

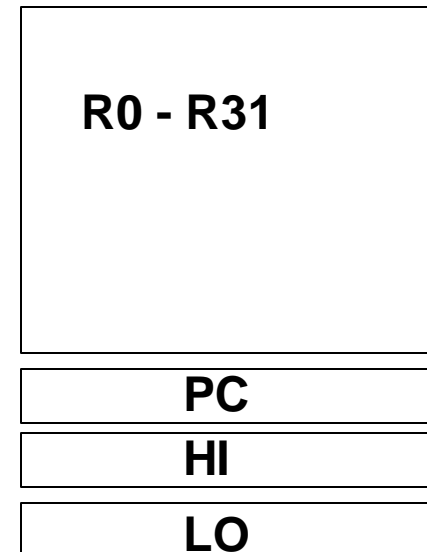
# Reduced Instruction Set Computer (RISC)

- Focuses on reducing the number and complexity of instructions of the machine.
- Reduced number of cycles needed per instruction.
  - Goal: At least one instruction completed per clock cycle.
- Designed with CPU instruction pipelining in mind.
- Fixed-length instruction encoding.
- Load-Store: Only load and store instructions access memory.
- Simplified addressing modes.
  - Usually limited to immediate, register indirect, register displacement, indexed.
- Delayed loads and branches.
- Prefetch and speculative execution.
- Examples: MIPS, UltraSpark, Alpha, PowerPC.

# RISC Instruction Set Architecture Example: MIPS R3000

- **Memory:** Can address  $2^{32}$  bytes or  $2^{30}$  words (32-bits).
- **Instruction Categories:**
  - Load/Store.
  - Computational: ALU.
  - Jump and Branch.
  - Floating Point.
    - coprocessor
  - Memory Management.
  - Special.

Registers



31  
GPRs  
R0 = 0

- **3 Instruction Formats:** all 32 bits wide:

R-Type



I-Type: ALU  
Load/Store, Branch



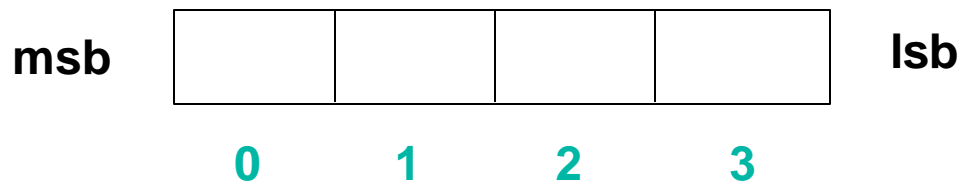
J-Type: Jumps



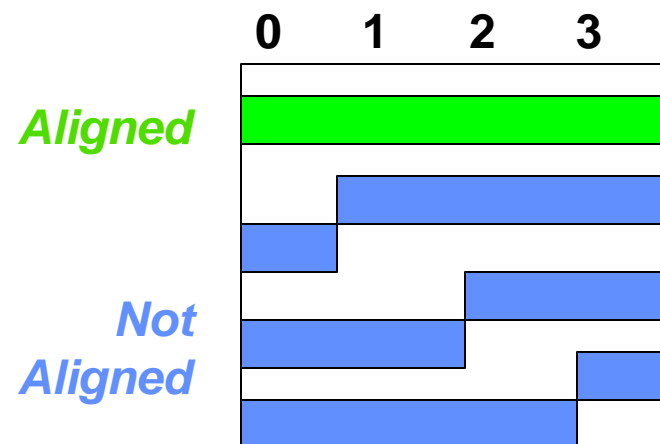
**EECC550 - Shaaban**

# MIPS Memory Addressing & Alignment

- MIPS uses Big Endian operand storage in memory where the most significant byte is in low memory (this is similar to IBM 360/370, Motorola 68k, Sparc, HP PA).



- MIPS requires that all words (32- bits) to start at memory addresses that are multiple of 4
- In general objects must fall on memory addresses that are multiple of their size.



# MIPS Register Usage/Naming Conventions

- In addition to the usual naming of registers by \$ followed with register number, registers are also named according to MIPS register usage convention as follows:

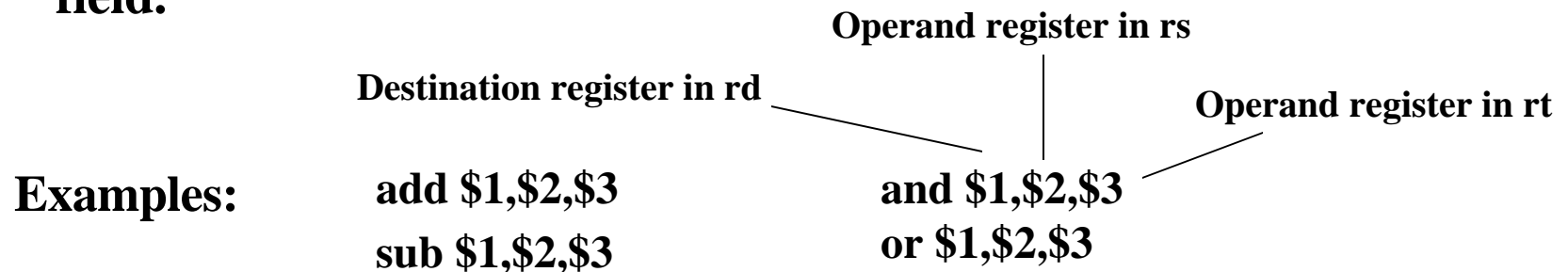
Register Number	Name	Usage	Preserved on call?
0	\$zero	Constant value 0	n.a.
1	\$at	Reserved for assembler	no
2-3	\$v0-\$v1	Values for result and expression evaluation	no
4-7	\$a0-\$a3	Arguments	yes
8-15	\$t0-\$t7	Temporaries	no
16-23	\$s0-\$s7	Saved	yes
24-25	\$t8-\$t9	More temporaries	no
26-27	\$k0-\$k1	Reserved for operating system	yes
28	\$gp	Global pointer	yes
29	\$sp	Stack pointer	yes
30	\$fp	Frame pointer	yes
31	\$ra	Return address	yes

# MIPS R-Type (ALU) Instruction Fields

R-Type: All ALU instructions that use three registers

OP	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- **op:** Opcode, basic operation of the instruction.
  - For R-Type  $op = 0$
- **rs:** The first register source operand.
- **rt:** The second register source operand.
- **rd:** The register destination operand.
- **shamt:** Shift amount used in constant shift operations.
- **funct:** Function, selects the specific variant of operation in the op field.



# MIPS ALU I-Type Instruction Fields

I-Type ALU instructions that use two registers and an immediate value  
Loads/stores, conditional branches.

OP	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

- **op:** Opcode, operation of the instruction.
- **rs:** The register source operand.
- **rt:** The result destination register.
- **immediate:** Constant second operand for ALU instruction.

**Examples:**

add immediate:	addi \$1,\$2,100	Source operand register in rs
and immediate	andi \$1,\$2,10	Constant operand in immediate

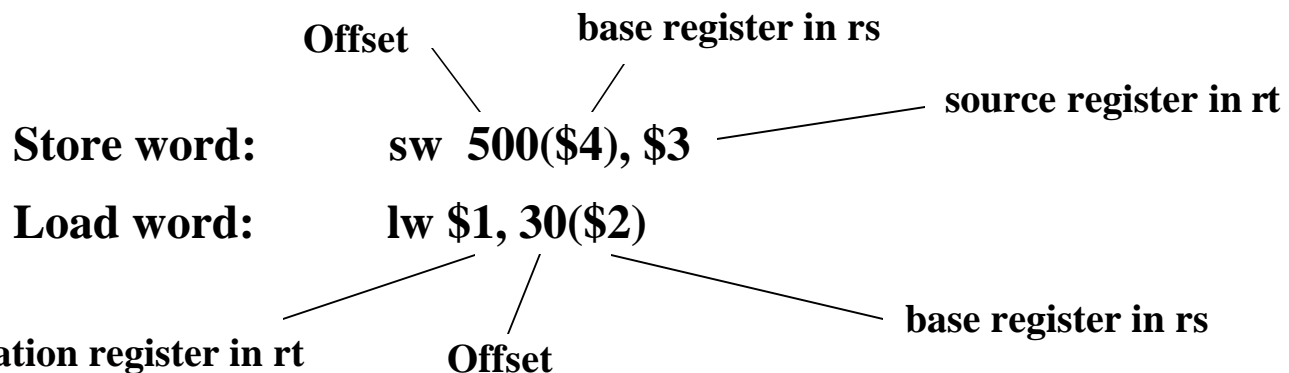
Result register in rt

# MIPS Load/Store I-Type Instruction Fields

OP	rs	rt	address
6 bits	5 bits	5 bits	16 bits

- **op: Opcode, operation of the instruction.**
  - For load op = 35, for store op = 43.
- **rs: The register containing memory base address.**
- **rt: For loads, the destination register. For stores, the source register of value to be stored.**
- **address: 16-bit memory address offset in bytes added to base register.**

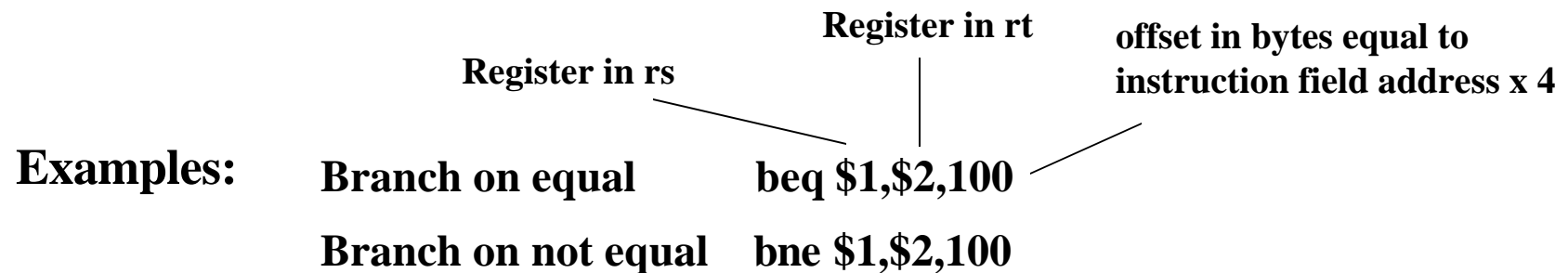
**Examples:**



# MIPS Branch I-Type Instruction Fields

<b>OP</b>	<b>rs</b>	<b>rt</b>	<b>address</b>
6 bits	5 bits	5 bits	16 bits

- **op: Opcode, operation of the instruction.**
- **rs: The first register being compared**
- **rt: The second register being compared.**
- **address: 16-bit memory address branch target offset in words added to PC to form branch address.**





# MIPS J-Type Instruction Fields

J-Type: Include jump j, jump and link jal



6 bits

26 bits

- **op: Opcode, operation of the instruction.**
  - Jump j op = 2
  - Jump and link jal op = 3
- **jump target: jump memory address in words.**

Jump memory address in bytes equal to instruction field jump target x 4

**Examples:**

**Jump**

**j 10000**

**Jump and link**

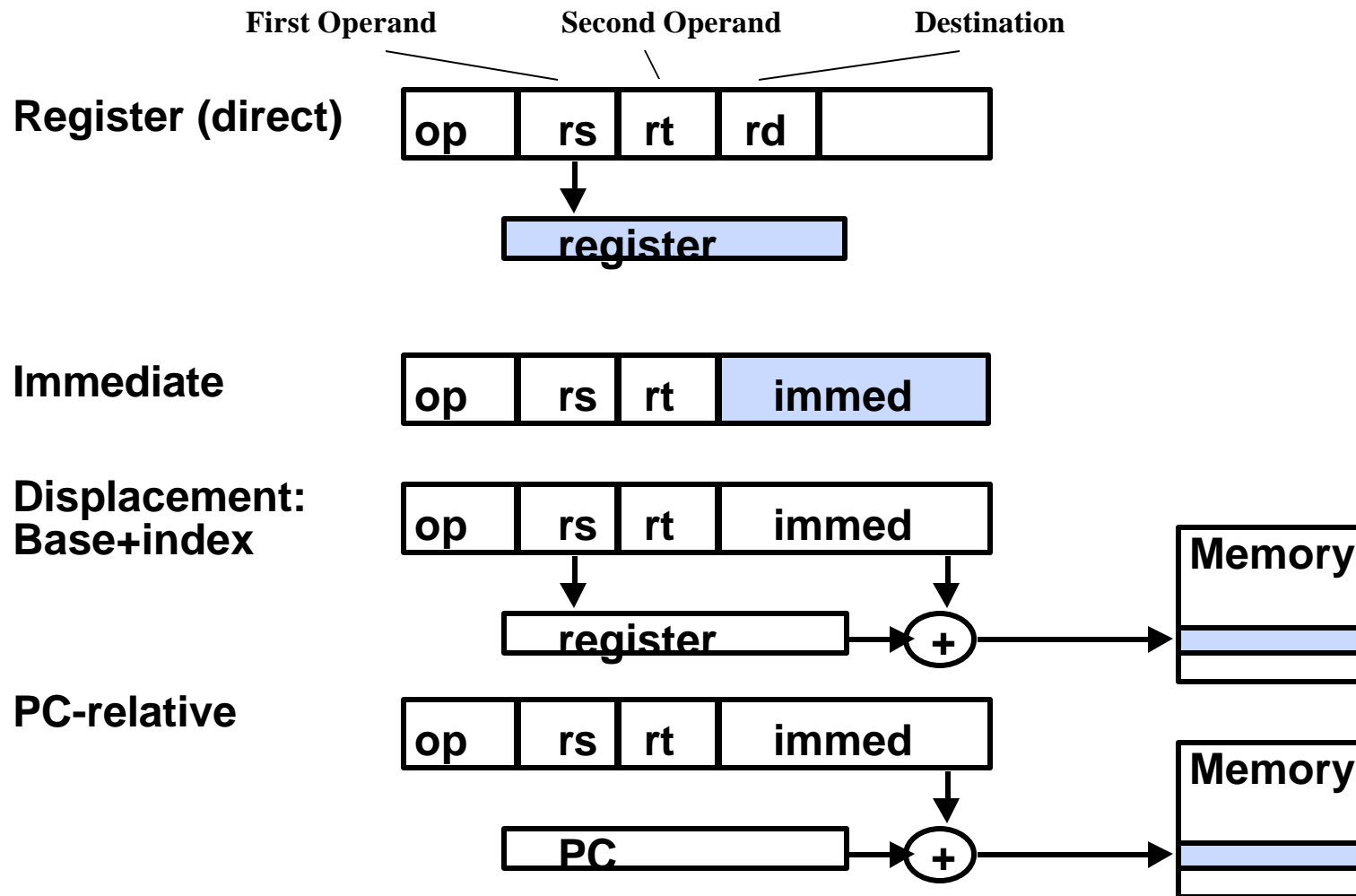
**jal 10000**

PC(31-28)



# MIPS Addressing Modes/Instruction Formats

- All instructions 32 bits wide



# MIPS Arithmetic Instructions Examples

<u>Instruction</u>	<u>Example</u>	<u>Meaning</u>	<u>Comments</u>
add	add \$1,\$2,\$3	$\$1 = \$2 + \$3$	3 operands; <a href="#">exception possible</a>
subtract	sub \$1,\$2,\$3	$\$1 = \$2 - \$3$	3 operands; <a href="#">exception possible</a>
add immediate	addi \$1,\$2,100	$\$1 = \$2 + 100$	+ constant; <a href="#">exception possible</a>
add unsigned	addu \$1,\$2,\$3	$\$1 = \$2 + \$3$	3 operands; <a href="#">no exceptions</a>
subtract unsigned	subu \$1,\$2,\$3	$\$1 = \$2 - \$3$	3 operands; <a href="#">no exceptions</a>
add imm. unsign.	addiu \$1,\$2,100	$\$1 = \$2 + 100$	+ constant; <a href="#">no exceptions</a>
multiply	mult \$2,\$3	Hi, Lo = $\$2 \times \$3$	64-bit signed product
multiply unsigned	multu \$2,\$3	Hi, Lo = $\$2 \times \$3$	64-bit unsigned product
divide	div \$2,\$3	Lo = $\$2 \div \$3$ , Hi = $\$2 \bmod \$3$	Lo = quotient, Hi = remainder
divide unsigned	divu \$2,\$3	Lo = $\$2 \div \$3$ , Hi = $\$2 \bmod \$3$	Unsigned quotient & remainder
Move from Hi	mfhi \$1	$\$1 = \text{Hi}$	Used to get copy of Hi
Move from Lo	mflo \$1	$\$1 = \text{Lo}$	Used to get copy of Lo

# MIPS Logic/Shift Instructions Examples

<u>Instruction</u>	<u>Example</u>	<u>Meaning</u>	<u>Comment</u>
and	and \$1,\$2,\$3	$\$1 = \$2 \& \$3$	3 reg. operands; Logical AND
or	or \$1,\$2,\$3	$\$1 = \$2   \$3$	3 reg. operands; Logical OR
xor	xor \$1,\$2,\$3	$\$1 = \$2 \oplus \$3$	3 reg. operands; Logical XOR
nor	nor \$1,\$2,\$3	$\$1 = \sim(\$2   \$3)$	3 reg. operands; Logical NOR
and immediate	andi \$1,\$2,10	$\$1 = \$2 \& 10$	Logical AND reg, constant
or immediate	ori \$1,\$2,10	$\$1 = \$2   10$	Logical OR reg, constant
xor immediate	xori \$1, \$2,10	$\$1 = \sim\$2 \& \sim 10$	Logical XOR reg, constant
shift left logical	sll \$1,\$2,10	$\$1 = \$2 \ll 10$	Shift left by constant
shift right logical	srl \$1,\$2,10	$\$1 = \$2 \gg 10$	Shift right by constant
shift right arithm.	sra \$1,\$2,10	$\$1 = \$2 \gg 10$	Shift right (sign extend)
shift left logical	sllv \$1,\$2,\$3	$\$1 = \$2 \ll \$3$	Shift left by variable
shift right logical	srlv \$1,\$2, \$3	$\$1 = \$2 \gg \$3$	Shift right by variable
shift right arithm.	srav \$1,\$2, \$3	$\$1 = \$2 \gg \$3$	Shift right arith. by variable

# MIPS data transfer instructions Examples

## Instruction

## Comment

sw 500(\$4), \$3

Store word

sh 502(\$2), \$3

Store half

sb 41(\$3), \$2

Store byte

lw \$1, 30(\$2)

Load word

lh \$1, 40(\$3)

Load halfword

lhu \$1, 40(\$3)

Load halfword unsigned

lb \$1, 40(\$3)

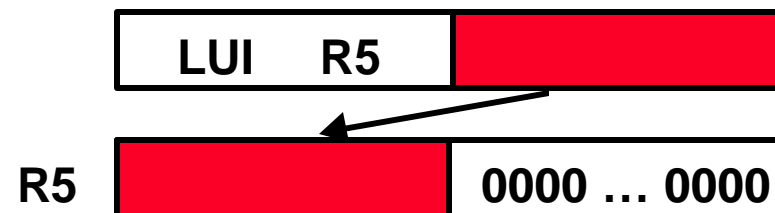
Load byte

lbu \$1, 40(\$3)

Load byte unsigned

lui \$1, 40

Load Upper Immediate (16 bits shifted left by 16)



# MIPS Branch, Compare, Jump Instructions Examples

<u>Instruction</u>	<u>Example</u>	<u>Meaning</u>
branch on equal	beq \$1,\$2,100	if ( $\$1 == \$2$ ) go to PC+4+100 <i>Equal test; PC relative branch</i>
branch on not eq.	bne \$1,\$2,100	if ( $\$1 \neq \$2$ ) go to PC+4+100 <i>Not equal test; PC relative branch</i>
set on less than	slt \$1,\$2,\$3	if ( $\$2 < \$3$ ) $\$1=1$ ; else $\$1=0$ <i>Compare less than; 2's comp.</i>
set less than imm.	slti \$1,\$2,100	if ( $\$2 < 100$ ) $\$1=1$ ; else $\$1=0$ <i>Compare &lt; constant; 2's comp.</i>
set less than uns.	sltu \$1,\$2,\$3	if ( $\$2 < \$3$ ) $\$1=1$ ; else $\$1=0$ <i>Compare less than; natural numbers</i>
set l. t. imm. uns.	sltiu \$1,\$2,100	if ( $\$2 < 100$ ) $\$1=1$ ; else $\$1=0$ <i>Compare &lt; constant; natural numbers</i>
jump	j 10000	go to 10000 <i>Jump to target address</i>
jump register	jr \$31	go to \$31 <i>For switch, procedure return</i>
jump and link	jal 10000	$\$31 = PC + 4$ ; go to 10000 <i>For procedure call</i>

# Details of The MIPS Instruction Set

- Register zero **always** has the value **zero** (even if you try to write it).
- Branch/jump **and link** put the return addr. PC+4 into the link register (R31).
- All instructions change **all 32 bits** of the destination register (including lui, lb, lh) and all read all 32 bits of sources (add, sub, and, or, ...)
- Immediate arithmetic and logical instructions are extended as follows:
  - logical immediates ops are zero extended to 32 bits.
  - arithmetic immediates ops are sign extended to 32 bits (including addu).
- The data loaded by the instructions lb and lh are extended as follows:
  - lbu, lhu are zero extended.
  - lb, lh are sign extended.
- Overflow can occur in these arithmetic and logical instructions:
  - add, sub, addi
  - it **cannot** occur in addu, subu, addiu, and, or, xor, nor, shifts, mult, multu, div, divu

# Example: C Assignment To MIPS

- Given the C assignment statement:

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

- Assuming the variables are assigned to MIPS registers as follows:

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

- MIPS Instructions:

```
add $s0,$s1,$s2 # $s0 = g+h
```

```
add $t1,$s3,$s4 # $t1 = i+j
```

```
sub $s0,$s0,$t1 # f = (g+h)-(i+j)
```



# Example: C Assignment With Operand In Memory To MIPS

- For the C statement:

```
g = h + A[8];
```

- Assume the following MIPS register mapping:

```
g: $s1, h: $s2, base address of A[ ]: $s3
```

- Steps:

- Add **32** bytes to `$s3` to select `A[8]`, put into `$t0`
- Next add it to `h` and place in `g`

- MIPS Instructions:

```
lw $t0, 32($s3) # $t0 gets A[8]
```

```
add $s1, $s2, $t0 # $s1 = h + A[8]
```

## Example: C Assignment With Variable Index To MIPS

- For the C statement with a variable array index:

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

- Assume:  $g: \$s1$ ,  $h: \$s2$ ,  $i: \$s4$ , base address of  $A[ ]: \$s3$

- Steps:

- Turn index  $i$  to a byte offset by multiplying by four or by addition as done here:  $i + i = 2i$ ,  $2i + 2i = 4i$
- Next add  $4i$  to base address of  $A$
- Load  $A[i]$  into a temporary register.
- Finally add to  $h$  and put sum in  $g$

- MIPS Instructions:

```
add $t1,$s4,$s4      # $t1 = 2*i
add $t1,$t1,$t1      # $t1 = 4*i
add $t1,$t1,$s3      # $t1 = address of A[i]
lw  $t0,0($t1)       # $t0 = A[i]
add $s1,$s2,$t0      # g = h + A[i]
```

# Example: C If Statement to MIPS

- For The C statement:

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

- Assume the following MIPS register mapping:

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

- Mips Instructions:

```
beq $s3,$s4, True      # branch if i==j  
sub $s0,$s1,$s2        # f = g-h (false)  
j Exit                 # go to Exit  
True: add $s0,$s1,$s2  # f = g+h (true)  
Exit:
```

# Example: Simple C Loop to MIPS

- Simple loop in C:

```
Loop:   g = g + A[i];  
        i = i + j;  
        if (i != h) goto Loop;
```

- Assume MIPS register mapping:

g: \$s1, h: \$s2, i: \$s3, j: \$s4, base of A[ ]: \$s5

- MIPS Instructions:

```
Loop:   add $t1,$s3,$s3      # $t1= 2*i  
        add $t1,$t1,$t1     # $t1= 4*i  
        add $t1,$t1,$s5     # $t1=address of A[I]  
        lw  $t1,0($t1)      # $t1= A[i]  
        add $s1,$s1,$t1     # g = g + A[i]  
        add $s3,$s3,$s4     # I = i + j  
        bne $s3,$s2,Loop    # goto Loop if i!=h
```

# Example: C Less Than Test to MIPS

- Given the C statement:

```
if (g < h) go to Less
```

- Assume MIPS register mapping:

```
g: $s0, h: $s1
```

- MIPS Instructions:

```
slt $t0,$s0,$s1      # $t0 = 1 if
                     # $s0<$s1 (g < h)
bne $t0,$zero, Less  # goto Less
                     # if $t0 != 0
                     # (if (g < h)
                     . . .
```

Less:

# Example: While C Loop to MIPS

- While loop in C:

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

- Assume MIPS register mapping:

i: \$s3, j: \$s4, k: \$s5, base of save[ ]: \$s6

- MIPS Instructions:

```
Loop:  add $t1,$s3,$s3      # $t1 = 2*i
        add $t1,$t1,$t1    # $t1 = 4*i
        add $t1,$t1,$s6    # $t1 = Address
        lw  $t1,0($t1)     # $t1 = save[i]
        bne $t1,$s5,Exit   # goto Exit
                                # if save[i]!=k
        add $s3,$s3,$s4    # i = i + j
        j   Loop           # goto Loop
```

Exit:

# Example: C Case Statement To MIPS

- The following is a C case statement called switch:

```
switch (k) {  
    case 0: f=i+j; break; /* k=0*/  
    case 1: f=g+h; break; /* k=1*/  
    case 2: f=g-h; break; /* k=2*/  
    case 3: f=i-j; break; /* k=3*/  
}
```

- Assume MIPS register mapping:

**f: \$s0, g: \$s1, h: \$s2, i: \$s3, j: \$s4, k: \$s5**

- Method: Use k to index a jump address table in memory, and then jump via the value loaded.
- Steps:
  - 1st test that k matches one of the cases ( $0 \leq k \leq 3$ ); if not, the code exits.
  - Multiply k by 4 to index table of words.
  - Assume 4 sequential words in memory, base address in \$t2, have addresses corresponding to labels L0, L1, L2, L3.
  - Load a register \$t1 with jump table entry address.
  - Jump to address in register \$t1 using jump register jr \$t1.

## Example: C Case Statement To MIPS (Continued)

### MIPS Instructions:

```
    slti $t3,$s5,0           # Test if k < 0
    bne $t3,$zero,Exit      # if k < 0, goto Exit
    slti $t3,$s5,4         # Test if k < 4
    beq $t3,$zero,Exit     # if k >= 4, goto Exit
    add $t1,$s5,$s5        # Temp reg $t1 = 2*k
    add $t1,$t1,$t1        # Temp reg $t1 = 4*k
    add $t1,$t1,$t2        # $t1 = addr JumpTable[k]
    lw $t1,0($t1)         # $t1 = JumpTable[k]
    jr $t1                 # jump based on $t1
L0:  add $s0,$s3,$s4      # k=0 so f = i + j
     j  Exit              # end case, goto Exit
L1:  add $s0,$s1,$s2      # k=1 so f = g + h
     j  Exit              # end case, goto Exit
L2:  sub $s0,$s1,$s2      # k=2 so f = g - h
     j  Exit              # end case, goto Exit
L3:  sub $s0,$s3,$s4      # k=3 so f = i - j
Exit:                               # end of switch statement
```



# Example: Single Procedure Call In MIPS

- C Code:

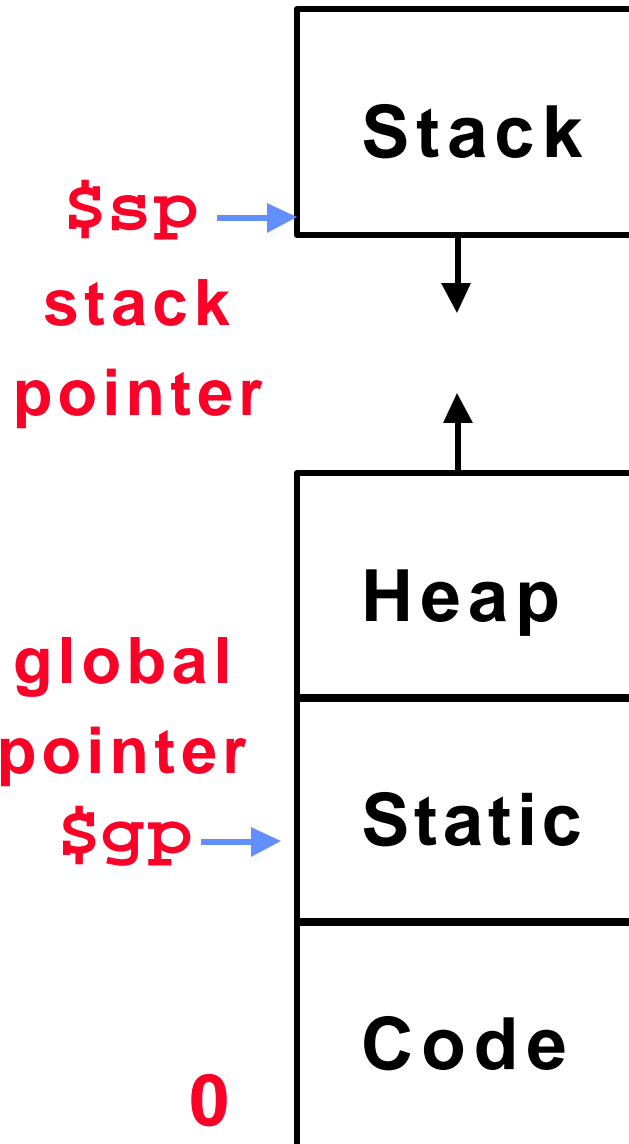
```
... sum(a,b);... /* a,b:a: $s0, b: $s1 */
}
...
int sum(int x, int y) {
    return x+y;
}
```

- MIPS Instructions:

address

```
1000      add    $a0,$s0,$zero    # x = a
1004      add    $a1,$s1,$zero    # y = b
1008      jal    sum             # $ra=1012,go to sum
1012      ...
2000 sum:  add    $v0,$a0,$a1
2004      jr    $ra
```

# C Memory Allocation Seen By MIPS Programs



**Space for saved  
procedure information**

**Explicitly created space,  
e.g., malloc( ); C pointers**

**Variables declared  
once per program**

**Program**

# Example: Nested Procedure Call In MIPS

- C Code:

```
int sumSquare(int x, int y) {  
    return mult(x,x)+ y;  
}
```

- MIPS Code:

sumSquare:

```
    subi $sp,$sp,12           # space on stack  
    sw   $ra,$ 8($sp)        # save return address  
    sw   $a0,$ 0($sp)        # save x  
    sw   $a1,$ 4($sp)        # save y  
    addi $a1,$a0,$zero        # mult(x,x)  
    jal  mult                 # call mult  
    lw   $ra,$ 8($sp)        # get return address  
    lw   $a0,$ 0($sp)        # restore x  
    lw   $a1,$ 4($sp)        # restore y  
    add  $v0,$v0,$a1         # mult()+ y  
    addi $sp,$sp,12          # => stack space  
    jr   $ra
```