# CHƯƠNG 3 - CẤP MÁY VI CHƯƠNG TRÌNH

Trong chương này chúng ta sẽ nghiên cứu cách thức vi chương trình điều khiển các thành phần phần cứng (máy cấp 1) và phiên dịch (interpreter) các chỉ thị của cấp máy qui ước (máy cấp 2).

Chúng ta sẽ bắt đầu khảo sát cấp vi chương trình bằng việc nhắc lại một cách khái quát các khối cơ bản trong mạch số, chúng được xem như là một bộ phận trong cấu trúc của cấp vi chương trình.

Tiếp theo chúng ta nghiên cứu cách thức các chỉ thị phức tạp được thiết lập từ chuỗi các chỉ thị thô sơ. Chủ đề này sẽ được thảo luận kỹ và được minh hoạ bằng một ví dụ.

Sau đó chúng ta tìm hiểu các yếu tố cần phải khảo sát khi thiết kế cấp vi chương trình cho một máy tính và các phương pháp cải tiến hiệu suất một máy tính.

Phần cuối của chương chúng ta sẽ xem xét một ví dụ về cấp vi chương trình của họ chip vi xử lí Intel.

## 1. Một số thành phần cơ bản trong mạch số

Các vi chương trình điều khiển các thanh ghi, các bus, các ALU, các bộ nhớ, và các thành phần phần cứng khác của máy tính. Việc khảo sát chi tiết các thành phần trên thuộc phạm vi của các môn học như là vi xử lí, kiến trúc máy tính,... Ở đây chúng ta chỉ nhắc lại một cách khái quát phục vụ cho công việc khảo sát cấp vi chương trình của máy tính.

### 1.1. Thanh ghi

Cấp vi chương trình luôn sử dụng một số thanh ghi để lưu trữ thông tin cần thiết cho việc xử lí chỉ thị đang được phiên dịch. Các thanh ghi được đặt ở bên trong CPU.

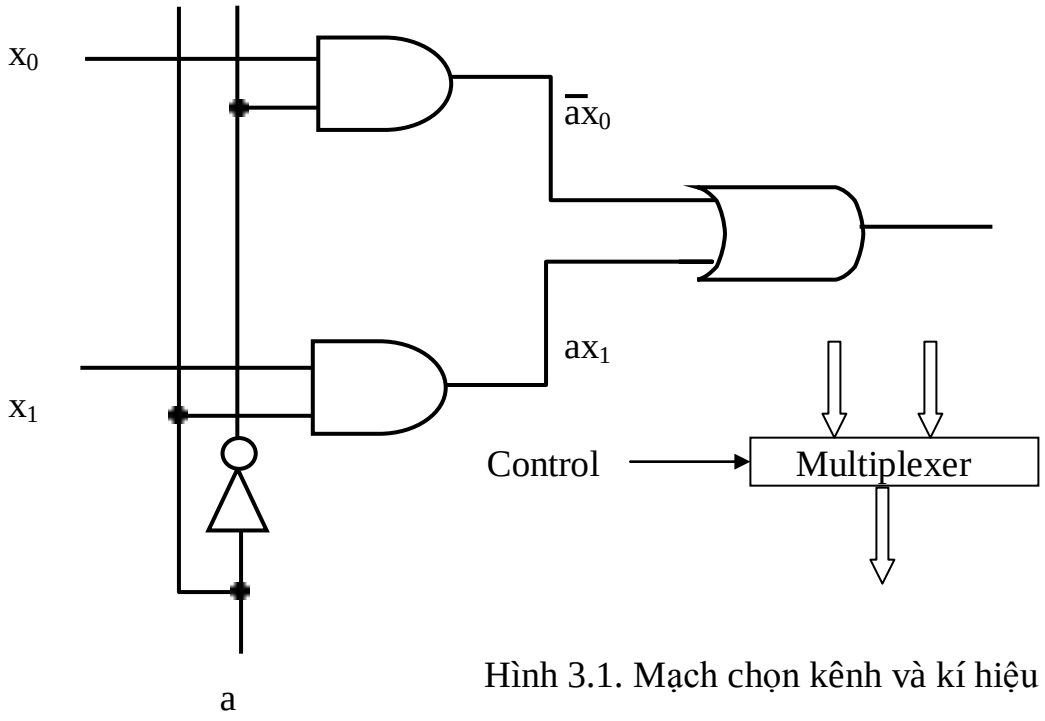
Ở cấp vi chương trình các thanh ghi còn được gọi là bộ nhớ cục bộ (local) hay bộ nhớ nháp (scratch). Thông tin được chứa trong thanh ghi sẽ tồn tại cho tới khi có một thông tin khác thay thế. Khi đọc thông tin ra khỏi thanh ghi, thông tin trong thanh ghi vẫn còn nguyên vẹn.

### 1.2. Bus

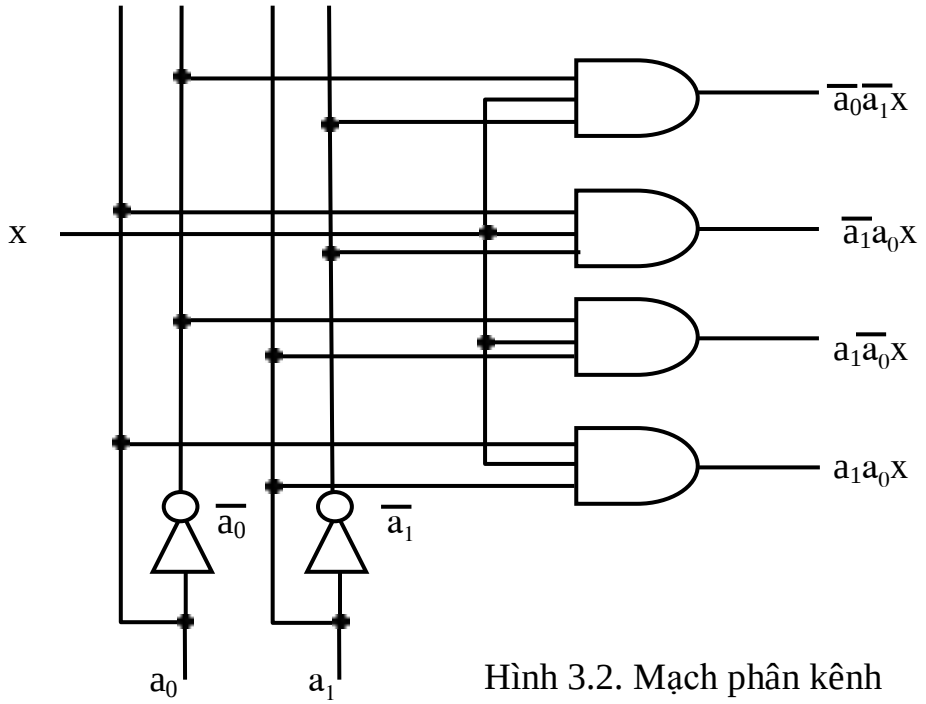
Các bus được để cập ở đây là các đường truyền dữ liệu từ một thành phần này sang một thành phần khác (bus kết nối 2 thành phần). Một bus có thể đơn hướng, có thể đa hướng. Một đường của bus có thể có trạng thái 0, 1, hoặc thả nổi, các bus có tính chất này gọi là bus 3 trạng thái. Bus 3 trang thái được sử dụng khi có nhiều thành phần nối vào bus và đều có khả năng truyền thông tin trên bus.

### 1.3. Mạch chọn kênh / phân kênh và mạch giải mã

### a. Mạch chọn kênh / phân kênh




Hình 3.1. Mạch chọn kênh và kí hiệu

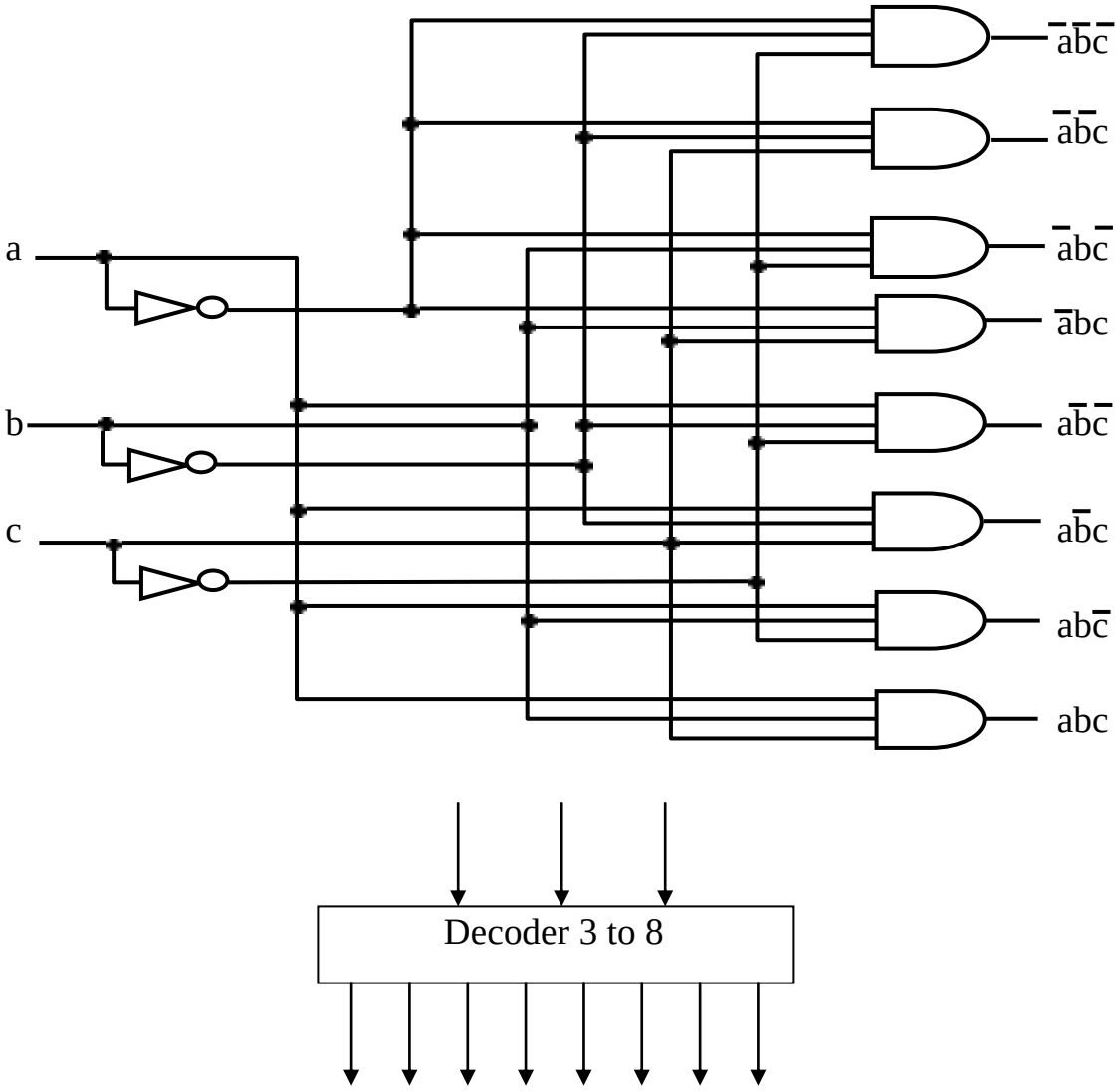



Hình 3.2. Mạch phân kênh

+ Mạch chọn kênh có $2^n$ đường vào dữ liệu, n đường điều khiển và một đường ra dữ liệu.

+ Mạch phân kênh có n đường điều khiển, một đường vào dữ liệu và có thể xuất ra một trong $2^n$ đường ra.

**b. Mạch giải mã**

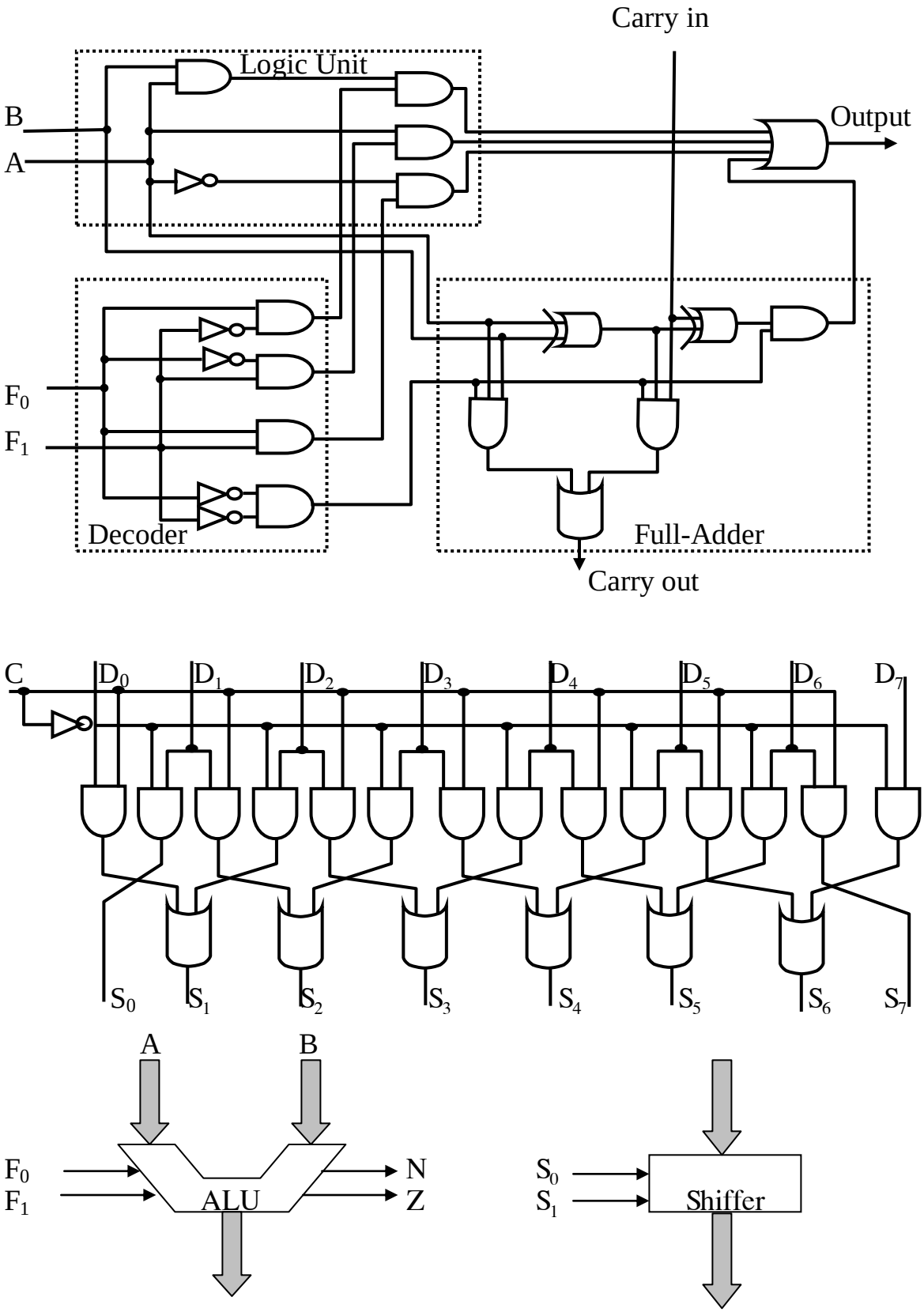



Hình 3.3. Mạch giải mã và ký hiệu

Mạch giải mã có n đường vào và $2^n$ đường ra. Mạch giải mã luôn có một đường ra giá trị 1, còn lại là giá trị 0.

## 1.4. ALU và mạch dịch bit

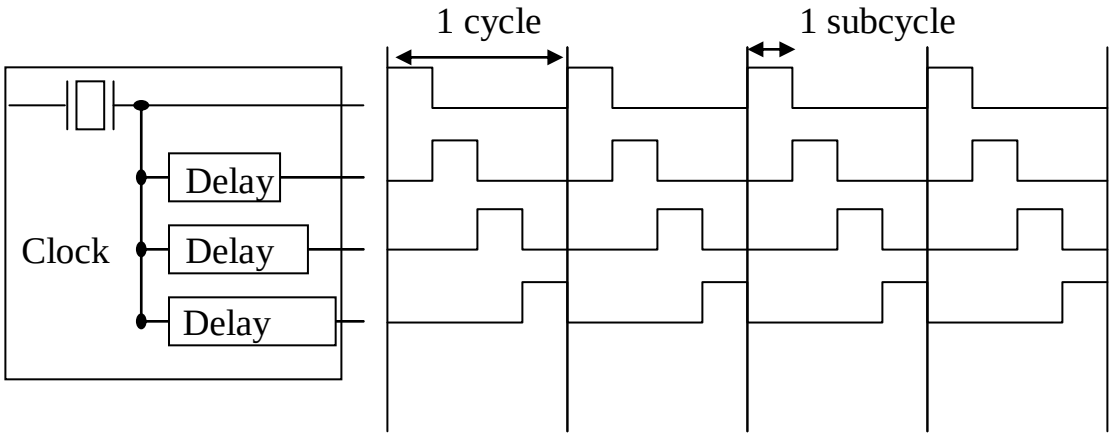
+ ALU (Arithmetic Logical Unit) có 2 bus dữ liệu vào, 1 bus dữ liệu ra. 2 tín hiệu chức năng $F_0$, $F_1$ dùng để xác định chức năng ALU phải thực hiện. Ví dụ trong hình 3.4, ALU thực hiện các chức năng: A and B, not A , A + B và cho A đi qua. Ngoài ra, ALU còn có thể có các tín hiệu ra điều khiển N, Z.




Hình 3.4. ALU - mạch dịch bit và ký hiệu

+ Mạch dịch bit cho phép dịch các bit sang trái hoặc sang phải một bit.

## 1.5. Mạch tạo xung đồng hồ



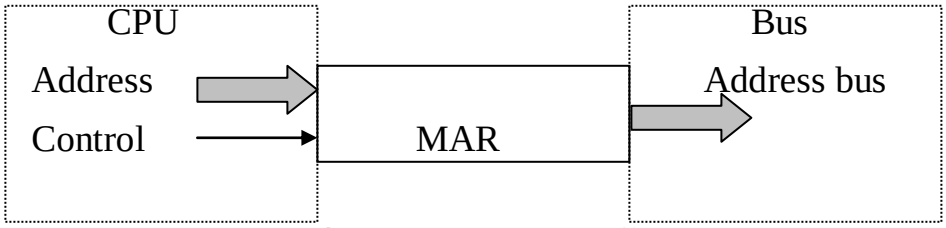

Hình 3.5. Mạch tạo xung đồng hồ và giản đồ thời gian

Các mạch của máy tính thường được điều khiển bởi một mạch tạo xung đồng hồ. Xung đồ hồ xác định chu kỳ của máy tính. Trong mỗi chu kì máy, một hoạt động nào đó xẩy ra (thực thi một vi lệnh chẳng hạn). Một chu kì máy lại thường được chia thành các chu kì con, sao cho các công đoạn khác nhau của một hoạt động được thực thi theo một trật tự xác định.

Hình 3.5. trình bày một mạch tạo xung đồng hồ có 4 đường ra. Tín hiệu trên cùng là tín hiệu chính, 3 tính hiệu còn lại được tạo ra bằng cách làm trễ tín hiệu chính với 3 khoảng thời gian khác nhau. Xung đồng hồ chính có độ rộng xung bằng 1/4 chu kì, các xung còn lại được làm trễ các khoảng thời gian là 1, 2, 3 độ rộng xung so với xung chính.

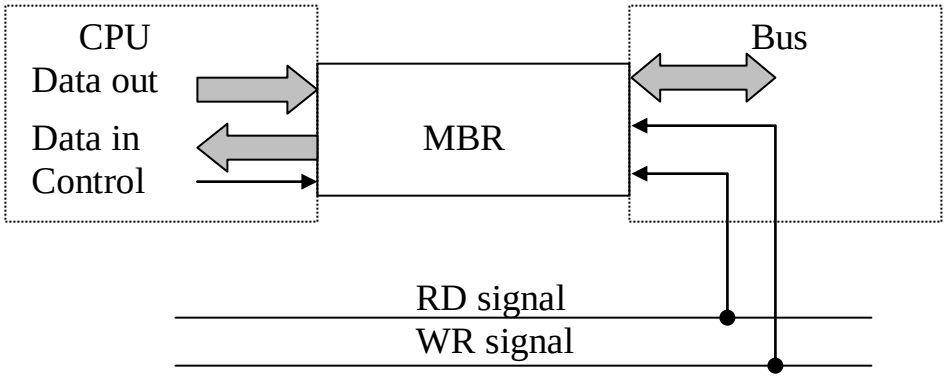
## 1.6. Bộ nhớ chính

Khi máy tính làm việc, CPU thường lấy dữ liệu từ bộ nhớ (bộ nhớ chính - RAM) và ghi dữ liệu lên chính bộ nhớ đó. Hoạt động đọc / ghi bộ nhớ đã được trình bày trong chương 2. Thời gian đọc / ghi bộ nhớ luôn dài hơn thời gian thực hiện một vi lệnh, vì vậy vi chương trình phải lưu các giá trị trên bus địa chỉ và bus dữ liệu trong vài vi lệnh. Để đơn giản hoá công việc này ngươì ta sử dụng 2 thanh ghi là MAR (Memory Address Register) và MBR (Memory Buffer Register). 2 thanh ghi này được đặt giữa CPU và bus hệ thống.




Hình 3.6. Thanh ghi địa chỉ bộ nhớ

Địa chỉ được nạp từ CPU vào MAR khi đường điều khiển ở mức tích cực, và từ MAR đi ra các đương địa chỉ bus hệ thống.




Hình 3.7. Thanh ghi đệm bộ nhớ

Khi có tín hiệu điều khiển phù hợp, dữ liệu từ MBR đi vào CPU theo data in và từ CPU ra MBR theo data out. Bus dữ liệu hệ thống là 2 chiều, khi WR tích cực, dữ liệu đi từ MBR vào bus hệ thống, khi RD tích cực, dữ liệu đi bus hệ thống vào MBR.

## 2. Một vi cấu trúc (Micro-Architecture) mẫu

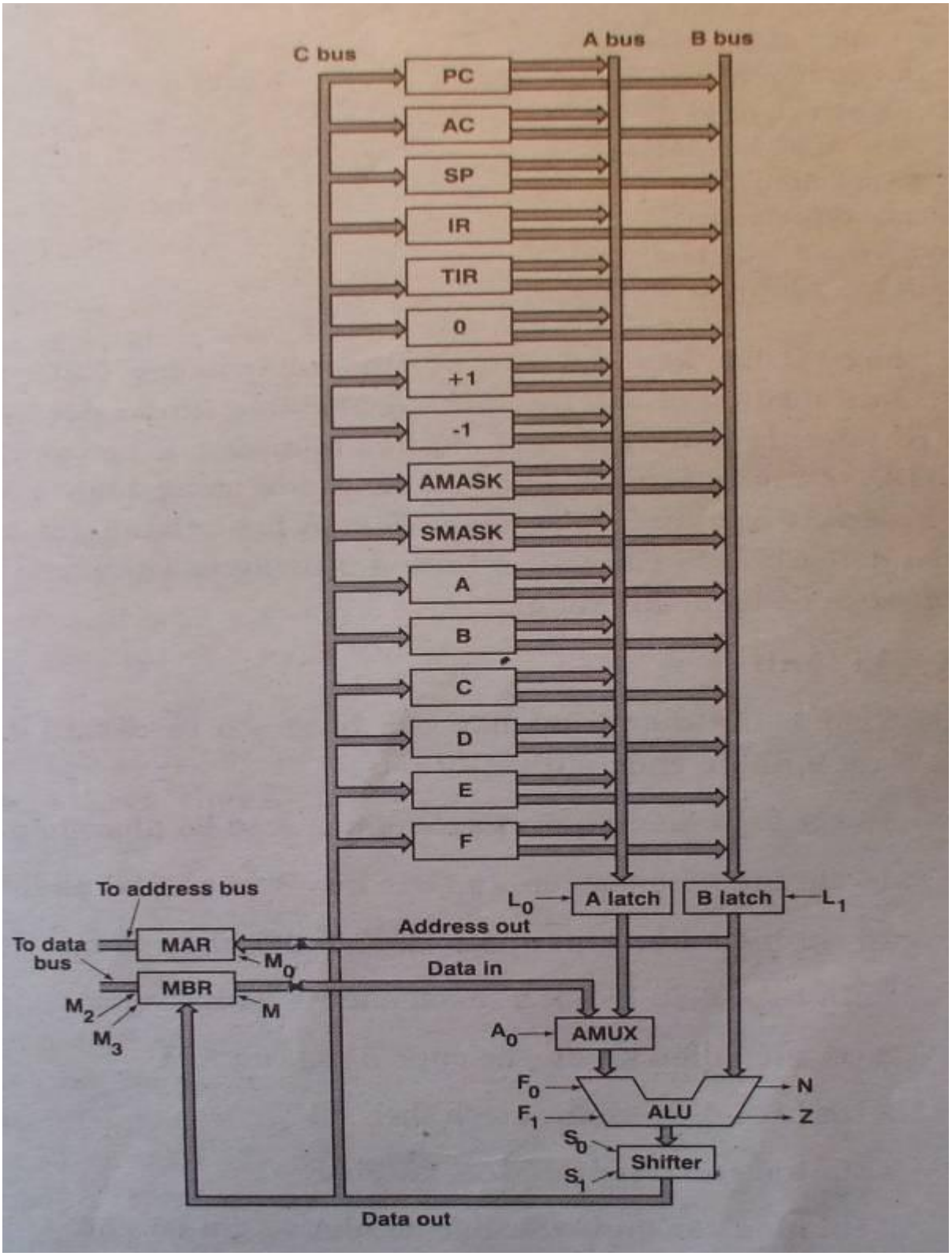
### 2.1. Đường dữ liệu

Đường dữ liệu của vi cấu trúc mẫu được trình bày trong hình 3.8:

+ 16 thanh ghi 16 bit được kí hiệu PC, AC,... tạo thành bộ nhớ đệm chỉ được truy xuất bởi cấp vi chương trình. Mỗi thanh ghi có thể xuất nội dung lên 2 bus A, B và nạp nội dung từ bus C.

+ Các bus A, B 16 bit dẫn dữ liệu đến ALU 16 bit. ALU này thực hiện được các chức năng: A or B, A and B, not B, và A + B. Chức năng cần thực hiện được xác định nhờ 2 đường điều khiển $F_0$, $F_1$. N (Negative) tích cực khi kết quả của ALU âm, Z (Zero) tích cực khi kết quả của ALU bằng 0. Kết quả của ALU được đưa tới mạch dịch bit.

+ 2 bus A, B được nối với ALU thông qua 2 mạch chốt. Vì ALU là mạch tổ hợp, trạng thái ngõ ra tuỳ thuộc tức khắc vào trạng thái ngõ vào và các tín hiệu điều khiển. Điều này nẩy sinh vấn đề khi tính toán, chẳng hạn khi tính A := A + B, thanh ghi A chứa kết quả nên giá trị trên bus A thay đổi kéo theo ngõ ra của ALU thay đổi, kéo theo giá trị trên bus C thay đổi, dẫn đến kết quả sai được lưu vào thanh ghi A. Việc thêm vào các mạch chốt có chức năng chốt các giá trị của A, B trước khi thực hiện phép toán (việc nạp cho mạch chốt được điều khiển bởi $L_0$, $L_1$). Khi đó ALU bị cách ly với các giá trị mới trên bus.




Hình 3.8. Đường dữ liệu của vi cấu trúc mẫu

+ Việc truyền thông với bộ nhớ, thông qua 2 thanh ghi MAR và MBR:

- Thanh ghi MAR được nạp địa chỉ từ mạch chốt B nhờ tín hiệu điều khiển $M_0$ (lúc này, nội dung của mạch chốt B là địa chỉ chứ không phải dữ liệu).

- Khi ghi kết quả lên bộ nhớ, nội dung của mạch dịch bit được nạp cho thanh ghi MBR. $M_1$ điều khiển việc nạp dữ liệu cho MBR, $M_2$ và $M_3$ điều khiển việc đọc / ghi bộ nhớ. Khi đọc bộ nhớ, dữ liệu vào qua mạch chọn kênh Amux. Tín hiệu điều khiển $A_0$ xác định dữ liệu vào ALU từ MBR hoặc từ thanh ghi chốt A.

## 2.2. Vi lệnh

Để điều khiển đường dữ liệu mô tả trong hình 3.8 cần tới 59 tín hiệu:

+ 16 tín hiệu điều khiển nạp dữ liệu cho bus A từ bộ nhớ đệm.

+ 16 tín hiệu điều khiển nạp dữ liệu cho bus B từ bộ nhớ đệm.

+ 16 tín hiệu điều khiển nạp dữ liệu cho bộ nhớ đệm từ bus C.

+ 2 tín hiệu điều khiển 2 mạch chốt.

+ 2 tín hiệu điều khiển các chức năng của ALU.

+ 2 tín hiệu điều khiển mạch dịch bit.

+ 4 tín hiệu điều khiển MAR, MBR và đọc / ghi bộ nhớ.

+ 1 tín hiệu điều khiển Amux.

Với 59 tín hiệu trên, chúng có thể thực hiện một chu kì của đường dữ liệu (đưa giá trị lên bus A và B, chốt dữ liệu vào 2 mạch chốt, thực hiện tính toán trong ALU, dịch bit và cuối cùng cất kết quả vào bộ nhớ đệm hay đồng vào thời bộ nhớ đệm và MBR). Như vậy, chúng ta cần một thanh ghi 59 bit (mỗi bit tương ứng với 1 tín hiệu điều khiển). Bit giá trị 1 tương ứng với tín hiệu tích cực và ngược lại.

Tuy nhiên để giảm giá thành cho mạch, người ta tìm cách giảm tới mức có thể các đường điều khiển:

- 16 tín hiệu điều khiển 16 thanh ghi đưa dữ liệu lên bus A, nhưng tại mỗi thời điểm chỉ có 1 thanh ghi được xuất dữ liệu. Do vậy, có thể thay 16 tín hiệu này bằng một mạch giải mã 4 to 16 để giảm đi 12 tín hiệu. Làm tương tự như vậy với 16 tín hiệu điều khiển đưa dữ liệu lên bus B và nạp dữ liệu từ bus C. Số tín hiệu điều khiển giảm được là 36.

- 2 tín hiệu $L_0$, $L_1$luôn tuân theo một chu kì thời gian nhất định nên chúng được cung cấp bởi mạch tạo xung đồng hồ.

Như vậy số đường điều còn lại là 21 tương ứng với 21 bit. Cần thêm vào 1 bit (ENC - ENable C). Bit nay cho phép dữ liệu từ bus C được lưu vào một trong các thanh ghi ( ENC = 1) và không cho phép (ENC = 0).

| 1 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 4 | 4 | 4 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AMUX | COND | ALU | SH | MBR | MAR | RD | WR | ENC | C | B | A | ADDR |

Hình 3.9. Dạng vi lệnh điều khiển đường dữ liệu

Trong hình 3.9. có 2 trường được thêm vào là COND và ADDR, chúng sẽ được mô tả ở phần sau của giáo trình. Ý nghĩa của 11 trường còn lại được mô tả như dưới đây:

+ AMUX (A MUltipleXer): trường điều khiển chọn kênh dữ liệu đưa vào ALU (0 = dữ liệu từ chốt A, 1 = dữ liệu từ MBR).

+ ALU: trường điều khiển ALU (0 = A and B, 1 = not B, 2 = A or B, 3 = A + B).

+ SH: trường điều khiển dịch bit (0 = không dịch, 1 = dịch phải 1 bit, 2 = dịch trái 1 bit).

+ Các trường MBR, MAR, RD, WR, ENC: các trường điều khiển thanh ghi MBR, MAR, đọc bộ nhớ, ghi bộ nhớ, cất kết quả vào bộ nhớ đệm (0 = không điều khiển, 1 = điều khiển).

+ C: trường điều khiển chọn thanh ghi để cất kết quả của ALU nêu ENC = 1.

+ B: trường điều khiển chọn thanh ghi để đưa dữ liệu hoặc địa chỉ lên bus B.

+ A: trường điều khiển chọn thanh ghi để đưa dữ liệu hoặc địa chỉ lên bus A.

**2.3. Định thì vi lệnh**

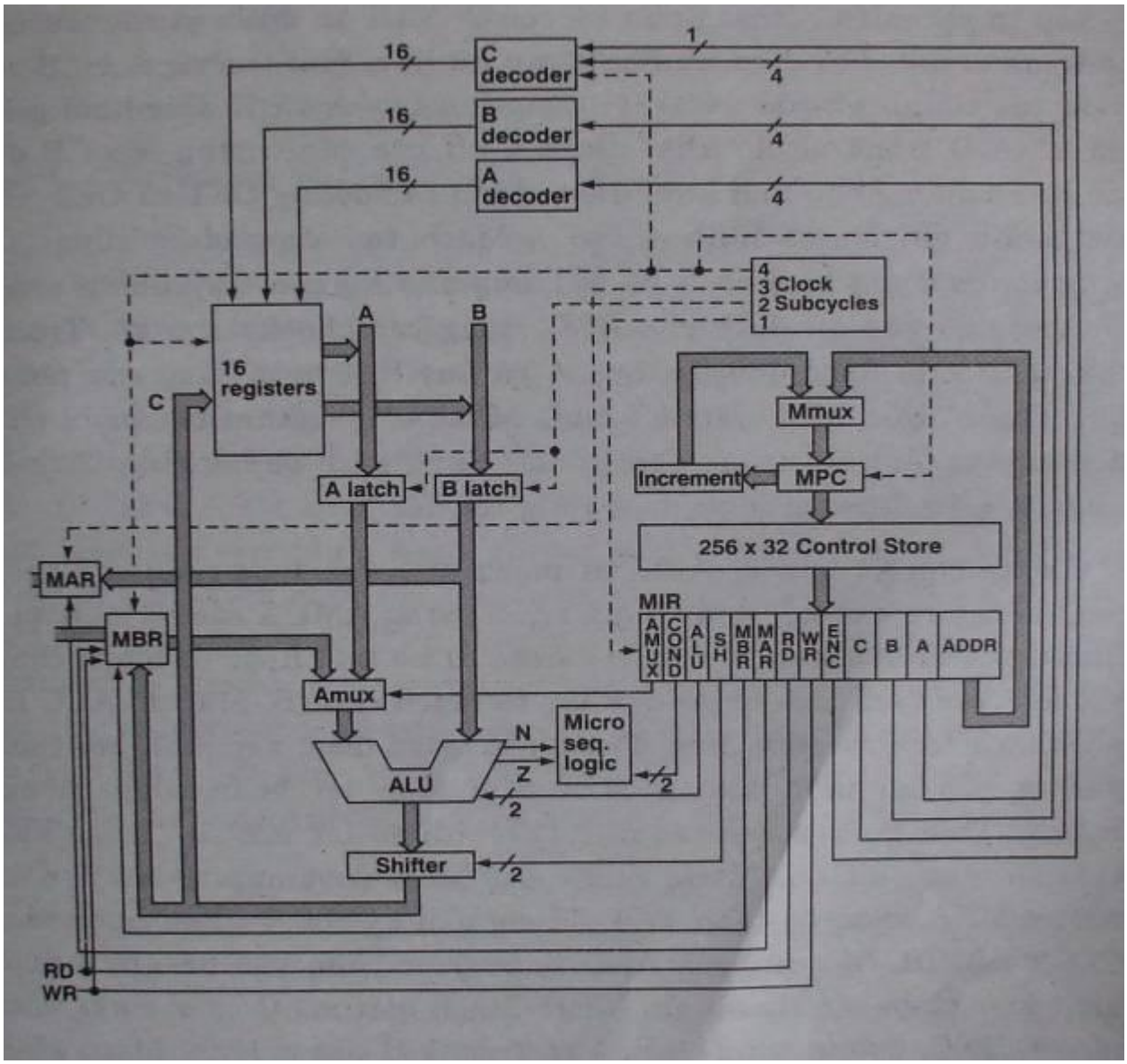
Một chu kì cơ bản của ALU bao gồm thời gian thiết lập các mạch chốt A, B + thời gian ALU và mạch dịch bit thực thi công việc + thời gian cất kết quả. Các sự kiện trên phải xẩy ra theo một chuỗi tuần tự. Để thực hiện được như vậy, người ta sử dụng một mạch tạo xung đồng hồ tạo ra 4 pha xung (có 4 chu kì con) như trong hình 7.5 để điều khiển các sự kiện. Các sự kiện trong chu kì lệnh của ALU được phân bổ theo thời gian của 4 chu lì con là:

1. Trong chu kì con thứ nhất: nạp vi lệnh kế tiếp sẽ được thực thi vào thanh ghi vi lệnh MIR.

2. Trong chu kì con thứ 2: đưa nội dung 2 trong các thanh ghi lên các bus A, B và chốt chúng vào các mạch chốt A, B.

3.Trong chu kì con thứ 3: dữ liệu đưa vào ALU, ALU và mạch dịch bit thực thi công việc, nạp địa chỉ cho MAR nếu có yêu cầu.

4.Trong chu kì con thứ 4: dữ liệu ổn định ở đầu ra của mạch dịch bit, cất dữ liệu trên bus C vào bộ nhớ đệm, và nạp dữ liệu cho MBR nếu được yêu cầu.




Hình 3.10. Sơ đồ khối đầy đủ của vi cấu trúc mẫu

Hình 3.10 là sơ đồ khối chi tiết hoàn chỉnh của vi cấu trúc mẫu. Nó bao gồm 2 phần, phần đường dữ liệu bên trái chúng ta đã khảo sát và phần điều khiển bên phải chúng ta sẽ khảo sát sau đây.

- Một số thành phần quan trọng của phần điều khiển:

1. Control store: bộ nhớ điều khiển dung lượng 256 từ x 32 bit = 8 Kb. Bộ nhớ điều khiển và bộ nhớ chính hoàn toàn khác nhau. Bộ nhớ điều khiển lưu giữ vi chương trình còn bộ nhớ chính chứa chương trình của một ngôn ngữ máy qui ước. Trong một số máy bộ nhớ điều khiển là ROM còn một số khác là RAM.

2. MPC (Micro-Program Counter): bộ đếm vi chương trình có chức năng trỏ đến vi lệnh kế tiếp sẽ được thực thi.

3. MIR (Micro-Instruction Register): thanh ghi vi lệnh chứa vi lệnh sẽ được thực thi.

4. Mmux ((M MUltipleXer): mạch chon kênh M.

5. Clock: mạch tạo xung đồng hồ tạo ra 4 xung đồng hồ lệch pha nhau.

6. A/B/C decoder: các bộ giải mã 4 to 16 chọn 1 trong 16 thanh ghi.

- Từ hình 3.10 chúng ta thấy:

1. Trong chu kì con thứ nhất, đơn vị điều khiển (Control Unit - CU) sao chép vi lệnh được địa chỉ hoá bởi MPC vào thanh ghi MIR (kí hiệu bởi đường nét đứt từ clock đến MIR).

2. Trong chu kì con 2, MIR ổn định, các trường A, B của vi lệnh chọn các thanh ghi để đưa nội dung lên các bus A, B thông qua các mạch giải mã A, B. Clock tác động lên các mạch chốt A, B (kí hiệu bởi đường nét đứt từ clock đến các mạch chốt) để điều khiển việc nạp nội dung trên các bus A, B vào các mạch chốt A, B. Khi dữ liệu đã được đưa lên bus A và B, MPC được tăng thêm 1 nhờ mạch tăng (increment).

3. Trong chu kì con 3, ALU và mạch dịch bit hoàn thành công việc tính toán. Trường AMUX của vi lệnh xác định ngõ vào bên trái cho ALU. Trong khi ALU và mạch dịch bit thực thi công việc, MAR được nạp từ chốt B nếu trường MAR trong vi lệnh là 1.

4. Trong chu kì con 4, tuỳ thuộc vào 2 trường ENC và MBR, kết quả được nạp vào bộ nhớ đệm hay MBR. Trong sơ đồ chúng ta thấy mạch giải mã C được nối với trường ENC, C của vi lệnh và đường xung đồng hồ 4. Như vậy, một thanh ghi nào đó của bộ nhớ đệm chỉ được nạp dữ liệu khi thoả mãn đồng thời 3 điều liện:

- ENC = 1.

- Thanh ghi được chọn bởi trường C.

- Tại chu kì con 4.

MBR được nạp khi trường MBR = 1.

### 2.4. Trình tự vi lệnh

Trong mỗi vi lệnh bổ xung thêm 2 trường (COND = CONDition và ADDR = ADDRess) để quyết định vi lệnh kế tiếp có địa chỉ là MPC + 1 hay ADDR. Việc chọn vi lệnh kế tiếp được quyết định bởi khối logic trình tự vi lệnh (micro sequencing logic) trong khoảng thời gian chu kì con 4 khi tín hiệu ra N, Z của ALU có giá trị. Ngõ ra của khối này sẽ điều khiển mạch chọn kênh Mmux để chọn MPC + 1 hay ADDR đưa vào thanh ghi MPC. Vi lập trình viên có 4 khả năng lựa chọn vi lệnh kế tiếp bằng cách thiết lập trường COND:

COND = 00: không nhẩy, vi lệnh kế tiếp lấy từ địa chỉ MPC + 1.

COND = 01: nhẩy tới ADDR nếu N = 1.

COND = 10: nhẩy tới ADDR nếu Z = 1.

COND = 11: nhẩy tới ADDR vô điều kiện.

## 3. Một cấu trúc Macro (Macro-Architecture) mẫu

Để tiếp tục ví dụ cho cấp máy vi lập trình (cấp máy 1), chúng ta chuyển qua cấu trúc cấp máy qui ước được hỗ trợ bởi trình phiên dịch chạy trên cấp máy vi lập trình. Chúng ta gọi cấu trúc của cấp máy 2 hoặc 3 là cấu trúc macro (macro-architecture). Với mục đích của chương này, chúng ta sẽ bỏ qua cấp máy 3 vì các chỉ thị của cấp 3 phần lớn thuộc cấp 2. Các chỉ thị của cấp máy 2 sẽ được gọi là các chỉ thị macro (macro-instruction). Vì vậy, các chỉ thị như là ADD, MOVE,... của cấp máy 2 sẽ được gọi là chỉ thị macro. Chúng ta sẽ đề cập đến máy cấp 1 (Mic-1) và máy cấp 2 (Mac-1) trong ví dụ. Trước hết chúng ta tìm hiểu một số vấn đề có liên quan.

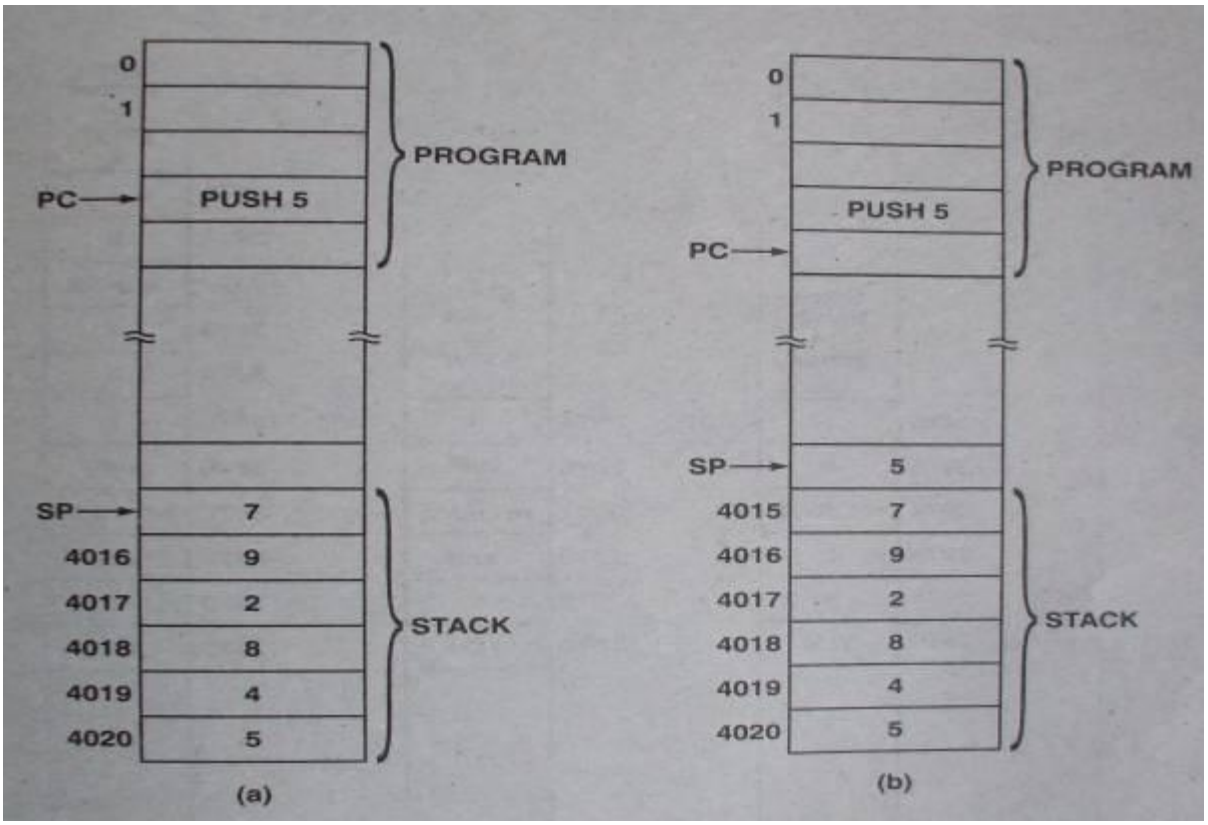
### 3.1. Stack

```
{CT tinh tich vo huong cua 2 vecto: x(1,2,...,20)}
{va y voi y[i] = (2*x[i]+1)}

Program InnerProduct;
uses crt;
const max = 20;
type Smallint = 0..100;
     vec = array[1..max] of Smallint;
var k:integer;
    x, y:vec;
```

```
Function pmul(a,b:Smallint):integer;
var p,j:integer;
begin
     if (a=0) or (b=0) then
        pmul:=0
        else
            begin
                P:=0;
                for j:=1 to a do
                   p:=p+b;
                pmul:=p;
            end;
end;
Procedure inner(var v:vec;var ans:integer);
var sum,i:integer;
begin
     sum:=0;
     for i:=1 to max do
         sum:=sum + pmul(x[i],v[i]);
     ans:=sum;
end;

Begin
     clrscr;
     For k:=1 to max do
     begin
          x[k]:=k;
          y[k]:=pmul(2,k)+1;
     end;
     inner(y,k);
     Write('Tich vo huong cua 2 vecto: ',k);
     readln;
End.
```

Hình 3.11. Chương trình tính tích vô hướng 2 vecto bằng Pascal

Stack (ngăn xếp) là một khối liên tục của bộ nhớ dùng để lưu trữ dữ liệu khi chương trình được thực hiện. Con trỏ stack SP (Stack Pointer) trỏ đến đỉnh của stack. Đáy của stack có địa chỉ cố định. Có 2 thao thao tác quan trọng nhất của stack là PUSH X (đẩy X vào đỉnh stack, SP giảm đi 1) và POP Y (lấy phần tử trên đỉnh stack lưu vào Y, SP tăng thêm 1). Hình 3.12 minh hoạ stack trước và sau khi thực hiện chỉ thị PUSH 5.




Hình 3.12. Stack trước khi thực hiện chỉ thị PUSH 5 và sau khi thực hiện chỉ thị

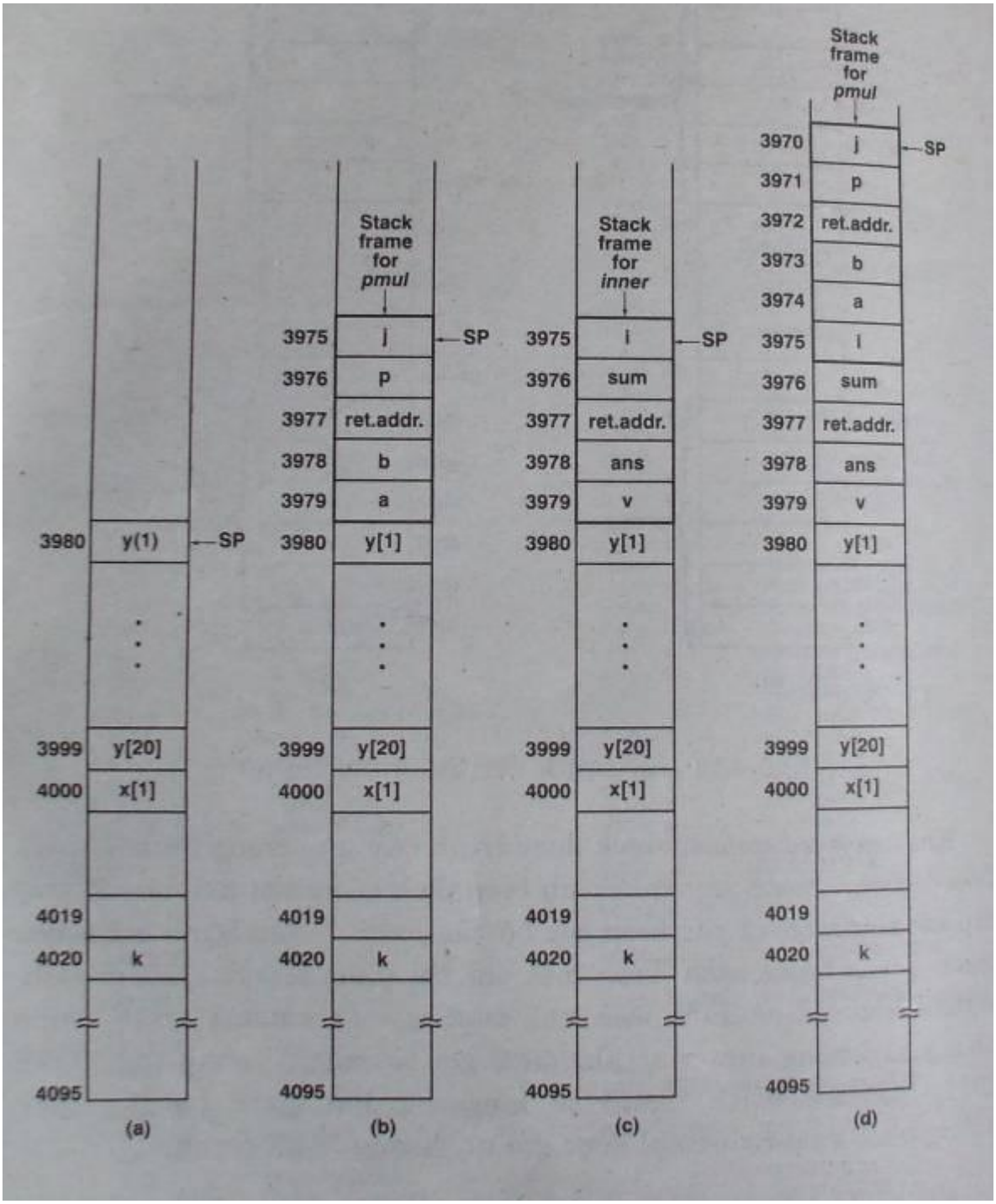
Hình 3.13 trình bày sự hoạt động của stack khi thực hiện chương trình ở hình 3.11. Chúng ta giả thiết bộ nhớ gồm 4096 từ 16 bit, các từ có địa chỉ 4021 - 4095 được sử dụng bởi hệ điều hành.

+ Hình 3.13 (a) trình bày cách thức bộ nhớ được cấp phát khi thực hiện chương trình chính. Biến k được cấp phát địa chỉ 4020, x[i] (i = 1, 2,...,20) được cấp phát từ địa chỉ 4000 - 4019, y[i] (i = 1, 2,...,20) được cấp phát từ địa chỉ 3980 - 3999. SP = 3980.

+ Hình 3.13 (b) trình bày cách thức bộ nhớ được cấp phát khi thực hiện hàm pmul. 5 địa chỉ trên đỉnh stack tạo thành một khung stack cho pmul sử dụng. Chúng sẽ được giải phóng khi pmul hoàn tất công việc. Khi chương trình chính muốn gọi pmul, trước tiên nó phải cất các tham số của chỉ thị gọi (2, k) vào stack tại địa chỉ 3979 và 3978 thay cho 2 biến hình thức a và b. Sau đó nó thực hiện chỉ thị gọi, chỉ thị này sẽ cất địa chỉ trở về vào stack (re.addr - chứa địa chỉ của lệnh tiếp theo lệnh gọi pmul) để pmul biết vị trí trở về khi hoàn tất công việc. Như vậy, trước khi pmul bắt đầu thực hiện, SP = 3977. Điều đầu tiên mà pmul làm là đẩy con trỏ stack lên 2 đơn vị dành 2 địa chỉ nhớ cho 2 biến p và j.

+ Khi pmul trở về, inner được gọi. Hình 3.13 (c) trình bày cách thức bộ nhớ được cấp phát khi thực hiện thủ tục inner trước khi inner gọi hàm pmul.

+ Khi inner gọi hàm pmul, việc cấp phát bộ nhớ được trình bày trong hình 3.13 (d).




Hình 3.13. Việc cấp phát bộ nhớ khi thực hiện chương trình InnerProduct

## 3.2. Tập chỉ thị macro

| Binary | Mnemonic | Instruction | Meaning |
|---|---|---|---|
| 0000xxxxxxxxxxxx | LODD | Load direct | *ac:=m*[x] |
| 0001xxxxxxxxxxxx | STOD | Store direct | *m*[x]:=ac |
| 0010xxxxxxxxxxxx | ADDD | Add direct | *ac:=ac+m* [x] |
| 0011xxxxxxxxxxxx | SUBD | Subtract direct | *ac:=ac-m* [x] |
| 0100xxxxxxxxxxxx | JPOS | Jump positive | if *ac*≥0 then *pc:=x* |
| 0101xxxxxxxxxxxx | JZER | Jump zero | if *ac*≠0 then *pc:=x* |
| 0110xxxxxxxxxxxx | JUMP | Jump | *pc:=x* |
| 0111xxxxxxxxxxxx | LOCO | Load constant | *ac:=x* (0≤*x*≤4095) |
| 1000xxxxxxxxxxxx | LODL | Load local | *ac:=m*[sp+x] |
| 1001xxxxxxxxxxxx | STOL | Store local | *m*[sp+x]:=ac |
| 1010xxxxxxxxxxxx | ADDL | Add local | *ac:=ac+m*[sp+x] |
| 1011xxxxxxxxxxxx | SUBL | Subtract local | *ac:=ac-m*[sp+x] |
| 1100xxxxxxxxxxxx | JNEG | Jump negative | if *ac*<0 then *pc:=x* |
| 1101xxxxxxxxxxxx | JNZE | Jump nonzero | if *ac*≠0 then *pc:=x* |
| 1110xxxxxxxxxxxx | CALL | Call procedure | *sp:=sp-1;m*[sp]:=*pc;pc:=x* |
| 1111000000000000 | PSHI | Push indirect | *sp:=sp-1;m*[sp]:=*m*[*ac*] |
| 1111001000000000 | POPI | Pop indirect | *m*[*ac*]:=*m*[sp];*sp:=sp*+1 |
| 1111010000000000 | PUSH | Push onto stack | *sp:=sp*-1;*m*[*sp*]:=*ac* |
| 1111011000000000 | POP | Pop from stack | *ac:=m*[*sp*];*sp:=sp*+1 |
| 1111100000000000 | RETN | Return | *pc:=m*[*sp*];*sp:=sp*+1 |
| 1111101000000000 | SWAP | Swap ac, sp | *tmp:=ac;ac:=sp;sp:=tmp* |
| 11111100yyyyyyyy | INSP | Increment sp | *sp:=sp+y*(0≤*y*≤255) |
| 11111110yyyyyyyy | DESP | Decrement sp | *sp:=sp-y*(0≤*y*≤255) |

xxxxxxxxxxxx is a 12-bit machine address; in column 4 it is called *x*.
yyyyyyyy is an 8-bit constant; in column it is called *y*.

Hình 3.14. Tập chỉ thị của Mac-1

Chúng ta xét cấu trúc của máy Mac-1:

+ Về cơ bản Mac-1 gồm một bộ nhớ 4 k từ 16 bit, 3 thanh ghi truy xuất được bởi lập trình viên trên máy cấp 2 là PC (Program counter = bộ đếm chương trình), SP (Stack Pointer = con trỏ stack), AC (ACcumulator = tích luỹ).

+ Có 3 kiểu định địa chỉ (phương pháp xác định vị trí nhớ của một toán hạng) được sử dụng: trực tiếp, gián tiếp và cục bộ. Chỉ thị dùng kiểu định địa chỉ trực tiếp chứa một địa chỉ bộ nhớ tuyệt đối 12 bit thấp, kiểu này thường được dùng để truy xuất các biến toàn cục. Trong kiểu định địa chỉ giám tiếp, địa chỉ nhớ không được đưa ra trực tiếp mà được đặt vào thanh ghi (lấy thanh ghi làm trung gian để xác định địa chỉ bộ nhớ), dạng địa chỉ này thường được dùng để truy xuất các phần tử dãy. Kiểu định địa chỉ cục bộ cho biết độ dời của SP, thường được dùng để truy xuất các biến cục bộ.

Tập chỉ thị của Mac-1 được trình bày trong hình 3.14. Mỗi chỉ thị chứa mã thao tác (opcode), một địa chỉ bộ nhớ hoặc một hằng số. Cột1: mã nhị phân của chỉ thị, cột 2: mã hợp ngữ, cột 3: tên chỉ thị, cột 4: mô tả chỉ thị thực hiện thao tác gì bằng ngôn ngữ Pascal.

Trong hình 3.14, xx...x ở cột 1 là địa chỉ / hằng số 12 bit, trong cột 4 gọi là x. m[x] là từ nhớ có địa chỉ x. Tương tự yy...y là hằng số 8 bit, trong cột 4 gọi là y.

+ Các lệnh LODD, STOD, ADDD, SUBD thực hiện 4 chức năng cơ bản: nạp, lưu trữ, cộng, trừ bằng cách dùng kiểu định địa chỉ trực tiếp.

+ Các lệnh LODL, STOL, ADDL, SUBL thực hiện 4 chức năng cơ bản: nạp, lưu trữ, cộng, trừ bằng cách dùng kiểu định địa chỉ cục bộ.

+ 5 lệnh nhảy JUMP (nhảy vô điều kiện), JPOS (nhảy nếu ac > 0), JZER (nhảy nếu ac = 0), JNEG (nhảy nếu ac < 0), và JNZE (nhảy nếu ac $\neq$ 0.

+ LOCO nạp hằng số 12 bit (0 - 4095).

+ CALL dùng để gọi thủ tục, chỉ thị này giảm SP đi 1 (lấy chỗ cất nội dung của PC vào đỉnh stack - m[sp] := pc), sau đó thực hiện thủ tục gọi (pc := x). Thực hiện xong quay trở lại vị trí chương trình đã cất vào stack (lệnh RETN). Lệnh RETN đưa con trỏ chương trình về vị trí đã cất vào stack, giải phóng địa chỉ nhớ lưu giữ địa chỉ quay về (sp := sp + 1).

+ PUHI cất vào đỉnh stack từ nhớ có địa chỉ chứa trong AC. POPI lấy từ nhớ ra khỏi stack và cất vào từ nhớ có địa chỉ chứa trong AC.

+ PUSH đẩy vào đỉnh stack nội dung của AC. POP lấy từ nhớ ra khỏi stack cất vào AC.

+ SWAP trao đổi nội dung của AC và SP, được sử dụng khi SP phải tăng hoặc giảm một đại lượng chưa biết tại thời điểm biên dịch.

+ INSP, DESP dùng để thay đổi SP một đại lượng đã biết tại thời điểm biên dịch.

## 4. Một vi chương trình mẫu

Chúng ta đã khảo sát vi cấu trúc (cấu trúc của máy cấp 1- cấp vi chương trình) và cấu trúc macro (cấu trúc của máy cấp 2 - cấp máy qui ước), trong phần này chúng ta sẽ xem xét vi chương trình sẽ thực hiện điều gì trên vi cấu trúc và vi chương trình sẽ phiên dịch điều gì trên cấu trúc macro. Trước hết, chúng ta xem xét kỹ việc sử dụng ngôn ngữ nào để thực hiện vi lập trình.

### 4.1. Vi hợp ngữ

Chúng ta có thể viết vi chương trình dưới dạng số nhị phân, mỗi vi lệnh dài 32 bit. Tuy nhiên, đây là một việc làm khó khăn mà chỉ những chuyên gia lập trình hệ thống mới quan tâm đến. Vì vậy, chúng ta cần một ngôn ngữ tượng trưng để diễn tả các vi chương trình. Vi chương trình viên có thể chỉ rõ một vi lệnh trên một dòng (gán giá trị khác 0 cho các trường trong vi lệnh). Ví dụ, lệnh cộng nội dung của thanh ghi AC với nội dung của thanh ghi A, kết quả chứa vào thanh ghi AC được viết như sau:

ENC = 1, C = 1, B = 1, A = 10. (1)

(ENC = 1, C = 1: cho phép cất kết quả từ bus C vào thanh ghi số 1 (AC) ; B = 1, A = 10 : đưa dữ liệu từ thanh ghi số 1 (AC) vào bus B và thanh ghi số 10 (A) vào bus A).

Tuy nhiên, viết như trên rất tối nghĩa và khó hiểu. Một ý tưởng tốt hơn là sử dụng ký hiệu của ngôn ngữ cấp cao và vẫn giữ khái niệm cơ bản của dòng ký hiệu trên cho mỗi vi lệnh. Người ta có thể dùng các ngôn ngữ bậc cao để viết các vi chương trình, nhưng do tính hiệu quả của vi chương trình nên chúng ta sẽ dùng hợp ngữ vì hợp ngữ có ánh xạ 1 - 1 với ngôn ngữ máy. Chúng ta sẽ gọi hợp ngữ được dùng để viết vi chương trình là vi hợp ngữ (Mciro Assembly Language - MAL). Ở đây, để tăng tính hiểu được, người ta dùng MAL gần giống như Pascal. Trong ngôn ngữ MAL, việc chứa vào 16 thanh ghi của bộ nhớ đệm hoặc MAR và MBR được biểu thị bằng phép gán. Như vậy, trong ngôn ngữ MAL, (1) được viết: ac := a + ac.

Ví dụ, các lệnh sử dụng các chức năng 0, 1, 2, 3 của ALU được viết trong ngôn ngữ MAL: ac := a + ac; a := band(ir, smack); ac := a; a := inv(a); trong đó, band = boolean and (phép and lôgic) và inv = invert (lấy phủ định).

Các thao tác dịch bit sang phải (trái) được ký hiệu là rshift (lshift). Lệnh tir := lshift(tir + tir); đặt nội dung của tir lên cả 2 bus A và B, thực hiện phép cộng và

dịch trái kết quả sang trái 1 bit (tương đương tir được nhân 4).

Chỉ thị nhẩy vô điều kiện được phát biểu bởi goto, các chỉ thị nhẩy có điều kiện sẽ kiểm tra n hoặc z. Ví dụ, if n then goto 27.

Các phát biểu gán và chỉ thị nhảy có thể được kết hợp trên cùng một dòng. Ví dụ: alu := tir; if n then goto 27. Dòng lệnh trên có nghĩa là: đưa nội dung của tir vào alu để kiểm tra bit cao nhất của tir (n là bản sao bit cao nhất trong ALU), nếu N = 1, nhẩy tới 27.

Để chỉ ra các thao tác đọc và ghi bộ nhớ, chúng ta chỉ phải đặt rd và wr vào chương trình nguồn. Về nguyên tắc, trật tự các phần khác nhau của phát biểu nguồn là tuỳ ý, nhưng để tăng khả năng đọc hiểu, chúng ta nên sắp xếp chúng theo thứ tự chúng được thực hiện. Hình 3.15 là một số phát biểu trong ngôn ngữ MAL và các vi lệnh tương ứng.

| | AMUX | COND | ALU | SH | MBR | MAR | RD | WR | ENC | C | B | A | ADDR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| mar:=pc;rd | 0 | 0 | 2 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 00 |
| rd | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 00 |
| ir:=mbr | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 3 | 0 | 0 | 00 |
| pc:=pc+1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 6 | 0 | 00 |
| mar:=ir, mbr:=ac; wr | 0 | 0 | 2 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 3 | 1 | 00 |
| alu:=tir;if n then goto 15 | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 15 |
| ac:=inv (mbn) | 1 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 00 |
| tir:=lshift (tir); if n then goto 25 | 0 | 1 | 2 | 2 | 0 | 0 | 0 | 0 | 1 | 4 | 0 | 4 | 25 |
| alu:=ac;if z then goto 22 | 0 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 22 |
| ac:=band (ir, amask);goto 0 | 0 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 8 | 3 | 00 |
| sp:=sp+(-1); rd | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 2 | 2 | 7 | 00 |
| tir:=lshift (ir+ir); if n then goto 69 | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 1 | 4 | 3 | 3 | 69 |

Hình 3.15. Một số phát biểu của MAL và các vi lệnh tương ứng

### 4.2. Vi chương trình mẫu

Đến đây, chúng ta có thể làm rõ vai trò của các thanh ghi trong hình 3.8:

+ PC (Program Counter), AC (ACcumulator), SP (Stack Pointer) là các thanh ghi bộ đếm chương trình, tích luỹ, và con trỏ stack.

+ IR (Instruction Register) là thanh ghi chỉ thị; TIR (Temporary Instruction Register) là thanh ghi chỉ thị nháp, được dùng để giải mã opcode.

+ 3 thanh ghi 0, +1, -1 được dùng để lưu giữ các hằng số đã được chỉ định.

+ AMASK (Address MASK) là thanh ghi mặt nạ địa chỉ, được dùng để phân biệt các bit opcode và các bit địa chỉ; SMASK (Stack MASK) là thanh ghi mặt nạ stack, được dùng để tách riêng offset 8 bit trong các chỉ thị INSP và DESP.

+ 6 thanh ghi còn lại A, B,..., F (các thanh ghi chung) không ấn định chức năng, lập trình viên tuỳ ý sử dụng.

Giống như tất cả các trình phiên dịch, vi chương trình có một vòng lặp chính:

1. Tìm nạp lệnh từ bộ nhớ vào thanh ghi IR.

2. Giải mã lệnh.

3. Thực hiện lệnh.

Hình 3.16 là một vi chương trình mẫu chạy trên máy Mic-1 và phiên dịch trên máy Mac-1.

```
0: mar := pc; rd;
1: pc := pc + 1; rd;
2: ir := mbr; if n then goto 28;
3: tir := lshift (ir + ir); if n then goto 19;
4: tir := lshift (tir); if n then goto 11;
5: alu := tir; if n then go to 9;
6: mar := ir; rd;
7: rd;
8: ac := mbr; goto 0;
9: mar := ir; mbr :=ac; wr;
10: wr; goto 0;
11: alu := tir; if n then goto 15;
12: mar := ir; rd;
13: rd;
14: ac := mbr + ac; goto 0;
15: mar := ir; rd;
16: ac := ac + 1; rd;
17: a := ivn(mbr);
18: ac := ac + a; goto 0;
19: tir := lshift (tir); if n goto 25;
20: alu := tir; if n goto 23;
21: alu := ac; if n goto 0;
```

```
22: pc := band (ir, amask); goto 0;
23: alu := ac; if z then goto 22;
24: goto 0;
25: alu :=tir; if n then goto 27;
26: pc := band (ir, amask); goto 0;
27: ac := band (ir, amask); goto 0;
28: tir := lshift (ir + ir);if n then goto 40;
29: tir := lshift (tir); if n then goto 35;
30: alu = tir; if n then goto 33;
31: a := ir + sp;
32: mar :=a; rd; goto 7;
33: a := ir + sp;
34: mar := a; mbr := ac; wr; goto 10;
35: alu := tir; if n then goto 38;
36: a := ir + sp;
37: mar := a; rd; goto 13;
38: a:= ir + sp;
40: tir := lshift (tir); if n then goto 46;
41: alu := tir; if n then goto 44;
42: alu := ac; if n then goto 22;
43: goto 0;
44: alu := ac; if z then goto 0;
45: pc := band (ir, amask); goto 0;
46: tir := lshift (tir); if n then goto 50;
47: sp := sp - 1;
48: mar:= sp; mbr := pc;wr;
49: pc:= band (ir, amask);wr; goto 0;
50: tir := lshift (tir); if n then goto 65;
51: tir := lshift (tir); if n then goto 59;
52: alu := tir; if n then goto 56;
53: mar := ac; rd;
54: sp := sp - 1; rd;
55: mar :=sp; wr; goto 10;
56: mar:= sp; sp := sp + 1;rd;
57: rd;
58: mar := ac; wr;goto 10;
60: sp := sp - 1;
61: mar := sp;mbr := ac; wr; goto 10;
62: mar:= sp; sp := sp + 1;rd;
63: rd;
```

64: ac := mbr;goto 0;
65: tir := lshift (tir);if n then goto 73;
66: alu := tir; if n then goto 70;
67: mar := sp; sp := sp + 1; rd;
68: rd;
69: pc := mbr; goto 0;
70: a := ac;
71: ac := sp;
72: sp := a; goto 0;
73: alu := tir; if n then goto 76;
74: a := band (ir, smask);
75: sp := sp + a;goto 0;
76: a := band (ir, smask);
77: a := inv(a);
78: a := a + 1; goto 75;

Hình 3.16: Vi chương trình

- **Giải thích:**

0: mar := pc; rd; {copy [PC] tới MAR, đọc ND ô nhớ có địa chỉ [MAR] vào MBR (mbr := m[mar])}

1: pc := pc + 1; rd; {tăng [PC] lên 1, vi CT đợi đọc m[mar] vào MBR}

2: ir := mbr; if n then goto 28; {copy [MBR] tới IR, đồng thời bit 15 được KT. Nếu bit 15 là 1 nhảy sang dòng 28 còn không vi CT chuyển sang dòng 3}

3: tir := lshift (ir + ir); if n then goto 19; {bit trái cùng là 0, [IR] được copy vào 2 bus A, B, thực hiệu phép cộng trong ALU (tương đương dịch trái 1 bit), dịch trái kết quả bit và cất trở lại TIR. Đồng thời bit cao nhất trong ALU (bit 14) được kiểm tra, nếu N = 1 nhảy sang dòng 19 còn không chuyển sang dòng 4}

4: tir := lshift (tir); if n then goto 11; {2 bit trái cùng là 00, [TIR] cho qua ALU, dịch trái 1 bit, cất trở lại TIR. Đồng thời bit cao nhất trong ALU (bit 13) được kiểm tra, nếu N = 1 nhảy sang dòng 11 còn không chuyển sang dòng 5}

5: alu := tir; if n then go to 9; {3 bit trái cùng là 000, KT bit 12, nếu là 1 nhảy sang dòng 9 còn không chuyển xuống dòng 6}

6: mar := ir; rd; {mã lệnh là 0000 = LOAD (ac := m[x]), copy [IR] tới MAR, vì MAR chỉ rộng 12 bit nên chỉ có 12 bit thấp (chứa địa chỉ) của IR vào MAR, m[mar] được đọc từ bộ nhớ và nạp vào MBR (mbr := m[mar])}

7: rd; {Vi CT không làm gì cả, đợi đọc từ nhớ}

8: ac := mbr; goto 0; {copy [MBR] tới AC, vi chương trình nhảy về dòng 0 - đầu vòng lặp}

9: mar := ir; mbr :=ac; wr; {mã lệnh là 0001 = STOD (m[x] := ac), copy [IR] tới MAR, copy [AC] vào MBR, ghi [MBR] địa chỉ [MAR] (m[mar] := [mbr])}

10: wr; goto 0; {đợi ghi, ghi xong vi CT nhảy về dòng 0}

11: alu := tir; if n then goto 15; {kiểm tra bit 12, nếu 1 nhảy sang dòng 15 thực hiện lệnh SUBD, còn không chuyển xuống dòng 12 thực hiện lệnh ADDD}

12: mar := ir; rd; {mã lệnh là 0010 = ADDD (ac := ac + m[x]), copy [IR] tới MAR, m[mar] được đọc từ bộ nhớ và nạp vào MBR (mbr := m[mar])}

13: rd; {đợi đọc}

14: ac := mbr + ac; goto 0; {m[mar] + [AC], kết quả chứa vào AC, vi CT nhảy về đầu vòng lặp}

15: mar := ir; rd; {mã là 0011 = SUBD (ac := ac - m[x]), các dòng 15, 16, 17, 18 thực hiện phép trừ. Dòng 15 copy [IR] tới MAR, m[mar] được đọc từ bộ nhớ và nạp vào MBR (mbr := m[mar])}

16: ac := ac + 1; rd; {vì x - y = x + ($\overline{y}$ + 1) = x + 1 + inv(y), lệnh này tăng ND AC lên 1, RD vẫn tích cực}

17: a := ivn(mbr); {lấy nghịch đảo của m[mar] cất vào thanh ghi A}

18: ac := ac + a; goto 0; {thực hiện phép trừ, cất kết quả vào AC}

19: tir := lshift (tir); if n goto 25; {2 bit trái cùng 01, bit 13 được kiểm tra, ND của tir được đi qua ALU, sau đó dịch trái 1 bit. Nếu n = 1, vi CT nhảy sang dòng 25 còn không xuống dòng 20}

20: alu := tir; if n goto 23; {3 bit trái 010, ND của tir qua ALU, bit 12 được kiểm tra, nếu 1 nhảy xuống dòng 23, 0 xuống dòng 21}

21: alu := ac; if n goto 0; {mã lệnh 0100 = JPOS (if ac $\geq$ 0 then pc := x). Dòng 21 cho [AC] qua ALU, kiểm tra bit cao nhất, nếu n = 1 tức AC < 0, vi CT nhảy về dòng 0, còn không xuống dòng 22}

22: pc := band (ir, amask); goto 0; {and 12 bit địa chỉ của chỉ thị, cất kết quả vào PC (pc := x, lệnh sắp thực hiện có địa chỉ x), sau đó vi CT nhảy về dòng 0}

23: alu := ac; if z then goto 22; {mã lệnh 0101 = JZER (if ac = 0 then pc := x). Dòng 23 cho [AC] qua ALU, nếu z = 1 tức AC = 0, vi CT nhảy tới dòng 22, còn không xuống dòng 24}

24: goto 0;

25: alu :=tir; if n then goto 27; { 3 bit trái cùng 011. Dòng 25 kiểm tra bit 12, nếu n = 1 chuyển sang dòng 27, còn không xuống dòng 26}

26: pc := band (ir, amask); goto 0; {mã là 0110 = JUMP (pc := x). Dòng này thực hiện pc := x, sau đó vi CT nhảy về dòng 0}

27: ac := band (ir, amask); goto 0; {mã lệnh là 0111 = LOCO (ac := x). Dòng này and 12 bit địa chỉ của chỉ thị, cất vào AC, sau đó nhảy về dòng 0}

28: tir := lshift (ir + ir);if n then goto 40; {bit trái cùng là 1. Dòng này đặt nội dung của tir lên cả 2 bus A và B, thực hiện phép cộng (tương đương dịch trái 1 bit) kiểm tra bit 14 của kết quả nếu 1 nhảy tới dòng 40 còn không xuống dòng 29. Kết quả ra của ALU được dịch trái 1 bit sau đó cất vào tir}

29: tir := lshift (tir); if n then goto 35; {2 bit trái cùng là 10. Dòng này thực hiện: [tir] được cho qua ALU bit cao nhất (bit 13) được kiểm tra, [tir] được dịch trái 1 bit và cất trở lại tir. Nếu n = 1 chuyển sang dòng 35 còn không xuống dòng 30}

30: alu = tir; if n then goto 33; {3 bit trái cùng là 100. Dòng này thực hiện: [tir] cho qua ALU để kiểm tra bit 12, nêú là 1 chuyển sang dòng 33 còn không xuống dòng 31}

31: a := ir + sp; {mã lệnh là 1000 = LODL (ac := m[sp + x]). Dòng này thực hiện: [IR] và [SP] đi vào 2 bus A, B thực hiện phép cộng trong ALU, cất keets quar vào A}

32: mar :=a; rd; goto 7; {copy 12 bit thấp của A vào MAR (MAR chỉ rộng 12 bit), m[mar] được đọc vào mbr, chuyển sang dòng 7}

33: a := ir + sp; {mã lệnh là 1001 = STOL (m[sp+x] := ac). Dòng này thực hiện: [IR] + [SP] (x + sp), cất kết quả vào A}

34: mar := a; mbr := ac; wr; goto 10; {copy 12 bit thấp của A vào MAR, copy [AC] vào MBR, ghi ND của MBR vào từ nhớ có địa chỉ [MAR], chuyển sang dòng 10}

35: alu := tir; if n then goto 38; {3 bit trái cùng là 101. Dòng này kiểm tra bit 12, nếu n = 1 chuyển sang dòng 38 còn không xuống dòng 36}

36: a := ir + sp; { mã lệnh là 1010 = ADDL (ac := ac + m[sp + x]). Dòng này thực hiện: [IR] + [SP] (x + sp), cất kết quả vào A}

37: mar := a; rd; goto 13; {copy 12 bit thấp của A vào MAR, đọc m[mar] vào MBR, chuyển sang dòng 13}

38: a:= ir + sp; { mã lệnh là 1011 = SUBL (ac := ac - m[sp + x])}

39: mar := a; rd; goto 16;{copy 12 bit thấp của A vào MAR, đọc m[mar] vào MBR, chuyển sang dòng 16}

40: tir := lshift (tir); if n then goto 46; {2 bit trái cùng là 11, [tir] được dịch trái, trước khi dịch trái kiểm tra bit 13, nếu 1 chuyển tới dòng 46, còn không xuống dòng 41}

41: alu := tir; if n then goto 44; {3 bit đầu là 110. Dòng này cho tir qua ALU, kiểm tra bit 12, nếu 1 chuyển tới dòng 44, còn không xuống dòng 42}

42: alu := ac; if n then goto 22; {4 bit trái cùng là 1100 = JNEG (if ac < 0 then pc := x). Vi lệnh này kiểm tra bit cao nhất của ac, nếu 1 tức ac < 0 thì vi CT nhảy tới dòng 22, còn không xuống dòng 43}

43: goto 0; {về đầu CT}

44: alu := ac; if z then goto 0; {mã lệnh là 1101 = JNZE (if ac ≠ 0 then pc := x). Vi lệnh này kiểm tra [ac], nếu ac ≠ 0 thì vi CT xuống dòng 45}

45: pc := band (ir, amask); goto 0; {lấy địa chỉ của chỉ thị đưa vào PC, vi chương trình về dòng 0}

46: tir := lshift (tir); if n then goto 50; {3 bit trái cùng là 111. Vi lệnh này cho [tir] qua ALU, kiểm tra bit 12, sau đó dịch trái 1 bit. Nếu bit 12 là 1 chuyển tới dòng 50, còn không xuống dòng 47}

47: sp := sp - 1; {mã lệnh là 1110 = CALL (sp := sp-1;m[sp] := pc; pc :=x). Dòng này thêm một từ nhớ trên đỉnh stack}

48: mar:= sp; mbr := pc;wr; {copy địa chỉ đỉnh stack vào MAR; copy [pc] vào MBR, WR tích cực (ghi ND của MBR vào ô nhớ có địa chỉ chứa ở MAR). Dòng này thực hiện m[sp] := pc}

49: pc:= band (ir, amask);wr; goto 0; {pc := x, chờ ghi, ghi xong vi CT về dòng 0}

50: tir := lshift (tir); if n then goto 65; {4 bit trái cùng là 1111. Vi lệnh này cho [tir] qua ALU, kiểm tra bit 11, sau đó dịch trái 1 bit. Nếu bit 11 là 1 chuyển tới dòng 65, còn không xuống dòng 51}

51: tir := lshift (tir); if n then goto 59; {5 bit trái cùng là 11110. Vi lệnh này cho [tir] qua ALU, kiểm tra bit 10, sau đó dịch trái 1 bit. Nếu bit 10 là 1 chuyển tới dòng 59, còn không xuống dòng 52}

52: alu := tir; if n then goto 56; {6 bit trái cùng là 111100, kiểm tra bit 9 nếu n = 1 chuyển tới dòng 56, còn không xuống dòng 53}

53: mar := ac; rd; {mã lệnh là 1111000 = PSHI (sp := sp-1; m[sp] := m[ac]. Dòng này thực hiện: copy [AC] vào MAR, đọc m[mar] vào MBR}

54: sp := sp - 1; rd; {thêm một ô nhớ trên đỉnh stack, chờ đọc}

55: mar :=sp; wr; goto 10; {vi lệnh này ghi ND của MBR vào ô nhớ có địa chỉ [sp]}

56: mar:= sp; sp := sp + 1;rd; {mã lệnh là 1111001 = POPI (m[ac] := m[sp]; sp := sp + 1, đọc từ nhớ ở đỉnh stack cất vào ô nhớ có địa chỉ [AC]). Dòng này thực hiện: copy [SP] vào MAR, [SP] tăng thêm 1, đọc từ nhớ có địa chỉ chứa ở MAR vào MBR, tức [MBR] := m[sp]}

57: rd; {đợi đọc}

58: mar := ac; wr;goto 10; {[MAR] := [AC], m[mar] := [MBR] = m[sp]}

59: alu := tir;if n then goto 62; {6 bit trái cùng là 111101. Kiểm tra bit 9 nếu 1, sang dòng 62, còn không xuống dòng 60}

60: sp := sp - 1; {Mã lệnh là 1111010 = PUSH (sp := sp-1; m[sp] := ac). Dòng này thêm 1 từ nhớ vào đỉnh stack}

61: mar := sp;mbr := ac; wr; goto 10; {[MAR] := [SP], [MBR] := [AC], ghi ND của MBR vào từ nhớ có địa chỉ [MAR] tức m[sp] := ac, sang dòng 10}

62: mar:= sp; sp := sp + 1;rd; {Mã lệnh là 1111011 = POP (ac := m[sp]; sp := sp + 1). Dòng này thực hiện: [MAR] := [SP], giải phóng từ nhớ trên đỉnh stack, đọc từ nhớ có địa chỉ ở MAR vào MBR tức MBR := m[sp]}

63: rd; {đợi đọc}

64: ac := mbr;goto 0; {[AC] := [MBR], tức ac := m[sp]}

65: tir := lshift (tir);if n then goto 73; {5 bit trái cùng là 11111. Dòng này thực hiện: Cho [TIR] qua ALU, Kiểm tra bit 10 của TIR, dịch trái 1 bit. Nếu bit 10 là 1 sang dòng 73, còn không xuống dòng 66}

66: alu := tir; if n then goto 70; {6 bit trai cùng là 111110. Kiểm tra bit 9, nếu 1 sang dòng 70, còn không xuống dong 67}

67: mar := sp; sp := sp + 1; rd; {mã lệnh là 1111100 = RETN (pc := m[sp]; sp := sp + 1), mã lệnh này lấy lại địa chỉ nhớ trở về đã cất vào stack đưa vào PC. Các dòng 67, 68, 69 thực hiện mã lệnh này}

68: rd;

69: pc := mbr; goto 0;

70: a := ac; {mã lệnh là 1111101 = SWAP = tmp := ac; ac := sp; sp := tmp). Các dòng vi lệnh 70, 71, 72 thực hiện mã lệnh này}

71: ac := sp;

72: sp := a; goto 0;

73: alu := tir; if n then goto 76; {6 bit trái là 111111. Kiểm tra bit 9, nếu 1 sang dòng 76, còn không xuống dòng 74}

74: a := band (ir, smask); {mã lệnh là 1111110 = INSP (sp := sp + y). Dòng 74, 75 thực hiện mã lệnh này}

75: sp := sp + a;goto 0;

76: a := band (ir, smask); {mã lệnh là 1111111 = DESP (sp := sp -y). Dòng 76, 77, 78 thực hiện mã lệnh này}

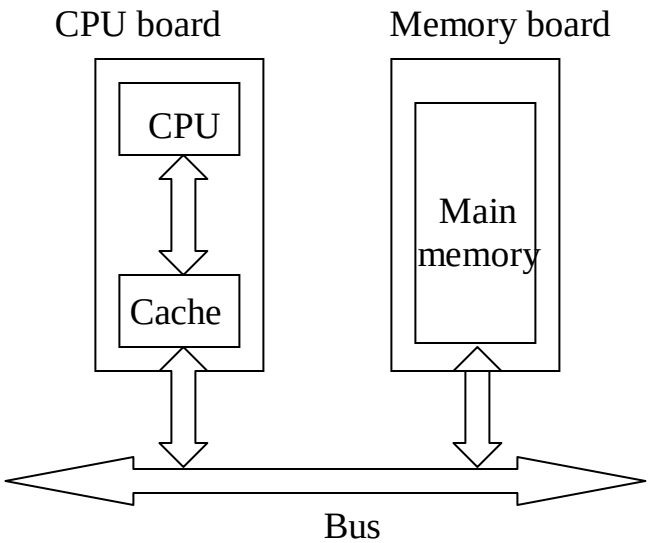
77: a := inv(a);

78: a := a + 1; goto 75;

**5. Bộ nhớ truy cập nhanh - bộ nhớ cache**

Trong lịch sử phát triển máy tính, CPU thường có tốc độ nhanh hơn bộ nhớ chính. Điều này dẫn đến sự bất cập khi máy tính hoạt động. Khi CPU yêu cầu bộ nhớ, CPU phải duy trì trạng thái nghỉ trong một thời gian để chờ bộ nhớ đáp ứng. Như chúng ta đã thấy, khi CPU thiết lập việc đọc bộ nhớ trong thời gian của một chu kỳ bus, và không nhận dữ liệu cho đến 2 hoặc 3 chu kỳ sau đó.

Để giải quyết vấn đề trên, người ta chọn giải pháp kết hợp một bộ nhớ nhỏ tốc độ nhanh với một bộ nhớ lớn tốc độ chậm để có một bộ nhớ tốc độ nhanh, dung lượng lớn và giá thành hạ. Bộ nhớ tốc độ nhanh, dung lượng nhỏ được gọi là cache được điều khiển bởi vi chương trình.

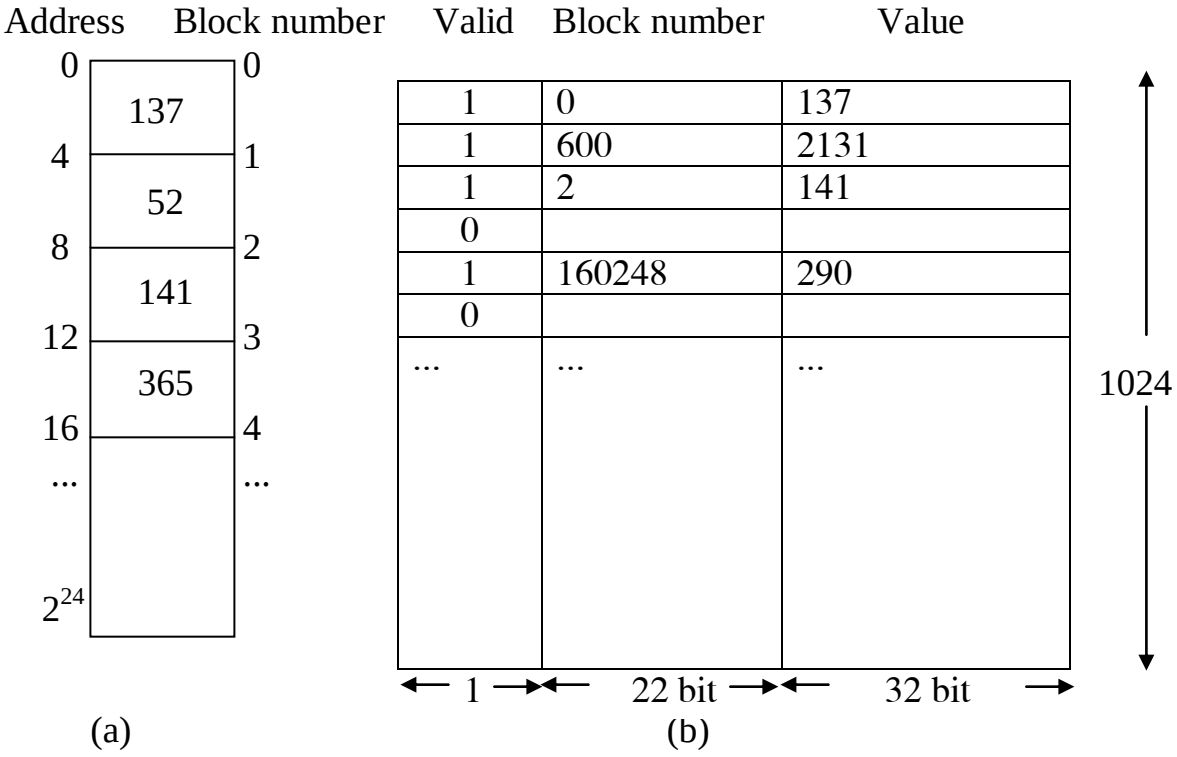



Hình 3.16. Bộ nhớ cache

Qua nghiên cứu, người ta nhận thấy rằng chương trình không truy cập bộ nhớ hoàn toàn ngẫu nhiên. Nếu địa chỉ nhớ được truy cập là A, thường địa chỉ nhớ

kế tiếp được truy cập sẽ là lân cận của A. Trong chương trình chính, nếu không có chỉ thị nhảy và chỉ thị gọi thủ tục, thì các chỉ thị luôn được tìm nạp từ những vị trí liên tiếp trong bộ nhớ. Mặt khác, hầu hết thời gian thực thi chương trình được sử dụng cho các vòng lặp, trong đó một số hữu hạn các chỉ thị được thực hiện nhiều lần.

Từ nhận xét trên, ý tưởng chung của việc sử dụng bộ nhớ cache là: khi một từ nhớ được truy cập từ bộ nhớ chính, từ này sẽ được đưa vào bộ nhớ cahe, để trong những lần truy cập tiếp theo, nó sẽ được truy cập với tốc độ nhanh hơn. Trong các máy tính hiện nay bộ nhớ cache được đặt trong CPU.




Hình 3.17. (a) Bộ nhớ chính với các khối nhớ 4 byte

(b) Cache có 1024 dòng

Hình 3.17 minh hoạ một bộ nhớ chính $2^{24}$ byte được chia thành $2^{22}$ khối ($2^{24}/2^2 = 2^{22}$) và một bộ nhớ cache có 1024 dòng, mỗi dòng 55 bit. Mỗi dòng (entry) có 3 trường: trường valid 1 bit cho biết dòng đó hiện tại có được sử dụng không, trường block number 22 bit chứa số của khối nhớ, trường value 32 bit chứa giá trị của khối.

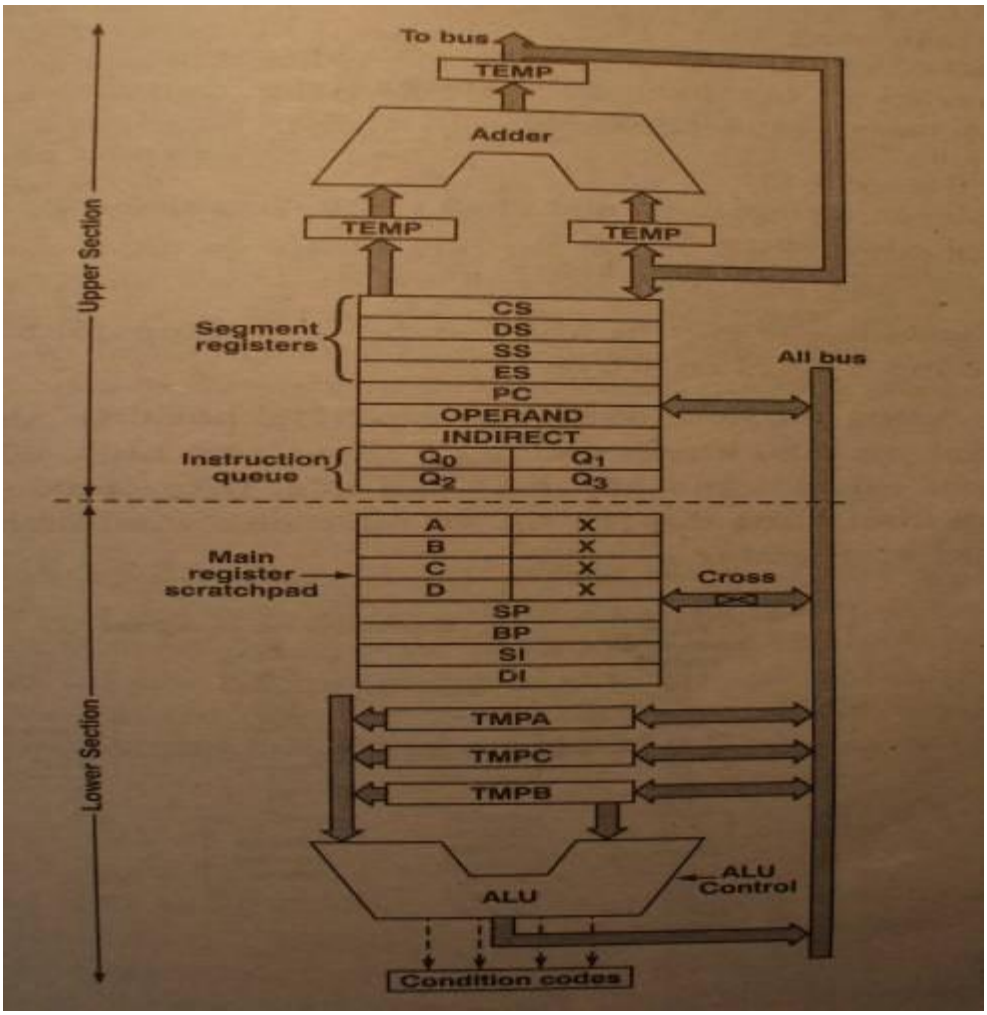
Khi cần truy cập một từ nhớ 32 bit, trước tiên vi chương trình kiểm tra tất cả các điểm nhập của cache để tìm ra điểm nhập có valid = 1 và trường block number

chứa địa chỉ của từ nhớ. Nếu không tìm thấy (cache miss), tìm nạp từ nhớ từ bộ nhớ chính, đồng thời đưa từ nhớ đó vào cache. Theo thời gian, các điểm nhập của cache được sử dụng tăng dần. Nếu chương trình cần phiên dịch sử dụng ít hơn 1024 từ nhớ thì toàn bộ chương trình và dữ liệu sẽ ở trong cache, chương trình sẽ được chạy với tốc độ nhanh nhất. Nếu chương trình cần phiên dịch cần nhiều hơn 1024 từ nhớ, tại một thời điểm nào đó, cache bị đầy và điểm nhập cũ sẽ bị loại bỏ để dành chỗ cho điểm nhập mới.

## 6. Vi cấu trúc của Intel

Vi cấu trúc của tất cả các CPU của Intel đều được phát triển từ CPU đầu tiên 8086/8088. Chúng ta sẽ xem xét vi cấu trúc của 8088 vì nó đơn giản nhất.

### 6.1. Đường dữ liệu




Hình 3.18. Đường dữ liệu của 8088

Đường dữ liệu của 8088 được minh hoạ trong hình 3.18. Nó bao gồm 2 phần: phần thấp và phần cao, 2 phần này hoạt động song song.

+ Phần thấp tương tự đường dữ liệu trong hầu hết các máy tính khác. Nó gồm 1 ALU, lấy dữ liệu ở 2 ngõ vào từ 3 thanh ghi 16 bit: TMPA, TMPB, TMPC, chúng được nạp trước chu kỳ của ALU. Ngõ ra đi vào bus, từ bus này có thể dẫn trở lại các thanh ghi. ALU có thể thực hiện các thao tác với dữ liệu 8 bit hay 16 bit. Các mã điều kiện sinh ra từ 1 thao tác của ALU có thể được chứa trong từ trạng thái chương trình (PSW) dưới sự điều khiển của vi chương trình. Phần thấp còn chứa một bộ nhớ đệm gồm 8 thanh ghi 16 bit AX, BX, CX, DX, SP, BP, SI, DI.

+ Phần trên của đường dữ liệu liên quan đến việc tính địa chỉ. Nó chứa 4 thanh ghi đoạn, một thanh ghi bộ đếm chương trình, 2 thanh ghi lưu trữ, 2 thanh ghi hàng đợi lệnh 16 bit. Phần trên còn chứa 1 bộ cộng để cộng 12 bit cao của địa chỉ offset với 12 bit thấp của địa chỉ segment. 4 bit thấp của offset đi thẳng tới 4 bit địa chỉ thấp của 20 đường địa chỉ. Kết quả của phép cộng đi vào 16 bit cao của bus địa chỉ.

- **Địa chỉ segment:offset của 8088**

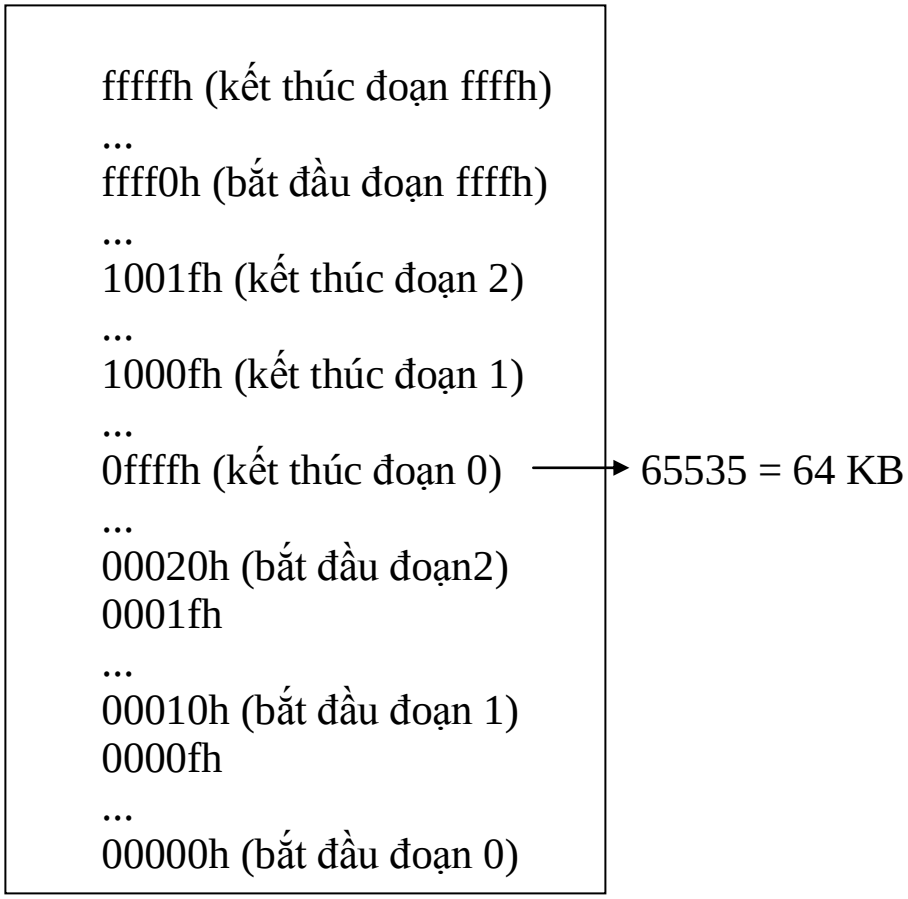
8088 có 20 đường địa chỉ, vì vậy nó có thể địa chỉ hoá được 1 MB bộ nhớ chính (RAM). Bắt đầu bằng địa chỉ 0 (0000 0000 0000 0000 0000) đến địa chỉ $2^{20}$ - 1 = 1048575 (1111 1111 1111 1111 1111), hay được viết dưới dạng hexa: 00000h - fffffh.

Tuy nhiên, 8088 chỉ có các thanh ghi 16 bit, chỉ có thể chứa được các giá trị: 0 - 65535. Vì vậy để lưu trữ được một địa chỉ 20 bit, 8088 phải dùng 2 thanh ghi. Một thanh ghi chứa địa chỉ đoạn (segment) và một thanh ghi chứa địa chỉ tương đối (offset). Một địa chỉ vật lí trong bộ nhớ được xác định bằng một địa chỉ đoạn và một địa chỉ tương đối và được viết segment:offset. Muốn có địa chỉ vật lý từ địa chỉ segment:offset ta làm như sau:

1. Nhân địa chỉ segment với 10h.

2. Cộng địa chỉ segment sau khi nhân với địa chỉ offset.

| Số TT của đoạn | Địa chỉ bắt đầu | | Địa chỉ kết thúc | |
|---|---|---|---|---|
| | Segment:offset | Vật lý | Segment:offset | Vật lý |
| 0 | 0000h:0000h | 00000h | 0000h:ffffh | 0ffffh |
| 1 | 0001h:0000h | 00010h | 0001h:ffffh | 1000fh |
| 2 | 0002h:0000h | 00020h | 0002h:ffffh | 1001fh |
| ... | ... | ... | ... | ... |
| ffffh | ffffh:0000h | ffff0h | ffffh:ffffh | fffffh |

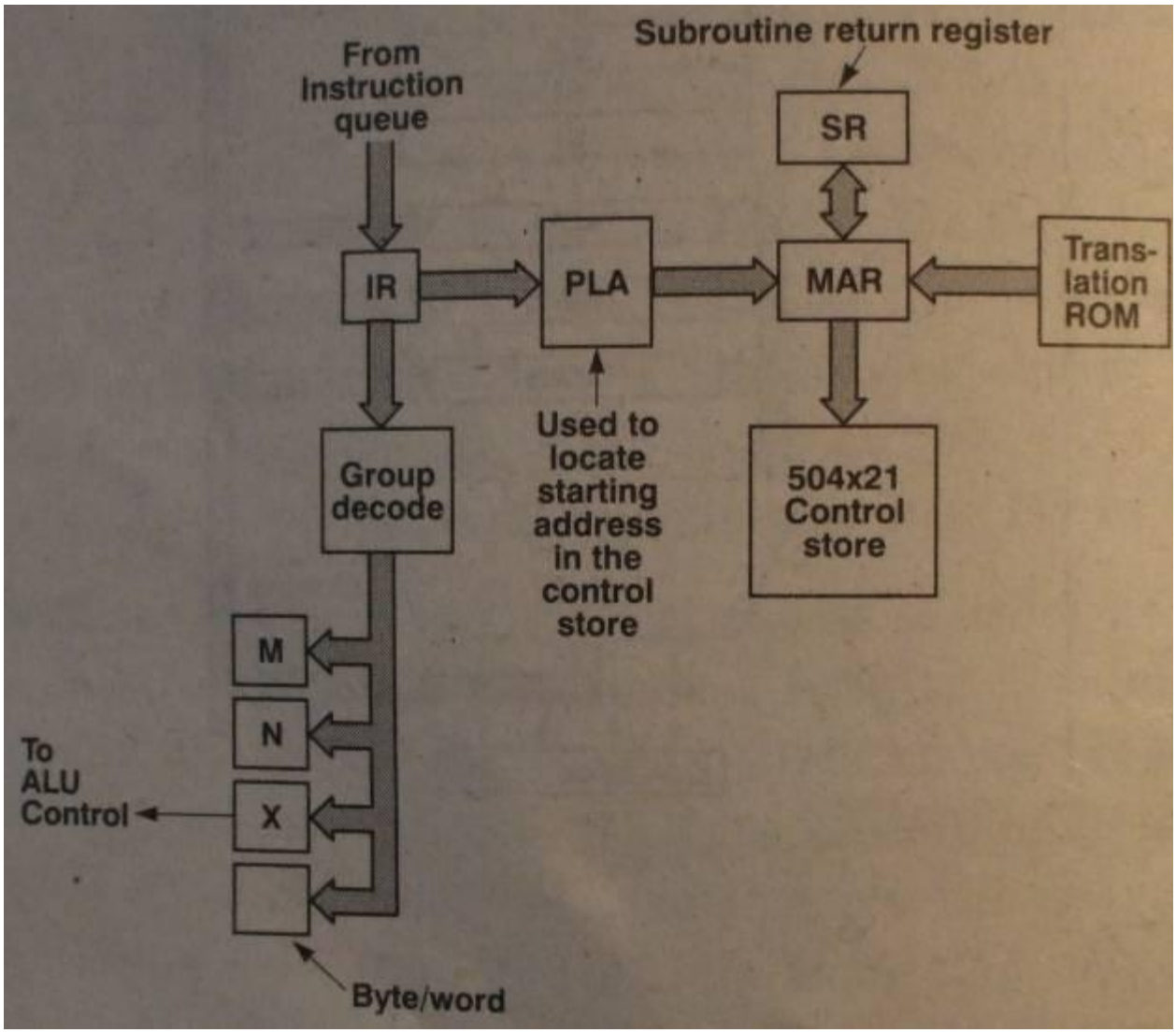



Hình 3.19. Địa chỉ segment:offset của 8088

+ 8088 được thiết kế để xử lí chỉ thị theo kiểu đường ống. Có 4 đơn vị chức năng để thực hiện: tìm nạp chỉ thị, giải mã chỉ thị, thực hiện tính toán địa chỉ, thực hiện chỉ thị. Bộ tìm nạp hoàn toàn độc lập so với 3 đơn vị còn lại. Bất kì khi nào bus rỗi, bộ tìm nạp phát một yêu cầu bộ nhớ để đọc byte kế tiếp trong luồng chỉ thị vào hàng đợi chỉ thị.

Kích thước của hàng đợi cũng cần phải được cân nhắc kỹ. Nếu nhỏ CPU sẽ thường xuyên phải chờ, nhưng nếu quá lớn cũng không tốt. Bởi vì, bộ tìm nạp không biết các byte tìm nạp có nghĩa gì, bộ này tìm nạp byte kế tiếp trong luồng chỉ thị khi hàng đợi còn trống. Nếu như sau chỉ thị nhẩy xa, bộ nạp sẽ nạp vào hàng đợi các byte không được sử dụng, điều này gây lãng phí. 8088 có một hàng đợi 4 byte còn 8086 có hàng đợi 6 byte.

**6.2. Khối điều khiển**




Hình 3.20. Khối điều khiển của 8088

Khối điều khiển chứa vi chương trình và điều khiển đường dữ liệu.

+ Khi bắt đầu 1 chỉ thị mới của cấp máy qui ước (cấp 2), byte opcode của chỉ thị được nạp từ hàng đợi vào thanh ghi chỉ thị IR.

+ Đơn vị phần cứng giải mã nhóm (Group decode) giải mã opcode và đưa tới các thành phần khác. Các thanh ghi M, N tiếp nhận các thông tin dùng cho việc tính địa chỉ của toán hạng nguồn và toán hạng đích. Thanh ghi X chứa thông tin dùng để báo cho ALU biết thực hiện chức năng nào. Bit byte/word điều khiển các thao tác của ALU 8 bit/16 bit và truyền thông tin với bộ nhớ đệm 8 bit/16 bit.

+ Các vi lệnh có độ rộng 21 bit. Có 504 vi lệnh chứa trong ROM (bộ nhớ điều khiển) có dung lượng 504 x 21 bit. Khi thực thi 1 chỉ thị máy cấp 2, PLA

chuyển đổi opcode thành địa chỉ bắt đầu của vi lệnh trong bộ nhớ điều khiển thực hiện chỉ thị đó.

+ Vi chương trình được chia thành các khối (burst), mỗi khối điều khiển một hoặc một số chỉ thị máy cấp 2, hoặc cung cấp 1 thủ tục như là tính địa chỉ chẳng hạn.

Có hai loại vi lệnh nhẩy: nhẩy ngắn thực hiện nhẩy trong khối, nhẩy dài nhảy tới bất cứ nơi nào trong vi chương trình. Các vi lệnh gọi vi thủ tục hoạt động tương tự như vi lệnh nhảy dài, chỉ khác là chúng gửi địa chỉ trở về vào thanh ghi SR (Sub-routine Register). Các vi thủ tục không được lồng nhau. ROM 504 từ chứa khoảng 90 burst (mỗi burst 5 / 6 từ).

## Tóm tắt chương

Ở cấp máy vi chương trình (cấp máy 1), CPU có 2 thành phần chính:

1. Đường dữ liệu: cơ bản bao gồm một bộ nhớ đệm, ALU và bộ dịch bit. Một chu kỳ dữ liệu bao gồm:

+ Lấy các toán hạng từ bộ nhớ đệm.

+ Đưa vào ALU / mạch dịch bit.

+ Cất kết quả trở lại bộ nhớ đệm / thanh ghi đệm bộ nhớ.

2. Phần điều khiển: chứa bộ nhớ điều khiển (lưu trữ vi chương trình) và điều khiển sự hoạt động của đường dữ liệu.