GAZETTE
The Official Newsletter of the
Richmond Heath Users Group

Volume I Issue 12
MAY 1983

Subscriptions: $5.00 per calendar year.
Editor: Jim Scott, 3464 Northview Place, Richmond, VA 23225.
Next editorial deadline: June 6, 1983.
Note to other HUGs: If you want to trade newsletters with us, please mail yours to Carlos Chafin, 4302 Smithfield Ave., Richmond, VA 23225.

(c) 1983 Richmond Heath Users Group. Contents may only be reproduced with proper credit to both the individual author and the Richmond Heath Users Group, but are not to be reproduced for commercial purpose without the express consent of RHUG.

MEETING NOTICE

The next meeting will be May 16 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. (See the article "Assembly Language Programming - Part 1" elsewhere in this issue for a summary of what was discussed at the April meeting.) Everyone is welcome!

MINUTES (Meetings of April 18, 1983)

Present: Carlos Chafin, Bill Ewing, David Harrington, John Purcell, Jim Scott, Ronn Stauffer, Hank Steigleder, Nelson Trinkle, Bobby Tulloh, Parks Watson.

There were no corrections to the minutes of the March meeting!!!

For the sake of accuracy, it should be stated that Chafin and Tulloh were "on site", but were preoccupied with some "government work" of some nature. However, we are giving them the benefit of the doubt in the above listing.

Purcell stated that his H25 is still being obstinate in refusing to obey all reasonable(?) escape sequences. Many suggestions were offered by those present, which no doubt clouded the issue further.

Trinkle gave H & H Enterprises an A1 rating for service, price and quality of product on their printer-parallel interface deal.

Scott presented an introduction to Assembly language programming, using listings of a short sample program in both HDOS and CP/M. The
questions generated confirmed the novice status of most of us.

The meeting was adjourned at 9:30.

Parks Watson
Secretary/Treasurer

FOR THE 4H CLUB
Hard Sectored, Hi-shotguns, HDOS Hombres
(who would rather fish than switch)
by Parks Watson

You loyal 4Hers are by this time no doubt used to my semi-
facetious comments about those misguided individuals who insist, for
some unfathomable reason, that CP/M is superior to HDOS. You probably
have also gathered that I am an outstanding non-authority on HDOS and
totally ignorant of CP/M! However, I am wholly dedicated to HDOS
since that’s the only operating system I’ve got and the reason I’ve
got it is that when I bought my H89, HDOS, including BHABASIC, cost
$150, while CP/M and MBASIC together cost $300. Since I knew nothing
about either system, I figured being confused $150 worth was
preferable to blowing my mind $300 worth. Of course, if I had waited
six months, I could have gotten both systems and MBASIC for free, and
saved three or four hundred dollars on the price of my H89 to boot.
But I’m used to that — I paid $120 for my first “four-banger” pocket
calculator too — doing my bit to further the horizons of technology.

So with that preamble, you can see where I’m coming from. I make
no pretense of being objective about said matters. I’m writing from
the point of view of the owner of a single disk drive H89 system with
HDOS/BHABASIC only and an MPI dot-matrix printer and my present goal is
to master what I’ve got before I confuse the issue with additional
operating systems, languages or peripherals. Already I find myself
trying to “CAT” or “TYPE” while in BASIC or “LIST” in HDOS and being
puzzled when I get an error message.

However, before leaving the subject of HDOS vs CP/M, the
following excerpts from several REMark and BUSS articles, which you
may have read but probably not in “contiguous files”, pretty well
settle the matter in my mind. As you will see, my stand is not
totally frivolous, judging from the opinions of others more
knowledgable than I.

Doc Campbell, the ubiquitous computing MD, writes: “Some folks
have asked us why we went to the trouble of preparing autoboot (HDOS)
disks, when CP/M can do the same thing. The reason is simply we like
the many features of HDOS. We like to have a date printed in the
directory when we manipulate files, etc. We have NEVER lost any data
using HDOS. These features are, and have been, very important to
us.”[1]

And D. C. Shoemake writes: “As a long-time software hacker
…….I’m sold on the efficiency and ease of operation of HDOS, and
wouldn’t give it up for any other operating system I’ve seen, short of
UNIX. However, as many have come to realize over the past year or so,
there’s no substitute for CP/M as a software bus, a medium for program
exchange. After long and careful deliberation………..I’ve finally
taken the step and made a small investment in Digital Research’s
wonder. As a result, I thought I’d share a few comments with you and
the readers who might be thinking of taking a similar step. First of all, I'd like to make my position perfectly clear. I would most definitely not recommend CP/M to someone who already has HDOS as their main disk operating system. CP/M makes a good developmental system, but that's as far as it goes. For normal day-to-day activity I don't think there's any comparison, especially since so many of the previously exclusive "bells and whistles" like one-step boot and submit (automatic job streams that load and execute programs upon boot-up) are now available under HDOS. Finally, don't think I'm "down" on CP/M. I'm not. For some things it's great. For others, it's cumbersome. Some things we've become accustomed to under Gordon Letwin's masterpiece just aren't there at all. Don't forget, CP/M is basically over four years old (this was written in 1980), and JGL had it to use as a model, both for the good and the bad. The two approaches to operating system design are quite different, and different goals were in the creators' minds. Remember that when you compare the two."

But Dave Jaffe writes: "As a long time computer person like D.C. Shoemaker, I would like to make my opinions known in regard to the utility of CP/M. The comments he makes could be misleading to those who are or will soon be in a position to make a choice whether to use (or not use) CP/M.....Without a doubt, the worthiness of CP/M is beyond question.....It is fair to say that if a new user wants to dive into the depths and breadth of software languages, assembly language, business programs, games, or just learning -- CP/M is definitely the way to go.....In conclusion, most of the objections that Mr. Shoemaker has are either without foundation or can be easily worked around. A wise computer hobbyist will soon come to realize the advantages of running CP/M on his system."

Then from Burton Hulland: "...(By the way, I have occasion to use CP/M in some of my work and can testify that HDOS is a much more versatile operating system. The only advantage of CP/M is the greater amount of software available for it.)"

And from Rick Lutowski: "Given the number of articles published in BUSYL advocating the merits of CP/M, I was pleased to note Burton Hulland's parenthetical endorsement of HDOS in issue #34. Like Burton, I also use a version of CP/M at work and, in addition, have been exposed to a new "UNIX-like" operating system. When comparing these two operating systems, it is apparent that CP/M is a "1st generation" operating system, while the UNIX-like software represents an advanced "2nd generation" of development. Thus it was with some amazement that I discovered the similarities (or lack thereof?) between HDOS and these other two operating systems.....it should be apparent that HDOS incorporates many "2nd generation" developments lacking in earlier operating systems like CP/M."

Finally, Tom Janderson to Jerry Zuckerman, on MicroNet: "Jerry - MEVER WOULD'VE BELIEVED THAT SUCH A CONTROVERSY COULD ARISE OVER SUCH THINGS AS THESE TWO OPERATING SYSTEMS. AS WITH ANYTHING ELSE, NEITHER SYSTEM IS ACTUALLY 'BETTER' OR MORE PROFESSIONAL THAN THE OTHER! (BESIDES, HAVING WORKED ON 'PROFESSIONAL MINIS AND LARGE-FRAMES FOR THE LAST DECADE, I CAN TELL YOU THAT BOTH ARE SUPERIOR TO MANY OF THE PROS). CP/M IS A NON-PROTECTIVE SYSTEM INTENTIONALLY, IT'S INTENDED MAINLY FOR THE ASSEMBLY PROGRAMMER (WHO GETS PERTURBED WHENEVER THE SYSTEM FIGHTS HIM WITH NON-RETRIEVABLE ERRORS). HDOS IS INTENDED FOR MAXIMUM PROTECTION OF THE SYSTEM FROM THE USER, AND TO BE EASY TO LEARN (WHICH CP/M ISN'T). BOTH SYSTEMS HAVE BOTH FLIGHTS OF GENIUS AND STRAINS OF FOOLISHNESS IN THEM (WHICH THEY KNOW AND ARE STUCK
WITH>...SO WHAT THE HEY!! I USE BOTH AS EACH IS INTENDED...BUT I SURE
HATE TO SEE EITHER OF THEM KNOCKED!!" [6]

So now you know! Or do you??

REFERENCES
1. "Turnkey" Operations with HDOS Version 1.6
2. D. C. Shoemaker Puts CP/M in Perspective...BUSS Iss. 27, Pg. 2
3. CP/M Comments & H89 Modifications...........BUSS Iss. 28, Pg. 5
4. Hulland Disk Operating System for H8/H89...BUSS Iss. 34, Pg. 4
5. Rick Kutowski Compares HDOS and CP/M........BUSS Iss. 37, Pg. 7
6. HUGBB on MicroNet............................REMark Iss. 16, Pg. 20

HELP! COLUMN

I use PIP (CP/M 2.2.02) to print my Pascal source listings, with
a command like the following:

    PIP LST:=B:PROGRAM1.PAS[PNT8]

The parameters in square brackets have the following meanings:

    P = insert form feed every 60 lines;
    N = add line numbers with leading zeros;
    T8 = expand tabs to every 8th column.

The problem is that the P parameter causes PIP to generate an
unnecessary initial form feed. In other words, after I set up my
printer so it is aligned at the top of a page, then enter the PIP
command, PIP causes the printer to skip to the top of the next page
before starting to print. Does anyone know of a patch to PIP to
eliminate this initial form feed?

Jim Scott

Assembly Language Programming - Part I
by Jim Scott

Introduction

This is the first of a series of articles which will 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.

Before writing an assembly language program, you have to know
three things:

1. What processor (CPU) will you be programming for?

The answer to this question consists of more than just the
information that you will be programming for an 8080A (which these
articles will assume), a 280, an 8086, a 68000, or whatever. It
must also include a lot of the information to be found in the
manual for that processor. In other words, you must know a lot
(but not necessarily everything) about the registers, memory
organization, and instruction set (machine language) of the
processor. In general, each assembly language instruction
corresponds to one machine language instruction.

2. What operating system (if any) will your program run under?

In general, you will design your program to run under a particular
operating system. That is, when you execute your finished
program, you will boot up the computer (unless it is already up)
by loading the operating system from disk. Then you will enter
a command to request the operating system to load in your program
and to start it running. As your program runs, it will "talk to"
the operating system to request it to perform certain types of
tasks, such as reading a disk record or displaying a character on
the terminal screen.

These articles will deal with two operating systems, CP/M and HDOS
(or as Parks Watson would say, HDOS and CP/M). Although it is
possible to write assembly language programs that are designed to
run "standalone" (not under any operating system), it is unusual.
An operating system itself is a standalone program written in
assembly language.

As with question 1, question 2 implies a lot of knowledge. Along
with knowing the name of the operating system, you must know
something (but not necessarily everything) about the interfaces
designed into the operating system for use by assembly language
programs. This information is found in the manual for the
operating system.

3. Which assembler will you use?

The assembler is the program which will take your source program
as its input, and create an executable program as its output.
There is more than one assembler available for CP/M, and more than
one for HDOS. The reason you must know which assembler you will
use is that different assemblers, even for the same operating system,
often have some differences as to what format they expect
your source program to be in. These articles will deal with the
assemblers that come free with CP/M and HDOS (both named ASM).

The information required to use a particular assembler will be in
the documentation for that assembler. Note that the assembler
documentation will not include any of the answers for questions 1
and 2 above.

It's important to remember that an assembler is a program, written
by a programmer (or programmers). The programmer made some
choices that may have been different from those made by the author
of some other assembler. Some assemblers offer features not
offered by others. These differences are just something that you
need to adapt to.

The Process

Once you have designed the logic of your program, the next steps
(under CP/M, and similar for HDOS) are these:

1. Code the source program, in assembly language, using pencil and
2. Key the source program, using a text editor. This creates a disk file. The file type should be ASM.

3. Execute the assembler. Its input will be the source program on disk. Its output will be a file with a file type of HEX.

4. Execute the program called LOAD. This program reads the HEX file and creates an executable COM file.

5. Execute the new program, and see if it works properly. If it doesn't, use the text editor to correct the source program, and go back to step 3.

If you have a printer (and it's almost a necessity for writing programs of any size), step 3 can produce what's called an assembly listing. The rest of this article will deal with this listing, while further explaining some ideas introduced above.

The Assembly Listings

Elsewhere in this newsletter are two assembly listings, one for CP/M and one for HDOS, for essentially the same program. The purpose of the program, which we don't really need to worry about at this point, is to get the user to type a character on the keyboard; then the program puts the terminal into graphic mode and displays the graphic version of that character.

Starting at the left of either listing is the address field, which shows where in memory the instruction or data on that line of the listing will appear when the program is executed. The CP/M assembler lists the address in hex (four hex digits); the HDOS assembler lists them in split octal. (This is a choice made by the people who wrote the assemblers.)

The second field from the left is the machine-language representation of the instruction or data. The CP/M assembler lists it in hex, up to 10 hex digits (five bytes). The HDOS assembler lists it in octal, up to six octal digits (three bytes) per line. This shows what the program will look like in memory when it is executed. In fact, the basic purpose of the assembler is to produce this machine-language program from the source program statements.

The third field from the left on the HDOS listing is a sequence number field, showing the sequence of the source program statements. The CP/M listing does not include this field. (Another decision made differently by different programmers.)

The rest of the page shows the source program itself. In other words, everything mentioned so far is produced by the assembler. Everything to the right of this point is simply a listing of the source program that you created.

If the leftmost byte of the source program line is a semi-colon (CP/M assembler) or an asterisk (HDOS assembler), the whole line is a comment. Otherwise, the source program line is divided into fields as follows.

The leftmost field in the source program is the label field. This field can be blank, or can contain a name which will be
associated with the memory address of that statement. For example, the label CLEAR is a name for the hex address 010C (CP/M listing), or for the octal address 042.214 (HDOS listing).

The next field is the operation field. It contains the name, or mnemonic, that specifies what the instruction does. For example, MVI represents a "move immediate" instruction. CALL and SCALL are instructions that call subroutines.

The next field is the operand field. It may be blank, or it may contain one or more operands for the instruction to operate on.

The last field is the comment field, and may be blank. In the CP/M listing, comments must start with a semi-colon. In the HDOS listing, they only need to be separated from the operand field by at least one space.

The instructions, other than comment lines, fall into two categories: the machine operation and the assembler directive. A machine operation will be assembled into a machine language instruction, and will be executed when the finished program is executed. An assembler directive, also called a pseudo operation, tells the assembler something about how to assemble the machine operation instructions; it does not become a machine language instruction.

The machine operations in these two listings are CALL, MVI, RET, STA, LDA, MOV, JMP, POP, CPI, JZ, INX, PCHL, PUSH, SCALL, and JC. The assembler directives are EQU, ORG, DB, DS, and END.

Individual types of instructions will be discussed in the next discussion and article.
0000 = BASE EQU 0H ;BASE ADDRESS OF CP/M.
0100 = TPA EQU BASE+100H ;TRANSIENT PROGRAM AREA.
0000 = BOOT EQU BASE+0H ;RE-BOOT ENTRY POINT.
005 = BDOS EQU BASE+5H ;BDOS ENTRY POINT.
J100 = ORG TPA

001B = ESC EQU 027 ;DEFINE THE ESCAPE CHAR.
0001 = READC EQU 1 ;READ CONSOLE INTO (A).
0002 = TYPEC EQU 2 ;WRITE TO CONSOLE FROM (E).

0100 CD0C01 START CALL CLEAR
0103 CD1E01 CALL INPUT
0106 CD3C01 CALL PRINT
0109 CD7F01 CALL DONE

010C 1E1B CLEAR MUI E,ESC ;PUT ESCAPE IN REGISTER E.
010E 0E02 MUI C.TYPEC
0110 CD6500 CALL BDOS ;SEND IT.
0113 1E45 MUI E,`E` ;PUT LETTER `E` IN REG. E.
0115 0E02 MUI C.TYPEC
0117 CD0500 CALL BDOS ;SEND IT FOR ESCAPE E.
011A C9 RET ;RETURN TO MAIN PROGRAM.

011B CD8201 INPUT CALL TVPTX ;PRINT THE NEXT SENTENCE.
011E 496E707574 DB `Input a character: `,0
132 0E01 LOOP1 MUI C.READC
0134 CD6500 CALL BDOS ;GET THE NEXT CHARACTER.
0137 323B01 STA CHAR ;STORE CHAR IN MEMORY.
013A C9 RET ;RETURN TO MAIN PROGRAM.
013B CHAR DS 1 ;RESERVE A SPACE FOR CHAR.

013C CD8201 PRINT CALL TVPTX ;PRINT THE NEXT SENTENCE.
013F 5468652063 DB `The cor. graph. char. is `,0
0159 1E1B MUI E,ESC ;PUT ESCAPE IN REGISTER E.
015B 0E02 MUI C.TYPEC
015D CD0500 CALL BDOS ;SEND IT.
0160 1E46 MUI E,`F` ;PUT LETTER `F` IN REG. E.
0162 0E02 MUI C.TYPEC
0164 CD0500 CALL BDOS ;SEND IT FOR ESCAPE F.
0167 3A3B01 LDA CHAR ;PUT THE CHAR
016A 5F MOV E,A ;INTO REG. E.
016B 0E02 MUI C.TYPEC
016D CD0500 CALL BDOS ;SEND IT.
0170 1E1B MUI E,ESC ;PUT ESCAPE IN REG. E.
0172 0E02 MUI C.TYPEC
0174 CD0500 CALL BDOS ;SEND IT.
0177 1E47 MUI E,`G` ;PUT LETTER `G` IN REG. E.
0179 0E02 MUI C.TYPEC
017B CD0500 CALL BDOS ;SEND IT FOR ESCAPE G.
017E C9 RET ;RETURN TO MAIN PROGRAM.

017F C30000 DONE JMP BOOT ;RETURN TO CP/M.
; DISPLAY MESSAGE ON CONSOLE.
; MESSAGE TEXT FOLLOWS CALL STATEMENT, AND ENDS WITH
; A BYTE OF 0.
; RETURN IS TO INSTRUCTION FOLLOWING 0 BYTE.
; REGISTER H,L,A ARE DESTROYED.

0182 =  
0182 E1  TYPTX EQU $  ;ADDRESS OF MSG IN H,L.
0183 =  
0183 7E  TYPTX1 EQU $  
0184 FE00  MOV A,M  ;PUT NEXT CHAR. INTO REG. A.
0186 CA9001  CPI 0  ;IS IT ZERO?
0188 CD9201  JZ TYPTX2  ;YES, QUIT.
018C 23  CALL DISPLAY  ;SEND IT.
018D C38301  JMP TYPTX1  ;LOOP TO SEND NEXT CHAR.
0190  
0190 23  TYPTX2 EQU $  
0191 E9  INX H  ;ADD 1 TO H,L (NEXT CHAR).
0192  
0192 E5  PCHL  ;RETURN TO BYTE AFTER 0.
0193 D5  
0194 C5  
0195 5F  
0196 0E02  
0198 CD0500  CALL BDOS  ;SENT IT.
019B C1  
019C D1  
019D E1  
019E C9  
019F END

; DISPLAY ON CONSOLE THE CHARACTER IN (A).

0192 =  
0192 E5  DISPLAY EQU $  
0193 D5  PUSH H  
0194 C5  PUSH D  
0195 5F  PUSH B  
0196 0E02  MOV E,A  
0198 CD0500  CALL C.TYPEC  
199B C1  POP B  
199C D1  POP D  
199D E1  POP H  
199E C9  RET
Our first ASM program

HEATH ASM #104.06.00
31-Mar-83 Page 1

042.200 00002  USERFWA EQU 042200A LOCATION OF USERFWA
042.200 00003  ORG USERFWA
000.033 00004  * 000.000 00005  *
000.000 00006  ESC EQU 027  DEFINE THE ESCAPE CHAR.
000.002 00007  .EXIT EQU 000Q  POINTER TO .EXIT SCALL
000.001 00008  .SCOUT EQU 002Q  POINTER TO .SCOUT SCALL
000.007 00009  .SCIN EQU 001Q  POINTER TO .SCIN SCALL
000.010 00010  .CLRCO EQU 007Q  POINTER TO .CLRCO SCALL
031.136 00011  TYPTX EQU 031136A LOC. OF TYPTX ROUTINE
000.012 00012  *
000.013 00013  *

042.200 315 214 042 00014  START CALL CLEAR
042.203 315 225 042 00015  CALL INPUT
042.206 315 266 042 00016  CALL PRINT
042.211 315 351 042 00017  CALL DONE
000.018 00018  *
000.019 00019  *

042.214 076 033 00020  CLEAR MVI A,ESC PUT ESCAPE IN REGISTER A
042.216 377 002 00021  SCALL .SCOUT SEND IT
042.220 076 105 00022  MVI A,'E' PUT LETTER 'E' IN REG. A
042.222 377 002 00023  SCALL .SCOUT SEND IT FOR ESCAPE E
042.224 311 00024  RET RETURN TO MAIN PROGRAM
000.025 00025  *
000.026 00026  *

042.225 315 136 031 00027  INPUT CALL $TYPTX PRINT THE NEXT SENTENCE
042.230 111 156 160 00028  DB 'Input a character: ',200Q
165 164 040
141 040 143
150 141 162
141 143 164
145 162 072
040 200
042.254 377 001 00029  LOOP1 SCALL .SCIN GET THE NEXT CHARACTER
042.256 332 254 042 00030  JC LOOP1 GO BACK IF NO CHAR. YET
042.261 062 265 042 00031  STA CHAR STORE CHAR. IN MEMORY
042.264 311 00032  RET RETURN TO MAIN PROGRAM
000.033 00033  CHAR DS 1 RESERVE A SPACE FOR CHAR
000.034 00034  *
000.035 00035  *

042.266 315 136 031 00036  PRINT CALL $TYPTX PRINT THE NEXT SENTENCE
042.271 124 150 145 00037  DB 'The cor. graph. char. is ',200Q
040 143 157
162 056 040
147 162 141
160 150 056
040 143 150
141 162 056
040 151 163
040 200
042.323 076 033 00038  MVI A,ESC PUT ESCAPE IN REGISTER A
042.325 377 002 00039  SCALL .SCOUT SEND IT
042.327 076 106 00040  MVI A,'F' PUT LETTER 'F' IN REG. A
042.331 377 002 00041  SCALL .SCOUT SEND IT FOR ESCAPE F
042.333 072 265 042 00042  LDA CHAR PUT THE CHAR. IN REG. A
042.336 377 002 00043  SCALL .SCOUT SEND IT
042.340 076 033 00044  MVI A,ESC PUT ESCAPE IN REG. A
Our first ASM program

<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>042.342</td>
<td>377 002</td>
<td>SCALL .SCOUT SEND IT</td>
</tr>
<tr>
<td>042.344</td>
<td>076 107</td>
<td>MVI A, 'G' PUT LETTER 'G' IN REG. A</td>
</tr>
<tr>
<td>042.346</td>
<td>377 002</td>
<td>SCALL .SCOUT SEND IT FOR ESCAPE G</td>
</tr>
<tr>
<td>042.350</td>
<td>311</td>
<td>RET RETURN TO MAIN PROGRAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00049 *</td>
</tr>
<tr>
<td>042.351</td>
<td>377 007</td>
<td>DONE SCALL .CLRCO CLEAR BUFFER</td>
</tr>
<tr>
<td>042.353</td>
<td>076 000</td>
<td>MVI A, 0 ZERO IN REG. A=NOR. EXIT</td>
</tr>
<tr>
<td>042.355</td>
<td>377 000</td>
<td>SCALL .EXIT RETURN TO HDOS</td>
</tr>
<tr>
<td>042.357</td>
<td>000</td>
<td>END START TELLS ASM WHERE STRT&amp;STP</td>
</tr>
</tbody>
</table>

00054 Statements Assembled
32494 Bytes Free
No Errors Detected