COP 4600 – Summer 2011

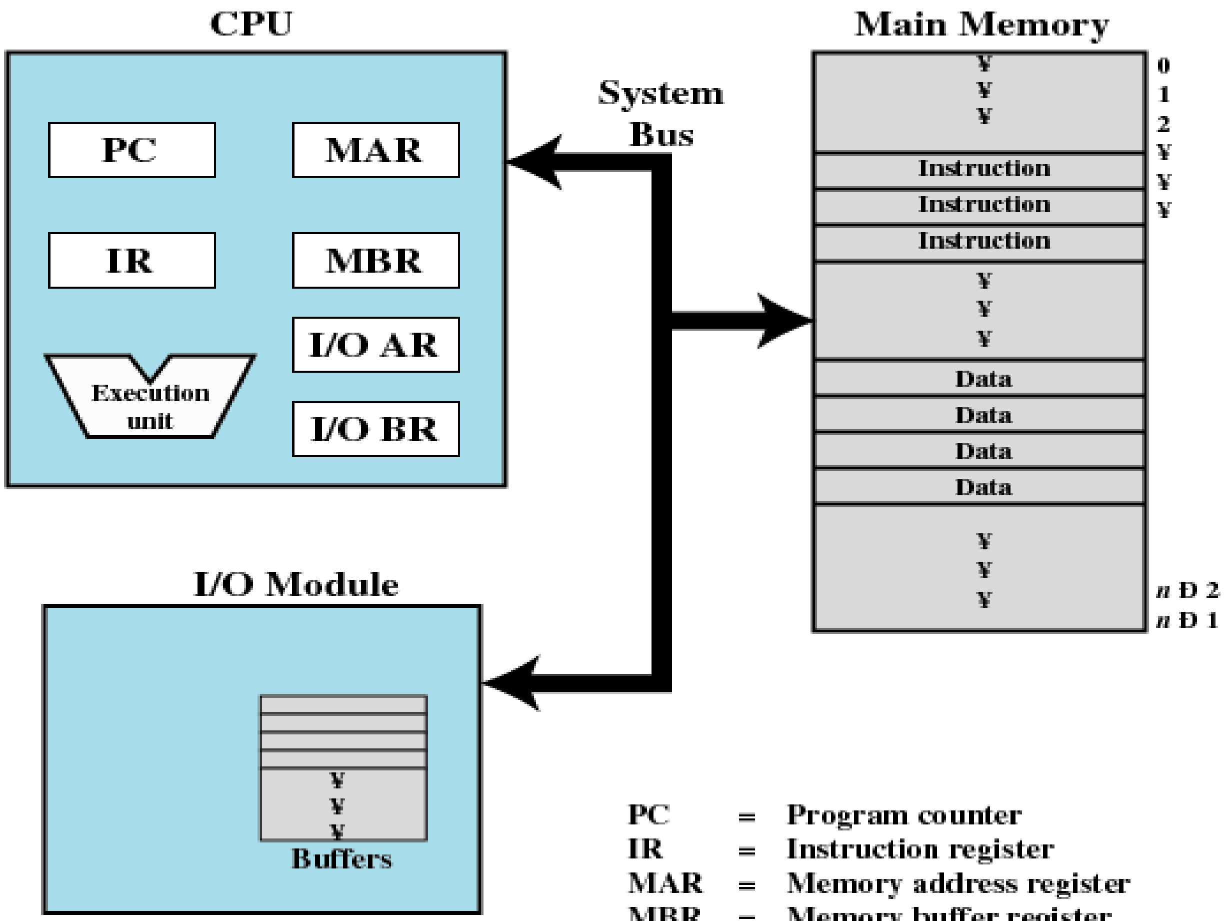
Introduction To Operating Systems

Hardware Considerations For OS

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop4600/sum2011

Department of Electrical Engineering and Computer Science
Computer Science Division
University of Central Florida

Top-Level Computer Components



PC	=	Program counter
IR	=	Instruction register
MAR	=	Memory address register
MBR	=	Memory buffer register
I/O AR	=	Input/output address register
I/O BR	=	Input/output buffer register

Processor Registers

- User-visible registers
  - Enable programmer to minimize main-memory references by optimizing register use
- Control and status registers
  - Used by processor to control operating of the processor
  - Used by privileged operating-system routines to control the execution of programs

User-Visible Registers

- May be referenced by machine language
- Available to all programs - application programs and system programs
- Types of registers
  - Data
  - Address
    - Index
    - Segment pointer
    - Stack pointer

User-Visible Registers

- Address Registers
  - Index
    - Involves adding an index to a base value to get an address
  - Segment pointer
    - When memory is divided into segments, memory is referenced by a segment and an offset
  - Stack pointer
    - Points to top of stack

Control and Status Registers

- Program Counter (PC)
  - Contains the address of an instruction to be fetched
- Instruction Register (IR)
  - Contains the instruction most recently fetched
- Program Status Word (PSW)
  - Condition codes
  - Interrupt enable/disable
  - Supervisor/user mode

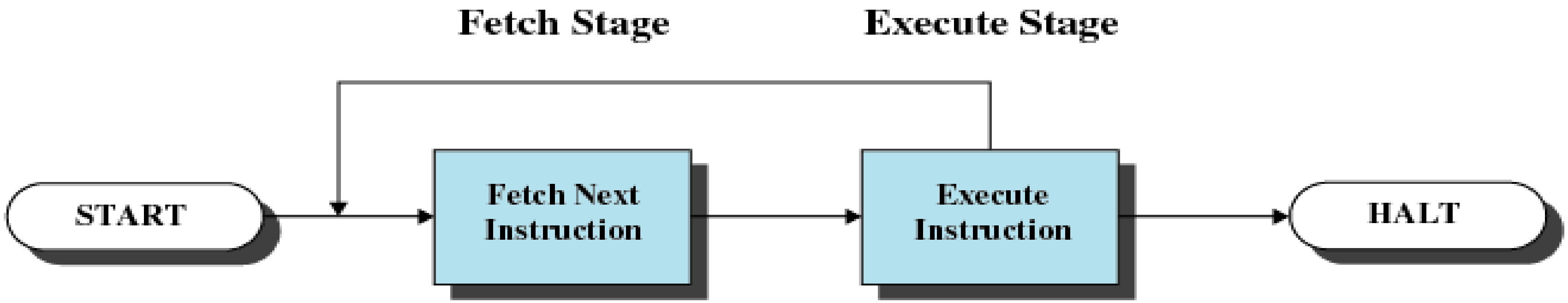
Control and Status Registers

- Condition Codes or Flags
  - Bits set by the processor hardware as a result of operations
  - Examples
    - Positive result
    - Negative result
    - Zero
    - Overflow

Instruction Execution

- Two steps (Fetch/Decode & Execute)
  - – Processor reads instructions from memory
    - Fetches into the MBR (MDR)
  - – Processor executes each instruction

Instruction Cycle



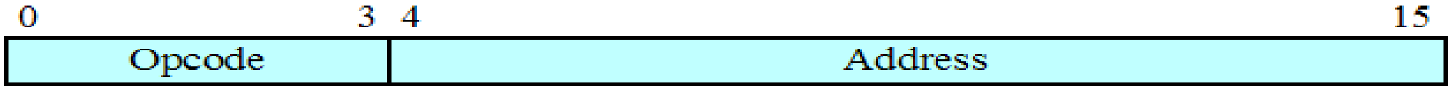
Instruction Fetch and Execute

- The processor fetches the instruction from memory
- Program counter (PC) holds address of the instruction to be fetched next
- Program counter is incremented after each fetch

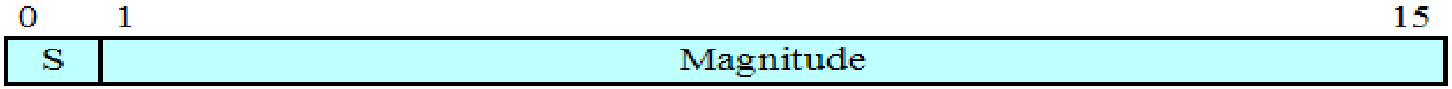
Instruction Register

- Fetched instruction is placed in the instruction register
- Categories
  - Processor-memory
    - Transfer data between processor and memory
  - Processor-I/O
    - Data transferred to or from a peripheral device
  - Data processing
    - Arithmetic or logic operation on data
  - Control
    - Alter sequence of execution

A Hypothetical Machine



Instruction format



Integer format

16-bit word = 4 bytes

Program Counter (PC) = Address of instruction
Instruction Register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

Internal CPU registers

0001 = Load AC from Memory
0010 = Store AC to Memory
0101 = Add to AC from Memory

Instructions are 4 bits = 1 byte = 1 hexadecimal digit

Partial list of opcodes

An Aside on Number Systems

Binary = base 2, digits are 0, 1

Decimal = base 10, digits are 0,1,2,3,4,5,6,7,8,9

Hexadecimal = base 16, digits are 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

1 **bi**nary digi**t** = 1 bit. 1 bit can represent either 0 or 1 (e.g. on or off)

Positional notation is represented via digit position as a power of 2.

$(x + 2^n) + \ldots + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$ (recall that $x^0 = 1$ and $0^x = 0$)

So, 2 bits can represent up to base 4 (digits 0,1,2,3), since

$(0 + 2^1) + (0 + 2^0) = 0$ $(0 + 2^1) + (1 + 2^0) = 1$

$(1 + 2^1) + (0 + 2^0) = 2$ $(1 + 2^1) + (1 + 2^0) = 3$

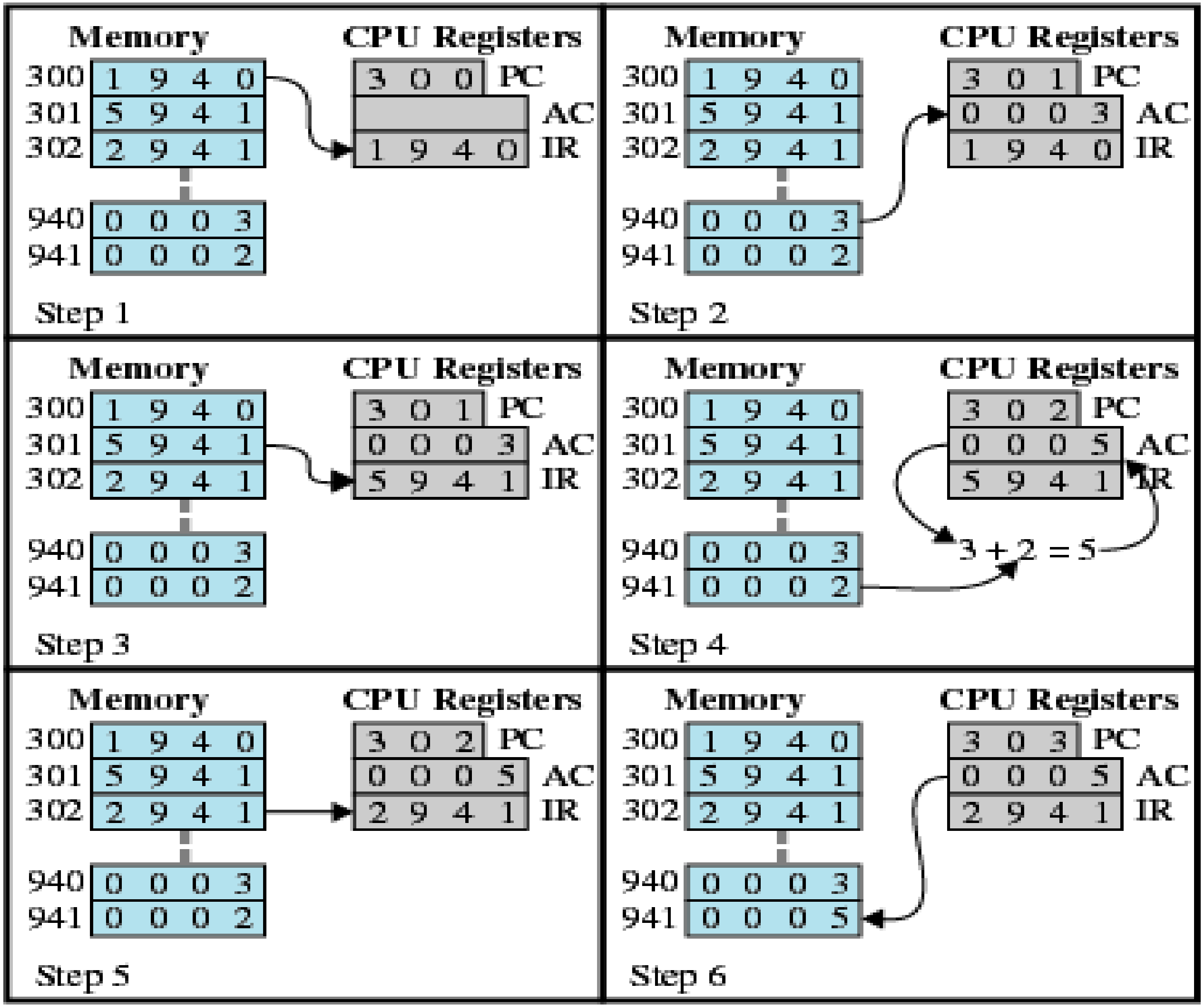
An Aside on Number Systems

HEX	DECIMAL	OCTAL	BINARY
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
A	10	12	1010
B	11	13	1011
C	12	14	1100
D	13	15	1101
E	14	16	1110
F	15	17	1111

For hexadecimal, 4 bits are required to represent 1 hex digit.

The hex number 345 in binary would be written as 1101000101 (leading zeros are dropped).

$$\underbrace{1\;1}_{=\,3_{16}}\;\underbrace{0\;1\;0\;0}_{=\,4_{16}}\;\underbrace{0\;1\;0\;1}_{=\,5_{16}}$$



Step 1

Memory	
300	1 9 4 0
301	5 9 4 1
302	2 9 4 1
940	0 0 0 3
941	0 0 0 2

CPU Registers	
3 0 0	PC
	AC
1 9 4 0	IR

Step 2

Memory	
300	1 9 4 0
301	5 9 4 1
302	2 9 4 1
940	0 0 0 3
941	0 0 0 2

CPU Registers	
3 0 1	PC
0 0 0 3	AC
1 9 4 0	IR

Step 3

Memory	
300	1 9 4 0
301	5 9 4 1
302	2 9 4 1
940	0 0 0 3
941	0 0 0 2

CPU Registers	
3 0 1	PC
0 0 0 3	AC
5 9 4 1	IR

Step 4

Memory	
300	1 9 4 0
301	5 9 4 1
302	2 9 4 1
940	0 0 0 3
941	0 0 0 2

CPU Registers	
3 0 2	PC
0 0 0 5	AC
5 9 4 1	IR

3 + 2 = 5

Step 5

Memory	
300	1 9 4 0
301	5 9 4 1
302	2 9 4 1
940	0 0 0 3
941	0 0 0 2

CPU Registers	
3 0 2	PC
0 0 0 5	AC
2 9 4 1	IR

Step 6

Memory	
300	1 9 4 0
301	5 9 4 1
302	2 9 4 1
940	0 0 0 3
941	0 0 0 5

CPU Registers	
3 0 3	PC
0 0 0 5	AC
2 9 4 1	IR

Explanation of Example Program Execution

1. The PC contains 300, the address of the first instruction. This instruction (with value 1940 in hexadecimal) is loaded into the instruction register (IR) and the program counter (PC) is incremented. Note that both the MAR and MDR are used in this step but are not shown in the previous slide.

2. The first 4 bits (first hex digit) in the IR indicates that the accumulator register (AC) is to be loaded from memory. Decode of opcode indicates 0001 which represents the load accumulator from memory instruction. The remaining 12 bits (3 hex digits) represents the memory address where the operand (the value to be loaded into the accumulator) is located. This address is 940 (also in hex).

Explanation of Example Program Execution

3. The next instruction (hex 5941), is fetched from memory location 301 and the PC is incremented. The decode of this instruction indicates an instruction (opcode 0101) that adds the value in the AC to a value obtained from the memory location specified in the operand of the instruction as (hex 941).

4. The old contents of the AC and the contents of memory location 941 are added together and the result is stored in the AC.

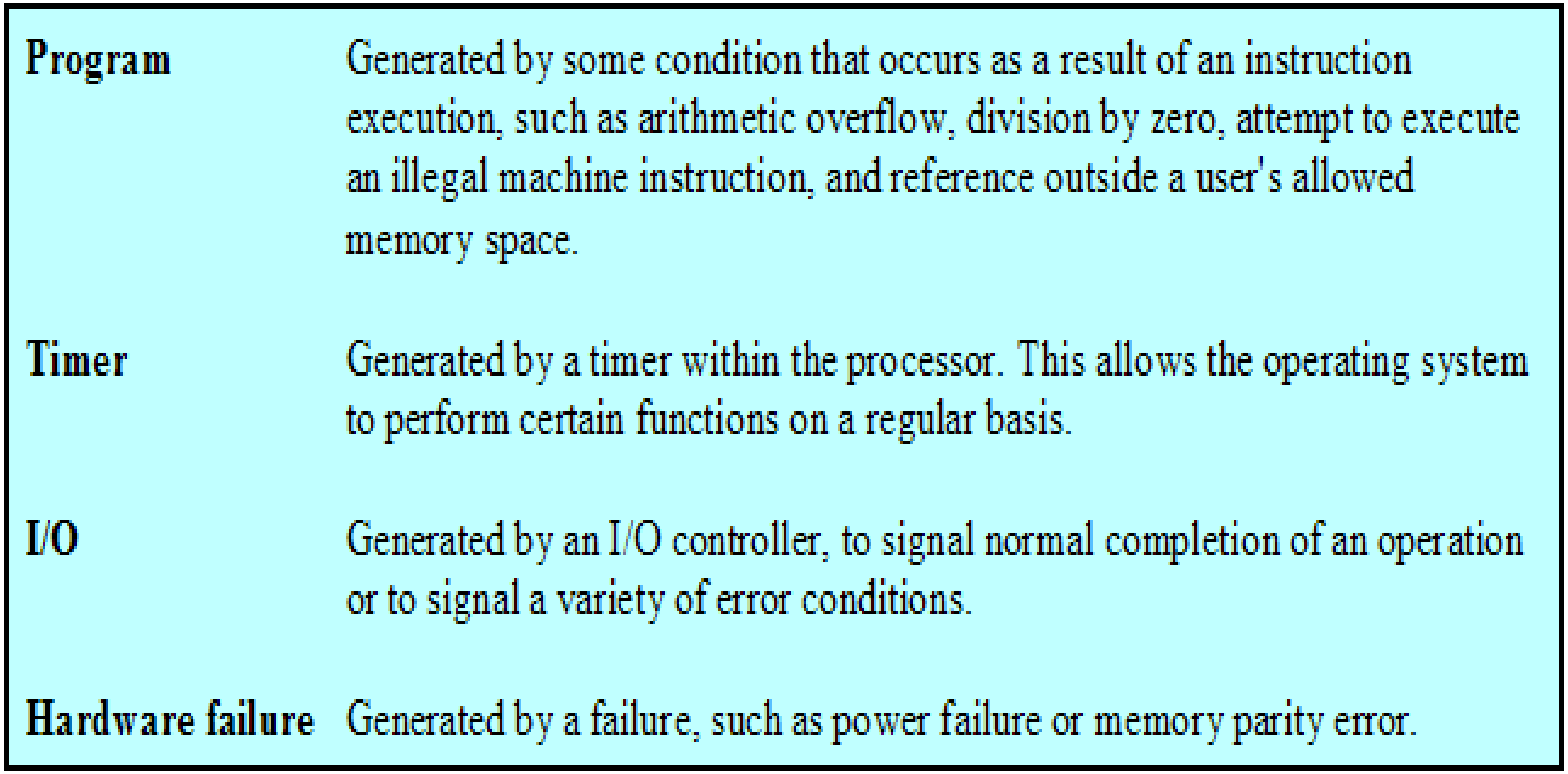
5. The next instruction (hex 2941), is fetched from memory location 302 and the PC is incremented. The decode of this instruction indicates an instruction (opcode 0010) that stores the contents of the AC into the memory location specified in the operand of the instruction as (hex 941).

6. The contents of the AC are stored in memory location 941.

Interrupts

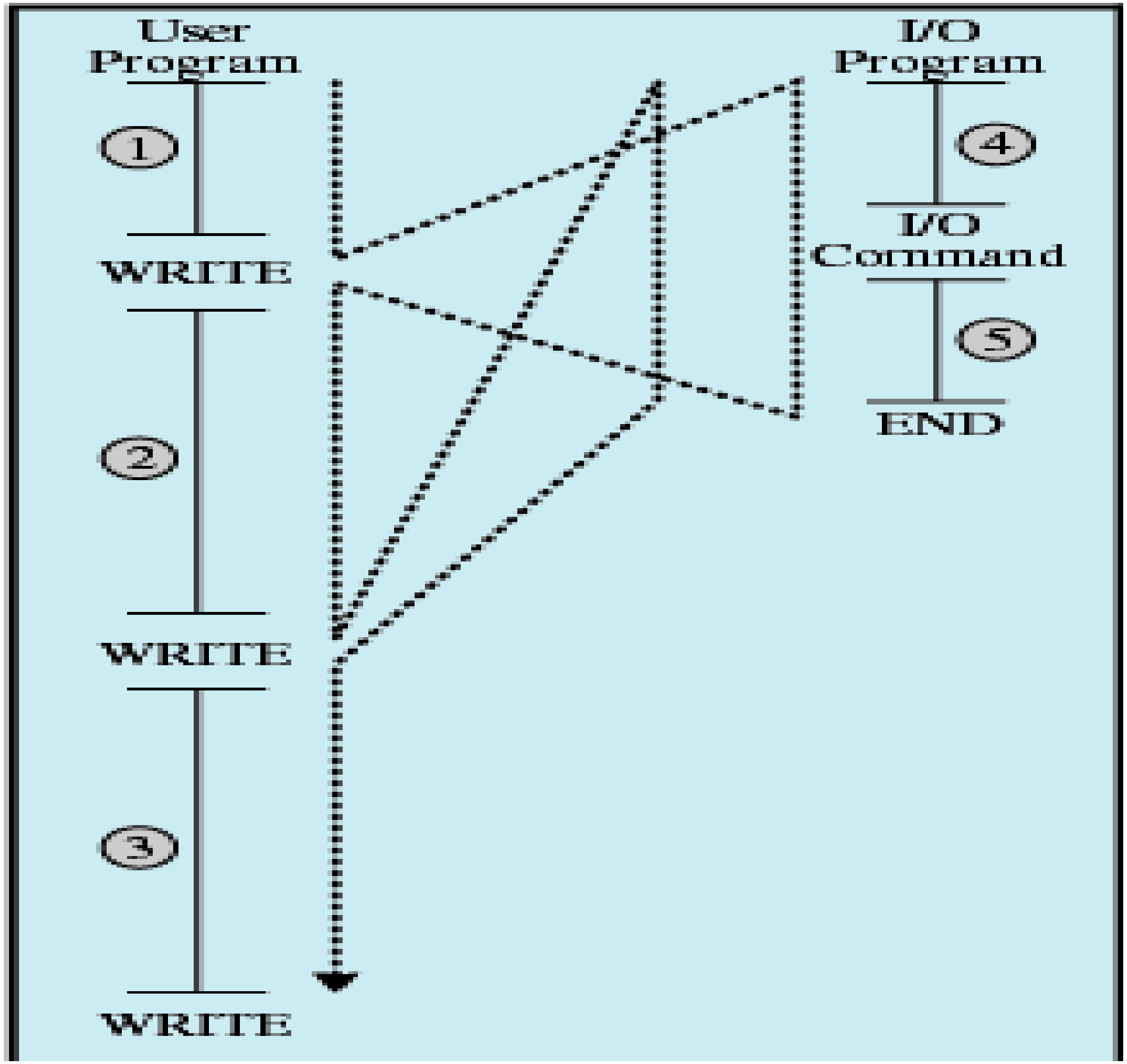
- Interrupt the normal sequencing of the processor
- Most I/O devices are slower than the processor
  - Processor must pause to wait for device

Classes of Interrupts

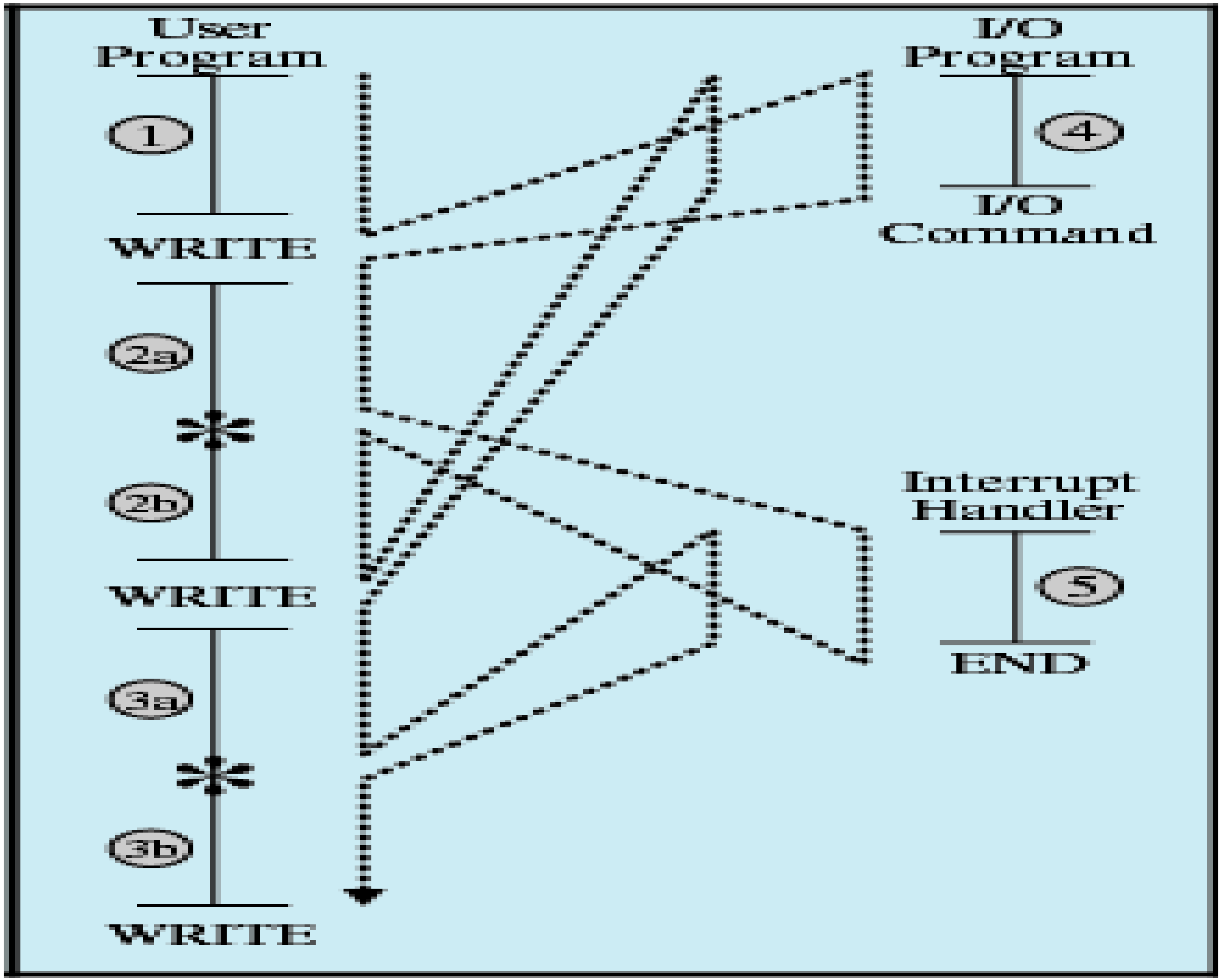


Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
Hardware failure	Generated by a failure, such as power failure or memory parity error.

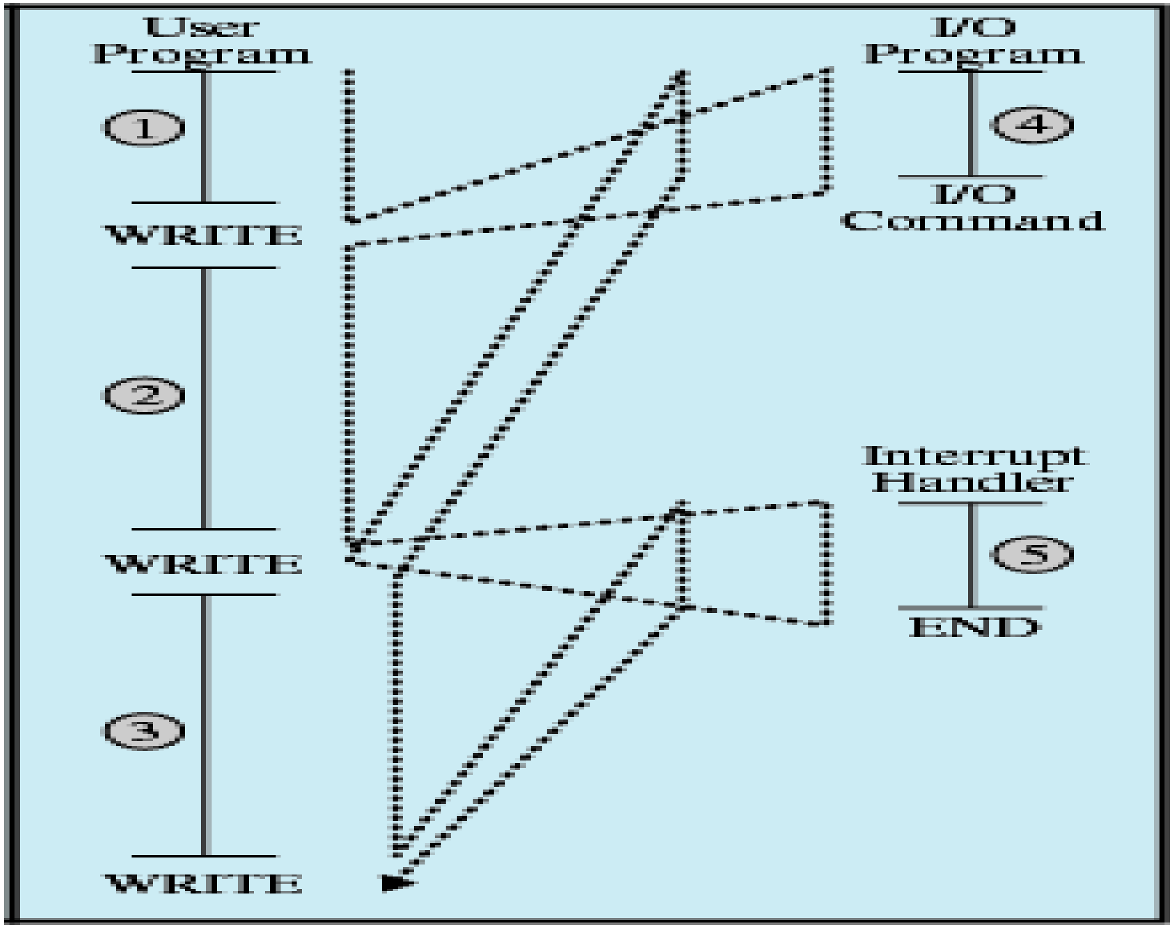
Program Flow of Control Without Interrupts



Program Flow of Control With Interrupts, Short I/O Wait



Program Flow of Control With Interrupts; Long I/O Wait

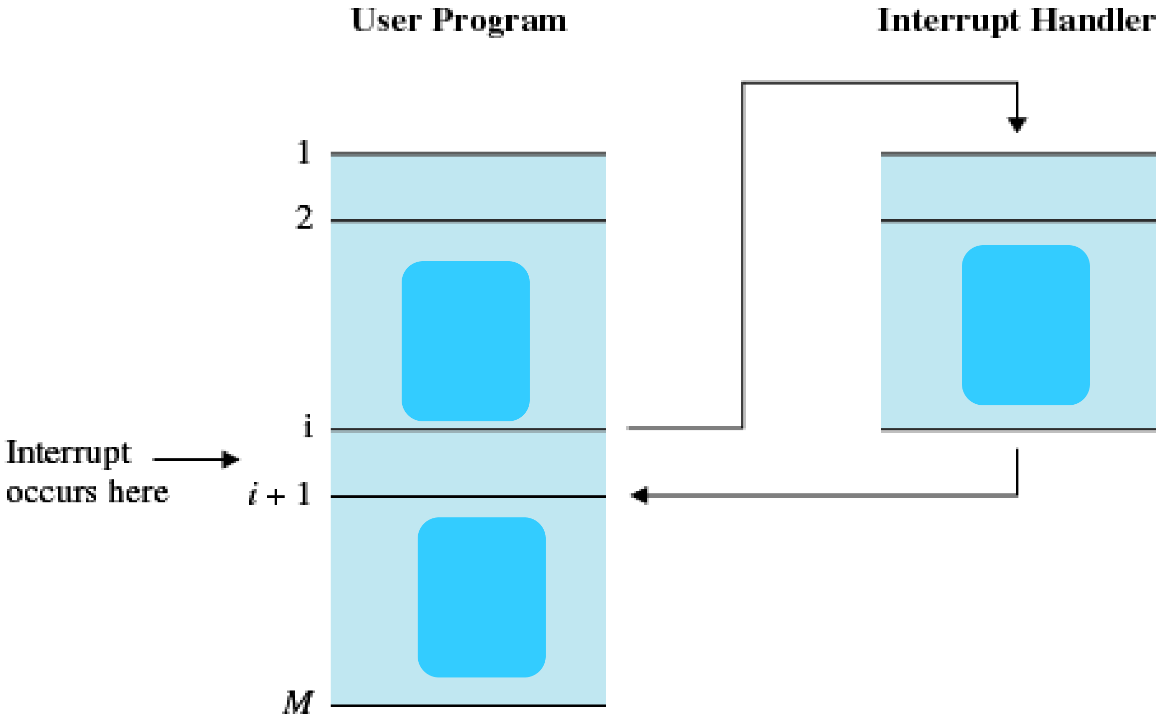


Interrupt Handler

- Program to service a particular I/O device
- Generally part of the operating system

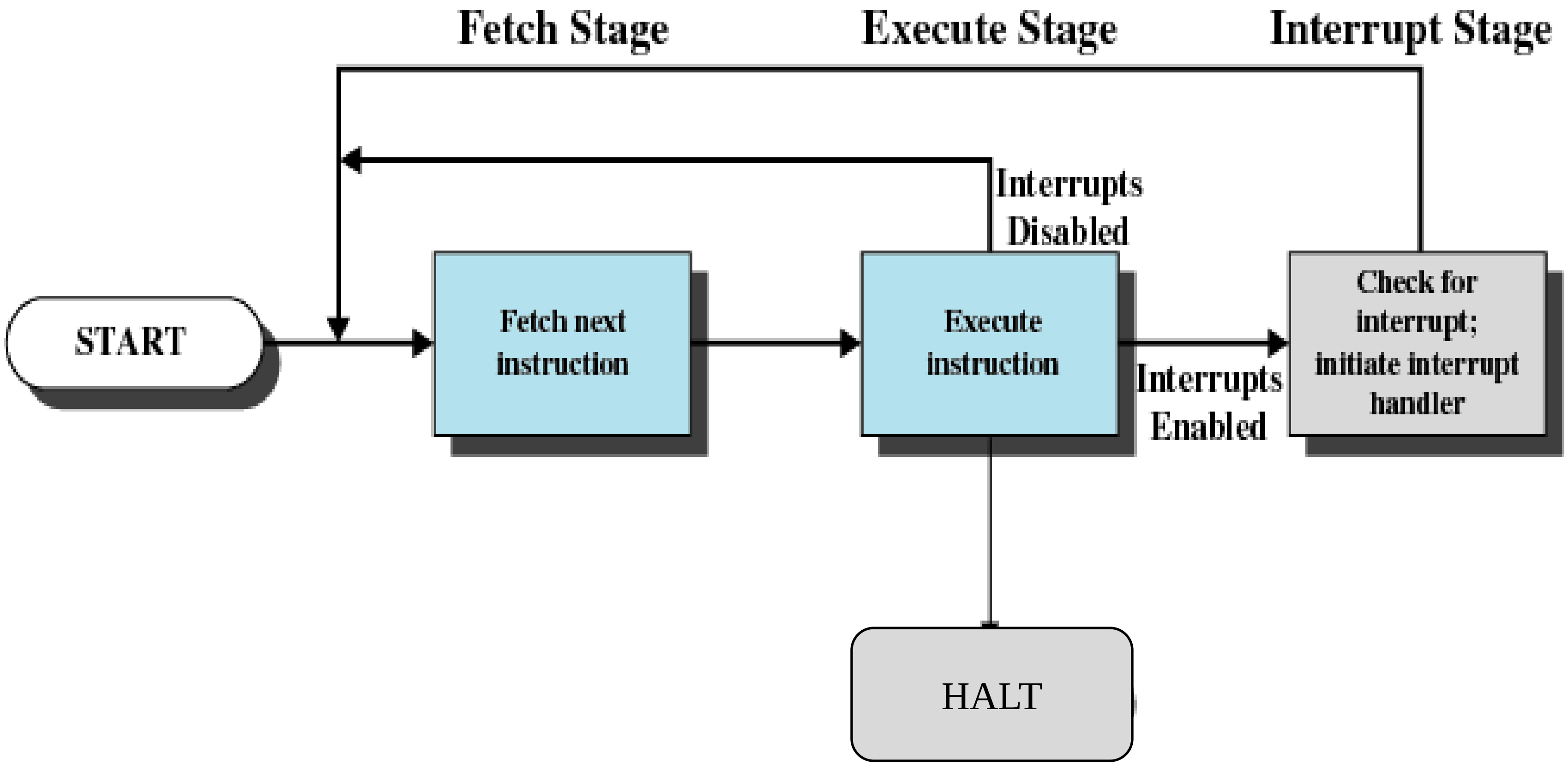
Interrupts

- Suspends the normal sequence of execution



Fig

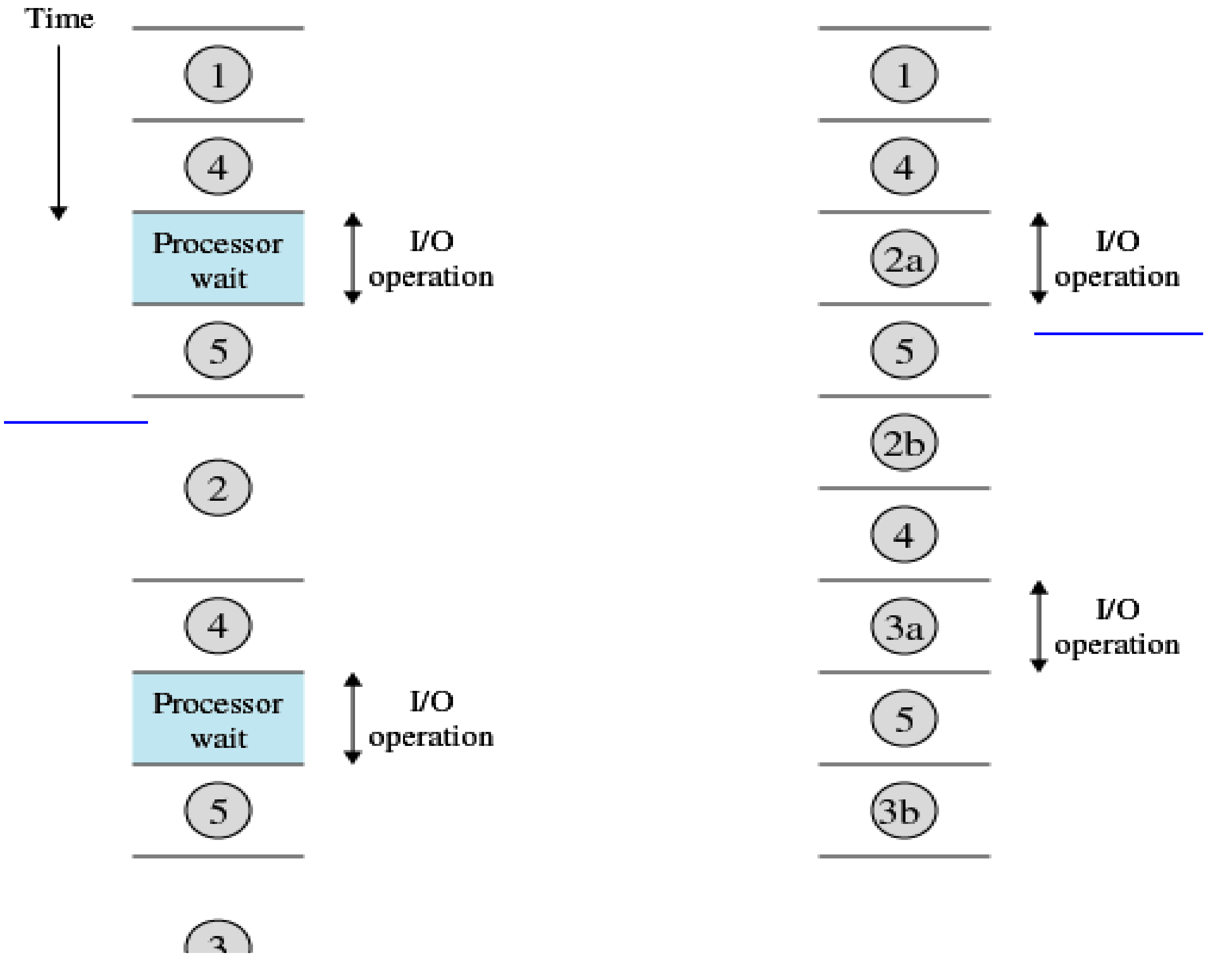
Interrupt Cycle



Interrupt Cycle

- Processor checks for interrupts
- If no interrupts fetch the next instruction for the current program
- If an interrupt is pending, suspend execution of the current program, and execute the interrupt-handler routine

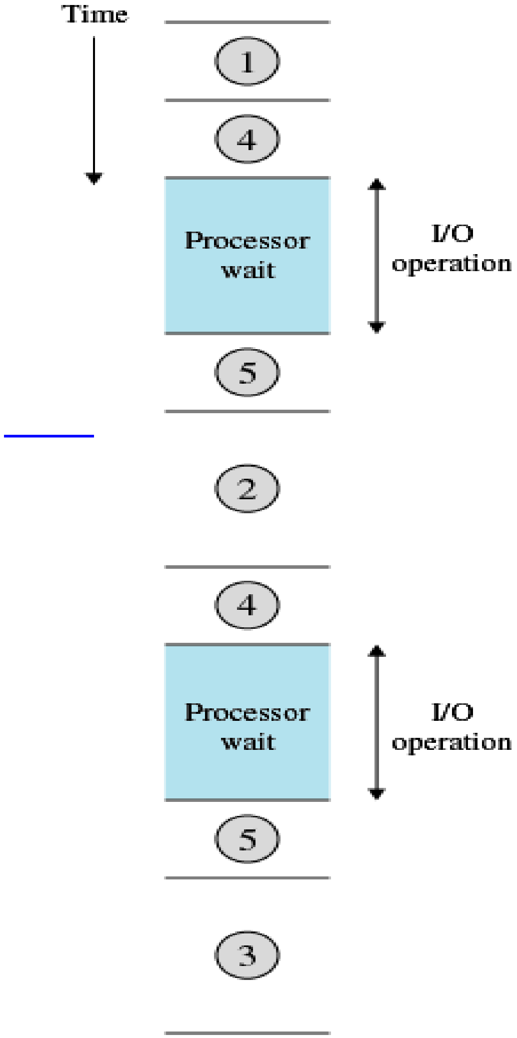
Timing Diagram Based on Short I/O Wait



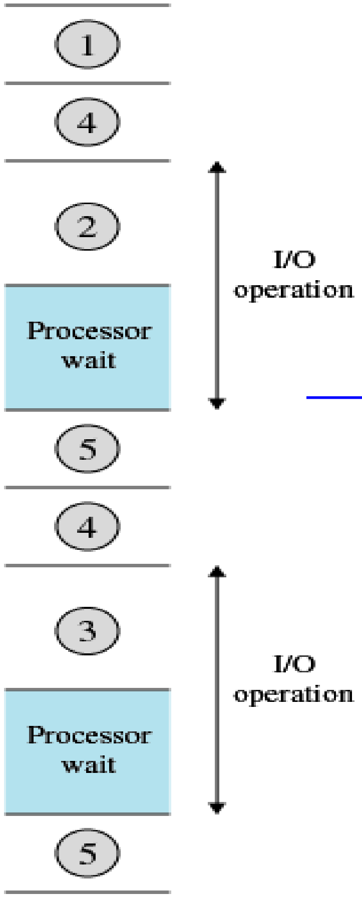
Numbers refer to page 20 – no interrupts

Numbers refer to page 21 – interrupts enabled – short I/O wait

Timing Diagram Based on Long I/O Wait

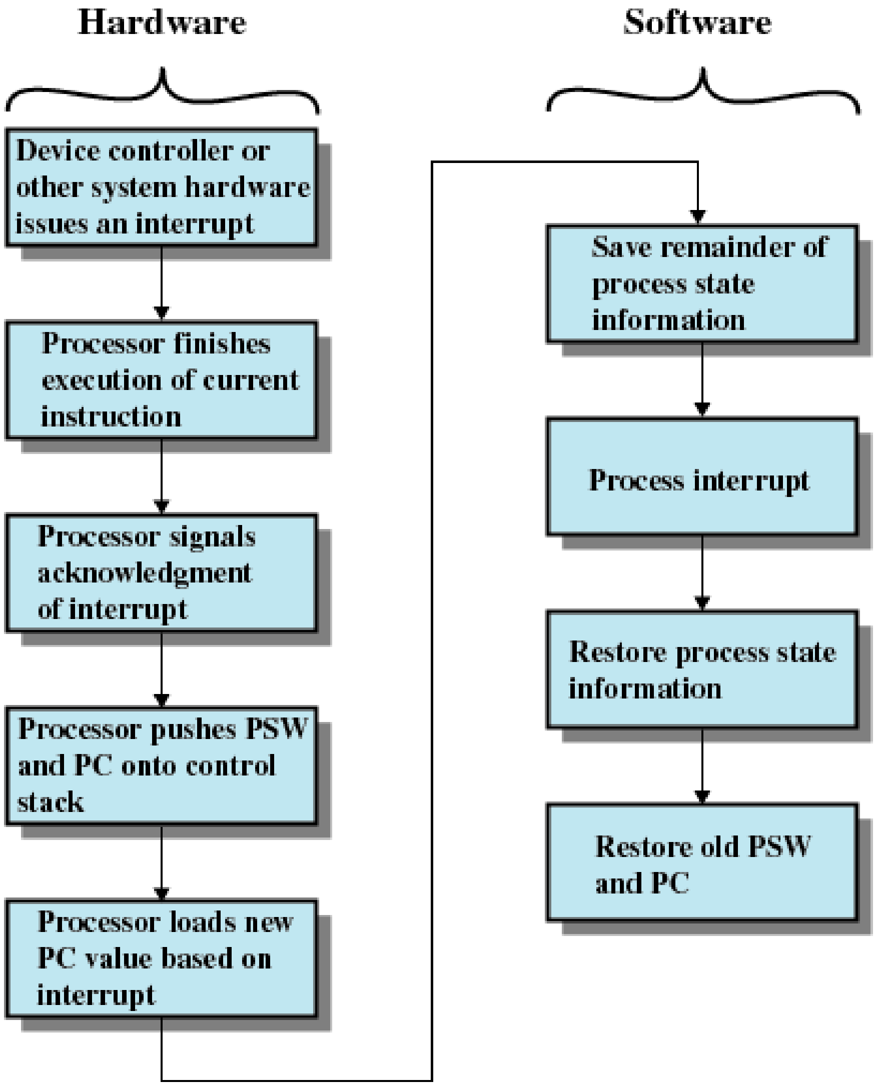


Numbers refer to page 20 – no interrupts

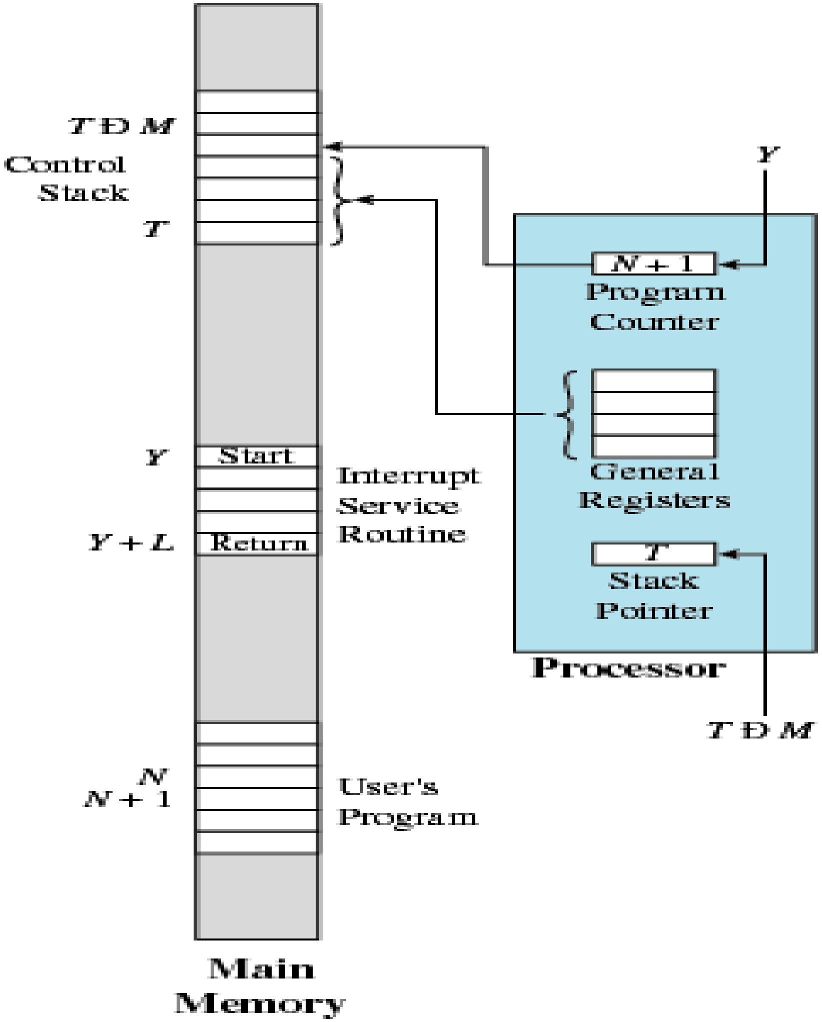


Numbers refer to page 22 – interrupts enabled with long I/O wait

Simple Interrupt Processing

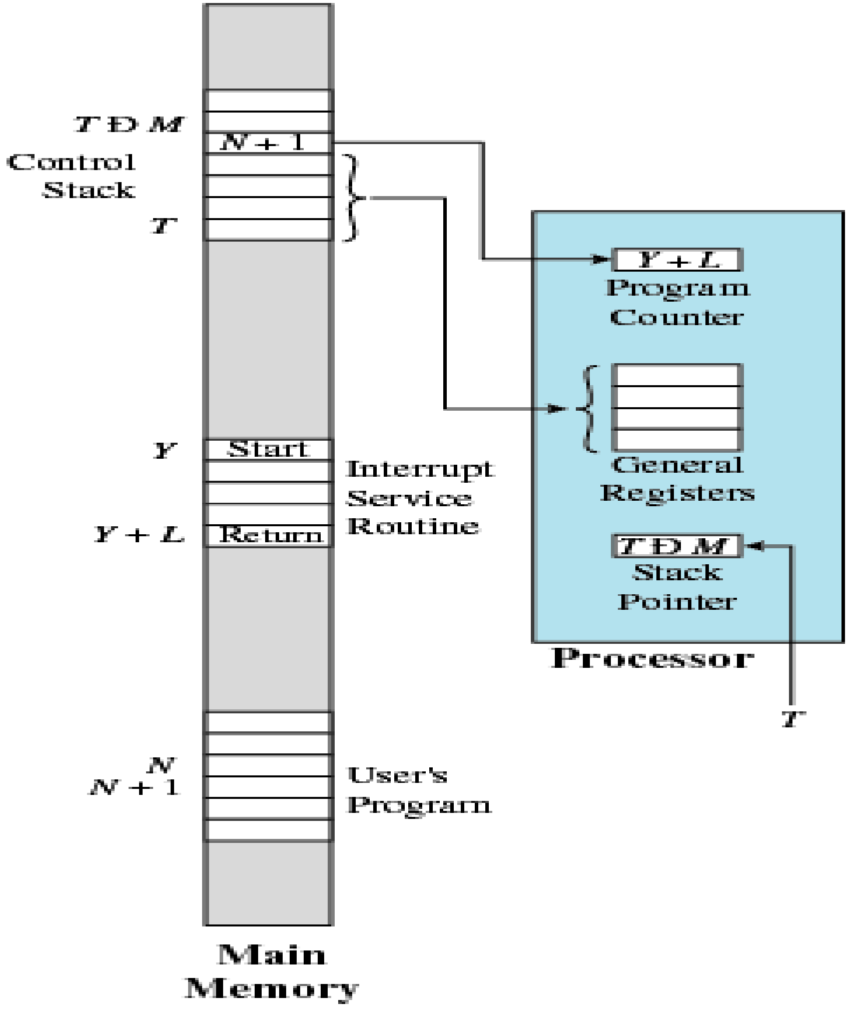

Hardware
Software
Device controller or other system hardware issues an interrupt
Processor finishes execution of current instruction
Processor signals acknowledgment of interrupt
Processor pushes PSW and PC onto control stack
Processor loads new PC value based on interrupt
Save remainder of process state information
Process interrupt
Restore process state information
Restore old PSW and PC

Changes in Memory and Registers for an Interrupt



Interrupt occurs after instruction at location N

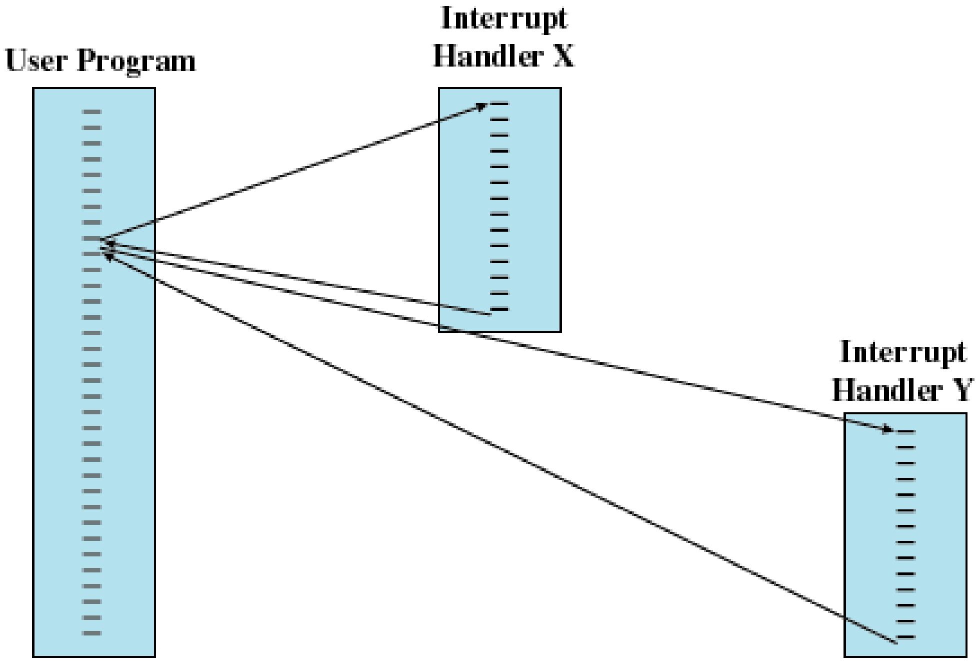
Changes in Memory and Registers for an Interrupt



Return from interrupt

Multiple Interrupts

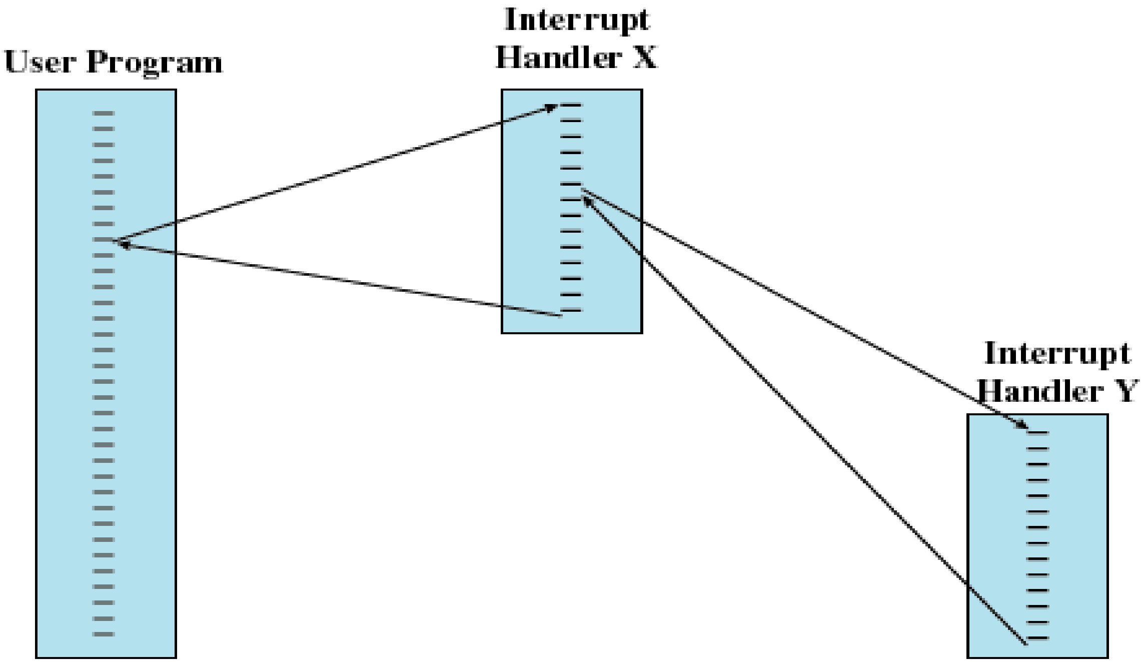
- Disable interrupts while an interrupt is being processed



Sequential interrupt processing

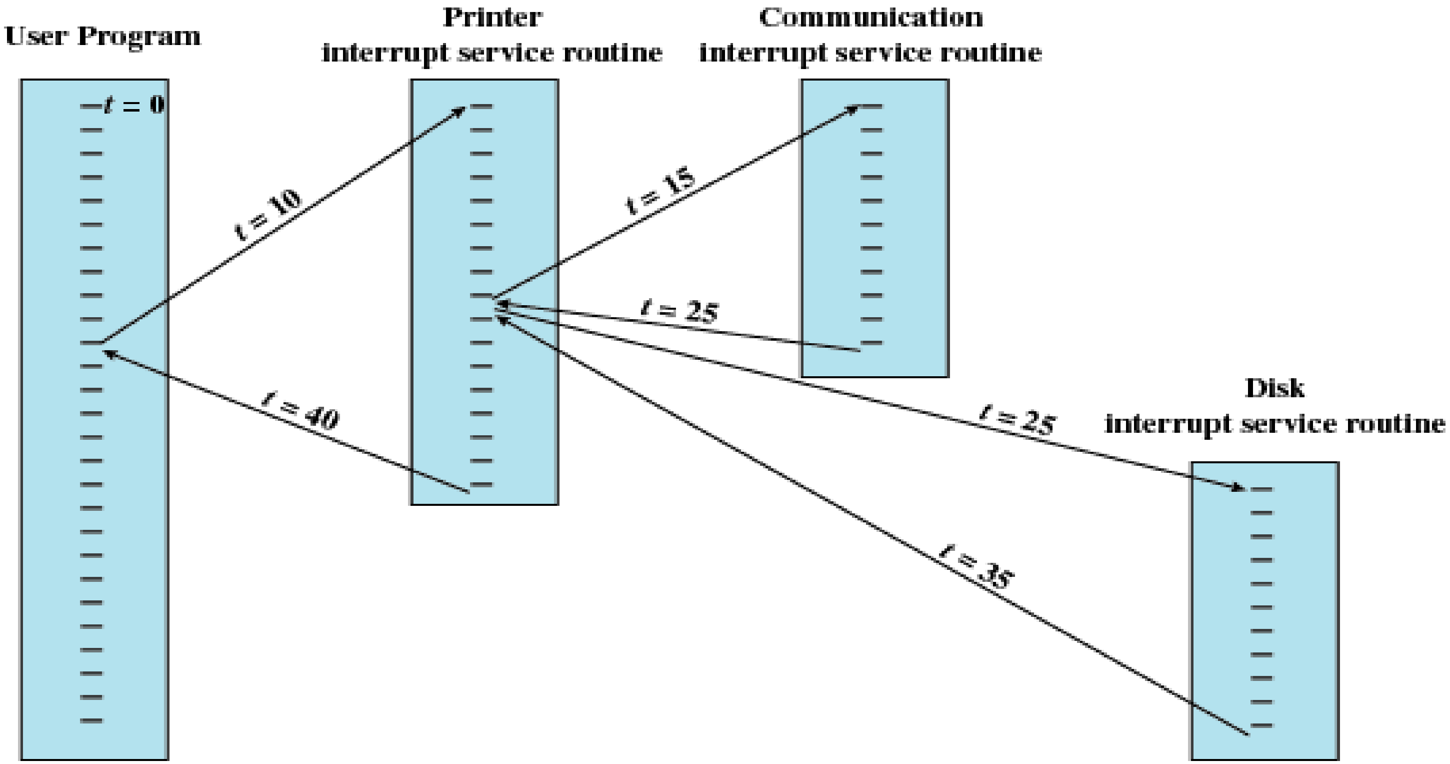
Multiple Interrupts

- Define priorities for interrupts



Nested interrupt processing

Multiple Interrupts



Example Time Sequence of Multiple Interrupts

Multiprogramming

- Processor has more than one program to execute
- The sequence the programs are executed depend on their relative priority and whether they are waiting for I/O
- After an interrupt handler completes, control may not return to the program that was executing at the time of the interrupt

Memory Hierarchy

- Faster access time, greater cost per bit
- Greater capacity, smaller cost per bit
- Greater capacity, slower access speed

Disk Cache

- A portion of main memory used as a buffer to temporarily to hold data for the disk

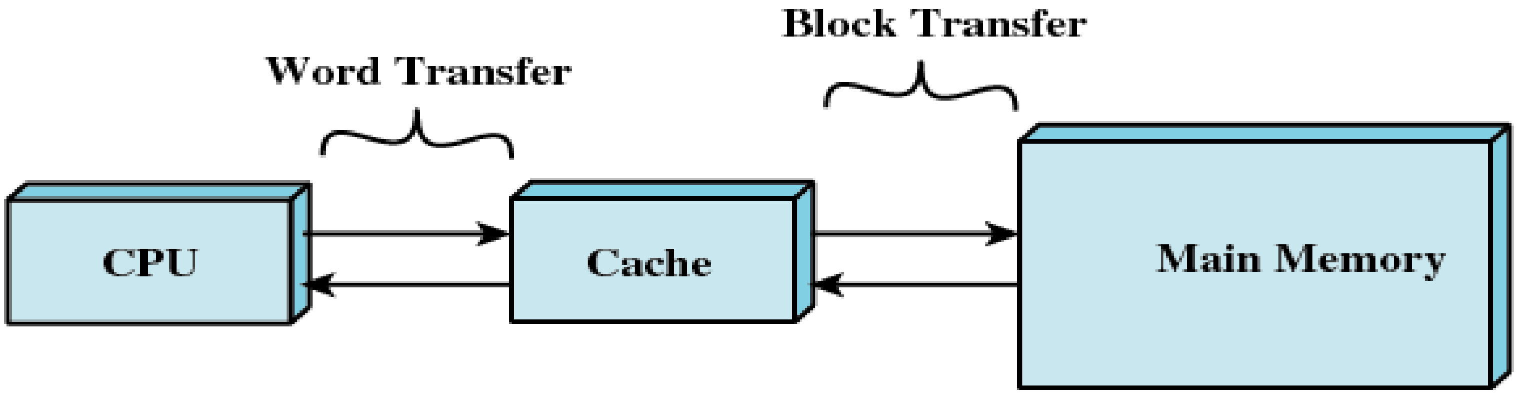
- Disk writes are clustered

- Some data written out may be referenced again. The data are retrieved rapidly from the software cache instead of slowly from disk

Cache Memory

- Invisible to operating system
- Increase the speed of memory
- Processor speed is faster than memory speed
- Exploit the principle of locality

Cache Memory

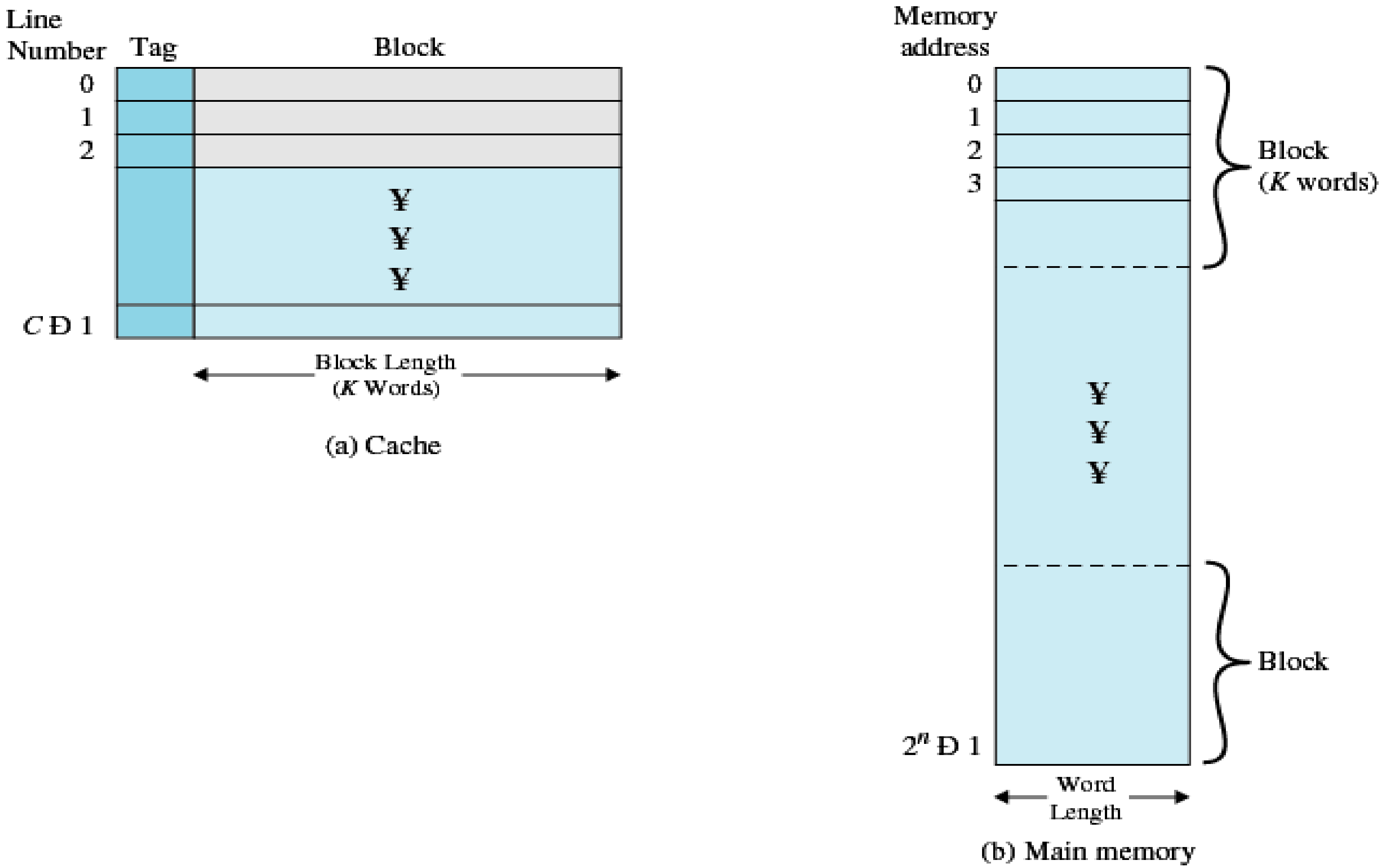


Cache and Main Memory

Cache Memory

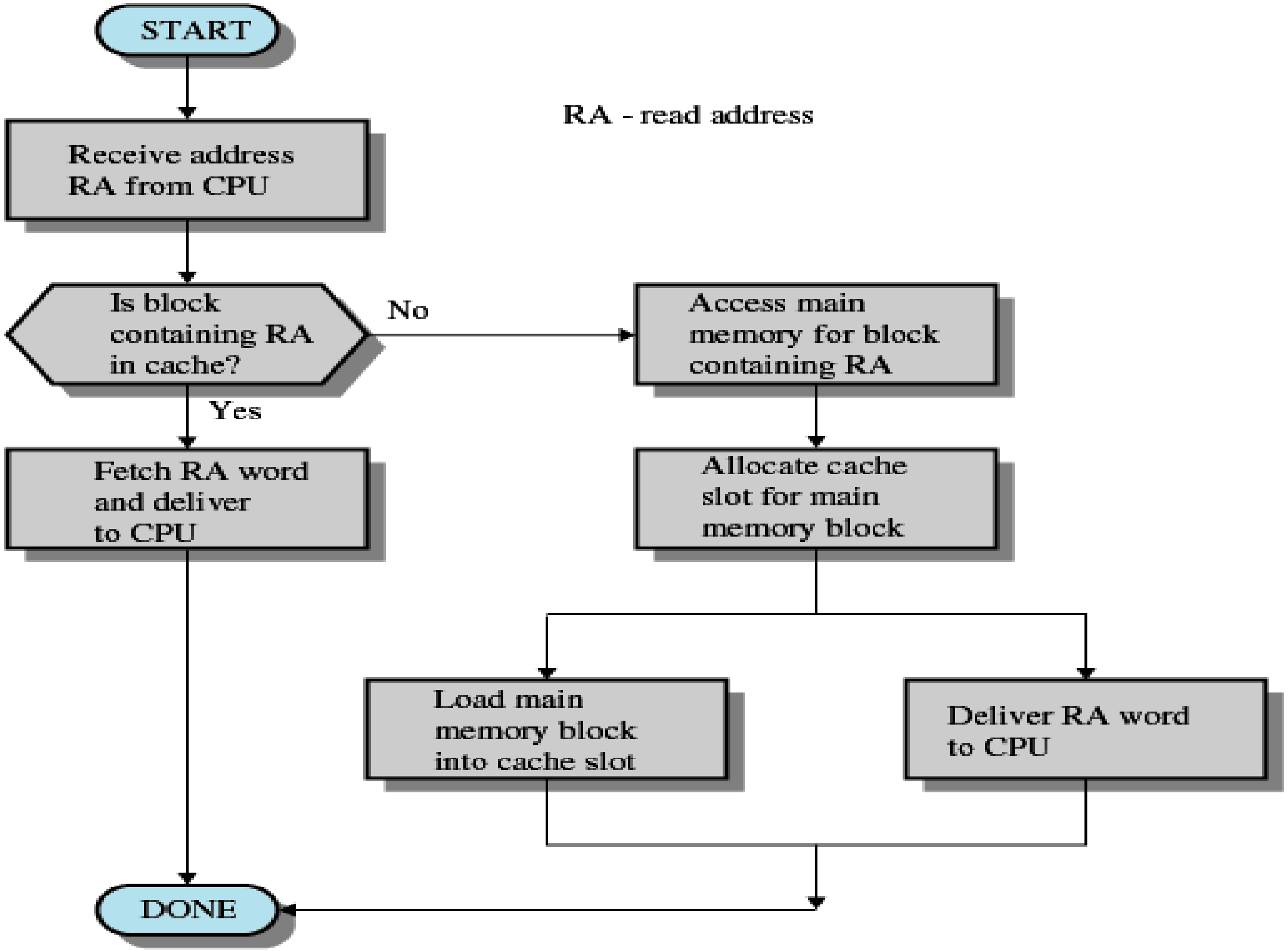
- Contains a copy of a portion of main memory
- Processor first checks cache
- If not found in cache, the block of memory containing the needed information is moved to the cache and delivered to the processor

Cache/Main Memory System



Cache/Main-Memory Structure

Cache Read Operation


START
RA - read address
Receive address RA from CPU
Is block containing RA in cache?
No
Yes
Access main memory for block containing RA
Fetch RA word and deliver to CPU
Allocate cache slot for main memory block
Load main memory block into cache slot
Deliver RA word to CPU
DONE

Cache Design

- Cache size
  - Small caches have a significant impact on performance
- Block size
  - The unit of data exchanged between cache and main memory
  - Larger block size more hits until probability of using newly fetched data becomes less than the probability of reusing data that have to be moved out of cache

Cache Design

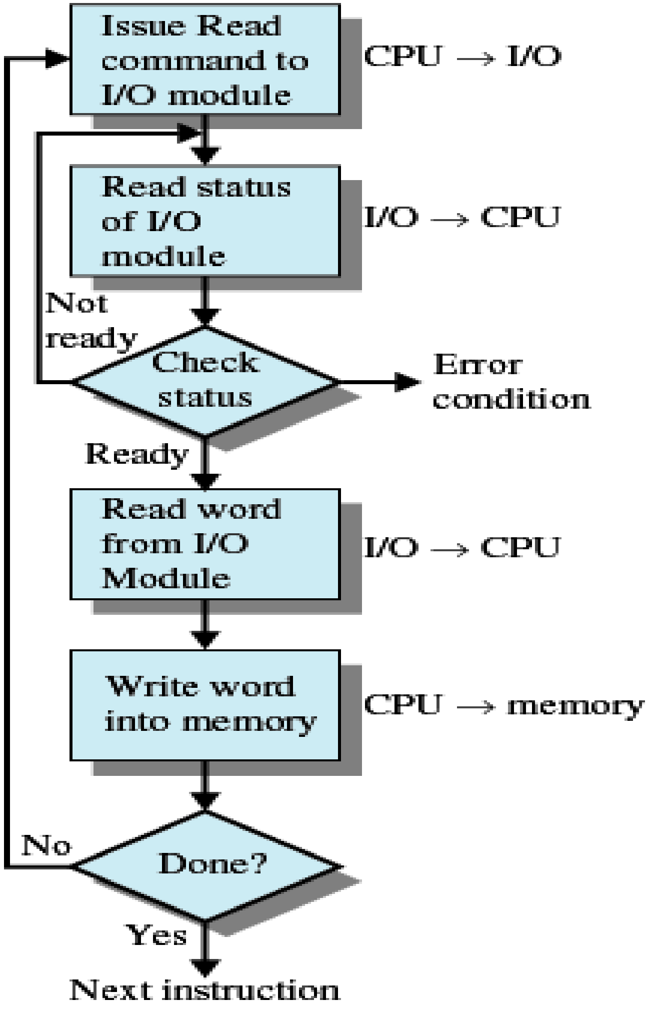
- Mapping function
  - Determines which cache location the block will occupy
- Replacement algorithm
  - Determines which block to replace
  - Least-Recently-Used (LRU) algorithm

Cache Design

- Write policy
  - When the memory write operation takes place
  - Can occur every time block is updated
  - Can occur only when block is replaced
    - Minimizes memory write operations
    - Leaves main memory in an obsolete state

Programmed I/O

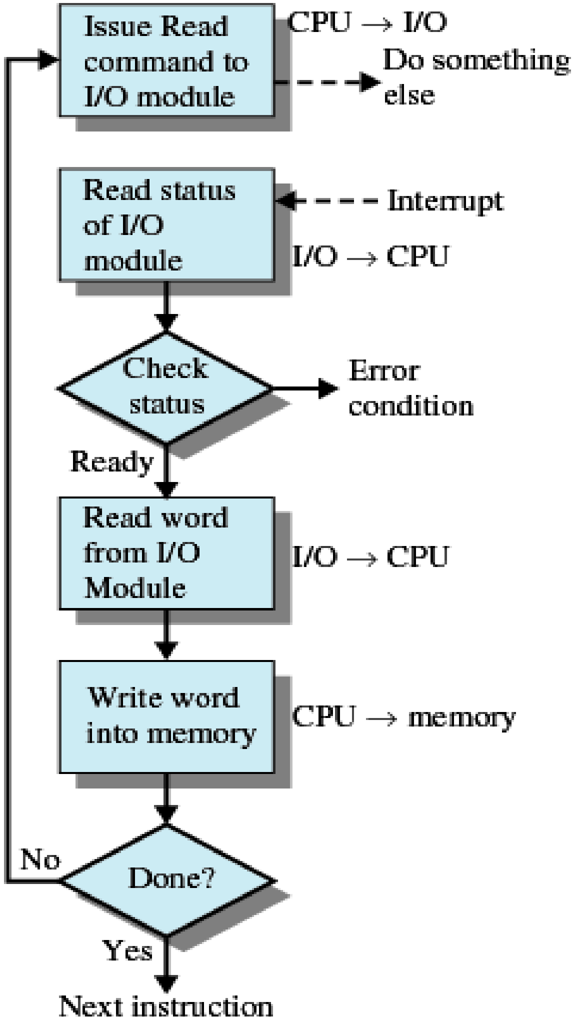
- I/O module performs the action, not the processor
- Sets appropriate bits in the I/O status register
- No interrupts occur
- Processor checks status until operation is complete



Programmed I/O

Interrupt-Driven I/O

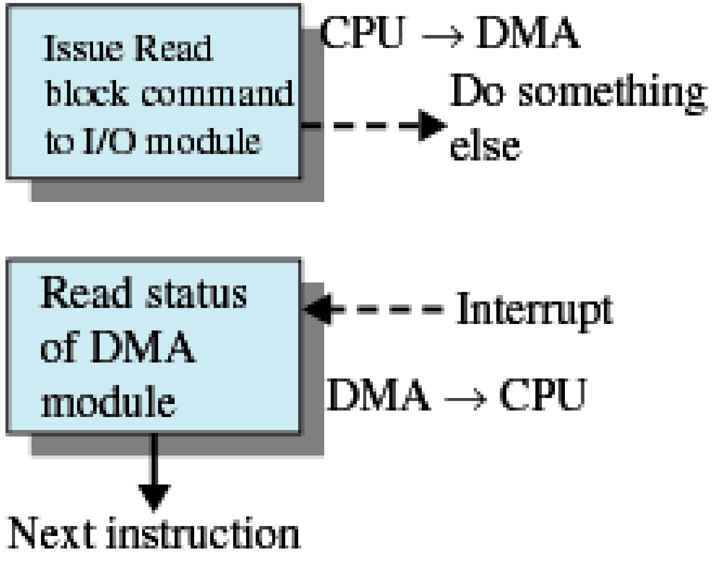
- Processor is interrupted when I/O module ready to exchange data
- Processor saves context of program executing and begins executing interrupt-handler
- No needless waiting
- Consumes a lot of processor time because every word read or written passes through the processor



Interrupt-driven I/O

Direct Memory Access

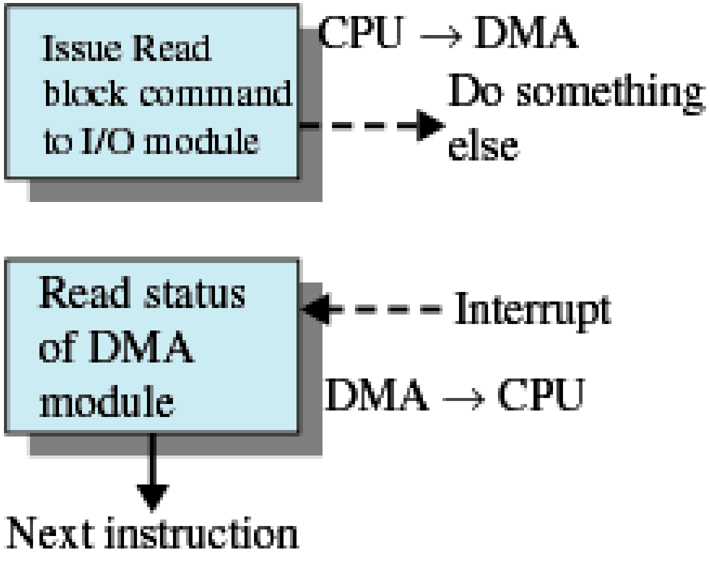
- Transfers a block of data directly to or from memory
- An interrupt is sent when the transfer is complete
- Processor continues with other work


Issue Read block command to I/O module
CPU → DMA
Do something else
Read status of DMA module
Interrupt
DMA → CPU
Next instruction

Direct memory access

Direct Memory Access

- Transfers a block of data directly to or from memory
- An interrupt is sent when the transfer is complete
- Processor continues with other work


Issue Read block command to I/O module
CPU → DMA
Do something else
Read status of DMA module
Interrupt
DMA → CPU
Next instruction

Direct memory access