TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

———————— * ————————

Báo cáo môn học:

Vi xử lý

Cài đặt thuật toán MD5 trên vi xử lý 8086

Sinh viên thực hiện:
Nguyễn Quang Quý - 20153108

Giáo viên hướng dẫn:
TS Ngô Lam Trung

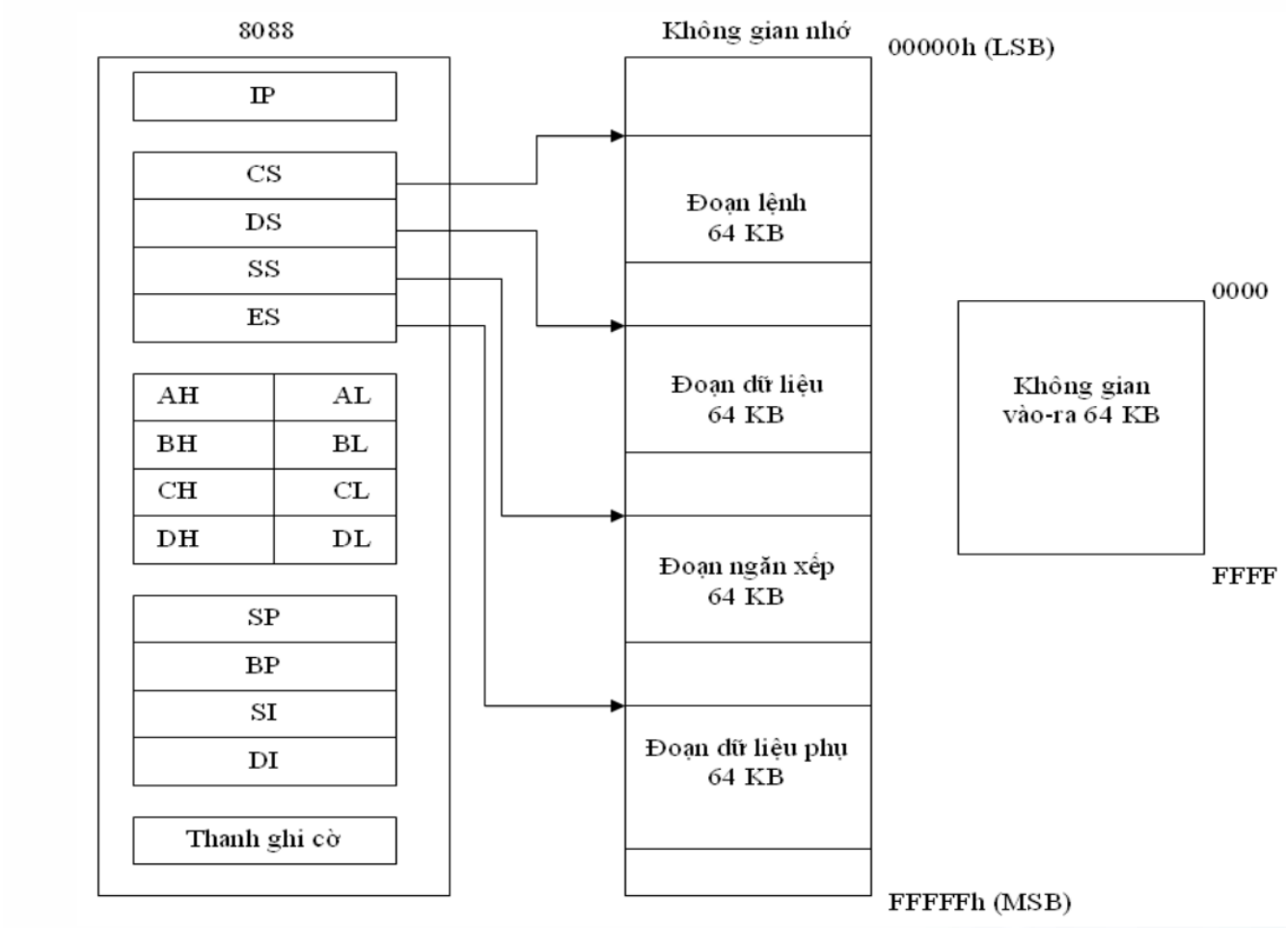
Hà Nội, Ngày 13 tháng 6 năm 2018

Mục lục

1 Tổng quan vi xử lý 8086

Intel 8086 là bộ vi xử lý 16 bít đầu tiên của Intel và là vi xử lý đầu tiên hỗ trợ tập lệnh x86. Vi xử lý được sử dụng trong nhiều lĩnh vực khác nhau, nhất là trong các máy IBM PC/XT. Các bộ vi xử lý thuộc họ này vẫn được sử dụng rộng rãi trong một thời gian dài do tính kế thừa của các sản phẩm trong họ x86. Các chương trình viết cho 8086 vẫn có thể chạy trên các hệ thống tiên tiến sau này.

1.1 Tập thanh ghi



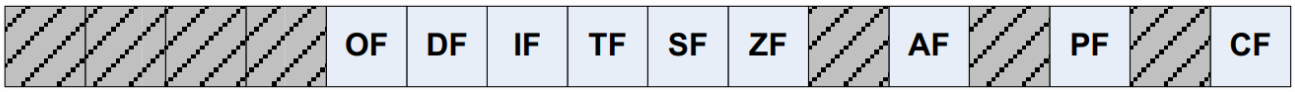
Hình 1: Tập thanh ghi, không gian nhớ và không gian vào ra trong 8086

Tập thanh ghi trong 8086 bao gồm:

- 4 thanh ghi đoạn:
  - CS (Code Segment): thanh ghi đoạn lệnh
  - DS (Data Segment): thanh ghi đoạn dữ liệu
  - SS (Stack Segment): thanh ghi đoạn ngăn xếp

  - ES (Extra Segment): thanh ghi đoạn dữ liệu phụ

- 3 thanh ghi con trỏ:
  - IP: con trỏ lệnh (Instruction Pointer). IP luôn trỏ vào lệnh tiếp theo sẽ được thực hiện nằm trong đoạn mã CS. Địa chỉ đầy đủ của lệnh tiếp theo có dạng CS:IP
  - SP: con trỏ ngăn xếp (Stack Pointer). SP luôn trỏ vào đỉnh hiện thời của ngăn xếp nằm trong đoạn ngăn xếp SS. Địa chỉ đỉnh ngăn xếp có dạng SS:SP
  - BP: con trỏ cơ sở (Base Pointer). BP luôn trỏ vào một dữ liệu nằm trong đoạn ngăn xếp SS. Địa chỉ đầy đủ của một phần tử trong đoạn ngăn xếp có dạng SS:BP
- 2 thanh ghi chỉ số:
  - SI: chỉ số gốc hay nguồn (Source Index). SI chỉ vào dữ liệu trong đoạn dữ liệu DS mà địa chỉ cụ thể đầy đủ có dạng DS:SI
  - DI: chỉ số đích (Destination Index). DI chỉ vào dữ liệu trong đoạn dữ liệu DS (hoặc ES) mà địa chỉ cụ thể đầy đủ có dạng DS:DI (hoặc ES:DI)
- 4 thanh ghi dữ liệu:
  - AX (Accumulator): thanh ghi tích lũy. Các kết quả của các thao tác thường được chứa ở AX (kết quả của phép nhân, chia).
  - BX (Base): thanh ghi cơ sở, thường dùng để chứa địa chỉ cơ sở của một dãy các ô nhớ.
  - CX (Count): thanh đếm. CX thường được dùng để chứa số lần lặp trong trường hợp các lệnh LOOP (lặp).
  - DX (Data): thanh ghi dữ liệu. DX tham gia các thao tác của phép nhân hoặc chia các số 16 bít. DX thường dùng để chứa địa chỉ của các cổng trong các lệnh vào/ ra dữ liệu.
- thanh ghi cờ:



Hình 2: Cấu trúc thanh ghi cờ

  - C hoặc CF (Carry Flag): cờ nhớ. CF = 1 khi có nhớ hoặc mượn từ bít có nghĩa lớn nhất MSB (Most Significant Bit).
  - P hoặc PF (Parity Flag): cờ chẵn lẻ. PF phản ánh tính chẵn lẻ của tổng số bít 1 có trong kết quả. Cờ PF =1 khi tổng số bít 1 trong kết quả là lẻ (odd parity) và PF =0 khi tổng số bít 1 trong kết quả là chẵn (even parity).
  - A hoặc AF (Auxiliary Carry Flag): cờ nhớ phụ rất có ý nghĩa khi ta làm việc với các số BCD (Binary Coded Decimal). AF = 1 khi có nhớ hoặc mượn từ một số BCD thấp (4 bít thấp) sang một số BCD cao (4 bít cao).
  - Z hoặc ZF (Zero Flag): cờ rỗng. ZF =1 khi kết quả = 0 và ZF = 0 khi kết quả $\neq 0$.

- S hoặc SF (sign flag): cờ dấu. SF = 1 khi kết quả âm và SF = 0 khi kết quả không âm
- O hoặc OF (Overflow Flag): cờ tràn. OF = 1 khi kết quả là một số bù 2 vượt qua ngoài giới hạn biểu diễn dành cho nó.
- T hoặc TF (Trap Flag): cờ bẫy. TF = 1 thì CPU làm việc ở chế độ chạy từng lệnh (chế độ này dùng khi cần tìm lỗi trong một chương trình).
- I hoặc IF (Interrupt Enable Flag): cờ cho phép ngắt. IF = 1 thì CPU cho phép các yêu cầu ngắt (che được) và IF = 0 thì CPU cấm các yêu cầu ngắt.
- D hoặc DF (Direction Flag): cờ hướng. DF = 1 khi CPU làm việc với chuỗi ký tự theo thứ tự từ phải sang trái, hoặc giảm địa chỉ (vì vậy D chính là cờ lùi) và DF = 0 khi CPU làm việc với chuỗi ký tự theo thứ tự từ trái sang phải, hoặc tăng địa chỉ .

1.2 Phân đoạn bộ nhớ của 8086

Do vi xử lý 8086 sử dụng 20 bít địa chỉ để quản lý bộ nhớ trong, như vậy nó có khả năng phân biệt ra được 220 = 1.048.576 = 1M ô nhớ hay 1Mbyte, vì các bộ nhớ thường tổchức theo byte. Vi xử lí chia không gian 1Mbyte bộ nhớ thành các vùng khác nhau theo nội dung mà chúng lưu trữ, gồm các vùng nhớ để:

- Chứa mã chương trình.
- Chứa dữ liệu và kết quả của chương trình.
- Tạo một vùng nhớ đặc biệt gọi là ngăn xếp (stack) dùng vào việc quản lý các thông số của bộ vi xử lý khi gọi thự hiện các chương trình con hoặc trở về từ chương trình con.

Để xác định địa chỉ vật lý 20 bít của một ô nhớ nào đó trong một đoạn bất kỳ. CPU 8086 phải dùng đến 2 thanh ghi 16 bít: một thanh ghi để chứa địa chỉ cơ sở, còn thanh kia chứa độ lệch. Từ nội dung của cặp thanh ghi đó tạo ra địa chỉ vật lý theo công thức sau:

Địa chỉ vật lý = Thanh ghi đoạn x 10h (16 hệ 10) + Thanh ghi lệch

Việc dùng 2 thanh ghi để ghi nhớ thông tin về địa chỉ thực chất để tạo ra một loại địa chỉ gọi là địa chỉ logic và được ký hiệu như sau:

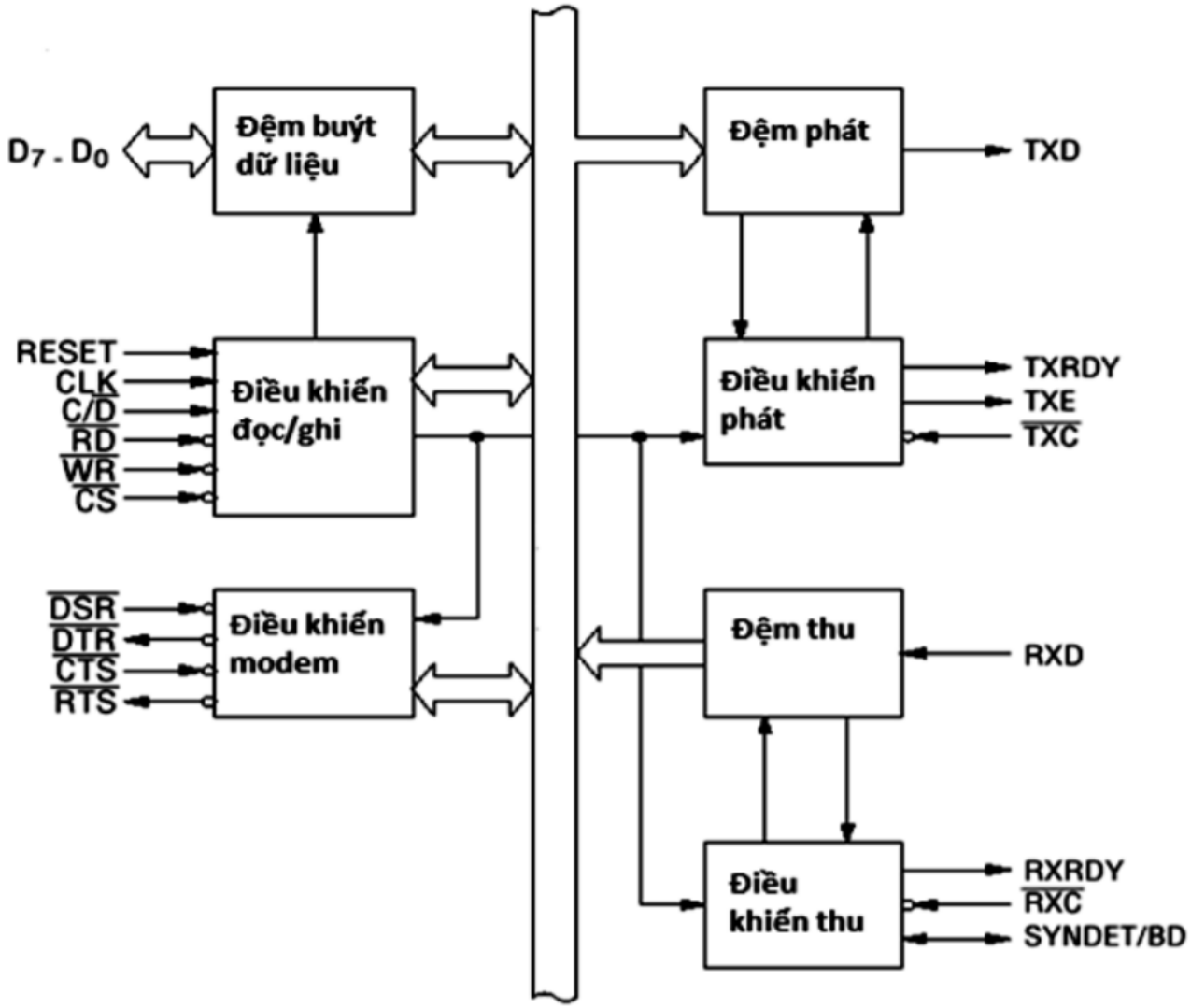
Thanh ghi đoạn: Thanh ghi lệch hay segment: offset

Địa chỉ kiểu **segment: offset** là logic vì nó tồn tại dưới dạng giá trị của các thanh ghi cụ thể bên trông CPU và khi cần thiết truy cập ô nhớ nào đó thì nó phải được đổi ra địa chỉ vật lý sau đó được đưa lên bus địa chỉ
Ví dụ: cặp CS:IP sẽ chỉ ra địa chỉ của lệnh sắp thực hiện trong đoạn mã. Tại một thời điểm nào đó ta có CS = F000H và IP = FFF0H thì

Địa chỉ vật lý = F000h x10h + FFF0h = F0000h + FFF0h = FFFF0h

2 Giao tiếp với 8251

2.1 Sơ đồ khối và tín hiệu



Hình 3: Sơ đồ khối 8251A

Các tín hiệu của mạch 8251A hầu hết là giống tín hiệu của 8086. Chân chọn vỏ của 8251A phải được nối với đầu ra của một mạch giải mã địa chỉ để đặt mạch 8251A vào một địa chỉ cơ bản nào đó.

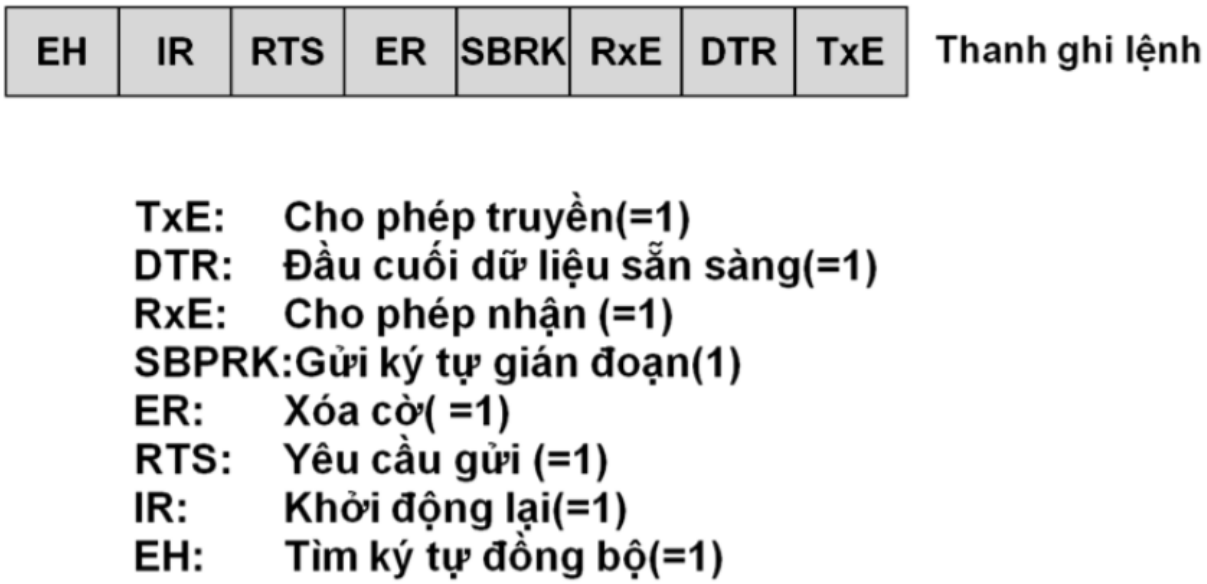
- CLK [I]: Chân nối đến xung đồng hồ của hệ thống.
- TxRDY [0]: Tín hiệu báo đệm giữ rỗng (sẵn sàng nhận ký tự mới từ CPU).
- RxRDY [0]: Tín hiệu báo đệm thu đầy (có ký hiệu nằm chờ CPU đọc vào).
- TxEMPTY [0]: Tín hiệu báo cả đệm thu và đệm phát đều rỗng.

- C/D [I]: CPU thao tác với thanh ghi lệnh/thanh ghi dữ liệu của 8251A, khi C/D=1 thì thanh ghi lệnh được chọn làm việc. Chân này thường được nối với A0 của bus địa chỉ để cùng với các tín hiệu WR và RD chọn ra 4 thanh ghi bên trong 8251A.
- RxC [I] và TxC [I]: Xung đồng hồ cung cấp cho các thanh ghi dịch của phần thu và phần phát. Thường 2 thanh này nối chung để phần thu và phần phát làm việc với cùng tầng số nhịp. Tần số của các khung đồng hồ đưa đến chân RxC và TxC được chọn sao cho là bội số (cụ thể là gấp 1, 16 hoặc 64) của tốc độ thu hay tốc độ phát theo yêu cầu.
- DSR, DTR: là cặp tín hiệu yêu cầu thiết bị modem sẵn sàng và trả lời của modem với tín hiệu yêu cầu.
- RTS, CTS: là cặp tín hiệu yêu cầu modem sẵn sàng phát và đáp ứng của modem với tín hiệu yêu cầu.
- SYNDET/BRKDET [O]: khi 8251A làm việc ở chế độ không đồng bộ, nếu RxD = 0 kéo dài hơn thời gian của 2 ký tự thì chân này có mức cao để báo là việc truyền hoặc đường truyền bị gián đoạn. Khi 8251A làm việc ở chế độ đồng bộ, nếu phần thu tìm thấy ký tự đồng bộ rong bản tin thu được thì chân này có mức cao.

2.2 Các thanh ghi bên trong

Thanh ghi lệnh

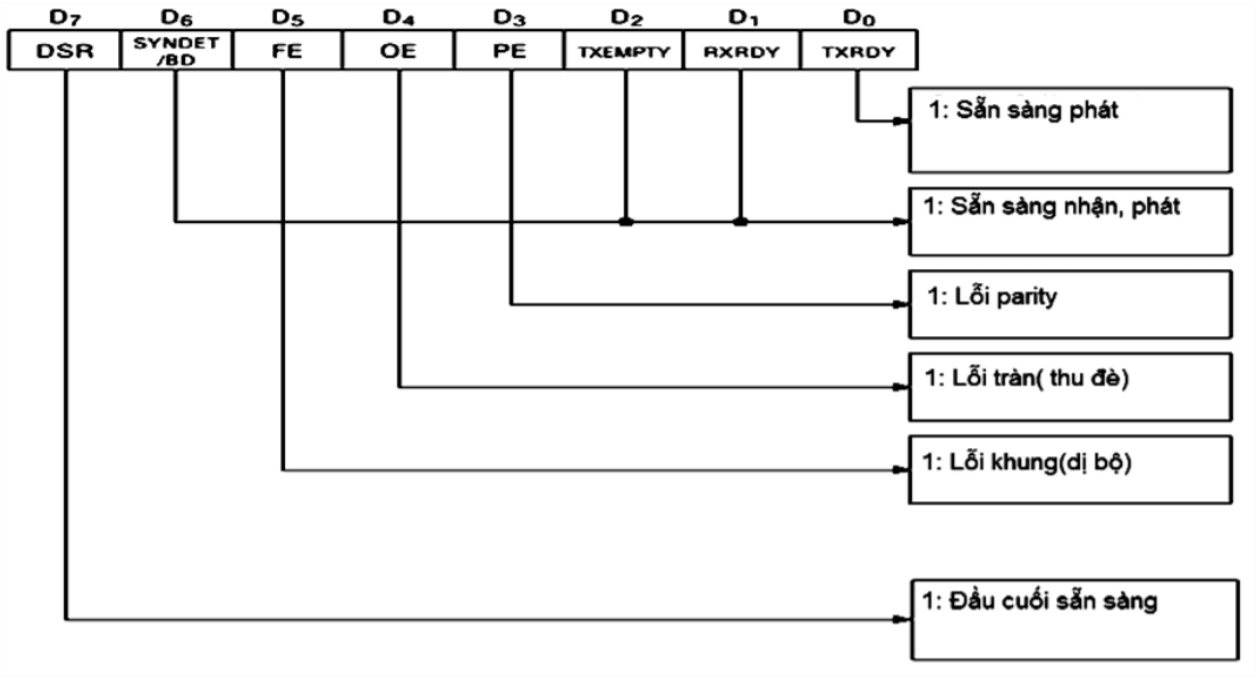
Cấu trúc thanh ghi lệnh như sau:



Hình 4: Cấu trúc thanh ghi lệnh

Thanh ghi trạng thái

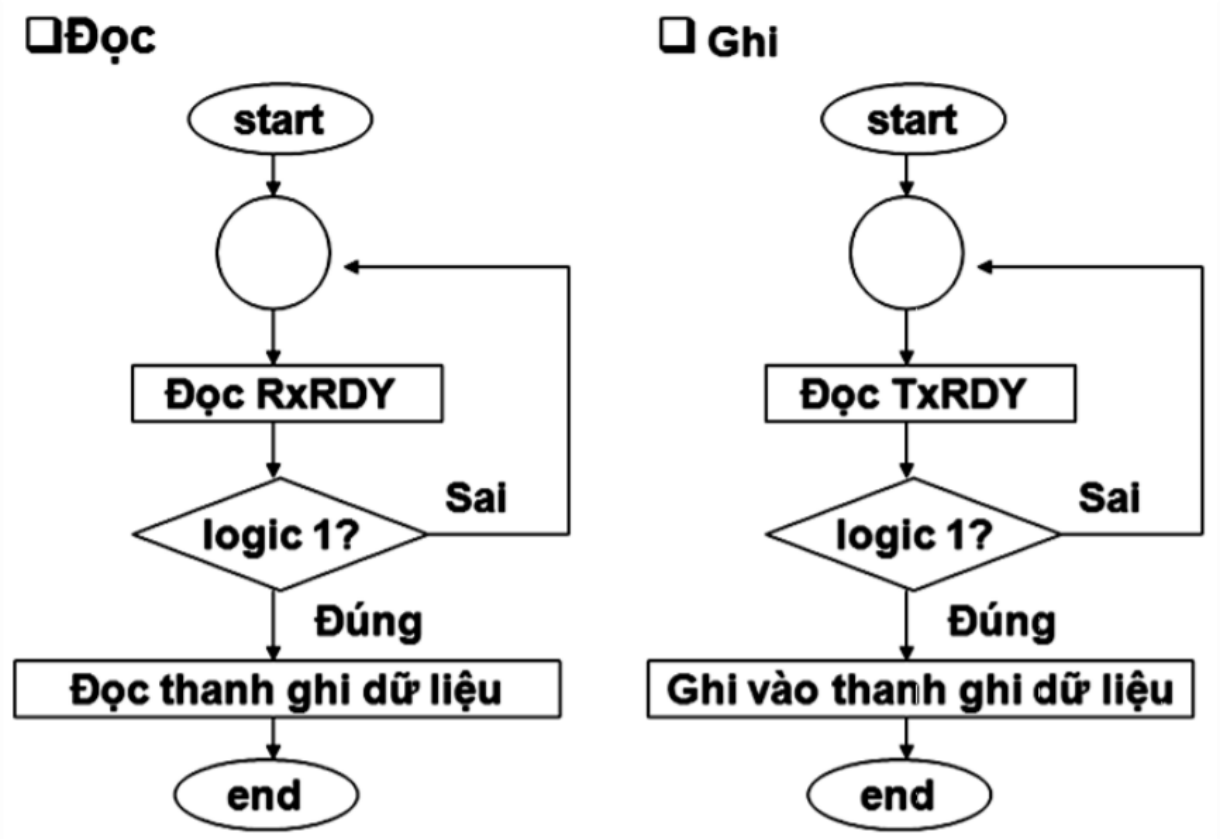
Giá trị trên các bít thanh ghi này cho ta biết tình trạng hoạt động của 8251A.



Hình 5: Cấu trúc thanh ghi trạng thái

2.3 Lập trình 8251A

Để lập trình cho 8251A trước tiên ta cần xác lập chế độ hoạt động bằng cách tính giá trị của thanh ghi chế độ và gửi ra cổng điều khiển. Để gửi hoặc nhận dữ liệu ta cần liên tục kiểm tra trạng thái của 8251A theo lưu đồ đọc/ghi đơn giản sau:



Hình 6: Lưu đồ đọc ghi đơn giản

3 Thuật toán MD5

Trong mật mã học, MD5 (viết tắt của tiếng Anh Message-Digest algorithm 5, giải thuật Tiêu hóa tin 5) là một hàm băm mật mã học được sử dụng phổ biến với giá trị Hash dài 128-bit. Là một chuẩn Internet (RFC 1321), MD5 đã được dùng trong nhiều ứng dụng bảo mật, và cũng được dùng phổ biến để kiểm tra tính toàn vẹn của tập tin. Một bảng băm MD5 thường được diễn tả bằng một số hệ thập lục phân 32 ký tự. MD5 được thiết kế bởi Ronald Rivest vào năm 1991 để thay thế cho hàm băm trước đó, MD4. Vào năm 1996, người ta phát hiện ra một lỗ hổng trong MD5; trong khi vẫn chưa biết nó có phải là lỗi nghiêm trọng hay không, những chuyên gia mã hóa bắt đầu đề nghị sử dụng những giải thuật khác, như SHA-1 (khi đó cũng bị xem là không an toàn). Trong năm 2004, nhiều lỗ hổng hơn bị khám phá khiến cho việc sử dụng giải thuật này cho mục đích bảo mật đang bị đặt nghi vấn.

Yêu cầu của thuật toán mã hõa này được mô tả đơn giản như sau:

- **Input:** thông điệp với độ dài bất kỳ (Ở đây đặt độ dài thông điệp tối đa là 20 bit)
- **Output:** giá trị băm (message digest) 128 bits

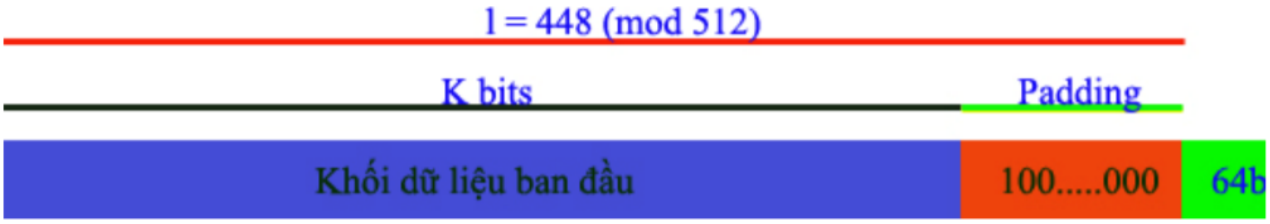
3.1 Giải thuật

Giải thuật gồm 5 bước thao tác trên khối 512 bits.

Tạo block

MD5 chuyển một đoạn thông tin chiều dài thay đổi thành một kết quả chiều dài không đổi 128 bit. Mẩu tin đầu vào được chia thành từng đoạn 512 bit; mẩu tin sau đó được độn sao cho chiều dài của nó chia chẵn cho 512. Công việc độn vào như sau: đầu tiên một bit đơn, 1, được gắn vào cuối mẩu tin. Tiếp theo là một dãy các số zero sao cho chiều dài của mẩu tin lên tới 64 bit ít hơn so với bội số của 512. Những bit còn lại được lấp đầy bằng một số nguyên 64-bit đại diện cho chiều dài của mẩu tin gốc với chú ý:

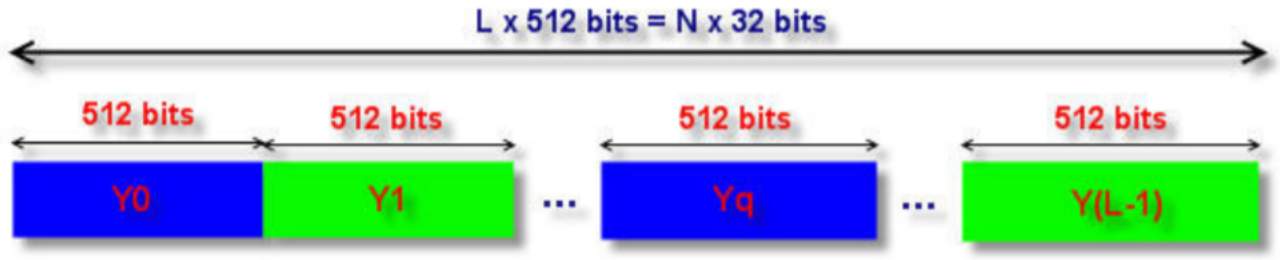
- Độ dài của khối dữ liệu ban đầu được biểu diễn dưới dạng nhị phân 64-bit và được thêm vào 64 bit cuối của block theo thứ tự từ byte thấp tới byte cao.
- Nếu độ dài của khối dữ liệu ban đầu $> 2^{64}$, chỉ 64 bits thấp được sử dụng, nghĩa là giá trị được thêm vào bằng K mod 2^{64}.



Hình 7: Tạo block 512 bit cho dữ liệu nhập vào

Kết quả ta thu được khối dữ liệu có độ dài là bội của 512. Khối dữ liệu được biểu diễn:

- Bằng một dãy L khối 512-bit $Y_0, Y_1, \ldots, Y_{L-1}$
- Bằng một dãy N từ (word) 32-bit M_0, M_1, M_{N-1}. Vậy N = L x 16



Hình 8: Khối 512 sau khi tạo xong

Khởi tạo bộ đệm MD

Một bộ đệm 128-bit được dùng lưu trữ các giá trị băm trung gian và kết quả. Bộ đệm được biểu diễn bằng 4 thanh ghi 32- bit (A, B, C, D) với các giá trị khởi tạo ở dạng little-endian (byte có trọng số nhỏ nhất trong từ nằm ở địa chỉ thấp nhất) như sau:

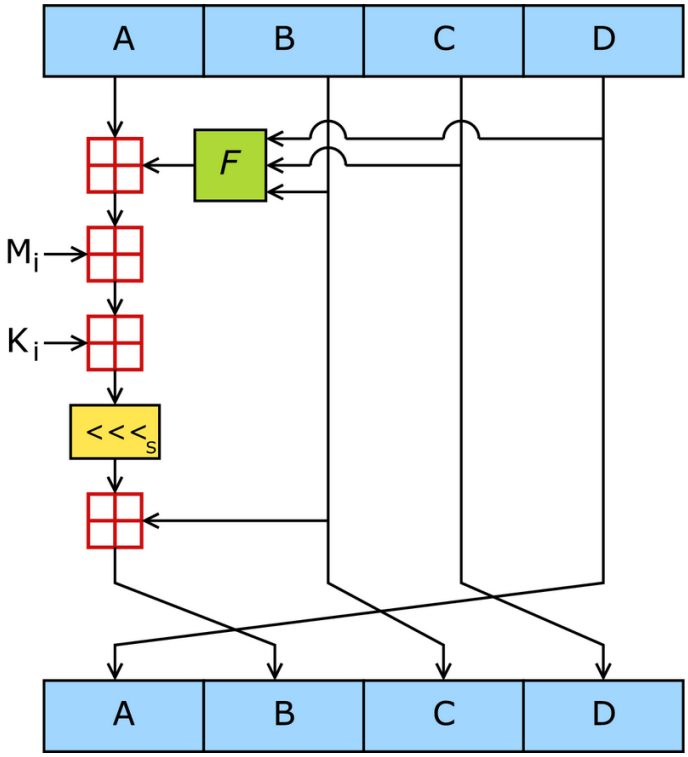
- A_0 = 0x67452301
- B_0 = 0xefcdab89
- C_0 = 0x98badcfe
- D_0 = 0x10325476

Xử lý các khối dữ liệu 512 bit

Trọng tâm của giải thuật đó là hàm xử lý các khối tin 512-bit (hàm nén), mỗi khối xác định một trạng thái. Quá trình xử lý khối tin bao gồm bốn giai đoạn giống nhau, gọi là vòng; mỗi vòng gồm có 16 tác vụ giống nhau dựa trên hàm phi tuyến F, cộng mô đun, và dịch trái. Hình vẽ mô tả một tác vụ trong một vòng. Có 4 khả năng cho hàm F; mỗi cái được dùng khác nhau cho mỗi vòng:

$$
\begin{aligned}
F(B,C,D) &= (B \wedge C) \vee (\neg B \wedge D) \\
G(B,C,D) &= (B \wedge D) \vee (C \wedge \neg D) \\
H(B,C,D) &= B \oplus C \oplus D \\
I(B,C,D) &= C \oplus (B \vee \neg D)
\end{aligned}
$$

Hình 9: Các hàm luận lý trong mỗi vòng



Hình 10: Các thao tác được thực hiện trong một vòng

3.2 Mã giả

Thuật toán MD5 được thực hiện theo thuật toán sau, tất cả các giá trị đều ở dạng little-edian:

```
//Note: All variables are unsigned 32 bit and wrap modulo 2^32 when calculating
var int[64] s, K
var int i

//s specifies the per-round shift amounts
s[ 0..15] := { 7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22 }
s[16..31] := { 5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20 }
s[32..47] := { 4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23 }
s[48..63] := { 6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21 }

//Use binary integer part of the sines of integers (Radians) as constants:
for i from 0 to 63
    K[i] := floor(2^32 × abs(sin(i + 1)))
end for
//(Or just use the following precomputed table):
K[ 0.. 3] := { 0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee }
K[ 4.. 7] := { 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 }
K[ 8..11] := { 0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be }
K[12..15] := { 0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 }
K[16..19] := { 0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa }
K[20..23] := { 0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 }
K[24..27] := { 0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed }
K[28..31] := { 0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a }
K[32..35] := { 0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c }
K[36..39] := { 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 }
K[40..43] := { 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 }
K[44..47] := { 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 }
K[48..51] := { 0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 }
K[52..55] := { 0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 }
K[56..59] := { 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 }
K[60..63] := { 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 }
```

```
//Initialize variables:
var int a0 := 0x67452301   //A
var int b0 := 0xefcdab89   //B
var int c0 := 0x98badcfe   //C
var int d0 := 0x10325476   //D

//Pre-processing: adding a single 1 bit
append "1" bit to message
// Notice: the input bytes are considered as bits strings,
//  where the first bit is the most significant bit of the byte.[48]

//Pre-processing: padding with zeros
append "0" bit until message length in bits ≡ 448 (mod 512)
append original length in bits mod 2^64 to message

//Process the message in successive 512-bit chunks:
for each 512-bit chunk of padded message
    break chunk into sixteen 32-bit words M[j], 0 ≤ j ≤ 15
//Initialize hash value for this chunk:
    var int A := a0
    var int B := b0
    var int C := c0
    var int D := d0
```

```
//Main loop:
    for i from 0 to 63
        var int F, g
        if 0 ≤ i ≤ 15 then
            F := (B and C) or ((not B) and D)
            g := i
        else if 16 ≤ i ≤ 31 then
            F := (D and B) or ((not D) and C)
            g := (5×i + 1) mod 16
        else if 32 ≤ i ≤ 47 then
            F := B xor C xor D
            g := (3×i + 5) mod 16
        else if 48 ≤ i ≤ 63 then
            F := C xor (B or (not D))
            g := (7×i) mod 16
//Be wary of the below definitions of a,b,c,d
        F := F + A + K[i] + M[g]
        A := D
        D := C
        C := B
        B := B + leftrotate(F, s[i])
    end for
//Add this chunk's hash to result so far:
    a0 := a0 + A
    b0 := b0 + B
    c0 := c0 + C
    d0 := d0 + D
end for

var char digest[16] := a0 append b0 append c0 append d0 //(Output is in little-endian)
```

Một chút cải tiến cho thuật toán đó là thay thế hàm F ở hai vòng đầu:

- (0 <= i <= 15): F:= D **xor** (B **and** (C **xor** D))
- (16 <= i <= 31): F:= C **xor** (D **and** (B **xor** C))

4 Các thuật toán phụ trợ thao tác với số 32 bit

4.1 Cộng hai số 32 bit

Với 8086, các thanh ghi đều là 16 bit, do đó sẽ không có một phép toán nào hỗ trợ trực tiếp cho chúng ta phép cộng đối với số 32 bit. Để cộng 2 số 32 bit với nhau ta sẽ sử dụng 2 thanh ghi AX, DX và một biến *kq* (4 byte) để lưu kết quả ra bộ nhớ. Thủ tục cộng 32 bit được thực hiện như sau:

1. Gán 16-bit thấp của mỗi số hạng vào thanh ghi AX và DX
2. Thực hiện cộng hai thanh ghi AX và DX theo lệnh $add\mathbf{AX}, \mathbf{DX} \rightarrow$ kết quả sẽ lưu vào AX

3. Đưa giá trị của AX ra byte 16-bit của biến kq
4. Gán byte cao của mỗi số hạng vào thanh ghi AX và DX
5. Thực hiện cộng hai thanh ghi AX và DX theo lệnh $adc\mathbf{AX}, \mathbf{DX} \rightarrow$ kết quả sẽ lưu vào AX
6. Đưa giá trị của AX ra 16-bit cao của biến kq

4.2 Quay trái

Đối với lệnh quay trái, ta thực hiện như sau:

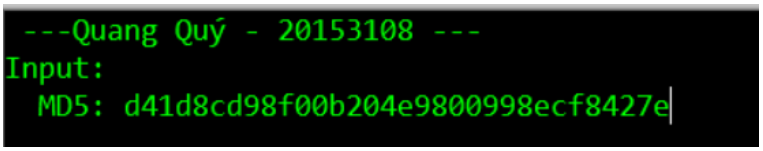
1. Gán 16-bit thấp, 16-bit cao của số 32 bit vào AX, DX
2. Thực hiện dịch trái 1 bit đối với AX và DX. Lưu lại bit được dịch lần lượt vào BH và BL
3. Tiến hành cộng AL với BL và DL với BH ta sẽ thu được kết quả của phép xoay trái 1 bit của số 32 bit với 16 bit cao ở DX và 16 bit thấp ở AX

Khi thực hiện quay n bit thì ta chỉ cần lặp lại thủ tục trên n lần.

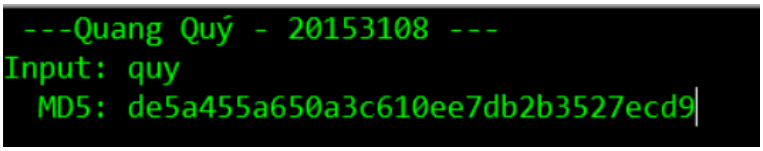
5 Kết quả

Thuật toán được cài đặt đúng đắn trên vi xử lý 8086. Sau đây là một số kết quả thu được (Các kết quả được so sánh với kết quả tại):

- MD5("") = d41d8cd98f00b204e9800998ecf8427e



- MD5("quy") = de5a455a650a3c610ee7db2b3527ecd9



Tài liệu tham khảo

[1] NGÔ LAM TRUNG *Slide bài giảng kĩ thuật vi xử lý.*

[2] Thuật toán MD5 - https://en.wikipedia.org/wiki/MD5

[3] PHẠM HOÀNG DUY & HOÀNG XUÂN DẬU *Slide bài giảng kĩ thuật vi xử lý (PTIT).*