System Architecture and Memory Management

Mr. Sumit Shinde
Assistant Professor
Computer Engineering Department

Pune Institute of Computer Technology

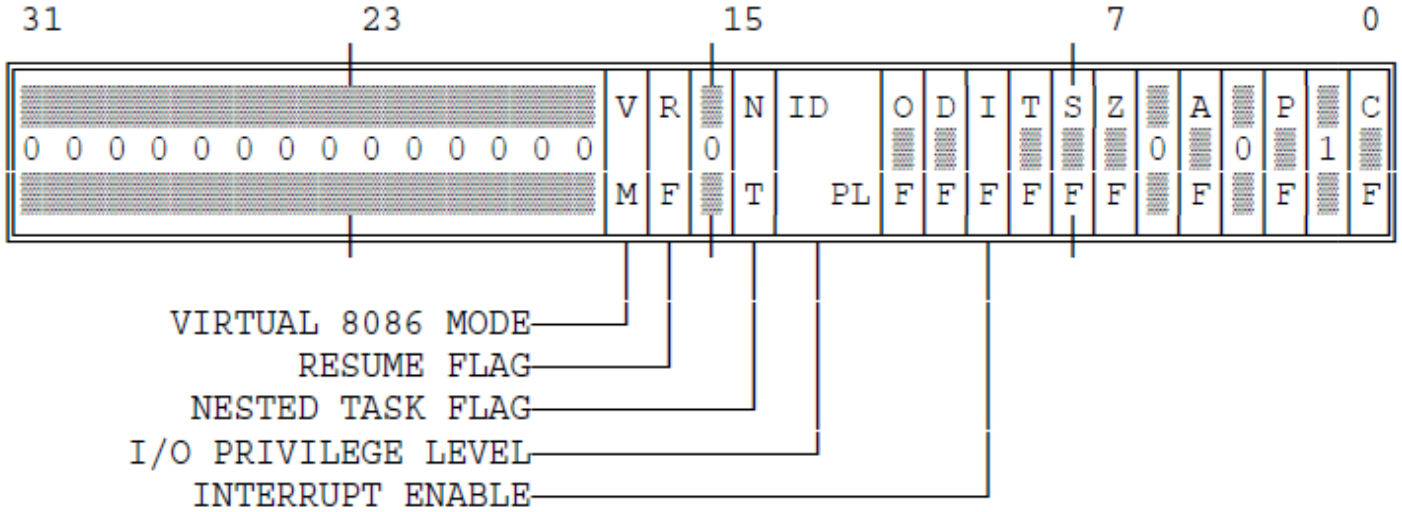
The systems-level features of the 80386 architecture include:

- Memory Management
- Protection
- Multitasking
- Input / Output
- Exceptions and Interrupts
- Initialization
- Coprocessing and Multiprocessing
- Debugging

System Registers

- EFLAGS
- Memory-management Registers
- CR_0 - CR_3 (Control Registers)
- TR (Task Register)
- DR_0 - DR_7 (Debug Registers)
- TR_6 - TR_7 (Test Registers)

EFLAGS- System Flags



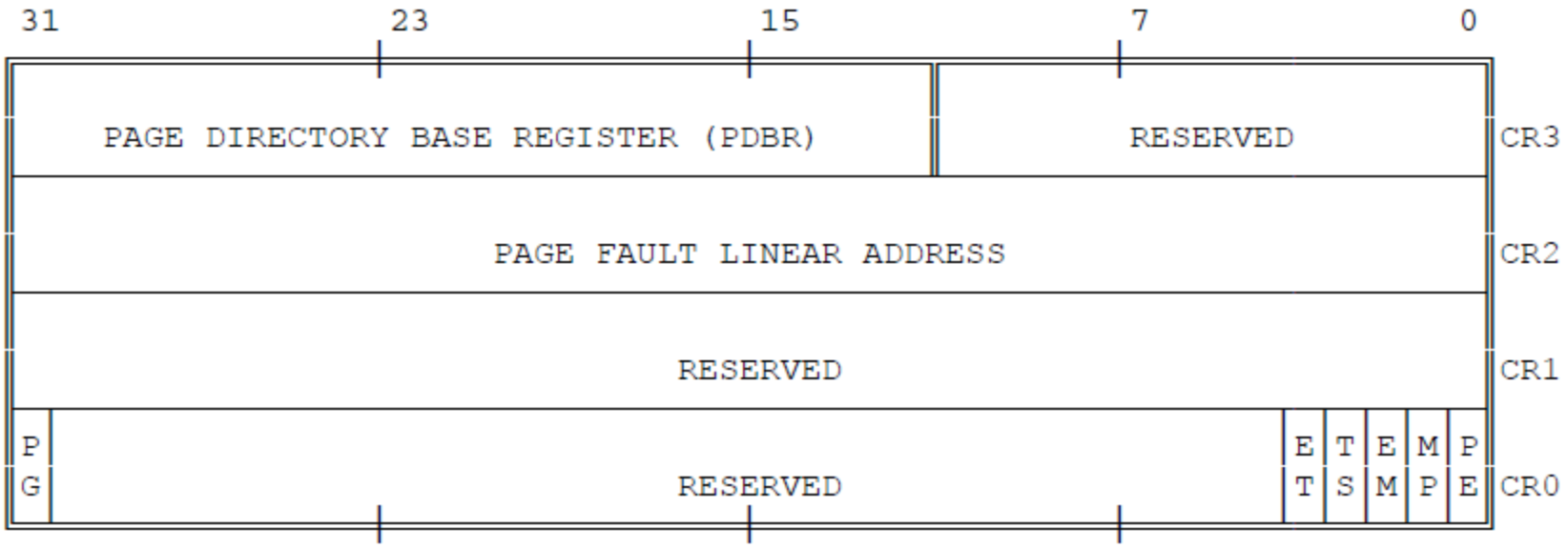
NOTE

0 OR 1 INDICATES INTEL RESERVED. DO NOT DEFINE.

Control Registers

- Intel386 DX has three control registers of 32 bits, CR0, CR2 and CR3, to hold machine state of a global nature
- These registers along with System Address Registers hold machine state that affects all tasks in the system.
- To access Control Registers, load and store instructions are defined.

Control registers



CR0 : Machine Control Register

- CR0 contains 6 defined bits for control and status purposes.
- The low-order 16 bits of CR0 is defined as Machine Status Word
- To operate only on the low-order 16-bits of CR0, LMSW and SMSW instructions are used.
- For 32-bit operations the system should use MOV CR0, Reg instruction.

CR0 : Machine Control Register

- **(PE Bit, Protection Enable) :**
  - The PE bit is set to enable the Protected Mode.
  - If PE is reset, the processor operates in Real Mode.
- **MP Bit, (Monitor Coprocessor / Math Present) :**
  - MP=1, assumes that real-floating point hardware is attached to it.
  - MP=0, assumes that no such coprocessor exists, and will not attempt to use one.

CR0 : Machine Control Register

- **(EM Bit, Emulate Coprocessor) :**

This bit is set to cause all coprocessor opcodes to generate a Coprocessor Not Available fault (exception 11).

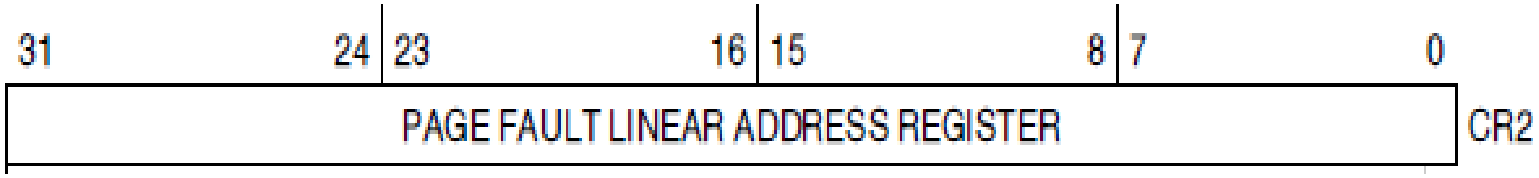
- **(TS Bit, Task Switched) :** TS is automatically set whenever a task switch operation is performed. Processor never clear this bit by its own. We can clear it using CLTS instruction.

- **ET (Extension Type):** This bit informs 80386 whether the numeric coprocessor is an 80387. 80386 is able to set this bit correctly by itself when power is applied.

- **PG (Paging Enable, bit 31):**the PG bit is set to enable the on-chip paging unit. It is reset to disable the on-chip paging unit. We can not change this bit more than once in running system.

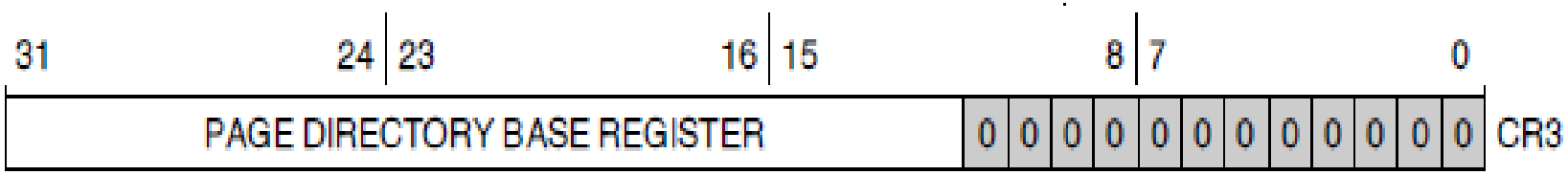
Control Registers

- **CR1 : Reserved** :CR1 is reserved for use in future Intel processors
- **CR2 : Page Fault Linear Address**:CR2 holds the 32-bit linear address that caused the last page fault detected. (Exception 14). This address will be helpful to write page fault handler.



Control Registers

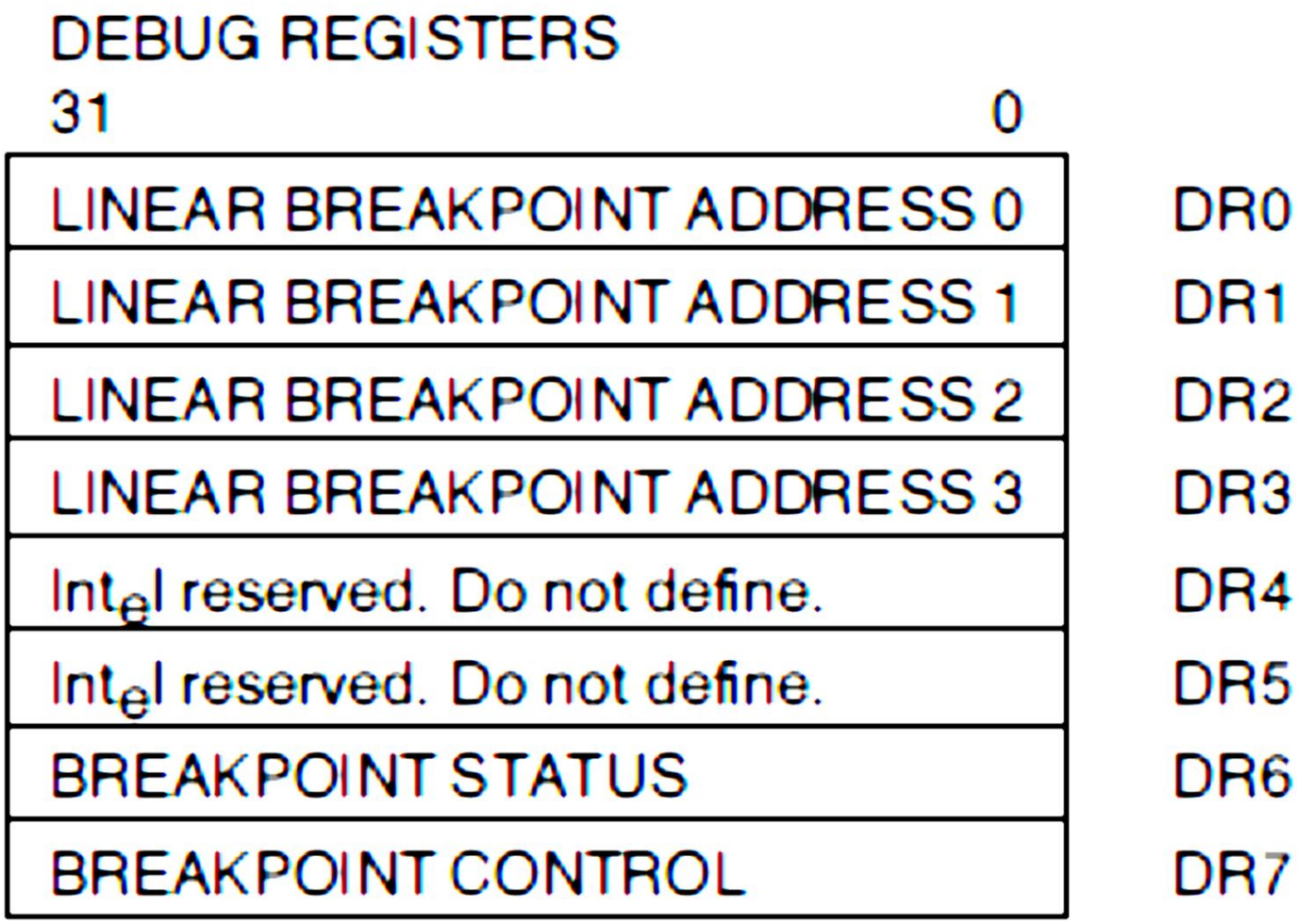
- **CR3 : Page Directory Base Address:**
- It holds the physical address of the root of the two-level paging tables used when paging is enabled.
- Paging table must be 4kb aligned.
- Lowest twelve bits of CR3 are ignored.



Debug Registers

- Debugging of 80386 allows **data access** breakpoints as well as **code execution** breakpoints.
- 80386 contains 6 debug registers to specify
  - 4 breakpoints
  - Breakpoint Control options
  - Breakpoint Status

Debug Registers



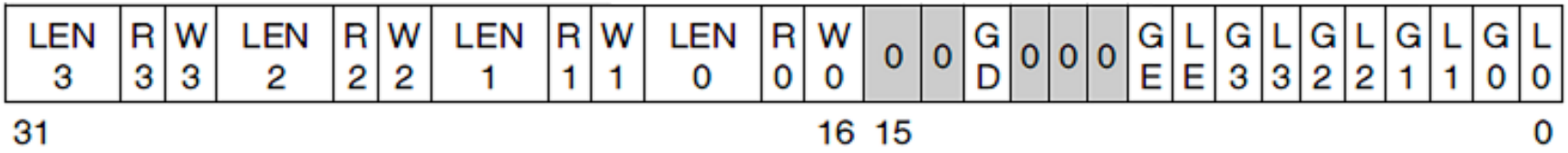
Linear Breakpoint Address Registers

- The breakpoint addresses specified are 32-bit linear addresses
- **DR0 to DR3:** While debugging, Intel 386 h/w continuously compares the linear breakpoint addresses in DR0-DR3 with the linear addresses generated by executing software. If match found , an exception 1 (debug fault) is generated.

Linear Breakpoint Address Registers

- **DR6:** It is also called as debug status register. 80386 sets appropriate bits in this register to inform you of the circumstances that may have caused the last debug fault (exception 1). These bits are never cleared by processor. We can clear it manually by writing into it.
- B0-B3: Indicates Breakpoint hit
- BD (break for debug register access): set when exception 1 handler is invoked by an illegal reference to one of the debug registers.
- BS (Break for single step)
- BT (break for task switch)

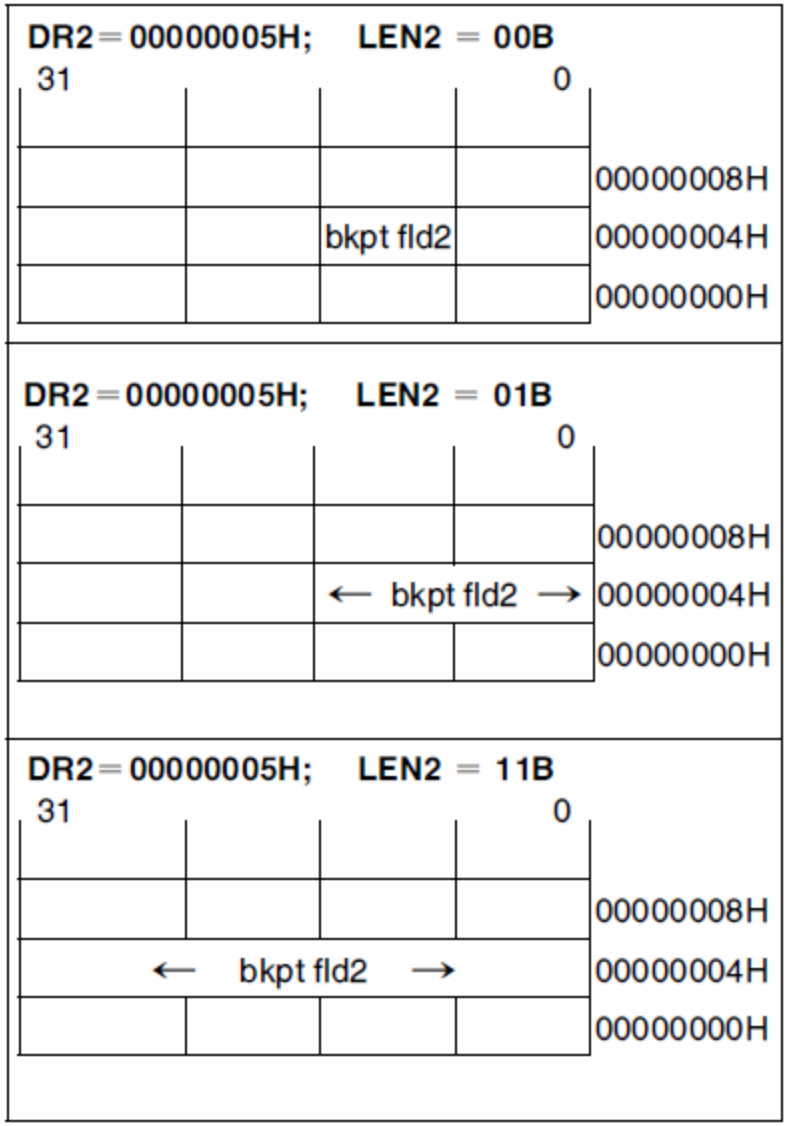
Debug Control Register



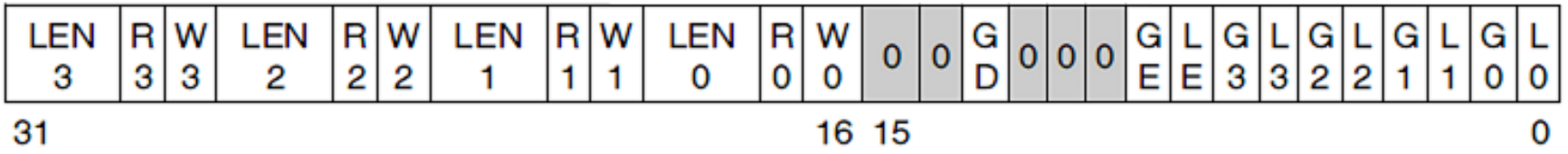
- **LEN_i**(i=0 - 3): Breakpoint Length Specification Bits:
  - 2 bit field for each breakpoint
  - Specifies length of breakpoint fields
  - The choices of data breakpoints are 1byte, 2bytes & 4bytes
  - For instruction execution breakpoint, the length is 1(beginning byte address)

LEN_i Encoding

LENi Encoding	Breakpoint Field Width	Usage of Least Significant Bits in Breakpoint Address Register i, (i = 0 – 3)
00	1 byte	All 32-bits used to specify a single-byte breakpoint field.
01	2 bytes	A1–A31 used to specify a two-byte, word-aligned breakpoint field. A0 in Breakpoint Address Register is not used.
10	Undefined—do not use this encoding	
11	4 bytes	A2–A31 used to specify a four-byte, dword-aligned breakpoint field. A0 and A1 in Breakpoint Address Register are not used.



Debug Control Register

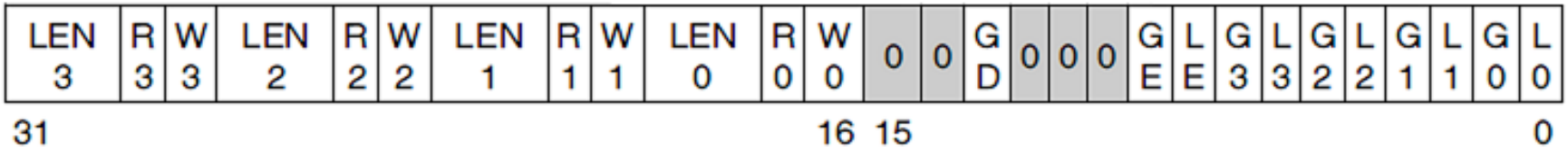


- **RW**$_i$(i=0 - 3): Memory Access Qualifier Bit
  - 2 bit field for each breakpoint
  - Specifies the type of usage which must occur inorder to activate the associated breakpoint

Debug Registers

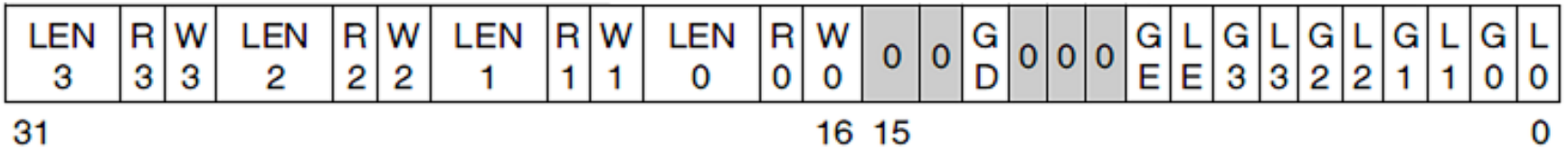
RW Encoding	Usage Causing Breakpoint
00	Instruction execution only
01	Data writes only
10	Undefined—do not use this encoding
11	Data reads and writes only

Debug Control Register



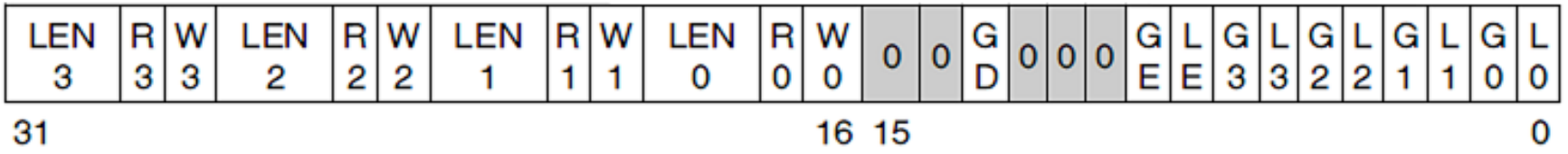
- **GD**: Global Debug Register Access Detect
  - Debug registers can only be accessed in real mode or at **privilege level 0** in protected mode
  - GD bit, when set, provides extra protection against any Debug Register access even in Real Mode or at privilege level 0 in Protected Mode.

Debug Control Register



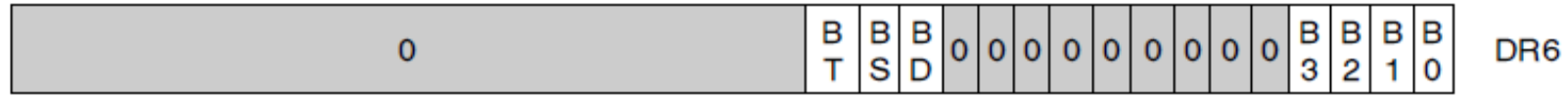
- **GE and LE bit**: Exact data breakpoint match, global and local
  - If either GE or LE is set, any data breakpoint trap will be reported exactly after completion of the instruction that caused the operand transfer.
  - LE bit is **cleared during task switch** and is used for task-local breakpoints.
  - GE bit is **unaffected during a task switch** and remain enabled during all tasks executing in the system.

Debug Control Register



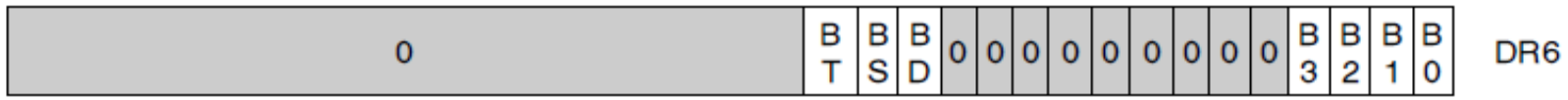
- **G_i** and **L_i**(i=0 - 3): Breakpoint Enable, global and local
  - If either G_i and L_i is set then the associated breakpoint is enabled.

Debug Status Register



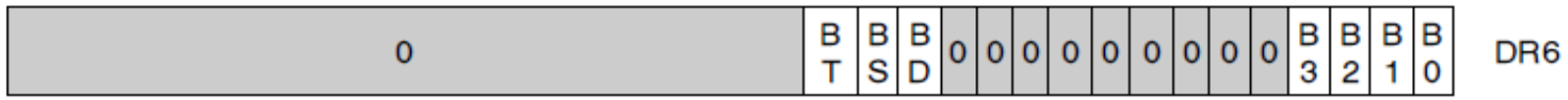
- **B_i** : Debug fault/trap due to breakpoint 0 -3
  - Four breakpoint indicator flags, B0-B3, correspond one-to-one with the breakpoint registers in DR0-DR3.
  - A flag B_i is set when the condition described by DR_i, LEN_i, and RW_i occurs.

Debug Status Register



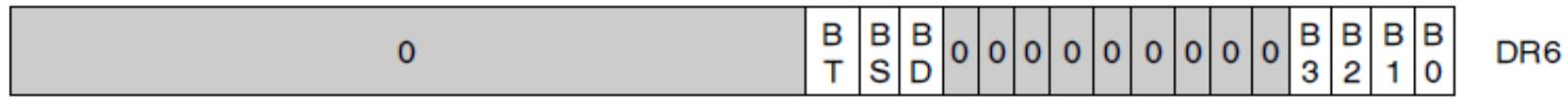
- **BD** : Debug fault due to attempted register access when GD bit is set
  - This bit is set if the exception 1 handler was invoked due to an instruction attempting to read or write to the debug registers when GD bit was set.

Debug Status Register



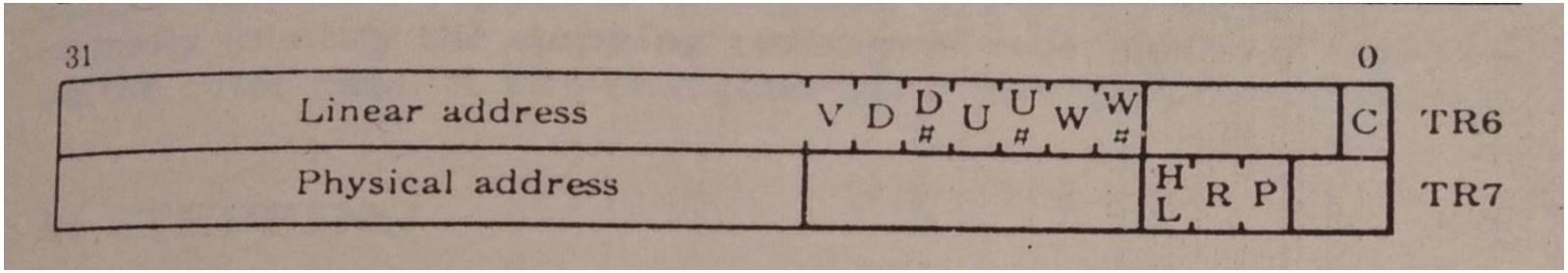
- **BS** : Debug trap due to single step
  - This bit is set if the exception 1 handler was invoked due to the TF bit in the flag register being set

Debug Status Register



- **BT** : Debug trap due to task switch
  - This bit is set if the exception 1 handler was invoked due to a task switch.

Test Register



- The test registers are used to perform confidence checking on the paging MMU's translation lookaside buffer(TLB).
- The method of testing the TLB is fairly involved and requires some understanding of the inner working of the TLB and of cache algorithms in general
- **Test Register 6:** Register TR6 is the testing command register . By writing into this register , you can either initiate write directly into the 80386's TLB or perform a mock TLB lookup.
- **Test Register 7 :** TR7 is the TLB testing data register . When a program is performing writes , the entry to be stored is contained in this register , along with cache set information .

Memory Management Registers

- **GDTR and IDTR**
- These registers hold:
  - 32-bit linear base address and
  - 16-bit limit of GDT and IDT respectively.
- GDT segments are global to all tasks in the system.
- IDT is used to locate Gates in Interrupt/Exception handling.

LDTR

- LDTR (Local Descriptor Table Register) is a 16-bit register always points to GDT to access LDTD (Local Descriptor Table Descriptor) in turn responsible for allocating and accessing local memory via LDT(Local Descriptor Table).

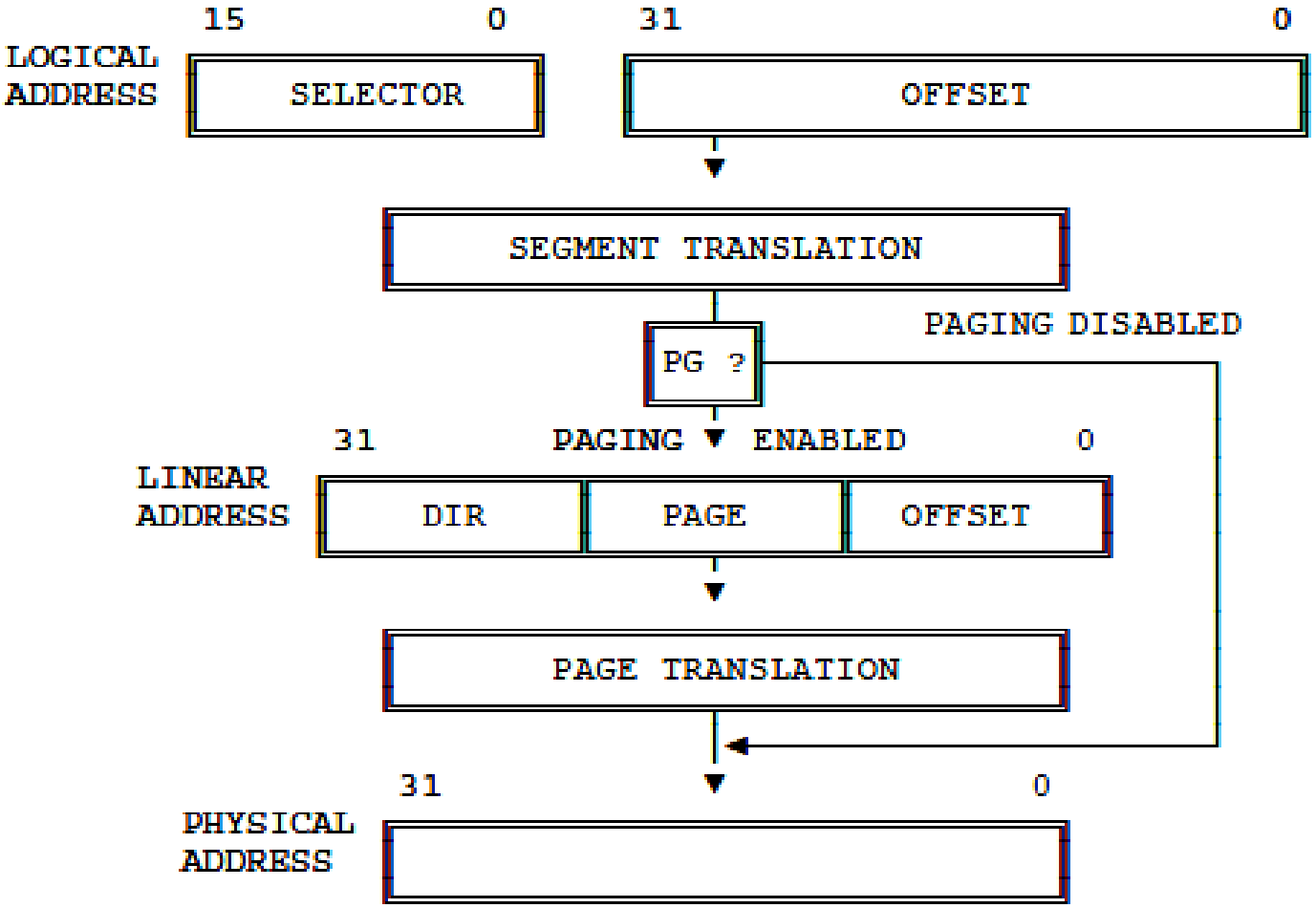
Memory Management

Address Translation Mechanism of 80386

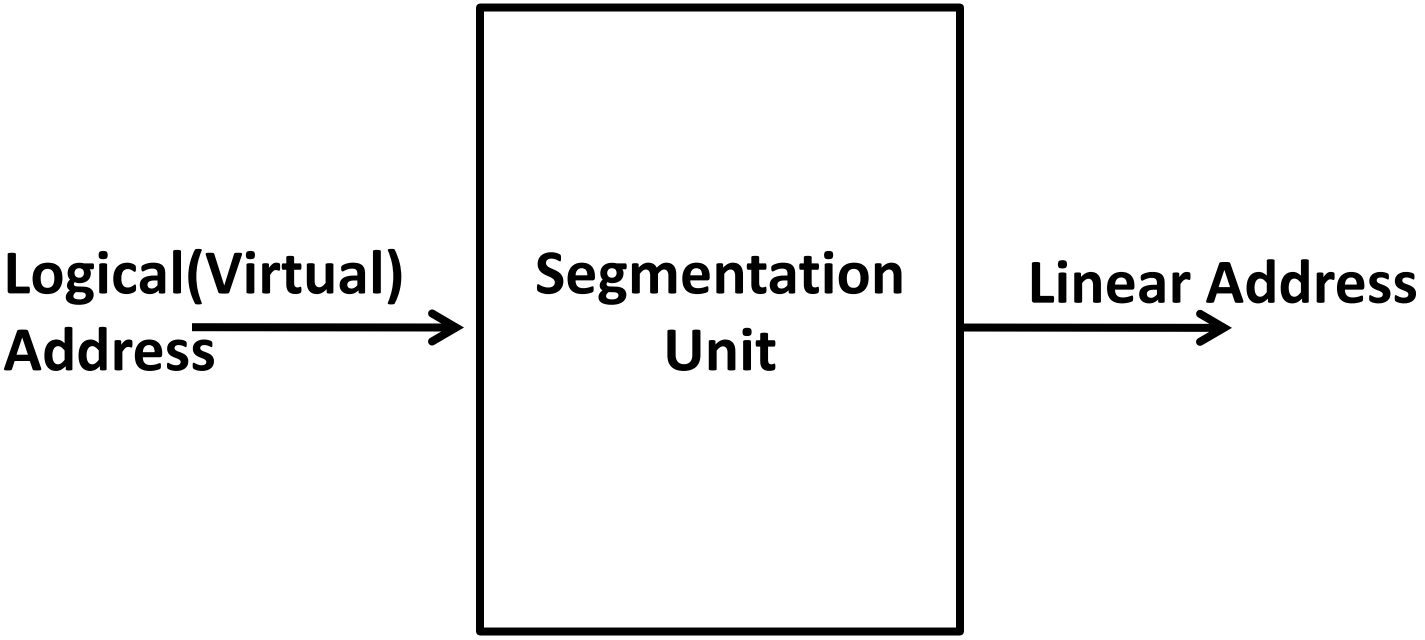
Protected Mode Addressing Mechanism

- 80386 transforms logical addresses into physical address two steps:
- **Segment translation:** a logical address is converted to a linear address.
- **Page translation:** a linear address is converted to a physical address.(optional)
- These translations are performed in a way that is not visible to applications programmers.

- The following figure illustrates the two translations:



Segmentation



Segment Descriptor

- Segment is described by a special structure called a segment descriptor.
- Descriptor includes its base address, its length, its type, its privilege level and some status information.
- If we don't describe an area of the address space in a descriptor, that address range is not addressable at all and processor will refuse to access it.
- Descriptors are created by compilers, linkers, loaders or the operating system.

Segment Descriptor

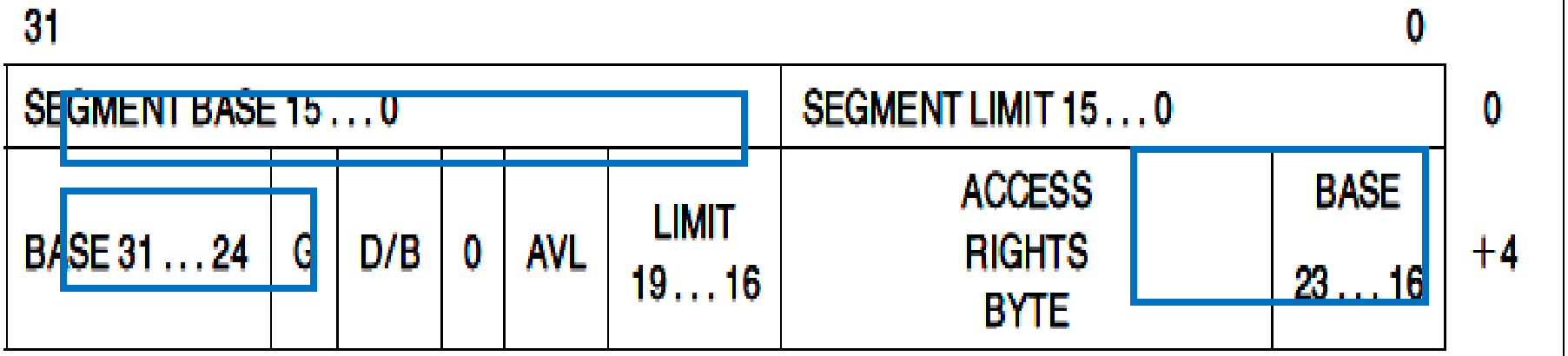


Figure 4-6. Segment Descriptors

Segment Descriptor

- **BASE:** defines location of the segment within 4GB linear address space. Combining three parts of the base address to form single 32 bit value.
- **LIMIT:** It define size of segment. It is 20 bit field.
- **G:** (Granularity Bit): Specifies unit with which the limit field is interpreted.

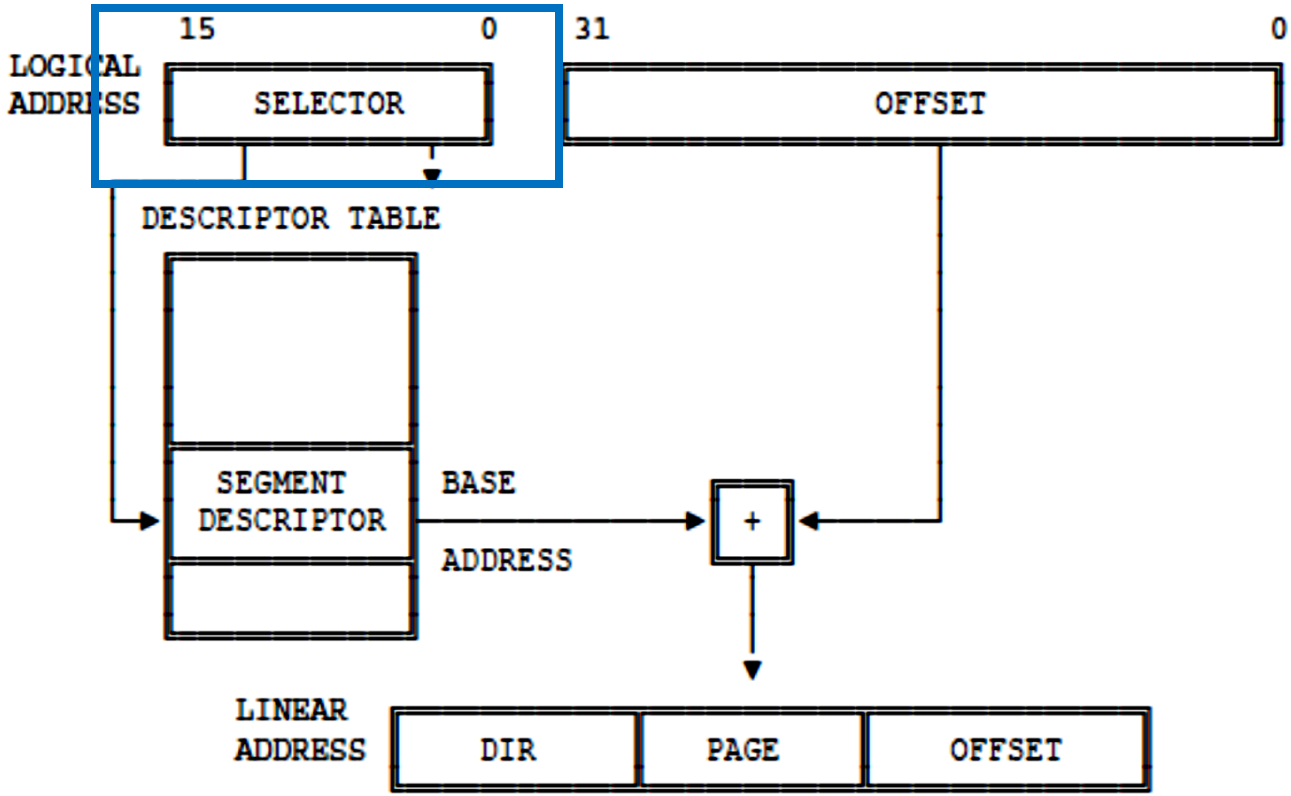
  G=0 unit of 1 byte

  G=1 unit of 4KB
- **TYPE:** 3 bit field defines type of segment you are defining.
- **DPL:** (Descriptor privilege level) Used by protection mechanism. 2 bit field defines level of privilege associated with the memory space that descriptor defines.
- **D: Default :** D=0 operands in segment will be considered 16bit, D=1 considered 32 bit

- P: (Segment Present):

if P=0 address range defines by descriptor is considered to be temporarily not present in physical memory.

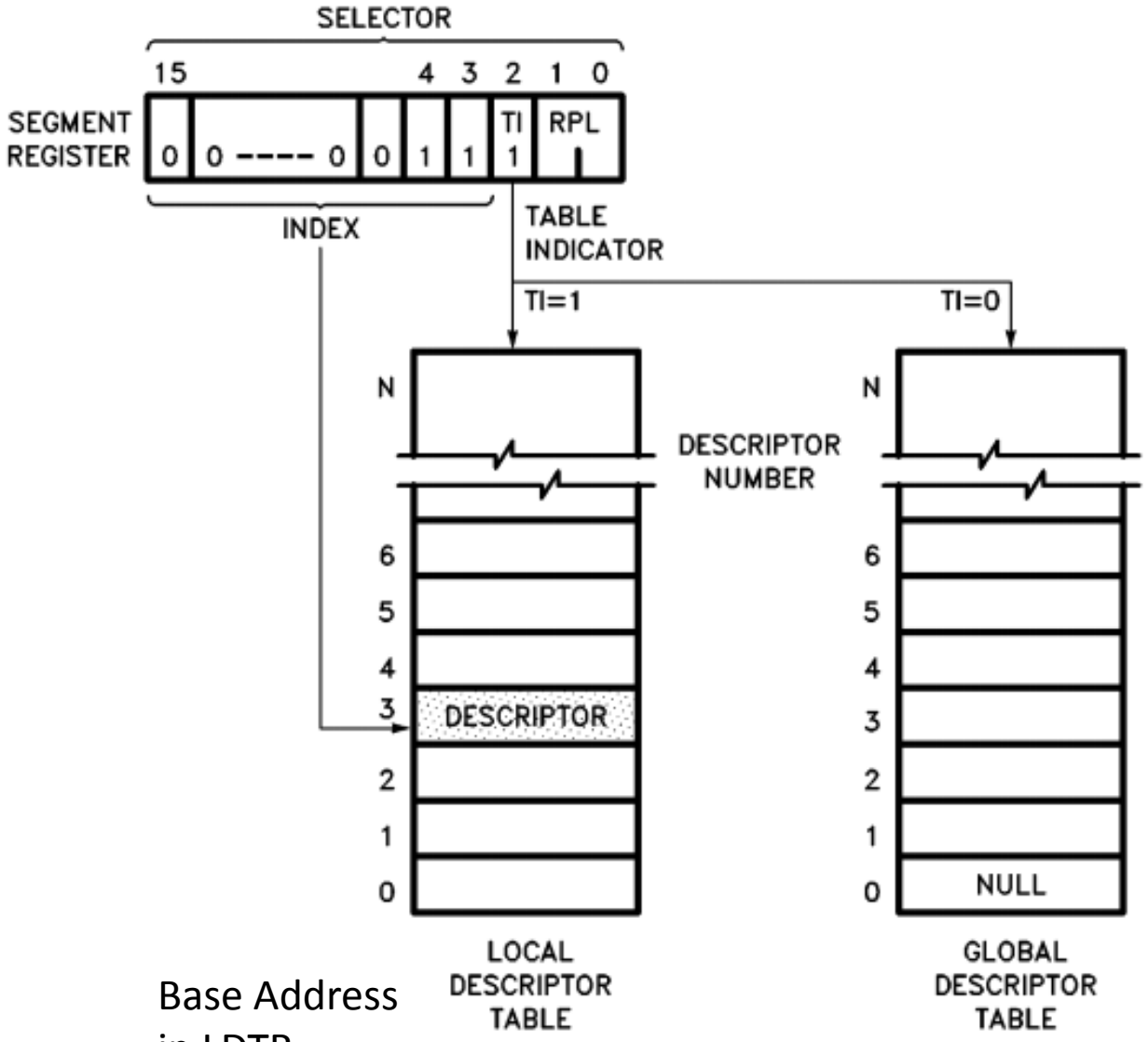
Segment Translation



Descriptor Tables

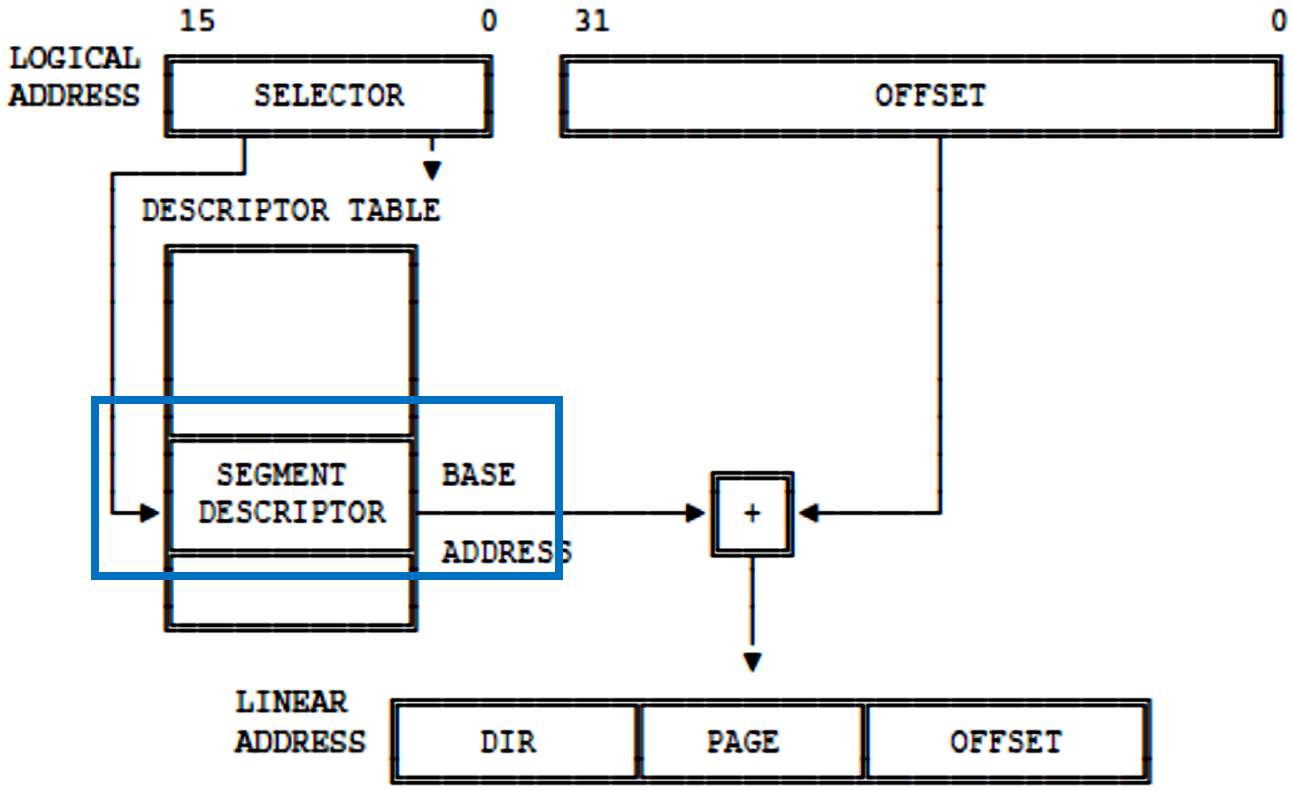
- GDT (Global Descriptor Table)
- LDT (Local Descriptor Table)
- IDT (Interrupt Descriptor Table)

- Descriptor table is array of 8 bytes entries that contains descriptors.
- Descriptor table is variable in length and may contain up to 8192 (2^13) descriptors.
- Processor locates GDT and LDT by GDTR and LDTR registers respectively.



Base Address in LDTR Register

Base Address in GDTR Register


15
0
31
0
LOGICAL
ADDRESS
SELECTOR
OFFSET
DESCRIPTOR TABLE
SEGMENT
DESCRIPTOR
BASE
ADDRESS
+
LINEAR
ADDRESS
DIR
PAGE
OFFSET

GDT:

- Can be used by all programs to reference segments of memory.

LDT:

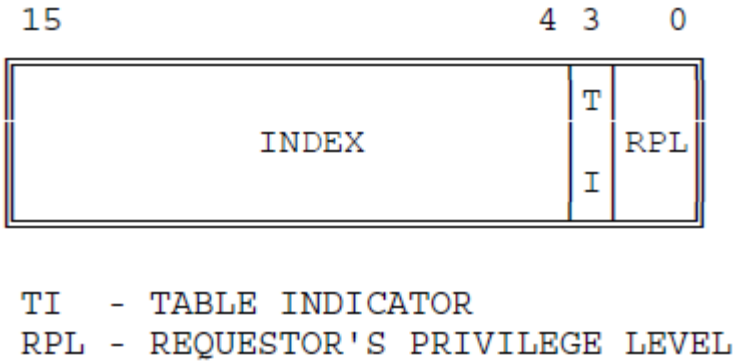
- Can be defined on a per-task basis in a multitasking system. But its use is purely optional.
- LDT is used to expand number of available descriptors and hence addressable range of selected task.

IDT:

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

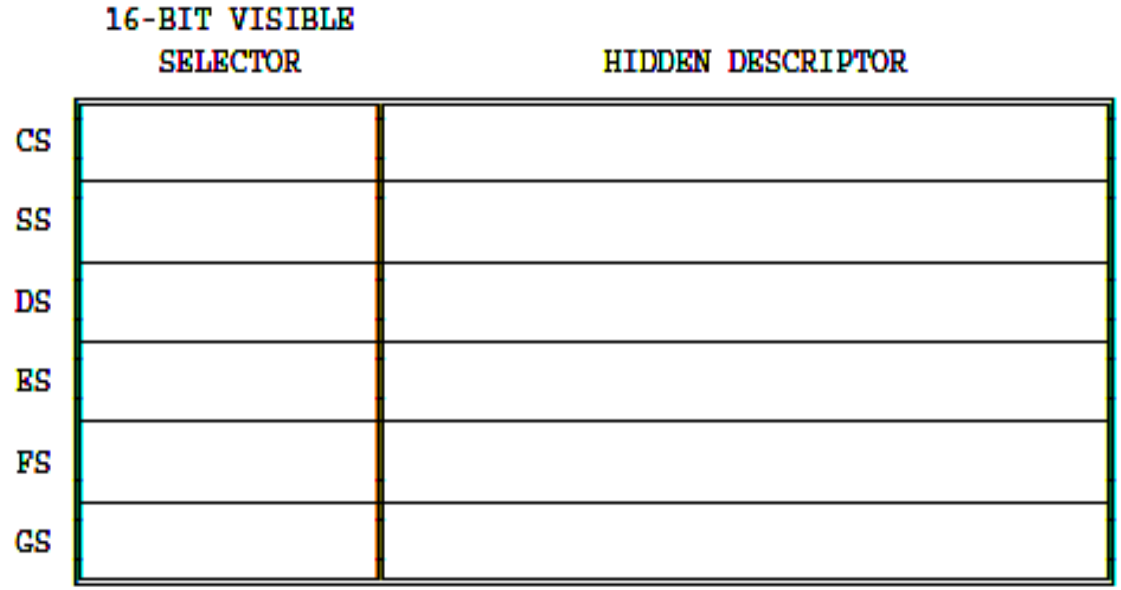
Segment Selectors

- **Index:** Selects one of 8192 descriptors in a descriptor table. The processor simply multiplies this index value by 8 (the length of a descriptor), and adds the result to the base address of the descriptor table in order to access the appropriate segment descriptor in the table.
- **Table Indicator:** Specifies to which descriptor table the selector refers. A zero indicates the GDT; a one indicates the current LDT.
- **Requested Privilege Level:** Used by the protection mechanism.

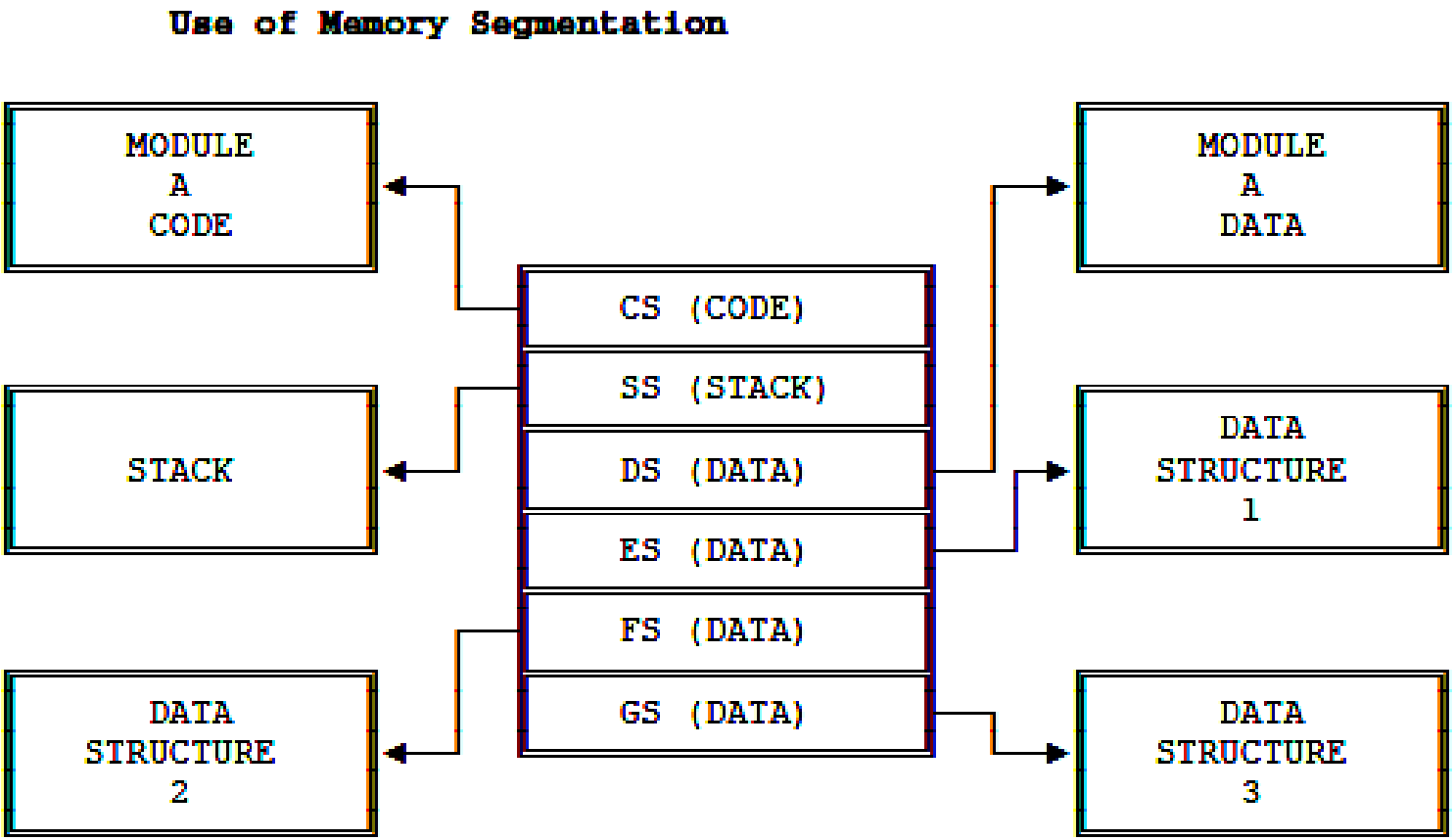


Segment Registers

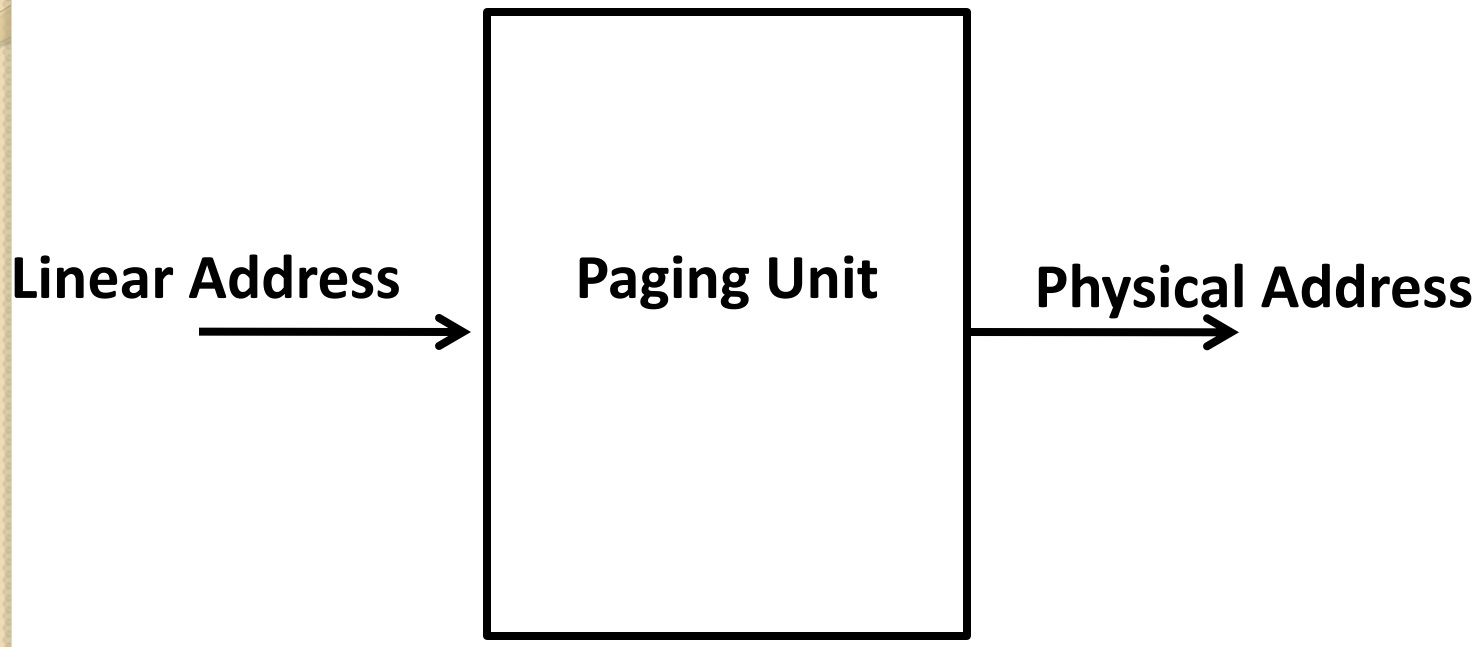
Figure 5-7. Segment Registers

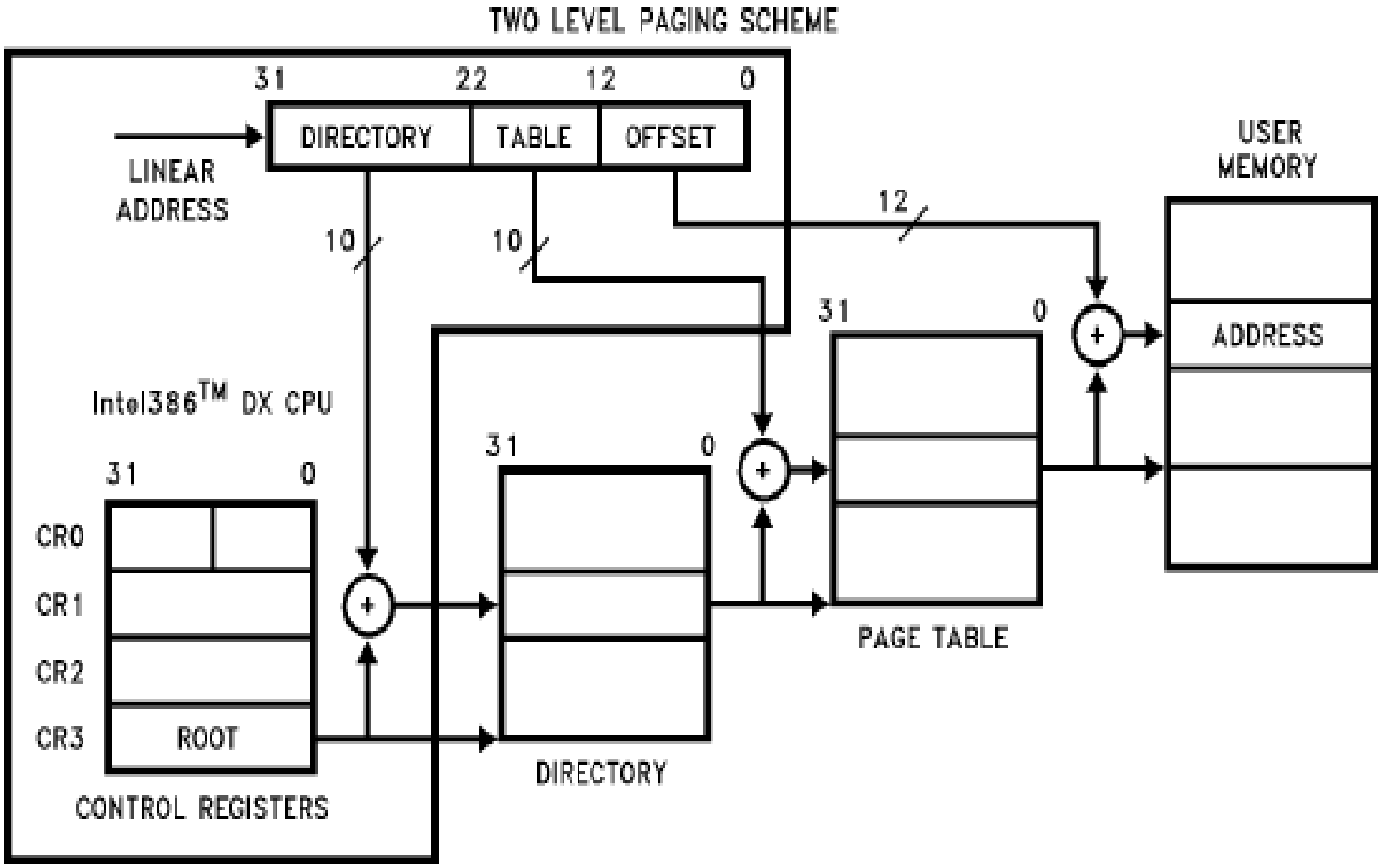


For each Task



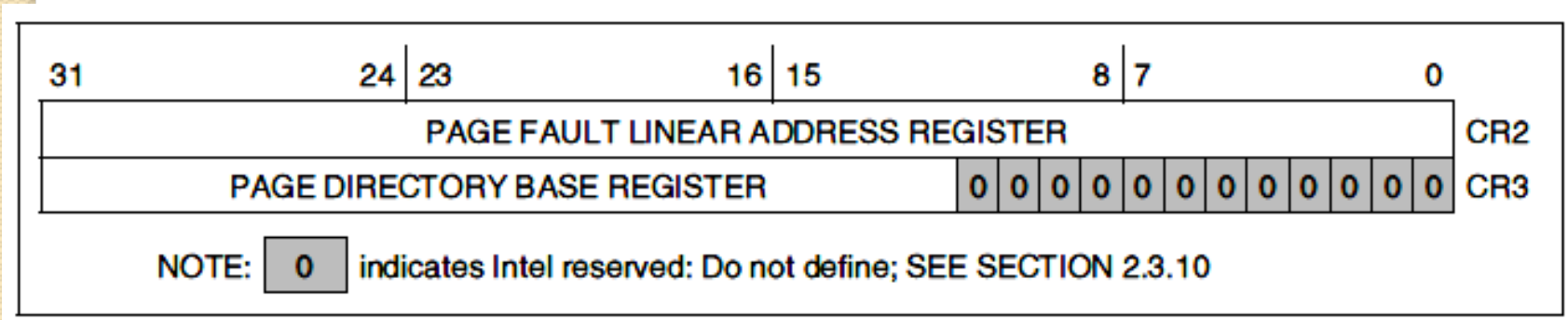
Page Translation




TWO LEVEL PAGING SCHEME
31
22
12
0
DIRECTORY
TABLE
OFFSET
LINEAR ADDRESS
10
10
12
USER MEMORY
ADDRESS
31
0
Intel386™ DX CPU
31
0
31
0
CR0
CR1
CR2
CR3
ROOT
CONTROL REGISTERS
DIRECTORY
PAGE TABLE

Page Descriptor Base Register

- CR2 is used to store the 32-bit linear address of page fault.
- CR3 (Page Directory Physical Base Address Register) stores the **physical starting address** of Page Directory.



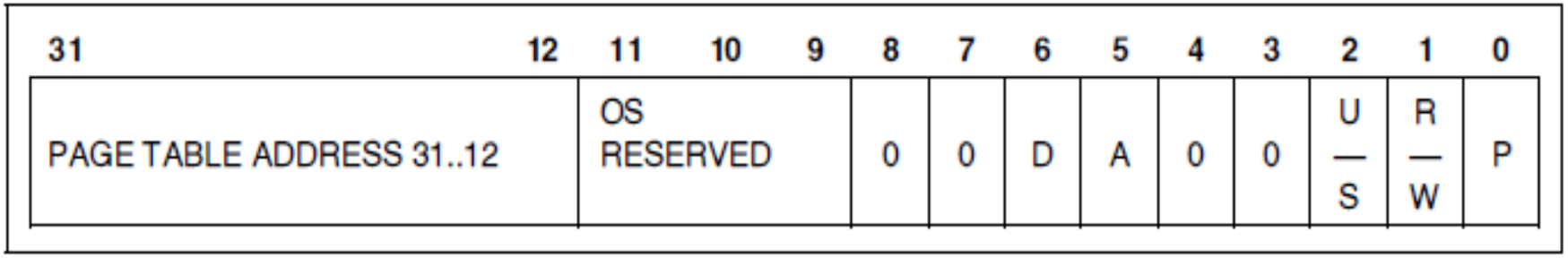
Page Descriptor Base Register

- The lower 12 bits of CR3 are always zero to ensure that the **Page Directory is always page aligned**
- A move operation to CR3 automatically loads the Page Table Entry caches and a **task switch** through a TSS changes the value of CR0.

Page Directory

- It is at the most **4KB in size** and allows upto **1024** entries are allowed.
- The **upper 10 bits** of the linear address are used as an **index** to corresponding page directory entry
- Page directory entry **points** to page tables.

Page Directory Entry

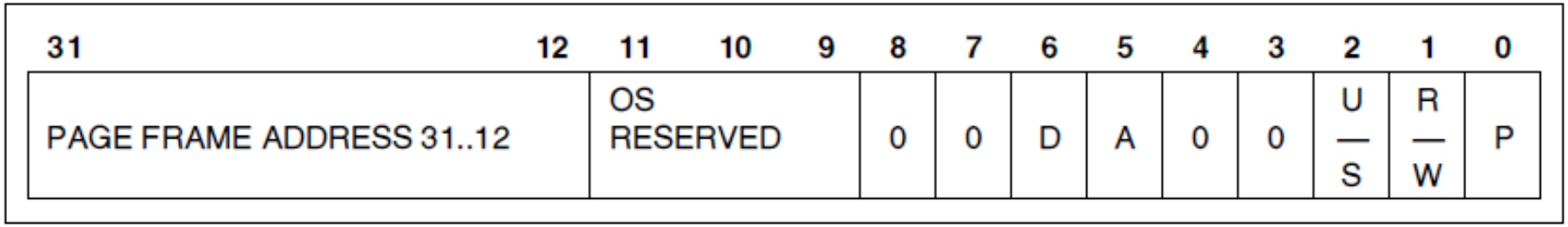


31 — 12	11 — 9	8	7	6	5	4	3	2	1	0
PAGE TABLE ADDRESS 31..12	OS RESERVED	0	0	D	A	0	0	U — S	R — W	P

Page Tables

- Each Page Table is 4KB and holds up to 1024 Page Table Entries(PTE).
- PTEs contain the starting address of the page frame and statistical information about the page.
- Upper 20 bit page frame address is concatenated with the lower 12 bits of the linear address to form the physical address.
- Page tables can be shared between tasks and swapped to disks.

Page Table Entry



31 — 12	11 — 9	8	7	6	5	4	3	2	1	0
PAGE FRAME ADDRESS 31..12	OS RESERVED	0	0	D	A	0	0	U — S	R — W	P

Page Table Entry (Points to Page)

- **P(Present)Bit:** indicates if the entry can be used in address translation. P-bit of the currently executed page is always high.
- **A (Accessed) Bit:** It is set before any access to the page.

Page Table Entry

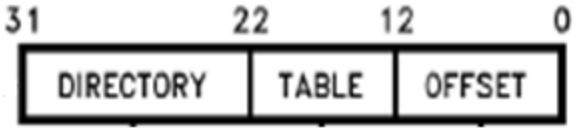
- **D (Dirty) bit:** It is set before a write operation to the page is carried out. The D bit is undefined for PDEs.
- **OS Reserved Bits:** They are defined by the operating system software.
- **U/S (User/Supervisor)Bit and R/W (Read/Write) Bit:** They are used to provide protection. They

U/S	R/W	Permitted Level 3	Permitted Access Levels 0, 1, or 2
0	0	None	Read/Write
0	1	None	Read/Write
1	0	Read-Only	Read/Write
1	1	Read/Write	Read/Write

Example

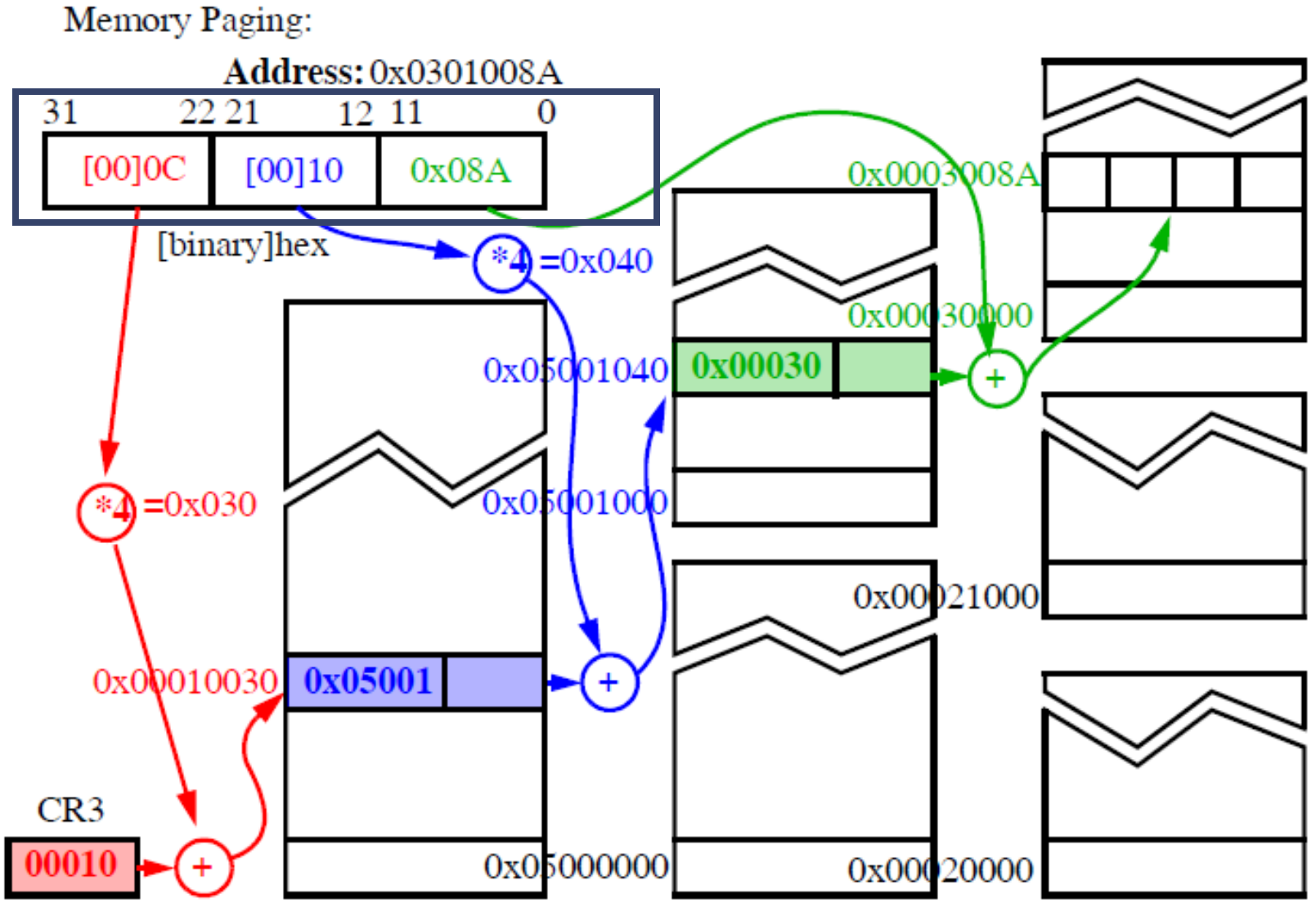
Linear Address : 0301008A

0000 0011 0000 0001 0000 0000 1000 1010



Binary	00 0000 1100 (10bits)	00 0001 0000 (10bits)	0000 1000 1010 (12bits)
Hex	00C	010	08A

Example

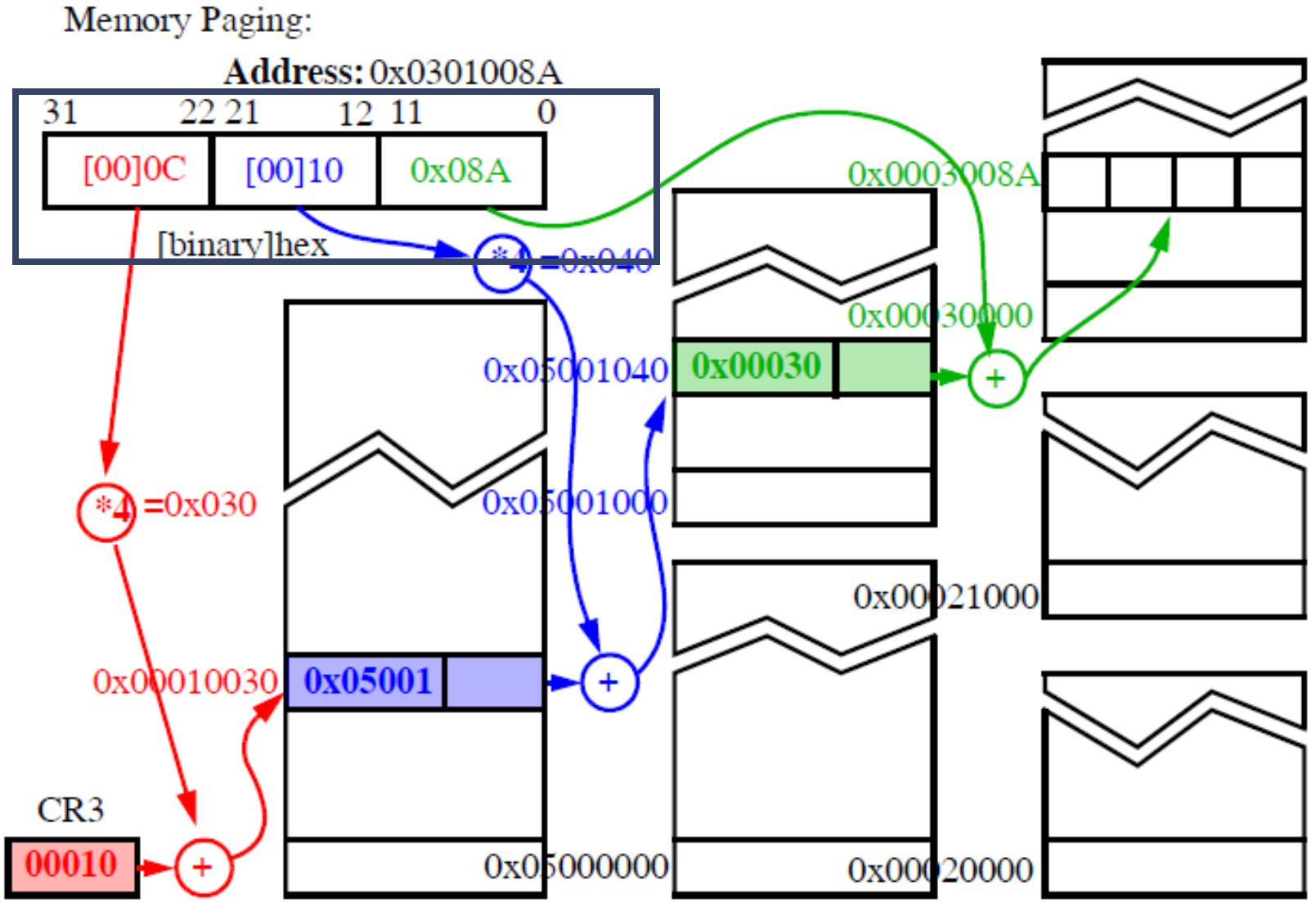

Memory Paging:
Address: 0x0301008A
31
22 21
12 11
0
[00]0C
[00]10
0x08A
[binary]hex
*4 =0x040
*4 =0x030
0x0003008A
0x00030000
0x05001040
0x05001000
0x00021000
0x00010030
0x05001
CR3
00010
0x05000000
0x00020000

Hex	00C(DIR)	x4	030
Binary	00 0000 1100	00 0000 1100 x 0100 ——————— 00 0011 0000	

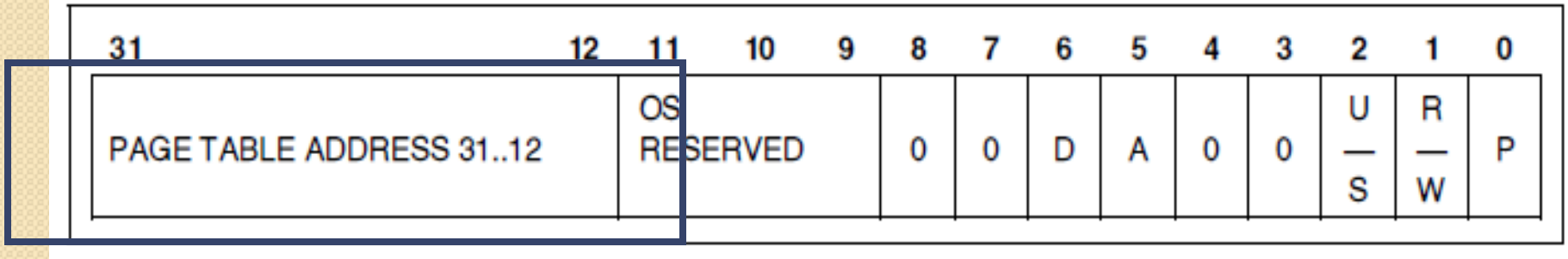
CR3	+	DIR*4	=	Index to PDE
(20-bit)		(12-bit)		

00010H	+	030H	=	00010030H

Example


Memory Paging:
Address: 0x0301008A
31
22 21
12 11
0
[00]0C
[00]10
0x08A
[binary]hex
*4 =0x040
*4 =0x030
0x05001040
0x05001000
0x00010030
0x05001
CR3
00010
0x05000000
0x0003008A
0x00030000
0x00030
0x00021000
0x00020000

Page Directory Entry

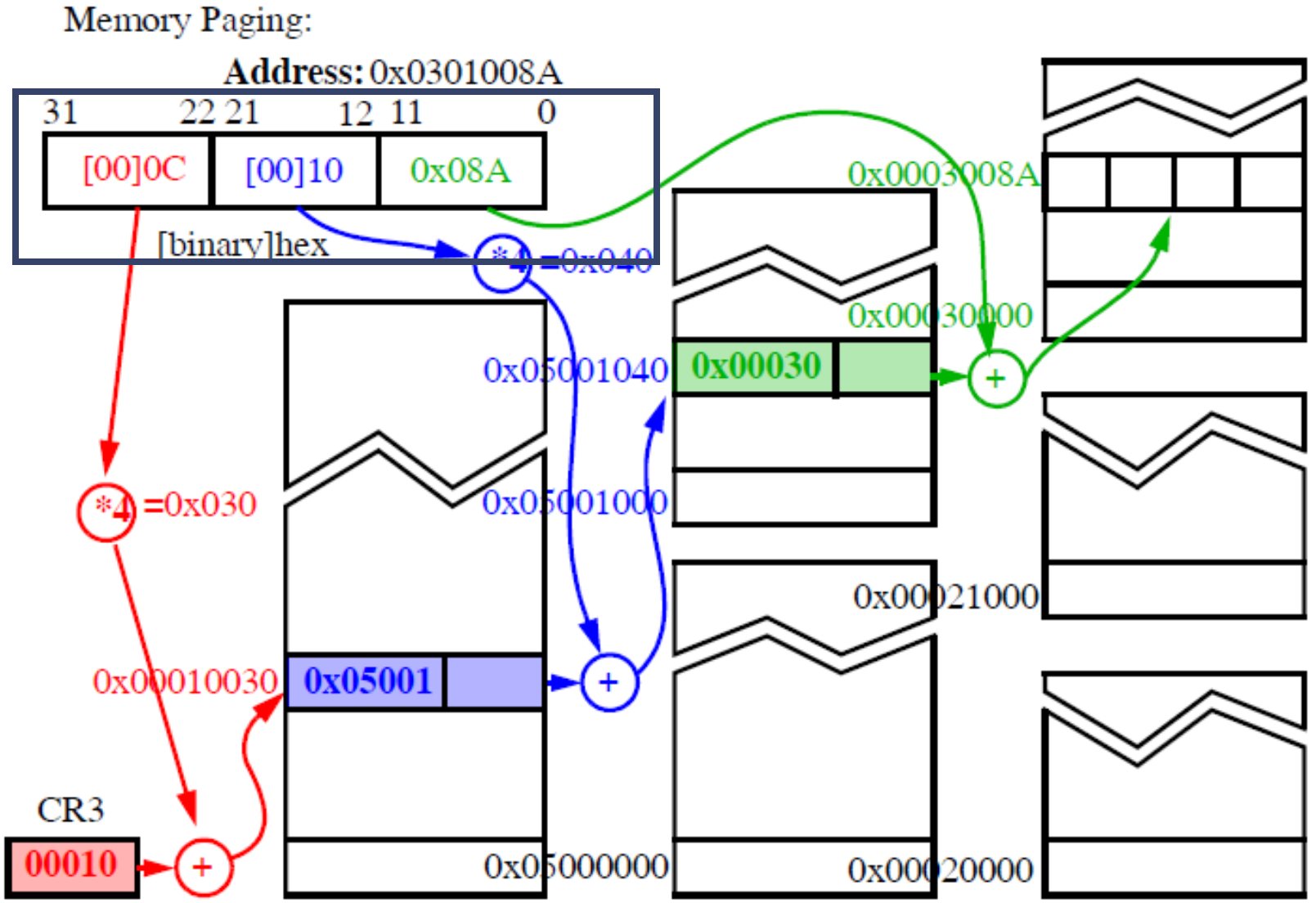


Hex	010(TABLE)	x4	040
Binary	00 0001 0000	00 0001 0000 x 0100 --- 00 0100 0000	

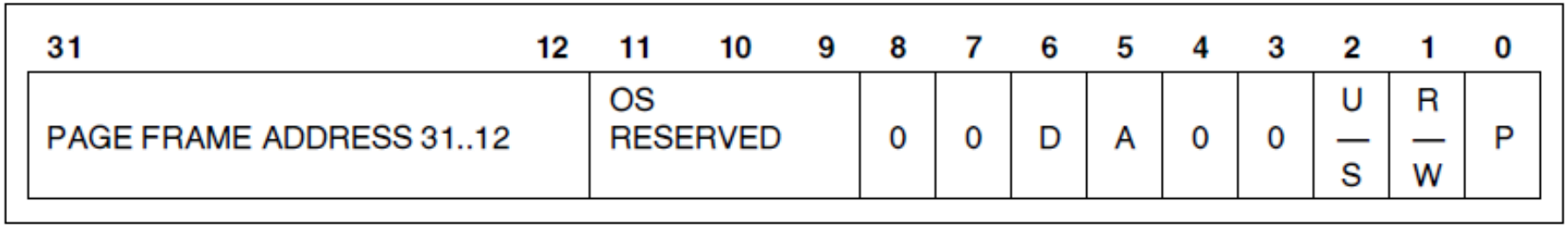
PTA	+	Table*4	=	Index to PTE
(20-bit)		(12-bit)		

05001H	+	040H	=	05001040H

Example


Memory Paging:
Address: 0x0301008A
31
22 21
12 11
0
[00]0C
[00]10
0x08A
[binary]hex
*4 =0x040
*4 =0x030
0x05001040
0x05001000
0x00010030
0x05001
CR3
00010
0x05000000
0x0003008A
0x00030000
0x00030
0x00021000
0x00020000

Page Table Entry



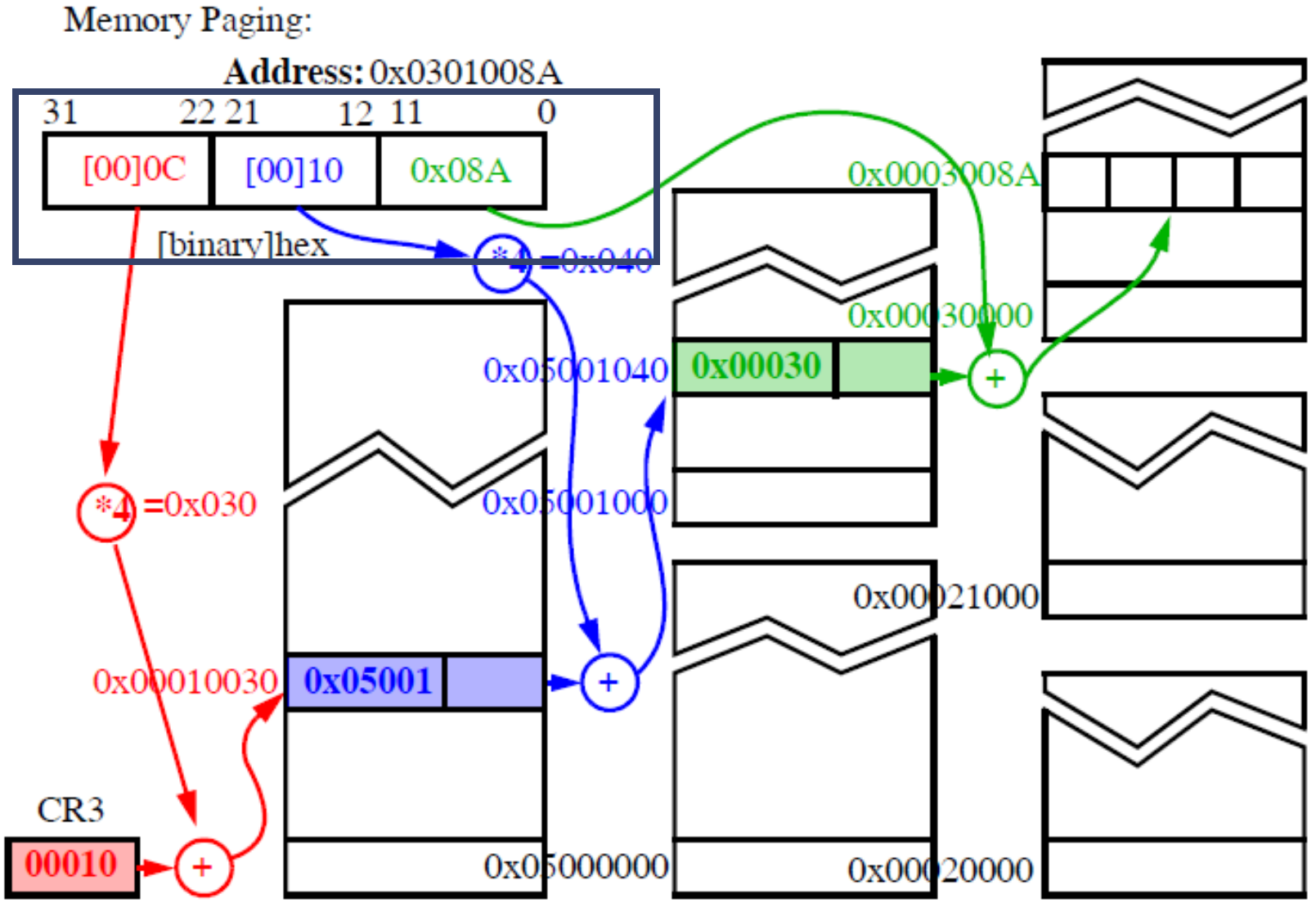
31 — 12	11 — 9	8	7	6	5	4	3	2	1	0
PAGE FRAME ADDRESS 31..12	OS RESERVED	0	0	D	A	0	0	U — S	R — W	P

Page Table Entry (Points to Page)

PFA	+	Offset	=	Physical Address
(20-bit)		(12-bit)		

03000H	+	08AH	=	03000 08AH

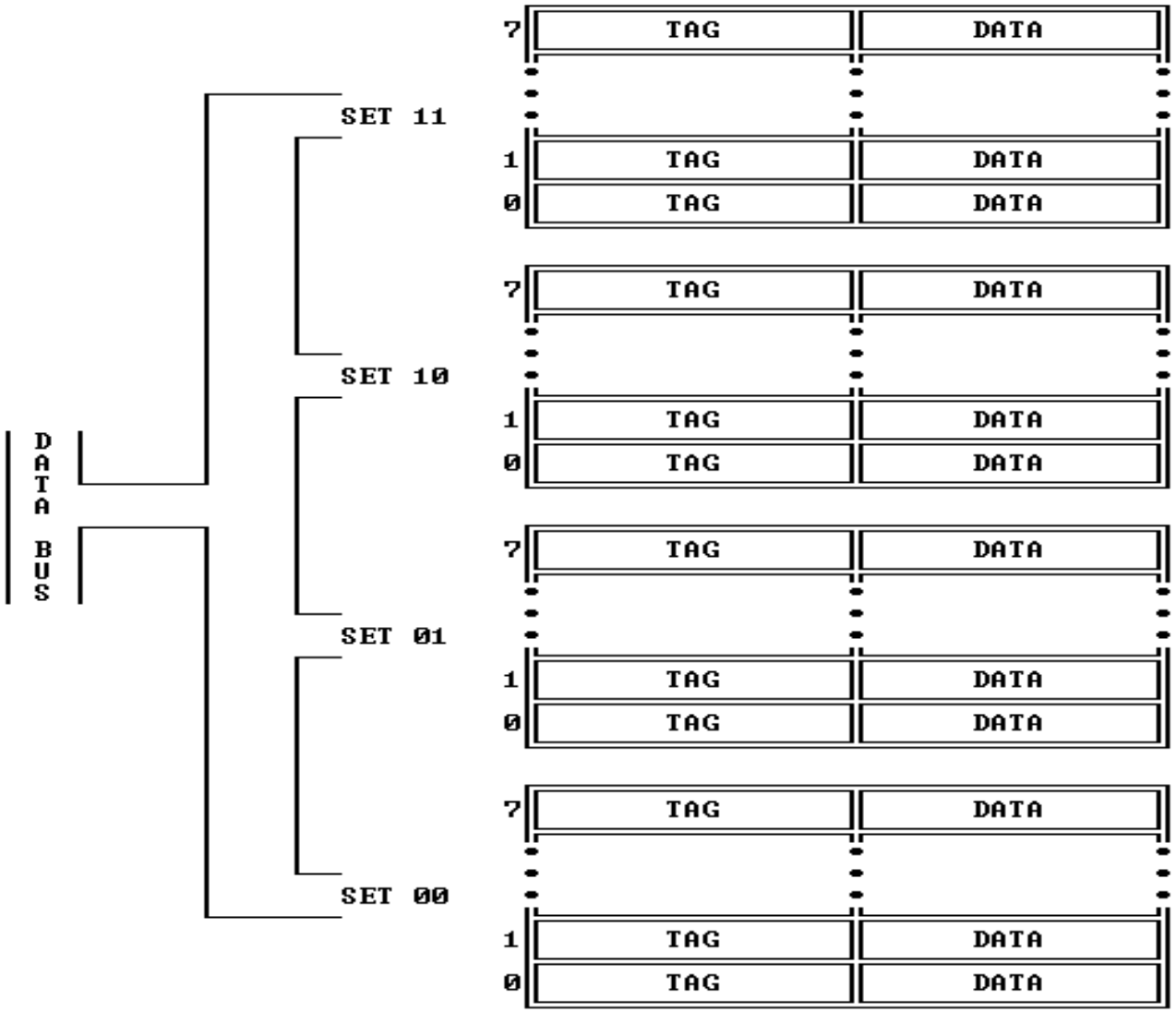
Example


Memory Paging:
Address: 0x0301008A
31
22 21
12 11
0
[00]0C
[00]10
0x08A
[binary]hex
*4 =0x040
*4 =0x030
0x05001040
0x05001000
0x00010030
0x05001
CR3
00010
0x05000000
0x0003008A
0x00030000
0x00030
0x00021000
0x00020000

Translation Lookaside Buffer(TLB)

- Performance degrades if the processor access two levels of tables for every memory reference.
- To solve this problem, the Intel386 DX keeps a cache of the most recently accessed pages and this cache is called **Translation Lookaside Buffer (TLB).**
- TLB is a 4 way set associative 32 entry page table cache

Translation Lookaside Buffer(TLB)



Translation Lookaside Buffer(TLB)

- TLB has 4 sets of eight entries each.
- Each entry consists of a TAG and a DATA.
- Tags are 24 bit wide. They contain 20 upper bits of linear address, a **valid bit** (Validation of Entry) and three attribute **bits(D,U/S and R/W**)
- Data portion of each entry contains upper 20 bits of the Physical address.

TLB Entry

V	D	U/S	R/W	Upper 20 bit Linear Address	Upper 20-bit Physical Address

Translation Lookaside Buffer(TLB)

- It **automatically** keeps the most commonly used Page Table Entries.
- 32-entry TLB coupled with a 4K page size results in the coverage of **128KB of memory addresses.**

Paging Operation

- The paging unit hardware receives a 32-bit linear address from the segmentation unit.
- The upper 20 linear address bits are compared with all 32 entries in the TLB to determine if there is a match.
- If there is a match (i.e. a TLB hit), then the 32-bit physical address is calculated and will be placed on the address bus.

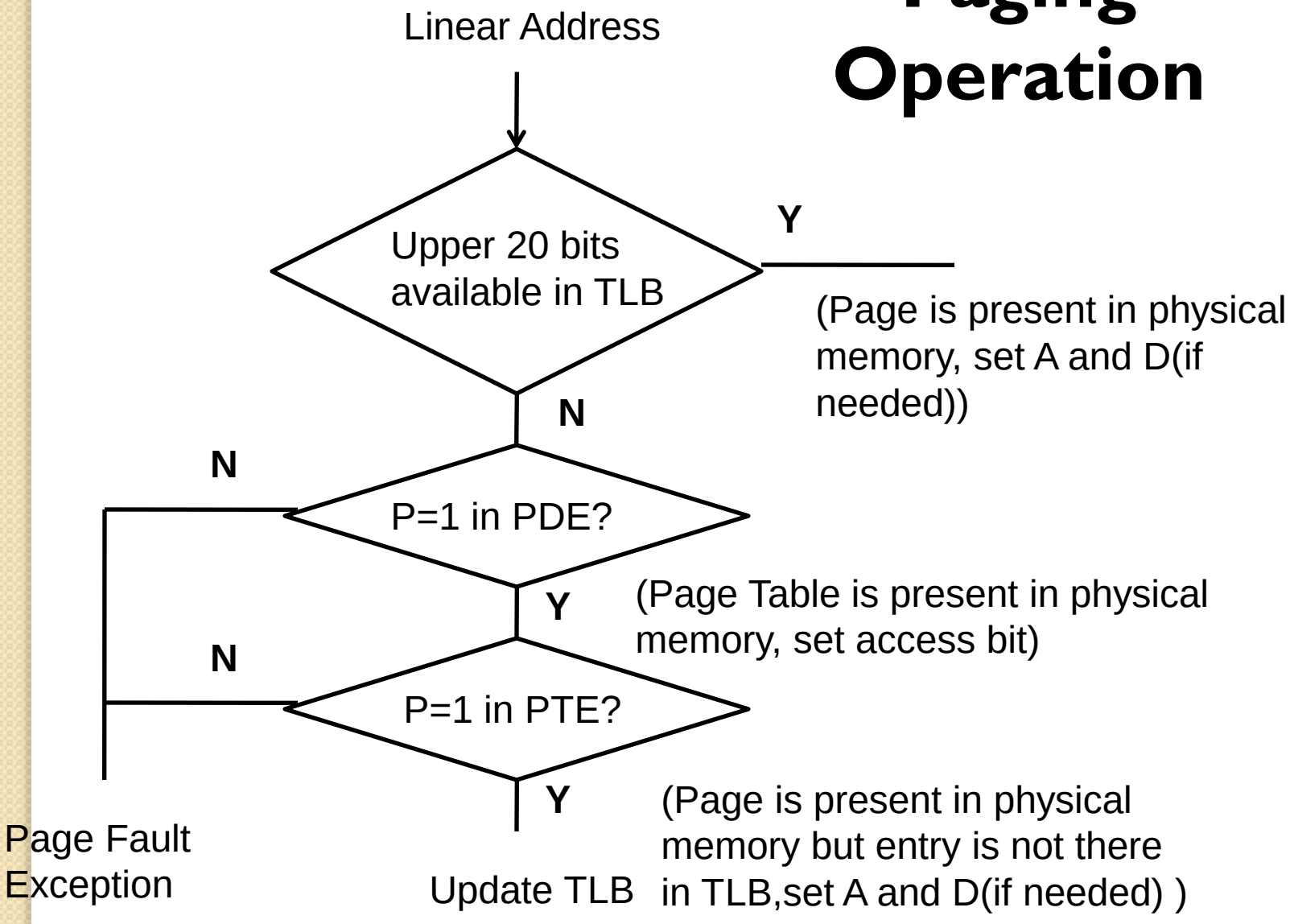
Paging Operation

- If PTE entry is not in TLB, the 80386 DX will read the appropriate PDE Entry.
- If P = 1 on PDE (→the page table is in memory), then the 80386 DX will read the appropriate PTE and set the Access bit.
- If P = 1 on PTE (→ the page is in memory), then the Intel386 DX will update the Access and Dirty bits as needed and fetch the operand.

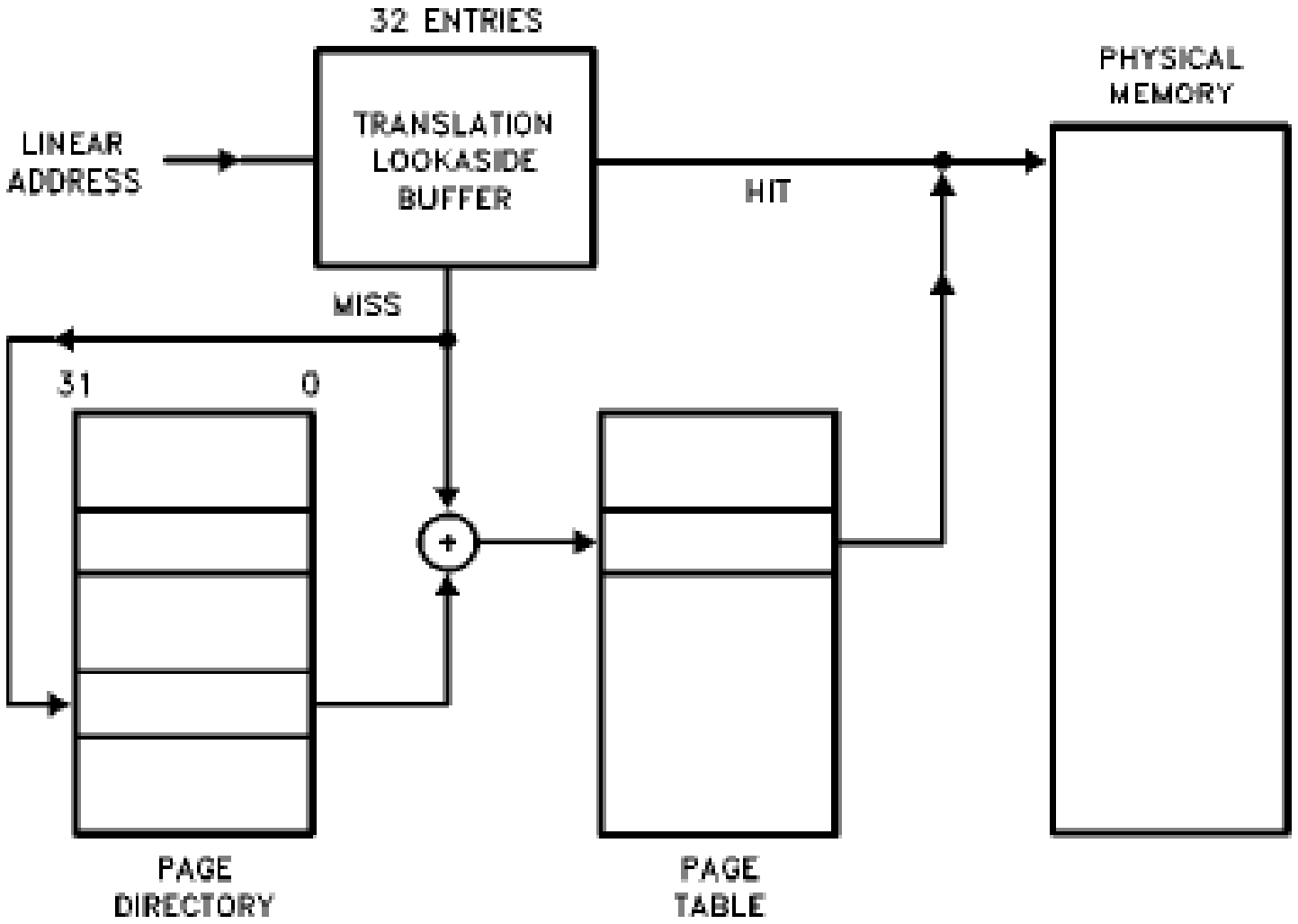
Paging Operation

- The upper 20 bits of the linear address read from the page table will be stored in the TLB for future accesses.
- **If P = 0** for either PDE or PTE, then the processor will generate a **page fault exception**
- This **exception** is also generated when **protection rules are violated** and the CR2 is loaded with the page fault address

Paging Operation

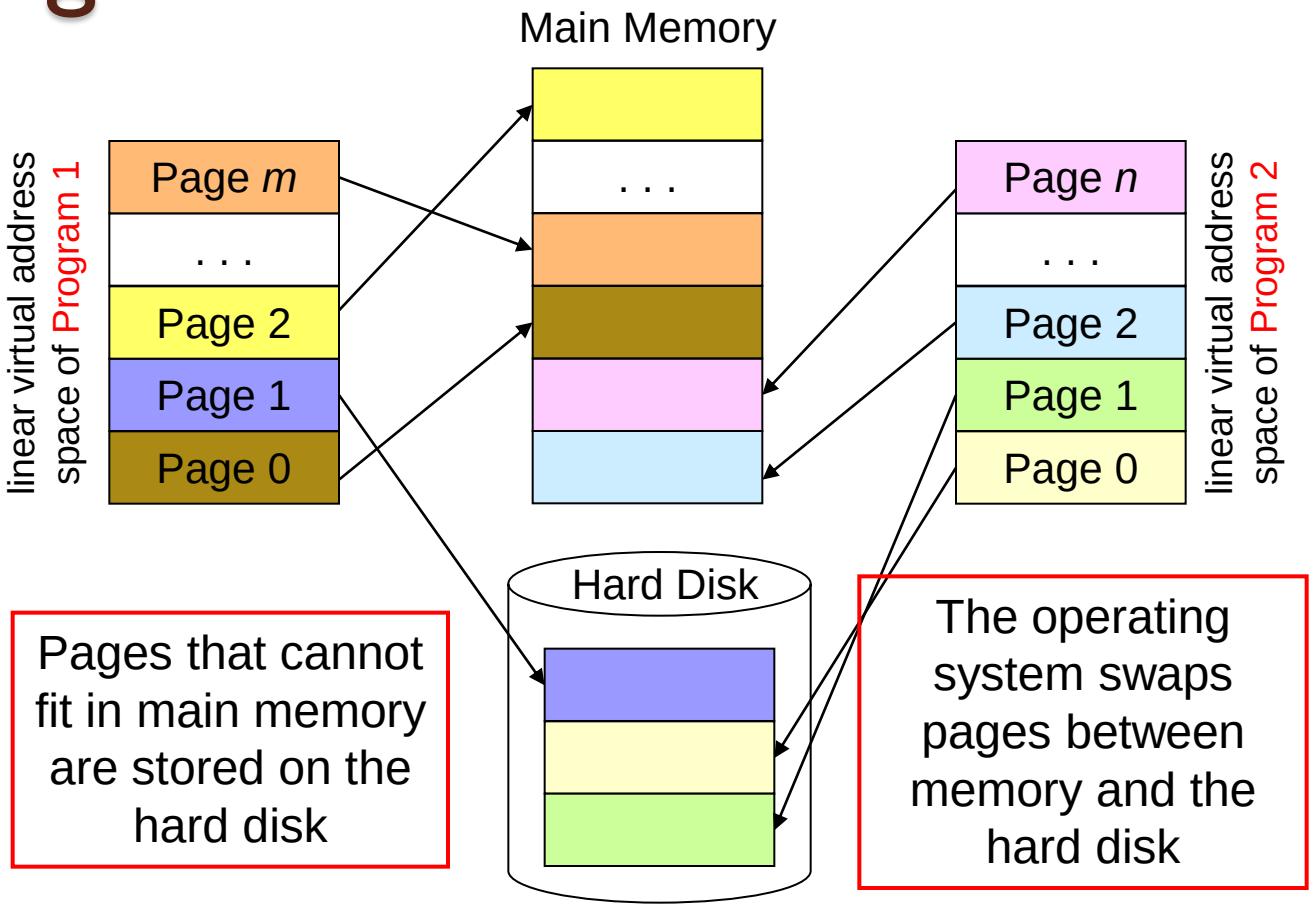


Paging Operation



Paging

The operating system uses page tables to map the pages in the linear virtual address space onto main memory



Each running program has its own page table

Pages that cannot fit in main memory are stored on the hard disk

The operating system swaps pages between memory and the hard disk

As a program is running, the processor translates the linear virtual addresses onto real memory (called also physical) addresses

Thanks for being patient!