

CS 2340 – Computer Architecture

7 Logical Operations, Shifters, Machine Language - Part 1
Dr. Alice Wang

Why did the programmer quit his job? Because he didn't get arrays.

Housekeeping

- I checked the Testing Center and only 53% students registered for Exam 1, 45% for Exam 2
- If you do not reserve your seat you will not be able to take the exam and I cannot do anything about it, so do not email me if you cannot take an exam because you failed to reserve your seat.
- There will be no makeup exams under normal circumstances.

Housekeeping

- Exam Reviews have been scheduled at the TI auditorium
 - Will cover practice exam questions
 - It will also be on MS Teams and recorded
- Exam 1 Review - Thurs, Oct 2, 12-1pm
- Exam 2 Review - Mon, Nov 3, 9-10am
- Exam 2 Review - Fri, Dec 5, 5:30-6:30pm

Review

Last time

- Arrays: Data and Strings
- Conditional operations: branch, set, jump
- If-the-else, For-loops, While-loops

Today:

- Logical operations
- Shifters
- Part 1 - Machine coding

Logical operations

What are MIPS Logical Operations?

- An operation that acts on binary numbers to produce a result according to the laws of Boolean logic.
- Examples: AND, OR, NOT, XOR

What are Logical / Boolean operations used for?

- **Bit Manipulation:** Essential for setting, clearing, and toggling specific bits in registers.
- **Control Flow:** Used in conditional statements and loops to control program execution.
- **Hardware Interaction:** Crucial for interacting with hardware components and performing low-level tasks.

Logical: AND operation



and

/an(d),(ə)n(d)/

noun

ELECTRONICS

a Boolean operator which gives the value one if and only if all the operands are one, and otherwise has a value of zero.

- a circuit which produces an output signal only when signals are received simultaneously through all input connections.

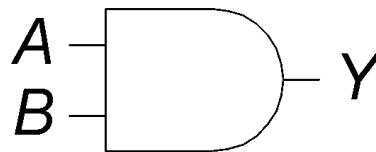
noun: **AND gate**

- AND is a fundamental Boolean operator

and \$t0, \$t1, \$t2

- Useful to “mask” bits in a word

AND



$$Y = AB$$

A	B	Y
0	0	
0	1	
1	0	
1	1	

What does it mean to “mask” bits



mask

/mask/

noun

noun: **mask**; plural noun: **masks**; noun: **masque**; plural noun: **masques**

1. a covering for all or part of the face, worn as a disguise, or to amuse or terrify other people.

Similar:

disguise

veil

false face

domino

stocking mask

fancy dress



6. PHOTOGRAPHY

a piece of something, such as a card, used to cover a part of an image that is not required when exposing a print.

7. ELECTRONICS

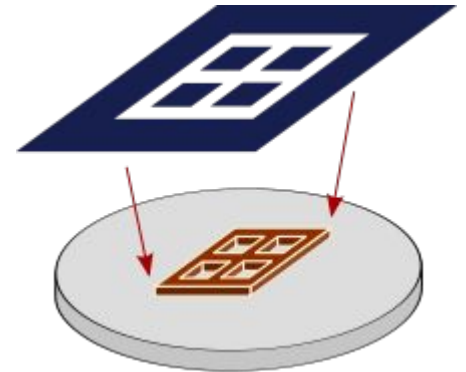
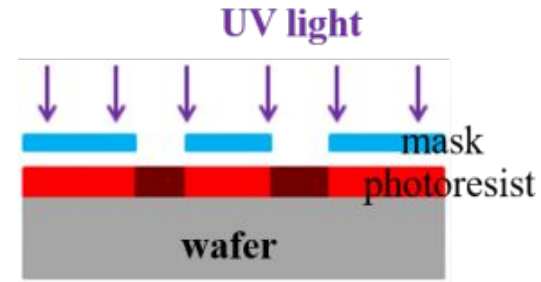
a patterned metal film used in the manufacture of microcircuits to allow selective modification of the underlying material.

A Mask lets only select face parts or light through.

AND gate - if one input is

1 - allows the other through

0 - blocks



AND operation Example – My Turn!

- Useful to “mask” bits in a word
 - Select some bits, clear others to 0
- and \$t0, \$t1, \$t2

\$t2	0 0 0 0 _ 0 0 0 0 _ 0 0 0 0 _ 0 0 0 0 _ 0 0 0 0 _ 1 0 1 1 _ 1 1 0 0 _ 0 0 0 0																															
\$t1 (mask)	0 0 0 0 _ 0 0 0 0 _ 0 0 0 0 _ 0 0 0 0 _ 0 0 1 1 _ 1 1 0 0 _ 0 0 0 0 _ 0 0 0 0																															
\$t0	0 0 0 0 _ 0 0 0 0 _ 0 0 0 0 _ 0 0 0 0 _ _ _ _ _ 0 0 0 0																															
Bit#	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

AND operation Example – Your Turn

- Useful to “mask” bits in a word
 - Select some bits, clear others to 0

and \$t0, \$t1, \$t2

Which bits in
\$t0 are '1'?

\$t2	0	0	1	0	_	0	1	0	1	_	1	0	0	0	_	1	1	0	0	_	1	0	0	1	_	0	0	1	1	_	1	0	0	0	_	0	1	0	1
\$t1 (mask)	0	0	0	0	_	1	1	1	1	_	0	0	0	0	_	1	1	0	0	_	0	0	0	0	_	0	1	1	1	_	0	0	0	0	_	0	0	0	0
\$t0																																							
Bit#	31	30	29	28		27	26	25	24		23	22	21	20		19	18	17	16		15	14	13	12		11	10	9	8		7	6	5	4		3	2	1	0

OR operation



or¹

/ôr/

noun

noun: **OR**; noun: **or**; plural noun: **ors**

a Boolean operator that gives the value one if at least one operand (or input) has a value of one, and otherwise has a value of zero.

- ELECTRONICS**

a circuit that gives an output signal if there is a signal on any of its inputs.

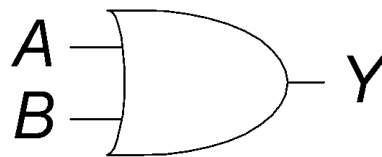
modifier noun: **OR**; noun: **OR gate**; plural noun: **OR gates**

- Fundamental Boolean operator

or \$t0, \$t1, \$t2

- Useful to include bits in a word

OR



$$Y = A + B$$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

OR operation Example – My Turn

- Useful to include bits in a word
 - Set some bits to 1, leave others unchanged
- or \$t0, \$t1, \$t2

\$t2	0	0	0	0	_	0	0	0	0	_	0	0	0	0	_	0	0	0	0	_	0	0	0	0	_	1	0	1	1	_	1	1	0	0	_	0	0	0	0
\$t1	0	0	0	0	_	0	0	0	0	_	0	0	0	0	_	0	0	1	1	_	1	1	0	0	_	0	0	0	0	_	0	0	0	0	_	0	0	0	0
\$t0	0	0	0	0	_	0	0	0	0	_	0	0	0	0	_					_					_					_					_	0	0	0	0
Bit#	31	30	29	28		27	26	25	24		23	22	21	20		19	18	17	16		15	14	13	12		11	10	9	8		7	6	5	4		3	2	1	0

OR operation Example – Your Turn

- Useful to include bits in a word
 - Set some bits to 1, leave others unchanged

or \$t0, \$t1, \$t2

Which bits in
\$t0 are '1'?

\$t2	0 0 0 0 _ 0 1 0 1 _ 0 0 0 0 _ 1 1 0 0 _ 1 0 0 1 _ 0 0 1 1 _ 1 0 0 0 _ 0 0 0 0																																						
\$t1	0 0 0 0 _ 1 1 1 1 _ 0 0 0 0 _ 1 1 0 0 _ 0 0 0 0 _ 0 1 1 1 _ 0 0 0 0 _ 0 0 0 0																																						
\$t0																																							
Bit#	31	30	29	28		27	26	25	24		23	22	21	20		19	18	17	16		15	14	13	12		11	10	9	8		7	6	5	4		3	2	1	0

NOT operation

- Useful to invert bits in a word
 - Change 0 to 1, and 1 to 0
- NOT is a pseudo-instruction → uses the NOR native instruction
 - $a \text{ NOR } b == \text{NOT} (a \text{ OR } b)$
 - `nor $t0, $t1, $zero`

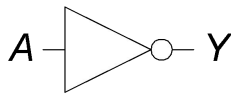
NOR



$$Y = \overline{A + B}$$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

NOT



$$Y = \overline{A}$$

A	Y
0	1
1	0

\$t1

0	0	0	0	_	1	1	1	1	_	0	0	0	0	_	1	1	0	0	_	0	0	0	0	_	0	1	1	1	_	0	0	0	0	_	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

\$t0

1	1	1	1	_	0	0	0	0	_	1	1	1	1	_	0	0	1	1	_	1	1	1	1	_	1	0	0	0	_	1	1	1	1	_	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Instructions: Logical immediates

- One of the inputs is an immediate instead of a register
- Examples:

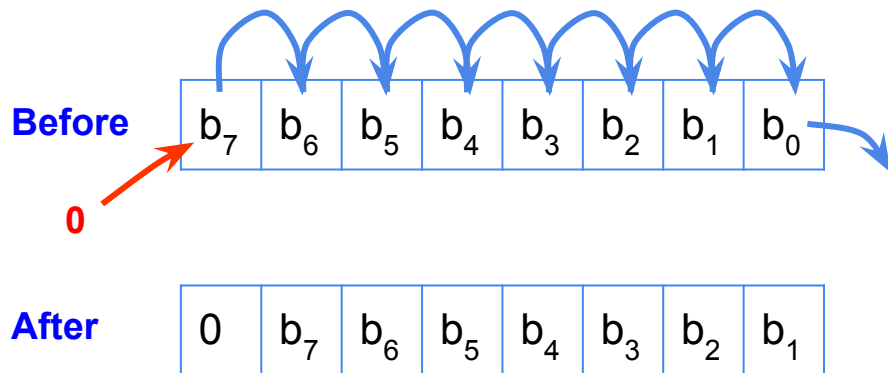
```
ori $t0, $t1, 5
```

```
andi $t0, $t1, 15
```

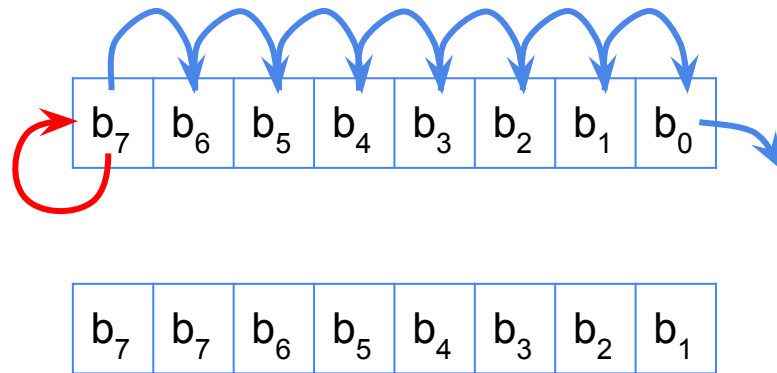
Shifters

- Shifters are used in computer programming to optimize mathematical operations, manipulating data at the bit level, implementing certain algorithms efficiently.
- Two types of shifters: Logical and Arithmetic

Shift Right Logical



Shift Right Arithmetic



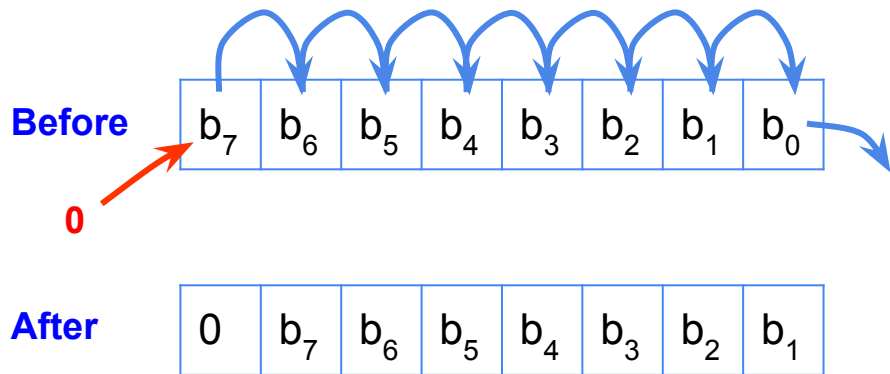
Shifters – Logical Example – My Turn

- **Logical shifter (<<, >>)** : shifts or moves bits to left (<<) or right (>>) and fills empty spaces with 0's
 - srl: Shift right logical:
 - sll: Shift left logical:

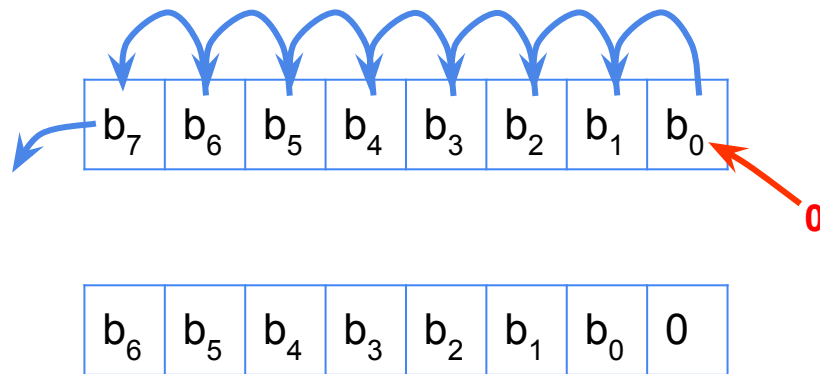
0b11001 >> 2 = 0b00110

0b11001 << 2 = 0b00100

Shift Right Logical



Shift Left Logical



Shifters – Logical Example – My Turn

- **Logical shifter (<<, >>)** : shifts or moves bits to left (<<) or right (>>) and fills empty spaces with 0's
 - `sr l`: Shift right logical
 - `sll`: Shift left logical

```
sll $t0, $t1, 3
```

What is \$t0 in binary and hex?

\$t1 = 0b _____

\$t0 = 0b _____

= 0x _____

8-bit Registers

\$t0	0x00
\$t1	0x0B
\$t2	0x0C
\$t3	0x0D

Shifters – Logical Example – Your Turn

- **Logical shifter (<<, >>)** : shifts or moves bits to left (<<) or right (>>) and fills empty spaces with 0's
 - `sr l`: Shift right logical
 - `sll`: Shift left logical

```
sr1 $t0, $t2, 2
```

What is \$t0 in binary and hex?

\$t2 = 0b _____

\$t0 = 0b _____

= 0x _____

8-bit Registers

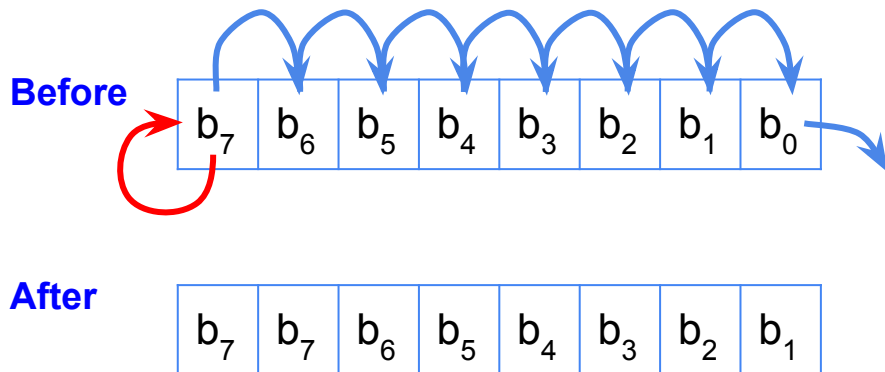
\$t0	0x00
\$t1	0x0B
\$t2	0x0C
\$t3	0x0D

Shifters – Arithmetic Example

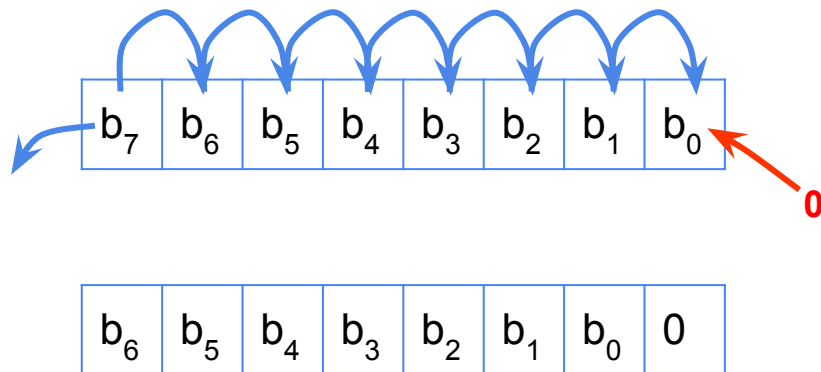
- **Arithmetic shifter** (\gg , \ll): right shift (\gg), fills empty spaces with the old most significant bit (MSB), left shift (\ll) is the same as logical shifter (\ll)
 - **sra**: Shift right arithmetic:
 - **sll**: Shift left logical:

$\begin{matrix} \nearrow \nearrow \nearrow \nearrow \\ 11001 \end{matrix} \ggg 2 = 11110$
 $\begin{matrix} \nwarrow \nwarrow \nwarrow \nwarrow \\ 11001 \end{matrix} \lll 2 = 00100$

Shift Right Arithmetic



Shift Left Logical



Where are shifters used?

1. Shifters as Multipliers and Dividers

- Arithmetic shifters are useful for low cost and quick math
 - $A \lll N = A \times 2^N \rightarrow \textit{Multiply by powers of 2}$
 - $A \ggg N = A \div 2^N \rightarrow \textit{Divide by powers of 2}$

2. Shifters are often used for memory address calculation

Shifters as Mult and Div – Example – My Turn

Assume Two's Complement numbers

- $A \ll N = A \times 2^N$

Decimal

- **Example:** $0b00001 \ll 2 = 0b______ \left(___ \times ___ = ___ \right)$

- $A \ggg N = A \div 2^N$

Decimal

- **Example:** $0b11000 \ggg 2 = 0b______ \left(___ \div ___ = ___ \right)$

Shift in the MSB to maintain sign (+)

Shifters as Mult and Div – Example – Your Turn

Assume Two's Complement numbers

- $A \ll N = A \times 2^N$

Decimal

- **Example:** $0b111101 \ll 3 = 0b______ \left(___ \times ___ = ___ \right)$

- $A \ggg N = A \div 2^N$

Decimal

- **Example:** $0b001000 \ggg 1 = 0b______ \left(___ \div ___ = ___ \right)$

Shift in the MSB to maintain sign (+)

Memory Address Calculation Example

- Shifters are often used for memory address calculation
- Example Python code: `g = A[8]`
 - Variable g in \$s1, index 8 in \$s2, base address of A in \$s3
- Compiled MIPS code:

```
sll $s2, $s2, 2      # multiply index by 4 (word -> byte)
add $s3, $s3, $s2    # add the offset to base address of A
lw  $s1, 0($s3)      # load word A[8] to $s1
```

Part 1 – Machine Coding – Basics

- We will go through the MIPS Reference Card that you will bring to the exam (find it on eLearning)
- Today, we will be Human Assemblers
 - Convert Assembly language to Binary

You will need the MIPS reference card to help me answer questions

The Stored Program

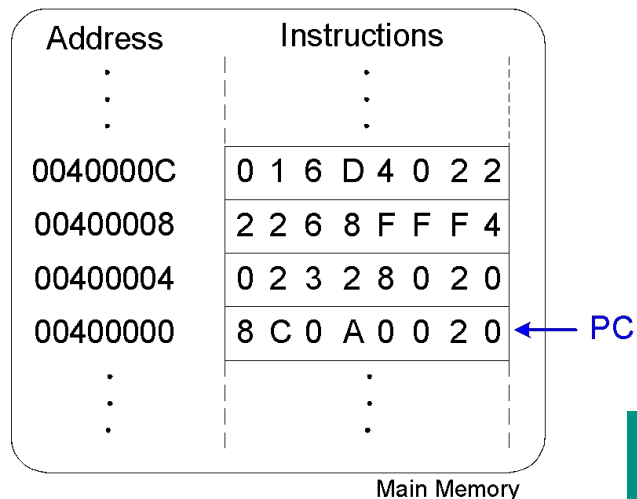
Assembly Code

```
lw    $t2, 32($0)
add   $s0, $s1, $s2
addi  $t0, $s3, -12
sub   $t0, $t3, $t5
```

Machine Code

```
0x8C0A0020
0x02328020
0x2268FFF4
0x016D4022
```

Stored Program



- 32-bit instructions & data stored in memory
- Sequence of instructions: only difference between two applications
- To run a new program:
 - No rewiring required
 - Simply store new program in memory (binary)
- "binary" would become shorthand for "binary executable compiler output"
 - Directories containing these files are called bin
- Program Counter (PC) is a register that holds the current instruction

We will be creating machine code from assembly code!

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x24080005	addiu \$8,\$0,0x00000005	9: li \$t0, 5
	0x00400004	0x24090009	addiu \$9,\$0,0x00000009	10: li \$t1, 9
	0x00400008	0x240a000f	addiu \$10,\$0,0x0000...	11: li \$t2, 15
	0x0040000c	0x240b0007	addiu \$11,\$0,0x0000...	12: li \$t3, 7
	0x00400010	0x240c0001	addiu \$12,\$0,0x0000...	13: li \$t4, 1
	0x00400014	0x240d0000	addiu \$13,\$0,0x0000...	14: li \$t5, 0
	0x00400018	0x21080005	addi \$8,\$8,0x00000005	17: addi \$t0, \$t0, 5
	0x0040001c	0x01484822	sub \$9,\$10,\$8	18: sub \$t1, \$t2, \$t0
	0x00400020	0x018d5820	add \$11,\$12,\$13	19: add \$t3, \$t4, \$t5

Labels

Label	Address
Exam 1.asm	
Else	0x004000bc
Exit	0x004000c0
NewProc	0x004000d0
NumArray	0x10010000

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000005	0x00000003	0x00000008	0x00000002	0x00000003	0x0000000a	0x0000000f	0x00000008
0x10010020	0x00000002	0x0000000a	0x00000022	0x00000038	0x00000017	0x00000045	0x0000004b	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Registers Coproc 1 Coproc 0

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffefffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Mars Messages Run I/O

Assembly: assembling /Users/alicewang/MARS code/Exam 1.asm

Assembly: operation completed successfully.

Clear

Program Counter (PC)

MARS instruction memory “text”

Address of Instruction in Memory

Machine Code

Assembly Code (Basic)

Source Assembly Code (Basic + Pseudo-code)

Run speed at max (no interaction)

Registers

Coproc 1

Coproc 0

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x24080005	addiu \$8,\$0,0x00000005	9: li \$t0, 5
	0x00400004	0x24090009	addiu \$9,\$0,0x00000009	10: li \$t1, 9
	0x00400008	0x240a000f	addiu \$10,\$0,0x0000...	11: li \$t2, 15
	0x0040000c	0x240b0007	addiu \$11,\$0,0x0000...	12: li \$t3, 7
	0x00400010	0x240c0001	addiu \$12,\$0,0x0000...	13: li \$t4, 1
	0x00400014	0x240d0000	addiu \$13,\$0,0x0000...	14: li \$t5, 0
	0x00400018	0x21080005	addi \$8,\$8,0x00000005	17: addi \$t0, \$t0, 5
	0x0040001c	0x01484822	sub \$9,\$10,\$8	18: sub \$t1, \$t2, \$t0
	0x00400020	0x018d5820	add \$11,\$12,\$13	19: add \$t3, \$t4, \$t5

Labels

Label	Address
Exam 1.asm	
Else	0x004000bc
Exit	0x004000c0
NewProc	0x004000d0
NumArray	0x10010000

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc0
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Mars Messages

Run I/O

Assembly: assembling /Users/alicewang/MARS code/Exam 1.asm

Assembly: operation completed successfully.

Clear

Everything is a number

- MIPS Instructions are encoded in binary = “Machine Code”
 - Encoded as 32-bit instruction words
 - The operation code (opcode) specifies which instruction to execute
- Operands are also binary (e.g. Registers are numbers)
 - \$t0 – \$t7 are reg’s 8 – 15
 - \$t8 – \$t9 are reg’s 24 – 25
 - \$s0 – \$s7 are reg’s 16 – 23

Register Table in MARS

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x24080005	addiu \$8,\$0,0x00000005	9: li \$t0, 5
	0x00400004	0x24090009	addiu \$9,\$0,0x00000009	10: li \$t1, 9
	0x00400008	0x240a000f	addiu \$10,\$0,0x0000...	11: li \$t2, 15
	0x0040000c	0x240b0007	addiu \$11,\$0,0x0000...	12: li \$t3, 7
	0x00400010	0x240c0001	addiu \$12,\$0,0x0000...	13: li \$t4, 1
	0x00400014	0x240d0000	addiu \$13,\$0,0x0000...	14: li \$t5, 0
	0x00400018	0x21080005	addi \$8,\$8,0x00000005	17: addi \$t0, \$t0, 5
	0x0040001c	0x01484822	sub \$9,\$10,\$8	18: sub \$t1, \$t2, \$t0
	0x00400020	0x018d5820	add \$11,\$12,\$13	19: add \$t3, \$t4, \$t5

Labels

Label	Address
Exam 1.asm	
Else	0x004000bc
Exit	0x004000c0
NewProc	0x004000d0
NumArray	0x10010000

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffcfc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000005	0x00000003	0x00000008	0x00000002	0x00000003	0x0000000a	0x0000000f	0x00000008
0x10010020	0x00000002	0x0000000a	0x00000022	0x00000038	0x00000017	0x00000045	0x0000004b	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Mars Messages Run I/O

Assembly: assembling /Users/alicewang/MARS code/Exam 1.asm

Assembly: operation completed successfully.

Clear

Registers

Name

Value (data)

Num in Machine Code

Register Table is found on the Green Card

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

Registers		
Registers Coproc 1 Coproc 0		
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Register Table - location on the MIPS card

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

MIPS Reference Data

CORE INSTRUCTION SET

NAME, MNEMONIC	FOR- MAT	OPERATION (in Verilog)	/FUNCT (Hex)
Add	add	R [Rd] = R[Rs] + R[Rt]	(1) 0/20hex
Add Immediate	addi	I R [Rd] = R[Rs] + SignExtImm	(1,2) 8hex
Add Imm. Unsigned	addiu	I R [Rd] = R[Rs] + SignExtImm	(2) 9hex
Add Unsigned	addu	R [Rd] = R[Rs] + R[Rt]	0/21hex
And	and	R [Rd] = R[Rs] & R[Rt]	0/24hex
And Immediate	andi	I R [Rd] = R[Rs] & ZeroExtImm	(3) 0hex
Branch On Equal	beq	I if(R[Rs]==R[Rt]) PC=PC+BranchAddr	(4) 4hex
Branch On Not Equal	bne	I if(R[Rs]!=R[Rt]) PC=PC+BranchAddr	5hex
Jump	j	J PC=JumpAddr	(5) 2hex
Jump And Link	jal	J PC[31]=PC+8;PC=JumpAddr	(5) 3hex
Jump Register	jr	R PC=R[Rs]	0/08hex
Load Byte Unsigned	lbu	I R [Rd] = (24'b0[M[Rs]] +SignExtImm(7:0))	(2) 24hex
Load Halfword Unsigned	lhu	I R [Rd] = (16'b0[M[Rs]] +SignExtImm(15:0))	(2) 25hex
Load Linked	ll	I R [Rd] = M[R[Rs]+SignExtImm]	(2,7) 30hex
Load Upper Imm.	lui	I R [Rd] = (imm, 16b0)	6hex
Load Word	lw	I R [Rd] = M[R[Rs]+SignExtImm]	(2) 23hex
Nor	nor	R [Rd] = ~(R[Rs] R[Rt])	(2) 27hex
Or	or	R [Rd] = R[Rs] R[Rt]	0/25hex
Or Immediate	ori	I R [Rd] = R[Rs] ZeroExtImm	(3) 0hex
Set Less Than	slt	R [Rd] = (R[Rs] < R[Rt]) ? 1 : 0	0/26hex
Set Less Than Imm.	slti	I R [Rd] = (R[Rs] < SignExtImm) ? 1 : 0	(2) 0hex
Set Less Than Imm. Unsigned	sltiu	I R [Rd] = (R[Rs] < SignExtImm) ? 1 : 0	(2,6) 0hex
Set Less Than Unsig.	sltu	R [Rd] = (R[Rs] < R[Rt]) ? 1 : 0	(6) 0/26hex
Shift Left Logical	sll	R [Rd] = R[Rt] << shamt	0/00hex
Shift Right Logical	srl	R [Rd] = R[Rt] >> shamt	0/02hex
Store Byte	sb	I M[R[Rs]+SignExtImm(7:0)] = R[Rt](7:0)	(2) 29hex
Store Conditional	sc	I M[R[Rs]+SignExtImm] = R[Rt]; R[Rs] = (atomic) ? 1 : 0	(2,7) 38hex
Store Halfword	sh	I M[R[Rs]+SignExtImm(15:0)] = R[Rt](15:0)	(2) 29hex
Store Word	sw	I M[R[Rs]+SignExtImm] = R[Rt]	(2) 29hex
Subtract	sub	R [Rd] = R[Rs] - R[Rt]	(1) 0/22hex
Subtract Unsigned	subu	R [Rd] = R[Rs] - R[Rt]	0/23hex

(1) May cause overflow exception

(2) SignExtImm = (16[immediate][15], immediate)

(3) ZeroExtImm = (16[1'b0], immediate)

(4) BranchAddr = (14[immediate][15], immediate, 2'b0)

(5) JumpAddr = (PC[4:31-28], address, 2'b0)

(6) Operands considered as signed numbers (vs. 2's comp)

(7) Atomic test-and-set; R[Rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
		26,25	21,20	16,15	11,10	6,5
I	opcode	rs	rt	immediate		
		31	26,25	21,20	16,15	
J	opcode		address			
		31	26,25			

ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-	OPERATION	/ FMT / / FUNCT
Branch On FP True	bfc I	if(FPcond) PC = PC + BranchAddr (4)	(Hex)
Branch On FP False	bfc I	if(!FPcond) PC = PC + BranchAddr (4)	11/8/0/-
Divide	div R	Lo = R[Rs]/R[Rt]; Hi = R[Rs]/R[Rt]	0/-/1a
Divide Unsigned	divu R	Lo = R[Rs]/R[Rt]; Hi = R[Rs]/R[Rt]	(6) 0/-/1b
FP Add Single	addsf FR	F[Rd] = F[Rs] + F[Rt]	11/10/-/0
FP Add	addf FR	F[Rd] = F[Rs] + F[Rt]	11/10/-/0
FP Compare Single	cmpsf FR	FPcond = (F[Rs] > F[Rt]) ? 1 : 0	11/10/-/0
FP Compare	cmpf FR	FPcond = ((F[Rs] > F[Rt]) & op = 0) (F[Rs] < F[Rt]) ? 1 : 0	11/10/-/0
FP Double	op = 0, 1, or 2	FPcond = ((F[Rs] > F[Rt]) & op = 0) (F[Rs] < F[Rt]) ? 1 : 0	11/10/-/0
FP Divide Single	divsf FR	F[Rd] = F[Rs] / F[Rt]	11/10/-/3
FP Divide	divf FR	F[Rd] = F[Rs] / F[Rt]	11/10/-/3
FP Multiply Single	mulsf FR	F[Rd] = F[Rs] * F[Rt]	11/10/-/2
FP Multiply	mulf FR	F[Rd] = F[Rs] * F[Rt]	11/10/-/2
FP Subtract Single	subsf FR	F[Rd] = F[Rs] - F[Rt]	11/10/-/1
FP Subtract	subf FR	F[Rd] = F[Rs] - F[Rt]	11/10/-/1
FP Double	op = 0, 1, or 2	F[Rd] = F[Rs] - F[Rt]	11/10/-/1
Load FP Single	lwc1 I	F[Rd] = M[R[Rs] + SignExtImm]	(2) 31/-/1a
Load FP Double	ldc1 I	F[Rd] = M[R[Rs] + SignExtImm]	(2) 35/-/1a
Move From Hi	mfr1 R	R[Rd] = Hi	0/-/10/-10
Move From Lo	mfr1 R	R[Rd] = Lo	0/-/10/-12
Move From Control	mfc0 R	R[Rd] = CR[Rs]	10/0/-/0
Multiply	mul R	R[Rd] = R[Rs] * R[Rt]	0/-/18
Multiply Unsigned	mulu R	R[Rd] = R[Rs] * R[Rt]	(6) 0/-/19
Shift Right Arith.	sra R	R[Rd] = R[Rt] >> shamt	0/-/0/3
Store FP Single	swc1 I	M[R[Rs] + SignExtImm] = F[Rs]	(2) 39/-/1a
Store FP Double	swd1 I	M[R[Rs] + SignExtImm] = F[Rs]	(2) 3d/-/1a

FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	funct	ft	fs	fd	funct
		31	26,25	21,20	16,15	11,10
FI	opcode	funct	ft	fs	fd	funct
		31	26,25	21,20	16,15	

PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt I R[Rd] = R[Rs] < R[Rt] ? PC = Label	
Branch Greater Than	bgt I R[Rd] = R[Rs] > R[Rt] ? PC = Label	
Branch Less Than or Equal	bte I R[Rd] = R[Rs] <= R[Rt] ? PC = Label	
Branch Greater Than or Equal	bgt I R[Rd] = R[Rs] >= R[Rt] ? PC = Label	
Load Immediate	li I R[Rd] = immediate	

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

OPCODES, BASE CONVERSION, ASCII SYMBOLS

MIPS (31:26)	MIPS (25:21)	MIPS (20:16)	Binary	Decimal	Hex	Char	ASCII
(1)	(1)	(1)	000000	0	NUL		64
j	al	sub	000001	1	SOH		65
s	ri	mul	000010	2	STX		66
l	al	div	000011	3	ETX		67
beq	al	sub	000100	4	ETX		68
bne	al	sub	000101	5	ENQ		69
lbu	al	sub	000110	6	ACK		70
lbu	al	sub	000111	7	BEL		71
addi	jr	sub	001000	8	BS		72
addi	jalr	sub	001001	9	HT		73
slti	movz	sub	001010	10	LF		74
sltiu	movz	sub	001011	11	b VT		75
andi	movn	sub	001100	12	c VT		76
andi	break	sub	001101	13	e CR		77
xori	floor	sub	001110	14	e SO		78
lui	sync	sub	001111	15	f SI		79
(2)	(2)	(2)	010000	16	DL		80
mtli	movf	sub	010001	17	DC1		81
mflr	movf	sub	010010	18	DC2		82
mflr	movf	sub	010011	19	DC3		83
010100	20	14	DC4	84	SA		84
010101	21	15	NAK	85	SS		85
010110	22	16	SYN	86	SV		86
010111	23	17	ETB	87	SW		87
011000	24	18	CAN	88	TX		88
011001	25	19	EM	89	SY		89
011010	26	20	SUB	90	Z		90
011011	27	21	ESC	91	SB		91
011100	28	22	FS	92	S		92
011101	29	23	Space	93	SA		93
100000	30	24	Space	94	SA		94
100001	31	25	Space	95	SA		95
100010	32	26	Space	96	SA		96
100011	33	27	Space	97	SA		97
100100	34	28	Space	98	SA		98
100101	35	29	Space	99	SA		99
100110	36	30	Space	100	SA		100
100111	37	31	Space	101	SA		101
101000	38	32	Space	102	SA		102
101001	39	33	Space	103	SA		103
101010	40	34	Space	104	SA		104
101011	41	35	Space	105	SA		105
101100	42	36	Space	106	SA		106
101101	43	37	Space	107	SA		107
101110	44	38	Space	108	SA		108
101111	45	39	Space	109	SA		109
110100	46	40	Space	110	SA		110
110101	47	41	Space	111	SA		111
110110	48	42	Space	112	SA		112
110111	49	43	Space	113	SA		113
111000	50	44	Space	114	SA		114
111001	51	45	Space	115	SA		115
111010	52	46	Space	116	SA		116
111011	53	47	Space	117	SA		117
111100	54	48	Space	118	SA		118
111101	55	49	Space	119	SA		119
111110	56	50	Space	120	SA		120
111111	57	51	Space	121	SA		121
110100	58	52	Space	122	SA		122
110101	59	53	Space	123	SA		123
110110	60	54	Space	124	SA		124
110111	61	55	Space	125	SA		125
111000	62	56	Space	126	SA		126
111001	63	57	Space	127	SA		127

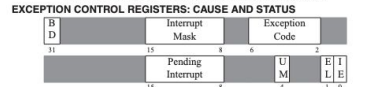
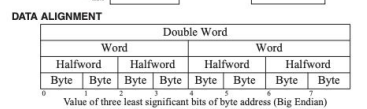
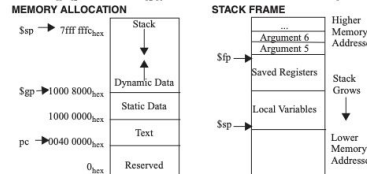
(1) opcode(31:26) = 0
(2) opcode(31:26) = 17_{hex} (11_{hex}), if funt(25:21) = 16_{hex} (10_{hex}) / = 3 (single),
if funt(25:21) = 17_{hex} (11_{hex}) / = 4 (double)

Copyright 2009 by Elsevier, Inc. All rights reserved. From Patterson and Hennessy, Computer Organization and Design, 3rd ed.

IEEE 754 FLOATING-POINT STANDARD

(-1)^S × (1 + Fraction) × 2^(Exponent - Bias)
where Single Precision Bias = 127,
Double Precision Bias = 1023.

IEEE Single Precision and Double Precision Formats:
S.P. MAX = 255, D.P. MAX = 2047



BD = Branch Delay, UM = User Mode, EL = Exception Level, EI = Interrupt Enable

EXCEPTION CODES					
Number	Name	Cause of Exception	Number	Name	Cause of Exception
0	Int	Interrupt (hardware)	9	Bp	Breakpoint Exception
4	AdEL	Address Error Exception (load or instruction fetch)	10	Ri	Reserved Instruction Exception
5	AdES	Address Error Exception (store)	11	CpU	Coprocessor Unimplemented Exception
6	IBE	Bus Error on Instruction Fetch	12	Ov	Arithmetic Overflow Exception
7	DBE	Bus Error on Load or Store	13	Tr	Trap
8	Sys	Syscall Exception	15	FPE	Floating Point Exception

SIZE PREFIXES (10⁹ for Disk, Communication; 2³⁰ for Memory)

SI Size	Prefix	Symbol	IEC Size	Prefix	Symbol
10 ³	Kilo-	K	2 ¹⁰	Kibi-	Ki
10 ⁶	Mega-	M	2 ²⁰	Mebi-	Mi
10 ⁹	Giga-	G	2 ³⁰	Gibi-	Gi
10 ¹²	Tera-	T	2 ⁴⁰	Tebi-	Ti
10 ¹⁵	Peta-	P	2 ⁵⁰	Pebi-	Pi
10 ¹⁸	Exa-	E	2 ⁶⁰	Exbi-	Ei
10 ²¹	Zetta-	Z	2 ⁷⁰	Zebi-	Zi
10 ²⁴	Yotta-	Y	2 ⁸⁰	Yobi-	Yi

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

CORE INSTRUCTION SET

BASIC INSTRUCTION FORMATS

ARITHMETIC CORE INSTRUCTION SET

We have learned every instruction column. Let's zoo

	Move	Movl	R	$R[r] = Lo$	$0 \rightarrow r[rd]-12$
Move From Control	mfc0	R	$R[r] = CR[r]$	$10\ 0/0 \rightarrow$	
Multiply	mult	R	$(HiLo) = R[r] * R[r]$	$0/0 \rightarrow 18$	
Multiply Unsigned	multui	R	$(HiLo) = R[r] * R[r]$	$(6) \rightarrow 19 \rightarrow 18$	
Shift Right Arith.	sra	R	$R[r] = R[r] >> shamt$	$0/0 \rightarrow 3$	
Shift Right Single	srl	I	$M[R[r]] = Sigmoid(Imm) = F[r]$	$(2) \ 39/0 \rightarrow 19$	
Store FP	stf	F	$M[R[r]] = SigmoidExt(Imm) = F[r+1]$	$(3) \ 3d/0 \rightarrow 19$	
Duplicate	sdcl	I	$M[R[r]] = SigmoidExt(Imm) = F[r+1]$	$(3) \ 3d/0 \rightarrow 19$	

FLOATING-POINT INSTRUCTION FORMATS

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	PRESERVEDACROSS A CALL?
Szero	0	The Constant Value 0	N.A.
Sat	1	Assembler Temporary	No
Sv0-Sv1	2-3	Values for Function Results and Expression Evaluation	No
Sa0-Sa3	4-7	Arguments	No
Sb0-Sb7	8-15	Temporaries	No
Sd0-Sd7	16-23	Saved Temporaries	Yes
St0-St9	24-25	Temporaries	No
Sb0-Sb1	26-27	Reserved for OS Kernel	No
Sgp	28	Global Pointer	Yes
Ssp	29	Stack Pointer	Yes
Sfp	30	Frame Pointer	Yes
Sra	31	Return Address	No

② OPCODE

PCODES, BASE CONVERSION, ASCII SYMBOLS									
MIPS (1) MIPS (2)			MIPS			Decimals		Hex	
opcode	func	func	func	func	func	dec	dec	hex	hex
(126)	(50)	(50)	(50)	(50)	(50)				
all	all	addsf	000000	0	NUL	64	40	40	
	sr	subf	000001	1	SOH	65	41	41	
	sr	subf	000010	2	STX	66	42	42	
	sr	divsf	000001	3		67	43	43	
	beg	sqr	000100	4	EOT	68	44	44	
	hqr	abaf	000101	5	ENQ	69	45	45	
	hqr	abaf	000110	6	STX	70	46	46	
	hqr	neqf	000111	7	BEL	71	47	47	
	addl	ir	000100	8	BS	72	48	48	
	addl	ir	000101	9	STX	73	49	49	
div	addl	ir	000110	10	LF	74	4A	4A	
	addl	ir	000111	11	b VT	75	4B	4B	
	rand	swf	000110	12	C	76	4C	4C	
	rand	swf	000111	13	SOH	77	4D	4D	
	rand	swf	000110	14	C	78	4E	4E	
	rand	swf	000111	15	SI	79	4F	4F	
	div	swf	001000	16	STX	80	50	50	
	div	swf	000101	17	DC1	81	51	51	
	div	swf	000110	18	DC2	82	52	52	
	div	swf	000111	19	STX	83	53	53	
divu	div	swf	001010	20	DC4	84	54	54	
	div	swf	000101	21	NAK	85	55	55	
	div	swf	000110	22	STX	86	56	56	
	div	swf	000111	23	ETB	87	57	57	
	div	swf	010000	24	CAN	88	58	58	
	div	swf	000101	25	EM	89	59	59	
	div	swf	000110	26	STX	90	5A	5A	
	div	swf	010111	27	BSC	91	5B	5B	
	div	swf	011110	28	FS	92	5C	5C	
	div	swf	011101	29	GS	93	5D	5D	
lwl	lwl	sub	011110	30	RS	94	5E	5E	
	lwl	sub	011111	31	US	95	5F	5F	
	lwl	sub	010000	32	Space	96	60	60	
	lwl	sub	000100	33	21	97	61	61	
	lwl	sub	000010	34	22	98	62	62	
	lwl	sub	000101	35	23	99	63	63	
	lwl	sub	000110	36	24	100	64	64	
	lwl	sub	000111	37	25	101	65	65	
	lwl	sub	001010	38	26	102	66	66	
	lwl	sub	001011	39	27	103	67	67	
swl	swl	nor	001000	40	28	104	68	68	
	swl	nor	000101	41	29	105	69	69	
	swl	nor	000110	42	30	106	6A	6A	
	swl	nor	000111	43	29	107	6B	6B	
	swl	nor	010000	44	2c	108	6C	6C	
	swl	nor	010110	45	2d	109	6D	6D	
	swl	nor	010111	46	2e	110	6E	6E	
	swl	nor	011000	47	2f	111	6F	6F	
	swl	nor	011001	48	30	112	70	70	
	swl	nor	011000	49	31	113	71	71	
pref	pref	tit	010010	50	32	114	72	72	
	pref	tit	010011	51	33	115	73	73	
	lcl	tqe	010010	52	34	116	74	74	
	lcl	tqe	010011	53	35	117	75	75	
	lcl	lcl	010110	54	36	118	76	76	
	lcl	lcl	010111	55	37	119	77	77	
	swl	swl	011000	56	38	120	78	78	
	swl	swl	011001	57	39	121	79	79	
	swl	swl	011010	58	3a	122	7A	7A	
	swl	swl	011011	59	3b	123	7B	7B	
sdcl	sdcl	swl	011100	60	3c	124	7C	7C	
	sdcl	swl	011101	61	3d	125	7D	7D	
	sdcl	swl	011110	62	3e	126	7E	7E	
	sdcl	swl	011111	63	3f	127	7F	7F	
	sdcl	swl	010000	64	40	128	80	80	
	sdcl	swl	010001	65	41	129	81	81	
	sdcl	swl	010010	66	42	130	82	82	
	sdcl	swl	010011	67	43	131	83	83	
	sdcl	swl	010100	68	44	132	84	84	
	sdcl	swl	010101	69	45	133	85	85	

Copyright 2009 by Elsevier, Inc.. All rights reserved. From Patterson and Hennessy. *Comm*

IEEE 754 FLOATING-POINT
STANDARD

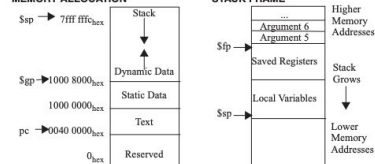
where Single Precision Bias = 127,
Double Precision Bias = 1023.

IEEE Single Precision and

Double Precision Formats:



MEMORY ALLOCATION



DATA ALIGNMENT

Double Word							
Word				Word			
Halfword		Halfword		Halfword		Halfword	
Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
0	1	2	3	4	5	6	7

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

EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS

B D	Interrupt Mask				Exception Code			
31	15	8	6	2				
Pending Interrupt				U M	E L	I E		

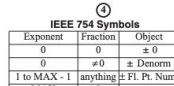
15 8 4 1 0

EXCEPTION CODES

Number	Name	Cause of Exception	Number	Name	Cause of Exception
0	Int	Interrupt (hardware)	9	Bp	Breakpoint Exception
4	AdEL	Address Error Exception (load or instruction fetch)	10	RI	Reserved Instruction Exception
5	AdES	Address Error Exception (store)	11	CpU	Coprocessor Unimplemented
6	IBE	Bus Error on Instruction Fetch	12	Ov	Arithmetic Overflow Exception
7	DBE	Bus Error on Load or Store	13	Tr	Trap
8	Sv	Syscall Exception	15	FPE	Floating Point Exception

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

Size	SI Prefix	Prefix	Symbol	IEC Size	Prefix	Symbol
10^3	Kilo-		K	2^{10}	Kibi-	Ki
10^6	Mega-		M	2^{20}	Mebi-	Mi
10^9	Giga-		G	2^{30}	Gibi-	Gi
10^{12}	Tera-		T	2^{40}	Tebi-	Ti
10^{15}	Peta-		P	2^{50}	Pebi-	Pi
10^{18}	Exa-		E	2^{60}	Exbi-	Ei
10^{21}	Zetta-		Z	2^{70}	Zebi-	Zi
10^{24}	Yotta-		Y	2^{80}	Yobi-	Yi



MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

MIPS Reference Data Card ("Green Card")

Arithmetic and Logic functions

CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add	R $R[rd] = R[rs] + R[rt]$	(1) $0 / 20_{\text{hex}}$
Add Immediate	addi	I $R[rt] = R[rs] + \text{SignExtImm}$	(1,2) 8_{hex}
Add Imm. Unsigned	addiu	I $R[rt] = R[rs] + \text{SignExtImm}$	(2) 9_{hex}
Add Unsigned	addu	R $R[rd] = R[rs] + R[rt]$	$0 / 21_{\text{hex}}$
And	and	R $R[rd] = R[rs] \& R[rt]$	$0 / 24_{\text{hex}}$
And Immediate	andi	I $R[rt] = R[rs] \& \text{ZeroExtImm}$	(3) c_{hex}
Nor	nor	R $R[rd] = \sim (R[rs] R[rt])$	$0 / 27_{\text{hex}}$
Or	or	R $R[rd] = R[rs] R[rt]$	$0 / 25_{\text{hex}}$
Or Immediate	ori	I $R[rt] = R[rs] \text{ZeroExtImm}$	(3) d_{hex}
Subtract	sub	R $R[rd] = R[rs] - R[rt]$	(1) $0 / 22_{\text{hex}}$
Subtract Unsigned	subu	R $R[rd] = R[rs] - R[rt]$	$0 / 23_{\text{hex}}$

This column explains in verilog, a hardware description language, that describes the instruction's operation

- **$R[*]$ is Register table**
- **$M[*]$ is the Memory**

Arithmetic and Logic functions

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 $R[rt] = R[rs] + \text{SignExtImm}$	(1,2) 8 _{hex}
Add Imm. Unsigned	addiu	I $R[rt] = R[rs] + \text{SignExtImm}$	(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 $R[rt] = R[rs] \& \text{ZeroExtImm}$	(3) c _{hex}
Nor	nor	R $R[rd] = \sim(R[rs] R[rt])$	0 / 27 _{hex}
Or	or	R $R[rd] = R[rs] R[rt]$	0 / 25 _{hex}
Or Immediate	ori	I $R[rt] = R[rs] \text{ZeroExtImm}$	(3) d _{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}

In this column are footnotes

- (#) are some instruction specific notes

- (1) May cause overflow exception
(2) $\text{SignExtImm} = \{ 16\{\text{immediate}[15]\}, \text{immediate} \}$
(3) $\text{ZeroExtImm} = \{ 16\{1b'0\}, \text{immediate} \}$

Example Footnote:

add and sub instructions may cause an overflow exception

Arithmetic and Logic functions

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR- MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add	R	$R[rd] = R[rs] + R[rt]$	(1) $0 / 20_{\text{hex}}$
Add Immediate	addi	I	$R[rt] = R[rs] + \text{SignExtImm}$	(1,2) 8_{hex}
Add Imm. Unsigned	addiu	I	$R[rt] = R[rs] + \text{SignExtImm}$	(2) 9_{hex}
Add Unsigned	addu	R	$R[rd] = R[rs] + R[rt]$	$0 / 21_{\text{hex}}$
And	and	R	$R[rd] = R[rs] \& R[rt]$	$0 / 24_{\text{hex}}$
And Immediate	andi	I	$R[rt] = R[rs] \& \text{ZeroExtImm}$	(3) c_{hex}
Nor	nor	R	$R[rd] = \sim (R[rs] R[rt])$	$0 / 27_{\text{hex}}$
Or	or	R	$R[rd] = R[rs] R[rt]$	$0 / 25_{\text{hex}}$
Or Immediate	ori	I	$R[rt] = R[rs] \text{ZeroExtImm}$	(3) d_{hex}
Subtract	sub	R	$R[rd] = R[rs] - R[rt]$	(1) $0 / 22_{\text{hex}}$
Subtract Unsigned	subu	R	$R[rd] = R[rs] - R[rt]$	$0 / 23_{\text{hex}}$

This column explain how to create the machine code

- Format is R, I or J type

3 Instruction formats (R-, I- and J- type)

MIPS Reference Data

CORE INSTRUCTION SET

NAME, MNEMONIC	MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add	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}

ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC	MAT	OPERATION	OPCODE / FUNCT (Hex)
Branch On FP True	bclt	FI if(FPcond)PC=PC+4+BranchAddr(4)	11/8/1--
Branch On FP False	bclt	FI if(!FPcond)PC=PC+4+BranchAddr(4)	11/8/0--
Divide	div	R Lo-R[rs]R[rt]; Hi-R[rs]R[rt]	0--/1a
Divide Unsigned	divu	R Lo-R[rs]R[rt]; Hi-R[rs]R[rt]	0--/1b
FP Add Single	add.s	FR F[rd] = F[rs] + F[rt]	(6) 0--/11/10--/0
FP Add	FPAdd	FR F[rd] = F[rs] + F[rt]	11/10--/0
Double	add.d	FR F[rd] = F[rs] + F[rt]	11/11--/0

OPCODE

OPCODES, BASE CONVERSION, ASCII SYMBOLS

MIPS (31-26)	MIPS (25-20)	MIPS (19-15)	Binary	Decimal	Hexa-decimal	Char-acter	Hexa-decimal	ASCII Char-acter
(1)	all	add	000000	0	0	NUL	64	@
		sub	000001	1	1	SOH	65	A
		mulf	000010	2	2	STX	66	B
		div	000011	3	3	ETX	67	C
		neg	000100	4	4	EOT	68	D
		abs	000101	5	5	ENQ	69	E
		mov	000110	6	6	ACK	70	F
		neg	000111	7	7	BEL	71	G

IEEE 754 FLOATING-POINT STANDARD

$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$
where Single Precision Bias = 127,
Double Precision Bias = 1023.

IEEE Single Precision and Double Precision Formats:

S	Exponent	Fraction
31	30	23

IEEE 754 Symbols

Exponent	Fraction	Object
0	0	± 0
0	≠ 0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠ 0	NaN

S.P. MAX = 255, D.P. MAX = 2047

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
31	26 25	21 20	16 15	11 10	6 5	0
I	opcode	rs	rt	immediate		
31	26 25	21 20	16 15	0		
J	opcode	address				
31	26 25	0				

Branch Less Than or Equal	blt	I R[rs] < R[rt] PC = Label	(2,7) 29 _{hex}
Branch Greater Than or Equal	bgt	I R[rs] > R[rt] PC = Label	(2,7) 29 _{hex}
Branch Immediate	bli	I R[rd] = immediate	(2,7) 29 _{hex}
Move	move	I R[rd] = R[rs]	(2,7) 29 _{hex}

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED/ACROSS
\$zero	0	The Constant Value 0	CALL?
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results	No
\$a0-\$a7	4-15	Arguments	No
\$t0-\$t7	16-23	Temporaries	No
\$s0-\$s7	24-25	Saved Temporaries	Yes
\$t8-\$t9	26-27	Temporaries	No
\$k0-\$k1	28-29	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

lil	tge	c.tge	110000	48	30	0	112	70	p
lwl	tgeu	c.tgeu	110001	49	31	1	113	71	q
lwc2	tlt	c.eqf	110010	50	32	2	114	72	r
pref	tltu	c.ueqf	110011	51	33	3	115	73	s
ldcl	teq	c.cltf	110100	52	34	4	116	74	t
ldc2	tne	c.cltf	110101	53	35	5	117	75	u
		c.cltf	110110	54	36	6	118	76	v
		c.cltf	110111	55	37	7	119	77	w
sc		c.nglef	111000	56	38	8	120	78	x
		c.nglef	111001	57	39	9	121	79	y
swc2		c.nglef	111010	58	3a	10	122	7a	z
		c.nglef	111011	59	3b	11	123	7b	1
		c.cltf	111100	60	3c	12	124	7c	2
addl		c.nglef	111101	61	3d	13	125	7d	3
addc2		c.cltf	111110	62	3e	14	126	7e	4
		c.nglef	111111	63	3f	15	127	7f	DEL

(1) opcode(31:26) = 0
(2) opcode(31:26) = 17_{hex} (11_{hex}) = 16_{hex} (10_{hex}) = s (single);
if inst(25:21) = 17_{hex} (11_{hex}) = d (double)

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, Computer Organization and Design, 4th ed.

SIZE PREFIXES (10⁹ for Disk, Communication; 2³⁰ for Memory)

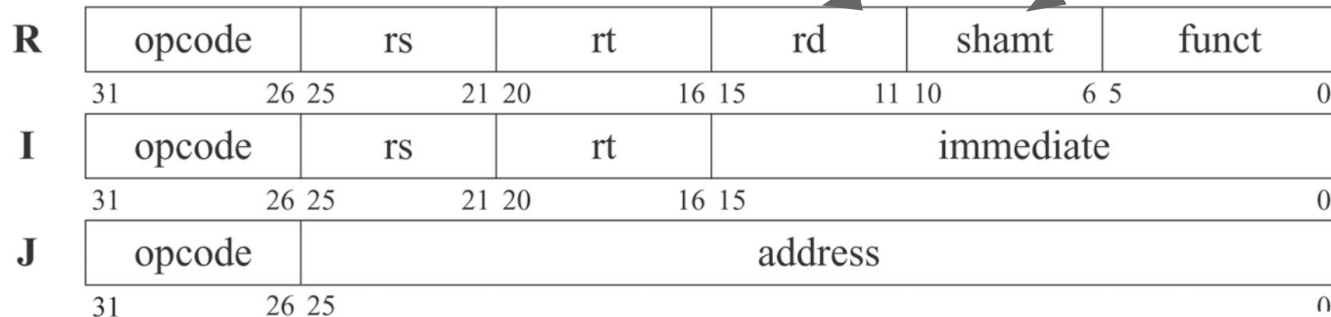
SI Size	Prefix	Symbol	IEC Size	Prefix	Symbol
10 ³	Kilo-	K	2 ¹⁰	Kibi-	Ki
10 ⁶	Mega-	M	2 ²⁰	Mebi-	Mi
10 ⁹	Giga-	G	2 ³⁰	Gibi-	Gi
10 ¹²	Tera-	T	2 ⁴⁰	Tebi-	Ti
10 ¹⁵	Peta-	P	2 ⁵⁰	Pebi-	Pi
10 ¹⁸	Exa-	E	2 ⁶⁰	Exbi-	Ei
10 ²¹	Zetta-	Z	2 ⁷⁰	Zebi-	Zi
10 ²⁴	Yotta-	Y	2 ⁸⁰	Yobi-	Yi

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

3 Instruction formats (R-, I- and J- type)

- Formats determine how to write instructions into machine language
 - Then the CPU can read the machine language to decode the instruction
- 3 instruction formats all 32-bit width
 - **R-Type**: All Register operands
 - **I-Type**: ImmEDIATE operand
 - **J-Type**: for Jumping
- The figure shows how the 32-bits are split into fields

BASIC INSTRUCTION FORMATS



Arithmetic and Logic functions

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 $R[rt] = R[rs] + \text{SignExtImm}$	(1,2) 8 _{hex}
Add Imm. Unsigned	addiu	I $R[rt] = R[rs] + \text{SignExtImm}$	(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 $R[rt] = R[rs] \& \text{ZeroExtImm}$	(3) c _{hex}
Nor	nor	R $R[rd] = \sim (R[rs] R[rt])$	0 / 27 _{hex}
Or	or	R $R[rd] = R[rs] R[rt]$	0 / 25 _{hex}
Or Immediate	ori	I $R[rt] = R[rs] \text{ZeroExtImm}$	(3) d _{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}

Look at the Opcode/Funct to fill in the opcode and funct fields for each instruction

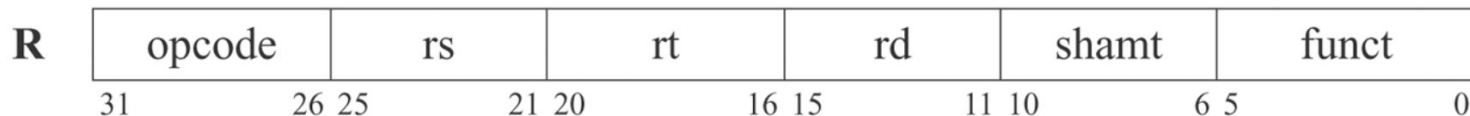
(1) May cause overflow exception

(2) $\text{SignExtImm} = \{ 16\{\text{immediate}[15]\}, \text{immediate} \}$

(3) $\text{ZeroExtImm} = \{ 16\{1b'0\}, \text{immediate} \}$

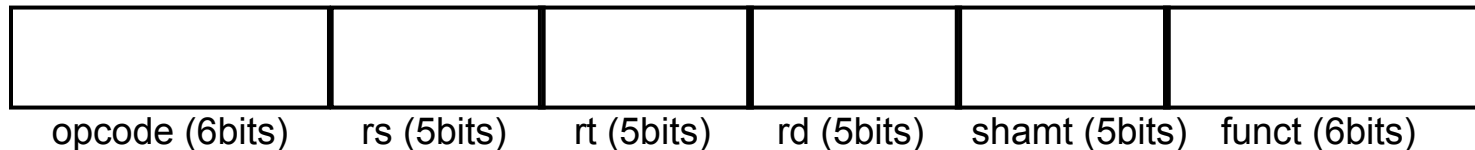
R-type Instruction

BASIC INSTRUCTION FORMATS



- All instructions that have register operands only
 - **add, sub, and, or, srl, sll**
- Instruction fields - all unsigned
 - **opcode:** operation code (opcode) - bitwidth = _____
 - **rs:** first source register number - bitwidth = _____
 - **rt:** second source register number - bitwidth = _____
 - **rd:** destination register number - bitwidth = _____
 - **shamt:** shift amount (used for srl and sll only) - bitwidth = _____
 - **funct:** function code (extends opcode) - bitwidth = _____

R-type Instruction Example - My Turn

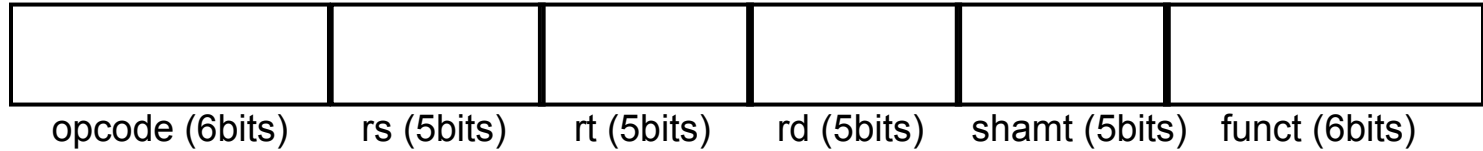


add \$t0, \$s1, \$s2

Looking at the Green Card, determine the machine code:

1. What is the opcode/funct for add?
2. What are the operands for add? What is the order?
3. Translate from binary to hexadecimal

R-type Instruction Example - My Turn



add \$t0, \$s1, \$s2

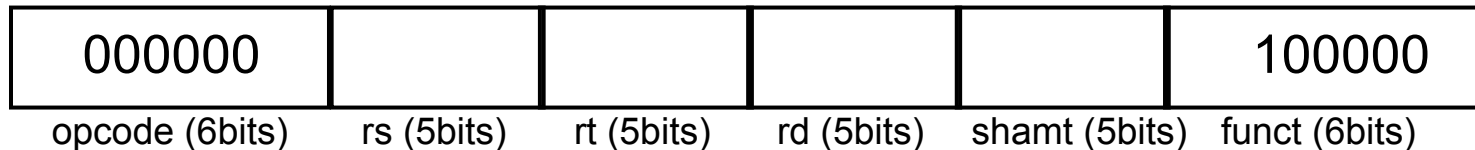
1. What is the opcode/funct for add?

CORE INSTRUCTION SET

NAME, MNEMONIC	FOR- MAT	OPERATION (in Verilog)
Add	add	R R[rd] = R[rs] + R[rt]

OPCODE
/ FUNCT
(Hex)
(1) 0 / 20_{hex}

R-type Instruction Example – My Turn



add \$t0, \$s1, \$s2

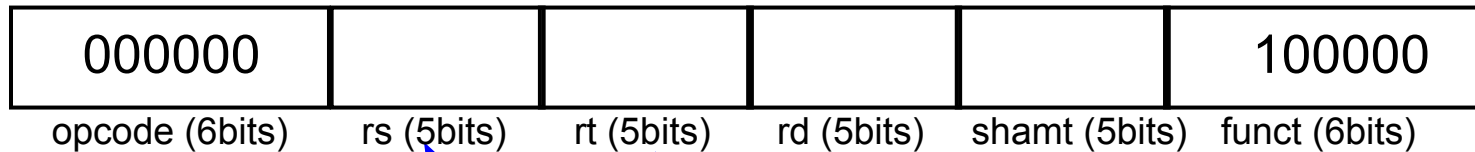
2. What are the operands for add? What is the order?

The operand order: add rd, rs, rt

CORE INSTRUCTION SET

NAME, MNEMONIC	FOR- MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add	R <u>R[rd] = R[rs] + R[rt]</u>	(1) 0 / 20 _{hex}

R-type Instruction Example - My Turn



add \$t0, \$s1, \$s2

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

Use the table for register
mapping

\$t0 = _____ (0b_____)

\$s1 = _____ (0b_____)

\$s2 = _____ (0b_____)

R-type Instruction Example - My Turn

**Instruction
in Binary**

000000	10001	10010	01000	00000	100000
opcode (6bits)	rs (5bits)	rt (5bits)	rd (5bits)	shamt (5bits)	funct (6bits)

**Convert
to Hex**

add \$t0, \$s1, \$s2 → _____₁₆ in machine language

MARS assembler

Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x02324020	add \$8,\$17,\$18	11: add \$t0, \$s1, \$s2

Machine Code

Basic Assembly Code (uses numbers for registers)

Original Assembly Code (can contain pseudo-instructions)

Labels

Label	Address
lecture 9 example.asm	
something	0x10010000

✓ Data ✓ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0	0	0	0	0	0	0	0
0x10010020	0	0	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0

0x10010000 (.data) ✓ Hexadecimal Addresses Hexadecimal Values ASCII

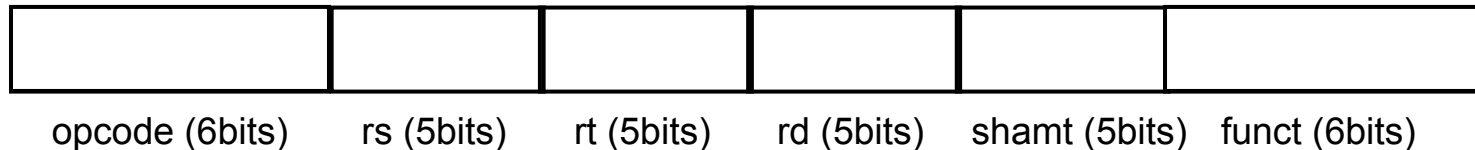
Mars Messages Run I/O

Assemble: assembling /Users/alicewang/MARS code/lecture 9 example.asm

Register List:

- \$zero
- \$at
- \$v0
- \$v1
- \$a0
- \$a1
- \$a2
- \$a3
- \$t0
- \$t1
- \$t2
- \$t3
- \$t4
- \$t5
- \$t6
- \$t7
- \$s0
- \$s1
- \$s2
- \$s3
- \$s4
- \$s5
- \$s6
- \$s7
- \$t8
- \$t9
- \$k0
- \$k1
- \$gp
- \$sp
- \$fp
- \$ra
- pc
- hi

R-type Instruction Example – Your Turn

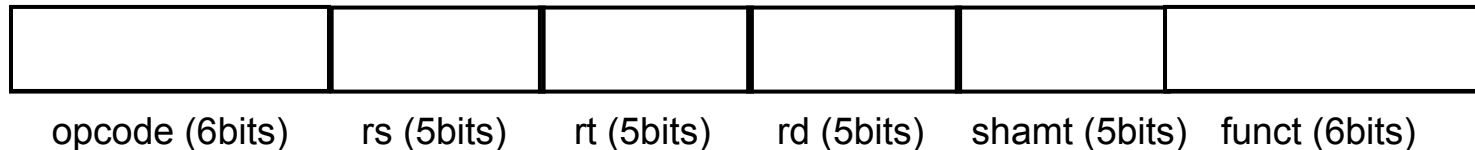


srl \$s0, \$t1, 2

Looking at the Green Card, determine the machine code:

1. What is the opcode/funct for srl?
2. What are the operands for srl? What is the order?
3. Translate from binary to hexadecimal

R-type Instruction Example – Your Turn



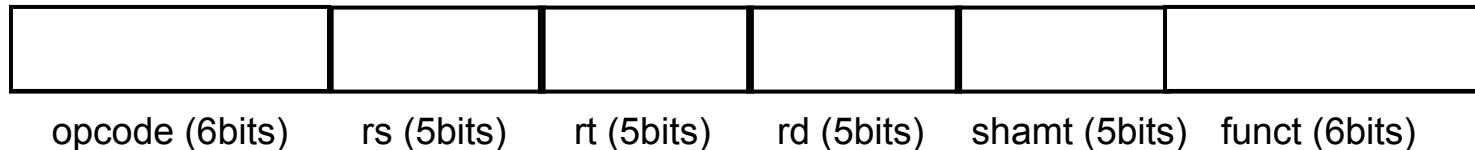
srl \$s0, \$t1, 2

1. What is the opcode/funct for srl?

CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (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}

R-type Instruction Example – Your Turn



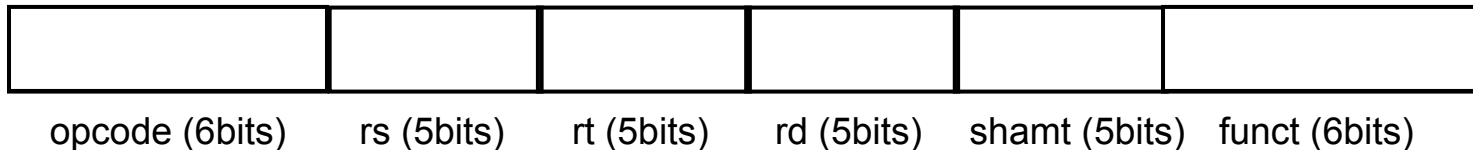
srl \$s0, \$t1, 2

1. What is the opcode/funct for srl?
2. What are the operands for srl? What is the order?

CORE INSTRUCTION SET

NAME, MNEMONIC	FOR- MAT	OPERATION (in Verilog)	OPCODE / FUNCT (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}

R-type Instruction Example – Your Turn



srl \$s0, \$t1, 2

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

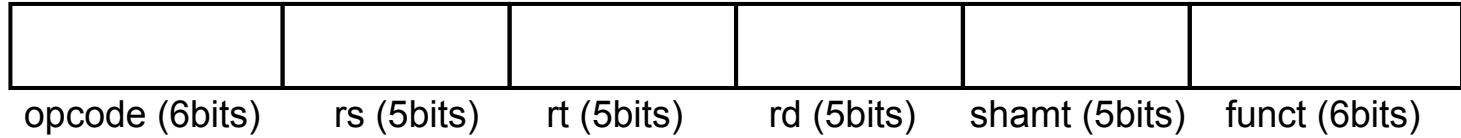
Use the table for register
mapping

\$s0 = _____ (0b_____)

\$t1 = _____ (0b_____)

R-type Instruction Example – Your Turn

Instruction
in Binary



Convert
to Hex

srl \$s0, \$t1, 2 → _____₁₆ in machine language

MARS assembler

Machine Code Basic Assembly Code (uses numbers for registers) Original Assembly Code (can contain pseudo-instructions)

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x02324020	add \$8,\$17,\$18	11: add \$t0, \$s1, \$s2
<input type="checkbox"/>	0x00400004	0x00098082	srl \$16,\$9,2	12: srl \$s0, \$t1, 2

Labels

Label	Address
lecture 9 example.asm	
something	0x10010000

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0	0	0	0	0	0	0	0
0x10010020	0	0	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0

0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

Mars Messages Run I/O

Assemble: operation completed successfully.

Reg

Name	Nu
\$zero	
\$at	
\$v0	
\$v1	
\$a0	
\$a1	
\$a2	
\$a3	
\$t0	
\$t1	
\$t2	
\$t3	
\$t4	
\$t5	
\$t6	
\$t7	
\$s0	
\$s1	
\$s2	
\$s3	
\$s4	
\$s5	
\$s6	
\$s7	
\$t8	
\$t9	
\$k0	
\$k1	
\$gp	
\$sp	
\$fp	
\$ra	
pc	
hi	
lo	

Summary

- Logical operations
 - `and`, `or`, `andi`, `ori`, `nor`
- Shifter operations
 - `sll`, `srl`, `sra`
- MIPS has 3 instruction formats
 - R-, I- and J- type
 - This time we covered R-type
 - Next time we will expand to I- and J- type

Next lecture

Machine Language – Part 2 (I- and J- type)

