1. a) Cơ chế ống lệnh (pipeline) của CPU thường gặp phải những vấn đề gì?
2. Nêu 1 hướng giải quyết xung đột dữ liệu trong pipeline khi thực hiện đoạn chương trình sau:

ADD R1, R2, R3 ;R1 <== R2+R3

ADD R4, R4, #300 ;R4 <== R4+300

CMP R1, #100 ;so sánh R1 với 100

SUB R5, #2000 ;R5 <== R5 - 2000

Biết rằng mỗi lệnh được chia thành 5 giai đoạn trong pipeline: Đọc lệnh (IF), giải mã & đọc toán hạng (ID), thực hiện (EX), truy nhập bộ nhớ (MEM) và lưu kết quả (WB).

1. Nêu 1 hướng giải quyết xung đột dữ liệu trong pipeline khi thực hiện đoạn chương trình sau:

ADD R4, R4, #300 ;R4 <== R4+300

ADD R1, R1, R3 ;R1 <== R1+R3

SUB R5, #2000 ;R5 <== R5 + 2000

SUB R1, R1, #100 ;R1 <== R1 - 100

Biết rằng mỗi lệnh được chia thành 5 giai đoạn trong pipeline: Đọc lệnh (IF), giải mã & đọc toán hạng (ID), thực hiện (EX), truy nhập bộ nhớ (MEM) và lưu kết quả (WB).

LG

1. Các vấn đề thường gặp với cơ chế ống lệnh (pipeline) của CPU:

* Vấn đề xung đột tài nguyên (resource conflict):

+ Xung đột truy cập bộ nhớ

+ Xung đột truy cập thanh ghi

* Xung đột/ tranh chấp dữ liệu (data hazard) (hầu hết là RAW hay Read After Write Hazard)
* Vấn đề với các lệnh rẽ nhánh (Branch Instruction):

+ Không điều kiện

+ Có điều kiện

+ Gọi thực hiện và trở về từ chương trình con

1. Ta có thể chèn thêm lệnh NO-OP, kết hợp sắp xếp lại các lệnh để giải quyết xung đột pipeline khi thực hiện chương trình:

ADD R1, R2, R3 ;R1 <== R2+R3

ADD R4, R4, #300 ;R4 <== R4+300

SUB R5, #2000 ;R5 <== R5 + 2000

NO-OP

CMP R1, #100 ;so sánh R1 với 100

1. Ta có thể chèn thêm lệnh NO-OP, kết hợp sắp xếp lại các lệnh để giải quyết xung đột pipeline khi thực hiện chương trình:

ADD R1, R1, R3 ;R1 <== R1+R3

ADD R4, R4, #300 ;R4 <== R4+300

SUB R5, #2000 ;R5 <== R5 + 2000

NO-OP

SUB R1, R1, #100 ;R1 <== R1 - 100

1. Cho đoạn chương trình sau (R1, R2 là các thanh ghi và lệnh quy ước theo dạng LỆNH <ĐÍCH> <GỐC>):

(1) STORE –100(R2), R1

(2) LOAD R1, (00FF)

(3) COMPARE R3, R4

(4) JUMP-IF-EQUAL Label

(5) ADD R3, R4

(6) ADD R2, 2

(7) Label:

1. Nêu hướng giải quyết xung đột dữ liệu trong pipeline khi thực hiện đoạn chương trình trên biết mỗi lệnh được chia thành 5 giai đoạn (IF, ID, EX, MEM và WB)
2. Giả thiết R3 != R4 và mỗi giai đoạn thực hiện lệnh đều thực hiện trong thời gian là 0.1ns, so sánh thời gian CPU chạy hết 6 lệnh đầu tiên trong trường hợp không sử dụng cơ chế pipeline và có sử dụng cơ chế pipeline trong ý 2.

LG

1. Có 2 hướng có thể sử dụng để giải quyết xung đột dữ liệu trong pipeline khi thực hiện đoạn chương trình trên:

* Chèn lệnh NO-OP:

STORE –100(R2), R1

LOAD R1, (00FF)

COMPARE R3, R4

JUMP-IF-EQUAL Label

NO-OP

NO-OP

ADD R3, R4

ADD R2, 2

Label:

* Có thể thấy xung đột nằm ở lệnh rẽ nhánh, chúng ta có thể chèn 2 lệnh độc lập R1 ở trên xuống vì chúng không ảnh hưởng đến R3, R4 và R2:

COMPARE R3, R4

JUMP-IF-EQUAL Label

STORE –100(R2), R1

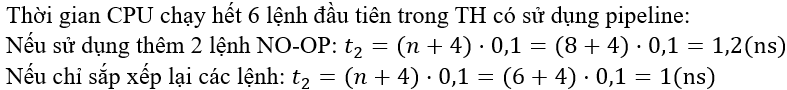
LOAD R1, (00FF)

ADD R3, R4

ADD R2, 2

Label:

1. Thời gian CPU chạy hết 6 lệnh đầu tiên trong TH không sử dụng pipeline:



1. Cho đoạn chương trình sau (R1, R2 là các thanh ghi và lệnh quy ước theo dạng LỆNH <ĐÍCH> <GỐC>):

(1) MOVE R0, #400

(2) LOAD R1, #2000

(3) STORE (R1), R0

(4) SUBSTRACT R0, #20

(5) ADD 2000, #10

(6) ADD R0, (R1)

Nêu hướng giải quyết xung đột dữ liệu trong pipeline khi thực hiện đoạn chương trình trên biết mỗi lệnh được chia thành 5 giai đoạn (IF, ID, EX, MEM và WB)

LG

Ta có thể chèn lệnh NO-OP để giải quyết xung đột dữ liệu trong pipeline khi thực hiện đoạn chương trình trên:

MOVE R0, #400

LOAD R1, #2000

NO-OP

NO-OP ;giải quyết xung đột chưa gán xong R0

NO-OP ;giải quyết xung đột chưa gán xong R1

STORE (R1), R0

SUBSTRACT R0, #20

ADD 2000, #10

NO-OP ;giải quyết xung đột chưa xong STORE (R1), R0

NO-OP ;giải quyết xung đột chưa xong SUBSTRACT R0, #20

NO-OP ;giải quyết xung đột chưa xong ADD 2000, #10 (liên ;quan đến ô nhớ 2000 lưu trong (R1)

ADD R0, (R1)

1. Cho đoạn chương trình sau (R1, R2 là các thanh ghi và lệnh quy ước theo dạng LỆNH <ĐÍCH> <GỐC>):

(1) LOAD R2, #400

(2) LOAD R1, #1200

(3) STORE (R1), R2

(4) SUBSTRACT R2, #20

(5) ADD 1200, #10

(6) ADD R2, (R1)

Nêu hướng giải quyết xung đột dữ liệu trong pipeline khi thực hiện đoạn chương trình trên biết mỗi lệnh được chia thành 5 giai đoạn (IF, ID, EX, MEM và WB)

LG

Ta có thể chèn lệnh NO-OP để giải quyết xung đột dữ liệu trong pipeline khi thực hiện đoạn chương trình trên:

LOAD R2, #400

LOAD R1, #1200

NO-OP

NO-OP ;giải quyết xung đột chưa gán xong R2

NO-OP ;giải quyết xung đột chưa gán xong R1

STORE (R1), R2

SUBSTRACT R2, #20

NO-OP

NO-OP ;giải quyết xung đột chưa xong STORE (R1), R2 (liên ;quan đến ô nhớ 1200 trong (R1) sẽ sử dụng sau đây)

ADD 1200, #10 ;giải quyết xung đột chưa xong SUBSTRACT R2, #20

NO-OP

NO-OP

NO-OP ;giải quyết xung đột chưa xong ADD 1200, #10 (liên ;quan đến ô nhớ 1200 lưu trong R1)

ADD R2, (R1)

1. Dịch lệnh mã C sau thành mã máy dạng lệnh CPU, sau đó nêu hướng giải quyết xung đột dữ liệu trong pipeline:

g = g – A[8];

A[12] = h + A[8];

Biết A là mảng các phần tử 32bit; g được lưu ở R1, h lưu ở R2 và R3 chứa địa chỉ cơ sở của mảng A.

LG

Chú ý rằng, trong 1 mảng các phần tử 32bit, địa chỉ cơ sở các phần tử sẽ cách nhau 4 đơn vị vì mỗi phần tử có kích thước 4 byte (32bit = 4 byte).

Ví dụ nếu phần tử đầu tiên trong mảng A[0] lưu tại địa chỉ 1000, thì A[1] sẽ lưu tại 1004, A[2] lưu tại 1008,…

Địa chỉ cơ sở của mảng A (hay A[0]) lưu tại R3 => địa chỉ cơ sở của phần tử A[x] bất kì là (4 \* x)(R3).

Chương trình tuần tự:

SUBSTRACT R1, 32(R3) ;R1 <- R1 – M[R3 + 32]

ADD R2, 32(R3) ;R2 <- R2 + M[R3 + 32]

STORE 48(R3), R2 ;M[R3 + 48] <- R2

Chương trình sử dụng pipeline kết hợp xử lí xung đột:

ADD R2, 32(R3) ;R2 <- R2 + M[R3 + 32]

SUBSTRACT R1, 32(R3) ;R1 <- R1 – M[R3 + 32]

NO-OP

NO-OP

STORE 48(R3), R2 ;M[R3 + 48] <- R2

1. Dịch lệnh mã C sau thành mã máy dạng lệnh CPU, sau đó nêu hướng giải quyết xung đột dữ liệu trong pipeline:

if (i == j)

f = g + h;

f = f – i;

Biết f, g, h, i, j đang được lưu giá trị ở thanh ghi R1, R2, R3, R4, R5.

LG

Chương trình tuần tự:

COMPARE R4, R5 ;so sánh R4, R5

BRANCH\_IF\_NOT\_EQUAL ELSE ;Nhảy đến nhãn ELSE nếu không bằng

ADD R1, R2, R3 ;R1 <- R2 + R3

ELSE:

SUBSTRACT R1, R1, R4 ;R1 <- R1 + R4

Chương trình sử dụng pipeline kết hợp xử lí xung đột:

COMPARE R4, R5 ;so sánh R4, R5

BRANCH\_IF\_NOT\_EQUAL ELSE ;Nhảy đến nhãn NEXT nếu không bằng

ADD R1, R2, R3 ;R1 <- R2 + R3

NO-OP ;Xử lí xung đột trong trường hợp EQUAL, cần cập nhật xong R1 để tiếp tục tính toán SUB bên dưới.

ELSE:

SUBSTRACT R1, R1, R4 ;R1 <- R1 + R4

1. Dịch lệnh mã C sau thành mã máy dạng lệnh CPU, sau đó nêu hướng giải quyết xung đột dữ liệu trong pipeline:

if (i == j)

f = g + h;

else

f = f – i;

Biết f, g, h, i, j đang được lưu giá trị ở thanh ghi R1, R2, R3, R4, R5.

LG

Chương trình:

COMPARE R4, R5 ;so sánh R4, R5

BRANCH\_IF\_NOT\_EQUAL ELSE ;Nhảy đến nhãn ELSE nếu không bằng

ADD R1, R2, R3 ;R1 <- R2 + R3

JUMP END

ELSE:

SUBSTRACT R1, R1, R4 ;R1 <- R1 + R4

END:

Chương trình trên không có xung đột pipeline.

1. Dịch lệnh mã C sau thành mã máy dạng lệnh CPU:

switch (amount) {

case 20: fee = 2; break;

case 50: fee = 3; break;

case 100: fee = 5; break;

default: fee = 0;

}

Biết amount được lưu tại R1, fee được lưu tại R2.

LG

Chương trình:

COMPARE R1, #20

BRANCH-IF-EQUAL CASE20

COMPARE R1, #50

BRANCH-IF-EQUAL CASE50

COMPARE R1, #100

BRANCH-IF-EQUAL CASE100

MOVE R2, #0

JUMP END

CASE20:

MOVE R2, #2

JUMP END

CASE50:

MOVE R2, #3

JUMP END

CASE100:

MOVE R2, #5

JUMP END

END:

1. Dịch lệnh mã C sau thành mã máy dạng lệnh CPU:

int count = 0;

int i = 0;

while (i < n) {

if (A[i] == k)

count += 1;

i += 1;

}

Biết i được lưu ở thanh ghi R0, n ở R1, k ở R2, count ở R4, địa chỉ mảng A ở R3 và A là mảng các phần tử 32bit.

LG

Chương trình:

CLEAR R0

CLEAR R4

CLEAR R5 ;R5 mục đích lưu địa chỉ A[i]

WHILE:

COMPARE R0, R1 ;so sánh i với n

BRANCH-IF-GREATER-EQUAL-THAN END

SHL R0, 2 ;lấy địa chỉ tương ứng A[i] là địa chỉ của A + i \* 4

MOV R5, R3

ADD R5, R0

SHR R0, 2

COMPARE (R5), R2

BRANCH-IF-NOT-EQUAL CONTINUE

INCREMENT R4

CONTINUE:

INCREMENT R0

JUMP WHILE

END:

1. Dịch lệnh mã C sau thành mã máy dạng lệnh CPU:

int arr[1000];

int i;

for (i = 0; i < 1000; i = i + 1) {

arr[i] = arr[i] \* 8;

}

Biết i được lưu ở thanh ghi R0, địa chỉ mảng arr ở R1 và A là mảng các phần tử 32bit.

LG

Chương trình:

CLEAR R0

CLEAR R2

FOR:

COMPARE R0, #1000

BRANCH\_IF\_GREATER\_EQUAL END

MOVE R2, (R1)

SHL R2, 3

INCREMENT R0

STORE (R1), R2

ADD R1, #4

JUMP FOR

END: