GAZETTE

The Official Newsletter of the
Richmond Heath Users Group

Volume I Issue 14
JULY 1983

Subscriptions: $5.00 per calendar year.
Editor: Jim Scott, 3464 Northview Place, Richmond, VA 23225.
Next editorial deadline: August 1, 1983.
Note to other HUGs: If you want to trade newsletters with us, please mail yours to Carlos Chafin, 4302 Smithdeal Ave., Richmond, VA 23225.

MEETING NOTICE

The next meeting will be July 18 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 8645 (eight six four five).

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

Everyone is welcome!

MINUTES (Meeting of June 20, 1983)

Present: Carlos Chafin, David Harrington, Harold Lana, John Purcell, Jim Scott, Ron Stauffer, Hank Steigleder, Parks Watson and Gary West.

The minutes of the May meeting were approved with the following correction: Chafin, discussing his Computer Innovation compiler, was quoted as saying "...is better than Aztec C." He stated that this comparison was unjustified and unintended, since he has little or no knowledge of Aztec C.

We extend a warm welcome to Gary West, who joined our group at this meeting. Gary is the proud owner of an H100 and the subject of envy of the rest of us! (Carlos excepted).

Chafin opened the meeting with the reminder that the Second Annual CHUG CONVENTION (CHUGCON) was coming up in October. He and Scott attended the first one and felt it was well worth the effort and midnight return ride. Scott added that the $12/yr. membership fee to CHUG was worth the cost just to get their newsletter. The current issue announced two one-evening seminars to be held at Fairfax High

School: "Introduction to C Programming Language" and "Introduction to 8080 Assembly Language".

Scott also noted in the CHUG Newsletter, that the Manassas Explorer Scout Post was soliciting donations of your surplus H8 and H89 boards and miscellaneous hardware, etc. Their value, which the donor sets, is understood to be tax deductible as a charitable contribution. He sent them several 8K boards and anticipates receiving a receipt documenting his contribution. The complete address, if you wish to donate any hardware, is:

Raymond Halverson
9272 Bayberry Ave.
Manassas, Virginia 22110

Scott and Chafin continued the discussion of assembly language programming with a brief review of material covered so far. Then a description of the 8080 registers, their similarities and differences and their special purposes. Sample programs were used to point out how the various registers were used and instructions relating to them.

The meeting was adjourned at 10:00 PM.

Parks Watson
Secretary/Treasurer

****

NEWS

QUERY! UPDATES AVAILABLE

Updates are available to the Query! database system from Hoyle & Hoyle Software (716 S. Elam Ave., Greensboro, NC 27403). Specifically, a new version of the SORT program, Version 10:04, sorts much faster than SORT Version 10:01. There are two visible changes: it warns you not to sort your database without backing it up first, because if the sort doesn't finish for some reason, the database would be destroyed; also, it asks you to specify which disk drive is to be used for temporary disk space during the sort.

The new sort also seems to have another advantage over the old one. You can now effectively sort on multiple fields by sorting multiple times, doing the major sort key last. For example, if you want your mailing list sorted on Last Name, and on First Name where duplicate Last Names exist, sort first on First Name; then sort on Last Name. When the second sort encounters duplicate Last Names, it keeps these records in the order produced by the first sort. The old version of SORT would not do this.

There is also a new version of the program WRITER, part of the optional Report Writer for use with Query! It is still designated 10:01, but I did find that it fixed one bug that had bothered me. When WRITER was used to print a formatted report of a database, and when a record would have been split between pages, the second part of that record would have been dropped. The new REPORT correctly splits the record between pages.

The updates are $3.00 each ($6.00 for both) from Hoyle & Hoyle.

Coming later are Query II and Query Utilities. According to Hughes Hoyle, Query II "has many minor changes to make use easier". I

understand that Query Utilities will have an AUTOADD program which will be the reverse of the F command of the OUTPUT program. (F creates a standard disk file, using the data in your database.) This could be quite a breakthrough for Query!, since it apparently will allow you to convert an existing file, created by a text editor, to a Query database. This will make Query! more valuable to people who have existing data files which they have been maintaining manually, using a text editor. The current Query! provides no way of converting such a file to a database other than by rekeying the data. AUTOADD may also provide a way of getting around the restriction of not being able to change the structure of a Query! database (number or size of fields, for example); if you could use the F command to write the database to a standard file, then use CREATE to create a new database structure, then use AUTOADD to read the data into the new database, the job would be done (except for adding any data to go into fields that didn't exist before). Of course, this is speculation, and we won't know what Query II and Query Utilities will really do until we get our paws on them.

++++++++++

COMPUTER BITES MAN

Parks Watson's regular column, "For the 4H Club", does not appear this month because his computer took a nose dive before he finished editing it. Best wishes for a speedy recovery, Heathcliffe!

****

I BUILT THE H100

by Gary West

What sounds like a real "claim to fame" is really not saying much at all when the actual amount of "building" consists only of one floppy controller card and some wiring harnesses.

I must admit that I only considered buying the kit form H100 computer since I had successfully built Heathkit stereo, test, ham radio equipment and two GR2000 televisions. Having seen several pieces of Heathkit equipment essentially charbroiled by someone who did not have the experience to tackle the project, is probably the most significant sign of an aggressive hobbyist who overestimates his talent. I certainly didn't want to fall into the ranks of the overcookers, at least not for the price tag on the H100. Therefore, hoping that there was truly "limited construction", as advertised, I purchased my H100.

The first problem encountered, as with most new toys, was "where to build it?". Easy! Any spare room in the house where during construction it will be unmolested by spouse (girlfriend), pets, kids, tornadoes, etc.

Second, as always, where did all this packing material come from and, have I lost any parts in the package of cardboard and foam that the Junk man is hauling away.

Trivial, you say, but for me these two items could constitute the most significant problems one encounters in kit building.

Now to the meat of the project, which involves a routine plug-in and solder task where one completes the disk controller card while

guided by a typical Heathkit construction manual. (I'm assuming most readers are familiar with such an animal.) This task is, of course, unexciting and much less eventful. In fact, while I was building it I knew if it hadn't been for the cost I would have bought the assembled unit. I hate the smell of solder smoke.

Once the disk controller card is complete you go directly to final assembly, that is unless you've purchased additional RAM to plug into the motherboard or video processor board. When the RAM is plugged in and you've completed a few jumper position checks, the next step is to mount the ominous-looking motherboard, the power supply, keyboard, floppy drives, and card cage. All of this takes no more than a couple of hours. One quiet evening.

It was during the mounting of the motherboard that I began to realize why constructing it wasn't a part of the kit building. I believe one would go blind soldering all the pin connections on the four-layer circuit board. I'm thankful that the motherboard and video processor boards were preassembled and tested. Oh yes, before I forget, the power supply comes preassembled too. Just mount it and plug in the wires. All in all, the assembly, including the disk controller card construction, takes roughly ten hours.

Well, now the moment we've all been waiting for, "smoke test"! And I didn't purchase a monitor! Now let's see - HUG . . . Richmond, VA . . . Jim Scott. Ahaa! Ring . . . Hello! . . . Hello, Jim, this is Gary, help! Carlos Chafin? Thanks. After a couple short phone calls to some unbelievably helpful RHUG members, I was loaned a monitor for this moment of truth. Many thanks to Carlos Chafin for the loan of his monitor and to Jim Scott for the lead.

My computer (he proudly exclaimed) worked fine from the first moment I turned it on. Well, let me clarify, there were a few moments when the machine didn't understand what I was telling it.

There is now one other small "glitch" in the machine, and I can't say I'm happy about it. I, who waited so patiently to be able to afford the additional video RAM, 2nd 5 1/4" floppy, and expansion RAM (to 192K) so that I wouldn't have to disassemble my computer, now have to do just that since the trasducer which provides the key click and beep is defective. The best laid plans of . . . !

For the last month, since my machine has been operating, I have spent many hours in awe by the power of the micro. Aside from the key click and beep (which I really don't miss) I have no complaints. The graphics are marvelous and with the S100 expansion bus my mind has been racing ahead of my present skills to the time when I can interface to about anything I choose simply by plugging in the appropriate board and related software.

This is my first micro, and I have had no trouble adapting myself to it. I guess that says a lot about the people who designed the package. Thank you Heath/Zenith (or the other way around to avoid offending anyone).

If you're borderline about a micro, look no further.

If I can answer any questions about construction or getting started with the H100, please call me.

Gary E. West

1117 Meadow Drive
Mechanicsville, VA 23111
730-1998

****

ASSEMBLY LANGUAGE PROGRAMMING - PART 2

by Jim Scott

INTRODUCTION

This is the second 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 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 (May issue of the Gazette), said that you need to know three things before writing an assembly language program (other than such extensive topics as how to design and structure a program):

1. What processor (CPU) will you be programming for?
This series of articles deals with the 8080 CPU.

2. What operating system will your program run under?
This series deals with HDOS and CP/M.

3. Which assembler will you use?
This series covers the standard HDOS and CP/M assemblers, both named ASM.

The previous article then described the various parts of the listing you get when you assemble a program, and included two examples of assembly listings, one for CP/M and one for HDOS.

THE 8080 CPU

At this point, we can take a look at the logical structure of the 8080 CPU, so we can understand what the machine language or assembly language instructions do. Note that we couldn't care less what the physical structure of the 8080 is; we just care how it behaves.

The 8080 has memory, up to 64K locations. Each memory location holds a byte, or 8 bits, of information. Each location has an address, which is an integer from 0 up to the highest address (65535 if the computer has 64K). A byte of memory, being 8 bits, can represent an integer from 0 to 255, or an ASCII character (such as 'G', 'a', '4', '%', or ctrl-C), or eight individual bits representing eight separate yes-or-no options, or any other interpretation your program wants to put on it (as long as it is understood that a byte can contain only 256 different values).

The 8080 also has registers. Each register is a byte, just like a memory location. (A very simple assembly language program might, for example, move a byte of data from a memory location into a register, and then move the data from that register to a different memory location.) Some of the registers may be used in pairs. A register pair is two bytes, or 16 bits, and can represent an integer

from 0 to 65535, or two ASCII characters, or a signed integer from -32768 to +32767, or a floating point number, or anything that requires up to 65536 combinations. Note that a register pair will very neatly contain an (unsigned) integer representing the address of a memory location.

The one-byte registers are as follows. The "symbol" for a register is how it is usually referred to in assembly language.

Symbol	Name	Description
A	Accumulator	All arithmetic operations use the A register.
F	Flag Word	This is a set of individual bits which represent such yes/no facts as whether the result of the preceding arithmetic instruction was zero. No instruction operates on this as an individual register, but many instructions affect its individual bits, or flags. The flags are: S (Sign) Z (Zero) 0 (unused) AC (Auxiliary Carry) 0 (unused) P (Parity) 1 (unused) Cy (Carry)
B	B register	A general purpose register.
C	C register	A general purpose register.
D	D register	A general purpose register.
E	E register	A general purpose register.
H	H register	A general purpose register.
L	L register	A general purpose register.

The register pairs are as follows. The first four are pairs of the individual registers listed above.

Symbol	Name	Description
PSW	A-F register pair, or Processor Status Word	This register pair is used as a pair only by the PUSH PSW and POP PSW instructions.
B	B-C register pair	A general purpose register pair.
D	D-E register pair	A general purpose register pair, often used with the XCHG instruction to contain an alternative value for the H-L register pair.

H	H-L register pair	Usually used to contain the address of a memory location, for use by instructions which read from or write to memory.
SP	Stack Pointer	Contains the address of the memory location which is the current "stack" location. The stack is a set of consecutive memory locations which are used for temporarily saving data so that it can be retrieved later. Instructions such as PUSH, POP, CALL, and RET make special use of the stack pointer.
PC	Program Counter	Contains the address of the memory location which contains the next instruction to be executed by the computer. Most instructions simply add their instruction length (one, two, or three bytes) to the PC, so that the instructions are executed in sequence. Instructions such as JMP, CALL, RET, and PCHL can put other values into PC, to alter the order with which the instructions are executed.

SOME INSTRUCTIONS

Now that we know what we need to know about the logical structure of the 8080, let's look at a few of the instructions in the 8080's instruction set, and see how they manipulate memory locations and registers. The examples are from the sample listings, printed in the May issue; if you don't have that issue, look at any assembly listing you can find. Remember that the assembly instructions are more or less in the middle of the page, whereas the corresponding machine language instructions are listed in hex or octal toward the left of the page.

The 8080 has machine language instructions of three lengths: one, two, and three bytes. The first byte is always the opcode, representing the type of instruction. The rest of the instruction, if any, is data or the address of data to be operated on. In some cases, some of the eight bits in the opcode are used to specify which register is to be operated on. In general, you do not need to be concerned about the machine language instructions; just remember that each assembly instruction is translated by the assembler into a machine language instruction.

Let's start with the instruction

```
MVI     A,'E'
```

This is a "Move Immediate" instruction. It moves a byte of data, representing the letter 'E', into the accumulator. "MVI" is the assembly language opcode, or instruction mnemonic. "A" is the first operand, representing the A register. "'E'" is the second operand. The term "immediate" in the name of the instruction has nothing to do with how fast the instruction is executed; it refers to the fact that the second operand is "immediately nearby", because it is contained in the instruction itself. If you look at the octal machine language

(HDOS sample, location 042.220), you see that it is 076 105. The 076 means "MVI A," and the 105 is the octal representation of the letter 'E'.

The "Store A" instruction

```
STA     CHAR
```

stores the contents of the accumulator into a memory location represented by the label CHAR. (As is true of most instructions, the "from" location, in this case the accumulator, is not changed.) Look at the machine language (CP/M sample, location 0137): it is 323B01. The 32 is the opcode, representing "STA". The 3B01 is the address of the operand, with the halves of the address switched (I have never figured out why the 8080 does this); in other words, the actual address of the memory location into which the contents of A are to be stored is 013B. Look at location 013B in the assembly listing, and you see

```
CHAR    DS      1
```

This is an assembler directive instruction. It does not get translated into machine language, but it tells the assembler to reserve a byte of memory at this point in the program, and to remember that other instructions will refer to it as CHAR. Of course, the programmer generally will not know or care what exact address CHAR actually represents, as long as the correct address gets put into the machine language instructions that need it.

The Load A instruction

```
LDA     CHAR
```

moves the value from memory location CHAR into the accumulator.

Another "Move Immediate" instruction,

```
MVI     C,TYPEC
```

(see CP/M listing, location 0115) moves the one-byte value represented by the label TYPEC into the C register. The value of TYPEC is defined earlier in the program by the assembler directive

```
TYPEC   EQU     2
```

The "Increment Register Pair" instruction

```
INX     H
```

(CP/M listing, location 018C) increments (adds one to) the H-L register pair. In machine language, this is a one-byte instruction. Two of the bits in the hex opcode 23 specify which register pair is to be incremented.

The "Jump" instruction

```
JMP     TYPTX1
```

(CP/M listing, location 018D) moves into the program counter (PC register pair) the two-byte value represented by the label TYPTX1, which in hex happens to be 0183. The value of the label is defined by

the assembler directive

```
TYPTX1  EQU     $
```

which equates the label TYPTX1 to the address of the next instruction. So the Jump instruction will cause the computer to change its sequence of execution; the next instruction to be executed, after the JMP at 018D, will be the MOV at 0183.

That "Move" instruction

```
MOV     A,M
```

moves a byte from a memory location into the accumulator. How does the computer know which memory location to get the data from? Instructions like this, which have an operand of "M", use the H-L register pair to point to the memory location. In other words, a preceding instruction will have put into H-L the address of the memory location.

So far, none of these instructions has affected or used the Flag Word. The "Compare Immediate" instruction

```
CPI     0
```

(CP/M listing, location 0184) compares the contents of the accumulator with an "immediate" value of zero. Various flags may be set to 0 or 1 according to the results of the comparison; among them, the Z flag will be 1 if the contents of the accumulator equals zero, and will be 0 otherwise.

This is followed by a "Jump If Zero" instruction

```
JZ      TYPTX2
```

which Jumps to (sets the PC to the address of) the instruction at address TYPTX2 if (and only if) the Z flag in the Flag Word is 1. Thus, the CPI/JZ instruction pair can be used to change the execution sequence of the program depending on the value in the accumulator. Think about this; this capability of executing different parts of a program depending on previous results is what gives computers their power.

We have examined only a sampling of the instructions available on the 8080, but you are probably beginning to get an understanding of what types of things can be done at the assembly language level. Next time, we can take a more organized approach to the whole instruction set.