A few considerations….

- Please do not abuse the CHAT feature on Zoom
- Do not post irrelevant content on Zoom chat. This includes lab discussions **during** lecture time. **You already have Discord for that!**
- Zoom saves a log of chat history along with user name. **So I know who posts irrelevant stuff**
- If you post irrelevant stuff, actual questions posed by students get lost in the sea of chats!

Operate: I-Type Instructions

- Arithmetic Immediate Signed
  - Example (using decimal notation):
    - ADDI $t1, $t2, 23
    - $t1 = $t2 + 23
  - Can use hex notation (useful for masking):
    - ANDI $t1, $t2, 0x000F
    - $t1 = $t2 & 0x000F

Logical Operations

Instruction	Example	Meaning
and	and $1,$2,$3	$1 = $2 & $3
or	or $1,$2,$3	$1 = $2 | $3
xor	xor $1,$2,$3	$1 = $2 ⊕ $3
nor	nor $1,$2,$3	$1 = ~($2 | $3)
and immediate	andi $1,$2,10	$1 = $2 & 10
or immediate	ori $1,$2,10	$1 = $2 | 10
xor immediate	xori $1, $2,10	$1 = ~$2 &~10
shift left logical	sll $1,$2,10	$1 = $2 << 10
shift right logical	srl $1,$2,10	$1 = $2 >> 10
shift right arithm.	sra $1,$2,10	$1 = $2 >> 10
shift left logical	sllv $1,$2,$3	$1 = $2 << $3
shift right logical	srlv $1,$2, $3	$1 = $2 >> $3
shift right arithm.	srav $1,$2, $3	$1 = $2 >> $3

Regarding SUB instructions

- SUB $t1, $t2, $t3
  - Means ($t1) = ($t2) – ($t3)

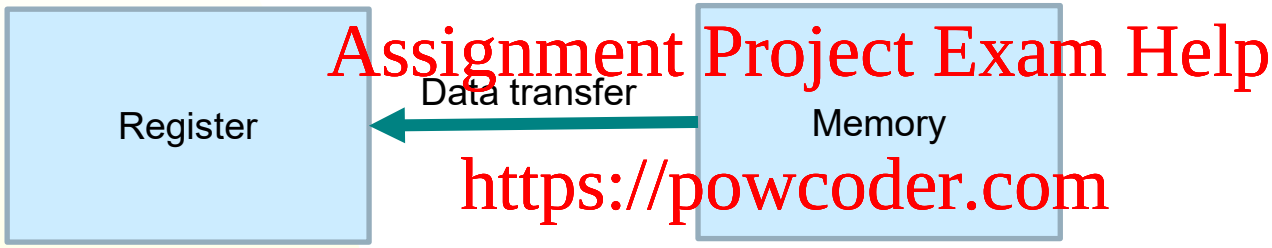
MIPS Overview - Commands

MIPS instructions can be broken down into 3 categories:

- Data Movement
  - Move data between memory and registers
    - For example: lw is load word, sw is store word
- Operate
  - Manipulate data directly
    - For example: add is addition, xor is logical
- Control
  - Change the sequence of instruction execution
    - For example: b is branch, jal is jump and link, ret is return

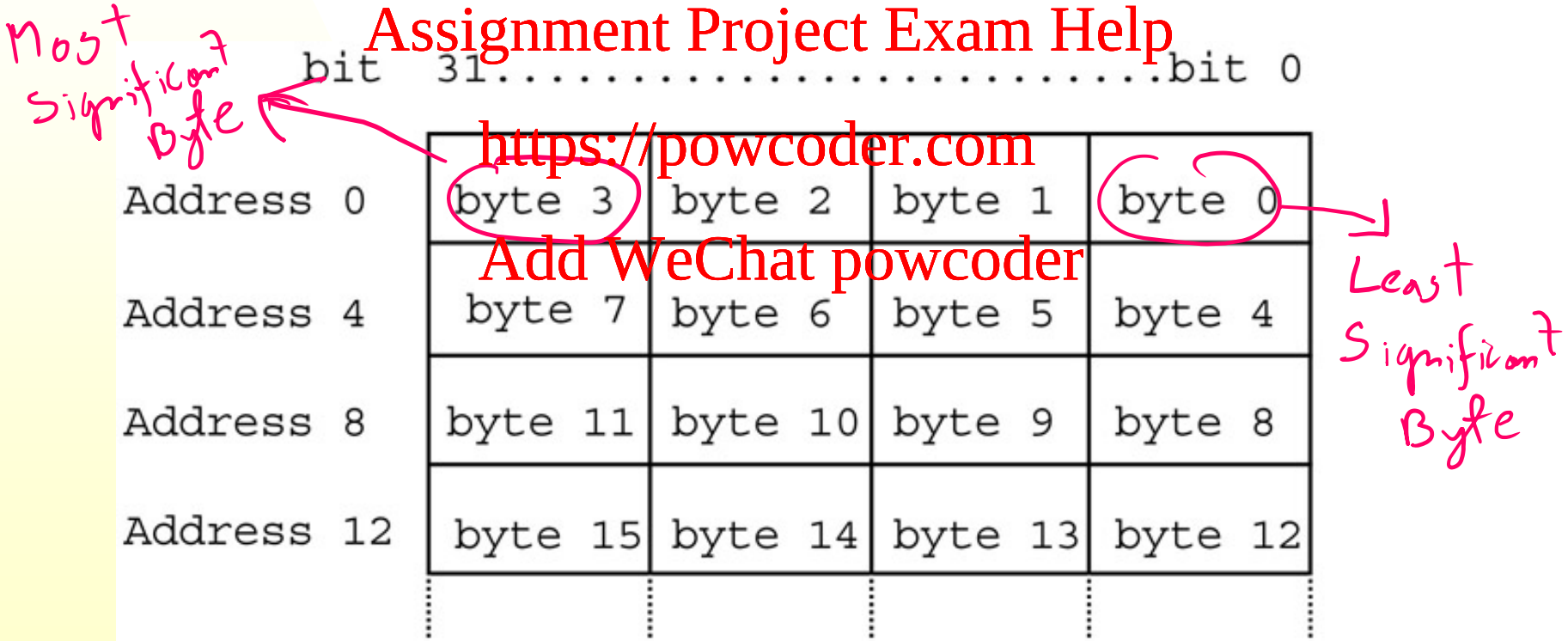
Data-Movement: Load Instructions

- Move data **from** memory to a register

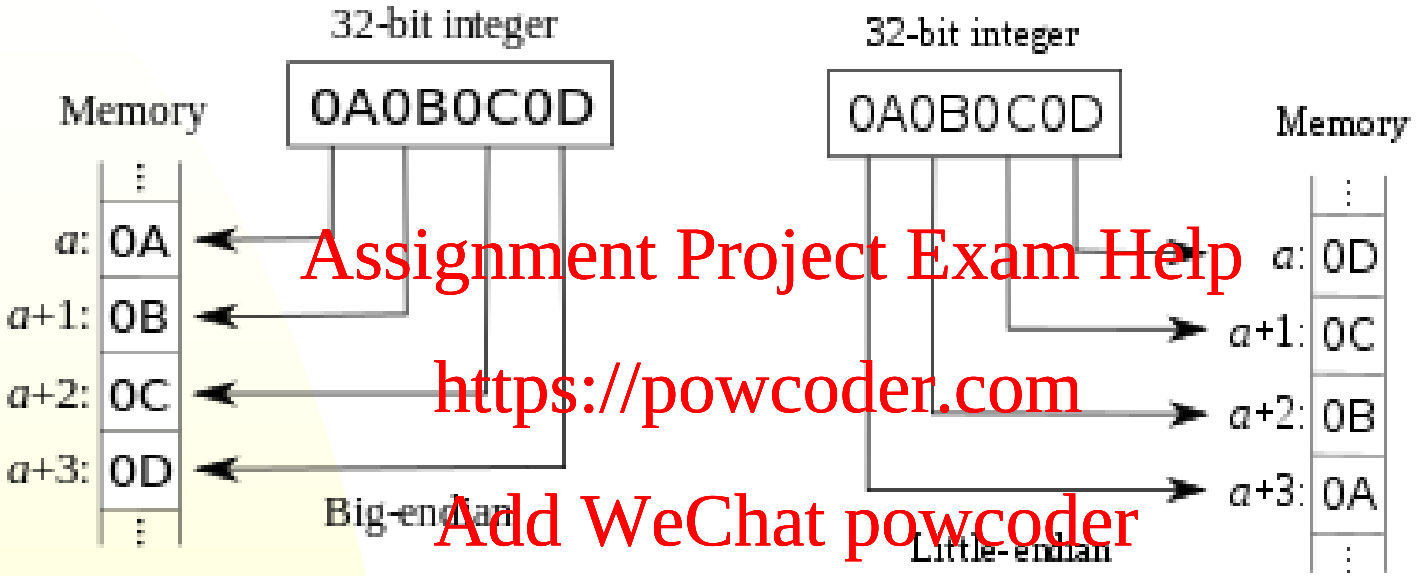


"Endian"-ness of Memory adressing

- MIPS is byte-addressable
- Little Endian byte ordering
  - Compare to Big Endian… (e.g. AVR or ARM)



Endianness



Big Endian: Most significant byte occupying **lower** address position

Little Endian: Least significant byte occupying **lower** address position

Endianness: Memory aid for exams

Least significant byte of data occupies **lower** address position in memory!

Most significant byte of data occupies **lower** address position in memory!

Little Endians

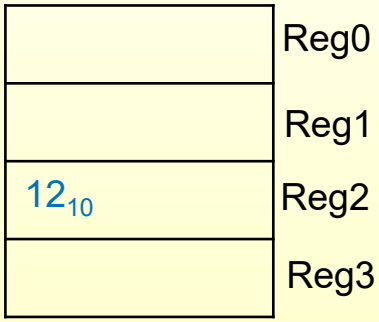
Big Endians

Indirect addressing

This "w" stands for word

- lw $t1, ($t2)
  - $t2 contains a 32-bit address
  - Effective address is $t2
  - ($t1) = memory[$t2]
- NOTE: lw $t1, $t2 is not valid syntax!

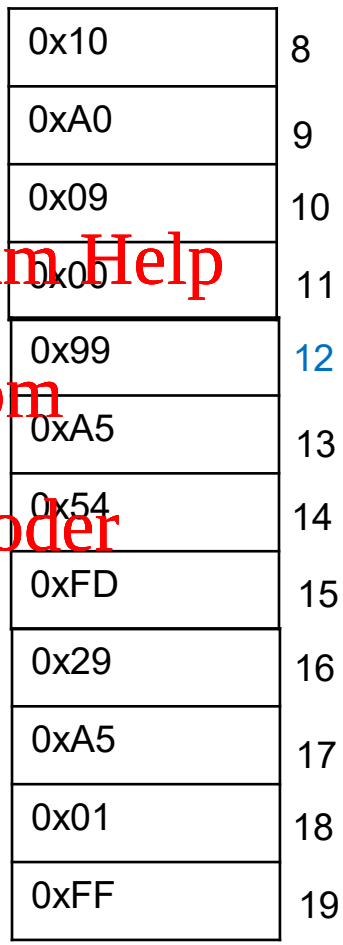
Example:
Assume four 32-bit registers, **Little Endian ordering in memory**



lw Reg1, (Reg2);

(Reg1)=?

(Reg1)= 0x FD 54 A5 99

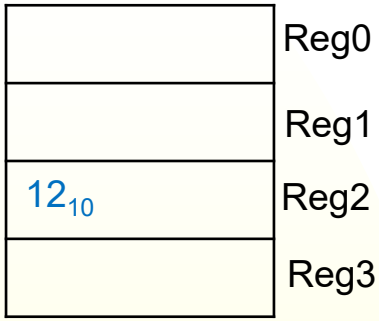


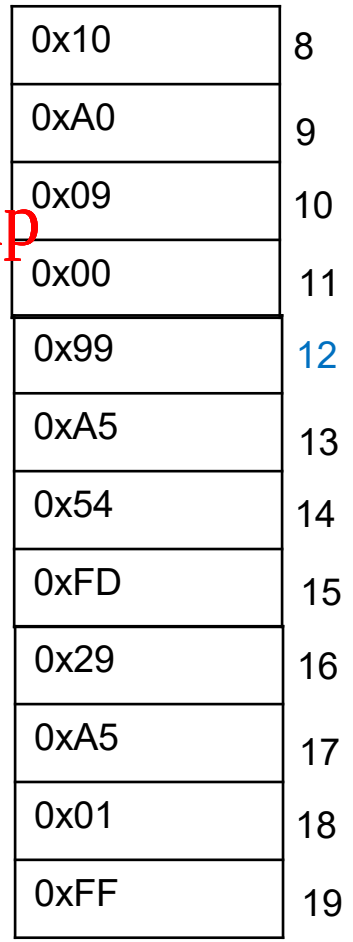
Base + offset addressing

- lw $t1, 4($t2)
  - $t2 contains a 32-bit address
  - Instruction contains 16-bit immediate offset (4)
  - Effective address is $t2+4
  - $t1 = memory[$t2+4]
- NOTE: lw $t1, 0($t2) is the same as indirect addressing!
- Immediate offset is 16-bit *signed* and we can load adjacent words:
  - lw $t1, -4($t2)
  - lw $t2, 0($t2)
  - lw $t3, 4($t2)

Indirect addressing

Example:
Assume four 32-bit registers, **Little Endian ordering in memory**





Problem 1:
lw Reg1, 0(Reg2);

(Reg1)=?
(Reg1)= 0x FD 54 A5 99

Problem 2:
lw Reg1, 4(Reg2);

(Reg1)=?
(Reg1)= 0x FF 01 A5 29

Problem 3:
lw Reg1, -4(Reg2);

(Reg1)=?
(Reg1)= 0x 00 09 A0 10

Load data sizes

- Words are 32-bit values
  - Load word (lw)
  - Same size as the registers
  - Same size as the addresses
- Half-Words
  - Load half word (lh)
  - Are 16-bit values
  - Same size as the immediate values
- Bytes
  - Load byte (lb)
  - Are 8-bit values

Data sizes (cont'd)

- Where are half word and bytes placed?
  - Half word is put in lower register bits [15:0]
  - Byte is put in lower register byte [7:0]
- What about the other bits?
  - Half word
    - LH fills upper bits [31:16] with sign
    - LHU fillers upper bits [31:16] with 0
  - Byte
    - LB fills upper bits [31:8] with sign
    - LBU fillers upper bits [31:8] with 0

Data-Movement: Store Instructions

- Move data **from** Register to Memory

Data-Movement: Stores

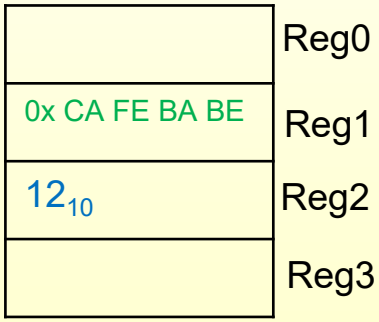
- Stores are similar to loads…
  - Address modes are the same.
  - Data sizes are the same.
    - Upper bits are ignored for byte and half word writes
  - Except… register contents are put in memory.
    - sw $t1, 4($t2)
    - Memory[$t2+4] = ($t1)

Indirect addressing

This "w" stands for word

- **sw** $t1, ($t2)
  - $t2 contains a 32-bit address
  - Effective address is $t2
  - memory[$t2] = ($t1)
- NOTE: sw $t1, $t2 is not valid syntax

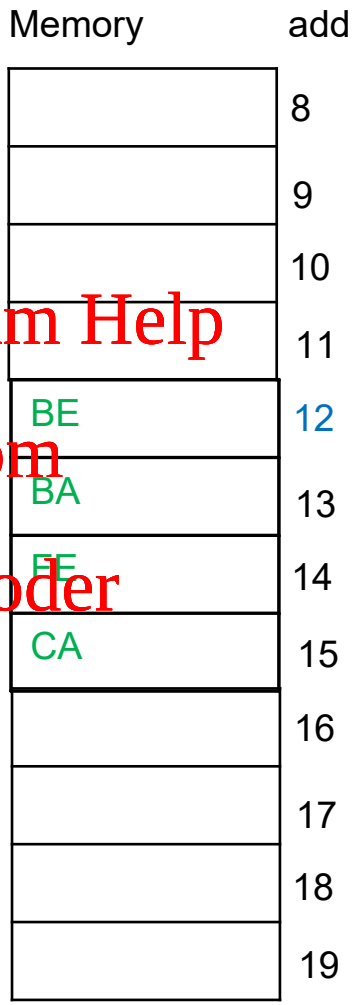
Example:
Assume four 32-bit registers, **Little Endian ordering in memory**



sw Reg1, (Reg2);

Mem[Reg2]=?

Mem[Reg2]= 0x CA FE BA BE



Summary of Loads/Stores

Mnemonic	Instruction
LB	Load Byte
LBU	Load Byte Unsigned
LH	Load Halfword
LHU	Load Halfword Unsigned
LL	Load Linked Word
LW	Load Word
LWL	Load Word Left
LWR	Load Word Right
PREF	Prefetch
SB	Store Byte
SC	Store Conditional Word
SD	Store Doubleword
SH	Store Halfword
SW	Store Word
SWL	Store Word Left
SWR	Store Word Right
SYNC	Synchronize Shared Memory

Use the GREEN!

Don't use Grey!

MIPS Overview - Commands

MIPS instructions can be broken down into 3 categories:

- Data Movement
  - Move data between memory and registers
    - For example: lw is load word, sw is store word
- Operate
  - Manipulate data directly
    - For example: add is addition, xor is logical
- Control
  - Change the sequence of instruction execution
    - For example: b is branch, jal is jump and link, ret is return

Program Flow

- Branches change the flow of instructions
  - Default is to execute the next instruction
  - If the condition is met it will execute the instruction at a label

Conditional Branch (1)

- Compares two registers
  - Branch Equal (BEQ)
  - Branch Not Equal (BNE)
- Example:
  - BEQ $t0, $t1, label
    - If $t0 == $t1, execute instruction at label next
    - Otherwise, execute instruction after BEQ
- Often used to “skip over” some instructions

Branch Example

```
beq $0, $0, blah_label
addi $t1, $0, 1
addi $t2, $0, 2
blah_label:
        addi $t3, $0, 3
```

Bkpt	Address	Code	Basic	Source
	0x00400000	0x10000002	beq $0,$0,0x00000002	1: beq $0, $0, blah_label
	0x00400004	0x20090001	addi $9,$0,0x00000001	2: addi $t1, $0, 1
	0x00400008	0x200a0002	addi $10,$0,0x0000...	3: addi $t2, $0, 2
	0x0040000c	0x200b0003	addi $11,$0,0x0000...	5: addi $t3, $0, 3

Only 2 command executed:

- beq
- addi $t3, $0, 3

- Notice blah_label get translated to 0x2….

Time for a class poll…

Conditional Branch (2)

- Considers only a single register for comparison and compares implicitly with zero
  - Branch Greater than or Equal to Zero (BGEZ)
  - Branch Greater than Zero (BGTZ)
  - Branch Less than or Equal to Zero (BLEZ)
  - Branch Less than Zero (BLTZ)
- Example:
  - BGTZ $t0, label
  - If $t0>0, execute instruction at label next
  - Otherwise, execute instruction after BGTZ