Computer Systems Organization
CSCI-UA.0201 (002), Fall 2015

1 Exercises

(a) Suppose we want to add two numbers on an x86 processor. These numbers are 86 and -94. We know the answer is -8, but we want to see how these numbers are stored in memory, added, then stored again. The number 86 is stored at address `0x0`, and the number -94 is stored at the address `0x4`. The result, -8, will then be stored at address `0x8`. Draw the memory before the addition operation and after the addition operation in hexadecimal. Rather than convert -8 into 32-bit signed integer directly, calculate -8 via the addition of 86 and -94 as 32-bit signed integers.

2 Solutions

(a) i. convert 86 to binary. If we decompose this into powers of two, we get 64 + 16 + 4 + 2, or $2^6 + 2^4 + 2^2 + 2^1$, so our binary representation is
`0000 0000 0000 0000 0000 0000 0101 0110`.
In order to convert this to hexadecimal, we note that binary is base 2, and hexadecimal is base 16. So 4 binary digits represent 1 hex digit, and vice versa (which is why the binary is split up into groups of 4). The first 6 groups all represent 0, so our first 6 hex digits are 0, the 7th group is `0101`, or 5 in decimal, which is 5 in hexadecimal. The 8th group is `0110`, or 6 in decimal, which is 6 in hexadecimal. Then our hexadecimal representation is `0x00000056`.

ii. convert -94 to binary. Notice how the number is negative, so we first calculate what 94 is and take the 2s complement. Decomposing 94 into powers of two yields 64 + 16 + 8 + 4 + 2, or $2^6 + 2^4 + 2^3 + 2^2 + 2^1$, so our binary representation is
`0000 0000 0000 0000 0000 0000 0101 1110`
(as an aside, converting this to hex, we get `0x0000005e`). Now we take the 2s complement (this involves taking the 1s complement, or flipping each bit, then adding 1). First by flipping each bit we get
`1111 1111 1111 1111 1111 1111 1010 0001`,
then we add 1 to get
`1111 1111 1111 1111 1111 1111 1010 0010`.
Converting to this hex, we get `0xffffffa2`.

iii. store these in memory. Recall that x86 processors are little Endian, so the least significant byte gets the lowest address, and a byte is defined as 8 bits, or 2 hex digits. So our memory layout looks as follows (values on top, addresses on bottom):

```
0x56 0x00 0x00 0x00        0xa2 0xff 0xff 0xff
0x0                        0x4                        0x8
```

iv. add the two numbers. (aside: notice how we're basically doing 86 - 94, but this problem is worded as if we were to do 86 + (-94).) We can straight up add the two numbers in binary (or in hexadecimal).

```
  0000 0000 0000 0000 0000 0000 0101 0110
+ 1111 1111 1111 1111 1111 1111 1010 0010
  1111 1111 1111 1111 1111 1111 1111 1000
```

Or represented in hexadecimal, `0xfffffff8`. Because the first bit of the binary string is 1, we know this must be a negative number. So we must flip the bits and add 1 to "undo" the 2s complement (note how we aren't subtracting a 1). We get our flipped binary string to be
`0000 0000 0000 0000 0000 0000 0000 0111`,
and adding 1 yields
`0000 0000 0000 0000 0000 0000 0000 1000`,
or `0x00000008`, or 8 in decimal. So we know our final answer is -8.

v. store the third value, -8, in memory (values on top, addresses on bottom):

```
0x56 0x00 0x00 0x00        0xa2 0xff 0xff 0xff        0xf8 0xff 0xff 0xff
0x0                        0x4                        0x8
```