Computer Architecture
aka DARK

—

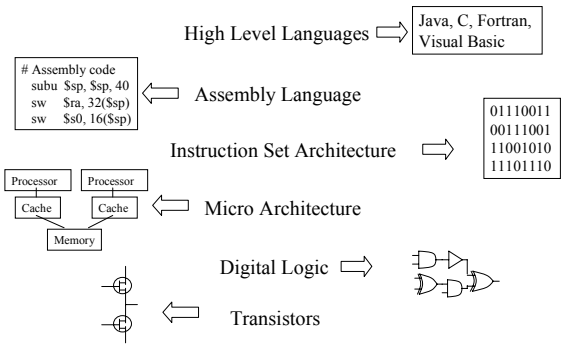
Welcome!

Erik Berg Jachim Parrow
Zoran Radovic Dan Wallin

Today's schedule

- Introduction
- Registration
- Computer accounts
- Entrance cards
- Numbers & data representation
- CPU, memory & I/O devices
- Assembly, part I

The Computer at Different Levels



Negative Numbers

Decimal	**Two's complement**
3	0000 0011
2	0000 0010
1	0000 0001
0	0000 0000
-1	1111 1111
-2	1111 1110
-3	1111 1101

Negate a Number

1. Complement all bits
2. Add one
   Done!

Example: Negate 23
23 = 0001 0111
Complement bits: => 1110 1000
Add one: => 1110 1001 = -23

And back again:
Complement bits: => 0001 0110
Add one: => 0001 0111 = 23

Sign Bit & Sign Extension

1110 1001 = -23 0001 0111 = +23

Sign bit
0 => positive (or zero)
1 => negative

Sign Extension
8-bit: 0010 0101 (= 37)
32-bit: 0000 0000 0000 0000 0000 0000 0010 0101

8-bit: 1110 1001 (=-23)

32-bit: 1111 1111 1111 1111 1111 1111 1110 1001

Hexadecimal Numbers

Decimal numbers use base 10:
$3125 = 3*10^4 + 10^2 + 2*10^1 + 5$

Binary numbers use base 2:
Digits: 0, 1
0101 1100 = $0*2^7+1*2^6+0*2^5+$
$+1*2^4+1*2^3+1*2^2+0*2^2+0*1 = 92$ decimalt

Hexadecimal numbers use base 16:
Digits: 0, 1, 2, ..., 9, a, b, c, d, e, f
0x a3f8 = $10*16^3 + 3*16^2 + 15*16 + 8 = 41976$ dec

Mark hex number

hex - dec
a = 10
b = 11
c = 12
d = 13
e = 14
f = 15

Characters and Strings - ASCII

•ASCII – American Standard Code for Information Interchange

•Modern Extension with åäö, Latin 1 or ISO 8859-15

Note:
lowercase = uppcase + 32

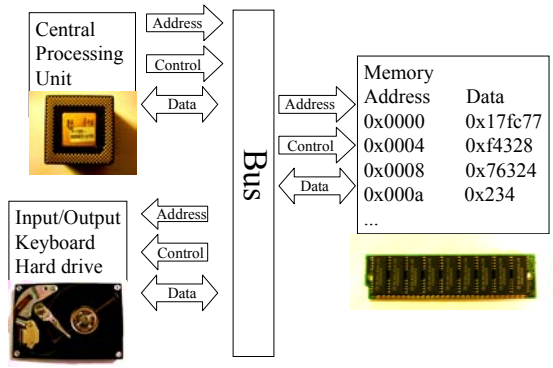
Character	Code (hex)	Code (decimal)
0	0x30	48
9	0x39	57
A	0x41	65
B	0x42	66
E	0x43	67
Z	0x5a	90
a	0x61	97
z	0x7a	122
newline	0x0a	10

Working with Bits

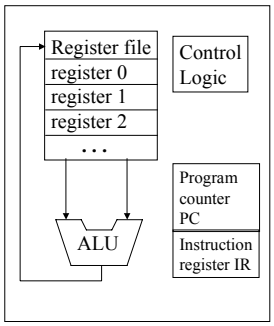
Bit-Wise Logical Operations
0110 0111 **and** 0100 1110 = 0100 0110
0110 0111 **or** 0100 1110 = 0100 0110
0110 0111 **xor** 0100 1110 = 0010 1001
complement 0110 0111 = 1001 1000

Notation in C	
and	&
or	|
xor	^
compl-ement	~

The von Neumann Architecture



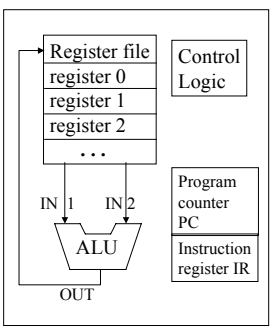
The Processor

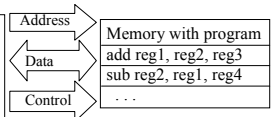


Execute instructions:
add **add** two numbers
sub **sub**tract
lw fetch data from memory,**l**oad **w**ord
sw store data in memory **s**tore **w**ord

ALU = Arithmetic Logic Unit

The Fetch-Execute Cycle



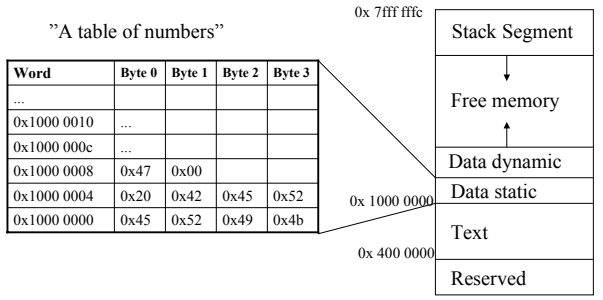


1. Put PC on address bus
2. Read instruction from memory
3. Put instruction in IR
4. Decode instruction
5. Execute
   Example: add reg1,reg2, reg3
   reg2 -> IN 1 on ALU
   reg3 -> IN 2 on ALU
   Set ALU to add
   OUT -> reg1
6. Increment PC
7. Go to 1

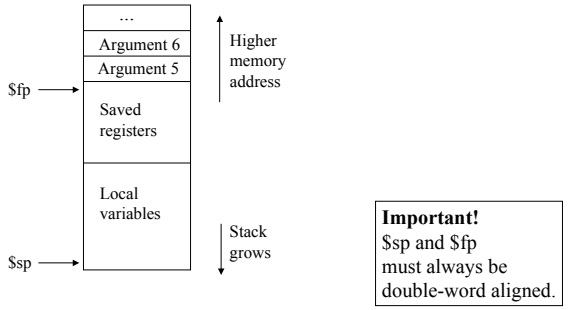
The MIPS processor

- RISC processor (few, simple instructions)
- 32 bit word length
- 32 integer registers
- 16 floating point registers
- Common in embedded systems
- Similar to SPARC (Sun) & PowerPC (Mac)

The Memory



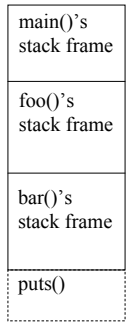
The Stack Frame



Important!
$sp and $fp must always be double-word aligned.

The Stack

- The stack is divided in **stack frames**
- Each function call creates a new stack frame
- **$sp** and **$fp** must allways be double-word aligned
- **Minimum** stack frame size is **24** bytes




```
main () {
  foo();
}

foo() {
  bar();
}

bar() {
  puts("Hello");
}
```

Register Convention

$zero	always zero
$at	reserved for assembler
$v0, $v1	return values
$a0 - $a3	parameters
$t0-$t9	temporary (caller saved)
$s0-$s7	temporary (callee saved)
$sp, $fp	stack and frame pointer
$ra	return address