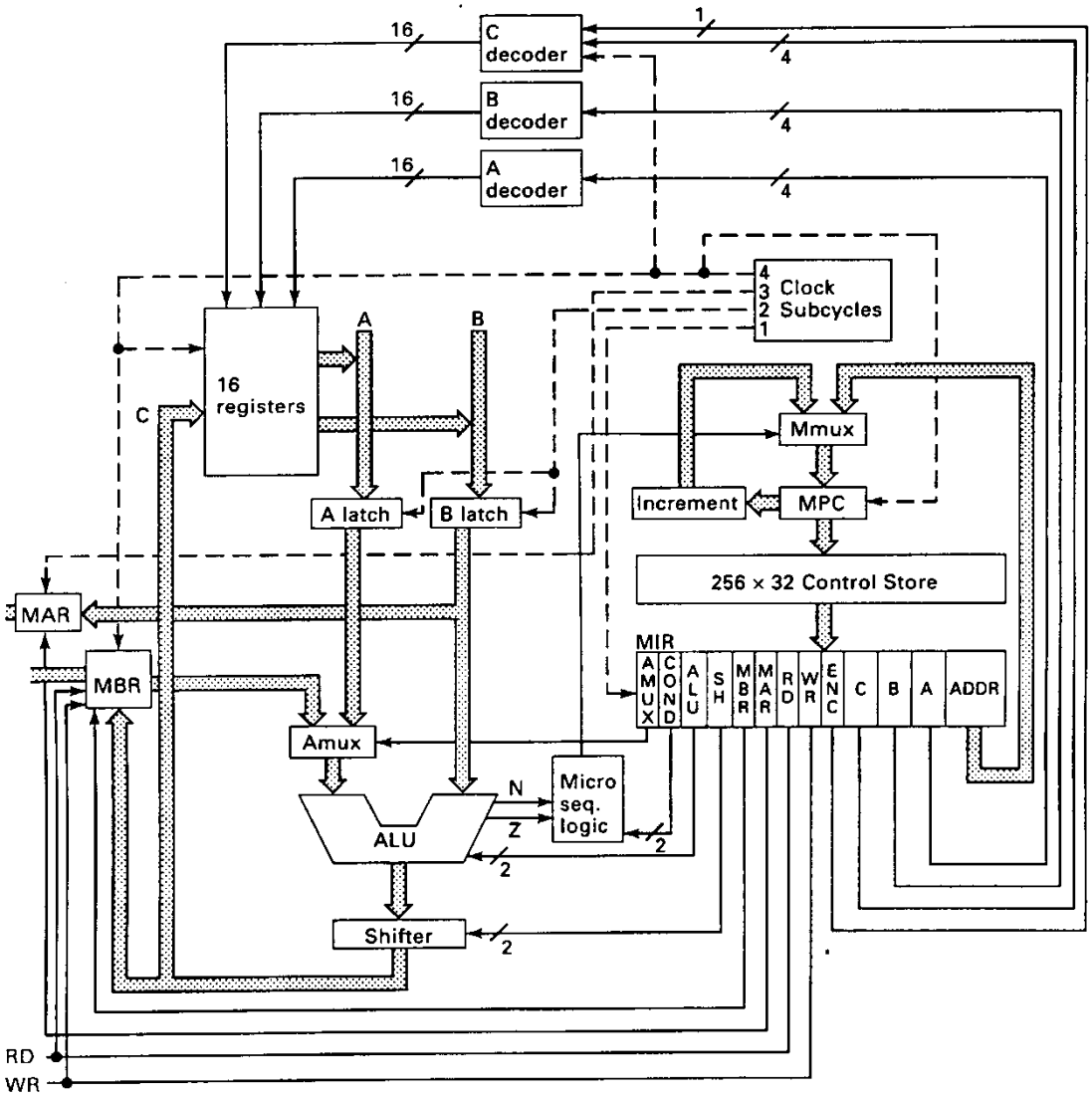
Binary	Mnemonic	Instruction	Meaning
0000xxxxxxxxxxxx	LODD	Load direct	$ac := m[x]$
0001xxxxxxxxxxxx	STOD	Store direct	$m[x] := ac$
0010xxxxxxxxxxxx	ADDD	Add direct	$ac := ac + m[x]$
0011xxxxxxxxxxxx	SUBD	Subtract direct	$ac := ac - m[x]$
0100xxxxxxxxxxxx	JPOS	Jump positive	**if** $ac \geq 0$ **then** $pc := x$
0101xxxxxxxxxxxx	JZER	Jump zero	**if** $ac = 0$ **then** $pc := x$
0110xxxxxxxxxxxx	JUMP	Jump	$pc := x$
0111xxxxxxxxxxxx	LOCO	Load constant	$ac := x\ (0 \leq x \leq 4095)$
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** $ac \neq 0$ **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	$sp := sp + y\ (0 \leq y \leq 255)$
11111110yyyyyyyy	DESP	Decrement sp	$sp := sp - y\ (0 \leq y \leq 255)$

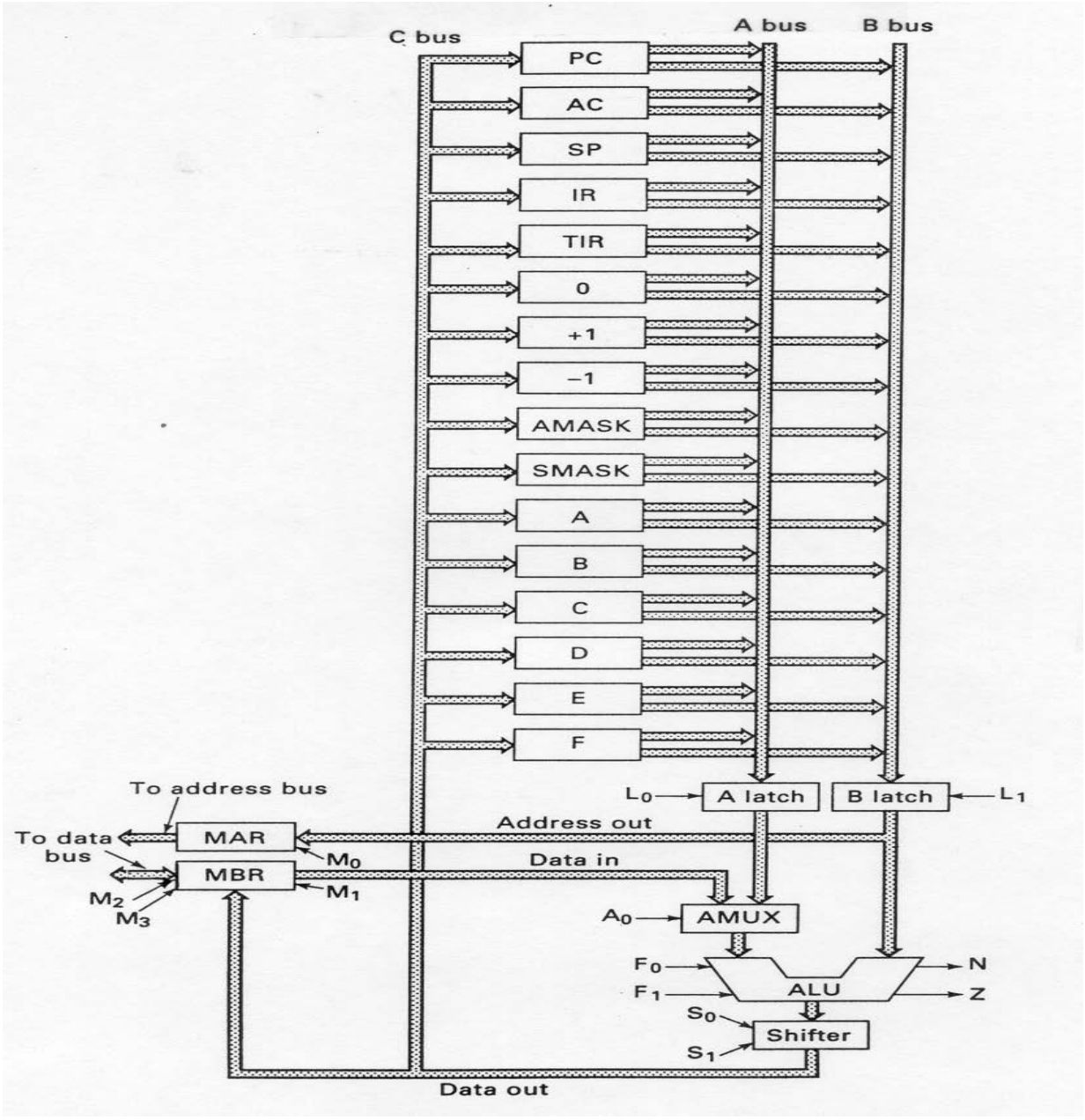
xxxxxxxxxxxx is a 12-bit machine address; in column 4 it is called x.
yyyyyyyy is an 8-bit constant; in column 4 it is called y.



AMUX	COND	ALU	SH	MBR,MAR,RD,WR,ENC
0 = A latch	0 = no jmp	0 = A + B	0 = no shift	0 = no
1 = MBR	1 = jmp if n=1	1 = A and B	1 = shift rt	1 = yes
	2 = jmp if z=1	2 = A	2 = shift lt	
	3 = always jmp	3 = not A		

Some Powers of 2

$2^{15} = 32768$
$2^{14} = 16384$
$2^{13} = 8192$
$2^{12} = 4096$
$2^{11} = 2048$
$2^{10} = 1024$
$2^{9} = 512$ you're on your own for the rest


C bus
A bus
B bus
PC
AC
SP
IR
TIR
0
+1
−1
AMASK
SMASK
A
B
C
D
E
F
To address bus
To data bus
MAR
MBR
M0
M1
M2
M3
L0
A latch
B latch
L1
Address out
Data in
A0
AMUX
F0
F1
ALU
N
Z
S0
S1
Shifter
Data out

```
0:mar := pc; rd;                                    { main loop  }
1:pc := 1 + pc; 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 := ac + mbr; goto 0;
15:mar := ir; rd;                                   { 0011 = SUBD }
16:ac := 1 + ac; rd;                              { Note: x-y=x+1+not y }
17:a := inv(mbr);
18:ac := a + ac; 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 := sp + ir;                                    { 1000 = LODL }
32:mar := a; rd; goto 7;
33:a := sp + ir;                                    { 1001 = STOL }
34:mar := a; mbr := ac; wr; goto 10;
35:alu := tir; if n then goto 38;                   { 1010 or 1011? }
36:a := sp + ir;                                    { 1010 = ADDL }
37:mar := a; rd; goto 13;
38:a := sp + ir;                                    { 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:tir := tir + tir; if n then goto 80;
77:a := band(ir, smask);                         { 11111110 = DESP }
78:a := inv(a);
79:a := a + 1; goto 75;
80:rd; wr;                                       { 11111111 = HALT }
```