

The World of Microcontrollers: A Beginning Guide to the HC11 Microkit

By Jason Guilford
150 Heller Dr #364
Santa Cruz, CA 95064
(831) 502-9619
jguil4d@cats.ucsc.edu

TABLE OF CONTENTS

Introduction	2
Section 1. What is a Microcontroller?	2
Section 2. Where are Microcontrollers used?	3
Section 3. What is the HC11 Microkit?	4
The HC11 Microkit and Programming Environment	4
The HC11 Registers	6
Beginning to Code	7
Syntax	9
Section 5. Conclusion	11
Glossary	11
References	12

Introduction

You've had four, five, maybe six weeks of your first computer engineering course here at the University of California at Santa Cruz. So far you've learned the basics of the MIPS Assembly Language. Now it is time to change gears to the HC11 microkit. In the labs for the rest of this course you will be programming in the assembly language of this microcontroller. In this tutorial I will describe to you the advantages of microcontrollers in Section 1. In Section 2 I will give you examples of where microcontrollers are used in everyday life. Section 3 will cover examples of code for the HC11, as well as syntax and a brief list of the operations that the HC11 language can do. In Section 5 I will conclude this manual.

Section 1. What is a Microcontroller?

Microcontrollers are task-specific chips that are typically very cheap to build and quite reliable in the field. Most microcontrollers are single-chip-integrated systems. Typically such a microcontroller would have many on-chip peripheral features. Most microcontrollers have the ability to control program and data memory and input-output ports. *IEEE Spectrum* has a very good description of microcontrollers on page 36 in their October issue of 1996 [3].

Reliability in microcontrollers is well documented, as my professor, Richard Hughey [2], impressed upon my class. Such reliability is achieved by limiting end-user data-input. By "end-user" I refer to the final purchaser of a device with such a microcontroller.

If someone were to suggest to you that you let computers affect your everyday life, you'd probably shriek at the very idea. No one in his or her right mind would ever want to let a PC run the daily lives of normal people. How often does the computer on your desk crash? How many of you and your peers, parents, and teachers know what is meant when someone says "three-finger salute" or "blue screen of death"?

Yet we already allow computers to control objects in our lives, with hardly any ill effects. The reason so many of these devices are running smoothly is that they use microcontrollers.

By limiting end-user data-input, the microcontroller's operating software is less likely to develop bugs. A virus is virtually impossible to put into a microcontroller, as end-users don't usually have the means to enter complex data structures or programs into the manufactured product. Many older microcontrollers are not capable of multi-processing [3], which means that once a particular program is activated the chip will not relinquish control to any other program. Thus even if a new program could be created and loaded, somehow, into the microprocessor's memory there would be no way to get it to execute.

Microcontrollers take many different forms [2]. The great majority of microcontrollers through the early eighties were 4-bit microcontrollers. Microcontrollers are typically talked of in terms of number of bits. This is a measure of the size of the data that can be computed by a particular microcontroller. Common sizes of microcontrollers today are 4-bit, 8-bit, 16-bit, and 32-bit. The most common size from 1991 through 1996 is 8-bit [3]. Eight-bit microcontrollers continue through today as the most common on the market [2].

Section 2. Where are Microcontrollers used?

The reason I find microcontrollers fascinating is that they have been and continue to be such an important part of the electronics industry. Over the past decade and more microcontrollers have been creeping into our daily lives. Figure 1 shows how microcontrollers increased in popularity from 1991 through 1996.

In today's world, microcontrollers are used in just about every electronic object in the household and place of business. Just about the only common object in the house that does not have a microcontroller in it is the light bulb [3]. In fifteen years or so even that may not be the case.

The reason microcontrollers have become so common is that they are more than merely reliable. By adding a small computer to many devices it is possible to increase efficiency or safety, or any number of other features.

Timing devices are now composed almost entirely of microcontrollers. This has made them unbelievably accurate. They are also cheap, and much more reliable. A digital watch today, which has no moving parts, is almost impossible to break through normal use. It is easy to adapt digital watches to extreme environments such as the deep sea or vacuum.

Telephones and other personal communication devices also use microcontrollers. With such devices it is possible to have wireless telephones and cellular phones, each capable of maintaining a connection between the phone unit and some sort of base station. These phones can also encrypt data as it leaves and decrypt it as it comes in.

Televisions and stereos use microcontrollers. Neither is the mess of tubes that was synonymous with televisions and radios. As a result, both produce better quality picture and sound, have more features, and weigh less per unit volume.

All kinds of transportation systems use microcontrollers. Cars use them in fuel injection systems, brakes, airbags, and just about any other piece of equipment. Airplanes are going to a "fly-by-wire" control system. This is a complex computer interface between the controls that the pilot uses and the control surfaces of the plane. Such interfaces are controlled by microcontrollers.

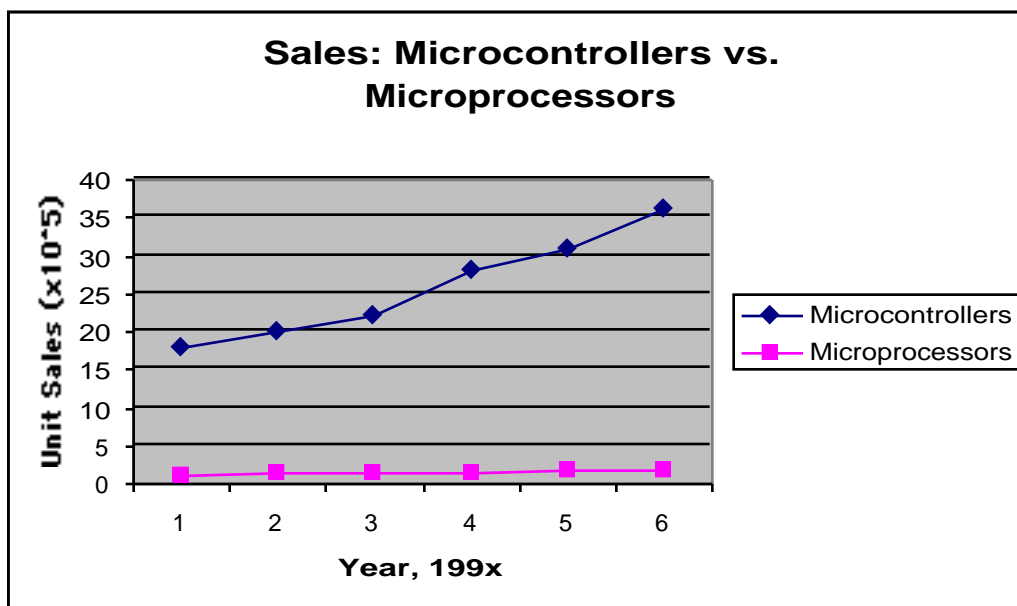


Figure 1. Sales of microcontrollers vs. microprocessors, showing microcontrollers to be almost ten times more common in daily use than microprocessors.

Modern traffic lights are controlled by microcontrollers. Emergency phones by the highways of America have solar panels, batteries, and a microcontroller circuit to operate in remote areas. In e-mail of February 29, 2000, Stephen Petersen mentioned to me that microcontrollers are ubiquitous in American life [5]. And they are. Try counting the number of devices with microcontrollers in your home. I have at least seventeen of them in my dorm room.

Section 3. What is the HC11 Microkit?

Through the rest of this quarter you will be learning about microcontrollers in lab. The HC11 chip is a derivative of the Motorola 6800 series of microcontrollers. The 6800 series is one of the first successful eight-bit microcontrollers to enter the market [5]. You will learn in the lab course you are taking that it is quite powerful for only an eight-bit chip.

The HC11 Microkit and Programming Environment

The microkits that you use have one HC11 microcontroller chip on a printed circuit board, PCB. There are eight red light emitting diodes, LED's, and eight binary switches on the PCB. The board has on it two buttons, labeled RESET and INTERRUPT. The RESET button temporarily cancels power to the box. The INTERRUPT button will be introduced later in the quarter as one of your lab assignments.

The top portion of the box has a two-line dot matrix LCD. There are 16 spaces visible on each line of the LCD. The lines do not wrap. If you print a character string of

more than 16 characters in length to the LCD, it will not conclude on the next line. Only the first sixteen characters will be shown.

The boxes are built to be portable. They get power from a 9-volt power source, which can be found in the lab. A serial cable is used to download the compiled program onto the box. At present, March 2000, both the serial cable and power supply can be found in the Social Sciences 2, Baskin Engineering 213, and Ming Ong Computer labs.

The compiler software can be found on the WindowsNT machines in each of these labs. The software is stored on the F: / drive. The path to get to the programming environment is F:/Class Folders/Computer Engineering/CE 12c. The file *Win IDE PEMicro - Start* is the programming environment. Another file, called *Sim11a - Start*, is very useful. You will need this other file when you do work on the simulator. *Sim11a* also has a complete list of the HC11 instruction set.

The programming environment has its own quirks. It has automatic alignment on indents. When it compiles the *.asm file it creates a host of other files. The one that needs to be downloaded to your box is the *.s19 file. To compile you can either left-click on the third button from the right or press the F4 key on your keyboard. Compiling the program also saves it.

There is one more piece of software, used for downloading the program to the box. This is found under the path Start Menu/Internet tools/_Unix - Telnet. The last choice in this folder is labeled Other, as shown in Figure 2. Open this file and select "serial." Connect the box to the power supply and serial cable. The power supply plugs into a round port under the right side of the PCB, and the serial cable plugs into a ribbon that comes off the top of the PCB and has a blue free end. Once you are plugged in and have the downloader open, press the RESET button. If you have the correct communication port selected you will see the screen refresh itself. If the screen does not refresh itself, then the PC you are using is hooked up to a different communication port than you have selected. Restart the downloader and try a different port. Some labs are on the first port, COM1, and some are on the second, COM2.

If you have problems with any of the software in your lab section, talk to your lab tutor. That's what they are there for. And read the news group. The professor, TA, and tutors can and will answer your questions.

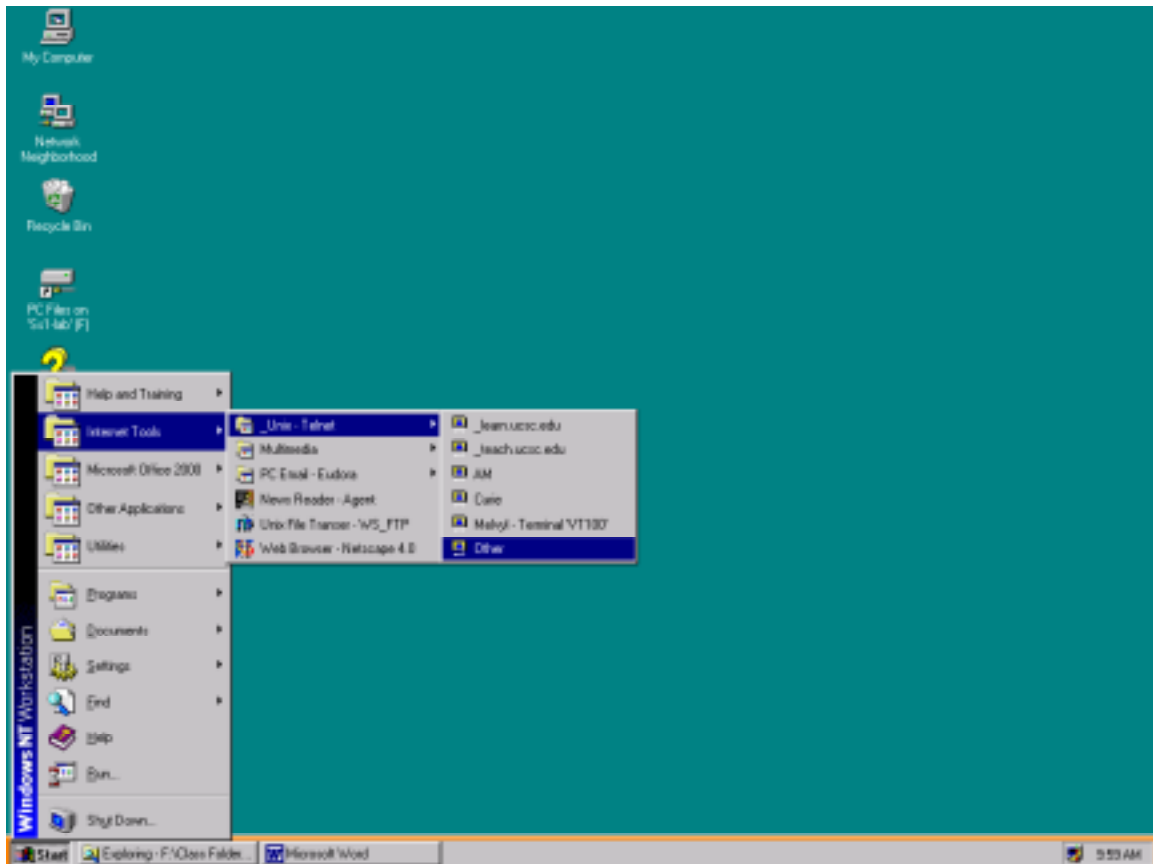


Figure 2. This shows the path to get to the download software for the box.

HC11 Registers

The HC11 has only four registers that the user can put information into by hand. These are registers A, B, X, and Y. A and B are both eight-bit accumulator registers. These are the main registers of the HC11 chip. Almost all of the computations that you perform will be done in these registers. X and Y are sixteen-bit registers, and are used more often for storing information. A good diagram of the registers can be found in the *HC11 Reference Manual*, on page 1-4 [4].

There are three other registers that you will be able to affect, but no data can be stored in them directly. Two of these are the stack pointer, SP, and program counter, PC, registers, which are also sixteen bits each. The last is the condition code register, CCR, which is only eight bits. You can access the stack pointer by pushing and pulling from the system stack. You can change the program counter by branches and jumps. The CCR is used to record the result of a comparison. All of these will be covered in greater detail later.

There is one more register, called the double accumulator, or D register. The D register is another sixteen-bit register. The D register is the combination of A and B, which is why the double accumulator is shown as it is in Figure 3. D can be used in most

of the ways that A and B can be. It is important to remember how this works. A value stored in D will destroy any values stored in A or B, inclusive. The least-significant eight bits of D are B, and the most-significant eight bits of D are A.

Beginning to Code

The first program you do in the HC11 language will run on a simulator. This first program has already been written for you, so I'm not going to say much about it. If you want to play with this sample program all you will need is the instruction set. This can be found in Appendix A of the *HC11 Reference Manual* and in the *HC11 Programming Reference Guide*.

Starting with your second program, you will be writing a program to run on the actual lab microkit, which I will refer to as the box. For this, the syntax changes slightly from what you saw in the program that ran on the simulator. The basic layout of the code is as follows:

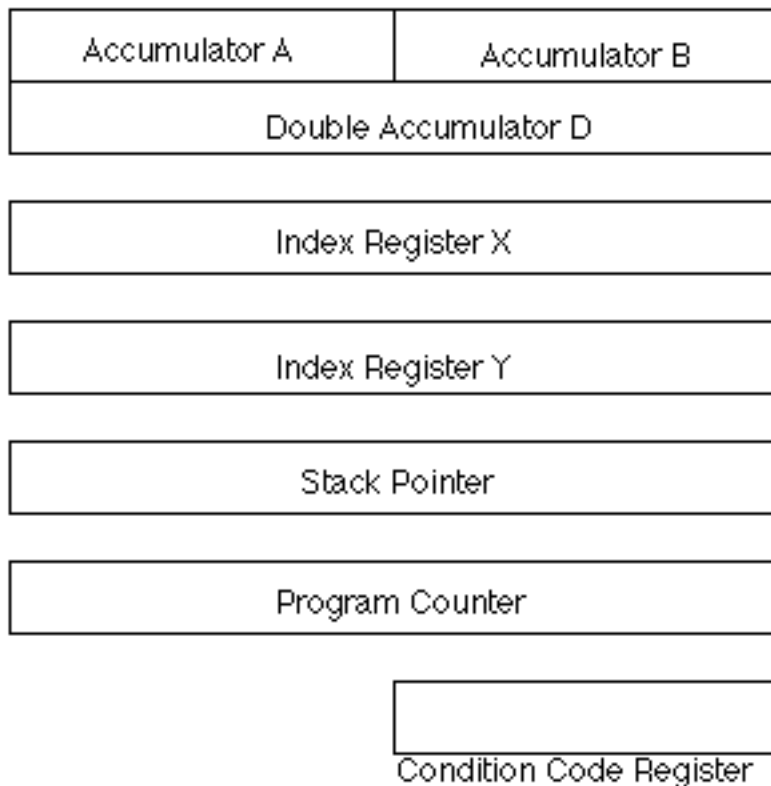


Figure 3. This shows the registers on the HC11 chip. Notice that D is touching A and B. This is because D is a false register. D is really just A and B used as a single register. All the long registers are sixteen bits, and the short ones are eight bits.

```

$SETNOT simulate
$include "v2_17.asm"

*****
*this section includes data structures such as character
*strings and variables in memory, like the .data section in
*MAL.
*****

main:

*****
*this section is where the code goes, as in the .text section
*of MAL. the label "main:" is analogous to the "__start:"
*label in MAL. Anything that is above this label is not
*considered to be part of the code except as it forms a
*character string or variable in memory.
*****

    stop ;the end of code, similar to "done" in MAL.

```

\$SETNOT simulate and \$SET simulate are two compliments of the same thing. One of these must be included at the beginning of the code to tell the compiler what the code is intended for. It should be noted that the compiler is not case sensitive. If you are going to use upper or lower cases, though, it is important to be consistent. I like to use lower case for everything except variable names and calls to subroutines.

For all of the programs that you will write in HC11 there is a file called v2_17.asm which you will need to include. This is similar to `stdio.h` in the C language. Stephen Petersen wrote v2_17.asm. This file contains all of the basic I/O function definitions for the box. It contains a number of other useful functions, such as wait commands and interrupt protocol.

The first label that the code has in it is always going to be `main`. This is where the code will begin execution. As mentioned above, the compiler for HC11 is not case sensitive. So you could write `main` as `MAIN` if you prefer. Just be consistent with your label names. The HC11 syntax requires a few things of the labels. You cannot put a

label anywhere except the far left-hand margin. From experience Alex Carey [1] and I have discovered that labels must also start with an upper- or lower-case letter.

Syntax

The HC11 manuals that are printed by Motorola do not cover some of the syntax I will show you in this manual. This is because I have tailored this manual to the design environment used here at UCSC. You should be aware, for future reference, that each compiler has its own syntax [1]. An example of the syntax used here at UCSC is shown below in Figure 4.

```
*****
*A Strange Program, by Jason Guilford*
*****
$SETNOT simulate
$include "v2_17.asm"

MYTEXT:  DW  `Hello.          `
          DB  NULL

main:
    ldx #MYTEXT      ;loads address of MYTEXT into register X.
    ldaa #$1         ;tells which line of the LCD to print to.
    jsr LCDLINE      ;jump subroutine, LCDLINE is defined in
                    ;v2_17.asm.  it prints to the LCD.

    ldd #$1          ;load double accumulator D with the
                    ;hexadecimal value 1.
    xgdx             ;exchange X with D.

main2:              ;another label
    xgdx             ;notice than not all of the instructions
                    ;in HC11 take arguments.
    cpd #$10         ;compare D with hexadecimal 10, or which
                    ;is decimal 16.
    bge end_of_prog ;branch if greater than or equal.  The
                    ;compare is needed before this.
    addd #$1         ;add to D the value 1, store in D.
    xgdx
```

```

    ldaa #$2          ;load hex 2 into A.
    ldab #$3          ;load hex 3 into B.

do_strange_things:
    cmpa #$10         ;compare A with hex 16. Notice the
                      ;syntax is slightly different than when
                      ;comparing D with a value.
    bgt main2         ;branch if greater than.

    staa LEDS         ;store value in A to the LED's.
    tba               ;copy value in B to A.
    adab              ;add A to B and store in A.
    addb #$1          ;add to B the hex value 1, store in B.
    jmp do_strange_things ;a jump command, as in MAL.

end_of_prog:
    stop              ;this is the same as done in MAL.

```

Figure 4. This strange program demonstrates the proper syntax for an HC11 program written using the PEMicro compiler we have here at UCSC.

Starting from the top, this is a strange program. The first two lines we have already discussed. The portion labeled MYTEXT is a character string in memory, declared by DW, and terminated by a NULL character which takes a single byte and is declared by DB. By having sixteen characters in it, MYTEXT will overwrite the entire line of the LCD that it is called on.

Notice that syntax is often quite different from what you saw in MAL. There are no instructions in HC11 that take more than one argument. This is one of the advantages of accumulator registers [2]. Many of the instructions don't take any arguments, as is the case with tba, xgdx, or adab. Some instructions have been broken down into two instructions, as is the case with branches. In HC11, to do a conditional branch you must compare a value in a register to an immediate value, then branch based upon the result of the comparison. It is also possible to compare A to B, but all other registers must be compared with an immediate.

Immediates in HC11 can be input in three different forms. The hexadecimal values begin with the signifier #\$. It is possible to input binary. The signifier for binary is #%. Decimal can be inputted with the signifier #!.

Of these three forms, hexadecimal is the most reliable and convenient. But to use it reliably you must be familiar with the hexadecimal values. Binary is equally useful,

but be careful to input the correct number of bits into the register. Decimal is the easiest to use, but it has problems. No one really knows why, but some of the boxes don't like decimal. This was discovered through many hours of lab testing when I took CMPE 12c.

Stop is a keyword just discovered by one of my associates, Alex Carey [1]. I've never used it myself, but I am told that it causes the box to freeze at the end of the program. I know that if there is nothing to occupy the chip, when the program reaches the end it simply stops executing anything, the registers get reset and the default settings are restored to the LCD. The way I have traditionally solved this problem is to include a loop at the bottom of my code that never stops running, like this:

```
do_nothing_loop:
    jmp do_nothing_loop
```

Section 5. Conclusion

Now you should have the skills necessary to write a basic program in the HC11 language. As the quarter progresses you will probably leave this tutorial behind. To include everything you would need to know through the quarter in this volume is just impossible. With what is presented here you should be ready to rely more heavily on the manuals provided by the Motorola Corporation.

Glossary:

Accumulator Register: a register that does basic arithmetic by adding or subtracting values to itself and storing the new values in itself.

Immediate: an integer constant value. In the HC11 immediate values can be entered as hexadecimal, decimal, or binary.

Subroutine: Like a function in C. These are freestanding sections of code that can be called by any program, even programs that do not specifically include the subroutine in their main file. The subroutine's file must be included, as is the case with `v2_17.asm` for my sample program.

References

1. Carey, Alex. Personal conversation. March 5, 2000.
2. Hughey, Richard. "CMPE12c." UCSC. Santa Cruz, November 1999.
3. Kahn, Ata R. "Workhorses of the electronic era." *IEEE Spectrum*, October, 1996: 36 - 42
4. Motorola, Inc. *HC11 Reference Manual*. USA: Motorola, Inc, 1991.
5. Petersen, Stephen. E-mail to the author. Feb 29, 2000