ICS 233
Computer Architecture & Assembly Language

MIPS PROCESSOR INSTRUCTION SET

ICS 233
Computer Architecture & Assembly Language

Lecture 11

Lecture Outline

- ❑ **MIPS Procedure Call Instructions**
- ❑ **MIPS Procedure Return Instructions**
- ❑ **Parameter Passing**

Instructions for Procedures

- **JAL (Jump-and-Link) used as the call instruction**
  - Save return address in **$ra = PC+4** and jump to procedure
  - Register **$ra = $31** is used by **JAL** as the **return address**
- **JR (Jump Register) used to return from a procedure**
  - Jump to instruction whose address is in register Rs (PC = Rs)
- **JALR (Jump-and-Link Register)**
  - Save return address in Rd = PC+4, and
  - Jump to procedure whose address is in register Rs (PC = Rs)
  - Can be used to call methods (addresses known only at runtime)

Instruction	Meaning	Format					
jal label	$31=PC+4, jump	$op^6 = 3$	imm^{26}				
jr Rs	PC = Rs	$op^6 = 0$	rs^5	0	0	0	8
jalr Rd, Rs	Rd=PC+4, PC=Rs	$op^6 = 0$	rs^5	0	rd^5	0	9

MIPS – Procedure Call Instruction

- **jal (jump and link)**

- **Instruction Mnemonic :**

  jal addr ;where addr is the label of the target location

- **Meaning :**

  Jump to the location addr and store the address of the next instruction in $ra (i.e., $ra = PC +4 & jump to target address addr)

- **Example :**

  jal loop ; $ra = PC + 4; goto location having the label loop (used for procedure call)

MIPS – Procedure Return Instruction

- **jr (jump to address in register)**

- **Instruction Mnemonic :**

  jr rs ;where rs specifies the target address for jump

- **Meaning :**

  jump to target address specified in register rs

- **Example :**

  jr $ra ; goto $ra (used for procedure return)

MIPS – Procedure Call Instruction

- **jalr (jump and link register)**

- **Instruction Mnemonic :**

  jalr rd, rs ;where register rs contains the address of the target location and return address is stored in rd

- **Meaning :**

  Jump to the location in rs, and store the address of the next instruction in rd (used for procedure call)

- **Example :**

  jalr $ra, $t3 ; $ra = PC + 4; goto to target address in register $t3

Procedures

- Consider the following swap procedure (written in C)

```
void swap(int v[], int k)
{  int temp;
   temp = v[k]
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

- **Translate this procedure to MIPS assembly language**

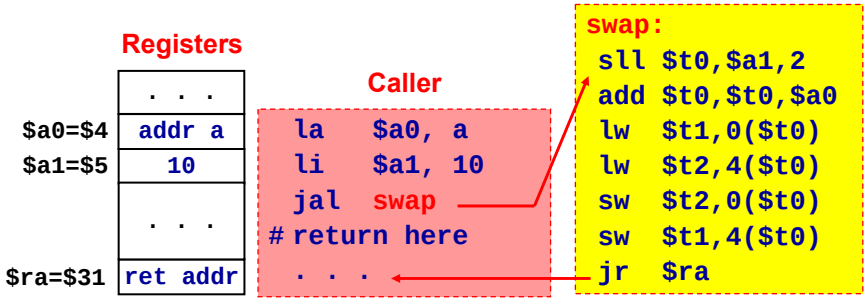
Parameters:

$a0 = Address of **v[]**
$a1 = **k**, and
Return address is in **$ra**

```
swap:
 sll $t0,$a1,2    # $t0=k*4
 add $t0,$t0,$a0  # $t0=v+k*4
 lw  $t1,0($t0)   # $t1=v[k]
 lw  $t2,4($t0)   # $t2=v[k+1]
 sw  $t2,0($t0)   # v[k]=$t2
 sw  $t1,4($t0)   # v[k+1]=$t1
 jr  $ra          # return
```

Call / Return Sequence

- Suppose we call procedure swap as: **`swap(a,10)`**
  - Pass **address** of array **`a`** and **`10`** as arguments
  - Call the procedure swap saving **return address** in **$31 = $ra**
  - Execute procedure swap
  - Return control to the point of origin (return address)



Details of JAL and JR

Address	Instructions	Assembly Language
00400020	lui $1, 0x1001	la $a0, a
00400024	ori $4, $1, 0	
00400028	ori $5, $0, 10	ori $a1,$0,10
0040002C	jal 0x10000f	jal swap
00400030	. . .	# return here
0040003C	sll $8, $5, 2	swap: sll $t0,$a1,2
00400040	add $8, $8, $4	add $t0,$t0,$a0
00400044	lw $9, 0($8)	lw $t1,0($t0)
00400048	lw $10,4($8)	lw $t2,4($t0)
0040004C	sw $10,0($8)	sw $t2,0($t0)
00400050	sw $9, 4($8)	sw $t1,4($t0)
00400054	jr $31	jr $ra

Pseudo-Direct Addressing

`PC = imm26<<2`

`0x10000f << 2 = 0x0040003C`

$31 0x00400030

Register $31 is the return address register

Parameter Passing

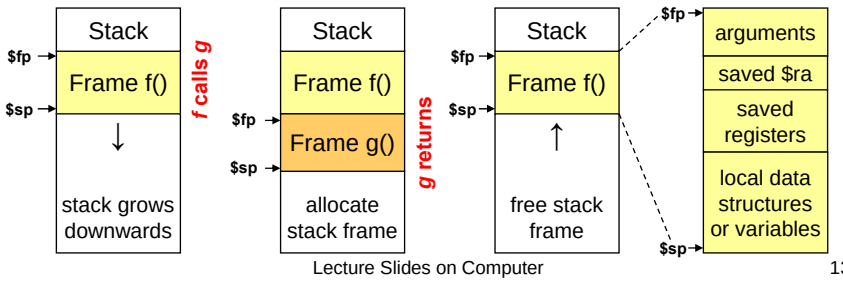
- **Parameter passing in assembly language is different**
  - More complicated than that used in a high-level language
- **In assembly language**
  - Place all required parameters in an accessible storage area
  - Then call the procedure
- **Two types of storage areas used**
  - Registers: general-purpose registers are used (register method)
  - Memory: stack is used (stack method)
- **Two common mechanisms of parameter passing**
  - Pass-by-value: parameter **value** is passed
  - Pass-by-reference: **address** of parameter is passed

Parameter Passing

- **By convention, registers are used for parameter passing**
  - **`$a0 = $4 .. $a3 = $7`** are used for **passing arguments**
  - **`$v0 = $2 .. $v1 = $3`** are used for **result values**
- **Additional arguments/results can be placed on the stack**
- **Runtime stack is also needed to …**
  - Store variables / data structures when they cannot fit in registers
  - Save and restore registers across procedure calls
  - Implement recursion
- **Runtime stack is implemented via software convention**
  - The **stack pointer `$sp = $29`** (points to top of stack)
  - The **frame pointer `$fp = $30`** (points to a procedure frame)

Stack Frame

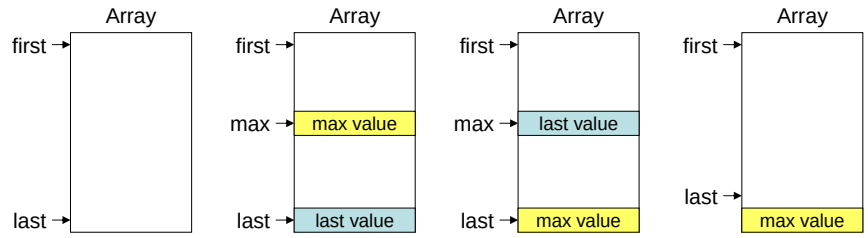
- **Stack frame is the segment of the stack containing …**
  - Saved arguments, registers, and local data structures (if any)
- **Called also the activation frame or activation record**
- **Frames are pushed and popped by adjusting …**
  - Stack pointer **`$sp = $29`** and Frame pointer **`$fp = R30`**
  - Decrement **`$sp`** to allocate stack frame, and increment to free



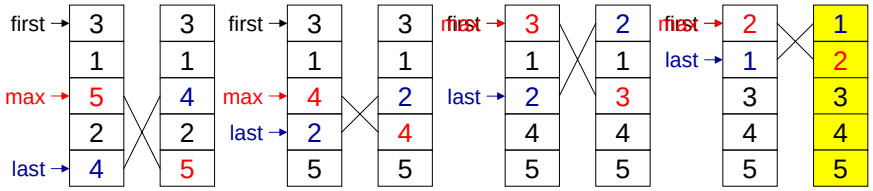
Preserving Registers

- **Need to preserve registers across a procedure call**
  - Stack can be used to preserve register values
- **Which registers should be saved?**
  - Registers modified by the called procedure, and
  - Still used by the calling procedure
- **Who should preserve the registers?**
  - **Called Procedure**: preferred method for modular code
    - Register preservation is done inside the called procedure
  - By convention, registers **$s0**, **$s1**, …, **$s7** should be preserved
  - Also, registers **$sp**, **$fp**, and **$ra** should also be preserved

Selection Sort



- Example



Selection Sort Procedure

```
# Objective: Sort array using selection sort algorithm
#     Input: $a0 = pointer to first, $a1 = pointer to last
#    Output: array is sorted in place
############################################################
sort: addiu $sp, $sp, -4   # allocate one word on stack
      sw    $ra, 0($sp)    # save return address on stack
top:  jal   max            # call max procedure
      lw    $t0, 0($a1)    # $t0 = last value
      sw    $t0, 0($v0)    # swap last and max values
      sw    $v1, 0($a1)
      addiu $a1, $a1, -4   # decrement pointer to last
      bne   $a0, $a1, top  # more elements to sort
      lw    $ra, 0($sp)    # pop return address
      addiu $sp, $sp, 4
      jr    $ra            # return to caller
```

Max Procedure

```
# Objective: Find the address and value of maximum element
#     Input: $a0 = pointer to first, $a1 = pointer to last
#    Output: $v0 = pointer to max,   $v1 = value of max
############################################################
max:  move  $v0, $a0        # max pointer = first pointer
      lw    $v1, 0($v0)     # $v1 = first value
      beq   $a0, $a1, ret   # if (first == last) return
      move  $t0, $a0        # $t0 = array pointer
loop: addi  $t0, $t0, 4     # point to next array element
      lw    $t1, 0($t0)     # $t1 = value of A[i]
      ble   $t1, $v1, skip  # if (A[i] <= max) then skip
      move  $v0, $t0        # found new maximum
      move  $v1, $t1
skip: bne   $t0, $a1, loop  # loop back if more elements
ret:  jr    $ra
```


Example of a Recursive Procedure

int fact(int n) { if (n<2) return 1; else return (n*fact(n-1)); }

```
main: lw $a0, N
      jal fact
      sw $v0, FAC
```

```
.data
N:    .word 3
FAC:  .space 4
```

```
fact:  slti   $t0,$a0,2     # (n<2)?
       beq    $t0,$0,else   # if false branch to else
       li     $v0,1         # $v0 = 1
       jr     $ra           # return to caller
else:  addiu  $sp,$sp,-8    # allocate 2 words on stack
       sw     $a0,4($sp)    # save argument n
       sw     $ra,0($sp)    # save return address
       addiu  $a0,$a0,-1    # argument = n-1
       jal    fact          # call fact(n-1)
       lw     $a0,4($sp)    # restore argument
       lw     $ra,0($sp)    # restore return address
       mul    $v0,$a0,$v0   # $v0 = n*fact(n-1)
       addi   $sp,$sp,8     # free stack frame
       jr     $ra           # return to caller
```

Reading Assignments

Chapter 2 : **Instructions : Language of the Computer**
Sections 2.1 to 2.16

- **From CD**

Appendix A : **Assemblers, Linkers & SPIM Simulator**
Sections A.1 to A.12