“SO MANY GADGETS, SO MANY ACHES” NYT

Laptops “do not meet any of the ergonomic requirements for a computer system”. Touch screens “should not be used heavily for typing” Texting is a problem because thumb bones have two bones instead of three ... “if you want to get injured, do a lot of texting”. Advice? Take a break

www.nytimes.com/2010/02/19/technology/19china.html

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

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
- E.g., lb $s0, 3($s1)
  - contents of memory location with address = sum of “3” + contents of register s1 is copied to the low byte position of register s0.

Loading, Storing bytes 2/2
- What do with other 24 bits in the 32 bit register?
  - lb: sign extends to fill upper 24 bits
  - ...is copied to “sign-extend”
  - 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 (most C implementations)
- MIPS solution is 2 kinds of arithmetic instructs:
  - 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
Two “Logic” Instructions

- 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 (they fall off end), inserting 0’s on right; << in C.
  - Before: 0000 0002 0000 0000 0000 0000 0000 0010
  - After: 0000 0008 0000 0000 0000 0000 0000 1000
- What arithmetic effect does shift left have?
- Shift Right: srl is opposite shift; >>

Loops in C/Assembly (1/3)

- Simple loop in C: A[i] 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:
  - g, h, i, j, &A[0]
  - $s1, $s2, $s3, $s4, $s5

Loops in C/Assembly (2/3)

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

Administrivia

- HW2 is due Sunday at 23:59:59

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: slt reg1,reg2,reg3
  - Meaning: reg1 = (reg2 < reg3);
  - If reg2 < reg3, reg1 = 1;
  - else reg1 = 0;
  - "set" means "change to 1";
  - "reset" means "change to 0".
Inequalities in MIPS (2/4)

- How do we use this? Compile by hand:
  if (g < h) goto Less; #g:$s0, h:$s1
- Answer: 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:
- 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 (3/4)

- 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 branches?
    - What about >?
    - What about ≥?

Inequalities in MIPS (4/4)

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

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

Immediates in Inequalities

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

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

An slt \( \rightarrow \) beq pair means if(… ≥ ...)goto…

What about unsigned numbers?

- Also unsigned inequality instructions:
  sltu, sltiu
  ...which sets result to 1 or 0 depending on unsigned comparisons
- What is value of $t0, $t1?
  \[
  (\mathrm{ss0} = \mathrm{FFFF FFFA}\_\text{Hex}, \mathrm{ss1} = 0000 \mathrm{FFFA}\_\text{Hex})
  \]
  \[
  \begin{align*}
  \text{slt} & \quad $t0, \mathrm{ss0}, \mathrm{ss1} \\
  \text{sltu} & \quad $t1, \mathrm{ss0}, \mathrm{ss1} \\
  \end{align*}
  \]

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, sltiu)
Peer Instruction

Loop:  
add $s0,$s0,-1  # i = i - 1  
slli $s0,$s1,2  # $i0 = (j < 2)  
beq $i0,$0,Loop  # goto Loop if $i0 == 0  
slt $s0,$s1,$s0  # $i0 = (j < i)  
bne $i0,$0,Loop  # goto Loop if $i0 != 0  

($s0=i, $s1=j)  
What C code properly fills in the blank in loop below?  
do (j--;) 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

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 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:

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