6800 Programming - Introduction

This section of our manual has been written to help you begin programming your 6800 Computer System. Programming is a complicated subject. This manual describes only the two more basic levels of programming, machine language and assembly language. With the material contained in this manual you should be able to learn the basic principles of programming. Take things slow and reread the material as many times as necessary to understand what is being said. Since trial and error is one of the fastest learning methods, don't be afraid to sit down and experiment with some short programs. Unlike hardware, errors in programming, or software as it is often called; cannot damage anything. So experiment with programs until you understand the material presented here.

The "What Is An Assembler" section of this manual should be mastered as a first step. Most of the material in the Motorola "Microprocessor Programming Manual" assumes an understanding of both assembler and machine language programming. Chapter five of this manual goes into detail describing the 6800 assembler. It is important that you learn it now since all user programs should be written using assembler mnemonics and then hand assembled into machine language. Not doing this will make it impossible for you to look at each program statement and know what it does. Chapter five's material on editing, listing and saving programs applies only to timeshare services.

You should read pages PROG-1 through PROG-5 of the programming section in this notebook next. Pages PROG-1 through PROG-3 describe in detail the various methods of addressing and must be learned. Without knowing which instruction addressing mode to use, writing a program will be virtually impossible. Pages PROG-4 through PROG-5 describe the calculations that are needed when hand assembling programs that contain branch instructions. When a resident assembler program is used with the computer, labels are provided is the source code and the assembler program makes the necessary calculations for you.

Appendix A of the Motorola "Microprocessor Programming Manual" contains all of the assembler mnemonic instructions as well as their hexadecimal machine language equivalents. You should read through this section several times to get familiar with the instructions that are available to you in the 6800 processor. When you start writing your programs you will find this information indispensable.

Appendix B contains the list of assembler directives which are instructions used only for the assembler. They have no function when programming in machine language, however, are nice to know when reading the assembler source listings in the systems manual.

Pages PROG-6 through PROG-21 contain sample programs which may be useful to the reader. Pages PROG-21 through PROG-29 contain some sample PIA polling routines which would be useful to those using parallel interface options.

What is an Assembler?

Throughout this notebook as well as the Motorola Programming Manual you will repeatedly encounter the word "assembler" as well as printouts of its source code listings. To those of you already familiar with assemblers these terms should be easily understood, To others just learning about computers, this can be a confusing subject. Before we can explain the term "assembler" though, you must understand how programs are loaded, stored and executed within the computer's memory.

The SWTPC 6800 Computer System has a read-only-memory (ROM) stored minioperating system with a memory examine and change function which allows the user to enter either programs, or data into the computer's memory from the terminal's keyboard in convenient hexadecimal (base 16) notation. The data is entered from whatever starting location the user chooses and is loaded sequentially with the operating system incrementing the memory address after each location has been loaded.

If we were to look at a listing of the data that was loaded into memory, it might look like this:

A017	2B
A018	FE
A019	A0
A01A	02
A01B	86
A01C	01
A01D	A7
A01E	00
A01F	A1
A020	00
.	.
.	.
.	.
.	.
.	.

The column of numbers on the left is the hexadecimal address at which the hexadecimal data on the right is stored. As it happens, the data loaded into these ten locations is a portion of a program loaded into memory using the memory examine/change function of the mini-operating system. The first location A017 although part of the program is used only for storing data, the rest of the nine addresses starting with address A018 contain actual program instructions. Before the program is to be started, the program counter must be loaded with the address of the starting byte of the program using the "display contents of MPU registers" function of the mini-operating system. To actually start the program, one uses the "go to user's program" function of the mini-operating system which transfers processor control to the instruction pointed to by the program counter. In this case the instruction is FE which translates to load the index register with the contents of the memory address given in the next two bytes (A002). Since the index register is a two byte register, the least significant byte is filled with the contents of the next sequential address which is A003. For simplicity this and other instructions are abbreviated to three letter terms called mnemonics. The mnemonic for this instruction is LDX for LoaD indeX register. The particular type of addressing used here is referred to as "extended" and is described in detail later in the literature. So now we can say the instruction FE is the same thing as LDX, extended. This is defined as a three byte instruction since a total of three memory bytes are used for the entire instruction. The data is stored in locations A018, A019, and A01A. The program counter was incremented by one as each of the preceding memory locations were processed and at the completion of the last byte of the instruction, was left pointing to the next instruction location at address A01B which is 86. The instruction 86 means to LoaD Accumulator A with the contents of the memory location immediately following the instruction which is 01. This is referred to as the immediate mode of addressing and is

described in detail later in the literature. Our mnemonic for this instruction is LDA A, immediate which stands for Loan Accumulator A and it is a two byte instruction. At the completion of this instruction, our program counter is left pointing to memory address AOID whose contents are an A7 which is a STore Accumulator A indexed by 0, instruction. The mnemonic here is STA A, indexed.

This means the contents of accumulator A are stored at the address contained within the index register plus the index value, which is contained in the memory location immediately following the instruction which in this case is zero. This instruction like the one following it is a two byte instruction. The next instruction is an A1 whose mnemonic is CMP A, indexed, which translates to CoMPare accumulator A to the memory location pointed to by the address contained in the index register plus zero. And so the program continues.

It's probably obvious by now that having to write a program in two digit hexadecimal form usually referred to as machine language can really be hard to interpret unless you are able to memorize the mnemonic translation for all of the hexadecimal instructions. Wouldn't it be easy if you could write your program using the easy to remember mnemonics and let the computer translate them to their machine language equivalents to be loaded into memory? Well this is what the assembler does and in addition allows the programmer to use labels and comments with statements and add assembler directives which allocate memory storage locations and start the program in the selected memory address just to mention a few. The assembler also detects and prints out detected errors in the source program. So as you can see the assembler is simply a program which allows the programmer to save time and simplify his program writing by using labels, simple mnemonic commands, and assembler directives. The assembler program itself is several thousand bytes in length and is usually loaded from a tape reader. It is far too long to be typed in manually.

The mnemonic written program to be assembled is generally entered from a tape that has been generated by what is called an editor. The editor is a program used to generate a new or modify an already existent source program. The editor allows the user to enter, delete, modify or insert data to a source file. When the programmer is satisfied with the accuracy of the file, a tape is generated which may then be assembled by the assembler. If there are errors or if you choose to modify the program, the editor/assembler sequence may be repeated as often as necessary. Like the assembler, the editor is several thousand bytes in length and is far too long to be typed in manually.

Since the SWTPC editor/assembler software package will not be available until early 1976, many users will have to enter their programs in machine language. It is suggested that you write your program in assembler form using mnemonics as detailed later in the literature and then at the same time jot down the hexadecimal machine code just to the left of the instructions. You should be able to fill in all of the machine codes as you go along with the exception of forward referenced branch and jump instructions which you can go back and fill in after you have completed writing the program. This hand assembled method was used when writing the diagnostic programs in the software section of this notebook. You may even find that it is quicker to hand assemble those programs less than fifty words or so than to use the editor/assembler package.