Lecture 4
Addressing Modes

School of Computer Science and Engineering
Soongsil University

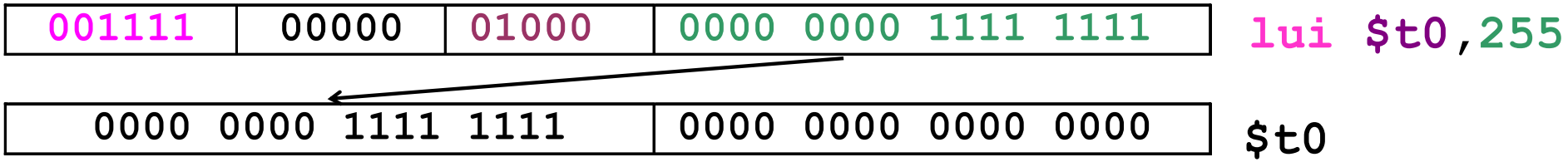
2. Instruction: Language of the Compute

2.9 MIPS Addressing for 32-Bit Immediates and Addresses

- **Addressing modes** (cf) Wikipedia
  - defines how machine language instructions in that architecture identify the operand (or operands) of each instruction
  - specifies how to calculate the effective memory address of an operand by using information held in registers and/or constants contained within a machine instruction or elsewhere
- **Number of addressing modes**
  - MIPS: 6 modes
  - ARMv7: 9 modes
  - IA-32: 12 modes

32-Bit Immediate Operands

- **Load upper immediate(lui) Instruction**



- **Example**

  $t0 <= 0000 0000 1111 1111 0000 1001 0000 0000

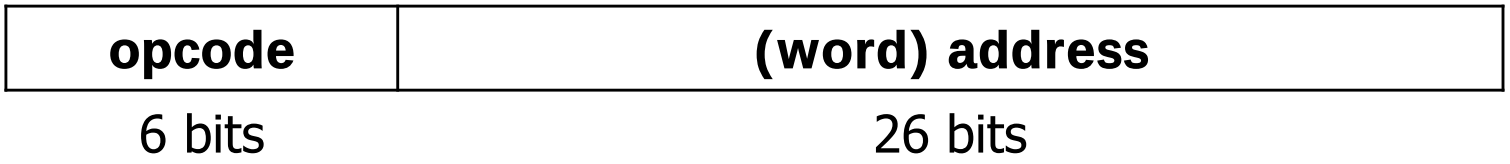
  li $t0, 0b 0000 0000 0011 1101 0000 1001 0000 0000

  [Answer]

```
lui   $t0,0b 0000 0000 1111 1111
ori   $t0,$t0,0b 0000 1001 0000 0000
```

Addressing in Branches and Jumps

- **J-type instruction format**

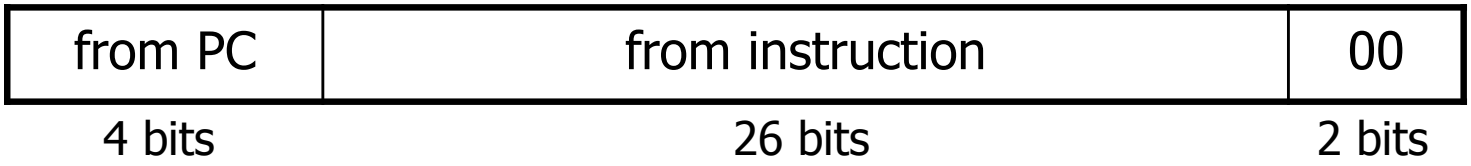


- **Direct addressing mode**
  - `j  10000  # branch address = 10000`
- **Word addressing**
  - Addressing in jump instruction
  - Alignment restriction → instruction address = 4n → LS 2 bits = 00

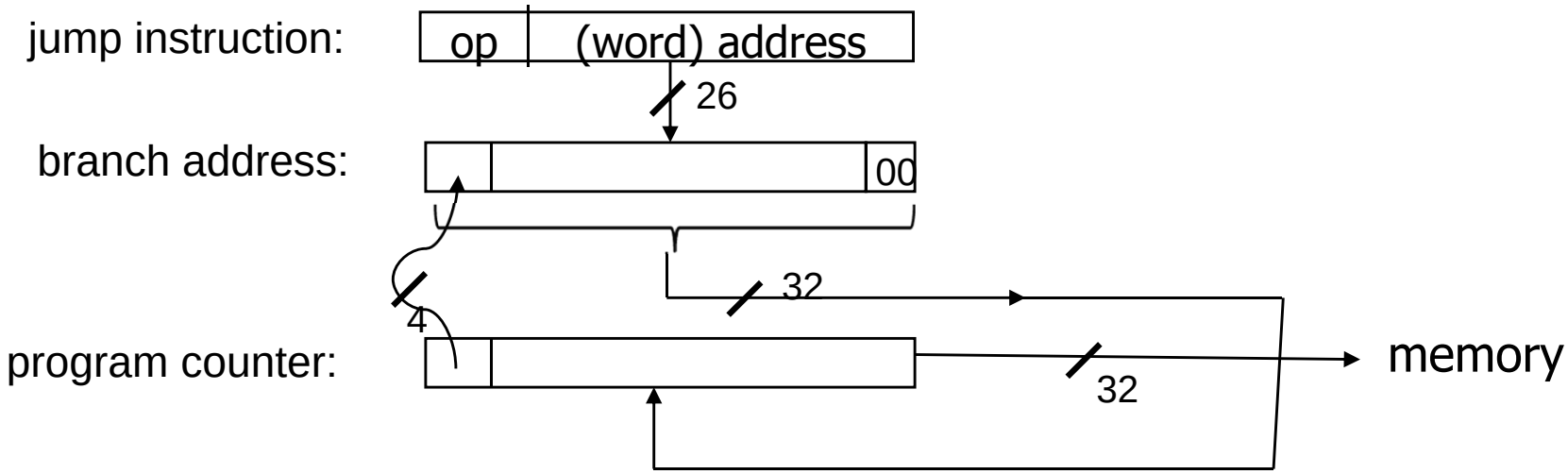
  - Word address
    - Branch address = address * 4
  - `j  10000  # branch address = 40000`

Pseudo-Direct Addressing Mode

- Upper 4 bits of PC are unchanged.
- Address boundary of 2^{28} B = 256 MB
- Jump address

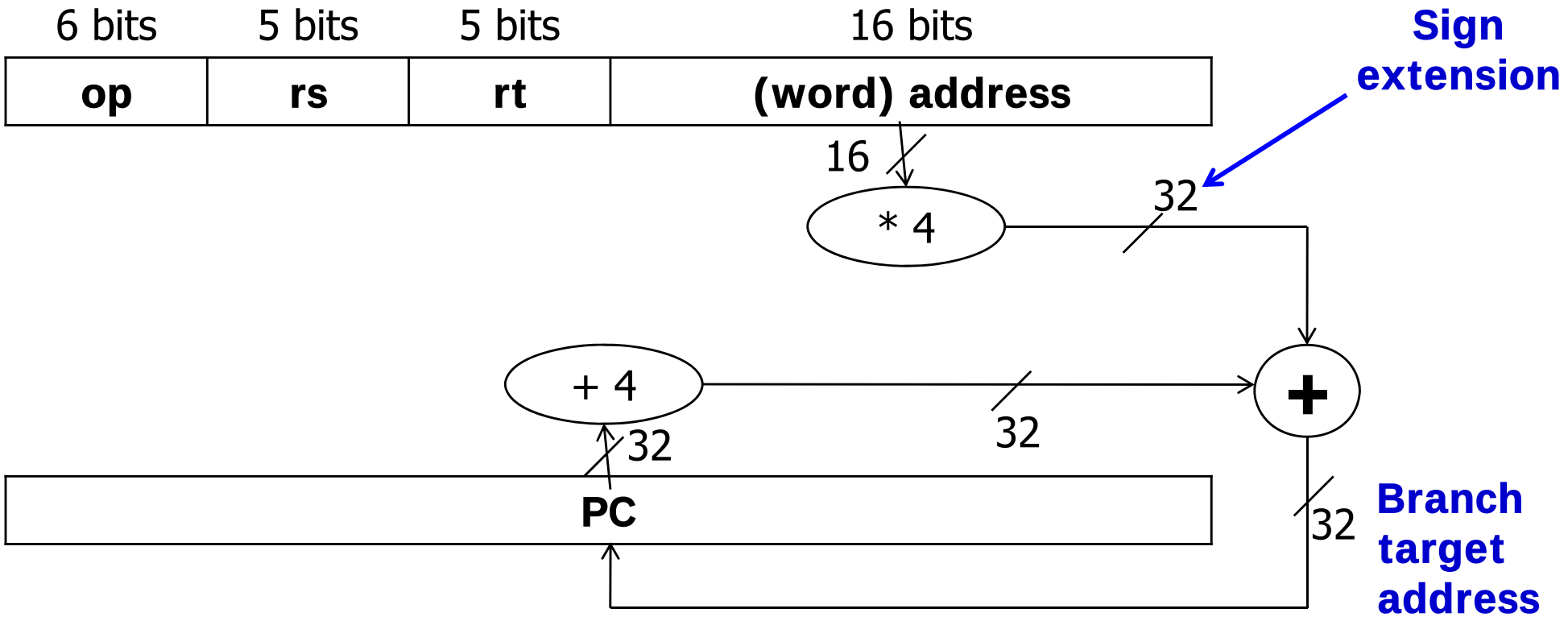


from the low order 26 bits of the jump instruction



PC-relative Addressing Mode

- Branch target address = (PC+4) + Branch offset
- New PC ≈ (PC+4) ± 2^{15} words = (PC+4) ± 2^{17} bytes
- PC - 131068 ≤ new PC (=branch target address) ≤ PC + 131072



Example: Showing Branch Offset

```
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:
```

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**				

Example: Branching Far Away

- `beq $s0,$s1,L1`

  But `L1` is too far.

[Answer]

```
        bne $s0,$s1,L2
        j   L1
   L2 :
```

MIPS Addressing Modes

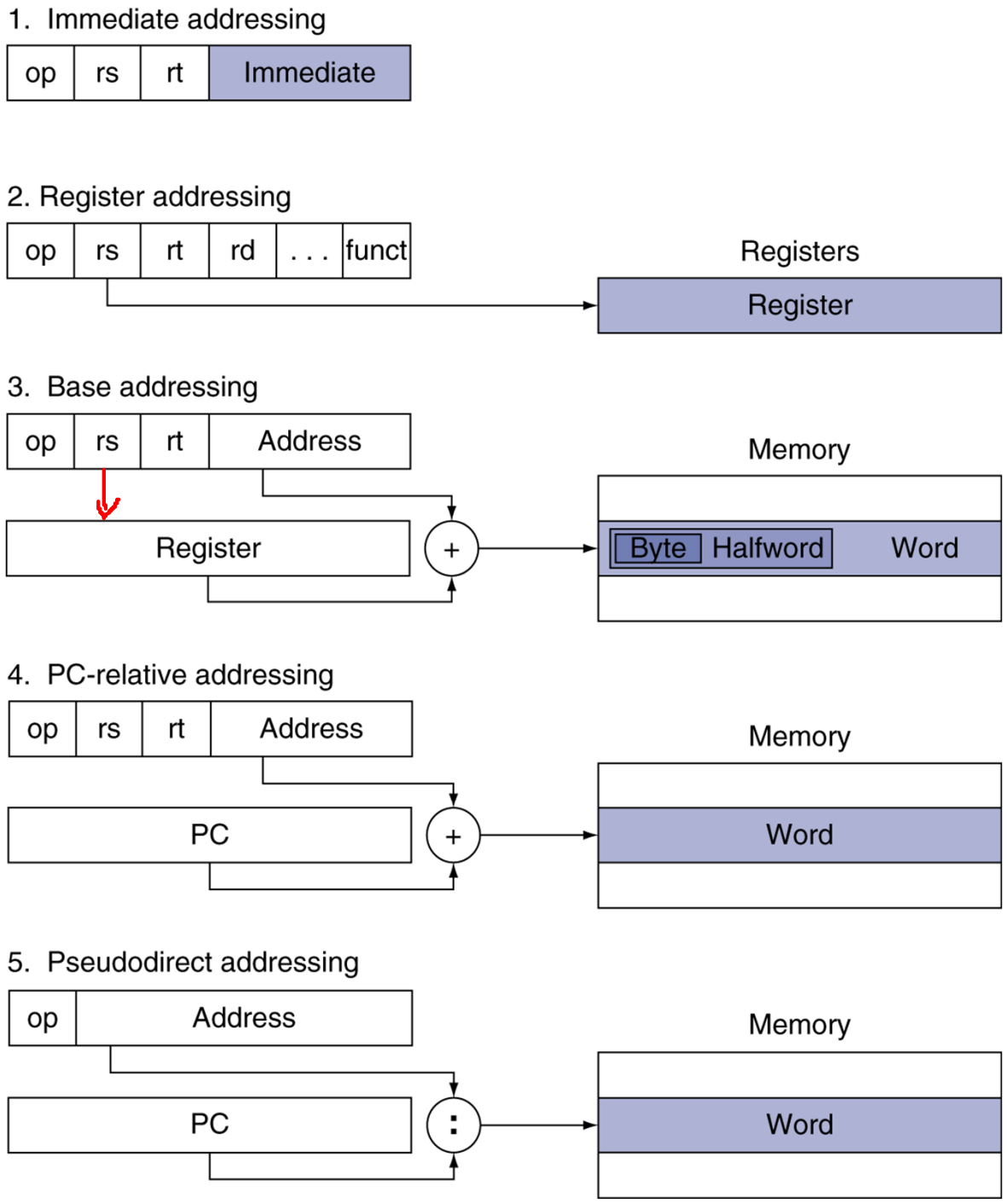


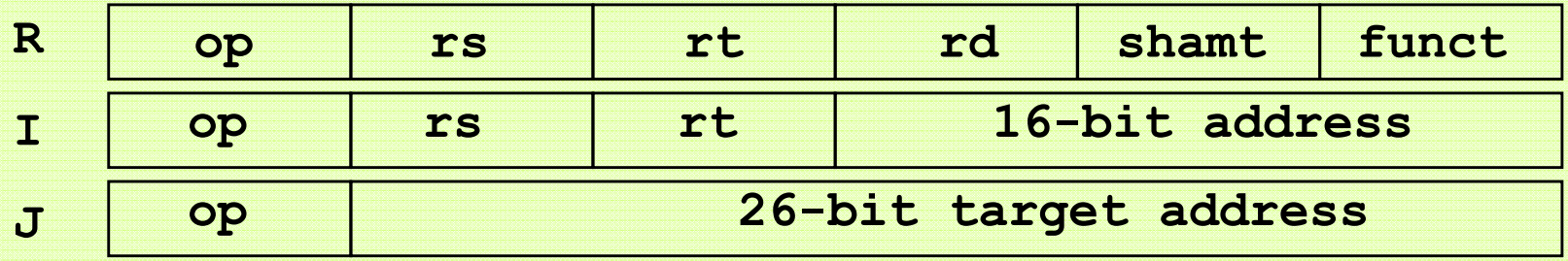
Figure 2.16

Summary

- **Instruction** **Meaning**

```
add $s1,$s2,$s3  $s1 = $s2 + $s3
sub $s1,$s2,$s3  $s1 = $s2 - $s3
lw $s1,100($s2)  $s1 = Memory[$s2+100]
sw $s1,100($s2)  Memory[$s2+100] = $s1
bne $s4,$s5,L    Next instr. is at Label if $s4≠$s5
beq $s4,$s5,L    Next instr. is at Label if $s4=$s5
j Label          Next instr. is at Label
```

- **Instruction formats**



MIPS Instruction Encoding (1)

op(31:26)								
28–26 31–29	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	R-format	Bltz/gez	jump	jump & link	branch eq	branch ne	blez	bgtz
1(001)	add immediate	addiu	set less than imm.	set less than imm. unsigned	andi	ori	xori	load upper immediate
2(010)	TLB	FlPt						
3(011)								
4(100)	load byte	load half	lwl	load word	load byte unsigned	load half unsigned	lwr	
5(101)	store byte	store half	swl	store word			swr	
6(110)	load linked word	lwc1						
7(111)	store cond. word	swc1						

Figure 2.17

MIPS Instruction Encoding (2)

op(31:26)=010000 (TLB), rs(25:21)								
23–21 25–24	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(00)	mfc0		cfc0		mtc0		ctc0	
1(01)								
2(10)								
3(11)								

op(31:26)=000000 (R-format), funct(5:0)								
2–0 5–3	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	shift left logical		shift right logical	sra	sllv		srlv	srav
1(001)	jump register	jalr			syscall	break		
2(010)	mfhi	mthi	mflo	mtlo				
3(011)	mult	multu	div	divu				
4(100)	add	addu	subtract	subu	and	or	xor	not or (nor)
5(101)			set l.t.	set l.t. unsigned				
6(110)								
7(111)								

Figure 2.17

Assembler Pseudo-instructions

- Most assembler instructions represent machine instructions one-to-one
- Pseudoinstructions
  - Figments of the assembler's imagination
  - $at (register 1): assembler temporary

```
move $t0, $t1       ⇨   add $t0, $zero, $t1

blt $t0, $t1, L     ⇨   slt $at, $t0, $t1
                        bne $at, $zero, L
```

Decoding Machine Language

- **Example**

  $00AF8020_{hex}$

[Answer]

0000 0000 1010 1111 1000 0000 0010 0000

Opcode=000000 -> R-type

000000	00101	01111	10000	00000	100000
op	rs	rt	rd	shamt	funct

Funct = 100000 -> **`add`** (See Fig. 2.25)

`$5 = $a1, $15 = $t7, $16 = $s0` (See Fig. 2.18)

```
add $s0,$a1,$t7
```

2.18 Concluding Remarks

1. **Simplicity favors regularity.**
   - fixed size instructions
   - small number of instruction formats
   - opcode always the first 6 bits
2. **Smaller is faster.**
   - limited instruction set
   - limited number of registers in register file
   - limited number of addressing modes
3. **Good design demands good compromises.**
   - three instruction formats

Instruction Categories

Instruction class	MIPS example	Frequency	
		Integer	FP
Arithmetic	`add,sub,addi`	16%	48%
Data transfer	`lw,sw,lb,lbu,lh, lhu,sb,lui`	35%	36%
Logical	`and,or,nor,andi, ori,sll,srl`	12%	4%
Conditional branch	`beq,bne,slt,slti, sltiu`	34%	8%
Jump	`j,jr,jal`	2%	0%

Figure 2.41

MIPS Instructions

MIPS instructions	Name	Format	Pseudo MIPS	Name	Format
add	add	R	move	move	R
subtract	sub	R	multiply	mult	R
add immediate	addi	I	multiply immediate	multi	I
load word	lw	I	load immediate	li	I
store word	sw	I	branch less than	blt	I
load half	lh	I	branch less than or equal	ble	I
load half unsigned	lhu	I			
store half	sh	I	branch greater than	bgt	I
load byte	lb	I	branch greater than or equal	bge	I
load byte unsigned	lbu	I			
store byte	sb	I			
load linked	ll	I			
store conditional	sc	I			
load upper immediate	lui	I			
and	and	R			
or	or	R			
nor	nor	R			
and immediate	andi	I			
or immediate	ori	I			
shift left logical	sll	R			
shift right logical	srl	R			
branch on equal	beq	I			
branch on not equal	bne	I			
set less than	slt	R			
set less than immediate	slti	I			
set less than immediate unsigned	sltiu	I			
jump	j	J			
jump register	jr	R			
jump and link	jal	J			

Figure 2.40

MIPS Organization So Far