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?
  - lb: sign extends to fill upper 24 bits 
    
    \[ \text{xxxx xxxx xxxx xxxx xxxx xxxx} \rightarrow \text{xxxx xxxx xxxx xxxx} \]
    
    byte loaded
    
    This bit

- 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 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
- **Shift Left:** sll $s1,$s2,2 #s1=s2<<2
  - Store in $s1 the value from $s2 shifted 2 bits to the left, inserting 0’s on right; << in C
  - Before: 0000 0000 0000 0000 0000 0000 0000 0001
  - After: 0000 0000 0000 0000 0000 0000 0000 10
- **Shift Right:** srl $t1,$t2,2
  - What arithmetic effect does shift left have?
- **Shift Right:** srl is opposite shift; >> in C

### 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 is equal to (reg2 < reg3):
    - if (reg2 < reg3) reg1 = 1; else reg1 = 0;
  - In computerese, “set” means “set to 1”, “reset” means “set to 0”.
- Example:
  - Original code:
    
    Loop: do {
    
    if ($t0 != 0) goto Loop;

### Inequalities in MIPS (2/3)

- How do we use this? Compile by hand:
  
  if ($t0 = 1 if $t0 < $s1

- **Answer:** compiled MIPS code...
  
  ```
  slt $t0,$s0,$s1 # if $t0 < $s1
  bne $t0,$0,Less # goto Less if $t0 != 0
  ```

- **Branch if $t0 != 0:**
  
  ```
  if (g < h)
  ```

- **Register $s0 always contains the value 0, so bne and beq often use it for comparison after an slt instruction.**

### Loops in C/Assembly (1/3)

- Simple loop in C; A[] is an array of ints
  ```
  Loop: do {
  
  g = g + A[i];
  i = i + j;
  
  if (i != h) goto Loop;
  
  } while (i != h);

### Loops in C/Assembly (2/3)

- Final compiled MIPS code:
  ```
  Loop: sll $t1,$s3,2 #s1=4*i
  add $t1,$t1,$s5 #s1=addr A
  lw $t1,0($t1) #t1=A[i]
  add $s3,$s1,$s1 #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
  ```

- 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 (3/3)

• 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
  
Immediates in Inequalities

MIPS
  slti $t0,$s0,1  # $t0 = 1 if $s0<1 (g<1)
  beq  $t0,$0,Loop  # goto Loop if $t0==0 (if (g>=1))

A slt ➔ beq pair means if(… ≥ …)goto…

What about unsigned numbers?

• Also unsigned inequality instructions:
  - slt, sltiu
  ...which sets result to 1 or 0 depending on unsigned comparisons
• What is value of $t0, $t1?
  ($s0 = FFFF FFFA hexadecimal, $s1 = 0000 FFFA hexadecimal)
  - slt $t0, $s0, $s1
  - 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, mulu, divu)
  - Do signed/unsigned compare
    (slt, slti, sltu, sltiu)

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:
  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

Webcasts

Due to the recent budget crunch, our dept may not be able to pay for WebCasts anymore. We could either drop the service or treat it as a ‘course material fee’ (CMF). I.e., enrolled students in classes that are webcast would share the cost. Estimated costs would be ~$12 / student / semester. We want feedback!

A. On the whole, are Webcasts a useful service we should keep providing?
B. Would you support keeping webcasts if the only way to do so would be to treat them as CMFs?
C. Would an extra $12 cause you financial hardship?

Peer Instruction

We want to translate *x = *y into MIPS

(x, y ptrs stored in: $s0 $s1)
A: add $s0, $s1, zero
B: add $s1, $s0, zero
C: lw $s0, 0($s1)
D: lw $s1, 0($s0)
E: lw $t0, 0($s1)
F: sw $t0, 0($t0)
G: lw $s0, 0($t0)
H: sw $s1, 0($t0)

„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, situ, sltiu
• One can store and load (signed and unsigned) bytes as well as words
• Unsigned add/sub don’t cause overflow
• New MIPS Instructions:
  slt, slti
  sbl, situ, sltiu
  addu, addiu, subu

Peer Instruction

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

What C code properly fills in the blank in loop below?
do (i=--) while(__);