X86 Assembly/Protected Mode

This page is going to discuss the differences between real mode and protected mode operations in the x86 processors. It will also discuss how to enter protected mode, and how to exit protected mode. Modern Operating Systems (Windows, Unix, Linux, BSD, etc...) all operate in protected mode, so most assembly language programmers won't need this information. However, this information will be particularly useful to people who are trying to program kernels or bootloaders.

Contents

Real Mode Operation

When an x86 processor is powered up or reset, it is in real mode. In real mode, the x86 processor essentially acts like a very fast 8086. Only the base instruction set of the processor can be used. Real mode memory address space is limited to 1MiB of addressable memory, and each memory segment is limited to 64KiB. Real Mode is provided essentially for backwards-compatibility with 8086 and 80186 programs.

Protected Mode Operation

In protected mode operation, the x86 can address 16 Mb or 4 GB of address space. This may map directly onto the physical RAM (in which case, if there is less than 4 GB of RAM, some address space is unused), or paging may be used to arbitrarily translate between virtual addresses and physical addresses. In

Protected mode, the segments in memory can be assigned protection, and attempts to violate this protection cause a "General Protection" exception.

Protected mode in the 386, amongst other things, is controlled by the **Control Registers**, which are labelled CR0, CR2, CR3, and CR4.

Protected mode in the 286 is controlled by the **Machine Status Word**.

Long Mode

Long mode was introduced by AMD with the advent of the Athlon64 processor. Long mode allows the microprocessor to access 64-bit memory space, and access 64-bit long registers. Many 16 and 32-bit instructions do not work (or work correctly) in Long Mode. x86-64 processors in Real mode act exactly the like 16 bit chips, and x86-64 chips in protected mode act exactly like 32-bit processors. To unlock the 64-bit capabilities of the chip, the chip must be switched into Long Mode.

Entering Protected Mode

The lowest 5 bits of the control register CR0 contain 5 flags that determine how the system is going to function. This status register has 1 flag that we are particularly interested in: the "Protected Mode Enable" flag (PE). Here are the general steps to entering protected mode:

1. Create a Valid GDT (Global Descriptor Table)
2. Create a 6 byte pseudo-descriptor to point to the GDT
3.
    1. If paging is going to be used, load CR3 with a valid page table, PDBR, or PML4.
    2. If PAE (Physical Address Extension) is going to be used, set CR4.PAE = 1.
    3. If switching to long mode, set IA32_EFER.LME = 1.
4. Disable Interrupts (CLI).
5. Load an IDT pseudo-descriptor that has a null limit (this prevents the real mode IDT from being used in protected mode)
6. Set the PE bit (and the PG bit if paging is going to be enabled) of the MSW or CR0 register
7. Execute a far jump (in case of switching to long mode, even if the destination code segment is a 64-bit code segment, the offset must not exceed 32-bit since the far jump instruction is executed in compatibility mode)
8. Load data segment registers with valid selector(s) to prevent GP exceptions when interrupts happen
9. Load SS:(E)SP with a valid stack
10. Load an IDT pseudo-descriptor that points to the IDT

11. Enable Interrupts.

Following sections will talk more about these steps.

Entering Long Mode

To enter Long Mode on a 64-bit x86 processor (x86-64):

1. If paging is enabled, disable paging.
2. If CR4.PAE is not already set, set it.
3. Set IA32_EFER.LME = 1.
4. Load CR3 with a valid PML4 table.
5. Enable paging.
6. At this point you will be in compatibility mode. A far jump may be executed to switch to long mode. However, the offset must not exceed 32-bit.

Using the CR Registers

Many bits of the CR registers only influence behavior in protected mode.

CR0

The CR0 32-bit register has 6 bits that are of interest to us. The low 5 bits of the CR0 register, and the highest bit. Here is a representation of CR0:

```
CR0: |PG|----RESERVED----|NE|ET|TS|EM|MP|PE|
```

PE
: Bit 0. The Protected Environment flag. This flag puts the system into protected mode when set.

MP
: Bit 1. The Monitor Coprocessor flag. This flag controls the operation of the "WAIT" instruction.

EM
: Bit 2. The Emulate flag. When this flag is set, coprocessor instructions will generate an exception.

TS
: Bit 3. The Task Switched flag. This flag is set automatically when the processor switches to a new task.

ET
: Bit 4. The Extension Type flag. ET (also called "R") tells us which type of

coprocessor is installed. If ET = 0, an 80287 is installed. if ET = 1, an 80387 is installed.

NE

Bit 5. New exceptions. If this flag is clear, FPU exceptions arrive as interrupts. If set, as exceptions.

PG

Bit 31. The Paging flag. When this flag is set, memory paging is enabled. We will talk more about that in a second.

CR2

CR2 contains a value called the **Page Fault Linear Address** (PFLA). When a page fault occurs, the address that access was attempted on is stored in CR2.

CR3

The upper 20 bits of CR3 are called the **Page Directory Base Register** (PDBR). The PDBR holds the physical address of the page directory.

CR4

CR4 contains several flags controlling advanced features of the processor.

Paging

Paging is a special job that microprocessors can perform to make the available amount of memory in a system appear larger and more dynamic than it actually is. In a paging system, a certain amount of space may be laid aside on the hard drive (or on any secondary storage) called the **swap file** or **swap partition**. The **virtual memory** of the system is everything a program can access like memory, and includes physical RAM and the swap space.

The total virtual memory is broken down into chunks or **pages** of memory, each usually being 4096 bytes (although this number can be different on different systems). These pages can then be moved around throughout the virtual memory, and all pointers inside those pages will be automatically directed to point to the new locations by referencing them to a global paging directory that the microprocessor maintains. The pointer to the current paging directory is stored in the CR3 register.

A **page fault** occurs when the system attempts to read from a page that is marked as "not present" in the paging directory/table, when the system attempts to write data beyond the boundaries of a currently available page, or when any number of other errors occur in the paging system. When a page fault occurs, the accessed memory address is stored in the CR2 register.

Other Modes

In addition to real, protected, and long modes, there are other modes that x86 processors can enter, for different uses :

- Virtual 8086 Mode: This is a mode in which application software that was written to run in real mode is executed under the supervision of a protected-mode, multi-tasking OS.
- System Management Mode: This mode enables the processor to perform system tasks, like power management, without disrupting the operating system or other software.

Retrieved from "http://en.wikibooks.org/w/index.php?title=X86_Assembly/Protected_Mode&oldid=2614995"
