW, the data at A will be inverted and transferred to B on the LOW to HIGH transition of the clock input $\overline{DRCP}$. When $\overline{BE}$ is HIGH, the outputs of the driver are forced to a high impedence state.

When $\overline{RLE}$ is LOW, the data at B will be inverted and transferred to R if $\overline{OE}$ is LOW. If on the other hand $\overline{RLE}$ is HIGH, then loading of the receiver will be disabled and it will retain its previous data.

The block diagram of the I/O Unit is shown in Fig. 6.2. Four Am2917s are required for a 16 bit I/O port. The 'A' inputs of the Am2917s were used as the input port lines while the 'R' outputs were used as the output port lines. The bidirectional 'B' pins of the Am2917s were connected to the system Data Bus. During an input operation, the data at the 'A' inputs are routed to the Data Bus. Similarly, during an output operation, the data at the Data Bus is sent to the 'R' outputs of the Am2917s.

The two control bits IOR (I/O request) and the IOM (I/O mode) control the I/O Unit as follows:-

IOR	IOM	I/O Operation
0	0	Output request (Data Bus to 'R')
0	1	Input request ('A' to Data Bus)
1	X	No I/O operation

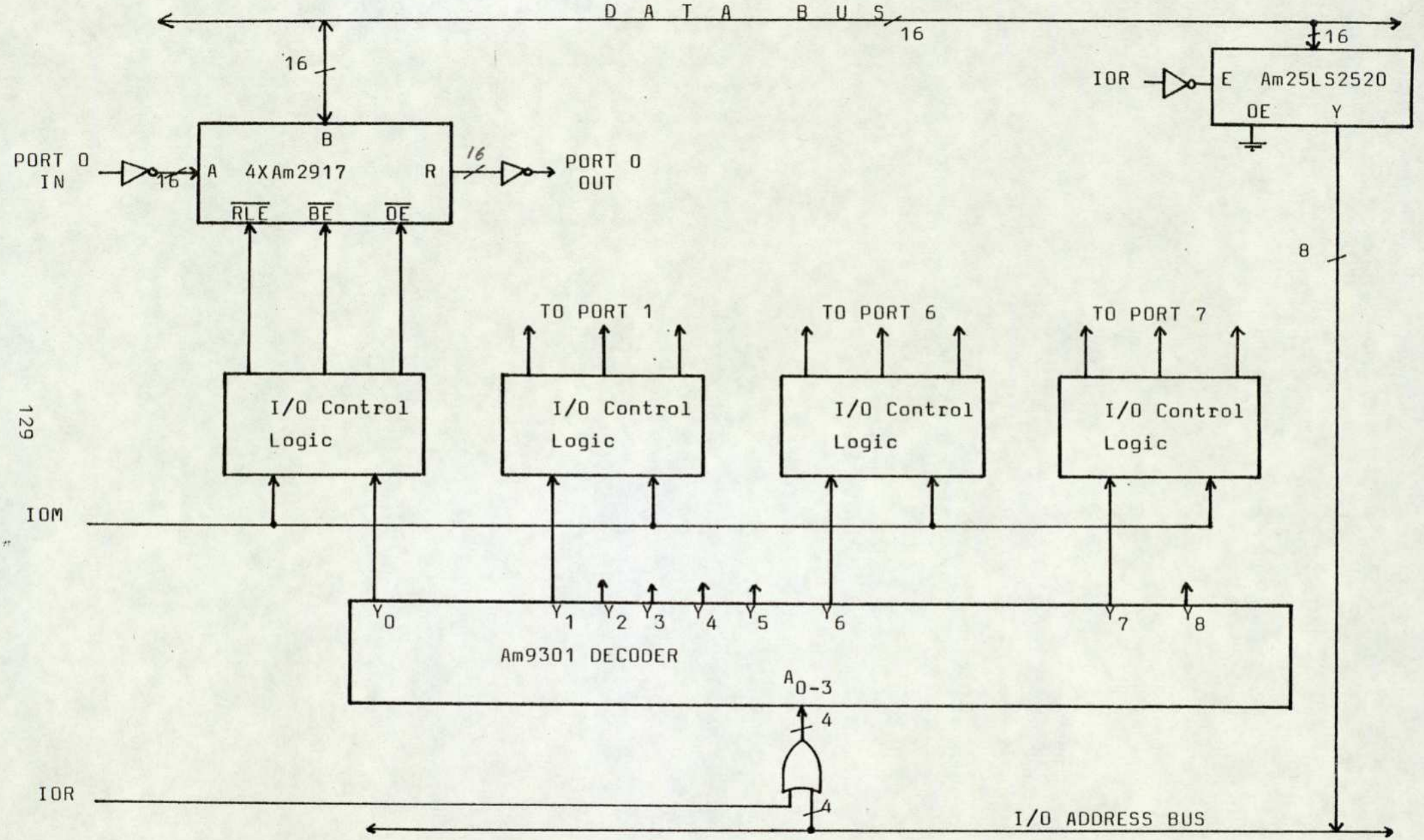


Figure 6.2 Block diagram of the I/O Unit (only PORT 0 shown) .

The I/O port addresses were derived from the higher byte of the Data Bus (this was because during 8 bits of operation only the higher byte of the Data Bus is available) via a clocked register (Am74LS2520) which is updated after every clock pulse whenever IOR is HIGH. The port addresses were decoded by a 1 to 10 decoder (Am9301) such that only one output (Y) is LOW whenever an I/O operation is requested (IOR = LOW), otherwise all outputs of the decoder are forced to be HIGH (no I/O request). The truth table of the Am9301 is shown in Table 6.1.

The Am2917s were controlled from the I/O control logic as described by the following truth table (Y is any output of the decoder):

Y	IOM	$\overline{BE}$	$\overline{RLE}$	$\overline{OE}$	
0	0	1	0	0	output request
0	1	0	1	0	input request
1	0	1	1	0	no I/O request
1	1	1	1	0	

The Am2917 control signals were therefore derived from the following Boolean operations:-

$$\overline{BE} = Y + \overline{IOM}$$

$$\overline{RLE} = Y + IOM$$

and, $\overline{OE} = 0$

A_3	A_2	A_1	A_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0
1	0	1	0	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 6.1 Truth table of the Am9301 Decoder .

6.3 The Number of I/O Ports

Although by cascading several decoders it is possible to obtain a maximum of 256 I/O ports, only four ports were connected to the microcomputer to be used for the following purposes:-

i. PORT 0: This is a general purpose 16 bit I/O port that can be used to input or output data in parallel form. The output lines of PORT 0 were connected to sockets and Light Emitting Diodes (LEDs) mounted on the front panel of the microcomputer.
The logic diagram of PORT 0 is shown in Fig. 6.3.

ii. PORT 1: This is a 12 bit I/O port which was used as a data acquisition channel. The input of PORT 1 was connected to an analog to digital converter (A/D). Similarly, the output of PORT 1 was connected to a digital to analog converter (D/A).

iii. PORT 6 and PORT 7: These two ports were used to control a UART (Universal Asynchronans Receiver Transmitter) so that a wide range of terminals could be connected to the microcomputer.

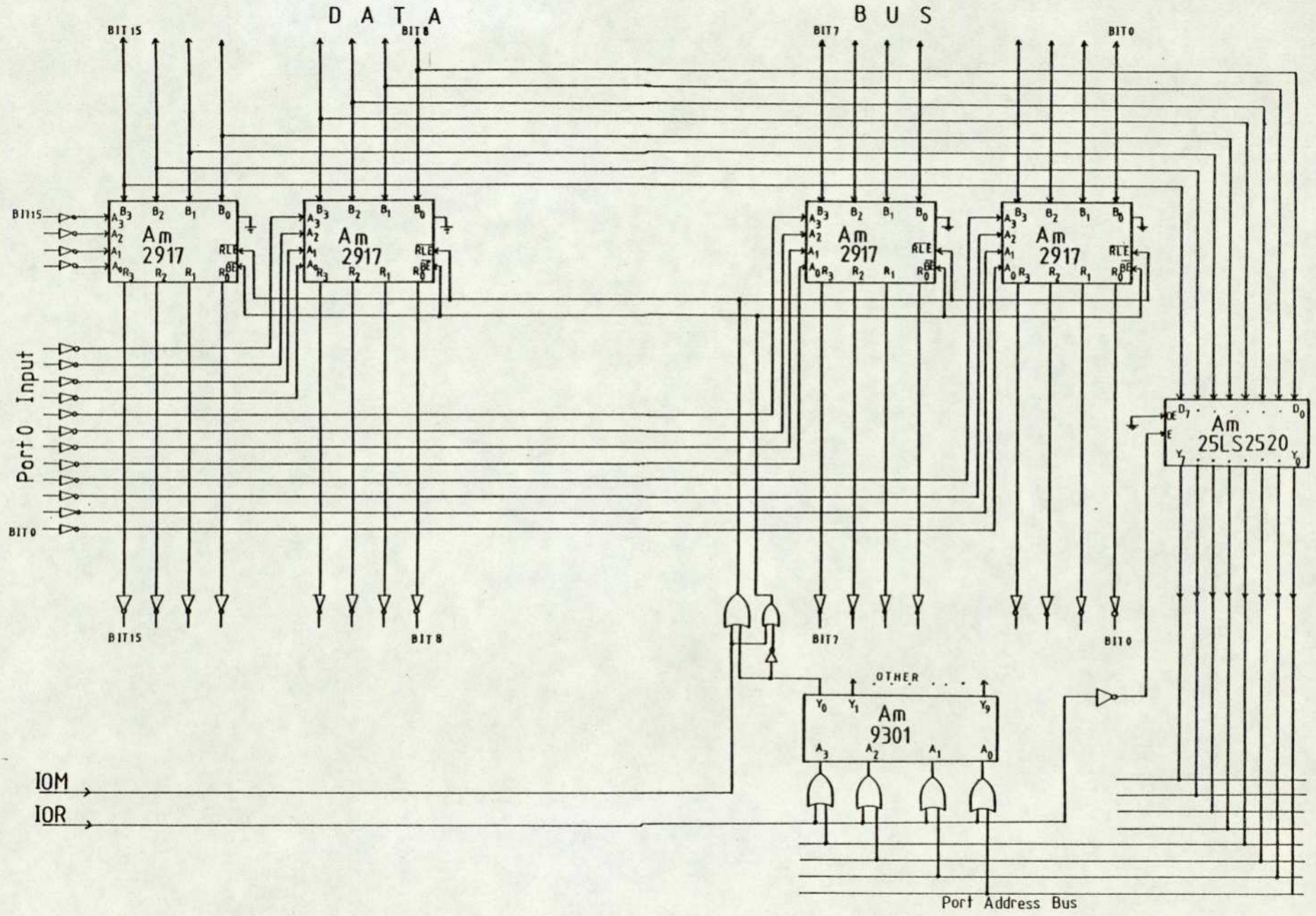


Figure 6.3 Logic diagram of PORT 0

6.4 Design of the Data Acquisition Channel (Port 1)

The block diagram of the data acquisition channel is shown in Fig. 6.4. The functions and design of different elements of the data acquisition channel are described below:-

6.4.1 Sample/Hold Amplifier

In many real time applications, the analog signals vary quite rapidly. Because an A/D converter takes a finite time to digitize an input signal, large changes in the signal level during the actual conversion time could give errors in the digitized output. Sample/Hold amplifiers are usually introduced into the data path before the A/D converter and they 'hold' the analog signal at a constant level for the duration of the A/D conversion process.

The following example indicates the improvements possible with a sample/hold amplifier: if the input signal, E, is a full-scale sine wave, $E_{FS} \sin 2\pi ft$, then the rate-of-change of this signal is

$$\frac{dE}{dt} = 2\pi f E_{FS} \cos 2\pi ft$$

It then follows that the maximum rate-of-change occurs at $2\pi ft = n\pi$ and equals to $2\pi f E_{FS}$. If the accuracy of the conversion is to be ΔE volts within the conversion time

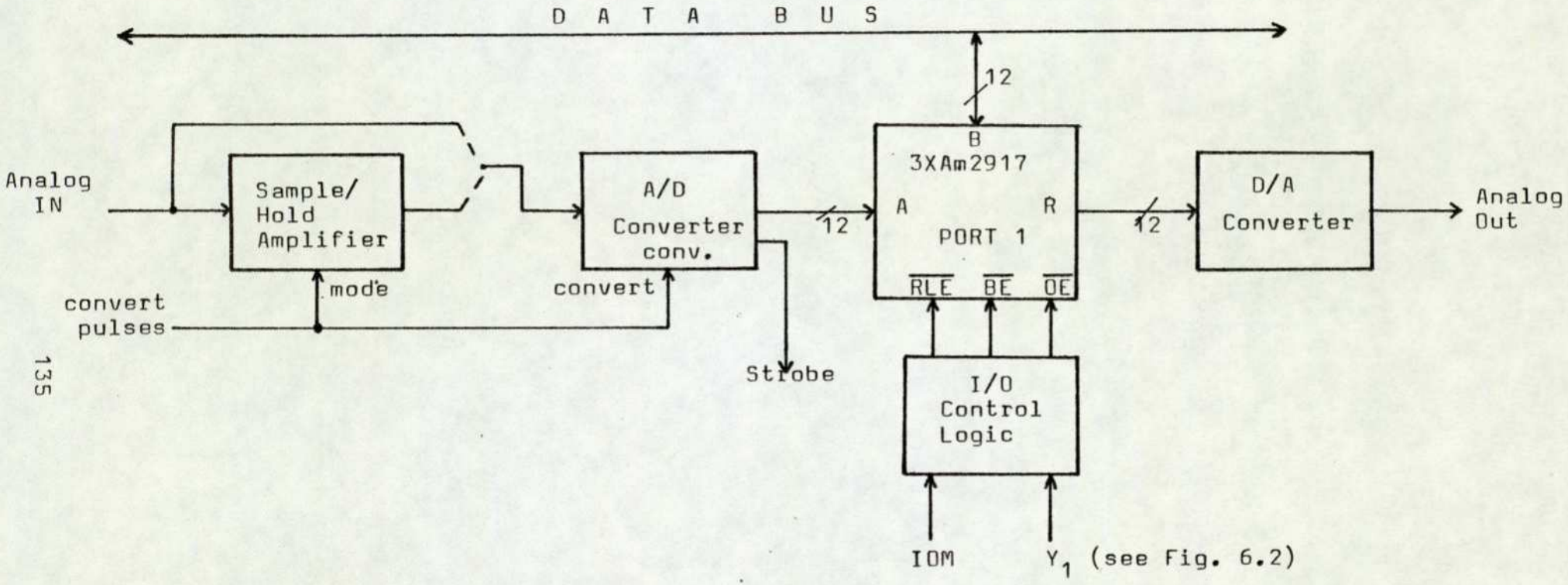


Figure 6.4 Block diagram of the data acquisition channel (PORT 1) .

Δt_c, then the highest frequency input signal that can be applied without a sample/hold can be calculated from the following formula:-

$$f_{max} = \frac{\Delta E}{E_{FS}} \cdot \frac{1}{2\pi \Delta t_c}$$

As an example, consider a data acquisition channel where the conversion time of a 12 bit A/D converter is 20 μs and the required conversion accuracy is $\pm \frac{1}{2}$ LSB. Then the maximum input frequency should be:-

$$f_{max} = \frac{1}{2^{12}} \cdot \frac{1}{2\pi \times 20 \times 10^{-6}} \simeq 2 \text{ Hz}$$

This example shows that any full-scale input signal with a frequency greater than 2 Hz cannot be digitized with an accuracy of $\pm \frac{1}{2}$ LSB.

Now consider using a sample/hold amplifier. The maximum frequency input signal can now be calculated from the following formula:-

$$f_{max} = \frac{\Delta E}{E_{FS}} \quad \frac{1}{2\pi \Delta t_a}$$

Where Δt_a is the aperture-time uncertainty (range of variation in the time required after the 'hold' command to actually 'hold' the input signal) of the sample/hold amplifier, which is typically 5 ns. Thus, an improvement in f_{max} of 4000:1 (Δtc/ Δta) is quite feasible by using a sample/hold amplifier.

The sample/hold amplifier used in the design is the SHC85, manufactured by Burr-Brown [9]. This particular device was mainly chosen because of its high speed. The device is controlled from its 'mode' input such that when 'mode' is HIGH a 'sample' operation is performed and when 'mode' is LOW, the device is in the 'hold' mode (see Fig. 6.5).

6.4.2 Analog to Digital Converter

The analog to digital converter used in the microcomputer designed is the ADC85C-12*, manufactured by Burr-Brown [9]. This is a 12 bit converter with a conversion time of 10 μs. The conversion is initiated by the trailing edge of the pulses applied to the 'convert command' input. The end of conversion is indicated by the HIGH-to-LOW transition of the 'strobe' output (see Fig. 6.6).

The ADC85C-12 was connected such that it can either be operated in the 'Bipolar' mode ($\pm$ voltage levels) or in the 'Unipolar' mode (only positive voltage levels) selected by a switch, (mode select switch) mounted on the front panel of the microcomputer. The full-scale analog input voltage range can be selected from another switch (range select switch) also mounted on the front panel of the microcomputer. The Bipolar (BPLR) operation provides Two's complement data to the microcomputer while the Unipolar (UPLR) operation provides Straight binary data. The mode select and the

* This device was chosen because of its fast ~~conversion~~ time and the wide range of input voltage levels.

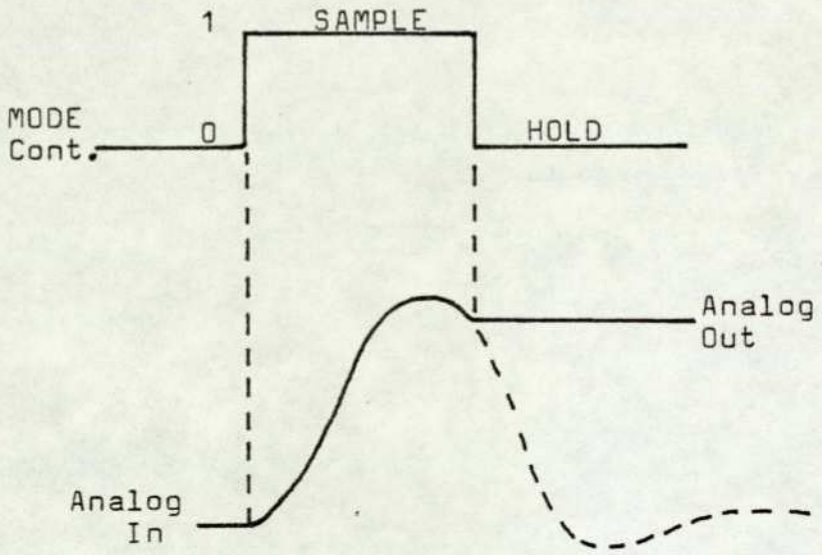
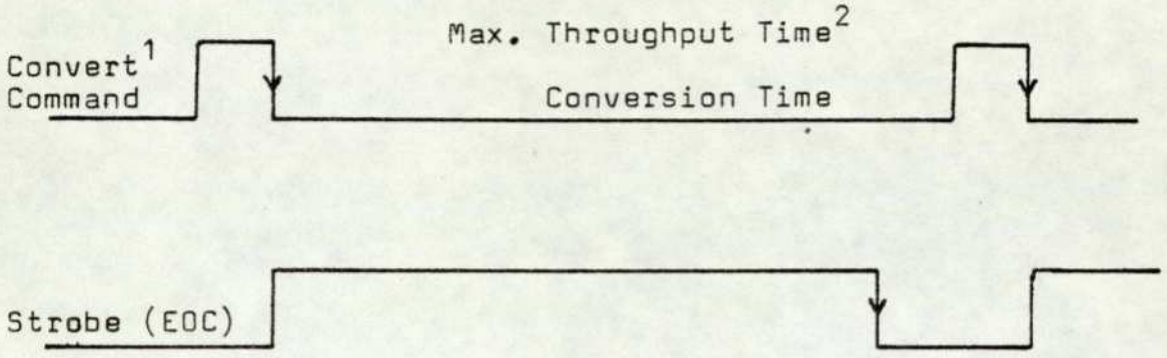


Figure 6.5 The 'mode' input of the SHC85 controls the 'hold' and 'sample' operations .



Notes:-

i. The Convert Command must be at least 50 ns wide and must remain LOW during a conversion. The conversion is initiated by the 'trailing edge' of the Convert Pulses .

ii. 10.5 μs for 12 bits of conversion .

Figure 6.6 Timing diagram of the A/D Converter .

range select switches were connected such that they can select the mode and the full-scale range as follows:-

Mode Select Switch	Range Select Switch	Operation Code
BPLR	± 10Vor ± 5Vor ± 2.5 V	Two's Complement
UPLR	0 to +5 Volts	Straight binary

The Two's Complement and the Straight Binary Codes are defined as follows:-

	Two's Complement	Straight Binary
+ Full Scale	011111111111	111111111111
½ Full Scale	010000000000	100000000000
1 LSB	000000000001	000000000001
Zero	000000000000	000000000000
-1 LSB	111111111111	Not defined
-½ Full Scale	110000000000	Not defined
- Full Scale	100000000000	Not defined

The voltage levels for the ranges of ± 10V Bipolar and 0 to 10 V Unipolar operation are defined as in the following table (for ± 5V or 0 to 5V range divide the values shown by 2. For ± 2.5 V range, divide the values shown by 4).

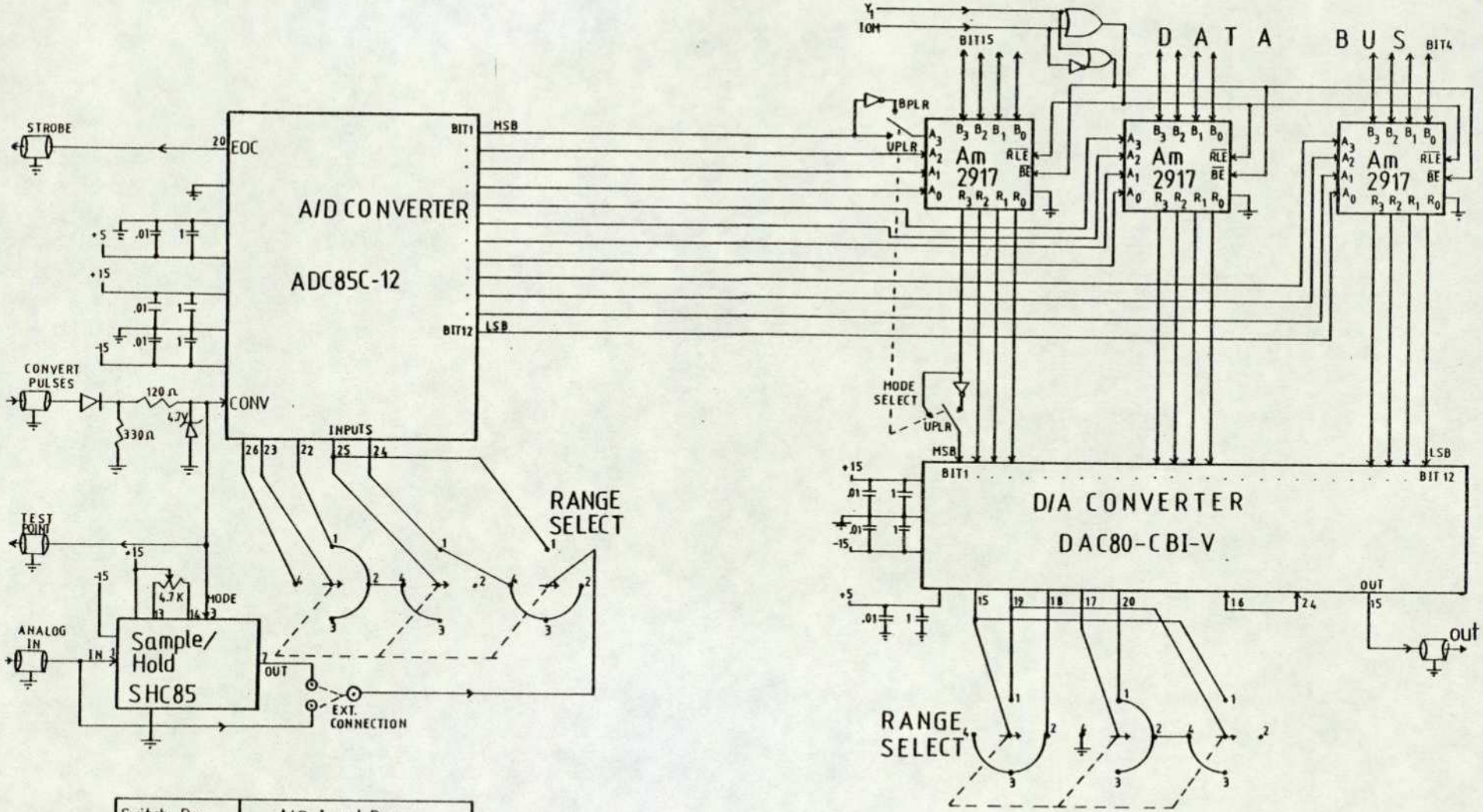
	Two's Complement	Straight Binary
+Full Scale	9.9951V	9.9976V
½ Full Scale	5.0000V	5.0000V
1 LSB	4.88mV	2.44mV
-1 LSB	-4.88mV	Not defined
-½ Full Scale	-5.0000V	Not defined
- Full Scale	-10.0000V	Not defined

6.4.3 Digital to Analog Converter

The digital to analog converter used in the data acquisition channel is the DAC80-C8I-V, manufactured by Burr-Brown [9]. This is a 12 bit converter with a conversion time of 3 μs*. The D/A converter was directly connected to the 12 output lines of PORT 1. The mode and the full-scale ranges of operation were selected by two switches as described in section 6.4.2. Also the same binary codes of operation and voltage levels apply to both D/A and the A/D converter.

The overall logic diagram of the data acquisition channel is shown in Fig. 6.7. The amplitude limiter at the input of the external convert pulses was designed to limit the pulses to +5 volts and hence protect the S/H amplifier and the A/D converter from overvoltage.

* The conversion time is 5 μs with ± 10 v full-scale.



Switch Pos.	A/D Input Range
1	± 10V Two's Comp.
2	± 5V Two's Comp.
3	± 2.5V Two's Comp.
4	0 to 5 Straight Binary

Switch Position	D/A Output
1	± 10V Two's Comp.
2	± 5V Two's Comp.
3	± 2.5V Twos Comp.
4	0 to 5V Straight

Figure 6.7 The Data Acquisition Channel

6.5 Design of the Serial Communication Channel (Port 6, Port 7)

6.5.1 Introduction

Before describing the design of the serial communication channel, it is worthwhile to review the asynchronous serial communication techniques commonly used by computers to communicate with terminals.

There are usually two serial data links between a computer and a terminal, one for transmission in each direction. The rate of transmission (bits per second) is called the baud rate. Generally the same baud rate is used for the transmission in either direction in a serial communication link. Transmission rates for typical computer terminals are between 110 baud and 9600 baud. A standard ASR-33 teletype transmits and receives at 110 baud rate. The visual display units (VDU) usually operate with a wide range of baud rates (typically 75 to 9600).

At any time a serial communication line may be transmitting one of two signals: either a logical 1 or a logical 0. However, the actual signals that are used to represent logical 1 or 0 vary in different serial communication systems depending upon the terminals used. If the computer uses TTL logic then to communicate with the standard teletypes, it is necessary to convert the TTL logic levels to 20 mA

current loop signals. In a 20 mA current loop interface, a pair of wires are used between the transmitter and receiver in which a current flow of 20 mA represents a logical 1 and the absence of current represents a logical 0.

Most terminals transmit and receive signals specified by the EIA Standard RS-232C or the CCITT Standard V.24. Terminals that use these standards for communication can be connected to each other directly (This is because here the logic levels are specified as voltage).

In a serial asynchronous communication interface, the data is transmitted and received serially as illustrated in Fig. 6.8, when no data is being sent, the transmitter is in the logical 1. state. The transmitter initiates transmission by sending a start bit, that is, by sending a logical 0 for the duration of one bit-time. It then tranmitts the data bits (least significant bit first) followed by an optional parity bit and 1, 1½ or 2 stop bits. A stop bit is always a logical 1. Although the transmitter may maintain the logical 1 state for a certain time before sending the next start bit, usually the start bit is sent immediately after the stop bit(s). The total bits between a start bit and the stop bits form a character. A typical teletype operating at 110 baud rate transmits a start bit, 7 data bits, an odd parity bit and two stop bits for each character, with a spacing of 9.09 mS between each bit. Since 11 bits are required to send each character, at 110 baud rate, there are 10 characters transmitted per second.

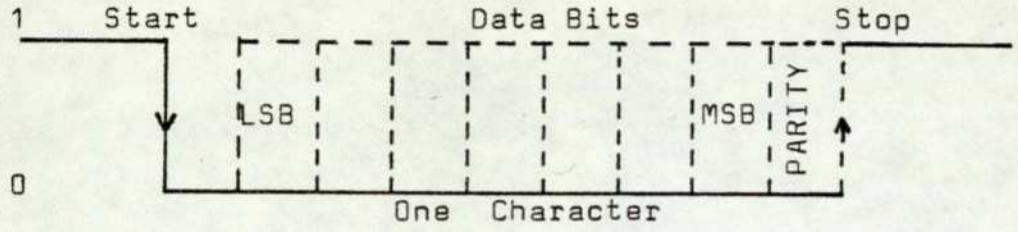


Figure 6.8 Data transmission in a serial asynchronous communication interface. (shown is an example of transmitting one start bit, 7 data bits, one parity bit, and two stop bits)

6.5.2 The Design

The serial communication channel was designed by using a IM6402 type UART (Universal Asynchronous Receiver Transmitter), manufactured by Intersil [10]. This is a CMOS device that converts parallel data to serial form with the appropriate start, stop and parity bits and presents it to the output at the required baud rate. Similarly, during an input operation, the UART converts the serial data to parallel form and presents it to the microcomputer.

The block diagram of the serial communication channel is shown in Fig. 6.9. The parallel input and output lines of the UART were connected to the higher byte of the Data Bus via buffers. The buffers were necessary to interface the UART to the TTL Data Bus.

Port 7 was used as a 2 bit input port and its inputs were connected to the TBRE and the DR status outputs of the UART. The definition of these status signals are:-

DR: A HIGH level at this pin indicates that a serial data has been received and converted to parallel form and is available at the parallel outputs (RBRI to RBR8).

TBRE: A HIGH level at this pin indicates that the UART is ready to accept a new parallel data from its inputs TBRI to TBR8.

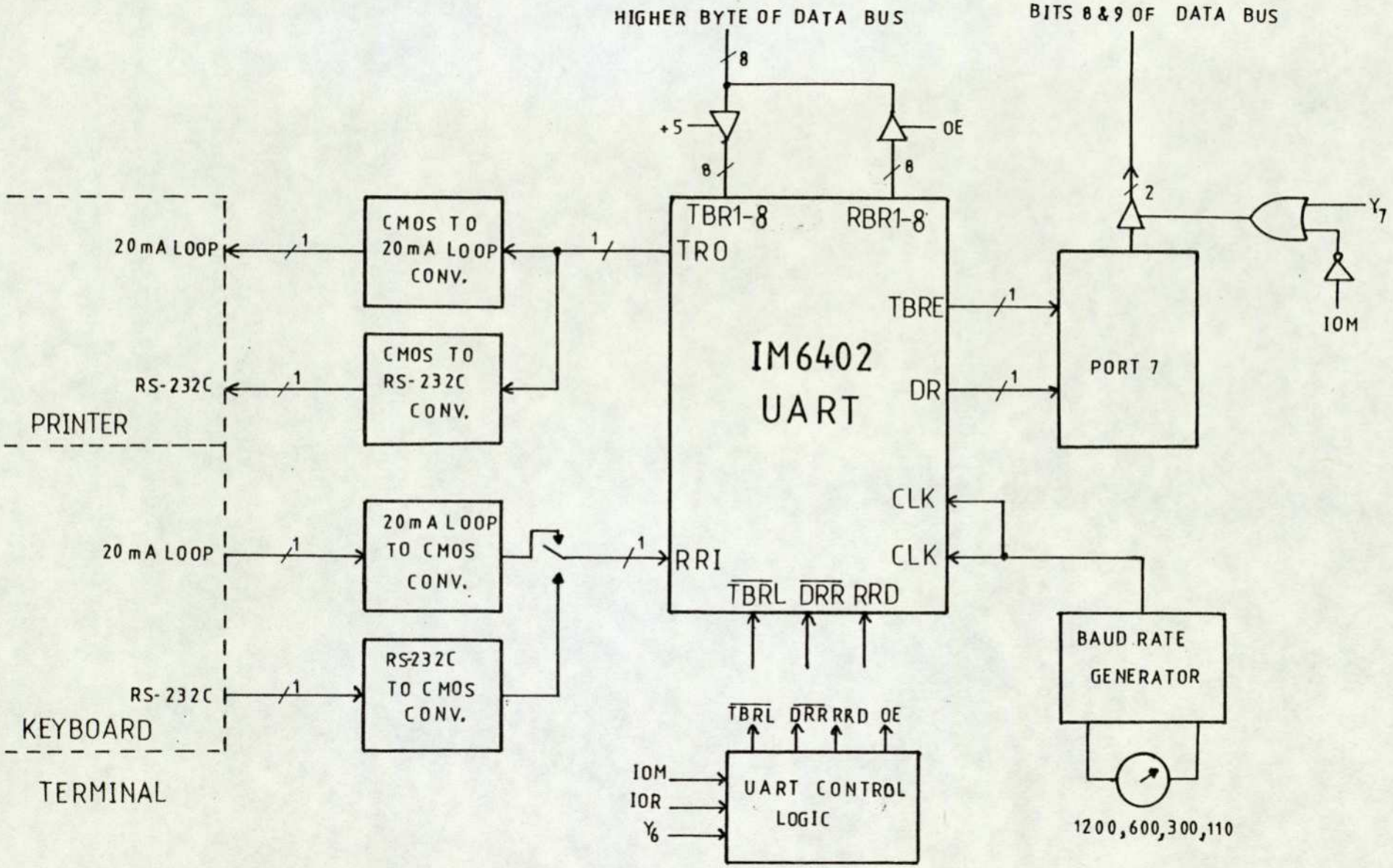


Figure 6.9 Block diagram of the serial communication channel

Before a parallel data is sent to the UART, it is necessary to ensure that TBRE is at logic HIGH. Similarly, before inputting a parallel data to the microcomputer, it is necessary to ensure that the status bit DR is at logic HIGH.

Since PORT 7 was used as a 2 bit input only port, it was designed from a relatively cheap dual 'D' type flip-flop (SN7474) instead of an AM2917. The two bit outputs of the flip-flop were connected to bit 8 and bit 9 of the Data Bus via the three-state buffers AM74LS241. The three-state buffers were controlled by the $\overline{BE}$ logic control as described in Chapter 6.2 ($\overline{BE} = Y_7 + \overline{IOM}$).

The UART control logic (see Fig. 6.9) was designed to control the UART such that it operates as the I/O Port 6. Thus, during an input via the I/O Port 6, the serial data at the input of the UART is converted to parallel form and is presented to the higher byte of the system Data Bus. During an output via the I/O Port 6,the parallel data at the higher byte of the Data Bus is converted to serial form and appears at the output of the UART with the correct start, stop and data bits.

Before describing the design of the UART control logic, the function of the several bits controlled are summarized below:-

TBRL: When $\overline{TBRL}$ is LOW, the UART receives parallel data from its inputs (TBR1 to TBR8). A low to high

transition on $\overline{TBRL}$ initiates the transmission.

$\overline{DRR}$: A LOW level on this pin clears the DR status to a LOW level.

RRD: When RRD is HIGH, the parallel outputs (RBR1 to RBR8) of the UART are forced to a high impedance state.

EO: When EO is HIGH, the tristate buffers at the output lines of the UART are forced to a high impedance state.

The UART control logic was derived from the Port 6 select bit Y_6 (see Fig. 6.2), the input-output request, IOR, and the input-output mode, IOM, as described in the following truth table:-

IOR	IOM	Y_6	$\overline{TBRL}$	$\overline{DRR}$	RRD	EO	
0	0	0	0	1	1	1	output request
0	0	1	1	1	1	1	no request
0	1	0	1	0	0	0	input request
0	1	1	1	1	1	1	no request
1	0	0	1	1	1	1	no request
1	0	1	1	1	1	1	no request
1	1	0	1	1	1	1	no request
1	1	1	1	1	1	1	no request

The control bits $\overline{TBRL}$, $\overline{DRR}$, RRD and EO were therefore derived from the following Boolean operations:-

$$\overline{DRR} = RRD = EO = IOR + Y_6 + \overline{IOM}$$

and, $$\overline{TBRL} = IOR + IOM + Y_6$$

The clock to the UART was provided by using a LM555 free running oscillator together with two binary dividers (SN 7473) to provide four different baud rates as illustrated below (clock to the UART must be 16 times the required baud rate).

Baud Rate	Clock Frequency
1200	19200 Hz
600	9600 Hz
300	4800 Hz
110	1760 Hz

The serial input and output logic levels of the UART were converted to 20 mA loop and the CCITT V.24 (or EIA RS-232C) interface levels so that a wide range of terminals could directly be connected to the microcomputer.

The character length was chosen to be 10 bits: 1 start bit, 7 data bits, no parity bit and 2 stop bits (see Fig. 6.10).

The overall logic diagram of the serial communication channel is shown in Fig. 6.11.

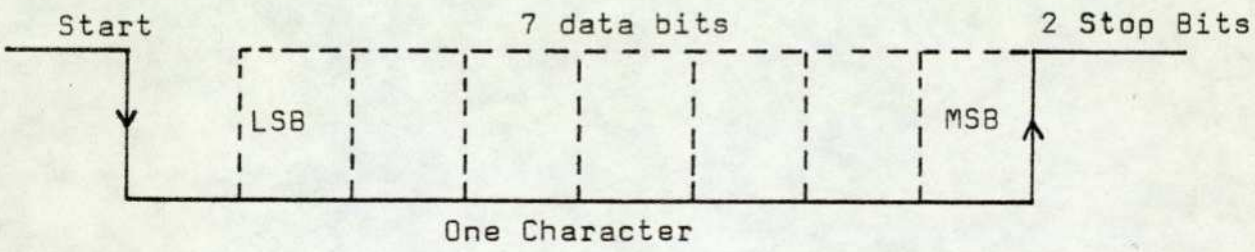


Figure 6.10 The character length was chosen as 1 start bit, 7 data bits, and 2 stop bits .

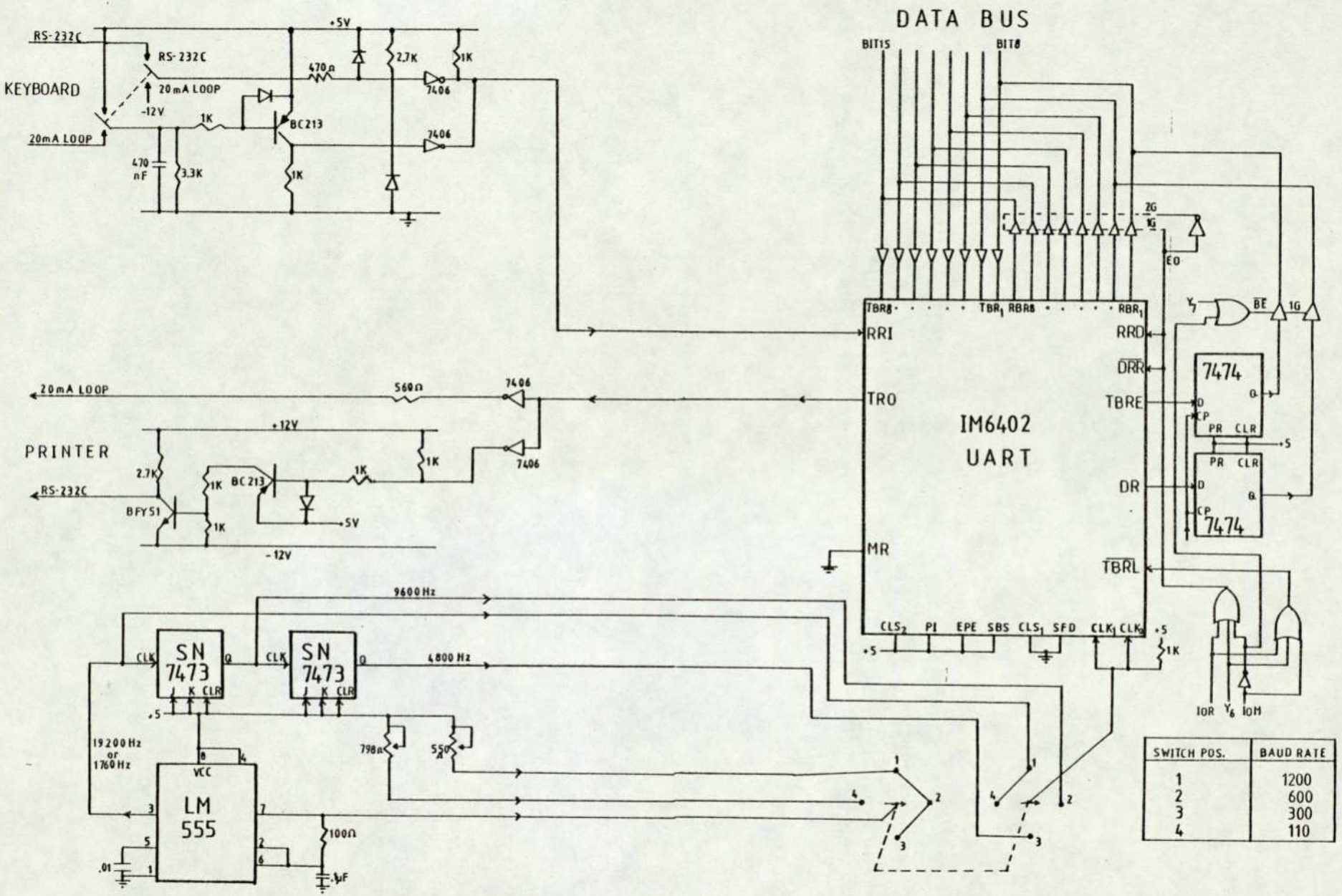


Figure 6.11 The serial communication channel

6.6 Conclusions

Chapter 6 has described the design of the input-output Unit of the microcomputer. It was shown that only two bits (IOM and IOR) were required to control the input-output Unit. With only four Am2917 type registers and a few logic gates it was possible to design a high spped 16 bit I/O port. A total of 256 I/O ports can easily be connected to the microcomputer.

Port 0 was connected as a 16 bit general purpose I/O port. Port 1 was connected as a data acquisition channel, interfacing the microcomputer to a sample/hold amplifier, a 12 bit A/D converter and a 12 bit D/A converter. Ports 6 and 7 were used to control a UART so that a wide range of user terminals could be connected to the microcomputer.

CHAPTER 7

7. THE PROGRAM CONTROL UNIT (PCU)

7.1 Introduction

The Program Control Unit was designed to provide addresses to the main memory so that data could be input or output from the required locations of the main memory.

The simplified block diagram of the PCU is shown in Fig. 7.1. The basic functions of the PCU can be summarized as follows:-

i. Selecting the Sequential Addresses

The sequential addresses of the main memory are selected by incrementing the Program Counter Register (PC) by one and then forcing the multiplexer to select the PC as the source of data for the Address Bus.

ii. Controlling the Unconditional Jump Macroinstructions

The unconditional jump macroinstructions occupy two successive locations in the main memory. The first word specifies that this is an unconditional jump macroinstruction, while the second word specifies the jump address (address of the next macroinstruction

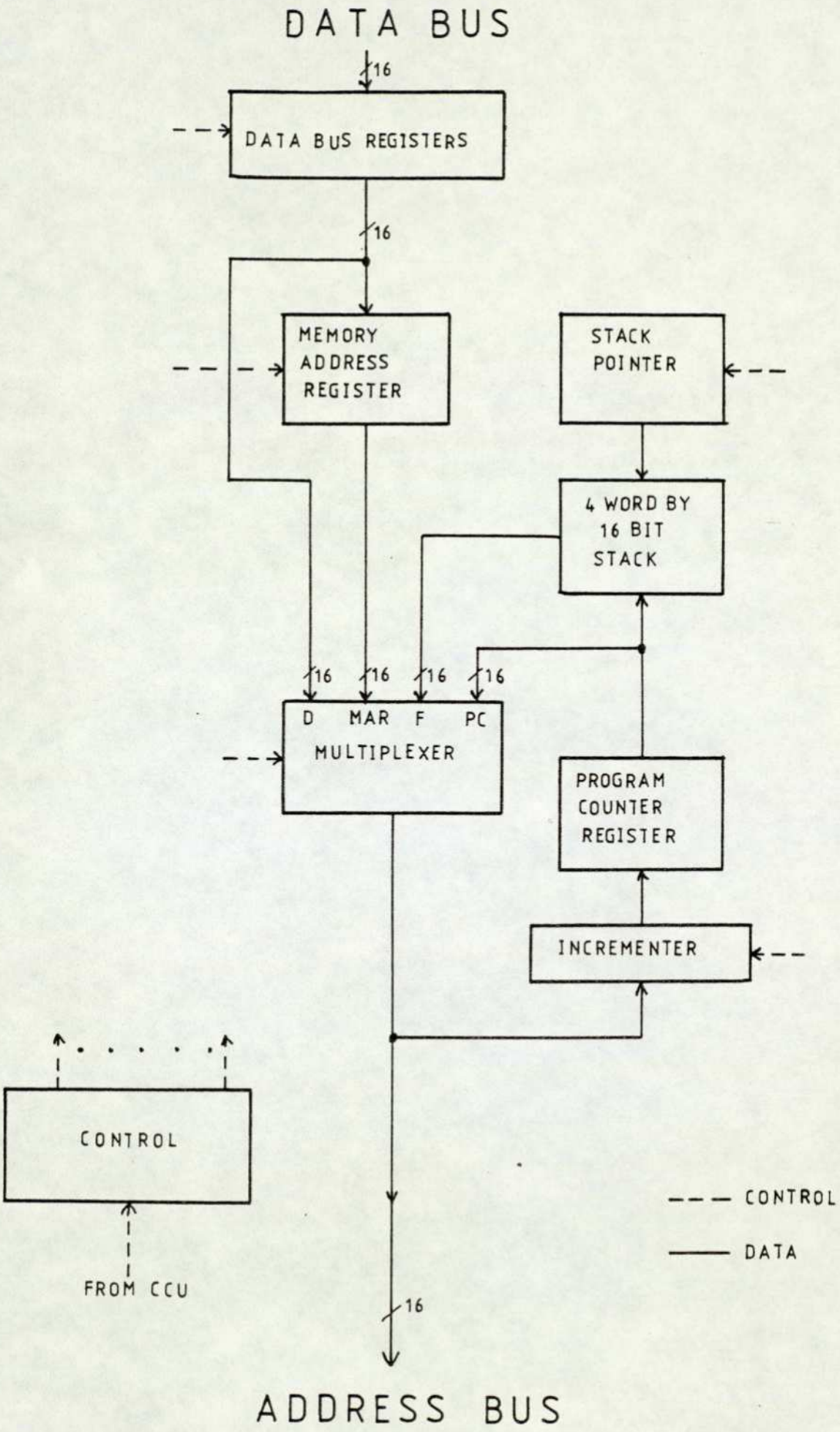


Figure 7.1 Simplified block diagram of the Program Control Unit (PCU) .

to be executed).

During the execution of an unconditional jump macroinstruction the PC is incremented by one so that the jump address will be output from the main memory and it will be available at the Data Bus. The multiplexer is then forced to select the 'D' input as the next address for the main memory, thus enabling an unconditional jump macroinstruction to be executed by the microcomputer.

iii. Controlling the Conditional Jump Macroinstructions

The conditional jump macroinstructions perform jump operations based upon the condition of the status register bits. If the required condition is true then the program control unit presents the jump address to the main memory as described in section ii. If on the other hand the required condition is false, then the address of the next sequential macroinstruction is selected and presented to the Address Bus.

iv. Controlling the Subroutine Call and Return Macroinstructions

During a subroutine call macroinstructions, the stack pointer is incremented by one and contents of the PC are pushed onto the stack. (see Fig.7.1).

At the same time, the address of the subroutine is output from the main memory and is available on the Data Bus. The multiplexer is then forced to select the 'D' input as the next address for the main memory, thus enabling a subroutine call macroinstruction to be executed.

During a return from subroutine, the stack pointer is decremented by one and the contents of the stack are selected at the output of the PCU. The PC is then incremented by one and presented to the Address Bus, thus enabling the next macroinstruction after the subroutine call macroinstruction to be available.

7.2 The Design of the PCU

The functions of a typical PCU are very similar to those of a microprogram sequencer. The major difference is that a PCU addresses the main memory where a microprogram sequencer is used to provide addresses to a microprogram memory. Because of this analogy it was thought to use a microprogram sequencer for addressing the main memory. An examination of the commercially available microprogram sequencers (Table 3.3) reveal that the Am2910 provides the maximum addressing range amongst the fixed address sequencers. However, the 4K maximum addressing range provided by this

device is not sufficient in many main memory applications. The only other choice was then to use a bit-slice microprogram sequencer and expand its addressing range by cascading the required number of identical devices.

The Program Control Unit was designed by using the Am2911 type sequencers [8], as shown in Fig. 7.2. As the Am2911s are the basic building blocks of the PCU, it is essential to understand their operation in detail.

The Am2911 is a four-bit bit slice sequencer. It contains a four-input multiplexer that is used to select either the memory address register, direct inputs, program counter register or file (stack) as the source for the Y outputs (see Fig. 7.3). This multiplexer is controlled by the S_o and S_1 inputs as described in the following table:-

S_1	S_o	Source for Y outputs	Symbols
0	0	Program counter register	PC
0	1	Memory address register	MAR
1	0	File (stack)	F
1	1	Direct inputs	D

The four-bit incrementer at the input of the program counter register has carry-in (C_n) and carry-out (C_{n+4}) pins so that any number of devices can be cascaded. The 4x4

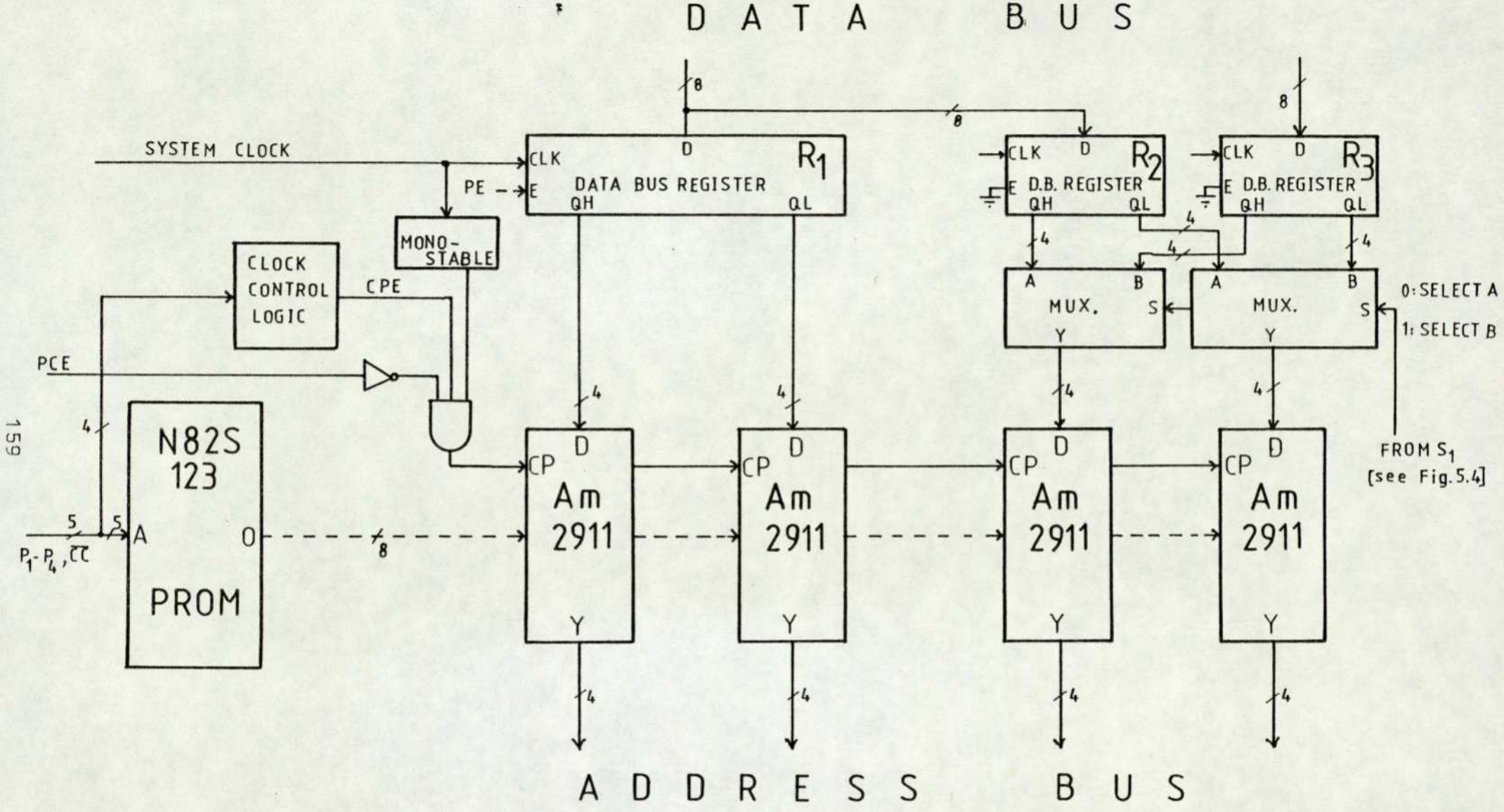


Figure 7.2 Block diagram of the Program Control Unit .

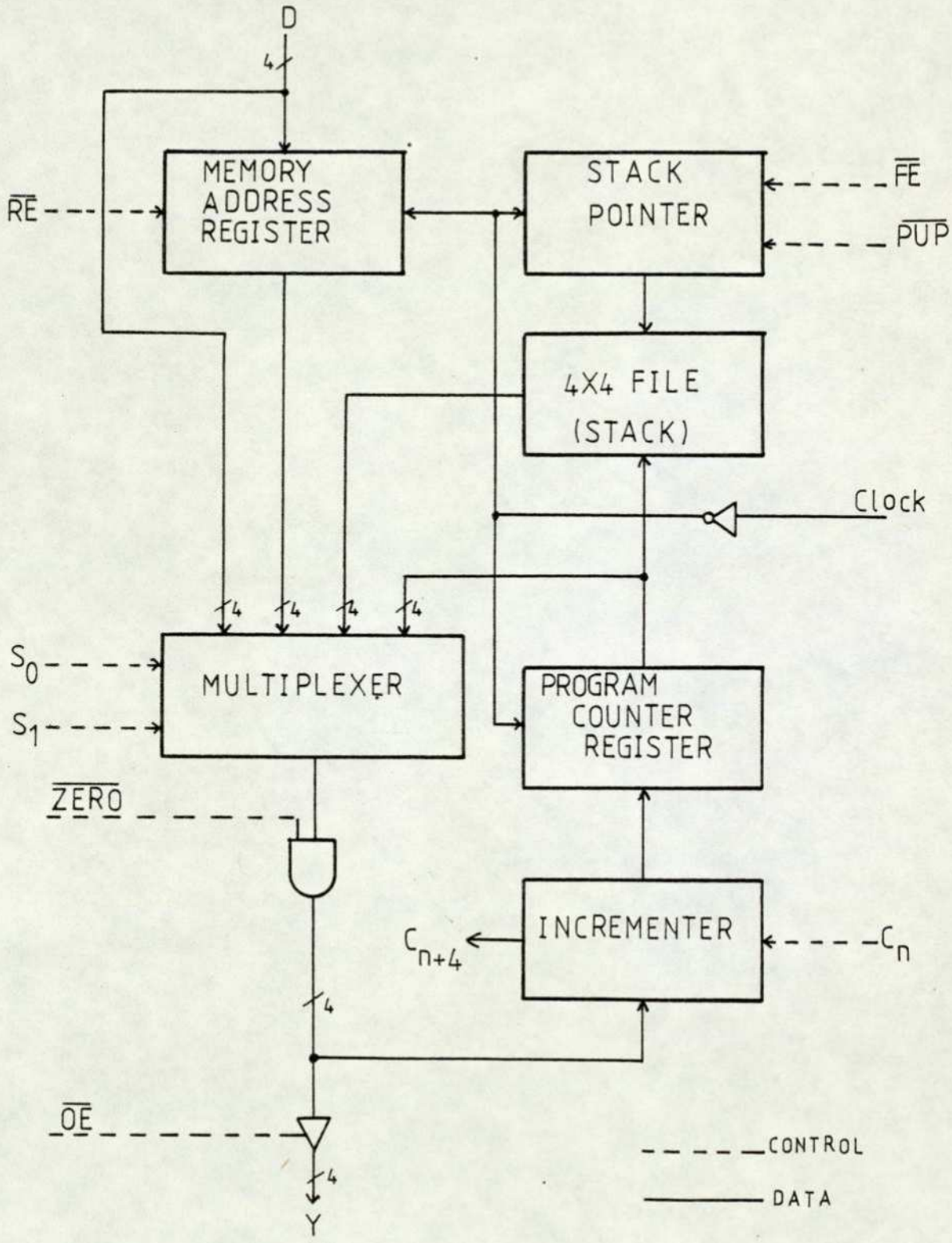


Figure 7.3 Block diagram of the Am2911

stack is addressed by a built-in stack pointer. The stack pointer operates as an up/down counter with separate push/pop (PUP) and file enable ($\overline{FE}$) inputs that control the stack as follows:-

$\overline{FE}$	$\overline{PUP}$	Stack Operation
0	0	Pop (decrement stack pointer)
0	1	Increment stack pointer, then push PC onto stack.
1	0	No change
1	1	No change

The memory address register is controlled by the register enable input, $\overline{RE}$, such that when $\overline{RE}$ is LOW, new data is entered into the memory address register on the clock LOW-to-HIGH transition.

The $\overline{ZERO}$ and the $\overline{OE}$ control bits are used to control the outputs of the Am2911 as follows:-

ZERO	OE	Y
0	0	0
X	1	High impedance
1	0	Source selected by S_0, S_1

7.2.1 Cascading and Control of the Am2911s

Four Am2911s were cascaded by the ripple carry connection scheme, to provide a 16 bit Address Bus. The width of the Address Bus can be incremented further by cascading more identical devices.

The eight control inputs of the Am2911s were derived from a 32 word by 8 bit bipolar PROM (N82S123) which was programmed to implement 16 different PCU instructions. As shown in Table 7.1, only four bits (P_1 to P_4) were required (a saving of four bits) to control the Am2911s. Referring to Table 7.1, the function of the various PCU instructions are described below:-

Instruction 0 clears the outputs of the Am2911s by forcing the $\overline{ZERO}$ input LOW. This instruction was used to initialize (see Chapter 12) the microcomputer so that execution starts from address zero of the main memory.

Instruction 1 increments the PC by one and selects it as the source for the Address Bus. This instruction can be used whenever it is required to address the main memory sequentially, such as during the macroinstruction fetch cycle, or during the execution of the macroinstructions that occupy two successive locations in the main memory.

Instruction 2 loads the memory address register from the

INST No.	PCU CONTROL					PCU OPERATION		
	P_4	P_3	P_2	P_1	$\overline{CC}$		Combinational	Sequential (after clock)
0	0	0	0	0	X	CLEAR	ADDR ← 0	PC ← 0
1	0	0	0	1	X	CONTINUE	ADDR ← (PC)	PC ← (PC)+1 ; ADDR ← (PC)
2	0	0	1	0	X	LOAD MAR	ADDR ← (PC)	MAR ← D ; ADDR ← (PC)
3	0	0	1	1	X	LOAD PC	ADDR ← D	PC ← D ; ADDR ← D
4	0	1	0	0	X	LOAD PC INC	ADDR ← D	PC ← D+1 ; ADDR ← D
5	0	1	0	1	X	SOURCE MAR	ADDR ← (MAR)	ADDR ← (MAR)
6	0	1	1	0	X	SOURCE D	ADDR ← D	ADDR ← D
7	0	1	1	1	0	CONTINUE	ADDR ← (PC)	PC ← (PC)+1 ; ADDR ← (PC)
7	0	1	1	1	1	SUSPEND	ADDR ← 3-STATE	ADDR ← 3-STATE
8	1	0	0	0	0	RET. INT.	ADDR ← ((SP))	PC ← ((SP)) ; SP ← (SP)-1 ; ADDR ← ((SP))
8	1	0	0	0	1	HOLD	ADDR ← (PC)	ADDR ← (PC)
9	1	0	0	1	0	CONTINUE	ADDR ← (PC)	PC ← (PC)+1 ; ADDR ← (PC)
9	1	0	0	1	1	JUMP MAR	ADDR ← (MAR)	PC ← (MAR)+1 ; ADDR ← (MAR)
10	1	0	1	0	0	CONTINUE	ADDR ← (PC)	PC ← (PC)+1 ; ADDR ← (PC)
10	1	0	1	0	1	JUMP D	ADDR ← D	PC ← D+1 ; ADDR ← D
11	1	0	1	1	X	CALL SUB D	ADDR ← D	SP ← (SP)+1 ; ((SP)) ← (PC) ; PC ← D ; ADDR ← D
12	1	1	0	0	0	CONTINUE	ADDR ← (PC)	PC ← (PC)+1 ; ADDR ← (PC)
12	1	1	0	0	1	RTN SUB	ADDR ← ((SP))	PC ← ((SP))+1 ; SP ← (SP)-1 ; ADDR ← ((SP))
13	1	1	0	1	X	SUSPEND	ADDR ← 3-STATE	ADDR ← 3-STATE
14	1	1	1	0	0	CONTINUE	ADDR ← (PC)	PC ← (PC)+1 ; ADDR ← (PC)
14	1	1	1	0	1	CALL SUB MAR	ADDR ← (MAR)	SP ← (SP)+1 ; ((SP)) ← (PC) ; PC ← (MAR) ; ADDR ← (MAR)
15	1	1	1	1	X	RTN SUB	ADDR ← ((SP))	PC ← ((SP))+1 ; SP ← (SP)-1 ; ADDR ← ((SP))

Table 7.1a PCU instructions (ADDR = Address Bus ; SP = Stack Pointer ; D = 'D' inputs of Am2911)

INST. NO.	CLK	PROM ADDRESS					PROM DATA (Am2911 Control bits)							
		A_4	A_3	A_2	A_1	A_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
		P_4	P_3	P_2	P_1	$\overline{CC}$	$\overline{OE}$	C_n	$\overline{Z}$	S_0	S_1	$\overline{FE}$	PUP	$\overline{RE}$
0	↑	0	0	0	0	X	0	0	0	X	X	1	X	1
1	↑	0	0	0	1	X	0	1	1	0	0	1	X	1
2	↑	0	0	1	0	X	0	0	1	0	0	1	X	0
3	↑	0	0	1	1	X	0	0	1	1	1	1	X	1
4	↑	0	1	0	0	X	0	1	1	1	1	1	X	1
5	0	0	1	0	1	X	0	0	1	1	0	1	X	1
6	0	0	1	1	0	X	0	1	1	1	1	1	X	1
7	↑	0	1	1	1	0	0	1	1	0	0	1	X	1
7	↑	0	1	1	1	1	1	0	1	0	0	1	X	1
8	↑	1	0	0	0	0	0	0	1	0	1	0	0	1
8	↑	1	0	0	0	1	0	0	1	0	0	1	X	1
9	↑	1	0	0	1	0	0	1	1	0	0	1	X	1
9	↑	1	0	0	1	1	0	1	1	1	0	1	X	1
10	↑	1	0	1	0	0	0	1	1	0	0	1	X	1
10	↑	1	0	1	0	0	0	1	1	1	1	1	X	1
11	↑	1	0	1	1	X	0	0	1	1	1	0	1	1
12	↑	1	1	0	0	0	0	1	1	0	0	1	X	1
12	↑	1	1	0	0	1	0	1	1	0	1	0	0	1
13	↑	1	1	0	1	X	1	0	1	0	0	1	X	1
14	↑	1	1	1	0	0	0	1	1	0	0	1	X	1
14	↑	1	1	1	0	1	0	0	1	1	0	0	1	1
15	↑	1	1	1	1	X	0	1	1	0	1	0	0	1

X = don't care ; ↑ = LOW-to-HIGH transition of the clock

Table 7.1b PROM data for the PCU instructions .

'D' inputs of the Am2911s, without incrementing the PC. This instruction can be used to store a temporary address in the PCU while addressing the memory sequentially.

Instruction 3 loads the PC from the 'D' inputs and presents this address to the Address Bus. This instruction was used for controlling the unconditional jump macroinstruction.

Instruction 4 is similar to instruction 3 but here the PC is incremented by one.

During the execution of instructions 5 and 6, the clock is disabled from the PCU. Instruction 5 selects the contents of the memory address register as the source of data for the Address Bus. This instruction can be used to select a temporary address as the next memory address, without changing contents of the PC.

Instruction 6 selects the data at the 'D' inputs as the source for the Address Bus, without changing contents of the PC. This instruction was used to address a memory location when it is required not to destroy contents of the PC (e.g. during the execution of a macroinstruction which stores data at a location of the main memory not specified by the PC).

Instruction 8 was used for servicing the processor interrupts.

Instruction 9 and 10 can be used to control the conditional jump macoinstructions. If the condition tested is true ($\overline{CC}$ = HIGH), then the jump macroinstruction will be executed. If on the other hand the condition is false ($\overline{CC}$ = LOW) then the next sequential macroinstruction will be addressed by the PCU.

Instruction 11 was intended to be used to control an unconditional subroutine call macroinstruction. This instruction increments the stack pointer by one and pushes the current PC onto stack. The source of data for the Address Bus (subroutine service address) is derived from the 'D' inputs of the Am2911s.

Instruction 12 is the conditional return from subroutine instruction. If the condition tested is true ($\overline{CC}$ = HIGH) then the stack pointer is decremented by one and the source of data for the Address Bus is selected from the stack. If the condition is false ($\overline{CC}$ = LOW) then the next sequential macroinstruction will be addressed by the PCU.

Instruction 13 forces the outputs of the Am2911s to a high impedance state and performs no other operations.

Instruction 14 was intended to be used to control the conditional subroutine call macroinstructions. If the condition tested is true ($\overline{CC}$ = HIGH) then the stack pointer is incremented by one and the current PC is pushed onto

stack. The source of data for the Address Bus is derived from the memory address register. If on the other hand the condition is false ($\overline{CC}$ = LOW) then the next sequential macroinstruction will be addressed.

Instruction 15 is the unconditional return from subroutine instruction. This instruction decrements the stack pointer by one and selects the stack as the source for the Address Bus.

7.2.2 The Clock to the Am2911s

The clock to the Am2911s was delayed by 120 ns, by using a monostable multivibrator (SN74121). By this approach it was possible to generate addresses and access the main memory in the same single microinstruction (see Chaper 12).

The clock to the Am2911s was controlled (after the delay) by the control bit 'PCE' such that whenever 'PCE' is HIGH, the clock is disabled. By controlling 'PCE', it is therefore possible to enable or disable the clocked logic of the Am2911s. (This feature is very useful during the execution of microinstructions which do not require any PCU operation to be performed).

The clock control logic disables the clock from the Am2911s during the execution of the PCU instructions 5 and

6 (see Table 7.1). The truth table of the clock control logic is shown in Table 7.2. The Boolean operations for the clock control logic were derived from Table 7.2 and were implemented as follows:-

$$CP_E = \overline{\overline{P}_4 \cdot P_3 \, (P_1 \cdot \overline{P}_2 + P_2 \cdot \overline{P}_1)}$$

7.2.3 The Data Bus Registers

The program control unit receives external addresses from the Data Bus via clocked registers (see Fig. 7.2). These registers were controlled such that whenever 'PE' is LOW, new data is clocked into the registers and when 'PE' is HIGH the registers retain their data independent of the clock input. These registers provide data for the 'D' inputs of the Am2911s.

7.2.4 Selecting the 8/16 bits of operation

When working with 8 bits of data length, a typical jump macroinstruction occupies three successive 16 bit locations in the main memory. The first higher byte specifies the operation to be performed. The second higher byte of the macroinstruction specifies the higher byte of the jump address. Similarly the third higher byte of the macroinstruction specifies the lower byte of the jump address.

P_4	P_3	P_2	P_1	CP_E
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table 7.2. The truth table of the Clock Control Logic of the PCU.

The multiplexers at the output of the Data Bus registers (see Fig. 7.2) ensure that the correct addresses are presented to the 'D' inputs of the Am2911s.

When working with 16 bits of data length, only one microinstruction is required to clock the data at the Data Bus to the 'D' inputs of the Am2911s.

When working with 8 bits of data length, two microinstructions are required to present a 16 bit address to the 'D' inputs of the Am2911s. During the first microinstruction, 'PE' is enabled and the higher byte of the jump address is loaded to the data bus register R_1 (see Fig. 7.2). During the second microinstruction, 'PE'is disabled and the lower byte of the jump address is loaded to the data bus register R_2 (Fig. 7.2) . The multiplexers then select the 16 bit address as the source for the 'D' inputs of the Am2911s. The multiplexers were controlled from the 8/16 bit select switch S_1 (see section 5.1.2).

The overall logic diagram of the Program Control Unit is shown in Fig. 7.4.

7.3 Conclusions

Chapter 7 has described the design of the Program Control Unit of the microcomputer. It is shown that six

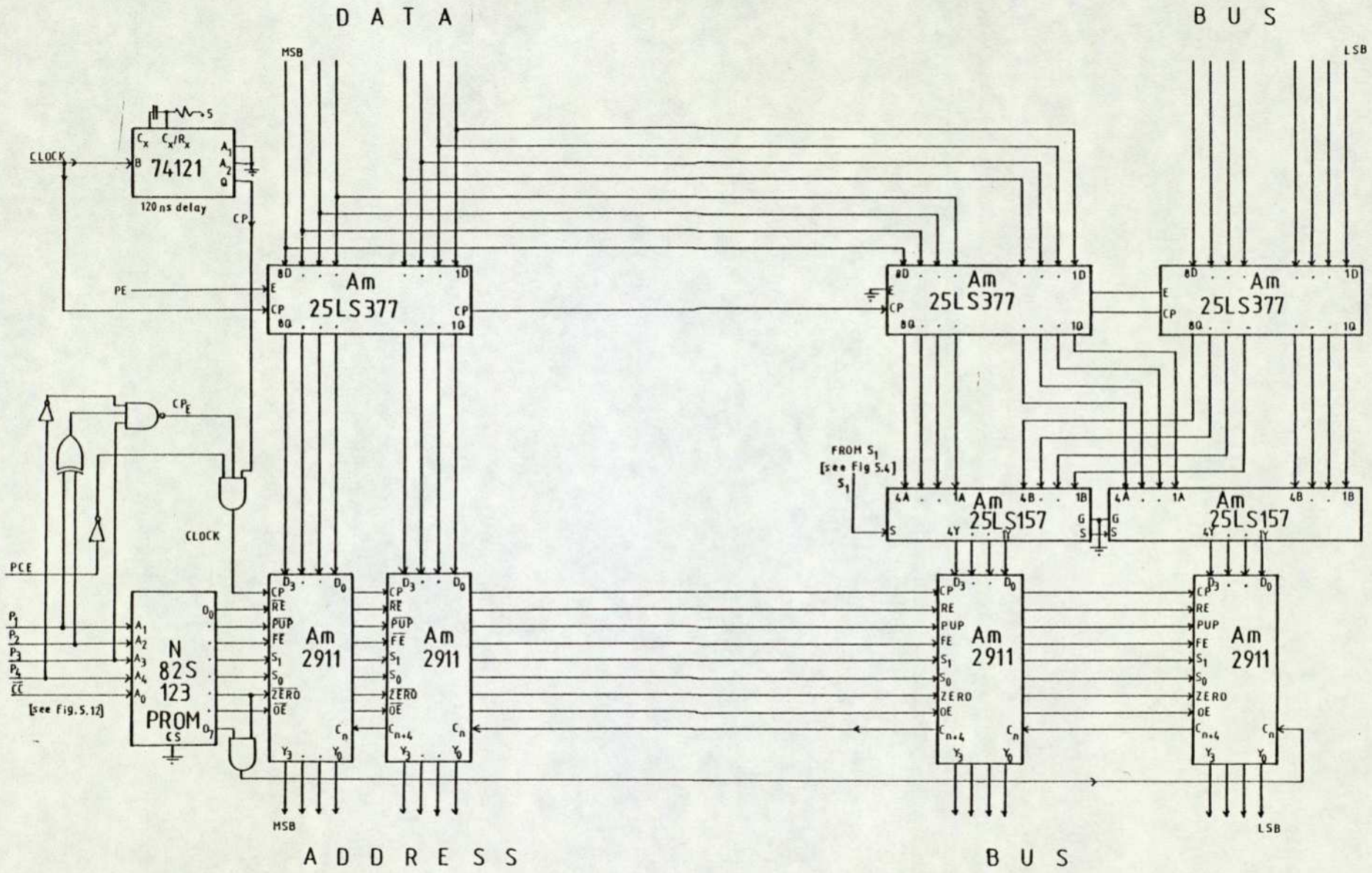


Figure 7.4 The Program Control Unit

bits (P_1,P_2,P_3,P_4, PCE and PE) are required to control the PCU. A 16 bit Address Bus was created by cascading four Am2911 type four-bit sequencers. A bipolar PROM was programmed to control the Am2911s and it provided 16 different PCU instructions.

CHAPTER 8

8. THE MAIN MEMORY

8.1 Introduction

Memories are a major subsection of the hardware in every microcomputer system. A typical system uses memory for storage of both programs and data. The program memory is usually required to be non-volatile (i.e. the programs are not lost when power is removed) and semiconductor read-only-memories (ROM) are used in most cases. There are four main types of ROMs:-

i. Mask Programmable: This type of ROMs are programmed during their manufacture. Once programmed, the data pattern of the ROM cannot be altered. Mask programmable ROMs are best suited to applications where a program requires no more modifications and where a very large number of identical ROMs are required.

ii. Fusible Link (PROM): These are programmable read-only-memories. The programming is carried out by applying high-voltage pulses to the required locations of the device. Once programmed, the data pattern of the PROMs can not be altered. These type of memories are best suited to applications where a program is in its final form and

where a small to medium quantity of such devices are required.

iii. Electrically reprogrammable, U.V. Erasable (EPROM):

These memories are programmed by applying high-voltage pulses to the required locations. They have the advantage that they may be erased by shining ultra-violet (UV) light upon them and then subsequently reprogrammed. EPROMs are very useful for development work, where the program is likely to be changed quite frequently.

iv. Electrically Erasable, Electrically Reprogrammable (EAROM):

The main advantage of these memories are that they are word-alterable and the whole program does not have to be re-written to make a small modification. However they have the disadvantage that they are slower compared to the other read-only-memories. These memories are usually used in low speed prototype systems.

For the data storage or small development programs it is frequently required to change the contents of the memory. Semiconductor Random Access Memories (RAM) are often used for this kind of applications. The main disadvantage of these memories is that they are volatile and all data is lost

when the power is removed. Semiconductor RAM memories can either be static or dynamic. Static memories have the advantages that they are easy to use. Dynamic memories are often used where high memory density is required, but they have the disadvantage that special refresh circuitry must be incorporated to cycle the memory every few milliseconds.

8.2 Design of the Main Memory Unit

The main memory unit was designed from a combination of PROM and RAM type of semiconductor memories. The PROM occupies the first 512 locations (see Fig. 8.1) of the main memory and it was programmed to store the Operating System (Monitor) of the microcomputer (see Chapter 13). The RAM occupies 1K (1024 words) of the main memory and it was intended to be used to store small development programs and data.

The block diagram of the main memory unit is shown in Fig. 8.2. The address inputs of the memories were connected to the Address Bus via the address bus buffers. The function of these buffers are to minimize the A.C. loading on the Address Bus and to provide high D.C. drive capability to the Address Bus.

The main memory was controlled from the CCU by the two control signals MRQ (memory request) and the MMD

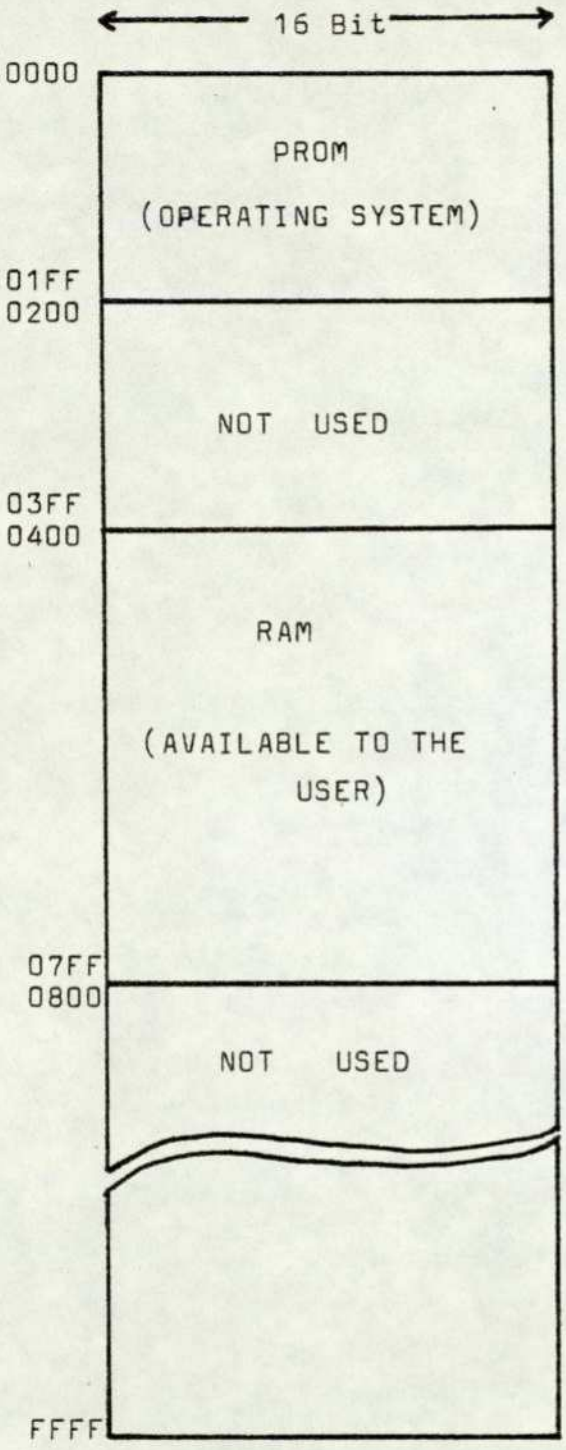


Figure 8.1 Main memory map of the microcomputer .

(memory mode) as described by the following truth table:-

MRQ	MMD	Main Memory Operation
0	0	output request (read)
0	1	input request (write)
1	0	No request
1	1	No request

The buffers (Am74LS241) at the output of the memories drive the Data Bus during a memory read cycle. During a memory write cycle or whenever there is no main memory operation, these buffers are forced to high impedance state. The memory output control logic (see Fig. 8.2) controls these buffers as described in the following table:-

MRQ	MMD	1G (buffer enable input)
0	0	0, enable the buffers
0	1	1, disable the buffers
1	0	1, disable the buffers
1	1	1, disable the buffers

The buffers were thus controlled as described by the following Boolean operations:-

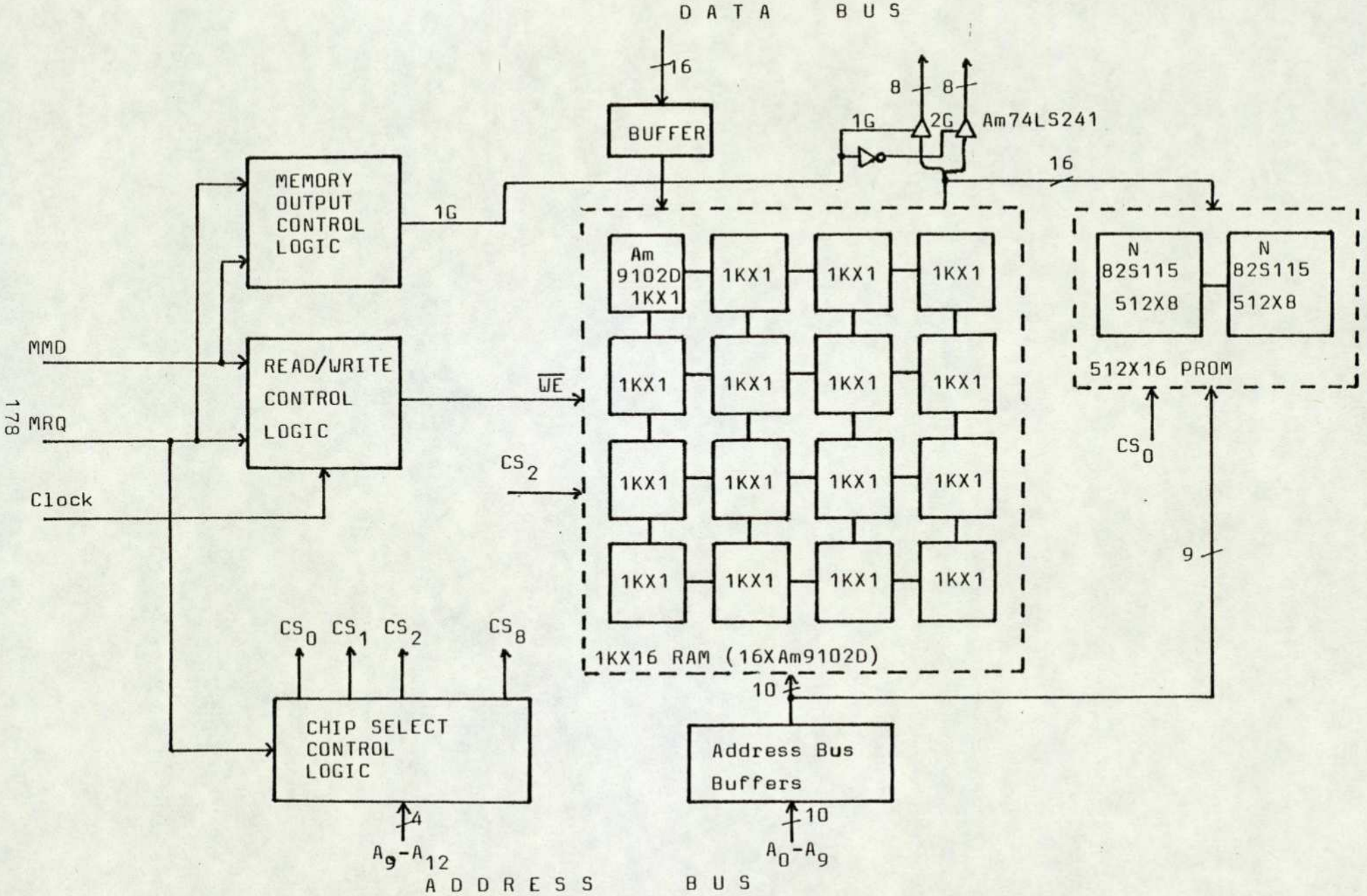


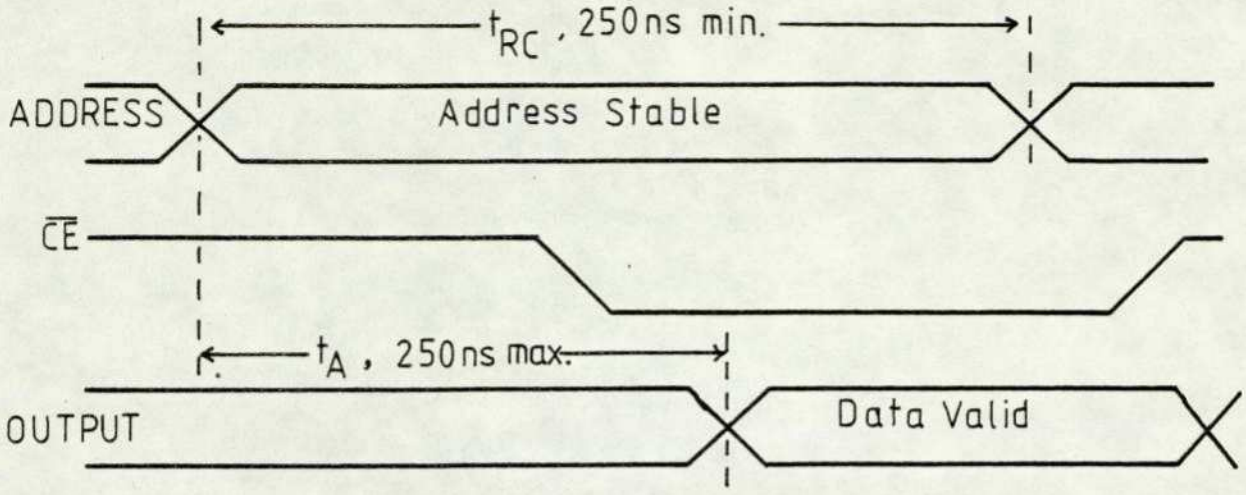
Figure 8.2 Block diagram of the Main Memory Unit .

1G = MRQ + MMD

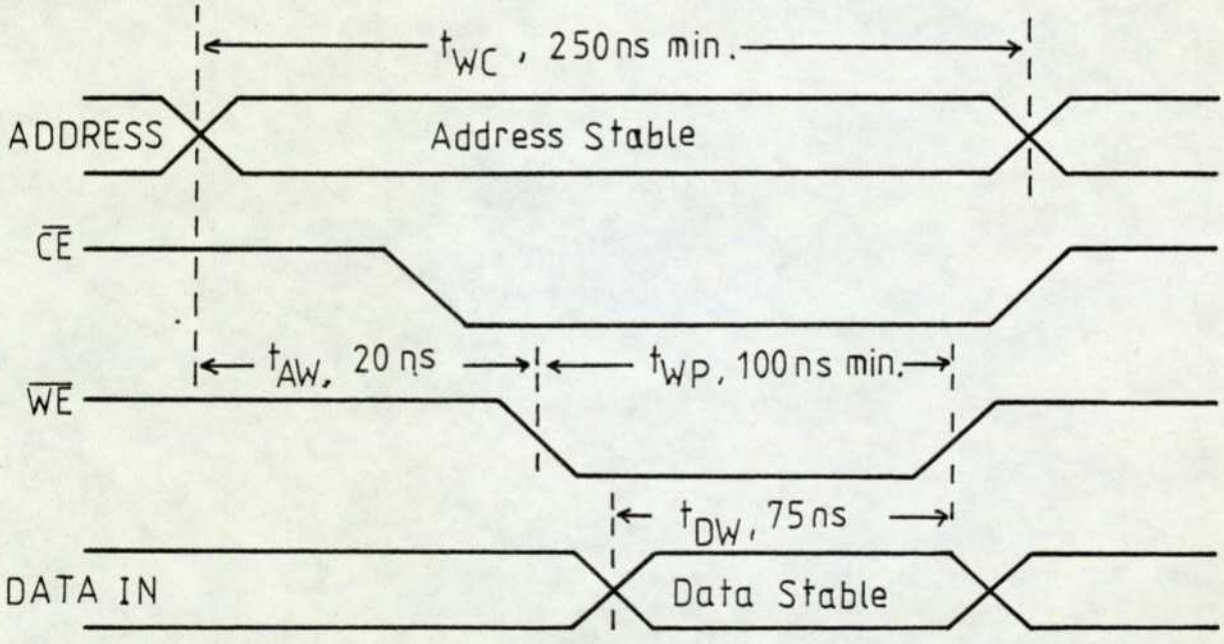
The RAM part of the main memory was designed by using Am9102D type static MOS memories, with an access time of 250 ns. The main reason of using MOS memories was because of their low cost (for faster response times these memories can be replaced by bipolar memories with an increase in the overall cost by a factor of 5). The Am9102D memories are manufactured as 1K X 1 bit and thus 16 such devices were used for a 16 bit data length. A memory operation can only be performed when the Chip Enable ($\overline{CE}$) inputs of the memories are LOW. When $\overline{CE}$ is HIGH, the memories are disabled and their outputs are forced to a high impedance state. When the RAM is enabled, it can be in either Read or Write mode, determined by the logic level on the Write Enable ($\overline{WE}$) input, as described follows:-

i. Read Mode: In the Read mode, the $\overline{WE}$ input of the RAM must be HIGH. The correct data will then be output from the addressed location after a time delay of t_A, where t_A is the access time of the memory (see Fig. 8.3a).

ii. Write Mode: The write mode is initiated by applying a logic LOW to the $\overline{WE}$ input of the RAM. (see Fig.8.3b). This condition must last for a minimum length of time, specified as t_{wp}, minimum required write pulse width. Backtracking in time from the end of the write pulse, the Address inputs must be stable for $t_{AW} + t_{wp}$, where t_{AW} is the



a. READ MODE



b. WRITE MODE

Figure 8.3 Switching waveforms of the Am9102D RAM .

Address set-up time. Also, the data input must be stable for t_{DW} before the end of the write pulse. The write mode must always be terminated by applying a logic HIGH to the $\overline{WE}$ input of the RAM.

The read/write control logic controls the $\overline{WE}$ inputs of the RAMs so that the read and the write operations can be performed correctly. This logic was designed by using a monostable multivibrator (see Fig. 8.4) together with a few logic gates and its truth table is as follows:-

MRQ	MMD	CLK	$\overline{WE}$
0	0	X	1 read mode
0	1		write mode·
1	X	X	1 no memory request

In a write operation, the HIGH-to-LOW transition of the system clock triggers the monostable multivibrator to initiate the write mode. After 150 ns, the $\overline{WE}$ is forced HIGH to terminate the write mode.

The PROM part of the main memory was designed by using N82S115 type bipolar memories, with an access time of 60 ns. These memories are manufactured as 512 words by 8 bits each and thus 2 such devices were used for a 16 bit data length.

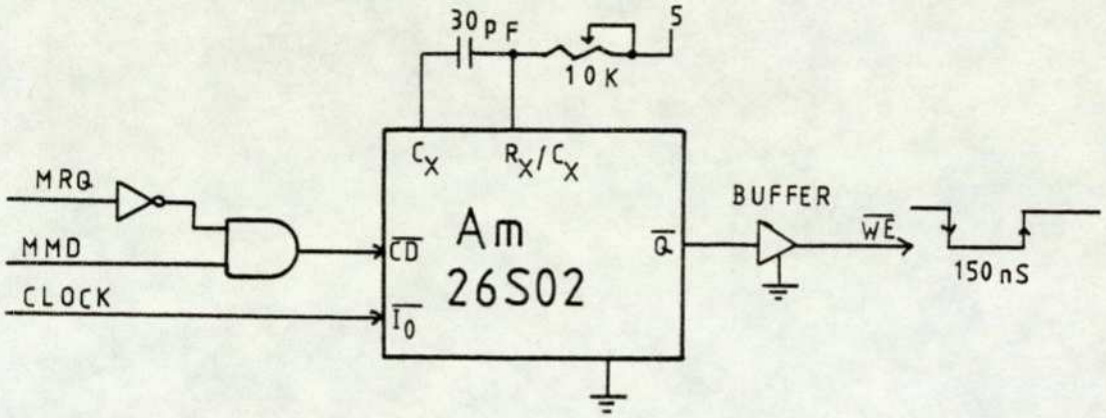


Figure 8.4 Logic diagram of the read/write control logic .

The chip select control logic was designed such that initially up to 8K of memory can directly be connected (without further decoding) to the microcomputer as segments of 1K. The block diagram and the addressing range of the chip select control logic is shown in Fig. 8.5. The 1-of-8 decoder operates such that only one output is LOW at any time depending upon the address inputs. The 512 word select logic was used to enable the first two blocks of the main memory, each 512 words. The truth table of this logic is as follows:-

CS	A_9	CS_0	CS_1	
0	0	0	1	select 1st block
0	1	1	0	select 2nd block
1	X	1	1	1st and 2nd blocks not selected

The first two blocks of the main memory were therefore enabled as defined by the following Boolean operations:-

$$CS_0 = A_9 + CS$$

and,

$$CS_1 = \overline{A_9} + CS$$

The overall logic diagram of the main memory unit is shown in Fig. 8.6.

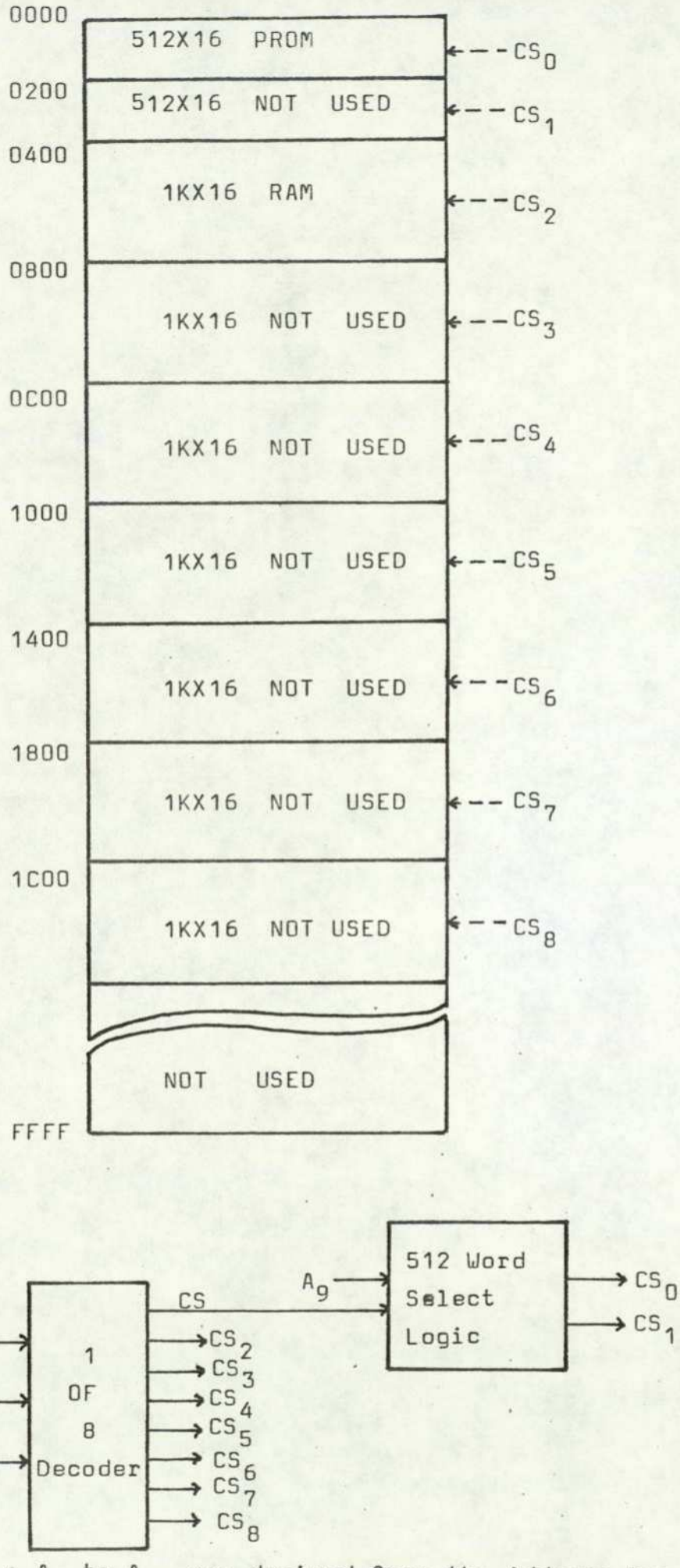


Figure 8.5 The chip select control logic was designed to select 8K of memory in blocks of 1K .

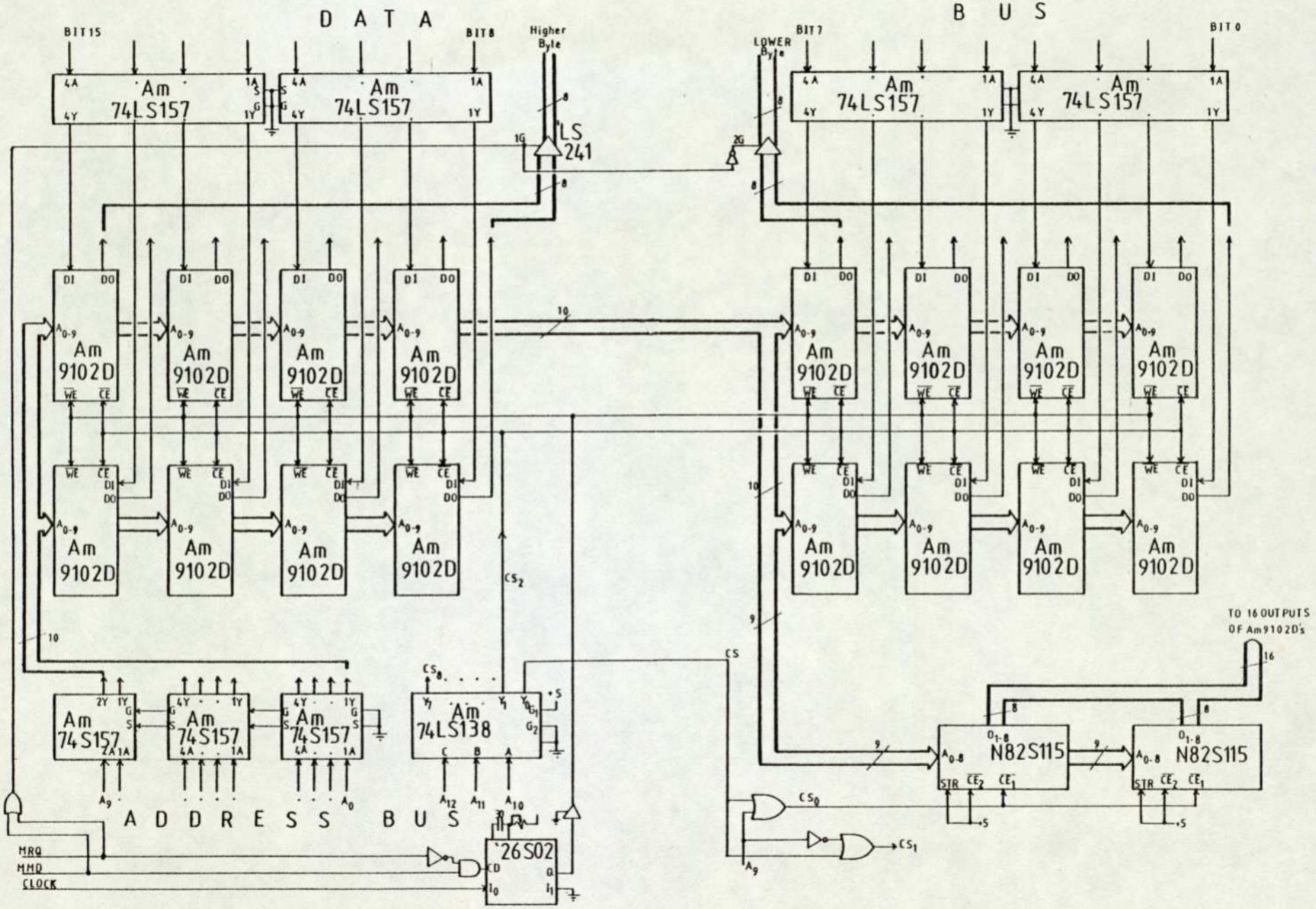


Figure 8.6 The Main Memory

8.3 Conclusions

This Chapter has described the design of the main memory unit of the microcomputer. With 16 bits of Address Bus, it is possible to connect up to 64K of memory to the system. Only 1½K of memory was connected to the microcomputer and the address lines were decoded such that up to 8K of memory could be used without the need of any further decoding. PROM and RAM type of memories were used in the design. The PROM part (512 words) was programmed to store the Operating System of the microcomputer (see Chapter 13). The RAM part (1024 words) was intended to be used to store small development programs and data.

CHAPTER 9

9. THE INTERRUPT CONTROL UNIT

9.1 Introduction

Interrupts are asynchronous hardware-generated signals that change the execution sequence of the main program. The occurence of the interrupts are not under control of the program sequence. An interrupting device indicates that an event external to the current executing program has occurred and it requires immediate service by the processor. Upon receiving and accepting an interrupt, the processor will complete the execution of the current microinstruction and then jump to a routine that identifies and services the requirements of the interrupting device. After servicing an interrupt, the processor will resume its normal main program sequence by returning to the point in the main program at which the interrupt has occurred. Since the processor cannot predict what state it will be in when an interrupt request is received, it should save the contents of certain registers (context switching) prior to servicing an interrupt, so that an interrupt will appear transparent to the normal execution of the main program.

The typical operations performed at the recognition of an interrupt are illustrated in Fig. 9.1. The first operation of the interrupt service routine disables (by hardware) the interrupt control unit from further recognition

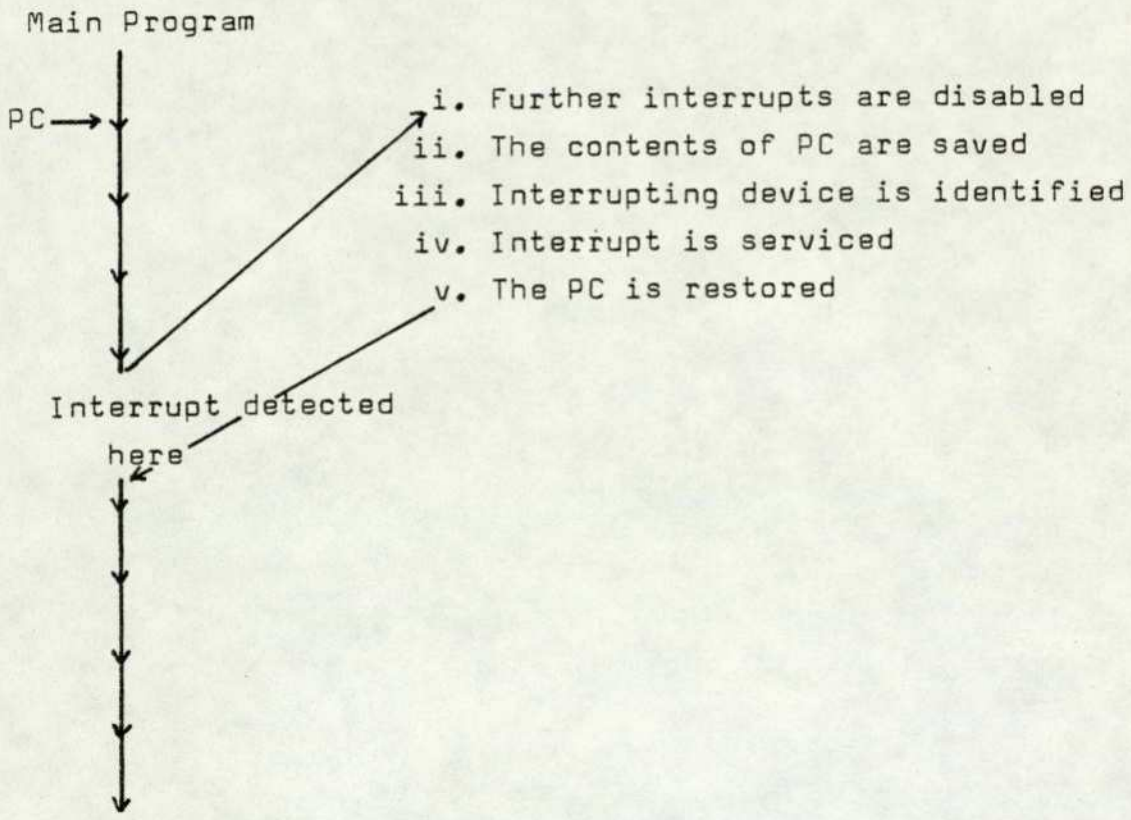


Figure 9.1 The typical operations performed at the recognition of an interrupt .

of interrupt requests. Interrupt requests may be re-enabled when the processor is ready to handle a new interrupt. The contents of the Program Counter Register, PC, are then saved. The interrupting device is identified and the software associated with it is executed. The last operation of the interrupt service routine restores the PC, so that the execution of the main program will resume from the interrupted point.

9.2 Interrupt Structures

There are several interrupt structures [11] that can be implemented and the choice of a particular interrupt mechanism is mainly influenced by the trade-off between hardware and software. Some of the commonly used interrupt structures are described below:-

9.2.1 Single-Level, Multiple Poll Interrupt Structure

This is the basic structure from which all other hardware interrupt structures are modelled. In this structure, all the interrupt signals from the various peripheral devices are connected together (wire or) to form a single line that is physically brought into the processor (see Fig. 9.2a). When the processor recognizes an interrupt request, it polls all the devices to find the highest priority device requesting service. Thus the

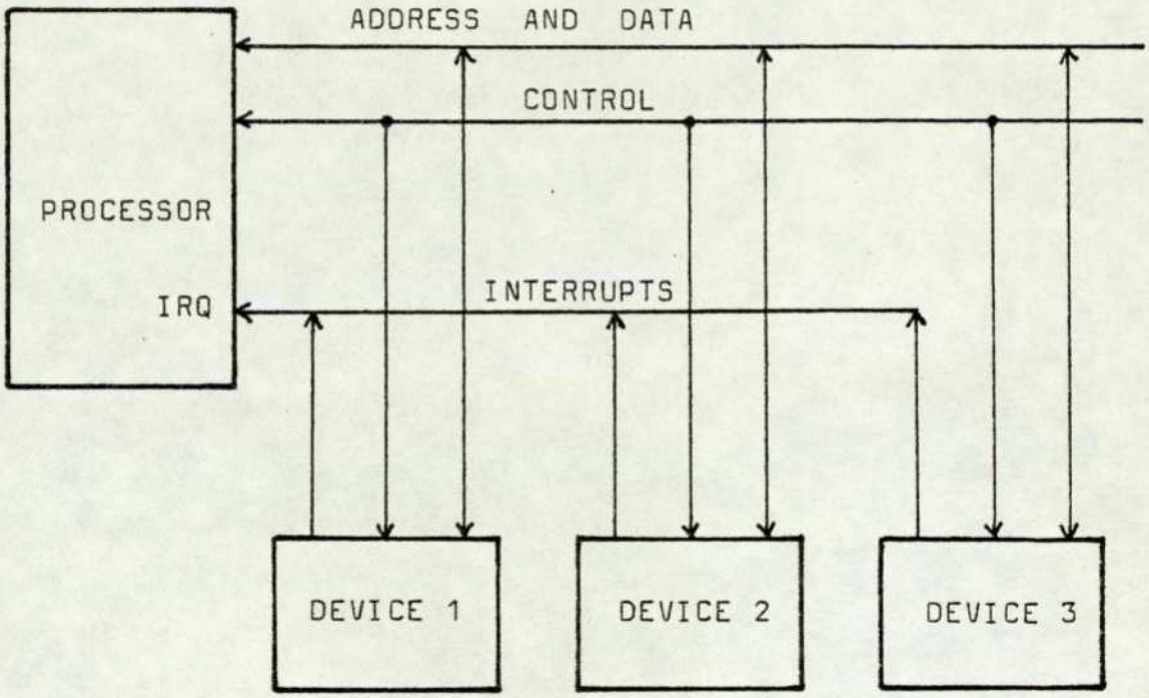


Figure 9.2a Single-level, multiple poll interrupt structure .

basic identification method is strictly software-oriented and priority is introduced via the order in which the devices are polled (see Fig. 9.2b). The devices that issue concurrent interrupt requests are serviced as they are encountered in the polling sequence. Although this scheme provides dynamic reallocation of priority, it has the major drawbacks that the interrupt response time to identifying the interrupting device is slow and multiple interrupts cannot be nested (an interrupt cannot be accepted before servicing of the current one is finished).

9.2.2 Single-Level, Daisy Chain Interrupt Structure

In this structure there is one request line which is shared among all interrupting devices. The interrupting devices are interconnected in such a way that the processor may interrogate them automatically in order to determine the active one. When the processor recognizes an interrupt request, it sends out an acknowledge (ACK) signal. The acknowledge signal is passed from one device to the next (see Fig. 9.3a) until the interrupting device. At this point the ACK signal is captured and prevented from propagating further down the chain. The interrupting device then sends an address to the Data Bus, together with a synchronizing (SYNC) pulse, that is used to inform the processor that the interrupt service address of the highest priority device requesting service is on the Data Bus (see Fig. 9.3b).

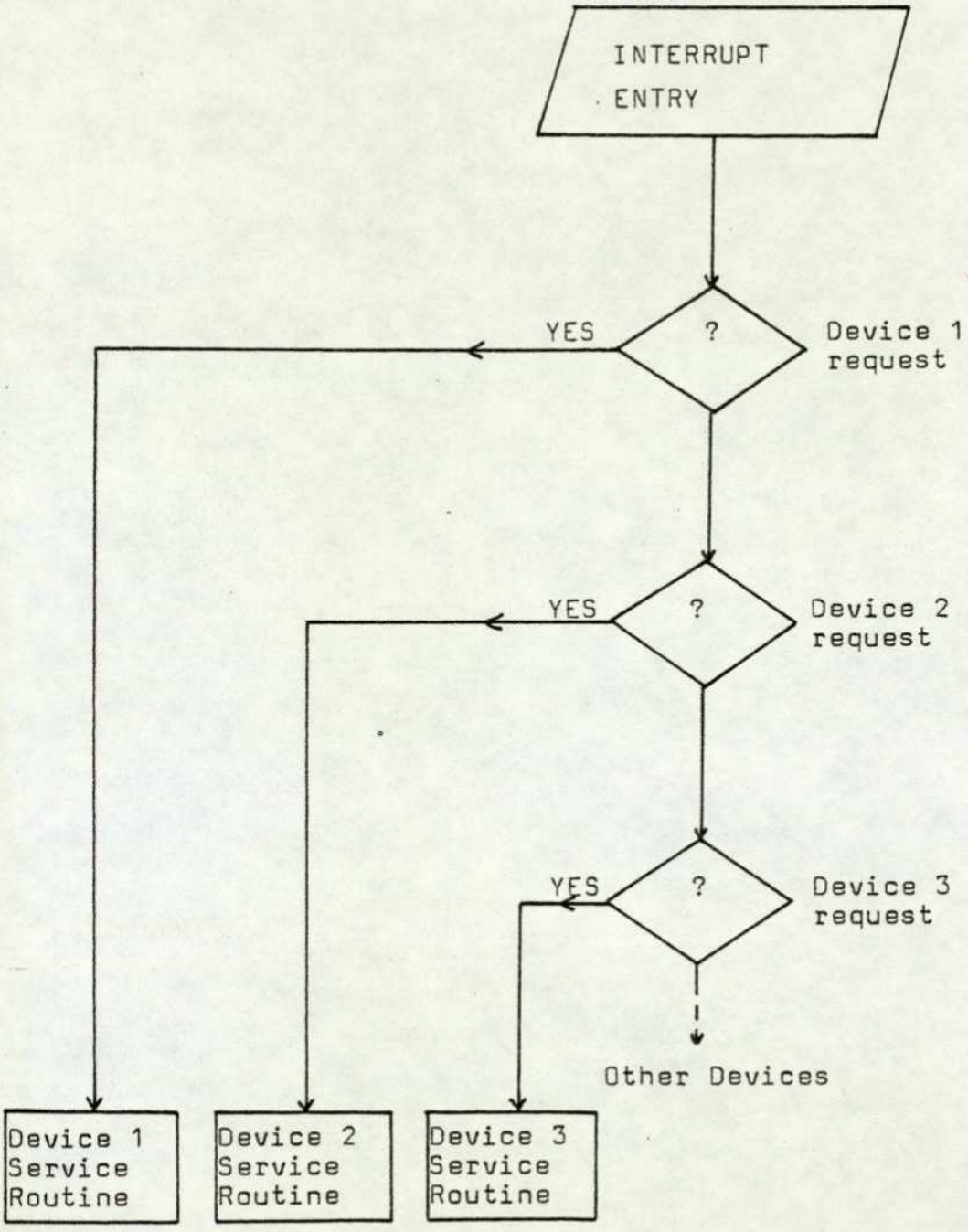
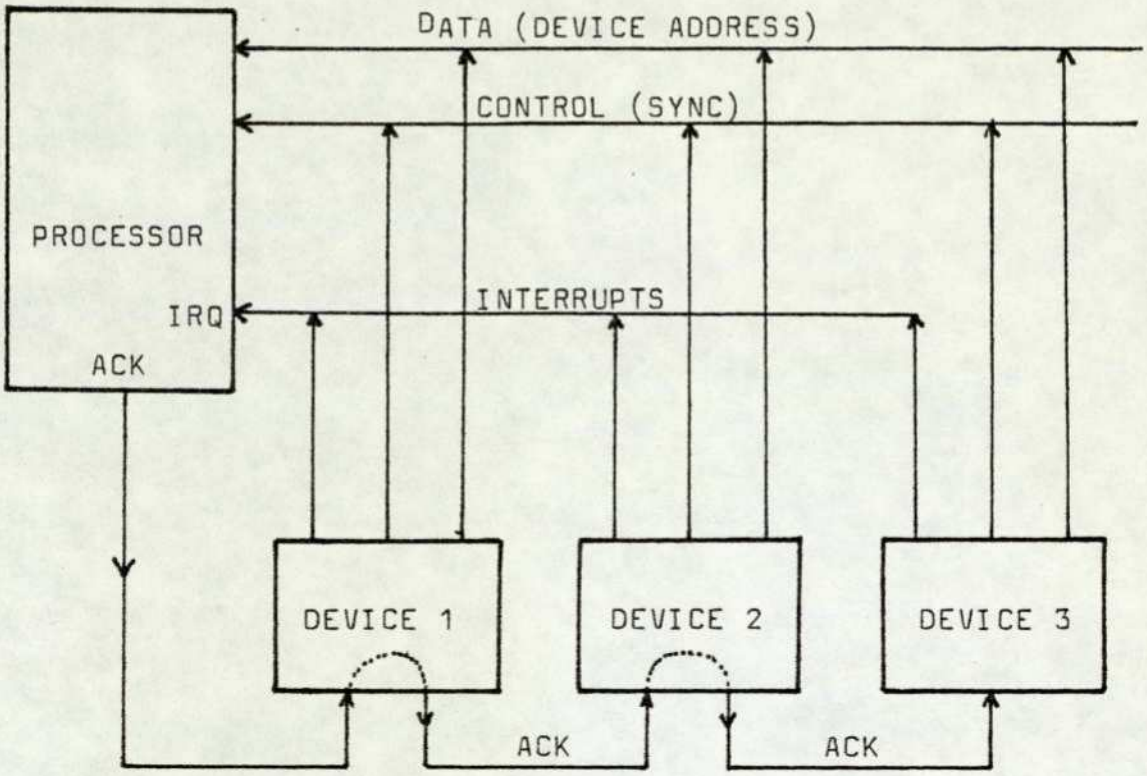


Figure 9.2b Single-level multiple poll interrupt servicing .



a.

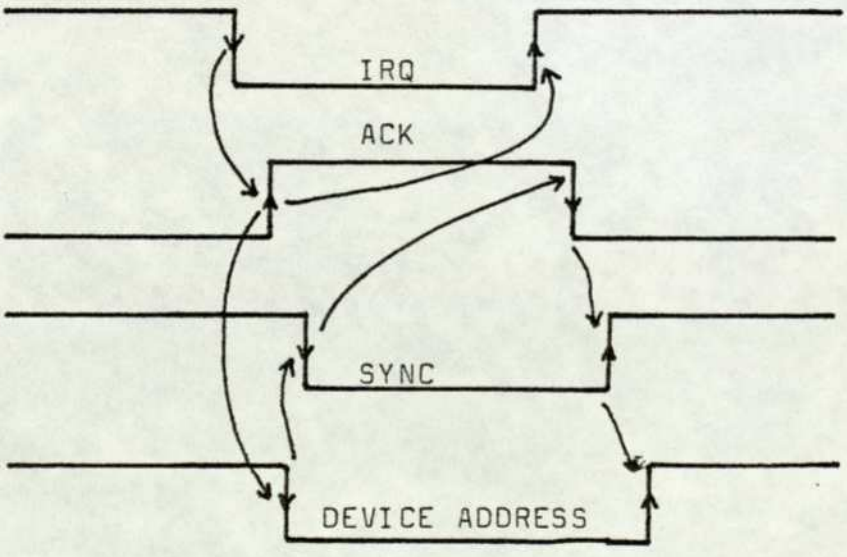


Figure 9.3 a. Single-level, daisy chain interrupt structure .
b. Timing diagram .

The daisy chain structure allows a rapid mechanism for identifying the highest priority interrupting device when several interrrupt requests are issued concurrently. However the major disadvantage of the daisy chain structure is the delay caused as the ACK signal proceeds down the daisy chain. Another disadvantage of this structure is that the priority level of the interrupting devices is fixed.

9.2.3 Multilevel Interrupt Structure

This structure features separate external request lines dedicated to each interrupting device. Consequently, the processor does not have to interrogate the individual devices to identify the highest priority device requesting service. This mechanism uses two registers to service interrupts. The interrupt request register (IRR) and the interrupt mask register (IMR) (see Fig. 9.4). For each bit in the IRR, there is a corresponding bit in the IMR and each bit of the IRR is dedicated to a specific device. A device will interrupt the processor only if its corresponding bits in the IRR and IMR are reset. If any IMR bits are set then interrupts from the corresponding channels of the IRR will be masked out. The priority encoder generates an interrupt vector which is used to determine the unique service routine of the interrupting device. Because the interrupt requests from any device can be masked, the multilevel interrupt structure allows

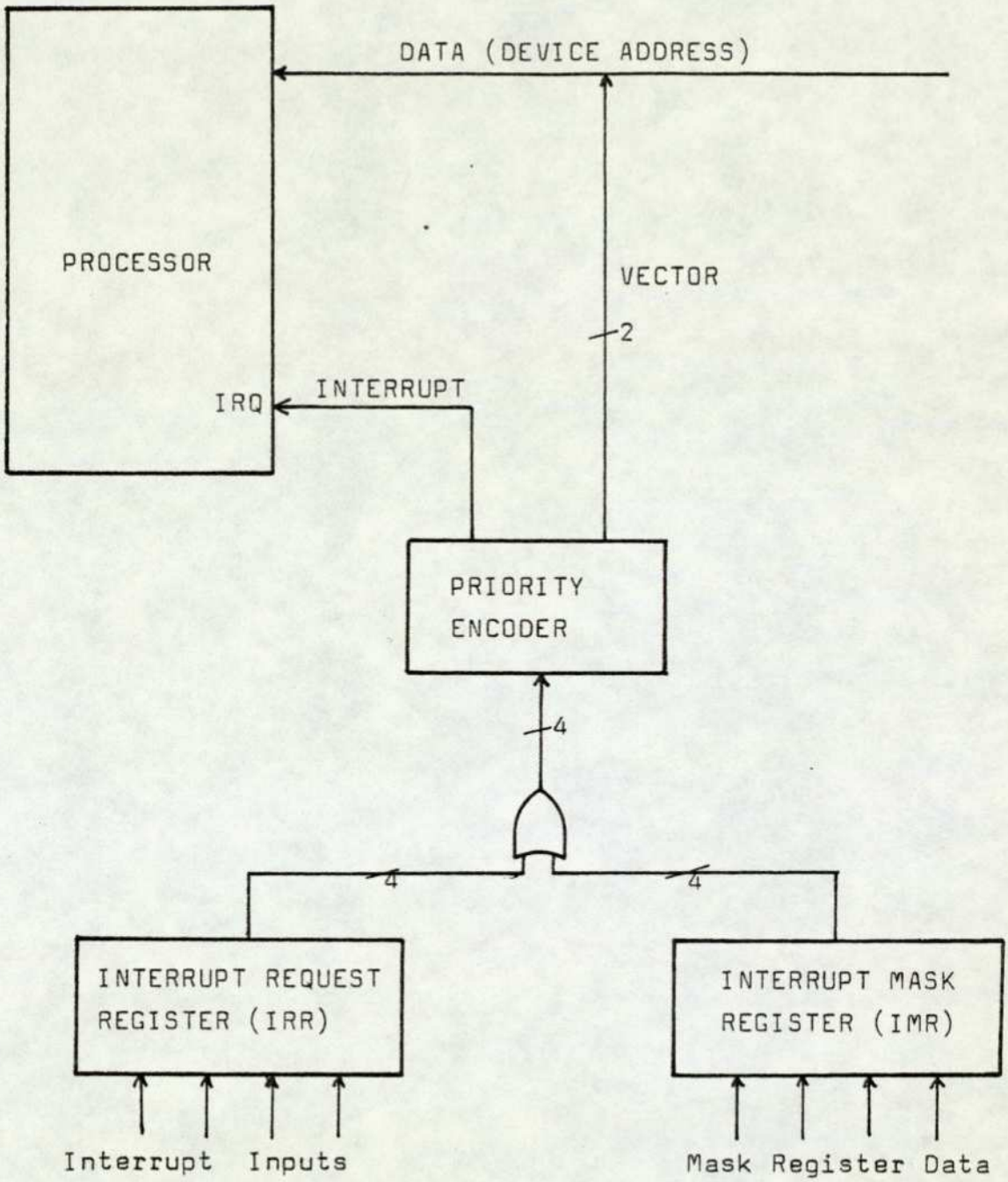


Figure 9.4 Multilevel interrupt structure .

dynamic priority reallocation of the interrupting devices. This structure also provides interrupt nesting capability such that once a particular device interrupt service routine has begun, all interrupts from lower priority devices can be masked and only higher priority requests can be enabled.

The response time of a processor using the multilevel interrupt structure is very quick and the software complexity of the interrupt service routine is minimal.

9.2.4 Multilevel Daisy Chain Interrupt Structure

This structure combines the Single-level daisy chain with the multilevel interrupt structure (see Fig. 9.5). For each priority level there is an interrupt acknowledge (ACK) signal and a mask bit. If the mask bit for a given level is reset, interrupt requests from all devices connected to that level will be enabled. However, individual devices at a given priority level may not be enabled or masked out.

In this structure, interrupt nesting between different priority levels is possible. But interrupts from the individual devices at a particular priority level cannot be nested.

9.3 Design of the Interrupt Control Unit (ICU)

Interrupt Control Unit is one of the most complicated

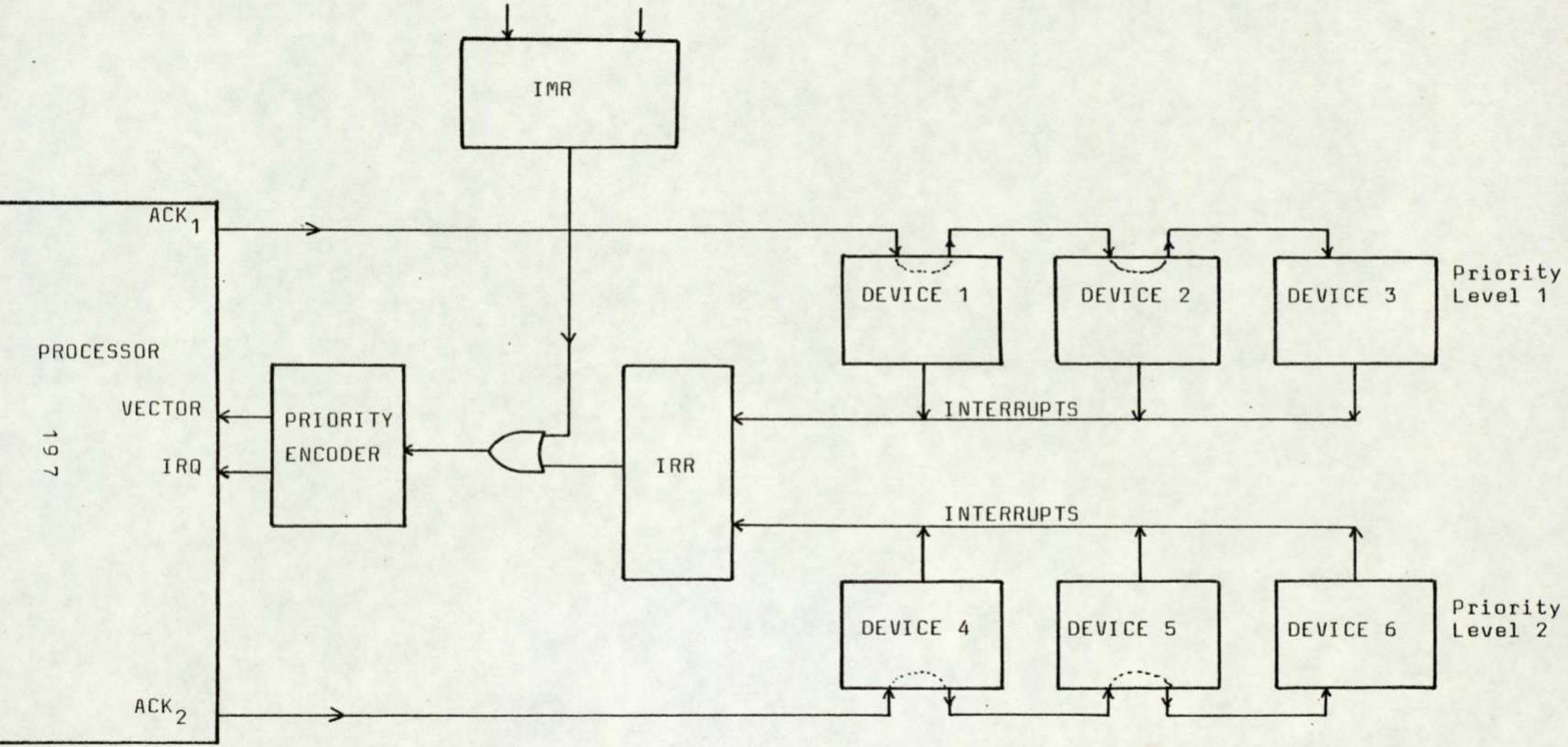


Figure 9.5 Multilevel daisy chain interrupt structure (The SYNC signal and the Data/Control lines not shown) .

subsystems of the microcomputer. It is complicated from the conceptualization, design and implementation point of views.

The block diagram of the ICU is shown in Fig. 9.6. The ICU accepts seven external interrupts and one internal (Timer) interrupt. Interrupts are input to the ICU on the HIGH-to-LOW transition of any interrupt input. The Interrupt Latches hold the interrupt inputs and they act as negative pulse catchers. A HIGH-to-LOW transition on any interrupt input forces the corresponding latch to a logic LOW level.

The Mask Registers (see Fig. 9.6) hold the eight mask bits associated with the eight interrupt levels. The registers are loaded from the higher byte of the Data Bus. The eight outputs of the Mask Register are logical OR'd with the outputs of the corresponding Interrupt Latches so that a HIGH level on any Mask Register bit maskes out the requests from the corresponding interrupt input.

The truth table of the Priority Encoder is as follows:-

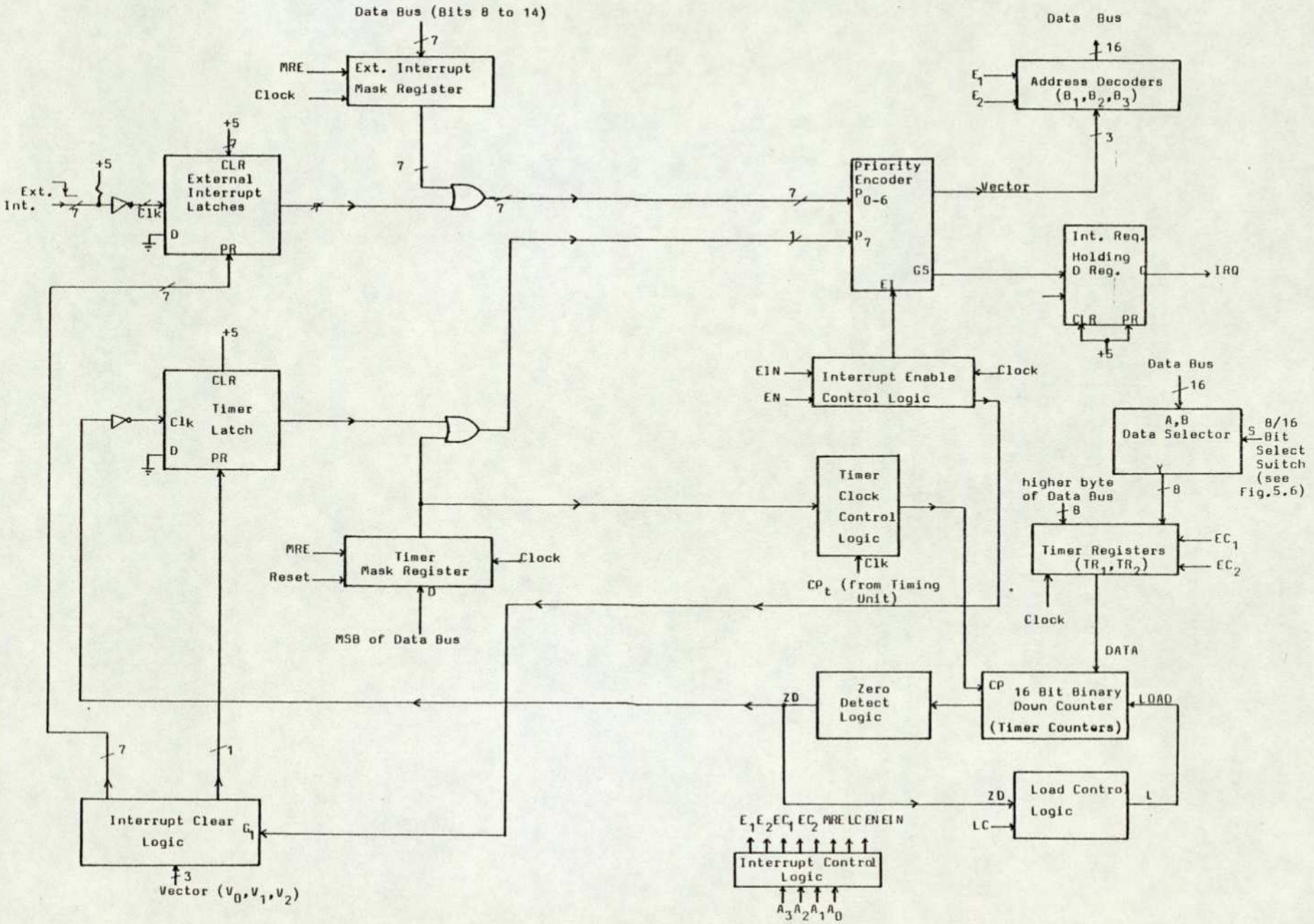


Figure 9.6 Block diagram of the ICU

	INPUTS								OUTPUTS				INTERRUPT TO CCU
EI	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	V_2	V_1	V_0	GS	
1	X	X	X	X	X	X	X	X	1	1	1	1	No request
0	1	1	1	1	1	1	1	1	1	1	1	1	No request
0	X	X	X	X	X	X	X	0	0	0	0	0	Request from P_7
0	X	X	X	X	X	X	0	1	0	0	1	0	Request from P_6
0	X	X	X	X	X	0	1	1	0	1	0	0	Request from P_5
0	X	X	X	X	0	1	1	1	0	1	1	0	Request from P_4
0	X	X	X	0	1	1	1	1	1	0	0	0	Request from P_3
0	X	X	0	1	1	1	1	1	1	0	1	0	Request from P_2
0	X	0	1	1	1	1	1	1	1	1	0	0	Request from P_1
0	0	1	1	1	1	1	1	1	1	1	1	0	Request from P_0

P_7 = Highest priority, P_0 = lowest priority

If interrupts are enabled (EI = LOW) and service is required (GS = LOW), the Priority Encoder determines the highest priority, non-masked interrupt input and forms a three-bit binary coded interrupt vector (V_0, V_1, V_2). This vector is decoded by the Address Decoder (see Fig. 9.6.) whose output points to one of eight preassigned locations in the main memory (RAM), each location contaning the first addresses of the eight interrupt service routines. At the same time, an interrupt request (IRQ = LOW) is sent to the Computer Control Unit (CCU). When the CCU is

ready to service an interrupt, it will execute a special set of microinstructions to initiate the servicing of the interrupt. The first microinstruction will save the contents of the PC in the stack and then it will load the PC with the address which is at the memory location pointed to by the Address Decoder. Thus, the next macroinstruction to be executed will be the first macroinstruction in the service routine of the interrupting device. The CCU then disables the interrupt facilities (EI = HIGH) so that any other interrupt requests will not be recognized. This is necessary so that the processor cannot be interrupted during the service of a particular interrupt (if interrupt nesting is to be allowed, the interrupt facilities can be enabled by software). The Interrupt Latch of the interrupting input is then forced HIGH automatically by the Interrupt Clear Logic (see Fig. 9.6). If this feature was not incorporated then the processor would be interrupted continously whenever the interrupt facilities were reenabled (EI = LOW).

The Timer Interrupt generates interrupt requests at fixed intervals of time. This is useful when it is required to synchronise the input-output operations to 'real-time'. The Timer Interrupt basically consists of a 16 bit binary down counter which can be loaded from the Data Bus via the Timer Registers (see Fig. 9.6). If the Timer Mask Register is enabled, the counters will count down at each clock pulse. When the contents of the counters are all zero, the Zero

Detect Logic will make a LOW-to-HIGH transition thus forcing the output of the Timer Latch to logic LOW. If the interrupt facilities are enabled (EI = LOW), an interrupt request will be presented to the CCU by the Timer Interrupt. At the same time, the 16 bit counters will be re-loaded so that the next Timer Interrupt sequence can be initiated. Following this load operation, the Timer Latch will be forced HIGH automatically by the Interrupt Clear Logic. The Timer Interrupt was assigned the highest priority between the eight interrupt inputs.

The sequence of events during an external or Timer Interrupt are summarized in Fig. 9.7.

9.3.1 Interrupt Latches

The External and Timer Interrupt Latches were designed by using 'D' type flip-flops (SN7474)and inverters (SN7404). The truth table of the SN7474 is as follows:-

INPUTS				OUTPUTS	
PR	CLR	CLK	D	Q	$\overline{Q}$
0	1	X	X	1	0
1	0	X	X	0	1
0	0	X	X	Not Stable	
1	1	↑	1	1	0
1	1	↑	0	0	1
1	1	0	X	No change	

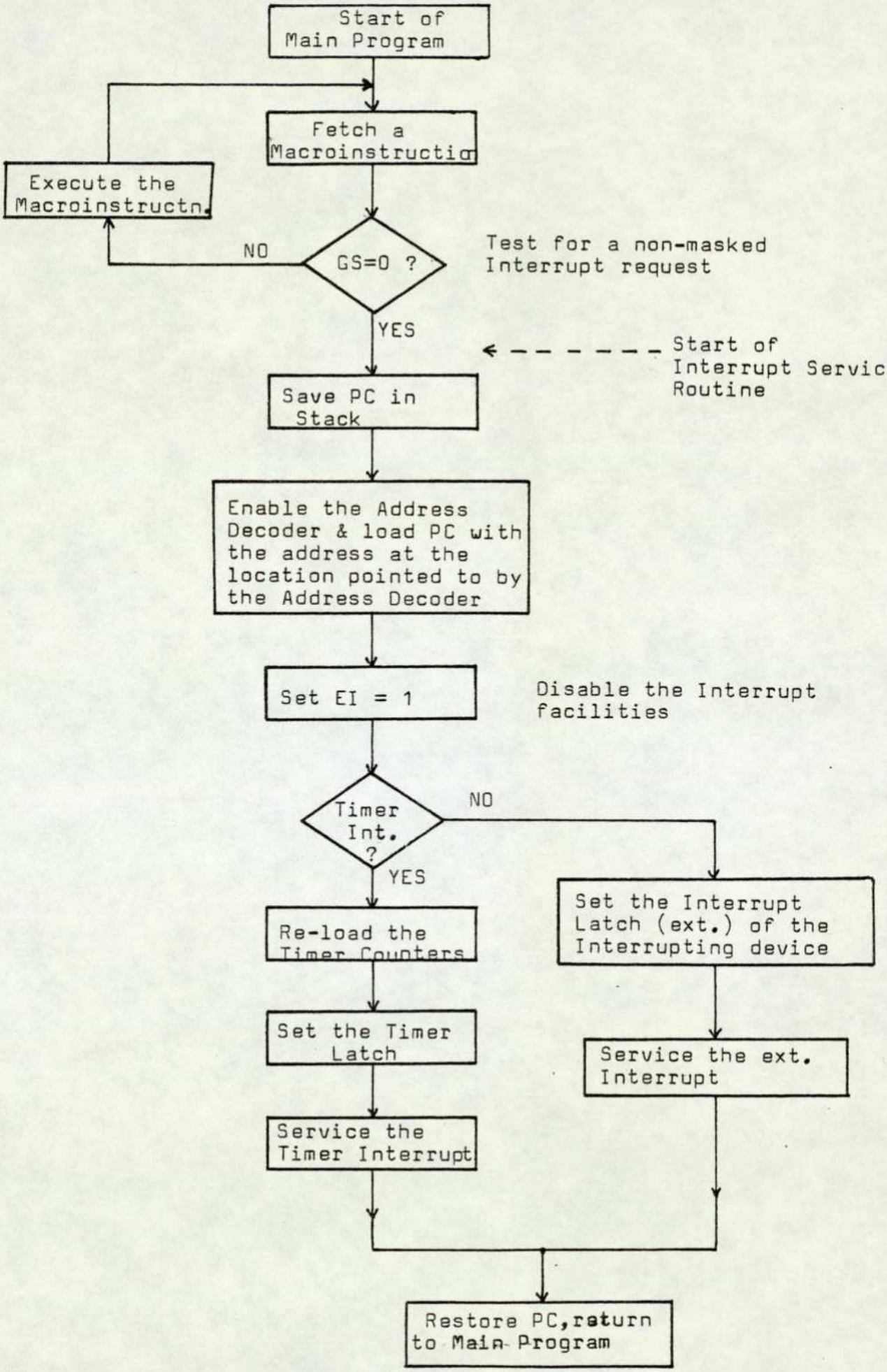


Figure 9.7 Sequence of events during an external or Timer Interrupt

The 'D' inputs of the flip-flop were connected to a fixed LOW level (see Fig. 9.6). Similarly, the clear (CLR) inputs were tied to a fixed HIGH level. The preset (PR) inputs were derived from the Interrupt Clear Logic and during normal operation these inputs are at logic HIGH. The interrupt inputs were inverted and sent to the clock (CLK) inputs of the flip-flops so that a HIGH-to-LOW transition of any interrupt input forces the corresponding 'Q' output to a logic LOW. The operation of the Interrupt Latches are illustrated in the following table:-

	Latch Inputs	Latch Outputs	Interrupt Function
PR	Interrupt Inputs	Q	
1	0	No change	No request
1	1	No change	No request
1	↓	0	Request
0	X	1	Set the latch HIGH

9.3.2 Mask Registers

The External Interrupt Mask Register was designed by using an Am74LS273 type eight bit register. The seven inputs of this register were connected to the Data Bus (bit 8 to bit 14 of the Data Bus). Data is loaded into

the register on the LOW-to-HIGH transition of the clock when the control input MRE is LOW. When MRE is HIGH the register retains its present data independent of the clock. The seven outputs of this register were logical OR'd with the outputs of the External Interrupt Latches and were presented to the Priority Encoder (P_0 to P_6).

The Timer Mask Register was designed by using a 'D' type flip-flop. The 'D' input of the flip-flop was connected to the MSB(Bit 15) of the Data Bus. The clear input was tied to a logic HIGH level. The preset input of the flip-flop was derived from the Timing Unit of the microcomputer such that during the system initialization routine, this input is forced temporarily to a LOW level to mask the Timer interrupt requests. The clock input of the flip-flop was controlled such athat when MRE is LOW, the flip-flop is loaded with new data from the MSB of the Data Bus on the LOW-to-HIGH transition of the system clock. When MRE is HIGH the Timer Mask Register retains its present data independent of the clock. The logic operations describing the control of the clock input were derived from the following truth table:- (CLK = System clock, CK = Clock input of the flip-flop).

MRE	CLK	CK
0	0	0
0	1	1
1	X	0

Hence, CK = $\overline{MRE}$.CLK

The logic diagram of the Timer Mask Register is shown in Figure 9.8 .

9.3.3 Priority Encoder

The Priority Encoder utilizes a SN74148 type device whose truth table is as illustrated in Chapter 9.3. Whenever a non-masked interrupt request is present, the GS output of the Priority Encoder is forced LOW. The Interrupt Request Holding Register (see Fig. 9.6) stores and presents this request (IRQ = LOW) to the Computer Control Unit.

The EI input of the Priority Encoder is controlled by the Interrupt Enable Control Logic which enables or disables the interrupt requests to the CCU. The interrupt Enable Control Logic was designed by using two 'D' type flip-flops (SN7474) as shown in Figure 9.9. The clock to the first flip-flop was controlled by the EIN control bit such that when EIN is HIGH, new data is loaded into the flip-flop on the LOW-to-HIGH transition of the clock. When EIN is LOW the flip-flop retains its data independent of the system clock. The Boolean operations describing the control of this clock input were derived from the following truth table:- (CLK = System clock, CK = Clock input of the flip-flop)

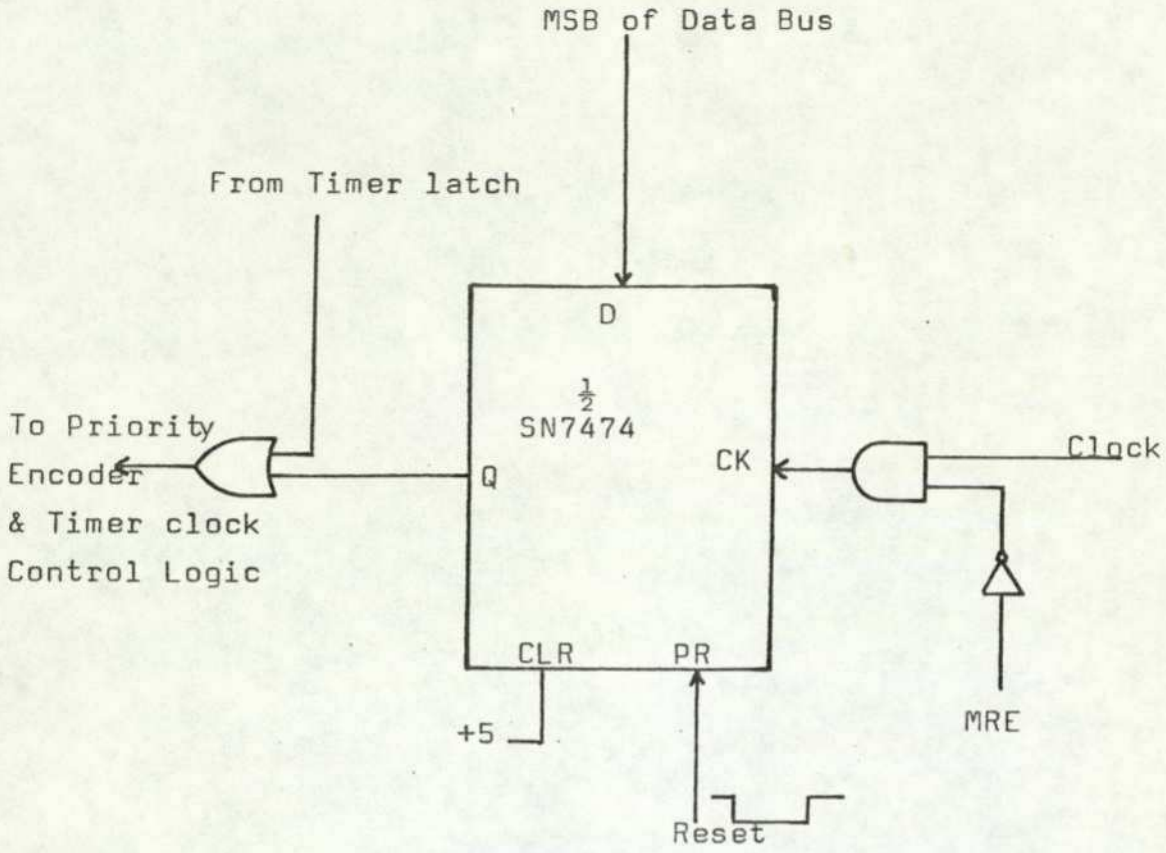


Figure 9.8 Logic diagram of the Timer Mask Register .

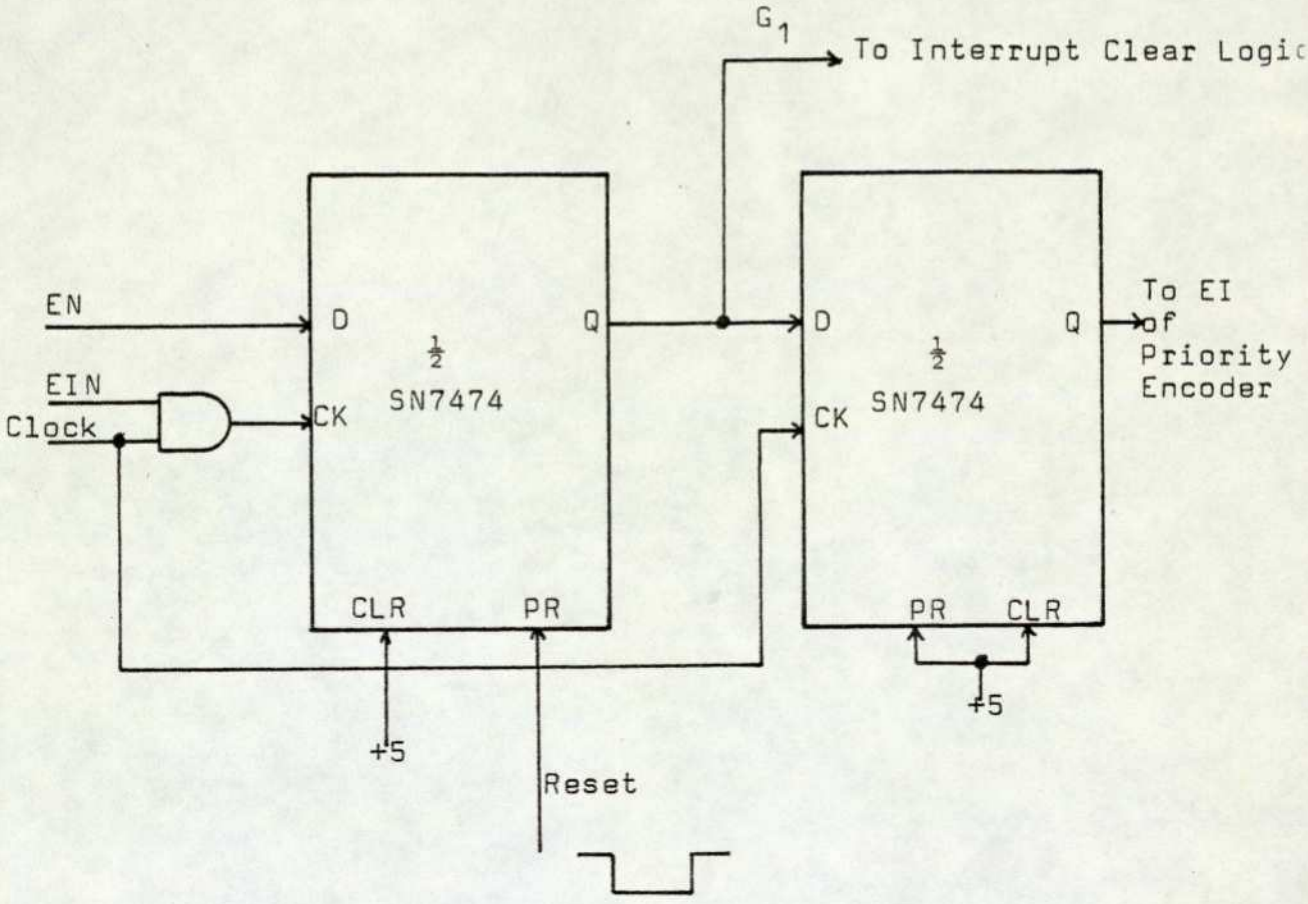


Figure 9.9 Logic diagram of the Interrupt Enable Control Logic

EIN	CLK	CK
1	0	0
1	1	1
0	X	0

Hence, CK = EIN.CLK

9.3.4 Address Decoder

The Address Decoder was designed by using three-state buffers (Am74LS241). These buffers were connected in such a way that (see Fig.9.10) when they are enabled they provide eight different 16 bit addresses corresponding to the vector outputs of the Priority Encoder.

The truth table of the Address Decoder is shown below (when it is enabled):-

Interrupt Input to Priority Encoder	Inputs of the Address Decoder V_2	V_1	V_o	Address Provided(Hex.) (output of the Address Decoder)
P_7 (Timer)	0	0	0	0400
P_6	0	0	1	0401
P_5	0	1	0	0402
P_4	0	1	1	0403
P_3	1	0	0	0404
P_2	1	0	1	0405
P_1	1	1	0	0406
P_o	1	1	1	0407

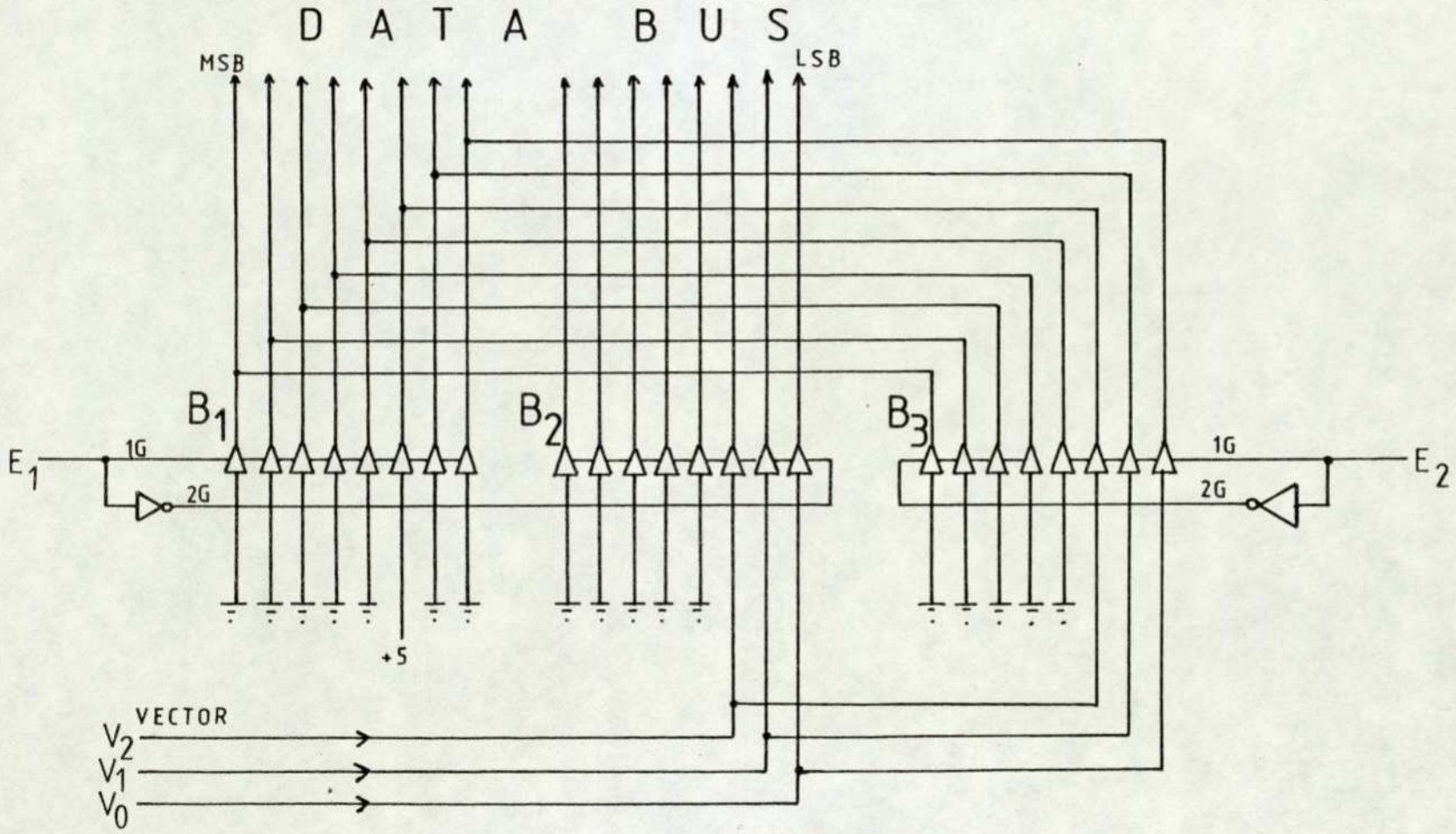


Figure 9.10 The address decoder was designed by using Am74LS241 type three-state buffers

As an example, suppose that a non-masked interrupt request is received from the interrupt input P_5. Then, the macroinstruction at the memory location pointed to by the contents of address 0402 will be executed as the first macroinstruction of the interrupt service routine.

If operation with 16 bits of data length is chosen then the buffers B_1 and B_2 (see Fig. 9.10) must be enabled whenever it is required to send a 16 bit interrupt address to the Program Control Unit. During 16 bits of operation the buffer B_3 must be forced to a high impedence state.

If operation with 8 bits of data length (higher byte of the Data Bus) is required then two steps are required to send a 16 bit interrupt address to the Program Control Unit. During the first step, the buffer B_1 must be enabled to send the higher byte of the interrupt address to the PCU. During the second step, the buffer B_3 must be enabled to send the lower byte of the interrupt address to the PCU. The overall 16 bit interrupt address can thenbe combined by the PCU as described in Chapter 7.2.4.

Although the Address Decoder contains 3 buffers, only 2 control bits were used to control them. The buffers B_1 and B_2 were controlled together by the control bit E_1 and the buffer B_3 was controlled by the control bit E_2 as described in the following table:-

E_1	E_2	Buffers enabled/disabled
0	0	Not allowed
0	1	Enable the buffers B_1 and B_2
1	0	Enable the buffer B_3
1	1	Disable (three-state) all the buffers

9.3.5 Timer Registers

The Timer Registers (see Fig. 9.6) provide data to the 16 bit Timer Counters. They were designed by using two Am74LS377 type eight-bit registers with register enable (E) control inputs. When E is LOW, new data is loaded into the registers on the LOW-to-HIGH transition of the system clock and when E is HIGH, the registers retain their data independent of the clock.

The logic diagram of the Timer Registers is shown in Fig. 9.11. Register TR_1 is connected to the higher byte of the Data Bus and this register is enabled by the control bit EC_1. The register TR_2 (enabled by EC_2) receives data from the Data Bus via two four-bit data selectors (Am74LS157). These data selectors are controlled from the 8/16 bit select switch S_1 (see Chapter 5.1.2)and their function is to route the correct data from the Data Bus to register TR_2. During 8 bits of operation, only the higher byte of the Data Bus is used. Consequently, two steps are required to load the

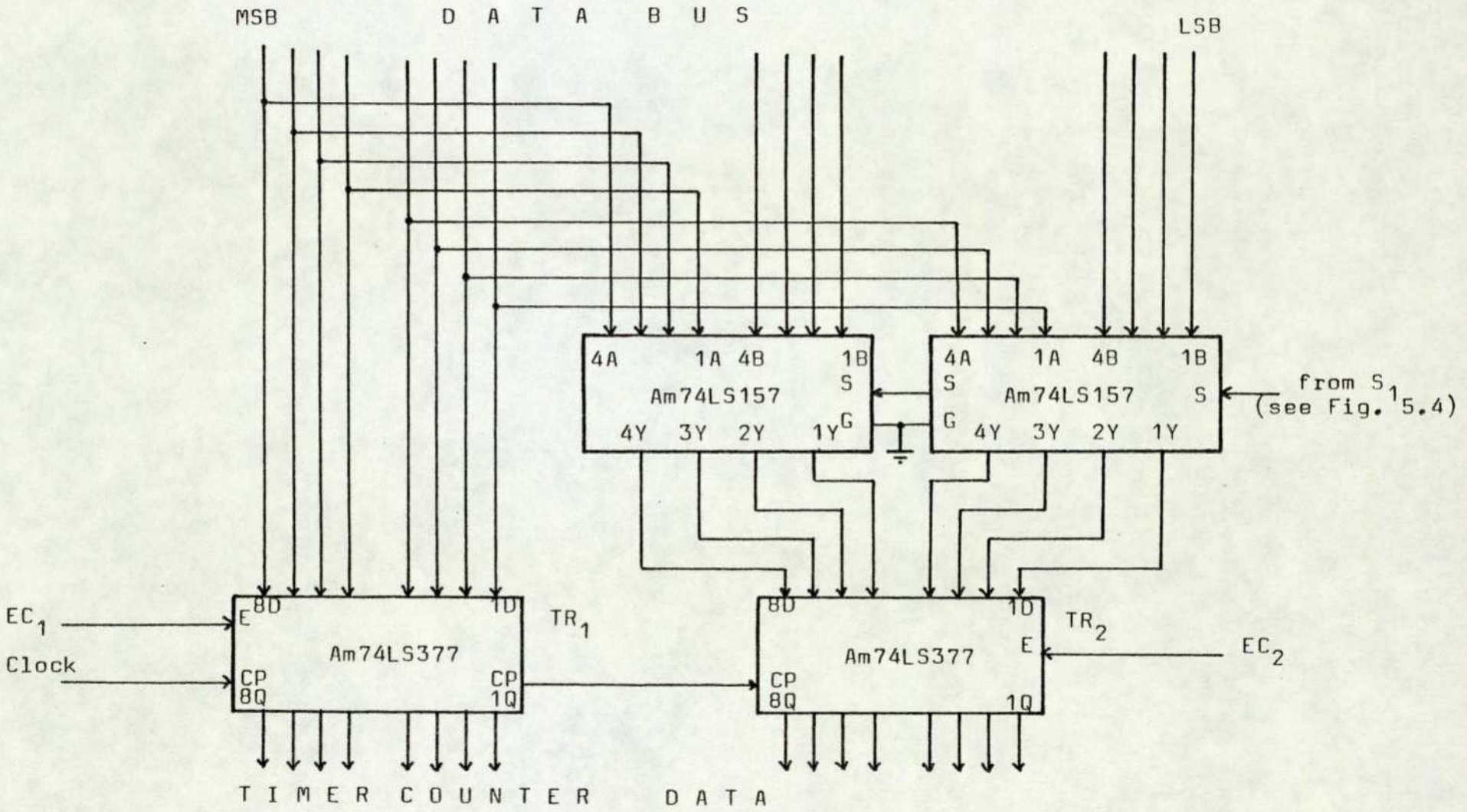


Figure 9.11 The Timer Registers were designed by using Am74LS377 type eight-bit registers

Timer registers with 16 bits of counter data. During the first step, register TR_1 must be enabled so that higher byte of the counter data can be loaded into TR_1. During the second step, register TR_1 is disabled and TR_2 is enabled. The lower byte of the counter data (contained in the higher byte of the Data Bus) can then be routed to register TR_2.

The two control bits EC_1 and EC_2 control the loading of the Timer Registers as described in the following table:- (CLK = System Clock).

EC_1	EC_2	CLK	TR_1	TR_2
0	0		Load	Load
0	1		Load	No change
1	0		No change	Load
1	1		No change	No change

9.3.6 Timer Counters

The Timer Counters were designed by using four four-bit synchronous binary counters (Am74LS191), connected by the carry look ahead scheme (see Fig. 9.12) to provide a 16 bit high speed binary counter. The counters were organized to operate in the count-down mode.

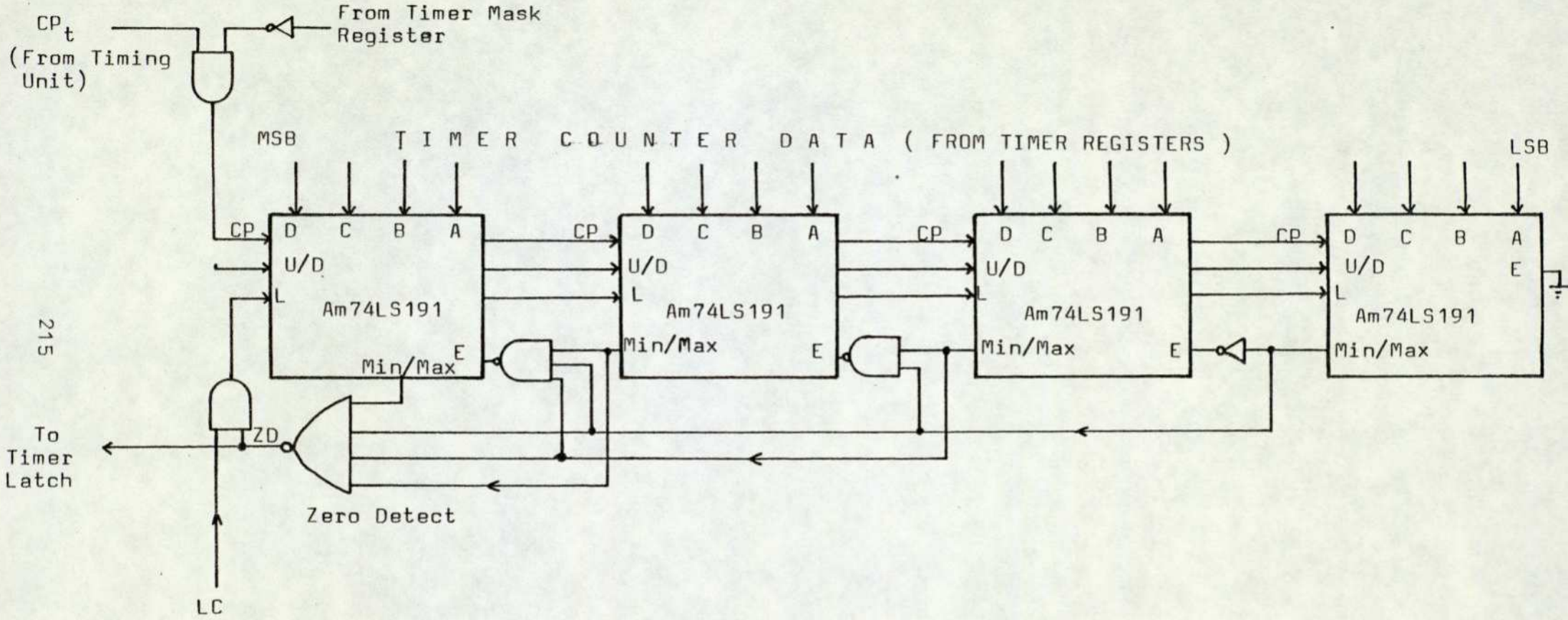


Figure 9.12 Logic diagram of the Timer Counters with the Zero Detect & Timer Clock Control Logic .

The load-input (L) of the counters is used to asynchronously enter new data into the counters. When L is LOW, the counters are loaded from the Timer Registers and when L is HIGH, the counters are in the 'count mode'.

Whenever the counters are in the underflow (outputs all LOW) or overflow (outputs all HIGH) state, the Min./Max. outputs are foced HIGH. The Zero Detect Logic is a four input NAND gate (SN7420) whose inputs were derived from the Min./Max. outputs of the four counters. Hence, when all the counters are in the underflow (or overflow) state, the Zero Detect Logic makes a HIGH-to-LOW transition thus generating an interrupt request. If the interrupt facilities are enabled (EI of the Priority Encoder LOW) and the Timer Mask is reset, this interrupt request will be accepted and serviced by the CCU.

Whenever a Timer interrupt request is generated it is necessary to re-load the Timer Counters to prepare for the next counting sequence. The Load Control Logic performs this task as described in the following table (LC = Control bit of the Load Control Logic, ZD = output of the Zero Detect Logic, L = Load input of the counters):-

ZD	LC	L	
0	0	0	Load counters by hardware
0	1	0	Load counters by hardware
1	0	0	Load counters by software
1	1	1	No loading

Hence, the Load Control Logic was implemented as a two input AND gate:-

$$\overline{L} = ZD.LC$$

The clock to the Timer Counters was derived from the Timer Clock Control Logic such that whenever the Timer Mask Register is set, the clock is disabled from the Timer Counters.

If the Timer Counters are loaded with a 16 bit data K, then the Timer Interrupts will be generated at every Kxtp seconds, where tp is the system clock period.

9.3.7 Interrupt Clear Logic

When an interrupt request is presented to the ICU, the interrupt latch of the corresponding input is forced LOW. The function of the Interrupt Clear Logic is to set (HIGH) the interrupt latch of the highest priority non-masked interrupting input. If this facility was not incorporated then the processor would be interrupted every time the interrupt

facilities were re-enabled.

The Interrupt Clear Logic was designed by using a 1 of 8 multiplexer (Am74LS138) which is driven by the Vector outputs of the Priority Encoder (see Figure 9.13). The seven outputs (corresponding to the external interrupt vectors) of this multiplexer were AND ed with the Mask Register control bit,MRE, and they were connected to the 'Preset' inputs of the seven External Interrupt Latches. The output of the multiplexer corresponding to the Timer vector was AND ed with the control bit,LC, of the Load Control Logic and it was connected to the 'Preset' input of the Timer Latch.

The operation of the Interrupt Clear Logic is such that if the interrupt facilities are enabled and there are no interrupt requests then the 'Preset' inputs of all the eight Interrupt Latches are forced HIGH (this is the situation during normal operation of the Interrupt Latches). If on the other hand the interrupt facilities are enabled and an interrupt request is present then the first task of the CCU will be to disable the interrupt requests (EI=HIGH) and read the address of the interrupting device. At the same time one of eight outputs of the Interrupt Clear Logic will be forced LOW, depending upon the 3-bit Vector. When the 'Preset' is LOW, the Interrupt Latch of the interrupting device will be set HIGH (ie. interrupt will be cleared).

Another function of the Interrupt Clear Logic is to set

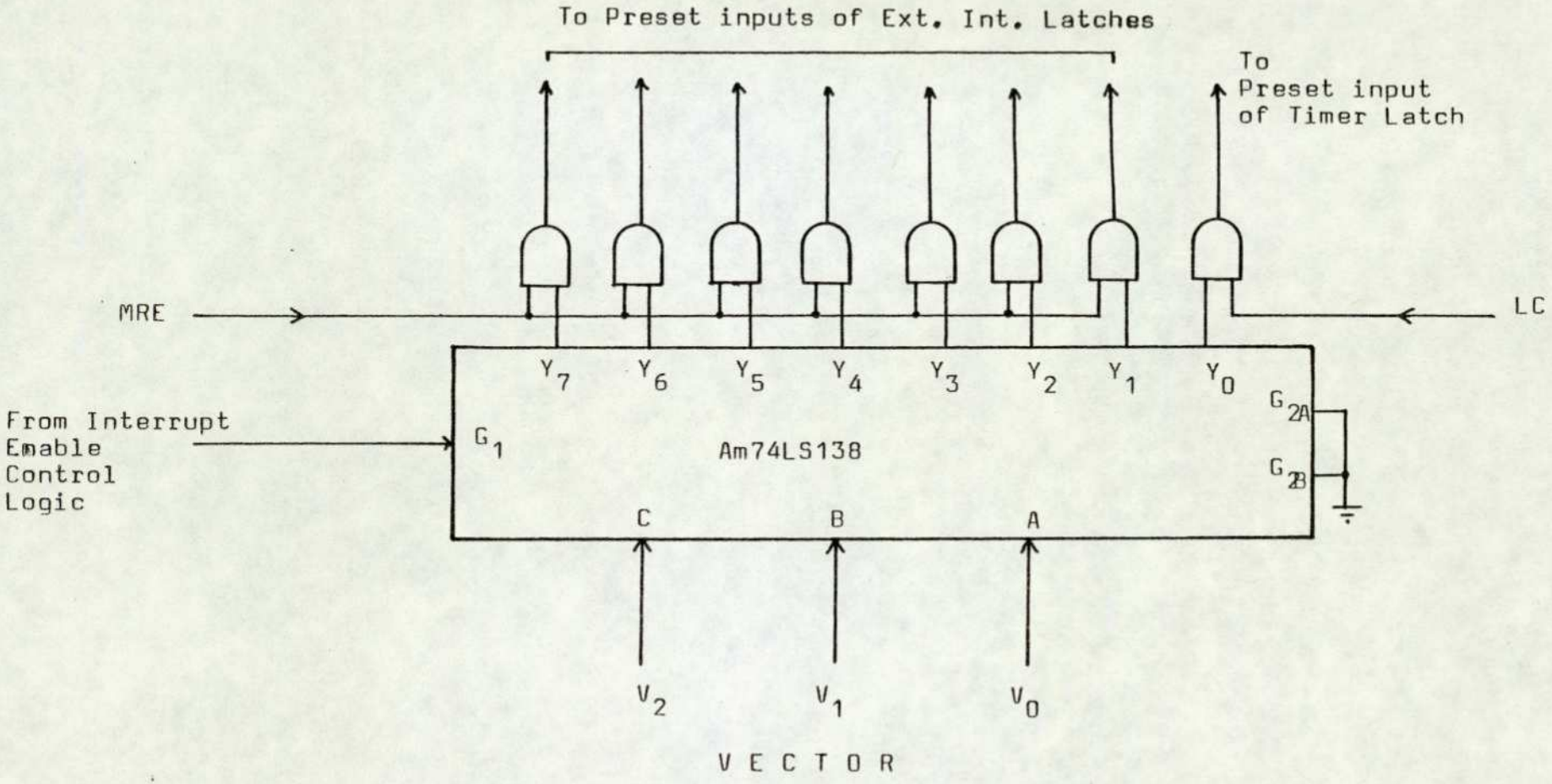


Figure 9.13 Logic diagram of the Interrupt Clear Logic .

the Interrupt Latches under certain other conditions. The seven External Interrupt Latches are set whenever MRE=LOW (ie. whenever the Mask Register is up-dated). Also, the Timer Interrupt Latch is set whenever LC=LOW (ie. whenever the Timer Counters are re-loaded, either by software or hardware) .

9.3.8 Interrupt Control Logic

The Interrupt Control Unit is controlled by eight control bits (E_1, E_2, EC_1, EC_2, MRE, EIN, LC, and EN). The Interrupt Control Logic was designed to minimize the number of control signals required to control the ICU. The truth table of the Interrupt Control Logic is shown in Table 9.1. From this table, the following Boolean operations were derived for the control of each ICU control bits (A_0, A_1, A_2, and A_3 are the inputs of the Interrupt Control Logic) :-

$$E_2 = \overline{\overline{A_3} \cdot A_2 \cdot \overline{A_1} \cdot A_0}$$

$$E_1 = \overline{\overline{A_3} \cdot \overline{A_2} \cdot A_1 \cdot \overline{A_0}}$$

$$EC_1 = \overline{\overline{A_3} \cdot A_2 \cdot \overline{A_1} \cdot \overline{A_0}}$$

$$EC_2 = LC = \overline{\overline{A_3} \cdot A_2 \cdot \overline{A_0}}$$

$$EN = \overline{\overline{A_3} \cdot \overline{A_2} \cdot \overline{A_1} \cdot \overline{A_0}}$$

ICU CONTROL				TO SUBSYSTEMS OF ICU								ICU OPERATION PERFORMED
A_3	A_2	A_1	A_0	E_2	E_1	LC	EC_2	EC_1	EN	EIN	MRE	
0	0	0	0	1	1	1	1	1	0	1	1	Enable the Interrupt facilities
0	0	0	1	1	1	1	1	1	1	1	1	Disable the Interrupt facilities
0	0	1	0	1	0	1	1	1	1	0	1	Enable Address Decoders B_1 and B_2
0	0	1	1	1	1	1	1	1	1	0	0	Load the Mask Registers
0	1	0	0	1	1	0	0	0	1	0	1	Load the Timer Registers TR_1 and TR_2
0	1	0	1	0	1	1	1	1	1	0	1	Enable the Address Decoder B_3
0	1	1	0	1	1	0	0	1	1	0	1	Load the Timer Register TR_2
0	1	1	1	1	1	1	1	1	1	0	1	No change in ICU
1	X	X	X	1	1	1	1	1	1	0	1	No change in ICU

Table 9.1 Truth table of the Interrupt Control Logic .

$$EIN = \overline{A_3} \cdot \overline{A_2} \cdot \overline{A_1}$$

$$MRE = \overline{\overline{A_3} \cdot \overline{A_2} \cdot A_1 \cdot A_0}$$

9.4 Conclusions

In this Chapter, the operation and design of the Interrupt Control Unit has been described. The ICU was designed from MSI logic elements and it provides a highly sophisticated mechanism for handling the interrupts. The ICU can accept interrupts from 7 external sources. Additionally a Timer Interrupt was provided which can interrupt the processor at fixed multiples of the system clock period. The Timer Interrupt could further be improved if the clock rate of the Timer Counters could be changed by software.

The overall logic diagram of the Interrupt Control Unit is shown in Figure 9.14 .

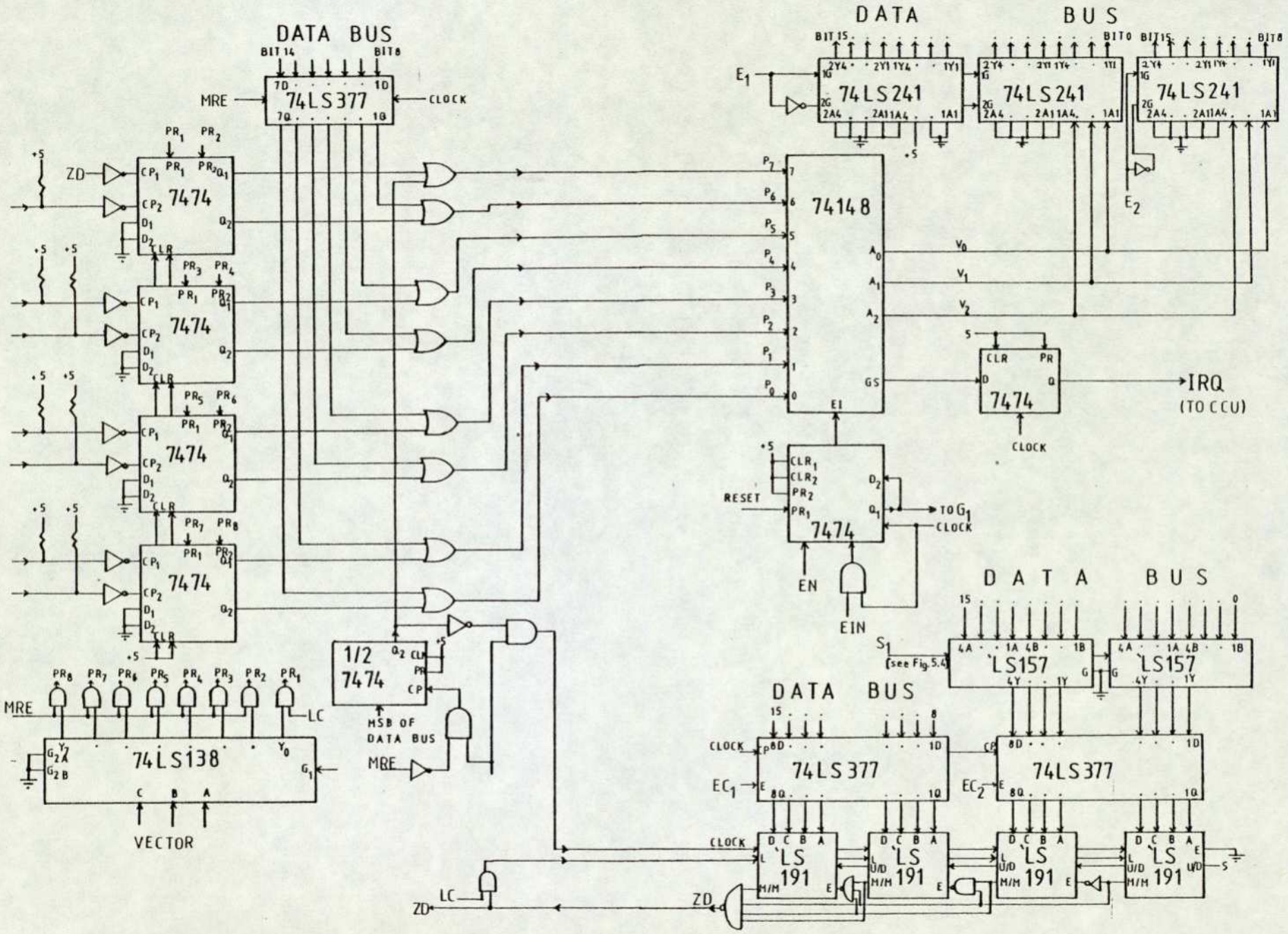

DATA BUS
BIT 14
BIT 8
MRE
74LS377
CLOCK
7D
1D
7Q
1Q
DATA BUS
BIT 15
BIT 0
BIT 15
BIT 8
E1
74LS241
74LS241
74LS241
E2
+5
ZD
7474
74148
P7
P6
P5
P4
P3
P2
P1
P0
A0
A1
A2
V0
V1
V2
GS
EI
S
CLR
PR
D
Q
IRQ
(TO CCU)
CLOCK
RESET
CLR1
CLR2
PR2
PR1
D2
Q1
TO G1
CLOCK
EN
EIN
DATA BUS
S1
(see Fig. 5.4)
LS157
LS157
4A
1A
4B
1B
4Y
1Y
G
1/2
7474
Q2
CLR
PR
CP
MSB OF
DATA BUS
MRE
LC
74LS138
G2A
G2B
G1
C
B
A
VECTOR
DATA BUS
CLOCK
EC1
EC2
74LS377
74LS377
8D
1D
8Q
1Q
LS
191
U/D
M/M
E
LC
ZD
ZD
S

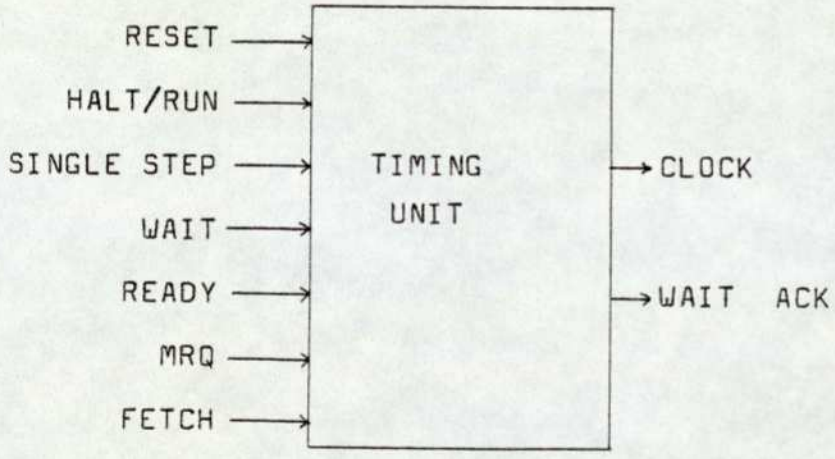
Figure 9.14

CHAPTER 10

10. THE TIMING UNIT

10.1 Introduction

The Timing Unit generates the system clock, initializes the microcomputer and responds to external supervisory timing signals. Although the clock runs continuously, it can be disabled from the microcomputer under the control of the CCU and the external supervisory signals. The black box representation of the Timing Unit is as follows:-



Under the control of the HALT/RUN switch, the processor can be HALTed. In this mode, the clock is disabled from the microcomputer. The processor can only be HALTed during the macroinstruction FETCH cycle. When HALTed, the SINGLE STEP switch can be used to send single clock* pulses to the

* One alternative would be to provide clocks from one FETCH cycle to the next.

microcomputer. This facility is very useful for tracing the execution steps of a macroinstruction. When the HALT/RUN switch is in the RUN mode, the processor resumes its normal operation by receiving continous clock pulses from the Timing Unit. The SINGLE STEP switch is inactive during this mode of operation.

The RESET switch is used to initialize the microcomputer*. When active, it forces the CCU to start execution from address zero of the microprogram memory (see Chapter 12).

The WAIT signal can be used to synchronize the microcomputer with the outside world. This signal is generated by the CCU and when active, it disables the clock from the microcomputer. When the system is in the WAIT mode, the WAIT ACK output signal is LOW. The READY input (pulse sensitive) can be pulsed to re-enable the clock to the microcomputer. Both WAIT and READY inputs are overridden by the HALT/RUN input (when in the HALT mode).

The memory request (MRQ) signal can be used to enable slow memories to be connected to the microcomputer. If this facility is used, the clock period will be increased during the memory access operations (synchronous memory control).

* A more sophisticated initialization sequence may be started by one of three events: application of main power to the system; an operator generated "Master Reset" command; or an error that the system cannot recover from, but can detect.

10.2 Design of the Timing Unit

The logic diagram of the Timing Unit is shown in Fig. 10.1. The clock was generated by using a crystal controlled oscillator (SN74LS124) operating at a frequency of 1 MHz (see Chapter 14 for the selection of this frequency) with 50% duty-cycle.

The RUN/HALT switch (a single-pole double-throw switch) was connected to a S-R flip-flop whose purpose was to eliminate the contact bounce problems associated with the mechanical switches. The truth table describing the operation of this flip-flop is as follows:-

S	R	Q_{SR}	$\overline{Q}_{SR}$	
0	1	1	0	RUN mode
1	0	0	1	HALT mode

The Q_{SR} output of the S-R flip-flop was connected to the K input of a J-K flip-flop (SN74LS112). Similarly, the $\overline{Q}_{SR}$ output was ANDed with the FETCH signal (active HIGH) and connected to the J input of the J-K flip-flop. The operation of the J-K flip-flop is then as follows:-

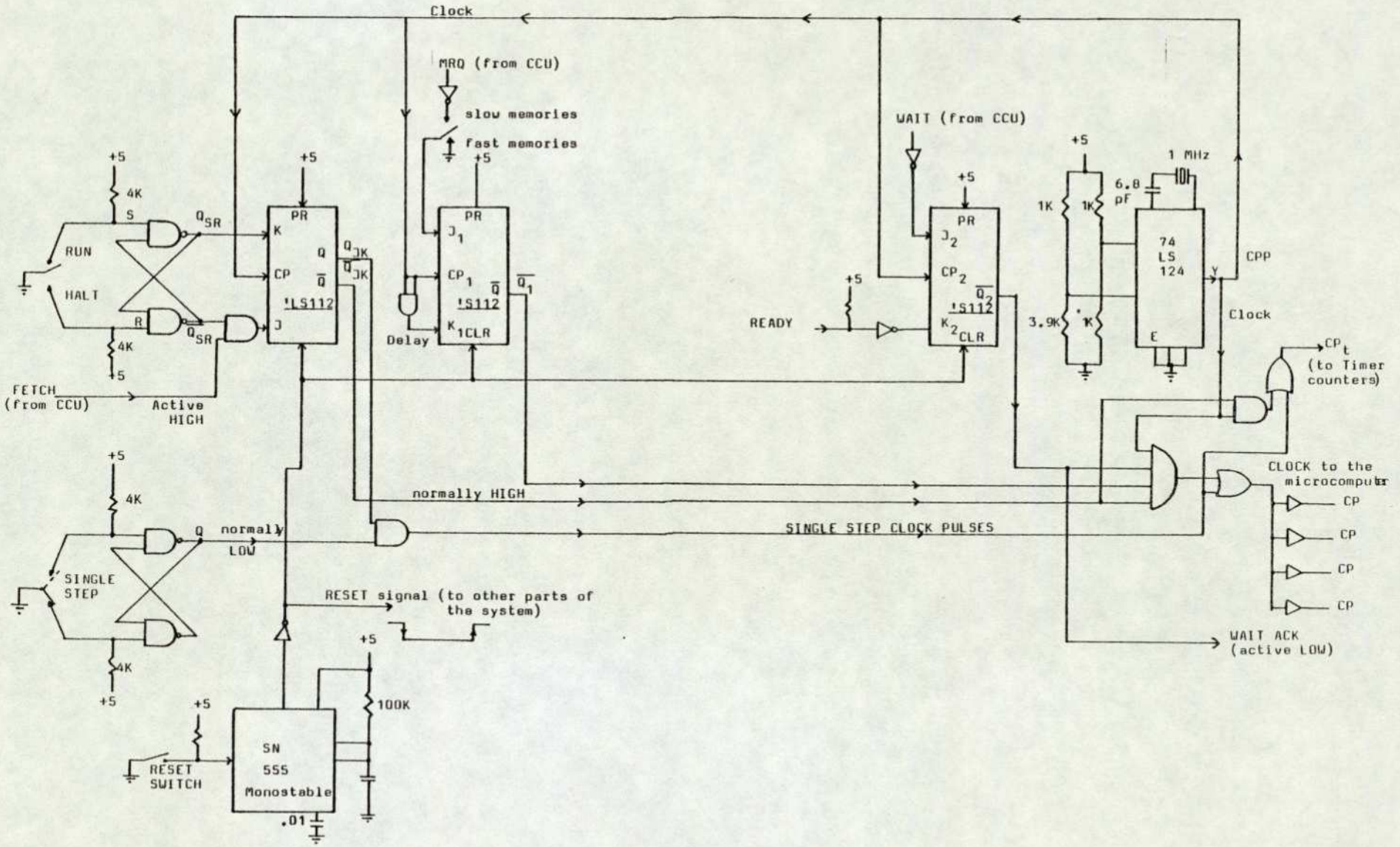


Figure 10.1 Logic diagram of the Timing Unit

S	R	FETCH	Q_{JK}	$\overline{Q}_{JK}$	
0	1	X	0	1	RUN
1	0	0	0	1	RUN
1	0	1	1	0	HALT

During the RUN mode, Q_{JK} is LOW, thus disabling the SINGLE STEP switch (a push-button switch). Also, $\overline{Q}_{JK}$ is HIGH so that the clock is enabled to the microcomputer (see Fig. 10.1). When the RUN/HALT switch is in the HALT position and the CCU is in the macroinstruction fetch cycle (FETCH = HIGH) then $\overline{Q}_{JK}$ output of the J-K flip-flop is forced LOW, thus disabling the clock from the microcomputer. Similarly, Q_{JK} output is forced HIGH to enable the SINGLE STEP switch (the R-S flip-flop at the output of this switch eliminates the contact bounce problems). In this mode of operation, whenever the SINGLE STEP switch is pressed, one clock pulse is sent to the microcomputer.

The WAIT signal (generated by the CCU) was inverted and connected to the J input of a J-K flip-flop (see Fig. 10.1). The READY signal was also inverted and connected to the K input of the same flip-flop . When the WAIT signal is active (WAIT = LOW), the clock is disabled ($\overline{Q}_{JK}$= LOW) from the microcomputer and the WAIT ACK output is forced LOW. When in the WAIT mode, the clock can only be re-enabled by sending a LOW signal to the READY input.

The memory request (MRQ) signal was used to enable slow memories to be connected to the microcomputer. For asynchronous memory access operations (using fast memories), the MRQ signal should be disabled from the Timing Unit (by the memory speed select switch, Fig. 10.1). If synchronous memory access (using slow memories) is required then this signal should be enabled to the Timing Unit. During synchronous memory access operations, the clock period is increased by a factor of two. (see Fig. 10.2). It then follows that during sychronous memory access operations the clock to the Timer Counters of the ICU (see Section 9.3.6) will not be the true clock of 1 MHz. However this problem was avoided by deriving the clock of the Timer Counters (C_{pt}) from the clock generator of the Timing Unit. This clock was also under control of the RUN/HALT and the SINGLE STEP switches (see Fig. 10.1).

The RESET signal was derived by using a monostable multivibrator with a pulse width of 470 ms. When the RESET button is activated the RESET signal is forced LOW for the duration of 470 ms (see Fig. 10.1).

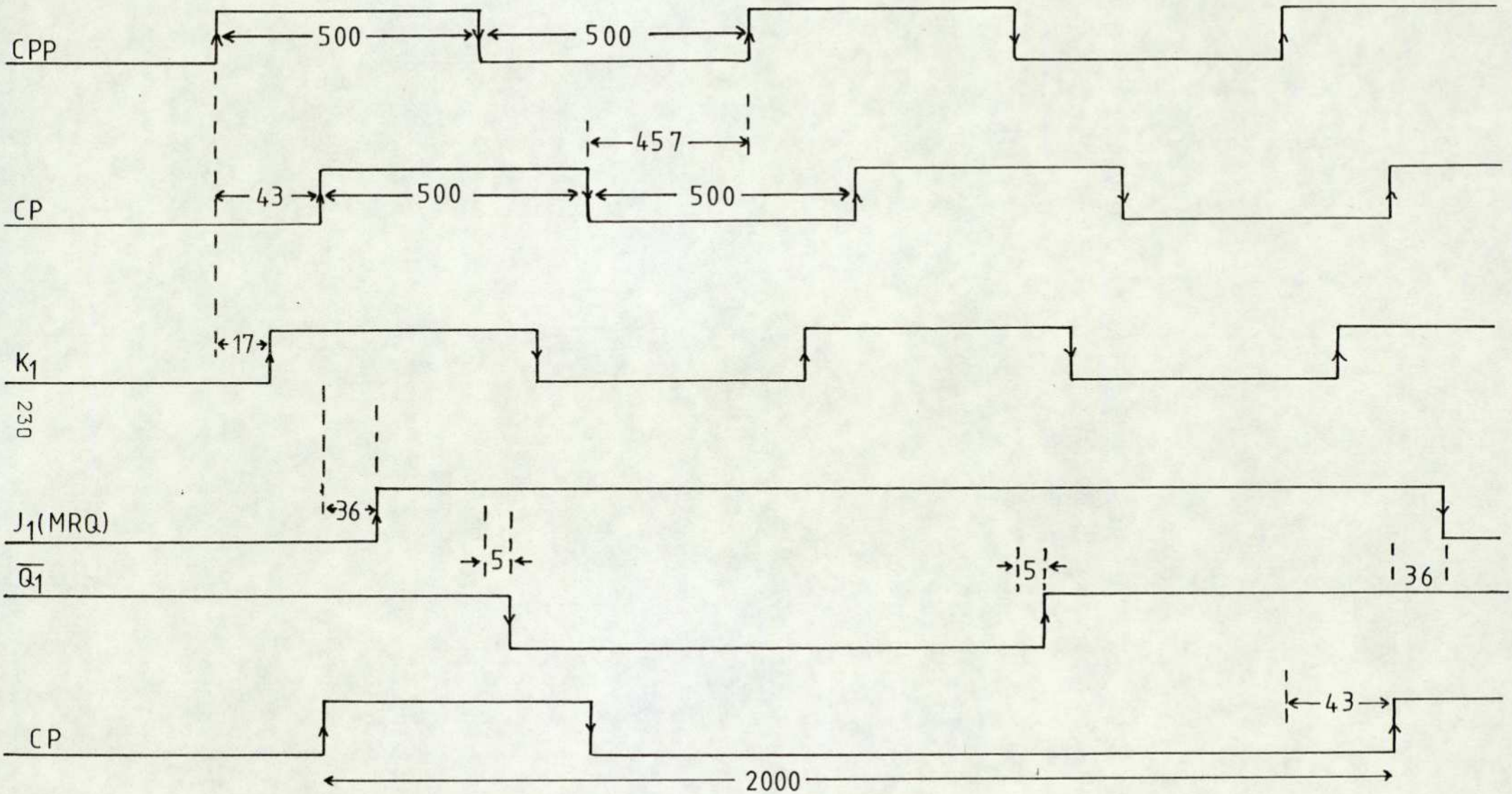


Figure 10.2 The clock period can be increased by a factor of 2 during the memory access operations (all delays in ns)

CHAPTER 11

11. THE COMPUTER CONTROL UNIT (CCU)

11.1 Introduction

The Computer Control Unit is the single most complicated subsystem of the microcomputer. It is this subsystem that controls all other subsystems of the microcomputer, fetches and decodes the macroinstruction and sends the appropriate commands to other subsystems of the microcomputer to perform the required task.

The functions of the CCU are described in the following flow table:-

Initialize the processor
Loop 1 If interrupt detected do interrupt routine
Loop 2 Fetch an instruction
Decode the instruction
Increment the Program Counter
If another operand or data required go to Loop 2
Execute the instruction
Go to Loop 1

Interrupt Routine
Disable further interrupts
Service the current interrupt
Go to Loop 1

This Chapter will describe the principles and design of the Computer Control Unit of the microcomputer.

11.2 Typical Architecture of a 16 bit Microprogrammable CCU

The functional block diagram of a typical microprogrammable CCU [12] is shown in Fig. 11.1. The Instruction Register (a 16 bit clocked register) receives instructions from the main memory via the Data Bus. This register consists of two eight bit fields:- The Op-Code and the Operand. The Operand specifies the number of a selected register in the Arithmetic Processing Unit and the Op-Code specifies the macrooperation (macroinstruction) to be executed.

There is never one to one correspondence between the Op-Codes and the microprogram starting addresses for the execution of the Op-Codes. This is because each macroinstruction is executed by combining more than one microinstructions in the microprogram memory. Also usually different types of macroinstructions require a different number of microinstructions. To avoid this problem, a PROM or ROM type of memory is used which decodes the Op-Codes to their starting addresses in the microprogram memory. This memory is usually called the Mapping Memory. The width of the Mapping Memory should be equal to or wider than the width of the Op-Code register. A usual choice is to make the Mapping Memory as 12 bits wide for an 8 bit Op-Code. This allows the Op-Codes to be decoded up to 4K in the microprogram memory.

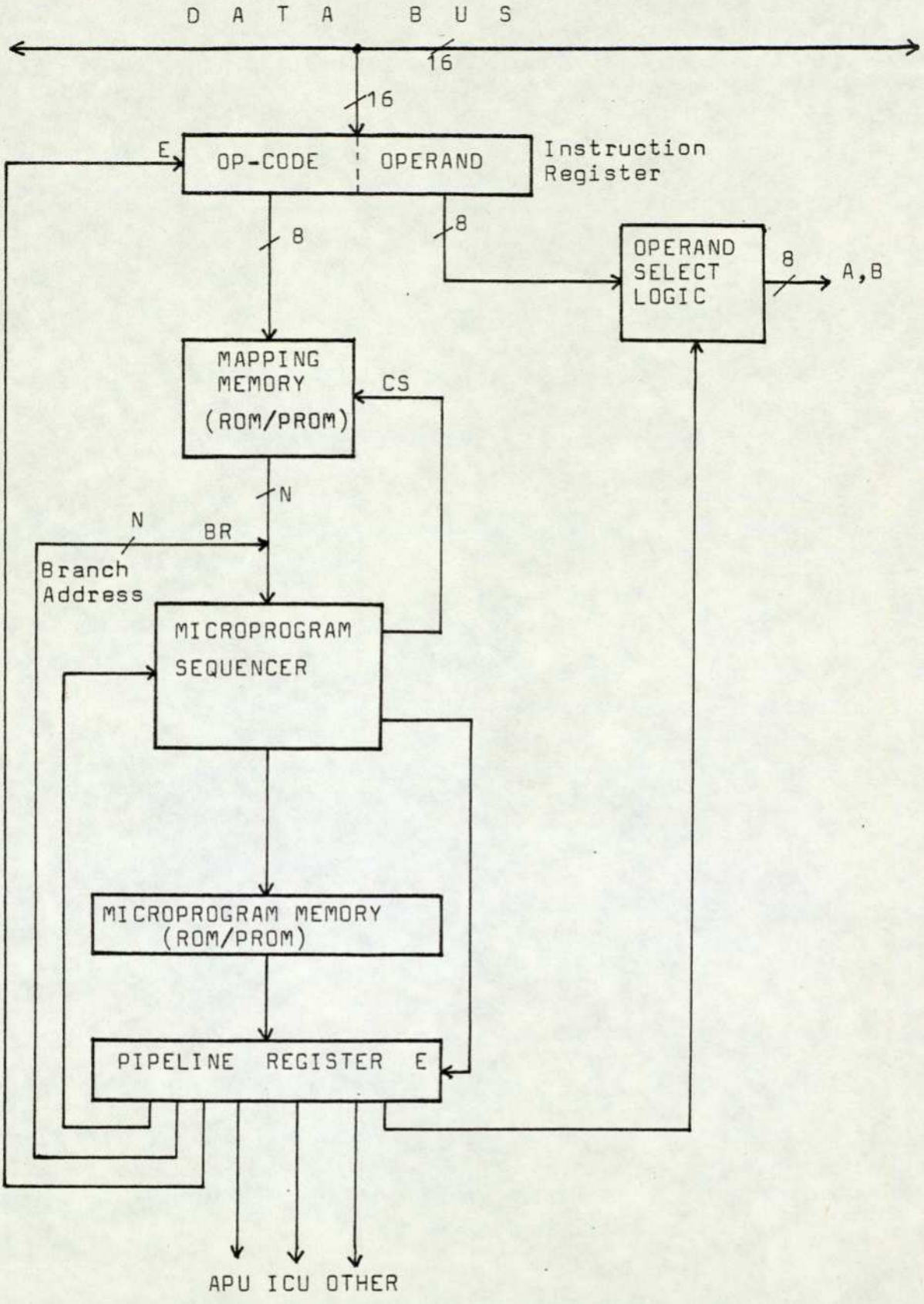


Figure 11.1 Block diagram of a typical 16 bit microprogrammable CCU .

The output of the Mapping Memory is loaded into a Microprogram Sequencer which points to the first microinstruction of the current macroinstruction. The main purpose of the Microprogram Sequencer is to present addresses to the microprogram memory so that the macroinstructions can be fetched and executed.

The output of the Microprogram Sequencer is the address input to the microprogram memory which is usually a fast Bipolar ROM/PROM, having a width in the range of 30 to 90 bits and containing 256 words to 4K words of microcode.

The microprogram data is loaded into a clocked register (Pipeline Register). The Pipeline Register speeds up the operation of the system because it allows the address of the microprogram memory to be changed while executing the current microinstruction (at the output of the Pipeline Register).

The Operand Select Logic provides the register addresses ('A' and 'B') to the Arithmetic Processing Unit. Some macroinstructions may only require a single 4 bit operand (such as incrementing a register) and some may require two 4 bit operands such as, $R_1 + R_2 \longrightarrow R_2$, where the operand field of the Instruction Register may take the hexadecimal value H'12'. Provision can also be made to select any register entirely by the CCU without having to specify any operand in the macroinstruction (e.g.

when working with 8 bits of data length where only the Op-Codes are used to specify the operation and the register addresses).

11.3 An Improved CCU

Although the above mentioned architecture constitutes the basic building blocks of a microprogrammable CCU, it has some major drawbacks such as loading and debugging the microcode is costly, time consuming and not easy.(This is because the microcode is implemented in non-volatile memory). Also loading and modifying the Mapping Memory suffers from the same disadvantages. The CCU can be improved by replacing all or part of the ROM/PROM memories with random access memories. By this method the user can write, test and modify his microcode easily on the actual system. When the user is satisfied with his microcode he can then make it non-volatile by loading it into a PROM (or ROM) type of memory. The advantages of having two types of memories in the CCU can be summarized as follows:-

i. The user can use the microcomputer as a fixed instruction set microcomputer if he does not wish to generate his own instructions.

ii. The user gains a better understanding of micro-programming the microcomputer by analysing the microcode already in the PROM.

iii. The user can generate his own microcode and at the same time he can use the microcode available in the PROM so that the need to write very lengthy micro-programs is eliminated.

iv. A small monitor can be developed in the main memory by using the available instructions in the micro-program PROM. Thus, the user can simply write programs in the main memory to test his microcode.

v. Operating the microcomputer is simplified because the user is not required to write microcode to initialize the system, service the interrupts etc.

The microcomputer designed utilizes a combination of PROM and RAM in its CCU. Some of the PROM part is loaded (see Chapter 12) with microinstructions which can execute several fixed macroinstructions. The PROM part of the Mapping Memory (Mapping PROM) was allocated the first 128 addresses of the Op-Code (H'00' to H'7F'). The remaining 128 addresses of the Op-Code (H'80 to H'FF') were reserved for the RAM part (Mapping RAM) of the Mapping Memory. Similarly, the PROM part of the microprogram memory (micro-program PROM) was allocated the first 256 addresses (H'000 to H'0FF'). The remaining addresses (H'100' to H'FFF') were reserved for the RAM part (Microprogram RAM) of the microprogram memory. The memory map of the CCU is shown in Fig. 11.2.

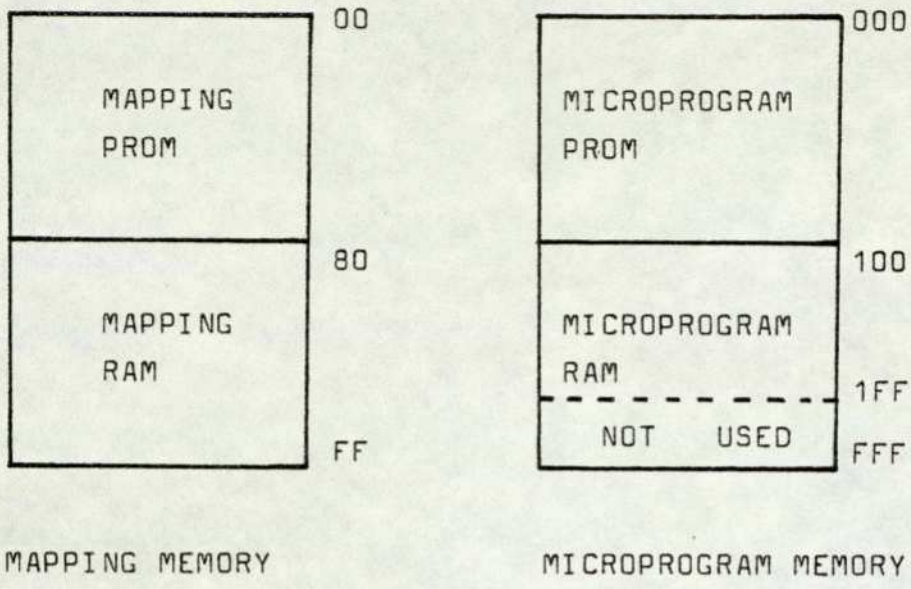


Figure 11.2 The Mapping Memory and the Microprogram Memory of the CCU were designed from a combination of PROM and RAM type memories .

11.4 Design of the CCU

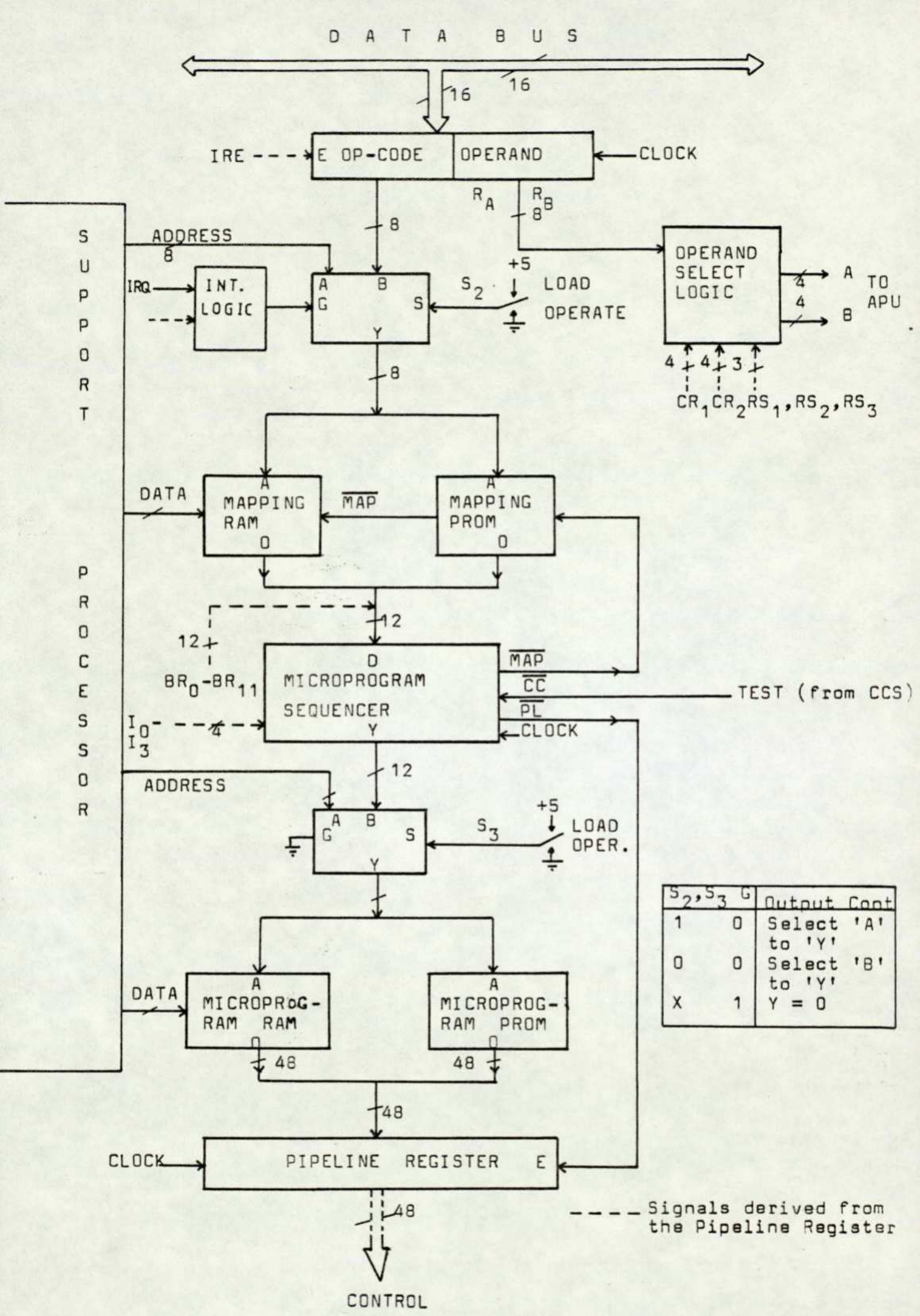
The design of the CCU was mainly based upon operation with 16 bits of data length. But hardware was provided at appropriate stages in the design to enable operation with 8 bits of data length. The Mapping PROM and the Microprogram PROM were loaded with microinstructions to execute fixed macroinstructions based upon only 16 bits of data length.

The simplified block diagram of the CCU designed is shown in Fig. 11.3. The design of the several parts of the CCU are described in the following sub-sections.

11.4.1 The Instruction Register

The Instruction Register is loaded every time a macroinstruction is fetched or an operand is required. It is therefore desirable to choose a register which incorporates a load control facility.

The Instruction Register was designed by using two Am74LS377 type 8 bit registers (see Fig. 11.4) with register enable control inputs, IRE. When IRE is LOW, new data is loaded into the Instruction Register on the LOW-to-HIGH transition of the system clock. When IRE is HIGH the Instruction Register retains its data independent of the clock.



S_2,S_3	G	Output Cont
1	0	Select 'A' to 'Y'
0	0	Select 'B' to 'Y'
X	1	Y = 0

Fig. 11.3

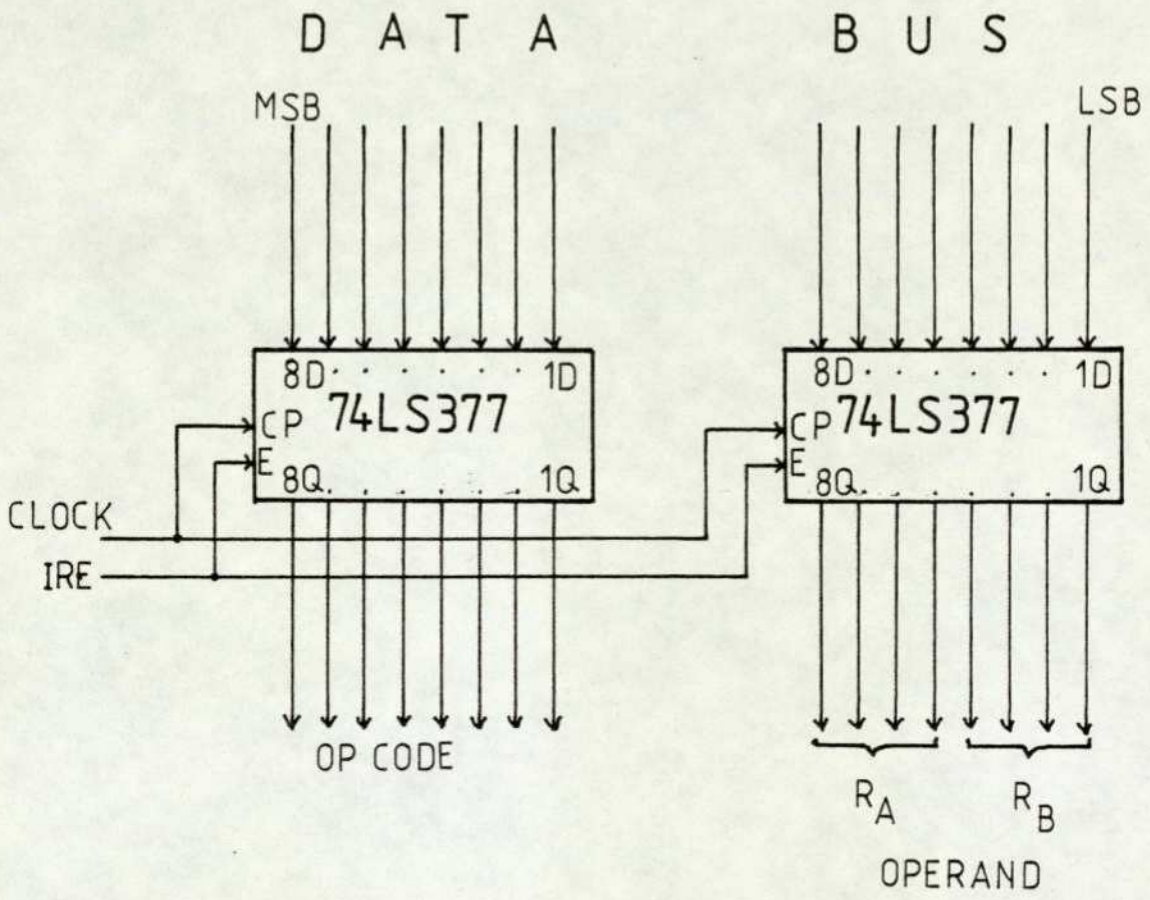


Figure 11.4 Logic diagram of the Instruction Register .

11.4.2 The Operand Select Logic

The function of the Operand Select Logic is to provide the register addresses of the Arithmetic Processing Unit. Although the register addresses are specified in the Operand field of the Instruction Register, connecting them directly to the 'A' and 'B' inputs of the Am2903s has some disadvantages. This is because only the 'B' addresses can be used as the destination register addresses. It is usually necessary to select the 'A' and 'B' register addresses from several combinations of the Instruction Register Operand field (via multiplexers).

The Operand Select Logic was designed by using Am74LS253 and Am74LS257 type multiplexers (see Fig. 11.5). The Operand part of the Instruction Register was divided into two four bit fields 'R_A' and 'R_B' and were presented to the Operand Select Logic. The Operand Select Logic also receives an eight-bit data (CR_{10}-CR_{13} and CR_{20}-CR_{23}) from the microprogram memory*. Under the control of the inputs RS_1, RS_2, and RS_3, the register addresses of the Am2903s can be derived from one of the following sources:-

* This is necessary during 8 bit operation. Also some special instructions (eg. multiplication) of the Am2903 requires the register addresses to be specified by the microprogram memory .

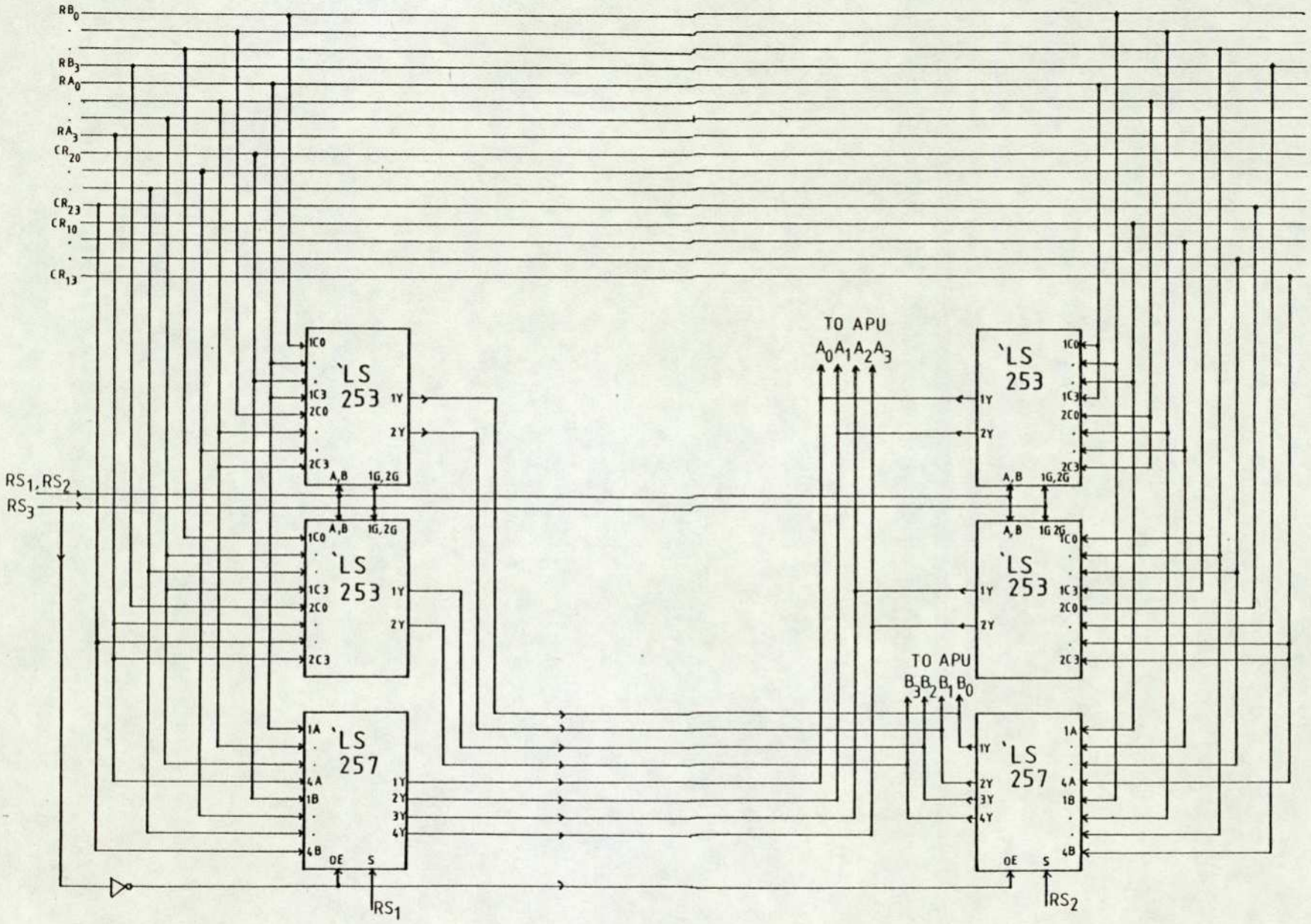


Figure 11.5 The Operand Select Logic

Operand Select Logic Control bits			Source of the Am2903 Register Addresses	
RS_3	RS_2	RS_1	'A'	'B'
0	0	0	R_A	R_B
0	0	1	R_B	R_A
0	1	0	CR_1	CR_2
0	1	1	R_A	R_A
1	0	0	R_A	CR_1
1	0	1	CR_2	CR_1
1	1	0	R_A	R_B
1	1	1	CR_2	R_B

11.4.3 The Microprogram Sequencer

The Microprogram Sequencer chosen in the design was the Am2910, manufactured by the Advanced Micro Devices [13]. The reason of choosing this particular sequencer was because of its powerful and compact sequencing capabilities.

The Am2910 is a Bipolar microprogrammable sequencer that can execute 16 different control instructions (most of them conditional). It can be used to address up to 4096 words of the microprogram memory. Basically it contains a four input multiplexer (see Fig. 11.6) which selects

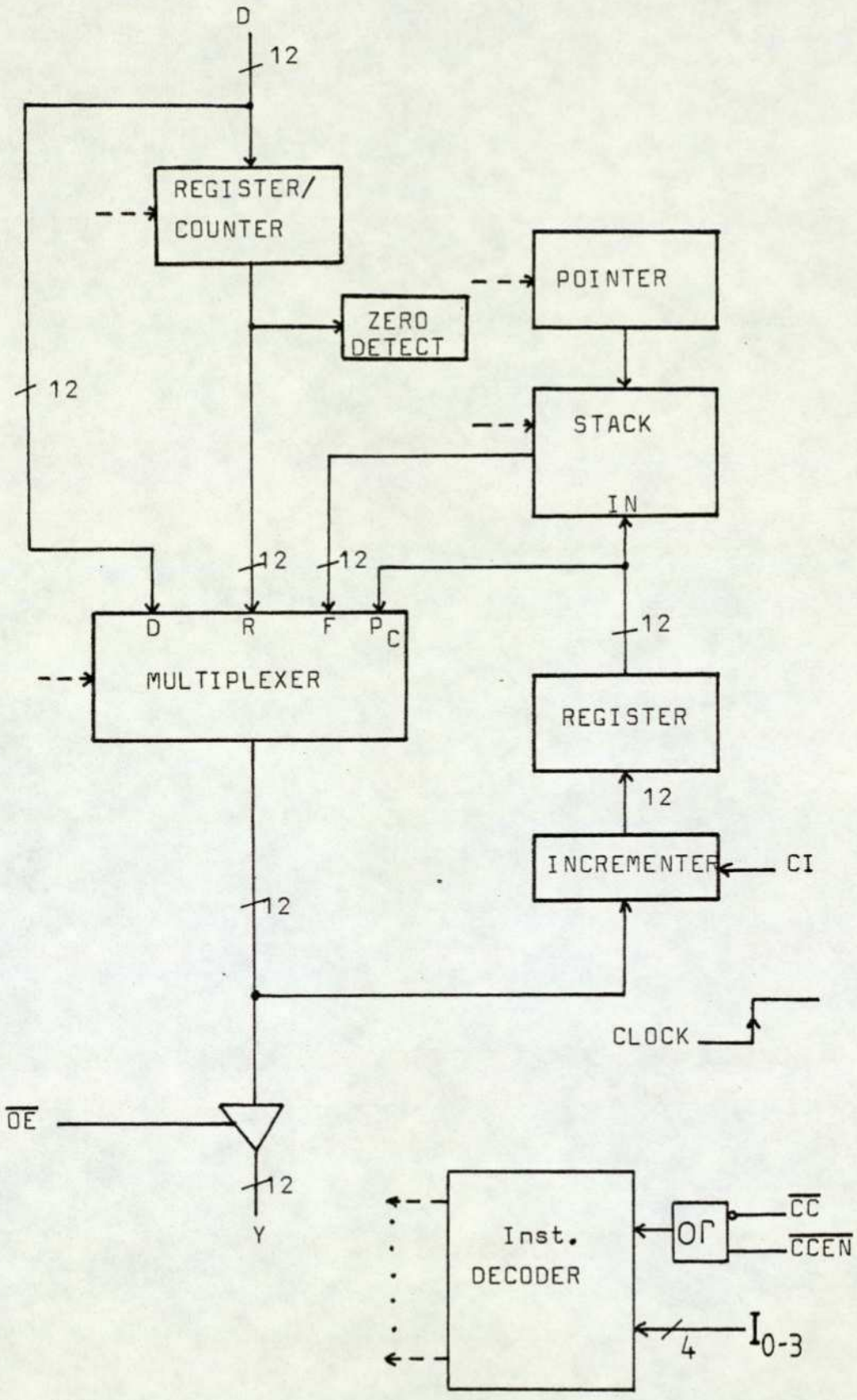


Figure 11.6 Block diagram of the Am2910 Microprogram Sequencer .

the source of the next microinstruction address. In addition to its sequencing and branching functions, the Am2910 provides microsubroutine and looping capability. A 5 word deep stack (file) saves the microsubroutine return addresses.

The 'D' input (12 bits wide) is used for branching or loading data to the sequencer. The '$\overline{CC}$' input is used during the conditional instructions and a test is applied to this pin.

The logic diagram of the Microprogram Sequencer is shown in Fig. 11.7. The four control inputs (I_0, I_1, I_2 and I_3) control the operations of the Am2910 as described in Table 11.1. The best way to understand the operation of the Am2910 is to look at some examples (see Fig. 11.8).

The microprogram addresses used in the examples are arbitrarily chosen (except the first example) and have no meaning other than to show instruction flow. Each dot refers to the time that the contents of the microprogram memory word is in the Pipeline Register.

Instruction 0 is the JUMP ZERO instruction and the output of the sequencer (Y_0 to Y_{11}) is forced to zero from the current value. This instruction is used during the initialization of the microcomputer where the first macroinstruction to be executed will be the one at microprogram address zero. When the RESET button is pressed (see Section 10.2) the RESET signal goes LOW temporarily thus forcing

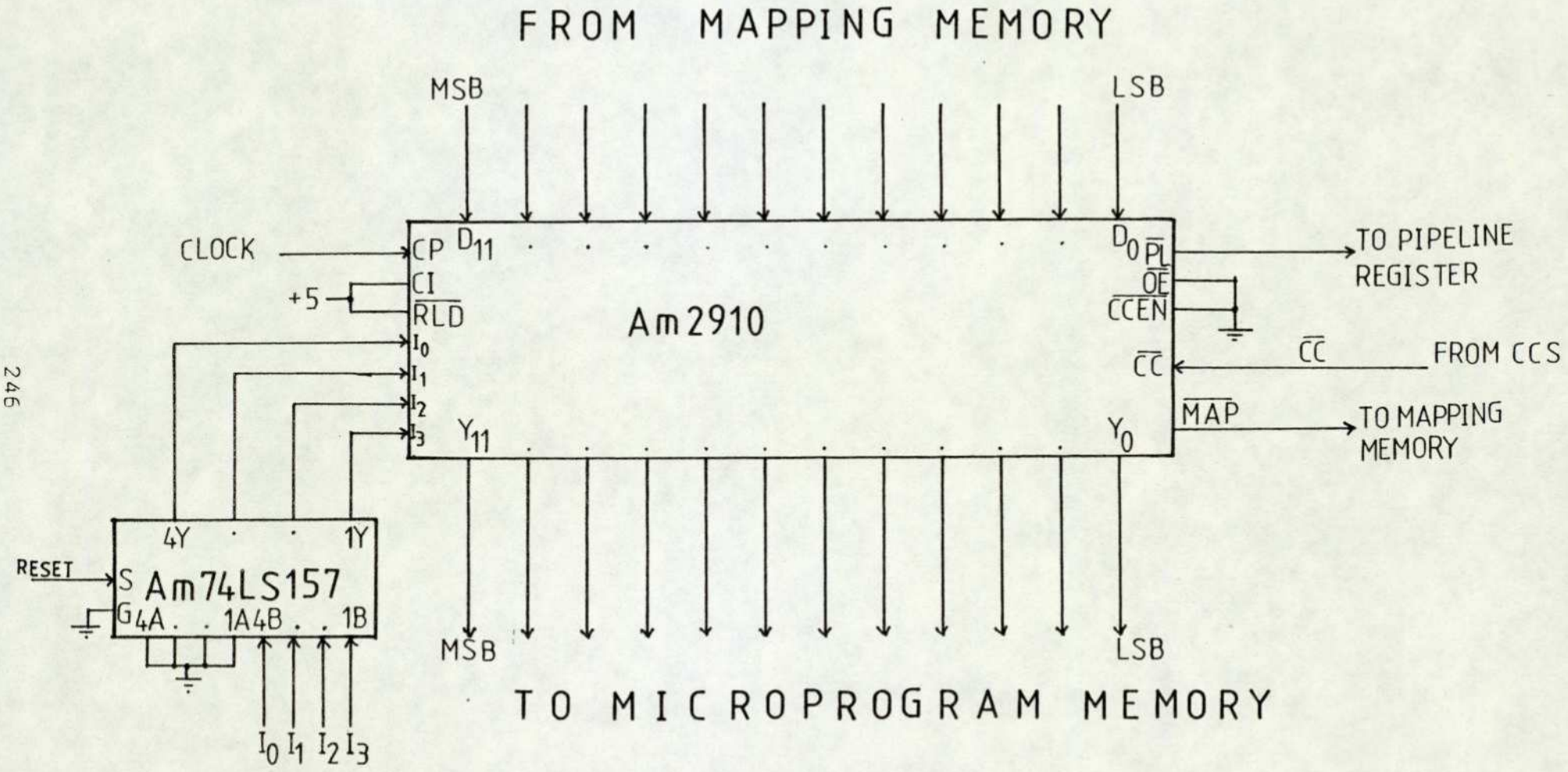
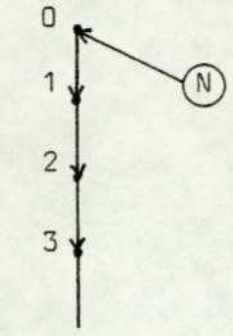


Figure 11.7 Logic diagram of the Microprogram Sequencer .

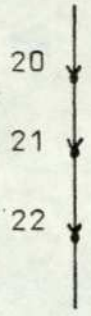
	I_3	I_2	I_1	I_0	INSTRUCTION	Reg/Cntr contents	FAIL(CC=HIGH) Y	FAIL(CC=HIGH) STACK	PASS($\overline{CC}$=LOW) Y	PASS($\overline{CC}$=LOW) STACK	Reg. cntr	$\overline{MAP}$	$\overline{PL}$
0	0	0	0	0	JUMP ZERO	X	0	CLEAR	0	CLEAR	HOLD	1	0
1	0	0	0	1	Cond. Jump Sub. via PL	X	PC	HOLD	D	PUSH	HOLD	1	0
2	0	0	1	0	JUMP MAP	X	D	HOLD	D	HOLD	HOLD	0	1
3	0	0	1	1	Cond. Jump via PL	X	PC	HOLD	D	HOLD	HOLD	1	0
4	0	1	0	0	Push/Cond. Load Counter	X	PC	PUSH	PC	PUSH	LOAD	1	0
5	0	1	0	1	Cond. Jump Sub. via R/PL	X	R	PUSH	D	PUSH	HOLD	1	0
6	0	1	1	0	Jump Vector	X	PC	HOLD	D	HOLD	HOLD	1	1
7	0	1	1	1	Cond. Jump via R/PL	X	R	HOLD	D	HOLD	HOLD	1	0
8	1	0	0	0	Repeat Loop,counter ≠ 0	≠ 0	F	HOLD	F	HOLD	DEC	1	0
						= 0	PC	POP	PC	POP	HOLD	1	0
9	1	0	0	1	Repeat PL,counter ≠ 0	≠ 0	D	HOLD	D	HOLD	DEC	1	0
						= 0	PC	HOLD	PC	HOLD	HOLD	1	0
10	1	0	1	0	Conditional Return	X	PC	HOLD	F	POP	HOLD	1	0
11	1	0	1	1	Cond. Jump PL and POP	X	PC	HOLD	D	POP	HOLD	1	0
12	1	1	0	0	Load counter and Continue	X	PC	HOLD	PC	HOLD	LOAD	1	0
13	1	1	0	1	Test End Loop	X	F	HOLD	PC	POP	HOLD	1	0
14	1	1	1	0	Continue	X	PC	HOLD	PC	HOLD	HOLD	1	0
15	1	1	1	1	Three-way Branch	≠ 0	F	HOLD	PC	POP	DEC	1	0
						= 0	D	POP	PC	POP	HOLD	1	0

R = Register counter ; F = File ; D = 'D' inputs ; PL = Pipeline Register ; PC = Microprogram Counter .

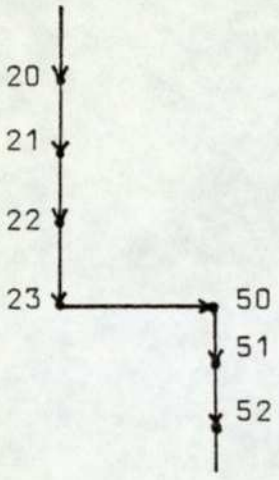
Table 11.1 Am2910 Instructions .



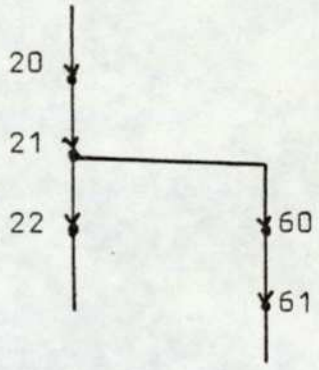
JUMP ZERO



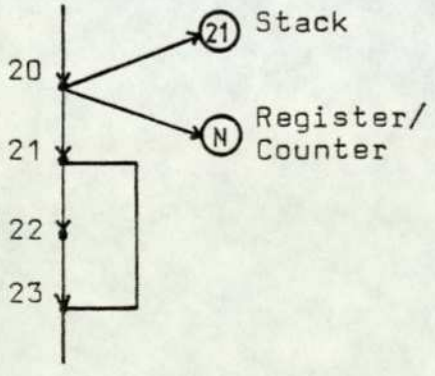
CONTINUE



JUMP MAP



CONDITIONAL JUMP



REPEAT LOOP, COUNTER ≠ 0

Figure 11.8 Am2910 Execution Examples .

all four control inputs of the Am2910 LOW (JUMP ZERO). After a delay, the RESET signal goes HIGH thus selecting the data for the Am2910 control inputs from the 'B' inputs of the data selector (see Fig. 11.7).

Instruction 14 (CONTINUE) is used for single stepping through the microprogram addresses.

Instruction 2 is the JUMP MAP instruction and upon execution, the Mapping Memory will be enabled ($\overline{MAP}$ = LOW in Fig. 11.7) and the Am2910 will receive data from its 'D' inputs. In the example of Fig. 11.8, the Mapping Memory contains the data 50 when a JUMP MAP instruction is executed. This instruction is used during the FETCH cycle.

Instruction 3 is a CONDITIONAL JUMP instruction where the jump address must be applied to the 'D' inputs of the Am2910. During the execution of this instruction the Mapping Memory is disabled to avoid two levels of data at the 'D' inputs. If the condition tested is true ($\overline{CC}$ input LOW) then the value currently at the 'D' inputs (60 in this example) will be selected at the 'Y' outputs of the sequencer, otherwise the output of the sequencer will be incremented by one (CONTINUE).

Instruction 8 is the REPEAT LOOP, COUNTER $\neq$ ZERO instruction. This instruction makes use of the decrementing capability of the register/counter (see Fig. 11.6). To be

useful, some previous instruction, such as 4, must have loaded a count value into the register/counter. This instruction checks to see whether the register/counter contains a non-zero value. If so, the register/counter is decremented and the address of the next microinstruction is taken from the top of the stack. If on the other hand the register/counter contains zero, then the next sequential address is selected at the output and the stack is POP'd by decrementing the stack pointer. The register/counter is arranged such that if it is pre-loaded with a number N and then used as a loop termination counter, the sequence will be executed exactly N+1 times.

11.4.4 The Mapping Memory

The Mapping Memory was designed from a combination of PROM and RAM type of memories. The PROM part of the Mapping Memory (Mapping PROM) was allocated the first 128 addresses of the Op-Code (00 to H'7F') and it was designed such that the first 256 words of the microprogram PROM (000 to H'0FF') could be addressed by the Mapping PROM. Similarly, the RAM part of the Mapping Memory (Mapping RAM) was allocated the addresses H'80' to H'FF'of the Op-Code and it was designed such that the microprogram addresses H'100' to H'1FF' could be addressed by the Mapping RAM.

Although the Mapping PROM was allocated 128 addresses,

only the first 64 locations were used (for operation with 16 bits of data length) and the remaining 64 were reserved for future expansion.

Also, only the first 96 locations of the microprogram PROM were used (for operation with 16 bits of data length) and the remaining 160 locations were reserved for future expansion of the microprogram PROM.

The memory map of the Mapping Memory and the microprogram memory are shown in Fig. 11.9 and Fig. 11.10.

The logic diagram of the Mapping Memory is shown in Fig. 11.11. The address inputs of the Mapping Memory were derived from the Op-Code and the Support Processor (see Chapters 15 and 16) via two data selectors (Am74LS157). A switch, S_2, was used to control the data selectors such that when S_2 is in the LOAD mode (S_2 = HIGH), the memory addresses are derived from the Support Processor for loading the Mapping RAM. When S_2 is in the OPERATE mode (S_2 = LOW), the memory addresses are derived from the Op-Code field of the Instruction Register for normal operation. The data inputs to the Mapping RAM were also derived from the Support Processor.

The Mapping RAM was designed by using Am9101 type memories. To simplify loading, the Mapping RAM was divided into two fields:- field 1 (MR1 and MR2 in Fig. 11.11) was eight bits wide and field 2 (MR3 in Fig. 11.11) was four

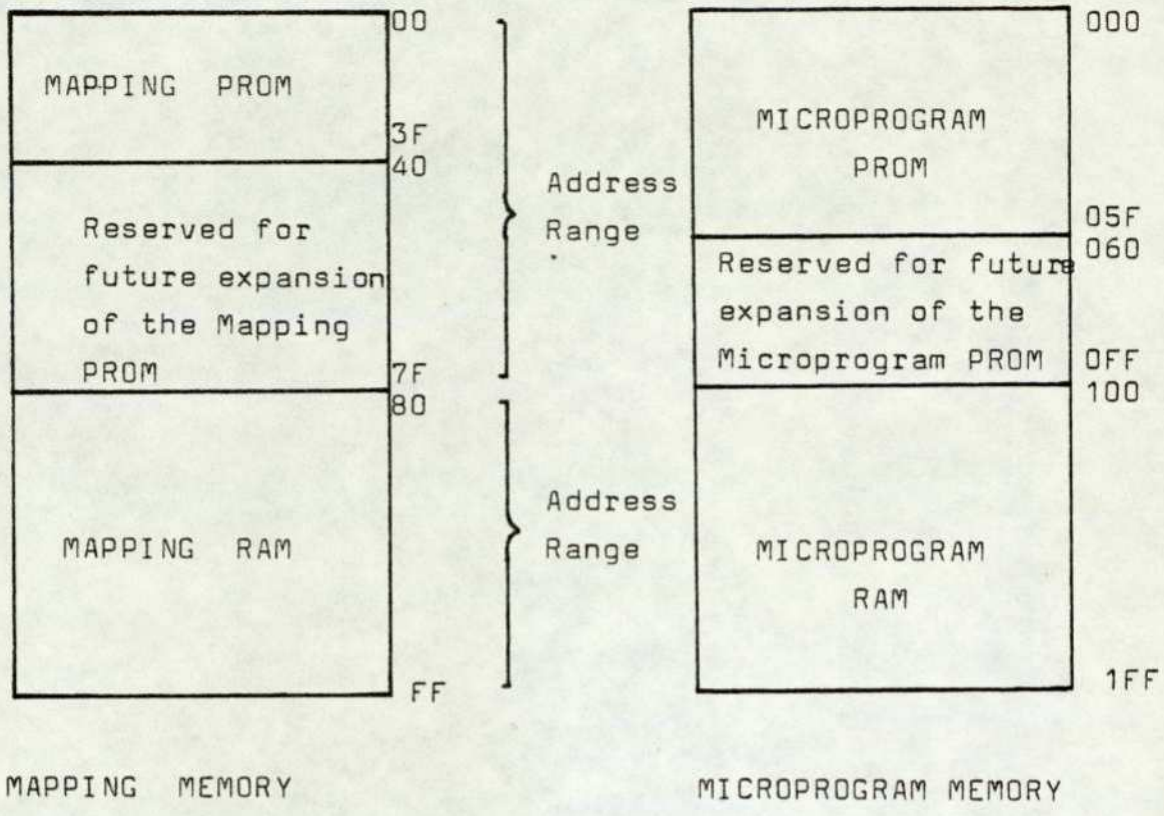


Figure 11.9 Memory map of the Mapping Memory and the Microprogram Memory (when working with 16 bits of data length) .

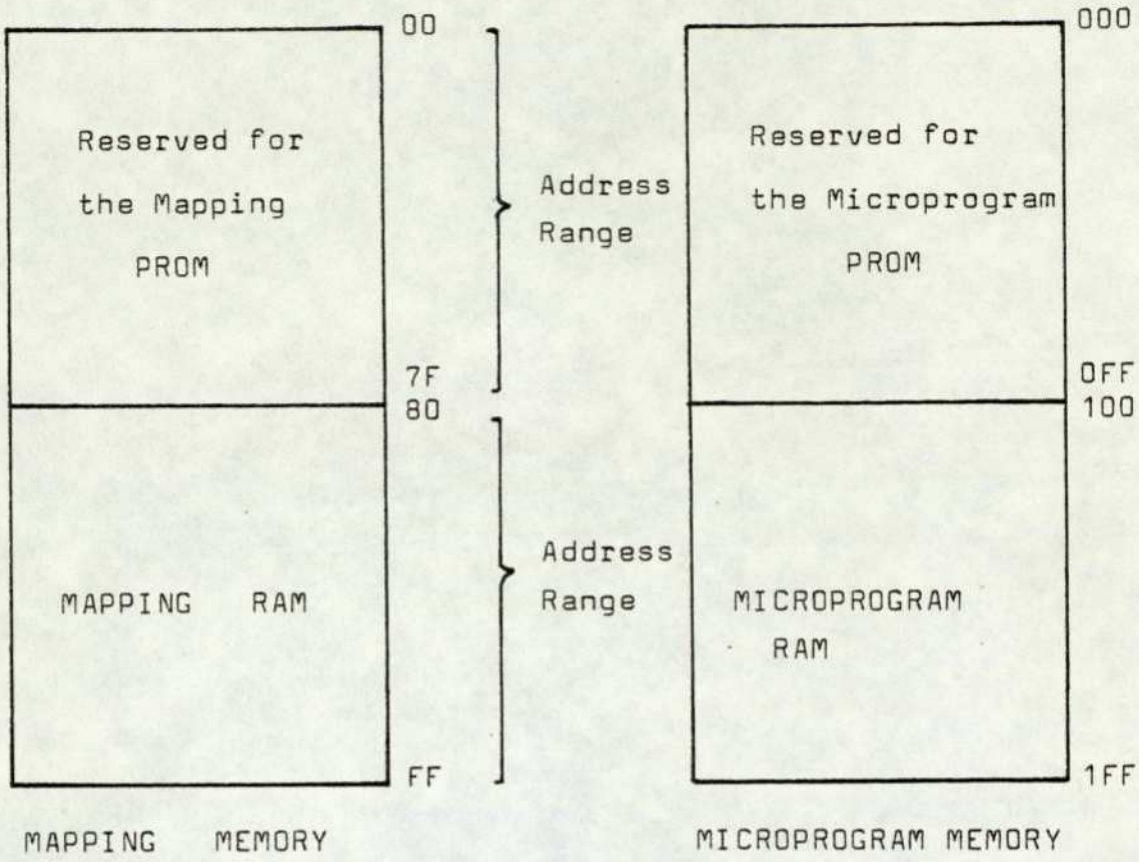


Figure 11.10 Memory map of the Mapping Memory and the Microprogram Memory (When working with 8 bits of data length)

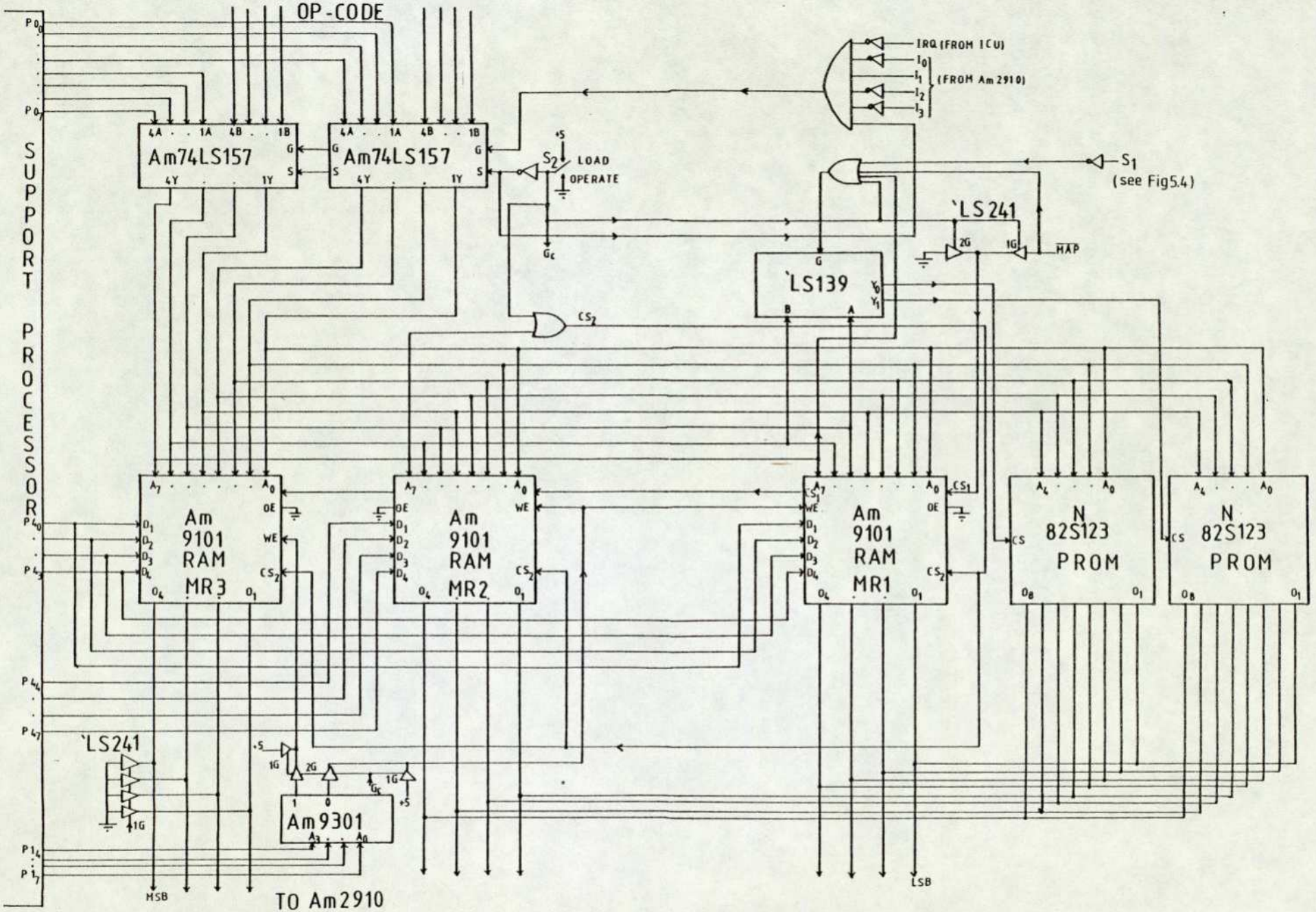


Figure 11.11 Logic diagram of the Mapping Memory

bits wide. During a load operation, the Support Processor controls (via a multiplexer and three-state buffers) the $\overline{WE}$ inputs of the Mapping RAM and loads the required data into the required field of the Mapping RAM. Thus, when S_2 is in the LOAD mode, the $\overline{WE}$ inputs of the Mapping RAM are under control of the Support Processor and when S_2 is in the OPERATE mode, the $\overline{WE}$ inputs are forced to logic HIGH (memory in READ mode).

The Am9101 memories are enabled by two control bits CE_1 and CE_2 as follows:-

CE_1	CE_2	Outputs
0	1	Enabled
1	X	Disabled (three-state)
X	0	Disabled (three-state)

During the LOAD mode it is necessary to enable the Mapping RAM and during the OPERATE mode, the Mapping RAM must be controlled by the $\overline{MAP}$ output of the microprogram sequencer. Thus, three-state buffers were used to control the CE_1 inputs of the Am9101s. As shown in Fig. 11.11, when S_2 is in the LOAD mode, the CE_1 inputs are forced to logic LOW and when S_2 is in the OPERATE mode, the CE_1 inputs are under control of the $\overline{MAP}$ output of the microprogram sequencer.

The CE_2 enable inputs were used to enable the Mapping

RAM in the address range H'80' to H'FF' when in the OPERATE mode. Hence, bit 7 (MSB) of the Mapping RAM address input was OR'd with S_2 and was used to control CE_2 inputs of the Mapping RAM. When S_2 is in the LOAD mode (S_2 = HIGH), the Mapping RAM is always enabled (CE_1= LOW and CE_2 = HIGH). When S_2 is in the OPERATE mode (S_2 = LOW), the Mapping RAM will only be enabled in the Op-Code range H'80' to H'FF' if $\overline{MAP}$ output of the microprogram sequencer is LOW (see Table 11.1).

The Mapping PROM was designed by using N82S123 type memories. These are 32 word by 8 bit PROM memories. Two such devices were used to make a 64 word by 8 bit Mapping PROM. A two input multiplexer (Am74LS139) was used to control the active LOW enable inputs (CS) of the Mapping PROM. This multiplexer is controlled by its 'G' input such that when 'G' is HIGH, all four outputs are HIGH (Mapping PROM not selected) and when 'G' is LOW only one output is LOW (only one memory selected). The 'G' input of the multiplexer was controlled as illustrated in the following truth table (S_1 is the 8/16 bit select switch and A_7 is the MSB of the Mapping Memory addresses).

$\overline{MAP}$	S_1	S_2	A_7	G	
1	X	X	X	1	Mapping PROM not enabled
X	X	X	1	1	Mapping PROM not enabled
X	X	1	X	1	Mapping PROM not enabled
0	1	0	0	0	Mapping PROM enabled

Hence, $G = \overline{MAP} + \overline{S}_1 + S_2 + A_7$

The Microprogram Sequencer requires a 12 bit input data. But the Mapping PROM can only supply 8 bits of data. Thus the four higher bits of the microprogram sequencer were forced to logic LOW (by using three-state buffers) whenever the Mapping PROM is enabled.

It was mentioned in Chapter 9 that an interrupt request (IRQ) can only be accepted by the CCU when the CCU is in the macroinstruction FETCH cycle. The data selectors at the address inputs of the Mapping Memory were used to provide the CCU an automatic capability to accept or reject an interrupt request from the ICU. When the 'G' inputs of the data selectors are LOW, they operate in the normal manner. But when the 'G' inputs are HIGH, all eight outputs of the data selectors are forced to logic LOW. The 'G' inputs of the data selectors were derived from the following truth table (I_o to I_3 are the microprogram sequencer control bits):-

IRQ	I_3	I_2	I_1	I_o	S_2	G
1	X	X	X	X	X	0
0	0	0	1	0	0	1
		otherwise				0

Hence, $\overline{G} = \overline{I}_o . \overline{I}_2 . \overline{I}_3 . I_1 . \overline{IRQ} . \overline{S}_2$

When the CCU is in the macroinstruction FETCH cycle

($I_0 = I_2 = I_3$ = LOW and I_1 = HIGH) and an interrupt request is present (IRQ = 0), the address inputs of the Mapping Memory will be forced to logic LOW. Hence, address zero of the Mapping Memory must contain the address for the interrupt service routine in the microprogram memory. By this method the need for testing an interrupt request at every FETCH cycle was eliminated.

11.4.5 The Microprogram Memory Width

The width of the microprogram memory depends upon the number of control signals required by the various subsystems of the microcomputer. The total number of control signals required by the microcomputer are summarized below:-

Subsystems	Control Signals required	Total No.
Arithmetic and Logic Unit	I_0-I_8, $\overline{E}_A$, $\overline{OE}_B$, $\overline{IEN}$, $\overline{OE}_Y$	13
Bit Manipulation Logic	E_4, Pol	2
Data Bus Buffers	EDB, EB	2
Register Extension Logic	P/P, FE	2
Condition Code Select Logic, Carry in Control Logic, Data Shifting Logic	S_1-S_5, C_3-C_6, C_8, ST_1, ST_2	12
Input-Output Unit	IOR, IOM	2
Program Control Unit	P_1, P_2, P_3, P_4, PCE, PE	6
Main Memory	MRQ, MMD	2
Interrupt Control Unit	A_0, A_1, A_2, A_3	4
Timing Unit	WAIT	1

Subsystems	Control Signals required	Total No.
Microprogram Sequencer	BR_0- BR_{11}, I_0- I_3	16
Instruction Register	IRE	1
Operand Select Logic	CR_{10}- CR_{13},CR_{20}-CR_{23},RS_1,RS_2,RS_3	11

A total of 74 signals are required to control all parts of the microcomputer. A microprogram memory of this width is both costly and complicated (large number of chips and address-lines are required) to implement. However, by careful choice of a microprogram format it was possible to decrease the width of the microprogram memory as described in the following section.

11.4.6 Choosing a Microprogram Format

A microprogram format may be fixed or variable. In a fixed format, each microprogram bit is always interpreted in the same way. In a variable format, the meaning of the microprogram bits change according to certain system conditions. Variable format leads to shorter microprogram width but it has the disadvantages that additional logic is required for format control.

A fixed microprogram format can be implemented in several ways [14] as shown in Fig.11.12. If each bit in the microprogram memory is allocated a separate control line in the system, then this format is called non-encoded. A non-encoded format requires a wider microprogram width

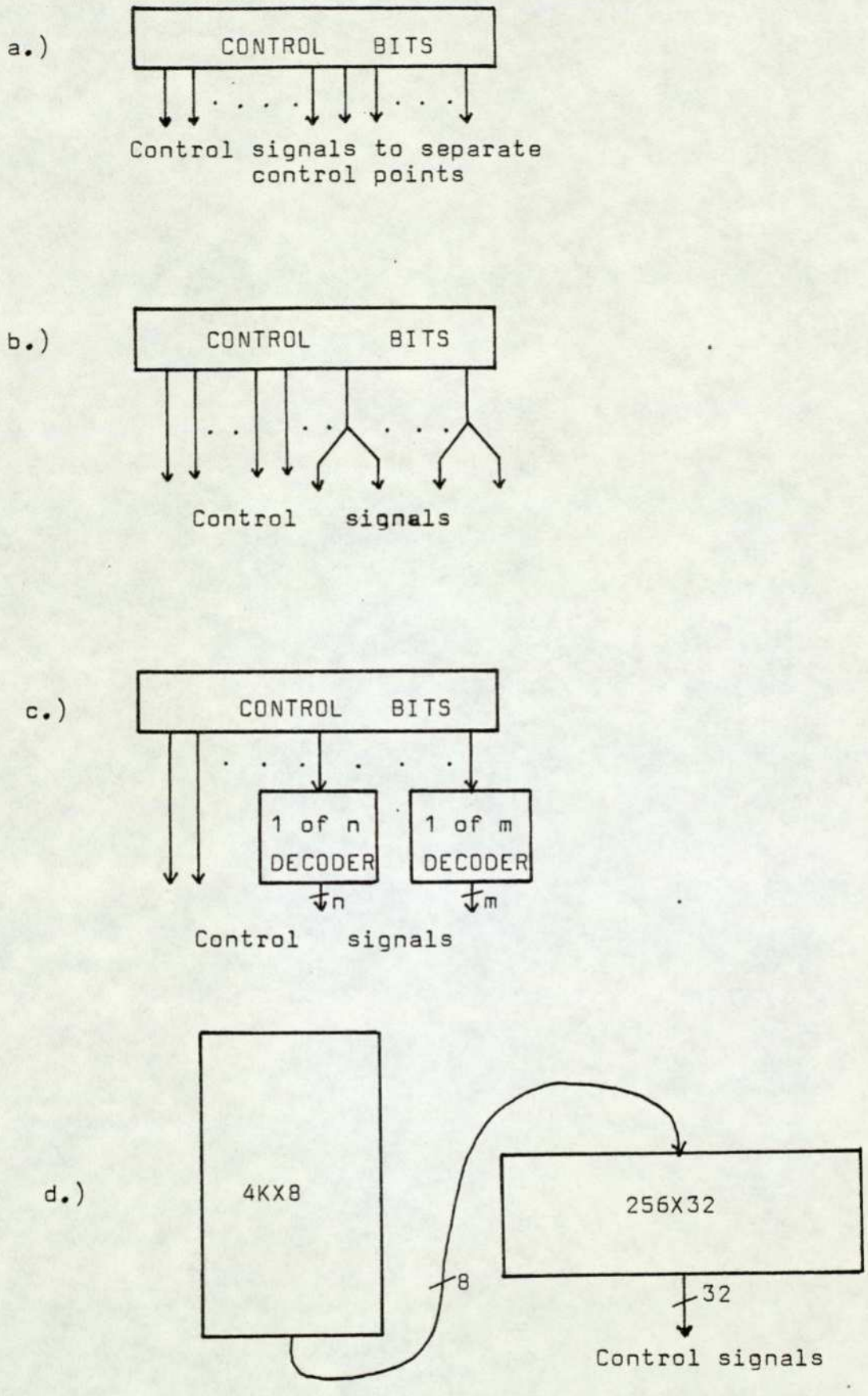


Figure 11.12 Fixed microprogram formats. a.) non-encoded format b.) semi-encoded format c.) encoded format d.) two level formatting .

but it leads to higher operating speeds. A modified version of the non-encoded format is to group the mutually exclusive control lines and control them with the same microprogram bits (semi-encoded format).

In an encoded microprogram format, groups of mutually exclusive control lines are controlled by an encoded single control field in the microprogram memory. If 2^n such lines exist, an n-bit field in the microprogram memory can be used to control them. These n bits can re-generate the required 2^n control signals by means of a 1 to n decoder (see Fig. 11.12). Although the encoded format leads to shorter microprogram widths, the speed of operation is reduced.

Another common formatting is to use two levels of microprogram memory [14]where the first memory has a large (1K) number of short words (8-bit), while the second memory is wide (32-bit) but contains fewer (256) words. Thus a total microprogram memory of 4K words by 32 bits can be implemented with only 4K words by 8 bits and 256 words by 32 bits. In this arrangement the wider memory is usually loaded with microsubroutines.

The microprogram format chosen in the design was the semi-encoded format since it is simple to implement and additional logic is not required. The following control lines were grouped to share the same bits of the microprogram memory (the grouping was done such that the probability of requiring

to use two different subsections of the microcomputer simultaneously in a given group is very small. eg. it is not normally expected to perform an ALU and an ICU operation at the same time) :-

Control signal	Shared with	Bits saved
$\overline{OE}_B, \overline{E}_A, I_0$ (ALU)	A_0, A_1, A_2 (ICU)	3
I_1 - I_8 (ALU)	BR_0 - BR_7 (Branch address)	8
CR_{10} -CR_{13}(Operand select)	BR_8 - BR_{11} (Branch address), Pol, E_4 (BML)	6
CR_{20}-CR_{23} (Operand select)	P_1, P_2, P_3, P_4 (PCU)	4
C_3 (CCS)	PE (PCU)	1
S_1 (CCS)	P/P (REL)	1
S_2 (CCS)	MMD (Memory Unit)	1
S_3 (CCS)	IOM (I/O Unit)	1
S_5 (DSL)	RS_3 (Operand select)	1

With the semi-encoded format, the width of the microprogram memory was reduced to 48 bits (a saving of 26 bits). The final form of the microprogram memory format is shown in Figure 11.13 .

11.4.7 Design of the Microprogram Memory

As shown in Figure 11.2 , the microprogram memory was

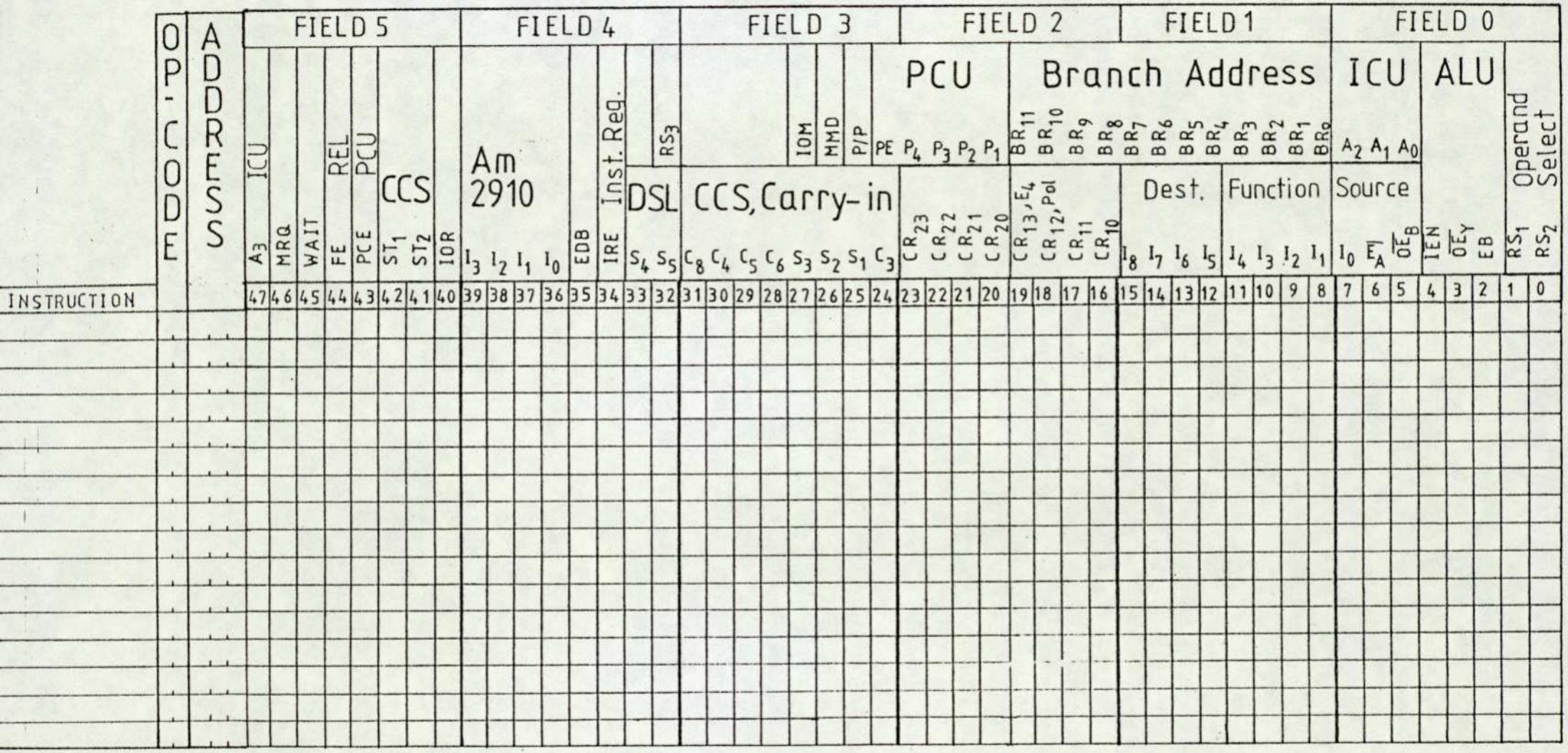


Figure 11.13 The microprogram memory format

designed from a combination of PROM and RAM type of memories. The address input to the microprogram memory can be selected from the microprogram sequencer or the Support Processor via three data selectors (Am74LS157). The data selectors were controlled with a switch ,S_3, (see Fig. 11.3) such that when S_3 is in the OPERATE mode (S_3 = LOW) the microprogram memory addresses are derived from the microprogram sequencer. When S_3 is in the LOAD mode (S_3 = HIGH) the memory addresses are derived from the Support Processor.

The microprogram RAM was designed by using Am9101 type 256 word by 4 bit memories. 12 such devices were used to generate a 256 word by 48 bit microprogram RAM (see Fig. 11.14). In order to simplify loading, these memories were divided into six fields, each field eight-bit wide. When S_3 is in the LOAD mode, the microprogram RAM can be loaded under the control of the Support Processor (see Chapters 15 & 16 for details of the Support Processor). During this mode of operation, a multiplexer (Am9301) selects the microprogram fields to be loaded. When in the OPERATE mode, the WE inputs of the memories are forced HIGH so that data can be READ out.

The microprogram PROM was designed by using N82S123 type 32 word by 8 bit memories. 18 such devices were used in three groups to obtain a 96 word by 48 bit microprogram PROM (see Figure 11.15). The addressing range of each group is as follows :-

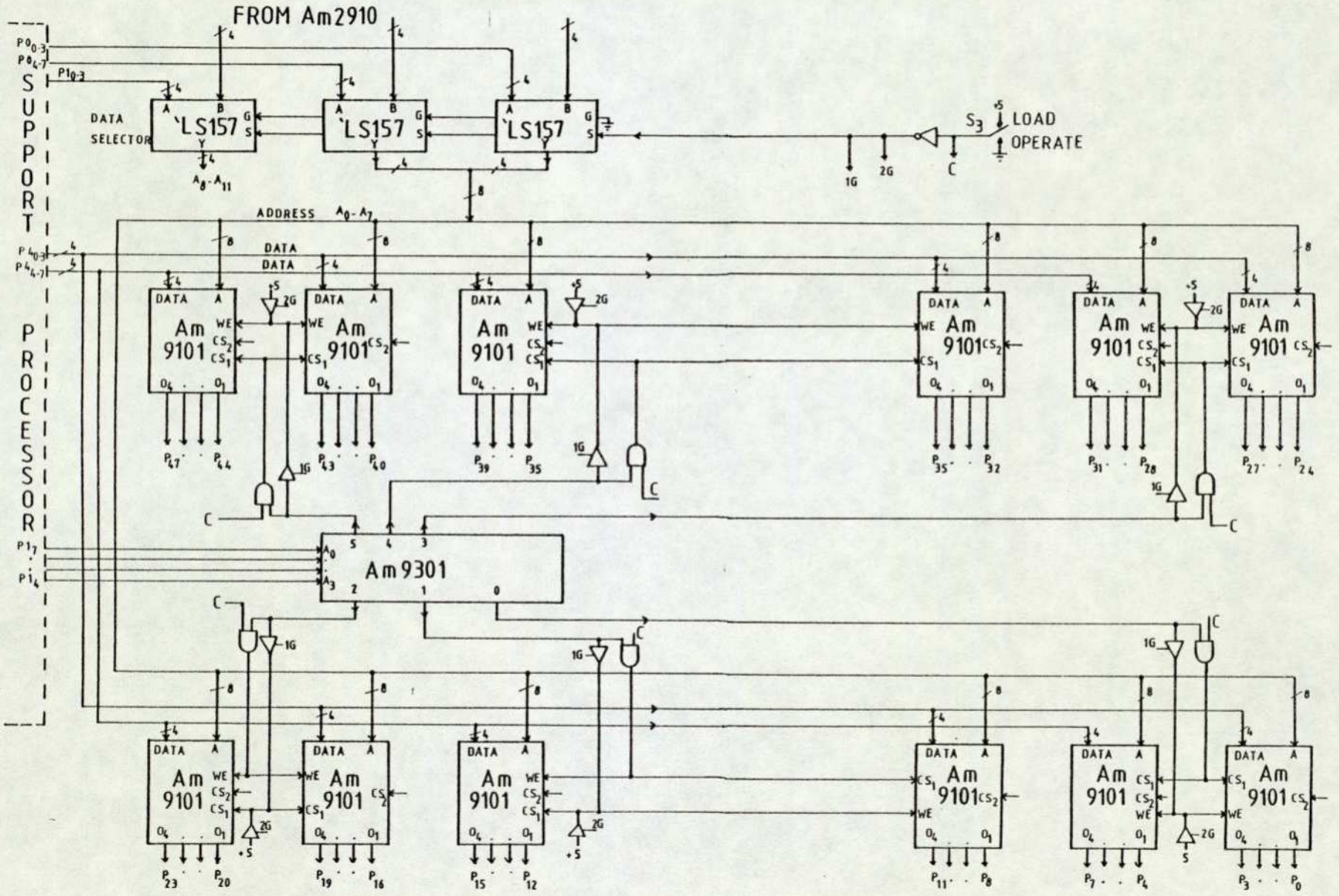


Figure 11.14 The microprogram RAM (P_0 to P_{47} are the 48-bit outputs)

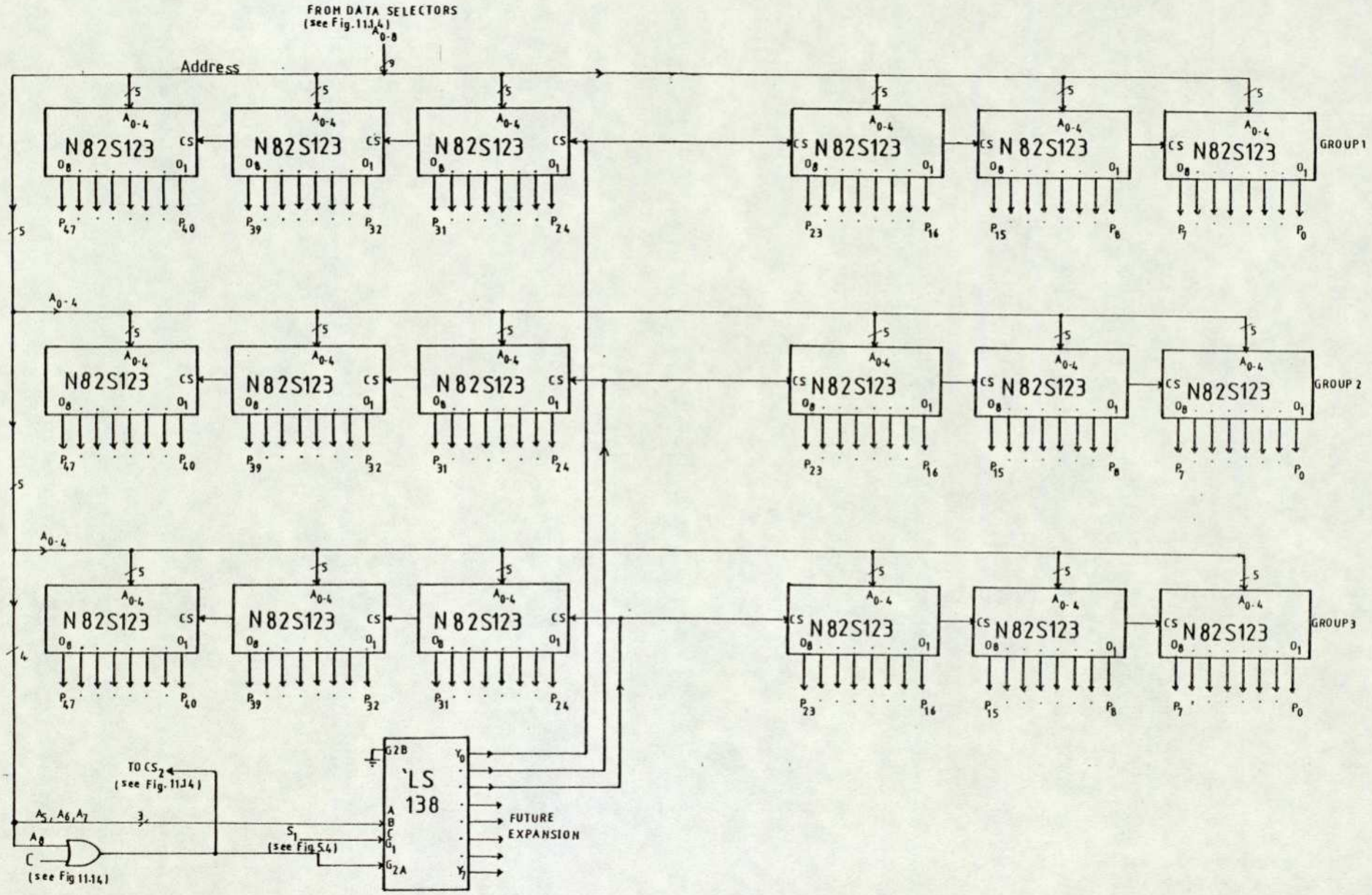


Figure 11.15 The microprogram PROM (P_0 to P_{47} are the 48-bit outputs)

Group	Addressing range (Hex.)
1	000 - 01F
2	020 - 03F
3	040 - 05F

Each group was controlled by using a 1 of 8 decoder (Am74LS138). If the microprogram PROM is selected then the appropriate group will be enabled (CS = LOW) and the microprogram RAM will be disabled (CS_2 = LOW). If on the other hand the microprogram RAM is selected (CS_2 = HIGH) then all three groups of the microprogram PROM will be disabled (CS = HIGH).

11.4.8 The Pipeline Register

The Pipeline Register receives the control data (microinstructions) from the microprogram memory and distributes them to various parts of the microcomputer on the LOW-to-HIGH transition of the system clock. The function of the Pipeline Register is to increase the speed of operation. While the current microinstructions are being executed, the next set of microinstructions will be fetched from the microprogram memory and they will be available at the inputs of the Pipeline Register (note that there is no sequential logic between the microprogram sequencer control inputs and the microprogram memory outputs).

When the source of data for the microprogram sequencer is to be derived from the Mapping Memory, the branch address (BR_0 - BR_{11}) must be forced to a high impedance state to avoid two levels of signals at the 'D' inputs of the microprogram sequencer. The Pipeline Register must therefore incorporate three-state output drive capability which can be controlled from the microprogram sequencer.

The Pipeline Register was designed by using Am29LS18 type registers. These registers receive a 4 bit data and provide two 4-bit outputs 'Q' and 'Y' . When enabled (OE = LOW) , both outputs are the same and when disabled (OE = HIGH) the 'Y' outputs are forced to a high impedance state while the 'Q' outputs provide the normal data. The $\overline{PL}$ output of the microprogram sequencer was used to control the Pipeline Register (see Figure 11.16) .

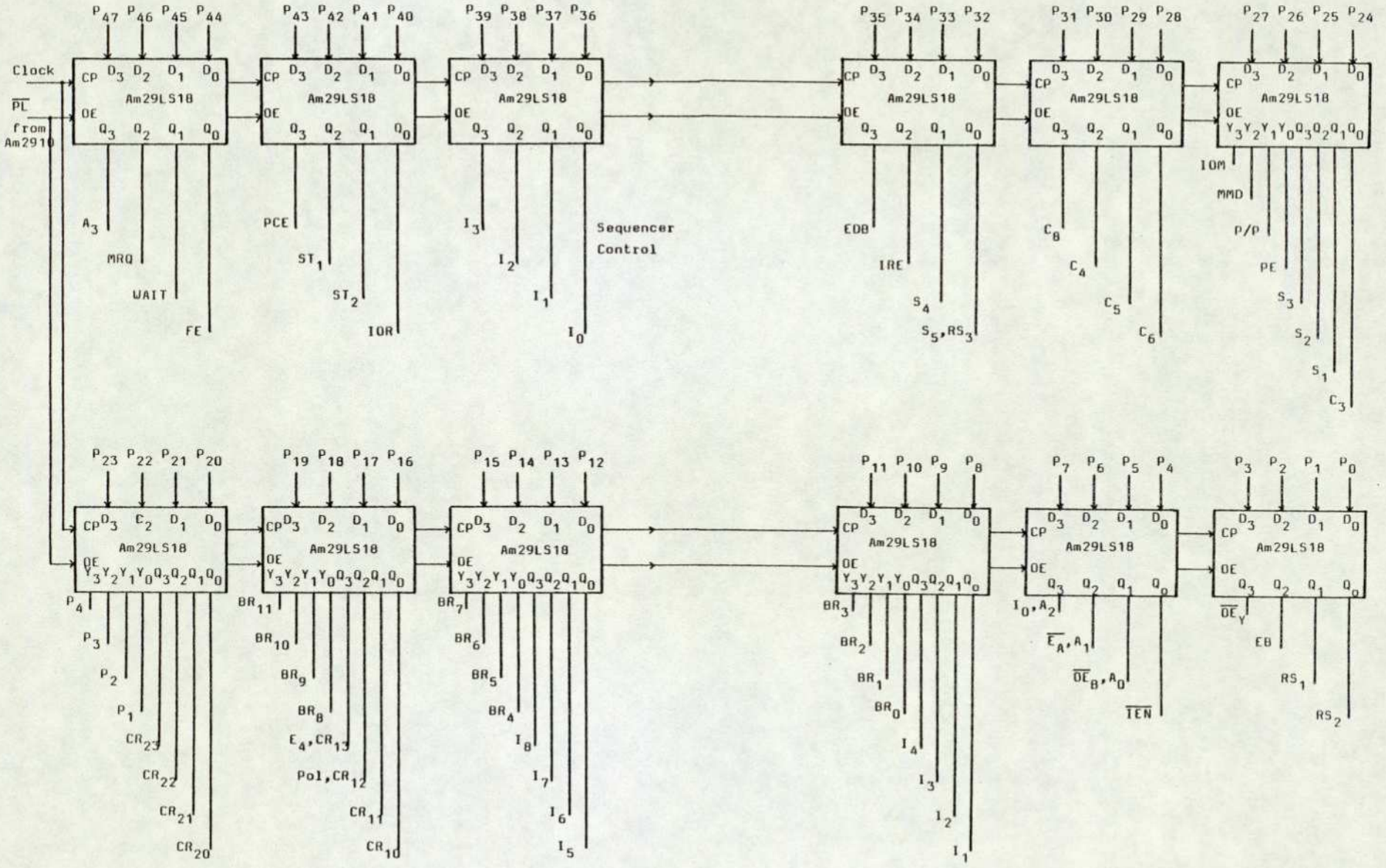


Figure 11.16 The Pipeline Register (P_0 to P_{47} are the 48-bit input to the Pipeline Reg.)

CHAPTER 12

12. MICROPROGRAMMING THE MICROCOMPUTER

12.1 Introduction

The design of the Computer Control Unit completes the basic hardware design of the microcomputer. Because of the complexity involved, it was felt necessary to derive a simple block diagram of the overall microcomputer which would ease the microprogramming. A block diagram of this nature should illustrate the data flow in critical data paths and the destination of the control signals which are output from the Pipeline Register. Fig. 12.1 shows the block diagram derived for this purpose. A reference to this diagram and a knowledge of functions of the various control bits (see Fig. 12.2) should greatly simplify the microprogramming of the microcomputer. This Chapter will describe the microcode implemented in the PROM part of the microprogram memory for operation with 16 bits of data length.

12.2. Instruction Formats

During the initial design phase it was intended to choose an instruction set which would enable the microcomputer to be used as a general purpose machine. This choice was based upon the fact that not all of the future users might wish to generate their own macroinstructions, but some might desire to use the microcomputer as a general purpose 16 bit

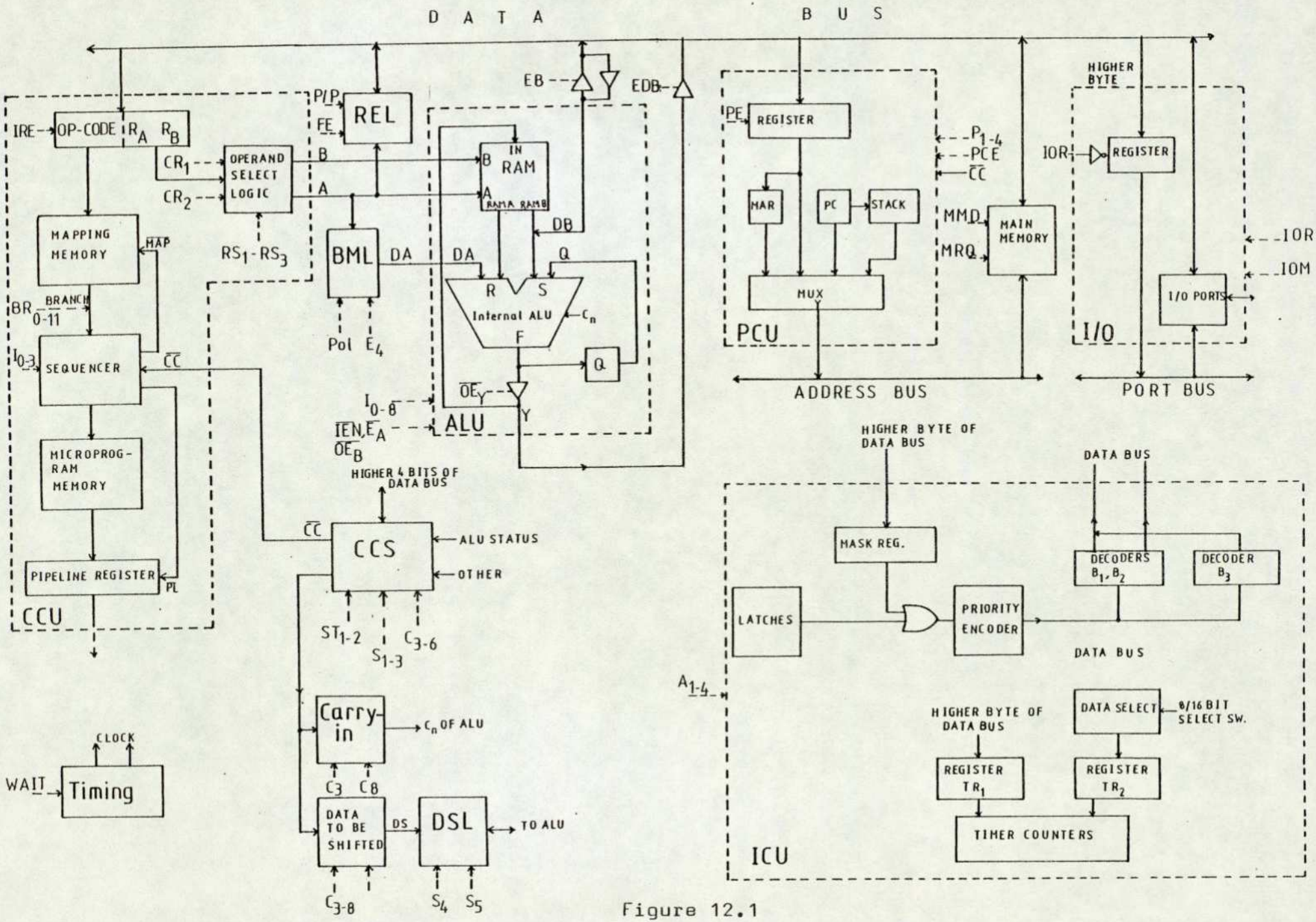

DATA BUS
EB
EDB
P/P
REL
FE
IRE
OP-CODE
R_A
R_B
CR_1
CR_2
OPERAND SELECT LOGIC
B
A
IN RAM
RAMA RAMB
DB
MAPPING MEMORY
MAP
RS_1-RS_3
BML
DA
DA
Q
BR
BRANCH 0-11
R
S
Internal ALU
C_n
F
Q
Pol
E_4
I_{0-3}
SEQUENCER
CC
OE_Y
Y
I_{0-8}
$\overline{IEN}$, E_A
$\overline{OE}_B$
ALU
MICROPROG-RAM MEMORY
HIGHER 4 BITS OF DATA BUS
PIPELINE REGISTER
PL
CCU
CC
CCS
ALU STATUS
OTHER
ST_{1-2}
S_{1-3}
C_{3-6}
Carry-in
C_n OF ALU
CLOCK
WAIT
Timing
C_3
C_8
DATA TO BE SHIFTED
DS
DSL
TO ALU
C_{3-8}
S_4
S_5
PE
REGISTER
P_{1-4}
PCE
CC
MAR
PC
STACK
MMD
MRQ
MAIN MEMORY
MUX
PCU
ADDRESS BUS
HIGHER BYTE
IOR
REGISTER
I/O PORTS
IOR
IOM
I/O
PORT BUS
HIGHER BYTE OF DATA BUS
DATA BUS
MASK REG.
DECODERS B_1, B_2
DECODER B_3
LATCHES
PRIORITY ENCODER
DATA BUS
A_{1-4}
HIGHER BYTE OF DATA BUS
DATA SELECT
8/16 BIT SELECT SW.
REGISTER TR_1
REGISTER TR_2
TIMER COUNTERS
ICU

Figure 12.1

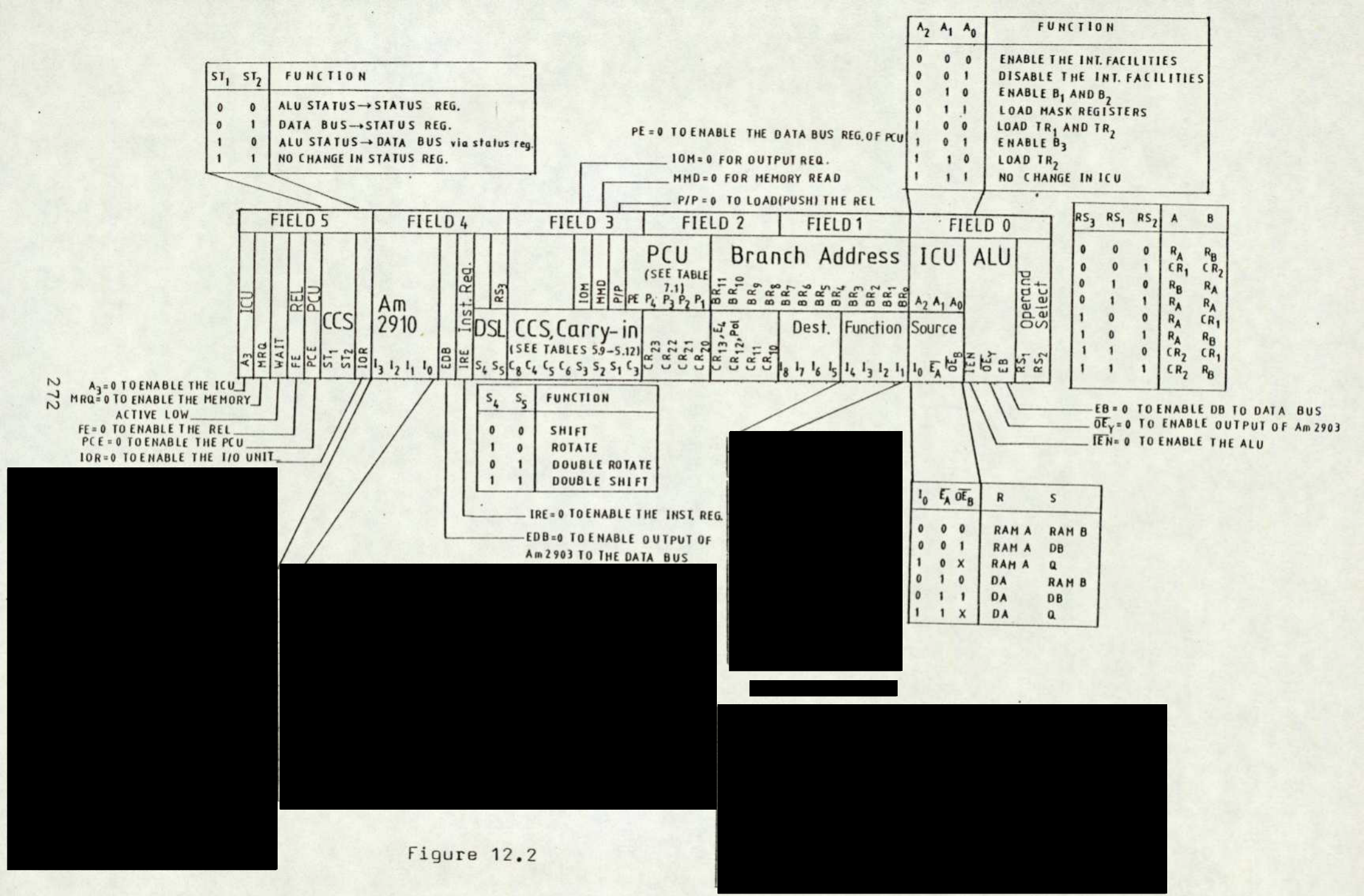

ST1 ST2 FUNCTION
0 0 ALU STATUS → STATUS REG.
0 1 DATA BUS → STATUS REG.
1 0 ALU STATUS → DATA BUS via status reg.
1 1 NO CHANGE IN STATUS REG.
A2 A1 A0 FUNCTION
0 0 0 ENABLE THE INT. FACILITIES
0 0 1 DISABLE THE INT. FACILITIES
0 1 0 ENABLE B1 AND B2
0 1 1 LOAD MASK REGISTERS
1 0 0 LOAD TR1 AND TR2
1 0 1 ENABLE B3
1 1 0 LOAD TR2
1 1 1 NO CHANGE IN ICU
PE = 0 TO ENABLE THE DATA BUS REG. OF PCU
IOM = 0 FOR OUTPUT REQ.
MMD = 0 FOR MEMORY READ
P/P = 0 TO LOAD(PUSH) THE REL
FIELD 5
FIELD 4
FIELD 3
FIELD 2
FIELD 1
FIELD 0
ICU
MRQ
WAIT
REL
FE
PCE
PCU
CCS
ST1
ST2
IOR
Am 2910
EDB
Inst. Req.
IRE
RS3
DSL
CCS, Carry-in
(SEE TABLES 5.9–5.12)
IOM
MMD
P/P
PCU
(SEE TABLE 7.1)
Branch Address
ICU
ALU
Operand Select
A2 A1 A0
Dest.
Function
Source
I8 I7 I6 I5
I4 I3 I2 I1
I0 EA OEB
IEN
OEY
EB
RS1 RS2
RS3 RS1 RS2 A B
0 0 0 RA RB
0 0 1 CR1 CR2
0 1 0 RB RA
0 1 1 RA RA
1 0 0 RA CR1
1 0 1 RA RB
1 1 0 CR2 CR1
1 1 1 CR2 RB
A3 = 0 TO ENABLE THE ICU
MRQ = 0 TO ENABLE THE MEMORY
ACTIVE LOW
FE = 0 TO ENABLE THE REL
PCE = 0 TO ENABLE THE PCU
IOR = 0 TO ENABLE THE I/O UNIT
S4 S5 FUNCTION
0 0 SHIFT
1 0 ROTATE
0 1 DOUBLE ROTATE
1 1 DOUBLE SHIFT
IRE = 0 TO ENABLE THE INST. REG.
EDB = 0 TO ENABLE OUTPUT OF Am2903 TO THE DATA BUS
EB = 0 TO ENABLE DB TO DATA BUS
OEY = 0 TO ENABLE OUTPUT OF Am 2903
IEN = 0 TO ENABLE THE ALU
I0 EA OEB R S
0 0 0 RAM A RAM B
0 0 1 RAM A DB
1 0 X RAM A Q
0 1 0 DA RAM B
0 1 1 DA DB
1 1 X DA Q

Figure 12.2

machine. It was also thought that by choosing a simple general purpose instruction set and implementing it in the microprogram PROM would assist a future user in generating his own microcode, by analysing the ones already implemented in the microprogram PROM.

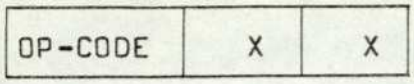
The PROM part of the microprogram memory was loaded with 96 microinstructions (in eighteen 32X8 PROMs) which can execute 62 different macroinstructions (it was the author's opinion that a general purpose microcomputer could be identified by an instruction set of about 60 carefully chosen macroinstructions). These macroinstructions were chosen to be of general use. Some of them were taken from the well known PDP-11 minicomputer [15] and some were invented by the author. Appendix D gives a description of all the fixed macroinstructions implemented. Note that UMM-16 (from: User Microprogrammable Microcomputer) is the name given to the microcomputer designed.

The macroinstructions chosen were based upon a two-address machine architecture [16] and the instruction formats were as follows:-

12.2.1 Single Word Instructions

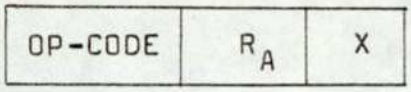
This type of instruction occupies only one location in the main memory. They are divided into three classes depending upon the operand required:-

i. Class 1 -No Operand : These types of instructions do not require any operand to be specified in the machine code. e.g. the subroutine return instruction.



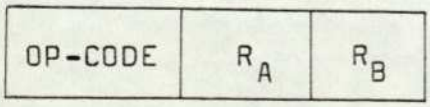
Op-Code = 8 bits
'X' = don't care

ii. Class 2 - One Operand : These types of instructions require one operand to be specified in the machine code. e.g. incrementing or decrementing a register.



Op-Code = 8 bits
R_A = 4 bits
'X' = don't care

iii. Class 3 - Two Operands : These instructions require two operands to be specified in the machine code. e.g. adding two registers. R_A and R_B fields of the operand specify the source register numbers. R_A field also specifies the destination register number.

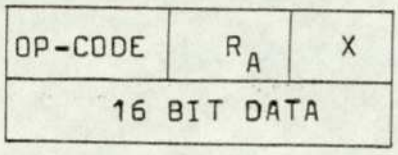


Op-Code = 8 bits
Operand R_A = 4 bits
Operand R_B = 4 bits

12.2.2 Two Word Instructions

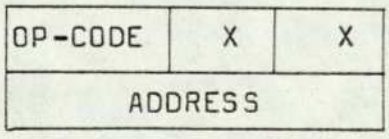
This type of instructions occupy two successive locations in the main memory. Two word instructions are divided into two classes depending upon the nature of the second word:-

i. Class 1 - Second Word is a Data : Here, the second word is data* and it operates upon the register addressed by the R_A field of the operand. e.g. loading a 16 bit data to a register.



Op-Code = 8 bits
Operand R_A = 4 bits
'X' = don't care

ii. Class 2 - Second Word is an Address : In this type of instructions, the second word is an address (a 'jump' address or an address for an input-output port)



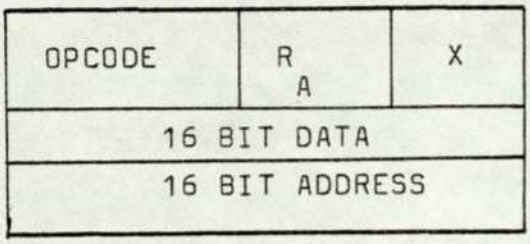
Op-Code = 8 bits
Address = 16 bits (8 bits for an I/O port)

12.2.3 Three Word Instructions

This type of instructions occupy three successive

* usually called "Immediate addressing" or "Immediate data"

locations in the main memory. The first word specifies the operation to be performed and the register number. The second word is data which is compared to the contents of this register. The third word is a jump address. If the register contains the required data then a 'jump' will be performed, such as 'jump on equal to'.



Op-Code= 8 bits

Operand R_A= 4 bits

12.3 The Microcode

The microcode implemented in PROM part of the microcomputer is shown in Fig. 12.1 to 12.6. Each micro-instruction is executed in a single clock cycle. A list of all the 96 microinstructions implemented in the micro-program PROM are given in Appendix B. This section gives a description of some of the microinstructions implemented (mI_{nnn} = microinstruction at microprogram memory address nnn):-

mI_{ooo}: This is the first microinstruction executed when the microcomputer is RESET. This microinstruction initializes the system. The Program Counter Register (PC) is forced to zero and interrupts are disabled. The microprogram sequencer pushes the first microinstruction of the macroinstruction

INSTRUCTION	OP-CODE	ADDRESS	FIELD 5								FIELD 4								FIELD 3								FIELD 2								FIELD 1								FIELD 0							
			ICU			REL	PCU	CCS			Am 2910					Inst. Reg.	DSL									IOM	MMD	P/P	PCU					Branch Address								ICU			ALU				Operand Select	
			A_3	MRQ	WAIT	FE	PCE	ST_1	ST_2	IOR	I_3	I_2	I_1	I_0	EDB	IRE	S_4	S_5	RS_3	C_8	C_4	C_5	C_6	S_3	S_2	S_1	C_3	PE	P_4	P_3	P_2	P_1	BR_{11}	BR_{10}	BR_9	BR_8	BR_7	BR_6	BR_5	BR_4	BR_3	BR_2	BR_1	BR_0	A_2	A_1	A_0				RS_1	RS_2
			CCS, Carry-in																												Dest.				Function				Source								
			47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	.	0 0 0	0	1	1	1	0	X	X	1	0	1	0	0	1	1	X	X	X	X	X	X	X	X	X	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	0	0	1	1	1	1	X	X
FETCH 1	.	0 0 1	1	0	1	1	0	1	1	1	1	1	1	0	1	0	X	X	X	X	X	X	0	X	X	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
FETCH 2	.	0 0 2	1	1	1	1	1	1	1	1	0	0	1	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
Interrupt-	0 0	0 0 3	0	1	1	1	1	1	1	1	1	1	1	0	1	1	X	X	X	X	X	X	X	X	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	0	1	1	1	X	X
Service -	.	0 0 4	0	0	1	1	0	1	1	1	1	1	1	0	1	1	X	X	X	X	X	X	0	X	0	1	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	1	1	1	1	X	X
Routine	.	0 0 5	1	0	1	1	0	1	1	1	1	1	1	0	1	0	X	X	X	1	1	1	X	0	X	1	0	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
~	.	0 0 6	1	1	1	1	1	1	1	1	0	0	1	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
R_A . AND. $R_B \rightarrow R_A$	0 1	0 0 7	1	1	1	1	1	0	0	1	1	1	0	1	1	1	X	0	X	1	1	1	X	X	X	1	X	X	X	X	X	X	X	X	0	1	0	0	1	1	0	0	0	0	0	0	0	1	1	0
R_A . OR . $R_B \rightarrow R_A$	0 2	0 0 8	1	1	1	1	1	0	0	1	1	1	0	1	1	1	X	0	X	1	1	1	X	X	X	1	X	X	X	X	X	X	X	X	0	1	0	0	1	1	1	1	0	0	0	0	0	1	1	0
R_A . EX-OR . $R_B \rightarrow R_A$	0 3	0 0 9	1	1	1	1	1	0	0	1	1	1	0	1	1	1	X	0	X	1	1	1	X	X	X	1	X	X	X	X	X	X	X	X	0	1	0	0	1	0	1	1	0	0	0	0	0	1	1	0
$\overline{R_A} \rightarrow R_A$	0 4	0 0 A	1	1	1	1	1	0	0	1	1	1	0	1	1	1	X	0	1	1	1	1	0	0	1	1	X	X	X	X	X	X	X	X	0	1	0	0	0	1	0	1	0	0	0	0	0	1	1	1
$\overline{Q} \rightarrow Q$	0 5	0 0 B	1	1	1	1	1	0	0	1	1	1	0	1	1	1	X	X	1	1	1	1	0	0	1	1	X	X	X	X	X	X	X	X	0	1	1	0	0	1	0	1	1	1	X	0	1	1	X	X
$\overline{R_A}$ +1 $\rightarrow R_A$	0 6	0 0 C	1	1	1	1	1	0	0	1	1	1	0	1	1	1	X	0	1	1	1	1	0	0	0	1	X	X	X	X	X	X	X	X	0	1	0	0	0	1	0	1	0	0	0	0	0	1	1	1
R_A . AND . Im $\rightarrow R_A$	0 7	0 0 D	1	0	1	1	0	0	0	1	1	1	0	1	1	1	X	0	X	1	1	1	X	0	X	1	0	0	0	1	X	X	X	X	0	1	0	0	1	1	0	0	0	0	1	0	0	1	1	1
R_A . OR . Imm $\rightarrow R_A$	0 8	0 0 E	1	0	1	1	0	0	0	1	1	1	0	1	1	1	X	0	X	1	1	1	X	0	X	1	0	0	0	1	X	X	X	X	0	1	0	0	1	1	1	1	0	0	1	0	0	1	1	1
R_A . EX-OR . Imm $\rightarrow R_A$	0 9	0 0 F	1	0	1	1	0	0	0	1	1	1	0	1	1	1	X	0	X	1	1	1	X	0	X	1	0	0	0	1	X	X	X	X	0	1	0	0	1	0	1	1	0	0	1	0	0	1	1	1

Figure 12.1 Microcode of the fixed instructions

cont...

INSTRUCTION	OP CODE	ADDRESS	FIELD 5								FIELD 4								FIELD 3								FIELD 2								FIELD 1								FIELD 0							
			ICU			REL	PCU	CCS			Am 2910					Inst. Reg.	DSL								PCU					Branch Address								Dest.				Function				ICU			ALU			Operand Select	
			A_3	MRQ	WAIT	FE	PCE	ST_1	ST_2	IOR	I_3 I_2 I_1 I_0	EDB	IRE	RS_3		S_4 S_5	IOM	MMD	P/P	C_8 C_4 C_5 C_6 S_3 S_2 S_1 C_3	PE	P_4 P_3 P_2 P_1	BR_{11} BR_{10} BR_9 BR_8	BR_7 BR_6 BR_5 BR_4 BR_3 BR_2 BR_1 BR_0	CR_{23} CR_{22} CR_{21} CR_{20}	CR_{13}, E_4 CR_{12}, Pol CR_{11} CR_{10}	I_8 I_7 I_6 I_5	I_4 I_3 I_2 I_1	I_0	$\overline{E_A}$	$\overline{OE_B}$	IEN	$\overline{OE_Y}$	EB	RS_1	RS_2
			47	46	45	44	43	42	41	40	39 38 37 36	35	34			33 32				31 30 29 28 27 26 25 24					23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7	6	5	4	3	2	1	0
$R_A + R_B \rightarrow R_A$	0A	0,1,0	1	1	1	1	1	0	0	1	1 1 0 1	1	1			X 0				1 1 1 1 0 0 1 1					X X X X	X X X X	0 1 0 0	0 0 1 1	0	0	0	0	0	1	1	0
$Q + R_A \rightarrow R_B$	0B	0,1,1	1	1	1	1	1	0	0	1	1 1 0 1	1	1			X 0				1 1 1 1 0 0 1 1					X X X X	X X X X	0 1 0 0	0 0 1 1	1	0	X	0	0	1	0	0
$R_A - R_B \rightarrow R_A$	0C	0,1,2	1	1	1	1	1	0	0	1	1 1 0 1	1	1			X 0				1 1 1 1 0 0 0 1					X X X X	X X X X	0 1 0 0	0 0 0 1	0	0	0	0	0	1	1	0
$R_A + 1 \rightarrow R_A$	0D	0,1,3	1	1	1	1	1	0	0	1	1 1 0 1	1	1			X 0				1 1 1 1 0 0 0 1					X X X X	X X X X	0 1 0 0	0 1 0 0	0	1	0	0	0	1	1	0
$Q + 1 \rightarrow Q$	0E	0,1,4	1	1	1	1	1	0	0	1	1 1 0 1	1	1			X X				1 1 1 1 0 0 0 1					X X X X	X X X X	0 1 1 0	0 1 0 0	1	1	X	0	1	1	X	X
$R_A - 1 \rightarrow R_A$	0F	0,1,5	1	1	1	1	1	0	0	1	1 1 0 1	1	1			X 0				1 1 1 1 0 0 1 1					X X X X	X X X X	0 1 0 0	0 0 0 1	0	1	0	0	0	1	1	0
$Q - 1 \rightarrow Q$	10	0,1,6	1	1	1	1	1	0	0	1	1 1 0 1	1	1			X X				1 1 1 1 0 0 1 1					X X X X	X X X X	0 1 1 0	0 0 0 1	1	1	X	0	1	1	X	X
$0 \rightarrow Q$	11	0,1,7	1	1	1	1	1	0	0	1	1 1 0 1	1	1			X X				X 1 1 1 X X X 1					X X X X	0 0 X X	0 1 1 0	1 1 0 0	1	1	X	0	1	1	X	X
$0 \rightarrow R_A$	12	0,1,8	1	1	1	1	1	0	0	1	1 1 0 1	1	1			X 0				X 1 1 1 X X X 1					X X X X	0 0 X X	0 1 0 0	1 1 0 0	0	1	0	0	0	1	1	0
$R_A + Imm \rightarrow R_A$	13	0,1,9	1	0	1	1	0	0	0	1	1 1 0 1	1	1			X 0				1 1 1 1 0 0 1 1					0 0 0 1	X X X X	0 1 0 0	0 0 1 1	0	0	1	0	0	1	1	1
$R_A - Imm \rightarrow R_A$	14	0,1A	1	0	1	1	0	0	0	1	1 1 0 1	1	1			X 0				1 1 1 1 0 0 0 1					0 0 0 1	X X X X	0 1 0 0	0 0 1 0	0	0	1	0	0	1	1	1
$Imm. \rightarrow R_A$	15	0,1B	1	0	1	1	0	1	1	1	1 1 0 1	1	1			X 0				X 1 1 1 X 0 X 1					0 0 0 1	0 0 X X	0 1 0 0	1 1 1 1	0	1	1	0	0	1	1	0
$Imm. \rightarrow Q$	16	0,1C	1	0	1	1	0	1	1	1	1 1 0 1	1	1			X X				X 1 1 1 X 0 X 1					0 0 0 1	0 0 X X	0 1 1 0	1 1 1 1	0	1	1	0	1	1	X	X
$R_B \rightarrow R_A$	17	0,1D	1	1	1	1	1	1	1	1	1 1 1 0	1	1			X 0				X X X X X X X X					X X X X	X X X X	0 1 0 0	1 0 0 0	0	0	0	0	0	1	1	0
		0,1E	1	1	1	1	1	1	1	1	1 1 0 1	1	1			X 0				X 1 1 1 X X X 1					X X X X	X X X X	0 1 0 0	1 1 1 1	0	0	0	0	0	1	1	0
$Q \rightarrow R_A$	18	0,1,F	1	1	1	1	1	1	1	1	1 1 0 1	1	1			X 0				X 1 1 1 X X X 1					X X X X	0 0 X X	0 1 0 0	1 1 1 1	1	1	X	0	0	1	1	0

Figure 12.2 cont.

cont...

	OP-CODE	ADDRESS	FIELD 5								FIELD 4								FIELD 3								FIELD 2				FIELD 1				FIELD 0							
			ICU			REL	PCU	CCS			Am 2910					Inst. Req.	DSL			IOM	MMD	P/P	PCU				Branch Address								ICU			ALU				Operand Select	
			A_3	MRQ	WAIT	FE	PCE	ST_1	ST_2	IOR	I_3	I_2	I_1	I_0	EDB	IRE	S_4	S_5	RS_3	C_8	C_4	C_5	C_6	S_3	S_2	S_1	C_3	PE	P_4	P_3	P_2	P_1	BR_{11}	BR_{10}	BR_9	BR_8	BR_7	BR_6	BR_5	BR_4	BR_3	BR_2	BR_1	BR_0	A_2	A_1	A_0	$\overline{TEN}$	$\overline{OE_Y}$	EB	RS_1	RS_2
			CR_{23}	CR_{22}	CR_{21}	CR_{20}	CR_{13}, E_4	CR_{12}, Pol	CR_{11}	CR_{10}	I_8	I_7	I_6	I_5	Dest.		I_4	I_3	I_2	I_1	Function		I_0	E_A	$\overline{OE_B}$																							
INSTRUCTION			47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$R_A \rightarrow Q$	19	0,20	1	1	1	1	1	1	1	1	1	1	0	1	1	1	X	0	X	1	1	1	X	X	X	1	X	X	X	X	0	0	X	X	0	1	1	0	1	1	1	1	0	1	0	0	1	1	1	0
Push $R_B \rightarrow R_A$	1A	0,21	1	1	1	0	1	1	1	1	1	1	0	1	0	1	X	0	X	1	1	1	X	X	0	1	X	X	X	X	0	0	X	X	0	1	0	0	1	1	1	1	0	1	0	0	0	1	0	0
Pop $R_A \rightarrow R_B$	1B	0,22	1	1	1	0	1	0	0	1	1	1	0	1	1	1	X	0	X	1	1	1	X	X	1	1	X	X	X	X	0	0	X	X	0	1	0	0	1	1	1	1	0	1	1	0	0	1	0	0
Status $\rightarrow R_A$	1C	0,23	1	1	1	1	1	1	0	1	1	1	0	1	1	1	X	0	X	1	1	1	X	X	X	1	X	X	X	X	0	0	X	X	0	1	0	0	1	1	1	1	0	1	1	0	0	1	1	1
$R_A \rightarrow$ Status	1D	0,24	1	1	1	1	1	0	1	1	1	1	0	1	0	1	X	0	X	1	1	1	X	X	X	1	X	X	X	X	0	0	X	X	0	1	0	0	1	1	1	1	0	1	0	0	0	1	1	1
OUTS R_A	1E	0,25	1	0	1	1	0	1	1	1	1	1	1	0	1	1	X	X	X	X	X	X	X	0	X	X	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
	.	0,26	1	1	1	1	1	1	1	0	1	1	0	1	0	1	X	0	X	1	1	1	0	X	X	1	X	X	X	X	0	0	X	X	0	1	0	0	1	1	1	1	0	1	0	0	0	1	1	1
INS R_A	1F	0,27	1	0	1	1	0	1	1	1	1	1	1	0	1	1	X	X	X	X	X	X	X	0	X	X	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
	.	0,28	1	1	1	1	1	0	0	0	1	1	0	1	1	1	X	0	X	1	1	1	1	X	X	1	X	X	X	X	0	0	X	X	0	1	0	0	1	1	1	1	0	1	1	0	0	1	1	1
R_A+Mem. $\rightarrow R_A$	20	0,29	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	0	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	0	0	1
	.	0,2A	1	0	1	1	0	0	0	1	1	1	1	0	1	1	X	0	X	X	X	X	X	0	X	1	0	1	1	0	X	X	X	X	0	1	0	0	0	0	1	1	0	0	1	0	0	1	1	1
	.	0,2B	1	1	1	1	1	1	1	1	1	1	0	1	1	1	X	0	1	1	1	1	0	0	0	1	1	1	1	1	0	0	X	X	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	1
Mem $\rightarrow R_A$	21	0,2C	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	0	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	0	0	1
	.	0,2D	1	0	1	1	0	0	0	1	1	1	1	0	1	1	X	0	X	X	X	X	X	0	X	1	0	1	1	0	0	0	X	X	0	1	0	0	1	1	1	1	0	1	1	0	0	1	1	1
	.	0,2E	1	1	1	1	1	1	1	1	1	1	0	1	1	1	X	0	1	1	1	1	0	0	0	1	1	1	1	1	X	X	X	X	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	1
WAIT	22	0,2F	1	1	0	1	1	1	1	1	1	1	0	1	1	1	X	X	X	1	1	1	X	X	X	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X

Figure 12.3 cont.

cont...

INSTRUCTION	OPCODE	ADDRESS	FIELD 5								FIELD 4								FIELD 3								FIELD 2								FIELD 1								FIELD 0							
			ICU			REL	PCU	CCS			Am 2910					Inst. Req.	DSL		CCS, Carry-in								PCU				Branch Address								ICU			ALU				Operand Select	
			A_3	MRQ	WAIT	FE	PCE	ST_1	ST_2	IOR	I_3	I_2	I_1	I_0	EDB	IRE	S_4	S_5	C_8	C_4	C_5	C_6	S_3	S_2	S_1	C_3	CR_{23}	CR_{22}	CR_{21}	CR_{20}	CR_{13}, E_4	CR_{12}, Pol	CR_{11}	CR_{10}	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	$\overline{E_A}$	$\overline{OE_B}$	$\overline{IEN}$	$\overline{OE_Y}$	EB	RS_1	RS_2
			47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$R_A - Imm.$	23	030	1	0	1	1	0	0	0	1	1	1	0	1	1	1	X	0	1	1	1	1	0	0	0	1	0	0	0	1	X	X	X	X	0	1	0	0	0	0	1	0	0	0	1	1	0	1	1	1
$R_A - R_B$	24	031	1	1	1	1	1	0	0	1	1	1	0	1	1	1	X	0	1	1	1	1	0	0	0	1	X	X	X	X	X	X	X	X	0	1	0	0	0	0	0	1	0	0	0	1	0	1	1	0
$R_A - Mem.$	25	032	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	0	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	0	0	1
	.	033	1	0	1	1	0	0	0	1	1	1	1	0	1	1	X	0	1	X	X	X	0	0	0	1	0	1	1	0	X	X	X	X	0	1	0	0	0	0	1	0	0	0	1	1	0	1	1	1
	.	034	1	1	1	1	1	1	1	1	1	1	0	1	1	1	X	0	1	1	1	1	0	0	0	1	1	1	1	1	X	X	X	X	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	1
JUMP	26	035	1	0	1	1	0	1	1	1	1	1	1	0	1	1	X	X	X	X	X	X	X	0	X	0	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
	.	036	1	0	1	1	0	1	1	1	1	1	1	0	1	0	X	X	X	X	X	X	0	X	X	0	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
	.	037	1	1	1	1	1	1	1	1	0	0	1	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
JUMP IF NON-ZERO	27	038	1	1	1	1	1	1	1	1	0	0	1	1	1	1	X	X	X	0	0	1	X	X	X	0	X	X	X	X	0	0	0	0	0	0	1	1	0	1	0	1	X	X	X	1	1	1	X	X
	.	039	1	1	1	1	0	1	1	1	1	1	0	1	1	1	X	X	X	1	1	1	X	X	X	1	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
JUMP IF ZERO	28	03A	1	1	1	1	1	1	1	1	0	0	1	1	1	1	X	X	X	0	0	1	X	X	X	1	X	X	X	X	0	0	0	0	0	0	1	1	0	1	0	1	X	X	X	1	1	1	X	X
	.	03B	1	1	1	1	0	1	1	1	1	1	0	1	1	1	X	X	X	1	1	1	X	X	X	1	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
JUMP IF POSITIVE	29	03C	1	1	1	1	1	1	1	1	0	0	1	1	1	1	X	X	X	1	0	0	X	X	X	0	X	X	X	X	0	0	0	0	0	0	1	1	0	1	0	1	X	X	X	1	1	1	X	X
	.	03D	1	1	1	1	0	1	1	1	1	1	0	1	1	1	X	X	X	1	1	1	X	X	X	1	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
JUMP IF NEGATIVE	2A	03E	1	1	1	1	1	1	1	1	0	0	1	1	1	1	X	X	X	1	0	0	X	X	X	1	X	X	X	X	0	0	0	0	0	0	1	1	0	1	0	1	X	X	X	1	1	1	X	X
	.	03F	1	1	1	1	0	1	1	1	1	1	0	1	1	1	X	X	X	1	1	1	X	X	X	1	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X

Figure 12.4 cont.

cont...

INSTRUCTION	OP CODE	ADDRESS	FIELD 5								FIELD 4								FIELD 3												FIELD 2								FIELD 1								FIELD 0							
			ICU			REL	PCU	CCS			Am 2910					Inst. Reg.	DSL								IOM	MMD	P/P	PCU					Branch Address								Dest.				Function				ICU			ALU				Operand Select	
			A_3	MRQ	WAIT	FE	PCE	ST_1	ST_2	IOR	I_3	I_2	I_1	I_0	EDB	IRE	S_4	S_5	C_8	C_4	C_5	C_6	S_3	S_2	S_1	C_3					PE	P_4	P_3	P_2	P_1	BR_{11}	BR_{10}	BR_9	BR_8	BR_7	BR_6	BR_5	BR_4	BR_3	BR_2	BR_1	BR_0	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	$\overline{E_A}$	$\overline{OE_B}$	$\overline{IEN}$	$\overline{OE_Y}$	EB	RS_1	RS_2
			47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JUMP IF EQUAL	2B	0,4,0	1	0	1	1	0	0	0	1	1	1	1	0	1	1	X	0	1	1	1	1	0	0	0	1	0	0	0	1	X	X	X	X	0	1	0	0	0	0	1	0	0	0	1	1	0	1	1	1
	.	0,4,1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	X	X	X	0	0	1	X	X	X	1	X	X	X	X	0	0	0	0	0	0	1	1	0	1	0	1	X	X	X	1	1	1	X	X
	.	0,4,2	1	1	1	1	0	1	1	1	1	1	0	1	1	1	X	X	X	1	1	1	X	X	X	1	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
DECREMENT & SKIP	2C	0,4,3	1	1	1	1	1	0	0	1	1	1	1	0	1	1	X	0	1	1	1	1	0	0	1	1	X	X	X	X	0	0	X	X	0	1	0	0	0	0	0	1	0	1	0	0	0	1	1	0
	.	0,4,4	1	1	1	1	1	1	1	1	0	0	1	1	1	1	X	X	X	0	0	1	X	X	X	0	X	X	X	X	0	0	0	0	0	0	1	1	0	1	0	1	X	X	X	1	1	1	X	X
	.	0,4,5	1	1	1	1	0	1	1	1	1	1	0	1	1	1	X	X	X	1	1	1	X	X	X	1	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
JUMP IF LESS THAN	2D	0,4,6	1	0	1	1	0	0	0	1	1	1	1	0	1	1	X	0	1	1	1	1	0	0	0	1	0	0	0	1	X	X	X	X	0	1	0	0	0	0	1	0	0	0	1	1	0	1	1	1
JUMP IF NON-CARRY	2E	0,4,7	1	1	1	1	1	1	1	1	0	0	1	1	1	1	X	X	X	0	1	0	X	X	X	0	X	X	X	X	0	0	0	0	0	0	1	1	0	1	0	1	X	X	X	1	1	1	X	X
	.	0,4,8	1	1	1	1	0	1	1	1	1	1	0	1	1	1	X	X	X	1	1	1	X	X	X	1	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
CALL TO SUB.	2F	0,4,9	1	0	1	1	0	1	1	1	1	1	1	0	1	1	X	X	X	X	X	X	X	0	X	0	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
	.	0,4,A	1	0	1	1	0	1	1	1	1	1	1	0	1	0	X	X	X	1	1	1	X	0	X	1	1	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
	.	0,4,B	1	1	1	1	1	1	1	1	0	0	1	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
$R_A \rightarrow$ Mem.	30	0,4,C	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	0	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	0	0	1
	.	0,4,D	1	0	1	1	0	1	1	1	1	1	1	0	0	1	X	0	X	X	X	X	X	1	X	1	0	1	1	0	0	0	X	X	0	1	0	0	1	1	1	1	0	1	0	0	0	1	1	1
	.	0,4,E	1	1	1	1	1	1	1	1	1	1	0	1	1	1	X	0	1	1	1	1	0	0	0	1	1	1	1	1	X	X	X	X	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	1
$R_A \rightarrow$ PC	31	0,4,F	1	1	1	1	1	1	1	1	1	1	1	0	0	1	X	0	X	X	X	X	X	X	X	0	X	X	X	X	0	0	X	X	0	1	0	0	1	1	1	1	0	1	0	0	0	1	1	1

Figure 12.5 cont.

cont...

	OP-CODE	ADDRESS	FIELD 5								FIELD 4								FIELD 3								FIELD 2								FIELD 1								FIELD 0							
			A_3 ICU	MRQ	WAIT	FE REL	PCE PCU	ST_1 CCS	ST_2 CCS	IOR	I_3 Am 2910	I_2 Am 2910	I_1 Am 2910	I_0 Am 2910	EDB	IRE Inst. Reg.	S_4 DSL	S_5 DSL	C_8 CCS, Carry-in	C_4 CCS, Carry-in	C_5 CCS, Carry-in	C_6 CCS, Carry-in	S_3 CCS, Carry-in	S_2 CCS, Carry-in	S_1 CCS, Carry-in	C_3 CCS, Carry-in	RS_3	IOM	MMD	P/P	PE PCU	P_4 PCU	P_3 PCU	P_2 PCU	P_1 PCU	CR_{23}	CR_{22}	CR_{21}	CR_{20}	BR_{11} Branch Address	BR_{10} Branch Address	BR_9 Branch Address	BR_8 Branch Address	BR_7 Branch Address	BR_6 Branch Address	BR_5 Branch Address	BR_4 Branch Address	BR_3 Branch Address	BR_2 Branch Address	BR_1 Branch Address	BR_0 Branch Address	CR_{13}, E_4	CR_{12}, Pol	CR_{11}	CR_{10}	I_8 Dest.	I_7 Dest.	I_6 Dest.	I_5 Dest.	I_4 Function	I_3 Function	I_2 Function	I_1 Function	A_2 ICU	A_1 ICU	A_0 ICU	I_0 Source	$\overline{E_A}$ Source	$\overline{OE_B}$ Source	IEN	$\overline{OE_Y}$ ALU	EB ALU	RS_1 Operand Select	RS_2 Operand Select
INSTRUCTION			47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		0,50	1	0	1	1	0	1	1	1	0	0	1	1	1	0	X	X	X	1	1	1	X	X	X	0	X	X	1	1	0	0	0	0	0	0	1	1	0	1	1	1	X	X	X	1	1	1	X	X
RETURN	32	0,51	1	1	1	1	0	1	1	1	1	1	1	0	1	1	X	X	X	X	X	X	X	X	X	X	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
		0,52	1	0	1	1	0	1	1	1	1	1	1	0	1	0	X	X	X	1	1	1	X	X	X	1	1	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
		0,53	1	1	1	1	1	1	1	1	0	0	1	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
ROT. RIGHT 2	33	0,54	1	1	1	1	1	0	0	1	1	1	1	0	1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X	0	0	X	X	0	0	0	1	1	1	1	1	0	1	0	0	0	1	1	1
ROT. RIGHT 1	3D	0,55	1	1	1	1	1	0	0	1	1	1	0	1	1	1	1	0	X	1	1	1	X	X	X	1	X	X	X	X	0	0	X	X	0	0	0	1	1	1	1	1	0	1	0	0	0	1	1	1
Enable Int.	34	0,56	0	1	1	1	1	1	1	1	1	1	0	1	1	1	X	X	X	1	1	1	X	X	X	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	1	1	1	X	X
Disable Int.	35	0,57	0	1	1	1	1	1	1	1	1	1	0	1	1	1	X	X	X	1	1	1	X	X	X	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	1	1	1	1	X	X
Load Mask Reg.	36	0,58	0	0	1	1	0	1	1	1	1	1	0	1	1	1	X	X	X	1	1	1	X	X	X	1	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	1	1	1	X	X
Load Timer Counter	37	0,59	0	0	1	1	0	1	1	1	1	1	0	1	1	1	X	X	X	1	1	1	X	X	X	1	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	1	0	0	1	1	1	X	X
Shift Right "1"	38	0,5A	1	1	1	1	1	0	0	1	1	1	0	1	1	1	0	0	X	1	1	1	0	0	0	1	X	X	X	X	0	0	X	X	0	0	0	1	1	1	1	1	0	1	0	0	0	1	1	1
Shift Left "0"	39	0,5B	1	1	1	1	1	0	0	1	1	1	0	1	1	1	0	0	X	1	1	1	0	0	1	1	X	X	X	X	0	0	X	X	1	0	0	1	1	1	1	1	0	1	0	0	0	1	1	1
Return from Int.	3A	0,5C	1	1	1	1	0	1	1	1	0	0	1	1	1	1	X	X	X	1	1	1	X	X	X	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	X	X	X	1	1	1	X	X
ROT. LEFT 1	3E	0,5D	1	1	1	1	1	0	0	1	1	1	0	1	1	1	1	0	X	1	1	1	X	X	X	1	X	X	X	X	0	0	X	X	1	0	0	1	1	1	1	1	0	1	0	0	0	1	1	1
No operation	3B	0,5E	1	1	1	1	1	1	1	1	1	1	0	1	1	1	X	X	X	1	1	1	X	X	X	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
Shift Right "0"	3C	0,5F	1	1	1	1	1	0	0	1	1	1	0	1	1	1	0	0	X	1	1	1	0	0	1	1	X	X	X	X	0	0	X	X	0	0	0	1	1	1	1	1	0	1	0	0	0	1	1	1

Figure 12.6 cont.

FETCH cycle (at microprogram address 001)onto the stack for future use and continues to the next microinstruction.

mI_{001}-mI_{002}: These two microinstructions implement the macroinstruction FETCH cycle. At the end of every set of microinstructions a jump to this location should be made in order to fetch the next sequential macroinstruction from the main memory. At mI_{001},the PC is incremented by one and a memory read operation is performed (MRQ = LOW, MMD = LOW). Also the Instruction Register is enabled (IRE = LOW). By the end of this cycle the memory data will be available at the input of the Instruction Register. On the LOW-to-HIGH transition of the system clock, the Instruction Register will be loaded from the Data Bus and mI_{002} will be output from the Pipeline Register. This microinstruction will control the microprogram sequencer so that the Mapping Memory will be enabled and an address will be provided to the microprogram memory corresponding to the macroinstruction to be executed. By the end of this cycle, the first microinstruction of the current macroinstruction will be at the input of the Pipeline Register and it will be distributed to various subsystems of the microcomputer on the next clock LOW-to-HIGH transition.

mI_{007}: This microinstruction implements the macroinstruction (R_A). AND. $(R_B) \longrightarrow R_A$. The microoperations performed are as follows:-

Operand Select Logic : $A = R_B$, $B = R_A$

ALU Source : R = RAM A, S = RAM B

ALU Function : F = R.AND.S

ALU Destination : $F \longrightarrow Y$

$Y \longrightarrow$ RAM B

Condition Code Select : $\overline{CC} = 1$

Next microinstruction : $\longrightarrow mI_{001}$

mI_{00A}: This microinstruction implements the macroinstruction $(\overline{R}_A) \longrightarrow R_A$. The microoperations performed are as follows:

Operand Select Logic : $A = R_A$, $B = R_A$

ALU Carry-in : $C_n = 0$

ALU Source : R = RAM A, S = RAM B

ALU Function : $F = \overline{S} + C_n$

ALU Destination : $F \longrightarrow Y$

$Y \longrightarrow$ RAM B

Condition Code Select : $\overline{CC} = 1$

Next microinstruction : $\rightarrow mI_{001}$

mI_{000}: This microinstruction implements the macroinstruction (R_A).AND.Imm $\longrightarrow R_A$ where 'Imm' is a 16 bit immediate data contained in the second word of the macroinstruction. The microoperations performed are as foolows:-

Program Control Unit : $(PC) + 1 \longrightarrow PC$

Memory Unit : Memory READ request

Operand Select Logic : $A = R_A$, $B = R_A$

ALU Source : R = RAM A, S = DB

ALU Function : F = R.AND.S

ALU Destination : $F \longrightarrow Y$

$Y \longrightarrow$ RAM B

Condition Code Select : $\overline{CC} = 1$

Next microinstruction : $\longrightarrow mI_{001}$

mI_{010}: This microinstruction implements the macroinstruction $(R_A) + (R_B) \longrightarrow R_A$. The microoperations performed are as follows:-

Operand Select Logic : $A = R_B$, $B = R_A$

ALU Carry-in : $C_n = 0$

ALU Source : R = RAM A, S = RAM B

ALU Function : $F = R + S + C_n$

ALU Destination : $F \longrightarrow Y$

$Y \longrightarrow$ RAM B

Condition Code Select : $\overline{CC} = 1$

Next microinstruction : $\longrightarrow mI_{001}$

mI_{019}: This microinstruction implements the macroinstruction $(R_A) + Imm \longrightarrow R_A$ where 'Imm' is a 16 bit immediate data contained in the second word of the macroinstruction. The microcode for this microinstruction is same as mI_{00D} except that here the Carry-in of the ALU is forced LOW and the ALU performs the operation $F = R + S + C_n$

mI_{01B}: This microinstruction implements the macroinstruction $Imm \longrightarrow R_A$ where 'Imm' is a 16 bit immediate data contained in the second word of the macroinstruction. The microoperations performed are as follows:-

Program Control Unit : $(PC) + 1 \longrightarrow PC$

Memory Unit : Memory READ request

Operand Select Logic : $A = R_B$, $B = R_A$

Bit Manipulation Logic : DA = 0

ALU Source : R = DA, S = DB

ALU Function : F = R.OR.S

ALU Destination : $F \longrightarrow Y$

$Y \longrightarrow$ RAM B

Condition Code Select : $\overline{CC} = 1$

Next microinstruction : $\rightarrow mI_{001}$

mI_{01D}-mI_{01E}: These two microinstructions implement the macroinstruction $(R_B) \longrightarrow R_A$. The microoperations

performed are as follows:-

mI_{01D}: Operand Select Logic : A = R_B, B = R_A

ALU Source : R = RAM A, S = RAM B

ALU Function : F = 0

ALU Destination : F $\longrightarrow$ Y
Y $\longrightarrow$ RAM B

Next microinstruction : $\longrightarrow$ mI_{01E}

mI_{01E}: Operand Select Logic : A = R_B, B = R_A

Bit Manipulation Logic : DA = 0

ALU Source : R = RAM A, S = RAM B

ALU Function : F = R.OR.S

ALU Destination : F $\longrightarrow$ Y

Condition Code Select : $\overline{CC}$ = 1

Next microinstruction : $\longrightarrow$ mI_{001}

mI_{024}: This microinstruction loads the status register from the 4 highest bits of the register addressed by the R_A field of the operand. The microoperations performed are as follows:-

Operand Select Logic : A = R_A, B = R_A

Bit Manipulation Logic : DA = 0

Status Register Control : ST_1 = 0, ST_2 = 1

ALU Source : R = DA , S = RAM B

ALU Function : F = R.OR.S

ALU Destination : F→Y

Y→RAM B

Y→Data Bus

Data Bus Status Register

Condition Code Select Logic : $\overline{CC}$ = 1

Next microinstruction : ⟶ mI_{001}

mI_{025} - mI_{026}: These two microinstructions send the contents of the register addressed by the R_A field of the operand to the output port addressed by the higher byte of the second word of the macroinstruction. The microoperations performed are as follows:-

mI_{025} : Program Control Unit : (PC)+1→PC

Memory Unit : Memory READ request

Next microinstruction : ⟶ mI_{0026}

mI_{026} : Operand Select Logic : $A=R_A$, $B=R_A$

Bit Manipulation Logic : DA = 0

I/O Control : IOR = 0, IOM = 0

ALU Source : R = DA, S = RAM B

ALU Function : F = R.OR.S

ALU Destination : F ⟶ Y

Y ⟶ RAM B

Y ⟶ Data Bus

Data Bus ⟶ Output port

Condition Code Select : $\overline{CC} = 1$

Next microinstruction : $\longrightarrow mI_{001}$

mI_{027} - mI_{028} : These two microinstructions load the register addressed by the R_A field of the operand from input port addressed by the higher byte of the second word of the macroinstruction. The microoperations performed are as follows:-

mI_{027} : Program Control Unit : $(PC)+1 \longrightarrow PC$

Memory Unit : Memory READ request

Next microinstruction : $\longrightarrow mI_{028}$

mI_{028} : Operand Select Logic : $A = R_A$, $B = R_B$

Bit Manipulation Logic : DA = 0

I/O Control : IOR = 0, IOM = 1

ALU Source : R = DA, S = DB

ALU Function : F = R.OR.S

ALU Destination : $F \longrightarrow Y$

$Y \longrightarrow$ RAM B

Condition Code Select : $\overline{CC} = 1$

Next microinstruction : $\longrightarrow mI_{001}$

mI_{029} - mI_{028} : These three microinstruction ADD the register addressed by the R_A field of the operand to the data contained in the memory location

addressed by register 15 (this register is used as a Data Counter). The result is stored in the register addressed by the R_A field of the operand. The contents of register 15 is incremented by one at the end of these microinstructions.

mI_{029} : Operand Select Logic : A=X, B=Register 15

ALU RAM output Control : EB = 0

Program Control Unit : PE = 0

Next microinstruction : $\longrightarrow mI_{02A}$

mI_{02A} : Operand Select Logic : $A=R_A$, $B=R_A$

ALU Carry-in : $C_n = 0$

Program Control Unit : (Data Bus) $\longrightarrow$ Address Bus

Memory Unit : Memory READ request

ALU Source : R = RAM A, S = DB

ALU Function : $F = R + S + C_n$

ALU Destination : $F \longrightarrow Y$

$Y \longrightarrow$ RAM B

Next microinstruction : $\longrightarrow mI_{02B}$

mI_{02B} : Operand Select Logic : A = X, B = Register 15

ALU Carry-in : $C_n = 1$

Bit Manipulation Logic : DA = 0

ALU Source : R = DA, S = RAM B

ALU Function : $F = S + C_n$

ALU Destination : $F \longrightarrow Y$

$Y \longrightarrow$ RAM B

Condition Code Select : $\overline{CC} = 1$

Next microinstruction : $\longrightarrow mI_{001}$

mI_{035} - mI_{037}: These three microinstructions implement the unconditional JUMP macroinstruction. The JUMP address (16 bits) is contained in the second word of the macroinstruction. The microoperations performed are as follows:-

mI_{035} : Program Control Unit : $(PC) + 1 \rightarrow PC$, $PE = 0$

Memory Unit : Memory READ request

Next microinstruction : $\longrightarrow mI_{036}$

mI_{036} : Program Control Unit : (Data Bus) $\longrightarrow$ Address Bus

(Data Bus) + $1 \longrightarrow$ PC

Memory Unit : Memory READ request

Instruction Register : IRE = 0

Next microinstruction : $\longrightarrow mI_{037}$

mI_{037} : Microprogram Sequencer : JUMP MAP

mI_{038} - mI_{039}: These two microinstructions implement the JUMP ON NON-ZERO macroinstruction. The

JUMP address (16 bits) is contained in the second word of the macroinstruction. The microoperations performed are as follows:-

mI_{038} : CCS Logic : ZERO Status $\rightarrow \overline{CC}$

Next microinstruction : $\rightarrow mI_{035}$ if condition TRUE (ZERO status = LOW)

$\rightarrow mI_{039}$ if condition FALSE (ZERO STATUS = HIGH)

mI_{039} : Program Control Unit : (PC) + 1 $\rightarrow$ PC

Next microinstruction : $\rightarrow mI_{001}$

mI_{040} - mI_{042}: These three microinstructions implement the JUMP ON EQUAL TO macroinstruction. This is a three word macroinstruction. First word contains the Op-Codeand the Operand R_A. Second word contains a 16 bit data and the third word contains a 16 bit address. The contents of the register addressed by the R_A field of the operand is compared to the 16 bit data contained in the second word of the macroinstruction. If they are equal then a JUMP is performed to the address contained in the third word of the macroinstruction. If they are not equal then the next sequential macroinstruction is executed. The micro-operations performed are as follows:-

mI_{040} : Operand Select Logic : A = R_A, B = R_A

ALU Carry-in : $C_n = 1$

Program Control Unit : (PC)+ 1 $\longrightarrow$ PC

Memory Unit : Memory READ request

ALU Source : R = RAM A, S = D8

ALU Function : F = R-S-1 + C_n

ALU Destination : F $\longrightarrow$ Y

Next microinstruction : $\longrightarrow$ mI_{041}

mI_{041} : CCS Logic : $\overline{ZERO}$ status $\longrightarrow$ $\overline{CC}$

Next microinstruction : $\longrightarrow$ mI_{035} if condition TRUE ($\overline{ZERO}$ status = LOW)

$\longrightarrow$ mI_{042} if condition FALSE ($\overline{ZERO}$ status = HIGH

mI_{042} : Program Control Unit : (PC) + 1 $\longrightarrow$ PC

Next microinstruction : $\longrightarrow$ mI_{001}

mI_{049} - mI_{04B}: These three microinstructions implement the subroutine CALL macroinstruction. This is a two word macroinstruction. First word contains the Op-Code and the second word contains the starting address of the subroutine. The microoperations performed are as follows:-

mI_{049} : Program Control Unit : (PC) +1 → PC, PE = 0

Memory Unit : Memory READ request

Next microinstruction : → mI_{04A}

mI_{04A} : Program Control Unit : PC → Stack, (Data Bus) → Address Bus

Memory Unit : Memory READ request

Instruction Register : IRE = 0

Next microinstruction : → mI_{048}

mI_{048} : Microprogram Sequencer: JUMP MAP

mI_{04F} - mI_{050} : These two microinstructions load the Program Counter Register from the ALU register addressed by the R_A field of the operand. The micro-operations performed are as follows:-

mI_{04F} : Operand Select Logic : A = R_A, B = R_A

Bit Manipulation Logic : DA = 0

Program Control Unit : PE = 0

ALU Source : R = DA, S = RAM B

ALU Function : F = R.OR.S

ALU Destination : F → Y

Y → RAM B

Y → Data Bus

Next microinstruction : → mI_{050}

mI_{050} : Program Control Unit : (Data Bus) ⟶ Address Bus

(Data Bus) + 1 ⟶ PC

Memory Unit : Memory READ request

Instruction Register : IRE = 0

Next microinstruction : ⟶ mI_{037}

mI_{051}- mI_{053} : These three microinstructions implement the subroutine RETURN macroinstruction. The microoperations performed are as follows:-

mI_{051} : Program Control Unit : (Stack) ⟶ PC

Next microinstruction : ⟶ mI_{052}

mI_{052} : Program Control Unit : (PC) ⟶ Address Bus

Memory Unit : Memory READ request

Instruction Register : IRE = 0

Next microinstruction : ⟶ mI_{053}

mI_{053} : Microprogram Sequencer : JUMP MAP

mI_{054}- mI_{055} : These two microinstructions logically rotate right (by two bits) the contents of the register addressed by the R_A field of the operand. The microoperations performed are as follows:-

mI_{054} : Operand Select Logic : A = R_A, B = R_B

Bit Manipulation Logic : DA = 0

Data Shifting Logic : S_4 = 1, S_5= 0 (rotate)

ALU Source : R = DA, S = RAM B

ALU Function : F = R.OR.S

ALU Destination : F/2 $\longrightarrow$ Y

Y $\longrightarrow$ RAM B

Next microinstruction : $\longrightarrow$ mI_{055}

mI_{055} : Same as mI_{054} except that the next micro-instruction executed is mI_{001}.

mI_{055} : This microinstruction logically rotates right (by one bit) the contents of the register addressed by the R_A field of the operand.

mI_{056} : This microinstruction enables the interrupts to be recognised by the CCU. The micro-operations performed are as follows:-

Interrupt Control Unit : $A_3 = A_2 = A_1 = A_0 = 0$

Condition Code Select : $\overline{CC}$ = 1

Next microinstruction : $\longrightarrow$ mI_{001}

Microcode for the Interrupt Service Routine

The CCU can only service an interrupt request during the execution of the second microinstruction of the macro-instruction FETCH cycle (mI_{002}). When an interrupt request is recognized by the CCU, the address inputs of the Mapping Memory are forced LOW (see Chapter 11.4.4). Address zero of the Mapping PROM was loaded with the hexadecimal data 03 which is the starting address of the interrupt service routine in the microprogram memory. Thus, when an interrupt request is recognized by the CCU, mI_{003} will be output from the Pipeline Register on the next clock LOW-to-HIGH transition. This microinstruction enables a 16 bit address to the Data Bus corresponding to the vector of the highest priority non-masked interrupt input (see Chapter 9.3.4). Also the Data Bus registers of the PCU are enabled (PE = LOW). On the next clock pulse, mI_{004} will be output from the Pipeline Register, the Data Bus registers of the PCU will be loaded and an address will be presented to the Address Bus. During this cycle, the interrupt facilities are disabled and PC is pushed onto the stack. Also a memory READ request is executed. By the end of this cycle, the address of the first macro-instruction of the main memory interrupt service routine will be available on the Data Bus. The Data Bus registers of the PCU are also enabled in this cycle.

On the next clock LOW-to-HIGH transition, mI_{005} will be output from the Pipeline Register, the Data Bus registers

of the PCU will be loaded and an address will be presented to the Address Bus. Also a memory READ request is executed and the Instruction Register is enabled (IRE = LOW). By the end of this cycle, the first macroinstruction of the main memory interrupt service routine will be available on the Data Bus.

On the next clock pulse, mI_{006} will be output from the Pipeline Register and the Instruction Register will be loaded from the Data Bus. The microprogram sequencer then fetches and presents the Op-Code to the Mapping Memory which in turn provides a starting address for the execution of the first macroinstruction of the interrupt service routine.

Microcode for Returning from the Interrupt Service Routine

The microinstruction mI_{05C} implements the RETURN FROM INTERRUPT macroinstruction. When executed, control is transferred to the macroinstruction which was fetched just before the recognition of the interrupt by the CCU. The micro-operations performed are as follows:-

mI_{05C} : Program Control Unit : (Stack) $\longrightarrow$ PC

Next microinstruction : $\longrightarrow$ mI_{052}

12.4 The Mapping PROM Data

The Mapping PROM was loaded with the starting addresses of the macroinstruction service routines in the microprogram memory. The Mapping PROM addresses correspond to the machine-codes (Op-Codes)of the macroinstructions. The Op-Codes of the macroinstructions implemented are given in Fig. 12.1 to 12.6.

12.5 Mnemonics for the Macroinstructions

In order to simplify programming the microcomputer, each macroinstruction was assigned a mnemonic name. The mnemonics were chosen as single words, made up from the first letters of the description of the macroinstructions. Table 12.1 gives a list of the macroinstructions implemented together with their functions and the mnemonics.

12.6 Conclusions

Chapter 12 has described some of the microinstructions implemented in the microprogram PROM of the microcomputer. By using 96 microinstructions, 62 general purpose macroinstructions were derived. The microprogramming process, in general, was not difficult. But some of the microinstructions were awkward to implement (e.g. the microinstructions for the interrupt service routine) .

Instruction	OPCODE	Microprogram Address	No of words	Condition Codes N ovr C Z	Clock Periods	Mnemonic	Operand
-	00	003	-	-	-	-	-
R_A.AND.$R_B \rightarrow R_A$	01	007	1	X 0 0 X	3	ANDR	R_A R_B
R_A.OR.$R_B \rightarrow R_A$	02	008	1	X 0 0 X	3	ORR	R_A R_B
R_A.EXOR.$R_B \rightarrow R_A$	03	009	1	X 0 0 X	3	EXR	R_A R_B
COMP. $R_A \rightarrow R_A$	04	00A	1	X X X X	3	COMR	R_A X
COMP. Q $\rightarrow$ Q	05	00B	1	X X X X	3	COMQ	X X
$\bar{R}_A + 1 \rightarrow R_A$	06	00C	1	X X X X	3	TCOM	R_A X
R_A.AND.Imm $\rightarrow R_A$	07	00D	2	X 0 0 X	3	ANDI	R_A X
R_A.OR.Imm $\rightarrow R_A$	08	00E	2	X 0 0 X	3	ORI	R_A X
R_A.EXOR.Imm $\rightarrow R_A$	09	00F	2	X 0 0 X	3	EXI	R_A X
$R_A + R_B \rightarrow R_A$	0A	010	1	X X X X	3	ADDR	R_A R_B
$Q + R_A \rightarrow R_B$	0B	011	1	X X X X	3	ADDQ	R_B R_A
$R_A - R_B \rightarrow R_A$	0C	012	1	X X X X	3	SUBR	R_A R_B
$R_A + 1 \rightarrow R_A$	0D	013	1	X X X X	3	INCR	R_A X
Q + 1 $\rightarrow$ Q	0E	014	1	X X X X	3	INCQ	X X
$R_A - 1 \rightarrow R_A$	0F	015	1	X X X X	3	DECR	R_A X
Q - 1 $\rightarrow$ Q	10	016	1	X X X X	3	DECQ	X X

Table 12.1 list of the fixed instructions implemented

cont..

Instruction	OPCODE	Microprogram Address	No of words	Condition Codes N ovr C Z	Clock Periods	Mnemonic	Operand
$0 \rightarrow Q$	11	017	1	X 0 0 X	3	CLRQ	X X
$0 \rightarrow R_A$	12	018	1	X 0 0 X	3	CLRR	R_A X
R_A + Imm $\rightarrow R_A$	13	019	2	X X X X	3	ADDI	R_A X
R_A - Imm $\rightarrow R_A$	14	01A	2	X X X X	3	SUBI	R_A X
Imm $\rightarrow R_A$	15	01B	2	No change	3	LIR	R_A X
Imm $\rightarrow$ Q	16	01C	2	No change	3	LIQ	X X
$R_B \rightarrow R_A$	17	01D	1	No change	4	MOVR	R_A R_B
$Q \rightarrow R_A$	18	01F	1	No change	3	MOVQ	R_A X
$R_A \rightarrow Q$	19	020	1	No change	3	LRQ	R_A X
Push $R_B \rightarrow R_A$	1A	021	1	No change	3	PUSH	R_A R_B
POP $R_A \rightarrow R_B$	1B	022		X 0 0 X	3	POP	R_B R_A
Status $\rightarrow R_A$	1C	023	1	No change	3	MOVS	R_A X
$R_A \rightarrow$ Status	1D	024	1	X X X X	3	LRS	R_A X
$R_A \rightarrow$ OUTPUT PORT	1E	025	2	No change	4	OUTS	R_A X
INPUT PORT $\rightarrow R_A$	1F	027	2	X 0 0 X	4	INS	R_A X
R_A + Mem. $\rightarrow R_A$	20	029	1	X X X X	5	ADDM	R_A X
Mem. $\rightarrow R_A$	21	02C	1	X 0 0 X	5	MOVM	R_A X

Table 12.1 cont.

cont...

Instruction	OPCODE	Microprogram Address	No of words	Condition Codes N ovr C Z	Clock Periods	Mnemonic	Operand
Wait	22	02F	1	No change	-	WAIT	X X
R_A - Imm	23	030	2	X X X X	3	CI	R_A X
R_A - R_B	24	031	1	X X X X	3	CR	R_A R_B
R_A - Mem.	25	032	1	X X X X	5	CM	R_A X
Jump	26	035	2	No change	3	JUMP	X X
Jump on non zero	27	038	2	No change	4	JNZ	X X
Jump on zero	28	03A	2	No change	4	JZ	X X
Jump on positive	29	03C	2	No change	4	JP	X X
Jump on negative	2A	03E	2	No change	4	JN	X X
Jump on equal to	2B	040	3	X X X X [1]	5	JE	R_A X
Decreament and skip	2C	043	2	X X X X [2]	5	DSZ	R_A X
Jump on less than	2D	046	3	X X X X [1]	5	JL	R_A X
Jump on non-carry	2E	047	2	No change	4	JNC	X X
Call to Subroutine	2F	049	2	No change	3	CALL	X X
$R_A \rightarrow$ Mem.	30	04C	1	No change	5	LRM	X X
$R_A \rightarrow$ PC	31	04F	1	No change	3	LRP	X X
Return	32	051	1	No change	3	RET	X X

Table 12.1 cont.

cont...

Instruction	OPCODE	Microprogram Address	No of Words	Condition Codes N ovr C Z	Clock Periods	Mnemonic	Operand
Rotate right 2	33	054	1	X 0 0 X	4	ROTR	R_A X
Enable Interrupt	34	056	1	No change	3	EI	X X
Disable Interrupt	35	057	1	No change	3	DI	X X
Set Mask	36	058	2	No change	3	SM	X X
Load Counter	37	059	2	No change	3	LC	X X
Shift right 1," 1"	38	05A	1	X 0 0 X	3	SR1	R_A X
Shift left 1," 0"	39	05B	1	X 0 0 X	3	SL1	R_A X
Ret. from Interrupt	3A	05C	1	No change	3	RETI	X X
No operation	3B	05E	1	No change	3	NOP	X X
Shift right 1," 0"	3C	05F	1	X 0 0 X	3	SR0	R_A X
Rot right 1, R_A	3D	055	1	X 0 0 X	3	ROTR1	R_A X
Rot left 1, R_A	3E	05D	1	X 0 0 X	3	ROTL1	R_A X
Not used	3F	060	-	-	-		

Table 12.1 cont.

Notes:

(1) Status changed according to the result of R_A - Imm.

(2) Status changed according to the result of R_A - 1

(3) Auxiliary register (registers of the REL) addressed by the R_A field of the operand

(4) 'X' in Condition Codes denotes that the corresponding status flag is set or reset according to the operation performed

(5) 'Imm' = 16 Bit Immediate data (6) 'Mem' = Contents of memory location addressed by register 15 (Data Counter) of the ALU.

The typical macroinstructions(e.g. register to register operations) are implemented in single locations of the microprogram memory and they are executed in three clock cycles: During the first cycle the ALU operation is performed. The remaining two cycles are used to fetch and decode(by executing the microinstructions 001 and 002) the next macroinstruction. It is also possible to execute the typical macroinstructions in two clock cycles by using two microprogram locations per macroinstruction: During the first cycle the ALU operation can be performed and at the same time the next macroinstruction can be fetched (overlapped fetching) from the main memory. The macroinstruction can then be decoded in the second clock cycle. The reason of not using overlapped fetching was to minimize the cost of the microprogram memory.

CHAPTER 13

13. THE OPERATING SYSTEM (MONITOR)

13.1 Introduction

A monitor* is a software tool enabling users to interactively monitor and control the operations of a microprocessor system via a teletype or VDU. It is basically a simple operating system providing facilities for loading, modifying, and displaying memory locations, executing programs etc. Microprocessor manufacturers ususally supply evaluation boards for their products which incorporate a monitor (in ROM and PROM), some RAM memory and teletype (or VDU) interface on one board. These are invaluable software development tools. The user can develop his software in the RAM part of the main memory by the help of the available monitor. When the software is in its final form, it can be made permanent by loading it into a non-volatile memory (PROM or ROM).

It was the author's opinion that a small monitor had to be included in the microcomputer designed, for the following reasons:-

i. A monitor can be a very useful tool in testing the microcode developed in the microprogram RAM. The monitor commands can be utilized to load and execute a sequence of macroinstructions in the main memory, ~~including the macroinstruction(s)~~ developed.

* Note that "Monitor" has a different meaning in the context of concurrent-processing.

The expected result can then be printed on the user's terminal (or sent to an I/O port) to verify the microcode.

ii. A monitor is an invaluable design aid in developing software. Without such a facility the users have to write, test and modify their software in a non-volatile memory or use a simulator (which does not operate in real-time).

This chapter describes the basic featuresof a monitor that was developed specifically for the microprogrammable microcomputer.

13.2 The UMM-16 Operating System

The monitor of the microcomputer was named the UMM-16 Operating System. It was developed from the fixed macro-instructions of the computer and it occupies the first 512 words of the main memory (in PROM). The monitor accepts several commands from the user terminal. In addition to these, it contains many subroutines for the teletype/VDU input and output routines.

Fig. 13.1 shows the structure of the UMM-16 Operating System. The command mode is recognized by the character '$'. When the microcomputer is reset, the monitor prints the character '$' on the user terminal to indicate that it is

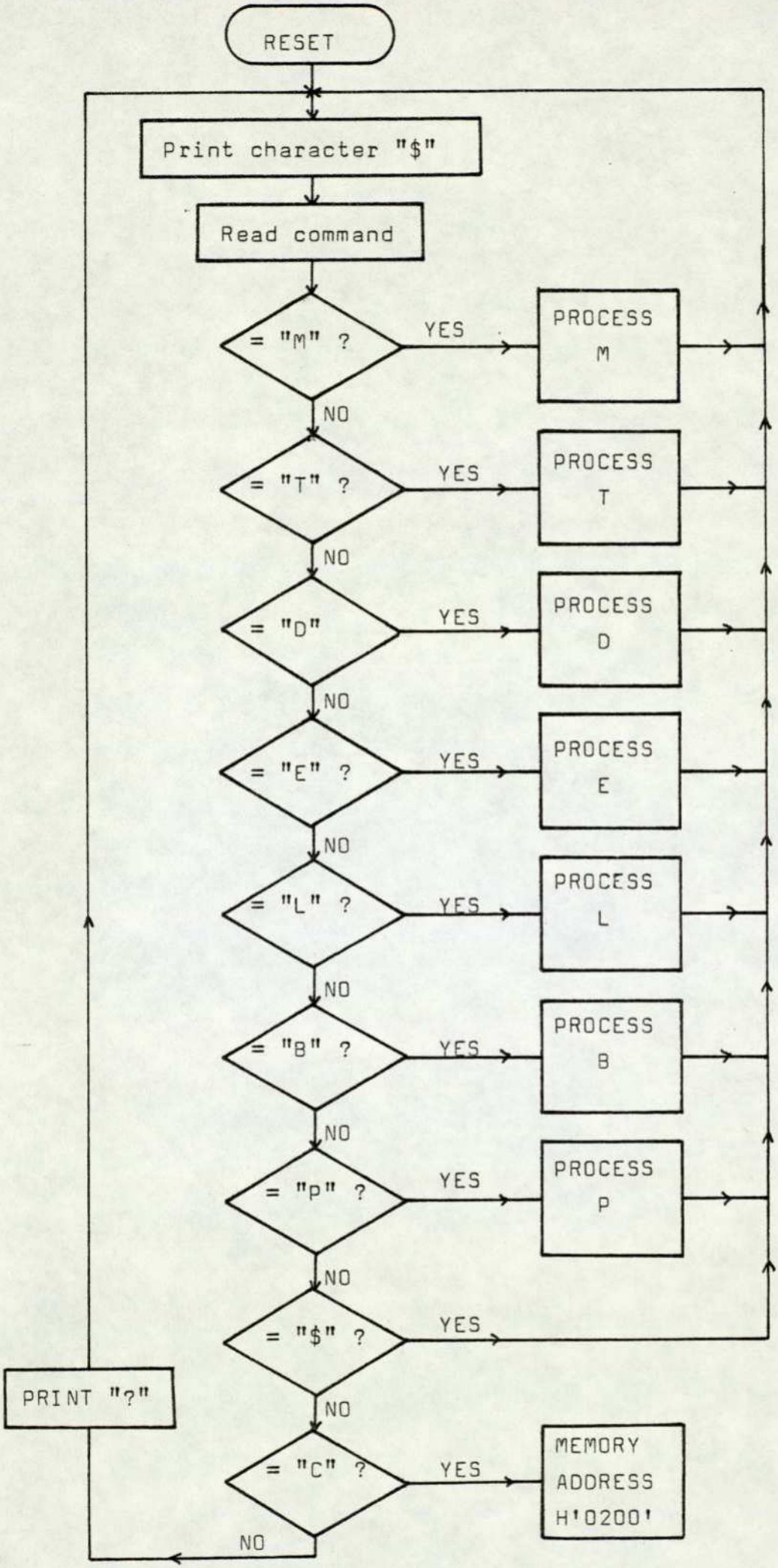


Figure 13.1 The structure of the UMM-16 Operating System .

waiting for a command to be typed in. The list of the valid commands is given below. Note that all addresses are in hexadecimal.

M xxxx — The contents of memory address xxxx is printed on the user terminal.

i. To examine the next word, type carriage return.

ii. To examine the previous word, type "<".

iii. To change the contents of address xxxx, type the new contents, followed by carriage return.

iv. To return to command mode, type "$".

T xxxx,yyyy — A block of memory from the start address xxxx to finish address yyyy is listed on the user terminal.

D xxxx,yyyy — Punches a block of memory data with start address xxxx and finish address yyyy in machine-code format (the paper tape punch mechanism of the teletype must be enabled)

E xxxx — Executes a program starting at memory address xxxx

L — Loads the memory with machine-code formatted paper tape (the tape reader must be enabled)

B xxxx,yyyy — Inserts a breakpoint at address yyyy and starts

executing from address xxxx

P Prints the contents of the ALU registers at the breakpoint.

$ Return to cammand mode

C This command is intended for future expansion of the Operating System. If the command 'C' is typed, a jump to address 0200 will be performed.

If the user types in a character which is not valid, the monitor will respond with a '?' and return back to the command mode.

The development of the UMM-16 Operating System was not an easy task. Although it occupies only 512 words of the main memory, there were no assemblers available (the fixed macro-instructions and their machine-codes were entirely chosen by the author) to simplify the programming. As a result, the Operating System had to be developed by programming in machine-code, which was a difficult process. Also, because the development of software usually requires many modifications until in the final form, it was necessary to develop the Operating System in the RAM part of the main memory. But this was not an easy task since extra hardware and a microprocessor development system was required to easily (and by low cost) load and modify the RAM part of the main memory. An F8 development system [17] was used for this purpose in developing the UMM-16

Operating System. When the software was in its final form, it was made permanent by loading it into a PROM.

The software listing of the UMM-16 Operating System is given in Appendix A.

CHAPTER 14

14. MINIMUM CYCLE TIME CALCULATIONS

14.1 Introduction

The minimum clock period that can be used in the microcomputer was determined by the component delays along the longest data and control paths. A visual inspection of the system architecture reveals that the longest delays occur in three major paths of the microcomputer: the control path, the data path, and the main memory path. This Chapter describes the typical delays (at 5V and 25°C operating temperature) expected in each of these paths, by referring to the propagation delays of the components used in the design. In order to minimize the cost and the power dissipation, medium speed (Standard TTL and Low-Power-Schottky TTL) devices were used in the design. It will be shown that by replacing the components in the critical speed paths with their high speed versions (Schottky TTL) , the minimum clock cycle can be improved by about a factor of 3 .

14.2 Speed Calculations in the Control Path

Figure 14.1 shows the block diagram of the CCU, simplified for the purposes of speed calculations. After considering all the possible propagation delay paths, it was found that the longest delay occurs when the second microinstruction of the macroinstruction FETCH cycle is

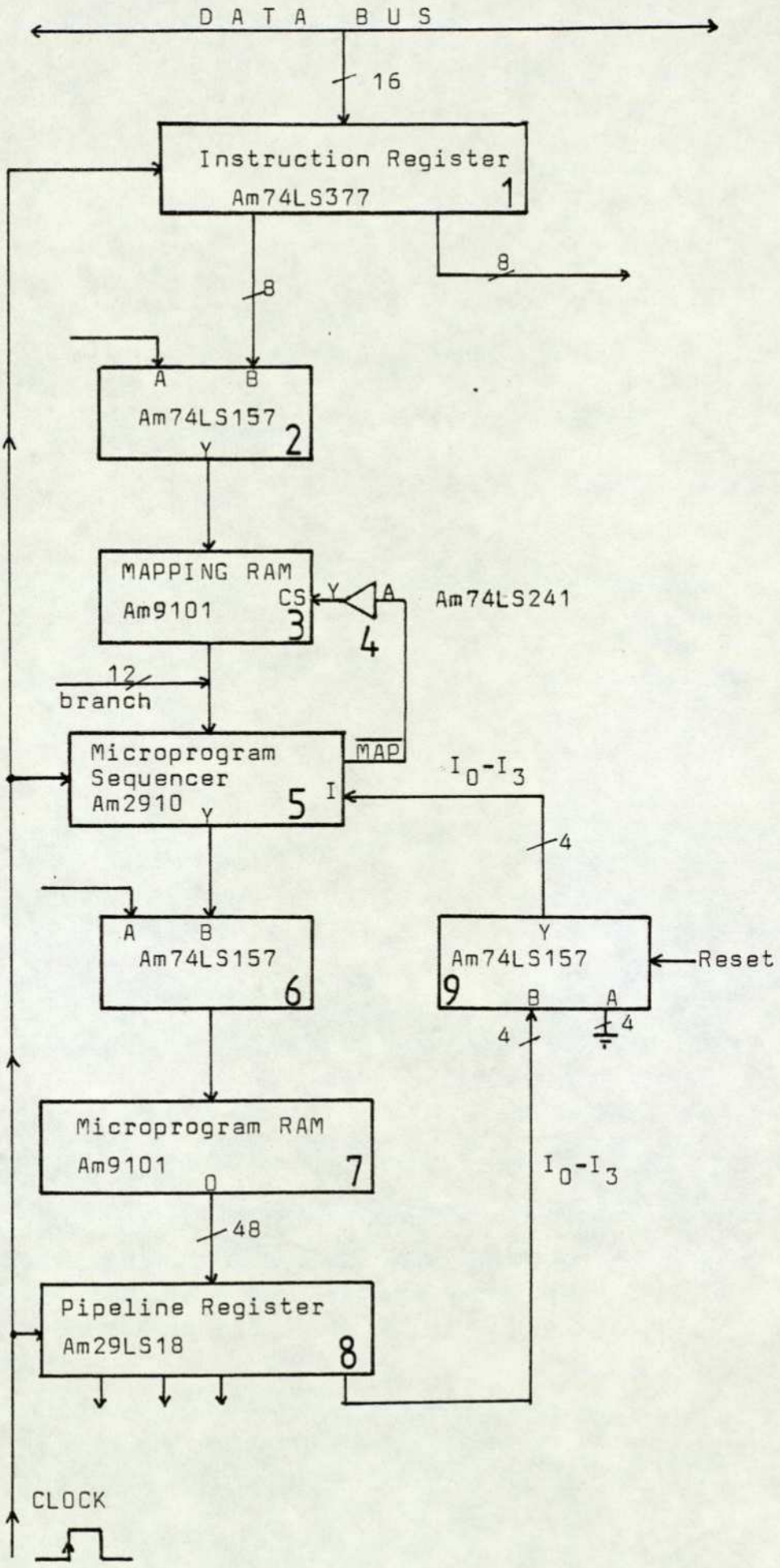


Figure 14.1 Block diagram of the CCU for calculating the max. propagation delays .

executed (i.e. when the microprogram sequencer executes the JUMP MAP microinstruction). The propagation delay for this path was calculated as follows (the brackets contain the data for faster versions of the devices used) :-

Device	Path	Delay(ns)
8. Am29LS18(Am2918)	D to Q	18(8.5)
9. Am74LS157(Am74S157)	B to Y	9(5)
5. Am2910	I to $\overline{\text{MAP}}$	27
4. Am74LS241(Am74S241)	A to Y	18(6)
3. Am91018(93L422)	CS to O	175(20)
5. Am2910	D to Y	14
6. Am74LS157(Am74S157)	B to Y	9(5)
7. Am91018(93L422)	Address to Output	400(45)
8. Am29LS18(Am2918)	Set-up time	15(5)
		685(125.5)

The maximum propagation delay in the control path was found to be 685 ns with the components used. However, by using faster devices as shown above in parentheses, it is possible to decrease the propagation delay to about 125 ns .

14.3 Speed Calculations in the Data Path

A block diagram of the APU (simplified for speed calculations) is shown in Figure 14.2 . It was found that the longest delay in the Data Path occurs when an arithmetic operation is performed together with shifting (via the condition multiplexer) in the same cycle. The propagation

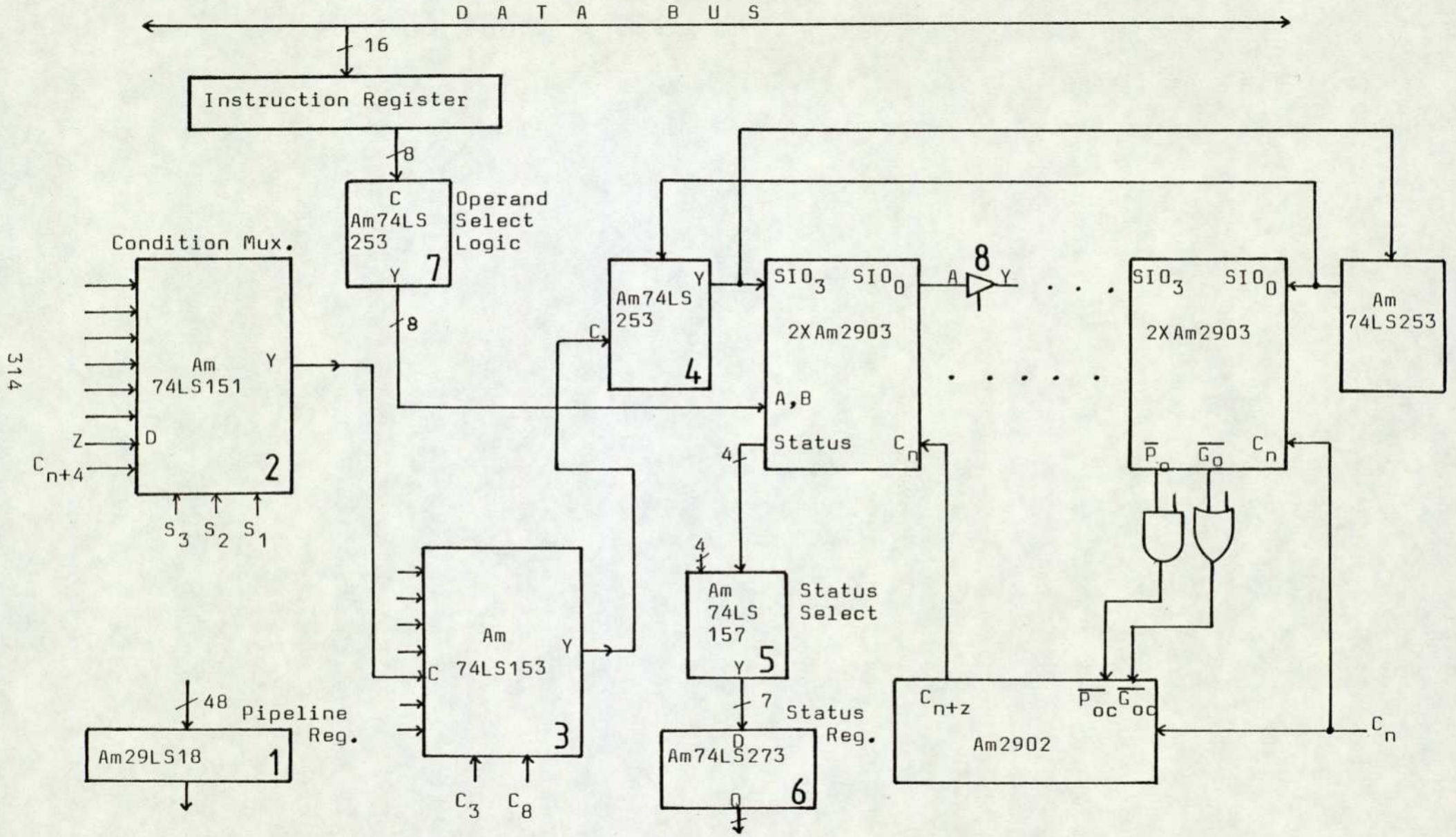


Figure 14.2 Block diagram of the APU for calculating the max. propagation delays .

delay for this path was calculated as follows:-

	Device	Path	Delay(ns)
1.	Am29LS18(Am2918)	D to Q	18(8.5)
7.	Am74LS253(Am74S253)	C to Y	21(6)
	Am2903	A,B to $\overline{G_0}$,$\overline{P_0}$	56
	SN7432(SN74S00)	$\overline{G_0}$,$\overline{P_0}$ to $\overline{G_{oc}}$,$\overline{P_{oc}}$	14(6)
	Am2902(Am2902A)	$\overline{G_{oc}}$,$\overline{P_{oc}}$ to C_{n+z}	8(5)
	Am2903	C_n to Y	25
	Am2903	Y to Z	16
2.	Am74LS151(Am74S151)	D to Y	20(12)
3.	Am74LS153(Am74S153)	C to Y	25(6)
4.	Am74LS253(Am74S253)	C to Y	21(6)
	Am2903	SIO_3 to Y	13
8.	Am74LS241(Am74S241)	A to Y	18(6)
	Am2903	Y to Z	16
5.	Am74LS157(Am74S157)	A to Y	9(5)
6.	Am74LS273	Set-up time	21
	Am2903	Y Set-up time	9
			310(216.5)

The maximum propagation delay in the data path was found to be 310 ns with the components used. By using faster devices as shown above in parentheses, it is possible to decrease the propagation delay to about 216 ns.

14.4 Speed Calculations in the Main Memory Path

The propagation delays in the main memory path are calculated from the time the PCU is enabled (via the Pipeline Register) until a valid memory data is available. Figure 14.3 shows a block diagram of the PCU and the main memory,simplified for the purposes of speed calculations. The longest delay in

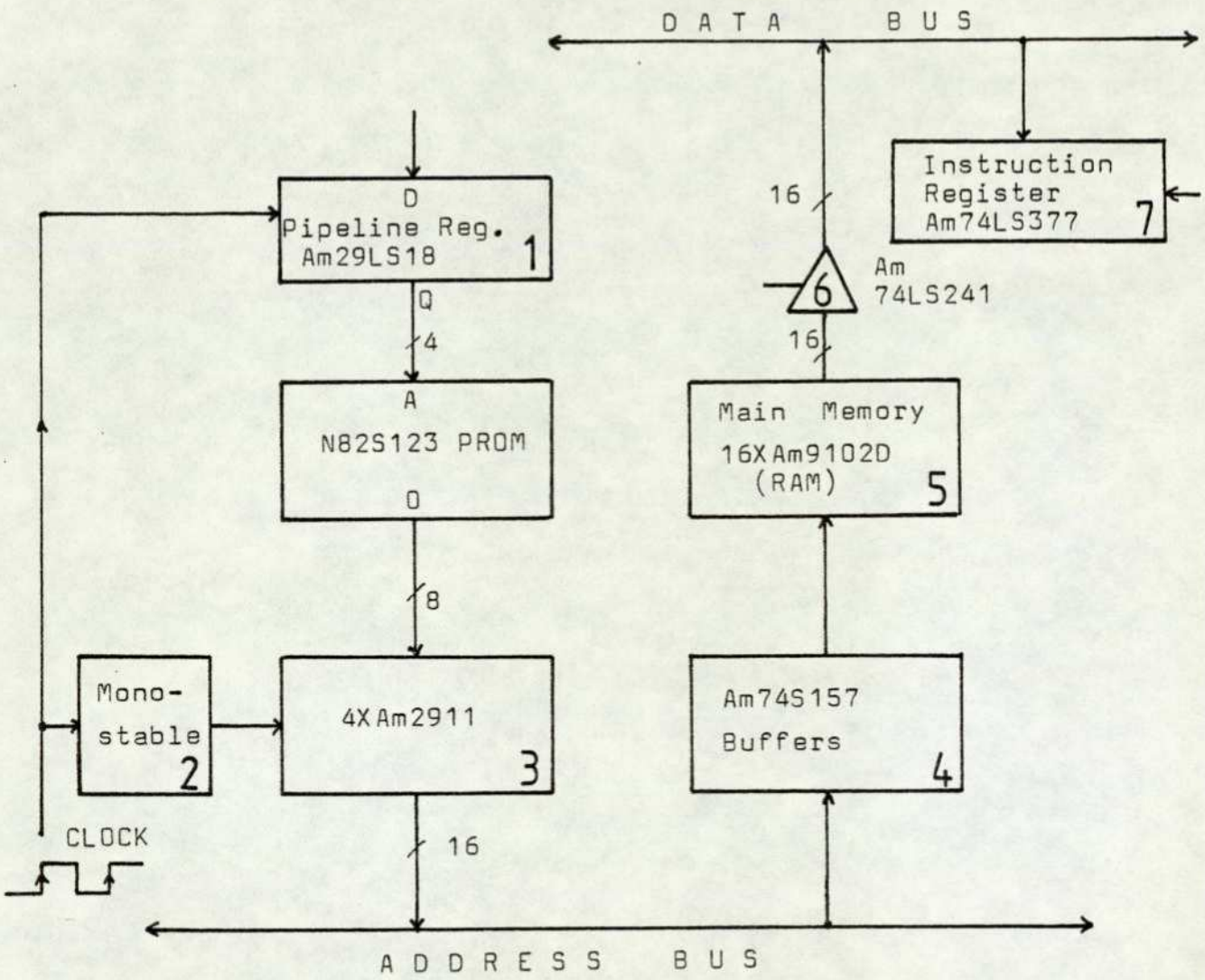


Figure 14.3 Block diagram for the speed calculations in the main memory path .

the main memory path occurs when the PC is incremented by one to present the next sequential address to the Address Bus and when a main memory access is required in the same cycle (e.g. during the execution of the microinstruction mI_{001}, as described in Chapter 12). The propagation delay for this path was calculated as follows (the brackets contain the data for faster versions of the devices used):-

Device	Path	Delay(ns)
1. Am29LS18(Am2918)	D to Q	18(8.5)
2. Monostable	Pulse width	120
3. Am2911	CP to Y via PC	29
4. Am74S157	Input to Y	5
5. Am9102D(93425)*	Address to Output	250(25)
6. Am74LS241(Am74S241)	A to Y	18(6)
7. Am74LS377	Set-up time	10
		450(203.5)

14.5 Conclusions

Minimum cycle time calculations indicate that the longest delays occur in the paths where MOS memories were used. As such, the clock period of the microcomputer should not theoretically be less than 685 ns (1.6 MHz). However, due to wide spread of timing parameters (with temperature and power supply variations), it was decided to choose the clock frequency to be 1 MHz, at which the microcomputer was operating satisfactorily. The clock frequency can in theory be increased to

* Not pin compatible

just below 5 MHz (200 ns) by replacing the critical components with their faster versions (at such high speeds special care should also be given to the component layout and wiring).

CHAPTER 15

15. A NOVEL TECHNIQUE FOR USER MICROPROGRAMMING (THE SUPPORT PROCESSOR)

15.1 Introduction

A special feature of a bit-slice processor is that it can allow changes in the instruction set, to meet new demands. The designers of bit-slice processors are concerned equally as much with the chips and their connections (hardware) as they are with the patterns of control bits in the micro-program memory. Conventionally a microprogram memory is implemented [18] with a read-only control store (PROM or ROM) and microprogramming support tools are developed for debugging, testing and maintenance of microprograms. A support system today generally includes a micro-assembler, a simulator, and a loader:-

Microassemblers

The purpose of the microassembler is to assist the user in writing more understandable code and to speed up the design process. A typical microassembler performs two distinct operations. The first, or definition phase, reads a specification of the microinstruction formats, default values of the several fields and the mnemonic names associated with the appropriate bit patterns of a given field. The second, or assembly, phase is like the assembly language processing of a fixed-architecture computer. Examples of typical microassemblers are AMDASM [19,20], DAPL [19] and

MICRO-AID [19].

Simulators

The next step in generating a meaningful microcode is to test the assembled microcode on a simulator before loading it into the non-volatile control store. Although simulators are invaluable microprogramming aids, there are no commercially available simulators for this purpose and the designers may be faced with the task of developing simulators for their own machines.

Loaders

The final step in implementing a microcode is to load it into the non-volatile control store. The manufacturers usually provide facilities for loading the PROM memories (some microassemblers incorporate PROM programming utilities)

Although a microprogram support system, as described above, is an invaluable design aid in generating a microcode, it has some major drawbacks: The commercially available microassemblers are very expensive and they are not generally available through commercial timesharing services. Also the development of a simulator may be a very time consuming, and hence an expensive process.

As discussed in Chapter 11, it was the author's own idea to design the microprogram memory from a combination

of PROM and RAM type of memories. By this technique, the user can develop his own microcode in the microprogram RAM and test on the actual system, hence eliminating the need for a simulator. When the microcode is satisfactory it can then be made non-volatile if desired. A Support Processor (An F8 microcomputer) was designed which accepts the user defined microcode and loads the microprogram RAM and the Mapping RAM as required. This technique is a very powerful and low cost solution for developing microinstructions.

The following sections of this Chapter will describe the hardware details of the Support Processor designed.

15.2 The Hardware of the Support Processor

The Support Processor was designed by using the low cost F8 microcomputer [21]. The block diagram of the Support Processor is shown in Fig. 15.1. The basic system consists of the 3850 Central Processing Unit (CPU), the 3853 Static Memory Interface (SMI) and the 3871 Peripheral Input/Output (PIO) chips of the F8 microprocessor family, supported by 2K of RAM and 2K of EPROM. Additional circuitry was designed to enable the Support Processor to be connected to a 20mA loop or RS232C (or CCITT V.24) type terminals. The functions of the various elements of the Support Processor are summarized below.

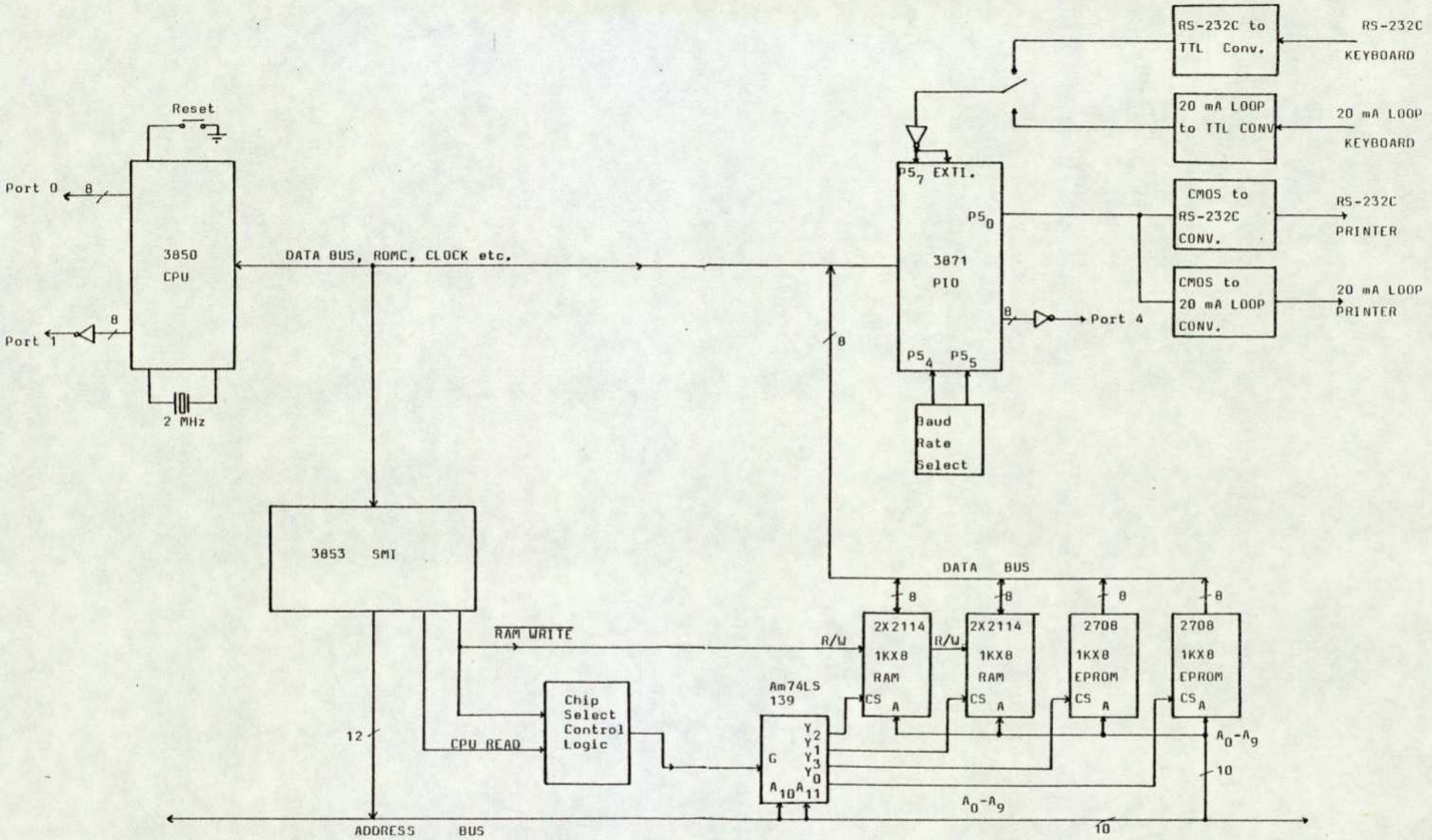


Figure 15.1 Block diagram of the Support Processor (Pn_m = Bit m of Port n)

15.2.1 The 3850 CPU [21]

The CPU of the F8 microprocessor has more than 70 instructions in its instruction set, and it operates on 8 bits of data. The device basically consists of a Control Unit (CU) and an Arithmetic and Logic Unit (ALU) with associated registers. The 3850 CPU has an 8-bit Accumulator register and a scratchpad consisting of 64 8-bit registers (see Fig. 15.2). The first 16 scratchpad registers can be referenced directly. The remaining scratchpad registers are addressed by using the 6-bit Indirect Sctatchpad Address Register (ISAR). Additionally the CPU contains a 5-bit Status register (the 'W' register) which stores the status conditions associated with the results of the CPU operations. The 3850 CPU communicates with the outside world via two 8-bit bidirectional input/output Ports (port 0 and Port 1). The five control lines (ROMC 0 - ROMC 4) transmit thirty-two different commands from the CPU to the rest of the devices in the system. At the beginning of each cycle, the CPU sends a command on these lines to indicate the operation to be performed during the current cycle by the rest of the system.

The clock of the Support Processor was derived by connecting a 2 MHz crystal to the CPU (2 MHz is the maximum clock frequency of the F8 microprocessor).

15.2.2 The 3853 SMI [21]

The SMI interfaces static memories to a 3850 CPU. One

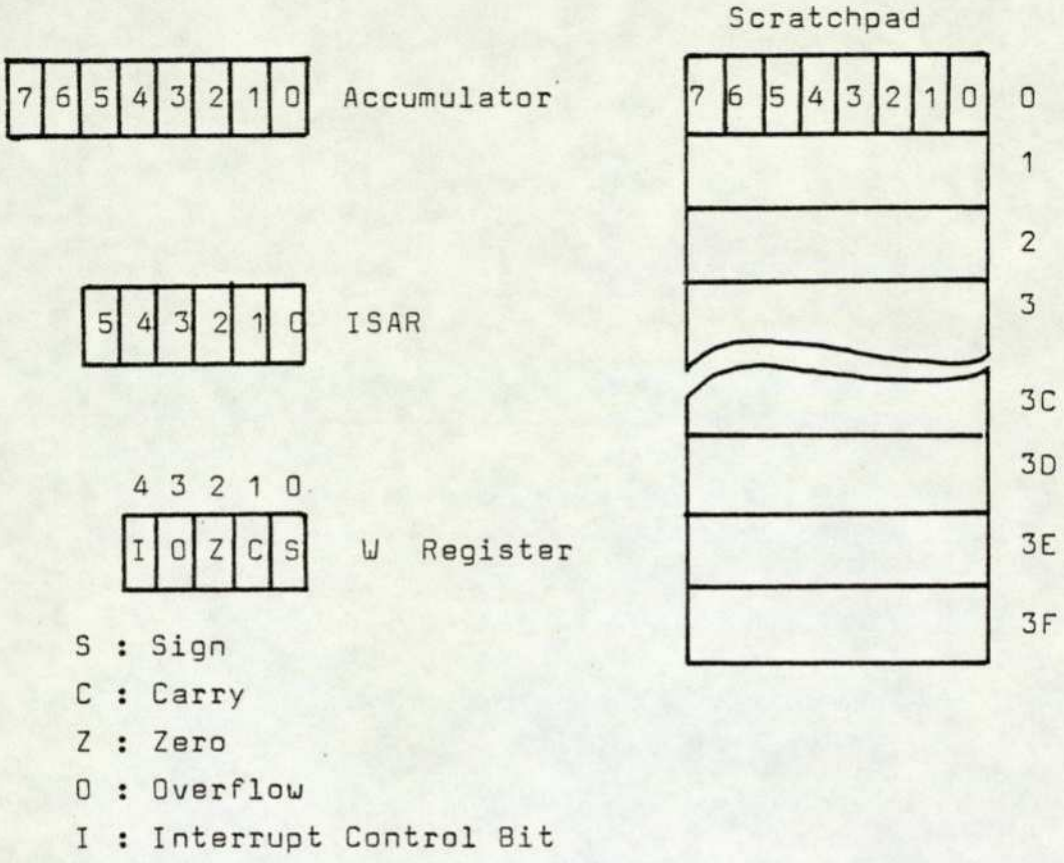


Figure 15.2 3850 CPU Programmable Registers .

SMI chip can interface up to 64K (65,536) bytes of static memory to the CPU. The device basically consists of four 16-bit registers : the program counter (PC0), the stack register (PC1), the primary Data Counter (DC0), and the secondary Data Counter (DC1). Additionally the SMI chip has four I/O ports which are reserved for special purpose. Ports H'0C' and H'0D' are used for the interrupt address vector upper and lower bytes, respectively. The interrupt control port address is H'0E' while the local timer port address is H'0F'

The two output signals CPU READ and the $\overline{\text{RAM WRITE}}$ of the SMI are important when RAM type memories are connected to the system. The CPU READ, when HIGH, specifies that data is to be read out of a RAM location. The $\overline{\text{RAM WRITE}}$, when LOW, specifies that data is to be written into a RAM location. The use of these two signals will be described later.

15.2.3 The 3871 PIO* [22]

The PIO provides four 8-bit ports, a programmable timer and external interrupt facilities for an F8 microprocessor system. Ports 4 and 5 are bidirectional I/O ports and they are used to interface the system to the external world. Port 6 and Port 7 are internal ports and they are used for the programmable timer.

The programmable timer of the PIO is an 8-bit binary

* As the PIO has four different versions, the one used in the design is described here (type 90071).

down counter. As shown in Figure 15.3, associated with the timer are an 8-bit register called the Interrupt Control Port (Port 6), a programmable prescaler and an 8-bit modulo-N register (Port 7). An output instruction (OUT or OUTS) to Port 7 loads the contents of the accumulator to both the timer and the 8-bit modulo-N register. Also an output to Port 6 selects the prescaler value and defines the mode of operation for the timer (the timer can operate in three different modes: the interval timer mode, the pulse width measurement mode, and the event counter mode).

The LSB and the MSB of Port 5 were used to interface the Support Processor to a teletype (or VDU). Bit 4 and bit 5 of Port 5 were used to select four different baud rates (1200,600, 300 and 110) for the Support Processor.

15.2.4 The RAM and the EPROM

The RAM of the Support Processor stores the user defined data which is to be loaded into the microprogram RAM and the Mapping RAM of the bit-slice microcomputer. The microprogram RAM was allocated (see Chapter 11) the address range H'100' to H'1FF'. Remembering that the width of the microprogram memory is 48 bits wide, each microinstruction occupies 6 locations in the RAM memory of the Support Processor. Thus, 1536 (=256X6) locations are required for a maximum of 256 microinstructions. Also, the Mapping RAM was allocated the address range H'80' to H'FF'. Remembering that the width of the Mapping RAM is 12

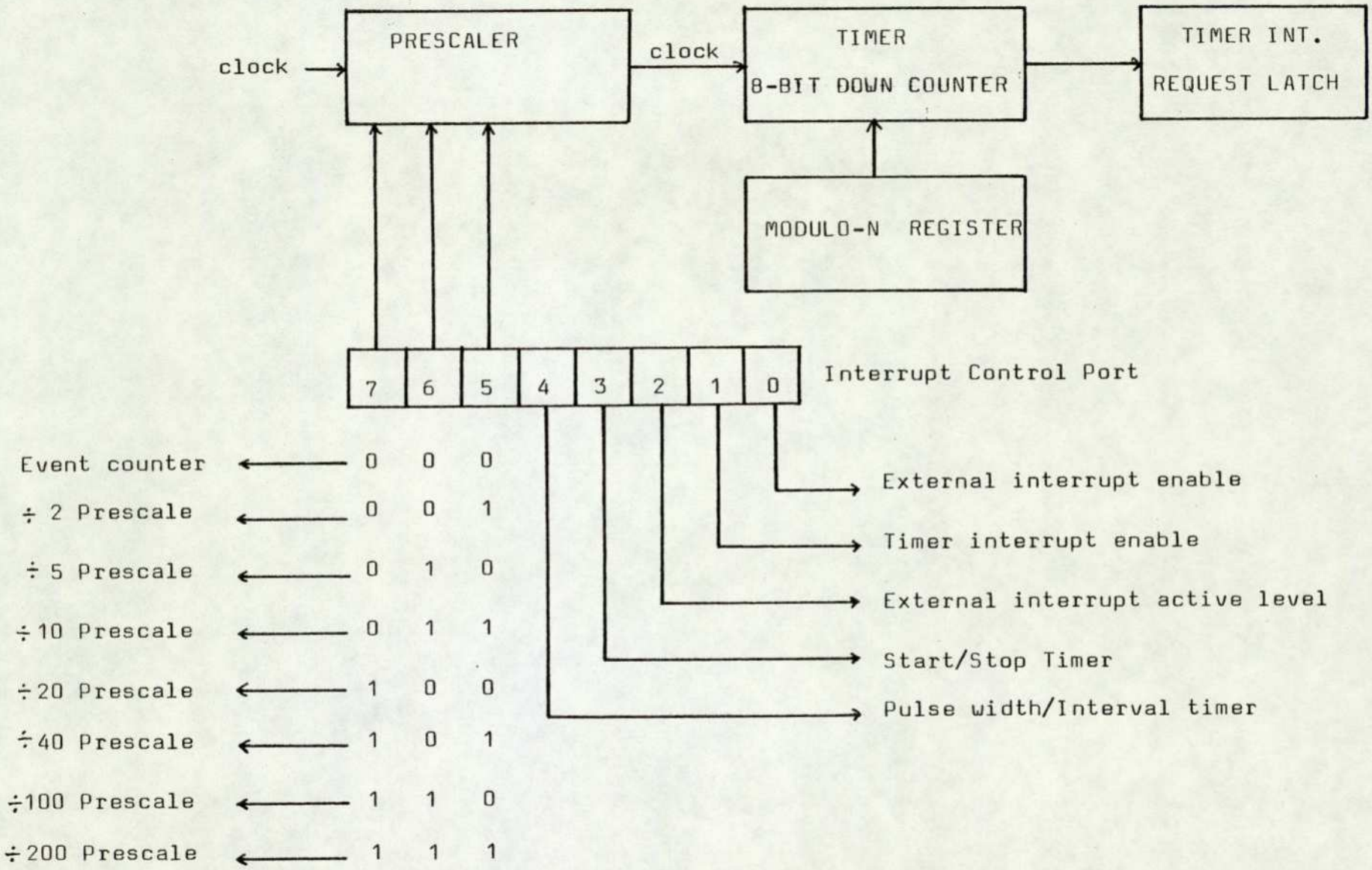


Figure 15.3 The registers of the 3871 PIO

bits wide, each Mapping RAM data occupies two locations in the RAM memory of the Support Processor. Thus, 256 (=128X2) RAM locations are required for a maximum of 128 Mapping RAM data. It can then be concluded that a minimum of 1792 (=1536 + 256) locations are required for the RAM memory of the Support Processor. The best choice was then to select 2K (2048) of RAM for the Support Processor.

The RAM chips used were SY2114 type high density (1KX4) memories. Four such devices were connected (see Fig. 15.1) for a 2KX8 RAM. The R/W inputs of the memories were derived from the $\overline{\text{RAM WRITE}}$ output of the 3853 SMI. The common input-output pins were directly connected to the Data Bus of the Support Processor.

The two EPROMs (2X2708) of the Support Processor store the system monitor (see Chapter 16), which occupies about 2K of locations. The outputs of the EPROMs were also directly connected to the Data Bus of the Support Processor.

15.2.5 Chip Select

The chip select inputs of the RAM and the EPROM were derived from a Am74LS139 type decoder (see Fig. 15.1) which has two inputs and four outputs. In addition to its normal inputs, this decoder also has a control input, 'G', such that when 'G' is LOW the decoder operates normally (only one output LOW), and when 'G' is HIGH, all four outputs of the decoder

are forced HIGH independent of the other two inputs. Since the memories are directly connected to the Data Bus, when there is no memory request then all of the memory chips must be disabled (CS = HIGH). If on the other hand a memory request is present then the correct memory chip must be enebled (CS = LOW). The following truth table shows how the 'G' input of the decoder was controlled to achieve a satisfactory operation:-

SMI OUTPUTS		DECODER CONTROL	
CPU READ (CRD)	$\overline{\text{RAM WRITE}}$ (R/W)	G	
1	X	0	read from memory
0	0	0	write to memory
0	1	1	no memory req.

Hence, $G = \overline{CRD}.R/W$

The memory map and the overall logic diagram of the Support Processor are given in Figure 15.4 and 15.5 respectively.

15.2.6 Interfacing the Support Processor to the Bit-Slice Microcomputer

The interconnections between the Support Processor and the Bit-Slice Microcomputer were straightforward (see Chapter 11). As described in Chapter 11, the microprogram RAM is divided into six fields, each field 8-bits wide and the Mapping RAM is divided into two fields, one 8-bits wide and the other

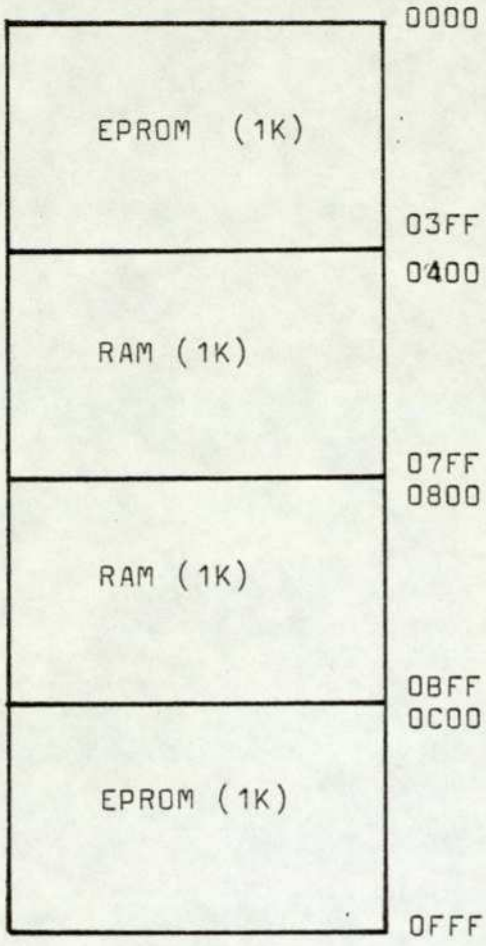


Figure 15.4 Memory map of the Support Processor .

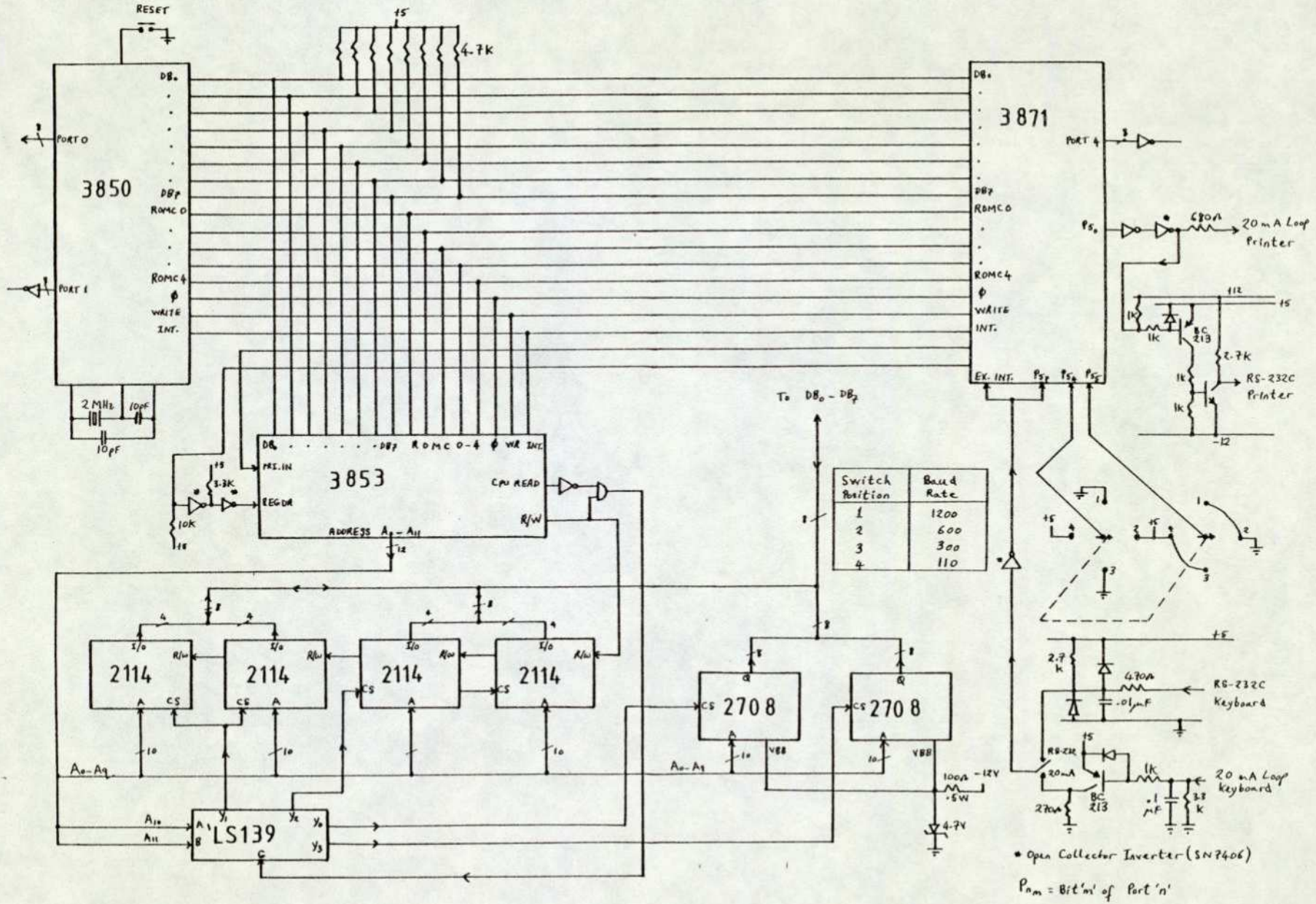


Figure 15.5 Logic diagram of the Support Processor

4-bits wide. During a microprogram load operation, Port 0 of the Support Processor provides addresses and Port 4 sends 8-bit data to the microprogram RAM. Port 1 selects the appropriate field and sends a signal to load that field with the 8-bit data. The Mapping RAM is loaded in a similar way to the microprogram RAM .

CHAPTER 16

16. THE MONITOR OF THE SUPPORT PROCESSOR

16.1 Introduction

As described in Chapter 15, the monitor of the Support Processor is stored in 2K of EPROM. The assembly listing of the Support Processor Monitor is given in Appendix C. This Chapter will describe the available monitor commands and examples will be given for their use.

The Support Processor monitor program[1] has the basic task as illustrated below:-

```
      print  'heading'
L1:   perform  print '*'
      read  'command'
            decode and implement command
      go to L1
```

Figure 16.1 gives a detailed description of how commands are decoded and implemented. The names of the monitor commands were chosen to resemble the operation to be performed. The command mode is recognized by the prompt character '*'. When the Support Processor is reset, the monitor prints a heading and waits for a user command to be typed in. A description of the valid commands is given in the following sections. Note that all addresses are in hexadecimal and the characters typed

1 The monitor of the Support Processor is named "Mini Meta-Assembler".

```
        print  '*'
LOOP    read   command
        if  (command = 'NMIC') goto L1
        if  (command = 'NMAP') goto L2
        if  (command = 'LMIC') load the microprogram RAM
        if  (command = 'LMAP') load the Mapping RAM
        if  (command = ' H  ') print list of available commands
        if  (command = ' DC ') goto L3
        if  (command = ' DP ') goto L4
        if  (command = ' T  ') goto L5
        if  (command = ' M  ') goto L6
        if  (command = ' E  ') goto L7
        if  (command = ' *  ') goto LOOP
        if  (command = ' LD ') load the Support Processor
        if  (wrong command   ) print '?'
        goto LOOP

L1      new microprogram data routine
        print  'ADDRESS:'
        read   command
        if  (command = '*') goto LOOP
        read   microprogram address and data
        goto L1

L2      new Mapping RAM data routine
        print  'ADDRESS:'
        read   command
        if  (command = '*') goto LOOP
        read   Mapping RAM address and data
        goto L2

L3      microprogram data dump routine
        read  terminating address 1XX
        dump  microprogram data from address H'100' to H'1XX'
        goto LOOP

L4      Mapping RAM data dump routine
        read  terminating address XX
        dump  Mapping RAM data from address H'80' to H'XX'
        goto LOOP

L5      print the Mapping RAM data
        read  Mapping RAM address XX
        print contents of address H'XX'
        goto LOOP

L6      print the microprogram RAM data
        read  microprogram address H'XXX'
        print contents of address H'XXX'
        goto LOOP

L7      change data routine
        read  data mn
        read  data pq
        change  mn to pq
        goto LOOP
```

Figure 16.1 Decoding and implementing the commands .

by the user (in the examples) are underlined for clarification. Also, all commands must be followed by a carriage-return unless specified otherwise .

16.2 The Command 'NMIC'

The NMIC (= New MICroprogram data) command is used to enter new microprogram data to the RAM memory of the Support Processor. When this command is typed the monitor prints " ADDRESS:" and waits for the address of the microprogram RAM to be entered. Valid addresses are H'100' to H'1FF'. If an invalid address is typed the monitor prints the message " INVALID ADDRESS" and jumps back to the command mode. After accepting an address, the monitor waits for the microprogram data to be typed in starting from field 5. The data in each field must be entered as two hexadecimal numbers. The monitor leaves space between the successive fields. When the data for the last field (field 0) is entered the monitor performs a carriage-return and is ready to accept the next microprogram address and the data. Although the valid addresses can be entered in any order (they need not be sequential), the microprogram data must be entered starting from field 5. When there are no more microinstructions to be entered the character '*' will take the monitor back to the command mode.

Example

As an example suppose that the three microinstructions

in Figure 16.2 are to be entered into the RAM memory of the Support Processor. The steps should then be as follows :-

```
* NMIC
ADDRESS: 100  FF EC 00 BD 01 1C
ADDRESS: 101  FF E8 01 FE 44 21
ADDRESS: 102  FE DC 00 00 AC 00
ADDRESS: *
*
```

16.3 The Command 'NMAP'

The NMAP (= New MAPping RAM data) command is used to enter new Mapping RAM data to the RAM memory of the Support Processor. When this command is typed the monitor prints " ADDRESS:" and waits for the address (Op-Code) of the Mapping RAM to be entered. Valid addresses are H'80' to H'FF'. If an invalid address is typed the monitor prints the message " INVALID ADDRESS" and jumps back to the command mode. Although the Mapping RAM is divided into two fields the user need only specify the data in a single field as three hexadecimal numbers (the software takes care of the field organization). Valid data range is H'100' to H'1FF'. When a Mapping RAM data is entered the monitor performs a carriage-return and is ready to accept the next Mapping RAM address and the data. Although the valid addresses can be entered in any order (they need not be sequential), the data must always start with a "1". When there are no more data to be entered the character '*' will

	OP CODE	ADDRESS	FIELD 5								FIELD 4								FIELD 3										FIELD 2								FIELD 1								FIELD 0							
			ICU			REL	PCU	CCS			Am 2910					Inst. Req.	DSL			IOM	MMD	P/P	CCS, Carry-in								PCU				Branch Address								ICU			ALU			Operand Select	
			A_3	MRQ	WAIT	FE	PCE	ST_1	ST_2	IOR	I_3	I_2	I_1	I_0	EDB	IRE	S_4	S_5	RS_3	C_8	C_4	C_5	C_6	S_3	S_2	S_1	C_3	CR_{23}	CR_{22}	CR_{21}	CR_{20}	PE	P_4	P_3	P_2	P_1	BR_{11}	BR_{10}	BR_9	BR_8	BR_7	BR_6	BR_5	BR_4	BR_3	BR_2	BR_1	BR_0	A_2	A_1	A_0	IEN	$\overline{OE}_Y$	EB	RS_1	RS_2
INSTRUCTION			47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Inst. 1	80	100	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	X	X	X	X	X	X	X	X	1	0	1	1	1	1	0	1	0	0	0	0	0	0	0	1	0	0	0	1	1	1	X	X
Inst. 2	81	101	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0	1	0	0	X	X	1	0	0	0	0	1
Inst. 3	82	102	1	1	1	1	1	1	1	0	1	1	0	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0	1	0	1	0	1	1	X	X	X	X	X	X	0	0	0	0	0

take the monitor back to the command mode.

Example

As an example suppose that the Op-Codes shown in Figure 16.2 are to be used to provide the microprogram starting addresses for the three instructions. The RAM memory of the Support Processor should then be loaded as follows :-

```
* NMAP
ADDRESS: 80  100
ADDRESS: 81  101
ADDRESS: 82  102
ADDRESS: *
*
```

16.4 The Command 'LMIC'

The LMIC (= Load MICroprogram RAM) command is used to load the microprogram RAM of the Bit-Slice microcomputer from the Support Processor. Before this command is used the microprogram data must have been loaded to the Support Processor with the 'NMIC' command. Also, the external switch S_3 (see Fig. 11.3) must be in the LOAD position. When the microprogram RAM is loaded the monitor prints the message " MICROPROGRAM MEMORY LOADED" and jumps back to the command mode.

Example

Suppose that the microprogram data has been loaded to the Support Processor with the 'NMIC' command and it is required to load the microprogram RAM of the Bit-Slice microcomputer. The steps should then be as follows :-

```
*                                  (S3 in OPERATE position)
* LMIC                             (S3 in LOAD position)
MICROPROGRAM MEMORY LOADED
*                                  (S3 in OPERATE position)
```

16.5 The Command 'LMAP'

The LMAP (= Load MAPping RAM) command is used to load the Mapping RAM of the Bit-Slice microcomputer from the Support Processor. Before this command is used the Mapping RAM data must have been loaded to the Support Processor with the 'NMAP' command. Also the external switch S_2 (see Fig. 11.3) must be in the LOAD position. When the Mapping RAM is loaded the monitor prints the message "MAPPING RAM LOADED" and jumps back to the command mode.

Example

Suppose that the Mapping RAM data has been loaded to the Support Processor with the 'NMAP' command and it is required to

load the Mapping RAM of the Bit-Slice microcomputer. The steps should then be as follows :-

```
*                                  ($S_2$ in OPERATE position)
* LMAP                             ($S_2$ in LOAD position)
MAPPING RAM LOADED
*                                  ($S_2$ in OPERATE position)
```

16.6 The Command 'M 1XX'

The 'M 1XX' command is used to print the contents of microprogram RAM address '1XX' on the user terminal in six fields, starting from field 5 and each field containing two hexadecimal numbers. Valid address range is H'100' to H'1FF'. If an invalid address is typed the monitor prints the message " INVALID ADDRESS" and jumps back to the command mode. It is important to note that the data printed is the data that has been loaded to the Support Processor with the 'NMIC' command and this data may not be same as the data contained in the microprogram RAM of the Bit-Slice microcomputer (unless the microprogram RAM has been loaded with the 'LMIC' command) .

Example

Suppose that the Support Processor has been loaded with the data as shown in the example in Section 16.2 . The 'M 1XX'

command can be used to display this data as follows :-

```
* M 100  FF EC 00 BD 01 1C
* M 101  FF E8 01 FE 44 21
* M 102  FE DC 00 00 AC 00
*
```

16.7 The Command 'T XX'

The 'T XX' command is used to print the contents of Mapping RAM address 'XX' on the user terminal as three hexadecimal numbers. Valid address range is H'80' to H'FF'. If an invalid address is typed the monitor prints the message " INVALID ADDRESS" and jumps back to the command mode. It is important to note that the data printed is the data that has been loaded to the Support Processor by the 'NMAP' command and this data may not be same as the data contained in the Mapping RAM of the Bit-Slice microcomputer (unless the Mapping RAM has been loaded recently with the 'LMAP' command) .

Example

Suppose that the Support Processor has been loaded with the data shown in the example in Section 16.3. The 'T XX' command can be used to display this data as follows :-

```
* I 80  100
* I 81  101
* I 82  102
*
```

16.8 The Command 'E/mn/pq/'

This command[1] is only available just after the 'M 1XX' command is executed. It changes the hexadecimal data 'mn' to 'pq' in the microprogram address H'1XX'. It is important to note that the data is changed only in the Support Processor and it is necessary to execute the command 'LMIC' to change the actual microprogram RAM data of the Bit-Slice microcomputer. If the data 'mn' can not be found in the microprogram address H'1XX', the monitor prints a question mark '?' and returns back to the command mode.

Example

Suppose that it is required to change the data H'BD' to H'2F' in the example given in Section 16.2 :-

```
* M 100  FF EC 00 BD 01 1C
* E/BD/2F/
* M 100  FF EC 00 2F 01 1C
* E/45/00/  ?
*
```

1 Carriage-return is not required at the end of this command .

a paper tape with the Mapping RAM data from address H'80' to H'XX' (excluding address H'XX') where XX must be greater than H'80'.

As an example suppose that it is required to dump out the Mapping RAM data given in the example in Section 16.3 :-

```
* DP 83
DX0100
X0101
X0102
*

*
```

The letter 'D' at the beginning of the data stream is used to identify that this is a Mapping RAM data. The letters 'X' identify the beginnings of each Mapping RAM address. The character '*' just after the data stream indicates the end of the dump routine. Note that the Mapping RAM data H'1XX' is stored by the Support Processor in two eight bit fields, as H'01XX' .

16.11 The Command 'LD'

The LD (=LoaD) command[1] is used to load the RAM memory of the Support Processor from a paper tape (the paper tape

1 This command is only available for 110 baud rate of operation .

format must be as described in Section 16.9 or 16.10). Both the Mapping RAM and the microprogram RAM data can be loaded by using this command. The steps should be as follows :-

```
* LD                    (enable the paper tape reader)
*
```

16.12 The Command 'H'

The H (= Help) command is used to provide the user with a list of the available monitor commands. When this command is typed the monitor prints the following message (note that the command 'E/mn/pq/'is not listed by this command) :-

```
* H
COMMANDS:
NMIC    NEW MICROPROGRAM
NMAP    NEW MAPPING RAM
H       ASK FOR HELP
*       DELETE COMMAND AND RESTART
LMIC    LOAD MICROPROGRAM
LMAP    LOAD MAPPING RAM
M 1XX   PRINT CONTENTS OF MICROPROGRAM ADDRESS 1XX
T XX    PRINT CONTENTS OF MAPPING RAM ADDRESS XX
LD      LOAD VIA PAPER TAPE
DC 1XX  DUMP OUT MICROPROGRAM DATA FROM 100 TO 1XX
DP XX   DUMP OUT MAPPING RAM DATA FROM 80 TO XX
*
```

16.13 Other Features of the Monitor

If the user types an invalid command the monitor prints a question mark '?' to tell the user that the command has not been recognized :-

```
* K ?
* P ?
* 7 ?
*
```

It is also possible to cancel a command (before typing a carriage-return) and jump back to the command mode by typing the character '*' :-

```
* LM*
* NMI*
* DM*
*
```

A typical run of the Support Processor (for the example given in Fig. 16.2) is shown in Figure 16.3 .

16.14 Conclusions

Chapters 15 and 16 have described a low cost novel technique for developing microinstructions. As a general purpose

```
THE CITY UNIVERSITY
F8 MINI META-ASSEMBLER V1

TYPE H FOR HELP
* H
COMMANDS:
NMIC    NEW MICROPROGRAM
NMAP    NEW MAPPING RAM
H       ASK FOR HELP
*       DELETE COMMAND AND RESTART
LMIC    LOAD MICROPROGRAM
LMAP    LOAD MAPPING RAM
M 1XX   PRINT CONTENTS OF MICROPROGRAM ADDRESS 1XX
T XX    PRINT CONTENTS OF MAPPING RAM ADDRESS XX
LD      LOAD VIA PAPER TAPE
DC 1XX  DUMP OUT MICROPROGRAM DATA FROM 100 TO 1XX
DP XX   DUMP OUT MAPPING RAM DATA FROM 80 TO XX
* K ?
* P ?
* NMIC
ADDRESS: 100  FF EC 00 BD 01 1C
ADDRESS: 101  FF E8 01 FE 44 21
ADDRESS: 102  FE DC 00 00 AC 00
ADDRESS: *
* NMAP
ADDRESS: 80  100
ADDRESS: 81  101
ADDRESS: 82  102
ADDRESS: *
* M 100  FF EC 00 BD 01 1C
* M 101  FF E8 01 FE 44 21
* M 102  FE DC 00 00 AC 00
* T 80  100
* T 81  101
* T 82  102
* T 2
INVALID ADDRESS
* LMIC
MICROPROGRAM MEMORY LOADED
* LMAP
MAPPING RAM LOADED
* M 100  FF EC 00 BD 01 1C
* E/BD/2F/
* M 100  FF EC 00 2F 01 1C
* E/45/00/ ?
* M 100  FF EC 00 2F 01 1C
* E/2F/BD/
* DC 103
SXFFEC00BD011C
XFFE801FE4421
XFEDC0000AC00
*

* DP 83
DX0100
X0101
X0102
*

* LM*
* M *
*
```

Figure 16.3 A typical run of the Support Processor .

microprogram design/test method the technique shown may be less attractive to the large, well established companies in the computer industry than typical methods which involve the use of mini or large host computers. There are, however, many small companies and research centres with limited resources in which the designers face the problems of developing microinstructions for their bit-slice processors. To designers working in such circumstances the technique described can offer a flexible design tool at an acceptable cost.

CHAPTER 17

17. THE THEORY OF DIGITAL FILTERS

17.1 Introduction

The aims of digital filtering are the same as those of continuous-time filtering, but here the actual filtering process is carried out by using digital hardware.

Until recently digital filter algorithms have been used primarily in computer simulations and sampled data analysis. As a result of the increasing speed and decreasing size and cost of digital components, the real-time digital filtering has been used during the last few years in different areas, such as communications, aerospace,radar-sonar, biomedicine, image processing etc. The advantages offered by the digital approach to signal processing can be summarized as follows :-

i. Greater flexibility is achieved since filter response can be changed easily by varying the proper arithmetic coefficients.

ii. Component tolerances associated with capacitors, inductors, and resistors do not exist.

iii. No impedance matching problems exist in the digital domain.

iv. High precision is achieved (limited only by the number of bits used).

v. Performance from unit to unit is stable and repeatable.

vi. Filter break frequencies can be placed without restriction.

vii. Performance limitations of physical analogue components are avoided.

viii. Periodic calibration as is required with analogue circuits is eliminated.

ix. Digital filters are especially economic in the very low frequency band (0.001 to 1 Hz) where the size of analogue components become appreciable.

Linear continuous-time filter theory is based on the mathematics of linear differential equations, while linear digital filter theory is based on the mathematics of linear difference equations with constant coefficients. The basic mathematical tool of digital filters is the z-transform calculus.

This Chapter presents a brief introduction to the theory of digital filters. The next Chapter describes how special instructions can be invented for the implementation of real-time

digital filters using the bit-slice microcomputer designed.

17.2 The Z-Transform

The z-transform is used to describe a sampled data system in much the same way as the Laplace transform is used to describe a continuous-time system.

It is convenient to think of the sampling process as an impulse modulation of a continuous-time input signal. If the input signal $x(t)$ is sampled every T seconds, the output, $x^*(t)$, can be represented as :-

$$x^*(t) = P(t)x(t) \qquad (17.1)$$

where $P(t)$ represents a unit impulse train (see Figure 17.1) defined by

$$P(t) = \sum_{n=0}^{\infty} \delta(t-nT) \qquad (17.2)$$

The output sequence can then be represented as

$$x^*(t) = x(t) \sum_{n=0}^{\infty} \delta(t-nT) \qquad (17.3)$$

Equation 17.3 can be rewritten to include the input signal as a time function when $t = nT$, as

$$x^*(t) = \sum_{n=0}^{\infty} x(nT)\ \delta(t-nT) \qquad (17.4)$$

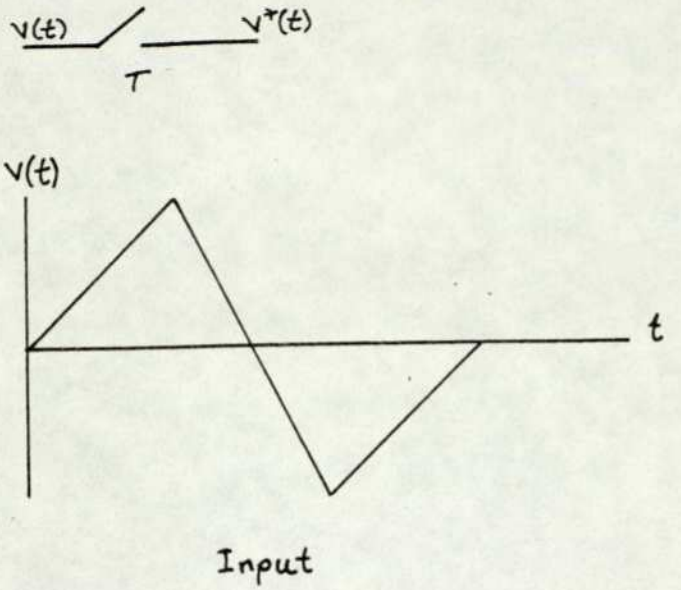


Figure 17.1 The sampling process

The Laplace transform of $\delta(t-nT)$ is

$$L\left[\delta(t-nT)\right] = e^{-nTs}$$

Therefore, the Laplace transform of the sampled waveform described by equation 17.4 is given by

$$X^*(s) = \sum_{n=0}^{\infty} x(nT)e^{-nTs} \qquad (17.5)$$

The z-transform is obtained by making the substitution

$$z = e^{sT}$$

or

$$s = \frac{1}{T} \ln(z)$$

This simple sustitution maps the periodic strip from $-w_s/2$ to $+w_s/2$ of the jw axis of the s-plane onto a unit circle in the z-plane where w_s is the sampling frequency ($w_s = \frac{2\pi}{T}$). The remainder of this strip in the left-hand s-plane is mapped inside the unit circle in the z-plane (see Fig. 17.2). Likewise the corresponding right-half strip is mapped as the exterior of the unit circle in the z-plane. A stable transfer function has its poles in the left-half of the s-plane: thus, the poles of the transformed function must lie within the unit circle in the z-plane. The interval from $-w_s/2$ to $+w_s/2$ is known as the Nyquist interval. This interval places a bound on the bandwidth of the input signal such that if the input signal is not bandlimited to below the radian frequency $w_s/2$, it cannot be recovered exactly at the output.

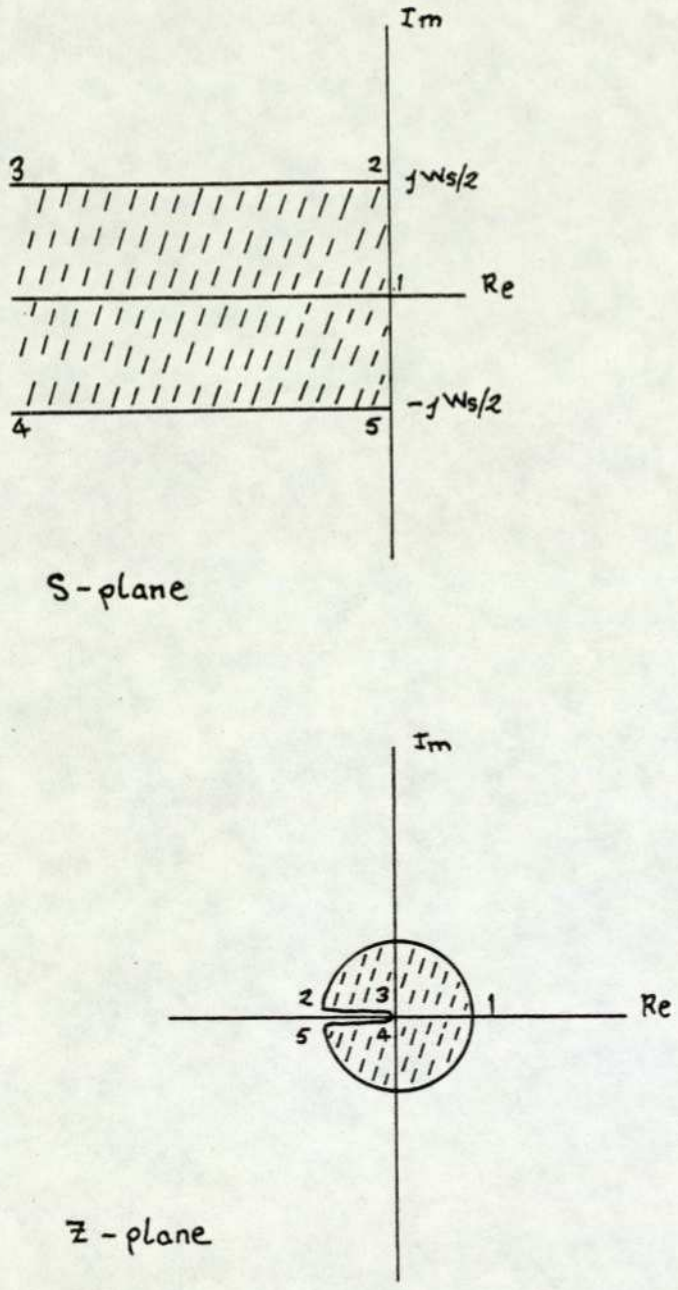


Figure 17.2 Relation between the s-plane and the z-plane .

Using the substitution $z = e^{sT}$ on equation 17.5 , the output z-transform $X(z)$ of a sampled input signal $x(t)$ is found as :

$$X(s)\Big|_{s = \frac{1}{T}\ln(z)} = X(z) = \sum_{n=0}^{\infty} x(nT)z^{-n} \qquad (17.6)$$

The z-transform gives a procedure by which the input sequence can be transferred into a function of the complex variable z . The variable z^{-n} can be interpreted as a type of operator [23] that upon multiplication shifts signal samples to the right (delays) by nT seconds :-

$$Z\left[x(nT-kT)\right] = z^{-k}X(z) \qquad (17.7)$$

17.3 The Difference Equation

The linear difference equations are used to define the sampled output pulse amplitude $y(t)$ as a function of the present input pulse and any number of past input and output pulses. A general form of the difference equation [24] is :-

$$y(nT) = \sum_{k=0}^{N} a_k x(nT-kT) + \sum_{k=1}^{M} b_k y(nT-kT) \qquad (17.8)$$

where $x(nT)$ represents the present input sample and the $x(kT)$ are past input samples. Likewise, $y(nT)$ is the present output sample and the $y(kT)$ are past output samples. The coefficients a_k and b_k determine the response of the filter.

The z-transform of the general difference equation 17.8 is derived by using equations 17.6 and 17.7 and is given as

$$y(z) = x(z) \sum_{k=0}^{N} a_k z^{-k} + y(z) \sum_{k=1}^{M} b_k z^{-k} \qquad (17.9)$$

The above equation can be written in the normal transfer function form as

$$H(z) = \frac{y(z)}{x(z)} = \frac{\sum_{k=0}^{N} a_k z^{-k}}{1 - \sum_{k=1}^{M} b_k z^{-k}} \qquad (17.10)$$

This is the general form of a transfer function in the z-plane.

17.4 Digital Filter Configurations

For the purposes of realization, digital filters are classified into nonrecursive and recursive types. In a nonrecursive realization the filter output y(nT) is obtained explicitly in terms of only past and present inputs ; i.e. previous outputs are not used to generate the current output. This is a special case of equation 17.8 in which all b_k are zero. The representation of a nonrecursive realization can be written as

$$y(nT) = \sum_{k=0}^{N} a_k x(nT-kT) \qquad (17.11)$$

or

$$H(z) = \frac{y(z)}{x(z)} = \sum_{k=0}^{N} a_k z^{-k} \qquad (17.12)$$

In a recursive realization, the current filter output y(nT) is obtained in terms of past filter outputs as well as in terms of past and present filter inputs.

17.4.1 Nonrecursive Filter Structures

The most widely known and used nonrecursive structure is the 'tapped delay line' form [24] as shown in Fig. 17.3. The transfer function of this structure is given by (see eq. 17.12)

$$H(z) = a_o + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + \ldots + a_N z^{-N} \quad (17.13)$$

Equation 17.13 can be re-expressed in many forms each of which is likely to produce different structures for H(z). e.g. see ref.[24]

17.4.2 Recursive Filter Structures

There are three well known structures [24] for the recursive filters. These are

i. Direct form

ii. Cascade form

iii. Parallel form

In the direct form the output sequence is calculated by implementing the difference equation directly (eq. 17.8).

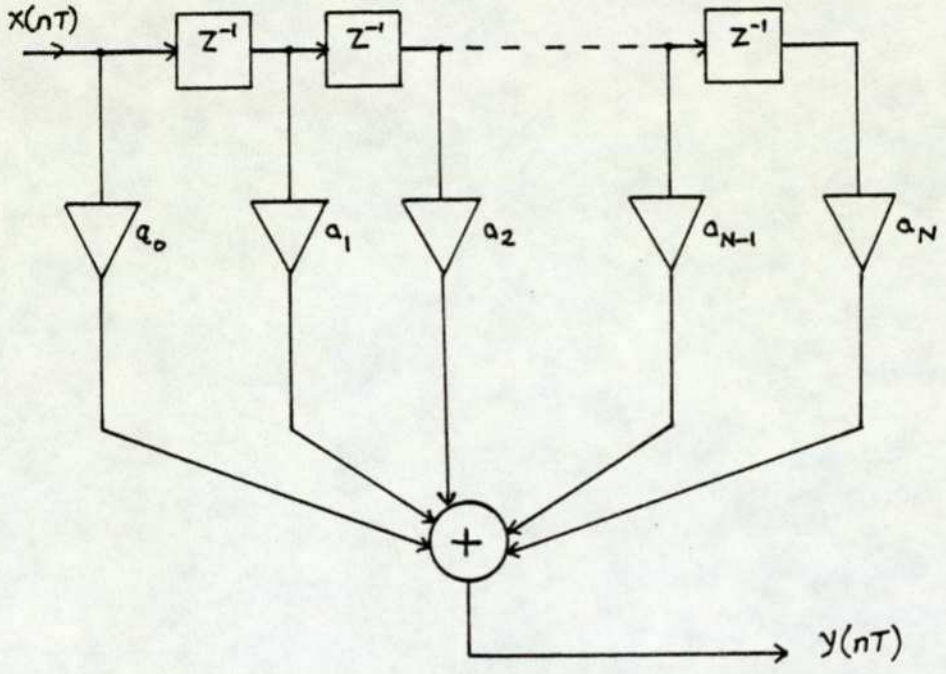


Figure 17.3 Nonrecursive filter structure .

The block diagram of the direct form is shown in Fig. 17.4 . It is evident from the figure that to synthesize the filter we need, in the case M = N , 2N memory elements and 2N sums and products. A different configuration of the direct form is the direct canonic form in which the memory elements are shared by the feedback and feedforward loops (see Fig. 17.5). The direct form has the disadvantage that the pole locations are extremely sensitive functions of the coefficients b_k for high order filters [24] .

In the cascade form the digital filter is implemented as the cascade of first and second order sections :

$$H(z) = A \prod_k \frac{1+c_k z^{-1}+d_k z^{-2}}{1+e_k z^{-1}+f_k z^{-2}} \cdot \frac{1+g_k z^{-1}}{1+h_k z^{-1}}$$

Each section can be synthesized both in the direct and in the canonic form, giving a block diagram as shown in Fig. 17.6. Here the poles are not as sensitive to the coefficients as the direct configuration [24] .

The parallel form is implemented by writing the transfer function as a sum of partial fractions

$$H(z) = A + \sum_{k=1}^{N} H_k(z)$$

where $H_k(z)$ are the second order sections of the form

$$H_k(z) = \frac{a_k+b_k z^{-1}}{1+c_k z^{-1}+d_k z^{-2}}$$

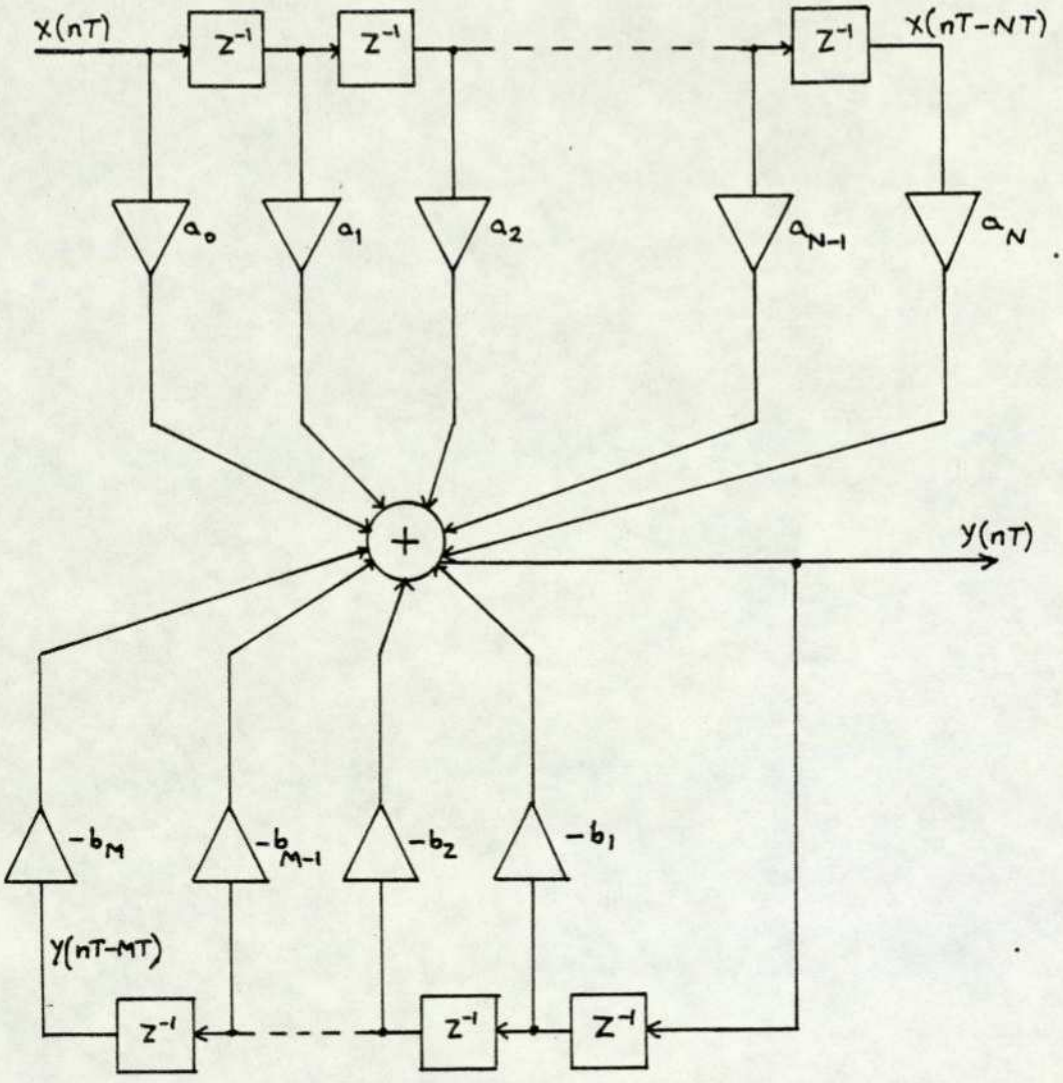


Figure 17.4 Direct realization .

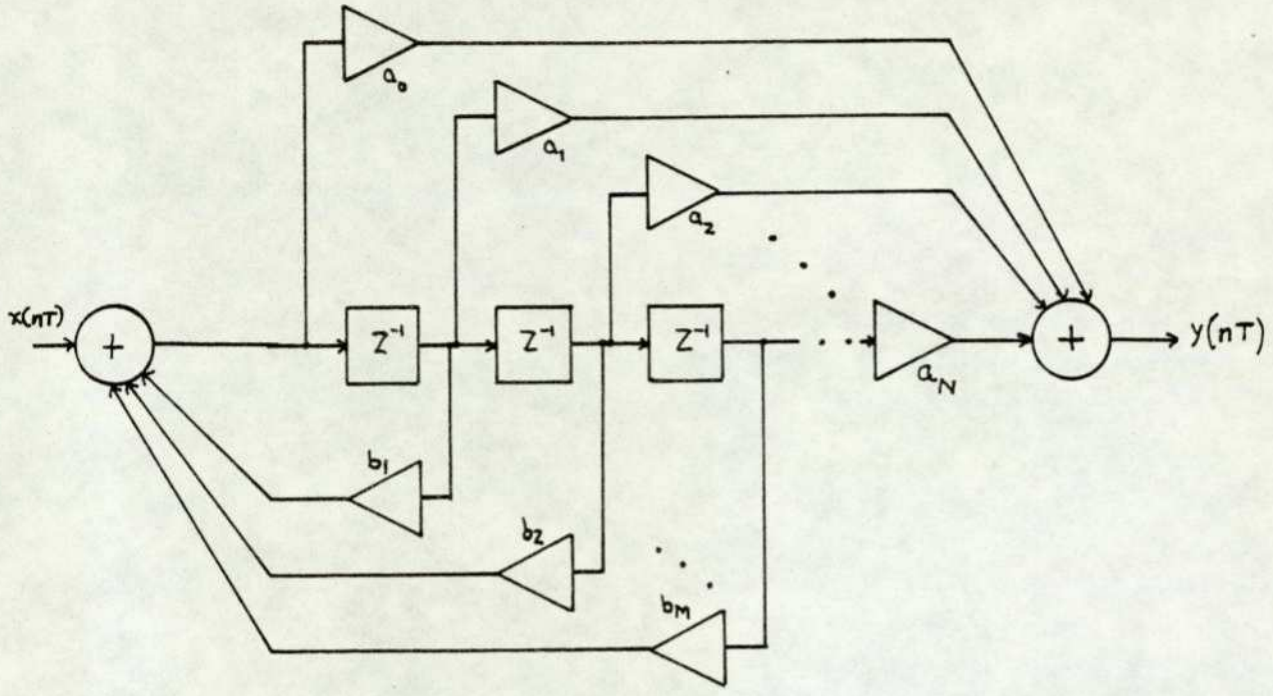


Figure 17.5 Direct canonic realization

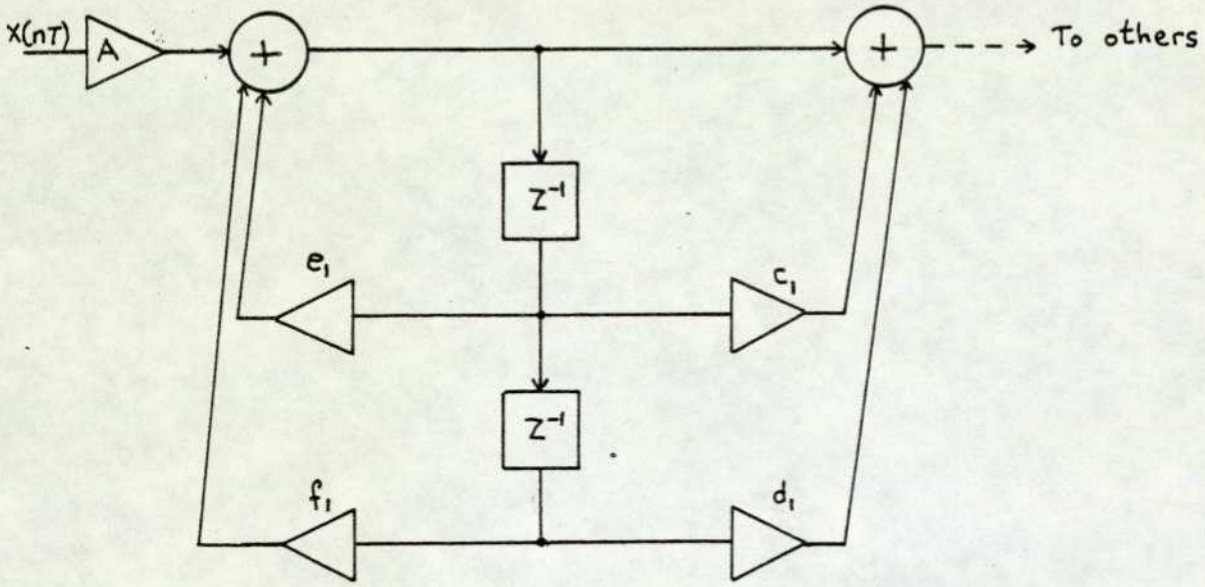


Figure 17.6 Cascade realization .

or first order sections of the form

$$H_k(z) = \frac{a_k}{1+b_k z^{-1}}$$

The block diagram of the papallel synthesis is shown in Figure 17.7.

17.5 Design of Infinite Impulse Response (IIR) Low Pass Digital Filters

The problem of IIR filter design is to determine the filter coefficients a_k and b_k such that the filter specifications are satisfied. There are two basic approaches to the design process. The first is a direct approach in which the filter coefficients are determined by analytic techniques directly from the filter specifications [24]. The second approach, to be considered here, is to determine the coefficients in a direct way from the analogue filters. This method consists of two parts :-

i. Choosing a suitable analogue filter transfer function H(s) which meets the required digital filter specification

ii. Digitization of this analogue filter by using suitable mathematical transformations.

The design of IIR digital filters by means of mathematical transformations makes it possible to view the design

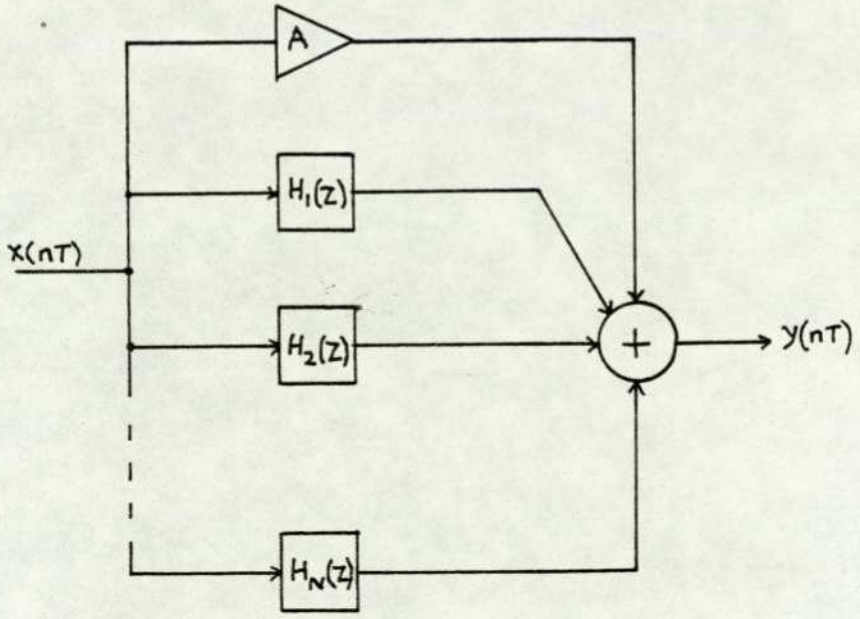


Figure 17.7 Parallel realization .

process as an extension of continuous-time filter design. The most commonly used transformation is the Bilinear transformation (see ref.[26]for other transformations).

17.5.1 Bilinear Transformation

Bilinear transformation is an algebraic method of designing digital filters from continuous-time filters. If H(s) is the transfer function of the analogue filter, the transfer function of the digital filter is obtained by making the transformation [24]

$$s \rightarrow \frac{z-1}{z+1}$$

This transformation maps the sampling interval from $-w_s/2$ to $+w_s/2$ in the s-plane onto the unit circle in the z-plane. It should be noted however that this transformation does not yield a linear map and the frequency scales are distorted relative to one another. If the analogue frequency variable is w_a , and the digital frequency variable is w_d, then the two are related by the equation

$$w_a = \tan \frac{w_d T}{2}$$

The bilinear transformation is a powerful tool in digital filtering and may be utilized with either the partial fraction expansion or the rational fraction form of the Laplace transfer function.

The steps in designing the IIR digital filters with the bilinear transformation can be summarized as follows :-

i. Let the critical frequencies (passband,stopband etc.) of the desired digital filter be w_{di} . Compute a new set of frequencies w_{ai} by

$$w_{ai} = \tan \frac{w_{di} T}{2}$$

ii. Design an analogue filter transfer function H(s) with the properties of the digital filter at the new frequencies and ranges.

iii. Replace s by (z-1)/(z+1) in H(s), and perform the algebra necessary to represent the resulting transfer function as a ratio of polynomials. This yields the desired digital filter.

Example

Design a digital filter for a 10 kHz sampling rate which is flat to -3dB in the passband of 0 to 1 kHz, and which is more than 10dB down at frequencies beyond 2 kHz. The filter must be monotonic in passband and stopband. It is known from the analogue filter theory that a Butterworth filter can meet these specifications.

The critical frequencies are

$$w_{d1}T = 2\pi X10^{3}X10^{-4} = 0.2\pi$$
$$w_{d2}T = 2\pi X2X10^{3}X10^{-4} = 0.4\pi$$

i. Compute w_{a1}, w_{a2}

$$w_{a1} = \tan\frac{0.2\pi}{2} = 0.3249$$

$$w_{a2} = \tan\frac{0.4\pi}{2} = 0.7265$$

ii. Design a Butterworth analogue filter with -3dB point at

$$w_c = w_{a1} = 0.3249$$

To find the order n, we solve

$$1+\left(\frac{w_{a2}}{w_c}\right)^{2n} = 10$$

or

$$1+\left(\frac{0.7265}{0.3249}\right)^{2n} = 10$$

giving n = 1.43, therefore we choose n = 2. A second order Butterworth filter with $w_c = 0.3249$ has poles at

$$s_{1,2} = 0.3249(-0.707 \pm j0.707)$$
$$= -0.23 \pm j0.23$$

The transfer function is given by

$$H(s) = \frac{s_1 s_2}{(s+s_1)(s+s_2)} = \frac{2X0.23^2}{(s+0.23)^2 + 0.23^2}$$

or,

$$H(s) = \frac{0.1058}{s^2+0.46s+0.1058}$$

iii. Replacing s by (z-1)/(z+1) in the above expression for H(s), we have

$$H(z) = \frac{0.1058}{\left(\frac{z-1}{z+1}\right)^2 + 0.46\left(\frac{z-1}{z+1}\right) + 0.1058}$$

or,

$$H(z) = 0.0675\ \frac{z^2+2z+1}{z^2-1.141z+0.413}$$

which is the transfer function of the required digital filter.

17.5.2 A Design Technique* for the Low Pass IIR Digital Filters

Although the bilinear transformation is a powerful tool for the design of IIR digital filters, the design process may be time consuming. This section develops the necessary equations to speed up the design process.

Since the implementation of many digital filters is typically a cascaded combination of two-pole sections, it is important to examine this case in detail :-

Two-Pole Analysis

Consider a two pole Low Pass transfer function in the

* The design technique described in this section is developed by the author .

s-plane

$$H(s) = \frac{s_1 s_2}{(s-s_1)(s-s_2)}$$

where,

$$s_1 = (-x+jy)w_a$$

$$s_2 = (-x-jy)w_a$$

and the poles are scaled to their actual analogue frequencies by the multiplier w_a .

Substituting s_1 and s_2 into H(s) gives the transfer function

$$H(s) = \frac{w_a^2(x^2+y^2)}{(s+xw_a)^2 + (yw_a)^2}$$

or,

$$H(s) = \frac{w_a^2(x^2+y^2)}{s^2 + 2xw_a s + w_a^2(x^2+y^2)}$$

Replacing s by (z-1)/(z+1) gives

$$H(z) = \frac{w_a^2(x^2+y^2)}{(\frac{z-1}{z+1})^2 + 2xw_a(\frac{z-1}{z+1}) + w_a^2(x^2+y^2)}$$

or,

$$H(z) = \frac{w_a^2(x^2+y^2)(z+1)^2}{\left[1+2xw_a+w_a^2(x^2+y^2)\right]\left[z^2-2\cdot\frac{1-w_a^2(x^2+y^2)}{1+2xw_a+w_a^2(x^2+y^2)}\cdot z + \frac{1-2xw_a+w_a^2(x^2+y^2)}{1+2xw_a+w_a^2(x^2+y^2)}\right]}$$

rewriting the above equation in a simple form:-

$$H(z) = \frac{K(1+2z^{-1}+z^{-2})}{1+Bz^{-1}+Cz^{-2}} \qquad (17.14)$$

where,

$$K = \frac{w_a^{\,2}(x^2+y^2)}{P} \qquad (17.15)$$

$$B = \frac{2.\left[w_a^{\,2}(x^2+y^2)-1\right]}{P} \qquad (17.16)$$

$$C = \frac{1-2xw_a+w_a^{\,2}(x^2+y^2)}{P} \qquad (17.17)$$

and

$$P = 1+2xw_a+w_a^{\,2}(x^2+y^2) \qquad (17.18)$$

Also it is important to recognize that the analogue frequency w_a is related to the digital frequency w_d by the relation

$$w_a = \tan \frac{w_d T}{2}$$

or,

$$w_a = \tan\left(\pi \frac{f_d}{f_s}\right) \qquad (17.19)$$

where,

f_d = desired cutoff frequency of the digital filter

f_s = sampling frequency

Single Pole Analysis

A single pole Low Pass analogue filter has the transfer function

$$H(s) = \frac{s_1}{s+s_1}$$

where, $s_1 = xw_a$

Substituting s_1 into H(s) gives

$$H(s) = \frac{xw_a}{s+xw_a}$$

Replacing s by (z-1)/(z+1) gives

$$H(z) = \frac{xw_a}{\frac{z-1}{z+1} + xw_a}$$

or,

$$H(z) = \frac{(z+1)}{\frac{1+xw_a}{xw_a} z + \frac{xw_a-1}{xw_a}}$$

Rewriting the above equation in a simple form

$$H(z) = \frac{D(1+z^{-1})}{1+Ez^{-1}} \qquad (17.19)$$

where,

$$D = \frac{xw_a}{1+xw_a} \qquad (17.20)$$

$$E = \frac{xw_a - 1}{xw_a + 1} \qquad (17.21)$$

Equations 17.14 to 17.21 can be used to design Low Pass digital filters from a knowledge of the pole positions (x and y) in the s-plane. Some examples are considered here:-

Design Steps for Low Pass digital Butterworth Filters

The poles of the n^{th} order analogue Low Pass Butterworth filter lie on a unit circle and are given by [27]

$$x_i \pm jy_i = \cos\left[\left(\frac{2i-1}{n} - 1\right)\frac{\pi}{2}\right] \pm j\sin\left[\left(\frac{2i-1}{n} - 1\right)\frac{\pi}{2}\right] \qquad (17.22)$$

where $i = 1,2,\ldots m$, $m = \frac{n}{2}$ for n even

$m = \frac{n+1}{2}$ for n odd

Since $x^2+y^2 = 1$, the only unknown in equation 17.17 to 17.21 is x . But from equation 17.22 ,

$$x = \cos\left[\left(\frac{2i-1}{n} - 1\right)\frac{\pi}{2}\right]$$

or,

$$x = \sin\left[\left(\frac{2i-1}{n}\right)\frac{\pi}{2}\right] \qquad (17.23)$$

The design steps of the Low Pass Butterworth digital filters thus can be summarized as follows (see equations 17.14 to 17.23) :-

GIVEN : Order of the filter, n

Cutoff frequency, f_d

Sampling frequency, f_s

DESIGN STEPS

1. Calculate w_a, $\quad w_a = \tan\left(\pi \frac{f_d}{f_s}\right)$

2. Calculate x_i, $\quad x_i = \sin\left[\left(\frac{2i-1}{2n}\right)\frac{\pi}{2}\right] \quad i = 1,2,.....,m$

$m = \frac{n}{2}$ for n even

$m = \frac{n+1}{2}$ for n odd

3. Calculate P_i, $\quad P_i = 1 + 2x_i w_a + w_a^2$

4. Calculate K_i, $\quad K_i = \frac{w_a^2}{P_i}$

5. Calculate B_i, $\quad B_i = \frac{2(w_a^2-1)}{P_i}$

6. Calculate C_i, $\quad C_i = \frac{1-2x_i w_a+w_a^2}{P_i}$

7. The required digital filter transfer function is given by:-

$$H(z) = \prod_{i=1}^{m} \frac{K_i(1 + 2z^{-1} + z^{-2})}{1 + B_i z^{-1} + C_i z^{-2}} \qquad \text{for n even}$$

for n odd, there is an additional 1st order factor:-

$$H(z) = \frac{D(1 + z^{-1})}{1 + Ez^{-1}}$$

where,

$$D = \frac{w_a}{1 + w_a}$$

$$E = \frac{w_a - 1}{w_a + 1}$$

Design Steps for the Low Pass Chebyshev I digital Filters

The poles of the n^{th} order Chebyshev I analog filter lie on an ellipse and are given by [28]

$$x_i = \sinh u \sin q_i$$
$$y_i = \cosh u \cos q_i \qquad (17.24)$$

where,

$$u = \frac{1}{n} \sinh^{-1} \frac{1}{\varepsilon} \qquad (17.25)$$

$$\varepsilon = \sqrt{10^{.1\Delta} - 1} \quad , \quad \Delta = \text{passband ripple (dB)} \qquad (17.26)$$

$$q_i = \left(\frac{2i - 1}{2n}\right)\pi \qquad i = 1,2,\ldots\ldots,m \qquad (17.27)$$

$$m = \frac{n}{2} \quad \text{for n even}$$

$$m = \frac{n+1}{2} \quad \text{for n odd}$$

The above design equations can be simplified as follows:-

From eq. 17.25, $u = \frac{1}{n} \sinh^{-1} \frac{1}{\varepsilon} = \frac{1}{n} \ln\left(\frac{1}{\varepsilon} + \sqrt{\frac{1}{\varepsilon^2} + 1}\right)$

Let, $\gamma = \left(\frac{1}{\varepsilon} + \sqrt{\frac{1}{\varepsilon^2} + 1}\right)^{1/n}$

Thus, $u = \ln \gamma$

and,

$$\sinh u = \frac{\gamma^2 - 1}{2\gamma}$$

$$\cosh u = \frac{\gamma^2 + 1}{2\gamma}$$

Let, $\alpha = \sinh u$, $\beta = \cosh u$

Then, equation 17.24 can be written as follows :-

$$x_i = \alpha \sin q_i$$

$$y_i = \beta \cos q_i \qquad (17.28)$$

The design steps of the digital Low Pass Chebyshev I filters can then be summarized as follows (see equations 17.14 to 17.21 and 17.26 to 17.28) :-

GIVEN : Order of the filter, n

Passband ripple (dB), Δ

Ripple bandwidth, f_r

Sampling frequency, f_s

DESIGN STEPS

1. Calculate w_a, $w_a = \tan (\pi \frac{f_r}{f_s})$

2. Calculate ε, $\varepsilon = \sqrt{10^{.1\Delta} - 1}$

3. Calculate γ, $\gamma = \left(\frac{1}{\varepsilon} + \sqrt{\frac{1}{\varepsilon^2} + 1}\right)^{1/n}$

4. Calculate α, $\alpha = \frac{\gamma^2 - 1}{2\gamma}$

5. Calculate β, $\beta = \frac{\gamma^2 + 1}{2\gamma}$

6. Calculate q_i, $q_i = \frac{(2i - 1)\pi}{2n}$ for $i = 1,2,.....,m$

$m = \frac{n}{2}$ for n even

$m = \frac{n+1}{2}$ for n odd

7. Calculate $\alpha \sin q_i$, $\beta \cos q_i$

8. Calculate t_i, $t_i = \left[(\alpha \sin q_i)^2 + (\beta \cos q_i)^2\right] w_a^2$

9. Calculate P_i $P_i = 1 + 2\alpha \sin q_i w_a + t_i$

10. Calculate K_i, $K_i = \frac{t_i}{P_i}$

11. Calculate B_i, $B_i = \frac{2(t_i - 1)}{P_i}$

12. Calculate C_i, $C_i = \frac{1-2\alpha \sin q_i w_a + t_i}{P_i}$

13. The required digital filter transfer function is :-

$$H(z) = \prod_{i=1}^{m} \frac{K_i(1 + 2z^{-1} + z^{-2})}{1 + B_i z^{-1} + C_i z^{-2}} \qquad \text{for n even}$$

for n odd, there is an additional 1st order factor :-

$$H(z) = \frac{D(1 + z^{-1})}{1 + Ez^{-1}}$$

where,

$$D = \frac{\alpha w_a}{1 + \alpha w_a}$$

and,

$$E = \frac{\alpha w_a - 1}{1 + \alpha w_a}$$

17.6 Calculating the digital Filter Coefficients Using a Programmable Pocket Calculator

The author has developed two programs for the design of Butterworth and Chebyshev I type Low Pass filters, using a Texas TI 58 type [25] programmable pocket calculator. The programs are based upon the design steps as described in the previous two sections.

17.6.1 The Butterworth Programme

This program accepts the order of the filter (n), the

cutoff frequency (f_d) and the sampling frequency (f_s) and provides the coefficients of the filter as in the following transfer function

$$H(z) = \frac{K_1(1+2z^{-1}+z^{-2})}{1+B_1z^{-1}+C_1z^{-2}} \cdot \frac{K_2(1+2z^{-1}+z^{-2})}{1+B_2z^{-1}+C_2z^{-2}} \cdots\cdots \frac{D(1+z^{-1})}{1+Ez^{-1}}$$

where the coefficients are displayed in the following order : K_1, B_1, C_1, K_2, B_2, C_2, K_3, B_3, C_3, ...D, E . A flashing display indicates that all coefficients have been calculated. The program listing is given in Appendix H and some design examples are considered here:-

Example 1

Design a 3rd order Low Pass Butterworth digital filter with a cutoff frequency of 1 KHz and a sampling frequency of 10 KHz.

The steps should be as follows (the characters entered by the user are underlined for clarity):-

	Enter	Press	Data Displayed
n	3	A	n = 3
f_s	10	B	f_s = 10
f_d	1	C	f_d = 1
		E	K_1 = 0.073801
		R/S	B_1 = -1.250516
		R/S	C_1 = 0.545723
		R/S	D = 0.24523
		R/S	E = -0.50952
		R/S	flashing

Thus the transfer function of the required filter is :-

$$H(z) = \frac{0.073801(1+2z^{-1}+z^{-2})}{1-1.250516z^{-1}+0.545723z^{-2}} \cdot \frac{0.24523(1+z^{-1})}{1-0.50952z^{-1}}$$

It took less than a minute to design this filter using the programmable pocket calculator.

Example 2

Design a 8^{th} order Low Pass digital filter with a cutoff frequency of 1 KHz and a sampling frequency of 10 KHz.

The steps should be as follows :-

	Enter	Press	Data Displayed
n	8	A	n = 8
f_s	10	B	f_s = 10
f_d	1	C	f_d = 1
		E	K_1 = 0.0856678
		R/S	B_1 = - 1.451579
		R/S	C_1 = 0.794251
		R/S	K_2 = 0.0641431
		R/S	B_2 = - 1.0868584
		R/S	C_2 = 0.3434309
		R/S	K_3 = 0.0605721
		R/S	B_3 = - 1.026351
		R/S	C_3 = 0.268640
		R/S	K_4 = 0.0719845
		R/S	B_4 = - 1.219725
		R/S	C_4 = 0.50766346
		R/S	flashing

This filter took less then two minutes to design using the programmable pocket calculator.

17.6.2 The Chebyshev I Programme

This program accepts the order of the filter (n), the ripple cutoff frequency (f_r), the sampling frequency (f_s) and the passband ripple (Δ in dB) and provides the coefficients of the filter as in the following transfer function:-

$$H(z) = \frac{D(1+z^{-1})}{1+Ez^{-1}} \cdot \frac{K_1(1+2z^{-1}+z^{-2})}{1+B_1z^{-1}+C_1z^{-2}} \cdot \frac{K_2(1+2z^{-1}+z^{-2})}{1+B_2z^{-1}+C_2z^{-2}} \cdots\cdots$$

where the coefficients are displayed in the following order : D, E, K_1, C_1, B_1, K_2, C_2, B_2, K_3, ...
Aflashing display indicates that all coefficients have been calculated. The program listing is given in Appendix I and some design examples are considered here :-

Example 1

Design a 3rd order Chebyshev I type digital filter with a ripple cutoff frequency (f_r) of 1 KHz, passband ripple (Δ) of 1 dB and a sampling frequency (f_s) of 10 KHz.

The steps should be as follows:-

	Enter	Press	Data Displayed
n	3	A	n = 3
Δ	1	B	Δ = 1
f_s	10	C	f_s = 10
f_r	1	D	f_r = 1
		E	D = 0.1383512
		R/S	E = -0.723297
		R/S	K_1 = 0.0829385
		R/S	C_1 = 0.7462467
		R/S	B_1 = -1.4144924
		R/S	-- flashing

Thus the transfer function of the required filter is:-

$$H(z) = \frac{0.1383512(1+z^{-1})}{1-0.723297} \cdot \frac{0.0829385(1+2z^{-1}+z^{-2})}{1-1.4144924z^{-1}+0.7462467}$$

It took about a minute to design this filter using the programmable pocket calculator.

Example 2

Design a 8^{th} order Chebyshev I type Low Pass digital filter with a ripple cutoff frequency (f_r) of 1 KHz, passband ripple (Δ) of 1 dB and a sampling frequency (f_s) of 10 KHz.

The steps should be as follows :-

	Enter	Press	Data Displayed
n	8	A	n = 8
Δ	1	B	Δ = 1
f_s	10	C	f_s = 10
f_r	1	D	f_r = 1
		E	K_1 = 0.0930689
		R/S	C_1 = 0.959653
		R/S	B_1 = -1.587377
		R/S	K_2 = 0.0669368
		R/S	C_2 = 0.8864575
		R/S	B_2 = -1.618710
		R/S	K_3 = 0.03176279
		R/S	C_3 = 0.8288376
		R/S	B_3 = -1.701786
		R/S	K_4 = 0.0066123
		R/S	C_4 = 0.7960923
		R/S	B_4 = -1.769642
		R/S	flashing

It took just over two minutes to design this filter using the programmable pocket calculator.

17.7 Design of Finite Impulse Response (FIR) Digital Filters

The FIR digital transfer function [23]

$$H(z) = \sum_{n=0}^{\infty} h(n) z^{-n} \qquad (17.29)$$

follows from equation 17.12 where all b_k are zero. The coefficients h(n) are known as the impulse response

coefficients of the filter and are given by [23]

$$h(n) = \frac{1}{w_s} \int_{-\frac{w_s}{2}}^{\frac{w_s}{2}} H(w) e^{jnwT} \, dw \qquad (17.30)$$

where H(w) is the frequency response of the filter given by[23]

$$H(w) = \sum_{n=-\infty}^{\infty} h(n) e^{-jnwT} \qquad (17.31)$$

As an example consider the ideal Low Pass frequency response characteristic with cutoff frequency w_c within the range $\pm w_s/2$ as shown in Figure 17.8. Applying equation 17.30, we have

$$h(n) = \frac{1}{w_s} \int_{-\frac{w_s}{2}}^{\frac{w_s}{2}} H(w) e^{jnwT} \, dw$$

$$= \frac{1}{w_s} \int_{-w_c}^{w_c} e^{jnwT} \, dw \cdot$$

$$= \frac{w_c T \sin(n w_c T)}{n\pi w_c T}$$

or,

$$h(n) = \frac{\sin(2\pi n f_c / f_s)}{n\pi} \qquad (17.32)$$

where, $n = 0, \pm 1, \pm 2, \ldots\ldots, \pm \infty$

This is an infinite series of terms which cannot be realized since it will require an infinite number of filter coefficients and an infinite delay. It is necessary therefore

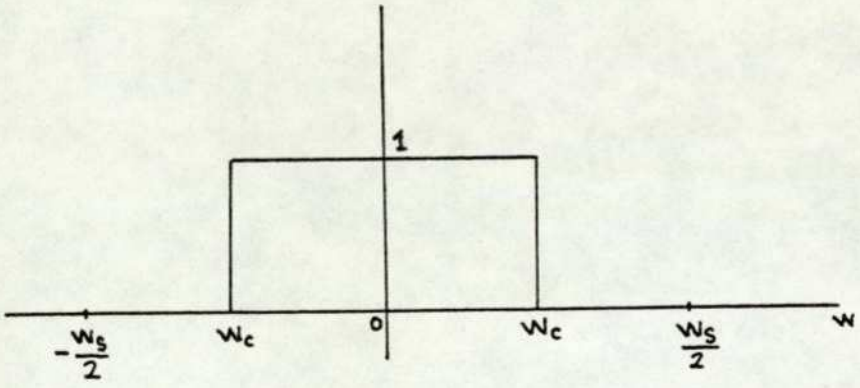


Figure 17.8 Ideal Low Pass filter

to truncate in some way the coefficients obtained. This truncation unfortunately leads to the Gibbs phenomenon [24] which causes the appearance of an overshoot in the approximated frequency response near a discontinuity. Also, the error introduced by truncation of the coefficients is independent of the length of the filter. When more coefficients are included in the design the maximum error is the same but the oscillations are confined to a smaller frequency range. One way to improve the response of a truncated filter is by using window functions to modify the coefficients $h(n)$ in a prescribed way by multiplication i.e. $h'(n) = h(n).w(n)$ where $h'(n)$ are the modified impulse response coefficients and $w(n)$ are the coefficients of the window function used. Some of the most commonly used window functions are described below :-

i. <u>The Hamming Window [24]</u>

The coefficients of the Hamming window are given by*

$$w(n) = 0.54 + 0.46\cos\left(\frac{2\pi n}{N-1}\right) \quad , n = -\frac{N-1}{2} , \ldots , \frac{N-1}{2}$$

ii. <u>The Lanczos Window [24]</u>

The coefficients of the Lanczos window are given by*

$$w(n) = \left[\frac{\sin\left(\frac{2\pi n}{N-1}\right)}{\frac{2\pi n}{N-1}}\right]^{L} \quad , n = -\frac{N-1}{2}, \ldots, \frac{N-1}{2}$$

* This equation applies when N is odd .

where L is an integer (Lanczos parameter) greater than zero.

A computer program (in FORTRAN) was developed by the author for the design of Low Pass, High Pass, Band Pass and Band Stop type FIR digital filters by using the two windows described above. The program calculates (both for N odd and even) the windowed impulse response coefficients $h'(n)$ and plots the resulting frequency response of the filter. A typical run of the program is shown in Figure 17.9 and Figure 17.10. The program listing is given in Appendix F .

17.7.1 Optimal FIR Digital Filter Design

The filters obtained by the window technique are not optimum because they do not satisfy any optimality constraint. However this technique can be useful in applications where the speed and simplicity of the design are the fundamental constraints.

The optimum FIR digital filters are defined in the sense that the maximum error $E(e^{jw})$ between the desired frequency response $D(e^{jw})$ and the approximated frequency response $H(e^{jw})$ is minimum [24] where

$$E(e^{jw}) = \left| W(e^{jw}).\left[D(e^{jw}) - H(e^{jw})\right]\right|$$

where $W(e^{jw})$ is a positive weighting function which

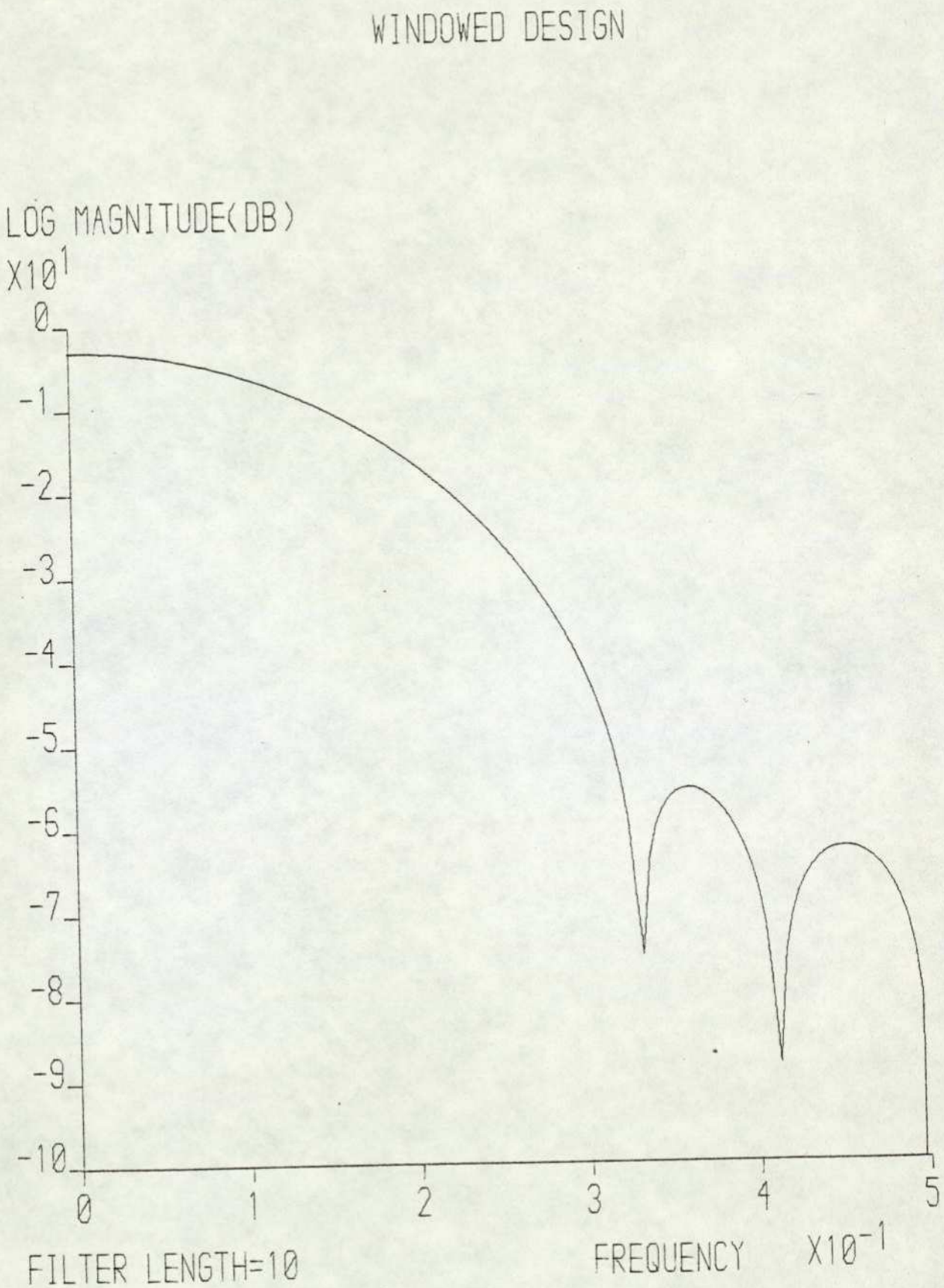


Figure 17.9 Lanczos windowed Low Pass filter (Lanczos parameter = 2, $f_d/f_s = 0.1$). Frequency normalized to f_s .

```
**********************************************************************

                FINITE IMPULSE RESPONSE(FIR)
                LINEAR PHASE DIGITAL FILTER DESIGN
                WINDOWING ALGORITHM

                LOW PASS FILTER

                LANCZOS WINDOWED DESIGN

          FILTER LENGTH =  10

          LANCZOS PARAMETER =  2

          *****IMPULSE RESPONSE*****
               H(  1) = 0.30015150E-22 =H(  10)
               H(  2) = 0.50917130E-02 =H(   9)
               H(  3) = 0.40537625E-01 =H(   8)
               H(  4) = 0.11741418E 00 =H(   7)
               H(  5) = 0.18886485E 00 =H(   6)

**********************************************************************
```

Figure 17.10

allows different errors to be defined for different intervals.

A computer program based upon the above optimality criterion is given by Mc Clellan [26]. The program is very general and it can be used to design multiple passband/stop-band filters, Hilbert transformers and differentiators. This program was expanded by the author to include the following options :-

i. The frequency response of the designed filter can be plotted.

ii. The filter coefficients h(n) can be rounded to the nearest binary number, where the width of the binary number is defined by the user.

iii. The frequency response of the filter can be plotted with the coefficients rounded as described in ii.

A typical run of the program is shown in Figure 17.11 and Figure 17.12. The program listing is given in Appendix G .

17.8 Frequency Transformation of Low Pass Filters

The problem of transforming a given Low Pass filter to another filter is achieved by the frequency transformation techniques.

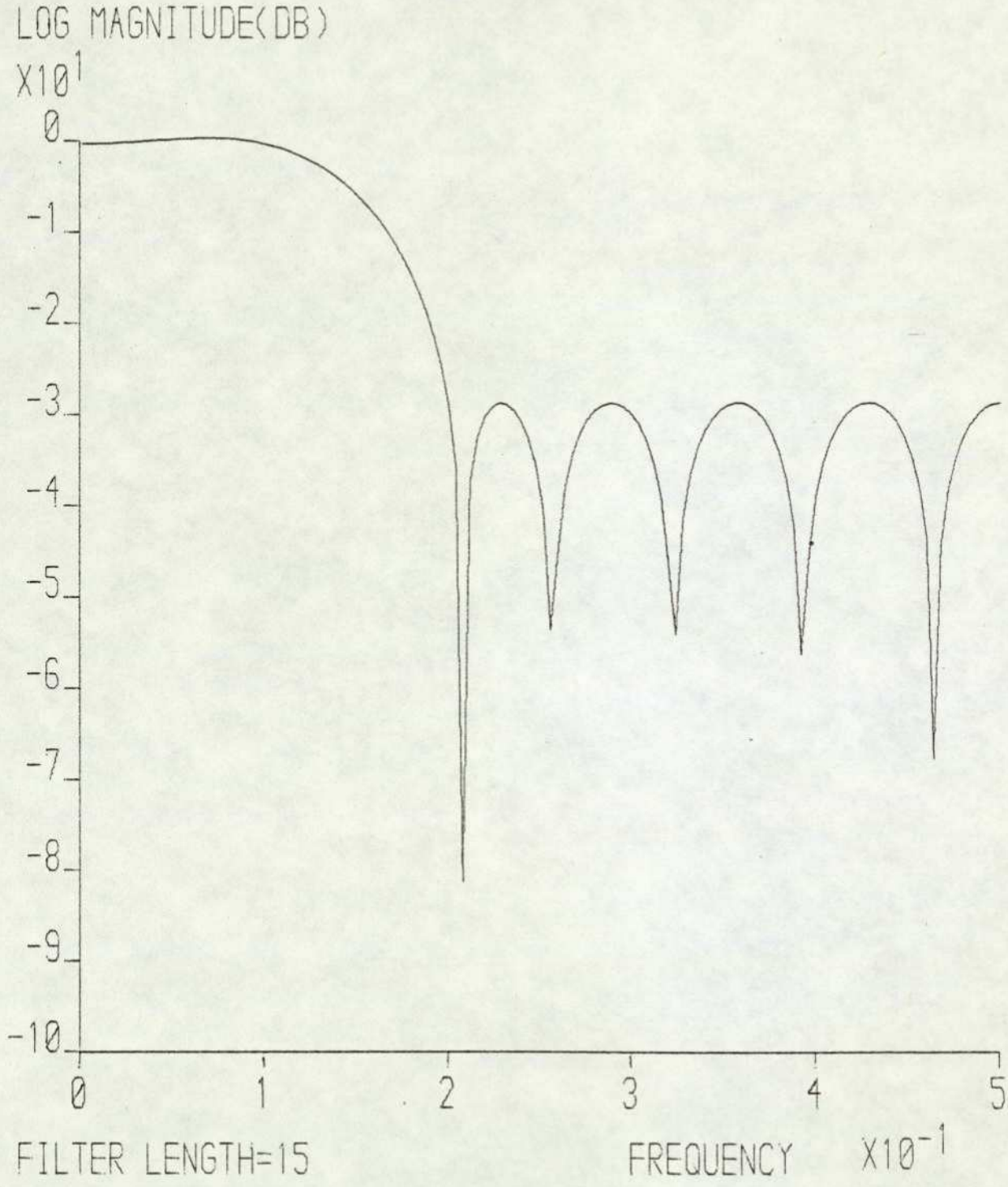


Figure 17.11 Optimum Low Pass filter design
(see Fig. 17.12 for the design data)

```
**********************************************************************

                    FINITE IMPULSE RESPONSE (FIR)
                    LINEAR PHASE DIGITAL FILTER DESIGN
                    REMEZ EXCHANGE ALGORITHM

                    BANDPASS FILTER

             FILTER LENGTH = 15

             *****IMPULSE RESPONSE****
                  H(  1) = 0.13256093E-01 =H(  15)
                  H(  2) =-0.22760641E-01 =H(  14)
                  H(  3) =-0.44760623E-01 =H(  13)
                  H(  4) =-0.38051214E-01 =H(  12)
                  H(  5) = 0.27122399E-01 =H(  11)
                  H(  6) = 0.14196026E 00 =H(  10)
                  H(  7) = 0.25438213E 00 =H(   9)
                  H(  8) = 0.30131086E 00 =H(   8)

                      BAND  1          BAND  2          BAND
 LOWER BAND EDGE      0.000000000      0.200000000
 UPPER BAND EDGE       .100000000      0.500000000
DESIRED VALUE         1.000000000      0.000000000
 WEIGHTING            1.000000000      1.000000000
DEVIATION             0.036392326      0.036392326
 DEVIATION IN DB      0.632478764     -28.779803827

 EXTREMAL FREQUENCIES
    0.00000000   0.0664063   0.1000000   0.2000000   0.2273438
    0.2898438   0.3562500   0.4265625   0.5000000

**********************************************************************


                    COEFFICIENT ACCURACY =16 BITS


             ****IMPULSE RESPONSE(ROUNDED)****
                  H(  1) = 0.13244629E-01 =H(  15)
                  H(  2) =-0.22735596E-01 =H(  14)
                  H(  3) =-0.44738770E-01 =H(  13)
                  H(  4) =-0.38024902E-01 =H(  12)
                  H(  5) = 0.27130127E-01 =H(  11)
                  H(  6) = 0.14196777E 00 =H(  10)
                  H(  7) = 0.25439453E 00 =H(   9)
                  H(  8) = 0.30130005E 00 =H(   8)

**********************************************************************
```

Figure 17.12

If we have a Low Pass digital filter with the transfer function $H(z^{-1})$ and the cutoff frequency w_d rad/s, then replacing z^{-1} by $-z^{-1}$ will yield a High Pass filter [24] with the cutoff frequency w_h, where

$$w_h = \frac{\pi}{T} - w_d$$

or,

$$f_h = \frac{f_s}{2} - f_d$$

where, f_h = cutoff frequency of the High Pass filter
f_d = cutoff frequency of the Low Pass filter
f_s = sampling frequency

Similarly, a Low Pass filter can be transformed to a Band Pass filter [24] by replacing z^{-1} by $-z^{-2}$ in which case the critical frequencies of the Band Pass filter are given by:-

$$f_1 = \frac{1}{2}\left(\frac{f_s}{2} - f_d\right)$$

$$f_2 = \frac{1}{2}\left(\frac{f_s}{2} + f_d\right)$$

$$f_0 = \frac{f_s}{4}$$

where, f_1, f_2 = cutoff frequencies of the Band Pass filter
f_0 = center frequency of the Band Pass filter
f_d = cutoff frequency of the Low Pass filter
f_s = sampling frequency

17.9 Fractional Number Representation

The digital filter coefficients are usually scaled in a way to prevent overflow. It is convenient to represent the filter coefficients and signal values in a pure fractional form in two's complement notation. The MSB of the binary number represents the sign (positive or negative) while the remaining bits represent the fractional part of the filter coefficients. Thus, for a 16-bit data:-

$$\boxed{s\ 2^{-1}\ 2^{-2}\ 2^{-3} \ldots\ldots\ldots\ldots\ 2^{-15}}$$

s = 0 for positive
s = 1 for negative

With this representation the largest positive number is +32767/32768 (= H'7FFF') and the largest negative number is -1 (= H'8000') .

Example 1

Represent the decimal number 0.40625 in two's complement using a 16-bit data width :-

$$0.40625 = 0X2^{-1} + 1X2^{-2} + 1X2^{-3} + 0X2^{-4} + 1X2^{-5}$$

Therefore, the required binary number is

0011 0100 0000 0000 or H'3400' in hexadecimal

Example 2

Represent the decimal number -0.40625 in two's complement using a 16-bit data width :-

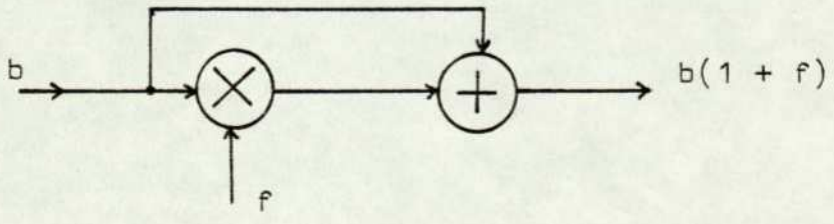
The negative numbers in two's complement are obtained by complementing the corresponding positive number and incrementing by one :-

$$
\begin{aligned}
N &= 0011\ 0100\ 0000\ 0000 \\
\overline{N} &= 1100\ 1011\ 1111\ 1111 \\
&\underline{+\qquad\qquad\qquad\qquad 1} \\
-N &= 1100\ 1100\ 0000\ 0000
\end{aligned}
$$

or, H'CC00' in hexadecimal

If some of the filter coefficients are greater than 1, they can be implemented as follows :-

Let the decimal number be $N = 1 + f$ where f is the fractional part represented in two's complement as described above. Suppose that it is required to multiply this number with the variable b. Then the product $N' = (1 + f)b$ can be implemented as :-



When two fractional numbers are multiplied using an integer multiplication-routine, it is necessary to adjust the product by multiplying it with 2 (i.e. shifting left by one position) as illustrated in the following example

Example 3

Multiply the fractional number 0.25 with 0.375 using an integer multiplication-routine (assume an 8-bit data width).

0.25 = 0010 0000

0.375 = 0011 0000

When these numbers are multiplied with an integer multiplication-routine, the product is :-

0000 0110 0000 0000

which is the fractional number 0.046875. But when 0.25 is multiplied with 0.375 the product must be 0.09375 which is represented as:-

0.09375 = 0000 1100 0000 0000

Hence, It is necessary to shift the product 0.046875 left by one position (0.046875X2 = 0.09375) to obtain the correct answer.

CHAPTER 18

18. DESIGN OF REAL-TIME LOW PASS IIR DIGITAL FILTERS USING THE BIT-SLICE MICROCOMPUTER

18.1 Introduction

This Chapter describes the special instructions invented for the design of real-time Low Pass IIR digital filters, using the bit-slice microcomputer.

The real-time digital filtering process can be carried out in two distinct phases (see Fig. 18.1) : the initialization routine, and the digital filter loop.

During the initialization routine the filter coefficients a_k and b_k are loaded to the system and some registers are set to a known state. Since the time taken to execute the initialization routine is not critical, there is generally no need to invent special instructions for this routine.

The digital filter loop is where the actual filtering process is carried out. This loop is executed every time the A/D converter generates a new sample. The execution time of the digital filter loop determines the maximum sampling frequency and hence the bandwidth of digital filtering. For high frequency filtering process the execution time of the digital filter loop should be minimized by the choice of special instructions.

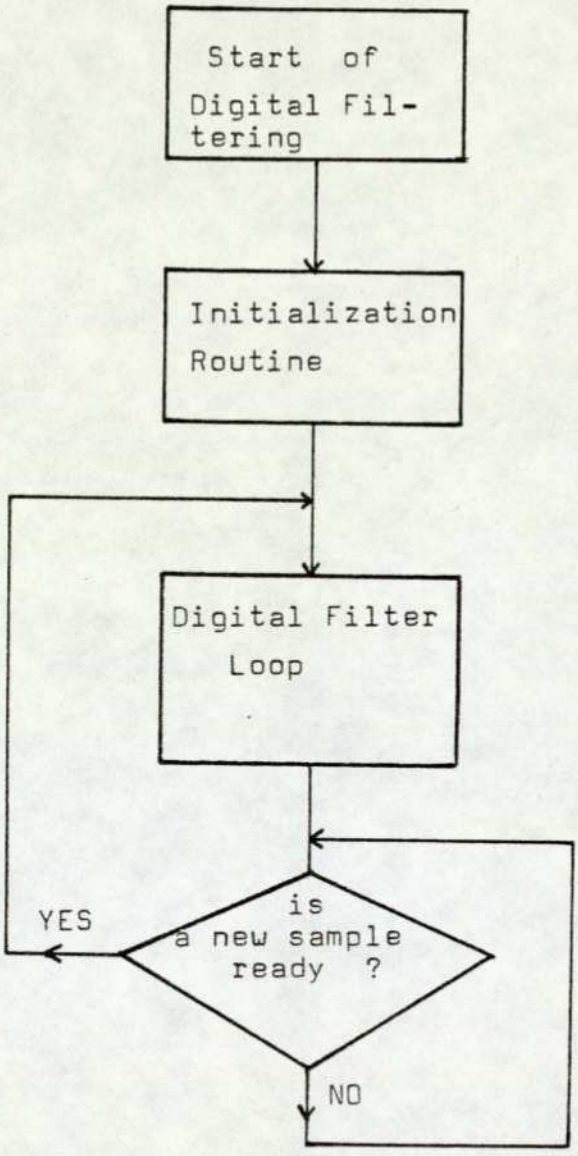


Figure 18.1 The two phases of real-time digital filtering are: the initialization routine, and the digital filter loop .

18.2 The Hardware Set-up

The external interrupt facilities were used to synchronize the processor with the sampling intervals such that the occurence of each sample interrupts the processor to initiate the execution of the digital filter loop.

Figure 18.2 shows the external connections made for the design of digital filters. The external clock generates a square-wave signal at the sampling frequency and drives the 'convert command' input (T_s) of the A/D converter (see Chapter 6.4.2). The trailing edge of the pulses applied to this input initiates the conversion process. After 10 us the 'strobe' output (STRB) of the A/D converter makes a HIGH-to LOW transition to indicate that the conversion is complete and the input signal (via the sample/hold amplifier) is digitized and it is available at the input of Port 1. The 'strobe' output was externally connected to one of the external interrupt inputs of the microcomputer. Thus, when a new sample is ready the processor is interrupted (a HIGH-to-LOW on an external interrupt input) and execution continues from the first instruction of the digital filter loop.

18.3 Computing the Filter Algorithm

It is shown in Chapter 17 that an n^{th} order Low Pass Butterworth or Chebyshev I type digital filter has the

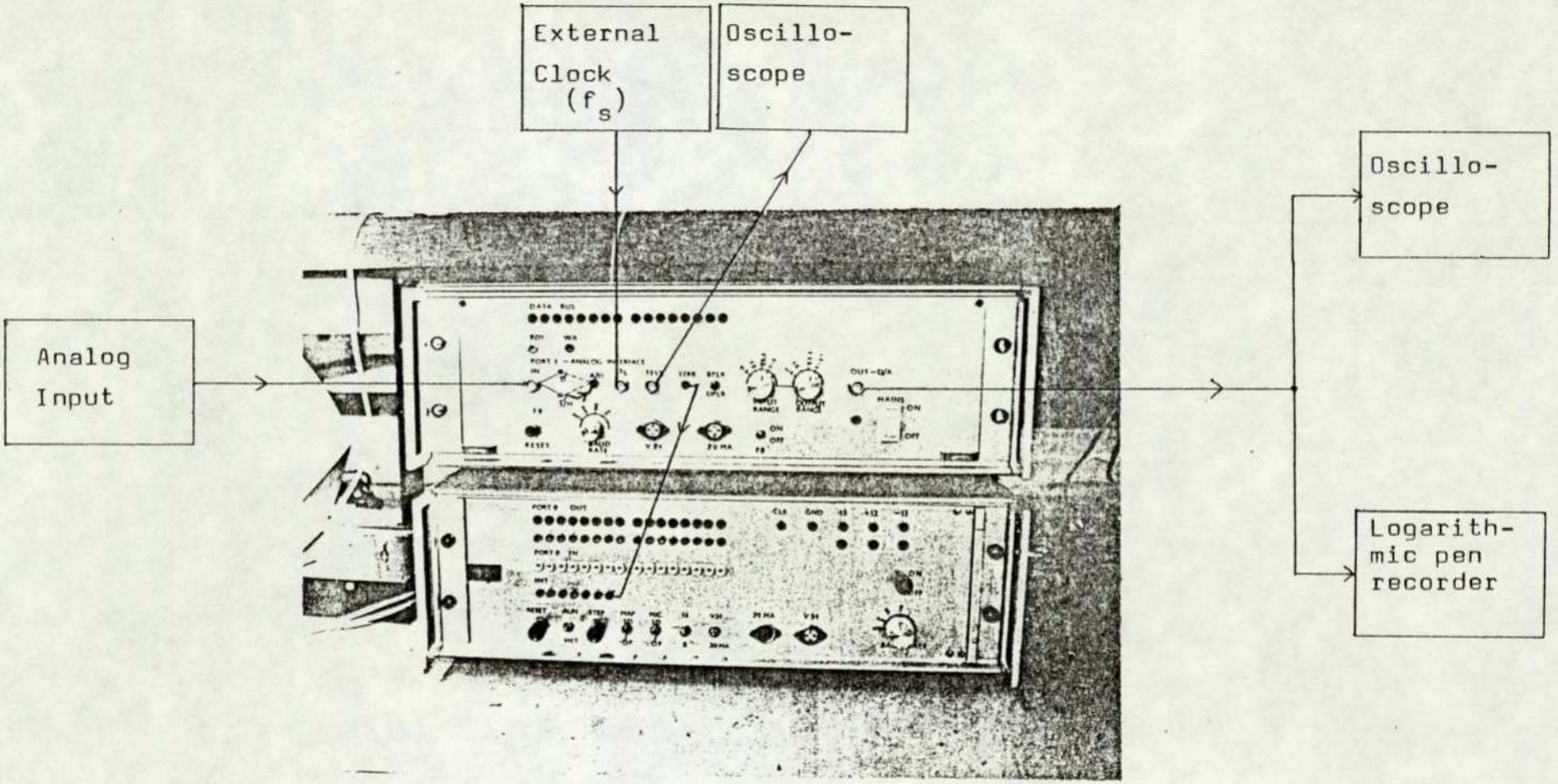


Figure 18.2 Connections for the implementation of digital filters.

transfer function of:-

$$H(z) = \frac{K_1(1+2z^{-1}+z^{-2})}{1+B_1z^{-1}+C_1z^{-2}} \cdot \frac{K_2(1+2z^{-1}+z^{-2})}{1+B_2z^{-1}+C_2z^{-2}} \cdots\cdots\cdots \frac{D(1+z^{-1})}{1+Ez^{-1}} \quad (18.1)$$

In order to implement the above equation, it is necessary first to investigate how a second order and a first order section can be implemented efficiently on their own.

18.3.1 A Second Order Section

Consider the transfer function of a second order section:

$$H(z) = \frac{y(z)}{u(z)} = \frac{K(1+2z^{-1}+z^{-2})}{1+Bz^{-1}+Cz^{-2}} \quad (18.2)$$

let, $$\frac{y(z)}{u(z)} = \frac{y(z)}{q(z)} \cdot \frac{q(z)}{u(z)} \quad (18.3)$$

where, $$\frac{q(z)}{u(z)} = \frac{K}{1+Bz^{-1}+Cz^{-2}} \quad (18.4)$$

Then, from equation 18.2 and 18.3 ,

$$\frac{y(z)}{q(z)} = 1+2z^{-1}+z^{-2} \quad (18.5)$$

Equations 18.4 and 18.5 can be written in a more appropriate form:-

writing equation 18.4 in terms of q(z) :

$$q(z) = Ku(z) - Bq(z)z^{-1} - Cq(z)z^{-2} \qquad (18.6)$$

and writing equation 18.5 in terms of q(z) :

$$y(z) = q(z) + 2q(z)z^{-1} + q(z)z^{-2} \qquad (18.7)$$

let the two state variables x_1 and x_2 be defined as :

$$x_1 = q(z)z^{-1} \qquad \text{and} \qquad x_2 = q(z)z^{-2} = x_1 z^{-1} \qquad (18.8)$$

then equations 18.6 and 18.7 can be written as :

$$q(z) = Ku(z) - Bx_1 - Cx_2 \qquad (18.9)$$

$$y(z) = q(z) + 2x_1 + x_2 \qquad (18.10)$$

with, $x_1 = q(z)z^{-1}$ (18.11)

and $x_2 = x_1 z^{-1}$ (18.12)

Equations 18.9 to 18.12 can be used to implement a second order section, which requires the following operations to be carried out (see also Fig. 18.3) during every sampling interval :-

$$Ku - Bx_1 - Cx_2 \longrightarrow q$$

$$q + 2x_1 + x_2 \longrightarrow y$$

$$x_1 \longrightarrow x_2$$

$$q \longrightarrow x_1$$

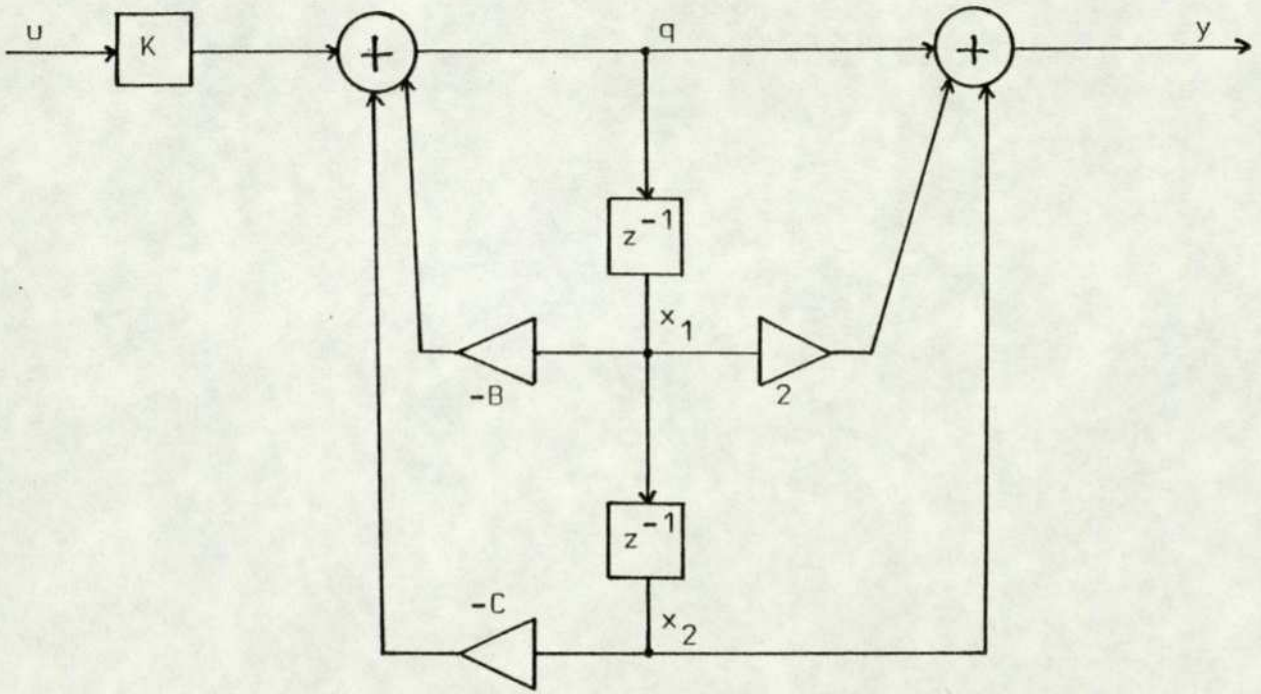


Figure 18.3 Implementing a second order Low Pass section .

Remembering that a new cycle is initiated every time an external interrupt is present, the steps in implementing a second order section can be summarized as follows (note that u, x_1, x_2, and q are general purpose registers) :-

$$\underline{\text{GIVEN}} : \quad H(z) = \frac{K(1+2z^{-1}+z^{-2})}{1+Bz^{-1}+Cz^{-2}}$$

IMPLEMENTATION :

Initialization routine

LOOP: Input to 'u' via A/D converter

Calculate : $Ku - Bx_1 - Cx_2$

Store the result in 'q'

Calculate : $q + 2x_1 + x_2$

Output the result via D/A converter

Perform : $x_1 \longrightarrow x_2$

Perform : $q \longrightarrow x_1$

Enable external interrupts

WAIT: If interrupt detected goto LOOP else goto WAIT

The choice of an instruction set for the efficient implementation of a second order section is described later.

18.3.2 A First Order Section

Consider the transfer function of a first order section:

$$H(z) = \frac{y(z)}{u(z)} = \frac{D(1 + z^{-1})}{1 + Ez^{-1}}$$

Following an analysis similar to the second order section, it can be shown that the implementation of a first order section requires the following operations to be performed during every sampling interval (see also Fig. 18.4) :-

$$Du - Ex_1 \longrightarrow q$$
$$q + x_1 \longrightarrow y$$
$$q \longrightarrow x_1$$

18.3.3 Cascading the Filter Sections

When several sections are cascaded, the output of one section becomes the input to the next section. Figure 18.5 shows the implementation of an n^{th} order filter by using the basic equations derived in the previous two sub-sections. This figure can be simplified as follows : since the contents of registers q_i are only used in the implementation of the filter section i , we can let $q_i = q$. Also, since the filters are processed in cascade and the inputs are not stored, we can let the outputs and inputs of all sections be stored in 'w', where 'w' is a general purpose register. The block diagram of the n^{th} order filter is then as shown in Figure 18.6. It is clear from this figure that the

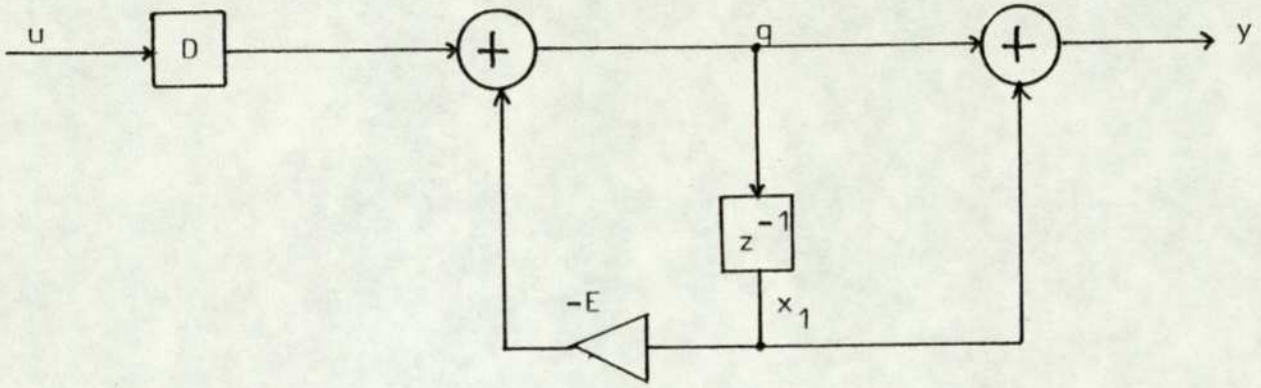


Figure 18.4 Implementing a first order Low Pass section .

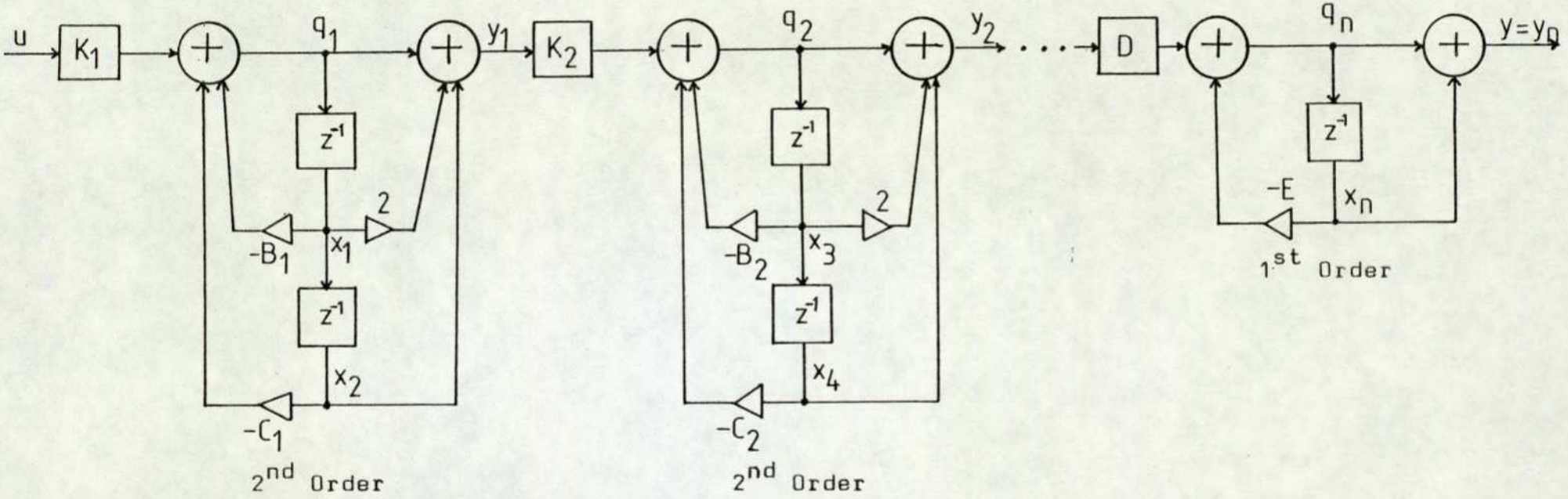


Figure 18.5 Implementing an n^{th} Order Low Pass Filter (n odd) .

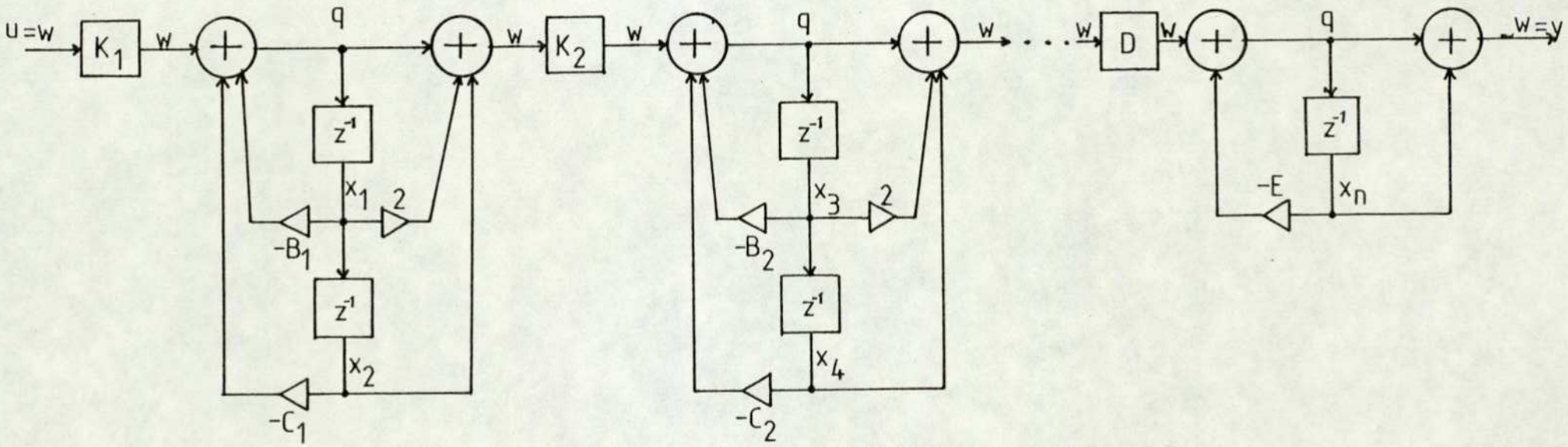


Figure 18.6

implementation of an n^{th} order filter requires the following operations to be performed at every sampling interval (x_i, q and w are general purpose registers) :-

1.	LOOP:	Input to 'w' via A/D converter	INPUT SECTION
2.		$K_i w \longrightarrow w$	INTERMEDIATE SECTIONS
3.		$w - B_i x_{2i-1} - C_i x_{2i} \longrightarrow q$	Repeat 2 to 6 for $i=1,2, \ldots, m$
4.		$2x_{2i-1} + x_{2i} + q \longrightarrow w$	$m = \frac{n}{2}$ for n even
5.		$x_{2i-1} \longrightarrow x_{2i}$	$m = \frac{n-1}{2}$ for n odd
6.		$q \longrightarrow x_{2i-1}$	
7.		$Dw \longrightarrow w$	FIRST ORDER SECTION
8.		$w - Ex_n \longrightarrow q$	(only for n odd)
9.		$x_n + q \longrightarrow w$	
10.		$q \longrightarrow x_n$	
11.		Output 'w' via D/A converter	OUTPUT SECTION
12.		Enable external interrupts	
13.	WAIT:	If interrupt detected goto LOOP else goto WAIT	

18.4. Choosing an Instruction Set

A set of special instructions were invented based upon the implementation technique described in the previous sub-section of this Chapter.

The filter coefficients were stored in sequential memory locations as : $K_1, -B_1, -C_1, K_2, -B_2, -C_2, \ldots\ldots, D, -E$ and were addressed by the Data Counter E (DCE). Similarly, the state variables x_i were also stored in sequential memory locations (see Fig. 18.7), addressed by the Data Counter F (DCF). Register 4 (R_4) of the ALU was used as the 'w' register and the 'Q' register of the ALU was used as the 'q' register. The following instructions were invented (implemented in the microprogram RAM and the Mapping RAM):-

18.4.1 Instruction 'IN'

This instruction implements the input section (see Section 18.3.3) of the filter. It performs the following operation :-

$$(\text{A/D Converter}) \longrightarrow R_4$$

The IN instruction occupies two successive locations in the main memory as :-

```
8000    IN
0100
```

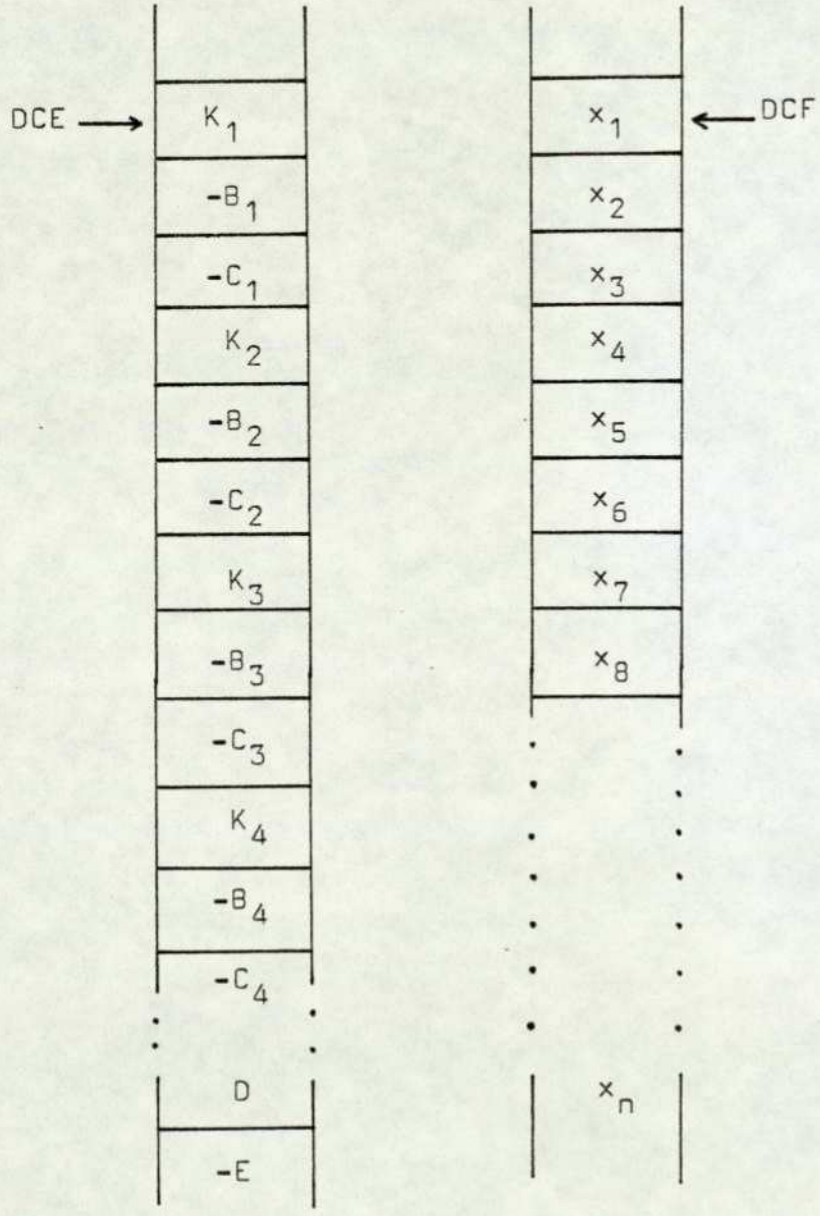

DCE
K1
-B1
-C1
K2
-B2
-C2
K3
-B3
-C3
K4
-B4
-C4
D
-E
x1
DCF
x2
x3
x4
x5
x6
x7
x8
xn

Figure 18.7

where H'80' is the Op-Code chosen for this instruction. The execution time of this instruction is 4 cycles (4 μs with 1 MHz clock).

18.4.2 Instruction 'LPFL 2'

The LPFL 2 instruction implements a second order section (cascadable) completely as described in Section 18.3.3. It performs the following operations :-

$((DCE)) \times (R_4) \longrightarrow R_4$

$(R_4) + ((DCE+1)) \times ((DCF)) + ((DCE+2)) \times ((DCF+1)) \longrightarrow Q$

$2 \times ((DCF)) + ((DCF+1)) + (Q) \longrightarrow R_4$

$((DCF)) \longrightarrow ((DCF+1))$

$(Q) \longrightarrow ((DCF))$

$(DCF) + 2 \longrightarrow DCF$

$(DCE) + 3 \longrightarrow DCE$

The LPFL 2 instruction occupies only one location in the main memory as :-

```
8100   LPFL 2
```

where H'81' is the Op-Code chosen for this instruction. The execution time of this instruction is 100 cycles (100 μs with 1 MHz clock).

18.4.3 Instruction 'LPFG 2'

This instruction is very similar to the LPFL 2 instruction, except that here the filter coefficients B_i are assumed to be greater than 1 and hence B_i is added (see Chapter 17.8) to the sum when calculating the 'Q'. It performs the following operations:-

((DCE)) X (R_4) $\longrightarrow$ R_4

(R_4) + ((DCE+1)) X ((DCF)) + ((DCE+2)) X ((DCF+1)) + ((DCF)) $\longrightarrow$ Q

2 X ((DCF)) + ((DCF+1)) + (Q) $\longrightarrow$ R_4

((DCF)) $\longrightarrow$ ((DCF+1))

(Q) $\longrightarrow$ ((DCF))

(DCF) + 2 $\longrightarrow$ DCF

(DCE) + 3 $\longrightarrow$ DCE

The LPFG 2 instruction occupies only one location in the main memory as :-

```
8200   LPFG 2
```

where H'82' is the Op-Code chosen for this instruction. The execution time of this instruction is 102 cycles (102 μs with 1 MHz clock).

18.4.4 Instruction 'LPF 1'

This instruction implements the first order section (see Section 18.3.3). The operations performed are as follows:-

$((DCE)) \times (R_4) \longrightarrow R_4$

$(R_4) + ((DCE+1)) \times ((DCF)) \longrightarrow Q$

$((DCF)) + (Q) \longrightarrow R_4$

$(Q) \longrightarrow ((DCF))$

$(DCF) + 1 \longrightarrow DCF$

$(DCE) + 2 \longrightarrow DCE$

The LPF 1 instruction occupies only one location in the main memory as :-

```
8300    LPF 1
```

where H'83' is the Op-Code chosen for this instruction. The execution time of this instruction is 62 cycles (62 μs).

18.4.5 Instruction 'OUT'

This instruction implements the output section of the filter. The operations performed are as follows :-

$(R_4) \longrightarrow$ D/A Converter

(Auxiliary Register E)* $\longrightarrow$ DCE

(Auxiliary Register F)* $\longrightarrow$ DCF

Enable interrupts

Wait for interrupt

The OUT instruction occupies three locations in the main memory as :-

```
84EF   OUT
0100
8500
```

where H'84' is the Op-Code chosen for this instruction. The execution time of this instruction is (up to the end of 'enable interrupts') 7 cycles (7 μs).

18.4.6 The Initialization Routine

As the execution time of the initialization routine is not critical, the fixed instructions (in the microprogram PROM) were used for this routine.

The following operations should be performed in the

* see the initialization routine.

initialization routine :-

1. Set the interrupt service address in the main memory .

2. Load the interrupt mask register such that the timer interrupts are disabled and one external interrupt input is enabled.

3. Set the Data Counter DCE to the beginning of the coefficient table.

4. Set the Data Counter DCF to the beginning of the table where the state variables x_i are to be stored.

5. Store the DCE in Auxiliary Register *E

6. Store the DCF in Auxiliary Register *F

Note that at steps 5 and 6 the Data Counters are stored in the Auxiliary Registers*. By this approach it is not necessary to re-initialize the Data Counters at every sampling interval since the instruction OUT performs this operation. This technique saves 4 cycles in the digital filter loop.

* The registers provided by the 'Register Extension Logic (REL)' of the APU are named as the 'Auxiliary Registers' (see Chapter 5.4).

18.5 Design Examples

This section gives examples of the digital filters designed using the special instructions described in the previous section. The filter coefficients were calculated using a programmable pocket calculator as described in Chapter 17.6 . The filter coefficients were stored starting from the memory location H'0550' . Similarly the locations starting from H'0500' were reserved to store the state variables. Interrupt input 0 (see Chapter 9) was used to synchronize the processor with the sampling rate (note that the service address of interrupt 0 is the address contained in memory location H'0407'). The frequency response of the filters were plotted by using a beat frequency oscillator (B & K type 1022) and a logarithmic pen recorder (B & K type 3205). The frequency response of the filters designed were verified by calculating the response using a calculator .

18.5.1 First Order Butterworth Filter

This kind of filter was designed for a sampling frequency of f_s = 5 KHz and a cutoff frequency of f_d = 500 Hz (i.e. $f_s/f_d = 10$). The transfer function of the filter was found to be :-

$$H(z) = \frac{0.245237(1 + z^{-1})}{1 - 0.509525z^{-1}}$$

Representing the coefficients in 16 bit fractional form, we get :-

D = 0.245237 = H'1F64'

-E = 0.509525 = H'4138'

The program written to implement this filter was as follows:-

```
*
* INITIALIZATION ROUTINE
*
0600   3600   SM H'FE00'        Mask all interrupts except 0
0601   FE00
0602   15E0   LI E, H'0550'     Set DCE to start of coefficient
0603   0550                     table
0604   15F0   LI F, H'0500'     Set DCF to start of state var-
0605   0500                     iable table
0606   1AEE   PUSH E,E          Store DCE in Aux. Reg. E
0607   1AFF   PUSH F,F          Store DCF in Aux. Reg. F
*
* DIGITAL FILTER LOOP
*
0608   8000   IN                Input section
0609   0100
060A   8300   LPF 1             First Order Section
060B   84EF   OUT               Output Section
060C   0100
060D   8500
*
* END
*
```

Also the following data was loaded to the memory :-

Address	Data	
0407	0608	Interrupt service address
0550	1F64	Coefficient D
0551	4138	Coefficient -E

The frequency response of this filter is as shown in Figure 18.8

18.5.2 First Order Chebyshev I Filter

This kind of filter was designed for a sampling frequency of f_s =5 KHz and a ripple cutoff frequency of f_r = 500 Hz (i.e. f_s/f_r = 10). The passband ripple was chosen to be 1 dB and the transfer function of the filter was found to be :-

$$H(z) = \frac{0.3897(1 + z^{-1})}{1 - 0.2206z^{-1}}$$

Representing the coefficients in 16 bit fractional form:-

$$D = 0.3897 = H'31E2'$$

$$-E = 0.2206 = H'1C3D'$$

The program written to implement this filter was exactly the same as the 1st order Butterworth program but the coefficient table was modified as follows :-

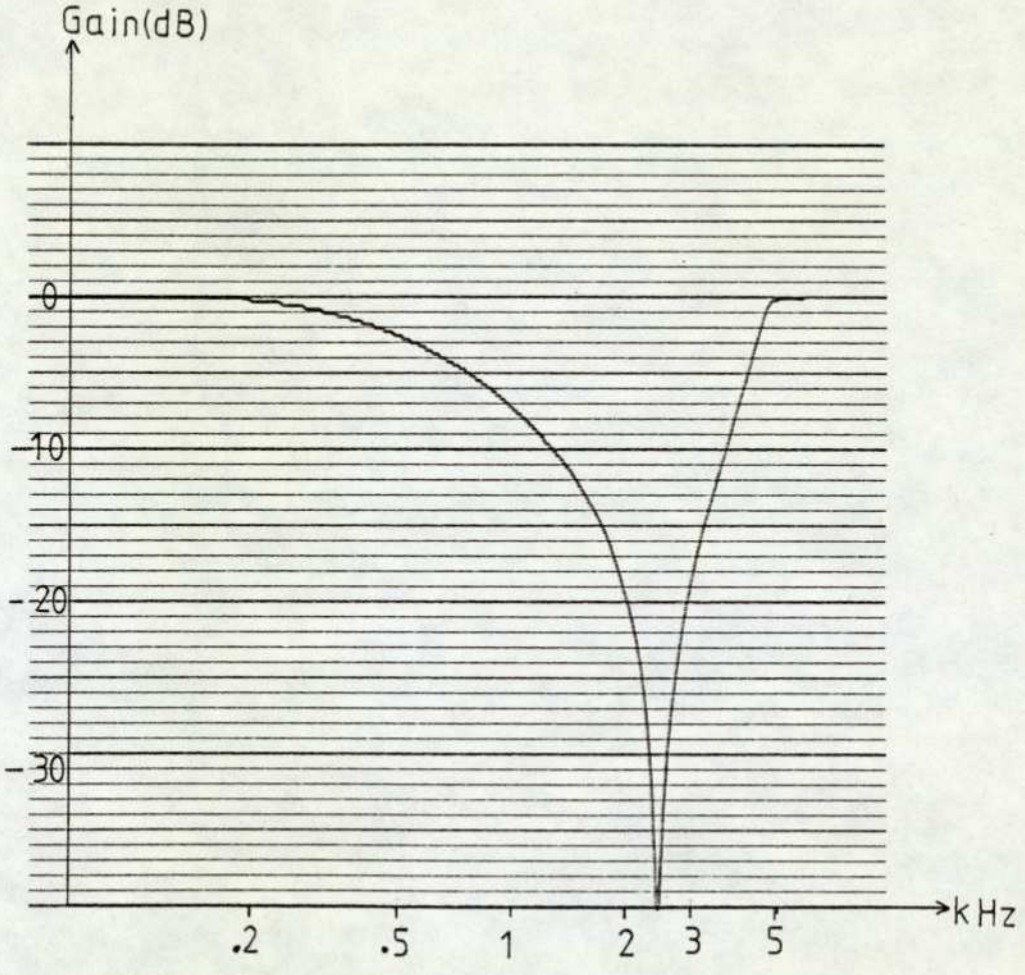


Figure 18.8 First Order Butterworth Digital Filter
(f_s = 5 kHz, f_d = 500 Hz)

Address	Data	
0550	31E2	Coefficient D
0551	1C3D	Coefficient -E

The frequency response of this filter is shown in Figure 18.9

18.5.3 Second Order Butterworth Filter

This kind of filter was designed for a sampling frequency of f_s = 5 KHz and a cutoff frequency of f_d = 500 Hz. The transfer function of the filter was found to be :-

$$H(z) = \frac{0.06745(1 + 2z^{-1} + z^{-2})}{1 - 1.14298z^{-1} + 0.412801z^{-2}}$$

Representing the coefficients in 16 bit fractional form, we have (note that only the fractional part considered):-

K = 0.06745 = H'08A2'

-B = 0.14298 = H'124D'

-C = -0.412801 = H'CB29'

The program written to implement this filter was as follows:-

```
*
* INITIALIZATION ROUTINE
*
```

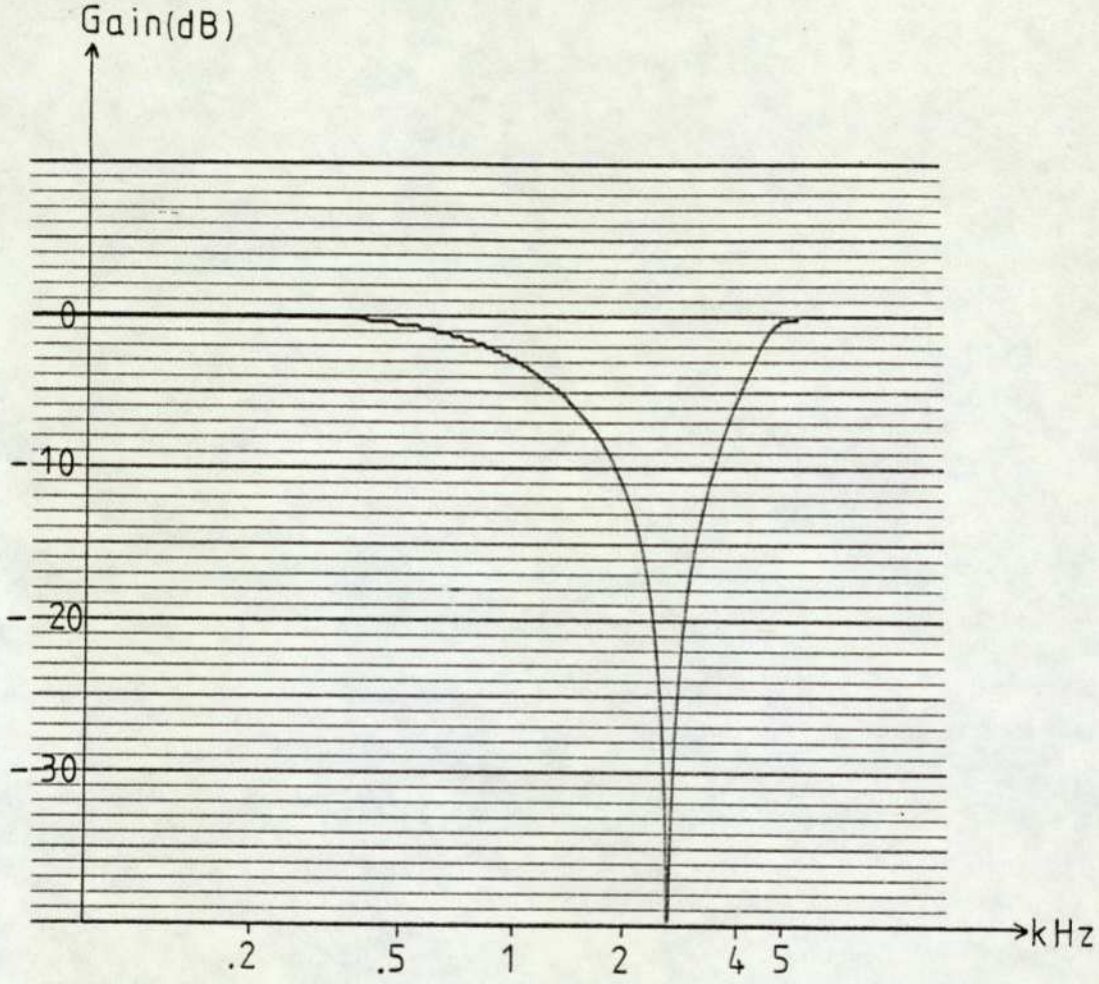


Figure 18.9 First Order Chebyshev I Digital Filter
(f_s = 5kHz, f_r = 500 Hz)

```
0600   3600   SM H'FE00'      Mask all interrupts except 0
0601   FE00
0602   15E0   LI E, H'0550'  Set DCE to start of coefficient
0603   0550                   table
0604   15F0   LI F, H'0500'  Set DCF to start of state variable
0605   0500                   table
0606   1AEE   PUSH E,E        Store DCE in Aux. Reg. E
0607   1AFF   PUSH F,F        Store DCF in Aux. Reg. F
*
* DIGITAL FILTER LOOP
*
0608   8000   IN             Input section
0609   0100
060A   8200   LPFG 2         Second Order section
060B   84EF   OUT            Output section
060C   0100
060D   8500
*
* END
*
```

Also the following data was loaded to the memory :-

Address	Data	
0407	0608	Interrupt service address
0550	08A2	Coefficient K
0551	124D	Coefficient -B
0552	CB29	Coefficient -C

The frequency response of this filter is shown in Figure 18.10

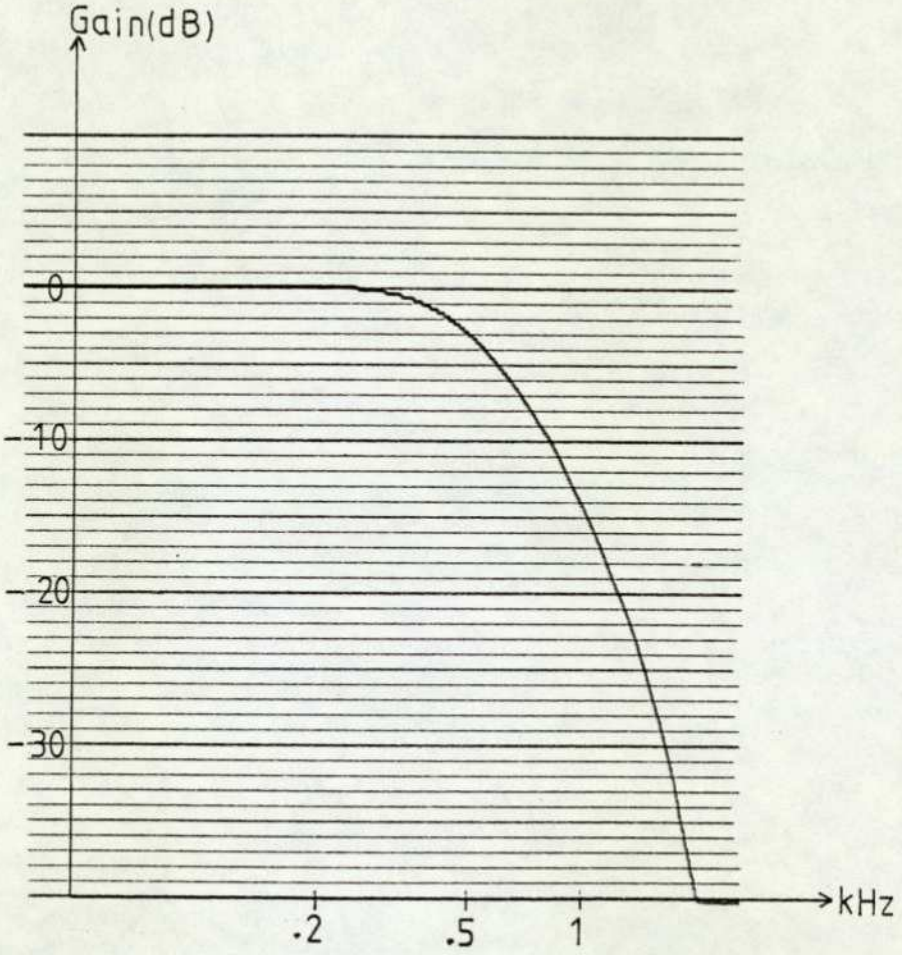


Figure 18.10 Second Order Butterworth Digital Filter
(f_s = 5 kHz, f_d = 500 Hz)

18.5.4 Second Order Chebyshev I Filter

This kind of filter was designed for a sampling frequency of fs= 3.5 KHz and a ripple cutoff frequency of f_r = 500 Hz (i.e. $f_s/f_r = 7$). The passband ripple was chosen to be 0.5 dB. The transfer function of this filter was found to be :-

$$H(z) = \frac{0.17252(1 + 2z^{-1} + z^{-2})}{1 - 0.6362777z^{-1} + 0.3263139z^{-2}}$$

Representing the coefficients in 16 bit fractional form, we have :-

K = 0.17252 = H'1615'

-B = 0.6362277 = H'5170'

-C = -0.3263139 = H'D63B'

The program written to implement this filter was as follows:-

```
*
* INITIALIZATION ROUTINE
*
0600   3600   SM H'FE00'      Mask all interrupts except 0
0601   FE00
0602   15E0   LI E, H'0550'   Set DCE to start of coefficient
0603   0550                   table
0604   15F0   LI F, H'0500'   Set DCF to start of state variable
0605   0500                   table
0606   1AEE   PUSH E,E        Store DCE in Aux. Reg. E
0607   1AFF   PUSH F,F        Store DCF in Aux. Reg. F
```

```
*
* DIGITAL FILTER LOOP
*
0608    8000    IN              Input section
0609    0100
060A    8100    LPFL 2          Second Order section
060B    84EF    OUT             Output Section
060C    0100
060D    8500
*
* END
*
```

Also the following data was loaded to the memory:-

Address	Data	
0407	0608	Interrupt service address
0550	1615	Coefficient K
0551	5170	Coefficient -B
0552	D63B	Coefficient -C

The frequency response of this filter is shown in Figure 18.11

18.5.5 Third Order Butterworth Filter

This filter was designed for a cutoff frequency of f_d = 500 Hz and a sampling frequency of f_s = 3.5 KHz (i.e. f_s/f_d = 7). The transfer function of this filter was found to be :-

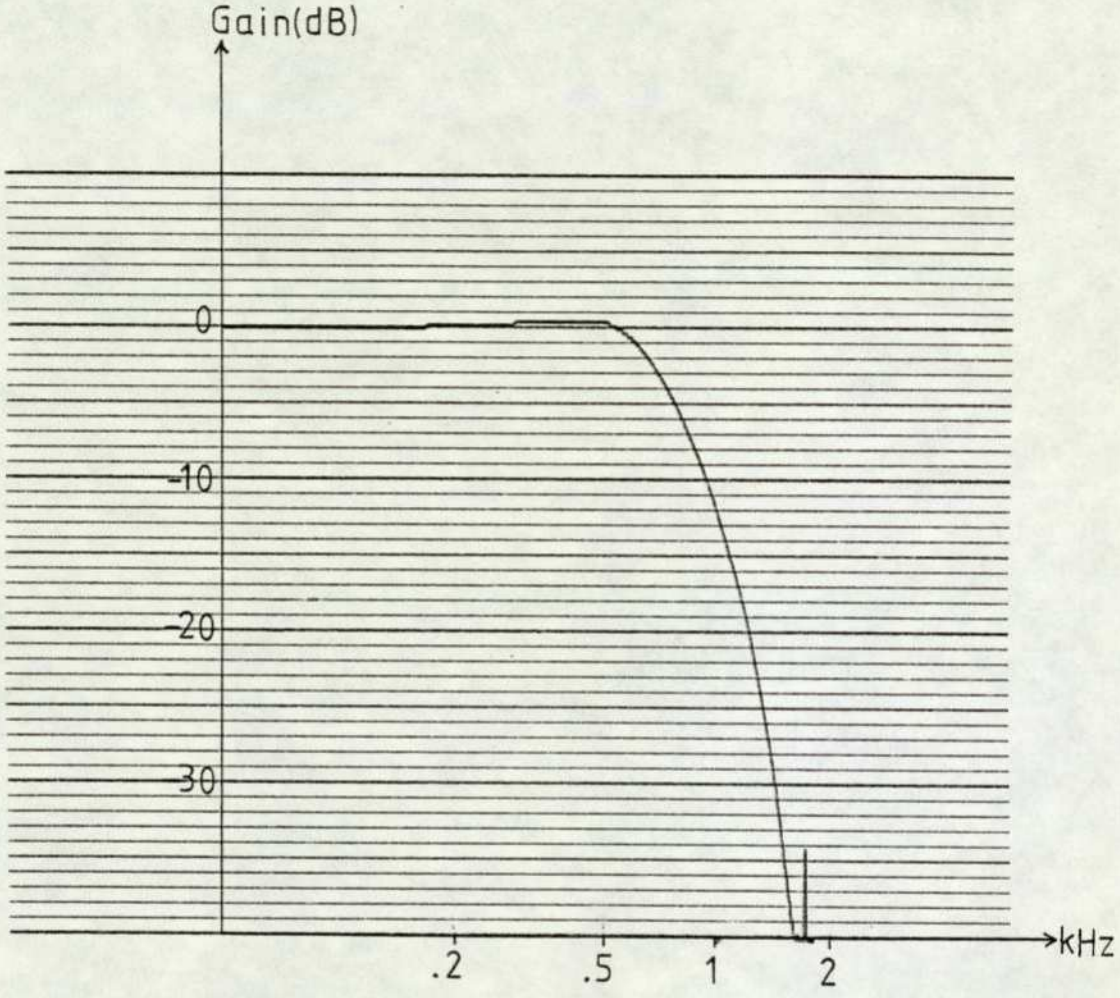


Figure 18.11 Second Order Chebyshev I Digital Filter
(f_s = 3.5 kHz, f_r = 500 Hz)

$$H(z) = \frac{0.1353461(1 + 2z^{-1} + z^{-2})}{1 - 0.896516997z^{-1} + 0.4379016z^{-2}} \cdot \frac{0.3250424(1+z^{-1})}{1 - 0.349915z^{-1}}$$

Representing the coefficients in 16 bit fractional form, we have :-

K = 0.1353461 = H'1153'

-B = 0.896516997 = H'72C1'

-C = -0.4379016 = H'C7F3'

D = 0.3250424 = H'299B'

-E = 0.349915 = H'2CCA'

The program written to implement this filter was as follows:-

```
*
* INITIALIZATION ROUTINE
*
0600   3600   SM H'FE00'        Mask all interrupts except 0
0601   FE00
0602   15E0   LI E, H'0550'     Set DCE to start of coefficient
0603   0550                     table
0604   15F0   LI F, H'0500'     Set DCF to start of state vari-
0605   0500                     able table
0606   1AEE   PUSH E,E          Store DCE in Aux. Reg. E
0607   1AFF   PUSH F,F          Store DCF in Aux. Reg. F
*
* DIGITAL FILTER LOOP
*
0608   8000   IN                Input section
0609   0100
060A   8100   LPFL 2            Second Order Section
060B   8300   LPF 1             First Order Section
060C   84EF   OUT               Output Section
060D   0100
060E   8500
*
* END
*
```

Also the following data was loaded to the memory:-

Address	Data	
0407	0608	Interrupt service address
0550	1153	Coefficient K
0551	72C1	Coefficient -B
0552	C7F3	Coefficient -C
0553	2998	Coefficient D
0554	2CCA	Coefficient -E

The frequency response of this filter is shown in Figure 18.12

18.5.6 Third Order Chebyshev I Filter

This filter was designed for a sampling frequency of f_s = 3.6 KHz and a ripple cutoff frequency of f_r = 600 Hz. The passband ripple was chosen to be 3 dB. The transfer function of this filter was found to be :-

$$H(z) = \frac{0.1926302(1+2z^{-1}+z^{-2})}{1 - 0.9920238z^{-1} + 0.7625445z^{-2}} \cdot \frac{0.14705494(1+z^{-1})}{1 - 0.705890z^{-1}}$$

Representing the coefficients in 16 bit fractional form, we have:-

K = 0.1926302 = H'18A8'
-B = 0.9920238 = H'7EFB'
-C = -0.7625445 = H'9E65'
D = 0.14705494 = H'12D3'
-E = 0.705890 = H'5A5B'

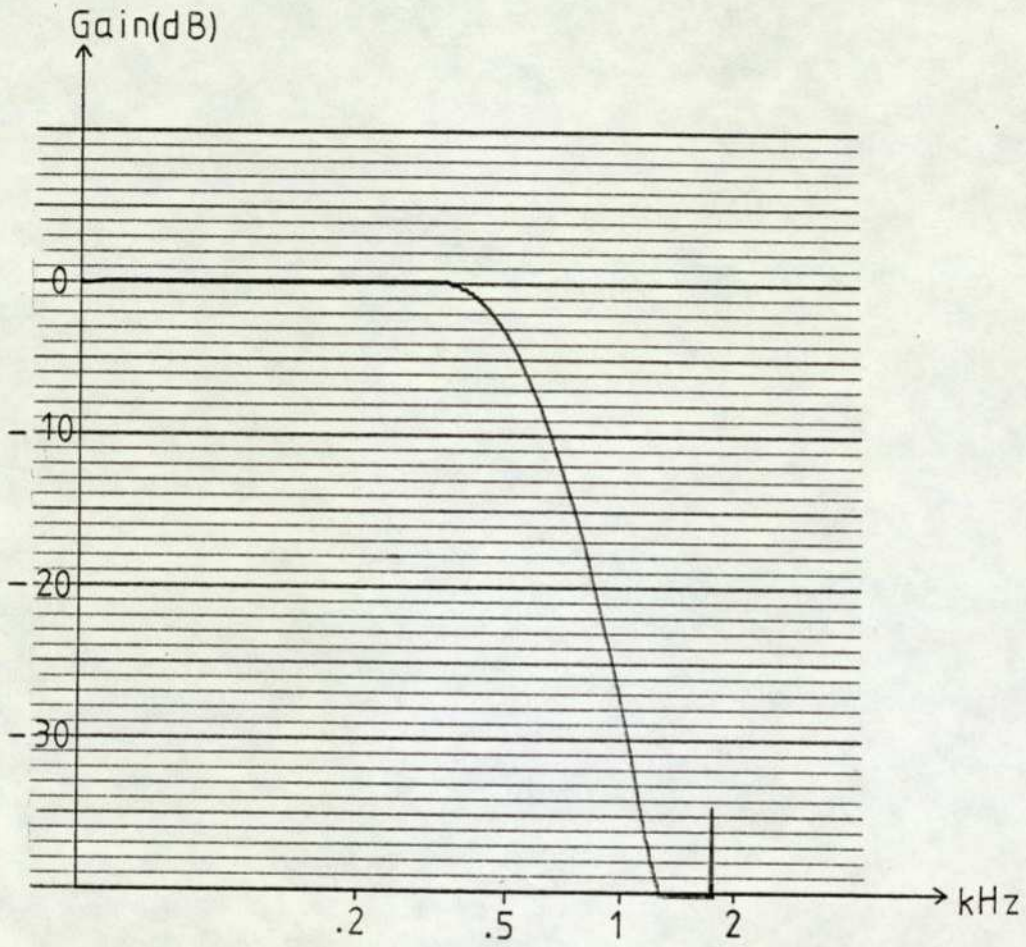


Figure 18.12 Third Order Butterworth Digital Filter
(f_s = 3.5 kHz, f_d = 500 Hz)

The program written to implement this filter was exactly the same as the 3rd order Butterworth program but the coefficient table was modified as follows:-

Address	Data	
0550	18A8	Coefficient K
0551	7EFB	Coefficient -B
0552	9E65	Coefficient -C
0553	12D3	Coefficient D
0554	5A5B	Coefficient -E

The frequency response of this filter is shown in Figure 18.13.

18.5.7 Fourth Order Butterworth Filter

This filter was designed for a sampling frequency of f_s = 3.5 KHz and a cutoff frequency of f_d = 500 Hz (i.e. f_s/f_d =7). The transfer function was found to be :-

$$H(z)= \frac{0.1449015(1+2z^{-1}+z^{-2})}{1 - 0.95981019z^{-1} + 0.5394160z^{-2}} \cdot \frac{0.1093033(1+2z^{-1}+z^{-2})}{1-0.724013z^{-1}+0.16123z^{-2}}$$

Representing the coefficients in 16 bit fractional form, we have:-

K_1 = 0.1449015 = H'128C'
$-B_1$ = 0.95981019 = H'7AD8'
$-C_1$ = -0.5394160 = H'BAF4'
K_2 = 0.1093033 = H'0DFE'

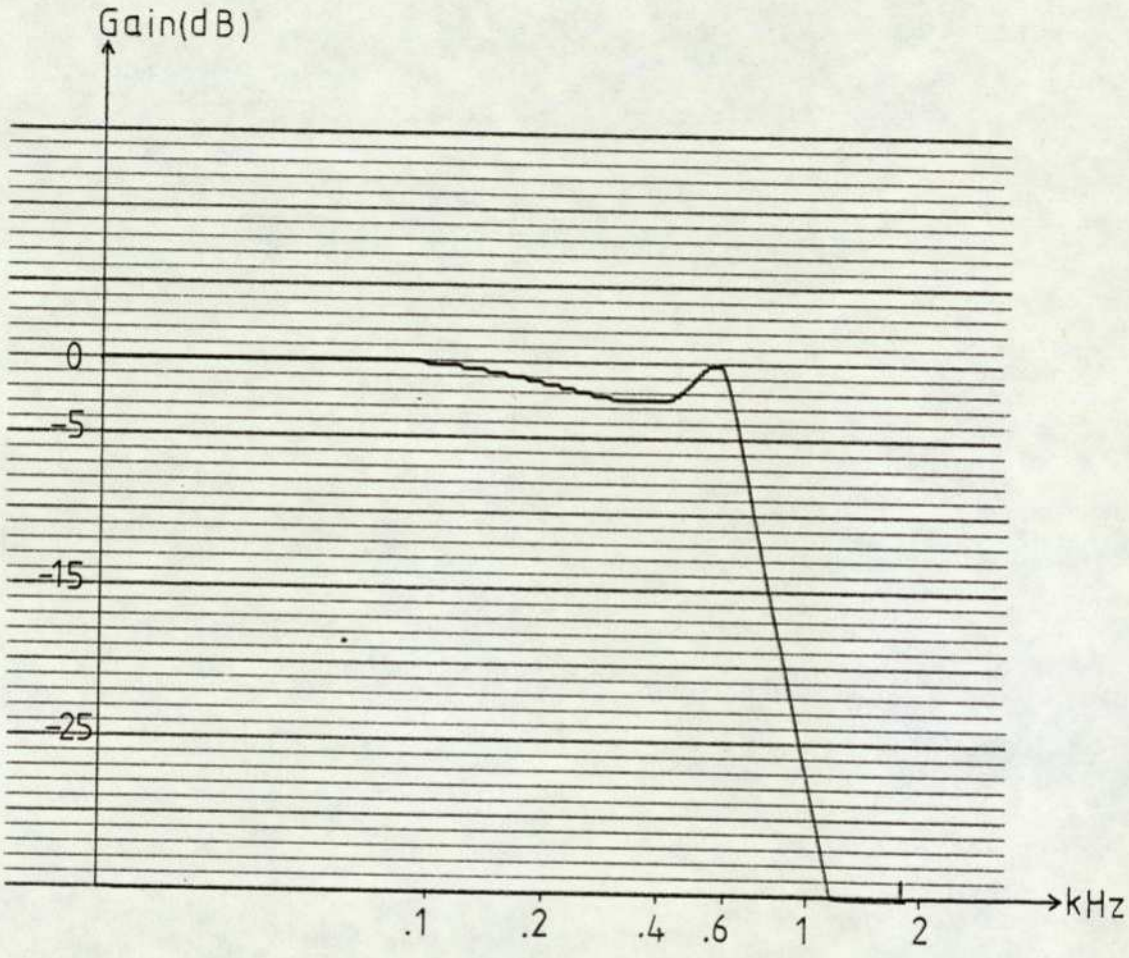


Figure 18.13 Third Order Chebyshev I Digital Filter
(f_s = 3.6 kHz, f_r = 600 Hz, ripple = 3 dB)

$-B_2$ = 0.724013 = H'5CAC'
$-C_2$ = -0.16123 = H'EB5D'

The program written to implement this filter was as follows:-

```
*
* INITIALIZATION ROUTINE
*
0600   3600   SM H'FE00'      Mask all interrupts except 0
0601   FE00
0602   15E0   LI E, H'0550'   Set DCE to start of coefficient
0603   0550                   table
0604   15F0   LI F, H'0500'   Set DCF to start of state variable
0605   0500                   table
0606   1AEE   PUSH E,E        Store DCE in Aux. Reg. E
0607   1AFF   PUSH E,E        Store DCF in Aux. Reg. F
*
* DIGITAL FILTER LOOP
*
0608   8000.  IN              Input section
0609   0100
060A   8100   LPFL 2          Second Order section
060B   8100   LPFL 2          Second Order section
060C   84EF   OUT             Output section
060D   0100
060E   8500
*
* END
*
```

Also the following data was loaded to the memory :-

Address	Data	
0407	0608	Interrupt service address
0550	128C	Coefficient K_1
0551	7ADB	Coefficient $-B_1$
0552	BAF4	Coefficient $-C_1$
0553	0DFE	Coefficient K_2
0554	5CAC	Coefficient $-B_2$
0555	EB5D	Coefficient $-C_2$

The frequency response of this filter is shown in Figure 18.14.

18.5.8 Fourth Order Chebyshev I Filter

This kind of filter was designed for a sampling frequency of f_s = 3.5 KHz and a ripple cutoff frequency of 350 Hz (i.e. $f_s/f_r = 10$). The passband ripple was chosen to be 3 dB. The transfer function of the filter was found to be :-

$$H(z) = \frac{0.08285596(1+2z^{-1}+z^{-2})}{1-1.57237793z^{-1}+0.9038018z^{-2}} \cdot \frac{0.017924268(1+2z^{-1}+z^{-2})}{1-1.696789z^{-1}+0.768486z^{-2}}$$

Representing the coefficients in 16 bit fractional form :-

K_1 = 0.08285596 = H'0A9B'
$-B_1$ = 0.57237793 = H'4944'
$-C_1$ =-0.9038018 = H'8C50'
K_2 = 0.017924268 = H'024B'
$-B_2$ = 0.696789 = H'5930'
$-C_2$ =-0.768486 = H'9DA2'

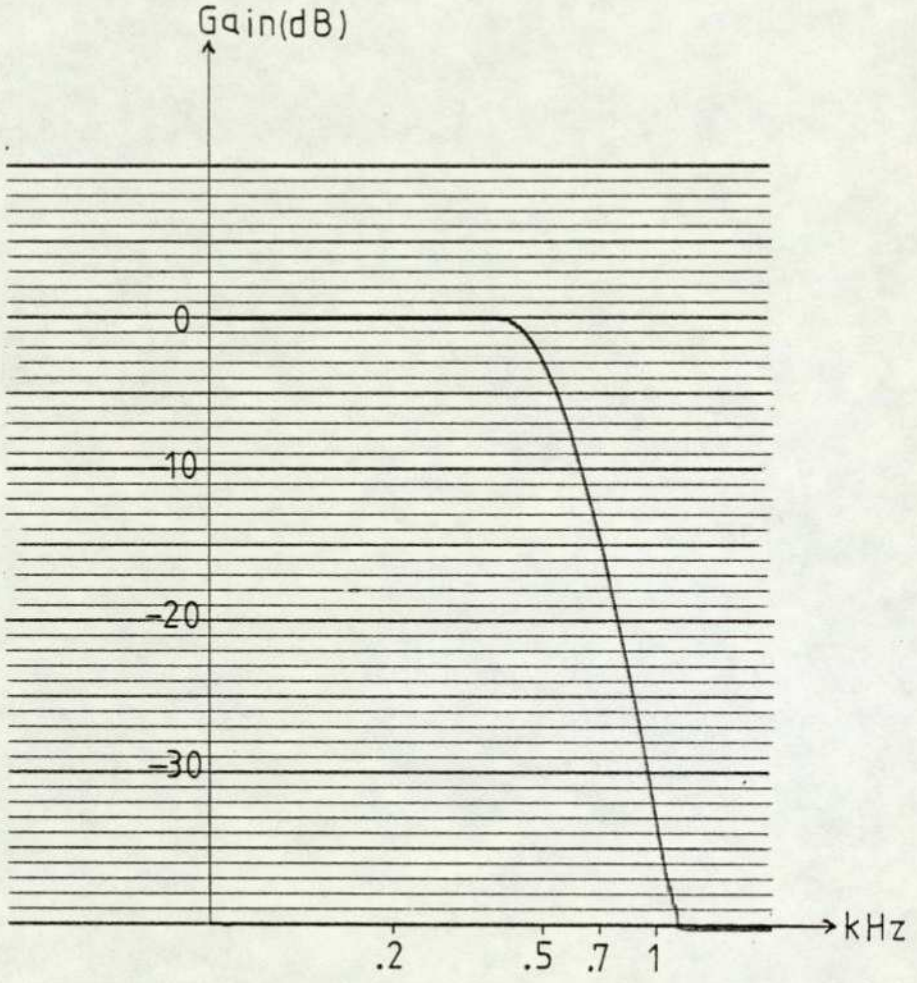


Figure 18.14 Fourth Order Butterworth Digital Filter
(f_s = 3.5 kHz, f_d = 500 Hz)

The program written to implement this filter was as follows:-

```
*
* INITIALIZATION ROUTINE
*
0600   3600   SM H'FE00'       Mask all interrupts except 0
0601   FE00
0602   15E0   LI E, H'0550'    Set DCE to start of coefficient
0603   0550                    table
0604   15F0   LI F, H'0500'    Set DCF to start of state variable
0605   0500                    table
0606   1AEE   PUSH E,E         Store DCE in Aux. Reg. E
0607   1AFF   PUSH F,F         Store DCF in Aux. Reg. F
*
* DIGITAL FILTER LOOP
*
0608   8000   IN               Input section
0609   0100
060A   8200   LPFG 2           Second Order section
0608   8200   LPFG 2           Second Order section
060C   84EF   OUT              Output section
060D   0100
060E   8500
*
* END
*
```

Also the following data was loaded to the memory :-

Address	Data	
0407	0608	Interrupt service address
0550	0A98	Coefficient K_1
0551	4944	Coefficient $-B_1$
0552	8C50	Coefficient $-C_1$
0553	0248	Coefficient K_2
0554	5930	Coefficient $-B_2$
0555	9DA2	Coefficient $-C_2$

The frequency response of this filter is shown in Figure 18.15 .

18.5.9 Sixth Order Butterworth Filter

This filter was designed for a sampling frequency of f_s = 2 kHz and a cutoff frequency of 400 Hz. The transfer function of the filter was found to be:-

$$H(z) = \frac{0.27724678(1+2z^{-1}+z^{-2})}{1-0.4959542z^{-1}+0.60494124z^{-2}} \cdot \frac{0.180070(1+2z^{-1}+z^{-2})}{1-0.322119z^{-1}+.042399z^{-2}}$$

$$\times \frac{0.20657208(1+2z^{-1}+z^{-2})}{1-0.3695273z^{-1}+0.195816z^{-2}}$$

Representing the coefficients in 16 bit fractional form, we have :-

K_1 = 0.27724678 = H'237D'
$-B_1$ = 0.4959542 = H'3F7B'
$-C_1$ =-0.60494124 = H'B291'
K_2 = 0.180070 = H'170D'
$-B_2$ = 0.322119 = H'2938'
$-C_2$ =-0.0423995 = H'FA93'
K_3 = 0.20657208 = H'1A71'
$-B_3$ = 0.3695273 = H'2F4D'
$-C_3$ =-0.195816 = H'E6F0'

The program written to implement this filter was as follows (only the digital filter loop is shown) :-

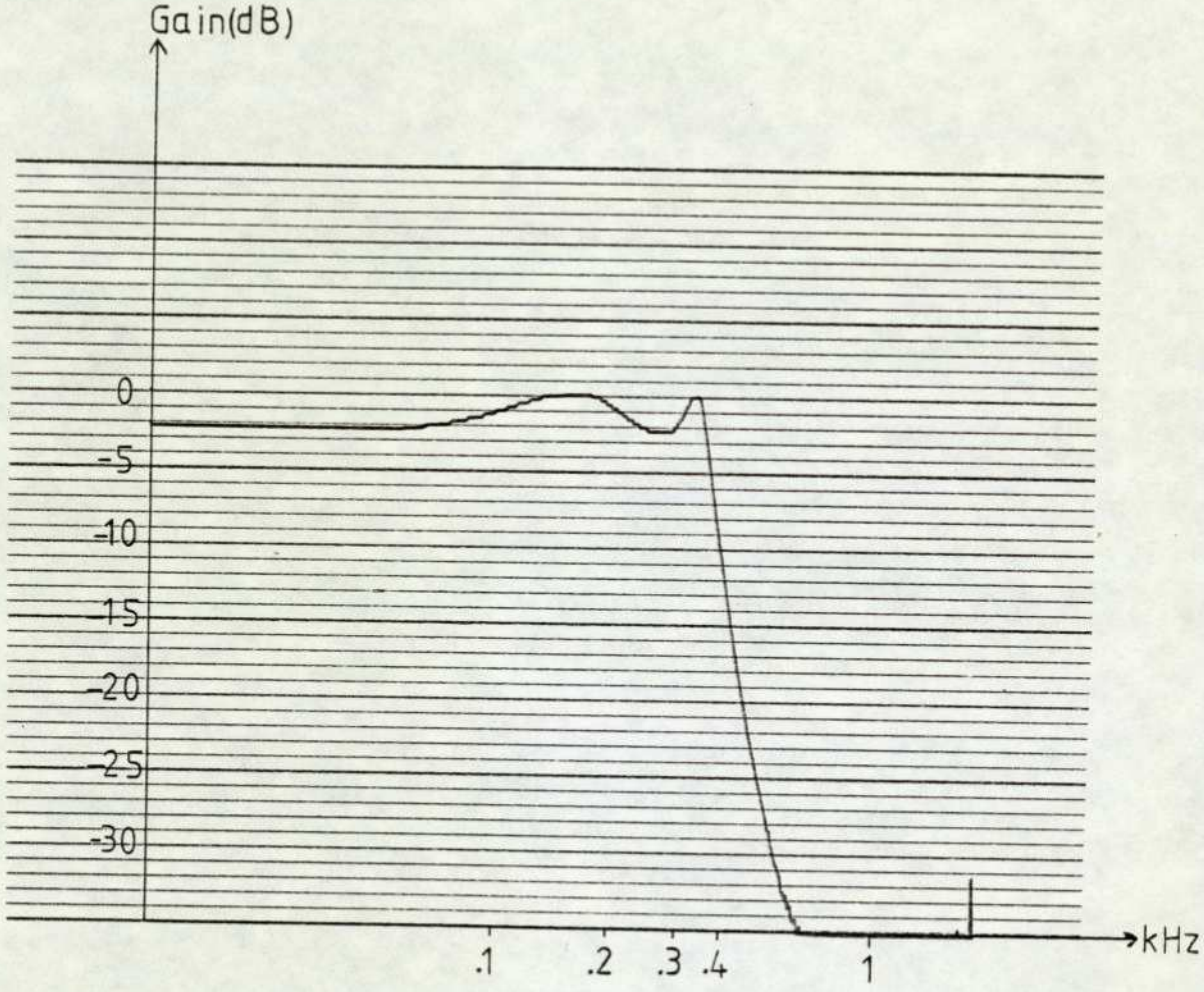


Figure 18.15 Fourth Order Chebyshev I Digital Filter
(f_s = 3.5 kHz, f_r = 350 Hz, ripple = 3 dB)

```
*
* DIGITAL FILTER LOOP
*
0608   8000   IN          Input section
0609   0100
060A   8100   LPFL 2      Second Order section
060B   8100   LPFL 2      Second Order section
060C   8100   LPFL 2      Second Order section
060D   84EF   OUT         Output section
060E   0100
060F   8500
*
* END
*
```

Also the following data was stored in the memory :-

Address	Data	
0407	0608	Interrupt service address
0550	237D	Coefficient K_1
0551	3F7B	Coefficient $-B_1$
0552	B291	Coefficient $-C_1$
0553	170D	Coefficient K_2
0554	2938	Coefficient $-B_2$
0555	FA93	Coefficient $-C_2$
0556	1A71	Coefficient K_3
0557	2F4D	Coefficient $-B_3$
0558	E6F0	Coefficient $-C_3$

The frequency response of this filter is shown in Figure 18.16 .

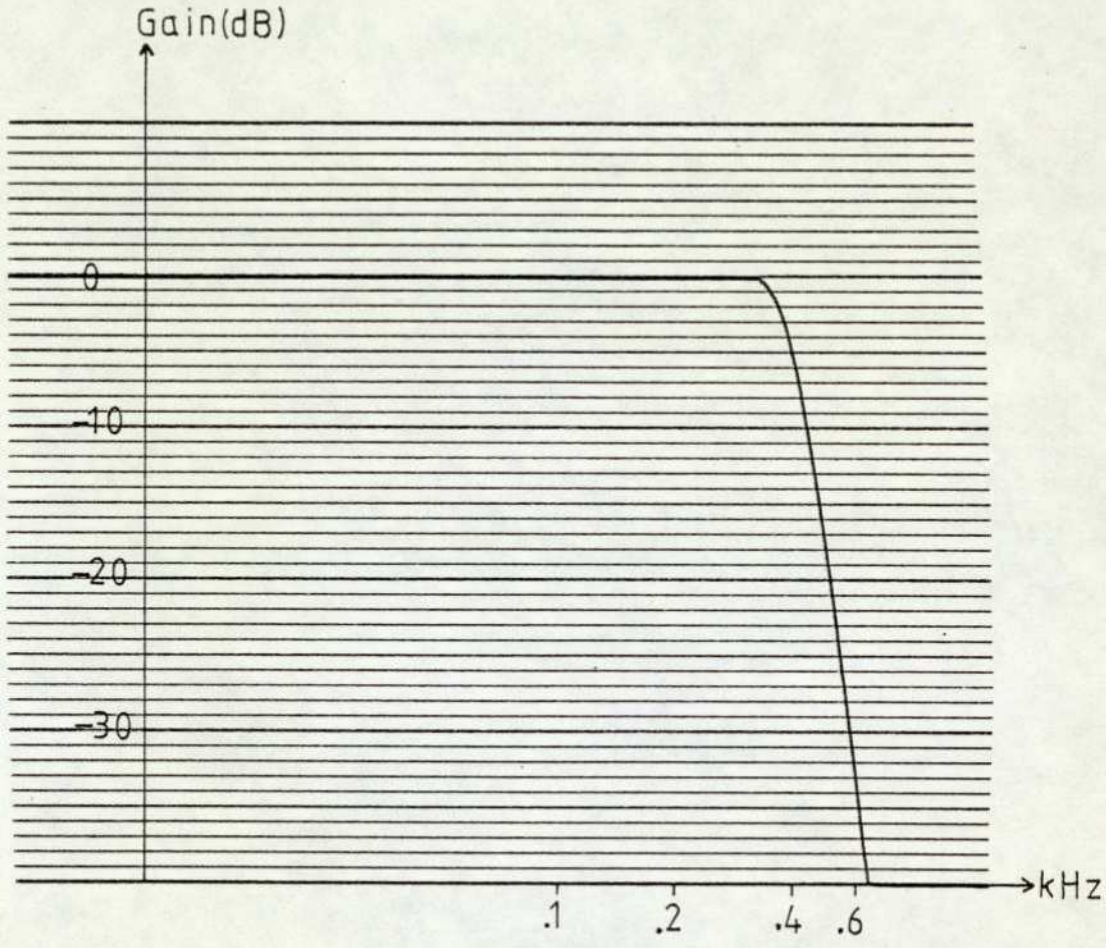


Figure 18.16 Sixth Order Butterworth Digital Filter
(f_s = 2 kHz, f_d = 400 Hz)

18.5.10 Eighth Order Butterworth Filter

This filter was designed for a sampling frequency of f_s = 2 kHz and a cutoff frequency of f_d = 400 Hz. The transfer function of the filter was found to be :-

$$H(z) = \frac{0.2914207(1+2z^{-1}+z^{-2})}{1-0.5213092z^{-1}+0.6869922z^{-2}} \cdot \frac{0.1929285(1+2z^{-1}+z^{-2})}{1-0.345121z^{-1}+0.1168351z^{-2}} X$$

$$X \frac{0.17875345(1+2z^{-1}+z^{-2})}{1-0.31976390z^{-1}+0.0347777z^{-2}} \cdot \frac{0.22605098(1+2z^{-1}+z^{-2})}{1-0.4043723z^{-1}+.308576z^{-2}}$$

Representing the coefficients in 16 bit fractional form,

K_1 = 0.2914207 = H'254D'
$-B_1$ = 0.5213092 = H'42BA'
$-C_1$ =-0.6869922 = H'A811'
K_2 = 0.1929285 = H'18B2'
$-B_2$ = 0.3451210 = H'2C2D'
$-C_2$ =-0.1168351 = H'F10C'
K_3 = 0.17875345 = H'16E1'
$-B_3$ = 0.31976390 = H'28EE'
$-C_3$ =-0.0347777 = H'F88C'
K_4 = 0.22605098 = H'1CF0'
$-B_4$ = 0.4043723 = H'33C2'
$-C_4$ =-0.3085762 = H'D881'

The program written to implement this filter was as follows (only the digital filter loop is shown):-

```
*
* DIGITAL FILTER LOOP
*
```

```
0608    8000    IN                  Input section
0609    0100
060A    8100    LPFL 2              Second Order section
060B    8100    LPFL 2              Second Order section
060C    8100    LPFL 2              Second Order section
060D    8100    LPFL 2              Second Order section
060E    84EF    OUT                 Output section
060F    0100
0610    8500
*
* END
*
```

The frequency response of this filter is shown in Figure 18.17 .

18.5.11 Design of Higher Order Filters

Any order digital filter can be designed by using the required number of the special filter instructions. It is also possible to use the filter instructions in a DO LOOP and execute the required number of times. As an example, a 20^{th} order filter can be implemented as follows (only the digital filter loop is shown) :-

```
*
* DIGITAL FILTER LOOP
*
```

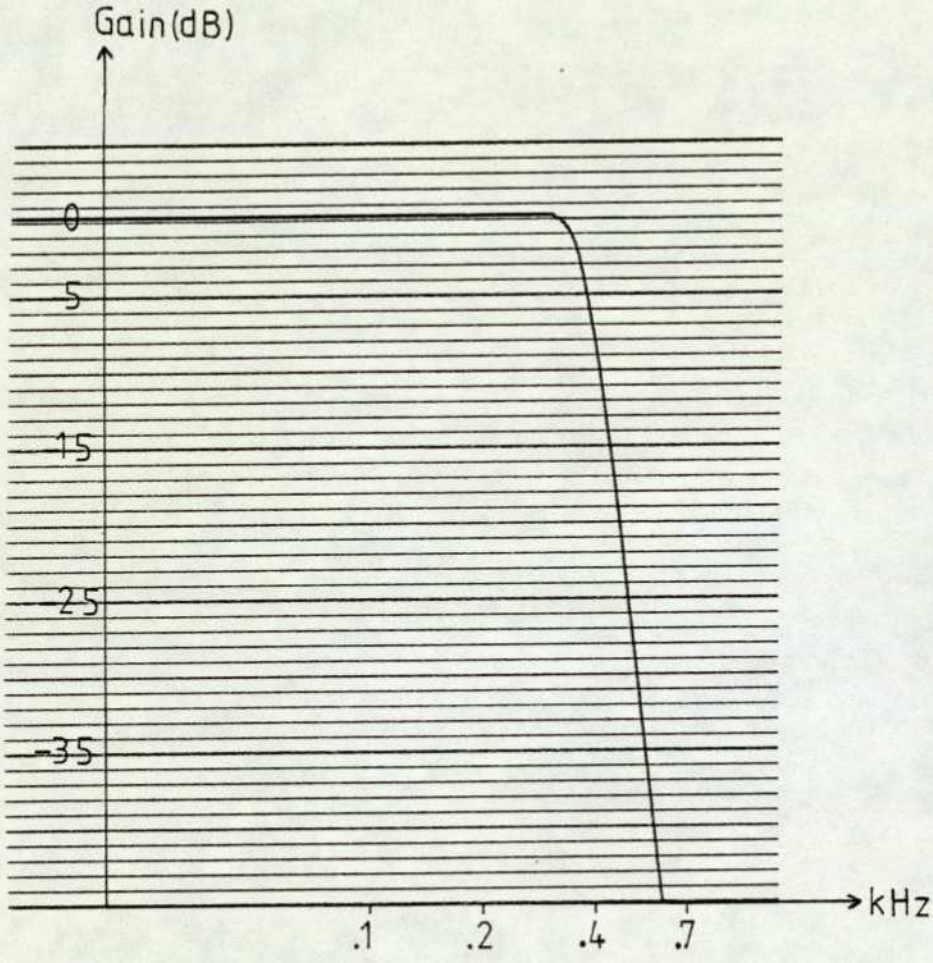


Figure 18.17 Eighth Order Butterworth Digital Filter
(f_s = 2 kHz, f_d = 400 Hz)

```
0608    1580    LI 8, H'000A'       Load n/2 to Register 8
0609    000A
060A    8000    IN                  Input section
060B    0100
060C    8100 W  LPFL 2              Second Order section
060D    0F80    DECR 8              Decrement Register 8
060E    2700    JNZ W               Jump on non-zero to W
060F    060C                        (Execute 10 times)
0610    84EF    OUT                 Output section
0611    0100
0612    8500
*
* END
*
```

18.6 Maximum Sampling frequency for a Given Order

The maximum sampling frequency that can be used is calculated from the time taken to execute the digital filter loop. The input section (instruction IN) is executed in 4 μs. Similarly the output section (instruction OUT) is executed in 7 μs. If the order (n) of the filter is even and the coefficients B_i are less than 1 (i.e. using the instruction LPFL 2) then the second order sections are executed in 100X(n/2) μs. If the coefficients B_i are greater than 1 (i.e. using the instruction LPFG 2) then the second order sections are executed in 102X(n/2) μs. When the order of the filter is odd, additional 62 μs is required to execute the instruction LPF 1. Also, the interrupt service routine (in microprogram PROM) is executed in 4 μs. Therefore, the maximum sampling frequency can be

calculated as follows :-

$$f_{s\ max.} = \frac{1000}{100m + 15 + 62k}\ kHz. \qquad \text{for } B_i < 1$$

and,

$$f_{s\ max.} = \frac{1000}{102m + 15 + 62k}\ kHz. \qquad \text{for } B_i > 1$$

where,

$$m = \frac{n-1}{2} \quad \text{for n odd,} \qquad m = \frac{n}{2} \quad \text{for n even}$$

$$k = 1 \quad \text{for n odd} \quad , \qquad k = 0 \quad \text{for n even}$$

or,

$$f_{s\ max.} = \frac{1000}{50i + 15 + 62k}\ kHz. \qquad \text{for } B_j < 1$$

$$f_{s\ max.} = \frac{1000}{51i + 15 + 62k} \qquad \text{for } B_j > 1$$

where,

$$i = n-1 \quad \text{for n odd} \quad , \qquad i = n \quad \text{for n even}$$

$$k = 1 \quad \text{for n odd} \quad , \qquad k = 0 \quad \text{for n even}$$

Table 18.1 gives the values of $f_{s\ max.}$ for filters up to the 10^{th} order.

Order	Maximum sampling frequency		No. of instructions required
	$B_i < 1$	$B_i > 1$	
1	12.9870	12.9870	3
2	8.6956	8.5470	3
3	5.6497	5.5866	4
4	4.6511	4.5662	4
5	3.6101	3.5587	5
6	3.1746	3.1153	5
7	2.6525	2.6109	6
8	2.4096	2.3640	6
9	2.0964	2.0618	7
10	1.9417	1.9048	7

Table 18.1 Maximum sampling frequency(in kHz) for a given order (with 1 MHz clock) .

18.7 The Microcode of the Filter Instructions

The five filter instructions were implemented by a total of 54 microinstructions in the microprogram RAM. Before describing the microinstructions, it is necessary to understand how the Am2903 performs a two's complement multiplication.

The steps required for a two's complement multiplication are given below (see Figure 18.18) :-

1. Reset R_0 to zero

2. Store the multiplier in R_2 and in the Q register

3. Store the multiplicand in R_1

4. Execute the two's complement multiplication instruction (see Table 5.4) 15 times with C_n = 0 and RAM address A = H'1' , RAM address B = H'0'

5. Execute the two's complement multiply (correction) instruction (see Table 5.4) with C_n = Z(= zero status) and RAM address A = H'1' , RAM address B = H'0'

6. The 32 bit product is stored in registers R_0 and Q , where R_0 holds the higher 16 bits and the Q register holds the lower 16 bits

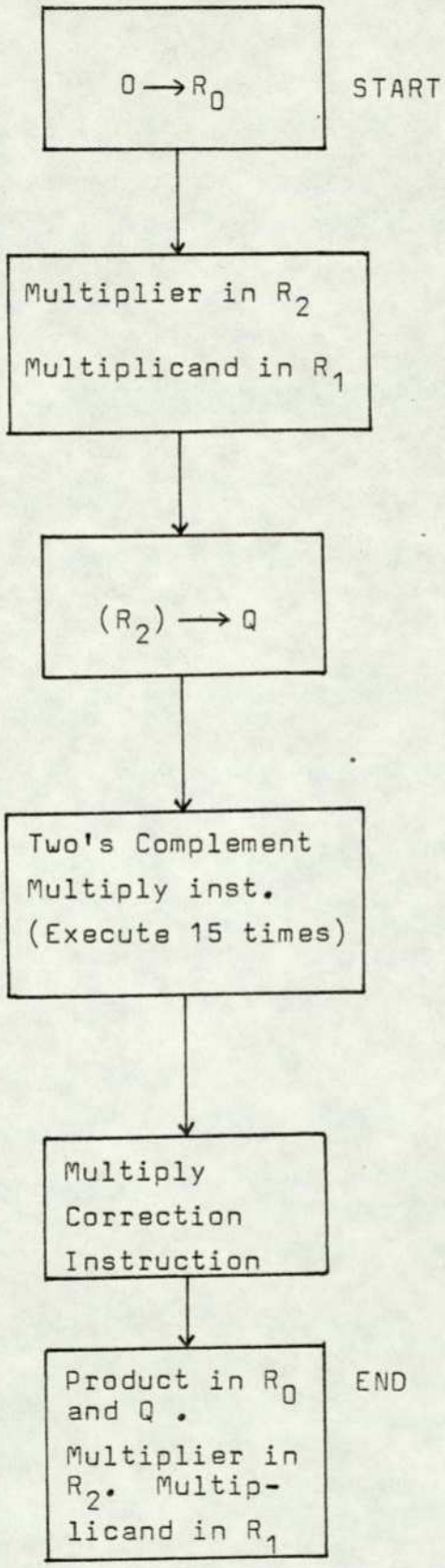


Figure 18.18 Steps in two's complement multiplication using the Am2903s .

7. After the multiplication, the multiplier and the multiplicand are stored in registers R_2 and R_1 respectively.

The microcode of the filter instructions is given in Figure 18.19. Their description is as follows:-

Macroinstruction		Microoperations performed
LPFL 2		$0 \longrightarrow R_6$
	MIC2:	Call to microsubroutine SUB2
		goto MIC1
LPFG 2		H'8000' $\longrightarrow R_6$
		goto MIC2
	MIC1:	$(R_0) \longrightarrow R_3$
		$(R_2) \longrightarrow R_5$
		Call to microsubroutine SUB1
		$(R_0) + (R_3) \longrightarrow Q$
		$(R_4) + (Q) \longrightarrow Q$
		If $(R_6) \neq 0$ goto MIC3
	MIC4:	$(R_2) + (Q) \longrightarrow R_4$
		$(R_4) + (R_5) \longrightarrow R_4$
		$(R_4) + (R_5) \longrightarrow R_4$
		$(DCF) - 1 \longrightarrow DCF$
		$(R_5) \longrightarrow ((DCF))$
		$(DCF) - 1 \longrightarrow DCF$
		$(Q) \longrightarrow ((DCF))$
		$(DCF) + 2 \longrightarrow DCF$
		Fetch the next macroinstruction
	MIC3:	$(R_5) + (Q) \longrightarrow Q$
		goto MIC4
LPF 1		Call to microsubroutine SUB2
		$(R_0) + (R_4) \longrightarrow Q$
		$(R_2) + (Q) \longrightarrow R_4$
		$(DCF) - 1 \longrightarrow DCF$
		$(Q) \longrightarrow ((DCF))$
		$(DCF) + 1 \longrightarrow DCF$
		Fetch the next macroinstruction

	OP CODE	ADDRESS	FIELD 5								FIELD 4								FIELD 3												FIELD 2								FIELD 1								FIELD 0								
			ICU			REL	PCU	CCS			Am 2910				EDB	Inst. Reg.	DSL	RS3					IOM	MMD	P/P	PCU					Branch Address								ICU			ALU				Operand Select	
			A_3	MRQ	WAIT	FE	PCE	ST_1	ST_2	IOR	I_3	I_2	I_1	I_0		IRE	S_4	S_5	C_8	C_4	C_5	C_6	S_3	S_2	S_1	C_3	PE	P_4	P_3	P_2	P_1	BR_{11}	BR_{10}	BR_9	BR_8	BR_7	BR_6	BR_5	BR_4	BR_3	BR_2	BR_1	BR_0	A_2	A_1	A_0	IEN	$\overline{OE}_Y$	EB	RS_1	RS_2
INSTRUCTION			47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SUB1	.	100	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	0	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	0	0	1
	.	101	1	0	1	1	0	1	1	1	1	1	1	0	1	1	X	1	X	X	X	X	X	0	X	1	0	1	1	0	0	0	1	0	0	1	1	1	1	1	1	0	1	1	0	0	1	0	0
DCF+1→DCF	.	102	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	0	X	X	X	X	X	X	X	1	1	1	1	X	X	X	X	0	1	0	0	0	1	0	0	0	1	0	0	1	0	1
SUB3	.	103	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	0	1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	0	0	1
	.	104	1	0	1	1	0	1	1	1	1	1	1	0	1	1	X	1	X	X	X	X	0	X	1	0	1	1	0	0	0	0	1	0	1	0	0	1	1	1	1	0	1	1	0	0	1	0	0
DCE+1→DCE	.	105	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	0	X	X	X	X	X	X	X	1	1	1	0	X	X	X	X	0	1	0	0	0	1	0	0	0	1	0	0	1	0	1
$R_0=0$	.	106	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	X	0	0	0	0	0	0	X	X	0	1	0	0	1	1	0	0	1	1	X	0	0	1	0	1
Ld cntr	.	107	1	1	1	1	1	1	1	1	0	1	0	0	1	1	X	X	X	1	1	1	X	X	X	0	X	X	X	X	0	0	0	0	0	0	0	0	1	1	1	0	X	X	X	1	1	1	X	X
Multiply	.	108	1	1	1	1	1	1	1	1	1	0	0	0	1	1	0	1	1	X	X	X	0	0	1	X	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	X	0	0	0	1	1	0
Correction	.	109	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1	X	X	X	1	1	0	X	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	X	0	0	0	1	1	0
Shift R_0	.	10A	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0	1	0	1	1	1	1	1	1	0	1	0	0	0	1	1	0
RETURN	.	10B	1	1	1	1	1	1	1	1	1	0	1	0	1	1	X	X	X	1	1	1	X	X	X	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
MIC1	.	10C	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	1	X	X	X	0	0	1	X	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	1	0	1
$R_2 \rightarrow R_5$	.	10D	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	1	X	X	X	0	0	1	X	0	1	0	1	0	0	1	0	0	1	0	0	0	1	1	0	0	0	0	0	0	1	0	1
Call SUB1	.	10E	1	1	1	1	1	1	1	1	0	0	0	1	1	1	X	X	X	1	1	1	X	X	X	0	X	X	X	X	0	0	0	1	0	0	0	0	0	0	0	0	X	X	X	1	1	1	X	X
$R_0+R_3 \rightarrow Q$	.	10F	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	X	X	X	0	0	1	X	0	0	1	1	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	1	0	1

Figure 18.19 Microcode of the filter instructions .

cont...

NSTRUCTION	OP CODE	ADDRESS	FIELD 5								FIELD 4								FIELD 3												FIELD 2												FIELD 1								FIELD 0									
			ICU			REL	PCU	CCS			Am 2910				Inst. Req.	DSL		RS3	CCS, Carry-in					IOM	MMD	P/P	PCU					Branch Address											Dest.				Function				ICU Source			ALU			Operand Select	
			A_3	MRQ	WAIT	FE	PCE	ST_1	ST_2	IOR	I_3	I_2	I_1	I_0	EDB	IRE	S_4	S_5	C_8	C_4	C_5	C_6	S_3	S_2	S_1	C_3	PE	P_4	P_3	P_2	P_1	BR_{11}	BR_{10}	BR_9	BR_8	BR_7	BR_6	BR_5	BR_4	BR_3	BR_2	BR_1	BR_0	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	$\overline{E_A}$	$\overline{OE_B}$	$\overline{IEN}$	$\overline{OE_Y}$	EB	RS_1	RS_2
			47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R4+Q→Q	.	1,1,0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	X	X	X	0	0	1	X	X	X	X	X	0	1	0	0	0	1	1	0	0	0	1	1	1	0	X	0	0	1	0	1
JNZ 10C	.	1,1,1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	X	X	X	0	0	1	X	X	X	0	X	X	X	X	0	0	0	1	0	0	0	1	1	1	0	0	X	X	X	1	1	1	X	X
R2+Q→R4	.	1,1,2	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	X	X	X	0	0	1	X	0	1	0	0	0	0	1	0	0	1	0	0	0	0	1	1	1	0	X	0	0	1	0	1
R4+R5→R4	.	1,1,3	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	X	X	X	0	0	1	X	0	1	0	0	0	1	0	1	0	1	0	0	0	0	1	1	0	0	0	0	0	1	0	1
R4+R5→R4	.	1,1,4	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	X	X	X	0	0	1	X	0	1	0	0	0	1	0	1	0	1	0	0	0	0	1	1	0	0	0	0	0	1	0	1
DCF-1→DCF	.	1,1,5	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	X	X	X	0	0	1	X	1	1	1	1	0	0	X	X	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1
	.	1,1,6	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	0	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	0	0	1
	.	1,1,7	1	0	1	1	0	1	1	1	1	1	1	0	0	1	X	1	X	X	X	X	X	1	X	1	0	1	1	0	0	1	0	1	0	1	0	0	1	1	0	0	0	1	0	0	0	1	1	0
DCF-1→DCF	.	1,1,8	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	X	X	X	0	0	1	0	1	1	1	1	0	0	X	X	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1
	.	1,1,9	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	0	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	0	0	1
	.	1,1,A	1	0	1	1	0	1	1	1	1	1	1	0	0	1	X	X	X	X	X	X	1	X	1	0	1	1	1	0	0	0	X	X	0	1	1	0	1	1	1	1	1	1	X	0	0	1	X	X
DCF+2→DCF	.	1,1,B	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	1	1	1	0	0	0	1	1	1	1	1	X	X	X	X	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1
R5+Q→Q	.	1,1,C	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	X	X	X	0	0	1	X	X	X	X	X	0	1	0	1	0	1	1	0	0	0	1	1	1	0	X	0	1	1	0	1
Jump 1/2	.	1,1,D	1	1	1	1	1	1	1	1	0	0	1	1	1	1	X	X	X	1	1	1	X	X	X	0	X	X	X	X	0	0	0	1	0	0	0	1	0	0	1	0	X	X	X	1	1	1	X	X
LPFL 2	81	1,1,E	1	1	1	1	1	0	0	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	X	0	1	1	0	0	0	X	X	0	1	0	0	1	1	0	0	0	1	0	0	0	1	0	1
Call SUB2	.	1,1,F	1	1	1	1	1	1	1	1	0	0	0	1	1	1	X	X	X	1	1	1	X	X	X	0	X	X	X	X	0	0	0	1	0	0	1	0	0	1	0	0	X	X	X	1	1	1	X	X

Figure 18.19 cont.

cont...

INSTRUCTION	OP CODE	ADDRESS	FIELD 5								FIELD 4									FIELD 3									FIELD 2								FIELD 1								FIELD 0								
			ICU			REL	PCU	CCS			Am 2910					Inst. Reg.	DSL			CCS, Carry-in									PCU					Branch Address									ICU			ALU				Operand Select	
			A_3	MRQ	WAIT	FE	PCE	ST_1	ST_2	IOR	I_3	I_2	I_1	I_0	EDB	IRE	S_4	S_5	RS_3	C_8	C_4	C_5	C_6	S_3	S_2	S_1	C_3	IOM	MMD	P/P	PE	P_4	P_3	P_2	P_1	BR_{11}	BR_{10}	BR_9	BR_8	BR_7	BR_6	BR_5	BR_4	BR_3	BR_2	BR_1	BR_0	A_2	A_1	A_0	IEN	$\overline{OE}_Y$	EB	RS_1	RS_2
			CR_{23}	CR_{22}	CR_{21}	CR_{20}	CR_{13}, E_4	CR_{12}, Pol	CR_{11}	CR_{10}	Dest.				Function				Source			$\overline{OE}_B$																															
			47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Jump MICI	.	1,20	1	1	1	1	1	1	1	1	0	0	1	1	1	1	X	X	X	1	1	1	X	X	X	0	X	X	X	X	0	0	0	1	0	0	0	0	1	1	0	0	X	X	X	1	1	1	X	X
LPFG 2	82	1,21	1	1	1	1	1	0	0	1	0	0	1	1	1	1	0	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	1	1	1	1	1	0	1	0	0	0	1	0	1
IN	80	1,22	1	0	1	1	0	1	1	1	1	1	1	0	1	1	X	X	X	X	X	X	0	X	X	X	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
	.	123	1	1	1	1	1	1	1	0	1	1	0	1	1	1	X	0	X	1	1	1	1	X	X	1	0	1	0	0	0	0	X	X	0	1	0	0	1	1	1	1	0	1	1	0	0	1	0	1
SUB2	.	1,24	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	1	X	X	X	0	0	1	X	0	0	1	0	0	1	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	1	0	1
Call SUB3	.	1,25	1	1	1	1	1	1	1	1	0	0	0	1	1	1	X	X	X	1	1	1	X	X	X	0	X	X	X	X	0	0	0	1	0	0	0	0	0	0	1	1	X	X	X	1	1	1	X	X
$R_0 \rightarrow R_4$	.	1,26	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	1	X	X	X	0	0	1	X	0	1	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	1	0	1
Call SUB1	.	127	1	1	1	1	1	1	1	1	0	0	0	1	1	1	X	X	X	1	1	1	X	X	X	0	X	X	X	X	0	0	0	1	0	0	0	0	0	0	0	0	X	X	X	1	1	1	X	X
RETURN	.	128	1	1	1	1	1	1	1	1	1	0	1	0	1	1	X	X	X	1	1	1	X	X	X	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
LPF 1	83	129	1	1	1	1	1	1	1	1	0	0	0	1	1	1	X	X	X	1	1	1	X	X	X	0	X	X	X	X	0	0	0	1	0	0	1	0	0	1	0	0	X	X	X	1	1	1	X	X
$R_0 + R_4 \rightarrow Q$	.	12A	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	X	X	X	0	0	1	X	0	1	0	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	1	0	1
$R_3 + Q \rightarrow R_4$	.	12B	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	X	X	X	0	0	1	X	0	1	0	0	0	0	0	1	0	0	1	0	0	0	1	1	1	0	X	0	0	1	0	1
DCF-1 $\rightarrow$ DCF	.	12C	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	X	X	X	0	0	1	X	1	1	1	1	0	0	X	X	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1
	.	12D	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	0	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	0	0	1
	.	12E	1	0	1	1	0	1	1	1	1	1	1	0	0	1	X	X	X	X	X	X	X	1	X	1	0	1	1	0	0	0	X	X	0	1	1	0	1	1	1	1	1	1	X	0	0	1	X	X
	.	12F	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	1	1	1	0	0	0	1	1	1	1	1	X	X	X	X	0	1	0	0	0	1	0	0	0	0	0	0	1	0	1

Figure 18.19 cont.

cont...

INSTRUCTION	OP CODE	ADDRESS	FIELD 5								FIELD 4								FIELD 3								FIELD 2								FIELD 1								FIELD 0							
			ICU			REL	PCU	CCS			Am 2910					Inst. Req.	DSL			IOM	MMD	P/P	PCU				Branch Address								Dest.				Function				ICU			ALU			Operand Select	
			A_3	MRQ	WAIT	FE	PCE	ST_1	ST_2	IOR	I_3	I_2	I_1	I_0	EDB	IRE	S_4	S_5	RS_3	C_8	C_4	C_5	C_6	S_3	S_2	S_1	C_3	PE	P_4	P_3	P_2	P_1	BR_{11}	BR_{10}	BR_9	BR_8	BR_7	BR_6	BR_5	BR_4	BR_3	BR_2	BR_1	BR_0	A_2	A_1	A_0	IEN	$\overline{OE_Y}$	EB	RS_1	RS_2
			CCS, Carry-in																								CR_{23}	CR_{22}	CR_{21}	CR_{20}	CR_{13}, E_4	CR_{12}, Pol	CR_{11}	CR_{10}	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	E_A	$\overline{OE_B}$							
			47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OUT	84	130	1	0	1	1	0	1	1	1	1	1	1	0	1	1	X	X	X	X	X	X	X	0	X	X	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
	.	131	1	1	1	1	1	1	1	0	1	1	1	0	0	1	X	0	X	X	X	X	0	X	X	X	0	1	0	0	0	X	X	X	0	1	0	0	1	1	1	1	0	1	0	0	0	1	0	1
Pop DCE	.	132	1	1	1	0	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	1	X	X	X	X	X	X	0	0	X	X	0	1	0	0	1	1	1	1	0	1	1	0	0	1	1	1
Pop DCF	.	133	1	1	1	0	1	1	1	1	1	1	1	0	1	1	X	1	X	X	X	X	X	1	X	1	1	1	1	0	0	X	X	0	1	0	0	1	1	1	1	0	1	1	0	0	1	1	1
EI	.	134	0	1	1	1	1	1	1	1	1	1	0	1	1	1	X	X	X	1	1	1	X	X	X	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	1	1	1	X	X
Wait for Int.	85	135	1	1	1	1	1	1	1	1	0	0	1	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
	.	.																																															
	.	.																																															
	.	.																																															
	.	.																																															
	.	.																																															
	.	.																																															
	.	.																																															
	.	.																																															
	.	.																																															
	.	.																																															

Figure 18.19 cont.

Macroinstruction	Microoperations performed
IN	Main memory READ request
	Port Address $\longrightarrow$ Port Bus
	I/O Input request
	(Data Bus) $\longrightarrow R_4$
	Fetch the next macroinstruction
OUT	Main memory READ request
	Port Address $\longrightarrow$ Port Bus
	I/O Output request
	$(R_4) \longrightarrow$ Data Bus
	Enable the REL of the APU
	Aux. Reg. E $\longrightarrow$ DCE
	Enable the REL of the APU
	Aux. Reg. F $\longrightarrow$ DCF
	Enable interrupts
	Fetch next macroinstruction
MIC5:	If interrupt not detected goto MIC5 else goto interrupt service routine

Microsubroutines :

SUB2: $(R_4) \longrightarrow R_2, Q$
Call to microsubroutine SUB3
$(R_0) \longrightarrow R_4$
Call to microsubroutine SUB1
Return

SUB1: $((DCF)) \longrightarrow R_2, Q$
$(DCF) + 1 \longrightarrow DCF$

SUB3: $((DCE)) \longrightarrow R_1$
$(DCE) + 1 \longrightarrow DCE$
$(R_1) \times (R_2) \longrightarrow R_0, Q$
Two's complement correction
Shift R_0 left by one position for fractional* multiplication (the MSB of Q shifted to LSB of R_0)

* see Chapter 17.9

Truncate the product (hence product in R_0)
Return

The hexadecimal representation of the microcode(for loading the Support Processor) is given in Figure 18.20.

18.8 Conclusions

Chapter 18 has described a novel technique for the design of real-time Low Pass IIR digital filters using the bit-slice microcomputer.

Five special instructions were invented which greatly simplified the design of digital filters. It is shown that only one instruction is required to cascade the second order filter sections. This minimizes the errors in developing software. e.g. only 4 instructions (7 memory locations) are required to design a 4^{th} order digital filter loop compared to more than 350 instructions (about 400 memory locations) for the same design on an F8 microcomputer using software multiplication with double precision [29]. Similarly the design of a 6^{th} order filter requires only 5 instructions (8 memory locations) compared to more than 400 instructions (about 750 memory locations) for the same design on an F8 microcomputer [29].

Another advantage of designing digital filters with

the bit-slice processors is the increase in bandwidth. e.g. the maximum sampling frequency for a 2^{nd} order filter on the bit-slice microcomputer designed is about 8500 Hz compared to 847 Hz [29] on an F8 microcomputer (using software multiplication with double precision).

A faster response could be obtained if the filter coefficients and the state variables were stored in the internal ALU registers. This is because a memory reference instruction can be executed in a minimum of 2 μs (with 1 MHz clock), while a register to register operation usually takes 1 μs (excluding the FETCH cycle). But this approach has the disadvantage that large number of registers are required to implement high order filters.

It is also important to mention that the clock frequency of the bit-slice microcomputer could be increased by a factor of 5 (200 ns cycle time) by careful layout of the components and by using high speed devices in the critical data paths of the system. Under these conditions the maximum sampling frequency for a second order section would be more than 40 kHz.

```
*
* NMIC
ADDRESS: 100  FF EC 00 F0 00 19
ADDRESS: 101  B7 ED 01 62 7F 64
ADDRESS: 102  FF EC 00 F0 44 45
ADDRESS: 103  FF EC 00 E0 00 19
ADDRESS: 104  B7 ED 01 61 4F 64
ADDRESS: 105  FF EC 00 E0 44 45
ADDRESS: 106  FF EC 00 00 4C C5
ADDRESS: 107  FF 4C 70 00 0E 1C
ADDRESS: 108  FF 8D 82 10 20 06
ADDRESS: 109  FF ED 8C 10 60 06
ADDRESS: 10A  FF ED 00 00 BF 46
ADDRESS: 10B  FF AC 70 00 00 1C
ADDRESS: 10C  FF EC 82 30 46 05
ADDRESS: 10D  FF EC 82 52 46 05
ADDRESS: 10E  FF 1C 70 01 00 1C
ADDRESS: 10F  FF EE 82 30 63 05
ADDRESS: 110  FF EE 82 04 63 85
ADDRESS: 111  FF 3C 10 01 1C 1C
ADDRESS: 112  FF EE 82 42 43 85
ADDRESS: 113  FF EE 82 45 43 05
ADDRESS: 114  FF EE 82 45 43 05
ADDRESS: 115  FF EE 82 F0 41 45
ADDRESS: 116  FF EE 00 F0 00 19
ADDRESS: 117  B7 E5 05 65 4C 46
ADDRESS: 118  FF EE 82 F0 41 45
ADDRESS: 119  FF EE 00 F0 00 19
ADDRESS: 11A  B7 E4 05 60 6F C4
ADDRESS: 11B  FF DE F1 F0 40 05
ADDRESS: 11C  FF EE 82 05 63 8D
ADDRESS: 11D  FF 3C 70 01 12 1C
ADDRESS: 11E  F9 EC 00 60 4C 45
ADDRESS: 11F  FF 1C 70 01 24 1C
ADDRESS: 120  FF 3C 70 01 0C 1C
ADDRESS: 121  F9 3C 70 61 1F 45
ADDRESS: 122  B7 EC 00 10 00 1C
ADDRESS: 123  FE DC 79 40 4F 65
ADDRESS: 124  FF EC 82 24 76 05
ADDRESS: 125  FF 1C 70 01 03 1C
ADDRESS: 126  FF EC 82 40 46 05
ADDRESS: 127  FF 1C 70 01 00 1C
ADDRESS: 128  FF AC 70 00 00 1C
ADDRESS: 129  FF 1C 70 01 24 1C
ADDRESS: 12A  FF EE 82 40 63 05
ADDRESS: 12B  FF EE 82 44 43 85
ADDRESS: 12C  FF EE 82 F0 41 45
ADDRESS: 12D  FF EC 00 F0 00 19
ADDRESS: 12E  B7 E4 05 60 6F C4
ADDRESS: 12F  FF DE F1 F0 44 05
```

Figure 18.20 Hexadecimal representation of the microcode for the filter instructions (for loading the Support Processor) .

cont...

```
ADDRESS: 130  B7 EC 00 10 00 1C
ADDRESS: 131  FE E4 00 40 4F 45
ADDRESS: 132  EF EC 02 00 4F 67
ADDRESS: 133  EF ED 02 F0 4F 67
ADDRESS: 134  7F DC 71 00 00 1C
ADDRESS: 135  FF 2C 00 00 00 1C
ADDRESS: *
* NMAP
ADDRESS: 80  122
ADDRESS: 81  11E
ADDRESS: 82  121
ADDRESS: 83  129
ADDRESS: 84  130
ADDRESS: 85  135
ADDRESS: *
* LMIC
MICROPROGRAM MEMORY LOADED
* LMAP
MAPPING RAM LOADED
*
*
*
```

Figure 18.20 cont.

CHAPTER 19

19. DESIGN OF REAL-TIME HIGH PASS IIR DIGITAL FILTERS USING THE BIT-SLICE MICROCOMPUTER

19.1 Introduction

This Chapter describes the special instructions invented for the design of real-time High Pass IIR digital filters, using the bit-slice microcomputer.

The hardware set-up to implement these filters was same as the set-up for the Low Pass filters (see Chapter 18.2)

19.2 Computing the Filter Algorithm

It is shown in Chapter 17 that an n^{th} order Low Pass Butterworth or Chebyshev I type digital filter has the general transfer function of :-

$$H_1(z) = \frac{K_1(1 + 2z^{-1} + z^{-2})}{1 + B_1z^{-1} + C_1z^{-2}} \cdot \frac{K_2(1 + 2z^{-1} + z^{-2})}{1 + B_2z^{-1} + C_2z^{-2}} \cdots\cdots \frac{D(1 + z^{-1})}{1 + Ez^{-1}}$$

....(19.1)

It is also shown in Chapter 17.8 that a High Pass filter can be obtained from a Low Pass filter by replacing z^{-1} by $-z^{-1}$ in the Low Pass filter transfer function.

Therefore, the transfer function of an n^{th} order High Pass filter can be written as (replace z^{-1} by $-z^{-1}$ in equation 19.1):-

$$H_h(z) = \frac{K_1(1 - 2z^{-1} + z^{-2})}{1 - B_1z^{-1} + C_1z^{-2}} \cdot \frac{K_2(1 - 2z^{-1} + z^{-2})}{1 - B_2z^{-1} + C_2z^{-2}} \cdot \cdots \cdot \frac{D(1 - z^{-1})}{1 - Ez^{-1}}$$

....(19.2)

The cutoff frequency f_h of the High Pass filter $H_h(z)$ is then related to the cutoff frequency f_d of the Low Pass filter $H_l(z)$ by the equation (see Chapter 17.7):-

$$f_h = \frac{f_s}{2} - f_d \qquad (19.3)$$

The steps in designing a High Pass filter can thus be summarized as follows :-

1. Given the cutoff frequency f_h and the sampling frequency f_s: Calculate f_d :-

$$f_d = \frac{f_s}{2} - f_h$$

2. Design a Low Pass filter with the cutoff frequency f_d and the sampling frequency f_s, as described in Chapter 17

3. Change the sign of the odd powers of z^{-1} in the Low Pass filter transfer function. This gives the required High Pass filter transfer function.

Following an analysis similar to those given in Chapter 18.3.1 to 18.3.3, it can be shown that the implementation of the High Pass filter (equation 19.2) requires the following operations to be performed at every sampling interval (x_i, q, and w are general purpose registers) :-

1.	LOOP: Input to 'w' via A/D converter	INPUT SECTION
2.	$K_i w \longrightarrow w$	INTERMEDIATE SECTIONS
3.	$w + B_i x_{2i-1} - C_i x_{2i} \longrightarrow q$	Repeat 2 to 6 for i=1,2,....,m
4.	$q + x_{2i} - 2x_{2i-1} \longrightarrow w$	$m = \frac{n}{2}$ for n even
5.	$x_{2i-1} \longrightarrow x_{2i}$	$m = \frac{n-1}{2}$ for n odd
6.	$q \longrightarrow x_{2i-1}$	
7.	$Dw \longrightarrow w$	FIRST ORDER SECTION
8.	$w + Ex_n \longrightarrow q$	(only for n odd)
9.	$q - x_n \longrightarrow w$	
10.	$q \longrightarrow x_n$	
11.	Output 'w' via D/A converter	OUTPUT SECTION
12.	Enable external interrupts	
13.	WAIT: If interrupt detected goto LOOP else goto WAIT	

19.3 Choosing an Instruction Set

A set of special instructions were invented based upon the implementation technique described in the previous section.

As with the Low Pass filters, the High Pass filter coefficients* were stored in the sequential memory locations as: K_1, B_1, $-C_1$, K_2, B_2, $-C_2$,....., D, E and were addressed by the Data Counter E(DCE). Similarly, the state variables x_i were stored in the sequential memory locations, addressed by the Data Counter F(DCF). Register 4 (R_4) of the ALU was used as the 'w' register and the Q register of the ALU was used as the 'q' register. The instructions IN and OUT (see Chapter 18) were also used in the design of High Pass filters.

The following instructions were invented specially for the High Pass filters (these instructions were implemented in the microprogram RAM and the Mapping RAM) :-

19.3.1 Instruction 'HPFL 2'

The HPFL 2 instruction implements a second order section completely (cascadable) as described in the previous section. It performs the following operations:-

* These are the coefficients in the form given in equation 19.2 .

((DCE)) X (R_4) ⟶ R_4

(R_4) + ((DCE+1)) X ((DCF)) + ((DCE+2)) X ((DCF+1)) ⟶ Q

(Q) + ((DCF+1)) - 2 X ((DCF)) ⟶ R_4

((DCF)) ⟶ ((DCF+1))

(Q) ⟶ ((DCF))

(DCF) + 2 ⟶ DCF

(DCE) + 3 ⟶ DCE

The HPFL 2 instruction occupies only one location in the main memory as:-

8600 HPFL 2

where H'86' is the Op-Code chosen for this instruction. The execution time of this instruction is 100 cycles.

19.3.2 Instruction 'HPFG 2'

This instruction is very similar to the HPFL 2 instruction, except that here the filter coefficients B_i are assumed to be greater than 1 and hence x_{2i-1} is subtracted from the sum when calculating the Q. It performs the following operations :-

((DCE)) X (R_4) ⟶ R_4

(R_4) + ((DCE+1)) X((DCF)) + ((DCE+2)) X ((DCF+1)) - ((DCF)) ⟶ Q

(Q) + ((DCF+1)) - 2 X ((DCF)) ⟶ R_4

((DCF)) ⟶ ((DCF+1))

(Q) ⟶ ((DCF))

(DCF) + 2 ⟶ DCF

(DCE) + 3 ⟶ DCE

The HPFG 2 instruction occupies only one location in the main memory as:-

```
8700    HPFG 2
```

where H'87' is the)p-Code chosen for this instruction. The execution time of this instruction is 102 cycles.

19.3.3 Instruction 'HPF 1'

This instruction implements a first order section. The operations performed are as follows :-

((DCE)) X (R_4) ⟶ R_4

(R_4) + ((DCE+1)) X ((DCF)) ⟶ Q

(Q) - ((DCF)) ⟶ R_4

(Q) ⟶ ((DCF))

(DCF) + 1 ⟶ DCF

(DCE) + 2 ⟶ DCE

The HPF 1 instruction occupies only one location in the main memory as :-

```
8800    HPF 1
```

where H'88' is the Op-Code chosen for this instruction. The execution time of this instruction is 62 cycles.

19.4 Design Examples

This section gives examples of the High Pass filters designed using the special instructions described in the previous section. The filter coefficients were stored starting from the memory location H'0550'. Similarly the locations starting from H'0500' were reserved to store the state variables. The initialization routine of the filters were same as the initialization routine for the Low Pass filters.

19.4.1 First Order Butterworth Filter

This kind of filter was designed for a cutoff frequency of f_h = 500 Hz, and a sampling frequency of f_s = 3500 Hz.

We first design a Low Pass filter with a cutoff frequency of f_d where,

$$f_d = \frac{f_s}{2} - f_h$$

$$= 1750 - 500$$

Hence, f_d = 1250 Hz

The transfer function of a Low Pass Butterworth filter with f_d = 1250 and f_s = 3500 Hz is as follows :-

$$H_l(z) = \frac{0.674957\,(1 + z^{-1})}{1 + 0.349915z^{-1}}$$

The transfer function of the required High Pass filter is then obtained by replacing z^{-1} with $-z^{-1}$. Therefore,

$$H_h(z) = \frac{0.674957\,(1 - z^{-1})}{1 - 0.349915z^{-1}}$$

Representing the coefficients in 16 bit fractional form, we have :-

D = 0.674957 = H' 5665'

E = 0.349915 = H' 2CCA'

The program written to implement this filter was as follows :-

```
*
* INITIALIZATION ROUTINE
*
0600   3600   SM H'FE00'      Mask all interrupts except 0
0601   FE00
0602   15E0   LI E, H'0550'   Set DCE to start of coefficient
0603   0550                   table
0604   15F0   LI F, H'0500'   Set DCF to start of state variable
0605   0500                   table
0606   1AEE   PUSH E,E        Store DCE in Aux. Reg. E
0607   1AFF   PUSH F,F        Store DCF in Aux. Reg. F
```

```
*
* DIGITAL FILTER LOOP
*
0608    8000    IN                  Input section
0609    0100
060A    8800    HPF 1               First Order section
060B    84EF    OUT                 Output section
060C    0100
060D    8500
*
* END
*
```

Also the following data was stored in the memory :-

Address	Data	
0407	0608	Interrupt service address
0550	5665	Coefficient D
0551	2CCA	Coefficient E

The frequency response of this filter is shown in Figure 19.1

19.4.2 Second Order Butterworth Filter

This filter was designed for a cutoff frequency of f_h = 500 Hz and a sampling frequency of f_s = 3500 Hz.

A Low Pass Butterworth filter with a cutoff frequency of f_d = 1250 Hz and a sampling frequency of 3500 Hz has the transfer function of :-

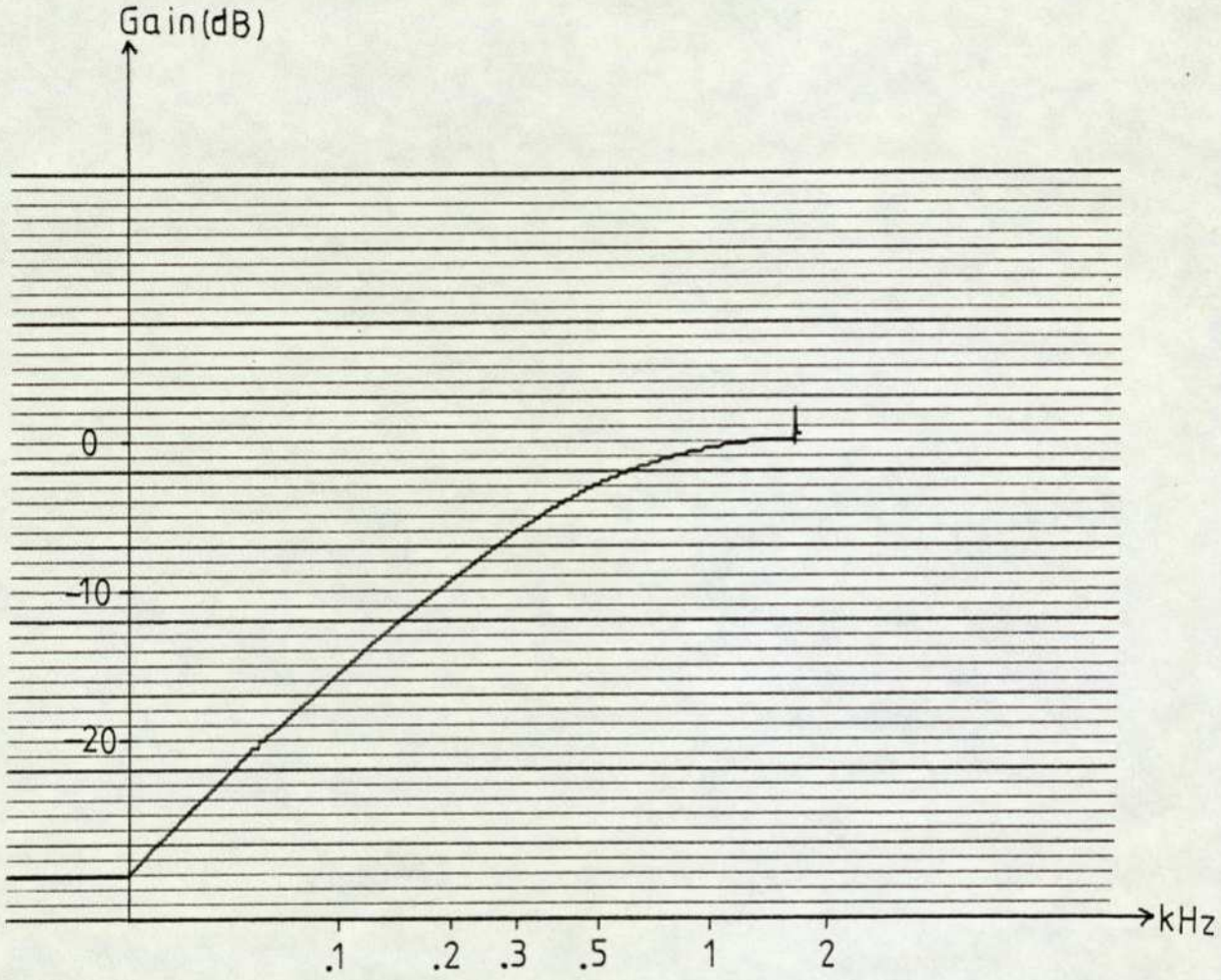


Figure 19.1 First Order Butterworth Digital Filter (High-Pass). f_h = 500 Hz, f_s = 3.5 kHz

$$H_1(z) = \frac{0.52274913(1 + 2z^{-1} + z^{-2})}{1 + 0.80303247z^{-1} + 0.2879641z^{-2}}$$

The transfer function of the required High Pass filter is obtained by replacing z^{-1} with $-z^{-1}$. Therefore,

$$H_h(z) = \frac{0.52274913(1 - 2z^{-1} + z^{-2})}{1 - 0.80303247z^{-1} + 0.2879641z^{-2}}$$

Representing the coefficients in 16 bit fractional form, we have :-

K = 0.52274913 = H'42E9'
B = 0.80303247 = H'66CA'
-C =-0.2879641 = H'DB24'

The program written to implement this filter was as follows (only the digital filter loop is shown) :-

```
*
* DIGITAL FILTER LOOP
*
0608   8000   IN                Input section
0609   0100
060A   8600   HPFL 2            Second Order section
060B   84EF   OUT               Output section
060C   0100
060D   8500
*
* END
*
```

Also the following data was stored in the memory :-

Address	Data	
0407	0608	Interrupt service address
0550	42E9	Coefficient K
0551	66CA	Coefficient B
0552	D824	Coefficient -C

The frequency response of this filter is shown in Figure 19.2 .

19.4.3 Second Order Chebyshev I Filter

This filter was designed for a ripple cutoff frequency of f_{rh} = 500 Hz, a sampling frequency of f_s = 3500 Hz and a passband ripple of 3 dB.

We first design a Low Pass Chebyshev I filter with a ripple cutoff frequency f_r where,

$$f_r = \frac{f_s}{2} - f_{rh}$$

$$= 1750 - 500$$

Hence, f_r = 1250 Hz

A Low Pass Chebyshev I filter with f_r = 1250 Hz, f_s = 3500 Hz and a passband ripple of 3 dB has the transfer function :-

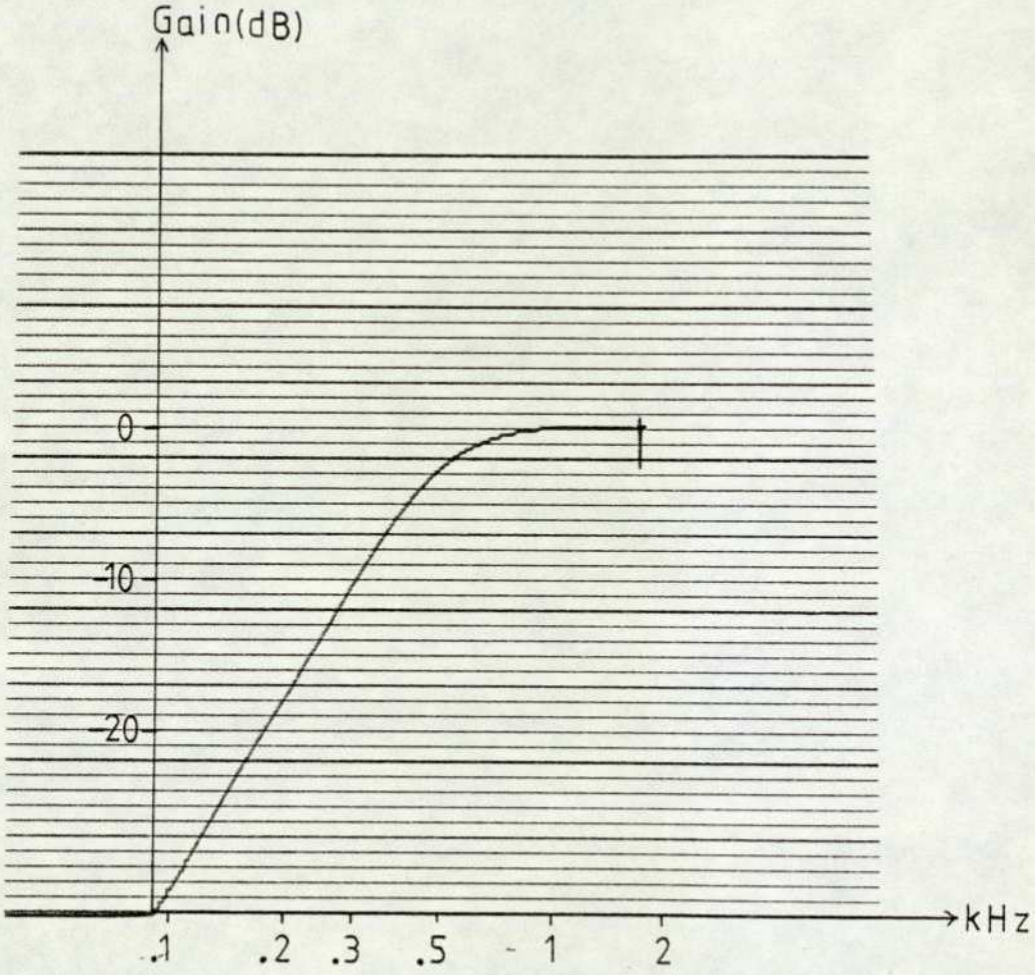


Figure 19.2 Second Order High Pass Butterworth Digital Filter. (f_s = 3.5 kHz, f_h = 500 Hz)

$$H_1(z) = \frac{0.566163827(1 + 2z^{-1} + z^{-2})}{1 + 0.7613924365z^{-1} + 0.50326287z^{-2}}$$

The transfer function of the required High Pass filter is given by :-

$$H_h(z) = \frac{0.566163827(1 - 2z^{-1} + z^{-2})}{1 - 0.7613924365z^{-1} + 0.50326287z^{-2}}$$

Representing the coefficients in 16 bit fractional form, we have :-

K = 0.566163827 = H'4878'
B = 0.761392436 = H'6175'
-C =-0.50326287 = H'BF95'

The program written to implement this filter was same as the 2nd Order Butterworth program but the coefficient table was modified as follows :-

Address	Data	
0550	4878	Coefficient K
0551	6175	Coefficient B
0552	BF95	Coefficient -C

The frequency response of this filter is shown in Figure 19.3 .

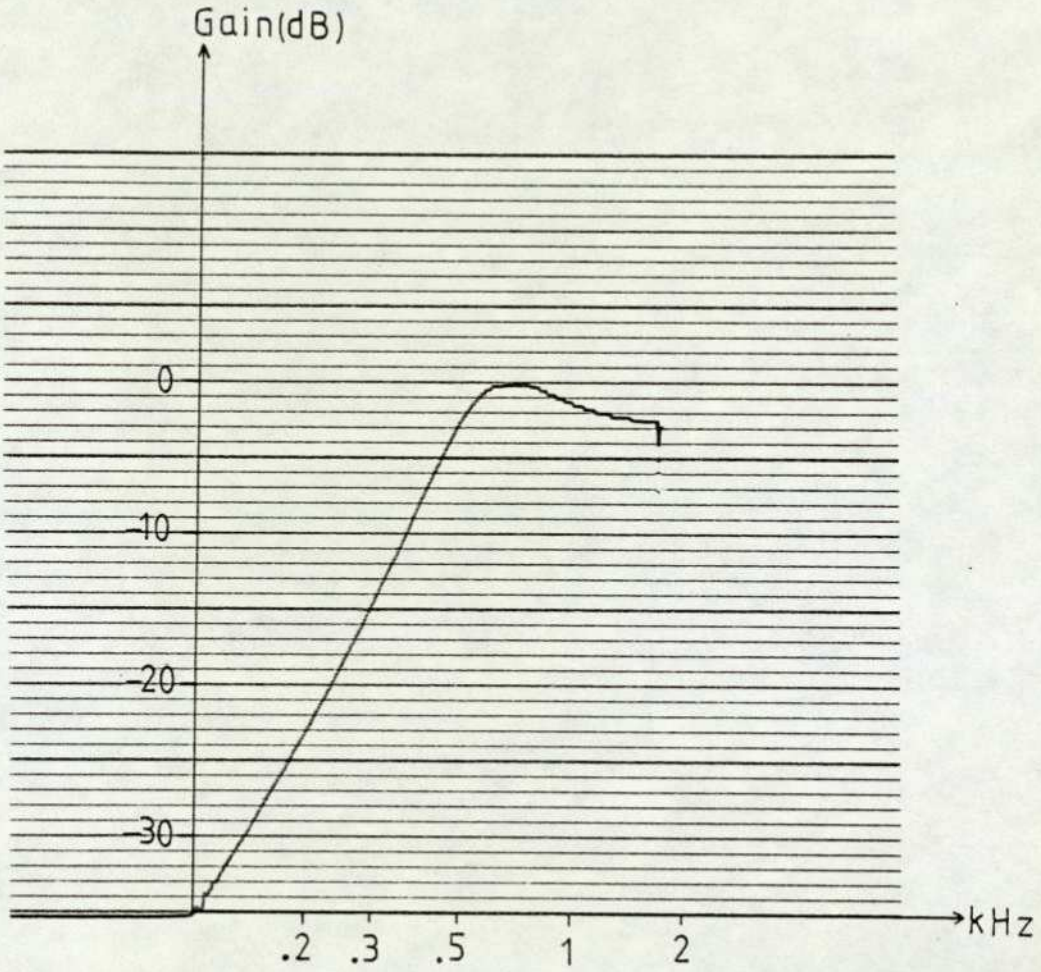


Figure 19.3 Second Order High Pass Chebyshev I Digital Filter. (f_s = 3.5 kHz, f_{rh} = 500 Hz, ripple = 3 dB)

19.4.4 Third Order Chebyshev I Filter

This filter was designed for a ripple cutoff frequency of f_{rh} = 500 Hz, a sampling frequency of f_s = 3500 Hz and a passband ripple of 3 dB.

A Low Pass Chebyshev I filter (3^{rd} Order) with a ripple cutoff frequency of f_r = 1250 Hz, a passband ripple of 3 dB and a sampling frequency of 3500 Hz has the transfer function :-

$$H_l(z) = \frac{0.6907373064(1+2z^{-1}+z^{-2})}{1+0.999690324z^{-1}+0.76325890z^{-2}} \cdot \frac{0.382751(1+z^{-1})}{1-.234498364z^{-1}}$$

The transfer function of the required High Pass filter is given by replacing z^{-1} with $-z^{-1}$. Therefore,

$$H_h(z) = \frac{0.6907373064(1-2z^{-1}+z^{-2})}{1-0.999690324z^{-1}+0.7632589z^{-2}} \cdot \frac{0.382751(1-z^{-1})}{1+.234498364z^{-1}}$$

Representing the coefficients in 16 bit fractional form, we have :-

K = 0.6907373064 = H'586A'
B = 0.999690324 = H'7FF6'
-C =-0.76325890 = H'9E4E'
D = 0.382751 = H'30FE'
E =-0.234498364 = H'E1FC'

The program written to implement this filter was as follows (only the digital filter loop is shown) :-

```
*
* DIGITAL FILTER LOOP
*
0608    8000    IN              Input section
0609    0100
060A    8600    HPFL 2          Second Order section
060B    8800    HPF 1           First Order section
060C    84EF    OUT             Output section
060D    0100
060E    8500
*
* END
*
```

Also the following data was stored in the main memory:-

Address	Data	
0407	0608	Interrupt service address
0550	586A	Coefficient K
0551	7FF6	Coefficient B
0552	9E4E	Coefficient -C
0553	30FE	Coefficient D
0554	E1FC	Coefficient E

The frequency response of this filter is given in Figure 19.4

19.4.5 Fourth Order Butterworth Filter

This filter was designed for a cutoff frequency of f_h = 500 Hz and a sampling frequency of f_s = 3500 Hz.

A Low Pass 4^{th} Order Butterworth filter with a cutoff

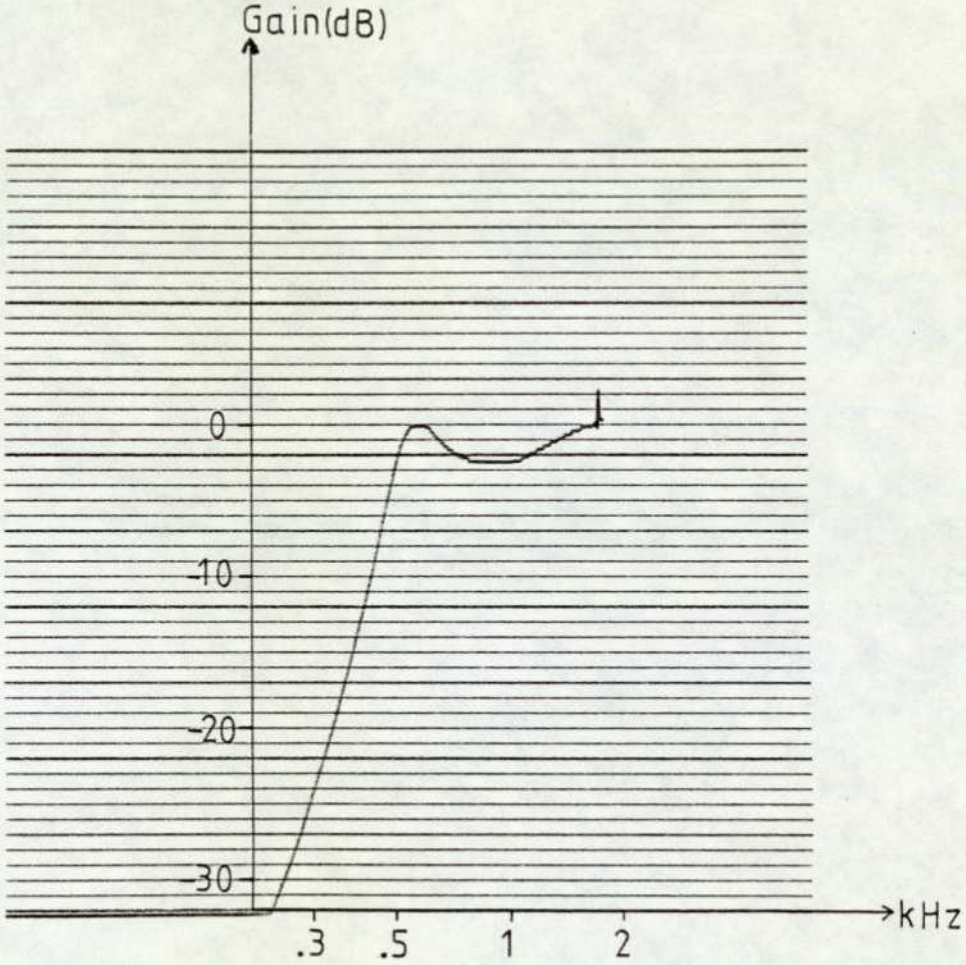


Figure 19.4 Third Order High Pass Chebyshev I Digital Filter. (f_s = 3.5 kHz, f_{rh} = 500 Hz, ripple = 3 dB)

frequency of f_d = 1250 Hz and a sampling frequency of 3500 Hz has the transfer function of :-

$$H_l(z) = \frac{0.624806556(1+2z^{-1}+z^{-2})}{1+0.9598102z^{-1}+0.53941603z^{-2}} \cdot \frac{0.47130951(1+2z^{-1}+z^{-2})}{1+.72401236z^{-1}+.1612257z^{-2}}$$

The transfer function of the required High Pass filter is obtained by replacing z^{-1} with $-z^{-1}$. Therefore,

$$H_h(z) = \frac{0.624806556(1-2z^{-1}+z^{-2})}{1-0.9598102z^{-1}+0.53941603z^{-2}} \cdot \frac{0.47130951(1-2z^{-1}+z^{-2})}{1-.72401236z^{-1}+.1612257z^{-2}}$$

Representing the coefficients in 16 bit fractional form, we have :-

K_1 = 0.624806556 = H'4FFA'
B_1 = 0.9598102 = H'7ADB'
$-C_1$ =-0.53941603 = H'BAF4'
K_2 = 0.47130951 = H'3C54'
B_2 = 0.72401236 = H'5CAC'
$-C_2$ =-0.1612257 = H'EB5D'

The program written to implement this filter was as follows (only the digital filter loop is shown) :-

```
*
* DIGITAL FILTER LOOP
*
0608   8000   IN              Input section
0609   0100
060A   8600   HPFL 2          Second Order section
060B   8600   HPFL 2          Second Order section
060C   84EF   OUT             Output section
```

```
060D    0100
060E    8500
*
* END
*
```

Also the following data was stored in the main memory:-

Address	Data	
0407	0608	Interrupt service address
0550	4FFA	Coefficient K_1
0551	7AD8	Coefficient B_1
0552	BAF4	Coefficient $-C_1$
0553	3C54	Coefficient K_2
0554	5CAC	Coefficient B_2
0555	EB5D	Coefficient $-C_2$

The frequency response of this filter is shown in Figure 19.5 .

19.4.6 Sixth Order Butterworth Filter

This filter was designed for a cutoff frequency of f_h = 400 Hz and a sampling frequency of f_s = 2000 Hz.

We first design a Low Pass filter with a cutoff frequency of f_d where,

$$f_d = \frac{f_s}{2} - f_h$$

$$= 1000 - 400$$

Hence f_d = 600 Hz

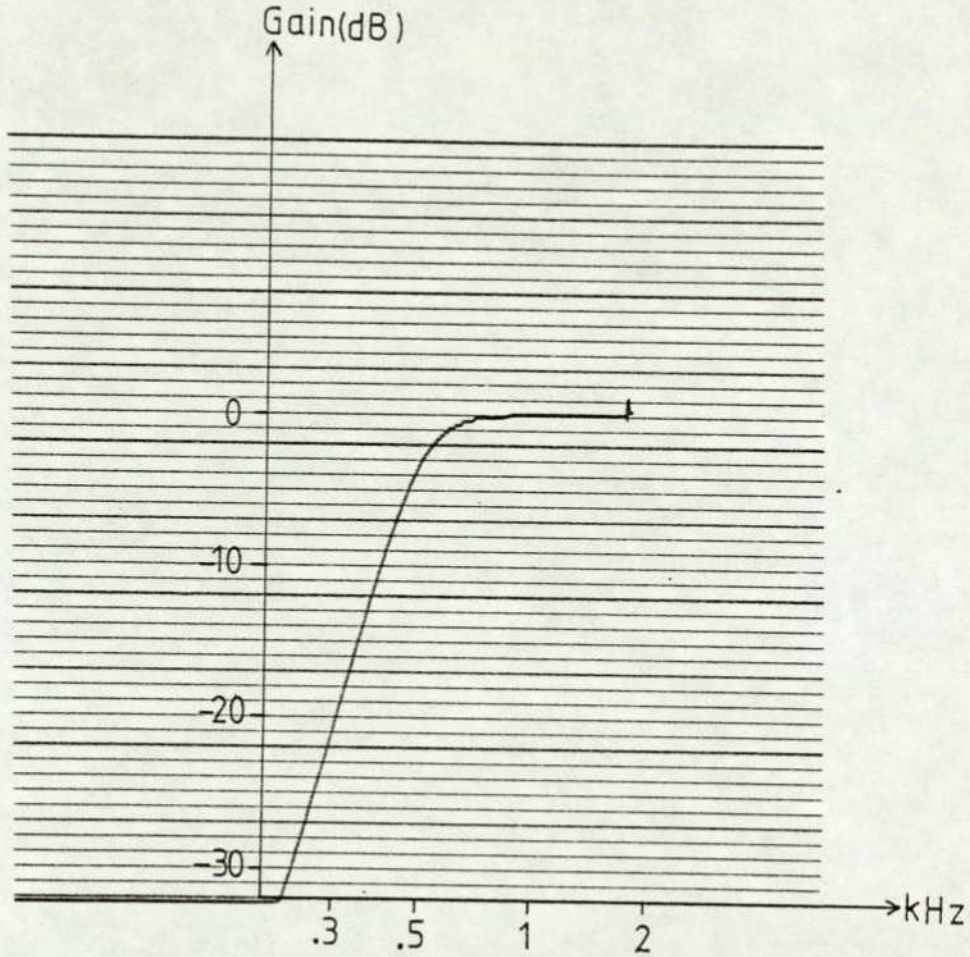


Figure 19.5 Fourth Order High Pass Butterworth Digital Filter. (f_s = 3.5 kHz, f_h = 500 Hz)

The transfer function of a 6^{th} Order Low Pass Butterworth filter with f_d = 600 Hz and f_s = 2000 Hz is as follows :-

$$H_1(z) = \frac{0.5252238(1+2z^{-1}+z^{-2})}{1+0.4959542z^{-1}+0.6049412z^{-2}} \cdot \frac{0.3411297(1+2z^{-1}+z^{-2})}{1+0.3221192z^{-1}+.0423995z^{-2}} X$$

$$X \frac{0.3913357(1+2z^{-1}+z^{-2})}{1+0.3695273z^{-1}+0.195816z^{-2}}$$

The transfer function of the required High Pass filter is obtained by replacing z^{-1} with $-z^{-1}$. Therefore,

$$H_h(z) = \frac{0.5252238(1-2z^{-1}+z^{-2})}{1-0.4959542z^{-1}+0.6049412z^{-2}} \cdot \frac{0.3411297(1-2z^{-1}+z^{-2})}{1-0.3221192z^{-1}+.0423995z^{-2}} X$$

$$X \frac{0.3913357(1-2z^{-1}+z^{-2})}{1-0.3695273z^{-1}+0.195816z^{-2}}$$

Representing the coefficients in 16 bit fractional form, we have :-

K_1 = 0.5252238 = H'433B'
B_1 = 0.4959542 = H'3F7B'
$-C_1$ =-0.6049412 = H'B291'
K_2 = 0.3411297 = H'28AA'
B_2 = 0.3221192 = H'2938'
$-C_2$ =-0.0423995 = H'FA93'
K_3 = 0.3913357 = H'3217'
B_3 = 0.3695273 = H'2F4D'
$-C_3$ =-0.195816 = H'E6F0'

The program written to implement this filter was as follows (only the digital filter loop is shown) :-

```
*
* DIGITAL FILTER LOOP
*
0608   8000   IN            Input section
0609   0100
060A   8600   HPFL 2        Second Order section
0608   8600   HPFL 2        Second Order section
060C   8600   HPFL 2        Second Order section
060D   84EF   OUT           Output section
060E   0100
060F   8500
*
* END
*
```

Also the following data was stored in the main memory:-

Address	Data	
0407	0608	Interrupt service address
0550	433B	Coefficient K_1
0551	3F7B	Coefficient B_1
0552	B291	Coefficient $-C_1$
0553	2BAA	Coefficient K_2
0554	2938	Coefficient B_2
0555	FA93	Coefficient $-C_2$
0556	3217	Coefficient K_3
0557	2F4D	Coefficient B_3
0558	E6F0	Coefficient $-C_3$

The frequency response of this filter is shown in Figure 19.6 .

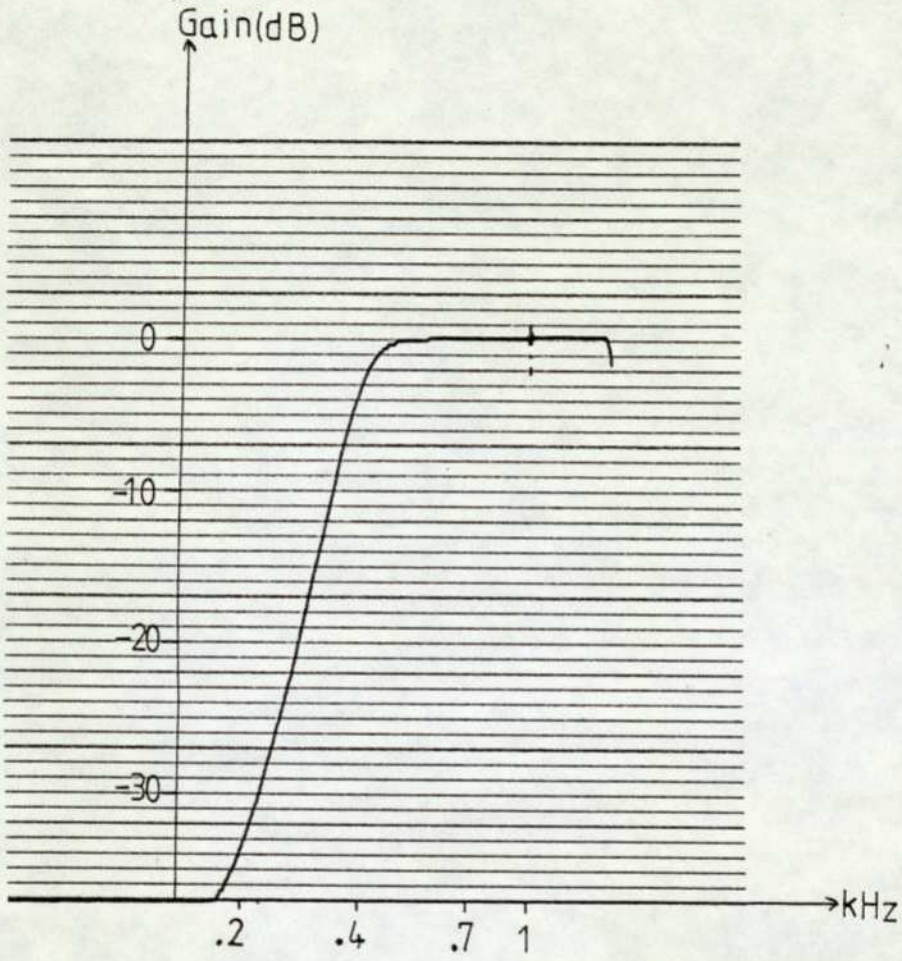


Figure 19.6 Sixth Order High Pass Butterworth Digital Filter. (f_s = 2 kHz, f_h = 400 Hz)

19.4.7 Eighth Order Butterworth Filter

This filter was designed for a cutoff frequency of f_h = 400 Hz and a sampling frequency of f_s = 2000 Hz.

Like the 6th Order filter, we first design a Low Pass filter for a cutoff frequency of f_d = 600 Hz and a sampling frequency of f_s = 2000 Hz :-

$$H_1(z) = \frac{0.55207537(1+2z^{-1}+z^{-2})}{1+0.5213092z^{-1}+0.6869922z^{-2}} \cdot \frac{0.36548904(1+2z^{-1}+z^{-2})}{1+0.3451210z^{-1}+.11683514z^{-2}} X$$

$$X \frac{0.3386354(1+2z^{-1}+z^{-2})}{1+0.31976390z^{-1}+0.0347777z^{-2}} \cdot \frac{0.4282371(1+2z^{-1}+z^{-2})}{1+0.4043723z^{-1}+.3085762z^{-2}}$$

The transfer function of the required High Pass filter is obtained by replacing z^{-1} with $-z^{-1}$. Therefore,

$$H_h(z) = \frac{0.55207537(1-2z^{-1}+z^{-2})}{1-0.5213092z^{-1}+0.6869922z^{-2}} \cdot \frac{0.36548904(1-2z^{-1}+z^{-2})}{1-0.3451210z^{-1}+.11683514z^{-2}}$$

$$X \frac{0.3386354(1-2z^{-1}+z^{-2})}{1-0.31976390z^{-1}+0.0347777z^{-2}} \cdot \frac{0.4282371(1-2z^{-1}+z^{-2})}{1-0.4043723z^{-1}+.3085762z^{-2}}$$

Representing the coefficients in 16 bit fractional form, we have:-

K_1 = 0.55207537 = H'46AA'
B_1 = 0.5213092 = H'42BA'
$-C_1$ =-0.6869922 = H'A811'
K_2 = 0.36548904 = H'2EC8'
B_2 = 0.3451210 = H'2C2D'
$-C_2$ =-0.11683514 = H'F10C'

K_3 = 0.3386354 = H'2B58'
B_3 = 0.31976390 = H'28EE'
$-C_3$ =-0.0347777 = H'F88C'
K_4 = 0.4282371 = H'36D0'
B_4 = 0.4043723 = H'33C2'
$-C_4$ =-0.3085762 = H'D881'

The program written to implement this filter was as follows (only the digital filter loop is shown) :-

```
*
* DIGITAL FILTER LOOP
*
0608   8000   IN              Input section
0609   0100
060A   8600   HPFL 2          Second Order section
060B   8600   HPFL 2          Second Order section
060C   8600   HPFL 2          Second Order section
060D   8600   HPFL 2          Second Order section
060E   84EF   OUT             Output section
060F   0100
0610   8500
*
* END
*
```

Also the following data was stored in the memory :-

Address	Data	
0407	0608	Interrupt service address
0550	46AA	Coefficient K_1
0551	42BA	Coefficient B_1
0552	A811	Coefficient $-C_1$
0553	2EC8	Coefficient K_2
0554	2C2D	Coefficient B_2
0555	F10C	Coefficient $-C_2$

0556	2B58	Coefficient K_3
0557	28EE	Coefficient B_3
0558	FB8C	Coefficient $-C_3$
0559	36D0	Coefficient K_4
055A	33C2	Coefficient B_4
055B	D881	Coefficient $-C_4$

The frequency response of this filter is shown in Figure 19.7 .

19.5 Microcode of the High Pass Filter Instructions

The microcode of the three High Pass filter instructions was implemented as an extension of the microcode written for the Low Pass filter instructions.

Figure 19.8 shows the microcode developed to implement the High Pass filters. The description of the microcode is as follows (see also Chapter 18.7) :-

Macroinstruction		Microoperations performed
HPFL 2		$0 \longrightarrow R_6$
	MIC7:	Call to microsubroutine SUB2
		goto MIC6
HPFG 2		H'8000' $\longrightarrow R_6$
		goto MIC7
HPF 1		Call to microsubroutine SUB2
		$(R_0) + (R_4) \longrightarrow Q$
		$(Q) - (R_2) \longrightarrow R_4$
		$(DCF) - 1 \longrightarrow DCF$

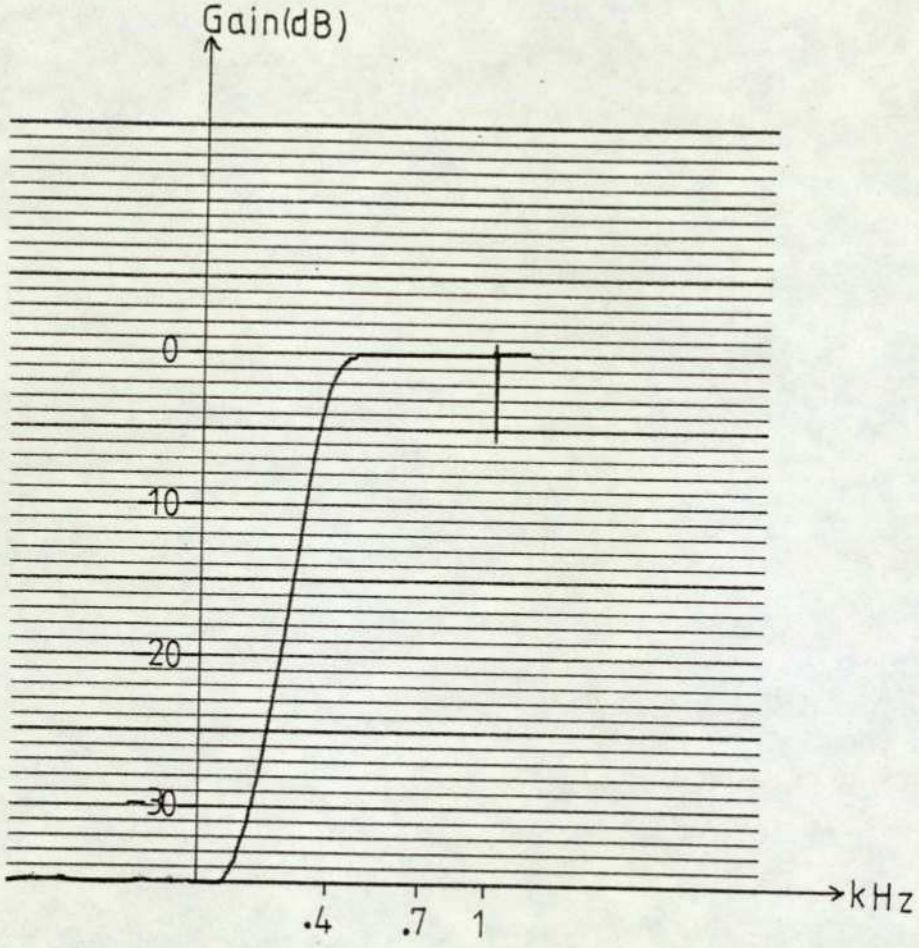


Figure 19.7 Eighth Order High Pass Butterworth Digital Filter. (f_s = 2 kHz, f_h = 400 Hz)

```
        (Q) ⟶ ((DCF))
        (DCF) + 1 ⟶ DCF
        Fetch the next macroinstruction

MIC6:   (R0) ⟶ R3
        (R2) ⟶ R5
        Call to microsubroutine SUB1
        (R0) + (R3) ⟶ Q
        (R4) + (Q) ⟶ Q
        If (R6) ≠ 0 goto MIC8
MIC9:   (R2) + (Q) ⟶ R4
        (R4) - (R5) ⟶ R4
        (R4) - (R5) ⟶ R4
        (DCF) - 1 ⟶ DCF
        R5 ⟶ ((DCF))
        (DCF) - 1 ⟶ DCF
        (Q) ⟶ ((DCF))
        (DCF) + 2 ⟶ DCF
        Fetch the next macroinstruction
MIC8:   (Q) - (R5) ⟶ Q
        goto MIC9
```

INSTRUCTION	OP CODE	ADDRESS	FIELD 5								FIELD 4								FIELD 3										FIELD 2								FIELD 1								FIELD 0							
			ICU			REL	PCU	CCS			Am 2910					Inst. Reg.	DSL			IOM	MMD	P/P	PCU				Branch Address								Dest.				Function				ICU Source			ALU			Operand Select	
			A_3	MRQ	WAIT	FE	PCE	ST_1	ST_2	IOR	I_3	I_2	I_1	I_0	EDB	IRE	S_4	S_5	RS_3	C_8	C_4	C_5	C_6	S_3	S_2	S_1	C_3	PE	P_4	P_3	P_2	P_1	BR_{11}	BR_{10}	BR_9	BR_8	BR_7	BR_6	BR_5	BR_4	BR_3	BR_2	BR_1	BR_0	A_2	A_1	A_0	IEN	$\overline{OE}_Y$	EB	RS_1	RS_2
			47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HPFL 2	86	136	1	1	1	1	1	0	0	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	X	0	1	1	0	0	0	X	X	0	1	0	0	1	1	0	0	0	1	0	0	1	0	1
Call SUB2	.	137	1	1	1	1	1	1	1	1	0	0	0	1	1	1	X	X	X	1	1	1	X	X	X	0	X	X	X	X	0	0	0	1	0	0	1	0	0	1	0	0	X	X	X	1	1	1	X	X
Goto MIC6	.	138	1	1	1	1	1	1	1	1	0	0	1	1	1	1	X	X	X	1	1	1	X	X	X	0	X	X	X	X	0	0	0	1	0	1	0	0	0	0	0	1	X	X	X	1	1	1	X	X
HPFG 2	87	139	1	1	1	1	1	0	0	1	0	0	1	1	1	1	0	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0	1	0	0	1	1	0	1	1	1	0	1	0	0	0	1	0	1
HPF 1	88	13A	1	1	1	1	1	1	1	1	0	0	0	1	1	1	X	X	X	1	1	1	X	X	X	0	X	X	X	X	0	0	0	1	0	0	1	0	0	1	0	0	X	X	X	1	1	1	X	X
R0+R4→Q	.	13B	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	X	X	X	0	0	1	X	0	1	0	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	1	0	1
Q-R2→R4	.	13C	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	0	X	0	0	1	0	1
DCF-1→DCF	.	13D	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	X	X	X	0	0	1	X	1	1	1	1	0	0	X	X	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1
	.	13E	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	0	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	0	0	1
	.	13F	1	0	1	1	0	1	1	1	1	1	1	0	0	1	X	X	X	X	X	X	X	1	X	1	0	1	1	0	0	0	X	X	0	1	1	0	1	1	1	1	1	1	X	0	0	1	X	X
DCF+1→DCF	.	140	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	1	1	1	0	0	0	1	1	1	1	1	X	X	X	X	0	1	0	0	0	1	0	0	0	0	0	0	1	0	1
MIC6	.	141	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	1	X	X	X	0	0	1	X	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	1	0	1
R2→R5	.	142	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	1	X	X	X	0	0	1	X	0	1	0	1	0	0	1	0	0	1	0	0	0	1	1	0	0	0	0	0	1	0	1
Call SUB1	.	143	1	1	1	1	1	1	1	1	0	0	0	1	1	1	X	X	X	1	1	1	X	X	X	0	X	X	X	X	0	0	0	1	0	0	0	0	0	0	0	0	X	X	X	1	1	1	X	X
R0+R3→Q	.	144	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	X	X	X	0	0	1	X	0	0	1	1	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	1	0	1
R4+Q→Q	.	145	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	X	X	X	0	0	1	X	X	X	X	X	0	1	0	0	0	1	1	0	0	0	1	1	1	0	X	0	1	1	0	1

Figure 19.8 Microcode of the High Pass filter instructions

cont...

	OP-CODE	ADDRESS	FIELD 5								FIELD 4								FIELD 3									FIELD 2								FIELD 1								FIELD 0							
			ICU			REL	PCU	CCS			Am 2910				EDB	Inst. Req.	DSL			IOM	MMD	P/P	CCS, Carry-in					PCU				Branch Address								Dest.				Function				ICU Source			ALU			Operand Select	
			A_3	MRQ	WAIT	FE	PCE	ST_1	ST_2	IOR	I_3	I_2	I_1	I_0		IRE	S_4	S_5	RS_3					C_8	C_4	C_5	C_6	S_3	S_2	S_1	C_3	PE	P_4	P_3	P_2	P_1	BR_{11}	BR_{10}	BR_9	BR_8	BR_7	BR_6	BR_5	BR_4	BR_3	BR_2	BR_1	BR_0	A_2	A_1	A_0				RS_1	RS_2
INSTRUCTION			47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
If $R_6 \neq 0$ goto MIC8	.	146	1	1	1	1	1	1	1	1	0	0	1	1	1	1	X	X	X	0	0	1	X	X	X	0	X	X	X	X	0	0	0	1	0	1	0	1	0	0	0	1	X	X	X	1	1	1	X	X
MIC9	.	147	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	X	X	X	0	0	1	X	0	1	0	0	0	0	1	0	0	1	0	0	0	0	1	1	1	0	X	0	0	1	0	1
$R_4 - R_5 \rightarrow R_4$	.	148	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	0	0	0	1	0	1	0	0	0	1	0	1	0	1	0	0	0	0	0	1	0	0	0	0	1	0	1
$R_4 - R_5 \rightarrow R_4$	.	149	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	0	0	0	1	0	1	0	0	0	1	0	1	0	1	0	0	0	0	0	1	0	0	0	0	1	0	1
DCF-1 → DCF	.	14A	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	X	X	X	0	0	1	X	1	1	1	1	0	0	X	X	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1
	.	14B	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	0	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	0	0	1
	.	14C	1	0	1	1	0	1	1	1	1	1	1	0	0	1	X	1	X	X	X	X	X	1	X	1	0	1	1	0	0	1	0	1	0	1	0	0	1	1	0	0	0	1	0	0	0	1	1	0
DCF-1 → DCF	.	14D	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	X	X	X	0	0	1	0	1	1	1	1	0	0	X	X	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1
	.	14E	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	0	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	0	0	1
	.	14F	1	0	1	1	0	1	1	1	1	1	1	0	0	1	X	X	X	X	X	X	1	X	1	0	1	1	1	0	0	0	X	X	0	1	1	0	1	1	1	1	1	1	X	0	0	1	X	X
DCF+2 → DCF	.	150	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	1	1	1	0	0	0	1	1	1	1	1	X	X	X	X	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1
MIC8	.	151	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	0	0	0	1	X	X	X	X	0	1	0	1	0	1	1	0	0	0	0	1	1	0	X	0	1	1	0	1
goto MIC9	.	152	1	1	1	1	1	1	1	1	0	0	1	1	1	1	X	X	X	1	1	1	X	X	X	0	X	X	X	X	0	0	0	1	0	1	0	0	0	1	1	1	X	X	X	1	1	1	X	X
	.																																																	
	.																																																
	.																																																

Figure 19.8 cont.

CHAPTER 20

20. DESIGN OF REAL-TIME FIR DIGITAL FILTERS USING THE BIT-SLICE MICROCOMPUTER

20.1 Introduction

The FIR digital filters are very important since they can be designed to have linear phase characteristics. This Chapter describes the instructions invented to design this type of filters, using the bit-slice microcomputer.

20.2 Computing the FIR Filter Algorithm

A length N FIR digital filter has the transfer function of (see Chapter 17.7) :-

$$H(z) = \sum_{n=0}^{N-1} h_n z^{-n} \qquad (20.1)$$

or, $$H(z) = \frac{y(z)}{u(z)} = \sum_{n=1}^{N} h_n z^{-(n-1)} \qquad (20.2)$$

where, h_n are the impulse response coefficients of the filter.

The output y(z) is given by :-

$$y(z) = h_1 u(z) + h_2 u(z) z^{-1} + h_3 u(z) z^{-2} + \ldots\ldots + h_N u(z) z^{-(N-1)} \qquad \ldots.(20.3)$$

Let the input samples be stored in registers x_i where,

$$u(z)z^{-i} = x_{i+1} \qquad i = 0,1,2,3, \ldots\ldots,(N-1) \qquad (20.4)$$

Then, equation 20.3 can be written as :-

$$y = h_1x_1 + h_2x_2 + h_3x_3 + h_4x_4 + \ldots\ldots\ldots + h_Nx_N \qquad (20.5)$$

Equation 20.5 can be used to implement the FIR filters as follows (see also Figure 20.1) :-

LOOP : Input to x_1 via A/D converter
Calculate : $h_1x_1 + h_2x_2 + h_3x_3 + \ldots\ldots + h_Nx_N$
Output the result via D/A converter
Perform : $x_{N-1} \longrightarrow x_N$
$x_{N-2} \longrightarrow x_{N-1}$
. .
. .
. .
. .
$x_2 \longrightarrow x_3$
$x_1 \longrightarrow x_2$
Enable external interrupts
WAIT : If external interrupt detected goto LOOP else goto WAIT

The above algorithm requires N multiplications and additions and N-1 register transfer operations. If N is large, the time required to execute the register transfer operations can be quite high. The following section describes a novel technique developed to minimize the operations required for the implementation of FIR digital filters.

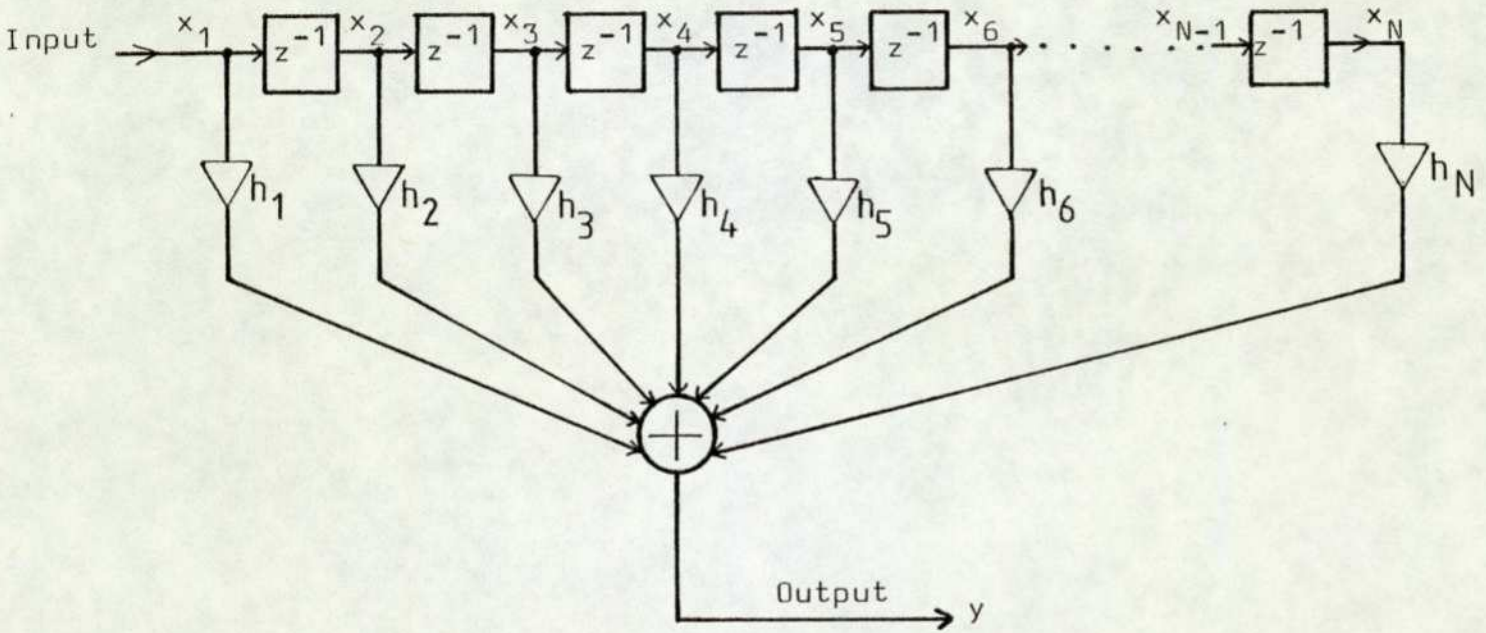


Figure 20.1 Implementing a length N FIR digital filter .

20.3 A Novel Technique for the Implementation of FIR Digital Filters

The technique developed is such that the filter coefficients h_n and the state variables x_i are stored in two blocks of the main memory as shown in Fig. 20.2, where the length of each block is equal to N. Also each memory block is organized to operate like a circulating shift register with N locations, where each location contains a 16 bit data (h_n and x_i). A new sample is always input to the location of the state variable table whose contents was multiplied by the coefficient h_N in a previous cycle of the digital filter loop (this is convenient since it is not required to store the contents of this location for the next cycle). The multiplications are performed by fetching the next coefficient from the coefficient table and the previous state variable from the state variable table. When the last data in any block is fetched (either from the beginning or end of that block), the appropriate Data Counter is modified to address the other end of that block of data i.e. the Data Counter is circulated from one end to the other end of the block.

As an example, consider the implementation of an FIR filter with N = 4. The coefficients and the state variables are stored in two blocks of the main memory as shown in Figure 20.3a. The operations required to implement this filter are then as follows (see Figures 20.3b to 20.3f) :-

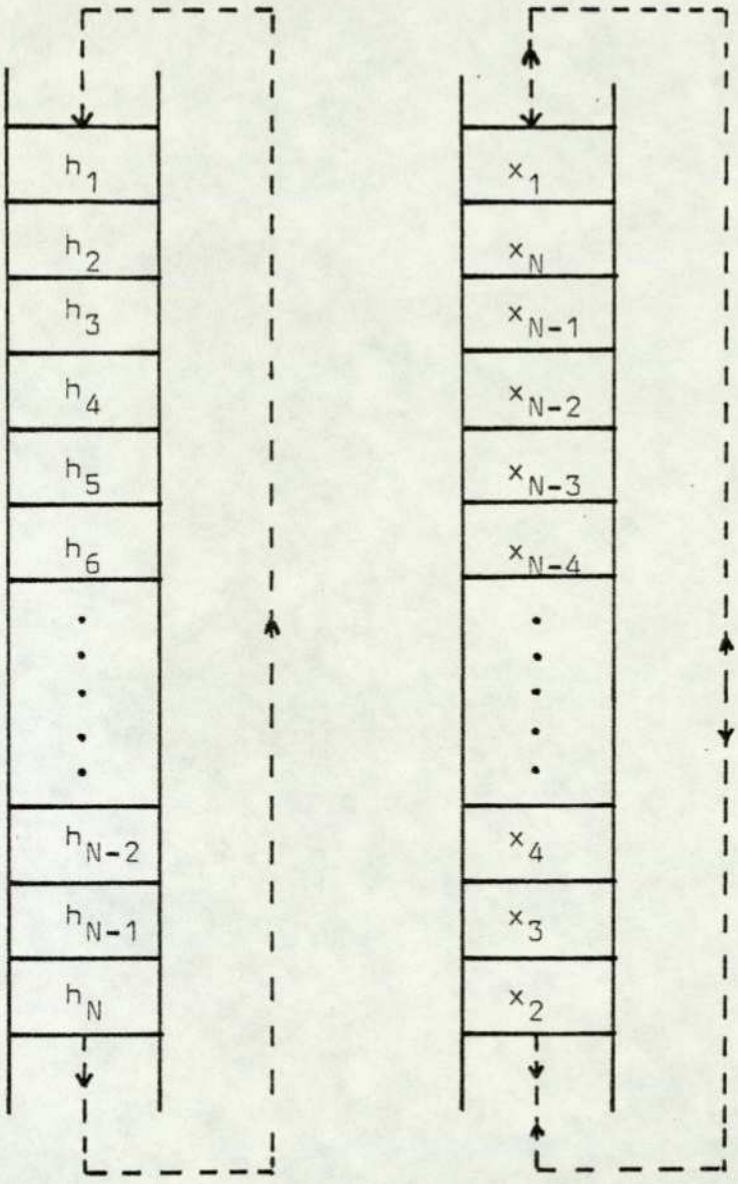


Figure 20.2 The filter coefficients and the state variables were stored in two different blocks of the main memory .

1^{st} cycle : Input to location x_1
Calculate : $h_1x_1 + h_2x_2 + h_3x_3 + h_4x_4$
Output the result
Wait for the next cycle

2^{nd} cycle : Input to location x_4
Calculate : $h_1x_4 + h_2x_1 + h_3x_2 + h_4x_3$
Output the result
Wait for the next cycle

3^{rd} cycle : Input to location x_3
Calculate : $h_1x_3 + h_2x_4 + h_3x_1 + h_4x_2$
Output the result
Wait for the next cycle

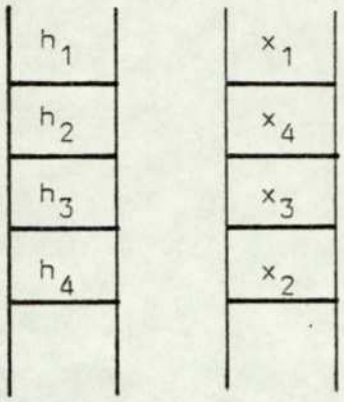
4^{th} cycle : Input to location x_2
Calculate : $h_1x_2 + h_2x_3 + h_3x_4 + h_4x_1$
Output the result
Wait for the next cycle

5^{th} cycle : goto 1^{st} cycle

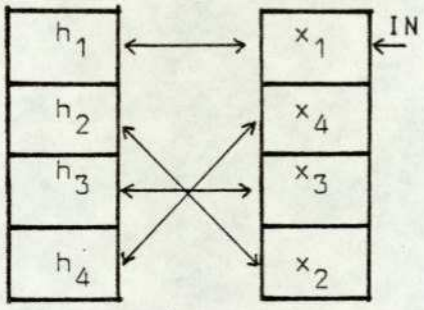
This technique has the advantage that it is not required to perform any register transfer operations and thus a higher speed is obtained. Also this algorithm can be used to design FIR filters of any length with a single program where the only external data required are the filter coefficients and the length of the filter.

20.4 Choosing an Instruction Set

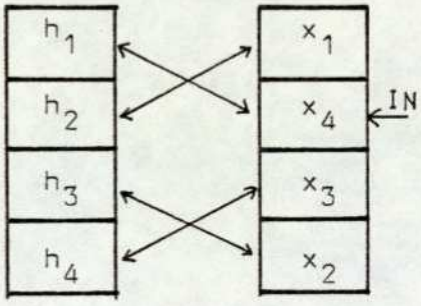
The FIR digital filter algorithm described in the previous section of this Chapter was implemented with only a single instruction (instruction 'FIR').



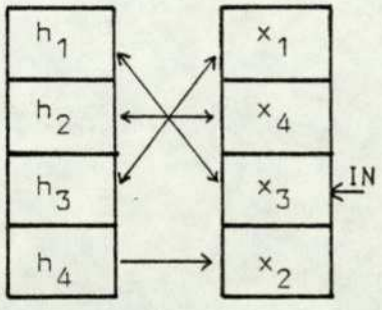
a.



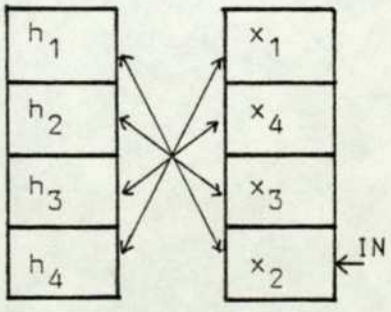
b.



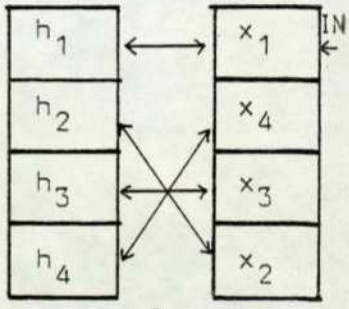
c.



d.



e.



f.

Note : Arrows indicate the quantities which are multiplied

Figure 20.3 a. two blocks of memory were used to store the coefficients and the state variables b. First cycle c. Second cycle d. Third cycle e. Fourth cycle f. Fifth cycle

The filter coefficients h_n were stored in a block of memory addressed by the Data Counter DCE. Similarly, the state variables x_i were stored in a block of memory addressed by the Data Counter DCF (see Fig. 20.4).

During the initialization routine, the DCE and the DCF are loaded with the starting addresses of the coefficient and the state variable tables respectively. Register 7 (R_7) of the ALU is loaded with the filter length, N. Similarly, register 6 (R_6) is loaded with N - 1 . The contents of DCE, DCF, and R_6 are copied to the auxiliary registers E, F, and 6 respectively.

The instruction 'FIR' performs the following operations:-

```
LOOP :  (A/D converter) ⟶ ((DCF))
        ((DCE)) X ((DCF)) ⟶ R3
MORE :  (DCF) - (Aux. reg. F)
        Jump on non-zero to 'MIC'
        (DCF) + (R7) ⟶ DCF
 MIC :  (DCF) - 1 ⟶ DCF
        (DCE) + 1 ⟶ DCE
        (R3) + ((DCE)) X ((DCF)) ⟶ R3
        (R6) - 1 ⟶ R6
        Jump on non-zero to 'MORE'
        (R3) ⟶ D/A converter
        (Aux. reg. E) ⟶ DCE
        (Aux. reg. 6) ⟶ R6
        Enable external interrupts
WAIT :  If interrupt detected goto LOOP else goto WAIT
```

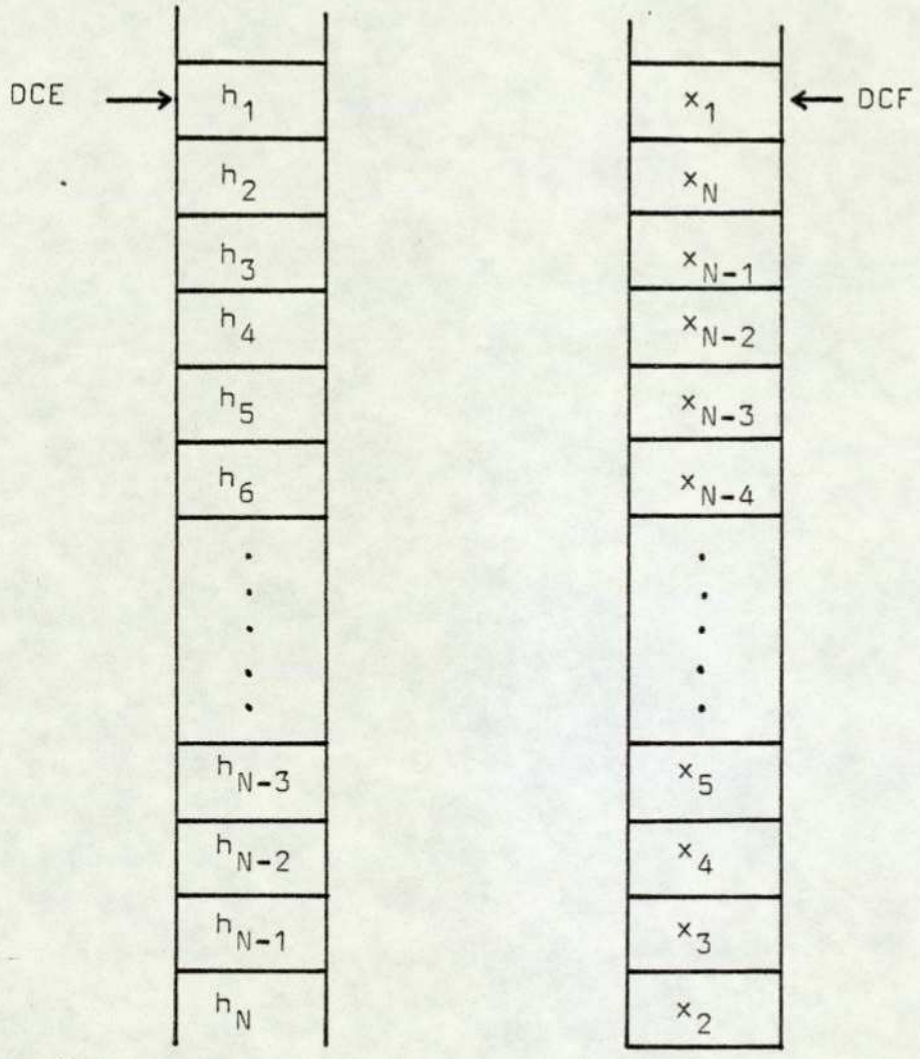


Figure 20.4 The coefficient table was addressed by the Data Counter DCE and the state variable table was addressed by the Data Counter DCF .

Note that after every multiplication the DCE is incremented by one to address the next coefficient. Also, after the first multiplication the DCF is modified such that it points to the other end of the state variable table (i.e. the location where the previous input sample was stored). Then, after every multiplication the DCF is decremented by one to address the previous input samples. After N multiplications, the DCE and register R_6 are loaded from the auxiliary registers E and 6 respectively. Thus DCE contains the address of the first coefficient and R_6 contains the filter length, N. A new sample is input to the location of the state variable table whose contents was multiplied by h_N in a previous cycle (i.e. the location addressed by the DCF at the N^{th} multiplication).

20.5 Design Examples

In the following examples, the filter coefficients were calculated as described in Chapter 17.7, and were stored in the main memory starting from H'0550'. Similarly, the locations starting from H'0500' were reserved to store the state variables. Interrupt 0 was used to synchronize the processor with the sampling rate.

A single program was written to implement FIR filters of any length, where the filter length was specified in the initialization routine :-

```
*
* INITIALIZATION ROUTINE
*
0600   3600   SM H'FE00'          Disable all interrupts except
0601   FE00                       interrupt 0
0602   15E0   LI E, H'0550'       Set DCE to start of coefficient
0603   0550                       table
0604   15F0   LI F, H'0500'       Set DCF to start of state vari-
0605   0500                       able table
0606   1570   LI 7, H'N'          Load R7 with filter length N
0607    N
0608   1560   LI 6, H'N-1'        Load R6 with N - 1
0609   N-1
060A   1AEE   PUSH E,E            Store DCE in Aux. Reg. E
060B   1AFF   PUSH F,F            Store DCF in Aux. Reg. F
060C   1A66   PUSH 6,6            Store R6 in Aux. Reg. 6
*
* DIGITAL FILTER LOOP
*
060D   8000   FIR                 FIR filter instruction
060E   0100
060F   81E6
*
* END
*
```

Note that the instruction FIR occupies only three successive locations in the main memory (H'80' is the Op-Code chosen for this instruction).

Also, the following data was loaded to the main memory :-

Address	Data	
0407	060D	Interrupt service address
0550	h_1	
0551	h_2	
0552	h_3	
0553	h_4	
.	.	
.	.	
.	.	
.	h_{N-1}	
.	h_N	

The theoretical frequency response of the filters were plotted as described in Chapter 17.7. The results obtained on the bit-slice microcomputer were plotted using a beat frequency oscillator (B & K type 1022) and a logarithmic pen recorder (B & K type 3205).

20.5.1 Low Pass Filter with N = 15 (Hamming Windowed Design)

This type of filter was designed for a cutoff frequency of f_d = 100 Hz and a sampling frequency of f_s = 1 kHz (i.e. $f_d/f_s = 0.1$). The filter coefficients were found to be:-

$$h_1 = h_{15} = -0.34597793 \times 10^{-2}$$
$$h_2 = h_{14} = -0.39151574 \times 10^{-2}$$
$$h_3 = h_{13} = 0.46912100 \times 10^{-12}$$
$$h_4 = h_{12} = 0.20470394 \times 10^{-1}$$

$h_5 = h_{11} = 0.64820658X10^{-1}$

$h_6 = h_{10} = 0.12514967$

$h_7 = h_9 = 0.17857474$

$h_8 = 0.2$

Representing the coefficients in 16 bit fractional form, we have :-

$h_1 = h_{15} =$ H'FF90'
$h_2 = h_{14} =$ H'FF80'
$h_3 = h_{13} =$ H'0000'
$h_4 = h_{12} =$ H'029F'
$h_5 = h_{11} =$ H'084C'
$h_6 = h_{10} =$ H'1005'
$h_7 = h_9 =$ H'16DC'
$h_8 =$ H'199A'

The theoretical frequency response of this filter is shown in Figure 20.5. The result obtained using the bit-slice microcomputer is shown in Figure 20.6.

20.5.2 Low Pass Filter with N = 10 (Optimum Design Algorithm)

This type of filter was designed for the following specifications:-

Sampling frequency = 1 kHz
Pass-band frequency = 0 to 100 Hz
Stop-band frequency = 200 Hz to 500 Hz

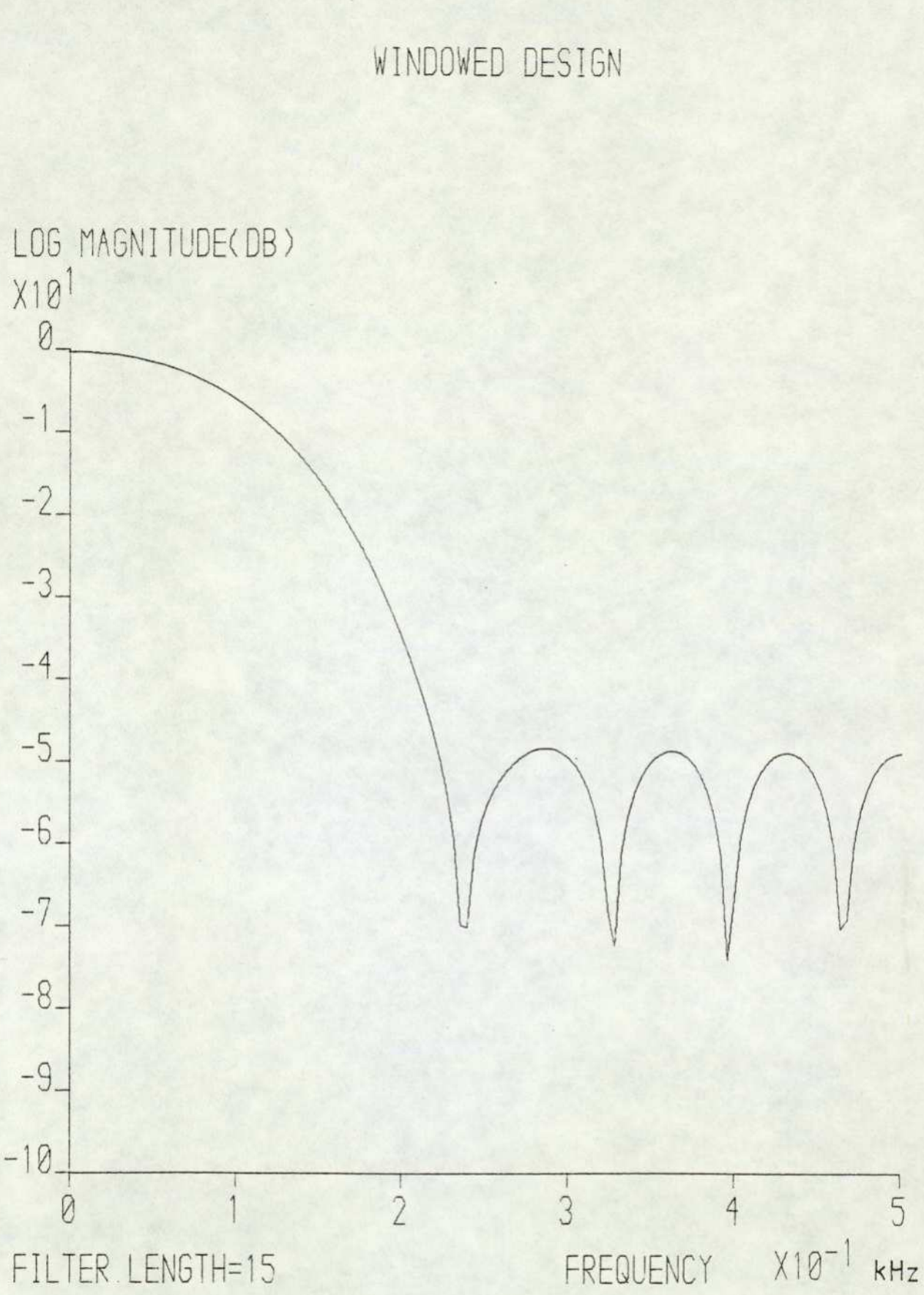


Figure 20.5 Theoretical frequency response of N=15 Low Pass FIR Digital Filter, designed with the Windowing Algorithm. (f_s = 1 kHz, f_d = 100 Hz)

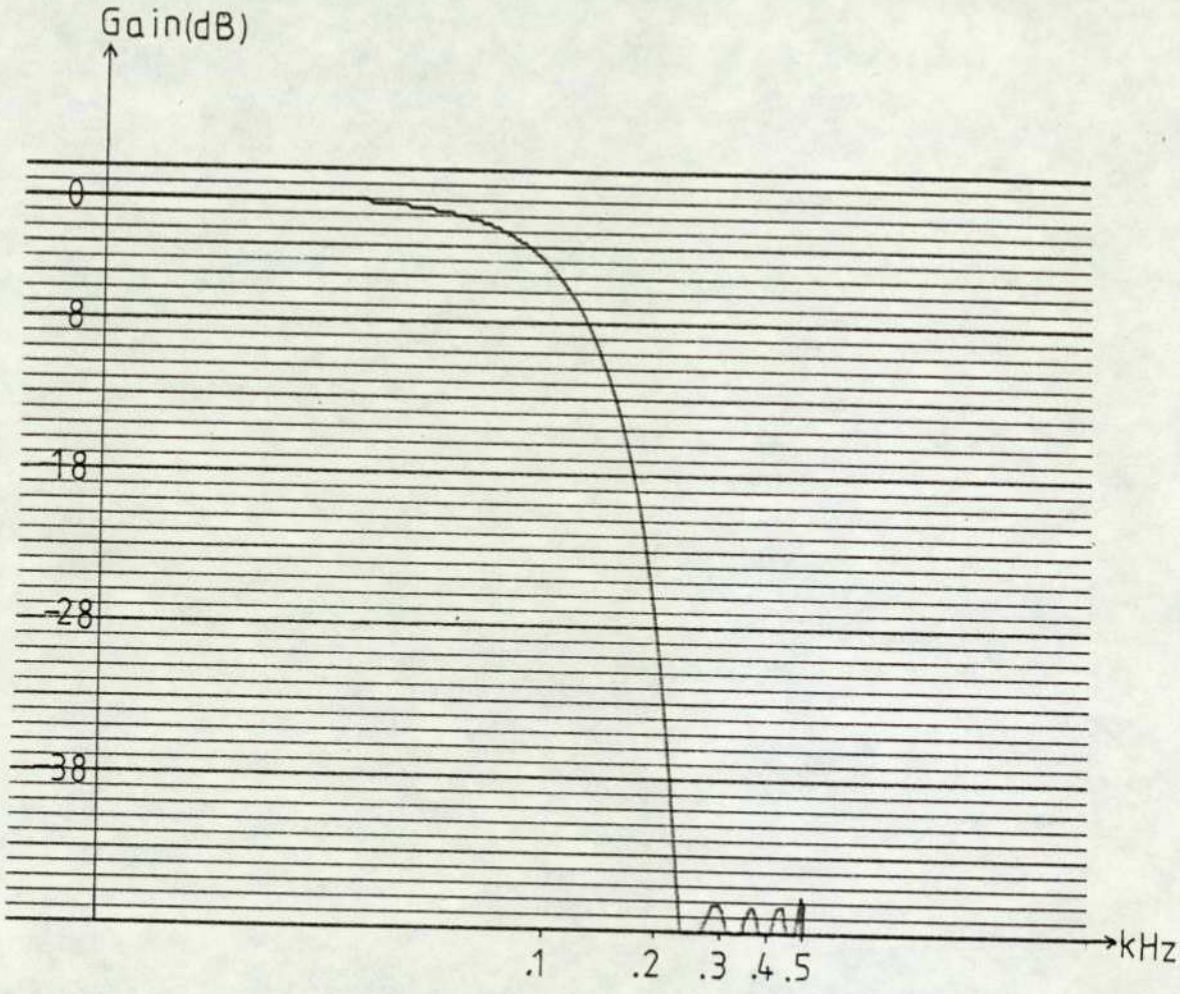


Figure 20.6 Frequency response obtained using the bit-slice microcomputer.

The filter coefficients were found to be :-

$$h_1 = h_{10} = -0.55755444X10^{-1}$$
$$h_2 = h_9 = 0.14300214X10^{-3}$$
$$h_3 = h_8 = 0.89301190X10^{-1}$$
$$h_4 = h_7 = 0.21014028$$
$$h_5 = h_6 = 0.29710663$$

Representing the coefficients in 16 bit fractional form, we have :-

$$h_1 = h_{10} = H'F8DD'$$
$$h_2 = h_9 = H'0005'$$
$$h_3 = h_8 = H'086E'$$
$$h_4 = h_7 = H'1AE6'$$
$$h_5 = h_6 = H'2608'$$

The frequency response of this filter is shown in Figures 20.7 and 20.8.

20.5.3 High Pass Filter with N = 15 (Optimum Design Algorithm)

This type of filter was designed for the following specifications :-

Sampling frequency = 1 kHz
Stop-band frequency = 0 to 100 Hz
Pass-band frequency = 200 to 500 Hz

The filter coefficients were found to be :-

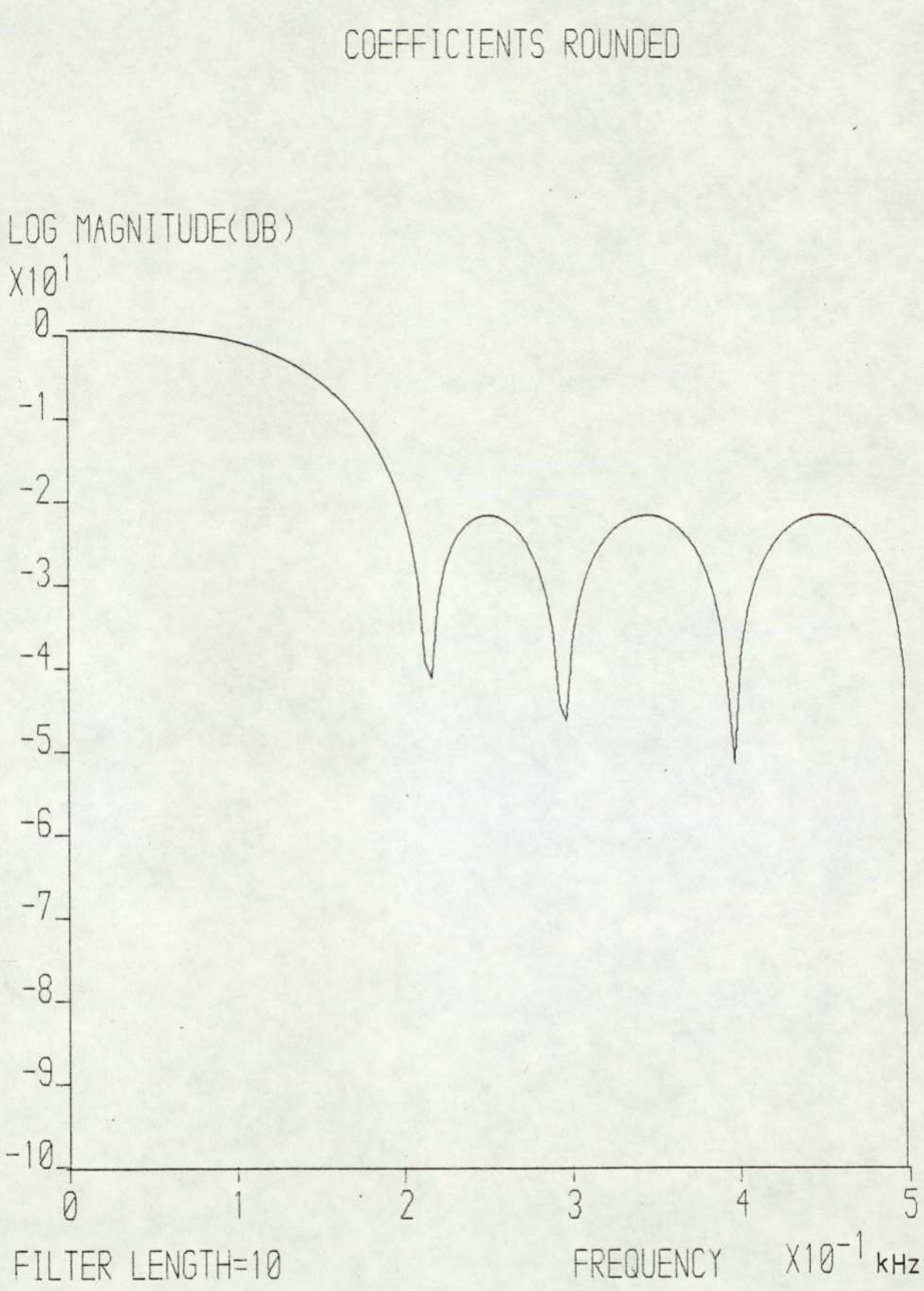


Figure 20.7 Theoretical frequency response of N=10 Low Pass FIR Digital Filter, designed with the Optimal design Algorithm.

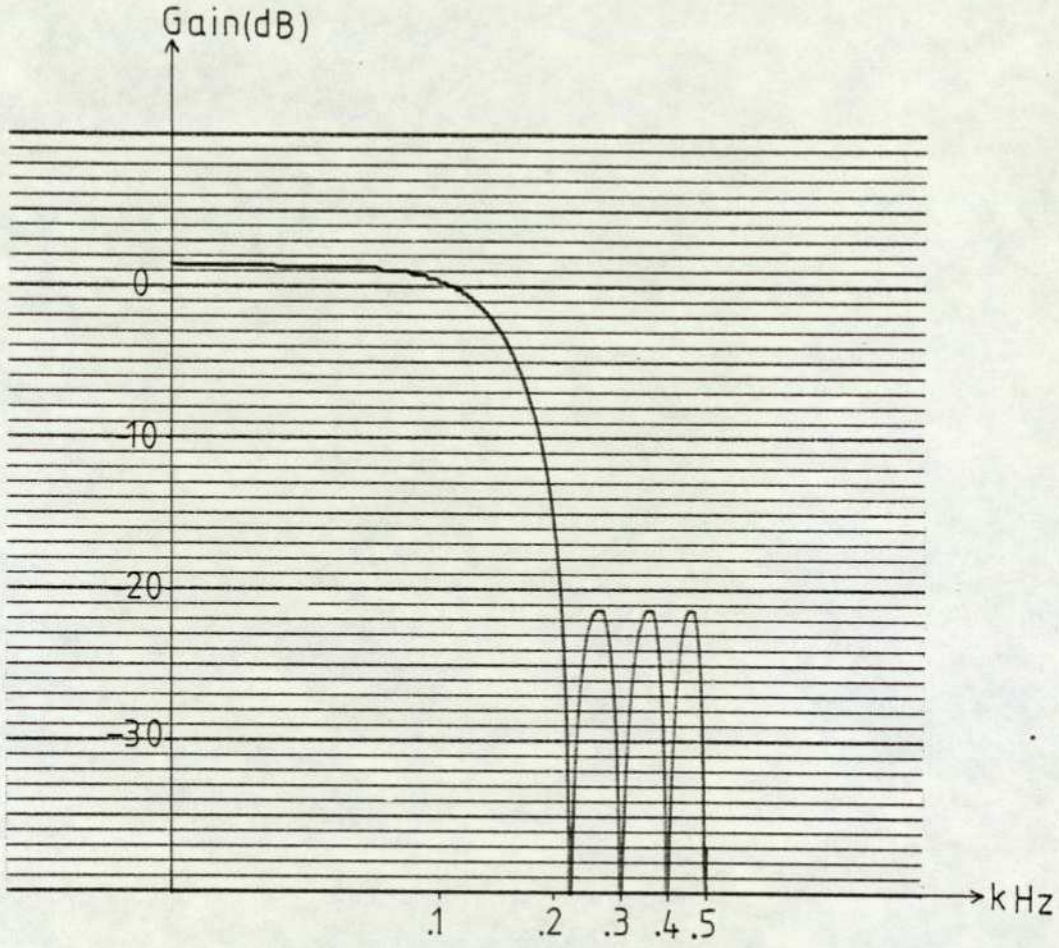


Figure 20.8 Frequency response obtained using the bit-slice microcomputer.

$h_1 = h_{15} = -0.13256093X10^{-1}$

$h_2 = h_{14} = 0.22760641X10^{-1}$

$h_3 = h_{13} = 0.44760623X10^{-1}$

$h_4 = h_{12} = 0.38051214X10^{-1}$

$h_5 = h_{11} = -0.27122399X10^{-1}$

$h_6 = h_{10} = -0.14196026$

$h_7 = h_9 = -0.25438213$

$h_8 = 0.69868914$

Representing the coefficients in 16 bit fractional form, we have :-

$h_1 = h_{15} =$ H'FE4E'

$h_2 = h_{14} =$ H'02EA'

$h_3 = h_{13} =$ H'05B8'

$h_4 = h_{12} =$ H'04DF'

$h_5 = h_{11} =$ H'FC87'

$h_6 = h_{10} =$ H'EDD4'

$h_7 = h_9 =$ H'DF70'

$h_8 =$ H'5970'

The theoretical frequency response of this filter is shown in Figure 20.9. The result obtained using the bit-slice microcomputer is shown in Figure 20.10.

20.5.4 Band Pass Filter with N = 15 (Optimum Design Algorithm)

This type of filter was designed for the following specifications :-

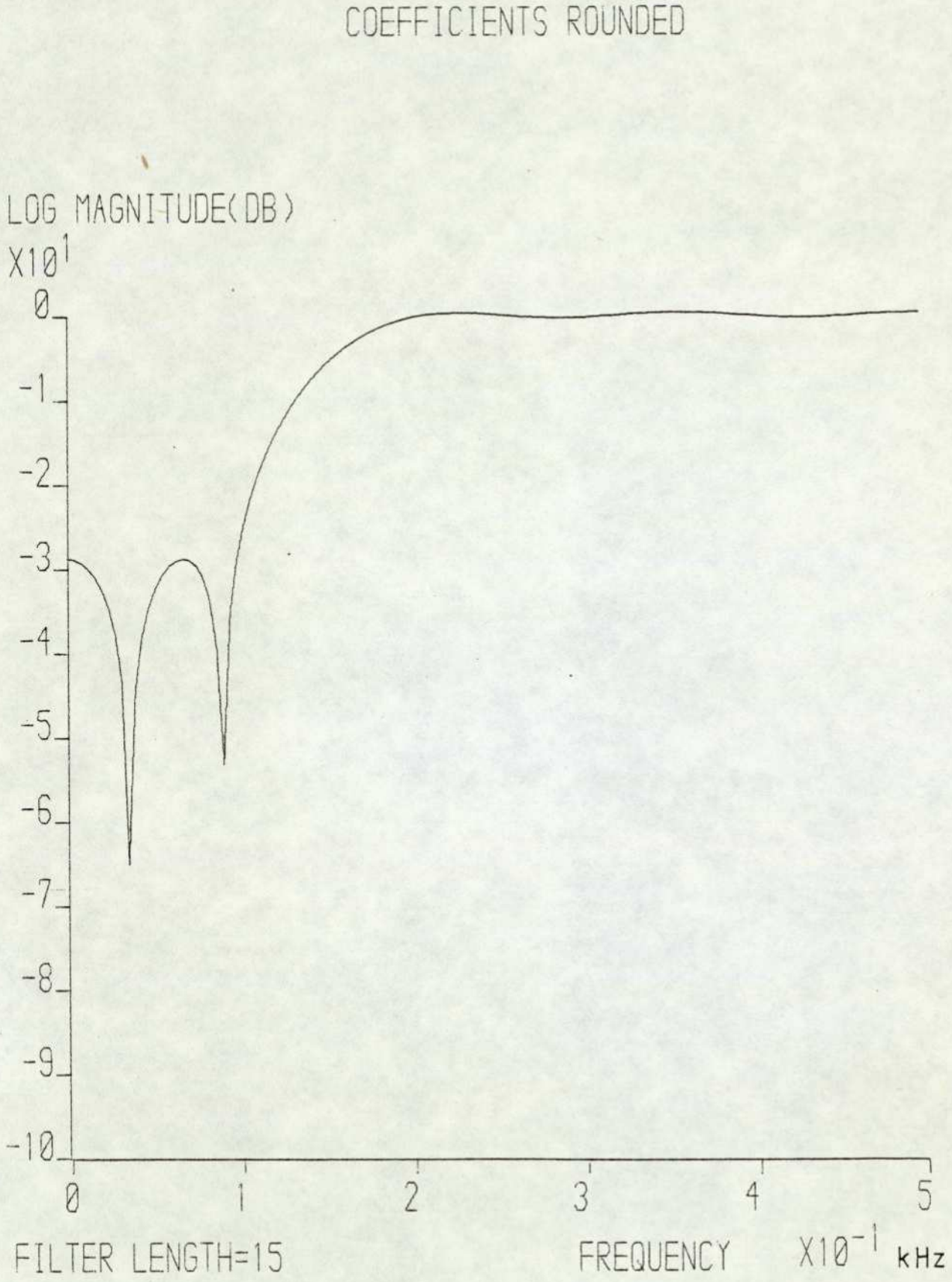


Figure 20.9 Theoretical frequency response of N=15 High Pass FIR Digital Filter, designed with the Optimal design Algorithm.

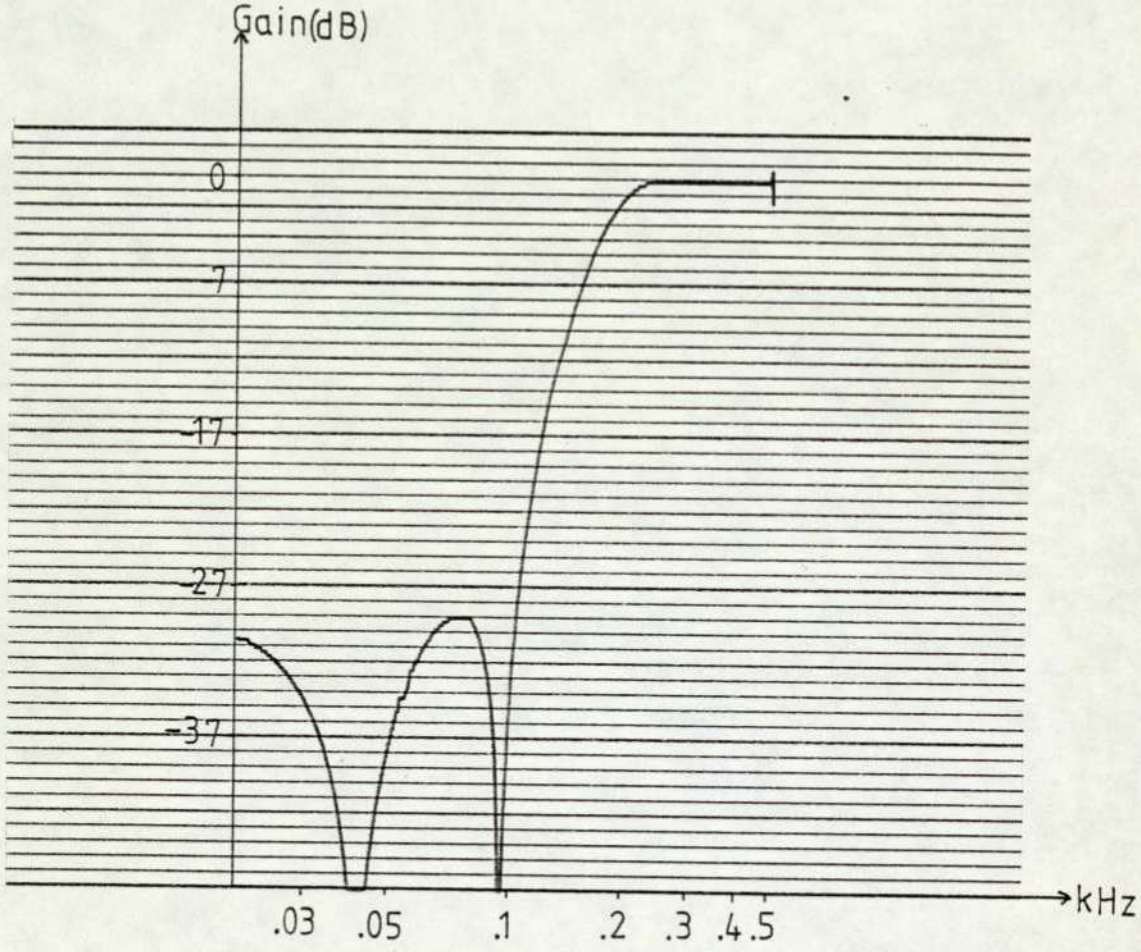


Figure 20.10 Frequency response obtained using the bit-slice microcomputer.

Sampling frequency = 1 kHz
First stop-band frequency = 0 to 50 Hz
Pass-band frequency = 100 to 300 Hz
Second stop-band frequency= 350 to 500 Hz

The coefficients of the filter were found to be :-

$h_1 = h_{15} = 0.74138168X10^{-1}$
$h_2 = h_{14} = -0.55595872X10^{-1}$
$h_3 = h_{13} = -0.84051212X10^{-1}$
$h_4 = h_{12} = -0.81631982X10^{-2}$
$h_5 = h_{11} = -0.12632789$
$h_6 = h_{10} = -0.25218972$
$h_7 = h_9 = 0.13624093$
$h_8 = 0.49357522$

Representing the coefficients in 16 bit fractional form, we have :-

$h_1 = h_{15} = H'097D'$
$h_2 = h_{14} = H'F8E2'$
$h_3 = h_{13} = H'F53E'$
$h_4 = h_{12} = H'FEF5'$
$h_5 = h_{11} = H'EFD4'$
$h_6 = h_{10} = H'DFB8'$
$h_7 = h_9 = H'1170'$
$h_8 = H'3F2D'$

The frequency response of this filter is shown in Figures 20.11 and 20.12.

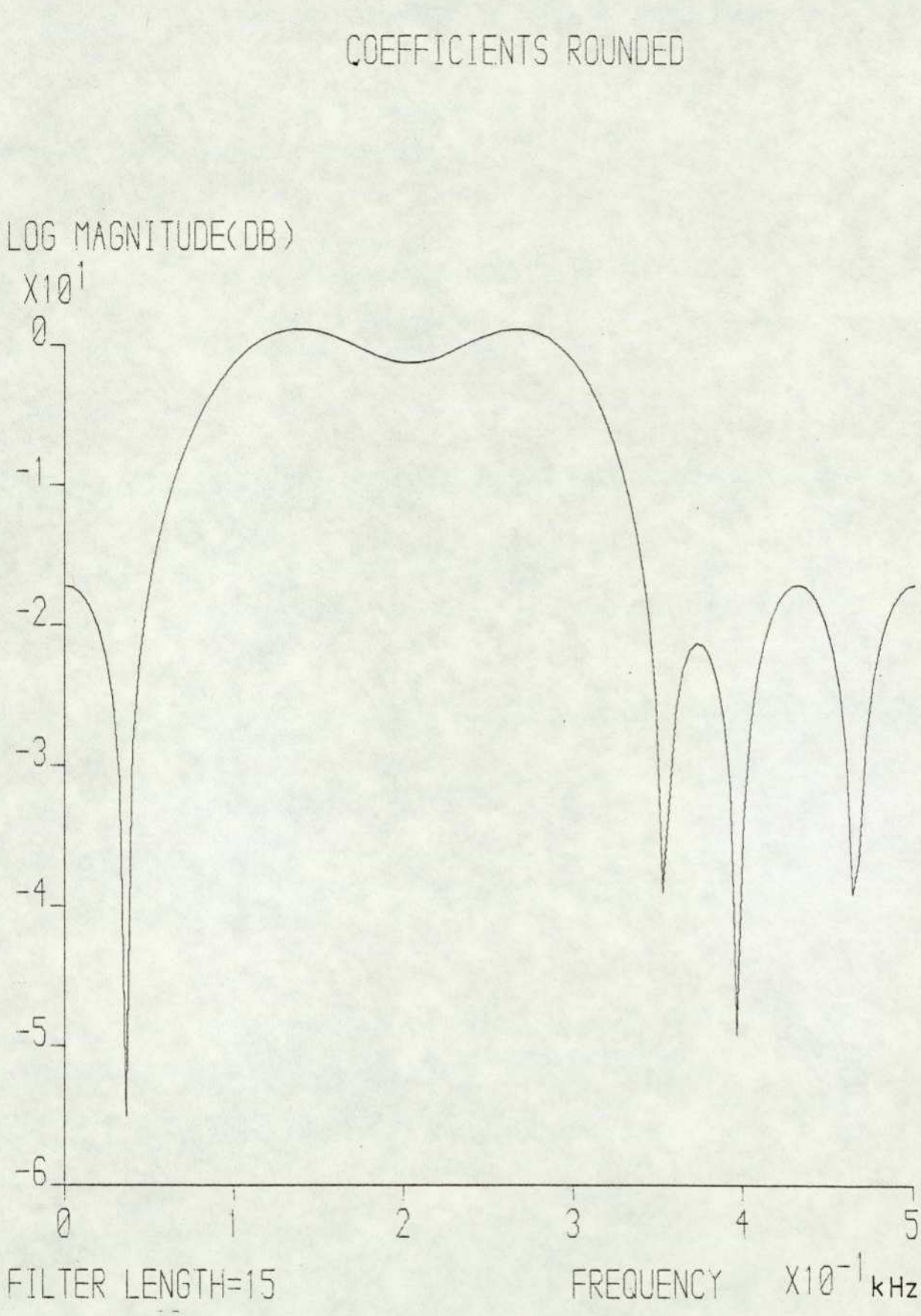


Figure 20.11 Theoretical frequency response of N=15 Band Pass FIR Digital Filter, designed with the Optimal design Algorithm.

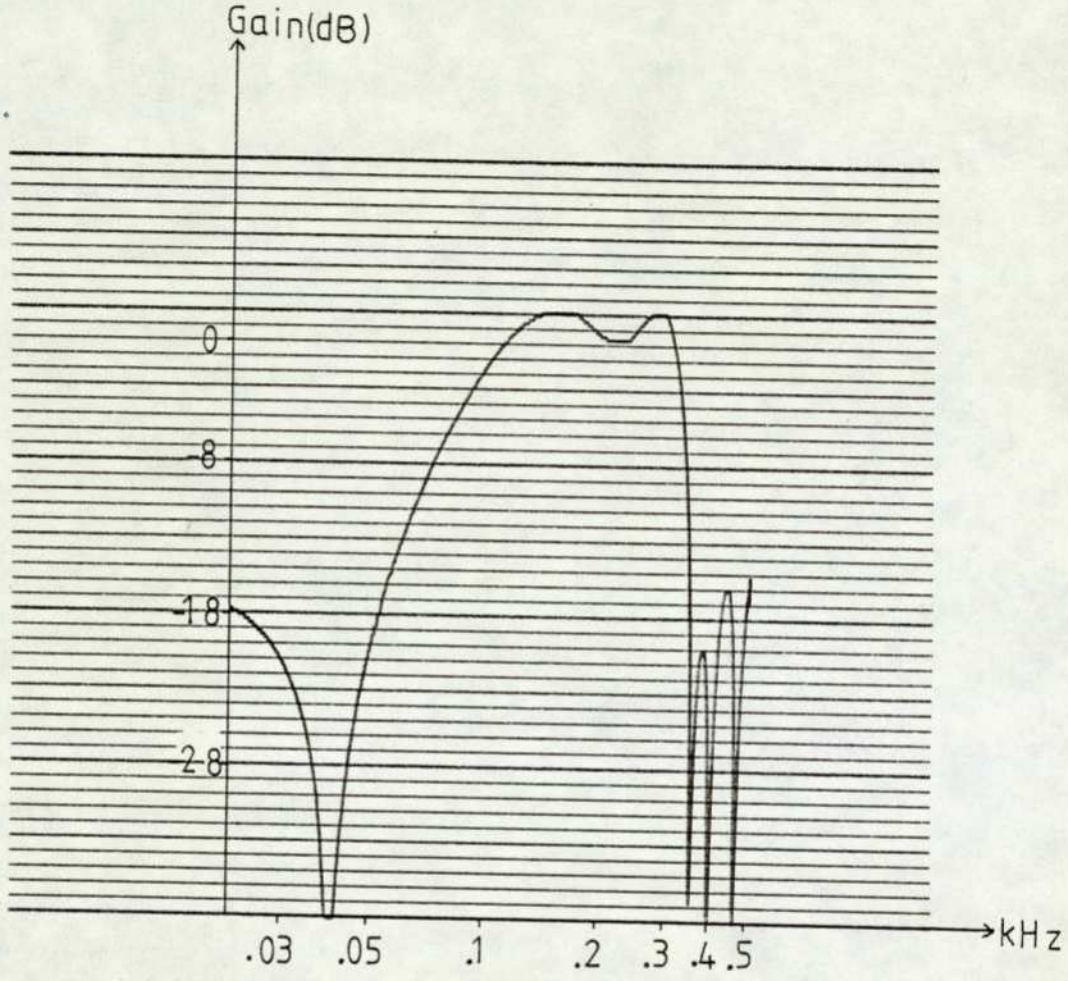


Figure 20.12 Frequency response obtained using the bit-slice microcomputer.

20.5.5 Band Stop Filter with N = 15 (Optimum Design Algorithm)

This kind of filter was designed for the following specifications :-

Sampling frequency = 1 kHz
First pass-band frequency = 0 to 100 Hz
Stop-band frequency = 150 to 350 Hz
Second pass-band frequency = 420 to 500 Hz

The coefficients of the filter were found to be :-

$h_1 = h_{15} = -0.43434413X10^{-4}$
$h_2 = h_{14} = -0.11961523$
$h_3 = h_{13} = -0.57997400X10^{-5}$
$h_4 = h_{12} = 0.60444560X10^{-4}$
$h_5 = h_{11} = -0.80381622X10^{-5}$
$h_6 = h_{10} = 0.31309696$
$h_7 = h_9 = 0.57272297X10^{-4}$
$h_8 = 0.50002976$

Representing the coefficients in 16 bit fractional form, we have :-

$h_1 = h_{15} =$ H'FFFF'
$h_2 = h_{14} =$ H'F080'
$h_3 = h_{13} =$ H'FFFF'
$h_4 = h_{12} =$ H'0002'
$h_5 = h_{11} =$ H'FFFF'
$h_6 = h_{10} =$ H'2814'
$h_7 = h_9 =$ H'0002'
$h_8 =$ H'4001'

The frequency response of this filter is shown in Figures 20.13 and 20.14.

20.6 Microcode of the Instruction 'FIR'

The instruction 'FIR' was implemented in the microprogram RAM with a total of 30 microinstructions. The microprogram data for this instruction is given in Figure 20.15. The details of the microoperations are as follows:-

Macroinstruction		Microoperations Performed
FIR		Main memory READ request
		Port Address $\longrightarrow$ Port Bus
		I/O Input request
		(Data Bus) $\longrightarrow R_4$
		$(R_4) \longrightarrow ((DCF))$
		Call to microsubroutine MULT
		$(R_0) \longrightarrow R_3$
	MORE :	Enable the REL of the APU
		(DCF) - (Aux. Reg. F)
		Jump on non-zero to MIC
		$(DCF) + (R_7) \longrightarrow DCF$
	MIC :	$(DCF) - 1 \longrightarrow DCF$
		$(DCE) + 1 \longrightarrow DCE$
		Call to microsubroutine MULT
		$(R_0) + (R_3) \longrightarrow R_3$
		$(R_6) - 1 \longrightarrow R_6$
		Jump on non-zero to MORE
		Main memory READ request
		Enable the instruction reg.
		Port Address $\longrightarrow$ Port Bus

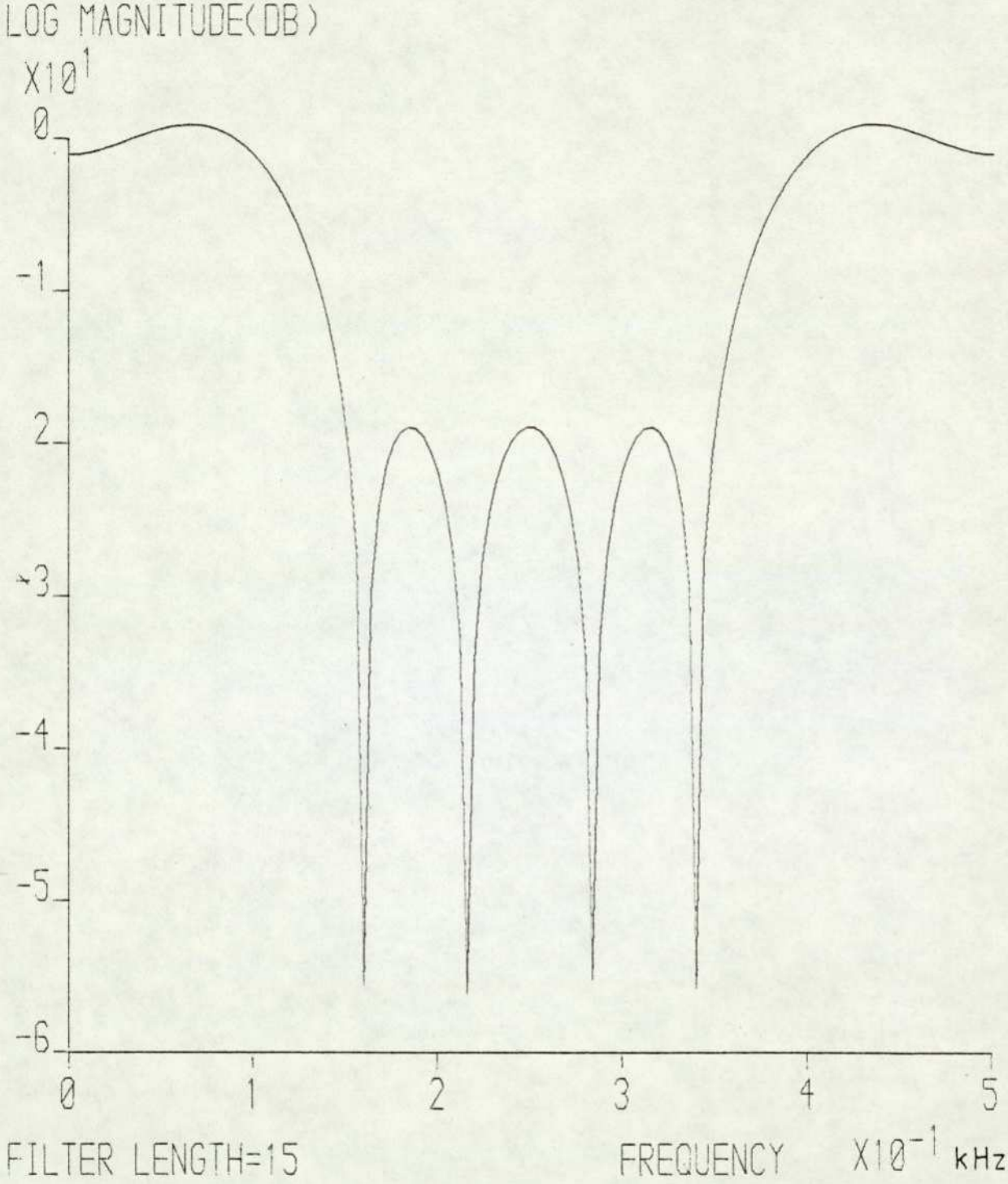


Figure 20.13 Theoretical frequency response of N=15 Band Stop FIR Digital Filter, designed with the Optimal design Algorithm.

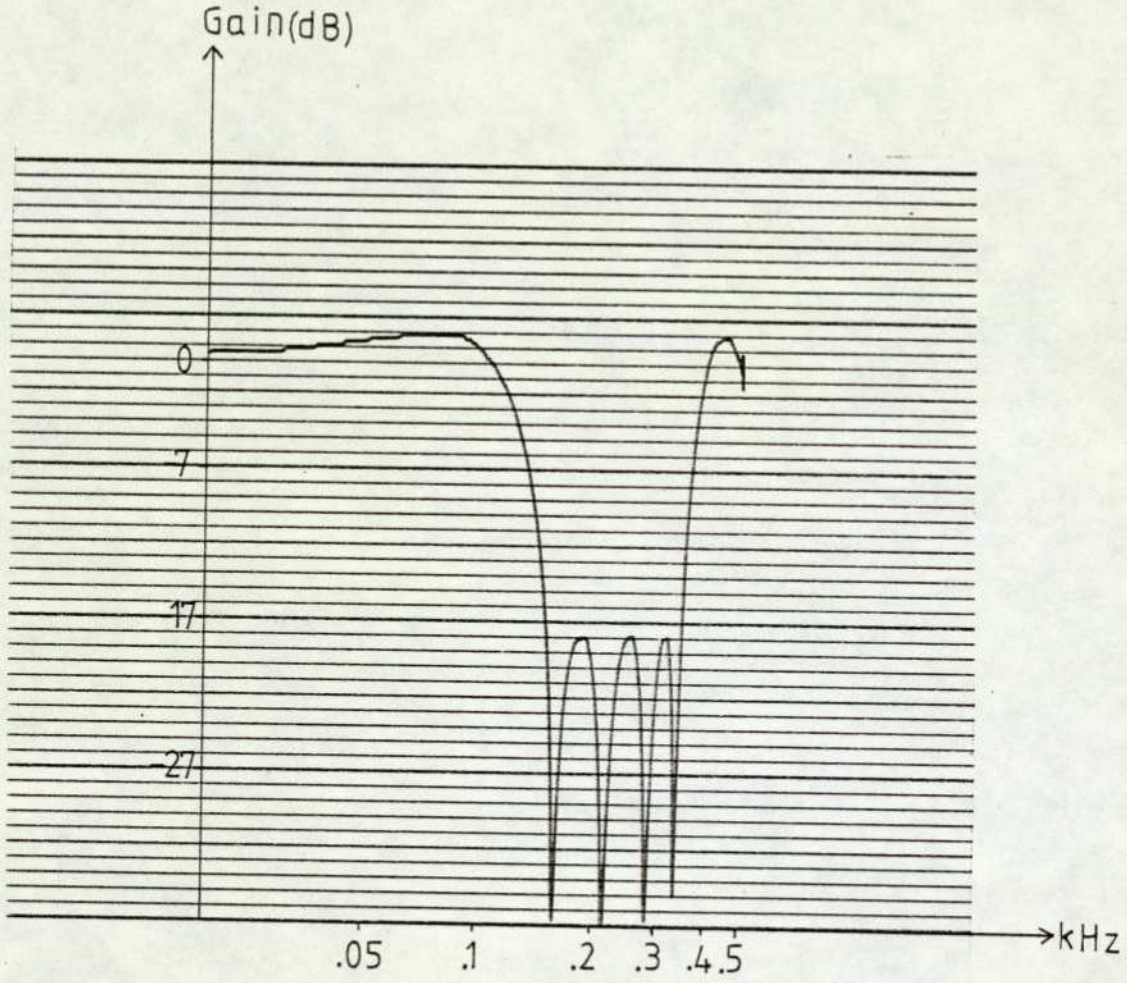


Figure 20.14 Frequency response obtained using the bit-slice microcomputer.

```
                    I/O Output request
                    (R3) ⟶ Data Bus
                    Enable the REL of the APU
                    Aux. Reg. E ⟶ DCE
                    Enable the REL of the APU
                    Aux. Reg. 6 ⟶ R6
                    Enable interrupts
WAIT :              if interrupt detected goto
                       interrupt service routine
                       else goto WAIT

MULT :              ((DCF)) ⟶ R2,Q
                    ((DCE)) ⟶ R1
                    (R1) X (R2) ⟶ R0,Q
                    Two's comp. correction
                    Shift R0 left by one position
                      for fractional multiplication
                      (the MSB of Q shifted to LSB
                       of R0)
                    Truncate the product (hence
                      product in R0)
                    Return
```

20.7 The Maximum Sampling Frequency

The maximum sampling frequency for a given length of filter is determined by the minimum time required to execute the instruction 'FIR'.

The microinstructions from 'LOOP' to 'MORE' (see Section 20.4) are executed in 29 cycles. Also the microinstructions from 'MORE' to the output section are executed in 32X(N-1) cycles. The output section (including the interrupt service

INSTRUCTION	OP-CODE	ADDRESS	FIELD 5								FIELD 4								FIELD 3								FIELD 2								FIELD 1								FIELD 0							
			ICU			REL	PCU	CCS			Am 2910					Inst. Reg.	DSL								PCU					Branch Address								ICU			ALU			Operand Select	
			A_3	MRQ	WAIT	FE	PCE	ST_1	ST_2	IOR	I_3	I_2	I_1	I_0	EDB	IRE	S_4	S_5	C_8	C_4	C_5	C_6	S_3	S_2	S_1	C_3	IOM	MMD	P/P	PE	P_4	P_3	P_2	P_1	BR_{11}	BR_{10}	BR_9	BR_8	BR_7	BR_6	BR_5	BR_4	BR_3	BR_2	BR_1	BR_0	A_2	A_1	A_0				RS_1	RS_2
																	RS_3		CR_{23}	CR_{22}	CR_{21}	CR_{20}	CR_{13}, E_4	CR_{12}, Pol	CR_{11}	CR_{10}																										
																	DSL		CCS, Carry-in																Dest.				Function				Source									
			47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FIR	80	100	1	0	1	1	0	1	1	1	1	1	1	0	1	1	X	X	X	X	X	X	X	0	X	X	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
	.	101	1	1	1	1	1	1	1	0	1	1	1	0	1	1	X	0	X	X	X	X	1	X	X	X	0	1	0	0	0	0	X	X	0	1	0	0	1	1	1	1	0	1	1	0	0	1	0	1
	.	102	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	0	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	0	0	1
	.	103	1	0	1	1	0	1	1	1	1	1	1	0	0	1	X	1	X	X	X	X	X	1	X	1	0	1	1	0	0	1	0	0	0	1	0	0	1	1	0	0	0	1	0	0	0	1	1	0
Call Sub. MULT	.	104	1	1	1	1	1	1	1	1	0	0	0	1	1	1	X	X	X	1	1	1	X	X	X	0	X	X	X	X	0	0	0	1	0	0	0	1	0	1	0	1	X	X	X	1	1	1	X	X
$R_0 \rightarrow R_3$	.	105	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	1	X	X	X	0	0	1	X	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	1	0	1
MORE	.	106	1	1	1	0	1	0	0	1	1	1	1	0	1	1	X	1	0	X	X	X	X	X	1	0	1	1	1	1	X	X	X	X	0	1	0	0	0	0	1	0	0	0	1	1	0	1	1	0
JNZ MIC	.	107	1	1	1	1	1	1	1	1	0	0	1	1	1	1	X	X	X	0	0	1	X	X	X	0	X	X	X	X	0	0	0	1	0	0	0	0	1	0	0	1	X	X	X	1	1	1	X	X
DCF+R7 → DCF	.	108	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	1	X	X	X	0	0	1	X	1	1	1	1	0	1	1	1	0	1	0	0	0	0	1	1	0	0	0	0	0	1	0	1
MIC	.	109	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	1	X	X	X	0	0	1	X	1	1	1	1	0	0	X	X	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1
DCE+1 → DCE	.	10A	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	0	X	X	X	X	X	X	0	1	1	1	0	X	X	X	X	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	1
Call Sub. MULT	.	10B	1	1	1	1	1	1	1	1	0	0	0	1	1	1	X	X	X	1	1	1	X	X	X	0	X	X	X	X	0	0	0	1	0	0	0	1	0	1	0	1	X	X	X	1	1	1	X	X
$R_0 + R_3 \rightarrow R_3$	.	10C	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	1	X	X	X	0	0	1	X	0	0	1	1	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	1	0	1
$R_6 - 1 \rightarrow R_6$	.	10D	1	1	1	1	1	0	0	1	1	1	1	0	1	1	X	0	1	X	X	X	0	0	1	X	0	1	1	0	0	0	X	X	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1
JNZ MORE	.	10E	1	1	1	1	1	1	1	1	0	0	1	1	1	1	X	X	X	0	0	1	X	X	X	0	X	X	X	X	0	0	0	1	0	0	0	0	0	1	1	0	X	X	X	1	1	1	X	X
	.	10F	1	0	1	1	0	1	1	1	1	1	1	0	1	0	X	X	X	X	X	X	0	X	X	0	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X

Figure 20.15 Microcode of the instruction 'FIR' .

cont....

INSTRUCTION	OP-CODE	ADDRESS	FIELD 5								FIELD 4								FIELD 3									FIELD 2								FIELD 1								FIELD 0							
			ICU			REL	PCU	CCS			Am 2910					Inst. Reg.	DSL			IOM	MMD	P/P	PCU					Branch Address														ICU			ALU				Operand Select	
			A_3	MRQ	WAIT	FE	PCE	ST_1	ST_2	IOR	I_3	I_2	I_1	I_0	EDB	IRE	S_4	S_5	RS_3	C_8	C_4	C_5	C_6	S_3	S_2	S_1	C_3	PE	P_4	P_3	P_2	P_1	BR_{11}	BR_{10}	BR_9	BR_8	BR_7	BR_6	BR_5	BR_4	BR_3	BR_2	BR_1	BR_0	A_2	A_1	A_0				RS_1	RS_2
																	CCS, Carry-in									CR_{23}	CR_{22}	CR_{21}	CR_{20}	CR_{13}, E_4	CR_{12}, Pol	CR_{11}	CR_{10}	Dest.				Function				Source			$\overline{IEN}$	$\overline{OE_Y}$	EB		
			47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	.	1,10	1	1	1	1	1	1	1	0	1	1	1	0	0	1	X	0	X	X	X	X	0	X	X	X	0	0	1	1	0	0	X	X	0	1	0	0	1	1	1	1	0	1	0	0	0	1	0	1
Pop DCE	.	1,11	1	1	1	0	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	1	X	X	X	X	X	0	0	X	X	0	1	0	0	1	1	1	1	0	1	1	0	0	1	1	1
Pop R_6	.	1,12	1	1	1	0	1	1	1	1	1	1	1	0	1	1	X	1	X	X	X	X	X	X	1	X	0	1	1	0	0	0	X	X	0	1	0	0	1	1	1	1	0	1	1	0	0	1	1	1
EI	.	1,13	0	1	1	1	1	1	1	1	1	1	1	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	1	1	1	X	X
WAIT	8,1	1,14	1	1	1	1	1	1	1	1	0	0	1	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	X	X
SUB. MULT	.	1,15	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	0	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	0	0	1
((DCF))→R_2,Q	.	1,16	1	0	1	1	0	1	1	1	1	1	1	0	1	1	X	1	X	X	X	X	X	0	X	1	0	1	1	0	0	0	1	0	0	1	1	1	1	1	1	1	0	1	1	0	0	1	0	0
	.	1,17	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	0	1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	0	0	1
((DCE))→R_1	.	1,18	1	0	1	1	0	1	1	1	1	1	1	0	1	1	X	1	X	X	X	X	X	0	X	1	0	1	1	0	0	0	0	1	0	1	0	0	1	1	1	1	0	1	1	0	0	1	0	0
0→R_0	.	1,19	1	1	1	1	1	1	1	1	1	1	1	0	1	1	X	0	X	X	X	X	X	X	X	X	0	0	0	0	0	0	X	X	0	1	0	0	1	1	0	0	1	1	X	0	0	1	0	1
ld cntr.	.	1,1A	1	1	1	1	1	1	1	1	0	1	0	0	1	1	X	X	X	1	1	1	X	X	X	0	X	X	X	X	0	0	0	0	0	0	0	0	1	1	1	0	X	X	X	1	1	1	X	X
Multiply	.	1,1B	1	1	1	1	1	1	1	1	1	0	0	0	1	1	0	1	1	X	X	X	0	0	1	X	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	X	0	0	0	1	1	0
Correction	.	1,1C	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1	X	X	X	1	1	0	X	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	X	0	0	0	1	1	0
Shift R_0, RETURN	.	1,1D	1	1	1	1	1	1	1	1	1	0	1	0	1	1	0	1	X	1	1	1	X	X	X	0	X	X	X	X	0	0	0	0	1	0	1	1	1	1	1	1	0	1	0	0	0	1	1	0
	.																																																	
	.																																																	

Figure 20.15 cont.

routine in the microprogram memory) is executed in 10 cycles. Therefore, the minimum time required to execute the instruction 'FIR' is given by :-

$$t_{min.} = 39 + 32(N-1) \quad \text{cycles}$$

or, $$t_{min.} = 32N + 7 \quad \text{cycles}$$

The maximum sampling frequency is therefore given by :-

$$f_{s\ max.} = \frac{1000}{32N + 7} \quad \text{kHz} \quad \text{(with 1 MHz clock)}$$

Table 20.1 gives a list of $f_{s\ max.}$ for filters up to N = 50.

20.8 Conclusions

Chapter 20 has described a novel technique for the implementation of real-time FIR digital filters, using the bit-slice microcomputer designed. The technique developed has the advantage that there is no need to perform any register transfer operations and thus a higher speed is obtained.

Several design examples have been considered, including band-pass and band-stop digital filters. A formula is derived which relates the maximum sampling frequency to the length of the filter. This formula gives a good indication to

Length of Filter N	Max. Sampling Frequency(kHz) $f_{s\ max.}$
5	5.9880
10	3.0581
15	2.0534
20	1.5456
25	1.2391
30	1.0341
35	0.8873
40	0.7770
45	0.6911
50	0.6223

Table 20.1 Maximum sampling frequency for a given length of FIR digital filter (with a clock frequency of 1 MHz) .

potential users whether or not to use a system of this kind for real-time FIR digital filtering.

CHAPTER 21

21. CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

21.1 Conclusions

(i) Although the bit-slice processors offer many advantages, the design of a bit-slice microcomputer is a very complex task. It requires the connection of several bit-slice devices and a large number of MSI and SSI supporting devices in an orderly manner.

To simplify the design of a bit-slice processor, several manufacturers in this field are now offering the basic MSI supporting devices in single LSI packages. Examples of these are:-

1. Am2904 : This chip integrates the condition code logic, status logic, carry-in logic and the data shifting logic of the CPU in a single LSI package.

2. Am29774 : This chip integrates 8-bits by 512 words of microprogram memory (PROM) and 8-bits of pipeline register in a single LSI package.

3. Am29116 : Although this chip is not bit-slice, it combines 4 bit-slice RALU elements (total 16 bits) and the associated MSI logic in a single LSI package.

Following the above discussion it is clear that the chip count of the bit-slice processors will decrease considerably as more and more parts of the processor elements are combined in a single chip. This will ease the design of bit-slice processors and make them more popular with the computer design engineers.

The design features presented in this thesis are an invaluable source of reference for those engaged with the design of bit-slice processors.

(ii) The Arithmetic Processing Unit (APU) of the bit-slice microcomputer is designed such that although the data length of the APU is 16-bits wide, the system can be microprogrammed to handle a data length of 8-bits. During 8-bits of operation only the higher byte of the Data Bus is used. Although this facility allows the 8-bit fixed instruction microprocessors to be emulated, it created difficulties in several stages of the design. Additional hardware was required at several subsystems of the microcomputer so that during 8-bits of operation the correct data could be transferred between the various points of the system.

The ALU contains 16 RAM registers and a 'Q' register. 16 additional high-speed registers (the auxiliary registers) are connected to the system. These additional registers proved to be useful during digital filtering applications.

(iii) The input-output ports of the microcomputer are designed by using bipolar MSI logic elements. This provided a low cost, high-speed interface capability to the microcomputer. Port 1 is connected as a 12-bit data acquisition channel, incorporating a sample/hold amplifier, A/D converter and a D/A converter. The data acquisition channel provided very clean waveforms during the real-time digital filtering applications.

(iv) The Program Control Unit (PCU) is designed by using four low cost microprogram sequencers. Although the PCU is simple, it provides all the necessary program control functions, such as subroutine call and returns, conditional and unconditional program jumps etc.

(v) Only a 1 K of user memory (RAM) is connected to the bit-slice microcomputer. Since special instructions can be implemented in the microprogram memory, a main memory of this size is adequate for most of the signal processing applications. The main memory can however be expanded up to 64 K if necessary.

(vi) The Interrupt Control Unit (ICU) provides 7 external and one internal (Timer) interrupt facilities to the bit-slice microcomputer. All interrupts are vectored and each has a different priority level. The ICU was very useful in real-time digital filtering applications. It was used to synchronize the bit-slice microcomputer with the external sampling rate.

(vii) The Timing Unit provides the system clock. A special facility included here is the ability to disable the system clock and provide single external clock pulses. This feature is very useful in developing microinstructions where the execution steps of each microinstruction can be analysed by executing them with a single clock pulse.

(viii) A novel technique is employed in the design of the control unit. The microprogram memory is designed from a combination of PROM and RAM type of memories. Some of the PROM part is loaded with microinstructions which implement a 16-bit fixed instruction set, with 62 general purpose instructions. Thus, the user can use the microcomputer as a fixed instruction set machine if he does not wish to generate his own instructions. Also, a user developing his own microcode gains a better understanding of microprogramming the microcomputer by analysing the microcode already in the PROM part of the microprogram memory.

The RAM part of the microprogram memory can be loaded with the user defined microinstructions. Here, a low cost 8-bit microprocessor is used as a support processor. The support processor accepts the user microinstructions from the user terminal and loads the RAM part of the microprogram memory. The support processor provides many additional features to the user. For example, the user microinstructions can be stored on a paper-tape for future use.

As a general purpose microprogram design/test method, the technique developed here may not be very attractive to the large well established companies in the computer industry. There are however many small companies and research centres with limited resources in which the designers face the problems of developing microinstructions for their microprogrammable machines. To designers working in such circumstances the technique described can offer a flexible design tool at an acceptable cost.

(ix) An operating system (monitor) is developed (in PROM) by using the fixed instructions of the bit-slice microcomputer. The operating system enables the users to enter, modify, display, and execute their programs in the main memory. The operating system is also a very useful feature in developing microinstructions. The user can develop his own microinstructions in the RAM part of the microprogram memory. He can then utilize the operating system and write simple programs in the main memory to test his microinstructions.

(x) Special instructions are invented for the implementation of real-time IIR digital filters on the bit-slice microcomputer. It is shown that the bit-slice approach provides both a higher speed and a simplicity of design. A complete second order IIR filter section was microcoded and it was implemented with a single instruction. Cascading to higher order filters was done simply by repeating the basic filter instruction the required number of times. Special instructions are invented

for both the high-pass and the low-pass IIR digital filters.

The design of the real-time FIR digital filters are investigated by deriving a novel algorithm, which gave rise to a higher bandwidth and a simpler design. The complete FIR filter algorithm was implemented with a single instruction, where the only external data required were the order and the coefficients of the filter.

Equations are derived which relate the maximum sampling frequency to the order of the filters, which give a good indication to potential users whether or not to use a system of this kind for real-time digital filtering.

(xi) Two manuals are prepared with the aim of providing the user the necessary information in using the fixed instruction set of the bit-slice microcomputer.

21.2 Suggestions for Future Work

(i) At present the bit-slice microcomputer operates at a clock frequency of 1 MHz. As speed is one of the major advantages of the bit-slice processors, it may be desirable to increase the speed of the system. The speed can in theory be increased by a factor of 5 by replacing the slow components in the critical data and control paths with their faster versions. The basic components to be replaced are the main memory and the RAM part of the microprogram memory (these parts of the system are designed using MOS memories in order to keep the costs down).

(ii) In high-speed real-time digital signal processing applications, it is required to perform multiplication in the minimum possible time. Unfortunately, the multiplication time of the Am2903 is rather slow. It is however possible to connect an external high-speed multiplier chip (e.g. the TRW series of multipliers) to the system which can improve the multiplication time by over a factor of 10.

(iii) At present the microprogramming process is carried out in binary representation which is essentially low level. A future work in this field may be to develop a high-level microassembler in order to simplify and speed-up the process of microprogramming.

(iv) Although mnemonics are defined for the fixed instructions of the microcomputer, there are no macroassemblers

available to help the user in developing software. A macroassembler may be designed (possibly on a time-sharing system) to interpret the fixed instructions of the bit-slice microcomputer.

(v) The digital filter microinstructions developed by the author can be made non-volatile by loading them into a PROM in the microprogram memory. The microcomputer can then be used as a machine specialized in real-time digital filtering applications.

(vi) The design of other types of IIR digital filters (e.g. elliptic and Chebyshev II) and other aspects of real-time signal processing can be investigated by developing microinstructions.

22. REFERENCES

1. Wilkes, M.V "The best way to design an automatic calculation machine" , Manchester University Computer Inaugural Conference, pp. 16-18 (1951)

2. Wilkes, M.V and Stringer, J.B "Microprogramming and the design of the control circuits in an electronic digital computer" , Proc. Comb. Phil. Soc. 49, pp. 230-238 (1953)

3. Tucker, S.G "Microprogram control for system/360" , IBM Systems Journal, Vol. 6, No. 4, pp. 222 (1967)

4. Husson, S.S "Microprogramming manual for IBM system 360 model 50" , IBM Technical Report TR-00.479-1

5. "Microprocessor encylopedia volume II: Bit-slice", SYBEX, 2161 Shattuck Ave. , Berkeley, California (1977)

6. Adams, W.T and Smith, S.M "How bit-slice families compare: Part 2, sizing up the microcontrollers", Electronics, pp. 96-102 (1978)

7. Davidson, S and Shriver, B.D "An overview of firmware engineering" , IEEE Computer, Vol. 11, No. 5, pp. 21-31 (1978)

8. "The Am2900 family data book" , Advanced Micro Devices Inc. Sunnyvale, California (1978)

9. "Burr-Brown general catalogue" , Burr-Brown Ltd., Tucson, Arizona

10. Intersil Ltd., Cupertino, California

11. Burton, D.P and Dexter, A.L "Microprocessor systems handbook" , Analog Devices Inc., Norwood, Massachusetts

12. Ibrahim, D "Design of a microprogrammable computer with bit-slice devices" , Microprocessors and Microsystems, Vol. 4, No. 2, pp.57-62 (1980)

13. "Am2910 microprogram controller" , Advanced Micro Devices Inc., Sunnyvale, California (1978)

14. Flynn, M.J "Microprogramming-another look at internal computer control" , Proc. IEEE, Vol. 63, No. 11 pp. 1554-1568 (1975)

15. "PDP 11 handbook" , Digital Equipment Corporation (1969)

16. Kraft, G.D and Toy, W.N "Mini/microcomputer hardware design" , Prentice Hall (1979)

17. "F-8 evaluation kit" , Mostek (1975)

18. Clymer, J.R.W "A computer control unit using the Am2909" Application note (1975), Advanced Micro Devices Inc., Sunnyvale, California

19. Powers, V.M and Hernandez, J.H "Microprogram assemblers for bit-slice microprocessors" , IEEE Computer, Vol. 11, No. 7, pp. 108-120 (1978)

20. "AMDASM/80 reference manual" , Advanced Micro Devices Inc., Sunnyvale, California (1977)

21. "F8 user's guide" , Fairchild Camera and Instrument Corporation (1976)

22. "MK 3871 peripheral input/output" , Mostek Corp. , Carrollton, Texas (1977)

23. Bozic, S.M "Digital and Kalman filtering" , Edward Arnold (Publishers) Ltd. (1979)

24. Cappellini, V; Constantinides, A.G and Emiliani, P "Digital filters and their applications" , Academic Press Inc., (1978)

25. "TI Programmable 58/59" , Texas Instruments Inc.

26. McClellan, J.H; Parks, T.W, and Rabiner, L.R "A computer program for designing optimum FIR linear phase digital filters" , IEEE Trans. Audio Electro-acoustics AU21, 6 , pp. 506-526 (1973)

27. Holbrook, J.G "Laplace transforms for electronic engineers" Pergamon Press Ltd. (1969)

28. Antoniou, A "Digital filters, analysis and design" , Mc. Graw-Hill (1979)

29. Al-Najar, M.A.S, "Real -time digital filtering using eight bit microprocessors" , M.Phil thesis, The City University, (1978)

23. BIBLIOGRAPHY

1. Kampe, T.W "The design of a general-purpose microprogram controlled computer with elementary structure" , IRE Trans., EC-9, pp. 208-213 (1960)

2. Tucker, S.G "Microprogram control for digital computers" , IBM Journal 6, pp.222 (1962)

3. Gasselli, A "The design of program-modifiable microprogrammed control units" , IRE Trans. on elec. comp., pp. 336-339 (1962)

4. Schlaeppi, H.P "Microprogrammed computer control" , IBM Tech. Disc. Bull., Vol. 5, No. 9, pp. 45-46 (1963)

5. Gerace, G.B "Microprogrammed control for computing systems" , IEEE Trans. on. elec. comp. pp. 733-747 (1963)

6. Vandling, G.C and Patzer, W.J "Systems implications of microprogramming" , Computer Design, pp. 62-66 (1967)

7. Morrison, G.E "Investigation of microprogramming as a technique in the control of digital computing systems" , Computing Reviews, Vol. 9, No. 1, pp.40-41 (1968)

8. Muething, G,F "Designing the maximum performance into bit-slice minicomputers" , Electronics (USA), Vol. 49 No. 20, pp. 91-96 (1976)

9. Colin, A "Intel 3000 and AM 2900 microprocessors-A comparison" , Microprocessors, Vol. 1, No. 5, pp.287-292, (1977)

10. Nemec, J and Lau, S.Y "Bipolar MPU'S An introduction to architecture and applications" , EDN(USA), Vol. 22, No.17, pp. 63-67 (1977)

11. Alexandridis, N.A "Bit-sliced microprocessor architecture" , IEEE Computer, Vol. 11, No. 6, pp. 56-80 (1978)

12. Conway, J.C "Hardware approaches to microprogramming with bipolar microprocessors" , Comput. Des. (USA), Vol.17, No.8, pp. 83-91 (1978)

13. Dimond, K.R; Martenstyn, H and Pearce, M.A "Microprogrammed interpreters for control" , Conference on microprocessors in automation and communications, 19-22 Sept., 1978, Canterbury, England

14. Hird, D.J and Elliot, D.M "Bit-slice microprocessors-their use and application in minicomputers" , Electron. and Power, Vol. 25, No. 3, pp. 179-184 (1979)

15. Lau, S.Y and Janak, M "To cut controller costs, replace random logic with microcoded bit slices" , Electron. Des. (USA), Vol. 27, No. 2, pp. 90-97 (1979)

16. Swartzlander, E.E "Microprogrammed control for specialized processors" , IEEE Trans. on Computers, Vol. C-28, No. 12, pp. 930-934 (1979)

17. Rader, C.M and Gold, B "Digital filter design techniques in the frequency domain" , Proc. IEEE, Vol. 55, No. 2, pp. 149-171 (1967)

18. Golden, R.M "Digital filter synthesis by sampled-data transformations" , IEEE Trans. on audio and electro-acoustics, Vol. AU-16, No. 3, pp. 321-329 (1968)

19. Helms, H.D "Nonrecursive digital filters" , IEEE Trans. on Audio and Electroacoustics, Vol. AU-16, No. 3, pp. 336-342 (1968)

20. Herrmann, O "Design of nonrecursive digital filters with linear phase", Electronics Letters, Vol.6, No. 11, pp. 328-329 (1970)

21. Brophy, F and Salazar, A.C "Recursive digital filter synthesis in the time domain" , IEEE Trans. Acoustics, Speech and Signal Processing Vol. ASSP-22, No. 1, pp. 45-55 (1974)

22. Allen, J "Computer architecture for signal processing" , IEEE Proceedings Vol. 63, No. 4, pp. 624-633 (1975)

24. PUBLICATIONS

Publications removed for copyright reasons (pp. 539-625)

Int. J. Elect. Enging. Educ., Vol. 16, Pp. 166-175. Manchester U.P., 1979.
A BASIS FOR LABORATORY WORK WITH BIT-SLICE MICROPROGRAMMABLE MICROPROCESSORS
A.C. DAVIES and D. IBRAHIM (pp. 339-548)

IEE Colloquium on Teaching Laboratories for Microprocessors. 11th January 1979.
AN INTERACTIVE-CONTROLLER FOR THE AMD 2900 LEARNING-KIT
A.C. Davies and D. Ibrahim (pp. 549-552)

Microprocessors and microsystems, Vol. 4, no. 2, pp. 57-62. March 1980.
Design of a microprogrammable computer with bit-slice devices
D. IBRAHIM (pp. 553-558)

Microprocessors and microsystems, Vol. 4, no. 3, pp. 95-100. April 1980.
Design of a 16-bit microprogrammable computer with bit-slice devices
D. IBRAHIM (pp. 559-564)

Electronic Engineering. May 1980.
Designing digital sequence controllers with microprogramming techniques
D. IBRAHIM (pp. 565-570)

Paper submitted for publication in "Microprocessors and Microsystems"
A MICROPROGRAMMED APPROACH TO DIGITAL FILTER DESIGN
D. IBRAHIM (pp. 571-593)

Paper submitted for publication in "Computers and Digital Techniques"
A 16 BIT MICROPROGRAMMABLE MICROCOMPUTER WITH WRITEABLE CONTROL STORE
D. IBRAHIM (pp. 594-619)

To be presented in IEE Colloquium on "Development of Bit-Slice Systems". 2 October 1980
DESIGN FEATURES OF A USER-MICROPROGRAMMABLE MICRGCOMPUTER
D.Ibrahim and A.C.Davies (pp. 620-625)