EECE 321: Computer Organization

Mohammad M. Mansour
Dept. of Electrical and Compute Engineering
American University of Beirut

Lecture 7: MIPS ISA

Announcements

- Reading assignment
  - Section 2.7
- Machine problem 1 posted
  - Due Friday March 12 @ 5:00pm
  - Submit on Moodle (one submission per team)
- Drop quiz next lecture

Making Decisions

- All instructions so far only manipulate data … we've built a *calculator*.
- In order to build a computer, we need ability to make decisions.
  - C (and MIPS) provide labels to support "goto" jumps to places in the code.
  - C: Horrible style; MIPS: Necessary!
- C Decisions: if Statements
- 2 kinds of if statements in C
  - if (condition) clause
  - if (condition) clause1 else clause2
- Rearrange 2nd if into following:
  - if (condition) goto L1;

    Clause2;

    goto L2;

    L1: clause1;

    L2: …
- Note: Labels are simply locations of statements in your code, and not instructions
- Not as elegant as if-else, but same meaning.

MIPS Decision Instructions: BEQ, BNE

- Decision instruction in MIPS:
  - beq register1, register2, L1
  - beq is "Branch if (registers are) equal"
    Same meaning as (using C):
    if (register1==register2) goto L1
- Complementary MIPS decision instruction
  - bne register1, register2, L1
  - bne is "Branch if (registers are) not equal"
    Same meaning as (using C):
    if (register1!=register2) goto L1
- beq and bne are called conditional branches

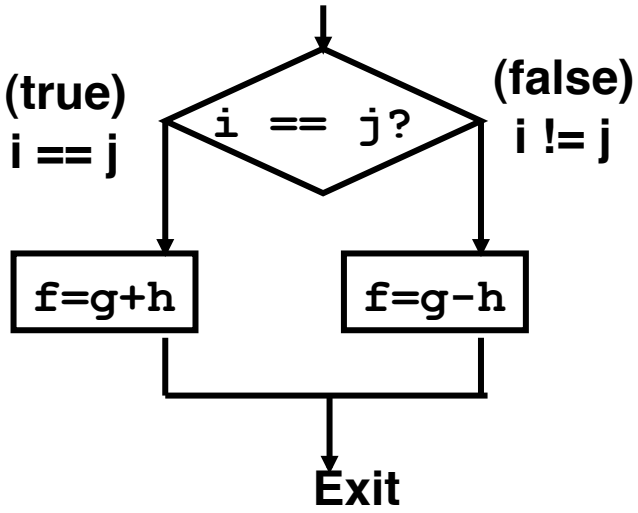
MIPS Goto Instruction: J

- In addition to conditional branches, MIPS has an unconditional branch:
  - j label
- Called a Jump Instruction: jump (or branch) directly to the given label without needing to satisfy any condition.
- Same meaning as (using C):
  - goto label
- Technically, it's the same as:
  - beq $0, $0, label

  since it always satisfies the condition.

Compiling C `if` Statements into MIPS

- Compile by hand the following C code:
  - `if(i == j)  f=g+h;`
    `else f=g-h;`
  - Use the following mappings:
    f: $s0
    g: $s1
    h: $s2
    i: $s3
    j: $s4



- Final compiled MIPS code:

```
        beq $s3, $s4, True      # branch i==j
        sub $s0, $s1, $s2       # f=g-h(false)
        j   Finish              # goto Finish
True:   add $s0, $s1, $s2       # f=g+h (true)
Finish:  …
```

- Note: Compiler automatically creates labels to handle decisions (branches). Generally not found in HLL code.

Loops in C/Assembly

- Simple loop in C; A[] is an array of int's
- Consider the following loop:

```
do{
   g = g + A[i];
   i = i + j;
} while (i != h);
```

- First, rewrite this as:

```
Loop:       g = g + A[i];
            i = i + j;
            if (i != h) goto Loop;
```

- Use this mapping:
  - `g:$s1, h:$s2, i:$s3, j:$s4, base of A:$s5`

- Final compiled MIPS code:

```
Loop:   sll $t1, $s3, 2       # $t1 = 4*i
        add $t1, $t1, $s5     # $t1 = address A
        lw  $t1, 0($t1)       # $t1 = A[i]
        add $s1, $s1, $t1     # g = g + A[i]
        add $s3, $s3, $s4     # i = i + j
        bne $s3, $s2, Loop    # goto Loop
                              # if i!=h
```

Loops in C/Assembly

- There are three types of loops in C:
  - while
  - do… while
  - for
- Each can be rewritten as either of the other two, so the method used in the previous example can be applied to while and for loops as well.
- Key Concept: Although there are multiple ways of writing a loop in MIPS, the key to decision making is conditional branch

Inequalities in MIPS

- Until now, we've only tested equalities (== and != in C).
  - General programs need to test < and > as well.
- Create a MIPS Inequality Instruction:
  - "Set on Less Than"
  - Syntax: `slt reg1, reg2, reg3`
- Meaning: set a register to '1' if a certain condition is satisfied

```
if (reg2 < reg3)
        reg1 = 1;
  else reg1 = 0;
```

- How do we use this? Compile by hand the following C statement:

```
if (g < h) goto Less;      // g:$s0, h:$s1
```

- Answer: compiled MIPS code…

```
     slt $t0, $s0, $s1        # $t0=1 if g<h
     bne $t0, $0, Less        # goto Less
                              # if $t0!=0
                              # (if (g<h))

Less: ...
```

- Branch if $t0 != 0 ⇔ (g < h)
- Register $0 always contains the value 0, so bne and beq often use it for comparison after an slt instruction.

Inequalities in MIPS

- Now, we can implement <, but how do we implement >, ≤ and ≥ ?
- We could add 3 more instructions, but:
  - MIPS goal: Simpler is Better
- Can we implement ≤ in one or more instructions using just slt and the branches?
  - What about >?
  - What about ≥?
- Ex: Implement ≤.

Immediates in Inequalities

- There is also an immediate version of slt to test against constants: slti
- Helpful in for loops
  - `if (g >= 1) goto Loop`

```
Loop:       . . .

            slti $t0, $s0, 1        # $t0 = 1 if $s0<1
                                    # (g<1)
            beq  $t0, $0, Loop      # goto Loop if $t0==0
                                    # (if (g>=1))
```

What About Unsigned Numbers?

- Also unsigned inequality instructions:
  - **`sltu, sltiu`**
  - They set result to 1 or 0 depending on unsigned comparisons
- Example: What is value of $t0, $t1 in the following instructions?
  - Let **`$s0 = 0xFFFF FFFA, $s1 = 0x0000 FFFA`**
  - **`slt $t0, $s0, $s1`**
  - **`sltu $t1, $s0, $s1`**
- So far, MIPS signed vs. unsigned:
  - Do/Don't sign extend (lb/lbu)
  - Don't overflow (addu, addiu, subu, multu, divu)
  - Do signed/unsigned compare (slt, slti/sltu, sltiu)

Example: The C Switch Statement

- Choose among four alternatives depending on whether k has the value 0, 1, 2 or 3. Compile this C code:
  - ```
    switch(k){
      case 0: f = i+j; break;        // k=0
      case 1: f = g+h; break;        // k=1
      case 2: f = g-h; break;        // k=2
      case 3: f = i-j; break;        // k=3
    }
    ```
- This is complicated, so simplify.
- Rewrite it as a chain of if-else statements, which we already know how to compile:
  - ```
    if(k==0)  f = i+j;
      else if(k==1) f = g+h;
                else if(k==2) f=g-h;
                     else if(k==3) f=i-j;
    ```
- Use this mapping:
  - `f:$s0, g:$s1, h:$s2, i:$s3, j:$s4, k:$s5`

Example: The C Switch Statement (cont'd)

- f:$s0, g:$s1, h:$s2, i:$s3, j:$s4, k:$s5
- Final compiled MIPS code:

```
if(k==0) f=i+j;
 else if(k==1) f=g+h;
   else if(k==2) f=g-h;
     else if(k==3) f=i-j;
```

```
    bne  $s5, $0, L1      # branch k!=0
    add  $s0, $s3, $s4    # k==0 so f=i+j
    j    Exit             # end of case so Exit

L1: addi $t0, $s5, -1     # $t0=k-1
    bne  $t0, $0, L2      # branch k!=1
    add  $s0, $s1, $s2    # k==1 so f=g+h
    j    Exit             # end of case so Exit

L2: addi $t0, $s5, -2     # $t0=k-2
    bne  $t0, $0, L3      # branch k!=2
    sub  $s0, $s1, $s2    # k==2 so f=g-h
    j    Exit             # end of case so Exit

L3: addi $t0, $s5, -3     # $t0=k-3
    bne  $t0, $0, Exit    # branch k!=3
    sub  $s0, $s3, $s4    # k==3 so f=i-j
Exit:
```

Example

- Consider the following MIPS assembly code:

```
Loop:   addi $s0,$s0,-1          # i = i - 1
        addi $s1,$s1, 1          # j = j + 1
        slti $t0,$s1,2           # $t0 = (j < 2)
        beq  $t0,$0 ,Loop        # goto Loop if $t0 == 0 => j >= 2
        slt  $t0,$s1,$s0         # $t0 = (j < i)
        bne  $t0,$0 ,Loop        # goto Loop if $t0 != 0 => j < i
```

- Assume the following mapping:
  - `i:$s0, j:$s1`

- What C code properly fills in the blank in loop below?

```
do {
  i--;
  j++;
}
while ( _________ );
```

Summary

- In order to help the conditional branches make decisions concerning inequalities, we introduced a single instruction:
  - "Set on Less Than" called slt, slti, sltu, sltiu
- One can store and load (signed and unsigned) bytes as well as words
- Unsigned add/sub don't cause overflow
- New MIPS Instructions:

  sll, srl

  slt, slti, sltu, sltiu

  addu, addiu, subu