Tổng hợp Kiến trúc máy tính

I. Các công thức cơ bản

1. Hiệu năng máy tính

- Định nghĩa Hiệu năng P (Performance):

$$\text{Hiệu năng} = \frac{1}{\text{Thời gian thực hiện}}$$

hay là: $P = \frac{1}{t}$

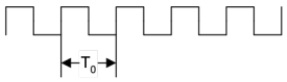
"Máy tính A nhanh hơn máy B k lần"

$$\frac{P_A}{P_B} = \frac{t_B}{t_A} = k$$

- Ví dụ: Thời gian chạy chương trình:
  - 10s trên máy A, 15s trên máy B
  - t_B / t_A = 15s / 10s = 1.5
  - Vậy máy A nhanh hơn máy B 1.5 lần

2. Tốc độ xung nhịp của CPU

- Về mặt thời gian, CPU hoạt động theo một xung nhịp (clock) có tốc độ xác định



- Chu kỳ xung nhịp T_0 (Clock period): thời gian của một chu kỳ
- Tốc độ (tần số) xung nhịp f_0 (Clock speed hay Clock rate): số chu kỳ trong 1s, đo bằng đơn vị Hz
  - $f_0 = 1/T_0$
- Ví dụ: Bộ xử lý có f_0 = 4GHz = 4×10^9Hz
  - $T_0 = 1/(4 \times 10^9) = 0.25 \times 10^{-9}$s = 0.25ns

3. Thời gian thực hiện của CPU

- Để đơn giản, ta xét thời gian CPU thực hiện chương trình (CPU time):

Thời gian thực hiện của CPU =

Số chu kỳ xung nhịp x Thời gian một chu kỳ

$$t_{CPU} = n \times T_0 = \frac{n}{f_0}$$

trong đó: *n* là số chu kỳ xung nhịp

- Hiệu năng được tăng lên bằng cách:
  - Giảm số chu kỳ xung nhịp n
  - Tăng tốc độ xung nhịp f_0

4. Số lệnh và số chu kỳ trên một lệnh

- Số chu kỳ xung nhịp của chương trình:

Số chu kỳ = Số lệnh của chương trình x Số chu kỳ trên một lệnh

$$n = IC \times CPI$$

- n - số chu kỳ xung nhịp
- IC - số lệnh của chương trình (Instruction Count)
- CPI - số chu kỳ trên một lệnh (Cycles per Instruction)

- Vậy thời gian thực hiện của CPU:

$$t_{CPU} = IC \times CPI \times T_0 = \frac{IC \times CPI}{f_0}$$

- Trong trường hợp các lệnh khác nhau có CPI khác nhau, cần tính CPI trung bình

5. CPI trung bình

CPI trung bình

- Nếu các loại lệnh khác nhau có số chu kỳ khác nhau, ta có tổng số chu kỳ:

$$n = \sum_{i=1}^{K} (CPI_i \times IC_i)$$

- Vậy CPI trung bình:

$$CPI_{TB} = \frac{n}{IC} = \frac{1}{IC} \sum_{i=1}^{K} (CPI_i \times IC_i)$$

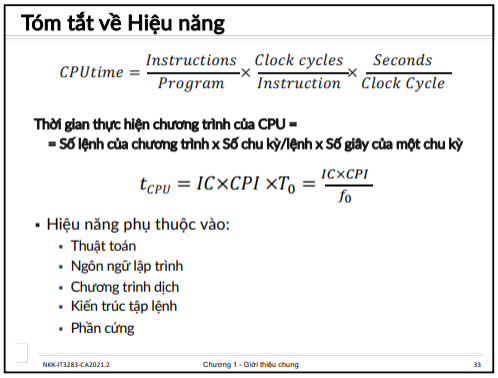
Ví dụ

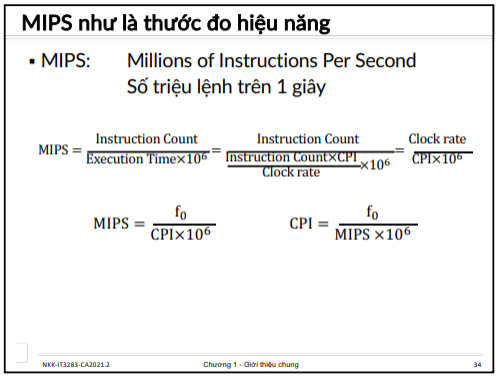
Cho bảng chỉ ra các dãy lệnh sử dụng các lệnh thuộc các loại A, B, C. Tính CPI trung bình?

Loại lệnh	A	B	C
CPI theo loại lệnh	1	2	3
IC trong dãy lệnh 1	20	10	20
IC trong dãy lệnh 2	40	10	10

- Dãy lệnh 1: Số lệnh = 50
  - Số chu kỳ =
  = 1x20 + 2x10 + 3x20 = 100
  - CPI_{TB} = 100/50 = 2.0
- Dãy lệnh 2: Số lệnh = 60
  - Số chu kỳ =
  = 1x40 + 2x10 + 3x10 = 90
  - CPI_{TB} = 90/60 = 1.5

6. Tóm tắt về hiệu năng





II. Hệ thống máy tính

1. Liên kết Bus trong máy tính

- Bus: tập hợp các đường kết nối để vận chuyển thông tin giữa các mô-đun của máy tính với nhau

- Các bus chức năng:

- Bus địa chỉ (Address Bus)
- Bus dữ liệu (Data Bus)
- Bus điều khiển (Control Bus)

- Độ rộng bus: là số đường dây của bus có thể truyền các bit thông tin đồng thời (chỉ dùng cho bus địa chỉ và bus dữ liệu)

1.1 Bus địa chỉ

Chức năng: vận chuyển địa chỉ để xác định vị trí ngăn nhớ hay cổng vào – ra

Độ rộng bus địa chỉ:

- N bit: $A_{N-1}, A_{N-2}, \dots, A_2, A_1, A_0$

⇨ Số lượng địa chỉ tối đa được sử dụng là 2^N địa chỉ (không gian địa chỉ)

- Địa chỉ nhỏ nhất 00…000
- Địa chỉ lớn nhất 11…111

Ví dụ:

- Máy tính sử dụng bus địa chỉ 32-bit, bộ nhớ chính được đánh địa chỉ cho từng byte

⇨ Có khả năng đánh địa chỉ cho 2^{32} bytes nhớ = 4GiB

1.2 Bus dữ liệu

Chức năng:

- Vận chuyển lệnh từ bộ nhớ đến CPU
- Vận chuyển dữ liệu giữa các thành phần của máy tính với nhau

Độ rộng bus dữ liệu: số bit được truyền đồng thời

- M bit: $D_{M-1}, D_{M-2}, \dots, D_2, D_1, D_0$
- M thường là 8, 16, 32, 64 bit

Ví dụ:

- Máy tính có bus dữ liệu kết nối đến CPU với bộ nhớ là 64-bit

⇨ Có thể trao đổi 8 byte nhớ một thời điểm

1.3 Bus điều khiển

Chức năng: Vận chuyển các tín hiệu điều khiển

Các loại tín hiệu điều khiển:

- Các tín hiệu điều khiển đọc/ghi
- Các tín hiệu điều khiển ngắt
- Các tín hiệu điều khiển bus

2. Xử lý các bus dùng chung (shared bus)

Nhiều mô-đun kết nối vào bus chung

⇨ Cần có bộ phân xử bus

Bus chỉ phục vụ được một yêu cầu trao đổi dữ liệu tại một thời điểm

⇨ Độ trễ lớn

Bus phải có tốc độ bằng tốc độ bus của mô-đun nhanh nhất trong hệ thống

Khắc phục:

- Đa bus (Multiple bus): chia thành nhiều bus:
    - Bus cho bộ xử lý
    - Bus cho bộ nhớ
    - Bus vào-ra
- Liên kết điểm-điểm (Point to point interconnection)

2.1 Liên kết điểm-điểm

Kết nối điểm-điểm có độ trễ nhỏ hơn, tốc độ dữ liệu cao hơn và khả năng mở rộng tốt hơn

Các loại kết nối điểm-điểm phổ biến:

- QPI – Quick Path Interconnect
- PCIE- PCI express

III. Kiến trúc tập lệnh

1. Cơ bản

Bộ nhớ máy tính được chia thành nhiều ô mỗi ô dài 1 byte (8 bit), được đánh số từ 0

Mỗi câu lệnh trong MIPS có độ dài là 32-bit (4 byte)

Bộ đếm chương trình PC (Program Counter) là thanh ghi của CPU giữ địa chỉ của lệnh cần nhận vào để thực hiện

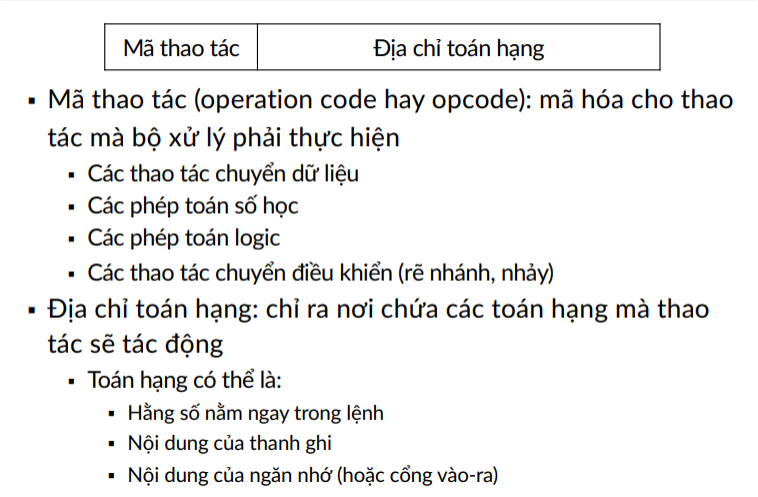
Sau khi lệnh được nhận vào CPU, nội dung PC tự động tăng lên để trỏ sang lệnh kế tiếp, vì mỗi lệnh có độ dài là 32 bit (4 byte) nên PC sẽ tăng lên 4

1 word = 4 byte = 32 bit (chỉ đúng trong MIPS)

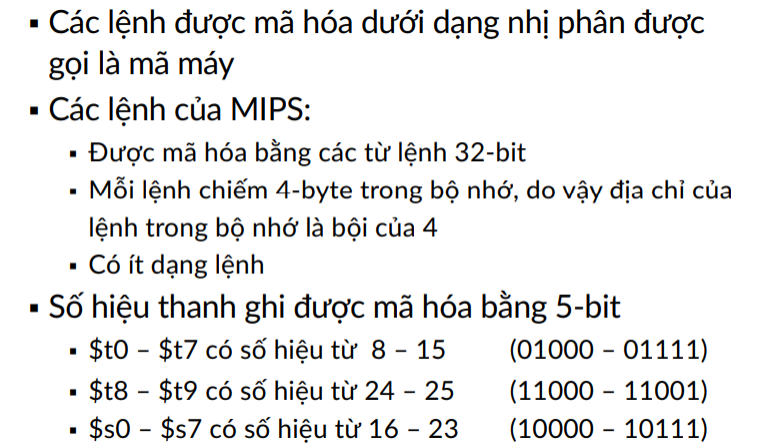
2. Thanh ghi con trỏ

Là thanh ghi chứa giá trị địa chỉ, CPU sẽ dùng nội dung của thanh ghi con trỏ để tìm ra ngăn nhớ dữ liệu cần đọc/ghi

3. Các thành phần của lệnh máy



4. Mã máy (Machine Code)



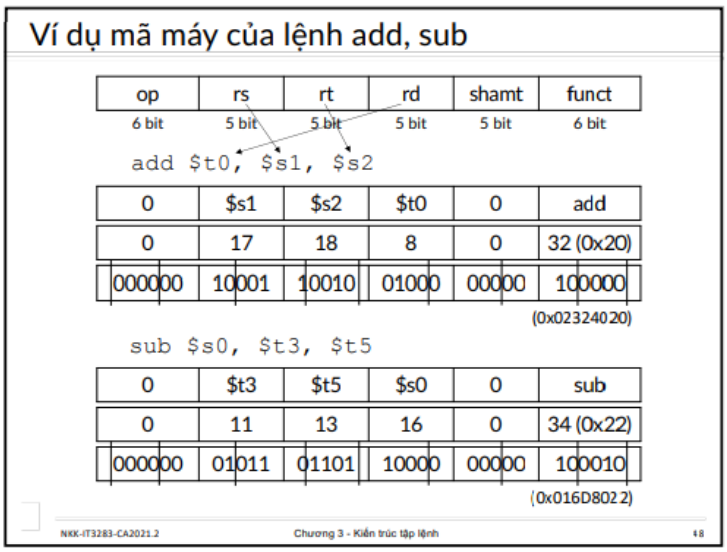
Có 3 kiểu lệnh trong MIPS:

- Lệnh kiểu R (Registers)
- Lệnh kiểu I (Immediate)
- Lệnh kiểu J (Jump)

4.1 Lệnh kiểu R (Registers)

op	rs	rt	rd	shamt	funct
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

- Các trường của lệnh
  - op (operation code - opcode): mã thao tác
    - với các lệnh kiểu R, op = 000000
  - rs: số hiệu thanh ghi nguồn thứ nhất
  - rt: số hiệu thanh ghi nguồn thứ hai
  - rd: số hiệu thanh ghi đích
  - shamt (shift amount): số bit được dịch, chỉ dùng cho lệnh dịch bit, với các lệnh khác shamt = 00000
  - funct (function code): mã hàm → mã hóa cho thao tác cụ thể



4.2 Lệnh kiểu I (Immediate)

Lệnh kiểu I (Immediate)

op	rs	rt	imm
6 bit	5 bit	5 bit	16 bit

- Dùng cho các lệnh số học/logic với toán hạng tức thì và các lệnh load/store (nạp/lưu)
  - Với lệnh addi:
    - rs số hiệu thanh ghi nguồn
    - rt số hiệu thanh ghi đích
  - Với lệnh lw, sw:
    - rs là số hiệu thanh ghi cơ sở
    - rt: số hiệu thanh ghi đích (lw), hoặc số hiệu thanh ghi nguồn (sw)
  - imm (immediate): hằng số nguyên 16-bit

```
addi rt, rs, imm    # (rt) = (rs)+SignExtImm
lw   rt, imm(rs)    # (rt) = mem[(rs)+SignExtImm]
sw   rt, imm(rs)    # mem[(rs)+SignExtImm] = (rt)
```

(SignExtImm: hằng số imm 16-bit được mở rộng có dấu thành 32-bit)

NKK-IT3283-CA2021.2 Chương 3 - Kiến trúc tập lệnh 49

Mở rộng bit cho hằng số theo số có dấu

- Với các lệnh addi, lw, sw cần cộng nội dung thanh ghi với hằng số:
  - Thanh ghi có độ dài 32-bit
  - Hằng số imm 16-bit, cần mở rộng có dấu thành 32-bit (Sign-extended)
- Ví dụ mở rộng có dấu số 16-bit thành 32-bit:

+5 =					0000	0000	0000	0101	16-bit
+5 =	0000	0000	0000	0000	0000	0000	0000	0101	32-bit
-12 =					1111	1111	1111	0100	16-bit
-12 =	1111	1111	1111	1111	1111	1111	1111	0100	32-bit

NKK-IT3283-CA2021.2 Chương 3 - Kiến trúc tập lệnh 50

Ví dụ mã máy của lệnh addi

op	rs	rt	imm
6 bit	5 bit	5 bit	16 bit

```
addi $s0, $s1, 5
```

8	$s1	$s0	5
8	17	16	5
001000	10001	10000	0000 0000 0000 0101

(0x22300005)

```
addi $t1, $s2, -12
```

8	$s2	$t1	-12
8	18	9	-12
001000	10010	01001	1111 1111 1111 0100

(0x2249FFF4)

NKK-IT3283-CA2021.2 Chương 3 - Kiến trúc tập lệnh 51

Ví dụ mã máy của lệnh load và lệnh store

op	rs	rt	imm
6 bit	5 bit	5 bit	16 bit

```
lw $t0, 32($s3)
```

35	$s3	$t0	32
35	19	8	32
100011	10011	01000	0000 0000 0010 0000

(0x8E680020)

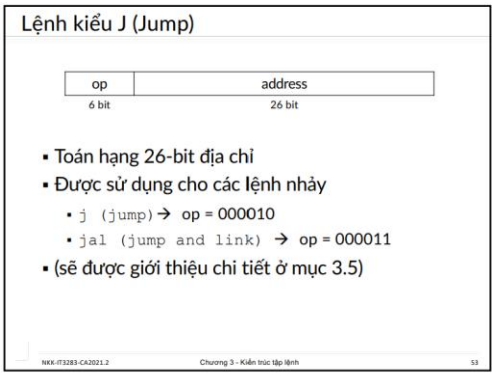
```
sw $s1, 4($t1)
```

43	$t1	$s1	4
43	9	17	4
101011	01001	10001	0000 0000 0000 0100

(0xAD310004)

NKK-IT3283-CA2021.2 Chương 3 - Kiến trúc tập lệnh 52

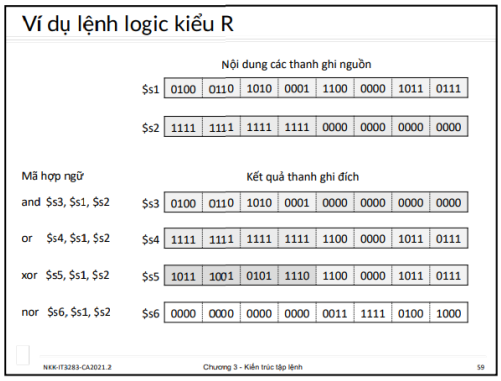
4.3 Lệnh kiểu J (Jump)

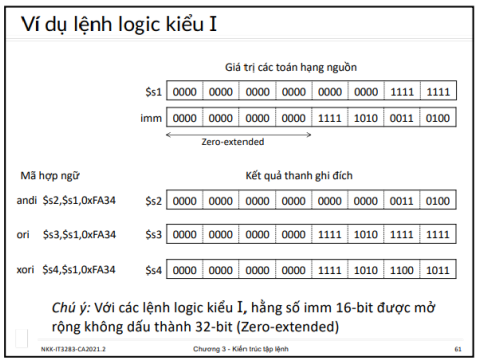


5. Các lệnh logic

Các lệnh logic: Thao tác trên các bit của dữ liệu

Phép toán logic	Toán tử trong C	Lệnh của MIPS
Shift left	<<	sll
Shift right	>>	srl
Bitwise AND	&	and, andi
Bitwise OR	|	or, ori
Bitwise XOR	^	xor, xori
Bitwise NOT	~	nor





6. Các lệnh dịch bit

Các lệnh dịch bit

op	rs	rt	rd	shamt	funct
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

- *shamt*: chỉ ra dịch bao nhiêu vị trí (shift amount)
- *rs*: không sử dụng, thiết lập = 00000
- Lấy nội dung thanh ghi nguồn rt dịch trái hoặc dịch phải rồi cất sang thanh ghi đích rd; rt không thay đổi nội dung
- `sll` - shift left logical (dịch trái logic)
  - Dịch trái các bit và điền các bit 0 vào bên phải
  - Dịch trái *i* bits là nhân với 2^i (nếu kết quả trong phạm vi biểu diễn 32-bit)
- `srl` - shift right logical (dịch phải logic)
  - Dịch phải các bit và điền các bit 0 vào bên trái
  - Dịch phải *i* bits là chia cho 2^i (chỉ với số nguyên không dấu)

Ví dụ lệnh dịch trái sll

Lệnh hợp ngữ:

```
sll $t2, $s0, 4  # $t2 = $s0 << 4
```

Mã máy:

op	rs	rt	rd	shamt	funct
0	0	16	10	4	0
000000	00000	10000	01010	00100	000000

(0x00105100)

Ví dụ kết quả thực hiện lệnh:

$s0	0000	0000	0000	0000	0000	0000	0000	1101	= 13
$t2	0000	0000	0000	0000	0000	0000	1101	0000	=13x16

Chú ý: Nội dung thanh ghi $s0 không bị thay đổi

Ví dụ lệnh dịch phải srl

Lệnh hợp ngữ:

```
srl $s2, $s1, 2  # $s2 = $s1 >> 2
```

Mã máy:

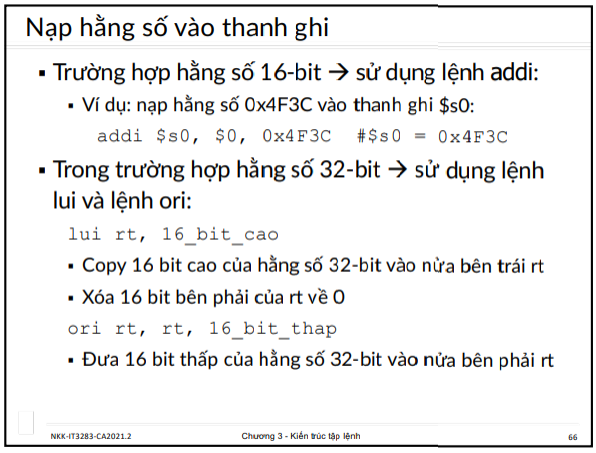
op	rs	rt	rd	shamt	funct
0	0	17	18	2	2
000000	00000	10001	10010	00010	000010

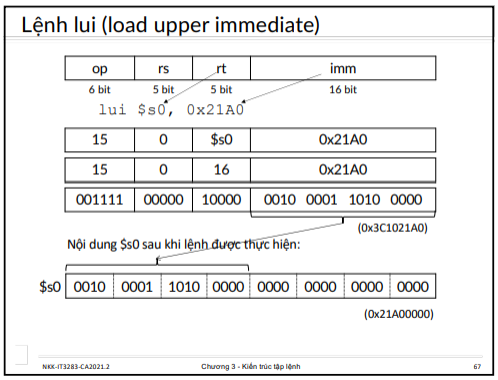
(0x00119082)

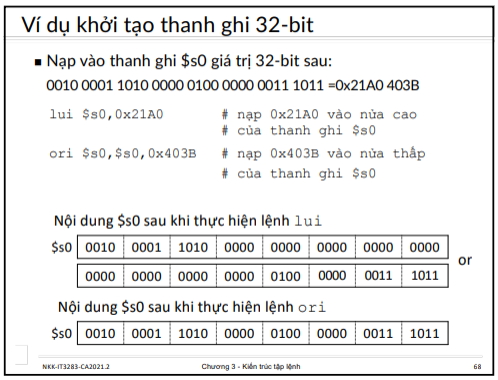
Ví dụ kết quả thực hiện lệnh:

$s1	0000	0000	0000	0000	0000	0000	0101	0110	= 86
$s2	0000	0000	0000	0000	0000	0000	0001	0101	=21 [86/4]

7. Nạp hằng số vào thanh ghi







8. Rẽ nhánh (Easy)

Lưu ý:

- beq, bne là lệnh thật, tất cả các lệnh như blt bge là lệnh giả

- khi sử dụng các lệnh giả thì sẽ dịch thành bne hoặc beq và slt (set less than) hoặc sgt (set greater than)

9. Lập trình mảng (Easy)

9.1 Các thao tác với byte/halfword (8 bit / 16 bit)

Đối với lệnh lw (Nạp word) thì 32 bit từ bộ nhớ luôn được nạp vào đầy đủ, nhưng khi nạp từ bộ nhớ ít hơn 32 bit thì 2 trường hợp sẽ xảy ra:

- Nạp vào có mở rộng dấu
- Nạp vào không mở rộng dấu

Các câu lệnh để nạp dữ liệu từ bộ nhớ vào thanh ghi có mở rộng dấu như sau:

- lb (Load byte)
- lh (Load half word)

Các câu lệnh để nạp dữ liệu từ bộ nhớ vào thanh ghi không mở rộng dấu (zero-extended) như sau:

- lbu (Load unsigned byte)
- lhu (Load unsigned half word)

Ví dụ nạp byte:

```
.data
    number: .word 0x80 # 0000 0000 0000 0000 0000 0000 1000 0000
.text
    lb $t0, number     # 1111 1111 1111 1111 1111 1111 1000 0000
    lbu $t1, number    # 0000 0000 0000 0000 0000 0000 1000 0000
```

Ví dụ nạp half word:

```
.data
    number: .word 0x8000 # 0000 0000 0000 0000 1000 0000 0000 0000
.text
    lh $t0, number       # 1111 1111 1111 1111 1000 0000 0000 0000
    lhu $t1, number      # 0000 0000 0000 0000 1000 0000 0000 0000
```

10. Chương trình con (hàm)

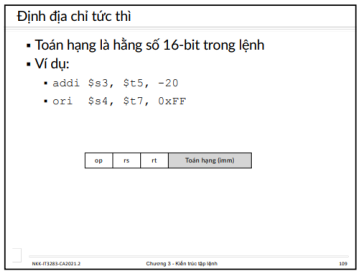
- Thường sẽ sao chép nội dung của các thanh ghi có nội dung cần được nhớ vào Stack và khôi phục khi hàm kết thúc (Nội dung của thanh ghi trước và khi gọi hàm là y hệt nhau – không bị thay đổi)

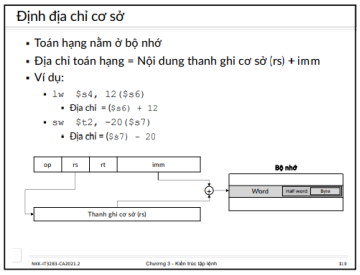
- Nếu hàm lại gọi một hàm khác thì phải lưu cả nội dung của thanh ghi $ra (return address)

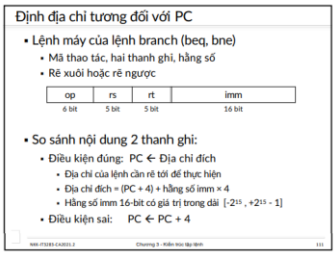
- Nhảy tới hàm bằng câu lệnh jal (jump and link)

11. Tìm lệnh máy của lệnh beq/bne và j, jal

11.1 Tìm hiểu về các loại định địa chỉ

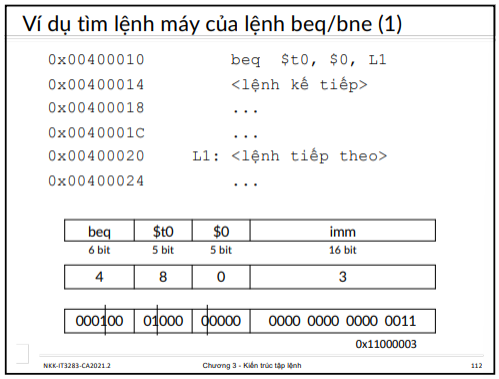


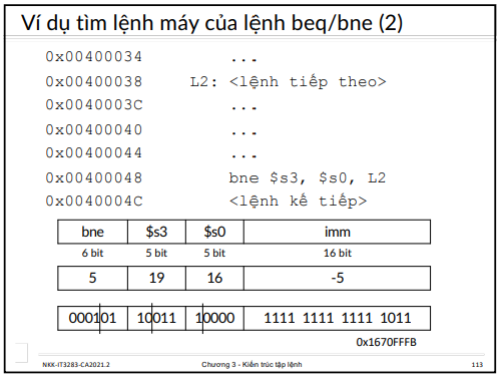




⇨ Các lệnh rẽ nhánh sử dụng định địa chỉ tương đối với PC (Program Counter)

11.2 Cách tìm lệnh máy của lệnh beq/bne





Công thức xác định địa chỉ đích:

Địa chỉ đích = (PC + 4) + imm x 4

Cho lệnh máy ở địa chỉ 0x00400050 là: 0x12110006

000100	10000	10001	0000 0000 0000 0110
4	16	17	6
beq	$s0	$s1	imm

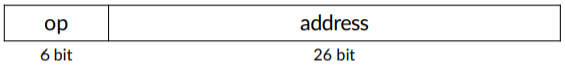
Địa chỉ đích = (PC+4) + imm x 4 = (0x00400050+4) + 6 x 4
= 0x00400054 + 0x18 = 0x0040006C

```
0x00400050      beq   $s0, $s1, L
0x00400054      <lệnh kế tiếp>
...             ...
...             ...
0x0040006C  L:  <lệnh tiếp theo>
```

11.3 Định địa chỉ giả trực tiếp

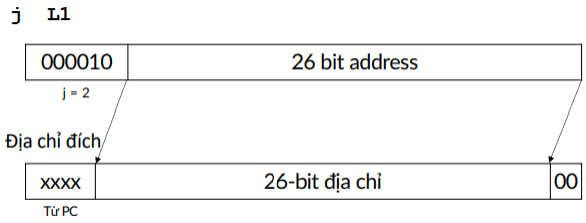
- Đích của lệnh Jump (j và jal) có thể là bất kì chỗ nào trong chương trình

=> Cần mã hóa đầy đủ địa chỉ trong lệnh



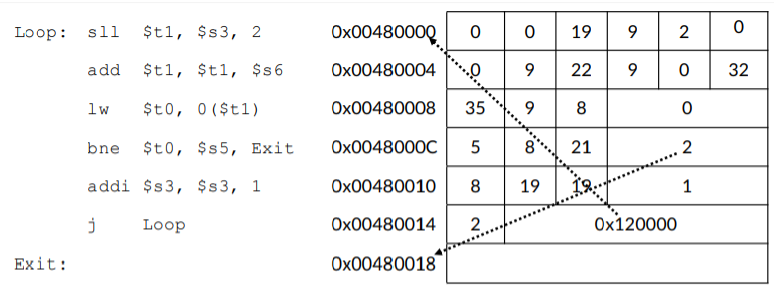
Địa chỉ đích = $PC_{31...28}$: (address): 00

Cách xác định địa chỉ đích:



- Bê nguyên 26 bit ra, thêm 2 bit 0 vào đằng sau, thêm 4 bit high order từ PC vào đằng trước

12. Ví dụ về mã hóa lệnh



sll $t1, $s3, 2	000000 00000 10011 01001 00010 000000	0x00134880
add $t1, $t1, $s6	000000 01001 10110 01001 00000 100000	0x01364820
lw $t0, 0($t1)	100011 01001 01000 0000 0000 0000 0000	0x8d280000
bne $t0, $s5, Exit	000101 01000 10101 0000 0000 0000 0010	0x15150002
addi $s3, $s3, 1	001000 10011 10011 0000 0000 0000 0001	0x22730001
j Loop	000010 00 0001 0010 0000 0000 0000 0000	0x08120000

IV. Số học máy tính

1. Cộng trừ số nguyên không dấu

Chỉ cần nhớ:

- 1 + 0 = 1 nhớ 0
- 0 + 0 = 0 nhớ 0
- 1 + 1 = 0 nhớ 1
- 1 – 1 = 0 nhớ 0
- 0 – 0 = 0 nhớ 0
- 0 – 1 = 1 nhớ 1

2. Cộng số nguyên có dấu

Có 2 trường hợp xảy ra:

- Khi cộng hai số khác dấu thì kết quả luôn đúng
- Khi cộng hai số cùng dấu, nếu kết quả cùng dấu với các số hạng thì kết quả đúng ngược lại thì kết quả sai và bị tràn số

Ví dụ bị tràn:

- (+ 75) = 0100 1011
  +(+ 82) = 0101 0010
  +157 1001 1101
  = - 128+16+8+4+1= -99 → sai

- (- 104) = 1001 1000 (+104=0110 1000)
  + (- 43) = 1101 0101 (+ 43 =0010 1011)
  - 147 1 0110 1101
  = 64+32+8+4+1= +109 → sai
- Cả hai ví dụ đều tràn vì tổng nằm ngoài dải biểu diễn: [-128, +127]

3. Trừ số nguyên có dấu

Phép trừ hai số nguyên: X – Y = X + (-Y)

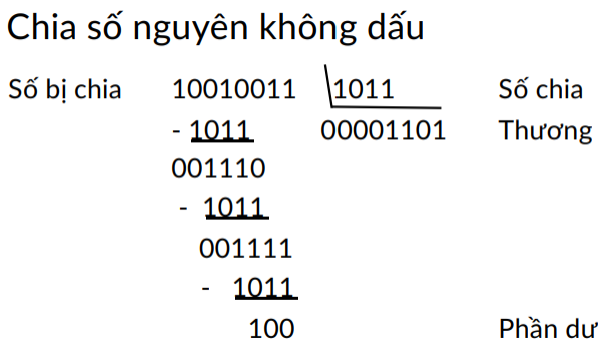
Nguyên tắc : Lấy bù hai của Y để được -Y rồi cộng với X

4. Phép nhân số nguyên không dấu (có dấu thì bỏ)

Nhân số nguyên không dấu

```
      1011      Số bị nhân  (11)
x     1101      Số nhân     (13)
      1011
     0000       Các tích riêng phần
    1011
   1011
--------
  10001111      Tích  (143)
```

5. Phép chia số nguyên không dấu (có dấu thì bỏ)



Trong MIPS : khi nhân 2 số 32 bit thì sẽ được số 64 bit phần low (32 bit low) nằm ở thanh ghi $lo có thể truy cập nội dung bằng mflo, và phần high (32 bit high) nằm ở thanh ghi mfhi, tương tự thì khi chia phần dư ở hi và lo chứa thương

```
mul rd, rs, rt  # tích chỉ trong 32-bit
```

MIPS có hai thanh ghi 32-bit: HI (high) và LO (low)

```
mult rs, rt    # nhân số nguyên có dấu
multu rs, rt   # nhân số nguyên không dấu
```

- Tích 64-bit nằm trong cặp thanh ghi HI/LO

```
div rs, rt     # chia số nguyên có dấu
divu rs, rt    # chia số nguyên không dấu
```

- HI: chứa phần dư, LO: chứa thương

```
mfhi rd      # Move from Hi to rd
mflo rd      # Move from LO to rd
```

6. Số dấu phẩy động

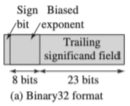
Nguyên tắc chung

- Floating Point Number → biểu diễn cho số thực
- Tổng quát: một số thực X được biểu diễn theo kiểu số dấu phẩy động như sau:

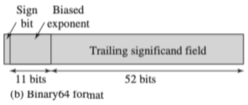
$$X = \pm M * R^E$$

  - M là phần định trị (Mantissa),
  - R là cơ số (Radix),
  - E là phần mũ (Exponent).

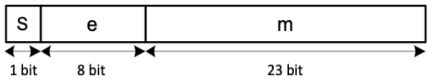
- Cơ số R = 2
- Các dạng:
  - Dạng 32-bit



  - Dạng 64-bit



  - Dạng 128-bit



  - S là bit dấu:
    - S = 0 → số dương
    - S = 1 → số âm
  - e (8 bit) là giá trị dịch chuyển của phần mũ E:
    - e = E + 127 → phần mũ E = e - 127
  - m (23 bit) là phần lẻ của phần định trị M:
    - M = 1.m
  - Công thức xác định giá trị của số thực:

$$X = (-1)^S * 1.m * 2^{e-127}$$

Ví dụ 1

Xác định giá trị của các số thực được biểu diễn bằng 32-bit sau đây:

1100 0001 0101 0110 0000 0000 0000 0000

- S = 1 → số âm
- e = $1000\ 0010_{(2)} = 130_{(10)}$ → E = 130 - 127 = 3

Vậy

$X = -1.10101100_{(2)} * 2^3 = -1101.011_{(2)} = -13.375_{(10)}$

0011 1111 1000 0000 0000 0000 0000 0000 = ?

Biểu diễn số thực $X= 83.75_{(10)}$ về dạng số dấu phẩy động IEEE754 32-bit

Giải:

- $X = 83.75_{(10)} = 1010011.11_{(2)} = 1.01001111 \times 2^6$
- Ta có:
  - S = 0 vì đây là số dương
  - $e = E + 127 = 6 + 127 = 133_{(10)} = 1000\ 0101_{(2)}$
- Vậy:

  X = 0100 0010 1010 0111 1000 0000 0000 0000

- Các bit của e bằng 0, các bit của m bằng 0, thì $X = \pm 0$
  x000 0000 0000 0000 0000 0000 0000 0000 → $X = \pm 0$
- Các bit của e bằng 1, các bit của m bằng 0, thì $X = \pm \infty$
  x111 1111 1000 0000 0000 0000 0000 0000 → $X = \pm \infty$
- Các bit của e bằng 1, còn m có ít nhất một bit bằng 1, thì nó không biểu diễn cho số nào cả (NaN - not a number)

Bài tập :

1. Biểu diễn các số nguyên có dấu theo mã bù 2 8-bit và 16-bit sau đó chuyển về dạng số Hexa

a. +104 b. -43 c. +1041 d. -528 e. -1

a. +104

8-bit :

+104 = 64 + 32 + 8

⇨ $+104_{10} = 0110\ 1000_2$ = 0x68

16-bit : Tương tự trên chỉ cần mở rộng sign-bit (bit ngoài cùng của 8 bit)

⇨ $+104_{10} = 0000\ 0000\ 0110\ 1000_2$ = 0x0068

b. -43

Tìm biểu diễn của 43 theo mã bù 2 trước :

8 bit :

43 = 32 + 8 + 2 + 1

$$43_{10} = 0010\ 1011_2$$

Đảo bit và cộng 1 thì được -43

$$-43_{10} = 1101\ 0101_2$$

Viết ở hexa

0xD5

16 bit :

Tương tự như trên hoặc có thể lấy kết quả của 8 bit mở rộng dấu ra là được

$$-43_{10} = 1111\ 1111\ 1101\ 0101_2$$

Viết ở hexa

0xFFD5

c. +1041

8 bit :

+1041 nằm ngoài dải biểu diễn của số có dấu 8 bit (-128 – 127) nên không biểu diễn được

16 bit :

1041 = 1024 + 16 + 1

$$1041_{10} = 0000\ 0100\ 0001\ 0001_2$$

Viết ở hexa :

0x0411

d. -528

8 bit :

-528 nằm ngoài dải biểu diễn của số có dấu 8 bit (-128 – 127) nên không biểu diễn được

16 bit :

Tìm biểu diễn của 528

528 = 512 + 16

$$528_{10} = 0000\ 0010\ 0001\ 0000_2$$

Đảo bit

⇨ 1111 1101 1110 1111

Cộng 1

⇨ 1111 1101 1111 0000

Vậy :

$$-528_{10} = 1111\ 1101\ 1111\ 0000_2$$

Hexa :

0xFDF0

e. -1

Đơn giản

8 bit : 1111 1111

hexa : 0xFF

16 bit : 1111 1111 1111 1111

hexa : 0xFFFF

*Ghi nhớ : -1 bất cứ độ dài bao nhiêu bit đều toàn số 1

2. Giả sử i, j, k là các biến số nguyên có dấu 8-bit. Cho đoạn chương trình sau :

i = -93 ;

j = -78 ;

k = i + j ;

a. Tìm biểu diễn của i và j dưới dạng nhị phân theo mã bù 2

b. Tính k theo nhị phân và cho biết kết quả của k nhận được dưới dạng thập phân. Giải thích tại sao có kết quả đó

Bài làm

*Tìm biểu diễn i dưới dạng nhị phân theo mã bù 2

i = -93

Bước 1 : Tìm biểu diễn của 93

93 = 64 + 16 + 8 + 4 + 1

⇨ $93_{10} = 0101\ 1101_2$

Đảo bit cộng 1 là ra -93

⇨ $-93_{10} = 1010\ 0011_2$

*Tìm biểu diễn j dưới dạng nhị phân theo mã bù 2

j = -78

Tương tự như i ta có biểu diễn của j dưới dạng nhị phân theo mã bù 2 là

$$-78 = 1011\ 0010_2$$

b. Tính k

k = i + j = 1010 0011 + 1011 0010 = 1 0101 0101 (số 1 ngoài cùng bị nhớ ra ngoài)

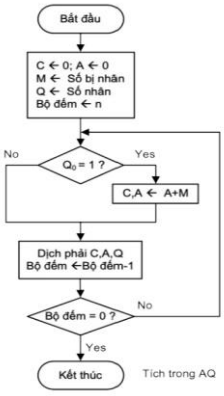
k thực chất là 0101 0101

Chuyển sang thập phân $0101\ 0101_2 = 85_{10}$

Điều này xảy ra do -93 + -78 = -171 nằm ngoài dải biểu diễn 8 bit nên kết quả có được bị sai

3. Nhân hai số sau đây theo thuật giải nhân số nguyên không dấu 8-bit :

M x Q = 25 x 18 = 18 x 25



a. 25x18

$$25_{10} = 0001\ 1001_2$$

$$18_{10} = 0001\ 0010_2$$

Số bị nhân M = 0001 1001

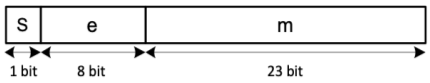
Số nhân Q = 0001 0010

Bộ đếm n = 8

C	A	Q	n	
0	0000 0000	0001 0010	8	
0	0000 0000	0000 1001	7	(Chỉ dịch phải do Q ngoài cùng = 0)
	+0001 1001			
0	0001 1001	0000 1001	7	(A = A + M do Q ngoài cùng = 1)
0	0000 1100	1000 0100	6	(Dịch phải)
0	0000 0110	0100 0010	5	(Chỉ dịch phải do Q ngoài cùng = 0)
0	0000 0011	0010 0001	4	(Chỉ dịch phải do Q ngoài cùng = 0)
	+0001 1001			
0	0001 1100	0010 0001	4	(A = A + M do Q ngoài cùng = 1)
0	0000 1110	0001 0000	3	(Dịch phải)
0	0000 0111	0000 1000	2	(Dịch phải)
0	0000 0011	1000 0100	1	(Dịch phải)
0	0000 0001	1100 0010	0	(Dịch phải) (Kết thúc vì n = 0)

b. 18x25 làm tương tự trên

4. Biểu diễn các số thực sau đây về dạng số dấu phẩy động IEEE754-2008 32-bit viết theo dạng số Hexa



- S là bit dấu:
  - S = 0 → số dương
  - S = 1 → số âm
- e (8 bit) là giá trị dịch chuyển của phần mũ E:
  - e = E + 127 → phần mũ E = e - 127
- m (23 bit) là phần lẻ của phần định trị M:
  - M = 1.m
- Công thức xác định giá trị của số thực:

$$X = (-1)^S * 1.m * 2^{e-127}$$

X = 450

B1: Tìm phần nguyên

$$450_{10} = 1\ 1100\ 0010_2$$

B2: Tìm phần thập phân

$$0_{10} = 0_2$$

B3: Nhân với 2^n sao cho số nhị phân chỉ còn 1 số 1 đằng trước dấu phẩy

$$1\ 1100\ 0010 = 1,1100\ 0010 * 2^8$$

⇨ m = 1100 0010 0000 0000 0000 000 (Mở rộng thành 23 bit)

B4: Tìm e

$$e = 8 + 127 = 135_{10}$$

Biểu diễn e ở nhị phân 8 bit:

$$e = 1000\ 0111_2$$

B5: Tìm S (Sign – dấu)

450 là số dương => S = 0

B6: Ghép tất cả lại với nhau theo format ở ảnh trên

$$0\ 1000\ 0111\ 1100\ 0010\ 0000\ 0000\ 0000\ 000$$

Phân chia lại theo nhóm 4 bit một để chuyển sang hexa:

$$0100\ 0011\ 1110\ 0001\ 0000\ 0000\ 0000\ 0000 = 0x43E10000$$

Y = -46.5

Z = 1/32

Bài này để ý kĩ thì thấy

Z = 1×2^{-5}

Tạm thời ta sẽ không quan tâm tới 2^{-5}

B1: Tìm phần nguyên

$$1_{10} = 1_2$$

B2: Phần thập phân

Không có

B3: Nhân 2^n sao cho đằng trước dấu phẩy chỉ còn 1,

Không cần nhân gì cả nó là 1, sẵn rồi

Tại đây ta thêm 2^{-5}

Số hiện tại là :$1,0 \times 2^{-5}$

⇨ m = 0000 0000 0000 0000 0000 000 (Mở rộng thành 23 bit)

B4 : Tìm e

e = -5 + 127 = 122

Biểu diễn dưới nhị phân 8 bit

$$e = 0111\ 1010$$

B5 : Tìm Sign (Dấu)

Là số dương => S = 0

B6 : Ghép lại

0 0111 1010 0000 0000 0000 0000 0000 000

Nhóm 4 để chuyển thành hexa

0011 1101 0000 0000 0000 0000 0000 0000

Hexa :

0x3D000000

V = 0.2

B1 : Tìm phần nguyên

$$0_{10} = 0_2$$

B2 : Tìm phần thập phân

1) 0.2 x 2 = 0 + 0.4

2) 0.4 x 2 = 0 + 0.8

3) 0.8 x 2 = 1 + 0.6

4) 0.6 x 2 = 1 + 0.2

Ta nhận thấy là nó không bao giờ kết thúc (lặp vô hạn và có quy luật)

Quy luật : 0011 0011 0011 0011

Ta chỉ cần lấy đủ bit (quá 23 bit là okay)

Phần thập phân :

$$0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 00$$

B3 : Nhân với 2^n sao cho đằng trước dấu phẩy là số 1

Kết hợp B1 và B2 ta có số là

$$0{,}0011\ 0011\ 0011\ 0011\ 001\ 1001\ 100 = 1{,}1001\ 1001\ 1001\ 1001\ 1001\ 100 \times 2^{-3}$$

⇨ m = 1001 1001 1001 1001 1001 100

B4 : Tìm e

e = -3 + 127 = 124 = $0111\ 1100$

B5 : Tìm dấu

Số dương => S = 0

B6 : Ghép lại

$$0\ 0111\ 1100\ 1001\ 1001\ 1001\ 1001\ 1001\ 100$$

Chuẩn hóa

$$0011\ 1110\ 0100\ 1100\ 1100\ 1100\ 1100\ 1100$$

Hexa :

$$0x3E4CCCCC$$

5. Cho các số dấu phẩy động theo chuẩn IEEE754 32-bit được viết theo dạng số Hexa như dưới đây. Hãy xác định giá trị của chúng theo dạng thập phân:

M = 0xC1E0 0000

B1 : Đổi thành nhị phân

0xC1E0 0000 = 1100 0001 1110 0000 0000 0000 0000 0000

B2 : Bit đầu tiên là 1 => số âm

B3 : 8 bit tiếp theo : 1000 0011 = 128 + 2 + 1 = 131

⇨ n = e – 127 = 131 – 127 = 4

B4 :23 bit còn lại : 1100 0000 0000 0000 0000 000

Ta có số là :

$$1{,}1100\ 0000\ 0000\ 0000\ 0000\ 000 \times 2^4 = 1\ 1100{,}0000\ 0000\ 0000\ 0000\ 000$$

B5 : Chuyển đổi thành thập phân :

$$28{,}0$$

B6 : Thêm dấu :

$$-28$$

N = 0x3F50 0000

B1 : Đổi thành nhị phân

0x3F50 0000 = 0011 1111 0101 0000 0000 0000 0000 0000

B2 : Bit đầu tiên là 0 => Số dương

B3 : 8 bit tiếp theo 0111 1110 = 64 + 32 + 16 + 8 + 4 + 2 = 126

⇨ n = e – 127 = 126 – 127 = -1

B4 :23 bit còn lại : 1010 0000 0000 0000 0000 000

Ta có số là :

$$1{,}1010\ 0000\ 0000\ 0000\ 0000\ 000 \times 2^{-1} = 0{,}1101\ 0000\ 0000\ 0000\ 0000\ 0000$$

B5 : Chuyển sang thập phân :

$$0{,}8125$$

B6 : Thêm dấu :

$$0{,}8125$$

P = 0x4000 0000

B1 : Chuyển sang nhị phân

0x4000 0000 = 0100 0000 0000 0000 0000 0000 0000 0000

B2 : Bit ngoài cùng = 0 => số dương

B3 : 8 bit tiếp theo

1000 0000 = 128

⇨ n = e – 127 = 128 – 127 = 1

B4 : 23 bit còn lại

Toàn 0

Ta có số là

$$1{,}0000\ 0000\ 0000\ 0000\ 0000\ 000 \times 2^{1} = 10{,}0000\ 0000\ 0000\ 0000\ 0000$$

B5 : Chuyển sang thập phân

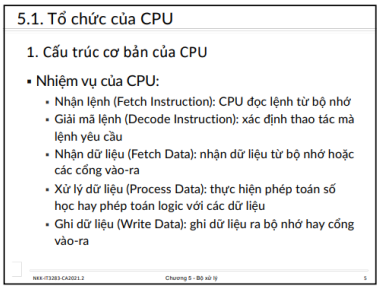
2

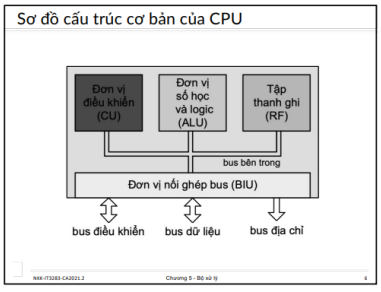
B6 : Thêm dấu

2

V. Bộ xử lý máy tính

1. Cấu trúc cơ bản của CPU

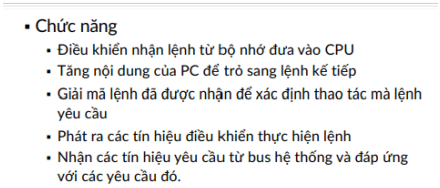




2. Đơn vị số học và logic (ALU)

- Thực hiện các phép toán số học và logic

3. Đơn vị điều khiển



4. Hoạt động của chu trình lệnh

Nhận lệnh => Giải mã lệnh => Nhận toán hạng => Thực hiện lệnh => Cất toán hạng => Ngắt

4.1 Tính địa chỉ của lệnh

- Bộ đếm chương trình PC chứa địa chỉ của lệnh được nhận vào
- Với MIPS :
    - Tuần tự : PC = PC + 4
    - Rẽ nhánh (đk đúng) PC = (PC + 4) + imm x 4 (lệnh bne, beq)
    - Nhảy : PC = PC_{31-28} : (26 bit địa chỉ) : 00 (lệnh j, jal)

4.2 Nhận lệnh

Bước 1 : CPU đưa địa chỉ của lệnh cần nhận từ bộ đếm chương trình PC ra bus địa chỉ đến bọ nhớ để xác định ngăn nhớ chứa lệnh đó

Bước 2 : CPU phát tín hiệu điều khiển đọc bộ nhớ

Bước 3 : Lệnh từ bộ nhớ được đặt lên bus dữ liệu và được CPU copy vào thanh ghi lệnh

Bước 4 : CPU tăng nội dung để trỏ vào lệnh kế tiếp

4.3 Giải mã lệnh

- Lệnh từ thanh ghi lệnh IR được đưa đến đơn vị điều khiển

- Đơn vị điều khiển tiến hành giải mã lệnh để xác định thao tác phải thực hiện

- Giải mã lệnh xảy ra bên trong CPU

4.4 Nhận dữ liệu từ bộ nhớ

- CPU đưa địa chỉ của toán hạng ra bus địa chỉ để xác định ngăn nhớ chứa dữ liệu cần nhận

- CPU phát tín hiệu điều khiện đọc

- Toán hạng được đọc vào CPU

- Tương tự như nhận lệnh

4.5 Thực hiện lệnh

- Có nhiều dạng tùy thuộc vào lệnh
- Có thể là :
    - Đọc/ghi bộ nhớ
    - Vào/Ra
    - Chuyển giữa các thanh ghi
    - Phép toán số học/logic
    - Chuyển điều khiển (rẽ nhánh)

4.6 Ghi toán hạng

- CPU đưa địa chỉ ra bus địa chỉ
- CPU đưa dữ liệu cần ghi ra bus dữ liệu
- CPU phát tín hiệu điều khiển ghi
- Dữ liệu trên bus dữ liệu được copy đến vị trí xác định

4.7 Ngắt

- Nội dung của bộ đếm chương trình PC (địa chỉ trở về sau khi ngắt) được đưa ra bus dữ liệu
- CPU đưa địa chỉ (thường được lấy từ con trỏ ngăn sếp SP) ra bus địa chỉ
- CPU phát tín hiệu điều khiển ghi bộ nhớ
- Địa chỉ trở về trên bus dữ liệu được ghi ra vị trí xác định (ở ngăn xếp)
- Địa chỉ lệnh đầu tiên của chương trình con điều khiển ngắt được nạp vào PC

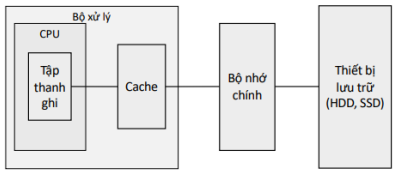
Đọc thêm Pipelining (đường ống)

VI. Bộ nhớ máy tính

1. Các đặc trưng bộ nhớ (Mang tính chất tham khảo)

- Vị trí :
  - Bên trong CPU :
    - Tập thanh ghi
  - Bộ nhớ trong :
    - Bộ nhớ chính (RAM)
    - Bộ nhớ đệm (cache)
  - Bộ nhớ ngoài :
    - Các thiết bị lưu trữ
- Dung lượng :
  - Độ dài từ nhớ (tính bằng bit)
  - Số lượng từ nhớ (1 word = 4 byte trong MIPS)
- Hiệu năng:
  - Thời gian truy nhập
  - Chu kỳ nhớ
  - Tốc độ truyền
- Kiểu vật lý:
  - Bộ nhớ bán dẫn
  - Bộ nhớ từ
  - Bộ nhớ quang
- Các đặc tính:
  - Khả biến/Không khả biến (volatile / nonvolatile – mất nội dung khi mất điện / không mất nội dung khi mất điện)
  - Xóa được / không xóa được
- Tổ chức

2. Phân cấp bộ nhớ



3. RAM (Bộ nhớ chính)

Tưởng tượng RAM như một dải băng vô tận được đánh số từ 0 – infinity

Mỗi ô nhớ là 1 byte (8 bit)

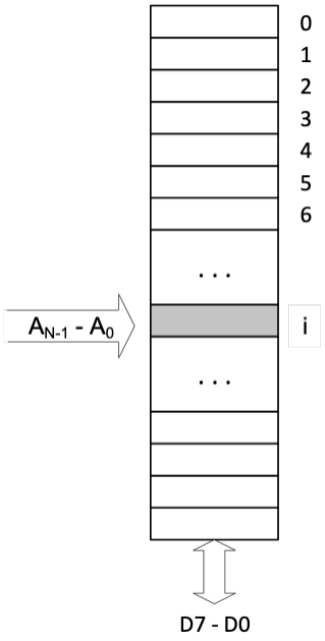
3.1 Tổ chức bộ nhớ đan xen

- Độ rộng của bus dữ liệu để trao đổi với bộ nhớ:

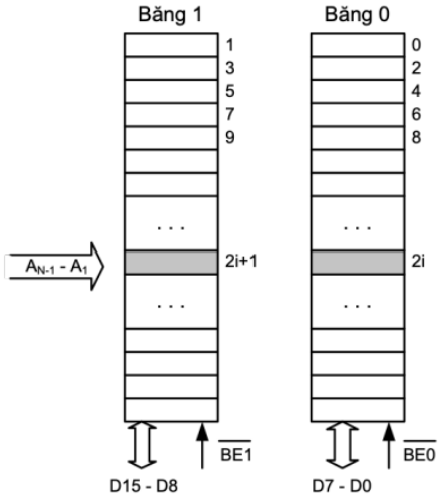
$$m = 8, 16, 32, 64, 128,\ldots \text{ bit}$$

- Các ngăn nhớ được tổ chức theo byte => Tổ chức bộ nhớ vật lý khác nhau

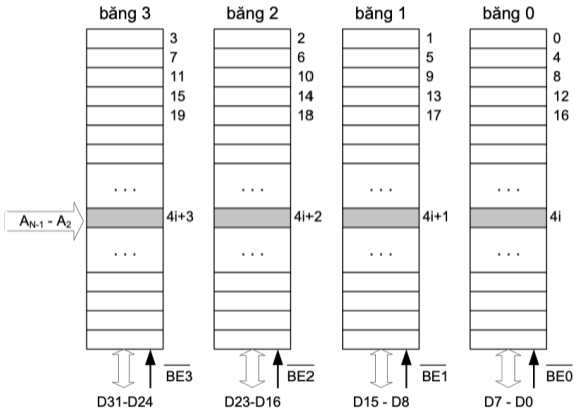
*m = 8 bit => Một băng nhớ tuyến tính



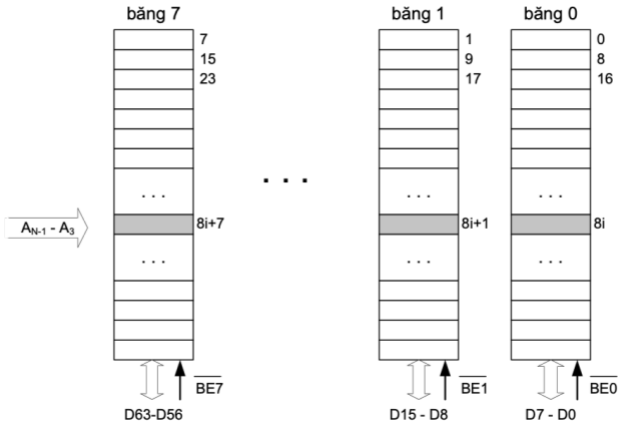
*m = 16 bit => Hai băng nhớ đan xen



m = 32 bit => bốn băng nhớ đan xen



m = 64 bit => tám băng nhớ đan xen



Ví dụ:

Máy tính dùng 32 bit địa chỉ để đánh địa chỉ cho bộ nhớ theo byte; bus dữ liệu để kết nối với bộ nhớ chính là 32 bit. Hãy cho biết

a. Số byte nhớ tối đa được đánh địa chỉ ? Địa chỉ đầu và địa chỉ cuối dưới dạng Hexa ?

b. Hãy cho biết các byte nhớ có địa chỉ sau đây 0x0FE12C3D, 0x10ABCD06 được bố trí ở băng nhớ nào?

Bài làm

a. Vì máy tính dùng 32 bit địa chỉ để đánh địa chỉ cho bộ nhớ theo byte nên số byte nhớ tối đa được đánh địa chỉ là $\mathbf{2^{32}}$ **byte** nhớ

Địa chỉ đầu: 0x0000 0000

Địa chỉ cuối 0xFFFF FFFF

b. Vì bus dữ liệu để kết nối với bộ nhớ chính là 32 bit nên sẽ có 32 / 8 = 4 băng nhớ

2 bit least significant của địa chỉ ô nhớ sẽ quyết định nó nằm ở băng nhớ nào

0x0FE12C3D

chuyển D sang nhị phân => 1101

có 2 bit LS là 01 = 1

=> Byte nhớ có địa chỉ là 0x0FE12C3D nằm ở băng nhớ 1

0x10ABCD06

Chuyển 6 sang nhị phân => 0110

có 2 bit LS là 10 = 2

=> Byte nhớ có địa chỉ là 0x10ABCD06 nằm ở băng nhớ 2

Ví dụ thêm :

Máy tính dùng 64 bit để đánh địa chỉ cho bộ nhớ theo byte ; bus dữ liệu để kết nối với bộ nhớ chính là 16 bit. Cho biết

a. Số byte nhớ tối đa được đánh địa chỉ ? Địa chỉ đầu và địa chỉ cuối dưới dạng Hexa ?

b. Hãy cho biết các byte nhớ có địa chỉ sau đây :

1. 0x1234 5678 F123 F691
2. 0x2607 2002 2611 2002

được bố trí ở băng nhớ nào ?

Máy tính dùng 32 bit để đánh địa chỉ cho bộ nhớ theo byte ; bus dữ liệu để kết nối với bộ nhớ chính là 64 bit. Cho biết

a. Số byte nhớ tối đa được đánh địa chỉ ? Địa chỉ đầu và địa chỉ cuối dưới dạng Hexa ?

b. Hãy cho biết các byte nhớ có địa chỉ sau đây :

1. 0x1234 567A
2. 0x2607 200F

được bố trí ở băng nhớ nào ?

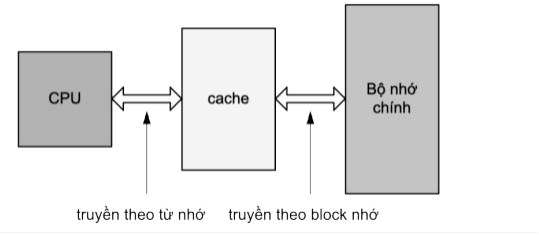
4. Bộ nhớ đệm (Cache)

4.1 Nguyên tắc chung của Cache (tham khảo)

- Cache có tốc độ nhanh hơn bộ nhớ chính

- Cache được đặt giữa CPU và bộ nhớ chính nhằm tăng tốc độ CPU truy cập bộ nhớ

- Cache có thể được đặt trên chip CPU



4.2 Thao tác của Cache (tham khảo)

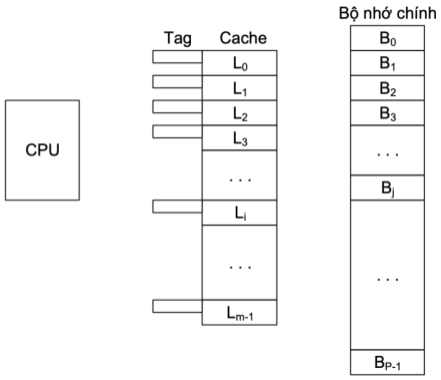
- CPU yêu cầu nội dung của ngăn nhớ

- CPU kiểm tra trên cache với dữ liệu này

- Nếu có CPU nhận dữ liệu từ cache (nhanh), nếu không có đọc Block nhớ chứa dữ liệu từ bộ nhớ chính vào Cache

- TIếp đó chuyển dữ liệu từ cache vào CPU

4.3 Cấu trúc chung của cache / bộ nhớ chính (RAM)



- Bộ nhớ chính có 2^N byte nhớ

- Bộ nhớ chính và cache được chia thành các khối có kích thước bằng nhau

  - Bộ nhớ chính: $B_0, B_1, B_2, \ldots, B_{p-1}$ (p Blocks)

  - Bộ nhớ cache: $L_0, L_1, L_2, \ldots, L_{m-1}$ (m Lines)

  - Kích thước của Block = Kích thước của Line = 8, 16, 32, 64, 128 byte (tùy)

- Mỗi Line trong cache có một thẻ nhớ (Tag) được gắn vào

4.3.1 Các phương pháp ánh xạ

- Ánh xạ trực tiếp (Direct mapping)

- Ánh xạ liên kết toàn phần (Fully associative mapping)

- Ánh xạ liên kết tập hợp (Set associative mapping)

4.3.1.1 Ánh xạ trực tiếp

- Mỗi Block của bộ nhớ chính chỉ có thể được nạp vào một Line của Cache

    - $B_0 \rightarrow L_0$

    - $B_1 \rightarrow L_1$

    -…

    - $B_{m-1} \rightarrow L_{m-1}$

    - $B_m \rightarrow L_0$

    - $B_{m+1} \rightarrow L_1$

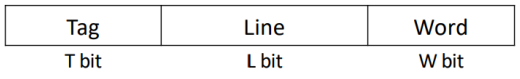
- Tổng quát :

    - B_j chỉ có thể được nạp vào $L_{j \bmod m}$

    - m là số Line của cache

- Phân chia địa chỉ bộ nhớ :

- Địa chỉ N bit của bộ nhớ chính chia thành ba trường:
    - Trường Word gồm W bit xác định một từ nhớ trong Block hay Line:
      2^W = kích thước của Block hay Line
    - Trường Line gồm L bit xác định một trong số các Line trong cache:
      2^L = số Line trong cache = m
    - Trường Tag gồm T bit:
      T = N - (W+L)



Ý nghĩa :

- W là số bit dùng để xác định một byte nhớ trong Block

- L là số bit dùng để xác định Line nào chúng ta đang làm việc với

- T là số bit dùng để xác định Block nào đang được lưu trong Line

Ví dụ :

Cho máy tính với 64Kbytes bộ nhớ chính được đánh địa chỉ theo byte, bộ nhớ cache gồm 32 lines được tổ chức ánh xạ trực tiếp, kích thước mỗi line là 8 bytes.

a. Xác định số bit của các trường địa chỉ: Tag, Line, Word

b. Chỉ ra mỗi byte nhớ của bộ nhớ chính có địa chỉ cho dưới đây được nạp vào line nào của cache:

0001 0001 0001 1011

1100 0011 0011 0100

1101 0000 1101 1101

1010 1010 1010 1010

c. Giả thiết byte nhớ có địa chỉ 0001 1010 0001 1010 được nạp vào cache, hãy chỉ ra địa chỉ theo dạng nhị phân của những byte nhớ khác cùng được nạp với byte nhớ đó trong cùng line.

a. Máy tính có 64Kbytes bộ nhớ = 64000 bytes nên cần ít nhất 16 bit để đánh địa chỉ (vì $2^{16} - 1 = 65535$)

=> N = 16 bit

+ Xác định Line :

Có 32 Line nên cần L = $log_2 32 = 5$ bit để biểu diễn 32 Line

+ Xác định Word :

1 Line = 1 Block = 8 bytes

Nên cần W = $log_2 8$ = 3 bit

+ Xác định Tag:

T = N – (L + W) = 16 – (5 + 3) = 8 bit

b.

0001 0001 0001 1011

Phân chia theo câu a có:

00010001 00011 011

T L W

Chuyển phần Line sang thập phân: 3 => Line 3

Tương tự với các địa chỉ còn lại

c.

0001 1010 0001 1010

Phân chia theo câu a:

00011010 00011 010

=> Dạng nhị phân của những byte nhớ khác cũng được nạp với byte nhớ đó trong cùng Line là:

$$a_{15}a_{14}a_{13}a_{12}a_{11}a_{10}a_9a_8\ 00011\ a_3a_2a_1$$

4.3.1.2 Ánh xạ liên kết toàn phần

- Mỗi Block có thể nạp vào bất kỳ Line nào của cache

- Địa chỉ của bộ nhớ chính được chia thành 2 trường:

    - Trường Word gồm W bit xác định 1 từ nhớ trong Block hay Line:

    2^W = kích thước của Block hay Line

    - Trường Tag gồm T bit dùng để xác định Block của bộ nhớ chính:

    T = N – W

- Mỗi thẻ nhớ (Tag) của một Line chứa được T bit
- Khi Block từ bộ nhớ chính được nạp vào Line của cache thì Tag ở đó được cập nhật giá trị là T bit địa chỉ bên trái của Block đó
- Khi CPU muốn truy nhập một từ nhớ thì nó phát ra một địa chỉ N bit cụ thể
    - So sánh T bit bên trái của địa chỉ vừa phát ra với lần lượt nội dung của các Tag trong cache
        - Nếu gặp giá trị bằng nhau: cache hit xảy ra ở Line đó
        - Nếu không có giá trị nào bằng: cache miss
- Ưu điểm: Xác suất cache hit cao
- Nhược điểm:
    - So sánh đồng thời với tất cả các Tag → mất nhiều thời gian
    - Bộ so sánh phức tạp
- Ít sử dụng

4.3.1.3 Ánh xạ liên kết tập hợp

- Dung hòa cho hai phương pháp trên
- Cache đươc chia thành các Tập (Set)
- Mỗi một Set chứa một số Line
- Ví dụ:
  - 4 Line/Set → 4-way associative mapping
- Ánh xạ theo nguyên tắc sau:
  - $B_0 \rightarrow S_0$
  - $B_1 \rightarrow S_1$
  - $B_2 \rightarrow S_2$
  - .....

- Địa chỉ N bit của bộ nhớ chính chia thành ba trường:
  - Trường Word gồm W bit, xác định một từ nhớ trong Block hay Line:
    2^W = kích thước của Block hay Line
  - Trường Set gồm S bit, xác định một trong số các Set trong cache:
    2^S = số Set trong cache
  - Trường Tag gồm T bit:
    T = N - (S+W)

Tag	Set	Word
T bit	S bit	W bit

- Khi CPU muốn truy nhập một từ nhớ thì nó phát ra một địa chỉ N bit cụ thể
  - Nhờ vào giá trị S bit của trường Set sẽ tìm ra Set tương ứng
  - So sánh T bit bên trái của địa chỉ vừa phát ra với lần lượt nội dung của các Tag trong Set đó
    - Nếu gặp giá trị bằng nhau: cache hit xảy ra ở Line tương ứng
    - Nếu không có giá trị nào bằng: cache miss
- Tổng quát cho cả hai phương pháp trên
- Thông dụng với: 2,4,8,16Lines/Set

Ý nghĩa:

- W xác định Byte trong Block

- S xác định Set trong Cache

- T xác định Block trong Line Set

VII. Hệ thống vào ra

1. Vào ra theo bản đồ bộ nhớ (MIPS dùng)

- Sử dụng lệnh trên hệ thống để thao tác như bình thường

- Vào/ra dữ liệu: sử dụng lệnh load/store

2. Vào ra riêng biệt (MIPS không dùng bỏ)

3. DMA (Direct Memory Access)

- Thay CPU để xử lý hoạt động vào-ra

- Các thành phần:

    - Thanh ghi dữ liệu
    - Thanh ghi địa chỉ
    - Bộ đếm dữ liệu
    - Logic điều khiển

- Phần cứng sử dụng:

BUS hoặc BUS vào – ra

Bài tập:

Với máy tính dùng bộ xử lý theo kiến trúc MIPS, các cổng vào-ra cần phải địa chỉ hóa bằng phương pháp vào-ra theo bản đồ bộ nhớ (Memory mapped IO). Giả sử hệ thống có hai cổng vào-ra 32-bit P1, P2 được gán các địa chỉ tương ứng là 0xFFFF0004 và 0xFFFF0008. Hãy viết đoạn chương trình hợp ngữ MIPS để thực hiện: đọc 100 dữ liệu vào từ cổng P1 rồi ghi lần lượt các dữ liệu đó ra cổng P2.

```
lui $s0, 0xFFFF
ori $s0, $s0, 0x0004
lui $s1, 0xFFFF
ori $s1, $s1, 0x0008

add $t0, $zero, $zero # i = 0
addi $t1, $zero, 100
loop:
        beq $t0, $t1, end_loop

        lw $s2, 0($s0)
        sw $s2, 0($s1)

        addi $t0, $t0, 1
        j loop
end_loop:
```

VII. Các kiến trúc song song

1. Phân loại

Phân loại kiến trúc máy tính (Michael Flynn -1966)

- SISD - Single Instruction Stream, Single Data Stream
- SIMD - Single Instruction Stream, Multiple Data Stream
- MISD - Multiple Instruction Stream, Single Data Stream
- MIMD - Multiple Instruction Stream, Multiple Data Stream

2. Bộ xử lý đồ họa (GPU)

- Kiến trúc SIMD
- Xuất phát từ bộ xử lý đồ họa GPU (Graphic Processing Unit) hỗ trợ xử lý đồ họa 2D và 3D: xử lý dữ liệu song song
- GPGPU – General purpose Graphic Processing Unit
- Hệ thống lai CPU/GPGPU
  - CPU là host: thực hiện theo tuần tự
  - GPGPU: tính toán song song