1 Assembly basics

1.1 Assembling an instruction: immediate format

Immediate-format instructions are assembled similarly to register-format instructions, except that the i value is a 16-bit twos complement number, and the opcode comes at the start of the instruction rather than at its end. For example, if we want to assemble \texttt{lw \$31, 23\texttt{\$11}}

- First, we need to convert 11 into a 5-bit binary word: 01011. This is \(s\).
- Next, we need to convert 31 into a 5-bit binary word: 11111. This is \(t\).
- Next, we need to convert 23 into a 16-bit binary word: 0000000000010111. This is \(i\).
- Next, we need to substitute the values for \(s\), \(t\), and \(i\) into the template for \texttt{lw}, in the same way that we did for register-format instructions. We end up with: 1000 1101 0111 1111 0000 0000 0001 0111
- Finally, we need to rewrite this as hexadecimal. Looking up a table or just remembering the bit patterns we end up with \texttt{8D7F0017}.

1.2 Exercise

Assemble the following program by first writing out its binary representation, then converting it to a hexadecimal representation.

\[
\text{\texttt{slt \$6, \$1, \$5}} \\
\text{\texttt{beq \$6, \$0, 1}}
\]
2 Assembly Language Programming

Problem 1 - A simple loop in MIPS
Recall that the factorial, $n!$, of $n$ is given as follows:

$$
0! = 1 \\
1! = 1 \\
n! = n \cdot (n - 1)! \\
n > 0
$$

Write a MIPS program which takes a non-negative integer $n$ in $\$1$ and stores $n!$ in $\$3$.

Problem 2 - More loops in MIPS
Recall that the Fibonacci sequence can be defined as follows:

$$
f_0 = 0 \\
f_1 = 1 \\
f_{n+2} = f_{n+1} + f_n \\
n \geq 0
$$

Write a MIPS program which takes a non-negative integer $n$ in $\$1$ and stores $f_n$ in $\$3$.

Problem 3 - Arrays in MIPS
Thus far we’ve only written programs which accept two integers as arguments, but with mips.array we can also write programs which manipulate arrays. Write a MIPS program which accepts the address of an array in $\$1$ and its length in $\$2$ and stores the product of the numbers in the array in $\$3$. 


Problem 4 - Basic I/O in MIPS

Recall that you can read from stdin and write to stdout by loading from or storing to addresses 0xffff0004 and 0xffff000c respectively. Note that EOF is represented by −1, and otherwise a single byte will be read or written at a time.

Write a MIPS program which reads in two characters from stdin (you may assume EOF is not encountered) and prints out the character 1 if the first is less than the second, or 0 otherwise. It should then print a newline.