

**CPS 104**  
**Computer Organization**  
**Lecture-6: MIPS ISA and Assembler**

**Sep. 15, 1999**

**Dietolf (Dee) Ramm**

**<http://www.cs.duke.edu/~dr/cps104.html>**

# Overview of Today's Lecture:

- **Review: The MIPS Instruction Set Architecture.**
- **The MIPS Assembly Language.**
- **MIPS Assembly Language Programming Conventions.**

★ **Reading Assignment: Chapter 3, Appendix A**

# MIPS arithmetic instructions

<u>Instruction</u>	<u>Example</u>	<u>Meaning</u>	<u>Comments</u>
add	add \$1,\$2,\$3	$\$1 = \$2 + \$3$	3 operands; exception possible
subtract	sub \$1,\$2,\$3	$\$1 = \$2 - \$3$	3 operands; exception possible
add immediate	addi \$1,\$2,100	$\$1 = \$2 + 100$	+ constant; exception possible
add unsigned	addu \$1,\$2,\$3	$\$1 = \$2 + \$3$	3 operands; no exceptions
subtract unsigned	subu \$1,\$2,\$3	$\$1 = \$2 - \$3$	3 operands; no exceptions
add imm. unsign.	addiu \$1,\$2,100	$\$1 = \$2 + 100$	+ constant; no exceptions
multiply	mult \$2,\$3	Hi, Lo = $\$2 \times \$3$	64-bit signed product
multiply unsigned	multu \$2,\$3	Hi, Lo = $\$2 \times \$3$	64-bit unsigned product
divide	div \$2,\$3	Lo = $\$2 \div \$3$ , Hi = $\$2 \bmod \$3$	Lo = quotient, Hi = remainder
divide unsigned	divu \$2,\$3	Lo = $\$2 \div \$3$ , Hi = $\$2 \bmod \$3$	Unsigned quotient & remainder
Move from Hi	mfhi \$1	$\$1 = \text{Hi}$	Used to get copy of Hi
Move from Lo	mflo \$1	$\$1 = \text{Lo}$	Used to get copy of Lo

**Which add for address arithmetic? Which add for integers?**

# Multiply / Divide

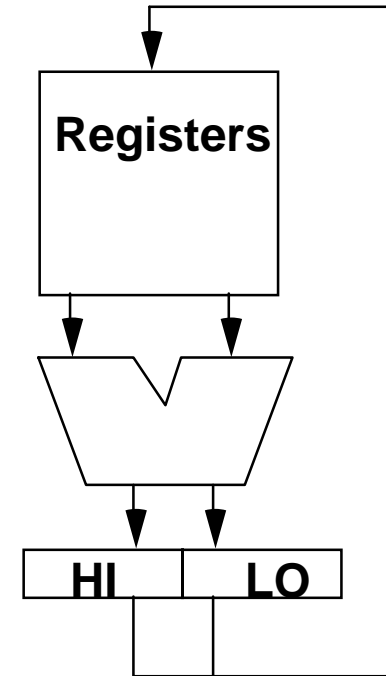
- Start multiply, divide

- mult rs, rt

- mthi rd

- mtlo rd

Move to HI or LO



- Why not Third field for destination?

(Hint: how many clock cycles for multiply or divide vs. add?)

## MIPS logical instructions

<u>Instruction</u>	<u>Example</u>	<u>Meaning</u>	<u>Comment</u>
and	and \$1,\$2,\$3	$\$1 = \$2 \& \$3$	3 reg. operands; Logical AND
or	or \$1,\$2,\$3	$\$1 = \$2   \$3$	3 reg. operands; Logical OR
xor	xor \$1,\$2,\$3	$\$1 = \$2 \oplus \$3$	3 reg. operands; Logical XOR
nor	nor \$1,\$2,\$3	$\$1 = \sim(\$2   \$3)$	3 reg. operands; Logical NOR
and immediate	andi \$1,\$2,10	$\$1 = \$2 \& 10$	Logical AND reg, constant
or immediate	ori \$1,\$2,10	$\$1 = \$2   10$	Logical OR reg, constant
xor immediate	xori \$1, \$2,10	$\$1 = \sim\$2 \& \sim 10$	Logical XOR reg, constant
shift left logical	sll \$1,\$2,10	$\$1 = \$2 \ll 10$	Shift left by constant
shift right logical	srl \$1,\$2,10	$\$1 = \$2 \gg 10$	Shift right by constant
shift right arithm.	sra \$1,\$2,10	$\$1 = \$2 \gg 10$	Shift right (sign extend)
shift left logical	sllv \$1,\$2,\$3	$\$1 = \$2 \ll \$3$	Shift left by variable
shift right logical	srlv \$1,\$2, \$3	$\$1 = \$2 \gg \$3$	Shift right by variable
shift right arithm.	srav \$1,\$2, \$3	$\$1 = \$2 \gg \$3$	Shift right arith. by variable

## MIPS data transfer instructions

### Instruction

### Comment

sw \$3, 500(\$4)

Store word, 32-bit, must be word aligned!!

sh \$3, 502(\$2)

Store half word, 16-bit, half word Aligned.

sb \$2, 41(\$3)

Store byte

lw \$1, 30(\$2)

Load word, word aligned.

lh \$1, 40(\$3)

Load halfword, half word aligned.

lhu \$1, 40(\$3)

Load halfword unsigned

lb \$1, 40(\$3)

Load byte

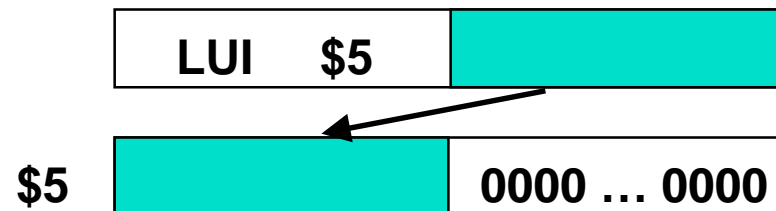
lbu \$1, 40(\$3)

Load byte unsigned

lui \$1, 40

Load Upper Immediate (16 bits shifted left by 16)

**Why need LUI?**



# MIPS Compare and Branch

○ **Compare and Branch**    **beq rs, rt, offset**    if (R[rs] == R[rt]) PC-relative branch

**bne rs, rt, offset**     $\neq$  **Compare to zero and Branch**

**blez rs, offset**    if R[rs]  $\leq$  0 then PC-relative branch

**bgtz rs, offset**     $>$

**bltz rs, offset**     $<$

**bgez rs, offset**     $\geq$

**bltzal rs, offset**    if (R[rs]  $<$  0) branch & link (into R 31)

**bgeal rs, offset**     $\geq$

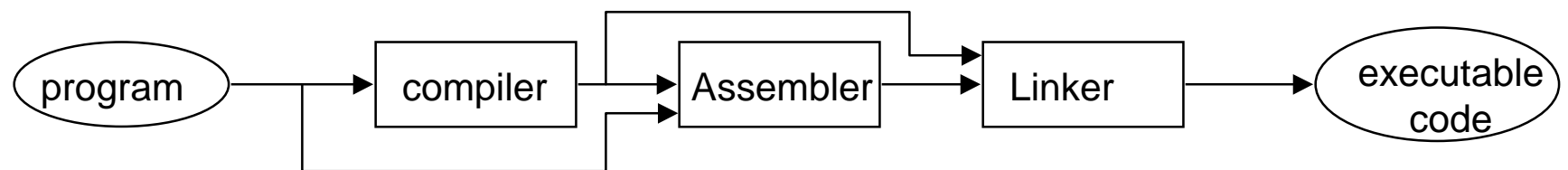
**Remaining set of compare and branch take two instructions Almost all comparisons are against zero!**

# MIPS branch, compare, jump instructions

<u>Instruction</u>	<u>Example</u>	<u>Meaning</u>
branch on equal	beq \$1,\$2,100	if ( $\$1 == \$2$ ) go to (PC+4) +100 <i>Equal test; PC relative branch</i>
branch on not eq.	bne \$1,\$2,100	if ( $\$1 \neq \$2$ ) go to (PC+4) +100 <i>Not equal test; PC relative</i>
set on less than	slt \$1,\$2,\$3	if ( $\$2 < \$3$ ) \$1=1; else \$1=0 <i>Compare less than; 2's comp.</i>
set less than imm.	slti \$1,\$2,100	if ( $\$2 < 100$ ) \$1=1; else \$1=0 <i>Compare &lt; constant; 2's comp.</i>
set less than uns.	sltu \$1,\$2,\$3	if ( $\$2 < \$3$ ) \$1=1; else \$1=0 <i>Compare less than; natural numbers</i>
set l. t. imm. uns.	sltiu \$1,\$2,100	if ( $\$2 < 100$ ) \$1=1; else \$1=0 <i>Compare &lt; constant; natural numbers</i>
jump	j 10000	go to 10000 <i>Jump to target address</i>
jump register	jr \$31	go to \$31 <i>For switch, procedure return</i>
jump and link	jal 10000	\$31 = PC + 4; go to 10000 <i>For procedure call</i>

# Assembler and Assembly Language

- Machine language is a sequence of binary words.
- Assembly language is a text representation for machine language plus extras that make assembly language programming easier (more readable too!).



# Assembly Language

- One instruction per line.
- **Numbers** are base-10 integers or Hex.
- **Identifiers**: alphanumeric, `_`, string starting in a letter or `_`
- **Labels** are identifiers starting at the beginning of a line followed by “:”
- **Comments** everything following `#` till end-of-line.
- **Instruction format**: Space and “,” separated fields.
  - `[Label:] <op> Arg1, [Arg2], [Arg3] [# comment]`
  - `[Label:] <op> arg1, offset(reg) [# comment]`
  - `.Directive [arg1], [arg2], ...`

## Assembly Language (cont.)

○ Pseudo-instructions: extending the instruction set for convenience.

○ Examples:

● `move $2, $4`

Translates to:

`add $2, $4, $0`

# `$2 = $4, (copy $4 to $2)`

● `li $8, 40`

`addi $8, $0, 40`

# `$8 = 40, (load 40 into $8)`

● `sd $4, 0($29)`

`sw $4, 0($29)`

`sw $5, 4($29)`

# `mem[$29] = $4; Mem[$29+4] = $5`

● `la $4, 0x1000056c`

`lui $4, 0x1000`

`ori $4, $4, 0x056c`

# Load address `$4 = <address>`

# Assembly Language (cont.)

- Directives: “.”<string> [arg1], [arg2] . . .
- Examples:
  - .align n # align datum on 2<sup>n</sup> byte boundary.
  - .ascii <string> # store a string in memory.
  - .asciiz <string> # store a null terminated string in memory
  - .data [address] # start a data segment.  
# [optional beginning address]
  - .text [address] # start a code segment.
  - .word w1, w2, . . . , wn # store n words in memory.

# The routine written in C

```
#include <stdio>

int main ( )
{
    int i;
    int sum = 0;
    for(i=0; i <= 100; i++) sum = sum + i*i ;
    printf("The sum from 0 .. 100 is %d\n", sum) ;
}
```

# Assembly Language Example1:

```
main:      .text
           align 2
           subi  $29,$29,32
           sw    $31,20($29)
           sd    $4,32($29)
           sw    $0,24($29)
           sw    $0,28($29)
bop:       lw    $14,28($29)
           mul   $15,$14,$14
           lw    $24,24($29)
           add   $25,$24,$15
           sw    $25,24($29)
           addi  $8,$14,1
           sw    $8,28($29)
           ble   $8,100,bop
           la    $4,str
           lw    $5,24($29)
           jal   printf
```

```
move      $2,$0
lw        $31,20($29)
addiu     $29,$29,32
jr        $31
```

```
data
align 0
```

```
str:
asciiz "The sum from 0 ..100 is %d\n"
```

# MIPS: Software conventions for Registers

0 zero constant 0

1 at reserved for assembler

2 v0 expression evaluation &

3 v1 function results

4 a0 arguments

5 a1

6 a2

7 a3

8 t0 temporary: caller saves

...

15 t7

16 s0 callee saves

...

23 s7

24 t8 temporary (cont'd)

25 t9

26 k0 reserved for OS kernel

27 k1

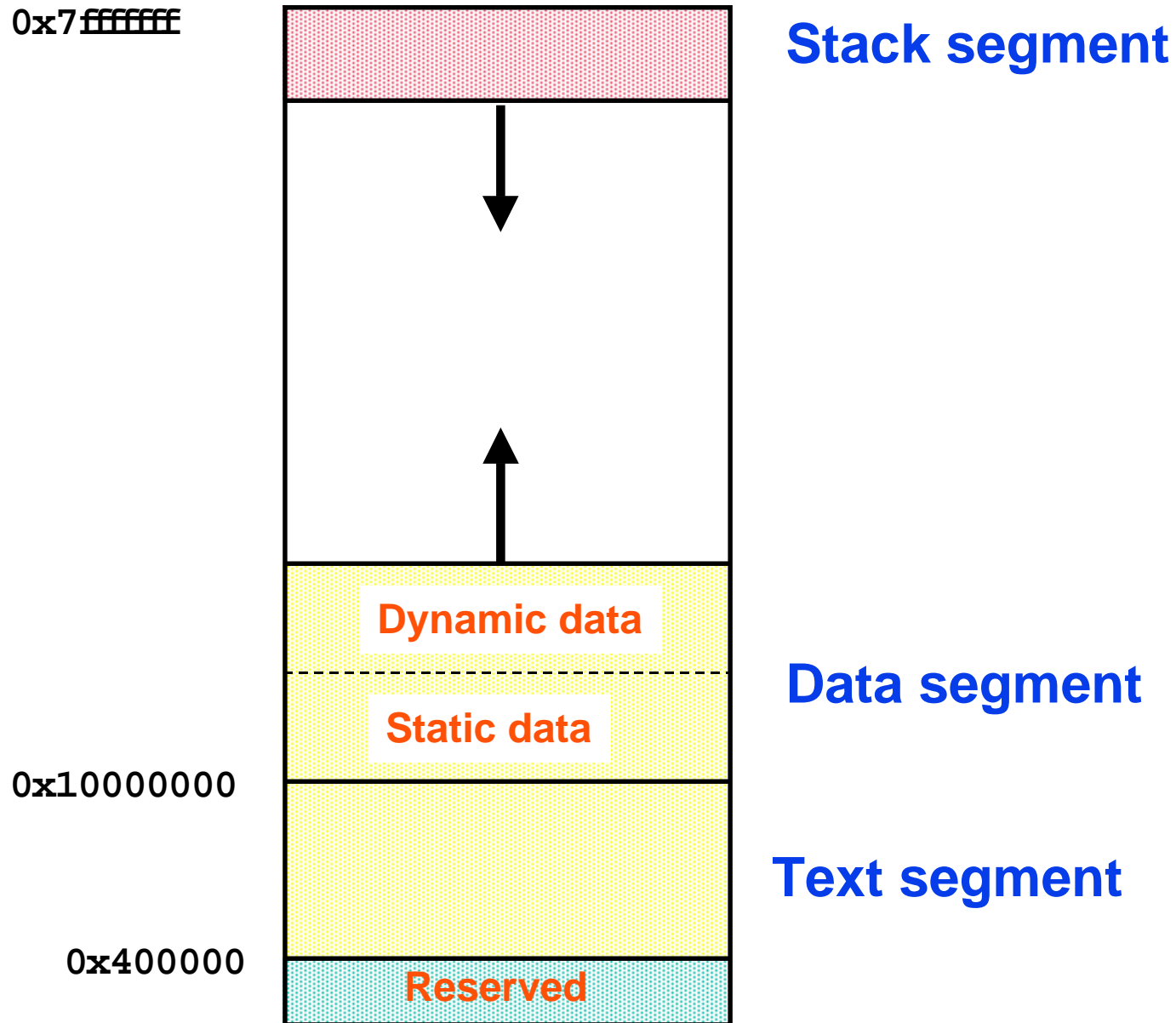
28 gp Pointer to global area

29 sp Stack pointer

30 fp frame pointer

31 ra Return Address (HW)

# Memory Layout



## Example2

```
# Example for CPS 104
# Program to add together list of 9 numbers.

        .text                # Code
        align 2
        globl main

main:   # MAIN procedure Entrance
        subi $sp, 40        # \ Push the stack
        sw   $ra, 36($sp)   # \ Save return address
        sw   $s3, 32($sp)   # \
        sw   $s2, 28($sp)   # \ > Entry Housekeeping
        sw   $s1, 24($sp)   # \ / save registers on stack
        sw   $s0, 20($sp)   # \ /
        move $v0, $0        # / initialize exit code to 0

        move $s1, $0        # \
        la   $s0, list       # \ Initialization
        la   $s2, msg        # \ /
        la   $s3, list+36    # /
```

## Example2 (cont.)

```
#                               Main code segment

again:                          # Begin main loop
    lw    $t6,0(s0)             #\
    add   $s1,$s1,$t6          #/ Actual "work"
                                # SPM IO

    li    $v0,4                 #\
    move  $a0,$s2              # > Print a string
    syscall                               #/
    li    $v0,1                 #\
    move  $a0,$s1              # > Print a number
    syscall                               #/
    li    $v0,4                 #\
    la    $a0,nh                # > Print a string (eol)
    syscall                               #/

    addiu $s0,$s0,4            #\ index update and
    bne   $s0,$s3,again       #/ end of loop
```

## Example2 (cont.)

```
#                               ExitCode
move $v0,$0                     #\
lw  $s0,20($sp)                 # \
lw  $s1,24($sp)                 #  \
lw  $s2,28($sp)                 #   \Closing Housekeeping
lw  $s3,32($sp)                 #    / restore registers
lw  $ra,36($sp)                 #   /bad return address
addiu $sp,40                     #  /Pop the stack
jr  $ra                         #/  exit(0);
end main                         # end of program
```

```
#                               Data Segment

data                             # Start of data segment
list: word 35,16,42,19,55,91,24,61,53
msg: asciiz "The sum is "
nh:  asciiz "\n"
```

# System call

- System call is used to communicate with the system and do simple I/O.
- Load system call code into Register **\$v0**
- Load arguments (if any) into registers **\$a0**, **\$a1** or **\$f12** (for floating point).
- do: **syscall**
- Results returned in registers **\$v0** or **\$f0**.

code	service	Arguments	Result	comments
1	print integer	\$a0		
2	print float	\$f12		
3	print double	\$f12		
4	print string	\$a0		(address)
5	read integer		integer in \$v0	
6	read float		float in \$f0	
7	read double		double in \$f0	
8	read string	\$a0=buffer, \$a1=length		
9	sbrk	\$a0=amount	address in \$v0	
10	exit			

## Details of the MIPS instruction set

- Register zero always has the value zero (even if you try to write it)
- Branch and jump instructions put the return address PC+4 into the link register
- All instructions change all 32 bits of the destination register (including lui, lb, lh) and all read all 32 bits of sources (add, sub, and, or, ...)
- Immediate arithmetic and logical instructions are extended as follows:
  - logical immediate are zero extended to 32 bits
  - arithmetic immediate are sign extended to 32 bits
- The data loaded by the instructions lb and lh are extended as follows:
  - lbu, lhu are zero extended
  - lb, lh are sign extended
- Overflow can occur in these arithmetic and logical instructions:
  - add, sub, addi
  - it **cannot** occur in addu, subu, addiu, and, or, xor, nor, shifts, mult, multu, div, divu

## Miscellaneous MIPS Instructions

- **break**                      A breakpoint trap occurs, transfers control to exception handler
- **syscall**                    A system trap occurs, transfers control to exception handler
- **coprocessor instrs.**      Support for floating point.
- **TLB instructions**        Support for virtual memory: discussed later
- **restore from exception**    Restores previous interrupt mask & kernel/user mode bits into status register
- **load word left/right**      Supports misaligned word loads
- **store word left/right**     Supports misaligned word stores