1 Assembly basics

• Recall that binary data has no meaning on its own: we need to be told (or come up with) an interpretation for it. Give five different possible ways we could interpret 0101:
  1. The decimal number 0101.
  2. The 4-bit unsigned number 5.
  3. The 4-bit signed number 5.
  4. The hex digit 0x5.
  5. An array of 4 boolean.
  6. An 4-bit color value.
  7. This list is by no means comprehensive: these are just a few ideas.

• Convert the 8-bit binary number 01101001 into decimal.
  Solution: 105.

• Convert the following numbers into unsigned 8-bit binary:
  1. 35
     - 35/2 = 17 remainder 1
     - 17/2 = 17 remainder 1
     - 8/2 = 17 remainder 0
     - 4/2 = 17 remainder 0
     - 2/2 = 17 remainder 0
     - 1/2 = 17 remainder 1
     So 35_{10} = 00100011_2. Working backwards, 100011_2 = 25 + 21 + 20 = 32 + 2 + 1 = 35. Could also use the subtraction method, subtracting 32, 2, and 1.
  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, $11011000_2 = 27 + 26 + 24 + 23 = 128 + 64 + 16 + 8 = 216$. Could also use the subtraction method, subtracting $128$, $64$, $16$, and $8$.

### 1.1 Two’s Complement

- Try the following 8-bit binary:

  1. $12$ in binary is $1100_2$, so with 8 bits, it is $00001100_2$. Flipping all the bits we get $11110100_2$, and finally adding 1 we get $-12_{10} = 11110100_2$.
  
  2. Since $b = 8$ we need to encode $2^8 - 123 = 133$. Using the above method: $-123_{10} = 133_{10} = 10000101_2$. Remember that both this number and the previous one hold in 8 bits!
  
  3. Well use the first method to get $-15$. First, notice that $15$ is 1 less than $2^4$, so $15_{10} = 00001111_2$. Flipping the bits we get $11110000_2$, and adding 1 we get $11110001_2$. Now flipping the bits again we get $00001110_2$, and adding 1 we get $00001111_2$, which is 15!

  Now lets try this with $128$. $128 = 2^7$ we can just observe that $128_{10} = 10000000_2$. Then $-128 = 01111111_2 + 1 = 10000000$. So $- - 128 = 10000000$ as well! $- - 128 = 128$ as we would like, but its also $-128$!

### 2 Assembly Basics

#### 2.1 Exercise

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.

```plaintext
lis $5
.word 7
add $1, $1, $5
sub $1, $5, $0
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 00000000000000000000001010001010100. After doing the same for all instructions (jr $31 is left as an exercise since its an assignment question):

- 0000 0000 0000 0000 0010 1000 0001 0100
- 0000 0000 0000 0000 0000 0000 0000 0111
- 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:

```
.word 0x2814
.word 0x7
.word 0x250820
.word 0xA00822
```

At this point (after adding the `jr $31` instruction) were ready to compile and run our program!