Kiến trúc máy tính

CPU

NGUYỄN Ngọc Hoá

Bộ môn Hệ thống thông tin, Khoa CNTT

Trường Đại học Công nghệ,

Đại học Quốc gia Hà Nội

Hoa.Nguyen@vnu.edu.vn

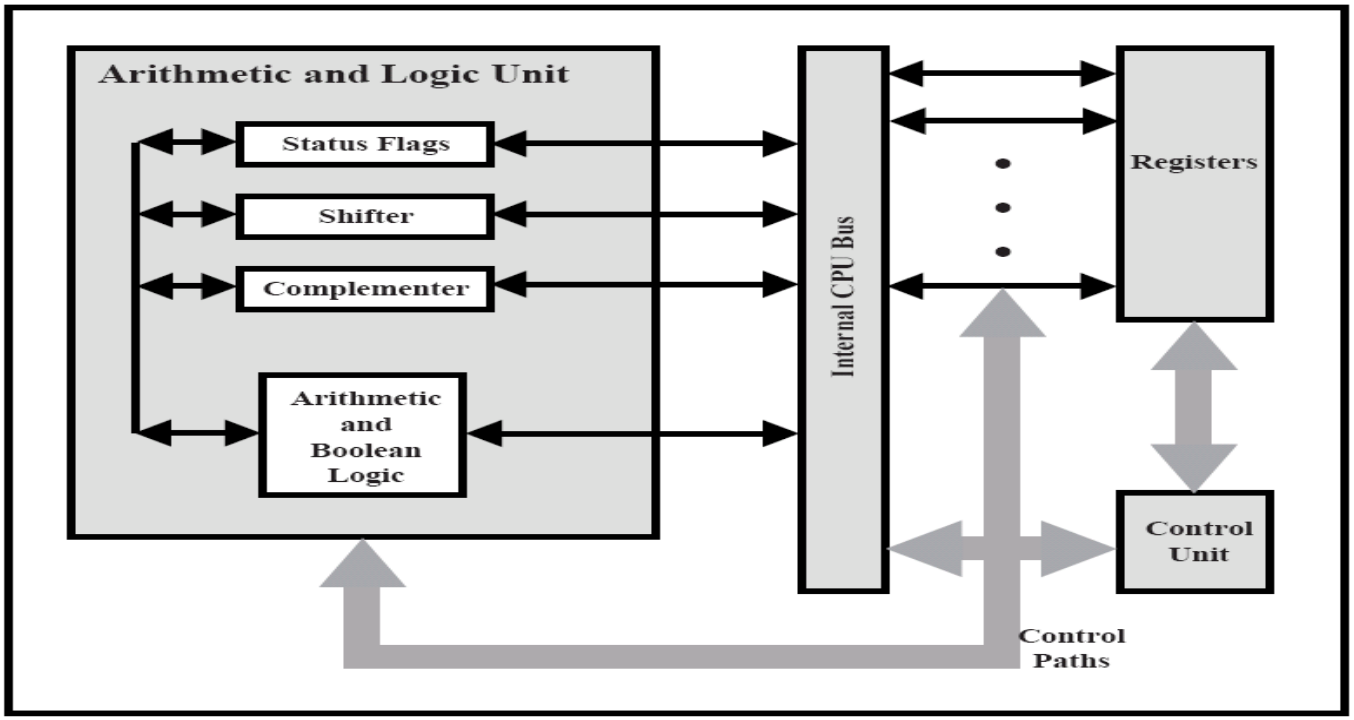
Tổ chức và chức năng của CPU

- Cấu trúc CPU
- Pipeline
- CISC & RISC
- Superscalar, VLIW

Tham khảo chương 12, 13 của [1]

1. Cấu trúc của CPU

- CPU đảm nhiệm
  - Tải lệnh
  - Dịch lệnh
  - Tải dữ liệu
  - Xử lý dữ liệu
  - Lưu dữ liệu
- Đơn vị tính toán (ALU, FPU)
- Đơn vị điều khiển
- Registers (data, address, instruction, control)
- Internal bus



Registers

- Bộ nhớ trong của CPU
  - registers được sử dụng trong các chương trình (user visible regs)
  - registers điều khiển và thể hiện trạng thái
    - được sử dụng bởi CPU
    - được sử dụng bởi OS
- Chú ý
  - Ít thanh ghi ➔ tham chiếu MM nhiều hơn
  - Quá nhiều registers cũng không làm giảm nhiều tham chiếu MM, giảm hiệu năng CPU
  - Đủ rộng để chứa được trường địa chỉ
  - Đủ rộng để chứa từ nhớ
  - Có thể ghép nhiều registers tạo word lớn hơn

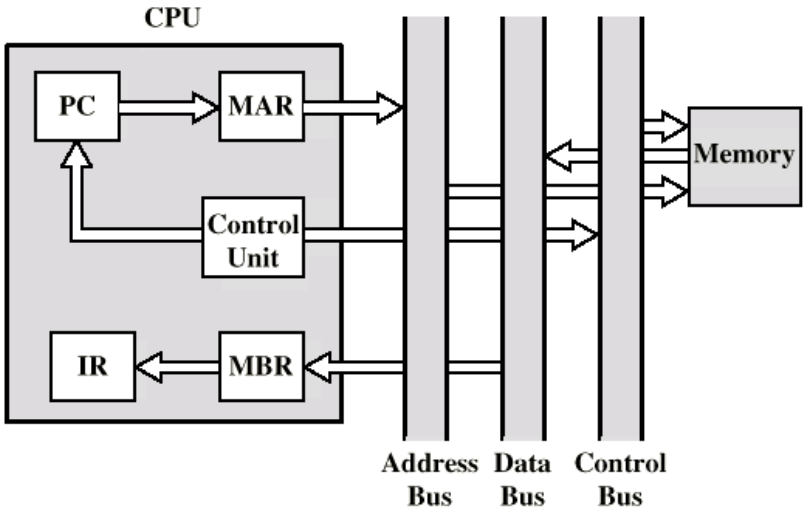
User Visible Registers

- Đa dụng - General Purpose
- Dữ liệu - Data
- Địa chỉ - Address: thường được sử dụng trong các mode đánh địa chỉ
  - Segment based address (e.g., pentium)
  - index
  - pointer to memory stack
- Cờ nhớ - Condition codes (flags) :
  - dãy các bits độc lập với nhau
  - chương trình không thể thay đổi giá trị, chỉ có thể được đọc

Control & Status Registers

Trao đổi dữ liệu với bộ nhớ chính

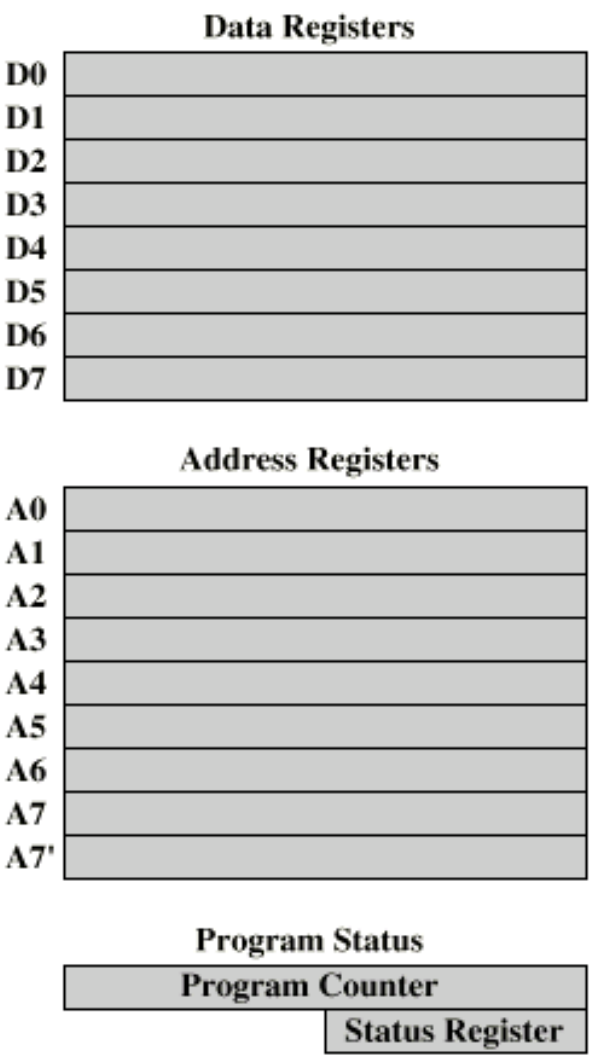
- Program Counter (PC) : địa chỉ của lệnh thi hành kế tiếp
- Instruction Decoding Register (IR) : lệnh đang thi hành
- Memory Address Register (MAR) : địa chỉ bộ nhớ, kết nối trực tiếp tới bus địa chỉ
- Memory Buffer Register (MBR) : từ dữ liệu, kết nối trực tiếp đến bus dữ liệu
- Những registers trung gian



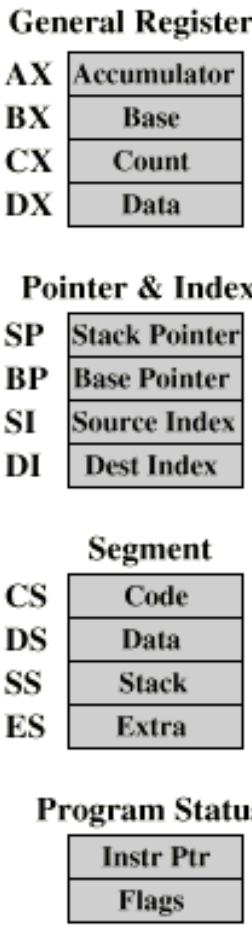
Program Status Word

- Tập các bits thể hiện trạng thái phép tính vừa thực hiện trong CPU - Condition Codes
  - Sign of last result
  - Zero
  - Carry
  - Equal
  - Overflow
  - Interrupt enable/disable
  - Supervisor
- Supervisor/Kernel mode
  - Cho phép thi hành những lệnh đặc quyền (system calls)
  - Được sử dụng bởi hệ điều hành và người sử dụng/programers không được cấp phép sử dụng chế độ này

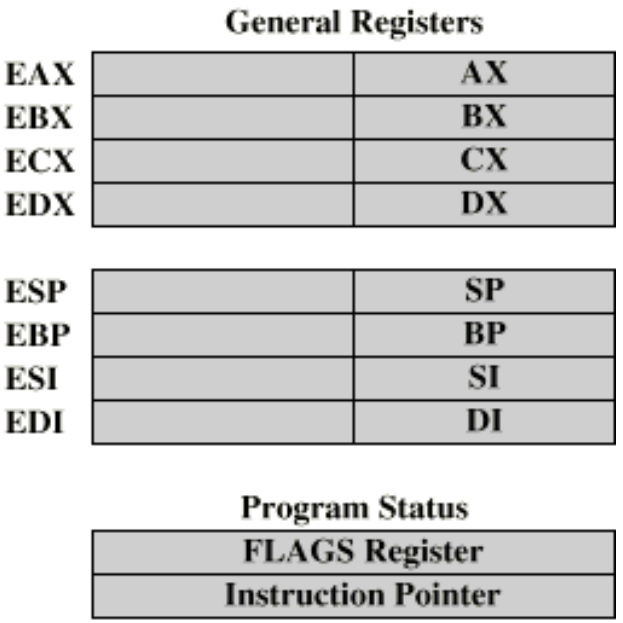
Ví dụ tổ chức thanh ghi



(a) MC68000

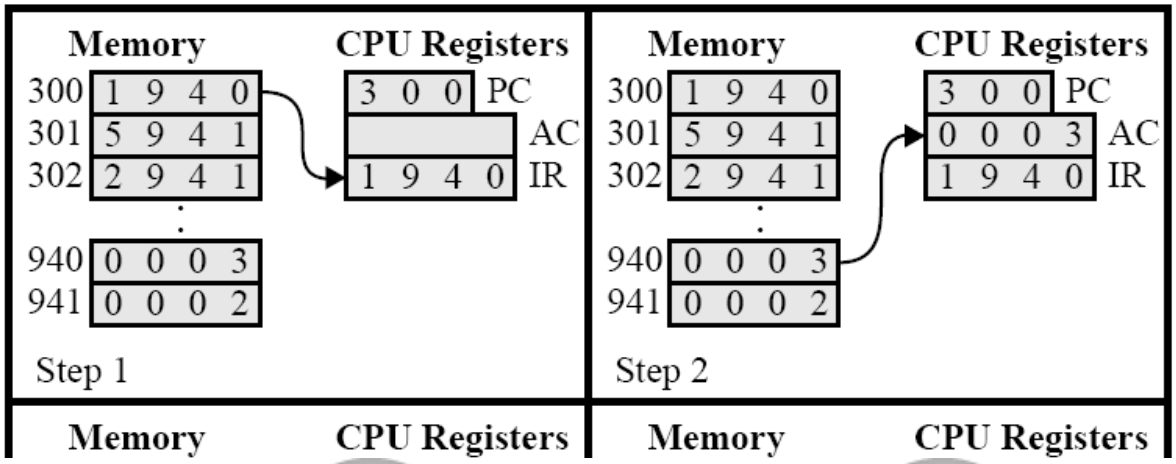


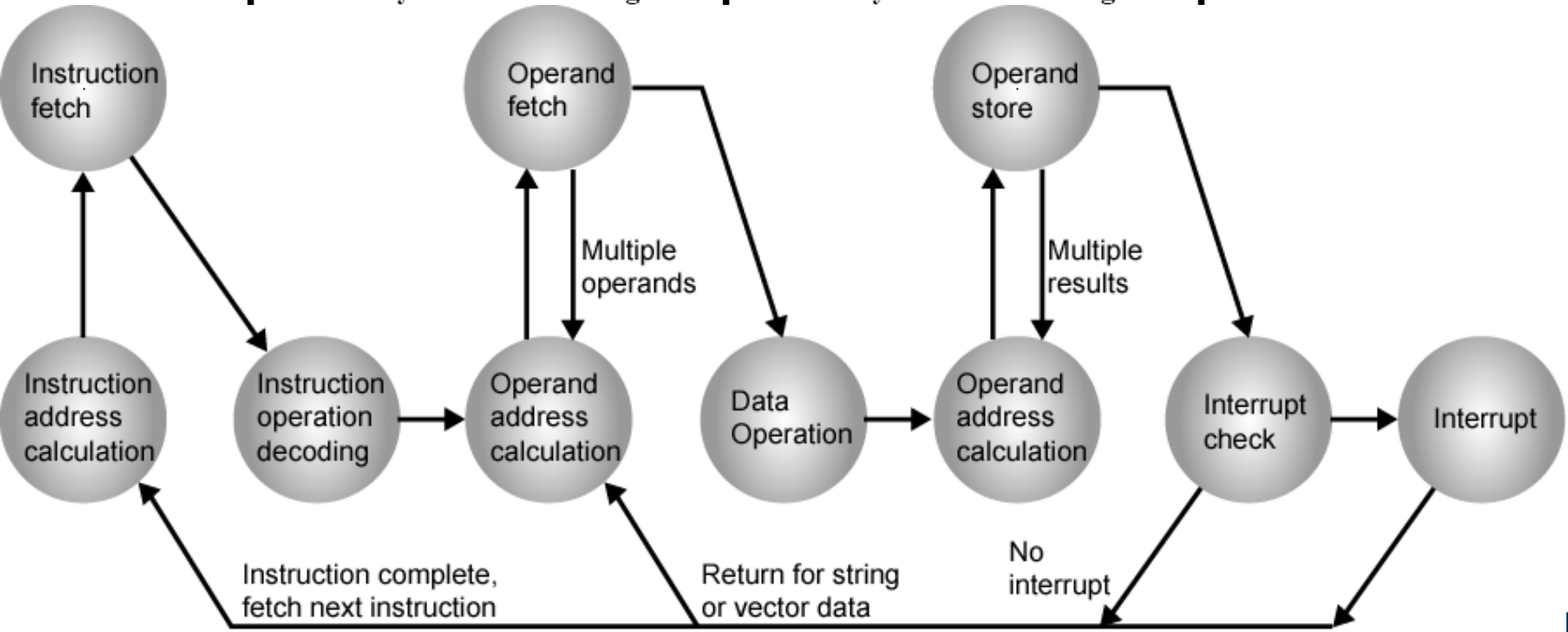
(b) 8086



(c) 80386 - Pentium II

Thi hành một chương trình

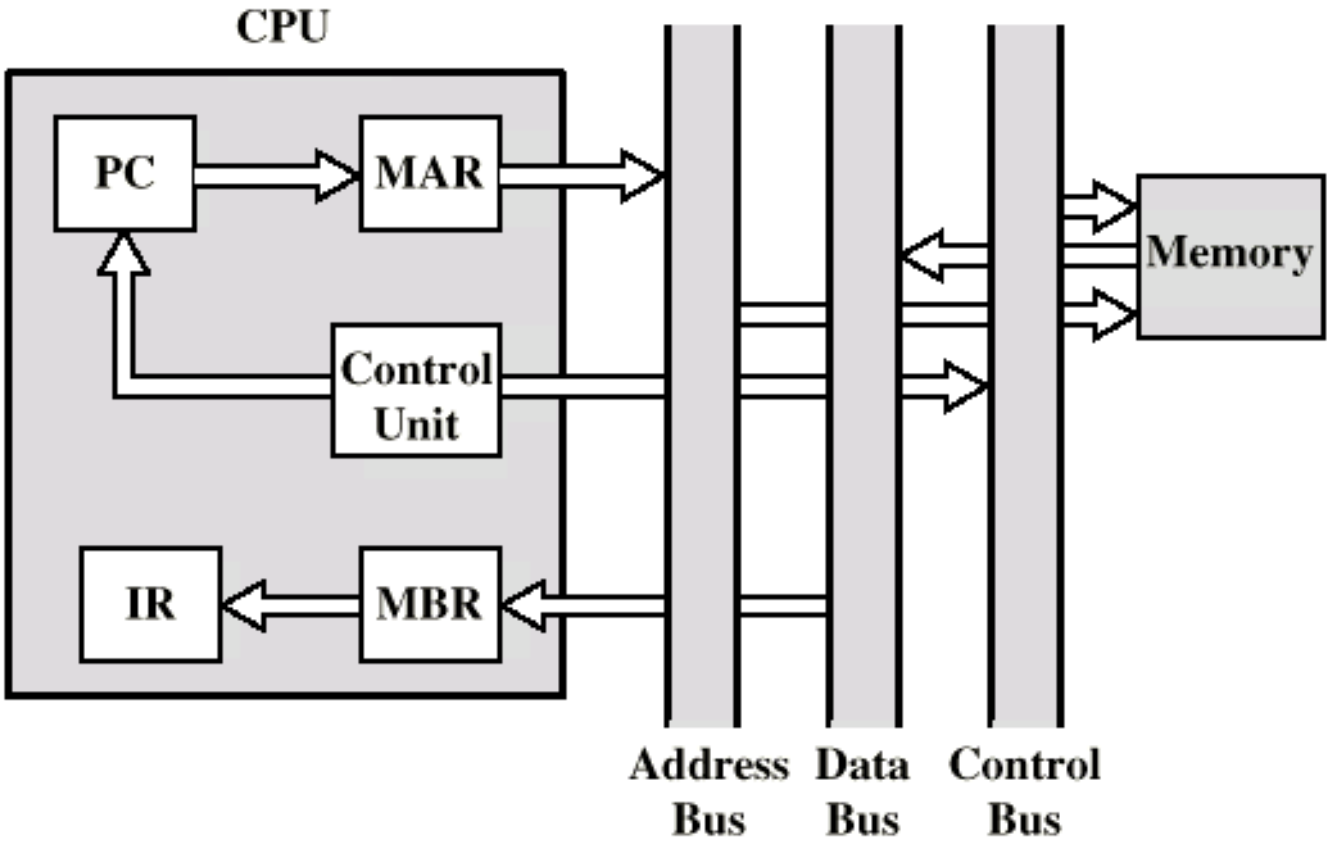




Luồng dữ liệu – data flow

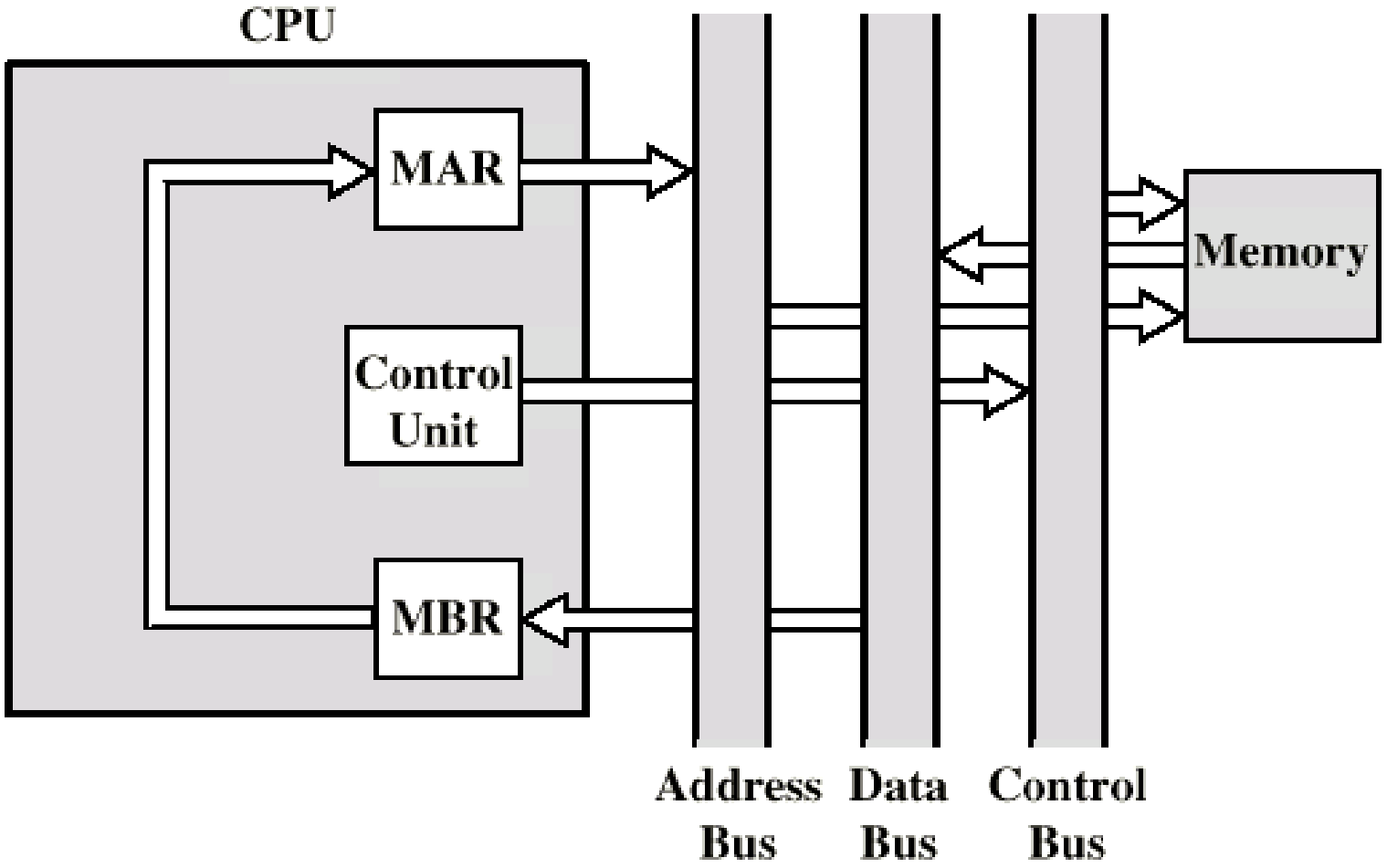
- Tải lệnh: (phụ thuộc vào thiết kế CPU)
  - Copy nội dung PC vào MAR → xác lập địa chỉ MM trên bus địa chỉ
  - CU gửi yêu cầu đọc MM
  - Kết quả sẽ có trên bus dữ liệu → copy vào MBR → copy đến IR
  - PC được tăng thêm 1 (von Neuman)
- Data fetch: IR được phân tích, tuỳ thuộc vào mỗi kiểu đánh địa chỉ để tiến hành tải toán hạng.
  - Địa chỉ trực tiếp: tải giống tải lệnh (chu trình trực tiếp)
  - Địa chỉ gián tiếp: (chu trình gián tiếp)
    - N bits phải nhất của MBR được chuyển đến MAR
    - CU yêu cầu đọc MM
    - Kết quả được chuyển đến MBR

Data Flow : Fetch Diagram



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

Data Flow : Indirect Diagram

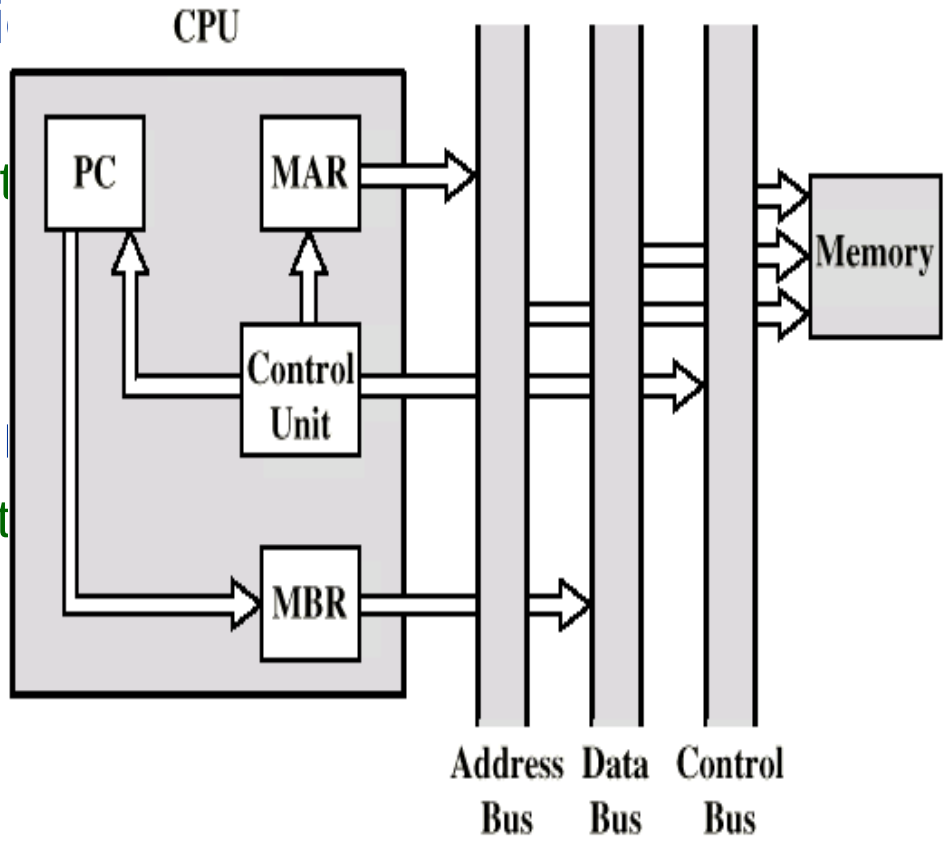


Data Flow : Execute

- Có thể thể hiện dưới nhiều hình thức khác nhau
- Phụ thuộc vào lệnh thi hành
- Có thể kèm theo
  - Memory-I/O read/write
  - ALU operations

Data Flow : Interrupt

- Lưu PC hiện thời để có thể ti
  - Contents of PC copied to MBR
  - Special memory location (e.g. st
  - MBR written to memory
- PC ← địa chỉ của hàm xử lý
  - Next instruction (first of interrupt



2. Pipeline

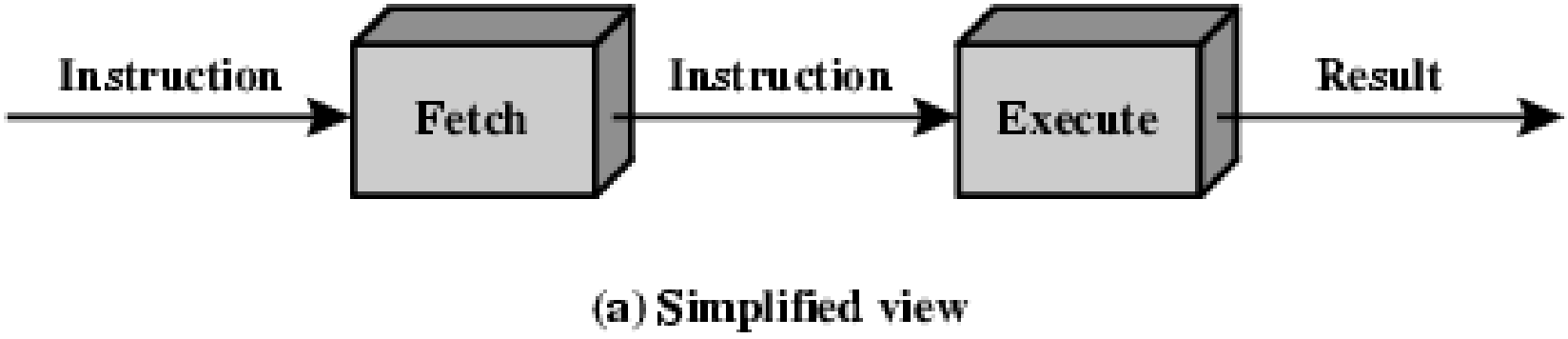
- Idea:
  - Fetching: thường truy cập bộ nhớ chính
  - Execution: thường không truy cập bộ nhớ chính

➔ Liệu có thể tải lệnh kế tiếp trong quá trình thi hành lệnh hiện thời ?

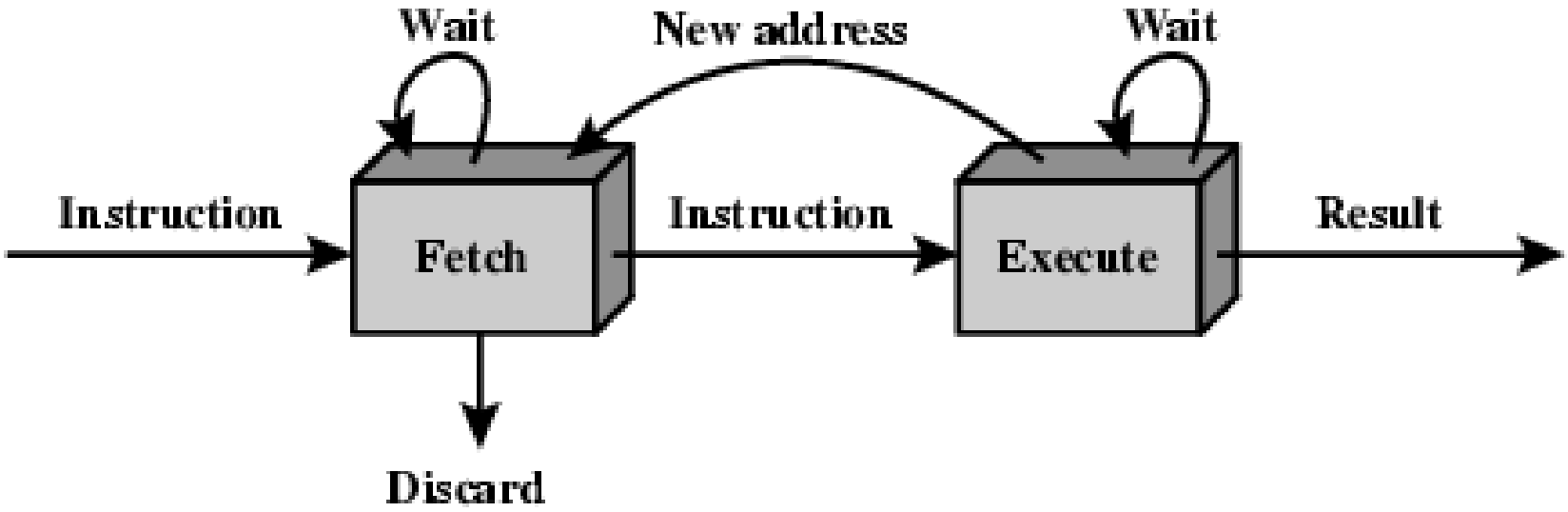
➔ Tải trước lệnh - instruction prefetch: thêm các tầng để cải thiện hiệu năng

- Fetch thường có thời gian thực thi ngắn hơn so với Execute
  - Prefetch more than one instruction?
- Tuy nhiên, với các lệnh rẽ nhánh/nhảy: tải trước có thể không cải thiện được hiệu năng!

Pipeline lệnh hai tầng



(a) Simplified view



(b) Expanded view

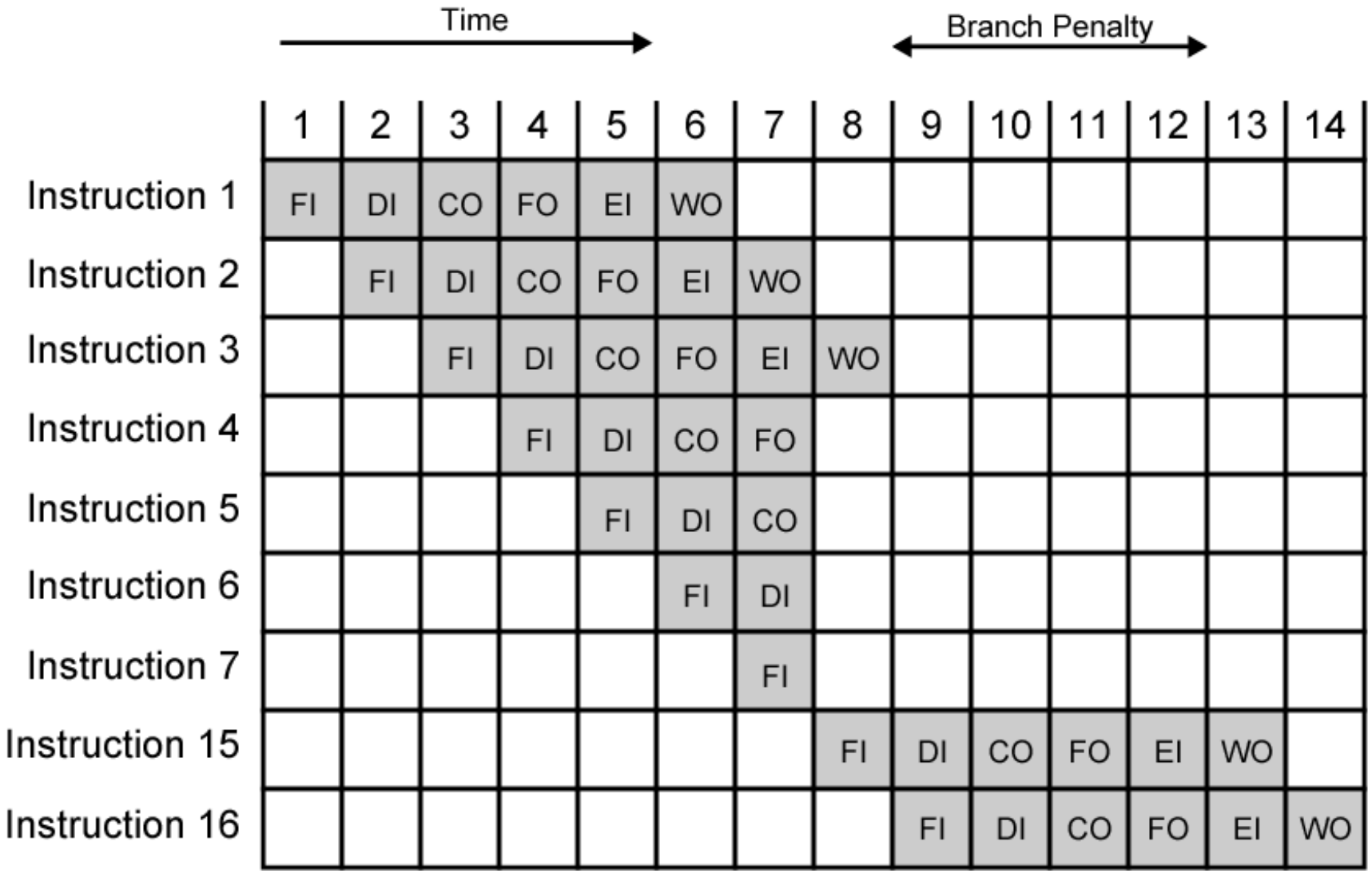
Pipelining

- Chia việc thi hành một lệnh thành nhiều bước con:
  - Fetch instruction
  - Decode instruction
  - Calculate operands (i.e. EAs)
  - Fetch operands
  - Execute instruction
  - Write operands
- Thi hành gối đầu nhau (overlap) các thao tác con trên

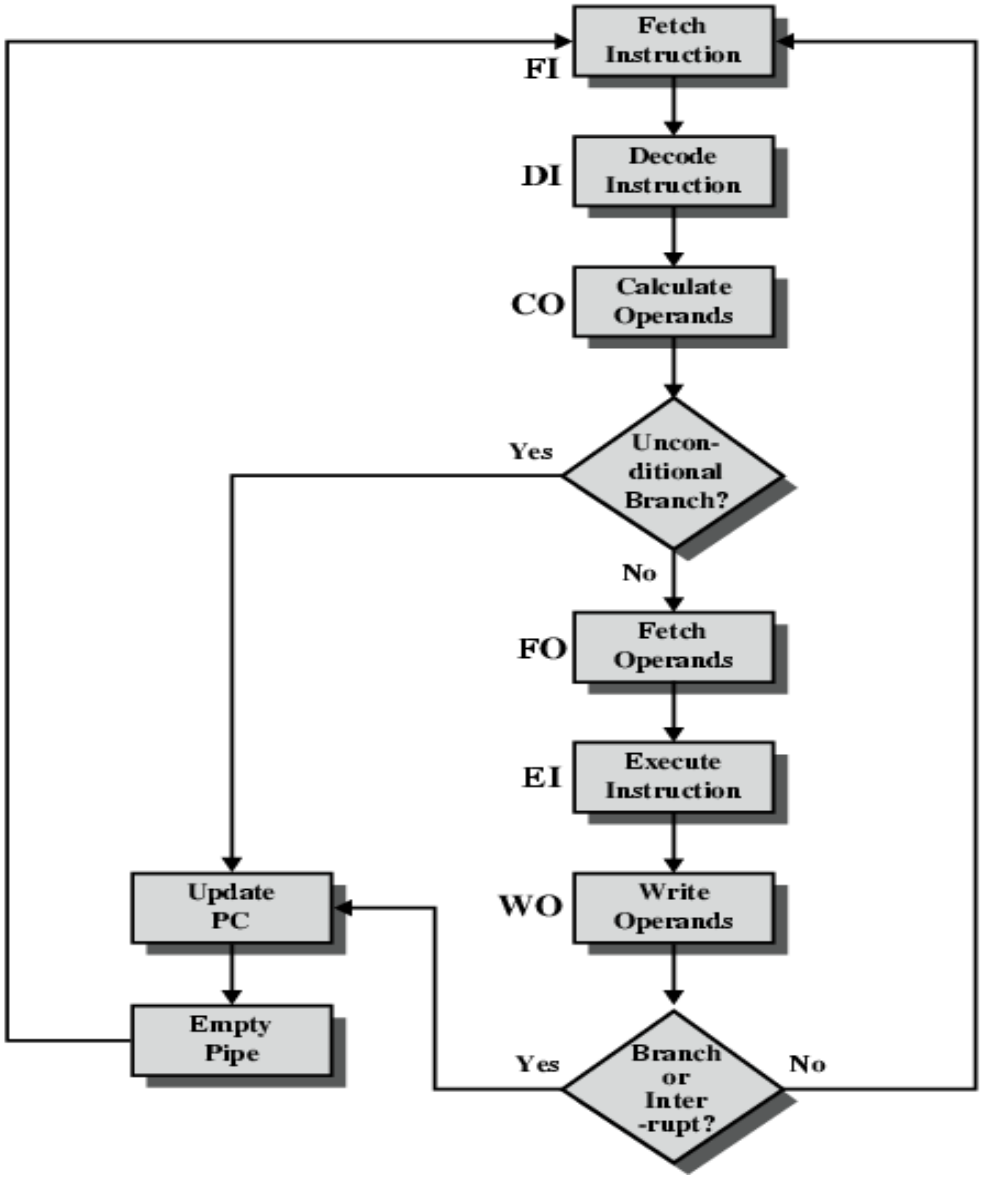
Biểu đồ thời gian thao tác trong pipeline lệnh

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

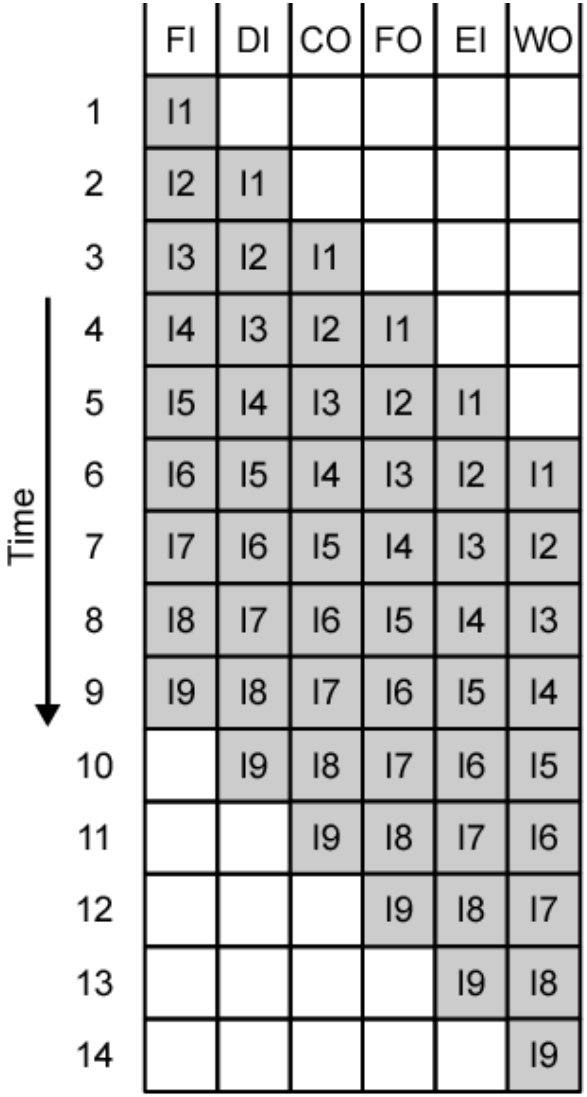
Ảnh hưởng của rẽ nhánh có điều kiện trong các bộ pipeline



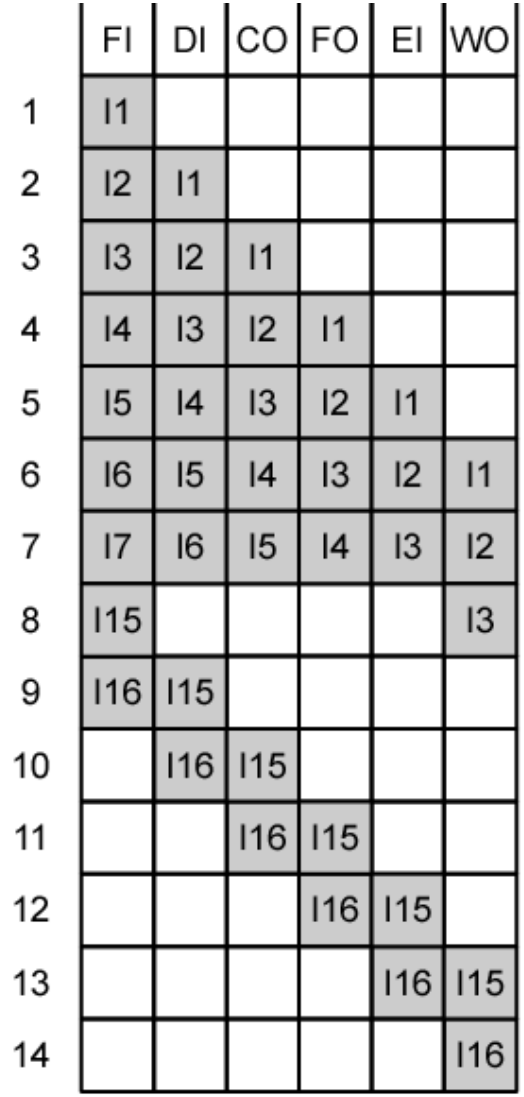
Pipeline lệnh sáu tầng



Thể hiện khác về bộ pipeline



(a) No branches



(b) With conditional branch

Hiệu năng

Thi hành *n* lệnh với một pipeline *k* tầng

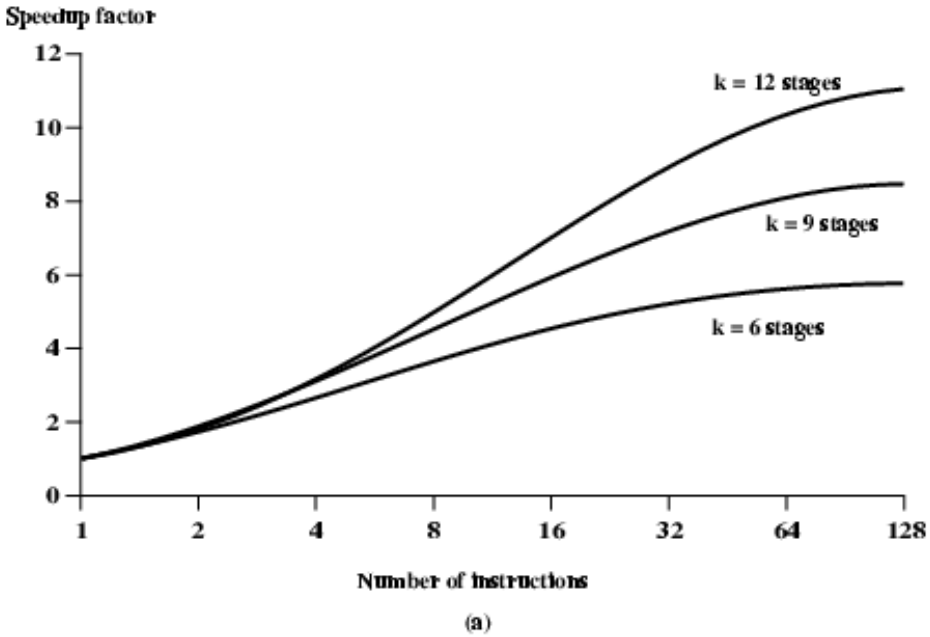
- Số chu trình :

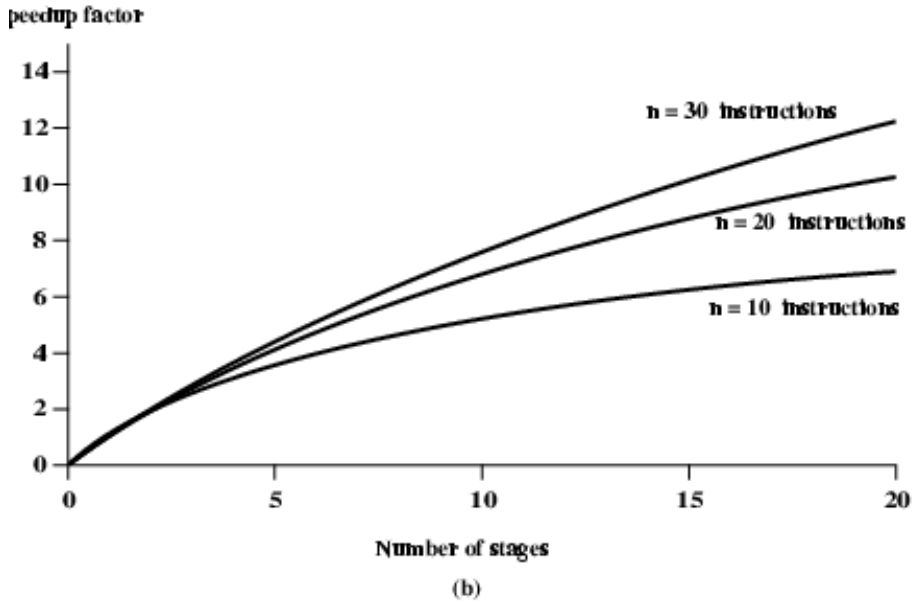
  $T_k = k + (n - 1)$

- Tăng tốc :

$$\frac{T_1}{T_k} = \frac{nk}{k + (n - 1)}$$

Hệ số tăng tốc



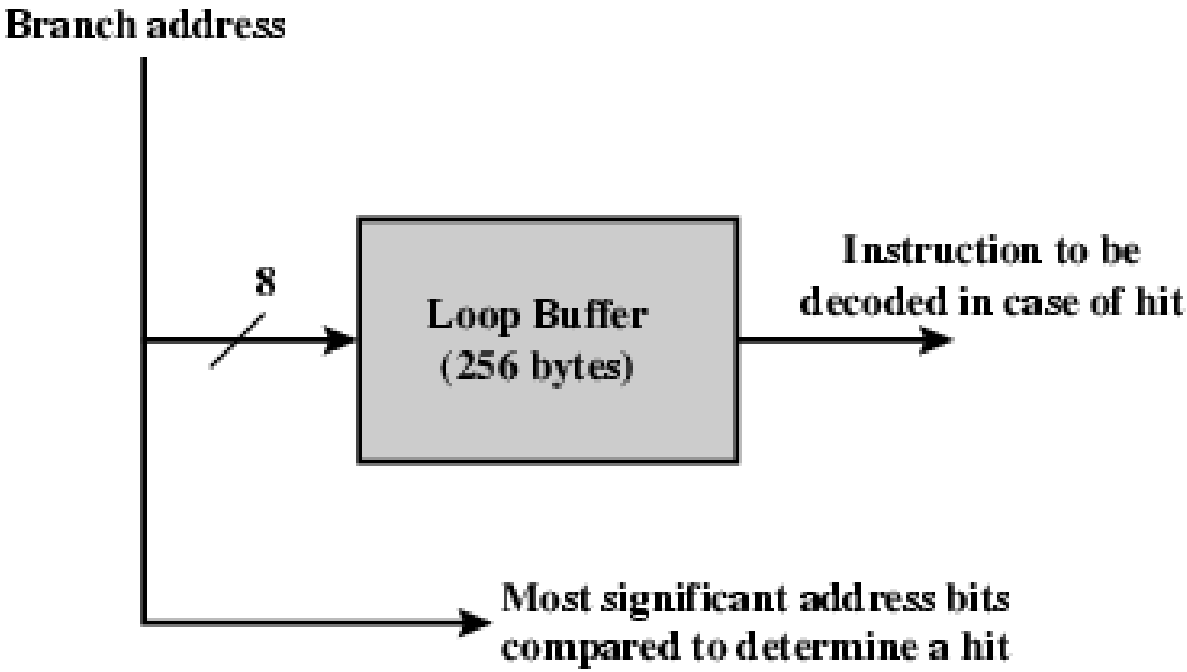


Xử lý rẽ nhánh

- Multiple Streams
  - 2 pipelines
  - prefetch mỗi nhánh vào mỗi pipeline phù hợp
  - ➔
  - Có thể dẫn đến tình trạng quá tải bus & register (contention)
  - Có thể cần nhiều hơn 2 pipelines khi có nhiều nhánh
- Prefetch Branch Target
  - Tải trước cả đích nhánh sau khi tải trước lệnh kế tiếp
  - Giữ lại cho đến khi lệnh rẽ nhánh đó thực thi xong
  - Được cài đặt trong IBM 360/91

Xử lý rẽ nhánh...

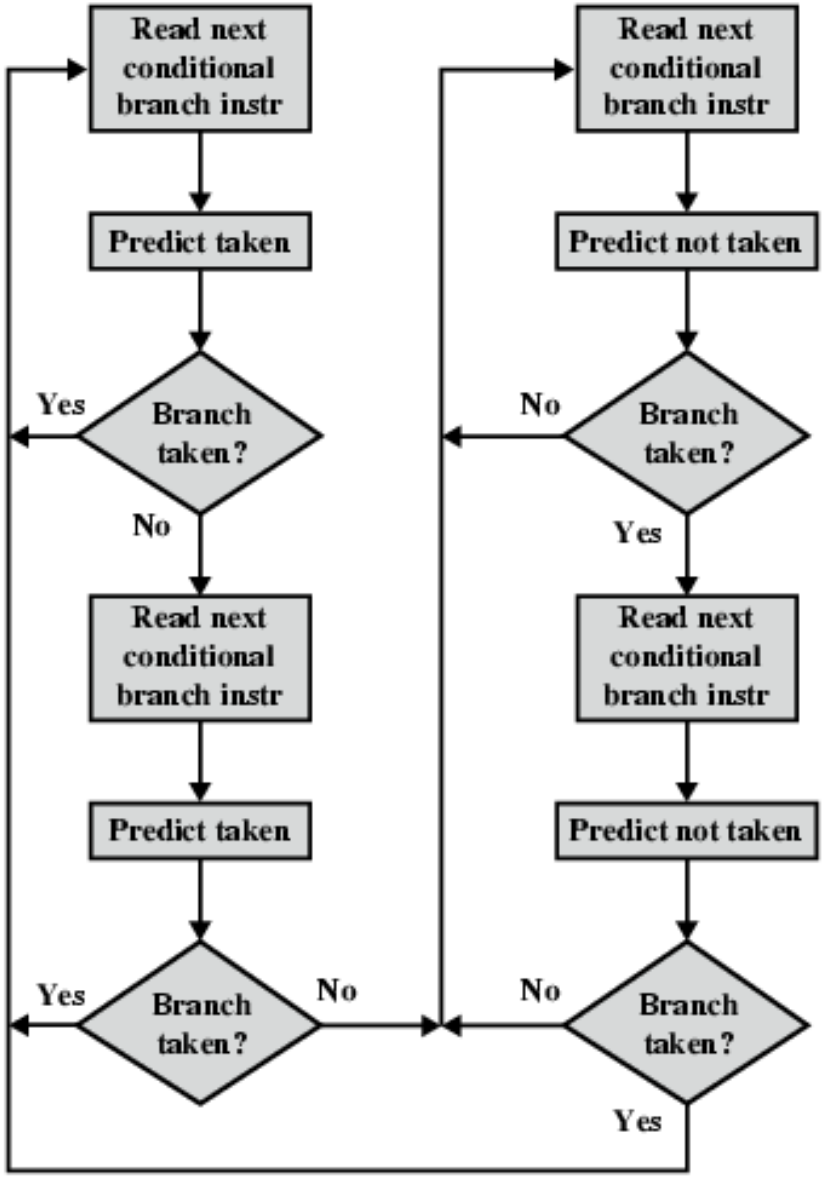
- Loop buffer
  - Very fast memory
  - Maintained by fetch stage of pipeline
  - Check buffer before fetching from memory
  - Very good for small loops or jumps
  - Used by CRAY-1



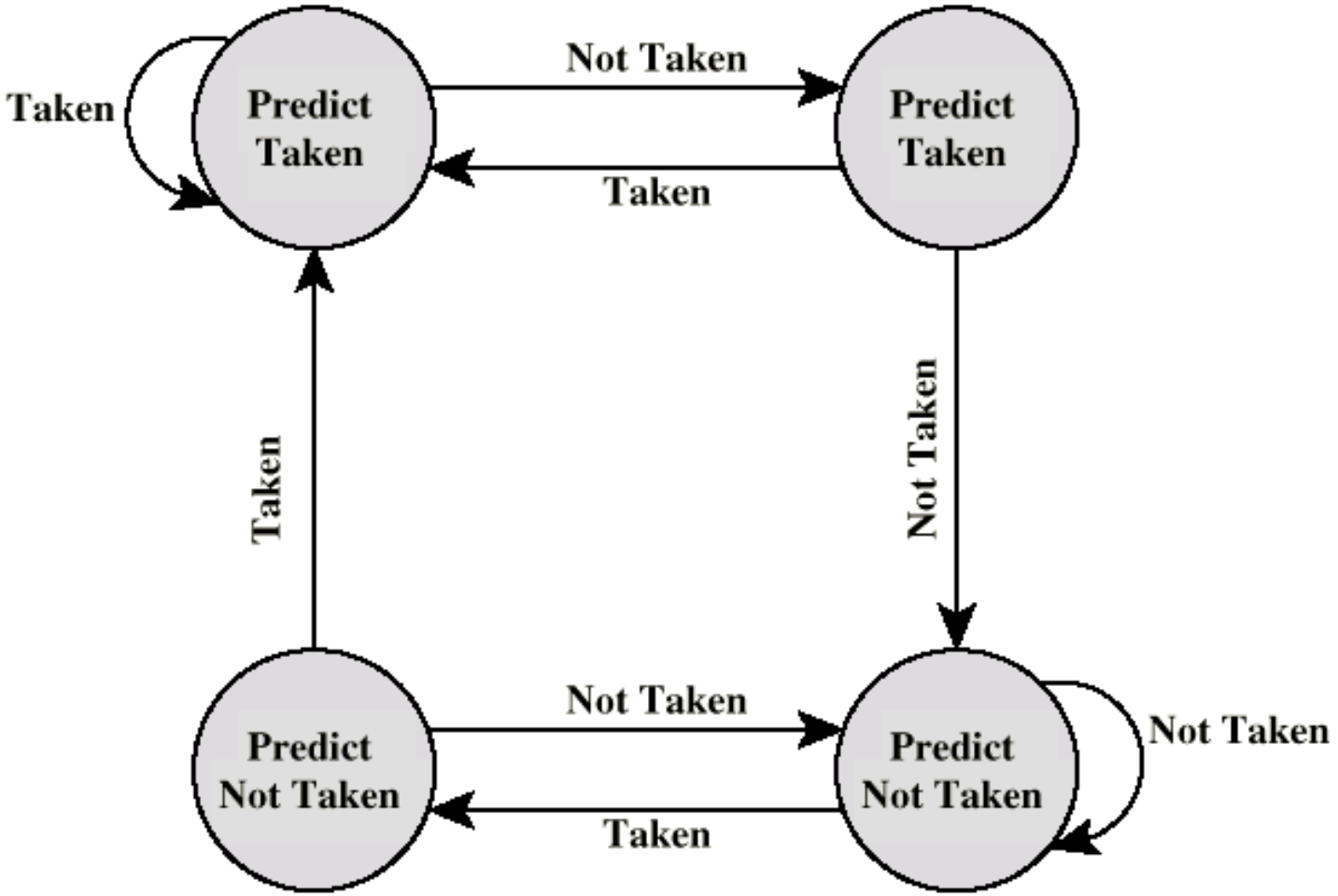
Xử lý rẽ nhánh...

- Branch prediction
  - Predict never taken
    - Assume that jump will not happen
    - Always fetch next instruction
    - 68020 & VAX 11/780
    - VAX will not prefetch after branch if a page fault would result (O/S v CPU design)
  - Predict always taken
    - Assume that jump will happen
    - Always fetch target instruction
  - Predict by Opcode
    - Some instructions are more likely to result in a jump than others
    - Can get up to 75% success
  - Taken/Not taken switch
    - Based on previous history
    - Good for loops

Biểu đồ dự đoán rẽ nhánh



Lược đồ trạng thái dự đoán rẽ nhánh



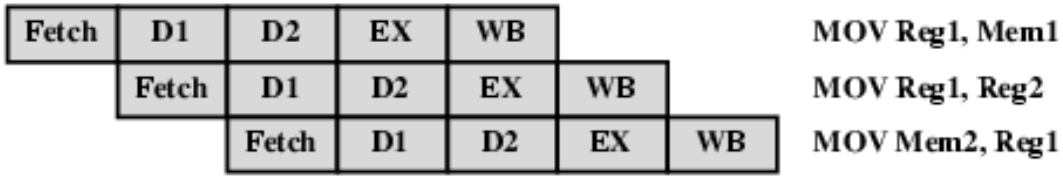
Xử lý rẽ nhánh...

- Delayed Branch
  - Do not take jump until you have to
  - Rearrange instructions

Intel 80486 Pipelining

- Fetch
  - From cache or external memory
  - Put in one of two 16-byte prefetch buffers
  - Fill buffer with new data as soon as old data consumed
  - Average 5 instructions fetched per load
  - Independent of other stages to keep buffers full
- Decode stage 1
  - Opcode & address-mode info
  - At most first 3 bytes of instruction
  - Can direct D2 stage to get rest of instruction
- Decode stage 2
  - Expand opcode into control signals
  - Computation of complex address modes
- Execute
  - ALU operations, cache access, register update
- Writeback
  - Update registers & flags
  - Results sent to cache & bus interface write buffers

80486 Instruction Pipeline Examples



(a) No Data Load Delay in the Pipeline

(b) Pointer Load Delay

(c) Branch Instruction Timing

3. CISC & RISC

- CISC – Complex Instruction Set Computers, máy tính có tập lệnh phức tạp <> RISC – Reduced Instruction Set Computers
- Một số đặc điểm của CISC
  - Đơn giản hoá quá trình biên dịch chương trình? Tuy nhiên
    - Khó khăn trong việc sử dụng hết các lệnh phức tạp
    - Khó khăn trong việc tối ưu hoá chương trình
  - Dung lượng chương trình bé đi?
    - Memory không phải là vấn đề quá trong trọng hiện nay (cheaper)
    - Việc sử dụng CIS có thể không làm giảm số dung lượng thực mà chỉ ảnh hưởng đến hình thức thể hiện (shorter symbolic form)
      - Nhiều lệnh ➔ opcodes sử dụng nhiều bits hơn
      - Trường tham chiếu đến register/memory được cấp phát ít bits hơn
  - Thi hành chương trình nhanh hơn?
    - CISC cần có CU phức tạp hơn ➔ các lệnh đơn giản thực thi chậm hơn

Đặc điểm của RISC

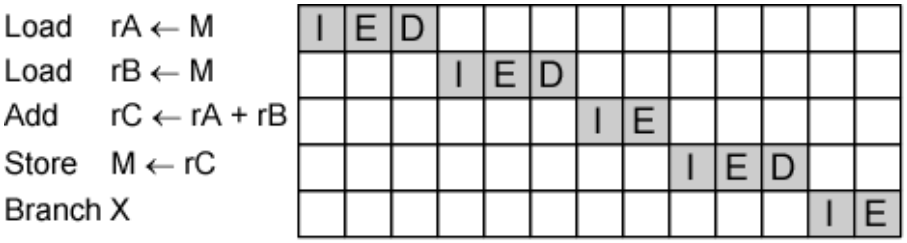
- Mỗi lệnh thực thi trong một chu kỳ
- Sử dụng chiến thuật thao tác register-to-register
- Ít kiểu đánh địa chỉ, đơn giản
- Format lệnh đơn giản và ít
- Thiết kế cứng hoá, không có microcode
- Format lệnh có kích thước cố định, tập lệnh đơn giản, số lượng ít
- Chú trọng hơn đến thời gian dịch chương trình
- Số lượng lớn thanh ghi đa dụng, có cơ chế tối ưu hoá việc sử dụng thanh ghi
- Chú trọng đến tối ưu pipeline

RISC Pipelining

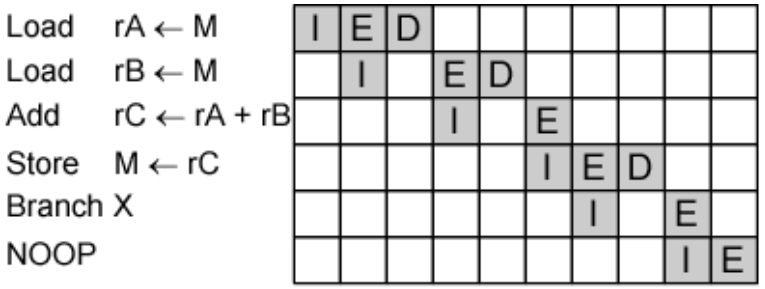
- Hai pha với những lệnh thi hành
  - I: Instruction fetch
  - E: Execute
    - ALU operation with register input and output

- Với những lệnh load và store
  - I: Instruction fetch
  - E: Execute
    - Calculate memory address
  - D: Memory
    - Register to memory or memory to register operation

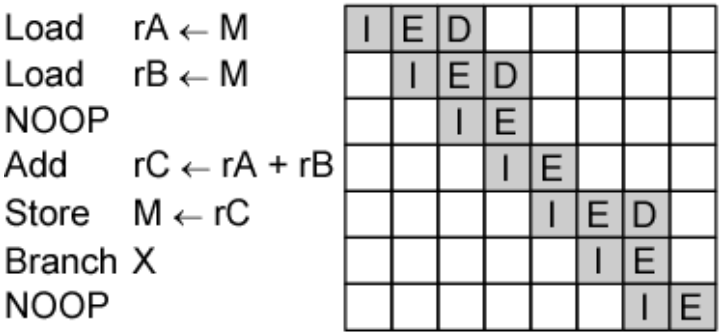
Tác dụng của Pipelining



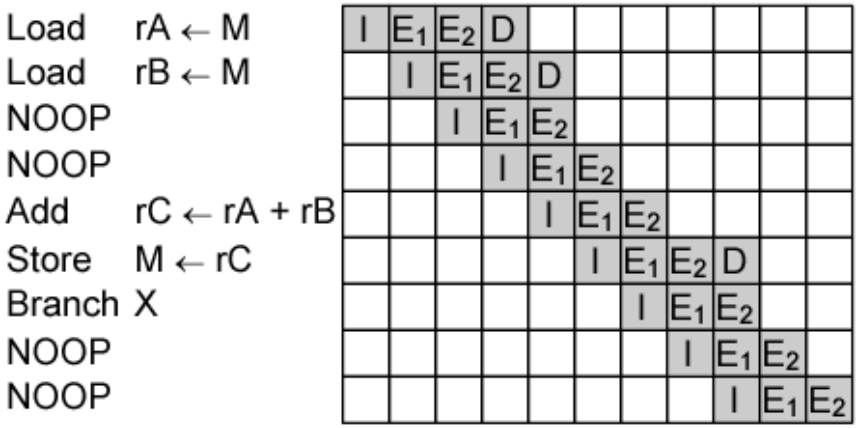
(a) Sequential execution



(b) Two-stage pipelined timing



(c) Three-stage pipelined timing



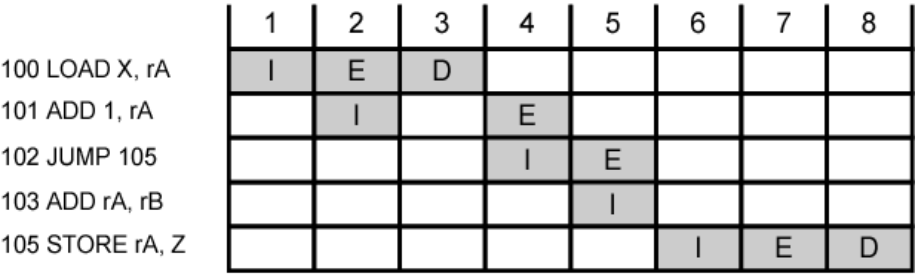
(d) Four-stage pipelined timing

Tối ưu hoá cơ chế Pipelining

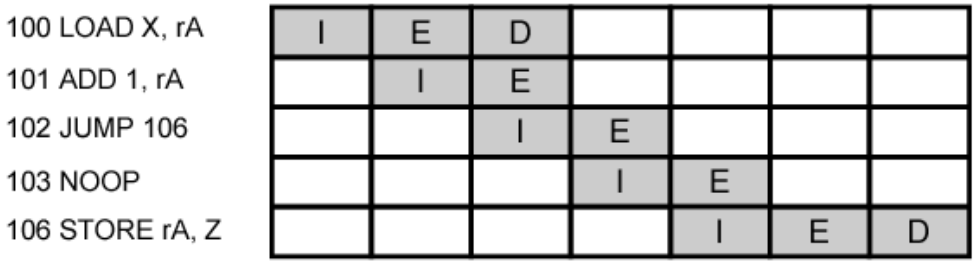
- Delayed branch
  - Chỉ ghi nhận (take effect) sau khi thi hành lệnh kế tiếp
  - Lệnh kế tiếp này phải ở slot chờ (delay slot)
    - Normal and Delayed Branch

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	

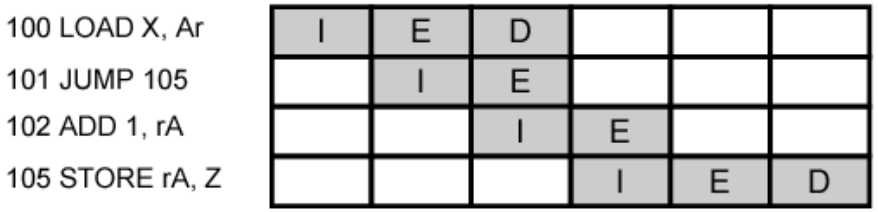
Minh hoạ cơ chế Delayed Branch



(a) Traditional Pipeline



(b) RISC Pipeline with Inserted NOOP



(c) Reversed Instructions

Tối ưu hoá cơ chế Pipelining...

- Delayed Load
  - Thanh ghi đích được CPU khoá lại
  - Tiếp tục thi hành dòng lệnh cho đến khi có nhu cầu sử dụng thanh ghi đó
  - Chờ cho đến khi tải xong
  - Sắp xếp lại lệnh có thể cho phép thi hành hiệu quả hơn trong khi tải lên
- Loop Unrolling
  - Replicate body of loop a number of times
  - Iterate loop fewer times
  - Reduces loop overhead
  - Increases instruction parallelism
  - Improved register, data cache or TLB locality

Bàn luận

- Định lượng - Quantitative
  - Đánh giá, so sánh về kích cỡ chương trình và tốc độ thi hành
- Định tính - Qualitative
  - Xem xét khả năng hỗ trợ ngôn ngữ bậc cao và kế thừa kết quả từ các mạch VLSI thực tế
- Problems
  - Không có cặp RISC & CISC được so sánh trực tiếp với nhau
  - Không có tập chương trình tests chuẩn
  - Rất khó để phân tách tác động của phần cứng từ tác động phần mềm
  - Thường các so sánh được thử nghiệm, đánh giá trên một vài mẫu chuyên biệt chứ không được tiến hành đại trà
  - Đa phần CPU thương mại sử dụng cả hai

So sánh các kiến trúc

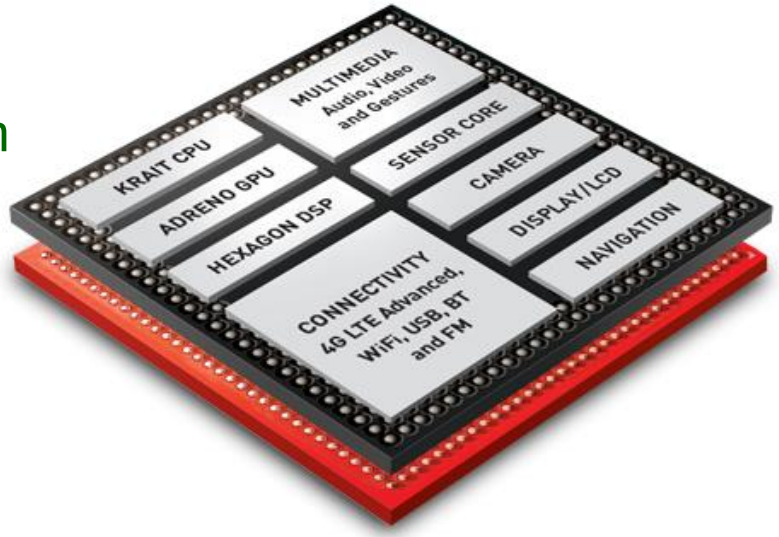
	Complex Instruction Set (CISC)Computer			Reduced Instruction Set (RISC) Computer		Superscalar		
Characteristic	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000	PowerPC	Ultra SPARC	MIPS R10000
Year developed	1973	1978	1989	1987	1991	1993	1996	1996
Number of instructions	208	303	235	69	94	225		
Instruction size (bytes)	2–6	2–57	1–11	4	4	4	4	4
Addressing modes	4	22	11	1	1	2	1	1
Number of general-purpose registers	16	16	8	40 - 520	32	32	40 - 520	32
Control memory size (Kbits)	420	480	246	—	—	—	—	—
Cache size (KBytes)	64	64	8	32	128	16-32	32	64

Apple A8 CPU

- 2 cores
- Max. CPU clock: 1.38 GHz
- Min. feature size: 20 nm
- Instruction set: ARMv8-A
- L1 cache: Per core: 64 KB instruction + 64 KB data
- L2 cache: 1 MB shared
- L3 cache: 4 MB
- 1 GB of LPDDR3 RAM included in the package
- GPU: PowerVR Series 6XT GX6450 (quad core)
- 2 billion transistors, physical size reduced by 13% to 89 mm^2
- Produced by Taiwan Semiconductor Manufacturing Company Limited (TSMC)

Qualcomm Snapdragon

- Snapdragon 805
  - ARMv7-A, Quad-core Krait 450 CPU at up to 2.7 GHz per core
  - 16 KiB / 16 KiB L1 cache per core; 2 MiB L2 cache
  - 4K UHD video upscale & play
  - Dual camera image signal processor supporting up to 55 Megapixel, stereoscopic 3D
  - Adreno 420 GPU
  - LPDDR3 25.6 GB/s memory bandwidth
  - IZat Gen8B GNSS location technology
  - USB 2.0 and 3.0
  - Hexagon, QDSP6V5A, 600 MHz
  - e-MMC V5.0, UFS V2.0
  - BT4.1, 802.11ac Wi-Fi
  - 28 nm HPm (high performance mobile)
  - Devices: Samsung S5, G G3, Samsung Note 4 , Note Edge

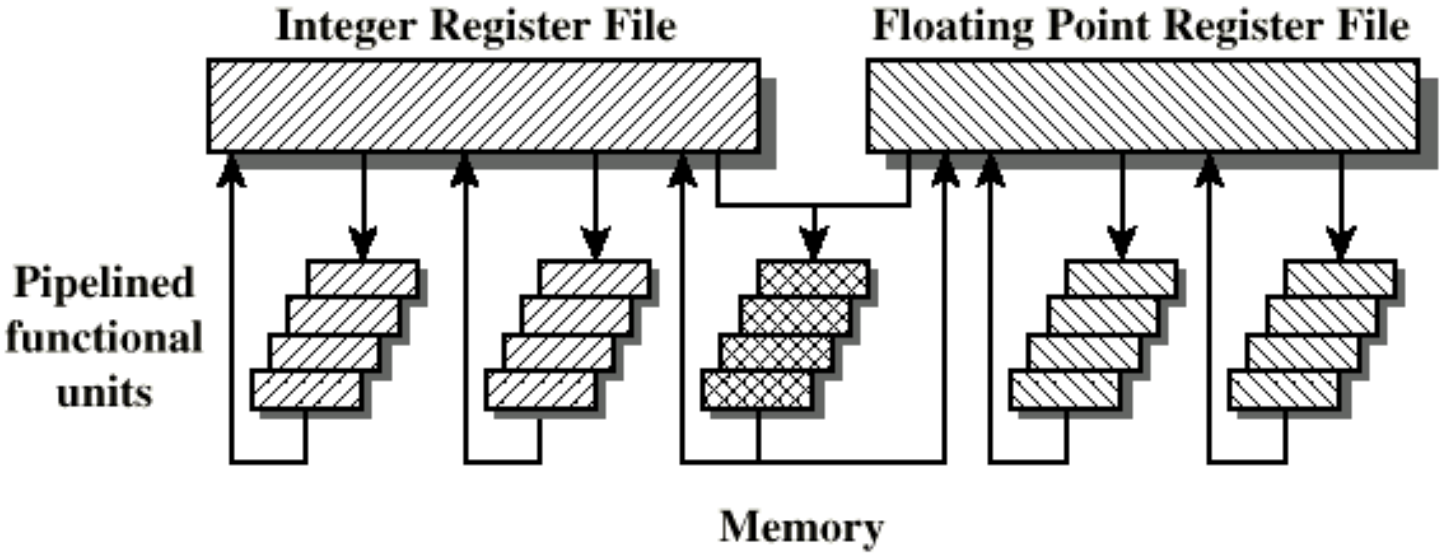


Qualcomm Snapdragon...

- Snapdragon 810
  - ARMv8-A, 4+4 cores, 2GHz?
  - 16 KiB / 16 KiB L1 cache per core; 2 MiB L2 cache
  - H.265/HEVC encoding/decoding
  - eMMC 5.0 support
  - 14-bit dual-ISP
  - support for triple-band (i.e. IEEE 802.11, IEEE 802.15 (Bluetooth) and IEEE 802.11ad (60 GHz).
  - Qualcomm acquired Wilocity
  - Adreno 430 GPU
  - LPDDR4 25.6 GB/s memory bandwidth
  - BT4.1, 802.11ac Wi-Fi
  - 20 nm

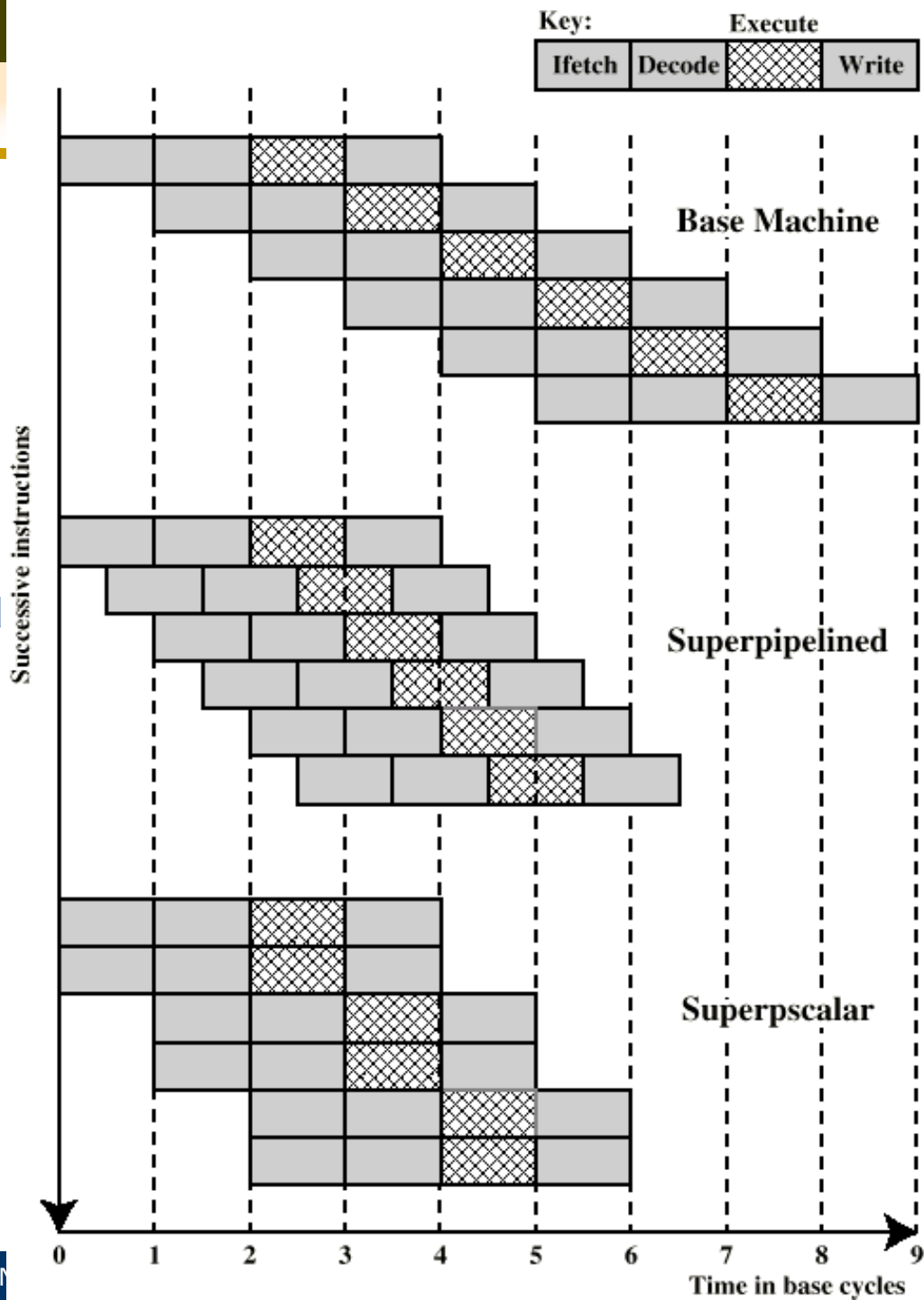
4. Superscalar & VLIW

- Các lệnh cơ bản (arithmetic, load/store, conditional branch) có thể được khởi tạo và thực thi một cách độc lập
- Cơ chế này áp dụng được với cả RISC & CISC, thực tế thì thường dùng trong các chip RISC
- Why?
  - Hầu hết các lệnh RISC đều là scalar quantities
  - Cải thiện các lệnh nầy sẽ cho phép cải thiện toàn bộ hệ thống



Superpipelined

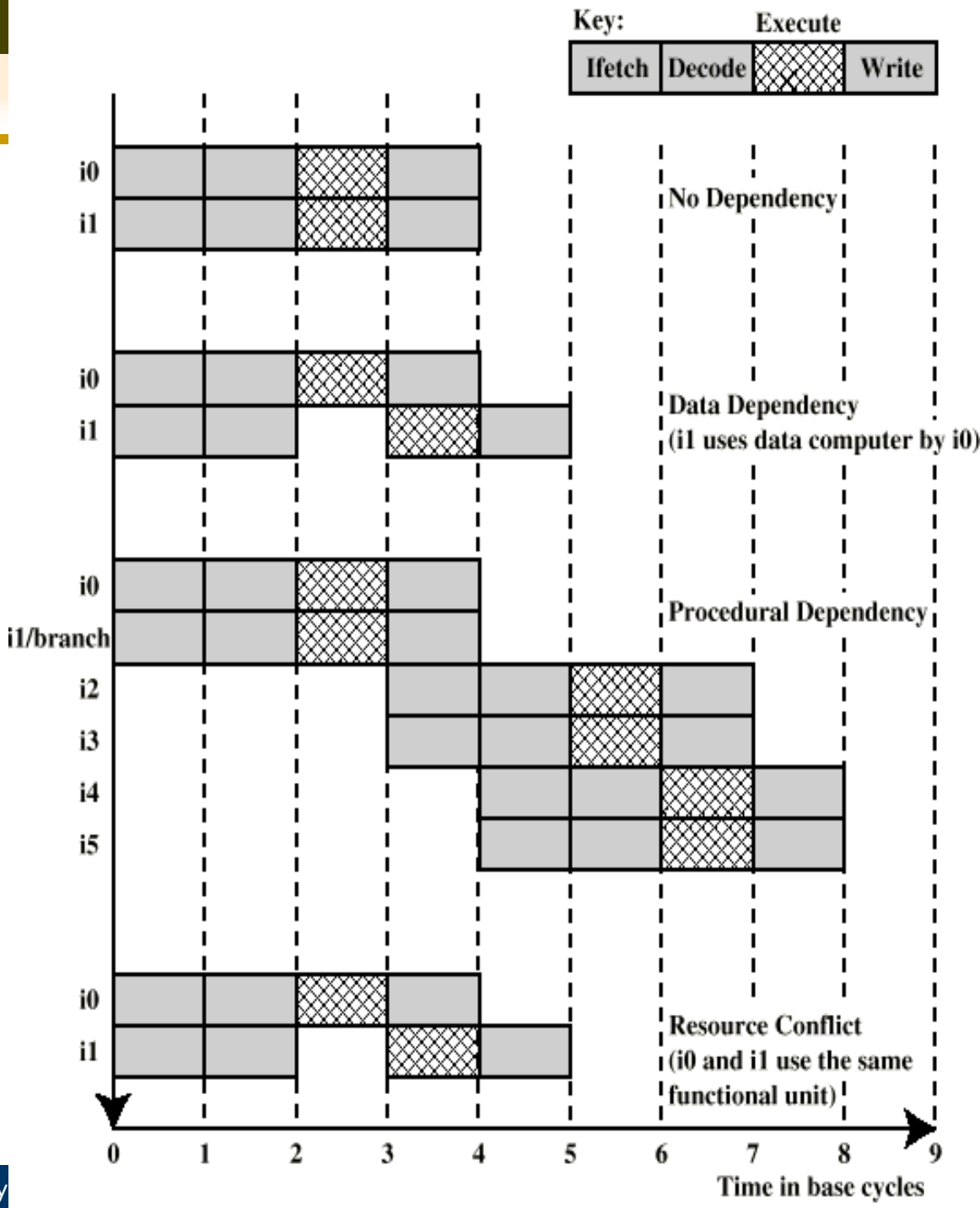
- Nhiều tầng pipeline cần ít hơn nửa chu kỳ đồng hồ
- Tăng gấp đôi tốc độ xung clock bên trong cho phép thi hành hai tác vụ mỗi chu kỳ clock bên ngoài
- Superscalar cho phép fetch/execute tiến hành song song



Hạn chế

- Đây là cơ chế song song hoá mức lệnh, phụ thuộc phần cứng
- Cần dựa trên compiler để tối ưu hoá
- Ngoài ra, còn phụ thuộc
  - True data dependency
  - Procedural dependency
  - Resource conflicts
  - Output dependency
  - Antidependency

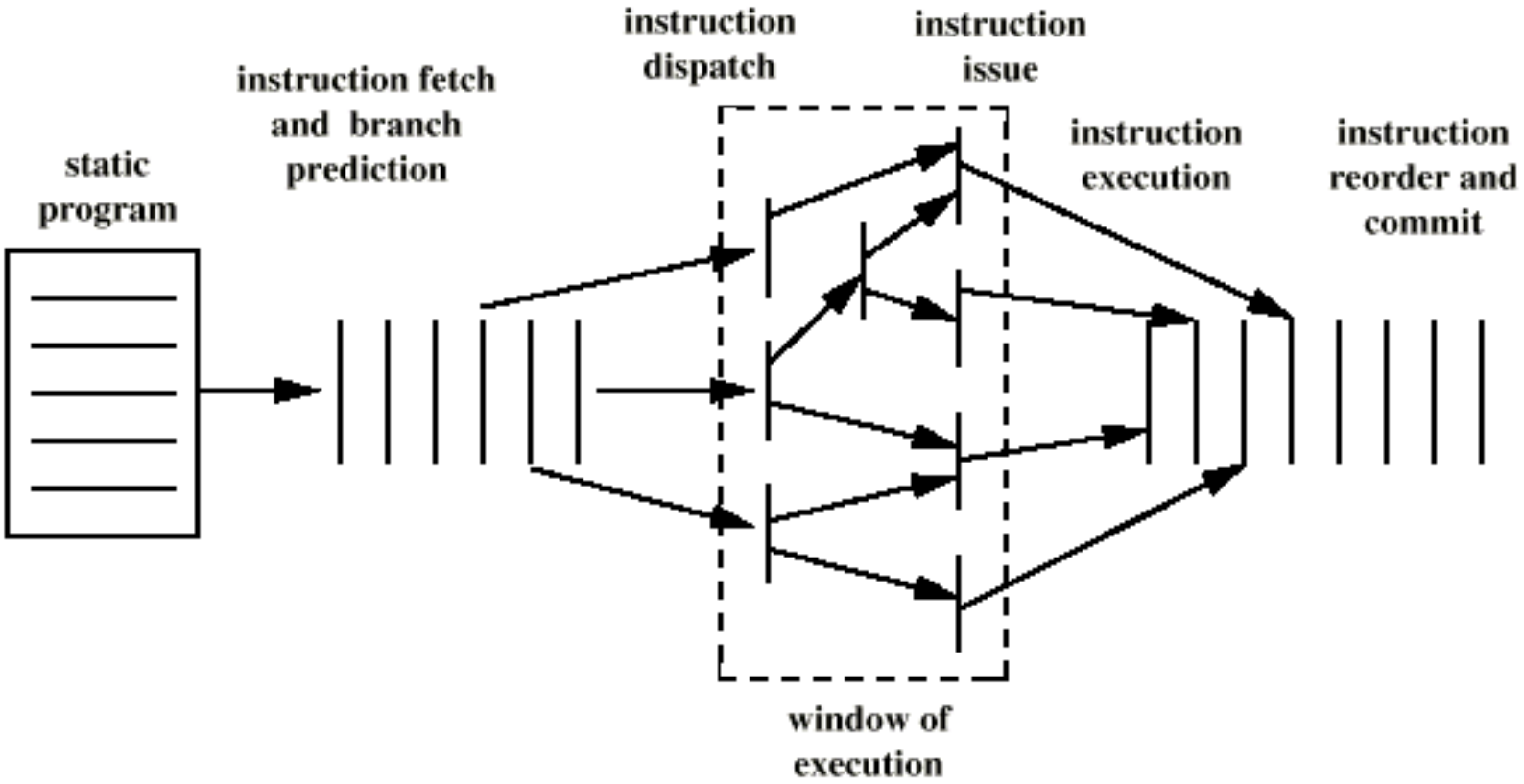
Tác động của phụ thuộc dữ liệu



Hướng thiết kế

- Song song hoá mức lệnh
  - Chuỗi lệnh được xem như độc lập
  - Quá trình thi hành có thể bị xếp chồng
  - Được quản trị thông qua các kỹ thuật kiểm soát phụ thuộc dữ liệu và hàm (procedural dependency)
- Song song hoá mức máy
  - Có khả năng mang lại lợi điểm hơn so với mức lệnh
  - Được quản trị bởi một tập các pipelines song song

Thi hành kiểu superscalar



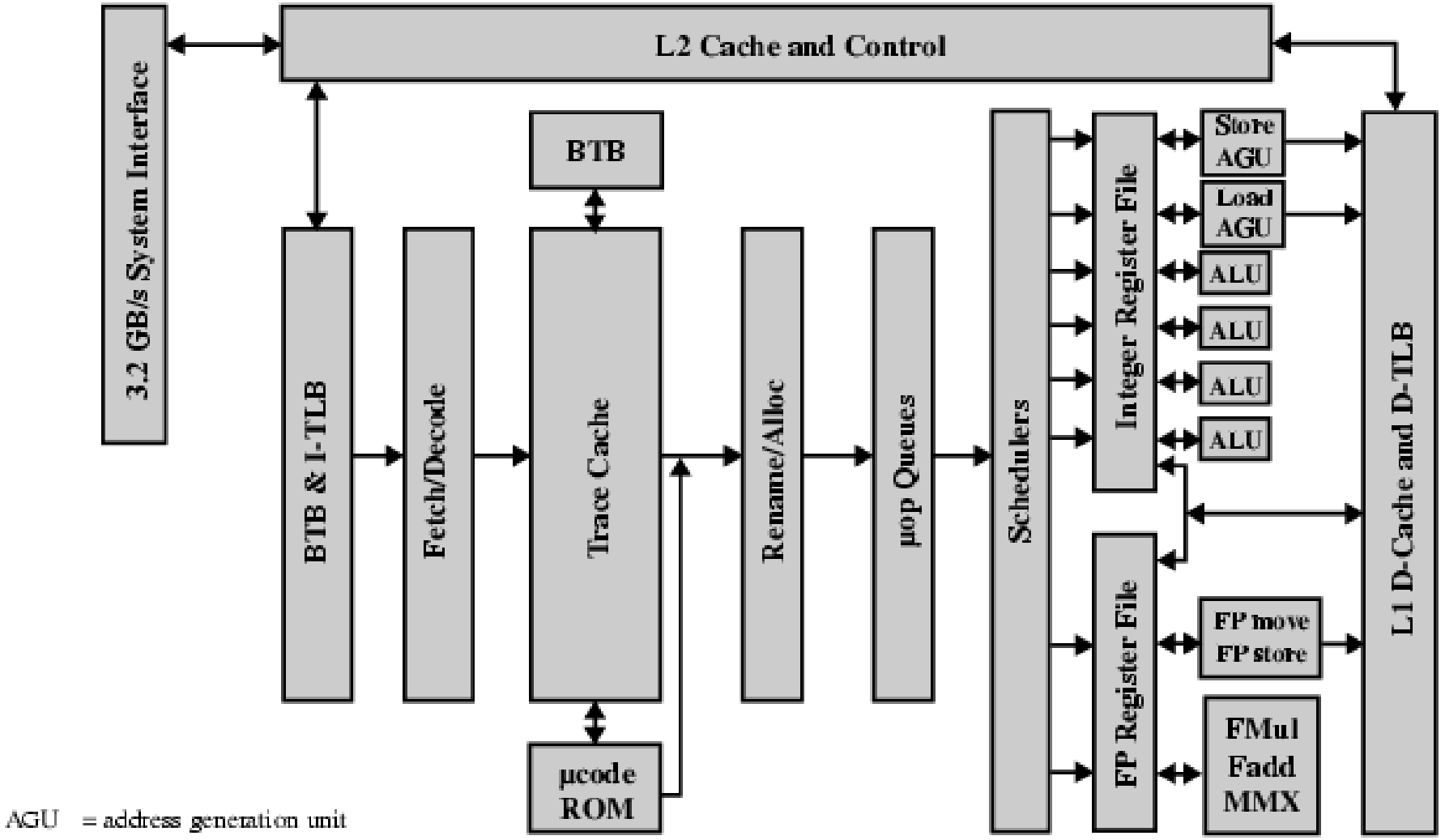
Cài đặt superscalar

- Tải nhiều lệnh đồng thời
- Sử dụng cổng logic để xác định phụ thuộc giữa các giá trị thanh ghi
- Cần cơ chế để trao đổi (communicate) các giá trị đó
- Cần cơ chế để khởi tạo đa lệnh song song
- Cần tài nguyên cho thi hành song song nhiều lệnh
- Cần cơ chế xác định trạng thái tiến trình (process state) đúng thứ tự

Ví dụ với Pentium 4

- 80486 - CISC
- Pentium – some superscalar components
  - Two separate integer execution units
- Pentium Pro – Full blown superscalar
- Subsequent models refine & enhance superscalar design

Pentium 4 Block Diagram

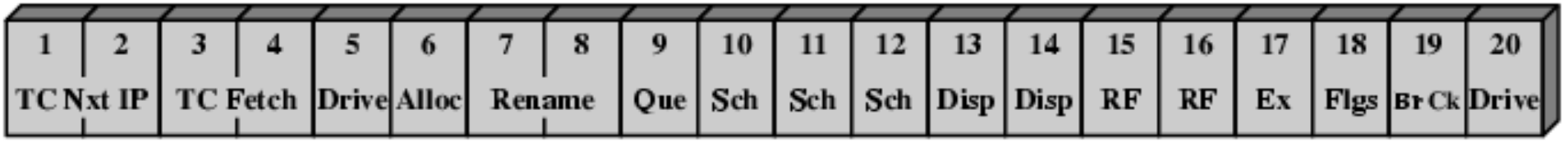


AGU = address generation unit
BTB = branch target buffer
D-TLB = data translation lookaside buffer
I-TLB = instruction translation lookaside buffer

Pentium 4 Operation

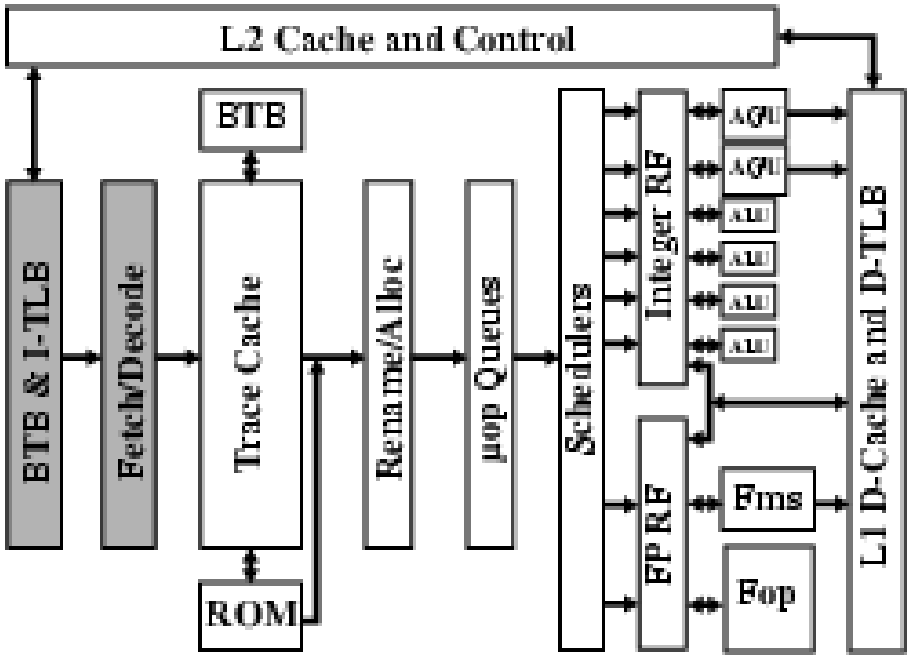
- Fetch instructions form memory in order of static program
- Translate instruction into one or more fixed length RISC instructions (micro-operations)
- Execute micro-ops on superscalar pipeline
  - micro-ops may be executed out of order
- Commit results of micro-ops to register set in original program flow order
- Outer CISC shell with inner RISC core
- Inner RISC core pipeline at least 20 stages
  - Some micro-ops require multiple execution stages
    - Longer pipeline
  - c.f. five stage pipeline on x86 up to Pentium

Pentium 4 Pipeline

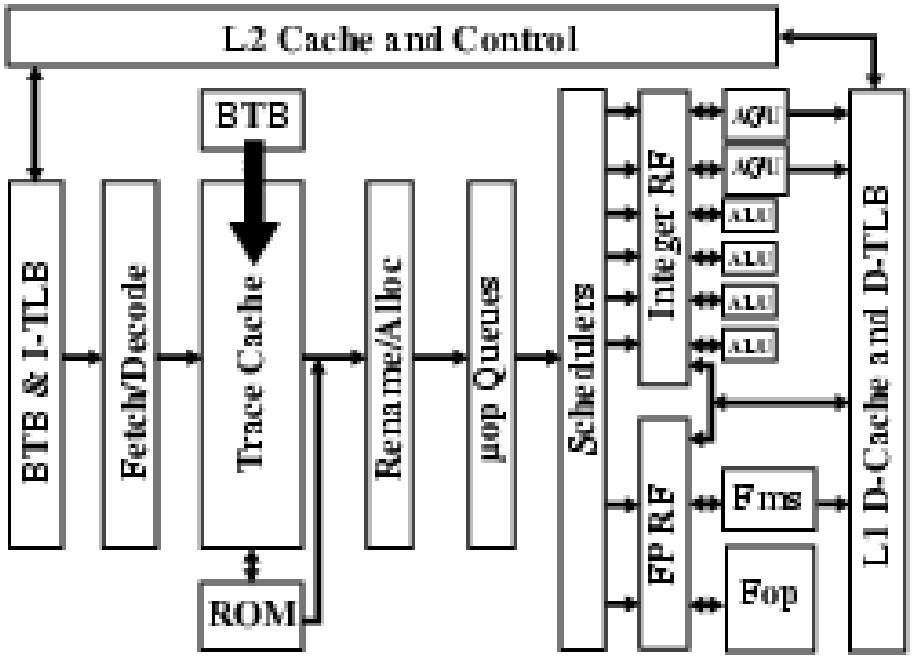


TC Next IP = trace cache next instruction pointer
TC Fetch = trace cache fetch
Alloc = allocate
Rename = register renaming
Que = micro-op queuing
Sch = micro-op scheduling
Disp = Dispatch
RF = register file
Ex = execute
Flgs = flags
Br Ck = branch check

Pentium 4 Pipeline Operation (1)

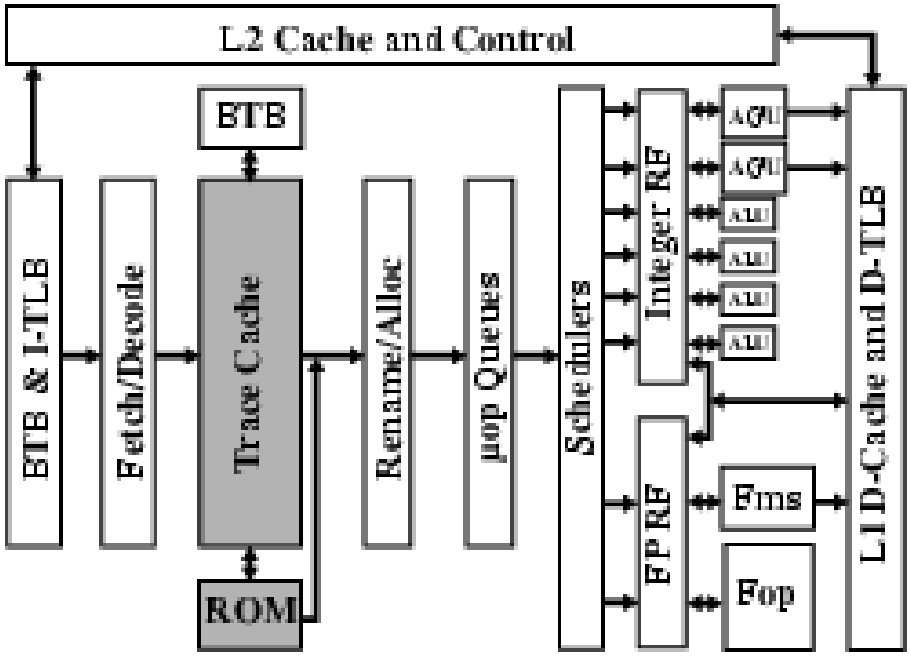


(a) Generation of micro-ops

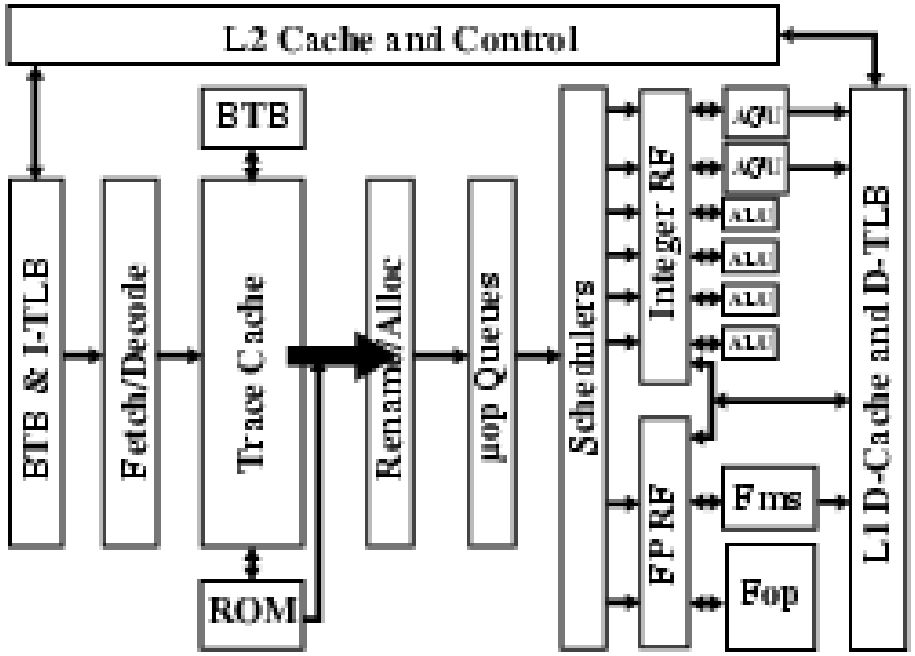


(b) Trace cache next instruction pointer

Pentium 4 Pipeline Operation (2)

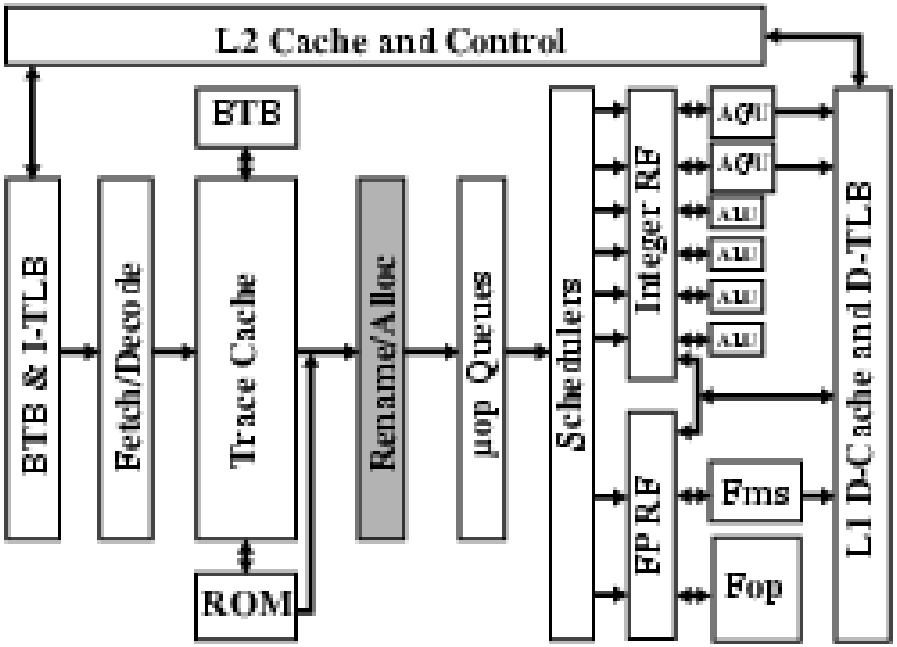


(c) Trace cache fetch

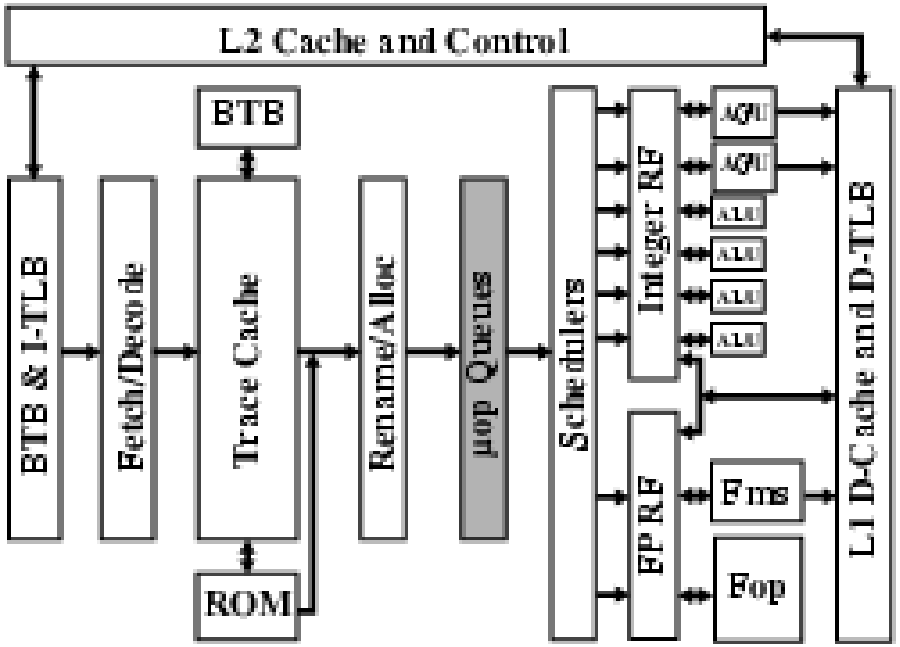


(d) Drive

Pentium 4 Pipeline Operation (3)

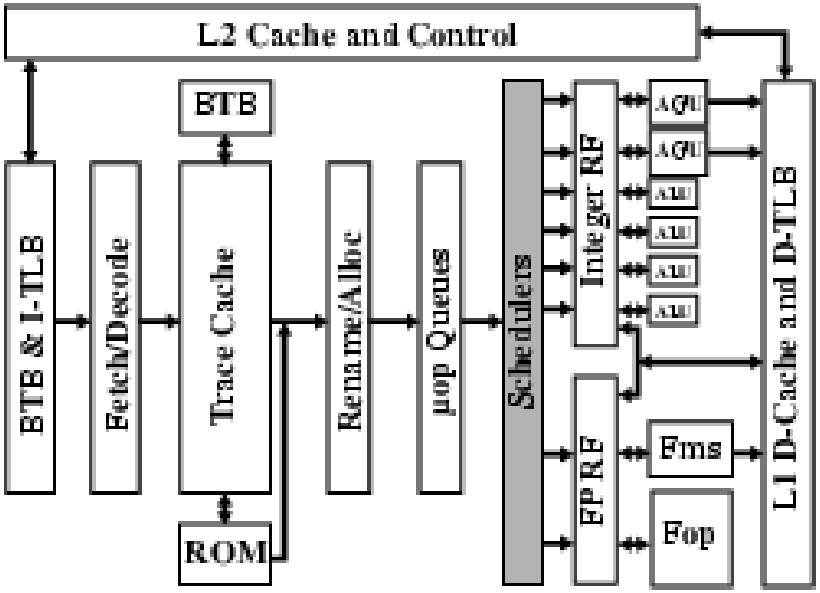


(e) Allocate; Register renaming

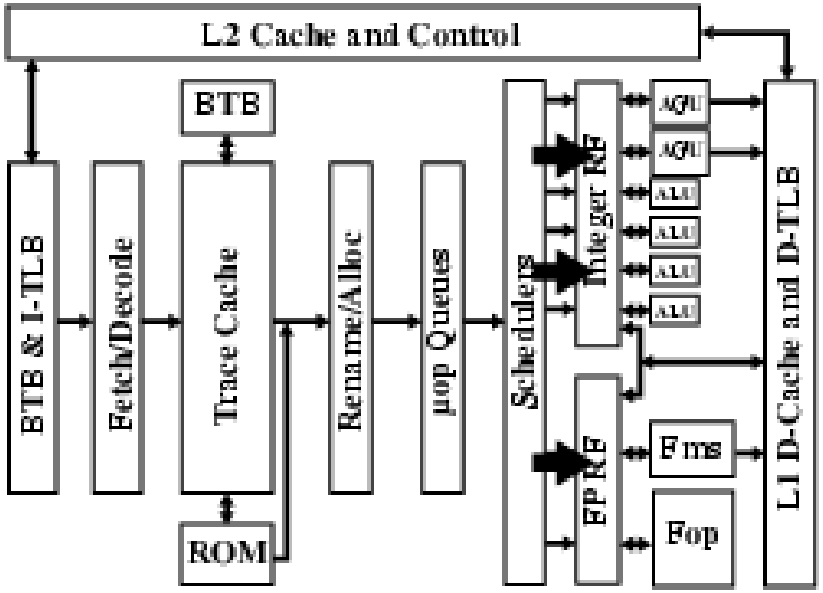


(f) Micro-op queuing

Pentium 4 Pipeline Operation (4)

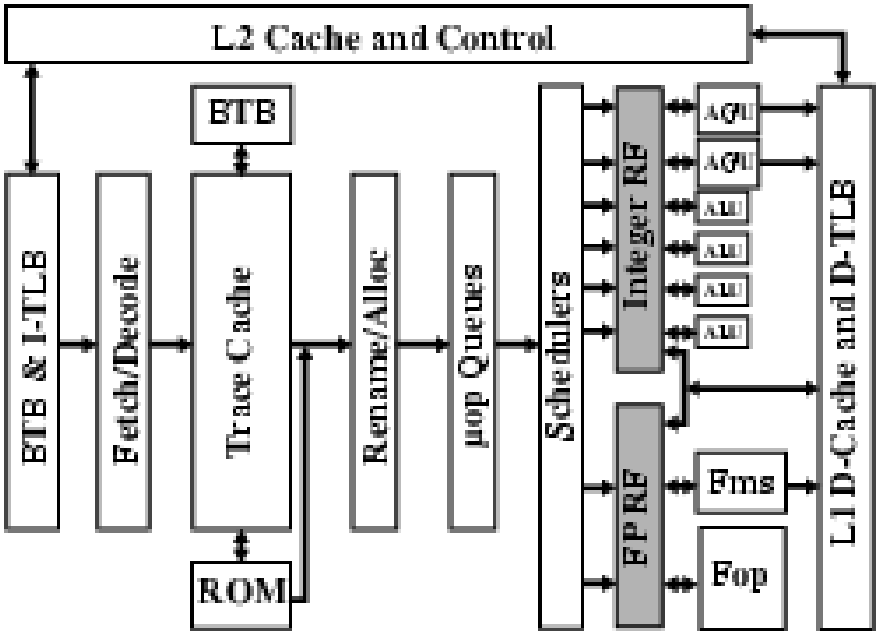


(g) Micro-op scheduling

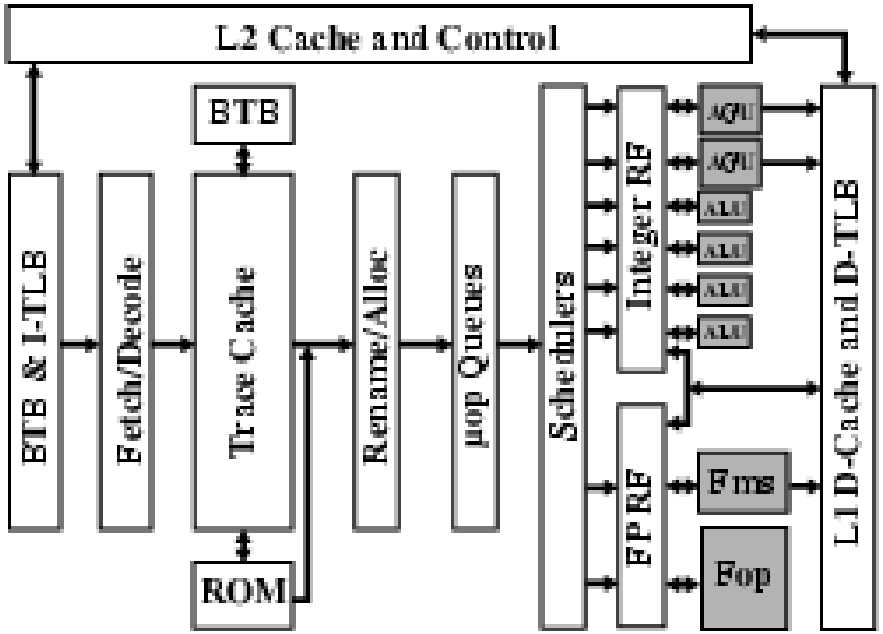


(h) Dispatch

Pentium 4 Pipeline Operation (5)

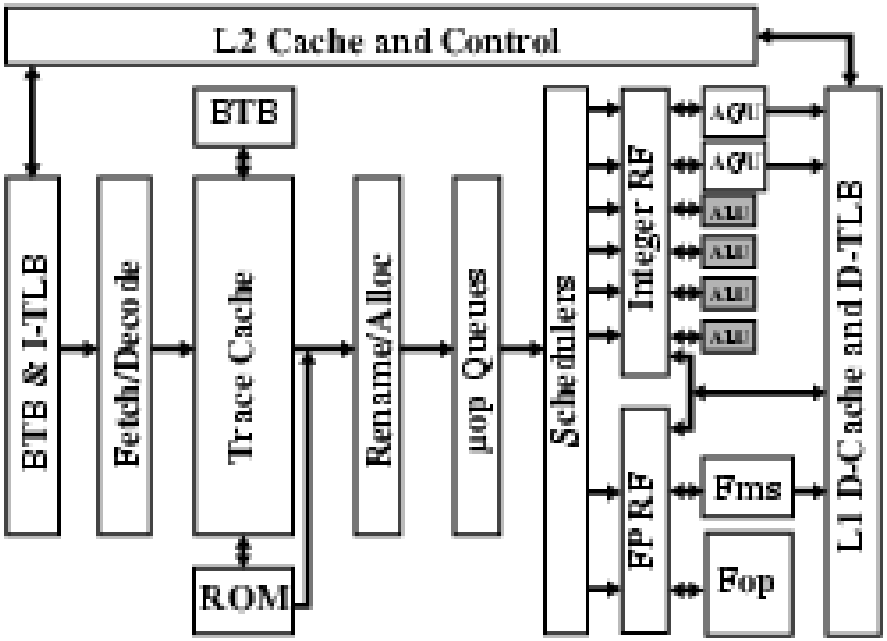


(i) Register file

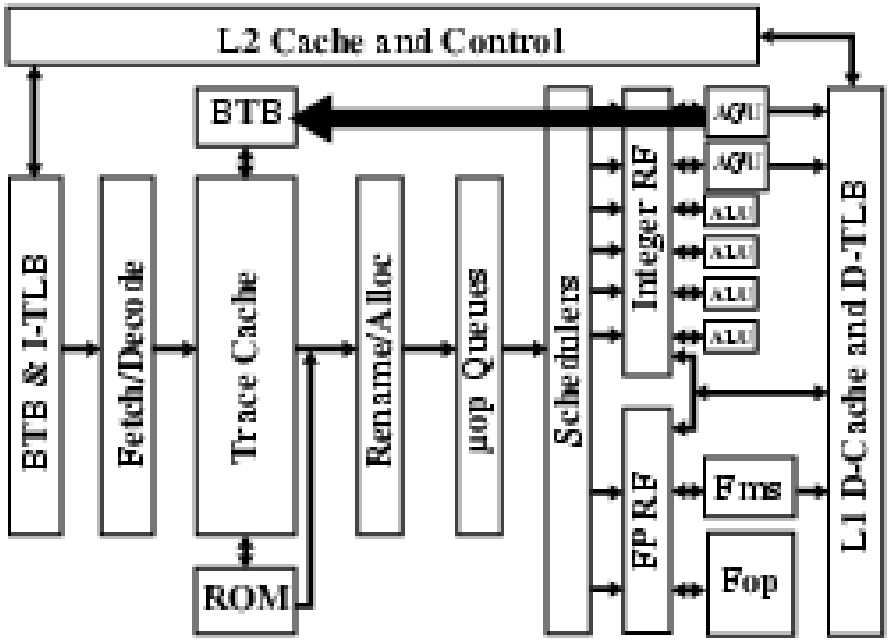


(j) Execute; flags

Pentium 4 Pipeline Operation (6)



(k) Branch check



(l) Branch check result

Tổng kết

- Cần nắm rõ cấu trúc và chức năng chính của CPU
- Kỹ thuật thực hiện các lệnh gối đầu nhau pipeline
- Phân biệt kiến trúc kiểu CISC & RISC