LỆNH XỬ LÝ LOGIC CỦA 8086 TRÊN ASSEMBLY (QUANG LIU)

Mục lục

Đại học Bách Khoa Hà Nội

File này Liu tổng hợp kiến thức cơ bản luôn nên nếu bro hiểu về thanh ghi , cách 8086 mã hóa câu lệnh thì đọc qua lý thuyết ở mục 1 là được , còn mục 2 mới là nội dung chính cần đưa vào slide nhé :)))

1 Một số khái niệm cơ bản

Cái này Liu ghi kiến thức sẽ sử dụng thôi,bro hiểu thì bỏ qua chứ t nghĩ bro không cần thêm vào slide

1.1 Thanh ghi cờ trạng thái

- CF : CF=1 khi có nhớ hoặc mượn từ MSB
- PF : PF=1 khi tổng số bit 1 trong kết quả là chẵn
- AF : AF=1 khi có nhớ hoặc mượn từ 1 số BCD thấp sang BCD cao
- ZF : ZF=1 khi kết quả bằng 0
- SF : SF=1 khi kết quả âm
- OF : OF=1 khi tràn số

1.2 Sử dụng MOV , các chế độ địa chỉ với MOV

Cơ bản thì cú pháp của lệnh này là : MOV ĐÍCH,NGUỒN với ĐÍCH, NGUỒN là 2 toán hạng của lệnh

1.2.1 Một số ký hiệu với các toán hạng

- Nếu thanh ghi ghi lẻ , ví dụ : AX , AL , AH ....thì là thể hiện đang sử dụng thanh ghi
- Nếu thanh ghi được để trong ngoặc vuông [], là viết tắt của DS:thanh ghi , thì thể hiện ô nhớ có địa chỉ là giá trị thanh ghi

Ví dụ [AL] là ô nhớ có địa chỉ AL, viết tắt của DS:AL

1.2.2 Nguyên lý hoạt động

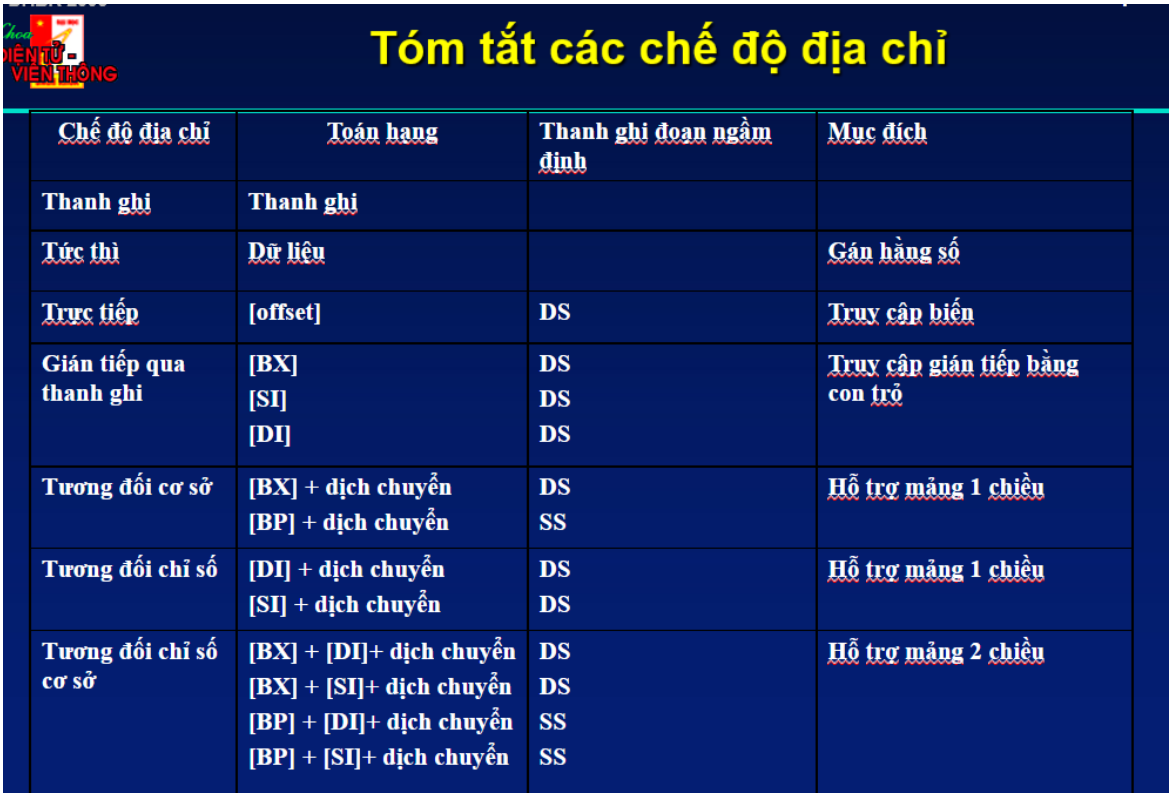
Viết cụ thể ra thì khá dài (trong slide thầy có 7 chế độ địa chỉ của MOV cơ , nhưng vì nhóm mình không nghiên cứu sâu về cái này nên Liu chỉ nói qua

về cách hoạt động của nó)
Cách thức hoạt động : copy nguồn vào đích và nội dung của nguồn không đổi :
- Nếu đích hoặc nguồn là thanh ghi thì thực hiện lệnh với giá trị ở thanh ghi
- Nếu đích hoặc nguồn là ô nhớ , thì thực hiện lệnh với giá trị có địa chỉ là giá trị trong ngoặc []
- Nêu nguồn là hằng số thì gán hằng số vào đích
=> Tuy nhiên cần ghi hệ đếm của hằng số , Assembly sẽ hiểu và tự động chuyển sang hệ hexa
Ví dụ :
-MOV AL,1234h -> Ở đây const là hệ hexa , không có gì để bàn , AL=34h
-MOV AL.1234 - > Ở đây Assembly sẽ hiểu 1234 là hệ thập phân và chuyển 1234 thành 04Dh,AL=4D
-MOV AL,1001b - > Assembly sẽ hiểu 1234 thuộc hệ nhị phân và chuyển thành 9h,AL=09
- Trường hợp nguồn có bit bậc cao nhất là chữ cáicái , ví dụ như A23Ch thì cần ghi thêm số 0 đằng trước (0A23Ch) để Assembly hiểu là hexa, để cho mấy nữa sẽ học một cái gì đó , vá cái đó sẽ không bị nhầm với hệ hexa
Một số ví dụ :
- MOV AL,BL -> gán giá trị ở thanh ghi BL vào thanh ghi AL , giá trị tại thanh ghi BL không đổi
- MOV AL,[1234h] -> gán giá trị chứa ở ô nhớ có địa chỉ 1234h , gán vào AL
- MOV [1234h],BL -> gán giá trị ở BL vào ô nhớ địa chỉ 1234h
- MOV [1234h],AX -> Vì AX chứa 16 bit , nhưng các ô nhớ chỉ được phép chứa 8 bit nên câu lệnh này sẽ gán cho 2 ô nhớ ở 2 vị trí liên tiếp có địa chỉ 1234h và 1235h
==> Tóm tắt các chế độ địa chỉ :

- **Khả năng kết hợp toán hạng của lệnh MOV**

Đích / Nguồn	Thanh ghi đa năng	Thanh ghi đoạn	ô nhớ	Hằng số
Thanh ghi đa năng	YES	YES	YES	NO
Thanh ghi đoạn	YES	NO	YES	NO
Ô nhớ	YES	YES	NO	NO
Hằng số	YES	NO	YES	NO

Khả năng kết hợp toán hạng của lệnh MOV



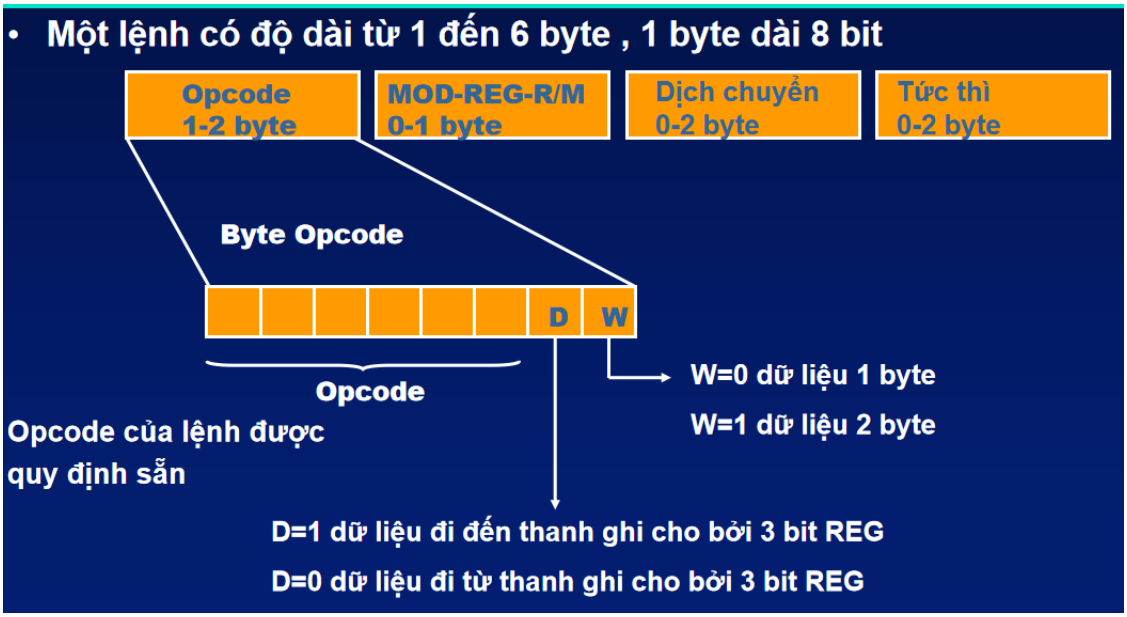
Tóm tắt các chế độ địa chỉ

Chế độ địa chỉ	Toán hạng	Thanh ghi đoạn ngầm định	Mục đích
Thanh ghi	Thanh ghi		
Tức thì	Dữ liệu		Gán hằng số
Trực tiếp	[offset]	DS	Truy cập biến
Gián tiếp qua thanh ghi	[BX] [SI] [DI]	DS DS DS	Truy cập gián tiếp bằng con trỏ
Tương đối cơ sở	[BX] + dịch chuyển [BP] + dịch chuyển	DS SS	Hỗ trợ mảng 1 chiều
Tương đối chỉ số	[DI] + dịch chuyển [SI] + dịch chuyển	DS DS	Hỗ trợ mảng 1 chiều
Tương đối chỉ số cơ sở	[BX] + [DI]+ dịch chuyển [BX] + [SI]+ dịch chuyển [BP] + [DI]+ dịch chuyển [BP] + [SI]+ dịch chuyển	DS DS SS SS	Hỗ trợ mảng 2 chiều

Tóm tắt các chế độ địa chỉ

1.3 Cách mã hóa lệnh của 8086

Một lệnh có độ dài từ 1-6 byte tùy từng lệnh khác nhau

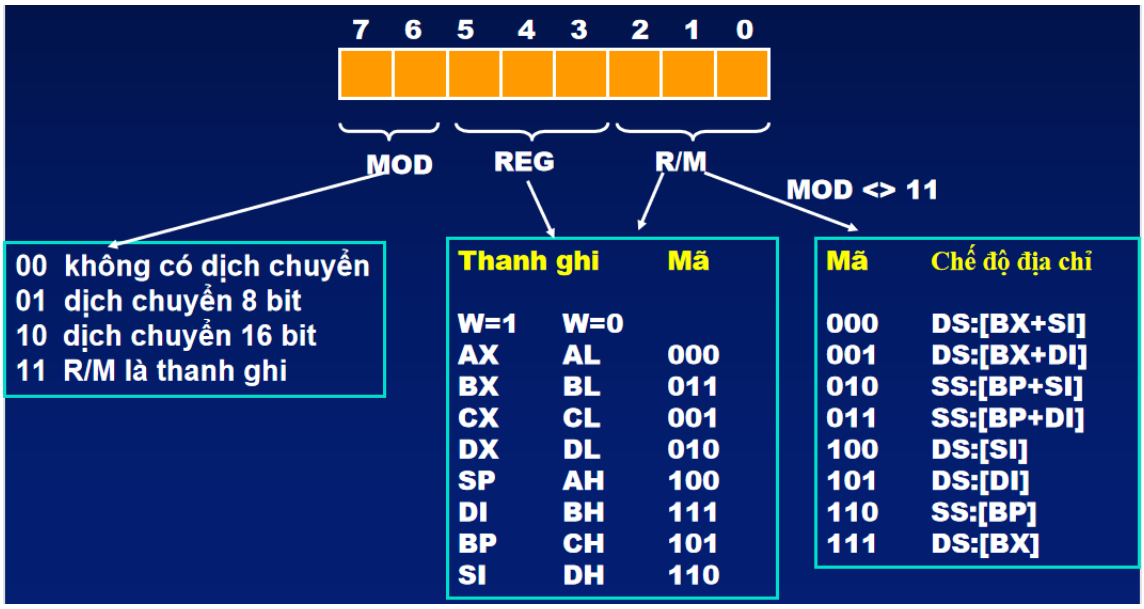


Trong đó :
MOD thể hiện chế độ dịch chuyển dữ liệu
REG thể hiện thanh ghi tham gia câu lệnh , nếu có 2 thanh ghi tham gia thì REG sẽ thể hiện thanh ghi đích
R/M là thanh ghi hoặc mã cho chế độ địa chỉ

1.3.1 Opcode 1-2 byte

-> thường thì có 8 bit , trong đó 6 bit là Opcode của lệnh sử dụng , Opcode của lệnh được quy định sẵn , D là chiều truyền dữ liệu của thanh ghi cho bởi 3 bit REG , W là kích thước dữ liệu * Mốt số Opcode của lệnh sẽ sử dụng :
- MOV : 100010
- AND : 001000
- OR : 000010
- XOR : 001100

1.3.2 MOD - REG - R/M



- MOD : Thể hiện chế độ dịch chuyển bit:
+ 00: không có dịch chuyển
+ 01 : dịch chuyển 8 bit -> dịch chuyển 1 số trong khoảng 0-2^8
+ 10 : dịch chuyển 16 bit -> dịch chuyển 1 số trong khoảng $2^8 - 2^16$
+ 11 R/M là thanh ghi , tức là câu lệnh làm việc với cả đích và nguồn đều là thanh ghi , khi đó REG sẽ mã hóa cho đích và R/M sẽ mã hóa cho nguồn
Ví dụ : [BX] là không dịch chuyển , [BX+5] là dịch chuyển 8 bit...
- REG : thanh ghi được sử dụng trongg câu lệnh , có thể là đích hoặc nguồn , trường hợp cả đích và nguồn là thanh ghi thì REG mã hóa cho đích
- R/M : nếu đích và nguồn là thanh ghi thì chọn mã theo ô thanh ghi , nếu không thì chọn mã theo ô chế độ địa chỉ

1.3.3 Dịch chuyển 0-2 byte

Nếu có dịch chuyển thì mã hóa nhị phân giá trị dịch chuyển

1.3.4 Tức thì 0-2 byte

Cái này Liu nghĩ là khi sử dụng câu lệnh với 1 hằng số , thì mã hóa nhị phân hằng số là được

1.3.5 Ví dụ

Chuyển lệnnh MOV [SI+F3H],CL sang mã máy
- Opcode MOV : 100010
- Dữ liệu là 8 bit (Vì CLCL và [SI] chỉ chứa 8 bit) : W=0
- Thanh ghi sử dụng là CL -> dữ liệu đi từ CL ra : D=0
- Dịch chuyển 8 bit : MOD=01
- Thanh ghi CL : REG=001
- R/M tương ứng với chế độ địa chỉ : R/M=100
- Dịch chuyển 0-2 byte F3H -> 1111 0011

Như vậy mã máy là 10001000 01001100 11110011

2 Một số câu lệnh LOGIC

2.1 AND

2.1.1 Đặc điểm

- Lệnh AND logic 2 toán hạng
- Cú pháp AND ĐÍCH,NGUỒN
- Thực hiện : máy sẽ nhị phân hóa đích và nguồn , sau đó AND lần lượt , khi đó ta có ĐÍCH=ĐÍCH AND NGUỒN , giá trị của NGUỒN không đổi
- Giới hạn : Toán hạng không được là 2 ô nhớ hoặc thanh ghi đoạn
- Lệnh này thay đổi cờ PF , SF , ZF , đưa cờ CF. OF về giá trị 0

2.1.2 Cách máy tính hiểu câu lệnh

Tương tự như ví dụ ở mục 1.3.5 ở trên thui
Ví dụ : AND AL,BL
- Opcode AND : 001000
- Dữ liệu là 1 8 bit (Vì CL và [SI] chỉ chứa 8 bit) : W=0
- Thanh ghi được REG mã hóa là AL -> dữ liệu đi vào AL : D=1

- Không dịch chuyển : MOD=00
- Thanh ghi AL : REG=000
- R/M tương ứng với thanh ghi BL : R/M=011
Như vậy mã máy là 0010001 00000011

2.2 OR

Tương tự như AND , Liu chỉ nói qua về AND thui chứ phần này DA viết

2.3 NOT

Phần này DA viết

2.4 XOR

2.4.1 Đặc điểm

- Lệnh XOR logic 2 toán hạng
- Cú pháp XOR ĐÍCH,NGUỒN
- Thực hiện : máy sẽ nhị phân hóa đích và nguồn , sau đó thực hiện XOR lần lượt từ bit thấp đến bit cao , khi đó ta có ĐÍCH=ĐÍCH XOR NGUỒN, giá trị của NGUỒN không đổi
- Nguyên lý XOR : cùng giá trị bit trả về 0 , khác giá trị bit trả về 1
0 XOR 0 = 0 , 1 XOR 1 = 0 , 0 XOR 1 = 1 , 1 XOR 0 = 1
- Giới hạn : Toán hạng không được là 2 ô nhớ hoặc thanh ghi đoạn

2.4.2 Cách máy tính hiểu câu lệnh

Tương tự như ví dụ ở mục 1.3.5 ở trên thui
Ví dụ : XOR AL,BL
- Opcode XOR : 001100
- Dữ liệu là 8 bit (Vì CL và [SI] chỉ chứa 8 bit) : W=0
- Thanh ghi được REG mã hóa là AL -> dữ liệu đi vào AL : D=1
- Không dịch chuyển : MOD=00
- Thanh ghi AL : REG=000
- R/M tương ứng với thanh ghi BL : R/M=011

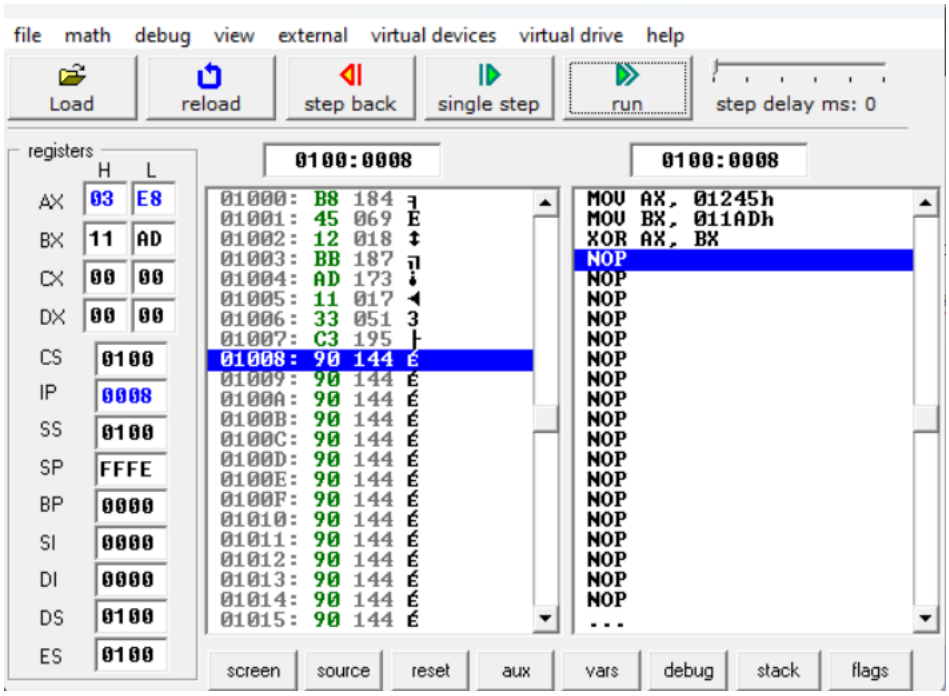
Như vậy mã máy là 00110001 00000011

2.4.3 Ví dụ

MOV AX,1245h
MOV BX,11ADh
XÔR AX,BX

=> Các bước thực hiện :
- Nhị phân hóa :
1245h - > 0001 0010 0100 0101
11ADh - > 0001 0001 1010 1101
- XOR 2 số :
0001 0010 0100 0101
0001 0001 1010 1101
0000 0011 1110 1000 => AX= 03E8h
Giá trị các thanh ghi cờ :
PF = 1 , SF = 0 , ZF = 0 ** Lưu ý : Sự khác nhau giữa XOR AX,BX và XOR BX,AX*
- XOR AX,BX thì AX là đích , giá trị của AX thay đổi , giá trị BX không đổi
- XOR BX,AX thì BX là đích , giá trị của ÃX không thay đổi

2.4.4 Test trên Assemblye emu8086



Test ví dụ 2.4.3

2.5 NEG

2.5.1 Đặc điểm

- Lệnh NEG đảo dấu của toán hạng , xác định số bù 2 của toán hạng
- Cú pháp ; NEG ĐÍCH
- Thực hiện : Nhị phân hóa đích , sau đó tìm số bù 2 của đích , ta có ĐÍCH = bù 2 của DÍCH
- Nguyên lý tìm số bù 2 : từ phải qua trái giữ bit 1 đầu tiên và các số còn lại bên trái số 1 lấy đảo lại (chỉ áp dụng cho số có bit cực phải là 1).
- Toán hạng : Là 1 thanh ghi hoặc 1 ô nhớ
- Lệnh này thay đổi hết cả thanh ghi cờ trạng thái :PF , ZF , SF , CF , OF ,AF

2.5.2 Cách máy tính hiểu câu lệnh

Vì nó chỉ có một toán hạng nên Liu cũng không chắc chắn có đúng không , nếu bro thích thì thêm vào có gì thầy sửa không thì thui :))))) Ví dụ : NEG AL
- Opcode NEG : 111101
- Dữ liệu là 1 8 bit (Vì AL chỉ chứa 8 bit) : W=0
- Thanh ghi được REG mã hóa là AL -> dữ liệu đi vào AL : D=1
- Không dịch chuyển : MOD=00
- Thanh ghi AL : REG=000
- R/M tương ứng với thanh ghi AL : R/M=000

Như vậy mã máy là 11110101 00000000

2.5.3 Ví dụ

MOV AX=0A23Ch
NEG AX
- => Các bước thực hiện :
- Nhị phân hóa :

A23C -> 1010 0010 0011 1100
- Đảo bit : 0101 1101 1100 0011
- Cộng thêm 1 :

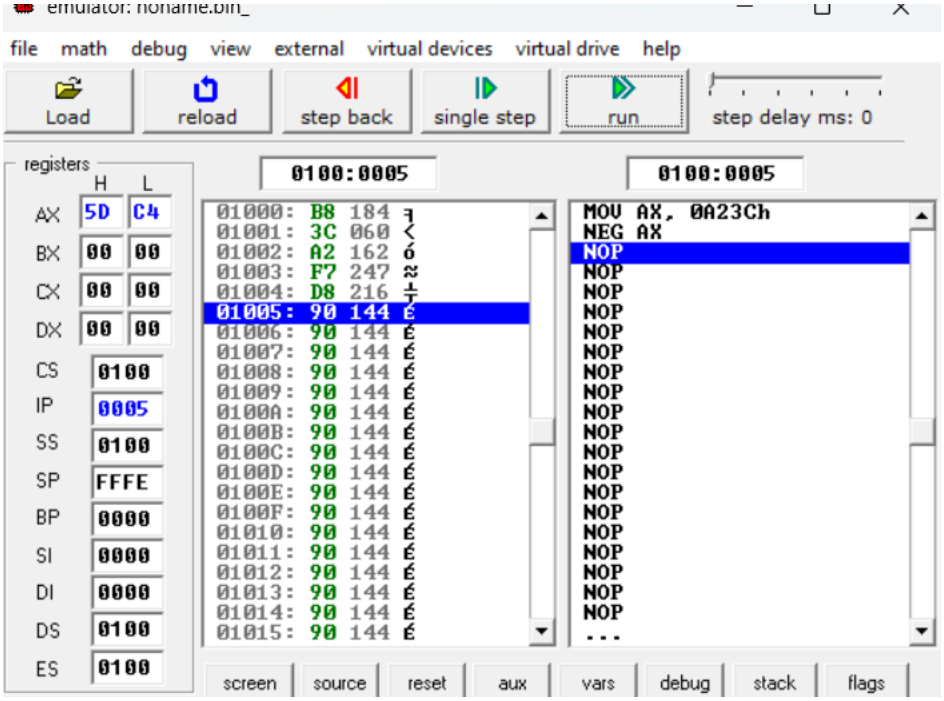
0101 1101 1100 0011 +1 = 0101 1101 1100 0100 => AX= 5DC4h

Giá trị các thanh ghi cờ :

- AF=1 , CF=1 Bởi vì thực chất lệnh NEG là tìm số âm của ĐÍCH , hay nói cách khác là 0-ĐÍCH , hoặc 0+bù 2 của ĐÍCH với 0 < ĐÍCH => AF và CF bằng 1

=> Khi sử dụng lệnh NEG thì có AF và CF bằng 1
- PF = 1 , ZF = 0 , SF = 0 , OF = 0

2.5.4 Test trên Assemblye emu8086



Test ví dụ 2.5.3

3 Câu lệnh so sánh

3.1 CMP - Compare 2 operands to Update the Flags

3.1.1 Đặc điểm

- Lệnh so sánh 2 toán hạng để cập nhật cờ
- Lệnh so sánh 2 byte hoặc 2 từ
- Cú pháp : CMP ĐÍCH,NGUỒN
- Thực hiện : thực hiện phép trừ ĐÍCH-NGUỒN rồi cập nhật cờ

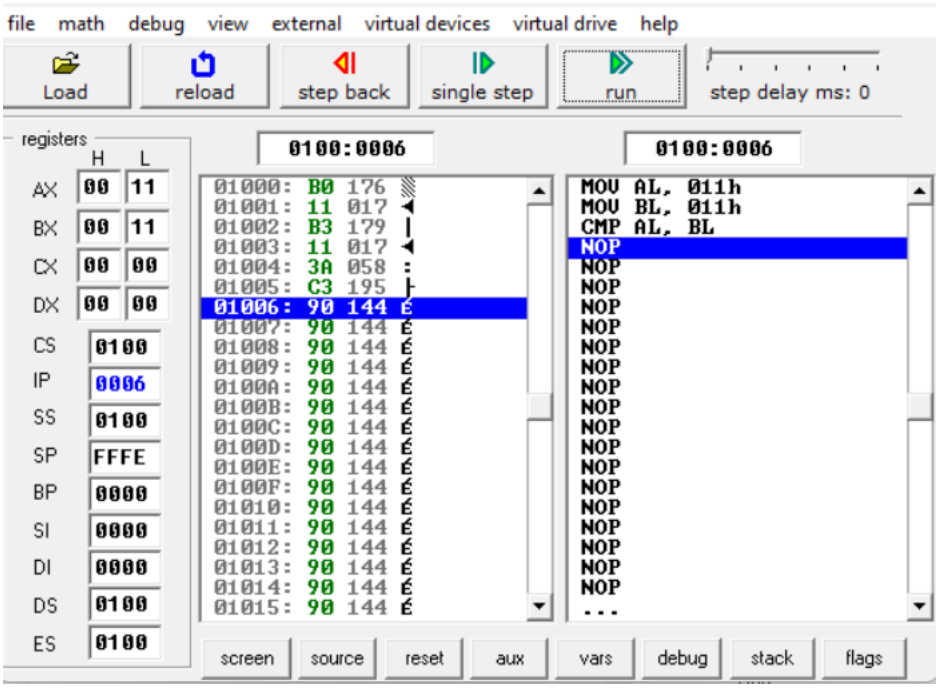
+ ĐÍCH = NGUỒN -> SF = 0 , ZF = 1
+ ĐÍCH < NGUỒN -> SF = 0 , ZF = 0
+ ĐÍCH > NGUỒN -> SF = 1 , ZF = 0

- Giới hạn : toán hạng phải cùng độ dài và không được là 2 ô nhớ

3.1.2 Ví dụ

MOV AL,11h
MOV BL,11h
CMP AL,BL
- Thực hiện :
Trừ :
11h
11h
0h
Như vậy SF = 0 , ZF = 11

3.1.3 Test trên Assemblye emu8086



Test ví dụ 3.1.2

3.2 CMPS

- Dùng để so sánh từng phần tử của 2 chuỗi có các phần tử cùng loại
- Cú pháp : CMPS CHUỖI ĐÍCH,CHUỖI NGUỒN

- Thực hiện :
+ DS:SI là địa chỉ của phần tử trong CHUỖI NGUỒN
+ ES:DI là địa chỉ của phần tử trong CHUỖI ĐÍCH
+ Sau mỗi lần so sánh SI=SI +/- 1 , DI=DI +/- 1 hoặc SI=SI +/- 2 , DI=DI +/- 2
+ Cập nhật cờ AF , CF , OF , PF , SF , ZF

3.3 TEST - AND 2 operands to Update the Flags

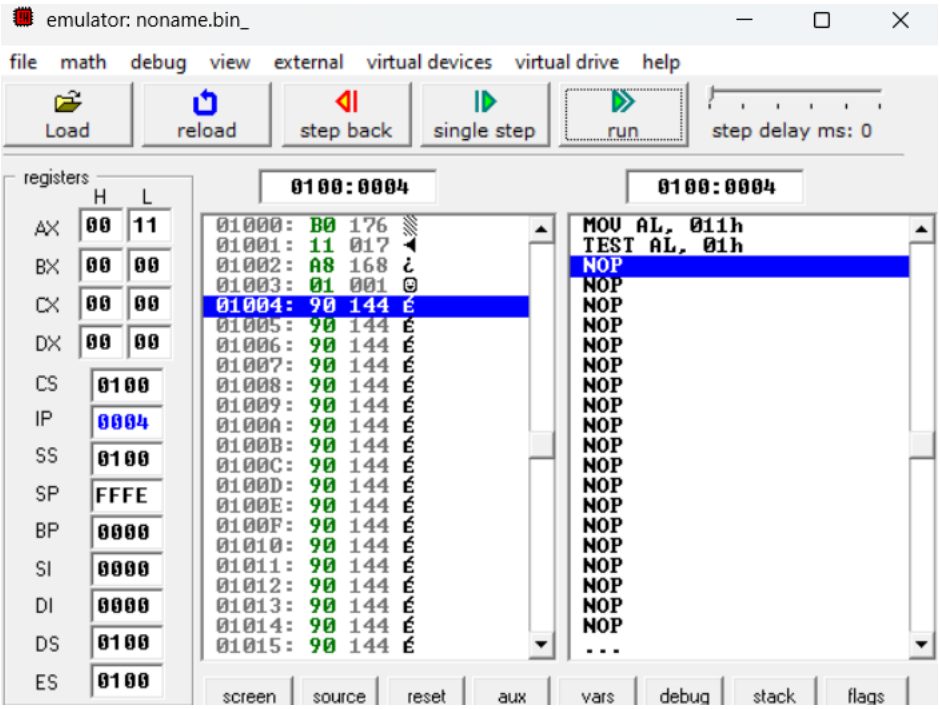
3.3.1 Đặc điểm

- Lệnh TEST sẽ AND 2 toán hạng sau đó cập nhật các bit của thanh ghi cờ
- Cú pháp : TEST ĐÍCH,NGUỒN - Thực hiện : ĐÍCH AND NGUỒN , sau đó update thanh ghi cờ

3.3.2 Ví dụ

Kiểm tra bit 0 của AL :
TEST AL,01h
+ ZF = 1 nếu AL.0 = 0 - bit 0 của AL bằng 0
+ ZF = 0 nếu AL.0 = 1 - bit 0 của AL bằng 1

3.3.3 Test trên Assemblye emu8086



Test ví dụ 3.3.2

3.4 Các lệnh thiết lập cờ

3.4.1 STC - Set the Carry Flag

CF=1

3.4.2 STD - Set the Direction Flag

DF=1

3.4.3 STI - Set the Interrupt Flag

IF=1

3.5 Các lệnh xóa cờ

3.5.1 CLC - Clear the Carry Flag

CF=0

3.5.2 CLD - Clear the Direction Flag

DF=0

3.5.3 CLI - Clear the Interrupt Flag

IF=0

3.6 CMC- Complement the Carry Flag

Lệnh đảo cờ nhớ -> sau khi dùng lệnh ta có CF bằng đảo của CF : CF=NOT(CF)

4 Các lệnh Dịch và quay

4.1 RCL - Rolate through Carry falg to the Left

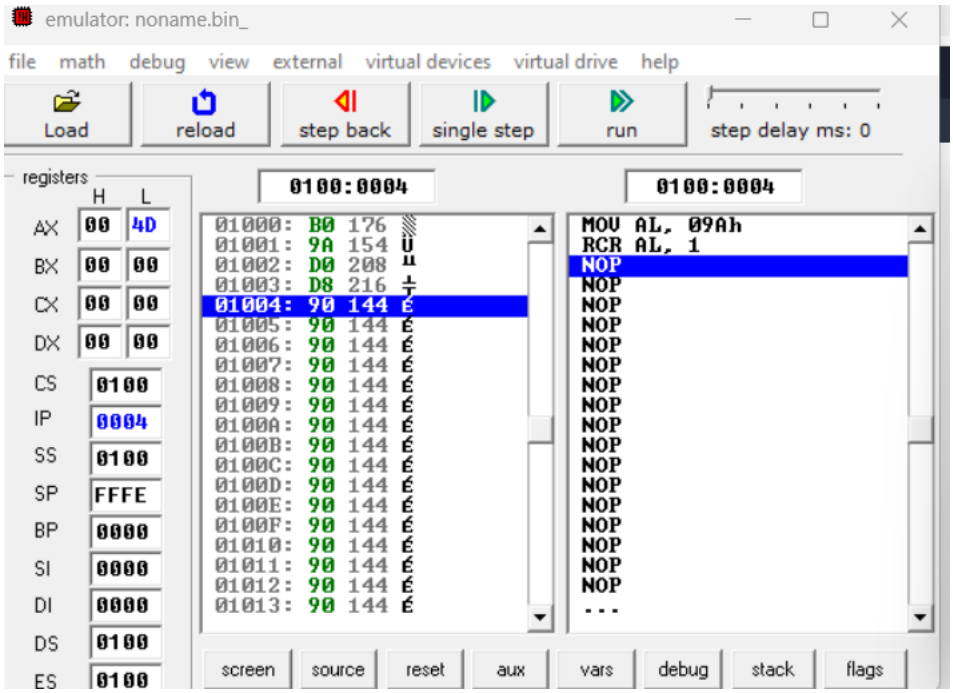
4.1.1 Đặc điểm

- Lệnh quay trái thông qua cờ nhớ
- Cú pháp : RCL ĐÍCH,CL
RCL ĐÍCH,SỐ LẦN QUAY
- Thực hiện : quay trái CL lần
- Lệnh thay đổi cờ CF, OF

4.1.2 Ví dụ

Cho AL=9Ah , CF= 0. thực hiện RCL AL,1
- Thực hiện :
AL ; 9Ah -> nhị phân 1001 1010
Dịch 1 lần : CF = 0 , AL = 0011 0100
=> AL = 34h

4.1.3 Test trên Assemblye emu8086



Test ví dụ 4.1.2

4.2 RCR - Rolate through Carry falg to the Right

4.2.1 Đặc điểm

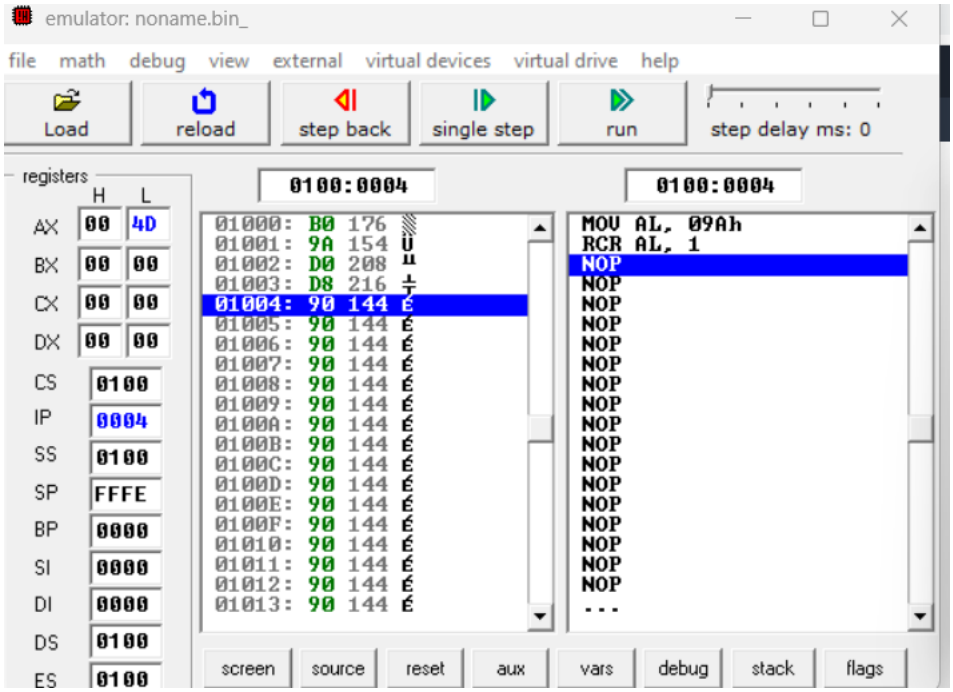
- Lệnh quay phải thông qua cờ nhớ
- Cú pháp : RCR ĐÍCH,CL
RCR ĐÍCH,SỐ LẦN QUAY
- Thực hiện : quay phải CL lần

- Lệnh thay đổi cờ CF, OF
- Tương tự RCL nhưng chiều ngược lại

4.2.2 Ví dụ

Cho AL=9Ah , CF= 0. thực hiện RCR AL,1
- Thực hiện :
AL ; 9Ah -> nhị phân 1001 1010
Dịch 1 lần : CF = 0 , AL = 0100 1101
=> AL = 34h

4.2.3 Test trên Assemblye emu8086



Test ví dụ 4.1.2

4.3 SAL - Shift Arithmetically Left

- Lệnh dịch trái số học
- Cú pháp : SAR ĐÍCH,CL
- Thực hiện : dịch trái ĐÍCH đi CL bit
- Lệnh này thay đổi cờ SF , ZF , PF , CF mang giá trị của LSB

4.4 SHL - Shift Left

Chức năng tương tự SAL

4.5 SAR - Shift Arithmetically Right

- Lệnh dịch phải số học
- Cú pháp : SAR ĐÍCH,CL
- Thực hiện : dịch phải ĐÍCH đi CL bitbit
- Lệnh này thay đổi cờ SF , ZF , PF , CF mang giá trị của LSB

4.6 SHR - Shift Right

- Lệnh dịch phải logic
- Cú pháp SHR ĐÍCH,CL
- Thực hiện : dịch phải CL bit
- Lệnh này thay đổi giá trị OF , SF , ZF , PF , CF

5 Lời kết

Vậy là đã xong cái deadline về các lệnh LOGIC sau mấy ngày rền cày =)))