C. CLOCK

Orderly execution of a program by the CPU is controlled by a 2-MHz crystal controlled clock. Crystal control is used to permit the clock to operate at the maximum permissible CPU speed. A clock without crystal regulation might occassionally speed up beyond the CPU's capability and program execution errors would result.

D. INPUT/OUTPUT

The *ALTAIR 8800* can be interfaced with a great many external devices. Generally, these devices provide input information to the computer and accept output information from the computer. The CPU monitors the status of program execution and Input/Output devices and provides the necessary signals for servicing external devices. The programmer can instruct the CPU to either ignore or respond to interrupt signals provided by an external device. These interrupt signals, when accepted by the CPU, cause the program execution to be temporarily halted while the external device is serviced by the computer. When the external device has been serviced, the program resumes normal execution. The *ALTAIR 8800* will service up to 256 Input and 256 Output devices.

This concludes the description of the organization of the *ALTAIR 8800*. The overall operation of the computer as a powerful and efficient data processing system will become more apparent in Part 3, a discussion of the operation of the *ALTAIR 8800*.

PART 3. OPERATION OF THE *ALTAIR 8800*

Access to the basic *ALTAIR 8800* is achieved via the front panel, and at first glance the array of 25 toggle switches and 36 indicator and status LEDs may appear confusing. Actually, operation of the *ALTAIR 8800* is very straightforward and most users learn to load a program into the machine and run it in less than an hour. If you are a typical user, you will spend far more time developing and writing programs than actually operating the machine.

This part of the *ALTAIR 8800* Operating Manual explains the purpose and application of the front panel switches and indicator and status LEDs. A sample program is then loaded into the machine and run. A detailed discussion of the role and efficient use of memory is included next. Finally, several operating hints which will help you edit and "debug" programs are included.

A. THE FRONT PANEL SWITCHES AND LEDs

Though the front panel contains 25 toggle switches and 36 indicator and status LEDs, most routine operations of the basic *ALTAIR 8800* (256 words of memory) can be performed with only 15 switches and by monitoring only 16 LEDs. The function of all the switches and LEDs is explained below:

ON-OFF Switch--The ON position applies power to the computer. The OFF position cuts off power and also erases the contents of the memory.

STOP-RUN Switch--The STOP position stops program execution. The RUN position implements program execution.

SINGLE STEP Switch--This switch implements a single machine language instruction each time it is actuated. A single machine language instruction may require as many as 5 machine cycles.

EXAMINE-EXAMINE NEXT Switch--The EXAMINE position displays the contents of any specified memory address previously loaded into the DATA/ADDRESS Switches (see below) on the 8 data LEDs. The EXAMINE NEXT position displays the contents of the next sequential memory address. Each time EXAMINE NEXT is actuated, the contents of the next sequential memory address are displayed.

DEPOSIT-DEPOSIT NEXT Switch--The DEPOSIT position causes the data byte loaded into the 8 DATA Switches to be loaded into the memory address which has been previously designated. The DEPOSIT NEXT position loads the data byte loaded into the 8 DATA Switches into the next sequential memory address. Each time DEPOSIT NEXT is actuated, the data byte loaded into the 8 DATA Switches is loaded into the next sequential memory address. The data byte loaded into the 8 DATA Switches can be changed before actuating DEPOSIT or DEPOSIT NEXT.

RESET-CLR Switch--The RESET position sets the Program Counter to the first memory address (0 000 000 000 000 000). RESET provides a rapid and efficient way to get back to the first step of a program which begins at the first memory address. CLR is a CLEAR command for external input/output equipment.

PROTECT-UNPROTECT Switch--The PROTECT position prevents memory contents from being changed. The UNPROTECT position

permits the contents of the memory to be altered.

AUX Switches--The basic *ALTAIR 8800* includes two auxiliary switches which are not yet connected to the computer. These switches will be used in conjunction with peripherals added to the basic machine.

DATA/ADDRESS Switches--The DATA Switches are those designated 7-0. The ADDRESS Switches are those designated 15-0. A switch whose toggle is in the UP position denotes a 1 bit. A switch whose toggle is in the DOWN position denotes a 0 bit. In the basic *ALTAIR 8800* (256 word memory), the ADDRESS Switches designated 8-15 are not used and should be set to 0 when an address is being entered.

2. INDICATOR LEDs

(NOTE: When machine is stopped, a glowing LED denotes a 1 bit or an active status of a specified condition; and a non-glowing LED denotes a 0 bit or inactive status. While running a program, however, LEDs may appear to give erroneous indications.)

ADDRESS--The ADDRESS LEDs are those designated A15-A0. The bit pattern shown on the ADDRESS LEDs denotes the memory address being examined or loaded with data.

DATA--The DATA LEDs are those designated D7-D0. The bit pattern shown on the DATA LEDs denotes the data in the specified memory address.

INTE--An interrupt has been enabled when this LED is glowing.

PROT--The memory is protected when this LED is glowing.

WAIT--The CPU is in a WAIT state when this LED is glowing.

HLDA--A HOLD has been acknowledged when this LED is glowing.

3. STATUS LEDs

(NOTE: A glowing LED denotes an active status for the designated condition.)

LED	DEFINITION
MEMR	The memory bus will be used for memory read data.
INP	The address bus containing the address of an input device. The input data should be placed on the data bus when the data bus is in the input mode.
M1	The CPU is processing the first machine cycle of an instruction.
OUT	The address contains the address of an output device and the data bus will contain the output data when the CPU is ready.
HLTA	A HALT instruction has been executed and acknowledged.
STACK	The address bus holds the Stack Pointer's push-down stack address.
WO	Operation in the current machine cycle will be a WRITE memory or OUTPUT function. Otherwise, a READ memory or INPUT operation will occur.
INT	An interrupt request has been acknowledged.

B. LOADING A SAMPLE PROGRAM

In Section G of Part 1, a simple addition program in machine language mnemonics is presented. The program is designed to retrieve two numbers from memory, add them together, and store the result in memory. The exact program in mnemonic form can be written thusly:

0. LDA

1. MOV (A→B)

2. LDA

3. ADD (A+B)

4. STA

5. JMP

The mnemonics for all 78 of the *ALTAIR 8800* instructions are explained in detail in Part 4 of this manual. For now, the following definitions will suffice:

1. LDA--Load the accumulator with the contents of a specified memory address.

2. MOV (A→B)--Move the contents of the accumulator into register B.

3. ADD (B+A)--Add the contents of register B to the contents of the accumulator and store the result in the accumulator.

4. STA--Store the contents of the accumulator in a specified memory address.

5. JMP--Jump to the first step in the program.*

*Once the computer has executed the program it will search its memory for something else to do. To maintain control of the CPU, we can end our sample program with a JMP instruction (followed by the memory address of the first instruction). The computer will "jump" back to the first instruction in the sample program and execute the program over and over again.

Notice how precise and specific each of these instructions is. The computer is instructed exactly how to solve the problem and where to place the result. Each of these machine language instructions requires a single byte bit pattern to implement the basic instruction. LDA and STA require two additional bytes to provide the necessary memory addresses.

To load this program into the *ALTAIR 8800*, you must first determine the memory addresses for the two numbers to be added, the result, and the program itself. In most cases, it's more convenient to store a new program by beginning at the first memory address (0). Therefore, the memory addresses for the data (the two numbers to be added and the result) should be placed at any arbitrary addresses higher in memory. Since the basic *ALTAIR 8800* has 256 words of memory, let's select a location for data addresses beginning at memory address 128. The first number to be added will be located at memory address 128 (10 000 000), the second at memory address 129 (10 000 001), and the result at memory address 130 (10 000 010). Now that the memory addresses have been specified, the program can be converted into its machine language bit patterns:

MNEMONIC	BIT PATTERN	EXPLANATION
0. LDA	00 111 010	Load Accumulator with contents
	10 000 000	of: Memory address 128 (2 bytes
	00 000 000	required for memory addresses)
1. MOV (A→B)	01 000 111	Move Accumulator to Register B
2. LDA	00 111 010	Load Accumulator with contents
	10 000 001	of: Memory address 129
	00 000 000	
3. ADD (B+A)	10 000 000	Add Register B to Accumulator

MNEMONIC	BIT PATTERN	EXPLANATION
4. STA	00 110 010	Store Accumulator contents
	10 000 010	at: Memory address 130
	00 000 000	
5. JMP	11 000 011	Jump to Memory location 0.
	00 000 000	
	00 000 000	

Usually the individual bit patterns of a machine language program are sequentially numbered to reduce the chance for error when entering them into the computer. Also, the octal equivalents of each bit pattern are frequently included since it is very easy to load octal numbers into the front panel switches. All that is necessary is to remember the binary / octal equivalents for the decimal numbers 0-7.

The resulting program may appear thusly:

STEP	MNEMONIC	BIT PATTERN	OCTAL EQUIVALENT
0.	LDA	00 111 010	0 7 2
1.	(address)	10 000 000	2 0 0
2.	(address)	00 000 000	0 0 0
3.	MOV (A→B)	01 000 111	1 0 7
4.	LDA	00 111 010	0 7 2
5.	(address)	10 000 001	2 0 1
6.	(address)	00 000 000	0 0 0
7.	ADD (B+A)	10 000 000	2 0 0
8.	STA	00 110 010	0 6 2

STEP	MNEMONIC	BIT PATTERN	OCTAL EQUIVALENT
9.	(address)	10 000 010	2 0 2
10.	(address)	00 000 000	0 0 0
11.	JMP	11 000 011	3 0 3
12.	(address)	00 000 000	0 0 0
13.	(address)	00 000 000	0 0 0

The program can now be entered into the computer by means of the front panel switches. To begin loading the program at the first memory address (0), actuate the RESET switch. The Program Counter is now loaded with the first memory address. The program is then entered into the DATA/ADDRESS switches 7-0 one step at a time. After the first step is entered, actuate the DEPOSIT switch to load the bit pattern into the memory. Then enter the second step into the DATA/ADDRESS switches and actuate the DEPOSIT NEXT switch. The bit pattern will be automatically loaded into the next sequential memory address (1). Continue loading the steps into the front panel switches and actuating DEPOSIT NEXT. The complete program loading procedure can be summarized as follows:

STEP	SWITCHES 0-7	CONTROL SWITCH
		RESET
0	00 111 010	
		DEPOSIT
1	10 000 000	
		DEPOSIT NEXT
2	00 000 000	
		DEPOSIT NEXT
3	01 000 111	

STEP	SWITCHES 0-7	CONTROL SWITCH
		DEPOSIT NEXT
4	00 111 010	
		DEPOSIT NEXT
5	10 000 001	
		DEPOSIT NEXT
6	00 000 000	
		DEPOSIT NEXT
7	10 000 000	
		DEPOSIT NEXT
8	00 110 010	
		DEPOSIT NEXT
9	10 000 010	
		DEPOSIT NEXT
10	00 000 000	
		DEPOSIT NEXT
11	11 000 011	
		DEPOSIT NEXT
12	00 000 000	
		DEPOSIT NEXT
13	00 000 000	
		DEPOSIT NEXT

The program is now ready to be run, but first it is necessary to store data at each of the two memory addresses which are to be added together. To load the first address, set the DATA/ADDRESS switches to 10 000 000 and actuate EXAMINE. You can now load any desired number into this address by loading the DATA/ADDRESS switches as appropriate. When the number has been loaded into the switches, actuate DEPOSIT to load it into the memory. To load the next address, enter the second number on the DATA/ADDRESS switches and actuate DEPOSIT NEXT. Since sequential memory addresses were selected, the number will be automatically loaded into the proper address (10 000 001). If non-sequential memory addresses had been selected, the procedure for finding the first address would have to be followed (load the address into the DATA/ADDRESS switches and actuate EXAMINE; then load the number into the DATA/ADDRESS switches and actuate DEPOSIT).

Now that the two memory addresses referenced in the program have been loaded with two numbers to be added together, the program can be run. This is accomplished by simply actuating the RESET switch and then the RUN switch. Wait a moment and then actuate the STOP switch. To see the result stored in memory, actuate the appropriate DATA/ADDRESS switches with the bit pattern for the address into which the result was stored (10 000 010) and then actuate the EXAMINE switch. The result will then be displayed on the DATA LEDs.

To test your ability to load and run this program, try changing the memory addresses for the numbers to be added and the result and then load and run the program again.

SAMPLE PROGRAM FOR BINARY MULTIPLY

MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
MVIA	000	076	Multiplier to A Register
	001	002	
MVID	002	026	Multiplicand to D,E Registers
	003	003	
MVIE	004	036	
	005	000	
LXIH	006	041	Clear H,L Registers to initialize
	007	000	Partial Product
	010	000	
MVIB	011	006	Iteration Count to B Register
	012	010	
DADH	013	051	Shift Partial Product left into Carry
RAL	014	027	Rotate Multiplier Bit to Carry
JNC	015	322	Test Multiplier at Carry
	016	023	
	017	000	
DADD	020	031	Add Multiplicand to Partial Product if Carry = 1
ACI	021	316	
	022	000	
DCRB	023	005	Decrement Iteration Counter
JNZ	024	302	Check Iterations
	025	013	
	026	000	
SHLD	027	042	Store Answer in Locations 100,101
	030	100	
	031	000	
JMP	032	303	Restart
	033	000	
	034	000	

C. USING THE MEMORY

By now it is probably apparent that the memory plays a vital role in the efficient operation of a computer. Higher language compilers generally include a software package which automatically keeps track of the various memory addresses. Machine language operation, however, requires the programmer to keep track of the memory. Otherwise, valuable data or program instructions might be accidentally erased or replaced by other data or instructions.

You can keep track of what is stored in the *ALTAIR 8800*'s memory by means of a simple technique called memory mapping. This techmique merely assigns various types of data to certain blocks of memory reserved for a specific purpose. The technique effectively organizes the available memory into an efficient and readily accessible storage medium.

A typical memory map for the *ALTAIR 8800* with 256 words of memory might assign programs to the first 100 words, subroutines to the second 100 words, and data to the remaining 56 words. Of course the various blocks of memory can be modified at will, and the main purpose of memory mapping is to provide a cohesive organization of the available memory.

You can make a memory map each time you change the program in the *ALTAIR 8800*. After the program is written, decide how much memory space should be reserved for the program itself, the subroutines (if any), and the data. Then make a table or chart to record where various items are stored in the memory. Be sure to update the table when the memory organization is modified.

D. MEMORY ADDRESSING

The machine language instruction set for the *ALTAIR 8800* provides several methods for addressing the memory. They include direct addressing, register pair addressing, Stack Pointer addressing, immediate addressing, and stack addressing of subroutines. Each of these addressing methods will be described below.

1. Direct Addressing--The instruction supplies the specified memory address in the form of two bytes immediately following the actual instruction byte.

2. Register Pair Addressing--The contents of a register pair can contain a memory address. The H and L registers must be used for this purpose in most instructions. The H register contains the most significant 8 bits and the L register the least significant 8 bits (H is high and L is low). Two instructions (STAX and LDAX) permit the B and C or D and E register pairs to contain memory addresses.

3. Stack Pointer Addressing--There are only two stack operations: PUSH and POP. PUSHing data onto the stack causes two bytes (16 bits) of data to be stored in a special block of memory reserved by the programmer and called the stack. POPing data from the stack causes this data to be retrieved. The PUSH and POP instructions are explained in detail in Part 4 of this manual. For now it is important to know that the programmer must reserve the stack location in memory by loading a memory address into the Stack Pointer. This is accomplished by means of the LXI instruction (see Part 4). The programmer should always make note of the stack's address on his memory map.

4. Immediate Addressing--Immediate instructions contain data which is loaded into memory during program loading. Since the data is loaded along with the program in a sequential fashion, it is stored in the block of memory reserved for programming by the operator. There is no need to make any changes to the memory map when loading immediate data.

5. Stack Addressing of Subroutines--When a subroutine is CALLed by a program, the address of the next sequential instruction in the main program is automatically saved by being PUSHed onto the stack. When the subroutine has been executed, a RETURN instruction POPs the address from the stack and the main program continues execution.

E. OPERATING HINTS

As you gain experience in the operation of the *ALTAIR 8800*, you will devise methods for improving both the efficiency of your programs and the operation of the computer. Listed below are several helpful hints which you will find quite useful as you learn to operate the machine.

1. Proofreading Programs--To be safe, always proofread a program after it has been entered into the computer. This is done by returning to the first address in memory at which the program begins (actuate RESET if the program begins at memory location 0; otherwise, set the address on the ADDRESS switches and actuate EXAMINE). Check the DATA LEDs to make sure the first program step has been correctly entered. Then actuate EXAMINE NEXT and check the second step against the DATA LEDs. Continue proofreading in this fashion until the entire program has been checked. If an error is found, simply reenter the correct bit pattern on the DATA switches, actuate DEPOSIT, and continue proofreading by means of the EXAMINE NEXT switch.

2. Using NOPs--NOP is an instruction which specifies "No Operation" and is seemingly of little value. However, by scattering NOP instructions throughout a complicated program, considerable time can be saved if a program error requiring the addition of a new step or steps is found. The new instruction or data is simply entered into the program in place of the NOP instruction during the program proofreading. Always be sure to use the appropriate number of NOPs if it is felt a particular new instruction might be necessary. For example, if you think it might be necessary to add an LDA instruction to the program if it fails to execute properly, use 3 NOPs in a row at the required location. Three NOPs are required since the LDA instruction requires three separate bytes.

3. Debugging Programs--Occassionally it will be neccessary to "debug" a program. The need for debugging occurs when a program fails to execute properly because of errors (bugs). Debugging can be enhanced by use of the SINGLE STEP switch. This switch steps the computer through the program in machine cycles rather than complete program steps and permits you to observe the condition of the eight STATUS LEDs. This procedure will permit you to detect illegal entries, improper program organization, and other programming errors.

PART 4. *ALTAIR 8800* INSTRUCTION SET

The *ALTAIR 8800* has 78 basic machine language instructions. Since many of the instructions can be modified to affect different registers or register pairs, more than 200 variances of the basic instructions are possible.

A detailed description of the *ALTAIR 8800* instruction set is provided in the remainder of this operating manual. For the purpose of this description, the 78 basic machine language instructions have been grouped into seven major subdivisions:

A. Command Instructions

B. Single Register Instructions

C. Register Pair Instructions

D. Accumulator Instructions

E. Data Transfer Instructions

F. Immediate Instructions

G. Branching Instructions

Each instruction is presented as a standardized mnemonic or machine language code. Instructions may occupy from one to three sequential (serial) bytes, and the appropriate bit patterns are included. A condensed summary of the complete instruction set showing the mnemonics and instructions in both binary and octal is included as an Appendix.

A. COMMAND INSTRUCTIONS

The *ALTAIR 8800* has nine special purpose command instructions which are used to service the remaining instructions. These special purpose instructions occupy four catagories: Input/Output Instructions (IN, OUT), Interrupt Instructions (EI, DI, HLT, RST), Carry Bit Instructions (STC, CMC), and the No Operation Instruction (NOP).

1. INPUT/OUTPUT INSTRUCTIONS

There are two Input/Output Instructions and each occupies two bytes. The first byte is the instruction, and the second byte is the Input/Output device number.

IN	(INPUT)	11 011 011	(Byte 1)
		(Device No.)	(Byte 2)

Operation: An 8-bit data byte is loaded from the specified external device into the Accumulator.

Status Bits: Unaffected.

Example: Assume an input device contains the following data byte: 00 001 000. Implementation of the IN instruction (including device number) will cause the data byte to replace the contents of the Accumulator.

OUT	(OUTPUT)	11 010 011	(Byte 1)
		(Device No.)	(Byte 2)

Operation: An 8-bit data byte is loaded from the Accumulator into the specified output device.

Status Bits: Unaffected.

Example: Assume the Accumulator contains the following data byte: 00 001 000. Implementation of the OUT instruction (plus device number) will cause the data byte to be sent to the specified external device.

2. INTERRUPT INSTRUCTIONS

There are two specific Interrupt instructions (EI and DI) and two auxiliary Interrupt instructions. Interrupt instructions permit implementation of a program by a computer to be temporarily interrupted so that input/output interfacing may take place. For example, interrupts may be utilized by a computer's output device while an input device is entering data or a program.

EI (ENABLE INTERRUPTS) 11 111 011 (Byte 1)

Operation: Implementation of the EI instruction sets the interrupt flip-flop. This alerts the computer to the presence of interrupts and causes it to respond accordingly.

Status Bits: Unaffected.

DI (DISABLE INTERRUPTS) 11 110 011 (Byte 1)

Operation: Implementation of the DI instruction resets the interrupt flip-flop. This causes the computer to ignore any subsequent interrupt signals.

Status Bits: Unaffected.

HLT (HALT INSTRUCTION) 01 110 110 (Byte 1)

Operation: Implementation of the HLT instruction steps the Program Counter to the next instruction address and stops the computer until an interrupt occurs. The HLT instruction should not normally be implemented when a DI instruction has been executed. Since the DI instruction causes the computer to ignore interrupts, the computer will not operate again until the main power switch is turned off and then back on.

Status Bits: Unaffected.

RST (RESTART INSTRUCTION) 11 (esp) 111 (Byte 1)

Operation: The data byte in the Program Counter is pushed onto the stack. This provides an address for subsequent use by a RETURN instruction. Program execution then continues at memory address: 00 000 000 00 (exp) 000 where exp ranges from 000 to 111.

The RST instruction is normally used to service interrupts. The external device may cause a RST instruction to be executed during an interrupt. Implementation of RST then calls a special purpose subroutine which is stored in up to eight 8-bit bytes in the lower 64 words of memory. A RETURN instruction is included to return the computer to the original program.

Status Bits: Unaffected.

Example: Assume the following RST instruction is present: 11 001 111. Implementation of the instruction will cause the Program Counter data byte to be pushed onto the stack. The program will then continue execution at the subroutine located at memory address: 00 000 000 00 001 000. Upon completion of the subroutine, a RETURN instruction will return the computer to the next step in the main program.

3. CARRY BIT INSTRUCTIONS

There are two instructions which can be used to directly modify the status of the Carry Bit. Each instruction requires one 8-bit byte.

CMC (COMPLEMENT CARRY) 00 111 111 (Byte 1)

Operation: The Carry Bit is complemented. If it is initially 0, it is set to 1. If it is initially 1, it is reset to 0.

Status Bit Affected: Carry.

STC (SET CARRY) 00 110 111 (Byte 1)

Operation: The Carry Bit is set to 1.

Status Bit Affected: Carry.

4. NO OPERATION INSTRUCTION

There is one NO OPERATION instruction. It occupies a single 8-bit byte.

NOP (NO OPERATION) 00 000 000 (Byte 1)

Operation: No operation occurs, and the Program Counter

proceeds to the next sequential instruction. Program execution then continues.

Status Bits: Unaffected.

B. SINGLE REGISTER INSTRUCTIONS

The *ALTAIR 8800* has four single register instructions. Each instruction occupies a single byte. Two of the instructions, INR and DCR, have eight variances each. The variances are specified according to any desired register, and the following register bit patterns apply:

Register	Bit Pattern
B	000
C	001
D	010
E	011
H	100
L	101
Memory Reference M	110
A	111

If Memory Reference M (110) is specified in the instruction byte, the memory byte addressed by the contents of the H and L registers is processed. The H register contains the most significant 8 bits of the memory address and the L register contains the least significant 8 bits of the address.

INR (INCREMENT REGISTER OR MEMORY) 00 (reg) 100 (Byte 1)

Operation: The specified byte is incremented by one.

Status Bits Affected: Zero, Sign, Parity, and Auxiliary Carry.

Example: Assume the following instruction is present: 00 000 100. According to the table of register bit patterns given above, the byte in register B is to be incremented by 1. If the initial byte is 00 000 000, the incremented byte will be 00 000 001.

DCR (DECREMENT REGISTER OR MEMORY) 00 (reg) 101 (Byte 1)

Operation: The specified byte is decremented by one.

Status Bits Affected: Zero, Sign, Parity, and Auxiliary Carry.

Example: Assume the following instruction is present: 00 001 101. According to the table of register bit patterns given above, the byte in register C is to be decremented by 1. If the initial byte is 00 000 001, the decremented byte will be 00 000 000.

CMA (COMPLEMENT ACCUMULATOR) 00 101 111 (Byte 1)

Operation: Each bit in the accumulator is complemented (1s become 0s and 0s become 1s).

Status Bits: Unaffected.

Example: Assume the accumulator byte is 11 001 100. The instruction CMA will complement each bit in the accumulator byte as shown below:

11 001 100	Accumulator
00 110 011	Complemented Accumulator

DAA (DECIMAL ADJUST ACCUMULATOR) 00 100 111 (Byte 1)

Operation: The 8-bit accumulator byte is converted into two 4-bit BCD (binary-coded-decimal) numbers. The instruction affected by the Auxiliary Carry Bit.

The DAA instruction performs two operations:

1. If the least significant 4 bits in the accumulator byte (bits 0-3) represent a BCD digit greater than 9 or if the Auxiliary Carry Bit is set to 1, the four bits are automatically incremented by 6. If not, the accumulator is unaffected.

2. If the most significant 4 bits in the accumulator byte (bits 4-7) represent a BCD digit greater than 9 or if the Carry Bit is set to 1 after the previous operation, the four bits are automatically incremented by 6. If not, the accumulator is unaffected.

Status Bits Affected: Zero, Sign, Parity, Carry, and Auxiliary Carry.

Example: Assume the accumulator byte is 10 100 100. The DAA instruction will automatically consider the byte as two 4-bit bytes: 1010 0100. Since the value of the least significant 4 bits is less than 9, the accumulator is initially unaffected. The value of the most significant 4 bits is greater than 9, however, so the 4 bits are incremented by 6 to give 1 0000. The most significant bit sets the Carry Bit to 1, and the accumulator now contains: 00 000 100.