Compiling C if into MIPS (2/2)

- Compile by hand
  \[
  \text{if} \ (i == j) \ f=g+h; \\
  \text{else} \ f=g-h;
  \]

- Final compiled MIPS code:
  \[
  \begin{align*}
  \text{beq} & \ \$s3,\$s4,\text{True} \quad \# \ \text{branch} \ i==j \\
  \text{sub} & \ \$s0,\$s1,\$s2 \quad \# \ f=g-h(\text{false}) \\
  j & \ \text{Fin} \quad \# \ \text{goto} \ \text{Fin} \\
  \text{true: add} & \ \$s0,\$s1,\$s2 \quad \# \ f=g+h(\text{true}) \\
  \text{Fin:} & 
  \end{align*}
  \]

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

Review

- 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).
- A Decision allows us to decide what to execute at run-time rather than compile-time.
- C Decisions are made using conditional statements within if, while, do while, for.
- MIPS Decision making instructions are the conditional branches: beq and bne.

- New Instructions:
  - lw, sw, beq, bne, j

From last time: 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

Loading, Storing bytes 2/2

- What do with other 24 bits in the 32 bit register?
  - \text{lb}: sign extends to fill upper 24 bits
  - \text{...is copied to “sign-extend”}

- 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):
  \[
  \begin{align*}
  +15 & \quad 1111 \\
  +3 & \quad 0011 \\
  +18 & \quad 10010
  \end{align*}
  \]

  - 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 to recognize 2 choices:
  - add (add), add immediate (addi) and subtract (sub) cause overflow to be detected
  - add unsigned (addu), add immediate unsigned (addiu) and subtract unsigned (subu) do not cause overflow detection
• Compiler selects appropriate arithmetic
  - MIPS C compilers produce addu, addiu, subu

Two Logic Instructions
• 2 lectures ago we saw add, addi, sub
• Here are 2 more new instructions

Loops in C/Assembly (1/3)
• Simple loop in C; A[] is an array of ints
  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:
  $s1$, $s2$, $s3$, $s4$, $s5$
  $g$, $h$, $i$, $j$, base of A

Loops in C/Assembly (2/3)
• Final compiled MIPS code:
  Loop: sll $t1,$s3,2
         add $t1,$t1,$s5
         lw $t1,0($t1)
         add $s1,$s1,$t1
         add $s3,$s3,$s4 #i=i+j
         bne $s3,$s2,Loop

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, the key to decision making is conditional branch

Inequalities in MIPS (1/3)
• 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: reg1 = (reg2 < reg3);
  • Same thing...
  • In computeeeze, “set” means “set to 1”, “reset” means “set to 0”.

Two Logic Instructions
• 2 lectures ago we saw add, addi, sub
• Here are 2 more new instructions

Loops in C/Assembly (1/3)
• Simple loop in C; A[] is an array of ints
  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:
  $s1$, $s2$, $s3$, $s4$, $s5$
  $g$, $h$, $i$, $j$, base of A
Inequalities in MIPS (2/3)

• How do we use this? Compile by hand:
  
  \[
  \text{if (g < h) goto Less; } \# g : s0, h : s1
  \]

• Answer: compiled MIPS code...
  
  \[
  \text{slt }$t0,$s0,$s1 \quad \# t0 = 1 if g < h} \\
  \text{bne }$t0,$0,Less \\
  \text{# if (t0 != 0) Less:
  \]

• 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.

A slt \( \Rightarrow \) bne pair means if(... < ...) goto...

Inequalities in MIPS (3/3)

• Now, we can implement <, but how do we implement >, \( \leq \) and \( \geq \) ?

• We could add 3 more instructions, but:
  
  - MIPS goal: Simpler is Better

• Can we implement \( \leq \) in one or more instructions using just slt and the branches?
  
  - What about >?
  
  - What about \( \geq \)?

Immediates in Inequalities

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

  
  \[
  \text{C: if (g >= 1) goto Loop}
  \]

  \[
  \text{Loop: ...}
  \]

  \[
  \text{MIPS: slti }$t0,$s0,1 \quad \# t0 = 1 if $s0<1 (g<1) \\
  \text{bne }$t0,$0,Loop \quad \# goto Loop \\
  \text{# if (t0 == 0)}
  \]

  An slt \( \Rightarrow \) beq pair means if(... \( \geq \) ... goto...

What about unsigned numbers?

• Also unsigned inequality instructions: sltu, sltiu

  ...which sets result to 1 or 0 depending on unsigned comparisons

  \[
  \text{What is value of}\: \text{t0, t1?}
  \] ($s0 = FFFF FFFA_{hex}, s1 = 0000 FFFA_{hex}$)

  \[
  \text{slt }$t0, $s0, $s1 \\
  \text{sltu }$t1, $s0, $s1
  \]

MIPS Signed vs. Unsigned – diff meanings!

• MIPS Signed v. Unsigned is an “overloaded” term
  
  - Do/Don’t sign extend (lb, lbu)
  
  - Don’t overflow (addu, addiu, subu, multu, divu)

  - Do signed/unsigned compare (slt, slti/sltu, sltiu)

Administrivia

• Proj1 due in 9 days – start EARLY!
  
  - Out on Wed, due Friday [extended date]
  
  - The following hw (smaller) still due Wed

• We have a midterm & review date
  
  - Review: Sun 2005-03-06, Loc/Time TBA
  
  - Midterm: Mon 2005-03-07, Loc/Time TBA

  - DSP or Conflicts? Email acarle@cs

• Dan’s OH cancelled tomorrow
  
  - Go to the BEARS conference!
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:

```c
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:

```c
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:

```
bne \$s5,\$0, L1 # branch k!=0
    j Exit # end of case so Exit
L1: add \$t0,\$s5, -1 # \$t0=k-1
    bne \$s0,\$s5, \# branch k!=0
    add \$s0,\$s1,\$s2 # \$k=0 so f=g+h
    j Exit # end of case so Exit
L2: add \$t0,\$s5, -2 # \$t0=k-2
    bne \$s0,\$s5, \# branch k!=2
    add \$s0,\$s1,\$s2 # \$k=2 so f=g-h
    j Exit # end of case so Exit
L3: add \$t0,\$s5, -3 # \$t0=k-3
    bne \$t0,\$0, Exit # branch k!=3
    sub \$s0,\$s3,\$s4 # \$k=3 so f=i-j
Exit:
```

Peer Instruction

We want to translate \( x = y \) into MIPS

\( x,y \) ptrs stored in: \( \$s0,\$s1 \)

```c
A: add \$s0, \$s1, \$zero
B: add \$s1, \$s0, \$zero
C: lw \$t0, 0(\$s1)
D: lw \$s1, 0(\$s0)
E: lw \$t0, 0(\$s1)
F: sw \$t0, 0(\$s0)
G: lw \$s0, 0(\$t0)
H: sw \$s1, 0(\$t0)
```

Peer Instruction

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

And in conclusion...

• In order to help the conditional branches make decisions concerning inequalities, we introduce 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`
- `sll, srl, slli, sltu, sltiu`
- `addu, addiu, subu`