The Assembly Language

Lesson 1 - Generalities

Some History

- Oldest non-machine language
- Allows a more readable method of writing programs than writing in binary bit patterns

Comparison with High-Level languages

- Assembly is close to a one to one correspondence between symbolic instructions and executable machine codes
- Includes directives to the assembler, the linker, for organizing data and macros
- HL languages are abstract, see this example

A comparison example

- The simpler and more raw way of writting a simple character in C language would be:
  - ```
    Void show_char(char c)
    {
            write (1, c, 1);
    }
    ```
- In assembly, it would be 10 times longer, since you have to initialize the character, the outputs, and start the display
    ```

More comparisons with HL languages

- Assembly is much more harder to program
- Much more detailed
- High quality assembly can be a source of higher speed (2 to 20 times faster is farily common, though you can have increases of hundreds of time faster)
- Assembly gives access to key machine features for low-level routines (OS kernel, microkernel, device drivers, machine control...)
- Development time increases 10 to 100 times faster with HL languages

Availibility

- Assemblers are avilable for just about every processor
- Native assemblers produce object code on the same hardware that the object code will run on
- Cross assemblers produce object code on different hardware that the object code will run on

Kinds of Processors

- Complex Instruction Set Computers (CISC)
- Reduced Instruction Set Computers (RISC)
- Hybrid
- Special purpose

Data Representation

- Bit / byte / word / longword
- Sometimes we find halfwords, doublewords or quadwords
- Some processors require data to be aligned
- The motorolla 68000 has 8 bit bytes, 16 bit bytes, 32 bit longwords and 64 bit quadwords
- Big/Little Endian

Numeric Systems

- Binary (%)
- Hexadecimal ($)
- Today's hint!

Arithmetic Operations - Adding

- In decimal:

		11
first number		127
second number	+	96
result		223

- Same with binary

	%01111111
+	%01100000
=	%11011111

Addition

- The same would happen in hexadecimal

```
        1
       30  => $1E
   +   52  => $34
   =   82  => $52
```

Adding Overflow

- Check this example:

	111
160	%10100000
+100	%01100100
= 4	%00000100

- So we have an overflow!

Overflow with multiplication

- You should know that $(a^x)*(a^y)=a^{(x+y)}$
- So if we multiply a number n1 with p bits to another n2 with q bits, we'll have
  - $N1max = 2^p - 1$ and $N2max = 2^q - 1$
  - So $N1max*N2max = (2^p - 1)(2^q - 1)$
  - Finally the biggest value would be:
    $$2^{(p + q)} - (2^p + 2^q)$$
- We will then need at least p+q bits

The negative numbers

- We can work with signed or unsigned numbers
- Usually a negative number is represented by taking the 2s complement of its positive representation:
- For example: the number 4 (%00000100)
  - We take the 1s complement: %11111011
  - We add 1: %11111011 + %1 = %11111100
  - So -4 is represented by %11111100

Arithmetic Operation with negative numbers

- Adding: Everything works the same way, but we ignore the overflow

11111100
- 4 => %11111100
+5 => %00000101
=1 => %00000001

Logical Operations

- AND
- OR
- EOR
- NOT

Shiftings

- A binary shift left is like multiplying by 2
- N binary shifts left is like multiplying by 2^N
- A binary shift right is like dividing by 2
- N binary shifts right is like dividing by 2^n

Why using shiftings?

- When doing a multiplication
  - 80386 -> 26 clock cycles
  - 80486 -> 26 clock cycles
  - Pentium -> 11 clock cycles
- When dividing
  - 80386 -> 38 clock cycles
  - 80486 -> 40 clock cycles
  - Pentium -> 40 clock cycles
- When shifting
  - 80386 -> 2 clock cycles
  - 80486 -> 3 clock cycles
  - Pentium -> 1 clock cycles

How to multiply without a multiplication?!

- Imagine you want to multiply by 320
- We can write 320 = 256+64 = 2^8+2^6
- If you want to have B=A*320=A*(256+64)
  B=A*256+A*64=A*2^8+A*2^6
- We can simply use:
  - 1 cycle to shift left A 8 times
  - 1 cycle to memorize the result in B
  - 1 cycle to shit right A 2 times (8-2=6)
  - 1 cycle to add the result to B
  - = 4 cylces on a pentium, instead of 11