CPT_S 260 Intro to Computer Architecture
Lecture 15

Intro to MIPS IV
February 14, 2022

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

Announcements

- **Homework 3 is online**
- **Exam 1**
  - In class on February 25
  - Material up to this Friday
  - One side cheat sheet is allowed
  - Must be handwritten, and not printed

Recap: Logical Operations

- **Instructions for bitwise manipulation**

Operation	C	Java	MIPS
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bitwise AND	&	&	and, andi
Bitwise OR	|	|	or, ori
Bitwise NOT	~	~	nor

- **Useful for extracting and inserting groups of bits in a word**

Recap: R-format Instructions

- op: Basic operation of the instruction, traditionally called the opcode.
- rs: The first register source operand
- rt: The second register source operand.
- rd: The register destination operand. It gets the result of the operation
- shamt: Shift amount (00000 for now)
- funct: Function. This field, often called the function code, selects the specific variant of the operation in the op field

Recap: R-format Example

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

```
add $t0, $s1, $s2
```

special	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

$$00000010001100100100000000100000_2 = 02324020_{16}$$

Recap: I-format Instructions

- **Immediate arithmetic and load/store instructions**
  - rt: destination or source register number
  - Constant: -2^{15} to $+2^{15} - 1$
  - Address: offset added to base address in rs
- ***Design Principle 4:* Good design demands good compromises**
  - Different formats complicate decoding, but allow 32-bit instructions uniformly
  - Keep formats as similar as possible

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

Compiling If Statements

- **C code:**

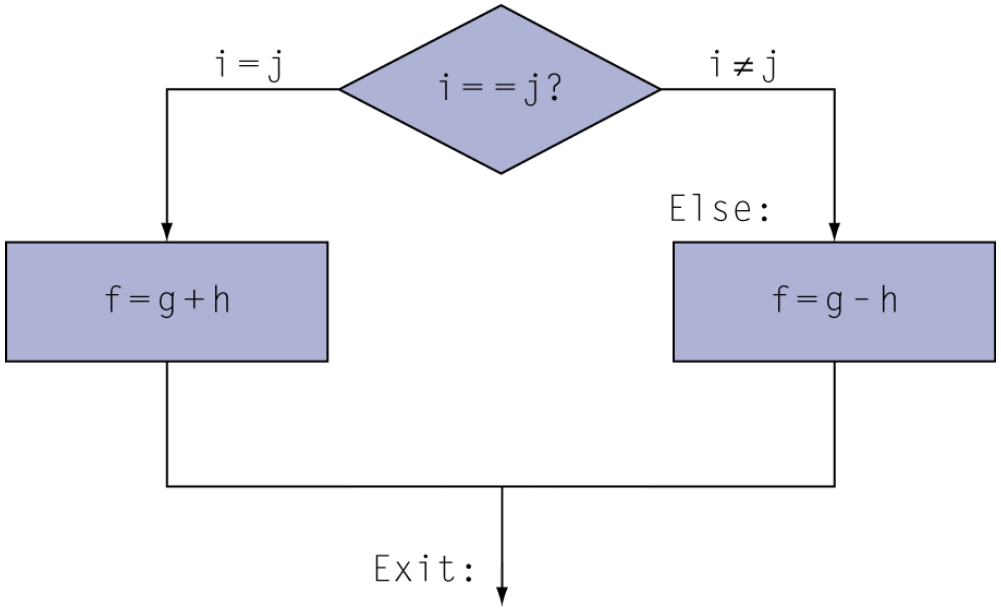
```
if (i==j) f = g+h;
else f = g-h;
```

  - f, g, … in $s0, $s1, …

- **Compiled MIPS code:**

```
        bne $s3, $s4, Else
        add $s0, $s1, $s2
        j   Exit
Else:   sub $s0, $s1, $s2
Exit:   …
```

Assembler calculates addresses



Compiling Loop Statements

- **C code:**

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

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

- **Steps**
  – Load save[i] into a temporary register.
    » Base Address of save
    » Multiply i to get the byte address of the index (by 4)
  – Loop test
    » Choose to use the bne or beq
  – Make branch labels for each portion of the loop test

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
```

Signed vs. Unsigned

- **Signed comparison: `slt`, `slti`**
- **Unsigned comparison: `sltu`, `sltui`**
- **Example**
  - $s0 = 1111 1111 1111 1111 1111 1111 1111 1111
  - $s1 = 0000 0000 0000 0000 0000 0000 0000 0001
  - `slt  $t0, $s0, $s1  # signed`
    - » –1 < +1 ⇒ $t0 = 1
  - `sltu $t0, $s0, $s1  # unsigned`
    - » +4,294,967,295 > +1 ⇒ $t0 = 0

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)**

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

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);
}
```

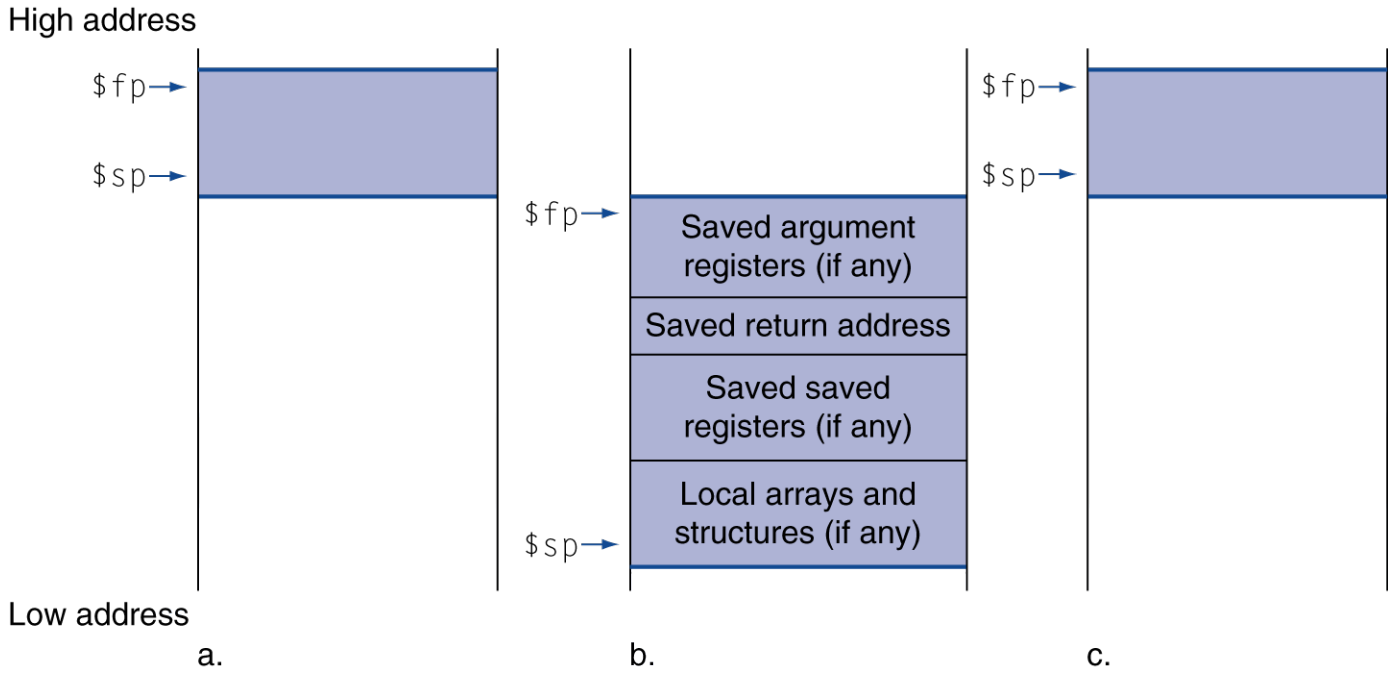
  - Argument n in $a0
  - Result in $v0

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