VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

KIẾN TRÚC MÁY TÍNH

Nhóm 07, học kì 231

Bài tập lớn

Advisor(s):	Thầy Vũ Trọng Thiên	
Student(s):	Đoàn Trí Hùng	ID 2211322
	Đỗ Phú Vinh Hiển	ID 2211035
	Châu Quốc Bảo	ID 2210195
	Hồ Lê Viết Bảo	ID 2210207

HO CHI MINH CITY, DECEMBER 2023

Contents

Phần 1: Chia 2 số thực chuẩn IEEE 754 chính xác đơn.

1.1 Giới thiệu đề tài

Đề 7:

Viết chương trình thực hiện phép chia 2 số thực chuẩn IEEE 754 chính xác đơn mà không dùng các lệnh tính toán số thực của MIPS. Dữ liệu đầu vào đọc từ file lưu trữ dạng nhị phân trên đĩa FLOAT2.BIN (2 trị x 4 bytes = 8 bytes).

Số thực dạng chuẩn IEEE 754

Là cách biểu diễn các số thực trong máy tính đã được chuẩn hóa bởi tổ chức Institute of Electrical and Electrocnic Engineers vào năm 1985. Trong máy tính có hai dạng sử dụng rộng rãi là kiểu dữ liệu float (32bit, "Chính xác đơn") và double (64bit, "Chính xác kép") trong ngôn ngữ C chuẩn (ANSI C).

	Phần dấu(S)	Phần mũ(E)	Phần phân số(F)
Chính xác đơn	1	8	23
Chính xác kép	1	11	52

$$\text{Giá trị hệ thập phân} = (\text{-}1)^S \times (1 + \text{phân số}) \times 2^{\text{số mũ}-"bias"}$$

$$\text{Giá trị phân số}_{\text{chính xác đơn}} = \sum_{i=-1}^{-23} (x_i \times 2^i)$$

Các giá trị đặc biệt

- E = 0, F = 0 $\Longrightarrow$ biểu diễn số thực 0.0 ở hệ thập phân.
- E = 0, F $\neq$ 0 $\Longrightarrow$ biểu diễn các "tái chuẩn hóa", các số này có giá trị nhỏ hơn giá trị cùng dấu của số nhỏ nhất biểu diễn được bằng dạng chuẩn tắc.

$$\text{Giá trị hệ thập phân} = (\text{-}1)^S \times (0 + \text{phân số}) \times 2^{\text{số mũ}-"bias"}$$

- E toàn 1, M = 0 $\Longrightarrow$ biểu diễn giá trị vô cùng (S = 0: $+\infty$, S = 1: $-\infty$).
- E toàn 1, M $\neq$ 0 $\Longrightarrow$ biểu diễn giá trị phi số (0/0 hoặc cộng trừ hai giá trị vô cùng).

Định dạng trong file input FLOAT2.bin: lưu trữ mã IEEE 754 của hai số thực.

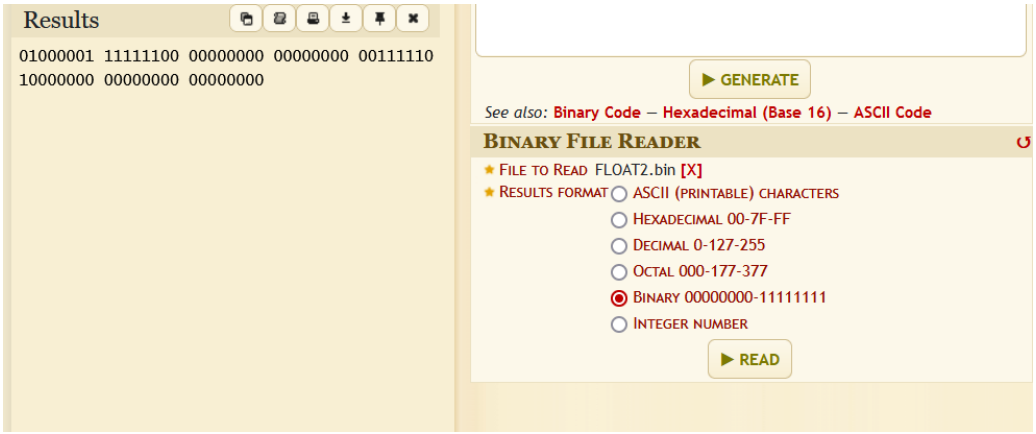


Figure 1.1: Nội dung của file bin

1.2 Giải pháp hiện thực

Phép chia của các số dạng dấu chấm động theo tiêu chuẩn IEEE 754 (X1 và X2) được thực hiện bằng cách chia phần định trị (mantissa) và trừ các số mũ.

$$\begin{aligned} X3 &= (X1/X2) \\ &= \frac{(-1)^{S1} \times (M1 \times 2^{E1})}{(-1)^{S2} \times (M2 \times 2^{E2})} \\ &= (-1)^{S3} \times (M1/M2) \times 2^{E1-E2} \end{aligned}$$

trong đó:

M là phần định trị (1 + phần phân số), đại diện cho các bit của phần thập phân hoặc nhị phân của số.

S là bit dấu (sign bit), xác định dấu của số (0 cho dương, 1 cho âm).

E là phần số mũ (exponent).

- Xét số chia X1 và số bị chia X2, ta có được các kết quả đặc biệt:
  - "NaN": Not a Number, nếu cả X1 và X2 là số 0 hoặc là "vô cùng" (Infinity).
  - Với X1 $\neq$ 0, X2 = 0: X1 dương $\longrightarrow$ X1/X2 = Infinity, X1 âm $\longrightarrow$ X1/X2 = -Infinity
  - Chỉ X2 là vô cùng: Kết quả là $\pm$ 0.0 (thế hiện dấu của phép chia).
- Bit dấu S3 = (S1 $\oplus$ S2).(phép toán xor)

- Tìm phần mantissa bằng cách chia M1/M2
- Số mũ E3 = (E1 - E2) + bias (chính xác đơn: bias = 127)
- Chuẩn hóa nếu cần, tức là dịch trái phần mantissa và giảm số mũ kết quả đi một.
- Kiểm tra overflow và underflow
  - Overflow: E3 > E_{Max} (1111 1111) trả về "âm vô cùng" (-Infinity) nếu bit dấu bằng 1 hoặc "dương vô cùng" (+Infinity) nếu bit dấu bằng 0.
  - Underflow: E3 < E_{Min} (0000 0000) trả về số 0.

Example:

	S	E	F
X1 = -30.75	1	10000011	11101100000000000000000

	S	E	F
X2 = 2.5	0	10000000	01000000000000000000000

1) Phần dấu: S3 = S1 $\oplus$ S2 = 1

2) Phần mũ:

$$E3 = (E1 - E2) + bias = (10000011) - (10000000) + (1111111)$$
$$= 131 - 128 + 127 = 130 \Longrightarrow (10000010)$$

3) Chia phần định trị M1/M2

M = 1.(phần phân số) $\Longrightarrow$ M1 = 1.11101100000000000000000, M2 = 1.01000000000000000000000

- Bước 1: Khởi tạo thương = 0, count = 0 ;
- Bước 2: M1 $\geq$ M2 ? M1 = M1 - M2, thương dịch trái, bit thấp nhất = 1: thương dịch trái, bit thấp nhất = 0 ;
- Bước 3: Dịch trái 1 bit phần định trị M1, count++ ;
- Bước 4: count == 24 ? dừng : quay lại bước 2 ;

Thương tìm được : 1.10001001100110011001101

1.01000000000000000000000) 1.11101100000000000000000
− 1.01000000000000000000000)
(> 0, thương = 1) 0.10101100000000000000000
(dịch trái 1) 1.01011000000000000000000
− 1.01000000000000000000000)
(> 0, thương = 1.1) 0.00011000000000000000000
(dịch trái 1) 0.00110000000000000000000
− 1.01000000000000000000000)
(< 0, thương = 1.10)
(cộng với số chia, dịch trái 1) 0.01100000000000000000000
− 1.01000000000000000000000)
(< 0, thương = 1.100)
(cộng với số chia, dịch trái 1) 0.11000000000000000000000
− 1.01000000000000000000000)
(< 0, thương = 1.1000)
(cộng với số chia, dịch trái 1) 1.10000000000000000000000
.....

$\Longrightarrow$ Phần phân số F3 = 10001001100110011001101

4) Kết quả của phép chia

	S	E	F
X3 = X1 / X2	1	10000010	10001001100110011001101

X3 in decimal = $(-1)^1 \times (1 + F3_{10}) \times 2^{130-127}$ = -12.30

1.3 Đánh giá chương trình

CPU time = IC * CPI * clock cycle time = (IC * CPI) / Clock rate

CR=1GHz, CPI = 1

Table 1.1: Kiểm tra chương trình

Begin of Table

Số bị chia	Số chia	Kết quả	Lệnh R	Lệnh I	Lệnh J	Thời gian chạy (10^{-7}s)
-43993.0	-6.1038784E7	7.2073843E-4	186	188	9	3.83
-4.552292E8	46397.0	-9811.607	181	183	8	3.72
-2.8832275E8	-7065.0	40810.01	182	184	9	3.75
-6718.0	184900.0	-0.03633315	177	179	8	3.64
-2369.0	1222475.0	-0.0019378719	180	182	9	3.71
-3.498227E7	1.5878598E7	-2.2031083	186	188	9	3.83
-98.0	1127300.0	-8.6933374E-5	175	177	8	3.60
-1.3209192E8	-5242.0	25198.764	181	183	8	3.72
-323.0	83315.0	-0.0038768528	183	185	9	3.77
-16281.0	24345.0	-0.66876155	183	185	8	3.76
358.0	11101.0	0.032249346	178	180	8	3.66
891.0	1.3835224E9	6.440083E-7	184	186	8	3.78
-23.0	-4312961.0	5.332763E-6	179	181	8	3.68
2.4154142E7	-1.613657E9	-0.014968572	187	189	9	3.85
-5.2897366E8	-5391.0	98121.62	184	186	8	3.78
532237.0	53936.0	9.867935	181	183	9	3.73
-720.0	363.0	-1.983471	184	188	9	3.79
-75472.0	2291222.0	-0.032939628	181	183	8	3.72
-510.0	-206.0	2.475728	180	182	8	3.70
-1.9333126E8	-28658.0	6746.1533	182	184	9	3.75
-2.629702E8	2423849.0	-108.49281	181	183	8	3.72
-2022.0	-10.0	202.2	180	182	8	3.70

Continuation of Table 1.1

Số bị chia	Số chia	Kết quả	Lệnh R	Lệnh I	Lệnh J	Thời gian chạy (10^{-7}s)
3307.0	-1221625.0	-0.00270705	179	181	8	3.68
-2.5104514E8	787.0	-318990.0	178	180	8	3.66
-282.0	-921050.0	3.061723E-4	177	179	9	3.65
-1.4830842E8	3.0	-4.9436136E7	184	186	9	3.79
-32738.0	-127.0	257.7795	179	181	8	3.68
-1.2897068E9	9.73413E8	-1.3249327	183	185	9	3.77
-2408956.0	624427.0	-3.8578663	184	186	9	3.79
-2.53565E8	-771.0	328878.06	177	179	8	3.64
-779.0	-1577.0	0.49397588	184	186	9	3.79
6034.0	7102.0	0.8496198	182	184	9	3.75
-3.4892426	0.0	-Infinity	65	68	5	1.38
3.8952712E9	0.0	Infinity	65	68	4	1.37
Infinity	49.5625	Infinity	72	71	4	1.47
-3.329552E7	291.0	-114417.59	183	185	8	3.76
-1618.0	-32.0	50.5625	173	175	8	3.56
-696568.0	-1.1821933E8	0.0058921664	179	181	9	3.69
1356103.0	125.0	10848.823	181	183	9	3.73
-3187949.0	197192.0	-16.166725	178	180	8	3.66
Infinity	Infinity	NaN	70	67	4	1.41
0.0	0.0	NaN	62	58	4	1.24
Infinity	-Infinity	NaN	70	67	4	1.41
-7.95445E8	1673.0	-475460.22	180	182	9	3.71
793.0	Infinity	0.0	72	70	4	1.46
3.791084E-37	-Infinity	-0.0	72	72	5	1.49

Continuation of Table 1.1

Số bị chia	Số chia	Kết quả	Lệnh R	Lệnh I	Lệnh J	Thời gian chạy (10^{-7}s)
0.0	168960.0	0.0 (Underflow)	93	95	10	1.98
71.0	-1.5810939E8	-4.4905616E-7	184	186	9	3.79
-700841.0	-1.3334226E9	5.2559556E-4	179	181	8	3.68
1.1869738E-38	5.541229E19	0.0 (Underflow)	93	95	10	1.98
5.5340232E19	5.877472E-39	Infinity (Overflow)	91	96	9	1.96
2.4016037E29	-3.5264853E-38	-Infinity (Overflow)	91	95	9	1.95

End of Table

Nhận xét:

- Với trường hợp cần chuẩn hóa phần định trị: dịch trái và giảm số mũ thì thời gian chạy là lâu nhất:
  Ví dụ: Phần định trị của -43993.0 (X1) là 1.01010111101100100000000 và của -6.1038784E7 (X2) là 1.11010001101100000110000. Ta cần dịch trái M1 1 bit thì M1 $>$ M2, đồng thời giảm số mũ của phép tính (E3) đi 1 đơn vị.
- Thời gian chạy nhanh nhất trong trường hợp 0/0 $\longrightarrow$ "NaN".

Hình ảnh về output:

```
The value of the dividend: 11001100111110111111001000000010
-1.3209192E8
The value of the divisor: 11000101101000111101000000000000
-5242.0
The result of the division: 01000110110001001101110110000111
25198.764
-- program is finished running --
```

Figure 1.2

```
The value of the dividend: 11001010000100110000011111110000
-2408956.0
The value of the divisor: 01001001000110000111001010110000
624427.0
The result of the division: 11000000011101101110011101001000
-3.8578663
-- program is finished running --
```

Figure 1.3

```
The value of the dividend: 00000000000000000000000000000000
0.0
The value of the divisor: 00000000000000000000000000000000
0.0
The result of the division: 01111111110000000000000000000000
NaN
-- program is finished running --
```

Figure 1.4: Not a Number

```
The value of the dividend: 11000000010111110100111111000000
-3.4892426
The value of the divisor: 00000000000000000000000000000000
0.0
The result of the division: 11111111100000000000000000000000
-Infinity
-- program is finished running --
```

Figure 1.5: Infinity case

```
The value of the dividend: 00000000100000010100000000000000
1.1869738E-38
The value of the divisor: 01100000010000000100000000000000
5.541229E19
The result of the division: 00000000000000000000000000000000
0.0
-- program is finished running --
```

Figure 1.6: Underflow

```
The value of the dividend: 01100000010000000000000000000000
5.5340232E19
The value of the divisor: 00000000010000000000000000000000
5.877472E-39
The result of the division: 01111111100000000000000000000000
Infinity
-- program is finished running --
```

Figure 1.7: Overflow

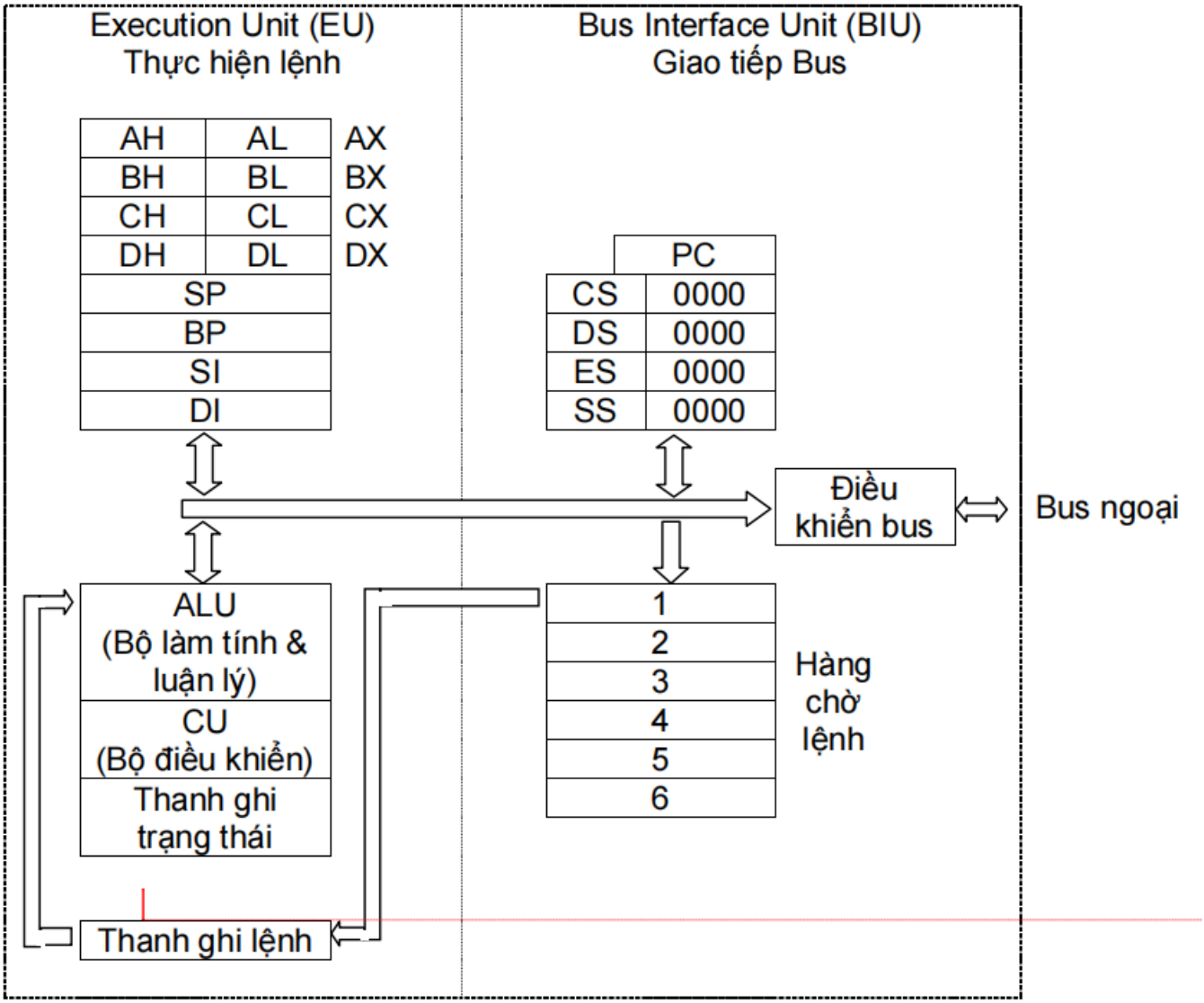
Xem tất cả output tại đây: https://github.com/hungdt31/BTL_KTMT_231

Phần 2: Tìm hiểu về kiến trúc tập lệnh 8086/80286

2.1 Tìm hiểu tổng quan về tập lệnh 8086/80286

2.1.1 Giới thiệu chung

Tập lệnh của họ vi xử lý 8086/80286 đảm bảo tương thích thế hệ sau với thế hệ trước. điều đó có nghĩa là các chương trình viết cho 8086 vẫn chạy được trên các bộ vi xử lý mới hơn mà không phải thay đổi (không đảm bảo thứ tự ngược lại). Tập lệnh của một bộ vi xử lý thường có rất nhiều lệnh (hàng trăm lệnh), vì thế mà việc tiếp cận và làm chủ chúng là trương đối khó khăn.



Hình: *Sơ đồ khối của CPU 8086*

Trong CPU 8086, bộ phận thực hiện lệnh (*EU - Execution Unit*) và bộ phận giao tiếp Bus (*BIU - Bus Interface Unit*) đóng vai trò chính.

EU kiểm soát các thanh ghi, giải mã và thực hiện các lệnh CPU yêu cầu. Đây là nơi chủ yếu thực hiện các tác vụ của CPU và hoạt động với thanh ghi và đường bus 16 bit. EU không trực tiếp kết nối với các bus hệ thống bên ngoài mà lấy lệnh từ hàng chờ lệnh được cung cấp bởi BIU. Khi cần truy xuất bộ nhớ hoặc thiết bị ngoại vi, EU sẽ yêu cầu BIU thực hiện nhiệm vụ đó. Ở mức độ địa chỉ, 8086 có thể truy cập đến 1MB thông qua việc sử dụng 20 đường địa chỉ ngoại.

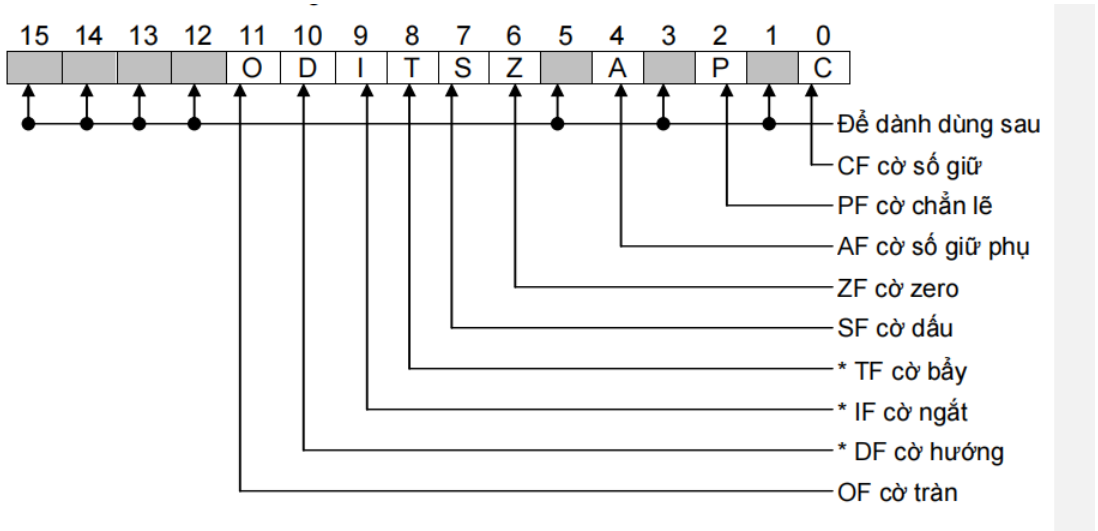
BIU, bộ phận giao tiếp bus, thực hiện các tác vụ liên quan đến việc truy cập bus của EU. Trong khi EU đang thực hiện lệnh, BIU sẽ lấy các lệnh từ bộ nhớ trong và lưu trữ chúng vào hàng chờ lệnh bên trong CPU. Điều này giúp EU không phải đợi lấy lệnh từ bộ nhớ mỗi khi cần, tạo ra một cơ chế đơn giản tương tự cache để tối ưu hóa việc lấy lệnh.

2.1.2 Các thanh ghi của 8086

1. **Thanh ghi đa dạng:** CPU 8086 có 4 thanh ghi đa dụng 16bit, có thể chia đôi thành 8 thanh, mỗi thanh 8 bit.
   (a) AX (accumulator): là thanh ghi tích luỹ cơ bản, mọi tác vụ vào/ra đều dùng thanh ghi này, tác vụ dùng số liệu tức thời, một số tác vụ chuỗi ký tự và các lệnh tính toán đều dùng thanh ghi AX.
   (b) BX (base register): là thanh ghi nền thường dùng để tính toán địa chỉ ô nhớ.
   (c) CX (count register): là thanh ghi đếm thường dùng để đếm số lần trong một lệnh vòng lặp hoặc xử lý chuỗi ký tự.
   (d) DX (data register): thường chứa địa chỉ của một số lệnh vào ra, lệnh tính toán số học (kể cả nhân và chia).
2. **Thanh ghi con trỏ:** Dùng để thâm nhập số liệu trên ngăn xếp.
   (a) SP (stack pointer): Thanh ghi con trỏ ngăn xếp.
   (b) BP (base pointer): Thanh ghi con trỏ nền dùng để lấy số liệu từ ngăn xếp.
3. **Thanh ghi chỉ số**
   (a) SI (source index): Thanh ghi chỉ số nguồn.
   (b) DI (destination index): Thanh ghi chỉ số đích.

4. **Thanh ghi đoạn:** Được dùng trong mọi tính toán địa chỉ ô nhớ. Mỗi thanh ghi đoạn xác định 64 KB ô nhớ trong bộ nhớ trong.
   (a) CS (code segment): Thanh ghi đoạn mã lệnh.
   (b) DS (data segment): Thanh ghi đoạn dữ liệu.
   (c) ES (extra segment): Thanh ghi đoạn thêm. Các pháp tính chuỗi dùng DI đều liên quan đến ES.
   (d) SS (stack segment): Thanh ghi đoạn ngăn xếp. Con trỏ SP luôn trỏ tới đỉnh của ngăn xếp.

5. **Thanh ghi cờ:** Phản ảnh kết quả của phép tính toán số học và luận lý, xác định trạng thái hoạt động của CPU. Các bit trên thanh ghi cờ có ý nghĩa được trình bày dưới đây.
   (a) CF: thể hiện số giữ thoát ra từ bit cao nhất của thanh ghi kết quả sau một phép tính toán.
   (b) OF: thể hiện việc tính toán vượt quá khả năng của CPU.
   (c) AF: thể hiện số giữ thoát ra từ bít thứ 4 (bit 3) của thanh ghi kết quả.
   (d) PF: bằng 1 nếu 8 bít thấp của thanh ghi kết quả một phép tính toán có số con số 1 chẳn (và ngược lại).
   (e) ZF: bằng 1 khi kết quả phép tính bằng 0 (và ngược lại).
   (f) DF: có thể lập trình được, bằng 1 thì SI và DI giảm 1 cho mỗi vòng lặp.
   (g) IF: có thể lập trình được, bằng 1 cho phép ngắt.
   (h) TF: có thể lập trình được, bằng 1 khi cho phép chương trình chạy từng bước để phục vụ sửa sai một chương trình.



2.1.3 Các nhóm lệnh

A. Nhóm các lệnh vận chuyển (sao chép) dữ liệu.

1. **MOV – MOV a byte or word** (chuyển một byte hay từ)
   Dạng lệnh: *MOV Đích, Nguồn* Mô tả: Đích←Nguồn
   Trong đó toán hạng đích và Nguồn có thể tìm được theo các chế độ địa chỉ khác nhau, nhưng phải có cùng độ dài và không được phép đồng thời là hai ô nhớ hoặc hai thanh ghi đoạn. Các cờ bị thay đổi: không

2. **OUT – Output a byte or a work to a port.**
   Dạng lệnh: *OUT Port, Acc* Mô tả: Acc→Port
   Trong đó port là dữ liệu của cổng có địa chỉ port. Port là địa chỉ 8 bit của cổng, nó có thể là các giá trị trong khoảng 00...FFH. Như vậy có thể có các khả năng sau đây.
   (a) Nếu Acc là AL thì dữ liệu 8 bit được đưa ra cổng Port.
   (b) Nếu Acc là AX thì dữ liệu 16 bit được đưa ra cổng Port và Port + 1.

3. **IN – Input data from a port** (đọc dữ liệu từ cổng vào thanh ghi Acc).
   Dạng lệnh: *IN Acc, địa_chỉ_cổng*
   Lệnh IN truyền một byte hoặc một từ từ một cổng vào lần lượt tới thanh ghi AL hoặc AX. Địa chỉ của cổng có thể được xác định là một hằng tức thì kiểu byte cho phép truy nhập các cổng từ 0...255 hoặc thông qua một số đã được đưa ra trước đó trong thanh ghi DX mà cho phép truy nhập các cổng từ 0...65535.

4. **POP – Pop word from top of Stack** (lấy lại 1 từ vào thanh ghi từ đỉnh ngăn xếp)
   Dạng lệnh: *POP Đích* Mô tả: Đích←SP SP←SP+2
   Toán hạng đích đích có thể là các thanh ghi đa năng, thanh ghi đoạn (nhưng không được là thanh ghi đoạn mã CS) hoặc ô nhớ.

5. **PUSH – Push word on the Stack** (cất 1 từ vào ngăn xếp)
   Dạng lệnh: *PUSH Nguồn* Mô tả: SP←SP-2 Nguồn→SP Toán hạng đích đích có thể là các thanh ghi đa năng, thanh ghi đoạn(kể cả CS) hoặc ô nhớ.

B. Nhóm các lệnh tính toán số học

1. **ADC – Add with Carry** (cộng có nhớ)
   Dạng lệnh: *ADC Đích, Nguồn* Mô tả: Đích ← Đích + Nguồn + CF
   Cộng hai toán hạng Đích và Nguồn với cờ CF kết quả lưu vào Đích.

2. **ADD** (cộng hai toán hạng)
   Dạng lệnh: *ADD Đích, Nguồn* Mô tả: Đích ← Đích + Nguồn
   Cộng hai toán hạng đích và Nguồn kết quả lưu vào đích.

3. **DEC – Decrement** (giảm byte hay word đi một giá trị)
   Dạng lệnh: *DEC Đích*. DEC trừ toán hạng Đích đi 1. Toán hạng Đích có thể là byte hay word.

4. **DIV – Division** (chia không dấu)
   Dạng lệnh: *DIV Nguồn*
   Toán hạng Nguồn là số chia. Tuỳ theo độ dài toán hạng Nguồn ta có hai trường hợp bố trí phép chia.

   (a) Nếu Nguồn là là số 8 bit: AX/Nguồn, thương để vào AL, số dư để vào AH

   (b) Nếu Nguồn là số 16 bit: DXAX/Nguồn, thương để vào AX, số dư để vào DX

   Nếu thương không phải là số nguyên nó được làm tròn theo số nguyên sát dưới. Nếu Nguồn bằng 0 hoặc thương thu được lớn hơn FFH hoặc FFFFH (tuỳ theo độ dài của toán hạng Nguồn) thì 8086 thực hiện lệnh ngắt INT 0.

5. **INC – Increment** (tăng toán hạng lên 1)
   Dạng lệnh: *INC đích* Mô tả: Đích ← Đích + 1
   Lệnh này tăng đích lên 1, tương đương với việc ADD đích, 1 nhưng chạy Nhãnh hơn.

6. **MUL – Multiply unsigned byte or word** (nhân số không dấu)
   Dạng lệnh: *MUL Nguồn*
   Thực hiện phép nhân không dấu với toán hạng Nguồn (ô nhớ hoặc thanh ghi) với thanh ghi tổng.

   (a) Nếu Nguồn là số 8 bit: AL*Nguồn. Số bị nhân phải là số 8 bit đặt trong AL, sau khi nhân tích lưu vào AX.

   (b) Nếu Nguồn là số 16 bit: AX*Nguồn. Số bị nhân phải là số 16 bit đặt trong AX, sau khi nhân tích lưu vào DXAX. Nếu byte cao (hoặc 16 bit cao) của 16 (hoặc 32) bit kết quả chứa 0 thì CF=OF=0.

7. **NEG – Negation** (lấy bù hai của một toán hạng, đảo dấu của một toán hạng).
   Dạng lệnh: *NEG Đích* Mô tả: Đích←0-Đích
   NEG lấy 0 trừ cho đích (có thể là 1 byte hoặc 1 từ) và trả lại kết quả cho toán hạng đích, nếu ta lấy bù hai của -128 hoặc -32768 ta sẽ được kết quả không đổi nhưng OF=1 để báo

là kết quả bị tràn vì số dương lớn nhất biểu diễn được là +127 và +32767.

8. **SUB – Substract** (trừ hai toán hạng)
   Dạng lệnh: *SUB Đích, Nguồn* Mô tả: Đích←Đích - Nguồn
   Toán hạng đích vào Nguồn phải chứa cùng một loại dữ liệu và không được đồng thời là hai ô nhớ, cũng không được là thanh ghi đoạn.

C. Nhóm các lệnh tính toán logic

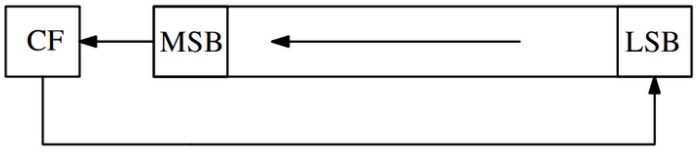
1. **AND** (phép và logic)
   Dạng lệnh: *AND Đích, Nguồn* Mô tả: Đích ← Đích ∧ Nguồn
   Thực hiện phép và logic hai toán hạng và lưu kết quả vào toán hạng đích. Người ta thường sử dụng để che đi/giữ lại một vài bit nào đó của một toán hạng bằng cách nhân logic toán hạng đó với toán hạng tức thì có các bit 0/1 ở các vị trí cần che đi/giữ lại tương ứng.

2. **NOT – Logical Negation** (phủ định logic)
   Dạng lệnh: *NOT Đích*
   NOT đảo các giá trị của các bit của toán hạng đích.

3. **OR – Logic OR** (phép hoặc logic)
   Dạng lệnh: *OR Đích, Nguồn* Mô tả: Đích = Đích ∨ Nguồn
   Toán hạng Đích và Nguồn phải chứa dữ liệu cùng độ dài và không được phép đồng thời là hai ô nhớ và cũng không được là thanh ghi đoạn.
   Phép OR thường dùng để lập một vài bit nào đó của toán hạng bằng cách cộng logic toán hạng đó với các toán hạng tức thời có các bit 1 tại vị trí tương ứng cần thiết lập.

D. Nhóm các lệnh dịch, quay toán hạng

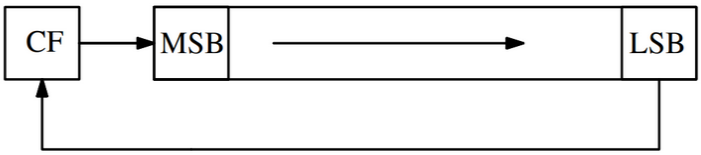
1. **RCL – Rotate though CF to the Left** (quay trái thông qua cờ nhớ)
   Dạng lệnh: *RCL Đích, CL*
   Mô tả:



   Lệnh này để quay toán hạng sang trái thông qua cờ CF, CL phải được chứa sẵn số lần quay. Trong trường hợp quay 1 lần có thể viết RCL Đích, 1 Nếu số lần quay là 9 thì toán

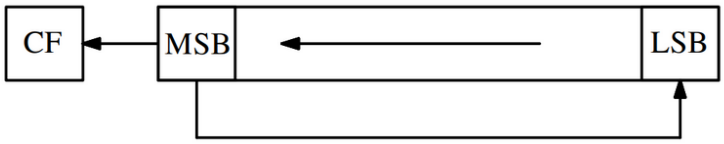
hạng không đổi vì cặp CF và toán hạng quay đúng một vòng (nếu toán hạng đích là 8 bit). Sau lệnh RCL cờ CF mang giá trị cũ của MSB, còn cờ OF←1 nếu sau khi quay 1 lần mà bit MSB bị thay đổi so với trước khi quay, cờ OF sẽ không được xác định sau nhiều lần quay. Các cờ bị thay đổi: CF, OF, SF, ZF, PF.

2. **RCR – Rotate though CF to the Right** (quay phải thông qua cờ nhớ)
   Dạng lệnh: *RCR Đích, CL*
   Mô tả:



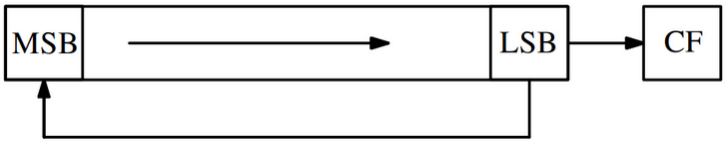
   Lệnh này để quay toán hạng sang phải thông qua cờ CF, CL phải được chứa sẵn số lần quay. Trong trường hợp quay 1 lần có thể viết RCR Đích, 1 Nếu số lần quay là 9 thì toán hạng không đổi vì cặp CF và toán hạng quay đúng một vòng (nếu toán hạng đích là 8 bit).

3. **ROL – Rotate all bit to the Left** (quay vòng sang trái).
   Dạng lệnh: *ROL Đích, CL.*
   Mô tả:



   Lệnh này dùng để quay vòng toán hạng sang trái, MSB được đưa sang cờ CF và LSB. CL phải chứa sẵn số lần quay mong muốn. Trong trường hợp quay 1 lần có thể viết ROL Đích, 1. Nếu số lần quay là 8 (CL=8) thì toán hạng không đổi vì toán hạng quay đúng một vòng (nếu toán hạng đích là 8 bit), còn nếu CL=4 thì 4 bit cao đổi chỗ cho 4 bit thấp.

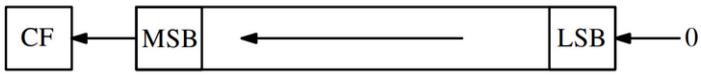
4. **ROR – Rotate all bit to the Left** (quay vòng sang phải).
   Dạng lệnh: *ROR Đích, CL* Mô tả:



   Lệnh này dùng để quay vòng toán hạng sang phải, LSB được đưa sang cờ CF và MSB. CL phải chứa sẵn số lần quay mong muốn. Trong trường hợp quay 1 lần có thể viết ROR

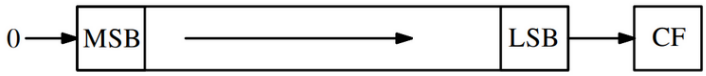
Đích, 1. Nếu số lần quay là 8 (CL=8) thì toán hạng không đổi vì toán hạng quay đúng một vòng (nếu toán hạng đích là 8 bit), còn nếu CL=4 thì 4 bit cao đổi chỗ cho 4 bit thấp.

5. **SAL/SHL - Shift Arithmetically Left (dịch trái số học)/Shift Logically Left (dịch trái logic).**
   Dạng lệnh: *SAL Đích, CL* *SHL Đích, CL*
   Mô tả:



   Hai lệnh này có tác dụng dịch trái số học toán hạng (còn gọi là dịch trái logic). Mỗi lần dịch MSB được đưa vào CF còn 0 được đưa vào LSB. CL phải chứa sẵn số lần quay mong muốn. Trong trường hợp quay 1 lần có thể viết SAL Đích, 1

6. **SHR – Shift logically Right** (dịch phải logic)
   Dạng lệnh: *SHR Đích, CL*
   Mô tả:



   Lệnh này có tác dụng dịch phải logic toán hạng. Mỗi lần dịch LSB được đưa vào CF còn 0 được đưa vào MSB. CL phải chứa sẵn số lần quay mong muốn. Trong trường hợp quay 1 lần có thể viết SHR Đích, 1

7. **XOR – Exclusive OR** (lệnh logic XOR (hoặc đảo)).
   Dạng lệnh: *XOR Đích, Nguồn* Mô tả: Đích←Đích $\oplus$ Nguồn
   Lệnh XOR thực hiện logic XOR (hoặc đảo) giữa hai toán hạng và kết quả được lưu vào trong đích, một bit kết quả được đặt bằng 1 nếu nếu các bit tương ứng hai toán hạng là đối nhau. Nếu toán hạng đích trùng toán hạng Nguồn thì kết quả bằng 0, do đó lệnh này còn được dùng để xoá thanh ghi về 0 kèm theo các cờ CF và OF cũng bị xoá.

E. Nhóm các lệnh nhảy (rẽ nhánh)

1. **JA/JNBE – Jump if Above/Jump if Not Below or Equal** (nhảy nếu cao hơn/nhảy nếu không thấp hơn hoặc bằng).
   Dạng lệnh: *JA Nhãn* *JNBE Nhãn*
   Mô tả: IP←IP+dịch chuyển
   Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới Nhãn nếu CF + ZF = 0. Quan hệ cao hơn/thấp là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ

lớn hai số không dấu. Nhãn phải nằm cách xa một khoảng -128...+127 byte so với lệnh tiếp theo sau lệnh JA/JNBE. Chương trình sẽ căn cứ vào vị trí Nhãn để xác định giá trị dịch chuyển.

2. **JAE/JNB/JNC – Jump if Above or Equal/Jump if Not Below/Jump if No Carry** (nhảy nếu lớn hơn hoặc bằng/nhảy nếu không thấp hơn/nhảy nếu không có nhớ).
   Dạng lệnh: *JAE Nhãn* *JNB Nhãn* *JNC Nhãn*
   Mô tả: IP←IP+dịch chuyển
   Ba lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới Nhãn nếu CF = 0.

3. **JB/JC/JNAE – Jump if Below/Jump if Carry/Jump if Not Above or Equal** (nhảy nếu thấp hơn/nhảy nếu có nhớ/nhảy nếu không cao hơn hoặc bằng).
   Dạng lệnh: *JB Nhãn* *JC Nhãn* *JNAE Nhãn*
   Mô tả: IP←IP+dịch chuyển
   Ba lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới Nhãn nếu CF = 1.

4. **JBE/JNA – Jump if Below or Equal/Jump if Not Above** (nhảy nếu thấp hơn hoặc bằng/nhảy nếu không cao hơn).
   Dạng lệnh: *JBE Nhãn* *JNA Nhãn*
   Mô tả: IP←IP+dịch chuyển
   Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới Nhãn nếu CF +ZF = 1

5. **JE/JZ – Jump if Equal/Jump if Zero** (nhảy nếu bằng nhau/nhảy nếu kết quả bằng không) Dạng lệnh: *JE Nhãn* *JZ Nhãn*
   Mô tả: IP←IP+dịch chuyển
   Lệnh trên biểu diễn thao tác nhảy có điều kiện tới Nhãn nếu ZF = 1

6. **JNE/JNZ – Jump if Not Equal/Jump if Not Zero** (nhảy nếu không bằng nhau/nhảy nếu kết quả không rỗng).
   Dạng lệnh: *JNE Nhãn* *JNZ Nhãn*
   Mô tả: IP←IP+dịch chuyển
   Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới Nhãn nếu ZF = 0

F. Nhóm các lệnh lặp

1. **LOOP – Loop if CX is not 0** (lặp nếu CX $\neq$ 0)
   Dạng lệnh: *LOOP Nhãn*
   Mô tả: Lệnh này dùng để lặp lại đoạn chương trình (gồm các lệnh nằm trong khoảng từ Nhãn đến hết lệnh LOOP Nhãn) cho đến khi số lần lặp CX=0. Điều này có nghĩa là

trước khi vào vòng lặp ta phải đưa số lần lặp mong muốn vào CX, và sau mỗi lần lặp thì CX tự động giảm đi 1.

2. **LOOPE/LOOPZ – Loop while CX=0 or ZF=0** (lặp lại đoạn chương trình cho đến khi CX=0 hoặc ZF=0).
   Dạng lệnh: *LOOPE Nhãn* *LOOPZ Nhãn*
   Mô tả: Lệnh này dùng để lặp lại đoạn chương trình (gồm các lệnh nằm trong khoảng từ Nhãn đến hết lệnh LOOPE Nhãn hoặc LOOPZ Nhãn) cho đến khi số lần lặp CX=0 hoặc cờ ZF=0. Điều này có nghĩa là trước khi vào vòng lặp ta phải đưa số lần lặp mong muốn vào CX, và sau mỗi lần lặp thì CX tự động giảm đi 1.

3. **LOOPNE/LOOPNZ – Loop while CX=0 or ZF=1** (lặp lại đoạn chương trình cho đến khi CX=0 hoặc ZF=1).
   Dạng lệnh: *LOOPNE Nhãn* *LOOPNZ Nhãn*
   Mô tả: Lệnh này dùng để lặp lại đoạn chương trình (gồm các lệnh nằm trong khoảng từ Nhãn đến hết lệnh LOOPNE Nhãn hoặc LOOPNZ Nhãn) cho đến khi số lần lặp CX=0 hoặc cờ ZF=1. Điều này có nghĩa là trước khi vào vòng lặp ta phải đưa số lần lặp mong muốn vào CX, và sau mỗi lần lặp thì CX tự động giảm đi 1.

G. Nhóm các lệnh điều khiển, đặc biệt khác

1. **CALL – Call a procedure** (gọi chương trình con)
   Dạng lệnh: *CALL Thủ_tục*
   Mô tả: Lệnh này dùng để chuyển hoạt động của vi xử lý từ chương trình chính (CTC) sang chương trình con (ctc). Nếu ctc nằm trong cùng một đoạn mã với CTC ta có gọi gần (near call). Nếu ctc và CTC nằm ở hai đoạn mã khác nhau ta có gọi xa (far call).

2. **NOP – No Operation** (CPU không làm gì)
   Dạng lệnh: *NOP*
   Lệnh này không thực hiện một công việc gì ngoài việc làm tăng nội dung của IP và tiêu tốn 3 chu kỳ đồng hồ. Nó thường được dùng để tính thời gian trễ trong các vòng trễ hoặc để chiếm chỗ các lệnh cần thêm vào chương trình sau này mà không làm ảnh hưởng đến độ dài chương trình.

3. **IRET – Interrupt Return** (trở về CTC từ ctc phục vụ ngắt)
   Dạng lệnh: *IRET*
   Trở về chương trình chính từ chương trình con phục vụ ngắt. Trả lại quyền điều khiển cho chương trình tại vị trí xảy ra ngắt bằng cách lấy lại các giá trị thanh ghi IP, CS và các cờ từ vùng Stack.

(a) SP→IP, SP←SP+2

(b) SP→CS, SP←SP+2

(c) SP→FR, SP←SP+2

2.2 So sánh với ISA MIPS32

Đặc điểm	8086/80286	MIPS
Kiến trúc	CISC	RISC
Số bit	16, 32, 64	64
Phiên bản	6	6
Năm ra mắt	1978	1981
Số lượng toán hạng tối đa	2 (cho lệnh số nguyên), 3 hoặc 4 (cho lệnh vector)	1-3
Kiểu kiến trúc	Register-Memory	Register-Register
Số thanh ghi (không bao gồm FP/vector)	8 (16 trong chế độ 64 bit), cộng với các thanh ghi đoạn cho quản lý bộ nhớ	4-32, bao gồm thanh ghi "zero"
Mã hóa lệnh	Có độ dài biến đổi, từ 1 đến 15 byte, tùy thuộc vào độ phức tạp	Có độ dài cố định 32 bit, đơn giản hóa việc giải mã và ống dẫn
Tính toán rẽ nhánh	Chủ yếu sử dụng mã trạng thái (cờ) được đặt bởi các lệnh trước đó để xác định kết quả nhánh	Sử dụng thanh ghi trạng thái để trực tiếp chỉ định điều kiện nhánh
Endianness	Little-endian	Little-endian và Big-endian

Giải thích chi tiết

Kiến trúc

- 8086/80286 là kiến trúc tập lệnh phức tạp (CISC) được phát triển bởi Intel, được sử dụng rộng rãi trong máy tính cá nhân và máy chủ.
- MIPS là kiến trúc máy tính có tập lệnh giảm thiểu (RISC) nổi tiếng về tính đơn giản và hiệu quả, thường được sử dụng trong các hệ thống nhúng và tính toán hiệu suất cao.

Số bit

- 8086/80286 đã phát triển từ 16 bit đến 32 bit đến 64 bit, cung cấp khả năng xử lý dữ liệu ngày càng tăng.
- MIPS chủ yếu hoạt động trên dữ liệu 64 bit, cung cấp không gian địa chỉ lớn và xử lý hiệu quả các tập dữ liệu lớn.

Phiên bản

- 8086/80286 đã trải qua nhiều lần sửa đổi, với các phần mở rộng như MMX, SSE và AVX mở rộng khả năng của nó cho xử lý đa phương tiện và vector.
- MIPS hiện đang ở phiên bản 6, với các phần mở rộng như MDMX và MIPS-3D nâng cao hiệu suất trong các miền cụ thể.

Năm ra mắt

- 8086/80286 được giới thiệu vào năm 1978, nắm giữ di sản đáng kể trong lĩnh vực điện toán.
- MIPS được giới thiệu vào năm 1981, được biết đến với sự đổi mới và tác động của nó đối với kiến trúc RISC.

Số lượng toán hạng tối đa

- 8086/80286 thường hỗ trợ tối đa 2 toán hạng cho lệnh số nguyên, với các phần mở rộng như AVX cho phép tối đa 3 hoặc 4 cho các thao tác vector nâng cao.
- MIPS nói chung cho phép 1-3 toán hạng, thúc đẩy tập lệnh hợp lý.

Kiểu kiến trúc

- 8086/80286 là kiến trúc Register-Memory, cho phép các lệnh hoạt động trực tiếp trên dữ liệu trong bộ nhớ.
- MIPS là kiến trúc Register-Register, yêu cầu dữ liệu được tải vào thanh ghi trước khi xử lý.

Tính toán rẽ nhánh

- 8086/80286 chủ yếu sử dụng mã trạng thái (cờ) được đặt bởi các lệnh trước đó để xác định kết quả nhánh.

- MIPS sử dụng thanh ghi trạng thái để trực tiếp chỉ định điều kiện nhánh.

Endianness

- 8086/80286 là kiến trúc little-endian, lưu trữ byte ít quan trọng nhất trước trong bộ nhớ.
- MIPS là kiến trúc bi-endian, hỗ trợ cả little-endian và big-endian để linh hoạt.

Kiến trúc 8086/80286 và MIPS là hai kiến trúc tập lệnh phổ biến, mỗi kiến trúc có những ưu điểm và nhược điểm riêng.
8086/80286 là kiến trúc CISC có lịch sử lâu đời và được sử dụng rộng rãi trong các hệ thống máy tính cá nhân và máy chủ. MIPS là kiến trúc RISC có hiệu suất cao và được sử dụng trong các hệ thống nhúng và tính toán hiệu suất cao.

2.3 Tìm hiểu và báo cáo về công cụ lập trình giả lập của 8086/80286: emu8086 hoặc TASM hoặc MASM

Tính fibonacci

```
.MODEL SMALL
.STACK 100H

.DATA
    RES DB 10 DUP (?) ; Reserve 10 bytes to store Fibonacci numbers
    CRLF DB 0DH, 0AH, '$' ; Carriage return and line feed character

.CODE
START:
    MOV AX, @DATA
    MOV DS, AX

    LEA SI, RES
    MOV CL, 10 ; To print 10 Fibonacci numbers

    MOV AX, 00H ; First Fibonacci number
    MOV BX, 01H ; Second Fibonacci number

L1:
    ADD AX, BX
    MOV [SI], AX
    MOV AX, BX
```

```
    MOV BX, [SI]
    INC SI
    LOOP L1

    ; Printing Fibonacci sequence
    MOV SI, OFFSET RES ; Starting address of Fibonacci sequence
    MOV CX, 10 ; Number of Fibonacci numbers to print

PRINT_FIBO:
    MOV DL, [SI] ; Retrieve Fibonacci number from RES
    ADD DL, 30H ; Convert to ASCII character
    MOV AH, 02H ; Call character output function
    INT 21H

    INC SI ; Move to the next Fibonacci number
    LOOP PRINT_FIBO ; Repeat the printing process

    ; Printing line break
    MOV DX, OFFSET CRLF
    MOV AH, 09H
    INT 21H

    MOV AH, 4CH ; Terminate the program
    INT 21H
END START
```

Listing 2.1: emu8086

2.4 Tìm hiểu và báo cáo về công cụ lập trình giả lập online của 8086/80286

Tính giai thừa của 6

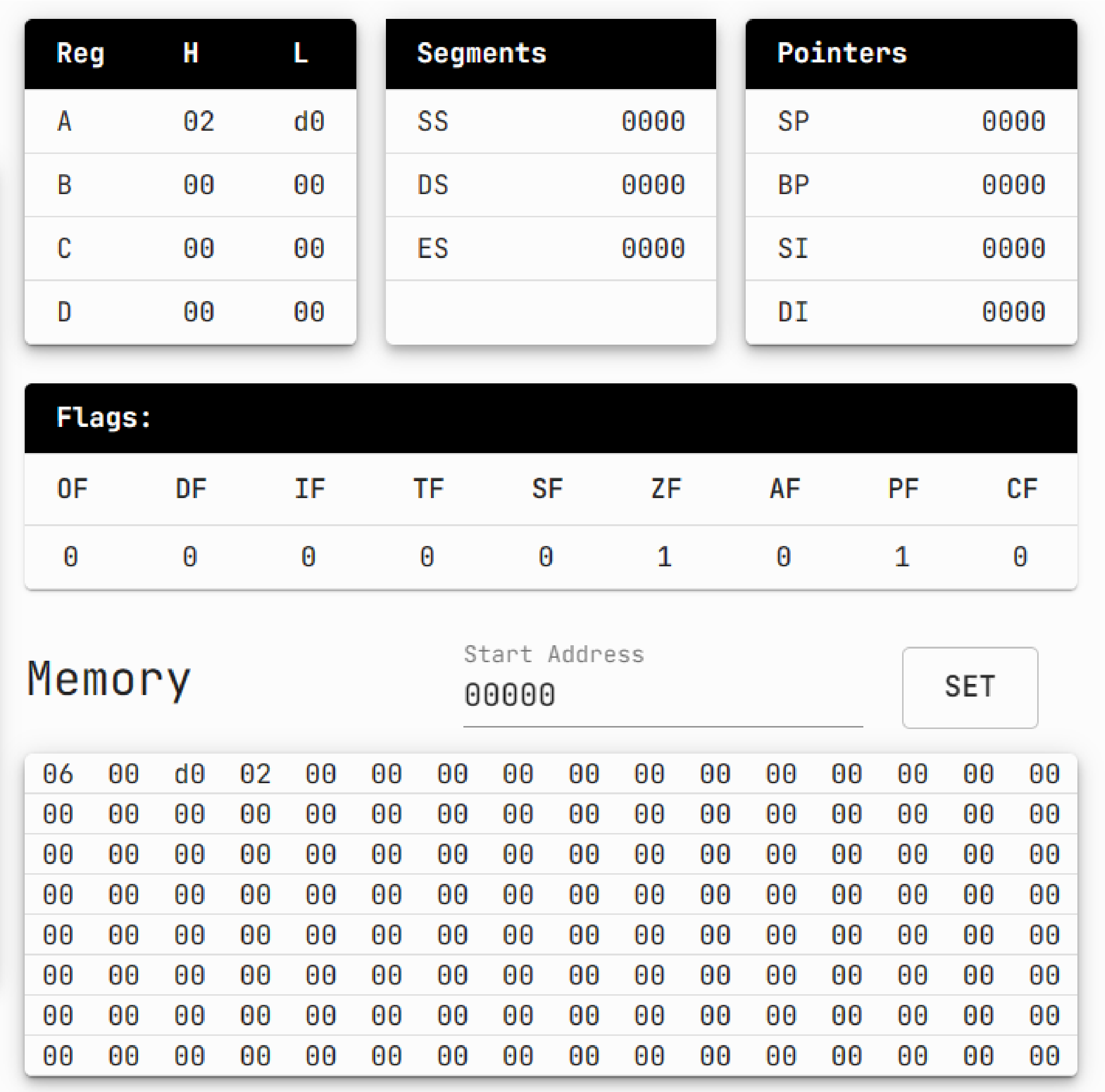
```
; Program to calculate factorial using looping
NUM: DW 0x6 ; calculate factorial of 6
RESULT: DW 0 ; place to store the result

; actual entry point of the program
start:
MOV CX,word NUM ; move number into CX
MOV AX, 0x1 ; initialize accumulator with 1
NOTZEROLOOP: ; label to jump back to
MUL CX ; multiple by the number: AX = AX.CX
DEC CX ; decrement the number: CX = CX - 1
JNZ NOTZEROLOOP ; if CX not zero jump back
MOV word RESULT,AX ; store the result in memory
print reg ; print the value of registers
```

Listing 2.2: x86 Code

DW: Dùng để lưu trữ một từ (word)

1. (label) DW số word có dấu/không dấu : Thiết lập một từ với giá trị được cung cấp.
2. (label) DW [số word không dấu] : Thiết lập số từ được chỉ định thành 0 (có thể được sử dụng để khai báo một mảng rỗng).
3. (label) DW [số word có dấu/không dấu ; số word không dấu] : Thiết lập số từ (đối số thứ hai) thành giá trị đã cho (đối số đầu tiên).
4. (label) DW "chuỗi" : Lưu trữ một chuỗi, ký tự không được bỏ qua.



CX = 0: dừng vòng lặp $\longrightarrow$ ZF = 1 (cờ zero), PF = 1 (do 8 bit thấp của CX có 0 số 1 (chẵn))
Kết quả đọc được ở thanh ghi AX là: $0x02d0_{16} = 720_{10}$