Review Problem 7

- Sometimes it can be useful to have a program loop infinitely. We can do that, regardless of location, by the instruction:
- LOOP: B LOOP
- Convert this instruction to machine code

\[
\text{New PC} = \text{Old PC} + 4 \times \text{B, Add 26}
\]
Conversion example

Compute the sum of the values 0...N-1

ADD X1, X31, X31
ADD X2, X31, X31
B TEST +3
TOP:
ADD X1, X1, X2
ADDI X2, X2, #1
TEST:
SUBS X31, X2, X0
B.LT TOP-3
END:
Assembly & Machine Language

Assembly

Simple instruction
Mnemonics for humans
(Almost) 1-to-1 relationship w/machine language

Machine Language

Numeric

Fixed format: simple encode & decode

Directly control the HW
Computer Arithmetic

Readings: 3.1-3.3, A.5

Review binary numbers, 2's complement

Develop Arithmetic Logic Units (ALUs) to perform CPU functions.

Introduce shifters, multipliers, etc.
Binary Numbers

Decimal: $469 = 4 \times 10^2 + 6 \times 10^1 + 9 \times 10^0$

Binary: $01101 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (13)_{10}$

Example: $0111010101 = (?)_{10}$

\[1 + 4 + 16 + 64 + 128 + 256\]

\[5 + 80 + 384\]

\[65 + 469\]

\[469_{10}\]
2’s Complement Numbers

Positive numbers & zero have leading 0, negative have leading 1

Negation: Flip all bits and add 1

\[ \overline{13} = \underline{10010} + 1 = \underline{10011}_2 \]

Ex: \[-(01101)_2 = \]

To interpret numbers, convert to positive version, then convert:

\[ 11010 = \overline{1010} = \underline{1010} + 1 = \underline{10101} = 5_{10} \]

\[ 01100 = +4 + \underline{8} = +12_{10} \]
Sign Extension

Conversion of n-bit to (n+m)-bit 2's complement: replicate the sign bit

\[ b_3b_2b_1b_0 = b_3b_3b_3b_2b_1b_0 = b_3b_3b_3b_3b_3b_3b_3b_3b_3b_2b_1b_0 \]

Ex - Convert to 8-bit:  \( 01101 = (13)_{10} \)

\[ 11101 = (-3)_{10} \]

Interpret it (decode)

\[ = -(-11111101) \]
\[ = -(0000010 + 1) \]
\[ = -3 \]
Arithmetic Operations

Decimal:

\[
\begin{array}{c}
5 7 8 9 2 \\
+ 7 8 9 5 6 \\
\hline
3 6 8 4 8 \\
\end{array}
\]

Binary:

\[
\begin{array}{c}
1 0 1 0 1 1 1 \\
+ 0 1 0 0 1 0 1 \\
\hline
1 0 1 0 0 1 1 0 \\
\end{array}
\]

Binary:

\[
\begin{array}{c}
1 0 1 0 0 1 1 0 \\
- 0 0 0 1 0 1 1 1 \\
\hline
1 0 1 0 0 1 1 0 \\
\end{array}
\]

\[
\begin{array}{c}
1 0 1 0 0 1 1 0 \\
+ 1 1 1 0 1 0 0 \\
\hline
1 0 0 0 1 1 1 \\
\end{array}
\]

"Add one"
Overflows

Operations can create a number too large for the number of bits
n-bit 2’s complement can hold \(-2^{(n-1)} \ldots 2^{(n-1)}-1\)
Can detect overflow in addition when highest bit has carry-in ≠ carry-out
\((\text{carry-in}) \oplus (\text{carry-out}) = 1\)

\[
\begin{array}{c}
5 \\
3 \\
-8
\end{array}
\begin{array}{c}
0101 \\
0011
\end{array}
\begin{array}{c}
-7 \\
-2 \\
7
\end{array}
\begin{array}{c}
1001 \\
1110
\end{array}
\text{Overflow}
\]

\[
\begin{array}{c}
5 \\
2 \\
7
\end{array}
\begin{array}{c}
0101 \\
0010
\end{array}
\begin{array}{c}
-3 \\
-5 \\
-8
\end{array}
\begin{array}{c}
1101 \\
1011
\end{array}
\text{No overflow}
\]
## Full Adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C\text{in}</th>
<th>C\text{out}</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Multi-Bit Addition

\[ A_3 A_2 A_1 A_0 + B_3 B_2 B_1 B_0 \]

\[ S_3 S_2 S_1 S_0 \]
Adder/Subtractor

\[
0 \quad A + B = \quad = A + B + 0 \\
1 \quad A - B = A + (\overline{B}) = A + \overline{B} + 1
\]
Debugging Complex Circuits

Complex circuits require careful debugging
Rip up and retry?
Ex. Circuit to see if \(A+B>7\).
module fullAdd (Cout, S, A, B, Cin);
    output Cout, S; input A, B, Cin;

    assign Cout = (A&B) | (A&Cin) | (B&Cin);
    assign S = A^B^Cin;
endmodule

module halfAdd (Cout, S, A, B);
    output Cout, S; input A, B;

    fullAdd a1(.Cout, .S, .A, .B, .Cin);
endmodule

module greaterThan7 (Out, A, B);

    halfAdd pos0(.Cout(C[0]), .S(S[0]), .A(A[0]), .B(B[0]));
    fullAdd pos1(.Cout(C[1]), .S(S[1]), .A(A[1]), .B(B[1]), .C(C[0]));
    fullAdd pos3(.Cout(C[3]), .S(Out), .A(0), .B(0), .C(C[2]));
endmodule
Debugging Approach

Test all behaviors.
   All combinations of inputs for small circuits, subcircuits.

Identify any incorrect behaviors.

Examine inputs and outputs to find earliest place where value is wrong.
   Typically, trace backwards from bad outputs, forward from inputs.
   Look at values at intermediate points in circuit.

DO NOT RIP UP, DEBUG!