By: Tôn Mai Hương

Chương 1:

1.Các thành phần chính của kiến trúc máy tính là gì?

Các thành phần chính của kiến trúc máy tính:

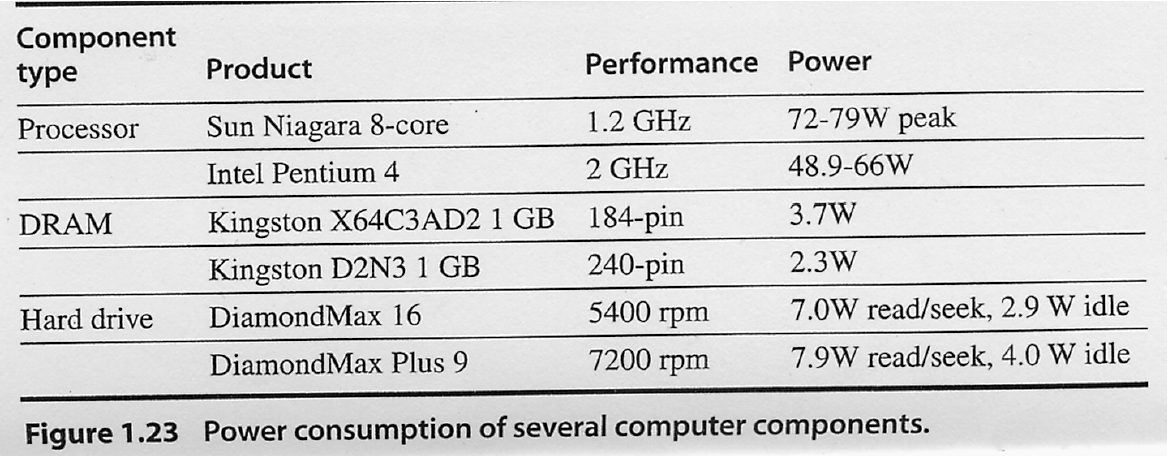
* Kiến trúc tập lệnh: đánh địa chỉ bộ nhớ, tập lệnh, mã hóa lệnh,...
* Tổ chức: cách kết nối bộ nhớ, cách kết nối cái vi xử lý, cách phối hợp vi xử lý, đường ống, bộ đệm ( vi kiến trúc)...
* Phần cứng: giá trị xung nhịp đồng hồ, hệ thống bộ nhớ...

2. Mục đích của kiến trúc máy tính là gì?

Mục đích của KTMT: thiết kế kiến trúc tập lệnh, tổ chức, phần cứng nhằm đáp ứng các tiêu chí định lượng:

* Các yêu cầu chức năng( ứng dụng máy chủ, học tập, chơi game…)
* Giá cả
* Mức tiêu thụ năng lượng
* Hiệu năng
* Tính sẵn sàng
* Tính tin cậy

3. Hình sau mô tả công suất tiêu thụ của các bộ phận máy tính



Khi thiết kế trung tâm dữ liệu, vấn đề quan trọng là làm mát.

a) Một cửa làm mát cho một tủ máy chủ giá 4000$ và thải được 14kW từ tủ vào phòng. Bao nhiêu máy chủ với cấu hình 1 Sun Niagara 8-core, 1GB 240 pin DRAM, và một đĩa cứng 5400rpm có thể được làm mát với 1 cửa làm mát.

b) Nếu chúng ta dùng RAID1 – tăng gấp đôi số lượng đĩa cứng – thì bao nhiêu máy chủ có thể được làm mát với 1 cửa làm mát.

a, 14000/(79+2,3+7,0)=158(Làm tròn xuống vì nếu làm tròn lên sẽ vượt quá 14kw=> Cháy)

b,Đĩa cứng-Hard drive

14000/(79+2,3+7,0\*2)=146

4. Giả sử chúng ta muốn tăng cường các bộ vi xử lý được sử dụng cho máy chủ Web . Vi xử lý mới

nhanh hơn 10 lần bộ xử lý ban đầu khi tính toán trong các ứng dụng Web. Giả sử rằng các bộ xử lý ban đầu bận rộn với tính toán trong 40 % thời gian và chờ I/O trong 60 % thời gian. Hỏi hiệu năng tổng thể tăng lên bao nhiêu khi sử dụng bộ vi xử lý mới.

Vi xử lý tăng 10 lần => H tăng=1/(0.4/10+0.6)=1.5625

5. Một chuyển đổi phổ biến cần thiết trong bộ xử lý đồ họa là căn bậc hai. Thực hiện căn bậc hai dấu phẩy động làm thay đổi đáng kể hiệu năng của các bộ xử lý đồ họa. Giả sử căn bậc hai dấu phẩy động chiếm 20 % thời gian thực thi của một ứng dụng đồ họa quan trọng. Một đề nghị là tăng cường phần cứng tăng tốc thực hiện căn bậc hai dấu phẩy động lên 10 lần. Một đề nghị khác là tăng tốc độ thực hiện phép tính dấu phẩy động lên 1,6 lần. Thực hiện phép tính dấu phẩy động chiếm một nửa thời gian thực thi của ứng dụng. So sánh hai phương án thiết kế .

Phương án 1:

Độ tăng H1=1/(0,2/10+(1-0,2))=1.2195

Phương án 2:

Độ tăng H2= 1/(0.5/1.6+(1-0.5))=1.23

6. Có ba cải tiến cho kết quả tăng hiệu năng như sau

Cải tiến A: Tăng hiệu năng 30

Cải tiến B: Tăng hiệu năng 20

Cải tiến C: Tăng hiệu năng 15

a) Nếu cải tiến A và B được sử dụng trong 25% thời gian thì cải tiến C phải được dùng trong bao nhiêu lâu để hiệu năng tổng thể tăng lên 10 lần.

b) Giả sử cải tiến A, B, C được sử dụng trong 25%, 35%, 10% thời gian thì số thời gian không áp dụng cải tiến chiếm bao nhiêu % trong khoảng thời gian chạy mới.

c) Giả sử A,B,C có thể được sử dụng trong 15%, 15%, 70% thời gian. Nếu chỉ sử dụng một cải tiến, bạn chọn cải tiến nào? Nếu sử dụng 2 cải tiến, bạn chọn cải tiến nào?

a,

1/(0,25/30+0,25/20+Fc/15+(1-0,25-0,25-Fc))=10

=>Fc=45%

b, Tổng thời gian ban đầu T

Ban đầu, số thời gian không áp dụng cải tiến là:

(100%-25%-35%-10%)T=30%T

Sau khi áp dụng cải tiến, tổng thời gian là:

25%T/30+35%T/20+10%T/15+30%T=133%T/4

Sau khi áp dụng cải tiến, thời gian không áp dụng cải tiến chiếm số % là:

(30%T):(133%T/4)\*100%=90,2%

c, Nếu chỉ sd 1 cải tiến:

+) A: H tăng= 1/(0,15/30+1-0,15)=1,1695

+) B: H tăng=1/(0,15/20+1-0,15)=1,166

+) C: H tăng=1/(0,7/15+1-0,7)=2,884

Sd cải tiến C

Nếu sd 2 cải tiến thì sd A và C

Chương 2:

1, Kiến trúc load – store là gì. Phân tích ưu điểm, nhược điểm của kiến trúc này với các kiến trúc khác có sử dụng các thanh ghi dùng chung.

- Kiến trúc Load-Store là giao diện giữa phần cứng và phần mềm

- Ưu nhược điểm của kiến trúc này với các kiến trúc khác có sử dụng các thanh ghi dùng chung:

+ Ưu điểm: đơn giản, mã hóa tập lệnh có độ dài cố định, giải mã đơn giản, các lệnh có số chu kì gần giống nhau. Đây là kiến trúc có tốc độ cao, hiệu quả nhất.

+ Nhược điểm: số tập lệnh lớn hơn các cấu trúc có lệnh tham chiếu bộ nhớ, nhiều câu lệnh dẫn đến chương trình lớn hơn.

2, Phân tích ưu điểm, nhược điểm của phương pháp mã hóa lệnh sử dụng độ dài thay đổi và độ dài cố định

Phân tích ưu nhược điểm của phương pháp mã hóa lệnh sử dụng độ dài thay đổi và độ dài cố định:

|  |  |
| --- | --- |
| Độ dài thay đổi | Độ dài cố định |
| - File lệnh ngắn  - Giải mã phức tạp  - Chạy chậm | - File lệnh dài  - Giải mã đơn giản  - Chạy nhanh |

3, Giả sử tập lệnh dùng 16 bit dài cố định. 6 bit dùng để chỉ toán hạng. Có 3 lệnh 2 toán hạng. 33 lệnh không toán hạng. Tính số lệnh có 1 toán hạng có thể mã hóa được.

Phân tích đề bài: tập lệnh dùng 16 bit dài cố đinh, mỗi toán hạng chiếm 6 bit

Có 3 loại toán hạng: 3 lệnh 2 toán hạng, 33 lệnh 0 toán hạng, x lệnh 1 toán hạng. Cần tìm x

- Xét lệnh 2 toán hạng

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Ops | Param1 | Param2 |
| 2 | 2 | 6 | 6 |

- Xét lệnh 0 toán hạng

|  |  |
| --- | --- |
| Type | Ops |
| 2 | 6 |

- Xét lệnh 1 toán hạng

|  |  |  |
| --- | --- | --- |
| Type | Ops | Param |
| 2 | 8 | 6 |

Vậy số lệnh 1 toán hạng có thể mã hóa được là 2 mũ 8 bằng 256

4,1 bộ VXL có 32 thanh ghi, dùng 16 bit immediate, và 142 lệnh ISA. Trong một chương trình có:

a. 20% số câu lệnh có 1 thanh ghi đầu vào, 1 thanh ghi đầu ra

b. 30% số câu lệnh có 2 thanh ghi đầu vào, 1 thanh ghi đầu ra

c. 25% số câu lệnh có 1 thanh ghi đầu vào, 1 thanh ghi đầu ra và 1 immediate đầu vào

d. 25% số câu lệnh có 1 immediate đầu vào, 1 thanh ghi đầu ra

Với mỗi loại câu lệnh trên cần bao nhiêu bit để biểu diễn (Phải là bội của 8).

Tính số memory tiết kiệm được nếu dùng câu lệnh độ dài thay đổi so sánh với dùng câu lệnh độ dài cố định.

a, 20% số câu lệnh có 1 thanh ghi đầu vào, 1 thanh ghi đầu ra

|  |  |  |
| --- | --- | --- |
| Ops | Reg in | Reg out |
| 8 | 5 | 5 |

Total: 8+5+5=18

Vì số bit để biểu diễn là bội của 8 nên cần dùng 24 bit.

b, 30% số câu lệnh có 2 thanh ghi đầu vào, 1 thanh ghi đầu ra

|  |  |  |  |
| --- | --- | --- | --- |
| Ops | Reg in 1 | Reg in 2 | Reg out |
| 8 | 5 | 5 | 5 |

Total: 8+5+5+5=23

Vì số bit để biểu diễn là bội của 8 nên cần dùng 24 bit.

c, 25% số câu lệnh có 1 thanh ghi đầu vào, 1 thanh ghi đầu ra và 1 immediate đầu vào

|  |  |  |  |
| --- | --- | --- | --- |
| Ops | Reg in | Reg out | Imm |
| 8 | 5 | 5 | 16 |

Total: 8+5+5+16=34

Vì số bit để biểu diễn là bội của 8 nên cần dùng 40 bit.

c, 25% số câu lệnh có 1 immediate đầu vào, 1 thanh ghi đầu ra

|  |  |  |
| --- | --- | --- |
| Ops | Imm | Reg out |
| 8 | 16 | 5 |

Total: 8+16+5=29

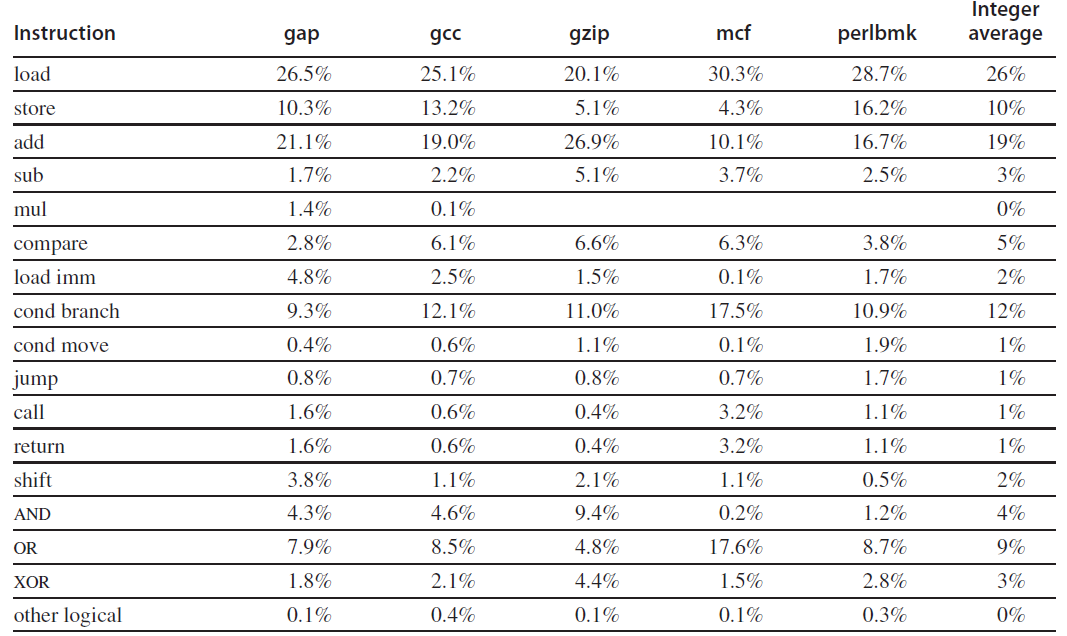
Vì số bit để biểu diễn là bội của 8 nên cần dùng 32 bit

Do đó, độ dài trung bình là: 24\*20%+24\*30%+40\*25%+32\*25%=30 bit

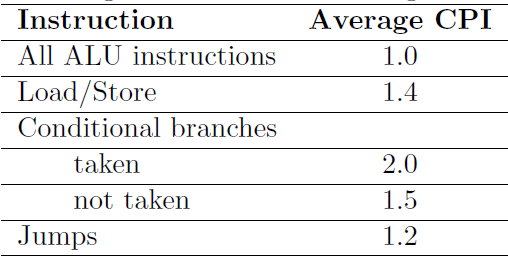
Độ dài cố định= Độ dài max=40 bit

Số memory tiết kiệm được là 40-30=10 bit

5, Hãy tính toán CPI cho MIPS sử dụng hình sau (Chỉ dùng gap và gcc để tính tần suất lệnh trung bình)



Giả sử chúng ta có số liệu CPI trung bình cho từng loại như sau



Với giả sử rằng 60% lệnh rẽ nhánh có điều kiện được thực hiện. Và các lệnh “other logical” thuộc loại lệnh ALU

Ta có bảng tính tần suất trung bình của từng loại lệnh: ( đơn vị %)

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction** | **Gap** | **Gcc** | **Ftb** |
| Load | 26.5 | 25.1 | 25.8 |
| Store | 10.3 | 13.2 | 11.75 |
| Add | 21.1 | 19 | 20.05 |
| Sub | 1.7 | 2.2 | 1.95 |
| Mul | 1.4 | 0.1 | 0.75 |
| Compare | 2.8 | 6.1 | 4.45 |
| Load imm | 4.8 | 2.5 | 3.65 |
| Cond branch | 9.3 | 12.1 | 10.7 |
| Cond move | 0.4 | 0.6 | 0.5 |
| Jump | 0.8 | 0.7 | 0.75 |
| Call | 1.6 | 0.6 | 1.1 |
| Return | 1.6 | 0.6 | 1.1 |
| Shift | 3.8 | 1.1 | 2.45 |
| AND | 4.3 | 4.6 | 4.45 |
| OR | 7.9 | 8.5 | 8.2 |
| XOR | 1.8 | 2.1 | 1.95 |
| Other logical | 0.1 | 0.4 | 0.25 |

CPItb(conditional branches)=2\*60%+1.5\*40%=1.8

Phân loại các lệnh thành 4 nhóm và tính tần suất từng nhóm:

-Nhóm 1(load/store): load, store, load imm

F1=25.8+11.75+3.65=41.2 (%)

- Nhóm 2( conditional branches):cond branch

F2=10.7(%)

-Nhóm 3(jumps): jump

F3=0.75(%)

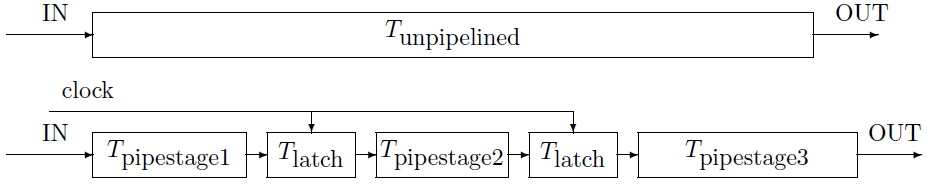
- Nhóm 4(All ALU Instructions): add,sub,mul,compare,cond move,call,return,shift,AND,OR,XOR,other logical

F4=20.05+1.95+0.75+4.45+0.5+1.1+1.1+2.45+4.45+8.2+1.95+0.25=47.2(%)

Vậy CPItb=41.2%\*1.4+10.7%\*1.8+0.75%\*1.2+47.2%.1=1.2504

**Chương 3:**

**1, Tính độ tăng hiệu năng của hệ thống pipeline sau**

****

**Biết**

**Tunpipelined = 5ms (Thời gian thực hiện khi không có pipeline)**

**Tlatch =0,2ms (thời gian để lưu trữ giữa 2 bước)**

**Max(Tpipestage)=1,3ms (Thời gian thực hiện của bước chậm nhất)**

Bài làm:

Tunpipeline=5 ms ( đề bài cho)

Tpipeline = Thời gian của bước dài nhất + Tlatch

= Max(Tpipestage)+Tlatch

= 1.3+0.2

= 1.5 ms

Độ tăng hiệu năng của hệ thống pipeline sau là:

H= Tunpipeline/ Tpipeline =5/1.5=10/3=3.33

**2, Hãy xem xét một bộ xử lý unpipelined . Giả sử rằng nó có chu kỳ đồng hồ 1 - ns và nó sử dụng 4 chu kỳ cho các hoạt động ALU và 5 chu kỳ cho các lệnh rẽ nhánh và 4 chu kỳ cho các lệnh bộ nhớ. Giả sử rằng các tần số của các lệnh này tương ứng là 50 % , 35 % và 15 % . Giả sử rằng do sự mất đối xứng của đồng hồ và thời gian thiết lập, bộ vi xử lý có đường ống thêm 0,15 ns cho mỗi chu kỳ đồng hồ. Bỏ qua các tác động độ trễ , hiệu năng sử dụng đường ống tăng lên bao nhiêu?**

Bài làm:

- Chu kỳ đồng hồ 1 ns nên thời gian 1 cycle: Tc=1ns.

Tunpipelined=(4\*50%+5\*35%+4\*15%)\*Tc=4.35\*1=4.35 (ns)

- Giả sử bộ vi xử lý pipeline là lý tưởng, CPI=1.

Do sự mất đối xứng của đồng hồ và thời gian thiết lập, bộ vi xử lí có đường ống thêm 0.15 ns cho mỗi chu kỳ đồng hồ nên Tcp=1+0.15=1.15 (ns)

Tpipeline=CPI\*Tcp=1\*1.15=1.15 (ns)

- Hiệu năng sử dụng đường ống tăng lên số lần là:

H= Tunpipelined/ Tpipeline =4.35/1.15=3.78

**3, Giả sử 1 bộ VXL chỉ có 1 cổng bộ nhớ. Giả sử rằng lệnh có truy cập bộ nhớ chiếm 40% tổng số lệnh. Giả sử rằng CPI của bộ VXL có pipeline lý tưởng (không có hazard) là 1. Giả sử rằng bộ VXL có pipeline không lý tưởng (có hazard cấu trúc) có xung nhịp đồng hồ cao hơn bộ VXL có pipeline lý tưởng là 1.05 lần. Hỏi bộ VXL nào (lý tưởng hay không lý tưởng) có hiệu năng cao hơn và cao hơn bao nhiêu lần?**

Bài làm:

CPI lt=1

CPI tb klt:

Lệnh có truy cập bộ nhớ chiếm 40% tổng số lệnh nên load store chiếm 40% nên

f load store op =40% (CPI=2 do phải hoãn đi 1 nhịp)

f non load store ops=100%-60% (CPI=1)

=>CPI tb klt = 40%\*2+60%\*1=1.4

Tpipeline lt=CPIlt\*Tc=1\*Tc=Tc

Tc ktl=Tc/1.05

Tpipeline klt=CPItb klt\*Tc klt=1.4\*Tc/1.05

H=Tpipeline klt/Tpipeline lt=(1.4\*Tc/1.05)/Tc=28/21=1.33

Vậy bộ VXL lý tưởng có hiệu năng cao hơn không lý tưởng 1.33 lần

**4,(Em làm theo đề trong video thầy giảng ạ, chính là bài 6 trên lms, bài 4 trong lms trùng với bài chương 4 nên em làm ở phần chương 4 ạ)**

**Đề bài: Xem xét thực hiện các mã sau đây trên pipeline 5 trạng thái: thanh ghi nào được đọc trong chu kì đồng hồ thứ 5, thanh ghi nào được viết vào cuối của chu kì đồng hồ thứ 5?**

**ADD R1,R2,R3**

**ADD R4,R5,R6**

**ADD R7,R8,R9**

**ADD R10,R11,R12**

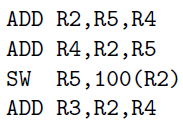
**ADD R13,R14,R15**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | CC1 | CC2 | CC3 | CC4 | CC5 | CC6 | CC7 | CC8 | CC9 | |
| ADD R1,R2,R3 | Load lenh | Giai ma/doc R2,R3 | Tinh R2+  R3 |  | Ghi R1 |  |  |  |  | |
| ADD R4,R5,R6 |  | Load lenh | Giai ma/  doc R5,R6 | Tinh R5+R6 |  | Ghi R4 |  |  |  | |
| ADD R7,R8,R9 |  |  | Load lenh | Giai ma/doc R8,R9 | Tinh R8+R9 |  | Ghi R7 |  |  | |
| ADD R10,R11,R12 |  |  |  | Load lenh | Giai ma/doc R11,R12 | Tinh R11+R12 |  | Ghi R10 |  | |
| ADD R13,R14,R15 |  |  |  |  | Load lenh | Giai ma/doc R14,R15 | Tinh R14+  R15 |  | Ghi R13 |  |

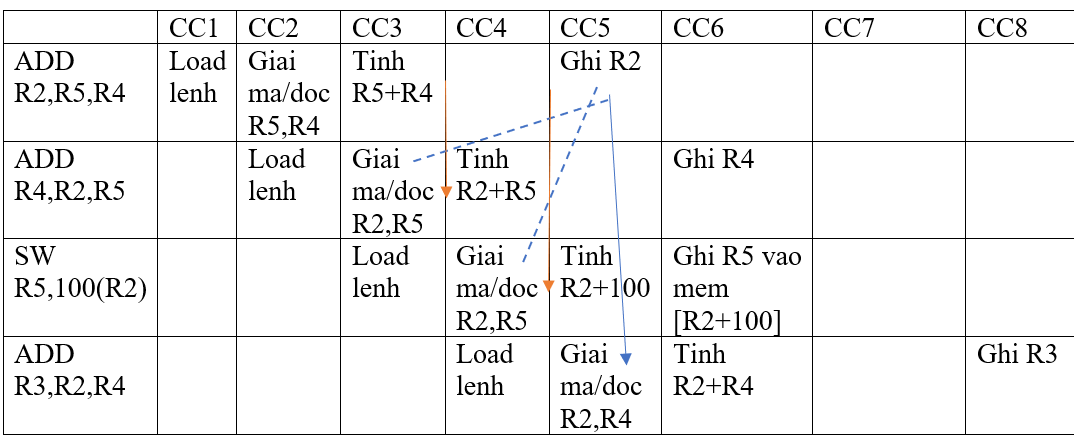
- R11,R12 được đọc vào chu kì đồng hồ thứ 5.

- Không có thanh ghi nào được viết vào cuối chu kì đồng hồ thứ 5. ( Chỉ có R1 được viết vào đầu chu kì đồng hồ thứ 5)

**5, (bài 6 trong video thầy giảng) Xác định tất cả các phụ thuộc dữ liệu trong đoạn mã sau , giả sử rằng chúng ta là sử dụng các đường dữ liệu MIPS 5 tầng pipeline. Phụ thuộc nào có thể được giải quyết thông qua phương pháp chuyển tiếp ?**

****

Bài làm: Mũi tên màu cam là những phụ thuộc được giải quyết thông qua phương pháp chuyển



Thêm mũi tên nét đứt xanh từ ghi R4 xuống giải mã, đọc R2,R4

Thêm 1 mũi tên cam từ trước ghi R4 xuống hàng cuối

**6, (Bài 5 trong video thầy giảng) Xem xét việc thực hiện các mã sau đây trên pipeline 5 trạng thái. Dùng kỹ thuật chuyển để giải quyết các phụ thuộc dữ liệu.**

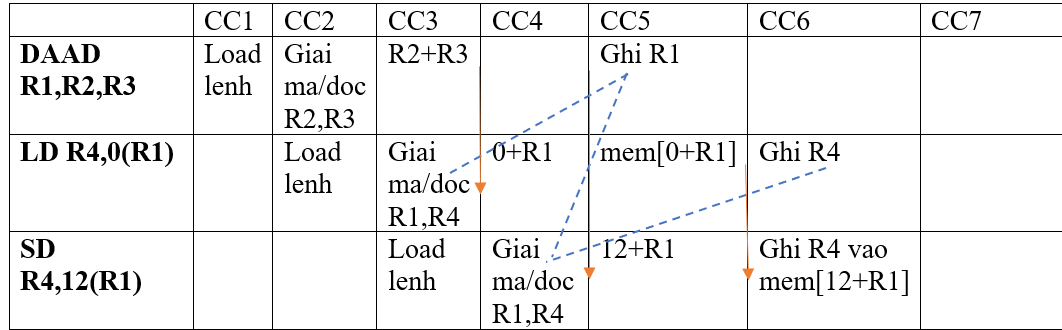
**DAAD R1,R2,R3**

**LD R4,0(R1)**

**SD R4,12(R1)**

Bài làm:

Mũi tên màu cam là những phụ thuộc được giải quyết thông qua phương pháp chuyển

****

**7, Tính độ tăng hiệu năng của hệ thống pipeline sau**

- **Số bước (trạng thái - noofstages): 5 (Thời gian thực hiện của các bước là như nhau)**

- **Tỷ lệ lệnh rẽ nhánh trong chương trình (branch\_freq): 20%**

- **Số chu trình bị mất do lệnh rẽ nhánh (branch\_penalty): 2**

-

Bài làm:

Độ tăng hiệu năng: 5/(1+2\*20%)=3.57

Chương 4:

1, Hazard là gì?

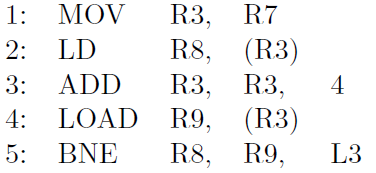
Hazard là một yếu tố quan trọng ảnh hưởng trực tiếp tới tốc độ của VXL trong kỹ thuật pipeline, yếu tố này là sự xung đột.

2,Phân biệt sự khác nhau giữa phụ thuộc tên và phụ thuộc thực sự

Phân biệt giữa phụ thuộc tên và phụ thuộc thực sự:

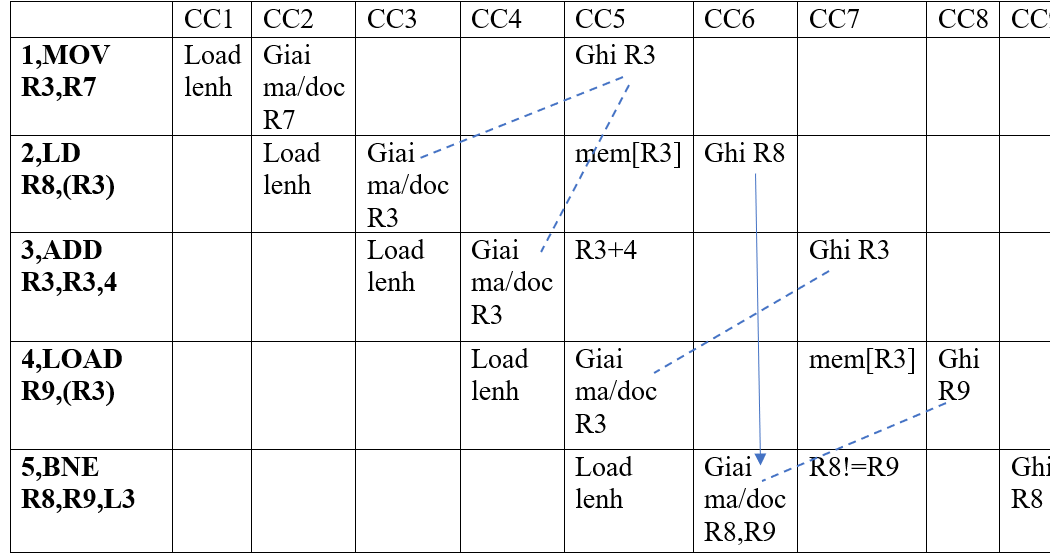
|  |  |
| --- | --- |
| **Phụ thuộc tên** | **Phụ thuộc thực sự** |
| - Hai câu lệnh cùng sử dụng 1 thanh ghi hay 1 ô nhớ gọi là tên.  - Không có dòng dữ liệu giữa 2 câu lệnh liên quan đến tên đó. | - Lệnh i tạo ra kết quả có thể được sử dụng bởi câu lệnh j  - Câu lệnh j phụ thuộc dữ liệu câu lệnh k vào câu lệnh k phụ thuộc dữ liệu câu lệnh i. |

3, Xem xét chương trình hợp ngữ sau



Chương trình này có tồn tại các quan hệ WAW, RAW, và WAR không? Hãy chỉ ra các quan hệ đó.

Quan hệ nào là phụ thuộc tên và quan hệ nào là phụ thuộc thực sự



- Quan hệ WAW: không xảy ra

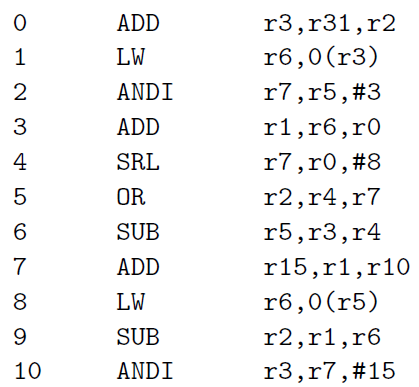
- Quan hệ RAW: 1 và 2, 1 và 3, 3 và 4, 4 và 5

- Quan hệ WAR: không xảy ra

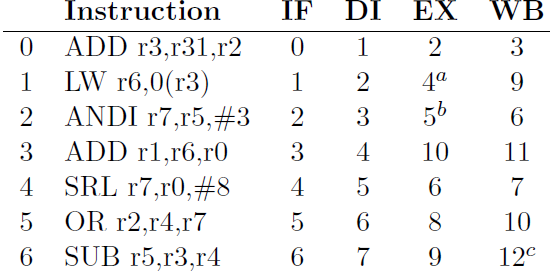
- Quan hệ phụ thuộc tên: 2 và 3, 2 và 4

- Quan hệ phụ thuộc thật sự: 1 và 2 và 5, 1 và 3 và 4 và 5.

4,Xem xét chương trình hợp ngữ sau



Giả sử pipeline 4 trạng thái (IF, DI, EX, WB) không có kỹ thuật chuyển tiếp. Với các lệnh ALU, mỗi trạng thái cần 1 cycle. Với lệnh load từ bộ nhớ, trạng thái EX cần 5 cycle. Giả sử các lệnh được lập lịch như sau:

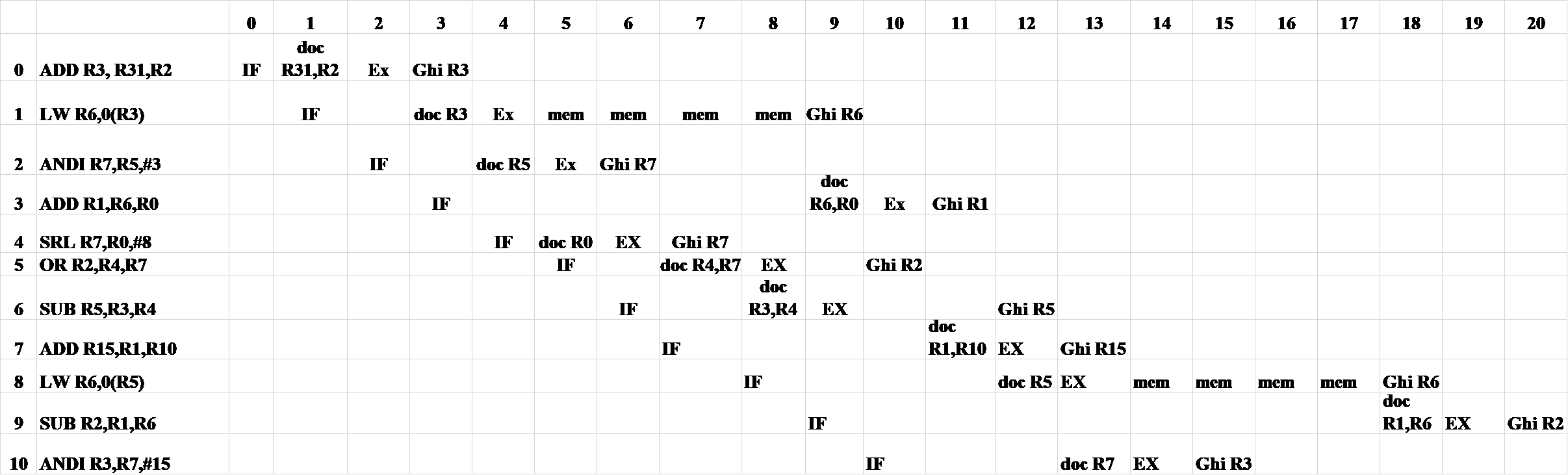


a) Giải thích tại sao ex của lệnh 1 lại ở cycle 4.

b) Giải thích tại sao ex của lệnh 2 lại ở cycle 5.

c) Giải thích tại sao wb của lệnh 6 lại ở cycle 12.

d) Hoàn thành bảng với các lệnh còn lại



a) EX của lệnh 1 ở CC4 là bởi vì hết lệnh 0 ( hết nửa đầu CC3) mới có ghi R3 nên ở lệnh 1, nửa sau CC3 mới đọc R3 và CC4 mới thực hiện EX.

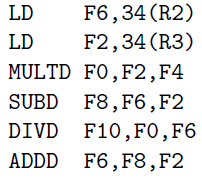
b) EX của lệnh 2 ở CC5 là vì EX của lệnh 1 ở CC4 rồi , không thể trùng được

c) Vì EX của lệnh 6 đã ở CC9, còn CC10,CC11 lần lượt là WB của lệnh 5, lệnh 3 nên WB của lệnh 6 phải trễ xuống CC12

d) Hoàn thành bảng với các lệnh còn lại:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Instruction | IF | ID | EX | WB |
| 0 | ADD R3, R31,R2 | 0 | 1 | 2 | 3 |
| 1 | LW R6,0(R3) | 1 | 3 | 4 | 9 |
| 2 | ANDI R7,R5,#3 | 2 | 4 | 5 | 6 |
| 3 | ADD R1,R6,R0 | 3 | 9 | 10 | 11 |
| 4 | SRL R7,R0,#8 | 4 | 5 | 6 | 7 |
| 5 | OR R2,R4,R7 | 5 | 7 | 8 | 10 |
| 6 | SUB R5,R3,R4 | 6 | 8 | 9 | 12 |
| 7 | ADD R15,R1,R10 | 7 | 11 | 12 | 13 |
| 8 | LW R6,0(R5) | 8 | 12 | 13 | 18 |
| 9 | SUB R2,R1,R6 | 9 | 18 | 19 | 20 |
| 10 | ANDI R3,R7,#15 | 10 | 13 | 14 | 15 |

5, Lập lịch đoạn mã sau dung thuật toán Tomasulo với các thông tin sau: VXL dùng pipeline 4 trạng thái (load lệnh, giải mã và phát hành, xử lý, ghi lại), không có chuyển về phía trước. Phần cứng có 3 đơn vị Load với độ trễ thực hiện là 2 chu trình, 3 đơn vị Add/Sub với độ trễ thực hiện là 2 chu trình, 3 đơn vị Mult/Div với độ trễ thực hiện Mult là 10 chu trình, Div là 40 chu trình. Giả sử lệnh đầu tiên được phát hành ở chu trình 1.



- Lệnh LD thứ 2 hoàn thành ở chu trình thứ mấy?

- Lệnh MULTD hoàn thành ở chu trình thứ mấy?

- Lệnh ADDD hoàn thành ở chu trình thứ mấy?

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | **ISSUE** | **EX BEGIN** | **EX END** | **WB** |
| **LD F6,34(R2)** | **2** | 1 | 2 | 3 | 4 |
| **LD F2,34(R3)** | **2** | 2 | 3 | 4 | 5 |
| **MULTD F0,F2,F4** | **10** | 3 | 6 | 15 | 16 |
| **SUBD F8,F6,F2** | **2** | 4 | 6 | 7 | 8 |
| **DIVD F10,F0,F6** | **40** | 5 | 17 | 56 | 57 |
| **ADDD F6,F8,F2** | **2** | 6 | 9 | 10 | 11 |

- Lệnh LD thứ 2 hoàn thành ở chu trình thứ 5

- Lệnh MULTD hoàn thành ở chu trình thứ 16

- Lệnh ADDD hoàn thành ở chu trình thứ 11

6, Một bộ VXL dùng lập lịch động phát hành lệnh và lấy toán tử cùng lúc. Bộ VXL có 3 đơn vị xử lý 1 đơn vị Load/Store, 1 đơn vị Add/Sub, 1 đơn vị Mul/Div. VXL có 1 trạm dự trữ với 1 khe cho một đơn vị xử lý và một file thanh ghi. VXL cho phép phát hành lệnh không theo thứ tự, xử lý không theo thứ tự, pipeline 4 trạng thái (load lệnh, giải mã và phát hành, xử lý, ghi lại), không có chuyển về phía trước. Load/Store với độ trễ thực hiện là 2 chu trình, Add/Sub với độ trễ thực hiện là 1 chu trình, độ trễ thực hiện Mult là 2 chu trình, Div là 4 chu trình. Bắt đầu với chuỗi lệnh sau đã ở trong bộ đệm tải lệnh, trạm dự trữ trống, với mỗi lệnh tìm chu trình nó được phát hành và chu trình nó ghi kết quả.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | **ISSUE** | **EX BEGIN** | **EX END** | **WB** |
| **LOAD R6,34(R12)** | **2** | 1 | 2 | 3 | 4 |
| **LOAD R2,45(R13)** | **2** | 4 | 5 | 6 | 7 |
| **MUL R0,R2,R4** | **2** | 1 | 8 | 9 | 10 |
| **SUB R8,R2,R6** | **1** | 1 | 8 | 8 | 9 |
| **DIV R10,R0,R6** | **4** | 10 | 11 | 14 | 15 |
| **ADD R6,R8,R2** | **1** | 9 | 10 | 10 | 11 |

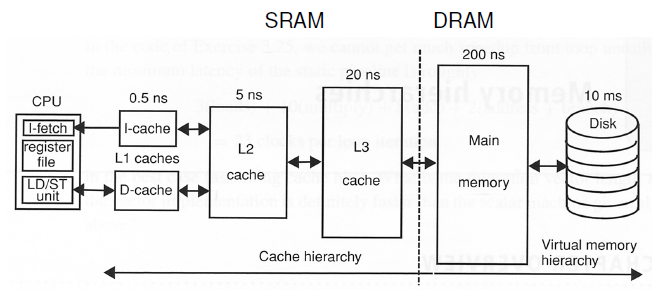
Chương 5\_1:

1,Mô tả khái niệm phân tầng bộ nhớ và giải thích tầm quan trọng của nó.

- Phân tầng bộ nhớ là sự kết hợp các kiểu bộ nhớ với mỗi tốc độ truy cập và dung lượng khác nhau thành một hệ thống để cung cấp hiệu suất cao và giá thành ít.

- Tầm quan trọng: cải thiện hiệu suất, tiết kiệm giá thành.

Phân tầng bộ nhớ:



2,Một bộ đệm kích thước 16kB. Bộ đệm dùng liên kết tập hợp, mỗi tập hợp có 4 khối, mỗi khối 16 byte. Xác định số khối, số tập hợp của bộ đệm đó.

Một bộ đệm có kích thước 16kB=16000 Byte

Mỗi khối 16 Byte => Số khối trong bộ đệm là : 16000/16= 1000 khối

Mỗi tập hợp có 4 khối => Số tập hợp trong bộ đệm là: 1000/4=250 tập hợp

3,Một hệ thống có bộ đệm 256 byte. Hệ thống sử dụng liên kết tập hợp với 16 tập hợp. Một địa chỉ có 16 bit, phần địa chỉ khối trong địa chỉ là 13 bit.

a,Xác định số khối trong bộ đệm

b,Xác định kích thước khối

c,Ta có một địa chỉ 333316 , xác định giá trị các trường (tag, index, offset) trong địa chỉ, xác định vị trí dữ liệu trong bộ đệm (vị trí khối trong bộ đệm, vị trí dữ liệu trong khối)

a, Một địa chỉ có 16 bit, phần địa chỉ khối trong địa chỉ là 13 bit nên:

-) Tag+ Index=13 bit

-) Offset = 16-13=3 bit

Do đó, kích thước khối là 2^3=8 byte

b, Hệ thống có 256 byte=> Số khối trong bộ đệm là 256/8=32 (khối)

c, Địa chỉ 333316 là 0011001100110011

Off set chiếm 3bit

Một tập hợp có 2 khối => cần 1 bit để phân biệt 2 giá chị => Index chiếm 1 bit

Ta biểu diễn như sau:

|  |  |  |
| --- | --- | --- |
| Block address | | Block offset |
| Tag | Index |
| 001100110011 | 0 | 011 |

Đổi Tag: 001100110011 sang hệ thập phân là 819

Hệ thống sử dụng liên kết tập hợp với 16 tập hợp

819 chia 16 dư 3

Do đó vị trí dữ liệu trong bộ đệm là: 3

4,Giải thích hai cơ chế ghi bộ nhớ

Giải thích 2 cơ chế ghi bộ nhớ:

|  |  |
| --- | --- |
| Ghi luôn | Ghi sau |
| - Thông tin được ghi vào khối trong bộ đệm và trong bộ nhớ  - Thường kết hợp với bộ đệm ghi nên CPU không phải đợi kết quả ghi vào bộ nhớ. | - Thông tin được ghi vào khối trong bộ đệm.  - Chỉ copy khối trong bộ đệm vào bộ nhớ khi khối đó được thay ra. |

5,Tại sao cần thay thế khối? Thuật toán thay thế khối là gì? Tại sao các thuật toán thay thế khối không thể áp dụng được cho bộ đệm liên kết trực tiếp?

Khi cache đầy mà lại cần chưa một nội dung khác từ bộ nhớ chính cần thay thế khối.

Thuật toán thay thế khối:

- Ngẫu nhiên

- LRU (least recently used) khối dùng ít nhất

- FIFO

Các thuật toán thay thế khối không thể áp dụng được cho bộ đệm liên kết trực tiếp vì liên kết trực tiếp chỉ có thể lựa chọn 1 vị trí duy nhất.

Chương 5\_2:

6,Giả sử ta có một máy tính với CPI=1.0 khi tất cả các truy cập bộ nhớ được thực hiện trong cache. Máy tính có 50% số lệnh load/store dùng truy cập dữ liệu. Miss penalty là 25 chu trình. Miss rate là 2%. Nếu tất cả các câu lệnh đều chạm bộ đệm (cache hit) thì số lần lỡ cho 1 câu lệnh (misses / instruction) là bao nhiêu, hiệu năng sẽ tăng bao nhiêu.

\* Khi tất cả các câu lệnh đều chạm bộ đệm(cache hit)thì số lần lỡ cho một câu lệnh misses/instruction=0

\* Tính hiệu năng tăng

Miss rate=2%=0.02

Miss penalty =25

50% số lệnh load/store dùng truy cập dữ liệu

=> mem access/instruction=1+0,5=1,5

Độ tăng hiệu năng

= Texe\_example\_cache/Texe\_ideal\_cache

=[IC\*(1+1.5\*0.02\*25)\*Tc]/(IC\*1\*Tc)

=1.75

7,Giả sử rằng CPI của một kiến trúc (với hệ thống bộ nhớ hoàn hảo với 32 bit địa chỉ) là 1.5. Giả sử rằng trễ bộ nhớ là 40 chu trình, 1 chu trình truyền được 4 bytes từ bộ nhớ vào cache. Một khối có 32 bytes. 20% tổng số lệnh là các lệnh truy cập bộ nhớ. Giả sử không cần phải ghi lại từ cache vào bộ nhớ. Xác định penalty của hệ thống. Xác định CPI của máy tính với hệ thống cache sau:

- Cache có miss ratio = 2.9%. Không tăng thời gian của 1 chu trình

- Cache có miss ratio = 2.2%. Thời gian của 1 chu trình tăng 1.2 lần

- Cache có miss ratio = 2.0%. Thời gian của 1 chu trình tăng 1.25 lần

Hệ thống cache nào có hiệu năng cao nhất.

assume CPIexecution=1.5

1 chu trình truyền được 4 bytes từ bộ nhớ vào cache => 1 khối hết 32/4=8 chu trình

Miss penalty=8+40=48

k= mem access/instruction=1+0.2=1.2

Cache 1:

Miss ratio=miss rate =2.9%=0.029

CPI= CPIexecution+k\*miss rate\*miss penalty=1.5+1.2\*0.029\*48=3.1704

Cache 2:

Miss ratio=miss rate =2.2%=0.022

CPI= CPIexecution+k\*miss rate\*miss penalty=1.5+1.2\*0.022\*48=2.7672

Cache 3:

Miss ratio=miss rate =2.0%=0.02

CPI= CPIexecution+k\*miss rate\*miss penalty=1.5+1.2\*0.02\*48=2.652

\* Hệ thống cache có hiệu năng cao nhất là hệ thống cache có thời gian thực hiện 1 lệnh thấp nhất Texec 1 instuction=CPI\*Tc

Ta tính thời gian thực hiện 1 lệnh của 3 hệ thống cache:

Cache 1: T1= CPI \*Tc=3.1704\*Tc

Cache 2: T2=CPI\*Tc\*1.2 =2.7672\*Tc\*1.2=3.32064\*Tc

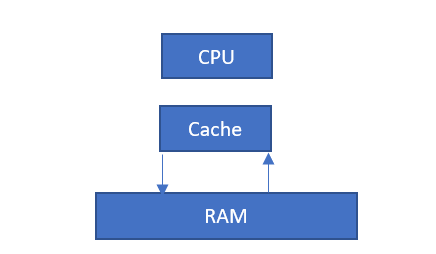
Cache 3: T3=CPI\*Tc\*1.25 =2.652\*Tc\*1.25=3.315\*Tc

=> Cache 1 có hiệu năng cao nhất

8,Câu hỏi giống bài 7 nhưng hệ thống sử dụng write\_back với 50% số block trong cache bị đánh dấu có thay đổi nội dung.

Vì hệ thống sử dụng write\_back với 50% số block trong cache bị đánh dấu có thay đổi nội dung nên có 2 trường hợp:

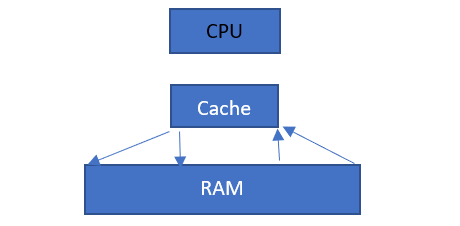
TH1: chiếm 50%



Gặp block trong cache không thay đổi thì chỉ cần xuống đọc RAM rồi đưa lên.

Penalty=read+transfer=40+8=48

TH2: chiếm 50%



Penalty= transfer+write+read+transfer = 8+40+40+8=96

=> miss penalty= 0.5\*48+0.5\*96=72

Cache 1:

Miss ratio=miss rate =2.9%=0.029

CPI= CPIexecution+k\*miss rate\*miss penalty=1.5+1.2\*0.029\*72=4.0056

Cache 2:

Miss ratio=miss rate =2.2%=0.022

CPI= CPIexecution+k\*miss rate\*miss penalty=1.5+1.2\*0.022\*72=3.4008

Cache 3:

Miss ratio=miss rate =2.0%=0.02

CPI= CPIexecution+k\*miss rate\*miss penalty=1.5+1.2\*0.02\*72=3.228

\* Hệ thống cache có hiệu năng cao nhất là hệ thống cache có thời gian thực hiện 1 lệnh thấp nhất Texec 1 instuction=CPI\*Tc

Ta tính thời gian thực hiện 1 lệnh của 3 hệ thống cache:

Cache 1: T1= CPI \*Tc=4.0056\*Tc

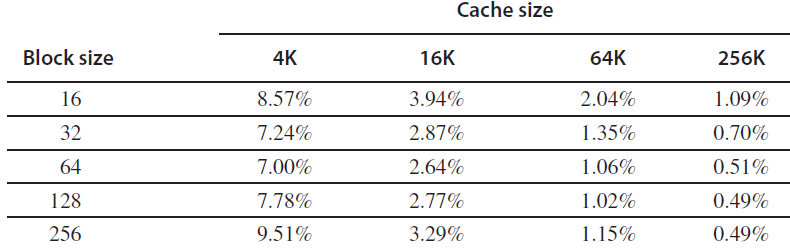
Cache 2: T2=CPI\*Tc\*1.2 =3.4008\*Tc\*1.2=4.08096\*Tc

Cache 3: T3=CPI\*Tc\*1.25 =3.228\*Tc\*1.25=4.035\*Tc

=> Cache 1 có hiệu năng cao nhất

Chương 5\_3:

1,Bảng sau mô tả tỷ lệ lỡ với các kích thước khối và kích thước bộ đệm khác nhau. Giả sử rằng hệ thống có tổng phí để đọc/ghi bộ nhớ là 80 chu trình. Thời gian chạm bộ đệm là 1 chu trình. Bộ nhớ chuyển 16 bytes giữa bộ đệm và bộ nhớ mất 2 chu trình. Như vậy miss penalty cho khối 16 bytes là 82 chu trình, 32 bytes là 84 chu trình . . . Xác định xem kích thước khối tối ưu cho bộ đệm với kích thước khác nhau để có thời gian truy cập trung bình nhỏ nhất.



Thời gian chạm bộ đệm là 1 chu trình => hit time =1

Vì hệ thống có tổng phí để đọc/ ghi bộ nhớ là 80 chu trình; bộ nhớ chuyển 16 bytes giữa bộ đệm và bộ nhớ mất 2 chu trình nên miss penalty cho khối 16 bytes, 32 bytes,64 bytes,128 bytes, 256 bytes lần lượt là 82,84,88,96,112 chu trình.

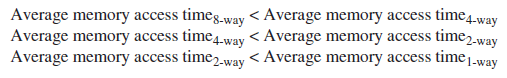
Áp dụng công thức Average memory access time (Ttb)= Hit time+ Miss rate\*Miss penalty ta có bảng sau:

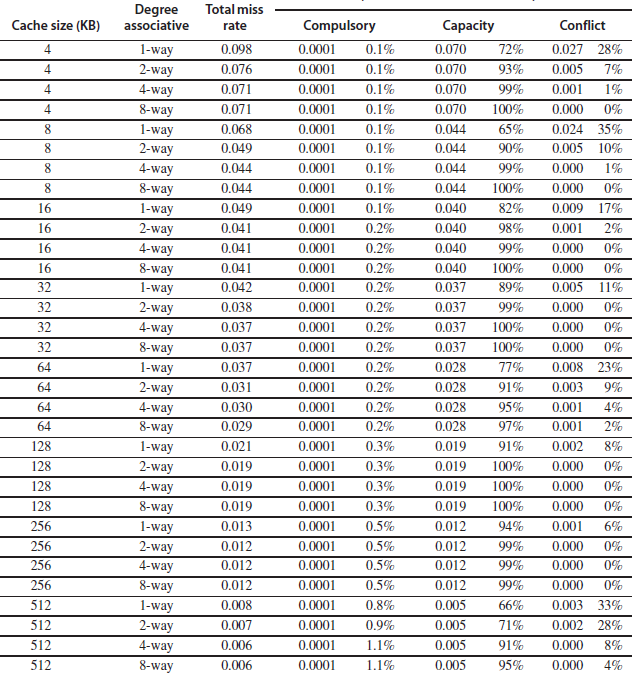
**Cache size**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Block size** | **Miss penalty** | **4K** | | **16K** | | **64K** | | **256K** | |
| **Miss rate** | **Ttb** | **Miss rate** | **Ttb** | **Miss rate** | **Ttb** | **Miss rate** | **Ttb** |
| 16 | 82 | 0.0857 | 8.0274 | 0.0394 | 4.2308 | 0.0204 | 2.6728 | 0.0109 | 1.8938 |
| 32 | 84 | 0.0724 | 7.0816 | 0.0287 | 3.4108 | 0.0135 | 2.134 | 0.07 | 6.88 |
| 64 | 88 | 0.07 | 7.16 | 0.0264 | 3.3232 | 0.0106 | 1.9328 | 0.051 | 5.488 |
| 128 | 96 | 0.0778 | 8.4688 | 0.0277 | 3.6592 | 0.0102 | 1.9792 | 0.049 | 5.704 |
| 256 | 112 | 0.0951 | 11.6512 | 0.0329 | 4.6848 | 0.0115 | 2.288 | 0.049 | 6.488 |

Vậy kích thước tối ưu cho bộ đệm là Block size 16 bytes và Cache size 256K

2, Giả sử tăng số khối trong một tập hợp sẽ làm tăng thời gian chạm. Giả sử thời gian chạm của liên kết trực tiếp là 1 chu trình. Thời gian chạm của 2,4,8 khối trong một tập hợp tương ứng là 1.36, 1.44, 1.52. Miss penalty là 25 chu trình. Bảng tỷ lệ lỡ cho như hình sau. Xác định xem với trường hợp kích thước bộ đệm nào thì mệnh đề sau là đúng.





(1) Average memory access time8-way< Average memory access time4-way

(2) Average memory access time4-way< Average memory access time2-way

(3) Average memory access time2-way< Average memory access time1-way

Miss penalty= 25

Liên kết trực tiếp thì số khối trong 1 tập hợp là 1 nên Hit time 1-way=1

Ta có bảng:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Cache size(KB)** | **Degree associative** | **Total miss rate** | **Miss penalty** | **Hit time** | **Average memory access time** | **Mệnh đề đúng** |  |
|  |
| 4 | 1-way | 0.098 | 25 | 1 | 3.45 | 2  3 |  |
| 4 | 2-way | 0.076 | 25 | 1.36 | 3.26 |  |
| 4 | 4-way | 0.071 | 25 | 1.44 | 3.215 |  |
| 4 | 8-way | 0.071 | 25 | 1.52 | 3.295 |  |
| 8 | 1-way | 0.068 | 25 | 1 | 2.7 | 2 |  |
| 8 | 2-way | 0.049 | 25 | 1.36 | 2.585 |  |
| 8 | 4-way | 0.044 | 25 | 1.44 | 2.54 |  |
| 8 | 8-way | 0.044 | 25 | 1.52 | 2.62 |  |
| 16 | 1-way | 0.049 | 25 | 1 | 2.225 | Không có |  |
| 16 | 2-way | 0.041 | 25 | 1.36 | 2.385 |  |
| 16 | 4-way | 0.041 | 25 | 1.44 | 2.465 |  |
| 16 | 8-way | 0.041 | 25 | 1.52 | 2.545 |  |
| 32 | 1-way | 0.042 | 25 | 1 | 2.05 | Không có |  |
| 32 | 2-way | 0.038 | 25 | 1.36 | 2.31 |  |
| 32 | 4-way | 0.037 | 25 | 1.44 | 2.365 |  |
| 32 | 8-way | 0.037 | 25 | 1.52 | 2.445 |  |
| 64 | 1-way | 0.037 | 25 | 1 | 1.925 | Không có |  |
| 64 | 2-way | 0.031 | 25 | 1.36 | 2.135 |  |
| 64 | 4-way | 0.03 | 25 | 1.44 | 2.19 |  |
| 64 | 8-way | 0.029 | 25 | 1.52 | 2.245 |  |
| 128 | 1-way | 0.021 | 25 | 1 | 1.525 | Không có |  |
| 128 | 2-way | 0.019 | 25 | 1.36 | 1.835 |  |
| 128 | 4-way | 0.019 | 25 | 1.44 | 1.915 |  |
| 128 | 8-way | 0.019 | 25 | 1.52 | 1.995 |  |
| 256 | 1-way | 0.013 | 25 | 1 | 1.325 | Không có |  |
| 256 | 2-way | 0.012 | 25 | 1.36 | 1.66 |  |
| 256 | 4-way | 0.012 | 25 | 1.44 | 1.74 |  |
| 256 | 8-way | 0.012 | 25 | 1.52 | 1.82 |  |
| 512 | 1-way | 0.008 | 25 | 1 | 1.2 | Không có |  |
| 512 | 2-way | 0.007 | 25 | 1.36 | 1.535 |  |
| 512 | 4-way | 0.006 | 25 | 1.44 | 1.59 |  |
| 512 | 8-way | 0.006 | 25 | 1.52 | 1.67 |  |

3, Giả sử trong 1000 truy cập bộ nhớ có 40 lần lỡ ở cache L1 và 20 lần lỡ ở cache L2. Giả sử miss penalty của L2 là 200 chu trình, thời gian chạm L2 là 10 chu trình, thời gian chạm L1 là 1 chu trình. Có 1.5 truy cập bộ nhớ cho 1 lệnh. Bỏ qua ảnh hưởng của việc ghi bộ nhớ.

a) Hãy tính các loại tỷ lệ lỡ (địa phương, toàn cục).

b) Xác định thời gian truy cập bộ nhớ trung bình (Average memory access time)

c) Xác định số trì hoãn bộ nhớ trung bình cho 1 câu lệnh (Average memory stalls per instruction)

a, Tỷ lệ lỡ địa phương:

-) miss rate L1=40/1000=0.04

-) miss rate L2=20/40=0.5

Tỷ lệ lỡ toàn cục = miss rate L1\* miss rate L2=0.04\*0.5=0.02

b, Miss penalty L2=200

Hit time L1=1

Hit time L2=10

Average memory access time

= Hit time L1+ Miss rate L1\*(Hit time L2+Miss rate L2 \*Miss penalty L2)

= 1+0.04\*(10+0.5\*200)

=5.4

c, Ta có: mem access/instruction=1.5

miss per instruction L1=miss rate L1\*mem access/instruction=0.04\*1.5=0.06

miss per instruction L2=miss rate L2\*mem access/instruction=0.5\*1.5=0.75

Average memory stalls per instruction

= miss per instruction L1\* hit time L2+miss per instruction L2\*miss penalty L2

= 0.06\*10+0.75\*200

=150.6

Chương 5\_4

1,Mô tả các khái niệm sau:

a. Trang

b. Lỗi trang

c. Địa chỉ ảo

d. Địa chỉ vật lý

a, Trang : Bộ nhớ được chia thành các khối cố định gọi là trang

b, Lỗi trang là trang bị vỡ ( trang ở trong đĩa)

c, Địa chỉ ảo là địa chỉ trong bộ nhớ ảo được sử dụng CPU

d, Địa chỉ vật lý là địa chỉ trong bộ nhớ vật lý.

2, Một hệ thống sử dụng địa chỉ ảo 32 bit. Một trang có kích thước 16KB. Mỗi mục trong bảng trang có kích thước 4 bytes. Hỏi bảng trang có kích thước bao nhiêu?

Bảng trang chứa virtual page ( trang ảo) và physical page ( trang vật lí)

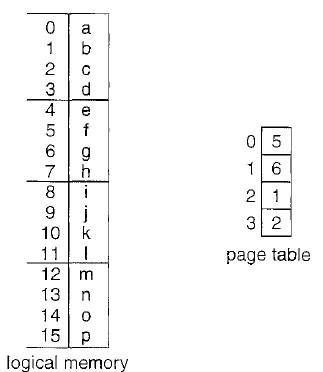
Hệ thống sử dụng địa chỉ ảo 32 bit => Kích thước bộ nhớ ảo là 232 byte

16 KB= 214 byte

Bảng trang có số trang là: 232/214=218 trang

Bảng trang có kích thước là 4\*218=1 MB

3,Xác định địa chỉ vật lý



|  |  |  |
| --- | --- | --- |
|  | **Virtual address** | **Physical address** |
| a | (0,0) | 5\*4+0=20 |
| b | (0,1) | 5\*4+1=21 |
| c | (0,2) | 22 |
| d | (0,3) | 23 |
| e | (1,4) | 6\*4+0=24 |
| f | (1,5) | 25 |
| g | (1,6) | 26 |
| h | (1,7) | 27 |
| i | (2,8) | 1\*4+0=4 |
| j | (2,9) | 5 |
| k | (2,10) | 6 |
| l | (2,11) | 7 |
| m | (3,12) | 2\*4+0=8 |
| n | (3,13) | 9 |
| o | (3,14) | 10 |
| p | (3,15) | 11 |

Chương 6:

1,1 bộ VXL có khả năng xử lý 2 tỷ lệnh/s. Một chương trình người dùng cần 200000 lệnh người dùng và 100000 lệnh OS để thực hiện 1 thao tác I/O đĩa. Hỏi bộ VXL có thể thực hiện tối đa bao nhiêu I/O / s.

Một thao tác I/O cần dùng số lệnh là: 100000+200000=300000 ( lệnh)

Bộ VKL có thể thực hiện tối đa số I/O/s là: 2000000/300000=20/3=6.66

2,Bộ nhớ DDR2 512Mhz có thể đọc/ghi 8 bytes cho một chu trình. Một đĩa cứng đọc, ghi theo khối 32KB. Hỏi đường bộ nhớ này có thể thực hiện tối đa bao nhiêu I/O / s.

512 Mhz = 512000000 chu trình/s

32KB=215 B

Số chu trình cần thực hiện 1 I/O là: 215:8=212

Đường bộ nhớ này có thể thực hiện tối đa số I/O/s là:

512000000:212=125000

3,Một đĩa cứng đọc, ghi theo khối 16KB. Đường I/O bus nối với bộ nhớ có băng thông 1.5 GB/s. Hỏi đường bus này có thể thực hiện tối đa bao nhiêu I/O / s.

1.5 GB=1.5\*220KB

Đường bus này có thể thực hiện tối đa số I/O/s là:

1.5GB: 16KB=98304

4,Bộ quản lý đĩa SCSI có thể truyền dữ liệu với tốc độ 160MB/s. Thời gian xử lý 1 I/O tại đây là 0.1ms. Một đĩa cứng đọc, ghi theo khối 64KB. Hỏi bộ quản lý đĩa này có thể thực hiện tối đa bao nhiêu I/O / s

160 MB/s=160\*210KB/s

Thời gian truyền dữ liệu là:

64: 160\*210=3.9\*10-4s=0.39 ms

Tổng thời gian là: 0.39+0.1=0.49

Bộ quản lý đĩa này có thể thực hiện tối đa số I/O/s là:

1/0.49=2.04

5,Một đĩa cứng đọc, ghi theo khối 64KB. Tốc độ quay của đĩa là 5400 RPM. Tốc độ truyền dữ liệu của đĩa là 50MB/s. Thời gian tìm kiếm trung bình 5ms. Trung bình đĩa cần quay 0.5 vòng để tìm vị trí dữ liệu. Hỏi đĩa này có thể thực hiện tối đa bao nhiêu I/O / s

Tservice=Tcontroller+Tseek+Trotation+Ttransfer

Vì Tcontroller quá bé nên coi bằng 0.

Tốc độ quay của đĩa là 5400 RPM= 90 RPS

Tseek=5ms=0.005s

Trotation=0.5/90=1/180s

Ttransfer=64KB/50MB=64/(50\*2^10)=1.25\*10^-3s

=> Tservice= 0.0118s

Đĩa có thể thực hiện tối đa số I/O/s là:

1/0.0118=84 (I/O/s)

6,Card mạng của máy tính có tốc độ 1Gbps. Một đĩa cứng đọc, ghi theo khối 32KB. Hỏi card mạng này có thể thực hiện tối đa bao nhiêu I/O / s.

Card mạng này có thể thực hiện tối đa số I/O/s là:

1Gbps/32KB=4096

7,1 cluster lưu trữ dữ liệu internet gồm 40 nút lưu trữ kết nối qua switch 1Gbps, mỗi nút có cấu hình như sau:

- CPU VIA 1GHz có thể thực hiện 1 tỷ lệnh/s

- 512MB RAM 266Mhz có thể đọc/ghi 8 bytes cho một chu trình.

- Card mạng 10/100/1000 Mbps

- Bộ điều khiển đĩa PATA tốc độ 133MB/s. T/g xử lý 0.1ms cho 1 I/O.

- 4 đĩa cứng, mỗi đĩa dung lương 500GB, tốc độ quay 7200RPM, thời gian tìm kiếm 8.5ms. Trung bình đĩa cần quay 0.5 vòng để tìm vị trí dữ liệu. Đĩa đọc/ghi khối 16KB.

- Hệ thống cần 50000 lệnh CPU để thực hiện 1 thao tác I/O.

- Giao thức mạng sử dụng 100000 lệnh CPU để truyền 1 khối 16KB từ cluster ra bên ngoài.

- Mỗi nút lưu trữ chưa kể đĩa cứng có giá 500$

- Mỗi đĩa cứng có giá 375$

- Hệ thống mạng có giá 3000$.

Hỏi chi phí cho một I/O trong 1 giây (IOPS) là bao nhiêu.

\*CPU: 10^9/(50000+100000)=6666,67 I/O/s

\*PATA: 1s truyền được: 133\*2^10/16=8512(I/O)

Thời gian truyền 1 khối : 1/8512 (s)

Thời gian thực hiện 1 I/O là: 1/8512+0,1\*10^-3=2.17\*10^-4 s

1s truyền đc : 1:(2.17\*10^-4)=4608 I/O/s

\*RAM: 8 byte-> 1CC

16KB->2048 CC

-> Ghi theo khối 16KB hết 2048 CC

-> 266\*10^6/2048=130\*10^3 I/O/s

\*BUS

Khối 16KB

Băng thông 1,5 GB/s=1572864 KB/s

bus: 1572864/16=98304 I/O/s

\* 4 đĩa cứng

Tservice=......

Tổng 4 đĩa thực hiện tối đa: 4.1/T=308 I/O/s

\*Switch 1Gbps có số I/O/s là:

2^20/(16\*8)=8192(I/O/s)

\*Tính tiền: hệ thống phụ thuộc vào Disk vì số I/O/s nhỏ nhất

=> 40 nút cùng thực hiện 40\*308=12320 I/O/s

Switch 1 Gbps cps số I/O/s là 8192

Tổng chi phí 1 I/O/s là: Tổng số tiền hệ thống/ Số I/O/s=((500+375\*4)\*40+3000)/8192=10,13 $

8,Một máy tính có cấu hình như sau:

* CPU VIA 2 GHz có thể thực hiện được 2 tỷ lệnh/s
* 2048 MB RAM 1.33 GHz có thể đoc/ghi 8 bytes cho 1 chu trình
* Bộ điều khiển đĩa PATA tốc độ 133MB/s. Thời gian xử lý 0.1ms cho 1 I/O
* Hệ thống cần 60000 nghìn lệnh CPU để thực hiện 1 thao tác I/O
* Đường I/O bus nối với bộ nhớ có băng thông 1.5GB/s
* Máy tính sử dụng loại đĩa cứng có dung lượng 500GB, tốc độ quay 7200 RPM, thời gian tìm kiếm 8.5ms, tốc độ truyền dl của đĩa là 50MB/s. Trung bình đĩa cần quay 0.5 vòng để tìm vị trí dữ liệu. Đĩa đọc/ghi khối 16KB

Hỏi ta có thể gắn song song bao nhiêu đĩa cứng để không làm lãng phí lưu lượng

\*CPU: 2 tỷ lệnh/s, 1 I/O/s thực hiện 60000 lệnh

=> Thực hiện tối đa 2\*10^9/60000=33333,33 ( I/O/s)

\*RAM: đọc/ghi khối 16KB hết 16\*2^10/8=2048 ( chu trình)

Số I/O/s là: 1.33\*10^9/2048=649414(I/O/s)

\*PATA:

Thời gian xử lý 0,1ms=0.0001

Thời gian truyền 1 I/O là: 16/(133\*2^10)=1/8512(s)

Thời gian thực hiện 1 I/O là: 0.0001+1/8512=2.17\*10^-4(s)

Bộ điều khiển có thể thực hiện tối đa là:

1/(2.17\*10^-4)=4608(I/O/s)

\*I/O bus: có thể thực hiện tối đa:

1.5\*2^20/16=98304(I/O/s)

\*Disk

Tốc độ quay: 7200 RPM=120RPS

Tservice=Tcontroller+Tseek+Trotation+Ttransfer

Có: Tcontroller=0; Tseek=8,5ms=0.0085s;Trotation=0.5/120=0.0041s

Ttransfer=16KB/50MB=16/(50\*2^10)=3.125\*10^-4(s)

=> Tservice= tổng=0.0129

Số lệnh thực hiện tối đa là 1/0.0129=77 I/O/s

=> Gắn song song được số đĩa là:

4608/77=59