Decisions in C / Assembly Language

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.
- 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:

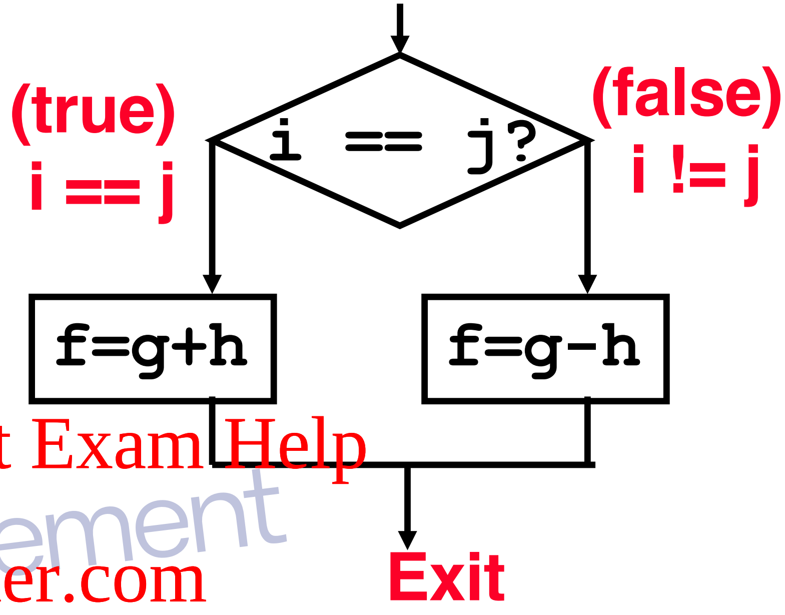
```
beq     $0,$0,label
```

since it always satisfies the condition.

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

- **Compile by hand**

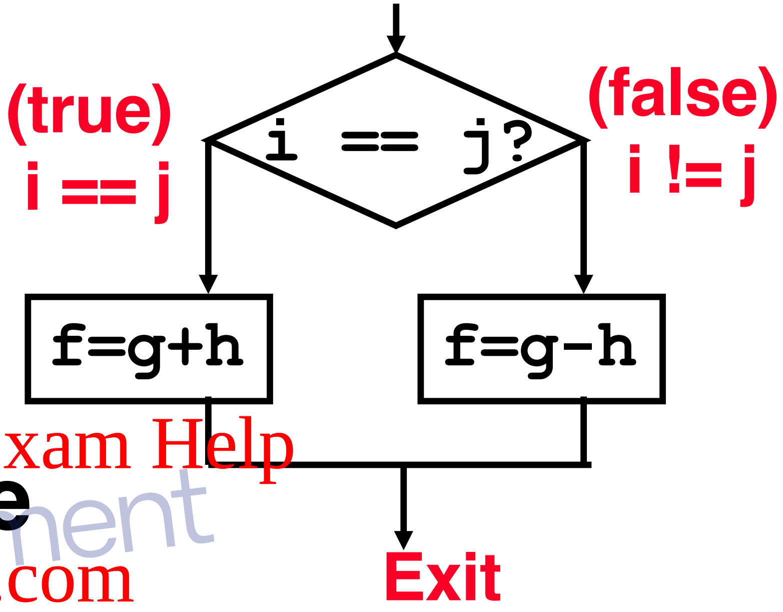
```
if (i == j) f=g+h;
else f=g-h;
```



- **Use this mapping:**

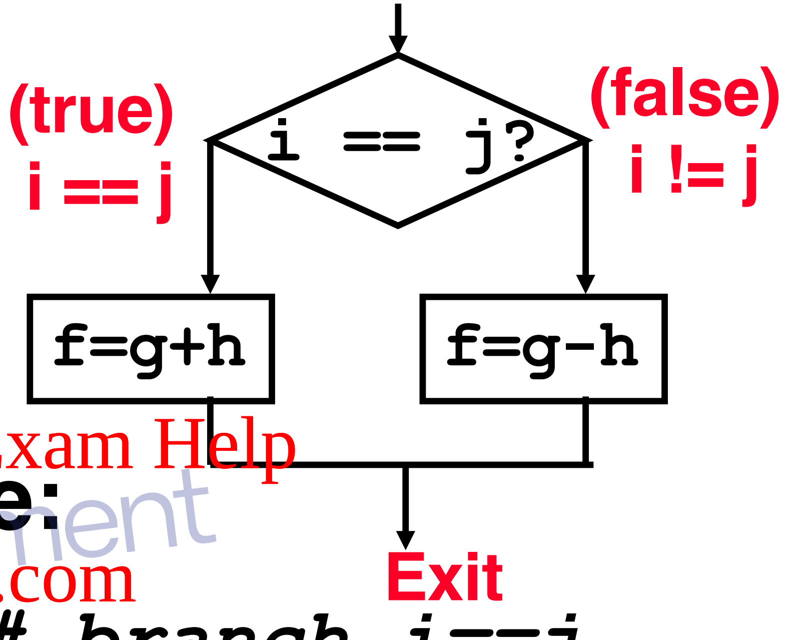
  `f: $s0, g: $s1, h: $s2, i: $s3, j: $s4`

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



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

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



- Final compiled MIPS code:

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

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

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):*

Loops in C/Assembly (2/3)

- **Final compiled MIPS code:**

```
Loop: sll $t1,$s3,2    #$t1= 4*i
      add $t1,$t1,$s5 #$t1=addr 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 (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 (1/5)

- 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 (2/5)

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

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

- Use this mapping:

  g: $s0, h: $s1

Inequalities in MIPS (3/5)

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

Inequalities in MIPS (3/5)

- Final 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 (4/5)

- 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 >=?
- 4 combinations of slt & beq/bneq

Inequalities in MIPS (5/5)

- 4 combinations of slt & beq/bneq:

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

slt $t0,$s1,$s0 # $t0 = 1 if g>h
bne $t0,$0,Grtr # if(g>h) goto Grtr

slt $t0,$s0,$s1 # $t0 = 1 if g<h
beq $t0,$0,Gteq # if(g>=h) goto Gteq

slt $t0,$s1,$s0 # $t0 = 1 if g>h
beq $t0,$0,Lteq # if(g<=h) goto Lteq
```

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

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

```
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?

- there are unsigned inequality instructions:

```
sltu, sltiu
```

- which set result to 1 or 0 depending on unsigned comparisons

- $\$s0 = \text{FFFF FFFA}_{hex}, \$s1 = \text{0000 FFFA}_{hex}$

- 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):*

Example: The C Switch Statement (3/3)

- Final compiled MIPS code:

```
      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:
```

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`.

- To help the conditional branches make decisions concerning inequalities, we introduce a single instruction: “Set on Less Than”called `slt`, `slti`, `sltu`, `sltiu`

Things to Remember (2/2)

- **New Instructions:**

```
beq, bne

j

slt, slti, sltu, sltiu
```