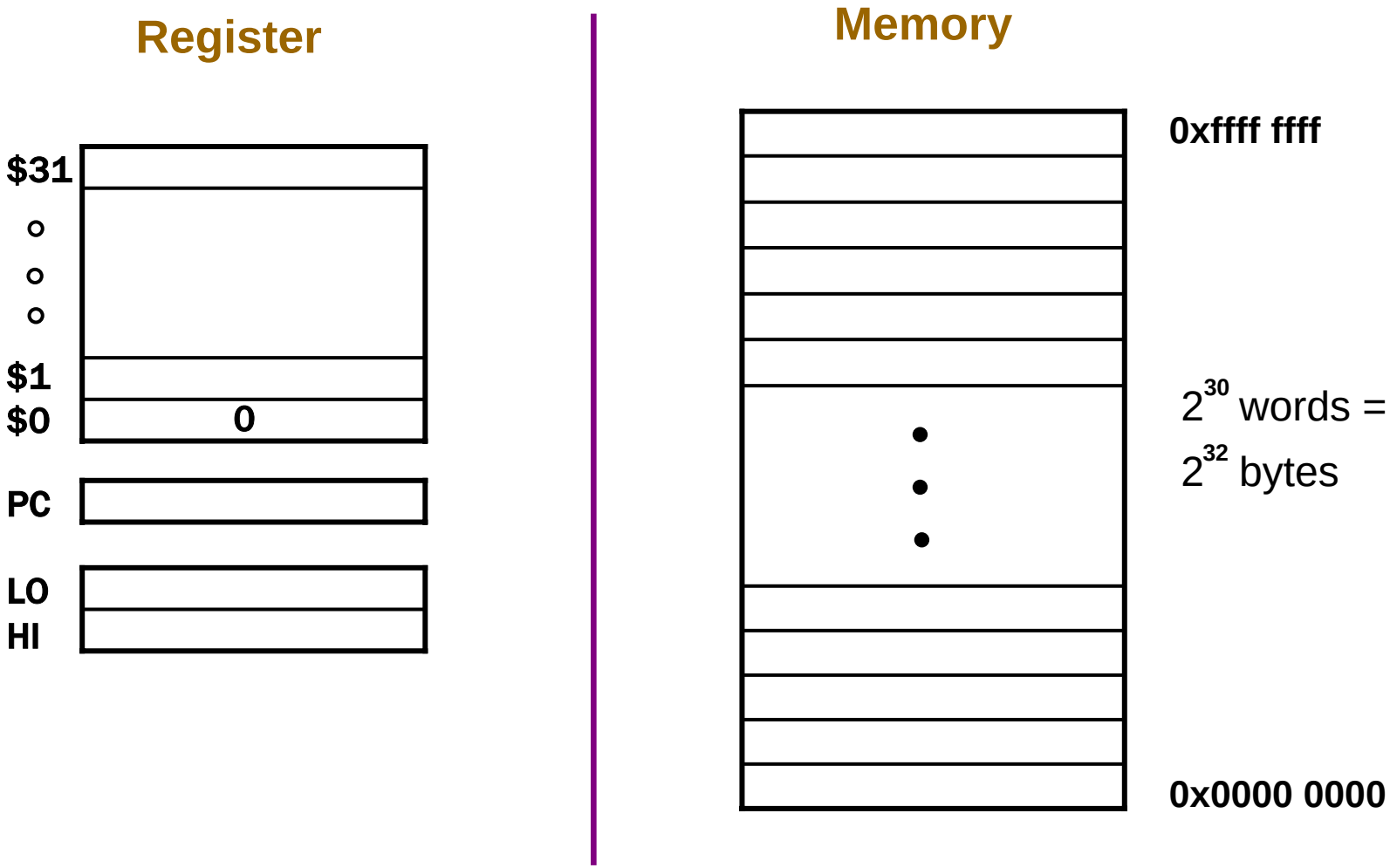
MIPS

Instruction Set Architecture

MIPS Register & Memory Model



MIPS Register Naming

number	name	usage
$0	zero	constant 0 (항상 0만 저장)
$1	at	Assembler가 사용함
$2 ~ $3	v0, v1	수식 계산 및 함수 결과값 저장
$4 ~ $7	a0 ~ a3	arguments
$8 ~ $15	t0 ~ t7	temporary
$16 ~ $23	s0 ~ s7	saved (preserved across call)
$24, $25	t8, t9	temporary
$26, $27	k0, k1	OS가 사용함
$28	gp	pointer for global area
$29	sp	stack pointer
$30	fp	frame pointer
$31	ra	return address

MIPS Instructions

- **Arithmetic/Logic instructions**
- **Data Transfer (Load/Store) instructions**
- **Conditional branch instructions**
- **Unconditional jump instructions**

MIPS Instruction Format

Fields						Comments
6bits	5bits	5bits	5bits	5bits	6bits	All MIPS insturctions 32 bits
op	rs	rt	rd	shamt	funct	Arithmetic instruction format
op	rs	rt	address/immediate			Transfer, branch, imm. format
op	target address					Jump instruction format

Arithmetic Operations

Instruction	Example	Meaning	Comments
add	add $s1, $s2, $s3	$s1= $s2 + $s3	Three operands;
subtract	sub $s1, $s2, $s3	$s1= $s2 - $s3	Three operands;
add immediate	addi $s1, $s2, 100	$s1= $s2 + 100	+ constant;

Arithmetic Operations

Instruction	Example	Meaning	Comments
multiply	mult $s2, $s3	HI, LO = $s2 * $s3	64 bit product
divide	div $s2, $s3	LO = $s2 / $s3 HI = $s2 % $s3	LO ← quotient HI ← remainder
move from HI	mfhi $s1	$s1 ← HI	Copy the contents of HI to $s1
move from LO	mflo $s1	$s1 ← LO	Copy the contents of LO to $s1

Logic Operations

Instruction	Example	Meaning	Comments
and	and $s1, $s2, $s3	$s1 = $s2 & $s3	Three reg. operands; logical AND
or	or $s1, $s2, $s3	$s1 = $s2 | $s3	Three reg. operands; logical OR
xor	xor $s1, $s2, $s3	$s1 = $s2 ^ $s3	Three reg. operands; logical EXCLUSIVE OR
and immediate	andi $s1, $s2, 100	$s1 = $s2 & 100	Logical AND reg, constant
or immediate	ori $s1, $s2, 100	$s1 = $s2 | 100	Logical OR reg, constant

Data Transfer Instructions

Instruction	Example	Meaning	Comments
load word	lw $s1, 100($s2)	$s1 = Memory[$s2 + 100]	word from memory to register
store word	sw $s1, 100($s2)	Memory[$s2 + 100] = $s1	word from register to memory
load byte unsigned	lbu $s1, 100($s2)	$s1 = Memory[$s2 + 100]	Byte from memory to register
store byte	sb $s1, 100($s2)	Memory[$s2 + 100] = $s1	Byte from register to memory

MIPS Addressing

- 모든 메모리 접근은 **load/store**를 통해서만 가능

- 메모리로부터 **halfword** 또는 **byte** 단위 **load** 시
  — signed operation의 경우 sign-extended
  — unsigned operation의 경우 zero-extended

- **Alignment restriction**
  — Word address는 4의 배수여야 함
  — Halfword address는 2의 배수여야 함

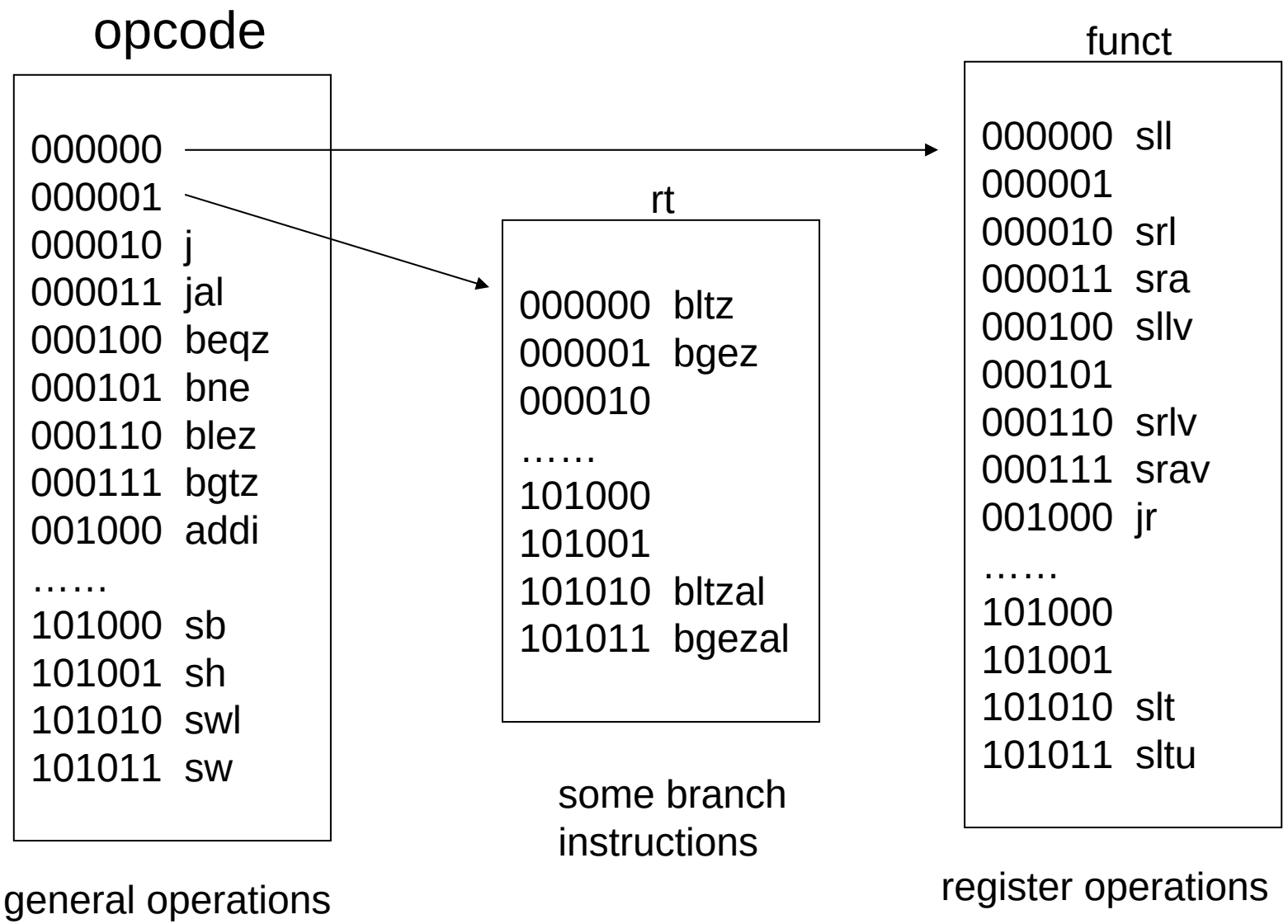
Conditional Branch Instructions

Instruction	Example	Meaning	Comments
branch on equal	beq $s1, $s2, 25	if($s1== $s2)go to PC+4+100	Equal test; PC-relative branch
branch on not equal	bne $s1, $s2, 25	if($s1!= $s2)go to PC+4+100	Not equal test; PC-relative
set on less than	slt $s1, $s2, $s3	if($s2<$s3) $s1 =1; else $s1=0	Compare less than; two's complement
set less than immediate	slti $s1, $s2, 100	if($s2<100) $s1 =1; else $s1=0	Compare < constant; two's complement

Unconditional Jump Instructions

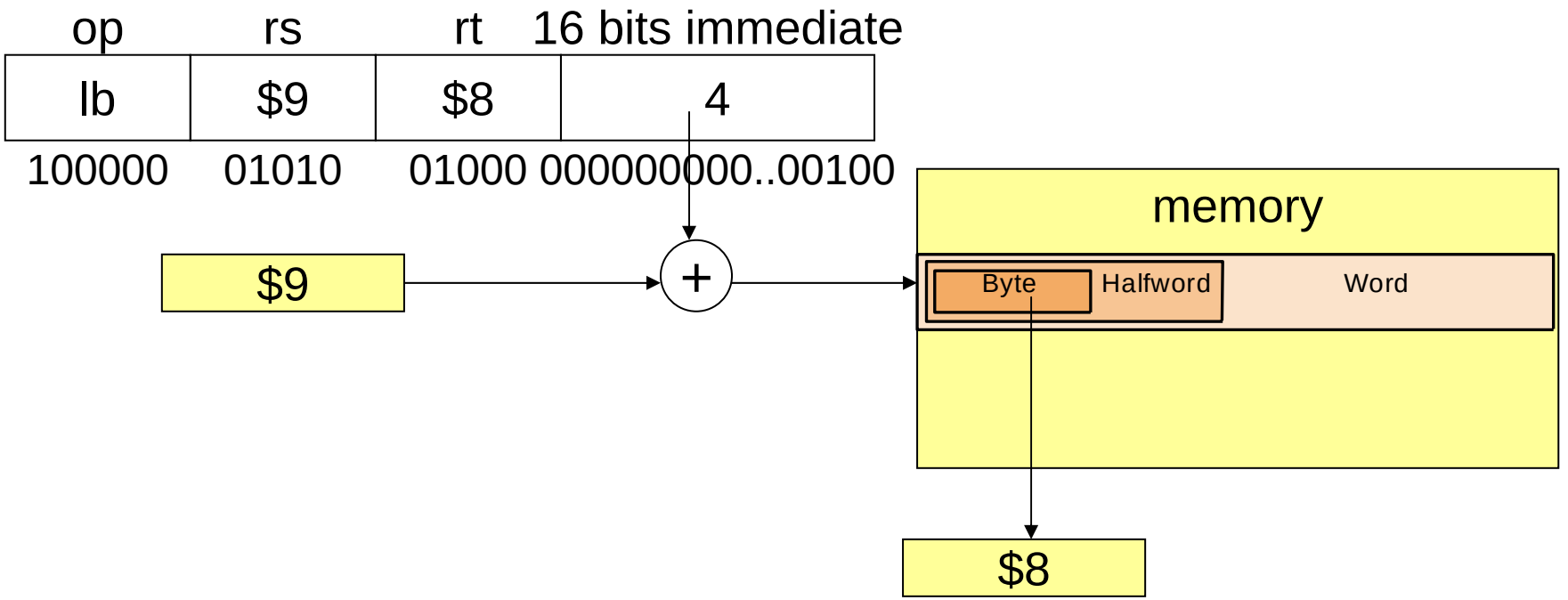
Instruction	Example	Meaning	Comments
jump	j 2500	go to 10000	Jump to target address
jump register	jr $ra	go to $ra	For switch, procedure return

opcode structure



Base Addressing

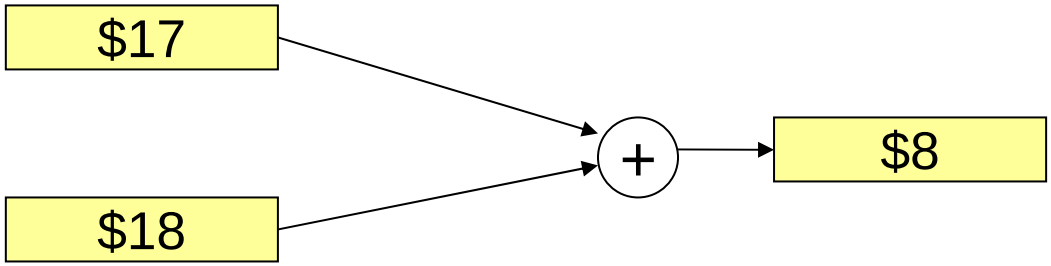
lb $8, 4($9) # load a byte from memory into $8



Register Addressing

add $8, $17, $18 **# $8 = $17 + $18**

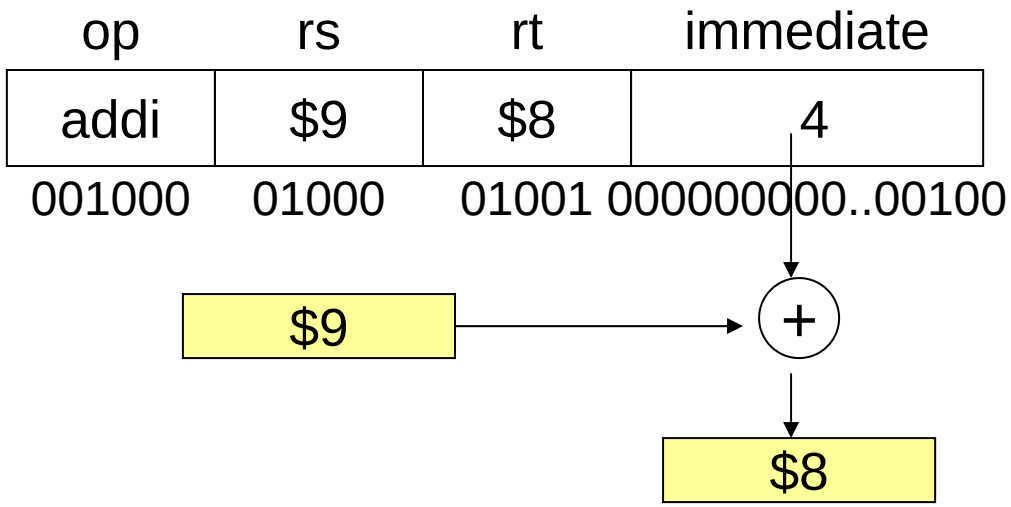
op	rs	rt	rd	shamt	function
000000	$17	$18	$8	unused	add
	17	18	8	0	32



- **function field**가 실제 **operation**을 결정 **!**

Immediate addressing

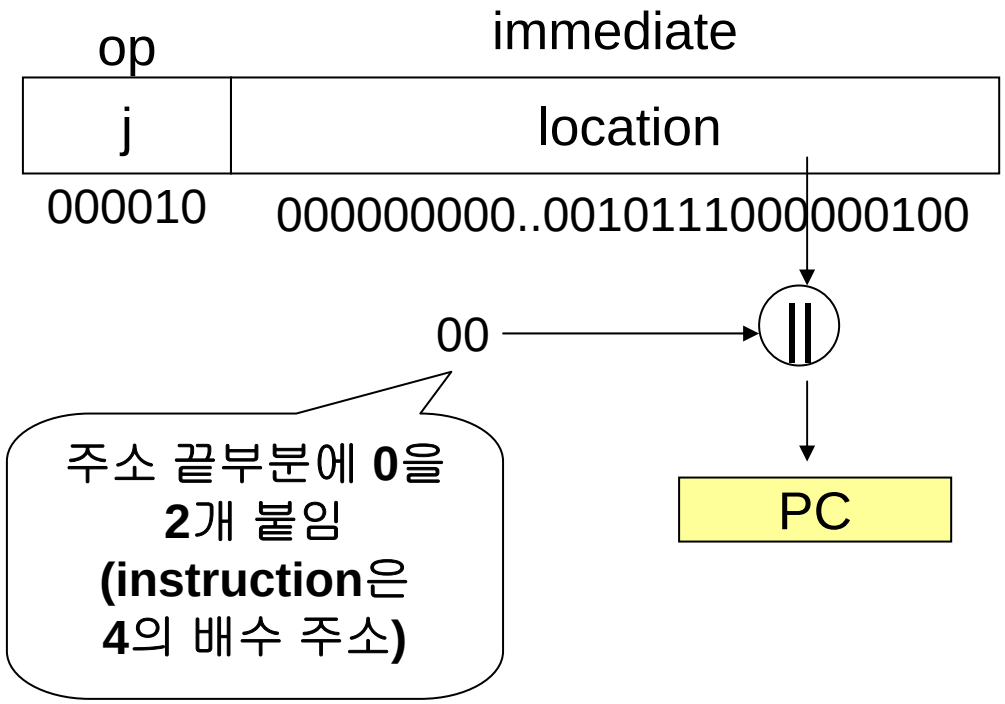
addi $8, $9, 4 **# $8 = $9 + 4**



- Immediate는 instruction 자체에 직접 포함되는 상수 필드
  - 재사용 불가
  - 32 bit 내에 포함되므로 용량의 한계
  - 추가적 메모리 접근이 없어 빠른 연산 가능

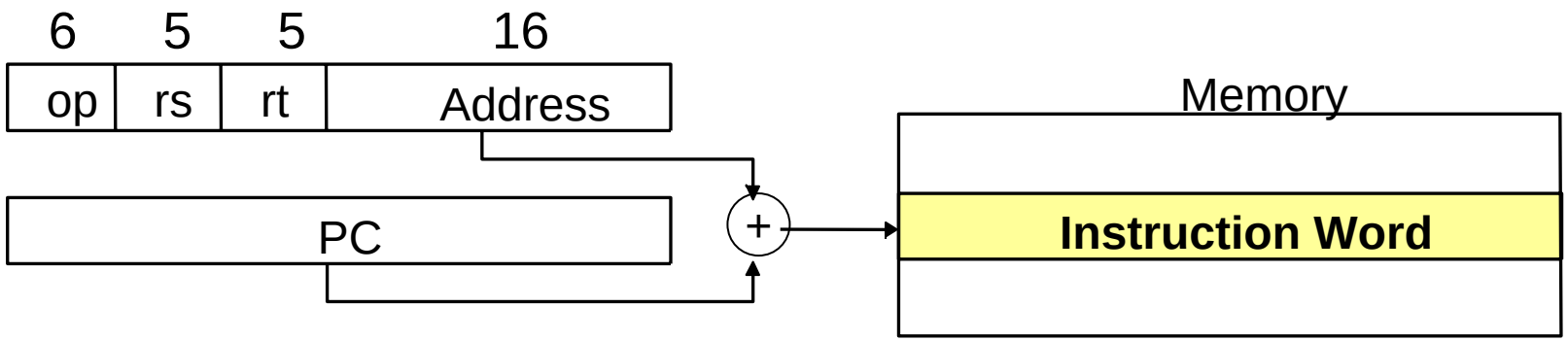
Jump Instruction

j next_location # jump to the instruction at next_location



PC Relative Addressing

beq $5, $7, XXX **# jump to XXX if $5 == $7**



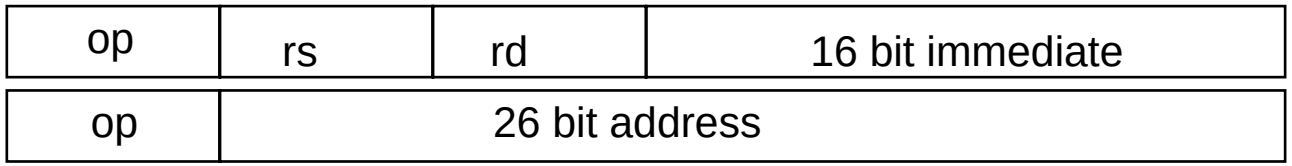
- **address field**에 **PC value**를 더해서 **target address** 결정
  - —there are only 16 bits left for the address !!
  - —there is a program locality

Branch and Jump

- **Instructions:**

```
bne $8,$9,Label        # Jump to Label if $8 != $9
beq $8,$9,Label        # Jump to Label if $8 = $9
j Label                # Jump to Label
```

- **Formats:**



- **Addresses are not always 32 bits**

  - branch target address = immediate value + PC
  - jump target address = immediate value

C vs. Assembly

```
f = (g + h) - (i + j);

f is mapped to s0.
g is mapped to s1.
h is mapped to s2.
i is mapped to s3.
j is mapped to s4.
```

```
add $t0, $s1, $s2
add $t1, $s3, $s4
sub $s0, $t0, $t1
```

C vs. Assembly

```
g = h + A[i];

g is mapped to s1.
h is mapped to s2.
s3 contains the base
   address of array A[].
i is mapped to s4.
```

```
add $t1, $s4, $s4
add $t1, $t1, $t1
add $t1, $t1, $s3
lw  $t0, 0($t1)
add $s1, $s2, $t0
```

C vs. Assembly

```
if (i==j)
 f = g + h;
else
 f = g - h;

f is mapped to s0.
g is mapped to s1.
h is mapped to s2.
i is mapped to s3.
j is mapped to s4.
```

```
      bne $s3, $s4, Else
      add $s0, $s1, $s2
      j   Exit
Else: sub $s0, $s1, $s2
Exit:
```

C vs. Assembly

```
while (save[i] == k)
    i = i + j;

i is mapped to s3.
j is mapped to s4.
k is mapped to s5.
s6 contains the base
   address of array
   save[].
```

```
Loop: add $t1, $s3, $s3
      add $t1, $t1, $t1
      add $t1, $t1, $s6
      lw  $t0, 0($t1)
      bne $t0, $s5, Exit
      add $s3, $s3, $s4
      j   Loop
Exit:
```