TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

_____ * _____

BÀI TẬP LỚN

Môn: Vi xử lý

ĐỀ TÀI: Tính MD5 sử dụng vi xử lý 8086 và ngôn ngữ lập trình assembly

Sinh viên : **Phạm Minh Tâm**

MSSV : **20153298**

Lớp : **KSTN-CNTT-K60**

Giáo viên hướng dẫn: **TS. Ngô Lam Trung**

Hà Nội, ngày 12 tháng 6 năm 2018

Mục lục

Lời nói đầu

Trong thời buổi hiện nay, khoa học kĩ thuật và công nghiệp máy tính đang phát triển mạnh mẽ và đã trở thành một phần quan trọng không thể thiếu trong bất cứ một ngành nghề lĩnh vực cuộc sống nào. Các thiết bị công nghệ xuất hiện ngày một nhiều hơn, chứa bên trong nó những vi mạnh và vi xử lý phức tạp, hiện đại. Đi kèm với sự phức tạp của phần cứng, lập trình phần mềm cho các thiết bị vi xử lý này cũng trở lên phức tạp hơn rất nhiều. Vì thế, muốn làm việc tốt với phần cứng của thiết bị công nghệ nào cũng đòi hỏi cần có sự hiểu biết kĩ càng về các vi xử lý và kiến thức lập trình vi xử lý bằng Assembly.

Sự ra đời của bộ vi xử lý Intel 8086 năm 1978 là một sự kiện trọng đại đối với ngành công nghiệp máy tính. Bộ vi xử lý 8086 là trung tâm của bất kỳ máy tính nào, từ Windows, Mac hay Linux, đã biến Intel từ một công ty sản xuất chip bán dẫn nhỏ thành một tên tuổi nổi tiếng nhất trên thế giới.

Sinh viên

Phạm Minh Tâm.

Chương I: Giới thiệu về vi xử lý 8086

1.1 Tổng quan về họ vi xử lý 8086

Intel-8086 là một CPU 16 bit (bus dữ liệu ngoại có 16 dây). Nó được dùng để chế tạo các máy vi tính PC-AT đầu tiên của hãng IBM vào năm 1981. Trong thực tế, hãng IBM đã dùng CPU 8088 (là một dạng của CPU 8086 với bus số liệu giao tiếp với ngoại vi là 8 bit) để chế tạo máy vi tính cá nhân đầu tiên gọi là PC-XT.

Cho đến nay CPU 8086 đã không ngừng cải tiến và đã trải qua các phiên bản 80186, 80286, 80386, 80486, Pentium (80586), Pentium Pro, Pentium MMX, Pentium II, III, 4. Các CPU trên tương thích từ trên xuống (downward compatible) nghĩa là tập lệnh của các CPU mới chế tạo gồm các tập lệnh của CPU chế tạo trước đó được bổ sung thêm nhiều lệnh mạnh khác.

Các thông số của 8086 như sau:

- Năm sản xuất: 6/1978
- fclkmax (đồng hô nhịp): 10MHz
- MIPS (triệu lệnh/s): 0, 33
- Số tranzitor: 29000
- Bus số liệu: 16 bit
- Bus địa chỉ: 20 bit
- Khả năng địa chỉ: 1 MB
- Số chân: 40
- Độ dài bộ nhớ đệm lệnh (hàng đợi): 6 byte
- Có thể thao tác với bit, byte, từ, từ khối.
- Có khả năng thực hiện phép tính với các số 8 và 16 bit có dấu hoặc không có dấu dạng nhị phân hoặc thập phân, bao gồm cả phép chia và nhân.

Cấu trúc tổng quát của CPU-8086 gồm 2 bộ phận chính là: Bộ thực hiện lệnh và bộ phận giao tiếp bus.

1. Bộ phận thực hiện lệnh (EU):

Thi hành các tác vụ mà lệnh yêu cầu như: Kiểm soát các thanh ghi (đọc/ghi), giải mã và thi hành lệnh. Trong EU có bộ tính toán và luận lý (ALU) thực hiện được các phép toán số học và luận lý. Các thanh ghi đa dụng là các ô nhớ bên trong CPU chứa dữ liệu tương tự như ô nhớ trong bộ nhớ. Cờ cũng là một thanh ghi dùng để ghi lại trạng thái hoạt động của ALU. Thanh ghi lệnh chứa nội dung lệnh hiện tại mà CPU đang thực hiện.

Các thanh ghi và bus trong EU đều là 16 bit. EU không kết nối trực tiếp với bus hệ thống bên ngoài. Nó lấy lệnh từ hàng chờ lệnh mà BIU cung cấp. Khi có yêu cầu truy xuất bộ nhớ hay ngoại vi thì EU yêu cầu BIU làm việc. BIU có thể tái định địa chỉ để cho phép EU truy xuất đầy đủ 1 MB (8086 có 20 đường địa chỉ ngoại).

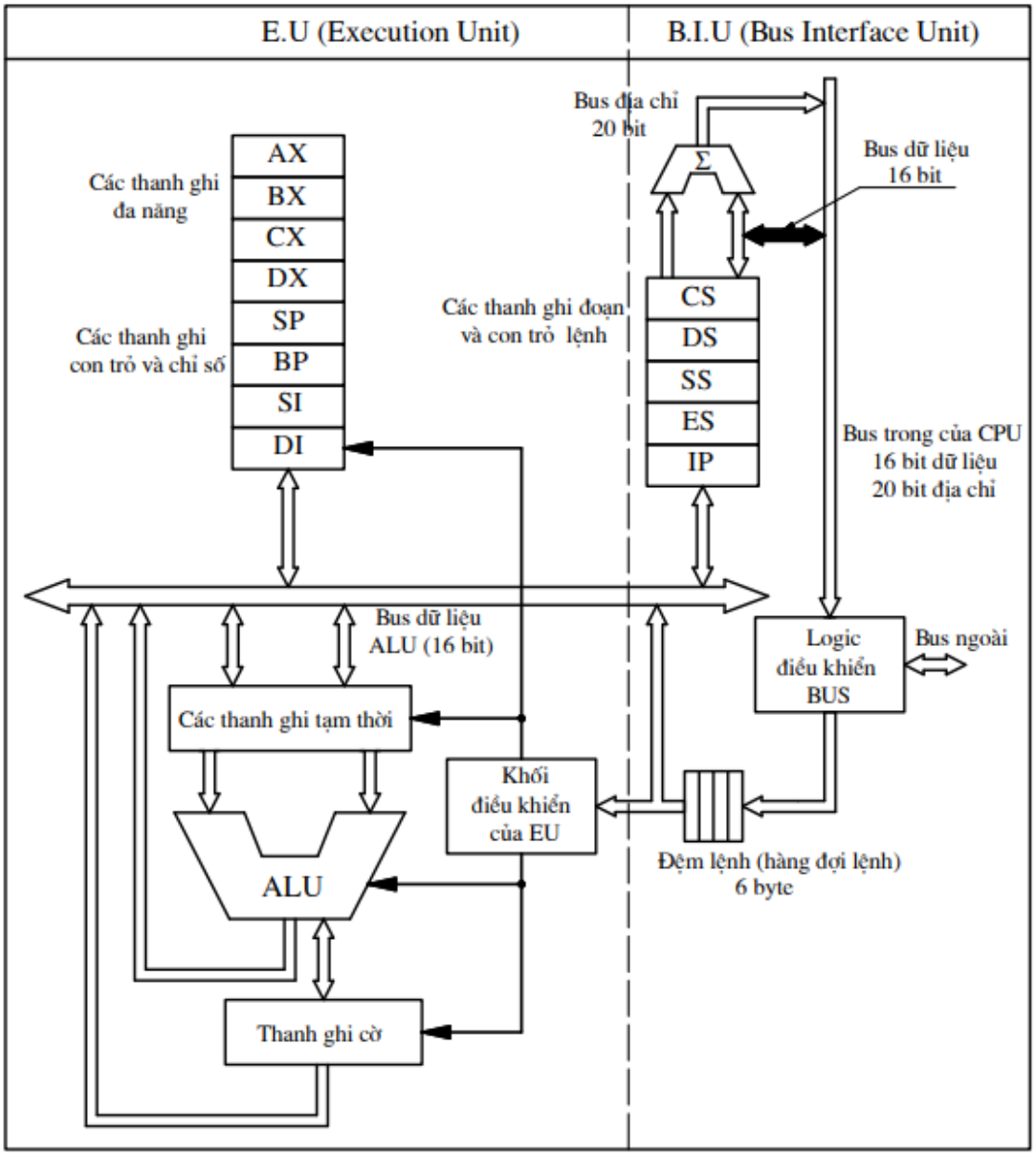
2. Bộ giao tiếp bus (BIU):

BIU thực hiện chức năng giao tiếp giữa EU với bên ngoài (Bộ nhớ, thiết bị ngoại vi …) thông qua hệ thống BUS ngoại (bus dữ liệu và bus địa chỉ). BIU thực hiện tất cả các tác vụ về bus mỗi khi EU có yêu cầu. Khi EU cần trao đổi dữ liệu với bên ngoài, BIU sẽ tính toán địa chỉ và truy xuất dữ liệu để phục vụ theo yêu cầu EU.

Trong BIU có 5 thanh ghi CS, DS, ES, SS và IP chứa địa chỉ. Thanh ghi IP chứa địa chỉ của lệnh sẽ được thi hành kế tiếp nên gọi là con trỏ lệnh.

EU và BIU liên lạc với nhau thông qua hệ thống bus nội. Trong khi EU đang thực hiện lệnh thì BIU lấy lệnh từ bộ nhớ trong nạp đầy vào hàng chờ lệnh (6 bytes). Do đó EU không phải đợi lấy lệnh từ bộ nhớ. Đây là một dạng đơn giản của cache để tăng tốc độ đọc lệnh.

Sơ đồ khối của vi xử lý 8086:



Sơ đồ chân:

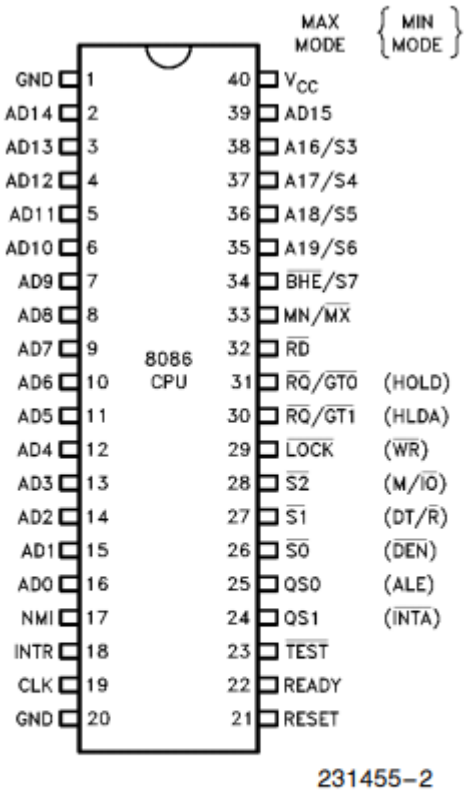


Figure 2. 8086 Pin Configuration

- ✓ AD0 ÷ AD15 : Các chân dồn kênh cho các tín hiệu của bus dữ liệu và bus địa chỉ. Xung ALE sẽ báo cho mạch ngoài biết khi nào trên các đường đó có tín hiệu dữ liệu (ALE=0) hoặc địa chỉ (ALE=1). Tín hiệu này chuyển sang trạng thái trở kháng cao khi Bus nội bộ ghi nhận tín hiệu treo
- ✓ A16/S3, A17/S4, A18/S5, A19/S6: Địa chỉ/trạng thái. Đây là 4 đường địa chỉ cao nhất. Địa chỉ A16 – A19 sẽ có mặt tại các chân đó khi ALE=1 còn khi ALE=0 thì trên các chân đó có tín hiệu trạng thái S3 – S6. Bit S6=0 liên tục, bit S5 phản ánh giá trị bit IF của thanh ghi cờ, hai bit S3, S4 phối hợp với nhau để chỉ ra việc truy nhập các thanh ghi đoạn. Tín hiệu này chuyển sang trạng thái trở kháng cao khi Bus nội bộ ghi nhận tín hiệu treo.
- ✓ RD : Đọc. Tín hiệu đọc cho biết bộ vi xử lý đang thực hiện đọc bộ nhớ hay đọc I/O phụ thuộc vào trạng thái chân S2. RD=0 thì bus dữ liệu sẵn sàng nhận số liệu từ bộ nhớ hoặc thiết bị ngoại vi. Tín hiệu này chuyển sang trạng thái trở kháng cao khi Bus nội bộ ghi nhận tín hiệu treo.
- ✓ READY : Tín hiệu báo cho CPU biết tình trạng sẵn sàng của thiết bị ngoại vi hay bộ nhớ. Khi READY=1 thì CPU thực hiện đọc/ghi mà không cần chèn thêm các chu kỳ đợi. Ngược lại khi thiết bị ngoại vi hay bộ nhớ có tốc độ hoạt động chậm, chúng có

thể đưa tín hiệu READY=0 để báo cho CPU biết mà chờ chúng, lúc này CPU tự động kéo dài thời gian thực hiện lệnh đọc/ghi bằng cách chèn thêm các chu kỳ đợi.

- ✓ INTR : tín hiệu yêu cầu ngắt che được. Khi có yêu cầu ngắt mà cờ cho phép ngắt IF=1 thì CPU kết thúc lệnh đang làm dở, sau đó đi vào chu kỳ chấp nhận ngắt và đưa ra bên ngoài tín hiệu INTA=0.
- ✓ TEST : Tín hiệu tại chân này được kiểm tra bởi lệnh WAIT (xem phần tập lệnh). Khi CPU thực hiện lệnh WAIT mà lúc đó tín hiệu TEST=1 nó sẽ chờ cho đến khi TEST=0 thì nó mới thực hiện lệnh tiếp theo.
- ✓ NMI : Tín hiệu yêu cầu ngắt không che được. Tín hiệu này không chịu sự khống chế của cờ IF và nó sẽ được CPU nhận biết bằng tác động của sườn lên của xung yêu cầu ngắt. Nhận được yêu cầu này CPU kết thúc lệnh đang làm dở sau đó nó chuyển sang chương trình phục vụ ngắt kiểu INT 2.
- ✓ RESET : Tín hiệu khởi động lại 8086. Khi RESET=1 kéo dài ít nhất trong thời gian 4 chu kỳ đồng hồ thì 8086 buộc phải khởi động lại: nó xoá các thanh ghi DS, ES, SS, IP, FR về 0 và bắt đầu thực hiện chương trình tại địa chỉ CS:IP=FFFF:0000H (cờ IF=0 để cấm các yêu cầu ngắt tác động vào CPU và cờ TF=0 để bộ vi xử lý không bị đặt trong chế độ chạy từng lệnh).
- ✓ CLK : Tín hiệu đồng hồ (xung nhịp). Xung nhịp có độ rỗng là 77% và cung cấp nhịp làm việc cho CPU.
- ✓ Vcc : Chân nguồn nuôi, tại đây CPU được cung cấp nguồn +5V±10%, 340mA.
- ✓ GND : Hai chân nguồn để nối với điểm 0V của nguồn nuôi.
- ✓ MN/MX : Chân điều khiển hoạt động của CPU theo chế độ MIN/MAX.

Chương II: Thuật toán MD5

2.1 Giới thiệu 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.

Các đồng hóa MD5 được dùng rộng rãi trong các phần mềm trên toàn thế giới để đảm bảo việc truyền tập tin được nguyên vẹn. Ví dụ, máy chủ tập tin thường cung cấp một checksum MD5 được tính toán trước cho tập tin, để người dùng có thể so sánh với checksum của tập tin đã tải về. Những hệ điều hành dựa trên nền tảng Unix luôn kèm theo tính năng MD5 sum trong các gói phân phối của họ, trong khi người dùng Windows sử dụng ứng dụng của hãng thứ ba.

Tuy nhiên, hiện nay dễ dàng tạo ra xung đột MD5, một người có thể tạo ra một tập tin để tạo ra tập tin thứ hai với cùng một checksum, do đó kỹ thuật này không thể chống lại một vài dạng giả mạo nguy hiểm. Ngoài ra, trong một số trường hợp checksum không thể tin tưởng được (ví dụ, nếu nó được lấy từ một lệnh như tập tin đã tải về), trong trường hợp đó MD5 chỉ có thể có chức năng kiểm tra lỗi: nó sẽ nhận ra một lỗi hoặc tải về chưa xong, rất dễ xảy ra khi tải tập tin lớn.

MD5 được dùng rộng rãi để lưu trữ mật khẩu. Để giảm bớt sự dễ thương tổn đề cập ở trên, ta có thể thêm salt vào mật khẩu trước khi băm chúng. Một vài hiện thực có thể áp dụng vào hàm băm hơn một lần-xem làm mạnh thêm khóa.

2.2 Giải thuật tính MD5

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.

Giải thuật MD5 chính hoạt động trên trạng thái 128-bit, được chia thành 4 từ 32-bit, với ký hiệu *A*, *B*, *C* và *D*. Chúng được khởi tạo với những hằng số cố định. Giải thuật chính sau đó sẽ xử lý các khối tin 512-bit, 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. Có 4 khả năng cho hàm *F*; mỗi cái được dùng khác nhau cho mỗi vòng:

$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$

$\oplus, \wedge, \vee, \neg$ lần lượt chỉ phép XOR, AND, OR và NOT.

Đây là quá trình thực hiện xử lý của 4 hàm F, G, H, I ở trên:

Vòng 1 (Round 1): Ký hiệu [abcd j s i] là bước thực hiện của phép toán a = b + ((a + F(b, c, d) + M[j] + K[i]) <<< s) Quá trình thực hiện 16 bước sau:

[ABCD 0 7 1] [DABC 1 12 2] [CDAB 2 17 3] [BCDA 3 22 4]

[ABCD 4 7 5] [DABC 5 12 6] [CDAB 6 17 7] [BCDA 7 22 8]

[ABCD 8 7 9] [DABC 9 12 10] [CDAB 10 17 11] [BCDA 11 22 12]

[ABCD 12 7 13] [DABC 13 12 14] [CDAB 14 17 15] [BCDA 15 22 16]

Giải thích: ví dụ biểu thức thứ 2 là [DABC 1 12 2], tương đương với:

D = A + ((D + F(A,B,C) + M[1] + K[2]) <<< 12) Nhận xét: Vòng 1 dùng hàm F, Với giá trị j từ 0 -> 15 và i từ 1 -> 16

Vòng 2 (Round 2): Tương tự, ký hiệu [abcd j s i] là của biểu thức: a = b + ((a + G(b, c, d) + M[j] + K[i]) <<< s) Quá trình thực hiện 16 bước:

[ABCD 1 5 17] [DABC 6 9 18] [CDAB 11 14 19] [BCDA 0 20 20]

[ABCD 5 5 21] [DABC 10 9 22] [CDAB 15 14 23] [BCDA 4 20 24]

[ABCD 9 5 25] [DABC 14 9 26] [CDAB 3 14 27] [BCDA 8 20 28]

[ABCD 13 5 29] [DABC 2 9 30] [CDAB 7 14 31] [BCDA 12 20 32]

Nhận xét: Vòng 2 dùng hàm G, với i từ 17 -> 32 và j = 1 + 5j mod 16

Vòng 3 (Round 3):

Tương tự, ký hiệu [abcd j s i] là của biểu thức: a = b + ((a + H(b, c, d) + M[j] + K[i]) <<< s)

Quá trình thực hiện 16 bước:

[ABCD 5 4 33] [DABC 8 11 34] [CDAB 1 16 35] [BCDA 14 23 36]

[ABCD 1 4 37] [DABC 4 11 38] [CDAB 7 16 39] [BCDA 10 23 40]

[ABCD 13 4 41] [DABC 0 11 42] [CDAB 3 16 43] [BCDA 6 23 44]

[ABCD 9 4 45] [DABC 12 11 46] [CDAB 5 16 47] [BCDA 2 23 48]

Nhận xét: Vòng 3 dùng hàm H, với i từ 33 -> 48 và j =5 + 3j mod 16

Vòng 4 (Round 4):

Tương tự, ký hiệu [abcd j s i] là của biểu thức:

a = b + ((a + I(b,c,d) + M[j] + K[i]) <<< s)

Quá trình thực hiện 16 bước:

[ABCD 0 6 49] [DABC 7 10 50] [CDAB 14 15 51] [BCDA 5 21 52]

[ABCD 12 6 53] [DABC 3 10 54] [CDAB 10 15 55] [BCDA 1 21 56]

[ABCD 8 6 57] [DABC 15 10 58] [CDAB 6 15 59] [BCDA 13 21 60]

[ABCDb 4 6 61] [DABC 11 10 62] [CDAB 2 15 63] [BCDA 9 21 64]

Nhận xét: Vòng 4 dùng hàm I, với i từ 49 -> 64 và j =7j mod 16

/* Sau đó làm các phép cộng sau. (Nghĩa là cộng vào mỗi thanh ghi giá trị của nó trước khi vào vòng lặp) */

A = A + AA

B = B + BB

C = C + CC

D = D + DD

End /* of loop on i */

Bước 5: Tính kết quả message digest. Sau khi thực hiện xong bước 4, thông điệp thu gọn nhận được từ 4 thanh ghi A, B, C, D, bắt đầu từ byte thấp của thanh ghi A và kết thúc với byte cao của thanh ghi D bằng phép nối như sau: Message Digest = A || B || C || D. (|| phép toán nối)

2.3 Mã giả

```
//Chú ý: Tất cả các biến đều là biến không dấu 32 bit và bao phủ mô đun 2^32 khi tính toán
var int[64] r, k

//r xác định số dịch chuyển mỗi vòng
r[ 0..15]:= {7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22}
r[16..31]:= {5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20}
r[32..47]:= {4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23}
```

```
r[48..63]:= {6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10,
15, 21}

//Sử dụng phần nguyên nhị phân của sin của số nguyên làm hằng số:
for i from 0 to 63
    k[i]:= floor(abs(sin(i + 1)) × (2 pow 32))

//Khởi tạo biến:
var int h0:= 0x67452301
var int h1:= 0xEFCDAB89
var int h2:= 0x98BADCFE
var int h3:= 0x10325476

//Tiền xử lý:
append "1" bit to message
append "0" bits until message length in bits ≡ 448 (mod 512)
append bit (bit, not byte) length of unpadded message as 64-bit
little-endian integer to message

//Xử lý mẩu tin trong đoạn 512-bit tiếp theo:
for each 512-bit chunk of message
    break chunk into sixteen 32-bit little-endian words w[i], 0 ≤ i ≤
15

    //Khởi tạo giá trị băm cho đoạn này:
    var int a:= h0
    var int b:= h1
    var int c:= h2
    var int d:= h3

    //Vòng lặp chính:
    for i from 0 to 63
        if 0 ≤ i ≤ 15 then
            f:= (b and c) or ((not b) and d)
            g:= i
        else if 16 ≤ i ≤ 31
            f:= (d and b) or ((not d) and c)
            g:= (5×i + 1) mod 16
        else if 32 ≤ i ≤ 47
            f:= b xor c xor d
            g:= (3×i + 5) mod 16
        else if 48 ≤ i ≤ 63
            f:= c xor (b or (not d))
            g:= (7×i) mod 16
```

```
        temp:= d
        d:= c
        c:= b
        b:= b + leftrotate((a + f + k[i] + w[g]), r[i])
        a:= temp

    //Thêm bảng băm của đoạn vào kết quả:
    h0:= h0 + a
    h1:= h1 + b
    h2:= h2 + c
    h3:= h3 + d

var int digest:= h0 append h1 append h2 append h3 //(expressed as
little-endian)
```

Chương III: Lập trình bằng ngôn ngữ assembly

Do vi xử lý 8086 chỉ có các thanh ghi 16bit nên khi cộng các số 32 bit ta chia thành 2 phần 16 bit rồi thực hiện 2 lần tính toán: tính các giá trị 16 bit thấp rồi tính các giá trị 16 bit cao

Với các giá trị 0<= i <= 15, các hàm được tính toán

```
f:= (b and c) or ((not b) and d)
g:= i
```

Triển khai với ngôn ngữ assembly:

```
hamf:
  mov ax,b   ;
  mov bx,c   ;
  and ax,bx  ; b_&c_
  mov tt1,ax
  mov ax,b
  mov bx,d
  not ax
  and ax,bx  ;~b_&d_
  mov bx,tt1
  or ax,bx   ;(b_&c_)|(~b_&d_)
  mov f,ax
  mov ax,b+2
  mov bx,c+2
  and ax,bx  ; b^&c^
  mov tt1,ax
  mov ax,b+2
  mov bx,d+2
  not ax
  and ax,bx  ;~b^&d^
```

```
mov bx,tt1
or ax,bx    ;(b^&c^)|(~b^&d^)
mov f+2,ax
mov ax,cx
neg ax
add ax,64
mov g,ax
jmp md5tiep
```

Với các giá trị 16<= i <= 31, các hàm được tính toán

```
f:= (d and b) or ((not d) and c)
g:= (5×i + 1) mod 16
```

Triển khai với ngôn ngữ assembly:

```
hamg:
mov ax,d    ;
mov bx,b    ;
and ax,bx   ; d_&b_
mov tt1,ax
mov ax,d
mov bx,c
not ax
and ax,bx   ;~d_&c_
mov bx,tt1
or ax,bx    ;(d_&b_)|(~d_&c_)
mov f,ax
mov ax,d+2
mov bx,b+2
and ax,bx   ; d^&b^
```

```
mov tt1,ax
mov ax,d+2
mov bx,c+2
not ax
and ax,bx   ;~d^&c^
mov bx,tt1
or ax,bx    ;(d^&b^)|(~d^&c^)
mov f+2,ax
mov ax,cx
neg ax
add ax,64   ;i     ;=64-i
mov bl,5
mul bl      ;i*5
add ax,1    ;5*i+1
mov dl,16
div dl      ;(5*i+1)%16
mov al,ah
mov ah,0
mov g,ax
jmp md5tiep
```

Với các giá trị 32<= i <= 47, các hàm được tính toán

```
f:= b xor c xor d
g:= (3×i + 5) mod 16
```

Triển khai với ngôn ngữ assembly:

```
hamh:
  mov ax,b    ;
  mov bx,c    ;
```

```
xor ax,bx   ; b_(+)c_
mov bx,d
xor ax,bx   ;b_(+)c_(+)d_
mov f,ax
mov ax,b+2
mov bx,c+2
xor ax,bx   ; b^(+)c^
mov bx,d+2
xor ax,bx   ;b(+)&c^(+)d^
mov f+2,ax
mov ax,cx
neg ax
add ax,64   ;i      ; = 64-i
mov bl,3
mul bl      ;i*3
add ax,5    ;3*i+5
mov dl,16
div dl      ;(3*i+5)%16
mov al,ah
mov ah,0
mov g,ax
jmp md5tiep
```

Với các giá trị 48<= i <= 63, các hàm được tính toán

```
f:= c xor (b or (not d))
g:= (7×i) mod 16
```

Triển khai với ngôn ngữ assembly:

```
hamt:
```

```
mov ax,b    ;
mov bx,d    ;
not bx
or ax,bx    ; b_|~d_
mov bx,c
xor ax,bx   ;c_(+)(b_|~d_)
mov f,ax
mov ax,b+2
mov bx,d+2
not bx
or ax,bx    ; b^|~d^
mov bx,c+2
xor ax,bx   ;c^(+)(b^|~d^)
mov f+2,ax
mov ax,cx
neg ax
add ax,64   ;i
mov dl,7
mul dl      ;7*i
mov dl,16
div dl      ;(7*i)%16
mov al,ah
mov ah,0
mov g,ax
```

Hàm dịch trái

```
leftrotate((a + f + k[i] + w[g]), r[i])
```

Triển khai bằng assembly

```
mov ax,cx
neg ax
add ax,64          ;i
mov bx,ax
mov al,r+bx        ;r[i]
 ; so lan dich trai
mov bx,tt1         ; bit thap
mov dx,tt2         ; bit cao
dichtrai:          ;   (a+f+k[i]+w[g]) duoc luu trong dx bx
   clc
   rcl bx,1       ;quay trai voi co nho
   rcl dx,1       ;
   rcr bx,1       ; quay phai voi co nho
   rol bx,1       ; quay trai          ;;dich 1 lan
   sub al,1
   cmp al,0
   jne dichtrai ;LEFTROTATE((a+f+k[i]+w[g]),r[i])
```

Chi tiết thuật toán có thể xem tại:

https://github.com/tampmbk/MD5Implement8086/blob/master/md5.asm

Tài liệu tham khảo

1. http://vhoc.net/wp-content/uploads/2015/05/Vhoc.Netnghien-cuu-8086.pdf
2. https://vi.wikipedia.org/wiki/X86
3. https://www.slideshare.net/dark_valley/h-vi-x-l-8086-intel
4. Proteus Manual - https://www.ele.uva.es/~jesman/BigSeti/ftp/Cajon_Desastre/Software-Manuales/EBook%20-%20Proteus%20Manual.pdf
5. Walter A. Triebel, Avtar Singh - The 8088 and 8086 Microprocessors: Programming, Interfacing, Software, Hardware and Applications - 1997.
6. Ngô Lam Trung - Slide môn Kỹ thuật vi xử lý