# Mini-Intermediate Code Engine (mICE) Instruction Set and Usage

Rules:

1- All instructions are of the form:

OPCODE operand1, operand2, operand3

2- A space between the opcode and first operand is mandatory.

3- Any other spaces will be ignored.

4- The operands must be separated by commas. (Even if no operands are present, every instruction must include the two commas.)

5- The operands are of the form:

[#]digitString

6- The # is optional, it indicates the operand is immediate data. (i.e. data is stored in the instruction)

7- If the data is directly addressed (i.e. operand is address of data), the # must not appear.

8- Direct addressing is the default.

9- The digitString is a string of decimal digits that represent an integer quantity.

10- It may begin with a plus + or minus -.

The opcode is a literal string from the following (may be upper or lower case or mixed):

NOP ADD SUB MUL DIV MOD INC DEC STO JMP JEQ JNE JLT JLE JGT JGE SYS AND OR XOR NOT HLT

All operands are assumed to be addresses or literal that will have been resolved by a parser, or manually by a programmer. Destinations of jump instructions are the actual instruction number of the instruction to jump to. This is not the memory (byte) address of the instruction. The parser only needs to keep track of the instruction number.

All data memory addresses are assumed to be the word address (address of an integer, not a byte address).

Instructions and their addressing modes.

(I = immediate data, d = direct address)

Opcode operand1 operand2 operand3

==================================================

NOP null null null

ADD I/d I/d d

SUB I/d I/d d

MUL I/d I/d d

DIV I/d I/d d

MOD I/d I/d d

INC d I d

DEC d I d

STO I/d null d

JMP null null I

JEQ I/d I/d i

JNE I/d I/d i

JLT I/d I/d i

JLE I/d I/d i

JGT I/d I/d i

JGE I/d I/d i

SYS I I/d I/d

AND I/d I/d d

OR I/d I/d d

XOR I/d I/d d

NOT I/d null d

HLT null null null

Instructions explained.

(I = immediate data, d = direct address)

Opcode

NOP no operation is performed. (wait one cycle)

Can be useful in constructing branch structures.

ADD op3 = op1 + op2 op1 and op2 can be immediate

SUB op3 = op1 - op2 or direct, op3 is destination

MUL op3 = op1 \* op2 so it must be an address.

DIV op3 = op1 / op2

MOD op3 = op1 % op2

INC d I d inc and dec are

DEC d I d useful in loops

For inc and dec ,the first operand is a direct address. The second is immediate (a fixed amount). The result goes into the third operand (directly addressed), normally same as operand 1.

STO I/d null d

STO, the data (immediate or directly addressed) from op1 is stored in op3.

For all jump instructions, the third operand is a code address that is stored in the instruction itself. The # is mandatory on op3. The code address is the instruction number, not the memory address of the instruction. For all conditional jumps, op1 and op2 may be direct or immediate.

JMP null null I unconditional

JEQ I/d I/d I jump if op1 = op2

JNE I/d I/d I jump if op1 != op2

JLT I/d I/d I jump if op1 < op2

JLE I/d I/d I jump if op1 <= op2

JGT I/d I/d i jump if op1 > op2

JGE I/d I/d I jump if op1 >= op2

Instructions explained (continued).

(I = immediate data, d = direct address)

Opcode

SYS I I/d I/d

SYS is a system call. The first operand is a service number. The 2nd and 3rd are data or null. Negative service numbers are used for services that do not return a data value (e.g. write integer is -1).

Positive service numbers are used for those that DO return a data value (e.g. read integer is 1).

AND I/d I/d d op3 = op1 AND op2

OR I/d I/d d op3 = op1 OR op2

XOR I/d I/d d op3 = op1 XOR op2

Logical AND, OR, XOR are similar to arithmetic instructions. Op1 and op2 can be immediate or direct address. Op3 must be an address.

Logical NOT follows:

NOT I/d null d op3 = NOT op1

HLT null null null

HALT PROGRAM EXECUTION.

AN example program (with embedded documentation)

for the following, assume:

x is integer, storage location 0

y is integer, storage location 1

z is integer, storage location 2

pseudo-language location mini-assembler

instruction (instruction #) instruction

read x 0 sys #1,0,

read y 1 sys #1,1,

write x 2 sys #-1,0,

write y 3 sys #-1,1,

z=x+y 4 add 0,1,2

write z 5 sys #-1,2,

z=x-y 6 sub 0,1,2

write z 7 sys #-1,2,

z=x\*y 8 mul 0,1,2

write z 9 sys #-1,2,

z=x/y 10 div 0,1,2

write z 11 sys #-1,2,

z=x%y 12 mod 0,1,2

write z 13 sys #-1,2,

if x>y 14 JLE 0,1,#17

write '1111' 15 sys -1,#1111,

16 JMP ,,#18

else

write '2222' 17 sys #-1,#2222,

endif

if x<y 18 JGE 0,1,#20

write '3333' 19 sys #-1,#3333,

20 JMP ,,#22

else

write '4444' 21 sys #-1,#4444

endif

z = x && y 22 and 0,1,2

write z 23 sys #-1,2,

z = x || y 24 or 0,1,2

write z 25 sys #-1,2,

z = x xor y 26 xor 0,1,2

write z 27 sys #-1,2,

AN example program (cont'd)

pseudo-language location mini-assembler

instruction (instruction #) instruction

z = !z 28 not 2,,2

write z 29 sys #-1,2,

inc z 30 inc 2,#1,2

write z 31 sys #-1,2,

dec z 32 dec 2,#1,2

write z 33 sys #-1,2,

x=y 34 sto 2,,0

write x 35 sys #-1,0,

write y 36 sys #-1,1,

if x==y 37 jne 0,1,#39

write '5555' 38 sys #-1,#5555,

endif

if x!=y 39 jeq 0,1,#41

write '6666' 40 sys -1,#6666,

endif

if x >= y 41 jlt 0,1,#43

write '7777' 42 sys -1,#7777,

endif

if x <= y 43 jgt 0,1,#45

write '8888' 44 sys -1,#8888,

endif

nop 45 nop ,,

nop 46 nop ,,

halt 47 hlt ,,

Program Example (as the file really appears)

sys #1,0,

sys #1,1,

sys #-1,0,

sys #-1,1,

add 0,1,2

sys #-1,2,

sub 0,1,2

sys #-1,2,

mul 0,1,2

sys #-1,2,

div 0,1,2

sys #-1,2,

mod 0,1,2

sys #-1,2,

JLE 0,1,#17

sys #-1,#1111,

JMP ,,#18

sys #-1,#2222,

JGE 0,1,#20

sys #-1,#3333,

JMP ,,#22

sys #-1,#4444

and 0,1,2

sys #-1,2,

or 0,1,2

sys #-1,2,

xor 0,1,2

sys #-1,2,

not 2,,2

sys #-1,2,

inc 2,#1,2

sys #-1,2,

dec 2,#1,2

sys #-1,2,

sto 2,,0

sys #-1,0,

sys #-1,1,

jne 0,1,#39

sys #-1,#5555,

jeq 0,1,#41

jlt 0,1,#43

sys #-1,#7777,

jgt 0,1,#45

sys #-1,#8888,

nop ,,

nop ,,

hlt ,,