CHƯƠNG 1

1. Khái niệm kiến trúc máy tính

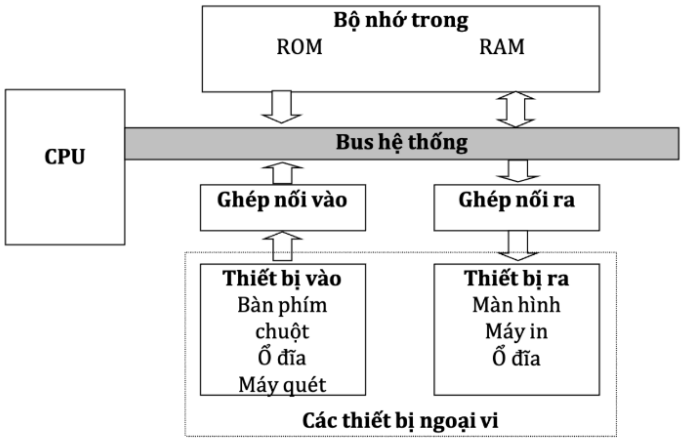
- Kiến trúc máy tính (computer architecture): là khoa học về lựa chọn và kết nối các thành phần phần cứng của máy tính nhằm đạt yêu cầu:

+ Hiệu năng: càng nhanh càng tốt, đạt tốc độ xử lý cao hơ
+ Chức năng: đáp ứng nhiều chức năng, có thêm nhiều tính năng phong phú và hữu ích
+ Giá thành: càng rẻ càng tốt

Ví dụ: Các thế hệ máy khác nhau với cùng kiến trúc phần cứng như Iphone 12, Iphone 13, Iphone 14..

2. Sơ đồ khối chức năng và chức năng của thành phần trong máy tính

*** Sơ đồ máy tính**



*** Chức năng của thành phần trong máy tính**

- Bộ xử lý trung tâm (CPU):

+ Chức năng:
  - Đọc lệnh từ bộ nhớ
  - Giải mã và thực hiện lệnh

+ Bao gồm:
  - Khối điều khiển (CU: Control Unit) đọc, giải mã và điều khiển quá trình thực hiện lệnh
  - Khối tính toán số học và logic (ALU: Arithmetic and Logic Unit): thực hiện các phép toán số học như cộng trừ, nhân, chia, và các phép toán lôgic như và, hoặc,phủ định và các phép dịch, quay

- Các thanh ghi (Registers) kho chứa lệnh và dữ liệu tạm thời cho CPU xử lý
- Bus trong CPU: truyền dẫn các tín hiệu giữa các bộ phận trong CPU và kết nối với hệ thống bus ngoài

- Bộ nhớ trong:

+ Chức năng: Lưu trữ lệnh và dữ liệu để CPU xử lý
+ Bao gồm:
  - ROM – Read Only Memory:
    + Lưu trữ lệnh và dữ liệu của hệ thống
    + Thông tin trong ROM được nạp từ khi sản xuất và thường chỉ có thể đọc ra trong quá trình sử dụng
    + Thông tin trong ROM vẫn tồn tại khi mất nguồn nuôi
  - RAM – Random Access Memory:
    + Lưu trữ lệnh và dữ liệu của hệ thống và người dùng

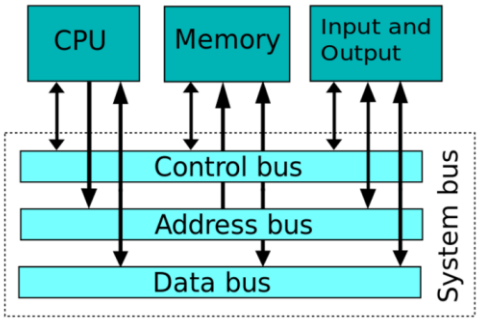
+ RAM thường có dung lượng lớn hơn nhiều so với ROM
+ Thông tin trong RAM sẽ mất khi mất nguồn nuôi

- Các thiết bị vào ra (thiết bị ngoại vi):

+ Thiết bị vào (input devices): nhập dữ liệu và điều khiển hệ thống và kết xuất dữ liệu ra
  - Bàn phím
  - Chuột
  - Ổ đĩa
  - Máy quét

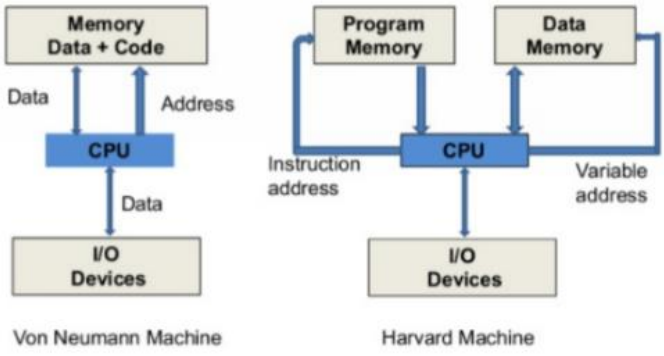
+ Thiết bị ra: kết xuất dữ liệu
  - Màn hình
  - Máy in
  - Ổ đĩa
  - Máy vẽ

- Bus hệ thống:



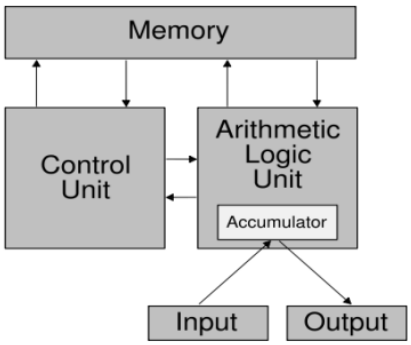
+ Tập các đường dây kết nối CPU với các thành phần khác của máy tính
+ Bao gồm 3 loại:
  - Bus địa chỉ (gọi là bus A): truyền tín hiệu địa chỉ từ CPU đến bộ nhớ và các thiết bị ngoại vi
  - Bus dữ liệu (gọi là bus D): vận chuyển các tín hiệu dữ liệu theo hai chiều đi và đến CPU
  - Bus điều khiển (bus C): truyền tín hiệu điều khiển từ CPU đến các thành phần khác, đồng thời truyền tín hiệu trạng thái của các thành phần khác đến CPU

3. Kiến trúc Von-Neuman

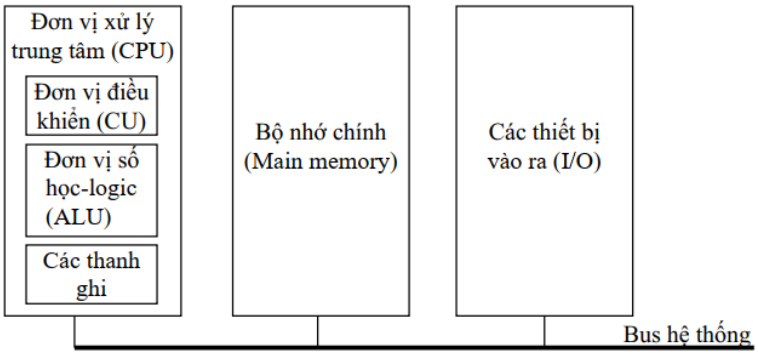


- Kiến trúc Von-Neumann được giới thiệu bởi John von-Neumann vào năm 1945.
- Các máy tính kiến trúc Von-Neumann dựa trên 3 khái niệm cơ bản:
  - Dữ liệu và lệnh được lưu trong một bộ nhớ đọc/viết chia sẻ - một bộ nhớ duy nhất được sử dụng để lưu trữ cả lệnh và dữ liệu

• Bộ nhớ được đánh địa chỉ dựa trên đoạn và không phụ thuộc vào nội dung nó lưu trữ
• Các lệnh của chương trình được chạy lần lượt, lệnh nọ tiếp sau lệnh kia: Stored-program digital computer



Hình 4 Kiến trúc máy tính von-Neumann nguyên thuỷ

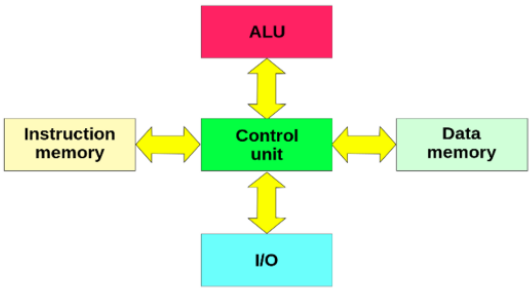


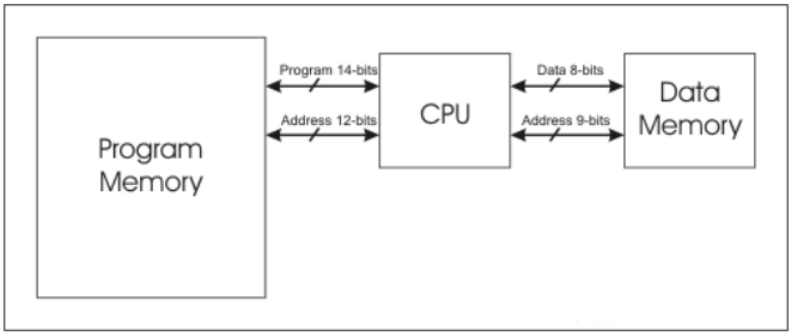
Hình 5 Kiến trúc máy tính von-Neumann hiện đại

- Quá trình thực hiện lệnh được chia thành 3 giai đoạn chính:
  (1) CPU đọc (fetch) lệnh từ bộ nhớ,
  (2) CPU giải mã và thực hiện lệnh; nếu lệnh yêu cầu dữ liệu, CPU đọc dữ liệu từ bộ nhớ;
  (3) CPU ghi kết quả thực hiện lệnh vào bộ nhớ (nếu có)

- Hạn chế: bộ nhớ lệnh và dữ liệu (cổ chai) không được truy cập cùng lúc nên thông lượng (throughput) nhỏ hơn rất nhiều so với tốc độ CPU có thể làm việc
+ Khắc phục: Dùng bộ nhớ cache giữa CPU và main memory
+ Khắc phục được khuyết điểm của kiến trúc Von-Neumann
+ Bộ nhớ được chia thành 2 phần:
  • Bộ nhớ chương trình
  • Bộ nhớ dữ liệu
+ CPU sử dụng 2 bus hệ thống để liên hệ với bộ nhớ:
  • CPU có thể đọc lệnh và truy cập dữ liệu bộ nhớ cùng một lúc.
  • Một bus A,D cho bộ nhớ chương trình và 1 bus A,D cho bộ nhớ dữ liệu (khác nhau về định dạng)

4. Kiến trúc Harvard





Hình 6 Kiến trúc máy tính Harvard

- Chia bộ nhớ trong thành hai phần riêng rẽ: Bộ nhớ lưu chương trình (Program Memory) và bộ nhớ lưu dữ liệu (Data Memory). Hai hệ thống bus riêng được sử dụng để kết nối CPU với bộ nhớ lưu chương trình và bộ nhớ lưu dữ liệu. Mỗi hệ thống bus đều có đầy đủ ba thành phần để truyền dẫn các tín hiệu địa chỉ, dữ liệu và điều khiển

+ Nhanh hơn vì băng thông bus rộng vì quá trình đọc lệnh không tranh chấp với quá trình truy xuất dữ liệu
+ Hỗ trợ nhiều truy cập đọc/viết bộ nhớ cùng lúc nên giảm xung đột truy cập bộ nhớ (đặc biệt khi CPU sử dụng kỹ thuật đường ống (pipeline))
+ Ngày này kiến trúc Havard cải tiến được ứng dụng cho các kiến trúc máy tính hiện đại: ARM, intel x86
+ Kiến trúc Harvard cũng được ứng dụng ở các hệ thống nhúng embed- ded system, chíp chuyên xử lý tín hiệu (DSP)

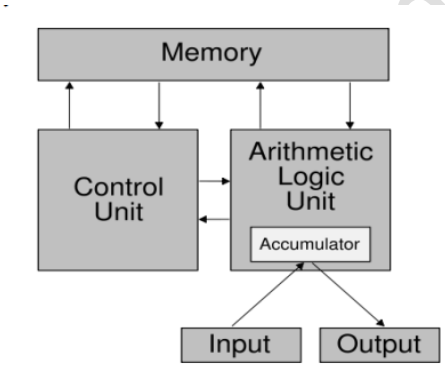
- Máy tính dựa trên kiến trúc Harvard có khả năng đạt được tốc độ xử lý cao hơn máy tính dựa trên kiến trúc von-Neumann do kiến trúc Harvard hỗ trợ hai hệ thống bus độc lập với bang thông lớn hơn

5. So sánh giữa kiến trúc Von Neumann hiện đại và Von Neumann cổ điển về các điểm tương đồng và khác biệt:

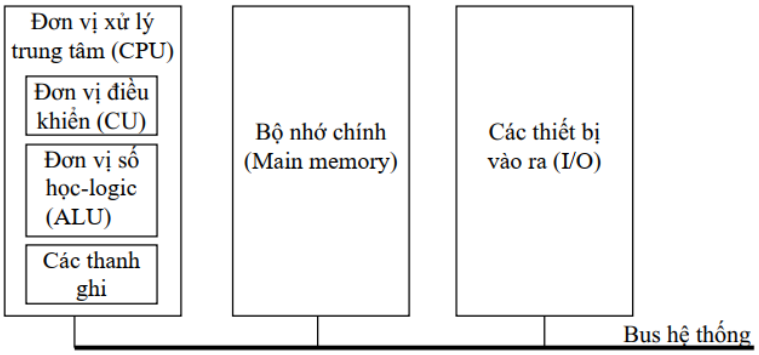
*** Giống:**

- Các máy tính kiến trúc Von-Neumann dựa trên 3 khái niệm cơ bản:
  - Dữ liệu và lệnh được lưu trong một bộ nhớ đọc/viết chia sẻ - một bộ nhớ duy nhất được sử dụng để lưu trữ cả lệnh và dữ liệu
  - Bộ nhớ được đánh địa chỉ dựa trên đoạn và không phụ thuộc vào nội dung nó lưu trữ
  - Các lệnh của chương trình được chạy lần lượt, lệnh nọ tiếp sau lệnh kia: Stored-program digital computer
- Quá trình thực hiện lệnh được chia thành 3 giai đoạn chính:
  (1) CPU đọc (fetch) lệnh từ bộ nhớ,
  (2) CPU giải mã và thực hiện lệnh; nếu lệnh yêu cầu dữ liệu, CPU đọc dữ liệu từ bộ nhớ;
  (3) CPU ghi kết quả thực hiện lệnh vào bộ nhớ (nếu có)

=> Khác biệt giữa Vonmann nguyên thủy và hiện đại:



Hình 4 Kiến trúc máy tính von-Neumann nguyên thuỷ



Hình 5 Kiến trúc máy tính von-Neumann hiện đại

- Kiến trúc Von Neumann cổ điển:

+ Bộ vi xử lý (CPU) trong kiến trúc Von Neumann cổ điển được tách thành hai phần chính: phần toán học và phần giải mã.

+ Phần toán học của CPU thực hiện các phép tính và các hoạt động toán học.

+ Phần giải mã của CPU chịu trách nhiệm giải mã các lệnh từ bộ nhớ và chuẩn bị dữ liệu để thực hiện các phép tính.

+ Đầu vào và đầu ra (input/output) trong kiến trúc Von Neumann cổ điển thường được xử lý riêng biệt và có các giao diện I/O đặc biệt để truyền dữ liệu giữa các thiết bị ngoại vi và bộ nhớ hoặc CPU.

- Kiến trúc Von Neumann hiện đại:

+ Trong kiến trúc Von Neumann hiện đại, phần toán học và phần giải mã của CPU thường được gộp chung thành một đơn vị xử lý trung tâm (CPU) duy nhất.

+ Đầu vào và đầu ra cũng được gộp chung trong một giao diện I/O chung, thường được gọi là bộ điều khiển bộ nhớ và I/O (Memory and I/O Controller).

+ Sự gộp chung này giúp tăng tính hiệu quả và giảm độ trễ trong quá trình xử lý và truyền dữ liệu.

+ Với sự gộp chung của phần toán học và giải mã trong CPU và đầu vào/đầu ra trong giao diện I/O, kiến trúc Von Neumann hiện đại đạt được sự tối ưu hóa và tính hiệu quả cao hơn so với kiến trúc Von Neumann cổ điển.

6. So sánh Kiến trúc Harvard và Kiến trúc Von-Neuman

*** Von-Neumann**
- Là kiến trúc máy tính cổ xưa dựa trên khái niệm máy tính chương trình được lưu trữ
- Dữ liệu và lệnh được lưu trong một bộ nhớ - một bộ nhớ duy nhất được sử dụng để lưu trữ cả lệnh và dữ liệu
- Địa chỉ bộ nhớ vật lí được sử dụng cho lệnh và dữ liệu giống nhau
- Rẻ hơn
- CPU không thể truy cập các lệnh đọc/ ghi 1 lúc
- Chung 1 bus đến CPU, bộ nhớ chính, thiết bị vào ra

*** Harvard:**
- Là kiến trúc máy tính hiện đại dựa trên mô hình Harvard Mark I.
- Bộ nhớ chia là 2 phần: 1 phần chứa dữ liệu, 1 phần chứa mã lệnh chương trình
- Địa chỉ bộ nhớ vật lí được sử dụng cho lệnh và dữ liệu tách biệt
- Đắt hơn
- CPU có thể truy cập các lệnh đọc/ ghi 1 lúc
- Đưa dữ liệu và chương trình vào CPU thông qua các bus riêng

=> Nhanh hơn vì băng thông bus rộng vì quá trình đọc lệnh không tranh chấp với quá trình truy xuất dữ liệu
=> Máy tính dựa trên kiến trúc Harvard có khả năng đạt được tốc độ xử lý cao hơn máy tính dựa trên kiến trúc von-Neumann do kiến trúc Harvard hỗ trợ hai hệ thống bus độc lập với bang thông lớn hơn

7. Máy tính lap hiện hay sử dụng theo kiến trúc nào

- Hiện nay, hầu hết máy tính laptop sử dụng kiến trúc Von Neumann.
- Trong kiến trúc Von Neumann, máy tính laptop:

+ Có một bộ nhớ chứa cả dữ liệu và lệnh, và CPU thực hiện việc lấy lệnh từ bộ nhớ, thực hiện nó và truy cập vào dữ liệu.
+ Việc truyền dữ liệu và lệnh giữa CPU và bộ nhớ được thực hiện qua một bus chung.

- Một số trường hợp đặc biệt, các máy tính laptop cải tiến hoặc biến thể của kiến trúc Von Neumann:

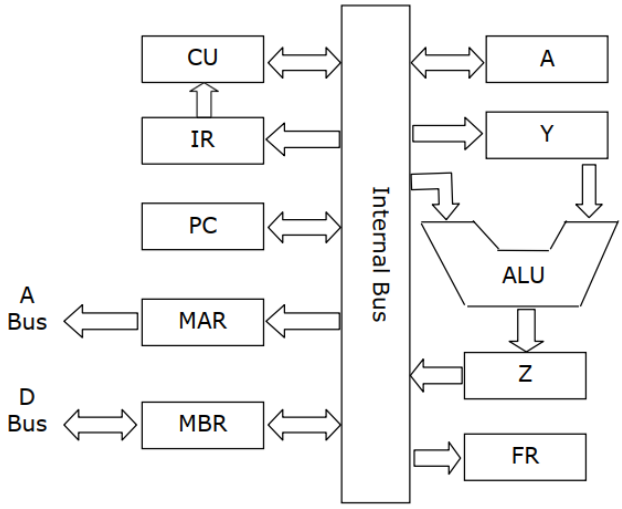
+ Sử dụng bộ nhớ cache để cải thiện hiệu suất
+ Sử dụng kiến trúc đa nhân với nhiều CPU hoạt động song song.

=> Dù thế cấu trúc chính của máy tính laptop vẫn tuân thủ theo kiến trúc Von Neumann.

CHƯƠNG 2:

1. Sơ đồ khối tổng quát của CPU

▶ CU: (Control Unit) Khối điều khiển
▶ IR: (Instruction Register) Thanh ghi lệnh
▶ PC: (Program Counter) Bộ đếm chương trình
▶ MAR: (Memory Address Register) Thanh ghi địa chỉ bộ nhớ
▶ MBR: (Memory Buffer Register) Thanh ghi nhớ đệm
▶ A: (Accumulator Register) Thanh ghi tích lũy
▶ Y, Z: (Temporary Register) Thanh ghi tạm thời
▶ FR: (Flag Register) Thanh ghi cờ
▶ ALU: (Arithmetic and Logic Unit) Khối tính toán số học -logic



Hình 14 Sơ đồ khối tổng quát của CPU

2. Chu kỳ xử lý lệnh của CPU

- Chu kỳ lệnh (Instruction Cycle) là khoảng thời gian để CPU thực hiện xong một lệnh kể từ khi CPU cấp phát tín hiệu địa chỉ ô nhớ chứa lệnh đến khi nó hoàn tất việc thực hiện lệnh đó.

Mỗi chu kỳ lệnh của CPU được mô tả theo các bước sau:

1. Khi một chương trình được chạy, hệ điều hành tải mã chương trình vào bộ nhớ trong RAM
2. Địa chỉ lệnh đầu tiên của chương trình được đưa vào thanh ghi PC
3. Địa chỉ của ô nhớ chứa lệnh được chuyển tới bus A qua thanh ghi MAR
4. Bus A truyền địa chỉ tới khối quản lý bộ nhớ MMU (Memory Man- agement Unit)
5. MMU chọn ô nhớ và sinh ra tín hiệu READ
6. Lệnh chứa trong ô nhớ được chuyển tới thanh ghi MBR qua bus D
7. MBR chuyển lệnh tới thanh ghi IR. Sau đó IR lại chuyển lệnh tới CU
8. CU giải mã lệnh và sinh ra các tín hiệu xử lý cho các đơn vị khác, ví dụ như ALU để thực hiện lệnh cộng
9. Địa chỉ trong PC được tăng lên để trỏ tới lệnh tiếp theo của chương trình sẽ được thực hiện
10. Thực hiện lại các bước 3->9 để chạy hết các lệnh của chương trình

- Ví dụ: Trình bày chu trình CPU xử lý cho lệnh sau và trạng thái CPU (giá trị các thanh ghi trong CPU) khi thực hiện lệnh sau: z = x + y Với giả thiết lệnh cộng được mã hóa có giá trị là 1001H, Câu lệnh ở địa chỉ ô 1000H và x có giá trị là 1 và y có giá trị là 2.

3. Các thanh ghi

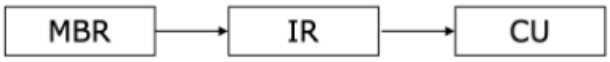
Thanh ghi tích lũy hay thanh ghi A

- Là một trong những thanh ghi quan trọng nhất của CPU
  + Lưu trữ các toán hạng đầu vào
  + Lưu kết quả đầu ra
- Kích thước của thanh ghi A tương ứng với độ dài từ xử lý của CPU: 8, 16, 32, 64 bit
- Cũng được sử dụng để trao đổi dữ liệu với các thiết bị vào ra
- Ví dụ: thực hiện phép tính x + y -> z
  • Toán hạng x được đưa vào thanh ghi A

• Toán hạng y được đưa vào thanh ghi Y
• ALU thực hiện phép cộng A+Y , kết quả được lưu vào Z
• Kết quả sau đó lại đươc đưa vào A

Bộ đếm chương trình PC
- Program Counter/Instruction Pointer lưu địa chỉ của lệnh tiếp theo
- PC chứa địa chỉ ô nhớ chứa lệnh đầu tiên của chương trình khi nó được kích hoạt và được tải vào bộ nhớ
- Khi CPU chạy xong 1 lệnh, địa chỉ lệnh tiếp theo được tải vào PC
- Kích thước của PC phụ thuộc vào thiết kế CPU: 8, 16, 32, 64 bit

***Thanh ghi lệnh IR* (Instruction register)**



- Lưu trữ lệnh đang được thực hiện
- IR lấy lệnh từ MBR và chuyển nó tới CU để giải mã lệnh

**** Các thanh ghi MAR và MBR***
- MAR
+ Là thanh ghi địa chỉ bộ nhớ (Memory address register) - giao diện giữa CPU và bus địa chỉ.
+ Nhận địa chỉ ô nhớ chứa lệnh tiếp theo từ PC và chuyển tiếp ra bus địa chỉ.
- MBR
+ Là thanh ghi đệm dữ liệu (Memory buffer register) - giao diện giữa CPU và bus địa chỉ.
+ Nhận lệnh từ bus địa chỉ và chuyển tiếp lệnh đến IR thông qua bus trong CPU.

**** Các thanh ghi tạm thời***
- CPU thường sử dụng một số thanh ghi tạm thời để chứa toán hạng đầu vào và kết quả đầu ra, như các thanh ghi tạm thời X, Y và Z. Ngoài ra, các thanh ghi tạm thời còn tham gia trong việc hỗ trợ xử lý song song (thực hiện nhiều lệnh cùng một thời điểm) và hỗ trợ thực hiện lệnh theo cơ chế thực hiện tiên tiến kiểu không theo trật tự (OOO – Out Of Order execution).

- CPU thường sử dụng một số thanh ghi tạm thời để:
+ Lưu trữ các toán hạng đầu vào
+ Lưu các kết quả đầu ra
+ Hỗ trợ xử lý song song (tại một thời điểm chạy nhiều hơn 1 lệnh)
+ Hỗ trợ thực hiện lệnh theo cơ chế thực hiện tiên tiến kiểu không trật tự (OOO – Out Of Order execution)

**** Con trỏ ngăn xếp SP***
- Ngăn xếp là 1 đoạn bộ nhớ đặc biệt hoạt động theo nguyên tắc vào sau ra trước (LIFO)
- PC chứa địa chỉ ô nhớ chứa lệnh đầu tiên của chương trình khi nó được kích hoạt và được tải vào bộ nhớ
- Con trỏ ngăn xếp là thanh ghi luôn trỏ tới đỉnh của ngăn xếp
- 2 thao tác với ngăn xếp:
+ Push: đẩy dữ liệu vào ngăn xếp

```
SP <– SP + 1     ; tăng địa chỉ đỉnh ngăn xếp
SP <– Data       ; nạp dữ liệu vào ngăn xếp
```

+ Pop: lấy dữ liệu ra khỏi ngăn xếp

```
Register <– SP   ; chuyển dữ liệu từ đỉnh ngăn xếp vào thanh ghi
```

SP <– SP – 1 ; giảm địa chỉ đỉnh ngăn xếp

*** *Các thanh ghi tổng quát* - thanh ghi đa năng**

- Có thể sử dụng cho nhiều mục đích:
  + Lưu các toán hạng đầu vào
  + Lưu các kết quả đầu ra
- Ví dụ: CPU 8086 có 4 thanh ghi đa năng:
  - AX: Accumulator Register
  - BX: Base Register
  - CX: Counter Register
  - DX: Data Register

5. Chức năng và ý nghĩa của các thanh ghi cờ (thanh ghi trạng thái) FR. Ví dụ:

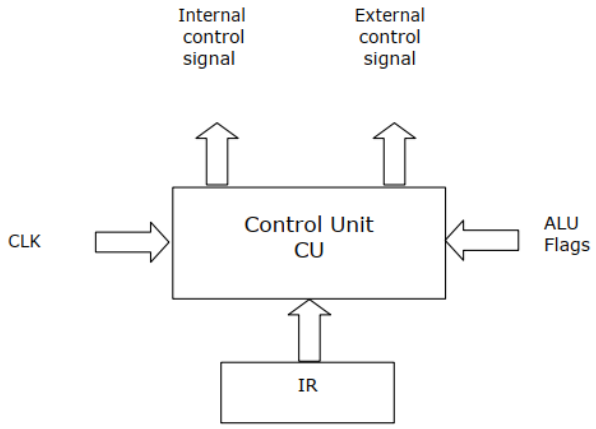
Flag	ZF	SF	CF	AF	IF	OF	PF	1
Bit No	7	6	5	4	3	2	1	0

Hình 16 Các bit của thanh ghi cờ FR 8 bit

- Mỗi bit của thanh ghi cờ lưu trữ trạng thái kết quả phép tính được ALU thực hiện
- Có 2 kiểu cờ:
  + Cờ trạng thái: CF, OF, AF, ZF, PF, SF
  + Cờ điều khiển: IF, TF, DF
- Các bit cờ thường được dùng là các điều kiện rẽ nhánh lệnh tạo logic chương trình
- Kích thước FR phụ thuộc thiết kế CPU
- ZF: Zero Flag, ZF=1 nếu kết quả =0 và ZF=0 nếu kết quả <>0.
- SF: Sign Flag, SF=1 nếu kết quả âm và SF=0 nếu kết quả dương
- CF: Carry Flag, CF=1 nếu có nhớ/mượn ở bit trái nhất
- AF: Auxiliary Flag, AF=1 nếu có nhớ ở bit trái nhất của nibble
- OF: Overflow Flag, OF=1 nếu có tràn, OF=0 ngược lại
- PF: Parity Flag, PF=1 nếu tổng số bit 1 trong kết quả là số lẻ, PF=0 ngược lại
- IF: Interrupt Flag, IF=1: ngắt được phép, IF=0: cấm ngắt

*** Ví dụ:**

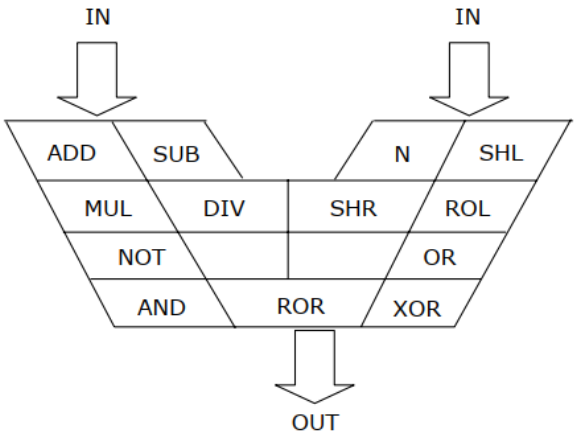
6. Sơ đồ khối và chức năng của các khối: CU, ALU và bus ở bên trong CPU
*** CU:**



Hình 17 Khối điều khiển CU và các tín hiệu

- Nhận 3 tín hiệu đầu vào:
+ Lệnh từ IR
+ Giá trị các cờ trạng thái
+ Xung đồng hồ
- CU sinh 2 nhóm tín hiệu đầu ra:
+ Nhóm tín hiệu điều khiển các bộ phận bên trong CPU
+ Nhóm tín hiệu điều khiển các bộ phận bên ngoài CPU
- Sử dụng nhịp đồng hồ để đồng bộ hóa các đơn vị bên trong CPU và giữa CPU với các thành phần bên ngoài

*** ALU KHỐI SỐ HỌC VÀ LOGIC**



Hình 18 Bộ tính toán ALU

- ALU có:
+ 2 cổng IN để nhận đầu vào từ các thanh ghi
+ 1 cổng OUT được nối với bus trong để gửi kết quả tới các thanh ghi

- Bao gồm các đơn vị chức năng con để thực hiện các phép toán số học và logic:
    + Bộ cộng (ADD), bộ trừ (SUB), bộ nhân (MUL), bộ chia (DIV), . . .
    + Các bộ dịch (SHIFT) và quay (ROTATE)
    + Bộ logic như phủ định (NOT), bộ và (AND), bộ hoặc (OR), và bộ hoặc loại trừ (XOR)

*** BUS:**
- Bus trong là kênh liên lạc của tất cả các thành phần trong CPU
- Hỗ trợ liên lạc 2 chiều
- Bus trong có giao diện để trao đổi thông tin với bus ngoài
- Bus trong có băng thông lớn và tốc độ nhanh hơn so với bus ngoài

CHƯƠNG 3

1. Khái niệm về lệnh máy tính, chu kì thực hiện lệnh và các giai đoạn, các pha trong 1 chu kỳ của lệnh // và khuôn dạng của lệnh

*** Lệnh máy tính**
- Lệnh máy tính (Instruction) là một từ nhị phân (binary word) được gán một nhiệm vụ cụ thể, hướng dẫn cho máy tính biết phải làm gì.

+ Lệnh chương trình được lưu trong bộ nhớ
+ Lệnh được đọc từ bộ nhớ vào CPU để giải mã và thực hiện
+ Mỗi lệnh có một chức năng riêng

- Mỗi bộ xử lý có một tập lệnh xác định:

+ Tập lệnh thường có hàng chục đến hàng trăm lệnh
+ Mỗi lệnh là một chuỗi số nhị phân mà bộ xử lý hiểu được để thực hiện một thao tác xác định.
+ Các lệnh được mô tả bằng các ký hiệu gợi nhớ dạng text: chính là các lệnh của hợp ngữ (assembly language). Ví dụ: ADD A, 1

- Tập lệnh gồm nhiều lệnh, được chia thành các nhóm theo chức năng:

+ Chuyển dữ liệu (data movement)
+ Tính toán (computational)
+ Điều kiện và rẽ nhánh (conditioning & branching)
+ Các lệnh khác

*** Pha, giai đoạn trong 1 chu kì lệnh**
- Quá trình thực hiện lệnh được chia thành các pha hay giai đoạn (stage). Mỗi lệnh có thể được thực hiện theo 4 giai đoạn:

+ Đọc lệnh IF (Instruction Fetch): lệnh được đọc từ bộ nhớ vào CPU
+ Giải mã lệnh ID (Instruction Decode): CPU giải mã lệnh
+ Chạy lệnh IE (Instruction Execution): CPU thực hiện lệnh
+ Ghi WB (Write Back): kết quả (nếu có) ghi vào thanh ghi/bộ nhớ

*** Chu kì lệnh**

- Chu kỳ lệnh (Instruction Cycle) là khoảng thời gian để CPU thực hiện xong một lệnh kể từ khi CPU cấp phát tín hiệu địa chỉ ô nhớ chứa lệnh đến khi nó hoàn tất việc thực hiện lệnh đó.
+ Một chu kỳ thực hiện lệnh gồm một số giai đoạn thực hiện lệnh
+ Một giai đoạn thực hiện lệnh có thể gồm một số chu kỳ máy
+ Một chu kỳ máy có thể gồm một số chu kỳ đồng hồ
+ Một chu kỳ thực hiện lệnh có thể gồm các thành phần sau:

- Chu kỳ đọc lệnh
- Chu kỳ đọc/ ghi bộ nhớ (memory read / write)
- Chu kỳ đọc / ghi thiết bị ngoại vi (I/O read/write)
- Chu kỳ bus rỗi (bus idle)

*** Khuôn dạng lệnh**

- Dạng tổng quát của lệnh máy tính gồm có 2 phần chính:
+ Mã lệnh (Opcode – Operation code): mã hóa cho thao tác mà bộ xử lý phải thực hiện
+ Địa chỉ của các toán hạng (Addresses of Operands): chỉ ra nơi chứa các toán hạng mà thao tác sẽ tác động:
  • Toán hạng nguồn (source operand): dữ liệu vào của thao tác
  • Toán hạng đích (destination operand): dữ liệu ra của thao tác

Mã lệnh	Địa chỉ của các toán hạng	
Opcode	Addresses of Operands	
Opcode	Des addr.	Source addr.

- Dựa vào số lượng các toán hạng, các lệnh đưa chia thành 5 loại:
+ Toán hạng 3 địa chỉ
+ Toán hạng 2 địa chỉ
+ Toán hạng 1.5 địa chỉ
+ Toán hạng 1 địa chỉ
+ Toán hạng 0 địa chỉ

- Một số quy ước về lệnh dạng CPU: (không)
+ Ri : là các thanh ghi của CPU
+ Racc : là thanh ghi Accumulation - tích luỹ / Tổng
+ (Ri): Nội dung ô nhớ có địa chỉ được lưu trong thanh ghi Ri
+ (100): Nội dung ô nhớ có địa chỉ được lưu trong ô nhớ 100
+ A, B, C, 1000 là địa chỉ các ô nhớ
+ M[100]: tham chiếu đến nội dung ô nhớ thứ 100
+ #100: Hằng số 100

- Toán hạng 3 địa chỉ
+ Khuôn dạng: OPCODE Addr1, Addr2, Addr3
+ Mỗi địa chỉ Addr1, Addr2, Addr3 tham chiếu đến một ô nhớ hoặc một thanh ghi.
+ Ví dụ:

- ADD R1, R2, R3;
  R3 + R2–> R1
  R2 cộng R3 sau đó kết quả đưa vào R1
  Ri là các thanh ghi CPU

- ADD A, B, C;
  M[B] + M[C] –> M[A]
  A, B, C là các vị trí trong bộ nhớ

- Toán hạng 1 địa chỉ
+ Khuôn dạng: OPCODE Addr
+ Mỗi địa chỉ Addr tham chiếu đến một ô nhớ hoặc một thanh ghi.
+ Khuôn dạng này sử dụng Racc (thanh ghi tích lũy) mặc định cho địa chỉ thứ 2
+ Ví dụ:
- ADD R1;
  R1 + Racc–> Racc
  Racc cộng R1 sau đó kết quả đưa vào Racc
  Ri là các thanh ghi CPU
- ADD A;
  M[A] + Racc –>Racc
  A là vị trí trong bộ nhớ

- Toán hạng 1.5 địa chỉ
+ Khuôn dạng: OPCODE Addr1, Addr2
+ Một địa chỉ tham chiếu tới 1 ô nhớ và địa chỉ còn lại tham chiếu tới 1 thanh ghi
+ Là dạng hỗn hợp giữa các toán hạng thanh ghi và vị trí bộ nhớ.
+ Ví dụ:
- ADD R1, B;
  M[B] + R1–> R1
  R1 cộng M[B] sau đó kết quả đưa vào R1
  Ri là các thanh ghi CPU
  B là vị trí trong bộ nhớ

- Toán hạng 0 địa chỉ
+ Khuôn dạng: OPCODE
+ Được thực hiện trong các lệnh mà thực hiện các thao tác ngăn xếp: PUSH & POP
+ Ví dụ:
PUSH a
PUSH b
ADD POP c
=> Có nghĩa c = a + b

- Phụ thuộc thiết kế CPU, tập lệnh của CPU có thể có số lượng các lệnh rất khác nhau bao gồm :
+ Các lệnh vận chuyển dữ liệu
+ Các lệnh số học và logic
+ Các lệnh điều khiển chương trình
+ Các lệnh vào & ra

- Chuyển dữ liệu giữa các phần nhớ của máy tính:
+ Giữa các thanh ghi trong CPU
- **MOVE Ri, Rj** ; Rj –> Ri

+ Giữa thanh ghi CPU và một vị trí trong bộ nhớ
    - MOVE Ri, 1000 ; M[1000] –> R
+ Giữa các vị trí trong bộ nhớ
    - MOVE 1000, (Rj) ; M[Rj] –> M[1000]

- Một số lệnh vận chuyển dữ liệu thông dụng:
+ MOVE: chuyển dữ liệu giữa các thanh ghi và ô nhớ
+ LOAD: nạp nội dung 1 ô nhớ vào 1 thanh ghi
+ STORE: lưu nội dung 1 thanh ghi ra 1 ô nhớ
+ PUSH/POP: đẩy/lấy dữ liệu vào/ra ngăn xếp

-Thực hiện các thao tác số học và logic giữa các thanh ghi và nội dung ô nhớ
+ Ví dụ:
    - ADD R1, R2, R3 ;R2 + R3 –> R1 ;
    - SUBSTRACT R1, R2, R3 ; R2 – R3 –> R1
- Một số lệnh toán học thông :
+ ADD: cộng 2 toán hạng
+ SUBSTRACT: trừ 2 toán hạng
+ MULTIPLY: nhân 2 toán hạng
+ DIVIDE: chia số học
+ INCREMENT: tăng 1
+ DECREMENT: giảm 1

- Các lệnh logic thông dụng:
+ NOT: phủ định
+ AND: và
+ OR: hoặc
+ XOR: hoặc loại trừ
+ COMPARE: so sánh
+ SHIFT: dịch
+ ROTATE: quay

- Được dùng để thay đổi trình tự các lệnh được thực hiện:
+ Các lệnh rẽ nhánh (nhẩy) có điều kiện (conditional branching/ jump)
+ Các lệnh rẽ nhánh (nhẩy) không điều kiện (unconditional branching/ jump)
    - CALL và RETURN: gọi thực hiện và trở về từ chương trình con
+ Đặc tính chung của các lệnh này là quá trình thực hiện lệnh của chúng làm thay đổi giá trị PC
+ Sử dụng các cờ ALU để xác định các điều kiện

- Một số lệnh vận chuyển dữ liệu thông dụng:
+ BRANCH – IF – CONDITION: chuyển đến thực hiện lệnh ở địa chỉ mới nếu điều kiện là đúng
+ JUMP: chuyển đến thực hiện lệnh ở địa chỉ mới
+ BRANCH-IF-EQUAL: chuyển đến thực hiện lệnh ở địa chỉ mới nếu kết quả của phép toán = 0
+ BRANCH-IF-GREATER-THAN: chuyển đến thực hiện lệnh ở địa chỉ mới nếu kết quả của phép toán > 0
+ CALL: chuyển đến thực hiện chương trình con

+ RETURN: trở về (từ chương trình con) thực hiện tiếp chương trình

- Ví dụ: Cộng nội dung 100 ô nhớ cạnh nhau, bắt đầu từ địa chỉ 1000. Kết quả lưu vào R0.

```
LOAD R1, #100              ; R1 <– 100
LOAD R2, #1000             ; R2 <– 1000
LOAD R0, #0                ; R0 <– 0
LOOP: ADD R0, (R2)         ; R0 <– R0 + M[R2]
INCREMENT R2               ; R2 <– R2 + 1
DECREMENT R1               ; R1 <– R1 - 1
BRANCH-IF-GREATER-THAN Loop;
                           ; Quay lại thực hiện lệnh sau nhãn Loop nếu R1 > 0.
```

- Các lệnh vào/ ra:

+ Được dùng để truyền dữ liệu giữa máy tính và các thiết bị ngoại vi
+ Các thiết bị ngoại vi giao tiếp với máy tính thông qua các cổng. Mỗi cổng có một địa chỉ dành riêng
+ Hai lệnh I/O cơ bản được sử dụng là các lệnh INPUT và OUTPUT
  • Lệnh INPUT được dùng để chuyển dữ liệu từ thiết bị ngoại vi vào tới bộ vi xử lý
  • Lệnh OUTPUT dùng để chuyển dữ liệu từ VXL ra thiết bị đầu ra

```
CLEAR R0            ; R0 <– 0
CLEAR R2            ; R2<– 0
MOVE R1, #100       ; R1<– 100
LAP:
ADD R0, 1000(R2)    ; R0<– R0+ M[R2+1000]
INCREMENT R2        ; R2<– R2+1
DECREMENT R1        ; R1<– R1-1
BRANCH_IF>0 LAP ; go to LAP if R1>0
STORE 2000, R0      ; M[2000] <– R0
```

2. Cpu pineline, bộ nhớ, bộ nhớ cache- cpu,

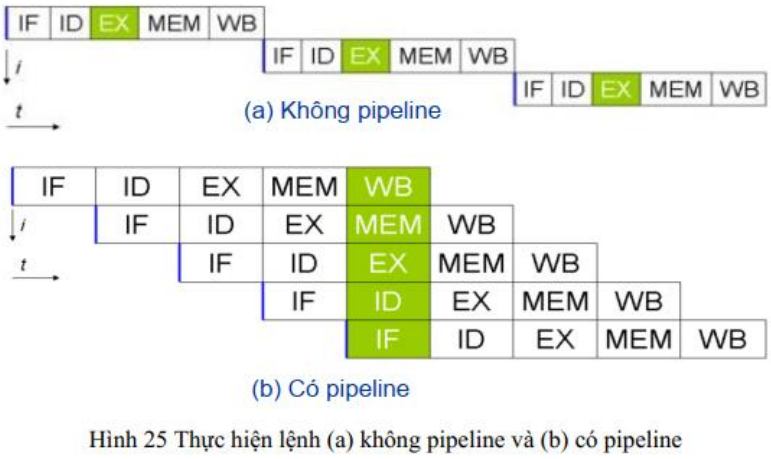
*** Cơ chế ống lệnh (pineline) -** cơ chế thực hiện xen kẽ các lệnh phương pháp thực hiện lệnh tiên tiến, cho phép đồng thời thực hiện nhiều lệnh, giảm thời gian trung bình thực hiện mỗi lệnh và như vậy tăng được hiệu năng xử lý lệnh của CPU. Việc thực hiện lệnh được chia thành một số giai đoạn và mỗi giai đoạn được thực thi bởi một đơn vị chức năng khác nhau của CPU. Nhờ vậy CPU có thể tận dụng tối đa năng lực xử lý của các đơn vị chức năng của mình, giảm thời gian chờ cho từng đơn vị chức năng.

* Giới thiệu về Nguyên lý CPU Pipeline:

- Quá trình thực hiện lệnh được chia thành các pha hay giai đoạn (stage). Mỗi lệnh có thể được thực hiện theo 5 giai đoạn của hệ thống load – store:

+ IF Đọc lệnh (Instruction Fetch): (IF): Lấy lệnh từ bộ nhớ (hoặc cache)
+ ID Giải mã lệnh (Instruction Decode): Thực hiện giải mã lệnh và lấy các toán hạng
+ IE Thực thi lệnh (Instruction Execution): Nếu là lệnh truy cập bộ nhớ thì tính toán địa chỉ bộ nhớ
+ (Memory access) Đọc - ghi bộ nhớ MEM : Đọ và ghi bố nhớ nếu không truy cập bộ nhớ thì không có giai đoạn này
+ WB Ghi (Write Back): kết quả (nếu có) CPU xử lý được ghi vào thanh ghi/bộ nhớ lưu

- Cải thiện hiệu năng bằng cách tăng số lượng lệnh vào xử lý:



Hình 25 Thực hiện lệnh (a) không pipeline và (b) có pipeline

=> Nhận xét: Chia thực hiện lệnh thành 5 bước –> Hiệu quả theo Pipeline hơn tương đương 5 lần

* Vấn đề xung đột và cách giải quyết
- Vấn đề xung đột tài nguyên (resource conflict)
    + Xung đột truy cập bộ nhớ
    + Xung đột truy cập thanh ghi
- Xung đột/ tranh chấp dữ liệu (data hazard)
    + Hầu hết là RAW hay Read After Write Hazard
- Các lệnh rẽ nhánh (Branch Instruction)
    + Không điều kiện
    + Có điều kiện
    + Gọi thực hiện và trở về từ chương trình con

**** Xung đột tài nguyên***
+ Tài nguyên không đủ
+ Ví dụ: nếu bộ nhớ chỉ hỗ trợ một thao tác đọc/ ghi tại một thời điểm, pipeline yêu cầu 2 truy cập bộ nhớ 1 lúc (đọc lệnh tại giai đoạn IF và đọc dữ liệu tại ID) –> nảy sinh xung đột
=> Giải pháp:
+ Nâng cao khả năng tài nguyên
+ Memory/ cache: hỗ trợ nhiều thao tác đọc/ ghi cùng lúc
+ Chia cache thành cache lệnh và cache dữ liệu để cải thiện truy nhập

**** Vấn đề tranh chấp dữ liệu***
⇒ Vấn đề Dữ liệu chưa sẵn sàng cho các lệnh phụ thuộc tiếp theo được gọi là Data Hazard - xung đột dữ liệu hay sự phụ thuộc dữ liệu giữa các lệnh.
=> Khắc phục:
+ Nhận biết các lệnh phụ thuộc dữ liệu
+ Các lệnh phụ thuộc cần thực thi (EX) sau khi các lệnh mà nó phụ thuộc thực hiện xong
⇒ Ngưng pipeline (stall): phải làm trễ hoặc ngưng pipeline bằng cách sử dụng một vài phương pháp tới khi có dữ liệu chính xác
+ Sử dụng complier để nhận biết RAW và:
    • Chèn các lệnh NO-OP vào giữa các lệnh có RAW
+ Lệnh NO-OP lệnh không thay đổi trạng thái CPU mà chỉ lấy đi 1 chu kỳ lệnh - thời gian 1 chu kỳ

• Thay đổi trình tự các lệnh trong chương trình và chèn các lệnh độc lập dữ liệu vào vị trí giữa 2 lệnh có RAW

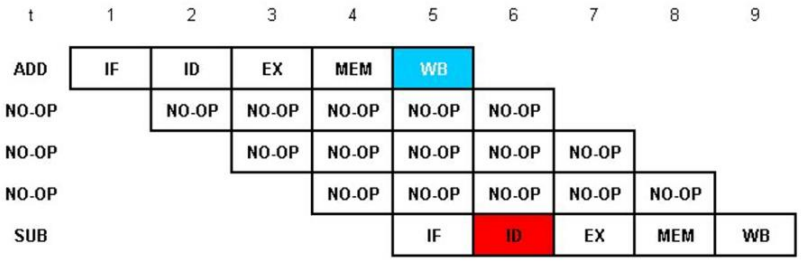
+ Sử dụng phần cứng để xác định RAW (có trong các CPUs hiện đại) và dự đoán trước giá trị dữ liệu phụ thuộc

+ Xét ví dụ sau:

ADD R1, R1, R3; R1 <– R1 + R3
SUB R4, R1, R2; R4 <– R1 - R2

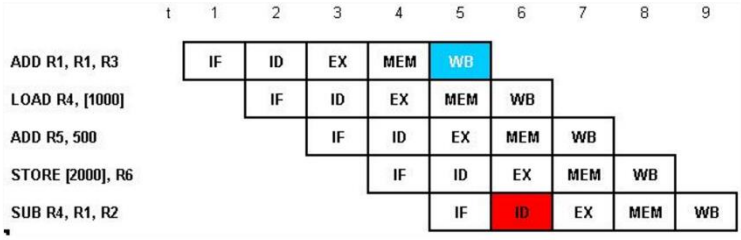
Hướng khắc phục xung đột dữ liệu

Làm trễ quá trình thực hiện lệnh SUB bằng cách chèn 3 NO-OP:



Hướng khắc phục xung đột dữ liệu

Chèn 3 lệnh độc lập dữ liệu vào giữa ADD và SUB:



*** Xử lý rẽ nhánh – branch (Branch Hazard)**

- Vấn đề trong kỹ thuật pipeline khi kiểm tra các điều kiện đối với các lệnh rẽ nhánh hay vòng lặp (ví dụ: while do)

+ Trường hợp này được gọi là Branch Hazard khi gặp câu lệnh rẽ nhánh và lặp.

+ Tỷ lệ các lệnh rẽ nhánh chiếm khoảng 10 - 30%. Các lệnh rẽ nhánh có thể gây ra:

• Gián đoạn trong quá trình chạy bình thường của chương trình
• Làm cho Pipeline rỗng nếu không có biện pháp ngăn chặn hiệu quả

+ Với các CPU mà pipeline dài (P4 với 31 giai đoạn) và nhiều pipeline chạy song song, vấn đề rẽ nhánh càng trở nên phức tạp hơn vì:

• Phải đẩy mọi lệnh đang thực hiện ra ngoài pipeline khi gặp lệnh rẽ nhánh
• Tải mới các lệnh từ địa chỉ rẽ nhánh vào pipeline. Tiêu tốn nhiều thời gian để điền đầy pipeline

- Khi 1 lệnh rẽ nhánh được thực hiện, các lệnh tiếp theo bị đẩy ra khỏi pipeline và các lệnh mới được tải
⇒ chèn các lệnh NO-OP đến khi nào câu lệnh rẽ nhánh / điều kiện thực hiện xong.

- Xử lý rẽ nhánh

+ Đích rẽ nhánh (branch target)

• Khi một lệnh rẽ nhánh được thực hiện, lệnh tiếp theo được lấy là lệnh ở địa chỉ đích rẽ nhánh (target) mà không phải lệnh tại vị trí tiếp theo lệnh nhảy
• Các lênh rẽ nhánh được xác định tại giai đoạn ID, vậy có thể biết trước chúng bằng cách giải mã trước
• Sử dụng đệm đích rẽ nhánh (BTB: branch target buffer) để lưu vết của các lệnh rẽ nhánh đã được thực thi

+ Rẽ nhánh có điều kiện (conditional branches)
• Làm chậm rẽ nhánh (delayed branching) (1)
• Dự báo rẽ nhánh (branch prediction) (2)

*** Lệnh rẽ nhánh có điều kiện**

- Khó quản lý các lệnh rẽ nhánh có điều kiện hơn vì:
• Có 2 lệnh đích để lựa chọn
• Không thể xác định được lệnh đích tới khi lệnh rẽ nhánh được thực hiện xong
• Sử dụng BTB riêng rẽ không hiệu quả vì phải đợi tới khi có thể xác định được lệnh đích.
- Chiến lược xử lý rẽ nhánh:
• Làm chậm rẽ nhánh
• Dự đoán rẽ nhánh
- Ý tưởng:
• Lệnh rẽ nhánh không làm rẽ nhánh ngay lập tức
• Mà nó sẽ bị làm chậm một vài chu kỳ đồng hồ phụ thuộc vào độ dài của pipeline
- Đặc điểm:
• Hoạt động tốt trên các vi xử lý RISC trong đó các lệnh có thời gian xử lý bằng nhau
• Pipeline ngắn (thông thường là 2 giai đoạn)
• Lệnh sau lệnh nhảy luôn được thực hiện, không phụ thuộc vào kết quả lệnh rẽ nhánh
- Cài đặt:
• Sử dụng complier để chèn NO-OP vào vị trí ngay sau lệnh rẽ nhánh
• Chuyển một lệnh độc lập từ trước tới ngay sau lệnh rẽ nhánh

- > Ưu nhược điểm của làm chậm rẽ nhánh:

+ Ưu điểm:
• Dễ cài đặt nhờ tối ưu trình biên dịch (complier)
• Không cần phần cứng đặc biệt
• Nếu chỉ chèn NO-OP làm giảm hiệu năng khi pipeline dài
• Thay các lệnh NO-OP bằng các lệnh độc lập có thể làm giảm số lượng NO-OP cần thiết tới 70%

+ Nhược điểm:
• Làm tăng độ phức tạp mã chương trình (code)
• Cần lập trình viên và người xây dựng trình biên dịch có mức độ hiểu biết sâu về pipeline vi xử lý: hạn chế lớn
• Giảm tính khả chuyển (portable) của mã chương trình vì các chương trình phải được viết hoặc biên dịch lại trên các nền VXL mới

-> Dự đoán rẽ nhánh

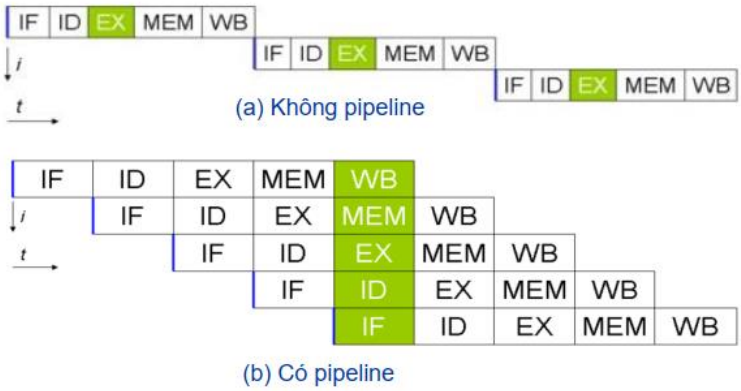
+ Có thể dự đoán lệnh đích của lệnh rẽ nhánh:
• Dự đoán đúng: nâng cao hiệu năng

• Dự đoán sai: đẩy các lệnh tiếp theo đã load và phải load lại các lệnh tại đích rẽ nhánh
• Trường hợp xấu của dự đoán là 50% đúng và 50% sai

+ Các cơ sở để dự đoán:
• Đối với các lệnh nhảy ngược (backward):
• Thường là một phần của vòng lặp
• Các vòng lặp thường được thực hiện nhiều lần
• Đối với các lệnh nhảy xuôi (forward), khó dự đoán hơn:
• Có thể là kết thúc lệnh loop
• Có thể là nhảy có điều kiện

3. So sánh: Sử dụng không sử dụng ống lệnh có gì khác nhau hay không

- Cơ chế thực hiện không pipeline, tại mỗi thời điểm chỉ có một lệnh được thực hiện và chỉ có một đơn vị chức năng của CPU làm việc, các đơn vị chức năng khác trong trạng thái chờ.
- Cơ chế thực hiện có pipeline, có nhiều lệnh đồng thời được thực hiện gối nhau trong CPU và hầu hết các đơn vị chức năng của CPU liên tục tham gia vào quá trình xử lý lệnh. Số lượng lệnh được xử lý đồng thời đúng bằng số giai đoạn thực hiện lệnh.
- Với 5 giai đoạn thực hiện lệnh, để xử lý 5 lệnh, CPU cần 9 nhịp đồng hồ với cơ chế thực hiện có pipeline, trong khi CPU cần đến 25 nhịp đồng hồ để thực hiện 5 lệnh với cơ chế thực hiện không pipeline.



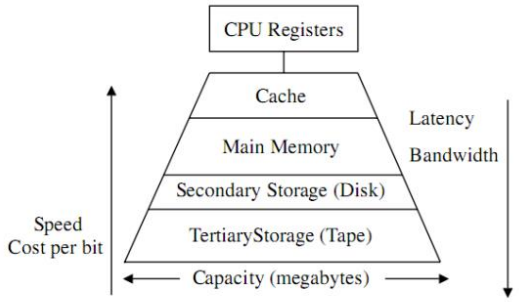
Hình 25 Thực hiện lệnh (a) không pipeline và (b) có pipeline

- Hình 25 minh hoạ cơ chế thực hiện lệnh (a) không pipeline và (b) có pipeline. Trong đó, việc thực hiện lệnh được chia thành 5 giai đoạn:

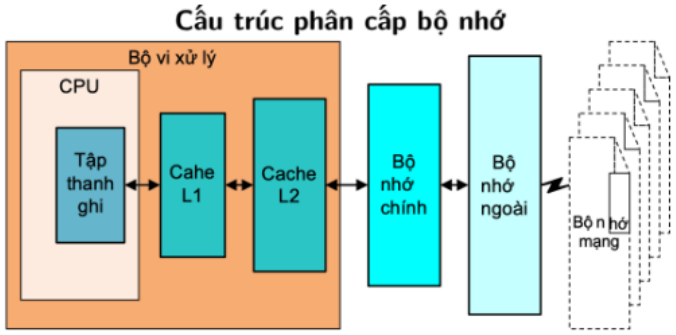
+ Instruction Fetch - IF: Đọc lệnh từ bộ nhớ (hoặc cache);
+ Instruction Decode - ID: giải mã lệnh và đọc các toán hạng;
+ Execute - EX: thực hiện lệnh; nếu là lệnh truy nhập bộ nhớ: tính toán địa chỉ bộ nhớ;
+ Memory Access - MEM: Đọc/ghi bộ nhớ; no-op nếu không truy nhập bộ nhớ; no-op là giai đoạn chờ, tiêu tốn thời gian CPU, nhưng không thực hiện thao tác có nghĩa;
+ Write Back - WB: Ghi kết quả vào các thanh ghi.

4. Sơ đồ phân cấp bộ nhớ trong thành phần máy tính, vai trò và ý nghĩa của việc phân cấp hệ thống máy tính

*** Sơ đồ phân cấp bộ nhớ**



Hình 33 Cấu trúc phân cấp hệ thống nhớ



Từ trái sang phải:

- Dung lượng tăng dần
- Tốc độ giảm dần
- Giá thành / 1 bit tăng dần

- Cấu trúc phân cấp hệ thống nhớ, gồm các phần chính:

+ Các thanh ghi của CPU (CPU Registers),
- Kích thước rất nhỏ (vài chục byte tới vài KB)
- Tốc độ rất nhanh, thời gian truy cập khoảng 0.25 ns, giá thành đắt
- Lưu trữ tạm thời dữ liệu đầu vào và ra cho các lệnh

+ Bộ nhớ cache (Cache)
- Kích thước nhỏ (64KB tới 16MB)
- Tốc độ nhanh, thời gian truy cập khoảng 1 – 5ns, giá thành đắt
- Lưu trữ lệnh và dữ liệu cho CPU
- Còn được gọi là "bộ nhớ thông minh"(smart memory)

+ Bộ nhớ chính (Main Memory)
- Kích thước lớn, dung lượng từ 256MB tới 4GB cho các hệ 32bits
- Tốc độ chậm, thời gian truy cập từ 50 – 70ns
- Lưu trữ lệnh và dữ liệu cho hệ thống và người dùng
- Giá thành rẻ

+ Bộ nhớ ngoài (Secondary / Tertiary Storage) , Bộ nhớ phụ:
- Kích thước rất lớn, dung lượng từ 20GB tới 1000GB
- Tốc độ rất chậm, thời gian truy cập khoảng 5ms
- Lưu trữ lượng dữ liệu lớn dưới dạng file trong thời gian lâu dài
- Giá thành rất rẻ

*** Vai trò:**

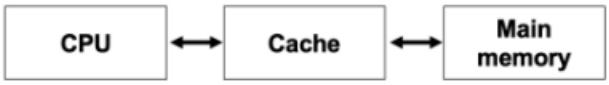
- Nâng cao hiệu năng hệ thống:
+ Dung hòa được CPU có tốc độ cao với bộ nhớ chính và bộ nhớ phụ có tốc độ thấp
+ Thời gian truy cập dữ liệu trung bình của CPU từ hệ thống bộ nhớ gần bằng thời gian truy cập cache

- Giảm giá thành sản xuất:
+ Các thành phần đắt tiền sẽ được sử dụng với dung lượng nhỏ hơn
+ Các thành phần rẻ hơn được sử dụng với dung lượng lớn hơn
+ Tổng giá thành của hệ thống nhớ theo mô hình phân cấp sẽ rẻ hơn so với hệ thống nhớ không phân cấp cùng tốc độ

4. Bộ nhớ cache là gì, vai trò, 2 nguyên lí của bộ nhớ cache: nâng cấp KG, TG

- Cache là thành phần nhớ trong sơ đồ phân cấp bộ nhớ máy tính.
+ Nó hoạt động như thành phần trung gian, trung chuyển dữ liệu từ bộ nhớ chính về CPU và ngược lại

- Vị trí của cache:
+ Với các hệ thống cũ, cache thường nằm ngoài CPU
+ Với các CPU mới, cache thường được tích hợp vào trong CPU



- Dung lượng cache thường nhỏ:
+ Với các hệ thống cũ: 16K, 32K,..., 128K
+ Với các hệ thống mới: 256K, 512K, 1MB, 2MB, . . .
- Tốc độ truy nhập của cache nhanh hơn so với tốc độ bộ nhớ chính
- Giá thành cache (tính trên bit) thường đắt hơn so với bộ nhớ chính

- Ví dụ về các thao tác của Cache: (Không cần)

1. CPU yêu cầu nội dung của ngăn nhớ
2. CPU kiểm tra trên cache với dữ liệu này
3. Nếu có, CPU nhận dữ liệu từ cache (nhanh)
4. Nếu không có, đọc Block nhớ chứa dữ liệu từ bộ nhớ chính vào cache
5. Tiếp đó chuyển dữ liệu từ cache vào CPU

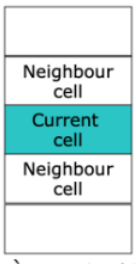
*** Vai trò:**
- Nâng cao hiệu năng hệ thống:
+ Dung hòa giữa CPU có tốc độ cao và bộ nhớ chính tốc độ thấp (giảm số lượng truy cập trực tiếp của CPU vào bộ nhớ chính)
+ Thời gian trung bình CPU truy cập hệ thống bộ nhớ gần bằng thời gian truy cập cache
- Giảm giá thành sản xuất:
+ Nếu 2 hệ thống có cùng hiệu năng thì hệ thống có cache sẽ rẻ hơn
+ Nếu 2 hệ thống cùng giá thành, hệ thống có cache sẽ nhanh hơn

*** Nguyên lí:**
- Cache được coi là bộ nhớ thông minh:

+ Cache có khả năng đoán trước yêu cầu về lệnh và dữ liệu của CPU
+ Dữ liệu và lệnh cần thiết được chuyển trước từ bộ nhớ chính về cache
–> CPU chỉ truy nhập cache –> giảm thời gian truy nhập bộ nhớ
- Cache hoạt động dựa trên 2 nguyên lý cơ bản:

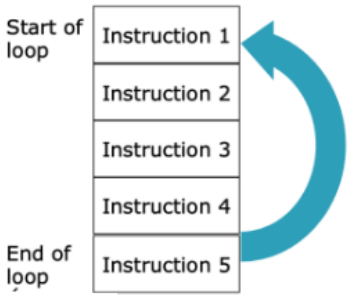
+ Nguyên lý cục bộ/ lân cận về không gian (spatial locality)

- Nếu một vị trí bộ nhớ được truy cập, thì xác suất các vị trí gần đó được truy cập trong thời gian gần tới là cao
- Áp dụng với dữ liệu và các lệnh có thứ tự tuần tự theo chương trình
- Lệnh trong chương trình thường có thứ tự tuần tự, do đó cache đọc một khối lệnh trong bộ nhớ, mà bao gồm cả các phần tử xung quanh vị trí phần tử hiện tại được truy cập.



+ Nguyên lý cục bộ/ lân cận về thời gian (temporal locality)

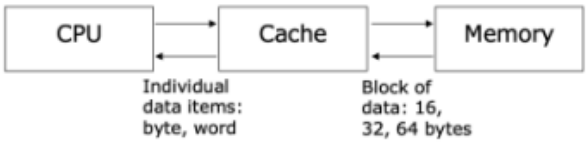
- Nếu một vị trí bộ nhớ được truy cập, thì xác suất nó được truy cập lại trong thời gian gần là cao
- Áp dụng vớicác mục dữ liệu và các lệnh trong vòng lặp
- Cache đọc khối dữ liệu trong bộ nhớ gồm tất cả các thành phần trong vòng lặp



2. Phương thức trao đổi giữa cpu và bộ nhớ cache, bộ nhớ cache và bộ nhớ chính

- Trao đổi dữ liệu:

+ CPU đọc/ ghi từng mục dữ liệu riêng biệt từ/ vào cache
+ Cache đọc/ ghi khối dữ liệu từ/ vào bộ nhớ (theo các khối, mỗi khối gồm nhiều byte kề nhau với mục đích bao phủ các mẫu dữ liệu lân cận theo không gian và thời gian)



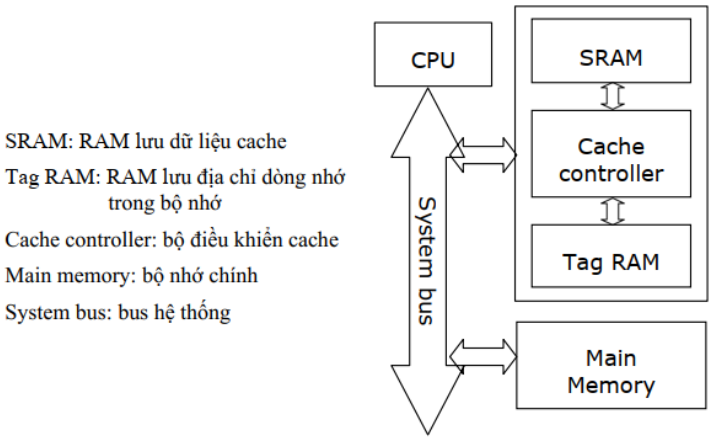
- CPU trao đổi dữ liệu với cache theo các đơn vị cơ sở như byte, từ và từ kép.
- Còn cache trao đổi dữ liệu với bộ nhớ chính theo các khối, với kích thước 16, 32 hoặc 64 bytes
- Trao đổi dữ liệu theo khối (hay mẻ) với bộ nhớ chính giúp cache tận dụng tốt hơn băng thông đường truyền và nhờ vậy có thể tăng tốc độ truyền dữ liệu.

- Hit là sự kiện CPU truy cập dữ liệu/lệnh tìm được trong cache
  + Xác suất xảy ra Hit được gọi là hệ số hit H: $0 \leq H \leq 1$
  + H càng cao thì cache càng tốt
- Miss là sự kiện CPU truy cập dữ liệu/lệnh không tìm thấy trong cache
  + Khả năng xảy ra Miss gọi là hệ số miss hay 1-H: $0 \leq (1 - H) \leq 1$
  + Hệ số miss thấp thì hiệu quả cache cao

3. 2 nguyên lí và đặc điểm của kiến trúc cấu trúc của bộ nhớ cache: look throught, look aside

- Kiến trúc cache đề cập tới việc Cache được bố trị vào vị trí nào trong mối quan hệ với CPU và Memory.
- Hai kiến trúc bố trí của Cache:

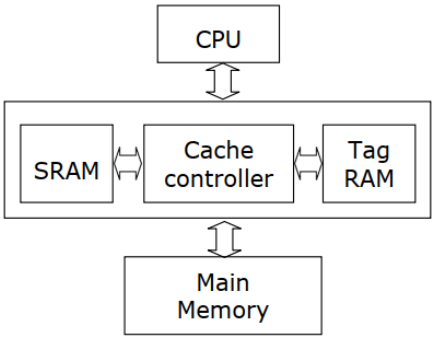
+ Kiến trúc look aside: Kiến trúc kiểu ngang khàng - bình đẳng



Hình 42 Kiến trúc cache kiểu Look Aside

- Cache và bộ nhớ cùng được kết nối tới bus hệ thống
- Cache và bộ nhớ chính "thấy" chu kỳ bus CPU tại cùng một thời điểm
- Ưu điểm:
  . Thiết kế đơn giản
  . Miss nhanh: khi CPU không tìm thấy mục dữ liệu trong cache nó đồng thời tìm trong bộ nhớ chính cùng 1 chu kỳ xung nhịp
- Nhược điểm:
  . Hit chậm: do cache kết nối với CPU sử dụng bus hệ thống có tần số làm việc không cao và băng thông hẹp

+ Kiến trúc look through: Kiến trúc kiểu phân chia - phân cấp



Hình 43 Kiến trúc cache kiểu Look Through

- Cache nằm giữa CPU và bộ nhớ chính
- Cache "thấy" chu kỳ bus CPU trước sau đó nó "truyền" lại cho bộ nhớ chính
- Ưu điểm:

  . Hit nhanh : Cache kết nối với CPU bằng bus riêng tốc độ cao và có băng thông lớn
- Nhược điểm:

  . Đắt

  . Thiết kế phức tạp

  . Miss chậm : khi CPU không tìm thấy dữ liệu trong cache nó cần tìm trong bộ nhớ tại 1 xung nhịp tiếp theo

4. Các phương pháp đọc ghi (hit, miss)

*** Thao tác đọc:**

- Trường hợp tìm thấy (hit case): mục dữ liệu yêu cầu tìm thấy trong cache
  + Mục dữ liệu được đọc từ cache vào CPU
  + Bộ nhớ chính không tham gia
- Trường hợp không tìm thấy (miss case):
  + Mục dữ liệu được đọc từ bộ nhớ vào cache
  + Sau đó dữ liệu được chuyển từ cache vào CPU
- Miss penalty: thời gian truy cập mục dữ liệu bằng tổng thời gian truy cập cache và bộ nhớ chính

*** Thao tác ghi:**

- Trường hợp hit (bản tin thuộc 1 khối trong cache)
  + Write through (ghi thẳng): mục dữ liệu được ghi vào cache và ghi vào bộ nhớ đồng thời
  + Write back (ghi trễ): mục dữ liệu trước tiên được ghi vào cache và cả dòng (block) chứa nó ở trong cache sẽ được ghi lại vào bộ nhớ sau đó, khi mà dòng đó bị thay thế
- Trường hợp miss (bản tin không có trong cache)
  + Write allocate (ghi có đọc lại): mục dữ liệu trước hết được ghi vào bộ nhớ sau đó cả dòng chứa nó sẽ được đọc vào cache
  + Write non-allocate (ghi không đọc lại): mục dữ liệu chỉ được ghi vào bộ nhớ

5. Chính sách thay thế bộ nhớ cache: FIFO, LRU

* Phải thay thế dòng cache vì: (không)

- Ánh xạ từ 1 dòng trong bộ nhớ –> dòng trong bộ nhớ cache thường là ánh xạ nhiều –> một
- Nhiều dòng bộ nhớ chia sẻ một dòng cache –> các dòng bộ nhớ được nạp vào cache sử dụng một thời gian và được thay thế bởi dòng khác theo yêu cầu thông tin phục vụ CPU.

* Chính sách thay thế (replacement policies): (không)
- Xác định cách thức lựa chọn các dòng trong cache để thay thế khi có dòng mới từ bộ nhớ cần chuyển vào

* Chiến lược thay thế dòng cache gồm 3 chiến lược chính:

- Vào trước ra trước FIFO (First In First Out)

+ Dựa trên nguyên lý FIFO: dòng cache được đọc vào cache trước sẽ được chọn để thay trước
+ Tỷ lệ miss thấp hơn so với phương pháp ngẫu nhiên
+ Tỷ lệ miss vẫn cao vì phương pháp này vẫn chưa thực sự xem xét tới block nào đang thực sự được sử dụng

- Một dòng cache "cũ" có thể vẫn đang được sử dụng

+ Cài đặt phức tạp vì cần thêm mạch để giám sát thứ tự nạp các dòng bộ nhớ vào cache

- Thay thế các dòng ít được sử dụng gần đây nhất LRU (Least Recently Used)

+ Các dòng cache ít được sử dụng nhất trong thời gian gần đây sẽ được chọn để thay
+ Xét tới các dòng đang được sử dụng
+ Tỷ lệ miss thấp nhất so với phương pháp ngẫu nhiên và FIFO
+ Cài đặt phức tạp vì cần thêm mạch để giám sát tần suất sử dụng các dòng cache

- Thay thế ngẫu nhiên (random replacement)

+ Các dòng trong cache được chọn ngẫu nhiên để thay
+ Đơn giản
+ Tỷ lệ miss cao vì phương pháp này không xét tới dòng cache nào đang thực sự được sử dụng

- Nếu một dòng cache đang được sử dụng mà bị thay thế thì sự kiện miss xảy ra và cần phải đọc lại vào cache

6. Các tham số ảnh hưởng đến hiệu năng bộ nhớ cache là gì ? Ví dụ kích thước của cache, cách tạo ra cache,,cách .. bus(3 yto)

* Các yếu tố ảnh hưởng đến hiệu năng cache bao gồm:

- Kích thước cache

+ Kích thước cache lớn:

- Có thể lưu trữ nhiều khối dữ liệu trong bộ nhớ hơn
- Giảm tần suất trao đổi các khối dữ liệu của chương trình khác nhau giữa bộ nhớ và cache
- Cache lớn chậm hơn cache nhỏ:
  . Tìm kiếm vị trí bộ nhớ trong không gian lớn hơn
- Xu hướng: cache càng ngày càng lớn
  . Hỗ trợ đa nhiệm (multi-tasking) tốt hơn
  . Hỗ trợ tốt hơn cho xử lý song song
  . Hỗ trợ tốt hơn cho các hệ thống CPU đa nhân

- Cache nhiều mức

+ Cache nhiều mức có thể dung hòa tốc độ của CPU và MEM tốt hơn cache một mức.

	CPU	L1	L2	L3	Bộ nhớ chính
Cache 3 mức:	1ns	5ns	15ns	30ns	60ns
Cache 1 mức:	1ns	5ns			60ns

+ Thực tế, cache thường có 2 mức: L1 và L2. Một số cache có 3 mức: L1, L2 và L3

+ Giảm giá thành hệ thống nhớ

- Sự phân chia cache

+ Cache có thể được chia thành cache lệnh và cache dữ liệu để hiệu năng tốt hơn vì:

- Dữ liệu và lệnh khác nhau về tính cục bộ :
  . Dữ liệu có tính chất cục bộ về thời gian hơn
  . Lệnh có tính chất cục bộ về không gian hơn
- Cache lệnh chỉ hỗ trợ thao tác đọc còn cache dữ liệu hỗ trợ cả 2 thao tác đọc ghi nên Dễ tối ưu cache hơn
- Hỗ trợ nhiều thao tác đọc/ ghi cùng lúc nên giảm xung đột tài nguyên
- Kết hợp các chức năng khác như giải mã trước lệnh vào trong cache lệnh nên xử lý lệnh tốt hơn

+ Trong thực tế, hầu hết cache L1 được chia thành 2 phần:

- I_cache (Instruction cache): cache lệnh
- D_cache (Data cache): cache dữ liệu.

+ Các cache mức cao hơn không được chia:

- Chia cache L1 có lợi ích cao nhất vì nó gần CPU nhất. CPU đọc/ ghi trực tiếp lên cache L1. Cache L1 cần hỗ trợ nhiều lệnh truy nhập đồng thời và các biện pháp tối ưu hóa
- Chia các cache mức cao hơn không đem lại hiệu quả cao và làm phức tạp trong hệ thống điều khiển cache

Chương 5:

1. Thanh ghi:

*** Các thanh ghi đoạn:**

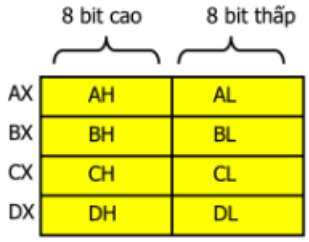
- CS (Code Segment): Thanh ghi đoạn mã. CS chứa địa chỉ bắt đầu đoạn mã
- DS (Data Segment): Thanh ghi đoạn dữ liệu. DS chứa địa chỉ bắt đầu đoạn dữ liệu
- SS (Stack Segment): Thanh ghi đoạn ngăn xếp. SS chứa địa chỉ bắt đầu đoạn ngăn xếp
- ES (Extra Segment): Thanh ghi đoạn dữ liệu mở rộng. ES chứa địa chỉ bắt đầu đoạn dữ liệu mở rộng.

*** Các thanh ghi của đa năng 8086/8088**

- 4 thanh ghi 16 bits:

+ AX (accumulator): Thanh ghi tổng, thường dùng để lưu kết quả
+ BX (base): Thanh ghi cơ sở, thường dùng chứa địa chỉ ô nhớ
+ CX (count): Thanh ghi đếm, thường dùng làm con đếm cho các lệnh lặp (Loop), chứa số lần dịch hoặc quay trong các lệnh dịch và quay thanh ghi
+ DX (data): Thanh ghi dữ liệu, cùng AX chứa dữ liệu trong các phép tính nhân/chia số 16 bit hoặc chứa địa chỉ cổng trong các lệnh vào ra dữ liệu trực tiếp (IN/OUT)

- Hoặc 8 thanh ghi 8 bits: AH AL, BH, BL, CH, CL, DH, DL



*** Các thanh ghi con trỏ và chỉ số:**

- IP (Instruction Pointer): Con trỏ lệnh (còn gọi là bộ đếm chương trình PC). IP luôn chứa địa chỉ của lệnh tiếp theo sẽ được thực hiện;
+ Thanh ghi IP (con trỏ lệnh) chứa địa chỉ offset của lệnh tiếp theo.
+ CS:IP chứa địa chỉ logic của lệnh tiếp theo đó.
- SP (Stack Pointer): con trỏ ngăn xếp. SP luôn chứa địa chỉ đỉnh ngăn xếp SS:SP
- BP (Base Pointer): Con trỏ cơ sở, chứa địa chỉ của dữ liệu trong đoạn ngăn xếp SS hoặc các đoạn khác SS:BP
- SI (Source Index): Thanh ghi chỉ số nguồn. SI thường dùng chứa địa chỉ ô nhớ nguồn ở đoạn dữ liệu DS như trong các lệnh chuỗi trong các thao tác chuyển dữ liệu DS:SI
- DI (Destination Index): Thanh ghi chỉ số đích. DI thường dùng chứa địa chỉ ô nhớ đích ở đoạn dữ liệu DS trong các thao tác chuyển dữ liệu DS:DI
+ SI và DI có thể được sử dụng như thanh ghi đa năng
+ 80386 trở lên 32 bit: EIP, EBP, ESP, EDI, ESI

*** Con trỏ lệnh và thanh ghi cờ:**

- IP (Instruction Pointer): Con trỏ lệnh (còn gọi là bộ đếm chương trình PC). IP luôn chứa địa chỉ của lệnh tiếp theo sẽ được thực hiện;
+ Thanh ghi IP (con trỏ lệnh) chứa địa chỉ offset của lệnh tiếp theo.
+ CS:IP chứa địa chỉ logic của lệnh tiếp theo đó.

- FR (Flag Register) hoặc SR (Status Register): Thanh ghi cờ hoặc thanh ghi trạng thái.

+ Cờ trạng thái: Các bit của FR lưu các trạng thái của kết quả phép toán ALU thực hiện
+ Cờ điều khiển: trạng thái của tín hiệu điều khiển.

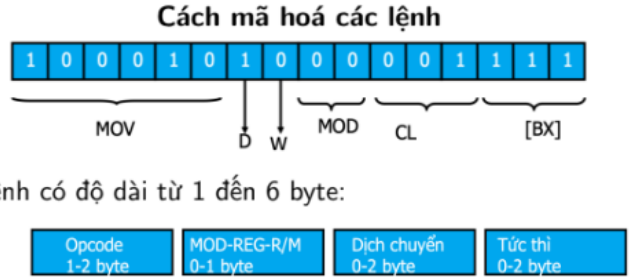
+ Các cờ trạng thái:
- CF (Carry): cờ nhớ. CF = 1 là phép tính có nhớ;
  ▶ CF = 0 là phép tính không nhớ. Nếu phép cộng có nhớ ra khỏi bit cao nhất hay phép toán trừ có mượn ra khỏi bit cao nhất thì CF được thiết lập (báo tràn với số nguyên không dấu).
- AF (Auxiliary): cờ nhớ phụ. AF = 1 là có nhớ phụ;
  ▶ AF = 0 là không nhớ phụ. Nếu phép cộng có nhớ từ bit 3 sang bit 4 hoặc phép trừ có mượn từ bit 3 sang bit 4 thì cờ AF được thiết lập.
- PF (Parity): cờ chẵn lẻ. PF = 1 khi tổng số bit 1 trong kết quả là lẻ và PF = 0 khi tổng số bit 1 trong kết quả là chẵn
- OF (Overflow): cờ tràn. OF = 1 khi kết quả bị tràn số. Nếu cộng 2 số cùng dấu mà kết quả có dấu ngược lại thì OF được thiết lập (báo tràn với số nguyên có dấu).
- ZF (Zero): cờ zero. ZF = 1 khi kết quả bằng 0; ngược lại ZF = 0
- SF (Sign): cờ dấu. SF = 1 khi kết quả âm.

+ Các cờ điều khiển:
- DF (Direction): cờ hướng, chỉ hướng tăng giảm địa chỉ với các lệnh chuyển dữ liệu.
  ▶ DF = 0 là địa chỉ tăng.
  ▶ DF = 1 là địa chỉ giảm.

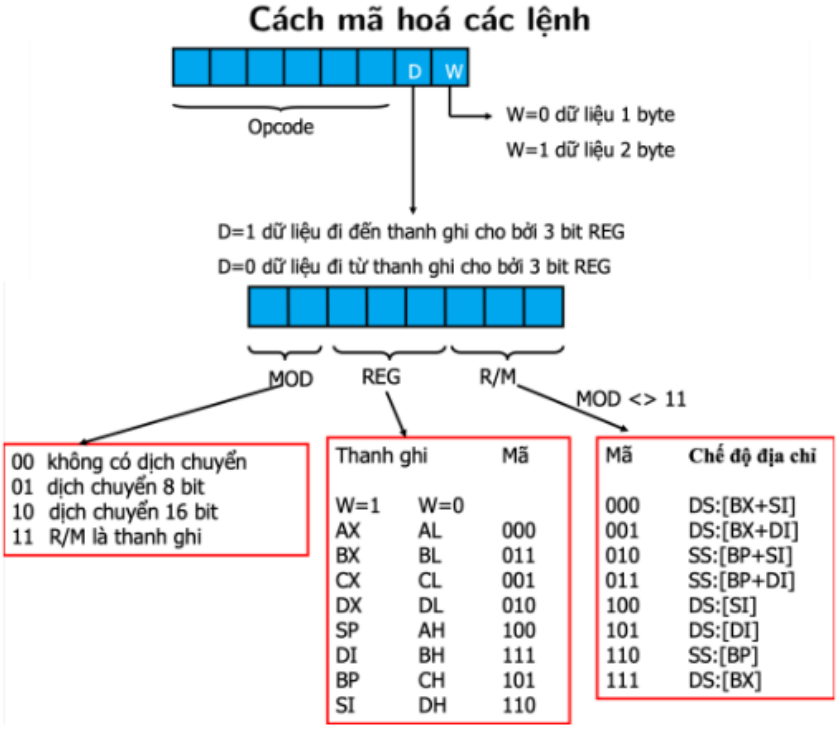
• TF (Trap/Trace): cờ bẫy/lần vết, được dùng khi gỡ rối chương trình. TF = 1 là CPU ở chế độ chạy từng lệnh (chế độ gỡ rối chương trình).
• IF (Interrupt): cờ ngắt.
▶ Nếu IF = 1 thì bộ vi xử lý cho phép ngắt với yêu cầu ngắt đưa đến chân tín hiệu INTR (Interrupt Request) của bộ vi xử lý.
▶ Nếu IF = 0 thì cấm ngắt.

2. Các thành phần trong 1 lệnh của 8086, 8088:

* 6 thành phần: OPCODE, D, W, MOD, REG, R/M



- Opcode: mã lệnh gồm 6 bít; Mã lệnh của MOV là 100010
- D: bít hướng, chỉ hướng vận chuyển dữ liệu;
  + D = 1: dữ liệu đi đến thanh ghi cho bởi 3 bit REG
  + D = 0: dữ liệu đi ra từ thanh ghi cho bởi 3 bit REG
- W: bít chỉ độ rộng toán hạng;
  + W=0: toán hạng 1 byte (8 bit);
  + W=1: toán hạng 2 bytes (16 bit)



(Không cần):

Cách mã hoá các lệnh: Mã hoá các thanh ghi

Các thanh ghi đoạn	Mã thanh ghi
CS	01
DS	11
ES	00
SS	10

Các thanh ghi		Mã thanh ghi
W=1	W=0	
AX	AL	000
BX	BL	011
CX	CL	001
DX	DL	010
SP	AH	100
DI	BH	111
BP	CH	101
SI	DH	110

3. 7 chế độ địa chỉ (Không cần ví dụ)

* Chế độ địa chỉ (Addressing Mode) là cách CPU tổ chức và lấy dữ liệu cho các toán hạng khi thực hiện lệnh.

* Một bộ vi xử lý có thể có nhiều chế độ địa chỉ. Vi xử lý 8086/8088 có 7 chế độ địa chỉ:

1. Chế độ địa chỉ thanh ghi

- Chế độ thanh ghi sử dụng các thanh ghi bên trong cpu như là các toán hạng để chứa dữ liệu cần thao tác.
- Cả toán hạng gốc và đích đều là các thanh ghi.
- Tốc độ thực hiện lệnh cao
- Ví dụ:
  - MOV BX, DX; BX <– DX
  - MOV DS, AX; DS <– AX
  - ADD AL, DL; AL <– AL + DL
  - MOV AL, BX ; không hợp lệ vì 2 thanh ghi có kích thước khác nhau
  - MOV ES, DS ; không hợp lệ (segment to segment)
  - MOV CS, AX ; không hợp lệ vì CS không dùng làm thanh ghi đích

2. Chế độ địa chỉ tức thì

- Toán hạng đích là một thanh ghi hay một ô nhớ
- Toán hạng gốc là một hằng số
- Dùng để nạp hằng số vào thanh thi (trừ thanh ghi đoạn và thanh cờ) hoặc vào ô nhớ trong đoạn dữ liệu DS
- Ví dụ:
  - MOV CL, 200; CL <–200
  - MOV AX, 0ff0h; AX <– 0ff0h
  - MOV [BX], 200; Chuyển 200 vào ô nhớ có địa chỉ là DS:BX
  - MOV AL, ‘A’ ; Copy mã ASCII của A vào thanh ghi AL
  - MOV DS, 0FF0H ; không hợp lệ

3. Chế độ địa chỉ trực tiếp

- Một toán hạng là một hằng số biểu diễn địa chỉ lệch (offset) của ô nhớ
- Toán hạng còn lại có thể là thanh ghi (không được là ô nhớ)
- Ví dụ:

• MOV CL, 200; CL <–200
• MOV AX, 0ff0h; AX <– 0ff0h
• MOV [BX], 200; Chuyển 200 vào ô nhớ có địa chỉ là DS:BX
• MOV AL, 'A' ; Copy mã ASCII của A vào thanh ghi AL
• MOV DS, 0FF0H ; không hợp lệ

4. Chế độ địa chỉ gián tiếp thanh ghi

- Một toán hạng là một thanh ghi chứa địa chỉ lệch của ô nhớ
- Toán hạng còn lại có thể là thanh ghi
- Ví dụ:

• MOV AL, [BX]; AL <– [DS:BX]
• MOV AL, [BP]; AL <– [SS:BP]
• MOV [DI], AX; Copy nội dung của AX vào 2 ô nhớ liên tiếp DS:DI và DS:(DI+1)

5. Chế độ địa chỉ tương đối cơ sở

- Một toán hạng là địa chỉ của ô nhớ.
  + Địa chỉ của ô nhớ được tạo bởi thanh ghi cơ sở như BX (đoạn DS) hoặc BP (đoạn SS) và 1 hằng số
  + Hằng số là giá trị dịch chuyển(displacement) để tính địa chỉ hiệu dụng của các toán hạng trong vùng nhớ DS và SS.
- Toán hạng còn lại có thể là thanh ghi (Không được là ô nhớ)
- Ví dụ:

• MOV AL, [BX+100]; AL <– [DS: BX+100]
• MOV AL, [BP+200]; AL <– [SS: BP+200]
• MOV AL, [BP]+200; Cách viết khác của lệnh trên

6. Chế độ địa chỉ tương đối chỉ số

- Một toán hạng là địa chỉ của ô nhớ.
  + Địa chỉ của ô nhớ tạo thanh ghi cơ sở SI hoặc DI và một hằng số.
  + Hằng số biểu diễn các giá trị dịch chuyển (displacement) được dùng để tính địa chỉ hiệu dụng của các toán hạng trong các vùng nhớ DS.
- Toán hạng còn lại có thể là thanh ghi (không được là ô nhớ)
- Ví dụ:

• MOV AL, [SI+100]; AL <– [DS: SI+100]
• MOV AL, [DI+200]; AX <– [DS: DI+200]
• MOV AX, [SI]+10; Copy nội dung 2 ô nhớ liên tiếp có địa chỉ DS:SI+10 và DS:SI+11 vào AX
• MOV AX, [SI+10]; Cách viết khác của lệnh trên

7. Chế độ địa chỉ tương đối chỉ số cơ sở

- Một toán hạng là địa chỉ của ô nhớ
  + Địa chỉ của ô nhớ tạo bởi các thanh ghi BX+SI/DI (đoạn DS) hoặc BP+SI/DI (đoạn SS) và một hằng số.
  + Hằng số biểu diễn các giá trị dịch chuyển (displacement) để tính địa chỉ hiệu dụng của các toán hạng trong các vùng nhớ DS và SS.
- Toán hạng còn lại có thể là thanh ghi (không được là ô nhớ)
- Ví dụ:

• MOV AL, [BX+SI+100]; AL <– [DS:BX+SI+100]

• MOV AX, [BX] [SI]+8; Copy nội dung 2 ô nhớ liên tiếp có địa chỉ DS:BX+SI+8 và DS:BX+SI+9 vào AX
• MOV AX, [BX+SI+8]; Cách viết khác của lệnh trên

Chương 6:

1. Vai trò, phương pháp vào ra dữ liệu là gì

- Kỹ thuật trao đổi dữ liệu giữa máy vi tinh và các thiết bị ngoại vi được gọi là vào/ra hay I/O (Input/Output).
- Thiết bị liên lạc với máy vi tính qua các giao tiếp vào/ra.
- Để hai bên có thể liên lạc được với nhau cần có các mạch giao tiếp giữa thiết bị vào/ra và máy tính. Giao tiếp cung cấp trao đổi dữ liệu vào/ra qua buýt vào/ra. Buýt này thông thường chuyển tại 3 loại tín hiệu:
+ Địa chỉ thiết bị
+ Dữ liệu
+ Lệnh.
- Có ba phương pháp trao đổi dữ liệu giữa máy vi tính và các thiết bị vào/ra:
+ Vào/ra lập trình (programmed I/O) hay vào ra thăm dò
    Vi xử lý chạy một chương trình thực hiện toàn bộ các trao đổi dữ liệu giữa vi xử lý và các thiết bị bên ngoài
    Đặc tính: thiết bị thực hiện các chức năng được chỉ định bởi chương trình bên trong bộ nhớ của vi xử lý (Vi xử lý điều khiển hoàn toàn các trao đổi dữ liệu)
+ Vào/ra bằng ngắt
    Thiết bị có thể bắt vi xử lý dừng việc thực hiện chương trình hiện thời để thiết bị có thể chạy chương trình khác gọi là chương trình phục vụ ngắt
    Sau khi kết thúc chương trình này, câu lệnh trở về từ ngắt để trả lại quyền điều khiển cho chương trình bị ngắt
+ Vào ra truy nhập trực tiếp bộ nhớ (Direct Memory Access DMA)
    Dữ liệu có thể được trao đổi giữa bộ nhớ của máy tính với thiết bị như ổ cứng mà không cần sự can thiệp của vi xử lý.
    Thường cần sử dụng vi mạch đặc biệt gọi là vi mạch DMA.

* Vai trò của phương pháp vào/ra dữ liệu là

- Cung cấp cách thức liên lạc giữa máy tính và các thiết bị ngoại vi, cho phép người dùng nhập liệu, hiển thị kết quả và thực hiện các chức năng khác thông qua các thiết bị vào/ra. Phương pháp vào/ra dữ liệu đảm bảo tính tiện lợi và hiệu quả trong việc kết nối máy tính với thế giới bên ngoài. Các thiết bị vào/ra phổ biến gồm có bàn phím, màn hình, máy in và ổ đĩa cứng..

2. Đưa ra thế nào là vào ra thăm dò, vào ra bằng ngắt..vào ra bằng truy cập trực tiếp với bộ nhớ. So sánh 2 / nh phương pháp này với nhau và đưa các VD minh họa cho 3 pp này

*** Vào ra bằng thăm dò:**

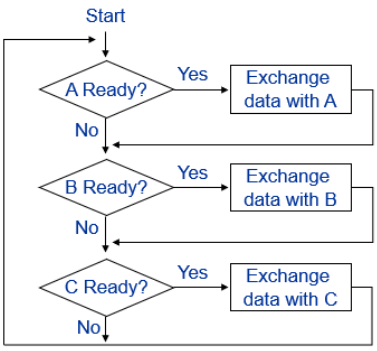
- Cơ chế vào ra bằng thăm dò:
+ CPU quản lý danh sách các thiết bị vào ra kèm theo địa chỉ các cổng giao tiếp;
+ Các thiết bị vào ra định kỳ cập nhật trạng thái sẵn sàng làm việc của mình lên các bít cờ trạng thái vào ra của mình;
+ CPU định kỳ lần lượt "quét" các thiết bị vào ra để "đọc" các bit cờ trạng thái vào ra;

- Nếu gặp một thiết bị sẵn sàng làm việc, 2 bên tiến hành trao đổi dữ liệu;
- Trao đổi dữ liệu xong, CPU tiếp tục quét thiết bị khác.

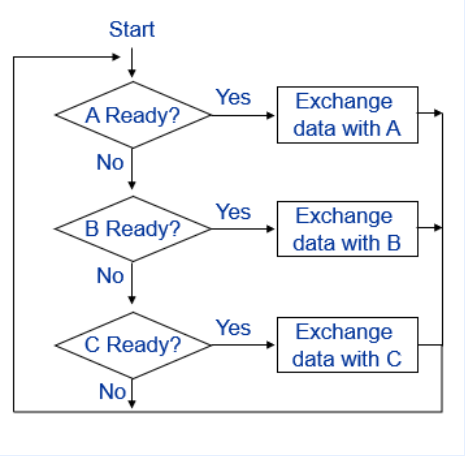
+ CPU là bên chủ động trong quá trình trao đổi dữ liệu

- Ưu điểm:
+ Đơn giản, dễ cài đặt
+ Có thể được cài đặt bằng phần mềm
- Nhược điểm:
+ Hiệu quả thấp do CPU tốn nhiều thời gian để thăm dò các thiết bị
+ Không thực sự khả thi khi có nhiều thiết bị trong danh sách thăm dò
- Ứng dụng của vào ra bằng thăm dò:
+ Thăm dò thường được sử dụng khi hệ thống khởi động: CPU thăm dò hầu hết các thiết bị để xác lập cấu hình
+ Thăm dò được sử dụng trong quá trình hoạt động với các thiết bị rời (removable) như ổ đĩa CD/DVD, ổ mềm, ...

- Vào ra bằng thăm dò – không ưu tiên
+ Ba thiết bị A, B, C được thăm dò không ưu tiên
+ CPU quét tất cả các thiết bị trong một chu trình thăm dò
+ CPU có thể trao đổi dữ liệu với nhiều hơn 1 thiết bị trong một chu trình thăm dò
+ Các thiết bị được "thăm" lần lượt, không phụ thuộc vào thiết bị đứng trước chu trình.
+ CPU bắt đầu 1 chu trình thăm dò mới sau khi đã quét qua tất cả các thiết bị.



- Vào ra bằng thăm dò – có ưu tiên
+ Ba thiết bị A, B, C được thăm dò có ưu tiên theo thứ tự: A, B, C;
+ CPU có thể quét tất cả các thiết bị trong một chu trình thăm dò
+ CPU chỉ trao đổi dữ liệu với tối đa 1 thiết bị trong một chu trình thăm dò
+ Các thiết bị có mức ưu tiên cao luôn được thăm trước;
+ Các thiết bị có mức ưu tiên thấp chỉ được thăm nếu các thiết bị đứng trước nó không sẵn sàng.
+ CPU bắt đầu 1 chu trình thăm dò mới ngay sau khi trao đổi dữ liệu với một thiết bị.



*** Vào ra bằng ngắt**

- Ngắt là gì?

+ Ngắt (Interrupt) là một sự kiện mà CPU tạm dừng thực hiện một chương trình để thực hiện một đoạn chương trình khác theo yêu cầu từ bên ngoài;

+ Thông thường các yêu cầu từ bên ngoài thường xuất phát từ các thiết bị vào ra. Các yêu cầu này gọi là các yêu cầu ngắt;

+ Đoạn chương trình CPU thực hiện trong thời gian ngắt được gọi là chương trình con phục vụ ngắt (CTCPVN).

- Các CTCPVN là các đoạn chương trình:

+ Được viết sẵn và lưu trong ROM;

+ Mỗi CTCPVN có nhiệm vụ riêng và thường là đảm nhiệm việc trao đổi dữ liệu với thiết bị vào ra.

- Khi nào CPU kiểm tra và xử lý ngắt: CPU kiểm tra yêu cầu ngắt tại chu kỳ đồng hồ cuối cùng của chu kỳ lệnh.

- Phân loại ngắt

+ Ngắt cứng: là các ngắt được kích hoạt bởi các bộ phận phần cứng gửi đến chân NMI và INTR của CPU; gồm:

- Ngắt không che được NMI (Non-Maskable Interrupt): ngắt gửi đến chân NMI của CPU, không chịu sự ảnh hưởng của cờ ngắt; VD: ngắt Reset;
- Ngắt che được INTR (Maskable Interrupt): ngắt gửi đến chân INTR của CPU, chịu sự chi phối của cờ ngắt; Cờ IF=1 → cho phép ngắt, IF=0 → cấm ngắt.

+ Ngắt mềm: là các ngắt được kích hoạt bởi các chương trình thông qua lệnh gọi ngắt INT <N>. N là số hiệu ngắt, N=0-255.

+ Các ngắt ngoại lệ: là các ngắt do các lỗi nảy sinh trong quá trình hoạt động của CPU:

- Ngắt chia cho 0 (divide by zero)
- Ngắt do tràn (overflow)

+ Trật tự ưu tiên trong xử lý các yêu cầu ngắt

- Các yêu cầu ngắt được gán một mức ưu tiên
- Khi nhận được nhiều yêu cầu ngắt đồng thời, CPU sẽ xử lý chúng theo mức ưu tiên định trước

+ Mức ưu tiên các yêu cầu ngắt (từ cao nhất đến thấp nhất)

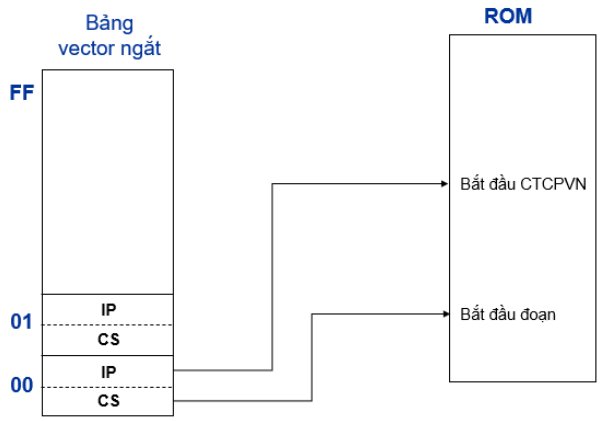
1) Ngắt nội bộ: INT 0 (chia cho 0), INT N (N<>0)
2) Ngắt không che được NMI
3) Ngắt che được INTR

4) Ngắt chạy từng lệnh: INT 1

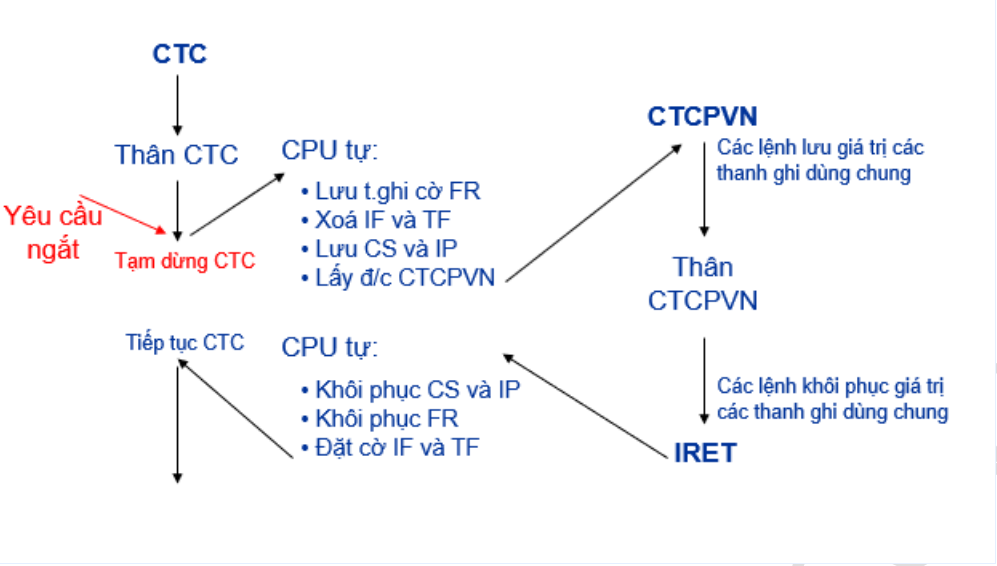
- Ưu điểm
+ Hiệu quả hơn vào ra bằng thăm dò, do CPU không phải thăm dò từng thiết bị
- Nhược điểm
+ Phức tạp hơn vào ra bằng thăm dò
+ Cần mạch phần cứng để điều khiển ngắt
- Bên chủ động trong vào ra bằng ngắt:
+ Thiết bị vào ra

- Bảng vector ngắt (không)
+ Vi xử lý 8086/8088 có 256 ngắt được đánh số từ 0-255
+ Một vector ngắt gồm các thông tin:
- Số hiệu ngắt N, N=0-255 hoặc 00-FFH
- Địa chỉ đầy đủ chương trình con phục vụ ngắt (CTCPVN) lưu trong bộ nhớ ROM. Địa chỉ đầy đủ gồm:
  - Địa chỉ đoạn (CS)
  - Địa chỉ lệch (IP)

+ Bảng vector ngắt lưu thông tin về 256 vector ngắt. Mỗi bản ghi của bảng gồm các thông tin:
- Số hiệu ngắt
- Địa chỉ đoạn và địa chỉ lệch của CTCPVN.



- Chu trình xử lý ngắt (không)



1. **Khi nhận được yêu cầu ngắt, CPU thực hiện các việc:**
   a. Hoàn tất lệnh đang thực hiện của chương trình chính (CTC)
   b. Lưu giá trị của thanh ghi cờ FR vào ngăn xếp
   c. Xoá cờ ngắt IF và cờ bẫy TF
   d. Lưu giá trị của các t.ghi CS và IP vào ngăn xếp
   e. Từ số hiệu ngắt N, lấy địa chỉ của CTCPVN từ bảng vector ngắt
2. **Nạp địa chỉ của CTCPVN vào CS và IP, CPU thực hiện CPCPVN, gồm:**
   a. Lưu giá trị các thanh ghi dùng chung vào ngăn xếp
   b. Thực hiện mã chính của CTCPVN
   c. Khôi phục giá trị các thanh ghi dùng chung
3. **Gặp lệnh IRET kết thúc CTCPVN, CPU thực hiện các việc:**
   a. Khôi phục giá trị của CS và IP
   b. Khôi phục giá trị của thanh ghi cờ FR
   c. Đặt cờ ngắt IF và cờ bẫy TF
4. **CPU tiếp tục thực hiện lệnh tiếp theo của CTC (nằm sau lệnh xảy ra ngắt).**

- Chu trình vào ra bằng ngắt(<u>không</u>)

1. Thiết bị vào ra có nhu cầu trao đổi dữ liệu, gửi yêu cầu ngắt đến chân tín hiệu INTR của CPU;
2. Khi nhận được yêu cầu ngắt, CPU thực hiện các việc:
   a. Hoàn tất lệnh đang thực hiện của chương trình chính (CTC)
   b. Lưu giá trị của thanh ghi cờ FR vào ngăn xếp
   c. Xoá cờ ngắt IF và cờ bẫy TF
   d. Lưu giá trị của các t.ghi CS và IP vào ngăn xếp
   e. Gửi tín hiệu xác nhận ngắt đến thiết bị vào ra qua chân tín hiệu INTA
3. Nhận được hiệu xác nhận ngắt của CPU, thiết bị vào ra gửi số hiệu ngắt N đến CPU
4. Nhận được số hiệu ngắt N, CPU lấy địa chỉ của CTCPVN tương ứng từ bảng vector ngắt
5. Nạp địa chỉ của CTCPVN vào CS và IP, CPU thực hiện CPCPVN, gồm:
   a. Lưu giá trị các thanh ghi dùng chung vào ngăn xếp
   b. Thực hiện mã chính của CTCPVN: đồng thời thực hiện việc trao đổi dữ liệu với thiết bị vào ra
   c. Khôi phục giá trị các thanh ghi dùng chung
6. Gặp lệnh IRET kết thúc CTCPVN, CPU thực hiện các việc:
   a. Khôi phục giá trị của CS và IP

b. Khôi phục giá trị của thanh ghi cờ FR
c. Đặt cờ ngắt IF và cờ bẫy TF

7. CPU tiếp tục thực hiện lệnh tiếp theo của CTC (nằm ngay sau lệnh xảy ra ngắt).

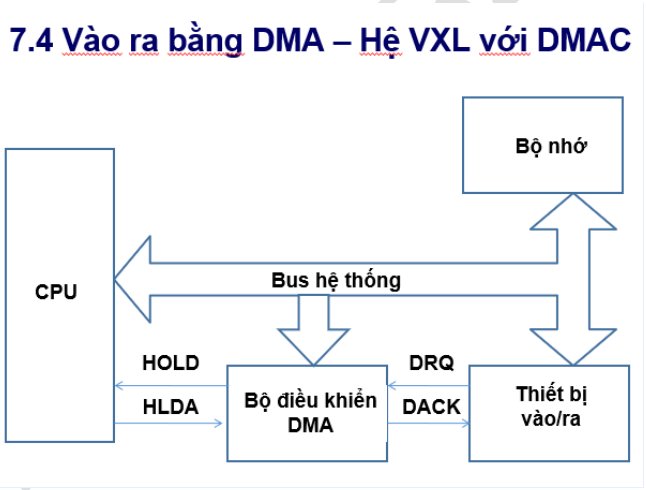
*** Vào ra bằng DMA**

- DMAC (DMA Controller) thay mặt CPU điều khiển quá trình trao đổi dữ liệu trực tiếp giữa thiết bị vào ra và bộ nhớ;
- DMA có tốc độ cao hơn nhiều lần so với vào ra bằng thăm dò và ngắt.

+ Ví dụ, với VXL 8088:

- Vào ra bằng DMA mất 4 chu kỳ đồng hồ để chuyển 1 byte thiết bị ngoại vi vào bộ nhớ;
- Vào ra thông qua CPU mất 39 chu kỳ đồng hồ để chuyển 1 byte thiết bị ngoại vi vào bộ nhớ:

```
                                    ;Số chu kỳ đồng hồ
LAP:        MOV AL, [SI];           10
            OUT PORT, AL;           10
            INC SI;                  2
            LOOP LAP;               17
                    ; Cộng:         39 chu kỳ
```



- Ưu điểm:

+ Hiệu suất rất cao do dữ liệu được trao đổi trực tiếp theo khối giữa thiết bị vào ra và bộ nhớ không thông qua CPU

- Nhược điểm:

+ Phức tạp hơn vào ra bằng thăm dò và ngắt
+ Cần mạch phần cứng để điều khiển quá trình DMA

- Bên chủ động trong vào ra bằng DMA:

+ Thiết bị vào ra

- Chu trình vào ra bằng DMA(không)

1. Thiết bị vào ra có yêu cầu trao đổi dữ liệu gửi yêu cầu DRQ đến CPU thông qua DMAC;
2. DMAC chuyển yêu cầu DRQ thành HRQ và gửi đến chân tín hiệu HOLD của CPU;

3. Nhận được yêu cầu sử dụng bus HRQ, CPU:
    a. Gửi các tham số điều khiển trao đổi dữ liệu và tín hiệu xác nhận yêu cầu sử dụng bus HACK cho DMAC qua chân tín hiệu HLDA;
    b. Tự tách ra khỏi bus hệ thống (100% các tín hiệu của bus A và D và một số tín hiệu của bus C)
4. Nhận được HACK, DMAC chiếm quyền điều khiển bus hệ thống và gửi tín hiệu xác nhận DACK cho thiết bị vào ra;
5. DMAC điều khiển quá trình trao đổi dữ liệu trực tiếp giữa thiết bị vào ra và bộ nhớ;
6. Kết thúc quá trình DMA, DMAC trả quyền điều khiển bus cho CPU.