Simple ACC Assembler

Assembly language is machine dependent, depending on a specific computer architecture.

Assembly language can be translated into lower level machine language, or interpreted.

- Software interpreter
- Virtual computer/machine

This virtual computer is a simple computer with

- ALU, RAM, just one register called ACCumulator, simple IO, and a built-in stack
- Word size is 2 bytes, and the addressability is thus 64k.

Program Format

- Each line
  - independent and self-contained
    - may be blank
    - a complete instruction
    - a complete storage directive
  - all delimiters are WS
- Instruction
  - for an accumulator machine
    (left argument and result are in an implicit accumulator ACC register, except for COPY),
    with the following format
    [Label:] XXX arguments
    - XXX is the reserved name
    - arguments as needed separated by spaces
- Storage directive
  - XXX val
    - XXX is storage name
    - val is the initial value
    - all storage are signed 2 byte integers
- Names
  - Instruction names are reserved in upper case
  - Variables start will letter and continue with letters and digits up to 8
- Numbers
  - The computer uses standard data 2's complement data representation so thus data range is -32k to +32k.

Instruction Set (# arguments, meaning)

- BR (1, jump to arg)
- BRNEG (1, jump to arg if ACC <0)
- BRZNEG (1, jump to arg if ACC <=0)
- BRPOS (1, jump to arg if ACC >0)
- BRZPOS (1, jump to arg if ACC >=0)
- BRZERO (1, jump to arg if ACC ==0)
- COPY (2, arg1 = arg2)
- ADD (1, ACC = ACC +arg
- SUB (1, ACC = ACC - arg)
- DIV (1, ACC = ACC / arg)
- MULT (1, ACC = ACC * arg)
- READ (1, arg=input integer)
- WRITE (1, put arg to output as integer)
- STOP (0, stop program)
- STORE (1, arg = ACC)
- LOAD (1, ACC=arg)
- NOOP (0, nothing)

Immediate Values

- ADD, DIV, MULT, WRITE, LOAD, SUB
  - can take either variable or immediate value as the argument
  - immediate value is positive integer or negative 2-byte integer

Stack

- PUSH (0, tos++)
- POP (0, tos--)
- STACKW (1,stack[tos-arg]=ACC)
- STACKR (1,ACC=stack[tos-arg])

PUSH/POP

- are only means to reserve/delete automatic storage.

STACKW/STACKR n

- these are stack access (random access) instructions.
- n must be a non-negative number
- the access is to nth element down from TOS
  NOTE: TOS points to the topmost element on the stack

Semantics

Execution begins with the first line and continues until STOP is reached

Invocation

> virtMach // read from stdin

> virtMach file.asm // read from file.asm

Location

/accounts/classes/janikowc/cs4280/asmInterpreter/virtMach

- readable executable

Examples

Simple flows

Read a number and print number+1

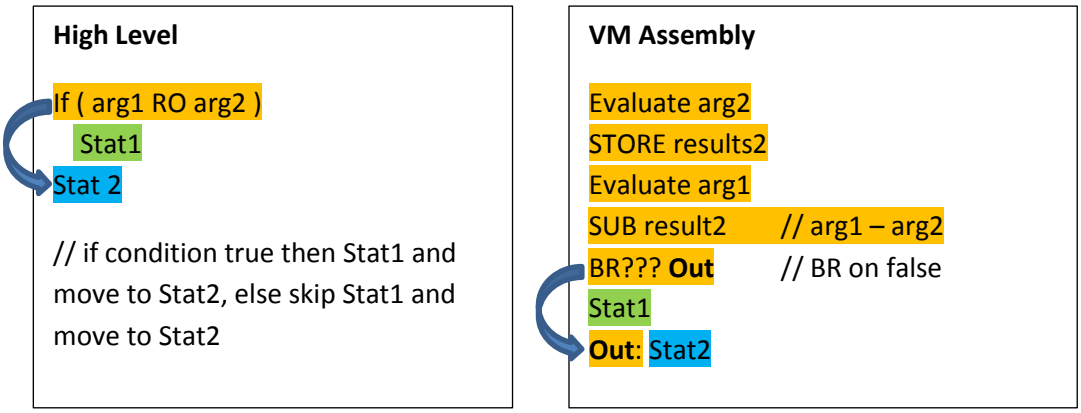
```
READ X
LOAD X
ADD 1
STORE X
WRITE X
STOP
X 0
```

Read 3 numbers, add them and display the total, using 3 variables

```
READ X
READ Y
READ Z
LOAD X
ADD Y
ADD Z
STORE X
WRITE X
STOP
X 0
Y 0
Z 0
```

Conditional Flow

- Simple relational conditions are processed by subtracting sides and implementing proper jump
- This is ‘if’ without else



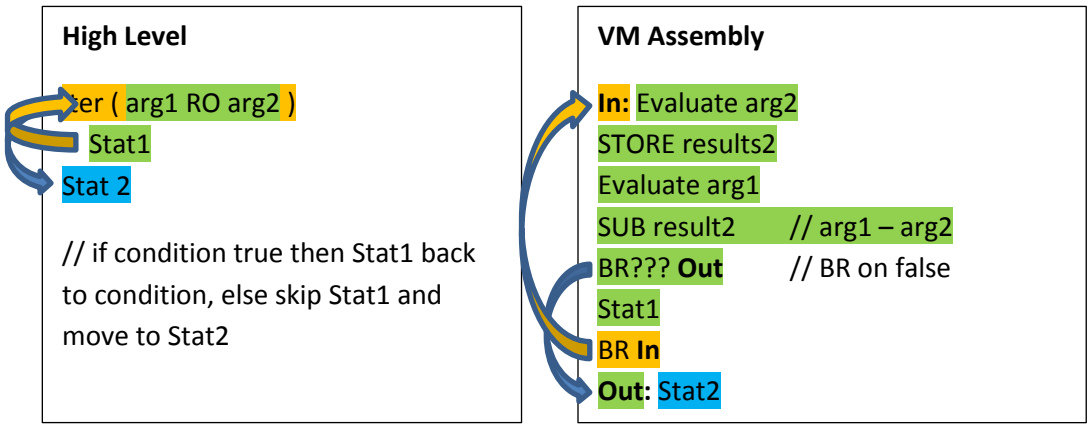
Read a number and print it if >= 1

```
program
var x = 0 .
start
  read x .
  iff ( x > > 1 )   & assume this is >= &
    print x .
stop
```

```
READ x
LOAD x
SUB 1
BRNEG Out
WRITE x
Out: STOP
x  0
```

Iteration

- Iteration is similar to conditional except that upon executing the 'true' statement there is unconditional jump back to evaluate the condition again.
- This will continue (iterate) until the condition is false



Read input number and print number then number–1 … down to 1

```
program
var x = 0 .
start
  read x .
  iter ( x > > 1  )
  start
     print x .
     let x = x – 1 .
  stop
stop
```

```
READ x
In: LOAD x
SUB 1
STORE x
BRNEG Out
WRITE x
LOAD x
SUB 1
BR In
Out: STOP
x 0
```