Reprinted from MACHINE LANGUAGE PROGRAMMING FOR THE "8008" (and similar microcomputers).

Author: Nat Wadsworth

MACHINE LANGUAGE

Chapter I

THE '8008' CPU INSTRUCTION SET

The '8008' microprocessor has quite a comprehensive instruction set that consists of 48 basic instructions, which, when the possible permutations are considered, result in a total set of about 170 instructions.

The instruction set allows the user to direct the computer to perform operations with memory, with the seven basic registers in the CPU, and with INPUT and OUTPUT ports.

It should be pointed out that the seven basic registers in the CPU consist of one "accumulator," a register that can perform mathematical and logic operations, plus an additional six registers, which, while not having the full capability of the accumulator, can perform various useful operations. These operations include the ability to hold data, serve as an "operator" with the accumulator, and increment or decrement their contents. Two of these six registers have special significance because they may be used to serve as a "pointer" to locations in memory.

The seven CPU registers have arbitrarily been given symbols so that we may refer to them in an abbreviated language. The first register is designated by the symbol 'A' in the following discussion and will be referred to as the "accumulator" register. The next four registers will be referred to as the 'B,' 'C,' 'D' and 'E' registers. The remaining two special memory pointing registers shall be designated the 'H' (for the HIGH portion of a memory address) and the 'L' (for the LOW portion of a memory address) registers.

The CPU also has several "flip-flops" which shall be referred to as "FLAGS." The flip-flops are set as the result of certain operations and are important because they can be "tested" by many of the instructions with the instruction's meaning changing as a consequence of the particular status of a FLAG at the time the instruction is executed. There are four basic flags which will be referred to in this manual. They are defined as follows:

The 'C' flag refers to the carry bit status. The carry bit is a one unit register which changes state when the accumulator overflows or underflows. This bit can also be set to a known condition by certain types of instructions. This is important to remember when developing a program because quite often a program will have a long string of instructions which do not utilize the carry bit or care about its status, but which will be causing the carry bit to change its state from time-to-time. Thus, when one prepares to do a series of operations that will rely on the carry bit, one often desires to set the carry bit to a known state.

The 'Z' for zero flag refers to a one unit register that when desired will indicate whether the value of the accumulator is exactly equal to zero. In addition, immediately after an increment of decrement of the B, C, D, E, H or L registers, this flag will also indicate whether the increment or decrement caused that particular register to go to zero.

The 'S' for sign flag refers to a one unit register that indicates whether the value in the accumulator is a positive or negative value (based on two's complement nomenclature). Essentially, this flag monitors the most significant bit in the accumulator and is "set" when it is a one.

The 'P' flag refers to the last flag in the group which is for indicating when the accumulator contains a value which has even parity. Parity is useful for a number of reasons and is usually used in conjunction with testing for error conditions on words of data especially when transferring data to and from external devices. Even parity occurs when the number of bits that are a logic one in the accumulator is an even value. Zero is considered an even value for this purpose. Since there are eight bits in the accumulator, even parity will occur when zero, two, four or six bits are in the logic one condition regardless of what order they may appear in within the register.

It is important to note that the Z, S, and P flags (as well as the previously mentioned C flag) can all be set to known states by certain instructions. It is also important to note that some instructions do not result in the flags being set so that if the programmer desires to have the program make decisions based on the status of flags, the programmer should ensure that the proper instruction, or sequence of instructions is utilized. It is particularly important to note that load register instructions do not by themselves set the flags. Since it is often desirable to obtain a data word (that is, load it into the accumulator) and test its status for such parameters as whether or not the value is zero, or a negative number, and so forth, the programmer must remember to follow a load instruction by a logical instruction (such as the NDA - "and the accumulator") in order to set the flags before using an instruction that is conditional in regards to a flag's status.

The description of the various types of instructions available using an '8008' CPU which follows will provide both the machine language code for the instruction given as three octal digits, and also a mnemonic name suitable for writing programs in "symbolic" type language which is usually easier than trying to remember octal codes! It may be noted that the symbolic language used is the same as that originally suggested by Intel Corporation which developed the '8008' CPU-on-a-chip. Hence users who may already be familiar with the suggested mnemonics will not have any relearning problems and those learning the mnemonics for the first time will have plenty of good company. If the programmer is not already aware of it, the use of mnemonics facilitates working with an "assembler" program when it is desired to develop relatively large and complex programs. Thus the programmer is urged to concentrate on learning the mnemonics for the instructions and not waste time memorizing the octal codes. After a program has been written using the mnemonic codes, the programmer can always use a lookup table to convert to the machine code if an assembler program is not available. It's a lot easier technique (and less subject to error) than trying to memorize

PROGRAMMING FOR THE "8008" and similar microcomputers

the 170 or so three digit combinations which make up the machine instruction code set!

The programmer must also be aware, that in this machine, some instructions require more than one word in memory. "Immediate" type commands require two consecutive words. JUMP and CALL commands require three consecutive words. The remaining types only require one word.

The first group of instructions to be presented are those that are used to load data from one CPU register to another, or from a CPU register to a word in memory, or vice-versa. This group of instructions requires just one word of memory. It is important to note that none of the instructions in this group affect the flags.

LOAD DATA FROM ONE CPU REGISTER TO ANOTHER CPU REGISTER

MNEMONIC	MACHINE CODE
LAA	300
LBA	310
.	.
.	.
LAB	301

The load register group of instructions allows the programmer to move the contents of one CPU register into another CPU register. The contents of the originating (from) register is not changed. The contents of the destination (to) register becomes the same as the originating register. Any CPU register can be loaded into any CPU register. Note that loading register A into register A is essentially a NOP (no operation) command. When using mnemonics the load symbol is the letter L followed by the "to" register and then the "from" register. The mnemonic LBA means that the contents of register A (the accumulator) is to be loaded into register B. The mnemonic LAB states that register B is to have its contents loaded into register A. It may be observed that this basic instruction has many variations. The machine language coding for this instruction is in the same format as the mnemonic code except that the letters used to represent the registers are replaced by numbers that the computer can use. Using octal code, the seven CPU registers are coded as follows:

Register A = 0
Register B = 1
Register C = 2
Register D = 3
Register E = 4
Register H = 5
Register L = 6

Also, since the machine can only utilize numbers, the octal number '3' in the most significant location of a word signifies that the computer is to perform a "load" operation. Thus, in machine coding, the instruction for loading register B with the contents of register A becomes '310' (in octal form). Or, if one wanted to get very detailed, the actual binary coding for the eight bits of information in the instruction word would be '11 001 000.' It is important to note that the load instructions do not affect any of the flags.

LOAD DATA FROM ANY CPU REGISTER TO A LOCATION IN MEMORY

LMA	370
LMB	371
LMC	372
LMD	373
LME	374
LMH	375
LML	376

This instruction is very similar to the previous group of instructions except that now the contents of a CPU register will be loaded into a specified memory location. The memory location that will receive the contents of the particular CPU register is that whose address is specified by the contents of the CPU H and L registers at the time the instruction is executed. The H CPU register specifies the HIGH portion of the address desired, and the L CPU register specifies the LOW portion of the address into which data from the selected CPU register is to be loaded. Note that there are seven different instructions in this group. Any CPU register can have its contents loaded into any location in memory. This group of instructions does not affect any of the flags.

LOAD DATA FROM A MEMORY LOCATION TO ANY CPU REGISTER

LAM	307
LBM	317
LCM	327
LDM	337
LEM	347
LHM	357
LLM	367

This group of instructions can be considered the opposite of the previous group. Now, the contents of the word in memory whose address is specified by the H (for HIGH portion of the address) and L (LOW portion of the address) registers will be loaded into the CPU register specified by the instruction. Once again, this group of instructions has no affect on the status of the flags.

LOAD IMMEDIATE DATA INTO A CPU REGISTER

LAI	006
LBI	016
LCI	026
LDI	036
LEI	046
LHI	056
LLI	066

An IMMEDIATE type of instruction requires two words in order to be completely specified. The first word is the instruction itself. The second word, or "immediately following" word, must contain the data upon which "immediate" action is taken. Thus, a load IMMEDIATE instruction in this group means that the contents of the word immediately following the instruction word is to be loaded into the specified register. For example, a typical load immediate instruction would be LAI 001. This would result in the value 001 (octal) being placed in the A register when the instruction was executed. It is important to remember that all IMMEDIATE type in-

structions MUST be followed by a data word. An instruction such as LDI by itself would result in improper operation because the computer would assume the next word contained data. If the programmer had mistakenly left out the data word, and in its place had another instruction, the computer would not realize the operator's mistake. Hence the program would be fouled-up! Note too, that the load immediate group of instructions does not affect the flags.

LOAD IMMEDIATE DATA INTO A MEMORY LOCATION

LMI 076

This instruction is essentially the same as the load immediate into the CPU register group except that now, using the contents of the H and L registers as "pointers" to the desired address in memory, the contents of the "immediately following word" will be placed in the memory location specified. This instruction does not affect the status of the flags.

The above rather large group of LOAD instructions permits the programmer to direct the computer to move data about. They are used to bring in data from memory where it can be operated on by the CPU. Or, to temporarily store intermediate results in the CPU registers during complicated and extended calculations, and of course allow data, such as results, to be placed back into memory for long term storage. Since none of them will alter the contents of the four CPU flags, these instructions can be called upon to set up data before instructions that may affect or utilize the flag's status are executed. The programmer will use instructions from this set frequently. The mnemonic names for the instructions are easy to remember as they are well ordered. The most important item to remember about the mnemonics is that the TO register is always indicated first in the mnemonic, and then the FROM register. Thus LBA equals "load TO register B FROM register A.

INCREMENT THE VALUE OF A CPU REGISTER BY ONE

INB	010
INC	020
IND	030
INE	040
INH	050
INL	060

This group of instructions allows the programmer to add one to the present value of any of the CPU registers except the accumulator. (Note carefully that the accumulator can NOT be incremented by this type of instruction. In order to add one to the accumulator a mathematical addition instruction, described later, must be used.) This instruction for incrementing the defined CPU registers is very valuable in a number of applications. For one thing, it is an easy way to have the L register successively "point" to a string of locations in memory. A feature that makes this type of instruction even more powerful is that the result of the incremented register will affect the Z, S, and P flags. (It will not change the C or "carry"flag.) Thus, after a CPU register has been incremented by this instruction, one can utilize a flag test instruction (such as the conditional JUMP and CALL instructions to be described later) to determine whether that particular register has a value of zero (Z flag), or if it is a negative number (S flag), or even parity (P flag). It is important to note that this group of instructions, and the decrement group (described in the next paragraph) are the only instructions which allow the flags to be manipulated by operations that are not concerned with the accumulator (A) register.

DECREMENT THE VALUE OF A CPU REGISTER BY ONE

DCB	011
DCC	021
DCD	031
DCE	041
DCH	051
DCL	061

The DECREMENT group of instructions is similar to the INCREMENT group except that now the value one will be subtracted from the specified CPU register. This instruction will not affect the C flag. But, it does affect the Z, S, and P flags. It should also be noted that this group, as with the increment group, does not include the accumulator register. A separate mathematical instruction must be used to subtract one from the accumulator.

ARITHMETIC INSTRUCTIONS USING THE ACCUMULATOR

The following group of instructions allow the programmer to direct the computer to perform arithmetic operations between other CPU registers and the accumulator, or between the contents of words in memory and the accumulator. All of the operations for the described addition, subtraction, and compare instructions affect the status of the flags.

ADD THE CONTENTS OF A CPU REGISTER TO THE ACCUMULATOR

ADA	200
ADB	201
ADC	202
ADD	203
ADE	204
ADH	205
ADL	206

This group of instructions will simply ADD the present contents of the accumulator register to the present value of the specified CPU register and leave the result in the accumulator. The value of the specified register is unchanged except in the case of the ADA instruction. Note that the ADA instruction essentially allows the programmer to double the value of the accumulator (which is the A register)! If the addition causes an overflow or underflow then the carry (C flag) will be affected.

ADD THE CONTENTS OF A CPU REGISTER PLUS THE VALUE OF THE CARRY FLAG TO THE ACCUMULATOR

ACA	210
ACB	211
ACC	212
ACD	213
ACE	214
ACH	215
ACL	216

This group is identical to the previous group except that the content of the carry flag is considered as an additional bit (MSB) in the specified CPU register. The combined value of the carry bit plus the contents of the specified CPU register are added to the value in the accumulator. The results are left in the accumulator. Again, with the exception of the ACA instruction, the contents of the specified CPU register are left unchanged. Again too, the carry bit (C flag) will be affected by the results of the operation.

SUBTRACT THE CONTENTS OF A CPU REGISTER FROM THE ACCUMULATOR

SUA	220
SUB	221
SUC	222
SUD	223
SUE	224
SUH	225
SUL	226

This group of instructions will cause the present value of the specified CPU register to be subtracted from the value in the accumulator. The value of the specified register is not changed except in the case of the SUA instruction. (Note that the SUA instruction is a convenient instruction with which to "clear" the accumulator.) The carry flag will be affected by the results of a SUBTRACT instruction.

SUBTRACT THE CONTENTS OF A CPU REGISTER AND THE VALUE OF THE CARRY FLAG FROM THE ACCUMULATOR

SBA	230
SBB	231
SBC	232
SBD	233
SBE	234
SBH	235
SBL	236

This group is identical to the previous group except that the content of the carry flag is considered as an additional bit (MSB) in the specified CPU register. The combined value of the carry bit plus the contents of the specified CPU register are SUBTRACTED from the value in the accumulator. The results are left in the accumulator. The carry

bit (C flag) is affected by the result of the operation. With the exception of the SBA instruction the content of the specified CPU register is left unchanged.

COMPARE THE VALUE IN THE ACCUMULATOR AGAINST THE CONTENTS OF A CPU REGISTER

CPA	270
CPB	271
CPC	272
CPD	273
CPE	274
CPH	275
CPL	276

The COMPARE group of instructions are a very powerful and somewhat unique set of instructions. They direct the computer to compare the contents of the accumulator against another register and to set the flags as a result of the comparing operation. It is essentially a subtraction operation with the value of the specified register being subtracted from the value of the accumulator except that the value of the accumulator is not actually altered by the operation. However, the flags are set in the same manner as though an actual subtraction operation had occured. Thus, by subsequently testing the status of the various flags after a COMPARE instruction has been executed, the program can determine whether the compare operation resulted in a match or non-match. In the case of a non-match, one may determine if the compared register contained a value greater or less than that in the accumulator. This would be accomplished by testing the Z flag and C flag respectively utilizing a conditional JUMP or CALL instruction (which will be described later).

ADDITION, SUBTRACTION, AND COMPARE INSTRUCTIONS THAT USE WORDS IN MEMORY AS OPERANDS

The five types of mathematical operations: ADD, ADD with CARRY, SUBTRACT, SUBTRACT with CARRY, and COMPARE, which have just been presented for the cases where they operate with the contents of CPU registers, can all be performed with words that are in memory. As with the LOAD instructions that operate with memory, the H and L registers must contain the address of the word in memory that it is desired to ADD, SUBTRACT, or COMPARE to the accumulator. The same conditions for the operations as was detailed when using the CPU registers apply. Thus, for mathematical operations with a word in memory, the following instructions are used.

ADD THE CONTENTS OF A MEMORY WORD TO THE ACCUMULATOR

ADM 207

ADD THE CONTENTS OF A MEMORY WORD PLUS THE VALUE OF THE CARRY FLAG TO THE ACCUMULATOR

ACM 217

SUBTRACT THE CONTENTS OF A MEMORY WORD FROM THE ACCUMULATOR

SUM 227

SUBTRACT THE CONTENTS OF A MEMORY WORD AND THE VALUE OF THE CARRY FLAG FROM THE ACCUMULATOR

SBM 237

COMPARE THE VALUE IN THE ACCUMULATOR AGAINST THE CONTENTS OF A MEMORY WORD

CPM 277

IMMEDIATE TYPE ADDITIONS, SUBTRACTIONS, AND COMPARE INSTRUCTIONS

The five types of mathematical operations discussed above can also be performed with the operand being the word of data immediately after the instruction. This group of instructions is similar in format to the previously described LOAD IMMEDIATE instructions. The same conditions for the mathematical operations as discussed for the operations with the CPU registers apply.

ADD IMMEDIATE

ADI 004

ADD WITH CARRY IMMEDIATE

ACI 014

SUBTRACT IMMEDIATE

SUI 024

SUBTRACT WITH CARRY IMMEDIATE

SBI 034

COMPARE IMMEDIATE

CPI 074

LOGICAL INSTRUCTIONS WITH THE ACCUMULATOR

There are several groups of instructions which allow BOOLEAN LOGIC operations to be performed between the contents of the CPU registers and the A (accumulator) register. In addition there are logic IMMEDIATE type instructions. The boolean logic operations are valuable in a number of programming applications. The instruction set allows three basic boolean operations to be performed. These are: the LOGICAL AND, the LOGICAL OR, and the EXCLUSIVE OR operations. Each type of logic operation is performed on a bit-by-bit basis between the accumulator and the CPU register or memory location specified by the instruction. A detailed explanation of each type of logic operation, and the appropriate instructions for each type is presented below. The logic instruction set is also valuable because all of them will cause the C (carry) flag to be placed in the zero condition. This is important if one is going to perform a sequence of instructions that will eventually use the status of the C flag to arrive at a decision as it allows the programmer to set the C flag to a known state at the start of the sequence. All other flags are set in accordance with the result of the logic operation. Hence, the group often has value when the programmer desires to determine the contents of a register that has just been loaded into a register. (Since the load instructions do not alter the flags.)

THE BOOLEAN 'AND' OPERATION INSTRUCTION SET

When the boolean AND instruction is executed, each bit of the accumulator will be compared with the corresponding bit in he register or memory location specified by the instruction. As each bit is compared a logic result will be placed in the accumulator for each bit comparison. The logic result is determined as follows. If both the bit in the accumulator and the bit in the register with which the operation is being performed are a logic one, then the accumulator bit will be left in the logic one condition. For all other possible combinations (A bit equals one, X bit equals zero; A bit equals zero, X bit equals one; or A bit equals zero, X bit equals zero), then the accumulator bit will be cleared to the zero state. An example will illustrate the logical AND operation.

INITIAL STATE OF THE ACCUMULATOR

1 0 1 0 1 0 1 0

CONTENTS OF OPERAND REGISTER

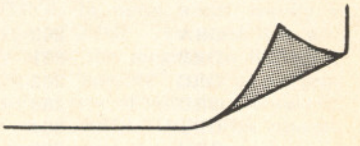
1 1 0 0 1 1 0 0

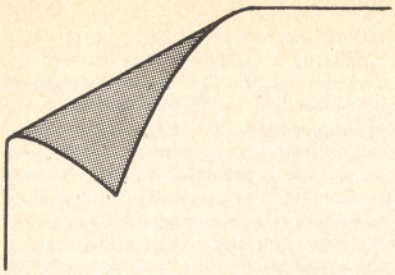
FINAL STATE OF THE ACCUMULATOR

1 0 0 0 1 0 0 0

There are seven logical AND instructions that allow any CPU register to be used as the AND operand. They are as follows.

NDA	240
NDB	241
NDC	242
NDD	243
NDE	244
NDH	245
NDL	246





The contents of the operand register is not altered by an AND logical instruction.

There is also a logical AND instruction that allows a word in memory to be used as an operand. The address of the word in memory that will be used is pointed to by the contents of the H and L CPU registers.

NDM 247

And finally there is also a logical AND IMMEDIATE type of instruction that will use the contents of the word immediately following the instruction as the operand.

NDI 044

The next group of boolean logic instructions direct the computer to perform the logical OR operation on a bit-by-bit basis with the accumulator and the contents of a CPU register or a word in memory. The logical OR operation will result in the accumulator having a bit set to a logic one if either that bit in the accumulator, or the corresponding bit in the operand register is a logic one. Since the case where both the accumulator bit and operand bit are a one also satisfies the criteria, that condition will also result in the accumulator bit being left in the one state. If neither register has a logic one in the bit position, then the accumulator bit for that position remains in the zero state. An example illustrates the results of a logical OR operation.

INITIAL STATE OF THE ACCUMULATOR

1 0 1 0 1 0 1 0

CONTENT OF THE OPERAND REGISTER

1 1 0 0 1 1 0 0

FINAL STATE OF THE ACCUMULATOR

1 1 1 0 1 1 1 0

There are seven logical OR instructions that allow any CPU register to be used as the OR operand.

ORA	260
ORB	261
ORC	262
ORD	263
ORE	264
ORH	265
ORL	266

By using the H and L registers as pointers one can also use a word in memory as an OR operand.

ORM 267

There is also the logical OR IMMEDIATE instruction.

ORI 064

As with the logical AND group of instructions, the logical OR instruction does not alter the contents of the operand register.

The last group of boolean logic instructions is a variation of the logic OR. The variation is termed the EXCLUSIVE OR logical operation. The EXCLUSIVE OR operation is similar to the OR except that when the corresponding bits in both the accumulator and the operand register are a one then the accumulator bit will be cleared to zero. Thus, the accumulator bit will be a one after the operation only if just one of the registers (accumulator register or operand register) has a one in the bit position. (Again, the operation is performed on a bit-by-bit basis.) An example provides clarification.

INITIAL STATE OF THE ACCUMULATOR

1 0 1 0 1 0 1 0

CONTENTS OF THE OPERAND REGISTER

1 1 0 0 1 1 0 0

FINAL STATE OF THE ACCUMULATOR

0 1 1 0 0 1 1 0

The seven instructions that allow the CPU registers to be used as operands are shown next.

XRA	250
XRB	251
XRC	252
XRD	253
XRE	254
XRH	255
XRL	256

The instruction that uses registers H and L as pointers to a memory location is:

XRM 257

And the EXCLUSIVE OR IMMEDIATE type instruction is:

XRI 054

As in the case of the logical OR operation, the operand register is not altered except for the special case when the XRA instruction is used. This instruction, which directs the computer to EXCLUSIVE OR the accumulator with itself, will cause the operand register, since it is the accumulator, to have its contents altered (unless it should happen to be zero at the time the instruction is executed). This is because, regardless of what value is in the accumulator, if it is EXCLUSIVE OR'ed with itself, the result will be zero! The example below illustrates the specific operation.

ORIGINAL VALUE OF ACCUMULATOR

1 0 1 0 1 0 1 0

EXCLUSIVE OR'ed WITH ITSELF

1 0 1 0 1 0 1 0

FINAL VALUE OF ACCUMULATOR

0 0 0 0 0 0 0 0

This only occurs when the logical EXCLUSIVE OR is performed on the accumulator itself. It can be shown that the results of performing the logical OR or logical AND between the accumulator and itself will result in the original accumulator value being retained.

INSTRUCTIONS FOR ROTATING THE CONTENTS OF THE ACCUMULATOR

It is often desirable to be able to shift the contents of the accumulator either right or left. In a fixed length register, a simple shift operation would result in some information being lost because what was in the MSB or LSB (depending on in which direction the shift occured) would be shifted right out of the register! Therefore, instead of just shifting the contents of a register, an operation termed ROTATING is utilized. Now, instead of just shifting a bit off the end of the register, the bit is brought around to the other end of the register. For instance, if the register is rotated to the right, the LSB (least significant bit) would be brought around to the position of the MSB (most significant bit) which would have been vacated by the shifting of its original contents to the right. Or, in the case of a shift to the left, the MSB would be brought around to the position of the LSB.

The carry bit (C flag) can be considered as an extension of the accumulator register. The instruction set for this machine allows two types of ROTATE instructions. One considers the carry bit to be part of the accumulator register for the rotate operation. The other type does not. In addition, each type of rotate can be done either to the right or to the left.

It should be noted that the rotate operations are particularly valuable when it is desired to multiply a number or divide a number. This is because shifting the contents of a register to the left effectively multiplies a binary number by a power of two. Shifting a binary number to the right provides the inverse operation.

ROTATING THE ACCUMULATOR LEFT

RLC 002

Rotating the accumulator left with the RLC instruction means the MSB of the accumulator will be brought around to the LSB position and all other bits will be shifted one position to the left. While this instruction does not shift through the carry bit, the carry bit will be set by the status of the MSB of the accumulator at the start of the ROTATE LEFT operation. (This feature allows the programmer to determine what the MSB was prior to the shifting operation by testing the C flag after the rotate instruction has been executed.

ROTATING THE ACCUMULATOR LEFT THROUGH THE CARRY BIT

RAL 022

The RAL instruction will cause the MSB of the accumulator to go into the carry bit. The initial value of the carry bit will be shifted around to the LSB of the accumulator. All other bits are shifted one position to the left.

ROTATING THE ACCUMULATOR RIGHT

RRC 012

The RRC instruction is similar to the RLC instruction except that now the LSB of the accumulator is placed in the MSB of the accumulator. All other bits are shifted one position to the right. Also, the carry bit will be set to the initial value of the LSB of the accumulator at the start of the operation.

ROTATING THE ACCUMULATOR RIGHT THROUGH THE CARRY BIT

RAR 032

Here, the LSB of the accumulator is brought around to the carry bit. The initial value of the carry bit is shifted to the MSB of the accumulator. All other bits are shifted a position to the right.

It should be noted that the C flag is the only flag that is altered by a rotate instruction. All other flags remain unchanged.

JUMP INSTRUCTIONS

The instructions discussed so far have all been "direct action" instructions. The programmer arranges a sequence of these types of instructions in memory. When the program is started the computer proceeds to execute the instructions in the order in which they are encountered. The computer automatically reads the contents of a memory location, executes the instruction it finds there, and then automatically increments a special address register called a PROGRAM COUNTER that will result in the machine reading the information contained in the next sequential memory location. However, it is often desirable to perform a series of instructions located in one section of memory, and then skip over a group of memory locations and start executing instructions in another section of memory. This action can be accomplished by a group of instructions that will cause a new address value to be placed in the PROGRAM COUNTER. This will cause the computer to go to a new section of memory and then execute instructions sequentially from the new memory location.

The JUMP instructions in this computer add considerable power to the machine's capabilities because there are a series of "conditional" JUMP instructions available. That is, the computer can be directed to test the status of a particular FLAG (C, Z, S or P). If the status of the flag is the desired one, then a JUMP will be performed. If it is not, the machine will continue to execute the next instruction in the current sequence. This capability provides a means for the computer to make "decisions" and to modify its operation as a function of the status of the various flags at the time that a program is being executed.

In a manner similar to IMMEDIATE types of instructions, the JUMP instructions require more than one word of memory. A JUMP instruction requires three words to be properly defined. (Remember that IMMEDIATE type instructions required two words.) The JUMP instruction itself is the first word. The second word must contain the LOW ADDRESS portion of the address of the word in memory that the PROGRAM COUNTER is to be set to point to, which is the new location from which the next instruction is to be fetched. The third word must contain the HIGH ADDRESS (sometimes referred to as the PAGE) of the memory address that the program counter will be set to. That is, the high order portion of the address in memory that the computer will JUMP to in order to obtain its next instruction.

THE UNCONDITIONAL JUMP INSTRUCTION

JMP 1X4

Note: The machine code 1X4 indicates that any code for the second octal digit of the machine code is valid. It is recommended as a standard practice that the code '0' be used. Thus, the typical machine code would be 104.

Remember, the JUMP instruction must be followed by two more words which contain the LOW, and then the HIGH (PAGE) portion of the address that the program is to JUMP to!

JUMP IF THE DESIGNATED FLAG IS TRUE (CONDITIONAL JUMP)

JTC	140
JTZ	150
JTS	160
JTP	170

As with the UNCONDITIONAL JUMP instruction, the CONDITIONAL JUMP instructions must be followed by two words of information. The LOW portion, then the HIGH portion, of the address that program execution is to continue from if the jump is executed. The JUMP IF TRUE group of instructions will only jump to the designated address if the condition of the appropriate flag is TRUE (logical one). Thus, the JTC instruction states that if the carry flag (C) is a logical one (TRUE) then the jump is to be executed. If it is a logical zero (FALSE) then program execution is to continue with the next instruction in the current sequence of instructions. In a similar manner the JTZ instruction states that if the ZERO FLAG is TRUE then the jump is to be performed. Otherwise the next instruction in the present sequence is executed. Likewise for the JTS and JTP instructions.

JUMP IF THE DESIGNATED FLAG IS FALSE (CONDITIONAL JUMP)

JFC	100
JFZ	110
JFS	120
JFP	130

As with all JUMP instructions these instructions must be followed by the LOW address then the HIGH address of the memory location that program execution is to continue from if the jump is executed. This group of instructions is the opposite of the jump if the flag is true group. For instance, the JFC instruction commands the computer to test the status of the carry (C) flag. If the flag is FALSE (a logic zero), then the jump is to be performed. If it is TRUE, then program execution is to continue with the next instruction in the current sequence of instructions. The same procedure holds for the JFZ, JFS and JFP instructions.

SUBROUTINE CALLING INSTRUCTIONS

Quite often when a programmer is developing computer programs the programmer will find that a particular algorithm (sequence of instructions for performing a function) can be used many times in different parts of the program. Rather than having to keep entering the same sequence of instructions at different locations in memory, which would not only consume the time of the programmer, but would also result in a lot of memory being used to perform one particular function, it is desirable to be able to be able to put an often used sequence of commands in just one location in memory. Then, whenever the particular algorithm is required by another part of the program, it would be convenient to jump to the section that contained the often used algorithm, perform the sequence of instructions, and then return back to the main part of the program. This is a standard practice in computer operations. A frequently used algorithm can be designated a SUBROUTINE. A special set of instructions allows the programmer to CALL a SUBROUTINE. In other words, specify a special type of JUMP command that will eventually allow the program to RETURN to the original "jumping" point in the program. A second type of instruction is used to terminate a SUBROUTINE. This special terminator will cause the program to revert back and pick up the next sequential in-

struction in memory that immediately follows the original CALLING instruction. A great deal of computer power is provided by the instruction set in this machine that allows one to CALL and RETURN from SUBROUTINES. This is because, in a manner similar to that provided for the CONDITIONAL JUMP instructions, there are a number of CONDITIONAL CALL and CONDITIONAL RETURN commands in the instruction set.

Like the JUMP instructions, the CALL instructions all require three words in order to be fully specified. The first word is the CALL instruction itself. The next two words must contain the LOW and HIGH portions of the starting address of the subroutine that is being "called."

When a CALL instruction is encountered by the computer, the CPU will actually save the current value of the PROGRAM COUNTER by storing it in a special PROGRAM COUNTER PUSH-DOWN STACK. This stack is capable of holding six addresses plus the current operating address. What this means is that the machine is capable of "nesting" up to seven subroutines at one time. Thus, one can have a subroutine, that in turn calls another subroutine, that in turn calls another one, up to seven levels, and the machine will still be able to return to the initial calling location. The programmer must ensure that subroutines are not nested more than seven levels otherwise the PROGRAM COUNTER PUSH-DOWN STACK will push the original calling address(es) completely out of the push-down stack. The program could then no longer automatically return to the initial calling location.

The RETURN instruction which terminates a SUBROUTINE only requires one word. When the CPU encounters a RETURN instruction it causes the PROGRAM COUNTER PUSH-DOWN STACK to "pop" up one level. This effectively causes the address saved in the stack by the calling routine to be taken as the new program counter. Hence, program execution returns to the calling location.

THE UNCONDITIONAL CALL INSTRUCTION

CAL 1X6

This instruction followed by two words containing the LOW and then the HIGH order of the starting address of the SUBROUTINE that is to be executed is an UNCONDITIONAL CALL. The subroutine will be executed regardless of the status of the FLAGS. The next sequential address after the CAL instruction is saved in the PROGRAM COUNTER PUSH-DOWN STACK.

THE UNCONDITIONAL RETURN INSTRUCTION

RET 0X7

This instruction directs the CPU to unconditionally "pop" the program counter push-down stack UP one level. Program execution will continue from the address saved by the subroutine calling instruction.

CALL A SUBROUTINE IF THE DESIGNATED FLAG IS TRUE

CTC	142
CTZ	152
CTS	162
CTP	172

In a manner similar to the conditional JUMP IF TRUE instructions, these instructions (which must all be followed by the LOW and HIGH portions of the called subroutine's starting address) will only perform the "call" if the designated flag is in the TRUE (logical one) state. If the designated flag is FALSE then the CALL instruction is ignored. Program execution then continues with the next sequential instruction.

RETURN FROM A SUBROUTINE IF THE DESIGNATED FLAG IS TRUE

RTC	043
RTZ	053
RTS	063
RTP	073

These one word instructions will cause a SUBROUTINE to be TERMINATED only if the designated flag is in the logical one (TRUE) state.

CALL A SUBROUTINE IF THE DESIGNATED FLAG IS FALSE

CFC	102
CFZ	112
CFS	122
CFP	132

These instructions are the opposit of the previous group of calling commands. The subroutine is called only if the designated flag is in the FALSE (logical zero) condition. Remember, these instructions must be followed by two words which contain the LOW and HIGH part of the starting address of the SUBROUTINE that is to be executed if the designated flag is FALSE. If the flag is TRUE, the subroutine will not be called and program operation will continue with the next instruction in the current sequence.

RETURN FROM A SUBROUTINE IF THE DESIGNATED FLAG IS FALSE

RFC	003
RFZ	013
RFS	023
RFP	033

These one word instructions will terminate a subroutine ("pop" the program counter stack UP one level) if the designated flag is FALSE. Otherwise, the instruction is ignored and program operation is continued with the next instruction in the subroutine.

THE SPECIAL RESTART SUBROUTINE CALL INSTRUCTIONS

There is a special purpose instruction available that effectively serves as a one word SUBROUTINE CALL. (Remember that it normally requires three words to specify a subroutine call.) This special instruction allows the programmer to call a subroutine that starts at any one of eight specially designated memory locations. The eight special memory locations are at locations: 000, 010, 020, 030, 040, 050, 060 and 070 on page zero. There are eight variations of the machine code for the RESTART instruction. One for each of the above addresses. Thus, the one word instruction can serve to CALL a SUBROUTINE at the specified starting location (instead of having two additional words to specify the starting address of a subroutine). It is often convenient to utilize a RESTART command as a quick CALL to an often used subroutine. Or, as an easy way to call short "starting" subroutines for large programs. Hence, the name for the type of instruction. The eight RESTART instructions, in their mnemonic and machine code forms, along with the starting address associated with each one is listed below.

RST 0	005	00 000
RST 1	015	00 010
RST 2	025	00 020
RST 3	035	00 030
RST 4	045	00 040
RST 5	055	00 050
RST 6	065	00 060
RST 7	075	00 070

INPUT INSTRUCTIONS

In order to receive information from an external device the computer must utilize a group of special signal lines. The typical '8008' computer is designed to handle up to eight groups (each group having eight signal lines) of INPUT signals. A group of signals is accepted at the computer by what is referred to as an INPUT PORT. The computer controls the operation of the INPUT PORTS. Under program control, the computer can be directed to obtain the information that is on a group of lines coming in to any INPUT PORT. When this is done the information will be transferred to the accumulator. Various types of external equipment, such as an electronic keyboard or measuring instruments, can be connected to the INPUT PORTS. The INPUT PORTS are typically referred to as having numbers from '0' to '7.' The typical mnemonics and machine codes for INPUT instructions are shown next.

INP 0	101
INP 1	103
.	.
INP 6	115
INP 7	117

It may be interesting to note that the machine codes for input ports increase by a factor of two for each port. Note too, that while the mnemonic for an input instruction has two parts, the machine code only requires one word in memory. It is also important to realize that while an input instruction brings data into the accumulator it does not affect the status of any of the CPU flags!

OUTPUT INSTRUCTIONS

In order to output information to an external device the computer utilizes another group of signal lines which are referred to as OUTPUT PORTS. A Typical '8008' system may be equipped to service up to twenty-four OUTPUT PORTS. (Each OUTPUT PORT actually consists of eight signal lines.) An OUTPUT instruction causes the contents of the accumulator to be transferred to the signal lines of the designated OUTPUT PORT. The output ports are normally designated by octal numbers in the range 10 to 37. The list below shows the typical mnemonics used to specify an OUTPUT PORT along with the associated machine code. (It may be interesting to note again that the machine code increases by a factor of two for each port.)

OUT 10	121
OUT 11	123
.	.
OUT 21	141
.	.
OUT 36	175
OUT 37	177

An OUTPUT instruction only requires one machine code word (even though the mnemonic is typically specified in two parts). OUTPUT PORTS are connected to external devices that one desires to have the computer transmit information to, such as a CRT display, or machinery that is to be placed under computer control.

THE HALT INSTRUCTION

There is one more instruction in the '8008' instruction set. This instruction directs the CPU to stop all operations and to remain in that state until an INTERRUPT signal is received. In a typical '8008' system an INTERRUPT signal may be generated by an operator pressing a switch or by an external piece of equipment sending an electronic signal to the CPU. This instruction is normally used when the programmer desires to terminate a program or when it is desired to have the computer wait for an operator or external device to perform some action. There are three machine codes that may be used for the HALT command.

HLT	000
HLT	001
HLT	377

The HALT instruction does not affect the status of the CPU flags.

INFORMATION ON INSTRUCTION EXECUTION TIMES

When programming for "real-time" applications it is important to know how much time each type of instruction requires to be executed. With this information the programmer can develop "timing loops" or determine with substantual accuracy how much time it will take to perform a particular series of instructions. This information is especially valuable when dealing with programs that control the operations of external devices which might require events to occur at specific times.

The following table provides the nominal instruction execution time for each category of instruction used in an '8008' system. The precise time needed for each instruction depends on how close the master clock has been set to a nominal value of 500 kilohertz. The table shows the number of cycle states required by the type of instruction followed by the nominal time required to perform the entire instruction. Since each state executes in four microseconds, the total time required to perform the instruction as shown in the table was obtained by multiplying the number of states by four microseconds. By knowing the number of states required for each instruction the programmer can often rearrange an algorithm or substitute different types of instructions to provide programs that have events occuring at precisely timed intervals.

INSTRUCTION EXECUTION TIME TABLE

LOAD DATA FROM A CPU REGISTER TO ANOTHER CPU REGISTER	5	20 Us
LOAD DATA FROM A CPU REGISTER TO A LOCATION IN MEMORY	7	28
LOAD DATA FROM MEMORY TO A CPU REGISTER	8	32
LOAD IMMEDIATE DATA INTO A CPU REGISTER	8	32
LOAD IMMEDIATE DATA INTO A LOCATION IN MEMORY	9	36
INCREMENT OR DECREMENT A CPU REGISTER	5	20
ARITHMETIC/COMPARE BETWEEN ACCUMULATOR & A CPU REGISTER	5	20
ARITH/COMPARE BETWEEN ACCUMULATOR & A WORD IN MEMORY	8	32
IMMEDIATE.ARITHMETIC AND COMPARE	8	32
BOOLEAN OPS BETWEEN ACCUMULATOR AND CPU REGISTERS	5	20
BOOLEAN OPS WITH ACCUMULATOR & A WORD IN MEMORY	8	32
IMMEDIATE BOOLEAN OPERATIONS	8	20
ROTATE THE ACCUMULATOR	5	20
JUMP AND CALL COMMANDS (UNCONDITIONAL)	11	44
JUMP/CALLS WHEN CONDITION NOT SATISFIED (CONDITIONAL)	9	36
JUMP/CALLS WHEN CONDITION SATISFIED (CONDITIONAL)	11	44
RETURN (UNCONDITIONAL)	5	20
RETURN WHEN CONDITION NOT SATISFIED (CONDITIONAL)	3	12
RETURN WHEN CONDITION SATISFIED (CONDITIONAL)	5	20
RESTART COMMAND	5	20
OUTPUT COMMAND	6	24
INPUT COMMAND	8	32
HALT COMMAND	4	16

Chapters 2 and 3 of **MACHINE LANGUAGE PROGRAMMING FOR THE "8008" (and similar microcomputers)** will appear in BYTE's August and September issues, respectively.■