CMSC 313 Lecture 15

- Project 4 Questions
- Reminder: Midterm Exam on Tuesday 10/26
- Interrupts
- Cache Memory [time permitting]

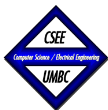
Motivating Example

```
; An Assembly language program for printing data
        MOV EDX, 378H           ;Printer Data Port
        MOV ECX, 0              ;Use ECX as the loop counter
XYZ:    MOV AL, [ABC + ECX]     ;ABC is the beginning of the memory area
                                ; that characters are being printed from
        OUT [DX], AL            ;Send a character to the printer
        INC ECX
        CMP ECX, 100000         ; print this many characters
        JL XYZ
```

Issues:

- What about difference in speed between the processor and printer?
- What about the buffer size of the printer?
  - Small buffer can lead to some lost data that will not get printed

Communication with input/output devices needs handshaking protocols



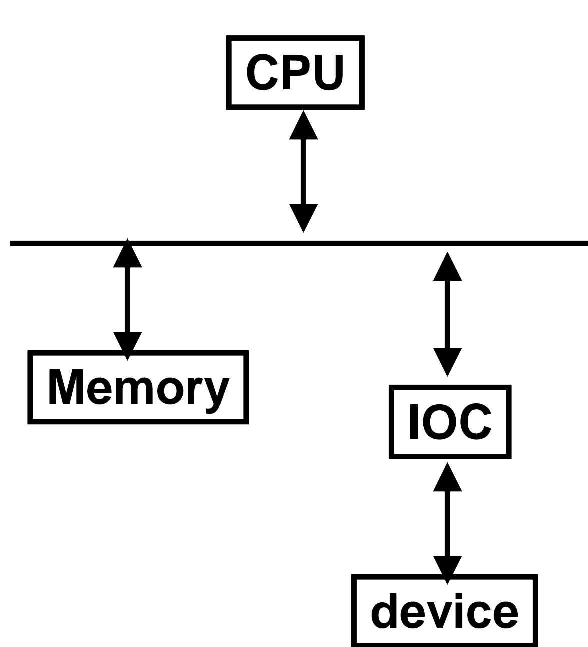
Communicating with I/O Devices

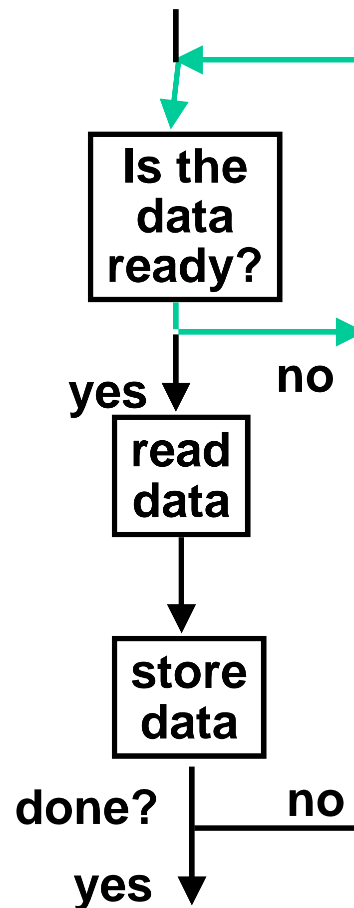
- The OS needs to know when:
  - The I/O device has completed an operation
  - The I/O operation has encountered an error
- This can be accomplished in two different ways:
  - Polling:
    - The I/O device put information in a status register
    - The OS periodically check the status register
  - I/O Interrupt:
    - An I/O interrupt is an externally stimulated event, asynchronous to instruction execution but does NOT prevent instruction completion
    - Whenever an I/O device needs attention from the processor, it interrupts the processor from what it is currently doing
    - Some processors deal with interrupts as special exceptions

These schemes requires heavy processor's involvement and suitable only for low bandwidth devices such as the keyboard

* Slide is partially a courtesy of Dave Patterson

Polling: Programmed I/O





busy wait loop not an efficient way to use the CPU unless the device is very fast!

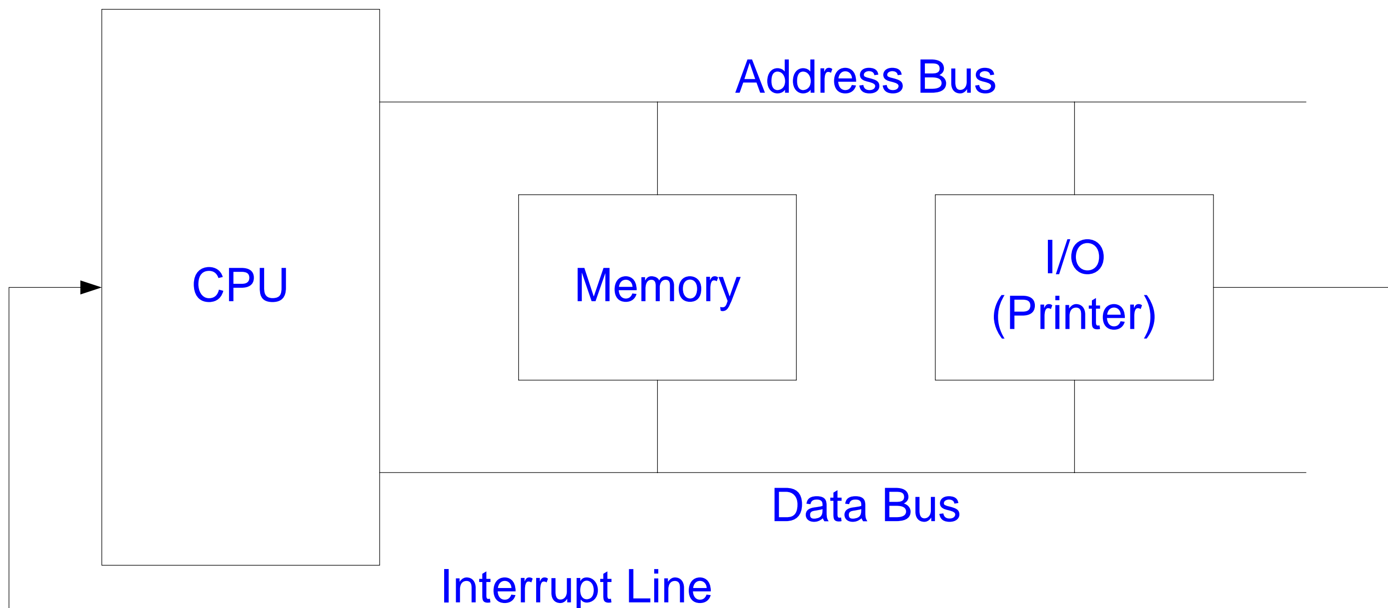
but checks for I/O completion can be dispersed among computation intensive code

- Advantage:
  - Simple: the processor is totally in control and does all the work
- Disadvantage:
  - Polling overhead can consume a lot of CPU time

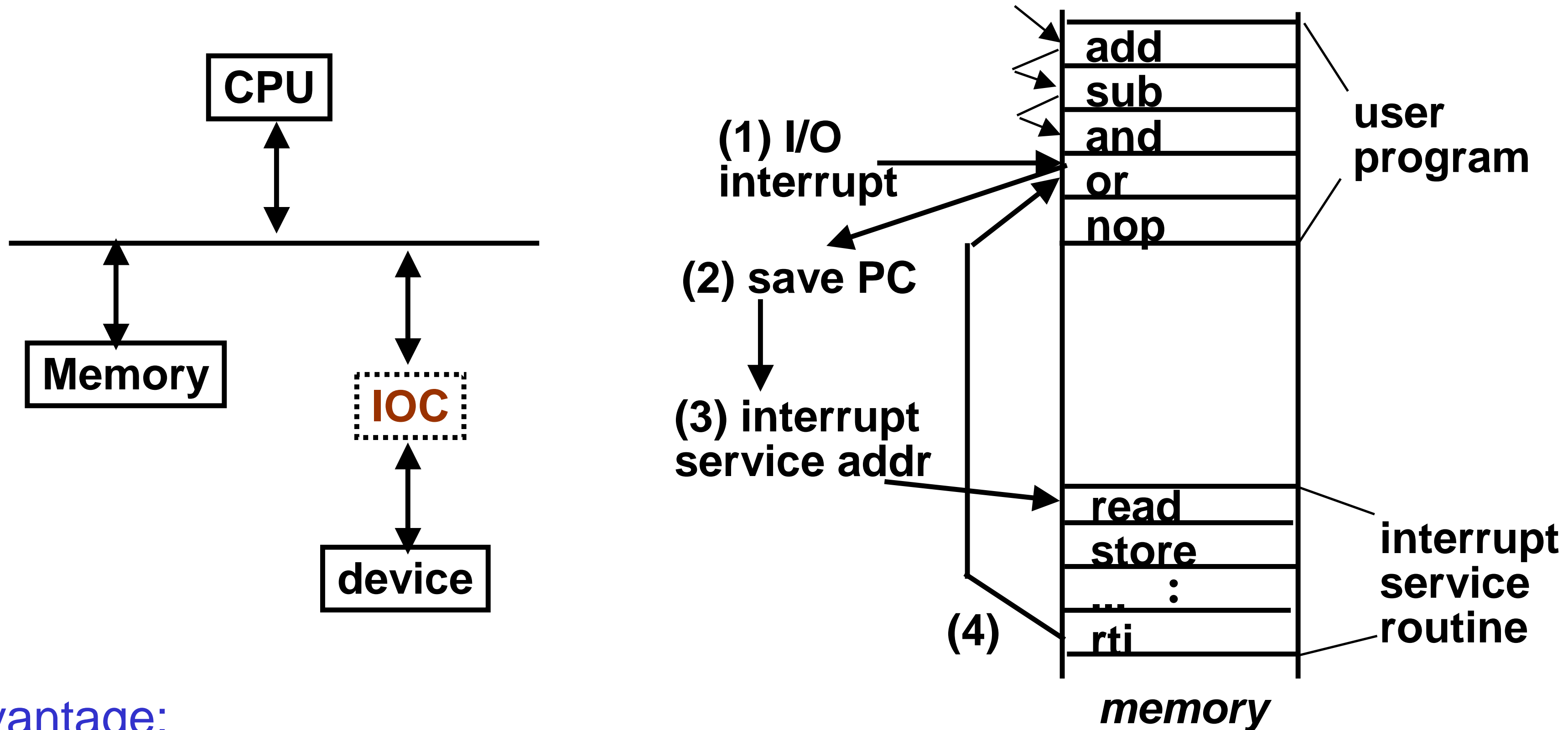
* Slide is courtesy of Dave Patterson

External Interrupt

- The fetch-execute cycle is a program-driven model of computation
- Computers are not totally program driven as they are also hardware driven
- An I/O interrupt is an externally stimulated event, asynchronous to instruction execution but does NOT prevent instruction completion
- Whenever an I/O device needs attention from the processor, it interrupts the processor from what it is currently doing
- Processors typically have one or multiple interrupt pins for device interface



Interrupt Driven Data Transfer

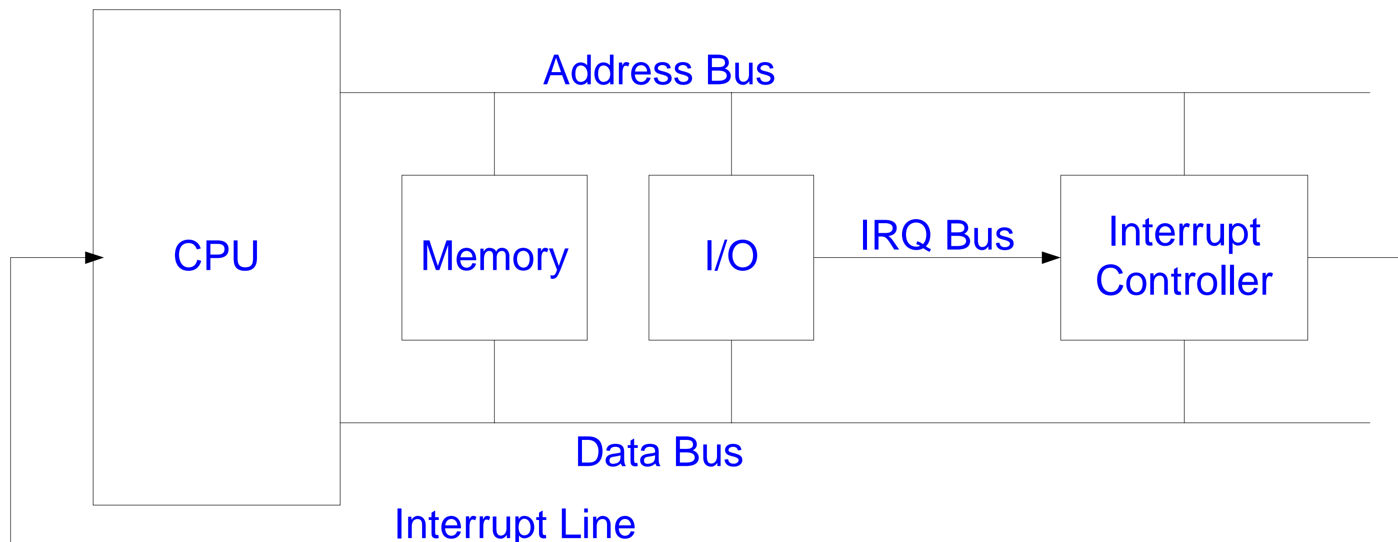


- Advantage:
  - User program progress is only halted during actual transfer
- Disadvantage: special hardware is needed to:
  - Cause an interrupt (I/O device)
  - Detect an interrupt (processor)
  - Save the proper states to resume after the interrupt (processor)

* Slide is courtesy of Dave Patterson

80386 Interrupt Handling

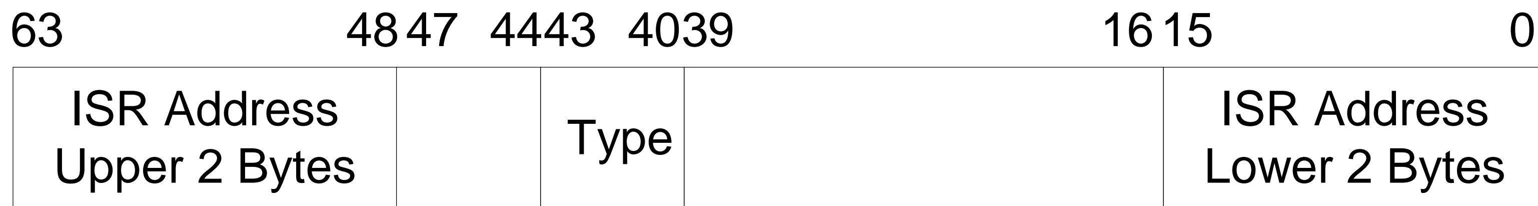
- The 80386 has only one interrupt pin and relies on an interrupt controller to interface and prioritize the different I/O devices
- Interrupt handling follows the following steps:
  - ❶ Complete current instruction
  - ❷ Save current program counter and flags into the stack
  - ❸ Get interrupt number responsible for the signal from interrupt controller
  - ❹ Find the address of the appropriate interrupt service routine
  - ❺ Transfer control to interrupt service routine
- A special interrupt acknowledge bus cycle is used to read interrupt number
- Interrupt controller has ports that are accessible through IN and OUT

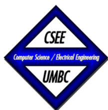


Interrupt Descriptor Table

Address	
b	Gate #0
b + 8	Gate #1
b + 16	Gate #2
b + 24	Gate #3
b + 32	Gate #4
b + 40	Gate #5
	. . .
b + 2040	Gate #255

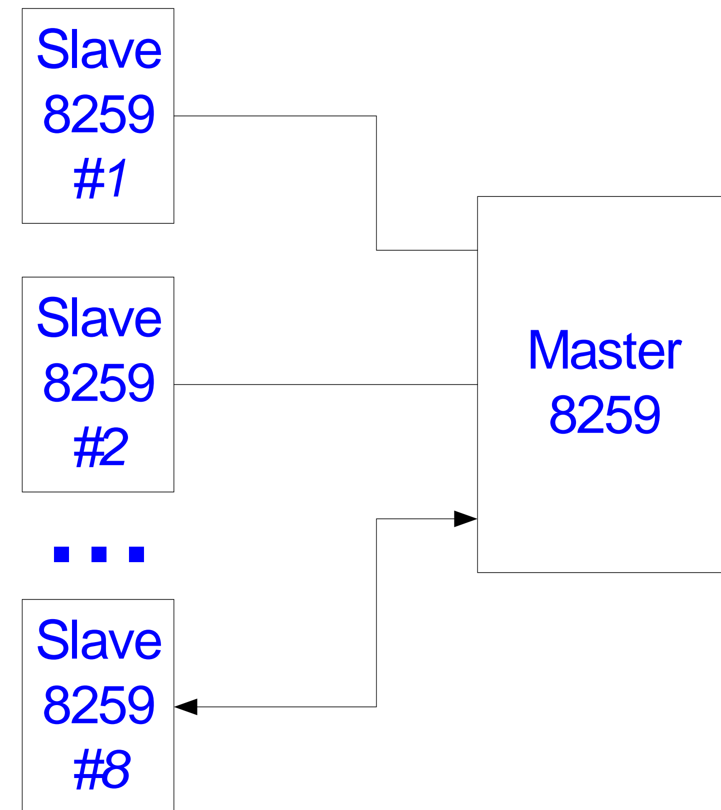
- The address of an ISR is fetched from an interrupt descriptor table
- IDT register is loaded by operating system and points to the interrupt descriptor table
- Each entry is 8 bytes indicating address of ISR and type of interrupt (trap, fault etc.)
- RESET and non-maskable (NMI) interrupts use distinct processor pins
- NMI is used to for parity error or power supply problems and thus cannot be disabled





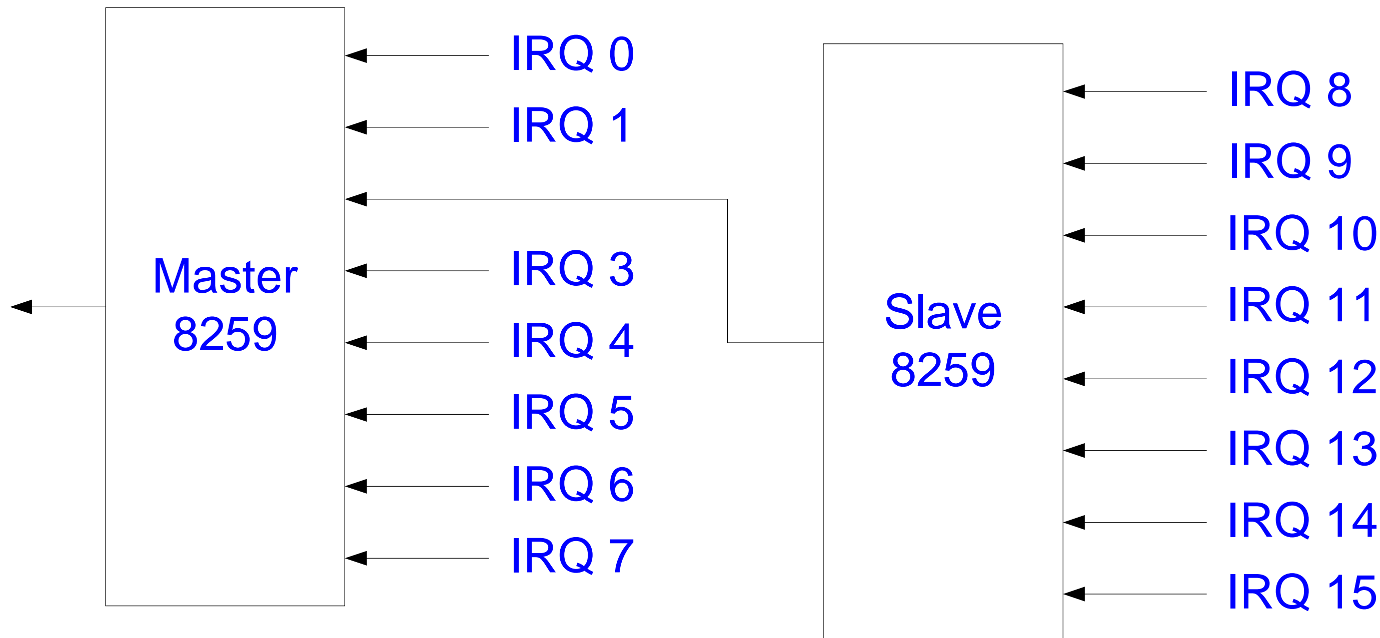
The 8259 Interrupt Controller

- Since the 80386 has one interrupt pin, an interrupt controller is needed to handle multiple input and output devices
- The Intel 8259 is a programmable interrupt controller that can be used either singly or in a two-tier configuration
- When used as a master, the 8259 can interface with up to 8 slaves
- Since the 8259 controller can be a master or a slave, the interrupt request lines must be programmable
- Programming the 8259 chips takes place at boot time using the OUT commands
- The order of the interrupt lines reflects the priority assigned to them



The ISA Architecture

- The ISA architecture is set by IBM competitors and standardizes:
  - The interrupt controller circuitry
  - Many IRQ assignments
  - Many I/O port assignments
  - The signals and connections made available to expansion cards
- A one-master-one-slave configuration is the norm for ISA architecture



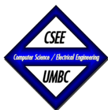
- Priority is assigned in the following order:

  IRQ 0, IRQ 1, IRQ 8, …, IRQ 15, IRQ 3, …, IRQ 7

ISA Interrupt Routings

IRQ	ALLOCATION	INTRRUPT NUMBER
IRQ0	System Timer	08H
IRQ1	Keyboard	09H
IRQ3	Serial Port #2	OBH
IRQ4	Serial Port # 1	OCH
IRQ5	Parallel Port #2	ODH
IRQ6	Floppy Controller	OEH
IRQ7	Parallel Port # 1	OFH
IRQ8	Real time clock	70H
IRQ9	available	71 H
IRQ10	available	72H
IRQ11	available	73H
IRQ12	Mouse	74H
IRQ13	87 ERROR line	75H
IRQ14	Hard drive controller	76H
IRQ15	available	77H

linux1% cat /proc/interrupts



I/O Interrupt vs. Exception

- An I/O interrupt is just like the exceptions except:
  - An I/O interrupt is asynchronous
  - Further information needs to be conveyed
  - Typically exceptions are more urgent than interrupts
- An I/O interrupt is asynchronous with respect to instruction execution:
  - I/O interrupt is not associated with any instruction
  - I/O interrupt does not prevent any instruction from completion
    - You can pick your own convenient point to take an interrupt
- I/O interrupt is more complicated than exception:
  - Needs to convey the identity of the device generating the interrupt
  - Interrupt requests can have different urgencies:
    - Interrupt request needs to be prioritized
    - Priority indicates urgency of dealing with the interrupt
    - High speed devices usually receive highest priority

* Slide is courtesy of Dave Patterson

Internal and Software Interrupt

- **Exceptions:**
  - Exceptions do not use the interrupt acknowledge bus cycle but are still handled by a numbered ISR
  - Examples: divide by zero, unknown instruction code, access violation, …
- **Software Interrupts:**
  - The INT instruction makes interrupt service routines accessible to programmers
  - Syntax: “INT imm” with *imm* indicating interrupt number
  - Returning from an ISR is like RET, except it enables interrupts

	Ordinary subroutine	Interrupt service routine
Invoke	CALL	INT
Terminate	RET	IRET

- **Fault and Traps:**
  - When an instruction causes an exception and is retried after handling it, the exception is called a fault (e.g. page fault)
  - When control is passed to the next instruction after handling an exception or interrupt, such exception is called a trap (e.g. division overflow)

Built-in Hardware Exceptions

Allocation	Int #
Division Overflow	00H
Single Step	01H
NMI	02H
Breakpoint	03H
Interrupt on Overflow	04H
BOUND out of range	05H
Invalid Machine Code	06H
87 not available	07H
Double Fault	08H
87 Segment Overrun	09H
Invalid Task State Segment	0AH
Segment Not Present	0BH
Stack Overflow	0CH
General Protection Error	0DH
Page Fault	0EH
(reserved)	0FH
87 Error	10H

Interrupt 13—General Protection Exception (#GP)

Exception Class Fault.

Description

Indicates that the processor detected one of a class of protection violations called "general-protection violations." The conditions that cause this exception to be generated comprise all the protection violations that do not cause other exceptions to be generated (such as, invalid-TSS, segment-not-present, stack-fault, or page-fault exceptions). The following conditions cause general-protection exceptions to be generated:

- Exceeding the segment limit when accessing the CS, DS, ES, FS, or GS segments.
- Exceeding the segment limit when referencing a descriptor table (except during a task switch or a stack switch).
- Transferring execution to a segment that is not executable.
- Writing to a code segment or a read-only data segment.
- Reading from an execute-only code segment.
- Loading the SS register with a segment selector for a read-only segment (unless the selector comes from a TSS during a task switch, in which case an invalid-TSS exception occurs).
- Loading the SS, DS, ES, FS, or GS register with a segment selector for a system segment.
- Loading the DS, ES, FS, or GS register with a segment selector for an execute-only code segment.
- Loading the SS register with the segment selector of an executable segment or a null segment selector.
- Loading the CS register with a segment selector for a data segment or a null segment selector.
- Accessing memory using the DS, ES, FS, or GS register when it contains a null segment selector.
- Switching to a busy task during a call or jump to a TSS.
- Using a segment selector on a non-IRET task switch that points to a TSS descriptor in the current LDT. TSS descriptors can only reside in the GDT. This condition causes a #TS exception during an IRET task switch.
- Violating any of the privilege rules described in Chapter 4, *Protection*.
- Exceeding the instruction length limit of 15 bytes (this only can occur when redundant prefixes are placed before an instruction).
- Loading the CR0 register with a set PG flag (paging enabled) and a clear PE flag (protection disabled).

Interrupt 14—Page-Fault Exception (#PF)

Exception Class Fault.

Description

Indicates that, with paging enabled (the PG flag in the CR0 register is set), the processor detected one of the following conditions while using the page-translation mechanism to translate a linear address to a physical address:

- The P (present) flag in a page-directory or page-table entry needed for the address translation is clear, indicating that a page table or the page containing the operand is not present in physical memory.
- The procedure does not have sufficient privilege to access the indicated page (that is, a procedure running in user mode attempts to access a supervisor-mode page).
- Code running in user mode attempts to write to a read-only page. In the Intel486 and later processors, if the WP flag is set in CR0, the page fault will also be triggered by code running in supervisor mode that tries to write to a read-only user-mode page.
- One or more reserved bits in page directory entry are set to 1. See description below of RSVD error code flag

The exception handler can recover from page-not-present conditions and restart the program or task without any loss of program continuity. It can also restart the program or task after a privilege violation, but the problem that caused the privilege violation may be uncorrectable.

Exception Error Code

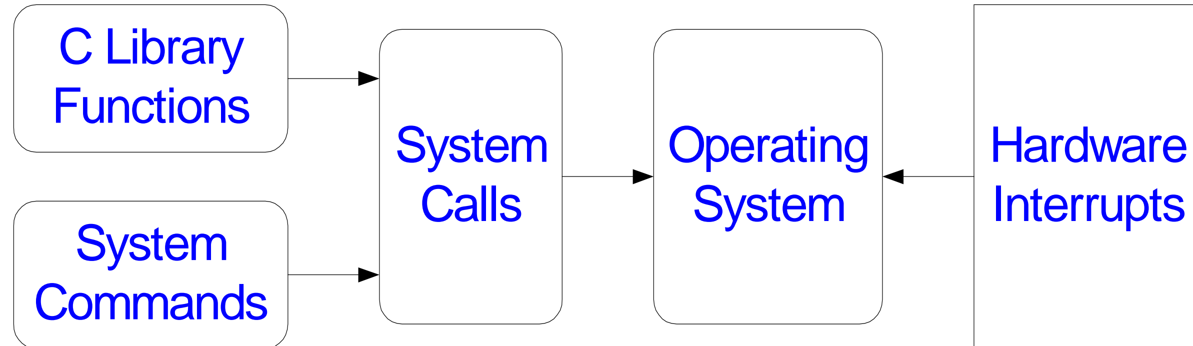
Yes (special format). The processor provides the page-fault handler with two items of information to aid in diagnosing the exception and recovering from it:

- An error code on the stack. The error code for a page fault has a format different from that for other exceptions (see Figure 5-7). The error code tells the exception handler four things:
  - — The P flag indicates whether the exception was due to a not-present page (0) or to either an access rights violation or the use of a reserved bit (1).
  - — The W/R flag indicates whether the memory access that caused the exception was a read (0) or write (1).
  - — The U/S flag indicates whether the processor was executing at user mode (1) or supervisor mode (0) at the time of the exception.
  - — 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. (The PSE flag is only available in the Pentium 4, Intel Xeon, P6 family, and Pentium processors, and the PAE flag is only available on the Pentium 4, Intel Xeon, and P6 family processors. In earlier IA-32 processor, the bit position of the RSVD flag is reserved.)

System Calls

- Linux conventions: parameters are stored left to right order in registers EBX, ECX, EDX, EDI and ESI respectively

```
main() {
   char s[] = "Hello world!\n";
   write(1,s,13);
}
```




```
;This program makes a system call
;
            global main
main:       MOV EAX, 4          ;Write is system call #4
            MOV EBX, 1          ;1 is number for standard output
            MOV ECX, ABC        ;ABC is the string pointer
            MOV EDX, 13         ;Write 13 bytes
            INT 80H             ;System call interrupt
            RET
 ABC: db "Hello world!", 0AH,0
```

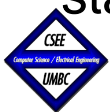
Privileged Mode

Privilege Levels

- The difference between kernel mode and user mode is in the privilege level
- The 80386 has 4 privilege levels, two of them are used in Linux
  - Level 0: system level (Linux kernel)
  - Level 3: user level (user processes)
- The CPL register stores the current privilege level and is reset during the execution of system calls
- Privileged instructions, such as LIDT that set interrupt tables can execute only when CPL = 0

Stack Issues

- System calls have to use different stack since the user processes will have write access to them (imagine a process passing the stack pointer as a parameter forcing the system call to overwrite its own stack
- There is a different stack pointer for every privilege level stored in the task state segment



Summary: Types of Interrupts

- **Hardware vs Software**
  - Hardware: I/O, clock tick, power failure, exceptions
  - Software: INT instruction
- **External vs Internal Hardware Interrupts**
  - External interrupts are generated by CPU's interrupt pin
  - Internal interrupts (exceptions): div by zero, single step, page fault, bad opcode, stack overflow, protection, ...
- **Synchronous vs Asynchronous Hardware Int.**
  - Synchronous interrupts occur at exactly the same place every time the program is executed. E.g., bad opcode, div by zero, illegal memory address.
  - Asynchronous interrupts occur at unpredictable times relative to the program. E.g., I/O, clock ticks.

Summary: Interrupt Sequence

- Device sends signal to interrupt controller.
- Controller uses IRQ# for interrupt # and priority.
- Controller sends signal to CPU if the CPU is not already processing an interrupt with higher priority.
- CPU finishes executing the current instruction
- CPU saves EFLAGS & return address on the stack.
- CPU gets interrupt # from controller using I/O ops.
- CPU finds “gate” in Interrupt Description Table.
- CPU switches to Interrupt Service Routine (ISR). This may include a change in privilege level. IF cleared.

Interrupt Sequence (cont.)

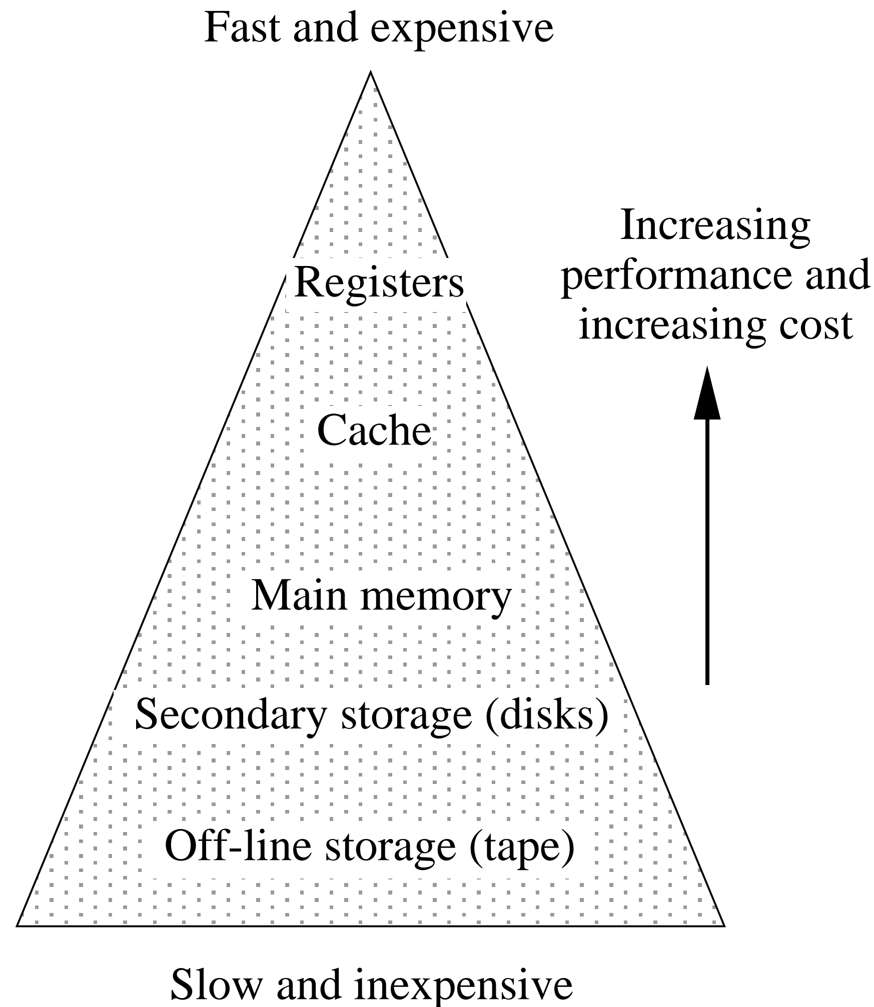
- ISR saves registers if necessary.
- ISR, after initial processing, sets IF to allow interrupts.
- ISR processes the interrupt.
- ISR restores registers if necessary.
- ISR sends End of Interrupt (EOI) to controller.
- ISR returns from interrupt using IRET. EFLAGS (inlcuding IF) & return address restored.
- CPU executes the next instruction.
- Interrupt controller waits for next interrupt and manages pending interrupts.

CACHE MEMORY

Recap Virtual Memory

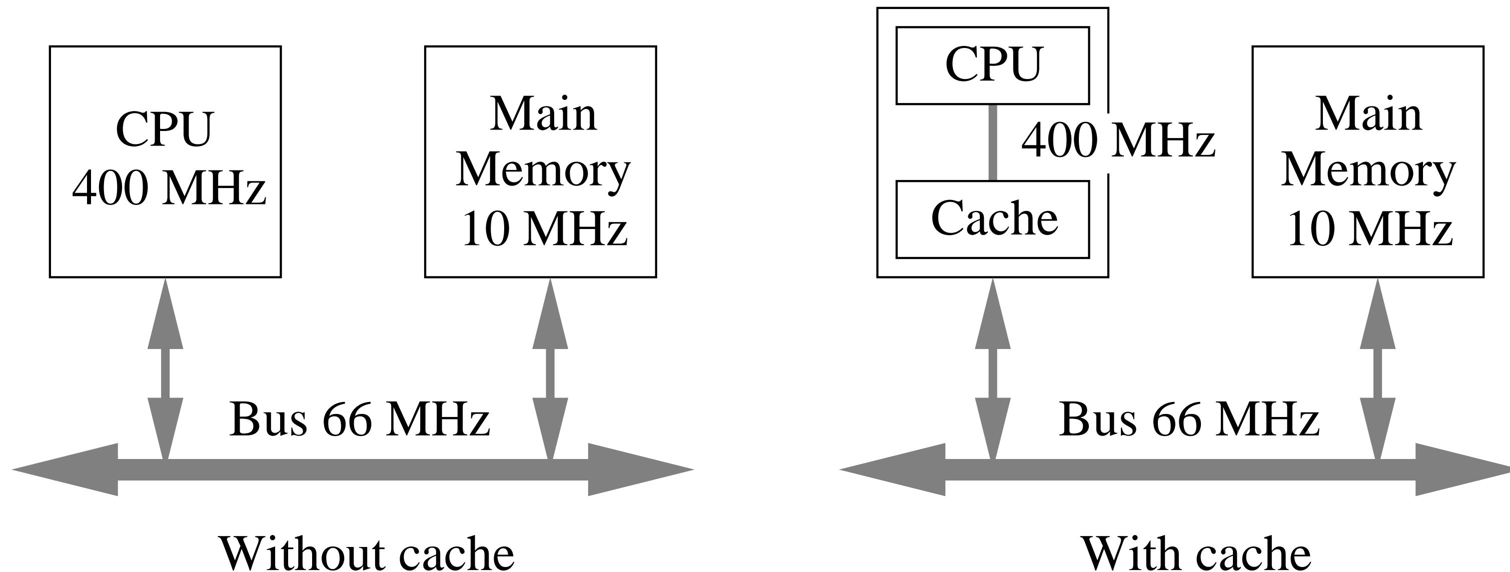
- **Not enough physical memory**
  - Uses disk space to simulate extra memory
  - Pages not being used can be swapped out
    (how and when you'll learn in CMSC 421 Operating Systems)
  - Thrashing: pages constantly written to and retrieved from disk
    (time to buy more RAM)
- **Fragmentation**
  - Contiguous blocks of virtual memory do not have to map to contiguous sections of real memory
- **Memory protection**
  - Each process has its own page table
  - Shared pages are read-only
  - User processes cannot alter the page table (must be supervisor)

The Memory Hierarchy



© 1999 M. Murdocca and V. Heuring

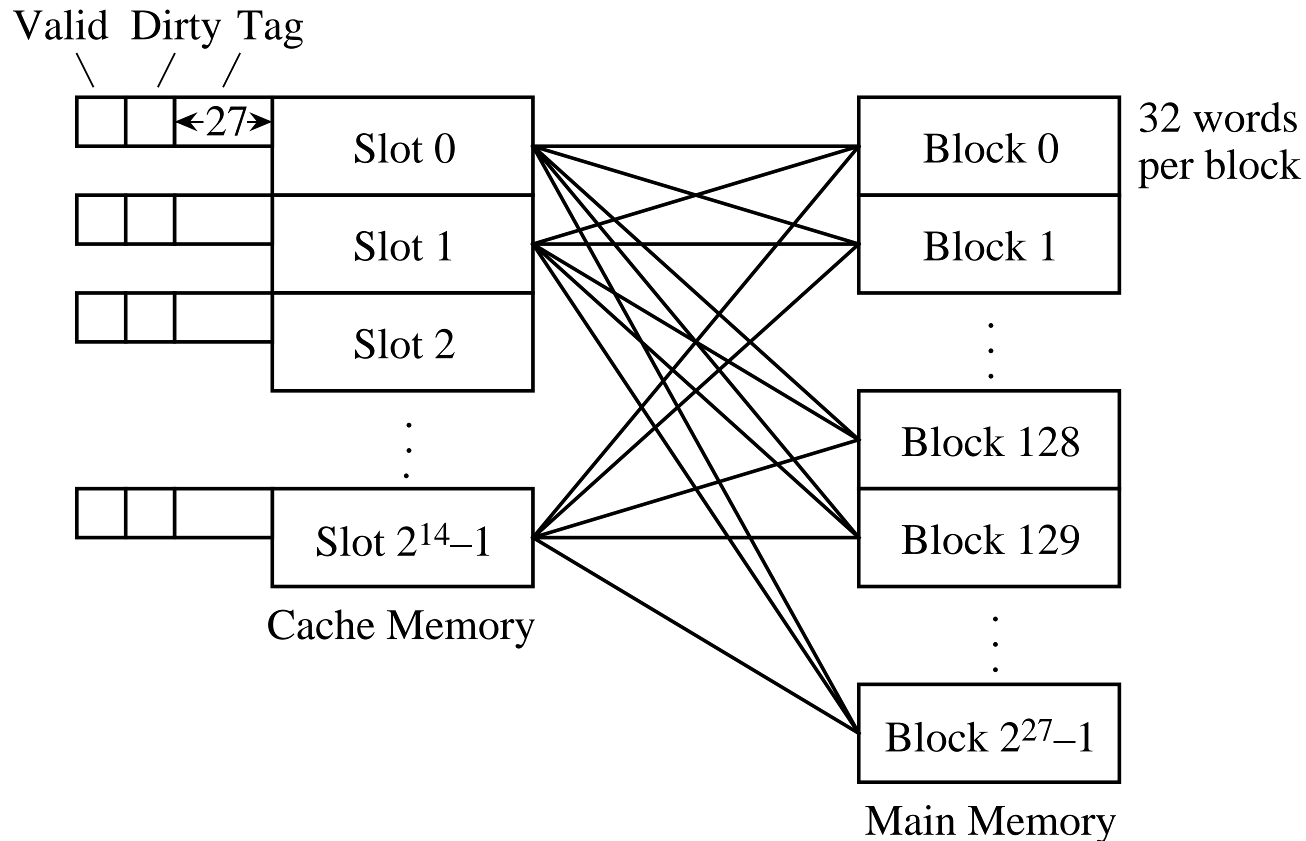
Placement of Cache in a Computer System



- **The *locality principle*: a recently referenced memory location is likely to be referenced again (*temporal locality*); a neighbor of a recently referenced memory location is likely to be referenced (*spatial locality*).**

© 1999 M. Murdocca and V. Heuring

An Associative Mapping Scheme for a Cache Memory



© 1999 M. Murdocca and V. Heuring

Associative Mapping Example

- Consider how an access to memory location $(A035F014)_{16}$ is mapped to the cache for a 2^{32} word memory. The memory is divided into 2^{27} blocks of $2^5 = 32$ words per block, and the cache consists of 2^{14} slots:

Tag	Word
27 bits	5 bits

- If the addressed word is in the cache, it will be found in word $(14)_{16}$ of a slot that has tag $(501AF80)_{16}$, which is made up of the 27 most significant bits of the address. If the addressed word is not in the cache, then the block corresponding to tag field $(501AF80)_{16}$ is brought into an available slot in the cache from the main memory, and the memory reference is then satisfied from the cache.

Tag	Word
1 0 1 0 0 0 0 0 0 0 1 1 0 1 0 1 1 1 1 1 0 0 0 0 0 0 0	1 0 1 0 0

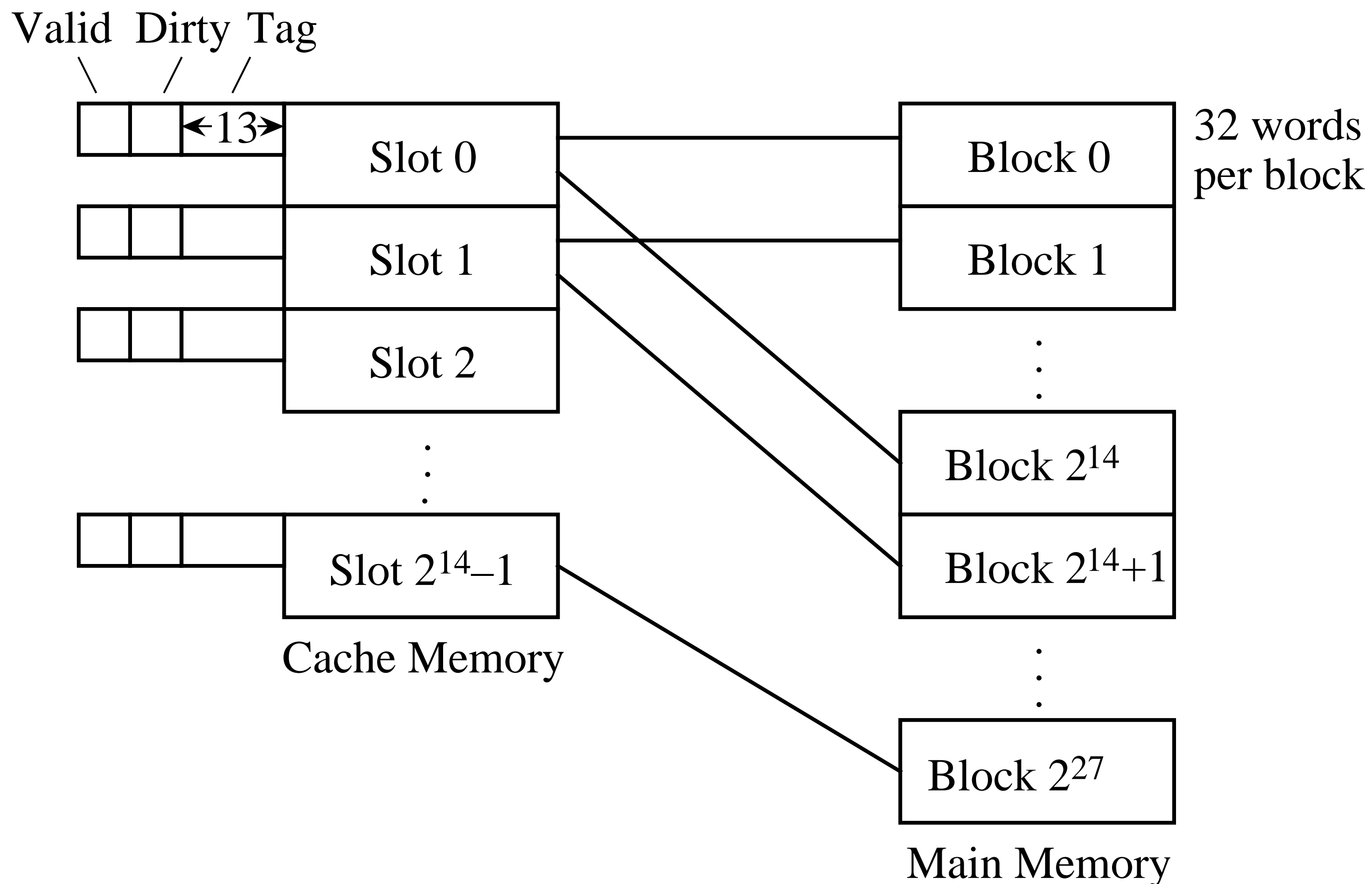
© 1999 M. Murdocca and V. Heuring

Replacement Policies

- When there are no available slots in which to place a block, a *replacement policy* is implemented. The replacement policy governs the choice of which slot is freed up for the new block.
- Replacement policies are used for associative and set-associative mapping schemes, and also for virtual memory.
- Least recently used (LRU)
- First-in/first-out (FIFO)
- Least frequently used (LFU)
- Random
- Optimal (used for analysis only – look backward in time and reverse-engineer the best possible strategy for a particular sequence of memory references.)

© 1999 M. Murdocca and V. Heuring

A Direct Mapping Scheme for Cache Memory



© 1999 M. Murdocca and V. Heuring

Direct Mapping Example

- For a direct mapped cache, each main memory block can be mapped to only one slot, but each slot can receive more than one block. Consider how an access to memory location $(A035F014)_{16}$ is mapped to the cache for a 2^{32} word memory. The memory is divided into 2^{27} blocks of $2^5 = 32$ words per block, and the cache consists of 2^{14} slots:

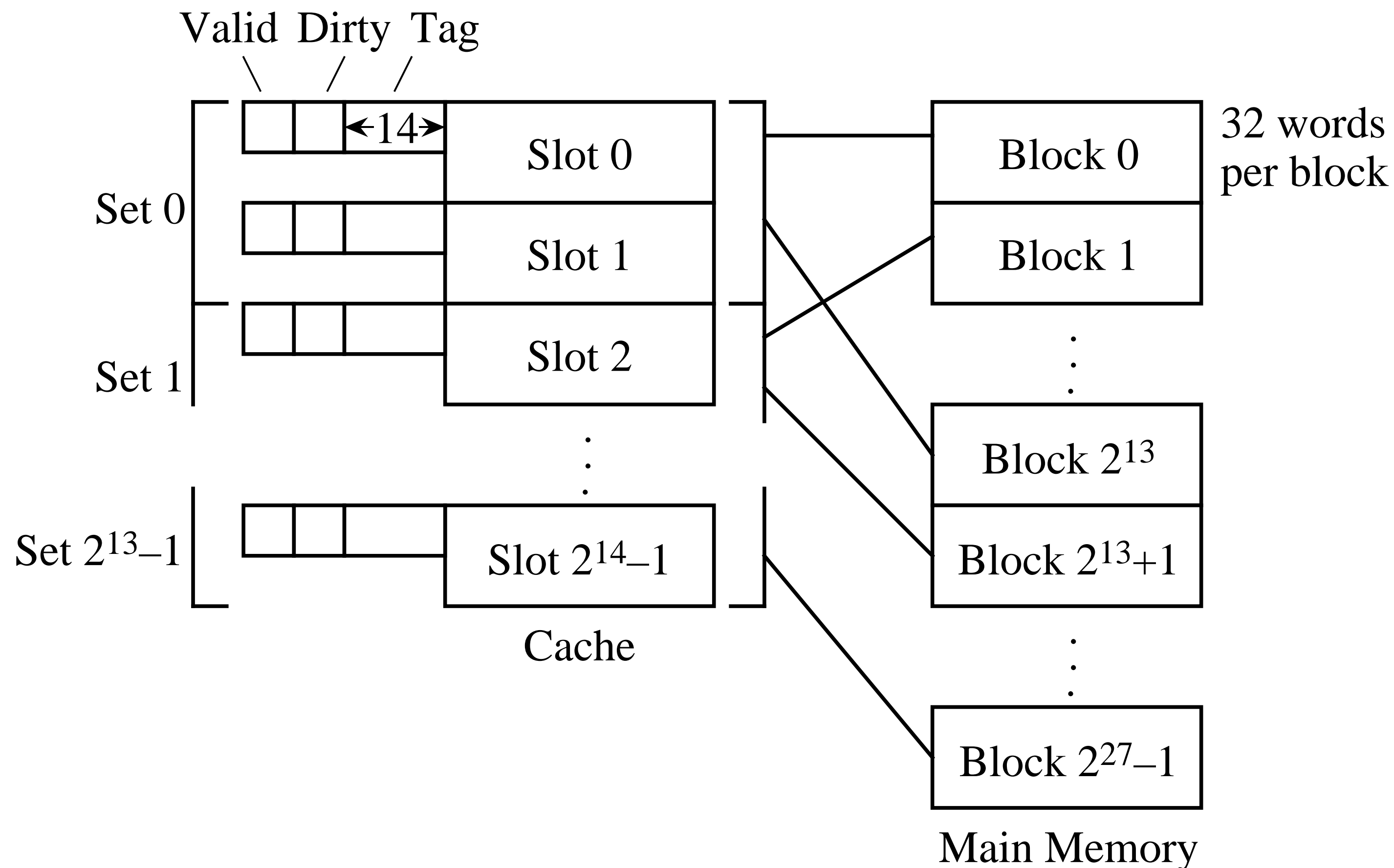
Tag	Slot	Word
13 bits	14 bits	5 bits

- If the addressed word is in the cache, it will be found in word $(14)_{16}$ of slot $(2F80)_{16}$, which will have a tag of $(1406)_{16}$.

Tag	Slot	Word
1 0 1 0 0 0 0 0 0 0 1 1 0	1 0 1 1 1 1 1 0 0 0 0 0 0 0	1 0 1 0 0

© 1999 M. Murdocca and V. Heuring

A Set Associative Mapping Scheme for a Cache Memory



 © 1999 M. Murdocca and V. Heuring

Set-Associative Mapping Example

- Consider how an access to memory location $(A035F014)_{16}$ is mapped to the cache for a 2^{32} word memory. The memory is divided into 2^{27} blocks of 2^5 = 32 words per block, there are two blocks per set, and the cache consists of 2^{14} slots:

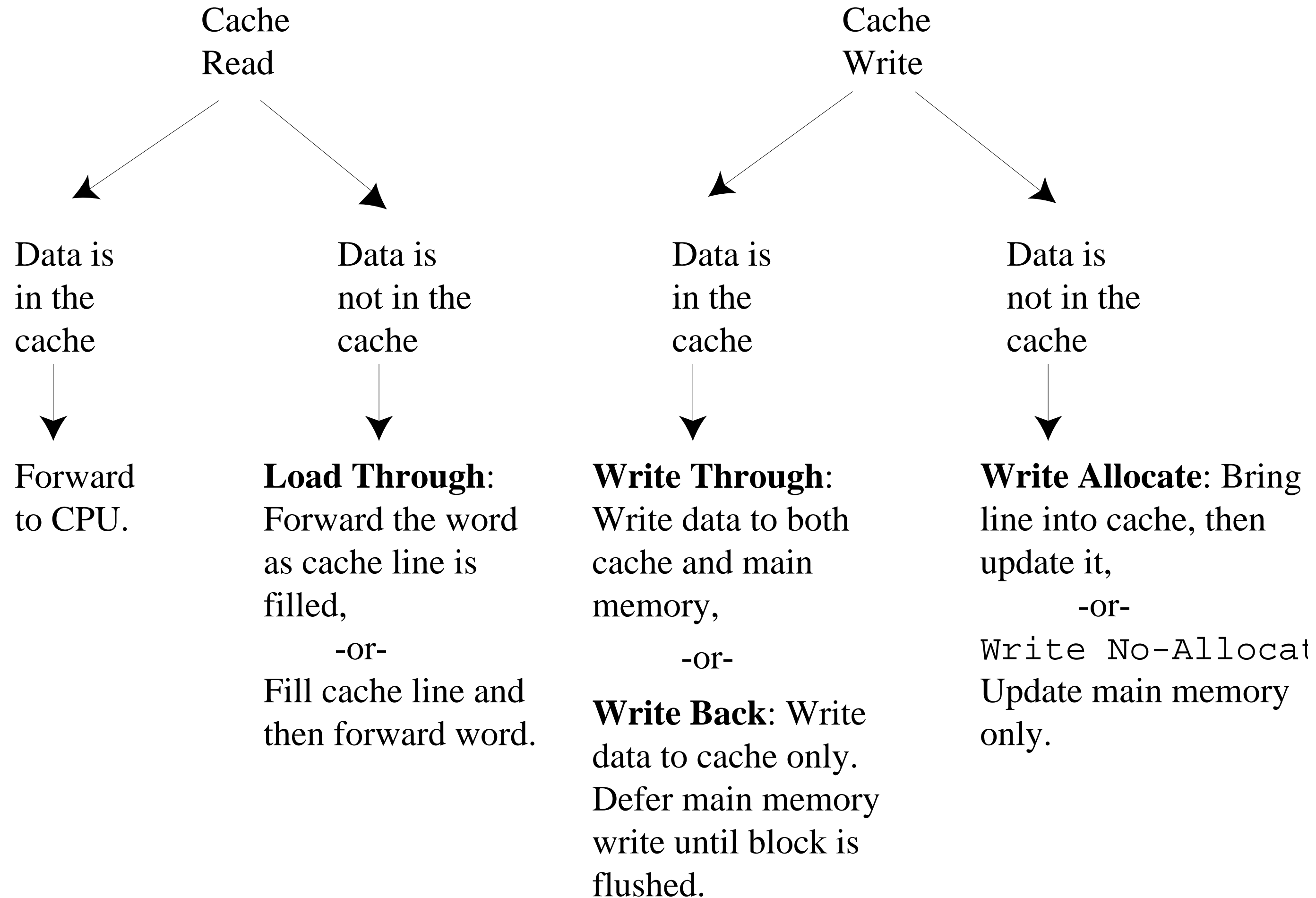
Tag	Set	Word
14 bits	13 bits	5 bits

- The leftmost 14 bits form the tag field, followed by 13 bits for the set field, followed by five bits for the word field:

Tag	Set	Word
1 0 1 0 0 0 0 0 0 0 1 1 0 1	0 1 1 1 1 1 0 0 0 0 0 0 0	1 0 1 0 0

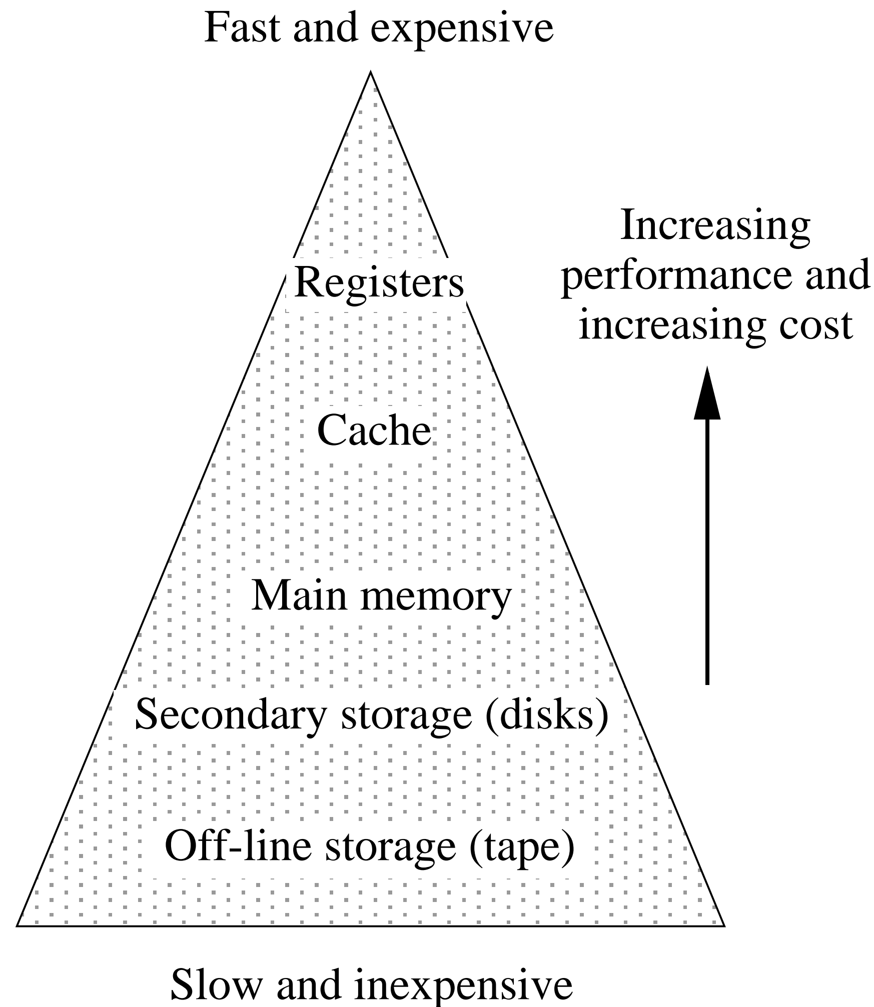
 © 1999 M. Murdocca and V. Heuring

Cache Read and Write Policies



 © 1999 M. Murdocca and V. Heuring

The Memory Hierarchy



© 1999 M. Murdocca and V. Heuring

Next

- Midterm Exam

Next -> Next

- Digital Logic