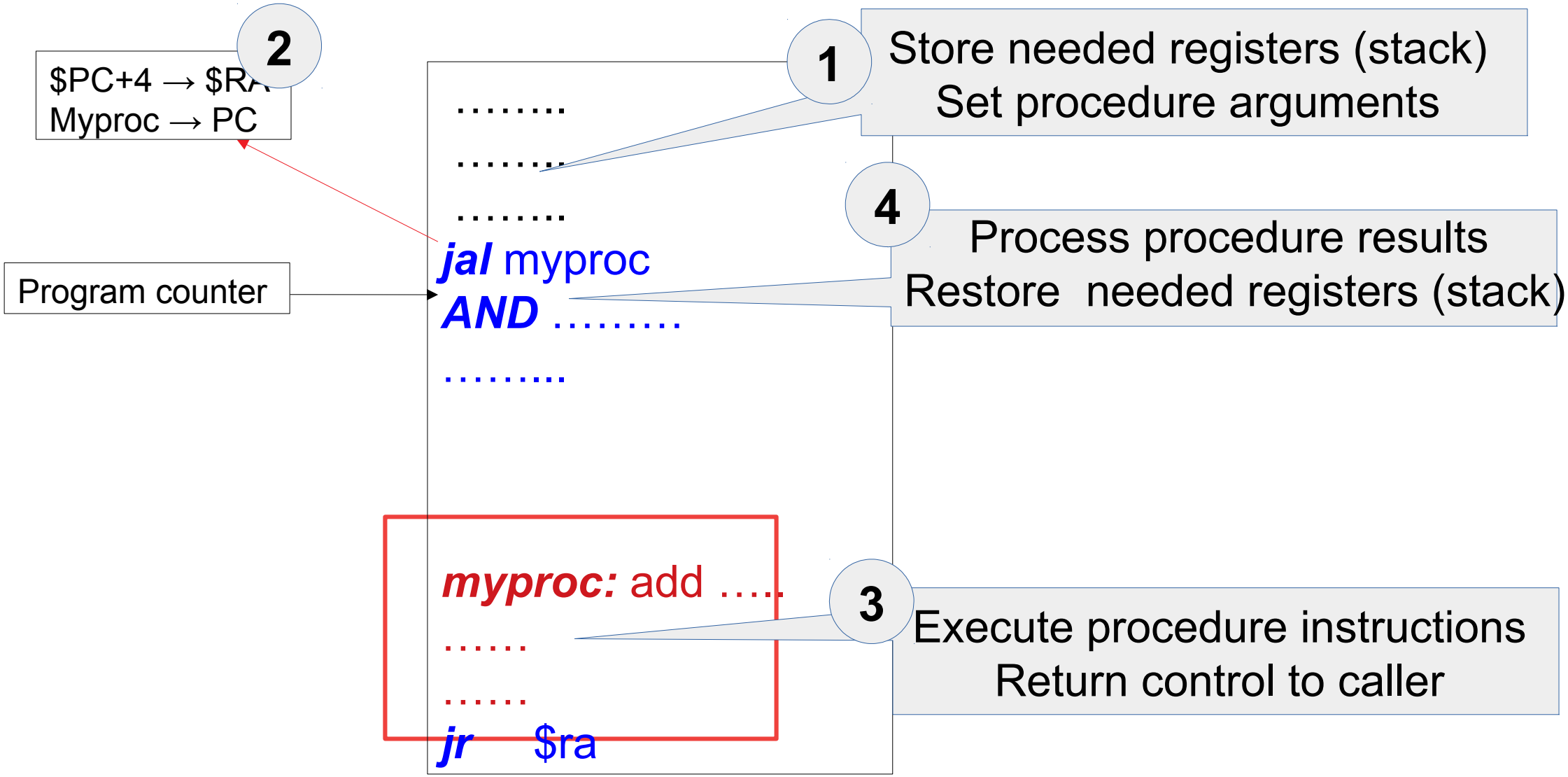
Instruction Set Architecture

Dr. Ahmed Zahran

WGB 182

a.zahran@cs.ucc.ie

Procedure Calling



Leaf Procedure and Stack Example

- Leaf procedure: A procedure that does not call another procedure

C code:

```
int leaf_example (int g, h, i, j)
{ int f;
  f = (g + h) - (i + j);
  return f;
}
```

- Arguments g, …, j in $a0, …, $a3
- Assume f in $s0 (hence, need to save $s0 on stack)
- Result in $v0

- MIPS code:

```
leaf_example:
  addi $sp, $sp, -4
  sw   $s0, 0($sp)
  add  $t0, $a0, $a1
  add  $t1, $a2, $a3
  sub  $s0, $t0, $t1
  add  $v0, $s0, $zero
  lw   $s0, 0($sp)
  addi $sp, $sp, 4
  jr   $ra
```

Save $s0 on stack

Procedure body

Result

Restore $s0

Return

Non-Leaf Procedures

- Procedures that call other procedures
- For nested call, caller needs to save on the stack:
  - Its return address
  - Any arguments and temporaries needed after the call
- Restore from the stack after the call

Preserved	Not preserved
Saved registers: `$s0–$s7`	Temporary registers: `$t0–$t9`
Stack pointer register: `$sp`	Argument registers: `$a0–$a3`
Return address register: `$ra`	Return value registers: `$v0–$v1`
Stack above the stack pointer	Stack below the stack pointer

Non-Leaf Procedure Example

- C code:

```
int fact (int n)
{
  if (n < 1)
return f;
   else return
n * fact(n - 1);
 }
```

  - Argument n in $a0
  - Result in $v0

- MIPS code:

```
fact:
   addi $sp, $sp, -8     # adjust stack for 2 items
   sw   $ra, 4($sp)      # save return address
   sw   $a0, 0($sp)      # save argument
   slti $t0, $a0, 1      # test for n < 1
   beq  $t0, $zero, L1
   addi $v0, $zero, 1    # if so, result is 1
   addi $sp, $sp, 8      #   pop 2 items from stack
   jr   $ra              #   and return
L1: addi $a0, $a0, -1    # else decrement n
   jal  fact             # recursive call
   lw   $a0, 0($sp)      # restore original n
   lw   $ra, 4($sp)      #   and return address
   addi $sp, $sp, 8      # pop 2 items from stack
   mul  $v0, $a0, $v0    # multiply to get result
   jr   $ra              # and return
```

Character Data

- Byte-encoded character sets
  - ASCII: 128 characters
  - Latin-1: 256 characters
    - ASCII, +96 more graphic characters
- Unicode: 32-bit character set
  - Used in Java, C++ wide characters, …
  - Most of the world’s alphabets, plus symbols
  - UTF-8, UTF-16: ***variable-length encodings***
- MIPS offers load byte (*lb*), store byte *(sb),* Load half (*lh*), and Store half (*sh*)
  - MIPS also offers unsigned version of these instructions

String Copy Example

- C code:
  - Null-terminated string

```
void strcpy (char x[],
char y[])
{ int i;
  i = 0;
  while ((x[i]=y[i])!='\0')
    i += 1;
}
```

  - Addresses of x, y in $a0, $a1
  - i in $s0

- MIPS code:

```
strcpy:
   addi $sp, $sp, -4      # adjust stack for 1 item
   sw   $s0, 0($sp)       # save $s0
   add  $s0, $zero, $zero # i = 0
L1: add  $t1, $s0, $a1     # addr of y[i] in $t1
   lbu  $t2, 0($t1)       # $t2 = y[i]
   add  $t3, $s0, $a0     # addr of x[i] in $t3
   sb   $t2, 0($t3)       # x[i] = y[i]
   beq  $t2, $zero, L2    # exit loop if y[i] == 0
   addi $s0, $s0, 1       # i = i + 1
   j    L1                # next iteration of loop
L2: lw   $s0, 0($sp)       # restore saved $s0
   addi $sp, $sp, 4       # pop 1 item from stack
   jr   $ra               # and return
```

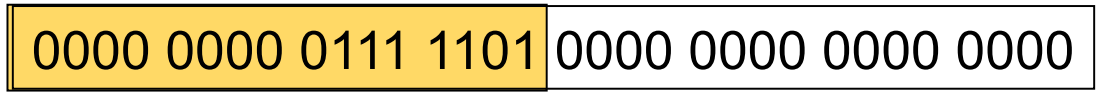
Supporting Large Constants

- Most constants are small
  - 16-bit immediate is sufficient (make common case fast)
- For the occasional 32-bit constant

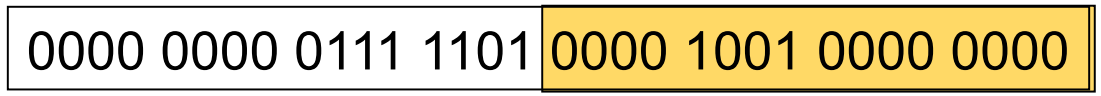
  lui rt, constant
  - Copies 16-bit constant to left 16 bits of rt
  - Clears right 16 bits of rt to 0
- $at (register 1): assembler temporary

Example: 0000 0000 0111 1101 0000 1001 0000 0000

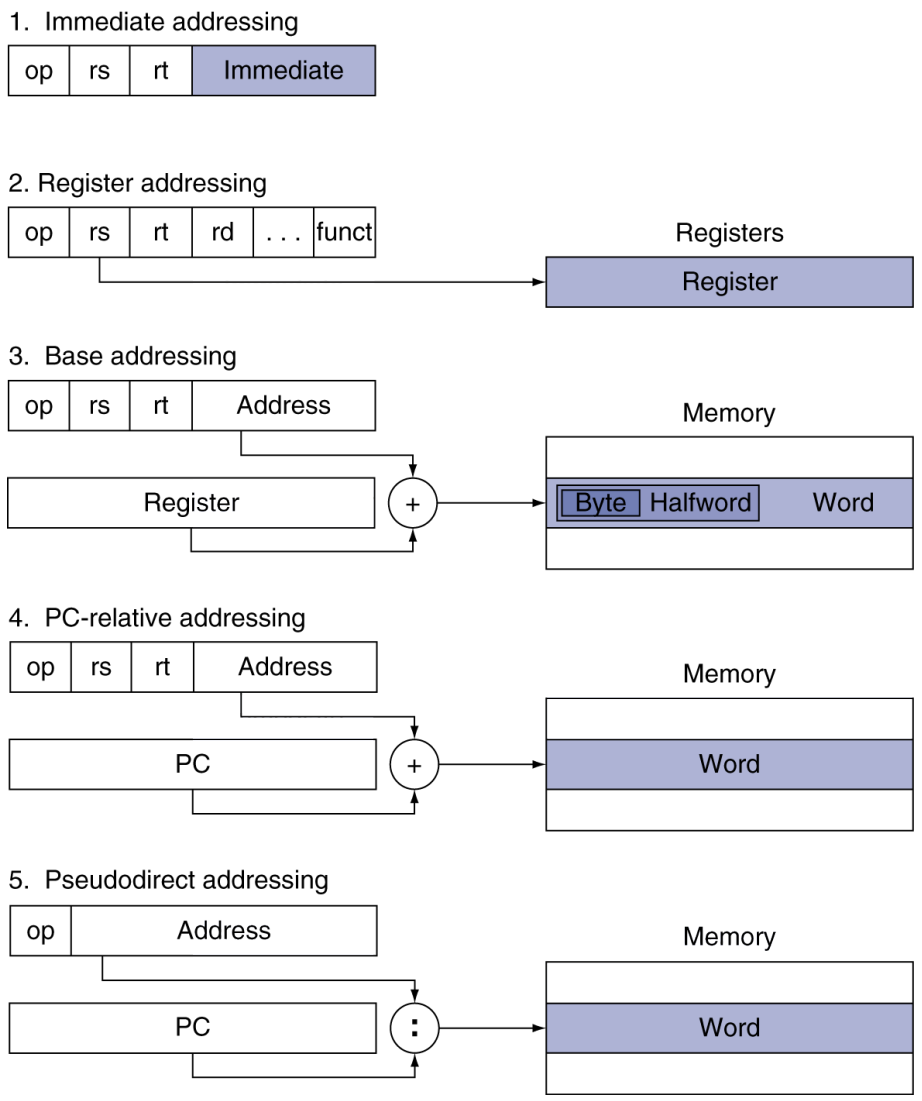
lui $at, 61



ori $at, $at, 2304



MIPS Addressing Mode Summary



Reading

- Section 2.8 - 2.10