CS52 MACHINE

David Kauchak
CS 52 – Spring 2017

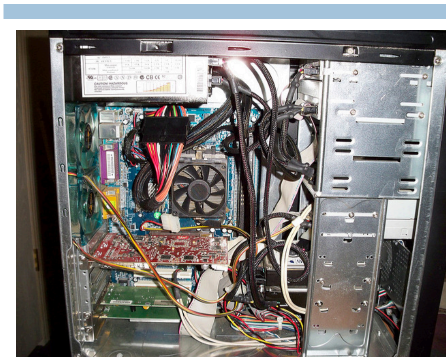
Admin

- Midterm 1
- Assignment 3
- Assignment 4

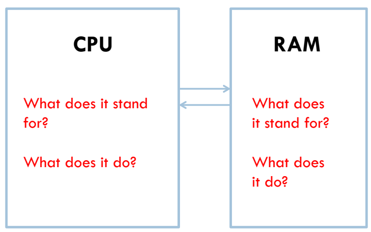
Examples from this lecture

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

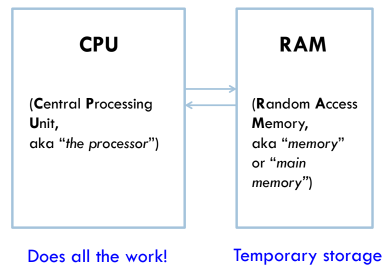
Computer internals



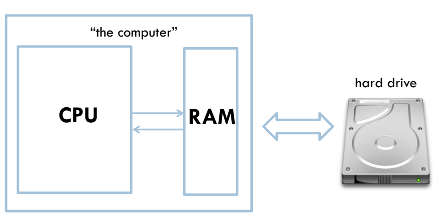
Computer internals simplified



Computer internals simplified

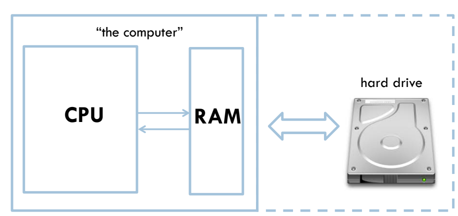


Computer internals simplified



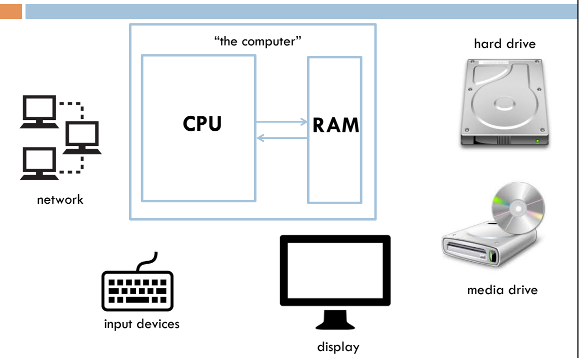
Why do we need a hard drive?

Computer internals simplified

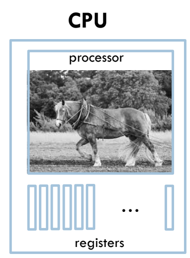


- Persistent memory
- RAM only stores data while it has power

Computer simplified



Inside the CPU



processor: does the work

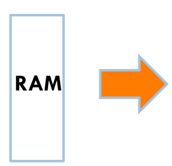
registers: local, fast memory slots

Why all these levels of memory?

Memory speed

operation	access time	times slower than register access	for comparison …
register	0.3 ns	1	1 s
RAM	120 ns	400	6 min
Hard disk	1ms	~million	1 month
google.com	0.4s	~billion	30 years

Memory

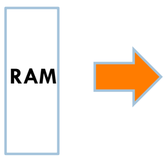


0101011110001010000010010 …

What is a byte?

?

Memory



01010111 10001010 00010010 …

byte = 8 bits
byte is abbreviated as B

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

Memory sizes

	bits
byte	8
kilobyte (KB)	2^10 bytes = ~8,000
megabyte (MB)	2^20 =~ 8 million
gigabyte (GB)	2^30 = ~8 billion

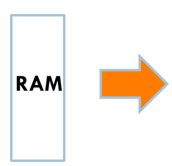
My laptop has 16GB (gigabytes) of memory. How many bits is that?

Memory sizes

	bits
byte	8
kilobyte (KB)	2^10 bytes = ~8,000
megabyte (MB)	2^20 =~ 8 million
gigabyte (GB)	2^30 = ~8 billion

~128 billion bits!

Memory



address	
0	01010111
1	10001010
2	00010010
3	01011010
…	…

Memory is byte addressable

Memory

RAM

address	
0	01010111
1	10001010
2	00010010
3	01011010
…	…

Memory is organized into "words", which is the most common functional unit

Memory

RAM

address	32-bit words
0	10101011 10001010 00010010 01011010
4	11001011 00001110 01010010 01010110
8	10111011 10010010 00000000 01110100
…	…

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

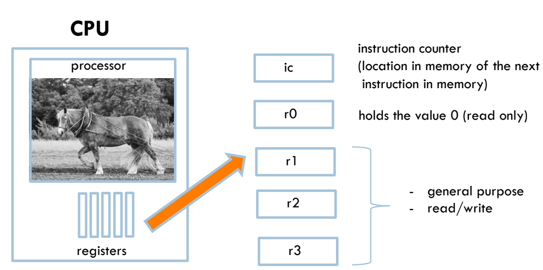
Memory in the CS52 Machine

RAM

address	16-bit words
0	10101011 10001010
2	00010010 01011010
4	11001011 00001110
…	…

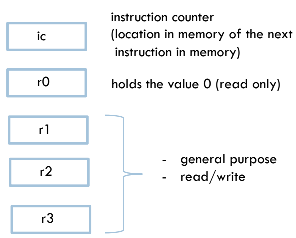
We'll use 16-bit words for our model (the CS52 machine)

CS52 machine



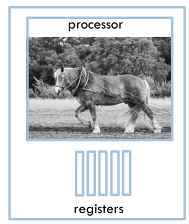
In executing a program, the CS52 Machine follows a simple loop:

- The machine fetches the value at `mem[ic]` for use as an instruction.
- The machine increments the value in `ic` by 2.
- 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 }  RRR or RRS
orr }
xor }
```

instruction name arguments

add
sub
and RRR or RRS
orr
xor

instruction/operation name
(always three characters)

---

instruction name arguments

add
sub
and RRR or RRS
orr
xor

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

---

instruction name arguments

add
sub
and RRR or RRS
orr
xor

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

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

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

add r2 r1 10

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 r2 r1 10

r2 = r1 + 10

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

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

add r1 r0 8
neg r2 r1
sub r2 r1 r2

What number is in r2?

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

```
add r1 r0 8      r1 = 8
neg r2 r1        r2 = -8, r1 = 8
sub r2 r1 r2     r2 = 16
```

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

Accessing memory

sto } RRS
loa }

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
```

Accessing memory

sto } RRS
loa }

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

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

```
brs  B
beq
bne
blt
bge   RRB
bgt
ble
```

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

beq r3 r0 done

What does this do?

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"
if not (else) 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

```
brs  B
beq ⎫
bne ⎪
blt ⎬ RRB
bge ⎪
bgt ⎪
ble ⎭
```

- 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[ic]` for use as an instruction.
- The machine increments the value in `ic` 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