Alexander Miller
Assembly 1 Report

Part 1.

- Address of median:

0x10010070

- Full text of assembly instructions:

```
lui	$1,0x00001001
ori	$8, $1, 0x00000070
```

- Instruction Encodings:

0x3c011001
0x34280070
[looks like these are masked away to get the assembly instructions]

Part 2.

- String: “median: “
- First 4 characters: {‘m’, ‘e’, ‘d’, ‘i’}
- Associated ASCII: {0x6D, 0x65, 0x64, 0x69}
- Address of String: 0x10010000
- Word Value: 0x6964656d

Part 3.

I noticed that the address used in the basic assembly instruction appeared to be a mask of the instruction. This raises the question of whether that’s how the assembler associates instructions with its data. It also raises the question of if this function is invertible; in other words, it appears you can navigate from the instruction to its data via a mask, but can you navigate from data to an associated instruction with an inverse operation? (Hunch is no, but don’t know).

I also learned that the bytes in a word are stored in what appears to be reverse order (not sure why this would be, or if this is event the correct conclusion). This raises a related question, if you wanted to access the first byte in a word, would you go to “address + 3” or “address + 1.”

To answer this question, I wrote the following toy code:

```
.data
MSG:	.asciiz	"abcd"

.text
# load the first byte of the string
```

```
   la      $t0, MSG
   lb      $t1, 1($t0)
# print that byte
   add    $a0, $t1, $zero
   addi   $v0, $zero, 11
   syscall
```

It printed out "b". So this implies that when addressing specific bytes, do so from 0 onward, in the logical order one would expect.