HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÀI GIẢNG MÔN

KIẾN TRÚC MÁY TÍNH

CHƯƠNG 3a – CPU PIPELINE

Giảng viên:

Điện thoại/E-mail:

Bộ môn:

Học kỳ/Năm biên soạn: Học kỳ 2 năm học 2009-2010

NỘI DUNG

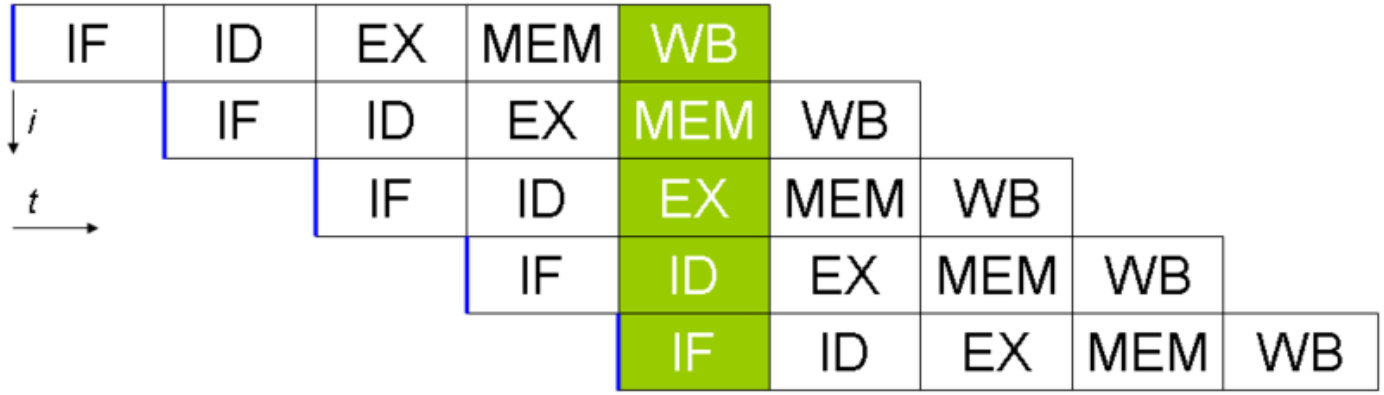
1. Giới thiệu về CPU pipeline
2. Các vấn đề của pipeline
3. Giải quyết vấn đề xung đột tài nguyên
4. Giải quyết vấn đề xung đột dữ liệu
5. Quản lý rẽ nhánh trong pipeline
6. Giới thiệu pipeline của một số CPU
7. Siêu pipeline
8. Câu hỏi ôn tập

3.2.1 Giới thiệu CPU pipeline – Dây chuyền lắp ráp ôtô

- Mỗi dây chuyền lắp ráp được chia thành nhiều công đoạn;
- Nhiều ôtô cùng được lắp ráp trên một dây chuyền;
- Tại mỗi công đoạn, một phần việc được hoàn thành;
- Sau mỗi nhịp thời gian một ôtô hoàn thiện ở cuối dây chuyền và một ôtô bắt đầu hình thành ở đầu dây chuyền.

3.2.1 Giới thiệu CPU pipeline – Nguyên lý

Không pipeline

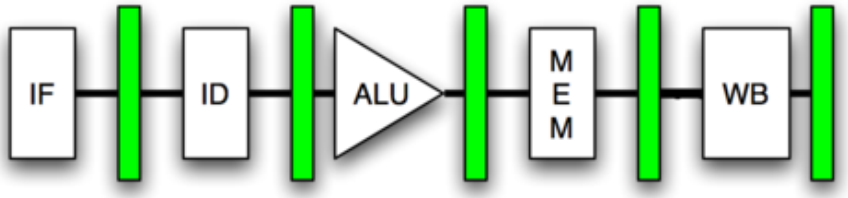


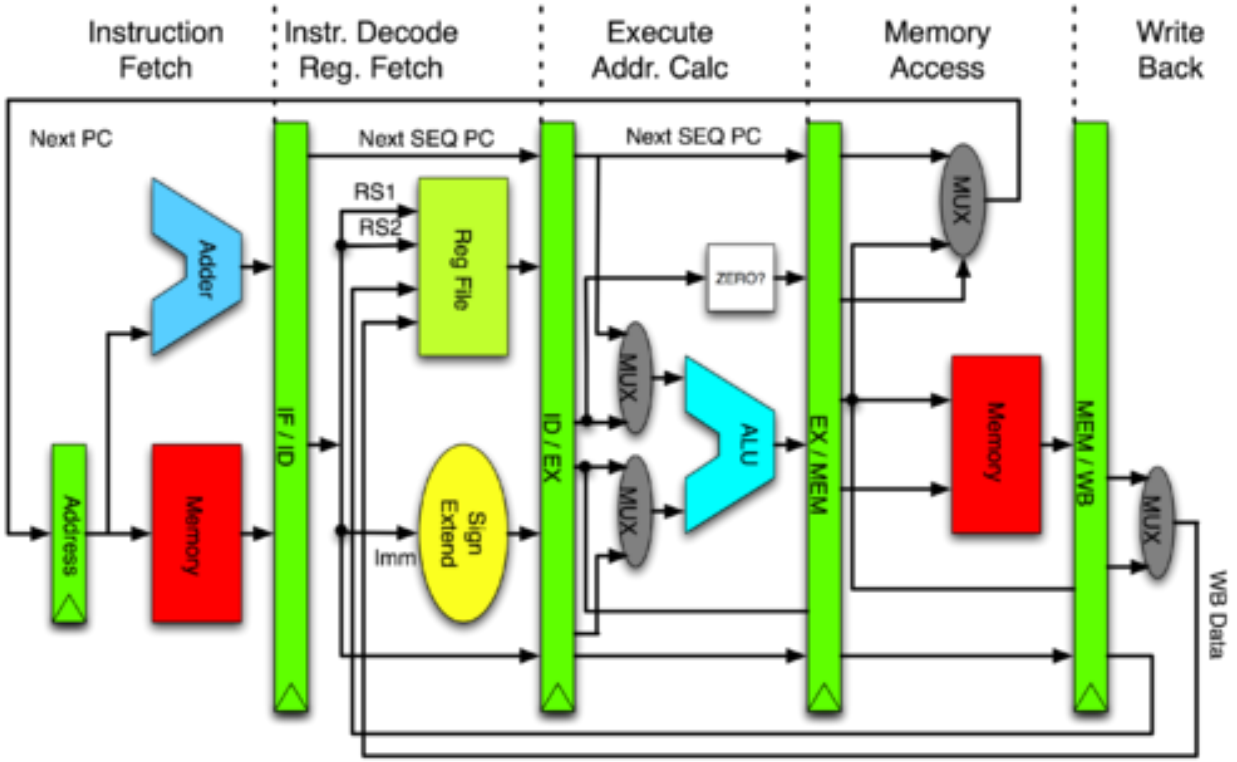
Có pipeline

3.2.1 Giới thiệu CPU pipeline – Nguyên lý

- Việc thực hiện lệnh được chia nhỏ thành các giai đoạn
- 5 giai đoạn của một hệ thống load-store:
  - Instruction Fetch - IF: Đọc lệnh từ bộ nhớ (hoặc cache)
  - Instruction Decode - ID: giải mã lệnh và đọc các toán hạng
  - Execute - EX: thực hiện lệnh; nếu là lệnh truy nhập bộ nhớ: tính toán địa chỉ bộ nhớ
  - Memory Access - MEM: Đọc/ghi bộ nhớ; no-op nếu không truy nhập bộ nhớ
  - Write Back - WB: Ghi kết quả vào các thanh ghi.

3.2.1 Giới thiệu CPU pipeline – Nguyên lý

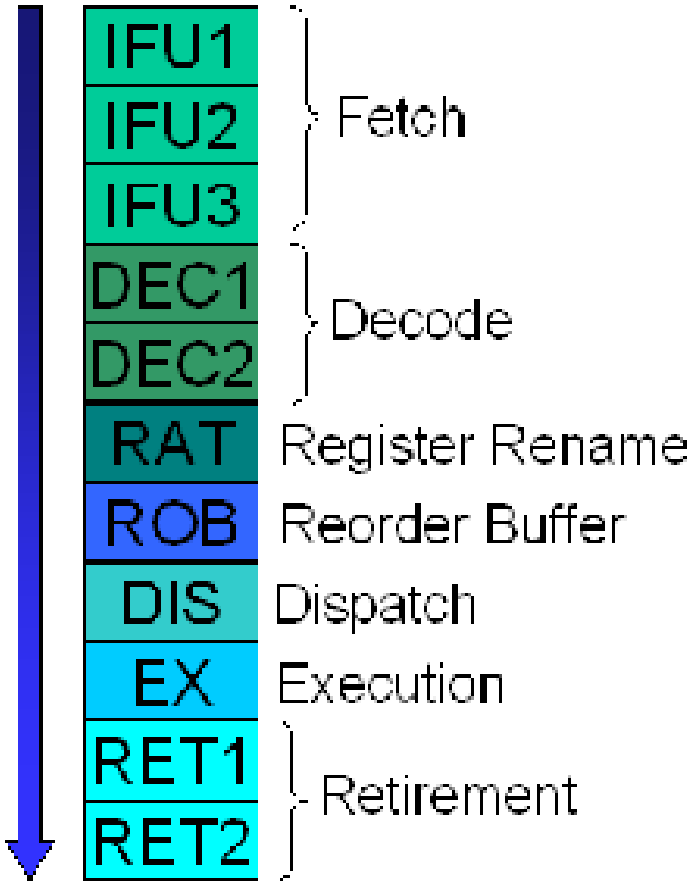




3.2.1 Giới thiệu CPU pipeline – Đặc điểm

- Là dạng xử lý song song ở mức lệnh (instruction level parallelism (ILP));
- Một pipeline là đầy đủ (*fully pipelined*) khi nó luôn tiếp nhận một lệnh mới tại mỗi chu kỳ đồng hồ;
- Ngược lại, một pipeline là không đầy đủ khi có một số chu kỳ trễ trong tiến trình thực hiện;
- Số lượng các giai đoạn (stages) trong pipeline phụ thuộc vào thiết kế vi xử lý:
  - 2,3, 5 giai đoạn (pipeline đơn giản)
  - 14 giai đoạn (PII, PIII)
  - 20-31 giai đoạn (P4)
  - 12-15 giai đoạn (Core)

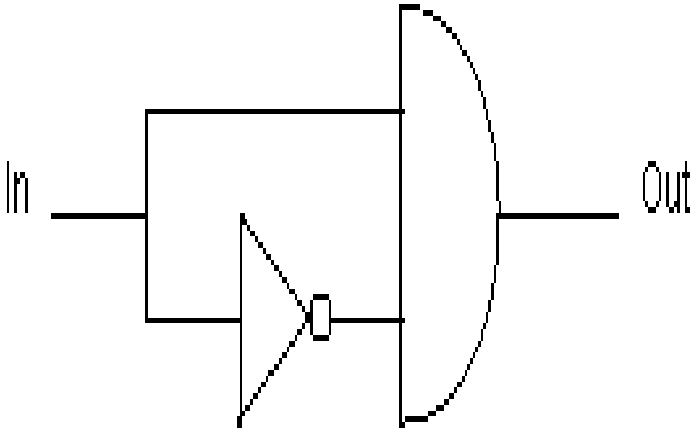
3.2.1 Giới thiệu CPU pipeline – P6 (PIII, M)



3.2.1 Giới thiệu CPU pipeline – Số lượng stages

- Thời gian thực hiện mỗi giai đoạn
  - Các stages nên có thời gian t/h bằng nhau
  - Các stages chậm cần được tách nhỏ
- Các vấn đề liên quan đến tài nguyên
  - Điều gì xảy ra khi hai giai đoạn đọc lệnh và đọc toán hạng đều truy nhập bộ nhớ?
  - Điều gì xảy ra khi hai giai đoạn đọc lệnh và thực hiện lệnh (tính địa chỉ bộ nhớ) đều truy nhập PC?
- Pipeline dài bao nhiêu là tốt?
  - Về nguyên tắc: càng nhiều stages, hiệu quả càng cao
  - Pipeline dài nếu bị trống rỗng vì một lý do nào đó sẽ tốn nhiều thời gian để điền đầy.

3.2.2 Các vấn đề của pipeline – Logic gate hazard



Logic Gate Hazard

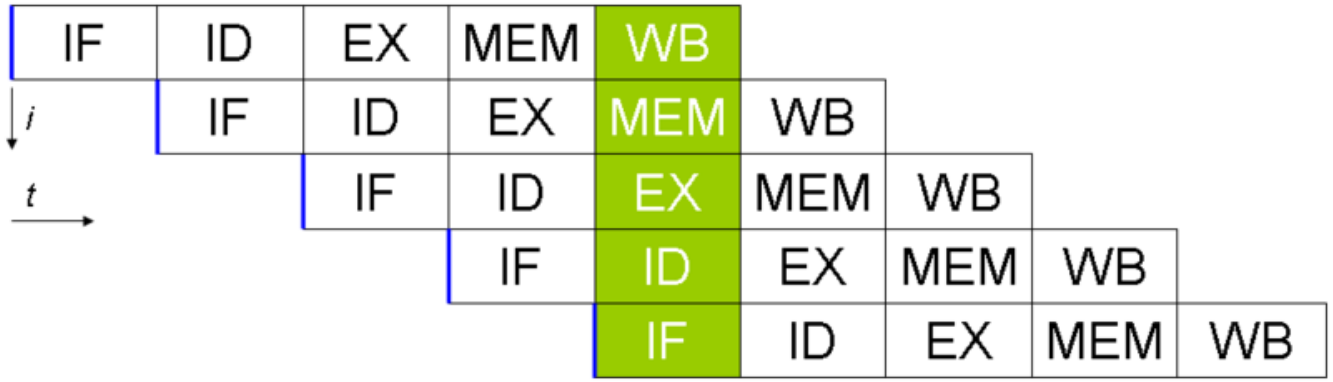
- Giá trị ra mong đợi (theo thiết kế) luôn là 0 (false)
- Tuy nhiên, trong một số thời điểm giá trị ra là 1 (true) → Hazard (không theo thiết kế).

3.2.2 Các vấn đề của pipeline

- Vấn đề xung đột tài nguyên (resource conflicts)
  - Xung đột truy nhập bộ nhớ
  - Xung đột truy nhập các thanh ghi
- Tranh chấp dữ liệu (Data hazards):
  - Vấn đề *read after write hazard* (RAW)
- Các lệnh rẽ nhánh (Branch instructions)
  - Không điều kiện
  - Có điều kiện
  - Gọi thực hiện và trở về từ chương trình con

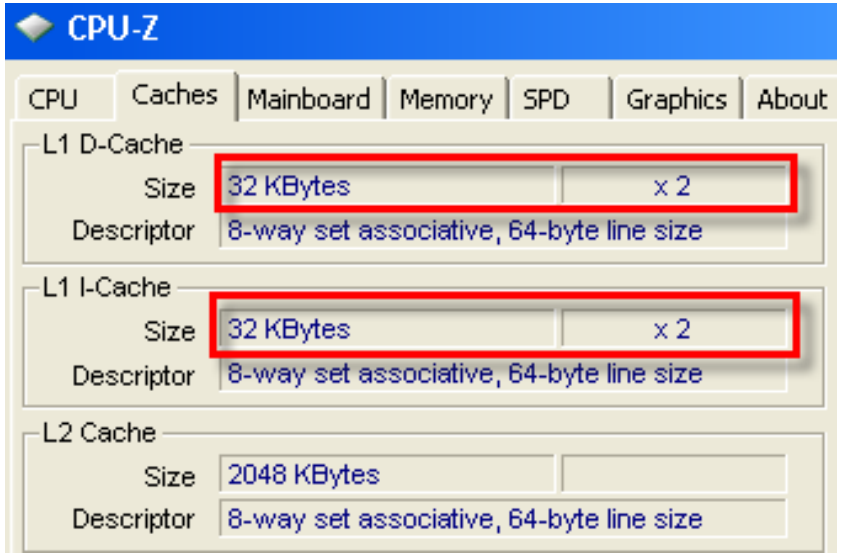
3.2.3 Xung đột tài nguyên

- Không đủ tài nguyên phục vụ CPU;
- Ví dụ: nếu bộ nhớ chỉ hỗ trợ một truy nhập tại mỗi thời điểm và nếu tại cùng một thời điểm, pipeline yêu cầu hai truy nhập bộ nhớ (đọc lệnh – tại giai đoạn IF và đọc dữ liệu – tại giai đoạn ID) → nảy sinh xung đột.



3.2.3 Xung đột tài nguyên

- Giải pháp: Thêm tài nguyên hoặc nâng cao năng lực phục vụ của tài nguyên:
  - Memory/Cache: hỗ trợ nhiều truy nhập tại một thời điểm;
  - Chia cache thành 2 phần: I-Cache và D-Cache để cải thiện khả năng truy nhập.



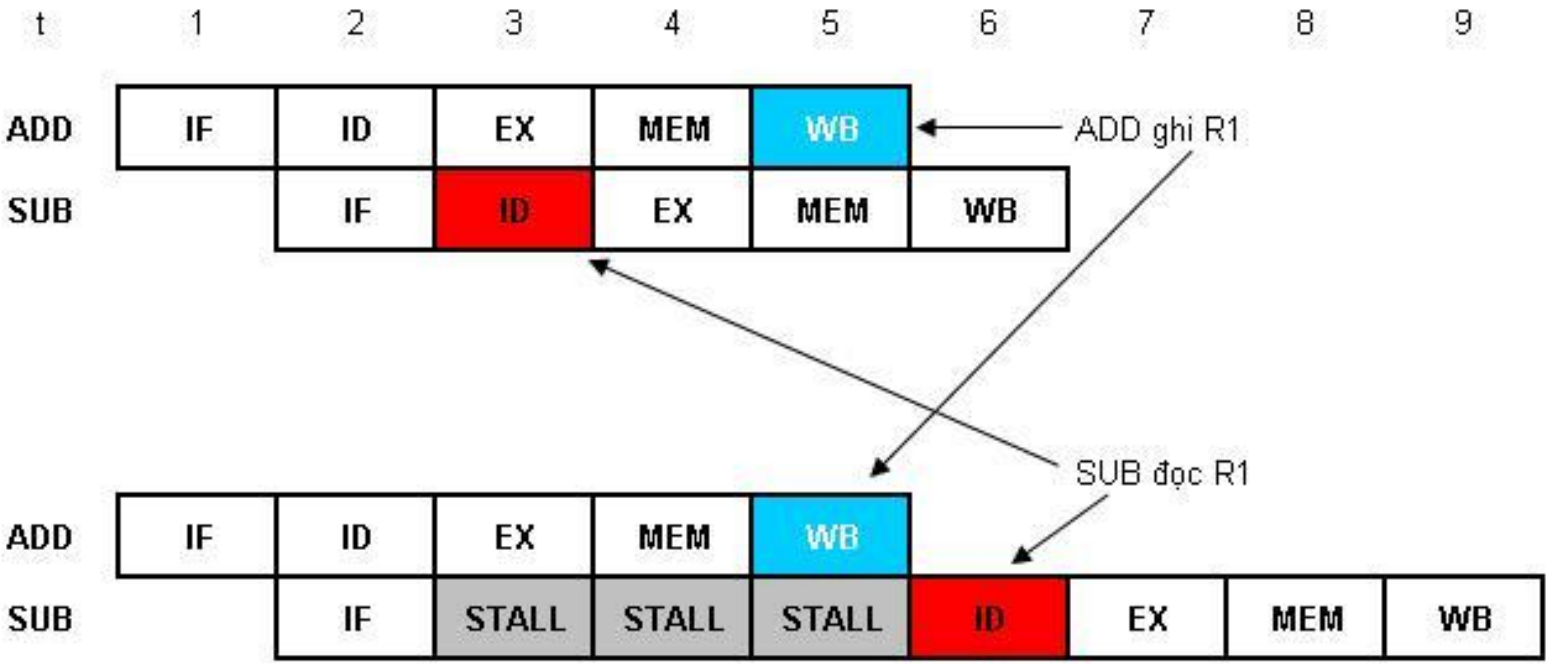
3.2.4 Xung đột dữ liệu RAW

- Xem xét hai lệnh:

```
ADD R1, R1, R3 ;R1 = R1+R3
MOVE R5, #10
SUB R4, R1, R2 ;R4 = R1-R2
```

- Lệnh SUB sử dụng kết quả của ADD – có sự phụ thuộc dữ liệu giữa hai lệnh
- SUB đọc R1 trong giai đoạn 2 (ID), còn ADD ghi kết quả trong giai đoạn 5 (WB)
  - SUB đọc giá trị cũ của R1 trước khi ADD lưu giá trị mới của R1.

3.2.4 Xung đột dữ liệu RAW

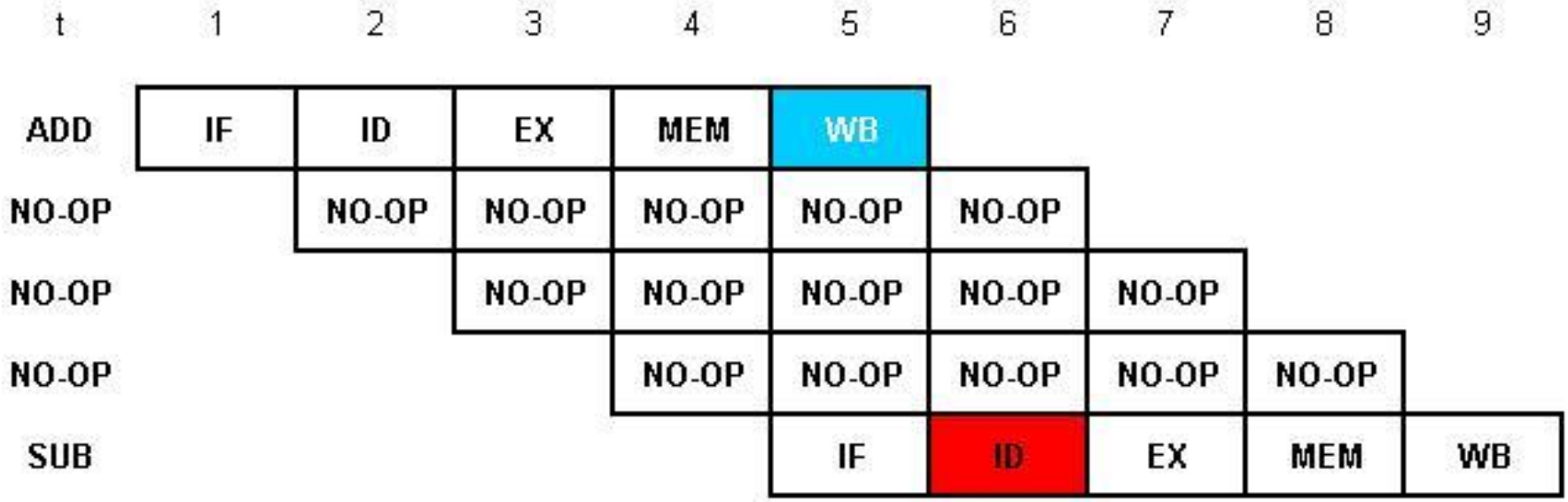


- ADD R1, R1, R3 ;R1 ← R1+R3
- SUB R4, R1, R2 ;R4 ← R1-R2

3.2.4 Xung đột dữ liệu RAW – Hướng khắc phục

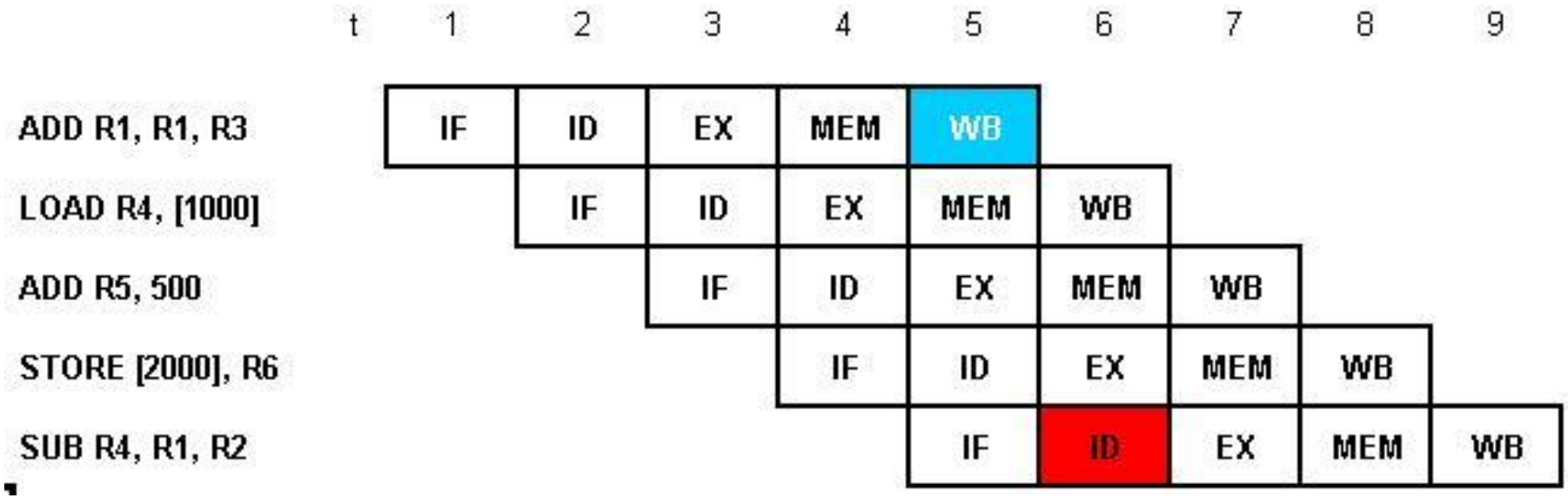
- Nhận dạng RAW hazard khi nó diễn ra
- Khi RAW hazard xảy ra, tạm dừng (stall) pipeline cho đến khi lệnh phía trước hoàn tất giai đoạn WB.
- Có thể sử dụng compiler để nhận dạng RAW và:
  - Chèn thêm các lệnh NO-OP vào giữa các lệnh có thể gây ra RAW;
  - Thay đổi trật tự các lệnh trong chương trình và chèn các lệnh độc lập vào giữa các lệnh có thể gây ra RAW;
- Sử dụng phần cứng để nhận dạng RAW và dự đoán trước giá trị dữ liệu phụ thuộc.

3.2.4 Xung đột dữ liệu RAW – Hướng khắc phục



Lùi thời điểm thực hiện SUB bằng cách chèn thêm 3 lệnh NO-OP

3.2.4 Xung đột dữ liệu RAW – Hướng khắc phục

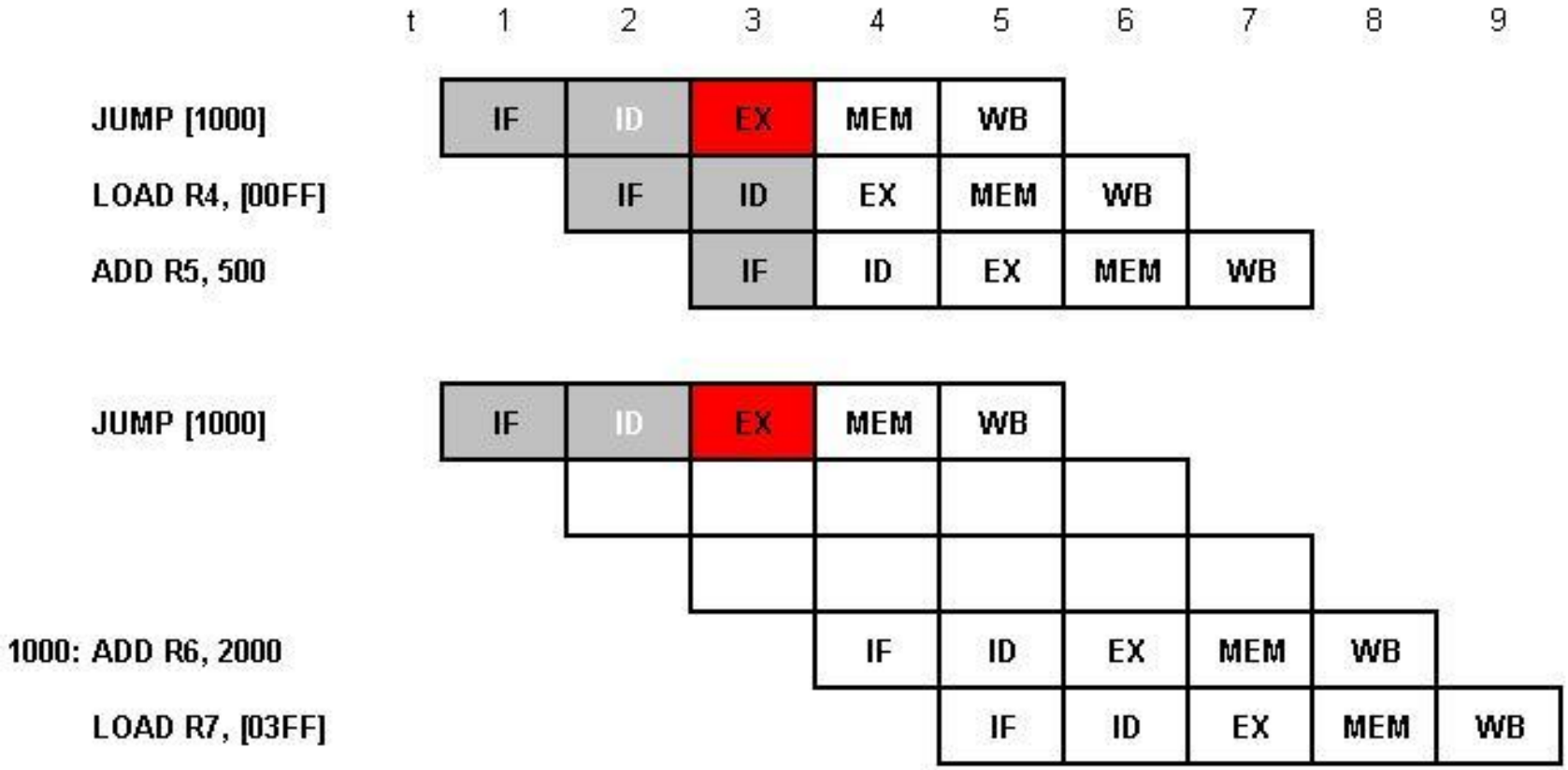


Chèn thêm 3 lệnh độc lập dữ liệu vào
giữa 2 lệnh ADD và SUB có thể sinh ra RAW

3.2.5 Quản lý rẽ nhánh trong pipeline

- Tỷ lệ các lệnh rẽ nhánh trong chương trình khoảng 10-30%. Lệnh rẽ nhánh gây ra:
  - Ngắt quãng quá trình thực hiện bình thường của chương trình;
  - Làm cho pipeline trống rỗng nếu không có biện pháp phòng ngừa/ngăn chặn.
- Với các VXL có pipeline dài như P4 (31 stages) và nhiều pipeline chạy song song, vấn đề lệnh rẽ nhánh càng trở nên phức tạp:
  - Phải đẩy toàn bộ các lệnh đang thực hiện ở các ống khi gặp lệnh rẽ nhánh;
  - Nạp mới các lệnh từ địa chỉ rẽ nhánh vào pipeline. Tiêu tốn nhiều thời gian để điền đầy pipeline.

3.2.5 Quản lý rẽ nhánh trong pipeline

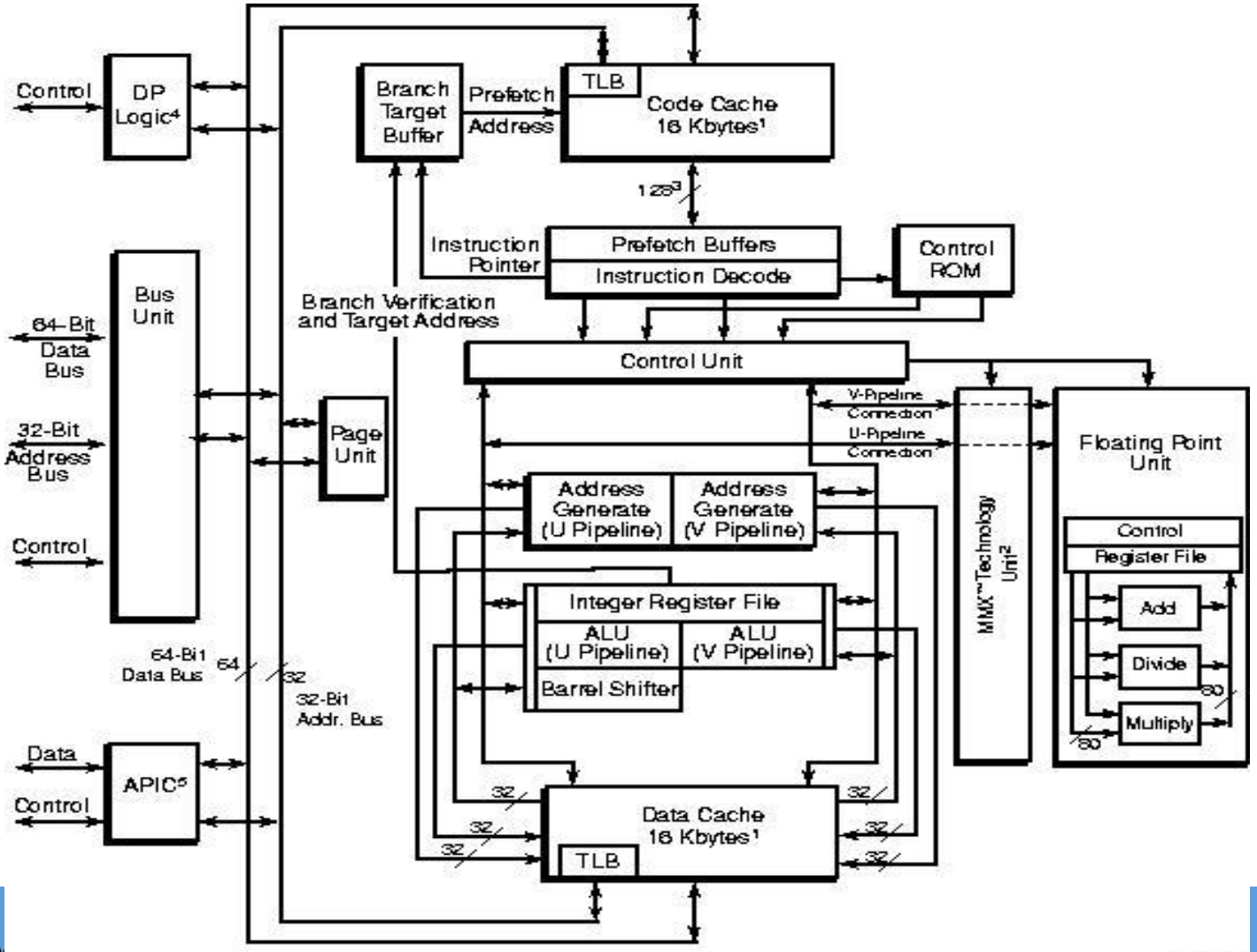


Khi gặp lệnh rẽ nhánh, các lệnh kế tiếp đã nạp bị đẩy ra khỏi pipeline và sau đó các lệnh từ địa chỉ đích được nạp vào

3.2.5 Quản lý rẽ nhánh – Các giải pháp

1. Branch Targets (đích rẽ nhánh)
2. Conditional Branches (rẽ nhánh có điều kiện)
   - Delayed Branching (làm chậm rẽ nhánh)
   - Branch Prediction (dự đoán rẽ nhánh)

3.2.5 Quản lý rẽ nhánh – PIII Branch Targets



3.2.5 Quản lý rẽ nhánh – Branch Targets

- Khi một lệnh rẽ nhánh được thực thi, lệnh tiếp theo được nạp là lệnh ở địa chỉ đích (target), không phải lệnh kết tiếp lệnh nhảy.

```
JUMP  <Address>
ADD R1, R2

Address: SUB R3, R4
```

3.2.5 Quản lý rẽ nhánh – Branch Targets

- Khi một lệnh rẽ nhánh được thực thi, lệnh tiếp theo được nạp là lệnh ở địa chỉ đích (target), không phải lệnh kết tiếp lệnh nhảy.

```
JUMP  <Address>
ADD R1, R2

Address: SUB R3, R4
```

3.2.5 Quản lý rẽ nhánh – Branch Targets

- Lệnh rẽ nhánh được nhận biết ở giai đoạn ID, vậy có thể biết trước chúng bằng cách giải mã sớm.
- Sử dụng *Branch Target Buffer* (BTB) để lưu vết của các lệnh rẽ nhánh đã được thực thi:
  - Các địa chỉ đích của các lệnh rẽ nhánh
  - Lệnh đích của các lệnh rẽ nhánh
- Nếu các lệnh rẽ nhánh được tái sử dụng (trong vòng lặp):
  - Đ/c đích rẽ nhánh của chúng có thể được sử dụng mà không cần tính toán lại;
  - Các lệnh rẽ nhánh có thể được sử dụng ngay mà không cần phải nạp lại từ bộ nhớ.

  ➔ Việc này thực hiện được do địa chỉ và lệnh đích rẽ nhánh thường không thay đổi.

3.2.5 Quản lý rẽ nhánh – Rẽ nhánh có điều kiện

- Việc quản lý các lệnh rẽ nhánh có điều kiện (conditional branches) phức tạp hơn do:
  - Có 2 lệnh rẽ nhánh đích phải lựa chọn;
  - Không thể xác định chính xác lệnh đích rẽ nhánh cho đến khi lệnh rẽ nhánh thực hiện.
  - Branch Target Buffer cũng không thể hạn chế được trễ do ta vẫn phải chờ cho đến khi biết được đích của lệnh rẽ nhánh.
- Giải pháp:
  - Delayed branching: làm chậm rẽ nhánh
  - Branch prediction: dự đoán rẽ nhánh

3.2.5.1 Rẽ nhánh có điều kiện - Delayed branching

- Ý tưởng chính: lệnh rẽ nhánh sẽ không gây ra sự rẽ nhánh tức thì mà được làm "trễ" một số chu kỳ, phụ thuộc vào chiều dài của pipeline.
- Đặc điểm của Delayed branching:
  - Làm việc tốt nhất trên các VXL RISC – các lệnh có thời gian thực hiện ngang nhau;
  - Pipeline ngắn (thường là 2 stages)
  - Lệnh ngay sau lệnh rẽ nhánh luôn được thực hiện, không phụ thuộc vào kết quả của lệnh rẽ nhánh.
- Cách thực hiện:
  - Sử dụng compiler để chèn thêm lệnh NO-OP vào sau lệnh rẽ nhánh, hoặc;
  - Chuyển 1 lệnh độc lập từ trước ra ngay sau lệnh rẽ nhánh.

3.2.5.1 Rẽ nhánh có điều kiện - Delayed branching

- Cho dãy lệnh:

```
ADD R2, R3, R4
CMP R1,0
JNE somewhere
```

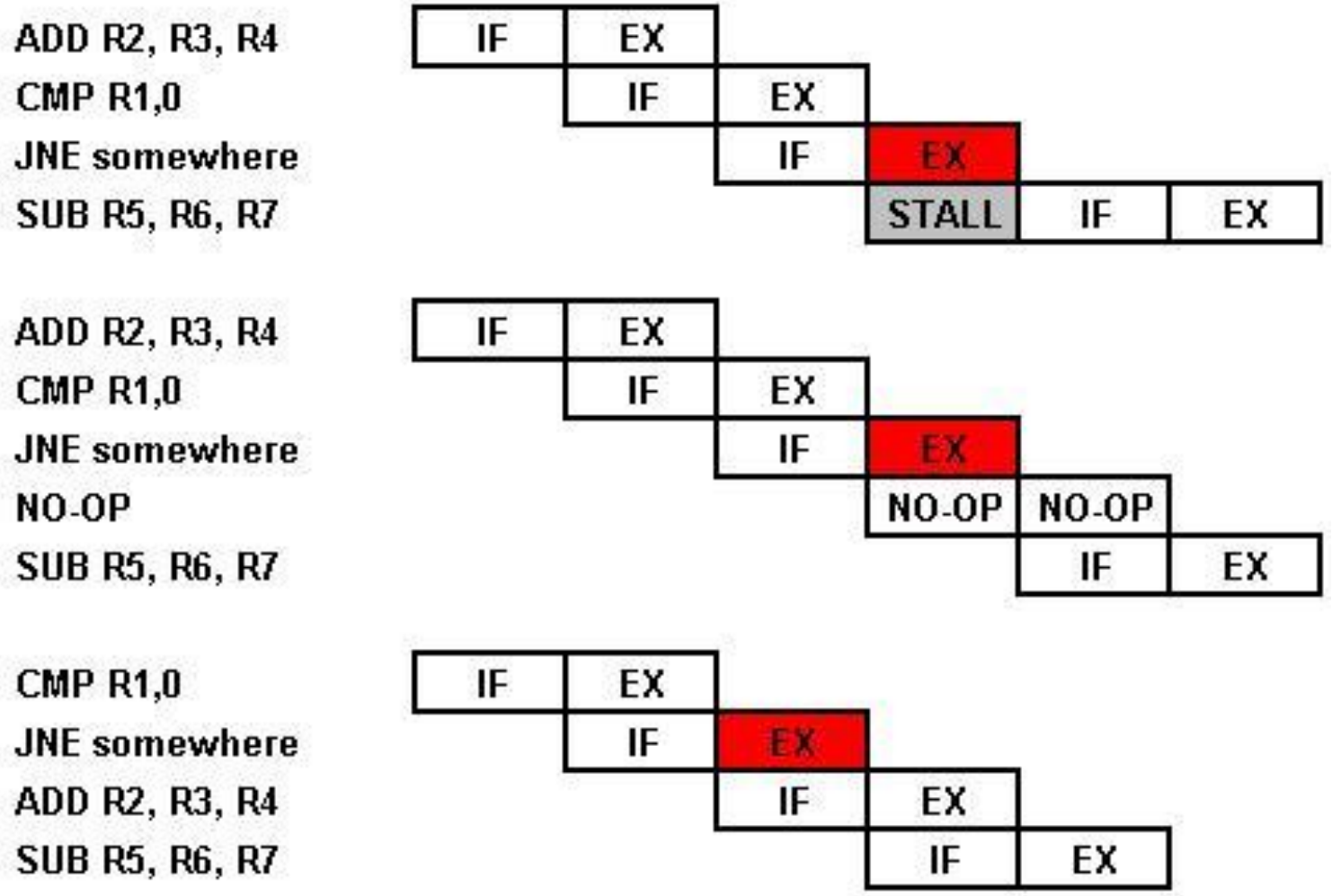
- Thêm NO-OP sau lệnh rẽ nhánh

```
ADD R2, R3, R4
CMP R1,0
JNE somewhere
NO-OP
```

- Chuyển lệnh độc lập phía trước ra ngay sau lệnh rẽ nhánh:

```
CMP R1,0
JNE somewhere
ADD R2, R3, R4
```

3.2.5.1 Rẽ nhánh có điều kiện - Delayed branching



3.2.5.1 Rẽ nhánh có điều kiện - Delayed branching

- Dễ thực hiện thông qua tối ưu trong hoá trình biên dịch, không đòi hỏi phần cứng đặc biệt;
- Giảm hiệu năng khi pipeline dài nếu chỉ chèn thêm NO-OP. Ví dụ nếu pipeline có 5 stages -> đòi hỏi 5 lệnh NO-OP.
- Thay NO-OP bằng lệnh độc lập có thể giảm ~70% NO-OP.
- Tăng độ phức tạp của mã chương trình.
- Đòi hỏi người lập trình và người viết trình biên dịch phải có hiểu biết sâu về kiến trúc pipeline của các VXL -> đây là một hạn chế lớn.
- Giảm tính khả chuyển của mã chương trình, do khi kiến trúc VXL thay đổi -> viết lại hoặc dịch lại.

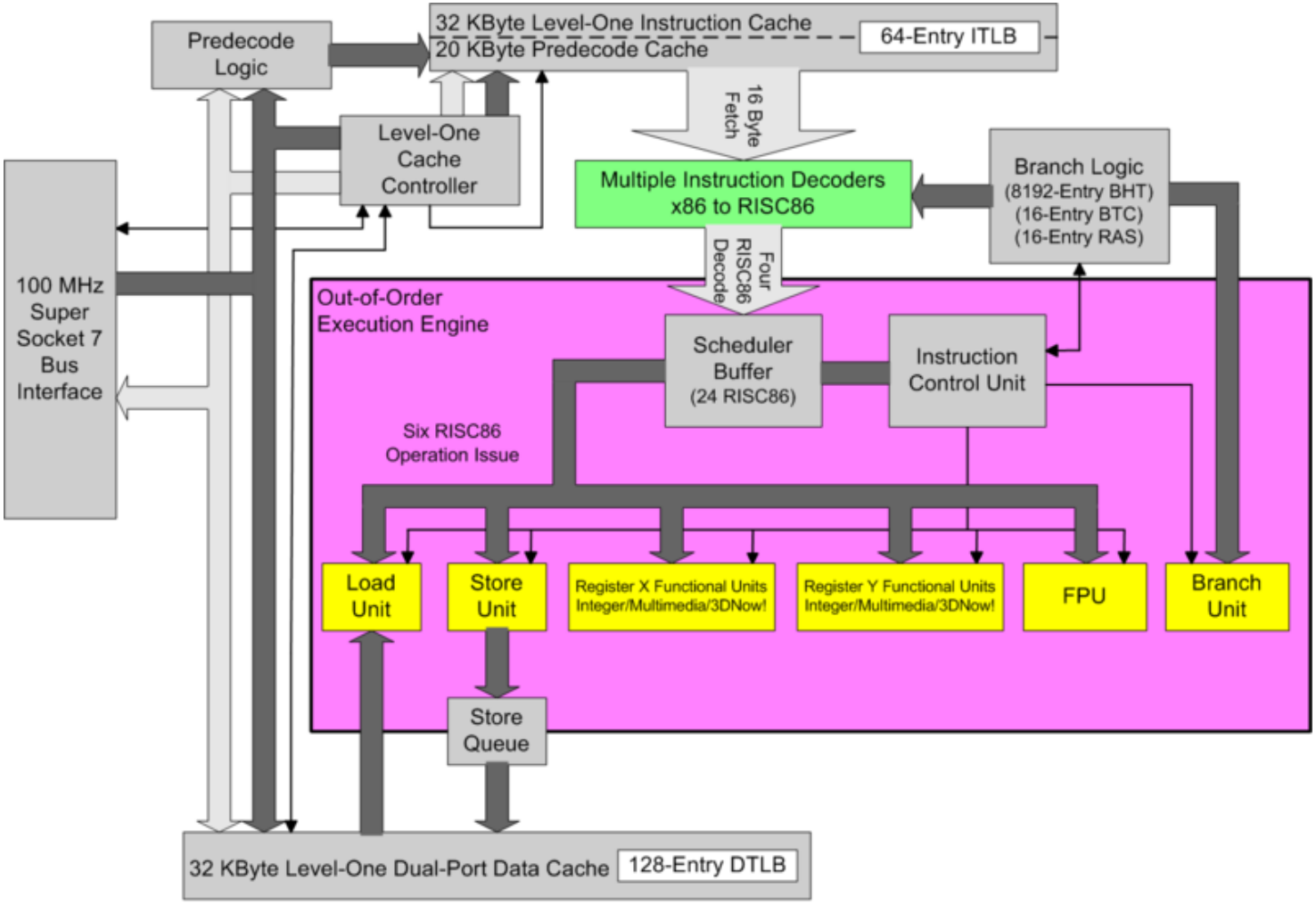
3.2.5.1 Rẽ nhánh có điều kiện - Branch prediction

- Ý tưởng: có thể dự đoán lệnh đích của lệnh rẽ nhánh có điều kiện.
  - Đoán đúng: giúp tăng hiệu năng
  - Đoán sai: đẩy các lệnh đã nạp phía sau lệnh rẽ nhánh, nạp lệnh đích rẽ nhánh.
- Trường hợp xấu nhất:
  - 50% đoán đúng và 50% đoán sai
  - Kết quả không xấu hơn không dự đoán

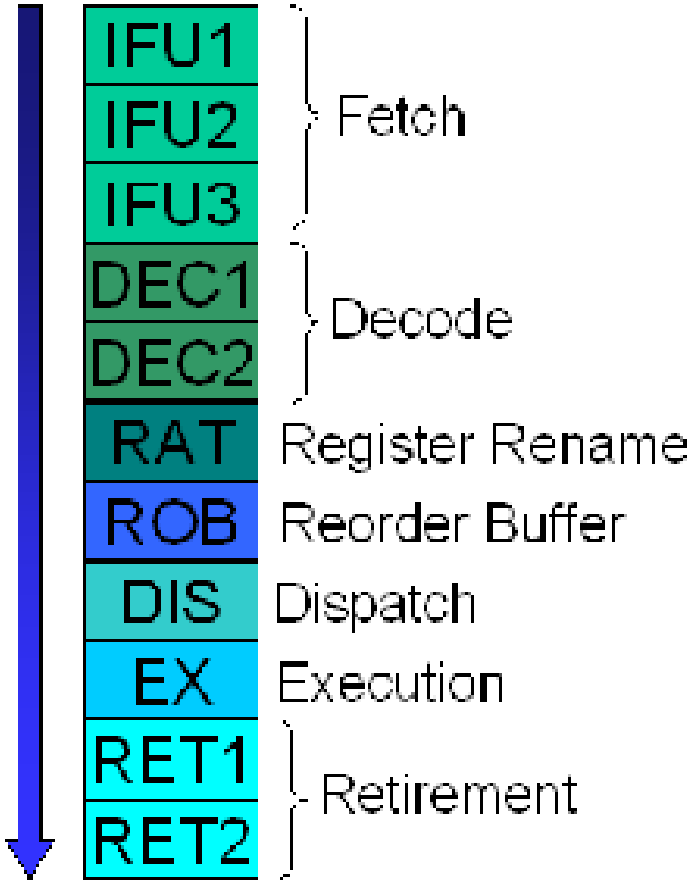
3.2.5.1 Rẽ nhánh có điều kiện - Branch prediction

- Các cơ sở để dự đoán rẽ nhánh:
  - Các lệnh nhảy ngược:
    - Thường là một phần của một vòng lặp;
    - Các vòng lặp thường được thực hiện nhiều lần
    - Lần lặp cuối có thể sai
  - Các lệnh nhảy xuôi khó dự đoán hơn:
    - Có thể là lệnh kết thúc vòng lặp
    - Có thể lệnh nhảy dựa trên một điều kiện

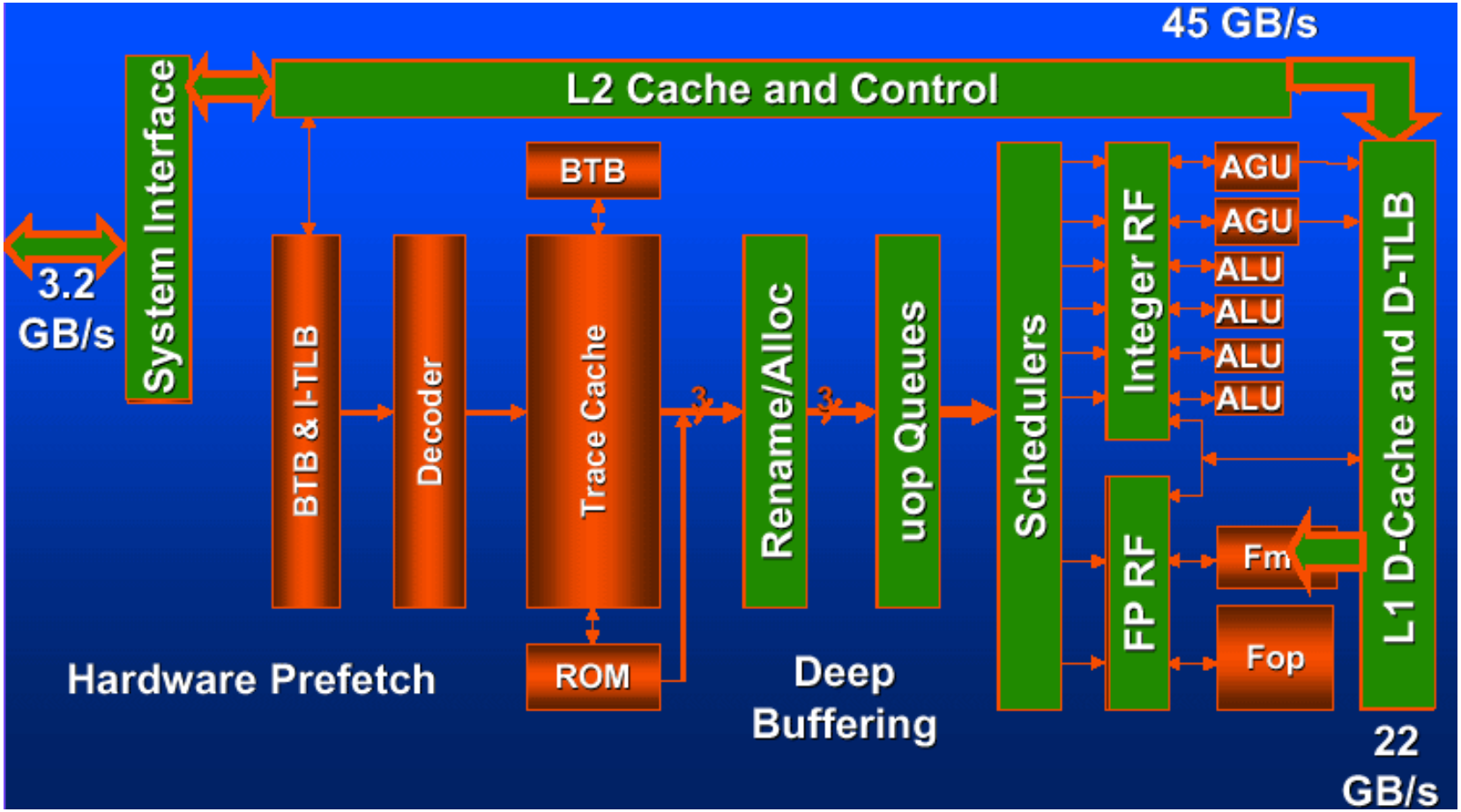
3.2.6 AMD K6-2 pipeline



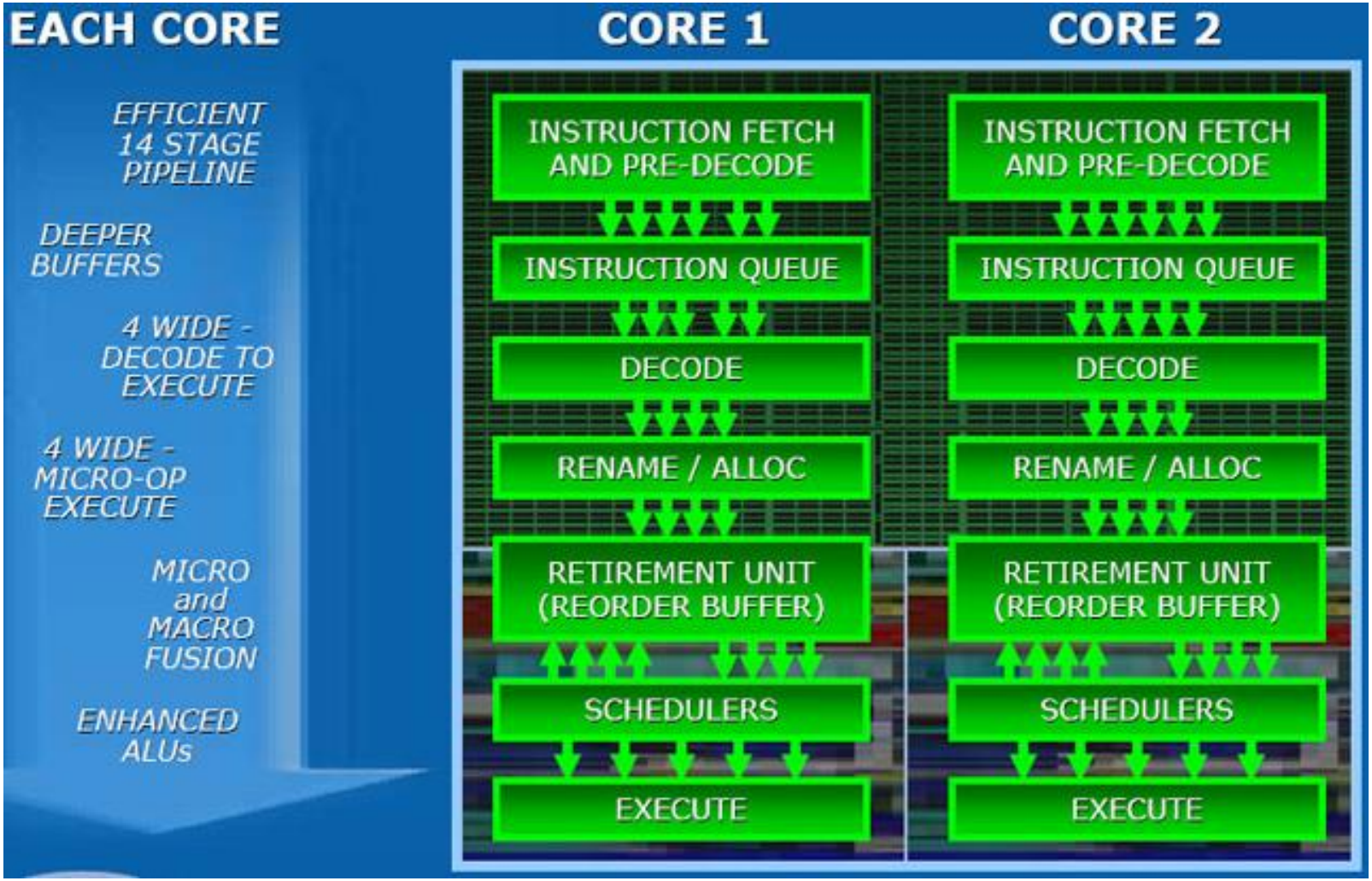
3.2.6 Intel P6 (PIII, M) pipeline



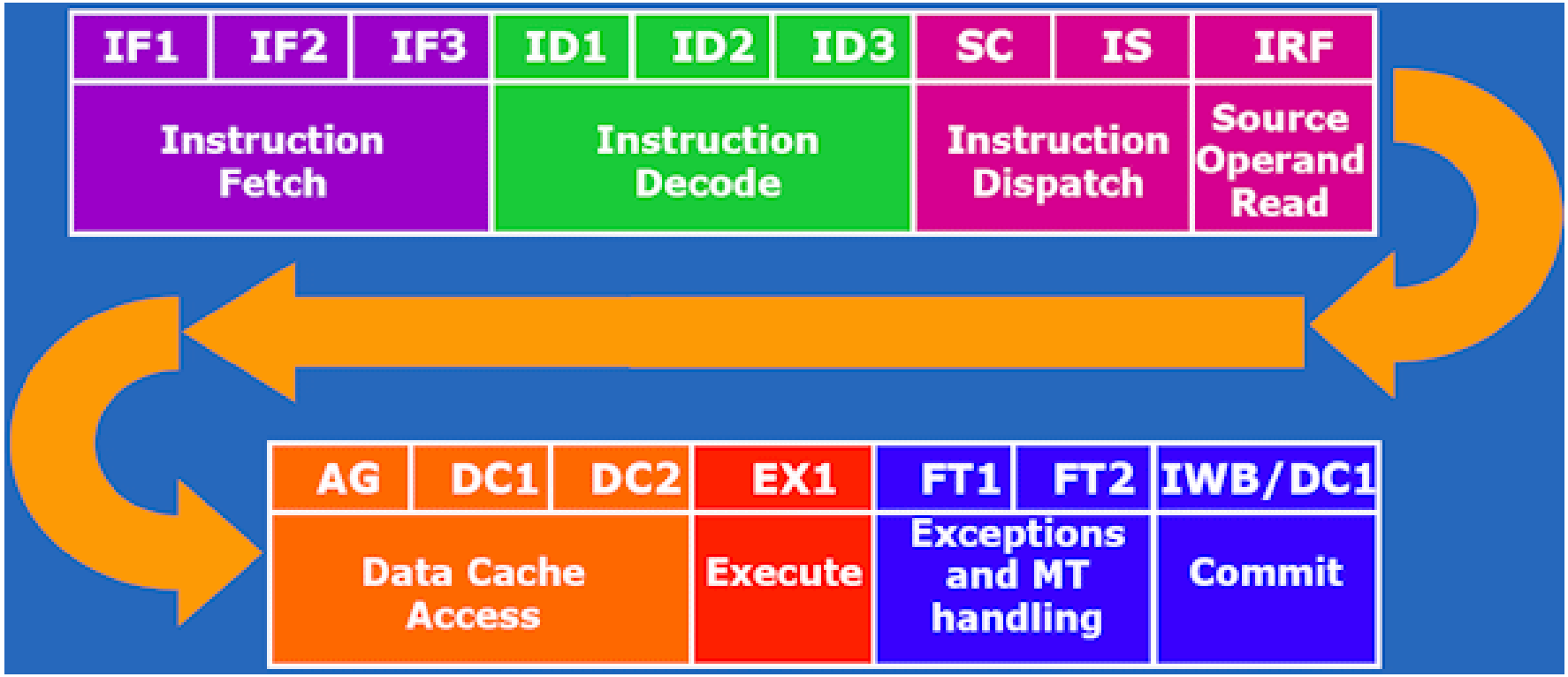
3.2.6 Intel P4 pipeline



3.2.6 Intel Core 2 Duo pipeline



3.2.6 Intel Atom 16-stage pipeline

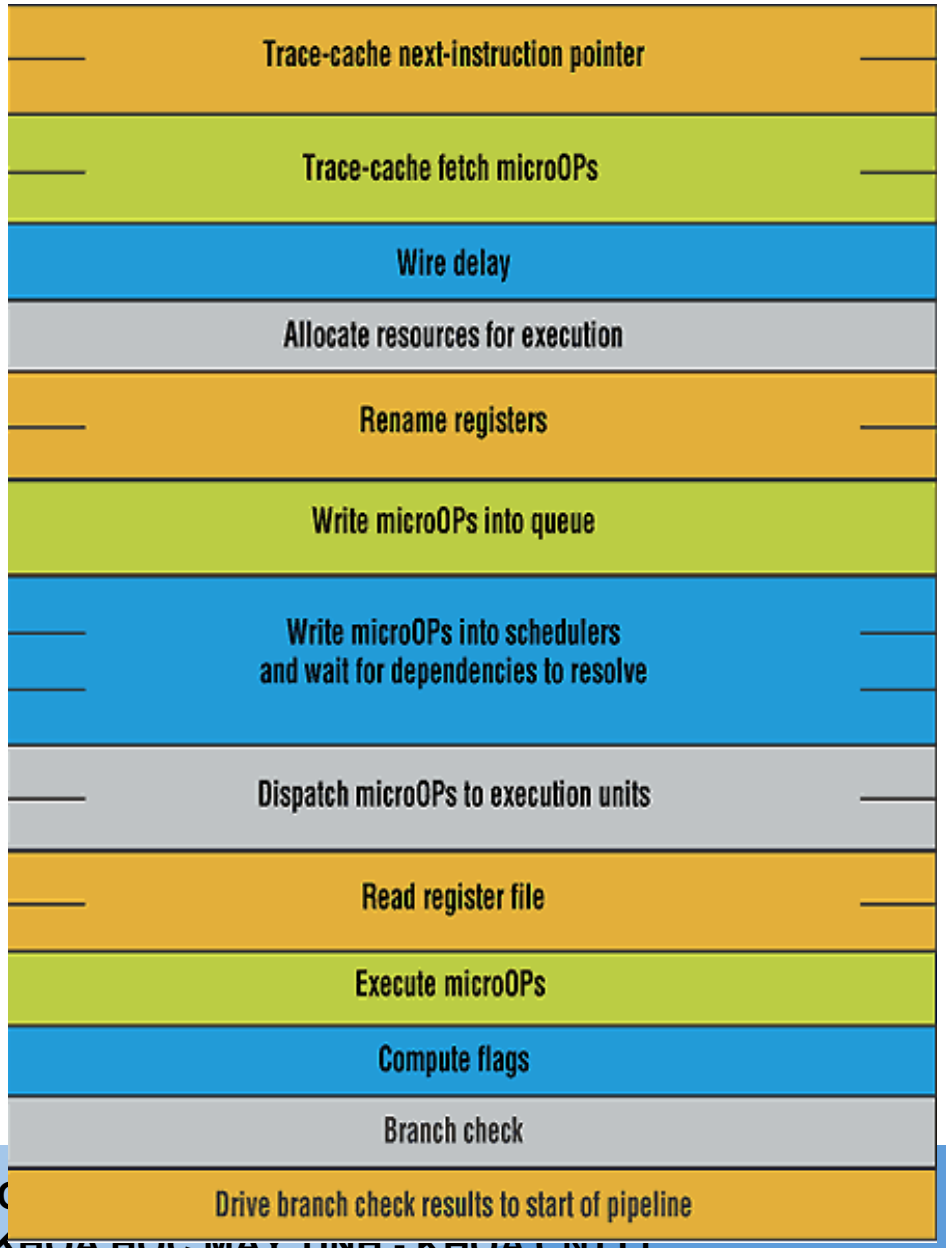


3.2. 7 Siêu pipeline

- Siêu pipeline (Superpipelining) là kỹ thuật cho phép:
  - Tăng độ sâu của ống lệnh
  - Tăng tốc độ đồng hồ
  - Giảm thời gian trễ cho từng giai đoạn thực hiện lệnh
- VD: nếu giai đoạn thực hiện lệnh bởi ALU kéo dài → chia thành một số giai đoạn nhỏ → giảm thời gian chờ cho các giai đoạn ngắn.
- Pentium 4 siêu ống với 20 giai đoạn:

3.2. 7 Siêu pipeline

Pentium 4 siêu ống với 20 giai đoạn



3.2.8 Câu hỏi ôn tập

1. CPU pipeline là gì và các đặc điểm?
2. Các vấn đề của pipeline
3. Vấn đề xung đột dữ liệu trong pipeline và giải pháp khắc phục
4. Vấn đề rẽ nhánh trong pipeline và giải pháp khắc phục