Module 5

Introduction : I/O module contains logic for operation performing a communication function between the peripheral and the bus.

| ✱ Peripheral : device used to put information into computer |

- Reasons we don't connect peripherals directly to the system bus (we use I/O modules between peripherals and system bus):

a) There are a ~~wide~~ variety of peripherals with various methods of operation. It would be impractical to incorporate the necessary logic ~~whi~~ whithin the processor to control a range of devices.

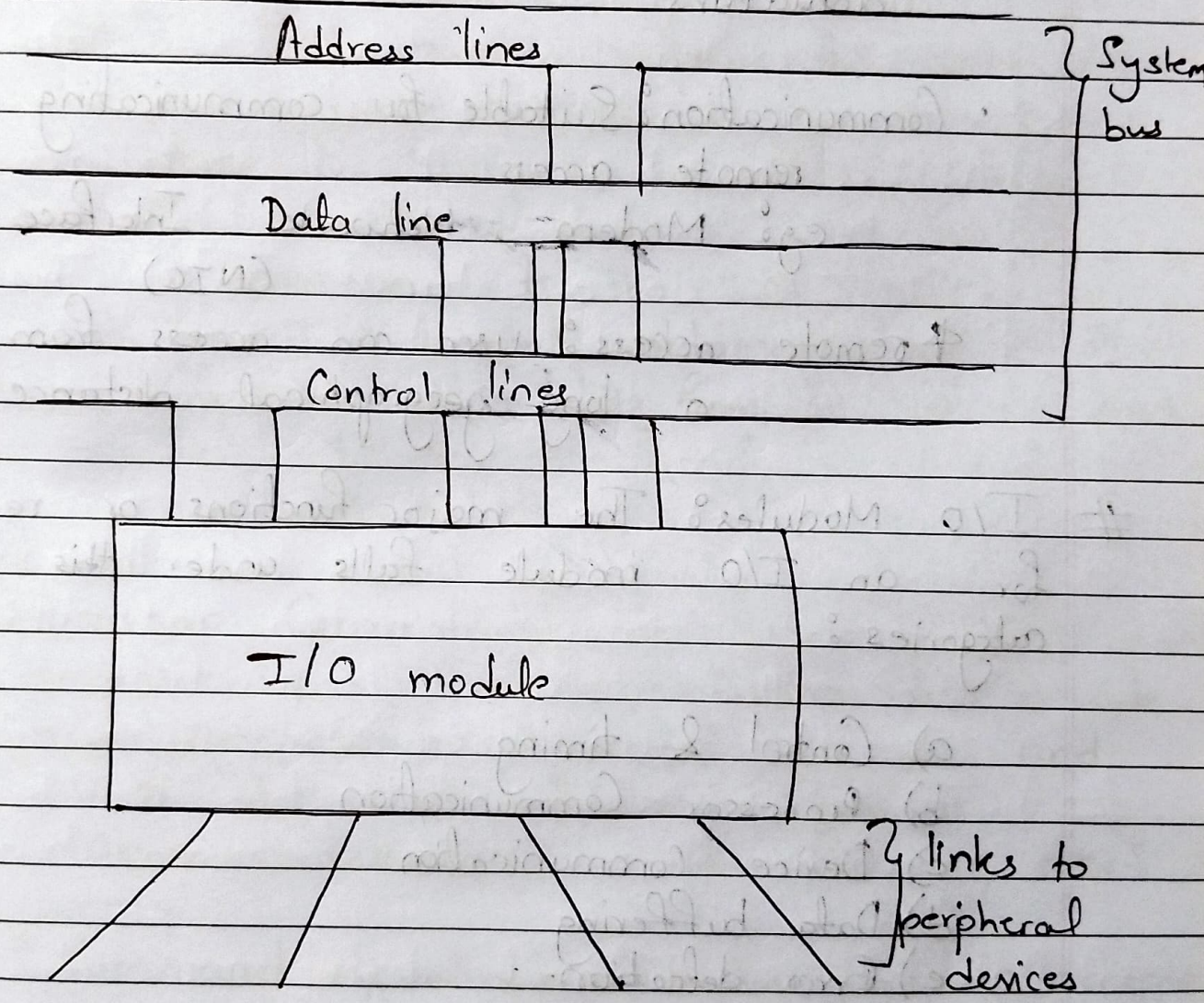
b) Some peripherals are very slow or very fast compared to a system bus. This mismatch can make ~~insufficiencies~~ inefficiency.

c) Peripherals often use different data formats and word lengths than the computer to which they are attached.

- I/O module has 2 major functions:

a) Acts like an interface to the processor and memory via the system bus or central switch.

b) Acts like an interface to one or more peripheral devices by tailored data ~~lines~~ links.



External Devices: External devices provide a means of exchanging data between the external environment and the computer. An external device connected to an I/O module is often ~~refl~~ referred to as a peripheral device or simply peripheral.

We can classify external devices into three categories

- Human Readable: Suitable for communicating with the computer user.
  eg: Screen, printer, keyboard.

- Machine Readable: Suitable for communicating with equipment.
  eg: magnetic disk & tape, sensors & actuators.

- Communication: Suitable for communicating with remote access.
  eg: Modem, Network Interface card (NIC)

* remote access: user can access from a long geographical distance.

I/O Modules: The major functions or requirements for an I/O module falls under ~~this~~ following categories:

a) Control & timing
b) Processor Communication
c) Device Communication
d) Data buffering
e) Error detection.

a) Control and Timing: The processor may communicate with multiple ~~I/O devices~~ external devices. The internal memory resources, such as main memory and system bus, must be shared among a number of activities, including data

I/O. Thus, the I/O function includes a control and timing requirement to coordinate the flow of traffic. The data transfer from an external device to the processor might involve the following sequence steps:

i) The processor interrogates the I/O module to check the status of the attached device.
ii) The I/O module returns the device status.
iii) If the ~~dive~~ ~~divi~~ device is operational and ready to transmit, the processor requests the transfer of data, by means of a command to the I/O module.
iv) The I/O module obtains a unit of data (eg 8 bits / 16 bits) from the external device.
v) The data are transferred from the I/O module to the processor.

b) Processor Communication: The previous simplified scenario also illustrates that the I/O module must communicate with the processor and with the external device. Processor communication involves the following ~~steps~~:

i) Command decoding: The I/O module accepts commands from the processor, typically sent as signals on the control bus.
Eg: An I/O module for a disk drive might accept the following commands:
READ SECTOR, WRITE SECTOR.

ii) Data: Data are exchanged between the processor and the I/O module over the data bus.

iii) Status Reporting: Since peripherals are mostly very slow, lets say an I/O module is asked to send data to the processor (read), it may not be ready to do so because it is still working on the previous I/O command. This is reported using status flags. Status flag examples: BUSY, READY.

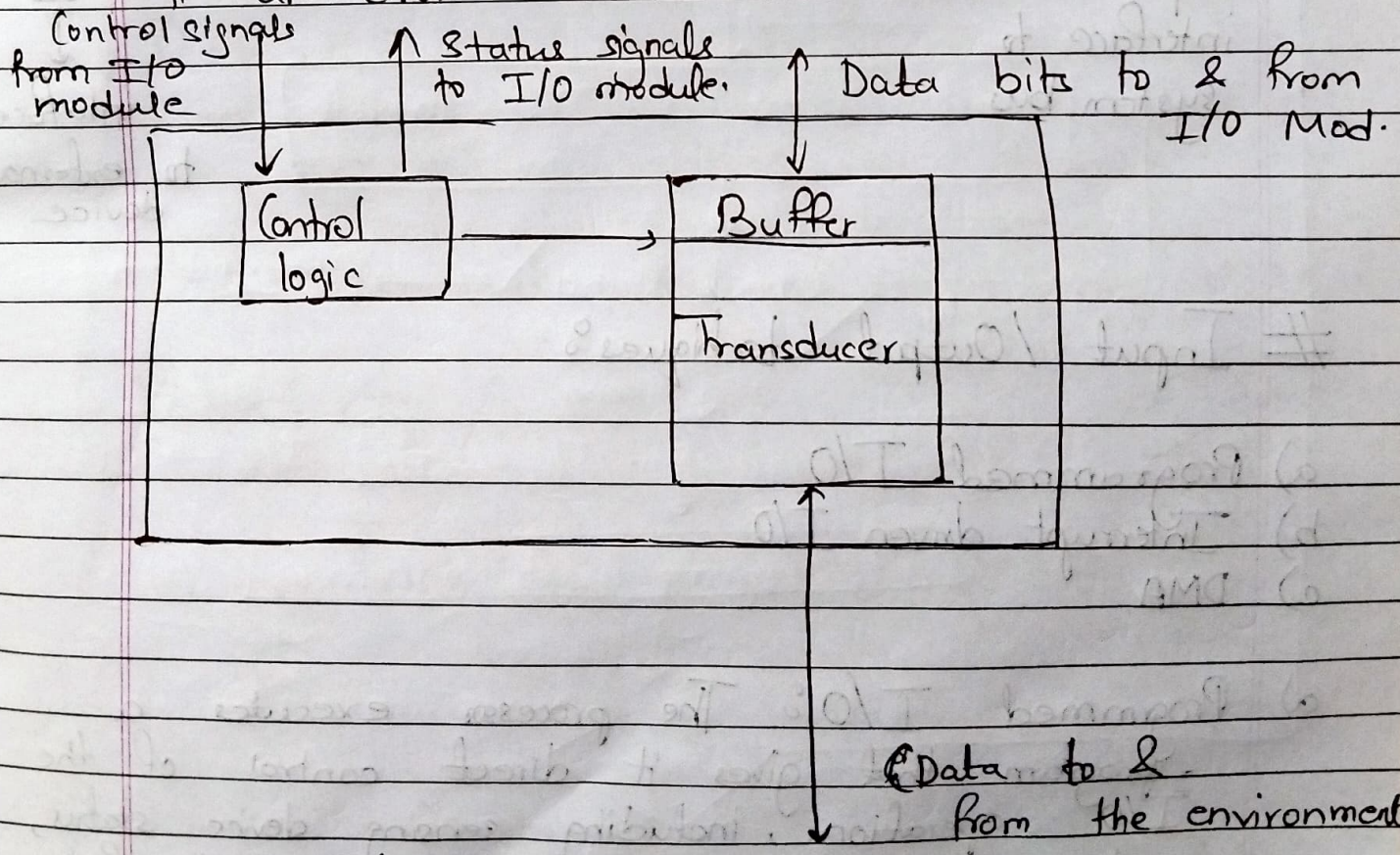
iv) Address recognition: Just as each memory has an address, so does each I/O device/ External device. Thus, an I/O module must recognize one unique address for each peripheral that it controls.

c) Device Communication: Comunications that involve commands, status information and data are called device communication.
(As we saw before, BUSY, READY, READ).
The I/O module must be able to perform these device communications.

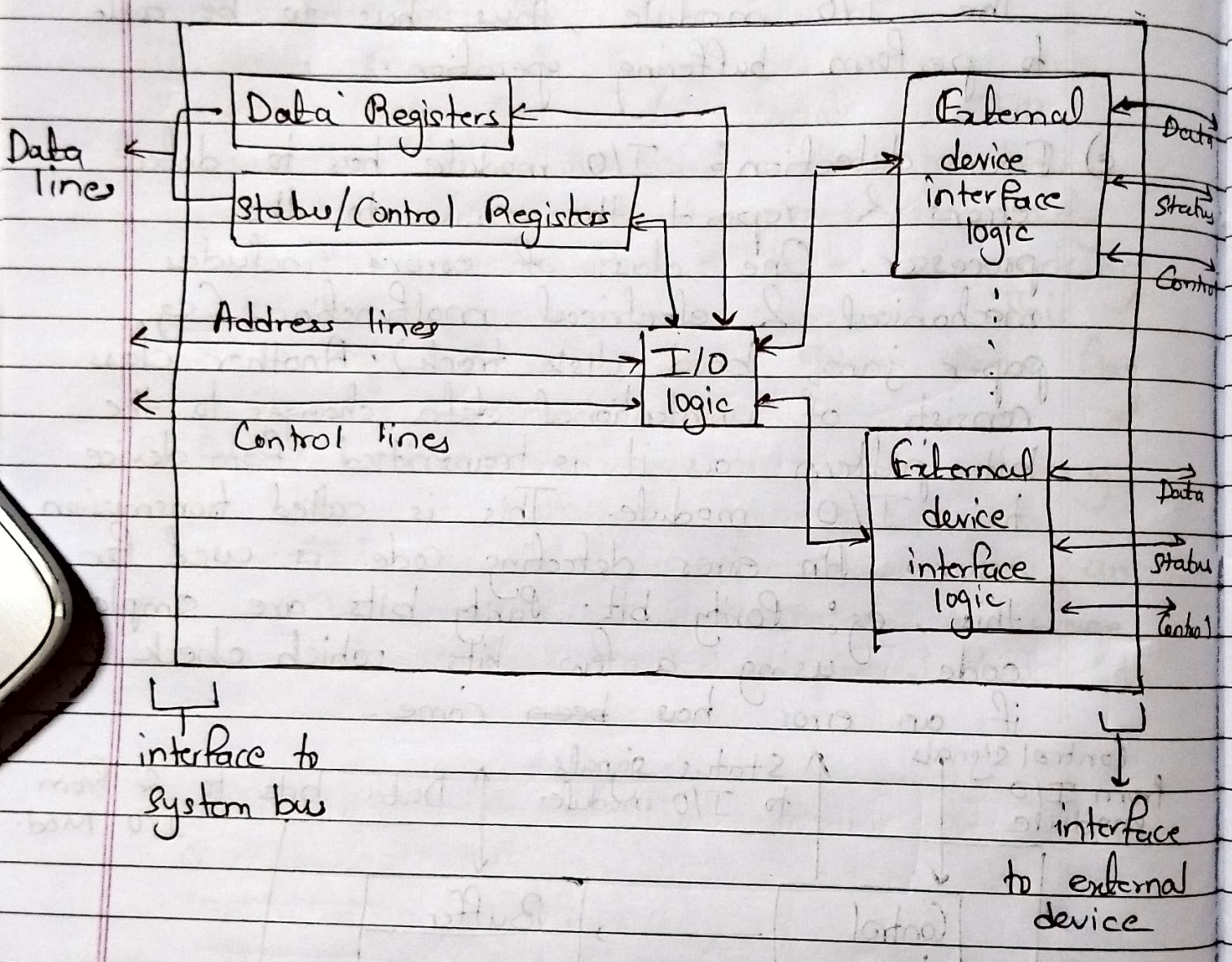
d) Data buffering: External devices are orders of magnitude slower than main memory.
The data transfer between main memory & I/O is much faster ~~that~~ than the data transfer between External device & I/O module. Thus, the data from the main memory has to be buffered (stored somewhere

for some time) ~~before~~ to match the speed of data transfer from external device. The I/O module, thus, has to be able to perform buffering operation.

e) Error detection: I/O module has to detect errors & report the errors to the processor. One class of errors includes mechanical & electrical malfunctions (eg paper jam, bad disk track). Another class consists of unintentional ~~data~~ changes to the bit pattern as it is transmitted from device to I/O module. This is called transmission error. An error detecting code is used for this, eg: Parity bit. Parity bits are simple code using a few bits which check if an error has ~~been~~ come.



Block diagram of an I/O module:



Input / Output techiques:

a) Programmed I/O
b) Interrupt driven I/O
c) DMA.

a) Programmed I/O: The processor executes a program that gives it direct control of the I/O operation, including sensing device status, sending read or write command, and transfering

the data. When the processor issues a command to the I/O module, it ~~mod~~ must wait till the I/O operation is complete. This is a wasteful process because it wastes processor time.
When the I/O module is done with the process it does not interrupt the processor. It is the responsibility of the processor to check the status periodically to tell if the I/O ~~mod~~ module has completed process.

- I/O commands: To execute an I/O instruction, the processor issues an address, specifying the particular I/O module & external device and an I/O command. There are 4 types of I/O commands:

i) Control: Used to activate a peripheral & tell it what to do. Eg: a tape instructed to rewind or forward.
These commands are tailored to the particular type of peripheral device.

ii) Test: Used to test various status conditions associated with an I/O module and its peripherals. The processor will want to know if the peripheral is available for use, or if the most recent I/O operation is completed & if any errors occured.

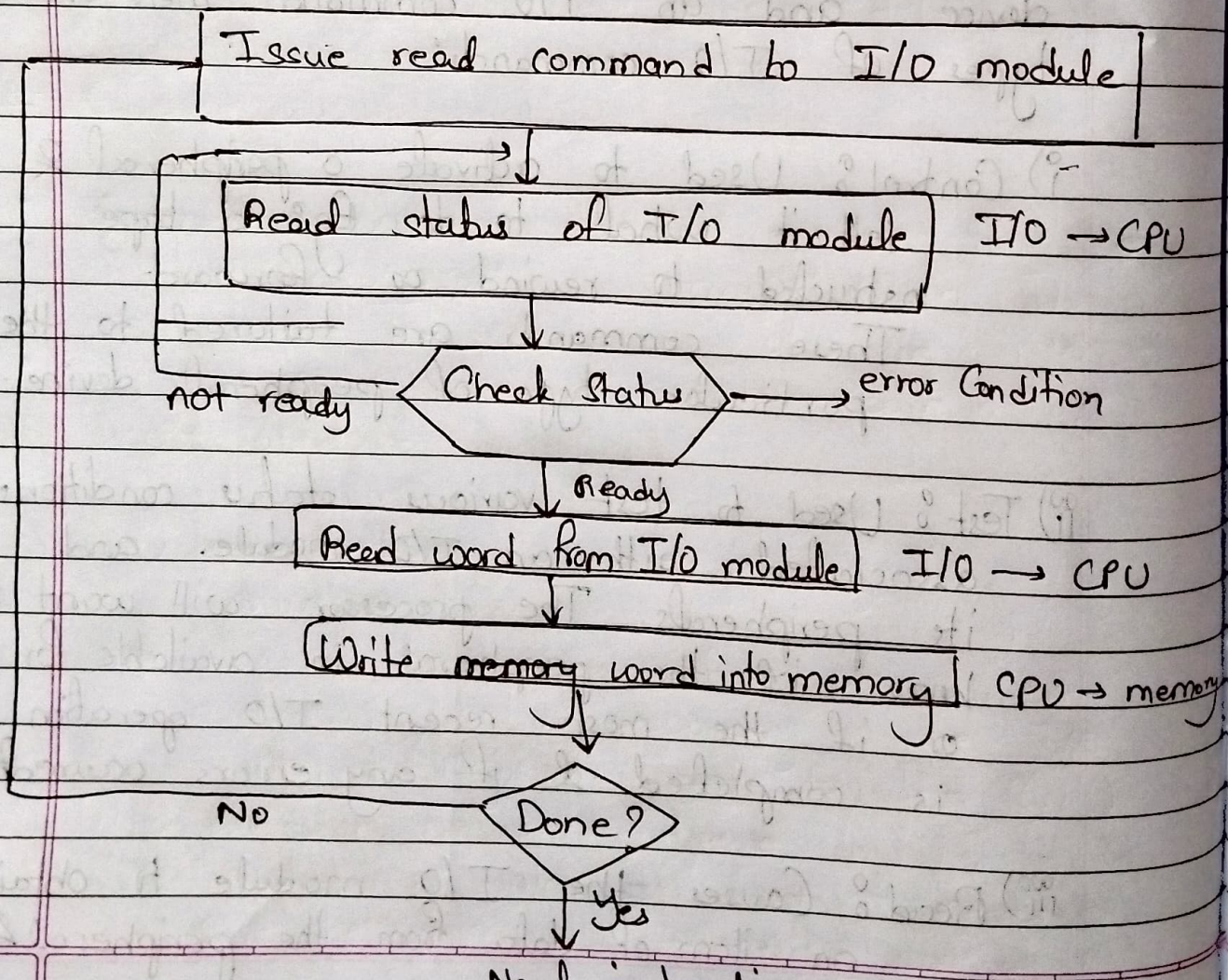
iii) Read: Causes the I/O module to obtain an item of data from the peripheral &

place it in the internal buffer. Processor obtains this data item by requesting that the I/O module place it on the data bus.

iv) Write: Causes the I/O module to take an item of data (byte or word) from the data bus & subsequently transmit that data item into the peripheral.

(The slides after this only explain what has already been explained & covered here before commands started)

Flowchart:



b). Interrupt driven I/O: As seen earlier, the problem with programmed I/O is that the processor's time is wasted since it waits for the ~~in~~ I/O module to be ready. The solution is for the processor to issue an I/O command ~~then~~ to a module then ~~gon~~ go on and do some other work. The I/O module will then interrupt the processor to request service when it is ready to exchange data. The processor then executes the data transfer and resumes its former processing.

- How this works:

i) I/O module's viewpoint:
I/O receives a READ command from the processor. The I/O module then proceeds to read data in the associated peripheral. Once the data is in I/O module's data register, the module signals an interrupt to the processor. The module waits till the data is requested by the processor and after it's requested the module places the data on the data bus & gets ready for the next I/O operation.

ii) Processor's viewpoint:
The processor issues a READ command to the I/O module then goes off to do something else. When an interrupt

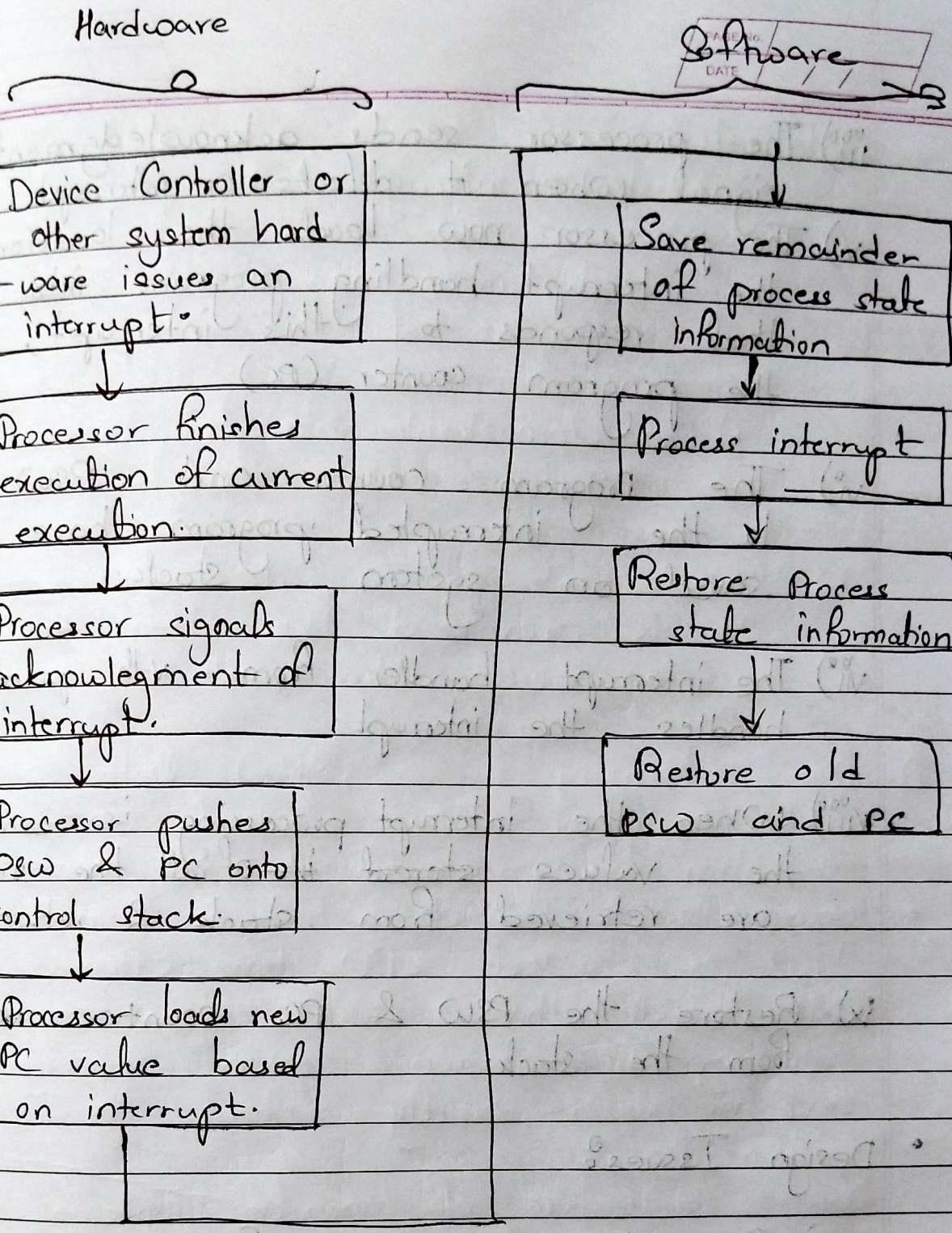
from the I/O module occurs, the processor saves the context of the current program & processes the interrupt. The processor reads the data from the I/O module & stores the data. It then resumes execution of the program it was working on.

iii) General viewpoint:
→ CPU issues read command
→ I/O module gets data from peripheral while CPU does other work
→ I/O module interrupts CPU.
→ CPU requests data.
→ I/O module transfers data.

~~Classes of interrupt:~~

- Interrupt processing: Let us consider the role of the processor in interrupt driven I/O in more detail. The occurence of an interrupt triggers a number of events, both in hardware and software.

i) Device issues an interrupt signal to the processor.
ii) Processor finishes execution of current execution befor responding to the I/O interrupt. (Forgot (iii) point, its after (iv))
iv) Processor needs to prepare for transfer of control to interrupt. First it needs to save the data of currently executing



program. The information required is PSW (program status word), which is the register that contains the status of the processor, and PC (program counter) which contains the location of the next instruction. These 2 are pushed into a stack called control stack.

iii) The processor sends acknowledgment signal when it detects an interrupt.

v) The processor now loads the location of the interrupt handling program ~~into the~~ that responds to this interrupt, into the program counter (PC)

vi) The Program counter and PSW relating to the interrupted program have been saved on system stack.

vii) The interrupt handler from vth point handles the interrupt.

viii) When the interrupt processing is complete, the values ~~stored in~~ of all the registers are retrieved from stack & returned.

ix) Restore the PSW & PC counter values from the stack.

- Design Issues:

i) For multiple I/O devices for ~~a~~ multiple peripherals, how does the processor determine which device issued the interrupt?

~~Ans~~

Solutions:

1) Provide multiple interrupt lines:
Provid multiple interrupt lines between the processor and I/O modules.

PAGE No.
DATE / /

2) Software polls: When the processor detects an interrupt, it branches to an interrupt - service routine whose job is to poll each I/O module to determine which module caused the interrupt.

3) Daisy chain: another name for hardware poll (don't understand ☺)

Types of interrupts:

a) Program interrupts: eg: divide by 0

b) Timer interrupts: Allows to perform a task at very specifically timed intervals.

c) I/O interrupts: Process of data transfer in which an external device communicates with the processor. (This module covers this interrupt).

d) Hardware failure interrupts: Interrupts processor when hardware failure occurs like power failure, memory parity error, etc.

Pentium interrupts:

a) Hardware interrupts: It can happen at any random time during process execution. There are two types of hardware
(Like compile time error)

interrupts:

i) Maskable interrupts: The processor does not recognize a maskable interrupt ~~it~~ unless an "interrupt enable" flag is set.

ii) ~~Un~~. Unmaskable interrupts: Processor can recognize the interrupt.

b) Software interrupts: Called exceptions. It is provoked ~~like~~ after the program is executed, like runtime exception.

4) Direct Memory Access (DMA): Both programmed and interrupted I/Os required constant intervention by processor, this slowed down the I/O data transfer (for interrupt I/O) or wasted CPU time (Programmed I/O). DMA tackles this issue by, essentially, allowing I/O module & main memory to exchange data directly, without processor involvement.

DMA is a feature of computer systems that allow certain hardware subsystems to access main system memory independently of the CPU.

The DMA module is capable of mimicking the processor & taking over control of the system. It needs this control to use the system bus for transferring data.

When the processor wishes to read or write

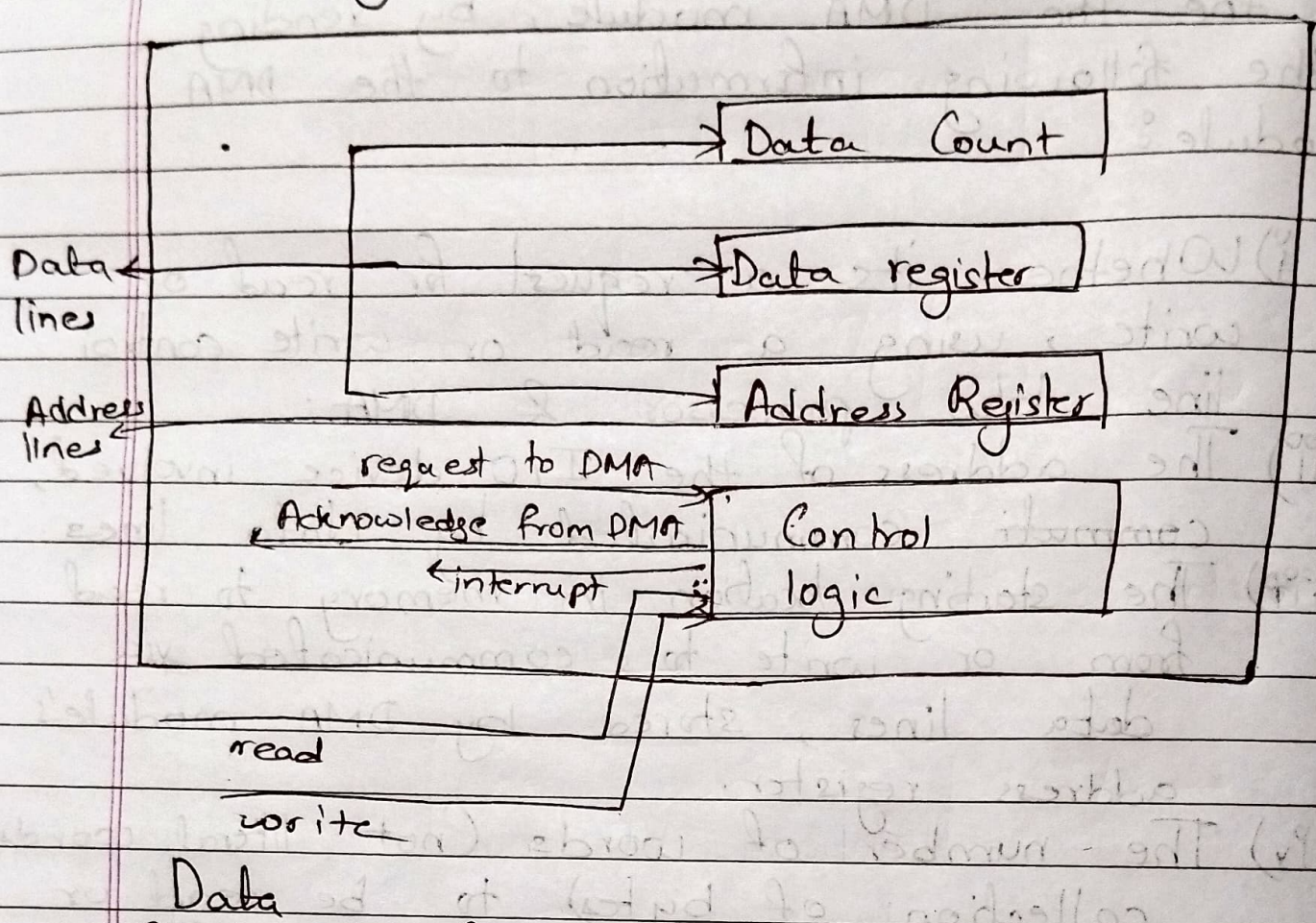
a block of data, it issues a command to the the DMA module, by sending the following information to the DMA module:

i) Whether its a request for read or write, using a read or write control line b/w processor & DMA.
ii) The address of the I/O device involved, ~~commuti~~ communicated by data lines.
iii) The starting location in memory to read from or write to, communicated via data lines, stored by DMA module's address register.
iv) The number of words (not literal words, collection of bytes) to be read or written, communicated via data lines.

Now the processor continues with other work. DMA module is basically an employee and processor is the boss. Processor has delegated / entrusted this I/O device to the DMA module.

DMA controller (term for DMA module in 80x86 family) deals with transfer by gaining access & control over buses from processors.
DMA controllers will send an ~~ite~~ interrupt when finished.

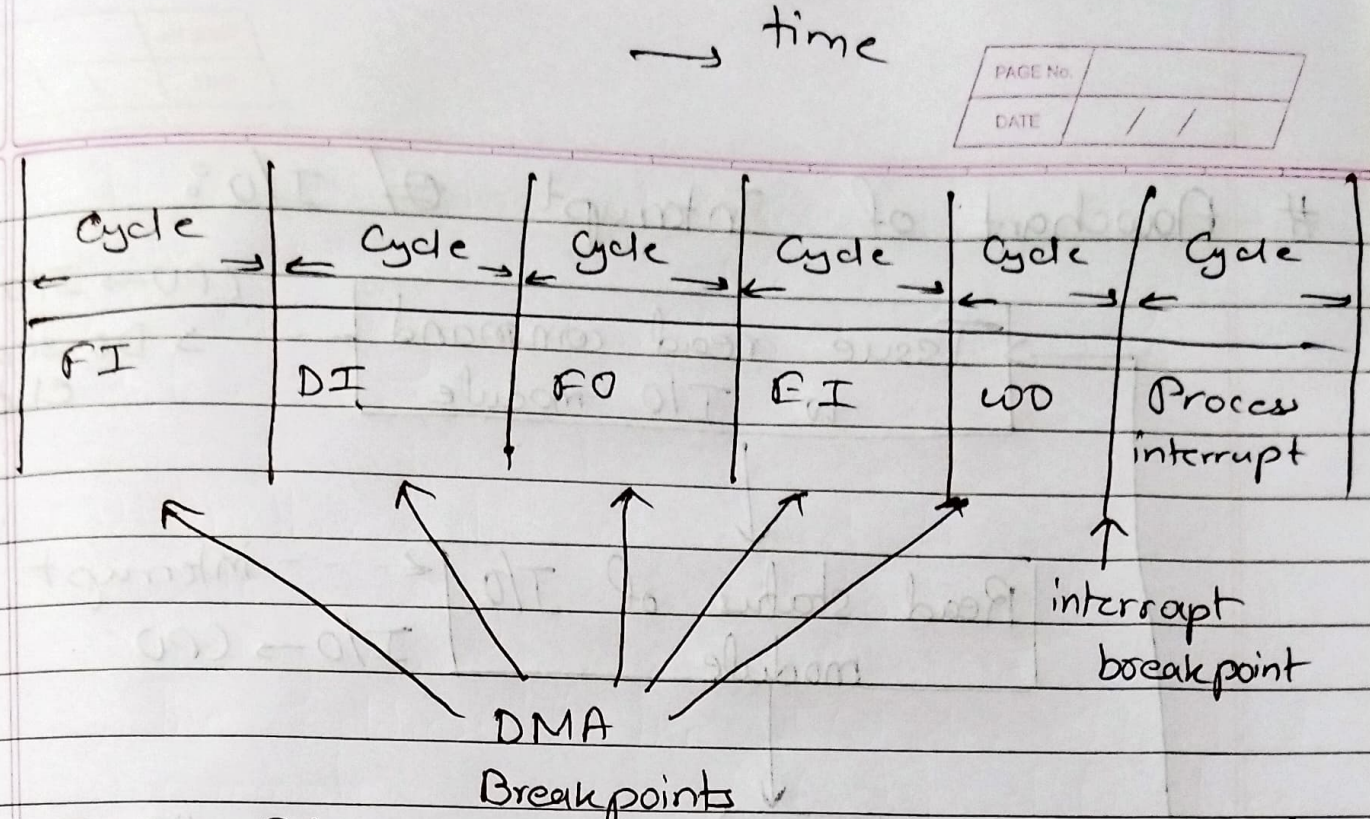
- Block diagram of DMA:



- ~~Block~~ Data transfer modes of DMA:

a) DMA block transfer: Burst of data is transferred before CPU takes control of the busses back.

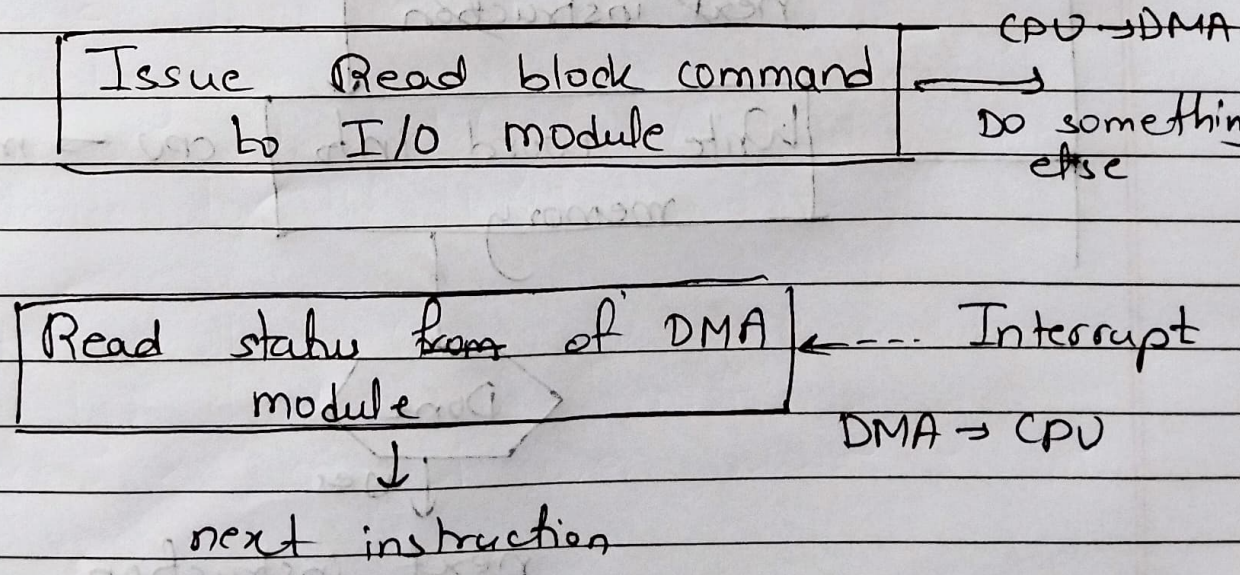
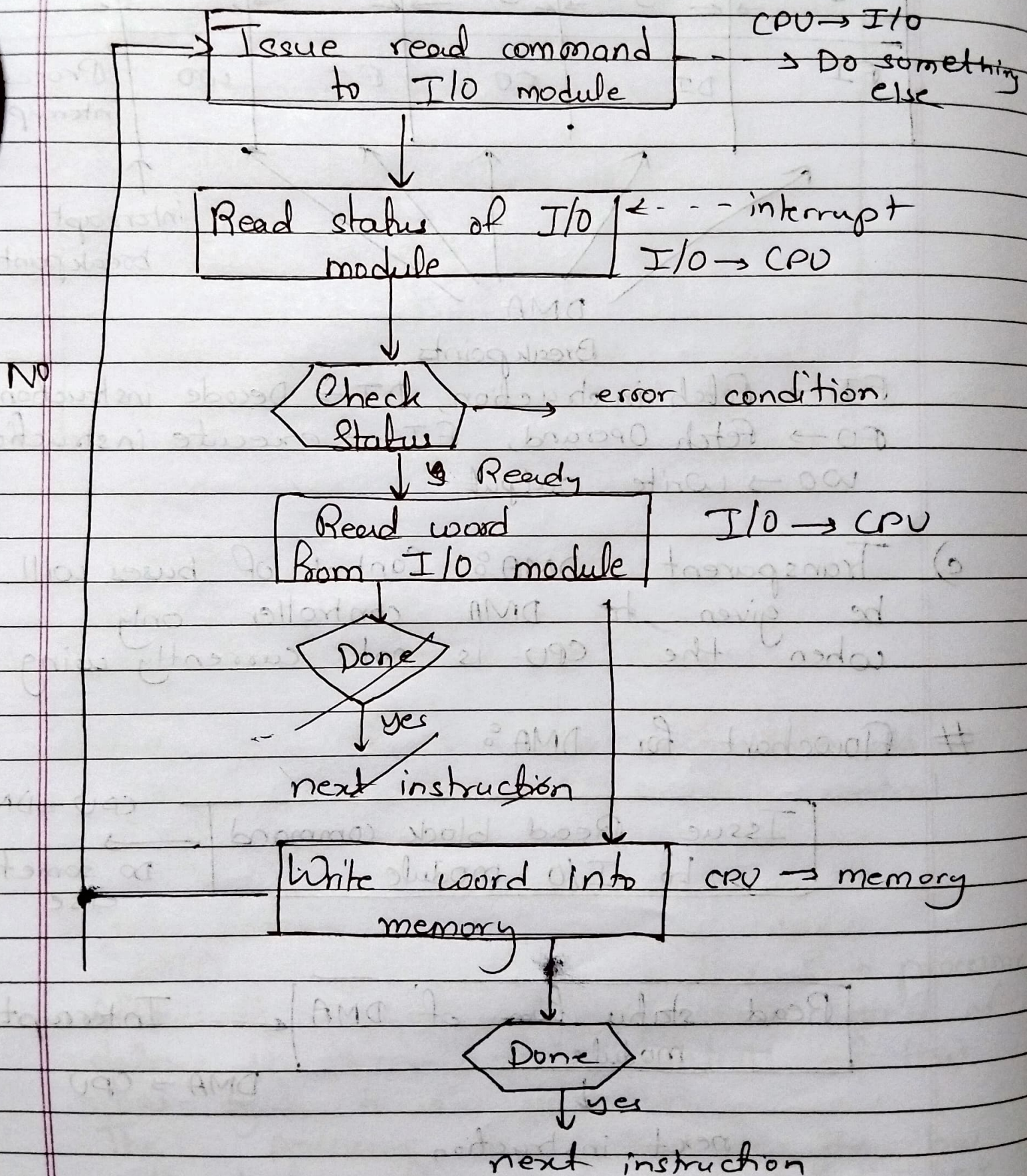
b) Cycle stealing mode: DMA module steals a bus cycle from processor. After the transfer of one ~~data~~ word of data it must return control of that bus cycle to the processor.
The processor & DMA use the bus alternatively, after each cycle (FI, DI, FO, EI, WO)



FI → Fetch instruction, DI → Decode instruction, FO → Fetch Operand, EI → execute instruction, WO → Write output.

c) Transparent DMA: Control of buses will be given to DMA controller only when the CPU is not currently using it.

Flowchart for DMA:



Flowchart of Interrupt ~~of~~ I/O: