TỔNG QUAN MÁY TÍNH:

Thế hệ máy tính:
1. Vaccum tube: Bóng đèn chân không: ENIAC=1/10 UNIAC. IBM 700
2. Transistor: Linh kiện bán dẫn: IBM-7094
3. Integrated Curcuit: Vi mạch tích hợp: IBM-360
4.Microprocessor: Vi xử lý: Intel. XT Computer
5.Parallel Processing:xử lý song song: Tương lai

Định luật Moore:số lượng transistor trên mỗi đơn vị diện tích vi mạch sẽ tăng x2 sau mỗi khoảng thời gian từ 18->24 tháng, ⇔hiệu suất của vi mạch tăng theo.
** trước đây tăng mức độ tích hợp trans để tăng tốc độ. bây h chuyển hướng tăng xử lý song song.

5 thành phần cơ bản của CPU: Đơn vị điều khiển(Control Unit: CU), Đơn vị số học và logic(Arithmetic and logic Unit: ALU), Thanh ghi (Registers), Bộ nhớ đệm (Cache memory), Bus (hệ thống các đường dẫn)

Mainboard(7): The processor (CPU),Memory Chips,Hard disk,Power supply (psu),Graphic card (AGP),Floppy drive,PCI slots for sound cards, modems and etc…

Wafer(đế chip):Tấm silicon mỏng đã được cấy vật liệu khác nhau để tạo ra những vi mạch, kích thước wafer được tăng lên đã làm giá thành của một vi mạch trở nên rất rẻ.

Chip:có thể hiểu là mạch tích hợp(Integrated Circuit)gắn trên đế chip (wafer) nhằm xử lý các công việc trên máy tính, chip có kích thước rất nhỏ nhưng có thể chứa hàng chục triệu transistors.Hiện nay có các loại chip xử lý: 4, 8, 16, 32, 64 bit

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

Chipset thông dụng: CPU(Đơn vị xử lí trung tâm),**GPU**(Đơn vị xử lý đồ họa),**RAM**(Bộ nhớ truy cập tức thời, phục vụ CPU),**BÁN CẦU BẮC** (tích hợp mainboard-hỗ trợ truyền thông tin cho CPU, RAM, nằm sát CPU(AMD ko có vì tích hợp sẵn trên CPU)),**BÁN CẦU NAM**(tích hợp trên mainboard-quản lý thiết bị ngoại vi: HDD, Mouse, Keyboard…nằm cuối mainboard)

BIỂU DIỄN SỐ NGUYÊN

Công thức hệ cơ số q tổng quát
$x_{n-1}\dots x_1x_0 = x_{n-1}q^{n-1} + \dots + x_1q^1 + x_0q^0$

Chuyển đổi giữa các hệ cơ sở
10->2: chia dư 2 đem vào kq;2,16->10: công thức
10->16:chia dư 16 đem vào kq
2->16: nhóm 4bit rồi dùng bảng
16->2: ngược lại ở trên

Hệ nhị phân: Bit trái nhất: **MSB,** phải nhất: **LSB**

Số nguyên ko dấu: Max **1 byte:**1111 1111=2^8-1
Max **1 word:** $2^{16}-1$ **LSB=1** -> số lẻ

Số nguyên có dấu:
Dấu lượng:- MSB làm dấu
-Miền giá trị $[-(2^n-1), 2^n-1]$
-Đặc biệt: biểu diễn 0: 1000 0000 / 0000 0000
Bù 1: -MSB làm dấu
Âm->đảo bit phần trị
-Miền giá trị $[-(2^n-1), 2^n-1]$
-Đặc biệt: biểu diễn 0: 1000 0000(+0)/ 0000 0000(-0)
Bù 2: -Dương: 0xxx xxxx -> giữ nguyên
Âm: 1xxx xxxx-> Đảo bit phần trị + **1**
*Bù 2->Hệ 10: (-1) $\times x_{n-1}\times 2^{n-1} + \dots + x_1\times 2^1 + x_0\times 2^0$
-Miền giá trị $[-(2^n), 2^n-1]$
-Đặc biệt: biểu diễn 0: 1000 0000(+0)/ 0000 0000(-0)
-Giải quyết TH: 2TH biểu diễn 0, bit nhớ phát sinh sau khi tính toán phải đc cộng trực tiếp vào kq
Quá K: Chọn 1 số nguyên K>0 làm gtri dịch số phải nằm trong khoảng $[-(2^n), 2^n-1]$

Dịch bit, xoay bit:
SHL: 1100 -> 1000 SHR: 1100->0110
ROL: 1100->1001 ROR: 1010->0101

Phép toán logic
AND: 1&1=1 **OR:**0|0 = 0
XOR: != nhau = 1 **NOT:** ! phủ định
Hệ quả: **x(SHL)y = x*2^y, x(SHR)y = x/(2^y)**
AND: tắt bit**(&0)** **OR:** bật bit **(OR 1)**
XOR,NOT: đảo bit(XOR với 1= đảo bit)

Hệ quả:
Lấy giá trị tại bit thứ i của x: (x SHR i) AND 1
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

Toán tử:
Cộng: Overflow: (+) + (+) = (-) / (-) + (-) = (+)
Trừ: Overflow: (+) - (-) = (-) / (-) – (+) = (+)
Nhân: (M x Q): Thuật toán Booth
Khởi tạo: A=0; k=n; Q_{-1}=0(thêm 1 bit =0 vào cuối Q)
Lặp khi k > 0{
Nếu 2 bit cuối Q_0Q_{-1} {= 10 thì A-M -> A / = 01 thì A+M -> A / = 00, 11: A ko đổi}
Shift right [A, Q, ; Q_{-1}] ; k= k-1}

Sau khi nhân thuật toán Booth thì đều gì xảy ra?
→ Thừa số thứ nhất M giữ nguyên giá trị còn thừa số Q mất hết giá trị

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: Q0 = 0 và A + M -> A #Ngược lại: Q0=1 k = k – 1} **Kết quả: Q là thương, A là số dư**

Chia có dấu:7/3=2 dư 1, 7/-3 = -2 dư 1,-7/3= - 2 dư -1 -7/-3 = 2 dư - 1

Chuẩn IEC: dùng trong phần mềm và lập trình để tính toán dung lượng(2^{10})
Chuẩn SI: Trong viễn thông và sản xuất phần cứng thì dùng chuẩn SI(10^3)
Lưu ý: 1 kilobyte = 1000 bytes theo tên gọi chuẩn SI còn trong chuẩn IEC nó gọi là kibibyte.

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 $X = -5.25_{10} = -101.01_2$
Bước 2: Chuẩn hóa theo dạng $\pm 1.F*2^E$ $X = -101.01 = -1.0101*2^2$
Bước 3: Biểu diễn Floating Point: - Số âm: bit dấu Sign = 1
Số mũ E=2->Phần mũ exponent với số thừa K=127 được biểu diễn
->Exponent = E + 127 = 2 + 127 = $129_{10} = 1000\ 0001_2$
Phần định trị=0101 0000 0000 0000 0000 0000(thêm 19 số 0 cho đủ 23bit)
Kết quả nhận được là 1 1000 0001 0101 0000 0000 0000 0000 0000
CHÍNH XÁC KÉP (64 bit) 1bit 11bits 52 bit
Exponent: số mũ(biểu diễn dưới số quá K) với (n là số bit lưu Exponent)
Chính xác đơn: K = $127(2^{n-1}-1)$ / **Chính xác kép**: K = 1023
Số thực đặc biệt
-Largest positive normalized number: +1.[23 số 1]*2^{127}
0 1111 1110 1111 1111 1111 1111 1111 1111
-Smallest positive normalized number: +1.[23 số 0] * 2^{-126}
0 0000 0001 0000 0000 0000 0000 0000 0000
-Tương tự cho số negative (số âm)
- Largest positive denormalized number: +0.[23 số 1] * 2^{-127}
0 0000 0000 1111 1111 1111 1111 1111 1111
Tuy nhiên IEEE 754 quy định là +0.[23 số 1]*2^{-126}.
"Smallest positive denormalized number: +1.[22 số 0]1 * 2^{-127}
0 0000 0000 0000 0000 0000 0000 0000 0001. Tuy nhiên IEEE 754 quy định là +0.[22 số 0]1 * 2^{-126}
-Tương tự cho số negative (số âm)
-Số 0: Exponent = 0, significand = 0
-Số không thể chuẩn hóa: Exponent = 0, Significand != 0
-Số vô cùng: Exponent = toàn bit 1, Signficand = 0
-Số báo lỗi (NaN): Exponent = toàn bit 1, Signifand != 0

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()...).
Độ phức tạp (trừu tượng) của các hướng dẫn này quyết định thứ bậc ngôn ngữ
¤ Độ phức tạp càng cao thì bậc càng thấp
¤ Ví dụ: C Sharp (C#) là ngôn ngữ bậc cao hơn C

NGÔN NGỮ MÁY (MACHINE LANGUAGE)
Ngôn ngữ máy cho phép người lập trình đưa ra các hướng dẫn đơn giản mà bộ vi xử lý (CPU) có thể thực hiện được ngay:

Các hướng dẫn này được gọi là chỉ thị / lệnh (instruction) hoặc mã máy.

Mỗi bộ vi xử lý (CPU) có 1 ngôn ngữ riêng, gọi là bộ lệnh (instruction set).
Trong cùng 1 dòng vi xử lý (processor family) bộ lệnh gần giống nhau.

INSTRUCTION
Là dãy bit chứa yêu cầu mà bộ vi 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

COMPILER - Trình biên dịch ngôn ngữ cấp cao -> hợp ngữ
Compiler phụ thuộc vào: Ngôn ngữ cấp cao được biên dịch và Kiến trúc hệ thống phần cứng bên dưới mà nó đang chạy
ASSEMBLER - Trình biên dịch hợp ngữ -> 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
Assembly program phụ thuộc vào Assembler mà nó sử dụng (do các mở rộng, đặc điểm khác nhau giữa các Assembler)
Làm sao để chạy những tập tin này trên máy tính? -> **Linker & Loader**
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)
Khi double click vào những tập tin thực thi, cần chương trình tính toán và tải vào memory để CPU xử lý -> **Loader**

ISA (Instruction Set Architecture)
-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..)

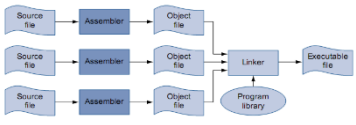
Thiết kế ISA: CISC & RISC

RISC (Reduced Instruction Set Computer)	CISC (Complex Instruction Set Computer)
Bộ lệnh giới hạn (có thể từ 80-100 lệnh).	Số lượng lệnh lớn (200-500 lệnh).
Độ dài lệnh cố định.	Độ dài lệnh thay đổi.
Định dạng/mã hóa lệnh đơn giản.	Định dạng/mã hóa lệnh phức tạp.
Ít chế độ địa chỉ.	Nhiều chế độ địa chỉ.
Lệnh chủ yếu dựa vào các thanh ghi, bộ nhớ chỉ được truy cập khi nạp và lưu trữ.	Lệnh có thể thực hiện nhiều hơn 1 việc (tức là, một lệnh có thể thực hiện 2 hoặc nhiều hành động).
Nhiều thanh ghi hơn.	Ít thanh ghi hơn.
Tập trung vào phần mềm để tối ưu hóa bộ lệnh.	Tập trung vào phần cứng để tối ưu hóa bộ lệnh.
Kiểm soát bằng mạch cứng.	Kiểm soát bằng vi lập trình (dựa trên ROM).
Giải mã lệnh đơn giản.	Giải mã lệnh phức tạp.
Dễ dàng triển khai đường ống (pipeline) để tăng tốc độ thực hiện.	Triển khai đường ống (pipeline) không dễ dàng (phức tạp).
Thời gian thực hiện lệnh ngắn hơn.	Thời gian thực hiện lệnh dài hơn.
Kích thước mã lớn.	Kích thước mã nhỏ.
Mở rộng mã có thể là một vấn đề.	Mở rộng mã không phải là một vấn đề.
Tiêu thụ ít điện năng.	Tiêu thụ nhiều điện năng.

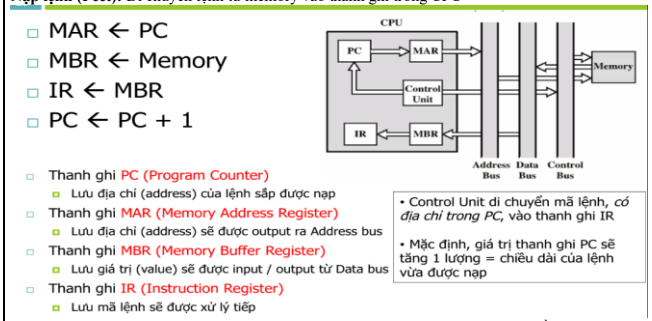
CHU KỲ XỬ LÝ LỆNH CỦA CPU

Thực thi lệnh (Excute): giải mã lệnh và thực thi thao tác yêu cầu
- 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

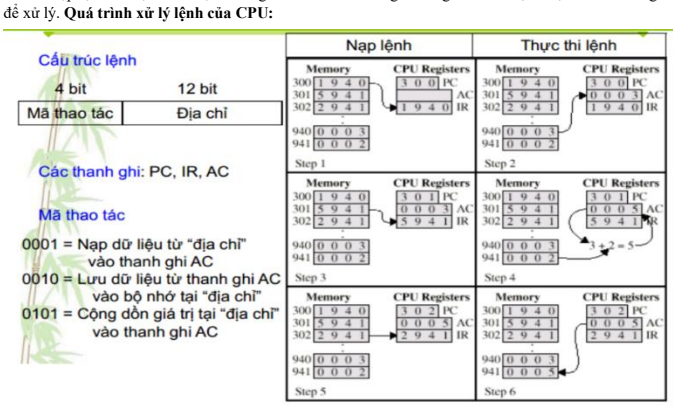
QUÁ TRÌNH TẠO FILE THỰC THI



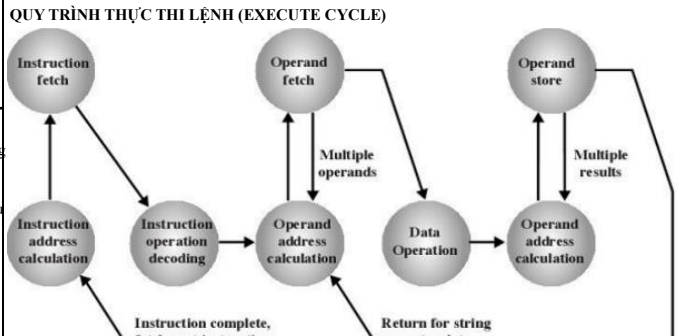
Nạp lệnh (Fect): Di chuyển lệnh từ memory vào thanh ghi trong CPU



→ Khi nạp lệnh thì đọc ô nhớ tại thanh ghi PC và sau đó tăng thanh ghi PC lên 1, rồi lệnh vào thanh ghi IR để xử lý. **Quá trình xử lý lệnh của CPU:**



QUY TRÌNH THỰC THI LỆNH (EXECUTE CYCLE)



1. Các bước này được lặp đi lặp lại cho tất cả các lệnh tiếp theo
2. Quy trình này gọi là Instruction cycle – vòng lặp xử lý lệnh

BỘ NHỚ

Từ trái sang phải: dung lượng tăng dần, tốc độ giảm dần, giá thành 1 bit giảm dần:
Thanh ghi -> Cache L1 -> Cache L2 -> bộ nhớ chính -> bộ nhớ ngoài -> đĩa từ bang từ.

Phân loại:
PP truy cập: tuần tự (băng từ), trực tiếp (các loại đĩa), ngẫu nhiên (bộ nhớ bán dẫn RAM, ROM), liên kết (cache).
Kiểu vật lý: bộ nhớ bán dẫn (bộ nhớ từ HDD, FDD), bộ nhớ quang (CD-ROM, DVD).
Bộ nhớ ngoài: băng từ (magnetic tape), đĩa từ (magnetic disk), đĩa quang (optical disk), flash disk.
Hệ thống nhớ lưu trữ lớn RAID (redundant array of inexpensive disk):
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 + Sử dụng lượng dư thừa để lưu trữ thông tin, khôi phục thông tin khi đĩa bị hỏng -> an toàn thông tin -> Có 7 loại RAID (0-6).

CÁC LOẠI RAID 0, 1, 5
-RAID 0: dung lượng = tổng dung lượng các ổ. Hỏng 1 ổ là mất hết dữ liệu.
-RAID 1: duy trì dung lượng của chỉ 1 ổ. Hỏng 1 ổ thì vẫn còn do còn bản copy trong ổ khác.
-RAID 5: cải tiến hơn, có dung lượng ít hơn 1 ổ (5 ổ dùng RAID 5 sẽ có dung lượng 4 ổ). Hỏng 1 ổ thì dữ liệu vẫn an toàn hỏng 2 ổ => mất.

- Bộ nhớ trong (Bộ nhớ chính): dạng các module nhớ DRAM (bit lưu trữ trên tụ điện -> cần mạch refresh, cấu trúc đơn giản, dung lượng lớn, speed chậm, rẻ tiền, dùng làm bộ nhớ chính).
*Chương trình đang thực hiện, dữ liệu đang thao tác, tồn tại trên mọi hệ thống, ngắn nhớ có đánh địa chỉ trực tiếp bởi CPU, dung lượng < gian địa chỉ bộ nhớ mà CPU quản lí, công nghệ lưu trữ DRAM.
***Phân loại Dram:** SIMM (cũ, chậm); DIMM (phổ biến); RIMM (mới, nhanh nhất).
***Bộ nhớ đệm:** tích hợp trên chip của CPU, sử dụng công nghệ lưu trữ SRAM (bit lưu trữ bằng các flip flop -> thông tin ổn định, cấu trúc phức tạp, dung lượng chip nhỏ, speed nhanh, đắt tiền, dùng làm bộ nhớ Cache).

Khi đọc 1 ô nhớ từ bộ nhớ, nếu chưa có cache miss: chép ô nhớ đó và 1 số ô nhớ lân cận từ bộ nhớ chính vào cache; nếu có cache hit: đọc từ cache, không cần truy xuất bộ nhớ chính.
***Cache** là bản copy một phần bộ nhớ chính, dùng công nghệ SRAM, truy xuất cao hơn bộ nhớ chính.
Nguyên lí cơ sở khi truy xuất: temporal locality (cục bộ về thời gian), spatial locality (cục bộ về không gian).
CPU (truy xuất từng byte/word) -> cache (truy xuất từng block) -> RAM.

PHÂN BIỆT RAM TĨNH (SRAM) VÀ RAM ĐỘNG (DRAM)

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 chip nhỏ - Tốc độ nhanh - Đắt tiền - Dùng làm bộ nhớ Cache	- 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

***PP ánh xạ:**
+Direct mapping (ánh xạ trực tiếp).
Ví dụ: bộ nhớ chính = 4GB = 2^{32}byte -> N=32 bit.
Dung lượng cache = 256KB = 2^{18} byte -> dùng 18 bit đánh địa chỉ ô nhớ trong Cache.
1 line = 1 block = 32 byte =2^5byte -> W=5 bit.
Số line trong Cache = $\frac{2^{18}}{2^5}$ = 2^{13} line -> L=13 bit.
T=N−(L+W)=32−(13+5)=14 bit. Xđịnh(W,L,T)
Line thứ: L = M % Số Line trong cache = M % 213 (13 bit)
Tag tại Line đó: T = M / Số Line trong cache = M / 213 (14 bit)
***Đánh giá : Bộ so sánh đơn giản**
``Xác suất cache hit thấp - ``Giả sử muốn truy xuất đồng thời từ nhớ (ô) X tại
***Block thứ 0 và ô thứ Y tại Block thứ 2L thì sao?**
(L: Tổng số Line trong Cache)
=>Bị xung đột thì cả 2 ô này đều sẽ được lưu ở Line thứ 0
(0 % 2L = 2L % 2L = 0)
+Associative mapping:
Ví dụ: Bộ nhớ chính = 4 GB = 2^{32} byte à N = 32 bit. Line (bao gồm nhiều từ nhớ) = 32 byte = 2^5 byte -> W = 5 bit (Dùng 5 bit để đánh địa chỉ nội bộ các từ nhớ (ô) trong 1 Line)
Tag = T = N – W = 32 – 5 = 27 bit.
Giả sử ta có Block thứ M (27 bit, giá trị từ 0 -> 2^27- 1) muốn lưu vào cache thì sẽ lưu ở bất kỳ Line nào miễn sao có Tag tại Line đó là: T = M (27 bit).
Đánh giá: Để tìm ra Line chứa nội dung của 1 Block, cần dò tìm và so sánh lần lượt với Tag của tất cả các Line của Cache => Mất nhiều thời gian.
``Xác suất cache hit cao. ``Cần bộ so sánh phức tạp.
+ Set associative mapping:
Ví dụ: Bộ nhớ chính = 4 GB = 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 = 25 byte => W = 5 bit (Dùng 5 bit để đánh địa chỉ nội bộ các từ nhớ (ô) trong 1 Line)
=> 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 = 2^2 Line
=> Số Set trong Cache = 2^13 / 2^2 = 2^11 Set => S = 11 bit (Dùng 11 bit để đánh địa chỉ các Set trong Cache)
¨Tag = T = N – (S+ W) = 32 – (11 + 5) = 16 bit.

THAM SỐ ẢNH HƯỞNG CACHE
Block size: Nhỏ quá: giảm tính lân cận (spatial locality). Lớn quá: số lượng block trong cache ít, thời gian chuyển block vào cache lâu (miss penalty).
Cache size:
¤ Nhỏ quá: số lượng Block có thể lưu trong cache quá ít, làm tăng tỷ lệ cache miss.
¤ Lớn quá: tỷ lệ giữa vùng nhớ thực sự cần thiết so với vùng nhớ lưu vào cache sẽ thấp, nghĩa là overhead (tổng chi phí) sẽ cao, tốc độ truy cập cache giảm

THUẬT TOÁN THAY THẾ
(KHÔNG DÙNG CHO DIRECT MAPPING)
Được sử dụng khi cần chuyển 1 Block mới vào trong Cache mà không tìm được Line trống.
¤ Random: Thay thế ngẫu nhiên
¤ FIFO (First In First Out): Thay thế Line nào nằm lâu nhất trong Cache.
¤ LFU (Least Frequently Used):
¤ LRU (Least Recently Used): **Tối ưu nhất.**

WRITE POLICY (CHÍNH SÁCH ĐỒNG BỘ)
Nếu 1 Line bị thay đổi trong Cache, khi nào sẽ thực hiện thao tác ghi lên lại RAM ?
¤ Write Through: ngay lập tức
¤ Write Back: khi Line này bị thay thế
"Nếu nhiều processor chia sẻ RAM, mỗi processor có cache riêng:
¤ Bus watching with WT: loại bỏ Line khi bị thay đổi trong 1 cache khác
¤ Hardware transparency: tự động cập nhật các cache khác khi Line bị 1 cache thay đổi
¤ Noncacheable shared memory: phần bộ nhớ dùng chung sẽ không được đưa vào cache

SỐ LƯỢNG VÀ CÁC LOẠI CACHE
Có thể sử dụng nhiều level: L1, L2, L3...

¨Các cache ở mức thấp gọi có thể là on-chip, cache mức cao thường là off-chip và được truy cập thông qua external bus hoặc bus dành riêng
¨Cache có thể dùng chung cho cả data và instruction hoặc riêng cho từng loại.

CACHE INTEL
80486: 8 KB cache L1 trên chip (on-chip)
¨Pentium: có 2 cache L1 trên chip Cache lệnh: 8 KB Cache dữ liệu: 8 KB
¨Pentium 4 (2000): có 2 level cache L1 và L2 trên chip
¤ Cache L1: 2 cache, mỗi cache 8 KB. Kích thước Line = 64 byte 4-way associative mapping
¤ Cache L2: 256 KB. Kích thước Line = 128 byte 8-way associative mapping

DUNG LƯỢNG ĐĨA
Dung lượng đĩa = số head x số track/cyclinder 1 head x số sector 1 track. => đổi qua MB, GB. 1MB = 1024KB, 1GB = 1024MB

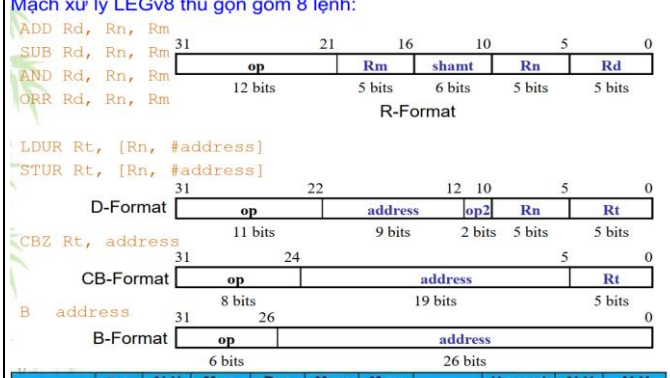
LEGv8
Lý thuyết LEGv8:
- 1 phần ARMv8, hỗ trợ 32-bit và 64-bit, nhưng tập trung 64-bit.
-Thanh ghi tổng quát: LEGv8 cung cấp 32 thanh ghi tổng quát kích thước 64 bit (DoubleWord) (X0-X31)
-Ngoài ra còn có **32 thanh ghi con** kích thước 32 bit (Word) (W0-W31)
→ Tính toán nhanh hơn, hỗ trợ format lệnh 32 bit
-Thanh ghi cờ: Negative (N), Zero (Z), Carry (C) và Overflow (V).
-Thanh ghi số thực: 32 bit: S0-S31. 64 bit: D0-D31
-X0-X7: Làm đối số / kết quả trả về của hàm. X8: Chứa (vị trí) địa chỉ kết quả trả về. X9 – X15: Thanh ghi tạm. X16 – X17 (IP0 – IP1): Thanh ghi tạm của linker hoặc cũng dung làm thanh ghi tạm cho các trường hợp khác. X18: platform register cho code không phụ thuộc flatform hoặc làm thanh ghi tạm. X19 – X27: Thanh ghi lưu trữ X28 (SP): Trỏ đến đỉnh ngăn xếp (stack pointer). X29 (FP): Trỏ đến khung trang (frame pointer). X30 (LR): Link register (địa chỉ quay về) XZR (X31): Chứa hằng số 0.
Cấu trúc chương trình hợp ngữ:
-section .data object tĩnh ko đổi
-section .bss biến thay đổi
-section .text viết code

LEGv8	x86
"Fixed Length Instructions" (Chỉ thị có độ dài cố định) - Tất cả các chỉ thị có kích thước 4 byte. - Mạch xử lý đơn giản hơn ⇒ Xử lý nhanh hơn. - Chỉ thị nhảy: bội số của 4 byte.	**"Variable Length Instructions" (Chỉ thị có độ dài thay đổi)** - Kích thước chỉ thị thay đổi từ 1 byte đến 16 byte. ⇒ Mã nguồn có thể nhỏ hơn (khoảng 30%?). - Sử dụng bộ nhớ đệm (cache) hiệu quả hơn. - Các chỉ thị có thể có hằng số/ngay lập tức 8-bit hoặc 32-bit.
"Three-Operand Architecture" (Kiến trúc ba toán hạng) - 2 toán hạng nguồn và 1 toán hạng đích. - Ví dụ: ADD X9, X10, X11 // X9=X10+X11 - **Ưu điểm:** Ít chỉ thị hơn ⇒ Xử lý nhanh hơn.	**"Two-Operand Architecture" (Kiến trúc hai toán hạng)** - 1 toán hạng nguồn và 1 toán hạng vừa đóng vai trò toán hạng đích và nguồn. - Ví dụ: add EBX, EAX ; EBX=EBX+EAX **- Ưu điểm**: Lệnh ngắn hơn ⇒ Mã nguồn nhỏ hơn.
"Load-Store Architecture" (Kiến trúc tải-lưu trữ) - Chỉ có chỉ thị Load/Store truy cập bộ nhớ; phần còn lại của các chỉ thị hoạt động trên các thanh ghi và hằng số. - Ví dụ: LDUR X9, [X19, #8] // X20=X20 + Mem[X19+8] **- Ưu điểm:** Mạch xử lý đơn giản hơn ⇒ Dễ dàng tăng tốc độ bằng cách sử dụng các kỹ thuật song song. **- RISC** - phẳng 0->2^n-1 => truy cập vào thẳng địa chỉ vật lý - có thanh ghi lưu địa chỉ trả về	**"Register-memory architecture" (Kiến trúc thanh ghi-bộ nhớ)** - Tất cả các chỉ thị đều có thể truy cập bộ nhớ. - Ví dụ: ADD EAX, [ESI+8]; EAX = EAX + Mem[ESI+8] **- Ưu điểm:** Ít lệnh hơn ⇒ Mã nguồn nhỏ hơn. **- CISC** - chia segments, địa chỉ đoạn địa chỉ offset - ko có thanh ghi chứa địa chỉ trả về

LỆNH LEGv8:
-Tính toán số học: ADD, SUB, ADDI, SUBI, MUL, SMULH, SMULL(signed MUL 64 High/Low), SDIV, UDIV. VD: ADD X1, X2, X3: X1 = X2 + X3. ADDS, SUBS, ADDIS, SUBIS thay đổi cờ.
-Di chuyển dữ liệu: **(Double word)** LDUR: load từ bộ nhớ sang thanh ghi. STUR: lưu từ thanh ghi ra bộ nhớ. VD: STUR X1,[X2, #40] : lưu giá trị X1 vào địa chỉ X2+40. X2 là base register. 1 phần từ mảng = +8. =>A[4]:địa chỉ A+4*8
-Di chuyển dữ liệu: (1 word) Load word: LDURSW, Store word: STURW. VD: LDURSW X2, [X1,#40]. Lệnh này nạp giá trị 1 word tại ô nhớ có địa chỉ (X1 + 40) vào 4 byte thấp của thanh ghi X2.
-Di chuyển dữ liệu: (1/2 word) Load half: LDURH , Store half: STURH. -Di chuyển dữ liệu: (1 byte) Load byte: LDURB, Store byte: STURB. -Di chuyển dữ liệu độc quyền: LEGv8 còn hỗ trợ load, store độc quyền trong trường có nhiều luồng, hoặc tiến trình cùng truy cập vùng nhớ. Load exclusive register: LDXR, Store exclusive register : STXR
-Gán giá trị vào thanh ghi: MOVZ X1, #20, LSL #0. gán 20 vào và gán các bit khác = 0. MOVK X1, #20, LSL #0 . gán 20 vào và giữ các bit khác. MOV X1, X2: gán thanh ghi X1 = X2. ADRP x0, num1 // Đưa địa chỉ cơ sở của biến num1 vào thanh ghi x0 ADD x0, x0, :lo12:num1 // Cộng offset của biến num1 vào thanh ghi x0
-Luận lý: AND, ORR, EOR. không hỗ trợ NOT. Cách thực hiện NOT: EOR với giá trị full 1. ANDI, ORRi, EORI cho phép toán hạng 2 là hằng số
-LSL, RSL. RSA, không có LSA.
-Rẽ nhánh: CBZ X1, label => if(X1 == 0) goto label CBNZ X1, label // if (X1 != 0) goto label. CMP X1, X2.B.EQ label => nếu = thì nhảy. Rẽ nhánh ko điều kiện: B label, BR X1(đi tới địa chỉ X1), BL label(X30 = PC+4 and go) Điều kiện của rẽ nhánh có điều kiện: CMP X1, X2. BL.cond label. Cond = EQ X1 == X2 NE X1 != X2 GT (unsigned: HI) X1 > X2 LT (unsigned: LO) X1 < X2 GE (unsigned: HS) X1 >= X2 LE (unsigned: LS) X1 <= X2
-Gọi hàm: bl function(function đc khai báo trong section .text) Đối số input (argument input): X0 - X7/Kết quả trả về (return …): X0 - X1/Các thanh ghi không bảo toàn: X0-X18/Các thanh ghi bảo toàn: X19-X28/Địa chỉ quay về LR (link return): X30
-Stack: Cấu trúc dữ liệu lý tưởng để chứa tạm các giá trị trong thanh ghi. Thường chứa địa chỉ trả về, các biến cục bộ của trình con, nhất là các biến có cấu trúc (array, list…) không chứa vừa trong các thanh ghi trong CPU. Được định vị và quản lý bởi stack pointer. Đưa dữ liệu từ thanh ghi vào stack: STP X0, [SP, #-8] // SP = SP – 8, Lưu X0 vào vị trí SP, Lấy dữ liệu từ stack chép vào thanh ghi LTP X0, [SP, #8] // Gán X0 = giá trị tại SP, SP = SP + 8 Trong LEGv8 dành sẵn 1 thanh ghi X28 (hoặc SP) để lưu trữ stack pointer Exit program: movz x0, #0; movz x8, #93; svc#0

MẠCH LEGv8
¤ **Reg2Loc**: Tín hiệu điều khiển này chọn nguồn cho toán hạng thanh ghi thứ hai. Nếu **Reg2Loc = 1**, toán hạng thứ hai sẽ được lấy từ trường Rt (thanh ghi đích trong một số lệnh nhánh hoặc điều kiện). Nếu **Reg2Loc = 0**, toán hạng thứ hai sẽ được lấy từ trường Rm (trong các lệnh toán học và logic thông thường).
¤ **UncondBranch**: Khi tín hiệu này được kích hoạt (**UncondBranch = 1**), CPU sẽ thực hiện nhảy đến một địa chỉ mới được chỉ định mà không phụ thuộc vào điều kiện. Điều này tương tự với các lệnh nhảy vô điều kiện như **B** trong bộ lệnh LEGv8.
¤ **FlagBranch**: Tín hiệu này quyết định nhánh có được thực hiện hay không dựa trên cờ điều kiện. Nó được sử dụng cho các lệnh nhảy có điều kiện như **CBNZ (Compare and Branch if Not Zero)** và **CBZ (Compare and Branch if Zero)**, phụ thuộc vào trạng thái của cờ điều kiện.
¤ **ZeroBranch**: Tín hiệu này chỉ đạo CPU nhảy nếu cờ zero (Z) được thiết lập. Khi **ZeroBranch = 1**, CPU sẽ kiểm tra cờ zero. Nếu cờ này được thiết lập, CPU sẽ nhảy đến địa chỉ chỉ định, ví dụ như lệnh **CBZ**.
¤ **MemRead**: Khi tín hiệu này được kích hoạt (**MemRead = 1**), CPU sẽ thực hiện thao tác đọc dữ liệu từ bộ nhớ, ví dụ trong các lệnh tải (**LDUR**).
¤ **MemToReg**: Khi **MemToReg = 1**, dữ liệu đọc từ bộ nhớ sẽ được ghi vào thanh ghi đích. Nếu **MemToReg = 0**, dữ liệu từ ALU sẽ được ghi vào thanh ghi đích. Điều này quyết định nguồn dữ liệu đầu ra của thanh ghi đích là từ bộ nhớ hay từ ALU.
¤ **MemWrite**: Khi **MemWrite = 1**, CPU sẽ ghi dữ liệu từ thanh ghi nguồn vào bộ nhớ, ví dụ trong các lệnh lưu (**STUR**).
¤ **FlagWrite**: Tín hiệu này điều khiển việc ghi cờ điều kiện sau khi thực hiện các lệnh số học. Ví dụ, trong các lệnh **ADDS** và **SUBS**, cờ **N, Z, C, V** sẽ được cập nhật dựa trên kết quả của phép tính.
¤ **ALUSrc**: Tín hiệu này quyết định toán hạng thứ hai của ALU đến từ đâu. Nếu **ALUSrc = 1**, toán hạng thứ hai đến từ giá trị hằng số (immediate) trong lệnh. Nếu **ALUSrc = 0**, toán hạng thứ hai đến từ thanh ghi thứ hai (Rm).
¤ **ALUOp**: Tín hiệu này chỉ đạo ALU thực hiện phép toán nào. Ví dụ, **ALUOp = 10** có thể điều khiển ALU thực hiện các phép toán logic như **AND**, **OR**, hoặc các phép tính số học như **ADD**, **SUB**. **ALUOp = 00** thường liên quan đến các lệnh tải và lưu (**LDUR**, **STUR**).
¤ **RegWrite**: Tín hiệu này quyết định xem dữ liệu có được ghi vào một thanh ghi không. Khi **RegWrite = 1**, kết quả của ALU hoặc dữ liệu từ bộ nhớ sẽ được ghi vào thanh ghi đích.

Mạch xử lý LEGv8 thu gọn gồm 8 lệnh:



Instruction	Reg 2Loc	ALU Src	Mem toReg	Reg Write	Mem Read	Mem Write	Branch	Uncond Branch	ALU Op1	ALU Op0
R-Format	0	0	0	1	0	0	0	0	1	0
LDUR	X	1	1	1	1	0	0	0	0	0
STUR	1	1	X	0	0	1	0	0	0	0
CBZ	1	0	X	0	0	0	1	0	0	1
B	X	X	X	0	0	0	0	1	X	X

MẠCH TỔ HỢP
Gồm n ngõ vào (input); m ngõ ra (output)
¤ Mỗi ngõ ra là 1 hàm luận lý của các ngõ vào
¨Mạch tổ hợp không mang tính ghi nhớ: Ngõ ra chỉ phụ thuộc vào Ngõ vào hiện tại, không xét những giá trị trong quá khứ.
Độ trễ mạch (Propagation delay / gate delay)=Thời điểm tín hiệu ra ổn định-thời điểm tín hiệu vào ổn định
¤ Mục tiêu thiết kế mạch: làm giảm thời gian độ trễ mạch.
Thiết kế: Viết hàm SOP khi số lượng đầu ra 1 < 0 ngược lại hàm POS khi 1 > 0.

MẠCH TOÀN CỘNG: ``Mạch tổ hợp thực hiện phép cộng số học 3 bit ``Gồm 3 ngõ vào (A, B: bit cần cộng – Ci: bit nhớ) và 2 ngõ ra (kết quả có thể từ 0 đến 3 với giá trị 2 và 3 cần 2 bit biểu diễn – S: ngõ tổng, C0: ngõ nhớ	**MẠCH MÃ HÓA NHỊ PHÂN:** ``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 GIẢI MÃ: ``Có n ngõ vào, 2n (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:** ``Còn gọi là mạch chọn dữ liệu ``Chọn n ngõ trong 2n ngõ vào để quyết định giá trị của duy nhất 1 ngõ ra ``Mạch dồn 2n – 1 có 2n ngõ nhập, 1 ngõ xuất và n ngõ nhập chọn
MẠCH TÁCH:``Chọn n ngõ trong 2n ngõ vào để quyết định giá trị của duy nhất 1 ngõ ra ``Mạch DEMUX 1-2n có 1 ngõ nhập, 2n ngõ xuất và n ngõ nhập chọn	

Biểu đồ Karnaugh đánh số chuẩn:

			Z		
	0	1	3	2	
	4	5	7	6	Y
X	11	12	14	13	
	8	9	11	10	
		T			

MỘT SỐ DẠNG THỨC CƠ BẢN

x + 0 = x	x . 0 = 0
x + 1 = 1	x . 1 = x
x + x = x	x . x = x
x + x' = 1	x . x' = 0
x + y = y + x	xy = yx
x + (y + z) = (x + y) + z	x(yz) = (xy)z
x(y + z) = xy + xz	x + yz = (x + y)(x + z)
(x + y)' = x'.y' (De Morgan)	(xy)' = x' + y' (De Morgan)
(x')' = x	