5. Where less computations are required.
6. In training kits.
7. While developing a program for microprocessor-based system for industrial control, instrumentation etc.
8. For industrial application.

5.4.3 High-level Language

High-level language is recommended for the following situations :

1. For large programs.
2. Where large volume of data is to be processed.
3. More computation is involved.
4. Applications requiring large memories.
5. For complex mathematical computation.
6. Applications where high cost is justified. The high cost is involved in peripherals and software supports associated with high-level language.

For microcomputers future trend is in the favour of high-level language due to the following reasons :

1. Hardware and memory are becoming less expensive.
2. Software and programmers are becoming more costly.
3. Large size of memory chips are available at low cost.
4. Compilers are easily available.
5. Efficient high-level languages are being developed.

5.5 STACK

During the execution of a program sometimes it becomes necessary to save the contents of certain registers because the registers are required for some other operation in subsequent steps. These contents are moved to certain memory locations by PUSH operation. Then the registers are used for other operations. After completing these operations those contents which were saved in the memory are transferred back to the registers by POP operation. Memory locations for this purpose is set aside by the programmer in the beginning. The set of memory locations kept for this purpose is called *stack*. The last memory location of the occupied portion of the stack is called stacktop. A special 16-bit register known as *stack pointer* holds the address of stacktop. The stack pointer is initialized in the beginning of the program by LXI SP or SPHL instruction. Any area of the RAM can be used as stack. There is no restriction on the location of the stack in the memory. Data are stored in the stack on last-in-first-out (LIFO) principle. Stack access is faster than memory access. Fig. 5.1 shows a typical stack and stacktop location. The SP register holds the address of stacktop location. Fig. 5.2 (*a*) shows the position of a stack before PUSH operation. Suppose that the contents of the register pair B-C is to be pushed. After PUSH operation the stack position has been shown in Fig. 5.2 (*b*). Similarly, POP operation is used to transfer the contents from the stack to the register. The stack position before and after the POP operation has been shown in Fig. 5.3(*a*) and (*b*) respectively.

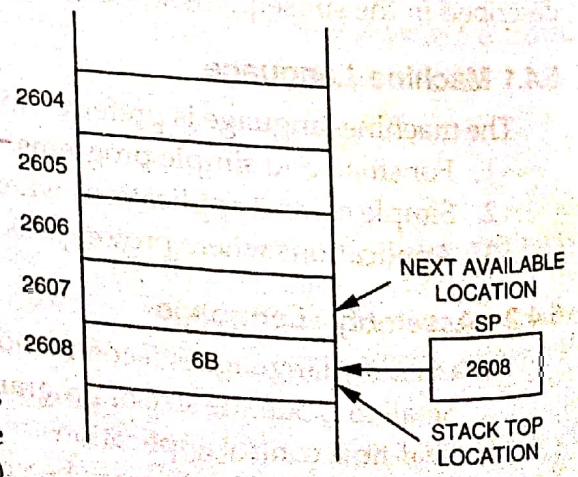

Fig. 5.1. Stack and stacktop location.

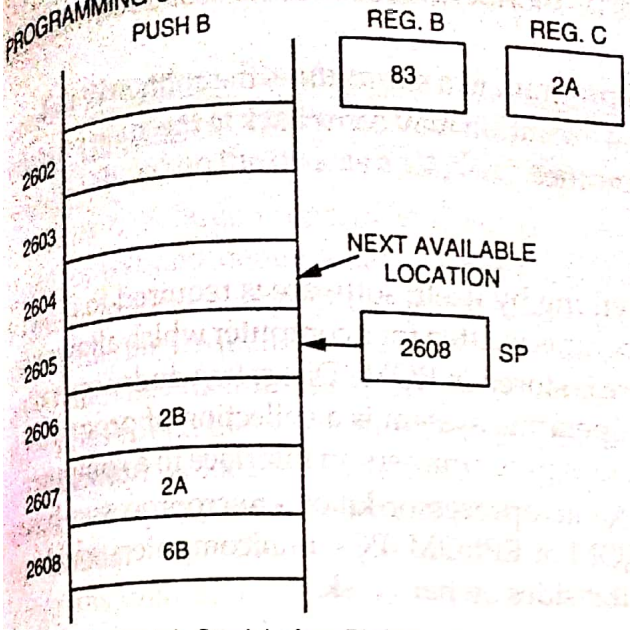

Fig. 5.2 (a). Stack before PUSH operation.

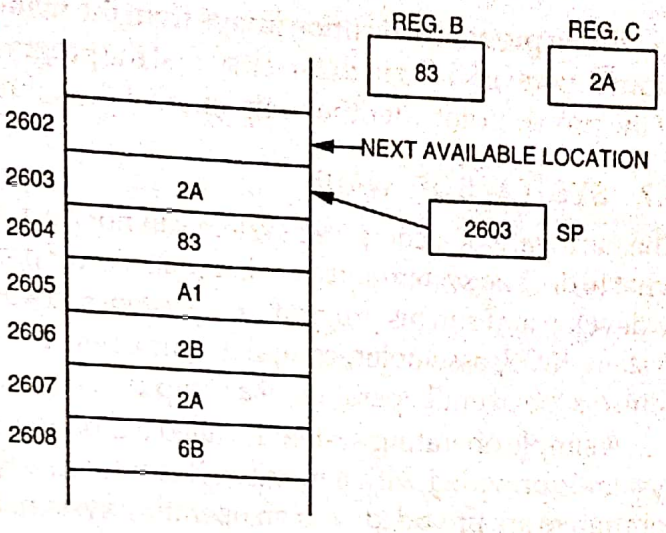

Fig. 5.2 (b). Stack after PUSH operation.

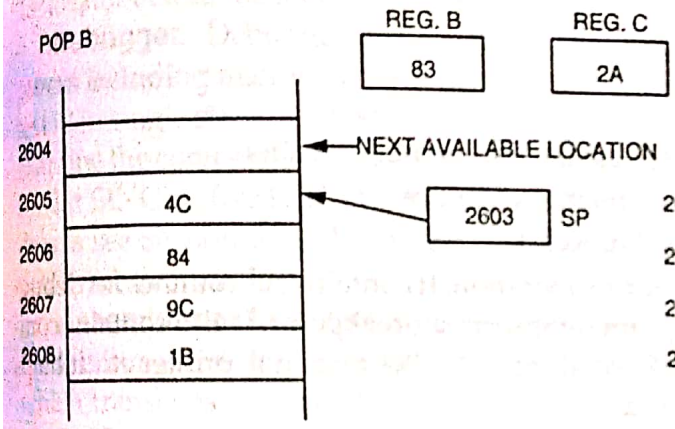

Fig. 5.3 (a). Stack before POP operation.

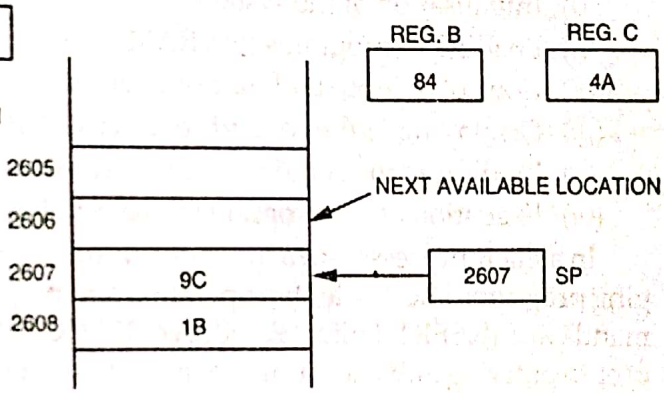

Fig. 5.3 (b). Stack after POP operation.

5.6 SUBROUTINES

While writing a program certain operations may occur several times and they are not available as individual instruction. The program for such operations are repeated again and again in the main program. A simple example of such operation is multiplication. The Intel 8085 does not have any instruction for multiplication and hence, a program is written to perform multiplication. Similarly, small programs are written for functions like sine, cosine, logarithm, square root etc. for Intel 8085. The concept of subroutines is used to avoid the repetition of smaller programs. The small program for a particular task is called *subroutine*. Subroutines are written separately and stored in the memory. They are called at various points of the main program by CALL instruction where they are required. The last instruction of the subroutine is RET. After the completion of a subroutine, the main program begins from the instruction immediately following the CALL instruction. There may be more than one subroutine in a program. The technique of subroutine saves memory locations. Some important subroutines may be written in a ROM by the manufacturer. Such subroutines are called by special instructions given by the manufacturer in the manual. Fig. 5.4 shows a CALL-RETURN structure.

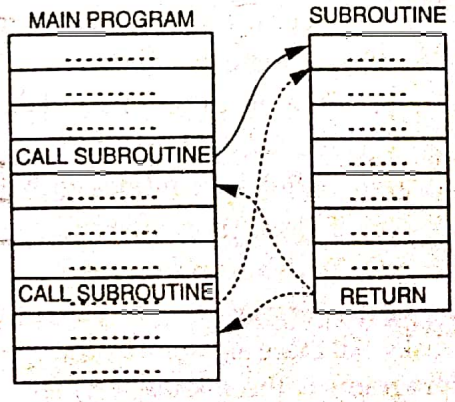

Fig. 5.4. CALL RETURN structure of subroutine.

When program execution jumps from the main program to a subroutine, the content of the program counter is saved in the stack so that the program execution may come back to the main program at the proper point after the completion of the subroutine.

7.5 INTERRUPTS OF INTEL 8085

The Intel 8085 has five interrupt inputs namely TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR. The TRAP has the highest priority, followed by RST 7.5, RST 6.5 and RST 5.5. The INTR has the lowest priority. When interrupts are to be used they are enabled by software using the instruction EI (Enable Interrupt) in the main program. Fig. 7.8 shows the schematic diagram of interrupts of Intel 8085. The instruction EI sets the interrupt enable flip-flop to enable the interrupts. The use of the instruction EI enables all the interrupts. The instruction DI (Disable Interrupt) is used to disable interrupts. In certain situation it may be desired to prevent the occurrence of interrupts while a particular task is being performed by the microprocessor. This can be done using DI instruction. The DI instruction resets the interrupt enable flip-flop and disables all the interrupts except nonmaskable interrupt TRAP, see Fig. 7.8. The system RESET also resets the Interrupt Enable flip-flop.

When an interrupt line goes high processor completes its current instruction and saves program counter on the stack. It also resets Interrupt Enable flip-flop before taking up ISS so that the occurrence of further interrupts by other devices is prevented during the execution of ISS. All the inter-

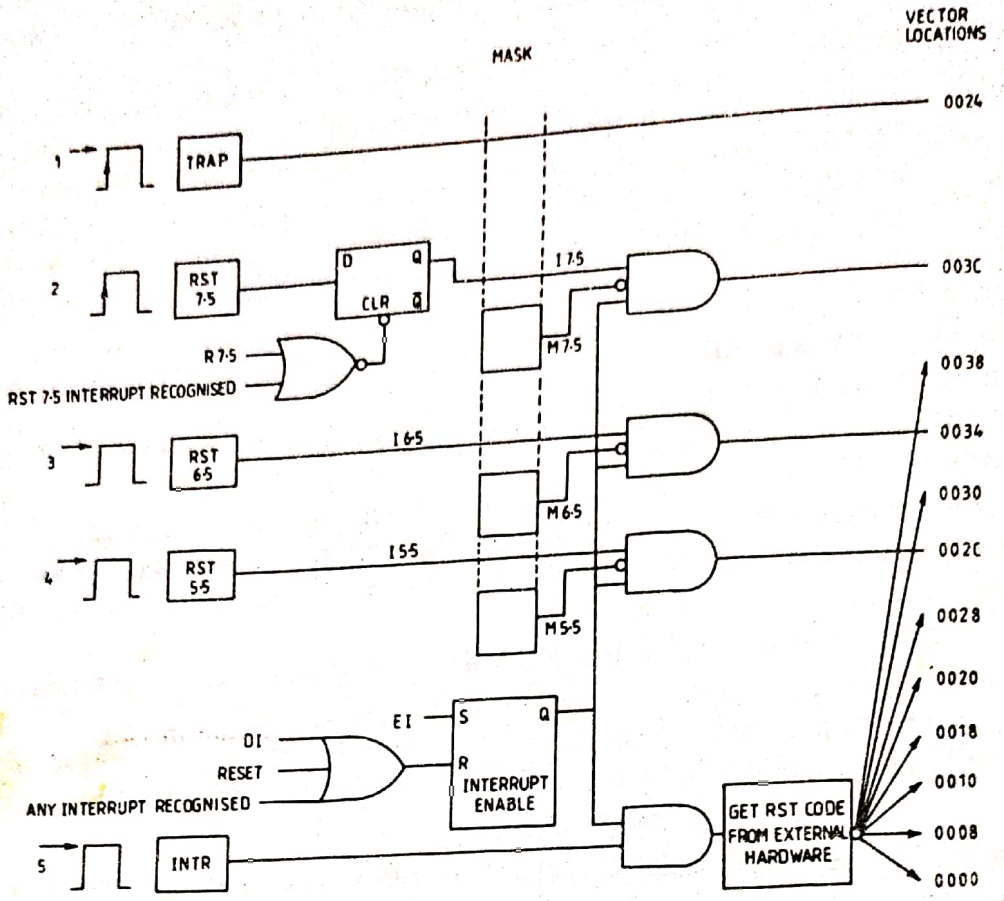

Fig. 7.8 Schematic Diagram of 8085 Interrupts.

rupts except TRAP are disabled by resetting the Interrupt Enable flip-flop.

The resetting of this flip-flop can be done in one of the three ways : by software using instruction DI, system reset or by recognition of an interrupt request.

Before the program returns back from ISS to the main program all the interrupts are to be enabled again. This is done using instruction EI in ISS before using the instruction RET.

At many occasions the programmer may like to prevent the occurrence of a few of several interrupts while microprocessor is performing certain tasks. This is done by masking off those interrupts which are not required to occur when certain task is being performed. The interrupts which can be masked off (*i.e.*, made ineffective) are called maskable interrupts. Masking is done by software. The Intel 8085 has two categories of interrupts : maskable and nonmaskable. The TRAP is a nonmaskable interrupt. It need not be enabled. It cannot be disabled. It is not accessible to user. It is used for emergency situation such as power failure and energy shut-off. RST 7.5, RST 6.5 and RST 5.5 are maskable interrupts.

7.5.1 Hardware and Software Interrupts

Interrupts caused by I/O devices are called *hardware interrupt*. The normal operation of a microprocessor can also be interrupted by abnormal internal conditions or special instructions. Such

an interrupt is called a *software interrupt*. RST n instructions of the 8085 are used for software interrupt. When RST n instruction is inserted in a program, the program is executed upto the point where RST n has been inserted. This is used in debugging of a program. See an example in Section 5.10.

The internal abnormal or unusual conditions which prevent the normal processing sequence of a microprocessor are also called exceptions. For example, divide by zero will cause an exception. Intel literatures do not use the term exception, whereas Motorola literatures use the term exception. Intel includes exception in software interrupt.

When several I/O devices are connected to INTR interrupt line, an external hardware is used to interface I/O devices. The external hardware circuit generates RST n codes to implement the multiple interrupt scheme. This will be discussed later in this chapter.

7.5.2 Interrupts Call-Locations

When an interrupt occurs the program is transferred to a specific memory location. Then the monitor transfers the program from the specific memory location to a memory location in RAM, from where the user can write the program for interrupt service sub-routine (ISS). For TRAP, RST 7.5, 6.5 and 5.5 the program is automatically transferred to specific memory locations without any external hardware. The necessary hardware is already provided within 8085. The specific memory locations for these interrupts are as follows :

Interrupt	*Call-Location in Hex*
TRAP	0024
RST 7.5	003C
RST 6.5	0034
RST 5.5	002C

An interrupt for which hardware automatically transfers the program to a specific memory location is known as *vectored interrupt*.

INTR CALL Locations There are 8 numbers of CALL-locations for INTR interrupt. Table 7.4 shows CALL-locations, RST n instructions and corresponding hex-code. For INTR external hardware is used to transfer program to specific CALL-location. The hardware circuit generates RST codes for this purpose. The INTR line is sampled by the microprocessor in the last state of the last machine cycle of each instruction. When INTR is high, the microprocessor saves the contents of the program counter on the stack and then sends an interrupt acknowledge signal, INTA to the external hardware. In response to INTA the external hardware generates a RST n code. When microprocessor receives this code, it transfers program to the corresponding CALL-location. Upto 8 number of I/O devices can be connected to INTR through an external hardware. Priority can be assigned to I/O devices connected to INTR through external hardware or interrupt controller. The external hardware recognizes which I/O device has interrupted and it generates proper RST code that causes microprocessor to take up ISS for that particular I/O device. An external hardware can be built employing prority encoder and a tri-state buffer, see Ref. 9. Alternatively, Priority Interrupt Controller 8214 can be used. Programmable interrupt controller 8259 is more versatile, and persent trend is to use programmable interrupt controller. INTA is used to activate 8259 or some other hardware. The 8259 has been described later in this chapter.

7.5.3 RST 7.5, 6.5 and 5.5

RST 7.5, RST 6.5 and RST 5.5 are maskable interrupts. These interrupts are enabled by software using instructions EI and SIM (Set Interrupt Mask). The execution of the instruction SIM enables/disables interrupts according to the bit pattern of the accumulator. Fig. 7.9 shows accumulator contents for SIM instruction.

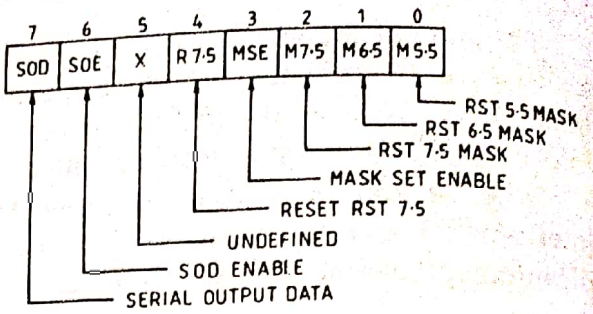

Fig. 7.9 Accumulator Content for SIM.

Bits 0 - 2 set/rest the mask bits of interrupt mask register for RST 5.5, 6.7 and 7.5. Bit 0 is for RST 5.5 mask, bit 1 for RST 6.5 mask and bit 2 for RST 7.5 mask. If a bit is set to 1 the corresponding interrupt is masked off (disabled). If it is set to 0, the corresponding interrupt is enabled. Bit 3 is set to 1 to make bits 0 - 2 effective. Bit 4 is an additional control for RST 7.5. If it is set to 1, the flip-flop for RST 7.5 is reset. Thus RST 7.5 is disabled regardless of whether bit 2 for RST 7.5 is 0 or 1.

Table 7.4 CALL-Locations and Hex-Codes for RST n

RST n	*Hex-Code*	*CALL-Locations*
RST 0	C7	0000
RST 1	CF	0008
RST 2	D7	0010
RST 3	DF	0018
RST 4	E7	0020
RST 5	EF	0028
RST 6	F7	0030
RST 7	FF	0038

Bits 6 and 7 are for serial data ouptput. The instruction SIM is also used for serial data transmission. Bit 6 is to enable SOD. If SIM instruction is executed the content of the 7th bit of the accumulator is output on SOD line of the microprocessor. The content of bit 7 may be either high (1) or low (0).

The instruction DI disables all the interrupts. This is not always desired. When the processor is performing a particular task it may be desired to prevent the occurrence of a few of the several interrupts. In such a situation the interrupts which are not desired to occur may be masked off using SIM instruction.

Triggering Levels. TRAP is edge as well as level triggered. This means that TRAP should go high and stay high until it is acknowledged. In this way false triggering caused by noise and transients is avoided. TRAP has also a flip-flop which is not shown in Fig. 7.8. The flip-flop is cleared when interrupt is acknowledged so that future interrupt may be entertained.

RST 7.5 is positive edge triggered interrupt. It can be triggered with a short duration pluse. RST 6.5 and 5.5 are level triggered interrupts. Triggering level for RST 6.5 and 5.5 has to be kept high until the microprocessor recognises these interrupts. If the processor does not recognise these interrupts immediately, their triggering level should be held by external hardware.

Example 1. Enable all the interrupts of Intel 8085.

The content of the accumulator for instructions SIM to enable RST 7.5, 6.5 and 5.5 are programmed as follows.

7	6	5	4	3	2	1	0
SOD	SOE	X	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	0	1	0	0	0 = 08

Bits 0, 1 and 2 are set to 0 to enable RST 7.5, 6.5 and 5.5. Bit 3 is set to make bits 0, 1 and 2 effective. Bit 4 is set to 0 to enable RST 7.5 as it is an additional control for RST 7.5.

Instructions

Mnemonic	*Operands*	*Comments*
EI		Enable all interrupts
MVI	A, 08	Get accumulator bit pattern to enable RST 7.5, 6.5 and 5.5
SIM		Enable RST 7.5, 6.5 and 5.5

Instruction EI and SIM both are essential to enable RST 7.5, 6.5 and 5.5. In addition to RST 7.5, 6.5 and 5.5 the instruction EI also enables INTR.

Example 2. Enable RST 6.5 and disable RST 7.5 and 5.5.

The contents of the accumulator to enable RST 6.5 and disable RST 7.5 and 5.5 are :

7	6	5	4	3	2	1	0
SOD	SOD	X	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	1	1	1	0	1 = 1D

Bit 1 is set to 0 to enable RST 6.5.

Bits 0 and 2 are set to 1 to mask off (disable) RST 5.5 and 7.5 respectively. Bit 4 which is an addition control for RST 7.5 is set to 1 to disable RST 7.5.

Bit 3 is set to 1 to make bits 0, 1 and 2 effective.

Instructions

EI	Enable interrupts
MVI A, 1D	Accumulator bit pattern to enable RST 6.5 and mask off RST 7.5 and 5.5.
SIM	Enable RST 6.5 and disable RST 7.5 and 5.5.

Example 3. Use RST 7.5 to interrupt Intel 8085.

For laboratory testing a very simple program, in which microprocessor is moving in a loop, has been considered. The interruption is made by applying a rising pulse at RST 7.5 terminal manually.

MAIN PROGRAM

Memory address	*Machine Codes*	*Labels*	*Mnemonics*	*Operands*	*Comments*
FC00	FB		EI		Enable all interrupts
FC01	3E, 08		MVI	A, 08	Get accumulator bit pattern to enable RST 7.5, 6.5 and 5.5.
FC03	30		SIM		

FC04	21, 50, FC	LOOP	LXI	H, FC50	Memory address of 1st number
FC07	7E		MOV	A, M	Get 1st number.
FC08	23		INX	H	
FC09	86		ADD	M	Add 1st and 2nd number.
FC0A	32, 52, FC		STA	FC52	Sum in FC52
FC0D	C3, 04, FC		JMP	LOOP	Jump to LOOP

DATA
FC50 — 05
FC01 — 02
SUM
FC52 — 07

In the beginning of the program all interrupts have been enabled as explained in Example 1. There is a simple program from FC04 to FC0C, to add two numbers and to store their sum in the memory location FC03. The program moves in a loop and repeats the same task again and again.

Now interrupt the microprocessor by applying a pulse to RST 7.5 line. For this purpose make contact of RST 7.5 line to ground terminal, then to 5 V supply. Now the program will jump to the memory location 003C which is the specific memory location for RST 7.5. The manufacturer of the Vinytics microprocessor kit has given the following monitor program :

003C C3, BD, FF JMP FFBD.

Therefore, the monitor transfers the program from 003C to the memory location FFBD. Now the user has to transfer the program from FFBD to a memory location in RAM from which service subroutine for RST 7.5 has been written.

User's JMP Instruction

Transfer program from FFBD to FC20.

FFBD C3, 20, FC JMP FC20 ; Transfer program from FFBD to FC20.

FC20 is the starting address of the interrupt service subroutine.

Interrupt Service subroutine

Memory address	Machine Codes	*Mnemonics*	*Operands*	*Comments*
FC20	3A, 00, FD	LDA	FD00	Get content of FD00.
FC23	32, 30, FC	STA	FC30	Store it in FC30.
FC26	2A, 01, FD	LHLD	FD01	Get contents of FD01 and FD02.
FC29	22, 31, FC	SHLD	FC31	Store them in FC31 and FC32.
FC2C	FB	EI		Enable interrupts
FC2D	C9	RET		Return to main program.

DATA
FD00 — 01
FD02 — 02
FD03 — 05
Result
FC30 — 01
FC31 — 02
FC32 — 05

The service subroutine is a simple program to transfer the contents of memory locations FD00, FD01 and FD02 to memory locations FC30, FC31 and FC32 respectively. When interrupt occurs the microprocessor transfers the program from main program to memory location 003C. Then the program is transferred to FFBD by the monitor. From FFBD the program is transferred to service subroutine. After performing the task of service subroutine the program goes back to the main program. The processor disables all the interrupts before it enters service subroutine. Therefore, at the end of service subroutine instruction EI is used to enable interrupts so that further interrupts will be recognised. After making interrupt see the result in FC30 – FC32.

Similarly user can try with RST 6.5 and 5.5. The monitor program for these interrupts for Vinytics kit are :

0034	C3, B7, FF,	JMP	FFB7	It is for RST 6.5
002C	C3, 09, 05	JMP	0509	It if for RST 5.5.

As RST 6.5 and 5.5 are level triggered, to make an interrupt the interrupt line has to be kept simply high till the processor recognises the interrupt. For laboratory testing as explained above the user may simply make contact of interrupt line to $5V_{d.c}$ terminal on the board.

Pending Interrupts. When one interrupt request is being served, other interrupt may occur resulting in a pending request. When more than one interrupts occur simultaneously, the interrupt having higher priority is served and the interrupts with lower priority remain pending. As 8085 contains several interrupt lines; an interrupt may occur while other interrupt is being served resulting in a pending interrupt. The 8085 has an instruction RIM using which the programmer can know the current status of the pending interrupts. This instruction gives the current status of only maskable interrupts. In case of INTR the processor reads its status in the last state of the last machine cycle of an instruction. When the processor returns to the main program after serving interrupt service subroutine it knows the status of INTR line as soon as it makes further execution of the main program.

Fig. 7.10 shows accumulator contents after the execution of instruction RIM. Bits 0-2 are for interrupt masks; 1 = masked. Bit 3 indicates interrupt enable flag; 1 = enabled. Bits 4-6 indicate pending interrupts; 1 = pending. Bit 7-serial input data, if any.

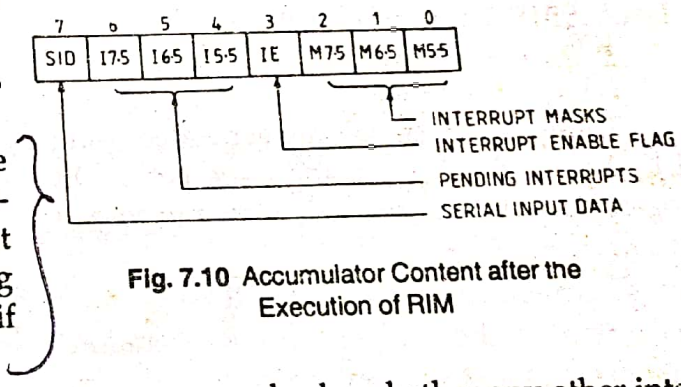

Fig. 7.10 Accumulator Content after the Execution of RIM

After executing the interrupt service subroutine the processor checks whether any other interrupt is pending using RIM instruction. If an interrupt is pending the processor executes its interrupt service subroutine before it returns to the main program.