Examples from this lecture

http://www.cs.pomona.edu/~dkauchak/classes/cs52/examples/cs52machine/

Computer internals
Computer internals simplified

CPU
What does it stand for?
What does it do?

RAM
What does it stand for?
What does it do?

CPU
(Central Processing Unit, aka "the processor")

RAM
(Random Access Memory, aka "memory" or "main memory")

Does all the work!
Temporary storage

Computer internals simplified

"the computer"

CPU
RAM
hard drive

Why do we need a hard drive?

"the computer"

CPU
RAM
hard drive

- Persistent memory
- RAM only stores data while it has power
**Computer simplified**

- Network
- Input devices
- Display
- CPU
- RAM
- Hard drive
- Media drive

**Inside the CPU**

- CPU processor: does the work
- Registers: local, fast memory slots

**Memory speed**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Access Time</th>
<th>Times Slower than Register Access</th>
<th>For Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>0.3 ns</td>
<td>1</td>
<td>1 s</td>
</tr>
<tr>
<td>RAM</td>
<td>120 ns</td>
<td>400</td>
<td>6 min</td>
</tr>
<tr>
<td>Hard disk</td>
<td>1 ms</td>
<td>million</td>
<td>1 month</td>
</tr>
<tr>
<td>google.com</td>
<td>0.4 s</td>
<td>billion</td>
<td>30 years</td>
</tr>
</tbody>
</table>

**Memory**

- RAM

- `010111100010000100010010...

- What is a byte?
Memory sizes

<table>
<thead>
<tr>
<th></th>
<th>bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8</td>
</tr>
<tr>
<td>kilobyte (KB)</td>
<td>$2^{10}$ bytes = ~8,000</td>
</tr>
<tr>
<td>megabyte (MB)</td>
<td>$2^{20}$ = ~8 million</td>
</tr>
<tr>
<td>gigabyte (GB)</td>
<td>$2^{30}$ = ~8 billion</td>
</tr>
</tbody>
</table>

My laptop has 16GB (gigabytes) of memory. How many bits is that?

~128 billion bits!

Memory is byte addressable
Memory is organized into “words”, which is the most common functional unit.

Most modern computers use 32-bit (4 byte) or 64-bit (8 byte) words.

We’ll use 16-bit words for our model (the CS52 machine).
In executing a program, the CS52 Machine follows a simple loop:

- The machine fetches the value at mem[ic] for use as an instruction.
- The machine increments the value in ic by 1.
- The machine decodes and carries out the instruction.

**CS52 machine instructions**

**CPU**

What types of operations might we want to do (think really basic)?

**CS52 machine code**

Four main types of instructions

1. math
2. branch/conditionals
3. memory
4. control the machine (e.g. stop it)
instruction name arguments
add     sub     and
        xor

Instruction/operation name
(always three characters)

operation arguments
R = register (e.g. r0)
S = signed number (byte)

1st R: register where the answer will go
2nd R: register of first operand
3rd S/R: register/value of second operand

What does this do?

1st R: register where the answer will go
2nd R: register of first operand
3rd S/R: register/value of second operand
**add r1 r2 r3**

\[ r1 = r2 + r3 \]

Add contents of registers r2 and r3 and store the result in r1

1\(^{st}\) R: register where the answer will go
2\(^{nd}\) R: register of first operand
3\(^{rd}\) S/R: register/value of second operand

**add r2 r1 10**

\[ r2 = r1 + 10 \]

Add 10 to the contents of register r1 and store in r2

1\(^{st}\) R: register where the answer will go
2\(^{nd}\) R: register of first operand
3\(^{rd}\) S/R: register/value of second operand

**add r2 r1 10**

What does this do?

1\(^{st}\) R: register where the answer will go
2\(^{nd}\) R: register of first operand
3\(^{rd}\) S/R: register/value of second operand

**add r1 r0 8**

neg r2 r1

sub r2 r1 r2

What number is in r2?

1\(^{st}\) R: register where the answer will go
2\(^{nd}\) R: register of first operand
3\(^{rd}\) S/R: register/value of second operand
add r1 r0 8  \hspace{1cm} r1 = 8
neg r2 r1  \hspace{1cm} r2 = -8, r1 = 8
sub r2 r1 r2  \hspace{1cm} r2 = 16

1st R: register where the answer will go
2nd R: register of first operand
3rd S/R: register/value of second operand

Adding 8 with the contents of r1 gives a result of 16.

Accessing memory

sto = save data in register TO memory
loa = put data FROM memory into a register

sto r1 r2  ; store the contents of r1 to mem[r2]
loa r1 r2  ; get data from mem[r2] and put into r1

Special cases:
- saving TO (sto) address 0 prints
- reading from (loa) address 0 gets input from user

Basic structure of CS52 program

; great comments at the top!

; 
; instruction1 ; comment
; instruction2 ; comment
; ...
; hlt

whitespace before operations/instructions
Running the CS52 machine

Look at subtract.a52
- load two numbers from the user
- subtract
- print the result

CS52 simulator

Different windows
- Memory (left)
- Instruction execution (right)
- Registers
- I/O and running program

```
beq r3 r0 done
```

What does this do?

```lisp
(brs B
  beq bne blt bge bgt ble)
```

1st R: first register for comparison
2nd R: second register in comparison
3rd B: label
beq r3 r0 done
If r3 = 0, branch to the label “done”
otherwise, ic is incremented as normal to
the next instruction

1st R: first register for comparison
2nd R: second register in comparison
3rd B: label

ble r2 r3 done
What does this do?

1st R: first register for comparison
2nd R: second register in comparison
3rd B: label

ble r2 r3 done
If r2 <= r3, branch to the label done

1st R: first register for comparison
2nd R: second register in comparison
3rd B: label

- Conditionals
- Loops
- Change the order that instructions are executed
CS52 machine execution

A program is simply a sequence of instructions stored in a block of contiguous words in the machine’s memory. In executing a program, the CS52 Machine follows a simple loop:
- The machine fetches the value at mem[pc] for use as an instruction.
- The machine increments the value in pc by 2.
- The machine decodes and carries out the instruction.

Basic structure of CS52 program

; great comments at the top!
; instruction1 ; comment
; instruction2 ; comment
... label1
; instruction ; comment
; instruction ; comment
label2
... hlt
end

- whitespace before operations/instructions
- labels go here

More CS52 examples

Look at max_simple.a52
- Get two values from the user
- Compare them
- Use a branch to distinguish between the two cases
  - Goal is to get largest value in r3
- print largest value