Allowable Instructions

When writing MIPS assembly, the only instructions that you are allowed to use (so far) are:

- `add`, `addi`, `sub`
- `beq`, `bne`, `j`
- `slt`, `slti`
- `and`, `andi`, `or`, `ori`, `nor`, `nori`, `xor`, `xori`
- `sll`, `srl`, `sra`
- `lw`, `lh`, `lb`, `sw`, `sh`, `sb`
- `la`
- `syscall`
- `mult`, `div`, `mfhi`, `mflo`

While MIPS has many other useful instructions (and the assembler recognizes many pseudo-instructions), do not use them! We want you to learn the **fundamentals** of how assembly language works - you can use fancy tricks after this class is over.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>The First Page</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Inequality Checks</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>MIPS Code 1</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>MIPS Code 2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Reading MIPS</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>100</strong></td>
<td></td>
</tr>
</tbody>
</table>
1. (a) (5 points) When we declare a string in MIPS, we don’t need a length variable. Why not? How do we find the end of the string?

Solution: All strings have a null terminator (a 0 byte) at the end.

(b) (5 points) We showed, in class, that a MUX can be implemented using a simple sum-of-products. Give the sum-of-products to implement a 2-input MUX.

- Use the variable C to represent the control bit.
- Use the variable X as the 0 input to the MUX.
- Use the variable Y as the 1 input to the MUX.

Solution:

\[ C \cdot X + CY \]

(c) (2 points) The I-format instructions (like ADDI) can’t hold a full 32-bits of data in their constant field - but the ALU requires 32 bit inputs. How is the constant field converted to something large enough?

Solution: sign extension

(d) (3 points) How many bits can an ADDI instruction store in its constant?

Solution: 16

(e) (5 points) Suppose that you have the following array:

```c
int arr[100];
```

Assume that the register $t8$ has been initialized to hold an index into the array. Write a code snippet which does three things:

- Gets the address of the array.
- Reads element [3] into register $s0$.
- Reads element [t8] into register $s1$. (Don’t error-check the index, just use it.)

Solution:

```assembly
la $t0, arr
lw $s0, 12($t0)
sll $s1, $t8,2
add $s1, $t0,$s1
lw $s1, 0($s1)
```
2. Suppose that you have two variables, stored in the registers \$s0, \$s1. Implement the conditional branches shown below. Use no more than 2 instructions per part.

(a) (2 points)
\[
\text{if (s0 \neq s1)}
\]
\[
\text{goto GO\_HERE;}
\]

Solution:
\[
\text{bne } \$s0,\$s1, \text{GO\_HERE}
\]

(b) (6 points)
\[
\text{if (s0 > s1)}
\]
\[
\text{goto GO\_HERE;}
\]

Solution:
\[
\text{slt } \$t0, \$s1,\$s0
\]
\[
\text{bne } \$t0,\$zero, \text{GO\_HERE}
\]

(c) (6 points)
\[
\text{if (s0 \leq s1)}
\]
\[
\text{goto GO\_HERE;}
\]

Solution:
\[
\text{slt } \$t0, \$s1,\$s0
\]
\[
\text{beq } \$t0,\$zero, \text{GO\_HERE}
\]

(d) (6 points)
\[
\text{if (s0 \geq s1)}
\]
\[
\text{goto GO\_HERE;}
\]

Solution:
\[
\text{slt } \$t0, \$s0,\$s1
\]
\[
\text{beq } \$t0,\$zero, \text{GO\_HERE}
\]
3. (25 points) Convert the following C code into MIPS assembly language.

- Use \texttt{sX} registers for all variables that have names in the C program.
- Use \texttt{tX} registers for all temporary values.

```c
int limit = ... ; // s6 - this is set by previous code
int val  = 1;    // choose a register to hold this
int count = 0;   // and also this

while (val < limit)
{
    val = val * 3;
    count++;
}

printf("%d", count);
```

NOTE: You're allowed to use \texttt{mult}, if you know how to use it correctly. But doing add/shift is probably easier!

<table>
<thead>
<tr>
<th>Solution:</th>
</tr>
</thead>
<tbody>
<tr>
<td>addi $s0, $zero,1  # val = 1</td>
</tr>
<tr>
<td>addi $s1, $zero,0  # count = 1</td>
</tr>
</tbody>
</table>

**LOOP:**

```assembly
slt $t0, $s0,$s6   # t0 = (val < limit)  
beq $t0,$zero, DONE # if (val >= limit) break  
add $t0, $s0,$s0   # t0 = val*2  
add $s0, $t0,$s0   # val = val*3  
addi $s1, $s1,1    # count++  
j LOOP  
```

**DONE:**

```assembly
addi $v0, $zero,1  # print_int(count)  
add $a0, $s1,$zero  
syscall  
```
4. (15 points) (Same instructions as the previous page.)

int asdf = ... ; // s0 - this is set by previous code
int jkl = ... ; // s1 - this is set by previous code
int qwerty = ... ; // s2 - this is set by previous code
int uiop = ... ; // s3 - this is set by previous code

int tmp; // allocate a register for this
if (asdf == jkl || qwerty == uiop)
    tmp = 1;
else
    tmp = asdf+qwerty;
printf("%d", tmp);

Solution:

    # NOTE: tmp will be in $s4

    beq $s0,$s1, TRUE  # if (asdf == jkl ) go into true block
    beq $s2,$s3, TRUE  # if (qwerty == uiop) go into true block
    j FALSE

TRUE:
    addi $s4, $zero,1  # tmp = 1
    J DONE

FALSE:
    add $s4, $s0,$s2  # tmp = asdf+qwerty

DONE:
    addi $v0, $zero,1  # print_int(tmp)
    add $a0, $s4,$zero
    syscall
5. Assume that we have the following variables, stored in the following registers:

- $s2  \text{ elway}
- $s4  \text{ montana}
- $s6  \text{ kosar}
- $s7  \text{ brady}

Read the following code snippets. Then write one or two lines of code (C or pseudocode) for each, which indicates what the assembly language does. When the MIPS writes to a register, you can use names like $sX to represent the variable that it is writing to, like this:

\[
\text{s3} = \text{foo+bar}; 
\]

(a) (5 points) \hspace{1cm} \text{add } s0, zero, s2

**Solution:** $s0 = \text{elway}$

(b) (5 points)

\begin{verbatim}
  add $t0, s2, s4  
  add $t1, s6, s7  
  add $t2, t0, t1  
  sub $s1, zero, t2
\end{verbatim}

**Solution:** $s1 = -(\text{elway+montana+kosar+brady})$

(c) (5 points)

\begin{verbatim}
  addi $v0, zero, 1  
  add $a0, zero, s7  
  syscall  
  addi $v0, zero, 11  
  add $a0, zero, 'n'  
  syscall
\end{verbatim}

**Solution:** printf("%d\n", \text{brady});

(d) (5 points)

\begin{verbatim}
  add $t8, s2, s4  
  slt $t7, t8, s6  
  beq $t7, zero, AFTER  
  sub $s0, s6, s7
\end{verbatim}
\[\text{AFTER:}\]

**Solution:**

\begin{verbatim}
  if (elway+montana < kosar)  
    s0 = kosar-brady;
\end{verbatim}