Chương 6 – Kiến trúc bộ lệnh

6.1. Phân loại kiến trúc bộ lệnh

- kiến trúc ngăn xếp (stack),
- kiến trúc thanh ghi tích lũy (Accumulator)
- kiến trúc thanh ghi đa dụng GPRA(general-purpose register architecture).

Ví dụ phép tính C = A + B được dùng trong các kiểu kiến trúc:

Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R1,B	Load R2,B
Add	Store C	Store C,R1	Add R3,R1,R2
Pop C			Store C,R3

Kiểu kiến trúc GPR

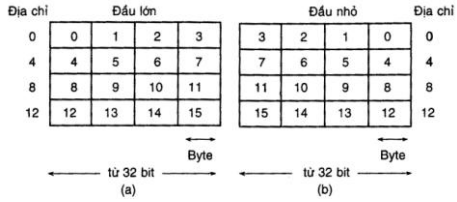
- Ưu điểm
  - Dùng thanh ghi, một dạng lưu trữ trong của CPU có tốc độ nhanh hơn bộ nhớ ngoài
  - Trình tự thực hiện lệnh có thể ở mọi thứ tự
  - Dùng thanh ghi để lưu các biến và như vậy sẽ giảm thâm nhập đến bộ nhớ => chương trình sẽ nhanh hơn
- Nhược điểm
  - Lệnh dài
  - Số lượng thanh ghi bị giới hạn
- Ngăn xếp (Stack) ?
- Thanh ghi tích luỹ (Accumulator Register) ?

Kiểu kiến trúc thanh ghi đa dụng

- lệnh có 2 toán hạng
  ADD A, B
- lệnh có 3 toán hạng
  ADD A, B, C
- Số toán hạng bộ nhớ có thể thay đổi từ 0 tới 3
- Các loại toán hạng
  - thanh ghi-thanh ghi (kiểu này còn được gọi nạp - lưu trữ),
  - thanh ghi - bộ nhớ
  - bộ nhớ - bộ nhớ.

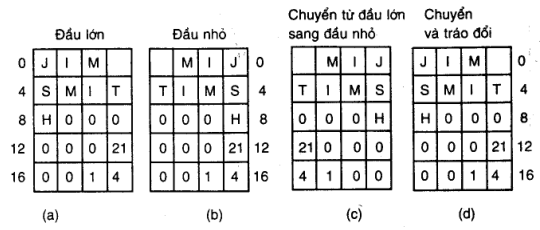
6.2. Địa chỉ bộ nhớ

- Các khái niệm:
  - Memory, bit, cell, address, byte, word
- **Sắp xếp thứ tự byte**
  - **Có vấn đề gì không trong cách sắp xếp thứ tự byte**



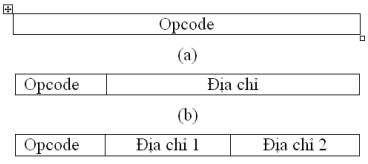
Vấn đề thứ tự byte

VD: Biểu diễn JIM SMITH, 21 tuổi, phòng 260



6.3. Mã hóa tập lệnh

- Các trường mã hóa:
  - ***mã tác vụ* (*operation code*):** Opcode
  - Địa chỉ



Các tiêu chuẩn thiết kế dạng thức lệnh

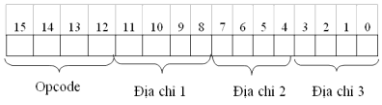
- ***Có 4 tiêu chuẩn thiết kế:***
  - ***Mã lệnh ngắn ưu việt hơn mã lệnh dài***
  - ***Độ dài mã lệnh đủ để biểu diễn tất cả phép toán mong muốn***
  - ***độ dài word của máy bằng bội số nguyên của độ dài ký tự***
  - ***số BIT trong trường địa chỉ càng ngắn càng tốt***

Ví dụ thiết kế máy với ký tự 8 bit và bộ nhớ chính chứa 2^{16} ký tự

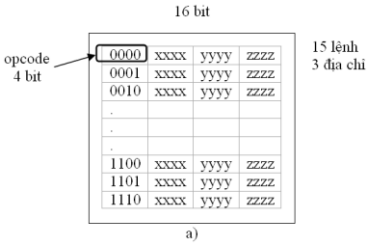
+ Ô nhớ kích thước 8 bit => trường địa chỉ cần 16 bit
+ Ô nhớ kích thước 32 bit => trường địa chỉ cần 14 bit

Opcode mở rộng

ví dụ một máy tính có lệnh dài 16 bit :

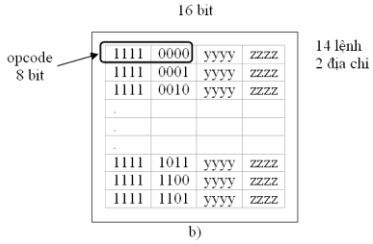


- Lệnh (n+k) bit với opcode chiếm k bit và địa chỉ chiếm n bit.
VD: 15 lệnh ba địa chỉ



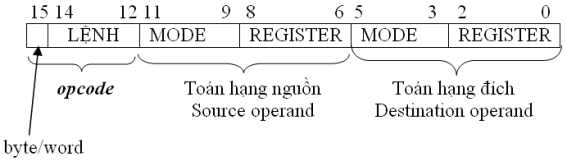
Opcode mở rộng

- 14 lệnh hai địa chỉ



dạng thức lệnh PDP-11

- Mã hóa lệnh trên máy PDP-11



- tám cách trên PDP-11
- opcode mở rộng có dạng x111
- các lệnh một toán hạng
  - opcode 10 bit: 4 bit opcode và 6 bit của trường toán hạng nguồn
  - mode/register 6 bit

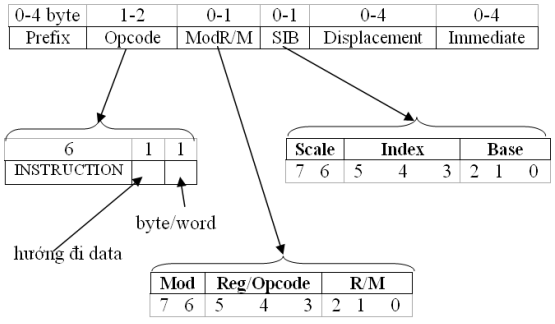
Họ Intel 8088/80286/80386/Pentium

- Dạng thức lệnh của các máy tính Intel:
  - Cấu tạo phức tạp
  - kế thừa từ nhiều thế hệ
  - bốn cách lập địa chỉ toán hạng (so với tám cách trên PDP-11)

CPU	PREFIX	OPCODE	MODE	SIB	DISPLACEMENT	IMMEDIATE
8088	0-3	1	0-1	0	0-2	0-2
80286	0-3	1	0-1	0	0-2	0-2
80386	0-4	1-2	0-1	0-1	0-4	0-4
Pentium	0-4	1-2	0-1	0-1	0-4	0-4

PREFIX byte:
- LOCK prefix: để đảm bảo việc dành riêng vùng nhớ chia sẻ trong môi trường đa bộ xử lý
- REPeat prefix: đặc trưng cho một chuỗi phép toán được lập đi lập lại

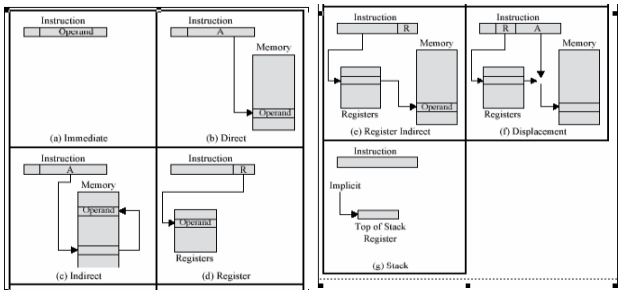
Format lệnh Pentium



Các chế độ lập địa chỉ

- Địa chỉ tức thời – Immediate
- Địa chỉ trực tiếp – Direct
- Địa chỉ gián tiếp – Indirect
- Địa chỉ thanh ghi – Register
- Địa chỉ gián tiếp thanh ghi – Register inderect
- Địa chỉ dịch chuyển – Displacement
- Địa chỉ ngăn xếp - Stack

Các chế độ lập địa chỉ



Cách tính địa chỉ thực

Chế độ	Cách tính	Ưu điểm	Khuyết điểm
Immediate	Operand = A	Không có tham chiếu bộ nhớ	Độ lớn toán hạng giới hạn
Direct	EA = A	Đơn giản	không gian địa chỉ giới hạn
Indirect	EA = (A)	không gian địa chỉ lớn	Tham chiếu bộ nhớ phức tạp
Register	EA = R	Không có tham chiếu bộ nhớ	không gian địa chỉ giới hạn
Register indirect	EA = (R)	không gian địa chỉ lớn	Tham chiếu bộ nhớ phụ
Displacement	EA = A + (R)	Linh động	Phức tạp
Stack	EA= đầu của ngăn xếp	Không có tham chiếu bộ nhớ	Ứng dụng giới hạn

Các chế độ lập địa chỉ

- ***Lập địa chỉ tức thời (Immediate Addressing)***:
  - OPERAND = A
  - MOV R1, #4
- ***Lập địa chỉ trực tiếp (Direct Addressing)***:
  - EA = A
- ***Lập địa chỉ gián tiếp (Indirect Addressing)***
  - EA = (A)
  - một con trỏ (trong C++)
- ***Lập địa chỉ thanh ghi (Register Addressing)***
  - trỏ tới một thanh ghi
  - Các máy ngày nay được thiết kế có các thanh ghi vì lý do?

Các chế độ lập địa chỉ

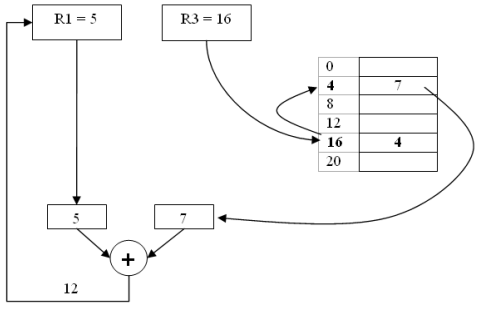
- ***Địa chỉ gián tiếp thanh ghi (Register Indirect)***
  - EA = (R)
- ***Địa chỉ Địa chỉ dịch chuyển – Displacement***
  - EA = A + (R)
- ***Địa chỉ ngăn xếp – Stack***
  - ***FILO (first in last out)***

VD:

Addressing mode	Example instruction	Meaning	When used
Register	Add R4,R3	Regs[R4] ← Regs[R4] + Regs[R3]	When a value is in a register.
Immediate	Add R4,#3	Regs[R4] ← Regs[R4] + 3	For constants.
Displacement	Add R4,100(R1)	Regs[R4] ← Regs[R4] + Mem[100+Regs[R1]]	Accessing local variables.
Register deferred or indirect	Add R4,(R1)	Regs[R4] ← Regs[R4] + Mem[Regs[R1]]	Accessing using a pointer or a computed address.
Indexed	Add R3,(R1 + R2)	Regs[R3] ← Regs[R3] + Mem[Regs[R1] + Regs[R2]]	Sometimes useful in array addressing: R1 = base of array; R2 = index amount.
Direct or absolute	Add R1,(1001)	Regs[R1] ← Regs[R1] + Mem[1001]	Sometimes useful for accessing static data; address constant may need to be large.
Memory indirect or memory deferred	Add R1,@(R3)	Regs[R1] ← Regs[R1] + Mem[Mem[Regs[R3]]]	If R3 is the address of a pointer p, then mode yields $*p$.
Autoincrement	Add R1,(R2)+	Regs[R1] ← Regs[R1] + Mem[Regs[R2]] Regs[R2] ← Regs[R2] + d	Useful for stepping through arrays within a loop. R2 points to start of array; each reference increments R2 by size of an element, d.
Autodecrement	Add R1,-(R2)	Regs[R2] ← Regs[R2] – d Regs[R1] ← Regs[R1] + Mem[Regs[R2]]	Same use as autoincrement. Autodecrement/increment can also act as push/pop to implement a stack.
Scaled	Add R1,100(R2)[R3]	Regs[R1] ← Regs[R1] + Mem[100+Regs[R2]+Regs[R3]*d]	Used to index arrays. May be applied to any indexed addressing mode in some machines.

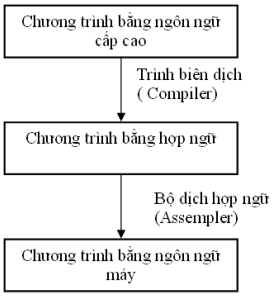
Ví dụ lệnh Add với tham chiếu bộ nhớ

- **Add R1, @(R3)**



6.4. Bộ lệnh

- Quá trình biên dịch ra ngôn ngữ máy



Nhóm lệnh truyền dữ liệu

- **MOVE Ri, Rj**
- **Một số ví dụ lệnh MOVE:**

Đích	Nguồn	Ví dụ	Giải thích
Bộ nhớ	Thanh ghi	MOVE 100H, AX	Chuyển nội dung trong AX vao vị trí nhớ 100H
Thanh ghi	Bộ nhớ	MOVE AX,MEM1	Chuyển nội dung trong vị trí nhớ MEM1 chỉ ra vào thanh ghi AX
Thanh ghi	Thanh ghi	MOVE AX, BX	Chuyển nội dung trong thanh ghi BX vào thanh ghi AX
Thanh ghi	Hằng số	MOVE AX, 0FFFFH	Chuyển giá trị hằng số ở hệ 16: FFFF vào thanh ghi AX, số 0 ở đầu để chỉ rõ FFFFH là một giá trị hằng chứ không phải là một nhãn

Nhóm lệnh truyền dữ liệu

- LOAD ***đích, nguồn***
  - ví dụ: LOAD Ri, M (địa chỉ) // Ri←M[địa chỉ]
- STORE ***đích, nguồn***
  - ví dụ: STORE M(địa chỉ), Ri // M[địa chỉ] ←Ri

Data movement operation	Meaning
MOVE	Move data (a word or a block) from a given source (a register or a memory) to a given destination
LOAD	Load data from memory to a register
STORE	Store data into memory from a register
PUSH	Store data from a register to stack
POP	Retrieve data from stack into a register

Nhóm lệnh tính toán số học

- ADD đích, nguồn // đích ← đích + nguồn
- SUB đích, nguồn // đích ← đích – nguồn
- Ví dụ:

```
ADD   AX, BX      // AX ←AX + BX
ADD   AL,74H      // AL ← AL + [74H]
SUB   CL, AL      // CL ← CL – AL
SUB   AX, 0405H   // AX ← AX – 0405H
```

Nhóm lệnh tính toán số học

- Các lệnh tính toán số học cơ bản

Tên lệnh	Ý nghĩa
ADD	Cộng
ADDD	Cộng số có dấu chấm động, chính xác kép
SUB	Trừ
SUBD	Trừ số có dấu chấm động, chính xác kép
MUL	Nhân
DIV	Chia
INC	Tăng lên 1
DEC	Giảm đi 1
NEG	Đảo dấu toán hạng

Nhóm lệnh logic

- AND đích, nguồn
- OR đích, nguồn
- Ví dụ:

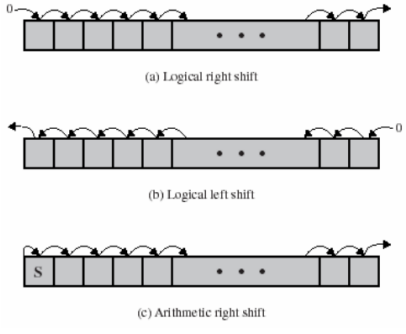
AND AL, BL

AL = 00001101B
BL = 00110011B
=> AL = ***00000001B***

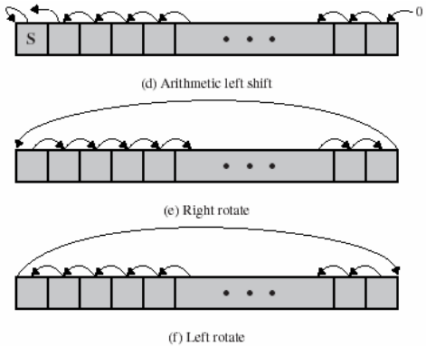
Nhóm các lệnh dịch chuyển số học hoặc logic (SHIFT)

- SRL (Shift Right Logical - dịch phải logic)
- SLL (Shift Left Logical - dịch trái logic)
- SRA (Shift Right Arithmetic - dịch phải số học)
- SLA (Shift Left Arithmetic – dịch trái số học)

Các lệnh dịch chuyển

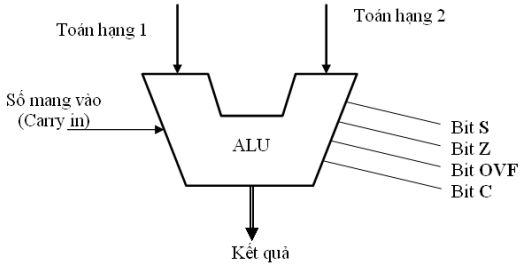


Các lệnh dịch chuyển



Các lệnh có điều kiện và lệnh nhảy

Nếu <điều kiện> thì <chuỗi lệnh 1> nếu không <chuỗi lệnh 2>
(IF <condition> THEN <instructions1> ELSE <instructions2>)



Các lệnh có điều kiện và lệnh nhảy

Ví dụ:

```
        LOAD        R1, #100
Loop:   ADD         R0, (R2)+
        DECREMENT   R1
        BEQZ        R1, Loop
```


Thống kê sử dụng CPU

Loại lệnh	% sử dụng thời gian
Chuyển dữ liệu	43%
Điều khiển dòng chảy	23%
Tính toán số học	15%
So sánh	13%
Phép toán Logic	5%
Các lệnh khác	1%

Cấu trúc lệnh CISC và RISC

RISC	CISC
– Độ dài lệnh cố định (32 bit) – Sử dụng kiến trúc load-store các lệnh xử lý dữ liệu hoạt động chỉ trong thanh ghi và cách ly với các lệnh truy cập bộ nhớ – Một số lớn các thanh ghi đa dụng 32 bit – Có một số ít lệnh (thường dưới 100 lệnh) – Có một số ít các kiểu định vị – Có một số ít dạng lệnh (một hoặc hai) – Chỉ có các lệnh ghi hoặc đọc ô nhớ mới thâm nhập vào bộ nhớ.	– Kích thước tập lệnh thay đổi – Giá trị trong bộ nhớ được dùng như như toán hạng trong các chỉ lệnh xử lý dữ liệu – Có rất nhiều thanh ghi, nhưng hầu hết chỉ để sử dụng cho một mục đích riêng biệt nào đấy – Có rất nhiều lệnh (khoảng 500) – Có nhiều kiểu định vị (xem phần 6.3.4) – Có nhiều dạng lệnh – Có nhiều lệnh khác cũng thâm nhập vào bộ nhớ được
– Giải mã lệnh logic bằng kết nối phần cứng – Thực thi chỉ lệnh theo cấu trúc dòng chảy (xem hình 7.9 trong chương sau) – Một lệnh thực thi trong 1 chu kì xung nhịp	– Sử dụng rất nhiều code trong ROM giải mã các chỉ lệnh – Các máy cũ phải tuần tự hết dòng lệnh này mới đến dòng lệnh khác – Cần nhiều chu kì xung nhịp để hoàn thành một lệnh

CÂU HỎI VÀ BÀI TẬP

1. Giả sử cần thiết kế máy với ký tự 8 bit và bộ nhớ chính chứa 2^{24} ký tự. Hãy cho biết trường địa chỉ cần bao nhiêu bit trong trường hợp:
   a) Ô nhớ kích thước 8 bit
   b) Ô nhớ kích thước 16 bit
   c) Ô nhớ kích thước 32 bit
2. Thiết kế opcode mở rộng nhằm cho phép mã hóa nội dung sau trong lệnh 36 bit
   – 7 lệnh có hai địa chỉ 15 bit và một số hiệu thanh ghi 3 bit
   – 500 lệnh có một địa chỉ 15 bit và một số hiệu thanh ghi 3 bit
   – 50 lệnh không có địa chỉ hoặc thanh ghi
3. Có thể thiết kế opcode mở rộng để cho phép mã hóa nội dung sau trong lệnh 12 bit được không? Trường thanh ghi rộng 3 bit.
   – 4 lệnh có ba thanh ghi
   – 255 lệnh có hai thanh ghi
   – 2048 lệnh không có thanh ghi