Solutions

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short Answer</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Conversions and Approximations</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Binary Addition</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>MIPS Assembly</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
1. (a) (4 points) What is the name of the process that we use to convert a 16-bit signed integer to the same value, but encoded as a 32-bit signed integer?

**Solution:** sign extension

Next, explain how we do this. What if the number is positive, does that matter? (NOTE: You are not required to explain why this works.)

**Solution:** You copy the MSB of the small number into all of the new bits. It doesn’t matter if it’s positive; if it’s positive, you copy zeroes, and if it’s negative, you copy ones.

(b) (4 points) Take the 2’s complement of the signed binary number 1000 1110. Show your work. (NOTE: You are not required to convert either number to decimal.)

**Solution:**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1110</td>
</tr>
<tr>
<td>0111</td>
<td>0001</td>
</tr>
<tr>
<td>0111</td>
<td>0010</td>
</tr>
</tbody>
</table>

1000 1110 original
0111 0001 bitwise negation
0111 0010 +1

(c) (4 points) Consider a single byte. If it is an **unsigned** integer, what are the maximum and minimum values that it can hold?

**Solution:** 0, 255

If it is a **signed** integer, What are the maximum and minimum values that it can hold?

**Solution:** -128, 127
(d) (4 points) Consider a 12-bit unsigned integer. Of course, the LSB is worth 1, and the next bit is worth 2. What is the MSB worth?

**Solution:** \(2^{11} = 2048\)

**Instructor’s Note:** Both the exponential form and the integer will be accepted.

If the 12-bit number is signed, what is the MSB worth?

**Solution:** \(-2^{11} = -2048\)

(e) (4 points) How many registers are there in a MIPS processor, and how many bits are in each one?

**Solution:** 32 registers, 32 bits each

2. (a) (5 points) Convert the following unsigned binary number to decimal: \(0101 \ 1101\). Show your work.

**Solution:** \(2^6 + 2^4 + 2^3 + 2^2 + 2^0\)
\(64 + 16 + 8 + 4 + 1\)
93

**Instructor’s Note:** Students may show either the first or second line for full credit; it is not required to show both.

(b) (5 points) Convert the following 16-bit number to both octal and hexadecimal:
\(1010 \ 1100 \ 0011 \ 1000\)

**Solution:** \(1 \ 010 \ 110 \ 000 \ 111 \ 000_2 = 126070_8\)
0xac38
(c) (5 points) Convert the decimal integer -427 to a 16-bit signed integer. Write the number in hexadecimal.

**Solution:** We'll convert 427, and then negate it:

\[ 427 = 256 + 171 \]
\[ 427 = 256 + 128 + 43 \]
\[ 427 = 256 + 128 + 32 + 11 \]
\[ 427 = 256 + 128 + 8 + 2 + 1 \]
\[ 427 = 2^8 + 2^7 + 2^5 + 2^3 + 2^1 + 2^0 \]
\[ 427 = 0000 \ 0001 \ 1010 \ 1011_2 \]

We now do 2's complement:

<table>
<thead>
<tr>
<th>Original</th>
<th>Bitwise Negation</th>
<th>+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0001 1010 1011</td>
<td>1111 1110 0101 0100</td>
<td>1111 1110 0101 0101</td>
</tr>
</tbody>
</table>

Hex: 0xfe55

(d) (5 points) Convert each of the powers-of-2 below to an approximate value. (That is, your answer should be something like “32 thousand,” not 32,768.)

\[ 2^{23} = \underline{8\text{million}} \]
\[ 2^{16} = \underline{64\text{thousand}} \]
\[ 2^{38} = \underline{256\text{billion}} \]
\[ 2^7 = \underline{128} \]
\[ 2^{42} = \underline{4\text{trillion}} \]
3. (a) (15 points) For the signed 16-bit numbers
\[
\begin{align*}
a &= 1010\ 1110\ 1001\ 1001 \\
b &= 1110\ 1011\ 1111\ 0110
\end{align*}
\]
calculate \(a + b\). Do all of your work in binary; do not convert any number to decimal. Then, state whether or not overflow occurred and explain your answer.

\[
\begin{array}{c}
1 \ 111 \ 1111 \ 111 \\
1010\ 1110\ 1001\ 1001 \\
+\ 1110\ 1011\ 1111\ 0110 \\
\hline
1001\ 1010\ 1000\ 1111
\end{array}
\]
No overflow. We added a negative to a negative, and got a negative result.

(b) (15 points) Now, calculate \(a - b\) for the following two numbers. You do not have to say anything about overflow.
\[
\begin{align*}
a &= 0101\ 0000\ 0011\ 0010 \\
b &= 0110\ 0111\ 1000\ 1010
\end{align*}
\]

\[
\text{Solution: Take the 2's complement of } b, \text{ then add:}
\begin{align*}
1001\ 1000\ 0111\ 0101 & \quad \text{(bitwise negation of the 2nd number)} \\
1001\ 1000\ 0111\ 0110 & \quad \text{(plus one)} \\
\hline
1 & 111 \ 11 \\
0101\ 0000\ 0011\ 0010 \\
+\ 1001\ 1000\ 0111\ 0110 \\
\hline
1110\ 1000\ 1010\ 1000
\end{align*}
\]
4. This question assumes some MIPS code (on the last page of this exam). The code sets up memory locations red, green, blue, black, white, dave. The code then loads the values of some of these variables into the indicated MIPS registers. In answering these questions, you can assume this code has already been executed, and that the value of some of the variables are already in the indicated registers.

**Special Limitations:**

- Each question is independent of the other questions - that is, assume that the program has started over from scratch each time.
- You may need to read from memory - but do not write to memory unless specifically instructed.

See the last page for the list of allowable instructions.

**NOTE:** You are not required to comment your code - but if you do, we may be able to offer more partial credit.

(a) (15 points) Calculate the value \((2*red) - green + white\), and store the result into $s7.

**Solution:**

```
add $t0, $s6,$s6          # t0 = 2*red
sub $t0, $t0,$s1          # t0 = 2*red - green
la $t1, white            # t1 = &white
lw $t1, 0($t1)            # t1 = white
add $s7, $t0,t1           # s7 = 2*red - green + white
```

(b) (15 points) First, read the variable black, add dave to it, and store it back to memory. Then do the same with white.

**Solution:**

```
la $t0, black             # t0 = &black
lw $t1, 0($t0)            # t1 = black
add $t1, $t1,$s2          # t1 = black+dave
sw $t1, 0($t0)            # black += dave (in memory)

la $t0, white             # t0 = &white
lw $t1, 0($t0)            # t1 = white
add $t1, $t1,$s2          # t1 = white+dave
sw $t1, 0($t0)            # white += dave (in memory)
```
# values are hidden so that you can't hardcode the answers!
.data
red: .word xxx
green: .word xxx
blue: .word xxx
black: .word xxx
white: .word xxx
dave: .word xxx
.text
main:
    # set $s1 = green
    la $s1, green
    lw $s1, 0($s1)

    # set $s2 = dave
    la $s2, dave
    lw $s2, 0($s2)

    # set $s6 = red
    la $s6, red
    lw $s6, 0($s6)

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

- add, addi, sub
- and, andi, or, ori, nor, nori, xor, xori
- lw, lh, lb, sw, sh, sb
- la
- syscall

(Later, we'll add lots more.)
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.