SSI → 10-100 transistor, gates, FF, latch

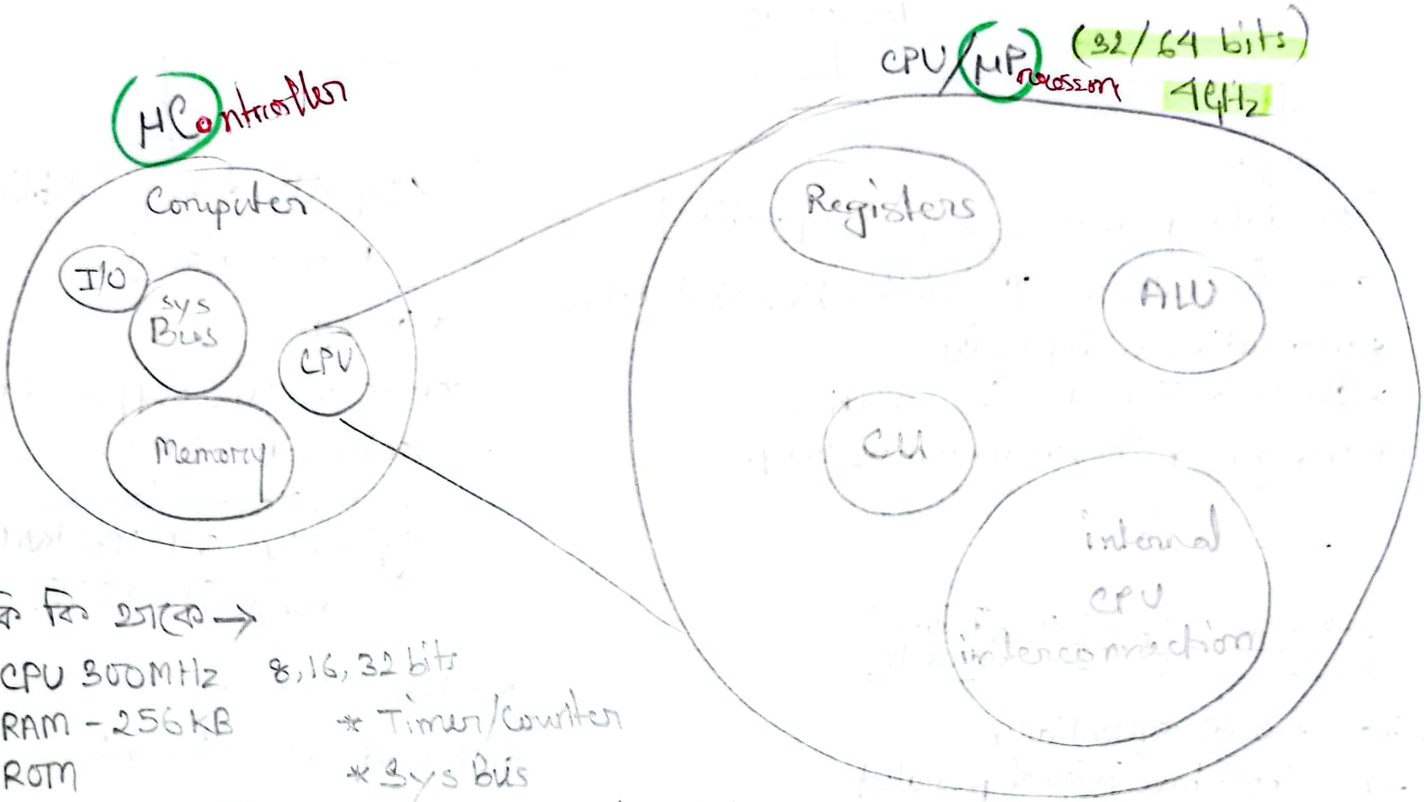
MSI → counter, mux, adder, ALU.

μp → Intel 4004 (first, 4 bit μP, CPU on a chip.

μC → Intel 8048, CPU, RAM, ROM BUS etc. on a chip.

VLSI → 20K - 50K transistors, DSP, RISC, 16/32 bit μp

ULSI → 750k transiston, 64bit μp.



কি কি থাকে →

* CPU 300MHz   8, 16, 32 bits
* RAM - 256KB
* ROM
* Timing
* I/O ports
* Timer/Counter
* Sys bus
* Serial port/Serial interface

4004 - 4bit
4040
8008
8080 - 8bits
8085
8086 - 16 bits, 1MB } sametype
8088
80286 - 16 bits   16MB
80386 - 32 "   4GM
80486.
Pentium (P5)
RISC
Itanium (P7)

smol cost, size,
low power consumption
(high) reliability
versatility.

Embedded System
[automation]

Memory

16bit MP: 8086, 80286, 80386SL → 2banks → low bank 8bit (even address থাকে) D_0-D_7
→ high " 8bit (odd " ") D_8-D_{15}.
bank 2x8 = 16 bits.

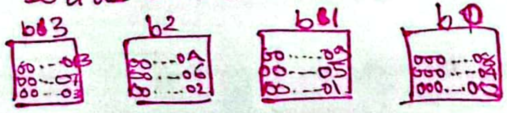
32bit: 80386DX, 80486 → 4 banks (4GB) each 8bits (1GB) banks 4x8bit = 32bits

64bit: Pentium, Pro → 8 banks. (8GB) → banks 8x8 = 64 bits

8088 → 8bits (1bank - 8bits)

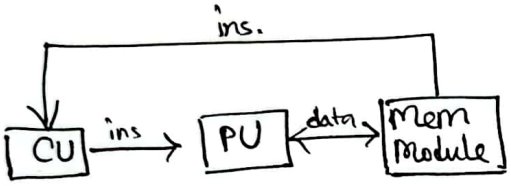
** Moore's law (1965) → cramming more components onto IC → transistors 4x in 3 years (DRAM rule)
↓
এটা কাজে দেয় না এখন আর

** bank-এর math – যত সংখ্যক bank তা দিয়ে mod করব আর serial wise bank-এ data রাখব.



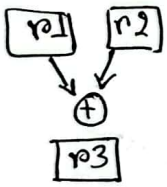
Processores

Single
(1/2 data-ই কাজ করে) [1 operation]
SISD formulation Single instruction on single data.
* instruction decoded by CU.
* CU sends it to PU. for execution
* Old gen, mini, workstation comp.



* ins. execute sequentially
- may/not have internal parallel processing capabilities
* can be achieved by multiple fn units or pipelining.

Scalar ISA



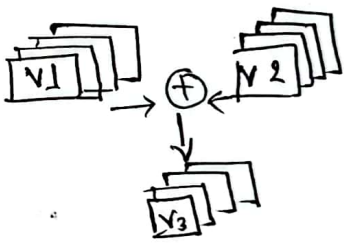
Add.d r3, r1, r2.

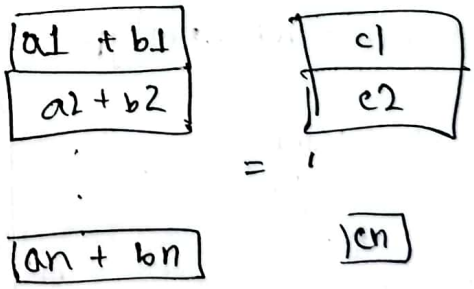
a1 + b1 = c3
a2 + b2 = c3
:
an + bn = cn

for i=1 to n
c[i] = a[i] + b[i]
end

Vector.
(Exploits data parallelism)
[N operation]
* arrays.
* max. vector supported by vector ISA → MVL
max. vector length.
typically = 64-128 for MVL
range 64-4096.

Vector ISA.
add v.d v3, v1, v2.





c[1:n] = a[1:n] + b[1:n]

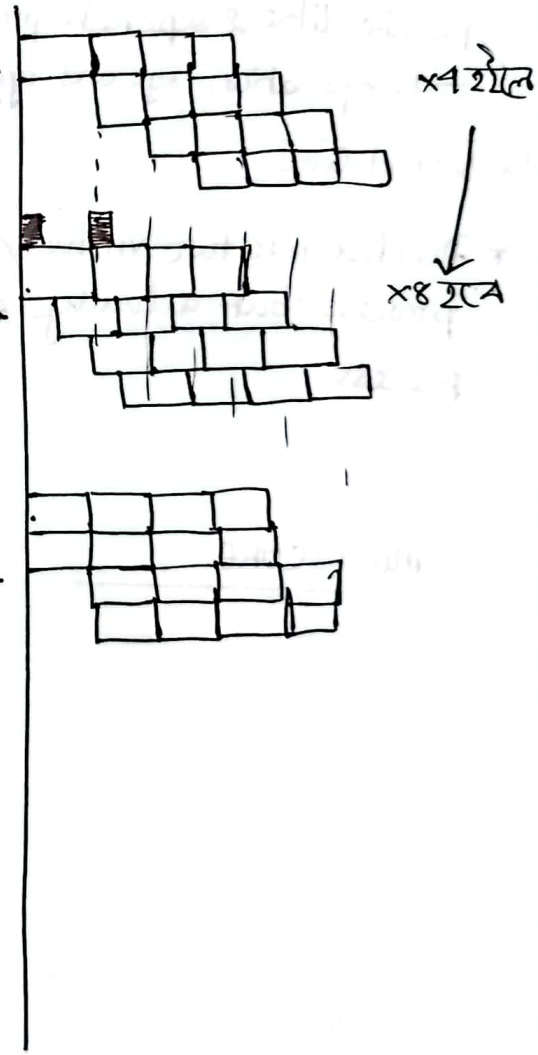
CISC (assembly) VS RISC, (mips)

* Hardware
* multiple ins of sizes + formats.
* less registers needed.
* More addressing modes
* extensive use of microprogramming
* ins. take varying amount of cycle
* pipeling → difficult

* Software
* some set with few formats
* more registers.
* fewer addressing mode
* complexity in compiler
* ins. take 1 cycle.
* easier → pipelining.

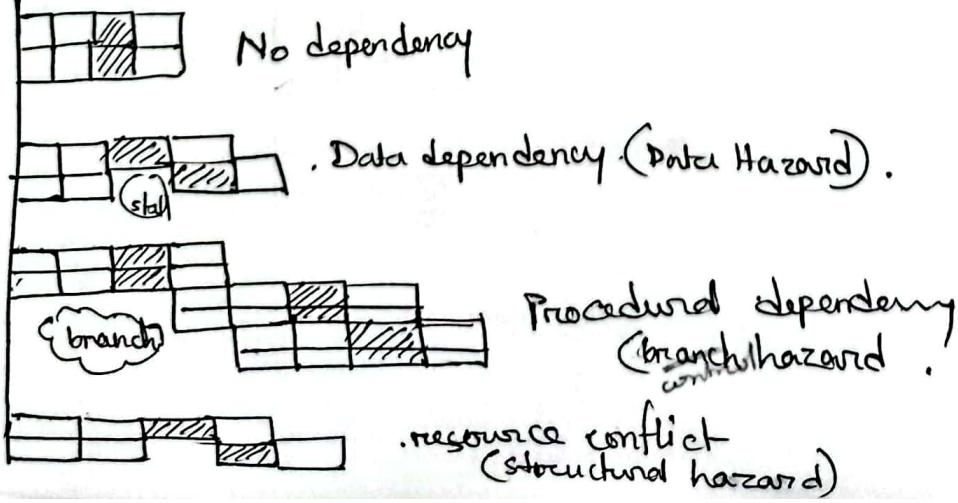
pipline: one p.stage at a clock cycle.

Superpipline: 2 p.stage at a clock cycle
Increases the level of parallelism. by more than 1 ins. being in pipline at the same time.
fn splits into 2 non overlapping parts and each executes in half a clock cycle



Superscalar processors: multiple fn unit is manipulated
maintain parallel pipline and perform one pipln stage at one clok cycle

Problems:



Solutions (PTO)

SOLUTIONS:

20 stages

- Multi-Core Processor
- Out-of-Order Processor
- SUPERSCALAR PROCESSOR
- DEEP-Pipelined Processor
- PIPELINED Processor
- SINGLE Cycle-processor

(Intel) Hyper-Threading techno: * better for multi-tasking.
* do more, wait less.
* like freeway with multiple car lanes instead just 1.

Multicore

→ *2 threads per core — 4 core sys. allow 8 threads.

HYPER THREADING

*Allows a single processor to operate like 2 seperate processors.
— to operation sys and app prog.

*2 virtual Cores

* Divided into two virtual/logical process but actually a physical process.

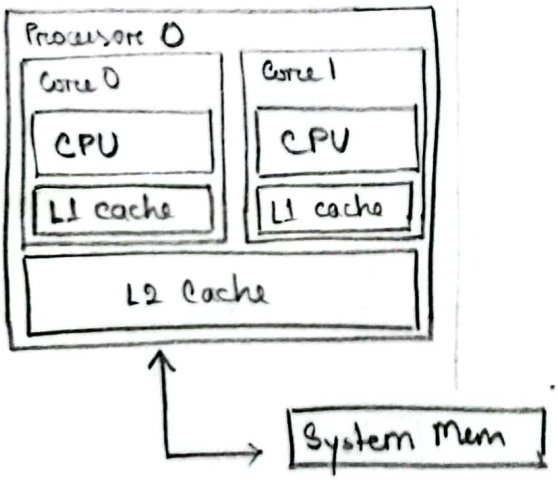
VS

MULTITHREADING

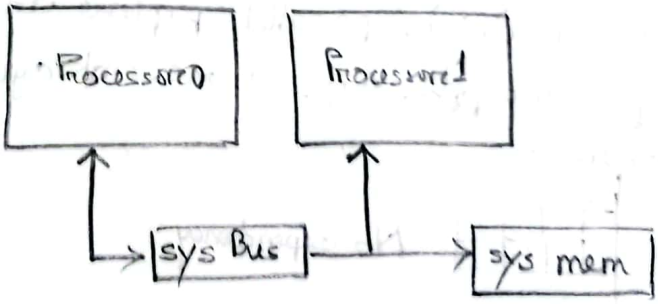
* Allows multiple threads to exist within the context to the process so they execute independently but share their process resources.

* process is divided into multiple thread.

MULTICORE



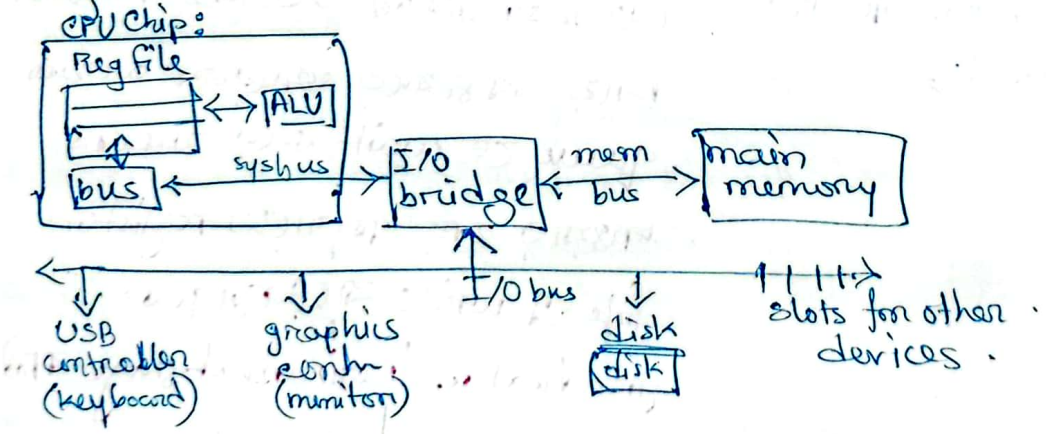
MULTIPROCESSORS



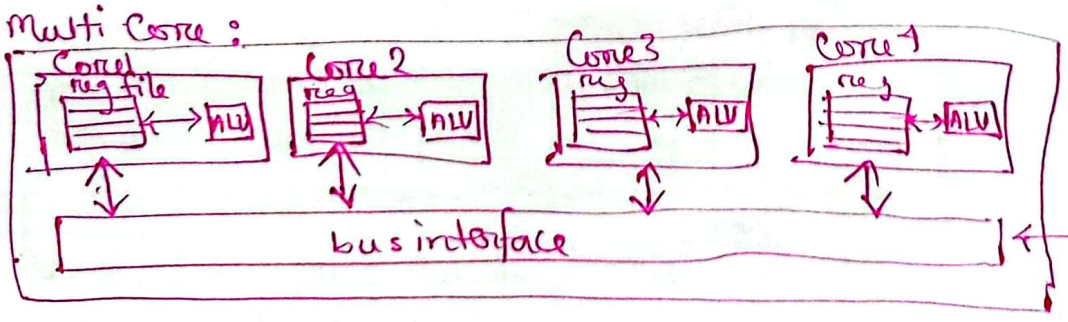
* Pipeline is within CPU.
* Cores are ~~the actual CPU~~ pipelines has it own L1 cache for data
* Concurrent execution – runs two/more prog in overlapping time phases.
  single core
* Parallel execution/Multicore – execution is broken to sub parts – done at the same time.
  ↓
  real way in which multiple tasks are done.

CORE	VS	THREAD
* unit of CPU ~~[illegible]~~		* unit of execution on ~~single cores~~ concurrent.
* increases work accomplished at a time		* improves throughput, computational speed-up.
* Actual hardware component		* Virtual component manages the tasks
* Content switching.		* use multiple CPUs for operating numerous processes.
* single process unit		* multiple processing unit

Single Core Architecture:



Multi Core:



IN ORDER

1. Ins. Fetch.
2. If operands (input) available in registes. then the ins. dispatched to the func. Unit.
3. If 1/more operands are during the current clock cycle, they are fetched mem.
4. Ins. executed by the appropriate FU.
5. FU writes results back to the register file.

So mainly INOrder (orderly) চলে.
Ins. fetch করে register-এ operand-এর জন্য wait করে, পেয়ে গেলে Func.Unit-এ চলে যায় – কাজ শেষে execute করে → writes back to register file.

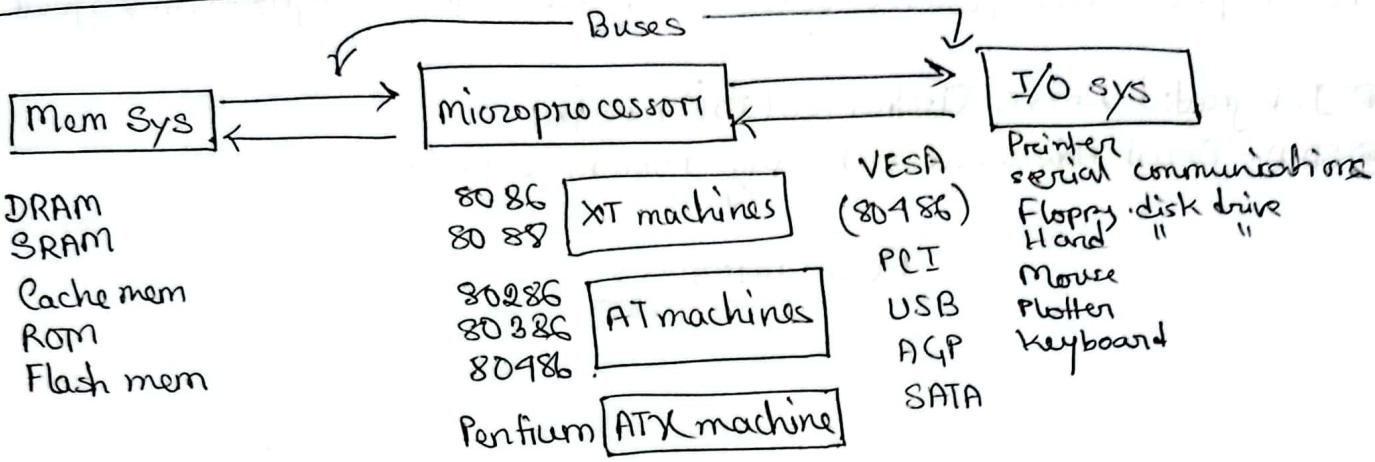
(VS)

OUT OF ORDER

1. Ins. Fetch.
2. Ins. dispatch to an ins. queue. (← ins. buffers or reservation stations)
3. Ins. stays in queue until input operands are available. It can leave the queue before older ins.
4. Ins. issued to the appropriate FU. and executed by that unit.
5. Results are queued.
6. Only after all older ~~files are~~ ins have their results written back to the register file then this result written back to the register file. It's called GRADUATION or RETIREMENT STAGE.

Out of Order-এ কোনো order নাই। ins. এসে queue/reservatio station-এ থাকে – যার operand যখন আসে সেই instruction তখনই Fn unit-এ গিয়ে কাজ করে আরেকটা answer queue তে wait করে সব ans পাওয়ার পর এক্সাথে register file-এ write করে যাকে graduation/retirement stage বলি।

μP Based Personal Computer Sys:



(*) Sys. Bus: Control lines, Address bus and data bus constitute the System bus and are guided in parallel to the bus slots.

Peripheral Bus: 8 bit devices [capable of transfering 8 bits of data at the same time]

(*) Bandwidth – Speed [key-diff]: bandwidth is the capacity available for use in data transmission while the speed is the data transferingrate.

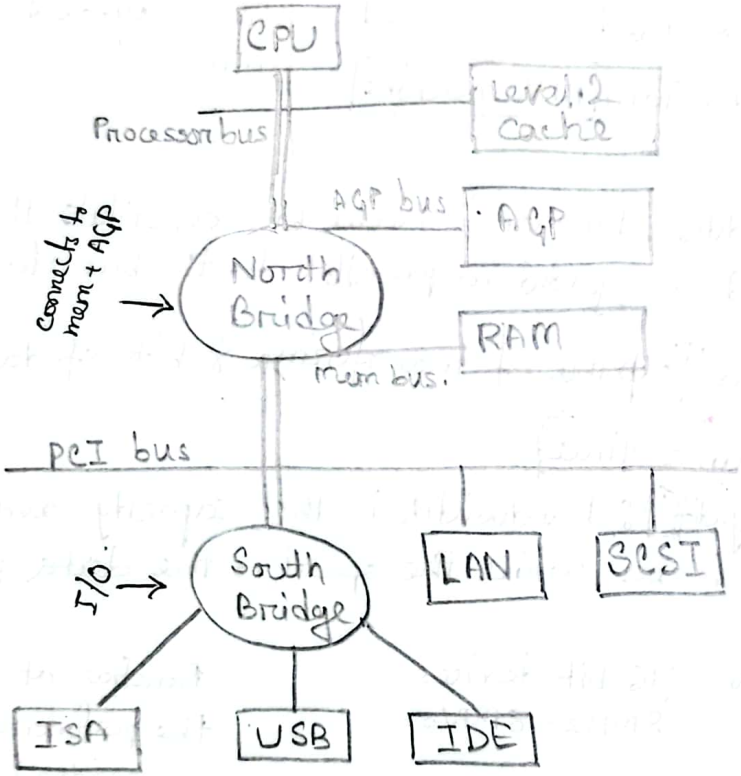
Bus architectures used:

ISA standard PIB 80286	· 16-bit devices 8 MHz – 8 MB/s.	Function at 8 MHz reduces the performance of disk and video interfaces.
EISA 80286-80486. (compatible with earlier)	32-bit devices 8 MHz – 33 MB/s.	
VESA Local Bus (VL) interface disk and video to μp.	· 32 bit μp (Latest 64 bits) competes with PCI	
PCI – Peripheral Component Interconnect BUS	32/64 bits Bus Speed 33 MHz	
VESA Local BUS.	64-bit same clock speed as μp.	
USB – Universal Serial Bus – connects keyboard, mouse, modem, and sound cards with μp. → serial Datapath and twisted pair of wires → Supports separate power supply for sound sys. reduces noises.	data sequentially USB 1 – 10 Mbps. USB 2 – 480 Mbps [wireless reduces wires, cost] 4 wire → red (+5) yellow white (−) blue → twisted pair → interfare reduce	
Advanced Graphics Port (AGP) – data transfer between video card and μp.	64 bits data path. 66 MHz – 533 Mbytes/second. Latest 8X (8×266 MB/s) or 2 Gbytes/s.	

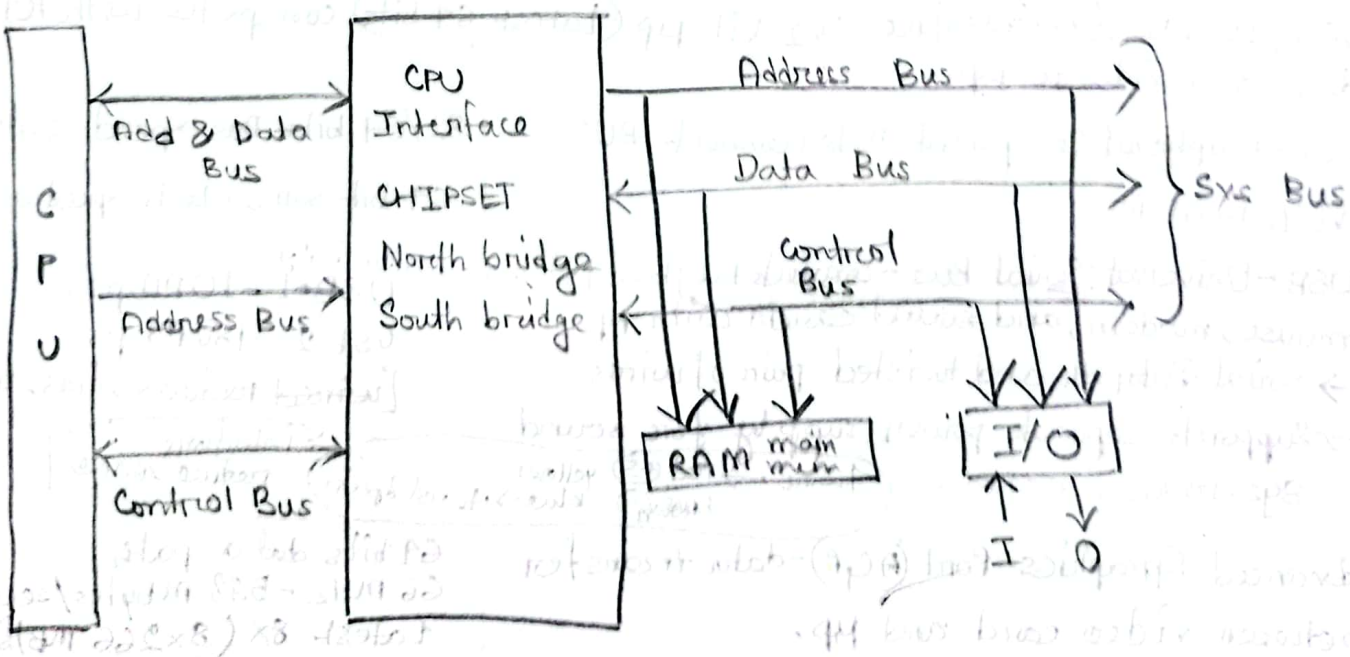
- Serial ATA interface (SATA) - with HD to PC | SATA - 2 300MB/s.
- PCI Express bus operates video cards at 63 GB/s | 4GHz (Gen 3) Speed.

- Integrated Drive Electronics (IDE) interface.
- SATA - Serial Advanced Technology Attachment - (or Serial ATA)

BUS Interfaces



BUS Operations



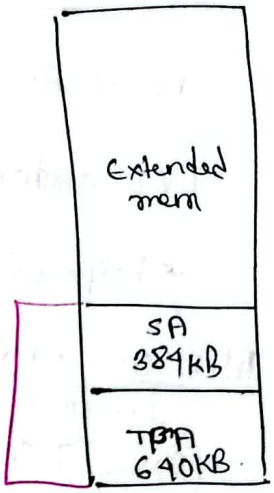
MEMORY MAP OF the PC.

XMS – Extended memory sys.
No extended mem sys in 8086, 8088

80286/80386SX → 15Mbytes.
80386SL/SLC → 31Mbytes
80386EX → 63Mbytes
80386DX, 80486, Pentium → 4095Mbytes.
Pentium Pro → 64Gbytes. (64GB – 16MB).

64G less 1M for Servers
4G less 1M " PC.

Extended Mem



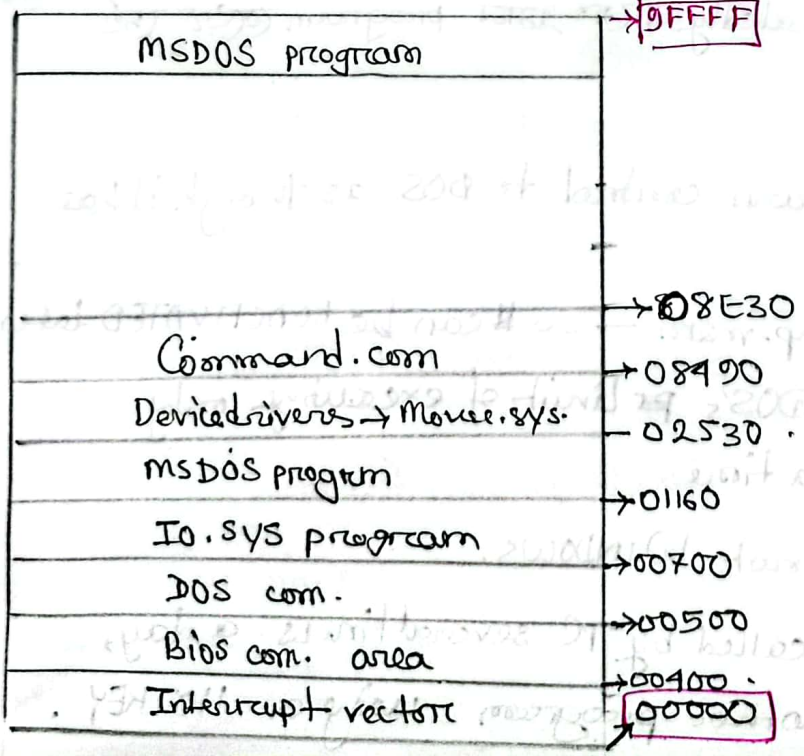
~~Real~~ Before 80286:
Real mode provides no support for memory protection, multitasking, or code pivilege levels.

After 80286:
Protected mode.

DIFFERENCE btw REAL and PROTECTED Mode.

* Transient- Program Area (TPA) is a DOS concept. 640KB *** – 1MB.
* System Area 384KB ***

*** TPA → 0x00000 – 9FFFF : 640~~MB~~KB (00000 – 9FFFF) ~~20~~ 20 bit address.



Interrupt Vector table: 32-255 → user defined (each 4 bytes)

Free TPA - তে code থাকে, — এটা access করছি (255 indexes) interrupt-এর 4 bytes each

14-31 Reserved

IVT contents →

8bit	8bit	8bit	8bit
Segm. High	Segm. Low	Offset high	Offset Low
3B	2B	1B	

IVT-14-31 OS reserved
32-255-user-এর

* helps to access features of DOS, BIOS and apps.

ISR (Interrupt Service routine)

BIOS and DOS com. Area: Temporary Data থাকে

↓ EPROM/ROM-এ থাকে

Basic I.O. system

* [transient data থাকে যেগুলো program use করে to access I/O devices and internal features]
* Hard disk-এ বাচ্চা না হেয়ে DOS-এ data থাকে
* program off হয়ে গেলে data vanish হয়ে যায়।

MSDOS area: Controls operation of Com.sys.

I/O sys:
* DOS start হলে HD থেকে load হয়
* Contains link programs for IO devices

Command.com: controls PC, I/O devices when working in DOS.

Free TPA: TSR program (Terminat and Stay resident)

↓ repeatedly কাজে এমন program থেকে নেই

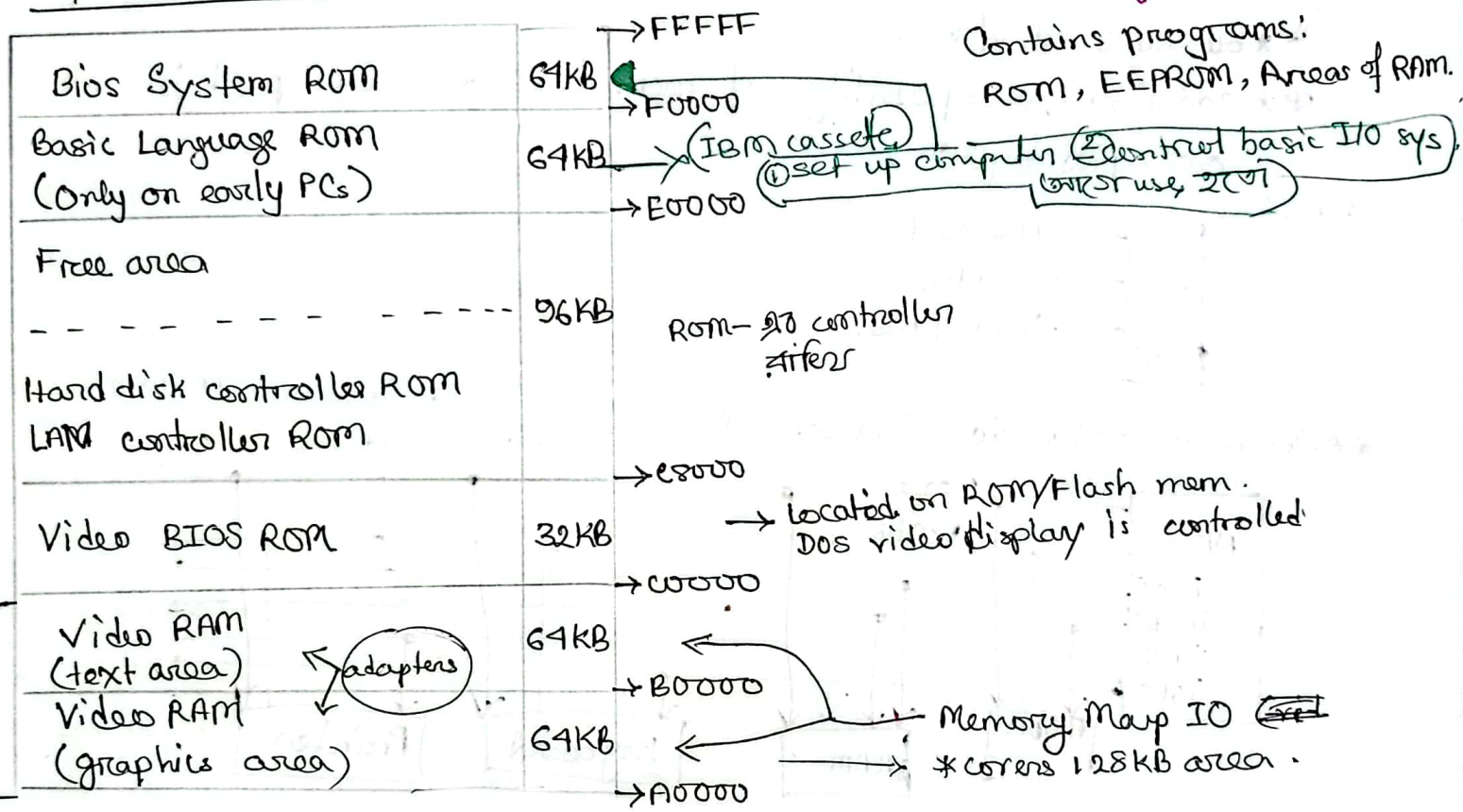
↓
* runs under DOS
* uses sys call to return control to DOS as though it has finish.
* but remain in comp. mem — so it can be REACTIVATED later
* partially overcame DOS's limit of executing only 1 proag/task at a time.
* Used only in DOS not WINDOWS.
* UTILITY SOFTWARE called by PC several times a day, while working in another program, using a HOTKEY.

* Early/popular example — BORLAND SIDEKICK.
* Serve as device drivers for HD. thats OS doesn't support

→ original call: INT 27h is called TSR.
(↳used to make up to 64 KB of program's mem resident

improved call (by mg: DOS E version 2) → INT 21h/31h.
(↳removed limitation of 64KB INT 27h and let program return an exit code.

⇒ System Area: ROM/Flashmem(EEPROM)/R/W mem for data storage



Video RAM: 128KB বুঝে থাকে।
- 64KB (Bitmapped graphics)
- 64KB (text)

depends on video adapter

Free memory sys:
* expanded mem in PC
* upper mem sys in AT.

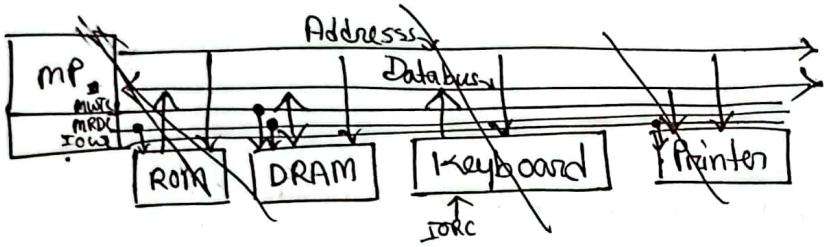
Video BIOS:
* program থাকে যেটা DOS videodisplay control করে.
* ROM/EEPROM-এ located.

HD controller :
LAN " : → HD ROM/BIOS.

3 ways system Bus can be allotted to Memory and I/O:

1. Separate set of address, control and data bus to I/O and mem (PMIO) isolated → mouse, keyboard
2. Common Bus (data - address).; separate control lines.
3. " " (data, address, control). memory mapped procedure (MMIO) → display

→ Simple – cause diff. address space and instruction – but more buses.



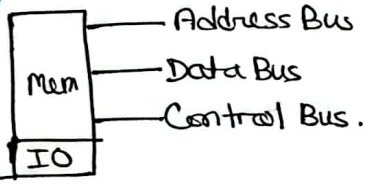
16 bit IO address
8086 ; 20 bit mem 00000-FFFFFH

MWTC – mem – write
MRDC – " read
IOWC – IO write
IORC – IO read

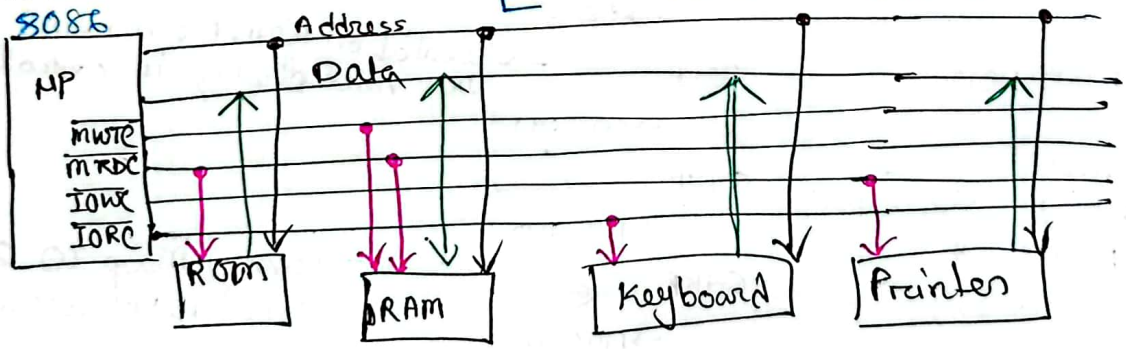
* ~~but~~ same set of ins.

** addressing capability of mem less as some part is occupied by IO.

MMIO

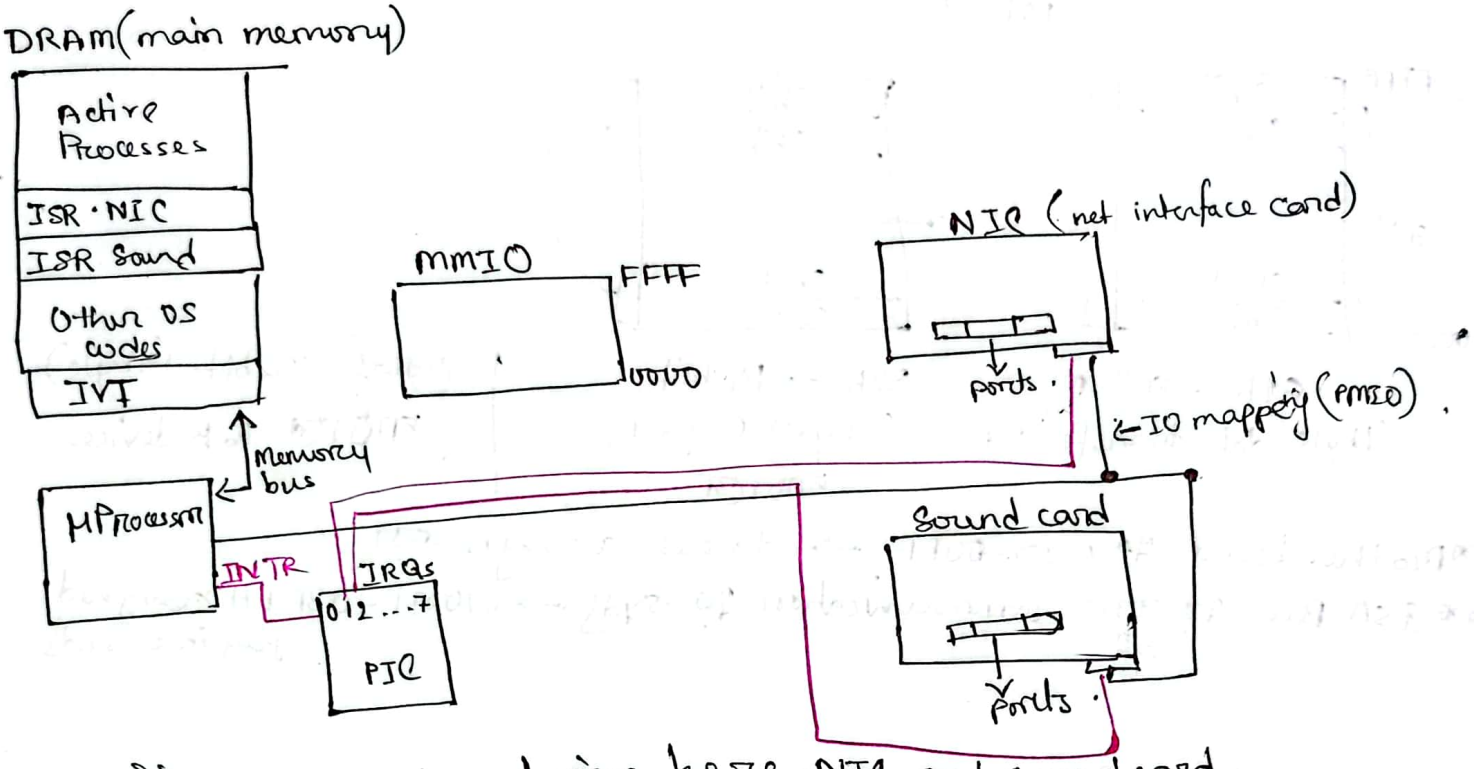


** Common data-address buses [MP 8086 → 20 bit memory address (00000-FFFFF)
→ 16 bit I/O address



Basic I/O Architecture

MMIO and PMIO combinely আমরা এই system-এ use করি

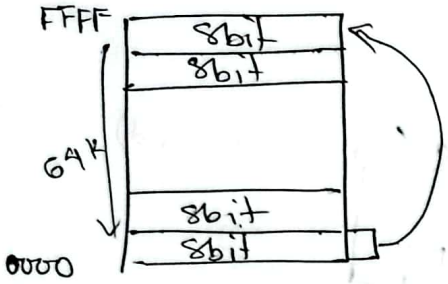


~~Suppose~~ There are two devices here, NIC and sound card. Suppose a data ~~can~~ coming from another computer, that NIC wants to transfer in memory - NIC needs help from μprocessor as it can not transfer data alone it has to do it through μprocessor. So, NIC will send a request to μprocessor (interrupt request (IRQs)) via PIC. Now μprocessor (PIC) will see if these interrupt request (INTR) is enable or not. If INTR is enable the μp will receive the request. After receiving μP will send an acknowledgement and through Databus PIC lets μprocessor know the interupt number. Then processor sends the number to memory (IVT). IVT will detect the address by the interupt number and find the code in ISR NIC.

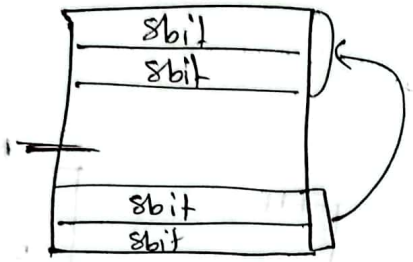
That code has some mapped port numbers which are actually the register numbers of NIC (exple - status register / flag register etc). Which means reaching ISR NIC and writting anything in the code will let NIC ports know the issue, then NIC can read the instructions and transfer the data. Thus both MMIO and IO mapping is

I/O Space 16 bits → 0000H - FFFFH

* helps accessing 64K - 8bits devices
  32K - 16 " "
  16K - 32 " "



64K – 8bit করে (1byte data transfer হয়)



32K – 16 bit (2 byte) করে data পাঠালে

একবারে 32bit (4byte) পাঠালে 16K devices.

** Mother board - এর 0000-00FF → address assign করা
** ISA Bus - এর মাধ্যমে communication - এর জন্য → 0100H - 03FFH assigned plug in cards

Programming Model

Visible
* Registers used visibly
* 8086 to Core 2 register

Invisible
* backend - এ use হয়
* 80286 - protected mode তে use হয়

Registers:

① General/multipurpose : (Visible)
- (AX) accumulator - mul, div, offset — 8bit করে (AL, AH)
- (BX) Base - offset address. — (BL, BH)
- (CX) Count - loop, rep, shift — (CL, CH)
- (DX) Data - memory, result of mull — (DL, DH)
- (BP) Base pointer - mem location
- (SI) Source index - source data str. — string - এ use করি
- (DI) Destination index - dest " "
- (SP) Stack pointer - address in stack

16 bit registers

General বলি কারণ 8086, 8088 (16bit) → 80286, 80386 and above 32 bit register হয় (অনেকে share করে)

multipurpose কারণ - * বিভিন্ন কাজে use করি

*** এগুলো general registers 80386 or 32 bits ~~reg~~ processor গুলোয় use করি। এদের নামের সামনে E থাকবে। [E থাকাগুলো 32 bit register]
example EAX, EBX, ~~etc~~ ESI etc.

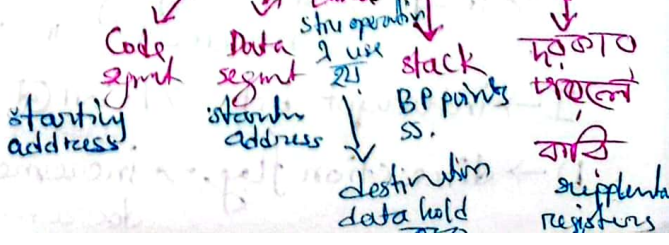
*** RAX, RBX etc 64 bit processor-এ use হয় না cause এরা 32 bit. এই কারণে আমরা EAX, EBX, ESI etc use করি। এরা 64 bit
But 64 bit registers (RAX, RBX etc) can be used for 32 bit processor
32 bit registers can be used for 16 bit processor.
কারণ 64 bit-এর মধ্যে 64 bit register-এর lower part টা নিয়ে use করা possible যেটা 32 bit (exple- EAX, EBX ~~etc~~ etc).
আবার 32 bit-এর lower part 16 bit so compatible for
16 bit processors (~~EAL, ECL etc~~). (EAL, ECL etc).

Thus, pentium4/Core2 is compatible to run in 80386 or 8086:
and 80386 is compatible for 8086, 8088 etc.

R10B, R9B → মানে 8 bits — byte. (B লাগানো
R8(B), R9W → " 16 bits — words
R9D → (D লাগানো মানে 32 bits) — Data
R9/R10/R8 ⟶ 64 bit

(2) Special Purpose register: যেগুলো functions বা value user visibly change করতে পারবে না — invisibly হয়তো possible but not visibly. Exple- Flags (condition/control operation; IR- Instruction pointer - next ins. address)
~~SP - segment regi~~

So, RFlags, RIP, RSP, ~~etc~~ segment Registers (CS, DS, ES, SS, FS, GS)

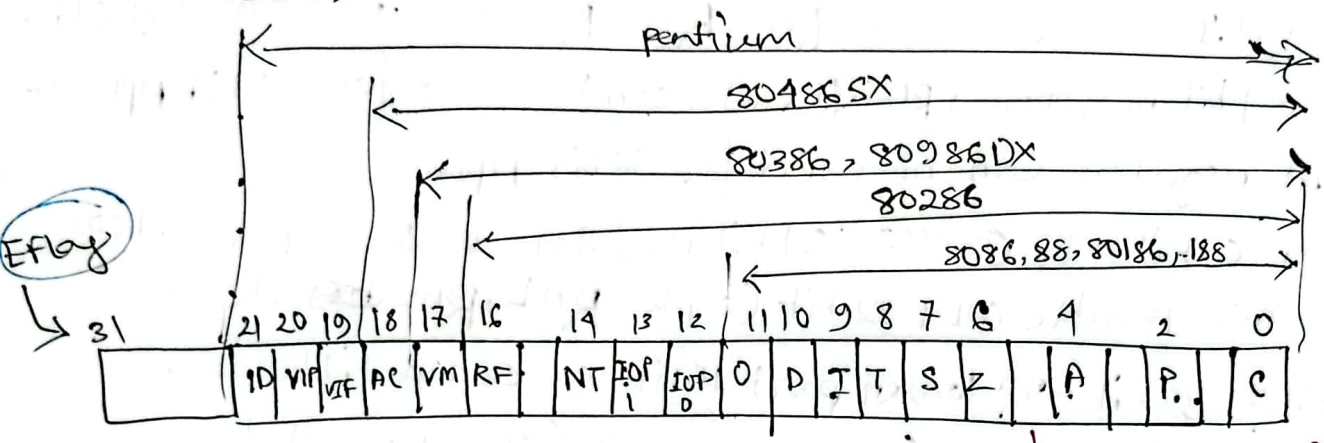


Flag Registers:

Microprocessor-এর status বলে দেয়

RFlags → 64 bit হয়, EFlags → 32 bits

8086, 88, 80186, 80188 → 16 bit-এর 11 bit use করতে



C → Carry bit, carry store করি

P → parity bit, even no. 1 → 0
odd no 1 → 1

Flag bit ALU-এর পর change হয় - just data transfer-এ হয় না।

A → auxillary Carry, binary-তে addition হলে carry A-তে store হয়

BCD addition

$(678)_{10}$ = 0110 0111 1000
(+) $(535)_{10}$ = 0101 0011 0101
$(1213)_{10}$ = 1011 1010 1101
12>10 11>10 13>10 → subtract 10
1 2 1 3

Binary-তে subtract 0110 $(6)_{10}$

1011 1010 1101
(-) 0110 0110 0110
(1) 0010 0001 0011

auxillary carry.

Z → zero flage — value zero থাকলে 0, না থাকলে 1

S → sign flag bit — positive - 0; negative 1

T → Trap → debugging mode → 1
normal " → 0

I → interrupt bit → 1 মানে enable, 0 মানে disable | STI, CLI
INRT pin

D → direction flag → increment → 0 | প্রোগ্রামরা set করতে পারি
decrement → 1 | STD (set D flag) হবে; CLD (Clear D flage) inc হবে

O → Overflow flag (addition-এর সময় ভাগের অমিলে 1, ঠিক থাকলে 0).

NT → nested task flag (subtask করার সময় 1 থাকে)

RF → resume flag (debugging mode-এর বাইরে, এসে আবার করার আগে 1 থাকে)

VM → Virtual mode (virtual mode-এ run করলে 1).

AC → alignment Technique (Only present in 80486SX that is set to 1 if a word or double word is addressed; 80487SX also uses it)

VIF → Virtual interrupt flag (Pentium-এ শুধু আছে; a copy of interrupt flag)

VIP → " " Pending (Pentium-এ শুধু, gives info of interrupt pending)

ID → (Identification) → Pentium-এর ~~CPUID~~, version number, manufacturers info CPUID-তে আছে কি নাই তা বলে

IOPL (IO privilege level) → used in protected mode.

IOP - 00 (মানে highest level privilege কি কে নাকি সেই status)
IOP - 11 (" lowest " " " " " ")

| REAL MODE Addressing | / Conventional / DOS memory ও বলা হয়

*** Just TPA আর system area-তে কাজ করে, বাইরে করে না (1m byte-এ)

Offset address → 16 bit
Segment " → 20 bit

Segment Addr (20 bit)	Offset address (16 bit)

Physical mem address (EA) = 20 bit segment (base) + 16 bit offset (limit)

8086 / 8088-এর ক্ষেত্রে segment address-এর upper 16 bit ব্যবহার base-এ বাকি ব্যবহার না

EA = seg (4 bit shl) + limit

Example: Segment register 2000H হলে starting address 20000H
Ending addr 2FFFF H → (20000 + FFFF)

or " " 2100 H হলে SA = 21000H; EAd = 21000 + FFFF = 30FFF H

Real mode allows relocation — emergency program-এর জন্য আমরা চাইলে free space-এ HD-থেকে reload না করে program-টিকে অন্য address-এ relocate করব (using relocatable code) just base register টা change করলে হবে।

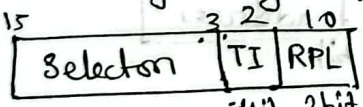
EA = CS : [IP]
start segment, offset

Advantage of Real Mode : Relocation of segment register
Disadvantage " " " : Complex h/w system for memory addressing..
- paragraphing use হয়: starting address-এর last byte always 0 হওয়া লাগে; 10200H-এ possible 10201H এলে possible না.
16 byte boundary in mem. system (Paragraph boundary)
- address computational delay.
- not protected.

Protected Mode

Both 1Mbyte + extended area-কে বলা হচ্ছে।
- used in 80286 to above
- windows operates. WIN32 environment
- no paragraph boundary (segments starts anywhere).
- segment registers hold selector (selector selects descriptor from descriptor table).
  ↓ describes memory segment location, privileges, length, access right

Math: Segment reg → DS/CS/ES.



RPL → requested privilege level.
00 highest (Kernel)
01 (Device drivers)
10 (OS service)
11 lowest (application)
TI = 0 global descriptor
TI = 1 local "

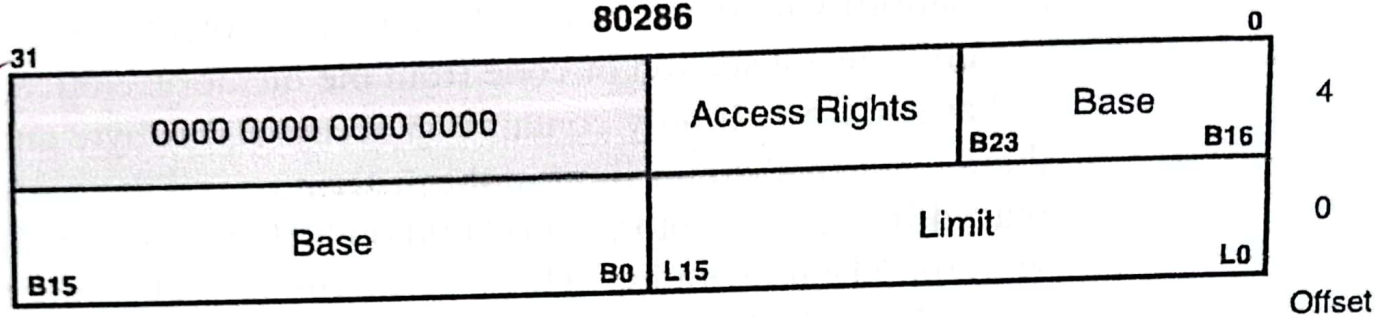
Flat Mode

Pentium base computer.
- not support real mode operation
- 64 bits system + flat memory system (— program as a single contiguous address space).
- no segmentation
- no use of segment register

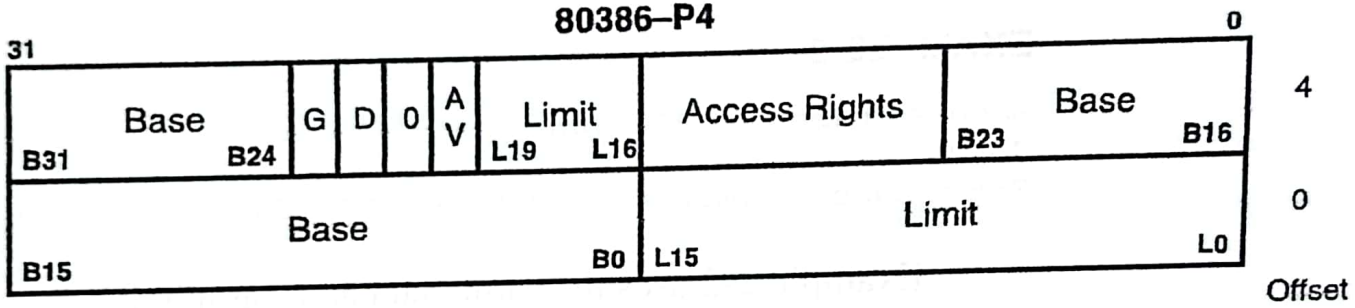
- Starting address = 0000 0000 00H
  End " = FFFF FFFF FFH
  So address 40 bits.

64 bit processor হলেও 40 bit address memory cause 40টা pin available — বেশি নাই, so 64 bit বানালেও possible না address length.

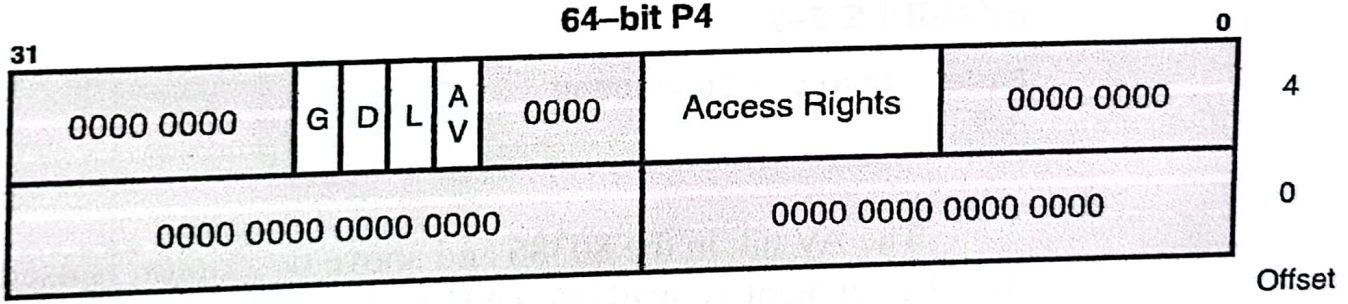
80286



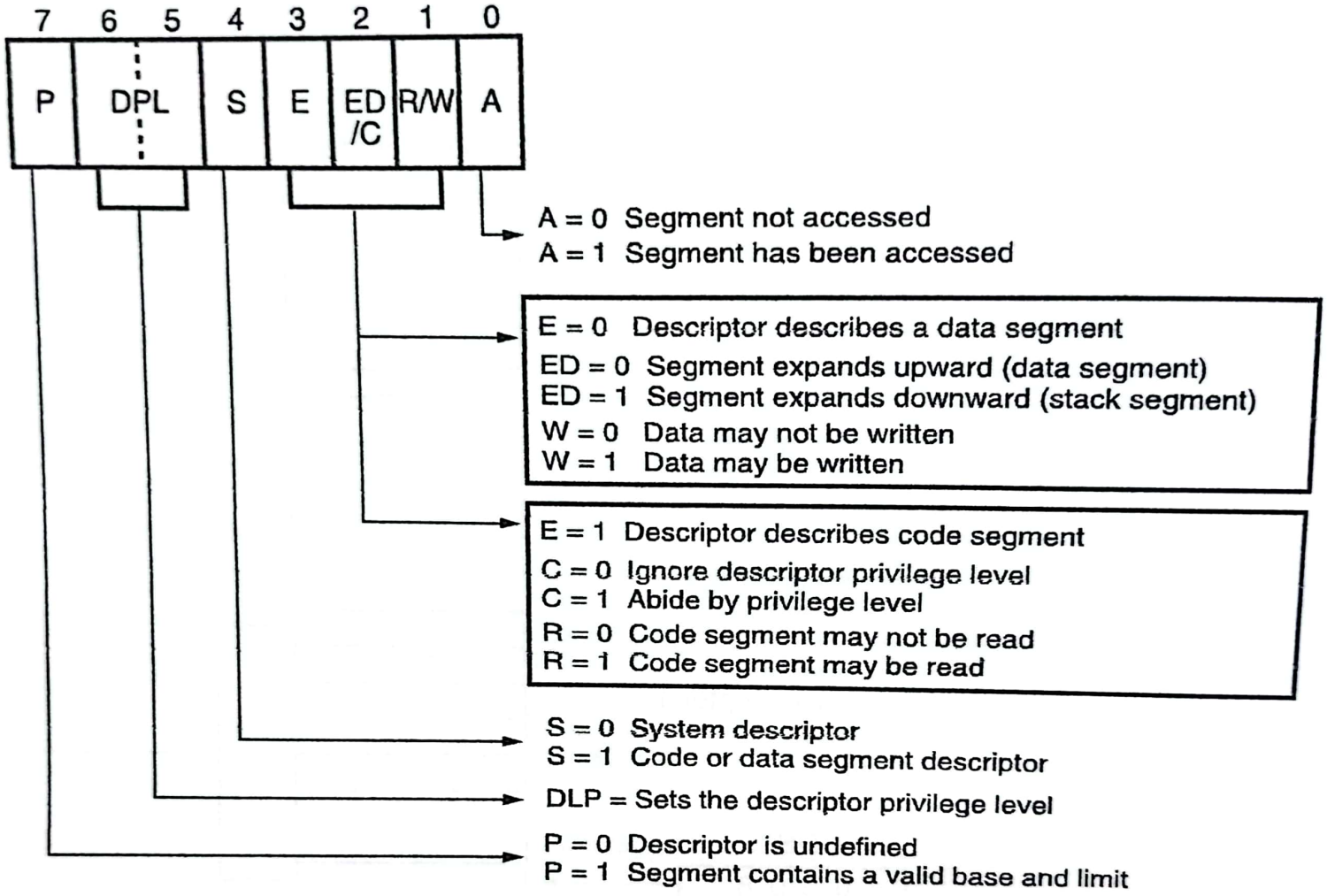
80386–P4



64–bit P4



descriptors.



Note: Some of the letters used to describe the bits in the access rights bytes vary in Intel documentation.

FIGURE 2–7 The access rights byte for the 80286 through Core2 descriptor.

FIGURE 2–7 The access rights byte for the 80286 through [illegible]

requested privilege level is 10 and the access rights byte sets the segment privilege level at 11. access is granted because 10 is higher in priority than privilege level 11. Privilege levels are used in multiuser environments. Windows uses privilege level 00 (**ring 0**) for the kernel and driver programs and level 11 (**ring 3**) for applications. Windows does not use levels 01 or 10. If privilege levels are violated, the system normally indicates an application or privilege level violation.

Figure 2–9 shows how the segment register, containing a selector, chooses a descriptor from the global descriptor table. The entry in the global descriptor table selects a segment in the memory system. In this illustration, DS contains 0008H, which accesses the descriptor number 1 from the global descriptor table using a requested privilege level of 00. Descriptor number 1 contains a descriptor that defines the base address as 00100000H with a segment limit of 000FFH. This means that a value of 0008H loaded into DS causes the microprocessor to use memory locations 00100000H–001000FFH for the data segment with this example descriptor table. Note that descriptor zero is called the null descriptor, must contain all zeros, and may not be used for accessing memory.

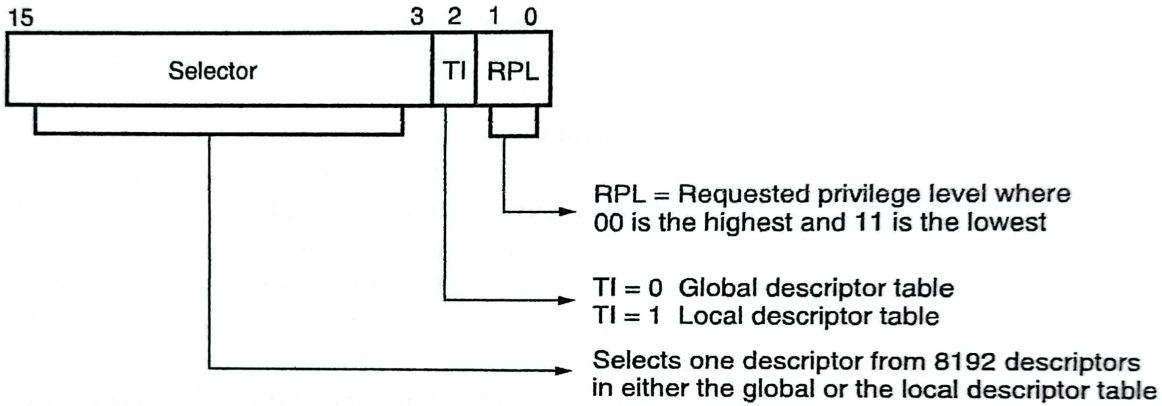


FIGURE 2–8 The contents of a segment register during protected mode operation of the 80286 through Core2 microprocessors.

FLAT MODE MEMORY

The memory system in a Pentium-based computer (Pentium 4 or Core2) that uses the 64-bit extensions uses a flat mode memory system. A *flat mode memory* system is one in which there is no segmentation. The address of the first byte in the memory is at 00 0000 0000H and the last location is at FF FFFF FFFFH (address is 40-bits). The flat model does not use a segment register to address a location in the memory. The CS segment register is used to select a descriptor from the descriptor table that defines the access rights of only a code segment. The segment register still selects the privilege level of the software. The flat model does not select the memory address of a segment using the base and limit in the descriptor (see Figure 2–6). In 64-bit mode the actual address is not modified by the descriptor as in 32-bit protected mode. The offset address is the actual physical address in 64-bit mode. Refer to Figure 2–15 for the flat mode memory model.

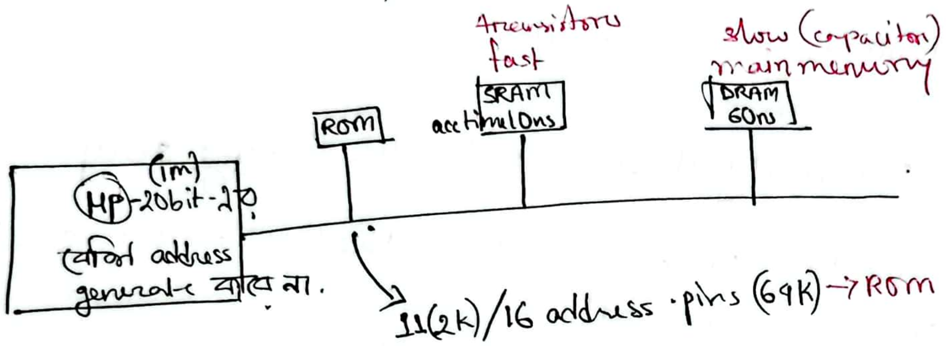
This form of addressing is much easier to understand, but offers little protection to the system, through the hardware, as did the protected mode system discussed in Section 2.3. The real mode system is *not* available if the processor operates in the 64-bit mode. Protection and paging are allowed in the 64-bit mode. The CS register is still used in the protected mode operation in the 64-bit mode.

In the 64-bit mode if set to IA32 compatibility (when the L bit −0 is in the descriptor), an address is 64-bits, but since only 40 bits of the address are brought out to the address pins, any address above 40 bits is truncated. Instructions that use a displacement address can only use a 32-bit displacement, which allows a range of ±2G from the current instruction. This addressing mode is called RIP relative addressing, and is explained in Chapter 3. The move immediate instruction allows a full 64-bit address and access to any flat mode memory location. Other instructions do not allow access to a location above 4G because the offset address is still 32-bits.

If the Pentium is operated in the full 64-bit mode (where the L = 1 in the descriptor), the address may be 64-bits or 32-bits. This is shown in examples in the next chapter with addressing modes and in more detail in Chapter 4. Most programs today are operated in the IA32 compatible mode so current versions of Windows software operates properly, but this will change in a

Memory Interface :

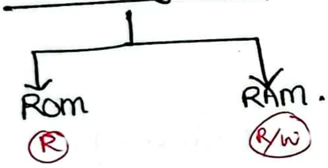
Objectives ① speed maintain.
② addressing

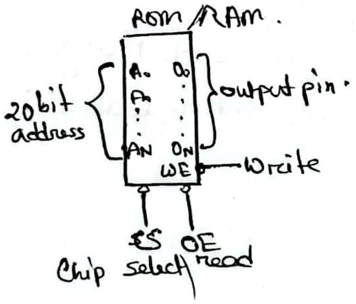


① speed mismatch
② address space "
এসব ঠিক করি
interfacing দিয়ে

তিনটা policy : ① Address decoder.
② Programmable Logic Device (PLDs)
③ Interfacing Memory to Data Bus.

use for storage
Memory Unit : write → new info in mem (store)
read → store mem out (transfer)





expl. 2716. [11 address (A) pins.
8 output (O) "
| 2Kx8 EPROM |
PD/PGM (Program down) CS select (read operation)
Vpp-তে 25V দিলে + pulsing PGM
while holding CS — program the device.
modify করা যায়

RAM Address = 11 pin. ($A_0 - A_{10}$)
Data in/out = 8 pin ($DQ_0 - DQ_7$)
CS - selection
OE - read
WE - write
] - active low

slowest SRAM → 250ns access time

Address Decoder : µm = 20 address pin (1MB)
EPROM = 11 " " (2KB
mismatch টা Decoder ঠিক করে

Exple - nand gate decoder

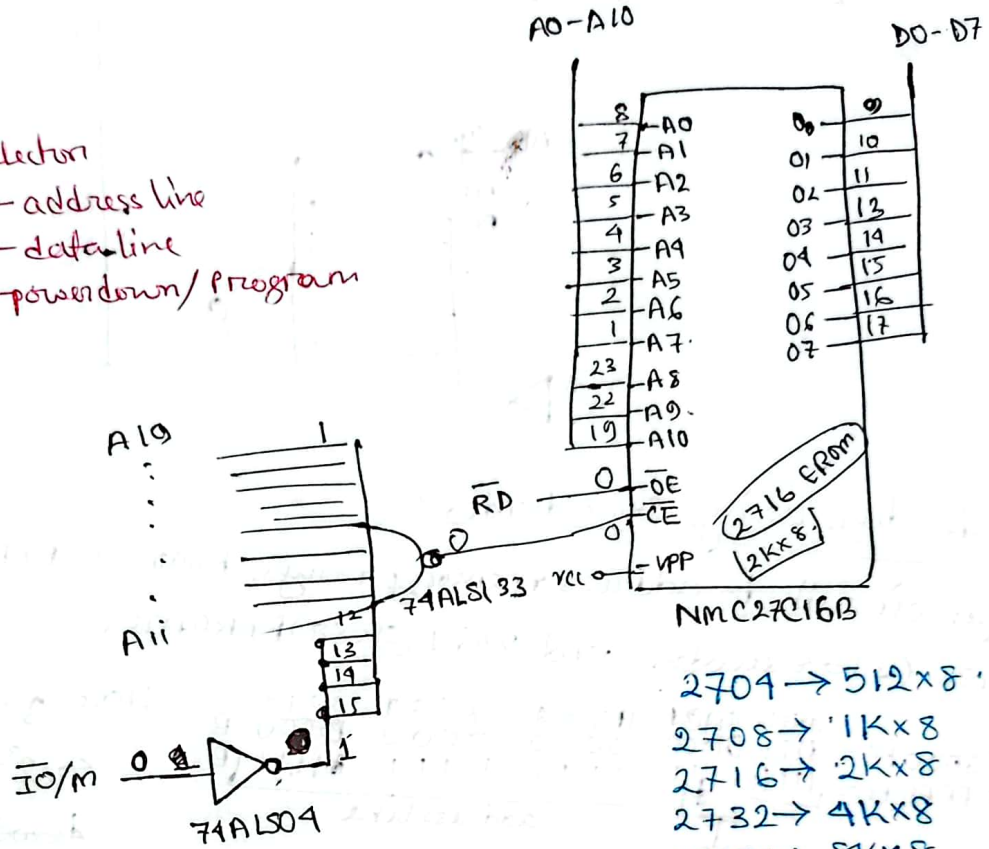
	.. 1111	1111	1XXX	XXXX	XXXX
FF800H =	1111	1111	1000	0000	000
FFFFFH =	1111	1111	1111	1111	1111

same গুলো Nand দিয়ে ঢুকাব

Ques: 2716 EPROM ~~Decodes~~ NAND GATE Decoder decodes Memory location FF800H-FFFFFH

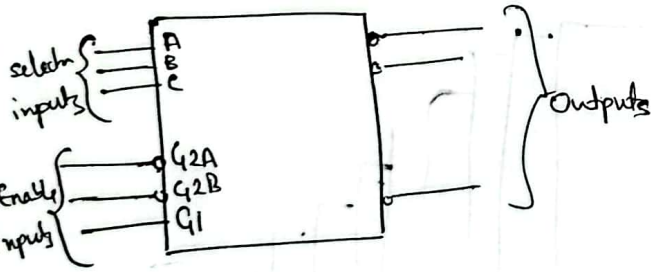
EPROM →
- $\overline{CS}$ → selection
- $A_0 - A_{10}$ — address line
- $D_0 - D_7$ — data line
- PD/PGM — power down / program

~~Vpp → 25/12.7V~~



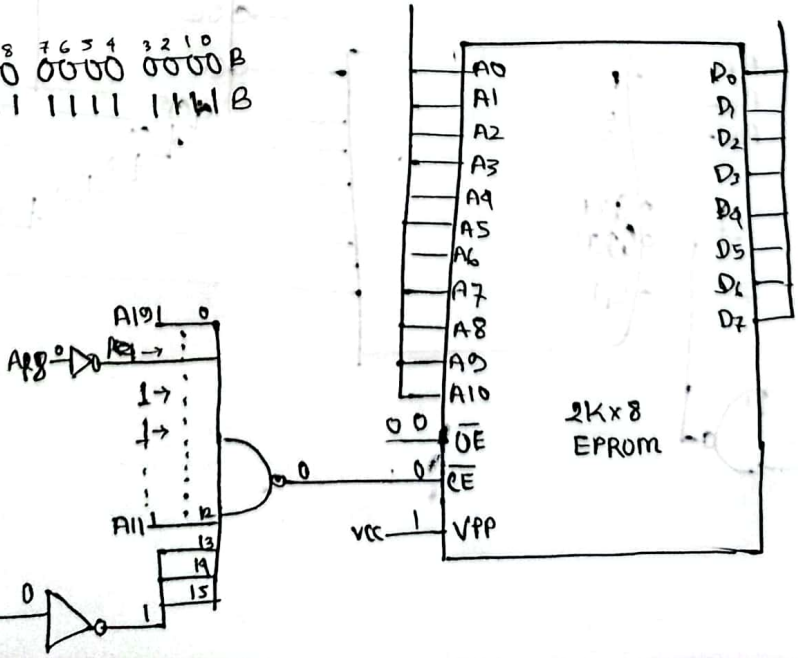
2704 → 512 x 8
2708 → 1K x 8
2716 → 2K x 8
2732 → 4K x 8
2764 → 8K x 8
27128 → 16K x 8
27256 → 32K x 8
27512 → 64K x 8
271024 → 128K x 8

3 to 8 line Decoder: (74LS138).

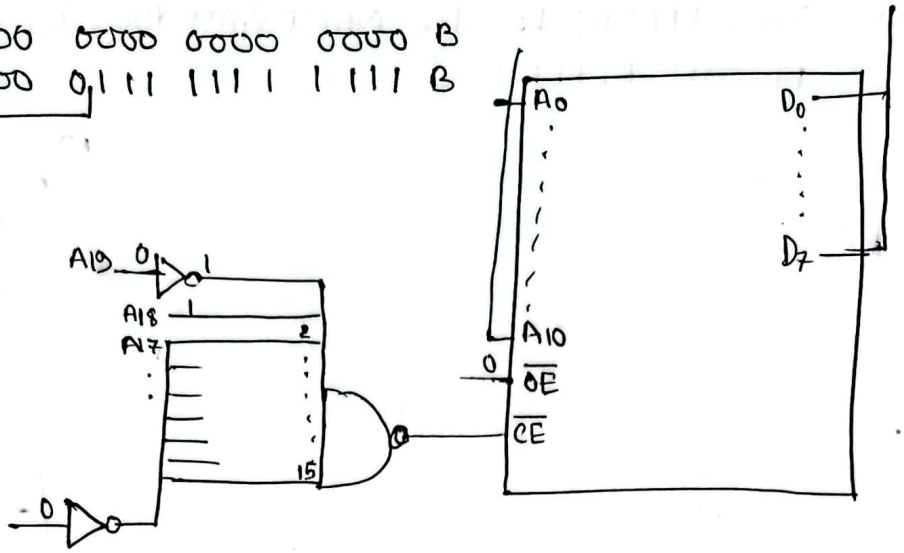


Exercise from book:

(18) DF800H = 1011 1111 1000 0000 0000 B
DFFFFH = 1011 1111 1111 1111 1111 B
same



(19) 40000H = 0100 0000 0000 0000 0000 B
407FFH = 0100 0000 0111 1111 1111 B



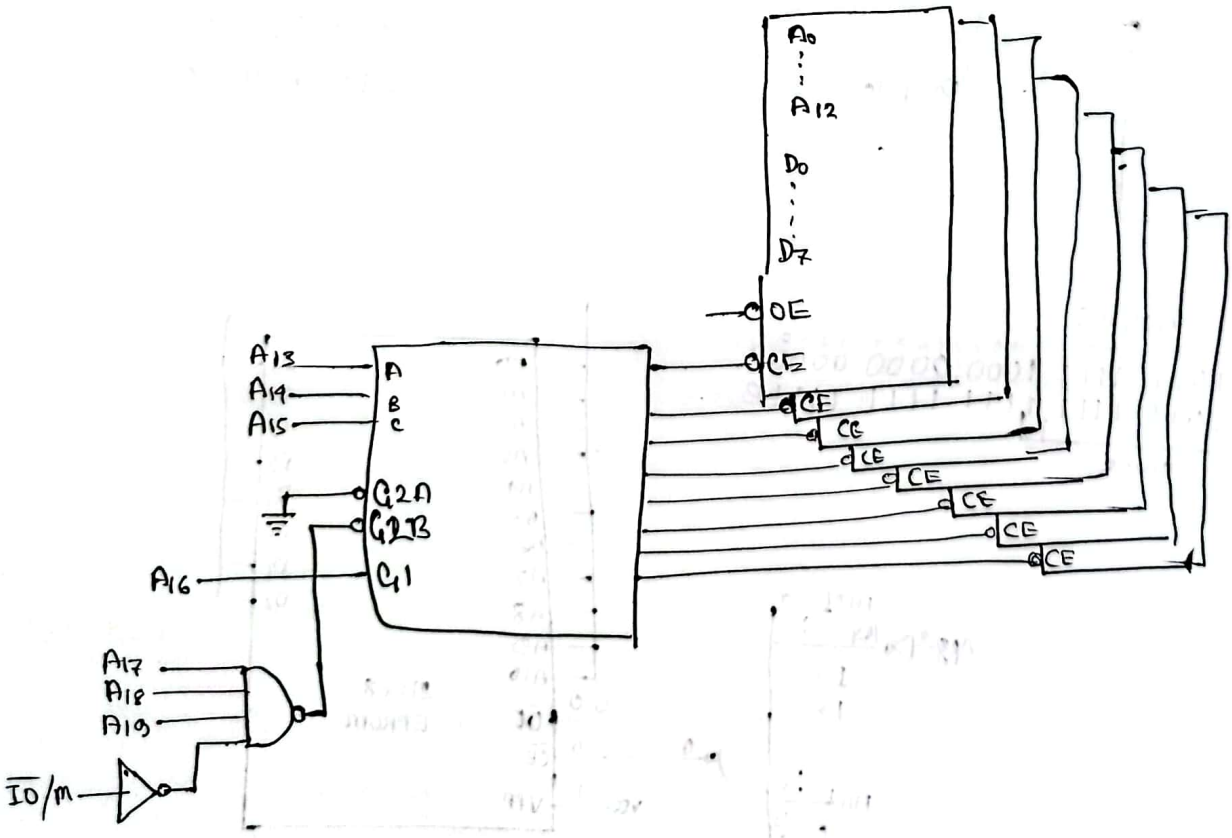
Address decoding using 3 to 8 decoder:

* Design circuit to address memory range F0000H - FFFFFH using 74LS138 (3 to 8 decoder) and 2764 (8K×8) EPROMs.

Soln:
19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
F0000H = 1111 0000 0000 0000 0000 B
FFFFFH = 1111 1111 1111 1111 1111 B
address line.

বাকি decoder-এ যাবে

Here, 8K×8 EPROM
So, $2^3 \times 2^{10} \times 8$ byte $= 2^{13} \times 8$.
~~decoder-এ input~~ 13 bit address line.



(21) range = 70000H - 7FFFFH ; EPROM → 2764 which means 8K×8 EPROM

$= 2^3 \times 2^{10} \times 8$ ll

$= 2^{13} \times 8$ ll

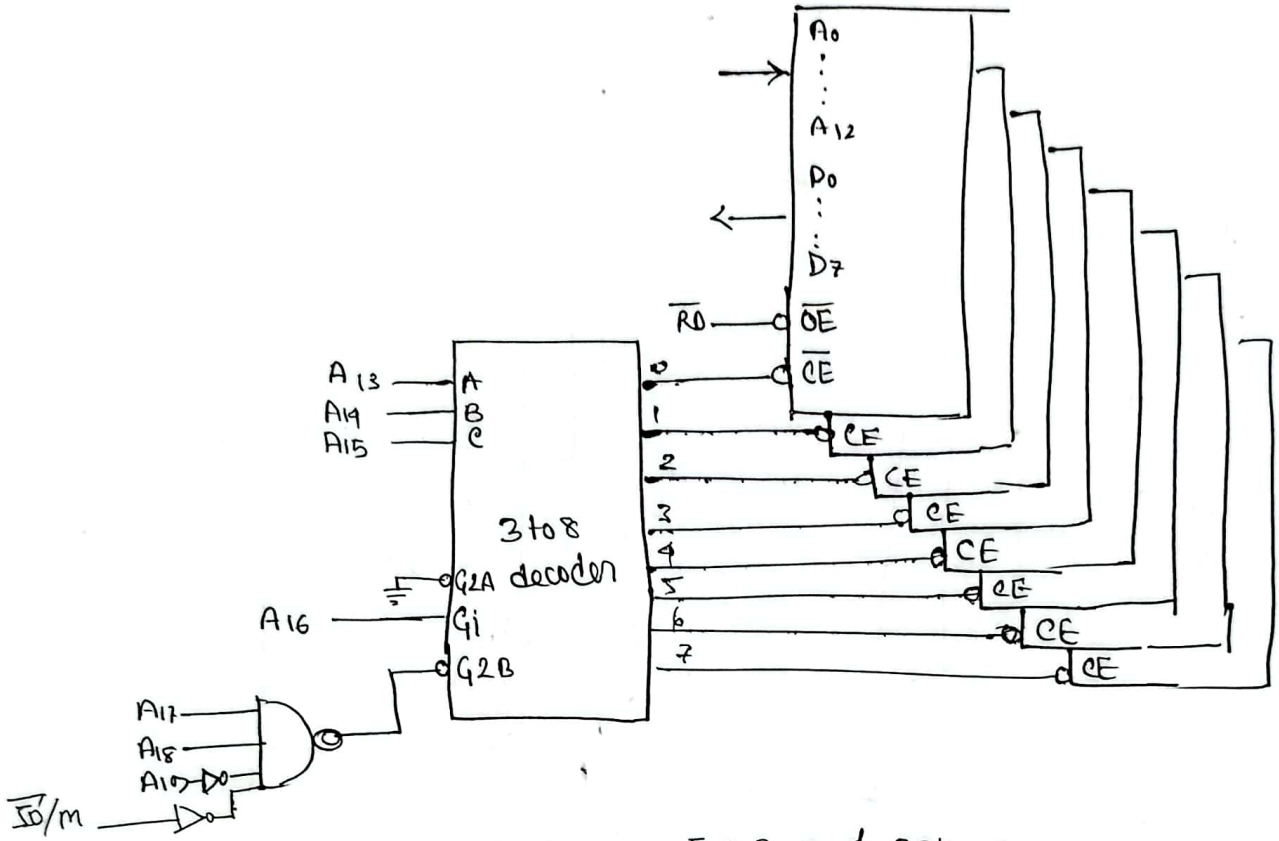
13 address lines.

19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

70000 = 0111 0000 0000 0000 0000 B

7FFFF = 0111 1111 1111 1111 1111 B

same. address line



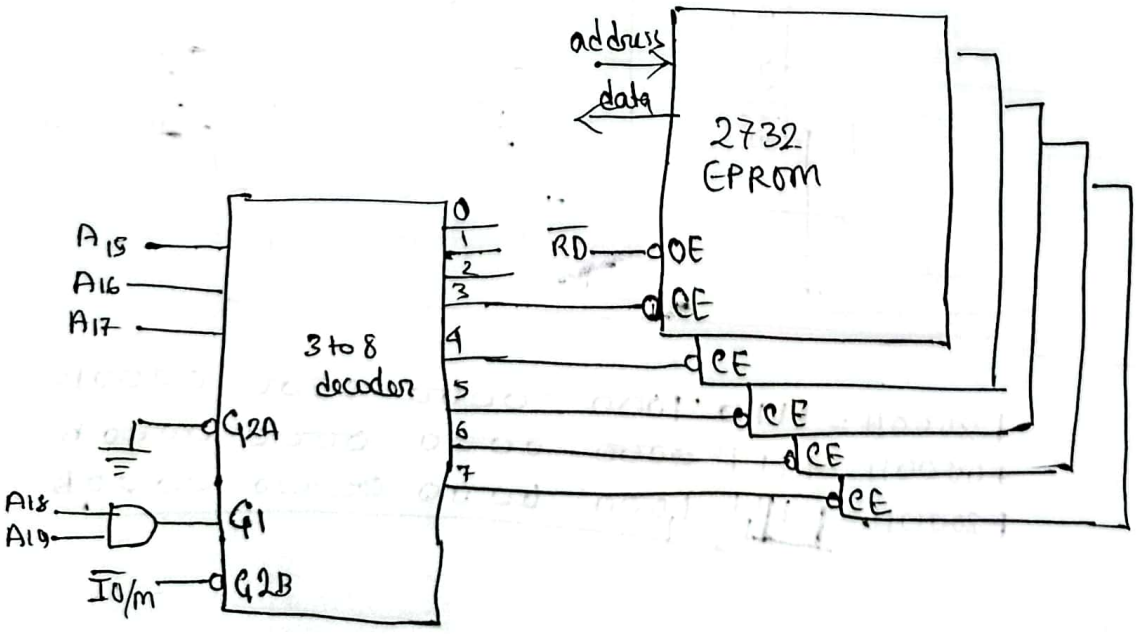
Quiz-3(3) Range : D8000H - FFFFFH 5 EROM of 32K×8.

$2^5 \times 2^{10} \times 8 \Rightarrow$ 15 bit address line

D8000H = 1101 1000 0000 0000 0000 B

FFFFFH = 1111 1111 1111 1111 1111 B

address line

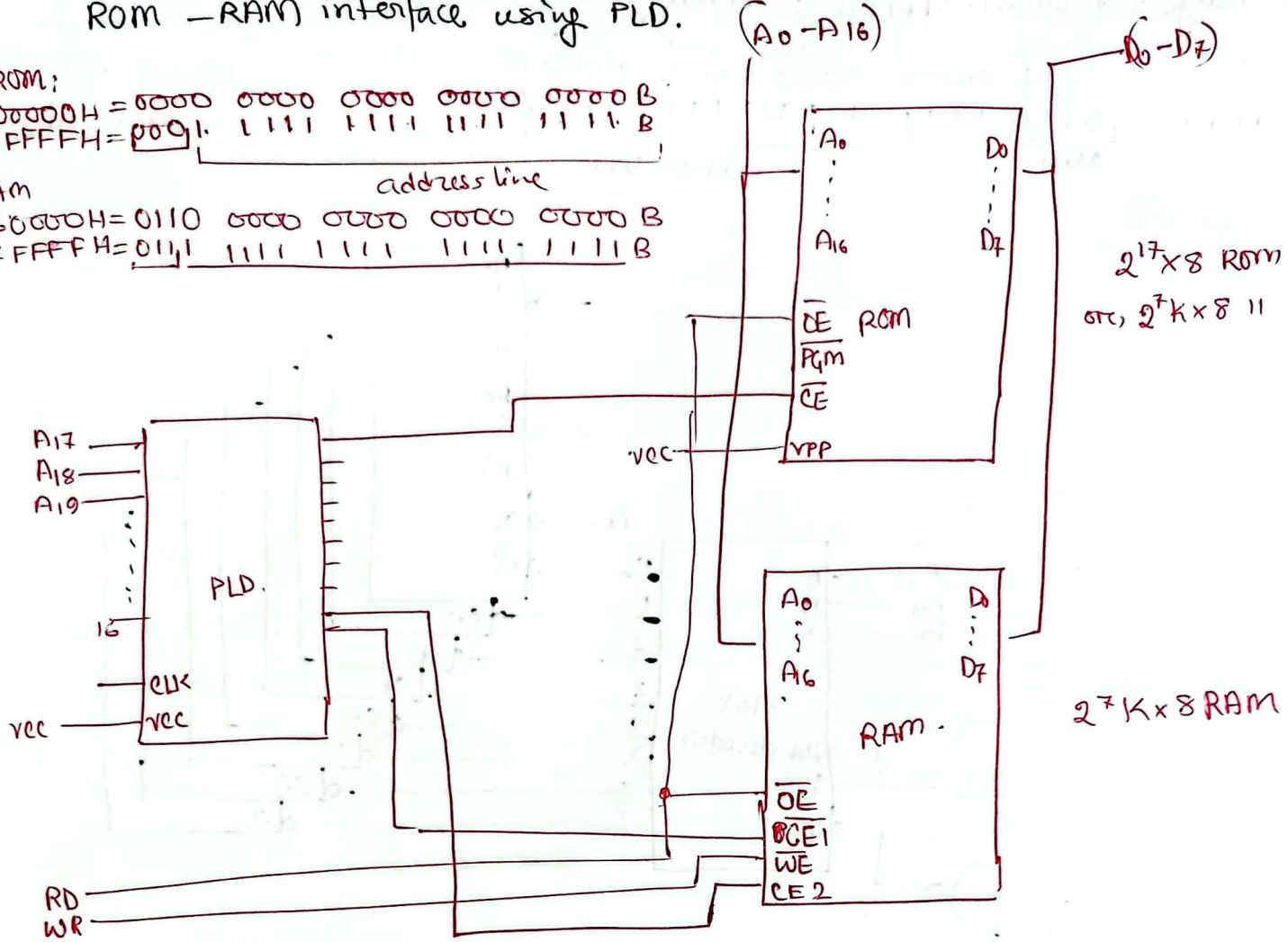


ROM – RAM interface using PLD.

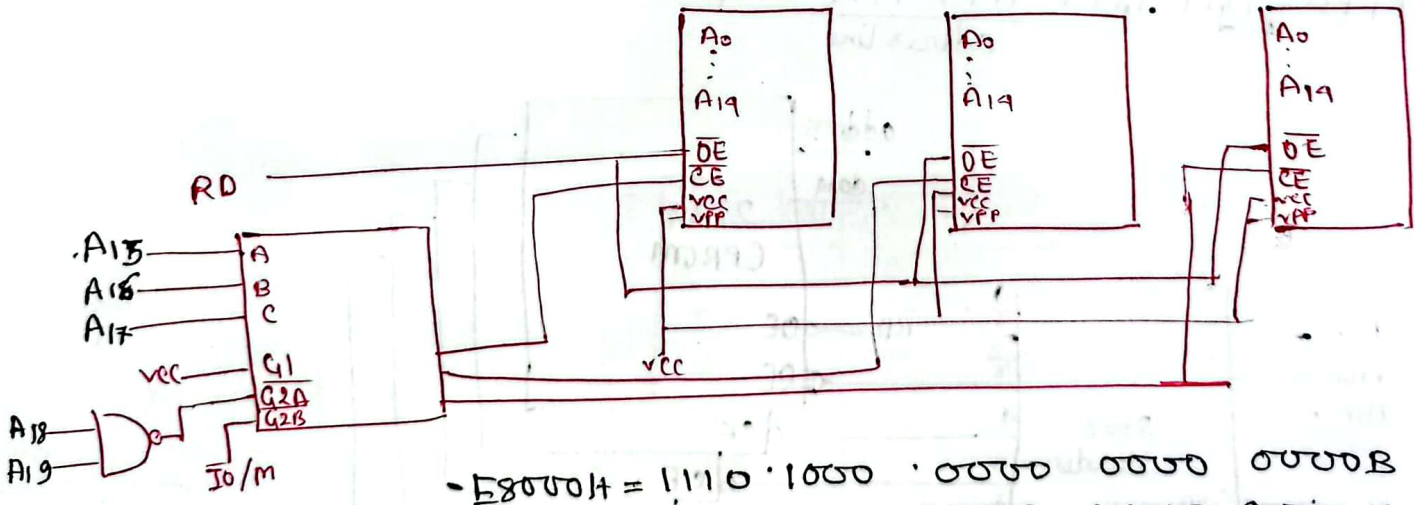
ROM:
00000H = 0000 0000 0000 0000 0000 B
1FFFFH = 0001 1111 1111 1111 1111 B

RAM
60000H = 0110 0000 0000 0000 0000 B
7FFFFH = 0111 1111 1111 1111 1111 B

address line



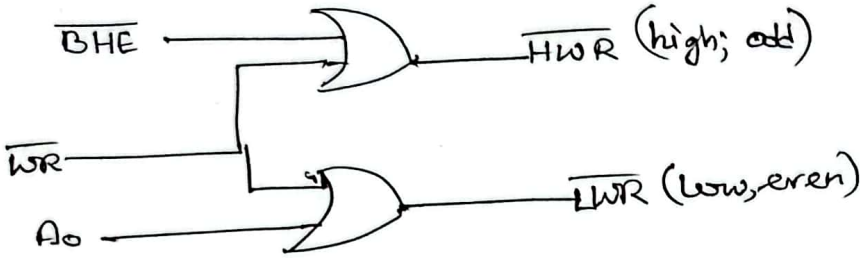
EPROM Interface with 8088 µp.



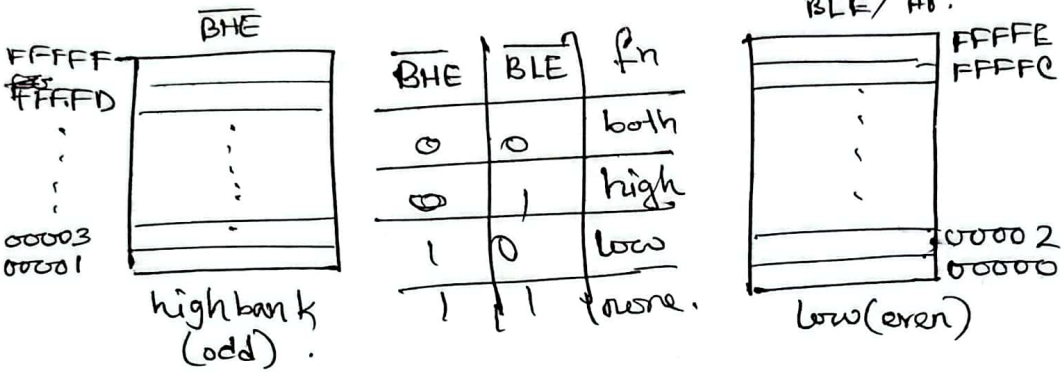
- E8000H = 1110 1000 0000 0000 0000B
- F0000H = 1111 0000 0000 0000 0000 B
- F8000H = 1111 1000 0000 0000 0000 B

Bank Selection : ① Separate decoders use
② Separate write signal.

Memory bank write selection input

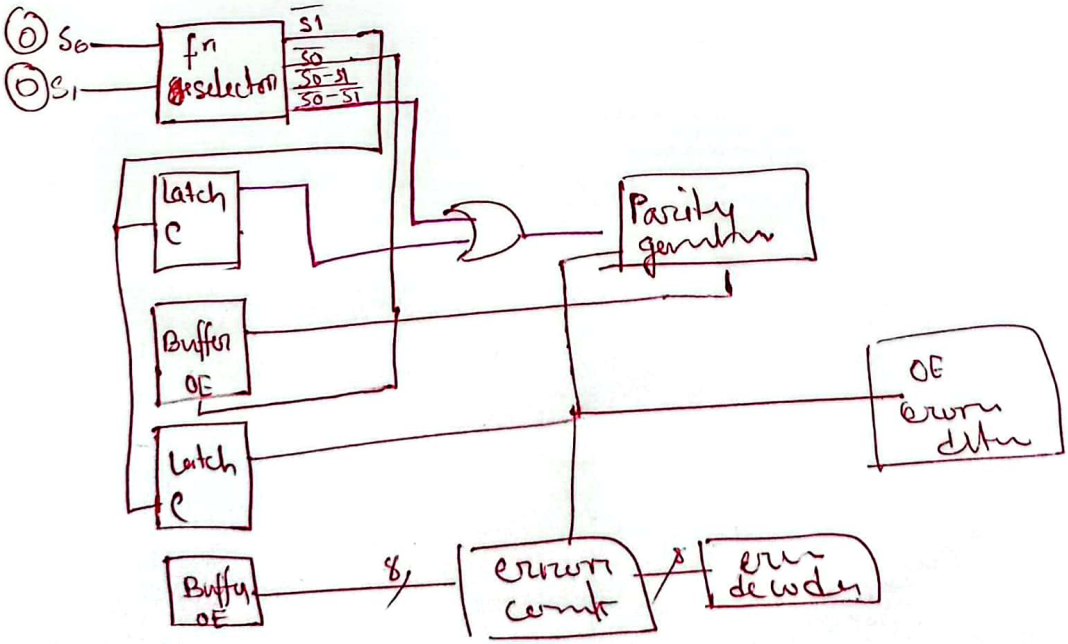


80286, 80386 → MWTC use হয় WR -এর পরিবর্তে



Error correcting Circuit :

	S_1	S_0		input	output
write →	0	0	EDAC general checkword	data inputs	output checkbank.
read →	0	1	" . read data & " "	" "	input " "
read →	1	0	Latch fury errors.	Latch data	latch " "
read →	1	1	correct data gen syndrom	Output corrected	Output syndrom



Dual 2 to 4 Decoder with EPROM (128K×8) and SRAM (128K×8)

271000 621000.

range → SRAM

19 18 17

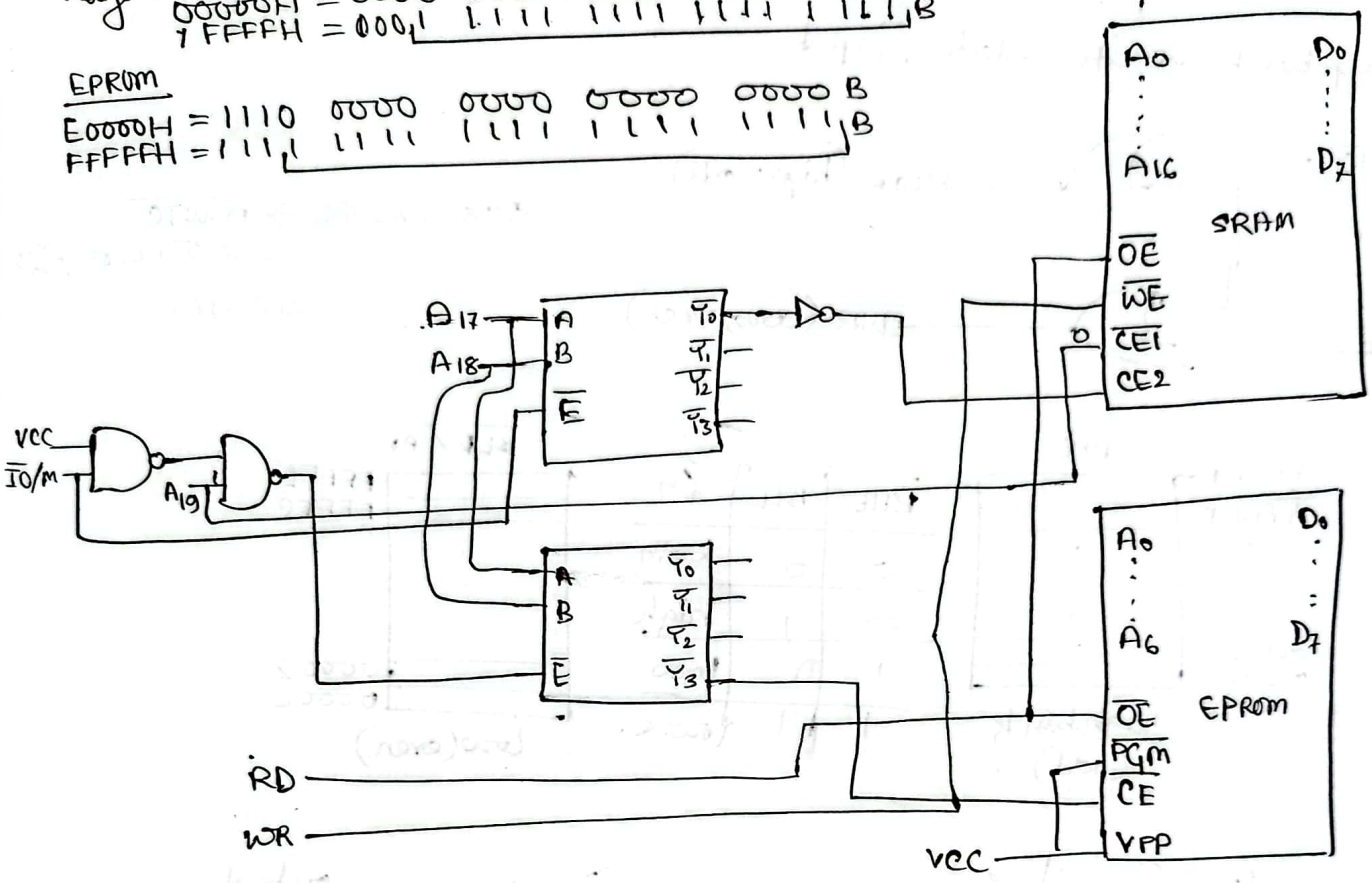
00000H = 0000 0000 0000 0000 0000 B

to FFFFH = 0001 1111 1111 1111 1111 B

EPROM

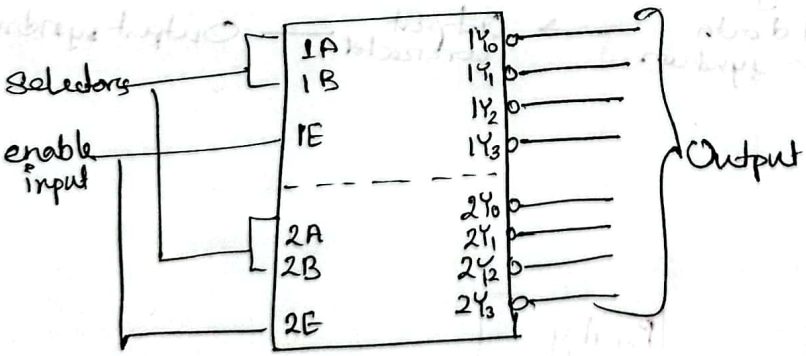
E0000H = 1110 0000 0000 0000 0000 B

FFFFFH = 1111 1111 1111 1111 1111 B



Dual 2 to 4 line Decoder :

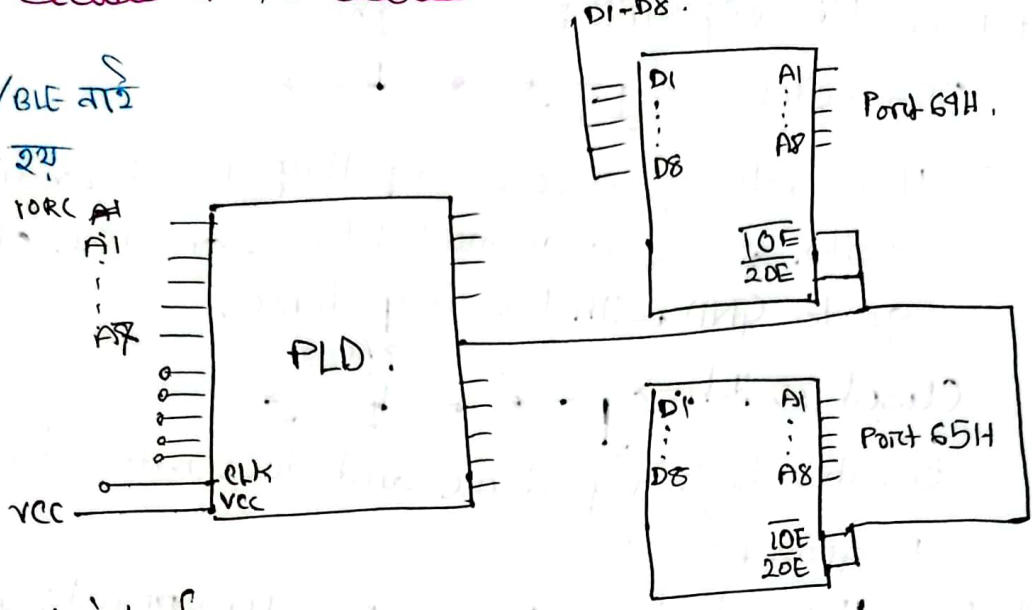
74LS139



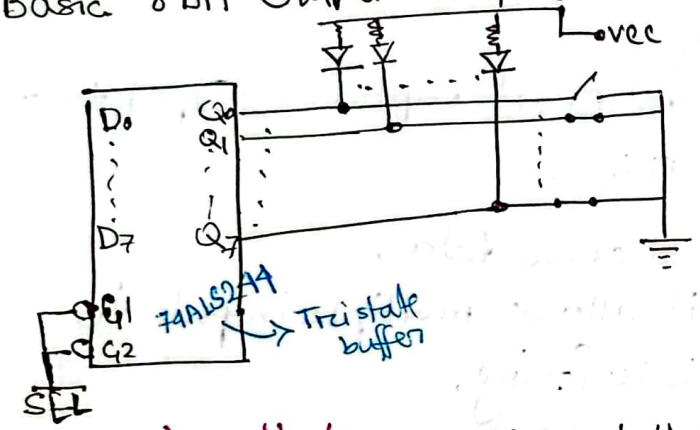
$\overline{E}$	A	B	$\overline{Y_0}$	$\overline{Y_1}$	$\overline{Y_2}$	$\overline{Y_3}$
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	X	X	1	1	1	1

✳ 16 bit wide port decoded at I/O addresses 64H and 65H

PLD - ला $\overline{BHE}$ or A_0/BLE नाही
so both bank select हवा



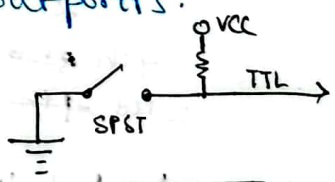
✳ Basic 8 bit Output interface.



Here SEL is activelow. When its 0, it decodes address for IO port and also reads operation.

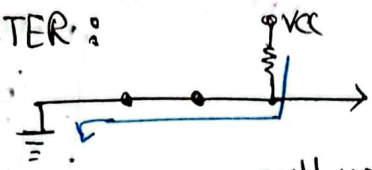
Here, when we give $Q_i = 0$ then ~~the~~ the led connected to that output lights up. In case the connect was ~~Q~~ ~~D~~, then we would have to give 1 to lighted the LED. Again when Q_0 is 1 but the gate is ~~open~~ closed power stays, for ~~closed~~ open. connections 1 becomes 0 as it is connected to GND.

Tri state buffer is used here to construct the input ports.
The external TTL data are inputs for buffers.
This ensures the output is held either 0 / 1 (never floats)

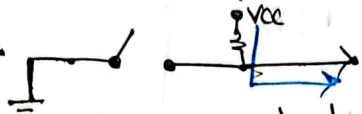


After µp gets IN instruction, I/O port address is decoded to generate $\overline{SEL}$ (active low).
So, G1 and G2 gets 0 and activate 74S244 buffer.
If ~~outputs get~~ inputs gets 1, the device enters tri state high impedance mode that disconnects switches from data bus.
Thus IN ins. helps contents of switches to be copied in A

** PULL UP REGISTER:

Open switch → 

Here, voltage accross a pull up resister vanished as switch wire is connected to Vcc and the ~~charges~~ voltage go to GND with low impedance.

Closed switch → 

Creates high impedance and does not affect the connection to ground

Together they help to derive an appropriate value for impedance of pull up register

OUT instruction, everytime SEL is 0, it captures output to latch
The data are held until next OUT ins.

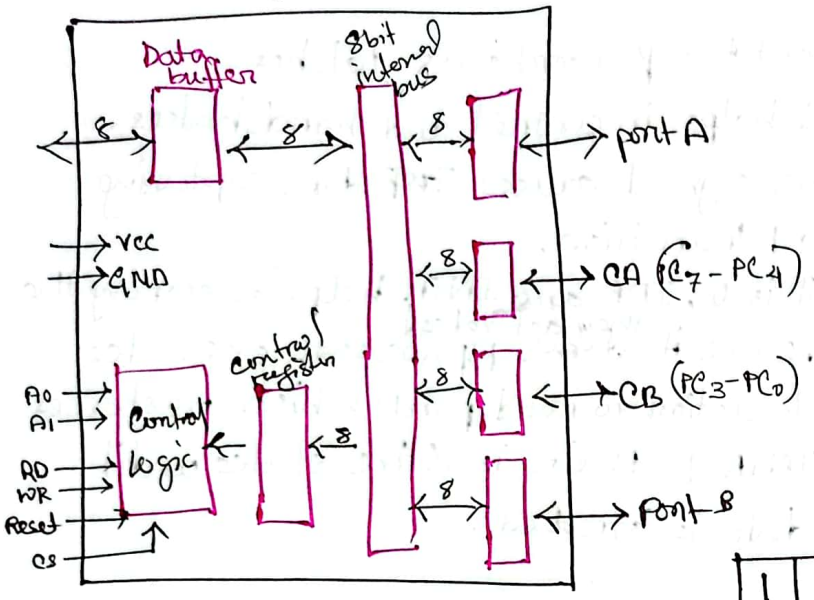
(*) LATCH used for an output port to store data.
(*) TRI state buffer for input port; it allows multiple inputs to share the same line without interfering.

Strobe Signal: Suppose there is 4 flipflops, current output is 1100 and next output needs to be 1010.

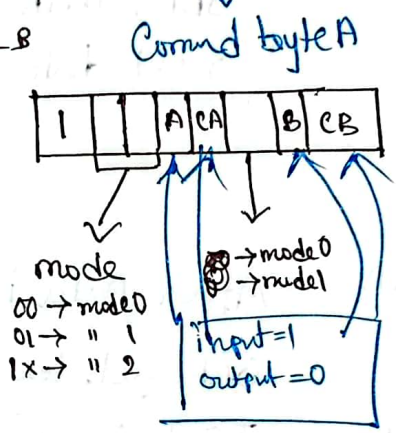
Here, 1st ff will ~~transforce~~ not need change.
2nd " " need a transformation from 1 to 0
4th ~~3rd~~ " don't need change
3rd ~~4th~~ needs to change 0 to 1.

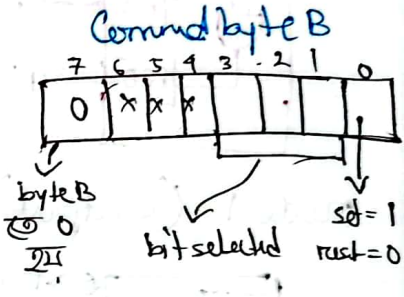
Here, to let the system know about ~~each~~ completion of each transformation one after another we need strobe signal. We send strobe after 1st ff tells you no need to change we again send strobe after 2nd ff completes transformation and so on.

82C55 Programmable Peripheral Interface (PPI)



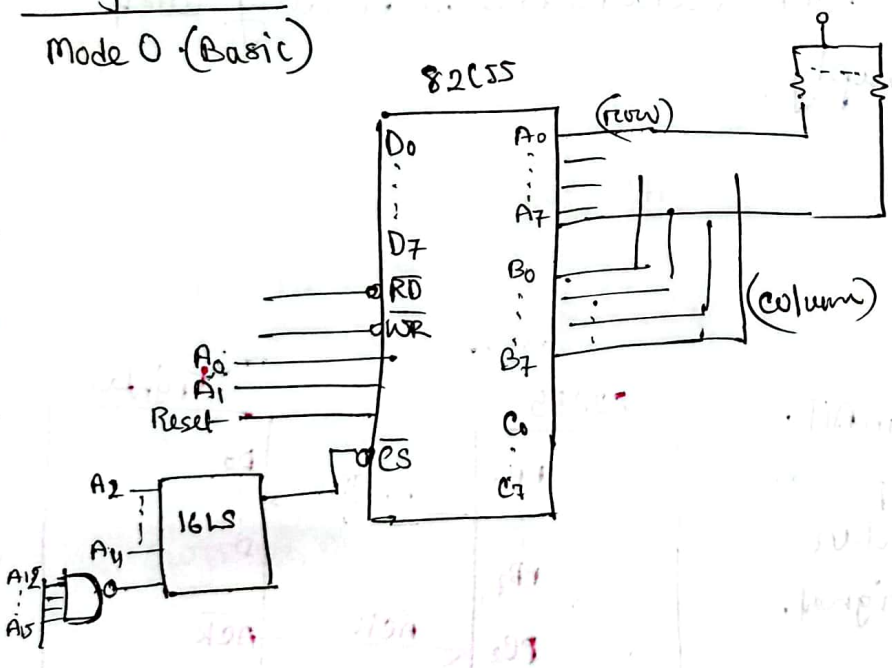
A1	A0	Function
0	0	port A
0	1	" B
1	0	" C
1	1	Command register



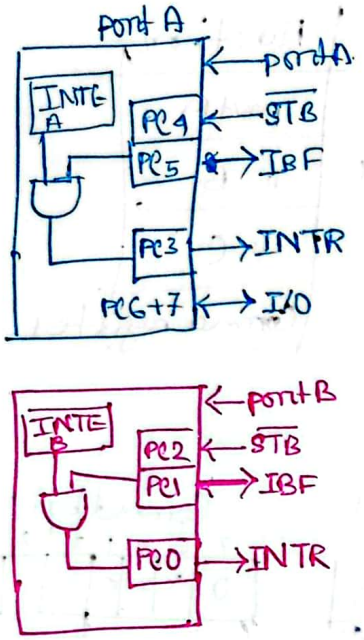


Keyboard:

Mode 0 (Basic)

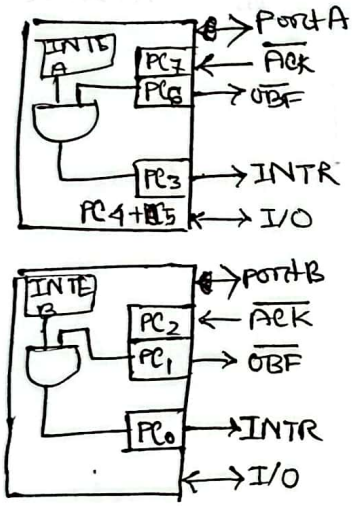


Mode 1 (keyboard input)



Here, port A & B works as latches. portC helps in control and handshakes. A strobe signal causes ~~IBF~~ data capturing in 0 to 1 transition.
When it is 0, IBF and INTR helps in storing the data. Once ~~it's done~~ they activated Microprocessor executes IN instruction to read ports which restores IBF, INTR to their inactive states until next data is strobed.

Mode 1 (Output)

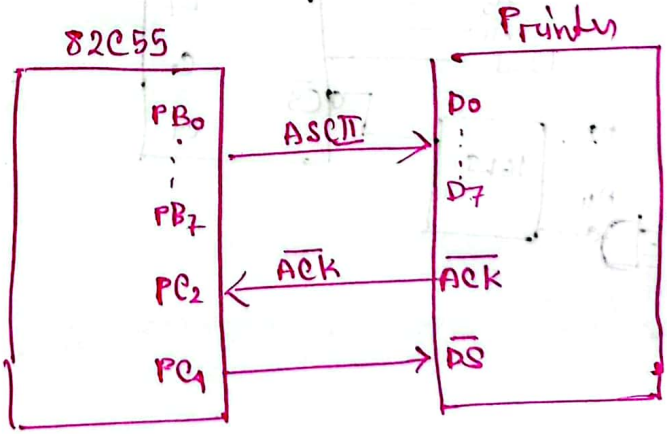


After Data is written in port Latch; $\overline{OBF}$ gets low indicating data are present. IO devices removes data by $\overline{ACK}$ (active low) and $\overline{OBF}$ returns to 1 indicating latch is empty.

Parallel Printer Interface:

ASCII coded characters are stored in AH. When $\overline{ACK}$ returns $\overline{OBF}$ to 1 procedure sends contents of AH to the printer through portB and sends $\overline{DS}$ signal.

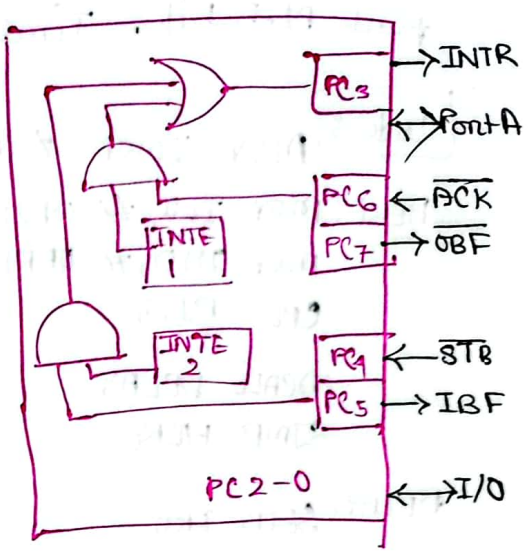
PC_4 generates $\overline{DS}$ signal. The $\overline{ACK}$ sends back Acknowledgement to 82C55.



Printer - এর Bidirectional Bus :-

Bus checks $\overline{OBF}$ if buffer is empty.
If empty data is send via OUT ins.
as soon as output circuitry sees $\overline{OBF}=0$ it sends back $\overline{ACK}$ to remove data from buffer and it sets $\overline{OBF}$ and enables tri state buffer to read data.

Here contents of AH is transmitted through bidirectional port A.



When IBF = 1, data is input using IN instruction. Data is sent by using $\overline{STB}$ signal. Data is held in port A latch. As IN ins. executes, IBF becomes 0 and data is moved to AL.

When INTE 1 and 2 both are enabled, output-input both buffers cause Interrupt requests, it occurs when data is Strobed by $\overline{STB}$ or when data are written by OUT.

Codes :

OUT (Write)

⇒ write 00H into Output port 62H

```
MOV AL, 00H ; AL-এ ডাটা নিচ্ছে
OUT 62H, AL ; 62H-তে AL out করছে
```

or,

```
MOV AL, 00H
MOV DX, 62H
OUT DX, AL
```

In (Read)

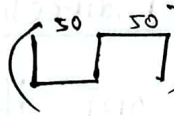
⇒ read a byte from input port 71H.

```
IN AL, 71H ; directly read
```

or,

```
MOV DX, 71H ; DX-এ ঠিকানা নিচ্ছে
IN AL, DX ; পড়ছে
```

Timer + Interrupt Codes:

(1) We need to create squre wave of 50% duty cycle 

on the ~~port~~ P1.5 bit. Timer 0 is used for delays. [XTAL = 11.0592 MHz]

Code:

```
MOV TMOD, #01
HERE: MOV TL0, #0F2H
      MOV TH0, #0FFH
      CPL P1.5
      ACALL DELAY
      SJMP HERE
DELAY: SETB TR0
AGAIN: JNB TF0, AGAIN
       CLR TR0
       CLR TF0
       RET
```

Explanation: (Analyze করতে বললে)

1. TMOD loaded, timer=0, mode=1
2. From HERE, we loaded F2H in TL0 and FFH in TH0. so, FFF2H → T
3. Then P1.5 is toggled for high and low pulses.
4. We called DELAY subroutine.
5. DELAY part starts with setting TR0 which means Timer 0 starts.
6. With the help of XTAL (crystal oscillator) → timer 0 counts with passing of each clock.
   After FFF2H starts FFF3H → FFF4H → FFF5H → FFF6H ..... upto FFFFH.
   Whereever TF0=1, JNB falls through.
7. timer 0 is stopped by CLR TR0.
   DELAY ends and process is repeated by ~~CLR~~ clearing both TR0 and TF0. from HERE. (We have to re load TL, TH too)

Delay: (calculate করতে বললে)

Last stoppage = FFFFH.
starts T from = FFF2H (TH+TL).

So, (FFFF − FFF2 + 1)H [∵ (last − starts + 1) formula]
= 000EH
= 14D

So, 14 × 1.085 μs [∵ clock period = 1.085 μs for 11.0592 MHz.]
= 15.19 μs

(প্রমাণ):
derivation চাইলে

freq = $\left(\frac{1}{12} \times 11.0592\right)$ MHz

period = $\frac{1}{\left(\frac{1}{12} \times 11.0592\right)}$

= 1.085 μs.

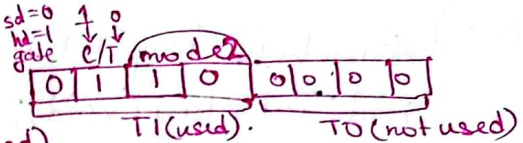
TMOD register-এর map

C/T = 1 → counter use করলে
C/T = 0 → Timer হিসেবে use হবে

(1) mode 01 → count 16 bit-এর হবে (FFFFH Last count)
(2) mode 10 → " 8 " " " (FFH " ")

Gate = 0 → software implementation
Gate = 1 → hardware "

(#) Suppos clock pulse is fed into T1, write program for counter1 in mode 2. to count pules and display the state of TL1 count on P2, which connects 8LEDs.



Code:

```
MOV TMOD, #0110 0000B.  ;
MOV TH1 , #0            ; TH1 = 0(cleared)
SETB P3.5.              ; making P3.5 input
AGAIN : SETB TR1        ; counter is set.
BACK : MOV A, TL1.      ; A = TL1.
       MOV P2, A        ; display A in Port 2.
       JNB TF1, Back    ; repeat back if TF1 = 0.
       CLR TR1          ; stops counting
       CLR TF1          ; TF1 = 0.
       SJMP AGAIN       ; Jump to AGAIN; repeat
```

(#) Write 8051 C program to toggle all bits of port1 with some delay. use Timer 0, 16 bit mode to generate delay. (delay time বলা নাই)

Code:

```
#include <reg51.h>
void TODelay(void);
void main(void).
{ while(1)
  { P1 = 0x55;  // toggle all bits of port1
    TODelay();  // delay
    P1 = 0xAA;  // toggle all bits
    ToDelay();  // delay
  }
}

void TODelay()
{ TMOD = 0x01;  // timer0, mode1 (16 bit).
  TL0 = 0x60;   // load TL0   | নিজে assume করে নিতে হবে
  TH0 = 0x35;   // load TH0
  TR0 = 1;      // TR0 counter turn on.
  while(TF0 == 0); // repeat until TF0 = 0.
  TR0 = 0;      // TR0 stops
  TF0 = 0;      // TF0 = 0.
}
```

(##) C program write — to toggle only P1.5 continously every 50ms.
Create delay by timer 0, mode 1. (Delay time দেওয়া → 50ms).

Code:

```
#include <reg51.h>
void ToDelay(void);
sbit mybit = P1^5;
void main(void)
{ while(1)
  { mybit = ~mybit; // toggle P1.5
    ToDelay();
  }
}
void ToDelay(void)
{
  TMOD = 0x01;  // timer 0, mode 1
  TL0 = 0xFD;
  TH0 = 0x4B;
  TR0 = 1;
  while (TF0 == 0);
  { TR0 = 0;
    TF0 = 0;
}
```

Given, 50ms. so, count = 50/1.085μs
= 46083D.

Last stop হয় FFFFH → we know, formula
(last − start + 1)

So, (last + 1) − start = 46083D
or, (FFFF + 1) − start = B403H
or, start = (FFFF + 1) − B403H
= 4BFDH.

So, TH0 → 4BH
TL0 → FDH

FFFF last count হয় cause
mode 1 বলা — mode 1 16 bit

(*) Write C program for 8051 to create 2500Hz frequency on pin P2.7. Use timer 1, mode 2 for delay.

Soln:

Timer 1, mode 2 so → | 0010 (T1) | 0000 (T0) | → 0x20 H

Given, freq = 2500Hz.

Time period = 1/2500Hz = 400 μs.

Ton = Toff = 400/2 = 200 μs (50% duty cycle হলে এরকম)

Count = 200/1.085 μs = 184.

mode-2 হলে 8 bit-এ so last count = FF H.

So, (last - start + 1) = 184D.

or, (FFH + 1) - start = B8H

or, start = (FF + 1 - B8) H

= 48H

Code:

```
#include<reg51.h>
void ToDelay(void);
sbit mybit=P2^7;
void main(void){
    unsigned character x.char x;
    while(1){
      mybit =~mybit;
      ToDelay();
    }
}
void ToDelay(void)
{ TMOD=0x20;
  TH1 = 0x48;   // 8bit এর TH-এ enough TL assign করা লাগবে না
  TR1=1
  while (TF1==0);
  TR1=0
  TF1=0;
}
```

Interrupt-IP instructions →

① enable serial interrupt → `MOV IE, #10010110B`
when timer 0,
external hardware
interrupt 1 (EX1)

② disable (mask) timer 0. → `CLR IE.1`

③ disable all interrupts → `CLR IE.7`

(#) Write a program (assembly) ~~con~~ gets data (8 bits) from P0 and sends to P1 while simultaneously creating square wave of 200 μs period on pin P2.1. Use timer 0.

Soln: As 8bits, mode 2 will be used.

timer = 0 ; so, TMOD = 0000 0010 → ~~0x20~~ 20H.

Here, period = 200μs.

So, Ton = Toff = 200/2 = 100μs.

Count = 100/1.085 μs = 92D = 5CH.

So, start count = FF + 1 − 5C = A4H

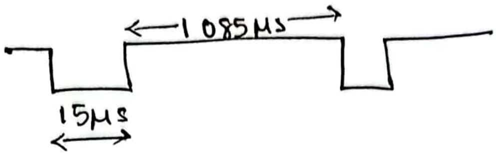
Code:

```
        ORG   0030H
MAIN:   MOV   P0, #FFH
        MOV   TMOD, #02H
        MOV   TH0, #A4H
        MOV   IE, #82H   ; enable timer 0; timer 1 এর জন্য 88H লিখতে
        SETB  TR0.
BACK:   MOV   A, P0
        MOV   P1, A
        SJMP  BACK
        END.
```

(*) Write a program (repeat earlier code ← বলে দিছে question-2) to create square as high portion 1085 μs and low of 15 μs. Use timer 1, XTAL = 11.0592 MHz

Soln:

15/1.085 = 14



Code!

```
        ORG 0030H

MAIN:   mov TMOD, #10H
        mov P0, #FFH
        mov TL1, #01AH
        mov TH1, #0FCH
        mov IE, #88H

        SETB TR1

BACK:   mov A, P0
        mov P1, A
        SJMP BACK

ISR_T1: CLR TR1
        CLR P2.1

        mov R2, #3

HERE:   DJNZ R2, HERE
        mov TL1, #18H
        mov TH1, #0FCH

        SETB TR1
        SETB P2.1

        RETI
        END
```

ই code বুঝি নাই ☺

Addressing Table:-

r/m	11 w=0(8)	11 w=1(16)	00 [ ]	01 (8 bits)	10 (16 bits)
0→000	AL	AX	BX+SI	BX+SI+08	BX+SI+16
1→001	CL	CX	BX+DI	BX+DI+08	BX+DI+16
2→010	DL	DX	BP+SI	BP+SI+08	BP+SI+16
3→011	BL	BX	BP+DI	BP+DI+08	BP+DI+16
4→100	AH	SP	SI	SI+08	SI+16
5→101	CH	BP	DI	DI+08	DI+16
6→110	DH	SI	Direct address	BP+08	BP+16
7→111	BH	DI	BX	BX+08	BX+16

Format → 100010 dw oo rrrr mmm disp. → for mov reg, reg
mov mem, reg
mov reg, mem.

Data Addressing Mode: (Fast)

(1) Register addressing

Exple: mov BL, AL

machine code → 100010 10 011 000
mov d w

(2) Immidiate addressing

format → 1011 w rrrr data.

example: mov AX, 5001H.

1011 1 000 0000 0001 0101 0000
w AX 01 50

mov BX, 23H

1011 1 0011 0010 0011 0000 0000.
w BX 23 00

mov CL, 23H

1011 0 001 0010 0011
w 23

Memory Addressing :

① Direct and Displacement

Destination Accumulator হলে mov opcode → 101000 dw data
" " register " " → 100010 dw mmmrrr.

example: mov AX, [1234H]

101000 | 1 (d) | 1 (w) | 0011 0100 (34) | 0001 0010 (12)

mov BL, [1234H]

101000 | 1 (d) | 0 (w) | 00 (mode) | 011 (BL) | 0011 0100 (34) | 0001 0010 (12)

② Register indirect addressing (format : 100010 dw mmrrrmmm disp)

example:
mov AX, [BX]

101000 100010 | 1 (d) | 1 (w) | 000 (mod) | 000 (AX) | 111 (BX(00)) ...

mov CX, [BX]

100010 | 1 (d) | 1 (w) | 00 (mode) | 001 (CX) | 111 (BX(00)) .....

③ Base Plus Index Addressing (100010 dw mmrrrmmm disp)

example: mov DX, [BX+DI] → data :
array DW 1234H, 1243H, 1111H.
code
mov DI, 8
mov BX, array.
mov DX, [BX,DI]
ret

100010 | 1 (d) | 1 (w) | 00010 (mod DX) | 001 ([BX+DI])

mov [BX+DI], BL

100010 | 0 (d) | 0 (w) | 00 (mode) | 011 (BL) | 001.

d → source; w → BL → 8bit

← শুধু just ২২ line change হবে

④ Register Relative Addressing:

example: mov DX, [BX+08H]

100010 | 11 | 01 | 010 | 111 | 0000 1000
d w | mode 8bit | DX | BX+08H | 08 data.

⑤ Base relative plus index:

example: mov AX, [BX+SI+160H]

100010 | 1 1 | 10 | 000 | 000 | 0000 0000 | 0000 0001
d w | mode 16bit | AX | BX+SI | 00 | 01

mov DX, [BX+DI+3H]

100010 | 1 1 | 01 | 010 | 001 | 0000 0011
d w | mode 8bit | DX | BX+DI | 03

⑥ Scaled Index. Addressing (80386 to above).

— two 32 bit reg (base + index).

multiplied by scaling factor like 1x, 2x(word), 4x(double word) 8x(quadword).

example: mov AX, [EDI + 2* ECX]

or, mov EAX, ARRAY[4*ECX]

⑦ RIP Relative Addressing:

— uses 64 bit instruction pointer register to 64 bit mode in flat memory.

possible to upgrade to a faster machine with more memory without sacrificing the investment in already-developed software. The characteristics of a family are as follows:

- **Similar or identical instruction set:** In many cases, the exact same set of machine instructions is supported on all members of the family. Thus, a program that executes on one machine will also execute on any other. In some cases, the lower end of the family has an instruction set that is a subset of that of the top end of the family. This means that programs can move up but not down.
- **Similar or identical operating system:** The same basic operating system is available for all family members. In some cases, additional features are added to the higher-end members.
- **Increasing speed:** The rate of instruction execution increases in going from lower to higher family members.
- **Increasing number of I/O ports:** The number of I/O ports increases in going from lower to higher family members.
- **Increasing memory size:** The size of main memory increases in going from lower to higher family members.
- **Increasing cost:** At a given point in time, the cost of a system increases in going from lower to higher family members.

How could such a family concept be implemented? Differences were achieved based on three factors: basic speed, size, and degree of simultaneity [STEV64]. For example, greater speed in the execution of a given instruction could be gained by the use of more complex circuitry in the ALU, allowing suboperations to be carried out in parallel. Another way of increasing speed was to increase the width of the data path between main memory and the CPU. On the Model 30, only 1 byte (8 bits) could be fetched from main memory at a time, whereas 8 bytes could be fetched at a time on the Model 75

10. Which flag bit controls the INTR pin on the microprocessor? I Flag
11. Which microprocessors contain an FS segment register? 80386
12. What is the purpose of a segment register in the real mode operation of the microprocessor?
13. In the real mode, show the starting and ending addresses of each segment located by the following segment register values:
    (a) 1000H SA = 10000H ~~limit~~ ending = 1FFFFH
    (b) 1234H starting = 12340H ; ending = 2233FH
    (c) 2300H starting = 23000H ; ending = 32FFFH
    (d) E000H starting = E0000H ; ending = EFFFFH
    (e) AB00H starting = AB000H ; ending = BAFFFH
14. Find the memory address of the next instruction executed by the microprocessor, when operated in the real mode, for the following CS:IP combinations:
    (a) CS = 1000H and IP = 2000H 10000 + 2000 = 12000H
    (b) CS = 2000H and IP = 1000H 21000H
    (c) CS = 2300H and IP = 1A00H 24A00H
    (d) CS = 1A00H and IP = B000H 25000H
    (e) CS = 3456H and IP = ABCDH 3F12DH
15. Real mode memory addresses allow access to memory below which memory address?
16. Which register or registers are used as an offset address for the string instruction destination in the microprocessor? DI, EDI, RDI
17. Which 32-bit register or registers are used to hold an offset address for data segment data in the Pentium 4 microprocessor? ~~EBX~~ RBX, RDI, RSI, RAX, REX, RDX.
18. The stack memory is addressed by a combination of the SS segment plus SP, BP offset.
19. If the base pointer (BP) addresses memory, the SS segment contains the data.
20. Determine the memory location addressed by the following real mode 80286 register combinations:
    (a) DS = 1000H and DI = 2000H 12000H
    (b) DS = 2000H and SI = 1002H 21002H
    (c) SS = 2300H and BP = 3200H 26200H
    (d) DS = A000H and BX = 1000H A1000H
    (e) SS = 2900H and SP = 3A00H 2CA00H
21. Determine the memory location addressed by the following real mode Core2 register combinations:
    (a) DS = 2000H and EAX = 00003000H 23000H
    (b) DS = 1A00H and ECX = 00002000H 1C000H
    (c) DS = C000H and ESI = 0000A000H CA000H
    (d) SS = 8000H and ESP = 00009000H 89000H
    (e) DS = 1239H and EDX = 0000A900H 1CC90H
22. Protected mode memory addressing allows access to which area of the memory in the 80286 microprocessor?
23. Protected mode memory addressing allows access to which area of the memory in the Pentium 4 microprocessor?
24. What is the purpose of the segment register in protected mode memory addressing?
25. How many descriptors are accessible in the global descriptor table in the protected mode?
26. For an 80286 descriptor that contains a base address of A00000H and a limit of 1000H, what starting and ending locations are addressed by this descriptor?
27. For a Core2 descriptor that contains a base address of 01000000H, a limit of 0FFFFH, and G = 0, what starting and ending locations are addressed by this descriptor?
28. For a Core2 descriptor that contains a base address of 00280000H, a limit of 00010H, and G = 1, what starting and ending locations are addressed by this descriptor?

holds segment address that defines the starting address of any 64KB memory segment.

SA = 01000000H
end = ~~end = limit~~ = 0100FFFFH
size = FFFFH
= 10000H

↳ actual limit = 00010FFFH

start = 00280000H
end = 00280000 + 00010FFF = 2900FFF H
size = 10FFF + 1H = 11000H

29. If the DS register contains 0020H in a protected mode system, which global descriptor table entry is accessed?
30. If DS = 0103H in a protected mode system, the requested privilege level is _____.
31. If DS = 0105H in a protected mode system, which entry, table, and requested privilege level are selected?
32. What is the maximum length of the global descriptor table in the Pentium 4 microprocessor?
33. Code a descriptor that describes a memory segment that begins at location 210000H and ends at location 21001FH. This memory segment is a code segment that can be read. The descriptor is for an 80286 microprocessor.
34. Code a descriptor that describes a memory segment that begins at location 03000000H and ends at location 05FFFFFFH. This memory segment is a data segment that grows upward in the memory system and can be written. The descriptor is for a Pentium 4 microprocessor.
35. Which register locates the global descriptor table?
36. How is the local descriptor table addressed in the memory system?
37. Describe what happens when a new number is loaded into a segment register when the microprocessor is operated in the protected mode.
38. What are the program-invisible registers?
39. What is the purpose of the GDTR?
40. How many bytes are found in a memory page?
41. What register is used to enable the paging mechanism in the 80386, 80486, Pentium, Pentium Pro, Pentium 4, and Core2 microprocessors?
42. How many 32-bit addresses are stored in the page directory?
43. Each entry in the page directory translates how much linear memory into physical memory?
44. If the microprocessor sends linear address 00200000H to the paging mechanism, which paging directory entry is accessed, and which page table entry is accessed?
45. What value is placed in the page table to redirect linear address 20000000H to physical address 30000000H?
46. What is the purpose of the TLB located within the Pentium class microprocessor?
47. Using the Internet, write a short report that details the TLB. *Hint:* You might want to go to the Intel Web site and search for information.
48. Locate articles about paging on the Internet and write a report detailing how paging is used in a variety of systems.
49. What is the flat mode memory system?
50. A flat mode memory system in the current version of the 64-bit Pentium 4 and Core2 allow these microprocessors to access _____ bytes of memory.