MEETING NOTICE

The next meeting will be Monday, October 17, at 7:30. The meeting location is Alpha Audio's third floor conference room, at 2049 West Broad Street. The nighttime phone number there is 358-3853. The front door has a touch-pad combination lock, and the combination for the night will be 5390 (five three nine zero).

Everyone is welcome!

MINUTES (Meeting of September 19, 1983)


We welcome aboard Charlie Gaines, our newest member who joined our group at this meeting. Charlie has an H89 and is interested in a payroll program. (HDOS or CP/M??). Any recommendations?

Chafin stated that he has been delving into the C programming language and will be prepared to discuss the subject at the October meeting.

Scott concluded the series of discussions on Assembly language programming by reviewing the use of various instructions in the CP/M version of the programs in REMark Iss. 40 and 42. He promised to complete his summaries in the newsletter of the material covered at the meetings, for the benefit of those of us with "volatile memories". These summaries will prove to be invaluable additions to our reference materials for those of us who want to continue our efforts with assembly language coding.

The meeting was adjourned at 9:30 PM.
NEWS

CHUGCON'83 ALMOST UPON US

CHUGCON'83, the second annual regional conference sponsored by the Capital Heath Users Group (CHUG), will be held on Saturday, October 22, at the Westpark Hotel in Tysons Corner, Va. A copy of the flyer and the registration form are included in this issue of the Gazette. If you have questions, you can call S. D. Quarles, CHUGCON'83 Publicity Chairman, at 1-703-354-7816. If you are interested in staying overnight at the hotel, call 1-800-533-3301 and ask for the "CHUG block".

The conference is free, but the luncheon and banquet cost money and require advance registration. To register for either meal or both, send the registration form and your check to CHUG so they receive it by Monday, October 17 (which is also the date of our upcoming meeting; i.e., it's almost here).

We may have a crew driving up from Richmond for the conference. If so, we will probably do up early Saturday morning and return relatively late Saturday night. We will finalize these plans at the meeting, but the meal registrations must be in before then.

+++++++ 

RHUG NOMINATIONS IN NOVEMBER; ELECTIONS IN DECEMBER

At our November meeting we will nominate our officers for 1984, and at the December meeting we will hold our election. The positions and current office-holders are:

President: Carlos Chafin
Vice President: Jerry Tiller
Secretary/Treasurer: Parks Watson
Software Librarian: Bobby Tulloh
Newsletter Editor: Jim Scott

Plan ahead! Consider what positions you might be interested in holding. If you are already an officer, do you want to continue? Do you need a rest? Might you be more useful in a different slot?

+++++++ 

RHUG DUES DUE IN DECEMBER

Our dues are $5.00 per calendar year, and are intended to cover the cost of mailing the Gazette. A question which will come up at the October meeting will be whether to continue the dues at $5.00. The results will be reported in the November Gazette.
MONITOR. MONITOR ON THE WALL
by Gary E. West

It's really been too long since my article on the H100 to jump right into this.

Since I completed construction of my H100 in April of this year, I have had several monitors connected to it. As mentioned in my last article, the first was the color monitor loaned to me by Carlos Chafin. This of course was the ZUM-134 by Zenith. As far as I was concerned at the time, the only fault it had was looking like a standard television set. Let's just say I'm a little more knowledgeable now but since this is not intended to be a technical comparison of monitors I will not get into data and specifications, only observations. Thank you, Carlos, your phosphor now carries the scars of my first crude etchings on a personal computer.

Well, that first chapter, with Carlos' monitor, was very short, these days I think, then I had nothing more to work with than a rumor that Zenith was coming out with a new monitor, the ZUM-135! It was May . . . I called the Heathkit store in Virginia Beach and they confirmed the rumor. I then called Papa Heath up there in Benton Harbor . . . "No such animal", he proclaimed. "Wanna buy a 134?" I smelled a rat, but dug in for the duration.

Meantime, I find out there's a relatively likeable fella named John Purcell, who just happens to be the local Zenith dealer, and a member of RHUG (a truly charitable organization). To make a long story short, John managed to dig up a ZUM-121 (green screen monitor) which he was able to lend/lease to me during May.

Well, time marches on; I went dry (no monitor) for June and July. By then Heath was admitting that there was such a thing as the ZUM-135. That was nice since the dealers for Zenith had been talking about it for some three or four months by then.

Anyhow, I got mine (the ZUM-135) in the mail (UPS, really) about the first of August. To my delight it really looked like a monitor (see how my priorities run). Of course, I think the first impression we have of something means a lot so I was really glad that it looked like a monitor.

There is nothing to using the ZUM-135, just plug in the computer (RGB or composite video) then plug in the power and turn it on. Only one small glitch, Zenith forgot the bracket on the RGB cable socket where you screw down the plug. Come on, Zenith!!

When I first turned the beast on, the letters and numbers on the screen seemed large and fuzzy. I guess that was because I was used to the ZUM-121 which has a 12" screen and uses only green in the display. The ZUM-135 has a 13" screen and uses white for the display (in the color mode). I "tweaked" a little with the controls on the monitor until the characters were a little more acceptable. Now, after using the monitor for a while, I have no problem with the characters. It must be all in what you get used to!

When using the monitor for color graphics the lines and colors appear very sharp. There is some "stepping" in the angled lines but only slight and not due to the monitor. Someday I will find out how to use "interlacing" and the steps will be much less pronounced.
I guess that I'm very satisfied with the ZUM-135 so far. I like the appearance, I'm accustomed to the characters (which no longer appear big and fuzzy to me), the color and graphics displays are great.

I have not yet used the audio capabilities of the monitor, but since I am still a virtual "rookie" at computing I'm taking things one step at a time - I'll get there.

Presently, based on my experiences to date with the ZUM-135, I highly recommend it as your next monitor.

Now to change the subject - I suppose most of you noticed I was not at the September meeting. Well, that was the beginning of a new era for me. I am moving to Gaithersburg, Maryland, to start a new job. I intend to remain a member of RHUG since you've all treated me so well. I also intend to join CHUG when settled in Maryland. I will pass along my new address to Jim Scott and/or Carlos Chafin, so that if any of you want further info on either the H100 or the ZUM-135 you can contact me.

Hope to see you on the "network" or otherwise.

******************************************************************************

ASSEMBLY LANGUAGE PROGRAMMING - PART 5
by Jim Scott

INTRODUCTION

This is the fifth 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 two articles (August and September issues of the Gazette) described the Data Transfer and Arithmetic groups of instructions for the 8080 CPU. Instructions in these groups move data between registers and memory, and perform arithmetic operations on data in registers and memory. This time we will discuss the Logical Group of instructions.

LOGICAL GROUP

Instructions in this group perform logical, comparison, and rotation operations on data in registers and memory. Let's consider these three types of operations one at a time.

Logical Operations

Logical, or Boolean, operations are those defined in symbolic logic: AND, OR, XOR, and NOT. Let's assume we have two one-bit operands called A and B. (These designations are not intended to represent the A and B registers.) We all know by this time that a bit has one of two values: 0 or 1. (Most of the instructions work with bytes, or 8-bit operands, of course; just consider each byte to be eight 1-bit operands.)
The first operation, **AND**, produces a 1 in the result if both A and B are equal to 1. If either A or B is 0, the result of AND is 0. We show all the combinations in a table as follows:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A AND B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

When two bytes A and B are ANDed, the leftmost bit of A is ANDed with the leftmost bit of B, and so forth.

The OR operation produces a 1 in the result if either A or B (or both) is equal to 1. If both A and B are 0, the result of OR is 0:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A OR B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The XOR, or **exclusive OR**, operation produces a 1 in the result if either A or B, but not both, is equal to 1. If A and B have the same value, the result of XOR is 0:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A XOR B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The NOT, or complement, operation operates on just one operand (let's say A); it inverts the value of the operand:

<table>
<thead>
<tr>
<th>A</th>
<th>NOT A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Comparison Operations**

A comparison instruction subtracts one operand from the other (without altering either operand), and sets the condition flags to indicate the results of the comparison. (More on the condition flags in a minute.)

**Rotation Operations**

A rotation instruction shifts the bit values in the accumulator (register A) to the left or to the right. Bits that would fall off the end get wrapped around to the other end.

**The Condition Flags**

In general, the logic 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 a logical or comparison operation is zero (in the case of a comparison, this means the two operands are
equal); 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. Usually this is a whoopee.

Carry (Cy): Reset to 0 by most of the logical instructions. The comparison instructions set the Carry flag to 1 if the contents of register A is less than the contents of the second operand. In the rotation instructions, the Carry flag is involved in the bit-shifting. The CMA, CMC, and STC instructions are special cases.

Auxiliary Carry (AC): In general, set to 1 if the instruction resulted in a carry out of bit 3 into bit 4; otherwise reset to 0. Most of the logical instructions either clear the Auxiliary Carry flag, or do not affect it. In other words, this is another whoopee.

The individual instructions are as follows.

**AND (AND Register)**

Format: AND 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 ANDs the one-byte contents of register or memory location r with the one-byte contents of register A (the accumulator). The result goes into register A. The Carry flag is reset to zero.

Example: AND E
This ANDs the contents of register E with the contents of register A, and puts the result into register A. Suppose E contained 00001111 and A contained 01010101; then the result in A would be 00000101. Only those bit positions that contained 1's in both operands would contain 1 in the result. In this example, the Zero flag would be 0, because the result is not zero (i.e., it's not 00000000 in bit format). The Sign flag would be 0, like the high-order flag of the result, in this case.

Note that the instruction AND A would not alter the value in register A.

This instruction is often used to mask out (turn off) certain bits of an operand. Suppose register A contains an arbitrary bit value, and you want to clear the high-order (left) half, while leaving the low-order half unchanged. If register D contains 00001111 (hex 0F), then AND D will do the job.

**ANI (AND Immediate)**

Format: ANI data
This instruction ANDs a byte of immediate data (the contents of the second byte of the machine language instruction itself) with the contents of register A. The result goes into register A. The Carry and Auxiliary flags are reset to zero.
Example: ANI 0FH
This ANDs a bit pattern of 10001111 with the contents of register A, and puts the result into register A. Suppose register A contained 10110101; then the result in register A will be 10000101.

This instruction is often used to mask out (turn off) certain bits of an operand. Suppose register A contains an arbitrary bit value, and you want to clear the high-order (left) half, while leaving the low-order half unchanged. The instruction ANI 0FH will do the job.

XRA (Exclusive OR Register)

Format: XRA 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 Exclusive-ORs the one-byte contents of register or memory location r with the one-byte contents of register A. The result goes into register A. The Carry and Auxiliary Carry flags are reset to zero.

Example: XRA B
This Exclusive-ORs the contents of register B with the contents of register A, and puts the result into register A. Suppose B contained 10010111 and A contained 11101010; then the result in A would be 01111101. Those bit positions that were different in the two operands are 1 in the result. The Zero and Sign flags would both be 0 in this case.

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

This instruction may be used to invert (i.e., to NOT, or complement, or "toggle") one or more bits in a byte. If register D contains 11111111 (hex FF), then XRA D will invert every bit in register A; what was a 0 will become a 1, and vice versa. (See the CMA instruction, also.) If the right-most bit of register A represents whose move it is in a two-person game, and if register C contains 00000001 (hex 01), then you can toggle the bit after each move in the game with the instruction XRA C.

XRI (Exclusive OR Immediate)

Format: XRI data
This instruction Exclusive-ORs a byte of immediate data (the contents of the second byte of the machine language instruction itself) with the contents of register A. The result goes into register A. The Carry and Auxiliary Carry flags are reset to zero.

Example: XRI 0A2H
This Exclusive-ORs the hex value A2 with the contents of register A, and puts the result into register A.

This instruction may be used to invert (i.e., to NOT, or complement, or "toggle") one or more bits in a byte. XRI 0FH will invert each bit in the left half of register A; what was a 0 will become a 1, and vice versa; the right half will not be affected.

ORA (OR Register)

Format: ORA 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 ORs the one-byte contents of register or memory location r with the one-byte contents of register A. The result goes into register A. The Carry and Auxiliary Carry flags are reset to zero.

Example: ORA H
This ORs the contents of register H with the contents of register A, and puts the result into register A. Suppose H contained 01101001 and A contained 10110010; then the result in A would be 11111011. Those bit positions that were 1 in either operand are 1 in the result. The Zero flag would be 0, and the Sign flag would be 1, in this case.

Note that the instruction ORA A would leave register A unchanged.

This instruction is often used to turn on certain bits of an operand. Suppose register A contains an arbitrary bit value, and you want to turn on the second and third bits from the left, while leaving the other bits unchanged. If register B contains 01100000 (hex 60), then ORA B will do the job.

ORI (OR Immediate)

Format: ORI data
This instruction ORs a byte of immediate data (the contents of the second byte of the machine language instruction itself) with the contents of register A. The result goes into register A. The Carry and Auxiliary Carry flags are reset to zero.

Example: ORI 370H
This ORs the octal value 370 (binary 11111000), with the contents of register A, and puts the result into register A. Suppose register A contained 10110110. Then the result would be 11111110. The Zero flag would be 0, and the Sign flag would be 1, in this case.

This instruction is often used to turn on certain bits of an operand. Suppose register A contains an arbitrary bit value, and you want to turn on the three right-most bits, while leaving the other bits unchanged. The instruction ORI 07H will do the job.

CMP (Compare Register)

Format: CMP 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. Register A is unchanged. The Zero flag is set to 1 if the two operands are equal. The Carry flag is set to 1 if the contents of register A are less than the contents of the second operand.

Example: CMP L
This subtracts the contents of register L from the contents of register A, without changing register A, and sets the flags according to the results. Suppose register A contained decimal 118 (hex 76) and register L contained decimal 125 (hex 7D). Then the Zero flag will contain 0 (the operands are not equal), and the Carry flag will contain 1 (118 < 125).
This instruction is usually used before a conditional jump instruction (to be covered in the next article of this series). The CMP instruction sets the flags, and the conditional jump instruction decides whether to jump based on the value of one of the flags.

**CPI (Compare Immediate)**

**Format:** CPI data

This instruction subtracts a byte of immediate data (the contents of the second byte of the machine language instruction itself) from the one-byte contents of register A. Register A is unchanged. The Zero flag is set to 1 if the two operands are equal. The Carry flag is set to 1 if the contents of register A are less than the contents of the immediate data.

**Example:** CPI 1320

This subtracts the octal value 132 from the contents of register A, without changing register A, and sets the flags according to the result. Suppose register A contained 1320. Then the Zero flag will be 1 (the operands are equal), and the Carry flag will be 0 (the contents of register A are not less than the immediate data).

This instruction is usually used before a conditional jump instruction (to be covered in the next article of this series). The CMP instruction sets the flags, and the conditional jump instruction decides whether to jump based on the value of one of the flags.

**RLC (Rotate Left)**

**Format:** RLC

This instruction shifts the contents of register A left by 1 bit. The right-most bit and the Carry flag are both set to the value shifted out of the left-most bit. The Carry flag is the only flag affected.

**Example:** RLC

Suppose register A contained 10110001. The result of this instruction will be that register A will contain 01100011, and the Carry flag will contain 1 (the same as the bit shifted around the corner).

There are various reasons for wanting to rotate the contents of a byte. To shift by more than one bit, repeat the instruction or execute it in a loop. In particular, the RLC instruction has the effect of multiplying the numeric value in register A by 2 (assuming the product still fits in a byte).

**RRC (Rotate Right)**

**Format:** RRC

This instruction shifts the contents of register A right by 1 bit. The left-most bit and the Carry flag are both set to the value shifted out of the right-most bit. The Carry flag is the only flag affected.

**Example:** RRC

Suppose register A contained 01011101. The result of this instruction will be that register A will contain 10111010, and the Carry flag will contain 1 (the same as the bit shifted around the corner).

There are various reasons for wanting to rotate the contents of a byte. To shift by more than one bit, repeat the instruction or execute it in a loop.
RAL (Rotate Left Through Carry)

Format: RAL
This instruction shifts the contents of register A left by 1 bit, through the Carry flag. The right-most bit is set to the value that was in the Carry flag, and the Carry flag is set to the value shifted out of the left-most bit. The Carry flag is the only flag affected.

Example: RAL
Suppose register A contained 00011010 and the Carry flag contained 1. The result of this instruction will be that register A will contain 00110101, and the Carry flag will contain 0 (the same as the bit shifted out of the left-most bit of register A).

Note the difference between the RLC and RAL instructions. I have always felt that these instructions should have each other's mnemonics; RLC means Rotate Left, and RAL means Rotate Left Through Carry.

There are various reasons for wanting to rotate the contents of a byte. To shift by more than one bit, repeat the instruction or execute it in a loop. In particular, if the Carry flag is initially zero (an AND A instruction, or an ORA A instruction, for example, will do this without modifying register A), the RAL instruction has the effect of multiplying the numeric value in register A by 2 (assuming the product still fits in a byte).

RAR (Rotate Right Through Carry)

Format: RAR
This instruction shifts the contents of register A right by 1 bit, through the Carry flag. The left-most bit is set to the value that was in the Carry flag, and the Carry flag is set to the value shifted out of the right-most bit. The Carry flag is the only flag affected.

Example: RAR
Suppose register A contained 10100011 and the Carry flag contained 0. The result of this instruction will be that register A will contain 01010011, and the Carry flag will contain 1 (the same as the bit shifted out of the right-most bit of register A).

Note the difference between the RRC and RAR instructions. I have always felt that these instructions should have each other's mnemonics; RRC means Rotate Right, and RAR means Rotate Right Through Carry.

There are various reasons for wanting to rotate the contents of a byte. To shift by more than one bit, repeat the instruction or execute it in a loop. In particular, if the Carry flag is initially zero (an AND A instruction, or an ORA A instruction, for example, will do this without modifying register A), the RAR instruction has the effect of dividing the numeric value in register A by 2 and discarding the remainder.

CMA (Complement Accumulator)

Format: CMA
This instruction complements (i.e., inverts, NOTs, toggles) the contents of the A register. What was a 1 becomes a 0, and vice versa. No flags are affected.
Example: CMA
This complements the value in register A. Suppose register A contained 10101011. Then the result in register A would be 01010100.

CMC (Complement Carry)

Format: CMC
This instruction complements (i.e., inverts, NOTs, toggles) the contents of the Carry flag. What was a 1 becomes a 0, and vice versa. No other flags are affected.

Example: CMC
This complements the value in the Carry flag. Suppose the Carry flag contained 1. Then the result in the Carry flag would be 0.

This instruction might be used before another instruction whose results depend on the value of the Carry flag, such as a conditional jump or a rotate through carry (RAL or RAR).

STC (Set Carry)

Format: STC This instruction sets the Carry flag to 1. No other flags are affected.

Example: STC
This sets the Carry flag to 1.

This instruction might be used before another instruction whose results depend on the value of the Carry flag, such as a conditional jump or a rotate through carry (RAL or RAR).

CONCLUSION

The next two articles will cover the Branch Group and the Stack, I/O, and Machine Control Group of instructions.