Assemblers and symbol tables

(Mostly) revision from last time: [see book table 4.2]
+ fig 4.8 ← show registers

MARIE instructions:

fill in interactively

Opcode	Mnemonic	Effect
1	Load X	AC = M[X]
2	Store X	M[X] = AC
3	Add X	AC = AC + M[X]
4	Subt X	AC = AC − M[X]
5	Input	AC = InReg
6	Output	OutReg = AC
7	Halt	—
8	Skipcond	see below
9	Jump X	PC = X

Recall SimpleAdd.mas [demo]

e.g.

```
Load 4
Add 5
Store 6
Halt
dec 12
dec 15
```

implements: M[6] = M[4] + M[5]

Behavior of skipcond:

```
skipcond 000    - skip next instruction if AC<0
skipcond 400    - skip next instruction if AC=0
skipcond 800    - skip next instruction if AC>0
```

Demo of simpleSkip.mas

pseudocode:

```
if M[008] < 0
    M[00B] = M[009]
else
    M[00B] = M[00A]
```

Note how unreadable the assembly language of this program is — mostly because it depends on numerical addresses. We'll see a better way soon.

An assembly language is a direct translation of machine language into human-readable form. It includes:

- mnemonics for instructions

- labels for addresses

- directives for other stuff, e.g. specifying constant values

- comments for additional info for a human reader

A mnemonic represents an opcode with a descriptive English word:

e.g. 300A becomes "Add 00A"

A label represents an address with a descriptive English word

e.g. Jump 0C3 becomes "Jump addNumbers"

- In MARIE's assembly language, a label is followed by a comma:

e.g.

```
loop,   load 0B3
        add  0B4
        jump loop
```

or

```
        load  data
        add   data
        store data
data,   dec   5
```

exercise: describe what these two programs do.

In MARIE assembler, the directive

"dec" means a constant value in decimal.
"hex" means a constant value in hex.

e.g. dec 33
hex 21

} represent the same binary word.

In MARIE assembler, the "/" character begins a comment

demo: see simpleSkip2 for how these features improve readability while maintaining 1-1 correspondence with machine language.

An assembler is a program that translates assembly language into machine language.

Assemblers build a symbol table mapping labels to addresses, then fill in actual addresses in instructions like 'load data'.

e.g.

```
000:          load data
001:          store dest
002:  data,   dec 7
003:  dest,   dec 0
```

builds table:

data	002
dest	003

so e.g. 'store dest' becomes '2003'.

demo using e.g. simpleSkip2

Activity:

Let X and Y be memory locations of your choice. (Use labels to specify them).

Implement the following pseudocode:

```
if (X > 6)
    Y = 3
else if (X > 1)
    Y = 4
end
```

"add indirect" "jump indirect"

If time, we also look at the AddI and JumpI instructions:

$$\text{AddI} \equiv AC = AC + M[M[x]]$$

$$\text{JumpI} \equiv PC = M[x]$$