CPT_S 260 Intro to Computer Architecture
Lecture 16

Intro to MIPS V
February 16, 2022

Ganapati Bhat
School of Electrical Engineering and Computer Science
Washington State University

Recap: Conditional Operations

- **Branch to a labeled instruction if a condition is true**
  - Otherwise, continue sequentially
- **`beq rs, rt, L1`**
  - if (rs == rt) branch to instruction labeled L1;
- **`bne rs, rt, L1`**
  - if (rs != rt) branch to instruction labeled L1;
- **`j L1`**
  - unconditional jump to instruction labeled L1

Recap: Compiling Loop Statements

- **C code:**

```
while (save[i] == k)
    i += 1;
```

  – i in \$s3, k in \$s5, address of save in \$s6

- **Compiled MIPS code:**

```
Loop:  sll $t1,$s3,2          # Temp reg $t1 = i * 4
       add $t1,$t1,$s6        # $t1 = address of save[i]
       lw $t0,0($t1)          # Temp reg $t0 = save[i]
       bne $t0,$s5, Exit      # go to Exit if save[i] ≠ k
       addi $s3,$s3,1         # i = i + 1
       j  Loop                # go to Loop

Exit:
```

More Conditional Operations

- **Set result to 1 if a condition is true**
  - Otherwise, set to 0
- **`slt rd, rs, rt`**
  - if (rs < rt) rd = 1; else rd = 0;
- **`slti rt, rs, constant`**
  - if (rs < constant) rt = 1; else rt = 0;
- **Use in combination with beq, bne**

```
slt $t0, $s1, $s2  # if ($s1 < $s2)
bne $t0, $zero, L  #   branch to L
```

Recap: Example Switch Statements

- **The simplest way to implement switch is via a sequence of conditional tests, turning the switch statement into a chain of if-then-else statements.**

What is the MIPS assembly code assuming f-k correspond to registers $s0-$s5 and $t2 contains 4 and $t4 contains base address of JumpTable?

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

MIPS Code for Switch

```
add $t1,$s5,$s5      #$t1 =2*k
add $t1,$t1,$t1      #$t1 =4*k
add $t1,$t1,$t4      #$t1=address of JumpTable[k]
lw $t0,0($t1)        #$t0=JumpTable[k]
jr $t0               #jump based on register $t0
L0:   add $s0,$s3,$s4
      j Exit
L1:   add $s0,$s1,$s2
      j Exit
L2:   sub $s0,$s1,$s2
      j Exit
L3:   sub $s0,$s3,$s4
Exit:
```

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

Procedure Calling

- Steps required

1. Place parameters in registers
2. Transfer control to procedure
3. Acquire storage for procedure
4. Perform procedure’s operations
5. Place result in register for caller
6. Return to place of call

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)**
- **Program counter: A fixed register to hold the address of the instructions being executed**

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

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

Leaf Procedure Example

- 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

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

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

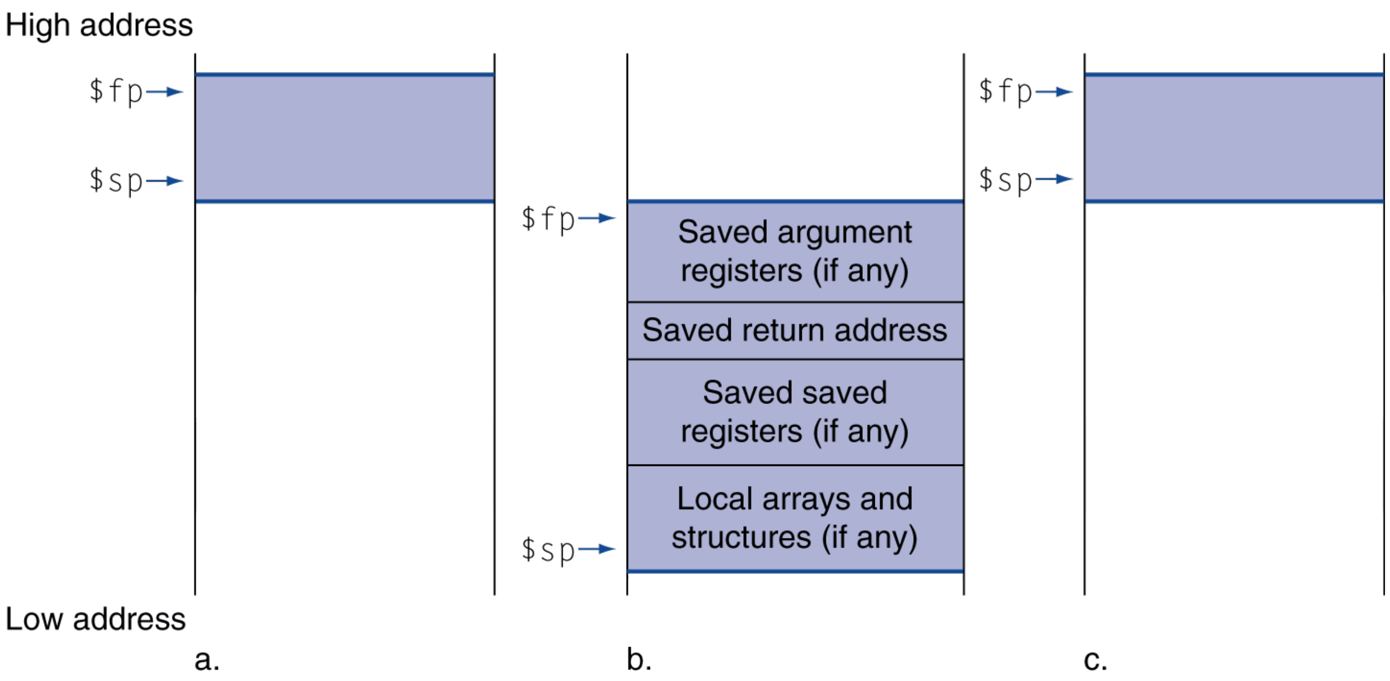
Non-Leaf Procedure Example

- 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
```

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

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

Branch Addressing

- **Branch instructions specify**
  - Opcode, two registers, target address
- **Most branch targets are near branch**
  - Forward or backward

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

- **PC-relative addressing**
  - **Target address = PC + offset × 4**
  - **PC already incremented by 4 by this time**

Jump Addressing

- **Jump (`j` and `jal`) targets could be anywhere in text segment**
  - Encode full address in instruction

op	address
6 bits	26 bits

- **(Pseudo)Direct jump addressing**
  - **Target address = $PC_{31\ldots28}$: (address × 4)**

Target Addressing Example

- **Loop code from earlier example**
  - Assume Loop at location 80000

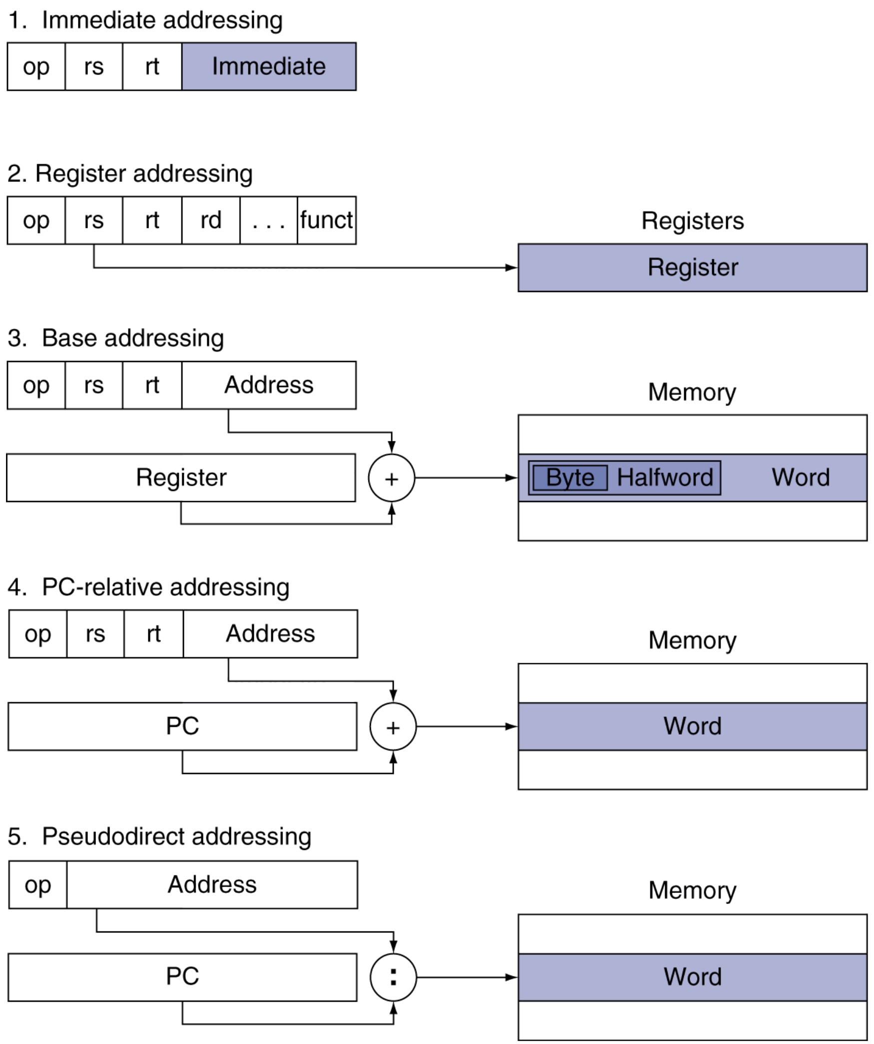
`Loop: sll $t1, $s3, 2`	80000	0	0	19	9	2	0
`add $t1, $t1, $s6`	80004	0	9	22	9	0	32
`lw $t0, 0($t1)`	80008	35	9	8	0		
`bne $t0, $s5, Exit`	80012	5	8	21	2		
`addi $s3, $s3, 1`	80016	8	19	19	1		
`j Loop`	80020	2	20000				
`Exit: …`	80024						

Branching Far Away

- If branch target is too far to encode with 16-bit offset, assembler rewrites the code
- Example

```
        beq $s0,$s1, L1
                ↓
        bne $s0,$s1, L2
        j L1
L2:     …
```

Addressing Mode Summary



QtSpim Tutorial

Program Structure

```
            .data       # variable declarations follow this line
                        # ...

            .text       # instructions follow this line

main:                   # indicates start of code (first instruction to execute)
                        # ...

# End of program, leave a blank line afterwards to make SPIM happy
```

System Calls

- To request a service, a program loads the system call:
- Code into register $v0 and
- The arguments into registers $a0, ..., $a3 (or $f12 for floating point values).
- System calls that return values put their result in register $v0 (or $f0 for floating point results)

Data Declarations

- **Declares variable names used in program;**
- **Storage allocated in main memory (RAM)**

format for declarations:

```
name:       storage_type   value(s)
```

Code

- Placed in section of text identified with assembler directive .text
- Contains program code (instructions)
- Starting point for code execution given label main:
- Ending point of main code should use exit system call (see below under System Calls)

System Calls

Service	System Call Code	Arguments	Result
print integer	1	$a0 = value	(none)
print float	2	$f12 = float value	(none)
print double	3	$f12 = double value	(none)
print string	4	$a0 = address of string	(none)
read integer	5	(none)	$v0 = value read
read float	6	(none)	$f0 = value read
read double	7	(none)	$f0 = value read
read string	8	$a0 = address where string to be stored $a1 = number of characters to read + 1	(none)
memory allocation	9	$a0 = number of bytes of storage desired	$v0 = address of block
exit (end of program)	10	(none)	(none)
print character	11	$a0 = integer	(none)
read character	12	(none)	char in $v0

Hello World Example

helloWorld.s or .asm file formats