Lecture 10 – Introduction to MIPS
Decisions II

2004-09-22

Lecturer PSOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

Bill Gates visits Cal!
Oct 1 @ 9am, he’ll speak @ Zel! Only Eng + L&S CS students (& fac) allowed in, free tix Sep 24th @ 9am at E side of McLaughlin Hall.

www.coe/engnews/Fall04/EN04F/bill.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`
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
What do with other 24 bits in the 32 bit register?

- **lb**: sign extends to fill upper 24 bits

\[
\begin{array}{cccccccc}
xxxx & xxxx & xxxx & xxxx & xxxx & xxxx & xxxx & xxxx \\
\end{array}
\]

...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):

  +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: \texttt{sll} \ $s1,$s2,2  \ #s1=s2\ll2
  
  • Store in \$s1 the value from \$s2 shifted 2 bits to the left, inserting 0’s on right; \ll in C
  
  • Before: \begin{align*}
    0000 & 0002_{\text{hex}} \\
    0000 & 0000 0000 0000 0000 0000 0000 0010_{\text{two}}
  \end{align*}
  
  • After: \begin{align*}
    0000 & 0008_{\text{hex}} \\
    0000 & 0000 0000 0000 0000 0000 0000 1000_{\text{two}}
  \end{align*}
  
  • What arithmetic effect does shift left have?

• Shift Right: \texttt{srl} is opposite shift; \gg
Loops in C/Assembly (1/3)

• Simple loop in C; A[] is an array of ints

\[
\begin{align*}
do & \{ \\
g & = g + A[i] \\
i & = i + j \\
\} \text{ while } (i \neq h) ;
\end{align*}
\]

• Rewrite this as:

\[
\text{Loop: } g = g + A[i] \\
i = i + j \\
\text{if } (i \neq h) \text{ goto Loop;}
\]

• Use this mapping:

\[
g, \quad h, \quad i, \quad j, \quad \text{base of A} \\
$s1, \quad s2, \quad s3, \quad s4, \quad s5$
\]
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
```

• 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 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: \texttt{slt reg1,reg2,reg3}
  • Meaning: \texttt{reg1 = (reg2 < reg3);}
  
  \begin{verbatim}
  if (reg2 < reg3)
    reg1 = 1;
  else reg1 = 0;
  \end{verbatim}

  • In computereese, “set” means “set to 1”, “reset” means “set to 0”.

Garcia, Fall 2004 © UCB
Inequalities in MIPS (2/3)

• 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:
```

• 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 ➔ 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 $\text{slt}$ 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

```
C             if (g >= 1) goto Loop

M          Loop: . . .

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:
  
  \texttt{sltu, sltiu}

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

• What is value of $t0, t1$?

  \((s0 = \text{FFFF FFFA}_{\text{hex}}, s1 = 0000 \text{ FFFA}_{\text{hex}})\)

  \texttt{slt \ $t0, s0, s1}

  \texttt{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/sltau, 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 time & place for the midterm & review
  • Review: Sun 2004-10-17, 2pm. 10 Evans
  • Midterm: Mon 2004-10-18, 7-10 pm. 1 Pim
  • DSP or Conflicts? Email acarle@cs

• Anyone can go to anyone’s OH

• UCBUGG (UCB Undergrad Graphics Group)
  • Openings 2005Sp; we want people w/3D experience or artists. Learn Maya PLE!
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:

```plaintext
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
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:
```
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?

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>6</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
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(\(\$s0\))
G: lw \(\$s0\), 0(\(\$t0\))
H: sw \(\$s1\), 0(\(\$t0\))
Peer Instruction

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

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

(\$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`
  - `slt`, `slti`, `sltu`, `sltiu`
  - `addu`, `addiu`, `subu`