

1) Các thế hệ máy tính				
Thế hệ	Thời gian	Công nghệ	Đại diện	
0	Cuối XIX	Non-digital computers	Bàn tính	
1	1940-1956	Vacuum tubes (Đèn chân không)	ENIAC, UNIVAC	
2	1956-1963	Transistors (Linh kiện bán dẫn)	IBM 7094, IBM 1401	
3	1964-1971	Integrated Circuits (Vi mạch tích hợp)	IBM System/360	
4	1971-nay	Microprocessors (Vi xử lý)	Intel 4004, 80486DX2	
5	Trương lai	Parallel Processing (Xử lý song song)	Siêu máy tính	

**2) Định luật MOORE:** Nội dung: Số lượng transistor trên một chip sẽ tăng gấp đôi sau mỗi 18-24 tháng, trong khi chi phí sản xuất không đổi hoặc giảm. (Gordon Moore)

**3) Các thành phần cơ bản của máy tính:**

- Phần cứng: CPU, Mainboard, RAM, Ổ cứng (HDD/SSD), Nguồn điện, Khe mở rộng PCI/PCIe, Thiết bị nhập/xuất: Màn hình, bàn phím, chuột, ổ quang.
- Mô hình 5 thành phần cơ bản (Von Neumann): Input, Output, Processor, Memory, Datapath

**4) Hai thành phần bên trong vi xử lý:**

*Control Unit:* Điều phối hoạt động của CPU, giải mã lệnh và điều khiển luồng dữ liệu.
*Datapath:* Thực hiện các phép toán số học và logic (thông qua ALU - Arithmetic Logic Unit), Bao gồm các thanh ghi (registers) và bus dữ liệu.

Mối quan hệ: Control Unit chỉ đạo → Data Path thực thi.

**5) Các khái niệm cơ bản:**

- Wafer* (Đĩa silicon): Tấm silicon mỏng đã được cắt vát để tạo ra vi mạch, kích thước trung bình 25.4mm(1inch) - 200mm(7.9inch). Ví dụ: Intel, TSMC, Samsung đã nâng kích thước wafer từ 300mm(12inch) lên 450mm(18inch).
- Chip:* IC gắn trên wafer nhằm xử lí công việc trên máy tính, kích thước rất nhỏ nhưng có chức triệu Transistor, số lượng Transistors lớn thì tốc độ truyền và xử lí tín hiệu càng nhanh.
Phân loại: 4.8,16,32,64 bit.
- Chipset:* Tập hợp nhiều chip phối hợp hoạt động. Một số chipset thông dụng: CPU, GPU, RAM, Mainboard, Northbridge: Kết nối CPU, RAM (trên mainboard) với (Hệ thống Mainboard AMD không có chipset này mà được tích hợp ngay trên CPU), Southbridge: Quản lý thiết bị ngoại vi (USB, HDD).

CHƯƠNG 2: BIỂU DIỄN SỐ NGUYÊN	
<b>2) Chuyển đổi giữa các cơ số:</b>	
- Dec → Bin: Chia số thập phân cho 2, ghi nhận số dư → Lập lại với thương số đến khi thương = 0 → Kết quả là dãy số dư viết ngược lại. Ví dụ: 123 <sub>10</sub> → 01111011;	
- Dec → Hex: Chia số thập phân cho 16, ghi nhận số dư → Lập lại với thương số cho đến khi thương = 0 → Kết quả là dãy số dư (10-15 → A-F) viết ngược lại. Ví dụ: 123 <sub>10</sub> → 7B <sub>16</sub> .	
- Bin → Hex: Nhóm từng 4 bit (từ phải sang trái) và chuyển đổi tương ứng	

Ví dụ: 01001011 → 0100 1011 → 4B

**3) Biểu diễn số nguyên không dấu:** Tất cả các bit đều biểu diễn giá trị. Phạm vi biểu diễn: 1 byte (8 bit): 0 → 255 (2<sup>8</sup>-1). Word (16 bit): 0 → 65535 (2<sup>16</sup>-1). Bit LSB = 1 → số lẻ.

**4) Biểu diễn số nguyên có dấu:**

**4.1) Dấu lượng:** MSB: 0 = dương, 1 = âm; Các bit còn lại biểu diễn độ lớn; Nhược điểm: Có 2 biểu diễn cho số 0 (+0 và -0). Phép toán phức tạp; Phạm vi: -127 đến 127 (8bit); Chức năng: dùng các thuật toán phức tạp và bất lợi vì luôn có hai cách biểu diễn của số 0

**4.2) Bù 1:** Số dương: giống dấu lượng; Số âm: đảo tất cả các bit của số dương tương ứng; Nhược điểm: Với 2 biểu diễn cho số 0, Khi cộng, nếu có bit nhớ phải cộng thêm 1 vào kết quả; Phạm vi: -127 đến 127 (8bit); Chức năng: dùng các thuật toán phức tạp và bất lợi vì luôn có hai cách biểu diễn của số 0, phép nhân của số 0 cần chăm đúng

**4.3) Bù 2:** Chuẩn máy tính hiện đại; Số dương: giống dấu lượng; Số âm: lấy bù 1 rồi cộng 1; Ưu điểm: Chỉ có 1 biểu diễn cho số 0, Phép toán đơn giản, không cần xử lý đặc biệt, Dễ dàng phát hiện tràn; Phạm vi: -128 đến 127 (8bit); Chức năng: Lưu trữ số có dấu và các phép tính của chúng trên máy tính.

Lưu ý: (Số bù 2 của x) + x = 1 dãy bit toàn 0 (không tính bit 1 cao nhất do vượt quá phạm vi lưu trữ)

**4.4) Số quá K:** (Biểu diễn N: K ≤ N < K (N là giá trị đích, với 8bit thì K=128); Chức năng: dùng cho số mũ của các số có dấu chăm đúng

Ví dụ chung: Biểu diễn -61 ở các dạng

Dấu lượng: Bit đầu: 1(âm), 7bit giá trị, |-61|=61=111101. 8bit: 00111101→Bit đầu: **00111101**

Bù 1: 61=00011101, đảo bit -61 = **11000010**

Bù 2: Lấy bù 1: 11000010, cộng thêm 1 → **11000011**

Quá K = 128: -61+ K = 67 = **01000011**

**5) Phép dịch bit và phép xoay**

- Shift left (SHL): **11001010 → 10010100**

Chuyển tất cả các bit sang trái, bỏ bit trái nhất, thêm 0 ở bit phải nhất

- Shift right (SHR): **10010101 → 01001010**

Chuyển tất cả các bit sang phải, bỏ bit phải nhất, thêm 0 ở bit trái nhất

- Shift arithmetic right (SAR): **10010101 → 11001010**

Chuyển tất cả các bit sang phải, bỏ bit phải nhất, thêm bit dấu (MSB ban đầu) ở bit trái nhất

- Rotate left (ROL): **11001010 → 10010101**

Chuyển tất cả các bit sang trái, bit trái nhất thành bit phải nhất

- Rotate right (ROR): **10010101 → 11001011**

Chuyển tất cả các bit sang phải, bit phải nhất thành bit trái nhất

6) Phép toán logic					
A	B	AND	OR	XOR	
0	0	0	0	0	
0	1	0	1	1	
1	0	0	1	1	
1	1	1	1	0	

*Mở rộng:* Lấy giá trị tại bit thứ i của x: (x SHR i) AND x

Gán giá trị 1 tại bit thứ i của x: (1 SHL i) OR x

Gán giá trị 0 tại bit thứ i của x: NOT(1 SHL i) AND x

Đảo bit thứ i của x: (1 SHL i) XOR x

**7) Các phép toán tử:**

- Phép trừ: Nguyên tắc cơ bản: Dưa về phép cộng: A - B = A + (-B) = A + (số bù 2 của B)
  - Phép nhân: Nguyên tắc cơ bản: 1\*1=1, còn lại bằng 0
- Ví dụ: 11 \* 13 = 1011 \* 1101 = 1011 + 101100 + 1011000 = 10001111 = 143
- Khởi tạo: [C, A] = 0; k = n. Lặp khi k > 0(Nếu bit cuối của Q = 1 thì Lấy (A + M) → [C, A]; Shift right ([C, A, Q]); k = k - 1}

**Thuật toán nhân nhị tiến bốn:**

Khởi tạo: A = 0, k = n, Q<sub>n+1</sub> (thêm 1 bit = 0 vào cuối Q), lặp khi k > 0 {Nếu 2 bit cuối của Q<sub>n</sub>Q<sub>n+1</sub> {= 10 thì A - M → A; = 01 thì A+M→A; = 00, 11 thì A không đổi}; SHR [A, Q, Q<sub>n+1</sub>] k = k - 1} Kết quả: [A, Q]

Ví dụ: M=7, Q=-3, n=4

	A	Q	Q <sub>n+1</sub>	M
Khởi đầu	0000	1101	0	0111
Bước 0: A=A-M	1001	1101	0	0111
SAR	1100	1110	1	0111
Bước 1: A=A+M	0011	1110	1	0111
SAR	0001	1111	0	0111
Bước 2: A=A-M	1010	1111	0	0111
SAR	1101	0111	1	0111
Bước 3: SAR	1110	1011	1	0111

Kết quả: 1110 1011 = -21

**- Phép chia:** Khởi tạo: A = n bit 0; k = n. Lặp khi k > 0 { Shift left (SHL) [A, Q], A - M → A; Nếu A < 0: Q<sub>n</sub>=0 và A + M → A; Ngược lại: Q<sub>n</sub> = 1, k=k-1} Kết quả: Q là thương, A là số dư

Ví dụ: Q=7, M=3

	A	Q	M	Ghi chú
Khởi đầu	0000	0111	0111	Thiết lập ban đầu
Bước 0	0000	1110	0111	Shift trái A,Q
A=A-M	1101	1110	0111	A = 0000 - 0011 = 1101 (âm)
Phục hồi	0000	1110	0111	A + M = 1101 + 0011 = 0000
Bước 1	0001	1100	0111	Shift trái A,Q
A=A-M	1110	1100	0111	A = 0001 - 0011 = 1110 (âm)
Phục hồi	0001	1100	0111	A + M = 1110 + 0011 = 0001
Bước 2	0011	1000	0111	Shift trái A,Q
A=A-M	0000	1000	0111	A = 0011 - 0011 = 0000 (dương)
Giữ A	0000	1001	0111	Q <sub>n</sub> =1
Bước 3	0001	0010	0111	Shift trái A,Q
A=A-M	1110	0010	0111	A = 0001 - 0011 = 1110 (âm)
Phục hồi	0001	0010	0111	A + M = 1110 + 0011 = 0001

Kết quả: Thương Q = 2, dư A = 1

8) Phán biệt chuẩn SI và IEC	
IEC	SI
Sử dụng lũy thừa của 2	Sử dụng lũy thừa của 10
Tên từ: kibi (Ki), mebi (Mi), gibi (Gi),...	Tên từ: kilo (k), mega (M), giga (G),...
1 KiB = 2 <sup>10</sup> = 1 024 byte	1 kB = 10 <sup>3</sup> =1 000 byte
1 MiB = 2 <sup>20</sup> = 1 048 576 byte	1 MB = 10 <sup>6</sup> =1 000 000 byte
Hệ thống máy tính dùng	Các nhà sản xuất ổ cứng dùng

CHƯƠNG 3: BIỂU DIỄN SỐ THỰC			
1) Cấu trúc số thực chính xác đơn (32 bit) và kép (64 bit)			
Bit dấu (Sign)	Số chính xác đơn (32 bit)	Số chính xác kép (64 bit)	
Bit mũ (Exponent)	8 (Bias 127)	11 (Bias 1023)	
Bit định trị (Significand)	23	52	

**2) Chuyển đổi số thực hệ 10 sang số thực chính xác đơn và ngược lại**

**a) Từ hệ 10 sang 32-bit:** B1: Chuyển sang nhị phân → B2: Chuẩn hóa dạng ±1.F × 2<sup>n</sup>E → B3: Biểu diễn: Sign: 0 (dương) hoặc 1 (âm), Exponent: E + 127 (biased), Significand: Phần F, bỏ 1 đầu, lấp đầy 0 hoặc làm tròn nếu vượt 23 bit.

Ví dụ: +12.625<sub>10</sub>: B1: 12.625<sub>10</sub>=1.100101...; B2: 1100.101=1.100101\*2<sup>3</sup> → B3: Sign = 0, Exponent = 3 + 127 = 130 (10000010), Significand = 100101 (táp 0)

KQ: 10000001010100000000000000000000

**b) Từ 32 bit sang hệ 10:** B1: Tách Sign, Exponent, Significand → B2: Tính E = Exponent - 127 → B3: Số ± = (1 + Significand) × 2<sup>E</sup>

Ví dụ: 1 10000001 010100000000000000000000 → B1: Sign = 1, Exponent = 10000001 = 129, Significand = 0101 và 19 bit 0 → B2: E = 129 - 127 = 2 → B3: Significand = 1.0101, Số = -1.0101 × 2<sup>-2</sup> = -1.01012 = -5.2510

**Denormalized:** Khi Exponent = 0, số được chuẩn hóa thành ±0.F × 2<sup>-126</sup>.

Ví dụ: 0 00000000 010000000000000000000000 biểu diễn ±0.012 × 2<sup>-126</sup>

3) Các số thực đặc biệt					
Loại	Sign	Exponent	Significand	Ví dụ	
Số 0	0/1	0000000	0...0	±0	
Denormalized	0/1	0000000	Khác 0	±0.F × 2 <sup>-126</sup>	
Vô cùng	0/1	1111111	0...0	±∞	
NaN	0/1	1111111	Khác 0	Cẩn(-1), 0/0	

**4) Miền biểu diễn số thực:**

- a) Normalized Numbers: Phạm vi: Số dương nhỏ nhất: +1.0 × 2<sup>-126</sup> ≈ 1.18 × 10<sup>-38</sup>
- Số dương lớn nhất: +1.111...1 × 2<sup>127</sup> ≈ 1.18 × 10<sup>38</sup>
- b) Denormalized Numbers: Phạm vi:Số dương nhỏ nhất: +0.000...1 × 2<sup>-126</sup> ≈ 1.4 × 10<sup>-45</sup>
- Số dương lớn nhất: +0.111...1 × 2<sup>-126</sup> ≈ 1.18 × 10<sup>-38</sup>

**5) Phương pháp làm tròn:** Khi bit vượt quá 23-bit (hoặc 52-bit), IEEE 754 quy định 4 chế độ làm tròn:

- Round to Nearest (Even) (Mặc định): Làm tròn đến số chẵn gần nhất. Ví dụ: 1.1011 (24 bit) → 1.110 (23 bit)
- Round Toward Zero: Cắt bỏ phần thừa. Ví dụ: 1.1011 → 1.101
- Round Up (+∞): Làm tròn lên. Ví dụ: 1.1011 → 1.110
- Round Down (-∞): Làm tròn xuống. Ví dụ: 1.1011 → 1.101

CHƯƠNG 4: KIẾN TRÚC BỘ LỆNH	
<b>1) Phán biệt khái niệm: ngôn ngữ lập trình, ngôn ngữ máy, hợp ngữ:</b>	
- Ngôn ngữ lập trình: Là ngôn ngữ nhân tạo gồm từ vựng (keyword) và ngữ pháp (syntax), giúp lập trình viên diễn đạt hướng dẫn cho máy tính. Ví dụ: C, C++, Java.	
- Ngôn ngữ máy (Machine Language): Là ngôn ngữ ở dạng nhị phân (0/1), trực tiếp được CPU hiểu và thực thi. Mỗi CPU có ngôn ngữ máy riêng.	
- Hợp ngữ (Assembly Language): Là ngôn ngữ trung gian, dùng ký hiệu mã giả thay cho mã máy, gần với ngôn ngữ máy nhưng dễ hiểu hơn cho con người. Ví dụ: ord \$4, \$0, 5	

**2) Compiler, Assembler là gì và phụ thuộc vào gì?**

Compiler	Assembler
Là trình biên dịch từ ngôn ngữ cấp cao (C, Java) sang hợp ngữ hoặc mã máy.	Là trình biên dịch từ hợp ngữ sang ngôn ngữ máy.
Phụ thuộc vào: <ul style="list-style-type: none"><li>Ngôn ngữ cấp cao được biên dịch.</li> <li>Kiến trúc phần cứng và hệ điều hành đích (ví dụ: compiler C trên Windows khác Linux).</li></ul>	Phụ thuộc vào: <ul style="list-style-type: none"><li>Bộ lệnh (ISA) của CPU (ví dụ: Assembler cho x86 khác MIPS).</li> <li>Nhà cung cấp và hệ điều hành (ví dụ: NASM, TASM).</li></ul>

**3) Cấu trúc của 1 lệnh máy:** Gồm 2 phần: Opcode (Mã lệnh): Chỉ định thao tác cần thực hiện (ví dụ: cộng, nhân). Operand (Toán hạng): Thông tin về dữ liệu bị tác động (ví dụ: địa chỉ ở nhỏ, thanh ghi).

Ví dụ: Lệnh 00110100 0000100 00000000 00000101 gồm opcode 00110100 và operand 0000100 00000000 00000101

**4) Kiến trúc bộ lệnh (ISA) và các kiến trúc thông dụng:**

ISA (Instruction Set Architecture): Là tập lệnh chuẩn cho các CPU có kiến trúc tương tự nhau. Một số ISA thông dụng: x86/x64: Của Intel (IA-16, IA-32, IA-64; MIPS: Dùng trong hệ thống nhúng; PowerPC: Của IBM

Trường phái	Đặc điểm	Đại diện
CISC (Complex Instruction Set Computer)	Bộ lệnh phức tạp, nhiều lệnh từ đơn giản đến phức tạp, tối ưu cho chương trình phức tạp.	x86 (Intel)
RISC (Reduced Instruction Set Computer)	Bộ lệnh đơn giản, tập trung vào hiệu suất cao, thực thi nhanh.	MIPS, PowerPC

**6) Quy trình tải file thực thi**

SOURCE file (ngôn ngữ cấp cao) → Compiler → Hợp ngữ

Hợp ngữ → Assembler → Object file (Mô tả thực thi (.exe, .sh)

Loader nạp thực thi vào RAM để chạy

Ví dụ: video.cpp → Compiler → video.obj → Linker → video.exe → Loader → RAM

**7) Quá trình xử lý lệnh của CPU:** Gồm 2 giai đoạn lặp lại (Instruction Cycle):

- Fetch (Nạp lệnh): PC (Program Counter) → MAR (Memory Address Register); Lấy lệnh từ bộ nhớ → MBR (Memory Buffer Register); MBR → IR (Instruction Register); PC tăng 1.
- Execute (Thực thi): Giải mã lệnh và thực hiện thao tác.

**8) Một số thanh ghi cơ bản trong CPU:**

PC (Program Counter): Lưu địa chỉ lệnh tiếp theo.

MAR (Memory Address Register): Lưu địa chỉ bộ nhớ để truy xuất.

MBR (Memory Buffer Register): Lưu giá trị dữ liệu từ/bộ nhớ.

IR (Instruction Register): Lưu mã lệnh đang xử lý.

##### CHƯƠNG 5: BỘ LỆNH LEGV8

- 1) Tập thanh ghi:**
- Thanh ghi tổng quát: LEGv8 cung cấp 32 thanh ghi tổng quát kích thước 64 bit (DoubleWord) (X0-X31):
    - X0-X7: Đối số/kết quả trả về của hàm, không bảo toàn.
    - X8: Chứa địa chỉ kết quả trả về, không bảo toàn.
    - X9-X15: Thanh ghi tạm, không bảo toàn.
    - X16-X17 (IPO-IP1): Thanh ghi tạm cho linker hoặc sử dụng tạm, không bảo toàn.
    - X18: Thanh ghi nền tảng hoặc tạm, không bảo toàn.
    - X19-X27: Thanh ghi lưu trữ, bảo toàn.
    - X28 (SP): Con trỏ ngăn xếp (stack pointer), bảo toàn.
    - X29 (FP): Con trỏ khung (frame pointer), bảo toàn.
    - X30 (LR): Thanh ghi liên kết (địa chỉ trả về), bảo toàn.
    - XZR (X31): Hằng số 0, không áp dụng vào toán.

+ Ngươi ra còn có 32 thanh ghi con kích thước 32 bit (Word) (W0-W31) → Tính toán nhanh hơn, hỗ trợ format lệnh 32 bit

- Thanh ghi có (NZCV): Negative, Zero, Carry, Overflow.

- Thanh ghi số thực: 32-bit (S0-S31), 64-bit (D0-D31).

**2) Tập lệnh:**

**a) Lệnh tính toán số học:**

- Cú pháp: op, opr1, opr2 với op (operator): Tên thao tác (toán tử), opr (operand): Thanh ghi (toán hạng) chứa số toán hạng 1, opr2 (operand 2): Thanh ghi / hằng số (toán hạng operand 2)
- Phép cộng (trừ): ADD(SUB) X1, X2, X3 // X1 = X2 +/- X3
- Phép cộng (trừ) hằng số: ADDI(SUBI) X1, X2, #20 // X1 = X2 +/- 20

→ Gán có thì thêm 'S' vào op

- Phép nhân: MUL X1,X2,X3 // X1 = X2 \* X3 (64 bit thấp của 128 bit KQ)
- SMULH X1,X2,X3 // X1 = X2 \* X3 (64 bit cao của 128 bit KQ do dấu)
- SMULL X1,X2,X3 // X1 = X2 \* X3 (64 bit thấp của 128 bit KQ có dấu)

- Phép chia: SDIV X1,X2,X3 // X1 = X2 / X3 (Phép chia 2 số có dấu)
- UDIV X1,X2,X3 // X1 = X2 / X3 (Phép chia 2 số không dấu)

**b) Lệnh logic:**

- Cú pháp: giống tập lệnh số học
- AND, ORR, EOR (OR): Toán hạng nguồn thứ 2 (opr2) phải là thanh ghi
- ANDI, ORRI, EORI: Toán hạng nguồn thứ 2 (opr2) là hằng số
- LEGv8 không hỗ trợ trực tiếp phép luận lý **NOT**. Để thực hiện phép not ta có thể thực hiện xor thanh ghi đó với giá trị toán bit 1

- Dịch luận lý: Dịch trái (LSL –left shift logical): Thêm vào các bit 0 bên phải
- Dịch phải (RSL –right shift logical): Thêm vào các bit 0 bên trái

Ví dụ: LSL X1,X1,#10 // Ý nghĩa: X1 = X1 << 10

- Dịch số học: Không có dịch trái số học

Dịch phải (RSA –right shift arithmetic): Thêm các bit = bit dấu bên trái

**3) Lệnh rẽ nhánh:**

- Rẽ nhánh có điều kiện: CBZ X1, label // if (X1 == 0) goto label
- CBNZ X1, label // if (X1 != 0) goto label

- + So sánh 2 giá trị với nhau: CMP opr1, opr2 // opr1: *thanh ghi*, opr2: *thanh ghi / hằng số*
- + Kiểm tra kết quả so sánh bằng lệnh nhảy có điều kiện: B.Cond label // *If(Cond) goto label*

Cond	EQ	NE	GT(HI)	LT(LO)	GE(HS)	LE(LS)
Ý nghĩa	X1==X2	X1!=X2	X1>X2	X1<X2	X1>=X2	X1<=X2

*Lưu ý:* trong ngoặc là danh cho unsigned

- Rẽ nhánh không điều kiện: B label // goto label (Branch)
- BR X1 // goto address in X1 (Branch to register)
- BL label // X30 = PC + 4, goto label (Branch with link)

Ví dụ: Biên dịch câu lệnh trong C thành lệnh hợp ngữ LEGV8: if (i == j) f = g + h; else f = g - h.
Ảnh xạ biến f, g, h, i, j tương ứng vào các thanh ghi: X0, X1, X2, X3, X4

B.EQ TrueCase // if(i == j) goto TrueCase

SUB X0,X1,X2 // f = g - h (false)

B.FIN // goto "Fin" label

TrueCase: ADD X0,X1,X2 // f = g + h (true)

Fin: ...

**4) Lệnh di chuyển dữ liệu (Load/Store):**

- Cú pháp: op, opr, [opr1, opr2] với op (operator): Tên thao tác (Load/Store); opr (operand): Thanh ghi lưu doubleword; opr1 (operand 1): Thanh ghi chứa địa chỉ vùng nhớ cơ sở (địa chỉ nền); opr2 (operand 2): Hằng số nguyên
- 2 thao tác chính:

+ Load register: Nạp 1 doubleword dữ liệu, từ bộ nhớ vào 1 thanh ghi trên CPU

LDUR X1, [X2, #40]; Gán X1 bằng giá trị tại ô nhớ có địa chỉ (X2 + 40)

Store register: Lưu 1 doubleword dữ liệu, từ thanh ghi trên CPU, ra bộ nhớ

STUR X1, [X2, #40]; Lưu giá trị trong thanh ghi X1 vào ô nhớ có địa chỉ (X2 + 40)

- Hỗ trợ load, store 1 Word: Load word: LDURSW; Store word: STURH; Cú pháp lệnh tương tự LDUR, STUR

Ví dụ: LDURSW X1, [X2, #40] (X1 = Memory[X2 + 40], 4-byte, có dấu)

STURW X1, [X2, #40] (Memory[X2 + 40] = X1, 4-byte)

- Hỗ trợ load, store 1/2 Word: Load half: LDURH; Store half: STURH; Cú pháp lệnh tương tự LDUR, STUR

Ví dụ: LDURH X1, [X2, #40] (X1 = Memory[X2 + 40], 2-byte)

STURH X1, [X2, #40] (Memory[X2 + 40] = X1, 2-byte)

- Hỗ trợ load, store đọc quyền trong trường có nhiều lưu trữ, hoặc tiến trình cùng truy cập vùng nhớ: Load exclusive register: LDXR; Store exclusive register: STXR; Cú pháp lệnh tương tự LDUR, STUR

Ví dụ: LDXR X1, [X2, #40] (Load độc quyền, 8-byte)

STXR X1, X3, [X2] (Store độc quyền, X3 = 0/1)

**5) Triển khai vòng lặp, if-else, switch-case**

– Thành phần: **PC**: Lưu địa chỉ, tăng PC + 4; **Instruction Memory**: Đọc lệnh 32-bit; **Register File**: 2 đọc (Rn, Rm), 1 ghi (Rd/Rt), tín hiệu RegWrite; **ALU**: Thực hiện phép toán, tín hiệu điều khiển 4-bit (ADD: 0010, SUB: 0110); **Data Memory**: Đọc (MemRead) / ghi (MemWrite); **Sign-Extend**: Mở rộng dấu cho địa chỉ/hàng.

– Công đoạn thực thi (5 giai đoạn): **Fetch**: Đọc lệnh từ bộ nhớ, tăng PC; **Decode**: Phân tích opcode, đọc thanh ghi, mở rộng dấu; **Execute**: Tính toán (ALU) hoặc địa chỉ bộ nhớ/nhảy; **Memory**: Đọc/ghi dữ liệu (LDUR, STUR); **Write Back**: Ghi kết quả vào thanh ghi (R-Format, LDUR).

– Lệnh CBZ, B tính vị trí cần nhảy tới: PC = PC + (address\*4).

– Lệnh LDUR và STUR tính địa chỉ bộ nhớ cần truy xuất Rn + address

**b) Control Unit**: Phát tín hiệu: RegWrite, MemRead, MemWrite, ALUSrc, RegDst, PCSrc, ALUOp. Ví dụ: R-Format: RegWrite=1, ALUOp=10; LDUR: MemRead=1, MemtoReg=1, ALUSrc=1; STUR: MemWrite=1, ALUSrc=1; CBZ: Branch=1, ALUOp=01; B: Uncond=1.

Tín hiệu	Vai trò chính	Lệnh ảnh hưởng
Reg2Loc	Chọn thanh ghi thứ 2	R-type, D-type
ALUSrc	Chọn nguồn toán hạng 2 cho ALU	I-type, D-type
ALUOp	Quy định chức năng của ALU Control	Tất cả
MemRead	Cho phép đọc từ bộ nhớ	LDUR
MemWrite	Cho phép ghi vào bộ nhớ	STUR
MemtoReg	Chọn nguồn ghi vào thanh ghi	LDUR, ADD...
RegWrite	Ghi kết quả vào thanh ghi	LDUR, ADD...
FlagWrite	Ghi có N,Z,C,V từ ALU	CMP, SUB...
UncondBranch	Rẽ nhánh không điều kiện	B
ZeroBranch	Rẽ nhánh khi Z=1	CBZ
FlagBranch	Rẽ nhánh theo có N/Z/C/V	B.cond

**c) Thiết kế CPU: Một chu kỳ**: Xử lý toàn bộ lệnh trong 1 chu kỳ (1000ps cho LDUR), không hiệu quả; **Đa chu kỳ**: Mỗi công đoạn 200ps, hiệu năng ~824ps × IC, dùng chung ALU, bộ nhớ; **Pipelining**: Song song hóa, giảm thời gian (200ps/giai đoạn), xử lý xung đột bằng forwarding, stalling, branch prediction.

SO SÁNH LEGV8 & X86		
	X86	LEGv8
Đặc điểm	- Là một kiến trúc CISC (Complex Instruction Set Computer). - Hỗ trợ một tập lệnh phức tạp và đa dạng. - Có nhiều biến thể và chế độ hoạt động khác nhau.	- Là một kiến trúc RISC (Reduced Instruction Set Computer). - Thiết kế đơn giản với một số lệnh cơ bản và ít biến thể. - Hỗ trợ các chế độ xử lý khác nhau như user mode, supervisor mode, và hypervisor mode.
Ưu	- Hỗ trợ các tính năng phong phú như SIMD (Single Instruction, Multiple Data), các lệnh phức tạp. - Được sử dụng rộng rãi trong nhiều ứng dụng máy tính và server.	- Hiệu suất cao với các lệnh đơn giản, dễ dàng thực hiện trên phần cứng. - Tiết kiệm năng lượng và không gian bộ nhớ.
Nhược	- Phức tạp trong việc thiết kế và triển khai trên phần cứng. Tốn nhiều nguồn tài nguyên hơn so với các kiến trúc RISC.	- Yêu cầu nhiều lệnh hơn để thực hiện các tác vụ phức tạp so với CISC. Cần phải có sự hỗ trợ phần cứng tốt để tối ưu hiệu suất.

## CHƯƠNG 6: BỘ LỆNH X86-32BIT

### 1) Tập thanh ghi

#### a) Thanh ghi đa năng (General Purpose Registers):

Thanh ghi 32-bit	16-bit	8-bit cao	8-bit thấp	Vai trò chính
EAX	AX	AH	AL	<b>Accumulator</b> : Tích lũy kết quả toán học, nhân/chia
EBX	BX	BH	BL	<b>Base</b> : Địa chỉ cơ sở trong truy cập bộ nhớ
ECX	CX	CH	CL	<b>Counter</b> : Đếm số vòng lặp, CL dùng cho dịch bit
EDX	DX	DH	DL	<b>Data</b> : Dùng trong toán tử mở rộng, chia, I/O

#### b) Thanh ghi đoạn (Segment Registers) (16 bit):

Thanh ghi	Chức năng chính
CS (Code Segment)	Chứa địa chỉ cơ sở của đoạn mã – nơi lưu trữ lệnh chương trình. CPU luôn thực thi lệnh tại địa chỉ CS:IP.
DS (Data Segment)	Chứa địa chỉ cơ sở của đoạn dữ liệu – nơi lưu các biến, mảng, cấu trúc...
SS (Stack Segment)	Chứa địa chỉ cơ sở của ngăn xếp – dùng cho các lệnh PUSH, POP, CALL, RET...
ES (Extra Segment)	<b>Đoạn dữ liệu bổ sung</b> , thường dùng trong các thao tác với lệnh MOVSB, STOSB...

**c) Thanh ghi con trỏ & chỉ số**: IP, SP, BP, SI, DI (16-bit, mở rộng thành EIP, ESP, EBP, ESI, EDI từ 80386).

IP: Instruction Pointer - Trỏ đến lệnh tiếp theo trong CS (CS:IP).

SP: Stack Pointer - Trỏ đến đỉnh ngăn xếp (SS:SP).

BP: Base Pointer - Trỏ đến dữ liệu trong stack (SS:BP).

SI: Source Index - Trỏ đến dữ liệu nguồn (DS:SI).

DI: Destination Index - Trỏ đến dữ liệu đích (ES:DI).

→ SI và DI có thể được sử dụng như thanh ghi đa năng.

Tên cờ	Bit#	Logi có	Khi = 1 thì...	Giải thích
CF (Carry Flag)	0	State	Có nhớ hoặc mượn xảy ra (không dấu)	Trần trong phép cộng/trừ không dấu
PF (Parity Flag)	2	State	Số lượng bit 1 trong byte thấp là chẵn	Kiểm tra lỗi truyền đơn giản
AF (Auxiliary Carry Flag)	4	State	Có nhớ giữa bit 3 và 4 (BCD)	Dùng trong số thập phân nhị phân (BCD)
ZF (Zero Flag)	6	State	Kết quả phép toán bằng 0	Dùng để kiểm tra so sánh, nhảy

SF (Sign Flag)	7	State	Kết quả là số âm (bit dấu = 1)	Phản ánh dấu của kết quả của câu
OF (Overflow Flag)	11	State	Tràn số có dấu xảy ra	Kết quả vượt quá phạm vi biểu diễn với số có dấu
TF (Trap Flag)	8	Control	CPU chạy từng lệnh một (single-step mode)	Dùng cho debug
IF (Interrupt Enable Flag)	9	Control	Cho phép CPU nhận ngắt (maskable interrupt)	Điều khiển ngắt ngoài; dùng STI, CLI
DF (Direction Flag)	10	Control	Xử lý chuỗi từ phải sang trái	Dùng với lệnh chuỗi (MOVS, LODS, STOS,...)

### 2) Tập lệnh:

#### a) Lệnh tính toán số học + Logic

Lệnh	Chức năng	Thực hiện	Cờ ảnh hưởng	Ví dụ
ADD D, N	Cộng	D ← D + N	CF, OF, ZF, SF, PF, AF	ADD AL, 10H
ADC D, N	Cộng có nhớ	D ← D + N + CF	Như ADD	ADC AL, 10H
SUB D, N	Trừ	D ← D - N	Như ADD	SUB AL, 30H
SBB D, N	Trừ có nhớ	D ← D - N - CF	Như ADD	SBB AL, 10H
INC D	Cộng 1	D ← D + 1	Không CF	INC AX
DEC D	Trừ 1	D ← D - 1	Không CF	DEC BX
NEG D	Đổi dấu (bù 2)	D ← -D	Có CF	NEG AL
MUL N	Nhân không dấu	AL*N → AX	CF, OF	MUL BL
IMUL N	Nhân có dấu	Có dấu như MUL	CF, OF	IMUL BL
DIV N	Chia không dấu	AL=AX/N, AH=dư	Không thay đổi	DIV BL
IDIV N	Chia có dấu	Tương tự DIV	–	IDIV BL

Lệnh	Chức năng	Kết quả	Cờ ảnh hưởng	Ví dụ
AND D, N	AND bit	D ← D & N	PF, SF, ZF; Xoá CF, OF	AND AL, 0Fh
OR D, N	OR bit	D ← D   N	PF, SF, ZF	OR AL, 0Fh
XOR D, N	XOR bit	D ← D ^ N	PF, SF, ZF	XOR AL, 0Fh
NOT D	Đảo bit	D ← ~D	Không đổi cờ	NOT AL
TEST D, N	AND kiểm tra	Không lưu kết quả	PF, SF, ZF	TEST AL, 01H

Lệnh	Chức năng	Thực hiện	Cờ	Ví dụ
RCL D, N	Quay trái qua CF	Dịch qua CF	CF, OF, RCL AL, 1	
RCR D, N	Quay phải qua CF	–	CF, OF, RCR AL, 1	
ROL D, N	Quay trái	Không qua CF	CF, OF, ROL AL, 1	
ROR D, N	Quay phải	–	CF, OF, ROR AL, 1	

Lệnh	Chức năng	Thực hiện	Cờ	Ví dụ
SAL D, N / SHL	Dịch trái số học	D ← D * 2 <sup>n</sup>	ZF, SF, PF, CF	SAR AL, 1
SAR D, N	Dịch phải số học	Giữ bit dấu	SF, ZF, PF, CF (MSB)	SAR AL, 1
SHR D, N	Dịch phải logic	Chèn 0 bên trái	SF, ZF, PF, CF (LSB)	SHR AL, 1

### b) Lệnh rẽ nhánh:

– Không điều kiện: JMP; Nhảy đến địa chỉ xác định; Short: ±128 byte; Near: ±32 KB; Far: Nhảy sang đoạn khác (CS:IP).

– Có điều kiện: JE/JZ; Nhảy nếu bằng (ZF=1); JNE/JNZ; Nhảy nếu không bằng (ZF=0); JG/JGE; Nhảy nếu lớn/lớn hơn hoặc bằng (dùng cờ số có dấu); JL/JLE; Nhảy nếu nhỏ/nhỏ hơn hoặc bằng (dùng cờ số có dấu); JA/JAE, JB/JBE; Tương tự cho số không dấu.

### c) Lệnh di chuyển dữ liệu (Load/Store):

- MOV: Chuyển dữ liệu (Đích = Nguồn); Nguồn: Thanh ghi, 0, nhớ, hằng số; Đích: Thanh ghi, 0, nhớ (không cho phép cả hai là 0 nhớ).
- XCHG: Hoán đổi nội dung (Đích = Nguồn).
- MOVSB/MOVS/MOVSQ: Chuyển chuỗi (byte/từ/từ DS:SI sang ES:DI).
- IN: Đọc từ cổng (AL/AX = Port).
- OUT: Ghi ra cổng (Port = AL/AX).
- LEA: Lấy địa chỉ hiệu quả (Đích = địa chỉ Nguồn).

### 3) Triển khai vòng lặp, if-else, switch-case:

a) Vòng lặp:	FOR-DO: MOV CX, 80 ; Số lần lặp HEN: INT 21H ; Công việc LOOP HEN ; Giảm CX, nhảy nếu CX ≠ 0 TIEP: CMP AL, 13 ; Điều kiện JE End ; Thoát nếu đúng INT 21H JMP TIEP End	WHILE-DO: CMP AL, 13 ; Điều kiện JNE TIEP ; Lặp nếu sai INT 21H JMP TIEP End	REPEAT-UNTIL-TIEP: INT 21H ; Công việc CMP AL, 13 ; Điều kiện JNE TIEP ; Lặp nếu sai INT 21H JMP TIEP End
b) IF-THEN:	CMP AX, 0 ; Điều kiện JNL End if ; Thoát nếu sai NEG AX ; Công việc End if MOV BX, AX	CMP AX, BX ; Điều kiện JL Then ; Nếu đúng	
c) IF-Then-Else:			

MOV CX, 1 ; Else JMP End if Then: MOV CX, 0 End if	Switch-Case: CMP AX, 0 ; Kiểm tra điều kiện JL AM ; AX < 0 JE Không ; AX = 0 JG ĐUONG ; AX > 0 AM: MOV CX, -1 JMP End case Không: MOV CX, 0 JMP End case DUONG: MOV CX, 1 End case
---	---

### 4) Thao tác với stack:

- PUSH: Đẩy từ (16-bit) vào ngăn xếp (SP = SP - 2, Nguồn → [SS:SP]). Ví dụ: PUSH BX, PUSH [BX].
- PUSHA: Đẩy tất cả thanh ghi đa năng vào ngăn xếp.
- POPHF: Đẩy thanh ghi có vào ngăn xếp.
- POP: Lấy từ từ ngăn xếp (Đích ← [SS:SP], SP = SP + 2). Ví dụ: POP BX, POP [BX].
- POPA: Lấy tất cả thanh ghi đa năng từ ngăn xếp.
- POPF: Lấy thanh ghi có từ ngăn xếp.

### 5) Thao tác với mảng 1 chiều:

- Khai báo: M1 DB 4, 5, 6, 7, 8, 9 ; Mảng byte  
M2 DW 100 DUP(0) ; Mảng từ, khởi tạo 0
- Truy cập: MOV AL, M1 ; Lấy phần tử đầu  
MOV AL, M1[2] ; Lấy phần tử thứ 3  
MOV BX, 1  
MOV AL, M1[BX] ; Truy cập qua chỉ số BX  
MOVS byte1, byte2 ; Chuyển byte từ DS:SI sang ES:DI

### - Xử lý chuỗi:

- Thủ tục/Hàm: Tong PROC NEAR ; Thủ tục gần  
ADD AX, BX  
RET  
Tong ENDP

- Thanh ghi liên quan: Tham số: Thường qua thanh ghi (AX, BX, CX, DX) hoặc ngăn xếp (SS:BP); Kết quả trả về: Thường trong AX hoặc DX:AX (cho kết quả lớn).

• Gọi hàm: CALL: Đẩy IP (gần) hoặc CS:IP (xa) vào ngăn xếp, nhảy đến thủ tục.

CALL Tong ; Gọi thủ tục gần  
CALL FAR PTR Tong ; Gọi thủ tục xa

+ Return: RET: Lấy IP (gần) hoặc CS:IP (xa) từ ngăn xếp, quay về chương trình gọi.

- Ngắt: INT: Gọi chương trình con thuộc IP (địa chỉ, CS, IP vào ngăn xếp); IRET: Quay về từ ngắt (lấy cờ, CS, IP từ ngăn xếp).

## CHƯƠNG 7: MẠCH LOGIC

**1) Mạch tổ hợp (Combinational Circuit)**: Mạch tổ hợp là mạch logic có n ngõ vào, m ngõ ra mà đầu ra chỉ phụ thuộc vào giá trị hiện tại của đầu vào, không phụ thuộc vào trạng thái quá khứ. Đặc điểm: Không có khả năng lưu trữ (không có thành phần nhớ); Cấu tạo từ các cổng logic cơ bản (AND, OR, NOT, NAND, NOR, XOR); Ví dụ: Mạch cộng, mạch giải mã, mạch mã hóa.

**2) Các bước thiết kế mạch tổ hợp**: B1: Lập bảng chân trị: Xác định giá trị ngõ ra (F) cho các tổ hợp ngõ vào (A, B, C,...) → B2: Viết hàm luận lý: Dựa trên bảng chân trị, biểu diễn hàm dưới dạng SOP hoặc POS → B3: Vẽ sơ đồ mạch và thi nghiệm: Sử dụng công luận lý (AND, OR, NOT,...) để xây dựng mạch và kiểm tra.

### 3) Đơn giản hóa hàm logic bằng bản đồ Karnaugh:

- Mục đích: Giảm số lượng cổng logic, tối ưu mạch.

- Cách thực hiện: B1: Biểu diễn hàm logic lên bản đồ Karnaugh (dựa trên bảng chân trị) → B2: Gom các ô chứa giá trị 1 (với SOP) hoặc 0 (với POS) thành nhóm lớn nhất có thể (2, 4, 8) → B3: Mỗi nhóm tương ứng với một số hạng đơn giản → B4: Viết lại hàm logic từ các nhóm đã gom.

Lưu ý: Các ô liên kế chỉ khác nhau 1 biến. Nhóm 2/4/8 ô loại bỏ 1/2/3 biến.

- Các biểu đồ Karnaugh cơ bản:

<div><div><div><div>0</div><div>1</div></div><div><div>2</div><div>3</div></div></div><div><div>A</div><div>B</div></div></div>	<div><div><div><div>0</div><div>1</div><div>3</div><div>2</div></div><div><div>4</div><div>5</div><div>7</div><div>6</div></div></div><div><div>A</div><div>B</div></div></div>
<div><div><div><div>0</div><div>1</div><div>3</div><div>2</div></div><div><div>4</div><div>5</div><div>7</div><div>6</div></div></div><div><div>A</div><div>B</div></div></div>	<div><div><div><div>0</div><div>1</div><div>3</div><div>2</div></div><div><div>4</div><div>5</div><div>7</div><div>6</div></div></div><div><div>A</div><div>B</div></div></div>
<div><div><div><div>0</div><div>1</div><div>3</div><div>2</div></div><div><div>4</div><div>5</div><div>7</div><div>6</div></div></div><div><div>A</div><div>B</div></div></div>	<div><div><div><div>0</div><div>1</div><div>3</div><div>2</div></div><div><div>4</div><div>5</div><div>7</div><div>6</div></div></div><div><div>A</div><div>B</div></div></div>

Ví dụ: Hàm F(A, B, C) = ∑(1,4,5,6,7) → Gom nhóm trên biểu đồ Karnaugh → F = A + B'

### 4) SOP & POS

SOP (Sum of Products)	POS (Product of Sums)
Biểu diễn hàm logic dưới dạng tổng các tích.	Biểu diễn hàm logic dưới dạng tích các tổng.
Ưu tiên dùng khi: Số lượng ô 1 trong bảng chân trị ít hơn 0.	Ưu tiên dùng khi: Số lượng ô 0 ít hơn 0.
Ví dụ: F = AB + A'C	Ví dụ: F = (A + B)(A' + C)

Kết hợp với Karnaugh: SOP: Gom nhóm các ô 1.  
POS: Gom nhóm các ô 0, sau đó lấy bù của kết quả.

**5) Điều kiện không cần thiết (kết hợp)**: Một số tổ hợp đầu vào không bao giờ xảy ra hoặc đầu ra không quan trọng (khi X trong bảng chân trị). Các điều kiện: Cõi X là 1 hoặc 0 để tạo nhóm lớn nhất trên bản đồ Karnaugh; Dùng được gom nhóm chỉ toán X.

Ví dụ: F(A,B,C) = ∑(0,2,6) + ∑(1,3,5) → Không X để gom nhóm A' + B'

### 6) Một số mạch tổ hợp cơ bản:

**a) Mạch toàn cộng (Full Adder - FA)**: Thực hiện phép cộng 3 bit (A, B, Cin) → 2 đầu ra (Sum, Cout). Ứng dụng: Mạch cộng nhị phân nhiều bit.

Sum = A ⊕ B ⊕ Cin; Cout = (A & B) ∨ (B & Cin) ∨ (A & Cin)

**b) Mạch mã hóa (Encoder)**: Chuyển đổi 1 đầu vào thành mã nhị phân n bit. Ví dụ: Mã hóa 4-2 (4 đầu vào → 2 bit đầu ra).

**c) Mạch giải mã (Decoder)**: Giải mã n bit đầu vào thành 1 trong 2<sup>n</sup> đầu ra. Ví dụ: Giải mã 2-4 (2 bit đầu vào → 4 đầu ra).

**d) Mạch dồn (Multiplexer - MUX)**: Mạch chọn dữ liệu. Chọn n ngõ trong 2<sup>n</sup> ngõ vào để quyết định giá trị của duy nhất 1 ngõ ra. Ví dụ: MUX 4-1 (4 đầu vào, 2 bit chọn, 1 đầu ra).

**e) Mạch tách (Demultiplexer - DEMUX)**: Phân phối 1 đầu vào đến 1 trong nhiều đầu ra.

**7) Mạch tuần tự (Sequential Circuit)**: Mạch có đầu ra phụ thuộc vào đầu vào hiện tại và trạng thái quá khứ (có khả năng lưu trữ).

Mạch lật: lưu trữ 1 bit, nhiều loại khác ở số ngõ vào + cách thức tác động.

- **Latch**: Lưu trữ 1 bit, ngõ ra thay đổi khi ngõ vào thay đổi. Ví dụ: RS Latch: S=1 (set Q=1), R=1 (reset Q=0), S=R=0 (giữ), S=R=1 (không xác định). Ứng dụng: Bộ nhớ trong mạch **bất đồng bộ**.

- **Flip-Flop**: Lưu trữ 1 bit, chỉ thay đổi trạng thái tại cạnh xung đồng hồ (clock edge). **RS Flip-Flop**: Thêm clock, chỉ hoạt động khi clock = 1. **D Flip-Flop**: 1 ngõ vào D, trạng thái thay đổi không xác định, Q(t+1) = D. **JK Flip-Flop**: Cài thiết RS, J=K=1 chuyển Q sang trạng thái bù. **T Flip-Flop**: T=1 (Q bù), T=0 (Q giữ). Ứng dụng: Bộ nhớ trong mạch **đồng bộ**.

- **Thanh ghi dịch (Shift Register)**: Dịch chuyển dữ liệu qua các Flip-Flop theo xung đồng hồ. Ví dụ: Thanh ghi dịch 4 bit, dịch 0100 → 0010. Ứng dụng: Lưu trữ và truyền dữ liệu nối tiếp.

## CHƯƠNG 8: BỘ NHỚ

Bộ nhớ trong (Primary Memory)	Bộ nhớ ngoài (Secondary Memory)				
- Thanh ghi (Registers): Nhanh nhất, nằm trong CPU, dung lượng nhỏ (KB). - Bộ nhớ Cache (L1, L2, L3): Tốc độ cao, dùng SRAM, dung lượng từ KB đến MB. - Bộ nhớ chính (RAM - DRAM): Tốc độ trung bình, dung lượng lớn (GB), dùng cho chương trình đang chạy.	- Địa chỉ (HDD): Tốc độ chậm, dung lượng lớn (TB), giá rẻ. - Flash (SSD, USB, thẻ nhớ): Tốc độ cao hơn HDD, dung lượng từ GB đến TB. - HDD, quang (CD/DVD/Blu-ray): Dung lượng trung bình (GB), dùng lưu trữ dài hạn.				
<b>1) Các dạng bộ nhớ bên trong máy tính:</b>					
<b>2) Một số thiết bị bộ nhớ hiện nay:</b> Sắp xếp (Dung lượng tăng dần, tốc độ giảm dần, giá thành tăng 1 bit giảm dần): Thanh ghi – L1 – L2 – L3 – Bộ nhớ chính – Bộ nhớ ngoài – Bộ nhớ mang					
<b>3) Phân loại SRAM &amp; DRAM</b>					
<table><tr><th>SRAM (Static RAM)</th><th>DRAM (Dynamic RAM)</th></tr><tr><td>Các bit được lưu trữ bằng các Flip-Flop → Thông tin ổn định; Cấu trúc phức tạp; Dung lượng thấp; Tốc độ nhanh; Đất ít.</td><td>Các bit được lưu trữ trên tụ điện → Cần phải có mạch refresh; Cấu trúc đơn giản; Dung lượng lớn; Tốc độ chậm hơn; Rẻ tiền hơn; <b>Dùng làm bộ nhớ chính</b></td></tr></table>	SRAM (Static RAM)	DRAM (Dynamic RAM)	Các bit được lưu trữ bằng các Flip-Flop → Thông tin ổn định; Cấu trúc phức tạp; Dung lượng thấp; Tốc độ nhanh; Đất ít.	Các bit được lưu trữ trên tụ điện → Cần phải có mạch refresh; Cấu trúc đơn giản; Dung lượng lớn; Tốc độ chậm hơn; Rẻ tiền hơn; <b>Dùng làm bộ nhớ chính</b>	
SRAM (Static RAM)	DRAM (Dynamic RAM)				
Các bit được lưu trữ bằng các Flip-Flop → Thông tin ổn định; Cấu trúc phức tạp; Dung lượng thấp; Tốc độ nhanh; Đất ít.	Các bit được lưu trữ trên tụ điện → Cần phải có mạch refresh; Cấu trúc đơn giản; Dung lượng lớn; Tốc độ chậm hơn; Rẻ tiền hơn; <b>Dùng làm bộ nhớ chính</b>				

### 4) Hai nguyên lý làm cơ sở truy xuất:

- Cục bộ thời gian (Temporal Locality): Nếu một ô nhớ được truy xuất hiện tại, nó có khả năng được truy xuất lại trong tương lai gần. Ví dụ: Vòng lặp, biến đếm.
- Cục bộ không gian (Spatial Locality): Nếu một ô nhớ được truy xuất, các ô ở lân cận có khả năng được truy xuất tiếp theo. Ví dụ: Truy cập mảng tuần tự.

### 5) Các phương pháp ánh xạ dữ liệu vào cache:

	Direct Mapping	Associative Mapping	Set-Associative Mapping
Cách hoạt động	Mỗi block bộ nhớ chính được ánh xạ có định vào 1 line cache.	Block có thể lưu ở bất kỳ line nào trong cache.	Cache chia thành các set, mỗi set chứa nhiều line. Block được ánh xạ vào 1 set cụ thể nhưng có thể lưu ở bất kỳ line nào trong set đó.
Các trường	<b>Tag</b> : Xác định block nào đang lưu trong line. <b>Line</b> : Chỉ số line trong cache. <b>Word</b> : Vị trí từ trong block.	<b>Tag</b> : Xác định block. <b>Word</b> : Vị trí từ trong block.	<b>Tag</b> : Xác định block trong set. <b>Set</b> : Chỉ số set trong cache. <b>Word</b> : Vị trí từ trong block.
Ưu	Đơn giản, tốc độ tra cứu nhanh.	Giảm xung đột, tăng cache hit rate.	Cân bằng giữa tốc độ và xung đột.
Nhược	Xung đột cao nếu nhiều block cùng ánh xạ vào 1 line, cache hit thấp.	Phức tạp, tốn tài nguyên, time so sánh tag.	Phức tạp hơn Direct Mapping.
Tính	Bộ nhớ chính: 2 <sup>N</sup> Line: 2 <sup>W</sup> Số cache line: cache / line = 2 <sup>L</sup> Số cache set = Số cache line / số set line (số way) = 2 <sup>S</sup> T = N - (W + L) Tổng số Block: M = main size / block size	Bộ nhớ chính: 2 <sup>N</sup> Line: 2 <sup>W</sup> Số cache line: cache / line = 2 <sup>L</sup> Số cache set = Số cache line / số set line (số way) = 2 <sup>S</sup> T = N - (S+W)	Bộ nhớ chính: 2 <sup>N</sup> Line: 2 <sup>W</sup> Số cache line: cache / line = 2 <sup>L</sup> Số cache set = Số cache line / số set line (số way) = 2 <sup>S</sup> T = N - (S+W)

### 6) Các phương pháp thay thế cache:

Phương pháp	Mô tả	Ưu/Nhược
Random	Chọn ngẫu nhiên 1 line để thay thế.	Đơn giản, nhưng hiệu suất thấp do không ổn định.
FIFO	Thay thế line được nạp vào sớm nhất.	Dễ triển khai, nhưng không tối ưu.
LFU	Thay thế line ít được truy cập nhất.	Tồn chi phí theo dõi tần suất.
LRU	Thay thế line lâu nhất không được dùng.	<b>Tối ưu nhất</b> , phổ biến trong thực tế.

### 7) Các yếu tố ảnh hưởng hiệu suất cache:

- **Block Size**: Nhỏ: Giảm spatial locality, tăng số lần truy cập RAM. Lớn: Tăng thời gian truyền dữ liệu (memory penalty); thường từ 8 tới 64 byte.
- **Cache Size**: Nhỏ: Tăng tỷ lệ miss. Lớn: Tăng độ trễ truy cập và chi phí; tỉ lệ thuận với giá thành.

### 8) Các phương pháp đồng bộ dữ liệu cache và RAM:

**Write Through**: Khi ngay dữ liệu thay đổi từ cache lên RAM; **Write Back**: Chỉ ghi lên RAM khi line trong cache bị thay thế; **Bus Watching with Write Through**: Loại bỏ line trong cache khác khi 1 cache thay đổi dữ liệu; **Hardware Transparency**: Tự động cập nhật các cache khác khi một cache thay đổi; **Noncacheable Shared Memory**: Phần bộ nhớ chung không được đưa vào cache.

### 9) Cấu tạo đĩa cứng và cách tính dung lượng:

Cấu tạo đĩa cứng (HDD): **Platter** (Đĩa từ): Gồm nhiều lớp đĩa phủ vật liệu từ tính; **Track**: Vòng tròn đồng tâm trên platter; **Sector**: Phần nhỏ trên track (thường 512 byte hoặc 4KB); **Cylinder**: Tập các track cùng bán kính trên nhiều platter. **Công thức tính dung lượng**: Dung lượng = Số platter × Số mặt sử dụng mỗi platter × Số track mỗi mặt × Số sector mỗi track × Kích thước sector (thường 512 byte). Ví dụ (HDD): 10k00 ổ cứng có 2 platter, 2 mặt sử dụng trên mỗi platter, 10,000 track/mặt, 500 sector/track, 512 byte/sector → Dung lượng = 2x2x10000x500x512=... (byte).

- Disk Latency = Seek Time + Rotation Time + Transfer Time