# Reference

1. [en.wikipedia.org/wiki/Code\_page\_437](https://en.wikipedia.org/wiki/Code_page_437)

We use this page to view the list of available ASCII characters we could use in our program. It is very useful for us as we wanted to tabulate and display data like we planned in our proposal.

1. [www.tutorialspoint.com/assembly\_programming/assembly\_arithmetic\_instructions.htm](http://www.tutorialspoint.com/assembly_programming/assembly_arithmetic_instructions.htm)

This page simplifies how Arithmetic Operations in assembly works. Which enable us to understand how Assembly do multiplication and division. It was also this page that made us realize 8086 doesn’t have EAX, EBX. ECX, EDX. Initially, we wanted to give up calculating 32bit values. But then we tried to stimulate 32bit division/multiplication referring to this page (for quite some time) and we succeeded (to a certain degree).

1. [www.tutorialspoint.com/microprocessor/microprocessor\_8086\_instruction\_sets.htm](http://www.tutorialspoint.com/microprocessor/microprocessor_8086_instruction_sets.htm)

Every 8086 instructions can be found here. CALL enabled us to make short jumps whilst executing more than 127 bytes of code, also to ease the strain on our eyes. ADC enabled us to preform 32bit addition and adding the carry (if there’s any) at the same time.

# Simulated 32bit division/multiplication on a 16bit System

Now this is interesting,

# Pointers

In our second semester AACS1084 Programming Concept and Design II, we were quite familiar with C. I remembered Chapter 4.9.1, Pointers. At first, I was not quite understood what it was for other than storing user input.

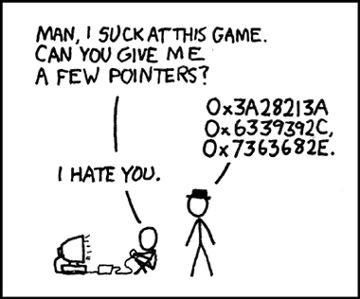
But it’s different when we go to Assembly, we even dove deeper into addressing, Segment: Offset, Direct, Indirect. And we would use them when we want to access arrays. Arrays in C, when we want to get the next value we would just add 1 to the index.

Now here’s the problem, array is different in Assembly. Increment of the index depends on the variable size. Array of 10 bytes would require index increment by 1. Array of 10 words would require index increment by 2. For example, I have to print out 7 days, I have an array of string, which stores Monday until Sunday. Each string has equal length of 10 characters, I would have to have an index increment by 10 as each character allocates one byte.

Then we have to print an array of strings, which contains apple names, right next to the days. Each apple names are 15 character in length. It was quite easy as we just have another index which increment by 15 each loop. Two indexes, two arrays, problem solved. Or so I thought.

Next is to print out an array of words, at this rate we knew we are going to either run out of index to play with or its going to be hundreds of lines to change index values from memory whilst updating each and every single index. Spaghetti code, is what I wanted to prevent. And thus, we reached into a conclusion, we take the addresses we need and store them.

Since address are 2 bytes in size, we would then store the higher address into one array, then lower address into another. Both are array of bytes, which increment by one. So, when we refer to the 4th word in an array, I would just have to take the 4th higher address and the 4th lower address and then we would find the 4th word in an array. Sounds hideous but this alone could save a lot of code in a loop, which is very critical as loop has a limited number of bytes it could jump.

With that being said, here’s a joke I’ve found on r/ProgrammerHumor.