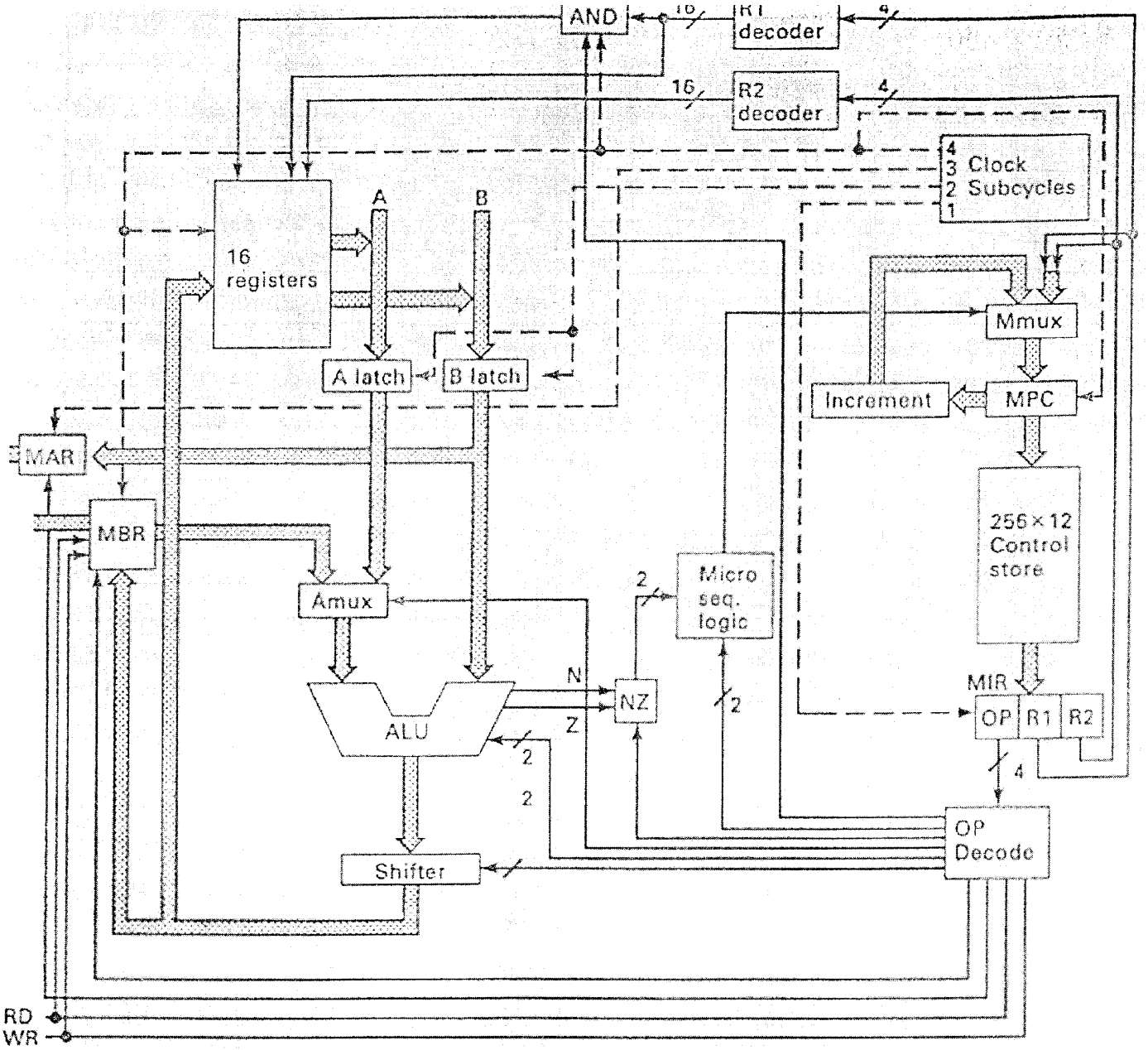
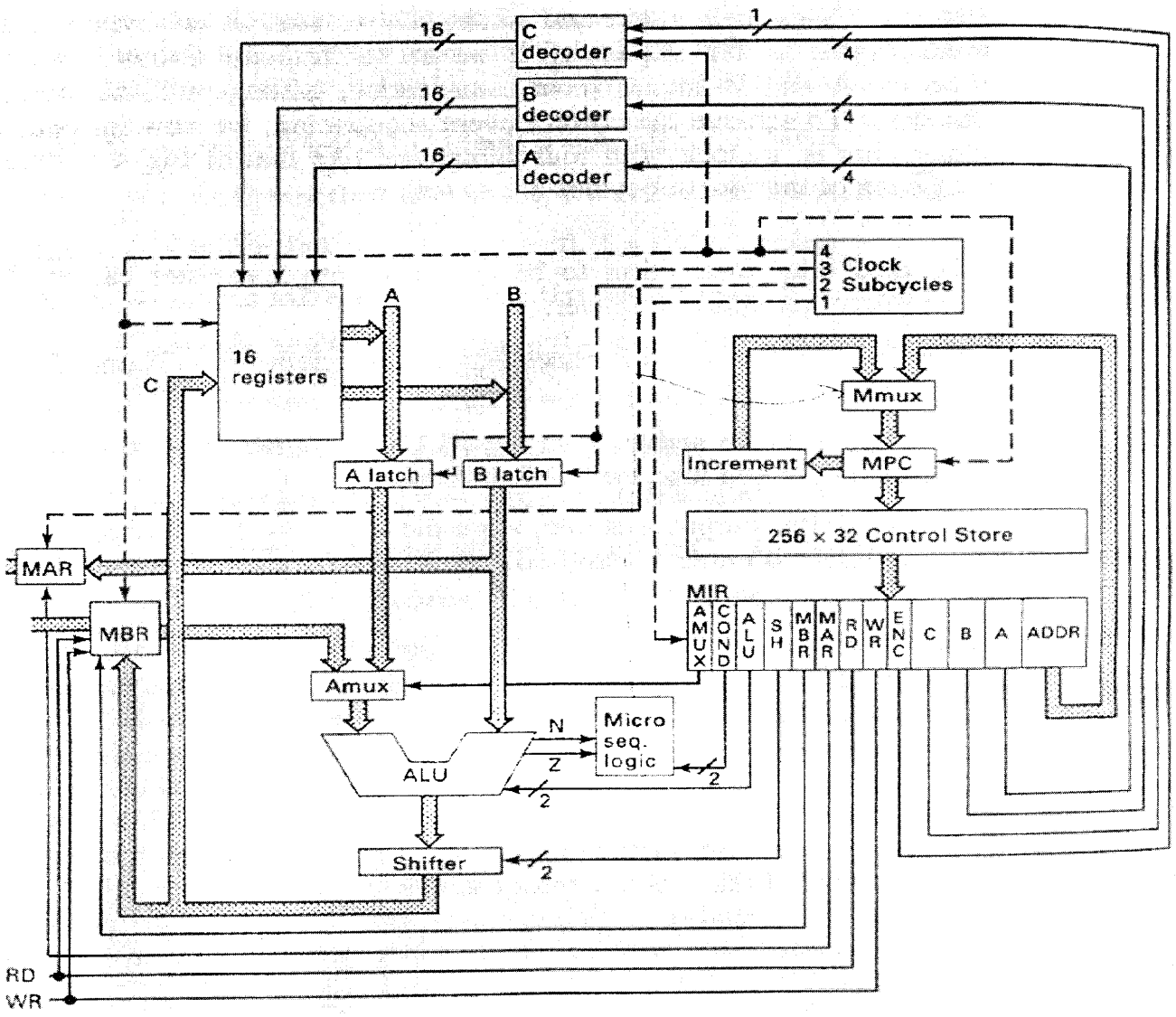


Binary	Mnemonic	Instruction	Meaning
0000	ADD	Addition	*r1* := *r1* + *r2*
0001	AND	Boolean AND	*r1* := *r1* AND *r2*
0010	MOVE	Move register	*r1* := *r2*
0011	COMPL	Complement	*r1* := *inv*(*r2*)
0100	LSHIFT	Left shift	*r1* := *lshift*(*r2*)
0101	RSHIFT	Right shift	*r1* := *rshift*(*r2*)
0110	GETMBR	Store MBR in register	*r1* := *mbr*
0111	TEST	Test register	if *r2* < 0 then *n* := *true*; if *r2* = 0 then *z* := *true*
1000	BEGRD	Begin read	*mar* := *r1*; *rd*
1001	BEGWR	Begin write	*mar* := *r1*; *mbr* := *r2*; *wr*
1010	CONRD	Continue read	*rd*
1011	CONWR	Continue write	*wr*
1100		(not used)	
1101	NJUMP	Jump if N=1	If *n* then goto *r*
1110	ZJUMP	Jump if Z=1	If *z* then goto *r*
1111	UJUMP	Unconditional jump	goto *r*

Microinstruction opcode		ALUL		SHL		AMUX	MAR		RD	MSLH	
		ALUH	SHH		NZ	AND	MBR		WR	MSLI	
0	ADD				+		+					
1	AND		+			+		+				
2	MOVE	+				+		+				
3	COMPL	+	+			+		+				
4	LSHIFT	+		+		+		+				
5	RSHIFT	+			+	+		+				
6	GETMBR	+				+	+	+				
7	TEST	+				+						
8	BEGRD	+							+		+			
9	BEGWR	+							+	+		+		
10	CONRD	+									+			
11	CONWR	+										+		
12														
13	NJUMP	+												+
14	ZJUMP	+											+	
15	UJUMP	+											+	+

```
 0: mar := pc; rd;
 1: rd;
    pc := pc + 1;
 2: ir := mbr;
    tir := lshift(ir);
    if n then goto 28;
 3: tir := lshift(tir);
    if n then goto 19;
 4: tir := lshift(tir);
    if n then goto 11;
 5: alu := tir;
    if n then goto 09;

 6: mar := ir; rd; {LODD}
 7: rd;
 8: ac := mbr;
    goto 0;

 9: mar := ir; mbr := ac; wr; {STOD}
10: wr;
    goto 0;

11: alu := tir;
    if n then goto 15;

12: mar := ir; rd; {ADDD}
13: rd;
14: a := mbr;
    ac := ac + a;
    goto 0;

15: mar := ir; rd; {SUBD}
16: rd;
99: ac := ac + 1;
17: a := mbr;
    a := inv(a);
18: ac := ac + a;
    goto 0;

19: tir := lshift(tir);
    if n then goto 25;
20: alu := tir;
    if n then goto 23;

21: alu := ac; {JPOS}
    if n then goto 0;
22: pc := ir;
    pc := band(pc, amask);
    goto 0;

23: alu := ac; {JZER}
    if z then goto 22;
24: goto 0;

25: alu := tir;
    if n then goto 27;
26: pc := ir; {JUMP}
    pc := band(pc, amask);
    goto 0;

27: ac := ir; {LOCO}
    ac := band(ac, amask);
    goto 0;

28: tir := lshift(tir);
    if n then goto 40;
29: tir := lshift(tir);
    if n then goto 35;
30: alu := tir;
    if n then goto 33;

31: a := ir; {LODL}
    a := a + sp;
32: mar := a; rd;
    rd;
    ac := mbr;
    goto 0;

33: a := ir; {STOL}
    a := a + sp;
34: mar := a; mbr := ac; wr;
    wr;
    goto 0;

35: alu := tir;
    if n then goto 38;

36: a := ir; {ADDL}
    a := a + sp;
37: mar := a; rd;
    rd;
    a := mbr;
    ac := ac + a;
    goto 0;

38: a := ir; {SUBL}
    a := a + sp;
39: mar := a; rd;
    rd;
    goto 99;

40: tir := lshift(tir);
    if n then goto 46;
41: alu := tir;
    if n then goto 44;

42: alu := ac; {JNEG}
    if n then goto 22;
43: goto 0;

44: alu := ac; {JNZE}
    if z then goto 0;
45: pc := ir;
    pc := band(pc, amask);
    goto 0;

46: tir := lshift(tir);
    if n then goto 50;

47: sp := sp + (-1); {CALL}
48: mar := sp; mbr := pc; wr;
    wr;
49: pc := ir;
    pc := band(pc, amask);
    goto 0;

50: tir := lshift(tir);
    if n then goto 65;
51: tir := lshift(tir);
    if n then goto 59;
52: alu := tir;
    if n then goto 56;

53: mar := ac; rd; {PSHI}
    rd;
54: sp := sp + (-1);
55: a := mbr;
    mar := sp; mbr := a; wr;
    wr;
    goto 0;

56: mar := sp; rd; {POPI}
57: rd;
    sp := sp + 1;

58: a := mbr;
    mar := ac; mbr := a; wr;
    wr;
    goto 0;
59: alu := tir;
    if n then goto 62;

60: sp := sp + (-1); {PUSH}
61: mar := sp; mbr := ac; wr
    wr;
    goto 0;

62: mar := sp; rd; {POP}
63: rd;
    sp := sp + 1;
64: ac := mbr;
    goto 0;

65: tir := lshift(tir);
    if n then goto 73;
66: alu := tir;
    if n then goto 70;

67: mar := sp; rd; {RETN}
68: rd;
    sp := sp + 1;
69: pc := mbr;
    goto 0;

70: a := ac; {SWAP}
71: ac := sp;
72: sp := a;
    goto 0;

73: alu := tir;
    if n then goto 76;

74: a := ir; {INSP}
    a := band(a, smask);
75: sp := sp + a;
    goto 0;

76: a := ir; {DESP}
    a := band(a, smask);
77: a := inv(a);
78: a := a + 1;
    sp := sp + a;
    goto 0;
```



Binary	Mnemonic	Instruction	Meaning
0000xxxxxxxxxxxx	LODD	Load direct	*ac* := *m*[*x*]
0001xxxxxxxxxxxx	STOD	Store direct	*m*[*x*] := *ac*
0010xxxxxxxxxxxx	ADDD	Add direct	*ac* := *ac* + *m*[*x*]
0011xxxxxxxxxxxx	SUBD	Subtract direct	*ac* := *ac* − *m*[*x*]
0100xxxxxxxxxxxx	JPOS	Jump positive	**if** *ac* ≥ 0 **then** *pc* := *x*
0101xxxxxxxxxxxx	JZER	Jump zero	**if** *ac* = 0 **then** *pc* := *x*
0110xxxxxxxxxxxx	JUMP	Jump	*pc* := *x*
0111xxxxxxxxxxxx	LOCO	Load constant	*ac* := *x* (0 ≤ *x* ≤ 4095)
1000xxxxxxxxxxxx	LODL	Load local	*ac* := *m*[*sp* + *x*]
1001xxxxxxxxxxxx	STOL	Store local	*m*[*x* + *sp*] := *ac*
1010xxxxxxxxxxxx	ADDL	Add local	*ac* := *ac* + *m*[*sp* + *x*]
1011xxxxxxxxxxxx	SUBL	Subtract local	*ac* := *ac* − *m*[*sp* + *x*]
1100xxxxxxxxxxxx	JNEG	Jump negative	**if** *ac* < 0 **then** *pc* := *x*
1101xxxxxxxxxxxx	JNZE	Jump nonzero	**If** *ac* ≠ 0 **then** *pc* := *x*
1110xxxxxxxxxxxx	CALL	Call procedure	*sp* := *sp* − 1; *m*[*sp*] := *pc*; *pc* := *x*
1111000000000000	PSHI	Push indirect	*sp* := *sp* − 1; *m*[*sp*] := *m*[*ac*]
1111001000000000	POPI	Pop indirect	*m*[*ac*] := *m*[*sp*]; *sp* := *sp* + 1
1111010000000000	PUSH	Push onto stack	*sp* := *sp* − 1; *m*[*sp*] := *ac*
1111011000000000	POP	Pop from stack	*ac* := *m*[*sp*]; *sp* := *sp* + 1
1111100000000000	RETN	Return	*pc* := *m*[*sp*]; *sp* := *sp* + 1
1111101000000000	SWAP	Swap *ac*, sp	*tmp* := *ac*; *ac* := *sp*; *sp* := *tmp*
11111100yyyyyyyy	INSP	Increment sp	*sp* := *sp* + *y* (0 ≤ *y* ≤ 255)
11111110yyyyyyyy	DESP	Decrement sp	*sp* := *sp* − *y* (0 ≤ *y* ≤ 255)

xxxxxxxxxxxx is a 12-bit machine address; in column 4 it is called *x*.
yyyyyyyy is an 8-bit constant; in column 4 it is called *y*.

Statement	AMUX	COND	ALU	SH	MBR	MAR	RD	WD	ENC	C	B	A	ADDR
mar := *pc*; *rd*	0	0	2	0	0	1	1	0	0	0	0	0	00
rd	0	0	2	0	0	0	1	0	0	0	0	0	00
ir := *mbr*	1	0	2	0	0	0	0	0	1	3	0	0	00
pc := *pc* + 1	0	0	0	0	0	0	0	0	1	0	6	0	00
mar := *ir*; *mbr* := *ac*; *wr*	0	0	2	0	1	1	0	1	0	0	3	1	00
alu := *tir*; **if** *n* **then goto** 15	0	1	2	0	0	0	0	0	0	0	0	4	15
ac := *inv* (*mbr*)	1	0	3	0	0	0	0	0	1	1	0	0	00
tir := *lshift* (*tir*); **if** *n* **then goto** 25	0	1	2	2	0	0	0	0	1	4	0	4	25
alu := *ac*; **if** *z* **then goto** 22	0	2	2	0	0	0	0	0	0	0	0	1	22
ac := *band* (*ir*, *amask*); **goto** 0	0	3	1	0	0	0	0	0	1	1	8	3	00
sp := *sp* + (−1); *rd*	0	0	0	0	0	0	1	0	1	2	2	7	00
tir := *lshift* (*ir* + *ir*); **if** *n* **then goto** 69	0	1	0	2	0	0	0	0	1	4	3	3	69

```
0: mar := pc; rd;                                  {main loop}
1: pc := pc + 1; rd;                               {increment pc}
2: ir := mbr; if n then goto 28;                   {save, decode mbr}
3: tir := lshift(ir + ir); if n then goto 19;
4: tir := lshift(tir); if n then goto 11;          {000x or 001x?}
5: alu := tir; if n then goto 9;                   {0000 or 0001?}

6: mar := ir; rd;                                  {0000 = LODD}
7: rd;
8: ac := mbr; goto 0;

9: mar := ir; mbr := ac; wr;                       {0001 = STOD}
10: wr; goto 0;

11: alu := tir; if n then goto 15;                 {0010 or 0011?}

12: mar := ir; rd;                                 {0010 = ADDD}
13: rd;
14: ac := mbr + ac; goto 0;

15: mar := ir; rd;                                 {0011 = SUBD}
16: ac := ac + 1; rd;                              {Note: x - y = x + 1 + not y}
17: a := inv(mbr);
18: ac := ac + a; goto 0;

19: tir := lshift(tir); if n then goto 25;         {010x or 011x?}
20: alu := tir; if n then goto 23;                 {0100 or 0101?}

21: alu := ac; if n then goto 0;                   {0100 = JPOS}
22: pc := band(ir, amask); goto 0;                 {perform the jump}

23: alu := ac; if z then goto 22;                  {0101 = JZER}
24: goto 0;                                        {jump failed}

25: alu := tir; if n then goto 27;                 {0110 or 0111?}
26: pc := band(ir, amask); goto 0;                 {0110 = JUMP}

27: ac := band(ir, amask); goto 0;                 {0111 = LOCO}

28: tir := lshift(ir + ir); if n then goto 40;     {10xx or 11xx?}
29: tir := lshift(tir); if n then goto 35;         {100x or 101x?}
30: alu := tir; if n then goto 33;                 {1000 or 1001?}

31: a := ir + sp;                                  {1000 = LODL}
32: mar := a; rd; goto 7;

33: a := ir + sp;                                  {1001 = STOL}
34: mar := a; mbr := ac; wr; goto 10;

35: alu := tir; if n then goto 38;                 {1010 or 1011?}

36: a := ir + sp;                                  {1010 = ADDL}
37: mar := a; rd; goto 13;

38: a := ir + sp;                                  {1011 = SUBL}
39: mar := a; rd; goto 16;

40: tir := lshift(tir); if n then goto 46;         {110x or 111x?}
41: alu := tir; if n then goto 44;                 {1100 or 1101?}

42: alu := ac; if n then goto 22;                  {1100 = JNEG}
43: goto 0;

44: alu := ac; if z then goto 0;                   {1101 = JNZE}
45: pc := band(ir, amask); goto 0;

46: tir := lshift(tir); if n then goto 50;

47: sp := sp + (-1);                               {1110 = CALL}
48: mar := sp; mbr := pc; wr;
49: pc := band(ir, amask); wr; goto 0;

50: tir := lshift(tir); if n then goto 65;         {1111, examine addr}
51: tir := lshift(tir); if n then goto 59;
52: alu := tir; if n then goto 56;

53: mar := ac; rd;                                 {1111000 = PSHI}
54: sp := sp + (-1); rd;
55: mar := sp; wr; goto 10;

56: mar := sp; sp := sp + 1; rd;                   {1111001 = POPI}
57: rd;
58: mar := ac; wr; goto 10;

59: alu := tir; if n then goto 62;

60: sp := sp + (-1);                               {1111010 = PUSH}
61: mar := sp; mbr := ac; wr; goto 10;

62: mar := sp; sp := sp + 1; rd;                   {1111011 = POP}
63: rd;
64: ac := mbr; goto 0;

65: tir := lshift(tir); if n then goto 73;
66: alu := tir; if n then goto 70;

67: mar := sp; sp := sp + 1; rd;                   {1111100 = RETN}
68: rd;
69: pc := mbr; goto 0;

70: a := ac;                                       {1111101 = SWAP}
71: ac := sp;
72: sp := a; goto 0;

73: alu := tir; if n then goto 76;

74: a := band(ir, smask);                          {1111110 = INSP}
75: sp := sp + a; goto 0;

76: a := band(ir, smask);                          {1111111 = DESP}
77: a := inv(a);
78: a := a + 1; goto 75;
```