COMPUTER ORGANIZATION AND DESIGN
The Hardware/Software Interface

EECS 2021

Computer Organization
Fall 2015

The slides are based on the publisher slides and contribution from Profs Amir Asif and Peter Lian
The slides will be modified, annotated, explained on the board, and sometimes corrected in the class

Based on slides by the author and prof. Mary Jane Irwin of PSU.

Procedure Calling

§2.8 Supporting Procedures in Computer Hardware

- Steps required
  1. Place parameters in a place where the procedure can access them
  2. Transfer control to procedure
  3. Acquire storage (resources) for procedure
  4. Perform procedure's operations
  5. Place result in a place where the caller can access them.
  6. Return to place of call

Chapter 2 — Instructions: Language of the Computer — 2

Register Usage

- $a0 – $a3: arguments (reg's 4 – 7)
- $v0, $v1: result values (reg's 2 and 3)
- $t0 – $t9: temporaries
  - Can be overwritten by callee
- $s0 – $s7: saved
  - Must be saved/restored by callee
- $gp: global pointer for static data (reg 28)
- $sp: stack pointer (reg 29)
- $fp: frame pointer (reg 30)
- $ra: return address (reg 31)

Chapter 2 — Instructions: Language of the Computer — 3

Procedure Call Instructions

- Procedure call: jump and link

  `jal ProcedureLabel`
  - Address of following instruction put in $ra
  - Jumps to target address
- Procedure return: jump register

  `jr $ra`
  - Copies $ra to program counter
  - Can also be used for computed jumps
    - e.g., for case/switch statements

Chapter 2 — Instructions: Language of the Computer — 4

Leaf Procedure Example

- C code:

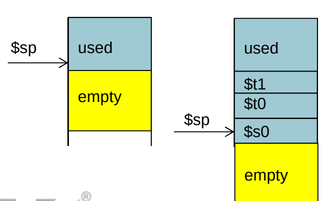
```
int leaf_example (int g, h, i, j)
{ int f;
  f = (g + h) - (i + j);
  return f;
}
```

  - Arguments g, …, j in $a0, …, $a3
  - f in $s0 (hence, need to save $s0 on stack)
  - Result in $v0
  - Will need $t0, and $t1 in the calculation of `f`

Chapter 2 — Instructions: Language of the Computer — 5

Stack

- The best way to store registers is a **stack**
- A stack is a first-in-last-out data structure
- Stack pointer points to the last element in the stack (or the first empty place).
- Traditionally stack grows from higher to lower addresses



The stack
The stack after ***pushing*** $t1 $t0 and $s0

Chapter 2 — Instructions: Language of the Computer — 6

Procedure Call

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

```
leaf_example:
addi   $sp, $sp, -12#adjust stack to make room for 3 items
sw     $t1, 8($sp)   # push $t1
sw     $t0, 4($sp)   # push $t0
sw     $s0, 0($sp)   # push $s0
```

??

Save registers

Chapter 2 — Instructions: Language of the Computer — 7

Procedure Call

```
add    $t0, $a0, $a1        #$t0 = g+h
add    $t1, $a2, $a3        #$t1 = i+j
sub    $s0, $t0, $t1        #$s0 = (g+h)-(i+j)

add    $v0, $s0, $zero      #put the result in $v0

lw     $s0, 0($sp)   #restore $s0
add    $t0, 4($sp)   #restore $t0
sub    $t1, 8($sp)   #restore $t1
addi   $sp, $sp, 12  #restore $sp

jr     $ra    #jump back to the calling routing
```

Do calculation

put result in $v0

Clean up (remove data from the stack)

Return control to caller

Chapter 2 — Instructions: Language of the Computer — 8

Leaf Procedure Example

- MIPS code:

```
leaf_example:
  addi $sp, $sp, -4        Save $s0 on stack
  sw   $s0, 0($sp)
  add  $t0, $a0, $a1       Procedure body
  add  $t1, $a2, $a3
  sub  $s0, $t0, $t1
  add  $v0, $s0, $zero     Result
  lw   $s0, 0($sp)         Restore $s0
  addi $sp, $sp, 4
  jr   $ra                 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

Non-Leaf Procedure Example

- C code:

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

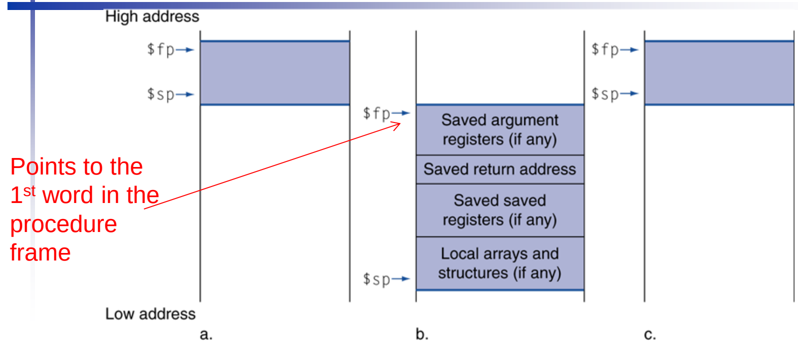
  - Argument n in $a0
  - Result in $v0

Non-Leaf Procedure Example

- 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
```

Local Data on the Stack

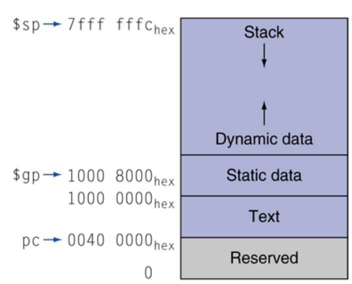


- Local data allocated by callee
  - e.g., C automatic variables
- **Procedure frame** (**activation record**)
  - Used by some compilers to manage stack storage
  - Fixed, does not change during the function execution
  - A stable base register to address for local memory reference

Chapter 2 — Instructions: Language of the Computer — 13

Memory Layout

- Text: program code
- Static data: global variables
  - e.g., static variables in C, constant arrays and strings
  - $gp initialized to address allowing ±offsets into this segment
- Dynamic data: heap
  - E.g., malloc in C, new in Java
- Stack: automatic storage



Chapter 2 — Instructions: Language of the Computer — 14

Character Data

§2.9 Communicating with People

- Byte-encoded character sets
  - ASCII: 128 characters
    - 95 graphic, 33 control
  - 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

Chapter 2 — Instructions: Language of the Computer — 15

String Copy Example

- C code (naïve):
  - 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

Chapter 2 — Instructions: Language of the Computer — 16

String Copy Example

- 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
```

Chapter 2 — Instructions: Language of the Computer — 17