ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG VIỆT - HÀN
Vietnam - Korea University of Information and Communication Technology

KIẾN TRÚC MÁY TÍNH

Chương 7

CẤU TRÚC VÀ CHỨC NĂNG VI XỬ LÝ

Nội dung

- Cấu trúc CPU
- Chu kỳ lệnh
- Kỹ thuật pipeline lệnh
- CISC & RISC

7.1. Cấu trúc CPU

- Đơn vị tính toán (ALU, FPU)
- Đơn vị điều khiển (Control Unit)
- Registers (data, address, instruction, control)
- Internal bus

7.1 Cấu trúc CPU

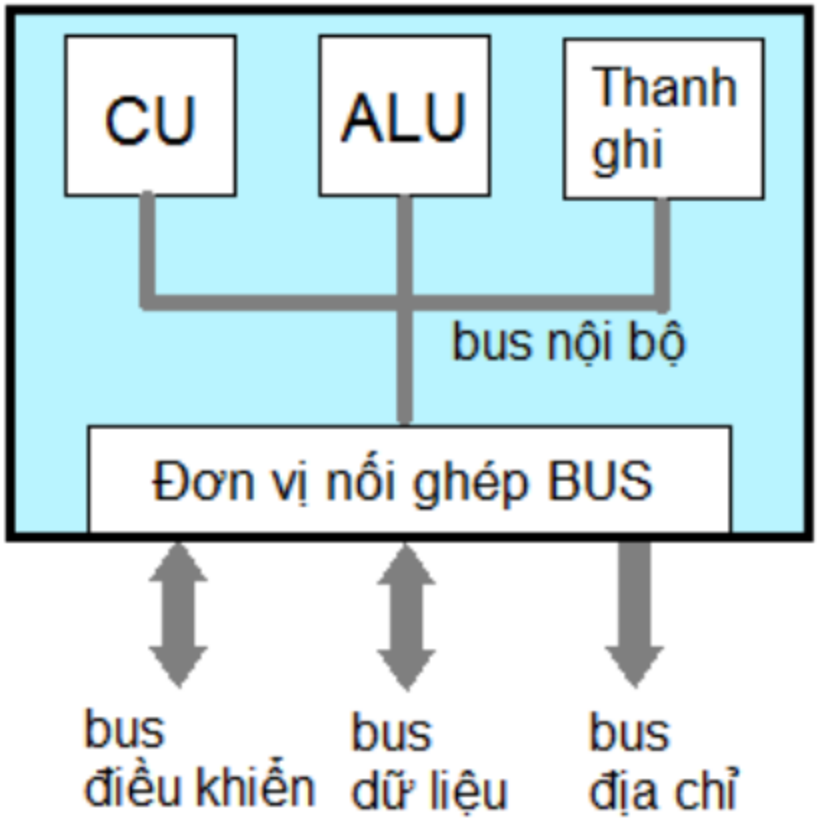
- Nhiệm vụ của CPU

- **Truy xuất lệnh**: Bộ xử lý đọc lệnh từ bộ nhớ (thanh ghi, bộ nhớ cache, bộ nhớ chính)
- **Giải mã lệnh**: Lệnh được giải mã để xác định hành động nào được yêu cầu
- **Truy xuất dữ liệu**: Việc thực thi một lệnh có thể yêu cầu thực hiện một số phép toán số học hoặc logic trên dữ liệu.
- **Xử lý dữ liệu**: Việc thực thi một lệnh có thể yêu cầu thực hiện một số phép toán số học hoặc logic trên dữ liệu.
- **Ghi dữ liệu**: Kết thúc việc thực hiện có thể yêu cầu ghi dữ liệu vào bộ nhớ hoặc module I/O.

→CPU cần lưu tạm thời một số dữ liệu → cần một bộ nhớ nhỏ bên trong.

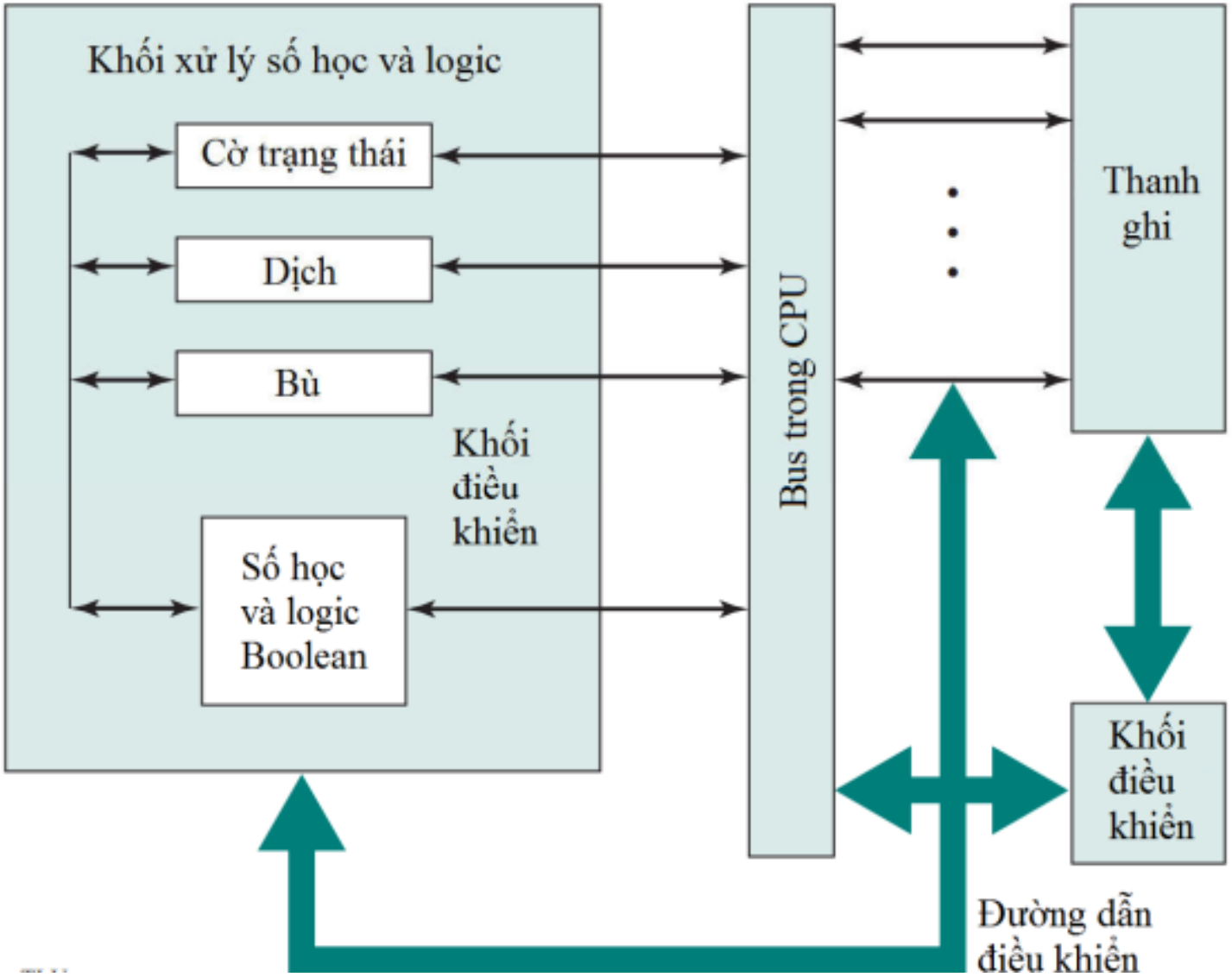
Các thành phần chính của CPU

- **Khối số học và logic (ALU):** thực hiện tính toán hoặc xử lý dữ liệu.
- **Khối điều khiển (CU):**
  - Kiểm soát việc di chuyển dữ liệu và lệnh vào/ra CPU
  - Điều khiển hoạt động của ALU
- **Thanh ghi:** bộ nhớ bên trong CPU, gồm một tập hợp các vị trí nhớ.
- **Đơn vị nối ghép bus**
- **Bus nội bộ**



Sơ đồ cấu trúc cơ bản của CPU

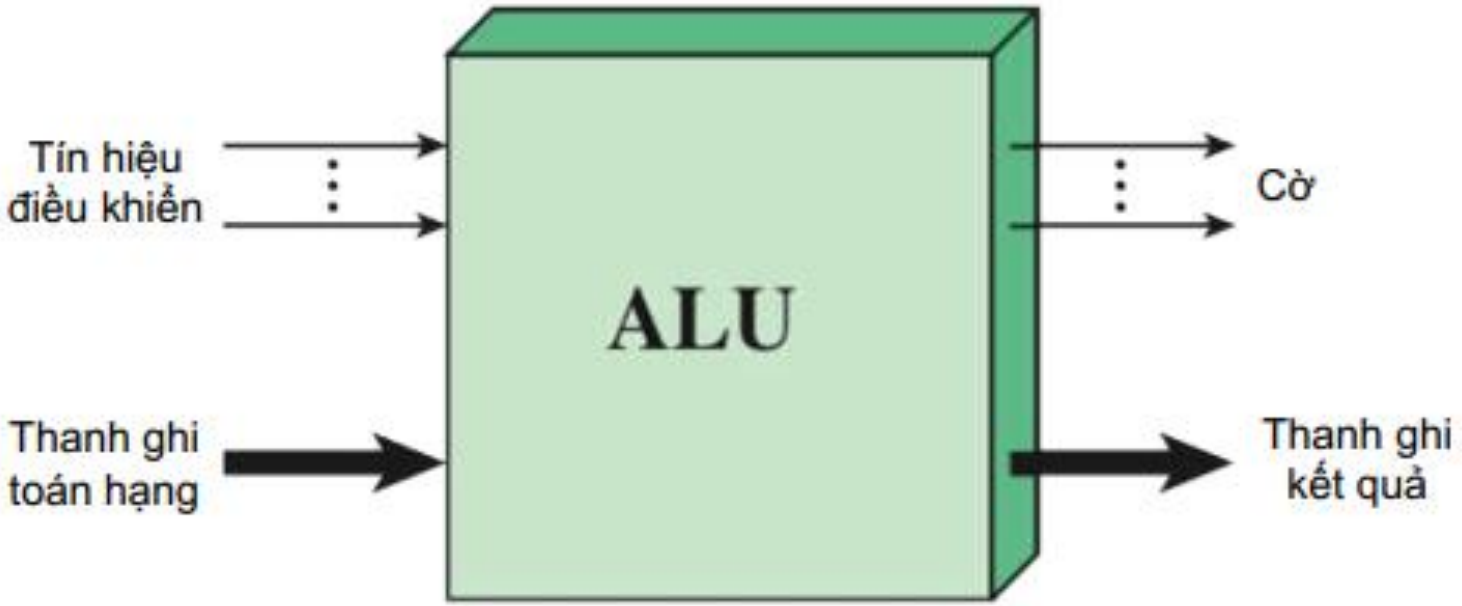
Cấu trúc bên trong CPU



7.1.1. Khối số học và logic (ALU)

- Là thành phần thực hiện phép toán số học và logic trên dữ liệu.
  - Xây dựng từ các linh kiện số đơn giản
  - Lưu trữ các chữ số nhị phân và thực hiện các phép toán logic Boolean đơn giản.
- Tất cả các bộ phận khác trong hệ thống máy tính đưa dữ liệu tới ALU để ALU xử lý rồi sau đó nhận lại kết quả.

Đầu vào và đầu ra ALU

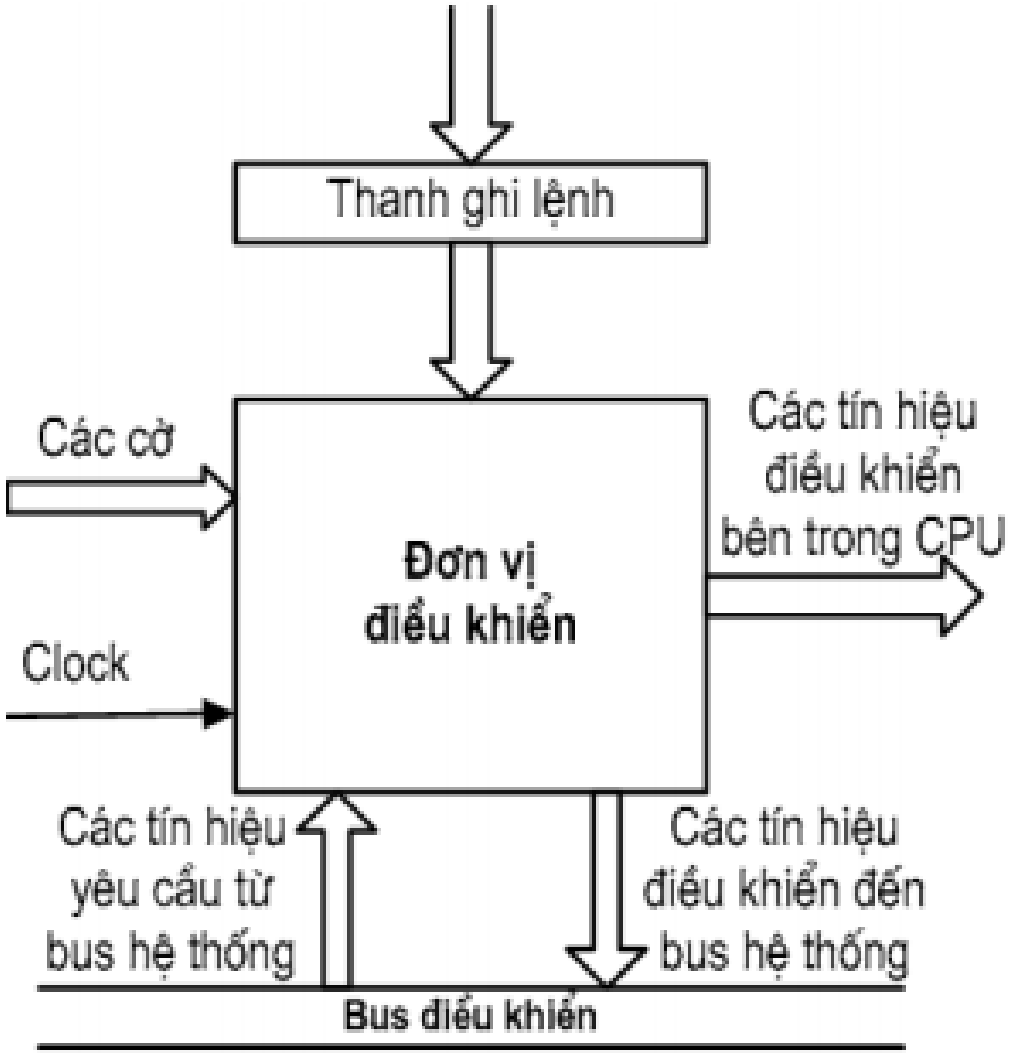


7.1.2 Khối điều khiển CU (Control Unit)

Chức năng:

- Điều khiển nhận lệnh từ bộ nhớ đưa vào thanh ghi lệnh
- Tăng nội dung của PC để trở sang lệnh kế tiếp
- Giải mã lệnh đã được nhận để xác định thao tác mà lệnh yêu cầu
- Phát ra các tín hiệu điều khiển thực hiện lệnh
- Nhận các tín hiệu yêu cầu từ bus hệ thống và đáp ứng với các yêu cầu đó.

Mô hình kết nối CU



- Tín hiệu đến CU
  - Clock: tín hiệu nhịp từ mạch tạo dao động bên ngoài.
  - Mã lệnh từ thanh ghi lệnh đưa đến để giải mã
  - Các cờ từ thanh ghi cờ cho biết trạng thái của CPU
  - Các tín hiệu yêu cầu từ bus điều khiển
- Tín hiệu đi ra từ CU
  - Điều khiển các thanh ghi
  - Điều khiển ALU
- Các tín hiệu điều khiển bên ngoài CPU
  - Điều khiển bộ nhớ
  - Điều khiển các module vào/ra

7.1.3 Thanh ghi

- Thanh ghi: Không gian lưu trữ tạm thời phục vụ cho hoạt động ở thời điểm hiện tại của CPU.
- Được coi là cấp bộ nhớ cao nhất trong hệ thống phân cấp
- Số lượng thanh ghi nhiều → tăng hiệu năng của CPU
- Phân thành 2 loại:
  - Thanh ghi hiện thị với người dùng
    - Lập trình được, giúp giảm thiểu các tham chiếu bộ nhớ chính
  - Thanh ghi điều khiển và trạng thái
    - Được CU sử dụng để điều khiển hoạt động của CPU và các chương trình hệ điều hành sử dụng để kiểm soát việc thực thi chương trình.

Thanh ghi hiển thị với người dùng

- Được tham chiếu bằng ngôn ngữ máy và CPU thực hiện
- Phân loại theo chức năng
  - **Thanh ghi đa năng**: có thể chưa địa chỉ hoặc dữ liệu
  - **Thanh ghi dữ liệu**: chứa tạm thời dữ liệu, không được sử dụng trong tính toán địa chỉ toán hạng
  - **Thanh ghi địa chỉ**: quản lý địa chỉ của ngăn nhớ hay cổng vào/ra
  - **Thanh ghi mã điều kiện** (trạng thái)
    - Còn gọi là thanh ghi cờ (flag)
    - Là các bit do phần cứng của bộ xử lý đặt theo kết quả của hoạt động

Thanh ghi đa năng

- Có thể được lập trình viên gán cho nhiều chức năng khác nhau
- Có thể được sử dụng cho dữ liệu hoặc định địa chỉ
- So sánh thanh ghi đa năng và chuyên dụng
  - Đa năng:
    - Tăng lựa chọn cho lập trình viên và linh hoạt
    - Tăng độ phức tạp và kích thước lệnh
  - Chuyên dụng:
    - Lệnh nhỏ hơn, nhanh hơn
    - Ít linh hoạt hơn

Thanh ghi dữ liệu

- Chứa các dữ liệu tạm thời hoặc các kết quả trung gian
- Cần có nhiều thanh ghi dữ liệu
- Các thanh ghi số nguyên 8, 16, 32, 64 bit
- Các thanh ghi số dấu phẩy động

Thanh ghi địa chỉ

- Con trỏ dữ liệu DP (Data Pointer)
- Con trỏ ngăn xếp SP (Stack Pointer)
- Thanh ghi cơ sở và thanh ghi chỉ số (Base Register & Index Register)

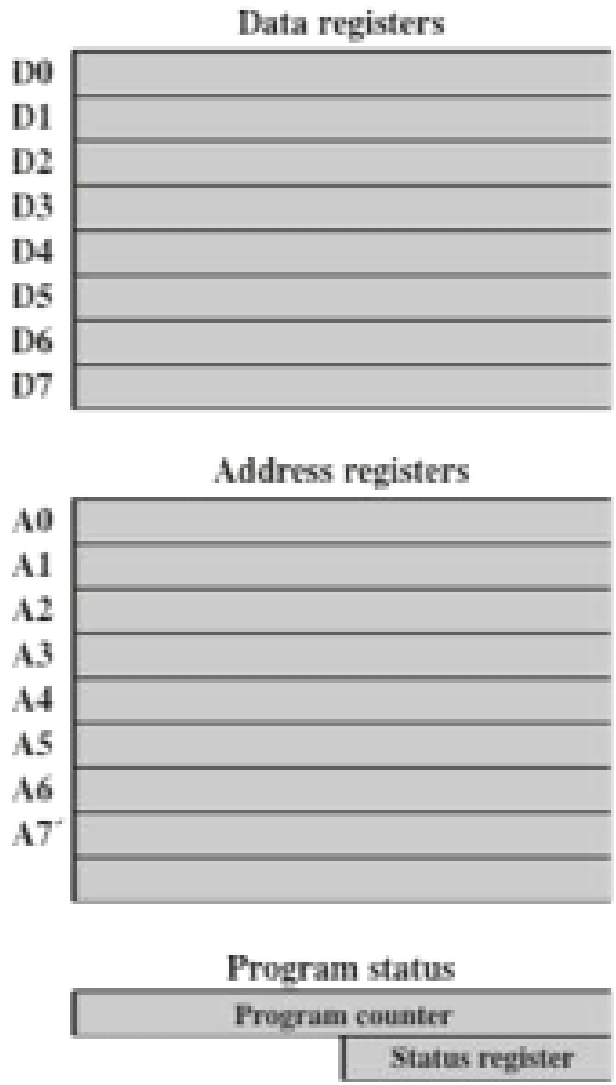
Từ trạng thái chương trình (Program Status Word – PSW)

- Thanh ghi hoặc tập hợp thanh ghi chứa thông tin trạng thái và mã điều kiện.
- Các trường hoặc cờ phổ biến gồm:
  - Cờ zero: được thiết lập lên 1 khi kết quả của phép toán bằng 0
  - Cờ Sign: được thiết lập lên 1 khi kết quả phép toán nhỏ hơn 0
  - Cờ Carry: được thiết lập lên 1 nếu phép toán có nhớ ra ngoài bit cao nhất MSB →Cờ báo tràn với số không dấu.
  - Cờ overflow: được thiết lập lên 1 nếu cộng 2 số nguyên cùng dấu mà kết quả có dấu ngược lại → cờ báo tràn với số có dấu.
  - Cờ Interrup (IF)
    - Nếu IF=1 → CPU ở trạng thái cho phép ngắt
    - Nếu IF=0 → CPU ở trạng thái cấm ngắt

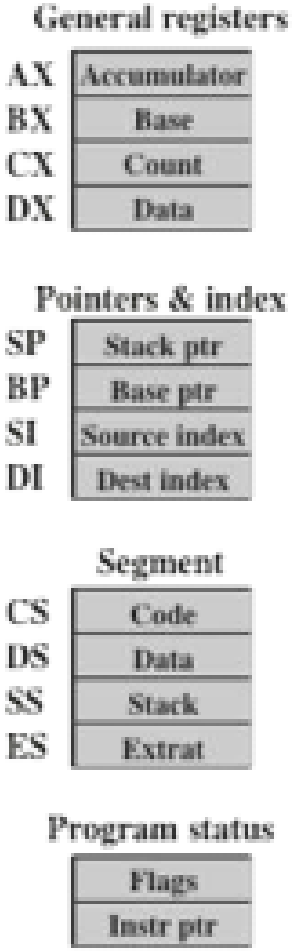
Thanh ghi điều khiển và trạng thái

Bốn thanh ghi cần thiết để thực thi lệnh

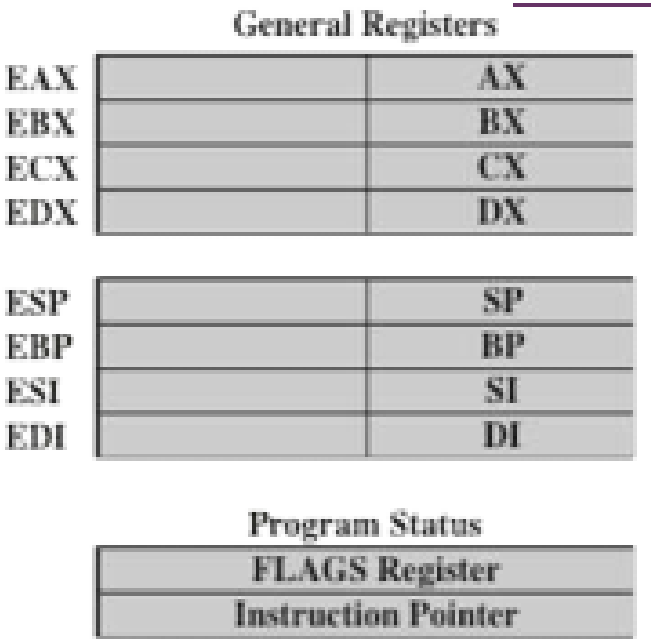
- Bộ đếm chương trình (PC)
  - Chứa địa chỉ của lệnh sắp được truy xuất
- Thanh ghi lệnh (IR)
  - Bao gồm hướng dẫn được truy xuất gần đây nhất
- Thanh ghi địa chỉ bộ nhớ (MAR)
  - Chứa địa chỉ của một vị trí trong bộ nhớ
- Thanh ghi đệm bộ nhớ (MBR)
  - Chứ một từ dữ liệu sắp được ghi vào bộ nhớ hoặc từ vừa được đọc.



(a) MC68000



(b) 8086



(c) 80386 - Pentium 4

Ví dụ tổ chức thanh ghi CPU

7.2. Chu trình lệnh

- Các bước trong một chu trình lệnh
- Lược đồ trạng thái chu trình lệnh đầy đủ
- Chu kỳ gián tiếp
- Luồng dữ liệu

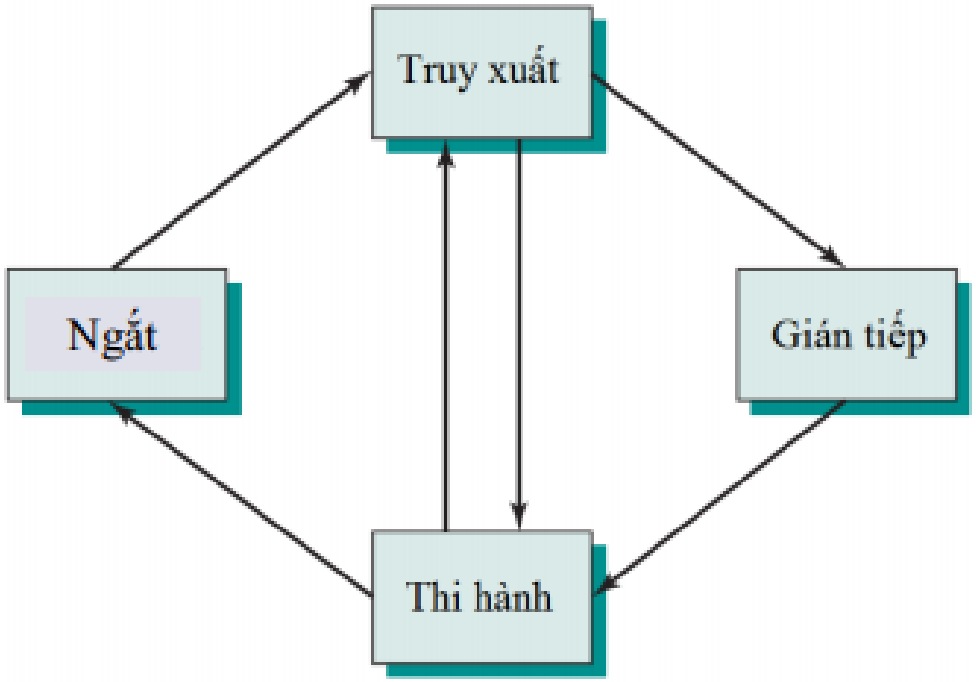
7.2. Chu trình lệnh

Bao gồm các giai đoạn sau:

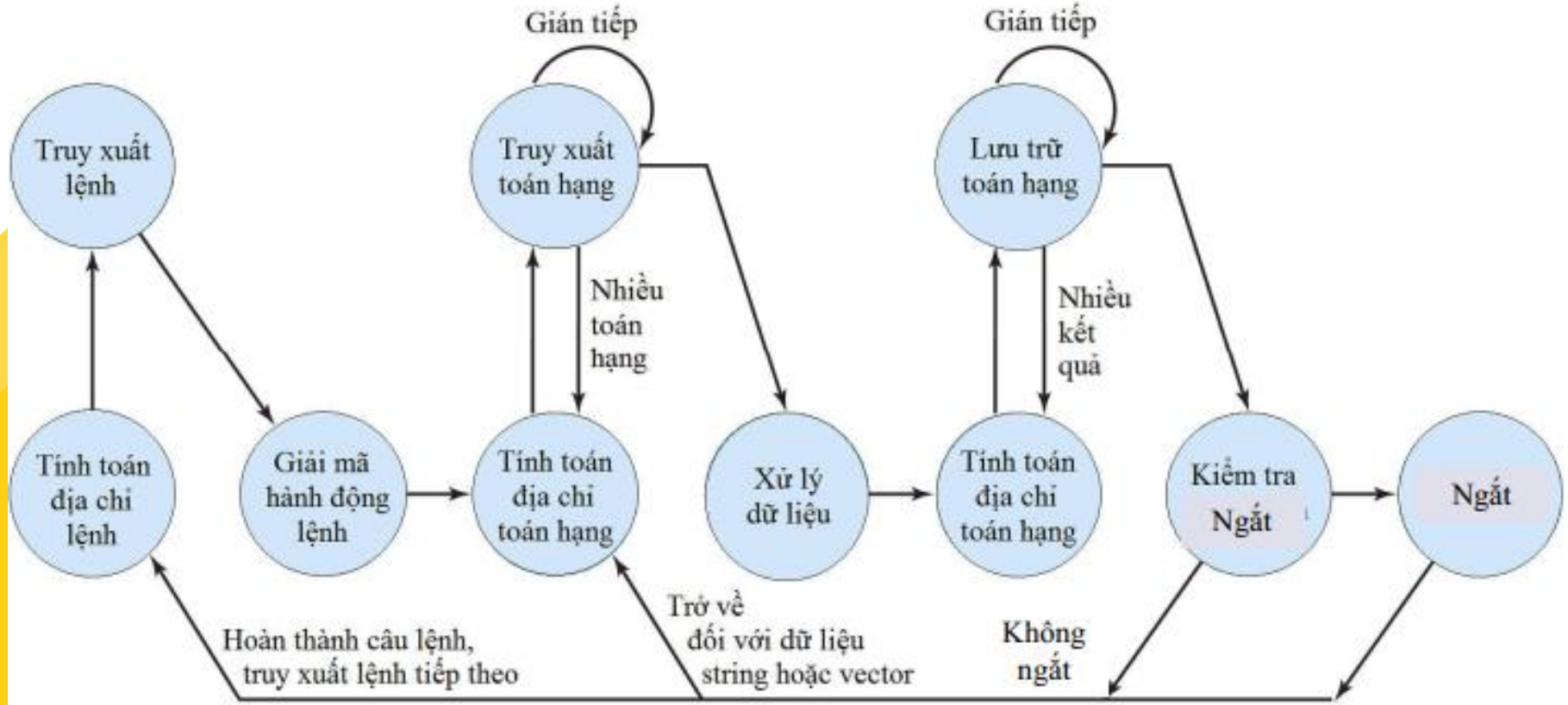
- Truy xuất
  - Đọc lệnh tiếp theo từ bộ nhớ vào bộ xử lý
- Thi hành
  - Giải nghĩa opcode
  - Truy xuất toán hạng
  - Thực hiện lệnh
  - Cất toán hạng
- Ngắt
  - Nếu CPU cho phép ngắt và nhận được tín hiệu ngắt, lưu trạng thái xử lý hiện tại và phục vụ ngắt đó

7.2. Chu trình lệnh

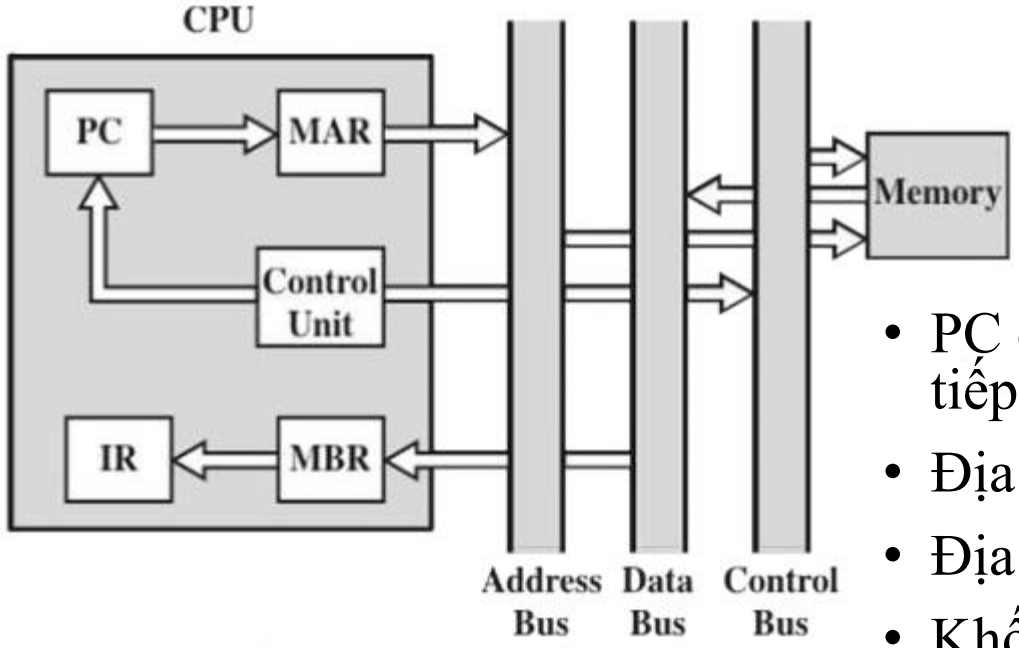
- Có thể cần truy cập bộ nhớ để truy xuất toán hạng
- Địa chỉ gián tiếp thì cần nhiều truy cập bộ nhớ hơn
- Có thể coi việc truy xuất địa chỉ gián tiếp như một giai đoạn khác trong chu kỳ lệnh



Sơ đồ trạng thái chu kỳ lệnh



Luồng dữ liệu , chu kỳ truy xuất



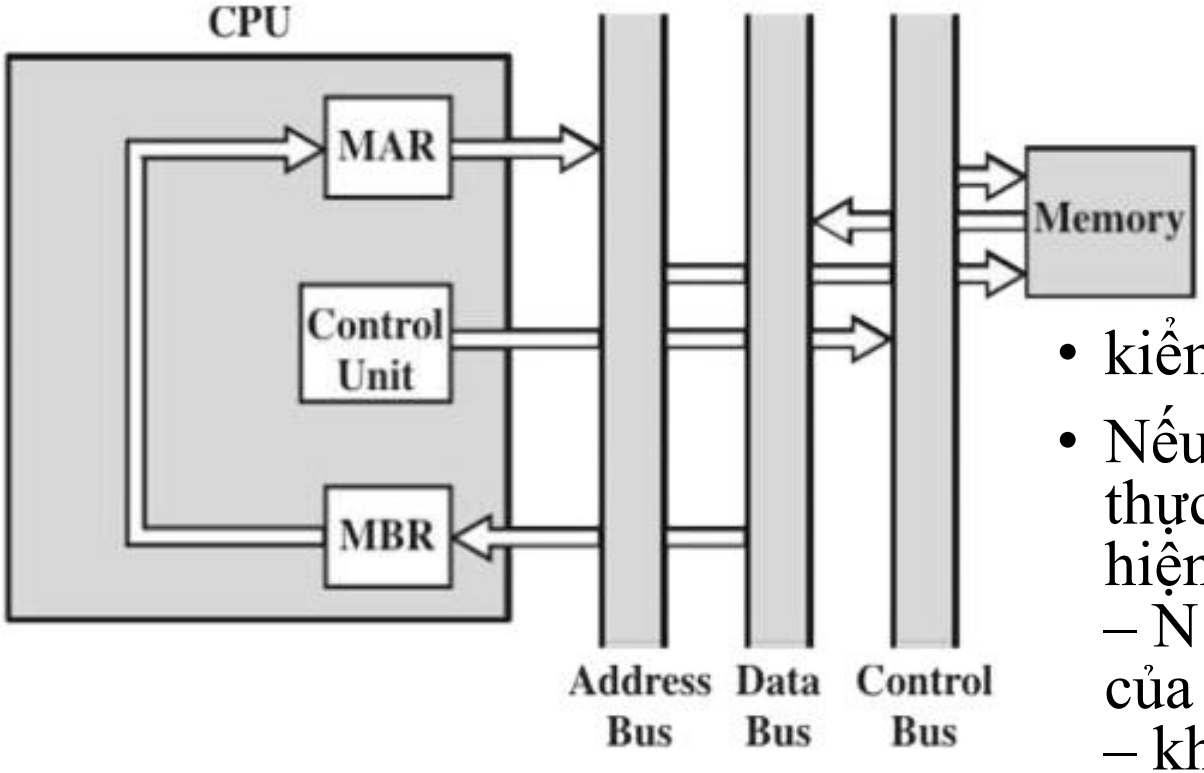
MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

- PC chứa địa chỉ của lệnh tiếp theo
- Địa chỉ chuyển đến MAR
- Địa chỉ đặt lên bus địa chỉ
- Khối điều khiển gửi yêu cầu một lần đọc bộ nhớ
- Kết quả được đặt trên bus dữ liệu, sao chép vào MBR rồi chuyển tới IR
- PC được tăng lên 1

Chu kỳ thi hành

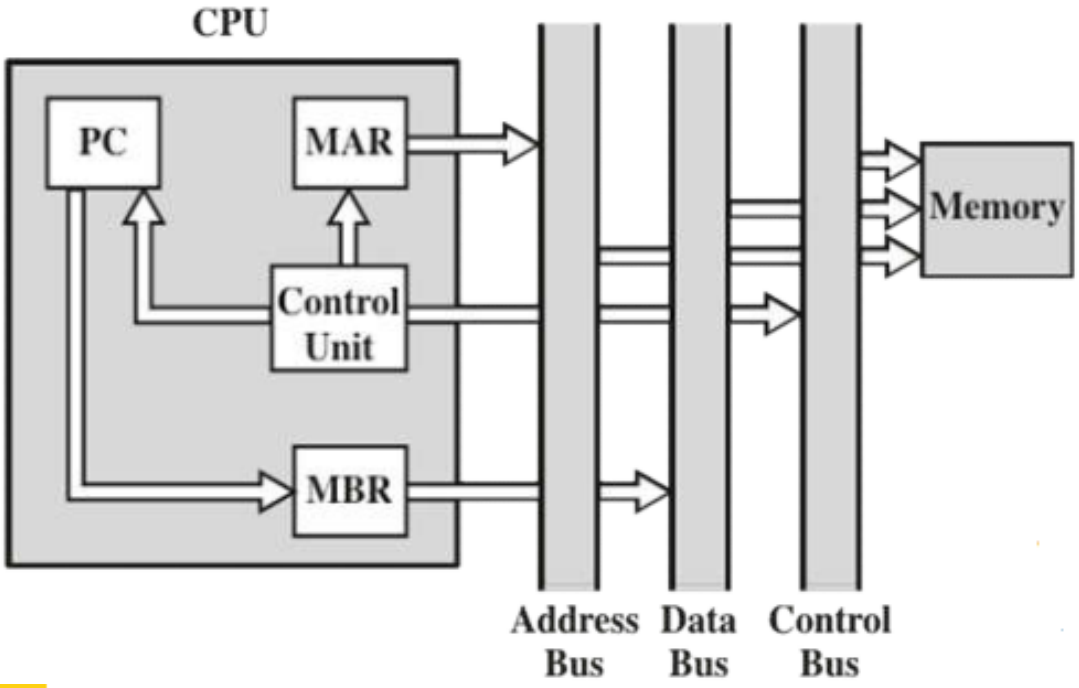
- Giải mã lệnh:
  - Lệnh từ thanh ghi lệnh IR được đưa đến CU
  - CU giải mã lệnh để xác định thao tác phải thực hiện
- Truy xuất dữ liệu: tương tự truy xuất lệnh
- Thi hành lệnh:
  - Có nhiều hình thức, phụ thuộc vào opcode
  - Có thể bao gồm:
  - đọc hoặc ghi bộ nhớ
  - Vào/ra
  - truyền dữ liệu giữa các thanh ghi
  - Hoạt động của ALU

Luồng dữ liệu, chu kỳ gián tiếp



- kiểm tra nội dung của IR
- Nếu địa chỉ gián tiếp thì thực hiện *chu kỳ gián tiếp*
  – N bit ngoài cùng bên phải của MBR chuyển tới MAR
  – khối điều khiển yêu cầu một lần đọc bộ nhớ
  – Chuyển toán hạng mong muốn vào MBR

Luồng dữ liệu, chu kỳ ngắt



- PC hiện tại được lưu lại để có thể tiếp tục hoạt động sau ngắt→ Nội dung của PC chuyển sang MBR.
- Vị trí bộ nhớ đặc biệt (VD: con trỏ ngăn xếp) được nạp vào MAR
- MBR ghi vào bộ nhớ
- Địa chỉ của trình xử lý ngắt được nạp vào PC
- Lệnh tiếp theo (lệnh đầu tiên trong xử lý ngắt) được truy xuất

7.3. Kỹ thuật pipeline lệnh

- Chiến thuật thi hành pipelining
- Ảnh hưởng của các lệnh rẽ nhánh có điều kiện và vô điều kiện
- Các phương pháp xử lý rẽ nhánh
- Hiệu năng
- Pipeline trong Intel và ARM

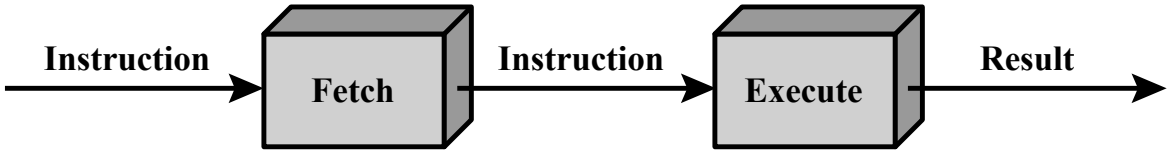
7.3. Kỹ thuật pipeline lệnh

- Tương tự việc sử dụng dây chuyền lắp ráp trong nhà máy sản xuất
  - Các đầu vào mới được tiếp nhận ở một phía trước khi các đầu vào trước đây trở thành đầu ra ở phía còn lại
- Một lệnh có nhiều giai đoạn → có thể sử dụng pipelining
- Chia chu trình lệnh thành các công đoạn và cho phép thực hiện gối lên nhau.

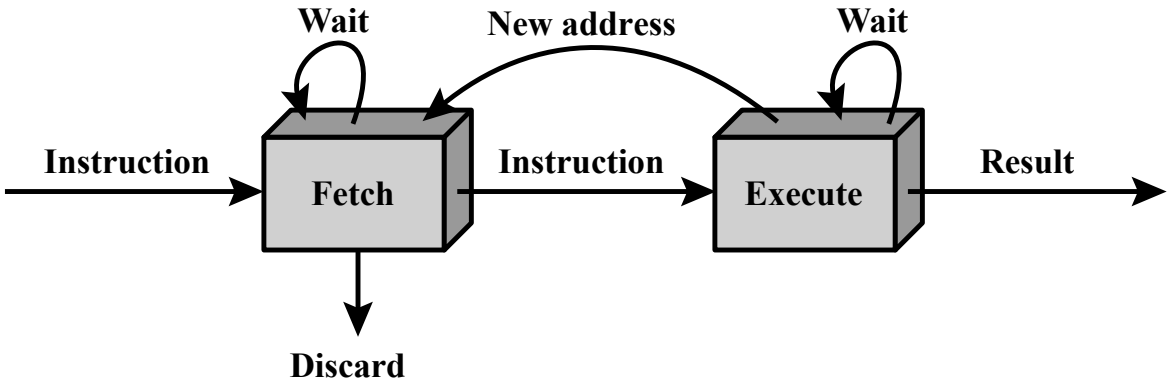
Pipeline hai giai đoạn

- 2 giai đoạn:
  - Truy xuất: Truy cập bộ nhớ chính
  - Thi hành: thường không truy cập bộ nhớ chính
- Có thể truy xuất lệnh tiếp theo trong khi thực thi lệnh hiện tại → truy xuất lệnh trước (prefetch)
- Cần thêm thanh ghi để lưu trữ dữ liệu giữa các giai đoạn
- Tăng tốc việc thực thi lệnh

Pipeline hai giai đoạn



(a) Simplified view

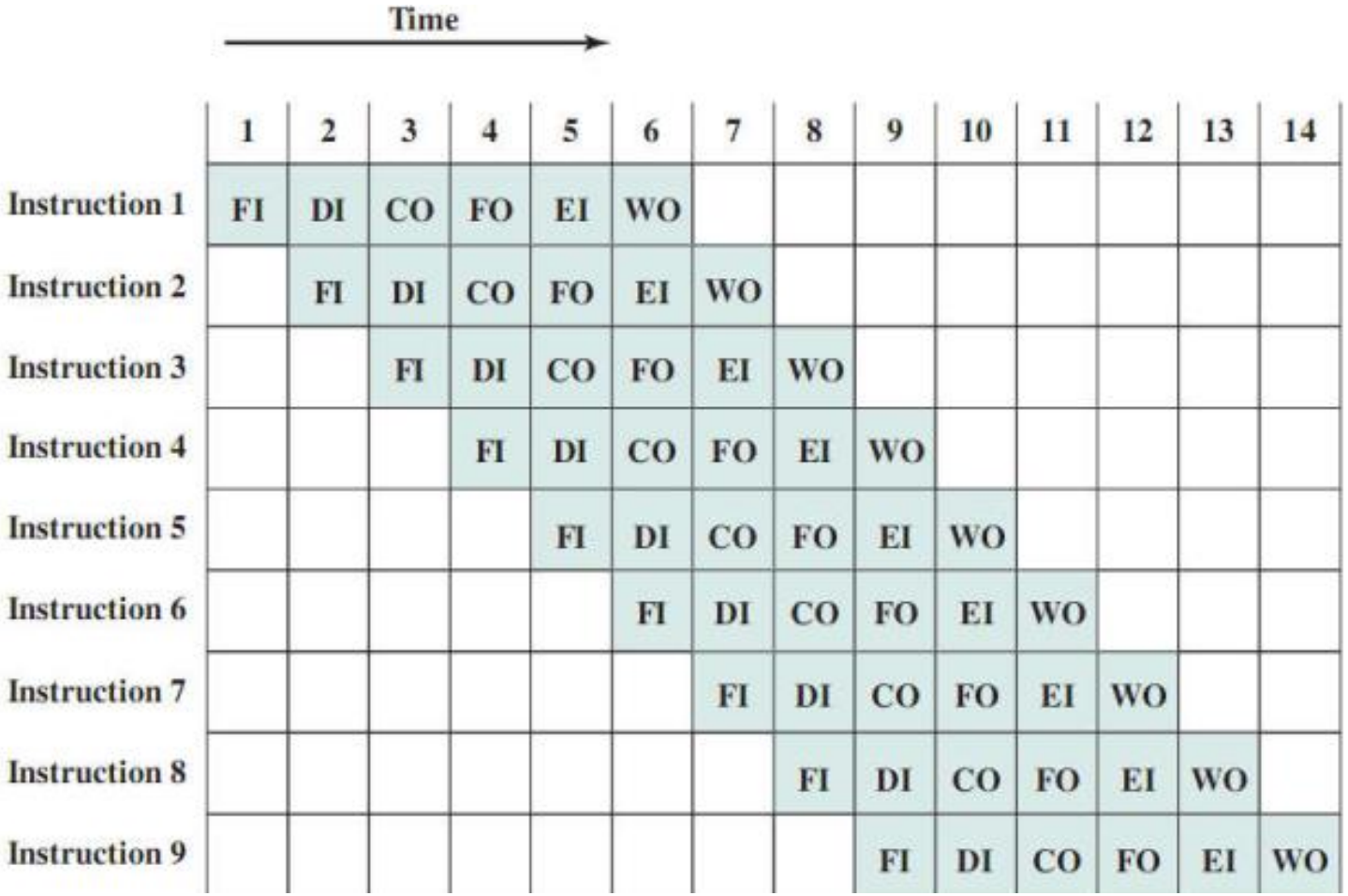


(b) Expanded view

Pipelining

- Giả sử việc thực hiện lệnh gồm 6 giai đoạn:
  - Truy xuất lệnh (FI)
  - Giải mã lệnh (DI)
  - Tính địa chỉ toán hạng (CO)
  - Truy xuất toán hạng (FO)
  - Thi hành lênh (EI)
  - Ghi toán hạng (WO)

Biểu đồ thời gian của hoạt động pipeline lệnh



Ảnh hưởng của các lệnh rẽ nhánh có điều kiện và vô điều kiện

Thời gian →

Trả giá cho rẽ nhánh ←→

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Lệnh 1	FI	DI	CO	FO	EI	WO								
Lệnh 2		FI	DI	CO	FO	EI	WO							
Lệnh 3			FI	DI	CO	FO	EI	WO						
Lệnh 4				FI	DI	CO	FO							
Lệnh 5					FI	DI	CO							
Lệnh 6						FI	DI							
Lệnh 7							FI							
Lệnh 15								FI	DI	CO	FO	EI	WO	
Lệnh 16									FI	DI	CO	FO	EI	WO

Pipeline 6 giai đoạn

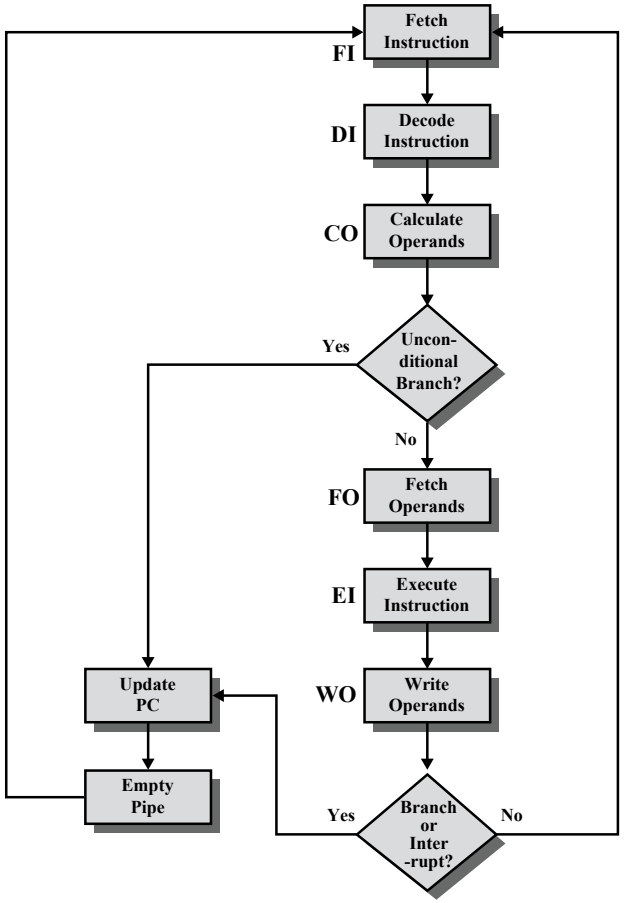


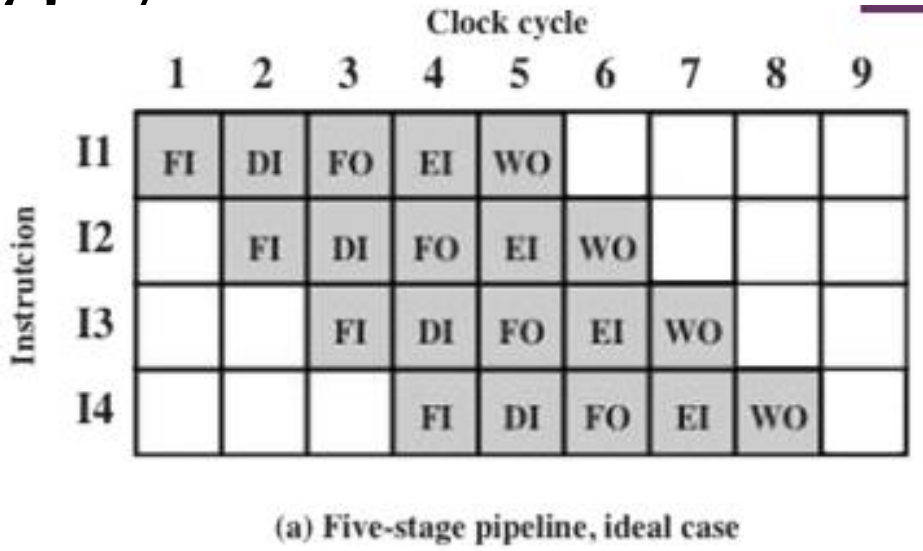
Figure 14.12 Six-Stage Instruction Pipeline

Hazard trong Pipeline

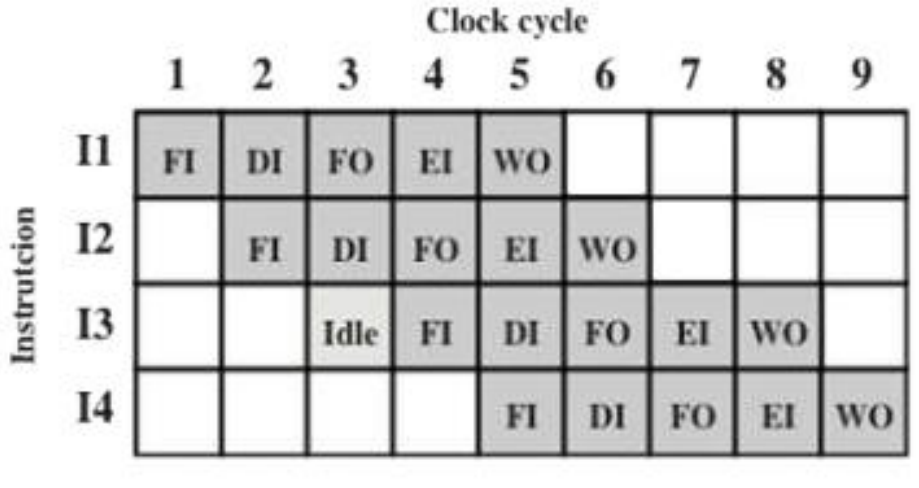
- Xảy ra khi pipeline, hoặc một phần của pipeline, phải đình
  trệ bởi vì điều kiện không cho phép tiếp tục thực hiện
- Còn được gọi là pipeline bubble
  - Có ba loại hazard:
  - Tài nguyên
  - Dữ liệu
  - Điều khiển

Hazard tài nguyên

- Hazard tài nguyên xảy ra khi hai hoặc nhiều lệnh đã ở trong đường ống cần dùng cùng một tài nguyên.
- Các lệnh này phải được thực hiện nối tiếp thay vì song song với một phần của đường ống.
- Còn được gọi là Hazard cấu trúc

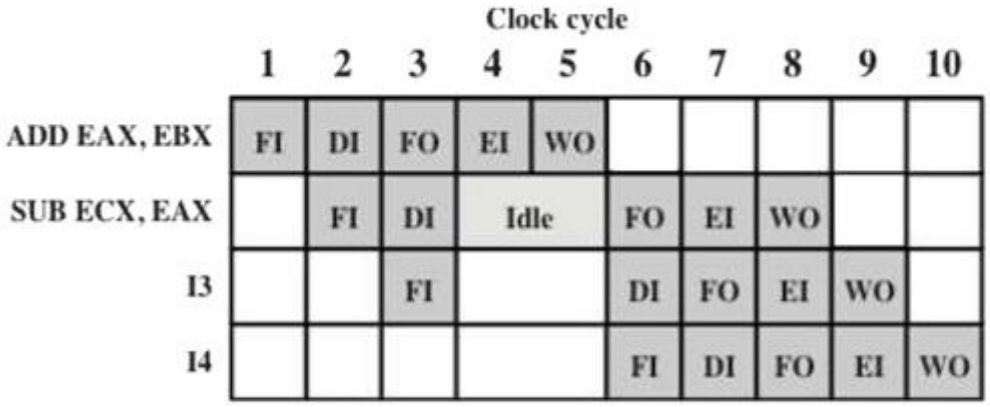


(a) Five-stage pipeline, ideal case



(b) I1 source operand in memory

Hazard dữ liệu



RAW

Hazard

Xảy ra khi có xung đột trong việc truy cập 1 vị trí toán hạng (Lệnh sau sử dụng dữ liệu kết quả của lệnh trước)

Các loại Hazard dữ liệu

- Read after write (RAW), hoặc true dependency
  - Một lệnh sửa đổi một thanh ghi hoặc vị trí bộ nhớ
  - Lệnh tiếp theo đọc dữ liệu trong bộ nhớ hoặc vị trí thanh ghi
  - Hazard xảy ra nếu việc đọc diễn ra trước khi hoạt động ghi hoàn tất
- Write after read (WAR), or antidependency
  - Một lệnh đọc một thanh ghi hoặc vị trí bộ nhớ
  - Lệnh tiếp theo ghi vào vị trí đó
  - Hazard xảy ra nếu thao tác ghi hoàn thành trước khi có thao tác đọc
- Write after write (WAW), or output dependency
  - Hai lệnh cùng ghi vào 1 vị trí
  - Hazard xảy ra nếu các thao tác ghi diễn ra theo thứ tự ngược với trình tự dự định

Hazard điều khiển

- Còn được gọi là Hazard nhánh
- Xảy ra khi đường ống đưa ra quyết định sai về dự báo nhánh
- Đưa lệnh đáng ra sau đó phải được loại bỏ vào đường ống
- Giải pháp với các nhánh có điều kiện:
  - Nhiều luồng
  - Mục tiêu nhánh truy xuất trước
  - Loop buffer
  - Dự báo nhánh
  - Nhánh trễ

Xử lý Hazard rẽ nhánh

- Phương pháp để đối phó với nhánh có điều kiện:
  - Sử dụng nhiều luồng
  - Truy xuất trước mục tiêu rẽ nhánh
  - Bộ đệm vòng lặp
  - Dự báo rẽ nhánh
  - Rẽ nhánh chậm

a. Nhiều luồng

- Một đường ống đơn giản gặp khó khăn khi lệnh rẽ nhánh vì nó phải chọn một trong hai lệnh để truy xuất tiếp theo và có thể chọn sai.
- 1 Cách ép buộc là nhân rộng phần mở đầu của đường ống và cho phép đường ống truy xuất cả hai lệnh, sử dụng hai luồng.
- Có hai vấn đề với cách làm này:
  - Trong nhiều đường ống có trễ tranh chấp để truy cập vào thanh ghi và bộ nhớ.
  - Lệnh nhánh thêm vào có thể đi vào đường ống (một trong hai luồng) trước khi quyết định lệnh nhánh ban đầu được giải quyết.

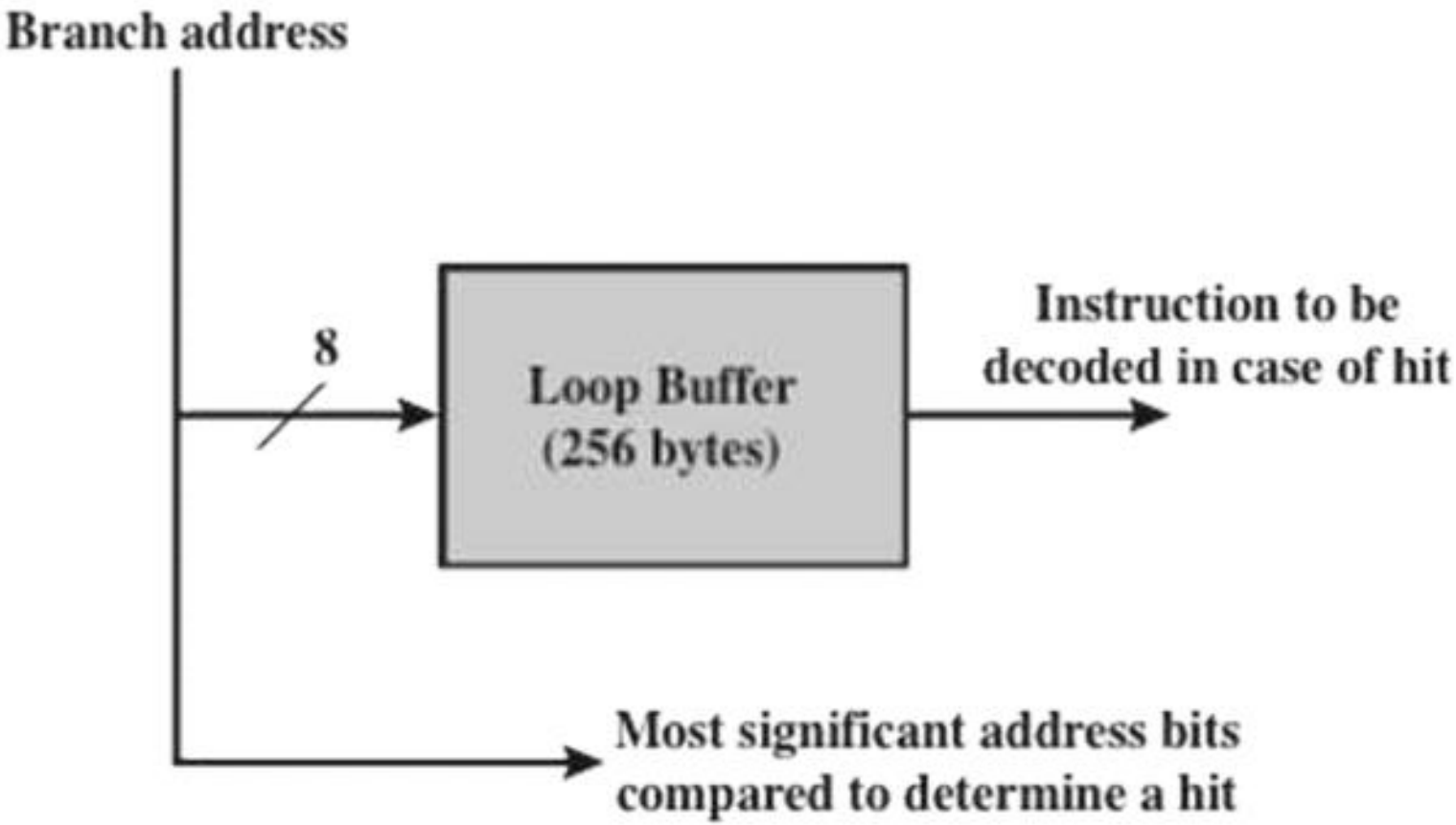
b. Truy xuất trước mục tiêu rẽ nhánh

- Khi nhận ra 1 nhánh điều kiện, mục tiêu của nhánh được truy xuất trước, bổ sung vào lệnh theo sau nhánh.
- Mục tiêu này được lưu lại cho tới khi lệnh rẽ nhánh được thực thi.
- Nếu nhánh này được chọn, mục tiêu đã được truy xuất trước.
- IBM 360/91 sử dụng cách này

c. Bộ đệm vòng lặp

- bộ nhớ nhỏ, tốc độ rất cao được duy trì bởi tầng truy xuất lệnh trong pipeline và chứa n lệnh truy xuất gần đây nhất, theo thứ tự
- Lợi ích:
  - lệnh được truy xuất theo trình tự là có sẵn mà không phải tốn thời gian truy cập bộ nhớ thông thường
  - Nếu có một rẽ nhánh tới mục tiêu nằm trước địa chỉ lệnh rẽ nhánh một vài vị trí, mục tiêu đó đã ở trong bộ đệm
  - Cách này đặc biệt phù hợp để xử lý vòng lặp
- nguyên tắc tương tự với bộ nhớ cache dành riêng cho lệnh
  - Điểm khác biệt:
    - bộ đệm vòng lặp chỉ trữ các lệnh theo trình tự
    - kích thước nhỏ hơn nhiều; do đó giảm chi phí

Loop Buffer


Branch address
8
Loop Buffer
(256 bytes)
Instruction to be
decoded in case of hit
Most significant address bits
compared to determine a hit

d. Dự đoán nhánh

- Nhiều kỹ thuật được sử dụng để dự đoán một nhánh sẽ được chọn hay không:

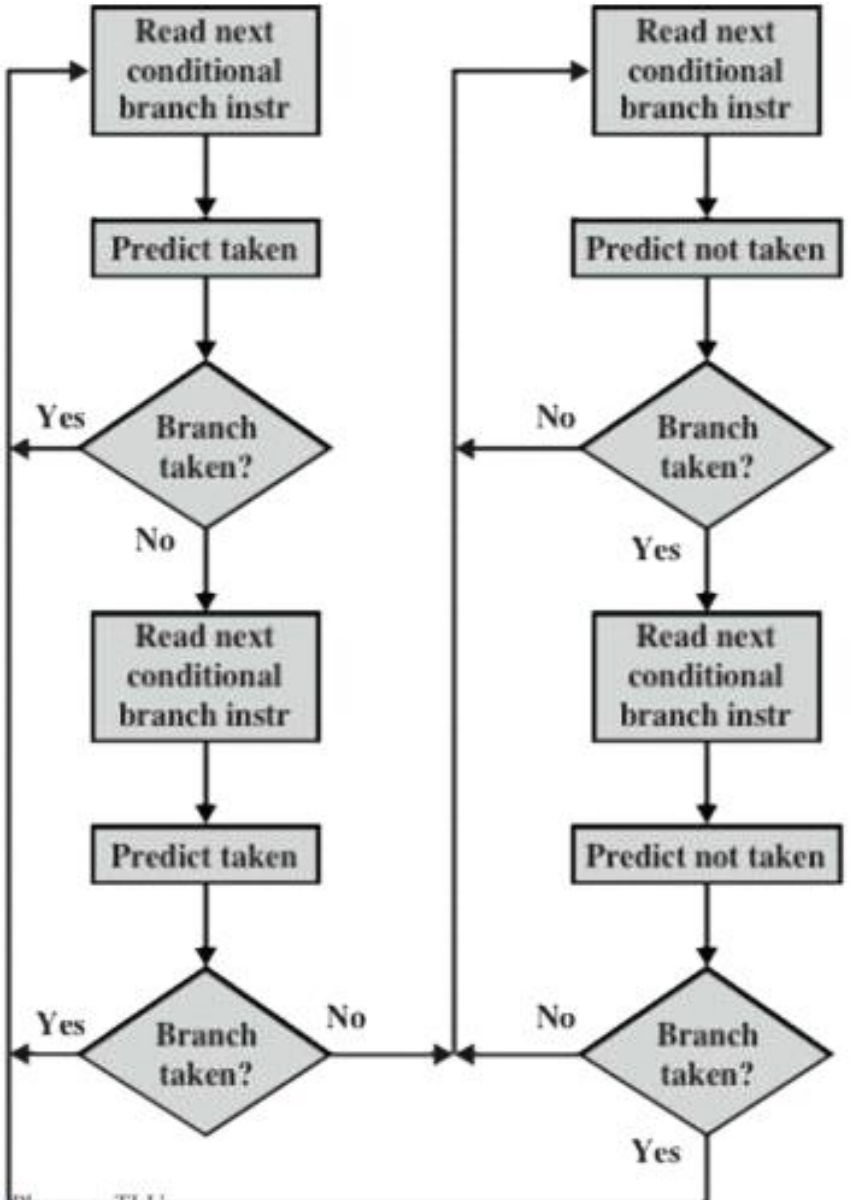
1. Predict never taken
2. Predict always taken
3. Predict by opcode

- ❑ Các phương pháp tĩnh
- ❑ Không phụ thuộc vào lịch sử thực thi tính tới thời điểm có lệnh rẽ nhánh có điều kiện

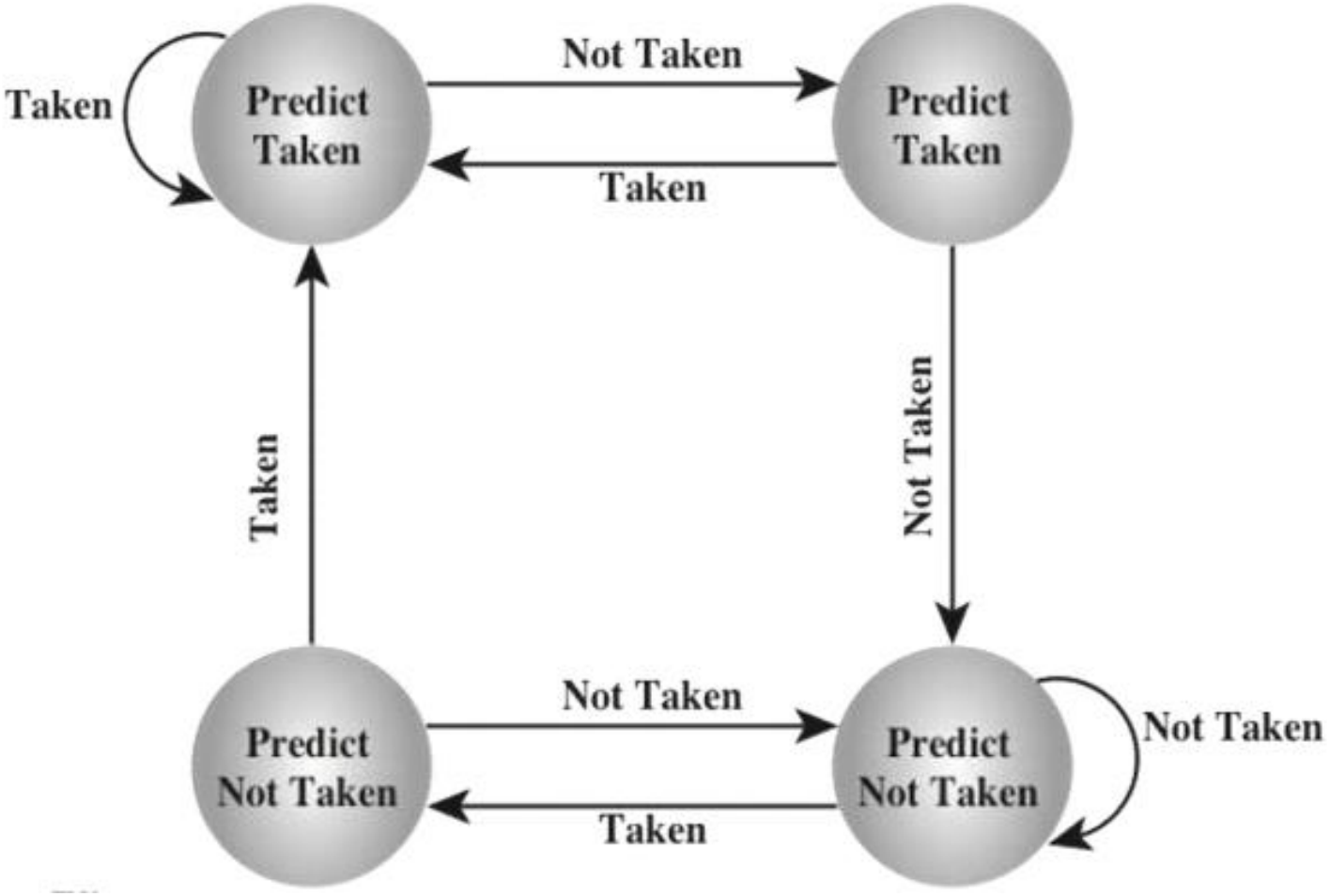
1. Taken/not taken switch
2. Branch history table

- ❑ Các phương pháp động
- ❑ phụ thuộc vào lịch sử thực thi

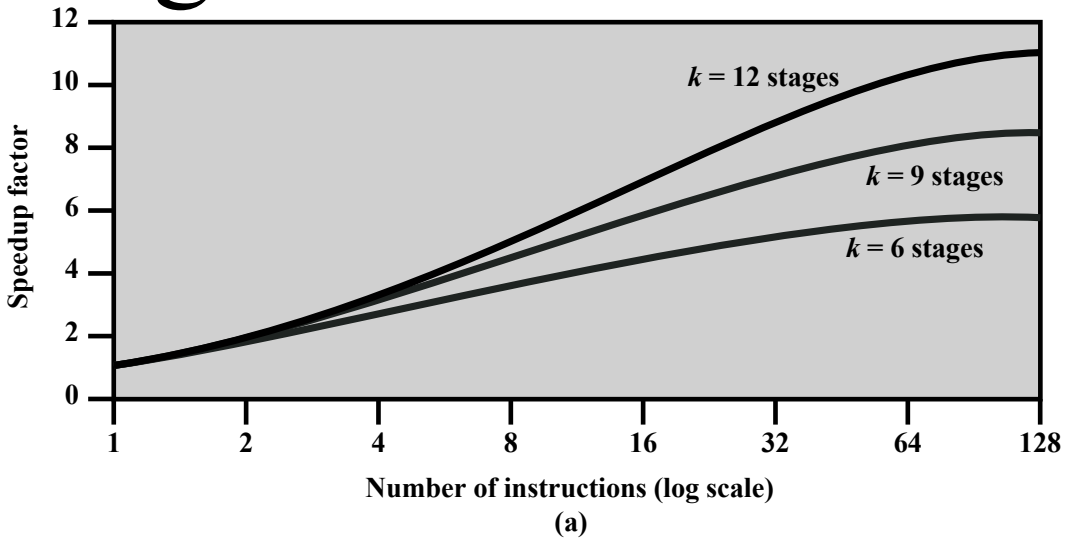
Lược đồ dự đoán nhánh

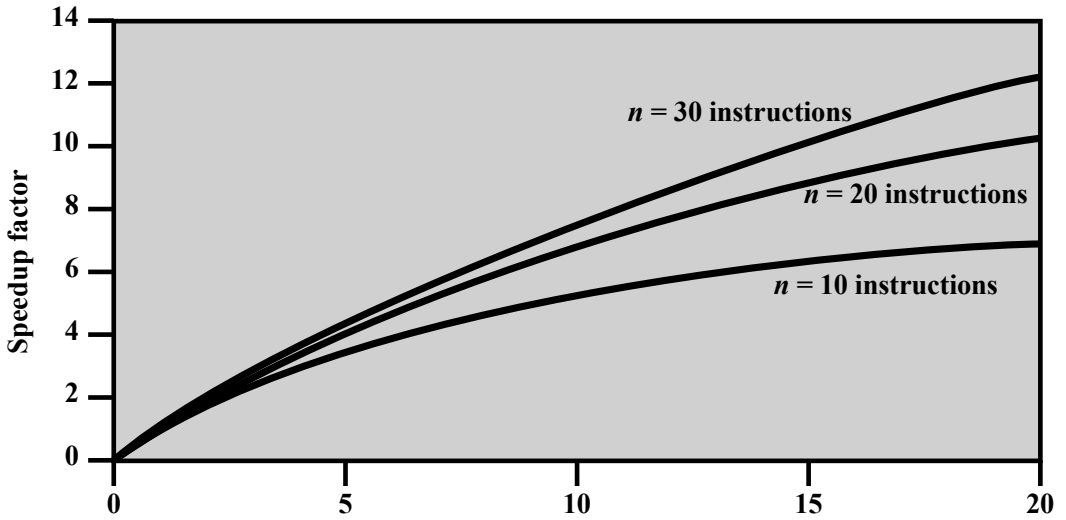


Sơ đồ trạng thái dự đoán nhánh



Hiệu năng


12
10
8
6
4
2
0
Speedup factor
k = 12 stages
k = 9 stages
k = 6 stages
1
2
4
8
16
32
64
128
Number of instructions (log scale)
(a)


14
12
10
8
6
4
2
0
Speedup factor
n = 30 instructions
n = 20 instructions
n = 10 instructions
0
5
10
15
20
Number of stages
(b)

Intel 80486 Pipelining

Fetch

- Nạp vào bộ đệm các lệnh mới ngay khi các lệnh cũ được đưa vào bộ giải mã
- Hoạt động độc lập với các giai đoạn khác

Decode stage 1

- Tất cả các opcode và thông tin chế độ địa chỉ đều được giải mã ở giai đoạn D1
- 3 byte lệnh được chuyển từ bộ đệm tìm nạp trước đến giai đoạn D1
- Sau đó, bộ giải mã D1 có thể chuyển sang giai đoạn D2 để giải mã phần còn lại của lệnh

Decode stage 2

- Mở rộng các opcode thành tín hiệu điều khiển cho ALU
- Đồng thời kiểm soát việc tính toán các chế độ địa chỉ phức tạp hơn

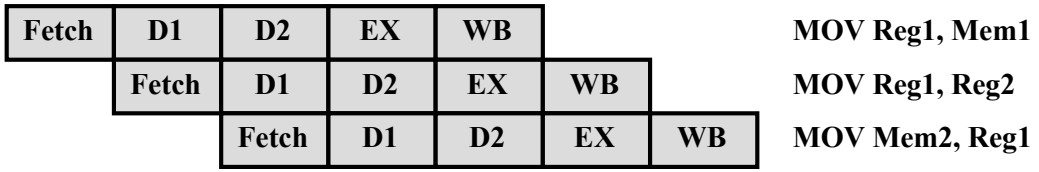
Execute

Giai đoạn bao gồm các hoạt động ALU, truy cập bộ nhớ cache và cập nhật thanh ghi

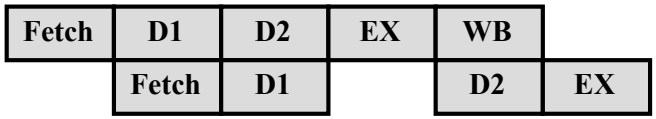
Write back

Cập nhật thanh ghi và trạng thái các cờ đã thay đổi trong lần thực thi trước

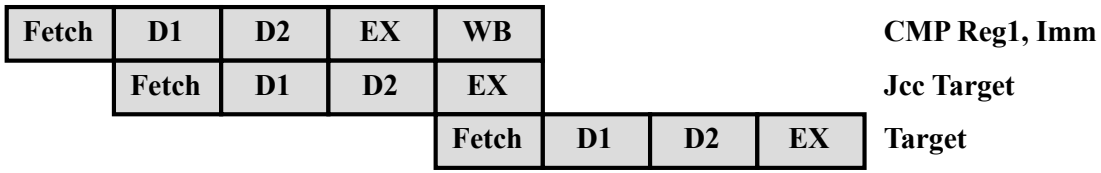
Ví dụ Pipeline 80486



(a) No Data Load Delay in the Pipeline



(b) Pointer Load Delay



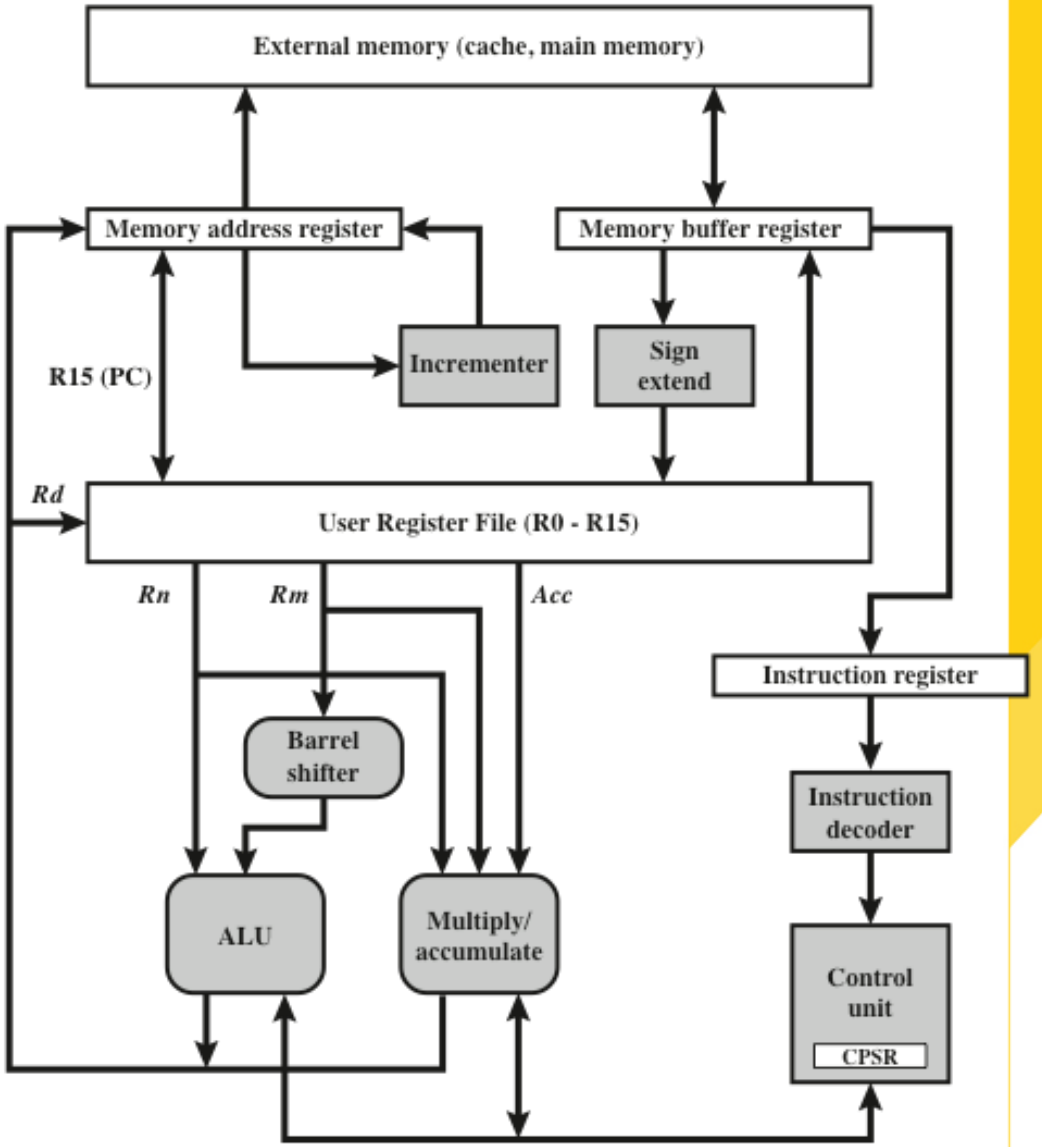
(c) Branch Instruction Timing

The ARM Processor

ARM là hệ thống RISC đầu tiên với các đặc tính sau:

- Gồm mảng các thanh ghi đồng dạng
- Mô hình lệnh load/store dư liệu chỉ thực hiện trên các toán hạng tại thanh ghi, không thực hiện trực tiếp trên bộ nhớ.
- Câu lệnh 32 bit cho tập lệnh tiêu chuẩn và 16 bit cho tập lệnh Thumb
- Tách riêng khối ALU và khối dịch
- Một số mode địa chỉ sẽ có địa chỉ trong lệnh load/store lấy từ thanh ghi hoặc các trường của câu lệnh
- Chế độ địa chỉ tự tăng, tự giảm dùng để cải thiện hoạt động của vòng lặp
- Thực thi có điều khiện của câu lệnh tối thiểu yêu cầu của câu lệnh rẻ nhánh có điều kiện, cải thiện được hiệu suất pipeline

Tổ chức ARM đơn giản



6.4 CISC & RISC

Khái niệm

- CISC → Complex Instruction Set Computer
  - Máy tính với tập lệnh phức tạp
  - Các bộ xử lý truyền thống: Intel x86, Motorola 680x0
- RISC → Reduced Instruction Set Computer
  - Máy tính với tập lệnh thu gọn
  - SunSparc, Power PC, MIPS, ARM,…

6.4 CISC & RISC

- Đặc điểm của CISC và RISC

	Complex Instruction Set (CISC) Computer			Reduced Instruction Set (RISC) Computer	
Characteristic	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000
Year developed	1973	1978	1989	1987	1991
Number of instructions	208	303	235	69	94
Instruction size (bytes)	2–6	2–57	1–11	4	4
Addressing modes	4	22	11	1	1
Number of general-purpose registers	16	16	8	40–520	32
Control memory size (Kbits)	420	480	246	—	—
Cache size (KBytes)	64	64	8	32	128

6.4 CISC & RISC

- Pipeline trong các CPU kiểu RISC

Load	rA ← M	I	E	D									
Load	rB ← M				I	E	D						
Add	rC ← rA + rB							I	E				
Store	M ← rC									I	E	D	
Branch	X											I	E

(a) Sequential execution

Load	rA ← M	I	E	D							
Load	rB ← M		I		E	D					
Add	rC ← rA + rB				I		E				
Store	M ← rC						I	E	D		
Branch	X							I		E	
NOOP										I	E

(b) Two-stage pipelined timing

Load	rA ← M	I	E	D				
Load	rB ← M		I	E	D			
NOOP				I	E			
Add	rC ← rA + rB				I	E		
Store	M ← rC					I	E	D
Branch	X						I	E
NOOP								I	E

(c) Three-stage pipelined timing

Load	rA ← M	I	E_1	E_2	D						
Load	rB ← M		I	E_1	E_2	D					
NOOP				I	E_1	E_2					
NOOP					I	E_1	E_2				
Add	rC ← rA + rB					I	E_1	E_2			
Store	M ← rC						I	E_1	E_2	D	
Branch	X							I	E_1	E_2	
NOOP									I	E_1	E_2
NOOP										I	E_1	E_2

(d) Four-stage pipelined timing

6.4 CISC & RISC

Tối ưu hóa lệnh trong các CPU kiểu RISC

- Delayed branch
- Delayed Load
- Loop Unrolling

Normal And Delayed Branch

Address	Normal Branch		Delayed Branch		Optimized Delayed Branch	
100	LOAD	X, rA	LOAD	X, rA	LOAD	X, rA
101	ADD	1, rA	ADD	1, rA	JUMP	105
102	JUMP	105	JUMP	106	ADD	1, rA
103	ADD	rA, rB	NOOP		ADD	rA, rB
104	SUB	rC, rB	ADD	rA, rB	SUB	rC, rB
105	STORE	rA, Z	SUB	rC, rB	STORE	rA, Z
106			STORE	rA, Z		

Delayed Branch

Time

	1	2	3	4	5	6	7	8
100 LOAD X, rA	I	E	D					
101 ADD 1, rA		I		E				
102 JUMP 105				I	E			
103 ADD rA, rB					I	E		
105 STORE rA, Z						I	E	D

(a) Traditional Pipeline

	1	2	3	4	5	6	7	8
100 LOAD X, rA	I	E	D					
101 ADD 1, rA		I		E				
102 JUMP 106				I	E			
103 NOOP					I	E		
106 STORE rA, Z						I	E	D

(b) RISC Pipeline with Inserted NOOP

	1	2	3	4	5	6
100 LOAD X, Ar	I	E	D			
101 JUMP 105		I	E			
102 ADD 1, rA			I	E		
105 STORE rA, Z				I	E	D

(c) Reversed Instructions

Loop unrolling

```
do i=2, n-1
     a[i] = a[i] + a[i-1] * a[i+1]
end do
```

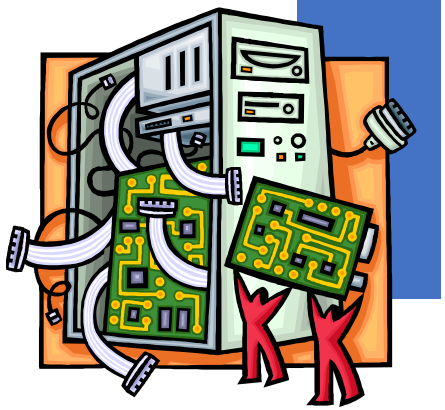
(a) original loop

```
do i=2, n-2, 2
     a[i] = a[i] + a[i-1] * a[i+i]
     a[i+1] = a[i+1] + a[i] * a[i+2]
end do

if (mod(n-2,2) = i) then
     a[n-1] = a[n-1] + a[n-2] * a[n]
end if
```

(b) loop unrolled twice

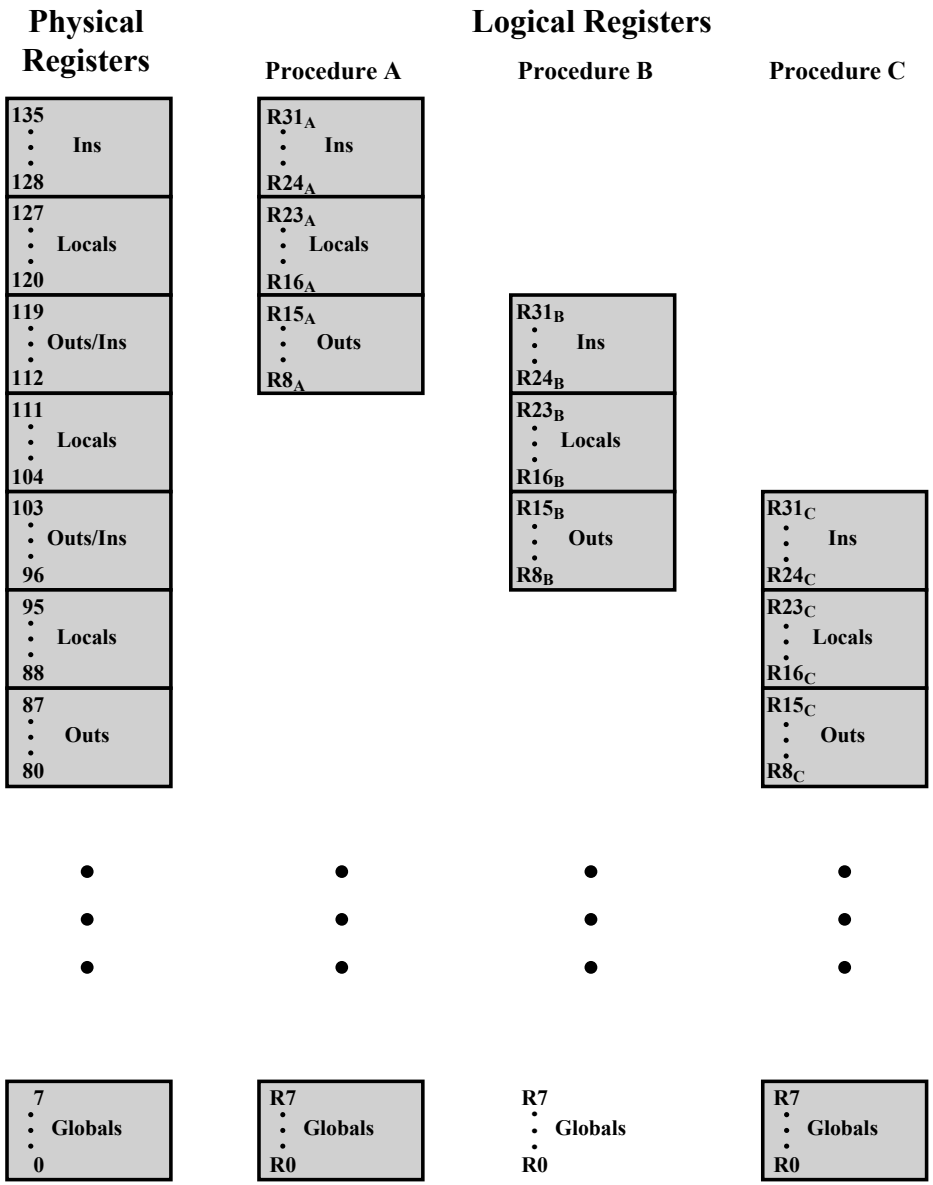
SPARC -Scalable Processor Architecture



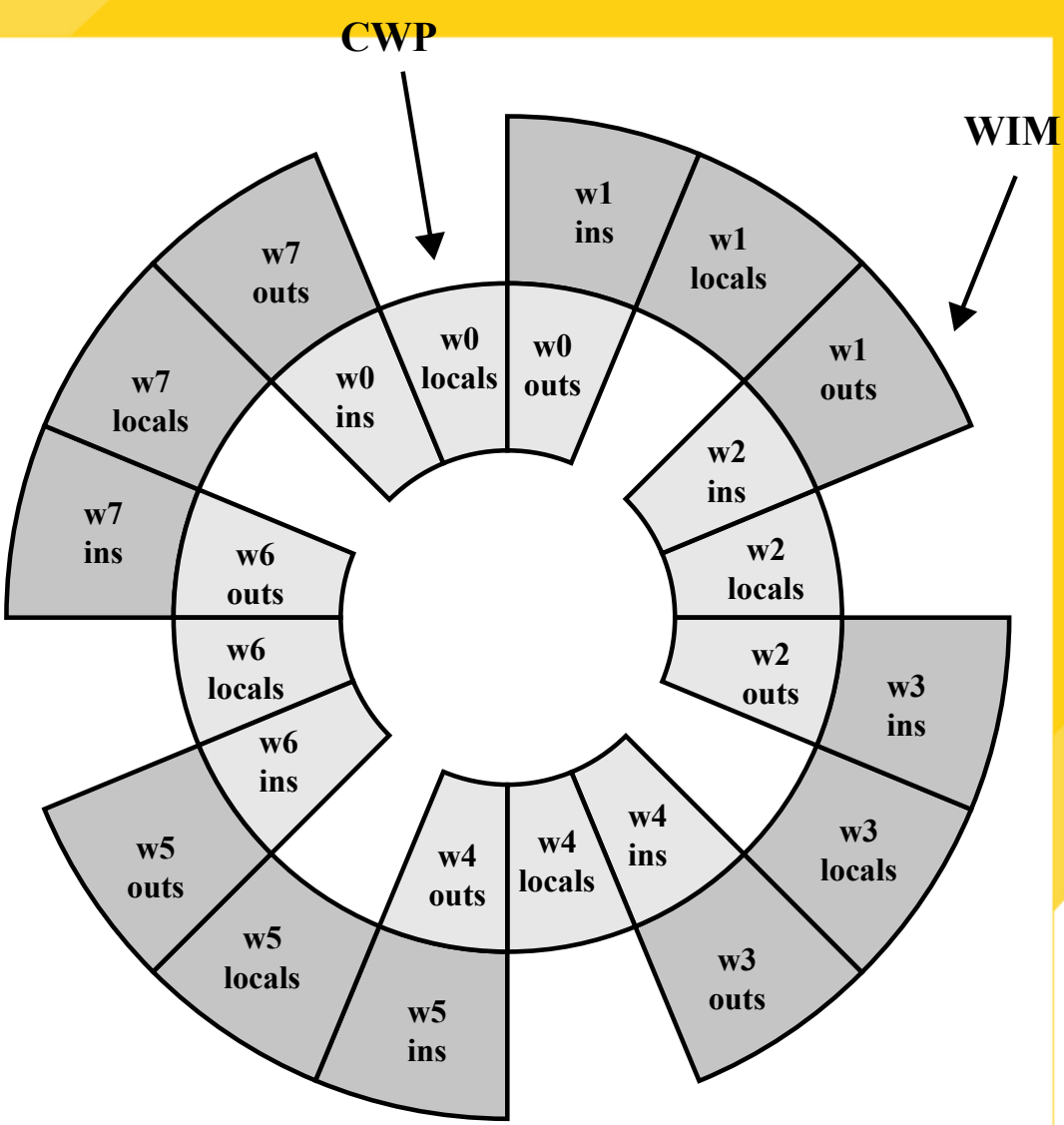
- Được xây dựng bởi Sun Microsystems
- Sun cấp phép cho các nhà sản xuất khác để sản xuất máy tính tương thích với SPARC
- Lấy cảm hứng từ Máy Berkeley RISC 1, và tổ chức thanh ghi và tập lệnh xây dựng dựa trên mô hình của Berkeley RISC

© 2016 Pearson Education, Inc., Hoboken, NJ.
All rights reserved.

Sparc Register window Layout with three procedures



8 register windows forming a circular Stack in SPARC



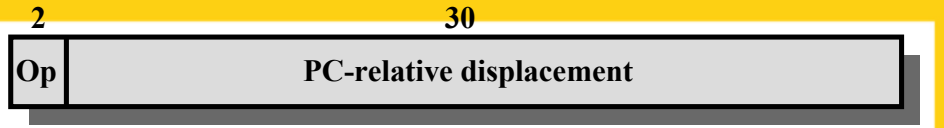
Synthesizing Other Addressing Modes with SPARC Addressing Modes

Instruction Type	Addressing Mode	Algorithm	SPARC Equivalent
Register-to-register	Immediate	operand = A	S2
Load, store	Direct	EA = A	R_0 + S2
Register-to-register	Register	EA = R	R_{S1}, R_{S2}
Load, store	Register Indirect	EA = (R)	R_{S1} + 0
Load, store	Displacement	EA = (R) + A	R_{S1} + S2

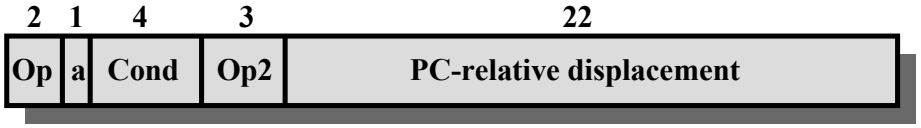
S2 = either a register operand or a 13-bit immediate operand

SPARC Instruction Formats

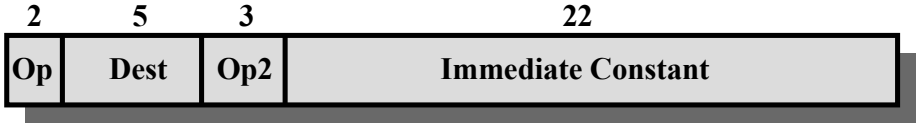
Call Format



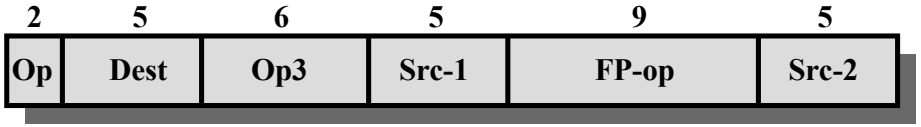
Branch Format



SETHI Format



Floating-Point Format



General Formats