CSE 31
Computer Organization

Lecture 15 – MIPS Conditionals

Announcements

- Labs
  - Lab 5 grace period ends ***next* week**
    - » No penalty for submission during grace period
    - » Demo is REQUIRED to receive full credit
  - Lab 6 out this week
    - » Due at 11:59pm on the same day of your lab after next (with **14 days grace period** after due date)
    - » You must demo your submission to your TA within 21 days from posting of lab
    - » Demo is REQUIRED to receive full credit
  - Lab 7 and Project 02 out next week
- Reading assignments
  - Reading 04 (zyBooks 4.1 – 4.9) due 20-MAR and Reading 05 (zyBooks 1.6 - 1.7, 6.1 - 6.3) due 03-APR
    - » Complete **Participation Activities** in each section to receive grade
    - » IMPORTANT: Make sure to submit score to CatCourses by using the link provided on CatCourses
- Homework assignment
  - Homework 04 (zyBooks 4.1 – 4.9) due 03-APR
    - » Complete **Challenge Activities** in each section to receive grade
    - » IMPORTANT: Make sure to submit score to CatCourses by using the link provided on CatCourses

Announcements

- Project 01
  - **Due 17-MAR**
  - Can work in teams of 2 students
    - » Each team member must identify teammate in “Comments…” text-box at the submission page
    - » If working in teams, each student must submit code (can be the same as teammate) and demo individually
    - » Grade can vary among teammates depending on demo
  - Demo required for project grade
    - » No partial credit for submission without demo
  - **No grace period**
    - » **Must complete submission and demo by due date.**
- Extra office hours to facilitate Project 01 demos posted on CatCourses

C Decisions: `if` Statements (review)

- 2 kinds of if-statements in C

  `if` (*condition*) *clause*

  `if` (*condition*) *clause1* `else` *clause2*

- Rearrange 2nd if-statement as shown below:

```
if   (condition) goto L1;
     clause2;
     goto L2;
L1:  clause1;
L2:
```

- Not as elegant as if-else, but same meaning

MIPS Decision Instructions

- 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
  ```

- Called conditional branches

MIPS Goto Instruction

- 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 effect as:

```
beq $0, $0, label
```

since it always satisfies the condition.

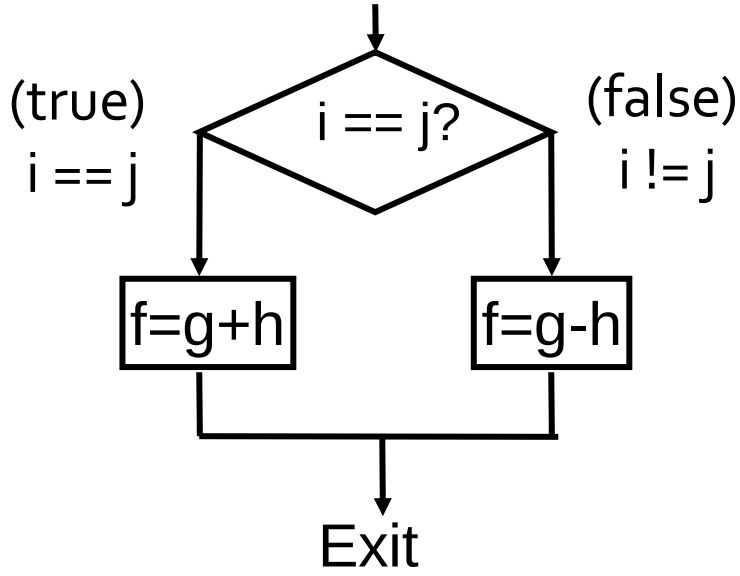
Compiling C `if` into MIPS (1/2)

- Compile by hand

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

- Use this mapping:

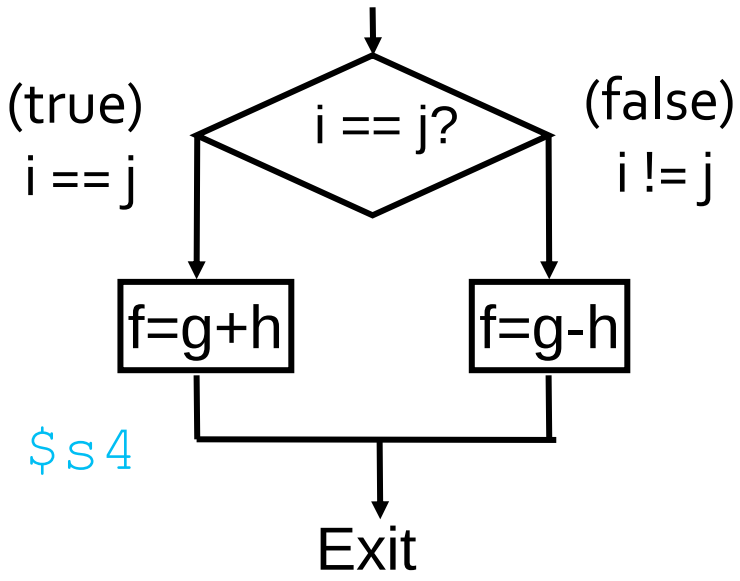
```
f: $s0
g: $s1
h: $s2
i: $s3
j: $s4
```



Compiling C `if` into MIPS (2/2)

- Compile by hand

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



`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   Fin            # goto Fin
True: add $s0,$s1,$s2    # f = g + h (true)
Fin:
```

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

Loading, Storing bytes 1/2

- In addition to word data transfers (**`lw`**, **`sw`**), MIPS has byte data transfers:
  - load byte: **`lb`**
  - store byte: **`sb`**

- Same format as `lw`, `sw`

- E.g., **`lb $s0, 3($s1)`**
  - contents of memory location with address = 3 + (contents of register ***`$s1`***) is copied to the **low byte position** of register ***`$s0`***.

Loading, Storing bytes 2/2

- What to do with other 24 bits in the 32-bit register?
  - lb: **sign extends** to fill upper 24 bits

- Normally don’t want to sign extend chars

- MIPS instruction that doesn’t sign extend when loading bytes:
  - load byte unsigned: **lbu**

Overflow in Arithmetic (1/2)

- Reminder: Overflow occurs when there is a mistake in arithmetic due to the limited precision in computers.

- Example (4-bit unsigned numbers):

15	**1111**
+ 3	**+ 0011**
18	**10010**

  – But we don't have room for 5-bit solution, so the solution would be **0010**, which is **+2**, and wrong.

Overflow in Arithmetic (2/2)

- Some languages detect overflow (Ada), some don't (C)

- MIPS solution is 2 kinds of arithmetic instructions:
  - These cause overflow to be detected
    - » add (**add**)
    - » add immediate (**addi**)
    - » subtract (**sub**)
  - These do not cause overflow detection
    - » add unsigned (**addu**)
    - » add immediate unsigned (**addiu**)
    - » subtract unsigned (**subu**)

- Compiler selects appropriate arithmetic
  - MIPS C compilers produce **addu, addiu, subu**

Two "Logic" Instructions

- Here are 2 more new instructions

- Shift Left: `sll $s1,$s2,2 # s1 = s2 << 2`
  - Store in `$s1` the value from `$s2` shifted 2 bits to the left (they fall off end), inserting 0's on right; << in C.
  - Before: $0000\ 0002_{hex}$
    $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two}$
  - After: $0000\ 0008_{hex}$
    $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1000_{two}$
  - What arithmetic effect does shift left have?
    » $n \times 2^i$

- Shift Right: `srl` is opposite shift; >>

Loops in C/Assembly (1/3)

- Simple loop in C; `A[]` is an array of `int`

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

How to write this in MIPS using what we have learned so far?

- Rewrite this as:

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

- Use this mapping:

```
g,   h,   i,   j,   base of A
$s1, $s2, $s3, $s4, $s5
```

Loops in C/Assembly (2/3)

- Final compiled MIPS code:

Why ???

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

- Original code:

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

```
g,  h,  i,  j,  base of A
$s1, $s2, $s3, $s4, $s5
```

Loops in C/Assembly (3/3)

- 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 these loops as well.

- Key Concept: Though there are multiple ways of writing a loop in MIPS, the key to decision-making is ***conditional branch***