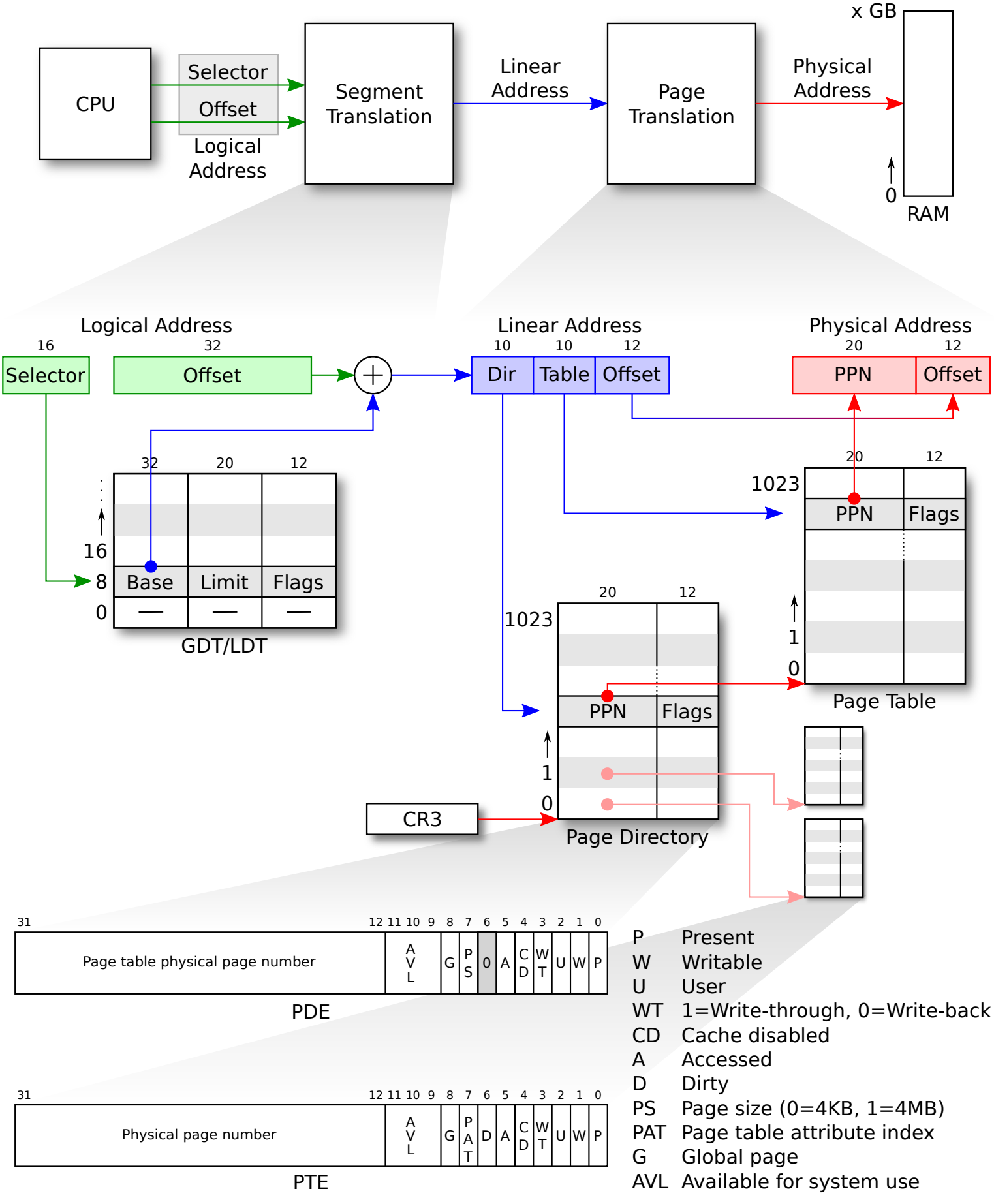
```
Handout for CS 372H
Class 3
24 January 2012

[Credit to Frans Kaashoek, Robert Morris, and Nickolai Zeldovich.]

1. Emulation of CPU in software

    for (;;) {
        read_instruction();
        switch (decode_instruction_opcode()) {
        case OPCODE_ADD:
            int src = decode_src_reg();
            int dst = decode_dst_reg();
            regs[dst] = regs[dst] + regs[src];
            break;
        case OPCODE_SUB:
            int src = decode_src_reg();
            int dst = decode_dst_reg();
            regs[dst] = regs[dst] − regs[src];
            break;
        ...
        }
        eip += instruction_length;
    }


2. Emulate PC's physical memory map

    #define KB      1024
    #define MB      1024*1024

    #define LOW_MEMORY  640*KB
    #define EXT_MEMORY  10*MB

    uint8_t low_mem[LOW_MEMORY];
    uint8_t ext_mem[EXT_MEMORY];
    uint8_t bios_rom[64*KB];

    uint8_t read_byte(uint32_t phys_addr) {
        if (phys_addr < LOW_MEMORY)
            return low_mem[phys_addr];
        else if (phys_addr >= 960*KB && phys_addr < 1*MB)
            return bios_rom[phys_addr − 960*KB];
        else if (phys_addr >= 1*MB && phys_addr < 1*MB+EXT_MEMORY) {
            return ext_mem[phys_addr−1*MB];
        else ...
    }

    void write_byte(uint32_t phys_addr, uint8_t val) {
        if (phys_addr < LOW_MEMORY)
            low_mem[phys_addr] = val;
        else if (phys_addr >= 960*KB && phys_addr < 1*MB)
            ; /* ignore attempted write to ROM! */
        else if (phys_addr >= 1*MB && phys_addr < 1*MB+EXT_MEMORY) {
            ext_mem[phys_addr−1*MB] = val;
        else ...
    }
```

Protected-Mode Address Translation



PDE

PTE

P	Present
W	Writable
U	User
WT	1=Write-through, 0=Write-back
CD	Cache disabled
A	Accessed
D	Dirty
PS	Page size (0=4KB, 1=4MB)
PAT	Page table attribute index
G	Global page
AVL	Available for system use

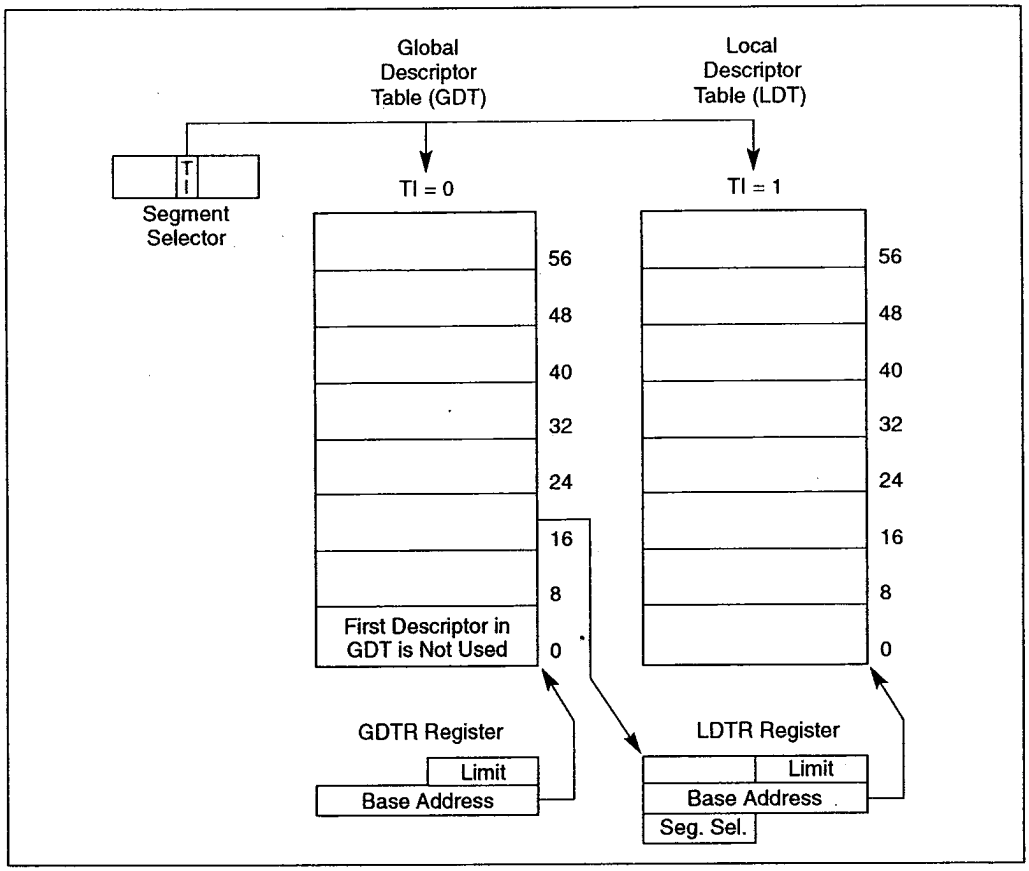


Figure 3-10. Global and Local Descriptor Tables

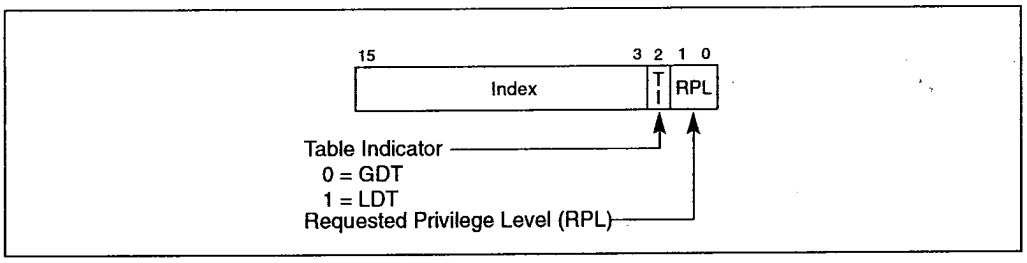


Figure 3-6. Segment Selector

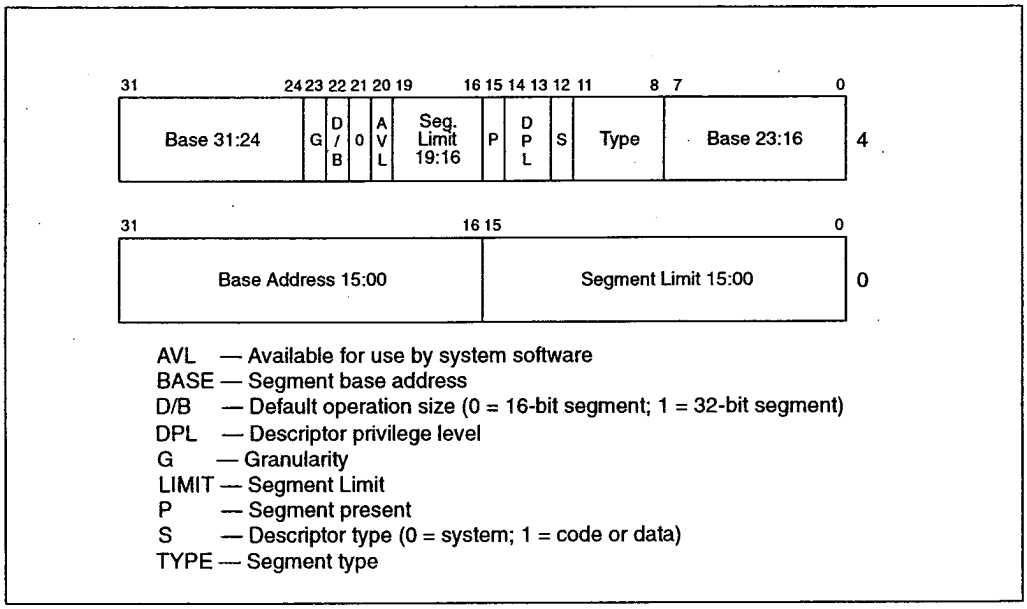


Figure 3-8. Segment Descriptor

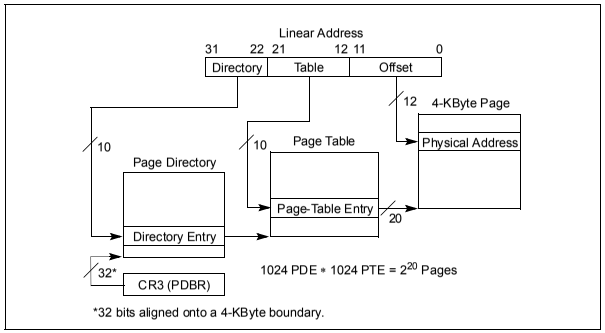


Figure 3-12. Linear Address Translation (4-KByte Pages)

To select the various table entries, the linear address is divided into three sections:

- **Page-directory entry** — Bits 22 through 31 provide an offset to an entry in the page directory. The selected entry provides the base physical address of a page table.
- **Page-table entry** — Bits 12 through 21 of the linear address provide an offset to an entry in the selected page table. This entry provides the base physical address of a page in physical memory.
- **Page offset** — Bits 0 through 11 provides an offset to a physical address in the page.

Memory management software has the option of using one page directory for all programs and tasks, one page directory for each task, or some combination of the two.

3.7.2 Linear Address Translation (4-MByte Pages)

Figure 3-13 shows how a page directory can be used to map linear addresses to 4-MByte pages. The entries in the page directory point to 4-MByte pages in physical memory. This paging method can be used to map up to 1024 pages into a 4-GByte linear address space.

interpreted as the 20 most-significant bits of the physical address, which forces pages to be aligned on 4-KByte boundaries.

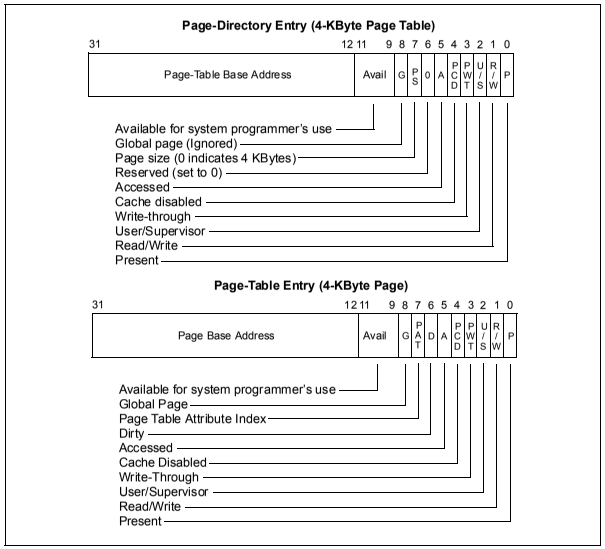


Figure 3-14. Format of Page-Directory and Page-Table Entries for 4-KByte Pages and 32-Bit Physical Addresses

(Page-directory entries for 4-KByte page tables) — Specifies the physical address of the first byte of a page table. The bits in this field are interpreted as the 20 most-significant bits of the physical address, which forces page tables to be aligned on 4-KByte boundaries.

(Page-directory entries for 4-MByte pages) — Specifies the physical address of the first byte of a 4-MByte page. Only bits 22 through 31 of this field are used (and bits 12 through 21 are reserved and must be set to 0, for IA-32 processors through the Pentium II processor). The

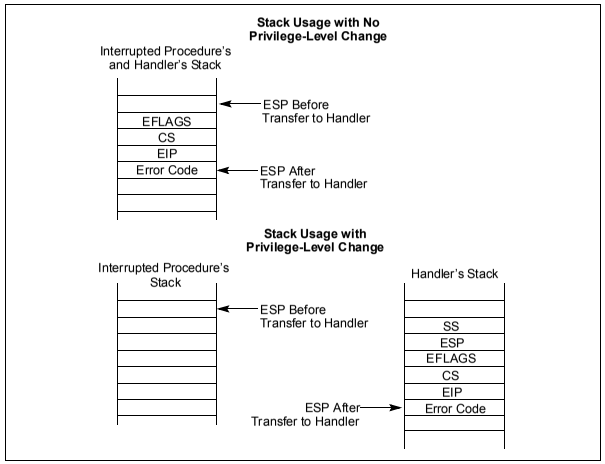


Figure 5-4. Stack Usage on Transfers to Interrupt and Exception-Handling Routines

To return from an exception- or interrupt-handler procedure, the handler must use the IRET (or IRETD) instruction. The IRET instruction is similar to the RET instruction except that it restores the saved flags into the EFLAGS register. The IOPL field of the EFLAGS register is restored only if the CPL is 0. The IF flag is changed only if the CPL is less than or equal to the IOPL. See Chapter 3, "Instruction Set Reference, A-M," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A,* for a description of the complete operation performed by the IRET instruction.

If a stack switch occurred when calling the handler procedure, the IRET instruction switches back to the interrupted procedure's stack on the return.

5.12.1.1 Protection of Exception- and Interrupt-Handler Procedures

The privilege-level protection for exception- and interrupt-handler procedures is similar to that used for ordinary procedure calls when called through a call gate (see Section 4.8.4, "Accessing a Code Segment Through a Call Gate"). The processor does

— The RSVD flag indicates that the processor detected 1s in reserved bits of the page directory, when the PSE or PAE flags in control register CR4 are set to 1. Note:

- The PSE flag is only available in recent Intel 64 and IA-32 processors including the Pentium 4, Intel Xeon, P6 family, and Pentium processors.
- The PAE flag is only available on recent Intel 64 and IA-32 processors including the Pentium 4, Intel Xeon, and P6 family processors.
- In earlier IA-32 processor, the bit position of the RSVD flag is reserved.

— The I/D flag indicates whether the exception was caused by an instruction fetch. This flag is reserved if the processor does not support execute-disable bit or execute disable bit feature is not enabled (see Section 3.10).

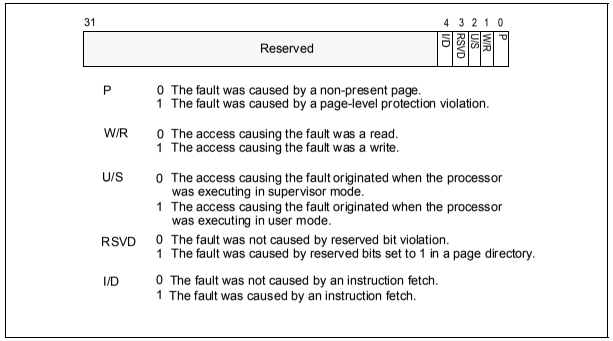


Figure 5-9. Page-Fault Error Code

- The contents of the CR2 register. The processor loads the CR2 register with the 32-bit linear address that generated the exception. The page-fault handler can use this address to locate the corresponding page directory and page-table entries. Another page fault can potentially occur during execution of the page-fault handler; the handler should save the contents of the CR2 register before a second page fault can occur.[1] If a page fault is caused by a page-level protection

1. Processors update CR2 whenever a page fault is detected. If a second page fault occurs while an earlier page fault is being delivered, the faulting linear address of the second fault will overwrite the contents of CR2 (replacing the previous address). These updates to CR2 occur even if the page fault results in a double fault or occurs during the delivery of a double fault.

JOS Virtual Memory Map