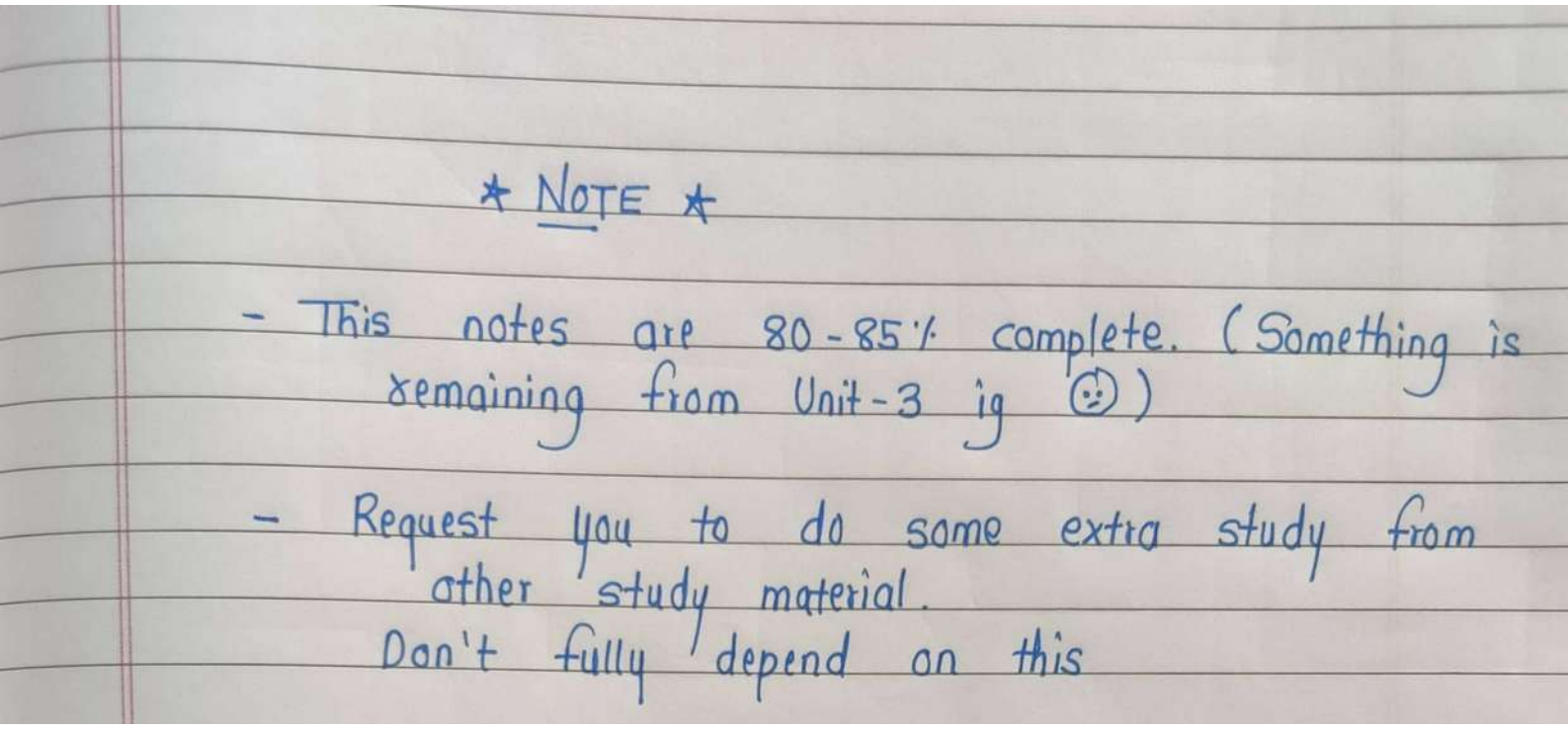

* NOTE *
- This notes are 80-85% complete. (Something is remaining from Unit-3 ig ☺)
- Request you to do some extra study from other study material.
Don't fully depend on this

Memory Management

★ Call Gate Descriptor -

→ • A gate descriptor is special type of descriptor used for performing protection checks

• They also control access to entry points within target code segment when control transfer instructions are executed

• Call gate is one of the types of gate descriptor

• They are used to call to function/ procedure which is of another privilege level i.e it can change privilege levels

★ Segment Descriptor -

→ • Diagram -

Segment Base 15.... 0					Limit 15-0						
Base	G	D	0	AV	Limit	P	DPL	S	TYPE	A	BASE
31... 24				L	19...16	7	6|5	4	3|2|1	0	23..16

- Segment descriptor is a special structure which describes segment.
- Exactly, one segment descriptor must be defined for each segment in memory

- From the Diagram,

BASE - Base address of segment
LIMIT - Length of segment
P - Present bit : 1 = Present ; 0 = Not present
DPL - Descriptor Privilege Level 0-3
S - Segment descriptor : 0 = System descriptor
1 = Code / Data segment desc.

TYPE - Type of segment
A - Access bit
G - Granularity bit : 1 = Segment length is page granular
0 = Segment length is byte granular

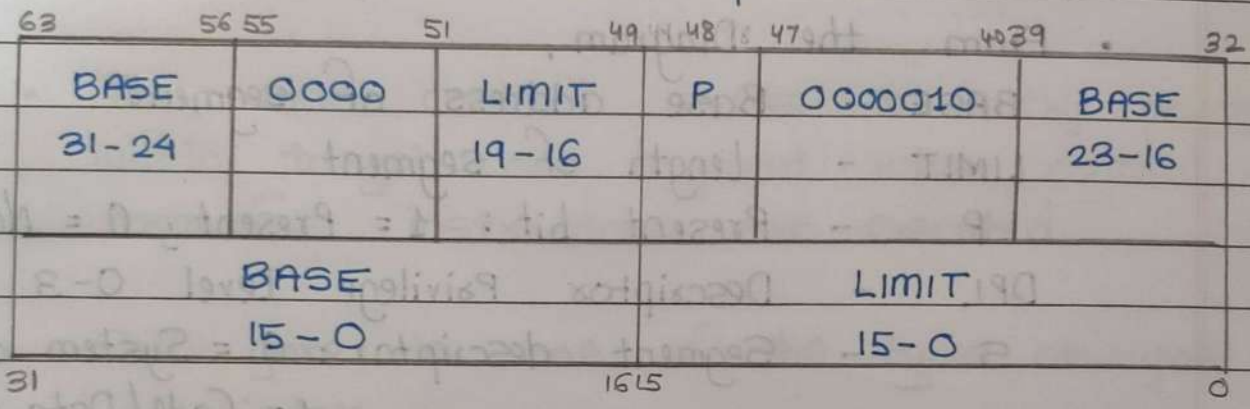
0 - Bit must be zero (0) for compatibility

AVL - Available field for user or OS

* LDT Descriptor –

→ • The LDT (Local Descriptor Tables) are set up in a system for individual task or closely related group of tasks

• Structure of LDT Descriptor –



• For the LDT descriptor, 'S' bit is 0 and 'type' bits are 2

• LDTR (Local Descriptor Register) selects LDT descriptor from GDT

• LDT descriptors can only be accessed, if privilege level is 0

★ Descriptor Tables - GDT, IDT & LDT :

→ • Segment descriptors are grouped and placed one after other in contiguous memory locations.

• This arrangement is known as Descriptor Table.

There are 3 types of Descriptor Tables:

i) GDT - (Global Descriptor Table)

• It is general purpose table of descriptors, that can be used by all programs to refer segments of memory.

• GDT can have any type of segment descriptors except for descriptor which are used for serving interrupts

• It addresses up to 512 MB of virtual address space.

ii) IDT - (Interrupt Descriptor Table)

• It holds segment descriptors that define interrupt or exception handling routines.

• IDT has 24 bit Base Address and 16 bit Limit

• It can handle up to 256 interrupt descriptor

iii) LDT - (Local Descriptor Table)

- They are set up in system for individual task or closely related group of tasks

- LDT has 24 bit Base Address and 16 bit Limit. It has 512 MB private address space

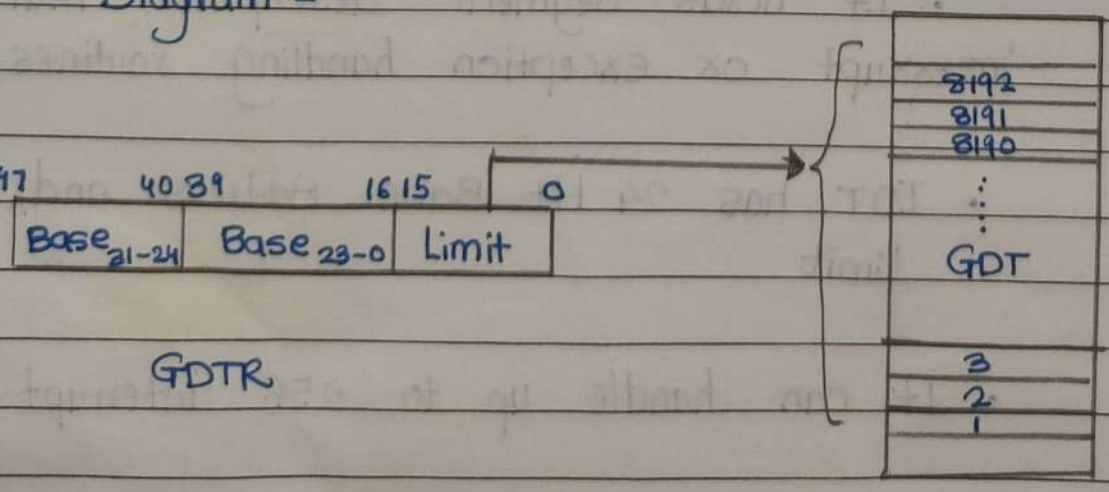
- LDT is essential to implement seperate address spaces for multiple processes.

★ Descriptor Registers - GDTR, LDTR & IDTR -

→ 1) GDTR -

- Global Descriptor Table Register is a 48-bit register located inside x86DX
- Lower two bytes of this register specifies the limit for GDT
- Upper four bytes of GDTR specifies 32-bit linear address of base of GDT.

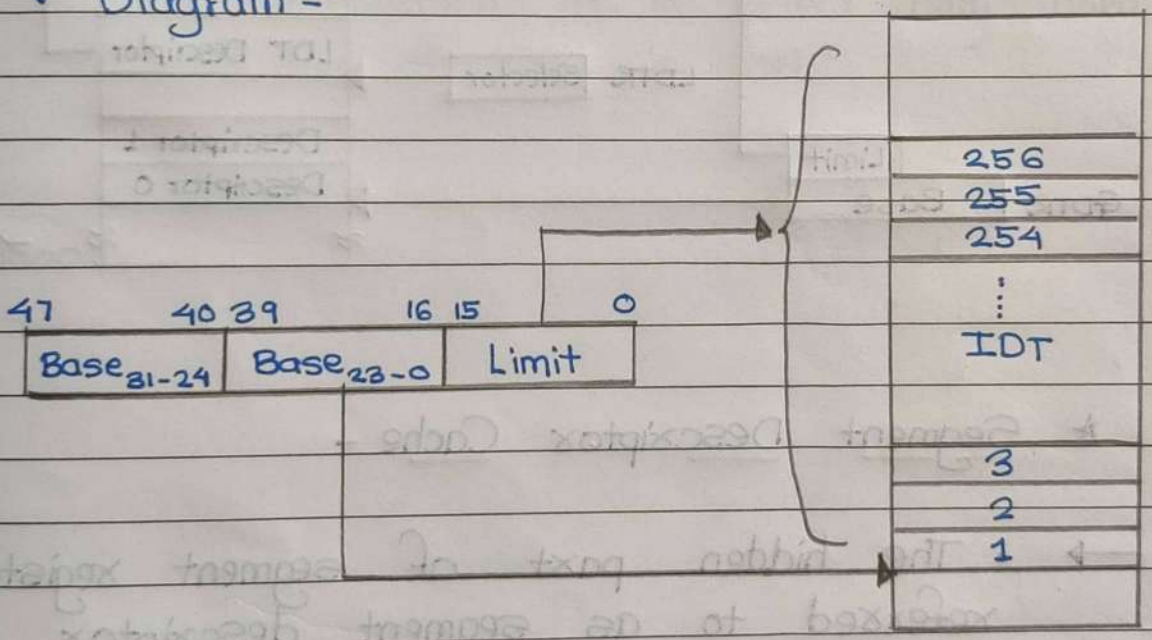
- Diagram -



2) IDTR -

- Interrupt Descriptor Table Register holds the 16-bit Limit and 32-bit Linear Address of base of IDT
- IDTR is 48 bit in length, with lower 2 byte defining Limit and upper 4 byte defining base address
- Size of IDT should not be set to support more than 256 interrupts.

- Diagram -

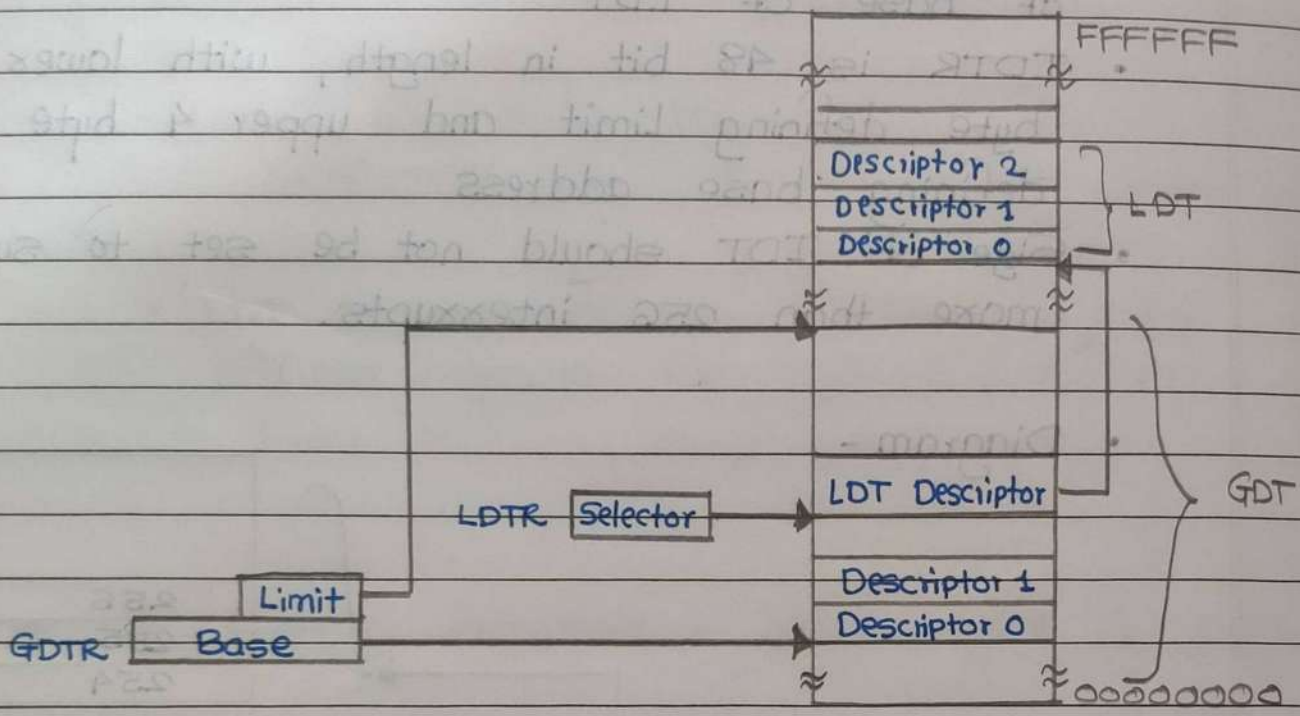


3) LDTR -

- Local Descriptor Table Register is a 16-bit register
- It specifies the address of LDT descriptor stored in GDT
- LDTR holds a selector that points to an LDT descriptor in GDT
- Whenever a selector is loaded into LDTR

corresponding descriptor is located in GDT.

- Diagram –



★ Segment Descriptor Cache –

→ • The hidden part of segment register is referred to as segment descriptor cache register.

- Using these registers, 80386DX stores information from descriptors, thereby avoiding need to consult descriptor table every time.

- Segment descriptor cache register contents are manipulated by processor.

★ Page Translation -
Addxess convexsion -

→ • Page translation is second phase of addxess translation

• In this phase, x86 transforms lineax address generated by segment translation into a physical addxess

• Page translation is in effect only when PG bit of CR0 is set.

UNIT - IV

PROTECTION

★ Four level of Hierarchical protection / five aspects of protection in 80386 DX :

→ • 80386 uses segment level protection and page level protection mechanism to protect critical sections.

• Protection in 86DX has 5 aspects :
1) Type Checking
2) Limit Checking
3) Restriction of Addressable Domain
4) Restriction of Procedure Entry points
5) Restriction of Instruction Set

• The concept of privilege applies to both, segment protection & page protection

★

▲ Definitions :

i) RPL -
- Requestor Privilege level
- It is the privilege level of original task that supplies the selector
- RPL is the 2 LSB of selector

↑ Least Significant Bit

ii) DPL -

- Descriptor Privilege Level
- It is least privilege level at which a task may access that descriptor and the segment associated with that descriptor
- Bits 6 & 5 of access rights byte of descriptor determine the DPL

iii) CPL -

- Current Privilege Level
- The privilege level at which a task / code segment is being currently executed
- Normally, CPL = DPL of segment that processor is currently executing

iv) EPL -

- Effective Privilege Level
- When access to new memory segment is desired, EPL is computed.
- It is defined as,

  EPL = max { RPL, CPL } (numerically)

* Privilege Transfer using Call Gate Descriptors -

→ • Call gate is only mechanism that allows to call a procedure located in any segment which has a higher privilege level

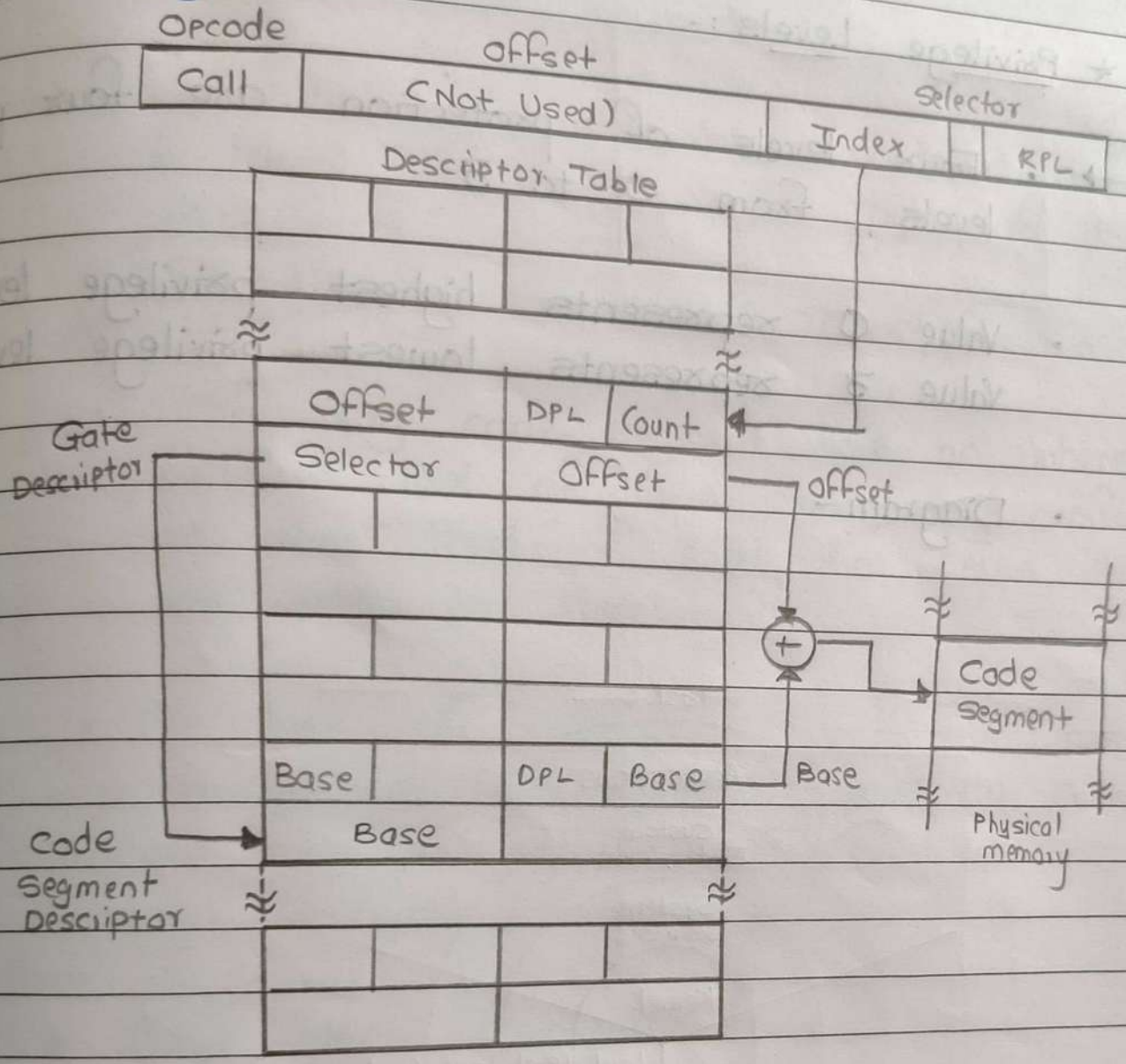
• Call gate descriptors are used by 'call' and 'jump' instructions in same way as code segment descriptors

• It contains 2 important things:
  a) Selector (which points to descriptor for segment where procedure is loaded)
  b) offset of called procedure in segment

• If call is valid, then selector is placed in visible portion of CS register and corresponding segment descriptor is placed in hidden portion of CS register

• The '86DX then uses base address from segment descriptor and offset from call gate to calculate physical address of called procedure

- Diagram –



- During this process, validity of control transfer is checked,
For valid control transfer, following privilege rules for CALL must be satisfied

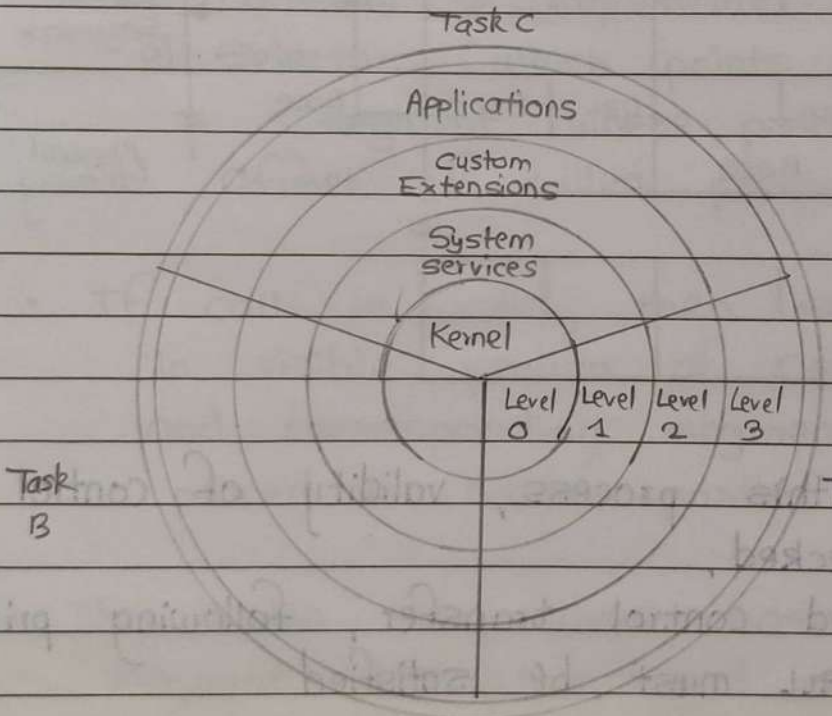
Target DPL ≤ max(RPL, CPL) ≤ Call gate DPL.

★ Privilege Levels :-

→ • Four levels of protection are four privilege levels, from 0 to 3

• Value 0 represents highest privilege level, Value 3 represents lowest privilege level

• Diagram -



• It shows that operating system Kernel is assigned to highest privilege level.

• System services such as BIOS procedures are assigned with PL1, whereas custom device drivers with PL2 and application programs with PL3

★ Conforming Code Segment -

→ • A code segment is considered conforming if bit 2 of access rights byte of its descriptor is set.
(C = 1 for conforming ; C = 0 for non-conforming)

• Conforming code segment have no inherent privilege level of their own ; they confirm to that level of code that CALLS them or JMPs to them.

• The DPL of conforming descriptor must always be less than or equal to current CPL.

★ Page Level Protection -

→ ▪ Aspects :

• Page level protection involves two kinds of protections.

1) Restricting Addressable Domain -
• U/S bit of PDE is 0 for OS, It is supervisor level. When x86 is executing at supervisor level, all pages are addressable
• If U/S bit is 1, x86 is executing at user level wherein only pages belonging to user are addressable

2) Type Checking -

- When x86 is executing at supervisor level, all pages are assigned with R|W access
- When executing at user level, access depends on R|W bit in PDE & PTE fields
- If R|W bit is 1, pages are only readable
  If R|W bit is 0, pages are both readable & writeable.

* Significance of IOPL in x86 -

→ • In IOPL mechanism, for execution of IN, INS, OUT, OUTS, CLI, STI instructions, the CPL of procedure / task must be same or lower than no. represented by IOPL bits i.e (CPL ≤ IOPL)

- Each task has its own unique copy of flag register.
- ~~Thus, each task has its own unique copy of flag regi~~
- Thus, each task can have different IOPL. The task can change IOPL.

★ I/O Permission Bit Map -

→ • I/O permission bit map is mechanism for protection I/O ports from unauthorized access

• It allows ports to be associated only with specific tasks.

• I/O Permission Bit Map is a bit vector. Size of map & its location in TSS are variable

• Each bit in the map corresponds to I/O port byte address

• To access I/O port, the corresponding bit in I/O bitmap must be 0

★ Privileged Instructions -

→ • Privileged Instructions are those that affect segmentation and protection mechanism, alter interrupt flag, or perform peripheral I/O.

• These instructions are divided in 2 groups

1) Privileged Instructions -
   • The instructions that affect system data structures come under this.
   • These must be executed when CPL is 0

- Example,

HLT	Halts the processor
LGDT, LLDT, LIDT	Loads GDT, LDT, IDT register
LTR	Loads Task Register
LMSW	Loads machine Status Word

2) IOPL Sensitive Instructions -

- IOPL field in Flag Register defines the right to use I/O instructions.
- These must be executed when (CPL ≤ IOPL)

- Example,

CLI	Disables interrupts
STI	Enables interrupts
IN, INS	Inputs data from I/O port
OUT, OUTS	Outputs data to I/O port

UNIT- V

MULTITASKING

★ Multitasking -

→ • Multitasking is ability of computer to run more than one program/task at same time

• The hardware support for multitasking is given by 80386.

• 80386 has special registers for implementing multitasking. These are:
  i) Task State Segment (TSS)
  ii) Task State Segment Descriptor
  iii) Task Register
  iv) Task Gate Descriptor

• With these registers, 80386 switches execution from one task to another task, saving environment of current task

• Apart from simple task switch, '86 supports two other task management features:
  i) Interrupts
  ii) Exceptions

★ Task State Segment :-

→ • Diagram –

31 – 23	15 – 7	0	
IIO Map Base	00000000 00000000	T	64
00000000 00000000	LDT		60
00000000 00000000	GS		5C
00000000 00000000	FS		58
00000000 00000000	DS		54
00000000 00000000	SS		50
00000000 00000000	CS		4C
00000000 00000000	ES		48
EDI			44
ESI			40
EBP			3C
ESP			38
EBX			34
EDX			30
ECX			2C
EAX			28
EFLAGS			24
EIP			20
CR3			1C
00000000 00000000	SS2		18
EIP2			14
00000000 00000000	SS1		10
EIP1			0C
00000000 00000000	SS0		8
EIP0			4
00000000 00000000	Back Link		0

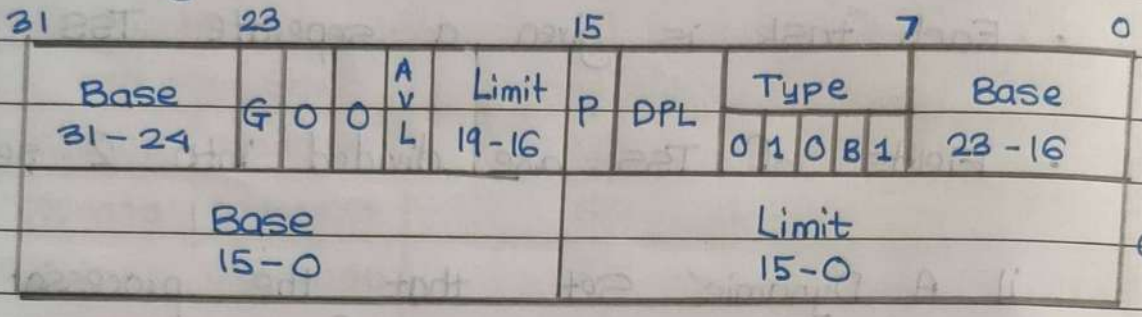
- The TSS stores entire context of task when task switching takes place.
- Each task is given a seperate TSS
- Fields of TSS are divided into 2 sets:

i) A Dynamic Set that the processor updates with each switch from task
This set includes:
   ⓐ General Registers (EAX, EBX, ECX, etc)
   ⓑ Segment Registers (CS, SS, DS, ES, FS, GS)
   ⓒ Flag Registers (EFLAGS)
   ⓓ Instruction pointer (EIP)
   ⓔ Back Link

ii) Static set that processor reads but does not change. This set includes:
   ⓐ Selector of task's LDT
   ⓑ Register (PDBR) that contains base address of task's page directory
   ⓒ Pointers to stacks
   ⓓ The T-bit (Debug Trap bit)

* TSS Descriptor :-

→ • Diagram :-

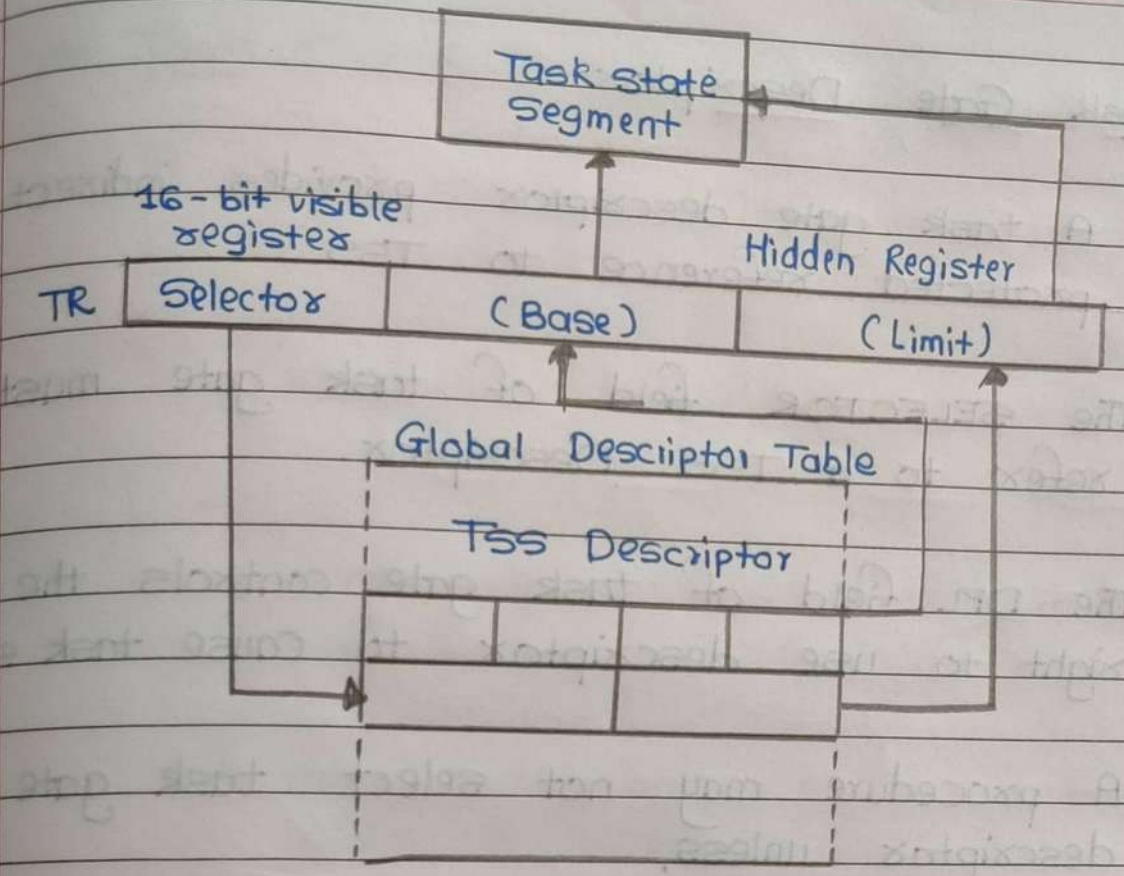


- B-bit in type field indicates whether task is busy
- BASE, LIMIT, DPL, G-bit, P-bit have functions similar to other descriptors
- To access TSS Descriptor, procedure must have privilege level less than / equal to privilege level specified by DPL field

* Task Register :-

→ • The Task Register (TR) identifies the currently executing task by pointing to TSS.

• Diagram :-



- '86 gives two instructions to read & modify visible portion of task

i) LTR (Load Task Register) -
It loads visible portion of task with selector and invisible portion with information from TSS descriptor selected by selector

ii) STR (Store Task Register) -
It stores visible portion of task register in general register or memory word.

* Task Gate Descriptor -

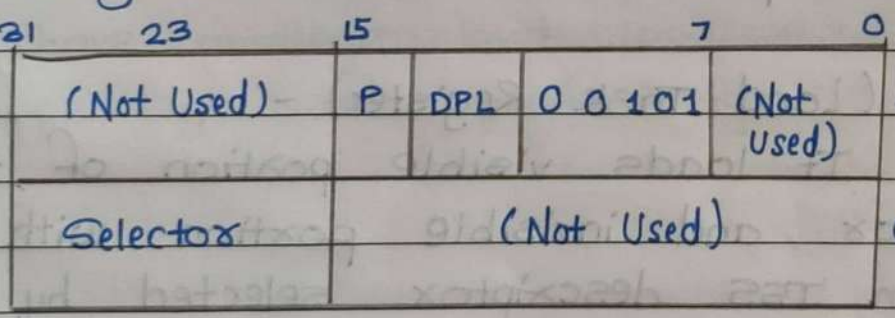
→ • A task gate descriptor provides indirect, protected reference to TSS

• The SELECTOR field of task gate must refer to a TSS Descriptor.

• The DPL field of task gate controls the right to use descriptor to cause task switch

• A procedure may not select task gate descriptor unless,
max (RPL, CPL) <= DPL

• Diagram -



★ Task Switching –

→ i) Steps involved in Task Switching (Without Task Gate)-

- Privilege Check:
  If DPL >= max(CPL, RPL), then only current task is allowed to switch
- Limit & Present bit Check:
  TSS descriptor for designated task is checked for its limit & presence
- Saving the state of current task:
  x86 copies cached registers into current TSS. Selector for current task is saved as back link selector in new task.
- Loading of Task Register:
  Visible portion of task register is loaded with selector of designated task's TSS descriptor
- Resuming Execution:
  Finally, x86 starts execution of designated task, with instructions pointed by new content of CS & EIP

ii) Steps involved in Task Switching (Using Task Gate)-

- Privilege Check of Task Gate –
  If DPL >= max(CPL, RPL), current task is allowed to switch

- Remaining steps are similar except that, for loading selector for TSS into TR, task gate is referred instead of CALL / JMP.

★ Task Linking (Nested Tasks)

→ • The back-field link of TSS and the NT (Nested Task) bit of flag word together allow x86 to automatically return to task that CALLed another task

- When a CALL instruction, interrupt instruction, etc causes switch to new task, x86 automatically fills back-link of new TSS with selector of current TSS & at same time, sets NT bit
- The NT flag indicates whether back-link field is valid.
- New task releases control by executing IRET instruction. When interpreting IRET, x86 examines NT flag
- If NT is set, x86 switches back to task selected by back-link field.

* Task Address Space -

→ • By appropriate choice of segment & page mappings for each task, tasks may share address spaces

• Ability for tasks to have distinct address spaces is important aspect of x86 protection

• Module in one task cannot interfere with module of another task, if modules do not have access to same address spaces.

* Virtual 8086 Mode -

→ • When processor starts / resets it operates in Real Mode.
In Real Mode, x86 executes 8086 instructions

• But once, processor switches from real mode to protected mode, it cannot return back to real mode until RESET

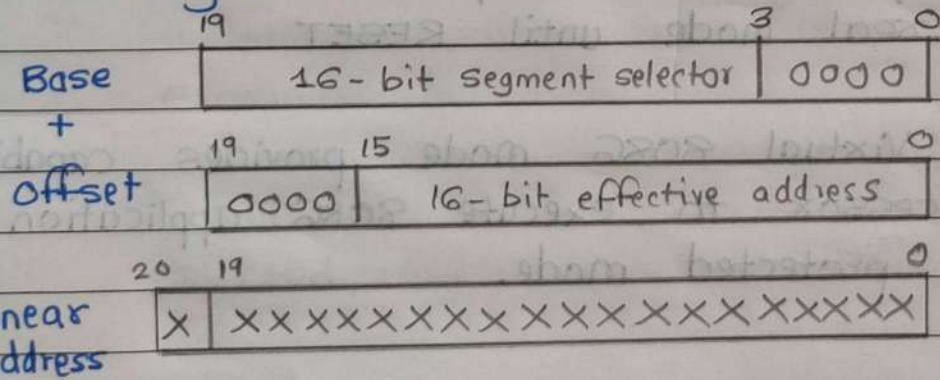
• So, Virtual 8086 mode provides capabilities to processor to execute 8086 application while in protected mode.

- Features of Virtual 8086 -
  - x86 operating in protected mode can easily switch back-forth to virtual 8086 mode
  - x86 allows one or more 8086, 8088, etc program execution as different task in VM mode
  - Address range of virtual mode task is 1 MB

* Linear Address Formation in VM -

→ • In Virtual 8086 mode, segment register contents are used to form linear address with help of offset.

- The linear address is formed by adding contents of appropriate segment register which are shifted left by 4 bit, to an effective offset.
- Diagram -



- If there is carry generated after addition, then unlike 8086, resulting 21 bit address is linear address.

★ Entering & Leaving Virtual Mode -

→ ▪ Entering Virtual Mode -
- The processor enters into Virtual 8086 mode from protected mode by setting VM bit of EFLAG.
- The processor enters into virtual mode by executing IRET instruction at CPL = 0 or a task switch at CPL.
- Virtual 8086 mode executes all programs at privilege level 3

▪ Leaving Virtual mode -
- The processor leaves the Virtual 8086 mode when an interrupt or exception occurs
- The Interrupt Service Routine (ISR) reset VM bit to enter the processor back to protected mode.

★ Difference – Real Mode v/s Protected Mode – (Also Virtual)

	→ Real Mode	Protected Mode
i)	Memory Addressing up to 1 MB physical memory	i) Memory Addressing up to 16 MB physical memory
ii)	No virtual memory support	ii) Supports up to 64 TB of virtual memory
iii)	Memory protection mechanism is not available	iii) Memory Protection Mechanism is available
iv)	Does not support virtual address space	iv) Gives virtual & physical address space
v)	Does not support LDT and GDT	v) Supports LDT & GDT
vi)	Segment Descriptor Cache not available	vi) Segment Descriptor Cache available
vii)	Support segmentation	vii) Support segmentation and paging.

UNIT - VI

Interrupts & Exceptions, Microcontroller

- **Definition -**

i) **Faults :**
- Faults are either detected before instruction begins to execute, or during execution of instruction
- If detected, during instruction, machine is restored to a state that permits instruction to restart.

ii) **Traps :**
They are reported at instruction boundary immediately after instruction in which exception was detected

iii) **Aborts :**
Used to report severe errors such as hardware errors & illegal values in system table. It neither permits precise location of instruction nor restart the program

★ Enabling & Disabling Interrupts -

→ • Certain conditions and flag settings cause processor to inhibit certain interrupts & exceptions. These are:

1) NMI masks further NMIs:
While NMI handler is executing, processor ignores further interrupt signals at NMI pin until next IRET instruction is executed.

2) IF masks INTR:
IF flag controls acceptance of external interrupt signalled via INTR pin.

3) RF masks Debug Faults:
The RF bit in EFLAGS controls recognition of debug faults.

4) Mov or POP to SS masks some Interrupts:
Software that need to change stack segments often use pair of instructions, eg MOV SS, AX.

★ Interrupt Descriptor Table -

→ • In protection mode, each interrupt / exception is associated with descriptor which gives information about ISR.

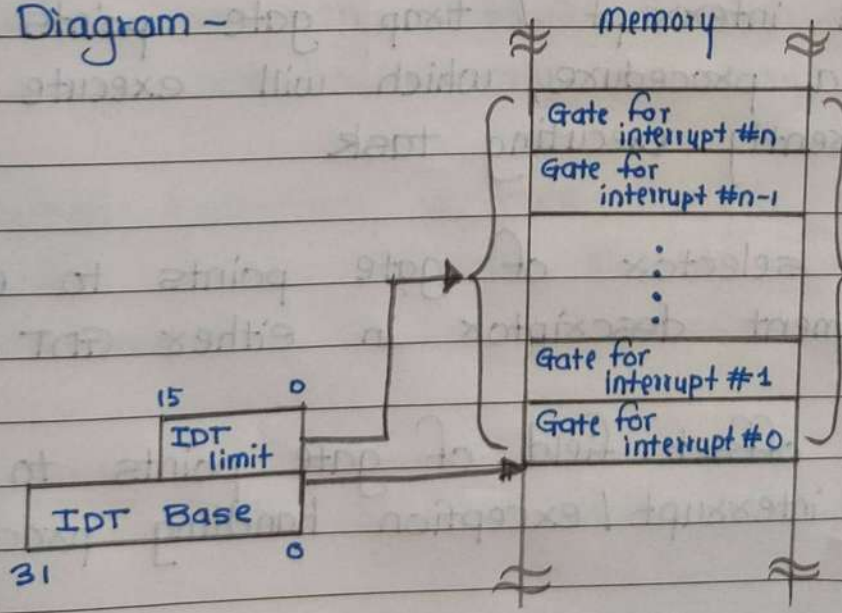
• These descriptors are stored in special descriptor table called Interrupt Descriptor Table (IDT)

• IDT may only contain task gates, interrupt gates & trap gates

• IDT should be atleast 256 bytes in size in order to hold descriptors for 32 Reserved Interrupts

• Processor locates IDT by means of IDT register (IDTR)
Instructions LIDT & SIDT operate on IDTR.

• Diagram -



Not Neccessary to Revise, If you can remember then only revise

- Interrupt Gate Descriptor – ~~Table~~

31 23	15		7	0	
(Not Used)	P	DPL	00101	(Not Used)	4
SELECTOR	(Not Used)				0

- Trap Gate Descriptor –

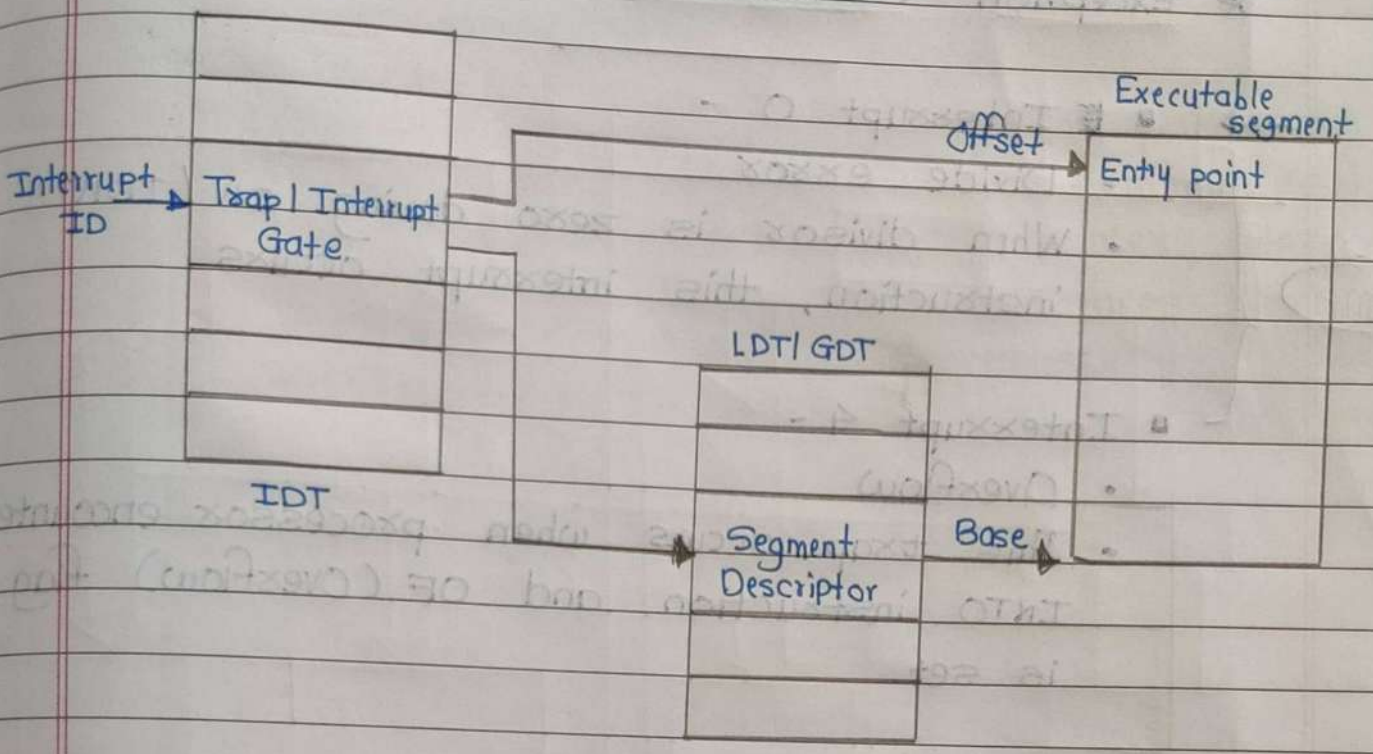
31 23	15		7	0	
Offset 31...16	P	DPL	00101	000 (Not Used)	4
SELECTOR	Offset 15....0				0

★ Interrupt Procedure –

→ • An interrupt / trap gate points indirectly to a procedure which will execute in the currently executing task

- The selector of gate points to executable-segment descriptor in either GDT or current LDT.
- The offset field of gate points to beginning of interrupt / exception handling procedure.

• Diagram -



★ Difference between Trap Gate & Interrupt Gate -

→ • Trap gate operates exactly like interrupt gate in all respect except one.

• When exception vectors through trap gate, all flags remain exactly as they were.

• When exception vectors through interrupt gate, x86 resets IF to disable further hardware interrupts.

★ Exception Conditions -

→ IMP

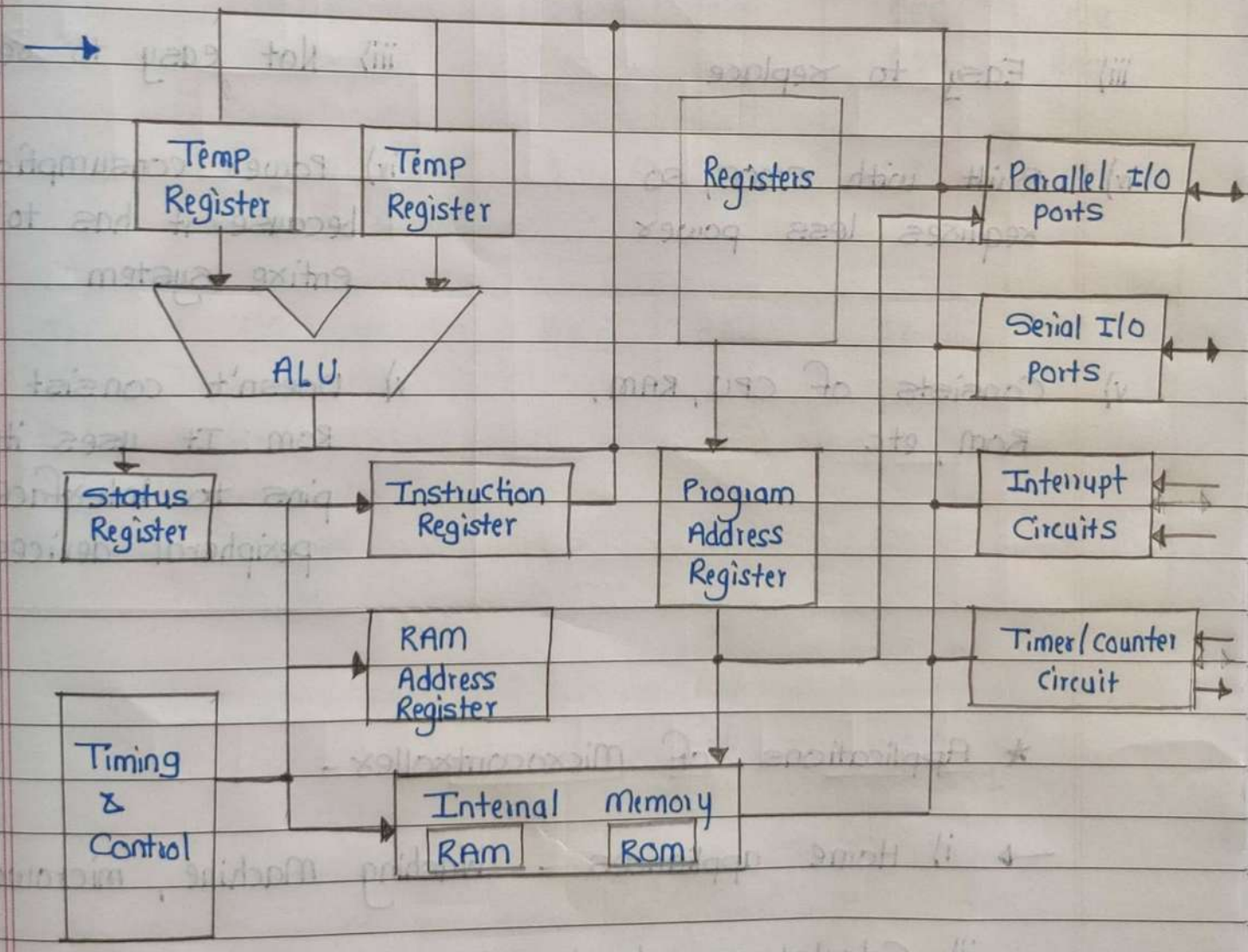
- Interrupt 0 -
  - Divide error
  - When divisor is zero during DIV / IDIV instruction, this interrupt occurs
- Interrupt 4 -
  - Overflow
  - This trap occurs when processor encounters INTO instruction and OF (Overflow) flag is set.
- Interrupt 2 -
  - This is only hardware interrupt assigned permanent vector
  - NMI
- Interrupt 10 -
  - Invalid TSS
  - Occurs if during task switch, the new TSS is invalid.

- **Definition :-**

Microcontroller -

A device which contains microprocessor with integrated peripherals like memory, serial ports, parallel ports, timer/counter, interrupt controller, data acquisition interfaces like ADC, DAC is called microcontroller

★ **Block Diagram of Microcontroller -**



★ Difference - Microcontroller v/s Microprocessor

→

Microcontroller	Microprocessor
i) They are used to execute a single task within an application	i) Microprocessor are used for big application
ii) Designing & Hardware cost is low	ii) Designing & Hardware cost is high
iii) Easy to replace	iii) Not easy to replace
iv) Built with CMOS, so requires less power	iv) Power consumption is high because it has to control entire system
v) Consists of CPU, RAM, Rom, etc	v) Doesn't consist RAM, Rom. It uses its pins to interface to peripheral devices

★ Applications of Microcontroller -

→ i) Home appliances - Washing Machine, microwave, etc

ii) Calculators, keyboards, mobile phones, etc

iii) Industrial controllers, communication system, etc

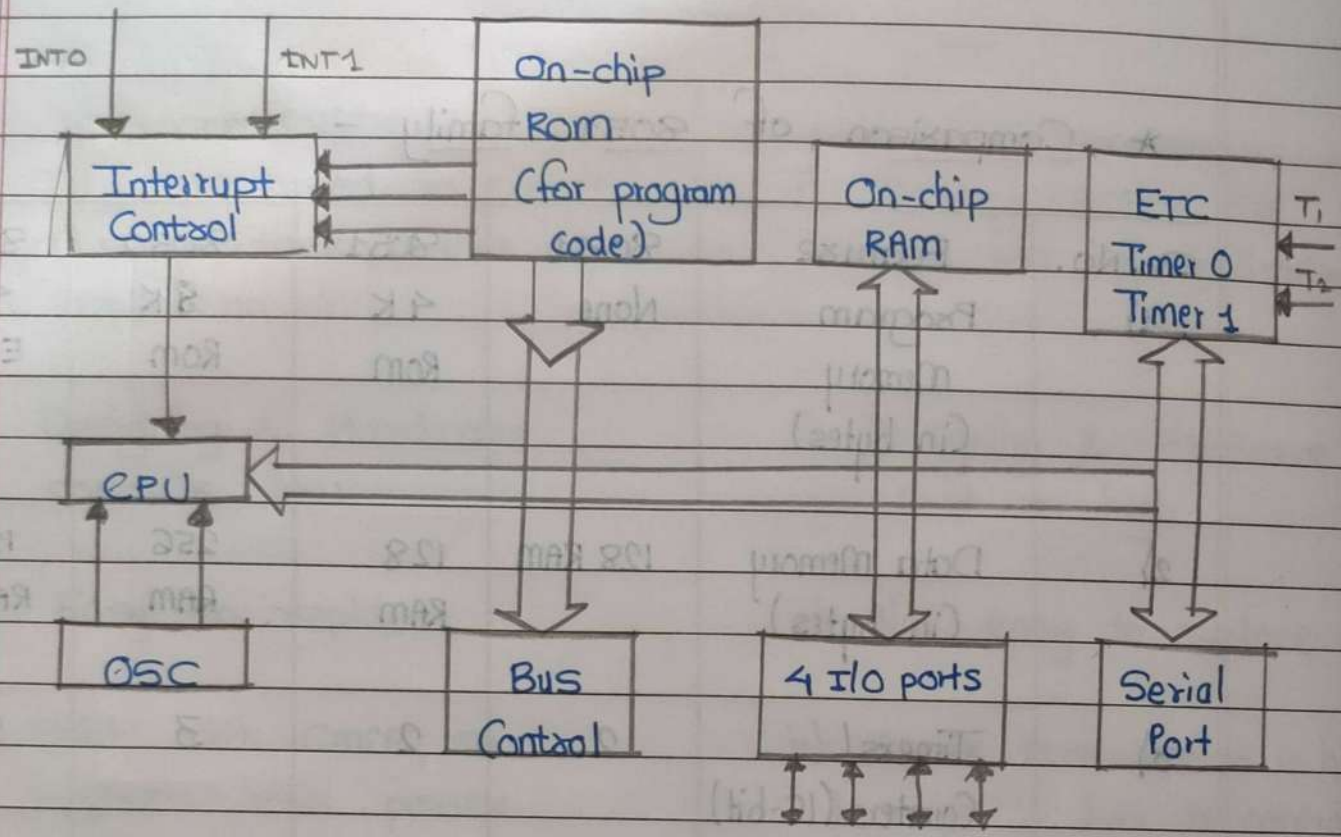
iv) Automobile engines, military applications.

★ Comparison of 8051 family -

Sr. No.	Feature	8031	8051	8052	8751
1)	Program Memory (in bytes)	None	4K ROM	8K ROM	4K EPROM
2)	Data Memory (in bytes)	128 RAM	128 RAM	256 RAM	128 RAM
3)	Timers/ Counters (16-bit)	2	2	3	2
4)	I/O pins	32	32	32	32
5)	Serial port	1	1	1	1
6)	Interrupt Sources	5	5	6	5

↑ You can write ★ features of 8051 from this.

★ Block - Diagram of 8051 Microcontroller -



★ Program Status Word (PSW) -

→ • PSW is also known as flag register

• Diagram -

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
CY	AC	F0	RS1	RS0	OV	-	P

• CY (Carry Flag) -
This flag is set, if there is overflow out of bit 7

- AC (Auxillary Carry Flag) -
  This flag is set if there is an overflow out of bit 3

- F0 -
  Available for user for general purpose

- RS1 - RS0 - They select working bank register

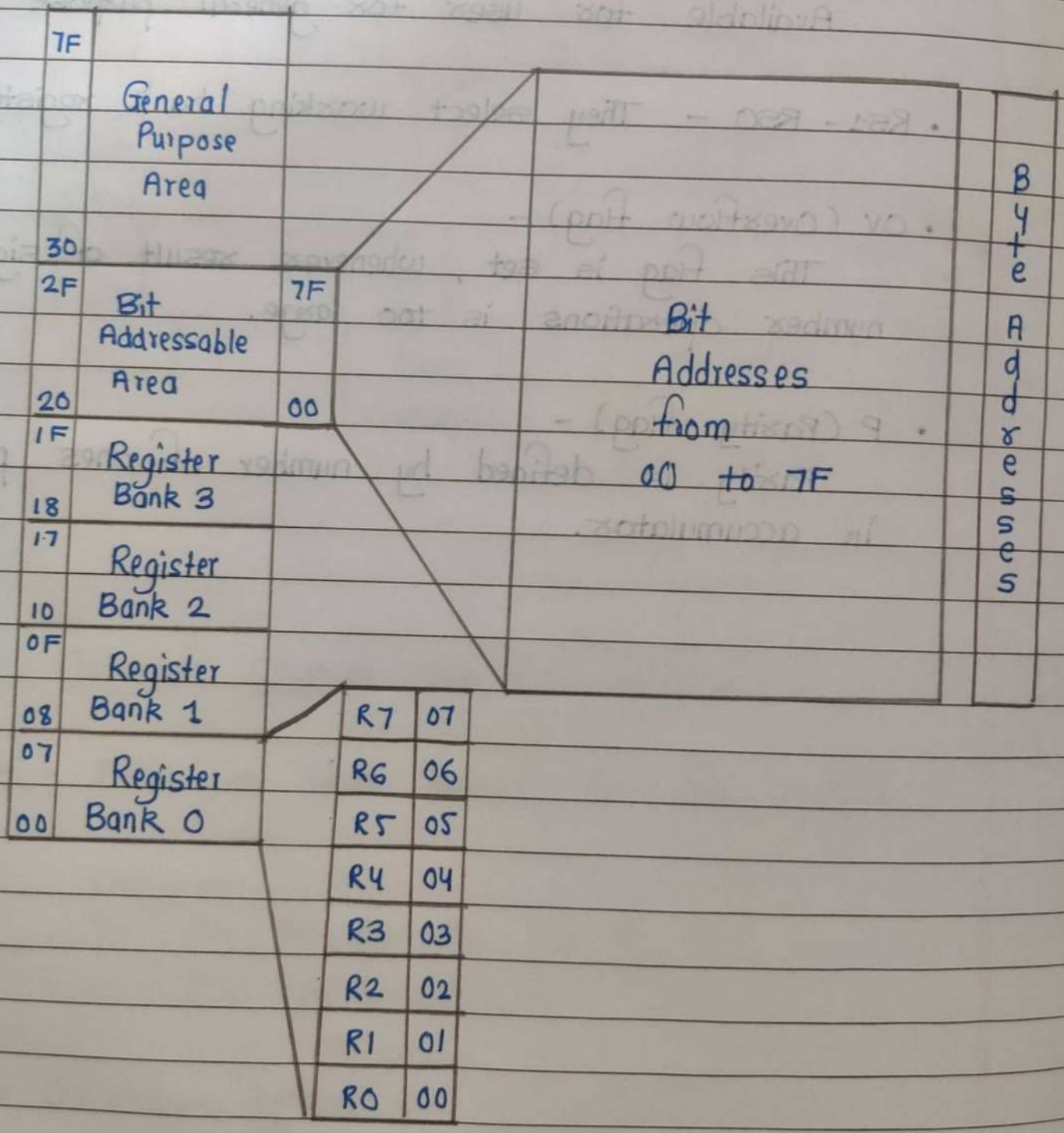
- OV (Overflow flag) -
  This flag is set, whenever result of signed number operations is too large.

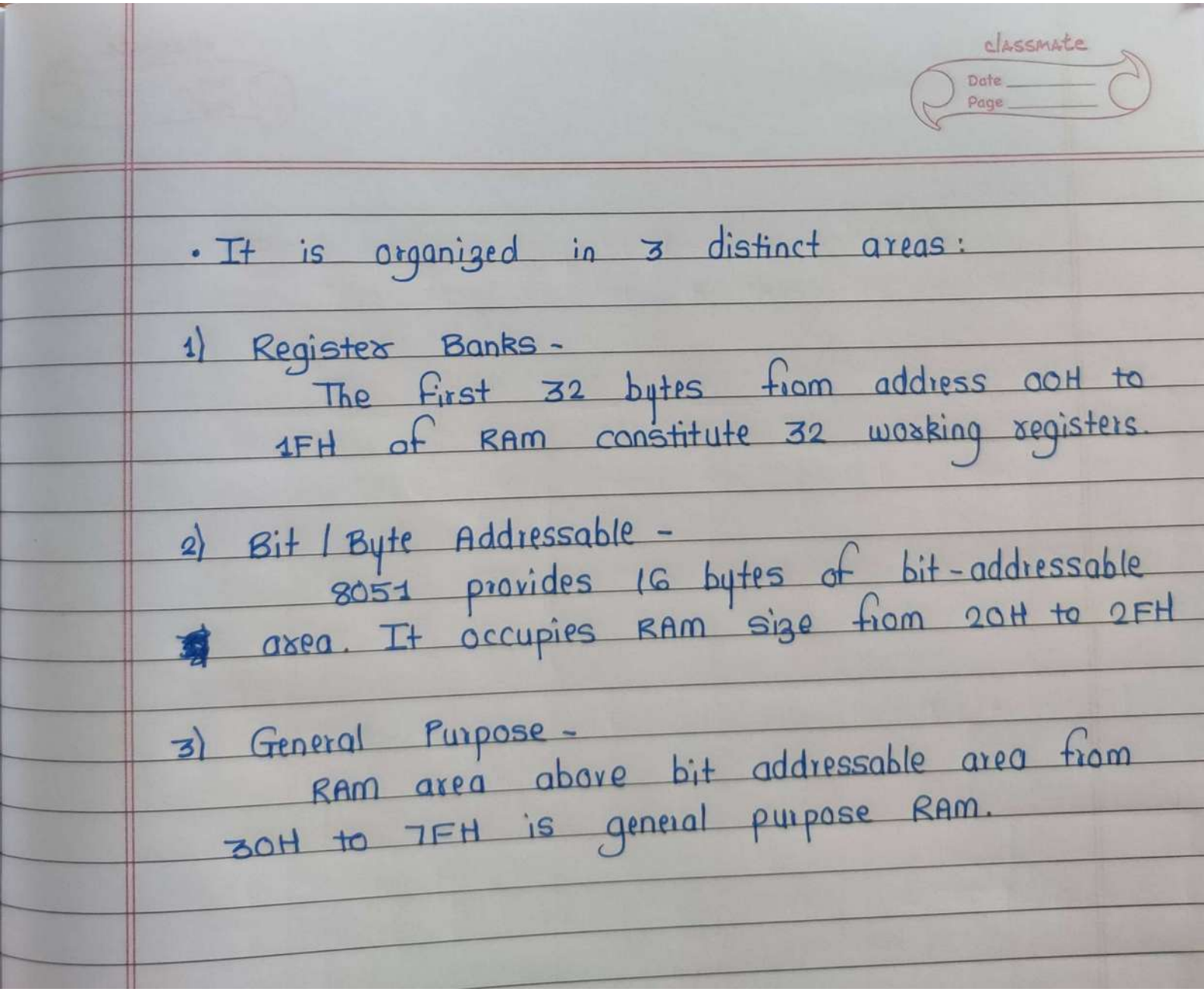
- P (Parity flag) -
  Parity is defined by number of ones present in accumulator.

★ Internal RAM Structure of 8051 -

→ • 8051 has 128-byte internal RAM

• Diagram -





- It is organized in 3 distinct areas:

1) Register Banks -
The first 32 bytes from address 00H to 1FH of RAM constitute 32 working registers.

2) Bit / Byte Addressable -
8051 provides 16 bytes of bit-addressable area. It occupies RAM size from 20H to 2FH

3) General Purpose -
RAM area above bit addressable area from 30H to 7FH is general purpose RAM.