CMPE 200
Computer Architecture & Design

Lecture 2. Processor Instruction Set Architecture & Language (3)

Haonan Wang

SJSU SAN JOSÉ STATE UNIVERSITY

Machine Code of R-Type Instructions

- **All instructions in MIPS have 32-bit width**
  - Within 32-bit data, command and three register ids should be presented

add Rd, Rs, Rt

6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
Opcode	Rs	Rt	Rd	Shamt	Function

32 bits

Machine Code of R-Type Instructions

6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
Opcode	Rs	Rt	Rd	Shamt	Function

32 bits

- **Opcode**
  - Indicate Command/Operation of an instruction with combination of Function field
  - Example:
    - **add**
    - **and**
    - **or**
    - **many more**

add	0	Rs	Rt	Rd	Shamt	0x20
and	0	Rs	Rt	Rd	Shamt	0x24
or	0	Rs	Rt	Rd	Shamt	0x25

  - R-type instructions always have all zeros in Opcode field and use Function field to distinguish the instructions

Machine Code of R-Type Instructions

6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
Opcode	Rs	Rt	Rd	Shamt	Function

32 bits

- **Rs, Rt, Rd**
  - Indicate source and destination register ids used by an instruction
  - Example:
    - **add $5, $4, $3**
    - **and $13, $8, $2**
    - **or $t8, $s0, $s1**

000000	$4	$3	$5	Shamt	100000
000000	$8	$2	$13	Shamt	100100
000000	$s0 (16)	$s1 (17)	$t8 (24)	Shamt	100101

  - Each field has 5 bits because we have 32 (=2^5) registers in MIPS

Machine Code of R-Type Instructions

6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
Opcode	Rs	Rt	Rd	Shamt	Function

32 bits

- **Shamt**
  - Indicate the shift amount that is used for shift instructions only; the other R-type instructions have all zeros in this field
  - Example:

• **add**	**$5, $4, $3**	000000	00100	00011	00101	00000	100000
• **and**	**$13, $8, $2**	000000	01000	00010	01101	00000	100100
• **or**	**$t8, $s0, $s1**	000000	10000	10001	11000	00000	100101

Shift Instructions

- **Special format R-type Instruction**
  - The second operand is an immediate value (shamt) that indicates the amount of shift

- **Logical Shifting**
  - Shift towards left/right with filling in 0s
  - Example:
    - **sll** Rd, Rt, shamt # Rd = Rt << shamt (Rt: unsigned data)
    - **srl** Rd, Rt, shamt # Rd = Rt >> shamt (Rt: unsigned data)

- **Arithmetic Shifting**
  - Shift towards left/right with maintaining the value's sign bit
  - Example:
    - **sra** Rd, Rt, shamt # Rd = Rt >> shamt (Rt: signed data)

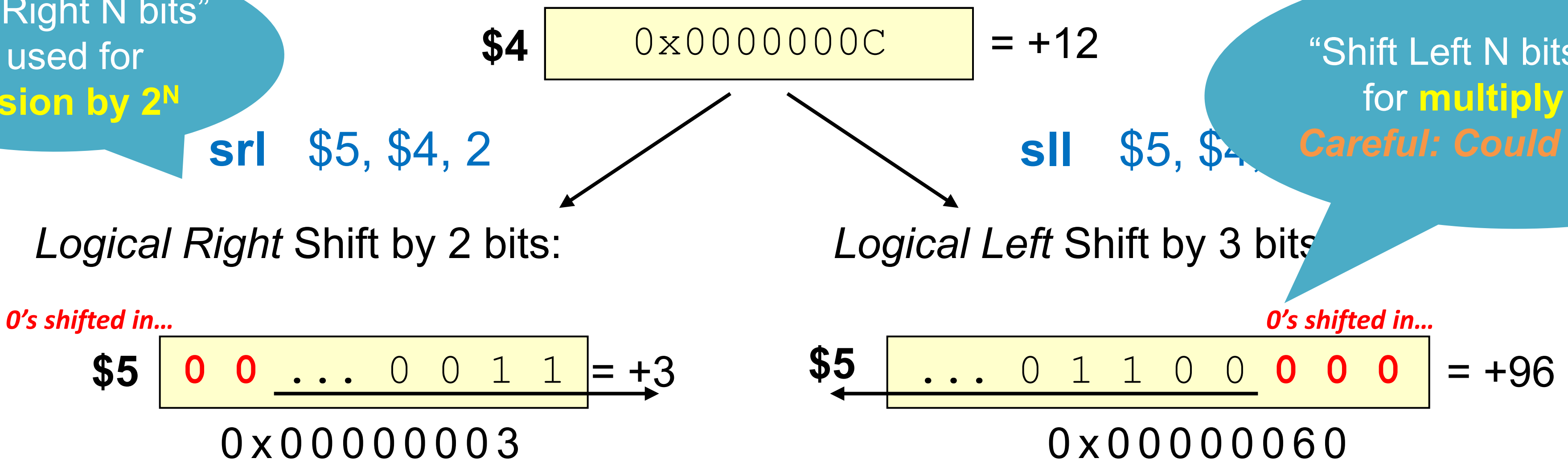
Example: Logical Shift

- **Shift towards left/right with filling in 0s**
- **Example:**
  - Assume that the original value of $4 is 0x0000000C
  - What is the value of $5 after executing the followi
  - **sll** $5, $4, 3 # shift left logical the value c
  - **srl** $5, $4, 2 # shift right logical the value of $ 2 bits

Shift operation should be done in bit level → **translate given value to binary first**

“Shift Right N bits” is used for **division by 2^N**

“Shift Left N bits” is used for **multiply by 2^N**
Careful: Could overflow!

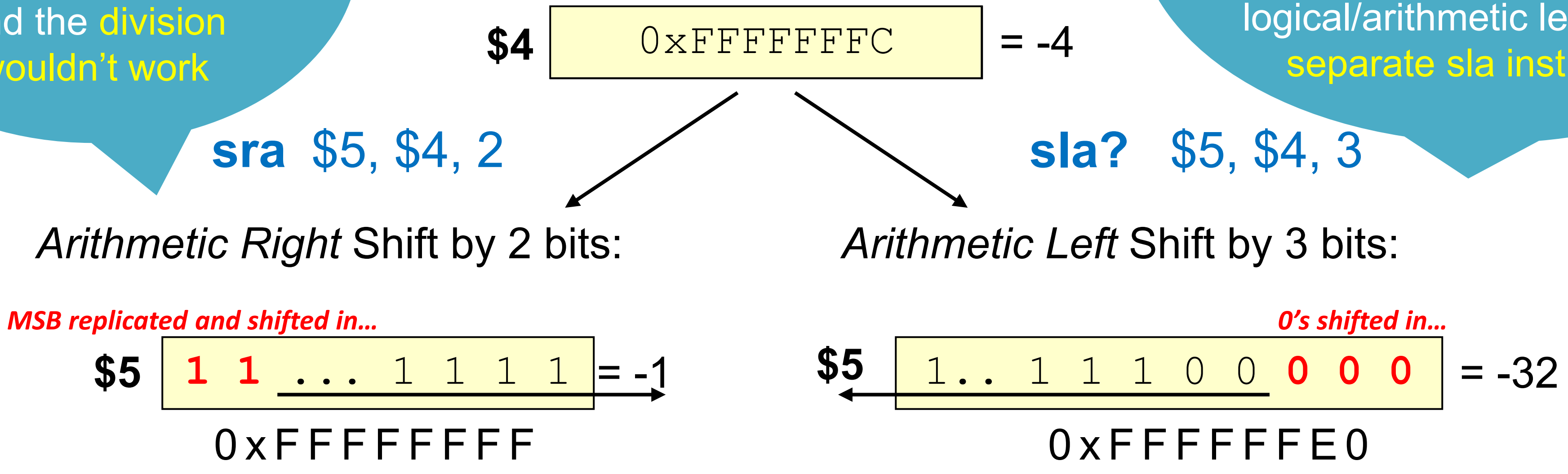


Example: Arithmetic Shift

- **Shift right with replicating MSB**
- **Shift left with filling in 0s**
- **Example:**
  - – the original value of $4 is 0xFFFFFFFC (= -4)
  - of $5 after executing the following shift instru
  - # shift right arithmetic the value of $4 b

Notice if we shifted in 0s (like a logical right shift) our result would be a positive number and the division wouldn't work

Notice there is no difference between an arithmetic and logical left shift. We always shift in 0s. MIPS uses sll for both logical/arithmetic left shift (no separate sla instruction)



SJSU SAN JOSÉ STATE UNIVERSITY

Machine Code of R-Type Instructions

6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
Opcode	Rs	Rt	Rd	Shamt	Function

32 bits

- **Shamt**
  - Indicate the shift amount that is used for shift instructions only; the other R-type instructions have all zeros in this field
  - Example:

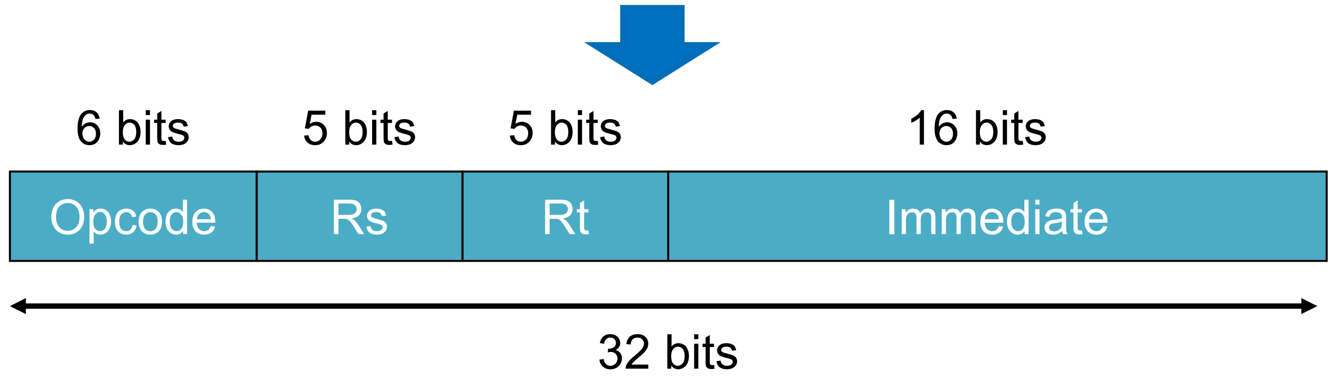
• **sll**	**$5, $4, 3**	000000	00000	00100	00101	3	000000
• **srl**	**$5, $4, 2**	000000	00000	00100	00101	2	000010
• **sra**	**$5, $4, 2**	000000	00000	00100	00101	2	000011

SJSU SAN JOSÉ STATE UNIVERSITY

Machine Code of I-Type Instructions

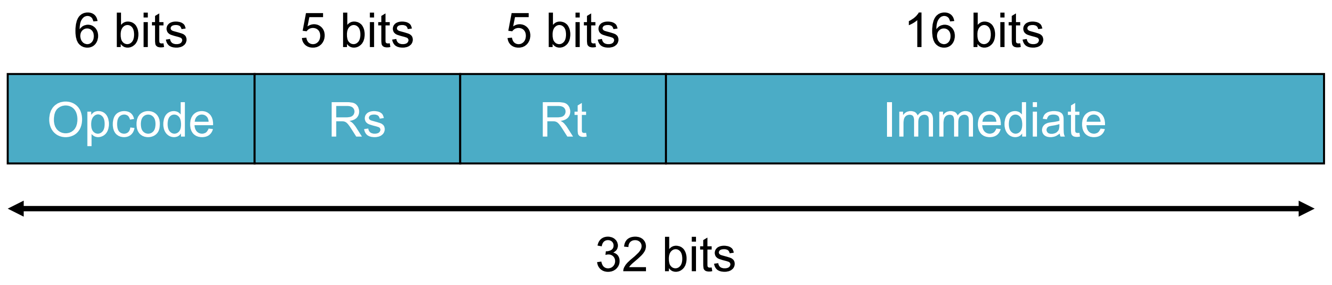
- **All instructions in MIPS are 32-bit wide**
  - Within 32-bit data, command, two register ids, and an immediate value should be presented

addi Rt, Rs, imm



SJSU SAN JOSÉ STATE UNIVERSITY

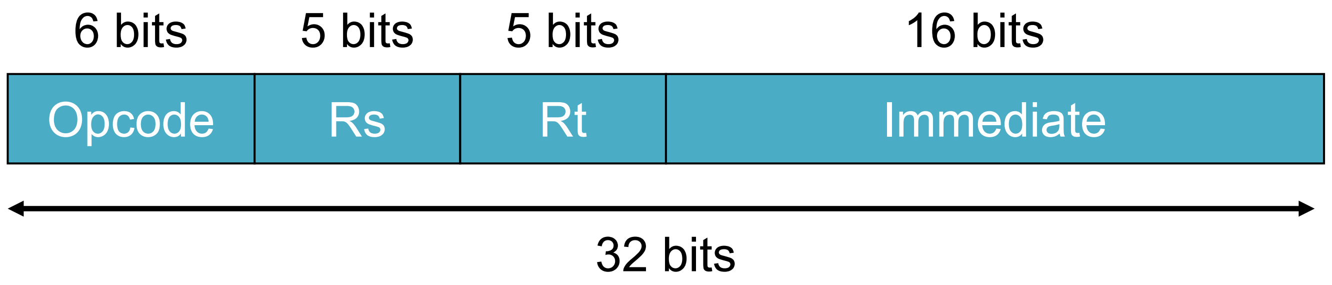
Machine Code of I-Type Instructions



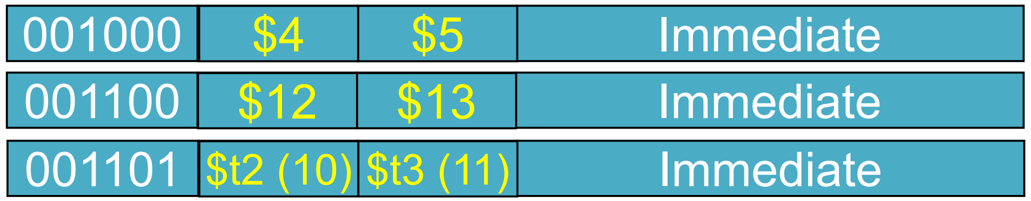
- **Opcode**
  - Indicate Command/Operation of an instruction
  - Example:

• **addi**	001000	Rs	Rt	Immediate
• **andi**	001100	Rs	Rt	Immediate
• **ori**	001101	Rs	Rt	Immediate
• **many more;**				

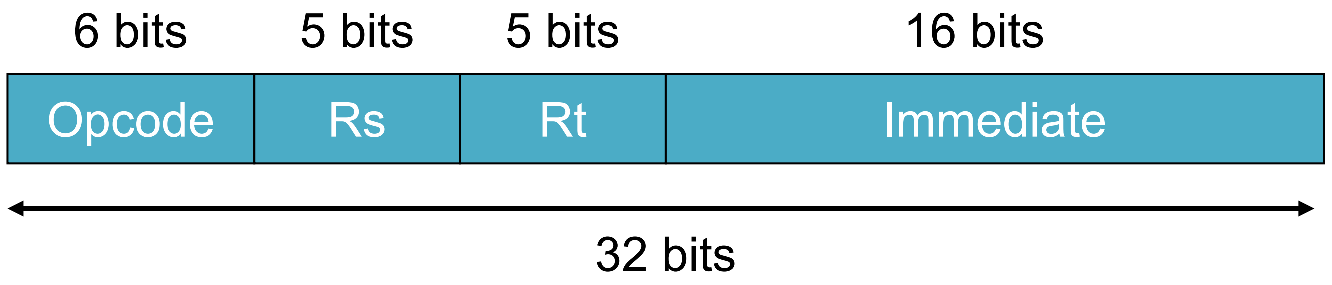
Machine Code of I-Type Instructions



- **Rs, Rt**
  - Indicate source and destination register ids used by an instruction
  - Example:
    - **addi $5, $4, 1**
    - **andi $13, $12, 0xA**
    - **ori $t3, $t2, 12**



Machine Code of I-Type Instructions



- **Immediate**
  - Indicate the immediate value
  - Example:

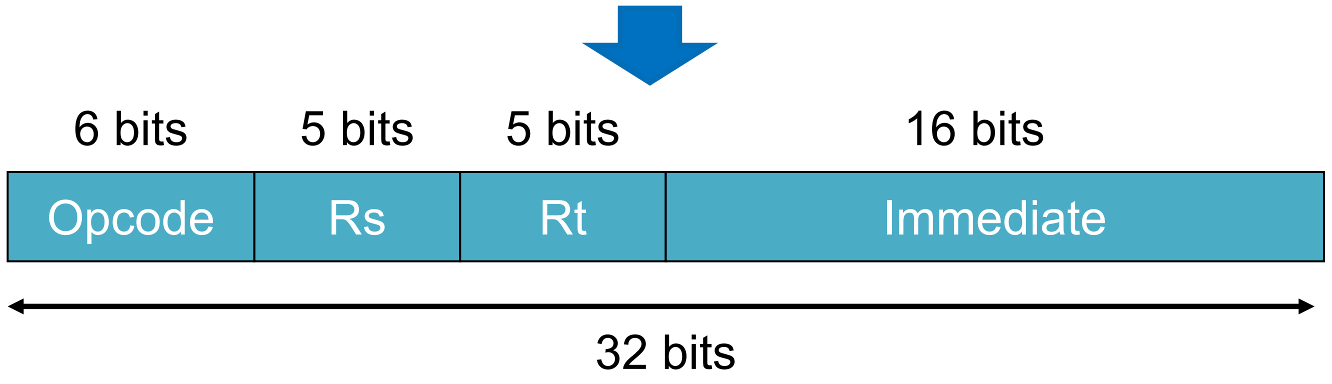
addi	**$5, $4, 1**	001000	00100	00101	1
andi	**$13, $12, 0xA**	001100	01100	01101	0xA
ori	**$t3, $t2, 12**	001101	01010	01011	12

SJSU SAN JOSÉ STATE UNIVERSITY

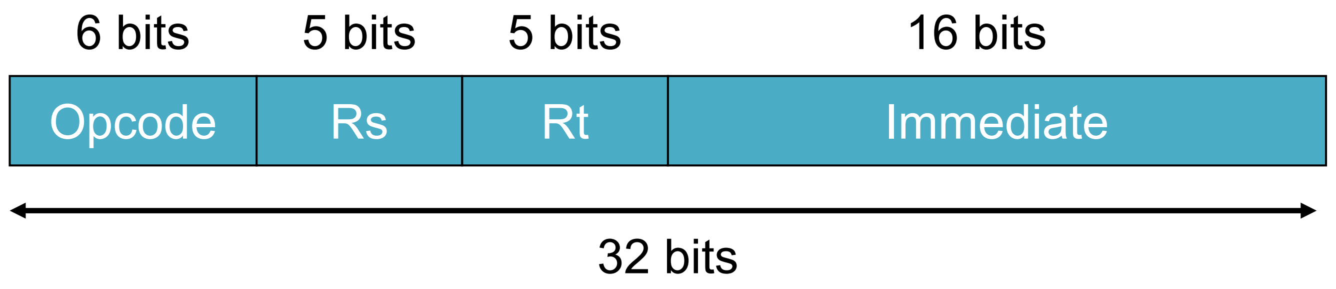
Machine Code of I-Type Instructions

- **Special I-Type instructions: Load/Store**
  - Within 32-bit data, command, two register ids, and an offset value should be presented

lw Rt, imm(Rs)



Machine Code of I-Type Instructions



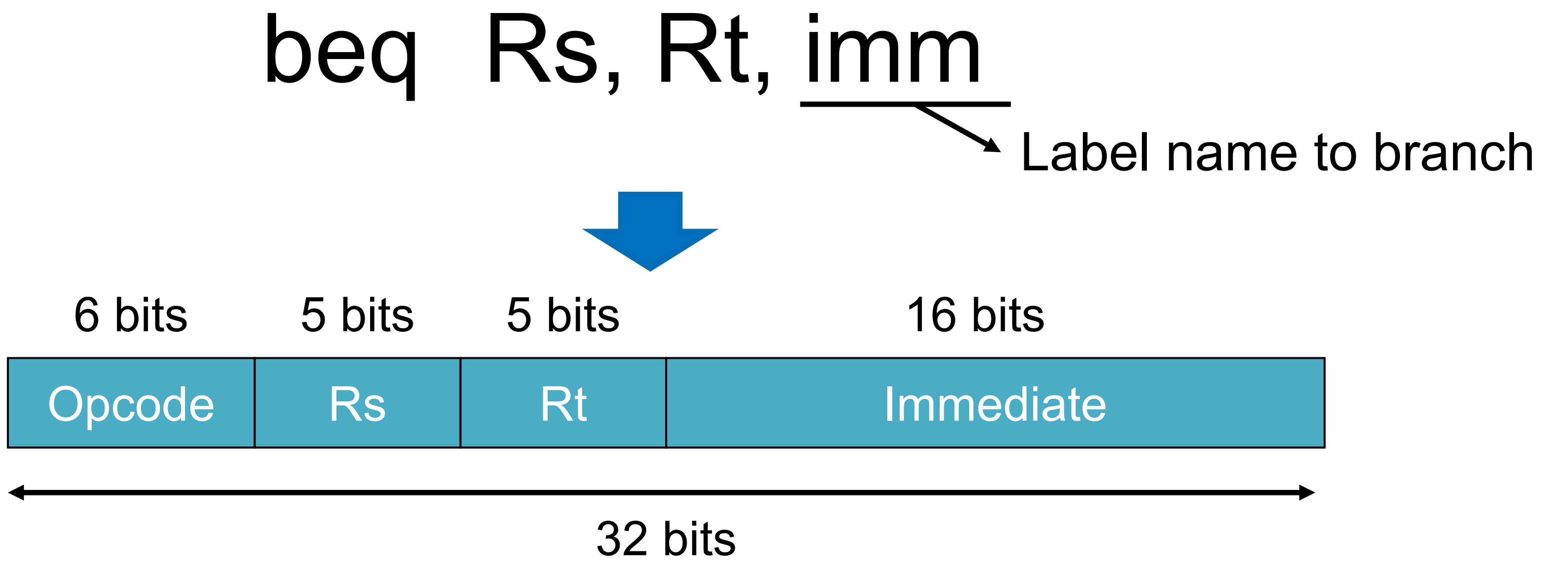
- **Load/Store in I-Type machine code format**
  - Example:

lb	**$5, 4($4)**	0x20	00100	00101	4
lh	**$13, 0x10($12)**	0x21	01100	01101	0x10
lw	**$t3, -16($t2)**	0x23	01010	01011	-16
sb	**$3, 0xfff0($4)**	0x28	00100	00011	0xfff0
sh	**$2, 8($8)**	0x29	01000	00010	8
sw	**$s1, 10($s0)**	0x2B	10000	10001	10

SJSU SAN JOSÉ STATE UNIVERSITY

Machine Code of I-Type Instructions

- **Special I-Type instructions: Branch**
  - uses the same format but needs a special treatment for immediate field value



Branch Target Addressing

- **PC-relative addressing**
  - MIPS calculates the branch target address based on the branch instruction's next instruction's address
  - *"Branch to N lines abo* branch to absolute a
  - necessary because in t do not know (at compile time) where our code will be loaded in the m

Assume that the code is loaded to **0x00080000** in memory

```
80000     Loop: sll  $t1, $s3, 2
80004           add  $t1, $t1, $s6
80008           lw   $t0, 0($t1)
8000C           bne  $t0, $s5, Exit
80010           addi $s3, $s3, 1
80014           b    Loop
80018     Exit: …
```

Branch two lines below from bne's next instruction (addi)
→ Immediate field will be filled with 2

Branch 6 lines above from b's next line (Exit)
→ Immediate field will be filled with -6

ng
ddress increases by 4 bytes

Branch Target Addressing

- **Why branch's next instruction should be considered?**
  - Because MIPS increments program counter (PC) by 4 before executing an instruction
  - This already incremented PC value is used for branch target address calculation

- **Branch Target Address Calculation**
  - **Target address = branch's next instruction address + offset x 4**
  - **bne $t0, $s5, Exit**
    - Exit (0x00080018) = addi address (0x00080010) + distance (2) x 4 byte/inst
  - **b Loop**
    - Loop (0x00080000) = Exit address (0x00080018) + distance (-6) x 4 byte/inst

```
80000    Loop: sll  $t1, $s3, 2
80004          add  $t1, $t1, $s6
80008          lw   $t0, 0($t1)
8000C          bne  $t0, $s5, Exit
80010          addi $s3, $s3, 1
80014          b    Loop
80018    Exit: …
```

Machine Code of I-Type Instructions

6 bits	5 bits	5 bits	16 bits
Opcode	Rs	Rt	Immediate

32 bits

```
80000s    Loop: sll  $t1, $s3, 2
80004           add  $t1, $t1, $s6
80008           lw   $t0, 0($t1)
8000C           bne  $t0, $s5, Exit
80010           addi $s3, $s3, 1
80014           b    Loop
80018     Exit: …
```

- **Branch in I-Type machine code format**
  - Example:
    - **bne $t0, $s5, Exit**
    - **beq $0, $0, Loop**

0x5	01000	10101	2
0x4	00000	00000	-6

SJSU SAN JOSÉ STATE UNIVERSITY

Loading an Immediate

- **What if you want to load an immediate value to a register?**
- **If immediate (constant) is 16 bits or less**
  - Use **ori** or **addi** instruction with $0 register
  - Examples : You want to load value 1 to $2
    - addi $2, $0, 1 // R[2] = 0 + 1 = 1
    - ori $2, $0, 0x1 // R[2] = 0 | 1 = 1

LUI: the immediate value is loaded to the MSB 16 bits of the target register

- **If immediate is more than 16 bits**
  - Immediates limited to 16 bits so we must load constant with a 2-instruction sequence using the special LUI (Load Upper Immediate) instruction
  - To load $2 with 0x12345678
    - lui $2, 0x1234
    - ori $2, $2, 0x5678

```
R[2]  12340000   LUI
   OR 00005678
R[2]  12345678   ORI
```

Exercise-2

- **Translate the given high-level language code to MIPS assembly.**
  - Assume that the address of integer variable x and y are in $4 and $1 respectively.

High-level language

```
if (x > y)
        x = x + y;
els
```

MIPS

```
        lw      $2,  0($4)
        lw      $3,  0($1)
        slt     $1,  $3,  $2
        beq     $1,  $0,  Else
        add     $2,  $2,  $3
        b       Next
Else:   addi    $2,  $0,  1
Next:   sw      $2,  0($4)
```

x > y

x <= y

If false, you should not execute add
→ Create a label for
and branch

Wait, when x > y, the calculation result should be also stored to the memory without executing the else code
→ Add unconditional branch and skip addi

5) ster
6) Store the result value to x in memory

SJSU SAN JOSÉ STATE UNIVERSITY

SAN JOSÉ STATE UNIVERSITY *powering* SILICON VALLEY