Microprocessors

Unit-2

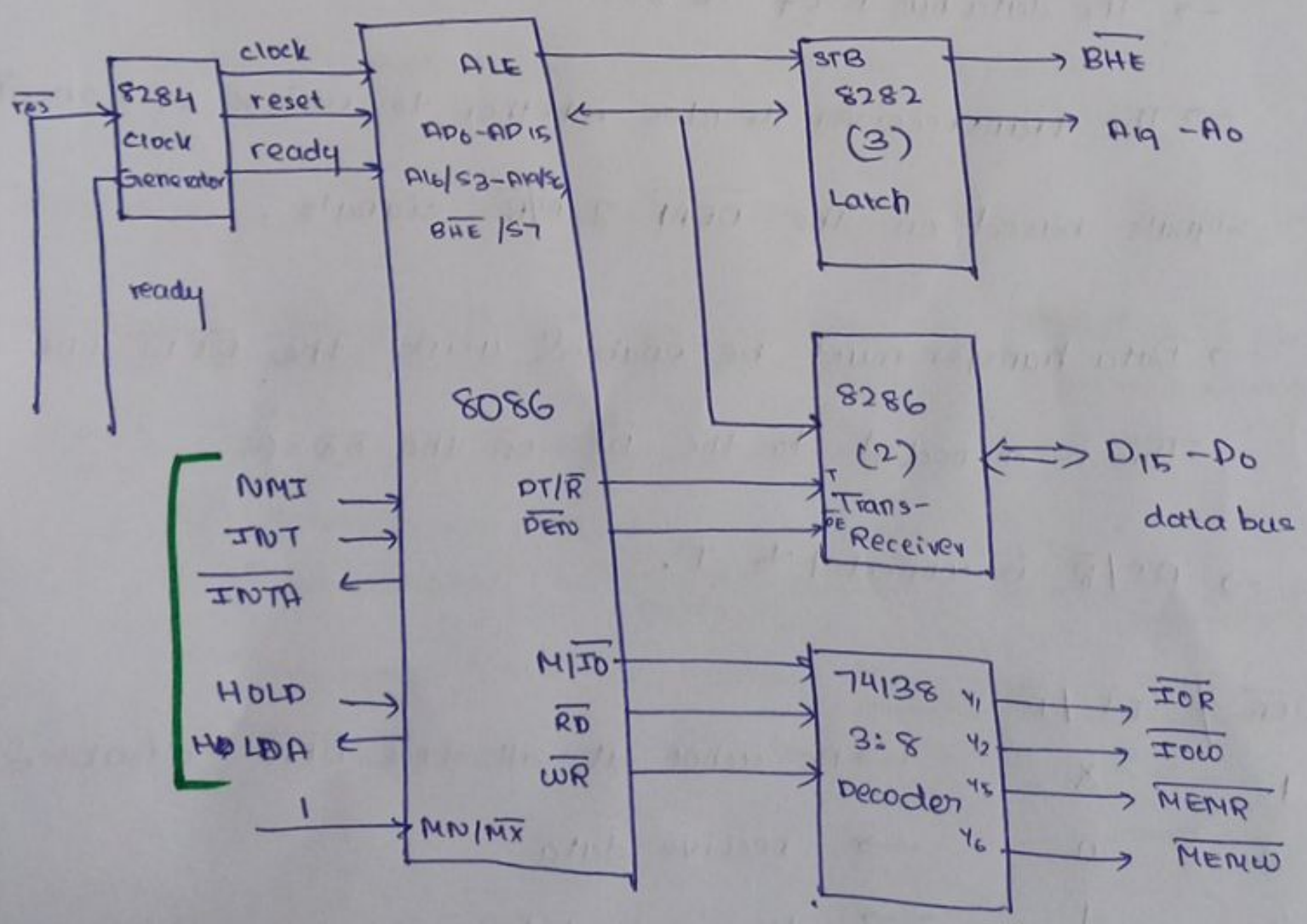
* Modes of Operation of 8086 (Minimum & Maximum Mode)

PDF-1

A. Minimum Mode

→ 8086 works in min mode when $MN/\overline{MX} = 1$. In this mode, 8086 is the only processor in the system.

Diagram



Clock - The clock is provided by the 8284 clock generator.

Addresses - → Addresses from the 8086 are given to the 8282 8-bit latch

→ The address bus is of 20 bits ⇒ use 3 8282 latches

→ ALE is connected to the STB.

Data - → The data bus is driven through the 8-bit 8286 transreceiver.

→ The data bus is of 16 bits ⇒ use 2 8286s.

→ The transreceiver decides whether to receive or transmit signals based on the $\overline{DEN}$ & $DT/\overline{R}$ signals.

→ Data transfer must be enabled using the $\overline{DEN}$ line. This is connected to the $\overline{OE}$ on the 8286.

→ $DT/\overline{R}$ is connected to T.

$\overline{DEN}$	$DT/\overline{R}$	
1	X	→ no action at all since $\overline{DEN}$ is disabled
0	0	→ receive data
0	1	→ transmit data

Control Signals → The different operations that can be performed are the memory read
memory write
I/O read
I/O write.

→ These operations are chosen by decoding the $M/\overline{IO}$, $\overline{RD}$ and $\overline{WR}$ signals.

→ $M/\overline{IO}$, $\overline{RD}$ and $\overline{WR}$ signals are decoded using a 3:8 decoder like the IC 74138.

Direct Memory Access → done using the HOLD & HLDA signals to make and respond to bus requests

Interrupts – There are pins for NMI, INTR and $\overline{INTA}$ lines

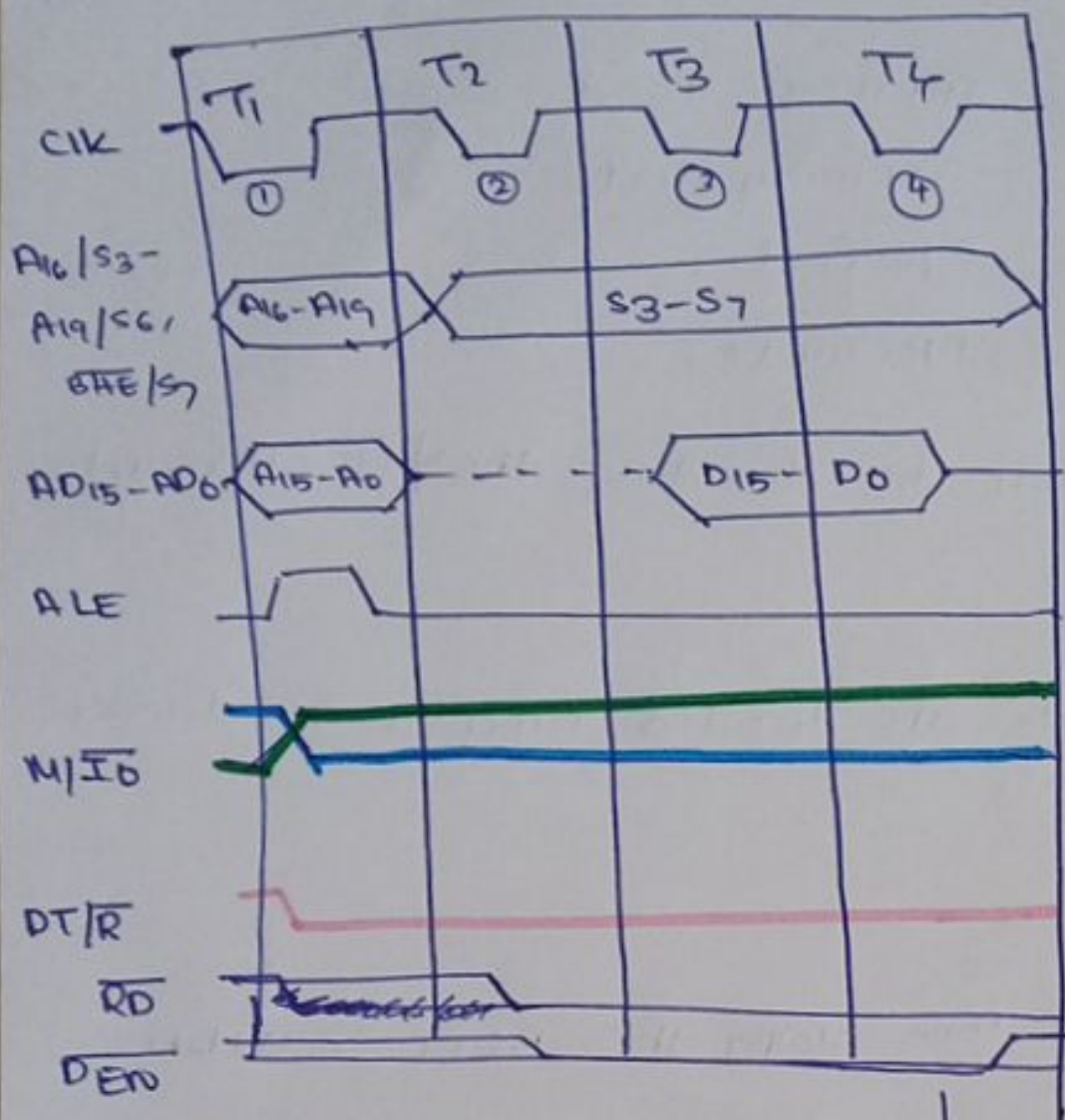
* Timing Diagrams for Minimum Mode

⇒ for the write operations

Steps: There are 4 T states, which are:

(i) Processor gives address
(ii) Processor gives $\overline{RD}$ (or $\overline{WR}$) asking for data (or writing data)
(iii) Data comes from the memory (Data goes out of memory)
(iv) Processor captures and stores the data (Processor writes all the data)

Read Cycle



Step0 : set up a clock pulse

Step2: Addresses are set, status also comes through

Step1 : set ALE to 1, so that addresses can be carried

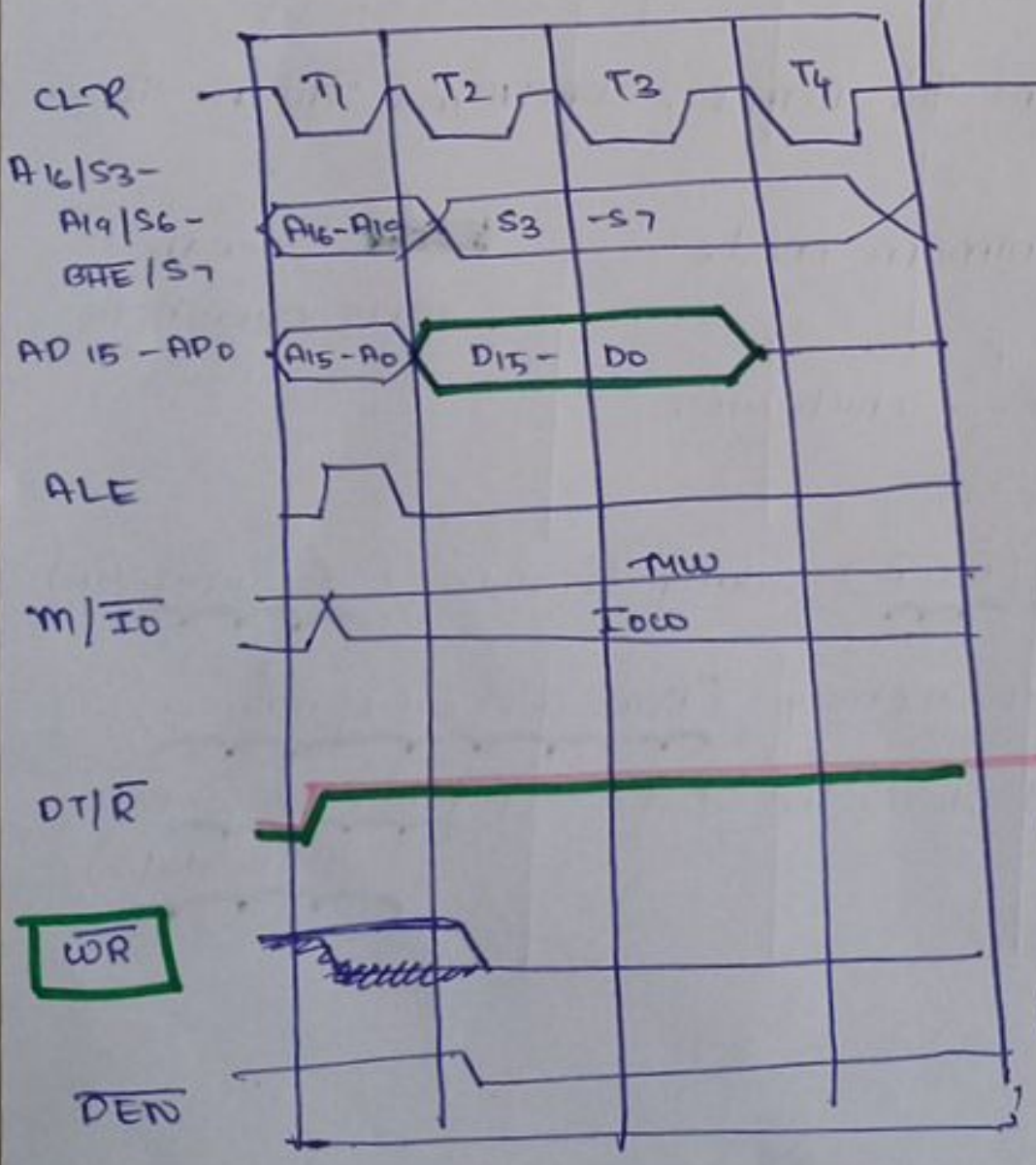
Step4 - designate the type of operation
low ⇒ IOR high ⇒ MR

Step5 - see if you're doing a data transfer or read operation
read ⇒ low

Step6 - $\overline{RD}$ goes low asking μP for address - make it go low after finding addr.

Step7 Make transreceiver go low after setting addr, and until data transfer is complete.

Write Cycle

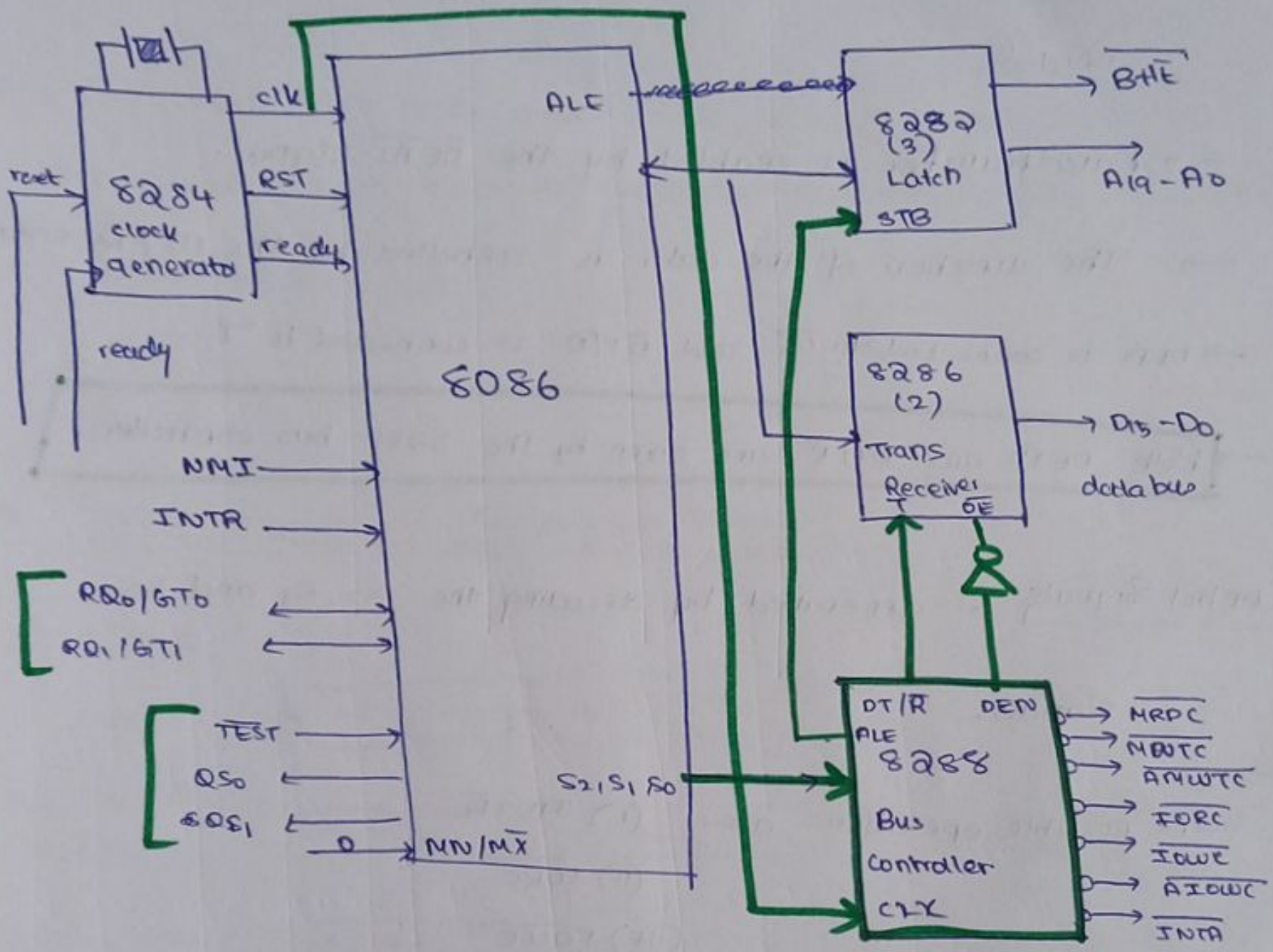


Changes vs Read Cycle

1. no delay in data
2. make DT/$\overline{R}$ - since data transfer is happening
3. change $\overline{RD}$ read signal to $\overline{WR}$ write signal

B | 8086 in Maximum Mode

Diagram



→ 8086 works in maximum mode when $MN/\overline{MX} = 0$

→ Additional processors like 8087 / 8089 can be connected.

Clock → The clock is provided by the 8284 clock generator.

Addresses → Addresses from the address bus is latched to the 8282 8-bit latch

→ There are 20 lines, so 3 latches are needed

→ The address latch enable (ALE) is given from the bus controller.

Data -> The data bus is driven through the 8-bit 8286 transreceiver.

→ Since the data is of 16 bit - two such transreceivers are needed.

→ Data transfer is enabled by the $\overline{DEN}$ signal.

→ The direction of the data is controlled by the $DT/\overline{R}$ signal

→ DEN is connected to $\overline{OE}$ and DT/R is connected to T

→ Both DEN and $DT/\overline{R}$ are given by the 8288 bus controller.

Control Signals - generated by decoding the S_2, S_1 and S_0 signals.

Some possible operations are:
(i) $INTA$
(ii) $\overline{IORC}$
(iii) $\overline{IOWC}$
(iv) $\overline{AIOWC}$
(v) $\overline{AMWTC}$
(vi) $\overline{MRDC}$
(vii) $\overline{MWTC}$

→ The S_2, S_1, S_0 signals are decoded through the 8288 bus controller.

Bus Arbitration - done with the RQ_0/GT_0 & RQ_1/GT_1 lines

→ RQ_0/GT_0 has the higher priority

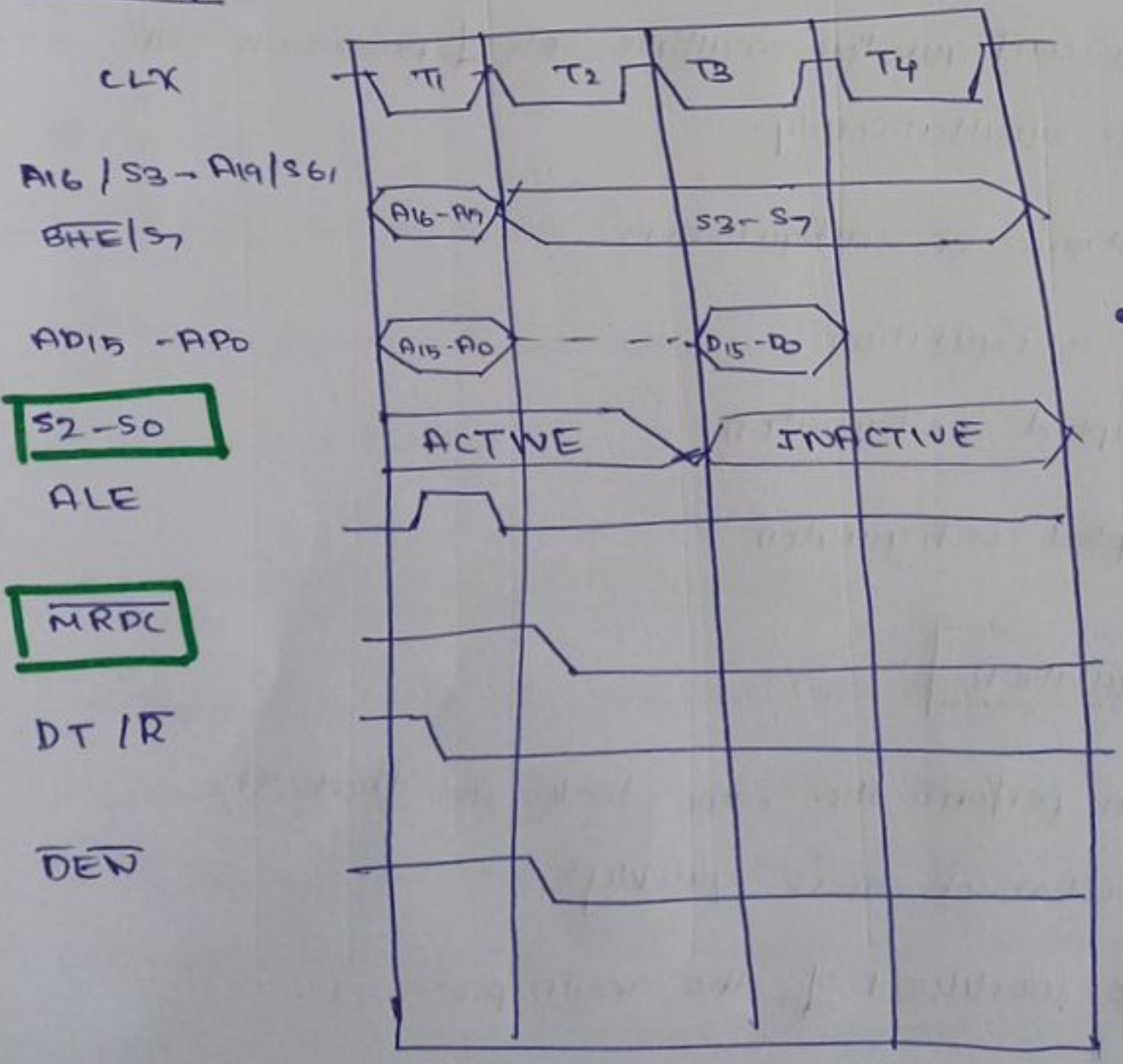
Interrupts - has NMI and INTR

- The $\overline{INTA}$ line is present in the 8288 bus controller.

→ In general, the max mode circuit is more complex than the min mode, but since it supports multiprocessing, it gives a better performance.

* Timing Diagrams for Maximum Mode

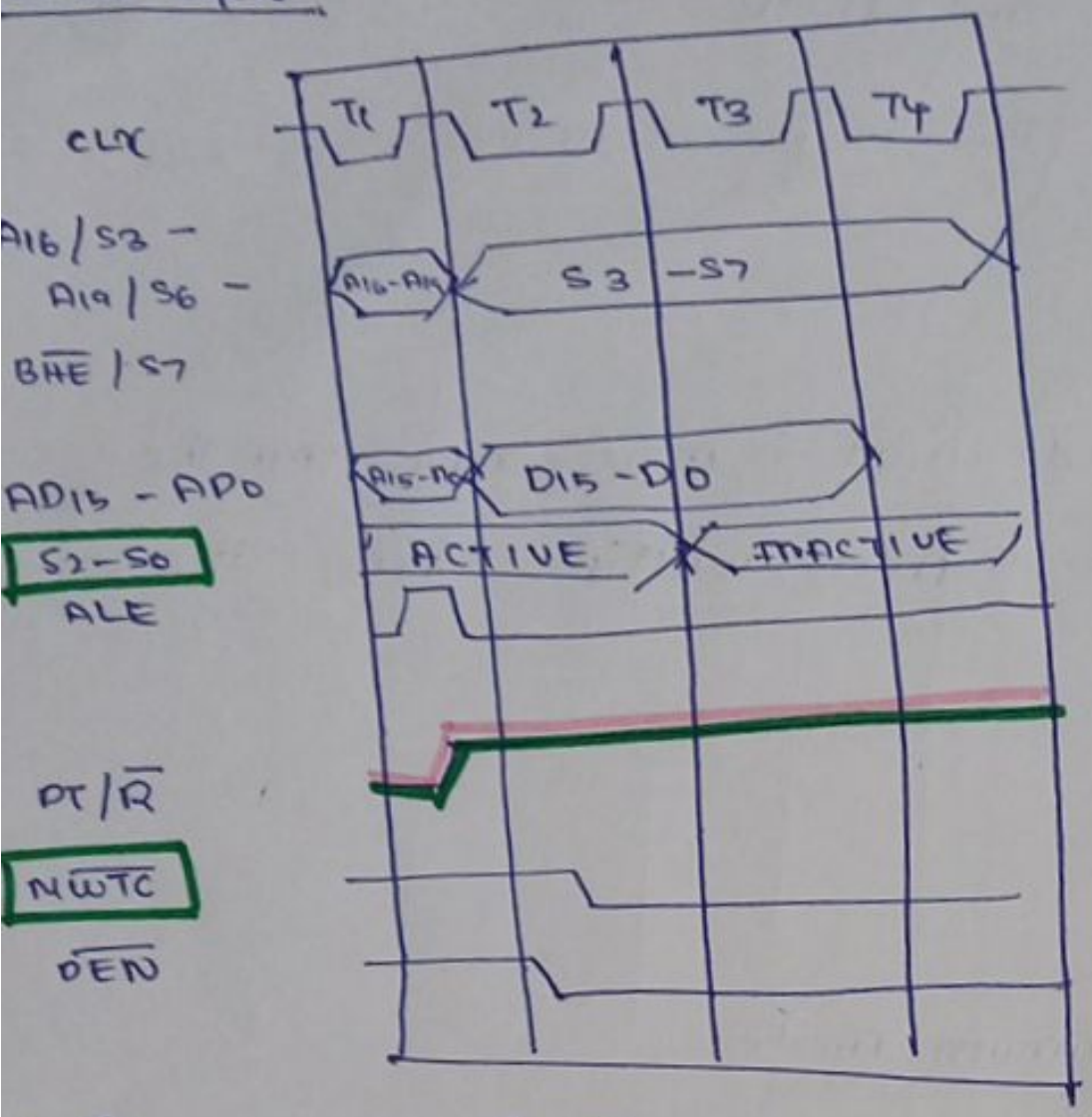
Read Cycle



changes in

- only difference from the reading cycle of min-mode is that instead of $\overline{RD}$, use $\overline{MRDC}$
- and the S2-S0 line
  ⇓
- no M/$\overline{IO}$

Write Cycle



* Multiprocessor Configuration

→ In multiprocessor configuration, multiple sets of processors can execute instructions simultaneously.

→ There are 3 multiprocessor configurations:

(i) coprocessor configuration

(ii) closely coupled configuration

(iii) loosely coupled configuration

A. Coprocessor Configuration

→ A coprocessor can perform the same tasks as that the microprocessor, albeit much more quickly.

→ It reduces the workload of the main processor.

→ The coprocessor shares the same memory IO system, bus control logic and clock generator.

<u>Example</u> - 8086 can be connected to the math coprocessor, which can perform complex mathematical operations easily.

Connecting the Processor and Coprocessor

→ connected via the TEST, RQ/GT and QS_0 & QS_1 signals

(i) TEST of processor ⟶ BUSY of coprocessor

(ii) other pins have the same name on both the processor and coprocessor.

→ TEST checks the status of the coprocessor - whether it is busy or idle.

→ The RQ/GT is used for bus arbitration.

→ The coprocessor uses QS_0 and QS_1 to check the status of the queue of the host processor.

B. Closely Coupled Configuration

→ Similar to the coprocessor configuration - both share the same memory, I/O system bus, control logic, clock generator.

→ However, the coprocessor & host processor fetch and execute their own instructions. The system bus is controlled by the coprocessor and host processor independently.

c. Loosely Coupled Configuration

→ has a number of modules of microprocessor systems - which are connected through a common system bus.

→ Each module has its own clock generator, memory, I/O devices which are all connected through a local bus.

→ The clocks of all the devices are of similar frequency, but are asynchronous in nature.

→ Each module is capable of being the bus master.

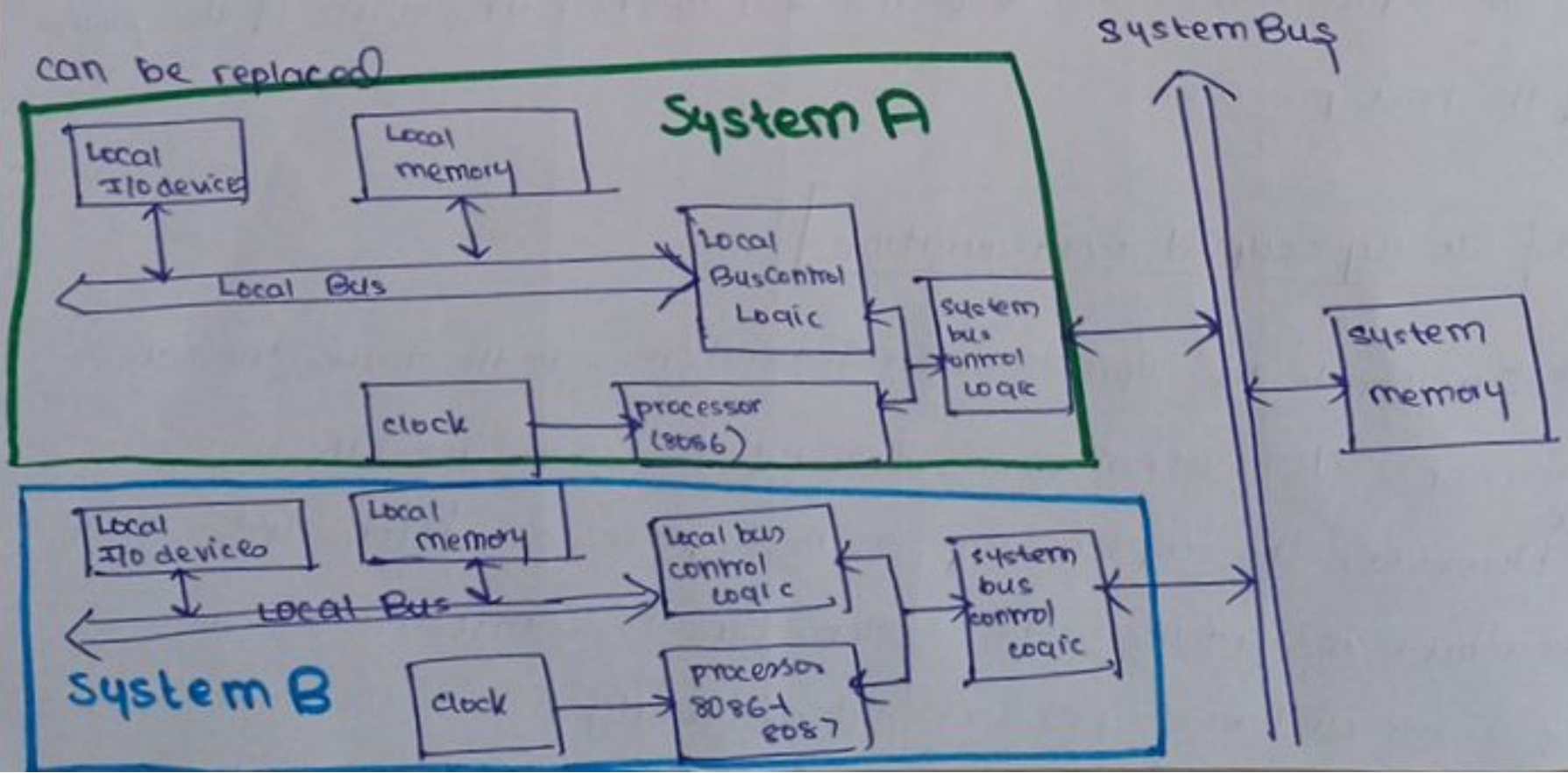
→ This results in an improved degree of concurrent processing

Advantages

→ increased efficiency

→ each processor has its own local bus - easy to achieve parallel processing.

→ flexible system structure - i.e the failure of one module doesn't cause the whole system to fail, the faulty module alone can be replaced



* Bus Allocation Schemes / Bus Arbitration

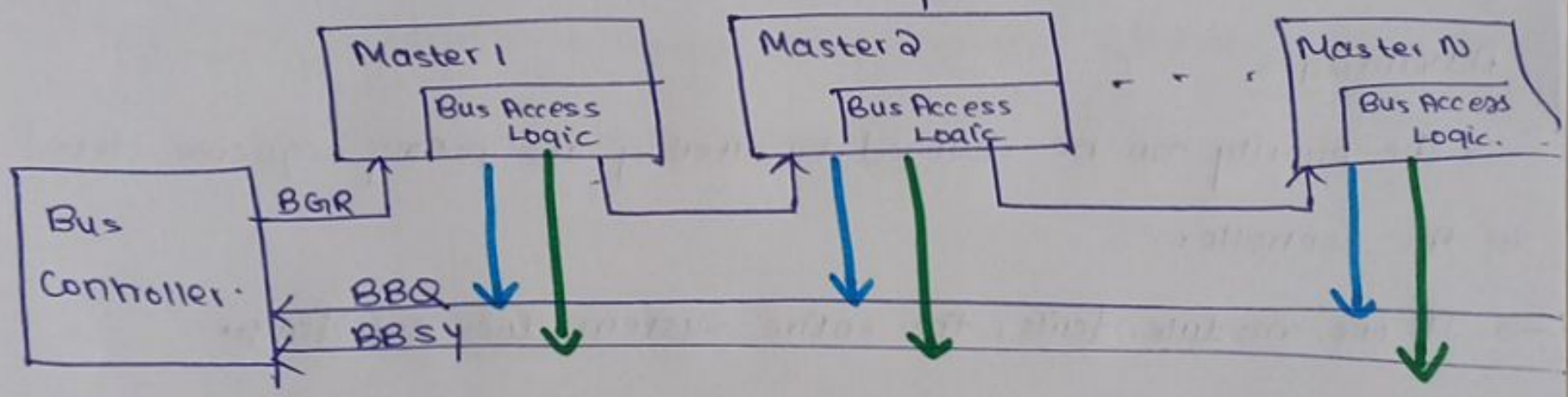
A. Daisy Chaining

→ There is a bus controller to monitor bus busy and bus request signals.

→ The bus controller sends a bus grant to a master, each master either keeps the service or passes it on.

→ The bus controller is responsible for synchronizing the clocks.

→ The master releases the bus busy signal when it is finished.



Advantages

(i) simple and cheap

(ii) requires the least no. of lines, which is independent of the number of masters in the system

Disadvantages

(i) propagation delay is proportional to the number of masters

(ii) priority of a master is fixed by its physical location

(iii) failure of one system causes the whole system to fail

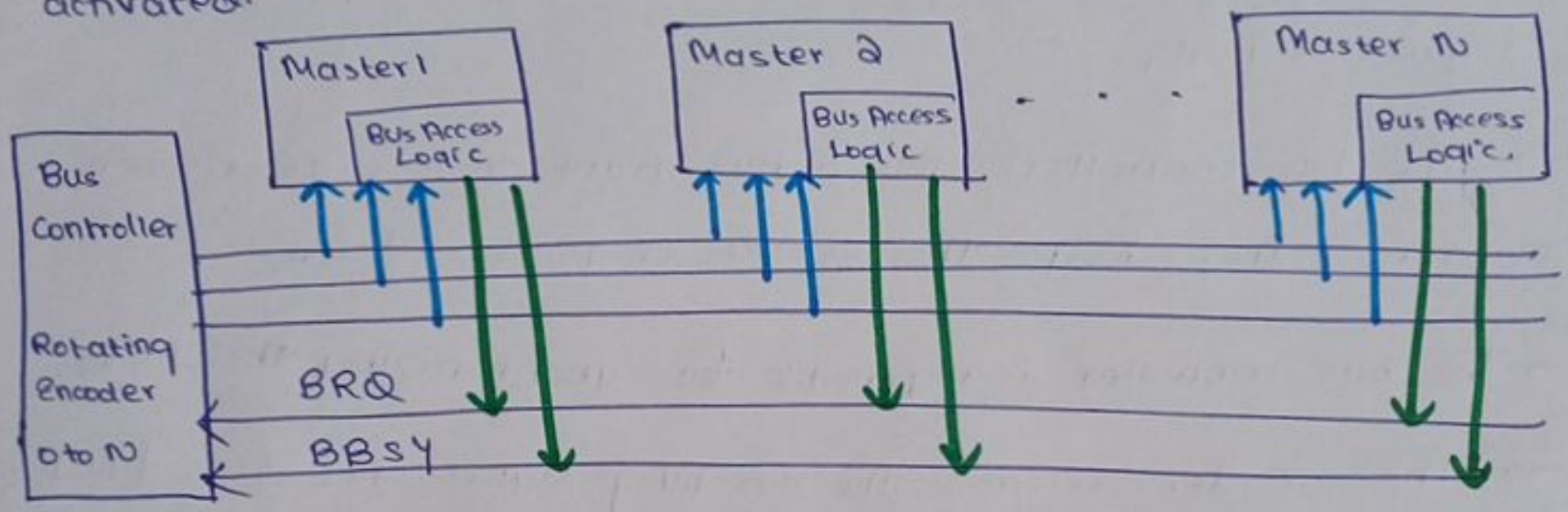
B. Polling

→ This method uses a set of lines sufficient to address each module.

→ In response to a bus request, the controller generates a sequence of module addresses.

→ When a requesting module recognizes its address, it activates the busy line and begins to use the bus.

→ The controller stops generating addresses when the busy line is activated.



Advantages

→ The priority can be changed by altering the polling sequence stored in the controller.

→ If one module fails, the entire system does not fail.

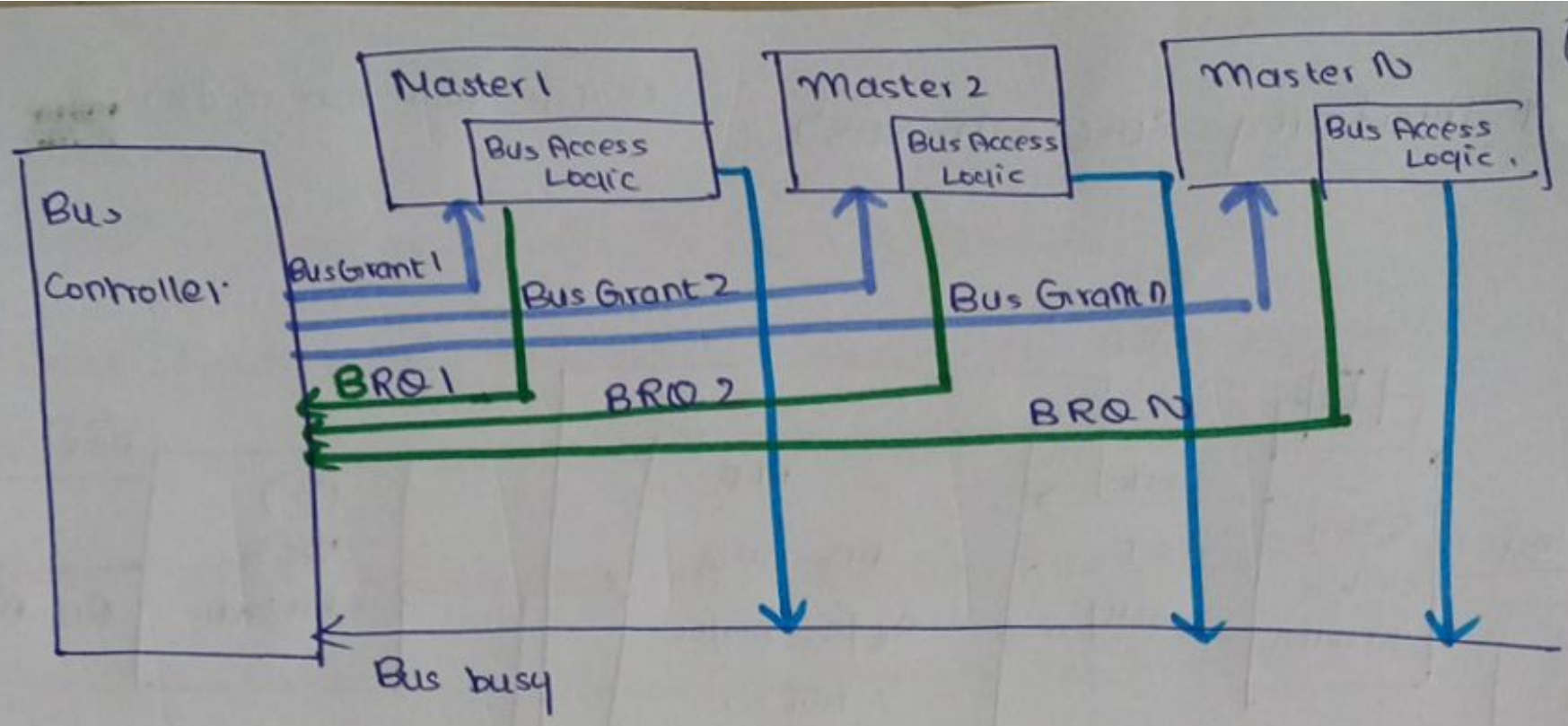
c. Independent Requesting

→ Each module has a separate pair of bus request and bus grant lines.

→ Each pair has a priority assigned to it.

→ The controller includes a priority decoder, which selects the request with the highest priority, and returns the corresponding grant signal.

→ The priority can be fixed priority or rotating priority.



Advantages – (i) since there are separate BRQ & BG lines, arbitration is fast and does not depend on the number of masters in the system

Disadvantages – requires a large number of lines (BRQ & BG lines) (2 * n signals for n modules)

* The 8087 Coprocessor

→ Using a general purpose microprocessor such as the 8086 to perform mathematical operations such as log, sin etc. is very time consuming for the CPU.

→ The 8087 is a math co-processor that can be interfaced with the 8086, to do floating point operations

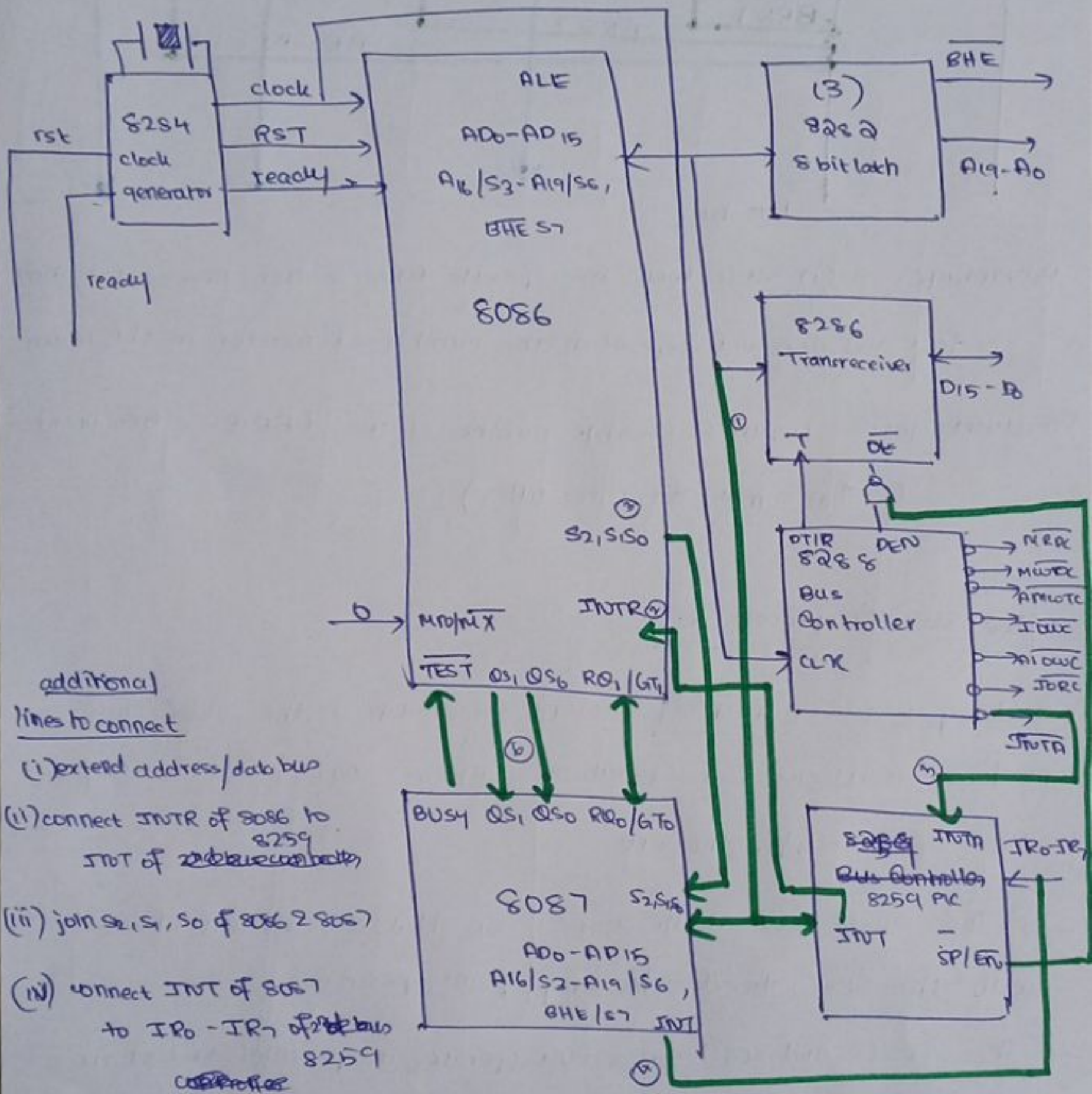
→ The 8086 and 8087 coprocessor operate in parallel and share buses and memory resources.

→ The 8086 marks 8087 floating pt. operations as ESC inst, so that the 8087 can execute them.

→ The 8086 operates in maximum mode when coupled with the 8087.

changes from max mode =

* Interfacing 8086 with 8087



additional lines to connect

(i) extend address/data bus

(ii) connect INTR of 8086 to INT of 8259

(iii) join S2, S1, S0 of 8086 & 8087

(iv) connect INT of 8087 to IR0 - IR7 of 8259

(v) connect $\overline{INTA}$ of 1st & 8259

(vi) connect TEST, QS1, QS0 & RQ0/GT0 pins

(vii) connect $\overline{SP}/\overline{EN}$ of 8259 to NAND gate of transreceiver

* Working

→ ESC is used as a prefix for 8087 instructions

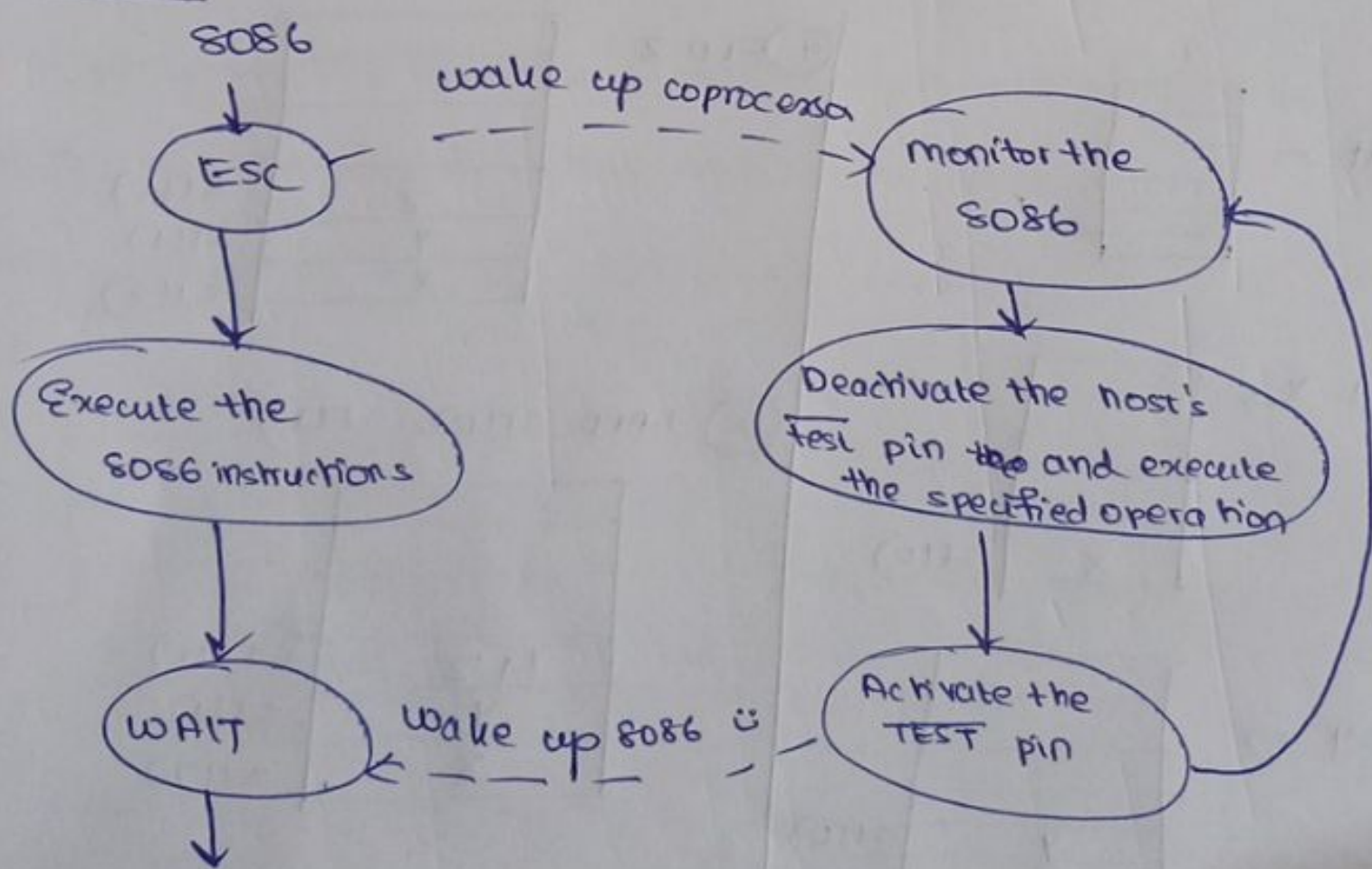
→ Once 8087 begins execution, it makes the BUSY o/p high, which is connected to the $\overline{TEST}$ pin of the μP.

→ While 8087 is executing its instruction, 8086 moves on with the next instruction. Hence multiprocessing takes place.

→ If 8086 requires the result of the 8087 operation, it first executes the wait state.

→ WAIT makes μP check the status of $\overline{TEST}$. If ~~to~~ $\overline{TEST}$ is high, μP enters the WAIT state. It comes out only when the 8087 has finished its execution.

* Flowchart



* Develop a MASM program for floating point addition using 8087 and illustrate the status of the stack after executing each instruction.

CAT-2 & End-Sem Q

Ans

finit → initialize stack

fld X → load X in ST(0)

fld Y → load Y in ST(0) (now ST(0)=Y and ST(1)=X)

fld Z → load Z in ST(0) (now ST(0)=Z, ST(1)=Y, ST(2)=X)

fadd ST(1) → add Y to Z & store result in ST(0)

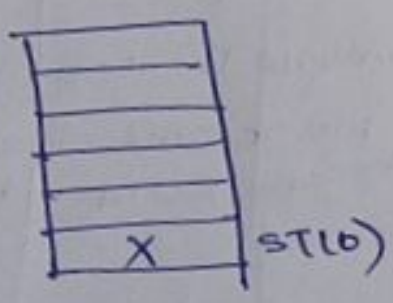
fadd ST(2) → add X to (Y+Z) and save in ST(0)

fst sum → store the sum in ST(0)

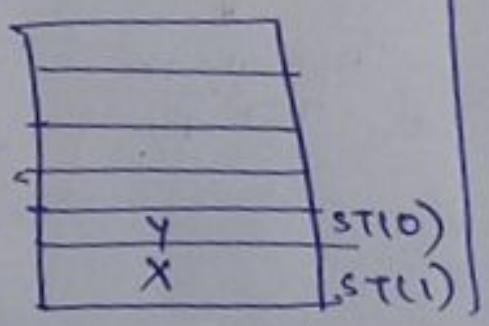
Stack

① finit ⇒

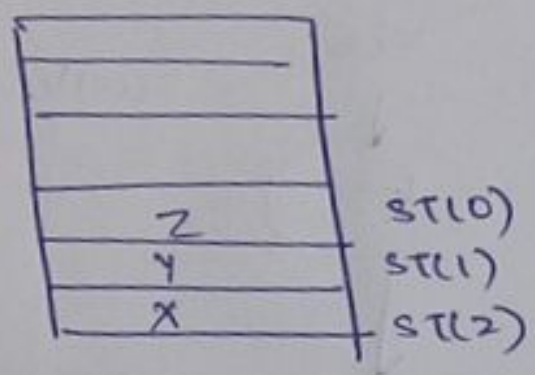
② fld X ⇒



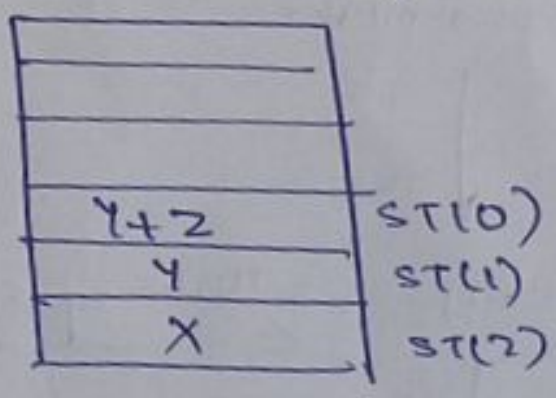
③ FLD Y ⇒



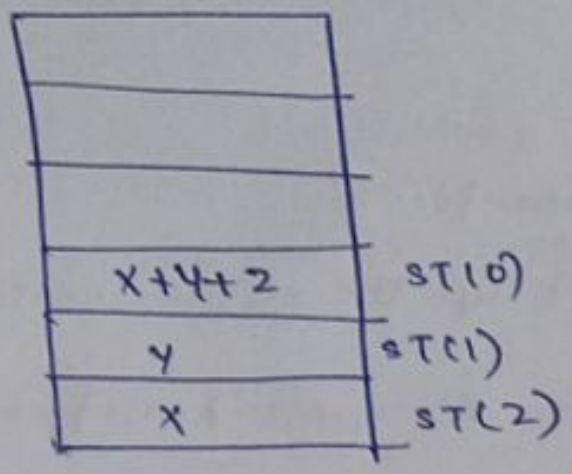
④ FLD Z



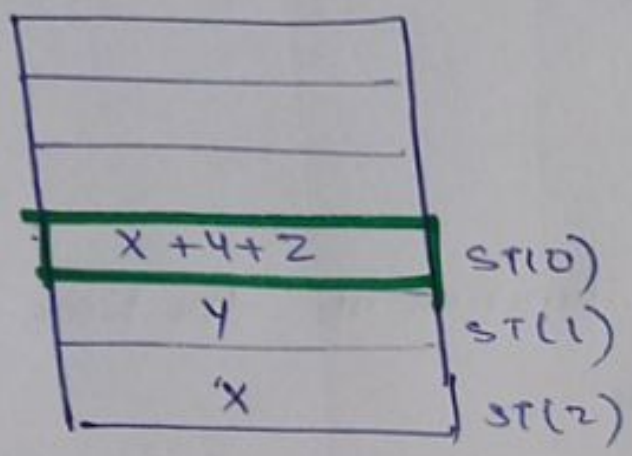
⑤ FADD ST(0), ST(1)



6 FADD ST(0), ST(2)



7 FST SUM

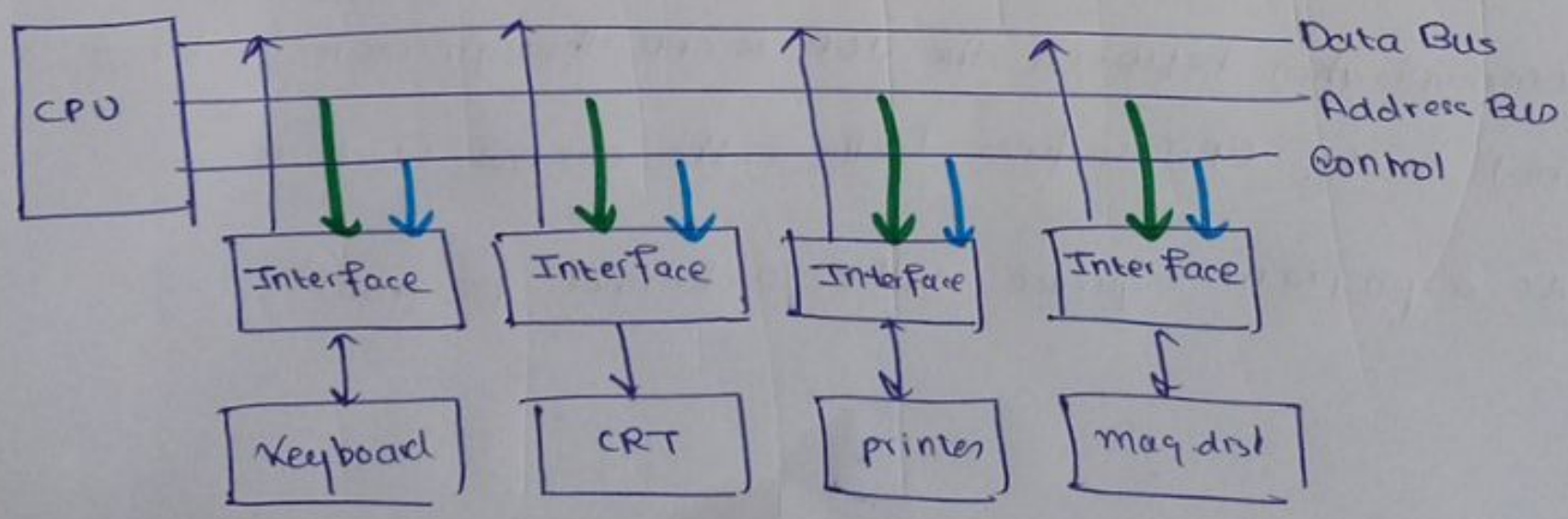


* 8089 - IO Processor

→ I/O processors handle all of the interactions between I/O devices and the CPU.

→ I/O processors communicate with input and output devices through separate address, data and control lines

→ This relieves the CPU of I/O device chores.



* I/O handled by microprocessor vs. I/O handled by IOP

By the microprocessor

→ by means of DMA - where data can be transferred

→ But the microprocessor still needs to set up the device controller, initiate the DMA operation and examine the post transfer status after the completion of each DMA operation.

I/O handled by the IOP

→ The μP can perform some other function at the time of I/O transfer. This increases the system speed

* Features of the 8089 IOP

→ can fetch and execute its own instructions

→ can also perform arithmetic, branching & logical instructions

→ The IOP does all the work in terms of device setup, programmed I/O & DMA.

→ The IOP can transfer data from an 8-bit source to a 16-bit destination & vice versa.

→ Communication between the IOP & CPU through memory-based control blocks. CPU defines tasks in the control block to locate a program sequence, called a channel program

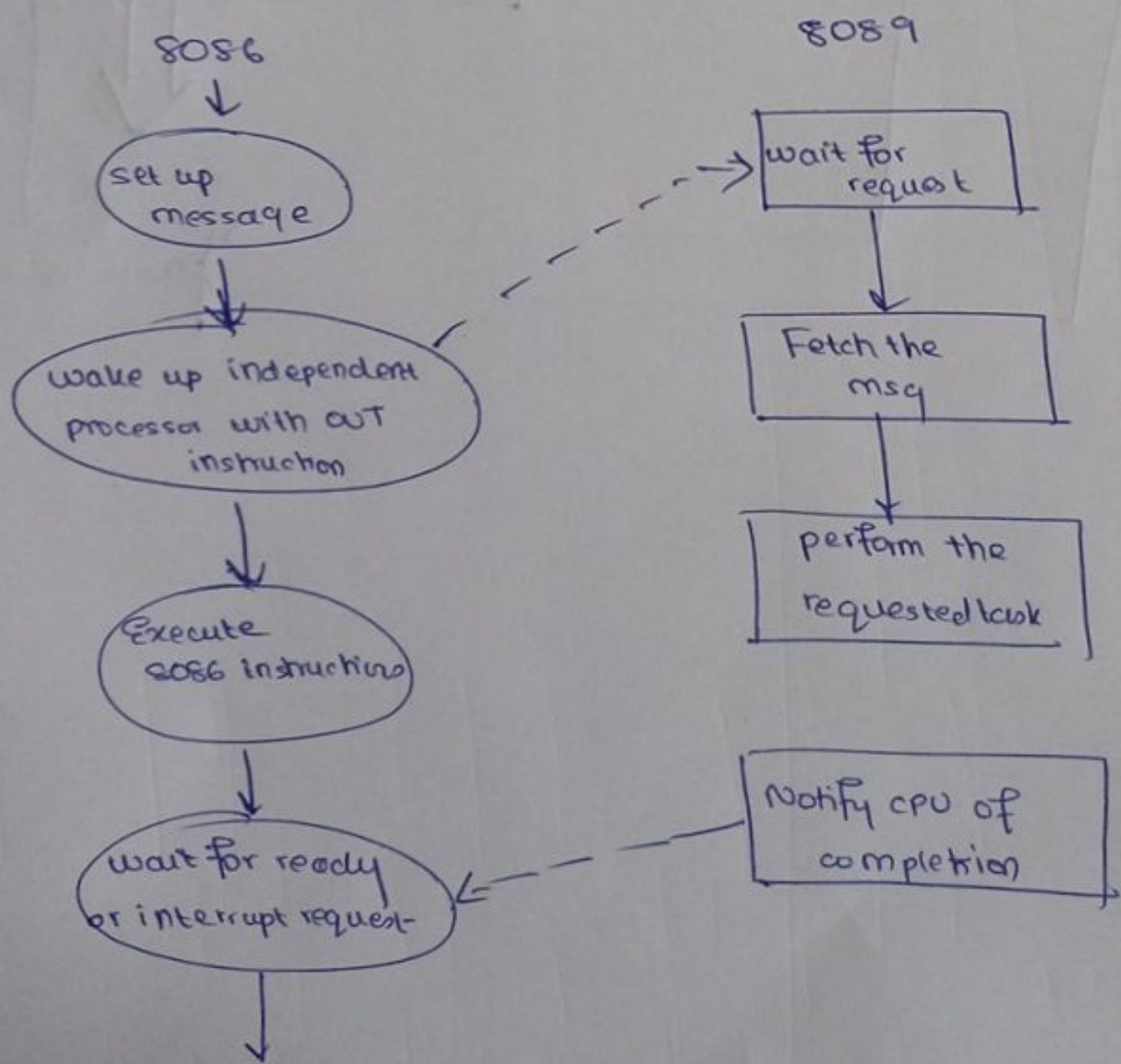
* Types of Commands

1. Block Transfer Commands - move block data to IOP - swap pages in and out of physical memory

2. Arithmetic, Logic & Branch Instructions - manipulate data so that the process time for the CPU is shortened

3. Control command - controls hardware - rewind the tape on a drive / eject CD

* Flow of Control



* Interfacing 8086 with 8089