GAZETTE

The Official Newsletter of the

Richmond Heath Users Group

Volume I Issue 16

SEPTEMBER 1983

Subscriptions: $5.00 per calendar year.
Editor: Jim Scott, 3464 Northview Place, Richmond, VA 23225.
Next editorial deadline: October 3, 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 Monday, September 19, 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 5039 (five zero three nine).

Everyone is welcome!

MINUTES (Meeting of August 15, 1983)

Present: Carlos Chafin, David Harrington, John Purcell, Jim Scott, Hank Steigleder, Nelson Trinkle, Parks Watson and Gary West.

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

Chafin opened the meeting by commenting on several items of interest, among them, that the August issue of BYTE was a special on "C" language.

Following the usual informal discussion period, Watson demonstrated the HDOS ASseMbler, using the programs by Pat Swayne presented in REMark Iss. 39 & 41. Unfortunately, no one brought the CP/M versions of the same programs from REMark Iss. 40 & 42 to the meeting, so the CP/M ASseMbler could not be demonstrated. Fortunately, however, "Heathcliffe" can still proudly proclain his monogamous status (HDOS only), having never suffered an incursion by CP/M, although he was prepared to make the supreme sacrifice at this meeting.

After several futile attempts by an unnamed warrior to sink the SEABATTLE carrier, the meeting was adjourned at 9:30 PM.

Parks Watson

Secretary/Treasurer

****

FOR THE 4H CLUB

Hard Sectored, Hip-shooting, HDOS Hombres
(Who would rather fight then switch)

From Parks Watson:

Now that "Heathcliffe" is back on line and feeling his oats, let's get on with it!

While membership in our prestigious club isn't limited to those with an H89, not over 48K memory and one disk drive, the majority of our members (the total escapes me at the moment) are probably in that category. With that in mind, let's investigate some ideas that will help us live with our limitations.

If you have only one drive, there is no alternative to a lot of disk swapping. But it doesn't have to be as awkward and cumbersome as Heath documentation implies. If we follow the Heath manual precisely we will SYSGEN every disk we use. On page 1-60 of the Software Reference Manual we read, "Only system volumes can be used to perform Bootstrap, so if you have a one-drive system, you will probably want to SYSGEN all of your diskettes." I was puzzled by the "probably want to" in this statement. What if I didn't want to? There is no hint anywhere in the manual that it shouldn't read "must SYSGEN all of your diskettes."

If we SYSGEN every disk, we limit the amount of usable space for our programs. And if we want to change disks, we have to type "BYE" and reboot. So we have these two problems that Heath doesn't help us with in their standard documentation.

There are several ways we can increase available free space, and the reboot hassle can be eliminated completely, although, with one drive, we can't avoid disk swapping. First, of course, you can free some space by deleting any files that are not absolutely essential. By using the /MIN switch when you SYSGEN your disks (type SYSGEN/MIN), you will have omitted copying all non-essential files, leaving 248 sectors free for your programs. Also, the size of some system files can be reduced without damaging the system (more on this next time). But the most effective help for both problems is "STAND-ALONE" operation. If you are not already using this mode of operation, you have surely read about it in one or more of the Heath-related publications. There is an article on this subject every few months but they usually seem to be written assuming a level of knowledge on the part of the reader which would obviate the need for the article in the first place and nearly always include the statement that "it is undocumented and probably unsupported by Heath", which is likely to make the novice shy away from the whole idea. But Heath does say on page 2-66 of the Software Manual, "Experimentation can cause no harm....do not be timid about exploring and enjoying the capabilities of HDOS". So let's experiment.

Running in STAND-ALONE mode will not only leave you more free space by allowing you to use a disk that has been INITialized but not SYSGENed, but will also allow you to switch disks without rebooting.

To demonstrate the advantage of STAND-ALONE operation, let's first try a command in the normal mode of operation. Boot a disk that has SET.ABS on it (SYSGEN/MIN doesn't copy SET.ABS from your System Volume). Now if you type DISMOUNT SY0: (which is always MOUNTed by the boot procedure), it will dismount the disk AND return you to the boot routine. Try it. It will do just what you asked it to do - dismount the disk in SY0:, but in the process you lose HDOS. Just as if you had typed BYE.

Now, reboot the disk with SET.ABS. Then type SET HDOS STAND-ALONE. Don't omit the hyphen. HDOS will respond: "It is Now Pitch Dark. If You Proceed, You Will Likely Fall Into a Pit." This sounds pretty ominous but since I had fallen into quite a few pits before I ever heard of STAND-ALONE, I wasn't overly intimidated by this warning. If you want to back up and take a few minutes to muster your confidence, type SET HDOS NOSTAND-ALONE, and HDOS will dismount the disk and you will be back where you started.

I assume you are all venturesome types, so repeat the above and you will notice that after the dire warning you still have the HDOS prompt, ">". Again type DISMOUNT SY0:. Now HDOS will dismount the disk in SY0: and return to the HDOS prompt. You can remove the disk you booted up on and replace it with any INITialized disk, even if it hasn't been SYSGENed. It can be a SYSGENed disk, but doesn't have to be. For now, insert any SYSGENed disk you have and type MOUNT SY0: and the system will respond with the same message as if you had just booted the new disk. But EUREKA!, you have changed disks without rebooting! You can now use this disk just as if you had booted it and you can change disks as often as you like, using the DISMOUNT/MOUNT sequence. Alternatively, you can you use the command RESET SY0:, which DISMOUNTs the disk in SY0:, requests that you change disks and then MOUNTs the new disk when you close the drive door. This reduces the procedure to a single command and also reminds you to change the disk, or at least open and close the drive door, in which case the same disk will be mounted again. The DISMOUNT command doesn't respond with this reminder. If you attempt to continue input from the keyboard without MOUNTing a disk you will get only error messages (HDOS doesn't know there is a disk in SY0: unless it is MOUNTed). After you become comfortable with using STAND-ALONE operation and begin using it a lot, it is easy, in a flurry of disk swapping, to get out of sync with your commands and swap disks before the disk in the drive is dismounted. For this reason, in addition to the one-command advantage, it is preferable to use the RESET command rather than the DISMOUNT/MOUNT sequence, as you will then be reminded when it is time to swap disks.

Now initialize a disk but don't SYSGEN it. Copy SYSCMD.SYS and PIP.ABS onto it from your System Disk. A disk to be used in STAND-ALONE mode must be initialized or it cannot be mounted. For all practical purposes it must also have SYSCMD.SYS and PIP.ABS on it. Without PIP, you cannot list a file or the directory (won't execute the TYPE, CAT or any COPY commands). With PIP on the disk, but not SYSCMD.SYS, you can MOUNT the disk and execute a TYPE or CAT command but when it tries to return to HDOS, without SYSCMD it returns to the BOOT routine. Thus, only one command can be executed without rebooting on a system disk and starting all over with RESET or DISMOUNT/MOUNT.

Again boot the disk with SET.ABS on it. Type RESET SY0: and replace it with the disk you initialized as above. It will be MOUNTed and you will be running without a system disk! Note that you did not

have to type SET HDOS STAND-ALONE this time. When you typed this command before, it set a flag and copied it to the disk and it remains there until the SET HDOS NOSTAND-ALONE command is given.

Now type CAT and the directory will be listed and "336 Sectors Free", or 88 sectors more than the SYSGEN/MIN disk has. You can now copy BASIC, ASM or any other program to this disk just as you had been using your SYSGENed disks before. Also, notice that when you are using ONECOPY, when you CTRL-D out of ONECOPY, you are returned to the HDOS prompt instead of to the boot routine as is the case when not using the STAND-ALONE mode. You can still use BYE as before to remove the disk in your drive and power down. While it isn't always necessary, it is always safe to reset SY0: and replace the disk with a SYSGENed disk before BYEing.

Next time we will look at some other schemes to save additional disk space and also some precautions which we should be aware of as we override some of the built-in features of HDOS which "protect it from the user".

****

ASSEMBLY LANGUAGE PROGRAMMING - PART 4

by Jim Scott

INTRODUCTION

This is the fourth 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 (August issue of the Gazette) described the Data Transfer Group of instructions for the 8080 CPU. 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.

ARITHMETIC GROUP

Instructions in this group perform arithmetic operations (that's the adjective "arithmetic", not the noun "arithmetic") on data in registers and memory. For the 8080, an arithmetic instruction does addition or subtraction. Other CPUs have instructions that do multiplication and division, but the 8080 does not.

In general, the arithmetic instructions affect the Zero, Sign, Parity, Carry, and Auxiliary Carry flags. These flags are individual bits of register F, the Flag Word, as described in the first article in this series. The flags are set as follows:

Zero (Z): Set to 1 if the result of the arithmetic operation is zero; reset to 0 if the result is non-zero.

Sign (S): Set to 1 if the high-order (left-most) bit of the result is 1; reset to 0 if the high-order bit of the result is 0. Usually, if the Sign flag is 1, the result is negative; if the Sign flag is 0, the

result is zero or positive.

Parity (P): Set to 1 if the number of one-bits in the result is even; reset to 0 if the number of one-bits is odd.

Carry (Cy): Set to 1 if an addition resulted in a carry out of the high-order bit, or if a subtraction resulted in a borrow out of the high-order bit; otherwise reset to 0.

Auxiliary Carry (AC): Set to 1 if the instruction resulted in a carry out of bit 3, into bit 4; otherwise reset to 0. This flag is rather obscure, and is generally used in conjunction with the DAA (Decimal Adjust Accumulator) instruction.

The individual instructions are as follows.

ADD (Add Register)

Format: ADD r
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 adds the one-byte contents of register or memory location r to the one-byte contents of register A (the accumulator). The result goes into register A.

Example: ADD E
This adds the contents of register E to the contents of register A, and puts the result into register A.

Note that the instruction ADD A would double the value in register A.

ADI (Add Immediate)

Format: ADI data
This instruction adds a byte of immediate data (the contents of the second byte of the machine language instruction itself) to the contents of register A. The result goes into register A.

Example: ADI 4
This adds 4 to the contents of register A, and puts the result into register A.

ADC (Add Register With Carry)

Format: ADC r
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 adds the one-byte contents of register or memory location r, and the contents (1 or 0) of the Carry flag, to the one-byte contents of register A. The result goes into register A.

Example: ADC B
This adds the contents of register B, plus 1 if the Carry flag is set, to the contents of register A, and puts the result into register A.

This instruction is designed to be used for calculating the high-order half of a 16-bit addition. You would first use ADD to calculate the low-order half; this would leave the Carry flag set if a carry occurred. Then you would use ADC to calculate the high-order half;

the carry, if any, would automatically be included. See also the DAD instruction.

ACI (Add Immediate With Carry)

Format: ACI data
This instruction adds a byte of immediate data (the contents of the second byte of the machine language instruction itself), and the contents (1 or 0) of the Carry flag, to the contents of register A. The result goes into register A.

Example: ACI 3FH
This adds the hex value 3F, plus 1 if the Carry flag is set, to the contents of register A, and puts the result into register A.

This instruction is designed to be used for calculating the high-order half of a 16-bit addition. You would first use ADI to calculate the low-order half; this would leave the Carry flag set if a carry occurred. Then you would use ACI to calculate the high-order half; the carry, if any, would automatically be included.

SUB (Subtract Register)

Format: SUB r
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 subtracts the one-byte contents of register or memory location r from the one-byte contents of register A. The result goes into register A.

Example: SUB L
This subtracts the contents of register L from the contents of register A, and puts the result into register A.

Note that the instruction SUB A would set register A to zero.

SUI (Subtract Immediate)

Format: SUI data
This instruction subtracts a byte of immediate data (the contents of the second byte of the machine language instruction itself) from the contents of register A. The result goes into register A.

Example: SUI 'A'
This subtracts the value of an ASCII letter 'A' (65, or hex 41), from the contents of register A, and puts the result into register A.

SBB (Subtract Register With Borrow)

Format: SBB r
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 subtracts the one-byte contents of register or memory location r, and the contents (1 or 0) of the Carry flag, from the one-byte contents of register A. The result goes into register A.

Example: SBB C
This subtracts the contents of register C, plus 1 if the Carry flag is set, from the contents of register A, and puts the result into register A.

This instruction is designed to be used for calculating the high-order half of a 16-bit subtraction. You would first use SUB to calculate the low-order half; this would leave the Carry flag set if a borrow occurred. Then you would use SBB to calculate the high-order half; the borrow, if any, would automatically be included.

SBI (Subtract Immediate With Borrow)

Format: SBI data
This instruction subtracts a byte of immediate data (the contents of the second byte of the machine language instruction itself), and the contents (1 or 0) of the Carry flag, from the contents of register A. The result goes into register A.

Example: SBI 071Q
This subtracts the octal value 071, plus 1 if the Carry flag is set, from the contents of register A, and puts the result into register A.

This instruction is designed to be used for calculating the high-order half of a 16-bit subtraction. You would first use SUI to calculate the low-order half; this would leave the Carry flag set if a borrow occurred. Then you would use SBI to calculate the high-order half; the borrow, if any, would automatically be included.

INR (Increment Register)

Format: INR r
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 adds 1 to the contents of register or memory location r, and puts the result into register or memory location r. The Carry flag (Cy) is not affected. The maximum value is 255; if a value of 255 is incremented, the result is zero.

Example: INR M
This adds 1 to the value in the memory location whose address is in register pair HL, and puts the result into that same memory location.

This instruction is frequently used for keeping count of how many times something happens. For example, if you wrote a program to read one byte at a time from the terminal keyboard until Return is pressed, you could execute an INR C instruction each time a key was pressed. Once a carriage return was detected, register C would contain the number of characters entered at the keyboard (unless you forgot to set register C to zero before starting).

DCR (Decrement Register)

Format: DCR r
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 subtracts 1 from the contents of register or memory location r, and puts the result into register or memory location r. The Carry flag (Cy) is not affected. The maximum value is 255; if a value of zero is decremented, the result is 255.

Example: DCR B
This subtracts 1 from the value in register B, and puts the result into register B.

This instruction is frequently used for making sure that a certain set of instructions is executed a certain number of times. For example, if you wrote a program to display your name 10 times on the terminal screen, you could start by putting a value of 10 into register D. Each time you display your name, execute a DCR D instruction, and test to see if it has gotten to zero yet. When it does, quit displaying.

INX (Increment Register Pair)

Format: INX rp
where rp can be B, D, H, or SP, representing register pairs BC, DE, HL, and the stack pointer, respectively. This instruction adds 1 to the 16-bit value in register pair rp, and puts the result into register pair rp. No flags are affected. The maximum value is 65535; if a value of 65535 is incremented, the result is zero.

Example: INX B
This adds 1 to the value in register pair BC, and puts the result into register pair BC.

This instruction is used much as INR is used, except that INX affects a pair of registers instead of a single register, and it can count up to 65535 instead of just 255.

DCX (Decrement Register Pair)

Format: DCX rp
where rp can be B, D, H, or SP, representing register pairs BC, DE, HL, and the stack pointer, respectively. This instruction subtracts 1 from the 16-bit value in register pair rp, and puts the result into register pair rp. No flags are affected. The maximum value is 65535; if a value of zero is decremented, the result is 65535.

Example: DCX H
This subtracts 1 from the value in register pair HL, and puts the result into register pair HL.

This instruction is used much as DCR is used, except that DCX affects a pair of registers instead of a single register, and it can count up to 65535 instead of just 255.

DAD (Add Register Pair to HL) (or Double Precision Add)

Format: DAD rp
where rp can be B, D, H, or SP, representing register pairs BC, DE, HL, and the stack pointer, respectively. This instruction adds the 16-bit contents of register pair rp to the 16-bit contents of register pair HL, and puts the result into register pair HL. Only the Carry flag (Cy) is affected; it is set to 1 if there is a carry out of the double precision add; otherwise it is reset to 0. The maximum value is 65535.

Example: DAD D
This adds the contents of register pair DE to the contents of register pair HL, and puts the results into register pair HL.

This instruction can be used to do 16-bit addition without having to write a subroutine to do it. It is frequently used to calculate a memory address; for example, if HL contains a base address, like the origin of an array, and BC contains an offset, like the address of an

array element relative to the origin, than a DAD B instruction will result in HL containing the absolute address of the array element. The reason why the result goes into HL is that a subsequent instruction with an operand of M will probably be used to refer to the memory address.

Note that a DAD H instruction will double the value in HL.

DAA (Decimal Adjust Accumulator)

Format: DAA

This instruction adjusts the 8-bit number in register A to form two 4-bit BCD digits. A BCD digit is defined as a decimal digit from 0 to 9, represented in binary by the values 0000 to 1001. The rules followed by this instruction, which may seem less than obvious, are as follows:
1. First, if the value of the least significant 4 bits of register A is greater than 9, or if the Auxiliary Carry flag (AC) is set to 1, then 6 is added to register A.
2. Second, if the value of the most significant 4 bits of register A is now greater than 9, or if the Carry flag (Cy) is set, 6 is added to the most significant 4 bits of register A.

Example: DAA

The purpose of the DAA instruction is to finish the process of adding two BCD numbers. A BCD number, in this context, is one in which each group of four bits represents a decimal digit from 0 to 9. For example, if register A contains a hex value of 24, representing a decimal number of 24, and register B contains a hex value of 18, representing a decimal number of 18, then the result of ADD B is a hex value of 3C in register A. Also, this particular addition would leave the AC and Cy flags reset to zero. Then the DAA instruction will do the following:
1. Since the value, C, of the least significant four bits is greater than 9, 6 is added to register A. The result is 3C + 6 = 42.
2. Since the value of the most significant 4 bits is not greater than 9, and the Cy flag is not set, no further adjustment is made.
Now register A contains a hex value of 42, which represents a decimal value of 42 (not a hex value of 42), which is indeed the sum of the decimal numbers 24 and 18.

CONCLUSION

Future issues will cover the remaining groups of instructions.