Overview 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

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
Two “Logic” Instructions

- Here are 2 new instructions.
- Shift Left: \texttt{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. \texttt{<<} in C.
  - Before: \texttt{000 0001 two 0000 0000 0000 0000 0101 hextwo}
  - After: \texttt{000 0001 two 0000 0000 0000 0000 1001 hextwo}
  - What arithmetic effect does shift left have?
- Shift Right: \texttt{srl} is opposite shift; G

Loops in C/Assembly (1/3)

- Simple loop in C: \texttt{A[]} is an array of \texttt{ints}
  \begin{verbatim}
  do { g = g + A[i];
       i = i + j;
  } while (i != h);
  \end{verbatim}
- Rewrite this as:
  \begin{verbatim}
  Loop: g = g + A[i];
       i = i + j;
  \end{verbatim}
  \begin{verbatim}
  if (i != h) goto Loop;
  \end{verbatim}
- Use this mapping:
  \begin{verbatim}
  g, h, i, j, base of A
  $s1, $s2, $s3, $s4, $s5
  \end{verbatim}

Loops in C/Assembly (2/3)

- Final compiled MIPS code:
  \begin{verbatim}
  Loop: sll $t1,$s3,2 # $t1= 4*i
  addu $t1,$t1,$s5 # $t1 = addr A[i]
  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
  \end{verbatim}
- Original code:
  \begin{verbatim}
  Loop: g = g + A[i];
       i = i + j;
  \end{verbatim}
  \begin{verbatim}
  if (i != h) goto Loop;
  \end{verbatim}

Loops in C/Assembly (3/3)

- There are three types of loops in C:
  \begin{itemize}
  \item while
  \item do … while
  \item for
  \end{itemize}
- 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.

Administrativa

- The schedule through week 7 has been determined
  - Midterm 7-9pm on 2011-10-06
- Other administrativa?

Inequalities in MIPS (1/4)

- Until now, we've only tested equalities (== and != in C). General programs need to test < and > as well.
- Introduce MIPS Inequality Instruction:
  - "Set on Less Than"
  - Syntax: \texttt{slt reg1,reg2,reg3}
  - Meaning: \texttt{reg1 = (reg2 < reg3)}
  \begin{verbatim}
  if (reg2 < reg3) 
    reg1 = 1;
  else reg1 = 0;
  \end{verbatim}
  - "Set" means "change to 1", "reset" means "change to 0".
Inequalities in MIPS (2/4)

- How do we use this? Compile by hand:
  \[ \text{if (} g < h \text{) goto Less; } \] 
  \[ \# g:$s0, h:$s1 \]
- Answer: compiled MIPS code...
  \[ \text{slt } \$t0,$s0,$s1 \# \$t0 = 1 \text{ if } g < h \]
  \[ \text{bne } \$t0,$0,\text{Less} \# \text{ goto Less} \]
  \# \text{ if } \$t0! = 0
  \# (if \( g<h \)) \text{ Less:}
- Register $0$ always contains the value 0, so \text{bne} and \text{beq} often use it for comparison after an \text{slt} instruction.
- A \text{slt} \rightarrow \text{bne} pair means if(...) \text{ goto...}

Inequalities in MIPS (3/4)

- 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 \text{slt} and branches?
  - What about >?
  - What about ≥?

Inequalities in MIPS (4/4)

\[
\begin{align*}
\# & a:$s0, b:$s1 \\
\text{slt } & \$t0,$s0,$s1 \# \$t0 = 1 \text{ if } a < b \\
\text{beq } & \$t0,$0,\text{skip} \# \text{ skip if } a \geq b \\
& \text{skip:}
\end{align*}
\]

Two independent variations possible:
- Use \text{slt} \$t0,$s1,$s0 instead of
- Use \text{bne} instead of \text{beq}

Immediates in Inequalities

- There is also an immediate version of \text{slt} to test against constants: \text{slti}
  - Helps in for loops

\[
\begin{align*}
& \text{C } \text{if (} g \geq 1 \text{) goto Loop} \\
& \text{Loop: . . .} \\
& \text{M } \text{slti } \$t0,$s0,1 \# \$t0 = 1 \text{ if } \\
& \text{P } \# $s0<1 \text{ (g<1)} \\
& \text{S } \text{beq } \$t0,$0,\text{Loop} \# \text{ goto Loop} \\
& \# \text{ if } \$t0==0 \\
& \# (if \( g<1 \))
\end{align*}
\]

An \text{slt} \rightarrow \text{beq} pair means if(...) \geq \text{ goto...}

What about unsigned numbers?

- Also unsigned inequality instructions:
  \text{sltu, sltiu}
  - which sets result to 1 or 0 depending on unsigned comparisons
- What is value of \$t0, \$t1?
  \[ \begin{align*}
  & (s0 = \text{FFFF FFFFA}, s1 = \text{0000 FFFFA}) \\
  & \text{slt } \$t0, s0, s1 \\
  & \text{sltu } \$t1, s0, s1
  \end{align*} \]

MIPS Signed vs. Unsigned – diff meanings!

- MIPS terms Signed/Unsigned “overloaded”:
  - Do/Don’t sign extend
    - (lb, lbu)
  - Do/Don’t overflow
    - (add, addi, sub, mult, div)
    - (addu, addiu, subu, multu, divu)
  - Do signed/unsigned compare
    - (slt, slti, sltu, sltiu)
Peer Instruction

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

($s0 = i, $s1 = j)

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

```c
do { --i } while(__);
```

"And in conclusion…"

- To help the conditional branches make decisions concerning inequalities, we introduce: "Set on Less Than" called
  - slt, slti, sltu, sltiu
- One can store and load (signed and unsigned) bytes as well as words with lb, lbu
- Unsigned add/sub don’t cause overflow
- New MIPS Instructions:
  - sll, srl, lb, lbu
- Unsigned add/sub don’t cause overflow
- New MIPS Instructions:
  - addu, addiu, subu

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:

```assembly
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
add $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:
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:

```assembly
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
add $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:
```