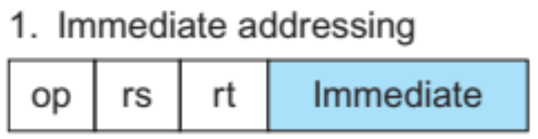
MIPS addressing modes and Operations

MIPS Addressing modes

- Addressing mode – a method for evaluating an operand.
- MIPS Addressing Modes

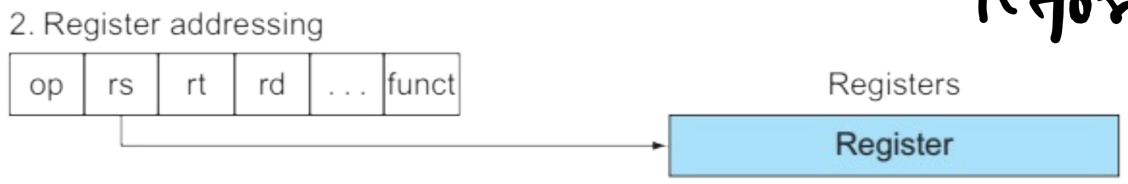
1. Immediate addressing, where the operand is a constant within the instruction itself



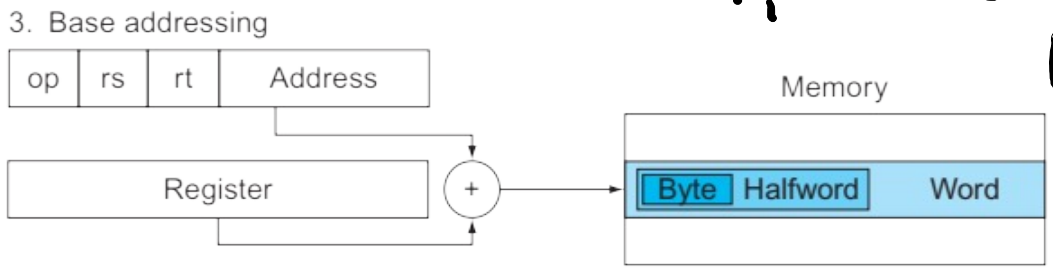
- addi, addiu

2. Register addressing, where the operand is a register

- R format



3. Base or displacement addressing, where the operand is at the memory location whose address is the sum of a register and a constant in the instruction — load, store



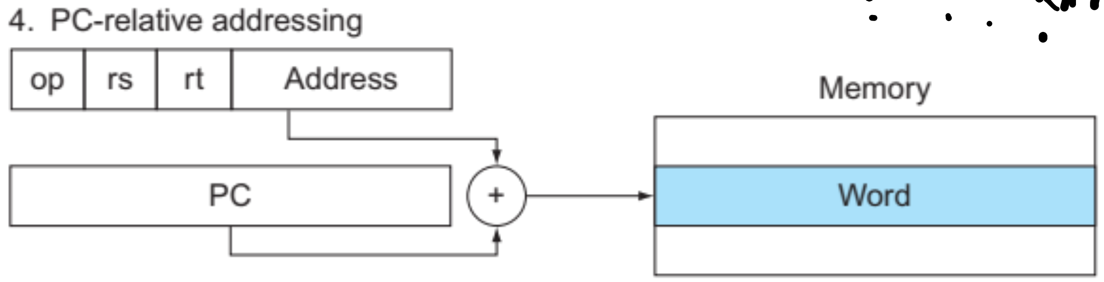
lw $s0, 100($s1)

$s1 + 100

4. PC-relative addressing, where the branch address is the sum of the PC and a constant in the instruction — branch

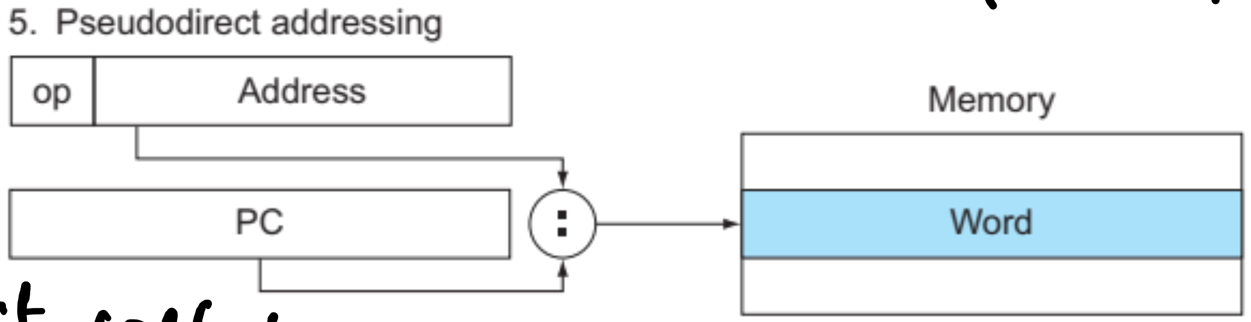
bne R1, R2, 1234

+ PC



5. Pseudo direct addressing, where the jump address is the 26 bits of the instruction concatenated with the upper bits of the PC

- J format



J target offset + PC = final

J target offset + PC → final address

Solving MIPS Instructions – Arithmetic and Logical Form

Form 1: `<operation> <dstreg>, <src1reg>, <src2reg>`

Example	Meaning	Comment
addu \$t0, \$t1, \$t2	\$t0 = \$t1 + \$t2	Addition (without overflow)
subu \$t1, \$t2, \$t3	\$t1 = \$t2 - \$t3	Subtraction (without overflow)

Form 2: `<operation> <dstreg>, <srcreg>, <constant>`

Example	Meaning	Comment
addiu \$t1,\$t2,1	\$t1 = \$t2 + 1	Addition immediate (without overflow)

- Consider the following C++ source code fragment.

```
unsigned int f,g,h,i,j;
...
f = (g+h)-(i+j);
```

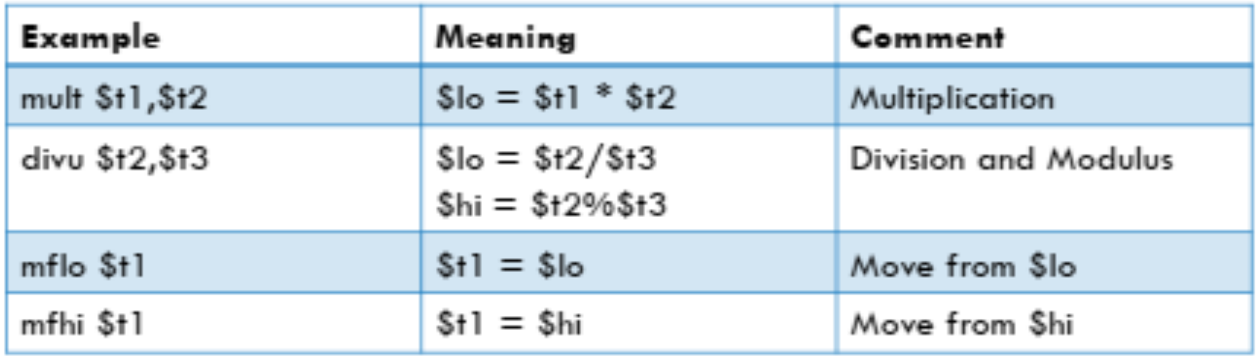
- Assume the values of f, g, h, i, and j are associated with registers $t2, $t3, $t4, $t5, and $t6 respectively. Write MIPS assembly code to perform this assignment assuming $t7 is available.

Solution :

```
Addu $t2,$t3,$t4  # $t2=g+h
Addu $t7,$t5,$t6   #t7=i+j
Subu  $t2,$t2,$t7   #t2=$t2-$t7
```

Multiply, Divide and Modulus instructions

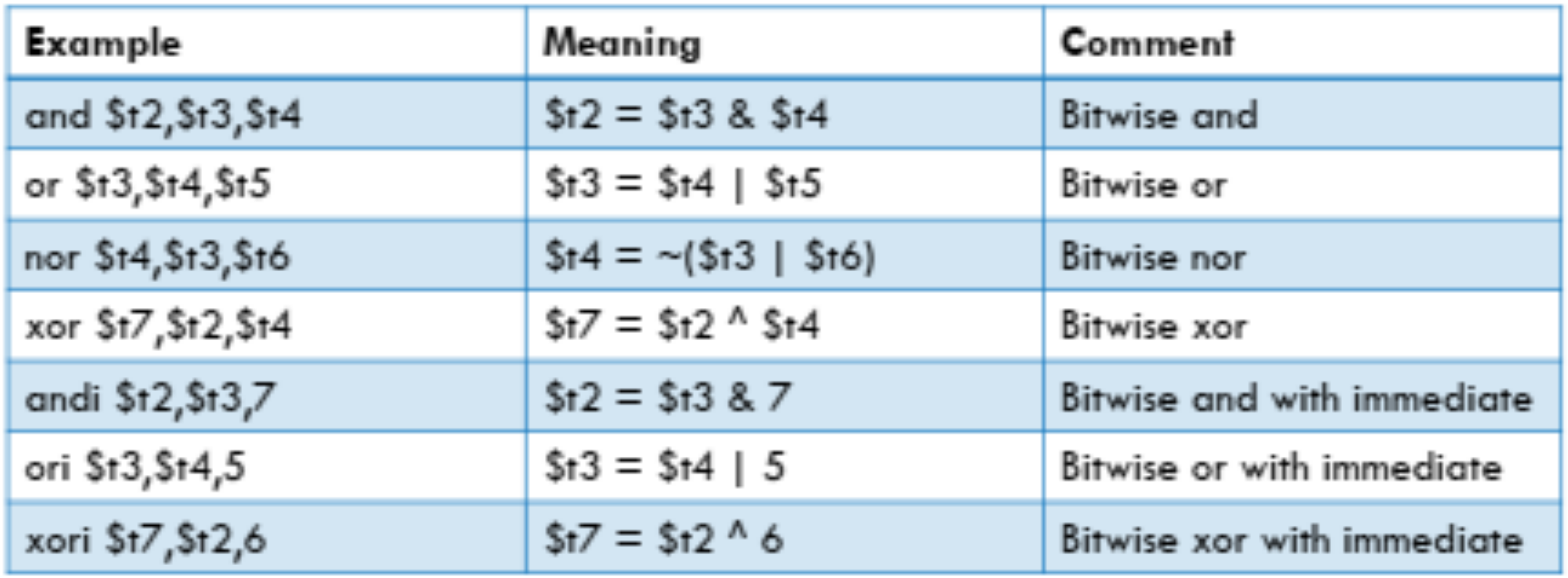
- Integer multiplication, division, and modulus operations can also be performed.
- MIPS provides two extra registers, hi and lo, to support division and modulus operations.
- hi and lo are not directly addressable, instead must use mfhi and mflo instructions



Example	Meaning	Comment
mult \$t1,\$t2	\$lo = \$t1 * \$t2	Multiplication
divu \$t2,\$t3	\$lo = \$t2/\$t3 \$hi = \$t2%\$t3	Division and Modulus
mflo \$t1	\$t1 = \$lo	Move from \$lo
mfhi \$t1	\$t1 = \$hi	Move from \$hi

General form of MIPS Bitwise instructions

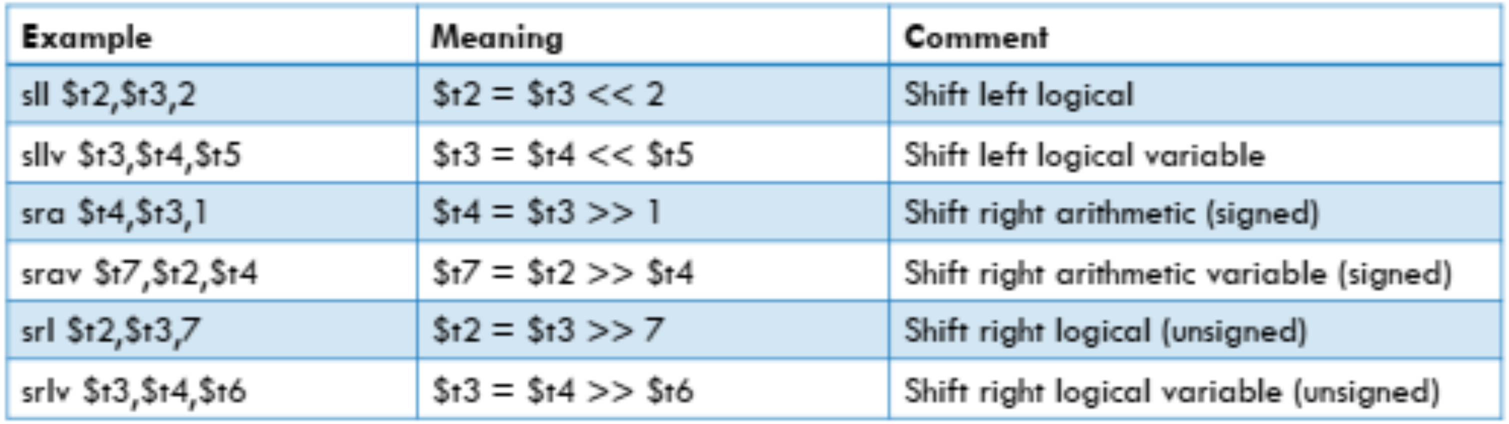
- Bitwise instructions apply Boolean operations on each of the corresponding pairs of bits of two values.



Example	Meaning	Comment
and \$t2,\$t3,\$t4	\$t2 = \$t3 & \$t4	Bitwise and
or \$t3,\$t4,\$t5	\$t3 = \$t4 | \$t5	Bitwise or
nor \$t4,\$t3,\$t6	\$t4 = ~(\$t3 | \$t6)	Bitwise nor
xor \$t7,\$t2,\$t4	\$t7 = \$t2 ^ \$t4	Bitwise xor
andi \$t2,\$t3,7	\$t2 = \$t3 & 7	Bitwise and with immediate
ori \$t3,\$t4,5	\$t3 = \$t4 | 5	Bitwise or with immediate
xori \$t7,\$t2,6	\$t7 = \$t2 ^ 6	Bitwise xor with immediate

General form of MIPS Shift instructions

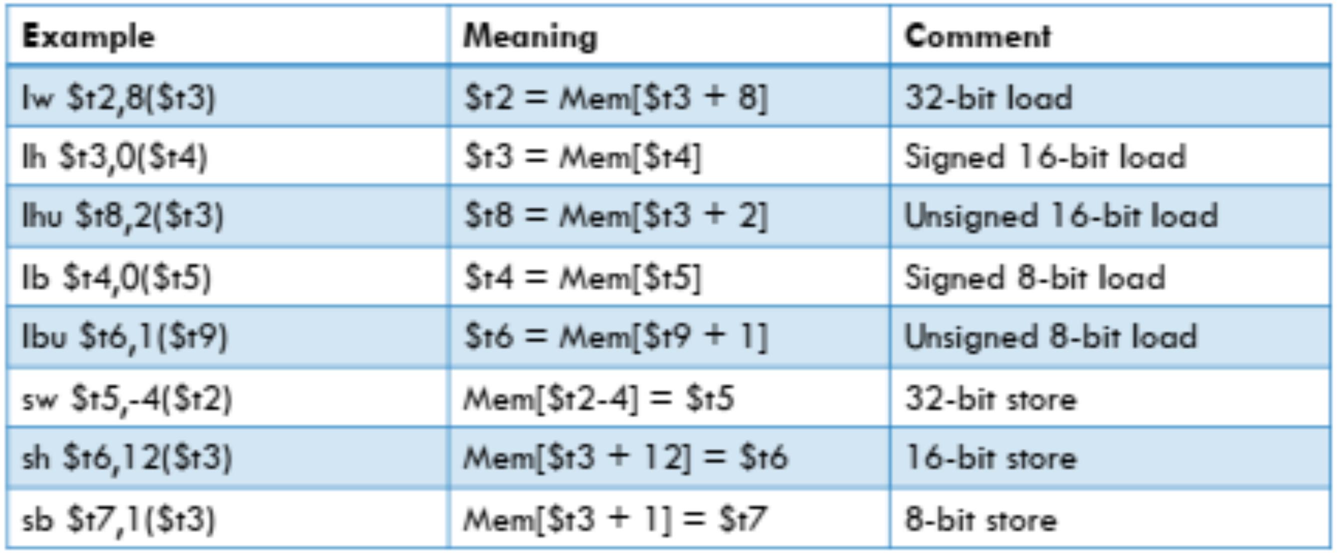
- Shift instructions move the bits in a word to the left or right by a specified amount.
- An arithmetic right shift replicates the most significant bit to fill in the vacant bits.
- A logical right shift fills in the vacant bits with zero.



Example	Meaning	Comment
sll \$t2,\$t3,2	\$t2 = \$t3 << 2	Shift left logical
sllv \$t3,\$t4,\$t5	\$t3 = \$t4 << \$t5	Shift left logical variable
sra \$t4,\$t3,1	\$t4 = \$t3 >> 1	Shift right arithmetic (signed)
srav \$t7,\$t2,\$t4	\$t7 = \$t2 >> \$t4	Shift right arithmetic variable (signed)
srl \$t2,\$t3,7	\$t2 = \$t3 >> 7	Shift right logical (unsigned)
srlv \$t3,\$t4,\$t6	\$t3 = \$t4 >> \$t6	Shift right logical variable (unsigned)

Data Transfer Instructions – general form

- MIPS can only access memory with load and store instructions
- Form: <operation> <reg1>, <constant>(<reg2>)



Example	Meaning	Comment
lw $t2,8($t3)	$t2 = Mem[$t3 + 8]	32-bit load
lh $t3,0($t4)	$t3 = Mem[$t4]	Signed 16-bit load
lhu $t8,2($t3)	$t8 = Mem[$t3 + 2]	Unsigned 16-bit load
lb $t4,0($t5)	$t4 = Mem[$t5]	Signed 8-bit load
lbu $t6,1($t9)	$t6 = Mem[$t9 + 1]	Unsigned 8-bit load
sw $t5,-4($t2)	Mem[$t2-4] = $t5	32-bit store
sh $t6,12($t3)	Mem[$t3 + 12] = $t6	16-bit store
sb $t7,1($t3)	Mem[$t3 + 1] = $t7	8-bit store

Using Data Transfer Instructions

- Consider the following source code fragment;

```
int a,b,c,d;
…
a=b+c-d;
```

Assume the addresses of a, b, c, and d are in the registers $t2, $t3, $t4, and $t5, respectively. The following MIPS assembly code performs this assignment assuming $t6 and $t7 are available.

```
lw $t6,0($t3)      # load b into $t6
lw $t7,0($t4)      # load c into $t7
add $t6,$t6,$t7   # $t6 = $t6 + $t7
lw $t7,0($t5)      # load d into $t7
sub $t6,$t6,$t7   # $t6 = $t6 - $t7
sw $t6,0($t2)      # store $t6 into
```

General Form of Jump and Branch instructions

- MIPS provides direct jumps to support unconditional transfers of control to a specified location.
- MIPS provides indirect jumps to support returns and switch statements.
- MIPS provides conditional branch instructions to support decision making. MIPS conditional branches test if the values of two registers are equal or not equal.

General Form	**Example**	**Meaning**	**Comment**
`j <label>`	`j L1`	goto L1;	Direct jump (J)
`jr <sreg>`	`jr $ra`	goto $ra;	Indirect jump (R)
`beq <s1reg>,<s2reg>,<label>`	`beq $t2, $t3, L1`	if($t2 == $t3) goto L1;	Branch equal (I)
`bne <s1reg>,<s2reg>,<label>`	`bne $t2, $t3, L1`	if($t2 != $t3) goto L1;	Branch not equal (I)

Consider the following source code:

Translate into MIPS instructions assuming the values of i, j, and k are associated with the registers $t2, $t3, and $t4, respectively.

```
if(i == j)
        k = k + i;

        bne $t2,$t3,L1          # if ($t2 != $t3) goto L1
        addu $t4,$t4,$t2        # k = k + i
        L1:
```

General form of Comparison instructions

General Form	Example	Meaning	Comment

slt <dreg>,<s1reg>,<s2reg>	slt $t2,$t3,$t4	if($t3<$t4) $t2 = 1;
		else $t2 = 0;
			Compare less than (R)

sltu <dreg>,<s1reg>,<s2reg>	sltu $t2,$t3,$t4	if($t3<$t4) $t2 = 1;
		else $t2 = 0;
			Compare less than unsigned (R)

slti <dreg>,<s1reg>,<const>	slti $t2,$t3,100	if($t3<100) $t2 = 1;
		else $t2 = 0;
			Compare less than constant (I)

sltiu <dreg>,<s1reg>,<const>	sltiu $t2,$t3,100	if($t3<100) $t2 = 1;
		else $t2 = 0;
			Compare less than constant unsigned (I)

- Consider the following source code:

Translate into MIPS instructions assuming the values of a, b, and c are associated with the registers $t2, $t3, and $t4 respectively. Assume $t5 is available.

```
if(a > b)
        c = a;

slt $t5,$t3,$t2       # b < a
beq $t5,$zero,L1      # if($t5==0)goto L1
or $t4,$t2,$zero      # c = a
L1:
```