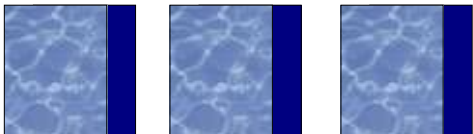
Agenda

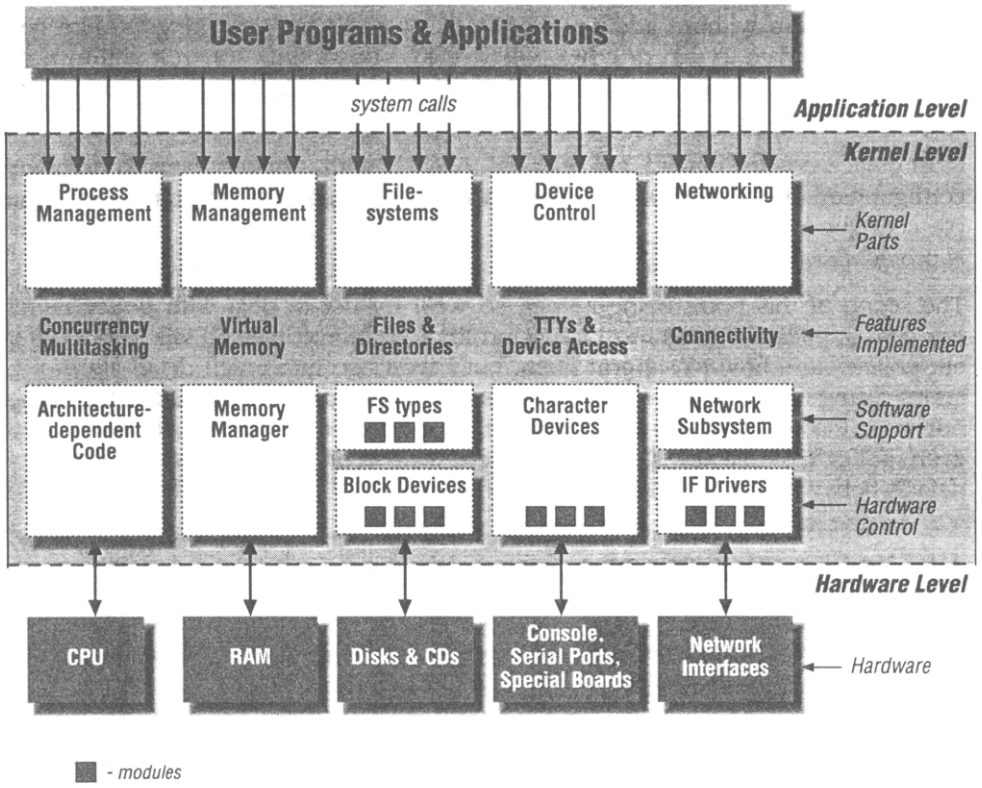
- Programming Model
  - Linux
  - x86
- Architecture and respective adaption
  - Memory
  - Exceptions / Interrupts
  - Multiprocessing



References

- Understanding the Linux Kernel,
  Bovet and Cesati
- Linux Device Drivers, Rubini
- 2.2.18
  - lxr.linux.no
  - http://os.inf.tu-dresden.de/dxr

Linux Kernel



IRQ - arch. Independent

- IRQ Handler
  - irq_desc array
  - irq_action for interrupt sharing
- Execution Modell
  - Top Half
  - Bottom Half
- Dynamic Management

IRQ - arch. Dependent

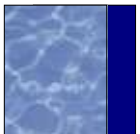
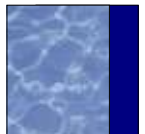
- CPU interrupt delivery
  - Entering to
  - Returning from
- Interrupt controller
- enabling/disabling
- Switch to kernel stack

Memory - arch. Independent

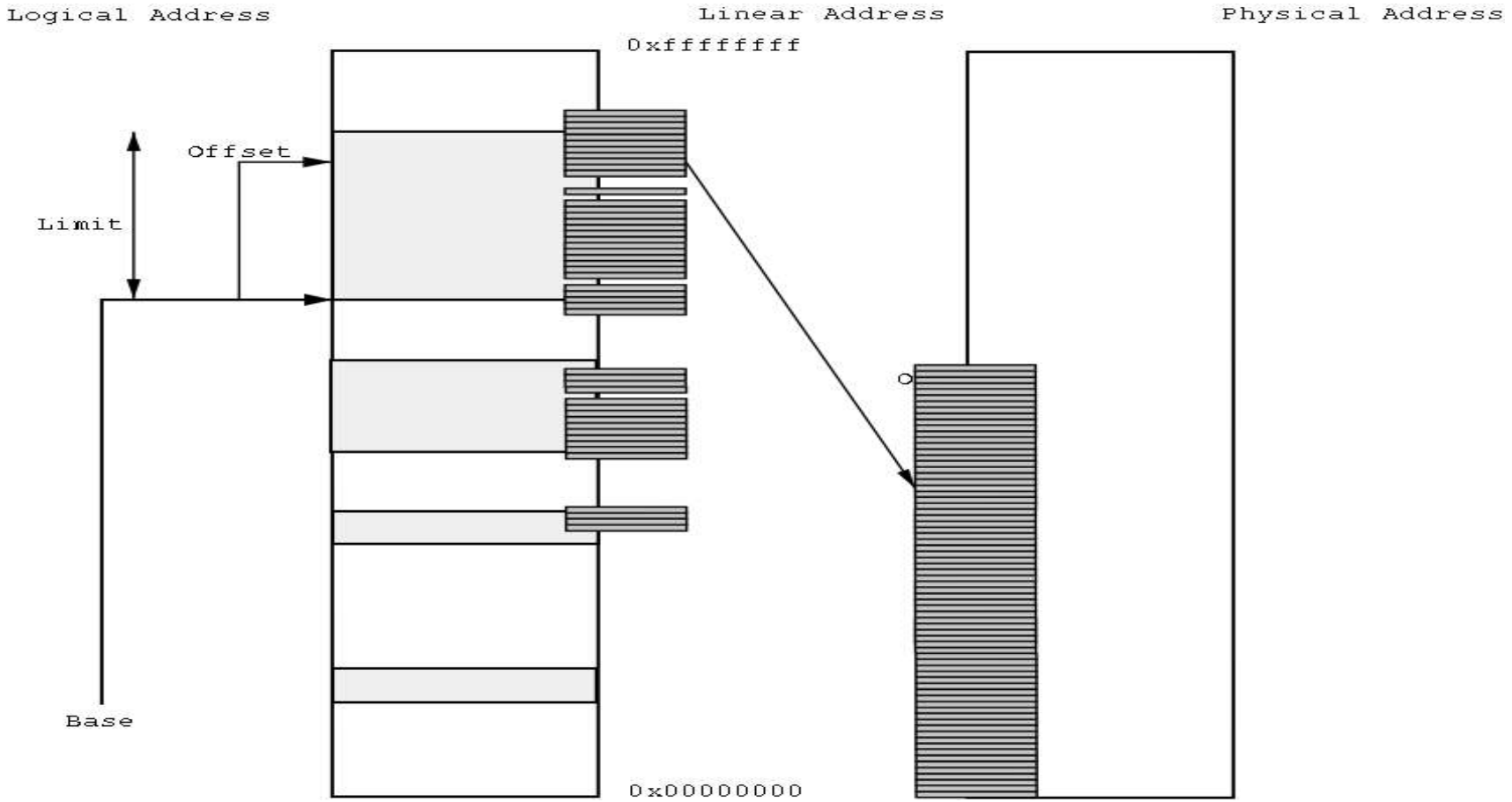
- Process
  - VMA , logical segmentation
  - 3 - level page table
- Physical Memory
  - Page frame
  - Memory areas
  - Non-contiguous memory areas

Memory - arch. Dependent

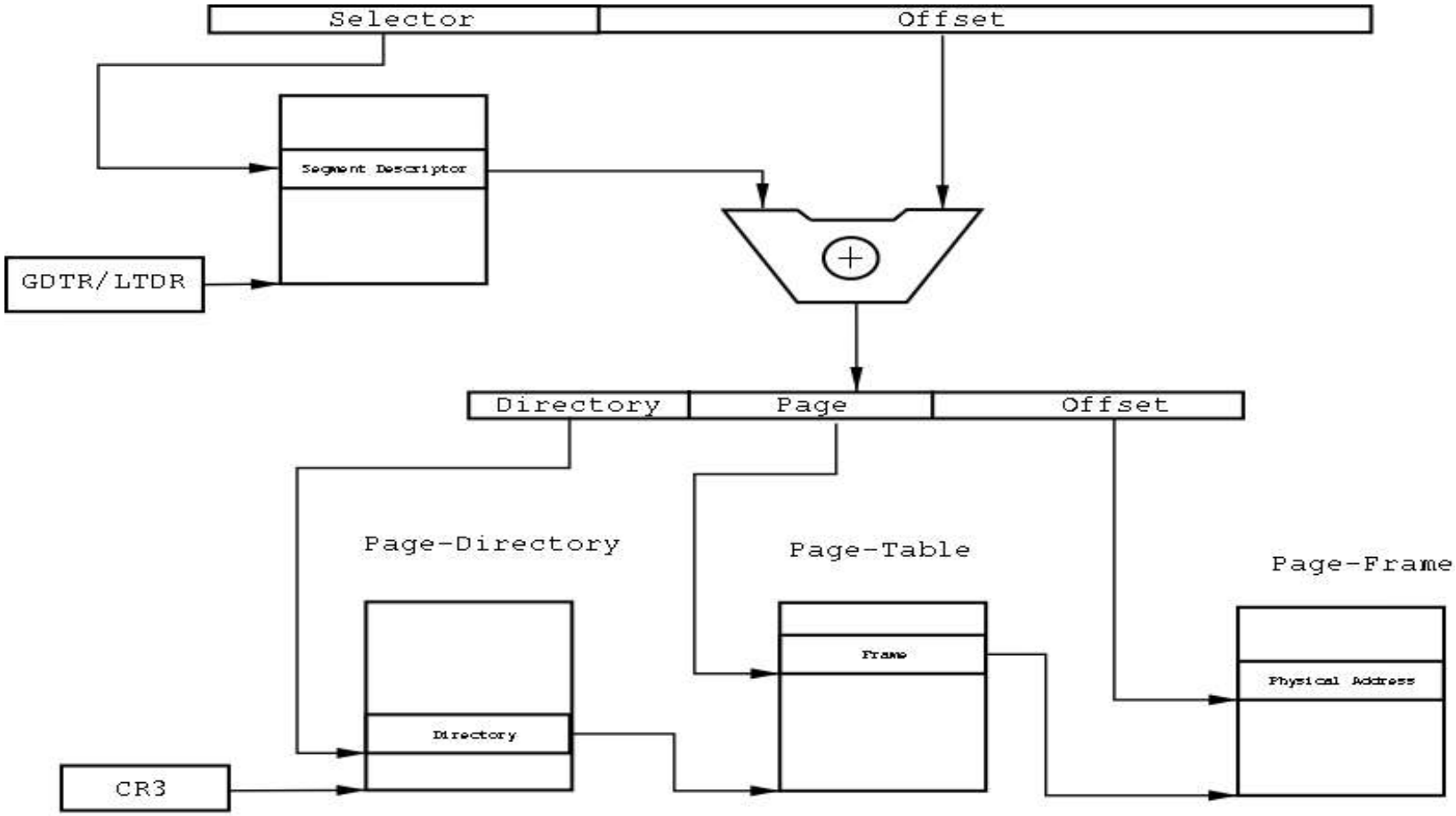
- x86 - Architecture
  - Segmentation
  - Paging
    - 2 - level
    - 4k and 4M pages
      - 2M for PAE....


Address generation

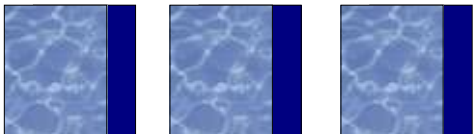


Address Generation (2)



Segment Descriptors

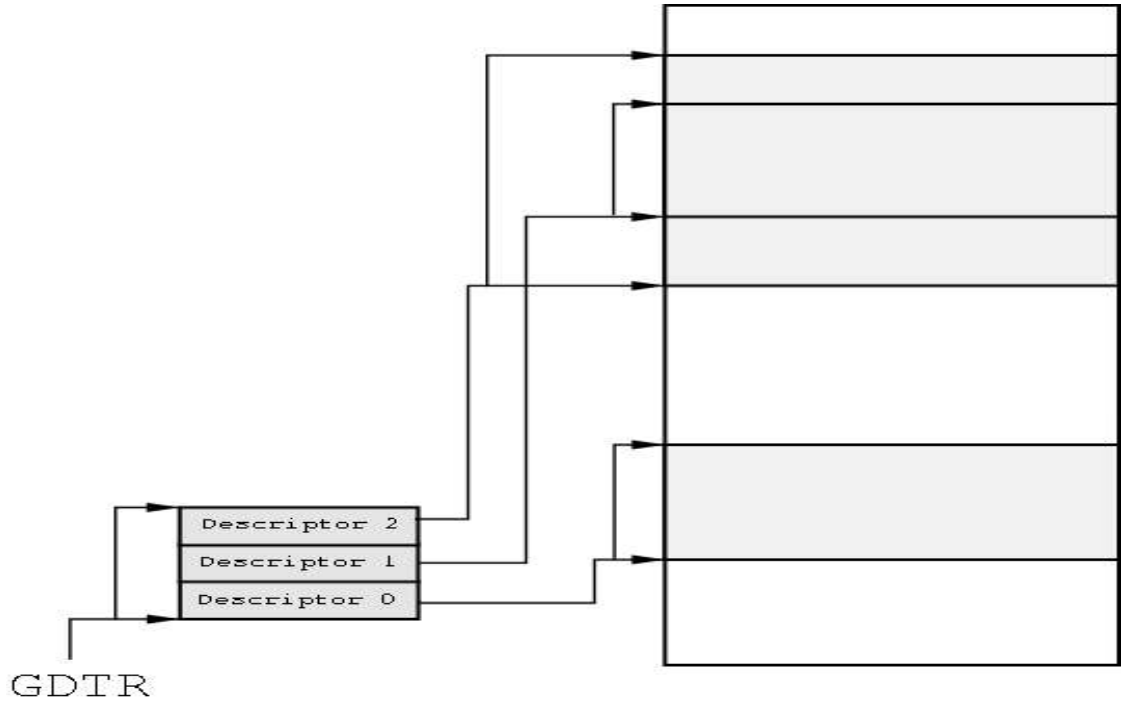
- 8-Byte representation in gdt or ldt
- Address generation
  - 32-bit Base
  - 20-bit Limit
- Protection
  - 2-bit Privilege level
- Type
  - 4-bit Type field



Global Descriptor Table

- Manipulation
  - lgdt
    - **l**oad **g**lobal **d**escriptor **t**able
  - sgdt
    - **s**tore **g**lobal **d**escriptor **t**able

```
struct pdesc{
  short pad;
  unsigned short limit;
  unsigned long linear_base;
};
struct pdesc pd;
asm volatile("sgdt %0"
    : :"m" ((pd)->limit));
```



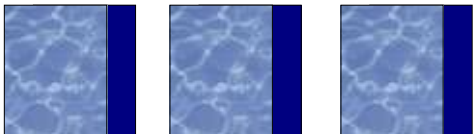
Assembly (off thread)

- AT&T syntax
  - operation src, dest (e.g. movl $0xf5, %eax)
- 7(+1) registers (eax, edx, ecx, ebx, ebp, esi ,edi, esp)
- 2-address instructions
- GCC inlining

```
asm volatile
   ("movl %%ebp, %0\n"
    "movl %%esp, %1\n"
    :"=q"(_ebp),"=q"(_esp));
```

Segment Selectors

- Part of logical address
- Hold in segment register
  - 13-bit index in gdt or ldt
  - TI Table indicator (gdt or ldt)
  - RPL requested privilege level
    - RPL of cs denotes the current privelege level
- Manipulation
  - `mov %ax, %ds`
  - `jmp $0x10, $0x800000`



Logical Address

- Logical address
  - Segment identifier (16 bit)
    - Implicit in segment registers
    - Explicit by prefix
  - Offset in segment (32 bit)
    - Part of the assembly instruction

Segment registers

- Implicitly used
- Invisible part caches descriptor
- Segment registers for fast access
  - cs — current privilege level
  - ds — data
  - ss — stack
  - es, fs, gs — extra addressing

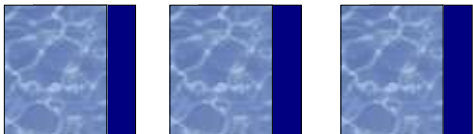
Segment Types

- Code segment
  - Executable code
- Data segment
  - Operands
  - Implicitly used
  - Prefix applicable

```
movl 0xc0080000, %eax
```

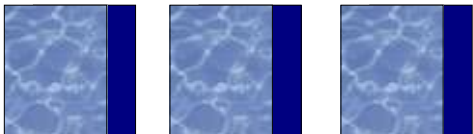
```
gs; movl 0xc0080000, %eax
```

- Task state segment
- Local descriptor table



x86 Memory Addressing

- Protected mode
  - 32-bit addressing
  - Logical address
  - Linear address
  - Physical addess (optional)
- Real mode
  - Compatibility
- Virtual Mode x86

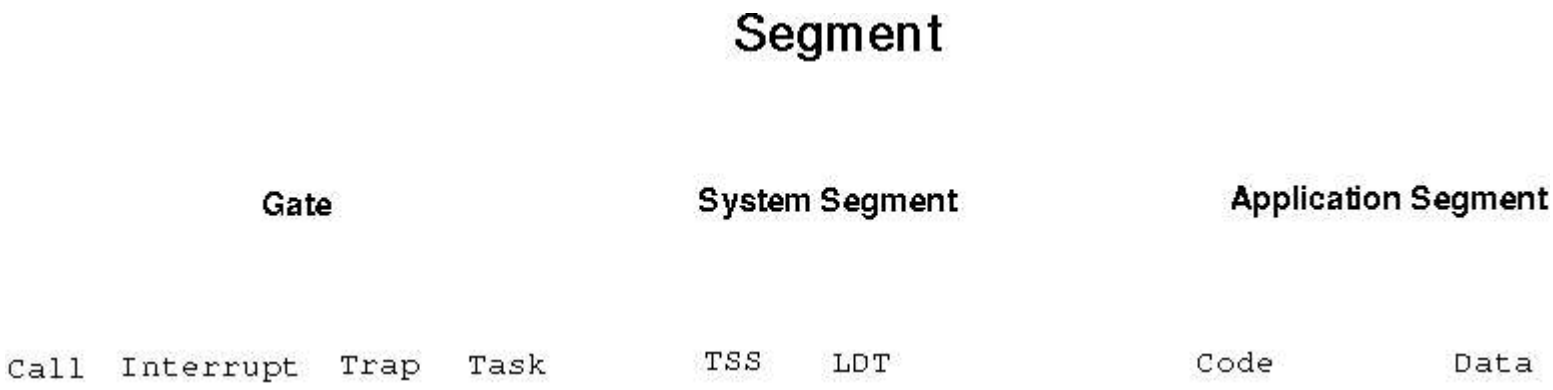


Gates

- Entry points
- Privilege supervision
- IDT interrupt descriptor table

	GDT	IDT
Call Gate	x	
Interrupt Gate		x
Trap Gate		x
Task Gate	x	x
Segments	x	

Segment Terminology (x86)



Linux Segments

	Code Segment	Data Segment
User Mode (DPL 3)	User Code	User Data
Kernel Mode (DPL 0)	Kernel Code	Kernel Data

Kernel code segment = Application segment with kernel privilege

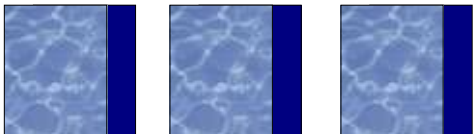
Logical abstraction of the **processor** address space.

Segments in Linux

- x86 segments
- Intel originally introduced segmentation without paging
- used in a limited way were inevitable in Linux
- Paging preferred for address space seperation
  - Easier memory management with shared linear addresses
  - Segmentation is not portable
- Don't confuse with VMA's

Linux Segments (2)

- Segments overlap totally
- Flat 32 bit address space
- Code and Data segment for kernel and user
- Shared by all processes



Linux System Segments

- TSS Task State Segment per process (2.2)
- DPL 0, no user access
- Important for user -> kernel transition

- Default LDT, shared by all processes
- One entry (null selector) - empty
- Filled when needed (e.g. windows emulation)

Segments and Processes

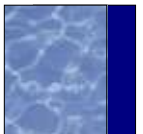
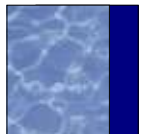
- 2 segments per process
- 4 main segment descriptors
- 4 segments for APM
- 4 left unused
- 8192 entries in GDT
- NR_TASKS <= (8192- 12) / 2
- 2.4 will overcome this limitations

Paging

- Translation from linear into physical addresses
- Privilege check
- Access type check
- Page fault exception on violation
- Enabled by bit 31 (PG) in cr0

Paging

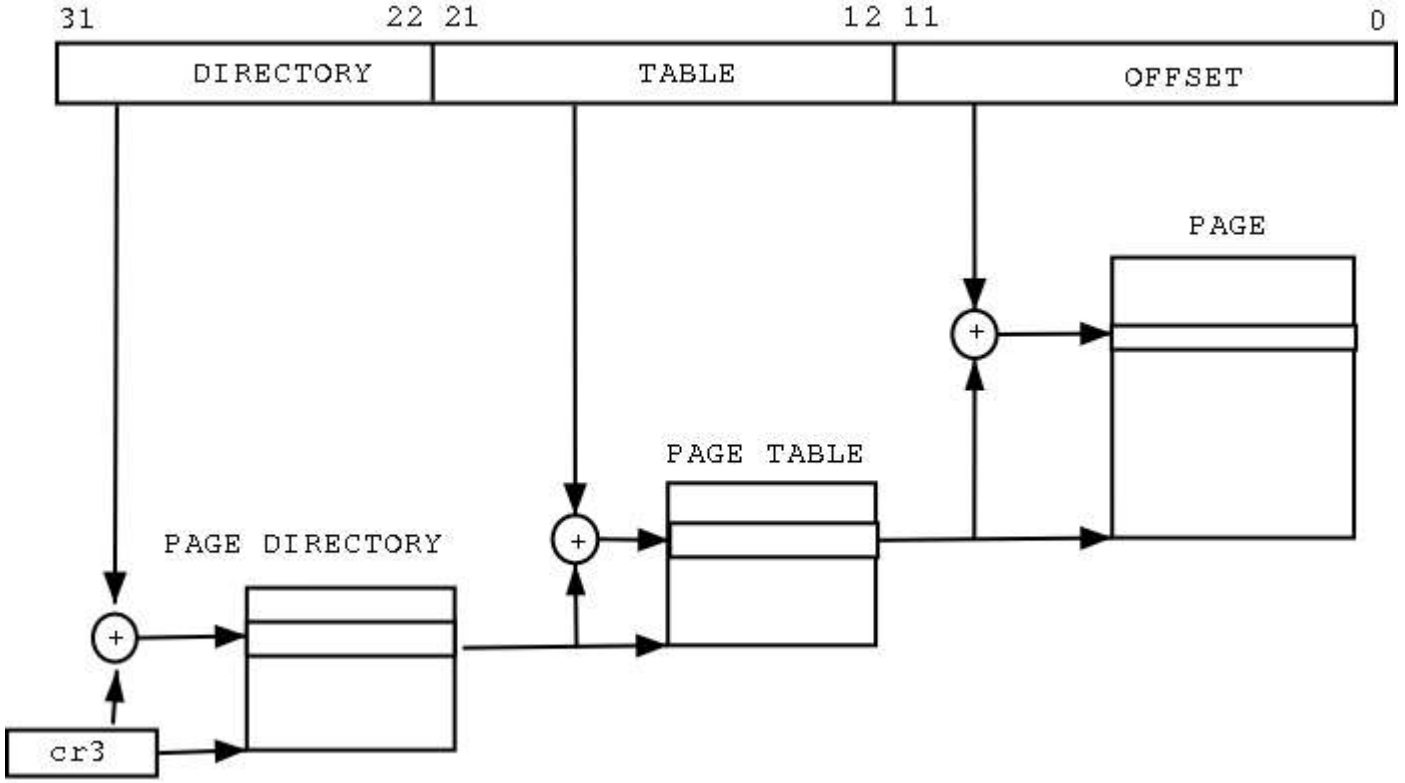
- 2 Level paging
  - 4KB page size (12 bit offset)
  - 10bit page directory
  - 10bit page table
- PG flag in cr0
- pdbr Page Directory Base register (cr3)


Extended Paging

- Extended paging
  - One Level paging
  - starting with Pentium
  - 4MB page size, 10bit page directory
- Saves TLB entries
- Enabled through PSE (page size extension) in cr4
- Page size flag in page directory entry
- Coexists with 4kb pages

Paging visualized

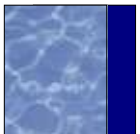


Paging in Linux

- 3 level paging modell
- feasible for 64 bit architectures
  - 43 bits used on Alpha
- Page middle directories are eliminated on x86
  - macro magic
- Each process has its own paging structures
  - address space protection
- cr3 saved in TSS during task switch

Arch. dependent paging

- include/asm-i386/page.h
  - `pte_t, pmd_t, pgd_t`
- include/asm-i386/pgtable.h
  - `pte_read, pte_write`
- mm/memory.c
  - `pte_alloc, pte_free`

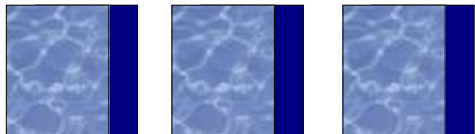


Interrupts and Exceptions

- Event that alters the sequence of instructions executed by a processor
  - Synchronous interrupts
    - system calls
    - page faults
    - privilege violations
    - Exception in Intel terminology
  - Asynchronous interrupts
    - Generated by hardware devices

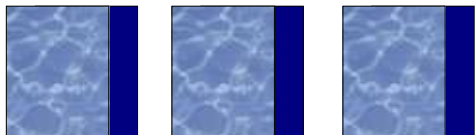
Interrupt Types

- Maskable interrupts
  - Sent to the INTR pin of the processor
  - Disabled by clearing the IF flag of the eflags register
  - cli / sti
- Nonmaskable interrupts
  - Sent to the NMI pin of the processor
  - Nothing can prevent them
  - Only for critical events



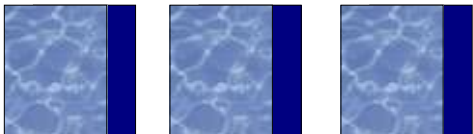
Exception Types

- Processor detected exceptions
  - Faults
    - eip (instruction pointer) of the instruction that caused the exception is saved on stack
    - e.g. page faults, general protection fault
  - Traps
    - eip of the instrunction that should be executed next to the instruction that caused the trap is saved
    - e.g. debug exceptions
  - Aborts
    - Serious error condition
    - No feasible eip available
    - e.g. double faults



Exception Types (2)

- Programmed exceptions
  - encoded in the instruction stream
  - also called software interrupts
  - handled as traps (eip of the following instr. is saved)
    - int
    - int3
    - into (check for overflow)
    - bound (check address on bound)
  - Used to implement system calls



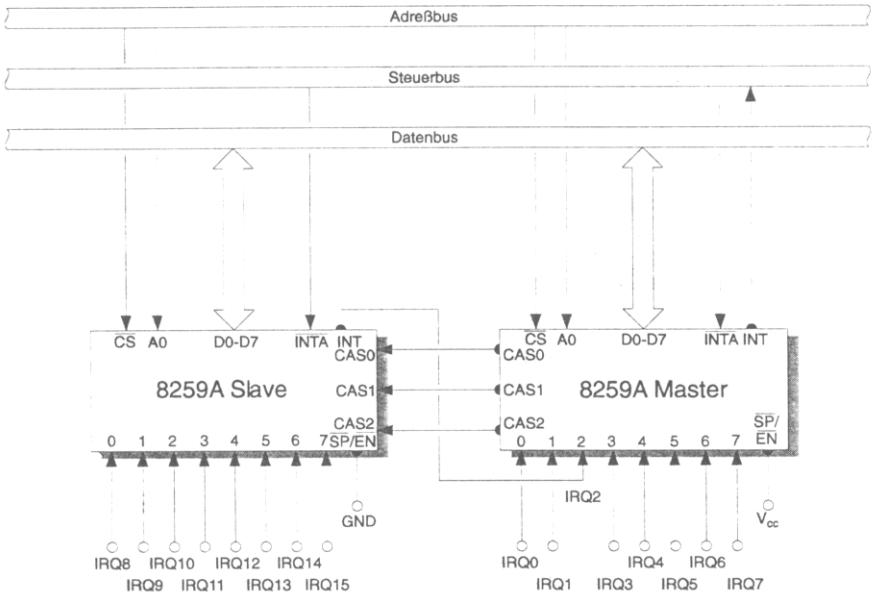
Vectors

- each interrupt or exception has a 8bit identifier
- fixed for exceptions and nmi
- Maskable interrupts can be assigned to different vectors (programming the PIC)
- 0-31 exceptions and nmi
- 32-47 maskable interrupts (IRQ) (2.2)
- 48-255 for software interrupts
  - 0x80 system call entry

IRQ

- 16 external interrupt sources in AT specification
- Programmable Interrupt Controler maps irqs to vectors and prioritizes them
- 8 input pins per PIC
- 2 PICs cascaded

PIC


Adreßbus
Steuerbus
Datenbus
CS A0 D0-D7 INTA INT
CAS0
CAS1
CAS2
8259A Slave
0 1 2 3 4 5 6 7 SP/EN
GND
IRQ8 IRQ10 IRQ12 IRQ14
IRQ9 IRQ11 IRQ13 IRQ15
CS A0 D0-D7 INTA INT
CAS0
CAS1
CAS2
8259A Master
SP/EN
0 1 2 3 4 5 6 7
IRQ2
Vcc
IRQ0 IRQ4 IRQ6
IRQ1 IRQ3 IRQ5 IRQ7

Exceptions and Signals

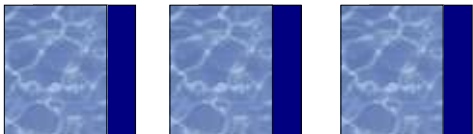
#	Exception	Exception Handler	Signal
0	„divide error	divide_error()	SIGFPE
3	Breakpoint	Int3()	SIGTRAP
6	Invalid opcode	invalid_op()	SIGILL
13	Gerneral protection fault	general_protection()	SIGSEGV
14	Page fault	page_fault()	SIGSEGV

Interrupt descriptor table

- Associates each vector with the address of the corresponding handler
- idtr register in cpu
- lidt/sidt used for manipulation
- Entry format similar to gdt and ldt

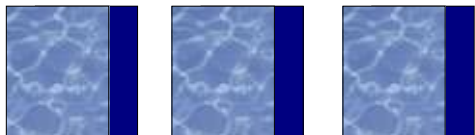
IDT entries

- Only three descriptor types possible
  - Task gate descriptor
  - Interrupt gate descriptor
  - Trap gate descriptor
- Descriptor content
  - Segment selector (tss for task gates, code otherwise)
  - offset in segment (except for task gates)
  - privilege level



Hardware Handling

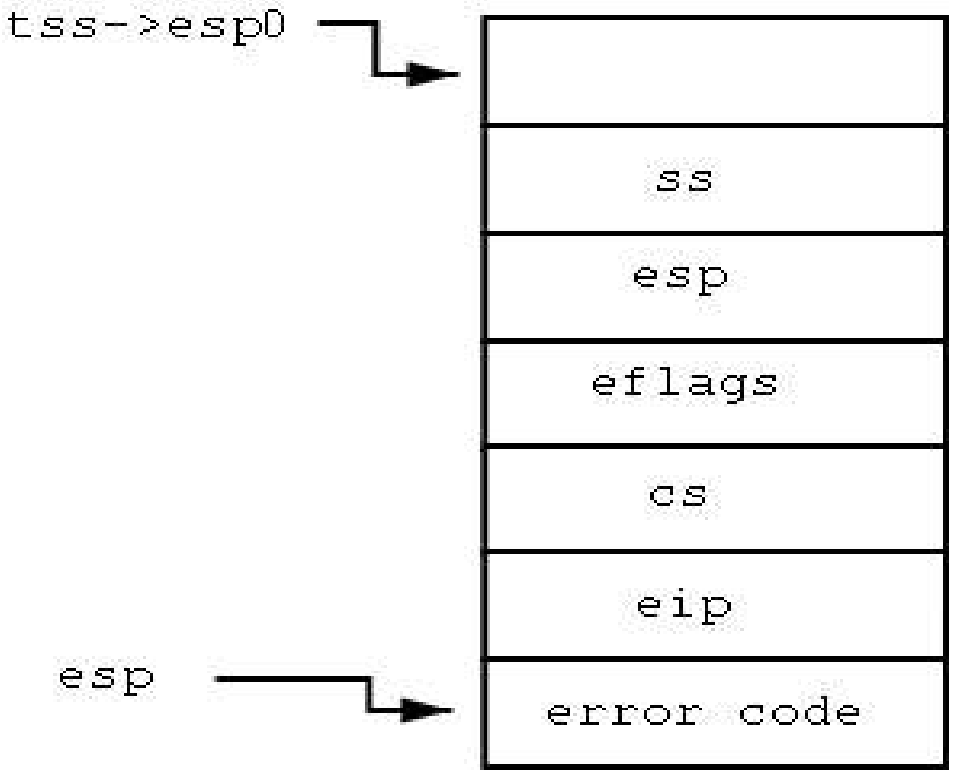
- Prior execution of each instruction, check on if an interrupt or exception has occured
- If so:
  (1) Determine the vector associated with the event
  (2) Read the ***i***th entry from the idt (designated by idtr) (we assume an interrupt or task gate)
  (3) Read the descriptor from the gdt identified by the selector of the gate
  (4) Check privilege level



Hardware Handling (2)

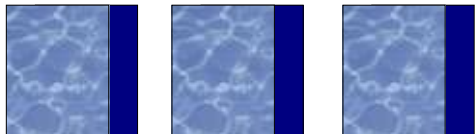
(5) Check if a change in privilege level has taken place, if so:
   (a) Read the tr register to access the TSS of the current process
   (b) Load the ss and esp registers with the proper values
   (c) Save the old values of esp and ss in the new stack
(6) If a fault has occured, adjust eip and cs to old values
(7) Save eflags, cs and eip in the stack
(8) Push error code if available
(9) Load cs and eip with the values from the gate descriptor

Exception Stack

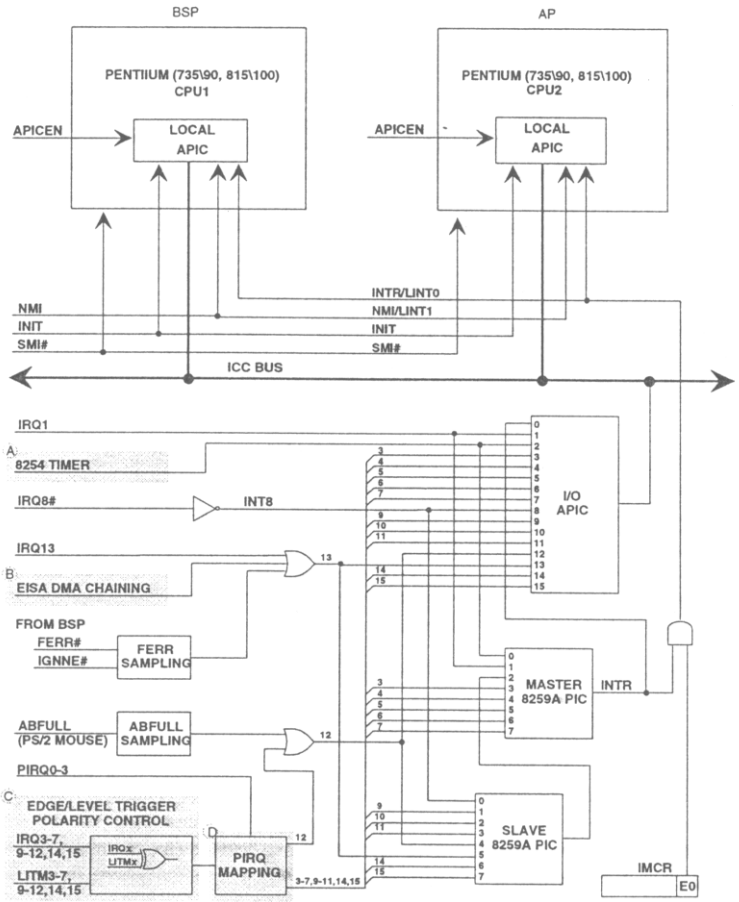


Returning from interrupts

- After interrupt processing relinquish control with iret instruction
  - (1) Load cs, eip and eflags from stack. An error code needs to be removed before.
  - (2) Check if the CPL has changed. If not, resume execution in old context.
  - (3) Load ss and esp from the stack.
  - (4) Clear ds, es, fs and gs if they hold selectors for higher privilegded segments.



Interrupt Infrastructure

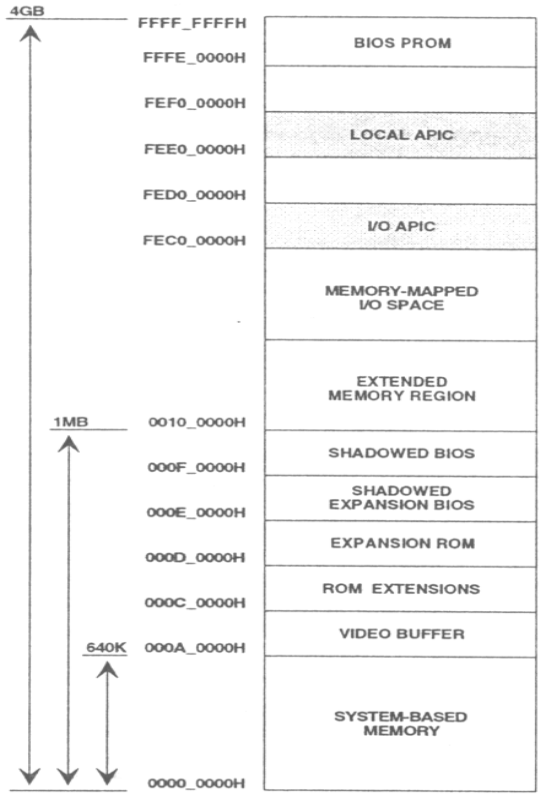


SHADED AREAS:
A,B: MAY NOT BE EXTERNALIZED WITH SOME EISA CHIPSETS
B,C: EISA BUS SPECIFIC
D: PCI BUS SPECIFIC

SMP Basics

- Shared memory
- Hardware cache synchronization
  - consistency (MESI protocol)
- Atomic operation
- Symmetry with respect to I/O-interactions
- Compatibility with uni-processor systems
- Interaction between cpus

SMP memory map

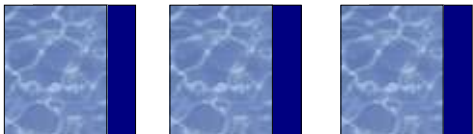


☐ PART OF THIS SPECIFICATION

UNSHADED ADDRESS REGIONS ARE FOR REFERENCE ONLY AND SHOULD NOT BE CONSTRUED AS THE SOLE DEFINITION OF A PC/AT-COMPATIBLE ADDRESS SPACE.

Local APIC

- Memory mapped control registers
- Processing local interrupts
- Interaction via the APIC - bus
- High precision timer



APIC schematics