A BIT-SLICE USER MICROPROGRAMMABLE MICROCOMPUTER :

DESIGN AND APPLICATIONS TO REAL-TIME DIGITAL FILTERING

Volume 2

-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

APPENDIX A

UMM-16 OPERATING SYSTEM SOFTWARE LISTING

```
*
* UMM-16 BIT-SLICE MICROCOMPUTER
*
* SOFTWARE LISTING OF THE UMM-16 OPERATING SYSTEM
*
* ******************************************************
* PROGRAMMER : D. IBRAHIM
*              THE CITY UNIVERSITY, 1980
* ******************************************************
*
* THE PROGRAM IS STORED IN TWO 512 WORD BY 8-BIT N82S115
* TYPE BIPOLAR PROMS
*
* SCRATCHPAD REGISTER D IS USED DURING THE TELETYPE/VDU
* INPUT AND OUTPUT ROUTINES
*
*
* SEE UMM-16 USER'S MANUAL FOR DETAILS OF THE COMMANDS AND
* THE TELETYPE/VDU INPUT AND OUTPUT SUBROUTINES AVAILABLE
*
*
*
*
*
0000  3800                  NOP
0001  2F00       BEGIN      CALL CRLF
0002  004D
0003  15D0                  LI D, H'2400'
0004  2400
0005  2F00                  CALL TTYOUT2       PRINT '$'
0006  002E
0007  2F00                  CALL TTYIN2        ACCEPT A COMMAND
0008  0056
0009  2F00                  CALL TTYOUT2       ECHO THE COMMAND
000A  002E
000B  2F00                  CALL SPACE
000C  0046
000D  28D0                  JE D,H'4D00',MEM
000E  4D00
000F  00ED
```

```
0010  28D0                    JE D,H'5400',TYPE
0011  5400
0012  009C
0013  28D0                    JE D,H'4400',DUMP
0014  4400
0015  0118
0016  28D0                    JE D,H'4200',BREAK
0017  4200
0018  019D
0019  28D0                    JE D,H'4C00',LOAD
001A  4C00
001B  01D2
001C  28D0                    JE D,H'4500',EXECUTE
001D  4500
001E  0118
001F  28D0                    JE D,H'4300',C
0020  4300
0021  0200
0022  28D0                    JE D,H'5000',P
0023  5000
0024  0178
0025  28D0                    JE D,H'2400',BEGIN
0026  2400
0027  0001
0028  15D0                    LI D, H'3F00'
0029  3F00
002A  2F00       PAPER        CALL TTYOUT2          PRINT '?'
002B  002E
002C  2600                    JUMP BEGIN
002D  0001
*
* SUBROUTINE TTYOUT2
*
002E  1FC0       TTYOUT2      INS C,H'0700'
002F  0700
0030  07C0                    ANDI C,H'0100'
0031  0100
```

```
0032  23C0                    CI C,H'0100'
0033  0100
0034  2700                    JNZ TTYOUT2
0035  002E
0036  1ED0                    OUTS D,H'0600'
0037  0600
0038  3200                    RET
*
* END OF SUBROUTINE TTYOUT2
*
* SUBROUTINE TTYOUT1
*
0039  2F00       TTYOUT1      CALL TTYOUT2
003A  002E
*
* SUBROUTINE TTYOUT3
*
003B  33D0       TTYOUT3      ROTR D
003C  33D0                    ROTR D
003D  33D0                    ROTR D
003E  33D0                    ROTR D
003F  2F00                    CALL TTYOUT2
0040  002E
0041  33D0                    ROTR D
0042  33D0                    ROTR D
0043  33D0                    ROTR D
0044  33D0                    ROTR D
0045  3200                    RET
*
* SUBROUTINE TO PRINT A 'SPACE'
*
0046  1ADD       SPACE        PUSH D,D
0047  15D0                    LI D, H'2000'
0048  2000
0049  2F00       BACK         CALL TTYOUT2
004A  002E
004B  1BDD                    POP D,D
004C  3200                    RET
*
* END OF SUBROUTINE
*
```

```
*
* SUBROUTINE TO PRINT A 'CARRIAGE RETURN'
*
004D  1ADD        CRLF          PUSH D,D
004E  15D0                      LI D, H'0A00'
004F  0A00
0050  2F00                      CALL TTYOUT2
0051  002E
0052  15D0                      LI D, H'0D00'
0053  0D00
0054  2600                      JUMP BACK
0055  0049
*
* SUBROUTINE TTYIN2
*
0056  1FC0        TTYIN2        INS C,H'0700'
0057  0700
0058  07C0                      ANDI C,H'0200'
0059  0200
005A  23C0                      CI C,H'0200'
005B  0200
005C  2700                      JNZ TTYIN2
005D  0056
005E  1FD0                      INS D,H'0600'
005F  0600
0060  07D0                      ANDI D,H'FF00'
0061  FF00
0062  3200                      RET
*
* SUBROUTINE TTYIN1
*
0063  2F00        TTYIN1        CALL TTYIN2
0064  0056
0065  17CD                      MOVR C,D
0066  1ACC                      PUSH C,C
0067  2F00                      CALL TTYIN2
0068  0056
0069  1BCC                      POP C,C
006A  33D0                      ROTR D
006B  33D0                      ROTR D
```

```
006C  33D0                    ROTR D
006D  33D0                    ROTR D
006E  0ADC                    ADDR D,C
006F  3200                    RET
*
* SUBROUTINE TTYIN3
*
0070  2F00       TTYIN3       CALL TTYIN2
0071  0056
0072  33D0                    ROTR D
0073  33D0                    ROTR D
0074  33D0                    ROTR D
0075  33D0                    ROTR D
0076  3200                    RET
*
* END OF SUBROUTINE
*
*
* SUBROUTINE CONVERT
*
*
0077  12A0       CONVERT      CLRR A
0078  2F00       CON1         CALL TTYIN3
0079  0070
007A  2F00                    CALL TTYOUT3
007B  003B
007C  2BD0                    JE D,H'0024',BEGIN
007D  0024
007E  0001
007F  2BD0                    JE D,H'002C',CON2
0080  002C
0081  009A
0082  2BD0                    JE D,H'0020',CON2
0083  0020
0084  009A
0085  2BD0                    JE D,H'000D', CON3
0086  000D
0087  0195
0088  2BD0                    JE D,H'000A',CON3
0089  000A
```

```
008A  0195
008B  23D0        SUB1          CI D,H'003A'
008C  003A
008D  29D0                      JP CON5
008E  0091
008F  13D0                      ADDI D,H'0007'
0090  0007
0091  14D0        CON5          SUBI D,H'0037'
0092  0037
0093  39A0                      SL1 A
0094  39A0                      SL1 A
0095  39A0                      SL1 A
0096  39A0                      SL1 A
0097  0AAD                      ADDR A,D
0098  2600                      JUMP CON1
0099  0078
009A  17DA        CON2          MOVR D,A
009B  3200                      RET
*
* END OF SUBROUTINE CONVERT
*
*
* COMMAND 'T'
*
009C  2F00        TYPE          CALL CONVERT
009D  0077
009E  179A                      MOVR 9,A
009F  2F00                      CALL CONVERT
00A0  0077
00A1  0CA9                      SUBR A,9
00A2  0DA0                      INCR A
00A3  17F9                      MOVR F,9
00A4  17D9        PEN           MOVR D,9
00A5  2F00                      CALL HEX1
00A6  008E
00A7  2F00                      CALL SPACE
00A8  0046
00A9  2F00                      CALL SPACE
00AA  0046
```

```
00AB  1500                    LI 0, H'0008'
00AC  0008
00AD  21D0        HERE        MOVM D              STORE IN MEMORY
00AE  2F00                    CALL HEX1
00AF  00BE
00B0  2F00                    CALL SPACE
00B1  0046
00B2  0FA0                    DECR A
00B3  2800                    JZ BEGIN
00B4  0001
00B5  0F00                    DECR 0
00B6  2700                    JNZ HERE
00B7  00AD
00B8  2F00                    CALL CRLF
00B9  004D
00BA  1390                    ADDI 9,H'0008'
00BB  0008
00BC  2600                    JUMP PEN
00BD  00A4
* END OF COMMAND 'T'
*
*
* SUBROUTINE HEX1
*
*
00BE  1ADD        HEX1        PUSH D,D
00BF  33D0                    ROTR D
00C0  33D0                    ROTR D
00C1  2F00                    CALL SUBROUTINE
00C2  00D5
00C3  1ADD        HEX2        PUSH D,D
00C4  2F00                    CALL SUBROUTINE
00C5  00D5
00C6  1ADD        HEX3        PUSH D,D
00C7  39D0                    SL1 D
00C8  39D0                    SL1 D
00C9  39D0                    SL1 D
00CA  39D0                    SL1 D
```

```
00CB  2F00                    CALL SUBROUTINE
00CC  00D5
00CD  1ADD      HEX4          PUSH D,D
00CE  33D0                    ROTR D
00CF  33D0                    ROTR D
00D0  33D0                    ROTR D
00D1  33D0                    ROTR D
00D2  2F00                    CALL SUBROUTINE
00D3  00D5
00D4  3200                    RET
00D5  07D0      SUBROUTINE    ANDI D,H'0F00'
00D6  0F00
00D7  23D0                    CI D,H'0A00'
00D8  0A00
00D9  2900                    JP THO
00DA  00DF
00DB  13D0                    ADDI D,H'3000'
00DC  3000
00DD  2600                    JUMP LOOM
00DE  00E1
00DF  13D0      THO           ADDI D,H'3700'
00E0  3700
00E1  1FC0      LOOM          INS C,H'0700'
00E2  0700
00E3  07C0                    ANDI C,H'0100'
00E4  0100
00E5  23C0                    CI C,H'0100'
00E6  0100
00E7  2700                    JNZ LOOM
00E8  00E1
00E9  1ED0                    OUTS D,H'0600'
00EA  0600
00EB  18DD                    POP D,D
00EC  3200                    RET
*
* COMMAND 'M'
*
00ED  2F00      MEM           CALL CONVERT
00EE  0077
00EF  2F00      FIN           CALL HEX1
```

```
00F0  008E
00F1  2F00                    CALL SPACE
00F2  0046
00F3  170D                    MOVR 0,D
00F4  17FD                    MOVR F,D
00F5  21D0                    MOVM D
00F6  2F00                    CALL HEX1
00F7  008E
00F8  2F00                    CALL SPACE
00F9  0046
00FA  2F00                    CALL TTYIN3
00FB  0070
00FC  2F00                    CALL TTYOUT3
00FD  003B
00FE  2BD0                    JE D,H'000D',MORE
00FF  000D
0100  0112
0101  2BD0                    JE D,H'000A',MORE
0102  000A
0103  0112
0104  2BD0                    JE D,H'0024',BEGIN
0105  0024
0106  0001
0107  2BD0                    JE D,H'003C',JUMP
0108  003C
0109  0199
010A  2F00                    CALL SUB1
010B  008B
010C  0FF0                    DECR F
010D  30D0                    LRM D
010E  0DD0                    INCR 0
010F  17D0                    MOVR D,0
0110  2600                    JUMP FIN
0111  00EF
0112  0DD0        MORE        INCR 0
0113  17D0                    MOVR D,0
0114  2F00        BOOK        CALL CRLF
0115  004D
0116  2600                    JUMP FIN
0117  00EF
```

```
0118  2F00      EXECUTE     CALL CONVERT
0119  0077
011A  31D0                LRP D
*
* COMMAND 'D'
*
011B  2F00      DUMP        CALL CONVERT
011C  0077
011D  17FA                MOVR F,A
011E  2F00                CALL CONVERT
011F  0077
0120  0CAF                SUBR A,F
0121  2F00                CALL PUNCH1
0122  0166
0123  2F00                CALL CRLF
0124  004D
0125  2F00                CALL PUNCH2
0126  016F
0127  2F00                CALL CRLF
0128  004D
0129  15D0                LI D, H'0053'
012A  0053
012B  2F00                CALL TTYOUT3
012C  003B
012D  0DA0                INCR A
012E  17DF                MOVR D,F
012F  2F00                CALL HEX1
0130  00BE
0131  2F00      DUMP4       CALL CRLF
0132  004D
0133  1500                LI 0, H'0008'
0134  0008
0135  15D0                LI D, H'0058'
0136  0058
0137  1210                CLRR 1
0138  2F00                CALL TTYOUT3
0139  003B
013A  21D0      DUMP3       MOVM D
013B  2F00                CALL HEX1
013C  00BE
```

```
013D  2F00                    CALL SPACE
013E  0046
013F  2F00                    CALL CHECKSUM
0140  0175
0141  0FA0                    DECR A
0142  2800                    JZ OUT
0143  014C
0144  0F00                    DECR 0
0145  2700                    JNZ DUMP3
0146  013A
0147  17D1                    MOVR D,1
0148  2F00                    CALL HEX1
0149  008E
014A  2600                    JUMP DUMP4
014B  0131
014C  2300          OUT       CI 0,H'0001'
014D  0001
014E  2800                    JZ DUMP5
014F  0159
0150  12D0                    CLRR D
0151  0F00                    DECR 0
0152  2F00          DUMP6     CALL HEX1
0153  008E
0154  2F00                    CALL SPACE
0155  0046
0156  0F00                    DECR 0
0157  2700                    JNZ DUMP6
0158  0152
0159  17D1          DUMP5     MOVR D,1
015A  2F00                    CALL HEX1
015B  008E
015C  2F00                    CALL CRLF
015D  004D
015E  2F00                    CALL PUNCH2
015F  016F
0160  2F00                    CALL CRLF
0161  004D
0162  2F00                    CALL PUNCH1
0163  0166
```

```
0164  2600                    JUMP BEGIN
0165  0001
0166  12D0      PUNCH1        CLRR D
0167  1510                    LI 1, H'0024'
0168  0024
0169  2F00      DUMP1         CALL TTYOUT3
016A  003B
016B  0F10                    DECR 1
016C  2700                    JNZ DUMP1
016D  0169
016E  3200                    RET
016F  15D0      PUNCH2        LI D, H'002A'
0170  002A
0171  1510                    LI 1, H'0012'
0172  0012
0173  2600                    JUMP DUMP1
0174  0169
0175  172D      CHECKSUM      MOVR 2,D
0176  0A12                    ADDR 1,2
0177  3200                    RET
*
* COMMAND 'P'
*
0178  2F00      P             CALL TTYIN2
0179  0056
017A  28D0                    JE D,H'0D00',YES
017B  0D00
017C  0182
017D  28D0                    JE D,H'0A00',YES
017E  0A00
017F  0182
0180  2600                    JUMP BEGIN
0181  0001
0182  15F0      YES           LI F, H'07ED'       SET DATA COUNTER
0183  07ED
0184  1500      KEY           LI 0, H'000A'
0185  000A
0186  2F00                    CALL CRLF
0187  004D
```

```
0188  21D0       KIM          MOVM D
0189  2F00                    CALL HEX1
018A  008E
018B  2F00                    CALL SPACE
018C  0046
018D  2BF0                    JE F,H'0800',BEGIN
018E  0800
018F  0100
0190  0F00                    DECR 0
0191  2700                    JNZ KIM
0192  0188
0193  2600                    JUMP KEY
0194  0184
0195  2F00       CON3         CALL CRLF
0196  004D
0197  2600                    JUMP CON2
0198  009A
0199  0F00       JUMP         DECR 0
019A  17D0                    MOVR D,0
019B  2600                    JUMP BOOK
019C  0114
*
* COMMAND 'B'
*
019D  2F00       BREAK        CALL CONVERT
019E  0077
019F  170A                    MOVR 0,A
01A0  2F00                    CALL CONVERT
01A1  0077
01A2  17FA                    MOVR F,A
01A3  21D0                    MOVM D
01A4  21E0                    MOVM E
01A5  15F0                    LI F, H'07FD'
01A6  07FD
01A7  30A0                    LRM A
01A8  30D0                    LRM D
01A9  30E0                    LRM E
```

```
01AA  17FA                    MOVR F,A
01AB  15D0                    LI D, H'2600'
01AC  2600
01AD  30D0                    LRM D
01AE  15D0                    LI D, H'01B2'
01AF  01B2
01B0  30D0                    LRM D
01B1  3100                    LRP 0
01B2  1AFF                    PUSH F,F
01B3  15F0                    LI F, H'07ED'
01B4  07ED
01B5  3000                    LRM 0
01B6  3010                    LRM 1
01B7  3020                    LRM 2
01B8  3030                    LRM 3
01B9  3040                    LRM 4
01BA  3050                    LRM 5
01BB  3060                    LRM 6
01BC  3070                    LRM 7
01BD  3080                    LRM 8
01BE  3090                    LRM 9
01BF  30A0                    LRM A
01C0  30B0                    LRM B
01C1  30C0                    LRM C
01C2  30D0                    LRM D
01C3  30E0                    LRM E
01C4  170F                    MOVR 0,F
01C5  18F1                    POP F,1
01C6  17F0                    MOVR F,0
01C7  3010                    LRM 1
01C8  2100                    MOVM 0
01C9  2110                    MOVM 1
01CA  2120                    MOVM 2
01CB  17F0                    MOVR F,0
01CC  3010                    LRM 1
01CD  3020                    LRM 2
01CE  15D0          PRINT     LI D, H'2E00'
01CF  2E00
```

```
01D0  2600                    JUMP PAPER
01D1  002A
*
* COMMAND 'L'
*
01D2  2F00        LOAD        CALL TTYIN3
01D3  0070
01D4  28D0                    JE D,H'002A',LOAD
01D5  002A
01D6  01D2
01D7  23D0                    CI D,H'0053'
01D8  0053
01D9  2700                    JNZ LOAD
01DA  01D2
01DB  2F00                    CALL CONVERT
01DC  0077
01DD  17FA                    MOVR F,A
01DE  173F        LOAD3       MOVR 3,F
01DF  1210                    CLRR 1
01E0  1500                    LI 0, H'0008'
01E1  0008
01E2  2F00        LOAD2       CALL TTYIN3
01E3  0070
01E4  28D0                    JE D,H'0058',LOAD1
01E5  0058
01E6  01EC
01E7  28D0                    JE D,H'0024',PRINT
01E8  0024
01E9  01CE
01EA  2600                    JUMP LOAD2
01EB  01E2
01EC  2F00        LOAD1       CALL CONVERT
01ED  0077
01EE  30A0                    LRM A
01EF  2F00                    CALL CHECKSUM
01F0  0175
01F1  2C00                    DSZ 0, LOAD1
01F2  01EC
01F3  2F00                    CALL CONVERT
01F4  0077
```

```
01F5  24A1                    CR A,1
01F6  2800                    JZ LOAD3
01F7  01DE
01F8  17D3                    MOVR D,3
01F9  2F00                    CALL HEX1
01FA  008E
01FB  2F00                    CALL CRLF
01FC  004D
01FD  2600                    JUMP LOAD3
01FE  01DE
01FF  3800                    NOP
*
*
* END OF UMM-16 OPERATING SYSTEM SOFTWARE LISTING
*
*
* THE FOLLOWING COMMAND IS RESERVED FOR FUTURE EXPANSION
* OF THE OPERATING SYSTEM
*
*
0200            C
*
*
```

APPENDIX B

MICROCODE OF THE FIXED INSTRUCTIONS OF UMM-16

The microcode implemented in PROM part of the microprogram memory is shown in Table B.1 . Each microinstruction is executed in a single clock cycle. The description of the microinstructions are as follows (mI_{nnn} = microinstruction at microprogram memory address nnn):-

mI_{000}: This is the first microinstruction executed when the microcomputer is RESET. This microinstruction initializes yhe system. The Program Counter Register (PC) is forced to zero and interrupts are disabled. The microprogram sequencer pushes the first microinstruction of the macroinstruction 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 os microinstructions a jump to this location should be performed 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}

INSTRUCTION	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		CCS, Carry-in								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	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
			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	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	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	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	X	X	X	X	X	X	X	X	0	1	0	0	1	1	0	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	X	X	X	X	X	X	X	X	0	1	0	0	1	1	1	1
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	X	X	X	X	X	X	X	X	0	1	0	0	1	0	1	1
$\overline{R_A} \rightarrow R_A$	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	X	X	X	X	X	X	X	X	0	1	0	0	0	1	0	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	X	X	X	X	X	X	X	X	0	1	1	0	0	1	0	1
$\overline{R_A}+1 \rightarrow R_A$	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	X	X	X	X	X	X	X	X	0	1	0	0	0	1	0	1
R_A . AND. Imm $\rightarrow R_A$	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	X	X	X	X	X	X	X	X	0	1	0	0	1	1	0	0
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	X	X	X	X	X	X	X	X	0	1	0	0	1	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	X	X	X	X	X	X	X	X	0	1	0	0	1	0	1	1

Table B.1 Microcode of the fixed instructions of UMM-16

cont...

INSTRUCTION	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	RS_3					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	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				CR, E_4 / CR, Pol / CR				Dest.				Function				Source							
			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	$\overline{E_A}$	$\overline{OE_B}$																															
			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	0,1F	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

Table B.1 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								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	RS_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	$\overline{I_8}$	$\overline{I_7}$	$\overline{I_6}$	$\overline{I_5}$	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
$R_A \rightarrow Q$	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

Table B.1 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														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	A_2	A_1	A_0	IEN	$\overline{OE}_Y$	EB	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
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	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	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	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	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	X	1	1	1	X	X

Table 8.1 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		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	$\overline{I_8}$	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	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	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

Table B.1 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								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	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
																	RS_3																																	
																	DSL		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							
																	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}	$\overline{I_8}$	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
		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

Table 8.1 cont.

cont...

will be output from the pipeline register. This microinstruction will control the microprogram sequencer so that the Mapping Memory will 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_{008}: This microinstruction implements the macroinstruction $(R_A).OR.(R_B) \longrightarrow R_A$. The microcode for this microinstruction is same as mI_{007} except that here the ALU performs a logical OR operation.

mI_{009}: This microinstruction implements the macroinstruction

(R_A).EX-OR.$(R_B) \longrightarrow R_A$. The microcode for this micro-instruction is same as mI_{007} except that here the ALU performs a logical Exclusive-OR operation.

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 : $\longrightarrow mI_{001}$

mI_{00B}: This microinstruction implements the macroinstruction $(\overline{Q}) \longrightarrow Q$. The microoperations performed are as follows:-

ALU Carry-in : $C_n = 0$

ALU Source : R = DA, S = Q

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

ALU Destination : $F \longrightarrow Q$

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

Next microinstruction : $\longrightarrow mI_{001}$

mI_{00C}: This microinstruction implements the macroinstruction $(\overline{R_A}) + 1 \longrightarrow R_A$. The microcode for this microinstruction is same as mI_{00A} except that here the carry-in of the ALU is forced HIGH.

mI_{00D}: 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 follows:-

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_{00E}: This microinstruction implements the macroinstruction (R_A).OR.Imm $\longrightarrow R_A$ where 'Imm' is a 16 bit immediate data contained in the second word of the macroinstruction. mI_{00E} is same as mI_{00D} except that here the ALU performs a logical OR operation.

mI_{00F}: This microinstruction implements the macroinstruction (R_A).EX-OR.Imm $\longrightarrow R_A$ where 'Imm' is a 16 bit immediate data contained in the second word of the

macroinstruction. mI_{00F} is same as mI_{000} except that here the ALU performs a logical Exclusive-OR operation.

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_{011}: This microinstruction implements the macroinstruction $(Q) + (R_A) \longrightarrow R_B$. The microoperations performed are as follows:-

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

ALU Carry-in : $C_n = 0$

ALU Source : $R = RAM\ A$, $S = Q$

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_{012}: 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 : $C_n = 1$

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

ALU Function : $F = S - R - 1 + C_n$

ALU Destination : $F \longrightarrow Y$

$Y \longrightarrow$ RAM B

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

Next microinstruction: $\longrightarrow mI_{001}$

mI_{013}: This microinstruction implements the macroinstruction $(R_A) + 1 \longrightarrow R_A$. The microoperations performed are as follows:-

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

ALU Carry-in : $C_n = 1$

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_{014}: This microinstruction implements the macroinstruction $(Q) + 1 \longrightarrow Q$. The microoperations performed are as follows:-

ALU Carry-in : $C_n = 1$

ALU Source : $R = DA, \quad S = Q$

ALU Function : $F = S + C_n$

ALU Destination : $F \longrightarrow Q$

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

Next microinstruction : $\longrightarrow mI_{001}$

mI_{015}: This microinstruction implements the macroinstruction $(R_A) - 1 \longrightarrow R_A$. The microcode for this microinstruction is same as mI_{013} except that here the ALU performs the operation $F = S - R - 1 + C_n$.

mI_{016}: This microinstruction implements the macroinstruction $(Q) - 1 \longrightarrow Q$. The microcode for this microinstruction is same as mI_{014} except that here the ALU performs the operation $F = S - R - 1 + C_n$.

mI_{017}: This microinstruction clears the Q register. The microoperations performed are as follows:-

Bit Manipulation Logic : $DA = 0$

ALU Source : $R = DA, \quad S = Q$

ALU Function : $F = R.AND.S$

ALU Destination : $F \longrightarrow Q$

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

Next microinstruction : $\longrightarrow mI_{001}$

mI_{018}: This microinstruction clears the register addressed by the R_A field of the operand. The microoperations performed are as follows:-

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

Bit Manipulation Logic : DA = 0

ALU Source : R = DA, 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_{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_{01A}: This microinstruction implements the macroinstruction $(R_A) - Imm \rightarrow R_A$ 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 HIGH and the ALU performs the operation $F = R - S - 1 + C_n$.

mI_{01B}: This microinstruction implements the macroinstruction $Imm \rightarrow R_A$ 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 : $\longrightarrow$ mI_{001}

mI_{01C}: This microinstruction implements the macroinstruction Imm $\longrightarrow$ Q where 'Imm' is a 16 bit immediate data contained in the second word of the macroinstruction. The microcode of this microinstruction is same as mI_{01B} except that here the result of the operation is loaded to the Q register.

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

Y $\longrightarrow$ RAM B

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

Next microinstruction : $\longrightarrow mI_{001}$

mI_{01F}: This microinstruction implements the macroinstruction $(Q) \longrightarrow R_A$. The microoperations performed are as follows:-

Operand Select Logic: A = R_B, B = R_A

Bit Manipulation Logic: DA = 0

ALU Source: R = DA, S = Q

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_{020}: This microinstruction implements the macroinstruction $(R_A) \longrightarrow Q$. The microoperations performed are as follows:-

Operand Select Logic: A = R_B, B = R_A

Bit Manipulation Logic: DA = 0

ALU Source: R = DA, S = RAM B

ALU Function: F = R.OR.S

ALU Destination: F $\longrightarrow$ Q

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

Next microinstruction: $\longrightarrow$ mI_{001}

mI_{021}: This microinstruction loads the auxiliary register (see Chapter 5.4) addressed by the R_A field of the operand from the scratchpad register addressed by the R_B field of the operand. The microoperations performed are as follows:-

Operand Select Logic: A = R_A, B = R_B

Bit Manipulation Logic: DA = 0

REL: P/P = 0, FE = 0

ALU Output Control: EDB = 0

ALU Source: R = DA, S = RAM B

ALU Function: F = R.OR.S

ALU Destination: F $\longrightarrow$ Y

Y $\longrightarrow$ RAM B

Y $\longrightarrow$ Data Bus

Data Bus $\longrightarrow$ Auxiliary register

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

Next microinstruction: $\longrightarrow$ mI_{001}

mI_{022}: This microinstruction loads the ALU register addressed by the R_B field of the operand from the auxiliary register (see Chapter 5.4) addressed by the R_A field of the operand. The microoperations performed are as follows:-

Operand Select Logic: A = R_A, B = R_B

Bit Manipulation Logic: DA = 0

REL: P/P = 1, FE = 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: $\longrightarrow$ mI_{001}

mI_{023}: This microinstruction loads the 4 highest bits of the register addressed by the R_A field of the operand from the status register. The micro-operations performed are as follows:-

Operand Select Logic: A = R_A, B = R_A

Bit Manipulation Logic: DA = 0

Status Register Control: ST_1 = 1, ST_2 = 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: $\longrightarrow$ 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 $\longrightarrow$ Y

Y $\longrightarrow$ RAM B

Y $\longrightarrow$ Data Bus

Data Bus $\longrightarrow$ Status register

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

Next microinstruction: $\longrightarrow$ 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$\rightarrow$PC

Memory unit : Memory READ request

Next microinstruction : $\longrightarrow$ 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 $\longrightarrow$ Y

Y ⟶ RAM B

Y ⟶ Data Bus

Data Bus ⟶ Output port

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

Next microinstruction: ⟶ 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 ⟶ PC

Memory Unit : Memory READ request

Next microinstruction : ⟶ 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 ⟶ Y

Y ⟶ RAM B

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

Next microinstruction : ⟶ mI_{001}

mI_{029}-mI_{028}: These three microinstructions 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 (F)

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 (F)

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_{02C}-mI_{02E}: These three microinstructions load the register addressed by the R_A field of the operand from the memory location addressed by the Data Counter (register 15). The contents of the Data Counter is incremented by one at the end of these microinstructions. The microoperations performed are as follows:-

mI_{02C}:

Operand Select Logic : A = X, B = Register 15 (F)

ALU RAM output control : EB = 0

Program Control Unit : PE = 0

Next microinstruction : $\longrightarrow mI_{02D}$

mI_{02D}:

Operand Select Logic : A = R_A, B = R_A

Program Control Unit : (Data Bus) $\longrightarrow$ Address Bus

Bit Manipulation Logic : DA = 0

Memory Unit : Memory READ request

ALU Source : R = DA, S = DB

ALU Function : F = R.OR.S

ALU Destination: F $\longrightarrow$ Y

Y $\longrightarrow$ RAM B

Next microinstruction : $\longrightarrow mI_{02E}$

mI_{02E}: Same as mI_{02B}

mI_{02F}: This microinstruction implements the macroinstruction WAIT. When executed, the system clock is put into a HALT state until the READY input is activated. The microoperations performed are as follows:-

Timing Unit : WAIT = 0

Next microinstruction : $\longrightarrow mI_{001}$ when the clock is re-enabled

mI_{030}: This microinstruction implements the macroinstruction R_A - Imm where 'Imm' is a 16 bit data contained in the second word of the macroinstruction. mI_{030} is same as mI_{01A} except that here the result is not stored in any register but only the status register is updated to reflect the results of the operation.

mI_{031}: This microinstruction implements the macroinstruction R_A - R_B. mI_{031} is same as mI_{012} except that here the result is not stored an any register but only the status register is updated to reflect the results of the operation.

mI_{032}-mI_{034}: These three microinstructions subtract the data at the memory location addressed by the Data Counter (Register 15) from the register addressed by the R_A field of the operand. The result of the operation is not stored but the status register is updated to reflect the result. The Data Counter is incremented by one at the end of these microinstructions. The microoperations performed are as follows:-

mI_{032}:

Operand Select Logic : A = X, B = Register 15 (F)

ALU RAM output control : EB = 0

Program Control Unit : PE = 0

Next microinstruction : $\longrightarrow mI_{033}$

mI_{033}: Operand Select Logic : A = R_A, B = R_A

ALU Carry-in : $C_n = 1$

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 - 1 + C_n

Next microinstruction : $\longrightarrow$ mI_{034}

mI_{034}: Same as mI_{028}

mI_{035}-mI_{037}: These three microinstructions implement the unconditional JUMP macroinstruction. The JUMP address 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 is contained in the second word of the macroinstruction. The microoperations performed are as follows:-

mI_{038}: Condition Code Select : Zero Status $\longrightarrow \overline{CC}$

Next microinstruction : $\longrightarrow mI_{035}$ if condition TRUE (Zero Status=LOW)

$\longrightarrow mI_{039}$ if condition FALSE (Zero Status=HIGH

mI_{039}: Program Control Unit : (PC) + 1 $\longrightarrow$ PC

Next microinstruction : $\longrightarrow mI_{001}$

mI_{03A}-mI_{038}: These two microinstructions implement the JUMP ON ZERO macroinstruction. The microcode for this macroinstruction is same as the microcode for the JUMP ON NON-ZERO macroinstruction except that here the Zero Status is inverted.

mI_{03C}-mI_{03D}: These two microinstructions implement the JUMP ON POSITIVE macroinstruction. The microcode for this macroinstruction is same as the microcode for the JUMP ON NON-ZERO macroinstruction except that here the $\overline{CC}$ input of the microprogram sequencer is derived from the Sign Status.

mI_{03E}-mI_{03F}: These two microinstructions implement the JUMP ON NEGATIVE macroinstruction. The microcode for this macroinstruction is same as the microcode for the JUMP ON POSITIVE macroinstruction except that here the Sign Status is inverted. .

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-Code and 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 microoperations 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 = DB

ALU Function : $F = R - S - 1 + C_n$

ALU Destination : $F \longrightarrow Y$

Next microinstruction : $\longrightarrow mI_{041}$

mI_{041}: CCS : $\overline{\text{Zero}}$ Status $\longrightarrow \overline{CC}$

Next microinstruction : $\longrightarrow mI_{035}$ if condition TRUE ($\overline{\text{Zero}}$ Status=LOW)

$\longrightarrow mI_{042}$ if condition FALSE ($\overline{\text{Zero}}$ Status=HIGH)

mI_{042}: Program Control Unit : (PC) + 1 $\longrightarrow$ PC

Next microinstruction : $\longrightarrow mI_{001}$

mI_{043}-mI_{045}: These three microinstructions implement the macroinstruction DECREMENT AND SKIP ON ZERO. This is a two word macroinstruction. First word specifies the Op-Code and the operand R_A. Second word contains a 16 bit address. The contents of the register addressed by the R_A field of the operand is decremented by one and compared with zero. If the result is non-zero then a JUMP is performed to the address provided in the second word of the macroinstruction. If the result is zero then the next sequential macroinstruction is executed. The microoperations performed are as follows:-

mI_{043}: Operand Select Logic : A = R_B, B = R_A

ALU Carry-in : C_n = 0

Bit Manipulation Logic : DA = 0

ALU Source : R = DA, S = RAM B

ALU Function : F = S - R - 1 + C_n

ALU Destination : F $\longrightarrow$ Y

Y $\longrightarrow$ RAM B

Next microinstruction : $\longrightarrow$ mI_{044}

mI_{044}: CCS Logic : Zero Status $\longrightarrow$ $\overline{CC}$

Next microinstruction : $\longrightarrow$ mI_{035} if condition FALSE (Zero Status=LOW)

$\longrightarrow$ mI_{045} if condition TRUE (Zero Status=HIGH)

mI_{045}: Program Control Unit : (PC) + 1 $\longrightarrow$ PC

Next microinstruction : $\longrightarrow$ mI_{001}

mI_{046}-mI_{048}: These three microinstructions implement the macroinstruction JUMP ON LESS THAN. This is a three word macroinstruction. First word contains the Op-Code and 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 it is less than (unsigned comparison) this data then a jump is performed to the address contained in the third word of the macroinstruction. If it is greater than or equal to this data then the next sequential macroinstruction is executed. The microcode of this macroinstruction is same as the micro-code of the JUMP ON EQUAL TO macroinstruction

except that here the $\overline{CC}$ input of the micro-program sequencer is derived from the Carry Status.

mI_{047}-mI_{048}: These two microinstructions implement the macroinstruction JUMP ON NON-CARRY. The microcode of this macroinstruction is same as the microcode of the JUMP ON NON-ZERO macroinstruction except that here the $\overline{CC}$ input of the microprogram sequencer is derived from the Carry Status.

mI_{049}-mI_{048}: These three microinstructions implement the subroutine CALL macroinstruction. The micro-operations performed are as follows:-

mI_{049}: Program Control Unit : (PC) + 1 $\longrightarrow$ PC, PE = 0

Memory Unit : Memory READ request

Next microinstruction : $\longrightarrow$ mI_{04A}

mI_{04A}: Program Control Unit : PC $\longrightarrow$ Stack, (Data Bus) $\longrightarrow$ Address Bus

Memory Unit : Memory READ request

Instruction Register : IRE = 0

Next microinstruction : $\longrightarrow$ mI_{04B}

mI_{04B}: Microprogram Sequencer : JUMP MAP

mI_{04C}-mI_{04E}: These three microinstructions load the memory location addressed by the Data Counter (register 15) from the ALU register addressed by the R_A

field of the operand. The contents of the Data Counter is incremented by one on the last cycle. The microoperations performed are as follows:-

mI_{04C}:

Operand Select Logic : A = X, B = Register 15 (F)

ALU RAM output control : EB = 0

Program Control Unit : PE = 0

Next microinstruction : $\longrightarrow mI_{04D}$

mI_{04D}:

Operand Select Logic : A = R_A, B = R_A

Program Control Unit : (Data Bus) $\longrightarrow$ Address Bus

Bit Manipulation Logic : DA = 0

ALU Source : R = DA, S = RAM B

ALU Function : F = R.OR.S

ALU Destination : F $\longrightarrow$ Y

Y $\longrightarrow$ RAM B

Y $\longrightarrow$ Data Bus

ALU output control : EDB = 0

Memory Unit : Memory WRITE request

Next microinstruction : $\longrightarrow mI_{04E}$

mI_{04E}: Same as mI_{028}

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 microoperations 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 $\longrightarrow$ Y

Y $\longrightarrow$ RAM B

Y $\longrightarrow$ Data Bus

Next microinstruction: $\longrightarrow$ mI_{050}

mI_{050}: 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_{051}-mI_{053}: These three microinstructions implement the subroutine RETURN macroinstruction. The microoperations performed are as follows:-

mI_{051}: Program Control Unit: (Stack) $\longrightarrow$ PC

Next microinstruction: $\longrightarrow$ mI_{052}

mI_{052}: Program Control Unit: (PC) $\longrightarrow$ Address Bus

Memory Unit: Memory READ request

Instruction Register: IRE = 0

Next microinstruction: $\longrightarrow$ 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 here the next microinstruction executed is mI_{001}.

mI_{055}: This microinstruction logically rotates right (by one position) the contents of the register addressed by the R_A field of the operand.

mI_{056}: This microinstruction enables the interrupts to be recognized by the CCU. The microoperations performed are as follows:-

Interrupt Control Unit : $A_3 = A_2 = A_1 = A_0 = 0$

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

Next microinstruction : $\longrightarrow mI_{001}$

mI_{057}: This microinstruction prevents the CCU from recognizing an interrupt. The microoperations performed are as follows:-

Interrupt Control Unit : $A_3 = A_2 = A_1 = 0, \quad A_0 = 1$

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

Next microinstruction : $\longrightarrow mI_{001}$

mI_{058}: This microinstruction loads the Mask Register of the ICU. This is a two word macroinstruction. First word contains the Op-Code. The data for the Mask Register is contained in the higher byte of the second word of the macroinstruction. The micro-operations performed are as follows:-

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

$(PC) \longrightarrow$ Address Bus

Memory Unit : Memory READ request

Interrupt Control Unit : $A_3 = A_2 = 0, \quad A_1 = A_0 = 1$

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

Next microinstruction : $\longrightarrow mI_{001}$

mI_{059}: This microinstruction loads the Timer of the ICU with the 16 bit data contained in the second word of the macroinstruction. mI_{059} is same as mI_{058} except that here the ICU performs the operation $A_3 = A_1 = A_0 = 0, A_2 = 1$.

mI_{05A}: This microinstruction shifts right (logically) the

contents of the register addressed by the R_A field of the operand. A '1' is shifted to the most significant bit position. The microoperations performed are as follows:-

Operand Select Logic : A = R_A, B = R_B

Bit Manipulation Logic : DA = 0

Data Shifting Logic : $S_4 = S_5 = 0$ (shift)
DS = 1

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 : $\longrightarrow mI_{001}$

mI_{058}: This microinstruction shifts left (logically) the contents of the register addressed by the R_A field of the operand. A '0' is shifted to the least significant bit position. The microoperations performed are as follows:-

Operand Select Logic : A = R_A, B = R_B

Bit Manipulation Logic : DA = 0

Data Shifting Logic : $S_4 = S_5 = 0$ (shift)
DS = 1

ALU Source : R = DA, S = RAM B

ALU Function : F = R.OR.S

ALU Destination : 2F $\longrightarrow$ Y
Y $\longrightarrow$ RAM B

Next microinstruction : $\longrightarrow mI_{001}$

mI_{05D}: This microinstruction logically rotates left (by one position) the contents of the register addressed by the R_A field of the operand. mI_{05D} is same as mI_{055} except that here the ALU performs the operation: $2F \longrightarrow Y$

mI_{05E}: This microinstruction implements the NO OPERATION macroinstruction. When executed, the Program Counter Register is just incremented by one and the next microinstruction is selected to be mI_{001}.

mI_{05F}: This microinstruction shifts right logically the contents of the register addressed by the R_A field of the operand. A '0' is shifted to the most significant bit position. mI_{05F} is same as mI_{05A} except that here a '0' is shifted instead of a '1'.

Microcode for the Interrupt Service Routine (mI_{003}-mI_{006})

The CCU can only service an interrupt request during the execution of the second microinstruction of the FETCH cycle (mI_{002}). When an interrupt request is recognized by the CCU, the address inputs of the Mapping Memory are forced LOW. 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. 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 and 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 macroinstruction 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 (mI_{05C})

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 microoperations performed are as follows:-

Program Control Unit : (Stack) $\rightarrow$ PC

Next microinstruction : $\longrightarrow mI_{052}$

APPENDIX C

SUPPORT PROCESSOR SOFTWARE LISTING

```
                *
                * F-8 RESIDENT META-ASSEMBLER PROGRAM
                *
                *
                * ******************************
                * PROGRAMMER: D.  IBRAHIM
                * ******************************
                *
                *
                      ORG H'0000'
                     PUNCH ON
                *
                *
                *
                * THE BASIC SYSTEM CONSISTS OF:
                *
                * 3850 CENTRAL PROCESSING UNIT (CPU)
                *
                * 3853 STATIC MEMORY INTERFACE (SMI)
                *
                * 3871 PROGRAMMABLE INPUT OUTPUT (PIO)
                *
                * 2K OF 2708 EPROM HOLDING THIS PROGRAM
                *
                * 2K OF RAM (SY2114) FOR DATA STORAGE
                *
                *
                *
0000 29 00 97         JMP POWE
                *
                *
                * TYPE THE 8 BIT CHARACTER IN REG. 5
                *
                * DATA FORMAT: 8 BIT DATA, 1 STOP BIT, NO PARITY
                *
                *
                *
                * BAUD RATE = 110,300,600,1200 SET BY THE TIMER OF THE PIO
                *
                *
0003 08         TTYOUT  LR K,P
0004 70                 CLR
0005 58                 LR 8,A
0006 78                 LIS H'08'
0007 57                 LR 7,A
0008 45                 LR A,5
0009 54                 LR 4,A
000A 56                 LR 6,A
000B 90 5B              BR TIMER
                *
                * SEND START BIT
                *
000D 70         SEND    CLR
MOSTEK F8 CROSS ASSEMBLER  THE CITY UNIVERSITY  LONDON,  PAGE   1
```

```
000E B5            OUTS 5
000F 1B        B1  EI
0010 44            LR A,4
0011 21 01         NI H'01'
0013 54            LR 4,A
0014 90 FF     B2  BR B2
0016 44        BB  LR A,4
0017 B5            OUTS 5
0018 46            LR A,6
0019 12            SR 1
001A 54            LR 4,A
001B 56            LR 6,A
001C 37            DS 7
001D 90 F1         BR B1
001F 2B            NOP
                   *
                   * TIMER ADDRESS
                   *
0020 47            LR A,7
0021 25 00         CI H'00'
0023 84 08         BZ OUT
0025 48            LR A,8
0026 25 00         CI H'00'
0028 84 ED         BZ BB
002A 90 31         BR PER
002C 71        OUT LIS H'01'
002D B5            OUTS 5
002E 75            LIS H'05'
002F 57            LR 7,A
0030 37        NT03  DS 7
0031 84 09         BZ NT05
0033 20 FF         LI H'FF'
0035 58            LR 8,A
0036 38        NT04  DS 8
0037 94 FE         BNZ NT04
0039 90 F6         BR NT03
003B 0C        NT05  PK
                   *
                   * TYPE CARRIAGE RET.
                   *
003C 7D        CR  LIS H'0D'
003D 55            LR 5,A
003E 90 C4         BR TTYOUT
                   *
                   *
                   *
                   * TYPE LINE FEED
                   *
                   *
                   *
0040 7A        LF  LIS H'0A'
0041 55            LR 5,A
0042 90 C0         BR TTYOUT
                   *
```

```
                    *
                    *
                    * TYPE SPACE
                    *
                    *
                    *
0044 20 20          SPACE  LI H'20'
0046 55                    LR 5,A
0047 90 BB                 BR TTYOUT
                    *
                    *
                    *
                    * RECEIVE 7 BIT CHARACTER TO REG. 5
                    * DATA FORMAT: 7 BIT DATA, 1 STOP BIT, NO PARITY
                    *
                    *
                    *
0049 08             TTYIN  LR K,P
004A 71                    LIS H'01'
004B 58                    LR 8,A
004C B5                    OUTS 5
004D 55                    LR 5,A
004E 77                    LIS H'07'
004F 57                    LR 7,A
0050 A5             N1   INS 5
0051 21 80                 NI H'80'
0053 25 80                 CI H'80'
0055 84 11                 BZ TIMER
0057 90 F8                 BR N1
0059 1B             BACC   EI
005A 90 FF          PEN    BR PEN
005C A5             PER    INS 5
005D 18                    COM
005E 21 80                 NI H'80'
0060 C5                    AS 5
0061 12                    SR 1
0062 55                    LR 5,A
0063 37                    DS 7
0064 90 F4                 BR BACC
0066 2B                    NOP
                    *
                    * SET THE BAUD RATE
                    *
                    *
                    * THE BAUD RATE IS SET AS FOLLOWS:
                    *
                    * 1200 BAUD: PORT 5 BIT 4 = 0
                    *           PORT 5 BIT 5 = 0
                    *
                    * 600 BAUD  : PORT 5 BIT 4 = 1
                    *           PORT 5 BIT 5 = 0
                    *
                    * 300 BAUD  : PORT 5 BIT 4 = 0
MOSTEK F8 CROSS ASSEMBLER  THE CITY UNIVERSITY  LONDON.  PAGE  3
```

```
                    *              PORT 5 BIT 5 = 1
                    *
                    * 110 BAUD :  PORT 5 BIT 4 = 1
                    *              PORT 5 BIT 5 = 1
                    *
                    *
                    *
0067 A5             TIMER  INS 5
0068 21 30               NI H'30'
006A 25 10               CI H'10'
006C 94 04               BNZ PARS
006E 29 03 F8            JMP BA3
0071 25 20          PARS  CI H'20'
0073 94 07               BNZ PARI
0075 20 21               LI H'21'
0077 B7             ASH    OUTS 7
0078 29 03 FB            JMP LON
007B 25 30          PARI   CI H'30'
007D 94 05               BNZ B11
007F 20 11               LI H'11'
0081 90 F5               BR ASH
0083 29 03 E8 B11   JMP BA11
                    *
                    *
                    *
                    * TYPE THE CHARACTER IN LOWER BYTE OF REG. 5 IN HEX
                    *
                    *
0086 45             HEX   LR A,5
0087 21 0F               NI H'0F'
0089 25 09               CI H'09'
008B 91 05               BM THO
008D 24 30               AI H'30'
008F 90 03               BR LOOB
0091 24 37          THO   AI H'37'
0093 55             LOOB  LR 5,A
0094 29 00 03            JMP TTYOUT
                    *
                    *
                    *
                    * THE ASSEMBLER PROGRAM
                    *
                    *
                    *
                    *MICROPROGRAM START ADDRESS=0400
                    *
                    *MAPPING RAM START ADDRESS =0A00
                    *
                    *
                    *
                    *
                    MICST EQU H'0400'
                    MAPST EQU H'0A00'
0097 20 FF          POWE LI H'FF'
MOSTEK F8 CROSS ASSEMBLER  THE CITY UNIVERSITY  LONDON.  PAGE  4
```

```
0099 0B              LR IS,A
009A 2A 0C 8A START  DCI HEAD      PRINT HEADING
009D 16         PRINT LM
009E 25 23           CI C'#'
00A0 84 07           BZ ABC
00A2 55              LR 5,A
00A3 28 00 03        PI TTYOUT
00A6 90 F6           BR PRINT
00A8 28 00 40 ABC    PI LF
00AB 28 00 3C        PI CR
00AE 20 2A           LI C'*'
00B0 55              LR 5,A
00B1 28 00 03        PI TTYOUT
00B4 28 00 44        PI SPACE
00B7 28 00 49        PI TTYIN
00BA 28 00 03        PI TTYOUT
00BD 45              LR A,5
00BE 25 4E           CI C'N'       START OF INSTRUCTION DECODING
00C0 84 26           BZ T1
00C2 25 44           CI C'D'
00C4 84 3E           BZ DDT2
00C6 25 4C           CI C'L'
00C8 84 6A           BZ T2
00CA 25 48           CI C'H'
00CC 84 3F           BZ VDU
00CE 25 45           CI C'E'
00D0 84 38           BZ EXC
00D2 50              LR 0,A
00D3 28 00 44        PI SPACE
00D6 40              LR A,0
00D7 25 54           CI C'T'
00D9 84 2C           BZ T4
00DB 25 4D           CI C'M'
00DD 84 31           BZ T5
00DF 20 3F      DEL  LI C'?'
00E1 55              LR 5,A
00E2 28 00 03        PI TTYOUT
00E5 90 C2           BR ABC
00E7 28 00 49 T1     PI TTYIN      START OF INSTRUCTION PROCESSING
00EA 28 00 03        PI TTYOUT
00ED 45              LR A,5
00EE 25 4D           CI C'M'
00F0 94 5C           BNZ ABDC
00F2 28 00 49        PI TTYIN
00F5 28 00 03        PI TTYOUT
00F8 45              LR A,5
00F9 25 49           CI C'I'
00FB 84 1C           BZ T6
00FD 25 41           CI C'A'
00FF 84 18           BZ T6
0101 90 A6      ABCD BR ABC
0103 29 03 7E DDT2     JMP DUMP
0106 29 02 60 T4    JMP T44
0109 29 0C 19 EXC     JMP EXCH
MOSTEK F8 CROSS ASSEMBLER  THE CITY UNIVERSITY  LONDON.  PAGE  5
```

```
010C 29 0C 00 VDU    JMP VUD
010F 29 01 D8 T5    JMP T55
0112 29 02 C7 MICC JMP MIC
0115 29 03 0E MAPP JMP MAP
0118 28 00 49 T6    PI TTYIN
011B 28 00 03       PI TTYOUT
011E 45             LR A,5
011F 51             LR 1,A
0120 28 00 49       PI TTYIN
0123 45             LR A,5
0124 25 0D          CI H'0D'
0126 94 DA          BNZ ABCD
0128 41             LR A,1
0129 25 43          CI C'C'
012B 84 E6          BZ MICC        NEW MICROPROGRAM
012D 25 50          CI C'P'
012F 84 E5          BZ MAPP        NEW MAPPING RAM DATA
0131 90 CF          BR ABCD
0133 28 00 49 T2    PI TTYIN
0136 28 00 03       PI TTYOUT
0139 45             LR A,5
013A 25 4D          CI C'M'
013C 94 34          BNZ POT
013E 28 00 49       PI TTYIN
0141 28 00 03       PI TTYOUT
0144 45             LR A,5
0145 25 49          CI C'I'
0147 84 07          BZ T7
0149 25 41          CI C'A'
014B 84 03          BZ T7
014D 90 B3     ABDC BR ABCD
014F 28 00 49 T7    PI TTYIN
0152 28 00 03       PI TTYOUT
0155 45             LR A,5
0156 51             LR 1,A
0157 28 00 49       PI TTYIN
015A 45             LR A,5
015B 25 0D          CI H'0D'
015D 94 EF          BNZ ABDC
015F 41             LR A,1
0160 25 43          CI C'C'
0162 84 15          BZ LMIC
0164 25 50          CI C'P'
0166 84 42          BZ LMAP
0168 90 E4          BR ABDC
016A 25 2A     SILK   CI H'2A'
016C 84 E0            BZ ABDC
016E 29 02 52         JMP MES
0171 25 44     POT    CI C'D'
0173 94 D9            BNZ ABDC
0175 29 03 12         JMP LOAD
                *
                * MICROPRAGRAM LOADING ROUTINE
                *
```

```
0178 2A 04 00 LMIC DCI MICST
017B 70            CLR
017C 52            LR 2,A
017D 75            LIS H'05'
017E 51            LR 1,A
                 * REGISTER 2 HOLDS THE ADDRESS
017F 42          HERE  LR A,2
0180 18            COM
0181 B0            OUTS 0
0182 16          BACK  LM
0183 B4            OUTS 4
0184 41          NOT   LR A,1
0185 15            SL 4
0186 B1          MORE   OUTS 1
0187 22 F0         OI H'F0'
0189 B1            OUTS 1
018A 41            LR A,1
018B 24 FF         AI H'FF'
018D 51            LR 1,A
018E 81 F3         BP BACK
                 * TEST FOR ENDING
0190 75            LIS H'05'
0191 51            LR 1,A
0192 42            LR A,2
0193 25 FF         CI H'FF'
0195 84 05         BZ LOOP
0197 1F            INC
0198 52            LR 2,A
0199 90 E5         BR HERE
                 *
                 * END OF MICROPROGRAM LOADING
                 *
019B 2A 0C CD LOOP  DCI MLOAD
019E 16         L1    LM
019F 25 23         CI C'#'
01A1 84 AB         BZ ABDC
01A3 55            LR 5,A
01A4 28 00 03      PI TTYOUT
01A7 90 F6         BR L1
                 *
                 * MAPPING RAM LOADING ROUTINE
                 *
01A9 2A 0A 00 LMAP  DCI MAPST
01AC 20 80         LI H'80'
01AE 51          MAP1 LR 1,A
01AF 18            COM
01B0 B0            OUTS 0
01B1 16            LM
01B2 B4            OUTS 4
01B3 20 10         LI H'10'
01B5 B1            OUTS 1
01B6 70            CLR
01B7 B1            OUTS 1
01B8 16            LM
MOSTEK F8 CROSS ASSEMBLER  THE CITY UNIVERSITY  LONDON,  PAGE  7
```

```
01B9 B4             OUTS 4
01BA 70             CLR
01BB B1             OUTS 1
01BC 20 FF          LI H'FF'
01BE B1             OUTS 1
                  * TEST FOR ENDING
01BF 41             LR A,1
01C0 25 FF          CI H'FF'
01C2 84 04          BZ LOOM
01C4 1F             INC
01C5 90 E8          BR MAP1
                  *
                  * END OF LOADING
                  *
01C7 2A 0C EA LOOM  DCI PLOAD
01CA 16       L2    LM
01CB 25 23          CI C'#'
01CD 84 07          BZ ABBA
01CF 55             LR 5,A
01D0 28 00 03       PI TTYOUT
01D3 90 F6          BR L2
01D5 29 00 A8 ABBA  JMP ABC
                  *
                  * THE M COMMAND
                  *
01D8 70       T55   CLR
01D9 5A             LR 10,A
01DA 76       TN1   LIS H'06'
01DB 50             LR 0,A
01DC 5C             LR 5,A
01DD 28 00 49 CAR   PI TTYIN
01E0 28 00 03       PI TTYOUT
01E3 45             LR A,5
01E4 25 31          CI H'31'
01E6 94 83          BNZ SILK
01E8 70             CLR
01E9 07           LR QL,A
01EA 28 00 49 CALE  PI TTYIN
01ED 28 00 03       PI TTYOUT
01F0 28 02 3F       PI CONV
01F3 43             LR A,3
                  *
                  * MULTIPLY BY SIX     .....
                  *
01F4 13             SL 1
01F5 03             AS 3
01F6 13             SL 1
                  *........
                  *................
01F7 53             LR 3,A
01F8 03             LR A,QL
01F9 25 01          CI H'01'
01FB 84 0E          BZ SWPZ
01FD 43             LR A,3
MOSTEK F8 CROSS ASSEMBLER  THE CITY UNIVERSITY  LONDON,  PAGE   8
```

```
01FE 14                 SR 4
01FF 24 04              AI H'04'
0201 06                 LR QU,A
0202 0F                 LR DC,Q
0203 43                 LR A,3
0204 15                 SL 4
0205 51                 LR 1,A
0206 71                 LIS H'01'
0207 07                 LR QL,A
0208 90 E1              BR CALE
020A 29 0E B9 SWPZ      JMP CORR
020D 0E       ALI       LR Q,DC
020E 4C                 LR A,S
020F 25 01              CI H'01'
0211 84 28              BZ DESK
0213 4A                 LR A,10
0214 25 01              CI H'01'
0216 84 20              BZ YES
0218 28 00 49           PI TTYIN
021B 45                 LR A,5
021C 25 2A              CI C'*'
021E 84 16              BZ BBA
0220 28 00 44           PI SPACE
0223 28 00 44 BA        PI SPACE
0226 16                 LM
0227 07                 LR QL,A
0228 14                 SR 4
0229 55                 LR 5,A
022A 28 00 86           PI HEX
022D 03                 LR A,QL
022E 55                 LR 5,A
022F 28 00 86           PI HEX
0232 30                 DS 0
0233 94 EF              BNZ BA
0235 90 9F    BBA       BR ABBA
0237 29 02 E1 YES       JMP YESS
023A 29 03 62 DESK      JMP NULL
023D 90 F7    BOOK      BR BBA
                        *
                        * CONVERT TO HEXADECIMAL
                        *
023F 45       CONV      LR A,5
0240 25 2A              CI C'*'
0242 84 F2              BZ BBA
0244 25 39              CI H'39'
0246 82 05              BC CON5
0248 25 46              CI H'46'
024A 82 03              BC CON6
024C 24 07    CON5      AI H'07'
024E 24 C9    CON6      AI H'C9'
0250 53                 LR 3,A
0251 1C                 POP
0252 2A 0C 78 MES       DCI WHAT
0255 16       F1        LM
MOSTEK F8 CROSS ASSEMBLER  THE CITY UNIVERSITY  LONDON.  PAGE 9
```

```
0256 25 23          CI C'#'
0258 84 DC          BZ BBA
025A 55             LR 5,A
025B 28 00 03       PI TTYOUT
025E 90 F6          BR F1
0260 70       T44    CLR
0261 5A             LR 10,A
0262 72       TN2    LIS H'02'
0263 50             LR 0,A
0264 5C             LR S,A
0265 28 00 49 CUP   PI TTYIN
0268 28 00 03       PI TTYOUT
026B 28 02 3F       PI CONV
026E 43             LR A,3
026F 15             SL 4
0270 2B             NOP
0271 81 E0          BP MES
0273 13             SL 1
0274 51             LR 1,A
0275 7A             LIS H'0A'
0276 06             LR QU,A
0277 28 00 49       PI TTYIN
027A 28 00 03       PI TTYOUT
027D 28 02 3F       PI CONV
0280 43             LR A,3
0281 C3             AS 3
0282 C1             AS 1
0283 07             LR QL,A
0284 0F             LR DC,Q
0285 4C             LR A,S
0286 25 03          CI H'03'
0288 84 B1          BZ DESK
028A 4A             LR A,10
028B 25 01          CI H'01'
028D 84 53          BZ  YESS
028F 28 00 49       PI TTYIN
0292 45             LR A,5
0293 25 2A          CI C'*'
0295 84 9F          BZ BBA
0297 2B             NOP
0298 2B             NOP
0299 28 00 44       PI SPACE
029C 28 00 44 TA    PI SPACE
029F 20 31          LI H'31'
02A1 55             LR 5,A
02A2 5C             LR S,A
02A3 28 00 03       PI TTYOUT
02A6 16             LM
02A7 16       RS5   LM
02A8 07             LR QL,A
02A9 14             SR 4
02AA 55             LR 5,A
02AB 28 00 86       PI HEX
02AE 03             LR A,QL
MOSTEK F8 CROSS ASSEMBLER  THE CITY UNIVERSITY  LONDON.  PAGE  10
```

```
02AF 55                 LR 5,A
02B0 28 00 86           PI HEX
02B3 4C                 LR A,S
02B4 25 31              CI H'31'
02B6 94 0D              BNZ RS6
02B8 90 84              BR BOOK
02BA 43           POR    LR A,3
02BB 25 01              CI H'01'
02BD 94 94              BNZ MES
02BF 17                 ST
02C0 73                 LIS H'03'
02C1 52                 LR 2,A
02C2 90 24              BR WRN
02C4 29 03 C2     RS6   JMP RS7
02C7 70           MIC    CLR
02C8 52                 LR 2,A
02C9 2A 0E 8A     K1    DCI NEW
02CC 16           C1     LM
02CD 25 23              CI C'#'
02CF 84 07              BZ Z1
02D1 55                 LR 5,A
02D2 28 00 03           PI TTYOUT
02D5 90 F6              BR C1
02D7 71           Z1     LIS H'01'
02D8 5A                 LR 10,A
02D9 42                 LR A,2
02DA 25 02              CI H'02'
02DC 84 85              BZ TN2
02DE 29 01 DA           JMP TN1
02E1 28 00 44 YESS    PI SPACE
02E4 28 00 44 PAP     PI SPACE
02E7 28 00 49 WRN     PI TTYIN
02EA 28 00 03           PI TTYOUT
02ED 28 02 3F           PI CONV
02F0 42                 LR A,2
02F1 25 02              CI H'02'
02F3 84 C6              BZ POR
02F5 43                 LR A,3
02F6 15                 SL 4
02F7 51                 LR 1,A
02F8 28 00 49           PI TTYIN
02FB 28 00 03           PI TTYOUT
02FE 28 02 3F           PI CONV
0301 43                 LR A,3
0302 C1                 AS 1
0303 17                 ST
0304 42                 LR A,2
0305 25 03              CI H'03'
0307 84 06              BZ MAP
0309 30                 DS 0
030A 94 D9              BNZ PAP
030C 90 BA              BR MIC
030E 72           MAP   LIS H'02'
030F 52                 LR 2,A
```

```
0310 90 B8              BR K1
                    *
                    * LOADING FROM PAPER TAPE
                    *
0312 28 00 49 LOAD   PI TTYIN
0315 2B              NOP
0316 2B              NOP
0317 2B              NOP
0318 45              LR A,5
0319 25 53           CI C'S'
031B 84 07           BZ S1
031D 25 44           CI C'D'
031F 84 0A           BZ S2
0321 90 F0           BR LOAD
0323 2A 04 00 S1     DCI MICST
0326 76              LIS H'06'
0327 52              LR 2,A
0328 90 06           BR S3
032A 2A 0A 00 S2     DCI MAPST
032D 72              LIS H'02'
032E 52              LR 2,A
032F 51         S3   LR 1,A
0330 28 00 49 S6     PI TTYIN
0333 2B              NOP
0334 2B              NOP
0335 2B              NOP
0336 45              LR A,5
0337 25 58           CI C'X'
0339 94 20           BNZ S4
033B 28 00 49 S5      PI TTYIN
033E 2B              NOP
033F 2B              NOP
0340 2B              NOP
0341 28 02 3F        PI CONV
0344 43              LR A,3
0345 15              SL 4
0346 50              LR 0,A
0347 28 00 49        PI TTYIN
034A 2B              NOP
034B 2B              NOP
034C 2B              NOP
034D 28 02 3F        PI CONV
0350 43              LR A,3
0351 C0              AS 0
0352 17              ST
0353 32              DS 2
0354 94 E6           BNZ S5
0356 41              LR A,1
0357 52              LR 2,A
0358 90 D7           BR S6
035A 25 2A      S4   CI C'*'
035C 94 D3           BNZ S6
035E 2B              NOP
035F 29 00 A8 PL6    JMP ABC
MOSTEK F8 CROSS ASSEMBLER  THE CITY UNIVERSITY  LONDON.  PAGE  12
```

```
                    *
                    *
                    *
                    *
                    * PRINT NULL LEADER
                    *
                    *
                    *
0362 20 30          NULL   LI H'30'
0364 51                    LR 1,A
0365 70                    CLR
0366 55                    LR 5,A
0367 28 00 03       RS1    PI TTYOUT
036A 31                    DS 1
036B 94 FB                 BNZ RS1
036D 28 00 40              PI LF
0370 28 00 3C              PI CR
0373 4C                    LR A,5
0374 25 2A                 CI C'*'
0376 84 E8                 BZ PL6
0378 25 01                 CI H'01'
037A 84 1C                 BZ RS2
037C 90 30                 BR RS3
                    *
                    * DUMPING ON PAPER TAPE
                    *
037E 28 00 49       DUMP    PI TTYIN
0381 28 00 03              PI TTYOUT
0384 45                     LR A,5
0385 25 43                 CI C'C'
0387 84 07                 BZ DMIC
0389 25 50                 CI C'P'
038B 84 19                 BZ DMAP
038D 90 D1                 BR PL6
038F 71             DMIC    LIS H'01'
0390 5C                    LR 5,A
0391 28 00 44              PI SPACE
0394 29 01 DD              JMP CAR
0397 2A 04 00       RS2    DCI MICST
039A 76                    LIS H'06'
039B 50                    LR 0,A
039C 51                    LR 1,A
039D 20 53                 LI C'S'
039F 55             CAP    LR 5,A
03A0 28 00 03              PI TTYOUT
03A3 90 13                 BR RS4
03A5 73             DMAP    LIS H'03'
03A6 5C                    LR 5,A
03A7 28 00 44              PI SPACE
03AA 29 02 65              JMP CUP
03AD 2A 0A 00       RS3    DCI MAPST
03B0 72                    LIS H'02'
03B1 50                    LR 0,A
03B2 51                    LR 1,A
```

```
03B3 20 44          LI C'D'
03B5 90 E9          BR CAP
03B7 03       RS4   LR A,QL
03B8 52             LR 2,A
03B9 20 58    BIG   LI C'X'
03BB 55             LR 5,A
03BC 28 00 03       PI TTYOUT
03BF 29 02 A7 RS8   JMP RS5
03C2 30       RS7   DS 0
03C3 94 FB          BNZ RS8
03C5 41             LR A,1
03C6 50             LR 0,A
03C7 28 00 40       PI LF
03CA 28 00 3C       PI CR
03CD 02             LR A,QU
03CE 53             LR 3,A
03CF 0E             LR Q,DC
03D0 03             LR A,QL
03D1 18             COM
03D2 1F             INC
03D3 C2             AS 2
03D4 84 03          BZ RSA
03D6 90 E2    RSB   BR BIG
03D8 02       RSA   LR A,QU
03D9 18             COM
03DA 1F             INC
03DB C3             AS 3
03DC 94 F9          BNZ RSB
03DE 20 2A          LI C'*'
03E0 55             LR 5,A
03E1 5C             LR S,A
03E2 28 00 03       PI TTYOUT
03E5 29 03 62       JMP NULL
                    *
                    *
                    * 110 BAUD RATE
                    *
                    *
03E8 20 B9    BA11   LI H'B9'
03EA B7              OUTS 7
03EB 20 CA           LI H'CA'
03ED B6              OUTS 6
03EE 48       SOLD   LR A,8
03EF 13              SL 1
03F0 84 04           BZ DNES
03F2 29 00 59        JMP BACC
03F5 29 00 0D DNES   JMP SEND
                    *
                    *
                    * 300 BAUD RATE
                    *
                    *
03F8 20 42    BA3   LI H'42'
03FA B7             OUTS 7
```

```
03FB 20 CA     LON  LI H'CA'
03FD B6             OUTS 6
03FE 90 EF          BR SOLD
                *
                *
                * SECOND PART OF PROM DATA
                *
                *
                * PROGRAM START ADDRESS H'0C00'
                *
                * THIS PROGRAM IS STORED IN A SECOND 2708
                *
                *
                *
                    ORG H'0C00'
0C00 28 00 49 VUD    PI TTYIN
0C03 45             LR A,5
0C04 25 0D          CI H'0D'
0C06 94 0F          BNZ ERG
0C08 2A 0C FF       DCI HELP
0C0B 16         L3  LM
0C0C 25 23          CI C'#'
0C0E 84 07          BZ ERG
0C10 55             LR 5,A
0C11 28 00 03       PI TTYOUT
0C14 90 F6          BR L3
0C16 29 01 D5 ERG   JMP ABBA
                *
                *
                *
                *
                * E/XX/YY/
                *EXCHANGE STRING XX WITH YY
                *AFTER THE M   COMMAND
                *
                *
                *
                *
0C19 70         EXCH  CLR
0C1A 5C             LR 5,A
0C1B 28 00 49 MART  PI TTYIN
0C1E 28 00 03       PI TTYOUT
0C21 45             LR A,5
0C22 25 2F          CI C'/'
0C24 94 24          BNZ INV
0C26 28 00 49       PI TTYIN
0C29 28 00 03       PI TTYOUT
0C2C 28 02 3F       PI CONV
0C2F 43             LR A,3
0C30 15             SL 4
0C31 50             LR 0,A
0C32 28 00 49       PI TTYIN
0C35 28 00 03       PI TTYOUT
0C38 28 02 3F       PI CONV
MOSTEK F8 CROSS ASSEMBLER  THE CITY UNIVERSITY  LONDON.  PAGE 15
```

```
0C3B 43             LR A,3
0C3C C0             AS 0
0C3D 50             LR 0,A
                    *
                    * REG. 1 HOLDS STRING XX
                    *
0C3E 4C             LR A,S
0C3F 25 01          CI H'01'
0C41 84 10          BZ AND
0C43 40             LR A,0
0C44 51             LR 1,A
0C45 71             LIS H'01'
0C46 5C             LR S,A
0C47 90 D3          BR MART
                    *
                    * REG. 0 HOLDS STRING YY
                    *
0C49 28 00 40 INV    PI LF
0C4C 28 00 3C       PI CR
0C4F 29 00 DF       JMP DEL
0C52 28 00 49 AND    PI TTYIN
0C55 28 00 03       PI TTYOUT
0C58 45             LR A,5
0C59 25 2F          CI C'/'
0C5B 94 ED          BNZ INV
0C5D 2B             NOP
                    *
                    *
                    * SUBTRACT 6 FROM THE DATA COUNTER
                    *
                    *
0C5E 0E             LR Q,DC
0C5F 03             LR A,QL
0C60 24 FA          AI H'FA'
0C62 82 08          BC CARRY
0C64 07             LR QL,A
0C65 02             LR A,QU
0C66 24 FF          AI H'FF'
0C68 06             LR QU,A
0C69 90 02          BR TELE
0C6B 07          CARRY  LR QL,A
0C6C 0F          TELE   LR DC,Q
0C6D 29 0E A8       JMP RADO
0C70 2B             NOP
0C71 2B             NOP
0C72 2B             NOP
0C73 2B             NOP
0C74 40          HEAT   LR A,0
                    *
                    * STORE STRING YY IN THE MEMORY
                    *
0C75 17             ST
0C76 90 9F          BR ERG
                    *
MOSTEK F8 CROSS ASSEMBLER  THE CITY UNIVERSITY  LONDON,  PAGE  16
```

```
                    *
                    * ASSEMBLER MESSAGES
                    *
                    *
0C78 0A 0D          WHAT  DC H'0A0D'
0C7A 49 4E 56             DC 16,C'INVALID ADDRESS#'
0C7D 41 4C 49
0C80 44 20 41
0C83 44 44 52
0C86 45 53 53
0C89 23

0C8A 0A 0D          HEAD  DC H'0A0D'
0C8C 54 48 45             DC 19,C'THE CITY UNIVERSITY'
0C8F 20 43 49
0C92 54 59 20
0C95 55 4E 49
0C98 56 45 52
0C9B 53 49 54
0C9E 59

0C9F 0A 0D                DC H'0A0D'
0CA1 46 38 20             DC 25,C'F8 MINI META-ASSEMBLER V1'
0CA4 4D 49 4E
0CA7 49 20 4D
0CAA 45 54 41
0CAD 2D 41 53
0CB0 53 45 4D
0CB3 42 4C 45
0CB6 52 20 56
0CB9 31

0CBA 0A 0D                DC H'0A0D'
0CBC 0A                   DC H'0A'
0CBD 54 59 50             DC 16,C'TYPE H FOR HELP#'
0CC0 45 20 48
0CC3 20 46 4F
0CC6 52 20 48
0CC9 45 4C 50
0CCC 23

0CCD 0A 0D          MLOAD  DC H'0A0D'
0CCF 4D 49 43             DC 27,C'MICROPROGRAM MEMORY LOADED#'
0CD2 52 4F 50
0CD5 52 4F 47
0CD8 52 41 4D
0CDB 20 4D 45
0CDE 4D 4F 52
0CE1 59 20 4C
0CE4 4F 41 44
0CE7 45 44 23
0CEA 0A 0D          PLOAD  DC H'0A0D'
0CEC 4D 41 50             DC 19,C'MAPPING RAM LOADED#'
0CEF 50 49 4E
MOSTEK F8 CROSS ASSEMBLER  THE CITY UNIVERSITY  LONDON,  PAGE  17
```

```
0CF2 47 20 52
0CF5 41 4D 20
0CF8 4C 4F 41
0CFB 44 45 44
0CFE 23

0CFF 0A 0D      HELP  DC H'0A0D'
0D01 43 4F 4D         DC 9,C'COMMANDS:'
0D04 4D 41 4E
0D07 44 53 53
0D0A 0A 0D            DC H'0A0D'
0D0C 4E 4D 49         DC 23,C'NMIC    NEW MICROPROGRAM'
0D0F 43 20 20
0D12 20 4E 45
0D15 57 20 4D
0D18 49 43 52
0D1B 4F 50 52
0D1E 4F 47 52
0D21 41 4D

0D23 0A 0D            DC H'0A0D'
0D25 4E 4D 41         DC 22,C'NMAP    NEW MAPPING RAM'
0D28 50 20 20
0D2B 20 4E 45
0D2E 57 20 4D
0D31 41 50 50
0D34 49 4E 47
0D37 20 52 41
0D3A 4D

0D3B 0A 0D            DC H'0A0D'
0D3D 48 20 20         DC 19,C'H       ASK FOR HELP'
0D40 20 20 20
0D43 20 41 53
0D46 4B 20 46
0D49 4F 52 20
0D4C 48 45 4C
0D4F 50

0D50 0A 0D            DC H'0A0D'
0D52 2A 20 20         DC 27,C'*       DELETE COMMAND AND R'
0D55 20 20 20
0D58 20 44 45
0D5B 4C 45 54
0D5E 45 20 43
0D61 4F 4D 4D
0D64 41 4E 44
0D67 20 41 4E
0D6A 44 20 52
0D6D 45 53 54         DC 6,C'ESTART'
0D70 41 52 54
0D73 0A 0D            DC H'0A0D'
0D75 4C 4D 49         DC 24,C'LMIC    LOAD MICROPROGRAM'
0D78 43 20 20
MOSTEK F8 CROSS ASSEMBLER  THE CITY UNIVERSITY  LONDON,  PAGE  18
```

```
0D7B 20 4C 4F
0D7E 41 44 20
0D81 4D 49 43
0D84 52 4F 50
0D87 52 4F 47
0D8A 52 41 4D
0D8D 0A 0D          DC H'0A0D'
0D8F 4C 4D 41       DC 23,C'LMAP   LOAD MAPPING RAM'
0D92 50 20 20
0D95 20 4C 4F
0D98 41 44 20
0D9B 4D 41 50
0D9E 50 49 4E
0DA1 47 20 52
0DA4 41 4D

0DA6 0A 0D          DC H'0A0D'
0DA8 4D 20 31       DC 27,C'M 1XX  PRINT CONTENTS OF MI'
0DAB 58 58 20
0DAE 20 50 52
0DB1 49 4E 54
0DB4 20 43 4F
0DB7 4E 54 45
0DBA 4E 54 53
0DBD 20 4F 46
0DC0 20 4D 49
0DC3 43 52 4F       DC 22,C'CROPROGRAM ADDRESS 1XX'
0DC6 50 52 4F
0DC9 47 52 41
0DCC 4D 20 41
0DCF 44 44 52
0DD2 45 53 53
0DD5 20 31 58
0DD8 58

0DD9 0A 0D          DC H'0A0D'
0DDB 54 20 58       DC 27,C'T XX   PRINT CONTENTS OF MA'
0DDE 58 20 20
0DE1 20 50 52
0DE4 49 4E 54
0DE7 20 43 4F
0DEA 4E 54 45
0DED 4E 54 53
0DF0 20 4F 46
0DF3 20 4D 41
0DF6 50 50 49       DC 20,C'PPING RAM ADDRESS XX'
0DF9 4E 47 20
0DFC 52 41 4D
0DFF 20 41 44
0E02 44 52 45
0E05 53 53 20
0E08 58 58

0E0A 0A 0D          DC H'0A0D'
```

```
0E0C 4C 44 20        DC 26,C'LD     LOAD VIA PAPER TAPE'
0E0F 20 20 20
0E12 20 4C 4F
0E15 41 44 20
0E18 56 49 41
0E1B 20 50 41
0E1E 50 45 52
0E21 20 54 41
0E24 50 45

0E26 0A 0D           DC H'0A0D'
0E28 44 43 20        DC 27,C'DC 1XX DUMP OUT MICROPROGRA'
0E2B 31 58 58
0E2E 20 44 55
0E31 4D 50 20
0E34 4F 55 54
0E37 20 4D 49
0E3A 43 52 4F
0E3D 50 52 4F
0E40 47 52 41
0E43 4D 20 44        DC 22,C'M DATA FROM 100 TO 1XX'
0E46 41 54 41
0E49 20 46 52
0E4C 4F 4D 20
0E4F 31 30 30
0E52 20 54 4F
0E55 20 31 58
0E58 58

0E59 0A 0D           DC H'0A0D'
0E5B 44 50 20        DC 27,C'DP XX  DUMP OUT MAPPING RAM'
0E5E 58 58 20
0E61 20 44 55
0E64 4D 50 20
0E67 4F 55 54
0E6A 20 4D 41
0E6D 50 50 49
0E70 4E 47 20
0E73 52 41 4D
0E76 44 41 54        DC 20,C'DATA FROM 80 TO XX#'
0E79 41 20 46
0E7C 52 4F 4D
0E7F 20 38 30
0E82 20 54 4F
0E85 20 58 58
0E88 23 23

0E8A 0A 0D     NEW   DC H'0A0D'
0E8C 41 44 44        DC 10,C'ADDRESS: #'
0E8F 52 45 53
0E92 53 53 20
0E95 23

                *
```

```
                    * SUBTRACT 1 FROM THE DATA COUNTER
                    *
0E96 0E             RADI  LR Q,DC
0E97 03                   LR A,QL
0E98 24 FF                AI H'FF'
0E9A 82 08                BC ABLE
0E9C 07                   LR QL,A
0E9D 02                   LR A,QU
0E9E 24 FF                AI H'FF'
0EA0 06                   LR QU,A
0EA1 90 02                BR SOS
0EA3 07             ABLE  LR QL,A
0EA4 0F             SOS   LR DC,Q
0EA5 29 0C 74             JMP HEAT
0EA8 77             RADO    LIS H'07'
0EA9 53                   LR 3,A
0EAA 33             DOLA    DS 3
0EAB 84 07                BZ DENI
0EAD 41                   LR A,1
                    *
                    *
                    * LOOK FOR STRING XX
                    *
                    *
0EAE 8D                   CM
0EAF 94 FA                BNZ DOLA
0EB1 90 E4                BR RADI
0EB3 28 00 44       DENI   PI SPACE
0EB6 29 00 DF             JMP DEL
0EB9 43             CORR   LR A,3
0EBA C1                   AS 1
0EBB 82 04                BC ADD1
0EBD 8E                   ADC
0EBE 90 07                BR STR
0EC0 8E             ADD1   ADC
0EC1 0E                   LR Q,DC
0EC2 02                   LR A,QU
0EC3 1F                   INC
0EC4 06                   LR QU,A
0EC5 0F                   LR DC,Q
0EC6 29 02 0D       STR   JMP ALI
                      END
NUMBER OF ERRORS=  0

MOSTEK F8 CROSS ASSEMBLER  THE CITY UNIVERSITY  LONDON.  PAGE  21
```

```
ABBA    01D5 ABC     00A8 ABCD    0101 ABDC    014D ABLE    0EA3 ADD1    0EC0
ALI     020D AND     0C52 ASH     0077 BA      0223 BACC    0059 BACK    0182
BA11    03E8 BA3     03F8 BB      0016 BBA     0235 BIG     03B9 BOOK    023D
B1      000F B11     0083 B2      0014 CALE    01EA CAP     039F CAR     01DD
CARRY   0C6B CONV    023F CON5    024C CON6    024E CORR    0EB9 CR      003C
CUP     0265 C1      02CC DDT2    0103 DEL     00DF DENI    0EB3 DESK    023A
DMAP    03A5 DMIC    038F DNES    03F5 DOLA    0EAA DUMP    037E ERG     0C16
EXC     0109 EXCH    0C19 F1      0255 HEAD    0C8A HEAT    0C74 HELP    0CFF
HERE    017F HEX     0086 INV     0C49 K1      02C9 LF      0040 LMAP    01A9
LMIC    0178 LOAD    0312 LON     03FB LOOB    0093 LOOM    01C7 LOOP    019B
L1      019E L2      01CA L3      0C0B MAP     030E MAPP    0115 MAPST  =0A00
MAP1    01AE MART    0C1B MES     0252 MIC     02C7 MICC    0112 MICST  =0400
MLOAD   0CCD MORE    0186 NEW     0E8A NOT     0184 NT03    0030 NT05    003B
NT04    0036 NULL    0362 N1      0050 OUT     002C PAP     02E4 PARI    007B
PARS    0071 PEN     005A PER     005C PLOAD   0CEA PL6     035F POR     02BA
POT     0171 POWE    0097 PRINT   009D RADI    0E96 RADO    0EA8 RSA     03D8
RSB     03D6 RS1     0367 RS2     0397 RS3     03AD RS4     03B7 RS5     02A7
RS6     02C4 RS7     03C2 RS8     03BF SEND    000D SILK    016A SOLD    03EE
SOS     0EA4 SPACE   0044 START   009A STR     0EC6 SWPZ    020A S1      0323
S2      032A S3      032F S4      035A S5      033B S6      0330 TA      029C
TELE    0C6C THO     0091 TIMER   0067 TN1     01DA TN2     0262 TTYIN   0049
TTYOUT  0003 T1      00E7 T2      0133 T4      0106 T44     0260 T5      010F
T55     01D8 T6      0118 T7      014F VDU     010C VUD     0C00 WHAT    0C78
WRN     02E7 YES     0237 YESS    02E1 Z1      02D7

                END
NUMBER OF ERRORS=  0
```

APPENDIX D

UMM-16 USER'S MANUAL

THE CITY UNIVERSITY

USER MICROPROGRAMMABLE BIT-SLICE MICROCOMPUTER

(UMM-16)

USER'S MANUAL

by

Dogan Ibrahim

Northampton Square London EC1V 0HB
telephone: 01-253 4399 telex: 263896

81

UMM-16 MICROCOMPUTER

USER'S MANUAL

by

D. Ibrahim

The City University, 1980

CONTENTS

INTRODUCTION

The UMM-16 is an 8/16 bit user microprogrammable microcomputer designed from bit-slice bipolar devices and supported by a large number of bipolar MSI and SSI logic elements. The UMM-16 consists of two basic systems : the 'Bit-Slice Microcomputer' and the 'Support Processor'. The Bit-Slice system is the user microprogrammable microcomputer and the Support Processor is an F8 microcomputer used to load the user defined microinstructions into the control memory of the Bit-Slice system. There are 62 general purpose fixed instructions built into the Bit-Slice system and many more instructions can be invented and implemented in the control memory of the Bit-Slice system.

This manual gives a detailed explanation of the fixed instructions of the Bit-Slice Microcomputer . The system is supported by the "UMM-16 Operating System" which resides in the PROM part of the main memory and is programmed by using the available fixed instructions. Although there are no assemblers available for the UMM-16 instruction set, a mnemonic is defined for every instruction to ease the programming.

Note: It is important that the user should read section 4.3 of this manual before attempting to connect the system to a mains supply.

ARCHITECTURE OF UMM-16

UMM-16 is an 8/16 bit* user imcroprogrammable microcomputer designed with bit-slice Bipolar devices. A block diagram of the internal architecture of UMM-16 is shown in Figure 1. Data and instructions are transmitted along the 16 bit bidirectional Data Bus. The addresses for the memory are sent along the 16 bit Address Bus.

The Arithmetic Processing Unit (APU) is the subsystem where computing and data manipulation are carried out. Arithmetic and logical operations, data shifting and rotation and the generation of status signals are all functions of the APU.

The Computer Control Unit (CCU) is the part that controls all of the other subsystems, fetches and decodes the instructions residing in the memory and sends the appropriate signals to other parts of the computer to perform the required operations. At the heart of the CCU is the microprogram memory (a 48 bit wide memory) which is designed from a combination of PROM and RAM memories. The PROM part of the microprogram memory is loaded with the control data (microinstructions) to execute the fixed macroinstructions (only 16 bits of operation available at present) of the UMM-16. The RAM part of the microprogram memory can be loaded (via a Support Processor) with the user defined microinstructions to execute the macroinstructions invented by the user.

The Program Control Unit (PCU) generates the memory addresses, steps through the addresses in the memory, performs branch, conditional branch and subroutine call and return operations under the control of the CCU.

UMM-16 Registers available to the User

All of the registers of UMM-16 are 16 bits wide (except the Status register and the Mask register). The Arithmetic Processing Unit contains 17 internal working registers (16 scratchpad registers and a 'Q' register). In addition to these, 16 more registers "auxiliary registers" are provided making a total of 33 general purpose registers available to the user (see Figure 2). The APU can only perform operations upon any 17 internal working registers. The data in any of the 16 scratchpad registers can be transferred to the auxiliary registers during the execution of certain instructions. Similarly, the data in any auxiliary register can be transferred to any of the 16 scratchpad registers during the execution of certain instructions.

The APU also contains a "Status Register ('S' register)" which contains the four ALU status (carry, overflow, sign and zero). The status register is loaded with new data after every arithmetic or logic operation. The definition of the ALU status bits are as follows:

* The 16 bit fixed instruction set part is described in this manual.

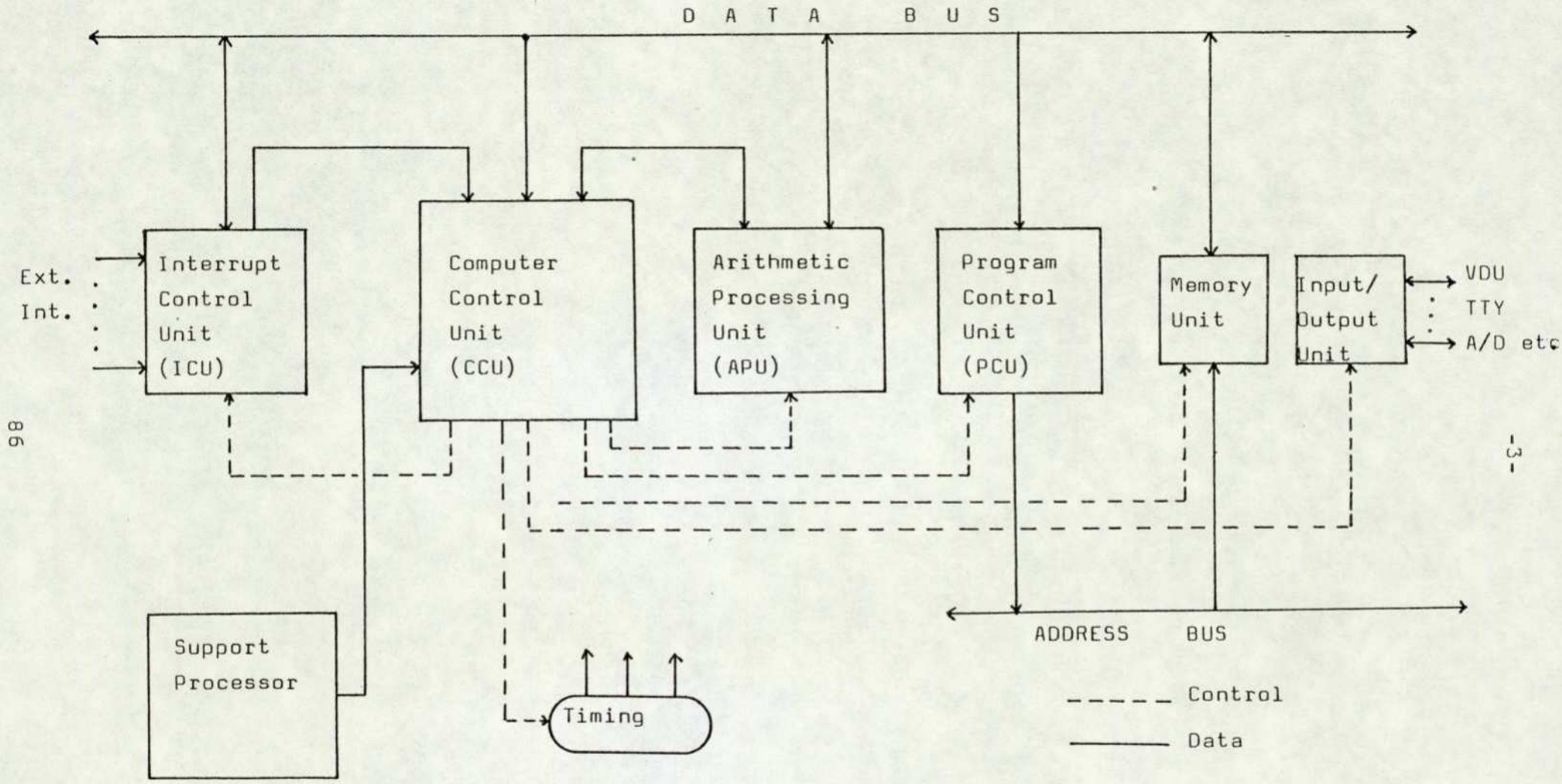


Figure 1. Simplified block diagram of the UMM-16 .

i. The 'carry' flag, C_{n+4} : This flag indicates whether an operation causes a 'carry' into the next higher order digit. If a 'carry' has occured then this flag is set ($C_{n+4} = 1$); otherwise it is reset ($C_{n+4} = 0$).

ii. The 'overflow' flag, OVR: This flag is set (OVR = 1) when two's complement overflow results from an arithmetic operation. In arithmetic operations, the OVR flag is reset (OVR = 0) if two's complement overflow does not occur.

iii. The 'sign' flag, N : The 'sign' flag is set (N = 1) if the most significant bit of the result is set (equal to 1). This indicates that the two's complement number, represented by the bit pattern of the result, is negative. The 'sign' flag is reset (N = 0) if the most significant bit of the result is reset (equal to zero), indicating that the two's complement number represented by the result is zero or positive.

iv. The 'zero' flag, Z : This flag is set (Z = 1) if the result of an arithmetic operation is zero, and is reset (Z = 0) if the result is not zero.

The data in the status register can be transferred to any of the 16 scratchpad registers (or vice-versa) during the execution of certain instructions.

The Program Control Unit of the UMM-16 contains a 16 bit "Program Counter Register" and a 4 word deep 'stack' controlled by an internal stack pointer. The stack is used during subroutine or interrupt operations where it is required to save the current value of the Program Counter Register. The data in the scratchpad registers can be transferred to the Program Counter Register.

The Interrupt Control Unit of the UMM-16 provides two registers to the user: the "Mask register" and the "Counter register". The "Mask register" is used to mask any interrupt inputs and the "Counter register" is used during the Timer Interrupt routines.

Figure 2 shows the registers of UMM-16 that are available to the user. Note that the scratchpad register 'F' is given a special name, the "Data Counter". The "Data Counter" is used during memory reference instructions. It is incremented by one after every instruction that transfers data to or from memory.

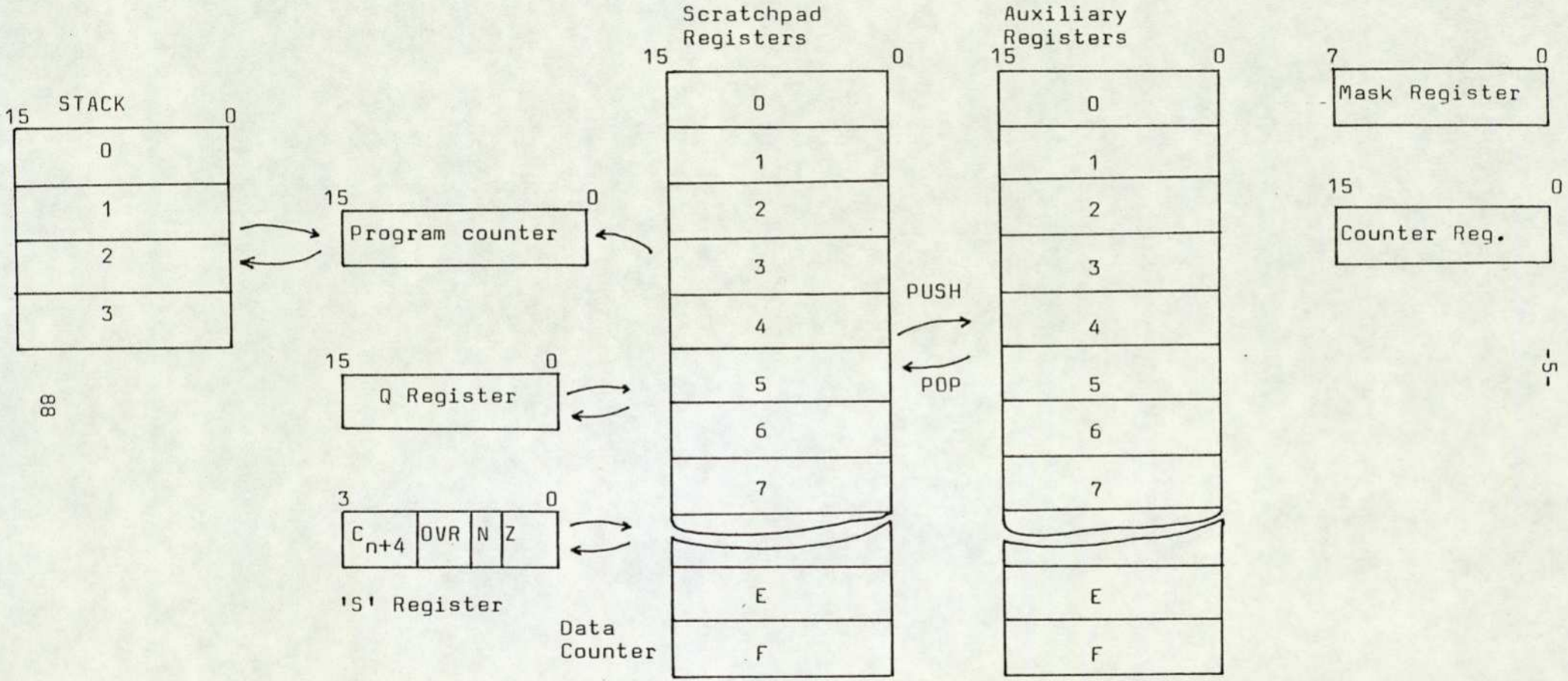


Figure 2. Registers available to the user .

UMM-16 MICRO-COMPUTER

SYSTEM SPECIFICATIONS

(as in October 1979)

1. MAIN COMPUTER

Central Processor	: 4-AM2903 bit slice ALU devices supported by MSI, LSI and SSI Bipolar chips
Data Bus*	: 16 or 8 bits wide
Address Bus	: 16 bits wide (expendable)
No. of Opcode available	: 256
Microprogram memory width	: 48 bits (expendable)
Microprogram memory size	: 512 words (expandable)
Fixed instruction set	: 62 fixed instructions
No. of opcodes used for fixed instructions	: 63
No. of microprogram words used for fixed instructions	: 96
System clock	: Crystal controlled oscillator at 1.000 MHz
System clock period	: 1 micro-second
Memory Size	: 512 words PROM, 1K RAM (expandable)
Memory access time	: 250 ns
I/O channels	: 4 (expandable)
Interface logic signals	: Standard TTL
Interrupt	: 7 external, 1 internal, priority encoded, maskable
No. of user registers	: 33
Stack size	: 4 words (expandable)
Support Software	: UMM-16 operating system
Communications Interface	: Asynchronous 20 mA loop or V24
Communications Baud rate	: 1200, 600, 300 and 110
DC Power Supply	: +5 ±5%, 10A ±12 ±10%, 0.5A ±15 ±10%, 0.5A

* The fixed instructions are designed for 16 bits of operation

2. Support Processor (F8 Microcomputer)

Central Processor	: MK 3850 CPU chip
CPU Support chips	: MK 3853 SMI chip, MK 3871 PIO chip
Memory size	: 2K words EPPOM (2708s), 2K words RAM (SY2114s)
I/O channels used	: 4
Support software	: 2K monitor (Meta assembler) stored in EPPOM
Communications Interface	: Asynchronous 20 mA loop or V.24
Communications Bond rate	: 1200, 600, 300 and 110
Communications control	: By software
System Clock	: Crystal controlled at 2.000 MHz
System Clock Period	: 0.5 micro-second
DC Power Supply	: Same as the Main computer

SECTION I

UMM-16 FIXED INSTRUCTION SET

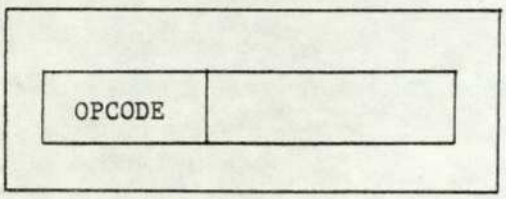
UMM-16 INSTRUCTION SET

This section provides a detailed description of every UMM-16 instruction. All instructions are one, two or three words in length. The first word always contains the opcode for the instruction. The instruction formats are described below.

Single word instructions

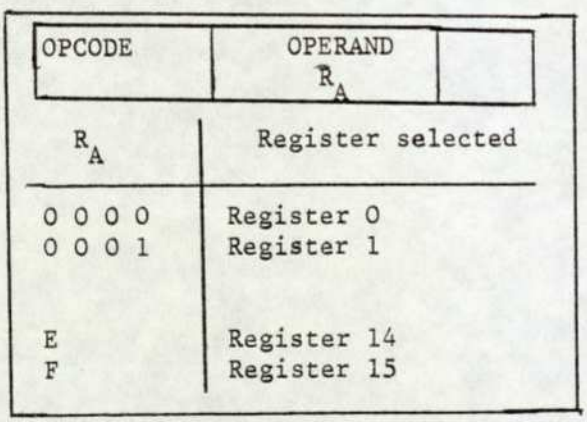
This type of instructions occupy only one location in the memory. They are divided into three classes depending upon the operand required.

Class 1 - no operand: This type of instructions do not require any operand to be specified in the machine code, such as subroutine return and other data manipulation instructions using only the 'Q' register.



A single word class 1 instruction

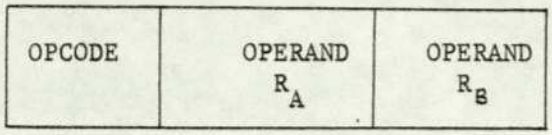
Class 2 - one operand: This type of instructions require only one operand to be specified in the machine code, such as incrementing or decrementing a register.



R_A	Register selected
0 0 0 0	Register 0
0 0 0 1	Register 1
E	Register 14
F	Register 15

A single word class 2 instruction

Class 3 - two operands: This type of instructions require two operands to be specified in the machine code, such as 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.



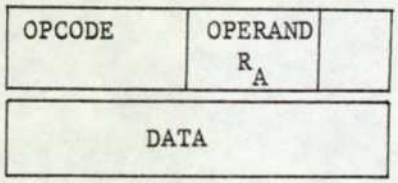
Opcode = 8 bits
Operand R_A = 4 bits
Operand R_B = 4 bits

A single word Class 3 instruction

Two word instructions

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

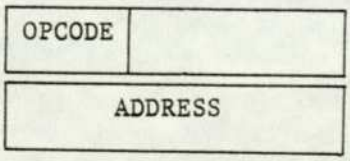
Class 1 - second word is a data : In this type of instructions, the second word is a data and it operates upon the register addressed by the R_A field of the operand, such as loading a data into the scratchpad register addressed by the R_A field.



Opcode = 8 bits
Operand R_A = 4 bits
Data = 16 bits

A two word class 1 instruction

Class 2 - second word is an address: In this type of instructions, the second word is a jump address or an address for an input/output device.

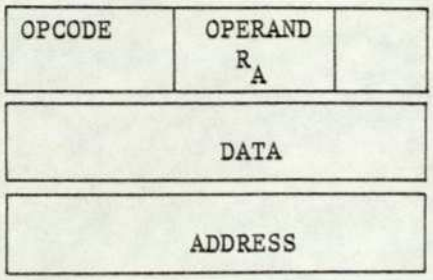


Opcode = 8 bits
Address = 16 bits (8 bits for I/O)

A two word class 2 instruction

Three word instructions

This type of instructions occupy three successive locations in the memory. The first word specifies the operation he performed and the register number. The second word is a 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' is performed to this address, such as 'jump on equal to'.



Opcode = 8 bits
Operand R_A = 4 bits
Data = 16 bits
Address = 16 bits

A three word instruction

Abbreviations

C_{n+4} or C	The carry flag
D_n	nth bit of the scratchpad register
INTE	Interrupt enable flip flop
N	The sign flag
n	Any 8 bit data
nn	Any 16 bit data
OVR or O	The overflow flag
PC	The Program Counter Register
Q	The 'Q' register
R_A	The scratchpad register addressed by the contents of 4 bit operand R_A
R_A^1	The auxiliary register addressed by the contents of 4 bit operand R_A
R_B	The scratchpad register addressed by the contents of 4 bit operand R_B
R_B^1	The auxiliaryregister addressed by the contents of 4 bit operand R_B
R_D	The scratchpad register D
R_F	The scratchpad register F (The Data Counter)
S	The status register
SP	The stack pointer
Z	The zero flag
(R)	Contents of register R
$(\overline{R})$	Complement of the contents of register R
((R))	Contents of location addressed (pointed to) by register R
$\longleftarrow$	Receives
$\longrightarrow$	Goes to
$\longleftrightarrow$	Is exchanged with
+	Arithmetic addition
-	Arithmetic subtraction
.OR.	Logical OR
.AND.	Logical AND
.EX-OR.	Logical exclusive -OR
X	Four bit don't care entry

AND SCRATCHPAD REGISTERS

ANDR

Binary Format

Words	Clock Periods
1	3

0 0 0 0 0 0 0 1	R_A	R_B	((PC))

Hexadecimal Format: 01

Operation: $R_A \leftarrow (R_A)$ .AND. (R_B)

Description: The scratchpad registers addressed by the R_A and R_B fields of the operand are AND'ed. The result is stored in the register addressed by the R_A field.

Status Flags: C_{n+4} and OVR are set to zero. N and Z are modified to reflect the results of the operation

Example: Assume that the scratchpad registers 1 and 2 contain the hexadecimal data 0005 and 0011 respectively. Then after executing the machine code 0112 register 1 will contain 0001.

R_1: 0005
R_2: 0011 ANDR 1,2
R_1: 0001

Similarly, if the machine code 0121 is executed then register 2 will contain the data 0001.

OR SCRATCHPAD REGISTERS

Binary Format

Words	Clock Periods
1	3

0 0 0 0 0 0 1 0	R_A	R_B	((PC))

Hexadecimal Format: 02

Operation: $R_A \leftarrow (R_A)$ .OR. (R_B)

Description: The scratchpad registers addressed by the R_A and R_B fields of the operandare OR'ed. The result is stored in the register addressed by the R_A field.

Status Flags: C_n+4 and OVR are set to zero. N and Z are modified to reflect the results of the operation.

Example: Assume that the scratchpad registers 1 and 2 contain the hexadecimal data 0004 and 0110 respectively. Then after executing the machine code 0212 register 1 will contain 0114.

R_1: 0004
R_2: 0110 ORR 1,2
R_1: 0114

Similarly, if the machine code 0221 is executed then register 2 will contain the data 0114.

EXCLUSIVE-OR SCRATCHPAD REGISTERS — EXR

Binary Format	Words	Clock Period
0 0 0 0 0 0 1 1 | R_A | R_B	1	3

Hexadeciamal Format: 03

Operation: $R_A \leftarrow (R_A)$ .EX-OR. (R_B)

Description: The scratchpad registers addresses by the R_A and R_B fields of the operand are exclussive -OR ed. The result is stored in the register addressed by the R_A field.

Status Flags: C_{n+4} and OVR are set to zero. N and Z are modified to reflect the results of the operation.

Example: Assume that the scratchpad registers 1 and 2 contain the hexadecimal data 0011 and 0101 respectively. Then after executing the machine code 0312 register 1 will contain 0110.

R_1: 0011
R_2: 0101 EXR 1,2
R_1: 0110

Similarly if the machine code 0321 is executed then register 2 will contain the data 0110.

1's COMPLEMENT SCRATCHPAD REGISTER — COMR

Binary Format	Words	Clock Period
0 0 0 0 0 1 0 0 | R_A | X	1	3

Hexadecimal Format: 04

Operation: $R_A \leftarrow (\overline{R_A})$

Description: The scratchpad register addressed by the R_A field of the operand is complemented

Status Flags: C_{n+4}, OVR, N, Z are modified to reflect the results of the operation.

Example: Assume that the scratchpad register 2 contains the hexadecimal data 0110. Then after executing the machine code 042X (X=don't care) register 2 will contain 1001.

R_2: 0110 COMR 2,0
R_2: 1001

1's COMPLEMENT Q REGISTER — COMQ

Binary Format	Words	Clock Periods
0 0 0 0 0 1 0 1 | X | X ((PC))	1	3

Hexadecimal Format: 05

Operation: $Q \leftarrow (\overline{Q})$

Description: The Q register is complemented

Status Flag: C_{n+4}, OVR, N, Z are modified to reflect the results of the operation

Example: Assume that the Q register contains the hexadecimal data 0110. Then after executing the machine code 05XX (X = don't care) the Q register will contain 1001.

```
Q: 0110   COMQ
   ----
Q: 1001
```

2's COMPLEMENT SCRATCHPAD REGISTER

Binary Format	Words	Clock Periods
0 0 0 0 0 1 1 0 | R_A | X ((PC))	1	3

Hexadecimal Format: 06

Operation: $R_A \longleftarrow (\overline{R_A}) + 1$

Description: The scratchpad register addressed by the R_A field of the operand is complemented and incremented by one.

Status Flags: C_{n+4}, OVR, N, Z are modified to reflect the results of the operation.

Example: Assume that the scratchpad register 2 contains the hexadecimal data 0011. The after executing the machine code 062X (X = don't care) register 2 will contain 1101.

```
R2: 0011   complement
    ----
    1100
  + 1      increment
    ----
R2: 1101
```

AND WITH SCRATCHPAD REGISTER IMMEDIATE ANDI

Binary Format	Words	Clock Period
0 0 0 0 0 1 1 1 | R_A | X ((PC))	2	3
16 BIT DATA ((PC)+1)		

Hexadecimal Format: 07

Operation: $R_A \leftarrow (R_A)$ .AND. ((PC)+1)

Description: The scratchpad register addressed by the R_A field of the operandis AND'ed with the contents of the second word of the instruction, (PC)+1. The result is placed in the scratchpad register.

Status Flags: C_{n+4} and OVR are set to zero. N and Z are modified to reflect the results of the operation.

Example: Assume that the scratchpad register 2 contains the hexadecimal data 0001. Assume that memory location 0400 and 0401 contain the data 072X (X = don't care) and 0033 respectively. Then after executing the machine code at the memory location 0400, the register 2 will contain the data 0001.

R_2:0001

Immediate Data:0033 ANDI 2, H '0033'

R_2:0001

OR WITH SCRATCHPAD REGISTER IMMEDIATE ORI

Binary Format				Words	Clock Period
0 0 0 0 1 0 0 0	R_A	X	((PC))	2	3
16 BIT DATE			((PC)+1)		

Hexadecimal Format: 08

Operation: $R_A \leftarrow (R_A)$ .OR. ((PC)+1)

Description: The scratchpad register addressed by the R_A field of the operand is OR'ed with the contents of the second word of the instruction, (PC)+1. The result is placed in the scratchpad register.

Status Flags: C_{n+4} and OVR are set to zero. N and Z are modified to reflect the results of the operation

Example: Assume that the scratchpad register 2 contains the hexadecimal data 0010. Assume that memory location 0400 and 0401 contain the data 082X (X = don't care) and 0033 respectively. Then after executing the machine code at the memory location 0400, the register 2 will contain the data 0033.

R_2: 0010

Immediate Data: 0033 ORI 2, H '0033'

R_2: 0033

EXCLUSIVE - OR WITH SCRATCHPAD REGISTER IMMEDIATE EXI

Binary Format				Words	Clock Period
0 0 0 0 1 0 0 1	R_A	X	((PC))	2	3
16 BIT DATA			((PC)+1)		

Hexadecimal Format: 09

Operation: $R_A \leftarrow (R_A)$ .EX-OR. ((PC)+1)

Description: The scratchpad register addressed by the R_A field of the operand is exclusive - OR'ed with the contents of the second word of the instruction, (PC)+1. The result is placed in the scratchpad register.

Status Flags: C_{n+4} and OVR are set to zero. N and Z are modified to reflect the results of the operation.

Example: Assume that the scratchpad register 2 contains the hexadecimal data 0011. Assume that memory location 0400 and 0401 contain the data 092X (X = don't care) and 0101 respectively. Then after executing the machine code at the memory location 0400, the register 2 will contain the data 0110.

R_2: 0011

Immediate Data: 0101 EXI 2, H '0101'

R_2: 0110

ADD SCRATCHPAD REGISTERS ADDR

Binary Format	Words	Clock Period
0 0 0 0 1 0 1 0 | R_A | R_B ((PC))	1	3

Hexadecimal Format: 0A

Operation: $R_A \leftarrow (R_A)+(R_B)$

Description: The scratchpad registers addressed by the R_A and R_B fields of the operand are ADD'ed. The result is stored in the register addressed by the R_A field.

Status Flags: C_{n+4}, OVR, N, Z are modified to reflect the results of the operation.

Example: Assume that the scratchpad registers 1 and 2 contain the hexadecimal data 0006 and 0002 respectively. The after executing the machine code 0A12 register 1 will contain 0008

R_1: 0006
R_2: 0002 ADDR 1,2
R_1: 0008

Similarly, if the machine code 0A21 is executed, then register 2 will contain the data 0008.

ADD Q REGISTER TO THE SCRATCHPAD REGISTER ADDQ

Binary Format:	Words	Clock Period
0 0 0 0 1 0 1 1 | R_A | R_B ((PC))	1	3

Hexadecimal Format: 0B

Operation: $R_B \leftarrow (R_A)+(Q)$

Description: The scratchpad register addressed by the R_A field of the operand is ADD'ed to the Q register. The result is stored in the register addressed by the R_B field.

Status Flags: C_{n+4}, OVR, N, Z are modified to reflect the result of the operation.

Example: Assume that the scratchpad register 1 contains the hexadecimal data 0006. Assume that the Q register contains the hexadecimal data 0002. The after executing the machine code 0B12 register 2 will contain 0008.

R_1: 0006
Q: 0002 ADDQ 1,2
R_2: 0008

Similarly, if the machine code 0B13 is executed, then register 3 will contain the data 0008.

SUBSTRACT SCRATCHPAD REGISTERS SUBR

Binary Format	Words	Clock Periods
0 0 0 0 1 1 0 0 | R_A | R_B ((PC))	1	3

Hexadecimal Format: 0C

Operation: $R_A \leftarrow (R_A) - (R_B)$

Description: The scratchpad registers addressed by the R_A and R_B fields of the operand are SUBTRACT'ed. The result is stored in the register addressed by the R_A field.

Status Flags: C_{n+4}, OVR, N, Z are modified to reflect the results of the operation.

Example: Assume that the scratchpad registers 1 and 2 contain the hexadecimal data 0006 and 0002 respectively. Then after executing the machine code 0C12 register 1 will contain 0004.

R_1: 0006
R_2: 0002 SUBR 1,2
R_1: 0004

Similarly, if the machine code 0C21 is executed, then register 2 will contain the data FFFC.

INCREMENT SCRATCHPAD REGISTER INCR

Binary Format	Words	Clock Periods
0 0 0 0 1 1 0 1 | R_A | X ((PC))	1	3

Hexadecimal Format: 0D

Operation: $R_A \leftarrow (R_A) + 1$

Description: The contents of the scratchpad register addressed by the R_A field of the operand is incremented by one.

Status Flags: C_{n+4}, OVR, N, Z are modified to reflect the results of the operation.

Example: Assume that the scratchpad register 2 contains the hexadecimal data 0009. Then after executing the machine code 0D2X (X = don't care) register 2 will contain 000A.

R_2: 0009 INCR 2,0
R_2: 000A

INCREMENT Q REGISTER — INCQ

Binary Format	Words	Clock Periods
0 0 0 0 1 1 1 0 | X | X ((PC))	1	3

Hexadecimal Format: 0E

Operation: $Q \leftarrow (Q)+1$

Description: The contents of the Q register are incremented by one.

Status Flags: C_{n+4}, OVR, N, Z are modified to reflect the results of the operation.

Example: Assume that the Q register contains the hexadecimal data 0009. Then after executing the machine code 0EXX (X = don't care) the Q register will contain 000A.

```
Q: 0009   INCQ
Q: 000A
```

DECREMENT SCRATCHPAD REGISTER — DECR

Binary Format	Words	Clock Periods
0 0 0 0 1 1 1 1 | R_A | X ((PC))	1	3

Hexadecimal Format: 0F

Operation: $R_A \leftarrow (R_A)-1$

Description: The contents of the scratchpad register addressed by the R_A field of the operand is decremented by one.

Status Flags: C_{n+4}, OVR, N, Z are modified to reflect the results of the operation.

Example: Assume that the scratchpad register 2 contains the hexadecimal data 0007. Then after executing the machine code 0F2X (X = don't care) register 2 will contain 0006.

```
R2: 0007   DECR 2,0
R2: 0006
```

DECREMENT Q REGISTER DECQ

Binary Format				Words	Clock Periods
0 0 0 1 0 0 0 0	X	X	((PC))	1	3

Hexadecimal Format: 10

Operation: $Q \leftarrow (Q)-1$

Description: The contents of the Q register are decremented by one.

Status Flags: C_{n+4}, OVR, N, Z are modified to reflect the results of the operation.

Example: Assume that the Q register contains the hexadecimal data 0007. Then after executing the machine code 10XX (X = don't care) the Q register will contain 0006.

```
Q: 0007
   ----  DECQ
Q: 0006
```

CLEAR THE Q REGISTER CLRQ

Binary Format:				Words	Clock Periods
0 0 0 1 0 0 0 1	X	X	((PC))	1	3

Hexadecimal Format: 11

Operation: $Q \leftarrow 0$

Description: The Q register is cleared.

Status Flags: C_{n+4} and OVR are set to zero. N and Z are modified to reflect the results of the operation

Example: When the machine code 11 XX (X = don't care) is executed, the Q register will be cleared.

CLEAR SCRATCHPAD REGISTER — CLRR

Binary Format		Words	Clock Periods
0 0 0 1 0 0 1 0 | R_A | X	((PC))	1	3

Hexadecimal Format: 12

Operation: $R_A \leftarrow 0$

Desription: The contents of the scratchpad register addressed by the R_A field of the operandis cleared.

Status Flags: C_{n+4} and OVR are set to zero. N and Z are modified to reflect the results of the operation.

Example: When the machine code 123X (X = don't care) is executed, register 3 will be cleared.

ADD TO SCRATCHPAD REGISTER IMMEDIATE — ADDI

Binary Format		Words	Clock Periods
0 0 0 1 0 0 1 1 | R_A | X	((PC))	2	3
16 BIT DATA	((PC)+1)		

Hexadecimal Format: 13

Operation: $R_A \leftarrow (R_A) + ((PC)+1)$

Description: The scratchpad register addressed by the R_A field is the operandis ADD'ed to the contents of the second word of the instruction, (PC)+1. The result is placed in the scratched register.

Status Flags: C_{n+4}, OVR, N, Z are modified to reflect the results of the operation.

Example: Assume that the scratchpad register 2 contains the hexadecimal data 0003. Assume that memory location 0400 and 0401 contain the data 123X (X = don't care) and 0005 respectively. Then after executing the machine code at the memory location 0400, the register 2 will contain the data 0008.

R_2: 0003
Immediate Data: 0005 ADDI 2, H '0005'
R_2: 0008

SUBTRACT FROM SCRATCHPAD REGISTER IMMEDIATE — SUBI

Binary Format	Words	Clock Periods
0 0 0 1 0 1 0 0 | R_A | X ((PC))	2	3
16 BIT DATA ((PC)+1)		

Hexadecimal Format: 14

Operation: $R_A \leftarrow (R_A) - ((PC)+1)$

Description: The scratchpad register contents addressed by the R_A field of the operand are SUBTRACT'ed from the contents of the second word of the instruction, (PC)+1. The result is placed in the scratchpad register.

Status Flags: C_{n+4}m OVR, N, Z are modified to reflect the results of the operation.

Example: Assume that the scratchpad register 2 contains the hexadecimal data 0009. Assume that memory location 0400 and 0401 contain the data 143X (X = don't care) and 0003 respectively. Then after executing the machine code at the memory location 0400, the register 2 will contain the data 0006.

R_2: 0009
Immediate Data: 0003 SUBI 2, H '0003'
R_2: 0006

LOAD SCRATCHPAD REGISTER IMMEDIATE — LI

Binary Format	Words	Clock Periods
0 0 0 1 0 1 0 1 | R_A | X ((PC))	2	3
16 BIT DATA ((PC)+1)		

Hexadecimal Format: 15

Operation: $R_A \leftarrow ((PC)+1)$

Description: The scratchpad register addressed by the R_A field of the operand is loaded with the contents of the second word of the instruction, (PC)+1.

Status Flags: No status flags are modified.

Example: Assume that the memory location 0400 and 0401 contain the hexadecimal 153X (X = don't care) and 1101 respectively. Then after executing the machine code at the memory location 0400, the register 3 will contain the data 1101.

LOAD Q REGISTER IMMEDIATE LIQ

Binary Format		Words	Clock Periods
0 0 0 1 0 1 1 0 | X | X	((PC))	2	3
16 BIT DATA	((PC)+1)		

Hexadecimal Format: 16

Operation: $Q \leftarrow ((PC)+1)$

Description: The Q register is loaded with the contents of the second word of the instruction, (PC)+1.

Status Flags: No status flags are modified.

Example: Assume that the memory location 0400 and 0401 contain the hexadecimal data 16XX (X = don't care) and 1101 respectively. Then after executing the machine code at the memory location 0400, the Q register will contain the data 1101.

MOVE SCRATCHPAD REGISTERS MOVR

Binary Format	Words	Clock Periods
0 0 0 1 0 1 1 1 | R_A | R_B	1	4

Hexadecimal Format: 17

Operation: $R_A \leftarrow (R_B)$

Description: The contents of the scratchpad register addresses by the R_B field of the operand are transferred to the scratchpad register addresses by the R_A field. The contents of the scratchpad register addresses by the R_B field are not destroyed.

Status Flags: No status flags are modified.

Example: Assume that the scratchpad registers 1 and 2 contain the hexadecimal data 1101 and 0010 respectively. Then after executing the machine code 1712, register 1 will contain the data 0010.

MOVE Q REGISTER TO SCRATCHPAD REGISTER — MOVQ

Binary Format	Words	Clock Periods
0 0 0 1 1 0 0 0 | R_A | X	1	3

Hexadecimal Format: 18

Operation: $R_A \leftarrow (Q)$

Description: The contents of the Q register are transfered to the scratchpad register addressed by the R_A field of the operand. The contents of the Q register are not destroyed.

Status Flags: No status flags are modified.

Example: Assume that the Q register contains the hexadecimal data A1F0. Then after executing the machine code 182X (X = don't care) the scratchpad register 2 will contain the data A1F0.

MOVE SCRATCHPAD REGISTER TO Q REGISTER — LRQ

Binary Format	Words	Clock Periods
0 0 0 1 1 0 0 1 | R_A | X	1	3

Hexadecimal Format: 19

Operation: $Q \leftarrow (R_A)$

Description: The contents of the scratchpad register addressed by the R_A field of the operand are transfered to the Q register. The contents of the scratchpad register are not destroyed.

Status Flags: No status flags are modified.

Example: Assume that the scratchpad register 3 contains the hexadecimal data 11F0. Then after executing the machine code 193X (X = don't care) the Q register will contain the data 11F0.

MOVE SCRATCHPAD REGISTER TO AUXILIARY REGISTER — PUSH

Binary Format				Words	Clock Periods
0 0 0 1 1 0 1 0	R_A^1	R_B	((PC))	1	3

Hexadecimal Format: 1A

Operation: $R_A^1 \leftarrow (R_B)$

Description: The contents of the scratchpad register addressed by the R_B field of the operand are transfered to the auxiliary register addressed by the R_A^1 field of the opland. The contents of the scratched register are not destroyed.

Status Flags: No status flags are modified.

Example: Assume that the scratchpad register 3 contains the hexadecimal data A011. Then after executing the machine code 1A23 the auxiliary register 2 will contain the data A011.

MOVE AUXILIARY REGISTER TO SCRATCHPAD REGISTER — POP

Binary Format				Words	Clock Periods
0 0 0 1 1 0 1 1	R_A^1	R_B	((PC))	1	3

Hexadecimal Format: 1B

Operation: $R_B \leftarrow (R_A^1)$

Description: The contents of the auxiliary register addresses by the R_A^1 field of the operand are transfered to the scratchpad register addressed by the R_B field of the opland. The contents of the auxiliary register are not destroyed.

Status Flags: C_{n+4} and OVR are set to zero. N and Z are modified to reflect the results of the operation.

Example: Assume that the auxiliary register 2 contains the hexadecimal data 110E. Then after executing the machine code 1B23 the scratchpad register 3 will contain the data 110E.

MOVE THE STATUS REGISTER TO SCRATCHPAD REGISTER (SAVE STATUS BITS) MOVS

Binary Format	Words	Clock Periods
0 0 0 1 1 1 0 0 | R_A | X ((PC))	1	3

Hexadecimal Format: 1C

Operation: $R_A \leftarrow (S)$

Description: The contents of the status register (4 bits) are transfered to the highest four bits of the scratchpad register addressed by the R_A field of the operand. The contents of the status register are not destroyed.

Status Flags: No status flags are modified.

Example: Assume that the status register contains the 4 bit data 1011. Then after executing the machine code IC3X (X = don't care) the highest four bits of the scratchpad register 3 will contain the data 1011. The lower 12 bits of register 3 will be loaded with all 1s (hexadecimal FFF).

MOVE SCRATCHPAD REGISTER TO STATUS REGISTER (RESTORE STATUS BITS) LRS

Binary Format	Words	Clock Periods
0 0 0 1 1 1 0 1 | R_A | X ((PC))	1	3

Hexadecimal Format: 1D

Operation: $S \leftarrow (R_A)$

Description: The highest 4 bits of the scratchpad register addressed by the R_A field of the operandare transfered to the status register. The contents of the scratchpad register are not destroyed.

Status Flags: Loaded with the data in the highest 4 bits of R_A.

Example: Assume that the highest four bits of the scratchpad register 3 contain the binary data 1011. Then after executing the machine code 1D3X (X = don't care) the status register will be loaded with the 4 bit data 1011.

OUTPUT FROM SCRATCHPAD REGISTER OUTS

Binary Format				Words	Clock Periods
0 0 0 1 1 1 1 0	R_A	X	((PC))	2	4
Device Address	X	X	((PC)+1)		

Hexadecimal Format: IE

Operation: Output Port $\leftarrow (R_A)$

Description: The contents of the scratchpad register addressed by the R_A field of the operand are copied to the output port addressed by the higher byte of the second word of the instruction. The contents of the scratchpad register are not destroyed.

Status Flags: No status flags are modified.

Example: Assume that the scratchpad register 3 contains the hexadecimal data E021. Assume that the memory location 0400 and 0401 contain the hexadeciaml data IE3X and 07XX (X = don't care) respectively. Then after executing the machine code at the memory location 0400, the 'Output Port 7' will contain the hexadecimal data E021.

INPUT TO SCRATCHPAD REGISTER INS

Binary Format	Words	Clock Periods
0 0 0 1 1 1 1 1 | R_A | X ((PC))	2	4
Device Address | X | X ((PC+1)		

Hexadecimal Format: 1F

Operation: $R_A \leftarrow$ (Input Port)

Description: The contents of the input port (addressed by the higher byte of the second word) are transfered to the scratchpad register addressed by the R_A field of the operand.

Status Flags: C_{n+4} and OVR are set to zero. N and Z are modified to reflect the results of the operation

Example: Assume that the 'input port 6' contains the hexadecimal data 2101. Assume that the memory location 0400 and 0401 contain the hexadecimal data 1F3X and 06XX (X = don't care) respectively. Then after executing the machine code at the memory location 0400, the scratchpad register 3 will contain the hexadecimal data 2101.

ADD MEMORY TO SCRATCHPAD REGISTER ADDM

Binary Format	Words	Clock Periods
0 0 1 0 0 0 0 0 | R_A | X | ((PC))	1	5

Hexadecimal Format: 20

Operation: $R_A \leftarrow (R_A) + ((R_F))$

$R_F \leftarrow (R_F) + 1$

Description: The data at the memory location addressed by the scratchpad register F (Data Counter) is ADD'ed to the scratchpad register addressed by the R_A field of the operand. The result is stored in the scratchpad addressed by the R_A field of the opland. The contents of the scratchpad register F are incremented by one after the execution of this instruction. The data in the specified memory location is not destroyed.

Status Flags: C_{n+4}, OVR, N, Z are modified to reflect the results of this operation.

Example: Assume that the scratchpad register F contains the hexadecimal address 0400. Assume that the memory location 0400 contains the hexadecimal data 0101. Also assume that the scratchpad register 3 contains the hexadecimal data 0011. Then after executing the machine code 203X (X = don't care) the scratchpad register 3 will contain the hexadecimal data 0112 (0101 + 0011 = 0112) and the scratchpad register F will contain the address 0401.

MOVE MEMORY TO SCRATCHPAD REGISTER MOVM

Binary Format	Words	Clock Periods
0 0 1 0 0 0 0 1 | R_A | X ((PC))	1	5

Hexadecimal: 21

Operation: $R_A \leftarrow ((R_F))$

$R_F \leftarrow (R_F) + 1)$

Description: The data at the memory location addressed by the scratchpad register F (Data Counter) is loaded to the scratchpad register addressed by the R_A field of the operand. The contents of the scratchpad register F are incremented by one after the execution of this instruction. The data in the specified memory location is not destroyed.

Status Flags: C_{n+4} and OVR are set to zero. N and Z are modified to reflect the results of this operation.

Example: Assume that the scratchpad register F contains the hexadecimal address 0400. Assume that the memory location 0400 contains the hexadecimal data 0210. Then after executing the machine code 213X (X = don't care) the scratchpad register 3 will contain the hexadecimal data 0210 and the scratchpad register F will contain the address 0401.

WAIT

WAIT

Binary Format	Words	Clock Periods
0 0 1 0 0 0 1 0 | X | X ((PC))	1	-

Hexadecimal Format: 22

Operation: WAIT

Description: The processor enters the WAIT state. Although the system clock runs it is disabled from the processor when this instruction is executed.

A ready signal must be sent to the processor externally to exit from the WAIT condition. More details on the use of this instruction are given in Section III of this manual.

Status Flags: No status flags are modified.

COMPARE SCRATCHPAD REGISTER IMMEDIATE

CI

Binary Format	Words	Clock Periods
0 0 1 0 0 0 1 1 | R_A | X ((PC))	2	3
16 BIT DATA ((PC) + 1)		

Hexadecimal Format: 23

Operation: (R_A) - ((PC) + 1)

Description: The contents of the scratchpad register addressed by the R_A field of the operand are SUBTRACT'ed from the contents of the second word of the instruction, (PC) + 1. The contents of the scratchpad register and the contents of word 2 of the instruction are not destroyed.

Status Flags: C_{n+4}, OVR, N, Z are modified to reflect the results of the operation.

COMPARE SCRATCHPAD REGISTERS CR

Binary Format	Words	Clock Periods
0 0 0 1 0 0 1 0 0 | R_A | R_B ((PC))	1	3

Hexadecimal Format: 24

Operation: $(R_A) - (R_B)$

Description: The scratchpad registers addressed by the R_A and R_B fields of the opland are SUBTRACT'ed The contents of the scratchpad registers are not altered but the status flags are set or reset accordingly.

Status Flags: C_{n+4}, OVR, N, Z are modified to reflect the results of the operation.

COMPARE SCRATCHPAD REGISTER WITH MEMORY CM

Binary Format	Words	Clock Periods
0 0 1 0 0 1 0 1 | R_A | X ((PC))	1	5

Hexadecimal Format: 25

Operation: $(R_A) - ((R_F))$

$R_F \leftarrow (R_F + 1$

Description: The data at the memory location addressed by the scratchpad register F (Data Counter) is SUBTRACT'ed from the contents of the scratchpad register addressed by the R_A field of the opland. The contents of the scratchpad register addressed by the R_A field are not altered. Also the data in the specified memory location is not altered, but the contents of the scratchpad register F are incremented by one.

Status Flags: C_{n+4}, OVR, N, Z are modified to reflect the results of this operation.

JUMP UNCONDITIONALLY

Binary Format		Words	Clock Periods
0 0 1 0 0 1 1 0 | X | X	((PC))	2	3
16 BIT ADDRESS	((PC)+1)		

Hexadecimal Format: 26

Operation: PC ← ((PC)+1)

Description: The Program Counter Register is loaded with the 16 bit address contained in the second word of the instruction.

The next instruction to be fetched will be at location ((PC)+1).

Status Flags: No status flags are modified.

JUMP CONDITIONALLY

Binary Format		Words	Clock Periods
0 0 1 0 COND | X | X	((PC))	2	4
16 BIT ADDRESS	((PC+1)		

Operation: PC ← ((PC)+1); if condition true

PC ← (PC)+2 ; if condition false

Description: One of four status flags is tested to meet the condition. If the condition is met then Program Counter Register it loaded with the contents of the address in the second word. Otherwise the Program Counter Register is incremented i.e. loaded with (PC)+2).

Status Flags: No status flags are modified.

Hexadecimal Format:

Instruction	Mneumonic	COND	Jump Condition	Hex. Format
Jump on non Zero	JNZ	0111	Z = 0	27
Jump on Zero	JZ	1000	Z = 1	28
Jump on Positive	JP	1001	N = 0	29
Jump on Negative	JN	1010	N = 1	2A
Jump on non Carry	JNC	1110	$C_{n+4} = 0$	2E

JUMP ON EQUAL TO — JE

Binary Format | Words | Clock Periods

0 0 1 0 1 0 1 1 | R_A | X	((PC))	3	5
16 BIT DATA	((PC)+1)		
16 BIT ADDRESS	((PC)+2)		

Hexadecimal Format: 2B

Operation: PC ← ((PC)+2); if (R_A) = ((PC)+1)

PC ← (PC)+3 ; of $(R_A) \neq$ ((PC)+1)

Description: The contents of the scratchpad register addressed by the R_A field of the operand are compared to the data in the second word of the instruction. If they are equal then the Program Counter Register is loaded with the address in the third word of the instruction. If they are not equal then the Program Counter Register is incremented i.e. loaded with (PC)+3 .

Status Flags: Status flags are modified according to the result of the operation 'Compare scratchpad register Immediate'.

JUMP ON LESS THAN — JL

Binary Format | Words | Clock Periods

0 0 1 0 1 1 0 1 | R_A | X	((PC))	3	5
16 BIT DATA	((PC)+1)		
16 BIT ADDRESS	((PC)+2)		

Hexadecimal Format: 2D

Operation: PC ← ((PC)+2); if $(R_A) <$ ((PC)+1)

PC ← (PC)+3 ; if $(R_A) \geqslant$ ((PC)+1)

Description: The contents of the scratchpad register addressed by the R_A field of the operand are compared to the data in the second word of the instruction. If the contents of the scratchpad register is less than (unsigned data) this data then the Program Counter Register is loaded with the address in the third word of the instruction. If it is greater or equal to the data then the Program Counter Register is incremented i.e. loaded with (PC)+3.

Status Flags: Status flags are modified according to the result of the operation 'Compare Scratchpad register Immediate'.

DECREMENT AND SKIP ON ZERO — DSZ

Binary Format	Words	Clock Periods
0 0 1 0 1 1 0 0 | R_A | X | ((PC))	2	5
16 BIT ADDRESS ((PC)+1)		

Hexadecimal format: 2C

Operation:

$R_A \leftarrow (R_A)-1$

$PC \leftarrow ((PC)+1)$; if $(R_A) \neq 0$

$PC \leftarrow (PC)+2$; if $R_A = 0$

Description: The contents of the scratchpad register addressed by the R_A field of the operand is decremented by one. If it is non zero then the 16 bit address in the second word of the instruction is loaded into the Program Counter Register. If it is zero, the second word is skipped and ((PC)+2) is loaded into the Program Counter Register.

Status Flags: Status flags are modified according to the result of the operation 'decrement scratchpad register'

CALL TO SUBROUTINE — CALL

Binary Format	Words	Clock Periods
0 0 1 0 1 1 1 1 | X | X | ((PC))	2	3
16 BIT ADDRESS ((PC)+1)		

Hexadecimal Format: 2F

Operation:

$SP \leftarrow (SP)+1$
$((SP)) \leftarrow (PC)$
$PC \leftarrow ((PC)+1)$

Description: The current Program Counter contents are pushed onto the stack (4 words deep) and the second word of the instruction is loaded into the Program Counter Register.

1. Stack Pointer is incremented.
2. Current Program Counter Contents are pushed onto the stack location addressed by the Stack Pointer.
3. The 16 Bit address ((PC)+1) is loaded into the Program Counter Register.

Status Flags: No status flags are modified.

MOVE SCRATCHPAD REGISTER TO MEMORY LRM

Binary Format	Words	Clock Periods
0 0 1 1 0 0 0 0 | R_A | X | ((PC))	1	5

Hexadecimal Format: 30

Operation: $((R_F)) \leftarrow (R_A)$

$R_F \leftarrow (R_F)+1$

Description: The contents of the scratchpad register addressed by the R_A field of the operand are transfered to the memory location addressed by the scratchpad register F (Data Counter). The contents of the scratchpad register addressed by the R_A field of the operand are not altered. The contents of the scratchpad register F are incremented by one.

Status Flags: No status flags are modified.

Example: Assume that the scratchpad register F contains the address 0400. Also assume that the scratchpad register 3 contains the hexadecimal data 0211. The after executing the machine code 303X (X = don't care), the memory location 0400 will contain the hexadecimal data 0211. Also the scratchpad register F will be incremented to 0401.

MOVE SCRATCHPAD REGISTER TO PROGRAM COUNTER REGISTER LRP

Binary Format	Words	Clock Periods
0 0 1 1 0 0 0 1 | R_A | X | ((PC))	1	3

Hexadecimal Format: 31

Operation: $PC \leftarrow (R_A)$

Description: The contents of the scratchpad register addressed by the R_A field of the operand are transfered to the Program Counter Register. The contents of the scratchpad regidter are not altered.

Status Flags: No status flags are modified.

Example: Assume that the scratchpad register 2 contains the hexadecimal data 0231. Then after executing the machine code 312X (X = don't care) the Program Counter Register will contain the hexadecimal data 0231.

RETURN FROM SUBROUTINE — RET

Binary Format	Words	Clock Periods
0 0 1 1 0 0 1 0 | X | X ((PC))	1	3

Hexadecimal Format: 32

Operation: SP ← (SP)-1

PC ← ((SP))+1

Description: The Stack Pointer is decremented and the contents of the stack are copied to the Program Counter Register.

Status Flags: No status flags are modified.

ROTATE SCRATCHPAD REGISTER RIGHT 2 POSITIONS — ROTR

Binary Format	Words	Clock Periods
0 0 1 1 0 0 1 1 | R_A | X ((PC))	1	4

Hexadecimal Format: 33

Operation: $D_n \leftarrow (D_n+2)$; $D_{15} \leftarrow (D_1)$

Description: The contents of the scratchpad register addressed by the R_A field of the operand are rotated right two positions.

Status Flags: C_{n+4} and OVR are set to zero. N and Z are modified to reflect the results of the operation.

Example: Assume that the scratchpad register 1 contains the hexadecimal data 0201. Then after executing the machine code 331X (X=don't care) the scratchpad register will contain the hexadecimal data 4080.

ENABLE INTERRUPT — EI

Binary Format	Words	Clock Periods
0 0 1 1 0 1 0 0 | X | X ((PC))	1	3

Hexadecimal Format: 34

Operation: INTE ← 0

Description: The interrupt enable flip-flop is enabled. The processor is ready to accept external or internal interrupts.

Status Flags: No status flags are modified.

DISABLE INTERRUPT DI

Binary Format	Words	Clock Periods
0 0 1 1 0 1 0 1 | X | X ((PC))	1	3

Hexadecimal Format: 35

Operation: INTE ← 1

Description: The interrupt enable flip-flop is disabled. The processor does not accept any external or internal interrupts.

Status Flags: No status flags are modified.

LOAD THE INTERRUPT MASK REGISTER SM

Binary Format	Words	Clock Periods
0 0 1 1 0 1 1 0 | X | X ((PC))	2	3
8 BIT DATA | X | X ((PC)+1)		

Hexadecimal Format: 36

Operation: Mask register ← ((PC)+1)

Description: The Interrupt Mask Register is loaded with the 8 bit data provided by the higher byte of the second word of the instruction.

Status Flags: No status flags are modified.

Example: Assume that the memory locations 0400 and 0401 contain the hexadecimal data 36XX and 21XX (X = don't care) respectively. Then after executing the machine code at the memory location 0400, the Interrupt Mask Register will contain the hexadecimal data 21.

LOAD THE INTERNAL TIMER LC

Binary Format		Words	Clock Periods
0 0 1 1 0 1 1 1 | X | X	((PC))	2	3
16 BIT DATA	((PC)+1)		

Hexadecimal Format: 37

Operation: Counter Register ← ((PC)+1)

Description: The counter register is loaded with the 16 bit data provided by the second word of the instruction.

Status Flags: No status flags are modified.

Example: Assume that the memory locations 0400 and 0401 contain the hexadecimal data 37XX (X = don't care) and 3021 respectively. Then after executing the machine code at the memory location 0400, the Counter Register will be loaded with the hexadecimal data 3021.

SHIFT SCRATCHPAD REGISTER RIGHT ONE POSITION

Binary Format		Words	Clock Periods
0 0 1 1 COND | R_A | X	((PC))	1	3

Operation: $D_n \leftarrow (D_{n+1})$; $D_{15} \leftarrow$ 1 or 0

Description: The contents of the scratchpad register addressed by the R_A field of the operand are shifted right one position. A '0' or '1' is shifted to the most significant bit position depending upon the opcode used.

Status Flags: C_{n+4} and OVR are set to zero. N and Z are modified to reflect the results of the operation.

Hexadecimal Formats:

COND	INSTRUCTION	MNEMONIC	HEX. FORMAT
1000	Shift '1' to MsB Position	SR1	38
1100	Shift '0' to MsB Position	SR0	3C

Example: Assume that the scratchpad register 2 contains the hexadecimal data 0200. Then after executing the machine code 382X (X = don't care) the scratchpad register 2 will contain the hexadecimal data 8100. Similarly after executing the machine code 3C2X (X = don't care) the scratchpad register 2 will contain the hexadecimal data 0100.

SHIFT SCRATCHPAD REGISTER LEFT ONE POSITION SLI

Binary Format		Words	Clock Periods
0 0 1 1 1 0 0 1 | R_A | X	((PC))	1	3

Hexadecimal Format: 39

Operation: $D_{n+1} \leftarrow (D_n)$; $D_o \leftarrow 0$

Description: The contents of the scratchpad register addressed by the R_A field of the operand are shifted left one position. A zero is shifted to the least significant bit position.

Status Flags: C_{n+4} and OVR are set to zero. N and Z are modified to reflect the results of the operation.

Example: Assume that the scratchpad register 2 contains the hexadecimal data 0201. Then after executing the machine code 392X (X=don't care) the scratchpad register 2 will contain the hexadecimal data 0402.

RETURN FROM INTERRUPT RETI

Binary Format		Words	Clock Periods
0 0 1 1 1 0 1 0 | X | X	((PC))	1	3

Hexadecimal Format: 3A

Operation: SP $\leftarrow$ (SP)-1

PC $\leftarrow$ ((SP))

Description: The Stack Pointer is decremented and the contents of the stack are copied to the Program Counter Register.

Status Flags: No status flags are modified.

NO OPERATION NOP

Binary Format	Words	Clock Periods
0 0 1 1 1 0 1 1 | X | X ((PC))	1	3

Hexadecimal Format: 3B

Operation: None

Description: No operation is performed.

Status Flags: No status flags are modified.

ROTATE SCRATCHPAD REGISTER ONE POSITION

Binary Format	Words	Clock Periods
0 0 1 1 COND | R_A | X ((PC))	1	3

Operation: $D_n \leftarrow (D_{n+1})$; $D_{15} \leftarrow (D_o)$

or

$D_{n+1} \leftarrow (D_n)$; $D_o \leftarrow (D_{15})$

Description: The contents of the scratchpad register addresses by the R_A field of the operand are rotated one position to the left or to the right.

Status Flags: C_{n+4} and OVR are set to zero. N and Z are modified to reflect the results of the operation.

Hexadecimal Format:

COND	INSTRUCTION	MNEMONIC	HEX. FORMAT
1101	Rotate Right one Position	ROTR1	3D
1110	Rotate Left one Position	ROTL1	3E

SECTION 2

UMM-16 OPERATING SYSTEM

2.1 UMM-16 Operating System Commands

The UMM Monitor is stored in a 512 words by 16 bits Programmable Read-only Memory (PROM) starting from address zero of the memory. 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.

The UMM-16 Monitor mode is recognized by the character '$'. When the reset button is activated the monitor prints the character '$' on the user terminal to indicate that it is waiting for a command to by typed in.

The list of the UMM-16 Monitor commands is given below. Note that all addresses must be in hexadecimal format.

UMM-16 Monitor Commands:

M xxxx	The contents of memory address xxxx is printed on the user terminal. i. To examine the nest 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 " " 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	Insert breakpoint at address yyyy and start executing from address xxxx.
P	Print the contents of scratchpad registers at the breakpoint
$	Return to command mode
C	This command is intended for future expansion of the Operating System. If the command 'C' is typed, a jump to address 0200 is performed.

If the user types in a character which is not valid, the monitor responds with a '?'.

2.2 Paper tape Punch Format

When the 'D xxxx,yyyy' command is typed, the monitor punches a paper tape with the following format:

	H'0024 Null Leader
* * * * * * * * * * * * * * * * * * *	H'0012' Data protection characters
S yyyy	Data start address yyyy
	8, 16 Bit Data Blocks
XY Y Y Y Y Y Y YC	
XY Y Y Y Y Y Y YC	
XY Y Y Y Y Y Y YC	
* * * * * * * * * * * * * * * * *	H'0012' Data protection characters
	H'0024' Null characters

where,

y 4 Bit data

Y 16 Bit data

S Start address identifier

X Data Block start character

C Check-sum of the data block (sum of all Y's in a block)

2.3 Breakpoint register display format

When the 'P' command is typed, the monitor prints the contents of the scratchpad registers in the following format:

R_0 R_1 R_2 R_3 R_4 R_5 R_6 R_7 R_8 R_9

R_{10} R_{11} R_{12} R_{13} R_{14} R_{15} B_1^* B_2^* B_3^*

When the 'B xxxx, yyyy' command is typed, the monitor inserts a breakpoint at address yyyy and starts executing the program starting at address xxxx. During execution one of the following will happen:

i. If the breakpoint is encountered, then the data at the breakpoint address is not altered. The monitor will respond with: $

ii. If the breakpoint is not encountered, then the data at the breakpoint address will be destroyed. The monitor will not print any characters Activating the reset button will return the processor to the command mode.

* B_1 = Breakpoint address , B_2 = Data at the breakpoint,
B_3 = Data at the breakpoint address+1

2.4 Teletype Input/Output Routines

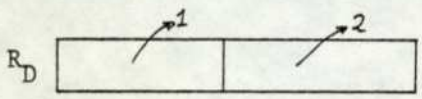
The UMM-16 Monitor contains many subroutines for the teletype/VDU input and output Routines. No 'start-up' sequence is required.

Scratchpad register D is used during the input and output routines. TTYOUT subroutines take the contents of scratchpad register D, interpret it as an ASCII code and print it on the teletype. TTYIN subroutines put the ASCII code for the character typed in scratchpad register D.HEX subroutines take the contents of scratchpad register D, interpret it as a hexadecimal data and print it on the teletype.

The list of the UMM-16 teletype/VDU input and output subroutines is given below.

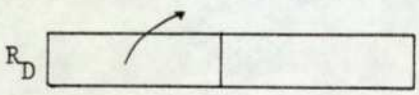
UMM-16 teletype input/output subroutines: (R_D = scratchpad register D)

Subroutine	Start address(Hex.)	Description
1. TTYOUT1	0039	Print the contents of R_D as 2 ASCII characters



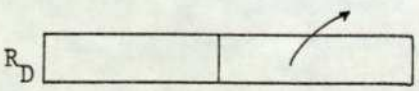
Example: Assume that R_D contains the hexadecimal data 3135. Then, after calling this subroutine. 15 will be printed on the teletype (ASCII code for 1 is 31_H, ASCII code for 5 is 35_H)

2. TTYOUT2	002E	Print the upper byte of R_D as one ASCII character



Example: Assume that R_D contains the hexadecomal data 3135. Then after calling this subroutine, 1 will be printed on the teletype.

3. TTYOUT3	003B	Print the lower byte of R_D as one ASCII character

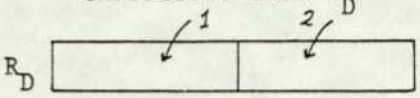


Example: Assume that R_D contains the hexadecimal data 3135. Then after calling this subroutine, 5 will be printed on the teletype.

4. SPACE	0046	Print one space
5. CRT	004D	Print line feed, carriage return

6. TTYIN1 0063

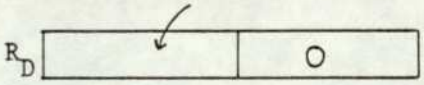
Receive two ASCII characters into R_D



Example: Assume that this subroutine is called and the two ASCII characters 'L' and 'S' are typed in. Then, register R_D will be loaded with the hexadecimal data 4C53 (ASCII code L is $4C_H$, ASCII code S is 53_H)

7. TTYIN2 0056

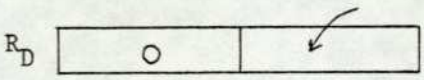
Receive one ASCII character into higher byte of R_D and clear the lower byte



Example: Assume that this subroutine is called and the ASCII character 'L' is typed in. Then, register R_D will be loaded with the hexadecimal data 4C00 (lower byte cleared)

8. TTYIN3 0070

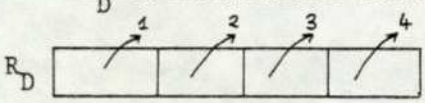
Receive one ASCII character into lower byte of R_D and clear the upper byte



Example: Assume that this subroutine is called and the ASCII character 'L' is typed in. Then, register R_D will be loaded with the hexadecimal data 004C (higher byte cleared)

9. HEXI 00BE

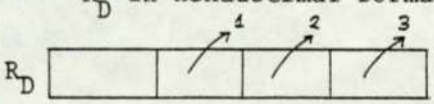
Print the contents of R_D in hexadecimal format



Example: Assume that register R_D contains the hexadecimal data 3512. Then, after calling this subroutine, 3512 will be printed on the teletype.

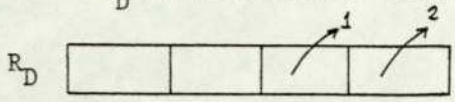
10. HEX2 00C3

Print 3 characters in R_D in hexadecimal format.



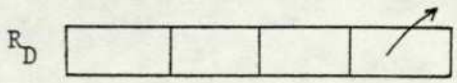
Example: Assume that register R_D contains the hexadecimal data 3512. Then, after calling this subroutine, 512 will be printed on the teletype.

11. HEX3 00C6 Print lower byte of R_D in hexadecimal format



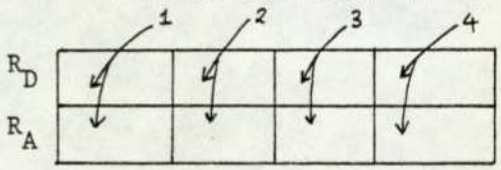
Example: Assume that register R_D contains the hexadecimal data 3512. Then, after calling this subroutine, 12 will be printed on the teletype.

12. HEX4 00CD Print lowest four bits R_D in hexadecimal format



Example: Assume that register R_D contains the hexadecimal data 3512. Then, after calling this subroutine, 2 will be printed on the teletype.

13. CONVERT 0077 Receive ASCII characters from the teletype up to a carriage return, line feed, comma or space and convert them into hexadecimal. Store the result in R_D and R_A (scratchpad register A). The leading zeros can be omitted. The characters received will be echoed on the user terminal. (Up to 4 characters maximum).



Example: Assume that this subroutine is called then,

ASCII characters typed in	Data loaded into R_D and R_A
1354 *	1354
320 *	0320
46 *	0046
8 *	0008

where "*" denotes carriage return or line-feed or space or a comma.

14. To return to the operating system: Perform a Jump to address H'0000'

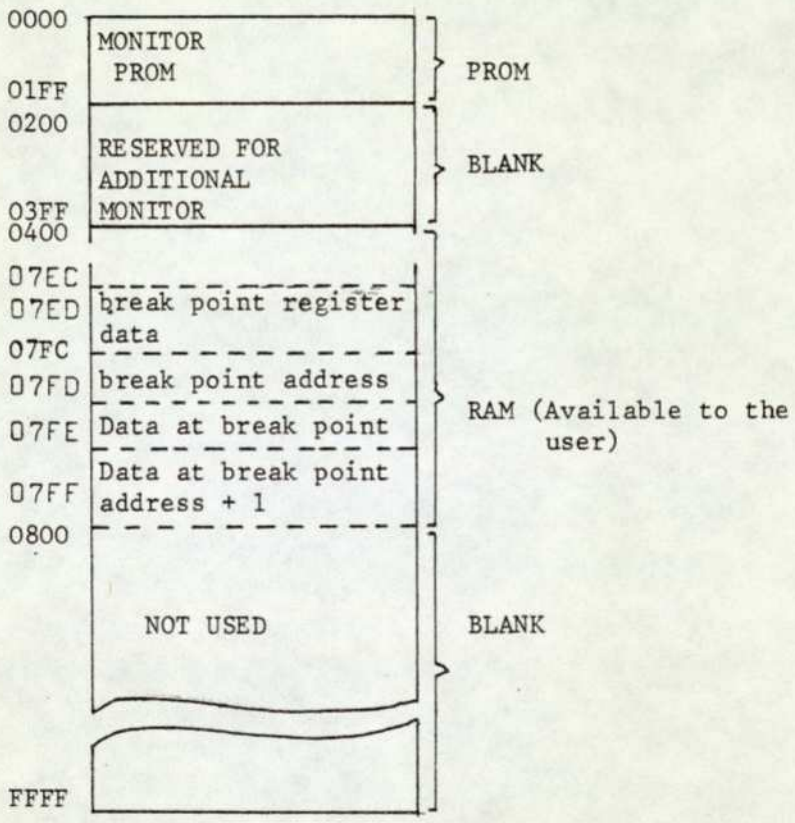
2.5 Teletype/VDU transmission rate

A switch on the front panel enables the transmission rates to be selected between 1200, 600, 300 or 110 baud. Each character contains 10 bits (1 start bit, 8 data bits and 1 stop bit). No parity bit is used. Thus, 1200 baud rate transmitts data at 1200 bits/sec or 120 characters/ sec.

2.6 Teletype/VDU transmission mode

The processor sends and receives serial data in asynchronous mode. Either 20 mA loop or V24 (RS232C) interface signals can be connected to the processor.

UMM-16 Memory Map



2.7 Pin connections of the TTY/VDU sockets

The TTY/VDU sockets are standard 3 pin DIN sockets with the following pin connections:

	20mA loop socket	V24 socket
Pin 1:	Input from KEYBOARD	Output to PRINTER
Pin 2:	Output to PRINTER	Ground (zero volts)
Pin 3:	+ 12 Volts	Input from KEYBOARD

SECTION 3

INPUT-OUTPUT AND INTERRUPTS

3.1 The Input-Output Interface

All the input-output port signals of VMM-16 are 16 bits wide, organised to accept 16 bits of parallel data on separate input and output lines (latched). The input-output port signals are at standard TTL level providing the following logic voltage levels:

i Input Port

Min. High Level Voltage, V_{IH} = 2 Volts
Max. Low Level Voltage, V_{IL} = 0.8 Volts

Max. High Level Input Current, I_{IH} = 40 µA
Max. Low Level Input Current, I_{IL} = -1.6 mA

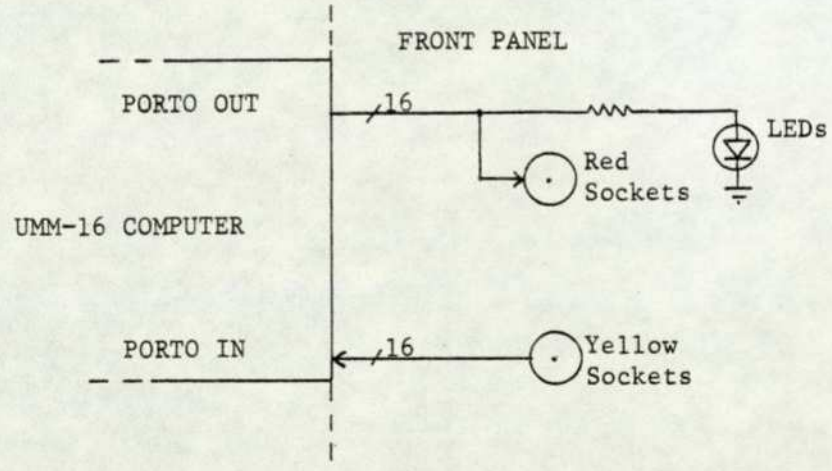
ii Output Port

Min. High Level Output Voltage, V_{OH} = 2.4 Volts
Max. Low Level Output Voltage, V_{OL} = 0.4 Volts

Max. High Level Output Current, I_{OH} = -400 µA
Max. Low Level Output Current, I_{OL} = 16 mA

Although a large number (Max. 256) of I/O ports can be connected to UMM-16, at present only 4 ports are in use. Port 0 input and output lines are connected to miniature sockets mounted on the front panel. The 'red' sockets are the output port lines while the 'yellow' sockets are the input lines. The Port 0 output lines are also connected to 16 LEDs mounted on the front panel of UMM-16.

The connection of Port 0 lines are shown below:



Example

An example will be given to illustrate the use of input-output instructions. Assume that it is required to write a program to increment the contents of scratchpad register 2 continuously and display the result on the 16 LEDs of Port 0.

Scratchpad register 2 should be cleared initially.

The program steps should be as follows:

Clear scratchpad register 2

Loop: output the contents of scratchpad register 2 via Port 0

Increment the contents of scratchpad register 2 by one

Go to loop

The assembly listing of the program is given below: (Assume that the don't care entries in the machine code are taken as zero and that program starts from memory address H '0400'.

Memory Location	Machine Code	Mnemonic	Comments
0400	1220	CLRR 2	Clear scratchpad register 2
0401	1E20	OUTS 2, H '0000'	Output the contents of register 2 via Port 0
0403	0D20	INCR 2	Increment register 2 by one
0404	2600	JMP H '0401'	Jump to address H '0401'
0405	0401		

The program (machine code) can be entered into the memory by using the monitor command M 400.

When this program is executed (monitor command E 400) the 16 LEDs on the front panel will start counting up as a 16 bit binary counter.

Port 1

Port 1 of the UMM-16 computer is used as a data acquisition channel. The input lines of Port 1 are connected to a 12 bit Analog to Digital Converter and a Sample/Hold amplifier. The user has the choice of whether or not to use the Sample/hold amplfier depending upon the external connections he makes on the front panel. The output lines of Port 1 are connected to a 12 bit Digital to Analog Coverter.

The A/D 'convert' pulses can be derived from an external pulse generator and fed to the 'Ts' input on the front panel of UMM-16. The 'convert' pulses are passed through an amplitude limiter (limit to 5 volts) and the output from the limiter is sent to the 'convert' input of the A/D converter. The same pulses are also sent out to the front panel 'TEST' socket. The purpose of the 'TEST' output is to observe the 'convert' commands entering the A/D converter. The 'convert' pulses must be positive going with an amplitude between 5 to 10 volts. When the A/D completes a conversion, the 'STRB' output on the front panel makes a high to low transition (a negative going pulse). The A/D and the D/A converter can either operate in the 'Bipolar' mode (± voltage levels) or in the 'Unipolar' mode (only positive voltage levels) selected by a SPDT switch (mode select switch) mounted on the front panel. The input and output full scale voltage ranges of A/D and D/A can be selected from the two switches mounted on the front panel of UMM-16 (range select switches). The Bipolar (BPLR) operation provides two's complement data to the processor while the Unipolar (UPLR) operation provides straight binary data to the processor. The mode select and the range select switches should be set as follows:

Mode Select Switch	Range Select Switch	Operation Code
BPLR	±10 or ±5 or ±2.5 Volts	Two's complement - Bipolar
UPLR	0 to +5 Volts	Straight binary - Unipolar

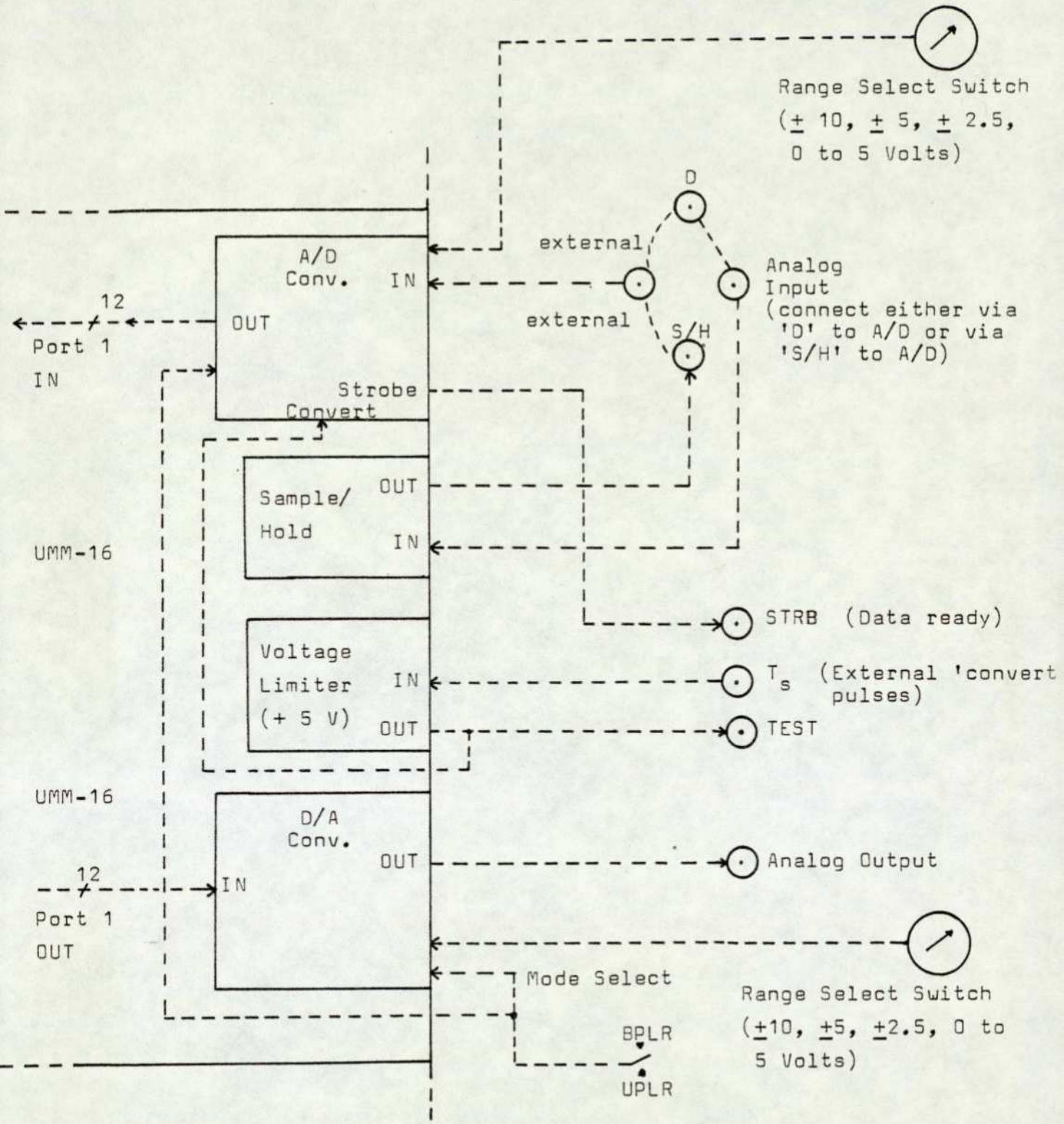
The two's complement and the straight binary codes are defined as in the following table:

	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 ±10 volt Bipolar and 0 to 10 volts Unipolar operation are defined in the following table (for ±5 volt or 0 to 5 volts range, divide the values shown by 2. For ±2.5 volt range, divide the values shown by 4):

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

Note: Full Scale = Full Scale -1 LSB on positive full scale

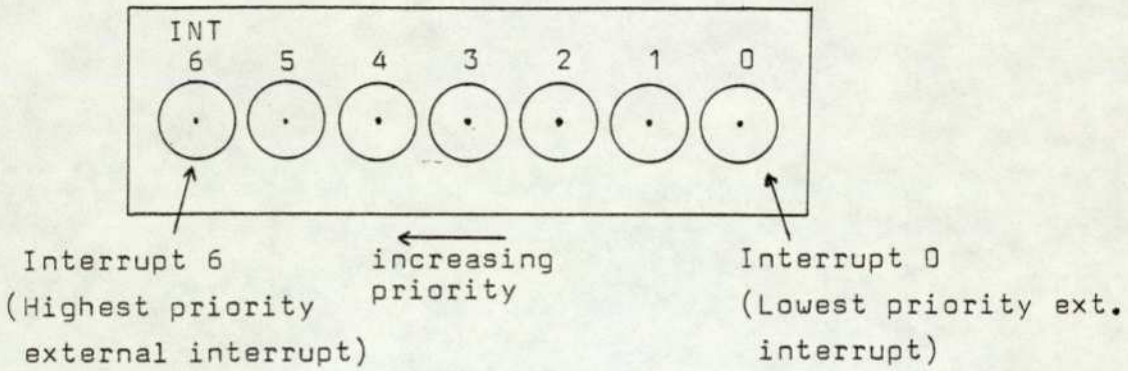


Block diagram of the Data Acquisition Channel

3.2 INTERRUPT PROCESSING

The UMM-16 computer accepts seven external interrupts and one internal (timer) interrupt . The interrupt inputs are organised in a priority level and any interrupt can be maskedunder software control. The 7 external interrupt inputs are mounted on the front panel of UMM-16 with an ascending priority level:

INTERRUPT SECTION OF THE FRONT PANEL

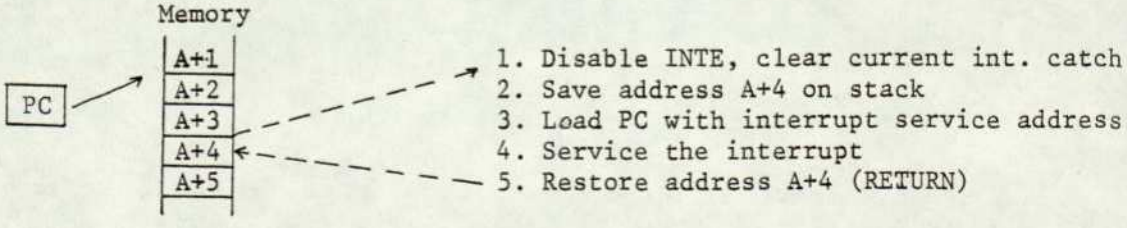


Interrupt 7 (internal timer) has the highest priority of all. For an interrupt to be processed by the computer, the interrupt enable flip-flop must be enabled by executing the instruction 'Enable Interrupt' . An interrupt will be recognised on the high to low transition of the interrupt input which is not masked. All eight interrupt inputs have pulse catching latches at their inputs so that the interrupt requests cab be latched to the processor and wait for their service. Interrupts will only be serviced during the instruction fetch cycle. Each interrupt input is assigned a unique location in the RAM part of the memory where the service address for that interrupt input is found.

The interrupt programming registers of the UMM-16 computer is shown below

8 BIT MASK REG.	
16 BIT COUNTER REGISTER	
INTE	In interrupt enable flip-flop

The tasks of the UMM-16 computer during an interrupt are shown below



At address A+4 a non masked interrupt has been detected. Then the first task of the processor is to disable the interrupt enable flip-flop so that no more interrupts will be accepted until the current one is serviced. Also at this stage, the interrupt latch of the current interrupt is cleared (set to 1). Then the current value of the program counter register (A+4) is pushed onto stack and the program counter is loaded with the interrupt service address. After servicing the interrupt, address A+4 is popped off the stack by executing the 'RETI' instruction. Thus program returns to the suspended point. Note that the contents of any register is not saved by the processor. The user can include this function in the interrupt service routine if it is necessary.

After returning from the interrupt service routine, no more interrupts can be accepted by the processor until the Interrupt enable flip-flop is re-enabled (The instruction 'Enable Interrupt' executed) by the user.

If multiple interrupts occur at the same time then the processor will respond to the interrupt input which is not masked and which has the highest priority level.

3.3 The Mask register

The Mask register is eight bits wide and it can be loaded under software control. The purpose of the mask register is to mask any interrupt input so that interrupts from that input become nonactive. A Logic 1 in any bit position of the Mask register masks the corresponding interrupt input:

MSB							LSB	
7	6	5	4	3	2	1	0	Mask register (8bits)
7	6	5	4	3	2	1	0	Interrupt Inputs (8 inputs)

e.g. If the Mask register contains the binary data 00110100 then interrupts from the interrupt inputs 2, 4 and 5 will be masked. Interrupts from other interrupt inputs will be enabled. If it is required to accept interrupts only from interrupt input 7 (timer) then the mask register must be loaded with the binary data 01111111.

The Mask register is loaded with the instruction "SM" (see UMM16 instruction set, opcode 36)

Example

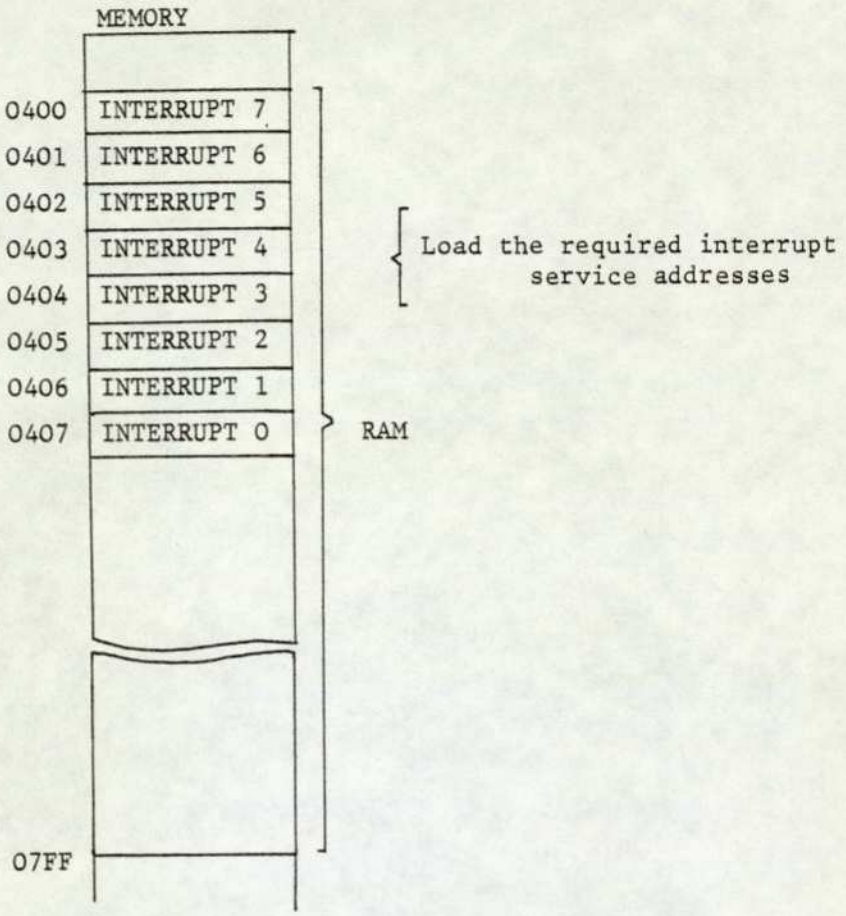
Suppose we wish to load the Mask register with the binary data 01111111. Then the instruction SM H'7F00' should be executed (machine code 3600 followed by 7F00).

It is important that the interrupt enable flip-flop should be enabled after the Mask register is loaded with the required bit pattern. This ensures that interrupts from unwanted channels do not get service (specially the timer interrupt).

When the 'reset' button is activated, the interrupt enable flip-flop is disabled automatically.

3.4 Interrupt Service Addresses

Each interrupt input has an unique location in the memory which can be loaded with the interrupt service address for that particular interrupt input as shown below:



e.g. If memory location 0400 is loaded with the 16 bit data 0700, then when an interrupt is accepted from Interrupt 07, the interrupt service address will be 0700.

3.5 The Timer Interrupt (Interrupt 7)

The Timer Interrupt logic consists of a software programmable 16 bit counter register and a 16 bit binary down counter. The counter is loaded from the counter register and it decrements on each clock pulse when Mask 7 is enabled. If Mask 7 is disabled (High) then the counter is in the "Hold" mode. When the contents of the counter are zero it sends an interrupt pulse (high to low transition) to the interrupt input 7 latches (internal). If the interrupt enable flip-flop is enabled and the system is in the instruction fetch cycle then this interrupt will be accepted and serviced by the processor. The interrupt service address of the timer interrupt is at location 0400 of the memory. When the contents of the counter are zero it is reloaded on the next clock pulse from the counter register. The maximum number that can be loaded into the counter register is 65536($FFFF_H$)

If the counter is loaded with a constant 'K' and Mask 7 is enabled (low) then the timer interrupts will be generated after every 'KT' seconds, where 'T' is the clock period of the system (1μs). There is no need to reload the counter register after a timer interrupt is generated. This is done automatically by the interrupt hardware.

When an interrupt is accepted by the processor, the processor executes some internal control functions to disable further interrupts, save the current Program Counter Register contents and to load the Program Counter Register with the interrupt service address. These functions take 4 clock cycles. Therefore the first instruction of the interrupt service address will be executed after 4T seconds. Assume that a constant 'K' is loaded into the counter register and Mask 7 is enabled. Then after 'KT' seconds the first timer interrupt will be sent to the processor. If the processor is in the instruction fetch cycle , this interrupt will be accepted and the first instruction of the timer interrupt service address (instruction at location 'xxxx' where 'xxxx' is the address loaded to location 0400 of the memory) will be executed after 4T seconds. Therefore the first instruction of the interrupt service address will be executed after (K+4)T seconds although the timer interrupts will be generated at precisely 'KT' seconds intervals.

Loading the Counter Register

The counter register is loaded with the instruction "load the counter register", LC, which has the opcode 37_H (see the instruction set of UMM-16).

3.6 Example on using the timer interrupt

Assume that we wish to increment the contents of scratchpad register 2 by one every sixty periods and that the contents of register 2 will be sent out via Port 0. Also assume that Port 0 should initially be cleaned.

Assume that the initialization part of the program is stored starting at memory location 0500 and that the interrupt service address starts at location 0600. The program steps for this example should be as follows:

At Memory Location 0400	Load the interrupt service address 0600
At Memory Location 0500	Clear scratchpad register 2
	Load the counter register with sixty (Hex. 3C)
	Enable Mask 7, disable other Masks
	Enable Interrupts
	Wait for timer interrupts

At Memory Location 0600 Output contents of register 2 via Port 0
Increment the contents of register 2 by one
Return from interrupt

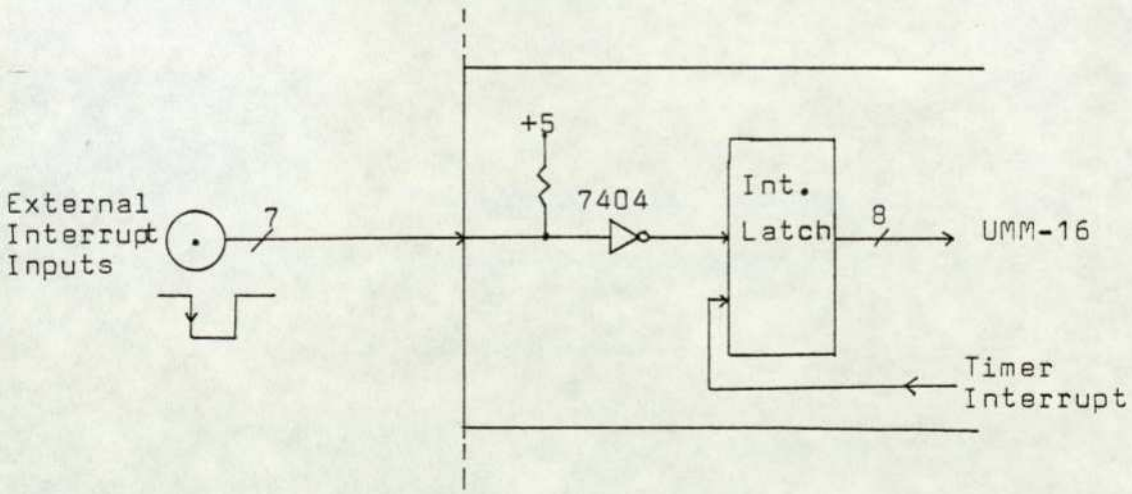
The actual program steps are given below:

Load memory location 0400 with 0600, using the monitor commant M 400 . Then,

Memory Location	Machine Code		Mnemonic	Comments
0500	1220		CLRR 2,0	; Clea register 2
0501	3700		LC,H'003C'	; Load the counter reg.
0502	003C			with 3C.
0503	3600		SM,H'7F00'	; Enable Mask 7,
0504	7F00			disable others.
0505	3400	Loop	EI	; Enable interrupts
0506	2600		JMP Loop	; Wait for interrupt
0507	0505			
0600	IE20		OUTS 2,0	; Output contents of
0601	0000			reg. 2 via Port 0.
0602	0D20		INCR 2	; Increment register 2
0603	3A00		RETI	; Return from interrupt.

3.7 Interrupt Input Logic Levels

The interrupt inputs are at standard TTL logic levels as shown below:



3.8 The Instruction 'WAIT'

The WAIT instruction (opcode 22_H) disables the system clock from the processor. When this instruction is executed, the processor enters into the WAIT state and sends out a 'wait acknowledge' signal. Once in the WAIT state, the processor can only be restarted by activating the external 'ready' input.

The 'wait acknowledge' output is normally HIGH and it will make a transition to the low state to indicate that the processor is in the WAIT state.

The 'ready' input is only recognised if the processor is in WAIT state. A negative going pulse (high-to-low transition) on the 'ready' input restarts the processor and the next sequential instruction after the WAIT instruction will be executed by the processor .

The four possible uses of the WAIT instruction are:

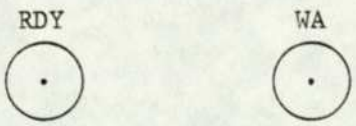
i. To introduce time delays during the execution of a program. Here, an external counter can be enabled by the 'wait acknowledge' signal and when the required delay is obtained a pulse from the counter can activate the 'ready' input to restart the processor.

ii. To insert a delay to accomodate the response time of slow memory or I/O circuits

iii. To insert trap points in the program so that the functions of a required set of instructions can be examined.

iv. To stop processing until an external service request arrives.

The 'wait acknowledge' output (WA) and the 'ready' input (RDY) are available on the front panel of the UMM-16 via two sockets:



WAIT instruction during an interrupt

Interrupts can not be processed during a WAIT instruction. However, it is possible to enter the WAIT state during servicing an interrupt.

3.9 The 'reset' Button

When the reset button is pressed the following happen:

i. The Program Counter is cleared and execution starts from address zero of the memory, where the monitor command $ will be printed on the user's terminal.

ii. The Timer interrupt is disabled (timer counter in the 'hold' mode).

iii. The interrupt enable flip flop is disabled so that no interrupts will be serviced unless the flip flop enabled by software (The instruction 'EI').

SECTION 4

OPERATOR'S CONSOLE AND THE OPERATING INSTRUCTIONS

4.1 The Operator's Console

The UMM-16 computer system is built into two sub-rack units (see Figure 4.1). Sub-rack 1 includes the main system hardware. Sub-rack 2 includes the support processor (for loading user defined microinstructions into the control memory of UMM-16), the data acquisition channel (A/D and D/A converters and Sample/Hold amplifiers) and the system power supply.

Sub-rack 1 provides the user with the following indicators and switches: (The numbers below refer to the numbers in Figure 4.2)

1.	PORT 0 OUT	The 16 LEDs indicate the data at the output of PORT 0. The MSB is at the far left.
2.	PORT 0 OUT	The 16 'red' sockets are connected to the output lines of PORT 0. The MSB is at the far left.
3.	PORT 0 IN	The 16 'yellow' sockets are connected to the input lines of PORT 0. The MSB is at the far left.
4.	INT	The external interrupt input lines. The socket on the far right is the interrupt input 0 and the socket on the far left if the interrupt input 6.
5.	CLK	An output socket connected to the system clock.
6.	GND	This socket is connected to the power supply ground terminal
7.	+5	This socket is connected to the +5 volt terminal of the power supply.
8.	+12	This socket is connected to the +12 volt terminal of the power supply.
9.	-12	This socket is connected to the -12 volt terminal of the power supply.
10.		The 3 LEDs indicate the status of each power supply
11.	RESET	The 'reset' button of the microprogrammable computer. Pressing the 'reset' button caused the processor to jump to the monitor mode.
12.	RUN/HLT	This switch must be in the 'RUN' position for the system to operate from its internal clock. In the 'HLT' mode, the processor will HALT at the instruction fetch cycle.
13.	STEP	Provides single step clocks to the processor each time it is pressed. The system must be in the 'HLT' position for this switch to be active.
14.	MAP LD/OP	This switch must be in the 'OP (=operate)' position. When in the 'LD (=load)' position, the Mapping RAM is enabled so that it can be loaded from the Support Processor (When the Support Processor executes the command 'LMAP') .

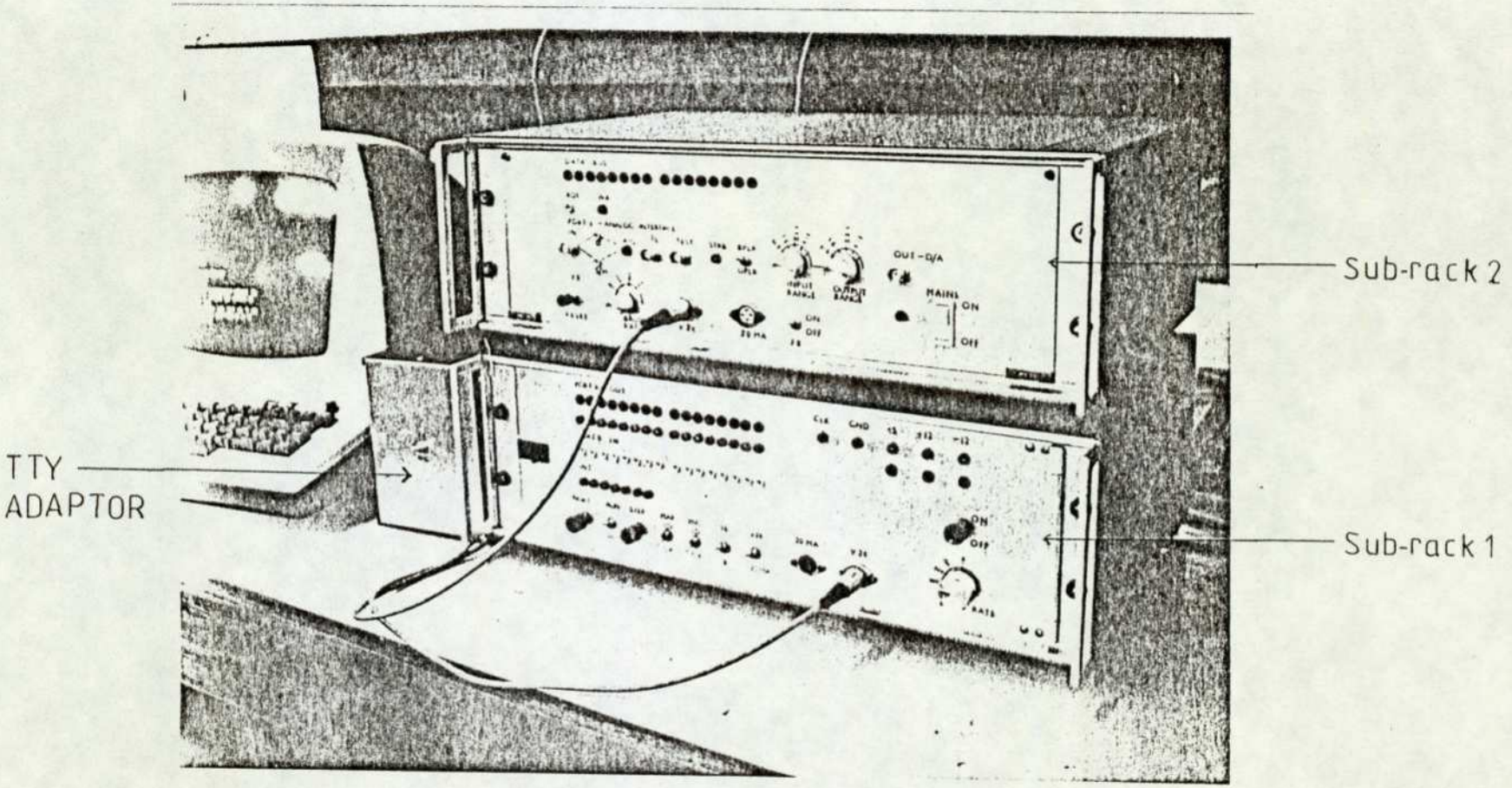


Figure 4.1 The UMM-16 Microcomputer

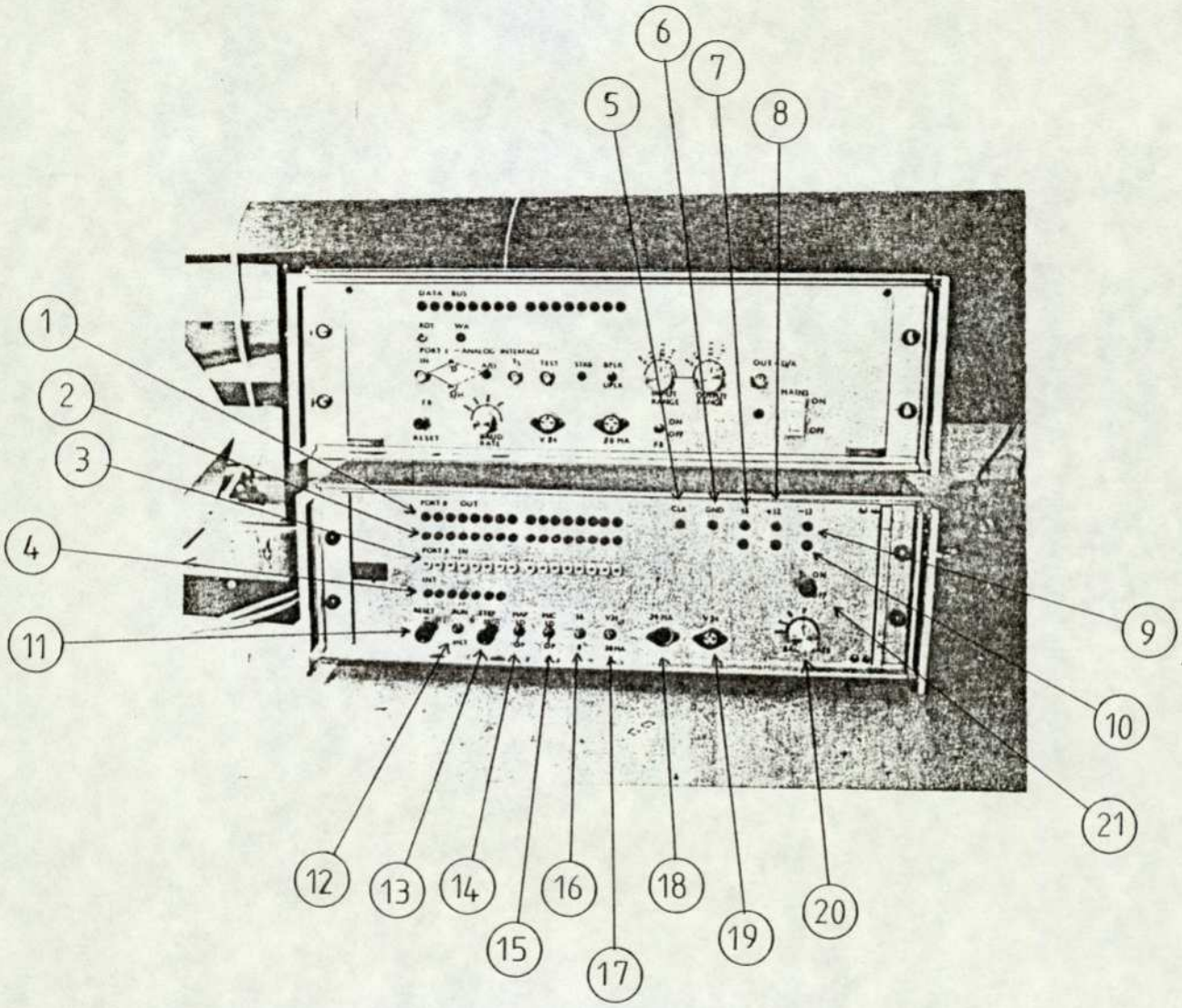


Figure 4.2 Sub-rack 1

15.	MIC LD/OP	This switch must be in the 'OP (= operate)' position. When in the 'LD (= load)' position, the microprogram RAM is enabled so that it can be loaded from the Support Processor (when the Support Processor executes the command 'LMIC') .
16.	16/8	This switch selects the data length of the Bit-Slice system. It must be in the '16' position for operation with 16 bits of data length.
17.	V24/20 mA	This switch selects either the V24 or the 20 mA loop interface for the Bit-Slice system. Set to V24 for VDUs and to 20 mA for teletypes.
18.	20 mA	The 20 mA loop TTY socket for the Bit-Slice system.
19.	V24	The V24 VDU socket for the Bit-Slice system.
20.	BAUD RATE	This switch selects the Baud Rate for TTY/VDU interface. The Baud Rates available are 1200,600,300 and 110.
21.	ON/OFF	The ON/OFF switch of the Bit-Slice system (Push button latching switch).

The sub-rack 2 provides the user with the following switches and indicators (the numbers below refer to the numbers in Fig. 4.3):

22.	DATA BUS	The 16 LEDs indicate the data on the Data Bus of the Bit-Slice system. The MSB is at the far left.
23.	RDY	The 'READY' input to the Bit-Slice system (see section 3 for details).
24.	WA	The 'WAIT ACKNOWLEDGE' output from the Bit-Slice system (see section 3 for details).
25.		Analog to digital converter input connections (see section 3 for details).
26.	T_s	The 'Convert pulses' to the Analog to Digital converter are applied to this input (see section 3).
27.	TEST	The test point for the Analog to Digital converter 'convert pulses' (see section 3
28.	STRB	The 'STROBE' output from the Analog to Digital converter. When the A/D conversion is complete, the STRB goes LOW.
29.	BPLR/UPLR	This switch selects the mode of operation of the A/D and the D/A.

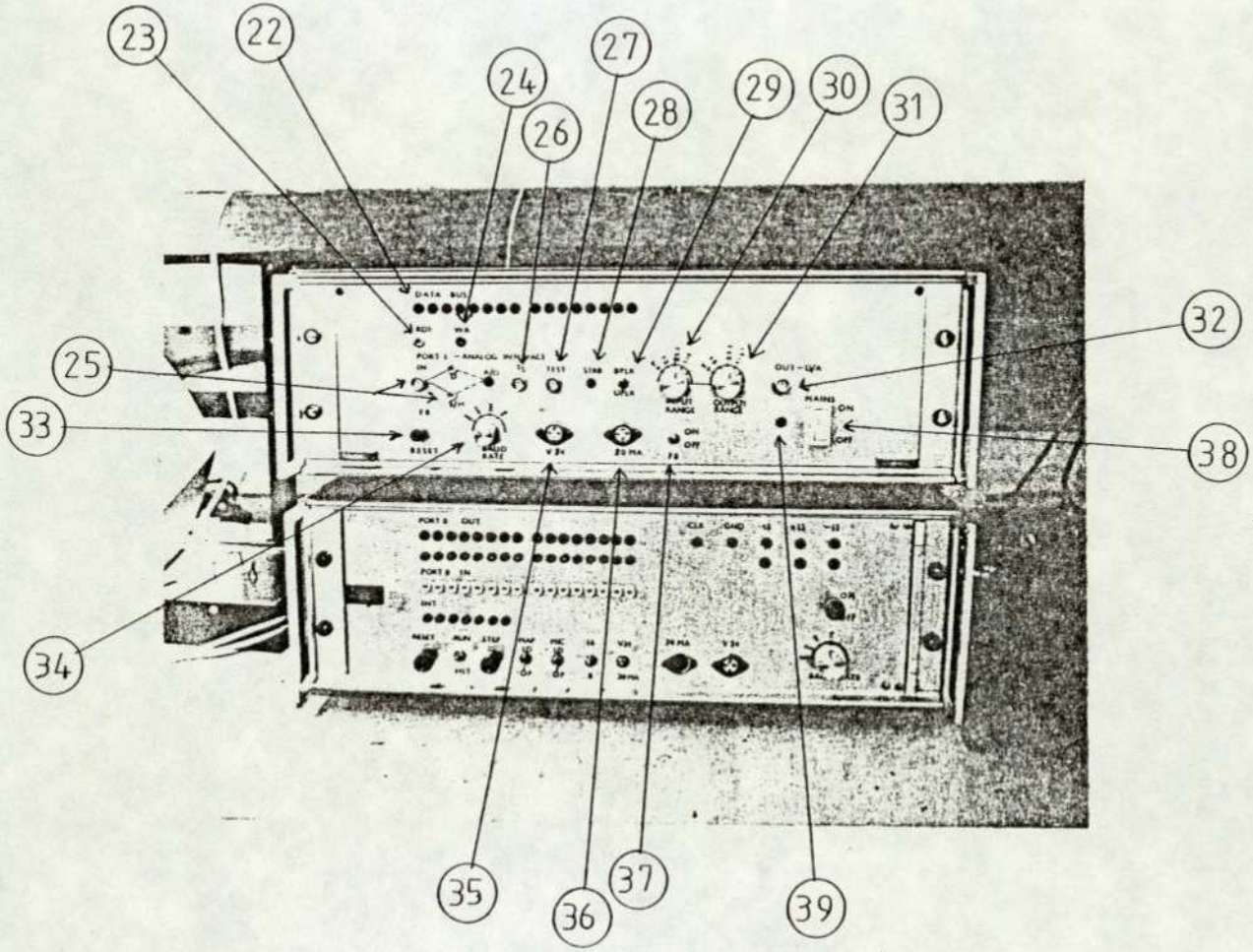


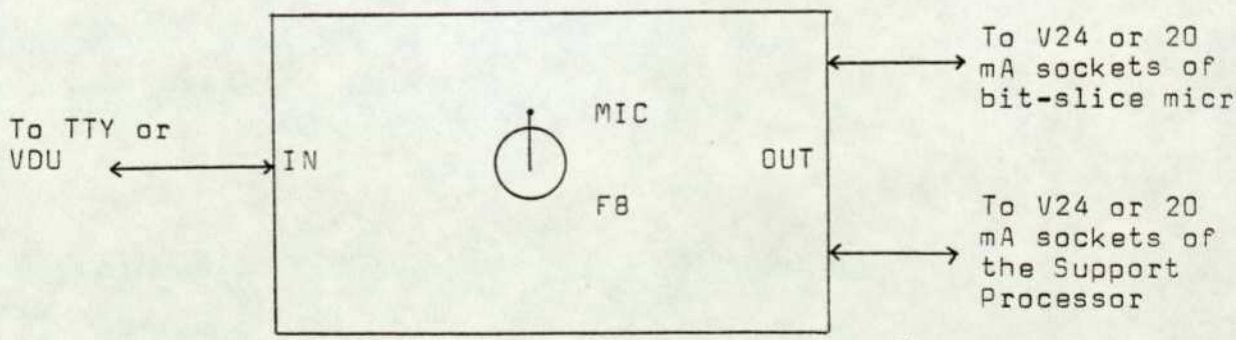
Figure 4.3 Sub-rack 2

30.	INPUT RANGE	This switch selects the A/D input range.
31.	OUTPUT RANGE	This switch selects the D/A output range.
32.	OUT D/A	Digital to Analog converter output socket.
33.	F8 RESET	The RESET button of the Support Processor.
34.	BAUD RATE	This switch selects the Baud Rate for the Support Processor.
35.	V24	The V24 VDU socket* for the Support Processor
36.	20 mA	The 20 mA loop TTY socket* for the Support Processor.
37.	F8 ON/OFF	The ON/OFF switch of the Support Processor.
38.	MAINS ON/OFF	Mains ON/OFF switch of the system
39.		Neon indicator for the mains supply.

* The V24/20 mA interface for the Support Processor is selected from the switch at the rear of sub-rack 2.

4.2 The TTY ADAPTOR

In addition to the two sub-rack units, a small unit (TTY ADAPTOR, see Fig. 4.1) is provided to enable the user to select the TTY/VDU interface signals either to the Bit-Slice system or to the Support Processor. The 3 external leads of the TTY ADAPTOR should be connected as follows:-



When the switch is in the F8 position the TTY/VDU can only communicate with the Support Processor. When the switch is in the MIC position the TTY/VDU communicates only with the Bit-Slice microcomputer.

4.3 OPERATING INSTRUCTIONS

Before the UMM-16 computer is connected to a mains supply the following steps should be followed:

1. Make sure that the front panels of sub-rack 1 and sub-rack 2 are firmly shut.
2. Place sub-rack 2 on top of sub-rack 1 (see Figure 4.1).
3. Connect the 25 way plug at the back of sub-rack 1 to the 25 way socket at the back of sub-rack 2.
4. Connect the 37 way plug at the back of sub-rack 1 to the 37 way socket at the back of sub-rack 2.
5. Connect the octal plug at the back of sub-rack 1 to the octal socket at the back of sub-rack 2.
6. Connect the TTY ADAPTOR as shown in Figure 4.4 and switch to the 'MIC' position.
7. Select the required baud-rate on sub-rack 1.
8. Connect the fan to a mains socket via the mains lead (black) provided at the rear of sub-rack 2.
9. Connect the mains lead to a mains supply and switch on the mains switch on the front panel of sub-rack 2. The neon bulb near the mains switch will indicate that mains power is applied to the system.
10. Press the ON/OFF switch on sub-rack 1 to connect the logic power supplies to the main computer. The 3 LEDS on the front right hand side of sub-rack 1 will indicate that power is applied to the main computer.
11. Press the 'reset' button on sub-rack 1 to operate the main computer. A '$' sign will be printed (by the UMM-16 monitor) on the user terminal, indicating that the system is ready to accept the several alternative user commands.

The procedure given above assumes that the support processor will not be used. If it is required to use the support processor, then turn the F8 ON/OFF switch to the 'ON' position, select the required baud-rate and the mode of communication (via the switch at the rear of sub-rack 2). Set the TTY ADAPTOR switch to the 'F8' position. Then, pressing the F8 RESET button will operate the support processor.

APPENDIX A

UMM-16 Instruction Set Summary

opCode (HEX)	WORDS	CYCLES	Mnemonic	Meaning	Action	Flags C O N Z
			1. Register Transfers			
17	1	4	MOVR R_A,R_B	Move scratchpad registers	$R_A \leftarrow (R_B)$	No change
18	1	3	MOVQ R_A	Move Q to scratchpad register	$R_A \leftarrow (Q)$	No change
19	1	3	LRQ R_A	Move scratchpad register to Q	$Q \leftarrow (R_A)$	No change
1A	1	3	PUSH $R_A{}^1$,R_B	Move scratchpad register to aux.	$R_A{}^1 \leftarrow (R_B)$	No change
1B	1	3	POP $R_A{}^1$,R_B	Move aux. to scratchpad register	$R_B \leftarrow (R_A{}^1)$	0 0 X X
1C	1	3	MOVS R_A	Move S to scratchpad register	$R_A \leftarrow (S)$	No change
1D	1	3	LRS R_A	Move scratchpad register to S	$S \leftarrow (R_A)$	X X X X
31	1	3	LRP R_A	Move scratchpad register to PC	$PC \leftarrow (R_A)$	No change
			2. Register Logical			
01	1	3	ANDR R_A,R_B	AND Scratchpad registers	$R_A \leftarrow (R_A).AND.(R_B)$	0 0 X X
02	1	3	ORR R_A,R_B	OR scratchpad registers	$R_A \leftarrow (R_A).OR.(R_B)$	0 0 X X
03	1	3	EXR R_A,R_B	EX-OR scratchpad registers	$R_A \leftarrow (R_A).EX\text{-}OR.(R_B)$	0 0 X X
04	1	3	COMR R_A	COMPLEMENT scratchpad regst.	$R_A \leftarrow (\overline{R}_A)$	X X X X
05	1	3	COMQ	COMPLEMENT Q register	$Q \leftarrow (\overline{Q})$	X X X X
06	1	3	TCOM R_A	2'S COMPLEMENT scratchpad register	$R_A \leftarrow (\overline{R}_A) + 1$	X X X X
			3. Register Arithmetic			
0A	1	3	ADDR R_A,R_B	ADD scratchpad registers	$R_A \leftarrow (R_A)+(R_B)$	X X X X
0B	1	3	ADDQ R_A,R_B	ADD Q to scratchpad register	$R_B \leftarrow (R_A)+(Q)$	X X X X
0C	1	3	SUBR R_A,R_B	SUBTRACT scratchpad registers	$R_A \leftarrow (R_A)-(R_B)$	X X X X
0D	1	3	INCR R_A	INCREMENT scratchpad register	$R_A \leftarrow (R_A)+1$	X X X X
0E	1	3	INCQ	INCREMENT Q register	$Q \leftarrow (Q)+1$	X X X X
0F	1	3	DECR R_A	DECREMENT scratchpad register	$R_A \leftarrow (R_A)-1$	X X X X
10	1	3	DECQ	DECREMENT Q register	$Q \leftarrow (Q)-1$	X X X X
11	1	3	CLRQ	CLEAR Q register	$Q \leftarrow 0$	0 0 X X
12	1	3	CLRR	CLEAR scratchpad register	$R_A \leftarrow 0$	0 0 X X

opCode (HEX)	WORDS	CYCLES	Mnemonic	Meaning	Action	Flags C O N Z
			4. Register Immediate			
07	2	3	ANDI R_A,n	AND scratchpad register Immediate	$R_A \leftarrow (R_A)$ .AND. ((PC+1)	0 0 X X
08	2	3	ORI R_A,n	OR scratchpad register Immediate	$R_A \leftarrow (R_A)$ .OR. ((PC)+1)	0 0 X X
09	2	3	EXI R_A,n	EX-OR scratchpad register Immediate	$R_A \leftarrow (R_A)$ .EX-OR. ((PC)+1)	0 0 X X
13	2	3	ADDI R_A,n	ADD to scratchpad register Immediate	$R_A \leftarrow (R_A)+((PC)+1)$	X X X X
14	2	3	SUBI R_A,n	SUBTRACT scratchpad register Immediate	$R_A \leftarrow (R_A)-((PC)+1)$	X X X X
15	2	3	LI R_A,n	LOAD scratchpad register Immediate	$R_A \leftarrow ((PC)+1)$	No change
16	2	3	LIQ n	LOAD Q register Immediate	$Q \leftarrow ((PC)+1)$	No change
			5. Compare Instructions			
23	2	3	CI R_A,n	Compare scratchpad register Immediate	$(R_A)-((PC)+1)$	X X X X
24	1	3	CR R_A,R_B	COMPARE scratchpad registers	$(R_A)-(R_B)$	X X X X
25	1	5	CM R_A	COMPARE scratchpad register to Memory	$(R_A)-((R_F))$ $R_F \leftarrow (R_F)+1$	X X X X
			6. Memory Reference			
20	1	5	ADDM R_A	ADD memory to scratchpad reg.	$R_A \leftarrow (R_A)+((R_F))$ $R_F \leftarrow (R_F)+1$	X X X X
21	1	5	MOVM R_A	MOVE memory to scratchpad reg.	$R_A \leftarrow ((R_F))$ $R_F \leftarrow (R_F)+1$	0 0 X X
30	1	5	LRM R_A	MOVE scratchpad reg. to memory	$((R_F)) \leftarrow (R_A)$ $R_F \leftarrow (R_F)+1$	No change
			7. Shift and Rotate			
33	1	4	ROTR R_A	Rotate right scratchpad register by two positions	$D_n \leftarrow (D_{n+2})$; $D_{15} \leftarrow (D_1)$	0 0 X X
38	1	3	SRI R_A	Shift right scratchpad register 1 position. "1" shifter to MSB	$D_n \leftarrow (D_n+1)$; $D_{15} \leftarrow 1$	0 0 X X
3C	1	3	SRO R_A	shift right scratchpad register 1 position . "0" shifted to MSB		

opCode (HEX)	WORDS	CYCLES	Mnemonic	Meaning	Action	Flags C C N Z
39	1	3	SL1 R_A	Shift left scratchpad register 1 position. "0" shifted to LSB	$D_{n+1} \leftarrow (D_n); D_o \leftarrow 0$	0 0 X X
3D	1	3	ROTR1 R_A	Rotate right scratchpad register 1 position	$D_n \leftarrow (D_{n+1}); D_{15} \leftarrow (D_o)$	0 0 X X
3E	1	3	ROTL1 R_A	Rotate left scratchpad register 1 position	$D_{n+1} \leftarrow (D_n); D_o \leftarrow (D_{15})$	0 0 X X
			8. Interrupt Instructions			
34	1	3	EI	Enable INTERRUPT	INTE ← 0	No change
35	1	3	DI	Disable INTERRUPT	INTE ← 1	No change
36	2	3	SM n*	Load Mask register	Mask register ← ((PC)+1)	No change
37	2	3	LC nn	Load Timer Counter	Timer Counter ← ((PC)+1)	No change
3A	1	3	RETI	Return from Interrupt	SP ← (SP)-1 PC ← ((SP))	No change
			9. JUMP Instructions			
26	2	3	JUMP nn	JUMP Unconditionally	PC ← ((PC)+1	No change
27	2	4	JNZ nn	Jump on non zero	PC ← ((PC)+1); if Z=0 PC ← (PC)+2 ; if Z=1	No change
28	2	4	JZ nn	JUMP on zero	PC ← ((PC)+1); if Z=1 PC ← (PC)+2 ; if Z=0	No change
29	2	4	JP nn	JUMP on positive	PC ← ((PC)+1); if N=0 PC ← (PC)+2 ; if N=1	No change
2A	2	4	JN nn	JUMP on negative	PC ← ((PC)+1); if N=1 PC ← (PC)+2 ; if N=0	No change
2E	2	4	JNC nn	JUMP on non carry	PC ← ((PC)+1); if C=0 PC ← (PC)+2 ; if C=1	No change
2B	3	5	JE R_A,nn,mm	JUMP on equal to	PC ← ((PC)+2); if (R_A)=((PC)+1) PC ← (PC)+3; if (R_A) ≠ ((PC)+1)	See Note 1
2D	3	5	JL R_A,nn,mm	JUMP on less then	PC ← ((PC)+2); if (R_A) < ((PC)+1) PC ← (PC)+3 ; if (R_A) ≥ ((PC)+1)	See Note 1

opCode (HEX)	WORDS	CYCLES	Mnemonic	Meaning	Action	Flags CO NZ
2C	2	5	DSZ R_A, nn	Decrement and skip on zero	$R_A \leftarrow (R_A)-1$ $PC \leftarrow ((PC)+1)$; if $(R_A) \neq 0$ $PC \leftarrow (PC)+2$; if $(R_A) = 0$	See Note 2
			10. SUBROUTINE Instructions			
2F	2	3	CALL nn	Call to Subroutine	$SP \leftarrow (SP)+1$ $((SP)) \leftarrow (PC)$ $PC \leftarrow ((PC)+1)$	No change
32	1	3	RET	Return from Subroutine	$SP \leftarrow (SP)-1$ $PC \leftarrow ((SP))+1$	
			11. OTHER Instructions			
3B	1	3	NOP	No Operation	$PC \leftarrow (PC)+1$	No change
22	1	-	WAIT	WAIT	Disable clock from the processor	No change
1F	2	4	INS R_A,n	Input to scratchpad register from the 'input port n'	$R_A \leftarrow$ (Input Port n)	0 0 X X
1E	2	4	OUTS R_A,n	Output from scratchpad register to the 'output port n'	Output Port n $\leftarrow (R_A)$	No change

Note 1 : Status changed according to the execution of the 'CI' instruction

Note 2 : Status changed according to the execution of the 'DECR' instruction.

'x' denotes that the status flag is set or reset according to the operation performed.

'aux.' denotes the auxiliary registers

* 8 higher byte of a 16 bit data

1 cycle = 1 microsecond

APPENDIX B

In order to halp the programmer examine the memory when debugging programs, this appendix provides the UMM-16 instruction set in increasing OPCODE order.

Opcode (Hex)	Mnemonic
01	ANDR R_A,R_B
02	ORR R_A,R_B
03	EXR R_A,R_B
04	COMR R_A
05	COMQ
06	TCOM
07	ANDI R_A,nn
08	ORI R_A,nn
09	EXI R_A,nn
0A	ADDR R_A,R_B
0B	ADDQ R_A,R_B
0C	SUBR
0D	INCR R_A
0E	INCQ
0F	DECR R_A
10	DECQ
11	CLRQ
12	CLRR R_A
13	ADDI R_A,nn
14	SUBI R_A,nn
15	LI R_A,nn
16	LIQ nn
17	MOVR R_A,R_B
18	MOVQ R_A
19	LRQ R_{A_1}
1A	PUSH R^1_A,R_B
1B	POP R^1_A,R_B
1C	MOVS R_A
1D	LRS R_A
1E	OUTS R_A,n
1F	INS R_A,n
20	ADDM R_A
21	MOVM R_A
22	WAIT

Opcode (Hex)	Mnemonic
23	CI R_A,nn
24	CR R_A,R_B
25	CM R_A
26	JUMP nn
27	JNZ nn
28	JZ nn
29	JP nn
2A	JN nn
2B	JE R_A,nn,nn
2C	DSZ R_A,n
2D	JL R_A,nn,nn
2E	JNC nn
2F	CALL nn
30	LRM R_A
31	LRP R_A
32	RET
33	ROTR R_A
34	EI
35	DI
36	SM n
37	LC nn
38	SRI R_A
39	SLI R_A
3A	RETI
3B	NOP
3C	SRO R_A
3D	ROTRI R_A
3E	ROTLI R_A

APPENDIX C

ASCII Character codes in Hexadecimal

(a) = 8-bit, even parity
(b) = 7-bit, no parity

(a)	(b)	Char	(a)	(b)	Char	(a)	(b)	Char	(a)	(b)	Char
00	00	NUL	A0	20	SP	C0	40	@	60	60	`
81	01	SOH	21	21	!	41	41	A	E1	61	a
82	02	STX	22	22	"	42	42	B	E2	62	b
03	03	ETX	A3	23	#	C3	43	C	63	63	c
84	04	EOT	24	24	$	44	44	D	E4	64	d
05	05	ENQ	A5	25	%	C5	45	E	65	65	e
06	06	ACK	A6	26	&	C6	46	F	66	66	f
87	07	BEL	27	27	'	47	47	G	E7	67	g
88	08	BS	28	28	(	48	48	H	E8	68	h
09	09	HT	A9	29	)	C9	49	I	69	69	i
0A	0A	LF	AA	2A	*	CA	4A	J	6A	6A	j
8B	0B	VT	2B	2B	+	4B	4B	K	EB	6B	k
0C	0C	FF	AC	2C	,	CC	4C	L	6C	6C	l
8D	0D	CR	2D	2D	-	4D	4D	M	ED	6D	m
8E	0E	SO	2E	2E	.	4E	4E	N	EE	6E	n
0F	0F	SI	AF	2F	/	CF	4F	O	6F	6F	o
90	10	DLE	30	30	0	50	50	P	F0	70	p
11	11	DC1	B1	31	1	D1	51	Q	71	71	q
12	12	DC2	B2	32	2	D2	52	R	72	72	r
93	13	DC3	33	33	3	53	53	S	F3	73	s
14	14	DC4	B4	34	4	D4	54	T	74	74	t
95	15	NAK	35	35	5	55	55	U	F5	75	u
96	16	SYN	36	36	6	56	56	V	F6	76	v
17	17	ETB	B7	37	7	D7	57	W	77	77	w
18	18	CAN	B8	38	8	D8	58	X	78	78	x
99	19	EM	39	39	9	59	59	Y	F9	79	y
9A	1A	SUB	3A	3A	:	5A	5A	Z	FA	7A	z
1B	1B	ESC	BB	3B	;	DB	5B	[	7B	7B	{
9C	1C	FS	3C	3C	<	5C	5C	\	FC	7C	|
1D	1D	GS	BD	3D	=	DD	5D	]	7D	7D	}
1E	1E	RS	BE	3E	>	DE	5E	↑	7E	7E	~
9F	1F	VS	3F	3F	?	5F	5F	←	FF	7F	DEL

CR = carriage return
FF = form (line) feed
HT = horizontal tab
VT = vertical tab

APPENDIX E

UMM-16 APPLICATIONS MANUAL

THE CITY UNIVERSITY

USER MICROPROGRAMMABLE BIT-SLICE MICROCOMPUTER

(UMM-16)

APPLICATIONS MANUAL

by

Dogan Ibrahim

Northampton Square London EC1V 0HB
telephone: 01-253 4399 telex: 263896

UMM-16 MICROCOMPUTER

APPLICATIONS MANUAL

by

Dogan Ibrahim,

The City University, 1980

CONTENTS

INTRODUCTION

The UMM-16 is a complete 16 bit* user microprogrammable microcomputer, designed from bipolar bit-slice LSI building blocks.

The UMM-16 consists of two basic systems : the 'Bit-Slice Microcomputer', and the 'Support Processor'. The Support Processor is an F8 microcomputer used to load the user defined microinstructions into the writeable control memory (microprogram RAM) of the Bit-Slice system.

There are 62 general purpose fixed instructions built into the Bit-Slice Microcomputer, and many more instructions can be invented and implemented by the user.

This manual gives examples of using the fixed instructions of the UMM-16. More details are available in the UMM-16 User's Manual.

* Although the UMM-16 can operate both with 8 and 16 bits of data length, this manual describes the operation when working with 16 bits of data length .

SECTION-1

The UMM-16 Microcomputer

1. ARCHITECTURE OF UMM-16

UMM-16 is an 8/16 bit* user imcroprogrammable microcomputer designed with bit-slice Bipolar devices. A block diagram of the internal architecture of UMM-16 is shown in Figure 1. Data and instructions are transmitted along the 16 bit bidirectional Data Bus. The addresses for the memory are sent along the 16 bit Address Bus.

The Arithmetic Processing Unit (APU) is the subsystem where computing and data manipulation are carried out. Arithmetic and logical operations, data shifting and rotation and the generation of status signals are all functions of the APU.

The Computer Control Unit (CCU) is the part that controls all of the other subsystems, fetches and decodes the instructions residing in the memory and sends the appropriate signals to other parts of the computer to perform the required operations. At the heart of the CCU is the microprogram memory (a 48 bit wide memory) which is designed from a combination of PROM and RAM memories. The PROM part of the microprogram memory is loaded with the control data (microinstructions) to execute the fixed macroinstructions (only 16 bits of operation available at present) of the UMM-16. The RAM part of the microprogram memory can be loaded (via a Support Processor) with the user defined microinstructions to execute the macroinstructions invented by the user.

The Program Control Unit (PCU) generates the memory addresses, steps through the addresses in the memory, performs branch, conditional branch and subroutine call and return operations under the control of the CCU.

1.1 UMM-16 Registers available to the User

All of the registers of UMM-16 are 16 bits wide (except the Status register and the Mask register). The Arithmetic Processing Unit contains 17 internal working registers (16 scratchpad registers and a 'Q' register). In addition to these, 16 more registers "auxiliary registers" are provided making a total of 33 general purpose registers available to the user (see Figure 2). The APU can only perform operations upon any 17 internal working registers. The data in any of the 16 scratchpad registers can be transferred to the auxiliary registers during the execution of certain instructions. Similarly, the data in any auxiliary register can be transferred to any of the 16 scratchpad registers during the execution of certain instructions.

The APU also contains a "Status Register ('S' register)" which contains the four ALU status (carry, overflow, sign and zero). The status register is loaded with new data after every arithmetic or logic operation. The definition of the ALU status bits are as follows:

* The 16 bit fixed instruction set part is described in this manual.

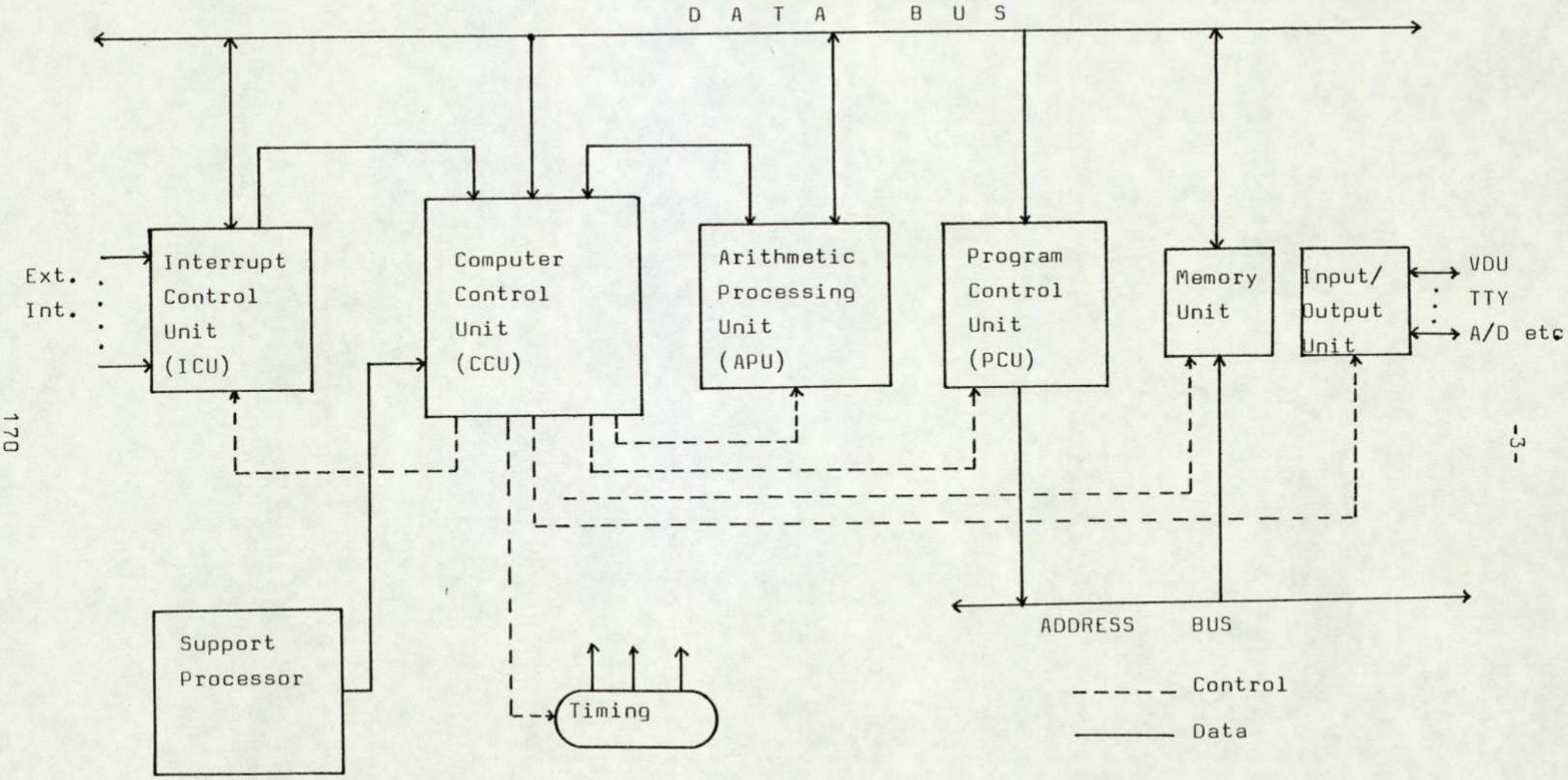


Figure 1. Simplified block diagram of the UMM-16 .

i. The 'carry' flag, C_{n+4} : This flag indicates whether an operation causes a 'carry' into the next higher order digit. If a 'carry' has occured then this flag is set ($C_{n+4} = 1$); otherwise it is reset ($C_{n+4} = 0$).

ii. The 'overflow' flag, OVR: This flag is set (OVR = 1) when two's complement overflow results from an arithmetic operation. In arithmetic operations, the OVR flag is reset (OVR = 0) if two's complement overflow does not occur.

iii. The 'sign' flag, N : The 'sign' flag is set (N = 1) if the most significant bit of the result is set (equal to 1). This indicates that the two's complement number, represented by the bit pattern of the result, is negative. The 'sign' flag is reset (N = 0) if the most significant bit of the result is reset (equal to zero), indicating that the two's complement number represented by the result is zero or positive.

iv. The 'zero' flag, Z : This flag is set (Z = 1) if the result of an arithmetic operation is zero, and is reset (Z = 0) if the result is not zero.

The data in the status register can be transferred to any of the 16 scratchpad registers (or vice-versa) during the execution of certain instructions.

The Program Control Unit of the UMM-16 contains a 16 bit "Program Counter Register" and a 4 word deep 'stack' controlled by an internal stack pointer. The stack is used during subroutine or interrupt operations where it is required to save the current value of the Program Counter Register. The data in the scratchpad registers can be transferred to the Program Counter Register.

The Interrupt Control Unit of the UMM-16 provides two registers to the user: the "Mask register" and the "Counter register". The "Mask register" is used to mask any interrupt inputs and the "Counter register" is used during the Timer Interrupt routines.

Figure 2 shows the registers of UMM-16 that are available to the user. Note that the scratchpad register 'F' is given a special name, the "Data Counter". The "Data Counter" is used during memory reference instructions. It is incremented by one after every instruction that transfers data to or from memory.

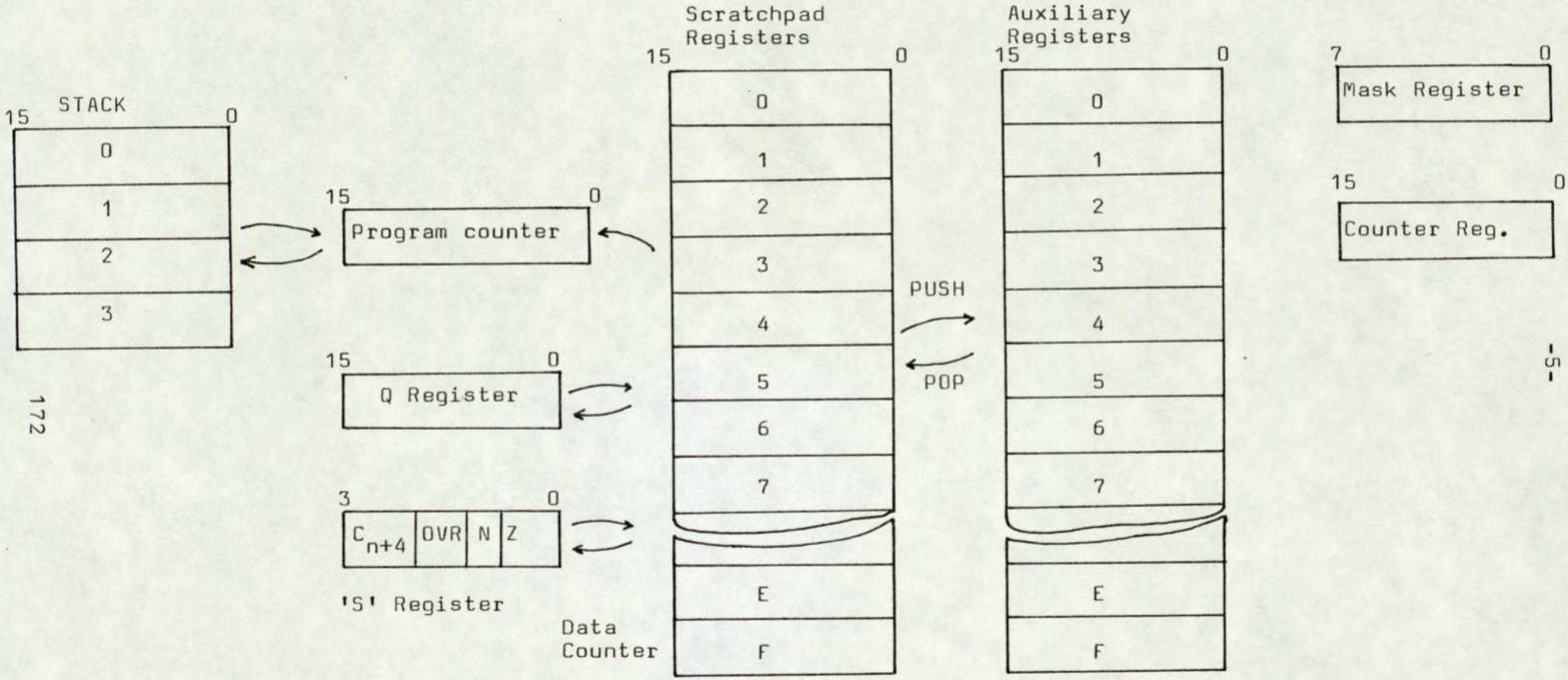


Figure 2. Registers available to the user .

1.2 Abbreviations

C_{n+4} or C	The carry flag
D_n	nth bit of the scratchpad register
INTE	Interrupt enable flip flop
N	The sign flag
n	Any 8 bit data
nn	Any 16 bit data
OVR or O	The overflow flag
PC	The Program Counter Register
Q	The 'Q' register
R_A	The scratchpad register addressed by the contents of 4 bit operand R_A
R_A^1	The auxiliary register addressed by the contents of 4 bit operand R_A
R_B	The scratchpad register addressed by the contents of 4 bit operand R_B
R_B^1	The auxiliaryregister addressed by the contents of 4 bit operand R_B
R_D	The scratchpad register D
R_F	The scratchpad register F (The Data Counter)
S	The status register
SP	The stack pointer
Z	The zero flag
(R)	Contents of register R
$(\overline{R})$	Complement of the contents of register R
((R))	Contents of location addressed (pointed to) by register R
$\longleftarrow$	Receives
$\longrightarrow$	Goes to
$\longleftrightarrow$	Is exchanged with
+	Arithmetic addition
-	Arithmetic subtraction
.OR.	Logical OR
.AND.	Logical AND
.EX-OR.	Logical exclusive -OR
X	Four bit don't care entry

1.3 UMM-16 Instruction Set Summary

opCode (HEX)	WORDS	CYCLES	Mnemonic	Meaning	Action	Flags C O N Z
			1. Register Transfers			
17	1	4	MOVR R_A,R_B	Move scratchpad registers	$R_A \leftarrow (R_B)$	No change
18	1	3	MOVQ R_A	Move Q to scratchpad register	$R_A \leftarrow (Q)$	No change
19	1	3	LRQ R_A	Move scratchpad register to Q	$Q \leftarrow (R_A)$	No change
1A	1	3	PUSH $R_A{}^1$,R_B	Move scratchpad register to aux.	$R_A{}^1 \leftarrow (R_B)$	No change
1B	1	3	POP $R_A{}^1$,R_B	Move aux. to scratchpad register	$R_B \leftarrow (R_A{}^1)$	0 0 X X
1C	1	3	MOVS R_A	Move S to scratchpad register	$R_A \leftarrow (S)$	No change
1D	1	3	LRS R_A	Move scratchpad register to S	$S \leftarrow (R_A)$	X X X X
31	1	3	LRP R_A	Move scratchpad register to PC	$PC \leftarrow (R_A)$	No change
			2. Register Logical			
01	1	3	ANDR R_A,R_B	AND Scratchpad registers	$R_A \leftarrow (R_A).AND.(R_B)$	0 0 X X
02	1	3	ORR R_A,R_B	OR scratchpad registers	$R_A \leftarrow (R_A).OR.(R_B)$	0 0 X X
03	1	3	EXR R_A,R_B	EX-OR scratchpad registers	$R_A \leftarrow (R_A).EX\text{-}OR.(R_B)$	0 0 X X
04	1	3	COMR R_A	COMPLEMENT scratchpad regst.	$R_A \leftarrow (\overline{R}_A)$	X X X X
05	1	3	COMQ	COMPLEMENT Q register	$Q \leftarrow (\overline{Q})$	X X X X
06	1	3	TCOM R_A	2'S COMPLEMENT scratchpad register	$R_A \leftarrow (\overline{R}_A) + 1$	X X X X
			3. Register Arithmetic			
0A	1	3	ADDR R_A,R_B	ADD scratchpad registers	$R_A \leftarrow (R_A)+(R_B)$	X X X X
0B	1	3	ADDQ R_A,R_B	ADD Q to scratchpad register	$R_B \leftarrow (R_A)+(Q)$	X X X X
0C	1	3	SUBR R_A,R_B	SUBTRACT scratchpad registers	$R_A \leftarrow (R_A)-(R_B)$	X X X X
0D	1	3	INCR R_A	INCREMENT scratchpad register	$R_A \leftarrow (R_A)+1$	X X X X
0E	1	3	INCQ	INCREMENT Q register	$Q \leftarrow (Q)+1$	X X X X
0F	1	3	DECR R_A	DECREMENT scratchpad register	$R_A \leftarrow (R_A)-1$	X X X X
10	1	3	DECQ	DECREMENT Q register	$Q \leftarrow (Q)-1$	X X X X
11	1	3	CLRQ	CLEAR Q register	$Q \leftarrow 0$	0 0 X X
12	1	3	CLRR	CLEAR scratchpad register	$R_A \leftarrow 0$	0 0 X X

opCode (HEX)	WORDS	CYCLES	Mnemonic	Meaning	Action	Flags C O N Z
			4. Register Immediate			
07	2	3	ANDI R_A,n	AND scratchpad register Immediate	$R_A \leftarrow (R_A)$ .AND. ((PC)+1)	0 0 X X
08	2	3	ORI R_A,n	OR scratchpad register Immediate	$R_A \leftarrow (R_A)$ .OR. ((PC)+1)	0 0 X X
09	2	3	EXI R_A,n	EX-OR scratchpad register Immediate	$R_A \leftarrow (R_A)$ .EX-OR. ((PC)+1)	0 0 X X
13	2	3	ADDI R_A,n	ADD to scratchpad register Immediate	$R_A \leftarrow (R_A)+((PC)+1)$	X X X X
14	2	3	SUBI R_A,n	SUBTRACT scratchpad register Immediate	$R_A \leftarrow (R_A)-((PC)+1)$	X X X X
15	2	3	LI R_A,n	LOAD scratchpad register Immediate	$R_A \leftarrow ((PC)+1)$	No change
16	2	3	LIQ n	LOAD Q register Immediate	$Q \leftarrow ((PC)+1)$	No change
			5. Compare Instructions			
23	2	3	CI R_A,n	Compare scratchpad register Immediate	$(R_A)-((PC)+1)$	X X X X
24	1	3	CR R_A,R_B	COMPARE scratchpad registers	$(R_A)-(R_B)$	X X X X
25	1	5	CM R_A	COMPARE scratchpad register to Memory	$(R_A)-((R_F))$ $R_F \leftarrow (R_F)+1$	X X X X
			6. Memory Reference			
20	1	5	ADDM R_A	ADD memory to scratchpad reg.	$R_A \leftarrow (R_A)+((R_F))$ $R_F \leftarrow (R_F)+1$	X X X X
21	1	5	MOVM R_A	MOVE memory to scratchpad reg.	$R_A \leftarrow ((R_F))$ $R_F \leftarrow (R_F)+1$	0 0 X X
30	1	5	LRM R_A	MOVE scratchpad reg. to memory	$((R_F)) \leftarrow (R_A)$ $R_F \leftarrow (R_F)+1$	No change
			7. Shift and Rotate			
33	1	4	ROTR R_A	Rotate right scratchpad register by two positions	$D_n \leftarrow (D_{n+2})$; $D_{15} \leftarrow (D_1)$	0 0 X X
38	1	3	SRI R_A	Shift right scratchpad register 1 position. "1" shifter to MSB	$D_n \leftarrow (D_{n+1})$; $D_{15} \leftarrow 1$	0 0 X X
3C	1	3	SRO R_A	shift right scratchpad register 1 position . "0" shifted to MSB		

opCode (HEX)	WORDS	CYCLES	Mnemonic	Meaning	Action	Flags C O N Z
39	1	3	SL1 R_A	Shift left scratchpad register 1 position. "0" shifted to LSB	$D_{n+1} \leftarrow (D_n); D_o \leftarrow 0$	0 0 X X
3D	1	3	ROTR1 R_A	Rotate right scratchpad register 1 position	$D_n \leftarrow (D_{n+1}); D_{15} \leftarrow (D_o)$	0 0 X X
3E	1	3	ROTL1 R_A	Rotate left scratchpad register 1 position	$D_{n+1} \leftarrow (D_n); D_o \leftarrow (D_{15})$	0 0 X X
			8. Interrupt Instructions			
34	1	3	EI	Enable INTERRUPT	INTE ← 0	No change
35	1	3	DI	Disable INTERRUPT	INTE ← 1	No change
36	2	3	SM n*	Load Mask register	Mask register ← ((PC)+1)	No change
37	2	3	LC nn	Load Timer Counter	Timer Counter ← ((PC)+1)	No change
3A	1	3	RETI	Return from Interrupt	SP ← (SP)-1 PC ← ((SP))	No change
			9. JUMP Instructions			
26	2	3	JUMP nn	JUMP Unconditionally	PC ← ((PC)+1	No change
27	2	4	JNZ nn	Jump on non zero	PC ← ((PC)+1); if Z=0 PC ← (PC)+2 ; if Z=1	No change
28	2	4	JZ nn	JUMP on zero	PC ← ((PC)+1); if Z=1 PC ← (PC)+2 ; if Z=0	No change
29	2	4	JP nn	JUMP on positive	PC ← ((PC)+1); if N=0 PC ← (PC)+2 ; if N=1	No change
2A	2	4	JN nn	JUMP on negative	PC ← ((PC)+1); if N=1 PC ← (PC)+2 ; if N=0	No change
2E	2	4	JNC nn	JUMP on non carry	PC ← ((PC)+1); if C=0 PC ← (PC)+2 ; if C=1	No change
2B	3	5	JE R_A,nn,mm	JUMP on equal to	PC ← ((PC)+2); if $(R_A) = ((PC)+1)$ PC ← (PC)+3; if $(R_A) \neq ((PC)+1)$	See Note 1
2D	3	5	JL R_A,nn,mm	JUMP on less then	PC ← ((PC)+2); if $(R_A) < ((PC)+1)$ PC ← (PC)+3 ; if $(R_A) \geq ((PC)+1)$	See Note 1

opCode (HEX)	WORDS	CYCLES	Mnemonic	Meaning	Action	Flags CO NZ
2C	2	5	DSZ R_A, nn	Decrement and skip on zero	$R_A \leftarrow (R_A)-1$ $PC \leftarrow ((PC)+1)$; if $(R_A) \neq 0$ $PC \leftarrow (PC)+2$; if $(R_A) = 0$	See Note 2
			10. SUBROUTINE Instructions			
2F	2	3	CALL nn	Call to Subroutine	$SP \leftarrow (SP)+1$ $((SP)) \leftarrow (PC)$ $PC \leftarrow ((PC)+1)$	No change
32	1	3	RET	Return from Subroutine	$SP \leftarrow (SP)-1$ $PC \leftarrow ((SP))+1$	
			11. OTHER Instructions			
3B	1	3	NOP	No Operation	$PC \leftarrow (PC)+1$	No change
22	1	-	WAIT	WAIT	Disable clock from the processor	No change
1F	2	4	INS R_A,n	Input to scratchpad register from the 'input port n'	$R_A \leftarrow$ (Input Port n)	0 0 X X
1E	2	4	OUTS R_A,n	Output from scratchpad register to the 'output port n'	Output Port n $\leftarrow (R_A)$	No change

Note 1 : Status changed according to the execution of the 'CI' instruction

Note 2 : Status changed according to the execution of the 'DECR' instruction.

'x' denotes that the status flag is set or reset according to the operation performed.

'aux.' denotes the auxiliary registers

* 8 higher byte of a 16 bit data

1 cycle = 1 microsecond

SECTION - 2

Simple Programming Examples on the UMM-16

2. Programming Examples

This section will describe simple programming techniques to make the user familiar with using the UMM-16 instructions. The programs will be listed in the following format:

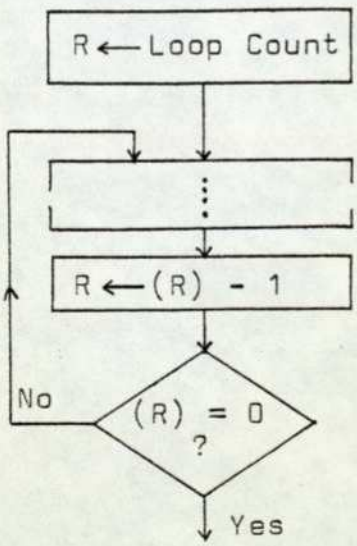
Label (optional) Mnemonic Comments (optional)

A flow diagram will be given for the examples where appropriate.

2.1 The LOOPING Structure (Do-while)

There are several methods that iteration (looping) can be performed in programs:

i Looping can be performed by loading a scratchpad register with the number of iterations required and then decrementing and testing the contents of that scratchpad register at the end of the loop :



Taking the scratchpad register 3 as the loop counter, the program will be as follows:

```
        LI  3,nn        ;  nn = 16 bit loop count
LOOP    .
        .
        .               ;  Program
        .
        .
        DECR 3          ;  Decrement Loop count
        JNZ LOOP        ;  If not zero go to LOOP
```

Number of memory locations required (except the program) = 5

Number of clock cycles (except for the program) = 7

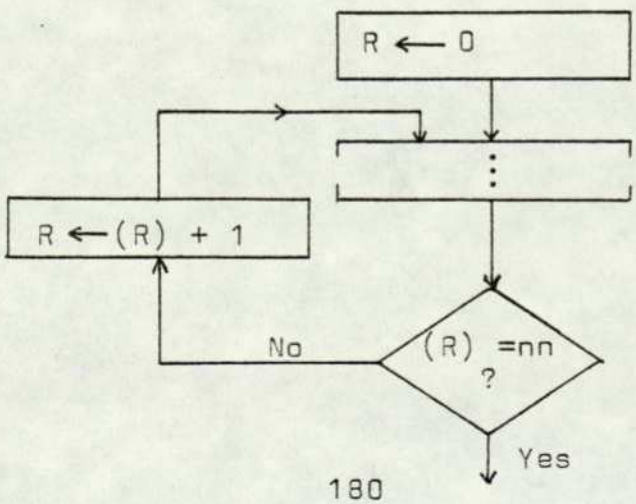
ii an alternative method for looping is to use the insturction DSZ (Decrement and Skip on zero):

```
        LI  3,nn        ;  nn = 16 bit loop count
LOOP    .
        .
        .
        .               ;  Program
        .
        .
        DSZ 3, LOOP     ;  Decrement loop count.  If not zero,
                           go to LOOP.  If zero, continue.
```

Number of memory locations required (except the program): 4

Number of clock cycles (except for the program): 5

iii Looping can also be performed by incrementing a register (initially cleared) and then testing to see if the required number of iterations have been performed:



Taking the scratchpad register 2 as the loop counter, the program will be as follows:

```
        CLR  3              ;  clear register 3
LOOP    .
        .
        .
        .                   ;  Program
        .
        .
        CR  3,nn            ;  Compare register 3 with loop count nn
        JZ OUT              ;  If zero, go out of loop.
        INCR 3              ;  Increment register 3
        JMP LOOP            ;  Go to LOOP for next iteration
OUT
```

Number of memory locations required (except the program): 8

Number of clock cycles (except for the program): 13

iv An alternative method to iii is to use instruction JE (Jump on equal to):

```
        CLR  3              ;  clear register 3
LOOP    .
        .
        .
        .                   ;  Program
        .
        .
        JE  3,nn,OUT        ;  Go to OUT if register 3 contains nn
        INCR 3              ;  Increment register 3
        JMP LOOP            ;  Go to LOOP for next iteration
OUT
```

Number of memory locations required (except the program): 7

Number of clock cycles (except for the program): 11

Comparing all the five methods it is clear that the second method (instruction DSZ) should be used when it is required to perform fast looping (iterations) in a program.

Example

Suppose that it is required to shift the contents of scratchpad

register 2 ten positions to the left. Assuming that the program will be stored starting from location 0400 of the memory, the program is as follows:

```
0400    1530            LI  3,H'000A'     ;  load loop count
0401    000A
0402    3920    LOOP    SL1 2             ;  shift left register 2
0403    2000            DSZ 3, LOOP       ;  perform looping
0404    0402
0405
```

2.2 Memory Reference Instructions

All memory reference instructions use the scratchpad register F (Data Counter) to address the memory work from which data is to be accessed. The contents of the Data Counter are incremented by one after every memory reference instruction. The following memory reference instructions are available on the UMM-16:

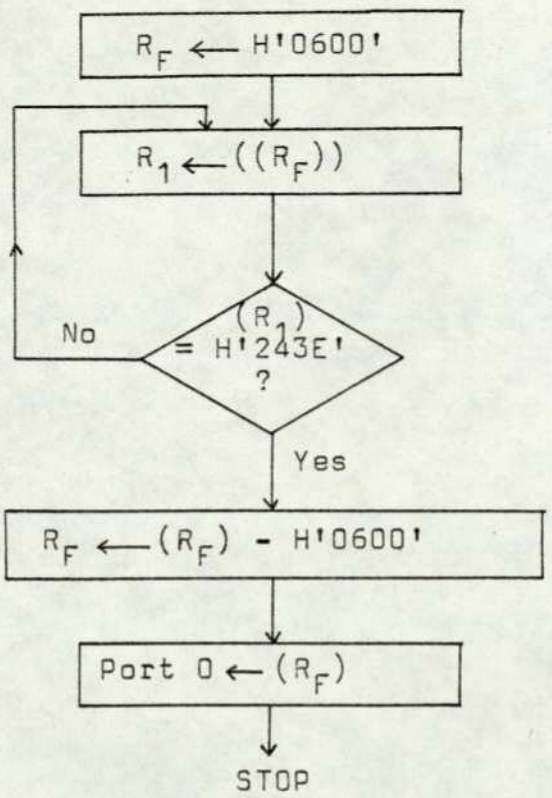
ADDM R_A	:	Add memory to scratchpad register
MOVM R_A	:	Move memory to scratchpad register
LRM R_A	:	Move scratchpad register to memory
CM R_A	:	Compare scratchpad register to memory.

Example 1

Count the number of ASCII characters up to and including a '$?', stored in memory from address 0600 onwards. Display the result via Port 0 .

0600	3230
0601	5240
0602	3521
0603	2461
0604	3041
0605	3321
0606	87A0
5243	
243E	'$?'

The flow diagram for this example is as follows:

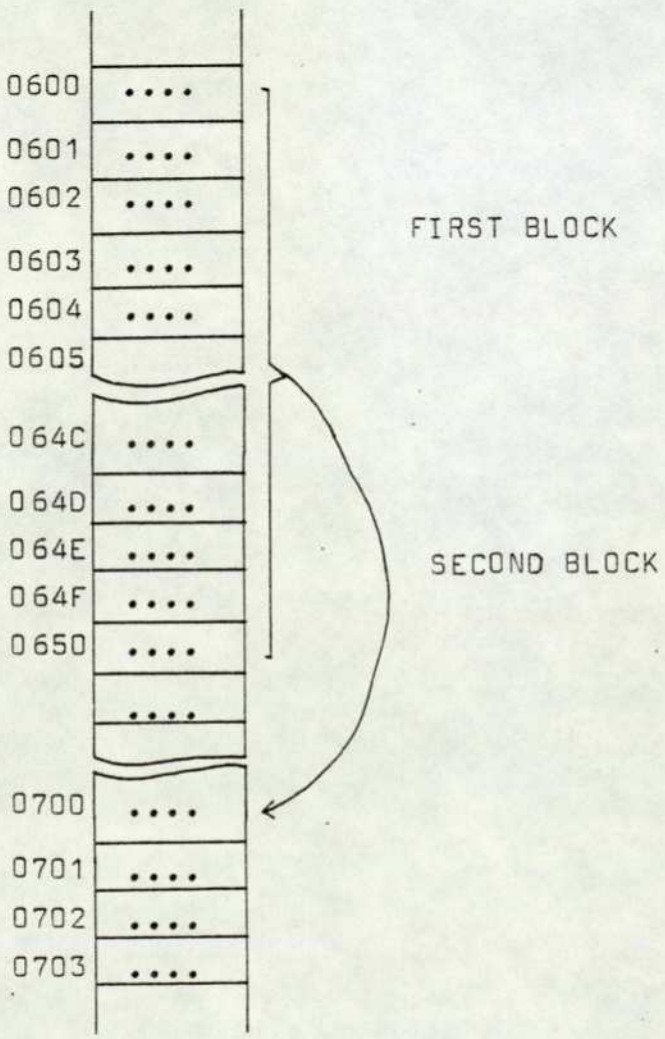


Assuming that the program will be stored starting from location 0400 of the memory, the program is as follows:

```
0400  1SF0        LI  F,H'0600'        ; Load Data Counter with
0401  0600                             ; the address H'o600'
0402  2110  BACK  MOVM 1               ; Move memory to register 1
0403  2B10        JE 1,H'243E',OUT     ; If register 1 contains H'243E'
0403  243E                             ; go to OUT, otherwise
0404  0407                             ; continue
0405  2600        JMP BACK             ; go to BACK
0406  0402
0407  14F0  OUT   SUBI F,H'0800'       ; Subtract H'0600' from the
0408  0600                             ; Data Counter
0409  1EF0        OUTS F,H'0000'       ; Output contents of the
040A  0000                             ; Data Counter via Port 0
040B  2200        WAIT                 ; Stop
```

Example 2

Copy the block of data starting from memory location 0600 and finishing at 0650 into the location starting from 0700.



Assuming that the program will be stored starting from location 0400 of the memory, the program is as follows:

```
0400   15F0   ONE     LI  F,H'0600'   ; load Data Counter with
0401   0600                           ; the start address
0402   1510   TWO     LI  1,H'0050'   ; load register 1 with
0403   0050                           ; the no. of words to be copied
0404   2120   LOOP    MOVM 2          ; Move memory to register 2
0405   13F0   THREE   ADDI F,H'00FF'  ; Move Data Counter to the
0406   00FF                           ; address in the second block
0407   3020   FOUR    LRM 2           ; Move register 2 to memory
0408   14F0   FIVE    SUBI F, H'0100'; Subtract H'0100' from
0409   0100                           ; the Data Counter
040A   2C10   SIX     DSZ 1,LOOP      ; If more to be copied
040B   0404                           ; go to LOOP
040C   2200           WAIT            ; Stop
```

The program steps proceed as follows:-

ONE Load the Data Counter with the starting address of the first block of data

TWO Load scratchpad register 1 with the number of words to be copied (H'0050'). H'0650' - H'0600' = H'0050'

LOOP Transfer the contents of the memory word addressed by the Data Counter to scratchpad register 2. The contents of the Data Counter are incremented by one.

THREE Add H'00FF' to the Data Counter so that is points to the appropriate address in the second block of memory.

FOUR Transfer the contents of scratchpad register 2 to memory.

FIVE Subtract H'0100' from the Data Counter so that it points to the appropriate address in the first block of memory.

SIX Test to see if more data is to be copied.

2.3 Manipulating Individual Bits

Immediate borlean instructions can be used to set or reset individual bits of any scratchpad register.

To reset one or more bits, AND the scratchpad register contents with a mask which is the complement of the bits to be reset. For example, the following instruction will reset bit 3 of scratchpad register 1:

```
ANDI 1, H'FFF7'
```

Similarly, individual bits can be set by OR ing the scratchpad registers with a mask which has a 1 in every bit position that is to be set. For example, bit 3 of scratchpad register 1 can be set to 1 as follows:

```
ORI 1, H'0004'
```

The data bits in any memory location or in a block of memory can be set or reset with the following instructions:

To reset:

```
.
.
LI  F,H'xxxx'        ; Load start of data address to Data Counter
MOVM R_A             ; Move memory to scratchpad register
ANDI R_A,H'mmmm'     ; Reset the required bits
DECR F               ; Decrement the Data Counter
LRM R_A              ; Move scratchpad register to memory
.
.
```

To set:

```
        .
        .
        .
        .
        LI   F,H'xxxx'        ; Load start of data address to Data Counter
        MOVM R_A              ; Move memory to scratchpad register
        ORI  R_A,H'mmmm'      ; Set the required bits
        DECR F                ; Decrement the Data Counter
        LRM  R_A              ; Move scratchpad register to memory
        .
        .
```

2.4 Testing for Status changes

There are many applications where it is necessary to keep a record of status for various control lines and to detect when individual control line statuses change and how they change.

Assume that scratchpad register 3 holds the status of 16 control lines. When new statuses are input from Port 0, they can be temporarily stored in scratchpad register 4. By EXCLUSIVE-OR ing the new and the old statuses, the statuses that changed can be detected. By AND ing the changed statuses with the old statuses the control lines that went from "on" to "off" can be identified. By EXCLUSIVE-OR ing this result with the changed statuses, those statuses which went from "off" to "on" can be identified.

The following program illustrates how the status changes can be detected and identified: (Assume scratchpad register 3 contains the old statuses)

```
ONE     INS 4,H'0000'     ; Input new statuses
TWO     MOVR 2,4          ; Save the new statuses
THREE   EXR 4,3           ; EXCLUSIVE-OR old and new statuses."Changed status"in 4
FOUR    ANDR 3,4          ; AND with old status
FIVE    MOVR 5,3          ; Save "statuses turned off' in 5
SIX     EXR 4,3           ; EXCLUSIVE-OR with changed status. Save "ststus
SEVEN   MOVR 3,2          ; turned on" in 4.  Save the new
                          ; statuses for next usage
```

Suppose the old status was:

Old Status: 1110111111111100

Suppose the new status is:

New Status: 1110111111110010

So that, bit 1 is turned on

bits 2 and 3 turned off

bits 1 and 2 changed

The result of instruction THREE is:

```
     Old Status: 1110111111111100
     New Status: 1110111111110010   EXR 4,3
Changed Ststuses: 0000000000001110
```

The result of instruction FOUR is:

```
Changed Statuses: 0000000000001110
Old Status      : 1110111111111100   ANDR 3,4
Turned off      : 0000000000001100
```

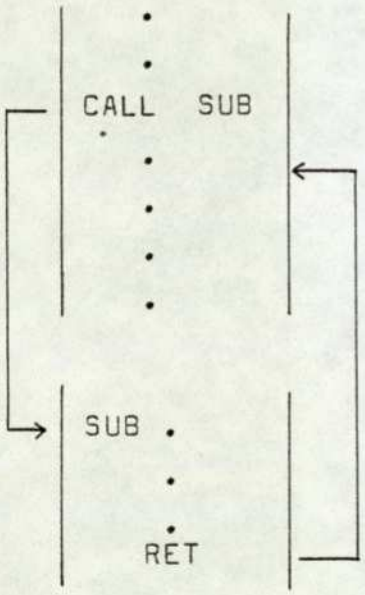
The result of instruction SIX is:

```
Turned off      : 0000000000001100
Changed Statuses: 0000000000001110   EXR 4,3
Turned on       : 0000000000000010
```

2.5 Programming with Subroutines

The CALL instruction is used to call a subroutine into execution. The starting address of a subroutine is identified by the subroutine name, which is the label of the first instruction to be executed in the subroutine.

Instruction CALL increments the stack pointer and saves the contents of the Program Counter Register (PC) in the stack and then loads PC with the subroutine starting address.



Example

Assume that it is required to multiply the contents of scratchpad register 1 by 3 at several points in a program. Defining the multiplication as a subroutine, the structure of the program will be as follows:

```
        .
        .
        CALL MULT           ;  Call to Subroutine MULT
        .
        .
        CALL MULT           ;  Call to Subroutine MULT
        .
        .
        CALL MULT           ;  Call to Subroutine MULT
        .
        .
        .
        .
        .
        .
MULT    MOVR 2,1            ;  Subroutine MULT
        SL1 1               ;  Multiply register 1 by 3
        ADDR 1,2            ;
        RET                 ;  Return from Subroutine
```

2.6 Nested Subroutines

"Nesting" is the term applied to subroutines being called from within other subroutines. Subroutines up to 4 levels can be nested in the M-16 computer.

The structure of a four-deep subroutine nest is as follows:

```
Main Program
        .
        .
        CALL SUB1
        .
        .
        .
        .
        .
SUB1    .
        .
        .
        CALL SUB2
        .
        .
        RET
```

```
SUB2
      .
      .
      CALL SUB3
      .
      .
      .
      RET

SUB3
      .
      .
      CALL SUB4
      .
      .
      RET

SUB4
      .
      .
      .
      .
      RET
```

The sequence in which four-deep subroutines are handled is shown in Figure 3 .

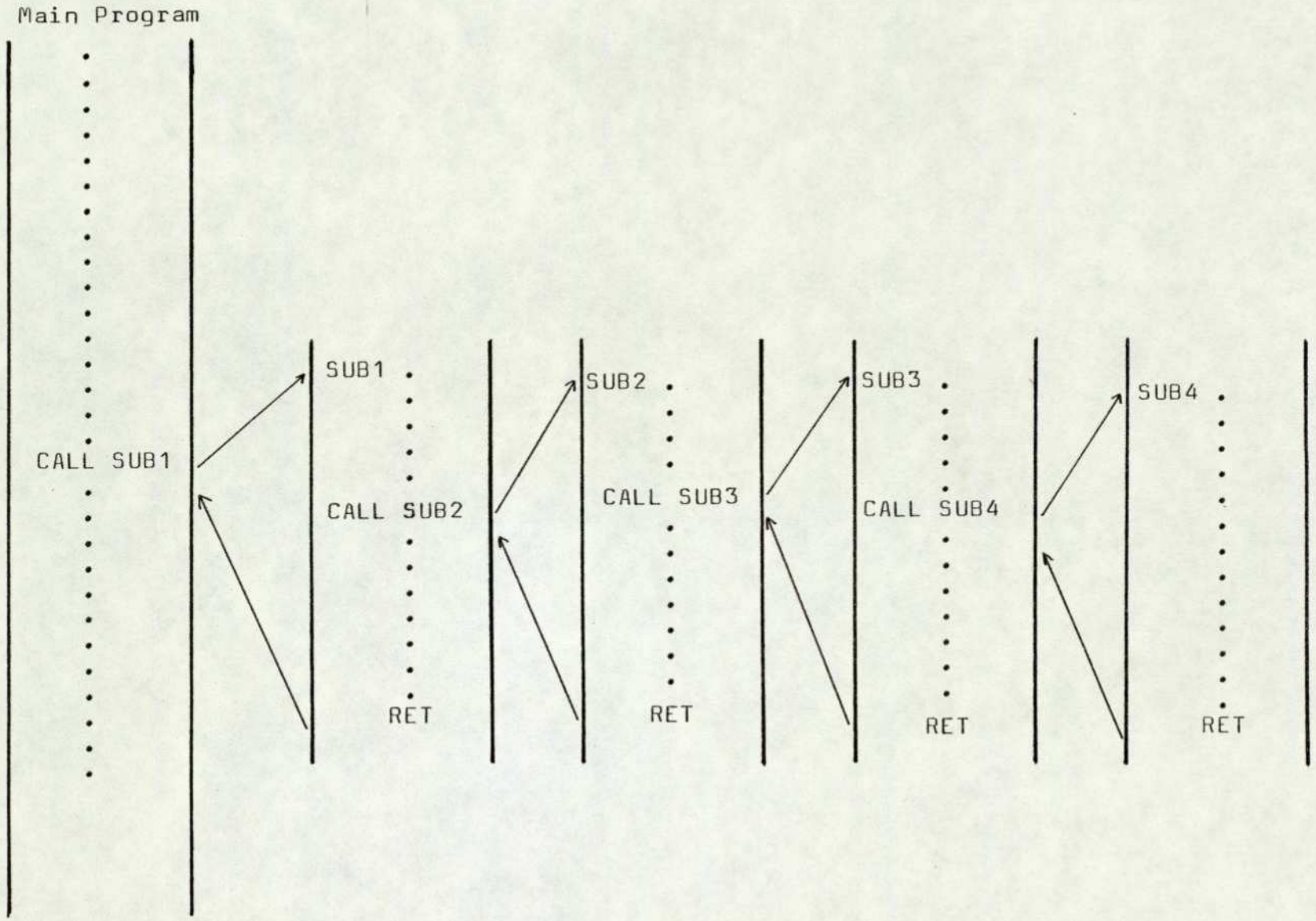


Figure 3. Four-level subroutine nesting .

2.7 Programming with the Teletype/VDU input and output subroutines

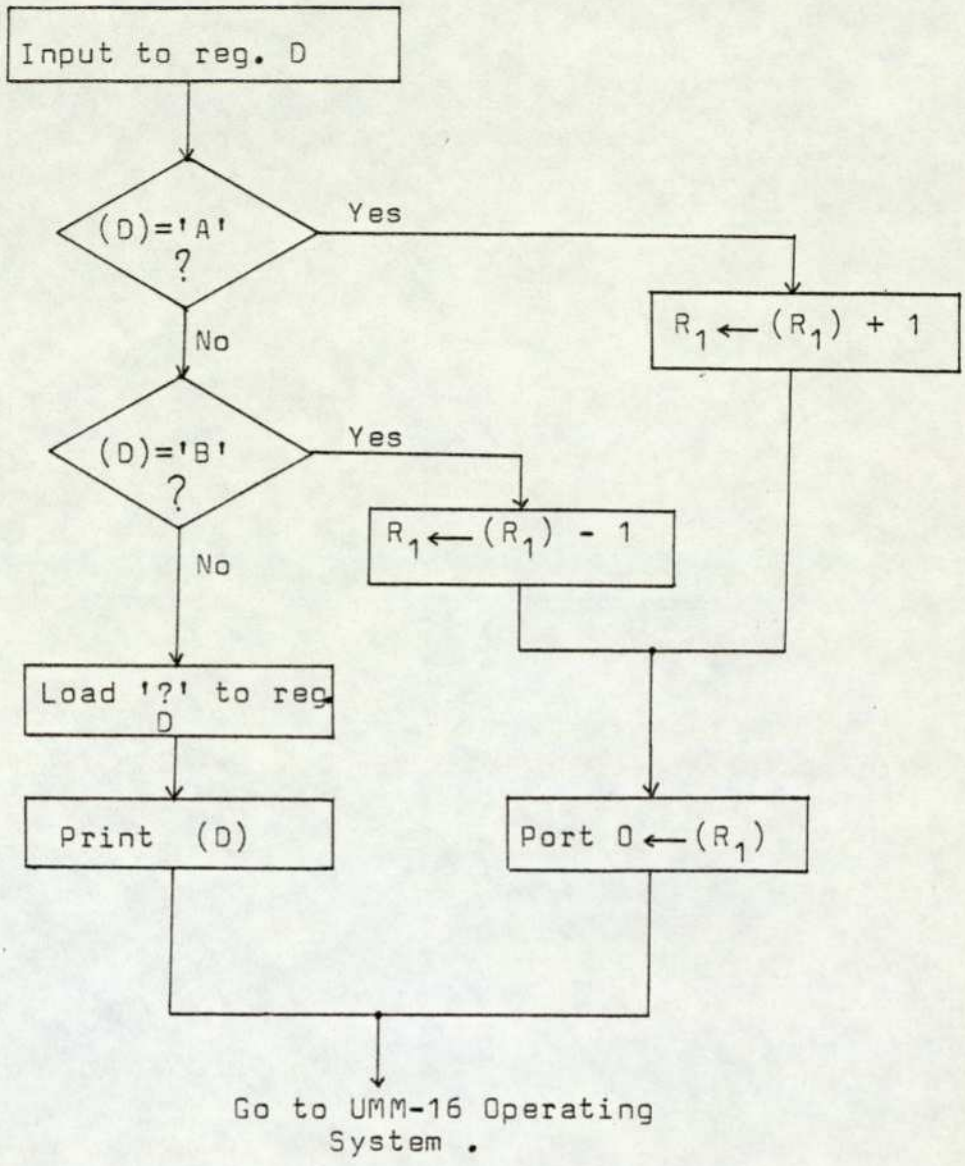
There are 13 subroutines available to the user for the teletype/VDU input and outpur routines (see details in the UMM-16 operating system). The scratchpad register D is used to transfer or receive data from the user terminal. This section will give some examples on the use of these subroutines in programs.

Example 1

It is required to write a program to accept two ASCII characters 'A' and 'B' from the user terminal. If character 'A' is typed, the contents of scratchpad register 1 will be incremented by one and the result will be displayed on Port 0. If character 'B' is typed then the contents of scratchpad register 1 will be decremented by one and the result will be displayed on Port 0. If an unrecognised character is typed then a '?' will be printed on the user terminal. In all cases the program will return back to the UMM-16 operating system after performing the required operation.

The flow diagram of the program is shown below:

(Note: 'A' = 41_H , 'B' = 42_H and '?' = 24_H in ASCII code)



Assuming that the program will be stored starting from address 0400 of the memory, the program is as follows:

```
0400  2F00          CALL TTYIN 3          ; receive a character
0401  0070                                ; from user terminal
0402  2F00          CALL TTYOUT 3         ; echo the character
0403  003B                                ;
0404  2BD0          JE D,H'0041',LOOP1    ; if character is 'A' go to
0405  0041                                ; LOOP 1
0406  040F                                ;
0407  2BD0          JE D,H'0042',LOOP2    ; if character is 'B' go to
0408  0042                                ; LOOP2
0408  0414
0409  15D0          LI  D,H'0024'         ; Load register 'D' with '?'
040A  0024
040B  2F00          CALL TTYOUT 3         ; Print contents of register
040C  003B                                ; 'D'
040D  2600          JMP H'oooo'           ; go to UMM-16 operating
040E  0000                                ; system
040F  0D10  LOOP 1  INCR 1                ; Increment register 1
0410  1E00  BACK    OUTS'1,H'oooo'        ; Display contents of
0411  0000                                ; register 1 via Porto
0412  2600          JMP H'oooo'           ; go to UMM-16 operating system
0413  0000
0414  0F10  LOOP 2  DECR 1                ; Decrement register 1
0415  2600          JMP BACK              ; go to display contents
0416  0410                                ; of register 1
```

The program can be entered into the memory by the 'M' command and can be executed by the 'E' command.

Example 2

Accept a 16-bit data from input Port 0 and print this data as 4 hexadecimal digits on the user terminal.

Assuming that the program will be stored starting from location 0400,

```
0400   2F00          INS D,H'0000'        ; Input via Port 0 to
0401   0000                               ; register 'D'
0402   2F00          CALL HEX1            ; Print the data in 'D' as
0403   00BE                               ; 4 hexadecimal digits
0404   2600          JMP H'0000'          ; go to UMM-16 operating
0405   0000                               ; system
```

Example 3

Accept two ASCII characters from the user terminal and display them at output Port 0.

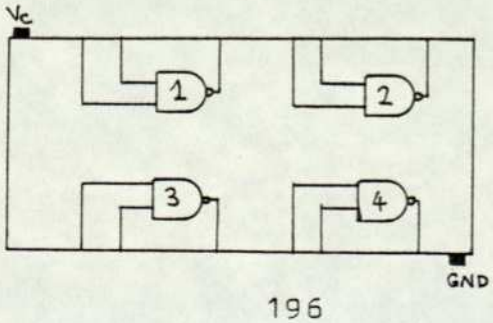
Assuming that the program will be stored starting from location 0400,

```
0400   2F00          CALL TTYINI          ; Input 2 ASCII characters
0401   0063                               ; to register D
0402   1ED0          OUTS D,H'0000'       ; output at Port 0
0403   0000
0404   2600          JMP H'0000'          ; go to UMM-16 operating
0405   0000                               ; system
```

Example 4

It is required to write a program to test the four NAND gates in the 7400 Integrated Circuits. Use I/O Port 0 to interface the NAND gates to the processor.

The 7400 IC contains four NAND gates:

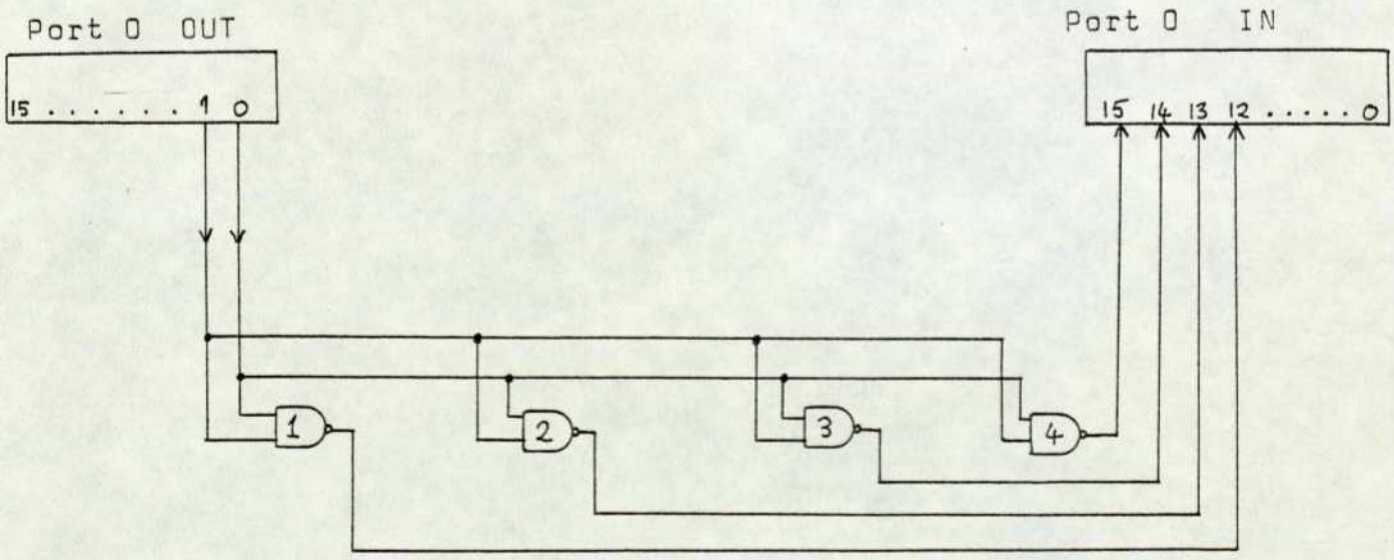


The processor is required to send the four possible input signals (00,01,10 and 11) to each gate and then test the output of each gate to see if the logic level is correct.

Assume that the results of the test will be printed on the user terminal as four ASCII characters corresponding to the four gates. The letter 'P (=Pass)' will indicate that a gate is working properly and the letter 'F (=Fail)' will indicate that a gate is faulty. Thus, if the letters 'PPFP' are printed on the user terminal then this will indicate that gate 3 is faulty and the others are working properly.

To simplify the programming and eliminate the need for large numbers of interface signals, connect the NAND gates to the processor I/O Port as follows:

The two inputs of all the NAND gates are connected to the two least significant bits of output Port 0



Assume that the program will be stored starting from location 0400 of the memory:

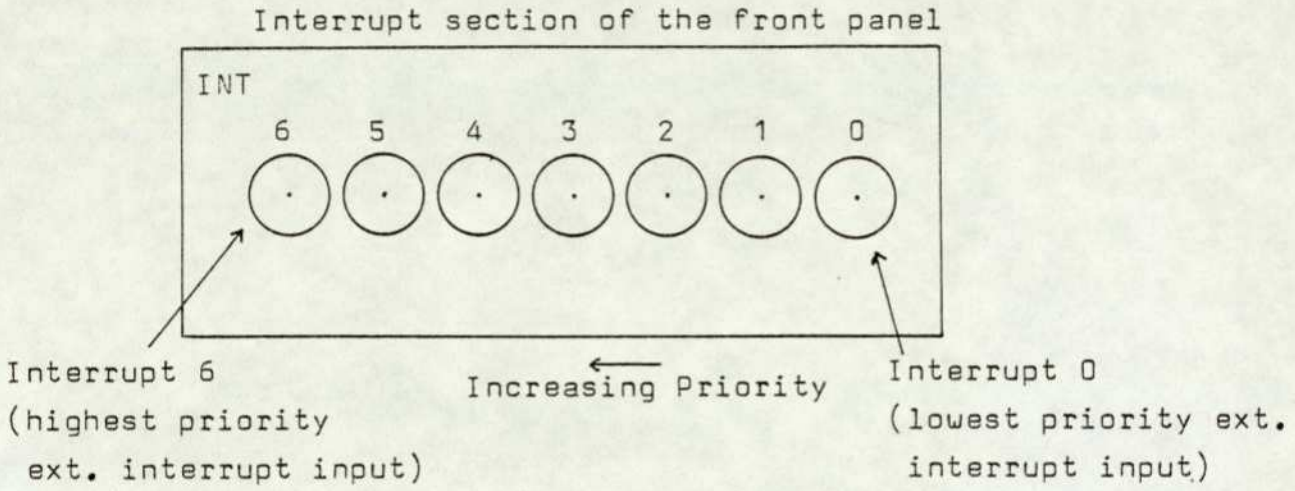
```
0400   1500          LI  0,H'1000'
0401   1000
0402   1210   LOOP   CLRR 1
0403   1E10   BACK   OUTS 1,H'0000'
0404   0000
0405   1F20          INS2,H'0000'
0406   0000
0407   2B10          JE 1,H'0003',OTHER
0408   0003
0409   0410
040A   0120          ANDR 2,0
040B   2800          JZ OUT
040C   041D
040D   0D10          INCR 1
040E   2600          JMP BACK
040F   0403
0410   0120   OTHER  ANDR 2,0
0411   2700          JNZ OUT
0412   041D
0413   15D0          LI  D,H'0050'
0414   0050
0415   2F00   PRINT  CALL TTYOUT 3
0416   003B
0417   2B00          JE 0,H'8000',H'0000'
0418   8000
0419   0000
041A   3900          SL1 0
041B   2600          JMP LOOP
041C   0402
041D   15D0   OUT     LI  D,H'0046'
041E   0046
041F   2600          JMP PRINT
0420   0415
```

SECTION - 3

Programming for the Interrupts

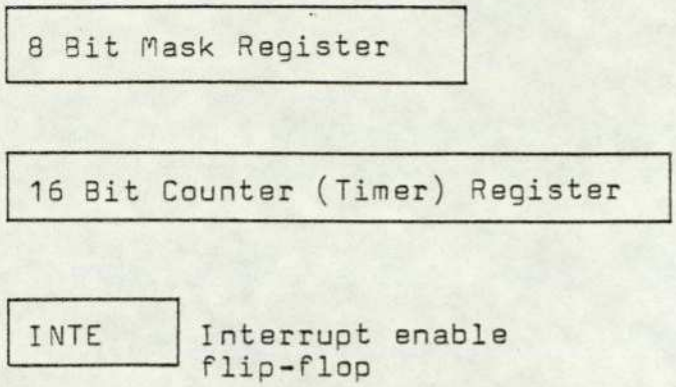
3. INTERRUPT Programming

The UMM-16 computer accepts seven external and one internal (timer) interrupts. The interrupt inputs are organised in a priority level and any interrupt can be masked under software control. The seven external interrupt inputs are mounted on the front panel of UMM-16 with an ascending priority level:

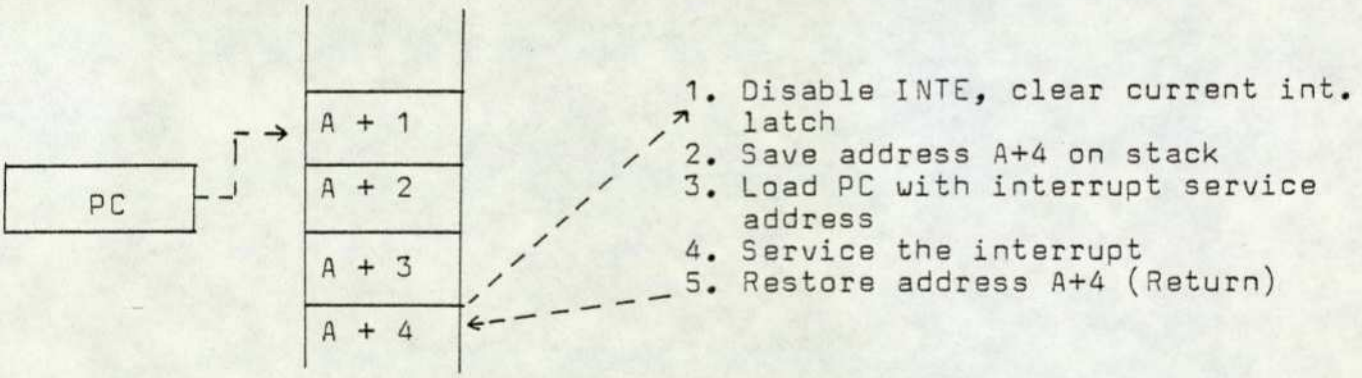


Interrupt 7 (internal timer interrupt) has the highest priority of all. For an interrupt to be processed by the computer, the interrupt enable flip-flop must be enabled (by executing the instruction 'Enable Interrupt'). An interrupt will be recognised on the high to low transition of the interrupt input which is not masked. All eight interrupt inputs have pulse catching latches at their inputs so that the interrupt requests can be latched to the processor and wait for their service. Each interrupt input is assigned a unique location in the RAM part of the memory where the service address for that interrupt input is found.

The interrupt programming model of the UMM-16 computer is shown below:



The tasks of the UMM-16 computer during an interrupt are shown below:



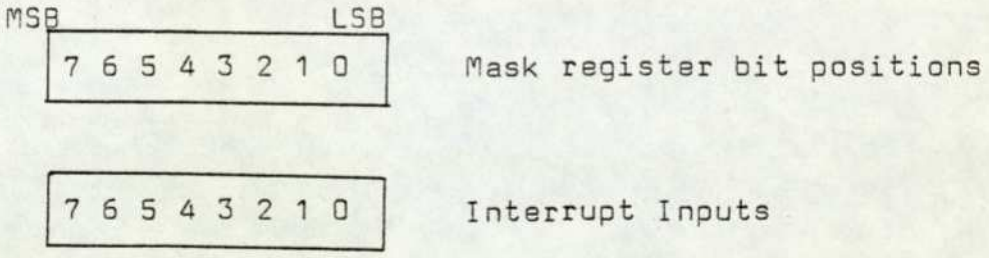
At address A+4 a non masked interrupt has been detected. Then the first task of the processor is to disable the interrupt enable flip-flop so that no more interrupts will be accepted until the current one is serviced. Also at this stage, the interrupt latch of the current interrupt is cleared (set to 1). Then the current value of the Program Counter register (A+4) is pushed onto stack and the program counter is loaded

with the interrupt service address. After servicing the interrupt, address A+4 is popped off the stack by executing the'RETI (Return from interrupt)' instruction. Thus the program returns to the suspended point. Note that the contents of any register is not saved by the processor. The user can include this function in the interrupt service routine if it is necessary (by using the instructions PUSH and POP).

If multiple interrupts occur at the same time then the processor will respond to the interrupt input which is not masked and which has the highest priority level.

3.1 The Mask register

The Mask register is eight bits wide and it can be loaded under software control. The purpose of the Mask register is to mask any interrupt input so that interrupts from that channel become non-active. A logic 1 in any bit porition of the Mask register masks the corresponding interrupt input:

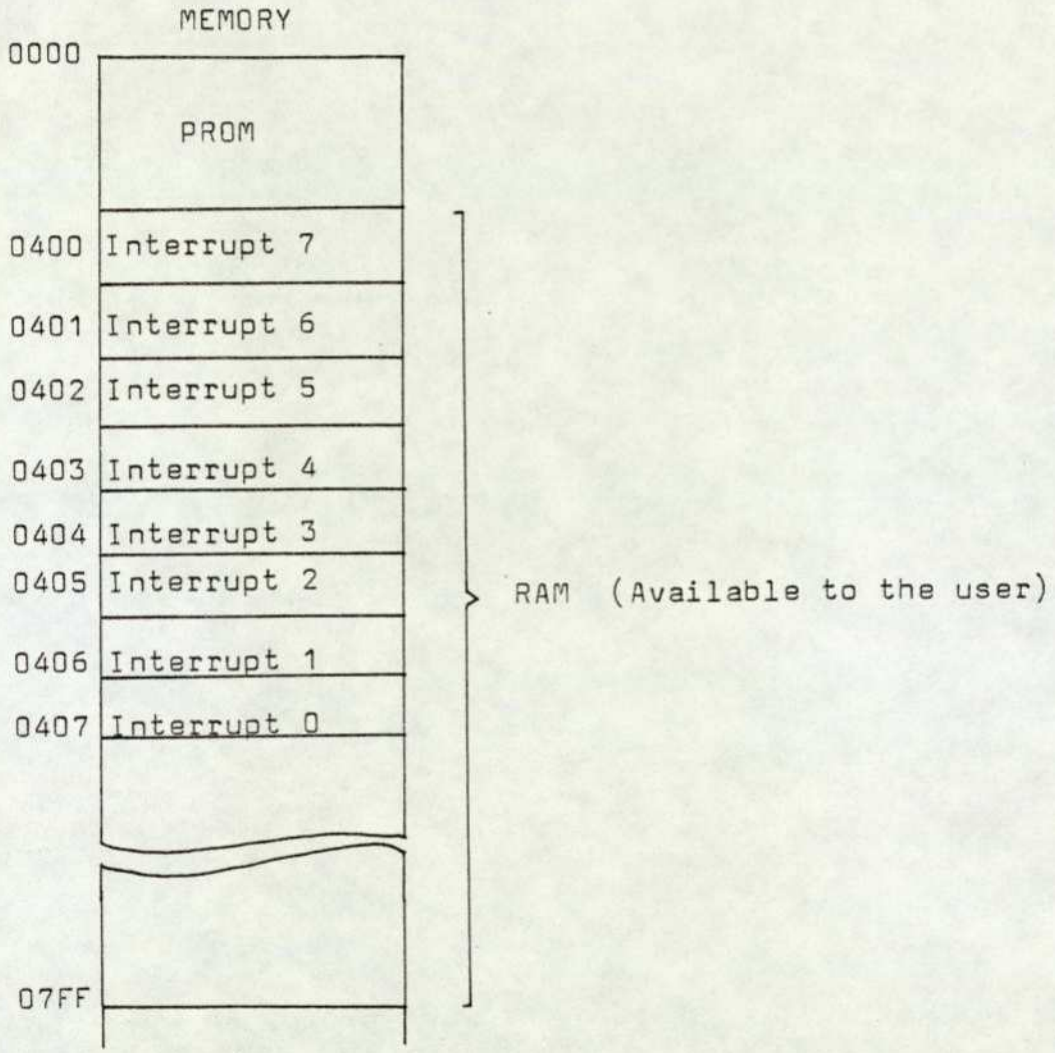


It is important that the interrupt enable flip-flop should be enabled after the Mask register is loaded with the required bit pattern.

When the 'reset' button is activated, the interrupt enable flip-flop is disabled automatically.

3.2 Interrupt Service Addresses

Each interrupt input has an unique location in the memory which can be loaded with the interrupt service address for that particular interrupt input as shown below:



e.g. If memory location 0400 is loaded with the 16 bit data 0700, then when an interrupt is accepted from interrupt 7, the interrupt service address will be 0700.

Example

Suppose that it is required to write a program to accept two external interrupts from interrupt inputs 0 and 1. The processor will clear the contents of scratchpad register 2 and will wait for an external interrupt. If interrupt 0 arrives then the processor will increment the contents of register 2 up to H'000A' and display the results on Port 0. If on the other hand interrupt 1 arrives, the processor will increment the contents of register 2 up to H'0020' and display the results on Port 0.

Assume that the program will be stored starting from memory location 0450 and that the service addresses for interrupt inputs 0 and 1 start from 0600 and 0700 respectively.

Initially, load the interrupt service addresses to the memory locations 0406 and 0407:

Memory Address	Interrupt Service Address	
0406	0700	interrupt 1
0407	0600	interrupt 0

The program for this example is as follows:

```
0450   3600          SM H'FC00'       ;  Load Mask register. Mask
0451   FC00                           ;  interrupts from inputs 2 to 7.
0452   1220   LOOP    CLRR 2           ;  Clear register 2
0453   3400          EI               ;  Enable interrupts
0454   2600          JMP LOOP         ;  Wait for an interrupt
0455   0452

0600   1530          LI  3,H'000A'    ;  Load loop count for interrupt 0
0601   000A
0602   1E20   MORE    OUTS 2,H'0000'   ;  Output contents of register 2
0603   0000                           ;  on Port 0
```

```
0604   0D20          INCR 2           ;  Increment contents of register 2
0605   2C30          DSZ 3, MORE      ;  If more, go to MORE
0606   0602
0607   3A00          RETI             ;  Return from interrupt


0700   1530          LI  3,H'0020'    ;  Load loop count for interrupt 1
0701   0020
0702   2600          JMP MORE         ;  go to MORE
0703   0602
```

If both interrupts occur at the same time then the processor will service interrupt 1 first (higher priority). Upon servicing interrupt 1, the processor will go to service interrupt 0 (lowest priority). The program can be loaded into the memory by the 'M' command and can be executed by the command 'E 450' (followed by carriage return).

3.3 The Timer Interrupt (Interrupt 7)

The Timer interrupt logic consists of a software programmable 16 bit counter register and a 16 bit binary down counter. The counter is loaded from the counter register and it decrements on each clock pulse when Mask 7 is enabled (LOW). If Mask 7 is disabled (HIGH) then the counter is in the 'Hold' mode. When the contents of the counter are zero it sends an interrupt pulse (high to low transition) to the interrupt input 7 latches (internal). If the interrupt enable flip-flop is enabled and the system is in the instruction fetch cycle then this interrupt will be accepted and serviced by the processor. When the contents of the counter are zero, it is re-loaded on the clock pulse from the counter register. The maximum number that can be loaded into the counter is 65535 (FFFF).

If the counter register is loaded with a constant 'K' and Mask 7 is enabled (LOW) then the timer interrupts will be generated after every 'KT'

seconds, where 'T' is the clock period of the system. There is no need to re-load the counter register after a timer interrupt is generated. This is done automatically by the processor.

3.4 Loading the Counter Register

The counter register is loaded with the instruction "load the Internal timer", LC, which has the opcode 37_H (see the instruction set of UMM-16).

Example

Assume that one wished to receive data from input Port 0 every 160 clock periods. The data received is required to be incremented by one and then sent to the output Port 0.

Assume that the program will be stored starting from memory location 0450 and that the interrupt service address starts from location 0600.

Initially, load the interrupt service adress for the timer interrupt to memory location 0400:

Memory Address	Timer interrupt service Address
0400	0600

The program for this example is as follows:

```
0450   3700          LC,H'00A0'      ;  Load the counter register
0451   00A0                          ;  with decimal 160 (00A0H)
0452   3600          SM,H'7F00'      ;  Enable Mask 7, disable others
0453   7F00
0454   3400   LOOP   EI              ;  Enable interrupts
0455   2600          JMP LOOP        ;  Wait for interrupts
0456   0454
```

```
0600  1F20        INS 2,H'0000'    ;  Input data to register 2 from
0601  0000                         ;  Port 0
0602  0D20        INCR 2           ;  Increment contents of register 2
0603  1E20        OUTS 2,H'0000'   ;  Output contents of
0604  0000                         ;  register 2
0605  3A00        RETI             ;  Return from interrupt
```

SECTION - 4

The Data Acquisition Channel

4. Programming for the Data Acquisition Channel

Port 1 of the UMM-16 computer is used as a data acquisition channel. The input lines of Port 1 are connected to a 12 bit Analog to Digital converter and a Sample/Hold amplifier. The user has the choice of whether or not to use the Sample/Hold amplifier depending upon the external connections he makes on the front panel. The output lines of Port 1 are connected to a 12 Digital to Analog Converter. (For more details on the Data Acquisition Channel, see UMM-16 Computer User's Manual).

The data acquisition channel operates either in the Bipolar mode (two's complement) or in the Unipolar mode (straight binary), defined as follows:

	Two's complement	Straight binary
+Full Scale	7FFX	FFFX
½Full Scale	400X	800X
1 LSB	001X	001X
zero	000X	000X
-1 LSB	FFFX	-
-½Full Scale	C00X	-
-Full Scale	800X	-

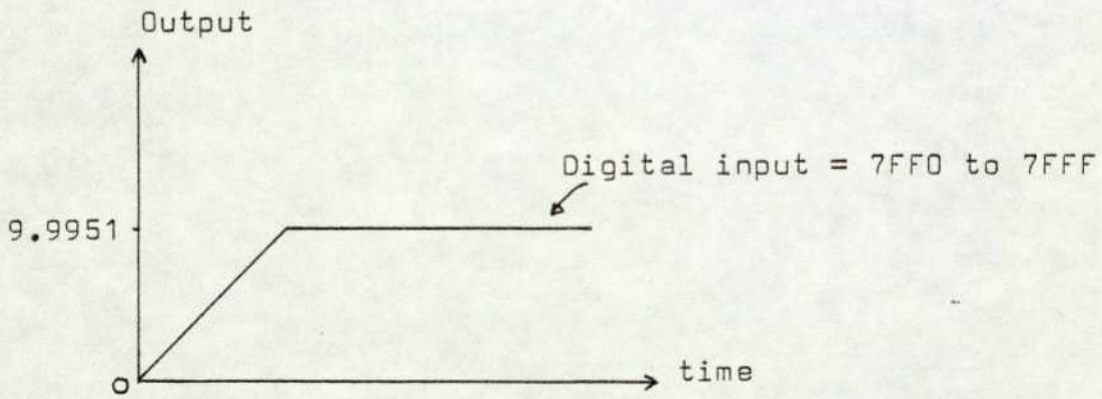
'X' = 4-bit don't care

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

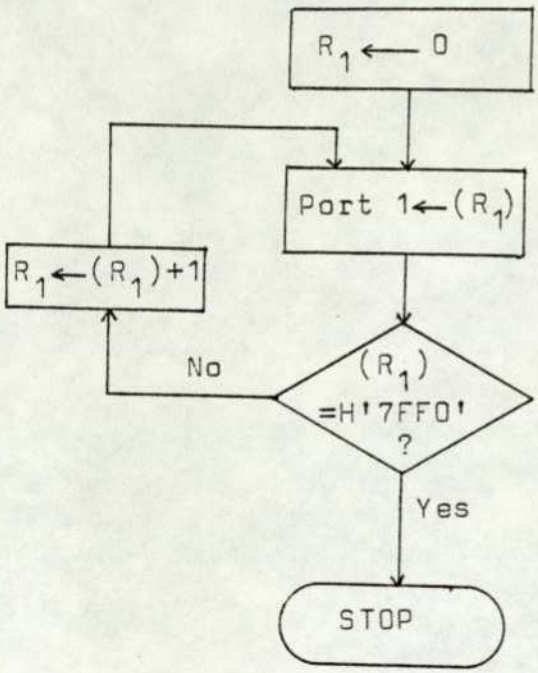
	Two's complement	Straight binary
+Full Scale	9.9951 V	9.9976 V
½Full Scale	5.0000 V	5.0000 V
1 LSB	4.88 mV	2.44 mV
-1 LSB	-4.88 mV	-
-½Full Scale	-5.0000 V	-
-Full Scale	-10.0000 V	-

Example 1

Write a program to generate the following waveform at the output of the Digital to Analog Converter. (Assume that the saturation time is not critical).



The flow diagram is shown below:

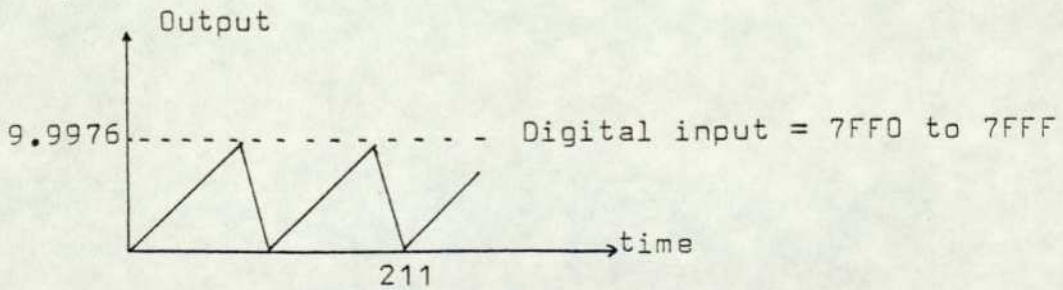


Assuming that the program will be stored starting from 0400,

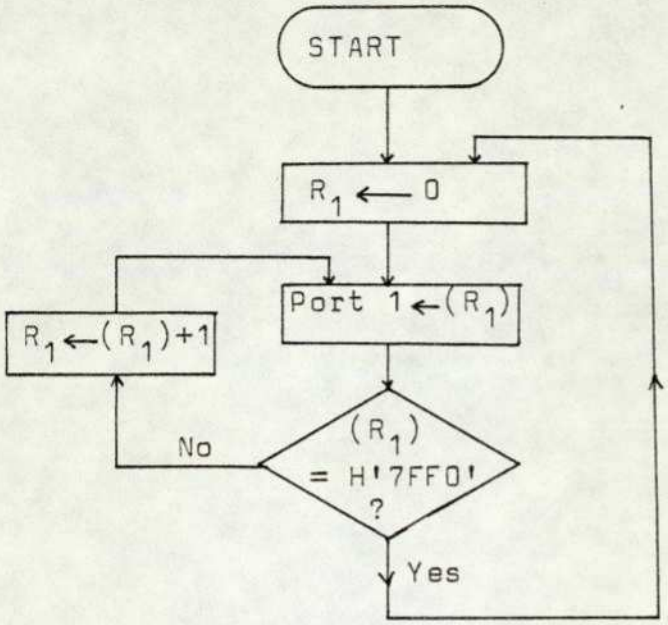
```
0400  1210        CLRR 1                ; clear register 1
0401  1E10  BACK  OUTS 1,H'0100'        ; output contents of register 1 to
0402  0100                              ; the DAC.
0403  2B10        JE 1,H'7FF0',OUT      ; go to OUT if register 1
0404  7FF0                              ; contains H'7FF0'
0405  0409
0406  0D10        INCR 1                ; Increment contents of register 1
0407  2600        JMP BACK
0408  0401
0409  2200  OUT   WAIT                  ; Stop
```

Example 2

Write a program to generate the following waveform at the output of the Digital to Analog Converter. (Assume that the frequency of the waveform is not critical).



The flow diagram for this example is shown below:

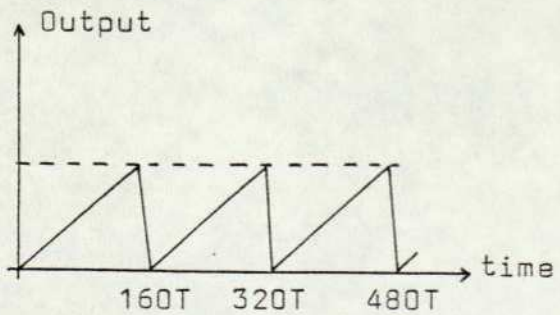


Assuming that the program will be stored starting from location 0400,

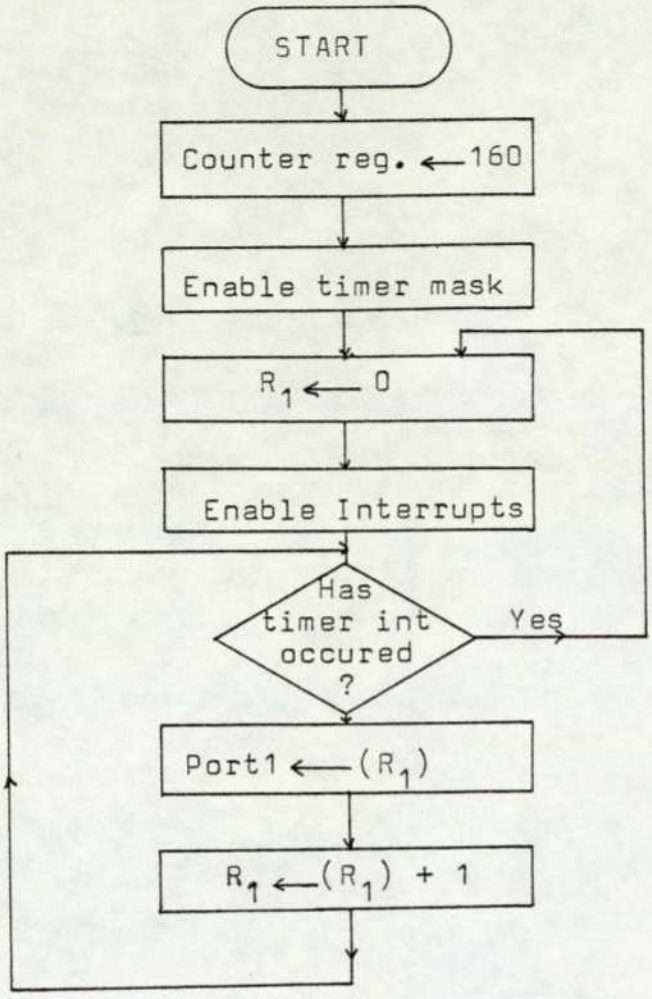
```
0400  1210  LOOP  CLRR 1                ; clear register 1.
0401  1E10  BACK  OUTS 1,H'0100'        ; output contents of register 1
0402  0100                              ; to the DAC.
0403  2B10        JE 1,H'7FF0',LOOP     ; go to LOOP for the
0404  7FF0                              ; next waveform.
0405  0400
0406  0D10        INCR 1                ; Increment contents of 1
0407  2600        JMP BACK
0408  0401
```

Example 3

Write a program to generate the following waveform at the output of the Digital to Analog Converter. Assume that the period of the waveform is 160T where, T = system clock period.



The flow diagram for this example is shown below:



Assuming that the program will be stored starting from location 0450 and the timer interrupt service address starts from location 0600,

Initially load the timer interrupt service address to location 0400,

Address	Timer Interrupt Service Address
0400	0600

Then the program is as follows:

```
0450  3700        LC,H'00A0'          ; Load counter with 160
0451  00A0
0452  3600        SM,H'7F00'          ; Enable Mask 7, disable others
0453  7F00
0454  1210        CLRR 1              ; Clear register 1
0455  3400        EI                  ; Enable interrupt
0456  1E10  BACK  OUTS 1,H'0100'      ; Output contents of register 1
0457  0100                            ; to the DAC.
0458  0D10        INCR 1              ; Increment contents of register 1
0459  2600        JMP BACK
045A  0456
```

```
0600  1210        CLRR 1              ; Clear register 1
0601  3400        EI                  ; Enable interrupt
0602  1E10        OUTS 1,H'0100'      ; Output contents of register 1
0603  0100                            ; to the DAC
0604  3A00        RETI                ; Return from interrupt
```

APPENDIX F

PROGRAM TO DESIGN FIR DIGITAL FILTERS WITH THE WINDOWING ALGORITHM

```
      MASTER WIND
C     PROGRAM TO DESIGN FINITE IMPULSE RESPONSE(FIR)
C     LINEAR PHASE DIGITAL FILTER WITH THE WINDOW
C     METHOD
C*************************
C PROGRAMMER: D. IBRAHIM, THE CITY UNIVERSITY, 1980
C*************************
C     TWO WINDOWS ARE CONSIDERED:THE HAMMING WINDOW
C     AND THE LANCZOS WINDOW
C
C     THE INPUT DATA CONSISTS OF 3 CARDS:
C CARD1: SPECIFIES THE FOLLOWING IN INTEGER FORMAT
C FILTER LENGTH(MAX. 150), PLOTTING OPTION(0=NO
C PLOTTING, 1=PLOT THE FREQUENCY RESPONSE), WINDOW TYPE
C (1=HAMMING WINDOW, 2=LANCZOS WINDOW)
C
C CARD2: SPECIFIES THE FOLLOWING IN INTEGER FORMAT
C FILTER TYPE(1=LOW PASS, 2=HIGH PASS, 3=BAND-PASS, 4=
C BAND-STOP), WINDOW PARAMETER(ONLY FOR LANCZOS WINDOW)
C
C CARD3: SPECIFIES THE FOLLOWING IN FREE FORMAT
C FIRST CUT-OFF FREQUENCY(LOW PASS AND HIGH PASS),
C SECOND CUT-OFF FREQUENCY(BAND-PASS AND BAND-STOP)
C..... CUT-OFF FREQUENCIES ARE NORMALIZED.....
C
C THE FOLLOWING INPUT DATA IS AN EXAMPLE FOR A LENGTH
C 24 HIGHPASS FILTER WITH LANCZOS WINDOWING WHERE THE
C CUT-OFF FREQUENCY IS 0.1, WINDOW PARAMETER IS 2 AND
C IT IS REQUIRED TO PLOT THE FREQUENCY RESPONSE OF THE
C FILTER
C
C 24 1 2
C 2 2
C 0.1
C
C
      DIMENSION H(150), F(150), FM(150)
      DOUBLE PRECISION PI, BEN, PER
      READ(5, 100) NFILT, JPLOT, NWIN
      IF(NWIN.EQ.2) READ(5, 100) NTYPE, NPARA
      IF(NWIN.EQ.1) READ(5, 100) NTYPE
      IF(NTYPE.GT.2) READ(5, 101) F1, F2
      IF(NTYPE.LE.2) READ(5, 101) F1
  101 FORMAT(2F0.0)
  100 FORMAT(3I0)
      NODD=NFILT-(NFILT/2)*2
      NEVE=NFILT/2
      L=NEVE+1
      PI=3.141592653589793
      CAL=0.5*FLOAT(1-NODD)
      BEN=PI/(FLOAT(NEVE)-CAL)
```

```
C
C CALCULATING THE IMPULSE RESPONSE ELEMENTS
C
      IF(NODD.EQ.0) GO TO 200
      IF(NTYPE.EQ.1) H(L)=2.*F1
      IF(NTYPE.EQ.2) H(L)=1.-2.*F1
      IF(NTYPE.EQ.3) H(L)=2.*(F2-F1)
      IF(NTYPE.EQ.4) H(L)=1.-2*(F2-F1)
  200 DO 210 J=1,NEVE
      K=L-J
      SEN=FLOAT(J)-CAL
      SF1=2.*PI*SEN*F1
      SF2=2.*PI*SEN*F2
      DENO=PI*SEN
      IF(NTYPE.EQ.1) H(K)=SIN(SF1)/DENO
      IF(NTYPE.EQ.2) H(K)=(SIN(DENO)-SIN(SF1))/DENO
      IF(NTYPE.EQ.3) H(K)=(SIN(SF2)-SIN(SF1))/DENO
      IF(NTYPE.EQ.4) H(K)=(SIN(DENO)-SIN(SF2)+SIN(SF1))/DENO
C
C WINDOWING THE COEFFICIENTS
C
      PER=BEN*SEN
      IF(NWIN.EQ.2) GO TO 220
      H(K)=H(K)*(0.54+0.46*(COS(PER)))
      GO TO 210
  220 H(K)=H(K)*(ABS(SIN(PER)/PER))**NPARA
  210 CONTINUE
      DO 230 J=1,NEVE
      K=NFILT+1-J
  230 H(K)=H(J)
C
C PROGRAM OUTPUT SECTION
C
      WRITE(6,300)
  300 FORMAT(1H1, 70(1H*)//25X,'FINITE IMPULSE RESPONSE(FIR)'/
     125X,'LINEAR PHASE DIGITAL FILTER DESIGN'/
     225X,'WINDOWING ALGORITHM'/)
      IF(NTYPE.EQ.1) WRITE(6,700)
      IF(NTYPE.EQ.2) WRITE(6,701)
      IF(NTYPE.EQ.3) WRITE(6,702)
      IF(NTYPE.EQ.4) WRITE(6,703)
  700 FORMAT(25X,'LOW PASS FILTER'/)
  701 FORMAT(25X,'HIGH PASS FILTER'/)
  702 FORMAT(25X,'BAND-PASS FILTER'/)
  703 FORMAT(25X,'BAND-STOP FILTER'/)
      IF(NWIN.EQ.1) WRITE(6,310)
  310 FORMAT(25X,'HAMMING WINDOWED DESIGN'/)
      IF(NWIN.EQ.2) WRITE(6,320)
  320 FORMAT(25X,'LANCZOS WINDOWED DESIGN'/)
      WRITE(6,330) NFILT
  330 FORMAT(15X,'FILTER LENGTH = ',I3/)
      IF(NWIN.EQ.2) WRITE(6,340) NPARA
  340 FORMAT(15X,'LANCZOS PAPAMETER = ',I2/)
```

```
      WRITE(6,350)
  350 FORMAT(15X,'*****IMPULSE RESPONSE*****')
      IF(NODD.EQ.1) NEVE=NEVE+1
      DO 360 J=1,NEVE
      K=NFILT+1-J
      WRITE(6,370) J,H(J),K
  360 CONTINUE
  370 FORMAT(20X,'H(',I3,') =',E15.8,' =H(',I4,')')
      WRITE(6,380)
  380 FORMAT(/1X,70(1H*)/1H1)
      IF(JPLOT.EQ.0) GO TO 461
      KLM=1
      FREQ=0.
      K=1
  600 HNOT=0.
      SUM=0.
      DO 468 J=1,NFILT
      RES=COS(2*PI*(J-1)*FREQ)
      RES=RES*H(J)
      HNOT=HNOT+RES
      DAN=SIN(2*PI*(J-1)*FREQ)
      DAN=DAN*H(J)
      SUM=SUM+DAN
  468 CONTINUE
      HNOT=HNOT*HNOT
      SUM=SUM*SUM
      SMAG=(SUM+HNOT)**0.5
      FM(K)=20*(ALOG10(SMAG))
      IF(ABS(FM(K)).GT.60.) KLM=0
      F(K)=FREQ
      K=K+1
      FREQ=FREQ+0.004
      IF(FREQ.LT.0.5004) GO TO 600
      CALL M19346
      CALL DEVPAP(350.,350.,0)
      CALL AXIPOS(0,30.,30.,150.,1)
      CALL AXIPOS(0,30.,30.,150.,2)
      CALL AXISCA(1,5,0.,0.5,1)
      CALL AXISCA(1,10,-100.,0.,2)
      IF(KLM.EQ.1) CALL AXISCA(1,6,-60.,0.,2)
      CALL AXIDRA(1,1,1)
      CALL AXIDRA(-1,-1,2)
      CALL MOVTO2(120.,10.)
      CALL CHAHOL(11HFREQUENCY*.)
      CALL MOVTO2(20.,10.)
      CALL CHAHOL(16HFILTER LENGTH=*.)
      CALL CHAINT(NFILT,-3)
      CALL MOVTO2(20.,197.)
      CALL CHAHOL(19HLOG.MAGNITUDE(DB)*.)
      CALL WINDO2(0.,200.,30.,350.)
      CALL GRAPOL(F,FM,126)
      CALL MOVTO2(85.,230.)
      CALL CHAHOL(17HWINDOWED DESIGN*.)
      CALL DEVEND
  461 STOP
      END
      FINISH
```

APPENDIX G

PROGRAM TO DESIGN OPTIMAL FIR DIGITAL FILTERS

```
      LIBRARY(SUBGROUPGRAF)
      LIBRARY(SUBGROUPSURF)
      LIBRARY(SUBGROUPGINO)
      LIST(LP)
      PROGRAM(PROGRJEE504)
      INPUT 1,5= CR0
      OUTPUT 2,6 = LP0
      TRACE 2,100
      COMPRESS INTEGER AND LOGICAL
      END
         MASTER FILT
C PROGRAM TO DESIGN FINITE IMPULSE RESPONSE(FIR)
C LINEAR PHASE OPTIMUM DIGITAL FILTER WITH THE
C REMEZ EXCHANGE ALGORITHM
C
C THE INPUT DATA CONSISTS OF 4 CARDS:-
C
C CARD 1 SPECIFIES THE FOLLOWING IN INTEGER FORMAT:
C FILTER LENGTH(MAX.150),FILTER TYPE(1=MULTIPLE
C PASSBAND/STOPBAND,2=DIFFERENTIATOR,3=HILBERT
C TRANSFORM FILTER),NUMBER OF BANDS(MAX.10),GRID
C DENSITY,COEFFICIENT ACCURACY(BITS),PLOTTING OPTION
C (0=NO PLOTTING,1=PLOT THE ACTUAL FREQUENCY RESPONSE,
C 2=PLOT THE FREQUENCY RESPONSE WITH THE COEFFICIENTS
C ROUNDED)
C
C CARD 2 SPECIFIES THE FOLLOWING IN FREE FORMAT:
C BANDEDGES(LOWER AND UPPER EDGES FOR EACH BAND)
C ......UNITY SAMPLING FREQUENCY ASSUMED.......
C
C CARD 3 SPECIFIES THE FOLLOWING IN FREE FORMAT:
C DESIRED MAGNITUDE(OR DESIRED SLOPE IF A DIFFERENTIATOR)
C FOR EACH BAND
C
C CARD 4 SPECIFIES THE FOLLOWING IN FREE FORMAT:
C WEIGHT FUNCTION IN EACH BAND
C
C
C EXAMPLE 1:
C THE FOLLOWING INPUT DATA SPECIFIES A LENGTH 24 HIGHPASS
C FILTER WITH PASSBAND FROM 0 TO 0.1 AND STOP BAND FROM
C 0.2 TO 0.5 WITH WEIGHING OF 1 IN BOTH BANDS.THE
C COEFFICIENT ACCURACY IS 16 BITS AND IT IS REQUIRED TO
C PLOT THE FREQUENCY RESPONSE WITH THE COEFFICIENTS
C ROUNDED.  THE GRID DENSITY IS TO BE 32
C
C 24 1 2 32 16 2
C 0.0 0.1 0.2 0.5
C 0.0 1.0
C 1.0 1.0
C
C EXAMPLE 2:
C THE FOLLOWING INPUT DATA SPECIFIES A LENGTH 15 BANDPASS
C FILTER WITH STOPBANDS FROM 0 TO 0.1 AND 0.425 TO 0.5 AND
C PASSBAND FROM 0.2 TO 0.35 WITH WEIGHING OF 10 IN THE
C STOPBANDS AND 1 IN THE PASSBAND.  THE COEFFICIENT ACCURACY
C IS 8 BITS AND IT IS REQUIRED TO PLOT THE FREQUENCY RESPONSE
C WITHOUT ROUNDING THE COEFFICIENTS.  THE GRID DENSITY IS TO
C BE 16
C
```

```
C 15 1 3 16 8 1
C 0.0 0.1 0.2 0.35 0.425 0.5
C 0.0 1.0 0.0
C 10.0 1.0 10.0
C
C
C
      DIMENSION IEXT(77),AD(77),ALPHA(77),X(77),Y(77),H(77)
      DIMENSION DES(1232),GRID(1232),WT(1232)
      DIMENSION EDGE(20),FX(10),WTX(10),DEVIAT(10),DBDEVI(10)
      DOUBLE PRECISION PI2,PI
      DOUBLE PRECISION AD,DEV,X,Y
      COMMON PI2,AD,DEV,X,Y,GRID,DES,WT,ALPHA,IEXT,NFCNS,NGRID
      PI2=6.283185307179586
      PI=3.141592653589793
C      THE PROGRAM IS SET UP FOR A MAXIMUM LENGTH OF 150,BUT
C      THIS UPPER LIMIT CAN BE CHANGED BY REDIMENSIONING THE
C       ARRAYS IEXT,AD,ALPHA,X,Y,H  TO BE NFMAX/2+2.
C      THE ARRAYS DES ,GRID, AND WT MUST BE DIMENSIONED
C      16(NFMAX/2+2)
      NFMAX = 150
      JTYPE=0
C        PROGRAM INPUT SECTION
      READ(5,120) NFILT,JTYPE,NBANDS,LGRID,JCOEF,JPLOT
  120  FORMAT(6I0)
      IF(NFILT.GT.NFMAX.OR.NFILT.LT.3) CALL ERROR
      IF(NBANDS.LE.0) NBANDS=1
C       GRID DENSITY IS ASSUMED TO BE 16 UNLESS SPECIFIED
C         OTHERWISE
      IF(LGRID.LE.0) LGRID=16
      JB=2*NBANDS
        READ(5,60) (EDGE(J),J=1,JB)
   60 FORMAT (50F0.0)
      READ(5,60) (FX(J),J=1,NBANDS)
       READ(5,60) (WTX(J),J=1,NBANDS)
      IF(JTYPE.EQ.0) CALL ERROR
      NEG=1
      IF(JTYPE.EQ.1) NEG=0
      ILOOP=1
      NODD=NFILT/2
      NODD=NFILT-2*NODD
      NFCNS=NFILT/2
      IF(NODD.EQ.1.AND.NEG.EQ.0) NFCNS=NFCNS+1
C       SET UP THE DENSE GRID.  THE NUMBER OF POINTS IN THE GRID
C         IS (FILTER LENGTH+1)*GRID DENSITY /2
      GRID(1)=EDGE(1)
      DELF=LGRID*NFCNS
      DELF=0.5/DELF
      IF(NEG.EQ.0) GO TO 135
      IF(EDGE(1).LT.DELF) GRID(1)=DELF
  135  CONTINUE
      J=1
      L=1
       LBAND=1
  140  FUP=EDGE(L+1)
  145 TEMP=GRID(J)
C          CALCULATE THE DESIRED MAGNITUDE RESPONSE AND THE WEIGHT
C        FUNCTION ON THE GRID
```

```
      DES(J)=EFF(TEMP,FX,WTX,LBAND,JTYPE)
      WT(J)=WATE(TEMP,FX,WTX,LBAND,JTYPE)
      J=J+1
      GRID(J)=TEMP+DELF
      IF(GRID(J).GT.FUP) GO TO 150
      GO TO 145
  150  GRID(J-1)=FUP
       DES(J-1)=EFF(FUP,FX,WTX,LBAND,JTYPE)
      WT(J-1)=WATE(FUP,FX,WTX,LBAND,JTYPE)
      LBAND=LBAND+1
      L=L+2
      IF(LBAND.GT.NBANDS) GO TO 160
       GRID(J)=EDGE(L)
      GO TO 140
  160  NGRID=J-1
      IF(NEG.NE.NODD) GO TO 165
       IF(GRID(NGRID).GT.(0.5-DELF)) NGRID=NGRID-1
  165  CONTINUE
C      SET UP A NEW APPROXIMATION PROBLEM WHICH IS EQUIVALENT
C      TO THE ORIGINAL PROBLEM
      IF(NEG) 170,170,180
  170  IF(NODD.EQ.1) GO TO 200
      DO 175 J=1,NGRID
      CHANGE=DCOS(PI*GRID(J))
      DES(J)=DES(J)/CHANGE
  175  WT(J)=WT(J)*CHANGE
      GO TO 200
  180  IF(NODD.EQ.1) GO TO 190
      DO 185 J=1,NGRID
       CHANGE=DSIN(PI*GRID(J))
      DES(J)=DES(J)/CHANGE
  185  WT(J)=WT(J)*CHANGE
      GO TO 200
  190  DO 195 J=1,NGRID
       CHANGE=DSIN(PI2*GRID(J))
      DES(J)=DES(J)/CHANGE
  195  WT(J)=WT(J)*CHANGE
C       INITIAL GUESS FOR THE EXTREMAL FREQUENCIES EQUALLY
C       SPACED ALONG THE GRID
  200  TEMP=FLOAT(NGRID-1)/FLOAT(NFCNS)
      DO 210 J=1,NFCNS
  210  IEXT(J)=(J-1)*TEMP+1
      IEXT(NFCNS+1)=NGRID
      NM1=NFCNS-1
       NZ=NFCNS+1
C       CALL THE REMEZ EXCHANGE ALGORITHM TO DO THE APPROXIMATOIN
C       PROBLEM
        CALL REMEZ(EDGE,NBANDS)
C      CALCULATE THE IMPULSE RESPONSE.
      IF(NEG) 300,300,320
  300  IF(NODD.EQ.0) GO TO 310
      DO 305 J=1,NM1
  305  H(J)=0.5*ALPHA(NZ-J)
      H(NFCNS)=ALPHA(1)
      GO TO 350
  310  H(1)=0.25*ALPHA(NFCNS)
      DO 315 J=2,NM1
  315  H(J)=0.25*(ALPHA(NZ-J)+ALPHA(NFCNS+2-J))
      H(NFCNS)=0.5*ALPHA(1)+0.25*ALPHA(2)
      GO TO 350
```

```
      320  IF(NODD.EQ.0) GO TO 330
           H(1)=0.25*ALPHA(NFCNS)
           H(2)=0.25*ALPHA(NM1)
           DO 325 J=3,NM1
      325  H(J)=0.25*(ALPHA(NZ-J)-ALPHA(NFCNS+3-J))
           H(NFCNS)=0.5*ALPHA(1)-0.25*ALPHA(3)
           H(NZ)=0.0
           GO TO 350
      330  H(1)=0.25*ALPHA(NFCNS)
           DO 335 J=2,NM1
      335  H(J)=0.25*(ALPHA(NZ-J)-ALPHA(NFCNS+2-J))
           H(NFCNS)=0.5*ALPHA(1)-0.25*ALPHA(2)
C           PROGRAM OUTPUT SECTION.
      350  WRITE(6,360)
      360  FORMAT(1H1, 70(1H*)//25X,'FINITE IMPULSE RESPONSE (FIR)'/
          125X,'LINEAR PHASE DIGITAL FILTER DESIGN'/
          225X,'REMEZ EXCHANGE ALGORITHM'/)
           IF(JTYPE.EQ.1) WRITE(6,365)
      365  FORMAT(25X,'BANDPASS FILTER'/)
           IF(JTYPE.EQ.2) WRITE(6,370)
      370  FORMAT(25X,'DIFFERTIATOR'/)
           IF(JTYPE.EQ.3) WRITE(6,375)
      375   FORMAT(25X,'HILBERT TRANSFORMER'/)
           WRITE(6,378)NFILT
      378  FORMAT(15X,'FILTER LENGTH =',I3/)
           WRITE(6,380)
      380  FORMAT(15X,'*****IMPULSE RESPONSE****')
      703   CONTINUE
           DO 381 J=1,NFCNS
           K=NFILT+1-J
           IF(NEG.EQ.0) WRITE(6,382)J,H(J),K
           IF(NEG.EQ.1) WRITE(6,383)J,H(J),K
      381  CONTINUE
      382  FORMAT(20X,'H(',I3,') =',E15.8,' =H(',I4,')')
      383  FORMAT(20X,'H(',I3,') =',E15.8,' =-H(',I4,')')
           IF(NEG.EQ.1.AND.NODD.EQ.1) WRITE(6,384)NZ
      384  FORMAT(20X,'H(',I3,') =0.0')
           IF(ILOOP.EQ.0) GO TO 704
           DO 450 K=1,NBANDS,4
           KUP=K+3
           IF(KUP.GT.NBANDS) KUP=NBANDS
           WRITE(6,385)(J,J=K,KUP)
      385  FORMAT(/24X,4('BAND',I3,8X))
           WRITE(6,390)(EDGE(2*J-1),J=K,KUP)
      390  FORMAT(2X,'LOWER BAND EDGE',5F15.9)
           WRITE(6,395)(EDGE(2*J),J=K,KUP)
      395  FORMAT(2X,'UPPER BAND EDGE',5F15.9)
           IF(JTYPE.NE.2) WRITE(6,400)(FX(J),J=K,KUP)
      400  FORMAT(2X,'DESIRED VALUE',2X,5F15.9)
           IF(JTYPE.EQ.2) WRITE(6,405)(FX(J),J=K,KUP)
      405  FORMAT(2X,'DESIRED SLOPE',2X,5F15.9)
           WRITE(6,410)(WTX(J),J=K,KUP)
      410  FORMAT(2X,'WEIGHTING',6X,5F15.9)
           DO 420 J=K,KUP
      420  DEVIAT(J)=DEV/WTX(J)
           WRITE(6,425)(DEVIAT(J),J=K,KUP)
      425  FORMAT(2X,'DEVIATION',6X,5F15.9)
           IF(JTYPE.NE.1) GO TO 450
           DO 430 J=K,KUP
```

```
    DBDEVI   (J)=20.0*ALOG10(DEVIAT(J))
430 IF(FX(J).EQ.1.)DBDEVI(J)=20.*ALOG10(ABS((1.+DEVIAT(J))/
   1(1.-DEVIAT(J))))
    WRITE(6,435)(DBDEVI   (J),J=K,KUP)
435  FORMAT(2X,'DEVIATION IN DB',5F15.9)
450  CONTINUE
    WRITE(6,455)(GRID(IEXT(J)),J=1,NZ)
455  FORMAT(/2X,'EXTREMAL FREQUENCIES'/(2X,5F12.7))
    WRITE(6,460)
460  FORMAT(/1X,70(1H*)/1H1)
    KLM=1
    DO 701 J=1,NFCNS
    KI=2**(JCOEF-1)
    ALPHA(J)=H(J)*KI
    JS=INT(ALPHA(J))
    IF((ALPHA(J)-JS).LE.0.5) GO TO 700
    JS=JS+1
700 ALPHA(J)=FLOAT(JS)/FLOAT(KI)
701 CONTINUE
    WRITE(6,900) JCOEF
900 FORMAT(/25X,'COEFFICIENT ACCURACY =',I2,' BITS'/)
    WRITE(6,702)
702 FORMAT(/15X,'****IMPULSE RESPONSE(ROUNDED)****')
    ILOOP=0
    DO 800 J=1,NFCNS
    WT(J)=H(J)
    H(J)=ALPHA(J)
800 CONTINUE
    GO TO 703
704 DO 801 J=1,NFCNS
    H(J)=WT(J)
801 CONTINUE
    WRITE(6,705)
705 FORMAT(/1X,70(1H*)/1H1)
    IF(JPLOT.EQ.0) GO TO 461
    IF(JPLOT.EQ.1) GO TO 706
    DO 707 J=1,NFCNS
707 H(J)=ALPHA(J)
706 CONTINUE
    IF(JTYPE.NE.1) GO TO 462
    DO 463 J=1,NFCNS
463 GRID(J)=H(J)
    K=NFCNS
    IF(NODD.EQ.1) K=K-1
    KK=NFCNS+1
    DO 464 J=KK,NFILT
    GRID(J)=H(K)
464 K=K-1
    GO TO 465
462 DO 466 J=1,NFCNS
466 GRID(J)=-H(J)
    K=NFCNS
    IF(NODD.EQ.1) K=K-1
    KK=NFCNS+1
    DO 467 J=KK,NFILT
    GRID(J)=-H(K)
467     K=K-1
465 FREQ=0.0
```

```
      K=NFILT+1
  600 HNOT=0.0
      SUM=0.0
      DO 468 J=1,NFILT
      RES=COS(PI2*(J-1)*FREQ)
      RES=RES*GRID(J)
      HNOT=HNOT+RES
      DAN=SIN(PI2*(J-1)*FREQ)
      DAN=DAN*GRID(J)
      SUM=SUM+DAN
  468 CONTINUE
      HNOT=HNOT*HNOT
      SUM=SUM*SUM
      GRID(K)=(SUM+HNOT)**0.5
      DES(K)=20*(ALOG10(GRID(K)))
      WT(K)=FREQ
      K=K+1
      FREQ=FREQ+0.004
      IF(FREQ.LT.0.5004) GO TO 600
      K=NFILT+1
      DO 605 J=1,126
      GRID(J)=GRID(K)
      DES(J)=DES(K)
      IF(ABS(DES(J)).GT.60.) KLM=0
      WT(J)=WT(K)
  605 K=K+1
      CALL M19346
      CALL DEVPAP(350.,350.,0)
      CALL AXIPOS(0,30.,30.,150.,1)
      CALL AXIPOS(0,30.,30.,150.,2)
      IF(JTYPE.NE.1) GO TO 469
      CALL AXISCA(1,5,0.,0.5,1)
      CALL AXISCA(1,10,-100.,0.,2)
      IF(KLM.EQ.1) CALL AXISCA(1,6,-60.,0.,2)
  501 CALL AXIDRA(1,1,1)
      CALL AXIDRA(-1,-1,2)
      CALL MOVTO2(20.,10.)
      CALL CHAHOL(16HFILTER LENGTH=*.)
      CALL CHAINT(NFILT,-3)
      CALL MOVTO2(120.,10.)
      CALL CHAHOL(11HFREQUENCY*.)
      CALL MOVTO2(20.,197.)
      IF(JTYPE.NE.1) GO TO 502
      CALL CHAHOL(19HLOG MAGNITUDE(DB)*.)
      GO TO 601
  469 CALL AXISCA(1,5,0.,0.5,1)
      CALL AXISCA(1,10,0.,1.,2)
      CALL WINDO2(0.,200.,0.,350.)
      GO TO 501
  502 CALL CHAHOL(11HMAGNITUDE*.)
      CALL GRAPOL(WT,GRID,126)
      GO TO 606
  601  CALL WINDO2(0.,200.,30.,350.)
      CALL GRAPOL(WT,DES,126)
  606 IF(JPLOT.EQ.2) CALL MOVTO2(80.,230.)
      IF(JPLOT.EQ.2) CALL CHAHOL(22HCOEFFICIENTS ROUNDED*.)
      CALL DEVEND
  461 STOP
      END
```

```
      FUNCTION EFF(TEMP,FX,WTX,LBAND,JTYPE)
C       FUNCTION TO CALCULATE THE DESIRED MAGNITUDE RESPONSE
C         AS A FUNCTION OF FREQUENCY
      DIMENSION FX(5),WTX(5)
      IF(JTYPE.EQ.2) GO TO 1
       EFF=FX(LBAND)
      RETURN
    1  EFF=FX(LBAND)*TEMP
       RETURN
       END
      FUNCTION WATE(TEMP,FX,WTX,LBAND,JTYPE)
C        FUNCTION TO CALCULATE THE WEIGHT FUNCTION AS A FUNCTION
C     OF FREQUENCY
      DIMENSION FX(5),WTX(5)
      IF(JTYPE.EQ.2) GO TO 1
      WATE=WTX(LBAND)
      RETURN
 1    IF(FX(LBAND).LT.0.0001) GO TO 2
      WATE=WTX(LBAND)/TEMP
      RETURN
 2    WATE=WTX(LBAND)
      RETURN
      END
       SUBROUTINE ERROR
       WRITE(6,1)
    1 FORMAT('****IS INPUT DATA O.K. ****')
       RETURN
       END
      SUBROUTINE REMEZ(EDGE,NBANDS)
C       THIS SUBROUTINE IMPLEMENTED THE REMEZ EXCHANGE ALGORITM
C      FOR THE WEIGHTED CHEBYSHEV APPROXIMATION OF A CONTINUOS
C        FUNCTION WITH A SUM OF COSINES.
      DIMENSION EDGE(20)
      DIMENSION IEXT(77),AD(77),ALPHA(77),X(77),Y(77)
      DIMENSION DES(1232),GRID(1232),WT(1232)
      DIMENSION A(77),P(77),Q(77)
       DOUBLE PRECISION PI2,DNUM,DDEN,DTEMP,A,P,Q
      DOUBLE PRECISION AD,DEV,X,Y,GEE,D
      COMMON PI2,AD,DEV,X,Y,GRID,DES,WT,ALPHA,IEXT,NFCNS,NGRID
C       THE PROGRAM ALLOWS A MAXIMUM NUMBER OF ITERATIONS OF 25
      ITRMAX=25
      DEVL=-1.0
       NZ=NFCNS+1
      NZZ=NFCNS+2
       NITER=0
  100  CONTINUE
      IEXT(NZZ)=NGRID+1
      NITER=NITER+1
      IF(NITER.GT.ITRMAX) GO TO 400
       DO 110 J=1,NZ
      II=IEXT(J)
      DTEMP=GRID(II)
      DTEMP=DCOS(DTEMP*PI2)
  110  X(J)=DTEMP
      JET=(NFCNS-1)/15+1
      DO 120 J=1,NZ
  120   AD(J)=D(J,NZ,JET)
      DNUM=0.0
       DDEN=0.0
      K=1
```

```
      DO 130 J=1,NZ
        L=IEXT(J)
      DTEMP=AD(J)*DES(L)
       DNUM=DNUM+DTEMP
       DTEMP=K*AD(J)/WT(L)
       DDEN=DDEN+DTEMP
  130  K=-K
      DEV=DNUM/DDEN
      NU=1
      IF(DEV.GT.0.0) NU=-1
      DEV=-NU*DEV
       K=NU
      DO 140 J=1,NZ
       L=IEXT(J)
       DTEMP=K*DEV/WT(L)
       Y(J)=DES(L)+DTEMP
  140  K=-K
       IF(DEV.GE.DEVL) GO TO 150
        CALL OUCH
      GO TO 400
  150 DEVL=DEV
      JCHNGE=0
      K1=IEXT(1)
      KNZ=IEXT(NZ)
      KLOW=0
      NUT=-NU
      J=1
C       SEARCH FOR THE EXTREMAL FREQUENCIES OF THE BEST
  200  IF(J.EQ.NZZ) YNZ=COMP
C       APPROXIMATION.
      IF(J.GE.NZZ) GO TO 300
      KUP=IEXT(J+1)
       L=IEXT(J)+1
      NUT=-NUT
      IF(J.EQ.2) Y1=COMP
      COMP=DEV
      IF(L.GE.KUP) GO TO 220
      ERR=GEE(L,NZ)
      ERR=(ERR-DES(L))*WT(L)
      DTEMP=NUT*ERR-COMP
      IF(DTEMP.LE.0.0) GO TO 220
       COMP=NUT*ERR
  210  L=L+1
      IF(L.GE.KUP) GO TO 215
      ERR=GEE(L,NZ)
      ERR=(ERR-DES(L))*WT(L)
      DTEMP=NUT*ERR-COMP
      IF(DTEMP.LE.0.0) GO TO 215
       COMP=NUT*ERR
      GO TO 210
  215  IEXT(J)=L-1
      J=J+1
      KLOW=L-1
       JCHNGE=JCHNGE+1
      GO TO 200
  220  L=L-1
  225 L=L-1
       IF(L.LE.KLOW) GO TO 250
      ERR=GEE(L,NZ)
      ERR=(ERR-DES(L))*WT(L)
```

```
      DTEMP=NUT*ERR-COMP
      IF(DTEMP.GT.0.0) GO TO 230
      IF(JCHNGE.LE.0) GO TO 225
      GO TO 260
230    COMP=NUT*ERR
235    L=L-1
       IF(L.LE.KLOW) GO TO 240
      ERR=GEE(L,NZ)
      ERR=(ERR-DES(L))*WT(L)
       DTEMP=NUT*ERR-COMP
      IF(DTEMP.LE.0.0) GO TO 240
      COMP=NUT*ERR
      GO TO 235
240    KLOW=IEXT(J)
      IEXT(J)=L+1
      J=J+1
       JCHNGE=JCHNGE+1
      GO TO 200
250    L=IEXT(J)+1
      IF(JCHNGE.GT.0) GO TO 215
255    L=L+1
      IF(L.GE.KUP) GO TO 260
       ERR=GEE(L,NZ)
       ERR=(ERR-DES(L))*WT(L)
      DTEMP=NUT*ERR-COMP
      IF(DTEMP.LE.0.0) GO TO 255
       COMP=NUT*ERR
      GO TO 210
260    KLOW=IEXT(J)
      J=J+1
      GO TO 200
300    IF(J.GT.NZZ) GO TO 320
      IF(K1.GT.IEXT(1)) K1=IEXT(1)
      IF(KNZ.LT.IEXT(NZ)) KNZ=IEXT(NZ)
      NUT1=NUT
       NUT=-NU
      L=0
       KUP=K1
      COMP=YNZ*(1.00001)
       LUCK=1
310    L=L+1
      IF(L.GE.KUP) GO TO 315
      ERR=GEE(L,NZ)
      ERR=(ERR-DES(L))*WT(L)
      DTEMP=NUT*ERR-COMP
      IF(DTEMP.LE.0.0) GO TO 310
      COMP=NUT*ERR
      J=NZZ
      GO TO 210
315    LUCK=6
      GO TO 325
320    IF(LUCK.GT.9) GO TO 350
      IF(COMP.GT.Y1) Y1=COMP
      K1=IEXT(NZZ)
325    L=NGRID+1
      KLOW=KNZ
      NUT=-NUT1
       COMP=Y1*(1.00001)
330    L=L-1
      IF(L.LE.KLOW) GO TO 340
       ERR=GEE(L,NZ)
       ERR=(ERR-DES(L))*WT(L)
```

```
      DTEMP=NUT*ERR-COMP
      IF(DTEMP.LE.0.0) GO TO 330
      J=NZZ
      COMP=NUT*ERR
      LUCK=LUCK+10
      GO TO 235
  340  IF(LUCK.EQ.6) GO TO 370
      DO 345 J=1,NFCNS
  345  IEXT(NZZ-J)=IEXT(NZ-J)
      IEXT(1)=K1
      GO TO 100
  350  KN=IEXT(NZZ)
      DO 360 J=1,NFCNS
  360  IEXT(J)=IEXT(J+1)
      IEXT(NZ)=KN
      GO TO 100
  370  IF(JCHNGE.GT.0) GO TO 100
C       CALCULATIONS OF THEBEST APPROXIMATION
C         USING THE INVERSE DISCRETE FOURIER TRANSFORM.
  400  CONTINUE
      NM1=NFCNS-1
      FSH=1.0E-06
       GTEMP=GRID(1)
      X(NZZ)=-2.0
      CN=2*NFCNS-1
      DELF=1.0/CN
      L=1
      KKK=0
       IF(EDGE(1).EQ.0.0.AND.EDGE(2*NBANDS).EQ.0.5) KKK=1
      IF(NFCNS.LE.3) KKK=1
       IF(KKK.EQ.1) GO TO 405
       DTEMP=DCOS(PI2*GRID(1))
      DNUM=DCOS(PI2*GRID(NGRID))
      AA=2.0/(DTEMP-DNUM)
      BB=-(DTEMP+DNUM)/(DTEMP-DNUM)
  405  CONTINUE
      DO 430 J=1,NFCNS
      FT=(J-1)*DELF
      XT=DCOS(PI2*FT)
       IF(KKK.EQ.1) GO TO 410
      XT=(XT-BB)/AA
      FT=ACOS(XT)/PI2
  410  XE=X(L)
      IF(XT.GT.XE) GO TO 420
      IF((XE-XT).LT.FSH) GO TO 415
      L=L+1
      GO TO 410
  415  A(J)=Y(L)
      GO TO 425
  420  IF((XT-XE).LT.FSH) GO TO 415
      GRID(1)=FT
      A(J)=GEE(1,NZ)
  425  CONTINUE
       IF(L.GT.1) L=L-1
  430  CONTINUE
      GRID(1)=GTEMP
      DDEN=PI2/CN
      DO 510 J=1,NFCNS
       DTEMP=0.0
      DNUM=(J-1)*DDEN
      IF(NM1.LT.1) GO TO 505
```

```
      DO 500 K=1,NM1
  500  DTEMP=DTEMP+A(K+1)*DCOS(DNUM*K)
  505  DTEMP=2.0*DTEMP+A(1)
  510  ALPHA(J)=DTEMP
       DO 550 J=2,NFCNS
  550 ALPHA(J)=2.0*ALPHA(J)/CN
      ALPHA(1)=ALPHA(1)/CN
       IF(KKK.EQ.1) GO TO 545
      P(1)=2.0*ALPHA(NFCNS)*BB+ALPHA(NM1)
      P(2)=2.0*AA*ALPHA(NFCNS)
      Q(1)=ALPHA(NFCNS-2)-ALPHA(NFCNS)
      DO 540 J=2,NM1
       IF(J.LT.NM1) GO TO 515
      AA=0.5*AA
       BB=0.5*BB
  515  CONTINUE
      P(J+1)=0.0
      DO 520 K=1,J
       A(K)=P(K)
  520  P(K)=2.0*BB*A(K)
       P(2)=P(2)+A(1)*2.0*AA
       JM1=J-1
      DO 525 K=1,JM1
  525  P(K)=P(K)+Q(K)+AA*A(K+1)
      JP1=J+1
      DO 530 K=3,JP1
  530  P(K)=P(K)+AA*A(K-1)
       IF(J.EQ.NM1) GO TO 540
      DO 535 K=1,J
  535  Q(K)=-A(K)
       Q(1)=Q(1)+ALPHA(NFCNS-1-J)
  540  CONTINUE
       DO 543 J=1,NFCNS
  543  ALPHA(J)=P(J)
  545  CONTINUE
      IF(NFCNS.GT.3) RETURN
      ALPHA(NFCNS+1)=0.0
       ALPHA(NFCNS+2)=0.0
      RETURN
      END
      DOUBLE PRECISION FUNCTION D(K,N,M)
C        FUNCTION TO CALCULATE THE LAGRANGE  INTERPOLATION
C       COEFFICIENTS FOR THE USE IN THE FUNCTION GEE.
      DIMENSION IEXT(77),AD(77),ALPHA(77),X(77),Y(77)
      DIMENSION DES(1232),GRID(1232),WT(1232)
      DOUBLE PRECISION AD,DEV,X,Y
       DOUBLE PRECISION Q
       DOUBLE PRECISION PI2
      COMMON PI2,AD,DEV,X,Y,GRID,DES,WT,ALPHA,IEXT,NFCNS,NGRID
      D=1.0
      Q=X(K)
      DO 3 L=1,M
      DO 2 J=L,N,M
      IF(J-K)1,2,1
    1  D=2.0*D*(Q-X(J))
    2  CONTINUE
    3  CONTINUE
       D=1.0/D
       RETURN
        END
```

```
      DOUBLE PRECISION FUNCTION GEE(K,N)
C       FUNCTION TO EVALUATE THE FREQUENCY RESPONSE USING THE
C       LAGRANGE INTERPOLATION FORMULA IN THE BARYCENTRIC FORM
      DIMENSION IEXT(77),AD(77),ALPHA(77),X(77),Y(77)
      DIMENSION DES(1232),GRID(1232),WT(1232)
      DOUBLE PRECISION P,C,D,XF
      DOUBLE PRECISION PI2
       DOUBLE PRECISION AD,DEV,X,Y
      COMMON PI2,AD,DEV,X,Y,GRID,DES,WT,ALPHA,IEXT,NFCNS,NGRID
      P=0.0
      XF=GRID(K)
      XF=DCOS(PI2*XF)
      DD=0.0
      DO 1 J=1,N
      C=XF-X(J)
       C=AD(J)/C
      DD=DD+C
    1  P=P+C*Y(J)
      GEE=P/DD
      RETURN
      END
      SUBROUTINE OUCH
      WRITE(6,1)
    1  FORMAT(' ******FAILURE TO CONVERGE******'/
     1'0PROBABLE CAUSE IS MACHINE ROUNDING ERROR'/
     2'0THE IMPULSE RESPONSE MAY BE CORRECT'/
     3'0CHECK WITH AFREQUENCY RESPONSE')
      RETURN
      END
      FINISH
```

APPENDIX H

PROGRAM TO DESIGN LOW PASS DIGITAL BUTTERWORTH FILTERS USING A TI 58/59 PROGRAMMABLE POCKET CALCULATOR

Address	Code	Key	Address	Code	Key
000	76	2nd lbl	036	42	STO
001	11	A	037	06	06
002	70	2nd Rad	038	00	0
003	42	STO	039	42	STO
004	00	0	040	08	08
005	91	R/S	041	01	1
006	76	2nd lbl	042	42	STO
007	12	B	043	03	3
008	42	STO	044	76	2nd lbl
009	01	1	045	85	+
010	91	R/S	046	02	2
011	76	2nd lbl	047	65	X
012	13	C	048	43	RCL
013	42	STO	049	03	3
014	02	2	050	75	-
015	91	R/S	051	01	1
016	76	2nd lbl	052	95	=
017	15	E	053	65	X
018	43	RCL	054	89	2nd π
019	00	0	055	55	$\div$
020	75	-	056	02	2
021	01	1	057	55	$\div$
022	95	=	058	43	RCL
023	32	x⇄t	059	00	0
024	43	RCL	060	95	=
025	02	2	061	38	2nd sin
026	65	X	062	42	STO
027	89	2nd π	063	04	04
028	55	$\div$	064	00	0
029	43	RCL	065	77	2nd x$\geq$t
030	01	1	066	65	X
031	95	=	067	01	1
032	30	2nd tan	068	85	+
033	42	STO	069	02	2
034	05	5	070	65	X
035	33	x^2	071	43	RCL

Address	Code	Key	Address	Code	Key
072	04	4	110	95	=
073	65	X	111	55	÷
074	43	RCL	112	43	RCL
075	05	5	113	07	7
076	85	+	114	95	=
077	43	RCL	115	91	R/S
078	06	6	116	00	0
079	95	=	117	32	x⇌t
080	42	STO	118	43	RCL
081	07	7	119	00	0
082	55	÷	120	55	÷
083	43	RCL	121	02	2
084	06	6	122	95	=
085	95	=	123	22	INV
086	35	1/x	124	59	2nd Int
087	91	R/S	125	67	2nd x=t
088	02	2	126	34	√
089	65	X	127	22	INV
090	43	RCL	128	86	2nd Stflg
091	06	6	129	01	1
092	75	-	130	61	GTO
093	02	2	131	53	(
094	95	=	132	76	2nd lbl
095	55	÷	133	34	√
096	43	RCL	134	86	2nd Stflg
097	07	7	135	01	1
098	95	=	136	76	2nd lbl
099	91	R/S	137	53	(
100	43	RCL	138	69	2nd Op
101	07	7	139	28	28
102	75	-	140	43	RCL
103	04	4	141	00	0
104	65	X	142	55	÷
105	43	RCL	143	02	2
106	04	4	144	95	=
107	65	X	145	59	2nd Int
108	43	RCL	146	32	x⇌t
109	05	5	147	43	RCL

Address	Code	Key
148	08	8
149	77	2nd x≥t
150	75	-
151	87	2nd If flg
152	01	1
153	54	)
154	43	RCL
155	03	3
156	85	+
157	01	1
158	95	=
159	76	2nd lbl
160	45	y^x
161	42	STO
162	03	3
163	61	GTO
164	85	+
165	76	2nd lbl
166	54	)
167	43	RCL
168	03	3
169	85	+
170	02	2
171	95	=
172	61	GTO
173	45	y^x
174	76	2nd lbl
175	75	-
176	87	2nd If flg
177	01	1
178	55	÷
179	76	2nd lbl
180	65	X
181	43	RCL
182	05	5
183	85	+
184	01	1
185	95	=

Address	Code	Key
186	55	÷
187	43	RCL
188	05	5
189	95	=
190	35	1/x
191	91	R/S
192	43	RCL
193	05	5
194	85	+
195	01	1
196	95	=
197	55	÷
198	53	(
199	43	RCL
200	05	5
201	75	-
202	01	1
203	54	)
204	95	=
205	35	1/x
206	91	R/S
207	76	2nd lbl
208	55	÷
209	01	1
210	55	÷
211	00	0
212	95	=
213	91	R/S

APPENDIX I

PROGRAM TO DESIGN LOW PASS CHEBYSHEV I DIGITAL FILTERS USING A TI 58/59 PROGRAMMABLE POCKET CALCULATOR

Address	Code	Key	Address	Code	Key
000	76	2nd lbl	036	01	1
001	11	A	037	00	0
002	42	STO	038	45	y^x
003	05	5	039	53	(
004	42	STO	040	93	.
005	11	11	041	01	1
006	91	R/S	042	65	X
007	76	2nd lbl	043	43	RCL
008	12	B	044	01	1
009	42	STO	045	54	)
010	01	1	046	95	=
011	91	R/S	047	75	-
012	76	2nd lbl	048	01	1
013	13	C	049	95	=
014	42	STO	050	34	$\sqrt{\ }$
015	02	2	051	42	STO
016	91	R/S	052	01	1
017	76	2nd lbl	053	33	x^2
018	14	D	054	35	1/x
019	42	STO	055	85	+
020	03	3	056	01	1
021	91	R/S	057	95	=
022	76	2nd lbl	058	34	$\sqrt{\ }$
023	15	E	059	85	+
024	43	RCL	060	01	1
025	03	3	061	55	÷
026	55	÷	062	43	RCL
027	43	RCL	063	01	1
028	02	2	064	95	=
029	65	X	065	45	y^x
030	89	2nd π	066	53	(
031	95	=	067	01	1
032	70	2nd Rad	068	55	÷
033	30	2nd tan	069	43	RCL
034	33	x^2	070	00	0
034	42	STO	071	54	)
035	02	02	072	95	=

Address	Code	Key	Address	Code	Key
073	42	STO	112	77	2nd x⇄t
074	01	1	113	85	+
075	33	x^2	114	01	1
076	75	-	115	94	+/-
077	01	1	116	32	x⇄t
078	95	=	117	43	RCL
079	55	÷	118	05	5
080	02	2	119	22	INV
081	55	÷	120	67	2nd x=t
082	43	RCL	121	75	-
083	01	1	122	43	RCL
084	95	=	123	02	2
085	42	STO	124	34	√
086	03	3	125	65	X
087	43	RCL	126	43	RCL
088	01	1	127	03	3
089	33	x^2	128	95	=
090	85	+	129	35	1/x
091	01	1	130	42	STO
092	85	=	131	05	5
093	55	÷	132	85	+
094	02	2	133	01	1
095	55	÷	134	95	=
096	43	RCL	135	35	1/x
097	01	1	136	91	R/S
098	95	=	137	53	(
099	42	STO	138	01	1
100	04	04	139	75	-
101	00	0	140	43	RCL
102	32	x⇄t	141	05	5
103	76	2nd lbl	142	54	)
104	85	+	143	55	÷
105	43	RCL	144	53	(
106	05	5	145	01	1
107	75	-	146	85	+
108	02	2	147	43	RCL
109	95	=	148	05	5
110	42	STO	149	54	)
111	05	5	150	95	=

Address	Code	Key	Address	Code	Key
151	91	R/S	190	55	÷
152	43	RCL	191	02	2
153	00	0	192	55	÷
154	75	-	193	43	RCL
155	01	1	194	00	0
156	95	=	195	95	=
157	42	STO	196	42	STO
158	11	11	197	07	7
159	76	2nd lbl	198	38	2nd sin
160	75	-	199	65	X
161	01	1	200	43	RCL
162	32	x % t	201	03	3
163	43	RCL	202	95	=
164	00	0	203	42	STO
165	67	2nd x=t	204	01	1
166	55	÷	205	43	RCL
167	02	2	206	07	7
168	22	INV	207	39	2nd cos
169	49	2nd Prd	208	65	X
170	11	11	209	43	RCL
171	00	0	210	04	4
172	42	STO	211	95	=
173	06	6	212	33	x^2
174	76	2nd lbl	213	85	+
175	34	√‾	214	43	RCL
176	43	RCL	215	01	1
177	06	6	216	33	x^2
178	85	+	217	95	=
179	01	1	218	65	X
180	95	=	219	43	RCL
181	42	STO	220	02	2
182	06	6	221	95	=
183	65	X	222	42	STO
184	02	2	223	08	8
185	75	-	224	43	RCL
186	01	1	225	02	2
187	95	=	226	34	√‾
188	65	X	227	65	X
189	89	2nd π	228	02	2

Address	Code	Key	Address	Code	Key
229	65	X	267	65	X
230	43	RCL	268	02	2
231	01	1	269	95	=
232	55	÷	270	42	STO
233	43	RCL	271	10	10
234	08	8	272	43	RCL
235	95	=	273	01	1
236	42	STO	274	35	1/x
237	09	9	275	91	R/S
238	43	RCL	276	43	RCL
239	08	8	277	10	10
240	35	1/x	278	55	÷
241	85	+	279	43	RCL
242	01	1	280	01	1
243	85	+	281	95	=
244	43	RCL	282	91	R/S
245	09	9	283	43	RCL
246	95	=	284	07	7
247	42	STO	285	55	÷
248	01	1	286	43	RCL
249	43	RCL	287	01	1
250	08	8	288	95	=
251	35	1/x	289	91	R/S
252	85	+	290	43	RCL
253	01	1	291	11	11
254	75	-	292	32	x⇆t
255	43	RCL	293	43	RCL
256	09	9	294	06	6
257	95	=	295	67	2nd x=t
258	42	STO	296	55	÷
259	07	7	297	61	GTO
260	43	RCL	298	34	√‾
261	08	8	299	76	2nd lbl
262	35	1/x	300	55	÷
263	94	+/-	301	01	1
264	85	+	302	55	÷
265	01	1	303	00	0
266	95	=	304	95	=
			305	91	R/S