CS61C - Machine Structures

Lecture 5 - Decisions in C/Assembly Language

September 13, 2000

David Patterson

http://www-inst.eecs.berkeley.edu/~cs61c/

Review (1/2)

- In MIPS Assembly Language:
  - Registers replace C variables
  - One Instruction (simple operation) per line
  - Simpler is Better
  - Smaller is Faster
- Memory is byte-addressable, but `lw` and `sw` access one word at a time.
- A pointer (used by `lw` and `sw`) is just a memory address, so we can add to it or subtract from it (using offset).

Review (2/2)

- New Instructions:

  `add, addi, sub, lw, sw`
- New Registers:

  C Variables: `$s0` - `$s7`

  Temporary Variables: `$t0` - `$t9`

  Zero: `$zero`

Overview

- C/Assembly Decisions: `if`, `if-else`
- C/Assembly Loops: `while`, `do while`, `for`
- Inequalities
- C Switch Statement

So Far...

- All instructions have allowed us to manipulate data.
- So we've built a calculator.
- In order to build a computer, we need ability to make decisions...
- Heads up: pull out some papers and pens, you'll do some in-class exercises today!

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;
    go to 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 as:**

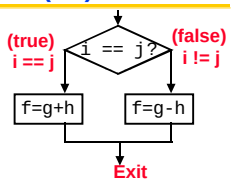
  `beq  $0,$0,label`

  since it always satisfies the condition.

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

- **Compile by hand**

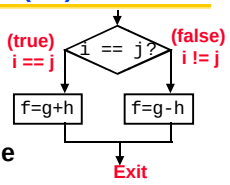
```
if (i == j) f=g+h;
else f=g-h;
```



- **Use this mapping:**

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

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



- **Final compiled MIPS code**
  (fill in the blank):

Loops in C/Assembly (1/3)

- **Simple loop in C**

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

- **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

Loops in C/Assembly (2/3)

- **Final compiled MIPS code**
  (fill in the blank):

Administrivia

- Kurt Meinz and Steve Tu heroically volunteer to add to their worloads, save Tu/Th 5-6 section

"What's This Stuff Good For?"

Breathing Observation Bubble: BOB pipes air from a tank under the handlebars into an acrylic dome, replacing a diver's face mask and breathing apparatus. Wireless technology lets riders talk to other BOBsters darting through the water nearby, as well as to armchair divers above in a boat or back on shore. Saving energy from not having to kick, divers can stay submerged almost an hour with the BOB. Like most modern scuba gear, the BOB features a computer that tells riders when to come up and calculates decompression times for a safe return to the surface. *One Digital Day*, 1998 www.intel.com/onedigitalday

What do applications ("apps") like these mean for reliability requirements of our technology?

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 `while` and `for` loops as well.
- Key Concept: Though there are multiple ways of writing a loop in MIPS, conditional branch is key to decision making

Inequalities in MIPS (1/4)

- 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:

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

  - In computereeze, "set" means "set to 1", "reset" means "set to 0".

Inequalities in MIPS (2/4)

- How do we use this?
- Compile by hand:

```
if (g < h) goto Less;
```

- Use this mapping:

g: $s0, h: $s1

Inequalities in MIPS (3/4)

- Final compiled MIPS code *(fill in the blank):*

Inequalities in MIPS (4/4)

- 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 >=?

Immediates in Inequalities

- There is also an immediate version of `slt` to test against constants: `slti`
  - Helpful in `for` loops

C

```
if (g >= 1) goto Loop
```

MIPS

What about unsigned numbers?

- there are unsigned inequality instructions:

  `sltu`, `sltiu`

- which set result to 1 or 0 depending on unsigned comparisons
- $s0 = FFFF FFFA$_{hex}$, $ s1 = 0000 FFFA$_{hex}$
- What is value of $t0, $t1?
- slt $t0, $s0, $s1
- sltu $t1, $s0, $s1

Example: The C Switch Statement (1/3)

- 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*/
  }
```


Example: The C Switch Statement (2/3)

- 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 (3/3)

- Final compiled MIPS code *(fill in the blank):*

Things to Remember (1/2)

- A Decision allows us to decide which pieces of code to execute at run-time rather than at compile-time.
- C Decisions are made using conditional statements within an `if`, `while`, `do while` or `for`.
- MIPS Decision making instructions are the conditional branches: `beq` and `bne`.
- In order to help the conditional branches make decisions concerning inequalities, we introduce a single instruction: "Set on Less Than"called `slt`, `slti`, `sltu`, `sltui`

Things to Remember (2/2)

- New Instructions:

```
beq, bne
j
slt, slti, sltu, sltiu
```