Mips Code Examples

• Peter Rounce
  P.Rounce@cs.ucl.ac.uk
Some C Examples

Assignment: \( \text{int } j = 10; // space must be allocated to variable } j \)

Possibility 1: \( j \) is stored in a register, i.e. register $2

then the MIPS assembler for this is :-

\[
\text{addi } \$2, \$0, 10 \quad : \$2 \leftarrow \$0 + \text{sign-extend}[10]
\]

Possibility 2: \( j \) is stored in memory, i.e. memory 0x12345678

then the MIPS assembler for this might be:-

\[
\begin{align*}
\text{lui } \$1, 0x1234 & \quad : \$1 \leftarrow 0x12340000 \quad \text{Get address in } \$1 \\
\text{ori } \$1, \$1, 0x5678 & \quad : \$1 \leftarrow 0x12345678 \\
\text{addi } \$8, \$0, 10 & \quad : \$8 \leftarrow \$0 + \text{sign-extend}[10] \quad \text{Get } 10 \text{ in } \$8 \\
\text{sw } \$8, 0(\$1) & \quad : \text{Mem}[\$1 + 0] \leftarrow \$8 \quad \text{Store } 10 \rightarrow 0x12345678
\end{align*}
\]
Program to calculate Absolute value of difference between 2 input numbers: \(|A - B|\) (demonstrates if)

Program reads A from 4 bytes of memory starting at address 12345670\(_{16}\).
Program reads B from 4 bytes of memory starting at address 12345674\(_{16}\).
Program writes \(|A-B|\) to 4 bytes of memory starting at address 12345678\(_{16}\).

Assembler

<table>
<thead>
<tr>
<th></th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>lui</td>
<td>$10, 0x1234</td>
</tr>
<tr>
<td>ori</td>
<td>$10, $10, 0x5670 # put address of A into register $10</td>
</tr>
<tr>
<td>lw</td>
<td>$4, 0($10) # read A from memory into register $4</td>
</tr>
<tr>
<td>lw</td>
<td>$5, 4($10) # read B from memory into register $5 (A address+4)</td>
</tr>
<tr>
<td>sub</td>
<td>$12, $5, $4 # subtract A from B =&gt; B-A into register $12</td>
</tr>
<tr>
<td>bgez</td>
<td>$12,+1 # branch if B-A is positive to ‘sw’ instruction</td>
</tr>
<tr>
<td>sub</td>
<td>$12, $4, $5 # subtract B from A =&gt; A-B into register $12</td>
</tr>
<tr>
<td>sw</td>
<td>$12, 8($10) # store register $12 value, (</td>
</tr>
</tbody>
</table>

N.B. program uses displacement to access other locations from address of memory storing value of A
Given the binary for an instruction e.g.:

10101101111010001000000000000000

What code would you write to get the $r_s$ register number into a register on its own, and into the low bits of this register?

```
Code: assume code in $4

// get masking value in $5

lui $5, 0x03e0

and $6, $5, $4

// masked value in $6

// so shift $6 right

srl $6, $6, 21
```

What is wanted.

Shift right logical
Change $r_s$ field in instruction to value $21_{10}$ ($10101_2$):

Code: 10101101111010001000000000000000

What is wanted.

Code: assume code in $4$

```
// get masking value in $5$
lui $5, 0xfc1f
ori $5, $5, 0xffffff
```

```
and $6, $5, $4
```

```
// new value into $5$
addiu $5, $0, 0x15
sll $5, $5, 21
or $6, $6, $5
```
**Shift Instructions:**

Shift left logical:  \texttt{sll rd, rt, shift-amount}  
\begin{center}
\text{rd} \leftarrow \text{rt} \ll \text{shift-amount} : 0s placed on right
\end{center}

Example: Let $4 == 2$, then  
\begin{center}
sll \ 5, \ 4, \ 3
\end{center}
shifts the contents of $4$ left 3 places: $(2 \ll 3) \rightarrow 16$ which is stored in $5$.

\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|c|c|c|}
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\hline
\end{tabular}

\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|c|c|c|}
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\hline
\end{tabular}

\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|c|c|c|}
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\hline
\end{tabular}

\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|c|c|c|}
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\hline
\end{tabular}
shift left logical variable: sllv rd, rt, rs
: rs holds shift- amount for shifting rt with result into rd
: rd ← rt << rs

shift right logical: reverse of shift left logical
srl rd, rt, shift-amount : 0s placed on left
e.g. $5 = 16 then srl $6, $5, 3 : $6 ← 16 >> 3
$6 == 2 after instruction

shift right logical variable: srlv rd, rt, rs as sllv but shift right

Shift right arithmetic: shift right with sign duplication

shift right arithmetic: sra rd, rt, shift-amount
shift right arithmetic variable: srav rd, rt, rs

arithmetic shifts duplicate the sign bit: Is are placed on right for -ve values

1111110000 (>> 2) → 1111111100
0011110000 (>> 2) → 0000111100

10/7/2012 GC03 Mips Code Examples
Branches - a **Reminder!!!!!!**

Instructions are always 4 bytes long in Mips.

Instructions are always stored at addresses that are an integer multiple of 4: - 0, 4, 8, … 0x2C, 0x30, …. 0x12345678, 0x1234567C…..

pc always points at an instruction,
    i.e. pc always holds a multiple of 4

Branches always change pc by a multiple of 4

Branch offset is number of instructions to branch, not number of addresses!

Branch target address calculation: - pc + (offset *4)
Conditional Branch Instructions – using labels
calculating offsets is difficult – use a label instead!

Branch Equal
\[
\text{beq rs, rt, Label}\\
\text{: if } rs == rt \text{ pc }<-\text{ pc + (address of label – pc)}\\
\text{: if } rs == rt \text{ pc }<-\text{ pc + offset*4}
\]

Branch Not-Equal
\[
\text{bne reg1, reg2, Label}\\
\text{: if } rs != rt \text{ pc }<-\text{ pc + (address of label – pc)}\\
\text{: if } rs != rt \text{ pc }<-\text{ pc + offset*4}
\]

Assembler Program calculates difference between address of instruction following
the branch and the address of Label (label address – pc), divides by 4 and stores
this value, the number of instructions to branch, in offset field of instruction.
Other Branches

These branches test the contents of a single register against 0.

branch on greater than or equal zero:

\[
\text{bgez register, label} \\
\text{if (register} \geq 0) \text{ pc } \leftarrow \text{ address of label} \\
\text{if (register} \geq 0) \text{ pc } \leftarrow \text{pc + offset*4}
\]

branch on greater than zero:

\[
\text{bgtz register, label} \\
\text{if (register} > 0) \text{ pc } \leftarrow \text{ address of label} \\
\text{if (register} > 0) \text{ pc } \leftarrow \text{pc + offset*4}
\]

branch on less than or equal zero:

\[
\text{blez register, label} \\
\text{if (register} \leq 0) \text{ pc } \leftarrow \text{ address of label} \\
\text{if (register} \leq 0) \text{ pc } \leftarrow \text{pc + offset*4}
\]

branch on less than zero:

\[
\text{bltz register, label} \\
\text{if (register} < 0) \text{ pc } \leftarrow \text{ address of label} \\
\text{if (register} > 0) \text{ pc } \leftarrow \text{pc + offset*4}
\]

Note: branches can only go –32768 instructions back & 32767 forward
memory address space in Mips is 1G instructions!!!!!!!!!!
What about comparing 2 registers for < and >=?

Use a Set instruction followed by a conditional branch.

Comparison Instructions
R-Format versions: compare 2 register and put result into 3\textsuperscript{rd} register
- Set less than (signed): \texttt{slt rd, rs, rt} : if \( rs < rt \) set rd=1 else set rd=0
- Set less than unsigned: \texttt{sltu rd, rs, rt} : if \( rs < rt \) set rd=1 else set rd=0

I-Format versions: compare register and constant, put result into 2\textsuperscript{nd} register
- Set less than immediate (signed): \texttt{slti rd, rs, imm} : if \( rs < \text{imm} \) set rd=1 else set rd=0
- Set less than unsigned immediate: \texttt{sltui rd, rs, imm} : if \( rs < \text{imm} \) set rd=1 else set rd=0

The immediate value, (imm), is 16-bits and is sign-extended to 32 bits before comparison.

Use \texttt{beq} or \texttt{bne} against reg \$0 to test result register rd after set.
MIPS ‘for loop’ example

Setting the elements of an array to zero

Data declarations:

unsigned i;
int array[10];

N.B. C creates the space for both these automatically
no new required.

for (i=0; i<10; i++) {
    array[i] = 0;
}

10/7/2012 GC03 Mips Code Examples
MIPS ‘for loop’ example

Let the variable i be stored in register $4
Let ‘int array’ start at address $12345678_{16}
Each integer occupies 4 addresses
Use $8 and $9 for temporary storage of intermediate values

\[
\begin{align*}
\text{i=0} & \\
\text{add} & \quad \text{set } $4=0 \rightarrow i \\
\text{loop:} & \\
\text{slti} & \quad \text{set } $8=1 \text{ if } $4 < 10 \text{ otherwise } $8=0 \\
\text{beq} & \quad \text{if } $8=0 \ (4>=10) \text{ branch to end label} \\
\text{lui} & \quad : $8 \leftarrow 0x12340000 \\
\text{ori} & \quad : $8 \leftarrow $8 | 0x5678 : $8 =0x12345678 \\
\text{sll} & \quad : $9 \leftarrow $4 \ll 2 : $9 \leftarrow i*4 \\
\text{add} & \quad : \text{form address of array}[i] \text{ in } $8 \\
\text{sw} & \quad : \text{store 32-bits of zero from } $0 \text{ into array}[i] \\
\text{i++} & \\
\text{addui} & \quad : i++ \\
\text{beq} & \quad : \text{branch to label loop - always branches}
\end{align*}
\]
**MIPS ‘for loop’ example**

Setting the elements of an array to zero, but using pointers to memory addresses!

Data declarations (C code – *NOT Java*!!!):

```c
unsigned i ;
int array[10] ;
int *ap ;
```

Variable ‘*ap*’ is of type ‘pointer to integer’ and will hold an address (a pointer in C)

```c
ap = array ; // put the address of array into ap
for (i=0; i<10; i++) {
    *ap = 0 ; // store 0 in the location pointed to by ap
    ap++ ; // increment the address in ap by 4
    ap++ ; // ap now points at the next element of array
}
```

10/7/2012

GC03 Mips Code Examples
**MIPS ‘for loop’ example**

Let the variable $i$ be stored in register $4$, and variable $ap$ in $6$
Let ‘array’ of integers be stored at addresses $12345678_{16}$-$1234569F_{16}$

Use $8$ for temporary storage

- **lui $6$, 0x1234** : $6 \leftarrow 0x12340000$
- **ori $6$, $6$, 0x5678** : $6 \leftarrow 6 \mid 0x5678$ : $6 = 0x12345678$

$i=0$

- **add $4$, $0$, $0$$** : set $4=0 : 0 \rightarrow i$

**loop:**

- **slti $8$, $4$, 10** : set $8=1$ if $4 < 10$ otherwise $0$
- **beq $8$, $0$, end** : if $8=0$ ($4\geq10$) branch to **end** label

*ap=0*

- **sw $0$, 0($6$$** : store 32-bits of zero in $0$ into array[$i$]

**ap++**

- **addui $6$, $6$, 4** : ap++; add4 to $6$ to point to array[$i+1$]

**i++**

- **addui $4$, $4$, 1** : i++; increment loop variable

**beq** $0$, $0$, **loop** : branch to label **loop** - always branches

**end:**
Other instructions that change the PC:

jump register : \texttt{jr \ rs} : \texttt{pc} \leftarrow \texttt{rs} : \text{register contents into \texttt{pc}}

\text{Register value must be multiple of 4 (or processor stops)}
\text{pc can be set to anywhere in memory (greater range than branches).}
This is used to perform function return, e.g. \texttt{jr} $31,$

\textit{N.B. Jumps can go a greater distance than branches.}
\textit{However jumps are never conditional unlike branches.}
\textit{Both are therefore necessary.}

jump and link register : \texttt{jalr \ rs, rd} : \texttt{rd} \leftarrow \texttt{pc} ; \texttt{pc} \leftarrow \texttt{rs}

\text{pc saved to register \texttt{rd} and then \texttt{rs} written into \texttt{pc}}

\text{Used for function (method) calls to any\textit{where in the address space}.}
Function (Method or Subroutine) Call

Some lines of program

Function call

0x0001AB2C ____________
0x0001AB30 ____________
0x0001AB34 lui $1, 0x04 ; $1 <- 0x00040000
0x0001AB38 ori $1, 0x5678 ; $1 <- 0x00045678
0x0001AB3C jalr $1, $31 ; pc -> $31, $1->PC

Code of method

Function return

Have to be sure that $31 has the value store by the jalr when the jr is executed!
Jump Instructions - \textit{J Format}

\begin{tabular}{|c|c|}
\hline
6 Bits & 26 Bits \\
\textit{op} & \textit{target} \\
\hline
\end{tabular}

Jump to target: \texttt{j target} : \texttt{pc}[\text{bits 27:0}] \leftarrow \texttt{target}^*4

Jump and link target: \texttt{jal target}

\hspace{1cm} \texttt{register 31} \leftarrow \text{contents of pc} \ ; \texttt{pc}[\text{bits 27:0}] \leftarrow \texttt{target}^*4

In both cases lower 28 bits of PC register are

\hspace{1cm} \text{loaded with (26 bits of target field} \times 4)

\texttt{jal} is a method call instruction saving the PC before changing it.

\textit{Detail: pc is always an integer multiple of 4: therefore value stored in target field of instruction for j and jal is target address divided by 4, i.e. least 2 bits are dropped, since they are always 00.}

\textbf{Note: the upper 4-bits of PC are unchanged by these instructions.}