Operating Systems

2. Computer Organization

What is an Operating System?

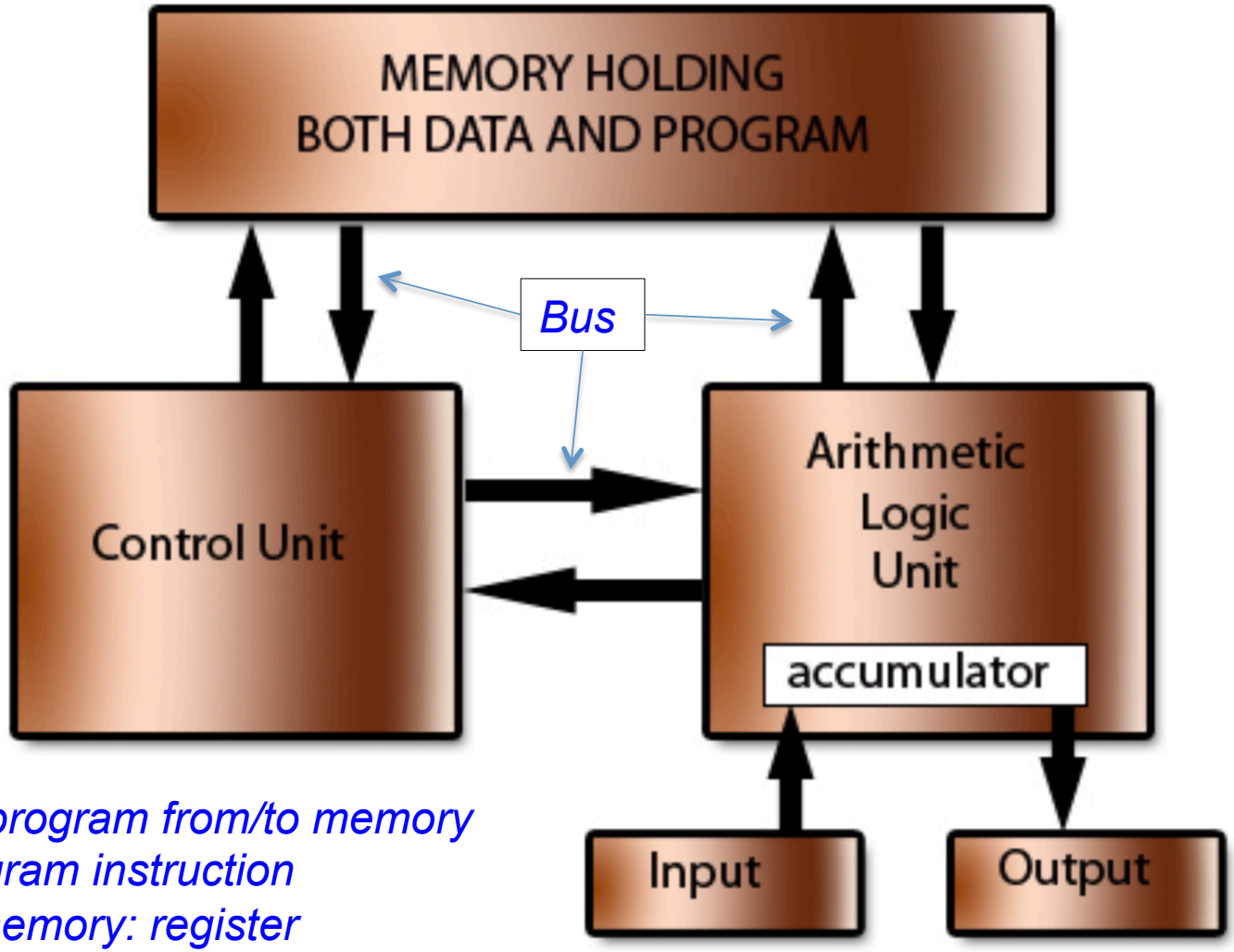
- Help the *user* use computer easily (Interface)
- Help the *software* use computer resource
- Help the *programmer* develop software (API)

- Operating system knows well the computer works: *Computer Architecture*

Von Neumann Machine

- Pre-Von-Neumann Machine
  - A computer can do only single computation
  - Program = HW, Data in memory
- Von-Neumann Machine
  - General purpose computer
  - *Program in memory*, Data in memory
  - Called *stored-program* architecture
  - Good: re-program easily

Von Neumann Architecture



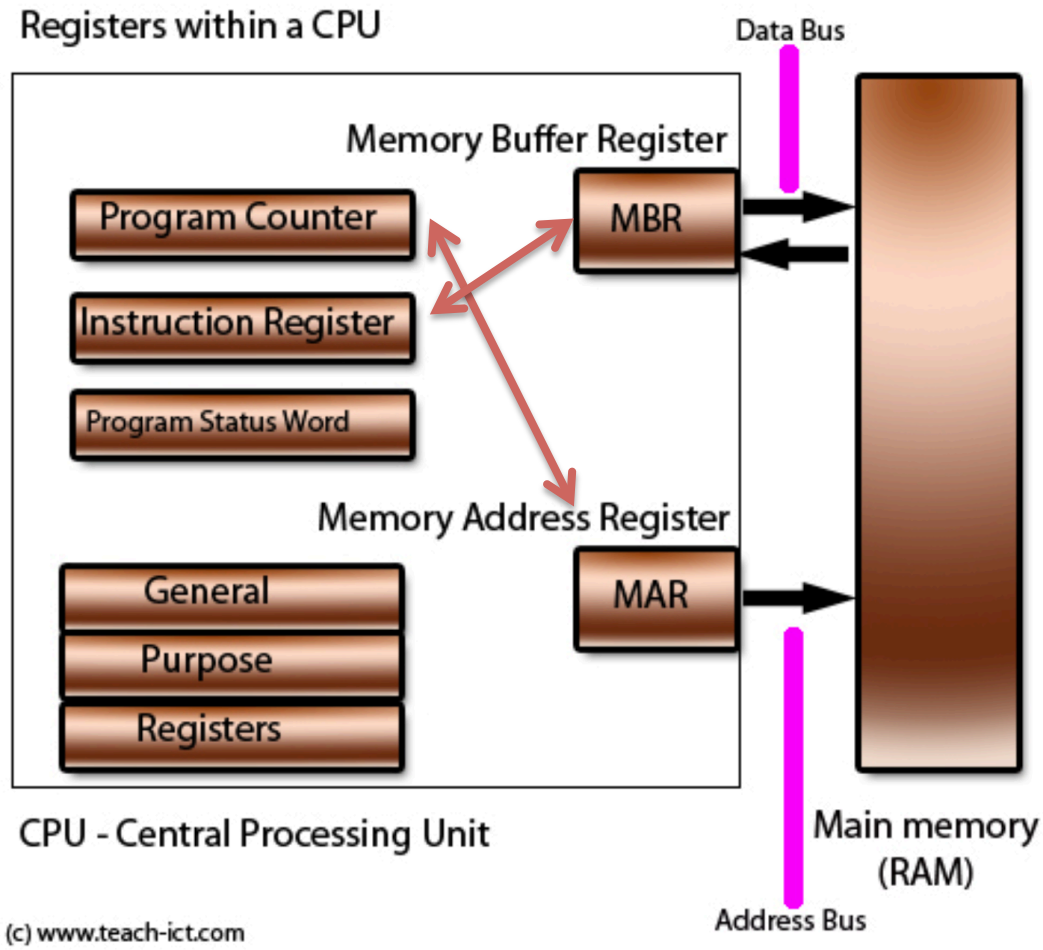
- *Moves data/program from/to memory*
- *Execute program instruction*
- *Temporary memory: register*

Registers

PC: Address of next instruction

IR: Current Instruction

PSW: Result (Condition, Logic, Error)



MBR: Buffer for data/prog to fetch/store

MBR: Address to fetch/store d ata/prog.

Problem 1 of Von-Neumann

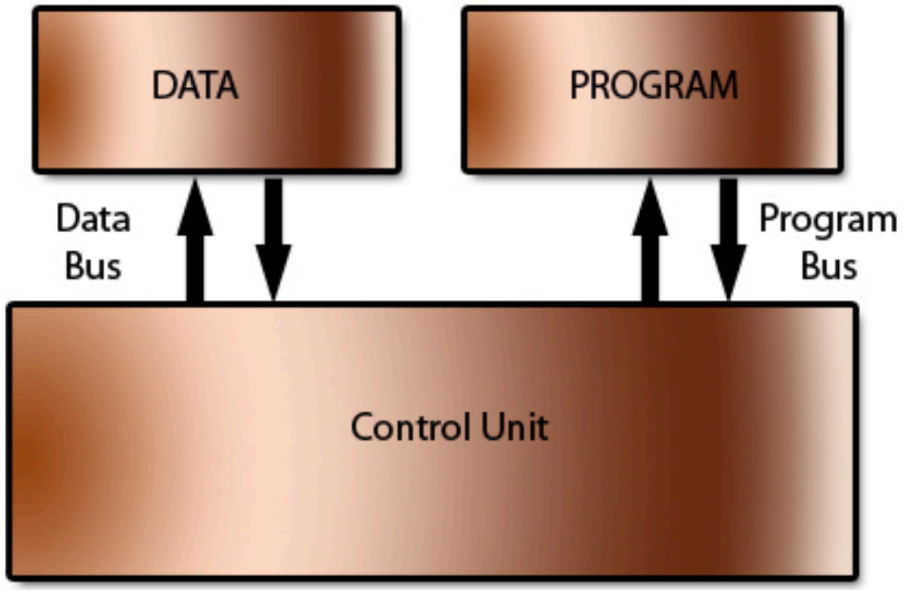
- Problem: Von Neumann Bottleneck = Bus
  - Too much to move over bus: slow
- Solution: *Cache*
  - *Temporary fast memory with recent data/prog*
  - *Code optimization*

Problem 2 of Von-Neumann

- Problem 2: Prog. & data in same memory
  - A program can mess-up instructions in memory
- Solution: *Harvard Architecture*
  - *Separate data and program memory*
  - *Memory restriction: Prevent accessing memory outside data*

Problem 3 of Von-Neumann

- Problem 3: Prog. & data over the same bus
  - Different fetch rate, slow bus
- Solution: *Harvard Architecture*
  - *Separate data and program bus*



Remarks on VNA

- Understand Von-Neumann Architecture
- Understand problems and solutions
- How CPU executes a program?
  - Bring to memory & run
  - *Sequential execution* of instructions
- How a computer can do many things simultaneously?
  - *Interrupts*

Basic Instruction Cycle

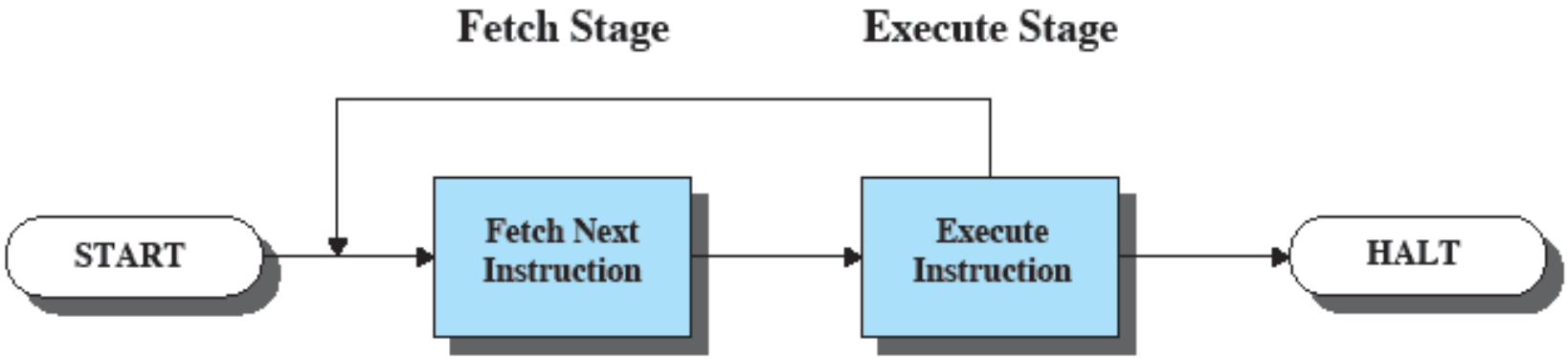
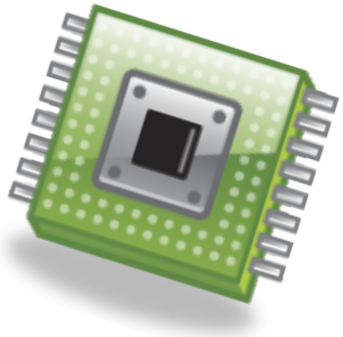


Figure 1.2 Basic Instruction Cycle

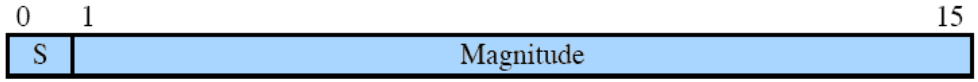
Instruction Fetch and Execute



- The processor fetches the instruction from memory into Instruction Register (IR)
- Program counter (PC) holds address of the instruction to be fetched next
  - PC is incremented after each fetch
- Processor interprets the instruction and performs required action:
  - Processor-memory
  - Processor-I/O
  - Data processing
  - Control

Program Execution

(a) Instruction format



(b) Integer format

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

(c) Internal CPU registers

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

(d) Partial list of opcodes

Figure 1.3 Characteristics of a Hypothetical Machine

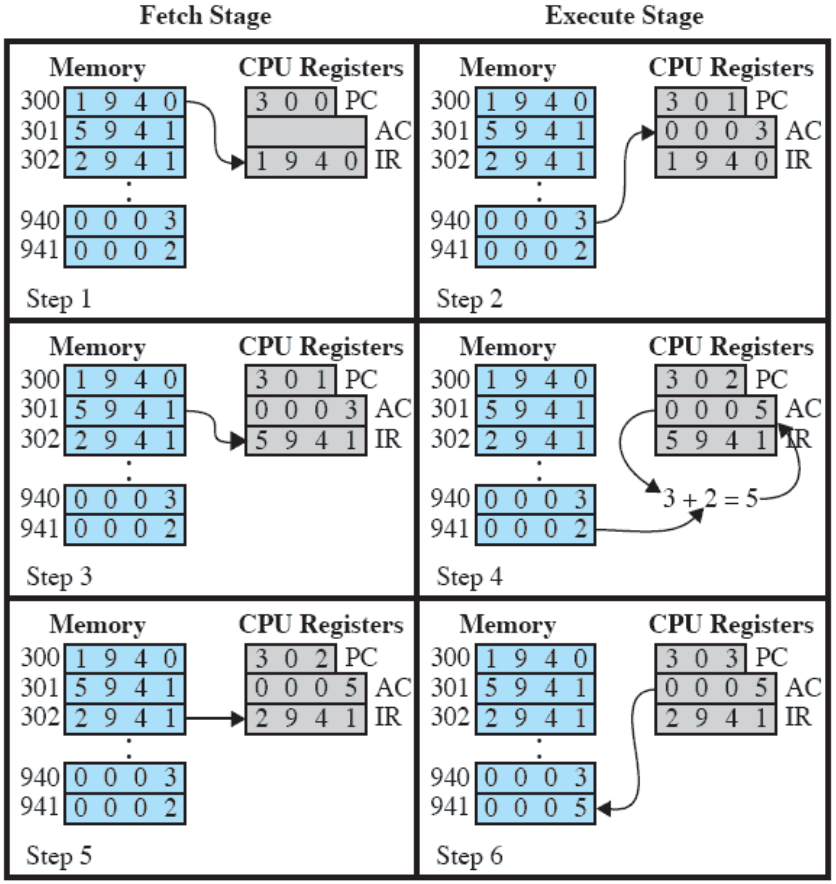


Figure 1.4 Example of Program Execution (contents of memory and registers in hexadecimal)

Interrupts

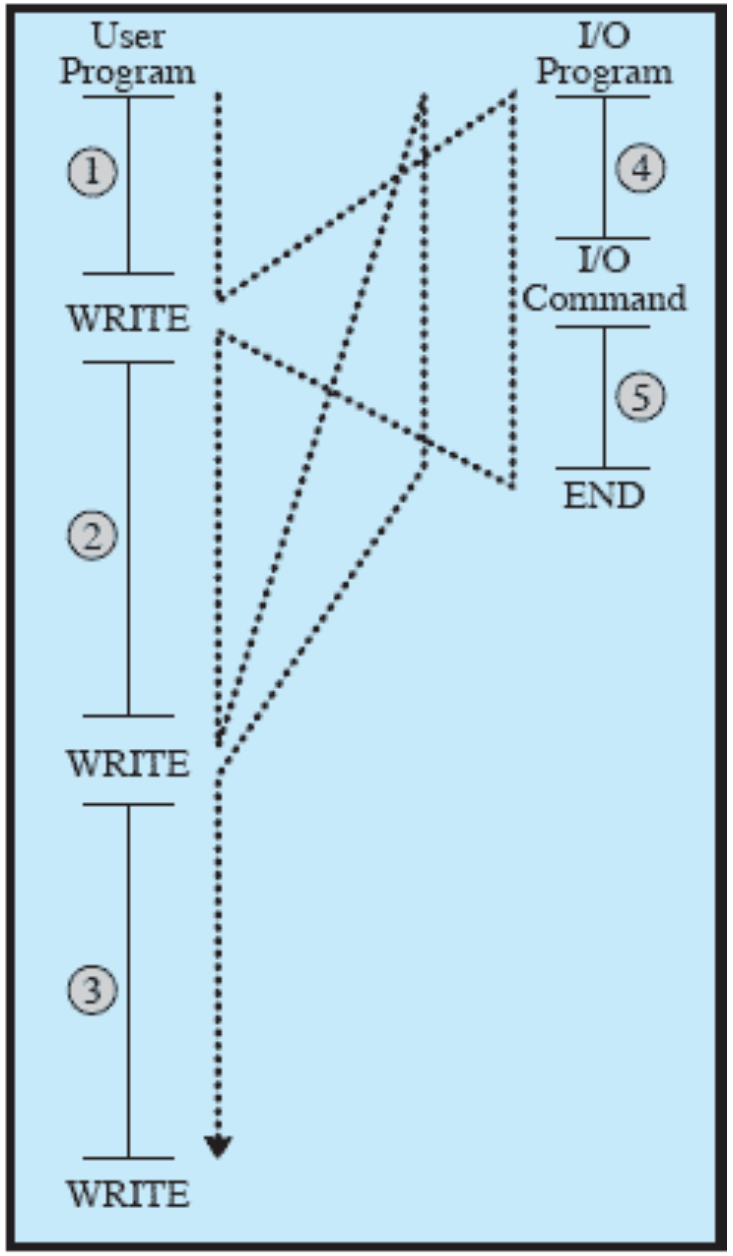
- Interrupt the normal sequencing of the processor
- Provided to improve processor utilization
    - most I/O devices are slower than the processor
    - processor must pause to wait for device
    - wasteful use of the processor

Common Classes of Interrupts

Table 1.1 Classes of Interrupts

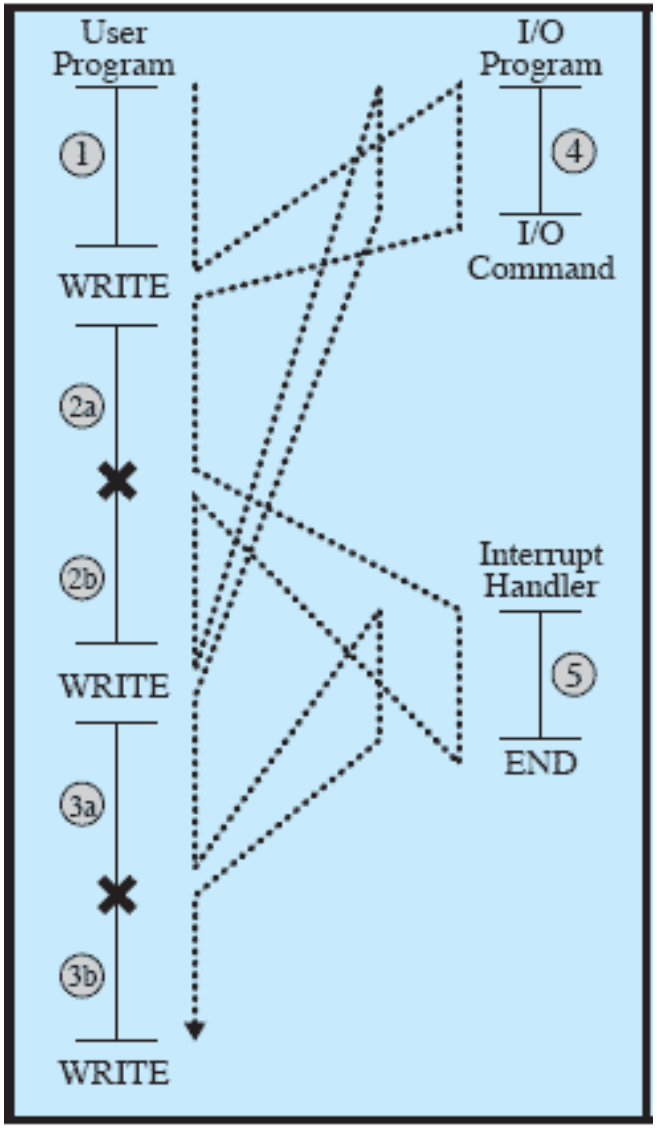
Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows 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.

Flow of Control Without Interrupts



(a) No interrupts

Interrupts: Short I/O Wait



(b) Interrupts; short I/O wait

Transfer of Control via Interrupts

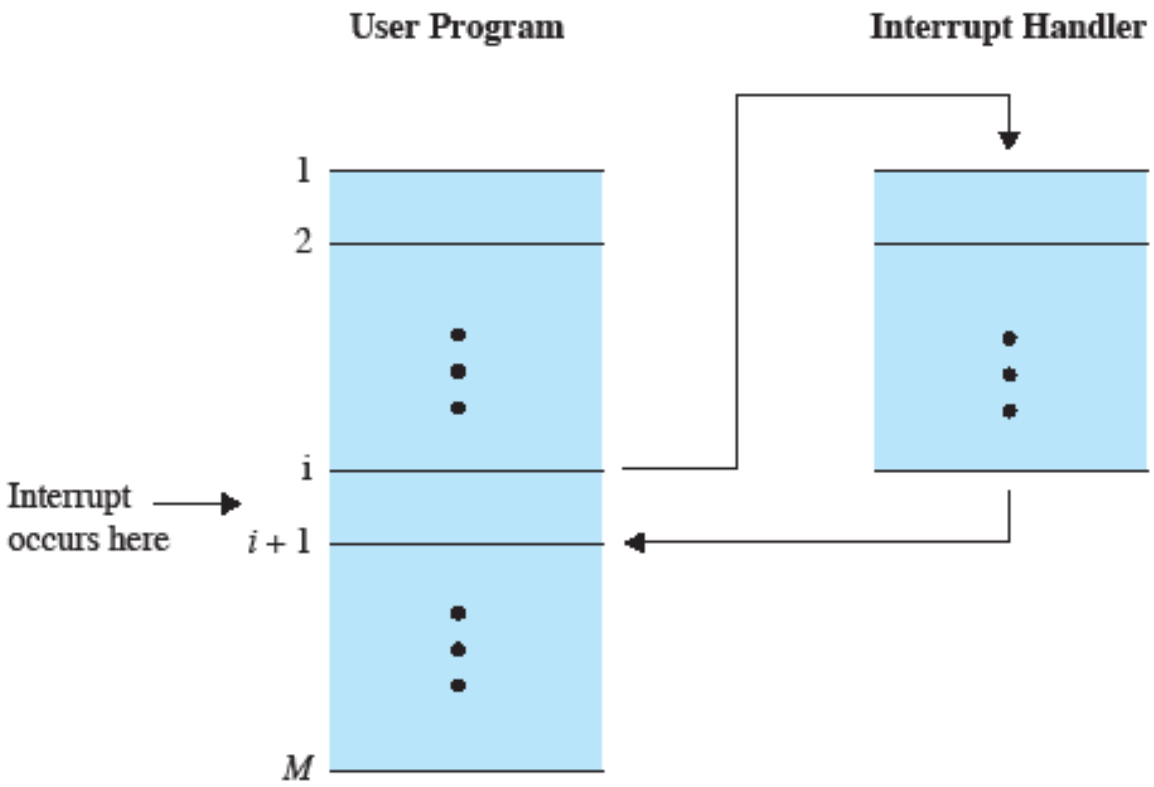
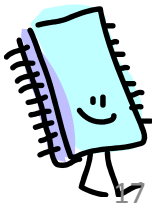


Figure 1.6 Transfer of Control via Interrupts



Instruction Cycle With Interrupts

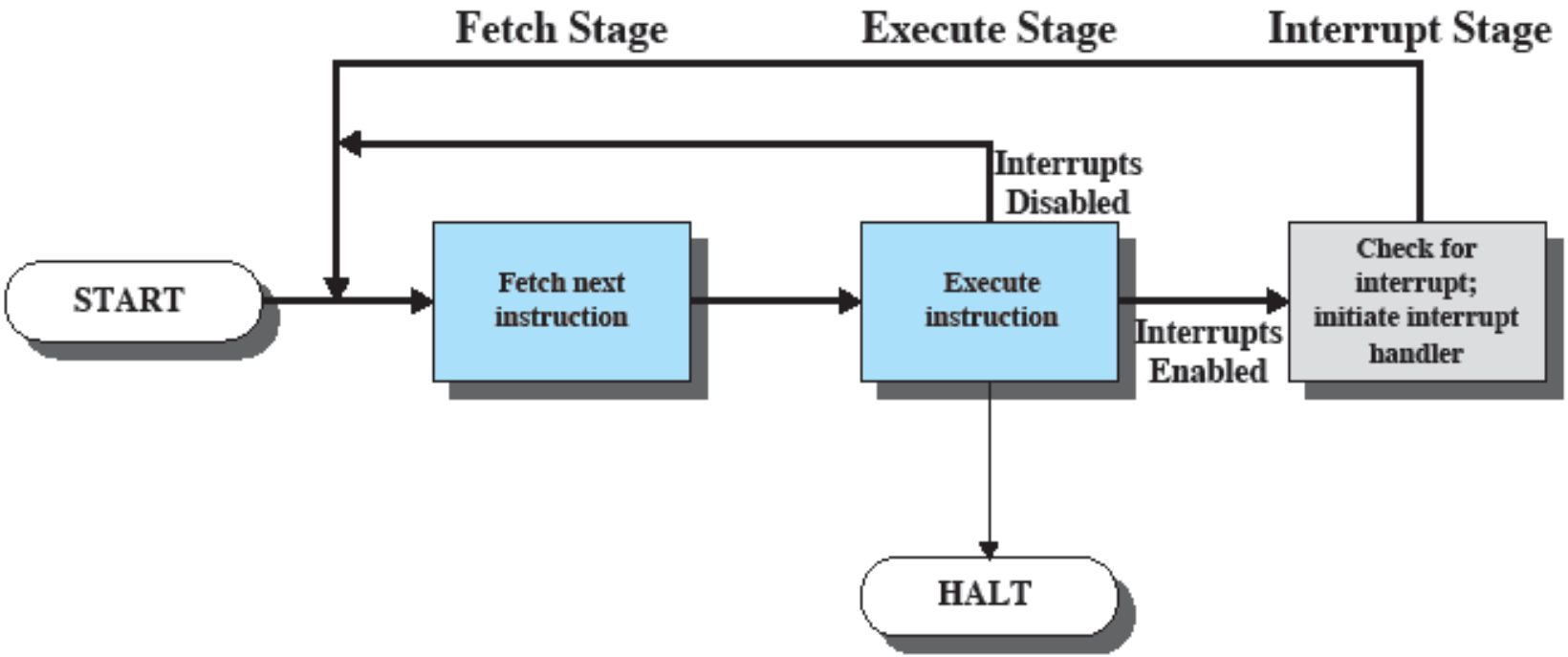
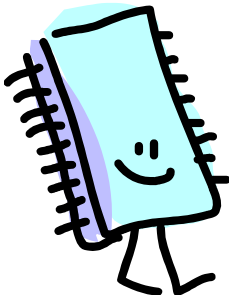


Figure 1.7 Instruction Cycle with Interrupts

MEMORY HIERARCHY

Memory

- Major constraints in memory
  - amount
  - speed
  - expense
- Memory must be able to keep up with the processor
- Cost of memory must be reasonable in relationship to the other components



The Memory Hierarchy

- Going down the hierarchy:
  - decreasing cost per bit
  - increasing capacity
  - increasing access time
  - decreasing frequency of access to the memory by the processor

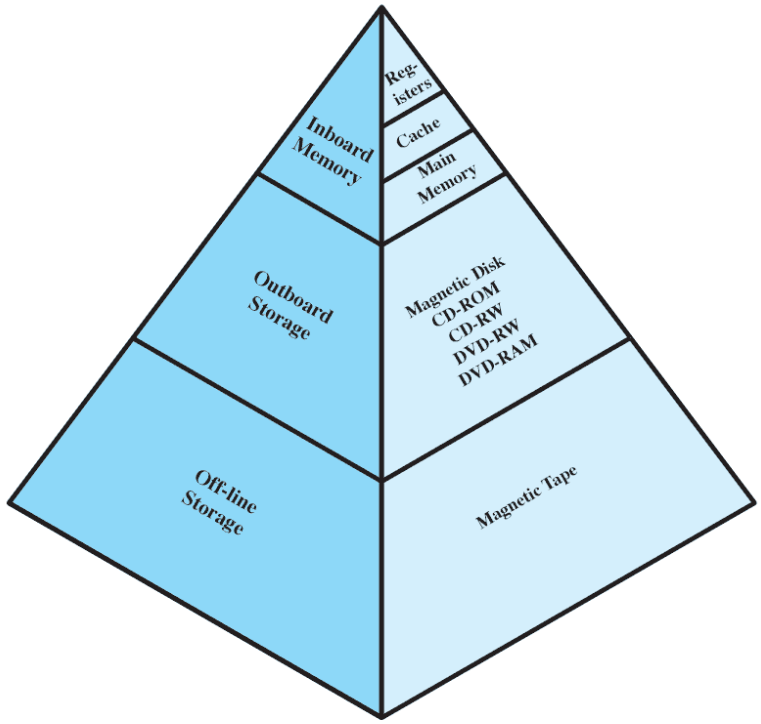
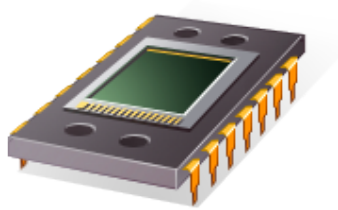


Figure 1.14 The Memory Hierarchy

Principle of Locality

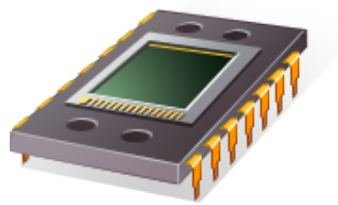
- **Fact**:
  - Memory references by the processor tend to cluster in **time** and **space**
- **How to exploit it**:
  - Data is organized so that the percentage of accesses to each successively lower level is substantially less than that of the level above
  - Can be applied across more than two levels of memory

Cache Memory



- Invisible to the processors, programmer, OS
- Interacts with other memory management hardware
- Reasons for its existence:
  - Processor must access memory at least once per instruction cycle
  - Processor execution is limited by memory cycle time
  - Exploit the principle of locality with a small, fast memory

Cache Principles

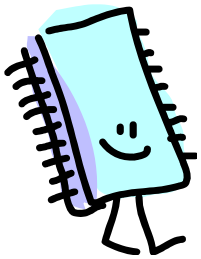


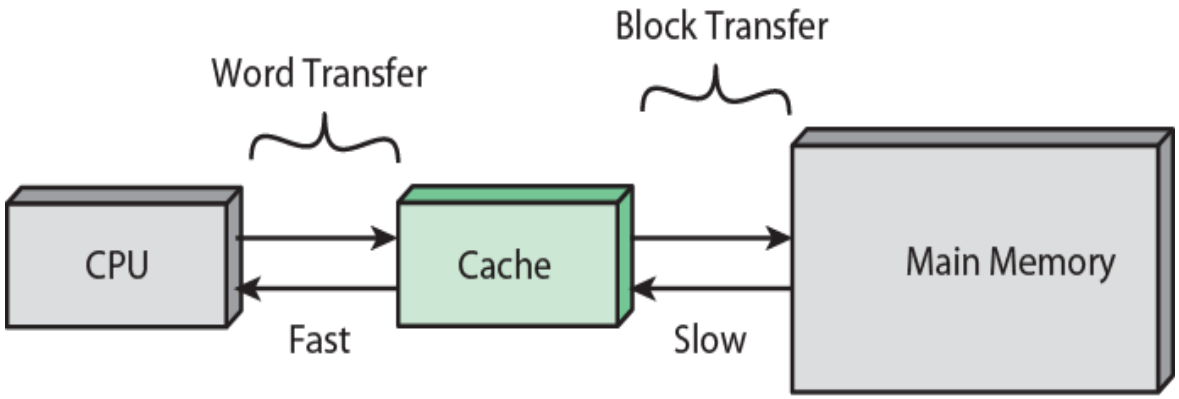
- Contains a copy of a portion of main memory
- Processor first checks cache

– If not found, a block of memory is read into cache

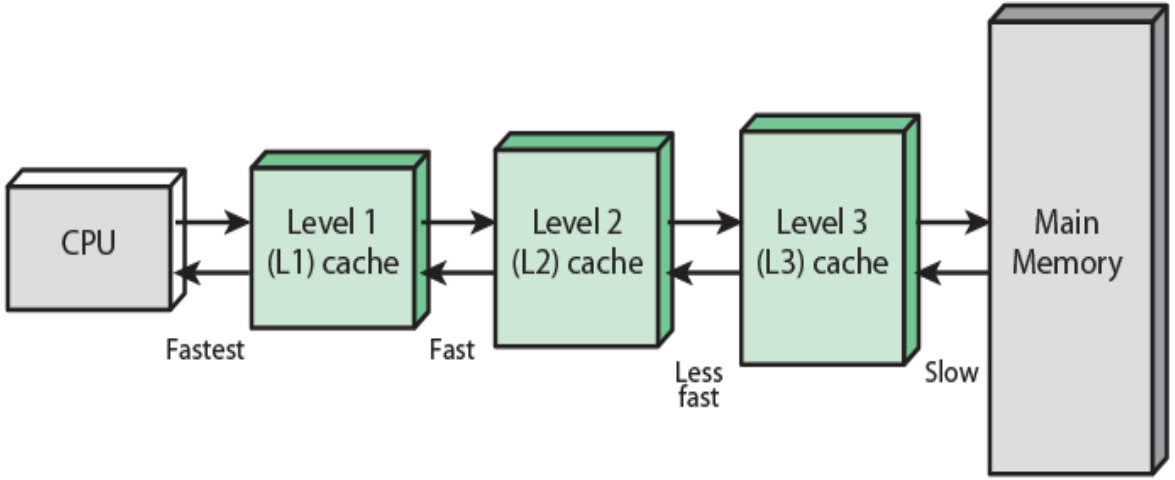
- Because of locality of reference, it is likely that many of the future memory references will be to other bytes in the block

Cache and Main Memory





(a) Single cache



(b) Three-level cache organization

Cache/Main-Memory Structure

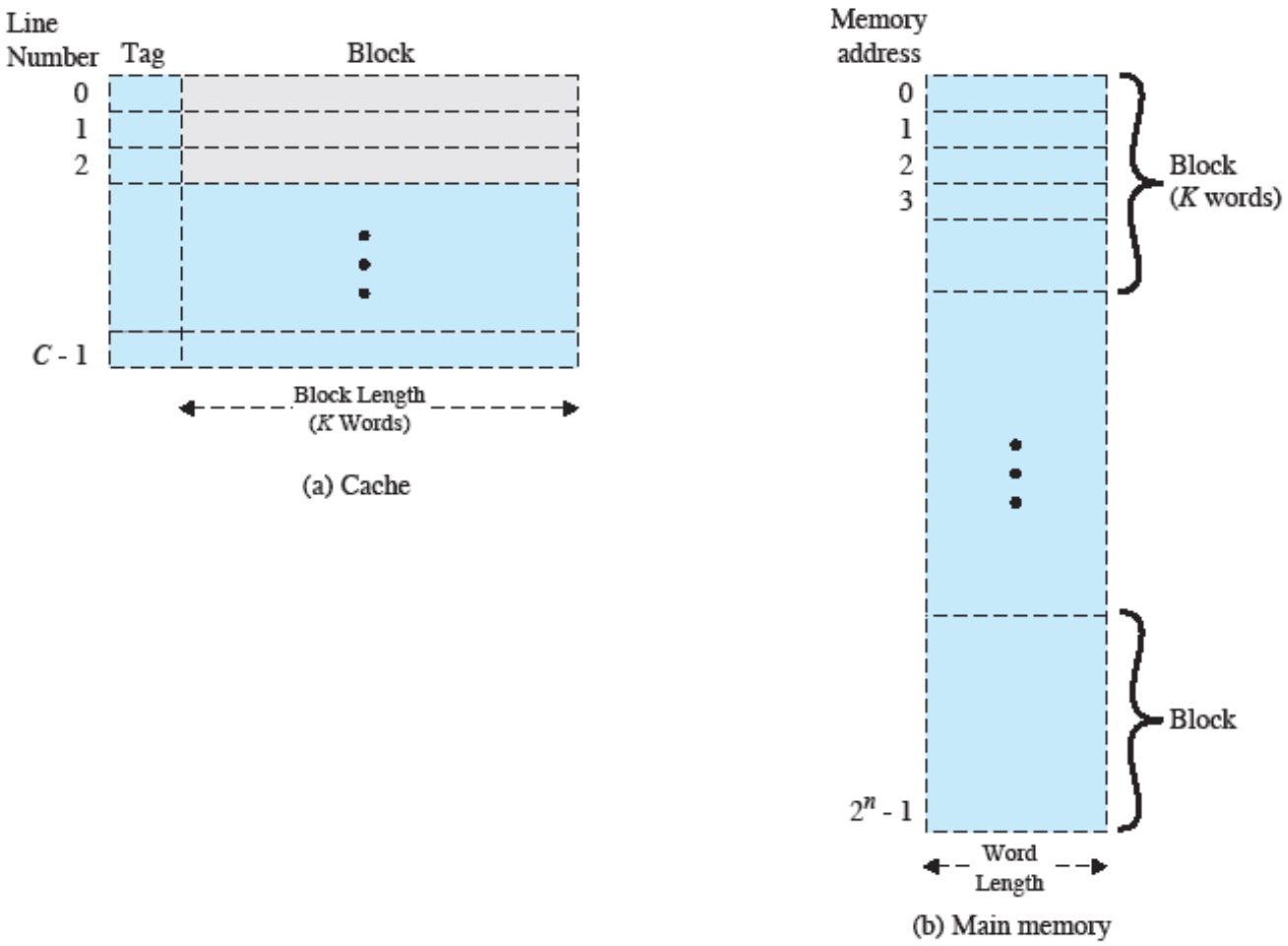
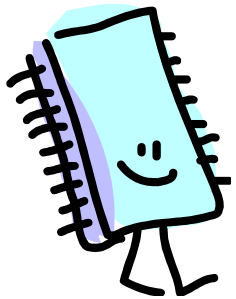


Figure 1.17 Cache/Main-Memory Structure

Cache Read Operation



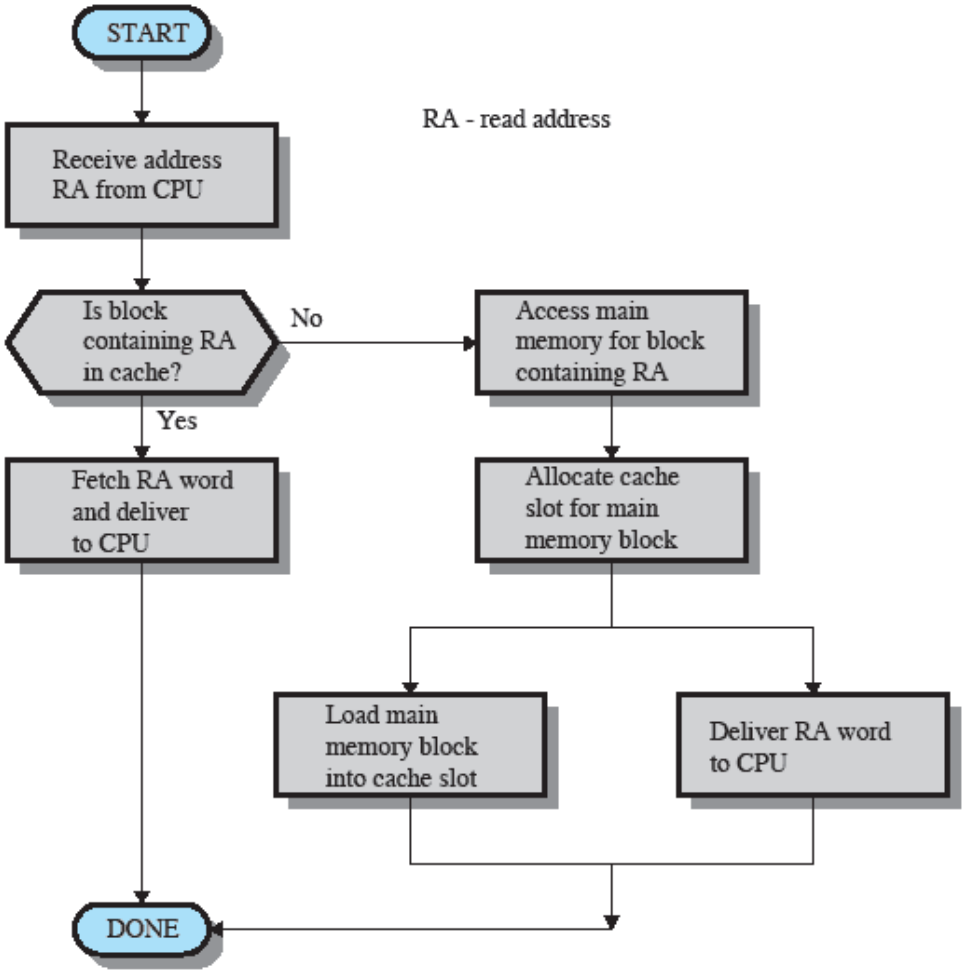
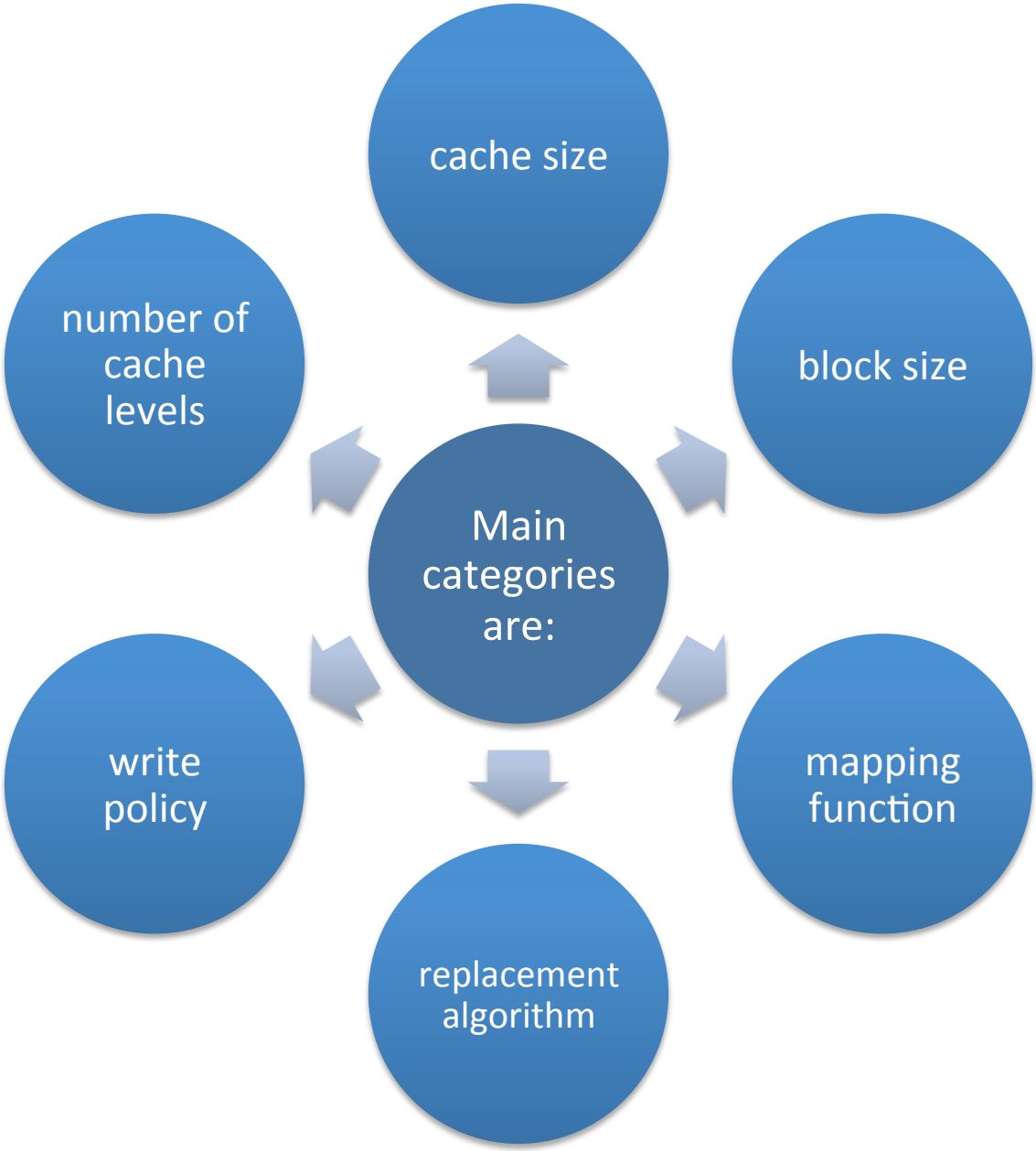


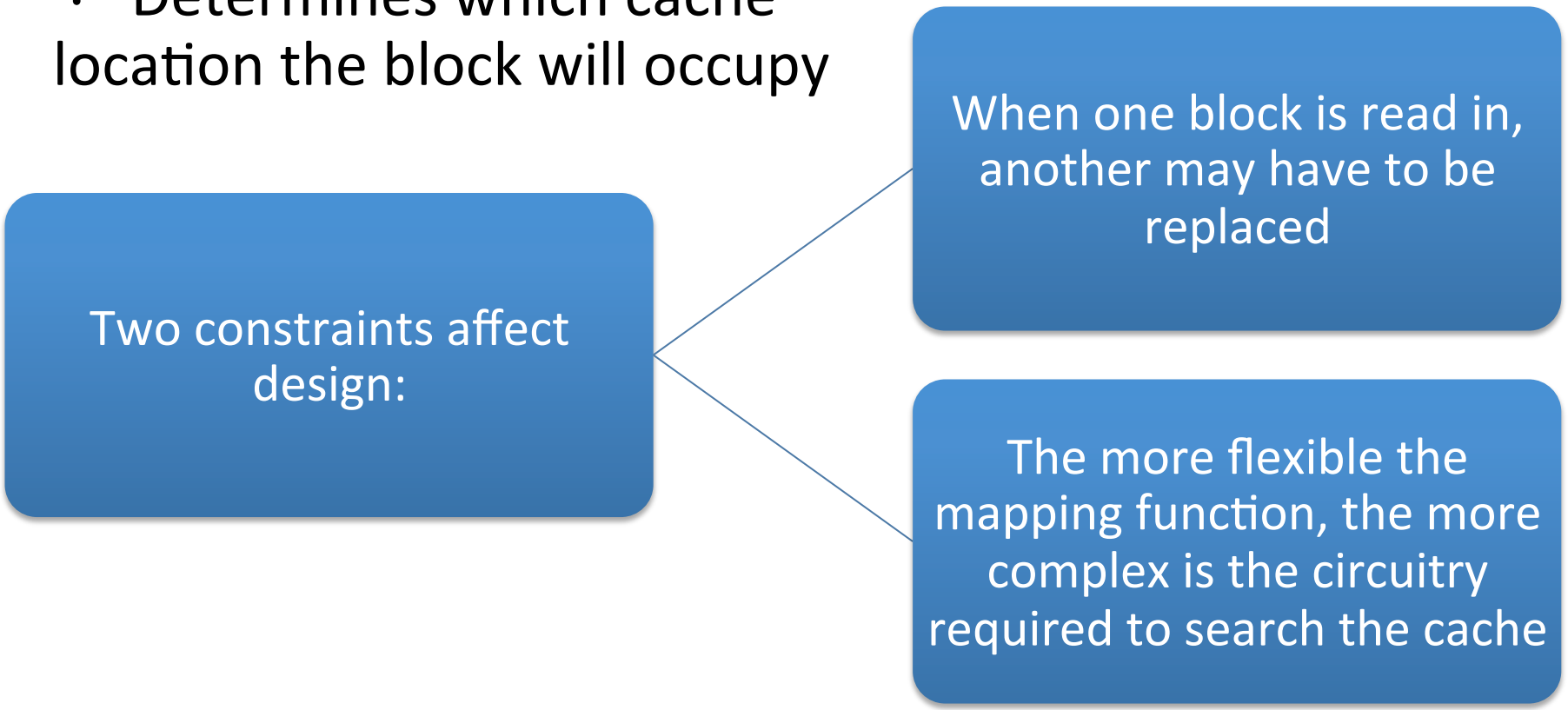
Figure 1.18 Cache Read Operation

CACHE DESIGN



Mapping Function

* Determines which cache location the block will occupy



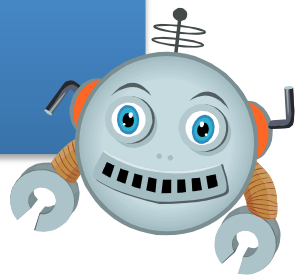
Replacement Algorithm

- Chooses which block to replace when a new block is to be loaded into the cache

- Least Recently Used (LRU) Algorithm
  - effective strategy is to replace a block that has been in the cache the longest with no references to it
  - hardware mechanisms are needed to identify the least recently used block

Write Policy

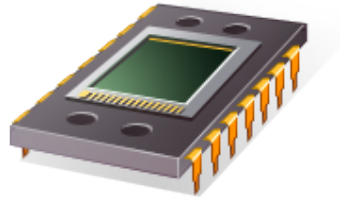
Dictates when the memory write operation takes place



- can occur every time the block is updated
- can occur when the block is replaced
  - minimizes write operations
  - leaves main memory in an obsolete state

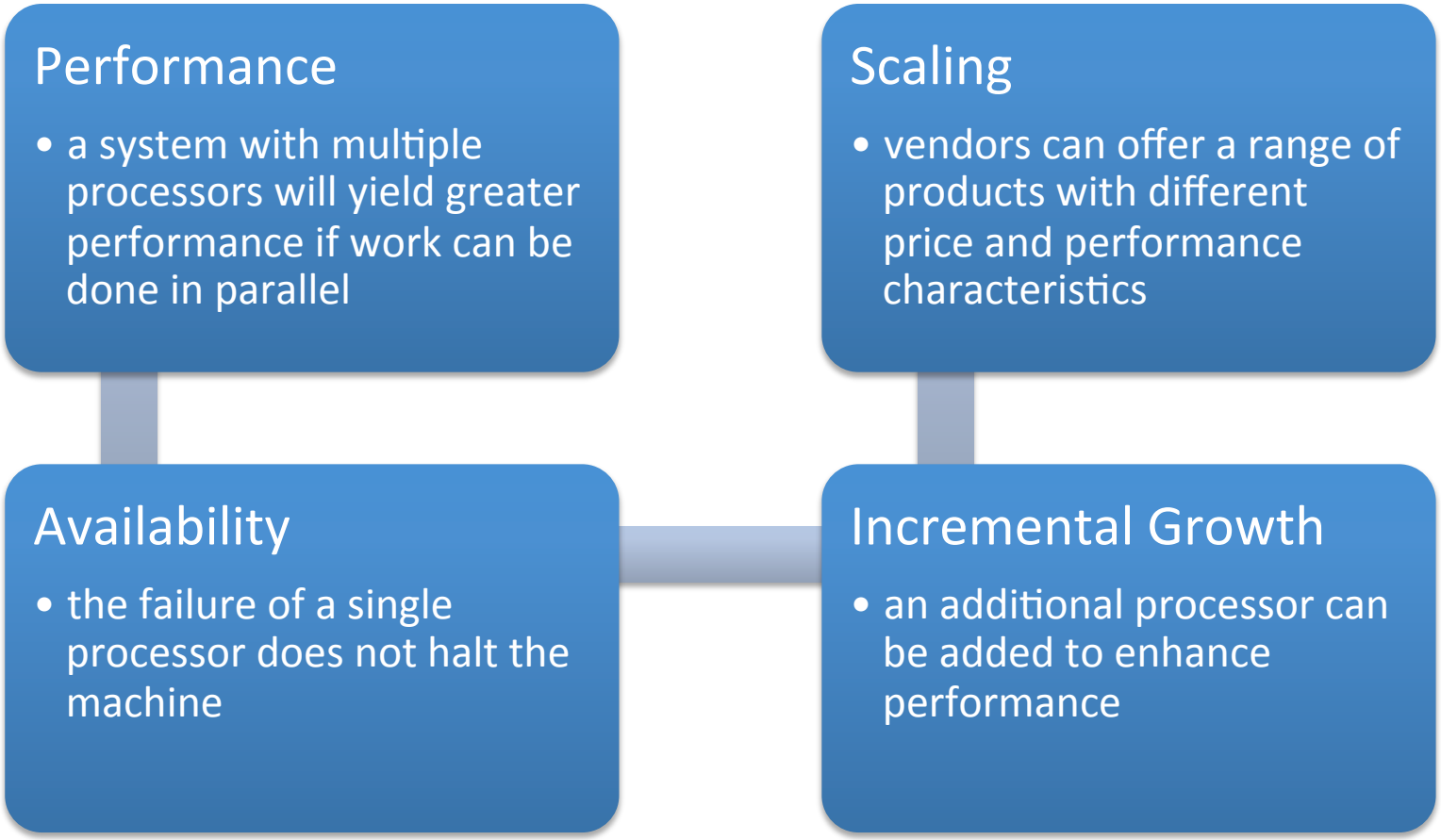
SMP AND MULTICORE

Symmetric Multiprocessors (SMP)

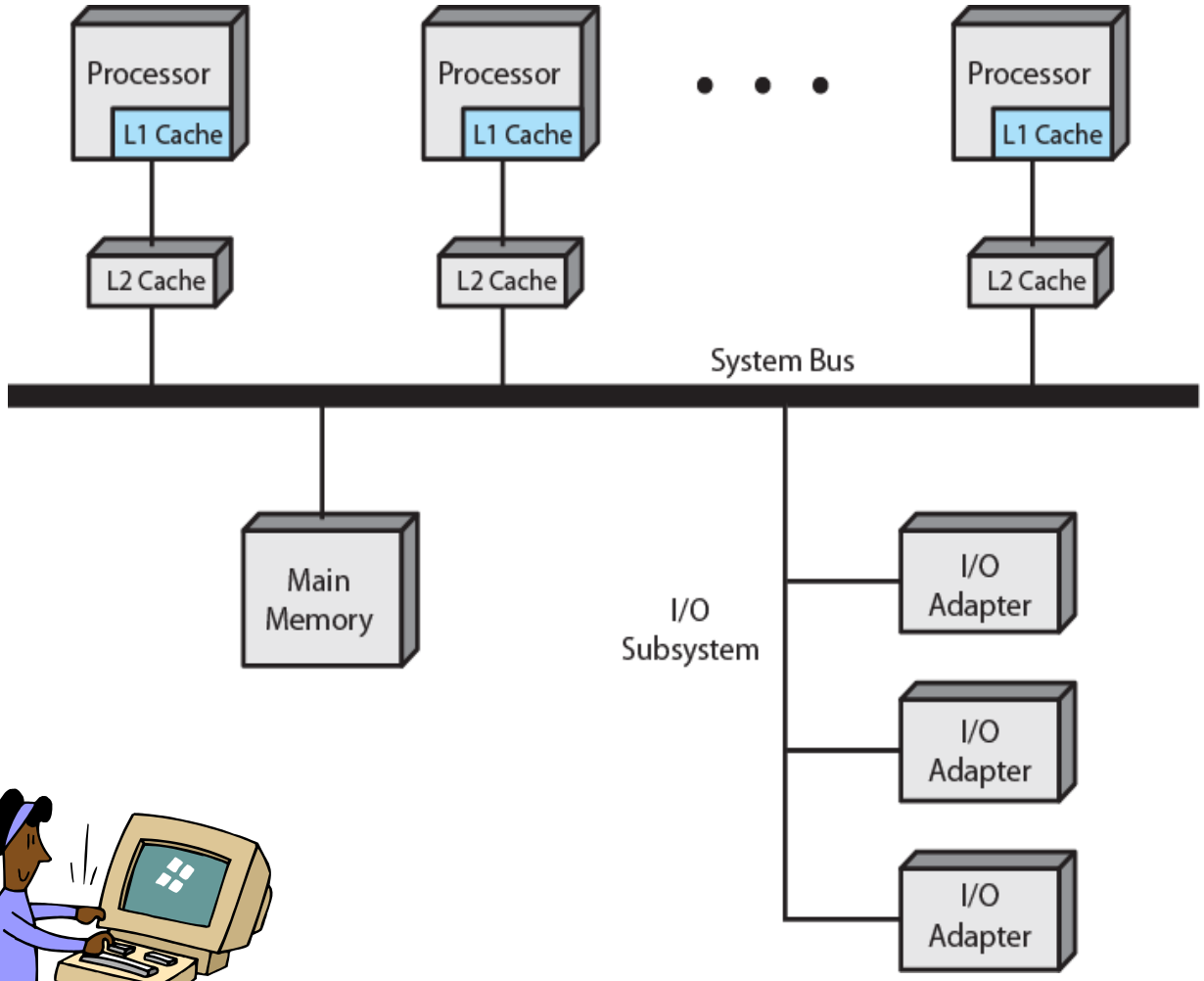


- A stand-alone computer system with the following characteristics:
  - two or more similar processors of comparable capability
  - processors share the same main memory and are interconnected by a bus or other internal connection scheme
  - processors share access to I/O devices
  - all processors can perform the same functions
  - the system is controlled by an integrated operating system that provides interaction between processors and their programs at the job, task, file, and data element levels

SMP Advantages



SMP Organization



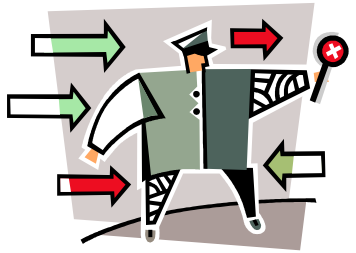
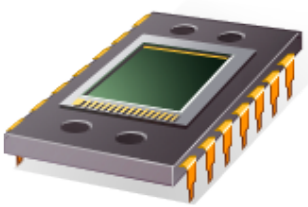
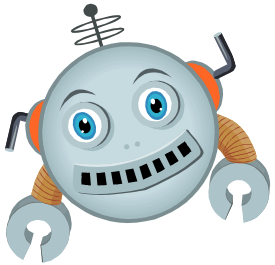


Figure 1.19 Symmetric Multiprocessor Organization



Multicore Computer

- Also known as a chip multiprocessor
- Combines two or more processors (cores) on a single piece of silicon (die)
  - each core consists of all of the components of an independent processor
- In addition, multicore chips also include L2 cache and in some cases L3 cache



Intel Core i7

Supports two forms of external communications to other chips:

DDR3 Memory Controller

- brings the memory controller for the DDR (double data rate) main memory onto the chip
- with the memory controller on the chip the Front Side Bus is eliminated

QuickPath Interconnect (QPI)

- enables high-speed communications among connected processor chips

Intel Core i7

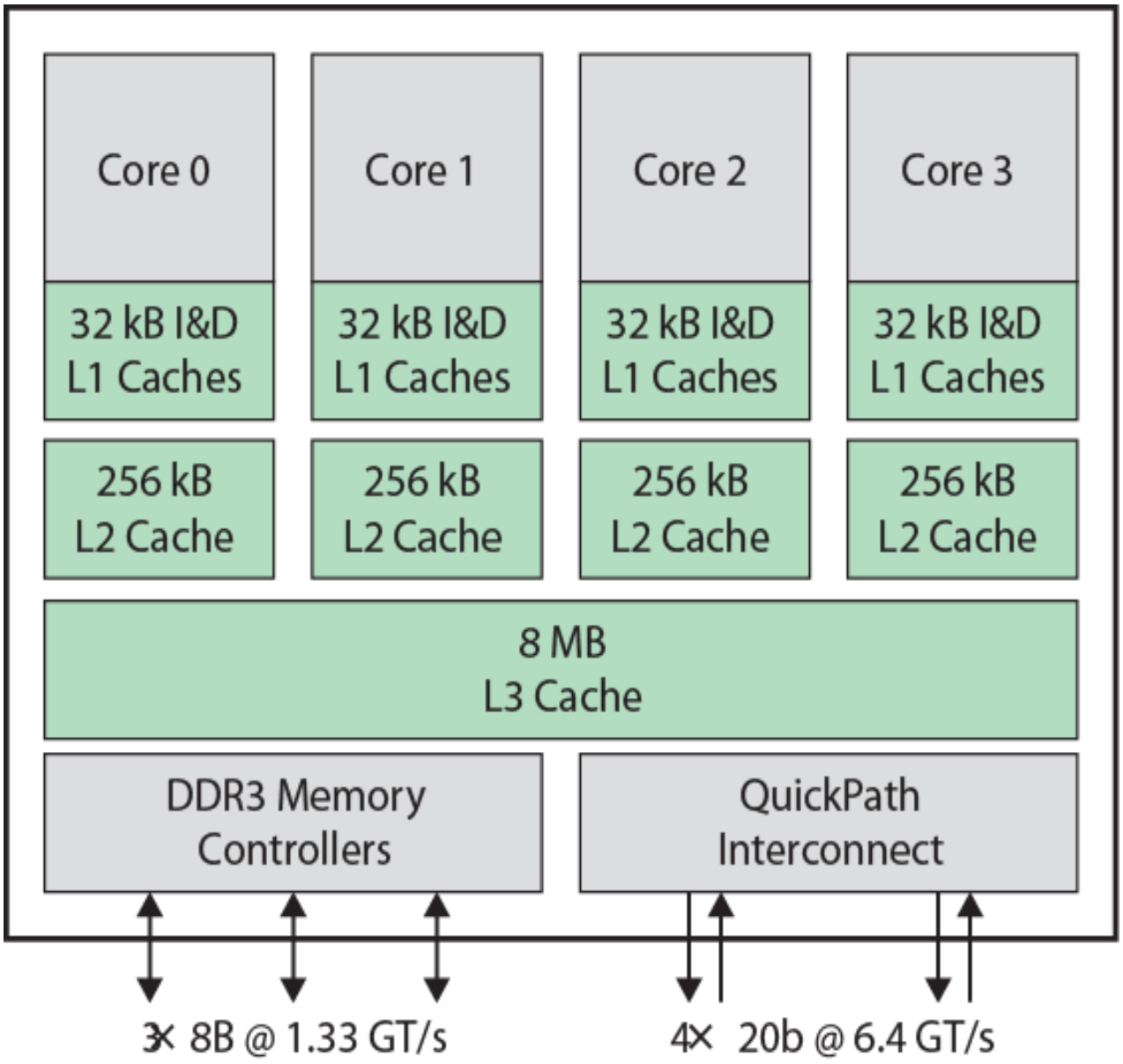


Figure 1.20 Intel Corei7 Block Diagram

DIRECT MEMORY ACCESS (DMA)

I/O Techniques

When the processor encounters an instruction related to I/O, it executes that instruction by issuing a command to the appropriate I/O module

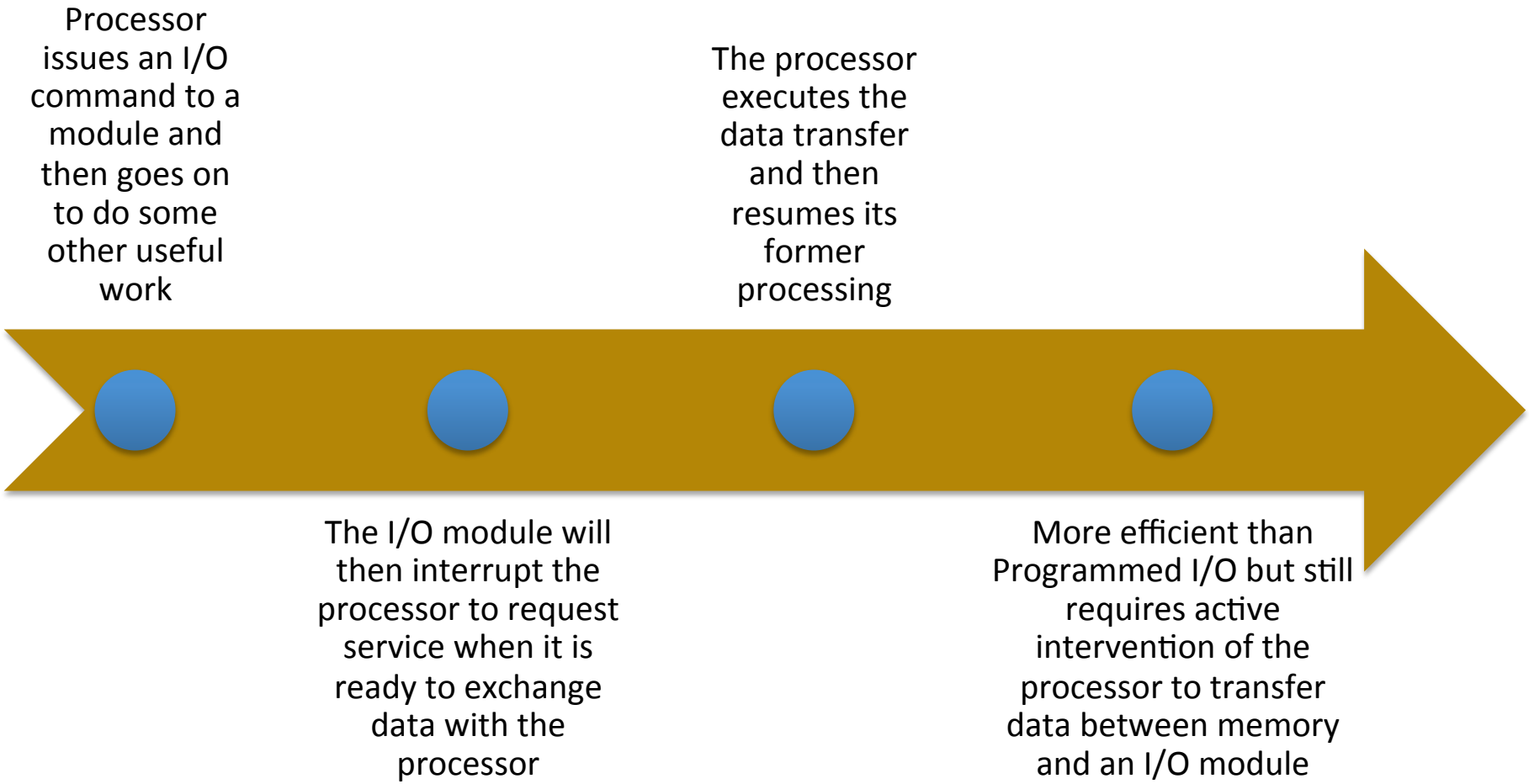
Three techniques are possible for I/O operations:

- Programmed I/O
- Interrupt-Driven I/O
- Direct Memory Access (DMA)

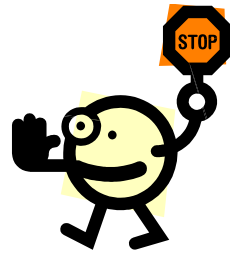
Programmed I/O

- The I/O module performs the requested action then sets the appropriate bits in the I/O status register
- The processor periodically checks the status of the I/O module until it determines the instruction is complete
- With programmed I/O the performance level of the entire system is severely degraded

Interrupt-Driven I/O



Interrupt-Driven I/O Drawbacks



- Transfer rate is limited by the speed with which the processor can test and service a device
- The processor is tied up in managing an I/O transfer
    - a number of instructions must be executed for each I/O transfer

Direct Memory Access (DMA)

Performed by a separate module on the system bus or incorporated into an I/O module

When the processor wishes to read or write data it issues a command to the DMA module containing:

- whether a read or write is requested
- the address of the I/O device involved
- the starting location in memory to read/write
- the number of words to be read/written

Direct Memory Access

- Transfers the entire block of data directly to and from memory without going through the processor
  - processor is involved only at the beginning and end of the transfer
  - processor executes more slowly during a transfer when processor access to the bus is required
- More efficient than interrupt-driven or programmed I/O

Summary

- Basic Elements
  - processor, main memory, I/O modules, system bus
  - GPUs, SIMD, DSPs, SoC
  - Instruction execution
    - » processor-memory, processor-I/O, data processing, control
  - Interrupt/Interrupt Processing
  - Memory Hierarchy
  - Cache/cache principles and designs
  - Multiprocessor/multicore