lecture 9

MIPS assembly language 2

R

op	rs	rt	rd	shamt	funct
6	5	5	5	5	6

eg. add, sub

I

op	rs	rt	immediate
6	5	5	16

e.g. lw, sw, beq

J

op	address
6	26

e.g. j

Today

- more on conditional braches
- 'immediate' versions
- signed vs unsigned
- Memory
- SPIM

```
#                 if (a == b)
#                    f = g + h
#

            bne   $17, $18, Exit1
            add   $19, $20, $21
   Exit1:
```

# bne branch if not equal.

# bne has different op code than beq

Other conditional branches ?

$a \leq b$	bgt	
$a < b$	bge	?
$a \geq b$	blt	
$a > b$	ble	

These instructions don't exist.

What to do ?

"Set less than"

```
slt     $s0, $s1, $s2
```

Assigns $\$s0 = \begin{cases} 1, & \text{if } \$s1 < \$s2 \\ 0, & \text{if } \$s1 \not< \$s2 \end{cases}$

Usage:

```
    blt   $s1, $s2, Exit1

≡ { slt   $t0, $s1, $s2
  { bne   $t0, $zero, Exit1
```

How to express inequalities using "less than" and "not"?

$a < b$

$a \le b \iff \overline{b < a}$

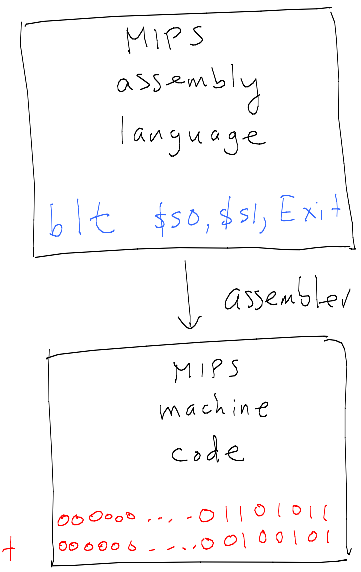
$a > b \iff b < a$

$a \ge b \iff \overline{a < b}$

Desired	Required
blt a, b, Exit	slt c, a, b bne c, $zero, Exit
ble a, b, Exit	slt c, b, a beq c, $zero, Exit
bgt a, b, Exit	slt c, b, a bne c, $zero, Exit
bge a, b, Exit	slt c, a, b beq c, $zero, Exit

* a, b, c should be registers eg. $s0, $s1, $t0

You can use "pseudo instructions" such as blt, ble, ... when you program. The assembler (SPIM) converts these to suitable real MIPS instructions.



MIPS assembly language

blt $s0, $s1, Exit

assembler

MIPS machine code

000000 ... 01101011
000000 ... 00100101

slt $t0, $s0, $s1
bne $t0, $zero, Exit

Why was MIPS designed this way?

It turns out that the benefits of having fewer instructions available (the "instruction set") outweigh the overall 'costs' of having more instructions in each program.

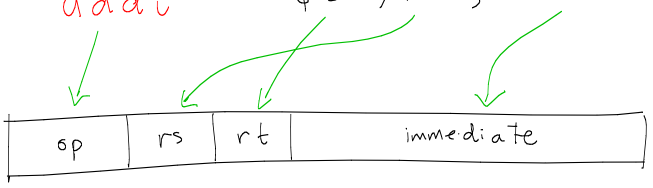
RISC - reduced instruction set computer

Today

- more on conditional braches
- 'immediate' versions
- signed vs unsigned
- Memory
- SPIM

f = h + (-13)

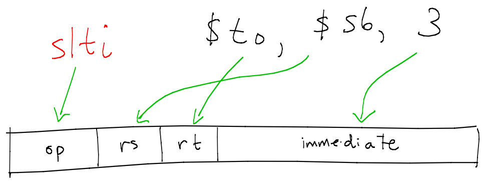
addi $s0, $s1, -13



op	rs	rt	immediate

```
#        if  (i < 3)
#            f = g + h

      slti   $t0, $s6, 3
      beq    $t0, $zero, Exit1
      add    $s0, $s1, $s2
Exit1:
```



	Signed	vs.	Unsigned
R	add		addu
	sub		subu
	slt		sltu
I	addi		addiu
	~~subi~~		~~subiu~~ → dont exist
	slti		sltiu

```
add    $s0, $s1, $s2
addu   $s0, $s1, $s2
```

Q: What is the key difference?

A: The overflow conditions.

Recall Exercises 2 Q11.

e.g.

$$\begin{array}{r} 01\sim\sim\sim \\ +\ 01\sim\sim\sim \\ \hline 1\sim\sim\sim \end{array}$$

signed: overflow

unsigned: not overflow

addi $s0, $s1, −357

↑

$[-2^{15}, 2^{15}-1]$

addiu $s0, $s1, 50000

↑

$[0, 2^{16}-1]$

```
slt    $t0, $s1, $s2
sltu   $t0, $s1, $s2
```

Signed vs. unsigned applies to registers too!

Example

Let { $s0 01~~~
      $s1 10~~~

Then slt $t0, $s0, $s1 ⇒ $t0 = 0
      sltu $t0, $s0, $s1 $t0 = 1

```
slti    $s0, $s1, −29241
sltiu   $s0, $s1, 50000
```

(also exist)

Sign extension

addi $16, $17, 3

op	rs	rt	00000000 00000011

↓

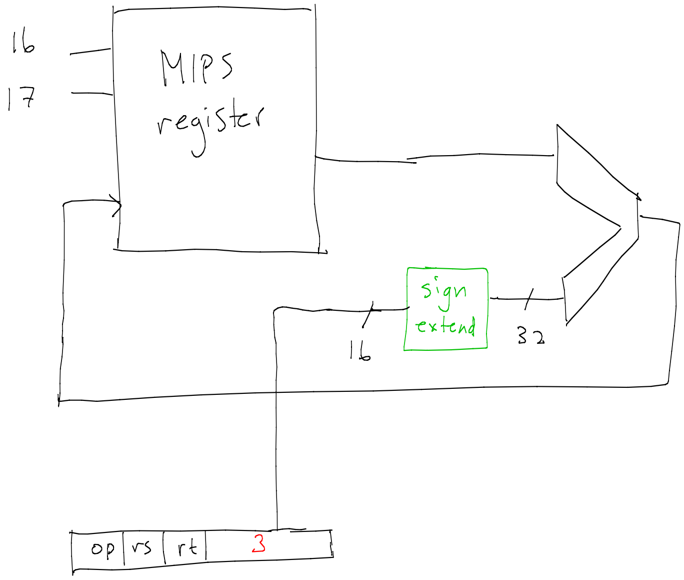
0000000000000000 00000000 00000011

addi $16, $17, −3

op	rs	rt	1111111111111101

↓

11111111111111111111111111111101



Manipulating Bits

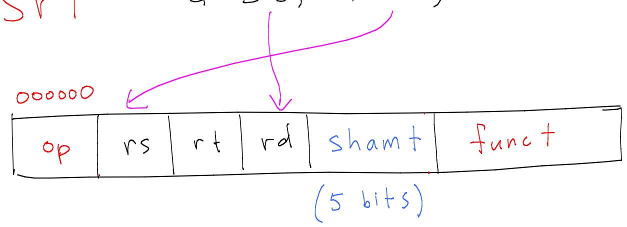
How to put some 32-bit pattern e.g. 0x3761fa93 into a register ?

lui $s0, 0x3761
ori $s0, $s0, 0xfa93

op	rs	rt	immediate

Shifting bits

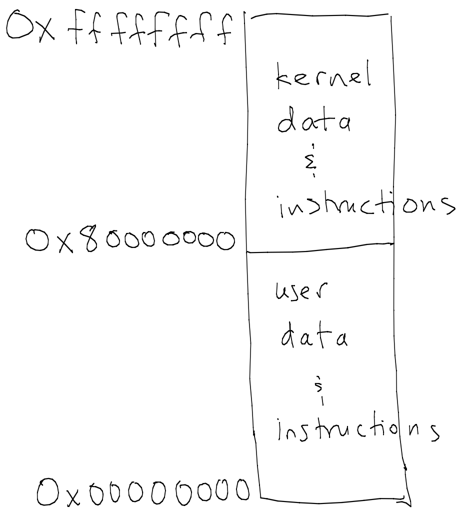
# shift left logical
sll $s0, $s1, 7
# shift right logical
srl $s0, $s0, 8

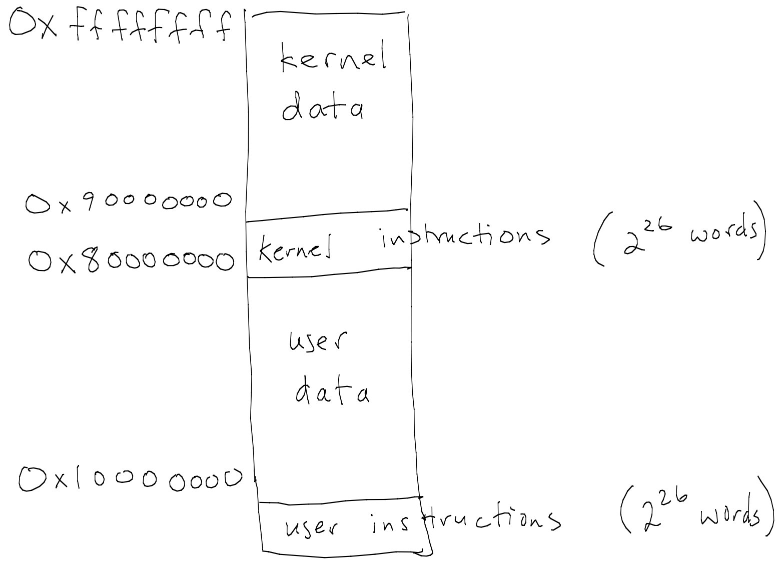


lui $s0, 0x322b	0x322b0000
srl $s0, $s0, 4	0x0322b000
srl $s0, $s0, 24	0x00000003
sll $s0, $s0, 1	0x00000006
sll $s0, $s0, 1	0x0000000c
sll $s0, $s0, 1	0x00000018

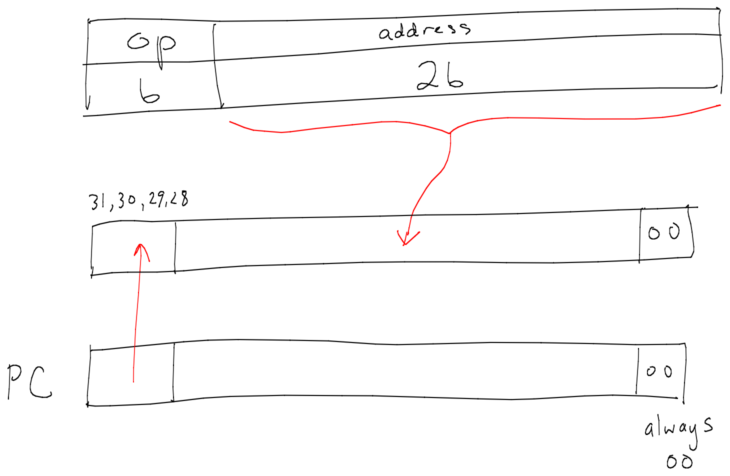
Let's take a break from MIPS instructions

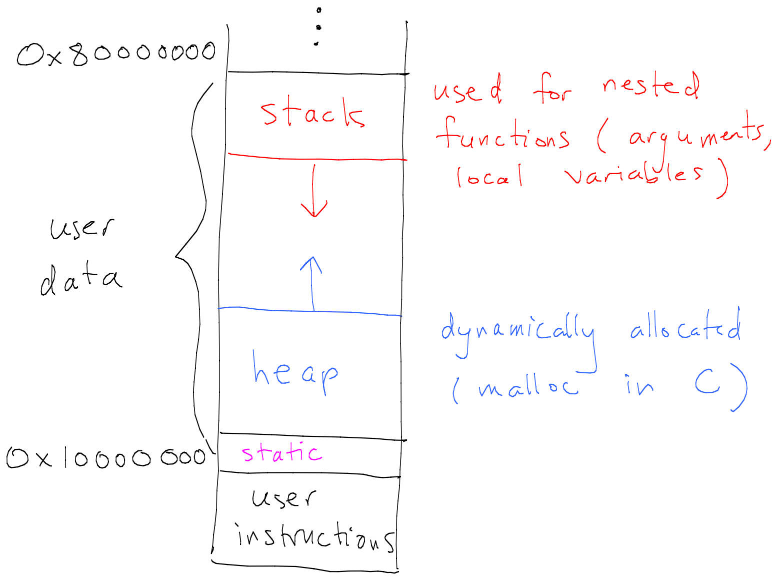
MIPS Memory



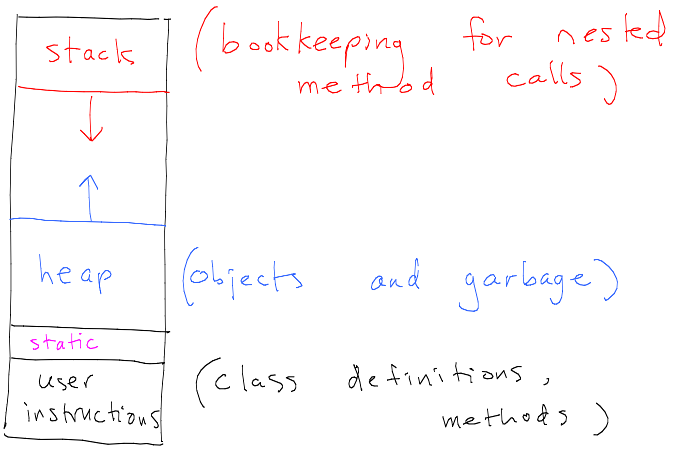


J format (jump)




0x80000000
stack
used for nested functions (arguments, local variables)
user data
heap
dynamically allocated (malloc in C)
0x10000000
static
user instructions

Java Virtual Machine (any relationships?)


stack
(bookkeeping for nested method calls)
heap
(objects and garbage)
static
user instructions
(class definitions, methods)

SPIM demo