**Các thế hệ máy tính:** **Thế hệ 0:** Non-digital computers; **thế hệ 1 (1940-1956)**: Vacuum-tube: đèn chân không, hệ thống ENIAC (electronic numerical intergrator and computer); **thế hệ 2 (1956-1963):** Transistor (linh kiện bán dẫn): IBM 7094; **thế hệ 3 (1964 – 1971):** Integrated curcuit (vi mạch tích hợp): IBM 360; **thế hệ 4 (1971-nay):** microprocessor (vi xử lý): Intel 4040, XT computer, Intel 80486DX2; **thế hệ 5 (tương lai):** parallel processing. **Định luật Moore:** Số lượng transistor tích hợp trong 1 IC tăng gấp đôi mỗi 1.5 năm (2 năm or 18 tháng) **Thành phần cơ bản:** 5 thành phần: Input, output, memory, processor, datapath, control. (bộ xử lý, bộ nhớ chính, hệ thống kết nối, thiết bị nhập xuất) **Chipset:** Chipset là tập hợp nhiều chip gắn kết lại với nhau trên cùng 1 đế chip (wafer) để xử lý nhiều công việc trên máy tính. Một số chipset thông dụng: CPU: Đơn vị xử lý trung tâm; GPU: Đơn vị xử lý đồ họa trên máy; RAM: Bộ nhớ truy cập tức thời chuyên phục vụ cho CPU; Bán cầu bắc (tích hợp trên mainboard): Hỗ trợ truyền thông tin cho CPU, RAM, nằm sát CPU (Hệ thống Mainboard AMD không có chipset này vì được tích hợp ngay trên CPU); Bán cầu nam (tích hợp trên mainboard): Quản lý thiết bị ngoại vị như HDD, Mouse, Keyboard…Nằm cuối mainboard.

**Wafer:** Đế chip. Tấm silicon mỏng đã được cấy vật liệu khác nhau để tạo ra những vi mạch, có kích thước trung bình từ 25.4 mm (1 inch) - 200 mm (7.9 inch). **Chip:** có kích thước rất nhỏ nhưng chứa hàng chục triệu transistor, số lượng transistor càng lớn thì tốc độ truyền và xử lý tín hiệu càng nhanh, các loại chip hiện nay: 4, 8, 16, 32, 64 bit.

**Chuẩn IEC (Prefix in byte)**: Theo thứ tự Name, Abbr, Factor: [kibi, Ki, 2^10], [mebi, Mi, 2^20], [gibi, Gi, 2^30], [tebi, Ti, 2^40], [pebi, Pi, 2^50], [exbi, Ei, 2^50], [zebi, Zi, 2^70], [yobi, Yi, 2^80]

**Chuẩn SI (Prefix in byte):** Theo thứ tự Name, Abbr, Factor, SI size: [Kilo, K, 2^10, 10^3], [Mega, M, 2^20, 10^6], [Giga, G, 2^30, 10^9], [Tera, T, 2^40, 10^12], [Peta, P, 2^50, 10^15], [Exa, E, 2^60, 10^18], [Zetta, Z, 2^70, 10^21], [Yotta, Y, 2^80, 10^24]

**Thuật toán \*:** M\*Q, output C,A,Q

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 \* cải tiến (ko dấu + có dấu)**

Khởi tạo: A = 0; k = n; Q\_-1 = 0 (thêm 1 bit = 0 vào cuối Q)

Lặp khi k > 0 {

Xét 2 bit cuối của Q\_0 (số cuối của Q) và Q\_-1 {

= 10 thì A – M -> A

= 01 thì A + M -> A

= 00, 11 thì A không thay đổi }

Shift right [A, Q, Q-1] (giữ bit dấu)

k = k – 1 }

**Kết quả: [A, Q]**

A screenshot of a computer

Description automatically generated with low confidence

**Phép chia:** Q/M

Khởi tạo: A = n bit 0 nếu Q > 0; A = n bit 1 nếu Q < 0; k = n

Lặp khi k > 0 {

Shift left (SHL) [A, Q]

A – M -> A

# Nếu A < 0: Q\_0 (số cuối của Q)= 0 và A + M -> A

# Ngược lại: Q\_0 = 1

k = k – 1}

**Kết quả: Q là thương, A là số dư**

A picture containing text, number, screenshot

Description automatically generated

**Biểu diễn số thực sau theo dạng số chấm động chính xác đơn** (32 bit): X = -5.25

- Bước 1: Đổi X sang hệ nhị phân

X = -5.25 = -101.01

- Bước 2: Chuẩn hóa theo dạng ±1.F \* 2E

X = -5.25 = -101.01 = -1.0101 \* 2^2

- Bước 3: Biểu diễn Floating Point

¤ Số âm: bit dấu Sign = 1

¤ Số mũ E = 2 à Phần mũ exponent với số thừa K=127 được biểu diễn: Exponent = E + 127 = 2 + 127 = 129 = 1000 0001

¤ Phần định trị = 0101 0000 0000 0000 0000 000 (Thêm 19 số 0 cho đủ 23 bit).

=> Kết quả nhận được: 1 1000 0001 0101 0000 0000 0000 0000 000.

**Số thực đặc biệt:**

- Số 0: Exponent = 0, Significand = 0

- Số không thể chuẩn hóa (denormalized): Exponent = 0, Significand != 0

- Số vô cùng (infinity): Exponent = 111…1 (toàn bit 1), Significand = 0

- Số báo lỗi (NaN – Not a Number): Exponent = 111…1 (toàn bit 1), Significand != 0

**Normalized number:**

- Largest positive normalized number: +1.[23 số 1] \* 2^127

0 1111 1110 1111 1111 1111 1111 1111 111

- Smallest positive normalized number: +1.[23 số 0] \* 2^-126

0 0000 0001 0000 0000 0000 0000 0000 000

- Tương tự cho số âm

**Denormalized number**

- Largest positive denormalized number: +0.[23 số 1]\*2^-127

0 0000 0000 1111 1111 1111 1111 1111 111

­Tuy nhiên IEEE 754 quy định là +0.[23 số 1] \* 2^-126 vì muốn tiến gần hơn với Smallest positive normalized number.

- Smalleset positive denormalized number: +1.[22 số 0] \* 2^-127

0 0000 0000 0000 0000 0000 0000 0000 001

Tuy nhiên IEEE 754 quy định là +0.[22 số 0] \* 2^-126

- Tương tự cho số âm

**Ngôn ngữ lập trình:** Là loại ngôn ngữ nhân tạo (Ví dụ: C/C++) được cấu thành bởi 2 yếu tố chính: **Từ vựng:** là các keyword (struct, enum, if, int…); **Ngữ pháp:** syntax (if(…){} else{}, do{} while()…). Ngôn ngữ lập trình giúp cho người sử dụng nó (gọi là lập trình viên) có thể diễn đạt và mô tả các hướng dẫn cho máy tính hoạt động theo ý muốn của mình. Độ phức tạp (trừu tượng) của các hướng dẫn này quyết định thứ bậc của ngôn ngữ: Độ phức tạp càng cao thì bậc càng thấp, vd C# là ngôn ngữ bậc cao hơn C.

**Instruction (lệnh máy):** Là dãy bit chứa yêu cầu mà bộ xử lý trong CPU (ALU) phải thực hiện. Instruction gồm 2 thành phần: Mã lệnh (opcode): thao tác cần thực hiện, Thông tin về toán hạng (operand): các đối tượng bị tác động bởi thao tác chứa trong mã lệnh.

**ISA (Instruction Set Architecture) – Kiến trúc bộ lệnh:** Tập lệnh dành cho những bộ vi xử lý có kiến trúc tương tự nhau. Một số ISA thông dụng: **Dòng vi xử lý 80x86 (gọi tắt x86)** của Intel (IA-16: Dòng xử lý 16 bit (Intel 8086, 80186, 80286); IA-32: Dòng xử lý 32 bit (Intel 80368 – i386, 80486 – i486, Pentium II, Pentium III …); IA-64: Dòng xử lý 64 bit (Intel x86-64 như Pentium D…)), **MIPS:** Dùng rất nhiều trong hệ thống nhúng (embedded system); **PowerPC của IBM.**

**Thiết kế ISA: CISC (x86) & RISC (MIPs):** Có 2 trường phái thiết kế bộ lệnh: **Complete Instruction Set Computer (CISC)** (bộ lệnh gồm rất nhiều lệnh, từ đơn giản đến phức tạp), **Reduced Instruction Set Computer (RISC)** (bộ lệnh chỉ gồm các lệnh đơn giản).

**Compiler:** Trình biên dịch ngôn ngữ cấp cao thành hợp ngữ. Compiler phụ thuộc vào: Ngôn ngữ cấp cao được biên dịch, Kiến trúc hệ thống phần cứng bên dưới mà nó đang chạy, Ví dụ: Compiler cho C <> Compiler cho Java, Compiler cho “C on Windows” <> “C on Linux”

**Assembler**: Trình biên dịch hợp ngữ thành ngôn ngữ máy. Một bộ vi xử lý (đi kèm 1 bộ lệnh xác định) có thể có nhiều Assembler của nhiều nhà cung cấp khác nhau chạy trên các OS khác nhau. Ví dụ: Cùng là kiến trúc x86, nhưng có thể dùng A86, GAS, TASM, MASM, NASM. Assembly program phụ thuộc vào Assembler mà nó sử dụng (do các mở rộng, đặc điểm khác nhau giữa các Assembler)

**Linker:** Thực tế khi lập trình, ta sẽ dùng nhiều file (header / source) liên kết và kèm theo các thư viện có sẵn -> Cần chương trình Linker để liên kết các file sau khi đã biên dịch thành mã máy này (Object file), tập tin thực thi (ví dụ: .exe, .bat, .sh)

**Quá trình tạo file thực thi:**

**A picture containing text, screenshot, font, diagram

Description automatically generated**

**Loader:** Khi double click vào những tập tin thực thi, cần chương trình tính toán và tải vào memory để CPU xử lý.

**Quá trình thực thi lệnh (Execute Cycle):**

Tính địa chỉ lệnh, nạp lệnh, giải mã lệnh, tính địa chỉ của toán hạng, nạp toán hạng, thực hiện lệnh, tính địa chỉ của toán hạng chứa kết quả, ghi kết quả.

=> Các bước này lặp đi lặp lại cho tất cả các lệnh tiếp theo, quy trình này gọi là Instruction cycle – vòng lặp xử lý lệnh.

**Giới hạn biểu diễn số nguyên:**

- Dấu lượng: -127\_10 -> +127\_10

- Bù 1: -127\_10 -> +127\_10 (có 2 dạng biểu diễn số 0

- Bù 2: -128\_10 -> +127\_10

**Lệnh so sánh MIPS:**

**A == B:** beq $s0, $s1

**A != B:** bne $s0, $s1, label

**A < B**

slt $t0, $s0, $s1 # if (a < b) then $t0 = 1

bne $t0, $0, label #if (a < b) then goto label

**A > B:**

slt $t0, $s1, $s0 #if (b < a) then $t0 = 1

bne $t0, $s0, label #if (b < a) then goto label

**A >= B:**

slt $t0, $s0, $s1 #if (a < b) then $t0 = 1

beq $t0, $0, label #if (a >= b) then goto label

**A <= B**

slt $t0, $s1, $s0 #if (b < a) then $t0 = 1

beq $t0, $0, label #if (b >= a) then goto label

A picture containing text, screenshot, font, diagram

Description automatically generated



**Bộ lệnh MIPS:** Có tổng cộng 32 thanh ghi đánh số từ $0 đến $31: Càng ít càng dễ quản lý, tính toán càng nhanh, có thể truy xuất thanh ghi qua tên của nó. Mỗi thanh ghi có kích thước cố định 32 bit: Bị giới hạn bởi khả năng tính toán của chip xử lý, kích thước toán hạng trong các câu lệnh MIPS bị giới hạn ở 32 bit, nhóm 32 bit gọi là từ (word)

A picture containing text, screenshot, number, font

Description automatically generated

**Bộ lệnh x86:**

**- EAX (Accumulator):** Thanh ghi này được sử dụng như một thanh ghi tính toán chính trong quá trình thực thi lệnh. Nó thường được sử dụng để lưu trữ kết quả của các phép tính và để truyền các giá trị đầu vào cho các hàm hệ thống. **EBX (Base**): Thanh ghi này thường được sử dụng như một con trỏ đến dữ liệu trong bộ nhớ. Nó cũng có thể được sử dụng như một thanh ghi tổng quát để lưu trữ các giá trị trung gian. **ECX (Counter):** Thanh ghi này thường được sử dụng như một biến đếm trong các vòng lặp. Nó cũng có thể được sử dụng để truyền các tham số cho các hàm hệ thống và các lệnh điều khiển vòng lặp. **EDX (Data):** Thanh ghi này thường được sử dụng để lưu trữ các giá trị dữ liệu tạm thời hoặc để truyền các tham số cho các hàm hệ thống. Ngoài ra, nó cũng được sử dụng như một thanh ghi phụ để lưu trữ các kết quả tính toán. **ESI (Source Index):** Thanh ghi này thường được sử dụng như một con trỏ đến một nguồn dữ liệu trong quá trình truyền dữ liệu hoặc sao chép dữ liệu. **EDI (Destination Index):** Thanh ghi này thường được sử dụng như một con trỏ đến một vị trí đích trong quá trình truyền dữ liệu hoặc sao chép dữ liệu. **EBP (Base Pointer):** Thanh ghi này thường được sử dụng để trỏ đến một khung ngăn xếp (stack frame). Nó thường được sử dụng để truy cập các biến cục bộ và tham số của một hàm. **ESP (Stack Pointer):** Thanh ghi này thường được sử dụng để trỏ đến đỉnh ngăn xếp (stack) hiện tại. Nó làm việc cùng EBP để quản lý việc gọi và trả về các hàm, cũng như quản lý việc lưu trữ các biến cục bộ và tham số trên ngăn xếp. 

**Mạch tổ hợp cơ bản:** (giá trị ngõ ra hiện tại chỉ phụ thuộc vào giá trị ngõ vào hiện tại)

- Mạch toàn cộng (Full adder – FA): Mạch tổ hợp thực hiện phép cộng số học 3 bit. Gồm 3 ngõ vào (A, B: bit cần cộng; C\_i: bit nhớ) và 2 ngõ ra (kết quả có thể có từ 0 đến 3 với giá trị 2 và 3 cần 2 bit biểu diễn. S : Ngõ tổng, C\_0: ngõ nhớ).

- Mạch mã hoá nhị phân (binary encoder): Có 2^n (hoặc ít hơn) ngõ vào, n ngõ ra. Quy định chỉ có duy nhất một ngõ vào mang giá trị = 1 tại một thời điểm. Nếu ngõ vào = 1 đó là ngõ thứ k thì các ngõ ra tạo thành số nhị phân có giá trị = k.

- Mạch mã hoá theo thứ tự (Priority encoder): Các ngõ vào được xem như có độ ưu tiên. Giá trị ngõ ra phụ thuộc vào các ngõ vào có độ ưu tiên cao nhất. VD: Độ ưu tiên ngõ vào x3 > x2 > x1 > x0.

- Mạch giải mã (Decoder): Có n ngõ vào, 2^n (hoặc ít hơn) ngõ ra. Quy định chỉ có duy nhất một ngõ ra mang giá trị = 1 tại một thời điểm. Nếu các ngõ vào tạo thành số nhị phân có giá trị = k thì ngõ ra = 1 đó là ngõ thứ k.

- Mạch dồn (multiplexer-mux): Còn gọi là mạch chọn dữ liệu. Chọn n ngõ trong 2^n ngõ vào để quyết định giá trị của duy nhất 1 ngõ ra. Mạch dồn (2^n) – 1 có 2^n ngõ nhập, 1 ngõ xuất và n ngõ nhập chọn.

- Mạch tách demultiplexer (DEMUX): Chọn n ngõ trong 2^n ngõ vào để quyết định giá trị của duy nhất 1 ngõ ra. Mạch DEMUX 1 – 2^n có 1 ngõ nhập, 2^n ngõ xuất và n ngõ nhập chọn.



**Bộ nhớ ngoài thông dụng**: Băng từ (Magnetic tape), Đĩa từ (Magnetic disk) tức đĩa cứng hoặc mềm, Đĩa quang (Optical disk) tức CD or DVD (Digital Video Disk: chỉ dùng trên ổ đĩa xem video, ghi 1 hoặc 2 mặt, mỗi mặt có 1 (single layer) hoặc 2 lớp (double layer), thông dụng là 4.7 gb/lớp, HD-DVD & Blue-ray Disc, Flash disk.

**Hệ thống lưu trữ lớn RAID (Redundant Array of Inexpensive):** Tập các đĩa cứng vật lý được OS xem như 1 ổ logic duy nhất có dung lượng lớn. Dữ liệu được lưu trữ phân tán trên các ổ đĩa vật lý -> truy cập song song (nhanh). Có thể sử dụng dung lượng dư thừa để lưu trữ các thông tin kiểm tra chẵn lẻ, cho phép khôi phục lại thông tin khi đĩa bị hỏng -> an toàn thông tin. Có 7 loại phổ biến (RAID 0 – 6).

**Bộ nhớ trong:**

**- Bộ nhớ chính:** Tồn tại dưới dạng các module nhớ DRAM (Dynamic Random Access Memory). Chứa các chương trình đang thực hiện và các dữ liệu đang thao tác. Tồn tại trên mọi htmt. Bao gồm các ngăn nhớ được đánh địa chỉ trực tiếp bởi CPU. Dung lượng của bộ nhớ chính < Không gian địa chỉ bộ nhớ mà CPU quản lý. Sử dụng công nghệ lưu trữ DRAM.

**- Bộ nhớ đệm:** Tích hợp trên chip của CPU. Sử dụng công nghệ lưu trữ SRAM (Static Random Access Memory). Là bộ nhớ trung gian giữa CPU và bộ nhớ chính, có tác dụng làm giảm thời gian truy xuất bộ nhớ RAM. Khi cần đọc 1 ô nhớ từ bộ nhớ: Kiểm tra xem có trong cache chưa? Nếu chưa có (cache miss): chép ô nhớ đó và 1 số ô nhớ lân cận từ bộ nhớ chính vào cache. Nếu đã có (cache hit): đọc từ cache, ko cần truy xuất bộ nhớ chính. Cache là bản copy 1 phần của bộ nhớ chính. Cache dùng công nghệ SRAM, có tốc độ truy xuất cao hơn so vs bộ nhớ chính xài công nghệ DRAM.

**Phân loại công nghệ RAM: SRAM (Static RAM):** Các bit được lưu trữ bằng các Flip-Flop -> Thông tin ổn định. Cấu trúc phức tạp. Dung lượng chip nhỏ. Tốc độ nhanh. Đắt tiền. Dùng làm bộ nhớ Cache. **DRAM (Dynamic RAM):** 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. Dùng làm bộ nhớ chính. Phân loại: **SIMM (Single Inline Memory Module):** Cũ, chậm;  **DIMM (Dual Inline Memory Module)**: Phổ biến; **RIMM (Rhombus Inline Memory Module):** Mới, nhanh nhất.

- Typical access time: SRAM: 5 – 25 ns; DRAM: 60 – 120 ns; Magnetic disk: 10 – 20 million ns / $ per Mbyte in 1997: SRAM $100 - $250; DRAM $5 - $10; Magnetic disk $0.10 - $0.20

**Hai nguyên lý truy xuất: Temporal locality (Cục bộ về thời gian)**: Nếu 1 ô nhớ đc dùng đến ở thời điểm hiện tại, nó dễ có khả năng đc dùng đến lần nữa trong tương lai gần. **Spatial locality (Cục bộ về không gian):** Nếu 1 ô nhớ đc dùng đến ở thời điểm hiện tại, những ô lân cận dễ có khả năng sắp đc dùng đến.

**Các pp ánh xạ:**

**- Direct mapping (Tag + Line + Word):** Một địa chỉ X trong bộ nhớ chính gồm N bit chia thành 3 trường: {**Trường Word** gồm W bit xác định kích thước 1 từ nhớ (ô) trong 1 Block = 1 Line -> Kích thước của Block/Line = 2^W. **Trường Line** gồm L bit xác định địa chỉ 1 line trong cache -> Số line trong cache = 2^L. **Trường Tag** gồm T bit -> T = N – (W + L)}. Xác định X có nằm trong Cache hay không (cache hit) hay vẫn nằm ở bộ nhớ chính (cache miss). Ví dụ: Không gian địa chỉ bộ nhớ chính = 4GB. Dung lượng cache = 256 KB. Kích thước 1 Line = 1 Block = 32 byte. Xác định cụ thể số bit cho 3 trường địa chỉ (W, L, T). **Giải:** Ta có bộ nhớ chính = 4GB = 2^32 byte -> N = 32 bit. Cache = 256 KB = 2^18 byte -> Ta có thể dùng 18 bit để đánh địa chỉ từng từ nhớ (ô) trong cache. Line (bao gồm nhiều từ nhớ) = 32 byte = 2^5 byte -> W = 5 bit. -> Số Line trong cache = 2^18 / 2^5 = 2^13 line -> L = 13 bit. Tag = T = N – (L + W) = 32 – (13 + 5) = 14 bit.

**- Associative mapping (Tag + Word):** Mỗi block có thể nạp vào bất kỳ Line nào của Cache, địa chỉ của bộ nhớ chính bao gồm 2 trường (trường **Word** giống như direct mapping, trường **Tag** dùng để xác định stt Block của bộ nhớ chính được lưu ở Cache, **Tag** xác định Block nào trong bộ nhớ chính đang nằm ở Line đó. Ví dụ: Ko gian địa chỉ bộ nhớ chính = 4GB. Kích thước 1 line = 1 block = 32 byte. Xđ (W, T). **Giải:** Bộ nhớ chính = 4GB = 2^32 byte = 2^5 byte -> W = 5bit. Tag = N – W = 32 – 5 = 27.

**- Set associative mapping (Tag + Set + Word):** Cache được chia thành các tập (Set). Mỗi một Set chứa 1 số Line (2, 4, 8, 16), ví dụ: 4 Line/Set -> 4-way associative mapping. Ánh xạ theo nguyên tắc sau: B0 -> S0, B1 -> S1, B2 -> S2,… Địa chỉ của bộ nhớ chính bao gồm 3 trường: Trường **Word** xác định kích thước 1 Block (= 1Line). Trường **Set** xác định thứ tự Set trong cache. Trường **Tag** dùng để xác định số thứ tự Block của bộ nhớ chính được lưu ở Cache. VD Không gian địa chỉ bộ nhớ chính = 4GB, dung lượng cache = 256 KB, kích thước 1 line = 1 block = 32 byte, hãy xđ (W, S, T) nếu tổ chức theo kiểu 4-way. **Giải:** Ta có bộ nhớ chính = 4GB = 2^32 byte -> N = 32 bit. Cache = 256 KB = 2^18 byte -> Ta có thể dùng 18 bit để đánh địa chỉ từng từ nhớ (ô) trong cache. Line = 32 byte = 2^5 byte -> W = 5 bit. -> Số Line trong cache = 2^18 / 2^5 = 2^13 Line -> L = 13 bit (dùng 13 bit để đánh địa chỉ từng Line trong Cache). Một Set trong Cache có 4 Line (xài 4 way) = 2^2 Line -> Số Set trong Cache = 2^13 / 2^2 = 2^11 Set -> S = 11 bit. Tag = T = N – (S + W) = 32 – (11 + 5) = 16 bit.

**So sánh các mapping:** **Direct:** bộ so sánh đơn giản, xác suất cache hit thấp. Bị xung đột thì cả 2 ô đều được lưu ở Line thứ 0. **Associative:** Để tìm ra Line chứa nội dung của 1 Block cần dò tìm và so sánh lần lượt Tag của tất cả các line của cache -> mất nhiều time. Xác suất cache hit cao, cần bộ so sánh phức tạp.

**Thuật toán thay thế:** Khi cần chuyển 1 Block mới vào trong Cache mà không tìm được Line trống, vậy phải bỏ Line nào ra? Một số cách chọn: **Random. FIFO:** Thay thế Line nào nằm lâu nhất trong Cache. **LFU (Least Frequently Used):** Thay thế Line nào trong Cache có số lần truy cập ít nhất trong cùng 1 khoảng thời gian. **LRU (Least Recently Used):** Thay thế Line nào trong Cache có thời gian lâu nhất không được tham chiếu đến. **Tối ưu nhất LRU**.