Initialization of 80386DX, Debugging and Virtual 8086 Mode

Mr. Sumit Shinde
Assistant Professor
Computer Engineering Department

Pune Institute of Computer Technology

Part 1 : Initialization

Processor State after Reset, Software Initialization for Real Address Mode, Switching to Protected Mode, Software Initialization for Protected Mode, TLB Testing

Part 2 : Debugging

Debugging Features of the Architecture, Debug Registers, Debug Exceptions, Breakpoint Exception

Part 3 :Virtual 8086 Mode

Executing 8086 Code, Structure of V86 Stack, Entering and Leaving Virtual 8086 Mode.

Part 1 : INITIALIZATION

After a signal on the RESET pin, certain registers of the 80386 are set to predefined values.

These values are adequate to enable execution of a bootstrap program, But additional initialization must be performed by software before all the features of the processor can be utilized.

- PROCESSOR STATE AFTER RESET

The contents of EAX : holds zero if the 80386 passed the test (Power on self test).
A nonzero value in EAX indicate 80386 unit is faulty.

DX : It holds a component identifier and revision number after RESET as Figure 5-1 illustrates. DH contains 3, which indicates an 80386 component. DL contains a unique identifier of the revision level.

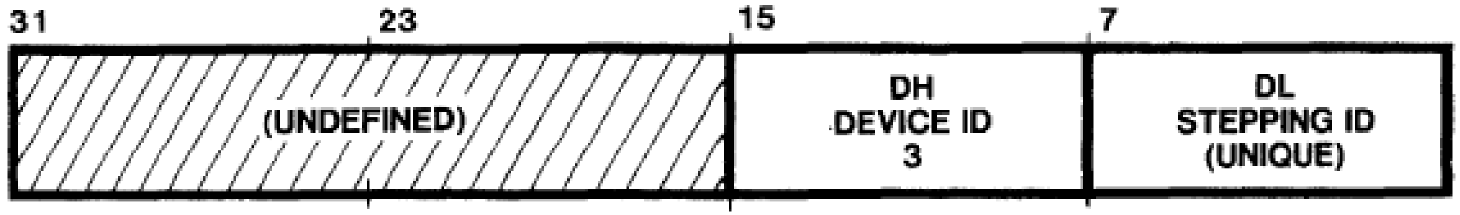


Figure 5-1. Contents of EDX after RESET

The remaining registers and flags are set as follows:

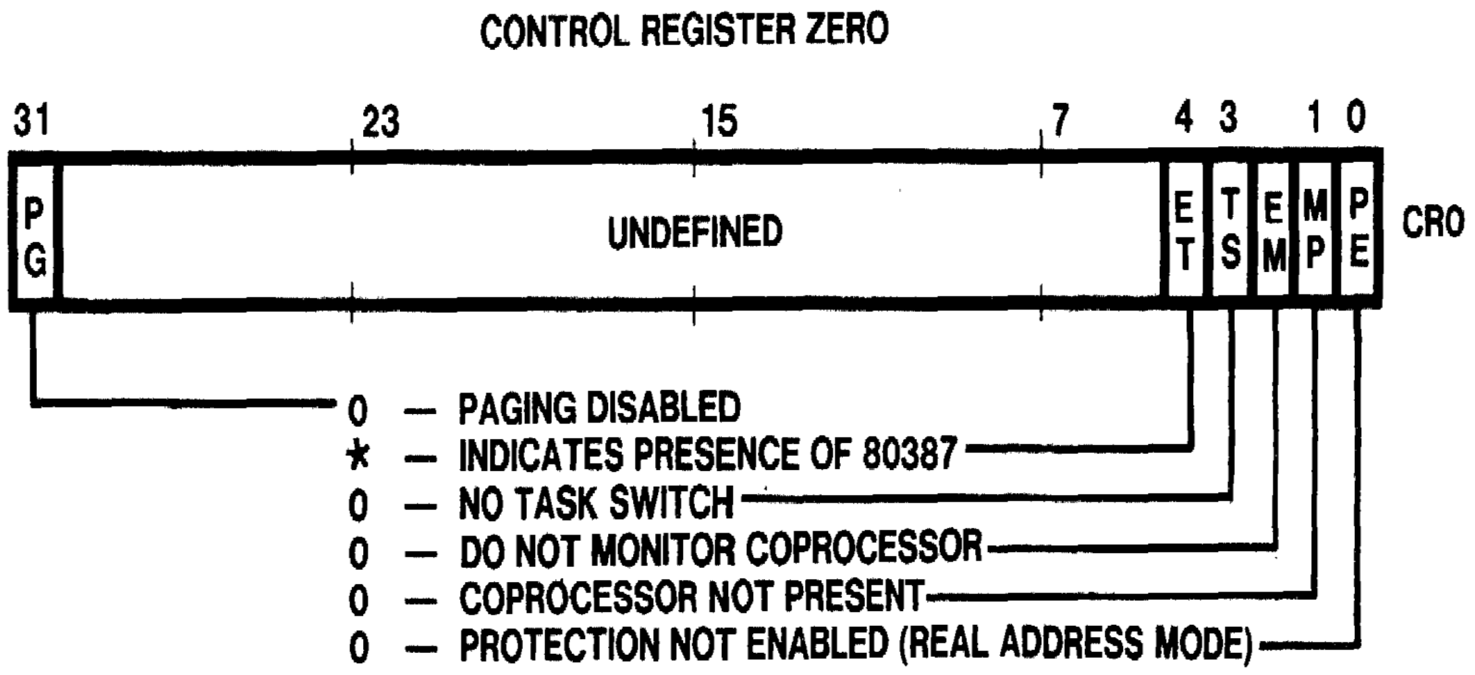
EFLAGS	= 00000002H
IP	= 0000FFFOH
CS selector	= 0000H
DS selector	= 0000H
ES selector	= 0000H
SS selector	= 0000H
FS selector	= 0000H
GS selector	= 0000H

IDTR:

Base	= 0
Limit	= 03FFH

All registers not mentioned above are undefined.

**** This indicates that processor begins in real mode.**


CONTROL REGISTER ZERO
31
23
15
7
4
3
1
0
PG
UNDEFINED
ET
TS
EM
MP
PE
CR0
0 — PAGING DISABLED
* — INDICATES PRESENCE OF 80387
0 — NO TASK SWITCH
0 — DO NOT MONITOR COPROCESSOR
0 — COPROCESSOR NOT PRESENT
0 — PROTECTION NOT ENABLED (REAL ADDRESS MODE)

5.2 SOFTWARE INITIALIZATION FOR REAL-ADDRESS MODE

As initially being in real-address mode a few structures must be initialized before a program can take advantage of all the features available in this mode.

- **Stack – load** stack-segment register (SS)

- **Interrupt Table – All interrupt disable.**

Actions Taken by Initialization Software

•Change limit of IDTR to 0

•Put pointers to valid interrupt handlers

•Change IDTR to point valid IT.

- **First Instruction -** The initial values of CS:IP, causes instruction execution to begin at physical address FFFFFFF0 H

5.3 SWITCHING TO PROTECTED MODE

- Setting the PE bit of the MSW in CRO causes the 80386 to begin executing in protected mode.

- The current privilege level (CPL) starts at zero.

- The segment registers continue to point to the same linear addresses as in real address mode

- flush the processor's instruction prefetch queue.

5.4 SOFTWARE INITIALIZATION FOR PROTECTED MODE

-Interrupt Descriptor Table – Disable all interrupt as real mode

As IT pointed by IDTR has different format for both mode.

-Global Descriptor Table – Valid address in RAM

Need to complete this, before modifying contents of segment register. GDTR must point to valid GDT.

-Page Tables – Valid Page Entries

PG bit must be set to 1 in protected mode.

CR3 PDBR must be loaded before setting PG=1

-First Task

- There must be valid TSS for new task.
- Stack pointers in TSS for privilege levels must point to valid stack segments
- Task register must point to an area in which to save the current task state.

5.5 TLB TESTING

- The 80386 provides a mechanism for testing the Translation Lookaside Buffer (TLB).
- It is the cache used for translating linear addresses to physical addresses.
- *As **failure of the TLB hardware is extremely unlikely,*** So include TLB confidence tests among other power-up confidence tests for the 80386.
- It is recommended to turn off paging PG=0 .

Structure of the TLB :

• The TLB is a four-way set-associative memory.

• Tags are 24-bits wide. They contain the High-order 20 bits of the linear address, The valid bit, and three attribute bits.

• Data portion of each entry contains high order 20 bits of the physical address.

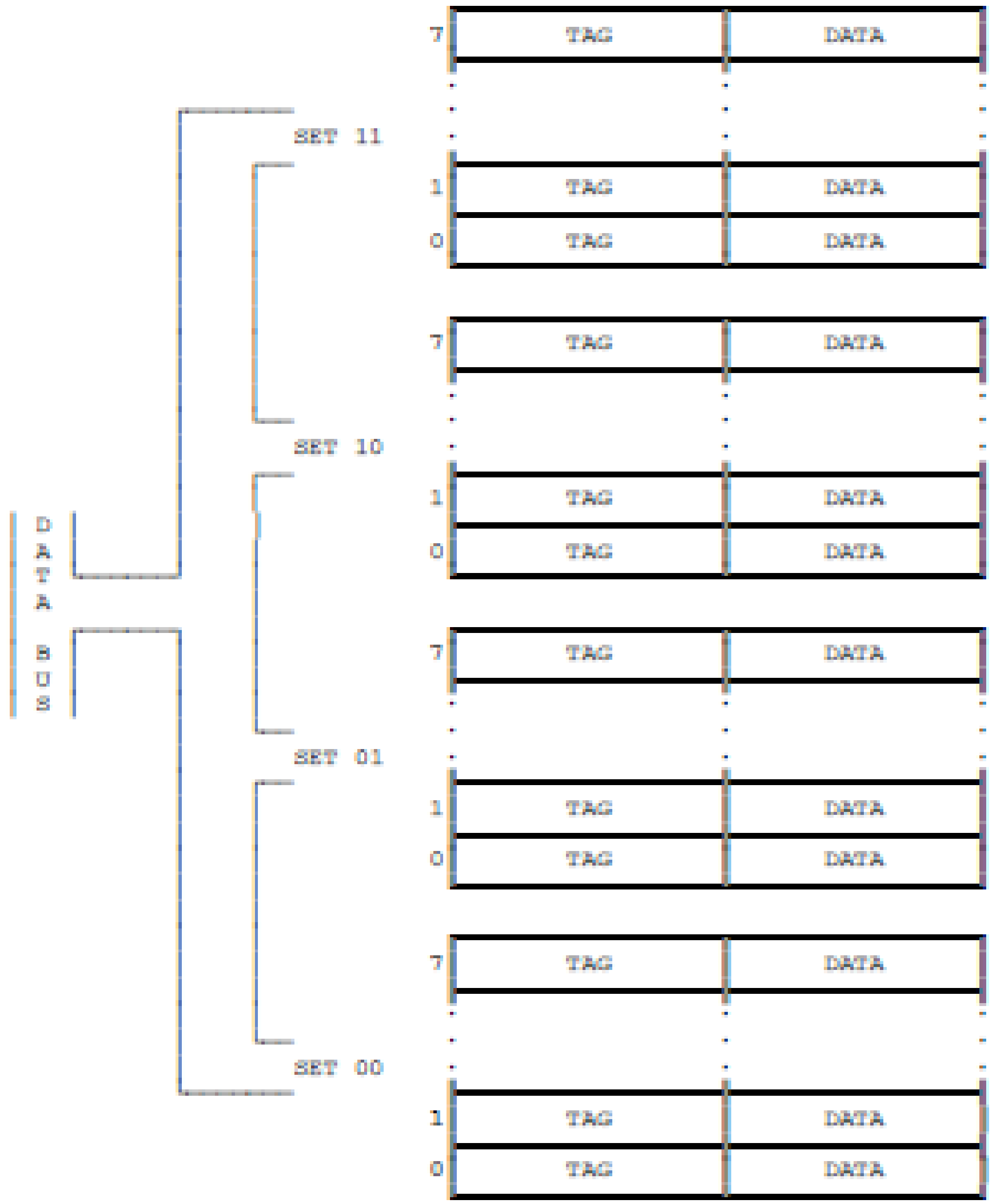
Test Registers

Two test registers for testing TLB

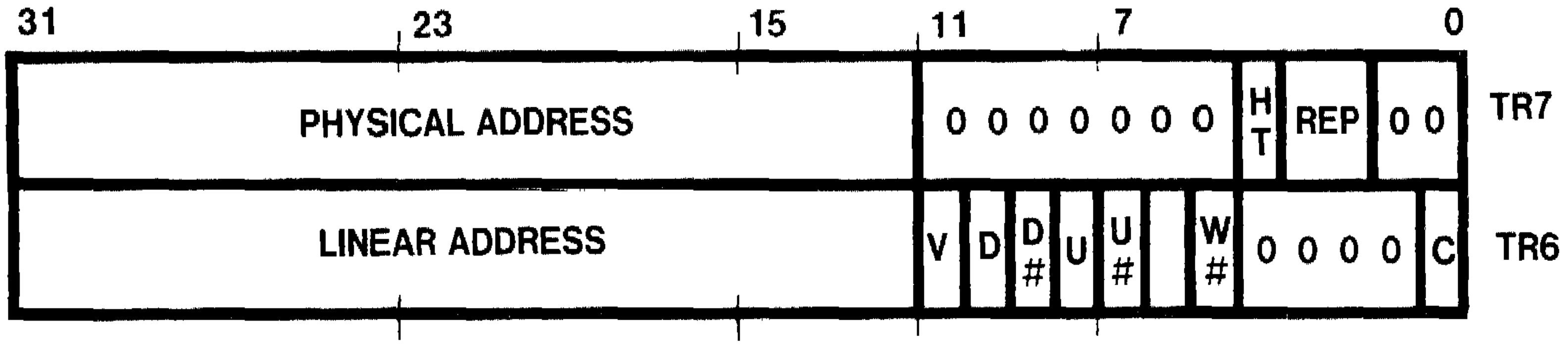
1) TR6 is the *test command register,*
2) TR 7 is the *test data register.*

Test registers are privileged resources, accessed at privilege level 0

Figure 10-3. TLB Structure



Test Registers



Test Command Register (TR6)

It contains a command and an address tag to use in performing the command.

C	This is command bit. C=0 for TLB lookup, C=1 for TLB write
V	Valid bit for TLB entry. TLB uses valid bit to identify entries that contain valid data.
D,D#	The Dirty Bit, for/from the TLB entry.
U, U#	The U/S bit, for/from the TLB entry.
W,W#	The R/W bit, for/from the TLB entry.

X	X#	Effect during TLB Lookup	Value of bit X after TLB Write
0	0	(undefined)	(undefined)
0	1	Match if X=0	Bit X becomes 0
1	0	Match if X=1	Bit X becomes 1
1	1	(undefined)	(undefined)

Test Data Register (TR7)

It contains a command and an address tag to use in performing the command.

Physical Address	Data field of TLB For TLB write, TLB entry set to this value For TLB lookup, HT=1 TLB is read out here, otherwise undefined.
HT	For Lookup HT=1, TLB Hit. HT=0, No Hit For Write, HT must be=1
REP	For Write Selects associative blocks of TLB For Lookup HT=1, reports in which TLB block tag was found. otherwise undefined.

Test Operations

To write a TLB entry:

1. Move a doubleword to TR7 that contains the desired physical address, HT, and REP values. HT must contain 1. REP must point to the associative block in which to place the entry.
2. Move a doubleword to TR6 that contains the appropriate linear address, and values for V, D, U, and W. Be sure C=0 for "write“ command.

To look up (read) a TLB entry:

1. Move a doubleword to TR6 that contains the appropriate linear address and attributes. Be sure C=1 for "lookup" command.
2. Store TR7. If the HT bit in TR7 indicates a hit, then the other values reveal the TLB contents. If HT indicates a miss, then the other values in TR7 are indeterminate.

Part – 2 Debugging

2.1 DEBUGGING FEATURES OF THE ARCHITECTURE

The features of the 80386 architecture that support debugging include:

Reserved debug interrupt vector

Permits processor to automatically invoke a debugger task or procedure when an event occurs that is of interest to the debugger.

Four debug address registers

Permit programmers to specify up to four addresses that the CPU will automatically monitor.

Debug control register

Allows programmers to selectively enable various debug conditions associated with the four debug addresses.

Debug status register

Helps debugger identify condition that caused debug exception.

Trap bit of TSS (T-bit)

Permits monitoring of task switches.

Resume flag (RF) **of flags register**

Allows an instruction to be restarted after a debug exception without immediately causing another debug exception due to the same condition.

Single-step flag (TF)

Allows complete monitoring of program flow by specifying whether the CPU should cause a debug exception with the execution of every instruction.

Breakpoint instruction

Permits debugger intervention at any point III program execution and aids debugging of debugger programs.

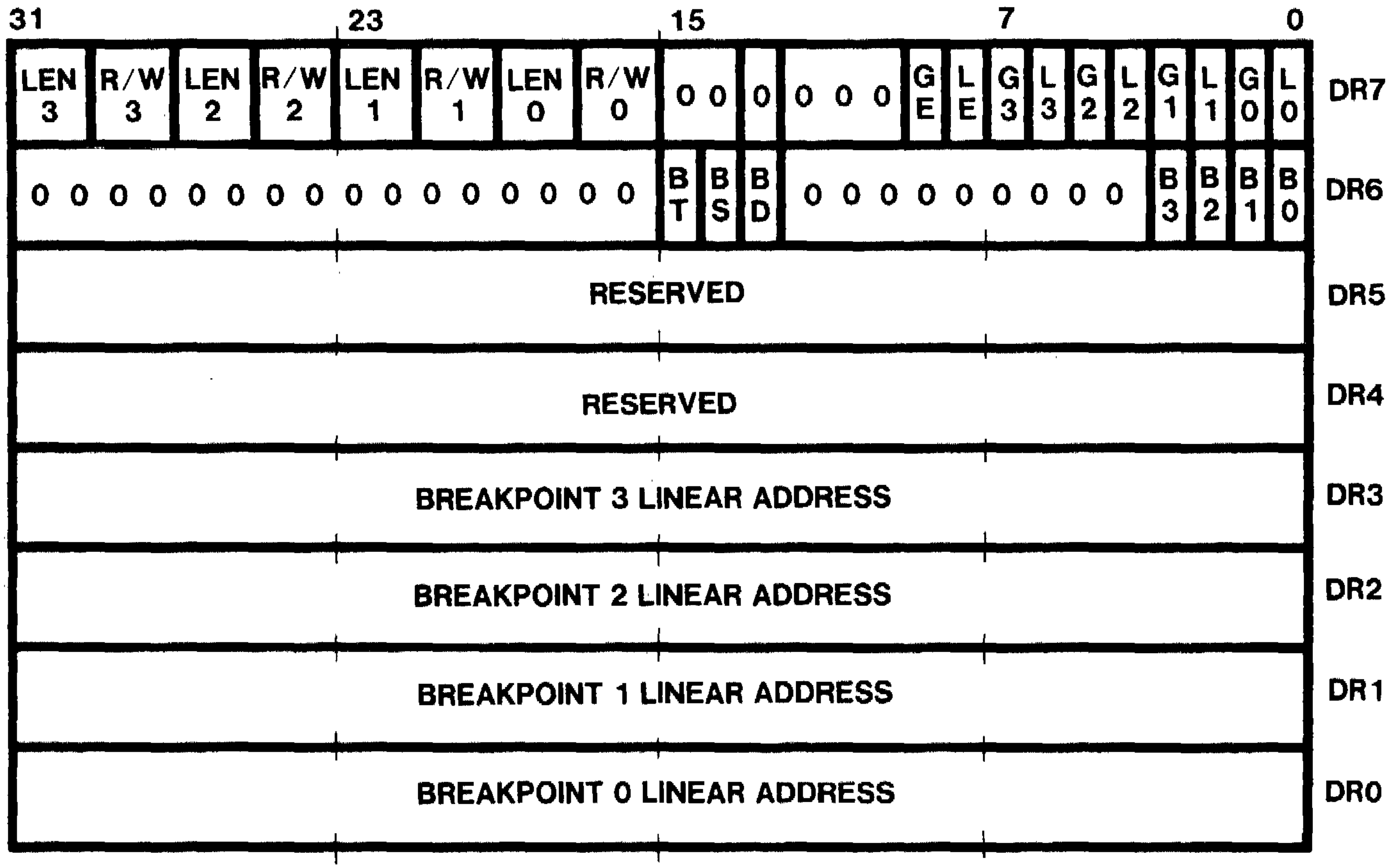
Reserved interrupt vector for breakpoint exception

The debugger can be invoked under any of the following kinds of conditions:

- Task switch to a specific task.
- Execution of the breakpoint instruction.
- Execution of every instruction.
- Execution of any instruction at a given address.

DEBUG REGISTERS

Debug Address Registers



-Debug Control Register (DR 7)

For each address in registers **DRO-DR3,** the corresponding fields **R/WO through R/W3** specify the type of action that should cause a breakpoint.

The processor interprets these bits as follows: -

00 -Break on instruction execution only

01-Break on data writes only

10-*undefined*

11 -Break on data reads or writes but not instruction fetches

Fields **LENO through LEN3** specify the length of data item to be monitored.

00 -one-byte length

01-two-byte length

10-undefined

11-four-byte length

The lower eight bits of DR7 **(LO through L3 and GO through G3)** selectively **enable the four address breakpoint conditions**. There are two levels of enabling: the **local (LO through L3) and global (GO through G3) levels**.

LE and GE bits control the "exact data breakpoint match" feature of the processor

- Debug Status Register (DR6)
The debug status register permits the debugger to determine which debug conditions have occurred.

Debug Exceptions

- Interrupt 1 —— Debug Exceptions
- Interrupt 3 —— Breakpoint Exception

PART 3

VIRTUAL 8086 MODE

The purpose of a V86 task is to form a "virtual machine" with which to execute an 8086 program.

A complete virtual machine consists not only of 80386 hardware but also of systems software. Thus, the emulation of an 8086 is the result of cooperation between hardware and software.

THE REGISTER SET AVAILABLE

8086 programs running as V86 tasks are able to take advantage of the new applications oriented instructions

15.1 EXECUTING 8086 CODE

The processor executes in V86 mode when the VM (virtual machine) bit in the EFLAGS register is set. The processor tests this flag under two general conditions:
1. When loading segment registers to know whether to use **8086-style address formation.**
2. When decoding instructions to determine which **instructions are sensitive to IOPL**.

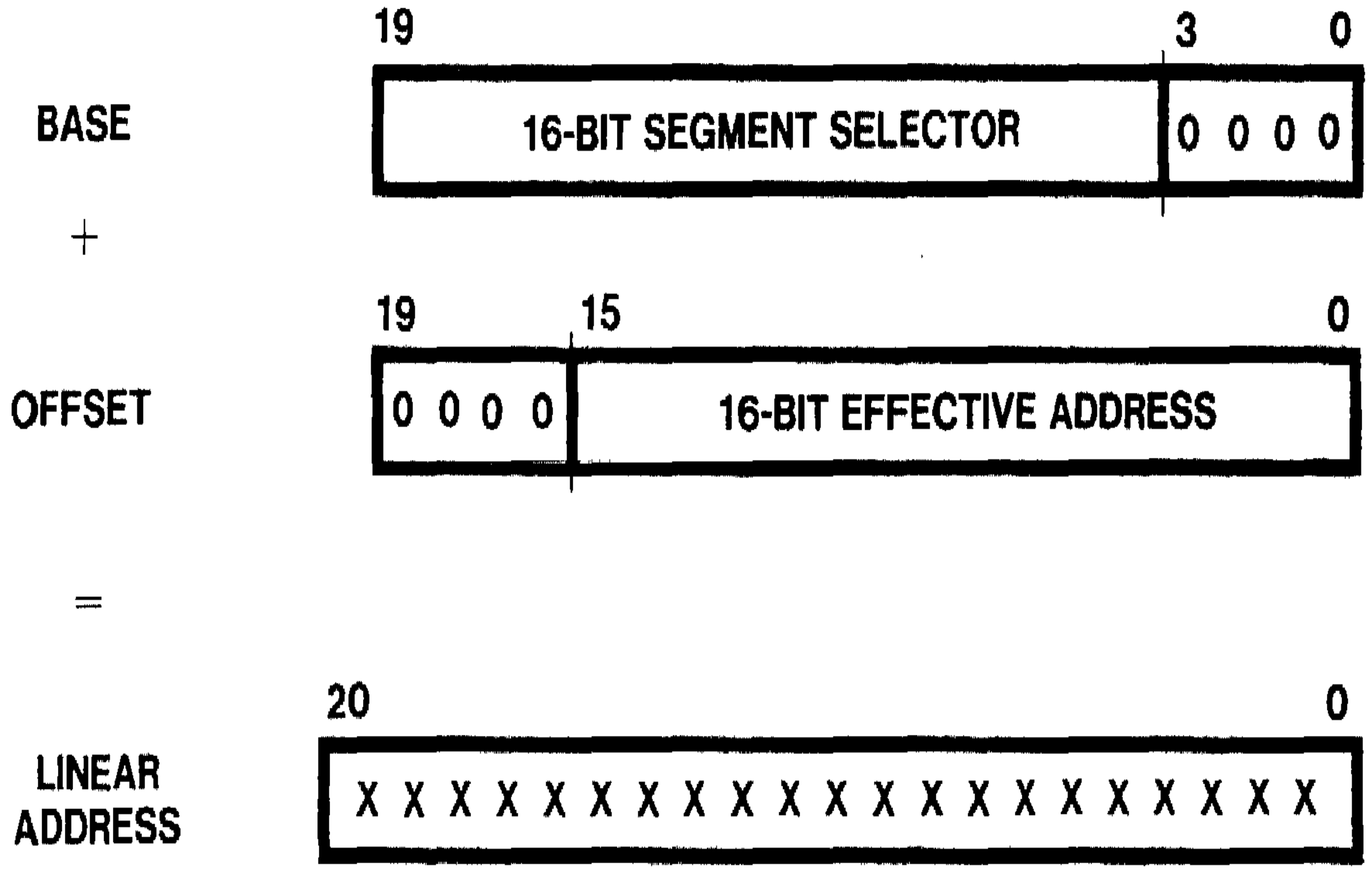
Except for these two modifications to its normal operations, the 80386 in V86 mode operated much as in protected mode.

-**Registers :**
The register set available in V86 mode includes all the registers defined for the 8086 plus the **new registers introduced by the 80386: FS, GS, debug registers, control registers, and test registers**

New instructions introduced by 80386.

- LSS, LFS, LGS instructions
- Long-displacement conditional jumps
- Single-bit instructions
- Bit scan
- Double-shift instructions
- Byte set on condition
- Move with sign/zero extension

- Linear Address Formation



STRUCTURE OF A V86 TASK

A V86 task consists partly of the 8086 program to be executed and partly of 80386 "native mode" code that serves as the virtual-machine monitor.

The task must be represented by an 80386 TSS (not an 80286 TSS). The processor enters V86 mode to execute the 8086 program and returns to protected mode to execute the monitor or other 80386 tasks.

To run successfully in V86 mode, an existing 8086 program needs the following:

- **A V86 monitor.**
- **Operating-system services**

V 86 Monitor : It consists primarily of initialization and exception-handling procedures

USING PAGING FOR V86 TASKS

Paging is not necessary for a single V86 task, but paging is useful or necessary for any of the following reasons:

• To create multiple V86 tasks. Each task must map the lower megabyte of linear addresses to different physical locations.

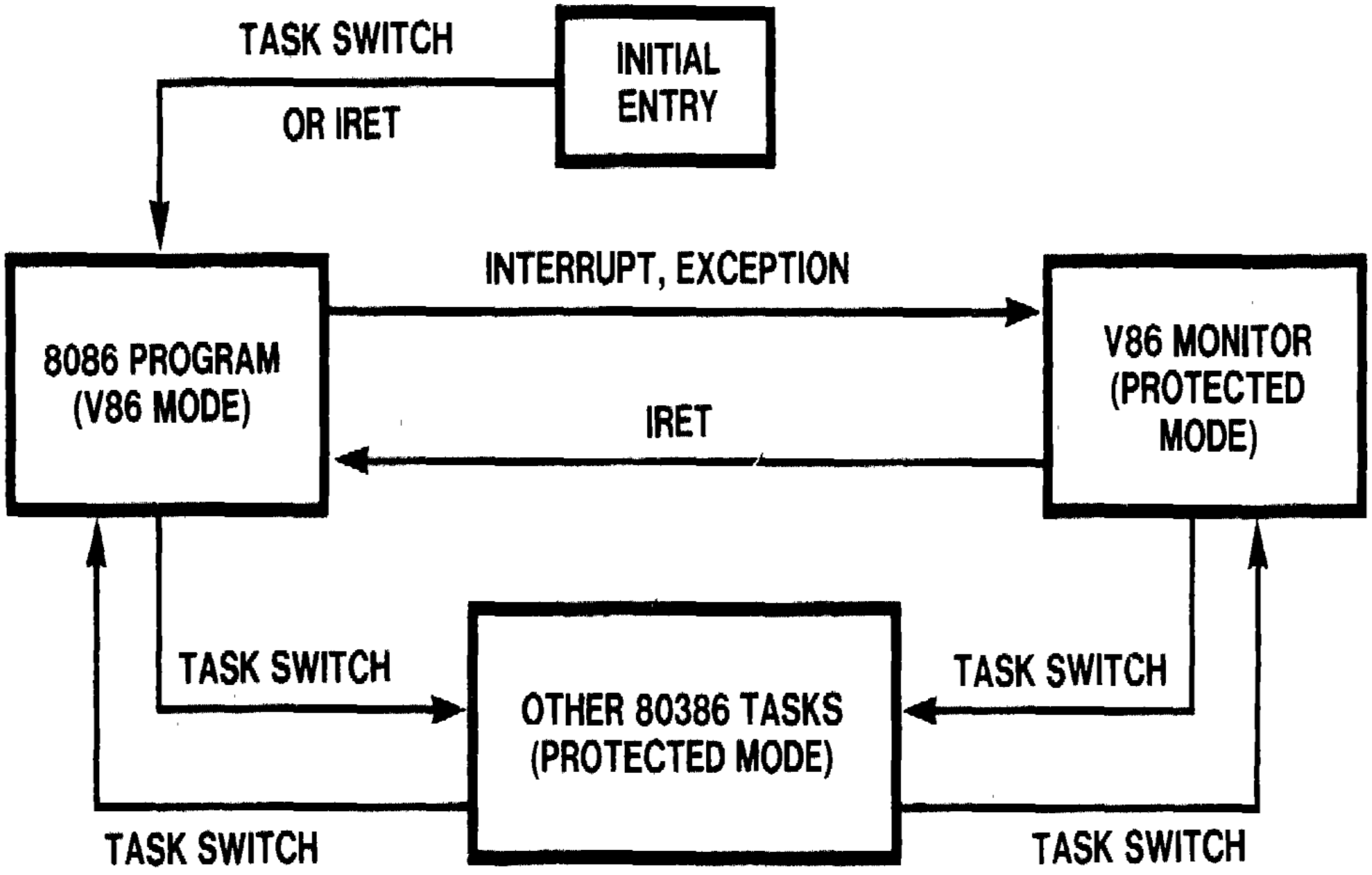
- Protection within a V86 Task

Because it does not refer to descriptors while executing 8086 programs, the processor also does not utilize the protection mechanisms offered by descriptors. To protect the systems software that runs in a V86 task from the 8086 program, software designers may follow either of these approaches:

• Reserve the **first megabyte (plus 64 kilobytes)** of each task's linear address space for the 8086 program.

• Use the **U /S bit** of page-table entries to protect the virtual-machine monitor and other systems software in each virtual 8086 task's space. When the processor is in V86 mode, CPL is 3. (user Priviliges) If the pages of the virtual-machine monitor have supervisor privilege, they cannot be accessed by the 8086 program.

ENTERING AND LEAVING V86 MODE



Thank You !!!