8086		8085
20 bits	Address bus	16
16	Data bus	8
2^{20}	Main mem.	2^{16}

2^{10} - KB

2^{20} - MB

2^{30} - GB

2^{40} - TB

$2^{16} = 2^6 \times 2^{10}$

$= 64$ KB.

17/01/2023

chp.1 The Intel M.P. 8086 Architechture.

- 8086 Architechture
- Programmer's model.
- functional pin diagram
- Memory segmentation.
- Memory Banks.
- Demultiplexing of address and data bus.
- Functioning of 8086 in min. and max. mode.
- Timing diagrams (imp.) (Read, Write, iowrite, ioread)
- Interrupt structure and its serviding.

- <u>features of 8086 processor</u>:

→ It is an enhanced version of 8086 processor by Intel in 1976

→ 8086 processor supports 2 stages of pipelining - fetching and execution.

→ The memory is divided into 4 segments - core, data, stack, extra.

→ It has 64K i/o ports. 19/01/2023.

→ There are total 14 registers of 16 bits.

→ The enhanced version of instruction set (x & ÷ can be done by 8086)

→ It supports 2 operating modes - minimum (single m.p. will be operating) and maximum (suitable for multiple processors)

→ It has instruction queue capable of storing 6 bytes of instruction from memory resulting in faster processing.

→ It was the first 16 bit processor having 16 bit ALU, 16 bit registors and internal databus resulting in faster processing.
→ Available 3 versions depending on operating frequency – 5MHz, 8MHz, 10MHz.
→ Single fac phase clock with 33% of duty cycle.
→ It supports 256 vector interrupts.

- Campanson with 8086 processor of 8085.

Parameters	8085	8086
→ ALU size	8bits	16 bits
→ Address bus	16 bits	20 bits
→ Main mem.	2^{16}	2^{20}
→ Pipelining	doesn't ~~no~~ support pipelining	supports two stages of pipelining
→ prefetch	you will not findit	you will find 6 bytes of it
→ I/O	supports 2^8 ie 256 I/Os	2^{16}
→ cost	cheaper	costly

- 8086 Architechture:

Bus Interface unit:
→ segment reg.
→ Addition generation L.U.
→ IP
→ Prefecth ~~generation~~ queue

Functions
(1) Fetch Instruction
(2) Calculate Physical Address.
(3) Manage 6 bytes of prefecth Queue

Execution unit:

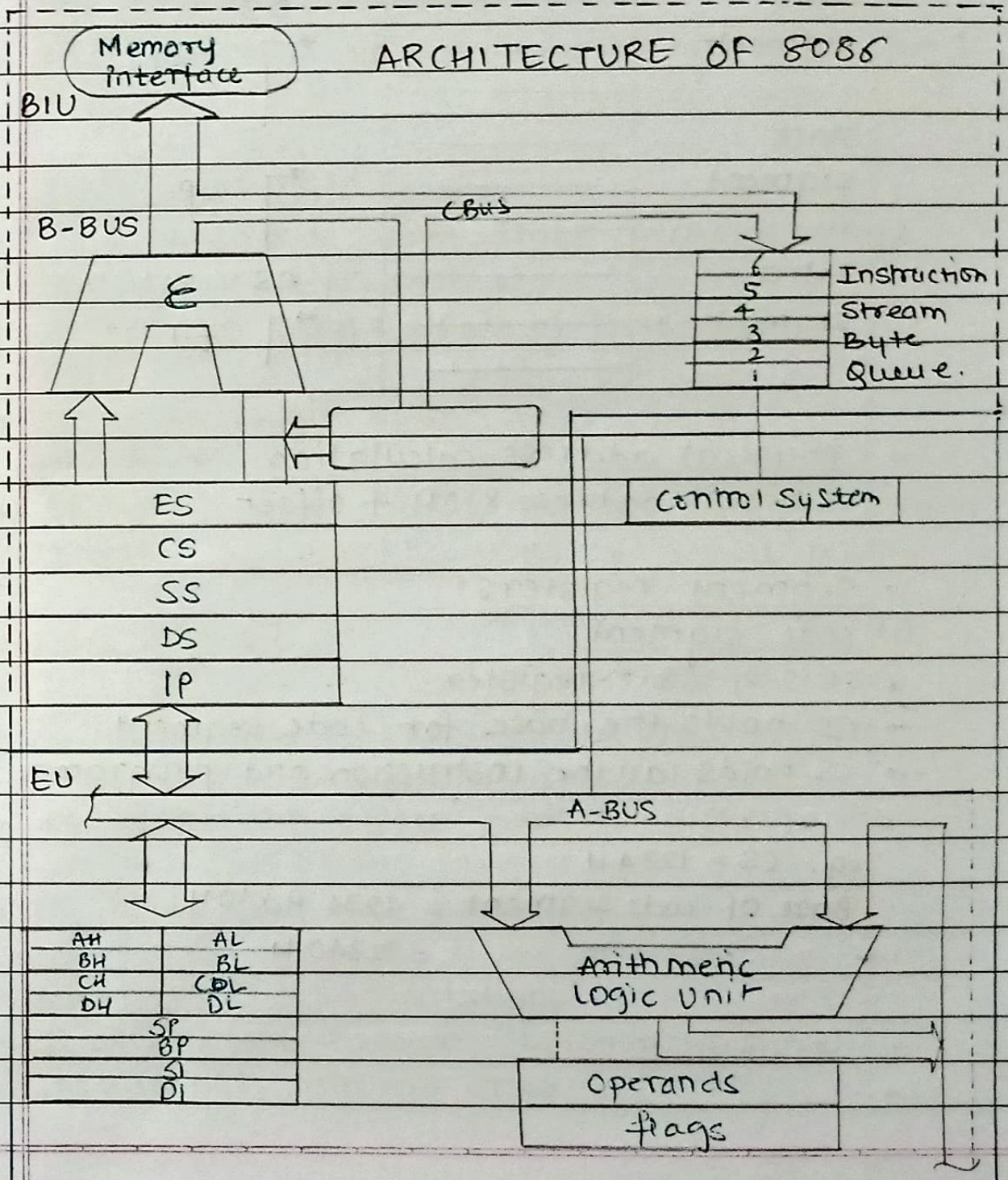
- general purpose reg.
- speed purpose reg.
- ALU
- status reg
- operand

Function

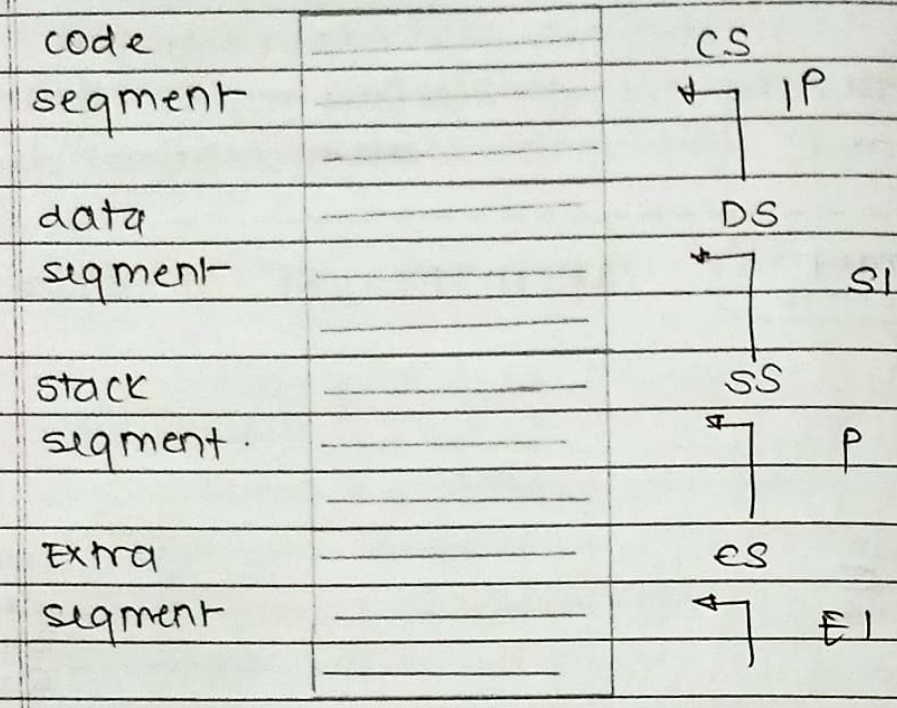
(1) Execute instruction.

(2) Perform Arithemetic/Logic /data transfer.

(3) Sind request to BIU to alter external devices.



- Memory segment:



- Physical address calculation:
  Segment address x10H + offset.

- Segment registers:

(1) Code segment:

→ It is of 16 bit register.
→ CS holds the base for code segment.
→ CS holds all the instructions and programs.

eg. CS = 1234H
Base of code segment = 1234 H x 10H
= 12340 H

(2) Data segment:

→ It is of 16 bit register.

→ This segment is used to hold the general data.

→ DS register holds 16 bits base address for this segment

→ SI register holds 16 bit offset address during the string operations.

→ BX register is used to hold 16 bit offset for this segment.

(3) Stack segment:

→ This segment holds stack memory which operates in LIFO manner.

→ SS register holds 16 bit of base address for stack.

→ SP holds 16 bit offset address of the top of the stack.

→ BP holds 16 bit offset address during random access.

(4) Extra segment.

→ This segment is used to hold general data.

→ Additionally, it is used as destination during string operations

→ DI holds the offset address during the string operation.

→ ES holds the base address

o Advanges of segmentation:

→ It divides the memory logically to store instructions, data and stack separately

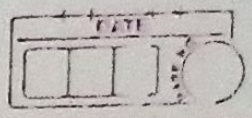
→ It permits the programmer to access 1MB mem using 16 bit of address

- Instruction pointer (IP):
→ It holds offset of the next instruction.
→ IP is incremented after every current instruction is fetched.
→ IP will get a new value whevever there is a branch instruction.

- Address Generation logic Unit (AGLU):
→ this generations the physical address.
→ Physical address = Sigment address X10H + Offset address.
→ Left address shifted by one.
and offset is added.

eg. CS = 1234H
IP = 0005H
calculate phisical address.

- 6Bytes prefetch Queue:
→ 6Byte FIFO RAM.
→ This fetches the next instruction while executing the current one.
→ This unit holds 6 instruction byte.
→ This prefetch queue is refilled when atleast 2 bytes are empty.
→ This pipelining increases the efficiency of up.
→



24/01/2023

- Execution Unit:
- Decodes and execute instructions.
- Perform arithmetic logic and internal data transfer operation.
- Send request signal to BIU to access the external module.
- It operates w.r.t. 'T states' (clock cycle).

Main components:

1. General purpose register
2. Special purpose registers.
3. ALU
4. Operand
5. Flag register.

(1) General Purpose Register:

- AX → AL, AH → Accumulator (default) (Holds operands during ×, ÷) (eg. MUL, BL → result.)
- BX → BL, BH → Memory pointer (only GPR that holds mem. add. in indirect addressing mode)
- CX → CL, CH → acts as default count in many register eg rotate & shift in. count CL reg. ~~loop and string in count CX reg.~~
- DX → DL, DH → acts as extension of accumulator when 32 bits are needing. eg. MUL, BX (AX × BX → 32 bits data.)

- 8086 has 16 bit G.P.R.s - AX, BX, CX, DX.
- These are available to user.
- Access is req. for storing the values during program execution.
- All these GPRs can be divided into two separate 8 bit registers eg. AX - AL & AH.

→ Apart from holding the data, GRRs have specific functions.

31/01/2023

(2) Special Purpose Registers. (16 bits).

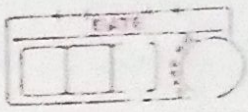
(a) SP It holds offset address of top of stack (Last of memory locations operating in LIFO manner). SP is used with stack segment register to calculate Physical address for stack segment. It is used during instructions like push, pop, CALL, RET instruction, etc.

(b) BP (Base Pointer (16bits)) : It can hold offset address of any location in the stack segment. It is used to access random locations of the stack.

(c) Source Index (SI) : It is normally used to hold the offset address of the data segment. But it can be also used for other segments using segment overriding. It holds offset address of the source data in the data segment during string operation.

(d) Destination Index (DI) : It is normally used to hold offset address for Extra segment but also can be used for other segments using segment overriding. It holds offset address of the destination of in extra segment during string operations

Any 2 flags would be asked to define for 2 marks.



(3) ALU:

→ 8086 supports 16 bit ALU

→ It performs 8 bit and 16 bit arithmetic & logical operations (enhancement - multiplication (MUL) & division (DIV) instructions).

→ control system: It is responsible to decode the I. which is already prefetched in prefetch Q. Generate appropriate internal control signals for execution purpose.

(4) Operand:

→ It is 16 bit register used by control register to hold the operands temporarily. (Exchange I.)

→ Not visible to the user.

(5) Flag register: 02/02/2023

Imp.

Control flags / control flag

93	X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

OF: Overflow flag
1 = overflow occured
0 = No overflow occured.

DF: Direction flag
1 = Auto Decrement
0 = Auto Increment

Interrupt flag: IF
1 = Enable Interrupt

TF: Trap flag
1 = Perform single stepping
0 = Do not perform single stepping

SF: Sign flag
1 = MSB of result is 1 (-ve)
0 = MSB of result is 0 (+ve)
(used for signed nos.)

ZF: Zero Flag
1 = Result = 0
0 = Result = 0

PF: Parity Flag
1 = Even Parity
0 = odd Parity

CF: Carry Flag
1 = Carry out of MSB
0 = No such carry

AF: Auxiliary Carry Flag
1 = Carry from lower Nibble to higher nibble
0 = No such carry
(used in 8-bit operation)

- Total 9 flags

Status flags (6)	Control Flags (3)
→ OF (f) → AF (c)	→ IF (g)
→ SF (e) → PF (b)	→ DF (h)
→ ZF (d) → CF (a)	→ TF (i)

(a) Carry Flag (CF): Whenever there is a carry (or borrow), the carry flag is set to 1 out of the MSB of the result.

(b) Parity Flag (PF): It is set if the ~~even~~ result has even parity.

(c) ~~Auxiliary Flag (AF)~~: ~~It is set if the~~ carry is generated out of the lower nibble. It is used only in 8 bits operations like DAA.

(d) Zero flag (ZF): This flag will be set if the result is 0.

(e) Sign flag (SF): It is set if MSB of result is 1. For signed operations, such a no. is treated as -ve.

(f) Overflow flag (OF): It will be set if the result of the signed operation is too large to fit in the no. of bits available to represent it. It can be checked using instruction 'INTO' – Interrupt on overflow.

eg. 31H + 23H = 54H ✓ (range: -80H to 7FH)

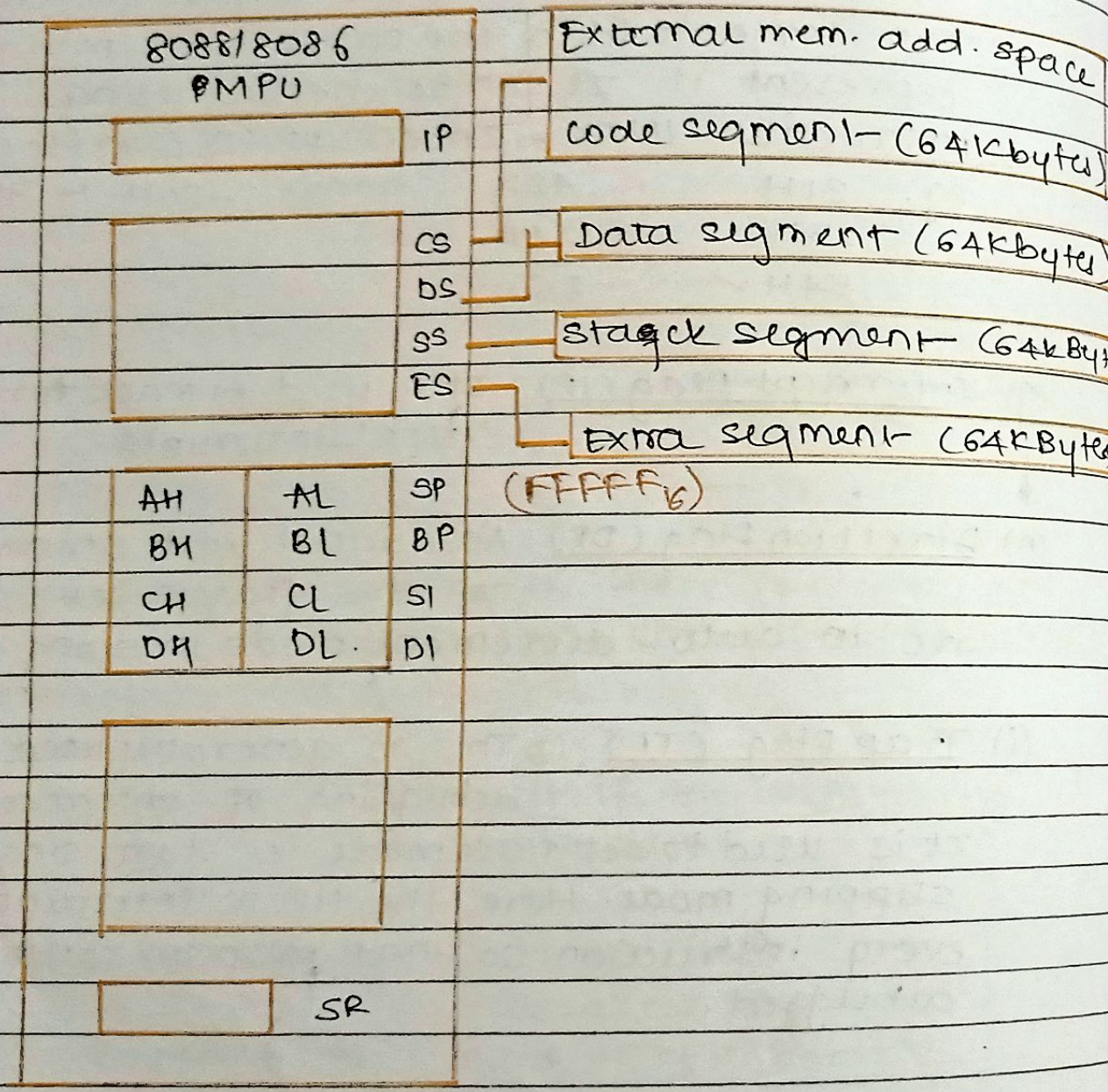
42H + 44H = 86H ✗

(g) Interrupt flag (IF): It is used to mask or disable 'INTR'-interrupts

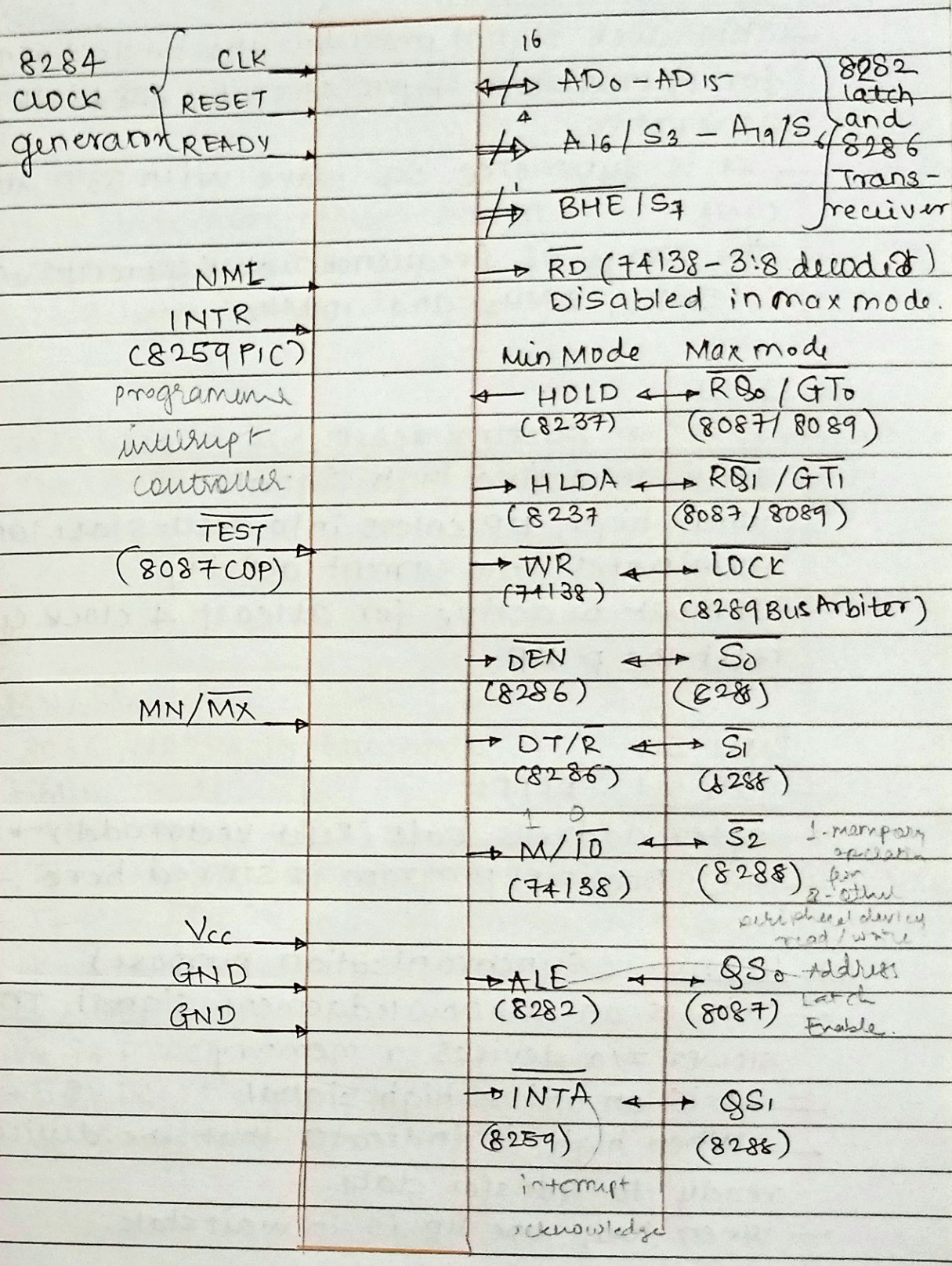
(h) Direction flag (DF): Associated with string operations. If this flag is set, SI and DI are in auto-decrement mode in string operations

(i) Trap flag (TF): This is generally used for debugging of programs. It is used to set trace mode ie. start single stepping mode. Here, the MP is interrupted after every instruction so that program can be debugged.

- Programmer's Model / Software model:
  All registers visible to the programmer
  - All GPRs
  - All segment registers
  - All offset registers (Sp. purpose)
  - Flag registers



07/02/2022


8284 clock generator
CLK
RESET
READY
NMI
INTR (8259 PIC) programmable interrupt controller
TEST (8087 COP)
MN/MX
Vcc
GND
GND
16
AD0 - AD15
A16/S3 - A19/S6
BHE/S7
8282 latch and 8286 Trans-receiver
RD (74138 - 3:8 decoder) Disabled in max mode.
Min Mode
Max mode
HOLD (8237)
RQ0/GT0 (8087/8089)
HLDA (8237)
RQ1/GT1 (8087/8089)
WR (74138)
LOCK (8289 Bus Arbiter)
DEN (8286)
S0 (8288)
DT/R (8286)
S1 (8288)
M/IO (74138)
S2 (8288)
1 - mem/port operation
0 - other peripheral device read/write
ALE (8282)
QS0 (8087)
Address Latch Enable
INTA (8259) interrupt acknowledge
QS1 (8288)

(1) CLK - clock signal.
- This clock signal provides the basic timing for operation of processor by 8284 clk generator
- It is symmetric sq. wave with 33% duty cycle.
- The range of frequency of different version is 5MHz, 8 MHz and 10 MHz.

(2) RESET :
- It is a system reset.
- It is an active high signal.
- When high, µp enters into reset state and terminates the current activity.
- It must be active for atleast 4 clock cycles to reset the µp.

Reset set :
- Code set - FFFFH
- after address calc (Reset vector add.) → FFFF0H
- BIOS (Booting) program is stored here.

(3) READY: (Synchronisation purpose)
- This is an acknowledgement signal from slower I/O devices or memory.
- It is an active high signal
- When high, it indicates that the device is ready to transfer data.
- When low, the µp is in wait state.

(4) NMI - (Non-Maskable Interrupt)
- It is an active high
- It is an edge triggered interrupt

(5) INTR:
- It is interrupt request signal.
- It is active high
- It is level triggered.

(6) $\overline{TEST}$
- It is used to test the status of math co-processor
- The $\overline{BUSY}$ pin of 8087 connected to this pin of 8086.

(7) MN/$\overline{MX}$
- 8086 works in two modes
  - Min. mode
  - Max. mode
- If MN/$\overline{MX}$ is high, it works in minimum mode
- If MN/$\overline{MX}$ is low, it works in maximum mode.

(8) V_{cc} & V_{ss}
- V_{cc} is power supply.
- +5V DC is supplied to this pin.
- V_{ss} is ground signal

(9) AD_0 to AD_{15} (Address & Data bus Multiplexed)
— These lines are multiplexed bi-directional add. / data bus.
— During T_1, they carry lower order 16-bit address.
— In remaining clock cycles, they carry 16 bit data.
— $AD_0 - AD_7$ carry lower order byte of data.
— $AD_8 - AD_{15}$ carry higher order byte of data.

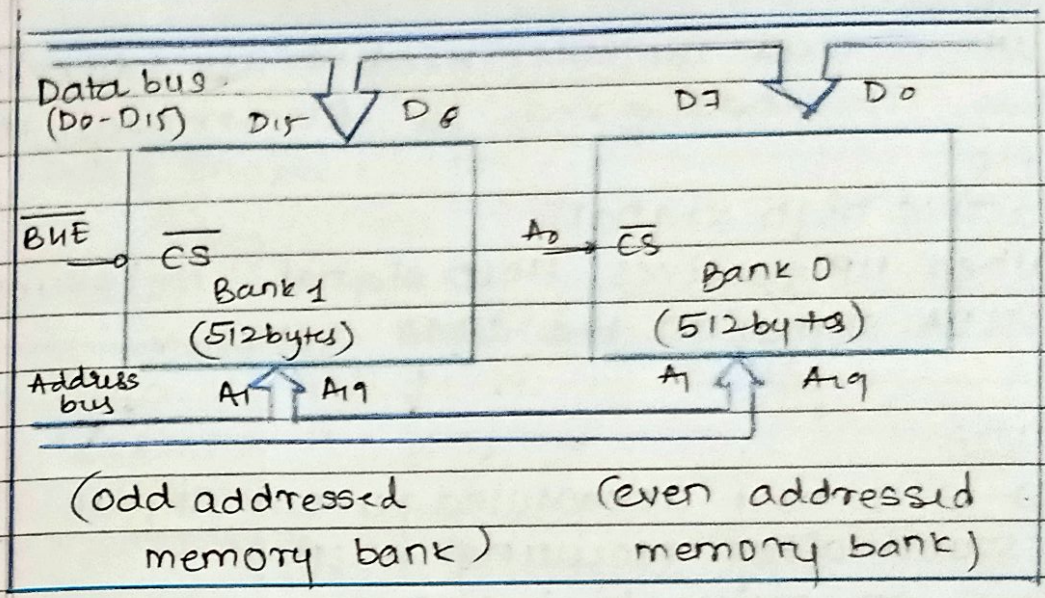
(10) A_{19}/S_6, A_{18}/S_5, A_{17}/S_4, A_{16}/S_3
— These lines are multiplexed undirectional address and status bus.
— During T_1, they carry higher order 4-bit address.
— In the remaining clock cycles, they carry status signals.

(11) $\overline{BHE}/S_7$.
— Bus High Enable
— used to indicate the transfer of data over higher order data bus ($D_8 - D_{15}$)
— 8 bits I/O devices use this signal.
— It is multiplexed with status pin S_7

memory segmentation, advantages, word

14/02/23.

Memory Banking:



0	0	Read or write 16 bit from both banks
0	1	" 8 bits from higher bank
1	0	" " lower bank
1	1	No operation (Processor is idle)

eg. MOV BL, [2000H]
MOV BX, [2000H] (aligned operation)
MOV BH, [2001H]
MOV BX, [2001H] (misaligned operation)

MOV BL [2000H]

→ 8 bit operation
→ BL will get value stored at ~~content of~~ offset address (2000H)
→ only one bank is ~~ste~~ selected (here lower/even)
→ $\overline{BHE} \rightarrow 1$, $A_0 \rightarrow 0$.

MOV BX, [2000H] (aligned)

→ 16 bits
→ BX will get values ~~of~~ stored at both 2000H & 2001H.
→ $\overline{BHE} \rightarrow 0$, $A_0 \rightarrow 0$
→ Both banks will be selected.

Pin configuration for Min Mode.

(1) HOLD
→ When DMA controller needs to use address bus, it sends a req. to CPU through this pin.
→ active high signal.
→ When up receives HOLD signal, it issues HLDA signal to the DMA controller.

(2) HLDA
→ It is a hold acknowledgement signal
→ Issued after receiving HOLD.
→ It is an active high signal.

(3) WR
→ Write signal
→ Used to write data in memory or output device depending upon the status of M/IO signal.
→ Active low signal.

(4) M/IO
→ The signal is issued by the up to distinguish memory access from I/O access.
→ When it is high memory is accessed
→ When low, I/O devices are accessed.

(5) DT/R
→ Data Transmit / receive signal (to processor)
→ It divides the direction of data flow through the transceiver.

→ High → data is transmitted out
→ Low → data is received

(6) $\overline{DEN}$
→ Data Enable signal.

(7) $\overline{INTA}$
→ Interrupt acknowledge signal
→ When µP receives INTA signal, it acknowledges the interrupt by generating this signal.
→ Active low signal.

(8) ALE
→ Address latch Enable
→ Indicates that valid address on bus $AD_0 - AD_7$
→ Connected to enable pin of latch 8282.

Pin configuration for MIN Mode.

(1) HOLD
→ When DMA controller needs to use address/data bus, it sends a req. to CPU through this pin.
→ active high signal.
→ When MP receives HOLD signal, it issues HLDA signal to the DMA controller.

(2) HLDA
→ It is a hold acknowledgement signal
→ Issued after receiving HOLD.
→ It is an active high signal.

(3) $\overline{WR}$
→ Write signal
→ Used to write data in memory or output device depending upon the status of M/IO signal.
→ Active low signal.

(4) M/$\overline{IO}$
→ The signal is issued by the MP to distinguish memory access from I/O access.
→ When it is high memory is accessed
→ When low, I/O devices are accessed.

(5) DT/$\overline{R}$
→ Data Transmit / Receive signal (to processor)
→ It decides the direction of data flow through the transceiver.

→ High → data is transmitted out
→ LOW → data is received

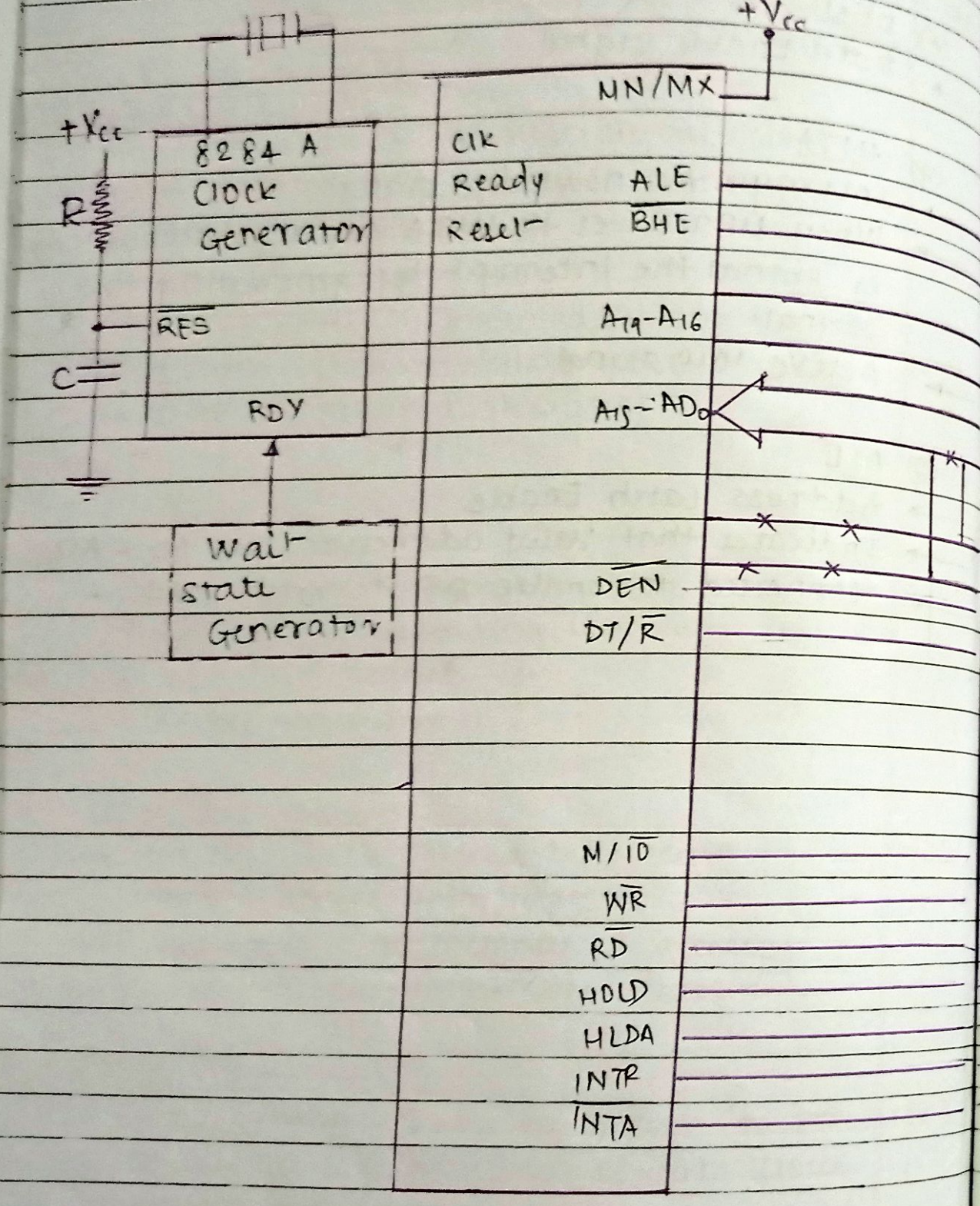
(6) $\overline{DEN}$
→ Data Enable signal.

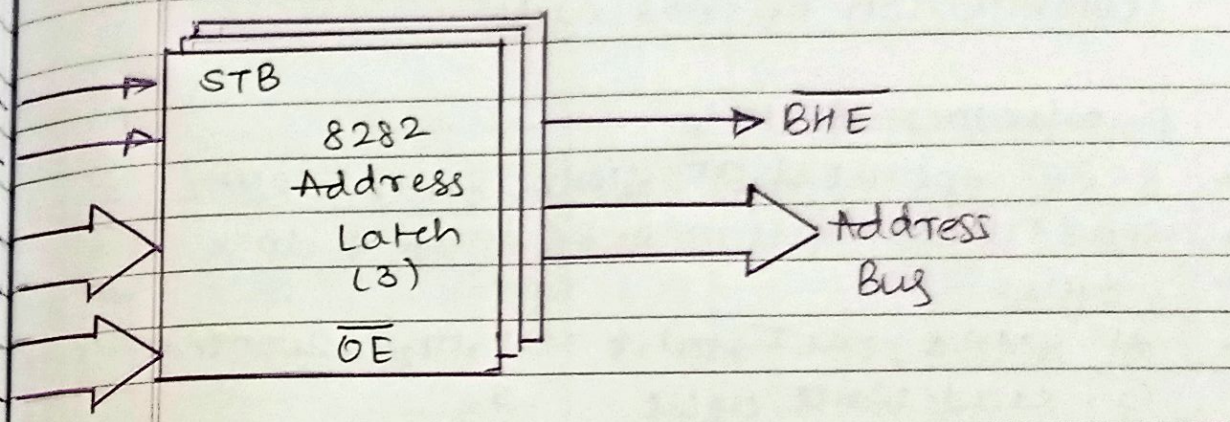
(7) $\overline{INTA}$.
→ Interrupt Acknowledge signal
→ When µP receives ~~IR~~ INTA signal, it acknowledges ~~signal~~ the interrupt by generating this signal.
→ Active low signal.

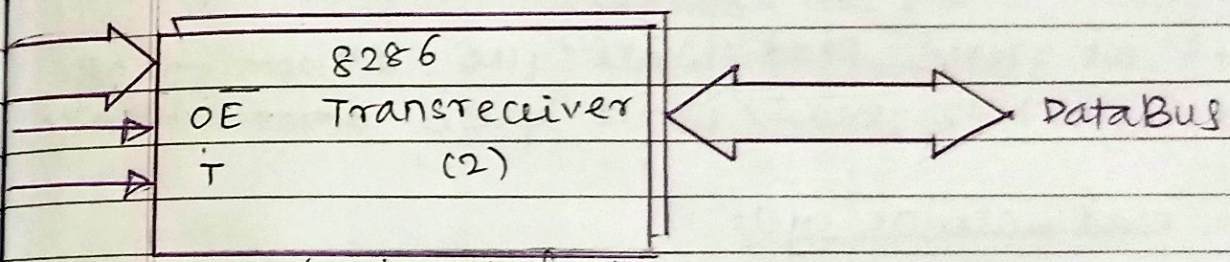
(8) ALE
→ Address Latch Enable
→ Indicates that valid addresson bus $AD_0 - AD_{15}$
→ connected to enable pin of latch 8282.

16/02/23

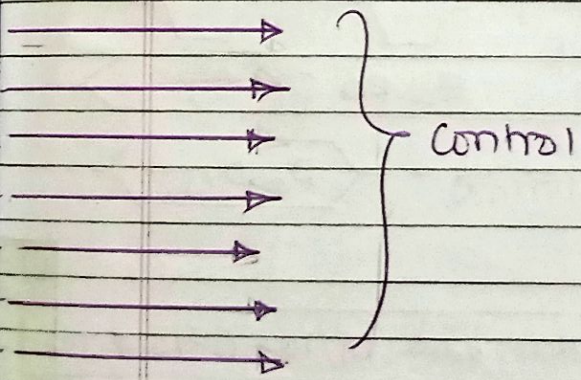
• Minimum Mode Configuration







(Optional for increased Data bus drive)



Timing diagrams: (Terms)

- T-State = 1 / clock frequency
- Machine or Bus cycle = Time to execute one Read or write cycle
- Instruction cycle = Total time req. to fetch and Execute one Instruction (combination of Bus cycle.

Significance of clock

→ 8086 operates at 5MHz, 8MHz & 10MHz

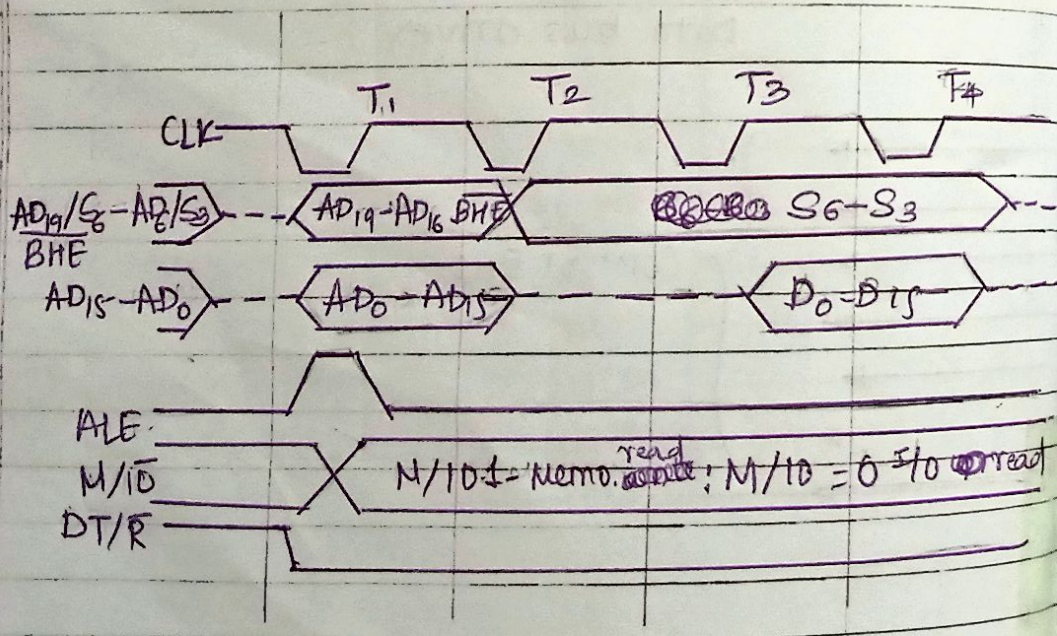
→ Read/write cycle in 8086 is of 4-clock cycles

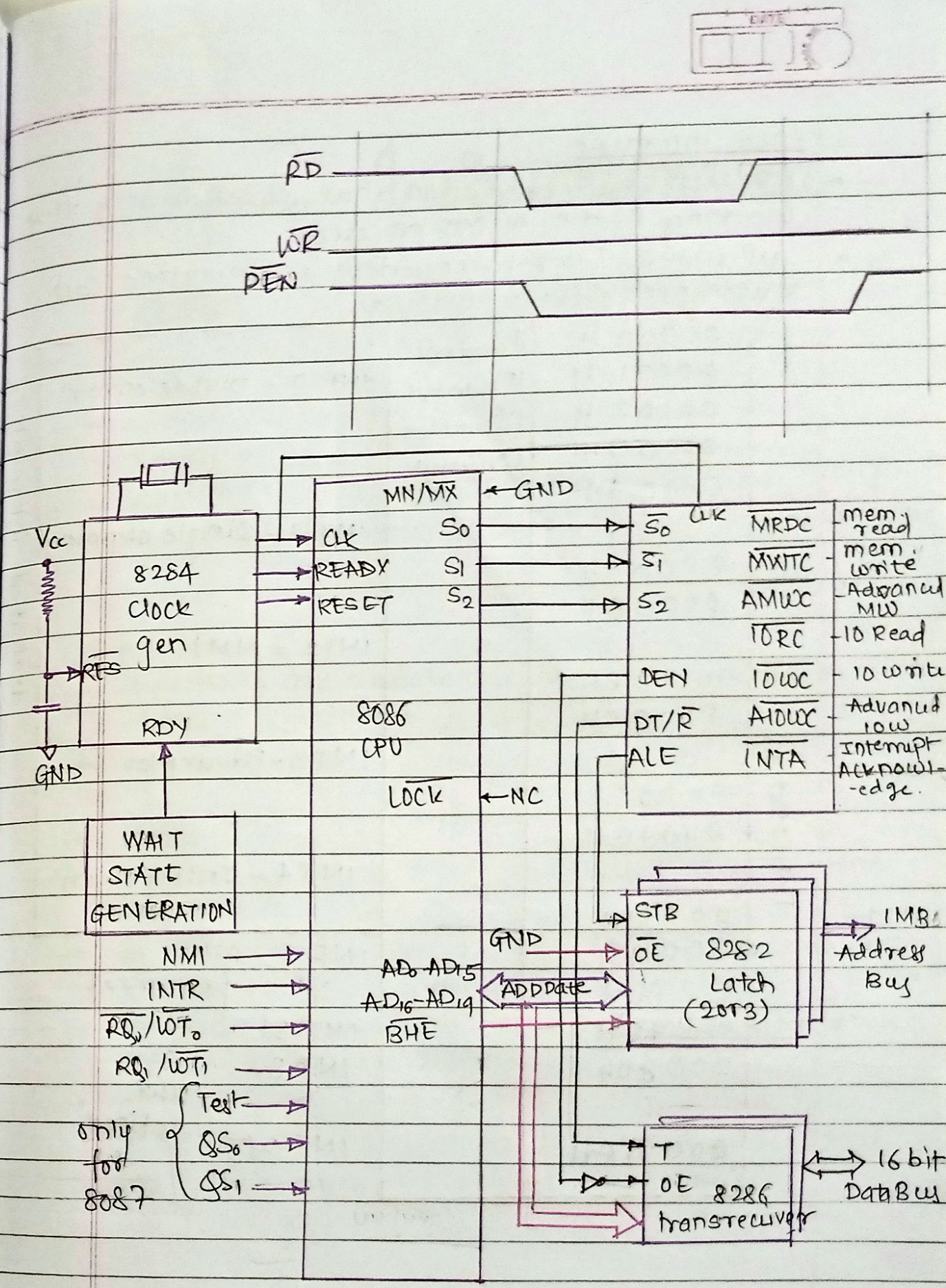
→ At 5MHz, one T states = $\frac{1}{5}$ MHz = 200nsec, so Read/Write cycle = 4 x 200 = 800nsec.

→ At 8MHz, Read/Write cycle = 4 * 125 = 500 nsec

→ At 10 MHz, Read/write cycle = 4 * 100 = 400 nsec

- Read Machine cycle

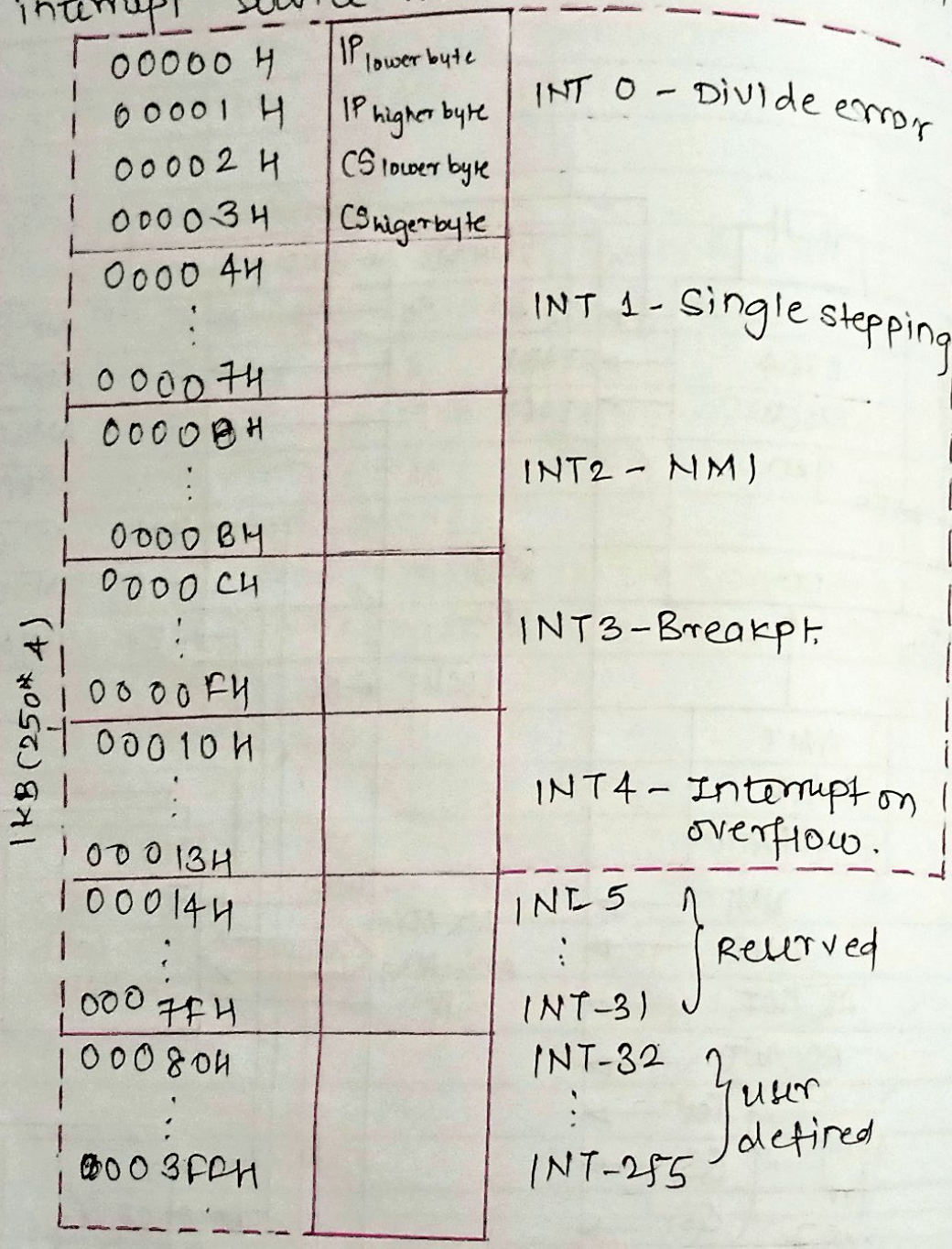



RD
WR
DEN
MN/MX
GND
Vcc
8284
clock
gen
CLK
READY
RESET
RES
RDY
GND
8086
CPU
LOCK
NC
S0
S1
S2
CLK
DEN
DT/R
ALE
MRDC
MWTC
AMWC
IORC
IOWC
AIOWC
INTA
mem read
mem write
Advanced MW
IO Read
IO write
Advanced IOW
Interrupt Acknowl-edge
WAIT STATE GENERATION
NMI
INTR
RQ0/WT0
RQ1/WT1
only for 8087
Test
QS0
QS1
AD0-AD15
A16-A19
BHE
ADDRESS
GND
STB
OE
8282
Latch
(2 of 3)
IMB
Address Bus
T
OE
8286
transceiver
16 bit
Data Bus

o 8086 interrupt

→ special case / condition that arises during the working of the micro processor.

→ µP service it by executing a subroutine call interrupt service routine.



chp.2 Instruction set and Programming

- Addressing modes: (Memory)

(1) Immediate A.M.:

→ Operand (8bits & 16bits) is specified in the instruction itself.

→ eg. MOV CL, 0AH – Move 0AH into the lower byte of CL.

(2) Register A.M.:

→ Operands are registers ie. they are specified through registers

→ eg. MOV CL, DL.

(3) Direct A.M.:

→ Address of the operand is directly given in the instruction.

→ eg. MOV CL, [2000H] (Offset address) - Moves data from 2000H (16 bits → 20 bits) in the data segment CL.

physical address = DS * 10H + 2000H (offset address)

PA = 50000+2000 = 52000

→ eg. MOV CX, [2000H]

CX - 16 bits, PA will be calculated twice, once for lower byte and 2nd for upper byte.

(4) Indirect A.M:

→ 1. Register I.A.M:
Address of the operand is given using a register.
The address should be in register as operand.
BIU calculates the P.A.

2. Register Relative addressing mode:
Address of the operand ~~by~~ is given by sum of register and displacement
eg. MOV CL, [BX+4] (constant) or MOV, 04H [BX].
↓
'Relative'

3. Base Indexed A.M:
(SI/DI)
BX - Data seg., BP - stack seg..
Only BX GPR's used for ~~atleast~~ address along with offset address registers.
segment to be operated by segment reg. and not by index register.

BX + SI
BX + DI
BP + SI
BP + DI.

eg. MOV CL, [BX+SI]
MOV CL, [BP+SI]

(4) Base Relative plus Indexed Addressing mode

eg. MOV CL, [BX + DI + 20]

MOV [BP + SI + 1000], CL.

(5) Implied A.M.

→ operands are implied, not specified in instructions.

eg. STC - set the carry flag

CLD - clears the direction flag.

- I/O Addressing Modes.

Direct	Indirect
→ 8 bits	→ 16 bits.
→ 00H to FFH	→ 0000H to FFFFH
→ 256 I/O port Address	→ 65536 I/O port Address
eg. IN AL, 47H	eg. MOV DX, 2000H IN AL, DX

port address number

move DX data to AL

AL gets data from I/O port add. 2000H given by DX.

- Assembler directives.

→ It is a statement / reserved keyword to give direction to the assembler to perform task of the assembly process.

→ Indicate how an operand or section of a program is to be processed by the assembler

→ controls the organization of the program.
→ Provides necessary info. to the assembler to understand assembly lang. programs to generate necessary machine codes.
→ An assembler supports directives to define data, to organize segments to control procedures.

(1) .MODEL: defines mem. model.
   i- SMALL
   ii- MEDIUM
   iii- LARGE
(2) . DATA.
(3) . CODE - start code segment.
(4) . STACK - start ~~stg~~ stack segment.
(5) . DB - define byte
(6) . DW. - define word
(7) SEGMENT, ENDS - end of the segment.
   L beginning of code/data/stack/extra segment.
(8) ASSUME - informs name of program or data segment that should be used for a specific segment.
   eg. ASSUME CS CODE, DS DATA.
(9) ORG: (origin) used to assign the starting address for program/data in segment.
(10) END
(11) EVEN - even address start
(12) PROC - indicates beginning of a proccdure
(13) ENDP - End of procedure
(14) FAR - Intersegment call
(15) NEAR - Intrasegment call

(16) AT: instruct the assembler to start the segment at
5000H

- 8086 Instruction set:
  - Data Transfer
  - Arithmetic
  - Logical
  - String
  - Control transfer / Shift Rotate.
  - Processor control