**UNIT 2. COMPUTER EVOLUTION**

**Moore’s law** \_\_ consequences:

1. The cost of a chip has **remained virtually unchanged** during this period of rapid growth in density. This means that the cost of computer logic and memory circuitry has fallen at a dramatic rate.

2. Because logic and memory elements are **placed closer** togetheron more densely packed chips, the electrical path length is shortened, **increasing operating speed.**

3. The **computer becomes smaller**, making it more convenient to place in a variety of environments.

4. There is a **reduction in power and cooling requirements**.

5. The interconnections on the integrated circuit are much more **reliable** than solder connections (kết nối hàn). With more circuitry on each chip, **there are fewer interchip connections.**

**Microprocessors**

* 1971 Intel developed 4004
* First chip to contain all of the components of a CPU on a single chip
* **Birth of microprocessor**
* 1972 Intel developed 8008
* First 8-bit microprocessor
* 1974 **Intel developed 8080**
* **First general purpose microprocessor**
* Faster, has a richer instruction set, has a large addressing capability

|  |  |
| --- | --- |
| **Technique** | **Description** |
| Pipelining | Processor **moves data or instructions into a conceptual pipe** with all stages of the pipe processing simultaneously |
| Branch prediction | Processor looks ahead in the instruction code fetched from memory and **predicts which branches, or groups of instructions, are likely to be processed next** |
| Data flow analysis | Processor analyzes which instructions are dependent on each other’s results, or data, to create **an optimized schedule** of instructions |
| Speculative (suy đoán)  execution | Using branch prediction and data flow analysis, some processors speculatively execute instructions ahead of their actual appearance in the program execution, **holding the results in temporary location**s, keeping execution engines as busy as possible |

**Improvements in Chip**

* **Increase hardware speed of processor**
  + Fundamentally due to shrinking logic gate size
    - More gates, packed more tightly, increasing clock rate
    - Propagation time for signals reduced
* **Increase size and speed of caches**
  + Dedicating part of processor chip
    - Cache access times drop significantly
* **Change processor organization and architecture**
  + Increase effective speed of instruction execution
  + Parallelism

**Multicore CPU**: CPU has some cores running concurrently.

**MIC:** Many integrated core

* + The multicore and MIC strategy involves a homogeneous (same kind) collection of general purpose processors on a single chip

**GPGPU**: General Purpose Graphical Processing Unit

* + Core designed to perform **parallel operations** on graphics data
  + Traditionally found on a plug-in graphics card, it is used to encode and render 2D and 3D graphics as well as process video
  + Used as vector processors for a variety of applications that require repetitive computations

**Performance Assessment**

Factors

* **Clock Speed** and **Instructions per Second (IPS)**
* Instruction execution rate

Methods: Benchmarks

Some laws: Amdahl’s Law, Little’s Law

**System Clock**

**clock generator:** a hardware using crystal oscillator, Pulses are created by it

**clock rate / clock speed:** rate of pulses

**cycle time:** time between pulses

**clock cycle / clock tick:** One increment, or pulse, of the clock

**Unit:** cycles per second, Hertz (Hz)

**Instruction Execution Rate**

**Unit: MIPS** (millions of instructions per second)

**Unit: MFLOPs** (Floating-point performance is expressed as millions of floating-point operations per second)

**Benchmark:** A test used to measure hardware or software performance.

**UNIT 3. A Top-Level View of Computer**

**The von Neumann architecture - 3 key concepts:**

* **Data** and **instructions** are stored **in a single read-write memory**
* The contents of this memory are **addressable by location**
* Execution occurs in a **sequential** fashion (unless explicitly modified) from **one instruction to the next**

**Software**

* A sequence of codes or instructions
* Part of the **hardware interprets** each instruction and generates control signals
* Provide a **new sequence of codes** for each new program instead of rewiring the hardware

**Major components**

* **CPU**
* Instruction interpreter
* Module of general-purpose arithmetic and logic functions
* **I/O Components**
* Input module: Contains basic components for accepting data and instructions and converting them into an internal form of signals usable by the system
* Output module: Means of reporting results

**MAR:** Specifies the address in memory for the next read or write

**MBR:** Contains the data to be written into memory or receives the data read from memory

**I/OAR:** Specifies a particular I/O device

**I/OBR:** Used for the exchange of data between an I/O module and the CPU

**AC:** The processor contains a single **data register**, called an accumulator (AC)

AC = **temporary storage**

**Interrupts**

**Program:** conditions ~ results of ins execution: arithmetic overflow, div by 0, attempt to execute an illegal machine ins, reference outside memory space.

**Timer:** gen bytimer within the processor

**I/O:** gen by I/O controller, to signal normal completion of an operation, request service from processor/ to signal a variety of error condition

**Hardware failure:** gen by failure: power failure, mem parity error.

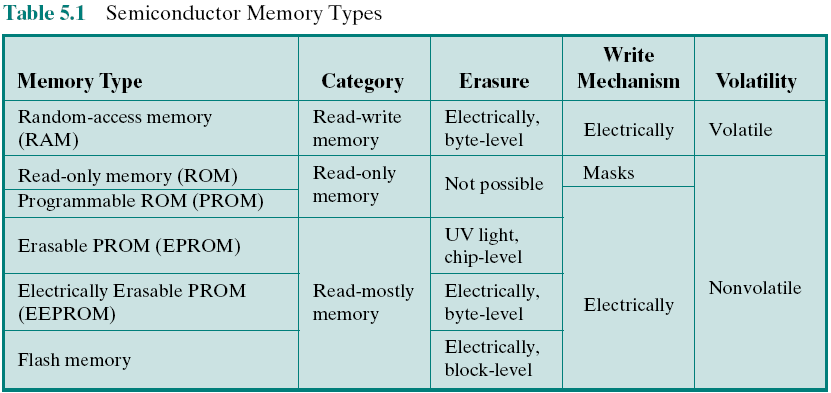
**Synchronous - đồng bộ -** At a time, only one device can uses the bus. The others must wait until the bus is idle.

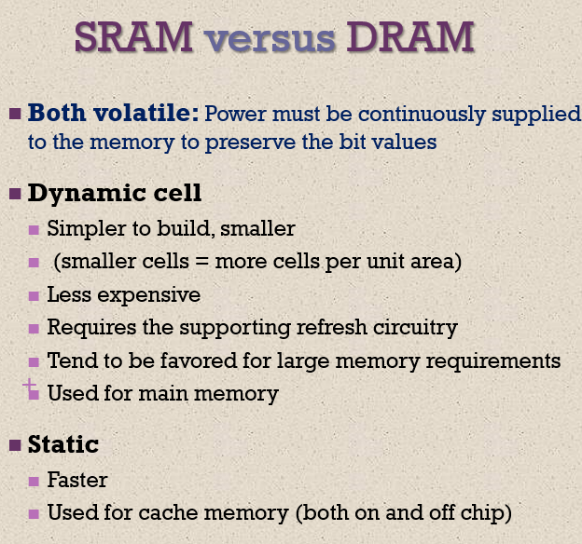
**Asynchronous - không đồng bộ -** At a time, some devices can use the bus concurrently

**UNIT 5. INTERNAL MEM**

**Semiconducor main mem**

Basic element of a semiconductor memory is the memory cell



**RAM**

***DRAM***

The term *dynamic* refers to tendency of the stored charge to leak away, even with power continuously applied

***SRAM***

Binary values are stored using traditional flip-flop logic gate configurations

Will hold its data as long as power is supplied to it

**ROM: permanent pattern of data, No power source**

* **EPROM**
  + Erasable programmable read-only memory
  + Erasure process can be performed repeatedly (exposure to UV)
  + More expensive than PROM but it has the advantage of the multiple update capability
* **EEPROM**
  + Electrically erasable programmable read-only memory
  + Can be written into at any time without erasing prior contents
  + Combines the advantage of non-volatility with the flexibility of being updatable in place
  + More expensive than EPROM
* **Flash Memory**
  + Intermediate between EPROM and EEPROM in both cost and functionality
  + Uses an electrical erasing technology, does not provide byte-level erasure
  + Microchip is organized so that a section of memory cells are erased in a single action or “flash”

**Advanced RAM**

***Synchronous DRAM (SDRAM):***

* Exchanges data with the processor synchronized to an external clock signal and running at the full speed of the processor/memory bus without imposing (long while) wait states

***Rambus Dynamic Random Access Memory (RDRAM)***

* Bus delivers address and control information using an **asynchronous** block-oriented protocol
* Developed by Rabus

***Double Data Rate SDRAM (DDR SDRAM):***

* SDRAM can only send data once per bus clock cycle
* Double-data-rate SDRAM can send data twice per clock cycle, once on the rising edge of the clock pulse and once on the falling edge
* Developed by the JEDEC Solid State Technology Association (Electronic Industries Alliance’s semiconductor-engineering-standardization body)

**Two generations:**

DDR2 increases the data transfer rate by increasing the operational frequency of the RAM chip and by increasing the prefetch buffer from 2 bits to 4 bits per chip. **The prefetch buffer is a memory cache located on the RAM chip.** The buffer enables the RAM chip to preposition bits to be placed on the data bus as rapidly as possible. DDR3, introduced in 2007, increases the prefetch buffer size to 8 bits.

DDR: 200 to 600 MHz;

DDR2: 400 to 1066 MHz;

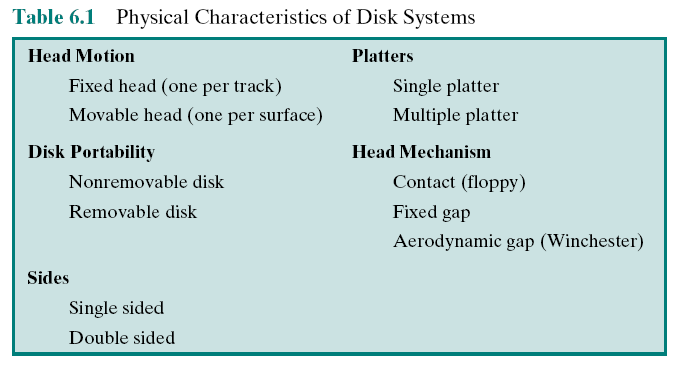
DDR3: 800 to 1600 MHz.

Cache DRAM (CDRAM): used in 2 ways

* used as a true cache, consisting of a number of 64-bit lines. The cache mode of the CDRAM is effective for ordinary random access to memory.
* used as a buffer to support the **serial access** of a block of data.

**UNIT 6. EXTERNAL MEM**

* Magnetic Disk
* Raid
* Solid State Drives
* Optical Memory
* Magnetic Tape

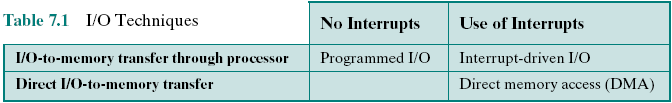


**UNIT 7. I/O**

**Why are devices not connected to system bus?**

* Impractical to **incorporate** the necessary logic within the processor to **control a range of devices**
* The data transfer rate of peripherals is **often much slower** than that of the memory or processor
* The data transfer rate of some peripherals **can be faster** than that of the memory or processor
* Peripherals often use **different data formats** and **word lengths** than the computer to which they are attached.

**Major functions or requirements for an I/O module**

* Control and timing: Coordinates the flow of traffic between internal resources and external devices.
* Processor communication: Involves command decoding, data, status reporting, address recognition.
* Device communication: Involves commands, status information, and data.
* Data buffering: Performs the needed buffering operation to balance device and memory speeds.
* Error detection: Detects and reports transmission errors..

**I/O commands:**

**Control:** used to activate a peripheral and tell it what to do

**Test:** used to test various status conditions associated with an I/O module and its peripherals

**Read:** causes the I/O module to obtain an item of data from the peripheral and place it in an internal buffer

**Write:** causes the I/O module to take an item of data from the data bus and subsequently transmit that data item to the peripheral

**Device Identification**

* Multiple interrupt lines
* Software poll (khảo sát phần mềm)
* Daisy chain (hardware poll, vectored)
* Bus arbitration (vectored) – Phân xử bus

**UNIT 8. OS support**

An OS is a program that **controls the execution of application programs** and acts as

an **interface between applications** and the **computer hardware**

**2 objectives:**

• **Convenience**: An OS makes a computer more convenient to use.

• **Efficiency**: An OS allows the computer system resources to be used in anefficient manner.

**Key interfaces in a typical computer system**

***Instruction Set Architecture*** (**ISA**) - Kiến trúc bộ chỉ lệnh

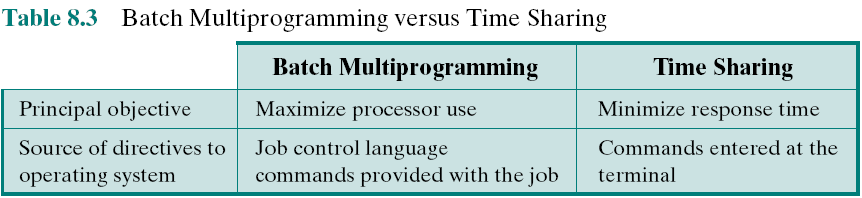
* + Defines the machine language instructionsthat a computer can follow
  + Boundary between hardware and software

***Application Binary Interface*** (**ABI**) - Giao diện nhị phân ứng dụng

* + Defines a standard for binary portability across programs (tính di động nhị phân)
  + Defines the system call interface to the operating system and the hardware resources and services available in a system through the user ISA

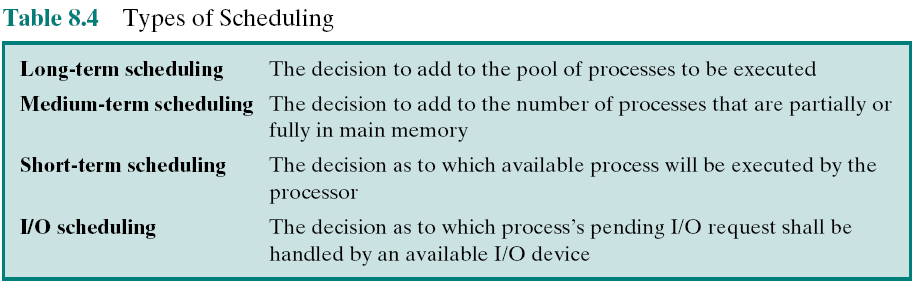
***Application Programming Interface*** (**API**) - giao diện lập trình ứng dụng

* + Gives a program access to the hardware resources and services available in a system through the user ISA supplemented with high-level language (HLL) library calls
  + Using an API enables application software to be ported easily to other systems that support the same API (chuyển đổi dễ dàng sang các hệ thống khác cùng API)

**Types of Operating Systems**

* + Interactive system
  + Batch system

**Scheduling**

**Long-term:** Thêm vào 1 nhóm quy trình

**Med-term:** Thêm vào 1 số lượng quy trình có 1 phần / toàn bộ trong main mem

**Short-term:** Quy trình nào có thể được thực thi bởi bộ xử lý

**I/O:** Yêu cầu IO chờ xử lý của quy trình nào sẽ được xử lý bởi một thiết bị IO có sẵn

Each element of the process table is called as **P**rocess **C**ontrol **B**lock

**Concept of PROCESS**

**Program**: executable file stored in external memory

**Process**:

A program in execution

• The “animated spirit” of a program

• That entity to which a processor is assigned

**5 Process states**

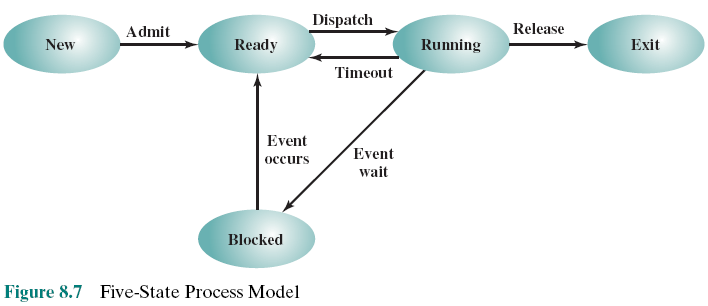
**New**: A program is admitted by the high-level scheduler but is not yet ready to execute. The OS will initialize the process, moving it to the ready state.

**Ready**: The process is ready to execute and is a waiting access to the processor.

**Running**: The process is being executed by the processor.

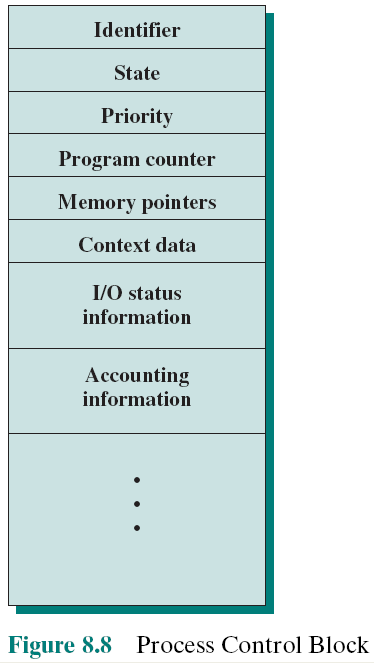
**Waiting**: The process is suspended from execution waiting for some system resource, such as I/O.

**Halted**: The process has terminated and will be destroyed by the OS.



**Process control block**

Khối điều khiền quy trình



• **Identifier**

• **State:** new, ready, and so on

• **Priority**

• **Program counter:** The address of the next instruction in the program to be executed.

• **Mem pointers:** The starting & ending locations of the process in memory.

• **Context data:** These are data that are present in registers in the processor while the process is executing. For now, it is enough to say that these data represent the “context” of the process. The context data plus the program counter are saved when the process leaves the running state. Chúng được bộ xử lý truy xuất khi nó tiếp tục thực thi quy trình.

• **I/O status information**

• **Accounting information:** include the amount of processor time and clock time used, time limits, account numbers, etc

**Memory Management**

\_\_\_ manage computer memory at the system level. Techniques:

* Swapping
* Partioning (phân vùng)
* Paging
* Virtual Memory
* Translation Lookaside Buffer (bộ đệm chuyển đổi)
* Segmentation (phân đoạn)

A \_\_\_\_\_\_\_\_\_ is a combinational circuit with a number of output lines, only one of which is asserted at any time

* decoder

**UNIT 12. INS SET**

**Machine Instruction Characteristics**

The operation of the processor is determined by the instructions it executes, referred to as ***machine instructions*** or ***computer instructions***

**Elements:**

• Operation code

• Source operand reference

• Result operand reference

• Next instruction reference

**Source and result operands can be in one of four areas:**

1. Main or virtual memory
2. I/O device
3. Processor register
4. Immediate

**Ins set design issues:**

**Operation repertoire:** How many and which operations to provide, how complex it should be

**Data types:** The various types of data upon which operations are performed

**Instruction format:** Instruction length (in bits), number of addresses, size of various fields, etc

**Registers:** Number of processor registers that can be referenced by instructions, and their use

**Addressing:** The mode or modes by which the address of an operand is specified

**Operands**

**Numbers:** Binary integer or binary fixed point, Binary floating point, Decimal

**Characters: text or character strings**

* Most commonly used character code is the International Reference Alphabet (IRA) -> ASCII
* Another: EBCDIC is used on IBM mainframes

**Logical Data:** 2 advantages

* Memory can be used most efficiently for storing array
* To manipulate the bits of a data item

**Address**

**Operations**

Categorization:

* Data transfer
* Arithmetic
* Logical
* Conversion: Instructions that **change the format** or operate on the format of data
* I/O
* System control
* Transfer of control

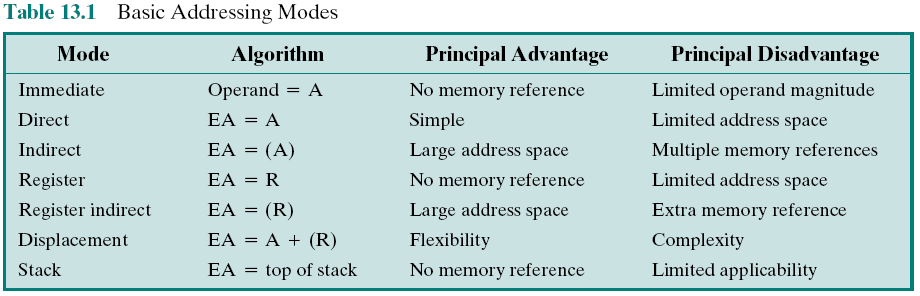
**Transfer of control operations:**

* **Branch instruction** (jump ins) has as 1 of its operands the address of the next ins to be executed.
* **Skip instruction** includes an implied address
* **Procedure Call Instructions** - Self-contained codes that is incorporated into a larger program

+ Reason: Economy (same code use many times), Modularity

+ Involve 2 basic instructions: **branches, returns**

**nesting of procedures:** The term refers to the occurrence of a procedure call inside a procedure.

**UNIT 13. ADDRESSING MODE & INS FORMAT**

**Immediate :** Operand is a specific value

**Direct :** Operand is the value of a variable

**Indirect :** Operand address is stored in another variable

**Register :** Operand is a specific register

**Register Indirect :** Operand address is stored in a register

**Displacement :** Replace the value of a variable with an expression

**Stack (Implicit**)**:** Operand address is stored in stack register

**Displacement addressing**

Combines the capabilities of direct addressing and register indirect addressing. Involves: **Relative addressing, Base-register addressing, Indexing**

**Instruction format**

**Instruction Length**

* Should be equal to the memory-transfer length or one should be a multiple of the other
* Should be a multiple of the character length, which is usually 8 bits, and of the length of fixed-point numbers

**Allocation of Bits**

* Number of addressing modes
* Number of operands
* Register versus memory
* Number of register sets
* Address range
* Address granularity (cốt lõi)

**Variable-Length Instructions**

* Provided efficiently and compactly
* Increases the complexity of the processor
* Does not remove the desirability of making all of the instruction lengths integrally related to word length

**Assembly Language**

**Low-level** programming language.

There is a very strong (**one-to-one**) correspondence between the language and the architecture's machine code instructions.

Assembly language is **converted** into executable machine code by a utility program referred to as an ***assembler***; the conversion process is referred to as ***assembly***, or ***assembling*** the code.

Assembly language uses a **mnemonic** to represent each low-level machine instruction or operation

**Macro assemblers** include a macro instruction facility so that (parameterized) assembly language text can be represented by a name. (tập lệnh, chỉ dẫn)

**UNIT 14. Processor Structure and Function**

**Processor Organization**

**Requirements:**

* Fetch instruction (from memory (register, cache, main memory))
* Interpret instruction (what action is required)
* Fetch data (data from memory or an I/O module)
* Process data (performing some operations on data)
* Write data (writing result to memory or an I/O module)

**2 roles:**

* **User-visible registers**: Enable the machine or assembly language programmer to minimize main memory references by optimizing use of registers.
* **General purpose:** Can be assigned to a variety of functions
* **Data:** hold data and not in the calculation
* **Address:** somewhat general purpose or devoted to a particular addressing mode
* **Condition codes (***flags):* Bits set by the processor hardware as the result of operations
* **Control and status registers**: Used by CU to control the operation of the processor and by privileged, OS programs to control the execution of programs.
* **PC:** Contains the address of an instruction to be fetched
* **IR:** Contains the instruction most recently fetched
* **Memory address register (MAR):** Contains the address of a location in memory
* **Memory buffer register (MBR):** Contains a word of data to be written to memory or the word most recently read

**Program Status Word (PSW)**

Register or set of registers that contain status information

* **Sign**: sign bit of the result
* **Zero**: set ~ res is 0.
* **Carry**: Set if an operation resulted in a carry (addition) into or borrow (sub- traction) out of a high-order bit. Used for multiword arithmetic operations.
* **Equal**: Set if a logical compare result is equality.
* **Overflow**: indicate arithmetic overflow.
* **Interrupt** **Enable/Disable**.
* **Supervisor**: Indicates whether the processor is executing in supervisor or user mode.

**Instruction Cycle**

* **Fetch**: Read the next instruction from memory into the processor.
* **Execute**: Interpret the opcode and perform the indicated operation.
* **Interrupt**: If interrupts are enabled and an interrupt has occurred, **save the current process** state and **service the interrupt**.

**Pipelining**

Additional Stages:

* **Fetch instruction (FI):** Read the next expected instruction into a buffer
* **Decode instruction (DI):** Determine the opcode and the operand specifiers
* **Calculate operands (CO):** Calculate the effective address of each source operand
* **Fetch operands (FO):** Fetch each operand from memory; Operands in registers need not be fetched
* **Execute instruction (EI):** Perform the indicated operation and store the result, if any, in the specified destination operand location
* **Write operand (WO):** Store the result in memory

**Hazards**

The pipeline must **stall** (trì hoãn) because conditions do not permit continued execution - as a pipeline **bubble**

* **Resource hazards:** when two or more instructions that are already in the pipeline need the same resource - structural hazard
* **Data hazards:** there is a conflict in the access of an operand location
* Read after write (RAW), or true dependency
* Write after read (WAR), or antidependency
* Write after write (WAW), or output dependency
* **Control Hazard:** pipeline makes the wrong decision on a branch prediction – branch hazard

**Dealing with Branches**

* **Multiple Streams**: brute-force approach, pipeline fetch both instructions, make use of 2 streams
* **Prefetch Branch Target:** target of the conditonal branch is prefetched (+instruction).
* **Loop Buffer:** Small, very-high speed memory maintained by FI, contain ***n***most recently fetched ins, in sequence
* **Branch Prediction:**
* Static: Predict **never taken**, Predict **always taken**, Predict **by opcode**
* Dynamic: Taken/not taken switch, Branch history table
* **Delayed Branch:** automatically rearranging instructions within a program, so that branch instructions occur later than actually desired.

The **cycle time** of an **ins pipeline** is the time needed to advance a set of instructions one stage through the pipeline.

**UNIT 15. RISC**

**Three elements characterize RISC architectures:**

* Use a **large number of registers** or use a compiler to optimize register usage
* Careful attention needs to be paid to the design of instruction pipelines
* Instructions should have predictable costs and be consistent with a high-performance implementation

**The Use of a Large Register File**

**Overlapping Register Windows**

* Register windows: a group of registers used to pass arguments between procedure calls.
* The use of register windows is a technique to **improve the performance of a particularly common operation**, the procedure call

**Global Variables**

* Variables declared as global in an HLL can be assigned memory locations by the compiler and all machine instructions that reference these variables will use memory reference operands -> inefficient
* Incorporate a set of **global registers** in the processor:
* These registers would be fixed in number and available to all procedures
* A unified numbering scheme can be used to simplify the instruction format (đánh số thống nhất)

**RISC Pipelining**

Instruction pipelining is often used to enhance performance. Most instructions in RISC are register to register.

**Instruction cycle: two stages:**

* I: Instruction fetch.
* E: Execute, ALU operation, Input and output are registers.

**Load and store operations, three stages:**

* I: Instruction fetch.
* E: Execute. Calculates memory address.
* D: (direction) Memory. Register-to-memory or memory-to-register operation.

**Optimization of Pipelining**

**Delayed branch**

* Does not take effect until after execution of following instruction
* This location immediately following the branch is the delay slot 🡪 Insert theinstruction NOOP

**Delayed Load**

* Register to be target is locked by processor
* Continue execution of instruction stream until register required
* Idle until load is complete
* Re-arranging instructions can allow useful work while loading

**Loop Unrolling (mở rộng vòng lặp)**

* Replicate body of loop a number of times
* Iterate loop fewer times
* Reduces loop overhead
* Increases instruction parallelism
* Improved register, data cache, or TLB locality

**RISC vs CISC**

**UNIT 16. SUPERSCALAR**

**Superscalar Implementation**

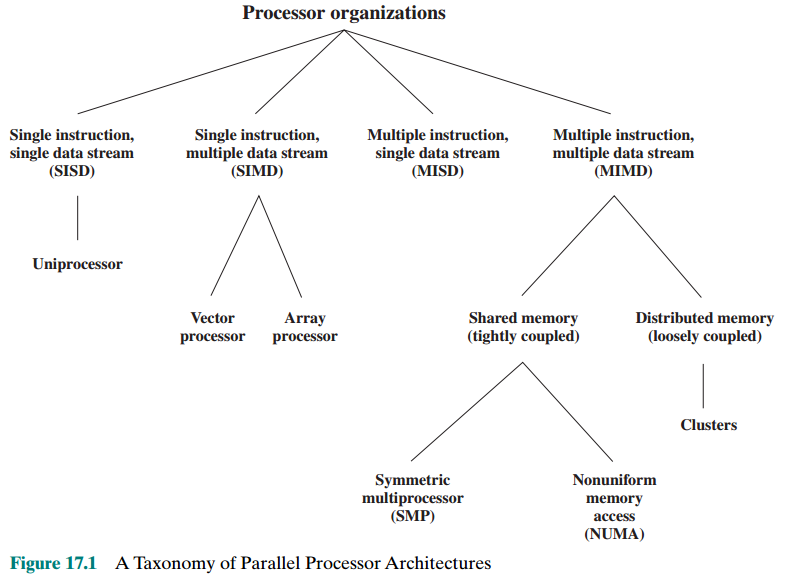
**Key elements:**

* Instruction fetch strategies that simultaneously fetch multiple instruction
* Logic for determining true dependencies involving register values, and mechanisms for communicating these values to where they are needed during execution
* Mechanisms for initiating, or issuing, multiple instructions in parallel
* Resources for parallel execution of multiple instructions, including multiple pipelined functional units and memory hierarchies capable of simultaneously servicing multiple memory references
* Mechanisms for committing the process state in correct order

**UNIT 17. Parallel Processing**

**Multiple Processor Organizations**

* **SISD** stream: A single processor executes a single instruction stream to operate on data stored in a single memory. Uniprocessors fall into this category.
* **SIMD** stream: A single machine instruction controls the simultaneous execution of a number of processing elements on a lockstep basis. Each processing element has an associated data memory, so that instructions are executed on different sets of data by different processors. Vector and array processors fall into this category, and are discussed in Section 18.7.
* **MISD** stream: A sequence of data is transmitted to a set of processors, each of which executes a different instruction sequence. This structure is not commercially implemented.
* **MIMD** stream: A set of processors simultaneously execute different instruction sequences on different data sets. SMPs, clusters, and NUMA systems fit into this category.



**Symmetric Multiprocessor (SMP)**

Can be defined as a stand alone computer with the following characteristics:

1. There are **two or more similar processors** of comparable capability.

2.These processors **share the same main memory and I/O facilities** and are interconnected by a bus or other internal connection scheme, such that memory access time is approximately the same for each processor.

3. All processors **share access to I/O devices**, either through the same channels or through different channels that provide paths to the same device.

4. All processors can **perform the same functions** (hence the term *symmetric).*

5. 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.

**Advantages:**

DMA:

• Addressing: <source, destination>

• Arbitration: Any I/O module can be “master.”

• Time-sharing

Performance

Availability

Incremental growth

Scaling

**Cache Coherence and the MESI Protocol**

**Write back**: Write operations are usually made only to the cache. Main memory is only updated when the corresponding cache line is flushed from the cache 🡪 inconsistency (mâu thuẫn)

**Write through:** All write operations are made to main memory as well as to the cache, ensuring that main memory is always valid. Even with the write-through policy, inconsistency can occur unless other caches monitor the memory traffic or receive some direct notification of the update

* **MESI** protocol is recommended

**Cache Coherence:** 2 categories:

* **Directory protocols:** the data being shared is placed in a common directory that maintains the coherence between caches
* **Snoopy protocols:** every cache controller monitors the bus, listening for broadcasts which may cause it to invalidate its cache line. – thăm dò, rình mò

To provide **cache consistency** on an SMP the data cache supports MESI:

* **Modified:** The line in the cache has been modified and is available only in this cache
* **Exclusive:** The line in the cache is the same as that in main memory and is not present in any other cache
* **Shared:** The line in the cache is the same as that in main memory and may be present in another cache
* **Invalid:** The line in the cache does not contain valid data

**Multithreading and Chip Multiprocessors**

Processor performance can be measured by the rate at which it executes instructions

**MIPS rate = f \* IPC** // Million Instructions Per second

* f = processor clock frequency (in **MHz**)
* IPC = average Instructions Per Cycle

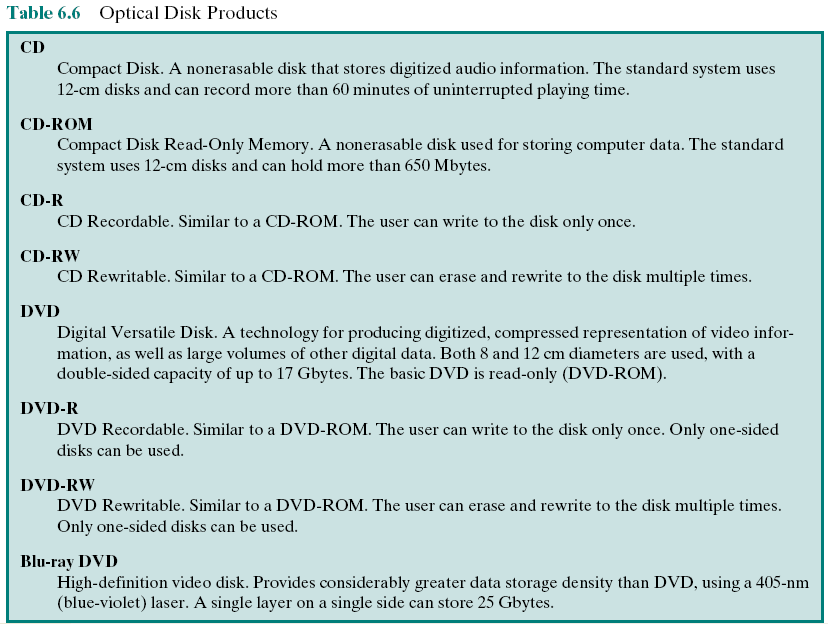
Increase performance by increasing clock frequency and increasing instructions that complete during cycle

**Multithreading**

* Allows for a high degree of **instruction-level parallelism** without increasing circuit complexity or power consumption 🡪 **Increase IPC**
* Instruction stream is divided into several smaller streams, known as threads, that can be executed in parallel

**Approaches to Explicit Multithreading**

* **Interleaved multithreading:** (a.k.a fine-grained multithreading). Processor deals with 2 or more thread contexts at a time, switching from one thread to another at each clock cycle. If a thread is blocked because of data dependencies or memory latencies, that thread is skipped and a ready thread is executed.
* **Blocked multithreading:** (a.k.a coarse-grained multithreading). The instructions of a thread are executed successively until an event occurs that may cause delay, such as a cache miss. This event induces a switch to another thread. This approach is effective on an **in-order** processor that would stall the pipeline for a delay event such as a cache miss.
* **Simultaneous multithreading (SMT):** Instructions are simultaneously issued from multiple threads to the execution units of a superscalar processor. This combines the wide superscalar instruction issue capability with the use of multiple thread contexts.
* **Chip multiprocessing:** In this case, the entire processor is replicated on a single chip and each processor handles separate threads. The advantage of this approach is that the available logic area on a chip is used effectively without depending on ever-increasing complexity in pipeline design.

CHAP 7

I/O module, channel, controller

Cycle stealing, fly-by

CHAP 15

Delay load, program, slot, register

**CHAP 16**

Instruction Issue Policy

Chiến lược phát lệnh

* **Instruction issue:** Refers to the process of initiating instruction execution in the processor’s functional units
* **Instruction issue policy:**
* Refers to the protocol used to issue instructions
* Instruction issue occurs when instruction moves from the **decode stage** of the pipeline to the **first execute stage of the pipeline**
* **Three types of orderings are important:**
* The order in which instructions are fetched
* The order in which instructions are executed
* The order in which instructions update the contents of register and memory locations
* **Superscalar instruction issue policies can be grouped into the following categories:**
* In-order issue with in-order completion
* In-order issue with out-of-order completion
* Out-of-order issue with out-of-order completion

**Optimization of Pipelining**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* **Pipelining**: Individual instructions are executed through a pipeline of stages so that while one instruction is executing in one stage of the pipeline, another instruction is executing in another stage of the pipeline. •
* **Superscalar**: Multiple pipelines are constructed by replicating execution resources. This enables parallel execution of instructions in parallel pipelines, so long as hazards are avoided.
* **Simultaneous** **multithreading** (SMT): Register banks are replicated (nhân rộng) so that multiple threads can share the use of pipeline resources.