3.

Hello again. In this third lesson, we reach a critical point... the point of explaining the whys and wherefores of the 6809 microprocessor. You should have spent some serious time getting familiar and comfortable with binary and hexadecimal counting, as well as with the arrangement of ASCII characters. The workbook provides some exercises and self-tests; please complete them before continuing with this lesson. Especially if you're a first-timer to assembly language, that's very important.

There are many general ways in which microprocessors are described and defined: they are smart circuits, they are calculating devices, they are (as I've said) a microscopic nest of electronic switches. Microprocessors are all of these things and more. I'll use several terms interchangeably throughout these lessons — processor, microprocessor (or MPU), central processing unit (or CPU). In your Color Computer, these all mean the same thing: the 6809 microprocessor.

Inside all microprocessors, inside all MPUs, are a number of data holding stations called registers. More about the term register later, but at the heart of a microprocessor is a special calculator register, formally called an arithmetic logic unit, or ALU. The ALU holds one binary word — that is, a certain number of binary digits of information. I'm talking here about the 6809 processor. It accepts data in an eight-bit binary group, called by the tongue-in-cheek name "byte". The word size of the 6809's binary data is the byte — eight bits.

To describe it another way, the 6809 has eight wires connected to it for data. All eight wires become "live" simultaneously, conducting eight binary digits to the processor. This information is one byte.

So it's got this arithmetic logic unit, the ALU, which holds a byte of data. The ALU can then perform simple calculations with that byte of data. The calculations, which I'll get back to in detail shortly, are: addition, subtraction, and multiplication. Also, there are incrementing and

```
Things pick up speed now.
There's lots of new information
coming up, so make sure you've
done all the exercises before
ending this session.

* What does CPU stand for?

Central Processing Unit, the
microprocessor.

* What is the Central Processing
Unit (CPU) in the Color
Computer?

The 6809 is the Color Computer's
CPU.

* What holds the data inside the
microprocessor?

Registers hold data inside the
microprocessor.

* What does ALU mean and what
does it do?

ALU means Arithmetic Logic Unit.
The ALU performs calculations.

* How many bits of information
does the ALU hold?

The ALU holds eight bits.

* What is the computer jargon
for eight binary digits?

A byte is computer jargon for
eight binary digits.
```

The ALU

*** What is a binary word?**

A certain number of binary digits.

*** What is the word size of the 6809 microprocessor?**

The 6809's word size is eight bits.

*** Again, what is the computer jargon for eight bits?**

Eight bits make up a byte.

*** Name three kinds of arithmetic the ALU performs.**

Addition, subtraction, and multiplication.

*** Name four kinds of logical functions the ALU performs.**

AND, OR, NOT and Exclusive OR.

*** Name the three other operations the ALU performs.**

Incrementing, decrementing, and comparison.

*** How many bytes can the ALU hold for doing its work?**

The ALU holds one byte.

*** Name all ten kinds of operations the ALU can perform on one byte.**

Addition, subtraction, multiplication, AND, OR, NOT, Exclusive OR, incrementing, decrementing, and comparison.

*** What is the ALU called in the 6809 processor?**

The ALU is called the accumulator.

*** How many accumulators are there in the 6809 processor?**

The 6809 has two accumulators.

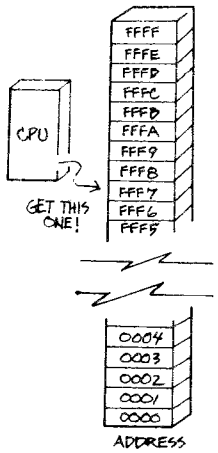
decrementing, which are essentially addition or subtraction by one. The ALU can perform logical operations such as AND, OR, NOT, and EXclusive OR. Finally, it can make a comparison with any other byte of data.

Most of the processor's hard work is done in the ALU. In fact, the 6809 is such an advanced processor that it contains two separate ALUs. Each one can add, subtract, increment, decrement, compare and perform logical operations. Together, they can be used to multiply.

The arithmetic logic units in most MPUs, in most processors, including the 6809, have several descriptive names. They are called accumulators. The ALU is also called an accumulator register. Finally, the 6809's own PAIR of accumulators are labeled A and B. So the arithmetic logic unit, the ALU, the accumulator, and the accumulator register effectively mean the same thing. In the 6809, they mean where the math is done — in A or B.

These A and B accumulators get the information they need by loading it. Loading: that's the term for obtaining data. The accumulators save the results by storing data. Load and store. Get data, save data.

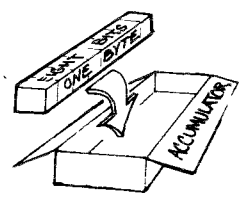
I have to answer several questions at once now, because the actions that they represent are so intertwined. Here are the questions: How does an accumulator load or store data? Where does it load data from, and where does it store it?



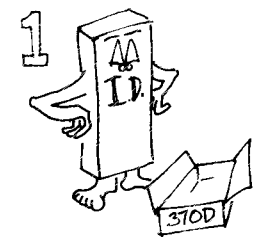
I'll start with the "where". The data the accumulator needs might be inside the microprocessor in another register. The term register is in fact quite general. The A and B registers of the 6809 are the arithmetic logic units, the accumulators. But there are other registers also capable of holding information, though these registers cannot by themselves do any mathematical calculations with the data. Their main purpose is to keep information handy for the accumulators.

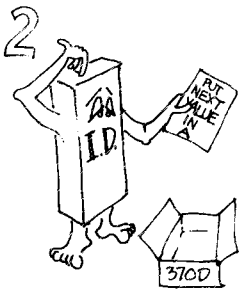
But the most important place the accumulator obtains its information is from memory. Memory is a line of storage locations outside the 6809 processor itself. Each memory location can hold an eight-bit word, a byte.

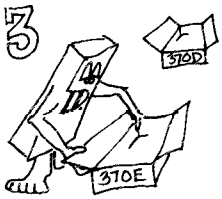
I'll back off from that briefly to tell you *how* an accumulator loads or stores data. It follows the commands of an instruction decoder, that part of the processor which determines the actions the processor is to take. Now here's where my answers get intertwined. The instruction decoder gets its instructions from the same memory that stores data. In other words, when the instruction decoder gets a byte from memory that says "load something into the accumulator", the next byte in memory is that very "something". It all comes from the *same* line of memory.

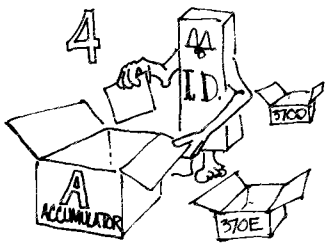


Recall the last lesson. I said that the 256 possible









rearrangements of binary digits represent all the information the processor will ever need — instructions, numbers, ASCII characters, whatever. That's precisely true. I also said that it's the context that determines what the binary pattern means. Context is what assembly language programming is all about: ordering the bytes sos that they turn into a useful program.

So far you know that the processor, the MPU, gets both its instructions and data from memory. How does it distinguish them? That is, how does it understand their context?

To discover the answer, you must know that the memory locations are each uniquely numbered, starting from zero. These identification numbers are called addresses. How many memory addresses a given MPU has available are determined by the number of its address bits. In keeping with its total logical binary nature, the 6809 has 16 address bits. The total number of rearrangements of 16 binary digits, from **0000 0000 0000 0000** to **1111 1111 1111 1111**, is 65,536. It's what you call 64K (since a "K" in computer terms is 1,024).

Get a pencil and paper. Breaking them into groups of four digits, write down **0000 0000 0000 0000**. That's 16 zeros. Now, elsewhere on the paper, write down 16 ones. Also break those into groups of four: **1111 1111 1111 1111**. Above each group of four binary digits, write its equivalent hexadecimal symbol. For the 16 zeros, the hexadecimal value would be: dollar sign **0 0 0 0**. Don't forget that dollar sign; it identifies a hex number. Also write the hex value for 16 ones: dollar sign **F F F F**.

What you have just written is the address range — that is, the number of individual memory locations — available to the 6809 processor. **$0000** running through **$FFFF** are the addresses of the 6809 MPU.

---

* What are the names of the two accumulators in the 6809?

The two accumulators are called A and B.

* What are the ten kinds of operations the A and B accumulators can perform?

Addition, subtraction, multiplication, AND, OR, NOT, Exclusive OR, incrementing, decrementing, and comparison.

* What is the term for obtaining data?

Loading means obtaining data.

* What is the term for saving data?

Storing means saving data.

* What is the word size the 6809's accumulators can hold?

One byte, that is, eight bits.

* Aside from other registers, where do the A and B accumulators get their information?

The A and B accumulators get their information from memory.

* What is the word size of a memory location?

One byte, that is, eight bits.

* How many locations are in the Color Computer memory map?

65,536 memory locations.

* Describe the map size and word size of the 6809 processor's memory.

65,536 locations; each location is one byte in size.

* There are how many bytes in one "K"?

1024 bytes.

* There are how many "K" in 65,536 bytes?

64K.

* What is the number of the first and the last memory location in the Color Computer.

The first memory location is number 0; the last memory location is number 65,535.

* How many binary digits are needed to represent the range 0 to 65,535?

There are 65,536 possible combinations of 16 bits needed to represent the range 0 to 65,535.

* Write the number 0 in 16 binary digits.

The number zero in binary is 0000 0000 0000 0000.

* What is 0000 0000 0000 0000 in hexadecimal?

0000 0000 0000 0000 in hexadecimal is $0000.

* What is the number 65,535 in binary digits? Hint: it is the largest number that can be written using 16 bits.

65,535 in binary is 1111 1111 1111 1111.

* What is 1111 1111 1111 1111 in hexadecimal?

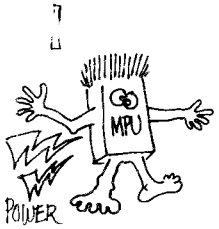
1111 1111 1111 1111 in hexadecimal is $FFFF.

* The 6809 microprocessor has a 64K memory map. How many bytes is 64K?

64K is 65,536 bytes (64K times 1,024 bytes per K)

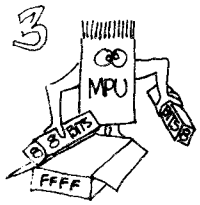
You have just identified the 6809 processor's address range. Knowing now that the 6809's addresses run from **$0000** to **$FFFF**, you are ready to discover how the 6809 MPU distinguishes instructions it performs from the data it uses.

The 6809 goes through a fixed set of electronic actions whenever the power is turned on, or whenever the reset switch is pressed. The processor first does up its internal housekeeping. It requests the contents of memory at address **$FFFE**, and following that, it requests the contents of memory at address **$FFFF**.

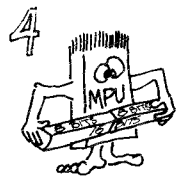


Follow carefully here. The two bytes loaded from memory locations **$FFFE** and **$FFFF** are concatenated — that is, combined end-to-end. A byte is 8 bits; two bytes end-to-end are 16 bits. 16 bits happens to be the same size as the 6809 processor's address . . . something that didn't happen by chance. In fact, those two bytes are used as the address of the memory location containing the very first instruction the microprocessor will follow.

I'll repeat that. When the power is turned on, the 6809 fetches the two bytes stored at memory locations **$FFFE** and **$FFFF**. The processor concatenates them, producing a 16-bit value. That value is used as an address, and at that address is found the first instruction 6809 must execute.

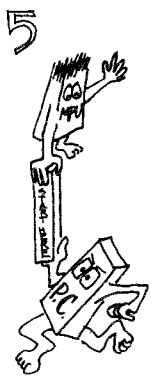


That address is put in a special 16-bit register called the program counter. From that point on, until the power is turned off, the program counter, called the PC, always keeps track of the next instruction the processor is going to follow. If the programmer has done a good job, the computer will begin executing the thousands of instructions that make up its language or operating system.



I think it's time for a summary.

I'm using microprocessor, MPU, processor, central processing unit, and CPU interchangeably. Inside the 6809 MPU are two arithmetic logic units, the ALUs, which each hold a single 8-bit word of data and perform simple calculations on that byte of data. The ALUs, also called accumulators, are identified as the A and B registers.

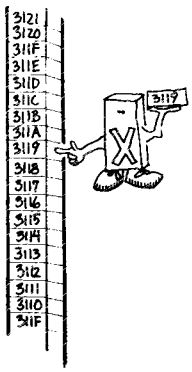


The registers load bytes of data from memory and store bytes of data in memory. There are 65,536 memory locations available to the 6809 MPU, and from them it gets both its instructions and data. The instruction decoder inside the MPU tells it what operations to perform in response to an instruction byte loaded from memory. The program counter register, the PC, keeps track of which instructions are next in line.

If you feel comfortable with this information, please continue with this tape. If any of it's shaky, start this lesson again; you might want to follow along in the text while reviewing the lesson.

What I'm discussing in this lesson is the 6809's architecture. That's the term for the logical organization of the processor. The water's about to get deeper, and I'm going to throw you in, so get ready to swim. Along with your documentation, there is a Motorola data booklet for the MC6809E processor. Find the booklet, and turn to page 5. Data booklets like these are meant for programming and hardware professionals, so much of it will initially appear incomprehensible. That fogginess is a trademark of data sheets.

We'll be concerned with the last two paragraphs on page 4, the first few on page 5, and most importantly Figure 5. Take a moment to locate those.



Look first at Figure 5. So far, you've found out about the program counter (PC) and the A and B accumulators. As you can see, there are actually several more registers.

The X and Y registers are effectively identical. They are called "index registers" because they act sort of like your index finger in a card file, pointing to a specific entry. Remember that registers are special data storage locations inside the processor. Each of these two index registers is 16 bits in size. Because X and Y are 16 bits, they can be used to point to a specific memory location, that is, to be indexed to that 16-bit address. Indexing is its most common function, but not its only use; here's an example of indexing.

Let's say there's a message to be displayed on the screen. I'll point my Y register to the video screen memory, and point my X register to the memory that contains the message. My program can then tell the A accumulator register to get the byte of data from the memory location indexed by X and put it in the memory location indexed by Y. Load A accumulator from memory indexed by X, store A accumulator to memory indexed by Y. The first letter of the message is then displayed. It's like telephone directory assistance. The operator indexes the number, the telephone transmits it, and you index it on your phone pad. If I increment both X and Y after displaying the first letter of the message — that is, if I add one to the present values of X and Y — they will be pointing to the next locations in memory. I can have the program repeat the process of loading from memory indexed by X and storing to memory indexed by Y. That would get the next letter of the message and display it in the next position on the screen. Load A from X-indexed memory, store A to Y-indexed memory.

I've got a program to do that.

* The 6809 microprocessor has a 64K memory map. What is its address range in decimal, in binary, and in hexadecimal.

The map runs from 0 to 65,535 in decimal, from 0000 0000 0000 0000 to 1111 1111 1111 1111 in binary, and from $0000 to $FFFF in hexadecimal.

* A byte of information is eight bits; an address is 16 bits. How many bytes are needed to describe an address.

Two bytes describe an address.

* What part of the processor determines what it must do?

The instruction decoder.

* Where does the instruction decoder get its instructions?

From memory.

* What memory locations does the processor use when the power is turned on?

It uses $FFFE and $FFFF when the power is turned on.

* What does the processor get from memory location $FFFE?

One byte.

* What does the processor get from memory location $FFFF?

One byte.

* The processor puts the bytes from memory locations $FFFE and $FFFF together. What is the process called, and what is the result in this case?

The process is called concatenation, and the result is a two-byte number.

* How many bits is two bytes?

16 bits.

Displaying a message

* How does the processor use the 16-bit number obtained by concatenating the contents of memory locations $FFFE and $FFFF?

It uses the 16-bit number as an address.

* What is the 16-bit number the address of?

The 16-bit number is the address of the first instruction the processor will execute.

* What part of the processor uses this instruction?

The instruction decoder.

* What keeps track of the instructions in the 6809 processor?

The program counter.

* What is the program counter and what is its shorthand name?

The program counter is a 16-bit register that contains the address of a computer instruction. Its shorthand name is PC.

* Name the index registers.

X and Y.

* What is the size of the X and Y index registers?

X and Y are each 16 bits in size.

* What is the most common function of the X and Y index registers?

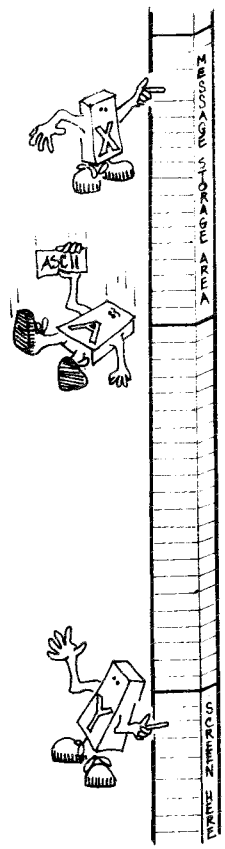
To index an address; that is, to hold the number of a memory location for reference.

* What is the starting address of the normal video display?

It starts at 1,024.

Program #5, a BASIC program. Turn on the power of your Extended Color BASIC computer. When the cursor appears, type CLOAD and press ENTER. The computer will search (S) and find (F). When the cursor reappears, LIST this program. If the program is not similar to the listing, or if an I/O error occurs, rewind to the start of the program and try again. For severe loading problems, see the Appendix.

```
10 DATA 8E.06.00
20 DATA 10.8E.04.00
30 DATA A6.80
40 DATA A7.A0
50 DATA 8C.08.00
60 DATA 26.F7
70 DATA 39
110 REM LOAD X WITH $0600 MESSGE
120 REM LOAD Y WITH $0400 SCREEN
130 REM LOAD A FROM X-INDEXED,          AND INCREMENT X BY 1
140 REM STORE A TO Y-INDEXED,           AND INCREMENT Y BY 1
150 REM COMPARE IF X IS $0800
160 REM BRANCH BACK IF NOT
200 FOR N = 16000 TO 16016
210 READ A$
220 A=VAL("&H"+A$)
230 POKEN,A
240 NEXT
250 CLS
260 PRINT"THE MESSAGE YOU ARE READING WAS ORIGINALLY DISPLAYED B
Y PRINTINGIT NORMALLY USING BASIC.  IT CANBE RECALLED AT ANY TIM
E -- ONCE THE BASIC PROGRAM HAS BEEN RUN  -- BY TYPING "CHR$(34)
"EXEC"CHR$(34)"."
270 PRINT"THE BASIC PROGRAM PLACED INTO   MEMORY THE 6809 MACHIN
E LANGUAGEPROGRAM DESCRIBED IN LESSON 3.  IN THIS PROGRAM, INDEX
 REGISTERSX AND Y ARE USED TO TRANSFER A  GROUP OF BYTES (IN THI
S CASE    ORDINARY ASCII CHARACTERS) FROM ";
280 PRINT"ELSEWHERE IN MEMORY DIRECTLY TO THE SCREEN MEMORY."
290 PRINT" "STRING$(30,191);
300 FOR N = &H400 TO &H5FF
310 POKEN+&H200,PEEK(N)
320 NEXT
330 CLS
340 FORN=1TO1000:NEXT
350 EXEC16000
999 GOTO999
```



RUN this program. A message printed by BASIC will appear on the screen, the screen will be cleared, and the message will appear again, this time printed by the machine language program I've just described. Now I feel bound to prove that I'm not fooling you with some fancy BASIC manipulations. Once you have RUN the program the first time, hit <BREAK> and then delete it. Type NEW to clear out the program. Now, I say smugly, type EXEC — that's E–X–E–C — and hit <ENTER>. The message reappears, partly obliterated by an "OK".

To see the whole thing, enter these two lines. Line 10. EXEC. Line 20. GOTO 20. That's it. Line 10. EXEC. Line 20. GOTO 20. Now RUN that. There's the message.

Have some fun. Try changing the program. Hit <BREAK>. POKE 16001,128 <ENTER>. POKE 16012,130. <ENTER>. Then RUN. You can POKE

16001 with any number from 0 to 253. POKE 16012 with the previous number plus 2. For example, Hit <BREAK>. POKE 16001,0 <ENTER>. POKE 16012,2 <ENTER>. RUN again. Take a break here to load and RUN the next program. When you RUN it, notice that it expects you to input a hexadecimal number this time.

Program #6, a BASIC program. Turn on the power of your Extended Color BASIC computer. When the cursor appears, type CLOAD and press ENTER. The computer will search (S) and find (F). When the cursor reappears, LIST this program. If the program is not similar to the listing, or if an I/O error occurs, rewind to the start of the program and try again. For severe loading problems, see the Appendix.

```
10 INPUT"MEMORY LOCATION (00 TO FE)";A$
20 A=VAL("&H"+A$):IFA>254ORA<0THEN10
30 POKE16001,A
40 IFA=254THENB=0ELSEB=A+2
50 POKE16012,B
60 EXEC
70 FORN=1TO1000:NEXT
80 CLS
90 GOTO10
```

You've RUN the program, and seen a number of curious screen displays. What you have done is simple. You redirected the X register, which was pointing to the message I stored in memory, to somewhere else in memory. You can see that it takes very few changes to spy anywhere into memory with even that little machine language program.

Return to Figure 5 in the MC6809E data booklet. At this point, I have introduced the A and B accumulators, the program counter PC, and the X and Y index registers. Again, if you feel you might need to review, this is the time to do it.

Turn your attention to Figure 5, and notice the bottom register marked CC — the condition codes. This special register gives the processor its limited intelligence. Also called the "flags", the condition code register contains bit-by-bit information about the processor's activities . . . what the processor does, and what the results indicate. In the beginning, the flags of most interest will be the Carry/Borrow Flag and the Zero Flag.

In this lesson's first program example, I had the A accumulator load a value from memory indexed by X, and store that value in memory indexed by Y. My program did that for exactly one screen full of information — 512 bytes. I should say that my program did that for exactly hex 200 bytes, which is an easier number to work with.

How did my program know to stop after **$200** bytes? Turn to your documentation book for this lesson, and follow

* That number was decimal since it didn't have a dollar sign in front of it. What is that starting address in hexadecimal?

1,204 in hexadecimal is $0400.

* If the X register is indexed to an ASCII character in memory and the Y register is indexed to the video display at $0400, how can the A accumulator get the message to the screen?

The A accumulator can load the value from memory indexed by X and store the value to memory indexed by Y.

* How does it do this?

It follows instructions.

* Where does it get the instructions?

It gets the instructions from memory.

* What is another name for the condition codes?

The flags.

* What information is held by the condition codes?

Bit-by-bit information about the processor's activities.

* What activities is the processor engaged in?

The instructions it is following.

* What keeps track of the instructions it is following?

The program counter, or PC.

* Now, where were we?

Talking about the condition codes, or flags.

* Oh yes. What does the instruction "compare" do?

It compares the contents of a register against another value.

* In what way does it compare?

It compares by performing a "ghost" subtraction.

* What does the ghost subtraction do?

It sets the condition codes (or flags) according to the results of the ghost subtraction.

* What are the results and the condition codes if the register's value is greater than the value being compared with?

If the register's value is greater than the value being compared with, the ghost subtraction causes no borrow and the result is not zero. The carry/borrow and zero flag is turned off.

* What are the results and condition codes if the register's value is the same as the value being compared with?

There is no borrow, but the result of the ghost subtraction is zero. The carry/borrow flag is off, but the zero flag turns on.

* Well, then, what if the register's value is less than the value being compared with?

The result isn't zero, but the ghost subtraction demands a "borrow". The carry/borrow flag goes on, but the zero flag is off.

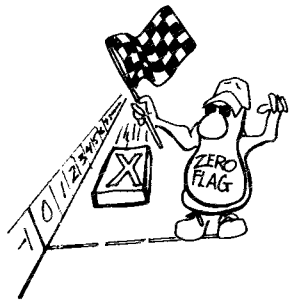
* So how do you use this information?

By learning the principles in this lesson very well before going on to the next lesson.

along with me as I describe a little more precisely how this program operates.

Step 1. Load X register with the immediate value of **$0600**. This is the address where the message is stored in memory.

Step 2. Load Y register with the immediate value of **$0400**. This is the address where the screen begins on the Color Computer.

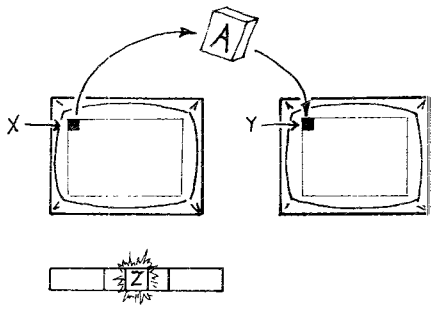


Step 3. Load A accumulator from memory indexed by X, and automatically increment the X register by one.

Step 4. Store A accumulator to memory indexed by Y, and automatically increment the Y register by one.

At the end of this step, there is a letter on the screen. X, having been incremented, now indexes the *second* character of the message, and Y, also having been incremented, indexes the *second* location on the screen. What I would like the program to do is somehow check to see if the job is finished. Here are the questions to consider:

When would the job be finished? When the screen is full. When is the screen full? If Y has been incremented, one step at a time, past the last screen position, and X has been incremented, one step at a time, past the last letter of the message. When is X past the last letter of the message? When it reaches **$0800**.



So the actions of Step 5 becomes clear.

Step 5. Compare X register to the immediate value **$0800**.

"Compare" is a microprocessor instruction which does what might be called a "ghost" subtraction. The only purpose of the ghost subtraction's result is to discover if the value being compared *with* is higher, lower, or equal to the register being compared *to*. Both original values remain unchanged — no actual result has been produced — but the result of this comparison can be discovered by the 6809 reading the condition codes. Here's how it goes:

If the register's value is greater than the value being compared with, then the ghost subtraction results in a non-zero, positive number. Both the carry/borrow flag and the zero flag are reset — turned off, that is.

If the register's value is equal to the value being compared with, then the result of the ghost subtraction is zero, turning on the zero flag but leaving the carry/borrow flag off.

And finally, if the register's value is less than the value being compared with, then the ghost subtraction results in

a negative (but non-zero) number. The carry/borrow flag goes on, the zero flag goes off.

So how does it fit here? At this point, I'm going to do something I like to avoid, and that's to explain assembly language using similar BASIC commands.

Program #7, a BASIC program. Turn on the power of your Extended Color BASIC computer. When the cursor appears, type CLOAD and press ENTER. The computer will search (S) and find (F). When the cursor reappears, LIST this program. If the program is not similar to the listing, or if an I/O error occurs, rewind to the start of the program and try again. For severe loading problems, see the Appendix.

```
10 X = &H0600
20 Y = &H0400
30 A = PEEK(X)  : X = X + 1
40 POKE Y,A     : Y = Y + 1
50 IF X <> &H0800 THEN 30
60 END
```

This is a short program, and I'd like you to list it before you run it. In case you've never used BASIC's peculiar notation for hexadecimal, it's "ampersand H". Now in the 6809, X, Y and A registers are *not* variables. I'm using those names here just for visual effect. But follow this through. In line 10, X is **$0600**, the first memory location of the message. In line 20, Y is **$0400**, the first memory location of the screen. In line 30, A takes the value indexed by X — here I use PEEK to create the same effect — and X is incremented by one. In line 40, A stores its value at the location indexed by Y — here I use POKE to create that effect — and Y is incremented by one. In line 50, the compare is done. X is compared with **$0800**; if it isn't **$0800**, then the program isn't done, and it branches back to line 30.

RUN the program. It does, quite slowly, exactly what the machine language program did. To finish this lesson, load and examine the source code that follows on this tape.

Program #8, an EDTASM+ program. Insert the EDTASM+ cartridge, and turn on the power to your computer. When the cursor appears, type L and press ENTER. The computer will search (S) and find (F). When the cursor reappears, display the program. Type P#:* and press ENTER. If the right-hand side of the program is not similar to the listing, or if an I/O error occurs, rewind to the program's start and try again. For severe loading problems, see the Appendix.

```
00100            LDX     #$0600
00110            LDY     #$0400
00120 LOOP       LDA     ,X+
00130            STA     ,Y+
00140            CMPX    #$0800
00150            BNE     LOOP
00160            RTS
00170            END
```

Registers vs. variables

Review:

* What does CPU mean; what CPU does the Color Computer use?

Central Processing Unit; the Color Computer uses a 6809 CPU.

* What are the terms for one binary digit and for eight binary digits?

The terms are bit and byte.

* What does ALU mean, and what kinds of arithmetic does the 6809's ALU perform?

ALU means Arithmetic Logic Unit, and the 6809's ALU performs addition, subtraction, multiplication, AND, OR, NOT, Exclusive OR, incrementing, decrementing, and comparison.

* How many ALUs does the 6809 have, and what are they called?

The 6809 has two ALUs called the A and B accumulators.

* Where do the accumulators get and save their information, and what are the terms for getting and saving data?

The accumulators the information from other registers and from memory; the process is loading and storing data.

* What is the address range of the 6809 CPU in hexadecimal?

The address range is $0000 to $FFFF.

* How does the processor get started, and what keeps track of its instructions?

By loading and concatenating the data at memory locations $FFFE and $FFFF, and using the result as the address of its first instruction. The program counter, or PC, keeps track of the instructions.

* What are the index registers, what do they hold, and what are they for?

The index registers are X and Y, they hold 16 bits each, and they are most often used to hold the address of a memory location.

* What are the condition codes?

The condition codes are bits that hold information about the processor's activities.

* Give another name for the condition codes, and name two of the codes.

Condition codes are also called flags; carry/borrow and zero are condition codes.