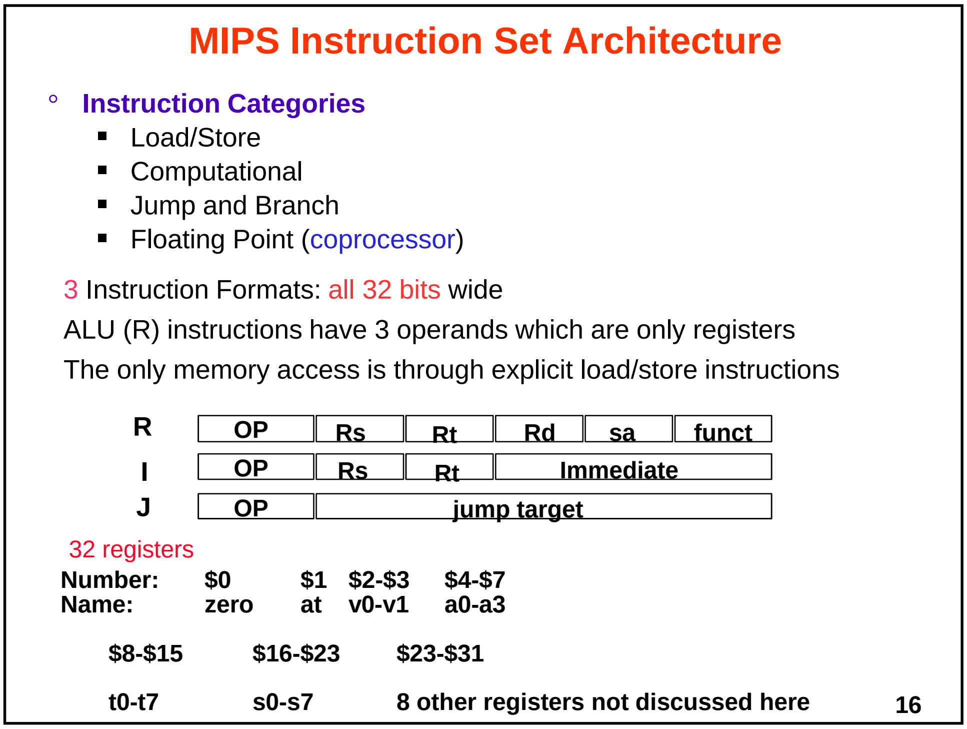


MIPS Instruction Set Architecture

- **Instruction Categories**
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point (coprocessor)

3 Instruction Formats: all 32 bits wide

ALU (R) instructions have 3 operands which are only registers

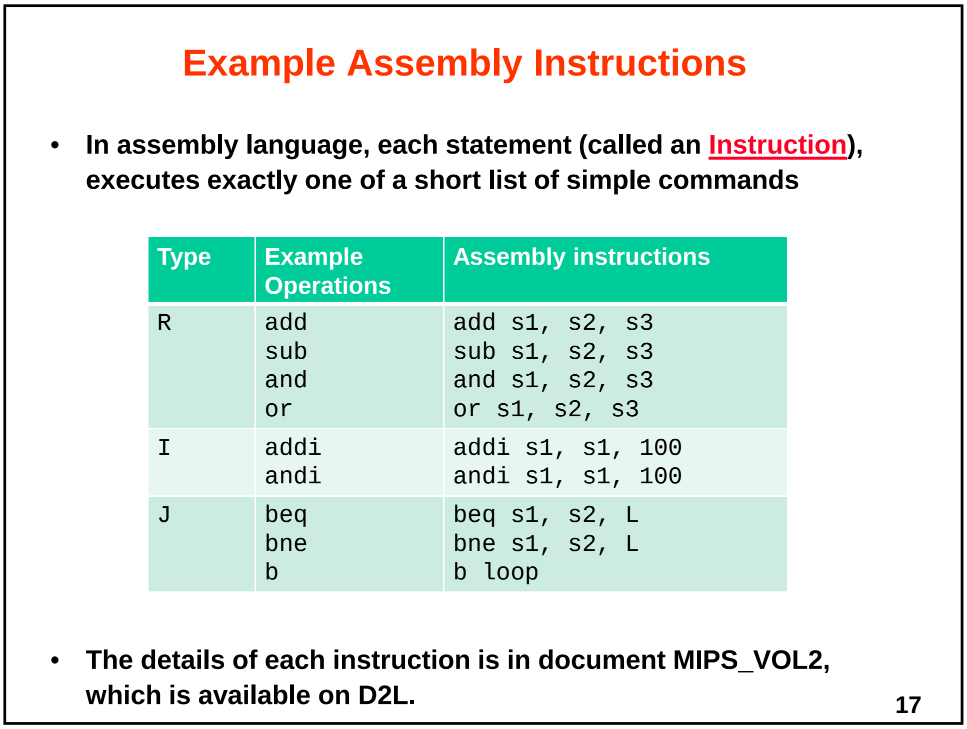
The only memory access is through explicit load/store instructions

R	**OP**	**Rs**	**Rt**	**Rd**	**sa**	**funct**
I	**OP**	**Rs**	**Rt**	**Immediate**		
J	**OP**	**jump target**				

32 registers

Number:	**$0**	**$1**	**$2-$3**	**$4-$7**
Name:	**zero**	**at**	**v0-v1**	**a0-a3**

$8-$15	**$16-$23**	**$23-$31**
t0-t7	**s0-s7**	**8 other registers not discussed here**



Example Assembly Instructions

- **In assembly language, each statement (called an Instruction), executes exactly one of a short list of simple commands**

Type	Example Operations	Assembly instructions
R	add sub and or	add s1, s2, s3 sub s1, s2, s3 and s1, s2, s3 or s1, s2, s3
I	addi andi	addi s1, s1, 100 andi s1, s1, 100
J	beq bne b	beq s1, s2, L bne s1, s2, L b loop

- **The details of each instruction is in document MIPS_VOL2, which is available on D2L.**

R-Type Arithmetic

- **Syntax of Instructions:**

  1   2, 3, 4

  where:

  1) operation by name

  2) operand getting result (destination)

  3) 1st operand for operation (source1)

  4) 2nd operand for operation (source2)

- **Syntax is rigid:**
  - 1 operator, 3 operands
  - Why? Keep Hardware simple via regularity

Arithmetic

Addition

```
    add    Rd,    Rs1,   Rs2
    addi   Rd,    Rs,    SIMM_16
    addu,  addiu
```

Subtraction

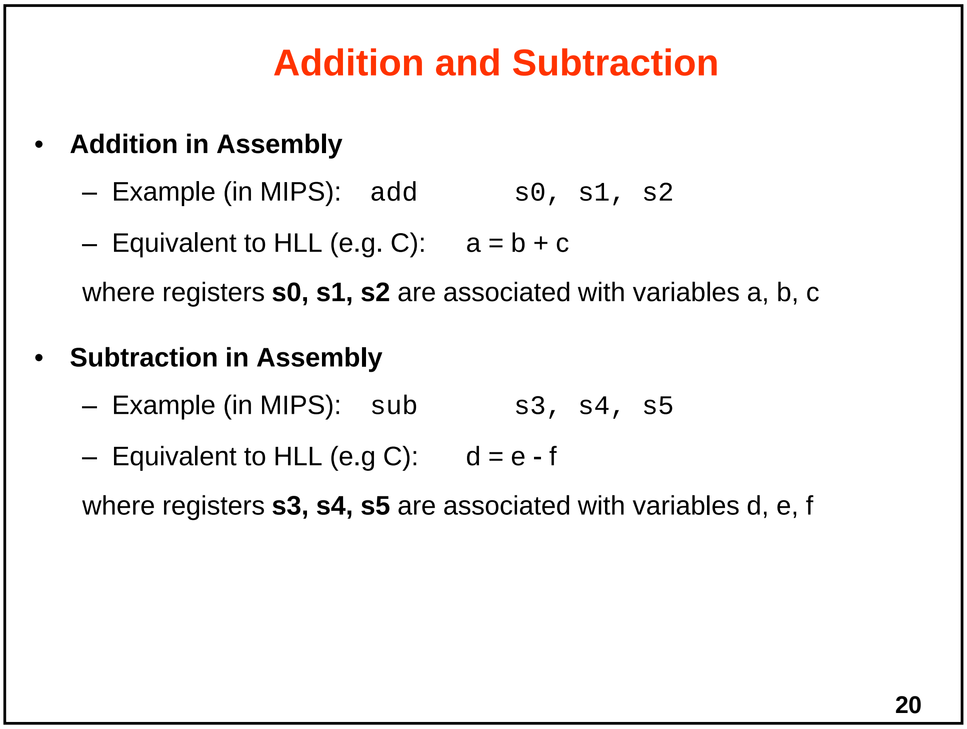
```
    sub    Rd,    Rs1,   Rs2
```

Multiplication

```
    mul    Rd,    Rs1,   Rs2
```

Division

```
    div Rs, Rt      #LO←quotient, HI← remainder
    MFHI Rd         #rd← HI
```



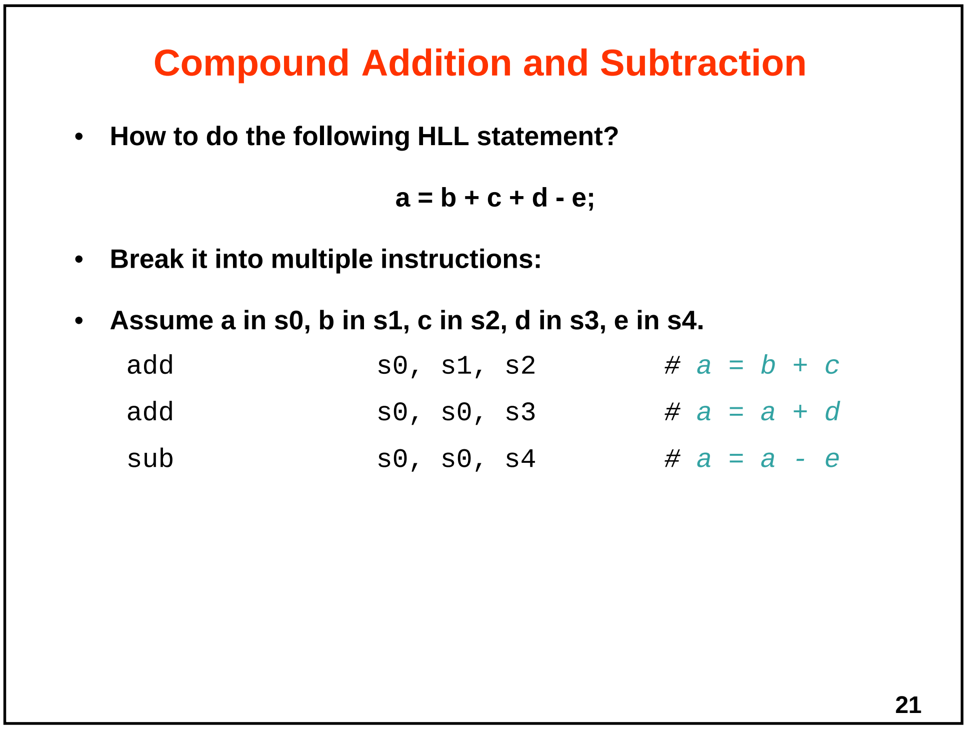
Addition and Subtraction

- **Addition in Assembly**
  - Example (in MIPS): `add s0, s1, s2`
  - Equivalent to HLL (e.g. C): a = b + c

  where registers **s0, s1, s2** are associated with variables a, b, c

- **Subtraction in Assembly**
  - Example (in MIPS): `sub s3, s4, s5`
  - Equivalent to HLL (e.g C): d = e - f

  where registers **s3, s4, s5** are associated with variables d, e, f



Compound Addition and Subtraction

- **How to do the following HLL statement?**

  a = b + c + d - e;

- **Break it into multiple instructions:**
- **Assume a in s0, b in s1, c in s2, d in s3, e in s4.**

```
add          s0, s1, s2       # a = b + c
add          s0, s0, s3       # a = a + d
sub          s0, s0, s4       # a = a - e
```

I Type Immediates

- **Immediates are numerical constants.**
- **They appear often in code, so there are special instructions for them.**
- **"Add Immediate":**

  addi s0, s1, 10 (in MIPS)

  f = g + 10 (in C)

  where registers **s0, s1** are associated with variables f, g
- **Syntax similar to add instruction, except that last argument is a number instead of a register.**

Register Zero

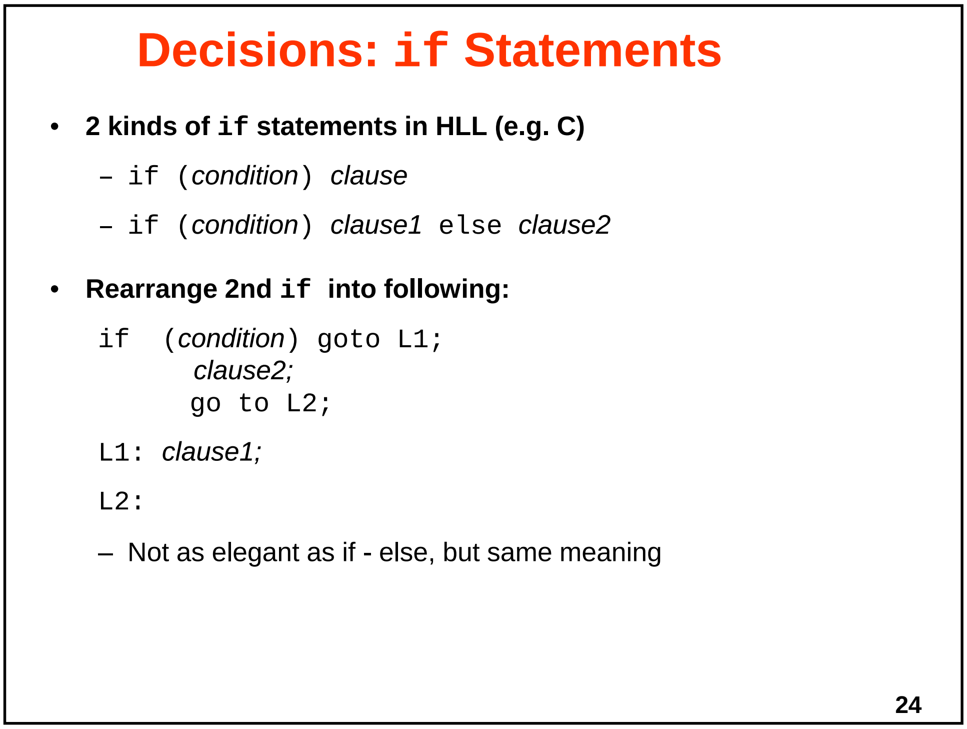
- **One particular immediate, the number zero (0), appears very often in code.**
- **So we define register zero (**$0 **or zero) to always have the value 0.**
- **This is defined in hardware, so an instruction like**

  addi $0, $0, 5

  Or

  addi zero, zero, 5

  will not do anything.
- **Use this register, it's very handy!**



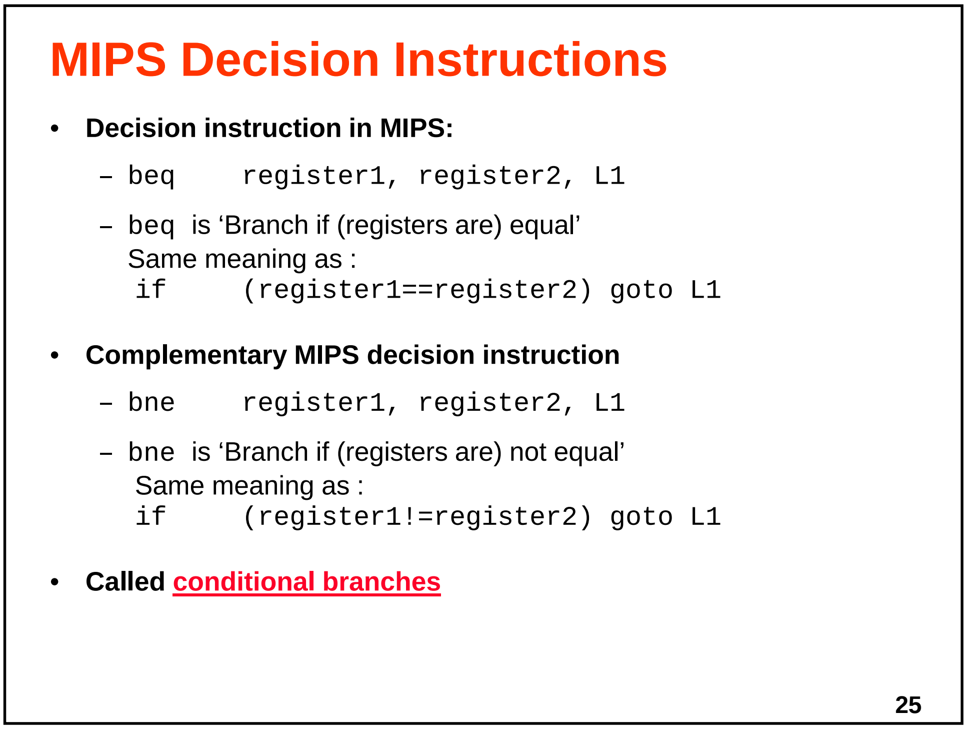
Decisions: `if` Statements

- **2 kinds of `if` statements in HLL (e.g. C)**
  - `if (`*condition*`)` *clause*
  - `if (`*condition*`)` *clause1* `else` *clause2*
- **Rearrange 2nd `if` into following:**

```
if  (condition) goto L1;
      clause2;
      go to L2;
L1: clause1;
L2:
```

  - Not as elegant as if - else, but same meaning

24



MIPS Decision Instructions

- **Decision instruction in MIPS:**
  - `beq    register1, register2, L1`
  - `beq` is ‘Branch if (registers are) equal’
    Same meaning as :
    `if    (register1==register2) goto L1`
- **Complementary MIPS decision instruction**
  - `bne    register1, register2, L1`
  - `bne` is ‘Branch if (registers are) not equal’
    Same meaning as :
    `if    (register1!=register2) goto L1`
- **Called conditional branches**

25

MIPS Goto Instruction

- In addition to conditional branches, MIPS has an unconditional branch:

```
b label
```

- Called a Jump Instruction: jump (or branch) directly to the given label without needing to satisfy any condition

- Same meaning as :

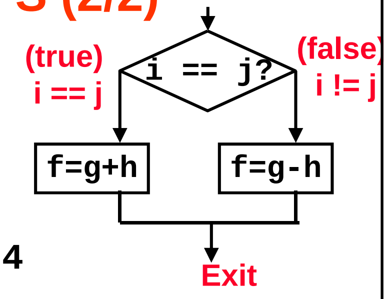
```
goto label
```

- Technically, it's the same as:

```
beq   $0, $0, label
```

since it always satisfies the condition.

Compiling `if` into MIPS (2/2)



- Compile by hand

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

- Use this mapping:

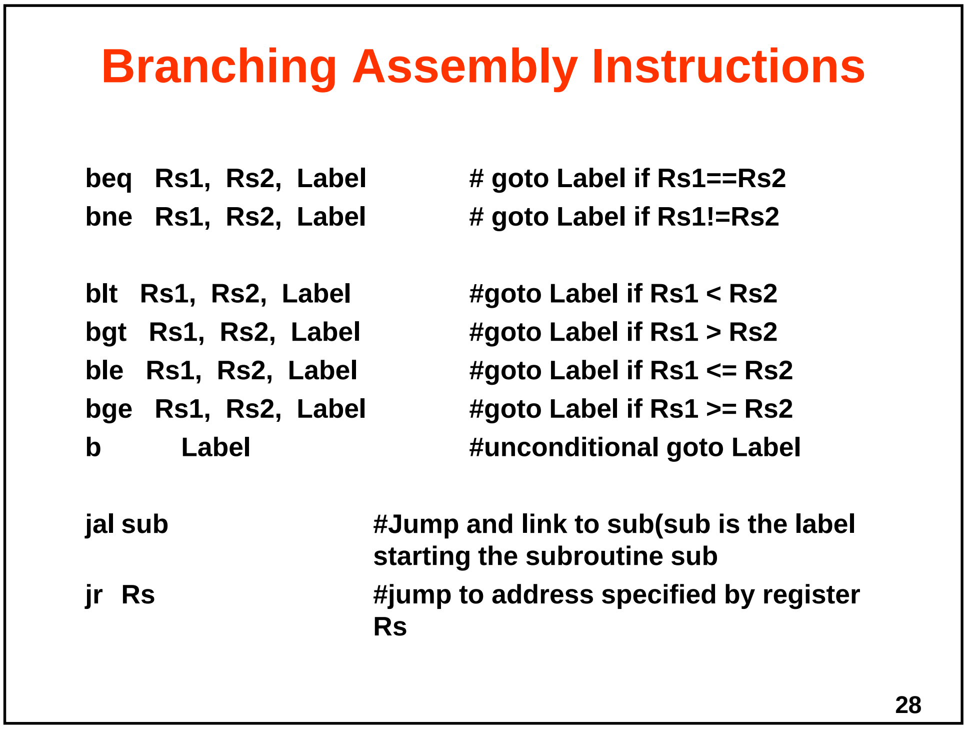
f: s0, g: s1,   h: s2,   i: s3, j: s4

°

°**Final compiled MIPS code:**

```
        beq s3, s4, True  # branch i==j

        sub s0, s1, s2    # f=g-h(false)

        b      Fin           # go to Fin

    True:
        add    s0,s1,s2   # f=g+h (true)

    Fin:
```

° **Note: Compilers automatically create labels to handle decisions (branches) appropriately. Generally not found in HLL code.**



Branching Assembly Instructions

```
beq   Rs1,  Rs2,  Label          # goto Label if Rs1==Rs2
bne   Rs1,  Rs2,  Label          # goto Label if Rs1!=Rs2

blt   Rs1,  Rs2,  Label          #goto Label if Rs1 < Rs2
bgt   Rs1,  Rs2,  Label          #goto Label if Rs1 > Rs2
ble   Rs1,  Rs2,  Label          #goto Label if Rs1 <= Rs2
bge   Rs1,  Rs2,  Label          #goto Label if Rs1 >= Rs2
b          Label                 #unconditional goto Label

jal sub                  #Jump and link to sub(sub is the label
                         starting the subroutine sub
jr   Rs                  #jump to address specified by register
                         Rs
```