MEETING NOTICE

The next meeting will be August 15 at 7:30. The meeting location is Alpha Audio's third floor conference room, at 2049 West Broad Street. The night-time phone number there is 358-3853. The front door has a touch-pad combination lock, and the combination for the night will be 8043 (eight zero four three).

Carlos Chafin and Jim Scott will continue to discuss Assembly language programming for novices.

Everyone is welcome!

MINUTES (Meeting of July 18, 1983)


There were no corrections to the minutes of the June meeting.

We were again glad to welcome new members, two who joined our group at this meeting, Tony Batchelor and Larry Keen. Tony has a 289 and Larry has an H89. Both use their computers in their business. We were also happy to have Karen Trinkle, Nelson's "better half", visit with us. We hope that Karen, as well as the other wives and girlfriends, will continue to visit (or Join) with us as we delve into the profound and fascinating details of our hobby!

Following an informal discussion period, Scott and Chafin resumed the discussion of Assembly Language Programming. Scott began the session with an oral quiz, presumably to determine our comprehension (or lack thereof) of the material covered thus far. Appearing not unduly frustrated by the feeble response, he proceeded with a brief explanation of each instruction of the 8080 instructions set, with
particular emphasis on addressing modes and registers and flags affected by each instruction.

At Chafin's suggestion, it was decided that next month some programs will be analyzed to illustrate the practical applications of the various instructions discussed.

The meeting was adjourned at 10:00 p.m.

Parks Watson  
Secretary/Treasurer

NEWS

COLUMN STILL MISSING

For the second month, Parks Watson's regular column, "For the 4H Club", does not appear because his computer, Heathcliffe, is still ailing. But prospects are good for a resumption of Parks's Pro-HDOS commentary next month.

H-19 ULTRA ROM CONVERSION

by Jerry Tiller, RHUG pseudo vice president

Several weeks ago we all received our copies of BUS$ #71 in the mail. If you were careful to read it like I know you are, you would have seen an article describing the Ultra Rom conversion for Heath H-19 terminals and H-89/90 computers. Well, after a small debate with my wife I convinced her to let me purchase the kit. I'm now going to attempt to tell you all about it. I said attempt because I'm sloppy at communications using the CP/M text editor. It's terrible. One of these days I'll break down and buy a word processor of some sort. Anyway on with the show.

The ROM conversion kit can be purchased from Tom Jordenson at his humble abode (Software Wizardry) for the paltry sum of $39.95. Paltry you say? Well, it is when you see what it can do. If you have money to spare you can even purchase the optional Second Page of memory. Still not convinced? Let me continue.

The kit comes from Software Wizardry in a nice box, shipped via UPS without being destroyed. In the box was a hardback binder with 30 pages of documentation and an envelope with the EPROM and RAM chips tucked quietly and safely away in non-conductive foam. The art work on the cover is nicely done and the printed matter inside is not Xeroxed from someone's original (we've all seen this type of documentation).

So much for the pretties; now on with the installation.

The installation is done in several steps, the easiest of which is to insert the IC's in the sockets. On the old H-19 terminals and H-89/90 computers there are 5 jumpers to remove from the terminal board and 6 to add. If you are installing the second page of memory there is 1 more to remove and 2 more to add. The newer H-19A terminals and H-89/90 computers do not have to remove or add any jumpers unless the second page of memory is installed. The
instructions in the book are clear and give you both the old and new IC socket designations. After the jumpers are in you can install the chips.

The Ultra ROM kit comes with 2 EPROMs. One is a 2532 (4K) EPROM for the terminal attributes and the second is a 2716 (2K) EPROM for the keyboard encoder. The optional second page is a 6116 (2K by 8) static RAM chip. It can be purchased for an additional $15.00. The 6116 can be purchased elsewhere for less but I purchased it at the same time for the sake of convenience.

The installation took me approximately 1 hour to complete as I double-checked all of the jumpers I added. This was to be sure the terminal did not generate a mushroom cloud when I powered it up.

Before turning the terminal on you must reconfigure the 2 DIP switches (S401 and S402) for a new configuration. Some of the old features have been deleted, such as 50/60 Hertz, Auto CR on LF, Auto LF on CR, several nonused Baud rates and the ANSI mode. These were deleted to make room for all the new features added. It isn’t much of a loss since very few people used them anyway.

The new ROM chips offer you several new hardware configurations of which 4 cursor styles, interlaced scan and preprogrammed function keys are a few.

The main features of the chips are as follows:

- Software Handshaking enable/disable
- CP/M commands for the function keys
- HDOS commands for the function keys
- MBASIC commands for the function keys
- User programming for the function keys
- Native Keyboard operation (unique single key codes a la Z-100)
- Enable/Disable clock on line 25
- Transparent mode (all ASCII char and Escape seq show on screen)
- Self Test of terminal memory (ROM and RAM)
- Back Tab
- Swap page 1 and 2 (if equipped)
- Copy to page 1 or 2 from screen (if equipped)
- Fill display with a character
- Fill Line Horizontally with a character
- Fill Line Vertically with a character
- Reverse Video on individual line
- Reverse entire screen

There are combinations of the above features available at the touch of a key or two.

There are just a few drawbacks to the conversion which are screen jitter when in Interlace Mode, slow operation of the backspace key when using the CP/M editor and a couple of 3 key Escape combinations to activate some of the features. They aren’t bad enough to detract from the rest of the features you get.

I’ve spent the last 6 hours playing with all of the features and have thoroughly enjoyed it. If you are used to using one of the newer Smart Terminals like the TAB 132 or the DEC VT125 then you will find it worthwhile to purchase the conversion kit.
This ends my very first article for the newsletter. I was also able to talk Kay into letting me buy a new TRONIX Motherboard too. So I’ll probably be back next month with the details on it.

ASSEMBLY LANGUAGE PROGRAMMING - PART 3
by Jim Scott

INTRODUCTION

This is the third of a series of articles which parallel and summarize the discussions about assembly language at our meetings. The purpose of the discussions and the articles is to present enough information about assembly language programming so that someone who knows how to program in a higher-level language, and is willing to use the proper manuals for reference, will at least have some idea how to get started at programming in assembly language.

The previous article (July issue of the Gazette) described the logical structure (memory and registers) of the 8086 central processing unit (CPU). It considered several examples of assembly language instructions, how they relate to the corresponding machine language instructions, and what they cause the computer to do.

This article is the first of several which will take a more organized look at the entire 8080 instruction set. If your primary interest is with a different CPU (the 280 in an H89 or a modified H8, the 8086 in a modified H89, or the 8088 in an H100), consider the information in this article as an example of what assembly language is like. You will need to learn a different instruction set before you can do assembly programming for your CPU. The 280 instruction set includes the entire 8080 instruction set plus a lot more, but the assemblers available for the 280 use different mnemonics (operation codes). The H100 includes an 8085 as well as an 8088, and the 8085 is almost identical to the 8080; the 8085 has just two additional instructions.

REFERENCES

In addition to the many books and articles written about 8080 assembly language programming, three publications are particularly valuable as references.

Each computer comes with a manual that describes the instruction set of that computer’s CPU. In the case of the H8, it is the "Heathkit Manual for the Digital Computer Model H8 - Operation". This manual has a 17-page section entitled "Instruction Set" (undoubtedly drawn almost verbatim from the "Intel 8080 Microcomputer Systems User’s Manual"), which gives all the details of how each instruction operates. It shows what the instruction looks like in machine language, bit by bit. It also gives the mnemonic operation code used in assembly language. This is the ultimate authority on the 8080’s instruction set. However, it does not give any examples of how the instructions are used.

A reference card generally comes with each computer. The reference card for the H8 lists all the instructions, alphabetically by mnemonic, and it also lists the mnemonics in a table which shows the hexadecimal or split octal value of the first byte of each machine language instruction.
The third reference is a concise summary of the information in the operation manual. The July 1979 issue of Byte contained an article, "Intel 8080 Microprocessor Instruction Set", which has a two-page summary with one line for each instruction, plus a definition of the symbols used in the summary. In a way, this is even more of a reference, and less of a learning tool, than the operation manual. But it helps to be able to see the entire instruction set in such a small area; at least, it helps you understand that there are not that many instructions to learn.

GROUPS OF INSTRUCTIONS

The 8080 instructions fall officially into five groups:

Data Transfer Group
Arithmetic Group
Logical Group
Branch Group
Stack, I/O. and Machine Control Group

There, that wasn't so hard, was it? Now let's look at the first group in detail.

DATA TRANSFER GROUP

Instructions in this group move one or two bytes of data from one register or register pair to another, from a register or register pair to memory, from memory to a register or register pair, and from within the instruction itself (immediate data) to a register or register pair. The individual instructions are as follows.

MOV (Move)

Format: MOV r2, ri
where r2 and ri can be registers A, B, C, D, E, H, or L; or the letter M, meaning the memory location whose address has been previously stored into register pair HL. This instruction moves a byte from ri to r2.

Example: MOV M,B
This moves the contents of register B into the memory location whose address is in HL.

The MOV instruction is the general purpose data movement instruction. Data can be moved from one memory location to another by making an intermediate move into a register.

MVI (Move Immediate)

Format: MVI r, data
where r can be registers A, B, C, D, E, H, or L; or the letter M, meaning the memory location whose address has been previously stored into register pair HL. This instruction moves a byte of immediate data (the contents of the second byte of the machine language instruction itself) into register or memory location r.

Example: MVI C, "$"
This moves a dollar sign (a byte whose binary value represents the ASCII character "$") into register C. The machine language instruction would be 0E24 in hex, where 0E is the opcode standing for "MVI C,"", and 24 is the hex value for the ASCII dollar sign.
The MVI instruction is used for moving a constant (a byte of data whose value is known at assembly time).

**LXI (Load Register Pair Immediate)**

Format: LXI rp, data16
where rp can be B, D, H, or SP, representing register pairs BC, DE, HL, and the stack pointer, respectively. This instruction moves the contents of the second and third bytes of the machine language instruction into register pair rp.

Example: LXI D, 89ABH
This moves the two-byte integer whose hex representation is 89AB into register pair DE. (In machine language, using hex, this instruction would be 11AB89; the two halves of the immediate data are stored backwards in the instruction.) Register D would then contain 89H, and register E would contain ABH.

The LXI instruction is generally used for loading a two-byte memory address into a register pair. Usually, the second operand (data16) in assembly language is a statement label, which the assembler converts into an actual numerical address at assembly time.

**LDA (Load Accumulator Direct)**

Format: LDA addr
where addr is a two-byte address of a memory location. This instruction moves into register A the byte of data at the memory location whose address is contained in the second and third bytes of the machine language instruction.

Example: LDA 3FEEH
This moves the byte from memory location 3FEE into register A (the accumulator).

The LDA instruction puts into the accumulator a byte of data whose address is known at assembly time. As with the LXI instruction, the addr operand is usually a label whose actual address is calculated by the assembler. This differs from MOV A,M in that the MOV instruction allows the memory address to be calculated at execution time, and requires that the address be put into HL.

**STA (Store Accumulator Direct)**

Format: STA addr
where addr is a two-byte address of a memory location. This instruction moves the byte of data in register A into the memory location whose address is contained in the second and third bytes of the machine language instruction.

Example: STA 0F12H
This moves the byte from register A into memory location 0F12.

See the comments for the LDA instruction.

**LHLD (Load HL Direct)**

Format: LHLD addr
where addr is a two-byte address of a memory location. This instruction moves into the HL register pair the two bytes of data at the memory location whose address is contained in the second and third
bytes of the machine language instruction. (The bytes of data are assumed to be in reverse order, so the byte at addr goes into L, and the byte at addr+1 goes into H.)

**Example:** LHLD 81E2H
This moves the byte from memory location 81E2 into register L, and the byte from memory location 81E3 into register H.

The LHLD instruction is not the same as an LXI H,addr instruction. Both are used to put two bytes of data (generally a memory address), into HL; LHLD sets the data from memory, but LXI sets the data from within the instruction itself. In other words, LHLD is different in that it doesn't require that the value of the data be known at assembly time.

**SHLD (Store HL Direct)**

**Format:** SHLD addr
where addr is a two-byte address of a memory location. This instruction moves the two bytes of data in the HL register pair into the memory location whose address is contained in the second and third bytes of the machine language instruction. (The bytes of data are stored in reverse order; the byte in L goes to addr, and the byte in H goes to addr+1.)

**Example:** SHLD 6412H
This moves the byte from register L into memory location 6412, and the byte from register H into memory location 6413.

The SHLD instruction is generally used for storing two-byte addresses into memory locations.

**LDAH (Load A Direct)**

**Format:** LDAH rp
where rp is B or D, representing the register pairs BC and DE respectively. This instruction moves the byte of data from the memory location whose address is contained in register pair rp, into register A.

**Example:** LDAH B
This moves the byte from the memory location whose address has previously been stored into register pair BC, into register A.

The LDAH instruction is different from the MOV A,M instruction in that LDAH uses register pair BC or DE to point to the memory location, whereas MOV uses HL.

**STAX (Store A Direct)**

**Format:** STAX rp
where rp is B or D, representing the register pairs BC and DE respectively. This instruction moves the byte of data in register A into the memory location whose address is contained in register pair rp.

**Example:** STAX D
This moves the byte from register A into the memory location whose address has previously been stored into register pair DE.
See the comments for LDAX.

XCHG (Exchange HL with DE)

**Format:** XCHG

This instruction moves the two bytes of data from register pair HL into DE, and vice versa.

**Example:** XCHG

The XCHG instruction is often used when a program needs to keep track of two memory addresses at once, using HL to point to them (e.g., for a MOV A,M instruction). HL can only contain one address at a time, but DE can be used to hold the other one temporarily.

**CONCLUSION**

Future issues will cover the remaining groups of instructions.

Richmond Heath Users Group
4302 Smithdeal Avenue
Richmond, VA 23225