# 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)/

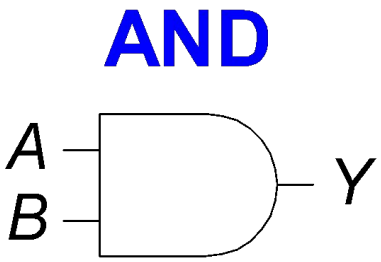
*noun*

**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**




$Y = AB$

| A | B | Y |
|---|---|---|
| 0 | 0 | |
| 0 | 1 | |
| 1 | 0 | |
| 1 | 1 | |

- AND is a fundamental Boolean operator

```
and $t0, $t1, $t2
```

- Useful to “mask” bits in a word

# What does it mean to “mask” bits

**mask**

/mask/

*noun*

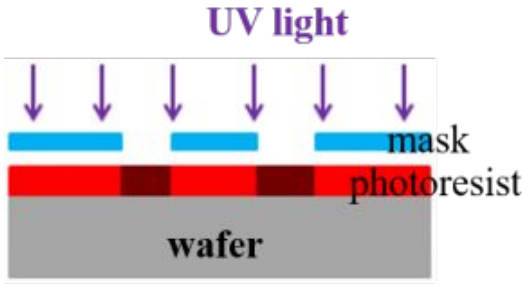
noun: **mask**; plural noun: **masks**; noun: **masque**; plural noun: **masques**

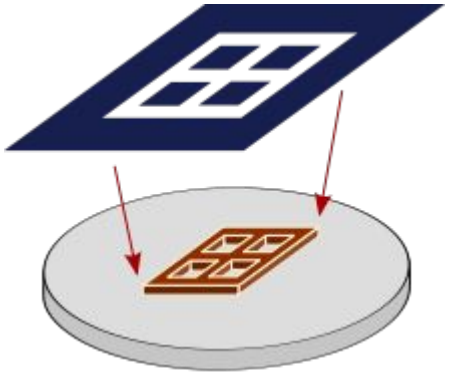
1. a covering for all or part of the face, worn as a disguise, or to amuse or terrify other people.

   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.

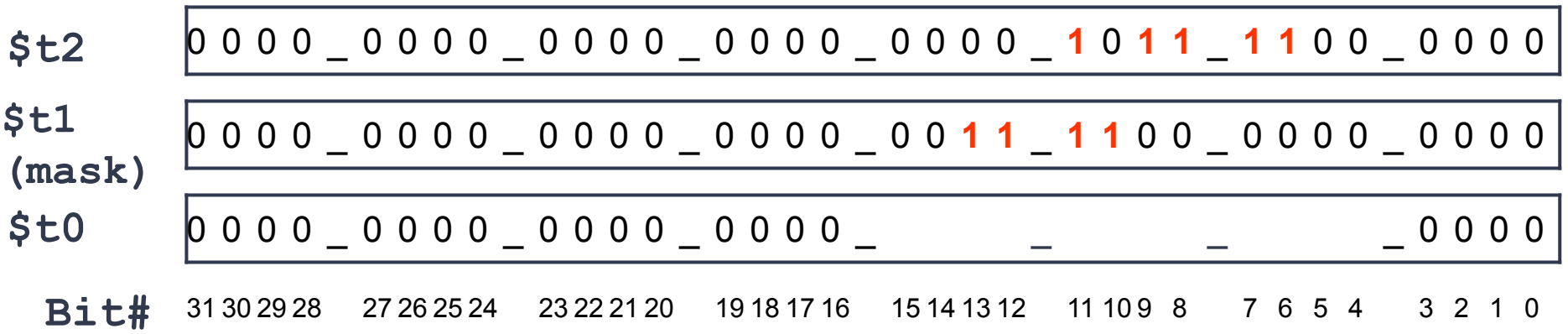
AND gate - if one input is

- 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
```

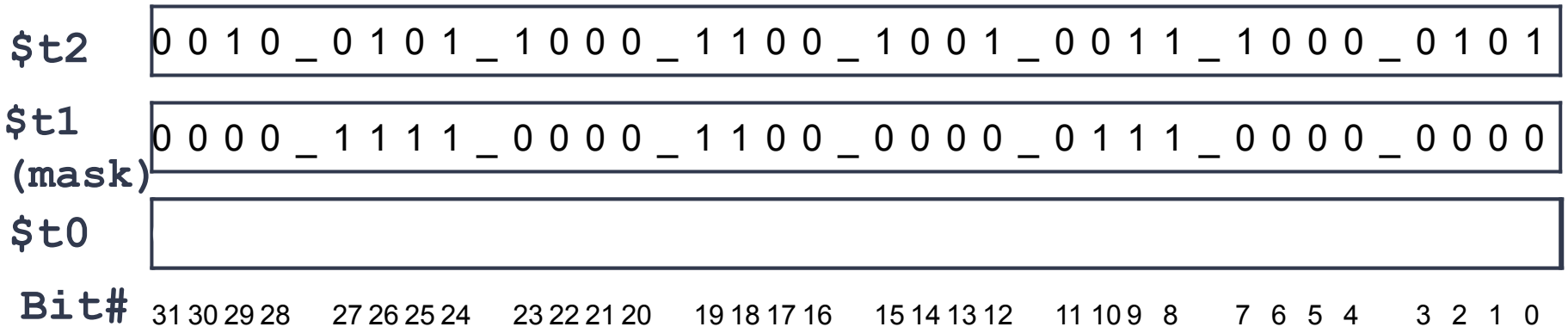


# AND operation Example - Your Turn

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

```
and $t0, $t1, $t2
```

Which bits in $t0 are ‘1’?



# OR operation

or[1]

/ôr/

*noun*

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

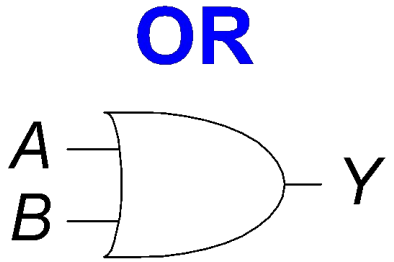
a Boolean operator that gives the value one if at least one operand (or input) has a value of one, and otherwise has a value of zero.

- **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



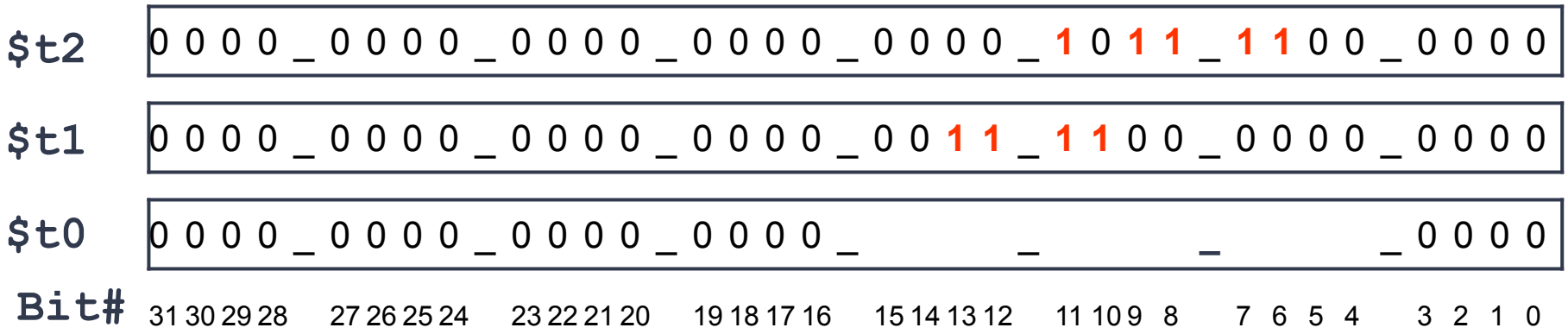

$$Y = A + B$$

| A | B | Y |
|---|---|---|
| 0 | 0 | |
| 0 | 1 | |
| 1 | 0 | |
| 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
```

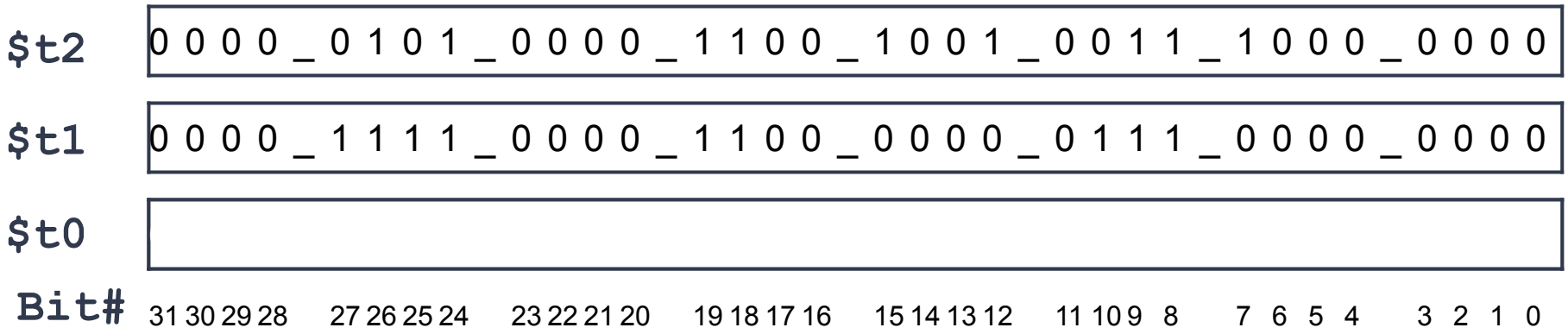


# OR operation Example - Your Turn

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

```
or $t0, $t1, $t2
```

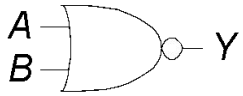
Which bits in $t0 are '1'?



# 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 NOR b == NOT ( a OR b )
  - `nor $t0, $t1, $zero`

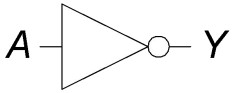
**NOR**




$Y = \overline{A + B}$

| 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

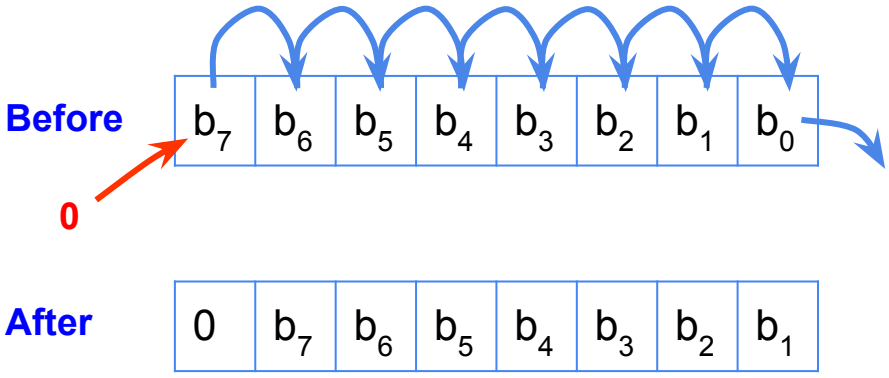
- One of the inputs is an immediate instead of a register
- Examples:

```
ori $t0, $t1, 5
andi $t0, $t1, 15
```

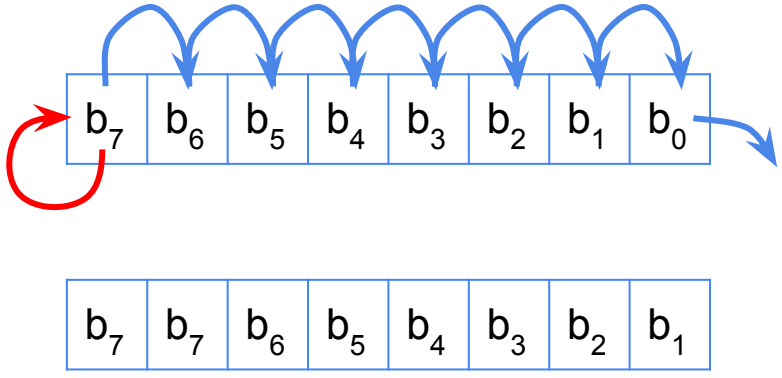
# Shifters

- 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**



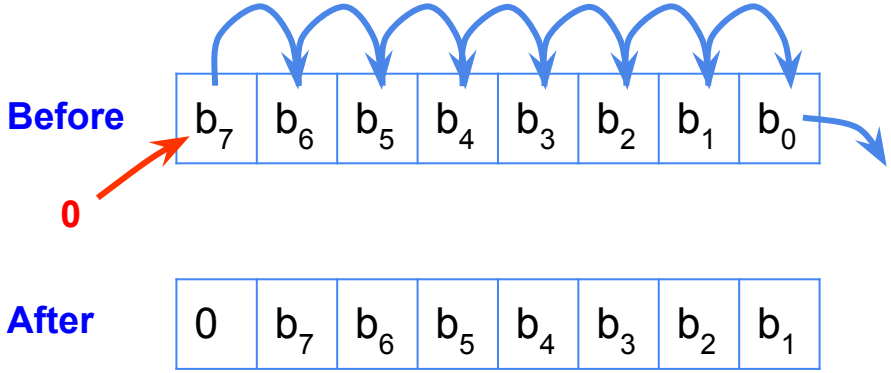
# Shifters - Logical Example - My Turn

- **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:

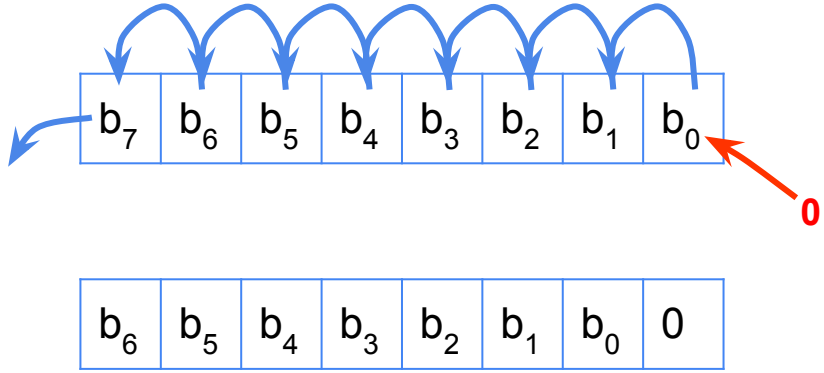
0b11001 >> 2 = 0b00110

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
  - `srl`: Shift right logical
  - `sll`: Shift left logical

8-bit Registers

| | |
|---|---|
| $t0 | 0x00 |
| $t1 | 0x0B |
| $t2 | 0x0C |
| $t3 | 0x0D |

```
sll $t0, $t1, 3
What is $t0 in binary and hex?
$t1    = 0b_____________________________
$t0    = 0b_____________________________
       = 0x_____________________
```

# Shifters - Logical Example - Your 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

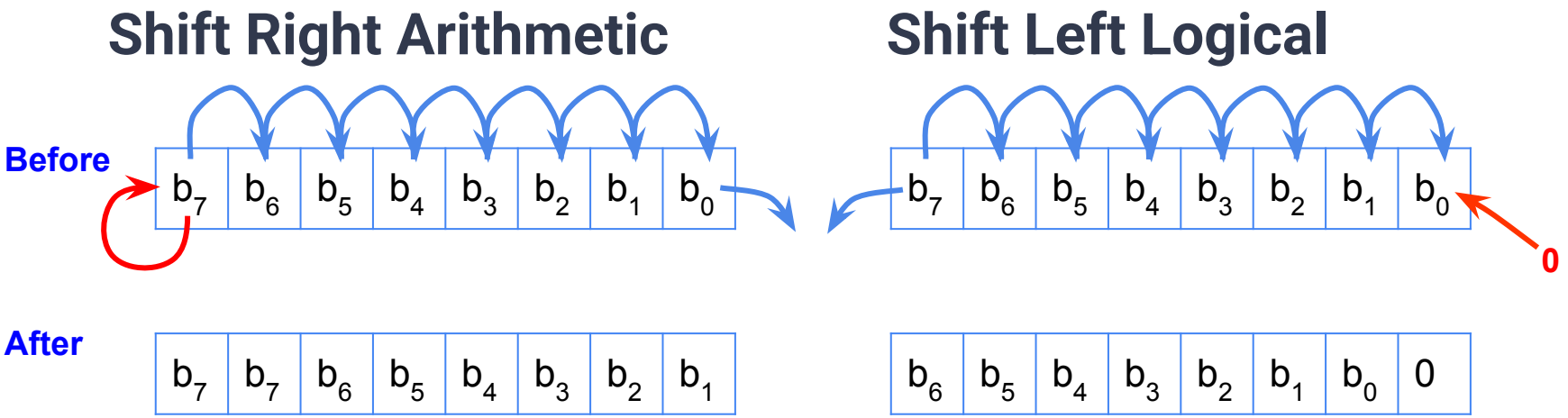
8-bit Registers

| | |
|---|---|
| $t0 | 0x00 |
| $t1 | 0x0B |
| $t2 | 0x0C |
| $t3 | 0x0D |

```
srl $t0, $t2, 2
What is $t0 in binary and hex?
$t2    = 0b________________________________
$t0    = 0b________________________________
       = 0x________
```

# Shifters - Arithmetic Example

- **Arithmetic shifter (>>>, <<<):** right shift (>>>), fills empty spaces with the old most significant bit (MSB), left shift (<<<) is the same as logical shifter (<<)
  - `sra`: Shift right arithmetic: 11001 >>> 2 = 11110
  - `sll` : Shift left logical: 11001 << 2 = 00100



# Where are shifters used?

1. Shifters as Multipliers and Dividers
   - Arithmetic shifters are useful for low cost and quick math
     - $A <<< N = A \times 2^N \rightarrow$ ***Multiply by powers of 2***
     - $A >>> N = A \div 2^N \rightarrow$ ***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

- $A << N = A \times 2^N$
  - **Example:** 0b00001 << 2 = 0b_____ (___ x ___ = ____)
    Decimal

- $A >>> N = A \div 2^N$
  - **Example:** 0b11000 >>> 2 = 0b_____ (___ ÷ ___ = ____)
    Decimal
    *Shift in the MSB to maintain sign (+)*

# Shifters as Mult and Div - Example - Your Turn

Assume Two's Complement numbers

- ***A << N = A × $2^N$***
  - **Example:** 0b111101 << 3 = 0b_____ (___ x ___ = ____)
    Decimal

- ***A >>> N = A ÷ $2^N$***
  - **Example:** 0b001000 >>> 1 = 0b_____ (___ ÷ ___ = ____)
    Decimal
    *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

# MIPS Reference Card – pdf on elearning

## you can bring this to the exam

1. Pull along perforation to separate card  2. Fold bottom side (columns 3 and 4) together

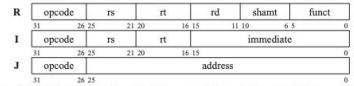
# MIPS Reference Data ①

## 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 / $23_{hex}$ |

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

## BASIC INSTRUCTION FORMATS





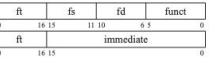

## ARITHMETIC CORE INSTRUCTION SET ②

| NAME, MNEMONIC | | FOR-MAT | OPERATION | OPCODE / FMT /FT / FUNCT (Hex) |
|---|---|---|---|---|
| Branch On FP True | bc1t | FI | if(FPcond)PC=PC+4+BranchAddr (4) | 11/8/1/-- |
| Branch On FP False | bc1f | 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] (6) | 0/--/--/1b |
| FP Add Single | add.s | FR | F[fd ]= F[fs] + F[ft] | 11/10/--/0 |
| FP Add Double | add.d | FR | {F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]} | 11/11/--/0 |
| FP Compare Single | c.x.s* | FR | FPcond = (F[fs] *op* F[ft]) ? 1 : 0 | 11/10/--/y |
| FP Compare Double | c.x.d* | FR | FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0 | 11/11/--/y |

* (*x* is eq, lt, or le) (*op* is ==, <, or <=) ( *y* is 32, 3c, or 3e)

| | | | | |
|---|---|---|---|---|
| FP Divide Single | div.s | FR | F[fd] = F[fs] / F[ft] | 11/10/--/3 |
| FP Divide Double | div.d | FR | {F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]} | 11/11/--/3 |
| FP Multiply Single | mul.s | FR | F[fd] = F[fs] * F[ft] | 11/10/--/2 |
| FP Multiply Double | mul.d | FR | {F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]} | 11/11/--/2 |
| FP Subtract Single | sub.s | FR | F[fd]=F[fs] - F[ft] | 11/10/--/1 |
| FP Subtract Double | sub.d | FR | {F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]} | 11/11/--/1 |
| Load FP Single | lwc1 | I | F[rt]=M[R[rs]+SignExtImm] (2) | 31/--/--/-- |
| Load FP Double | ldc1 | I | F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4] (2) | 35/--/--/-- |
| Move From Hi | mfhi | R | R[rd] = Hi | 0/--/--/10 |
| Move From Lo | mflo | R | R[rd] = Lo | 0/--/--/12 |
| Move From Control | mfc0 | R | R[rd] = CR[rs] | 10 /0/--/0 |
| Multiply | mult | R | {Hi,Lo} = R[rs] * R[rt] | 0/--/--/18 |
| Multiply Unsigned | multu | R | {Hi,Lo} = R[rs] * R[rt] (6) | 0/--/--/19 |
| Shift Right Arith. | sra | R | R[rd] = R[rt] >>> shamt | 0/--/--/3 |
| Store FP Single | swc1 | I | M[R[rs]+SignExtImm] = F[rt] (2) | 39/--/--/-- |
| Store FP Double | sdc1 | I | M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1] (2) | 3d/--/--/-- |

## 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 | 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 ③

| MIPS opcode (31:26) | (1) MIPS funct (5:0) | (2) MIPS funct (5:0) | Binary | Decimal | Hexadecimal | ASCII Character | Decimal | Hexadecimal | ASCII Character |
|---|---|---|---|---|---|---|---|---|---|
| (1) | sll | add.f | 00 0000 | 0 | 0 | NUL | 64 | 40 | @ |
| | | sub.f | 00 0001 | 1 | 1 | SOH | 65 | 41 | A |
| j | srl | mul.f | 00 0010 | 2 | 2 | STX | 66 | 42 | B |
| jal | sra | div.f | 00 0011 | 3 | 3 | ETX | 67 | 43 | C |
| beq | sllv | sqrt.f | 00 0100 | 4 | 4 | EOT | 68 | 44 | D |
| bne | | abs.f | 00 0101 | 5 | 5 | ENQ | 69 | 45 | E |
| blez | srlv | mov.f | 00 0110 | 6 | 6 | ACK | 70 | 46 | F |
| bgtz | srav | neg.f | 00 0111 | 7 | 7 | BEL | 71 | 47 | G |
| addi | jr | | 00 1000 | 8 | 8 | BS | 72 | 48 | H |
| addiu | jalr | | 00 1001 | 9 | 9 | HT | 73 | 49 | I |
| slti | movz | | 00 1010 | 10 | a | LF | 74 | 4a | J |
| sltiu | movn | | 00 1011 | 11 | b | VT | 75 | 4b | K |
| andi | syscall | round.w.f | 00 1100 | 12 | c | FF | 76 | 4c | L |
| ori | break | trunc.w.f | 00 1101 | 13 | d | CR | 77 | 4d | M |
| xori | | ceil.w.f | 00 1110 | 14 | e | SO | 78 | 4e | N |
| lui | sync | floor.w.f | 00 1111 | 15 | f | SI | 79 | 4f | O |
| | mfhi | | 01 0000 | 16 | 10 | DLE | 80 | 50 | P |
| (2) | mthi | | 01 0001 | 17 | 11 | DC1 | 81 | 51 | Q |
| | mflo | movs.f | 01 0010 | 18 | 12 | DC2 | 82 | 52 | R |
| | mtlo | movn.f | 01 0011 | 19 | 13 | DC3 | 83 | 53 | S |
| | | | 01 0100 | 20 | 14 | DC4 | 84 | 54 | T |
| | | | 01 0101 | 21 | 15 | NAK | 85 | 55 | U |
| | | | 01 0110 | 22 | 16 | SYN | 86 | 56 | V |
| | | | 01 0111 | 23 | 17 | ETB | 87 | 57 | W |
| | mult | | 01 1000 | 24 | 18 | CAN | 88 | 58 | X |
| | multu | | 01 1001 | 25 | 19 | EM | 89 | 59 | Y |
| | div | | 01 1010 | 26 | 1a | SUB | 90 | 5a | Z |
| | divu | | 01 1011 | 27 | 1b | ESC | 91 | 5b | [ |
| | | | 01 1100 | 28 | 1c | FS | 92 | 5c | \ |
| | | | 01 1101 | 29 | 1d | GS | 93 | 5d | ] |
| | | | 01 1110 | 30 | 1e | RS | 94 | 5e | ^ |
| | | | 01 1111 | 31 | 1f | US | 95 | 5f | _ |
| lb | add | cvt.s.f | 10 0000 | 32 | 20 | Space | 96 | 60 | ` |
| lh | addu | cvt.d.f | 10 0001 | 33 | 21 | ! | 97 | 61 | a |
| lwl | sub | | 10 0010 | 34 | 22 | " | 98 | 62 | b |
| lw | subu | | 10 0011 | 35 | 23 | # | 99 | 63 | c |
| lbu | and | cvt.w.f | 10 0100 | 36 | 24 | $ | 100 | 64 | d |
| lhu | or | | 10 0101 | 37 | 25 | % | 101 | 65 | e |
| lwr | xor | | 10 0110 | 38 | 26 | & | 102 | 66 | f |
| | nor | | 10 0111 | 39 | 27 | ' | 103 | 67 | g |
| | | | 10 1000 | 40 | 28 | ( | 104 | 68 | h |
| sb | | | 10 1001 | 41 | 29 | ) | 105 | 69 | i |
| swl | slt | | 10 1010 | 42 | 2a | * | 106 | 6a | j |
| sw | sltu | | 10 1011 | 43 | 2b | + | 107 | 6b | k |
| | | | 10 1100 | 44 | 2c | , | 108 | 6c | l |
| | | | 10 1101 | 45 | 2d | - | 109 | 6d | m |
| swr | | | 10 1110 | 46 | 2e | . | 110 | 6e | n |
| cache | | | 10 1111 | 47 | 2f | / | 111 | 6f | o |
| ll | tge | c.f.f | 11 0000 | 48 | 30 | 0 | 112 | 70 | p |
| lwc1 | tgeu | c.un.f | 11 0001 | 49 | 31 | 1 | 113 | 71 | q |
| lwc2 | tlt | c.eq.f | 11 0010 | 50 | 32 | 2 | 114 | 72 | r |
| pref | tltu | c.ueq.f | 11 0011 | 51 | 33 | 3 | 115 | 73 | s |
| | teq | c.olt.f | 11 0100 | 52 | 34 | 4 | 116 | 74 | t |
| ldc1 | | c.ult.f | 11 0101 | 53 | 35 | 5 | 117 | 75 | u |
| ldc2 | tne | c.ole.f | 11 0110 | 54 | 36 | 6 | 118 | 76 | v |
| | | c.ule.f | 11 0111 | 55 | 37 | 7 | 119 | 77 | w |
| sc | | c.sf.f | 11 1000 | 56 | 38 | 8 | 120 | 78 | x |
| swc1 | | c.ngle.f | 11 1001 | 57 | 39 | 9 | 121 | 79 | y |
| swc2 | | c.seq.f | 11 1010 | 58 | 3a | : | 122 | 7a | z |
| | | c.ngl.f | 11 1011 | 59 | 3b | ; | 123 | 7b | { |
| | | c.lt.f | 11 1100 | 60 | 3c | < | 124 | 7c | \| |
| sdc1 | | c.nge.f | 11 1101 | 61 | 3d | = | 125 | 7d | } |
| sdc2 | | c.le.f | 11 1110 | 62 | 3e | > | 126 | 7e | ~ |
| | | c.ngt.f | 11 1111 | 63 | 3f | ? | 127 | 7f | DEL |

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



## IEEE 754 FLOATING-POINT STANDARD ④

$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

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

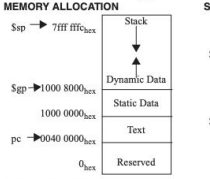
### IEEE 754 Symbols

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

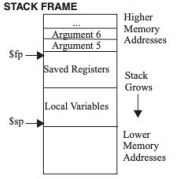
S.P. MAX = 255, D.P. MAX = 2047

### IEEE Single Precision and Double Precision Formats:

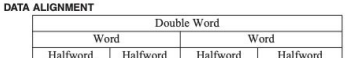

### MEMORY ALLOCATION




### STACK FRAME

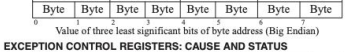



### DATA ALIGNMENT




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

### EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS




BD = Branch Delay, UM = User Mode, EL = Exception Level, IE =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 |
| 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 ($10^x$ for Disk, Communication; $2^x$ for Memory)

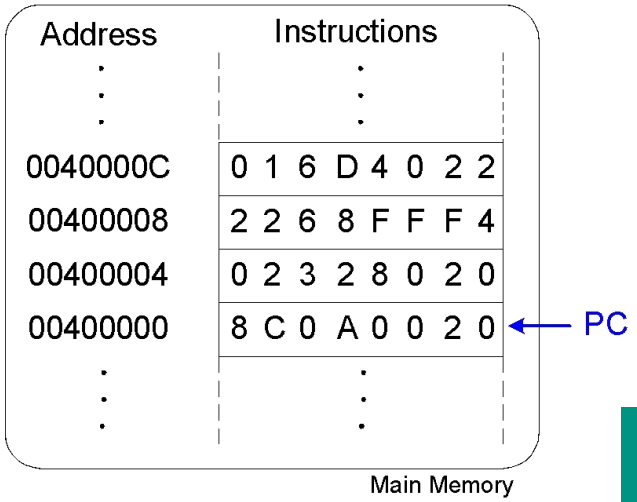
| SI Size | Prefix | Symbol | IEC Size | Prefix | Symbol |
|---|---|---|---|---|---|
| $10^3$ | Kilo- | K | $2^{10}$ | Kibi- | Ki |
| $10^6$ | Mega- | M | $2^{20}$ | Mebi- | Mi |
| $10^9$ | Giga- | G | $2^{30}$ | Gibi- | Gi |
| $10^{12}$ | Tera- | T | $2^{40}$ | Tebi- | Ti |
| $10^{15}$ | Peta- | P | $2^{50}$ | 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 |

1. Pull along perforation to separate card  2. Fold bottom side (columns 3 and 4) together

# The Stored Program

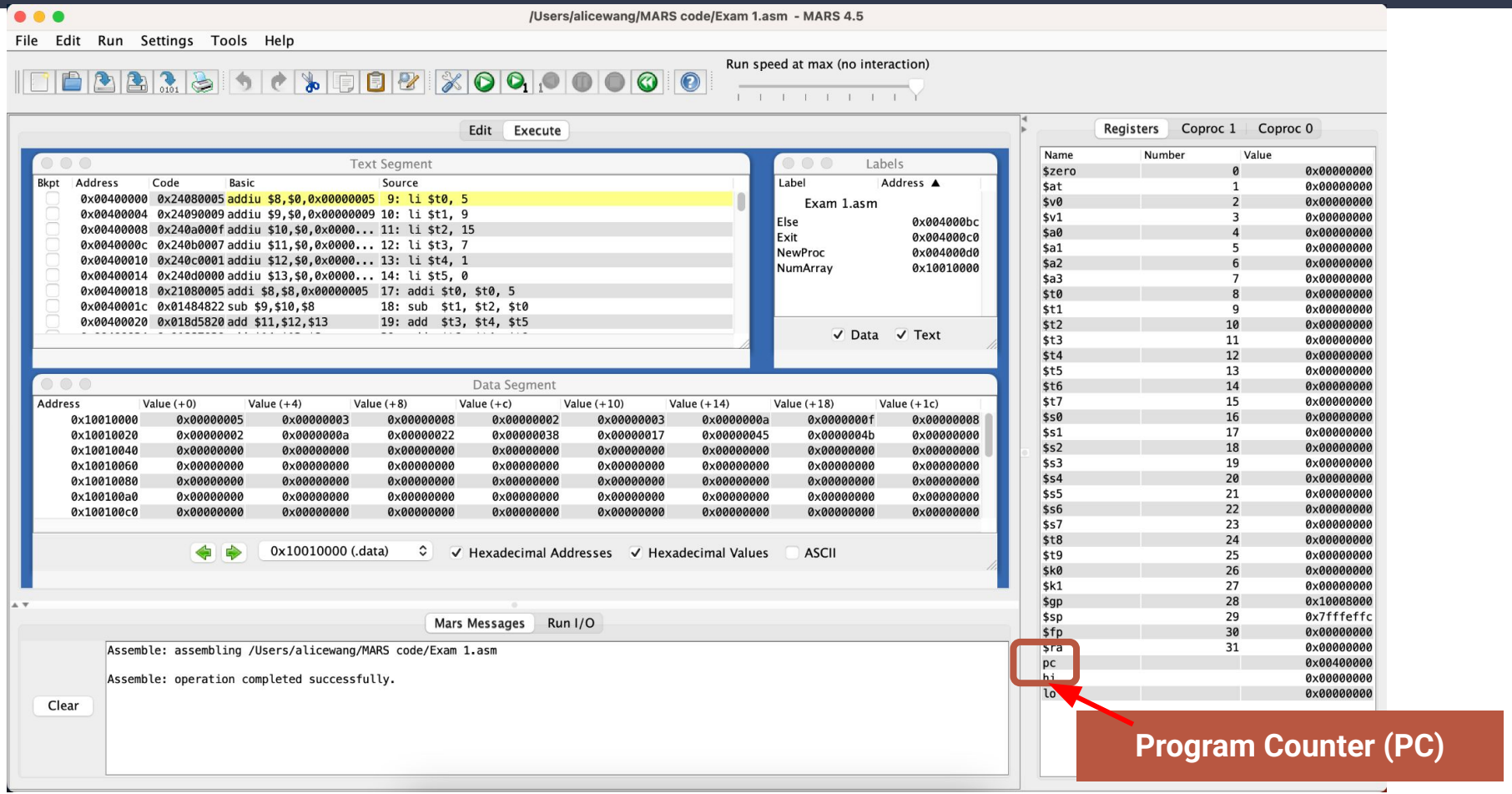
| Assembly Code | Machine Code |
|---|---|
| `lw $t2, 32($0)` | `0x8C0A0020` |
| `add $s0, $s1, $s2` | `0x02328020` |
| `addi $t0, $s3, -12` | `0x2268FFF4` |
| `sub $t0, $t3, $t5` | `0x016D4022` |

Stored Program

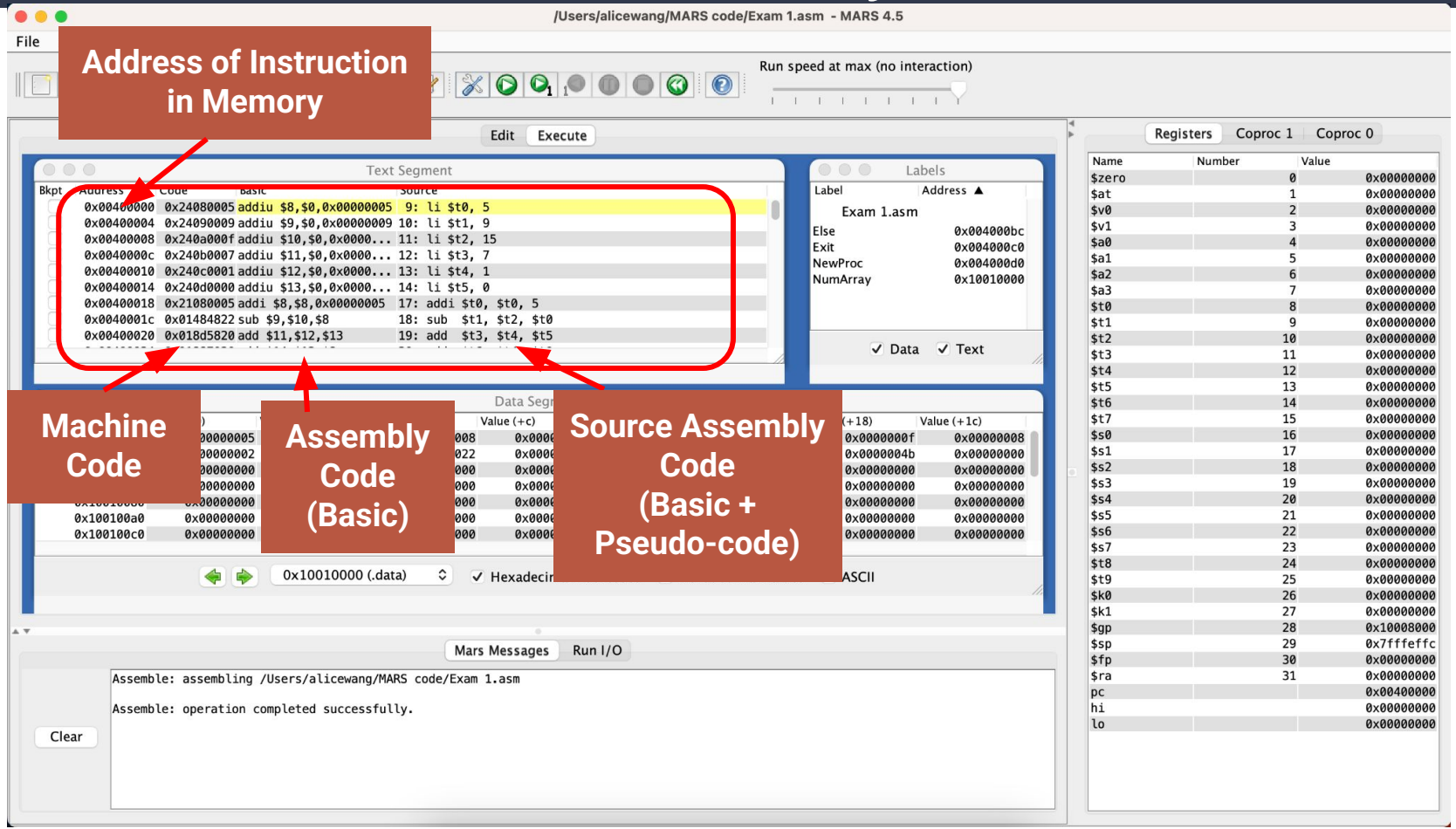



- 32-bit instructions & data stored in memory
- Sequence of instructions: only difference between two applications
- To run a new program:
  - No rewiring required
  - Simply store new program in memory (binary)
- "binary" would become shorthand for "binary executable compiler output"
  - Directories 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!**


/Users/alicewang/MARS code/Exam 1.asm - MARS 4.5
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 Text
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
0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII
Mars Messages Run I/O
Assemble: assembling /Users/alicewang/MARS code/Exam 1.asm
Assemble: operation completed successfully.
Clear
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 0x7fffeffc
$fp 30 0x00000000
$ra 31 0x00000000
pc 0x00400000
hi 0x00000000
lo 0x00000000
Program Counter (PC)

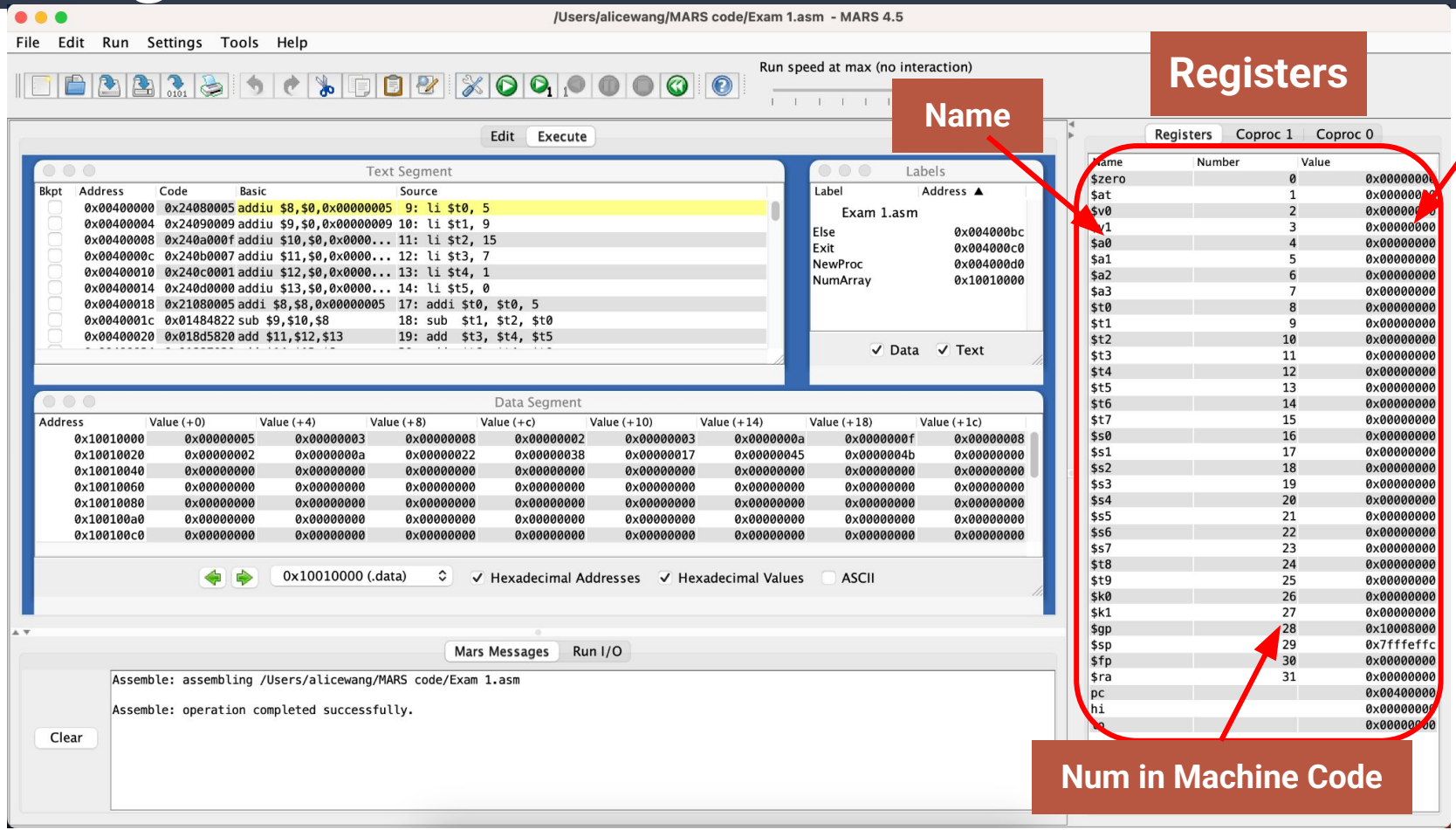
# MARS instruction memory ".text"



# 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

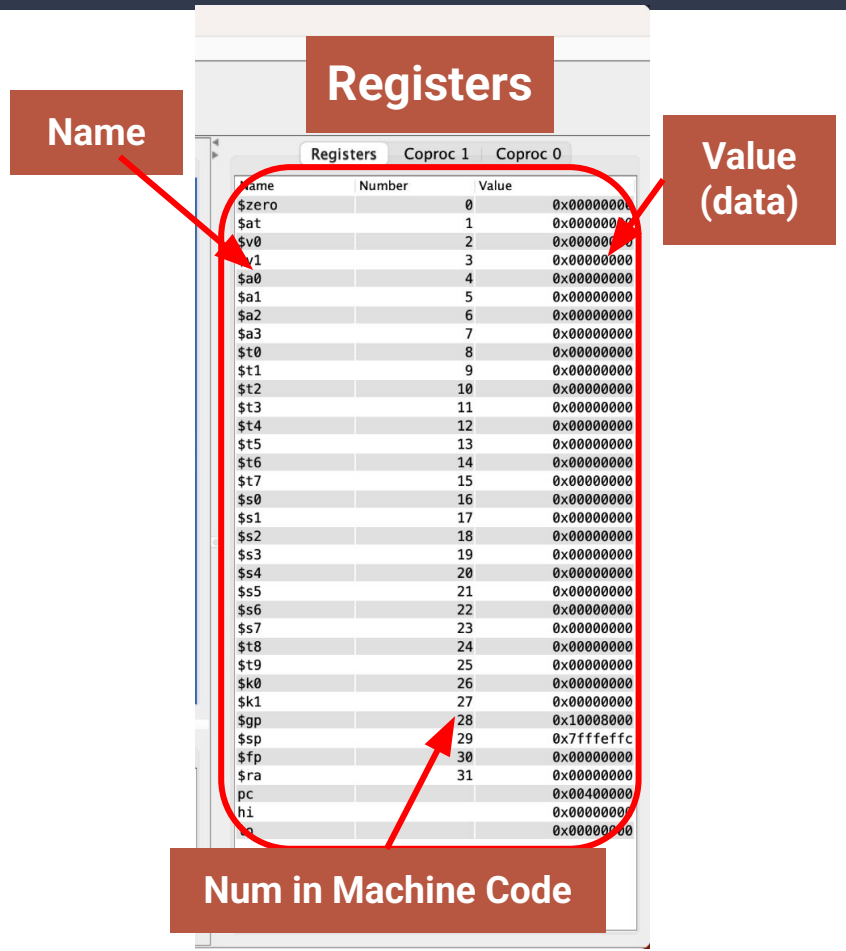
# Register Table in MARS



# 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 |



# Register Table – location on the MIPS card

1. Pull along perforation to separate card  2. Fold bottom side (columns 3 and 4) together

MIPS Reference Data Card ("Green Card")

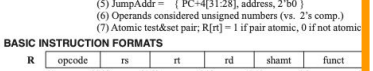
## MIPS Reference Data ①

### 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 / $23_{hex}$ |

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

### BASIC INSTRUCTION FORMATS

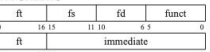





### ARITHMETIC CORE INSTRUCTION SET ②

| NAME, MNEMONIC | | FOR-MAT | OPERATION | OPCODE / FMT /FT / FUNCT (Hex) |
|---|---|---|---|---|
| Branch On FP True | bc1t | FI | if(FPcond)PC=PC+4+BranchAddr (4) | 11/8/1/-- |
| Branch On FP False | bc1f | 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] (6) | 0/--/--/1b |
| FP Add Single | add.s | FR | F[fd ]= F[fs] + F[ft] | 11/10/--/0 |
| FP Add Double | add.d | FR | {F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]} | 11/11/--/0 |
| FP Compare Single | c.x.s* | FR | FPcond = (F[fs] *op* F[ft]) ? 1 : 0 | 11/10/--/y |
| FP Compare Double | c.x.d* | FR | FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0 | 11/11/--/y |
| FP Divide Single | div.s | FR | F[fd] = F[fs] / F[ft] | 11/10/--/3 |
| FP Divide Double | div.d | FR | {F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]} | 11/11/--/3 |
| FP Multiply Single | mul.s | FR | F[fd] = F[fs] * F[ft] | 11/10/--/2 |
| FP Multiply Double | mul.d | FR | {F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]} | 11/11/--/2 |
| FP Subtract Single | sub.s | FR | F[fd]=F[fs] - F[ft] | 11/10/--/1 |
| FP Subtract Double | sub.d | FR | {F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]} | 11/11/--/1 |
| Load FP Single | lwc1 | I | F[rt]=M[R[rs]+SignExtImm] (2) | 31/--/--/-- |
| Load FP Double | ldc1 | I | F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4] (2) | 35/--/--/-- |
| Move From Hi | mfhi | R | R[rd] = Hi | 0/--/--/10 |
| Move From Lo | mflo | R | R[rd] = Lo | 0/--/--/12 |
| Move From Control | mfc0 | R | R[rd] = CR[rs] | 10 /0/--/0 |
| Multiply | mult | R | {Hi,Lo} = R[rs] * R[rt] | 0/--/--/18 |
| Multiply Unsigned | multu | R | {Hi,Lo} = R[rs] * R[rt] (6) | 0/--/--/19 |
| Shift Right Arith. | sra | R | R[rd] = R[rt] >>> shamt | 0/--/--/3 |
| Store FP Single | swc1 | I | M[R[rs]+SignExtImm] = F[rt] (2) | 39/--/--/-- |
| Store FP Double | sdc1 | I | M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1] (2) | 3d/--/--/-- |

### 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 | 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 ③

| MIPS opcode (31:26) | (1) MIPS funct (5:0) | (2) MIPS funct (5:0) | Binary | Decimal | Hexadecimal | ASCII Character | Decimal | Hexadecimal | ASCII Character |
|---|---|---|---|---|---|---|---|---|---|
| (1) | sll | add.f | 00 0000 | 0 | 0 | NUL | 64 | 40 | @ |
| | | sub.f | 00 0001 | 1 | 1 | SOH | 65 | 41 | A |
| j | srl | mul.f | 00 0010 | 2 | 2 | STX | 66 | 42 | B |
| jal | sra | div.f | 00 0011 | 3 | 3 | ETX | 67 | 43 | C |
| beq | sllv | sqrt.f | 00 0100 | 4 | 4 | EOT | 68 | 44 | D |
| bne | | abs.f | 00 0101 | 5 | 5 | ENQ | 69 | 45 | E |
| blez | srlv | mov.f | 00 0110 | 6 | 6 | ACK | 70 | 46 | F |
| bgtz | srav | neg.f | 00 0111 | 7 | 7 | BEL | 71 | 47 | G |
| addi | jr | | 00 1000 | 8 | 8 | BS | 72 | 48 | H |
| addiu | jalr | | 00 1001 | 9 | 9 | HT | 73 | 49 | I |
| slti | movz | | 00 1010 | 10 | a | LF | 74 | 4a | J |
| sltiu | movn | | 00 1011 | 11 | b | VT | 75 | 4b | K |
| andi | syscall | round.w.f | 00 1100 | 12 | c | FF | 76 | 4c | L |
| ori | break | trunc.w.f | 00 1101 | 13 | d | CR | 77 | 4d | M |
| xori | | ceil.w.f | 00 1110 | 14 | e | SO | 78 | 4e | N |
| lui | sync | floor.w.f | 00 1111 | 15 | f | SI | 79 | 4f | O |
| | mfhi | | 01 0000 | 16 | 10 | DLE | 80 | 50 | P |
| (2) | mthi | | 01 0001 | 17 | 11 | DC1 | 81 | 51 | Q |
| | mflo | movs.f | 01 0010 | 18 | 12 | DC2 | 82 | 52 | R |
| | mtlo | movn.f | 01 0011 | 19 | 13 | DC3 | 83 | 53 | S |
| | | | 01 0100 | 20 | 14 | DC4 | 84 | 54 | T |
| | | | 01 0101 | 21 | 15 | NAK | 85 | 55 | U |
| | | | 01 0110 | 22 | 16 | SYN | 86 | 56 | V |
| | | | 01 0111 | 23 | 17 | ETB | 87 | 57 | W |
| | mult | | 01 1000 | 24 | 18 | CAN | 88 | 58 | X |
| | multu | | 01 1001 | 25 | 19 | EM | 89 | 59 | Y |
| | div | | 01 1010 | 26 | 1a | SUB | 90 | 5a | Z |
| | divu | | 01 1011 | 27 | 1b | ESC | 91 | 5b | [ |
| | | | 01 1100 | 28 | 1c | FS | 92 | 5c | \ |
| | | | 01 1101 | 29 | 1d | GS | 93 | 5d | ] |
| | | | 01 1110 | 30 | 1e | RS | 94 | 5e | ^ |
| | | | 01 1111 | 31 | 1f | US | 95 | 5f | _ |
| lb | add | cvt.s.f | 10 0000 | 32 | 20 | Space | 96 | 60 | ` |
| lh | addu | cvt.d.f | 10 0001 | 33 | 21 | ! | 97 | 61 | a |
| lwl | sub | | 10 0010 | 34 | 22 | " | 98 | 62 | b |
| lw | subu | | 10 0011 | 35 | 23 | # | 99 | 63 | c |
| lbu | and | cvt.w.f | 10 0100 | 36 | 24 | $ | 100 | 64 | d |
| lhu | or | | 10 0101 | 37 | 25 | % | 101 | 65 | e |
| lwr | xor | | 10 0110 | 38 | 26 | & | 102 | 66 | f |
| | nor | | 10 0111 | 39 | 27 | ' | 103 | 67 | g |
| | | | 10 1000 | 40 | 28 | ( | 104 | 68 | h |
| sb | | | 10 1001 | 41 | 29 | ) | 105 | 69 | i |
| swl | slt | | 10 1010 | 42 | 2a | * | 106 | 6a | j |
| sw | sltu | | 10 1011 | 43 | 2b | + | 107 | 6b | k |
| | | | 10 1100 | 44 | 2c | , | 108 | 6c | l |
| | | | 10 1101 | 45 | 2d | - | 109 | 6d | m |
| swr | | | 10 1110 | 46 | 2e | . | 110 | 6e | n |
| cache | | | 10 1111 | 47 | 2f | / | 111 | 6f | o |
| ll | tge | c.f.f | 11 0000 | 48 | 30 | 0 | 112 | 70 | p |
| lwc1 | tgeu | c.un.f | 11 0001 | 49 | 31 | 1 | 113 | 71 | q |
| lwc2 | tlt | c.eq.f | 11 0010 | 50 | 32 | 2 | 114 | 72 | r |
| pref | tltu | c.ueq.f | 11 0011 | 51 | 33 | 3 | 115 | 73 | s |
| | teq | c.olt.f | 11 0100 | 52 | 34 | 4 | 116 | 74 | t |
| ldc1 | | c.ult.f | 11 0101 | 53 | 35 | 5 | 117 | 75 | u |
| ldc2 | tne | c.ole.f | 11 0110 | 54 | 36 | 6 | 118 | 76 | v |
| | | c.ule.f | 11 0111 | 55 | 37 | 7 | 119 | 77 | w |
| sc | | c.sf.f | 11 1000 | 56 | 38 | 8 | 120 | 78 | x |
| swc1 | | c.ngle.f | 11 1001 | 57 | 39 | 9 | 121 | 79 | y |
| swc2 | | c.seq.f | 11 1010 | 58 | 3a | : | 122 | 7a | z |
| | | c.ngl.f | 11 1011 | 59 | 3b | ; | 123 | 7b | { |
| | | c.lt.f | 11 1100 | 60 | 3c | < | 124 | 7c | \| |
| sdc1 | | c.nge.f | 11 1101 | 61 | 3d | = | 125 | 7d | } |
| sdc2 | | c.le.f | 11 1110 | 62 | 3e | > | 126 | 7e | ~ |
| | | c.ngt.f | 11 1111 | 63 | 3f | ? | 127 | 7f | DEL |

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



### IEEE 754 FLOATING-POINT STANDARD ④

$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

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

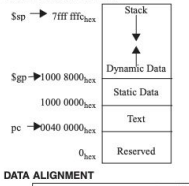
**IEEE Single Precision and Double Precision Formats:**


**IEEE 754 Symbols**

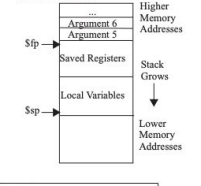
| Exponent | Fraction | Object |
|---|---|---|
| 0 | 0 | ± 0 |
| 0 | ≠0 | ± Denorm |
| 1 to MAX - 1 | anything | ± Fl. Pt. Num. |
| MAX | 0 | ±∞ |
| MAX | ≠0 | NaN |

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

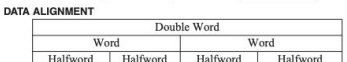
### MEMORY ALLOCATION




### STACK FRAME

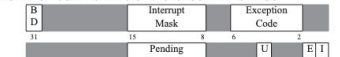



### DATA ALIGNMENT




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

### EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS




BD = Branch Delay, UM = User Mode, EL = Exception Level, IE =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 |
| 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 ($10^x$ for Disk, Communication; $2^x$ 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 |

1. Pull along perforation to separate card  2. Fold bottom side (columns 3 and 4) together

MIPS Reference Data Card ("Green Card")

# Core – Instructions / Operations

1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

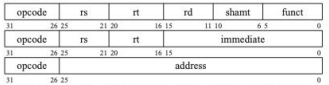
## MIPS Reference Data ①

### 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 / $23_{hex}$ |

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

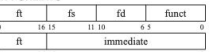
### BASIC INSTRUCTION FORMATS




### ARITHMETIC CORE INSTRUCTION SET ②

| NAME, MNEMONIC | FOR-MAT | OPERATION | OPCODE / FMT /FT / FUNCT (Hex) |
|---|---|---|---|
| Branch On FP True | bc1t FI | if(FPcond)PC=PC+4+BranchAddr (4) | 11/8/1/-- |
| Branch On FP False | bc1f 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] (6) | 0/--/--/1b |
| FP Add Single | add.s FR | F[fd ]= F[fs] + F[ft] | 11/10/--/0 |
| FP Add Double | add.d FR | {F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]} | 11/11/--/0 |
| FP Compare Single | c.x.s* FR | FPcond = (F[fs] *op* F[ft]) ? 1 : 0 | 11/10/--/*y* |
| FP Compare Double | c.x.d* FR | FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0 | 11/11/--/*y* |
| FP Div | | | |
| FP Div | | | |
| Doubl | | | |
| FP Mu | | | |
| FP Mu | | | |
| Doubl | | | |
| FP Sul | | | |
| FP Sul | | | |
| Doubl | | | |
| Load F | | | |
| Load F | | | |
| Doubl | | | |
| Move | | | |
| Move From Lo | mflo R | R[rd] = Lo | 0/--/--/12 |
| Move From Control | mfc0 R | R[rd] = CR[rs] | 10 /0/--/0 |
| Multiply | mult R | {Hi,Lo} = R[rs] * R[rt] | 0/--/--/18 |
| Multiply Unsigned | multu R | {Hi,Lo} = R[rs] * R[rt] (6) | 0/--/--/19 |
| Shift Right Arith. | sra R | R[rd] = R[rt] >>> shamt | 0/--/--/3 |
| Store FP Single | swc1 I | M[R[rs]+SignExtImm] = F[rt]; (2) | 39/--/--/-- |
| Store FP Double | sdc1 I | M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1] (2) | 3d/--/--/-- |

### 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 | 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 |



**We have learned almost every instruction in this column. Let's zoom in…**

### OPCODES, BASE CONVERSION, ASCII SYMBOLS ③

| MIPS opcode (31:26) | (1) MIPS funct (5:0) | (2) MIPS funct (5:0) | Binary | Decimal | Hexadecimal | ASCII Character | Decimal | Hexadecimal | ASCII Character |
|---|---|---|---|---|---|---|---|---|---|
| (1) | sll | add.f | 00 0000 | 0 | 0 | NUL | 64 | 40 | @ |
| | | sub.f | 00 0001 | 1 | 1 | SOH | 65 | 41 | A |
| j | srl | mul.f | 00 0010 | 2 | 2 | STX | 66 | 42 | B |
| jal | sra | div.f | 00 0011 | 3 | 3 | ETX | 67 | 43 | C |
| beq | sllv | sqrt.f | 00 0100 | 4 | 4 | EOT | 68 | 44 | D |
| bne | | abs.f | 00 0101 | 5 | 5 | ENQ | 69 | 45 | E |
| blez | srlv | mov.f | 00 0110 | 6 | 6 | ACK | 70 | 46 | F |
| bgtz | srav | neg.f | 00 0111 | 7 | 7 | BEL | 71 | 47 | G |
| addi | jr | | 00 1000 | 8 | 8 | BS | 72 | 48 | H |
| addiu | jalr | | 00 1001 | 9 | 9 | HT | 73 | 49 | I |
| slti | movz | | 00 1010 | 10 | a | LF | 74 | 4a | J |
| * | * | | 00 1011 | 11 | b | VT | 75 | 4b | K |
| | | round.w.f | 00 1100 | 12 | c | FF | 76 | 4c | L |
| | | trunc.w.f | 00 1101 | 13 | d | CR | 77 | 4d | M |
| | | ceil.w.f | 00 1110 | 14 | e | SO | 78 | 4e | N |
| | | floor.w.f | 00 1111 | 15 | f | SI | 79 | 4f | O |
| | | | 01 0000 | 16 | 10 | DLE | 80 | 50 | P |
| | | | 01 0001 | 17 | 11 | DC1 | 81 | 51 | Q |
| | | movs.f | 01 0010 | 18 | 12 | DC2 | 82 | 52 | R |
| | | movn.f | 01 0011 | 19 | 13 | DC3 | 83 | 53 | S |
| | | | 01 0100 | 20 | 14 | DC4 | 84 | 54 | T |
| | | | 01 0101 | 21 | 15 | NAK | 85 | 55 | U |
| | | | 01 0110 | 22 | 16 | SYN | 86 | 56 | V |
| | | | 01 0111 | 23 | 17 | ETB | 87 | 57 | W |
| | | | 01 1000 | 24 | 18 | CAN | 88 | 58 | X |
| | | | 01 1001 | 25 | 19 | EM | 89 | 59 | Y |
| | | | 01 1010 | 26 | 1a | SUB | 90 | 5a | Z |
| | divu | | 01 1011 | 27 | 1b | ESC | 91 | 5b | [ |
| | | | 01 1100 | 28 | 1c | FS | 92 | 5c | \ |
| | | | 01 1101 | 29 | 1d | GS | 93 | 5d | ] |
| | | | 01 1110 | 30 | 1e | RS | 94 | 5e | ^ |
| | | | 01 1111 | 31 | 1f | US | 95 | 5f | _ |
| lb | add | cvt.s.f | 10 0000 | 32 | 20 | Space | 96 | 60 | ` |
| lh | addu | cvt.d.f | 10 0001 | 33 | 21 | ! | 97 | 61 | a |
| lwl | sub | | 10 0010 | 34 | 22 | " | 98 | 62 | b |
| lw | subu | | 10 0011 | 35 | 23 | # | 99 | 63 | c |
| lbu | and | cvt.w.f | 10 0100 | 36 | 24 | $ | 100 | 64 | d |
| lhu | or | | 10 0101 | 37 | 25 | % | 101 | 65 | e |
| lwr | xor | | 10 0110 | 38 | 26 | & | 102 | 66 | f |
| | nor | | 10 0111 | 39 | 27 | ' | 103 | 67 | g |
| sb | | | 10 1000 | 40 | 28 | ( | 104 | 68 | h |
| sh | | | 10 1001 | 41 | 29 | ) | 105 | 69 | i |
| swl | slt | | 10 1010 | 42 | 2a | * | 106 | 6a | j |
| sw | sltu | | 10 1011 | 43 | 2b | + | 107 | 6b | k |
| | | | 10 1100 | 44 | 2c | , | 108 | 6c | l |
| | | | 10 1101 | 45 | 2d | - | 109 | 6d | m |
| swr | | | 10 1110 | 46 | 2e | . | 110 | 6e | n |
| cache | | | 10 1111 | 47 | 2f | / | 111 | 6f | o |
| ll | tge | c.f.f | 11 0000 | 48 | 30 | 0 | 112 | 70 | p |
| lwc1 | tgeu | c.un.f | 11 0001 | 49 | 31 | 1 | 113 | 71 | q |
| lwc2 | tlt | c.eq.f | 11 0010 | 50 | 32 | 2 | 114 | 72 | r |
| pref | tltu | c.ueq.f | 11 0011 | 51 | 33 | 3 | 115 | 73 | s |
| | teq | c.olt.f | 11 0100 | 52 | 34 | 4 | 116 | 74 | t |
| ldc1 | | c.ult.f | 11 0101 | 53 | 35 | 5 | 117 | 75 | u |
| ldc2 | tne | c.ole.f | 11 0110 | 54 | 36 | 6 | 118 | 76 | v |
| | | c.ule.f | 11 0111 | 55 | 37 | 7 | 119 | 77 | w |
| sc | | c.sf.f | 11 1000 | 56 | 38 | 8 | 120 | 78 | x |
| swc1 | | c.ngle.f | 11 1001 | 57 | 39 | 9 | 121 | 79 | y |
| swc2 | | c.seq.f | 11 1010 | 58 | 3a | : | 122 | 7a | z |
| | | c.ngl.f | 11 1011 | 59 | 3b | ; | 123 | 7b | { |
| | | c.lt.f | 11 1100 | 60 | 3c | < | 124 | 7c | \| |
| sdc1 | | c.nge.f | 11 1101 | 61 | 3d | = | 125 | 7d | } |
| sdc2 | | c.le.f | 11 1110 | 62 | 3e | > | 126 | 7e | ~ |
| | | c.ngt.f | 11 1111 | 63 | 3f | ? | 127 | 7f | DEL |

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

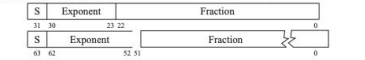


### IEEE 754 FLOATING-POINT STANDARD ④

$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

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

**IEEE Single Precision and Double Precision Formats:**

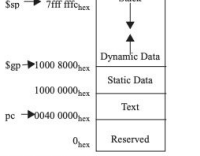



**IEEE 754 Symbols**

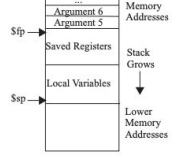
| Exponent | Fraction | Object |
|---|---|---|
| 0 | 0 | ± 0 |
| 0 | ≠0 | ± Denorm |
| 1 to MAX - 1 | anything | ± Fl. Pt. Num. |
| MAX | 0 | ±∞ |
| MAX | ≠0 | NaN |

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

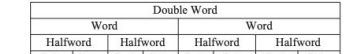
### MEMORY ALLOCATION




### STACK FRAME

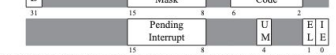



### DATA ALIGNMENT




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

### EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS




BD = Branch Delay, UM = User Mode, EL = Exception Level, IE =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 |
| 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 ($10^x$ for Disk, Communication; $2^x$ 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 |

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) | $0 / 20_{hex}$ |
| 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) | $c_{hex}$ |
| 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) | $d_{hex}$ |
| Subtract | `sub` | R | R[rd] = R[rs] - R[rt] | (1) | $0 / 22_{hex}$ |
| Subtract Unsigned | `subu` | R | R[rd] = R[rs] - R[rt] | | $0 / 23_{hex}$ |

**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] + 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) | $c_{hex}$ |
| 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) | $d_{hex}$ |
| Subtract | `sub` | R | R[rd] = R[rs] - R[rt] | (1) | 0 / $22_{hex}$ |
| Subtract Unsigned | `subu` | R | R[rd] = R[rs] - R[rt] | | 0 / $23_{hex}$ |

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

**In this column are footnotes**

- **(#) are some instruction specific notes**

**Example Footnote:**

`add and sub instructions may cause an overflow exception`

# Arithmetic and Logic functions

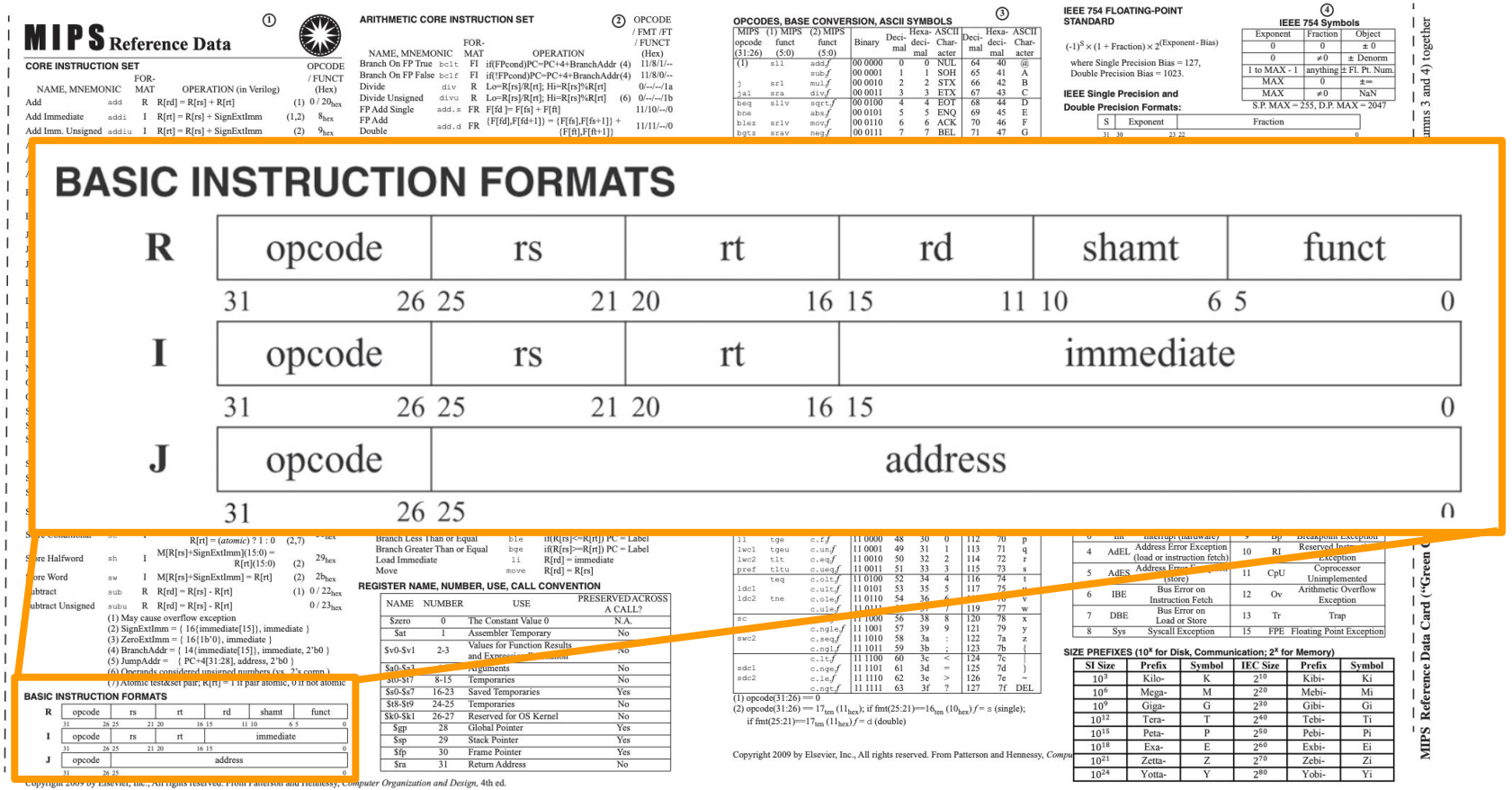
**CORE INSTRUCTION SET**

| NAME, MNEMONIC | | FOR-MAT | OPERATION (in Verilog) | | OPCODE / FUNCT (Hex) |
|---|---|---|---|---|---|
| Add | `add` | R | R[rd] = R[rs] + R[rt] | (1) | $0 / 20_{hex}$ |
| 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) | $c_{hex}$ |
| 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) | $d_{hex}$ |
| Subtract | `sub` | R | R[rd] = R[rs] - R[rt] | (1) | $0 / 22_{hex}$ |
| Subtract Unsigned | `subu` | R | R[rd] = R[rs] - R[rt] | | $0 / 23_{hex}$ |

**This column explain how to create the machine code**

- **Format is R, I or J type**

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

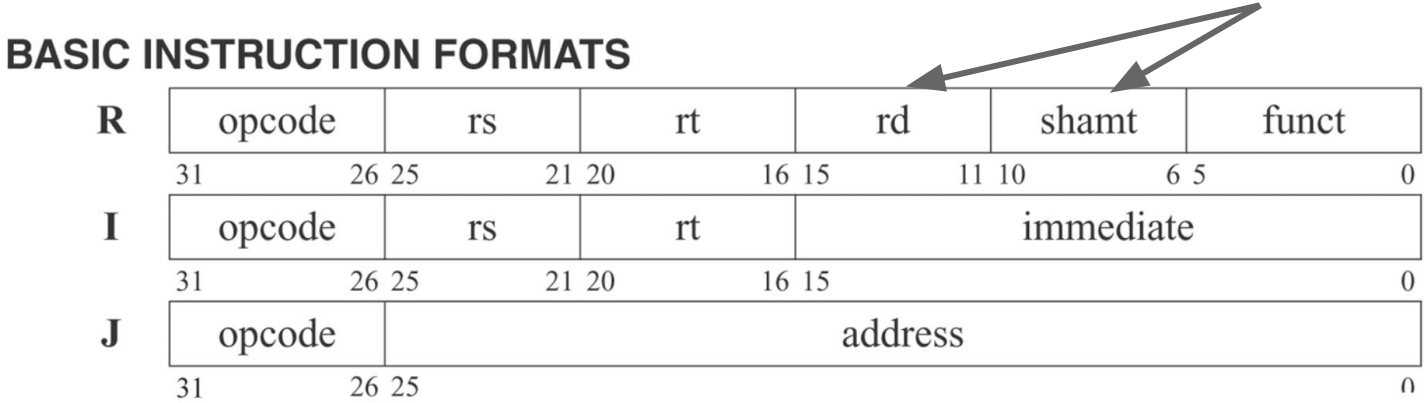


## BASIC INSTRUCTION FORMATS

| | | | | | | |
|---|---|---|---|---|---|---|
| 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 | | | | |

# 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



# Arithmetic and Logic functions

**CORE INSTRUCTION SET**

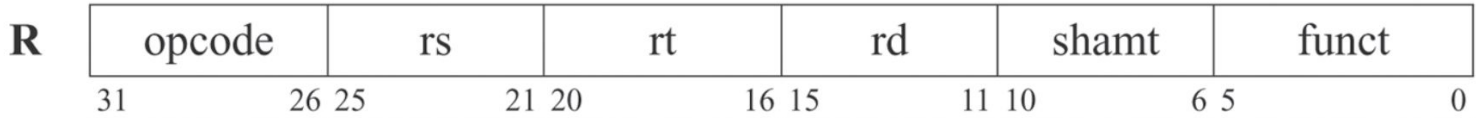
| NAME, MNEMONIC | | FOR-MAT | OPERATION (in Verilog) | | OPCODE / FUNCT (Hex) |
|---|---|---|---|---|---|
| Add | `add` | R | R[rd] = R[rs] + R[rt] | (1) | 0 / $20_{hex}$ |
| Add Immediate | `addi` | I | R[rt] = R[rs] + SignExtImm | (1,2) | $8_{hex}$ |
| Add Imm. Unsigned | `addiu` | I | R[rt] = R[rs] + 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) | $c_{hex}$ |
| 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) | $d_{hex}$ |
| 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) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }

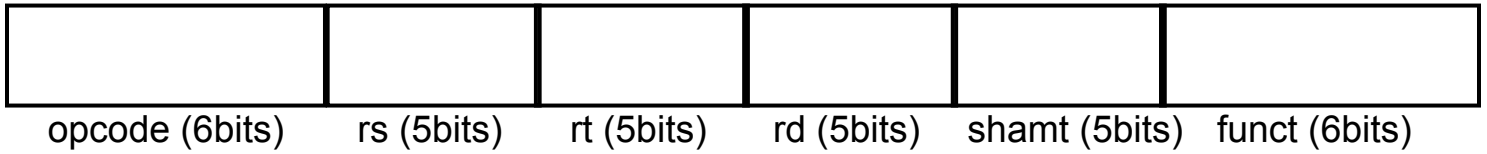
# R-type Instruction

**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

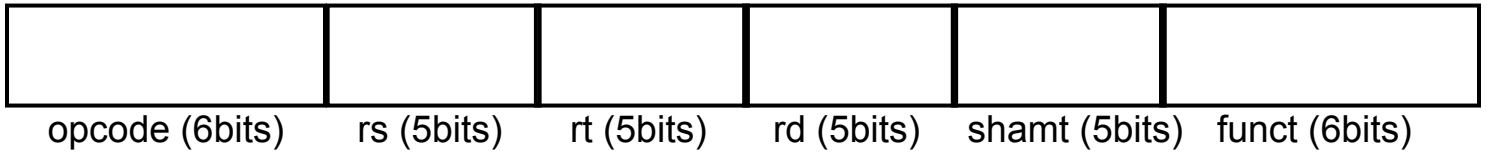



```
add $t0, $s1, $s2
```

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



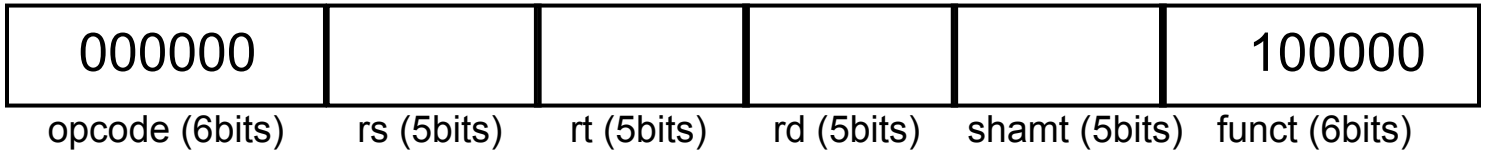

```
add $t0, $s1, $s2
```

1. What is the opcode/funct for add?

**CORE INSTRUCTION SET**

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

# R-type Instruction Example - My Turn




```
add $t0, $s1, $s2
```

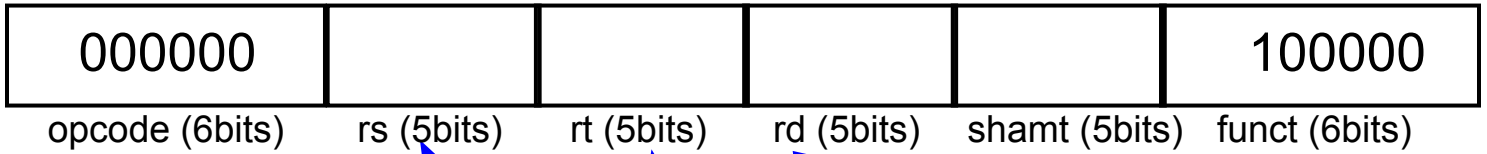
2. What are the operands for add? What is the order?

**The operand order: `add rd, rs, rt`**

**CORE INSTRUCTION SET**

| NAME, MNEMONIC | | FOR-MAT | OPERATION (in Verilog) | | OPCODE / FUNCT (Hex) |
|---|---|---|---|---|---|
| Add | `add` | R | R[rd] = R[rs] + R[rt] | (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

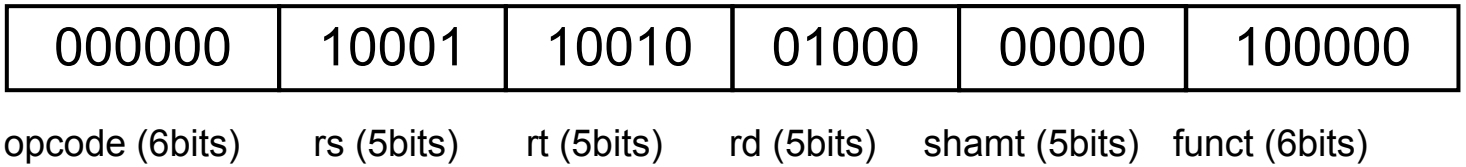
$t0 = _____ ( 0b_____)

$s1 = _____ ( 0b_____)

$s2 = _____ ( 0b_____)

# R-type Instruction Example - My Turn

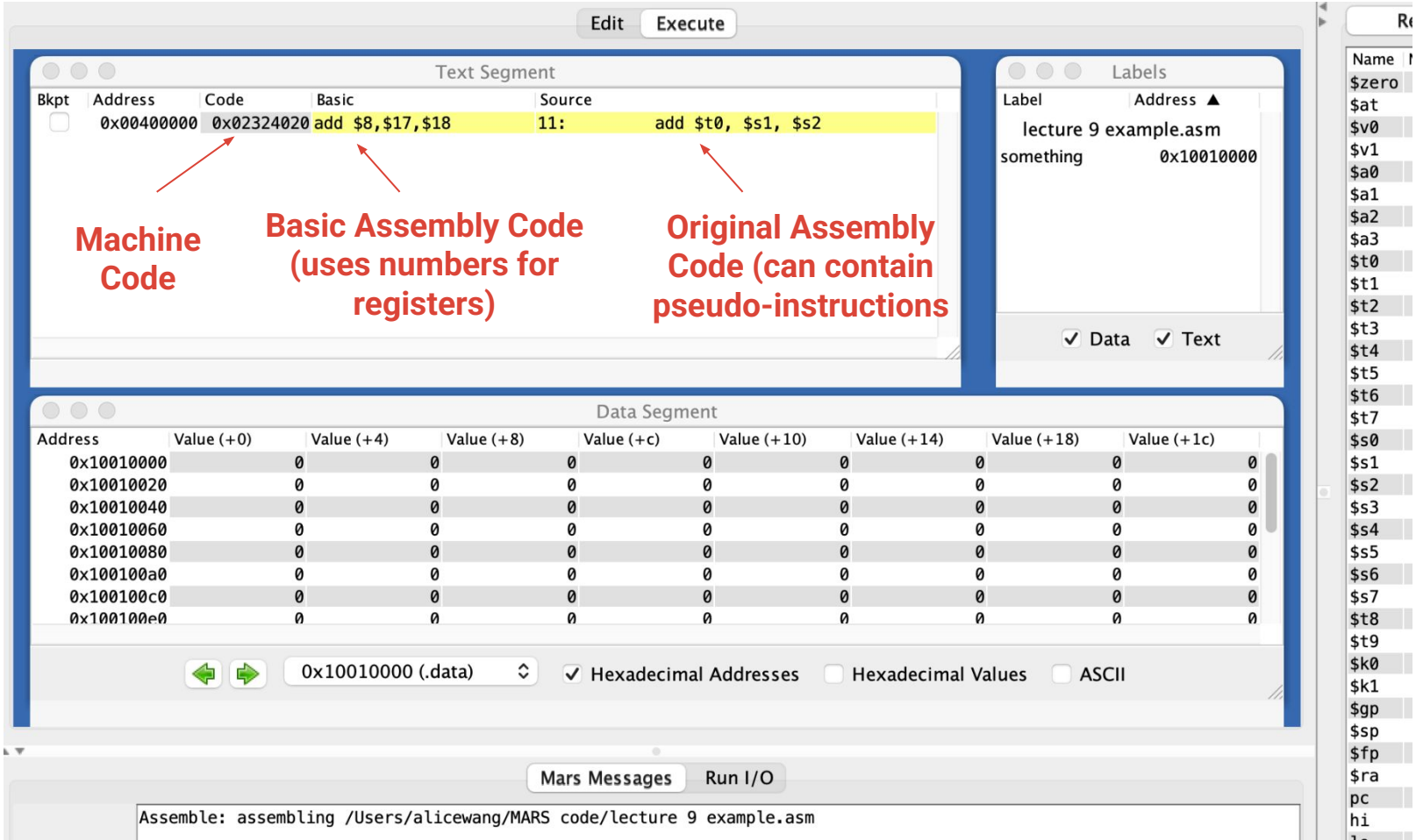
**Instruction in Binary**



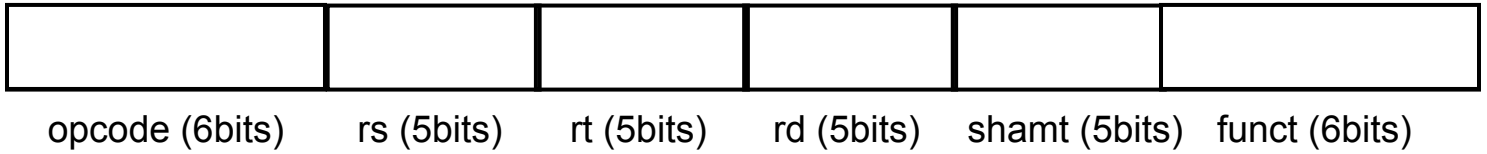

**Convert to Hex**

`add $t0, $s1, $s2` → _____$_{16}$ in machine language

# MARS assembler



# R-type Instruction Example - Your Turn

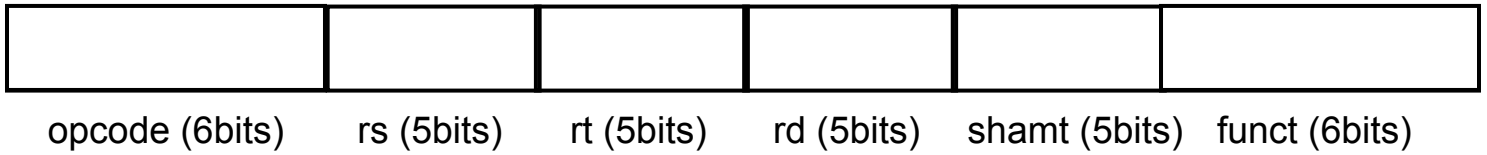



```
srl $s0, $t1, 2
```

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



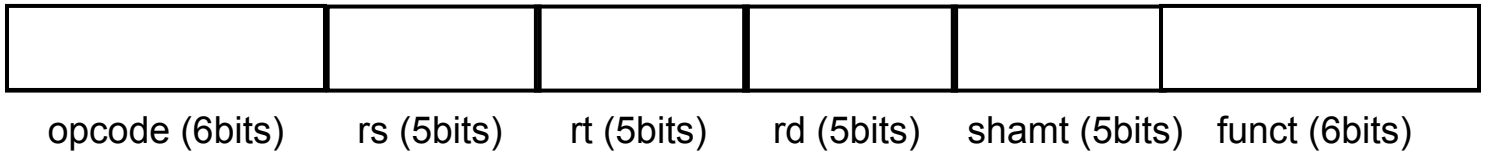

```
srl $s0, $t1, 2
```

1. What is the opcode/funct for srl?

**CORE INSTRUCTION SET**

| NAME, MNEMONIC | | FORMAT | 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



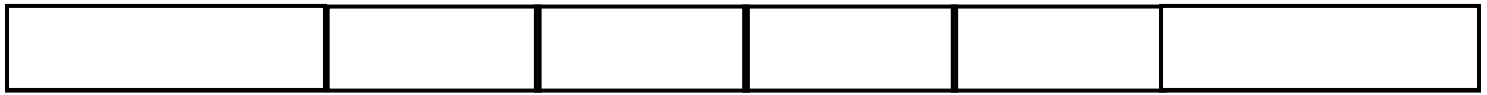

```
srl $s0, $t1, 2
```

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 |

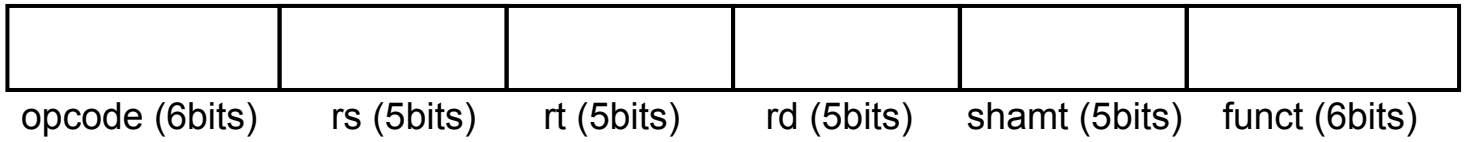
Use the table for register mapping

$s0 = _____ ( 0b_____)

$t1 = _____ ( 0b_____)

# R-type Instruction Example - Your Turn

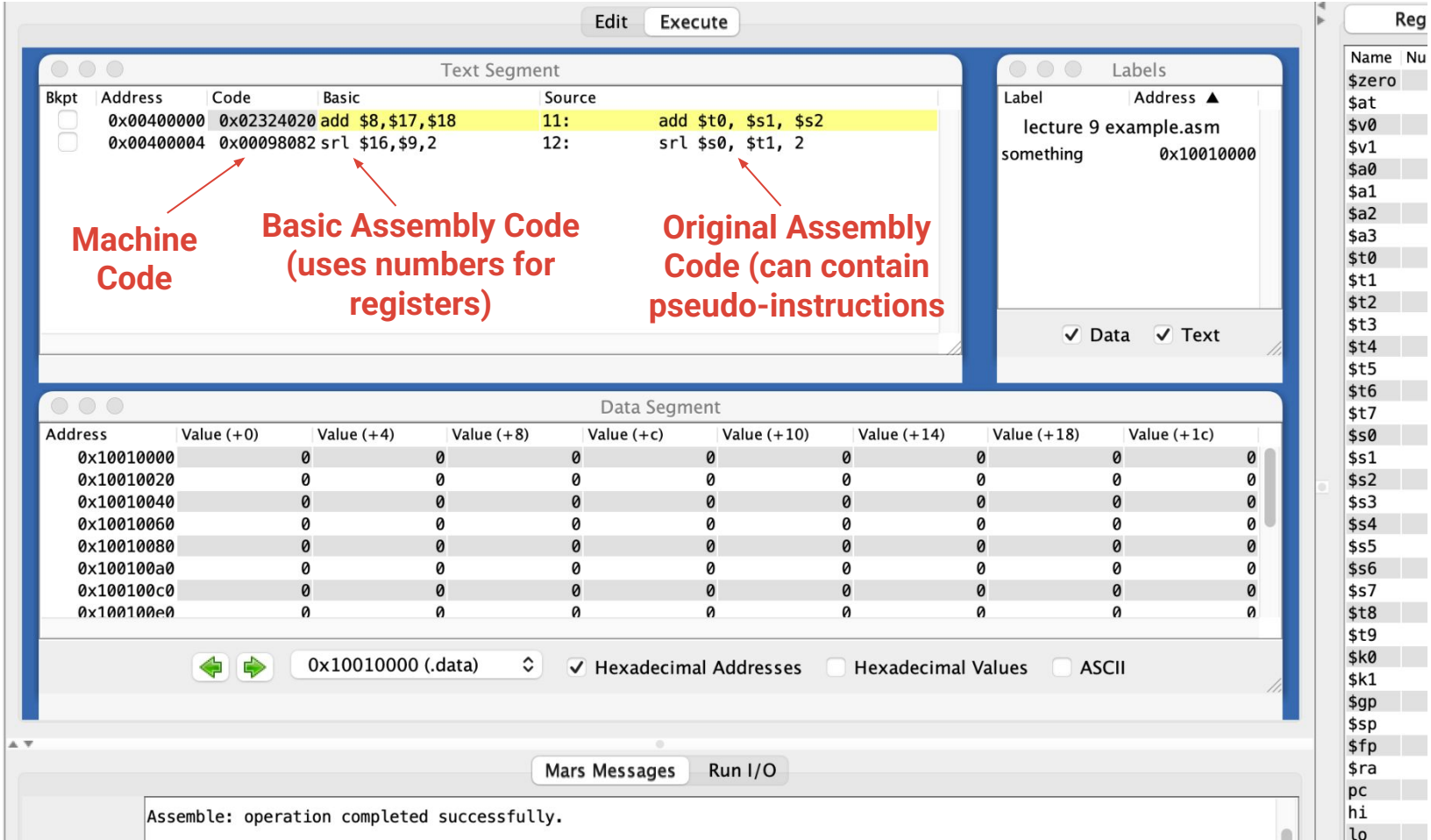
**Instruction in Binary**




**Convert to Hex**

`srl $s0, $t1, 2` → _____$_{16}$ in machine language

# MARS assembler



# 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)