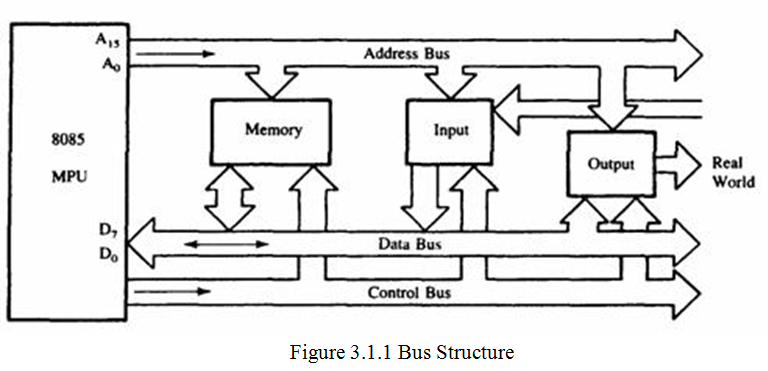
**Chapter 3: Bus Structure, Memory and I/O Interfacing**

**3.1 Bus Structure**

A microcomputer consists of a set of components or modules of three basic types: CPU, memory and I/O units which communicate with each other. A bus is a communication pathway between two or more such components. A bus actually consists of multiple communication pathway or lines. Each line is capable of transmitting signals representing binary 1 and 0. Several lines of the bus can be used to transmit binary data simultaneously. The bus that connects major microcomputer components such as CPU, memory or I/O is called the system bus.



System bus consists of number of separate lines. Each line assigned a particular function. Fundamentally, in any system bus, the lines can be classified into three group buses.

**1. Data Bus**:

Data bus provides the path for monitoring data between the system modules. The bus has various numbers of separate lines like 8, 16, 32, or 64 which referred as the width of data bus .These number represents the no. of bits they can carry because each carry 1 bit. The width of data bus is a key factor in determining the overall system performance.

**2. Address Bus**:

Each Lines of address bus are used to designate the source or destination of the data on data bus. For example, if the CPU requires reading a word (8, 16, 32) bits of data from memory, it puts the address of desired word on address bus. The address bus is also used to address I/O ports. Bus width determines the total memory the up can handle.

**3. Control Bus**:

The control bus is a group of lines used to control the access to control signals and the use of the data and address bus. The control signals transmit both command and timing information between the system modules. The timing signals indicate the validity of data and address information, where as command signals specify operations to be performed. Some of the control signals are:

**Memory Write (MEMW)**: It causes data on the bus to be loaded in to the address location.

**Memory READ (MEMR)**: It causes data from the addressed location to be placed on the data bus.

**I/O Write (IOW)**: It causes the data on the bus to be output to the addressed I/O port.

**I/O Read (IOR)**: It causes the data from the addressed I/O port to be placed on the bus.

**Transfer Acknowledge:** This signal indicates that data have been accepted from or placed on

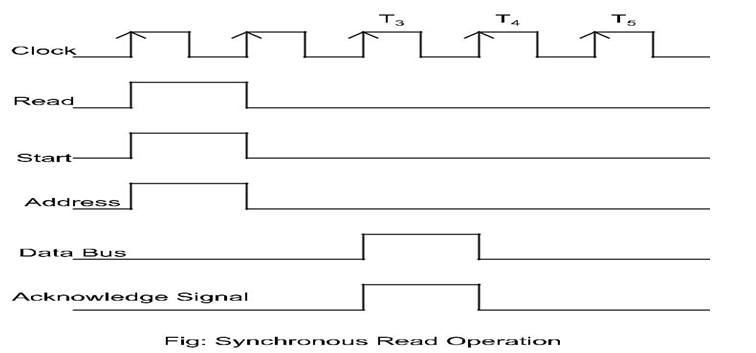
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| the bus. |  |  |  |  |  |  |  |  |  |
| **Bus Request**: It is used | |  | to indicate that a module wants to | | | | gain control of the bus. | | |
| **Bus Grant:** | It indicates | that a requesting module has been granted for the control of bus. | | | | | | | |
| **Interrupt** | **Request**: | It | indicates | that | an | interrupt | has | been | pending. |

**Interrupt Acknowledge:** it indicates that the pending interrupt has been recognized.

**3.1.1 Synchronous and Asynchronous Bus**

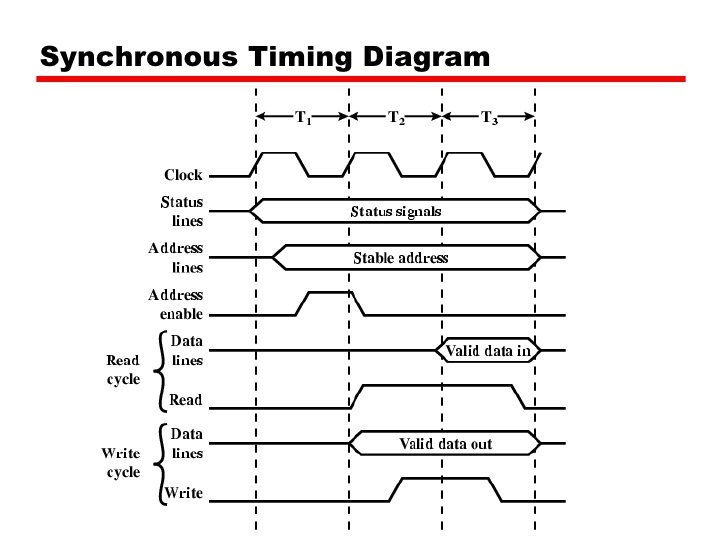
**3.1.1.1 Synchronous Bus**

In a synchronous bus, the occurrence of the events on the bus is determined by a clock. The clock transmits a regular sequence of 0’s & 1’s of equal duration. All the events start at beginning of the clock cycle.

****

* Here the CPU issues a START signal to indicate the presence of address and control information on the bus.
* Then it issues the memory read signal and places the memory address on the address bus.
* The addressed memory module recognizes the address and after a delay of one clock cycle it places the data and acknowledgment signal on the buses.

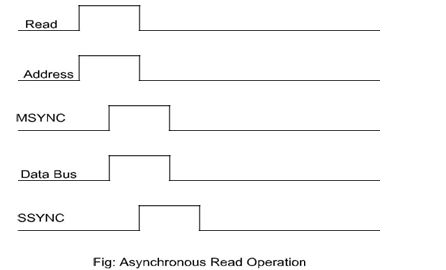
In synchronous bus, all devices are tied to a fixed rate, and hence the system cannot take advantage of device performance but it is easy to implement.



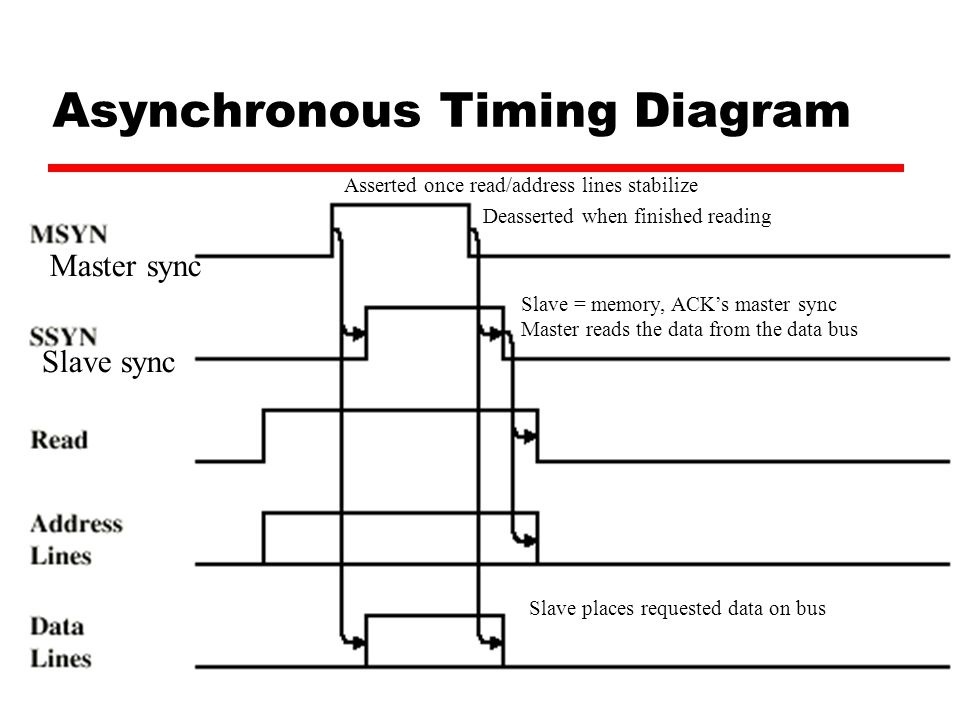
* Transmitter and receivers are synchronized of clock.
* Data bits are transmitted with synchronization of clock.
* Character is received at constant Rate.
* Data transfer takes place in block.
* Start and stop bit are required to establish communication of each character.
* Used in high – speed transmission.

**3.1.1.2 Asynchronous Bus**

In an asynchronous bus, the timing is maintained in such way that occurrence of one event on the bus follows and depends on the occurrence of previous event.

****

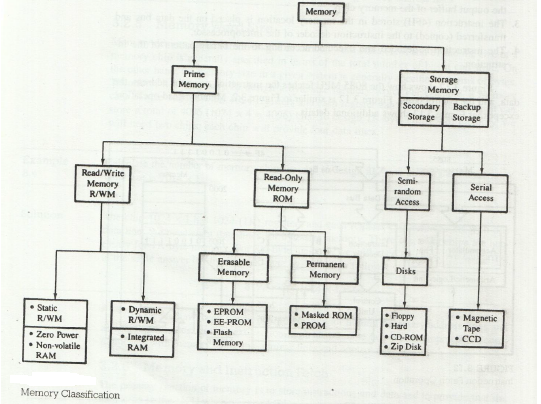
* Here the CPU places Memory Read (Control) and address signals on the bus.
* Then it issues master synchronous signal (MSYNC) to indicate the presence of valid address and control signals on the bus.
* The addressed memory module responds with the data and the slave synchronous signal (SSYNC)



* Transmitters and receivers are not synchronized by clock.
* Bits of data are transmitted at constant rate.
* Character may arrive at any rate at receiver.
* Data transfer is character oriented.
* Start and stop bits are required to establish communication of each character.
* Used in low – speed transmission.

**3.2 Memory Interfacing**

**3.2.1 Memory and its type**

****

Memory is an essential component of the microcomputer system. It is used to store both instructions and data. It is used to store both instructions and data. Memory is made up of registers and the number of bits stored in a register is called memory word .Memory word is identified by an address .If microprocessor uses 16 bit address , then there will be maximum of 216= 65536 memory addresses ranging from 0000H to FFFFH.

There are various types of memory which can be classified in to two main groups i.e. Primary memory and Secondary memory.

**1. Primary Memory**

It is the memory used by microprocessor to execute programs. The microprocessor can access only those items that are stored in this memory. Hence, all data and program must be within primary memory prior to its execution. Primary memory is much larger than processer memory that is included in the microprocessor chip.

Primary memory is divided into two groups.

1. **R/W Memory (RAM)**

Microprocessor can read for and write into this memory .This memory is used for information that is likely to be altered such as writing program or receiving data. This memory is volatile i.e. the content will be lost if the power is turned off and commonly known as RAM, RAM are basically of two types.

1. **Static RAM (SRAM)**

This memory is made up of flip flops and it stores bit as voltage. A single flip flop stores binary data either 1 or 0. Each flip flop is called storage cell. Each cell requires six transistors. Therefore, the memory chip has low density but high speed. This memory is more expensive and consumes more power.

1. **Dynamic RAM (DRAM)**

This memory is made up of MOS transistor gates and it stores the bit as charge. The advantage of DRAM are it has high density, low power consumption and cheaper than SRAM. But the bit information leaks therefore needs to be rewritten again every few milliseconds. It is called refreshing the memory and requires extra circuitry to do this. It is slower than SRAM.

1. **Read Only Memory (ROM)**

ROM contains a permanent pattern of data that cannot be changed. It is non volatile that is no power source is required to maintain the bit values in memory. ROM is basically of 5 types.

1. **Masked ROM**

A bit pattern is permanently recorded by the manufactures during production.

1. **Programmable ROM**

In this ROM, a bit pattern may be written into only once and the writing process is performed electrically. That may be performed by a supplier or customer.

1. **Erasable PROM (EPROM)**

This memory stores a bit in the form of charge by using EPROM programmer which applies high voltage to charge the gate .Information can be erased by exposing ultra violet radiation. It is reusable. The disadvantages are :(i) it must be taken out off circuit to erase it (ii). The entire chip must be erased (iii) the erasing process takes 15 to 20 minutes.

1. **Electrically Erasable Electrically Erasable PROM (EEPROM)**

It is functionally same as EPROM except that information can be altered by using electrical signal at the register level rather than erasing all the information. It is expensive compared to EPROM and flash and can be erased in 10 ms.

1. **Flash Memory**

It is variation of EPROM. The difference is that EPROM can be erased in register level but flash memory must be erased in register level but flash memory must be erased in its entirety or at block level.

**2. Secondary Memory**

The devices that provide backup storage are called secondary memory. It includes serial access type such as magnetic disks and random access type such as magnetic disks. It is nonvolatile memory.

**3.2.2 Performance of memory**

**1.** **Access time (ta)**

**Read access time:** It is the average time required to read the unit of information from memory.

**Write access time:** It is the average time required to write the unit of information on memory.

Access rate (ra) =/ta

**2.** **Cycle time (tc)**

It is the average time that lapses between two successive read operation.

Cycle rate (rc) = bandwidth = 1/tc

**3.2.3 Access modes of memory:**

**1. Random access:** In random access mode, ta is independent of the location from which the data is accessed like MOS memory.

**2. Sequential access:** In that mode, ta is dependent of the location form which the data is accessed like magnetic type.

**3. Semi random access:** The semi random access combines these two. For example, in magnetic disk, any track can be accessed at random. But the access within the truck must be in serial fashion.

**3.2.4 The Memory Hierarchy**

* Capacity, cost and speed of different types of memory play a vital role while designing a memory system for computers.
* If the memory has larger capacity, more application will get space to run smoothly.
* It's better to have fastest memory as far as possible to achieve a greater performance. Moreover for the practical system, the cost should be reasonable.
* There is a tradeoff between these three characteristics cost, capacity and access time. One cannot achieve all these quantities in same memory module because
  + - If capacity increases, access time increases (slower) and due to which cost per bit decreases.
    - If access time decreases (faster), capacity decreases and due to which cost per bit increases.
* The designer tries to increase capacity because cost per bit decreases and the more application program can be accommodated. But at the same time, access time increases and hence decreases the performance.

**So the best idea will be to use memory hierarchy.**

* Memory Hierarchy is to obtain the highest possible access speed while minimizing the total cost of the memory system.
* Not all accumulated information is needed by the CPU at the same time.
* Therefore, it is more economical to use low-cost storage devices to serve as a backup for storing the information that is not currently used by CPU.
* The memory unit that directly communicates with CPU is called the *main memory.*
* Devices that provide backup storage are called *auxiliary memory.*
* The memory hierarchy system consists of all storage devices employed in a computer system from the slow by high-capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory.
* The main memory occupies a central position by being able to communicate directly with the CPU and with auxiliary memory devices through an I/O processor.
* A special very-high-speed memory called **cache** is used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate.
* CPU logic is usually faster than main memory access time, with the result that processing speed is limited primarily by the speed of main memory.
* The cache is used for storing segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations.
* The memory hierarchy system consists of all storage devices employed in a computer system from slow but high capacity auxiliary memory to a relatively faster cache memory accessible to high speed processing logic. The figure below illustrates memory hierarchy.

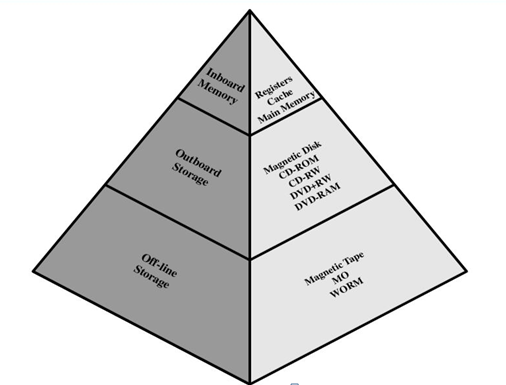


Figure: Memory Hierarchy

As we go down in the hierarchy,

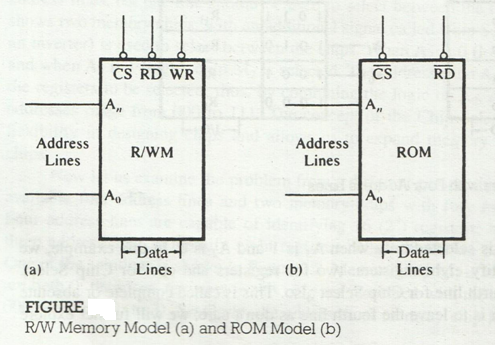
* Cost per bit decreases
* Capacity of memory increases
* Access time increases
* Frequency of access of memory by processor also decreases.

**3.2.4.1 Hierarchy List**

* Registers
* L1 Cache
* L2 Cache
* Main memory
* Disk cache
* Disk
* Optical
* Tape

|  |  |  |
| --- | --- | --- |
| **SRAM Vs DRAM** | | |
| 1 | Uses 6-transistors | DRAM uses only 1-transistor |
| 2 | Value stored as voltage | Value stored as charge |
| 3 | High power consumption | Low power consumption |
| 4 | Expensive | Cheaper |
| 5 | Faster | Slower |
| 6 | Fabrication density low | Higher fabrication density |
| 7 | Additional refreshingcircuitry not needed | Additional refreshingcircuitry needed |

3.2.5 R/W Memory Model



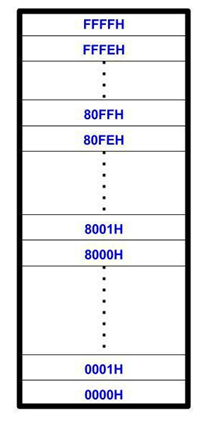
3.2.5.1 Memory Map and Addresses

**8085:**

* **8**-bit processor
* **16**-bit address lines
* Can identify **216** = **65,536** memory registers each with **16**-bit address
* The entire memory addresses can range from **0000H** to **FFFFH**

**Memory Map**

A memory map is a pictorial representation in which memory devices are located in the entire range of addresses.



3.2.6 Address Decoding

Microprocessor is connected with memory and I/O devices via common address and data bus. Only one device can send data at a time and other devices can only receive that data. If more than one device sends data at the same time, the data gets garbled. In order to avoid this situation, ensuring that the proper device gets addressed at proper time, the technique called address decoding is used.

In address decoding method, all devices like memory blocks, I/O units etc. are assigned with a specific address. The address of the device is determined from the way in which the addresses lines are used to derive a special device selection signal, chip select (CS). If the microprocessor has to write or to read from a device, the CS signal to that block should be enabled and the address decoding circuit must ensure that CS signal to other devices are not activated.

Depending upon the number of address lines used to generate chip select signal for the device, the address decoding is classified as:

* **I/O with 8-bit addresses (Peripheral-Mapped I/O)**
  + Microprocessor uses eight address lines to identify an input or output devices.
  + It is also known as I/O-mapped I/O.
  + I/O space is separate from memory space.
  + The eight address lines can have 256 addresses, thus, the microprocessor can identify 256 input devices and 256 output devices with addresses ranging from 00H to FFH.
  + The input and output devices are differentiated by the control signal.
  + The entire range of I/O addresses from 00H to FFH is known as an I/O map.
  + And individual addresses are referred to as I/O device addresses, or I/O port numbers.

In this method, a device is identified with an 8 bit address and operated by I/O related functions IN and OUT for that IO/M =1. Since only 8bit address is used, at most 256 bytes can be identified uniquely. Generally low order address bits A0-A7 are used and upper bits A8-A15 are considered don’t care. Usually I/O mapped I/O is used to map devices like 8255A, 8251A etc.

* **I/O with 16-bit addresses (Memory-Mapped I/O)**
  + Microprocessor uses 16 address lines to identify an input or output devices.
  + I/O devices are connected as if it is a memory register.
  + It is also known as Memory-mapped I/O.
  + The microprocessor uses the same control signal (memory read or write) and instructions as those of memory.
  + The 16 address lines can have 65,536 addresses, thus, the microprocessor can identify 65,536 input devices and 65,536 output devices with addresses ranging from 0000H to FFFFH.

In this method , a device is identified with 16 bit address and enabled memory related functions such as STA , LDA for which IO/M ~~=~~0, here chip select signal of each device is derived from 16 bit address lines thus total addressing capability is 64K bytes . Usually memory mapped I/O is used to map memories like RAM, ROM etc.

|  |  |  |
| --- | --- | --- |
| **S.N.** | **I/O mapped I/O Vs** | **Memory Mapped I/O** |
| 1 | In this device, address is 8-bit. | In this device, address is 16-bit. |
| 2 | IOR and IOW control signals are used to control read and write I/O operations. | MEMR and MEMW control signals are used to read and write I/O operations. |
| 3 | Instructions available are IN and OUT. | Instructions available are LDA, STA, MOV etc. |
| 4 | Data transfer is between accumulator and I/O device | Data transfer is between any register and I/O device |
| 5 | Maximum numbers of I/O devices are 256. | Maximum numbers of I/O devices are 65,536 theoretically. |

Depending on the addresses that are allocated to the device, the address decoding is categorized in the following two groups:

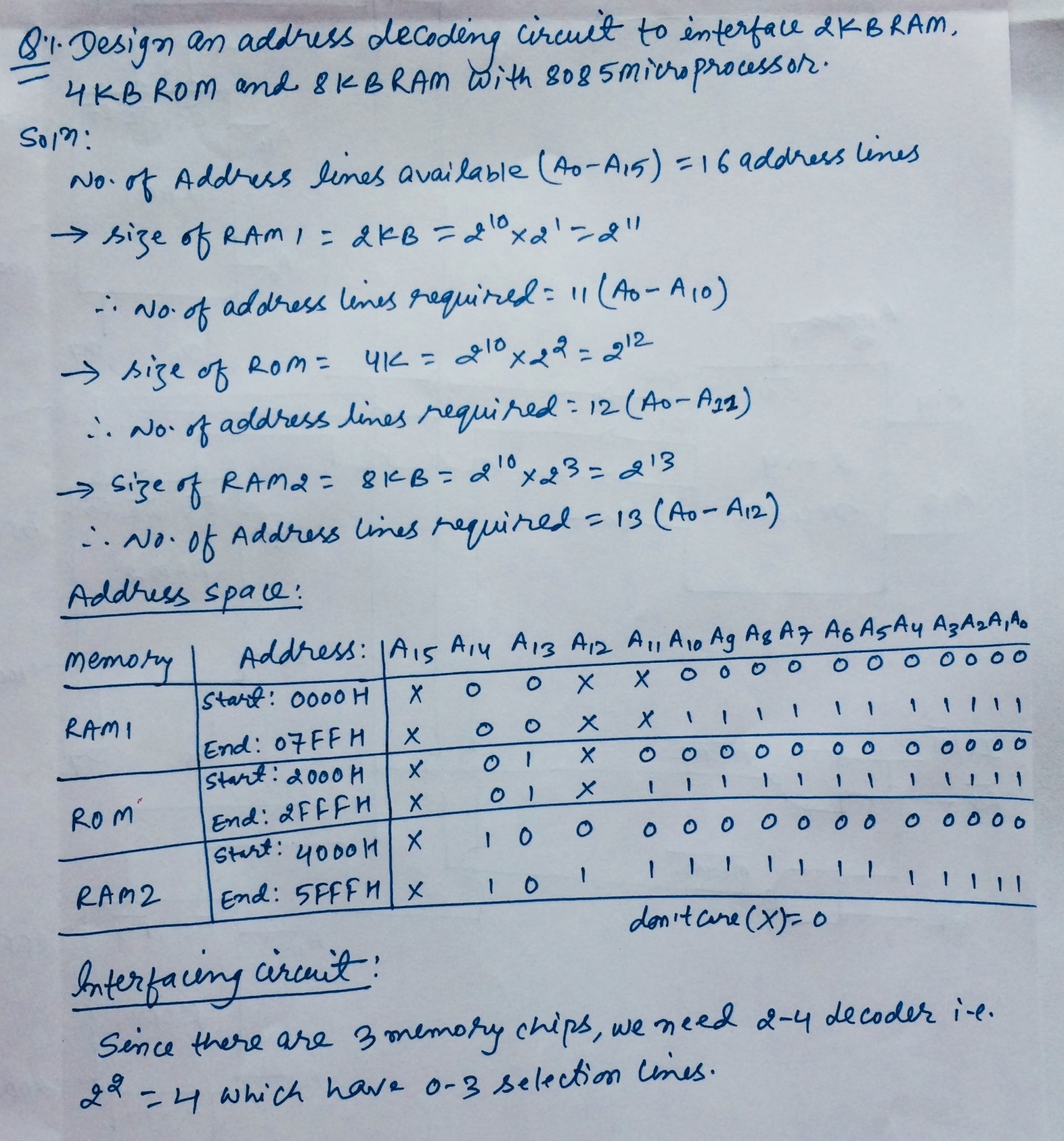
* **Unique Address Decoding**

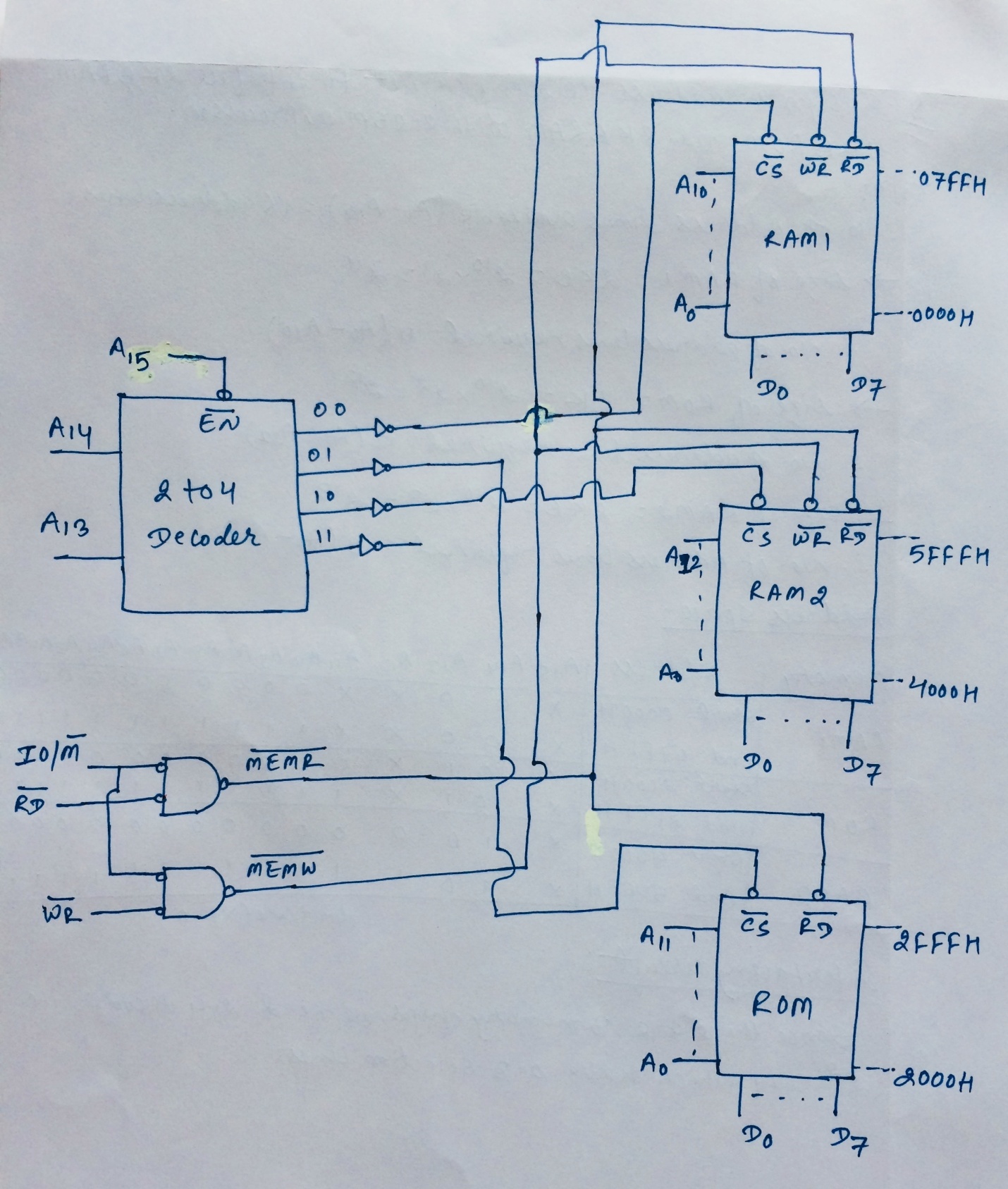
If all the address lines on that mapping mode are used for address decoding then that decoding is called unique address decoding. It means all 8-lines in I/O mapped I/O andall 16 lines in memory mapped I/O are used to derive **** signal. It is expensive and complicated but fault proof in all cases.

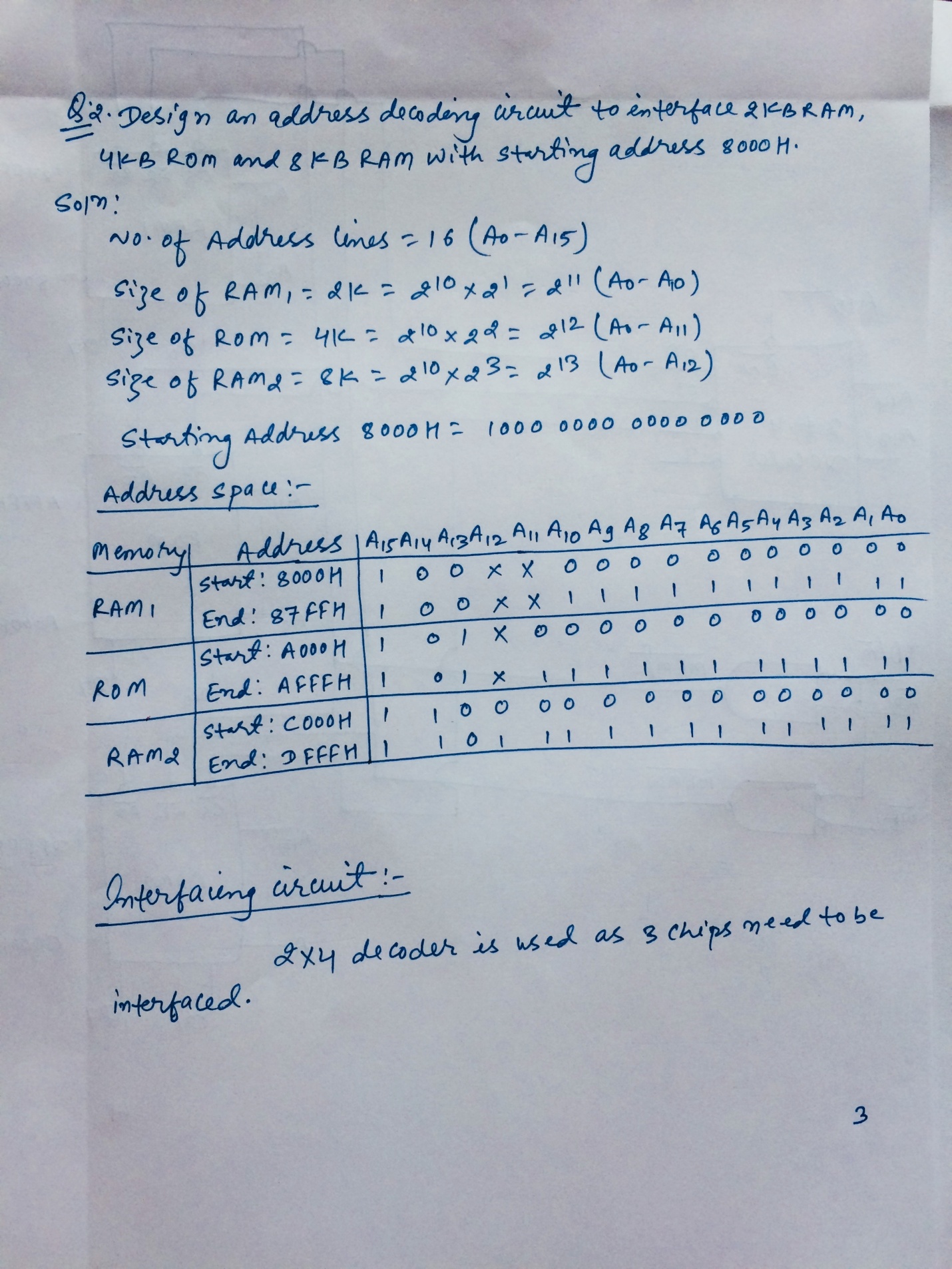
* **Non Unique Address decoding**

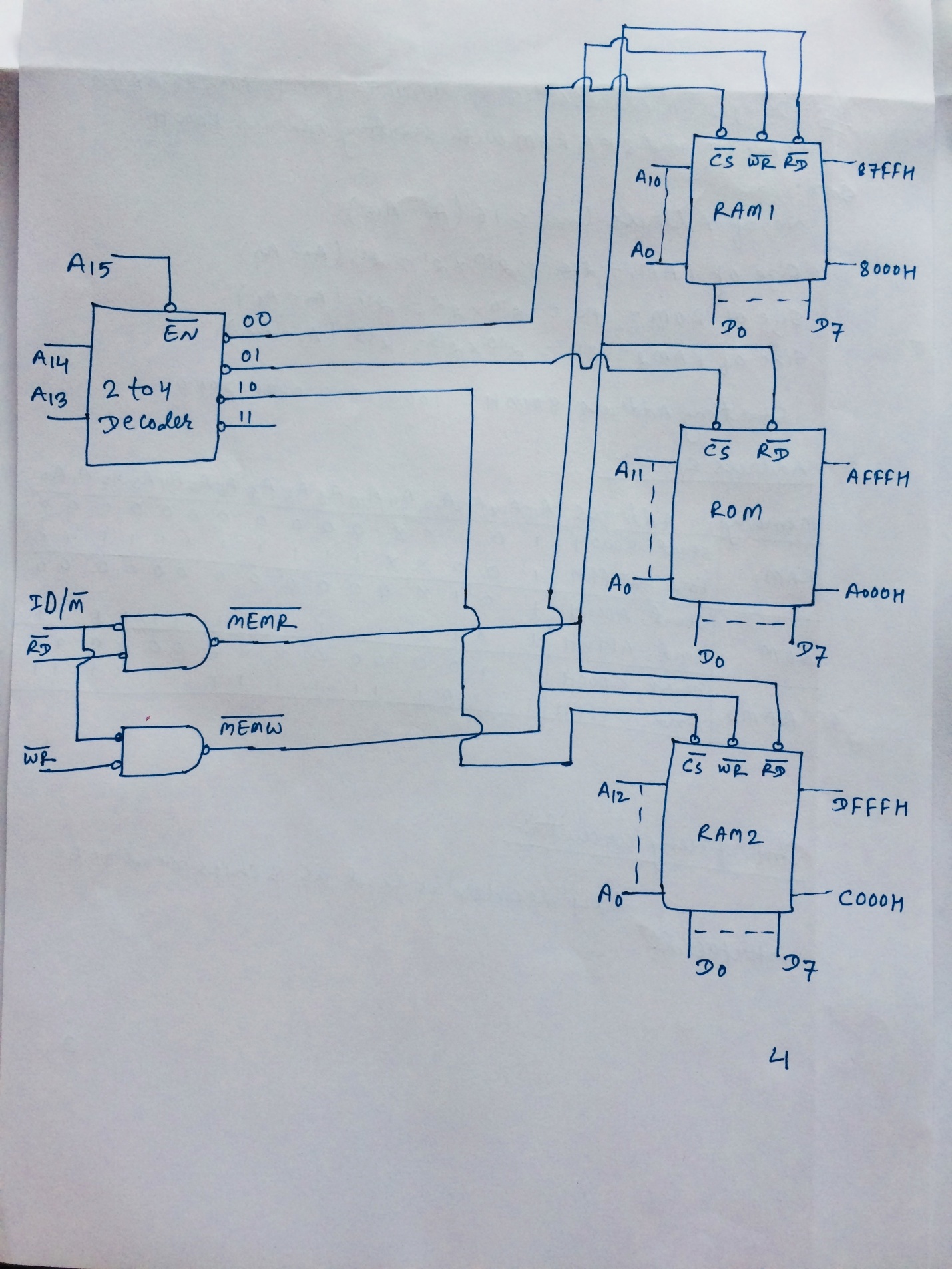
If all the address lines available on that mode are not used in address decoding then that decoding is called non unique address decoding. Though it is cheaper there may be a chance of address conflict.

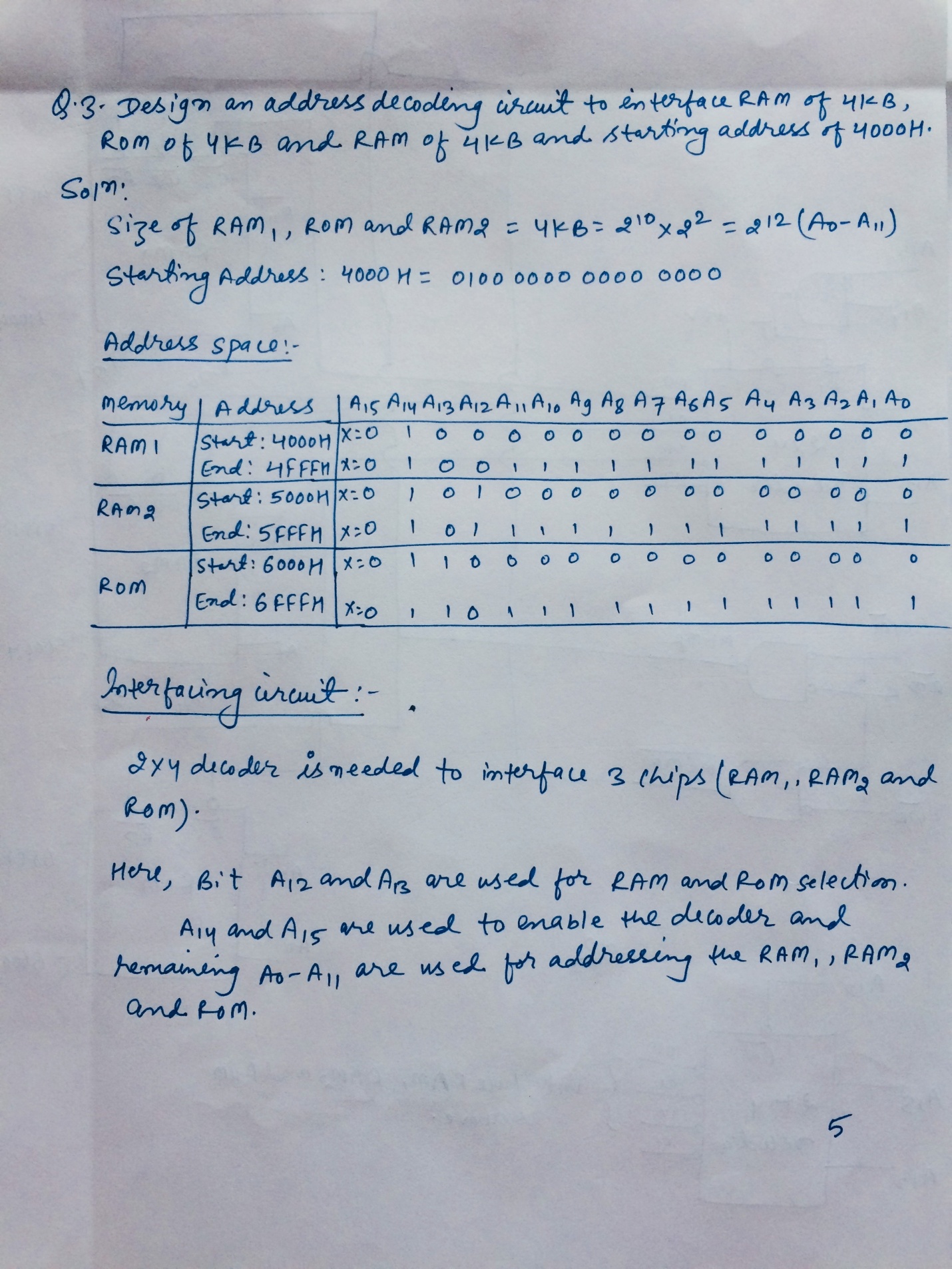
Some examples:

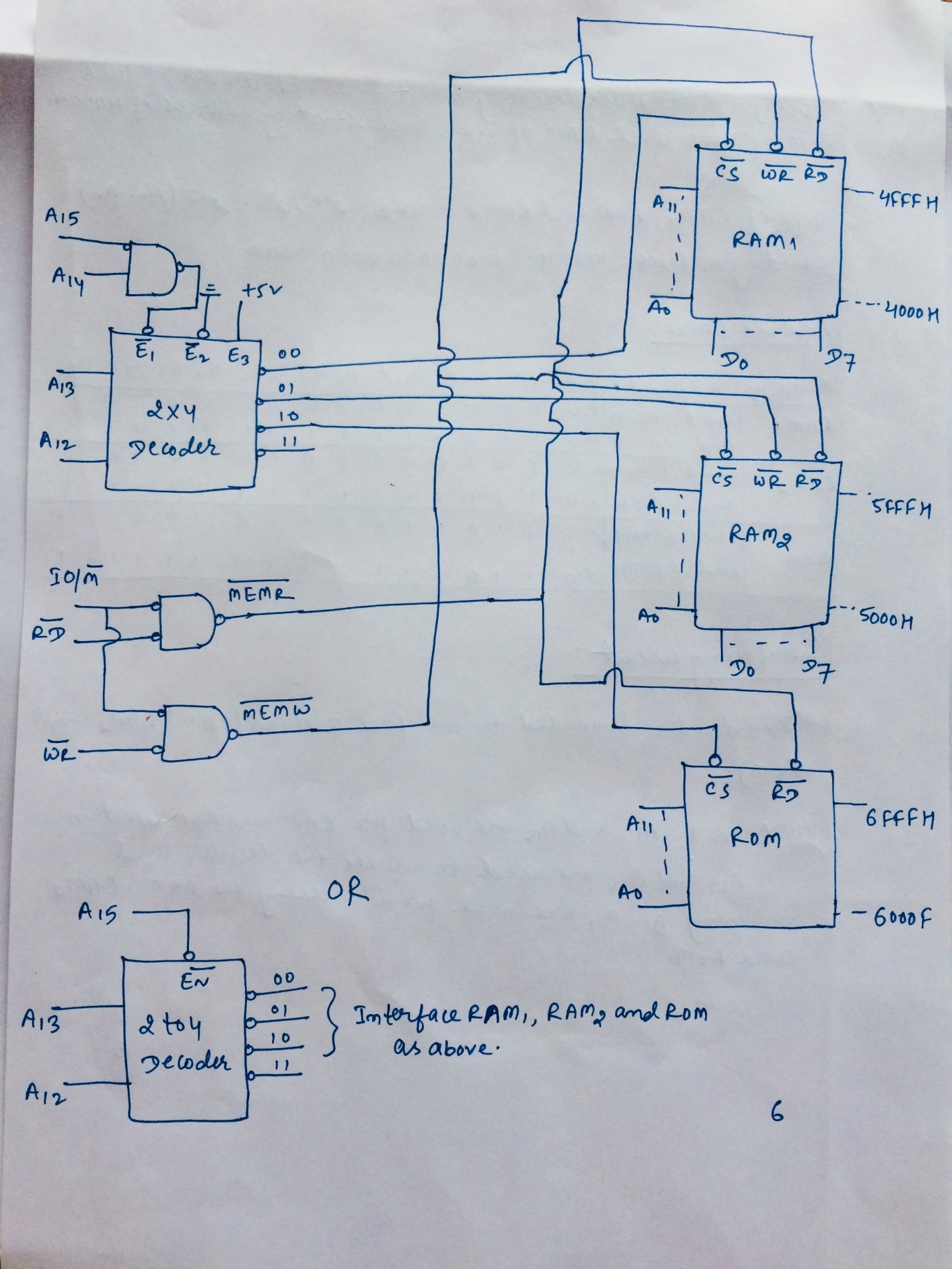


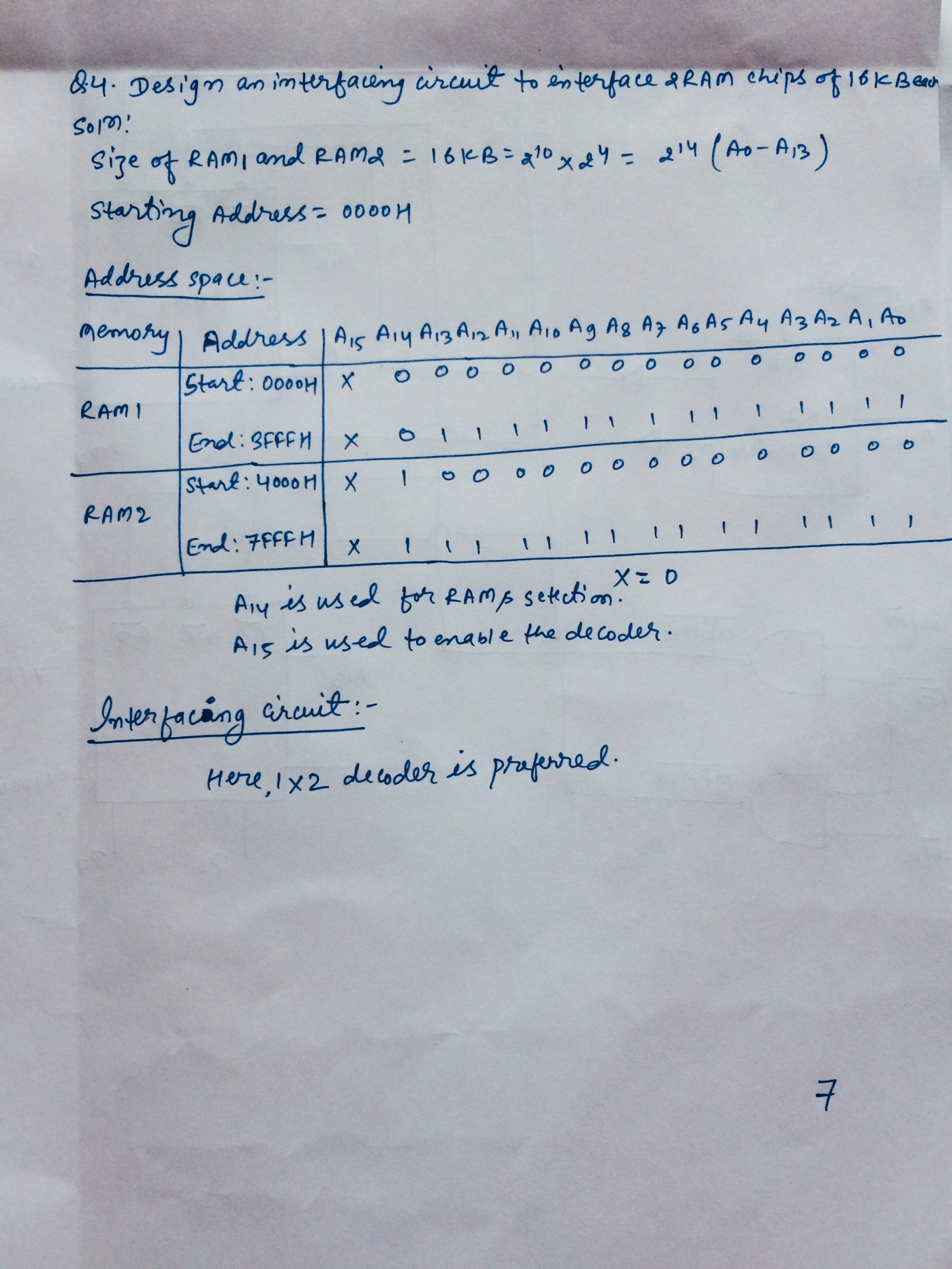


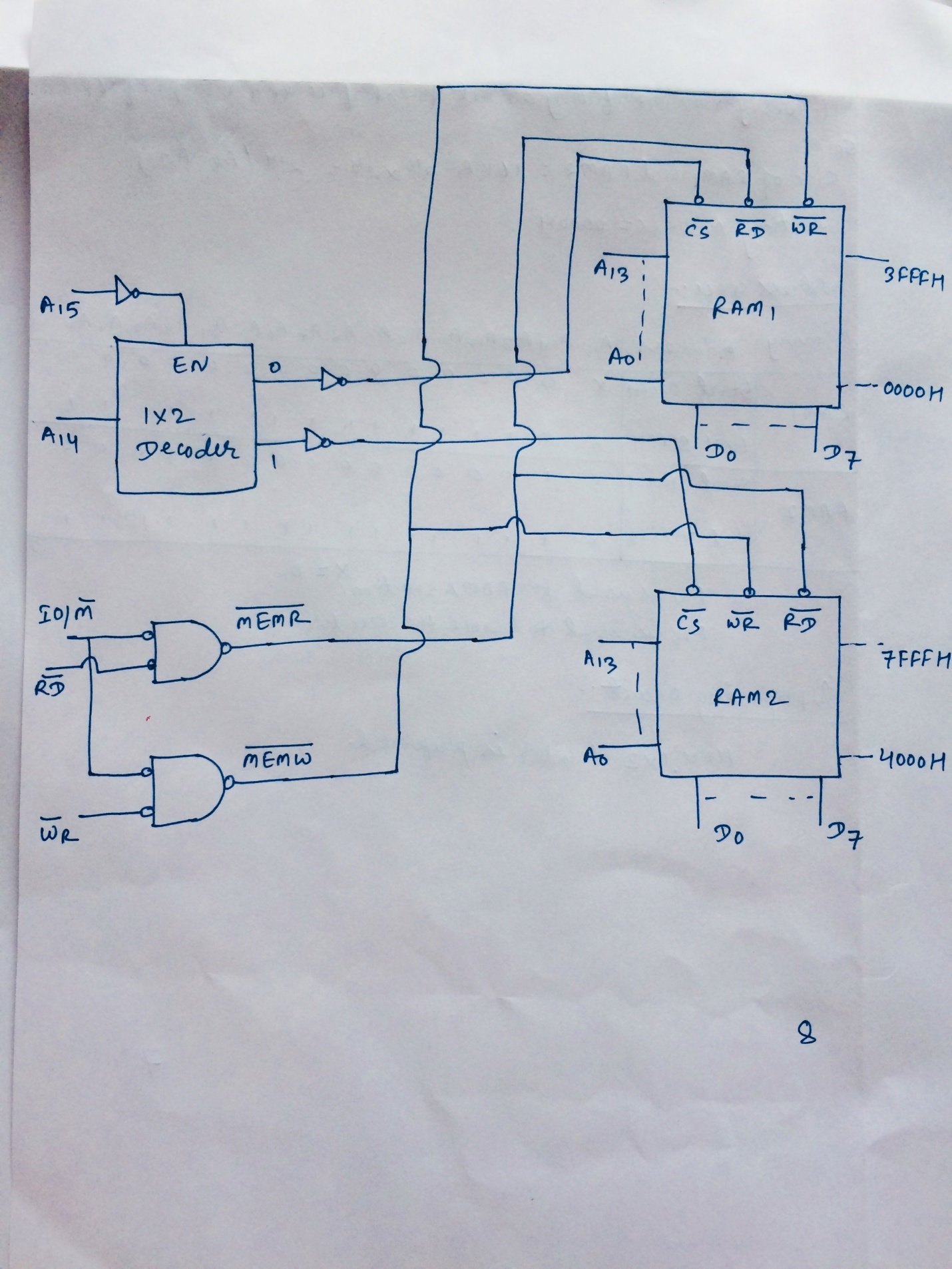












**3.3 I/O Interfacing**

**Concepts of Interrupts**

* Interrupt is signals send by an external device to the processor to request the processor to perform a particular task or work.
* Mainly in the microprocessor based system, the interrupts are used for data transfer between the peripheral and the microprocessor.
* The processor will check the interrupts always at the 2nd T-state of last machine cycle.
* If there is any interrupt, it accepts the interrupt and sends the INTA (active low) signal to the peripheral.
* The vectored address of particular interrupt is stored in program counter.
* The processor executes an interrupt service routine (ISR) addressed in program counter.
* It returned to main program by RET instruction.

**Need for Interrupt**

Interrupts are particularly useful when interfacing I/O devices that provideor require data at relatively low data transfer rate.

**Interrupt Operations**

The transfer of data between the microprocessor and input /output devices takes place using various modes of operations like programmed I/O, interrupt I/O and direct memory access. In programmed I/O, the processor has to wait for a long time until I/O module is ready for operation. So the performance of entire system degraded. To remove this problem CPU can issue an I/O command to the I/O module and then go to do some useful works. The I/O device will then interrupt the CPU to request service when it is ready to exchange data with CPU. In response to an interrupt, the microprocessor stops executing its current program and calls a procedure which services the interrupt.

The interrupt is a process of data transfer whereby an external device or a peripheral can inform the processor that it is ready for communication and it requests attention. The response to an interrupt request is directed or controlled by the microprocessor.

**Process of interrupt Operation**

**From the point of view of** I/O **unit**

* I/O device receives command from CPU.
* The I/O device then processes the operation.
* The I/O device signals an interrupt to the CPU over a control line.
* The I/O device waits until the request from CPU.

**From the point of view of processor**

* The CPU issues command and then goes off to do its work.
* When the interrupt from I/O device occurs, the processor saves its program counter & registers of the current program and processes the interrupt.
* After completion for interrupt, processor requires its initial task.

**Polling versus Interrupt**

* Each time the device is given a command, for example *``move the read head to sector 42* *of the floppy disk''* the device driver has a choice as to how it finds out that the commandhas completed. The device drivers can either poll the device or they can use interrupts.
* Polling the device usually means reading its status register every so often until the device's status changes to indicate that it has completed the request.
* Polling means the CPU keeps checking a flag to indicate if something happens.
* An interrupt driven device driver is one where the hardware device being controlled will cause a hardware interrupt to occur whenever it needs to be serviced.
* With interrupt, CPU is free to do other things, and when something happens, an interrupt is generated to notify the CPU. So it means the CPU does not need to check the flag.
* Polling is like picking up your phone every few seconds to see if you have a call. Interrupts are like waiting for the phone to ring.
* Interrupts win if processor has other work to do and event response time is not critical.
* Polling can be better if processor has to respond to an event ASAP; may be used in device controller that contains dedicated secondary processor.

**Advantages of interrupt over Polling**

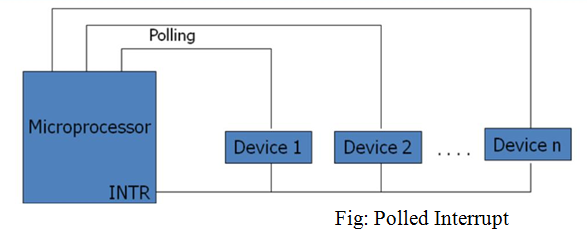
* Interrupts are used when you need the fastest response to an event. For example, you need to generate a series of pulses using a timer. The timer generates an interrupt when it overflows and within 1 or 2 sec, the interrupt service routine is called to generate the pulse. If polling were used, the delay would depend on how often the polling is done and could delay response to several msecs. This is thousands times slower.
* Interrupts are used to save power consumption. In many battery powered applications, the microcontroller is put to sleep by stopping all the clocks and reducing power consumption to a few micro amps. Interrupts will awaken the controller from sleep to consume power only when needed. Applications of this are hand held devices such as TV/VCR remote controllers.
* Interrupts can be a far more efficient way to code. Interrupts are used for program debugging.

**Interrupt structures**

A processor is usually provided with one or more interrupt pins on the chip. Therefore, a special mechanism is necessary to handle interrupts from several devices that share one of these interrupt lines. There are mainly two ways of servicing multiple interrupts which are polled interrupts and daisy chain (vectored) interrupts.

**1. Polled interrupts**

Polled interrupts are handled by using software which is slower than hardware interrupts. Here the processor has the general (common) interrupt service routine (ISR) for all devices. The priority of the devices is determined by the order in which the routine polls each device. The processor checks the starting with the highest priority device. Once it determines the source of the interrupt, it branches to the service routine for that device.

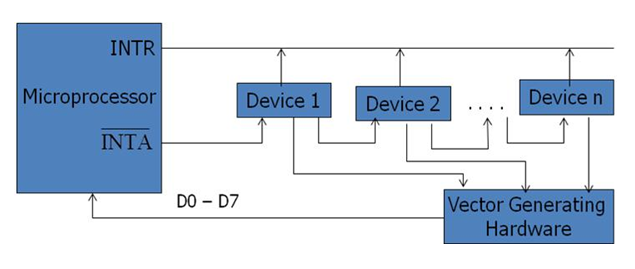
****

Here several eternal devices are connected to a single interrupt line (INTR) of the microprocessor. When INTR signal goes up, the processor saves the contents of PC and other registers and then branches to an address defined by the manufactures of the processor. The user can write a program at this address to find the source of the interrupt by starting the polled from highest priority device.

**2. Daisy chain (vectored) interrupt**

In polled interrupt, the time required to poll each device may exceed the time to service the device through software. To improve this, the faster mechanism called vectored or daisy chain interrupt is used. Here the devices are connected in chain fashion. When INTR pin goes up, the processor saves its current status and then generates INTA signal to the highest priority device. If this device has generated the interrupt, it will accept the INTA; otherwise it will push INTA to the next priority device until the INTA is accepted by the interrupting device.

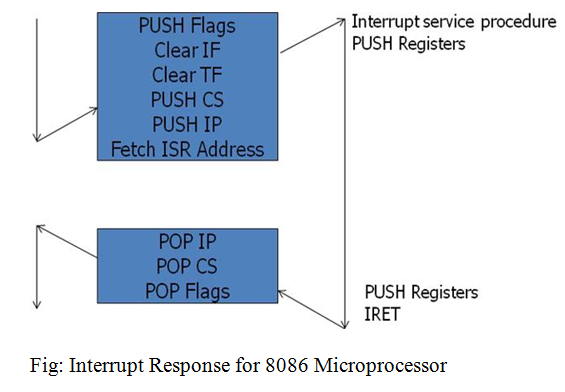
When INTA is accepted, the device provides a means to the processor for findings the interrupt address vector using external hardware. The accepted device responds by placing a word on the data lines which becomes the vector address with the help of any hardware through which the processor points to appropriate device service routine. Here no general interrupt service routine need first that means appropriate ISR of the device will be called.

Fig: Vectored (Daisy Chain) Interrupt

**Interrupt Processing Sequence**

The occurrence of interrupt triggers a number of events, both in processor hardware and in software. The interrupt driven I/O operation takes the following steps.

* The I/O unit issues an interrupt signal to the processor for exchange of data between them.
* The processor finishes execution of the current instruction before responding to the interrupt.
* The processor sends an acknowledgement signal to the device that it issued the interrupt.
* The processor transfers its control to the requested routine called “Interrupt Service Routine (ISR)” by saving the contents of program status word (PSW) and program counter (PC).
* The processor now loads the PC with the location of interrupt service routine and the fetches the instructions. The result is transferred to the interrupt handler program.
* When interrupt processing is completed, the saved register’s value are retrieved from the stack and restored to the register.
* Finally it restores the PSW and PC values from the stack.



The figure summarizes these steps. The processor pushes the flag register on the stack, disables the INTR input and does essentially an indirect call to the interrupt service procedure. An IRET function at the end of interrupt service procedure returns execution to the main program.

**Interrupt priority**

Microcomputers can transfer data to or from an external devices using interrupt through INTR pin. When device wants to communicate with the microcomputer, it connects to INTR pin and makes it high or low depending on microcomputer. The microcomputer responds by sending signal via its pin called interrupt acknowledgement INTA. In differentiation with the occurrence of interrupts, basically following interrupts exist.

**1. External interrupts**

These interrupts are initiated by external devices such as A/D converters and classified on following types.

* Maskable interrupt

It can be enabled or disabled by executing instructions such as EI and DI. In 8085, EI sets the interrupt enable flip flop and enables the interrupt process. DI resets the interrupt enable flip flop and disables the interrupt.

* Non-maskable interrupt

It has higher priority over maskable interrupt and cannot be enabled or disabled by the instructions.

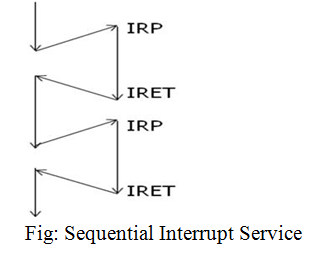
**2. Internal interrupts**

* These are indicated internally by exceptional conditions such as overflow, divide by zero, and execution of illegal op-code. The user usually writes a service routine to take correction measures and to provide an indication in order to inform the user that exceptional condition has occurred.
* There can also be activated by execution of TRAP instruction. This interrupt means TRAP is useful for operating the microprocessor in single step mode and hence important in debugging.
* These interrupts are used by using software to call the function of an operating system. Software interrupts are shorter than subroutine calls and they do not need the calling program to know the operating system’s address in memory.

If the processor gets multiple interrupts, then we need to deal these interrupts one at a time and the dealing approaches are:

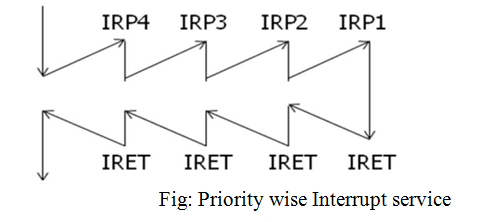
**a. Sequential processing of interrupts**

When user program is executing and an interrupt occurs, interrupts are disabled immediately. After the interrupt service routine completes, interrupts are enabled before resuming the user program and the processor checks to see if additional interrupts have occurred.



**b. Priority wise processing of interrupts**

The drawback of sequential processing is that it does not take account of relative priority or time critical needs. The alternative form of this is to define priorities for interrupts and to allow an interrupt of higher priority to cause a lower priority interrupts pause until high priority interrupt completes its function.



**Interrupt Service Routine**

* An interrupt service routine (ISR) is a software routine that hardware invokes in response to an interrupt.
* ISRs examine an interrupt and determine how to handle it.
* ISRs handle the interrupt, and then return a logical interrupt value.
* Its central purpose is to process the interrupt and then return control to the main program.
* An ISR must perform very fast to avoid slowing down the operation of the device and the operation of all lower priority ISRs.
* As in procedures, the last instruction in an ISR should be IRET.

ISR is responsible for doing the following things:

1. Saving the processor context

Because the ISR and main program use the same processor registers, it is the responsibility of the ISR to save the processor’s registers before beginning any processing of the interrupt. The processor context consists of the instruction pointer, registers, and any flags. Some processors perform this step automatically.

2. Acknowledging the interrupt

The ISR must clear the existing interrupt, which is done either in the peripheral that generated the interrupt, in the interrupt controller, or both.

3. Restoring the processor context

After interrupt processing, in order to resume the main program, the values that were saved prior to the ISR execution must be restored. Some processors perform this step automatically.

**Interrupt Processing in 8085**

* Interrupt is signals send by an external device to the processor, to request the processor to perform a particular task or work.
* Mainly in the microprocessor based system the interrupts are used for data transfer between the peripheral and the microprocessor.
* The processor will check the interrupts always at the 2nd T-state of last machine cycle.
* If there is any interrupt it accept the interrupt and send the INTA (active low) signal to the peripheral.
* The vectored address of particular interrupt is stored in program counter.
* The processor executes an interrupt service routine (ISR) addressed in program counter.
* It returned to main program by RET instruction.

**Types of Interrupts**

It supports two types of interrupts.

1. Hardware
2. Software

**Software interrupts**

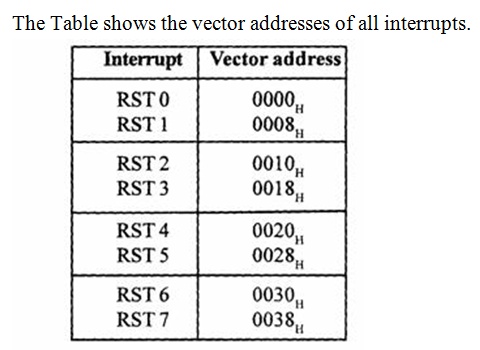
The software interrupts are program instructions. These instructions are inserted at desired locations in a program.

The 8085 has eight software interrupts from RST 0 to RST 7. The vector address for these interrupts can be calculated as follows.

Interrupt number \* 8 = vector address

For RST 5; 5 \* 8 = 40 = 28H

Vector address for interrupt RST 5 is 0028H

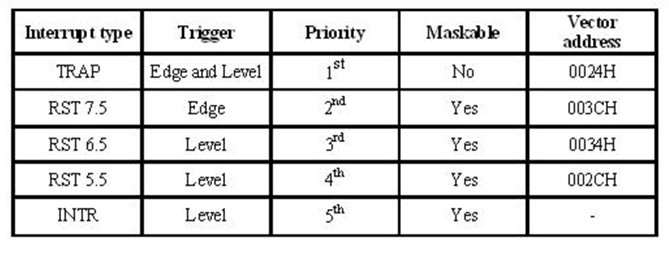


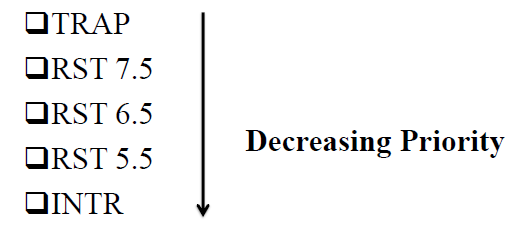
**Interrupt Pins and Priorities (Hardware interrupts)**

An external device initiates the hardware interrupts and placing an appropriate signal at the interrupt pin of the processor.

If the interrupt is accepted then the processor executes an interrupt service routine. The 8085 has five hardware interrupts:

(1) TRAP (2) RST 7.5 (3) RST 6.5 (4) RST 5.5 (5) INTR





On the basis of enabling and disabling the interrupts, there are two types of interrupts namely

1. Maskable Interrupts and
2. Non-maskable Interrupts

**Maskable Interrupts**

* + The interrupts which can be blocked or delayed by using instructions are called maskable interrupts.
  + In 8085, the RESET interrupts (RST 5.5, RST 6.5 and RST 7.5) and INTR are maskable interrupts. They can be enabled/ disabled by using instructions EI/DI.
  + In 8086, INTR is maskable interrupt. It can be enabled /disabled by using instructions STI/CLI.

**Non-Maskable Interrupts**

* Those interrupts which cannot be blocked by instructions are termed as non-maskable interrupts.
* In 8085, TRAP is only non-maskable interrupt and it is used for power failure and emergency cutoff.
* In 8086, NMI is non-maskable interrupt.

**Vectored Interrupts**

* The interrupts for which address of ISR is already known to MP are called vectored interrupts.
* In 8085, RESET interrupts (RST 5.5, RST 6.5 and RST 7.5) and TRAP are vectored interrupts.

|  |  |
| --- | --- |
| **Interrupt** | **Vector Address (Hex)** |
| RST 5.5 | 002C |
| RST 6.5 | 0034 |
| RST 7.5 | 003C |
| TRAP | 0024 |

**Non-Vectored Interrupt**

* In non-vectored interrupts, the interrupting device needs to supply the address of the ISR to the microprocessor.
* In 8085, INTR is non-vectored interrupt.

**TRAP**

* This interrupt is a non-maskable interrupt. It is unaffected by any mask or interrupt enable.
* TRAP bas the highest priority and vectored interrupt.
* TRAP interrupt is edge and level triggered. This means that the TRAP must go high and remain high until it is acknowledged.
* In sudden power failure, it executes a ISR and send the data from main memory to backup memory.
* The signal, which overrides the TRAP, is HOLD signal. (i.e.,if the processor receives HOLD and TRAP at the same time then HOLD is recognized first and then TRAP is recognized).
* There are two ways to clear TRAP interrupt.
  + By resetting microprocessor (External signal)
  + By giving a high TRAP ACKNOWLEDGE (Internal signal)

**RST 7.5:**

* The RST 7.5 interrupt is a maskable interrupt.
* It has the second highest priority.
* It is edge sensitive ie. Input goes to high and no need to maintain high state until it recognized.
* Maskable interrupt. It is disabled by
  + DI instruction
  + System or processor reset.
  + After reorganization of interrupt.
* Enabled by EI instruction.

**RST 6.5 and 5.5:**

* The RST 6.5 and RST 5.5 both are level triggered ie. Input goes to high and stay high until it recognized.
* Maskable interrupt. It is disabled by
  + DI, SIM instruction
  + System or processor reset.
  + After reorganization of interrupt.
* Enabled by EI instruction.
* The RST 6.5 has the third priority whereas RST 5.5 has the fourth priority.

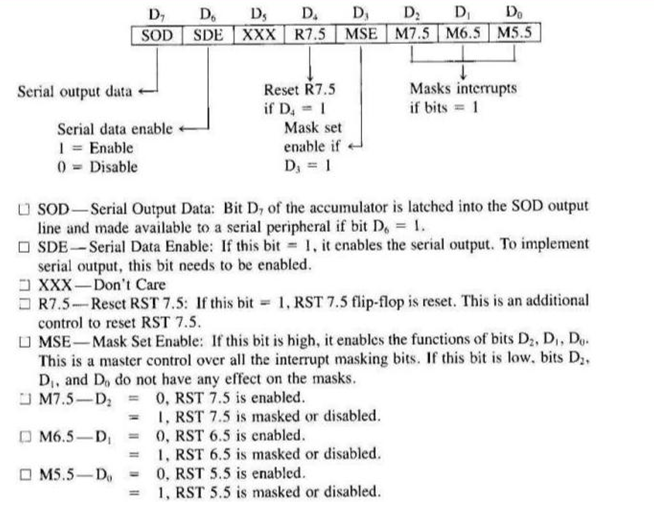
**INTR:**

* INTR is a maskable interrupt.
* It is disabled by
  + DI, SIM instruction
  + System or processor reset.
  + After reorganization of interrupt.
* Enabled by EI instruction.
* Non- vectored interrupt. After receiving INTA (active low) signal, it has to supply the address of ISR.
* It has lowest priority.
* It is a level sensitive interrupts. ie. Input goes to high and it is necessary to maintain high state until it recognized.
* The following sequence of events occurs when INTR signal goes high.
  + - The 8085 checks the status of INTR signal during execution of each instruction.
    - If INTR signal is high, then 8085 complete its current instruction and sends active low interrupt acknowledge signal, if the interrupt is enabled.
    - In response to the acknowledge signal, external logic places an instruction OPCODE on the data bus. In the case of multibyte instruction, additional interrupt acknowledge machine cycles are generated by the 8085 to transfer the additional bytes into the microprocessor.
    - On receiving the instruction, the 8085 save the address of next instruction on stack and execute received instruction.

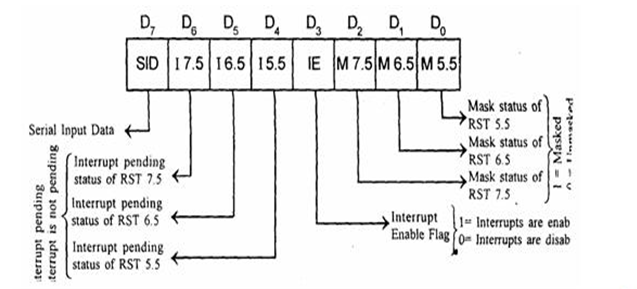
**Interrupt instructions**

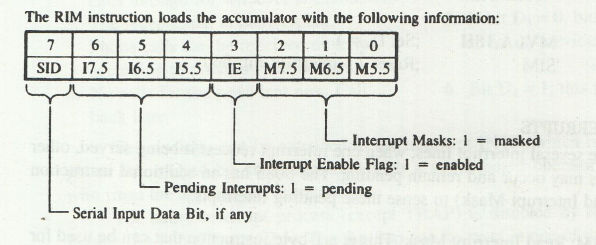
**SIM instruction**

* The 8085 provide additional masking facility for RST 7.5, RST 6.5 and RST 5.5 using SIM instruction.
* This is a multipurpose instruction and used to implement the 8085 interrupts 7.5, 6.5, 5.5, and serial data output.
* The masking or unmasking of RST 7.5, RST 6.5 and RST 5.5 interrupts can be performed by moving an 8-bit data to accumulator and then executing SIM instruction.
* The format of the 8-bit data is shown below.



**RIM instruction**

****



* The status of pending interrupts can be read from accumulator after executing RIM instruction.
* This is a multipurpose instruction used to read the status of RST 7.5, 6.5, 5.5 and read serial data input bit.
* When RIM instruction is executed an 8-bit data is loaded in accumulator, which can be interpreted as shown in above fig.
* 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 masks 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.
* 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.

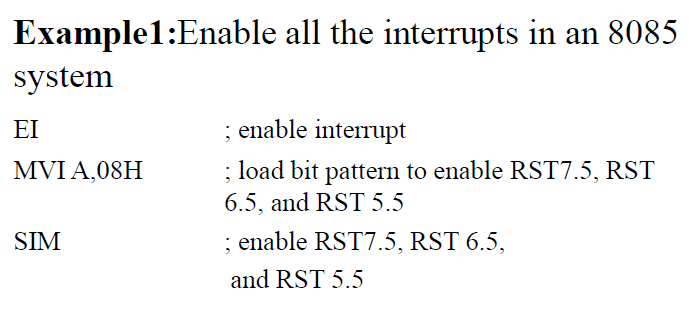
**DI**

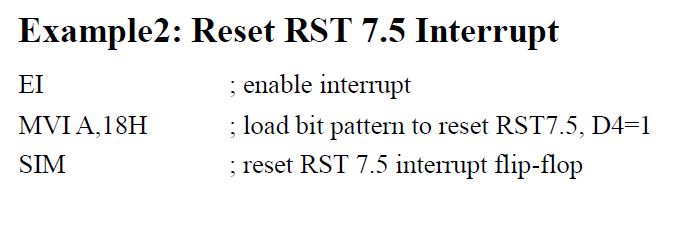
* Disable interrupts
* The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected.
* 1 byte instruction
* Example: DI

**EI**

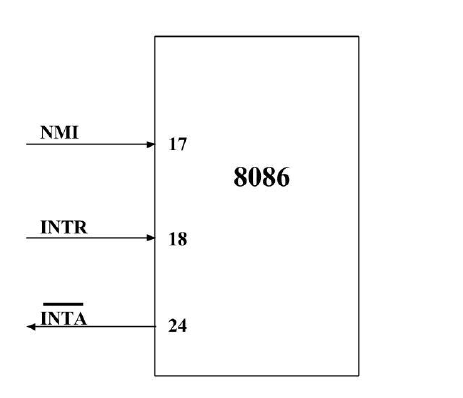
* Enable interrupts
* The interrupt enable flip-flop is set and all interrupts are enabled.
* No flags are affected.
* After a system reset or the acknowledgement of an interrupt, the interrupt enable flip flop is reset, thus disabling the interrupts.
* This instruction is necessary to enable the interrupts (except TRAP).
* 1 byte instruction
* Example: EI

**Interrupt Programming**





**Interrupt Processing in 8086**

****

An interrupt is either a **hardware-generated** CALL or a **software-generated** CALL. Software generated CALL is usually called an **exception**. Either type interrupts the program by calling an Interrupt Service Procedure (**ISP**).

The meaning of ‘interrupts’ is to break the sequence of operation. While the CPU is executing a program, on ‘interrupt’ breaks the normal sequence of execution of instructions, diverts its execution to some other program called Interrupt Service Routine (ISR).After executing ISR , the control is transferred back again to the main program. Interrupt processing is an alternative to polling.

**Interrupt Pins**

**INTR and NMI**

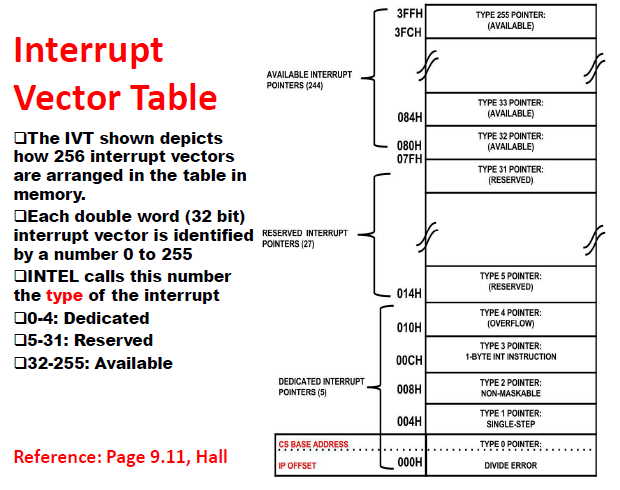
* **INTR** is a maskable hardware interrupt. The interrupt can be enabled/disabled usingSTI/CLI instructions or using more complicated method of updating the FLAGS register with the help of the POPF instruction.
* When an interrupt occurs, the processor stores FLAGS register into stack, disables further interrupts, fetches from the bus one byte representing interrupt type, and jumps to interrupt processing routine address of which is stored in location 4 \* <interrupt type>. Interrupt processing routine should return with the IRET instruction.
* **NMI** is a non-maskable interrupt. Interrupt is processed in the same way as the INTRinterrupt. Interrupt type of the NMI is 2, i.e. the address of the NMI processing routine is stored in location 0008h. This interrupt has higher priority than the maskable interrupt.
  + Ex: NMI, INTR.

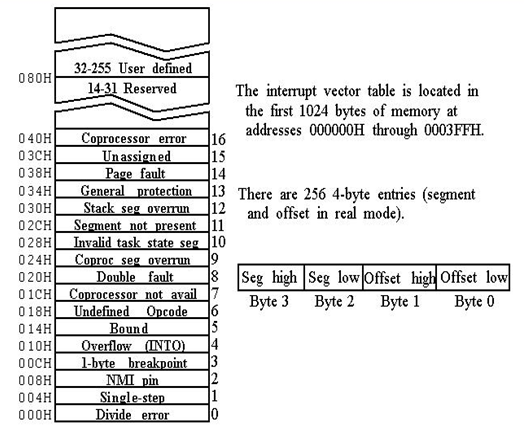
**Interrupt Vector Table and its Organization**

* 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 (00000H - 003FFH).
  + The purpose of the IVT is to hold the vectors that redirect the microprocessor to the right place when an interrupt arrives.

Interrupt Vector Table (IVT) is a 1024 bytes sized table that contains addresses of interrupts. Each address is of 4 bytes long of the form offset: segment, which represents the address of a routine to be called when the CPU receives an interrupt. The first 2 bytes of the vector contain the offset address (IP) and the last two bytes contain the segment address (CS). IVT can hold maximum of 256 addresses (0 to 255). The interrupt number is used as an index into the table to get the address of the interrupt service routine. IVT act as pointers, unlike function call IVT need number as an argument then as a result IVT point us to interrupt service routine (ISR). ISR executes its code, when ISR finished then returns back to original statement. Interrupt vector table is a global table situated at the address 00000:003FFH. The interrupt vector table is a feature of the Intel 8086/8088 family of microprocessors.



****

****

