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.
yyyyyyy is an 8-bit constant; in column 4 it is called y.

The various instruction formats include:

4 bit opcodes with remaining 12 bits used as either address or immediate value. In both cases the 12 bits are treated as an unsigned magnitude integer with range from 0 to 4095

0000 - 1110 Op Codes from LODD to CALL	Used an a 12 bit address range 0 to 4095 Or a 12 bit unsigned integer with this range

7 bit opcodes with the eighth bit unused and the low 8 bits used only as a positive value with range of 0 to 255 for the INSP and DESP (increment/decrement stack pointer) instructions (always zeros for other 7 bit opcodes)

1111000 - 1111111 Op Codes from PSHI to DESP		Low 8 bits unused except for INSP and DESP where 0 - 255 range

Eighth bit unused except with the halt instruction: 11111111

Data use is (for now) based on simple 16 bit 2s complement integers:

Sign Bit	15 bits of integer significance, providing values from -32K to +(32K - 1)

Below is a simple example of a program that includes a function called **adder** that takes two arguments that include the address of an array of 2s complement integers, and the number of elements in that array, such that its signature is:

adder array_count array_address

The program sets up the stack with the appropriate argument values and then calls **adder**. The **adder** routine finds the array of numbers, adds them together and then returns with the sum in the **AC** (as previously mentioned, the convention is to return function results in the AC). The main program, upon return from the adder call, then stores the AC contents into the memory **rslt:** location and calls halt to enter the debugger.

```
start:  lodd daddr:      ;load AC with data address
        push             ;push AC to stack (2nd arg)
        lodd dcnt:       ;load AC with data count
        push             ;push AC to stack (1st arg)
        call adder:      ;push return address on stack
        stod rslt:       ;store AC (has sum) to rslt: location
        halt             ;enter debugger
daddr:  data:            ;location holds data array address
data:   25               ;first of 5 data values
        50
        75
        100
        125              ;last of 5 data values
dcnt:   5                ;location holds data array element count
rslt:   0                ;location for the sum to be stored
        .LOC 20          ;forces adder routine to start at location 20
adder:  lodl 1           ;get 1st arg from stack into AC (data count)
        stod mycnt:      ;store count at location mycnt:
        lodl 2           ;get 2nd arg from stack into AC (data addr)
        pshi             ;push indirect first datum to stack
        addd myc1:       ;add 1 (value at myc1:) to addr in AC
        stod myptr:      ;store new addr to location myptr:
loop:   lodd mycnt:      ;load AC with value at mycnt: (data count)
        subd myc1:       ;subtract 1 (value at myc1:) from AC
        jzer done:       ;if new data count is 0 go to location done:
        stod mycnt:      ;if more data to add, store new data count
        lodd myptr:      ;load AC with addr of next datum
        pshi             ;push indirect next datum to stack
        addd myc1:       ;add 1 (value at myc1:) to addr in AC
        stod myptr:      ;store new addr to location myptr:
        pop              ;pop top of stack into AC (new datum)
        addl 0           ;add new top of stack location to AC
        insp 1           ;move stack pointer down one place
        push             ;push new sum in AC onto stack
        jump  loop:      ;jump to location loop:
done:   pop              ;come here when all data added, sum in AC
        retn             ;return to caller
        halt             ;should never get here (safety halt)
mycnt:  0                ;location for running count
myptr:  0                ;location for running data pointer
myc1:   1                ;location of a constant value of 1
```

The program from the previous page must be assembled, and then run with the Mic1 emulator. The following is a transcript of this activity using the mercury system:

```
bash-2.05$ cd cs305
bash-2.05$ pwd
/usr/cs/fac1/bill/cs305
bash-2.05$ ./masm < adder.asm > adder.obj
bash-2.05$ ./mic1 prom.dat adder.obj 0 1024

Read in 81 micro instructions
Read in 45 machine instructions
Starting PC is : 0000000000000000  base 10:       0
Starting SP is : 0000010000000000  base 10:    1024


ProgramCounter : 0000000000000111  base 10:       7
Accumulator    : 0000000101110111  base 10:     375
InstructionReg : 1111111100000000  base 10:   65280
TempInstr      : 1000000000000000  base 10:   32768
StackPointer   : 0000001111111110  base 10:    1022
ARegister      : 1111111111111110  base 10:   65534
BRegister      : 0000000000000000  base 10:       0
CRegister      : 0000000000000000  base 10:       0
DRegister      : 0000000000000000  base 10:       0
ERegister      : 0000000000000000  base 10:       0
FRegister      : 0000000000000000  base 10:       0

Total cycles   : 683

Type decimal address to view memory,  q  to quit or  c  to continue: 7
     the location    7 has value 0000000000001000 , or     8  or signed      8
Type  <Enter>  to continue debugging
Type        q  to quit
Type        f for forward range
Type        b for backward range: f
Type the number of forward locations to dump: 10
     the location    8 has value 0000000000011001 , or    25  or signed     25
     the location    9 has value 0000000000110010 , or    50  or signed     50
     the location   10 has value 0000000001001011 , or    75  or signed     75
     the location   11 has value 0000000001100100 , or   100  or signed    100
     the location   12 has value 0000000001111101 , or   125  or signed    125
     the location   13 has value 0000000000000101 , or     5  or signed      5
     the location   14 has value 0000000101110111 , or   375  or signed    375
     the location   15 has value 1111111111111111 , or 65535  or signed     -1
     the location   16 has value 1111111111111111 , or 65535  or signed     -1
     the location   17 has value 1111111111111111 , or 65535  or signed     -1
Type decimal address to view memory,  q  to quit or  c  to continue: 1024
     the location 1024 has value 1111111111111111 , or 65535  or signed     -1
Type  <Enter>  to continue debugging
Type        q  to quit
Type        f for forward range
Type        b for backward range: b
Type the number of reverse locations to dump: 6
     the location 1023 has value 0000000000001000 , or     8  or signed      8
     the location 1022 has value 0000000000000101 , or     5  or signed      5
     the location 1021 has value 0000000000000101 , or     5  or signed      5
     the location 1020 has value 0000000101110111 , or   375  or signed    375
     the location 1019 has value 0000000001111101 , or   125  or signed    125
     the location 1018 has value 1111111111111111 , or 65535  or signed     -1
Type decimal address to view memory,  q  to quit or  c  to continue: q
MIC-1 emulator finishing, goodbye

bash-2.05$
```