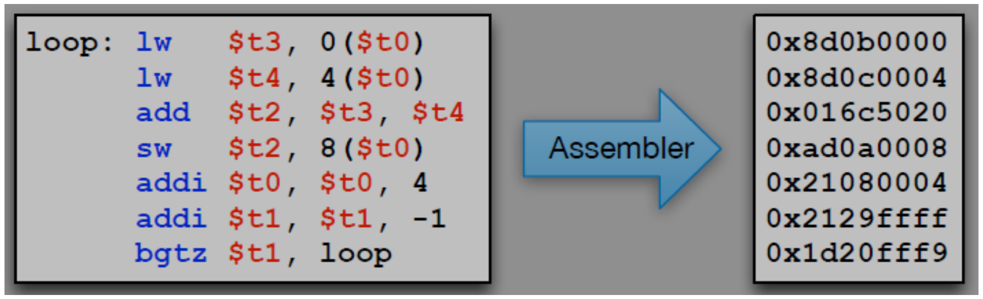


```
loop: lw   $t3, 0($t0)
      lw   $t4, 4($t0)
      add  $t2, $t3, $t4
      sw   $t2, 8($t0)
      addi $t0, $t0, 4
      addi $t1, $t1, -1
      bgtz $t1, loop
```

```
0x8d0b0000
0x8d0c0004
0x016c5020
0xad0a0008
0x21080004
0x2129ffff
0x1d20fff9
```

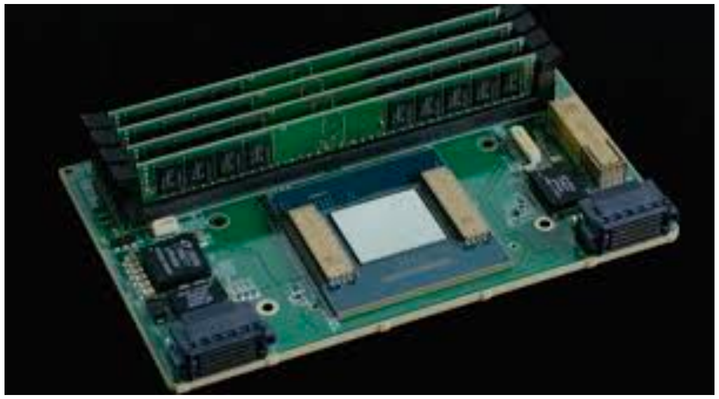
MIPS Input / Output
MIPS Instructions

CS 64: Computer Organization and Design Logic

Lecture #5

Winter 2020

Ziad Matni, Ph.D.
Dept. of Computer Science, UCSB



Gauchos Vote

EXTERNAL VICE PRESIDENT STATEWIDE AFFAIRS ASUCSB EVPSA

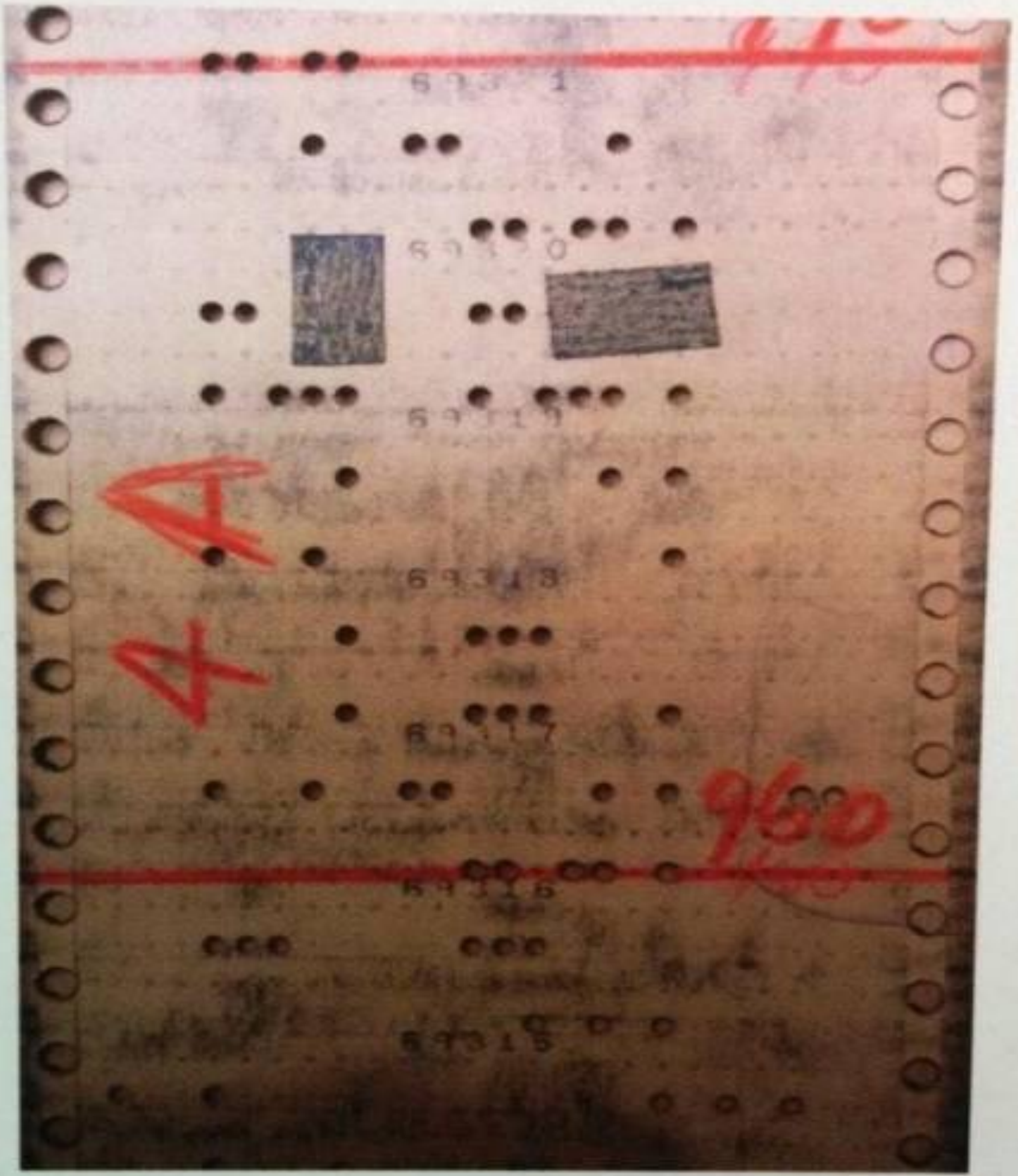
CALPIRG Students

IVote

Deadline is Feb 18th

This Week on "Didja Know Dat?!"

The "Patch"



Small corrections to the programmed sequence could be done by patching over portions of the paper tape and re-punching the holes in that section.

Image courtesy of the Smithsonian Archives Center.

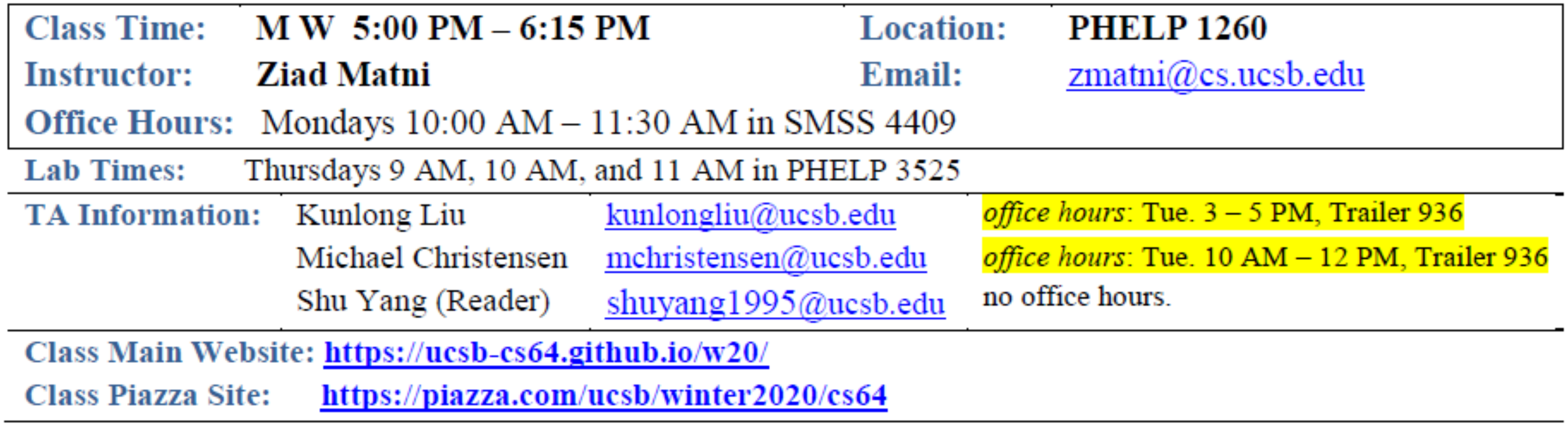
Lecture Outline

- Talking to the OS
  - Std I/O
  - Exiting
- General view of instructions in MIPS
- Operand Use
- **.data** Directives and Basic Memory Use

Administrative Stuff

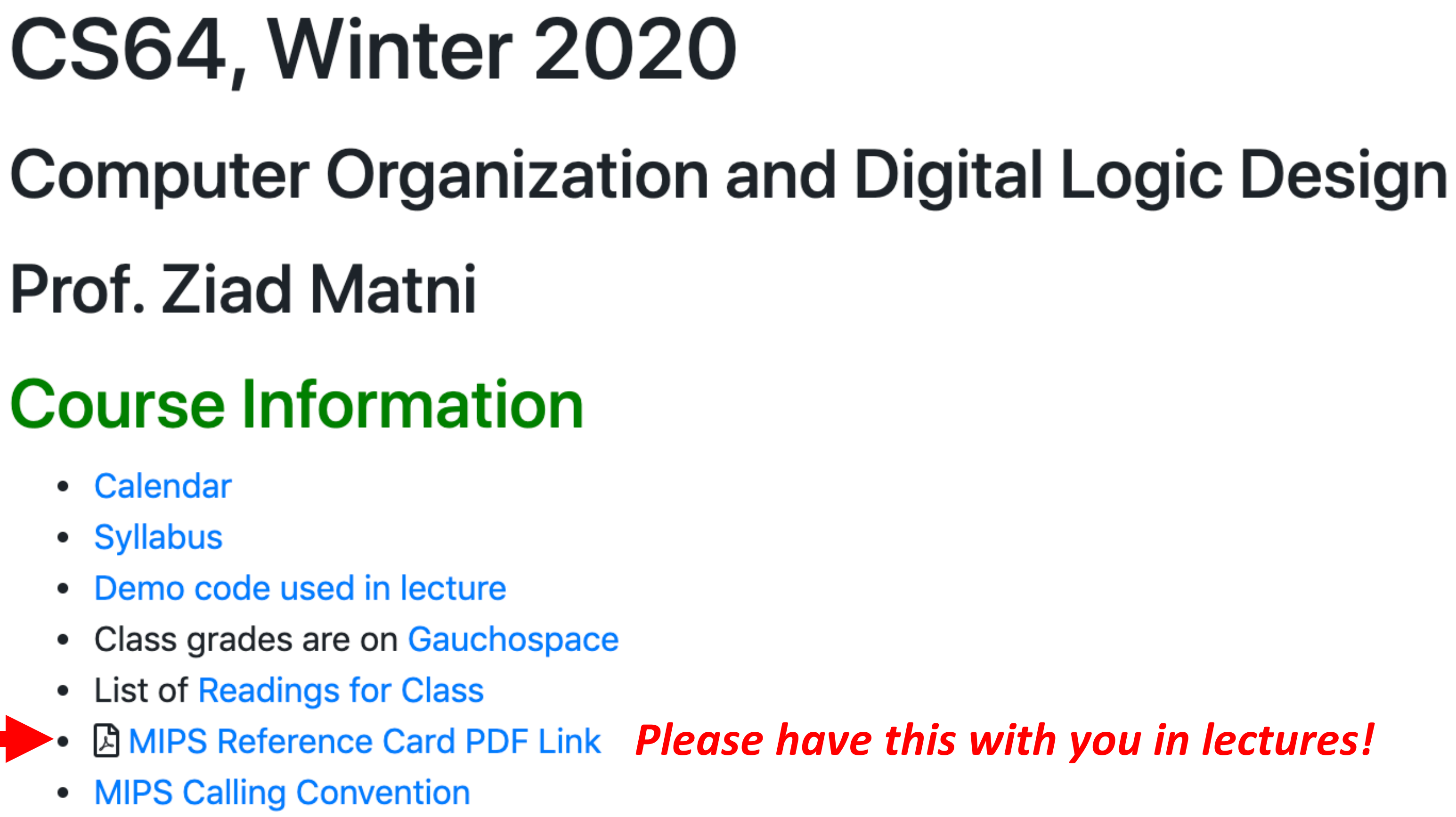
- How did Lab# 2 go?
  - Challenge level:
    HARD vs. **OK** vs. **EASY-PEASY**

- Remember, our office hours! ☺



Class Time:	**M W 5:00 PM – 6:15 PM**	**Location:**	**PHELP 1260**
Instructor:	**Ziad Matni**	**Email:**	zmatni@cs.ucsb.edu
Office Hours:	Mondays 10:00 AM – 11:30 AM in SMSS 4409		
Lab Times:	Thursdays 9 AM, 10 AM, and 11 AM in PHELP 3525		
TA Information:	Kunlong Liu	kunlongliu@ucsb.edu	*office hours*: Tue. 3 – 5 PM, Trailer 936
	Michael Christensen	mchristensen@ucsb.edu	*office hours*: Tue. 10 AM – 12 PM, Trailer 936
	Shu Yang (Reader)	shuyang1995@ucsb.edu	no office hours.
Class Main Website:	**https://ucsb-cs64.github.io/w20/**		
Class Piazza Site:	**https://piazza.com/ucsb/winter2020/cs64**		

MIPS Reference Card



CS64, Winter 2020

Computer Organization and Digital Logic Design

Prof. Ziad Matni

Course Information

- Calendar
- Syllabus
- Demo code used in lecture
- Class grades are on Gauchospace
- List of Readings for Class
- MIPS Reference Card PDF Link ***Please have this with you in lectures!***
- MIPS Calling Convention

Any Questions From Last Lecture?

Printing an Integer using **syscall**

```
# Main program
li $t0, 5
li $t1, 7
add $t3, $t0, $t1

# Print the integer that’s in $t3
# to std.output, so make $v0 = 1
li $v0, 1
move $a0, $t3
syscall
```

What About *Getting* an Input (via Std In)?

```
# Get an integer value from user
# Make $v0 = 5
li $v0, 5
syscall

# Your new input int is now in $v0
# You can move it around and compute with it
move $t0, $v0
sll $t0, $t0, 2    # Multiply it by 4
add $t0, $t0, $t0  # Add it to itself... etc...
```

Augmenting with Exiting

```
.text      # We always have to have this starting line
# Main program
main:
     li $t0, 5
     li $t1, 7
     add $t3, $t0, $t1
# Print an integer to std.output (so make $v0 = 1)
     li $v0, 1
     move $a0, $t3
     syscall
# End program
     li $v0, 10
     syscall
```

The Proper Format of an Assembly Program

```
.data
     …
.text
main:
     …
     …
     …

     li $v0, 10
     syscall
```

.data is where we declare variables in memory (not in registers)

All that follows **.text** are the program instructions

main: is an "instruction label" – necessary for ***spim*** to know that it should begin execution here.

These lines are always needed to tell the emulator (***spim***) that the program has finished

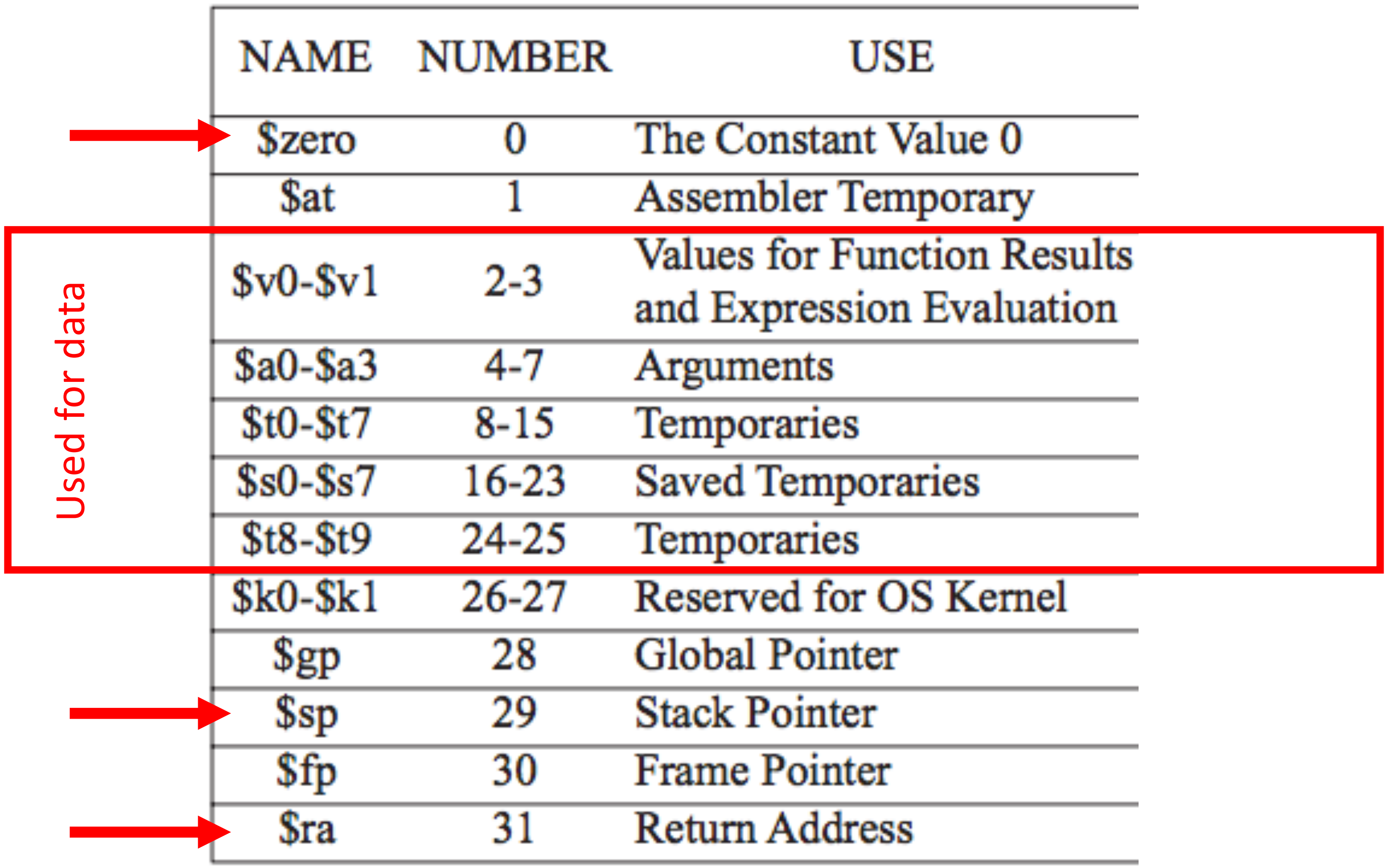
Printing Strings using **syscall**

```
.data        # This defines a value in memory (not in a register)
name: .asciiz "Porcupine Tree\n"

.text
main:
# Print string (not an int!!) to std.output
# Making $v0 = 4 tells syscall to expect a string to be printed...
      li $v0, 4
# Since a string is an array of characters,
# we load the address of that array into $a0
      la $a0, name
      syscall

# End program
      li $v0, 10
      syscall
```

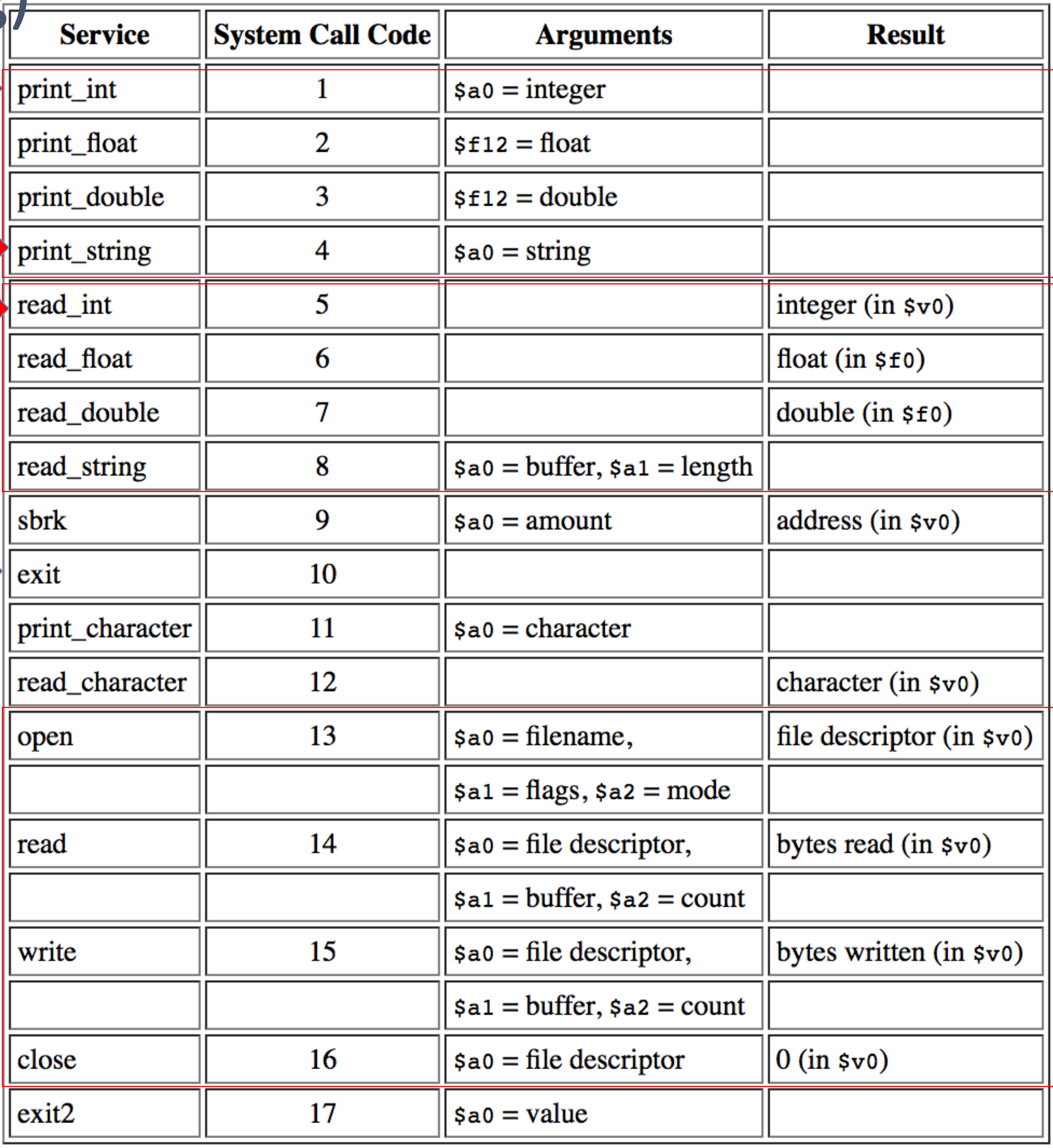
Ok… So About Those Registers
MIPS has 32 registers, each is 32 bits


Used for data

NAME	NUMBER	USE
$zero	0	The Constant Value 0
$at	1	Assembler Temporary
$v0-$v1	2-3	Values for Function Results and Expression Evaluation
$a0-$a3	4-7	Arguments
$t0-$t7	8-15	Temporaries
$s0-$s7	16-23	Saved Temporaries
$t8-$t9	24-25	Temporaries
$k0-$k1	26-27	Reserved for OS Kernel
$gp	28	Global Pointer
$sp	29	Stack Pointer
$fp	30	Frame Pointer
$ra	31	Return Address

MIPS System Services (Codes for *$v0* when *syscall'* ing)

Examples of what we'll be using in CS64



Service	System Call Code	Arguments	Result
print_int	1	`$a0` = integer	
print_float	2	`$f12` = float	
print_double	3	`$f12` = double	
print_string	4	`$a0` = string	
read_int	5		integer (in `$v0`)
read_float	6		float (in `$f0`)
read_double	7		double (in `$f0`)
read_string	8	`$a0` = buffer, `$a1` = length	
sbrk	9	`$a0` = amount	address (in `$v0`)
exit	10		
print_character	11	`$a0` = character	
read_character	12		character (in `$v0`)
open	13	`$a0` = filename,	file descriptor (in `$v0`)
		`$a1` = flags, `$a2` = mode	
read	14	`$a0` = file descriptor,	bytes read (in `$v0`)
		`$a1` = buffer, `$a2` = count	
write	15	`$a0` = file descriptor,	bytes written (in `$v0`)
		`$a1` = buffer, `$a2` = count	
close	16	`$a0` = file descriptor	0 (in `$v0`)
exit2	17	`$a0` = value	

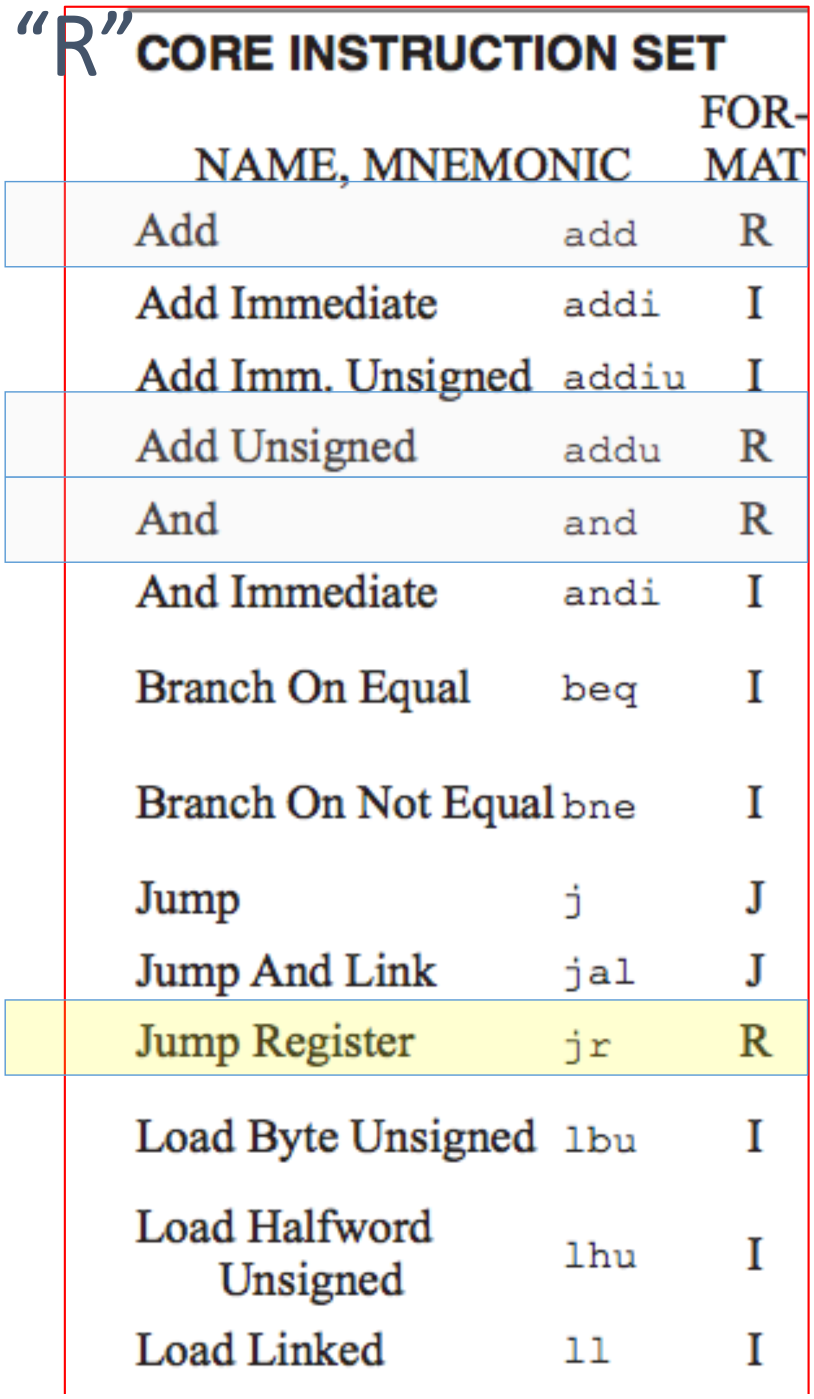
stdout

stdin

File I/O

List of all Core Instructions in MIPS

“R”

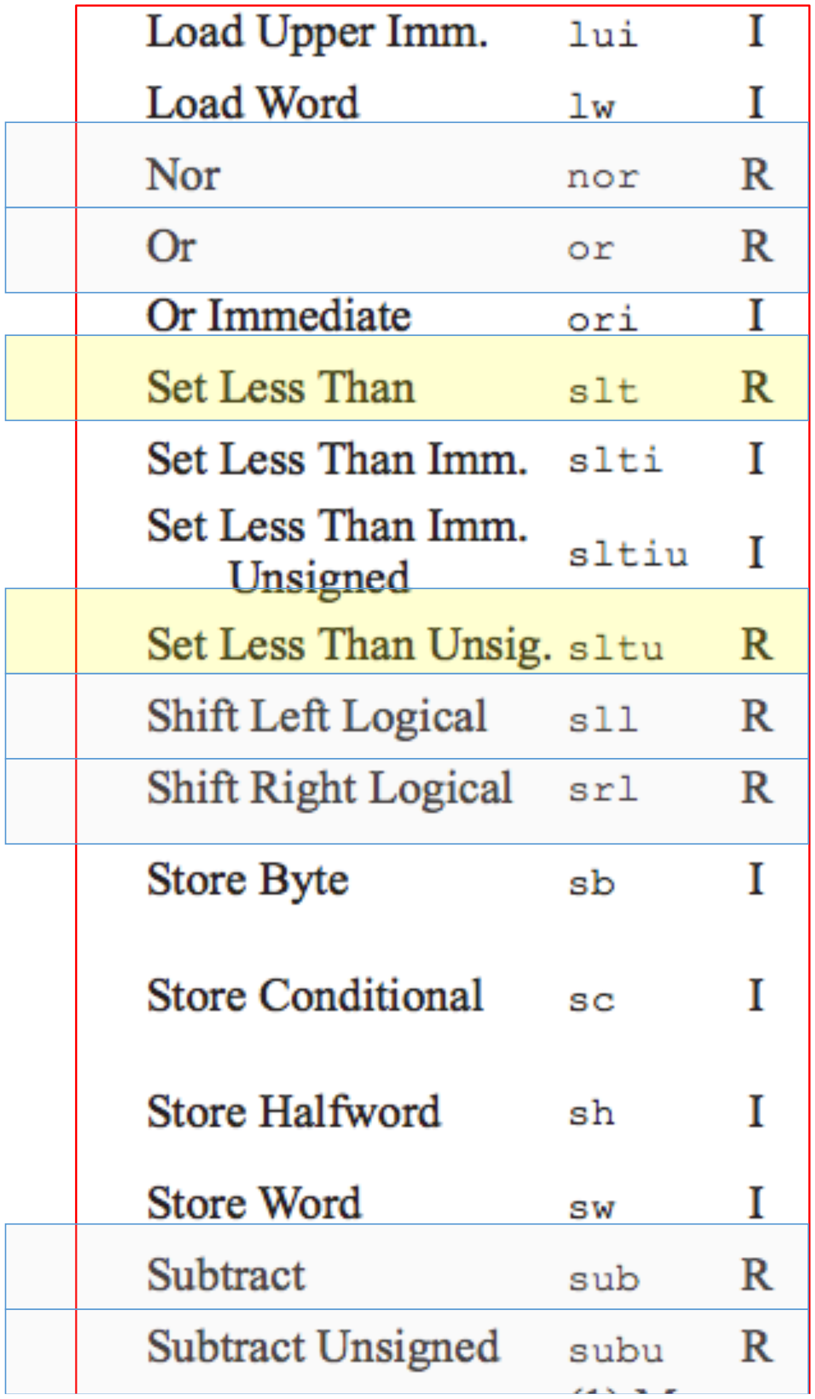


CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT
Add	add	R
Add Immediate	addi	I
Add Imm. Unsigned	addiu	I
Add Unsigned	addu	R
And	and	R
And Immediate	andi	I
Branch On Equal	beq	I
Branch On Not Equal	bne	I
Jump	j	J
Jump And Link	jal	J
Jump Register	jr	R
Load Byte Unsigned	lbu	I
Load Halfword Unsigned	lhu	I
Load Linked	ll	I

Arithmetic

Branching



Load Upper Imm.	lui	I
Load Word	lw	I
Nor	nor	R
Or	or	R
Or Immediate	ori	I
Set Less Than	slt	R
Set Less Than Imm.	slti	I
Set Less Than Imm. Unsigned	sltiu	I
Set Less Than Unsig.	sltu	R
Shift Left Logical	sll	R
Shift Right Logical	srl	R
Store Byte	sb	I
Store Conditional	sc	I
Store Halfword	sh	I
Store Word	sw	I
Subtract	sub	R
Subtract Unsigned	subu	R

R-Type Syntax

<op>   <rd>, <rs>, <rt>

op : operation

rd : register destination

rs : register source

rt : register target

Examples:

```
add $s0, $t0, $t2
```

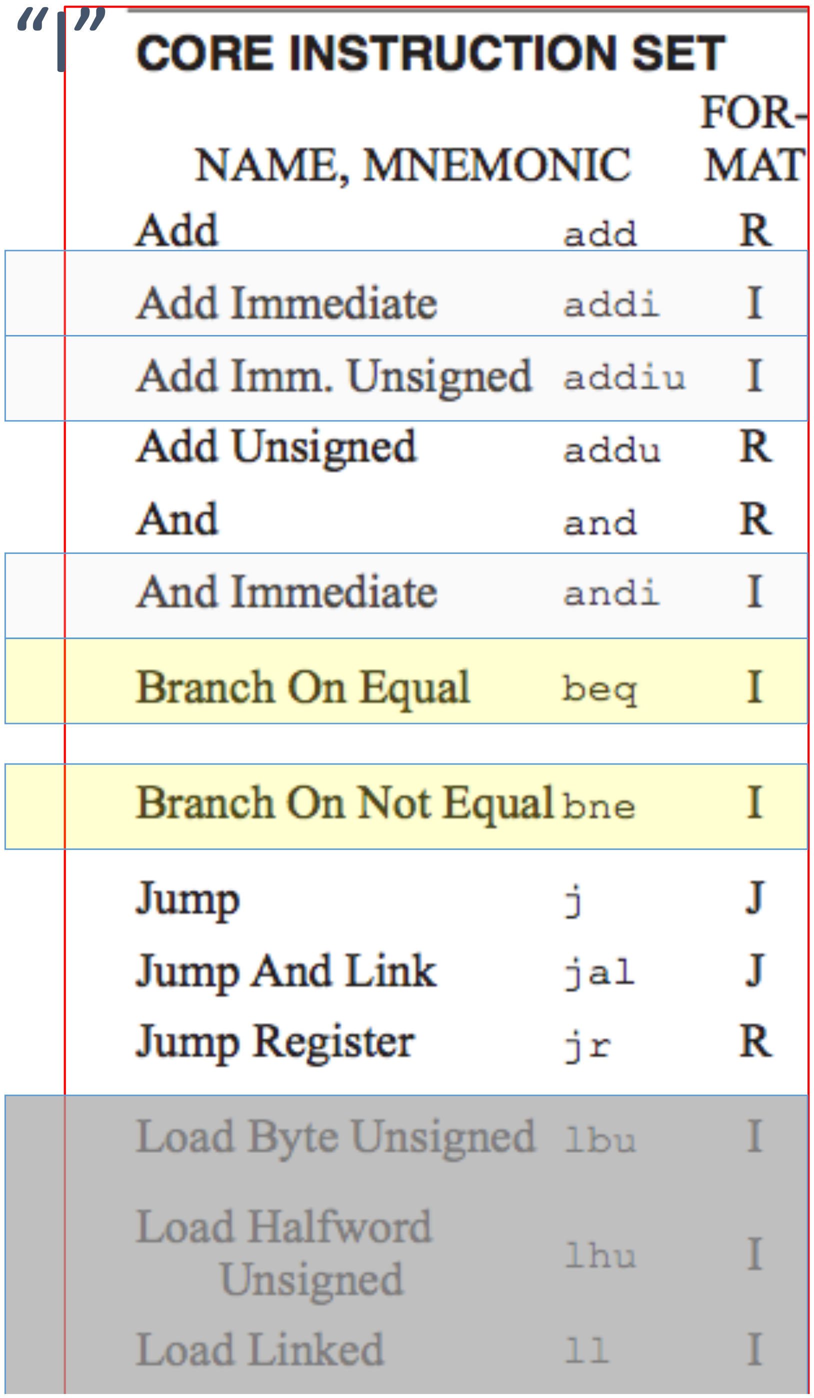
Add ($t0 + $t2) then store in reg. $s0

```
sub $t3, $t4, $t5
```

Subtract ($t4 – $t5) then store in reg. $t3

List of all Core Instructions in MIPS

“I”



CORE INSTRUCTION SET

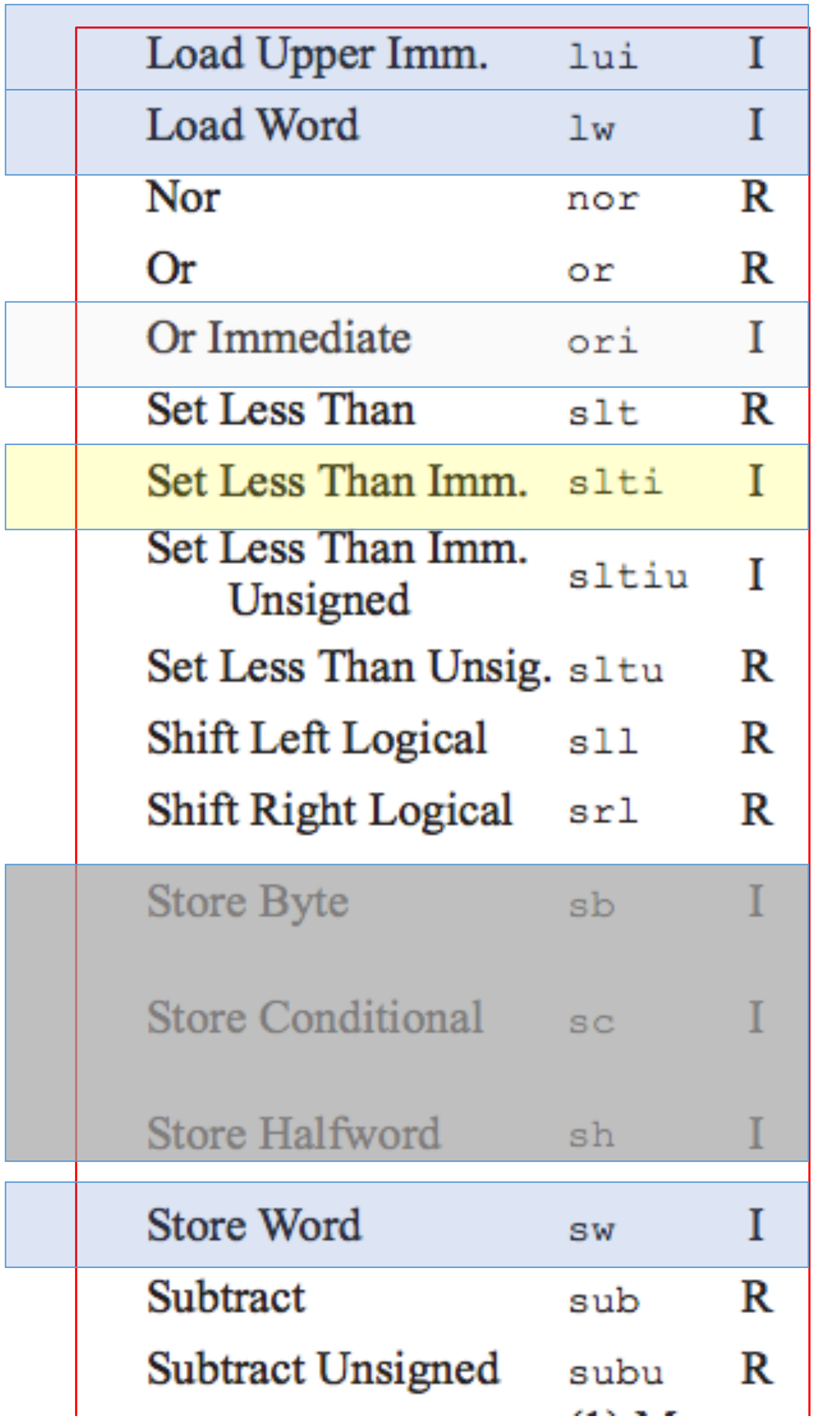
NAME, MNEMONIC		FOR-MAT
Add	add	R
Add Immediate	addi	I
Add Imm. Unsigned	addiu	I
Add Unsigned	addu	R
And	and	R
And Immediate	andi	I
Branch On Equal	beq	I
Branch On Not Equal	bne	I
Jump	j	J
Jump And Link	jal	J
Jump Register	jr	R
Load Byte Unsigned	lbu	I
Load Halfword Unsigned	lhu	I
Load Linked	ll	I

Arithmetic

Branching

Memory

Not for CS64



Load Upper Imm.	lui	I
Load Word	lw	I
Nor	nor	R
Or	or	R
Or Immediate	ori	I
Set Less Than	slt	R
Set Less Than Imm.	slti	I
Set Less Than Imm. Unsigned	sltiu	I
Set Less Than Unsig.	sltu	R
Shift Left Logical	sll	R
Shift Right Logical	srl	R
Store Byte	sb	I
Store Conditional	sc	I
Store Halfword	sh	I
Store Word	sw	I
Subtract	sub	R
Subtract Unsigned	subu	R

I-Type Syntax

<op>   <rt>, <rs>, immed

op : operation

rs : register source

rt : register target

Examples:

```
addi $s0, $t0, 33
```

Add ($t0 + 33) then store in reg. $s0

```
ori $t3, $t4, 0
```

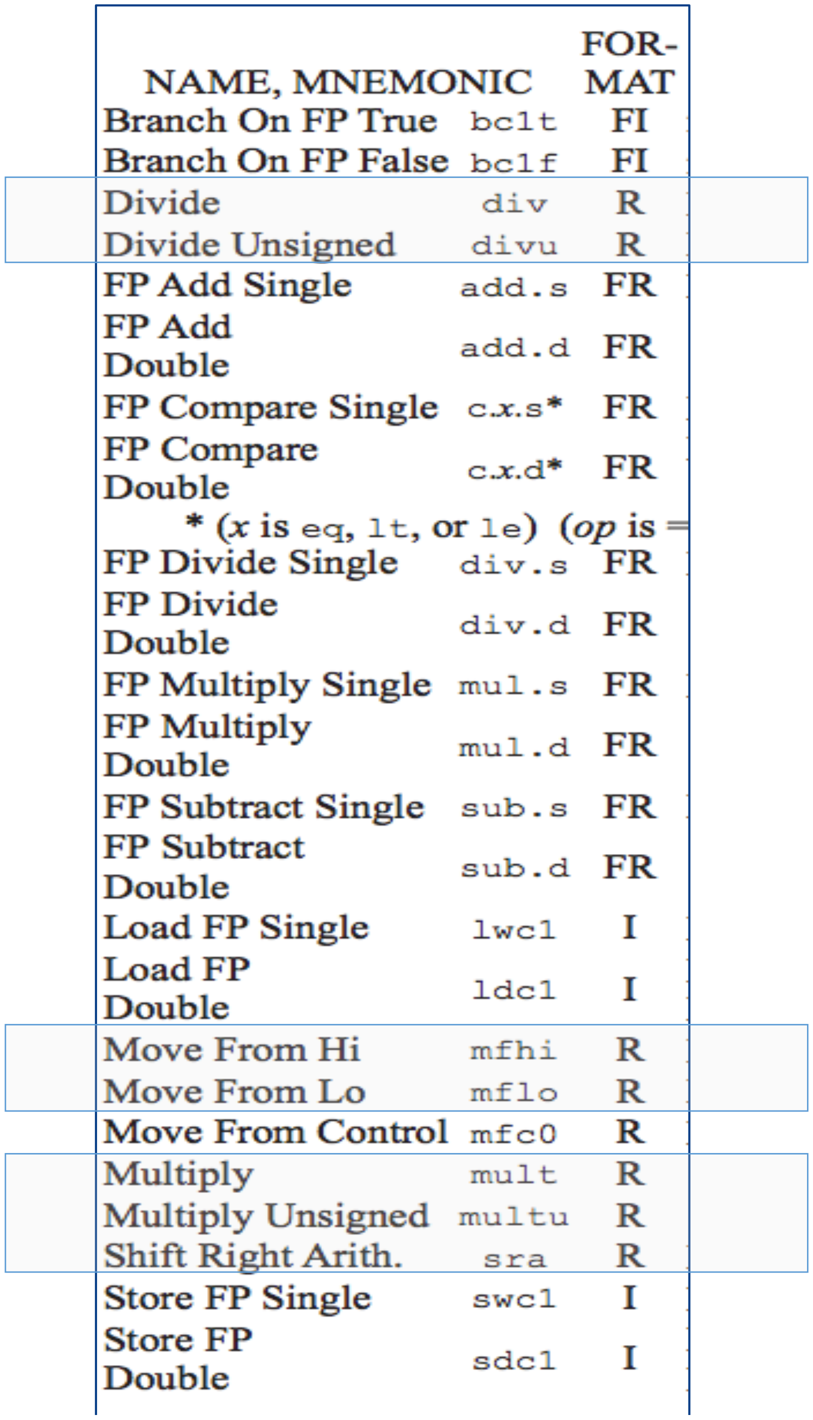
Logic OR ($t4 with 0) then store in reg. $t3

Note: this last one has the effect of just moving $t4 value into $t3

List of the Arithmetic Core Instructions in MIPS

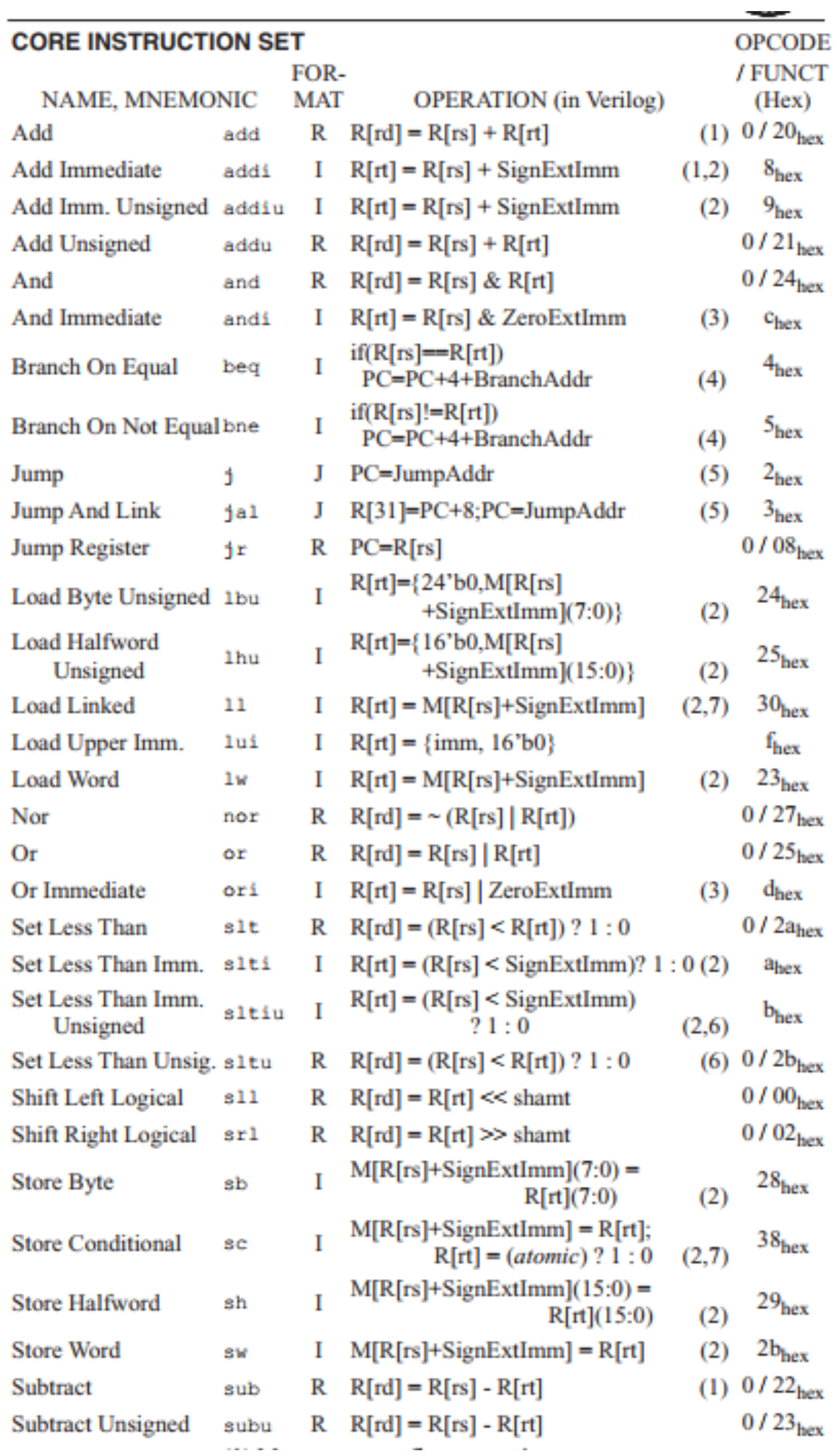
Mostly used in CS64

You are not responsible for the rest of them



NAME, MNEMONIC		FOR-MAT
Branch On FP True	bc1t	FI
Branch On FP False	bc1f	FI
Divide	div	R
Divide Unsigned	divu	R
FP Add Single	add.s	FR
FP Add Double	add.d	FR
FP Compare Single	c.*x*.s*	FR
FP Compare Double	c.*x*.d*	FR
* (*x* is eq, lt, or le) (*op* is =		
FP Divide Single	div.s	FR
FP Divide Double	div.d	FR
FP Multiply Single	mul.s	FR
FP Multiply Double	mul.d	FR
FP Subtract Single	sub.s	FR
FP Subtract Double	sub.d	FR
Load FP Single	lwc1	I
Load FP Double	ldc1	I
Move From Hi	mfhi	R
Move From Lo	mflo	R
Move From Control	mfc0	R
Multiply	mult	R
Multiply Unsigned	multu	R
Shift Right Arith.	sra	R
Store FP Single	swc1	I
Store FP Double	sdc1	I

Bring out your MIPS Reference Cards!



CORE INSTRUCTION SET

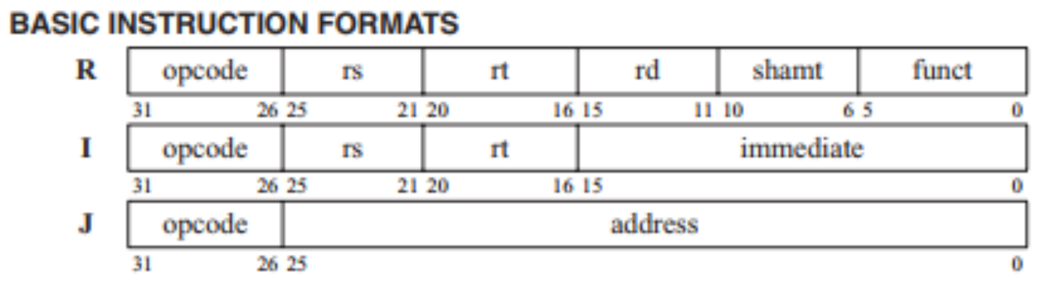
NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		0 / 08_{hex}
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27_{hex}
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25_{hex}
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / $2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00_{hex}
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		0 / 02_{hex}
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22_{hex}
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23_{hex}

NOTE THE FOLLOWING:

1. Instruction Format Types:
   R vs **I** vs **J**

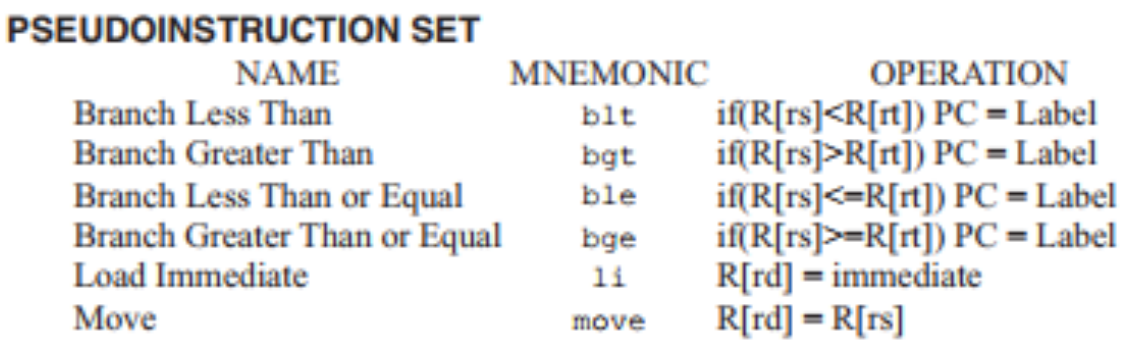
2. OPCODE/FUNCT (Hex)

BASIC INSTRUCTION FORMATS

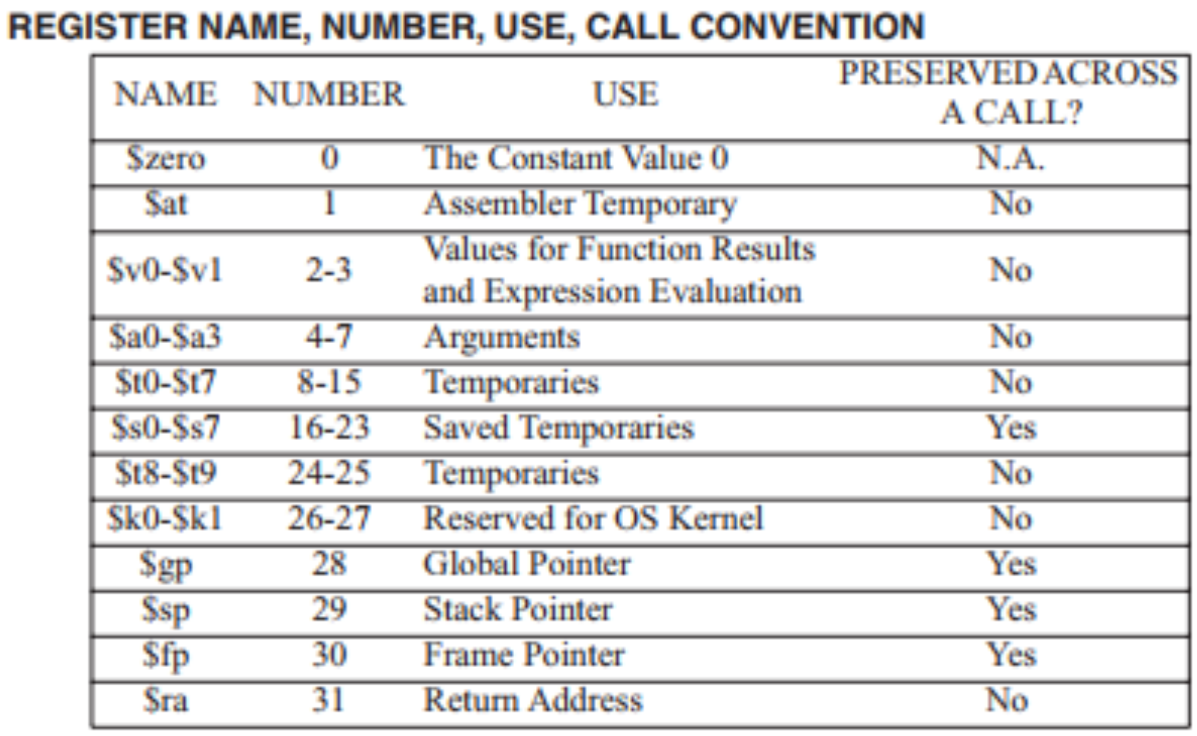


3. Instruction formats:
   Where the actual bits go
   (more on that in a later lecture)

NOTE THE FOLLOWING:


PSEUDOINSTRUCTION SET

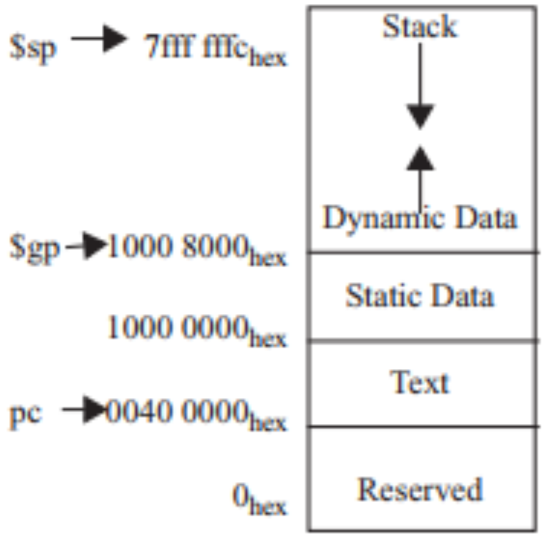
NAME	MNEMONIC	OPERATION
Branch Less Than	`blt`	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	`bgt`	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	`ble`	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	`bge`	if(R[rs]>=R[rt]) PC = Label
Load Immediate	`li`	R[rd] = immediate
Move	`move`	R[rd] = R[rs]


REGISTER NAME, NUMBER, USE, CALL CONVENTION

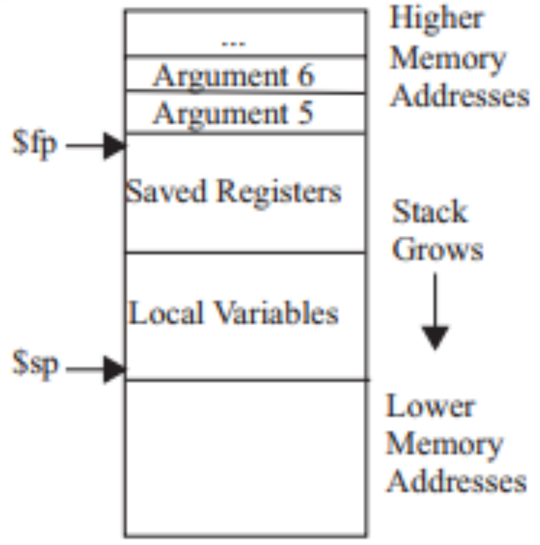
NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	No

1. Pseudo-Instructions
   - There are more of these, but in CS64, you are ONLY allowed to use these + **la**
2. Registers and their numbers
3. Registers and their uses
4. Registers and their calling convention
   - A LOT more on that later…

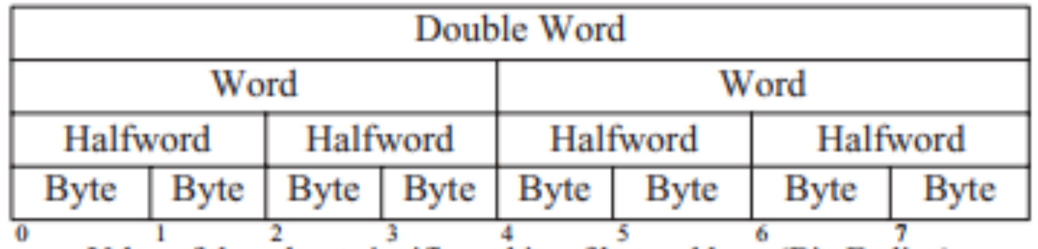
MEMORY ALLOCATION



STACK FRAME



DATA ALIGNMENT



Value of three least significant bits of byte address (Big Endian)

SIZE PREFIXES (10^x for Disk, Communication; 2^x for Memory)

SIZE	PRE-FIX	SIZE	PRE-FIX	SIZE	PRE-FIX	SIZE	PRE-FIX
10^3, 2^{10}	Kilo-	10^{15}, 2^{50}	Peta-	10^{-3}	milli-	10^{-15}	femto-
10^6, 2^{20}	Mega-	10^{18}, 2^{60}	Exa-	10^{-6}	micro-	10^{-18}	atto-
10^9, 2^{30}	Giga-	10^{21}, 2^{70}	Zetta-	10^{-9}	nano-	10^{-21}	zepto-
10^{12}, 2^{40}	Tera-	10^{24}, 2^{80}	Yotta-	10^{-12}	pico-	10^{-24}	yocto-

The symbol for each prefix is just its first letter, except μ is used for micro.

NOTE THE FOLLOWING:

1. This is only part of the 2nd page that you need to know

Bring Out Your MIPS Reference Cards!

Look for the following instructions:

- `nor`
- `addi`
- `beq`
- `move`

Tell me everything you can about them, based on what you see on the Ref Card!

The **move** Instruction…

… is suspicious…

- The move instruction does not actually show up in SPIM!

- It is a ***pseudo-instruction***
- It's easy for us to use, but it's actually a "macro" of another actual instruction

ORIGINAL:
```
move $a0, $t3
```
ACTUAL:
```
addu $a0, $zero, $t3
# what's addu? what's $zero?
```

Pseudo-instructions

- Instructions that are NOT core to the CPU
- They're "macros" of other actual instructions
- Often they are slower than core instructions
  - But usually easier to use in a program than the alternative
  - A little bit of High Level Language concept at play…

```
    li $t0, C
```

Is a macro for:

```
    lui $t0, C_hi
    ori $t0, $t0, C_lo
```

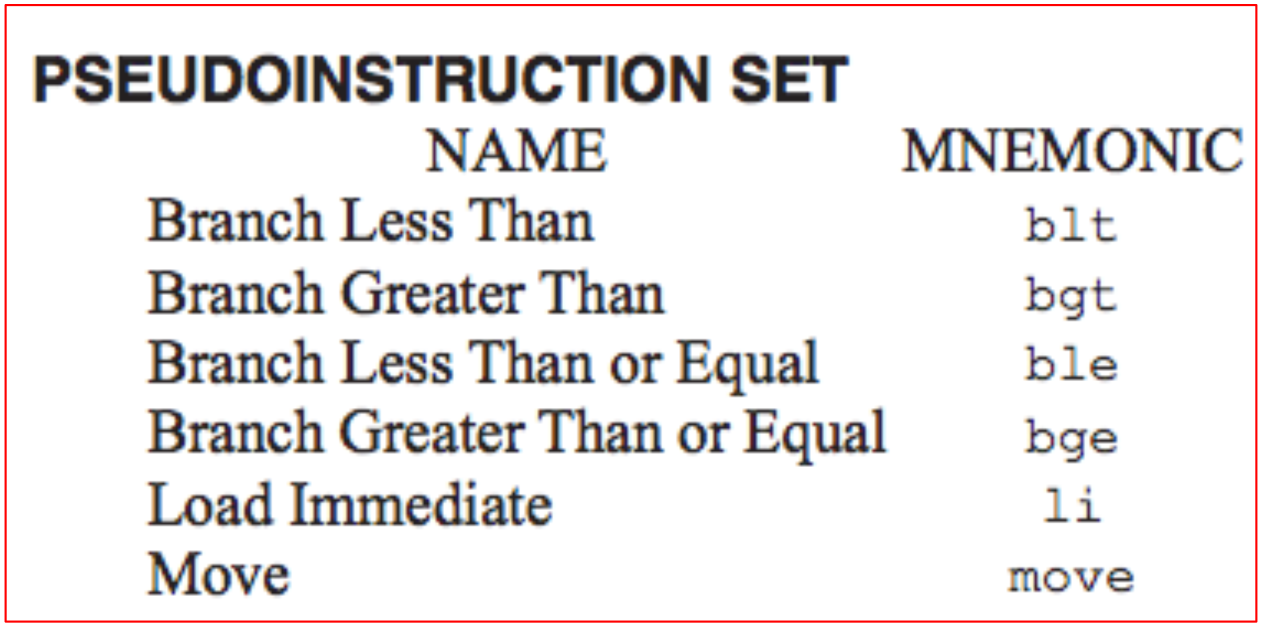
```
    move $t0, $t1
```

Is a macro for:

```
    addu $t0, $zero, $t1
```

https://github.com/MIPT-ILab/mipt-mips/wiki/MIPS-pseudo-instructions has more examples

List of all PsuedoInstructions in MIPS *That You Are Allowed to Use in CS64!!!*



PSEUDOINSTRUCTION SET

NAME	MNEMONIC
Branch Less Than	blt
Branch Greater Than	bgt
Branch Less Than or Equal	ble
Branch Greater Than or Equal	bge
Load Immediate	li
Move	move

plus this one → **Load Address** **la**

REMEMBER: USE YOUR "MIPS REFERENCE CARD" FOUND ON THE CLASS WEBSITE!!!

YOUR TO-DOs

- Do readings!
  - Check syllabus for details!
- Review ALL the demo codes
  - Available via the class website
- Work on Assignment #3

</LECTURE>