Computer Architecture
(ENE1004)

Lec – 9: Instructions: Language of the Computer (Chapter 2) - 8

Notice

- Midterm exam (tentative)
  - Apr. 24 (Monday) in regular class hours
  - No class on Apr. 20, Thursday
  - Sample questions will be provided on Apr. 21 or 22
- Assignment #1
  - Will be announced on Apr. 6 (Thursday)
  - Deadline: May. 7 (Sunday) at midnight

MIPS Addressing for 32-bit Immediates

- What is the MIPS assembly code to load this 32-bit constant into register **$t0**?

  0000 0000 0011 1101 0000 1001 0000 0000

  - ~~**addi $t0, $zero, 4000000**~~
- I-type instruction can express only 16-bit constants

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

  - The value of 4,000,000 does not fit into the 16-bit field
- Load upper immediate (**lui**): **lui $t0, 255** # 255 decimal = 0000 0000 1111 1111 binary
  - lui transfers the 16-bit constant field value into the leftmost 16 bits of the register
  - The lower 16 bits are filled with 0s

The machine language version of `lui $t0, 255    # $t0 is register 8:`

001111	00000	01000	0000 0000 1111 1111

Contents of register `$t0` after executing `lui $t0, 255:`

0000 0000 1111 1111	0000 0000 0000 0000

MIPS Addressing for 32-bit Immediates

- What is the MIPS assembly code to load this 32-bit constant into register **$t0**?

  $\boxed{0000\ \ 0000\ \ 0011\ \ 1101}\ \boxed{0000\ \ 1001\ \ 0000\ \ 0000}$

- Load upper immediate (**lui**): **lui $t0, 61** # 61decimal=0000 0000 0011 1101 binary
  - This loads the value of 61 onto the upper 16 bits of **$t0**
  - **$t0** = 0000 0000 0011 1101 0000 0000 0000 0000
- The next step is to insert the lower 16 bits with a binary value of 0000 1001 0000 0000
  - **ori $t0, $t0, 2304** # 2304 decimal = 0000 1001 0000 0000
  - $t0 = 0000 0000 0011 1101 0000 0000 0000 0000
  - 2304 = 0000 0000 0000 0000 0000 1001 0000 0000
  - The final value in $t0 is 0000 0000 0011 1101 0000 1001 0000 0000
- Two instructions, **lui** and **ori**, can collectively load a 32-bit constant into a register
  - **lui target_register upper_16_bit_value**
  - **ori target_register target_register lower_16_bit_value**

MIPS Addressing for 32-bit Addresses

- How do MIPS instructions express a memory address?
- Jump instruction (J-type)
  - **j 10000 # go to location 10000**

2	10000
6 bits	26 bits

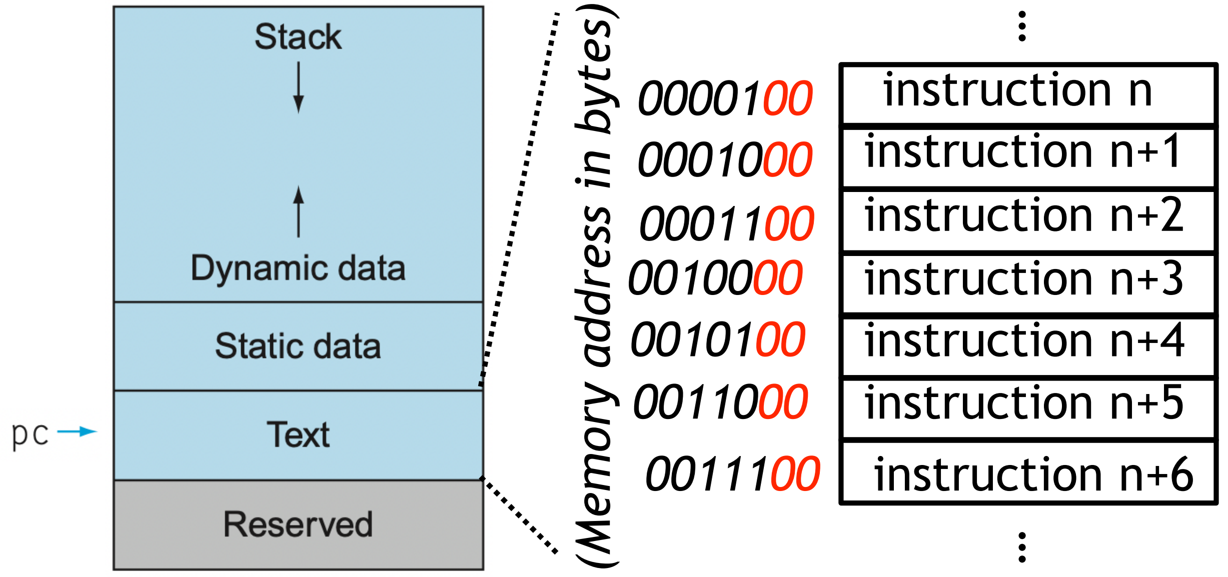
  - You have “26 bits” to express a memory address
  - What if a program is bigger than 2^26 bytes (an address larger than 2^26)?
- Conditional branch instructions (I-type)
  - **bne $s0, $s1, Exit # go to Exit if $s0 is not equal to $s1**

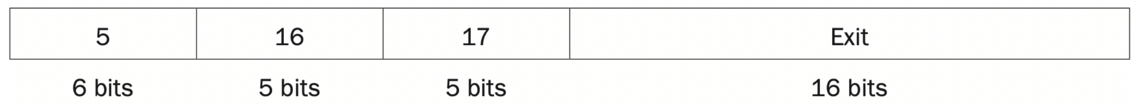
5	16	17	Exit
6 bits	5 bits	5 bits	16 bits

  - Here, you have only “16 bits” to express an address, which is much smaller than j-type
- Then, is there a way to express larger (e.g., 32-bit) memory addresses?

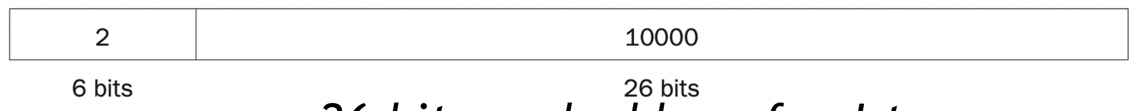
MIPS Addressing for 32-bit Addresses: in Word

- The address specifies a location where an instruction is stored in the text segment
  - **j 10000 # go to location 10000**
  - **bne $s0, $s1, Exit # go to Exit if $s0 is not equal to $s1**
  - 10000 and Exit indicate target instructions
- Due to the size (word) of instructions, we do not have to consider byte offset
  - The last two bits are always 00 (e.g., xxxx xxxx xxxx xxxx xx00), which wastes bits in the field
- If address is specified in word-address, we can express 4X larger address space





16-bit word address for I-type



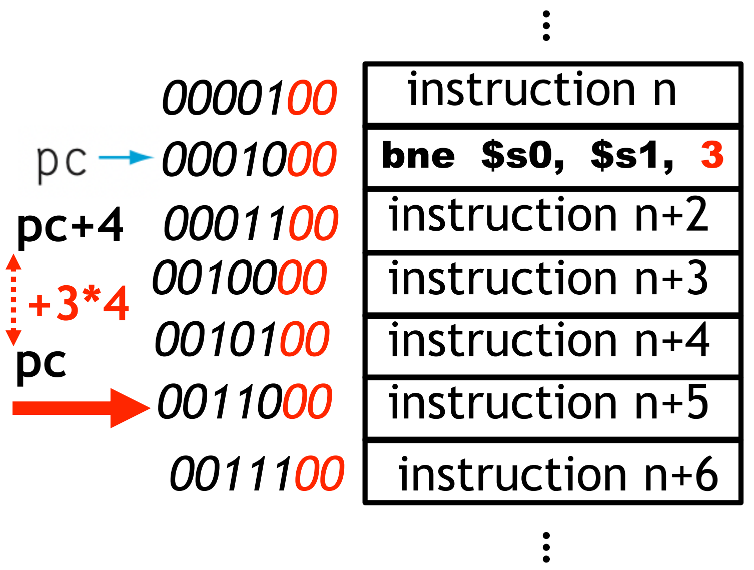
26-bit word address for J-type

PC-Relative Addressing for I-Type

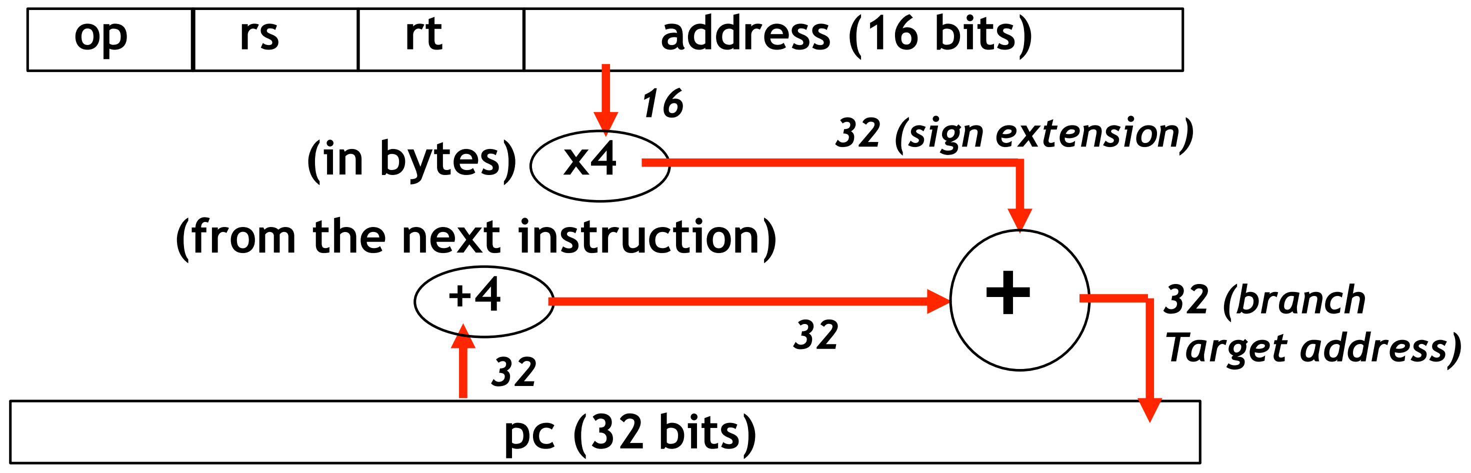
- The 16-bit field of I-type can still express only 2^16 words (and instructions)

5	16	17	Exit
6 bits	5 bits	5 bits	16 bits

- PC-relative addressing for I-type
  - Target address is specified based on PC (the address of the current instruction)
  - Target address (32-bit) = PC (32-bit) + branch offset (16-bit)
  - Actually, almost all loops and if statements are much smaller than 2^16 words
  - In MIPS, PC = (PC + 4) + (branch address in word * 4)

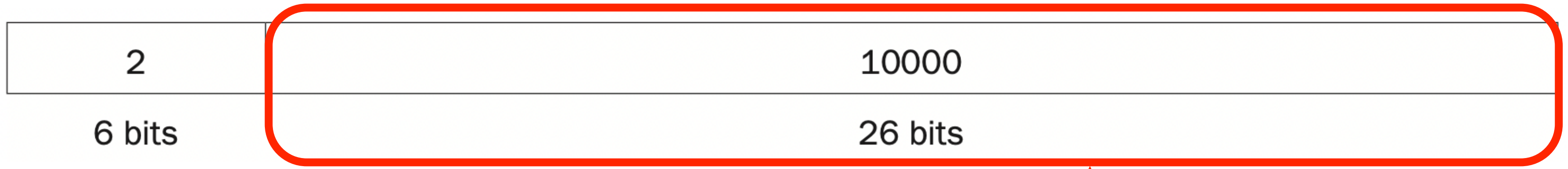


⋮
0000100 instruction n
pc → 0001000 bne $s0, $s1, 3
pc+4 0001100 instruction n+2
+3*4 0010000 instruction n+3
0010100 instruction n+4
pc 0011000 instruction n+5
0011100 instruction n+6
⋮



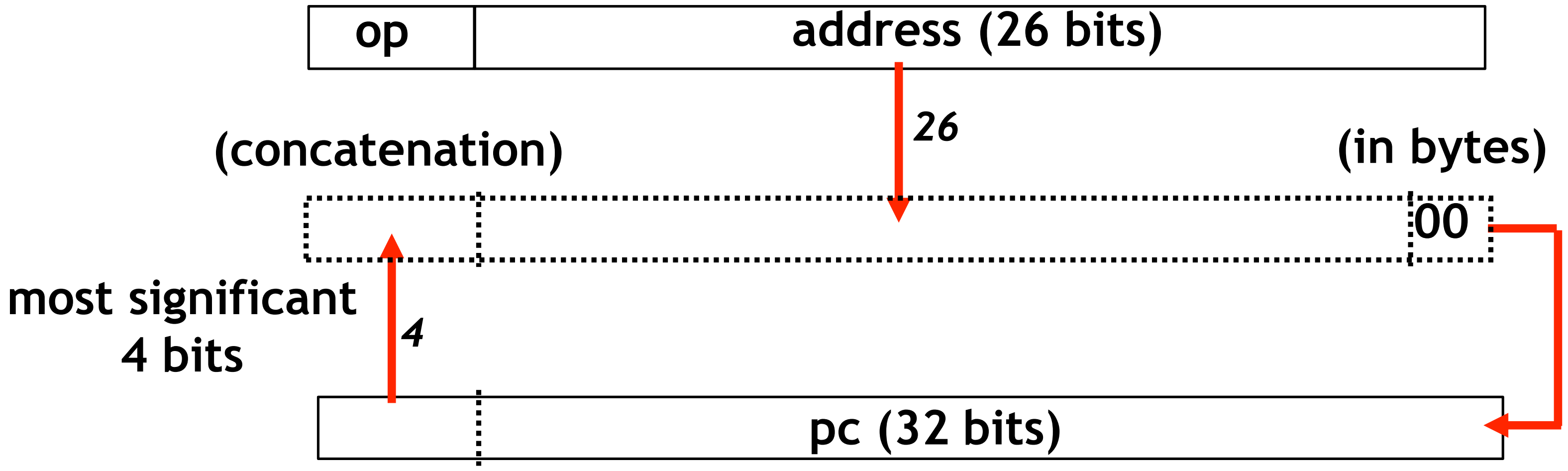
Pseudo-direct Addressing for J-Type

- The 26-bit field of J-type can still express only 2^26 words (and instructions)



2	10000
6 bits	26 bits

- Pseudo-direct addressing for J-type
  - Target address is specified partially based on PC (the address of the current instruction)
  - Target address (32-bit) = Upper 4 bits of PC (32-bit) ⊕ branch offset (26-bit)



MIPS Addressing for 32-bit Addresses: Example

```
Loop:   sll  $t1, $s3, 2      # Temp reg $t1 = 4 * i
        add  $t1, $t1, $s6    # $t1 = address of save[i]
        lw   $t0, 0($t1)      # Temp reg $t0 = save[i]
        bne  $t0, $s5, Exit   # go to Exit if save[i] ≠ k
        addi $s3, $s3, 1      # i = i + 1
        j    Loop             # go to Loop
Exit:
```

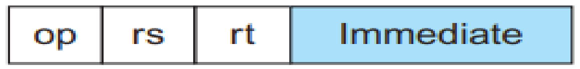
is assembled to

80000	0	0	19	9	2	0
80004	0	9	22	9	0	32
80008	35	9	8	0		
80012	5	8	21	2		
80016	8	19	19	1		
80020	2	20000				
80024	. . .					

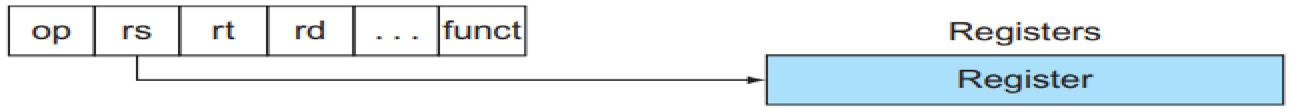
- Assumption: the loop starts at location 80000 in memory
- **bne $t0, $s5, Exit at 80012**
  - In PC-relative addressing mode, the target address (Exit) is set to 2
  - 80012+4 (the following instruction of PC) + 2*4 = 80024
- **j Loop at 80020**
  - In pseudo-direct addressing mode, the target address (Loop) is set to 20000
  - 0000 ⊕ 20000 * 4 = 80000

MIPS Addressing Modes

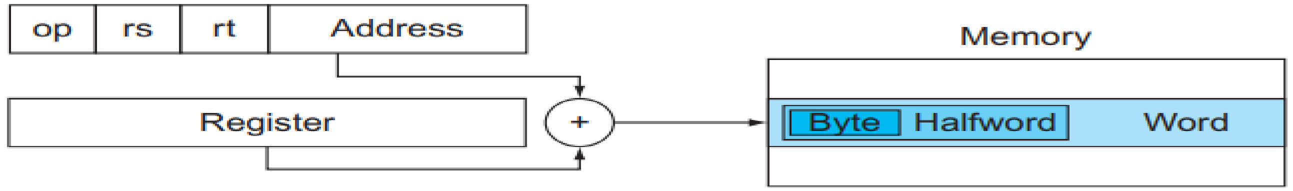
- Addressing mode: how machine instructions identify the operand(s)
- (1) Immediate addressing



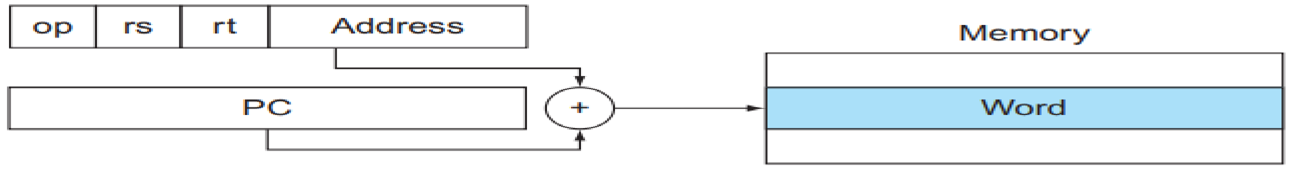
- (2) Register addressing



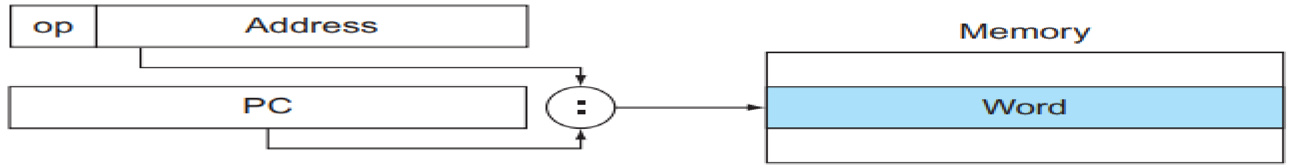
- (3) Base addressing



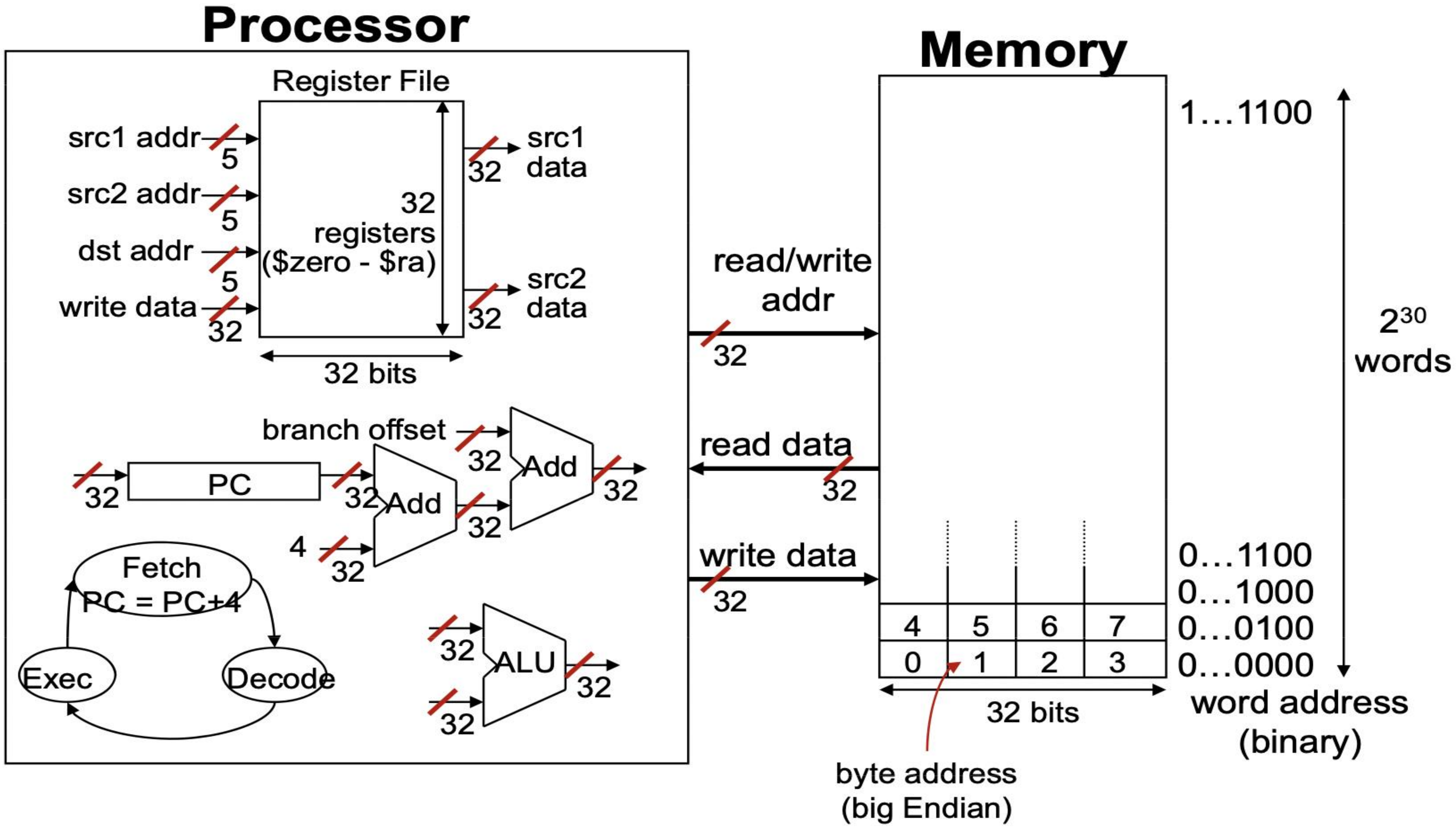
- (4) PC-relative addressing



- (5) Pseudo-direct addressing



MIPS Organization (Summary)


Processor
Register File
src1 addr
5
src2 addr
5
dst addr
5
write data
32
32 registers ($zero - $ra)
src1 data
32
src2 data
32
32 bits
branch offset
32
Add
32
PC
32
32
Add
32
4
32
Fetch PC = PC+4
Exec
Decode
32
ALU
32
32
read/write addr
32
read data
32
write data
32
Memory
1…1100
2^{30} words
0…1100
0…1000
0…0100
0…0000
4
5
6
7
0
1
2
3
32 bits
word address (binary)
byte address (big Endian)