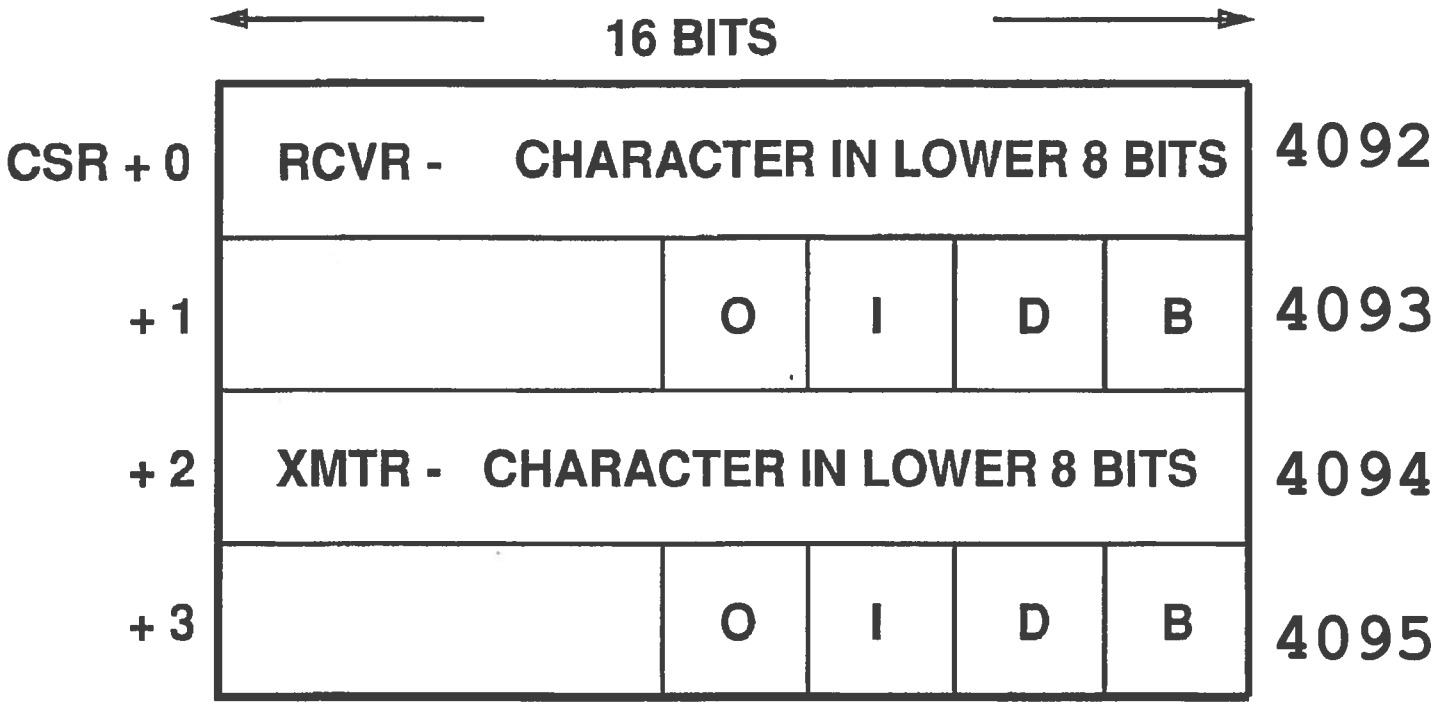
Instruction set of the Mic1 Macro Language

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	$sp := sp - y (0 \leq y \leq 255)$

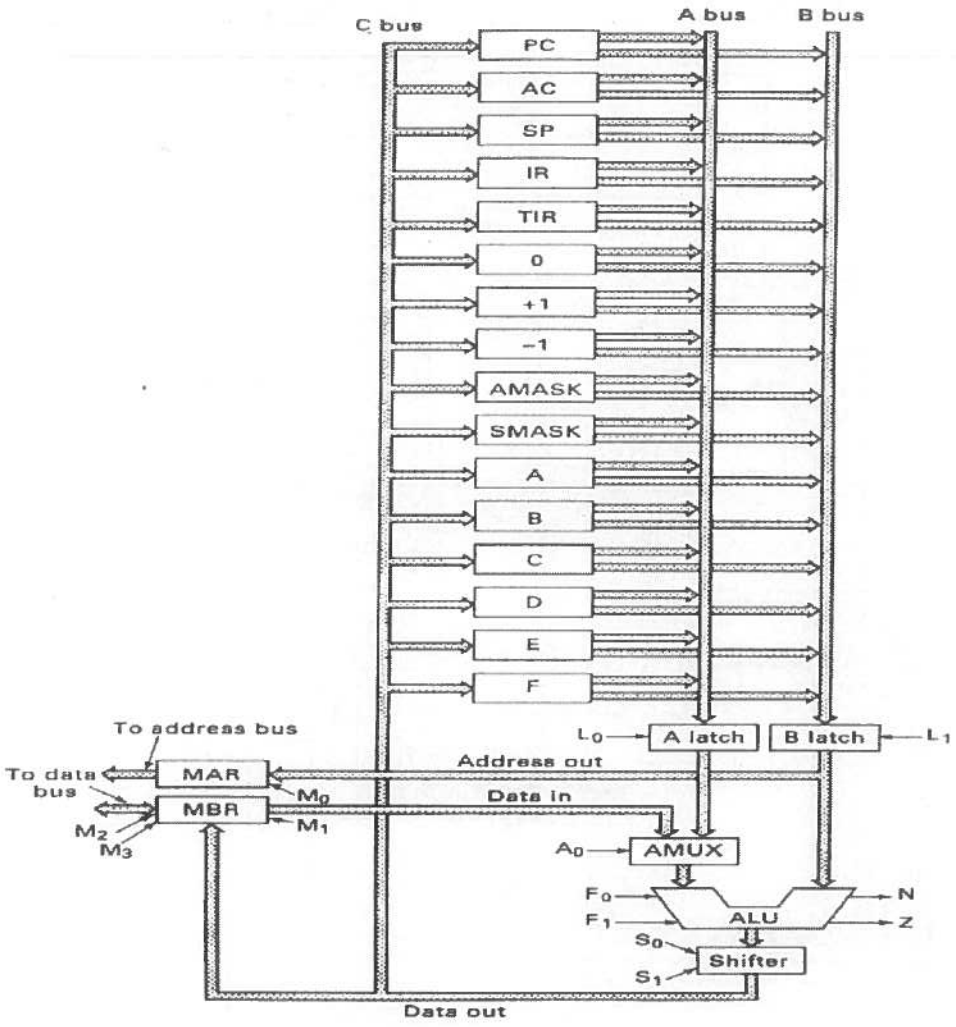
xxxxxxxxxxxx is a 12-bit machine address; in column 4 it is called x.
yyyyyyyy is an 8-bit constant; in column 4 it is called y.

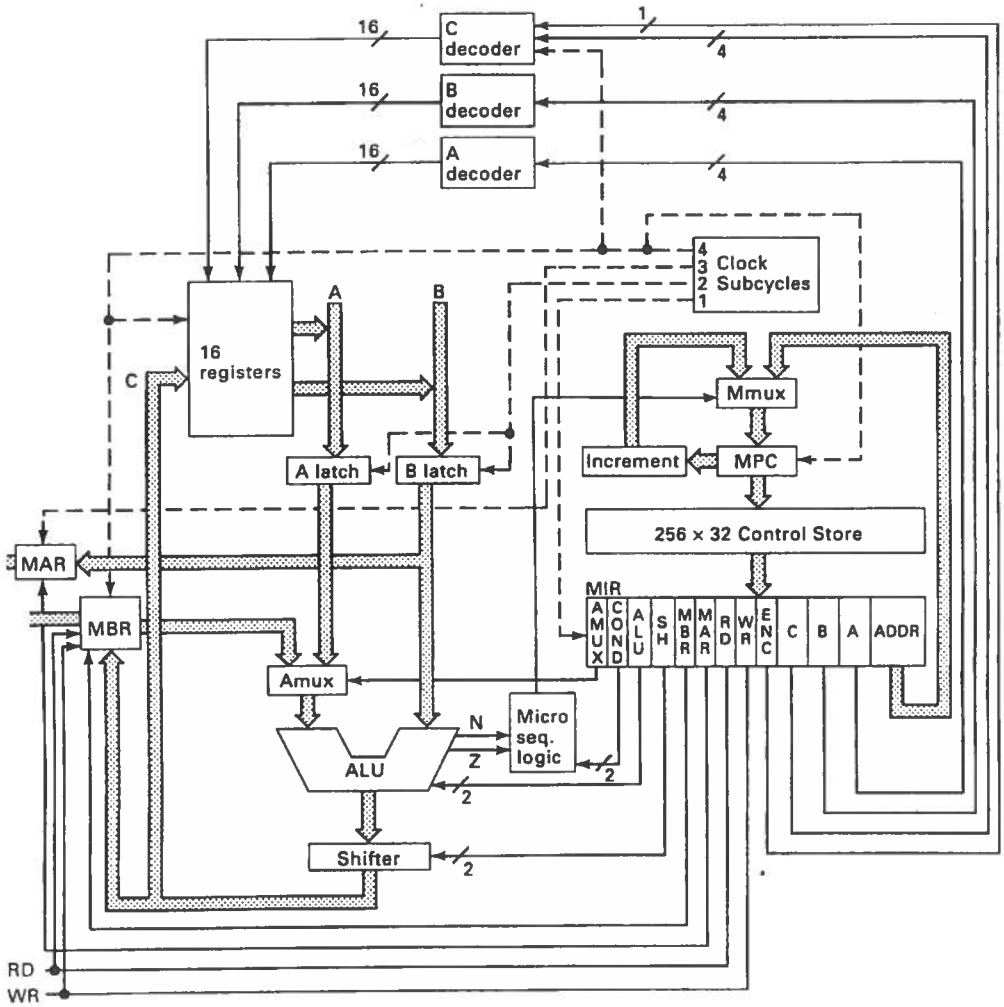
The serial line controller for our machine simulation uses the layout shown above. The receiver will accept characters from a terminal device only if it has been turned ON by setting the O bit in the RCVR status register. Turning the receiver on will set B and clear D. When a character has arrived in the RCVR, D will be set and B cleared. If the I bit is set in the RCVR status register then an interrupt will be posted each time the done bit is set. The removal of a character from the RCVR register will clear the interrupt and the done bit and will set busy. If the O bit is cleared the device is effectively turned off, clearing all other bits.

The transmitter must have its O bit set to become active. When O is set the transmitter will immediatlely set done and force an interrupt if I is set. Any characters moved into the XMTR character register at this point will cause D to clear and will clear any outstanding interrupt. Such a charcter will then be drawn on the terminal display and upon completion, the D bit will be set, the B bit cleared and an interrupt will be posted if I is set. Moving another character to the transmitter will clear D and any interrupt as discussed above.


16 BITS
CSR + 0
RCVR - CHARACTER IN LOWER 8 BITS
4092
+ 1
O
I
D
B
4093
+ 2
XMTR - CHARACTER IN LOWER 8 BITS
4094
+ 3
O
I
D
B
4095

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL


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


16
C decoder
1
4
16
B decoder
4
16
A decoder
4
4
3 Clock
2 Subcycles
1
16 registers
C
A
B
Mmux
A latch
B latch
Increment
MPC
256 x 32 Control Store
MAR
MBR
MIR
AMUX
COND
ALU
SH
MBR
MAR
RD
WR
ENC
C
B
A
ADDR
Amux
N
Micro seq. logic
Z
ALU
2
2
Shifter
2
RD
WR

```
 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;
```