CS 241 – Week 1 Tutorial Solutions

Binary and assembly basics

Winter 2018

1 Binary

1.1 Binary basics

1. What is the meaning of the binary word 00101101?

   Since computers represent everything in binary, it's hard to say what is meant by any given binary word unless the person who wrote it tells us how to interpret it. We could give different meanings to this word if we view it as a character, as a number, as a bitfield (array of booleans), as a note in a song, or many other things.

2. Rewrite the number 1011_2 in decimal.

   Recall that if 1011 were instead a decimal number, we could write it as $1*10^3+0*10^2+1*10^1+1*10^0$. We can use the same idea to convert binary to decimal, except using 2 instead of 10. So we get $1*2^3+0*2^2+1*2^1+1*2^0=11$.

1.2 Converting positive decimal numbers to binary

Convert the following numbers into unsigned 8-bit binary:

1. 35

   - $35/2 = 17$ remainder 1
   - $17/2 = 8$ remainder 1
   - $8/2 = 4$ remainder 0
   - $4/2 = 2$ remainder 0
   - $2/2 = 1$ remainder 0
   - $1/2 = 0$ remainder 1

   So $35_{10} = 00100011_2$. Working backwards, $100011_2 = 2^5 + 2^1 + 2^0 = 32 + 2 + 1 = 35$.

2. 216

   - $216/2 = 108$ remainder 0
   - $108/2 = 54$ remainder 0
   - $54/2 = 27$ remainder 0
   - $27/2 = 13$ remainder 1

- $13/2 = 6$ remainder 1
- $6/2 = 3$ remainder 0
- $3/2 = 1$ remainder 1
- $1/2 = 0$ remainder 1

So $216_{10} = 11011000_2$. Working backwards, $11011000_2 = 2^7 + 2^6 + 2^4 + 2^3 = 128 + 64 + 16 + 8 = 216$.

1.3 Converting negative decimal numbers to binary

Try the following in 8-bit binary:

1. Encode -12 using the first technique.

   First we need to encode 12 using the above method:

   - $12/2 = 6$ remainder 0
   - $6/2 = 3$ remainder 0
   - $3/2 = 1$ remainder 1
   - $1/2 = 0$ remainder 1

   So $12_{10} = 00001100_2$. Flipping all the bits we get 11110011, and finally adding 1 we get $-12_{10} = 11110100_2$.

2. Encode -123 using the second technique.

   Since $b = 8$ we need to encode $2^8 - 123 = 133$. Using the above method:

   - $133/2 = 66$ remainder 1
   - $66/2 = 33$ remainder 0
   - $33/2 = 16$ remainder 1
   - $16/2 = 8$ remainder 0
   - $8/2 = 4$ remainder 0
   - $4/2 = 2$ remainder 0
   - $2/2 = 1$ remainder 0
   - $1/2 = 0$ remainder 1

   So $-123_{10} = 133_{10} = 10000101_2$. Remember that both this number and the previous one only hold in 8 bits!

3. Compute the 8-bit two's complement binary encodings of 15, -15, and $--15$ to convince yourself that $15 = --15$.

4. Convince yourself that, for example, $--15 = 15$ by either two's complement technique listed above. What is $--128$?

   To find $--15$ we'll use the first method. First encode 15 to get:

   - $15/2 = 7$ remainder 1
   - $7/2 = 3$ remainder 1
   - $3/2 = 1$ remainder 1

- $1/2 = 0$ remainder 1

So $15_{10} = 00001111_2$. Flipping the bits we get 11110000, and adding 1 we get 11110001. Now flipping the bits again we get 00001110, and adding 1 we get 00001111, which is $--15$. As we would expect, this is identical to the binary representation of 15.

5. Repeat the above exercise using 128 instead of 15. What do you notice?

   As $128 = 2^7$ we we can just observe that $128_{10} = 10000000_2$. Then $-128 = 01111111 + 1 = 10000000$. So $--128 = 10000000$ as well. It shouldn't come as a surprise that $128 = --128$, but -128 also shares their representation!

   In general, in b bits the number 2^{b-1} is equal to itself when negated. When looking at the definition of a negative number we see why this might be: $2^{b-1} + 2^{b-1} = 2^b \equiv 0 \mod 2^b$. By convention we decide this number is negative, but this means we have one negative number with no positive equivalent!

2 Assembly basics

2.1 Assembling assembly language

Assemble the following program by first writing out its binary representation, then converting it to a hexadecimal representation, and finally using `cs241.wordasm` to make it executable. What does it do? Run it with `mips.twoints` to verify it behaves as you'd expect.

```
lis $5
.word 7
slt $6, $1, $5
beq $6, $0, 1
add $1, $1, $5
jr $31
```

Solution:

To encode `lis $5` we need to encode 5 in 5-bit unsigned binary and then insert it into the appropriate place in the lis instruction pattern described on the MIPS reference sheet. $5 = 00101$, so inserting $ddddd = 00101$ we get 00000000000000000010100000010100. After doing the same for all instructions (`jr $31` is left as an exercise since it's an assignment question):

```
0000 0000 0000 0000 0010 1000 0001 0100
0000 0000 0000 0000 0000 0000 0000 0111
0000 0000 0010 0101 0011 0000 0010 1010
0001 0000 1100 0000 0000 0000 0000 0001
0000 0000 0010 0101 0000 1000 0010 0000
```

Next we need to convert them into hex. Every group of 4 binary digits corresponds to one hex digit: you might find it easiest to use a conversion chart, to convert to from binary to decimal and then decimal to hex, or to memorize the bit patterns. Regardless of which method you choose you should end up with:

```
00002814
00000007
0025302A
10C00001
00250820
```

At this point (after adding the `jr $31` instruction) we're ready to put .words in front of each hexidecimal number and compile and run our program!