Tài liệu ôn thi học kì

Hệ thống máy tính (Đại học Khoa học Tự nhiên, Đại học Quốc gia Thành phố Hồ Chí Minh)

Phần 1: Tổng quan về máy tính:

⌂ Các thành phần của hệ thống máy tính:

Đơn vị xử lý trung tâm (Central Processor Unit – CPU)

Bộ nhớ chính: RAM & ROM

Hệ thống vào ra (Input-Output System)

Liên kết hệ thống (Buses): bus địa chỉ, bus dữ liệu hoặc bus cục bộ.

⌂ Các thế hệ máy tính:

-1940 - 1956: Vacuum Tubes, ngôn ngữ máy (Ex: ENIAC, UNIVAC, EDVAC, EDSAC) (Đèn chân không)

-1956 - 1963: Transistor, ngôn ngữ Assembly (Ex: FORTRAN, COBOL)

-1964 - 1971: Integrated Circuits (Ex: PDP8, IBM 360, ICL 2900) (bán dẫn)

-1972 - 2010: vi xử lí đầu tiên ra đời(intel 4004), GUI cũm dc phát triển trong gđ này (Ex: IBM, STAR 1000)

-2010 - now: sự ra đời của AI

Các loại máy tính:

Super Computer: xử lí lượng dữ liệu lớn

Work Station:hệ thống một người sử dụng

Microcomputer:cho cá nhân sử dụng

⌂ Cấu tạo:

*Processing devices: mother board, processer, RAM, ROM, switched mode power supply.

Phần cứng	Phần mềm
4 loại: thiết bị vào/ra, Secondary storage device, Internal component.	2 loại: phần mềm ứng dụng, phần mềm hệ thống

Định luật Moore:

Số lượng transistor trên một microchip sẽ gấp đôi mỗi 2 năm (24 tháng)

Khoảng của các biểu diễn:

Không dấu: $0 \rightarrow 2^n - 1$

Dấu lượng: $-(2^{n-1} - 1) \rightarrow 2^{n-1} - 1$

Bù 1: $-(2^{n-1} - 1) \rightarrow 2^{n-1} - 1$

Bù 2: $-2^{n-1} \rightarrow 2^{n-1} - 1$

Số quá: $0 - K \rightarrow 2^{n-1} - K - 1$

==========

Phần 2: Tính toán số nguyên

x SHL y = x . 2y; x SHR y = x / 2y

x AND 0 = 0; x XOR x = 0

AND dùng để tắt bit (AND với 0 luôn = 0)

OR dùng để bật bit (OR với 1 luôn = 1)

XOR, NOT dùng để đảo bit (XOR với 1 = đảo bit đó)

- Lấy giá trị tại bit thứ i của x: (x SHR i) AND 1
- Gán giá trị 1 tại bit thứ i của x: (1 SHL i) OR x
- Gán giá trị 0 tại bit thứ i của x: NOT(1 SHL i) AND x
- Đảo bit thứ i của x: (1 SHL i) XOR x

Vấn đề tràn số:

$x + y < -2^{w-1}$ negative overflow

$x + y > 2^{w-1} - 1$: positive overflow

Biểu diễn nhị phân cho số thực:

**Biểu diễn dấu chấm tĩnh (fixed point number)*

#temporary register $t0=h + A[8]

123.375(decimal) trong hệ 2?

Phần nguyên(8 bits): 123 = 0111 1011

Phần thực(8 bits) 0.375 = 0110 0000

⇒ Result: 0 0111 1011 0110 0000

n = 8 bits → số nguyên Max có thể biểu diễn: 255, phần thực Min có thể biểu diễn: 0.001(2^{-8})

Bị giới hạn → dùng dấu chấm động

**Biểu diễn bằng dấu chấm động*

Biểu diễn dưới dạng: $\pm F.2^E$

IEEE 754: máy tính hiện đại biểu diễn dưới dạng: $V = (-1)^S.F.2^E$

Gồm 3 phần: sign, exponent, significand sign

exponent: lưu dưới dạng n bits, biểu diễn theo quá K

+ độ chính xác đơn (32 bits): K = 127

+ độ chính xác kép(64 bits): K = 1023

significand(fraction)

Ex: -5.25(decimal) → -101.01 = -1.0101 $=2^2$ (E=2)

Exponent: E + 127 = 129 = 1000 0001

Fraction: 0101 0000 ….

Res: 1 1000 0001 0101 ….

***Các loại giá trị chấm động**

Quy ước số 0 là dãy toàn bit 0

- **Dạng không chuẩn:** Phần mũ toàn bit 0, phần trị không toàn bit 0

MAX: (1-2^(-23))*2^(-126)

MIN: 2^(-149)

- **Dạng chuẩn:** Phần mũ không toàn bit 0

MAX: (2-2^(-23))*2^127

MIN: 2^(-126)

- **Số vô cực:** Phần mũ toàn bit 1, phần định trị toàn bit 0

- **Số báo lỗi:** Phần mũ toàn bit 1, phần trị không toàn bit 0

==========

Phần 3: Tổ chức bộ vi xử lý

Các thành phần của bộ vi xử lý: 3

-Control unit : - Dùng để ra các tín hiệu điều khiển cho ALU, giải mã các câu lệnh.

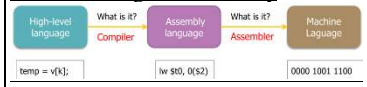
-ALU (Đơn vị toán học luận lí) :

Chịu trách nhiệm trong các nhiệm vụ thực hiện các câu lệnh, phép tính.

Đọc các toán hạng từ Tập thanh ghi, Sau đó hoặc là lưu lại vào Tập thanh ghi , hoặc là ghi vào trong bộ nhớ.

-Registers (Tập thanh ghi)

Cách hoạt động của bộ vi xử lý



**Instructions(lệnh/chỉ thị/ mã máy)*

Là 1 cái chuỗi bit chứa yêu cầu gửi đến CPU (ALU) thực hiện

Gồm 2 thành phần chính :

Opcode (Mã lệnh) : Cho ALU biết thao tác cần thực hiện

```
sw $t0, 48 ($s0)
```

Operand (Toán hạng) : Cho biết ALU biết toán hạng các đối tượng bị tác động bởi thao tác chứa trong mã lệnh

*Có 2 trường phái thiết kế bộ lệnh:

Complete Instruction Set Computer (CISC): bộ lệnh gồm rất nhiều lệnh, từ đơn giản đến phức tạp

Reduced Instruction Set Computer (RISC): bộ lệnh chỉ gồm các lệnh đơn giản

Cisc sẽ nhấn mạnh trên Hardware hơn, Risc thường sử dụng cho các hệ thống nhúng như MIPS

Để chạy những tập tin mã máy cần *Linker & Loader.*

KL: Code => Compiler => Hợp ngữ Assembler => Object file, Linker liên kết những cái Object file đó thành file Thực thi .exe ; File .exe đó sẽ được Loader đưa vào Bộ nhớ để cho CPU đọc instruction để thực hiện.

==========

Phần 4: Hợp ngữ MIPS

-Cấu trúc bộ vi xử lý MIPS

Được xây dựng theo kiến trúc (RISC) với 4 nguyên tắc:

đơn giản, ổn định, nhỏ gọn, xử lý nhanh

Tăng tốc xử lý cho những trường hợp thường xuyên xảy ra

Thiết kế đòi hỏi sự thỏa hiệp tốt

-Cấu trúc cơ bản của chương trình hợp ngữ trên MIPS:

```
        .data           # khai báo các data label (có thể hiểu là các biến)
                        # sau chỉ thị này
label1: <kiểu lưu trữ> <giá trị khởi tạo>
label2: <kiểu lưu trữ> <giá trị khởi tạo>
...
        .text           # viết các lệnh sau chỉ thị này
        .globl <các text label toàn cục, có thể truy xuất từ các file khác>
        .globl main     # Đây là text label toàn cục bắt buộc của program
        ...
main:                   # điểm text label bắt đầu của program
...
```

-Tập thanh ghi

Là đơn vị lưu trữ data duy nhất trong CPU. Trong kiến trúc MIPS: 32 thanh ghi đánh số từ $0 ⇒ $31

truy xuất thanh ghi qua tên, mỗi thanh ghi có kích thước cố định 32 bit

Trong kiến trúc MIPS không tồn tại khái niệm biến, thay vào đó là thanh ghi toán hạng. Ex:

+Save register:

MIPS lấy ra 8 thanh ghi ($16 - $23) dùng để thực hiện các phép tính số học, được đặt tên tương ứng là $s0 - $s7

Tương ứng trong C, để chứa giá trị biến (variable)

+Temporary register:

MIPS lấy ra 8 thanh ghi ($8 - $15) dùng để chứa kết quả trung gian, được đặt tên tương ứng là $t0 - $t7

Tương ứng trong C, để chứa giá trị biến tạm (temporary variable)

Tên	Thanh ghi	Ý nghĩa
$zero	0	Thanh ghi này luôn bằng 0
$at	1	Assembler Temporary
$v0,$v1	2,3	Lưu giá trị trả về của hàm
$a0-$a3	4,7	Lưu tham số truyền vào
$t0-$t7	8,15	Lưu biến tạm
$s0-$s7	16-23	Lưu biến
$t8-$t9	24-25	Như các $t ở trên
$k0,$k1	26, 27	Được dùng cho nhân HDH
$gp	28	Pointer to global area
$sp	29	Stack pointer
$fp	30	Frame pointer
$ra	31	Return address

-Các lệnh số học/logic

Trong MIPS, lệnh thao tác với số nguyên có dấu được biểu diễn dưới dạng bù 2

**Arithmetic operation (Thao tác số học):*

Syntax (R-format):

```
|op(6-bit) rd, rs, rt
```

|op: Tên thao tác

|rd: Thanh ghi đích (chứa kết quả)

|rs: Thanh ghi (toán hạng nguồn 1)

|rt: Thanh ghi(toán hạng nguồn 2)/const

Cộng có dấu: `add $s0, $s1, $s2`

Cộng không dấu:

`addu $s0, $s1, $s2` (u: unsigned)

Cộng với hằng số:

```
addi $s0, $s1, 3
```

Trừ với hằng số:

```
addi $s0, $s1, -3
```

Diễn giải: `$s0 ← $s1 + $s2`

MIPS cung cấp 2 loại lệnh số học:

`add, addi, sub`: Phát hiện tràn số

`addu,addiu,subu`: Ko p.hiện tràn số

Trừ (Subtract) :

Trừ có dấu: `sub $s0, $s1, $s2`

Trừ không dấu: `subu $s0,$s1,$s2`

Diễn giải: `$s0 ← $s1 - $s2`

Gán a = b ⇒ `add $t1, $t0, $zero`

Phép Nhân (Multiply) và Chia (Division)

Kết quả Nhân/Chia sẽ lưu trong cặp 2 thanh ghi là `$hi`*(lưu 32 bit cao (32-63))* và `$lo` *(lưu 32 bit thấp (0-31))*

Trong phép Chia thì *phần Dư* sẽ lưu trong `$hi` và *phần Thương* lưu trong `$lo`

Để truy xuất giá trị trong 2 thanh ghi `$hi` và `$lo` thì dùng 2 cặp lệnh `mflo` (move from lo), `mfhi` (move from hi) - `mtlo` (move to lo), `mthi` (move to high)

Ví dụ : `mflo $s0` ($s0 = $lo) ; `mfhi $s0` ($s0 = $hi)

Phép Nhân có 2 cú pháp :

`mult rs,rt` hoặc `multu rs,rt` (nhân ko dấu)⇒K.quả lưu trong `$hi&$lo`

`mul rd,rs,rt` ⇒ Sẽ lưu 32 bit thấp của kết quả vào `rd`

Phép chia :

Cú pháp : `div rs, rt` / `divu rs, rt` (`divu` dùng cho phép chia ko dấu)

Ý nghĩa là lấy `rs` chia cho `rt` , lưu kết quả vào `$hi` và `$lo`

Lưu ý là phép chia ko có check vấn đề Tràn số hay là Chia cho 0 (mà đó là do phần mềm)

**Dấu chấm động :*

MIPS sử dụng 32 thanh ghi dấu phẩy động (`$f0 - $f31`) để biểu diễn *độ chính xác đơn* của số thực.

Để biểu diễn *độ chính xác kép* thì MIPS sử dụng sự ghép đôi của 2 thanh ghi có độ chính xác đơn.

Và những cái lệnh thao tác với Số chấm động *chỉ hoạt động trên các thanh ghi dấu phẩy động* là `$f*`

Single-precision:

```
add.s $f0, $f1, $f2
```

Double-precision:

```
add.d $f0, $f1, $f2
```

Logical operation (Thao tác logic/Luận lí):

Operation	C	Java	MIPS
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bitwise AND	&	&	and, andi
Bitwise OR	|	|	or, ori
Bitwise NOT	~	~	nor

*Phép toán luận lý :

+Cú pháp : `opt opr, opr1, opr2`

`opt` (operator): Tên thao tác

`opr` (operand): Thanh ghi toán hạng đích (chứa kết quả)

`opr1` (operand 1): Thanh ghi (toán hạng nguồn 1)

`opr2` (operand 2): Thanh ghi (toán hạng nguồn 2) / hằng số

+MIPS hỗ trợ 2 nhóm lệnh cho các phép toán luận lý trên bit :

`and, or, nor`: Toán hạng nguồn thứ 2 (opr2) phải là thanh ghi

`andi, ori`: Toán hạng nguồn thứ 2 (opr2) là hằng số

+Lưu ý: MIPS không hỗ trợ lệnh cho các phép luận lý NOT, XOR, NAND… Vì với 3 phép toán luận lý and, or, nor ta có thể tạo ra tất cả các phép luận lý khác ⇒ Tiết kiệm thiết kế cổng mạch

Ví dụ : not (A) = not (A or 0) = A `nor` 0

*Phép dịch luận lý:

Cú pháp (R-format):

```
op rd, rs, shamt
```

`op`: Tên thao tác

`rd`: Thanh ghi toán hạng đích (chứa kq)

`rs`: Thanh ghi (toán hạng nguồn 1)

`shamt`: Hằng số < 32 (Số bit dịch)

MIPS hỗ trợ 2 nhóm lệnh cho các phép dịch luận lý trên bit :

+Dịch luận lý:

Dịch trái (`sll` – shift left logical): Thêm vào các bit 0 bên phải

Ví dụ : `sll $s0, $s1, 2 # Dịch trái $s1 2bit → lưu vào $s0`

Dịch phải (`srl` – shift right logical): Thêm vào các bit 0 bên trái

Ví dụ : `srl $s0, $s1, 2 # Dịch phải $s1 2bit → lưu vào $s0`

+Dịch số học :

Không có dịch trái số học

Dịch phải (`sra` – shift right arithmetic): Thêm vào bên trái các bit = giá trị bit dấu

Ví dụ : `sra $s1, $s2, 2`

$s2 = 1111 1111 1111 1111 1111 1111 1111 0000 = -16

⇒ $s1 = 1111 1111 1111 1111 1111 1111 1111 1100 = -4 (Sign-extended)

Các lệnh truy xuất bộ nhớ

Di chuyển dữ liệu từ bộ nhớ vào thanh thi, từ cpu vào bộ nhớ,…

*Có 2 thao tác chính :

Load (Lấy dữ liệu từ Bộ nhớ vào reg)

Store (Lưu dữ liệu từ reg vào bộ nhớ)

+ Syntax (I-format):

```
op rt,(constant/address)rs
```

`op` : Tên thao tác (Load / Save)

`rt` : Thanh ghi nguồn (đối với store) / Thanh ghi đích (đối với load) `Constant` : $-2^{15} \rightarrow 2^{15} - 1$ `Address` : offset added to base address in rs (offset is always a multiple of 4)[Độ dời]

`rs` : thanh ghi cơ sở chứa địa chỉ vùng nhớ cơ sở (địa chỉ nền)

`+ lw` (Load Word) : Nạp 1 word, từ bộ nhớ, vào 1 thanh ghi trên CPU

Ví dụ : `lw $t0, 12 ($s0)` ⇒ Nạp 1 word có địa chỉ `($s0 + 12)` chứa vào thanh ghi `$t0`

`+ sw` (Store Word) : Lưu 1 word, từ thanh ghi trên CPU, ra bộ nhớ

Ví dụ : `sw $t0, 12 ($s0)` ⇒ Lưu giá trị trong thanh ghi `$t0` vào ô nhớ có địa chỉ `($s0 + 12)`

`+ $s0` được gọi là thanh ghi cơ sở (base register) thường dùng để lưu địa chỉ bắt đầu của mảng / cấu trúc

Ví dụ : Giả sử A là mảng chứa 100 word với địa chỉ bắt đầu (địa chỉ nền - base address) chứa trong thanh ghi $s0. *Giá* trị của biến g , h lần lượt chứa trong các thanh ghi $s1 và $s2. *[Mỗi phần tử là 1 word = 4 byte ⇒Độ dời sẽ là i * 4]*

+Code trong C : `g = A[2]`

Trong MIPS là gì ? `lw $t0, 8($s0)`

⇒ Load dữ liệu A[2] vào thanh ghi $t0

`add $s1, $t0,$zero` ⇒ Lưu dữ liệu đó vào g

+Code trong C : `g = h + A[8]`

Trong MIPS là gì ?

```
lw $t0, 32($s0)
add $s1, $s2, $t0
```

+Code trong C : `A[12] = h+A[8].`

Trong MIPS là gì ?

```
lw $t0, 32($s0)
```

#load A[8] to the register $t0

```
add $t0, $s2,$t0
```

+*Code trong C :* `A[12] = h - A[8]` *. Trong MIPS là gì ?*

```
lw $t0, 32($s3)
```
Chứa A[8] vào $t0
```
sub $t0, $s2, $t0
sw $t0,48($s3)
```
Kết quả vào A[12]

+ `lb` (Load byte), `sb` (Save byte)

Giả sử nạp 1 byte có giá trị `xzzz zzzz` vào thanh ghi trên CPU (`x`: bit dấu của byte đó hoặc là Most-Significant Bit) bằng lệnh `lb` Thì Giá trị thanh ghi trên CPU (32 bit) sau khi nạp có dạng : `xxxx xxxx xxxx xxxx xxxx xxxx xzzz zzzz`

⇒ Tất cả các bit từ phải sang sẽ có giá trị = bit dấu của giá trị 1 byte vừa nạp (sign-extended)

Nếu muốn các bit còn lại từ phải sang có giá trị không theo bit dấu (=0) thì dùng lệnh : `lbu` (load byte unsigned)

+ Load, Save 2 byte (1/2 Word) ⇒ Load half: lh và Store half: sh

*Rẽ nhánh Có điều kiện :

Syntax (I-format):

`op rs , rt , target address`

`rs` : Thanh ghi (toán hạng nguồn 1)

`rt` : Thanh ghi (toán hạng nguồn 2)

`Target Address` : the address of the next instruction

```
beq opr1, opr2, label
```
=>if (opr1 == opr2) goto label
```
bne opr1, opr2, label
```
⇒ if (opr1 != opr2) goto label

*Rẽ nhánh Không điều kiện :

Syntax (J-format):

`op TargetAddress`

`rs` : the first source register number `rt` : the second source register number

`Target Address`: the address of the next instruction

```
j label #goto label
```
Có thể viết lại thành:
```
 beq $0, $0, label
```
**So sánh Lớn hơn/Bé hơn (Set less than)*

Syntax: slt rd, rs, rt

Nếu (rs < rt) thì rd = 1,ngược lại thì rd = 0

**So sánh với Hằng số (Set less than immediate)*:

Syntax: slti rd, rs, constant

Nếu (rs < constant) thì rd = 1, ngược lại thì rd = 0

Các lời gọi hệ thống (system call)

System Call là chỉ thị yêu cầu đến Hệ điều hành cung cấp những cái dịch vụ cần thiết như là Tương tác đến những thiết bị Nhập/Xuất, hoặc những dịch vụ mà chương trình người dùng không thể thực hiện mà phải nhờ HĐH

Muốn gọi dịch vụ thì :

Gán Code của dịch vụ đó vào thanh ghi `$v0`

Gán các tham số của dịch vụ vào các thanh ghi `$a0` → `$a3` (Nếu là số chấm động thì `$f12`)

Thanh ghi `$v0` lưu trữ kết quả trả về với Số nguyên, Thanh ghi `$f0` đối với Số chấm động

Name	Fields						Comments
Filed size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instruction 32 bits
R-format	op	rs	rt	rd	sham t	funct	Arithmetic instruction format
I-format	op	rs	rt	Address/ Immediate			Transfer, branch, immediate format
J-format	op	Target address					Jump instruction format

==========

Phần 5: Hợp ngữ X86

Size Lệnh từ 1 -16 bytes (lệnh dài I 15)

1 toán hạng vừa là nguồn và là đích

1 toán hạng có thể đến từ bộ nhớ

Mỗi segment có kích thước 64 KB

Overlapped segments là 16 bytes

Tập thanh ghi

*Thanh ghi đoạn 16 bits(segment register): CS, SS, DS, ES

segment	offset	Meaning
CS	IP	Địachỉlệnh
SS	SP or BP	Địachỉstack
DS	BX,DI,SI,8/16bits	Địachỉđoạndữliệu
ES	DI	Địachỉchứadữliệuthêm

*Thanh ghi đa dụng 16 bits(general register): AX, BX, CX, DX

8088/8086 đến 80286: 16 bits

80386 trở lên: 32bits EAX,EBX,…

Y86-64: 64bits RAX, RBX,…

*Thanh ghi cờ: chứa kq tính toán

2 nhóm:

+Trạng thái:

C/CF (carry flag): CF = 1: khi có số nhớ hoặc mượn từ MSB trong phép cộng hoặc trừ (tràn không dấu)

P/PF (parity flag): PF = 1 (0) khi số bit 1 trong kết quả là chẵn (lẻ)

A/AF (auxilary carry flag): giống với CF, nhưng vị trí nhớ ra là bit thứ tư trong nhóm 4 bit

Z/ZF (zero flag): ZF = 1: khi kết quả của một phép tính bằng 0

S/SF (sign flag): SF = 1 khi kết quả phép tính là âm (MSB=1)

O/OF (overflow flag): OF = 1: nếu kết quả vượt quá khả năng tính toán của CPU

+Điều khiển:

T/TF (trap flag): TF = 1: cho phép chương trình chạy từng bước

I/IF (Interrupt enable flag): IF = 1: cho phép ngắt phần cứng

D/DF (direction flag): DF = 1 -> chiều xuất của string từ địa chỉ lớn đến địa chỉ nhỏ.

Cấu trúc lệnh:

Chiều dài tối đa của 1 lệnh X86 là 16 bytes, lệnh dài nhất chỉ chiếm 15 bytes

Tiền tố (4 bytes),Mã lệnh tối đa 2 bytes

Toán hạng 1 byte, Độ dời 4 bytes

Hằng số 4 bytes

Cấu trúc lệnh trong X86

64-bit reg	32-bit reg	16-bit reg	8-bit reg
%rax	%eax	%ax	%al
%rbx	%ebx	%bx	%bl
%rcx	%ecx	%cx	%cl
%rdx	%edx	%dx	%dl
%rsi	%esi	%si	%sil
%rdi	%edi	%di	%dil
%rbp	%ebp	%bp	%bpl
%rsp	%esp	%sp	%spl

	Intel	AT&T
Comments	;	//
instructions	Untagged add	Tagged with operand size: addq
Register	eax, ebx,…	%eax, %ebx,…
Immediate	0x100	$0x100
Operand order	Mnemonic des, scr	Mnemonic scr, des
Indirect	[eax]	(%eax)
General Indirect	[base + reg*scale+displacement]	Displacement(reg,reg,scale)

*Lệnh move dữ liệu

+MOV: sao chép dữ liệu ở toán hạng thứ 2 vào toán hạng thứ 1, Syntax:

Mov <reg/mem>, <reg,mem,const>

+Push: đưa toán hạng vào trong stack

Syntax: push <reg32/mem/con32>

+Pop: bỏ 4 byte data của stack và đưa vào toán hạng chỉ định

Syntax: pop <reg32/mem>

+LEA: nạp địa chỉ

Syntax: lea <reg32> <mem>

*Các lệnh số học/logic

ADD: lưu kq vào operand đầu tiên

Syntax: add<reg/mem><reg/mem/con>

SUB, INC/DEC

Syntax: inc/dec<reg/mem>

iMUL : syntax :

imul <reg32><reg32/mem>(<con>)

iDIV: EAX: res, EDX:remainder,

EDX :EAX / reg/32/mem content

syntax : idiv <reg32/mem>

CMP:

Cmp <reg/mem><reg/mem/con>

- Đích = nguồn: CF=0, ZF=1
- Đích>nguồn: CF=0, ZF=0
- Đích<nguồn: CF=1, ZF=0

AND,OR, XOR(syntax giống cmp)

SHL,SHR: opcode<reg/mem><con8/cl>

Các lệnh truy xuất bộ nhớ

Immediate,Direct,Indirect

Register direct,Register indirect

Indexed

Assume the following are stored as an indicated memory address and register

Address	Value	Register	Value
0x100	0xFF	%rax	0x100
0x104	0xAB	%rcx	0x1
0x108	0x13	%rdx	0x3
0x10C	0x11		

áp dụng công thức ở trên để tính
*LƯU Ý: khi không có scale thì scale = 0
scale chỉ có là 0, 1, 2 ,3

Fill in the following tab showing the value for indicated operands:

Operand	Value
%rax	0x100
0x104	0xAB
$0x108	0x108
(%rax)	0xFF
4(%rax)	0xAB
9(%rax,%rdx)	0x11
260(%rcx,%rdx)	0x13
0xFC(,%rcx,4)	0xFF
(%rax,%rdx,4)	0x11

9(%rax,%rdx)

$0x100 + (2^0 \times 0x3) + 9 = 0x10C$

Các lệnh điều khiển

JMP: short/near/far,

Conditional: JE, JNE,…

LOOP: DEC+CX+JNZ decrement CX while CX!=0

Đoạn code trong X86:

==========

Phần 6 : Mạch Logic

Mạch tổ hợp mỗi output chỉ phụ thuộc vào input của một thời điểm

+3 cách biểu diễn : Bảng chân trị,Sơ đồ mạch,Hàm đại số bool.

+các bước thiết kế :

Bước 1: Lập bảng chân trị

Bước 2: Vẽ bản đồ Karnaugh

Bước 3: Vẽ mạch

Ex: f(x, y, z) = 3 + 5 + 6 + 7

3: 011,5: 101,6: 110,7: 111

co 3 te bao: yz, xz, xy

f = yz + xz + xy

+Một số mạch tổ hợp cơ bản:

Half adder: 2 ngõ vào, 2 ngõ ra

Full adder:3 ngõ vào, 2 ngõ ra

*Mạch mã hóa:*2n input và n output.

Tại 1 thời điểm, chỉ có 1 input có giá trị là 1. Nếu input thứ k có giá trị 1 thì output sẽ trả về kết quả là k.

Mạch giải mã: n input và 2n output.

Nếu input có giá trị là k thì output thứ k sẽ có giá trị là 1.

Mạch dồn(multiplexer)

Có 2^n input, n ngõ điều khiển và 1 output.

Khi s1, s0 có giá trị là 0 thì ngõ i0 quyết định giá trị ngõ ra. Khi s1, s0 có giá trị 1 thì ngõ i1 quyết định giá trị ngõ ra.

Mạch tách(demultiplexer):

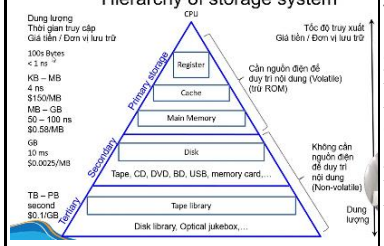
2^n output, n ngõ điều khiển và 1 input.

+ứng dụng: adder/ substractor,ALU…

==========

Phần 7: Bộ nhớ

Mô hình phân cấp bộ nhớ



*Bộ nhớ trong

+Thông tin:

Primary storage/ internal memory:

Thanh ghi, cache, ROM, RAM,...

Cần nguồn điện để duy trì nội dung (trừ ROM)

+Các ví dụ:

Register: Đơn vị lưu trữ nhỏ nhất, có tốc độ truy xuất nhanh nhất Nằm trong CPU: lưu trữ lệnh và dữ liệu được nạp từ bộ nhớ

Được làm bằng mạch tuần tự (flip-flops)

Được tổ chức thành "Register file"

ROM: Bộ nhớ chỉ đọc, bộ nhớ này đã chứa sẵn các chương trình từ trước. Với ROM các dữ liệu được dữ lại kể cả khi máy bị tắt nguồn

PROM - Programmable ROM:

Cho phép lập trình 1 lần sau khi tạo xong

Thường được sử dụng trong hệ thống BIOS của máy tính

EPROM - Erasable PROM: Có thể xóa và lập trình lại mà không cần thay thế chip mới, tiếp xúc với tia UV.

EEPROM - Electrically EPROM: Thay đổi dữ liệu ở mức byte – level, có thể xóa và lập trình lại = cách sử dụng điện tích.

Bộ nhớ flash là một loại EEPROM có mật độ cao hơn và số chu kỳ ghi thấp hơn

=> Cần nhiều thời gian ghi hơn đọc

FlashROM: Xóa ở mức block – level.

Đọc ghi dữ liệu tốc độ cao

RAM

Static RAM – SRAM: mạch lật

Dynamic RAM – DRAM: tụ điện

Main memory: làm từ DRAM,tạo ra bởi các ma trận bit nhớ SDR - SDRAM: một chu kì-một lần dữ liệu,Data bus: 64 bit.DDR - SDRAM (cải tiến của SDR - SDRAM): 1 chu kì-2 lần dữ liệu

+Các phương thức truy cập cache:

CPU và cache là truy xuất theo từ nhớ

Cache vs main truy xuất theo khối nhớ.

Cấu trúc chung gồm 3 phần:

Tag	Index	Byte offset (W)

Các thông tin để sẽ cho: Main memory size, block size = line size; 1 word = ? bytes; cache size.

Các bước tính toán:

1. Main size = 2^k bytes → K bytes address

2. Cache size / Line size = 2^L bytes →L bytes index (Line) (nếu là set associative chia thêm cho số lượng line trong mỗi set)

3. Line size = 2^W bytes → W bytes offset

4. Tag = K – L – W

@ Nếu cho chuyển đổi word=?byte, tính lại W=Line size(tính theo đơn vị word)

Direct mapping: Tag – Line – Word

Associative: Tag – Word

Set associative: Tag – Set – Word.

Thứ tự dò:

Direct: Line → Tag → Word

Associative: Tag → Word

Set associative: Set → Tag → Word.

=>Tỉ lệ cache hit cao, giảm thời gian so sánh, khó thực thi → tiền nhiều

+Cache friendly code

Làm cho những trường hợp phổ biến diễn ra nhanh chóng, Giảm tối thiểu số lượng cache miss bên trong vòng lặp:

Tổng số lượng loads and stores, loops với tỉ lệ hit cao sẽ chạy nhanh hơn⇒sử dụng đối tượng dữ liệu 1 cách liên tục⇒ Đảm bảo tính cục bộ về không gian, đọc đối tượng dữ liệu tuần tự, theo thứ tự chúng được lưu trữ trong bộ nhớ

*Bộ nhớ ngoài

Đĩa mềm(floppy disk)

Đĩa cứng(Hard disk driver)

SSD,Đĩa quang(optical disc),USB,CD

*Bộ nhớ lưu trữ dung lượng lớn

gồm hai hay nhiều ổ đĩa cứng vật lý ghép lại thành 1 đĩa logic

toring data in distributed physical disk

Using parity bits/ check byte to check data errors

RAID types: 0, 1, 0+1, 1+0, 2, 3, 4, 5, 5+0, 6…

EX:

RAID 0: ít nhất 2 ổ đĩa. Tốc độ đọc ghi nhanh (gấp đôi bình thường). Tính an toàn thấp: một đĩa hư thì tất cả đĩa còn lại ko dùng được. 2 ổ cứng phải cùng dung lượng(khác dung lượng thì lấy ổ thấp nhất)

RAID 1: An toàn vì được ghi vào 2 ổ giống nhau, khi 1 trong 2 bị hỏng thì ở còn lại vẫn xài được**.** Hiệu suất thấp, chi phí cao

RAID 1 + 0: 4 ổ cứng(2 ổ striping(raid 0), 2 ổ mirroring(raid 1), an toàn, nhanh, năng suất cao, chi phí cao

RAID 5: dùng kĩ thuật stripe và parity tối thiểu 3 ổ cứng. 1 ổ cứng chết tại 1 thời điểm nếu nhiu ổ cứng chết tại cùng thời điểm thì mất hết dữ liệu. Chi phí thấp hơn Raid 1+0, thao tác chậm

RAID 6: 4 ổ và chịu được 2 ổ đồng thời hỏng bảo mật, thao tác chậm

==========

Phần 8: Hệ thống nhập xuất

Chức năng: trao đổi thông tin giữa máy tính với thế giới bên Ngoài

Do các thiết bị ngoại vi đa dạng, để chậm hơn CPU và RAM → Cần có Mô-đun I/O

Mô-đun I/O: điều khiển, trao đổi thông tin vs CPU, nơi nhớ đệm, phát hiện lỗi

Địa chỉ hóa cổng vào-ra:

-Vào-ra riêng biệt: k.gian đ.chỉ riêng biệt

-Vào-ra theo bản đồ bộ nhớ: đánh địa chỉ theo k.gian đ.chỉ bộ nhớ → truy xuất giống bộ nhớ, lệnh truy xuất bộ nhớ.

Các phương pháp điều khiển:

-Vào-ra (V-R) bằng chương trình:

CPU điều khiển V-R bằng chương trình → Cần lập trình vào ra; 4 tín hiệu đ.khiển:

+Điều khiển: kích hoạt TBNV

+Kiểm tra: kiểm tra trạng thái

+Đọc: TBNV→Đệm module→ CPU

+Ghi: Bus→Đệm module→TBNV

Cách này cần đợi TBNV → Tốn time.

-V-R bằng ngắt:

+CPU gửi tín hiệu đọc – ghi → Không cần chờ TBNV→Nào TBNV sẵn sàng thì gửi tín hiệu ngắt để đọc ghi dữ liệu → Sau đó tiếp tục c.trình đang dừng.

-Truy cập bộ nhớ trực tiếp – DMA

CPU nói cho DMAC (DMA controller) : ra hay vào, đ.chỉ thiết bị, đ.chỉ đầu ngăn nhớ, số từ nhớ → CPU làm việc khác(DMA trao đổi dữ liệu) → ngắt CPU

@DMA không bảo mật do quá trình đọc – ghi ko được CPU quản lý, có thể bị tấn công bằng mã độc.