Chapter 3- Assembly is the key to the crack

It is extremely important to have a background in Assembly language if you want to get far cracking. To tell you the truth I used to be into programming the Apple II and IIgs in Assembly language. When I moved over to the Mac, I found out everything was C and Pascal, which disturbed me. C and Pascal allow you to program without having a clue what is really going on. I don't like the idea of it at all. Most people think cracking is something for people who know how to program, but the truth is, I haven't written a damn thing for the Mac as far as applications go, because I don't know how. I could probably write something in assembly, but I just don't have a nice assembler like Merlin on the Mac.

Some day, I recommend you buy a good 68000 reference manual so you can learn the processor. That's what I did but Apple II assembly knowledge helped me out alot. Assembly languange on the Mac is a bit more complicated than that of the Apple II; on the Mac memory is moved not loaded or stored.

On the Apple II and IIgs there are three registers you would normally use to store data in. The A or Accumulator, the X Register, and the Y Register. The Mac has 16 registers that can be used in this manner. They are D0-D7 which are 8 data registers for storing data, they are all capable of holding 32-bits worth of data. You can access or change these the low 8-bits, the low 16-bits or all 32-bits, changing the low byte or word has no affect on the remaining unchanged portion. There are also 8 address registers from A0-A7 but A7 is usually used as a stack pointer for the 68020 and is also known as the SP in this situation. The address registers are basically the same as the data registers but they can only be accessed using all 32-bits. Changing the low word of an address register replicates the bit 15 in bits 16-31. This is called sign extension, which converts a two's complement 16-bit quantity into an equivalent 32-bit quantity.

Another register on the Mac worth noting is the program counter (PC) register. This holds the address of the next instruction to be executed, and is very useful in tracing code.

There are many more opcodes in the 68000 instruction set then there are in the 6502 or 65c816, too many to list here. In the crack examples I will explain everything very thoroughly so you can get an idea of what is going on, and understand it.

MacsBug and DisAsm both list Assembler in the same manner. They list the address on the left hand side, the opcode and effected address or values in the middle and the hex values on the right hand side. This is how I will lit my code, but I will add in descriptions

on what it is the code is actually doing. Here is an example of what I mean:

```
611F04:    Move.L     D0,D1       ;2200       : this moves the contents of
                                                    D0 into D1
```

That's just an example, but it is very common to see code like that. The thing that is cool about MacsBug is that it lists all of the Address and data registers on the left part of the screen from top to bottom along with other system registers.

Another important aspect of 68000 and higher processors is that they use branches the same way the Apple II did, only that there are many more different branch instructions and they are much more powerful. Here are a examples of branch opcodes for the 68000 series processors (used in Macs):

Hex Instr.	Ex.	Meaning
[$60]	$60xx	BRA (BRanch Always). This instruction always branches to an address $xx bytes in front of where the instruction was passed.
[$66]	$66xx	BNE (Branch if Not Equal). This instruction will branch ahead $xx bytes in front of where the instruction was just passed.

There are 16 in all but I would rather not make this publication into an Assembler Reference book. I will put out one of those in the future maybe. I do suggest getting a 68000 quick reference though. On we go.