UNIT 5
Initialization of 80386DX, Debugging and Virtual 8086 Mode

Contents

- **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

i. Processor State After Reset

ii. Software Initialization for Real-Address Mode

iii. Switching to Protected Mode

iv. Software Initialization for Protected Mode

v. TLB Testing

i. Processor State After Reset

- **EAX:** holds zero if the 80386 passed the test (Power on self test).

- A nonzero value in EAX indicate 80386 unit is faulty.

- **EDX:** 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 10-1. Contents of EDX after RESET

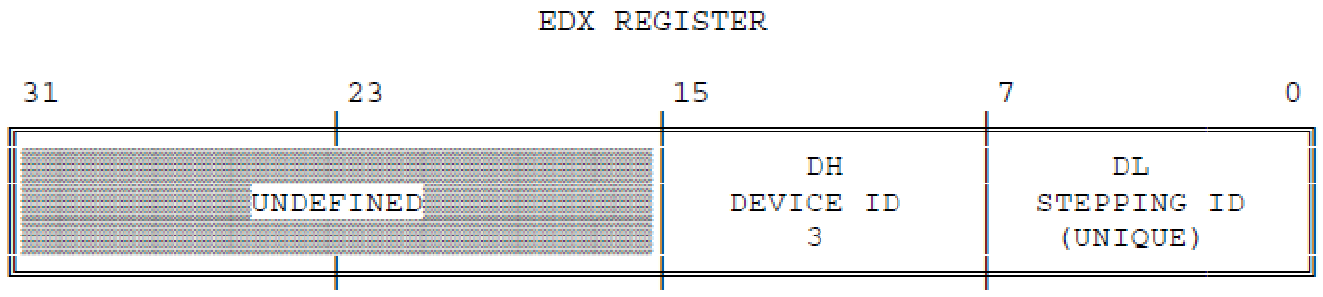
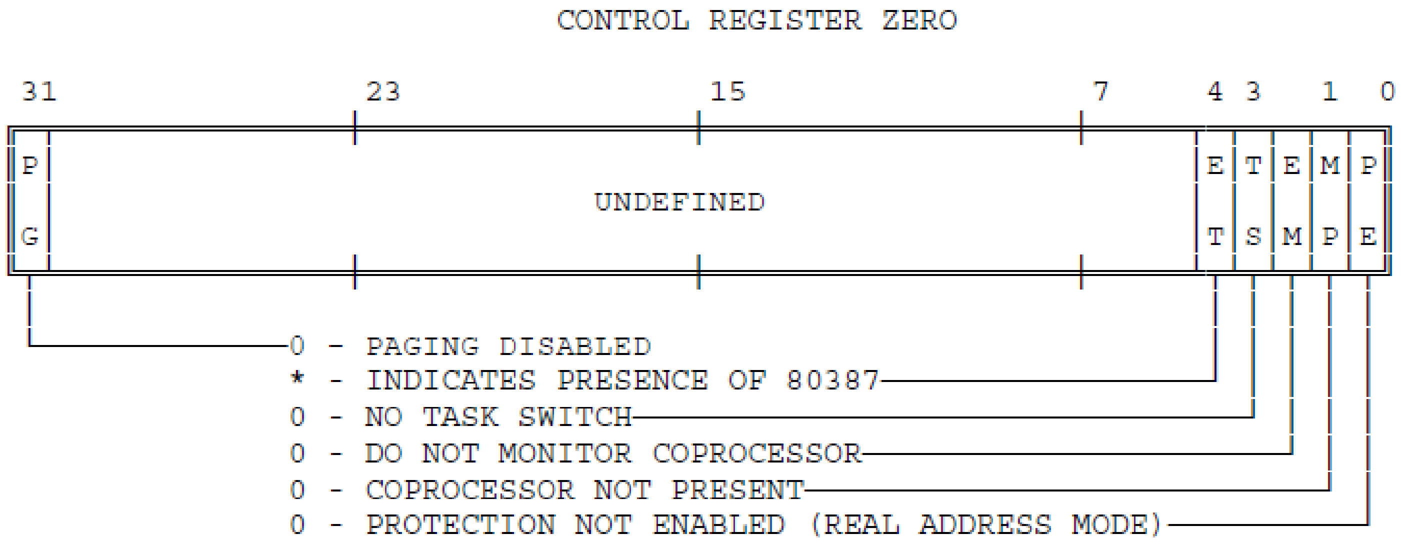


Figure 10-2. Initial Contents of CR0


CONTROL REGISTER ZERO
31
23
15
7
4
3
1
0
P G
UNDEFINED
E T
T S
E M
M P
P E
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)

The remaining registers and flags are set as follows:

EFLAGS	= 00000002 H
IP	= 0000FFF0 H
CS selector	= 0000 H
DS selector	= 0000 H
ES selector	= 0000 H
SS selector	= 0000 H
FS selector	= 0000 H
GS selector	= 0000 H

IDTR:

Base	= 0
Limit	= 03FF H

All registers not mentioned above are undefined.

ii. Software Initialization for Real-Address Mode

- 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**
  - **Interrupt Table**
  - **First Instruction**

- **Stack**
  - No instructions that use the stack can be used until the stack-segment register (SS) has been loaded. SS must point to an area in RAM.

- **Interrupt Table**

  The initial state of the 80386 leaves interrupts disabled;

  Initialization software should take one of the following actions:

  - Change the limit value in the IDTR to zero. This will cause a shutdown if an exception or nonmaskable interrupt occurs.
  - Put pointers to valid interrupt handlers in all positions of the interrupt table that might be used by exceptions or interrupts.
  - Change the IDTR to point to a valid interrupt table.

- **First Instructions**
  - Initially RESET instruction is executed by system.
  - After executing this instruction, CS:IP will have value = **FFFFFFF0 H.**
  - Near (intrasegment) forms of control transfer instructions may be used to pass control to other addresses in the upper 64K bytes of the address space.
  - The first far (intersegment) JMP or CALL instruction causes A {31-20} to drop low, and the 80386 continues executing instructions in the lower one megabyte of physical memory.

iii. Switching to Protected Mode

- **Setting the PE bit of the MSW in CR0** 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.

iv. Software Initialization for Protected Mode

- Most of the initialization needed for protected mode can be done either before or after switching to protected mode.

- If done in protected mode, however, the initialization procedures must not use protected-mode features that are not yet initialized.

- **Interrupt Descriptor Table**

  – Starting address of IDT will be taken from IDTR.

  – IDTR will be loaded when switched to protected mode from real mode.

  – IDT is disabled untill all descriptors are loaded to IDT.

- **Stack**
  - The SS register may be loaded in either real-address mode or protected mode. If loaded in real-address mode, SS continues to point to the same linear base-address after the switch to protected mode.

- **Global Descriptor Table**

  - Before any segment register is changed in protected mode, the GDT register must point to a valid GDT.
  - Initialization of the GDT and GDTR may be done in real-address mode.
  - The GDT (as well as LDTs) should reside in RAM, because the processor modifies the accessed bit of descriptors.

- **Page Tables**
  - Page tables and the PDBR in CR3 can be initialized in either real-address mode or in protected mode;
  - however, the paging enabled (PG) bit of CR0 cannot be set until the processor is in protected mode.
  - PG may be set simultaneously with PE, or later. When PG is set, the PDBR in CR3 should already be initialized with a physical address that points to a valid page directory.

v. TLB Testing

- **Translation Lookaside Buffer (TLB)** is the cache used for translating linear addresses to physical addresses.

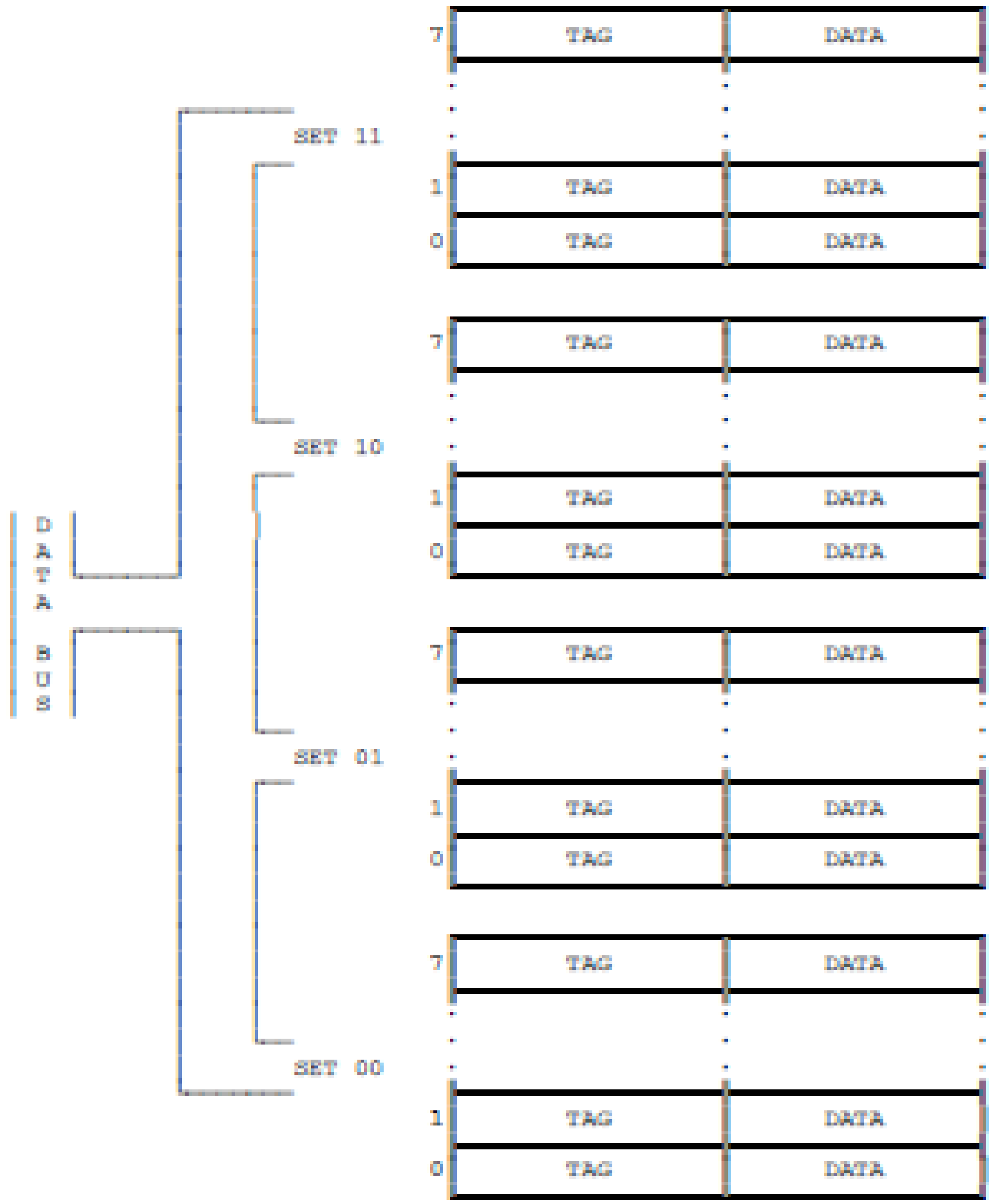
- The 80386 provides a mechanism for testing the TLB.

- Although failure of the TLB hardware is extremely unlikely, so user can include the separate test for it.

Structure of the TLB

- The TLB is a four-way set-associative memory.
- There are four sets of eight entries each.
- Each entry consists of a tag and data. Tags are 24-bits wide. They contain the high-order 20 bits of the linear address, the valid bit, and three attribute bits.
- The data portion of each entry contains the high-order 20 bits of the physical address.

Figure 10-3.   TLB Structure

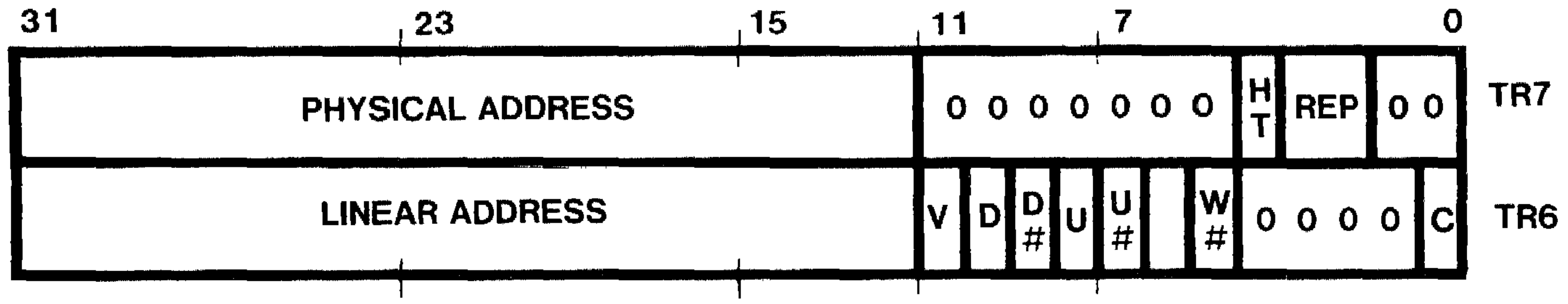


Test Registers

- Two test registers are **provided for the purpose of testing.**
- **TR6 is the Test Command Register, and TR7 is the Test Data Register.**
- These registers are accessed by variants of the MOV instruction.
- A test register may be either the source operand or destination operand.
- The MOV instructions are defined in both real-address mode and protected mode.

- **The test registers are privileged resources;** in protected mode, the MOV instructions that access them can only be executed at privilege level 0.

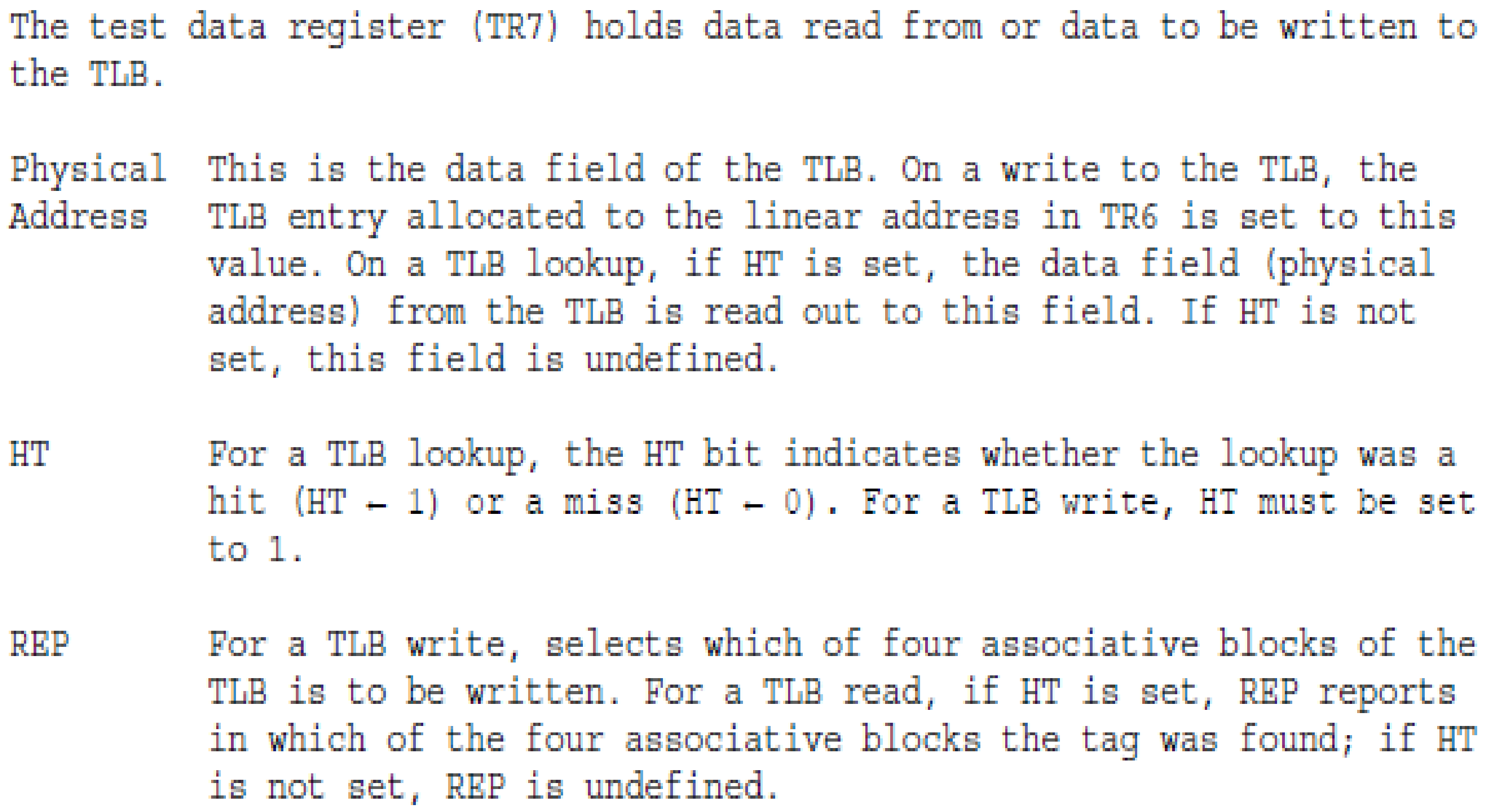
- An attempt to read or write the test registers when executing at any other privilege level causes a **General Protection Exception**.


31
23
15
11
7
0
PHYSICAL ADDRESS
0 0 0 0 0 0 0
HT
REP
0 0
TR7
LINEAR ADDRESS
V
D
D#
U
U#
W#
0 0 0 0
C
TR6

The test command register (TR6) contains a command and an address tag to use in performing the command:

C	This is the command bit. There are two TLB testing commands: write entries into the TLB, and perform TLB lookups. To cause an immediate write into the TLB entry, move a doubleword into TR6 that contains a 0 in this bit. To cause an immediate TLB lookup, move a doubleword into TR6 that contains a 1 in this bit.
Linear Address	On a TLB write, a TLB entry is allocated to this linear address; the rest of that TLB entry is set per the value of TR7 and the value just written into TR6. On a TLB lookup, the TLB is interrogated per this value; if one and only one TLB entry matches, the rest of the fields of TR6 and TR7 are set from the matching TLB entry.
V	The valid bit for this TLB entry. The TLB uses the valid bit to identify entries that contain valid data. Entries of the TLB that have not been assigned values have zero in the valid bit. All valid bits can be cleared by writing to CR3.
D, D#	The dirty bit (and its complement) for/from the TLB entry.
U, U#	The U/S bit (and its complement) for/from the TLB entry.
W, W#	The R/W bit (and its complement) for/from the TLB entry.

The meaning of these pairs of bits is given by Table 10-1, where X represents D, U, or W.



The test data register (TR7) holds data read from or data to be written to the TLB.

Physical Address	This is the data field of the TLB. On a write to the TLB, the TLB entry allocated to the linear address in TR6 is set to this value. On a TLB lookup, if HT is set, the data field (physical address) from the TLB is read out to this field. If HT is not set, this field is undefined.
HT	For a TLB lookup, the HT bit indicates whether the lookup was a hit (HT ← 1) or a miss (HT ← 0). For a TLB write, HT must be set to 1.
REP	For a TLB write, selects which of four associative blocks of the TLB is to be written. For a TLB read, if HT is set, REP reports in which of the four associative blocks the tag was found; if HT is not set, REP is 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.

Be careful not to write duplicate tags; the results of doing so are undefined.

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.

For the purposes of testing, the V bit functions as another bit of addresss. The V bit for a lookup request should usually be set, so that uninitialized tags do not match. Lookups with V=0 are unpredictable if any tags are uninitialized.

Part 2: Debugging

i. Debugging Features of the Architecture

ii. Debug Register

iii. Debug Exceptions

i. Debugging Features of the Architecture

The features of the 80386 architecture that support debugging include:

1. **Reserved debug interrupt vector**
   Permits processor to automatically call a debugger task or procedure when an event occurs that is of interest to the debugger.

2. **Four debug address registers**
   Permit programmers to specify up to four addresses that the CPU will automatically monitor.

3. **Debug control register**
   Allows programmers to selectively enable various debug conditions associated with the four debug addresses.

4. **Debug status register**
   Helps debugger identify condition that caused debug exception.

5. **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.

6. **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.

7. **Breakpoint instruction**

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

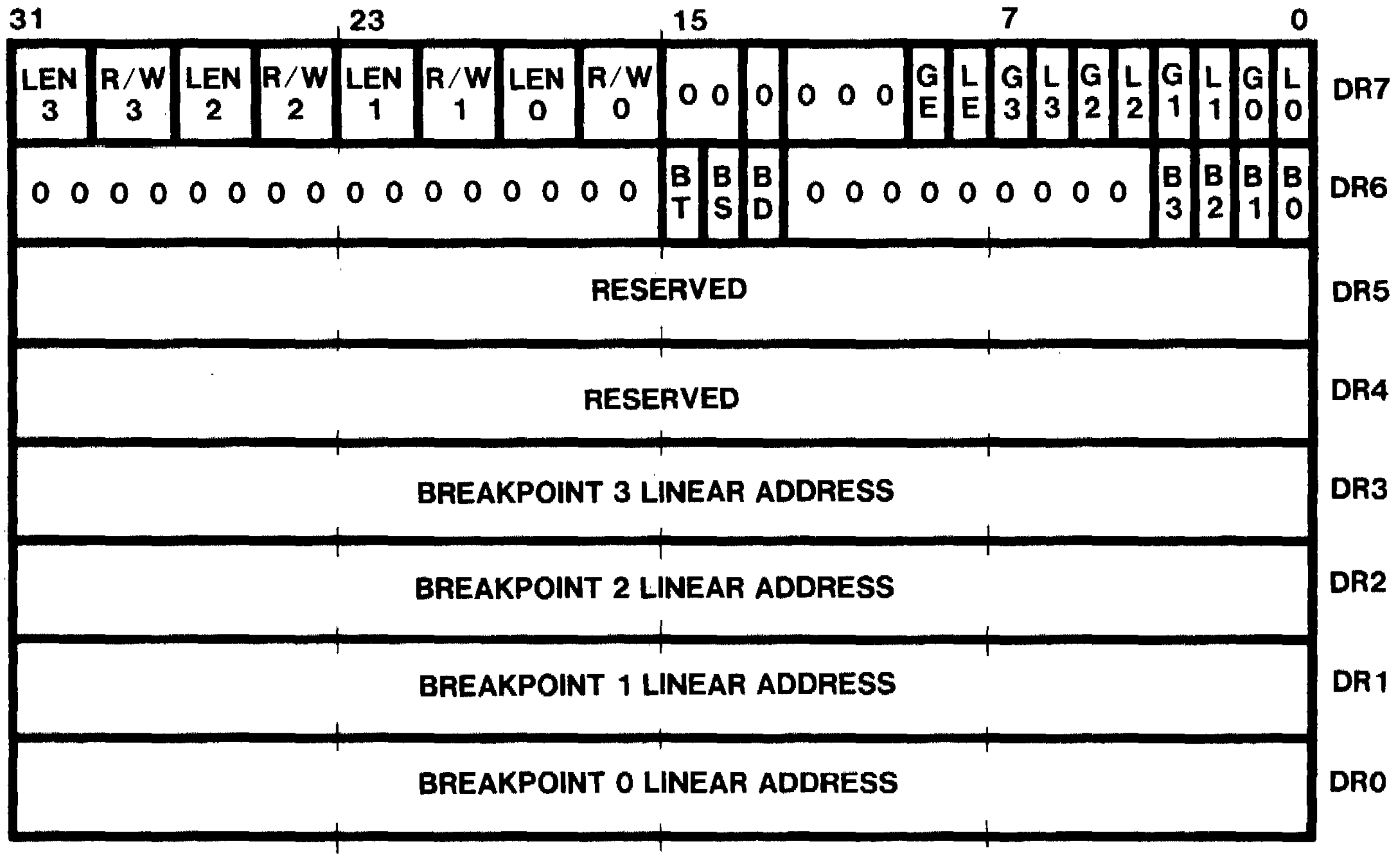
8. **Reserved interrupt vector for breakpoint exception**

   Permits processor to automatically invoke a handler task or procedure upon encountering a breakpoint instruction.

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.
- Read or write of a byte, word or doubleword at any specified address.
- Write to a byte, word or doubleword at any specified address.
- Attempt to change a debug register.

ii. Debug Register



- **Debug Address Registers (DR 0 - DR 3)**
  - Each of these registers **contains the linear address associated** with one of four breakpoint conditions.
  - Each breakpoint condition is further defined by bits in DR7.
  - If paging is enabled, the linear addresses are translated into physical addresses by the processor's paging mechanism .
  - If paging is not enabled, these linear addresses are the same as physical addresses.

- Note that when paging is enabled, **different tasks may have different linear-to-physical address mappings.**

- When this is the case, **an address in a debug address register may be relevant to one task but not to another.**

- For this reason the 80386 has both **Global and Local enable bits in DR7.**

- These bits indicate whether a given debug address has a global (all tasks) or local (current task only) relevance.

- **Debug Control Register (DR7)**
  - **LEN:**

    Defines the **size of access at the breakpoint** address as 00 (byte), 01 (word), 10 (currently not used) or 11 (double word).
  - **RW:**

    Selects the cause of action that enabled breakpoint address as 00 (instruction access), 01 (data write), 10 (Currently Not Used), 11 (data read n write).

- **Debug Status Register (DR6)**

  - **BT:**
    If set the **debug interrupt was caused by a Task Switch.**

  - **BS:**
    If set the **debug interrupt was caused by the TF bit** in the flag register.

  - **BD:**
    If set the **debug interrupt was caused by an attempt to read the debug register with the GD bit set.**

    <u>The GD bit protects access to the debug registers.</u>

  - **B3-B0:**
    Indicate which of the 4 debug breakpoints addresses caused the debug interrupt.

iii. Debug Exceptions

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

Debug Exceptions (INT 1)

- The processor triggers this interrupt for any of a number of conditions; whether the exception is a fault or a trap depends on the condition:
  - Instruction address breakpoint fault.
  - Data address breakpoint trap.
  - General detect fault. (attempt is made to use the debug registers at the same time that 80386 is using them)
  - Single-step trap.
  - Task-switch breakpoint trap.
- The processor does not push an error code for this exception. **An exception handler can examine the debug registers to determine which condition caused the exception.**

Breakpoint (INT 3)

- Exceptions generated because of execution of instruction INT 3.

Part 3: Virtual 8086 Mode

- The 80386 **supports execution of one or more 8086, 8088, 80186 or 80188 programs** in an 80386 protected-mode environment.

- An **8086 program runs in this environment as part of a V86 (virtual 8086) task**.

- V86 tasks take advantage of the hardware support of multitasking offered by the protected mode.

- Not only can there be multiple V86 tasks, each one executing an 8086 program, but **V86 tasks can be multiprogrammed with other 80386 tasks.**

- 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 **Hardware** provides a **Virtual set of Registers** (via the TSS), a **Virtual Memory Space** (the first megabyte of the linear address space of the task), and directly executes all instructions that deal with these registers and with this address space.
  - The **Software** controls the external interfaces of the virtual machine (**I/O, interrupts, and exceptions**) in a manner consistent with the larger environment in which it executes.
- Software that helps implement virtual 8086 machines is called a **V86 monitor**.

Executing 8086 Code

i. Registers and Instructions

ii. Linear Address Formation

i. Registers and Instructions

- 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 that explicitly operate on the segment registers FS and GS are available

8086 programs running as V86 tasks are able to take advantage of the new applications-oriented instructions added to the architecture by the introduction of the 80186/80188, 80286 and 80386:

- New instructions introduced by 80186/80188 and 80286.
  - PUSH immediate data
  - Push all and pop all (PUSHA and POPA)
  - Multiply immediate data
  - Shift and rotate by immediate count
  - String I/O
  - ENTER and LEAVE
  - BOUND

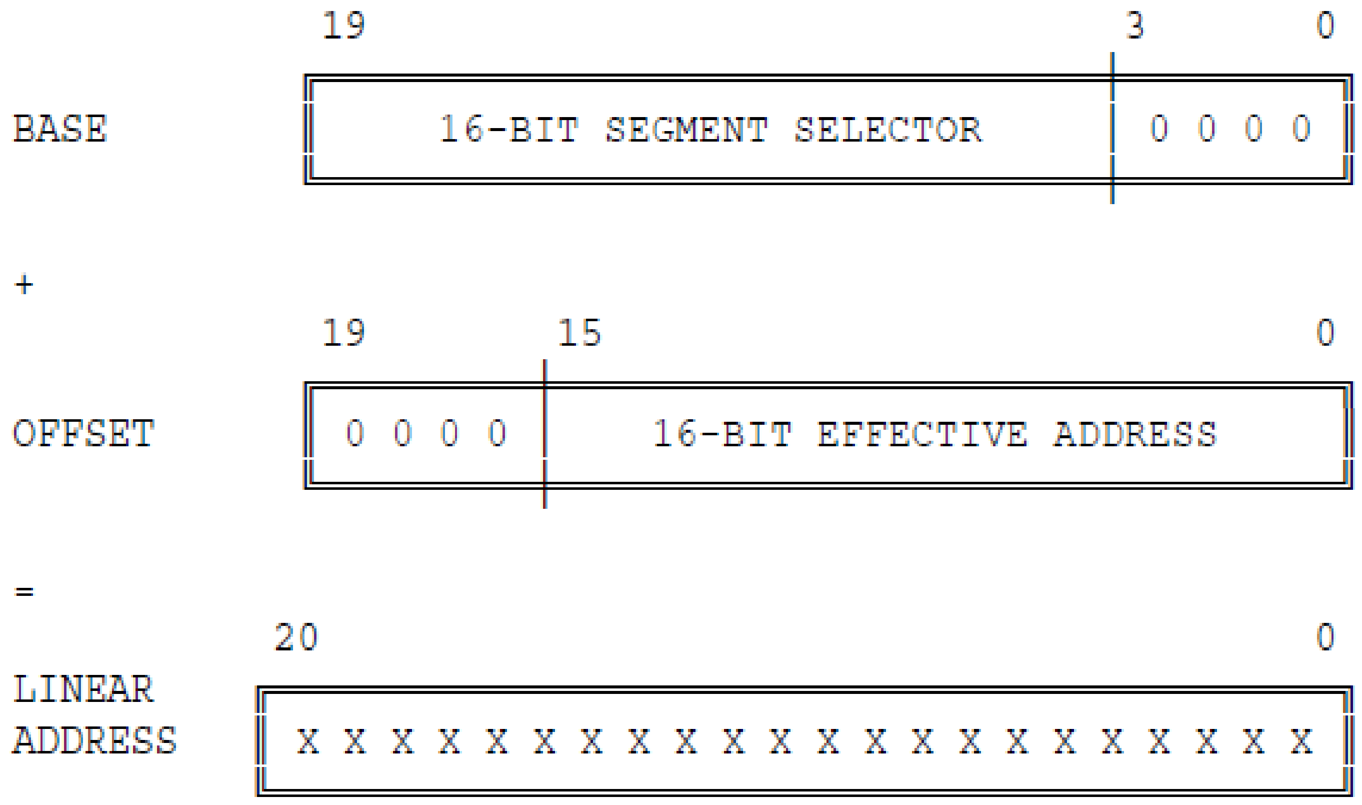
- New instructions introduced by 80386.
  - LSS, LFS, LGS instructions
  - Long-displacement conditional jumps
  - Single-bit instructions
  - Bit scan
  - Double-shift instructions
  - Byte set on condition
  - Move with sign/zero extension
  - Generalized multiply

ii. Linear Address Formation

- It shifts the selector left by four bits to form a 20-bit base address. The effective address is extended with four high-order zeros and added to the base address to create a linear address.

- Because of the possibility of a carry, the **resulting linear address may contain up to 21 significant bits**.

- **An 8086 program may generate linear addresses anywhere in the range 0 to 10FFEFH** (one megabyte plus approximately 64 Kbytes) of the task's linear address space.


19
3
0
BASE
16-BIT SEGMENT SELECTOR
0 0 0 0
+
19
15
0
OFFSET
0 0 0 0
16-BIT EFFECTIVE ADDRESS
=
20
0
LINEAR ADDRESS
X X X X X X X X X X X X X X X X X X X X X X

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.

- The **V86 monitor** is 80386 protected-mode code that executes at **privilege-level zero**.
- The monitor consists primarily of **initialization and exception-handling procedures.**
- As for any other 80386 program, executable-segment descriptors for the monitor must exist in the GDT or in the task's LDT.
- The linear addresses above 10FFEFH are available for the V86 monitor, the operating system, and other systems software.
- The monitor may also need data-segment descriptors so that it can examine the interrupt vector table or other parts of the 8086 program in the first megabyte of the address space.

Protection within a V86 Task

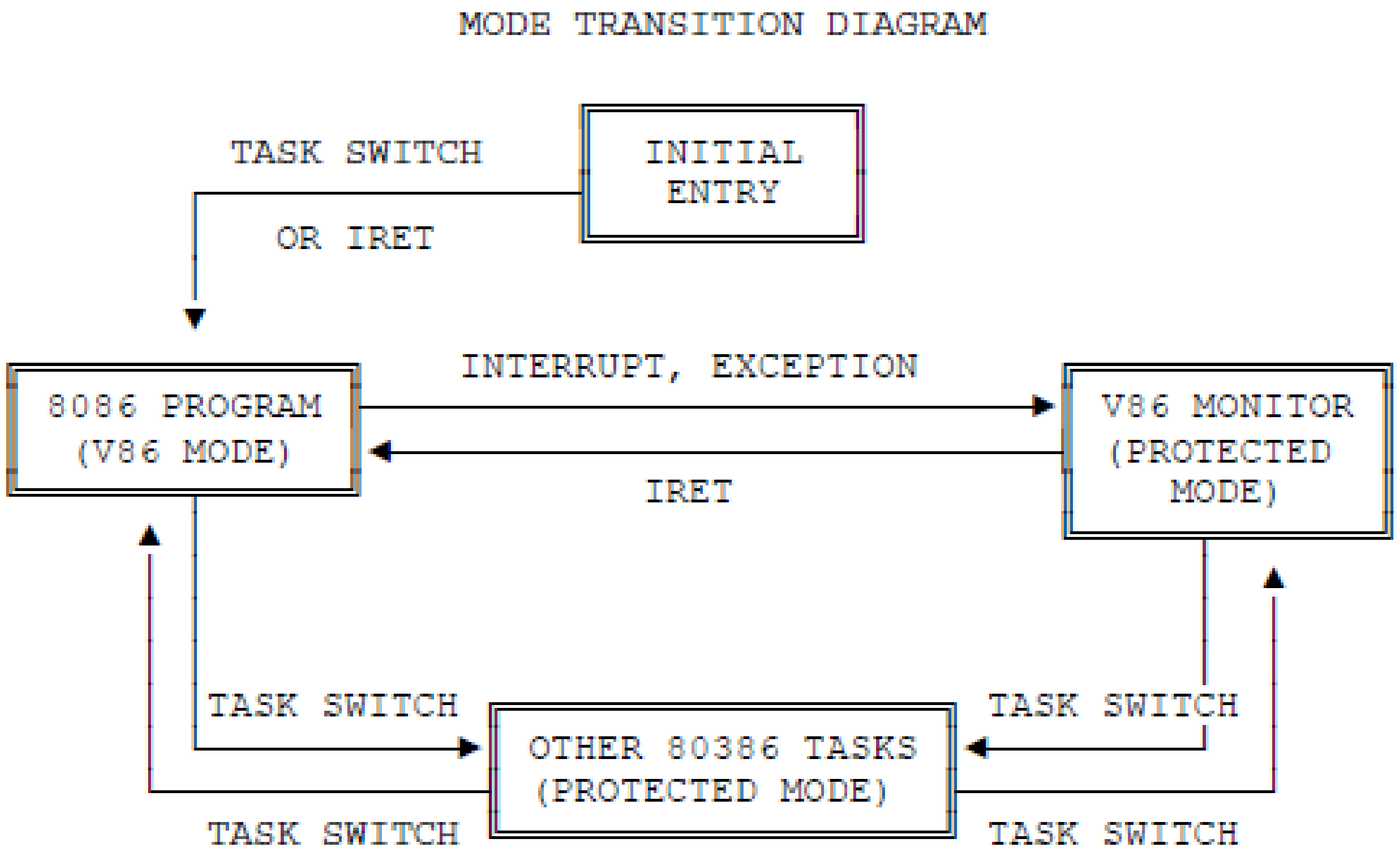
- Because it does not refer to descriptors while executing 8086 programs, the processor also **does not utilize the protection mechanisms offered by descriptors.**

- To protect the systems software that runs in a V86 task from the 8086 program, software designers may follow either of these approaches:

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

2. 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



- **The processor can enter V86 by either of two means:**

1. **A task switch to an 80386 task** loads the image of EFLAGS from the new TSS.
   - The TSS of the new task must be an 80386 TSS, not an 80286 TSS, because the 80286 TSS does not store the high-order word of EFLAGS, which contains the VM flag.
   - A value of one in the VM bit of the new EFLAGS indicates that the new task is executing 8086 instructions; therefore, while loading the segment registers from the TSS, the processor forms base addresses as the 8086 would.

2. An IRET from a procedure of an 80386 task loads the image of EFLAGS from the stack.

- A value of one in VM in this case indicates that the procedure to which control is being returned is an 8086 procedure.

- The CPL at the time the IRET is executed must be zero, else the processor does not change VM.

- **The processor leaves V86 mode when an interrupt or exception occurs. There are two cases:**

1. **The interrupt or exception causes a task switch.**
   - A task switch from a V86 task to any other task loads EFLAGS from the TSS of the new task.
   - If the new TSS is an 80386 TSS and the VM bit in the EFLAGS image is zero or if the new TSS is an 80286 TSS, then the processor clears the VM bit of EFLAGS, loads the segment registers from the new TSS using 80386-style address formation, and begins executing the instructions of the new task according to 80386 protected-mode semantics.

2. The interrupt or exception vectors to a privilege-level zero procedure.

– The processor stores the current setting of EFLAGS on the stack, then clears the VM bit.

– The interrupt or exception handler, therefore, executes as "native" 80386 protected-mode code.

– If an interrupt or exception vectors to a conforming segment or to a privilege level other than three, the processor causes a general-protection exception.