Data Transfer Schemes

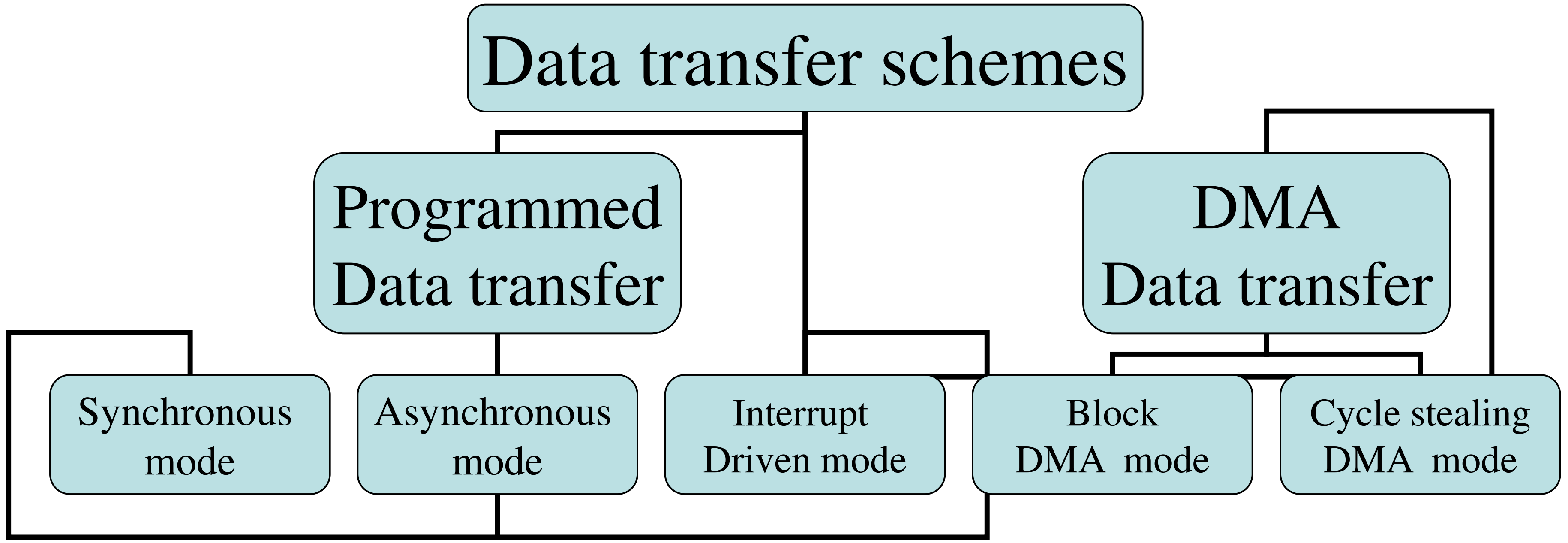
Why do we need data transfer schemes ?

- Availability of wide variety of I/O devices because of variations in manufacturing technologies e.g. electromechanical, electrical, mechanical, electronic etc.

- Enormous variation in the range of speed.

- Wide variation in the format of data.

Classification of Data Transfer Schemes



Programmed Data Transfer Scheme

- The data transfer takes place under the control of a program residing in the main memory.
- These programs are executed by the CPU when an I/O device is ready to transfer data.
- To transfer one byte of data, it needs to execute some instructions.
- This scheme is very slow and thus suitable when small amount of data is to be transferred.

Programmed data transfer

- Programmed data transfer is written and controlled by programmer and executed by the processor.
- The data transfer between processor and I/O devices or vice versa takes place by executing the corresponding instruction/program.
- Programmed I/O data transfers are identical to read and write operations for memories or devices.
- Example instructions – MOV M,A, IN 01, OUT 02 etc.

Programmed data transfer contd..

- The execution of programmed data transfer can take place at predefined period determined by the programmer.
- Based on the time of execution of the data transfer instruction, the programmed data transfer is divided into following three types namely:

a) **Synchronous mode of data transfer**
b) **Asynchronous mode of data transfer**
c) **Interrupt driven data transfer**

Synchronous Mode of Data Transfer

- Its used for I/O devices whose *timing characteristics* are fast enough to be compatible in speed with the communicating Processor.

- In this case the status of the I/O device is not checked before data transfer.

- The data transfer is executed using IN and OUT instructions.

- In simple or synchronous mode, the data is read from the input device by the processor irrespective of the status of the input device.

- It is assumed that the input device is ready with the data as and when the processor reads the data and the input device is in synchronism with the processor.

- Similarly, the data is written onto the output device irrespective of its status assuming that the output device is in synchronism with the processor.

- Memory compatible with Processors are available. Hence this method is invariably used with compatible memory devices.

- The I/O devices compatible in speed with Processor are usually not available. Hence this technique is rarely used for I/O

Asynchronous Data Transfer

- This method of data transfer is also called ***Handshaking mode***.

- This scheme is used when speed of I/O device does not match with that of Processor and the timing characteristics are not predictable.

- The Processor first sends a request to the device and then keeps on checking its status.

- The data transfer instructions are executed only when the I/O device is **ready** to accept or supply data.
- Each data transfer is preceded by a requesting signal sent by MPU and READY signal from the device.
- In this mode of data transfer, the data is read from an input device when the processor or CPU is ready and executes the data transfer instruction.
- If the input device is not ready the processor will wait until the device is ready with data.

- Similarly, the data is written onto an output device by the processor when it executes the data ‘write’ instruction to the corresponding output device.

- The program is written and the processor will wait in a loop until the output device is **ready** to receive data.

- As it can be seen clearly, the **processor’s time is wasted** in this mode of data transfer as it **waits for the device** to be ready.

Disadvantages

- A lot of Processor time is wasted during looping to check the device status which may be prohibitive in many time critical situations.

- Some simple devices may not have status signals. In such a case Processor goes on checking whether data is available on the port or not. e.g a keyboard interfaced to MPU.

Interrupt driven data transfer
Basic Interrupt Structure

- Interrupt is a mechanism by which the processor is made to transfer control from its current program execution to another program of more importance or higher priority.
- The interrupt signal may be given to the processor by any external peripheral device.
- Interrupts are in general generated by a variety of sources either internal or external to the CPU.
- Interrupts are the primary means by which Input and Output devices obtain the services of the CPU.

Interrupts

- Interrupt is a process where an external device can get the attention of the microprocessor.
  - The process starts from the I/O device

- Classification of Interrupts
  - Interrupts can be classified into two types:
    - Maskable Interrupts (Can be delayed or Rejected)
    - Non-Maskable Interrupts (Can not be delayed or Rejected)

- Interrupts can also be classified into:
    - Vectored (the address of the service routine is hard-wired)
    - Non-vectored (the address of the service routine needs to be supplied externally by the device)

Interrupts

- An interrupt is considered to be an emergency signal that may be serviced. The Microprocessor may respond to it as soon as possible.

- What happens when MP is interrupted ?
  - When the Microprocessor receives an interrupt signal, it suspends the currently executing program and jumps to an Interrupt Service Routine (ISR) to respond to the incoming interrupt. Each interrupt will most probably have its own ISR.

Responding to Interrupts

- Responding to an interrupt may be immediate or delayed depending on whether the interrupt is maskable or non-maskable and whether interrupts are being masked or not.
- There are two ways of redirecting the execution to the ISR depending on whether the interrupt is vectored or non-vectored.
  - Vectored: The address of the subroutine is already known to the Microprocessor
  - Non Vectored: The device will have to supply the address of the subroutine to the Microprocessor

The 8085 Interrupts:

- When a device interrupts, it actually wants the MP to give a service which is equivalent to asking the MP to call a subroutine. This subroutine is called ISR (Interrupt Service Routine)
- The ‘EI’ instruction is a one byte instruction and is used to Enable the non-maskable interrupts.
- The ‘DI’ instruction is a one byte instruction and is used to Disable the non-maskable interrupts.
- The 8085 has a single Non-Maskable interrupt.
  - The non-maskable interrupt is not affected by the value of the Interrupt Enable flip flop.

The 8085 Interrupts

- The 8085 has 5 interrupt inputs.
- The INTR input.
    - The INTR input is the only non-vectored interrupt.
    - INTR is maskable using the EI/DI instruction pair.

- RST 5.5, RST 6.5, RST 7.5 are all automatically vectored.
    - RST 5.5, RST 6.5, and RST 7.5 are all maskable.

- TRAP is the only non-maskable interrupt in the 8085
    - TRAP is also automatically vectored

The 8085 Interrupts

Interrupt name	Maskable	Vectored
INTR	Yes	No
RST 5.5	Yes	Yes
RST 6.5	Yes	Yes
RST 7.5	Yes	Yes
TRAP	No	Yes

Interrupt Vectors and the Vector Table

- An interrupt vector is a pointer to where the ISR is stored in memory.
- All interrupts (vectored or otherwise) are mapped onto a memory area called the Interrupt Vector Table (IVT).
    - The IVT is usually located in memory page 00 (0000H - 00FFH).
    - The purpose of the IVT is to hold the vectors that redirect the microprocessor to the right place when an interrupt arrives.

Cont..

- Example:
- Let , a device interrupts the Microprocessor using the RST 7.5 interrupt line.

  - Because the RST 7.5 interrupt is vectored, Microprocessor knows , in which memory location it has to go using a call instruction to get the ISR address. RST7.5 is knows as Call 003Ch to Microprocessor. Microprocessor goes to 003C location and will get a JMP instruction to the actual ISR address. The Microprocessor will then, jump to the ISR location

The 8085 Non-Vectored Interrupt Process

1. The interrupt process should be enabled using the EI instruction.
2. The 8085 checks for an interrupt during the execution of every instruction.
3. If INTR is high, MP completes current instruction, disables the interrupt and sends INTA (Interrupt acknowledge) signal to the device that interrupted
4. INTA allows the I/O device to send a RST instruction through data bus.
5. Upon receiving the INTA signal, MP saves the memory location of the next instruction on the stack and the program is transferred to ‘call’ location (ISR Call) specified by the RST instruction

The 8085 Non-Vectored Interrupt Process

6. Microprocessor Performs the ISR.
7. ISR must include the ‘EI’ instruction to enable the further interrupt within the program.
8. RET instruction at the end of the ISR allows the MP to retrieve the return address from the stack and the program is transferred back to where the program was interrupted.

The 8085 Non-Vectored Interrupt Process

- The 8085 recognizes 8 RESTART instructions: RST0 - RST7.
  - each of these would send the execution to a predetermined hard-wired memory location:

Restart Instruction	Equivalent to
RST0	CALL 0000H
RST1	CALL 0008H
RST2	CALL 0010H
RST3	CALL 0018H
RST4	CALL 0020H
RST5	CALL 0028H
RST6	CALL 0030H
RST7	CALL 0038H

Issues in Implementing INTR Interrupts

- How long must INTR remain high?
  - The microprocessor checks the INTR line one clock cycle before the last T-state of each instruction.
  - The INTR must remain active long enough to allow for the longest instruction.
  - The longest instruction for the 8085 is the conditional CALL instruction which requires 18 T-states.
- Therefore, the INTR must remain active for 17.5 T-states.
- If f= 3MHZ then T=1/f and so, INTR must remain active for [(1/3MHZ) * 17.5 ≈ 5.8 micro seconds].

Issues in Implementing INTR Interrupts

- How long can the INTR remain high?
  - The INTR line must be deactivated before the EI is executed. Otherwise, the microprocessor will be interrupted again.
  - Once the microprocessor starts to respond to an INTR interrupt, INTA becomes active (=0).

Therefore, INTR should be turned off as soon as the INTA signal is received.

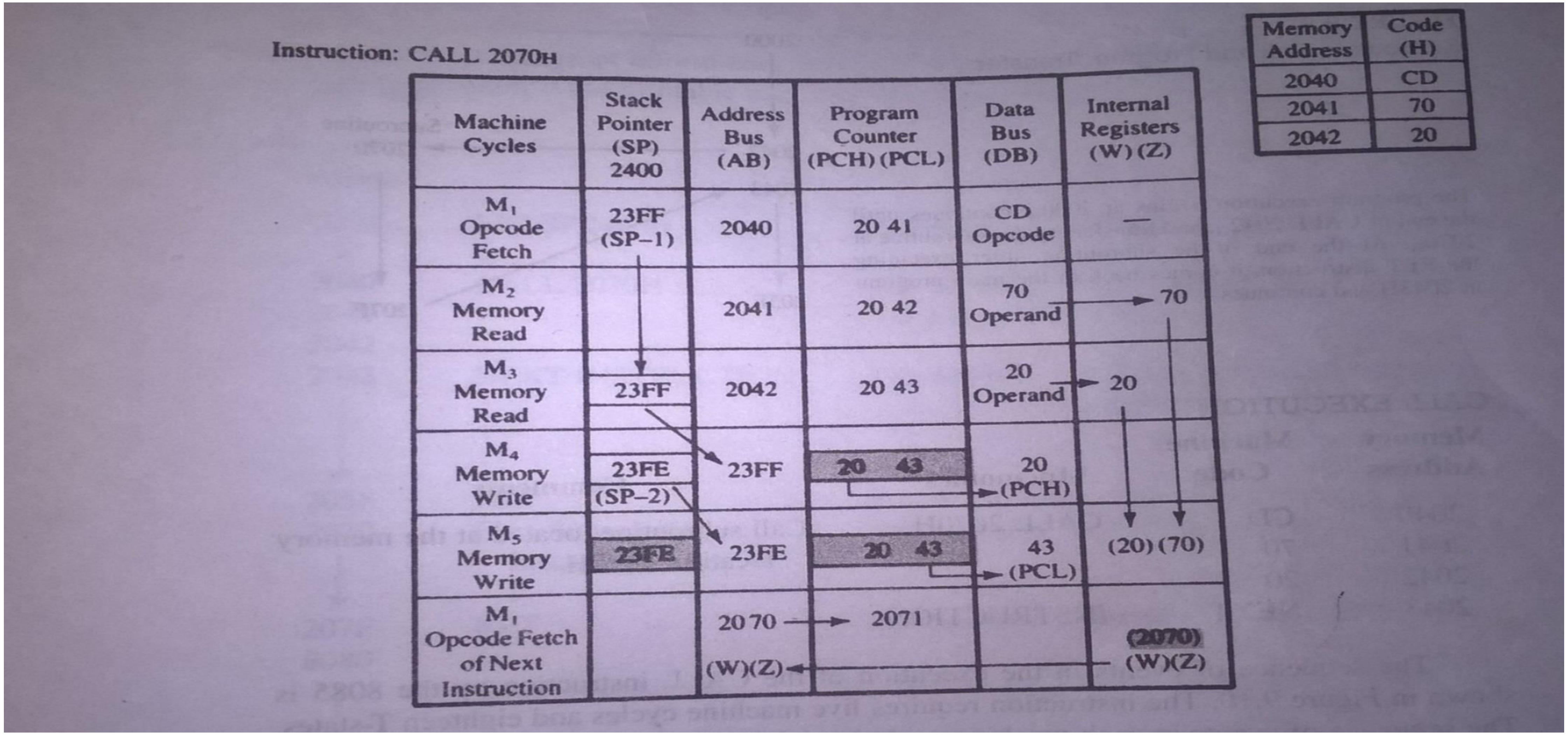
Issues in Implementing INTR Interrupts

- Can the microprocessor be interrupted again before the completion of the ISR?
  - As soon as the 1st interrupt arrives, all maskable interrupts are disabled.
  - They will only be enabled after the execution of the EI instruction.

Therefore, the answer is: “only if we allow it to”.

If the EI instruction is placed early in the ISR, other interrupt may occur before the ISR is done.

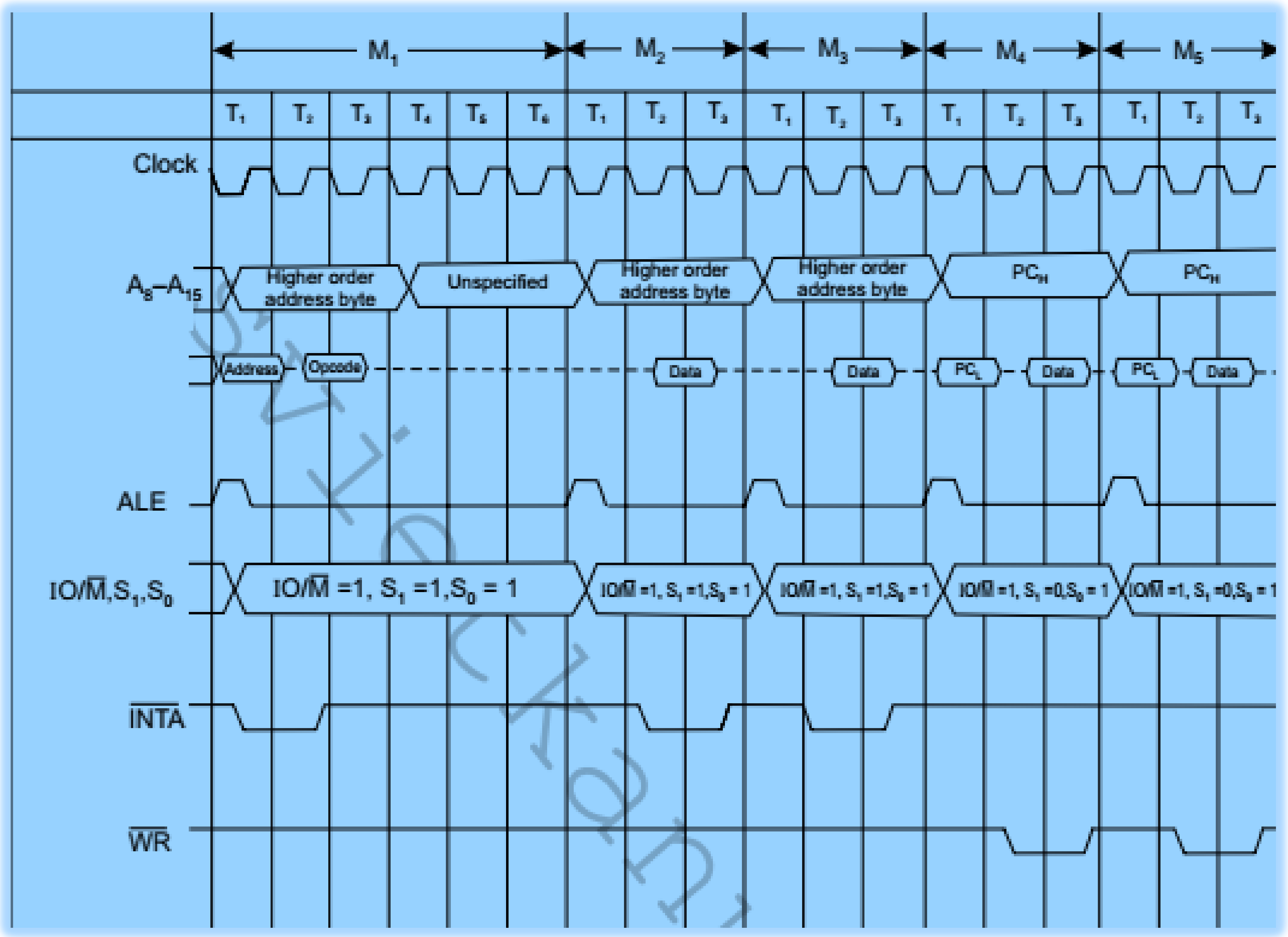
Call sequence..



Instruction: CALL 2070H

Memory Address	Code (H)
2040	CD
2041	70
2042	20

Machine Cycles	Stack Pointer (SP) 2400	Address Bus (AB)	Program Counter (PCH) (PCL)	Data Bus (DB)	Internal Registers (W) (Z)
M_1 Opcode Fetch	23FF (SP–1)	2040	20 41	CD Opcode	—
M_2 Memory Read		2041	20 42	70 Operand →	70
M_3 Memory Read	23FF	2042	20 43	20 Operand →	20
M_4 Memory Write	23FE (SP–2)	23FF	20 43	20 (PCH)	
M_5 Memory Write	23FE	23FE	20 43	43 (PCL)	(20) (70)
M_1 Opcode Fetch of Next Instruction		20 70 → ; (W)(Z) ←	2071		(2070) (W)(Z)



M1
M2
M3
M4
M5
T1 T2 T3 T4 T5 T6 T1 T2 T3 T1 T2 T3 T1 T2 T3 T1 T2 T3
Clock
A8–A15
Higher order address byte
Unspecified
Higher order address byte
Higher order address byte
PCH
PCH
Address
Opcode
Data
Data
PCL
Data
PCL
Data
ALE
IO/M̄,S1,S0
IO/M̄ =1, S1 =1,S0 = 1
IO/M̄ =1, S1 =1,S0 = 1
IO/M̄ =1, S1 =1,S0 = 1
IO/M̄ =1, S1 =0,S0 = 1
IO/M̄ =1, S1 =0,S0 = 1
INTA
WR

Restart Sequence

- The restart sequence is made up of three machine cycles
  - In the 1st machine cycle:

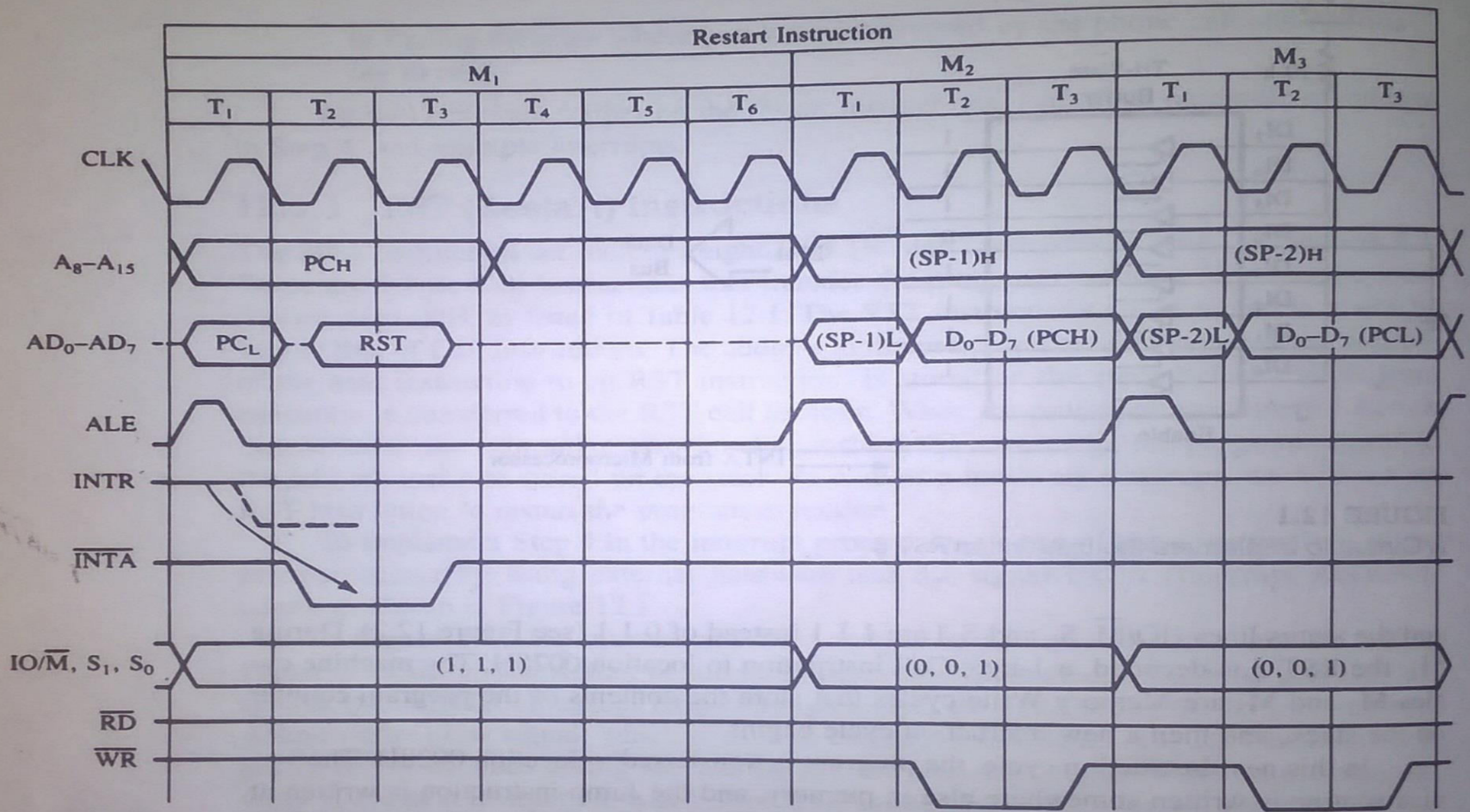
    The microprocessor sends the INTA signal.

    While INTA is active the microprocessor reads the data lines expecting to receive, from the interrupting device, the opcode for the specific RST instruction.
  - In the 2nd machine cycles:

    The address of stack pointer(next location address) is placed on address bus then high-order bits of PC is stored on stack.
  - In the 3rd machine cycles:

    The low order bits of PC is stored in the next location of the stack.

    Then the microprocessor jumps to the address associated with the specified RST instruction.


Restart Instruction
M1
M2
M3
T1
T2
T3
T4
T5
T6
T1
T2
T3
T1
T2
T3
CLK
A8–A15
PCH
(SP-1)H
(SP-2)H
AD0–AD7
PCL
RST
(SP-1)L
D0–D7 (PCH)
(SP-2)L
D0–D7 (PCL)
ALE
INTR
INTA
IO/M, S1, S0
(1, 1, 1)
(0, 0, 1)
(0, 0, 1)
RD
WR

Hardware Generation of RST Opcode

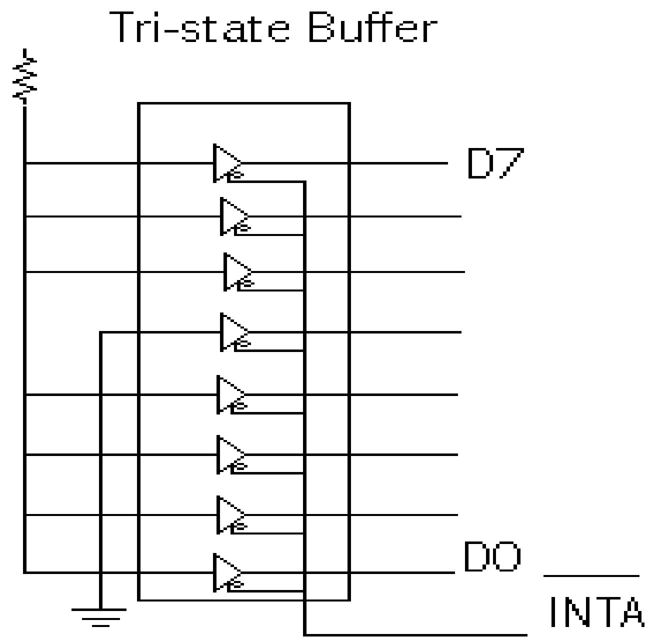
- How does the external device produce the opcode for the appropriate RST instruction?
  - The opcode is simply a collection of bits.
  - So, the device needs to set the bits of the data bus to the appropriate value in response to an INTA signal.

Hardware Generation of RST Opcode

The following is an example of generating RST 5:

RST 5’s opcode is EF =

D D
76543210
11101111



Hardware Generation of RST Opcode

- During the interrupt acknowledge machine cycle, (the 1st machine cycle of the RST operation):
  - The Microprocessor activates the INTA signal.
  - This signal will enable the Tri-state buffers, which will place the value EFH on the data bus.
  - Therefore, sending the Microprocessor the RST 5 instruction.
- The RST 5 instruction is exactly equivalent to CALL 0028H

The 8085 Maskable/Vectored Interrupts

- The 8085 has 4 Masked/Vectored interrupt inputs.
  - RST 5.5, RST 6.5, RST 7.5
    - They are all maskable.
      - They are automatically vectored according to the following table:

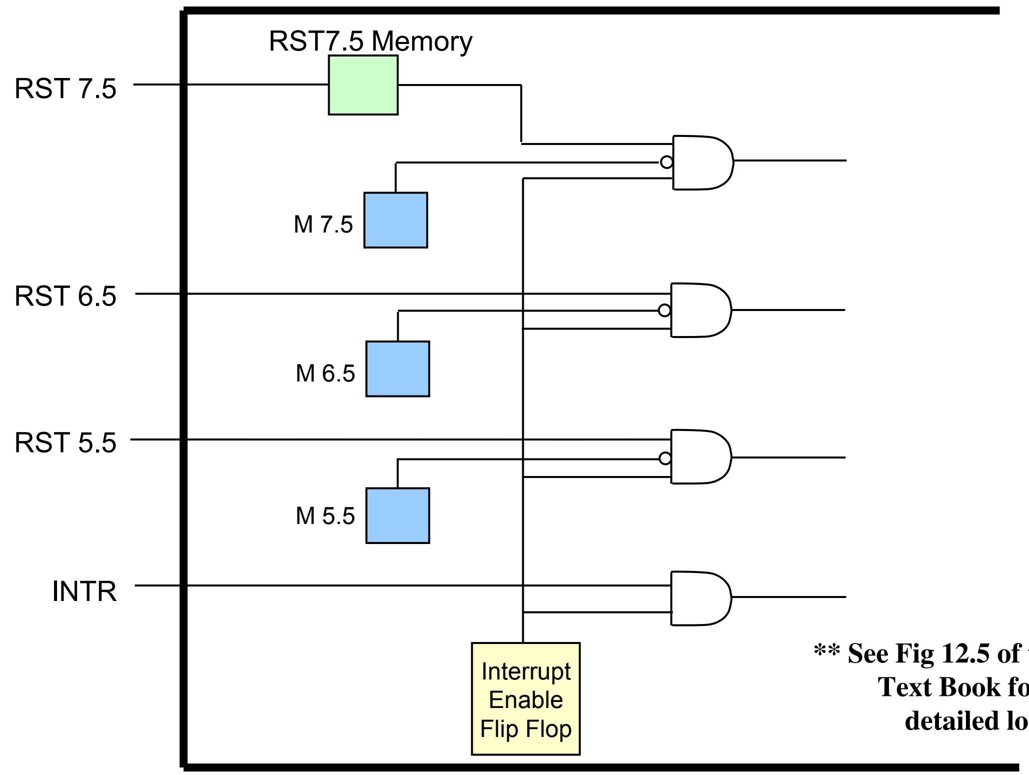
Interrupt	Vector
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH

    - The vectors for these interrupt fall in between the vectors for the RST instructions. That’s why they have names like RST 5.5 (RST 5 and a half).

Masking RST 5.5, RST 6.5 and RST 7.5

- **These three interrupts are masked at two levels:**
  - Through the Interrupt Enable flip flop and the EI/DI instructions.
    - The Interrupt Enable flip flop controls the whole maskable interrupt process.
  - Through individual mask flip flops that control the availability of the individual interrupts.These flip flops control the interrupts individually.

Maskable Interrupts and vector locations



**** See Fig 12.5 of the Text Book for a detailed look**

The 8085 Maskable/Vectored Interrupt Process

1. The interrupt process should be enabled using the EI instruction.
2. The 8085 checks for an interrupt during the execution of every instruction.
3. If there is an interrupt, and if the interrupt is enabled using the interrupt mask, the microprocessor will complete the executing instruction, and reset the interrupt flip flop.
4. The microprocessor then executes a call instruction that sends the execution to the appropriate location in the interrupt vector table.

The 8085 Maskable/Vectored Interrupt Process

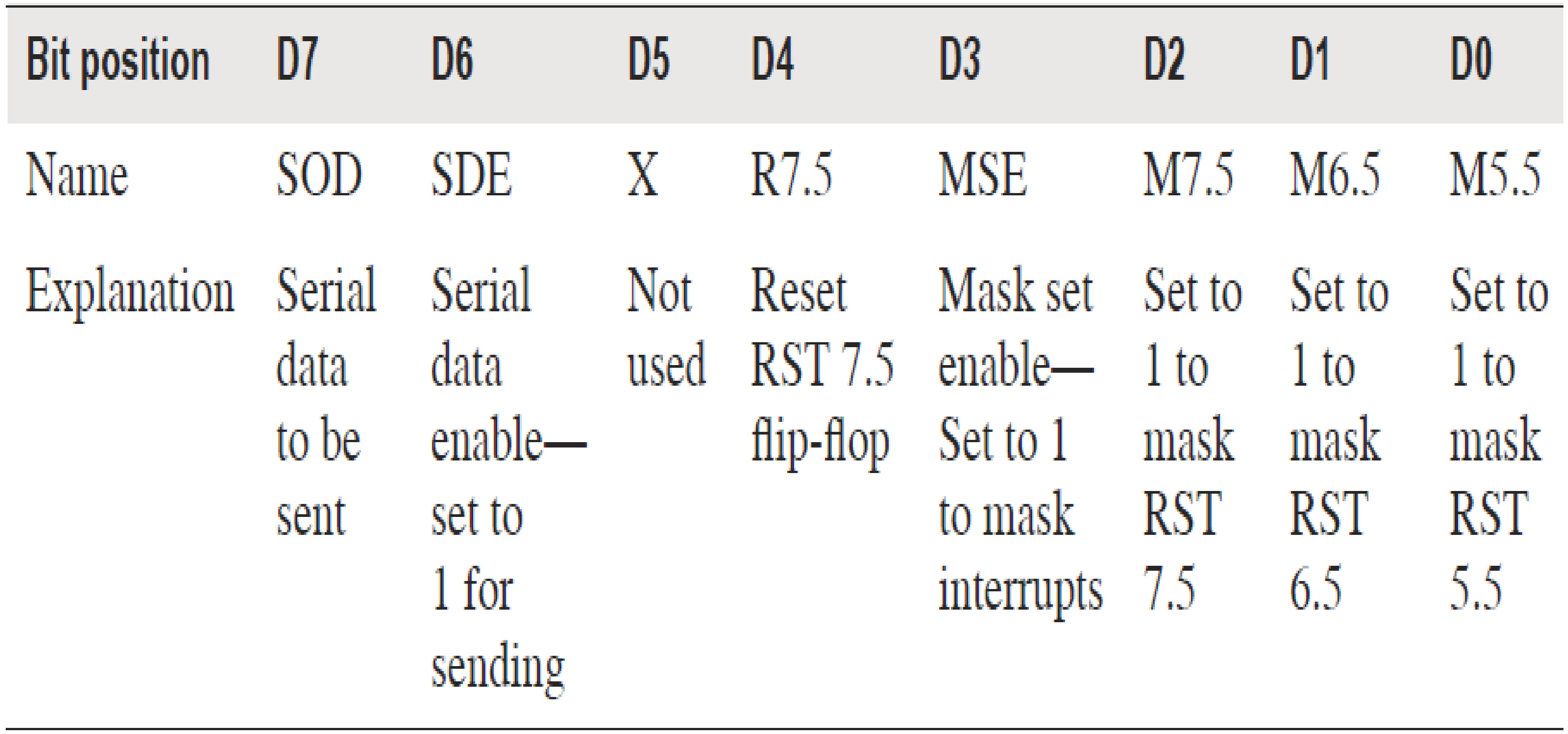
5. When the microprocessor executes the call instruction, it saves the address of the next instruction on the stack.
6. The microprocessor jumps to the specific service routine.
7. The service routine must include the instruction EI to re-enable the interrupt process.
8. At the end of the service routine, the RET instruction returns the execution to where the program was interrupted.

Manipulating the Masks

- The Interrupt Enable flip flop is manipulated using the EI/DI instructions.

- The individual masks for RST 5.5, RST 6.5 and RST 7.5 are manipulated using the SIM instruction.
  - This instruction takes the bit pattern in the Accumulator and applies it to the interrupt mask enabling and disabling the specific interrupts.

How SIM Interprets the Accumulator



Bit position	D7	D6	D5	D4	D3	D2	D1	D0
Name	SOD	SDE	X	R7.5	MSE	M7.5	M6.5	M5.5
Explanation	Serial data to be sent	Serial data enable—set to 1 for sending	Not used	Reset RST 7.5 flip-flop	Mask set enable—Set to 1 to mask interrupts	Set to 1 to mask RST 7.5	Set to 1 to mask RST 6.5	Set to 1 to mask RST 5.5

SIM and the Interrupt Mask

- Bit 0 is the mask for RST 5.5, bit 1 is the mask for RST 6.5 and bit 2 is the mask for RST 7.5.
    - If the mask bit is 0, the interrupt is available.
    - If the mask bit is 1, the interrupt is masked.
- Bit 3 (Mask Set Enable - MSE) is an enable for setting the mask.

  If it is set to 0 the mask is ignored and the old settings remain.

  If it is set to 1, the new setting are applied.
- The SIM instruction is used for multiple purposes and not only for setting interrupt masks.
- It is also used to control functionality such as Serial Data Transmission. Therefore, bit 3 is necessary to tell the microprocessor whether or not the interrupt masks should be modified

SIM and the Interrupt Mask

- The RST 7.5 interrupt is the only 8085 interrupt that has memory.
    - If a signal on RST7.5 arrives while it is masked, a flip flop will remember the signal.
    - When RST7.5 is unmasked, the microprocessor will be interrupted even if the device has removed the interrupt signal.
    - This flip flop will be automatically reset when the microprocessor responds to an RST 7.5 interrupt.

- Bit 4 of the accumulator in the SIM instruction allows explicitly resetting the RST 7.5 memory even if the microprocessor did not respond to it.
- Bit 5 is not used by the SIM instruction

Using the SIM Instruction to Modify the Interrupt Masks

- Example: Set the interrupt masks so that RST5.5 is enabled, RST6.5 is masked, and RST7.5 is enabled.
  - First, determine the contents of the accumulator

- Enable 5.5	bit 0 = 0
- Disable 6.5	bit 1 = 1
- Enable 7.5	bit 2 = 0
- Allow setting the masks	bit 3 = 1
- Don’t reset the flip flop	bit 4 = 0
- Bit 5 is not used	bit 5 = 0
- Don’t use serial data	bit 6 = 0
- Serial data is ignored	bit 7 = 0

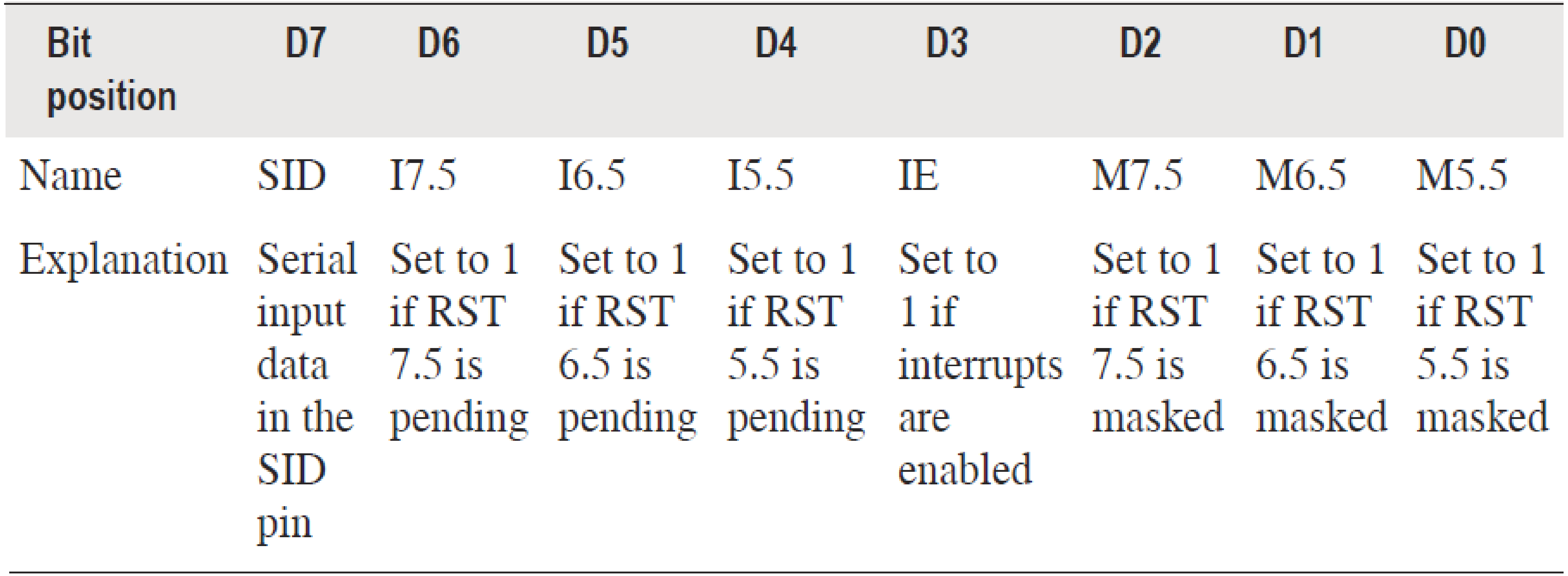
SDO	SDE	XXX	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	0	1	0	1	0

Contents of accumulator are: 0AH

```
EI                  ; Enable interrupts including INTR
MVI A, 0A           ; Prepare the mask to enable RST 7.5, and 5.5, disable 6.5
SIM                 ; Apply the settings RST masks
```

Determining the Current Mask Settings

- RIM instruction: Read Interrupt Mask
  - Load the accumulator with an 8-bit pattern showing the status of each interrupt pin and mask.



Bit position	D7	D6	D5	D4	D3	D2	D1	D0
Name	SID	I7.5	I6.5	I5.5	IE	M7.5	M6.5	M5.5
Explanation	Serial input data in the SID pin	Set to 1 if RST 7.5 is pending	Set to 1 if RST 6.5 is pending	Set to 1 if RST 5.5 is pending	Set to 1 if interrupts are enabled	Set to 1 if RST 7.5 is masked	Set to 1 if RST 6.5 is masked	Set to 1 if RST 5.5 is masked

The RIM Instruction and the Masks

- Bits 0-2 show the current setting of the mask for each of RST 7.5, RST 6.5 and RST 5.5
    - They return the contents of the three mask flip flops.
    - They can be used by a program to read the mask settings in order to modify only the right mask.
- Bit 3 shows whether the maskable interrupt process is enabled or not.

  It returns the contents of the Interrupt Enable Flip Flop.It can be used by a

  program to determine whether or not interrupts are enabled.

The RIM Instruction and the Masks

- Bits 4-6 show whether or not there are pending interrupts on RST 7.5, RST 6.5, and RST 5.5
    - Bits 4 and 5 return the current value of the RST5.5 and RST6.5 pins.
    - Bit 6 returns the current value of the RST7.5 memory flip flop.
- Bit 7 is used for Serial Data Input.
    - The RIM instruction reads the value of the SID pin on the microprocessor and returns it in this bit.

TRAP

- TRAP is the only non-maskable interrupt.
  - It does not need to be enabled because it cannot be disabled.
- It has the highest priority amongst interrupts.
  - It needs to be high and stay high to be recognized.
  - Once it is recognized, it won’t be recognized again until it goes low, then high again.
- TRAP is usually used for power failure and emergency shutoff.

The 8085 Interrupts

Interrupt Name	Maskable	Masking Method	Vectored	Memory	Triggering Method
INTR	Yes	DI / EI	No	No	Level Sensitive
RST 5.5 / RST 6.5	Yes	DI / EI SIM	Yes	No	Level Sensitive
RST 7.5	Yes	DI / EI SIM	Yes	Yes	Edge Sensitive
TRAP	No	None	Yes	No	Level & Edge Sensitive