



## Full bài tập tt ktmt Huỳnh Hoàng Hà

Trường Đại học Sư phạm Kỹ thuật Thành phố Hồ Chí Minh (Trường Đại học Sư phạm  
Kỹ Thuật Thành phố Hồ Chí Minh)



Scan to open on Studeersnel

## TỔNG KẾT BÀI TẬP HÀNG TUẦN

Họ và tên	
MSSV	

### Yêu cầu:

- Mỗi buổi học, tổng hợp ít nhất 50% câu hỏi/bài tập (là đạt).
- Làm mục lục, đánh số trang gọn gàng.
- Làm đúng yêu cầu, rõ ràng: đạt 30% tổng điểm môn học (3đ)

### Mục lục

[Buổi 1] Session 1 – Intro, setup, 8086 overall

[Buổi 2] ...

...

[Buổi 7] Lesson 06 – Tổng quan 8051

[Buổi 8] ...

...

[Buổi 15] Lập trình UART

(\*) Bài: lập trình ADC (không có bài tập)

## MỤC LỤC

[Buổi 1] Session 1 – Intro, setup, 8086 overall.....	3
INTRODUCTION.....	4
RUN SAMPLE.....	4
NUMBER SYSTEMS: BIN, DEC, HEX.....	5
ASCII CHARACTER.....	7
[Buổi 2] Session 2 - Register, memory and program.....	9
REGISTER AND MEMORY PARTITIONS.....	9
LOGICAL AND PHYSICAL ADDRESS.....	13
HOW TO RUN PROGRAM ?.....	15
[Buổi 3] Session 3 - Variables.....	17
VARIABLE AND MEMORY.....	17
ARRAY VARIABLE AND MEMORY.....	20
CONSTANT.....	21
[buổi 4] Session 4 - ISA, Move and Keyboard I/O Instructions.....	22
4 ISA INSTRUCTION GROUPS.....	22
MOV INSTRUCTION.....	23
KEYBOARD I/O INSTRUCTIONS.....	24
"PAUSE AND WAIT ANY KEYPRESS" INSTRUCTION.....	25
STACK MEMORY.....	25
[Buổi 5] Session 5 - Flag reg, Arithmetic instructions, loop.....	26
FLAG(STATUS) REGISTER.....	26

INSTRUCTION: ADD, SUB.....	27
INSTRUCTION: INC, DEC.....	28
INSTRUCTION: LOOP.....	29
INSTRUCTION: MUL.....	31
INSTRUCTION: DIV.....	35
[Buổi 6] Session 6 - Jump, Logic instructions.....	37
INSTRUCTION: JUMP.....	37
INSTRUCTION: CMP.....	39
CONDITIONAL JUMP INSTRUCTION.....	39
VÍ DỤ TẠO VÒNG LẶP (TỪ LỆNH NHảy CÓ ĐIỀU KIỆN).....	46
LOGIC OPERATIONS (CÁC PHÉP TOÁN LOGIC).....	46
[Buổi 7] Bài 06 – Tổng quan dòng vi điều khiển 8051.....	48
[Buổi 8] Bài 07 - Tập lệnh 8051.....	51
[Buổi 9] Bài 08 - Nhập xuất GPIO.....	54
[Buổi 10] Bài 09 - Timer/Counter.....	57
[Buổi 11] Bài 10 - Ngắt.....	60
[Buổi 12] Bài 11 - Lập trình GPIO sử dụng ngôn ngữ C.....	63
[Buổi 13] Bài 12 - Lập trình Led 7 đoạn.....	65
[Buổi 14] Bài 13 - Lập trình UART.....	68

## [Buổi 1] Session 1 – Intro, setup, 8086 overall

### INTRODUCTION

**1. 8086 là vi điều khiển(MCU-microcontroller) hay vi xử lý(CPU/MPU-Microprocessor)?**

- 8086 là 1 con vi xử lý (CPU) .

**2. Kiến trúc 8086 gồm 3 phần chính gì?**

- ALU – Arithmetic & Logic Unit.
- CU – Control Unit.
- Registers.

**3. Thanh ghi thực tế là gì? Chức năng của nó là gì?**

- Thanh ghi là những bit lưu trữ nhanh – nơi CPU lưu trữ các giá trị ngay lúc đó trong quá trình tính toán và thực thi chương trình.

- Chức năng : + theo dõi chu kỳ lệnh.
  - + Tải lệnh từ bộ nhớ.
  - + tích tủy data.

**4. 8086 có thể chạy 1 mình không? Tại sao? Và làm sao để 8086 có thể chạy được?**

- không vì nó cần phải giao tiếp với ram(nơi chứa chương trình và dữ liệu), bộ nhớ bên ngoài .

\*nạp dữ liệu từ RAM thông qua systembus thì CPU(8086) sẽ xử lý các phép toán và data.

**5. 8086 là chip xử lý bao nhiêu bit ?**

- CPU 8086 là vi xử lý 16 bit.

**6. Trong 1 lần tính, thì Bộ cộng, trừ, nhân, chia của 8086 có thể thực hiện giữa 2 số lớn nhất là bao nhiêu?**

- 8086 có thể xử lý số lớn nhất là **65.535** (16-bit) đối với cộng/trừ, đối với nhân/chia thì có thể xử lý số 32-bit (4.294.967.295) làm số bị chia hoặc kết quả trung gian.

**7. Chức năng của 8086 là gì? (xem video liên kè: “The Fetch-Execute Cycle: What's Your Computer Actually Doing?”)**

- Xử lý các phép toán và dữ liệu, quản lý bộ nhớ và giao tiếp I/O, đọc và thực thi lệnh từ bộ nhớ .

### RUN SAMPLE

**1. Install software (extract the emu8086\_Tutorials.rar, and run emu8086.exe)**

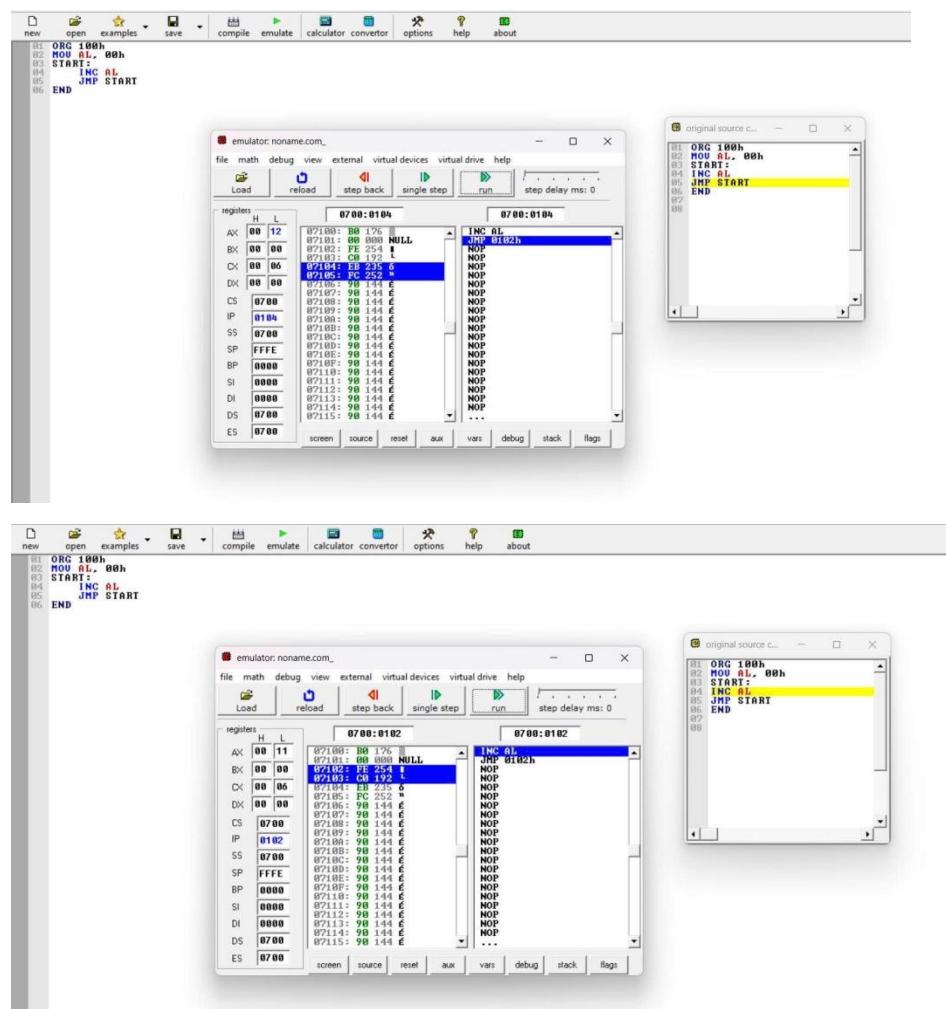
**2. Run sample code below -> run (emulate) -> run single step (many times). Analyze what happen and anything else?**

```
ORG 100h
MOV AL, 00h
START:
    INC AL
    JMP START
END
```

### Trả Lời

- Mỗi nhần nhán single step thì địa chỉ IP thay đổi giữa dòng lệnh “INC AL” và dòng lệnh “JMP START” (nhảy về start), tiến hành vòng lặp.

- Giá trị của bộ nhớ AL tăng lên 1 sau mỗi vòng lặp.



### NUMBER SYSTEMS: BIN, DEC, HEX

### 1. Cho chương trình:

**MOV AX, 1101b**

**MOV BX, 0Dh**

**MOV CX, 13**

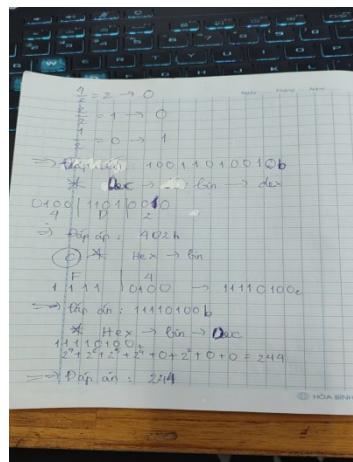
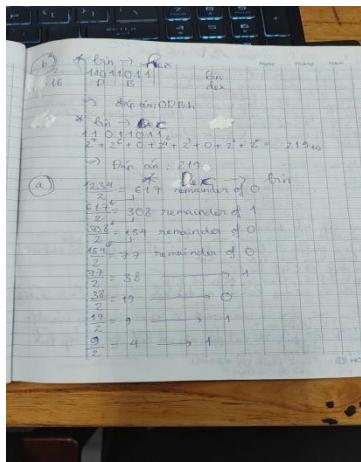
- **Hỏi: Kết quả trong 3 thanh ghi là như nhau hay khác nhau?**

- Kết quả trong 3 thanh ghi là giống nhau (0DH).

### 2. Chuyển đổi qua lại giữa các hệ số:

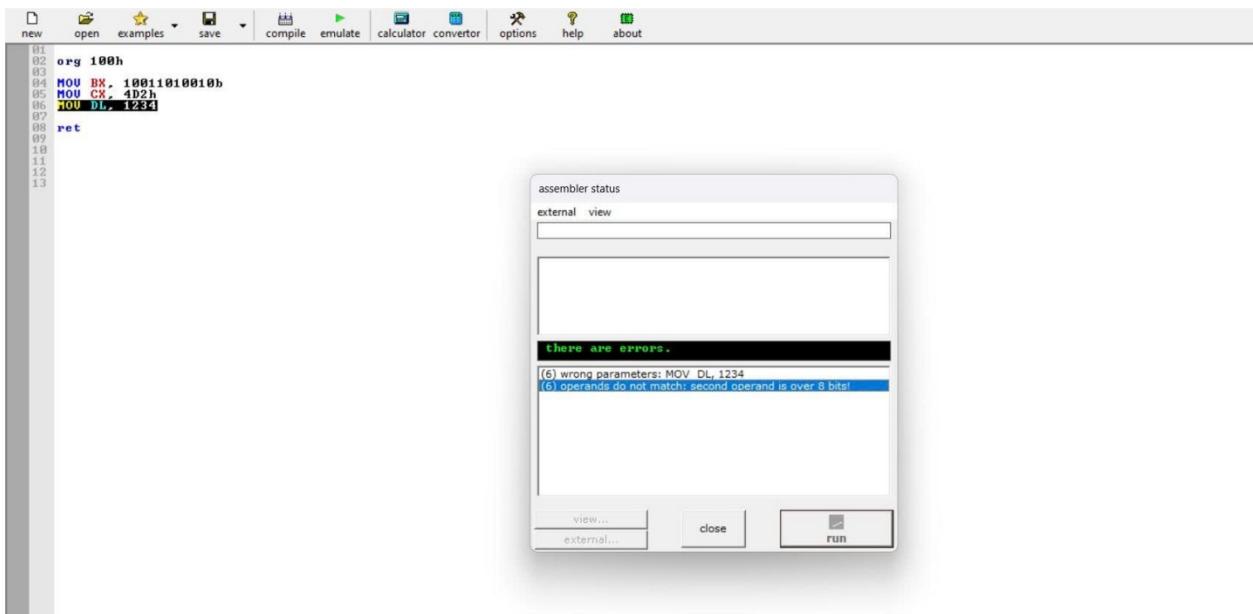
(Nêu cách tính toán, và tính bằng tay – Tính ra giấy và chụp hình lại kết quả nếu cần)

- Chuyển số 1234 (Thập phân – Decimal) sang 2 hệ số còn lại.**
- Chuyển số 11011011b (Nhị phân – Binary) sang 2 hệ số còn lại.**
- Chuyển số 0F4h (Thập lục phân – Hexadecimal) sang 2 hệ số còn lại.**

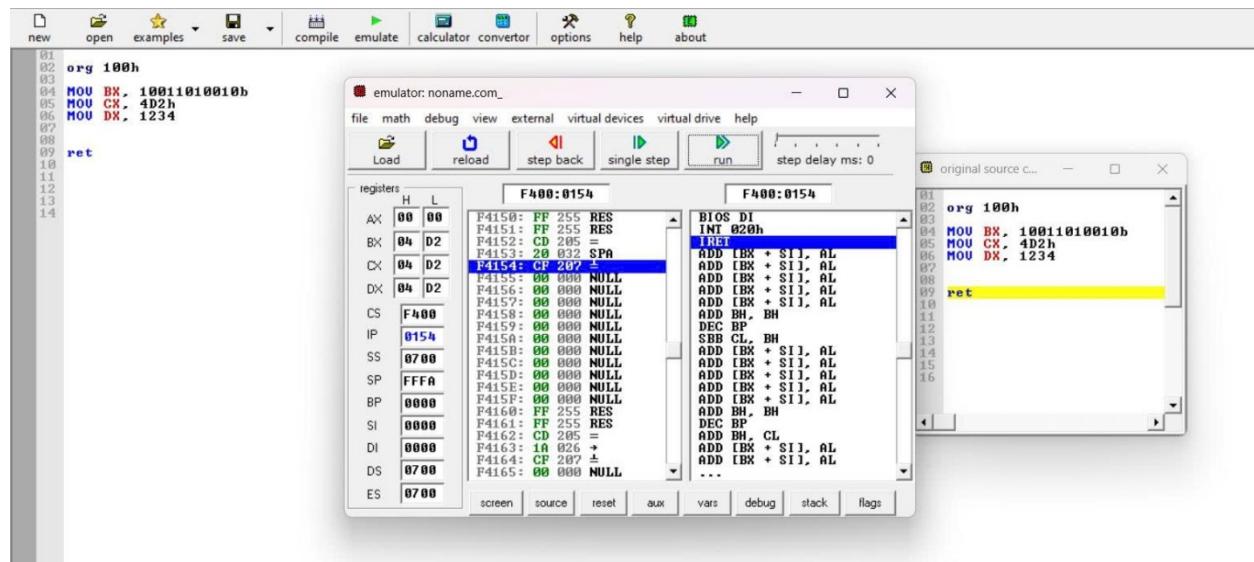


### 3. Viết chương trình nhập (MOV) giá trị 1234 vào các thanh ghi BX (dạng nhị phân), CX (dạng hexa), DL (dạng thập phân). Nếu chương trình ổn thì khởi giải thích, còn nếu không ổn thì tại sao và sửa lại cho ổn?

- **Chạy kết quả của chương trình trên, và xem giá trị trong các thanh ghi (trong emulator) xem có đúng không? (Chụp kết quả các thanh ghi)**



- Vì thanh ghi DL chỉ chứa được 8 bit trong khi dữ liệu vào là 11 bit gây ra lỗi khi chạy chương trình.



- Bạn sửa lại sau khi thay thanh ghi DL bằng thanh ghi DX có kích thước lớn hơn.

## ASCII CHARACTER

**1. VĐK, VXL có hiểu được kí tự ‘a’, ‘1’, ‘#’ không? (Gợi ý: Không, vì đây là cái kí tự trong ngôn ngữ của con người, máy móc không hiểu được). Vậy làm cách nào để Máy móc phân biệt được các kí tự này với nhau? (Trả lời câu số 2 là sẽ hiểu)**

- Không, vì đây là cái kí tự trong ngôn ngữ của con người, máy móc không hiểu được.

- Máy tính phân biệt các ký tự thông qua giá trị số khác nhau được gán cho mỗi ký tự trong các bảng mã hóa chuẩn.

## 2. Bản chất, khi các ký tự được lưu vào bộ nhớ, thì sẽ lưu dưới dạng gì?

- Tất cả ký tự đều được chuyển thành số nguyên .

- Số nguyên này được lưu dưới dạng nhị phân (0 và 1) .

- Mỗi ký tự chiếm một số byte nhất định tùy theo chuẩn mã hóa.

## 3. Lập bảng, xác định giá trị số (2 dạng: Decimal, Hexa) tương ứng với các ký tự sau: ‘a’, ‘D’, ‘#’, ‘1’, ‘0’, ‘9’.

DEC	HEX	ASCII
97	61	a
68	44	D
35	23	#
49	31	1
48	30	0
57	39	9

## 4. Chương trình ASM (8086) có lệnh giúp nhập 1 số “trực tiếp” từ bàn phím không? (Gợi ý: Không nha, chỉ có lệnh nhập vào 1 ký tự)

- Không, chỉ có lệnh nhập vào 1 ký tự

## 5. Vậy, muốn nhập 1 ký tự từ bàn phím thì mình làm sao? (Gợi ý: Học sau, Dùng INT21h/AH=1, học trong video về “Nhập/Xuất ký tự từ bàn phím” nha.)

- Học sau

## 6. Chuyển ký tự số thành số thì làm sao?

- Để chuyển ký tự số thành số, cần trừ đi giá trị ASCII của ký tự '0'.

Ví dụ : ah = ‘1’, ah = ah – 30h = 1

## 7. Muốn nhập 1 số có nhiều chữ số từ bàn phím thì phải làm sao? (Trả lời lý thuyết, từng bước làm là gì)

\*ý tưởng

- chuyển ký tự số thành số, trừ đi giá trị ASCII của ký tự '0'.

- Sử dụng thuật toán: số\_mới = số\_cũ × 10 + chữ\_số\_hiện\_tại

- Nhập ký tự “0” từ bàn phím rồi – 30h ta có số 0 (số cũ)

Ví dụ: nhập "123"

- Bước 1:  $0 \times 10 + 1 = 1$
- Bước 2:  $1 \times 10 + 2 = 12$
- Bước 3:  $12 \times 10 + 3 = 123$

**8. Chương trình ASM (8086) có lệnh giúp xuất 1 số (có 1 chữ số) “trực tiếp” ra màn hình không? (Gợi ý: Không nha, chỉ có lệnh xuất vào 1 kí tự)**

- Không, chỉ có lệnh xuất vào 1 kí tự

**9. Vậy, muốn xuất 1 kí tự ra màn hình thì làm sao? (Gợi ý: Học sau, dùng INT21h/AH=2, học trong video về “Nhập/Xuất kí tự từ bàn phím” nha.)**

- học sau

**10. Chuyển 1 số (có 1 chữ số) thành kí tự số thì làm sao?**

- cộng giá trị ASCII của ký tự '0'.

**11. Muốn xuất 1 số có nhiều chữ số ra màn hình thì phải làm sao? (Trả lời lý thuyết, từng bước làm là gì)**

\*Ý tưởng

- B1: Sử dụng phép chia cho 10 để tách từng chữ số từ phải sang trái

Ví dụ: 1234

$$123 \div 10 = 12 \text{ dư } 3 \rightarrow \text{chữ số } 3$$

$$12 \div 10 = 1 \text{ dư } 2 \rightarrow \text{chữ số } 2$$

$$1 \div 10 = 0 \text{ dư } 1 \rightarrow \text{chữ số } 1$$

- B2 : Chuyển đổi chữ số sang ASCII

Ví dụ: chữ số 5  $\rightarrow 5 + 30H = 35H = '5'$

## [Buổi 2] Session 2 - Register, memory and program

### REGISTER AND MEMORY PARTITIONS

**1. Trong môn học này, chúng ta quan tâm đến bao nhiêu thanh ghi? Liệt kê các thanh ghi và chức năng của nó?**

\*\*\*14 thanh ghi\*\*\*

- a. 8 thanh ghi dùng cho mục đích chung
  - Thanh ghi 16-bit và các phần 8-bit

AX - Accumulator Register (AH + AL)

BX - Base Address Register (BH + BL)

CX - Count Register (CH + CL)

DX - Data Register (DH + DL)

- Thanh ghi chỉ số và con trỏ:

SI - Source Index Register

DI - Destination Index Register

BP - Base Pointer

SP - Stack Pointer

- Lưu ý: 8 thanh ghi này có thể dùng để chứa các giá trị số tùy ý trong các phép toán.
- b. 4 thanh ghi phân đoạn (Segment Registers):

CS - Code Segment (chứa địa chỉ đoạn code)

DS - Data Segment (chứa địa chỉ đoạn dữ liệu/biến)

ES - Extra Segment (thanh ghi phụ cho lập trình)

SS - Stack Segment (chứa địa chỉ vùng nhớ Stack)

- Lưu ý: Không sử dụng các thanh ghi này chứa dữ liệu thông thường.
- c. 2 thanh ghi có chức năng đặc biệt:

IP (PC) - Instruction Pointer (kết hợp với CS để xác định địa chỉ lệnh chuẩn bị thực thi)

Flag Register - Thanh ghi cờ trạng thái (tự động thay đổi sau các phép toán, dùng để "nhảy" chương trình)

- Lưu ý: Không sử dụng các thanh ghi này chứa dữ liệu thông thường.

## 2. Thanh ghi nằm trong đâu?

- Thanh ghi nằm trong vxl 8086 (CPU).

## 3. Kích thước của thanh ghi là bao nhiêu? (2 loại kích thước). Có thể sử dụng 1 thanh ghi để chứa 2 số khác nhau không?

- 2 kích thước là 8 bit và 16 bit, có thể sử dụng 1 thanh ghi để chứa 2 số khác nha.

## 4. Hỏi DX bằng nhiêu, sau khi đoạn chương trình sau? Org 100h MOV DL, 50 MOV DH, 220 RET

- DC32h.

## 5. Lệnh, biến được nạp và lưu vào đâu? (Gợi ý: Mỗi lệnh, mỗi biến sẽ được lưu vào 1 hoặc vài ô nhớ, tùy vào kích thước giải mã của lệnh, hay kích thước của biến)

- Mỗi lệnh, mỗi biến sẽ được lưu vào 1 hoặc vài ô nhớ, tùy vào kích thước giải mã của lệnh, hay kích thước của biến.

## 6. Hệ thống dựa vào gì để truy cập vào 1 ô nhớ bất kỳ? (Gợi ý: Mỗi ô nhớ có 1 địa chỉ)

- Hệ thống dựa vào địa chỉ của mỗi ô nhớ để truy cập vào 1 ô nhớ bất kỳ.

**7. Địa chỉ của ô nhớ có mấy dạng hiển thị? Cho địa chỉ ô nhớ là 07213h, đây là địa chỉ gì, và bạn hãy chuyển sang dạng hiển thị còn lại?**

- Có 2 dạng hiển thị : địa chỉ vật lý và địa chỉ logical (dạng segment:offset)

⇒ 07213h là địa chỉ vật lý

Công thức :

**Physical Segment \* 10h + Offset**

segment	offset	tính
0700h	xh	$0700h \times 10h + xh = 07213h \Rightarrow x = 213$

Tính toán :

⇒ Địa chỉ logical : 0700h:0213h

**8. Cho 2 địa chỉ: 0700h:151h, 0700h:168h. Hỏi 2 địa chỉ này có phần giống nhau gọi là gì, phần khác nhau gọi là gì?**

- Phần khác nhau gọi là offset.

**9. Vùng bộ nhớ có thể sử dụng để chứa chương trình, dữ liệu chương trình là từ đâu đến đâu?**

- 00500h – A0000h.

**10. Trong chương trình mặc định (trong video),**

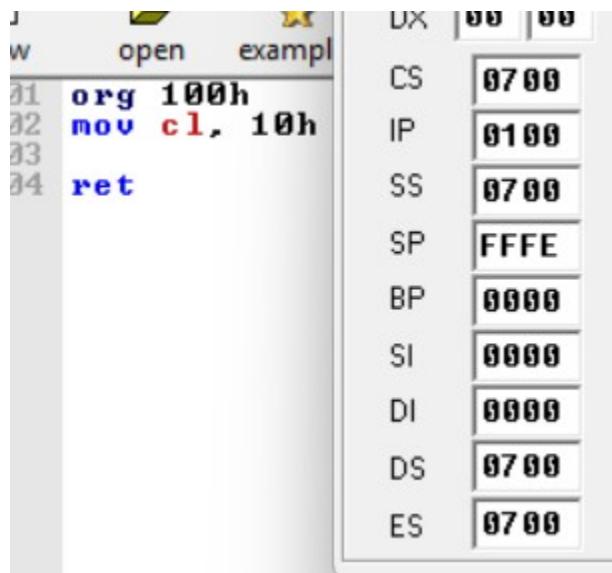
**a. Chương trình, dữ liệu, stack được lưu trong 3 vùng bộ nhớ khác nhau hay giống nhau, và bắt đầu từ bao nhiêu? Thanh ghi nào quy định(chứa) thông tin đó? Làm cách nào bạn có thể xem được thông tin đó (chụp hình ảnh).**

**b. Địa chỉ offset bắt đầu của vùng chứa chương trình, dữ liệu, stack là bao nhiêu?**

**c. Địa chỉ đầy đủ (2 dạng) của ô nhớ đầu tiên chứa chương trình, dữ liệu, stack là bao nhiêu?**

a)

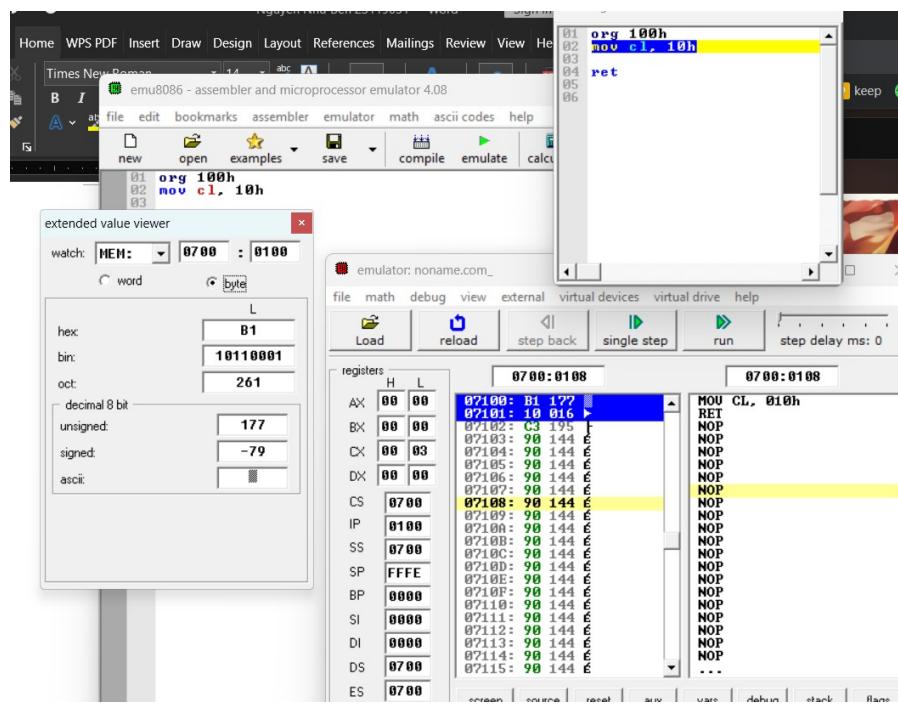
- Cùng 1 vùng bộ nhớ 0700h đến 0A0000h, bắt đầu từ 0700h
- Thanh ghi quy định vị trí Code, Data, Stack (CS,DS,SS)



- b) 100h
- c) 0700h:0100h

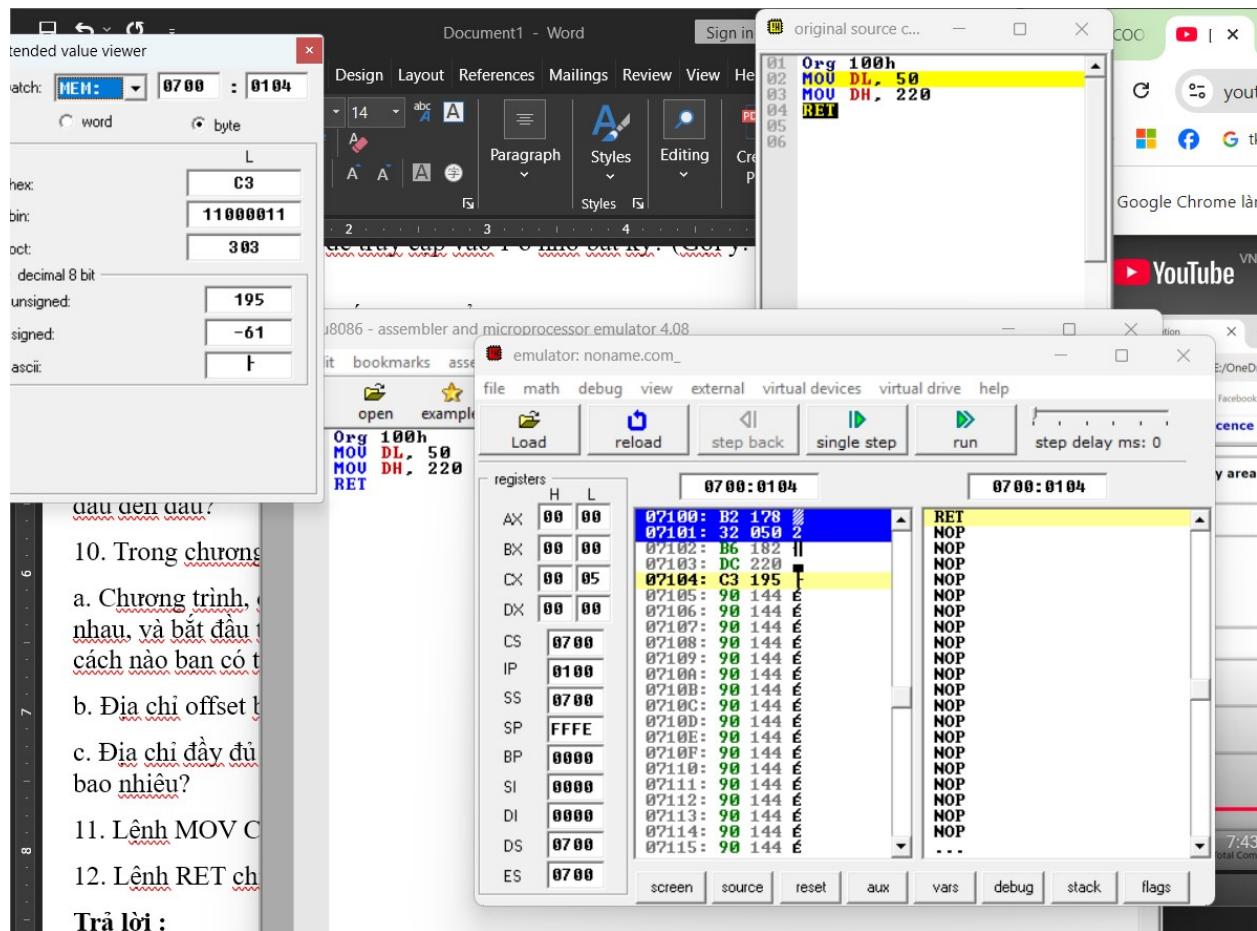
### 11. Lệnh MOV CL, 10h chiếm bao nhiêu byte? Chụp hình minh chứng.

- Chiếm 2 byte.



### 12. Lệnh RET chiếm bao nhiêu byte? Chụp hình minh chứng.

- Chiếm 1 byte



## LOGICAL AND PHYSICAL ADDRESS

### 1. Chương trình, dữ liệu được lưu vào đâu?

- lưu vào trong vùng nhớ, mỗi giá trị, mã lệnh thì được lưu vào 1 ô nhớ cụ thể có địa chỉ riêng

### 2. Địa chỉ của ô nhớ (1 byte) (trong bộ nhớ) có mấy dạng thể hiện?

- 2 dạng: địa chỉ logic, địa chỉ vật lý

### 3. Cho địa chỉ 0700h:105h. Đây là địa chỉ gì? Segment là bao nhiêu? Offset là bao nhiêu? Địa chỉ vật lý tương ứng bằng bao nhiêu?

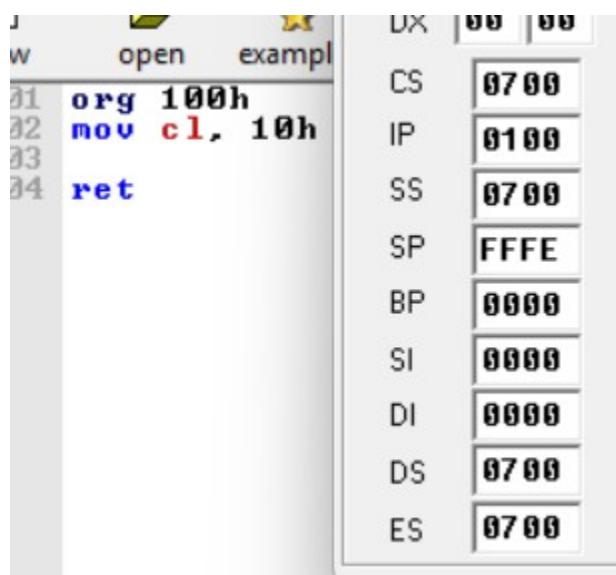
- đây là địa chỉ logical, segment là 0700h, offset là 105h, địa chỉ vật lý tương ứng = 700h x 10h + 105h = 7105h

### 4. Tất cả câu lệnh chương trình đều có chung địa chỉ Segment là bao nhiêu?

- 700h

### 5. Địa chỉ segment của: vùng chứa chương trình, vùng chứa dữ liệu, vùng chứa Stack được lưu trong 3 thanh ghi nào? Có giá trị mặc định là bao nhiêu? (Địa chỉ segment là cố định từ khi biên dịch chương trình). Chụp hình trong Emulator để chứng minh?

- được lưu trong 3 thanh ghi CS, DS, SS, có giá trị mặc định là 700h

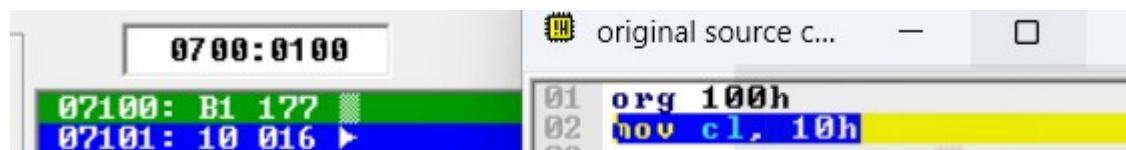


#### 6. Địa chỉ các câu lệnh chỉ khác nhau ở phần gì?

- chỉ khác nhau ở phần offset

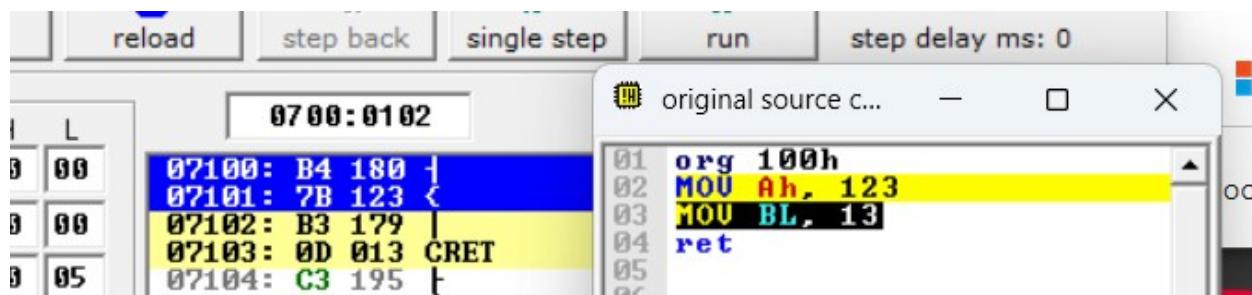
#### 7. Dòng lệnh đầu tiên của chương trình (dòng lệnh nằm sau ORG 100h) có địa chỉ logic, địa chỉ vật lý bằng bao nhiêu? Chụp hình thể hiện ở tab Emulator để chứng minh?

- có địa chỉ vật lý là 7100h và địa chỉ logic là 0700h:0100h



#### 8. Trong video, lệnh MOV BL, 13 được lưu vào mấy ô nhớ (chiếm bao nhiêu byte)? Có địa chỉ bao nhiêu? Mã lệnh (Machine code, dạng hexa) là gì? Chụp hình Emulator để chứng minh?

- chiếm 2 ô nhớ (2 bytes), có địa chỉ vật lý lần lượt là 07102h và 07103h với mã lệnh lần lượt là B3,0D



#### 9. Bạn viết lại chương trình trong video, và thêm 1 lệnh MOV BL, 14 vào sau lệnh trên, chạy kết quả và xem mã lệnh (Machine code) là gì? Chụp hình Emulator để chứng minh?

- mã lệnh lần lượt là B3,0E

```

$ H L
07100: B4 180
07101: 7B 123
07102: B3 179
07103: 0D 013 CRET
07104: B3 179
07105: 0E 014
07106: C3 195
07107: 90 144
07108: 90 144

```

```

original source c...
01 org 100h
02 MOU Ah, 123
03 MOU BL, 13
04 MOU BL, 14
05 ret
06
07

```

**10.** Bạn thêm 1 lệnh MOV BX, 14 vào sau lệnh trên, chạy kết quả và xem mã lệnh (Machine code) là gì? Lệnh này chiếm bao nhiêu byte? Chụp hình Emulator để chứng minh?

- mã lệnh lần lượt BB,0E,00, chiếm 3 bytes

```

$ H L
07100: B4 180
07101: 7B 123
07102: B3 179
07103: 0D 013 CRET
07104: B3 179
07105: 0E 014
07106: BB 187
07107: 0E 014
07108: 00 000 NULL
07109: C3 195
0710A: 90 144
0710B: 90 144

```

```

original source c...
01 org 100h
02 MOU Ah, 123
03 MOU BL, 13
04 MOU BL, 14
05 MOU BX, 14
06 ret
07
08

```

### HOW TO RUN PROGRAM ?

**1. Cấu trúc cơ bản của chương trình gồm có gì? Giải thích ý nghĩa của từng dòng lệnh?**

- gồm 3 phần : địa chỉ offset bắt đầu, chương trình và return

+ org 100h : chỉ định địa chỉ bắt đầu của chương trình là 100h

+ chương trình, code

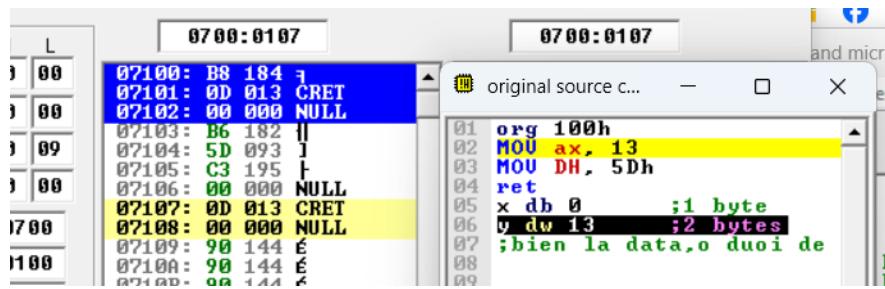
+ ret : lệnh trả về từ một hàm/chương trình con

**2. Vị trí biến khai báo nên để ở đâu?**

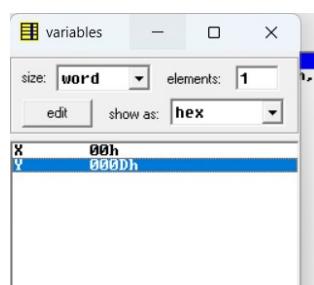
- Được thẻ ở cuối (sau ret).

**3. Có bao nhiêu cách xem kết quả của biến? Chụp hình minh chứng để xem biến y theo các cách bạn biết?**

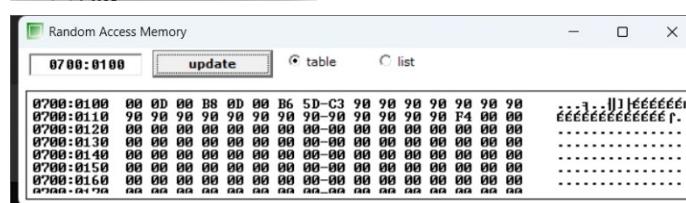
- Có 3 cách



view-> variables



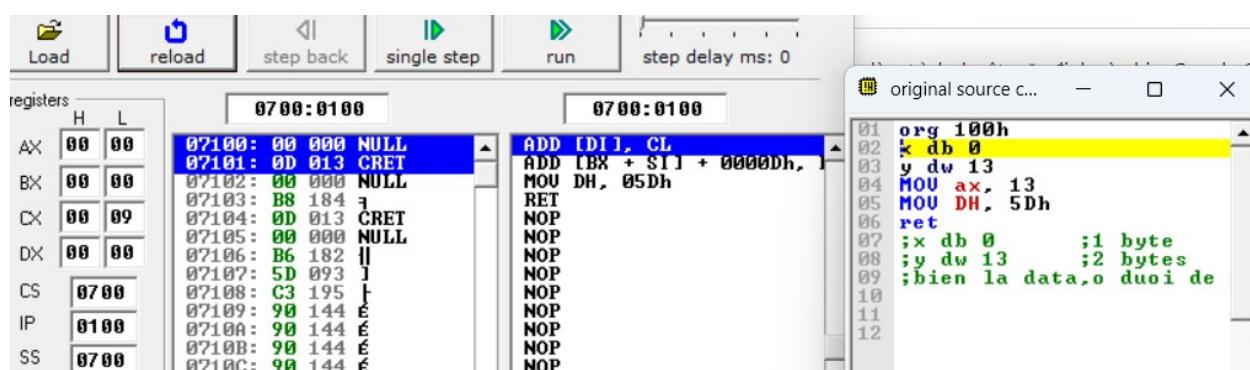
view->memory

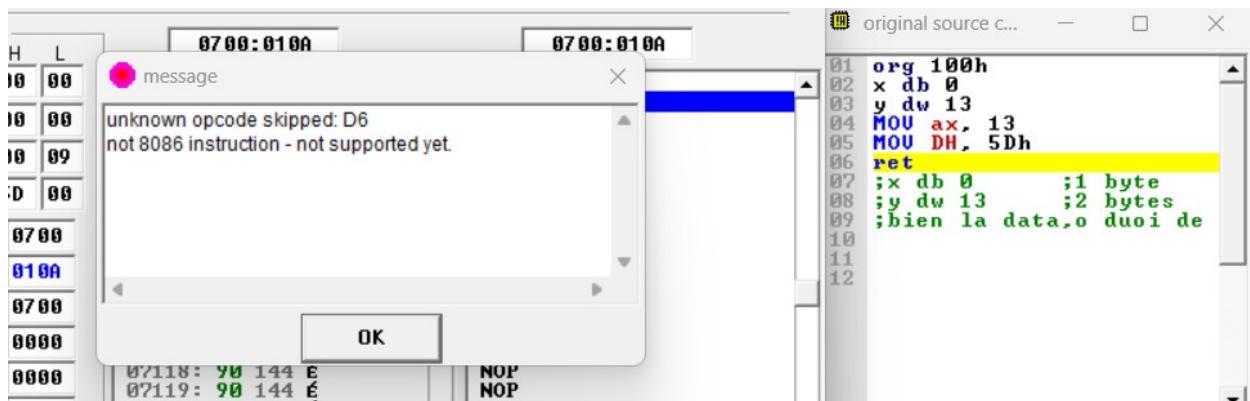


#### chương trình và phân tích kết quả xem?

- câu lệnh biến x dung db chỉ chứa 1 byte bị chuyển sang 2 byte, câu lệnh biến y thì từ 2 sang 4 byte, câu lệnh mov ax, 13 bị mất ô nhớ, không thể return chương trình

**4. Viết lại chương trình trong video, và di chuyển 2 lệnh khai báo biến x,y lên ngay phía sau ORG 100h (khai báo biến nằm trước các dòng lệnh MOV). Vậy chuyện gì sẽ xảy ra, bạn hãy chạy**





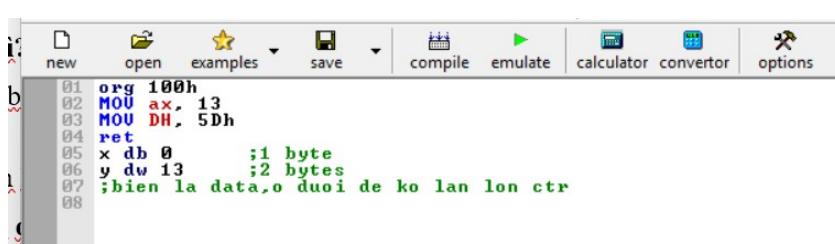
### 5. Thanh ghi IP chứa gì? Chức năng của thanh ghi IP là gì?

- instruction pointer (con trỏ lệnh) chứa địa chỉ của lệnh chuẩn bị được thực hiện, được nạp vào CPU

- chức năng : truy vết , theo dõi để xem chương trình chạy đến lệnh nào rồi

### 6. Trong video, khi lệnh MOV DH, 5Dh vừa thực hiện, thì dòng lệnh nào sẽ được highlight vàng, giá trị trong thanh ghi IP bằng bao nhiêu, lệnh chuẩn bị được nạp vào CPU để giải mã và thực hiện là lệnh nào?

- Dòng lệnh sẽ được highlight vàng là ret, giá trị IP là 105 , lệnh chuẩn bị được nạp là lệnh ret



code bài ví dụ chính

## [Buổi 3] Session 3 - Variables

### VARIABLE AND MEMORY

#### 1. Giá trị của biến được lưu trong vùng bộ nhớ chương trình hay bộ nhớ dữ liệu?

- được lưu trong bộ nhớ dữ liệu, còn bộ nhớ chương trình chứa các lệnh

#### 2. Chương trình có hiểu được tên biến không? Tên biến có lưu trong bộ nhớ không?

- không hiểu được tên biến mà chỉ hiểu địa chỉ ô nhớ của biến, tên biến chỉ là nhãn cho người hiểu

3. Địa chỉ segment của vùng bộ nhớ dữ liệu là bao nhiêu? Được lưu trong thanh ghi gì?

- 700h, được lưu trong thanh ghi DS

4. Theo chương trình trong video, câu lệnh và biến đều có địa chỉ segment là như nhau. Đúng hay sai? Giải thích và đưa hình ảnh để minh chứng?

- đúng, vì nó biến nằm trong bộ nhớ dữ liệu nên có cùng segment bắt đầu là 700h

The screenshot shows two windows. The left window is a memory dump viewer with two panes. The top pane shows memory starting at address 0700:0109, with the first byte being A0 (nop). The bottom pane shows memory starting at address 0700:0109, with the first byte being XOR AL, 012h. The right window is a source code editor titled "original source c...". It contains assembly code:

```
01 org 100h
02 mov al, var1
03 mov bx, var2
04 ret
05 var1 db ?
06 var2 dw 1234h
07
08
```

5. Có mấy cách để khai báo biến x có giá trị là 123. Viết ra câu lệnh đó?

- Có 3 cách : bin,hex,dec

+ var3 db 123

+ var3 db 7Bh

+ var3 db 01111011b

6. Theo video, lệnh MOV AX, var1 bị báo lỗi, lỗi này là gì?

- Báo lỗi vì kích thước biến phải tương ứng kích thước thanh ghi (địa chỉ biến có 8 bit không tương thích thanh ghi AX 16 bit)

7. Vị trí khai báo biến nên để ở đâu?

- Để ở cuối chương trình, cụ thể là sau RET

8. Biến var1, var2 chiếm bao nhiêu byte?

- var1,var2 lần lượt chiếm 1 byte và 2 byte

9. Mã lệnh(machine code) của lệnh RET là gì? Chiếm bao nhiêu byte?

- mã lệnh của ret là C3, chiếm 1 byte

10. Địa chỉ logic, địa chỉ vật lý của biến var1 là bao nhiêu?

- Biến var1 có địa chỉ vật lý và logic lần lượt là 07108h và 0700h:0108h

11. Địa chỉ logic, địa chỉ vật lý của biến var2 là bao nhiêu?

- Biến var2 có địa chỉ vật lý và logic lần lượt là 07109h và 0700h:0109h

12. Viết 1 câu lệnh Copy dữ liệu từ 1 ô nhớ (thông qua địa chỉ của ô nhớ) vào 1 thanh ghi?

org 100h

mov BL, var3 ;câu lệnh copy data từ 1 ô nhớ(có địa chỉ riêng)

ret

var3 db 7Bh ;khi khai báo biến thì nó lưu vô ô nhớ(kèm địa chỉ theo ô)

### 13. Phương thức lưu số lớn (lớn hơn 1 byte) vào bộ nhớ là BigEndian hay LittleEndian?

- Đó là phương thức little endian

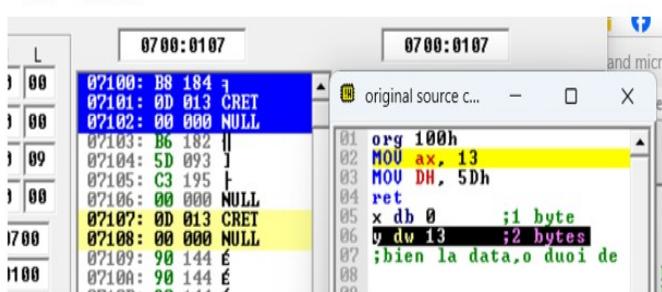
### 14. Cho số 1122h lưu vào ô nhớ bắt đầu địa chỉ là 07120h, thì số này sẽ lưu vào ô nhớ nào với giá trị là bao nhiêu?

- Ô nhớ 07120h: giá trị 22h

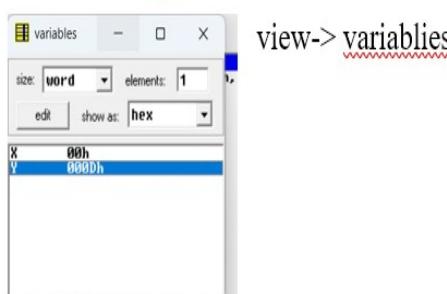
- Ô nhớ 07121h: giá trị 11h

### 15. Lúc chạy chương trình, có những cách nào để xem giá trị của biến trong bộ nhớ? (Gợi ý: Emulator, Emulator/view/Memory, Emulator/view/variables). Chụp hình theo 3 cách để chứng minh (chụp đúng chỗ cần chụp).

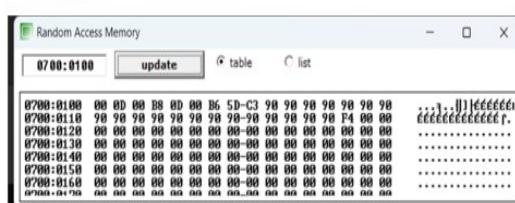
- Có 3 cách



Xem trực tiếp trong bảng emulator



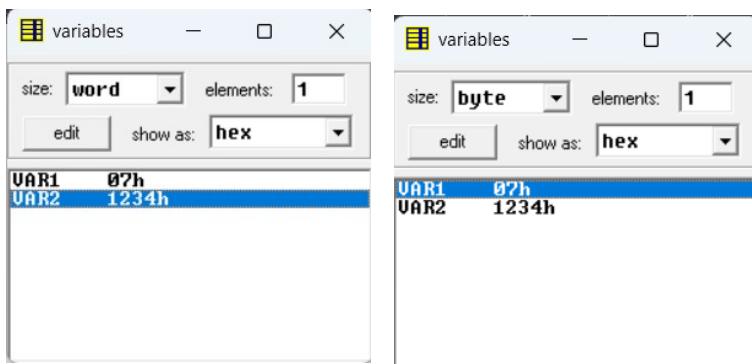
view->variables



view->memory

### 16. Giải thích các thông số trong cửa sổ

“Emulator/view/variables”?



hình 1 : 1234h là giá trị dạng hex, kích thước size là word vì giá trị chiếm 2 byte, elements : 1 là có 1 phần tử, biến

hình 2 : 07h là giá trị dạng hex, kích thước size là byte vì giá trị chiếm 1 byte, elements : 1 là có 1 phần tử, biến

### ARRAY VARIABLE AND MEMORY

**1. Khai báo 1 biến mảng X, chứa 5 số 16bit, ban đầu bằng 0.**

- X dw 5dup(0,1234h,0,0,0)

**2. Trong 8086 (ASM), có dạng chuỗi không? (Gợi ý: Không có nha.)**

- Không

**3. Vậy “abcd” là gì? Ghi ‘abcd’ có lỗi không và có khác gì so với “abcd” không?**

- Không

**4. Chạy lại Code mẫu trong video, khai báo 4 biến a,b,c,d. Bạn hãy mở cửa sổ Variable (Trong Emulator/view/variable) để xem kết quả của 4 biến này, xem giống nhau không?**

- Không

**5. Viết chương trình: khai báo 1 biến mảng X, chứa 5 số 16bit, ban đầu bằng 0. Nhập giá trị 1,2,3,4,5 cho 5 phần tử này. Yêu cầu: dùng chỉ số của mảng là 1 con số, vd: X[1]**

```

ORG 100h
x dw 5 dup(0)

mov si, 0
mov x[si], 1

mov si, 2
mov x[si], 2

mov si, 4
mov x[si], 3

mov si, 6
mov x[si], 4

mov si, 8
mov x[si], 5

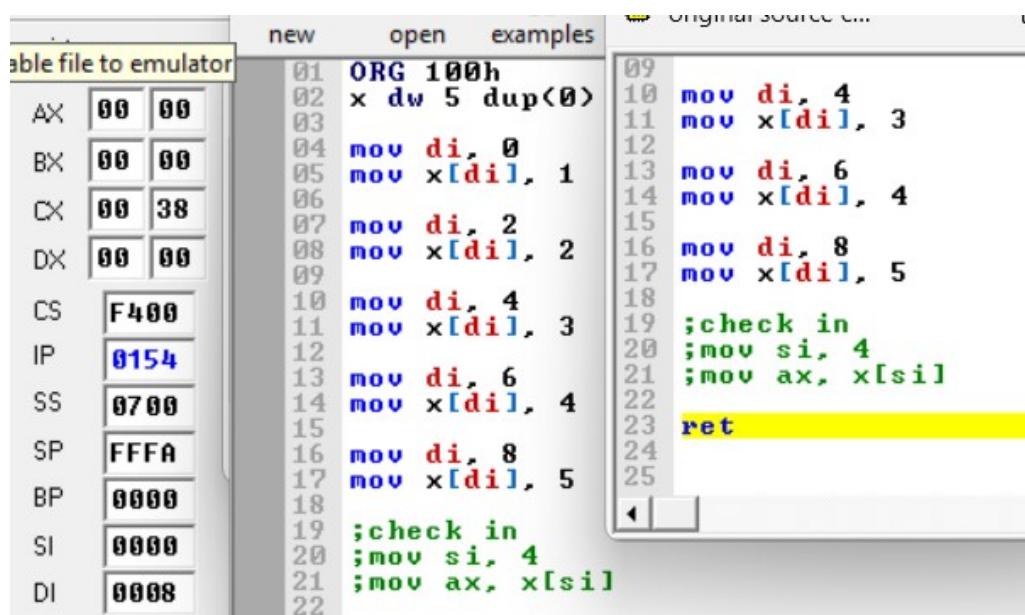
;check in
;mov si, 4
;mov ax, x[si]

ret

```

6. Các thanh ghi nào có thể sử dụng để chứa chỉ số để truy xuất đến các phần tử của mảng?  
Bạn hãy viết lại chương trình câu 5, dùng 1 thanh ghi để chứa chỉ số mảng.

- 4 thanh ghi hỗ trợ : BX, SI, DI, BP



### CONSTANT

1. Vị trí khai báo hằng số? (Gợi ý: tốt nhất là để như biến, phía sau lệnh RET)

- có thể đặt trong chương trình nhưng tốt nhất là để như biến, phía sau lệnh RET.

2. Khai báo 1 hằng số chứa giá trị là 150.

ORG 100H

A EQU 150

Mov ah, a

ret

### 3. Khác biệt giữa khai báo 1 hàng số và khai báo 1 biến chứa cùng 1 giá trị là gì?

- Đối với khai báo 1 hàng số là lấy con số đưa thẳng vào thanh ghi thông qua 1 biến.
- Đối với khai báo 1 biến là thanh ghi nó truy xuất đến ô nhớ có địa chỉ của biến .

### [buổi 4] Session 4 - ISA, Move and Keyboard I/O Instructions

#### 4 ISA INSTRUCTION GROUPS

##### Câu hỏi

##### 1. Tập lệnh ASM của 8086 chia làm mấy nhóm lệnh? Ý nghĩa của từng nhóm lệnh?

- Được chia làm 4 nhóm :

Nhóm 1: Lệnh di chuyển

Nhóm 2: Lệnh toán học

Nhóm 3: Lệnh rẽ nhánh

Nhóm 4: Lệnh toán logic trên bit

##### 2. Lập bảng mô tả ý nghĩa của từng lệnh (mỗi lệnh mô tả từ 1-2 dòng là được).

Lệnh di chuyển		Lệnh toán học		Lệnh rẽ nhánh		Lệnh toán logic trên bit	
MOV	di chuyển dữ liệu từ nguồn → đích.	ADD	cộng	JMP	Nhảy ko điều kiện	AND, OR, XOR, NOT	Toán tử logic
PUSH , POP	thao tác với stack (lưu/khôi phục giá trị).	SUB	Trừ	CMP	So sánh	SHL, SHR	dịch bit sang trái/phải
		MUL	Nhân	JE/JZ	Nhảy nếu : bằng nhau/bằng 0	ROL, ROR	xoay vòng bit sang trái/phải
		DIV	Chia	JNE/JNC	Nhảy nếu : không bằng nhau/bằng 0		
		INC	Tăng 1	JA,JAE,JB,JBE	Nhảy có điều kiện		

		DEC	Giảm 1				
--	--	-----	--------	--	--	--	--

## MOV INSTRUCTION

Câu hỏi :

### 1. Có bao nhiêu loại toán hạng(operand)?

- Có 4 loại toán hạng :

**REG**(Thanh ghi đa dụng): AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.

**SREG**(Thanh ghi Segment): DS, ES, SS, and only as second operand: CS.

**memory**(Ô nhớ & Biến): [BX], [BX+SI+7], variable, etc...(see [Memory Access](#)).

**immediate**(Hằng số): 5, -24, 3Fh, 10001101b, etc...

### 2. Các toán hạng: AX, DS, BL, SS, 1Fh, 20, variable, [DI], SI, [SI] thuộc loại toán hạng nào?

- Thanh ghi đa dụng : AX, DS, BL, SI
- Thanh ghi segment : SS
- memory : variable, [DI], [SI]
- immediate : 20, 1Fh

### 3. Tìm câu lệnh SAI trong các lệnh sau, và giải thích vì sao?

**ORG 100H**

**MOV AX, 0B800h**

**MOV DS, AX**

**MOV BX, CL**

**MOV DS, 123h**

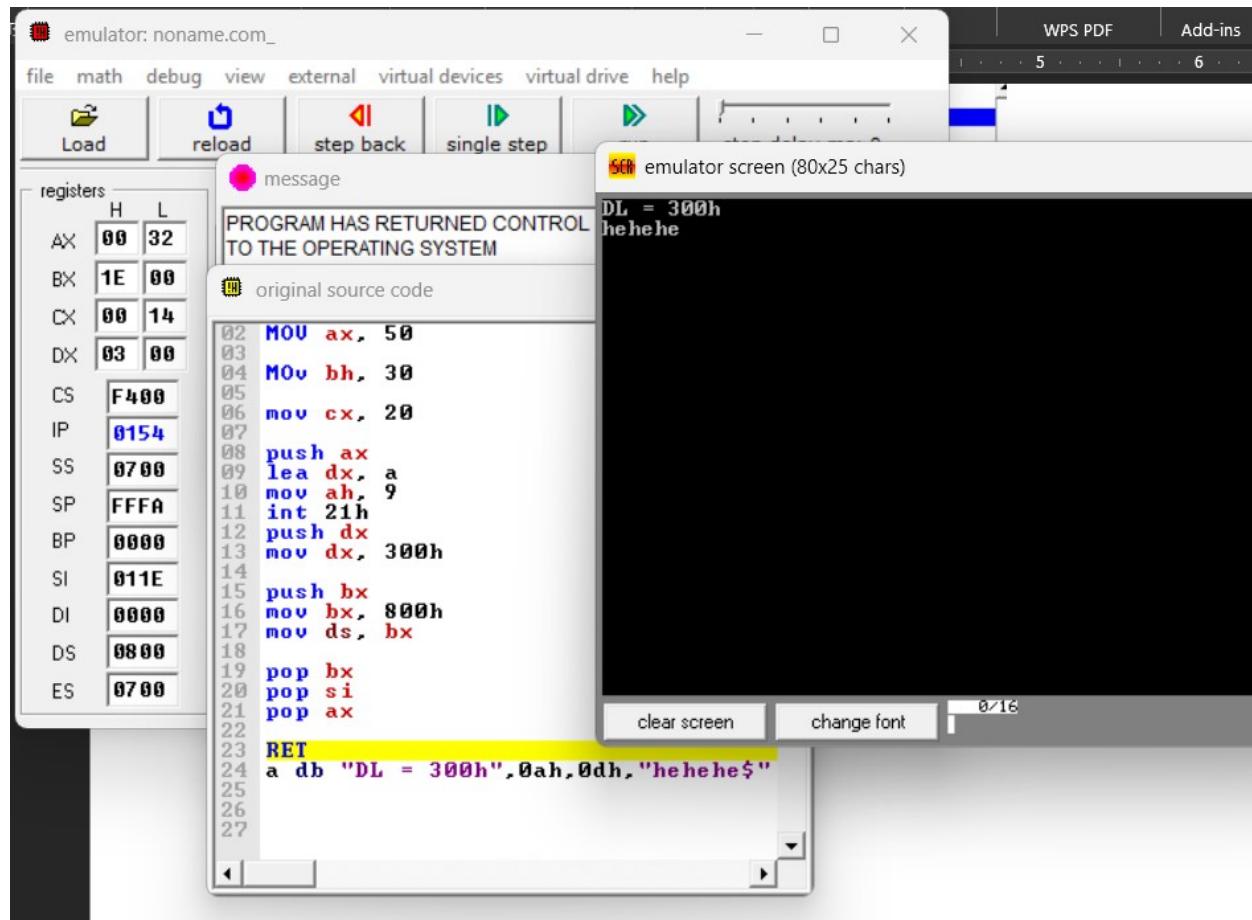
**MOV VAR, CX**

**RET**

**Var DB 13**

- Dòng lệnh MOV DS, 123h sai vì KHÔNG CÓ toán hạng MOV SREG, immediate.
- Dòng lệnh MOV VAR, CX sai vì VAR là biến có kiểu dữ liệu DB là 8 bit, mà cx là thanh ghi 16 bit, không tương thích kích thước.

**4. Viết chương trình gán các thanh ghi AX=50, BH=30, CX=20, DL=300h, DS=800h. Chạy chương trình và xem kết quả các thanh ghi trên giao diện "emulator" (Chụp hình kết quả các thanh ghi)**



### KEYBOARD I/O INSTRUCTIONS

1. Viết ứng dụng chuyển kí tự hoa sang thường, với các yêu cầu sau:

- Xuất ra 1 câu thông báo "Nhập 1 kí tự hoa bất kỳ."
- Rồi bạn tiến hành nhập 1 kí tự chữ hoa bất kỳ.
- Enter thì chương trình xuất ra: "Kí tự thường tương ứng: "
- Kèm với kí tự thường (tương ứng với kí tự hoa vừa nhập)
- Có dừng màn hình, đợi nhấn phím bất kỳ thì mới kết thúc chương trình. ((Xem video "Dừng màn hình...")

\*Gợi ý 1: xem lại về "Kí tự & bảng mã ASCII".

\*Gợi ý 2: xem thêm video "Lệnh ADD, SUB (cộng, trừ)"

\*Lưu kết quả vào 1 biến hoặc 1 thanh ghi đa dụng.

\*Màn hình xuất ra là "emulator screen".

```

    emulator: noname.com_
file math debug view external virtual
Load original source code
registers
AX 02 68
BX 08 68
CX 08 4F
DX 01 68
CS F400
IP 0154
SS 0700
SP FFFA
BP 0000
SI 0000
DI 0000
DS 0700
ES 0700
01 ORG 100h
02
03 lea dx, a
04 mov ah, 9
05 int 21h
06
07 mov ah, 1
08 int 21h
09
10 add al, 20h
11 mov bl, al
12
13 lea dx, b
14 mov ah, 9
15 int 21h
16
17 mov ah, 2
18 mov dl, bl
19 int 21h
20
21 ret
22 a db "Nhap 1 ki tu chu hoa : $"
23 b db 0ah,0dh,"Ki tu thuong tuong ung : $"
24
25
26
27

```

#### "PAUSE AND WAIT ANY KEYPRESS" INSTRUCTION

**Viết chương trình xuất: Họ tên, MSSV, Nơi sinh của bạn. Mỗi lần xuất thông tin, thì dừng lại đợi nhận 1 phím bất kỳ trước khi xuất nội dung tiếp theo.**

```

    emulator screen (80x25 chars)
Ho ten: Nguyen Nhu Ben
MSSV: 23119051
Noi sinh: Khanh Hoa - Nha Trang
01 ORG 100h
02
03 mov ah, 9
04 lea dx, a
05 int 21h
06
07 mov ah, 0
08 int 16h
09
10 mov ah, 9
11 lea dx, b
12 int 21h
13
14 mov ah, 0
15 int 16h
16
17 mov ah, 9
18 lea dx, c
19 int 21h
20
21 mov ah, 0
22 int 16h
23
24 ret
25 a db "Ho ten : Nguyen Nhu Ben",0ah,0dh,$"
26 b db "MSSV : 23119051",0ah,0dh,$"
27 c db "Noi sinh : Khanh Hoa - Nha Trang$"

```

#### STACK MEMORY

**Nhập vào 1 dãy gồm 5 kí tự từ bàn phím và lưu vào STACK. Sau đó, xuất ra dãy vừa nhập theo chiều ngược lại.**

```

org 100h
mov ah, 1      ;nhap
int 21h
push ax
mov ah, 1
int 21h
push ax
mov ah, 1
int 21h
push ax
mov ah, 1
int 21h
push ax
lea dx, a
mov ah, 9
int 21h
pop ax
mov dl, al
mov ah, 2
int 21h
pop ax
mov dl, al
mov ah, 2
int 21h
pop ax
mov dl, al
mov ah, 2
int 21h
pop ax
mov dl, al
mov ah, 2
int 21h
mov ah, 0
int 16h
RET
a db 0ah,0dh,"nguoc lai : $"

```

### [Buổi 5] Session 5 - Flag reg, Arithmetic instructions, loop

#### FLAG(STATUS) REGISTER

**1. Thanh ghi Cờ có bao nhiêu bit? Hiện tại sử dụng bao nhiêu bit? Ý nghĩa của từng bit Cờ là gì? (Lấy được ví dụ cho từng trạng thái của Cờ, hoặc bỏ qua nếu lười)**

- Có 16 bit nhưng hiện tại sử dụng 8 bit
- + Zero flag (Z) : 0/1 khi kết quả phép toán khác 0/bằng 0
- + Sign flag (S) : 0/1 kết quả >0/<0
- + Overflow flag (O) : 0 mặc định /1 khi giá trị thanh ghi bị tràn trong phép toán với số có dấu (signed overflow)
- + Carry flag (C) : 0 mặc định /1 khi giá trị thanh ghi bị tràn trong phép toán với số không dấu (unsigned overflow)
- + Parity flag (P) : 0/1 tổng số bit 1 là chẵn / lẻ
- + Auxiliary flag (A) : 0 mặc định /1 giá trị trong 4 bit thấp bị tràn ( số không dấu )

- + Interrupt enable flag (I) : 0 mặc định / 1 khi có ngắt xảy ra
- + Direction flag (D) : không sử dụng ( liên quan hướng xử lý 1 chuỗi process)

**2. Giá trị các bit Cờ thay đổi khi nào? (Gợi ý: Tự động cập nhật giá trị dựa trên kết quả của phép toán, có phép toán tác động, có phép toán không tác động)**

- Tự động cập nhật giá trị dựa trên kết quả của phép toán, có phép toán tác động, có phép toán không tác động

**3. Trong các lệnh sau, lệnh nào có tác động đến Cờ, lệnh nào không tác động: MOV, SUB, ADD, CMP, MUL, DIV, INC, DEC? Xem ở đâu để biết được điều đó?**

- chỉ có lệnh MOV (copy dữ liệu ) và lệnh DIV là không tác động đến cờ

**4. Theo video, trạng thái của bit Cờ được sử dụng để cho các lệnh nào hoạt động? Hay các lệnh tác động đến cờ (vd: CMP) phải được gọi trước khi sử dụng các lệnh nào?)**

- Để lệnh nhảy hoạt động

**INSTRUCTION: ADD, SUB**

**1. Viết chương trình tính tổng 2 số với các yêu cầu sau:**

- Xuất ra chuỗi "Nhập số 1: ", "Nhập số 2: ", "ket qua la: "
- Nhập vào 2 số nguyên dương (có 1 chữ số) từ bàn phím
- Tính tổng 2 số và xuất kết quả ra màn hình
- Có dừng màn hình, đợi nhấn phím bất kì thì mới kết thúc chương trình.
- Giá trị nhập vào và kết quả cần được lưu vào biến. (Tổng cộng là xài 3 biến)
- Để cho dễ, chỉ thử trường hợp: tổng là 1 số nhỏ hơn 10.

The screenshot shows a 6502 assembly language emulator interface. On the left, the assembly code is listed from line 01 to 39. Lines 01 through 30 are standard 6502 assembly instructions. Lines 31 to 35 contain string literals for input and output. Line 36 defines variable 'a' as a byte. Lines 37 and 38 define variables 'b' and 'c' as bytes. Line 39 is a RET instruction. The central window displays the emulator screen with the message "nhap so 1: 4", "nhap so 2: 3", and "ket qua la: 7". The bottom right of the screen shows font selection buttons for "clear screen", "change font", and a font size dropdown set to 8/16.

```
01 ORG 100h
02 lea dx,a
03 MOU ah,9
04 int 21h
05 mov ah,1
06 int 21h
07 mov cl, al
08
09
10 lea dx,b
11 mov ah,9
12 int 21h
13
14 mov ah,1
15 int 21h
16 mov ch,al
17
18 ADD cl,ch
19 SUB cl,30h
20
21 mov bl, cl
22
23 lea dx,c
24 mov ah,9
25 int 21h
26
27 mov ah,2
28 mov dl, bl
29 int 21h
30
31 mov ah,0
32 int 16h
33
34
35 ret
36 a db 'nhap so 1: $'
37 b db 0ah,0dh,'nhap so 2: $'
38 c db 0ah,0dh,'ket qua la: $'
39
```

2. Trong chương trình trên, lệnh nào tác động đến Cờ? (Gợi ý: Bạn mở tab “Emulator/view/flags”. Rồi bạn chạy từng lệnh, bấm “Single Step” và nhìn Flags xem có Cờ thay đổi giá trị không)

-không có lệnh nào tác động đến cờ

#### INSTRUCTION: INC, DEC

1. Tạo 1 biến, nhập giá trị từ bàn phím, dùng lệnh INC để tăng giá trị biến thêm 4. Xuất giá trị biến vừa tăng.

```

01 org 100h
02
03 lea dx,a
04 mov ah,9
05 int 21h
06
07 mov ah, 1
08 int 21h
09
10 inc al
11 inc al
12 inc al
13 inc al
14
15 mov bl, al
16
17 lea dx,b
18 mov ah,9
19 int 21h
20
21 mov ah, 2
22 mov dl, bl
23 int 21h
24
25 mov ah,0
26 int 16h
27
28 ret
29 a db 'nhap gia tri : $'
30 b db 0ah,0dh,'gia tri tang them 4 thanh : $'

```

### INSTRUCTION: LOOP

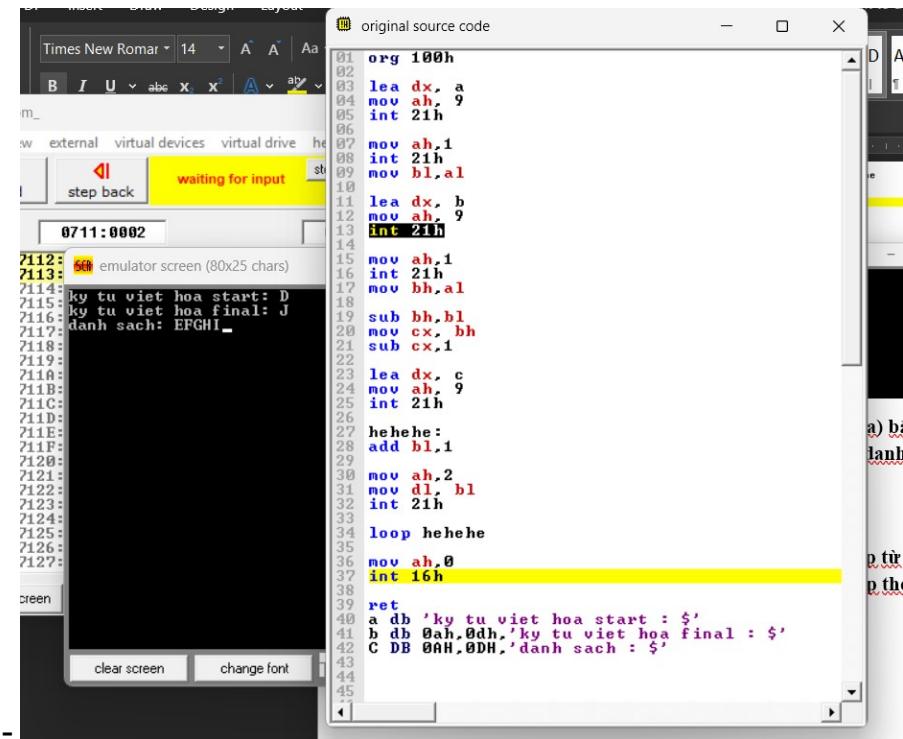
#### 1. Viết chương trình xuất ra bảng chữ cái (viết hoa) sử dụng lệnh LOOP.

```

01 org 100h
02 mov cx, 26
03 mov al, 65
04 hehehe:
05
06 mov ah,2
07 mov dl, al
08 int 21h
09 add al,1
10
11 loop hehehe
12
13 mov ah,0
14 int 16h
15
16 ret
17
18

```

#### 2. Viết chương trình nhập vào: kí tự chữ cái (viết hoa) bắt đầu và kí tự chữ cái (viết hoa) kết thúc. Sau đó, chương trình sẽ xuất ra danh sách các kí tự chữ cái nằm trong đoạn 2 kí tự vừa nhập.



3. Viết chương trình tính tổng 3 số (có 1 chữ số) nhập từ bàn phím, và lưu kết quả vào 1 biến. (\*)(Có thể ứng dụng cho các lệnh tiếp theo)

The screenshot shows a 16-bit assembly language debugger interface. The menu bar includes 'emulator', 'math', 'ascii codes', and 'help'. Below the menu is a toolbar with icons for 'save', 'compile', 'emulate', and 'calculator'. The main window has two panes: the left pane displays the 'emulator screen (80x25 chars)' with the output: 'tong 3 so: 5 +2 +2' and 'ket qua: 9'; the right pane shows the 'original source code' in assembly language:

```
01 org 100h
02
03 mov ah,9
04 lea dx,a
05 int 21h
06
07 mov cx,2
08 mov bh,0
09
10 bruh:
11
12 mov ah,1
13 int 21h
14 sub al,48
15 add bh,al
16
17 mov ah,9
18 lea dx,b
19 int 21h
20
21 mov al,0
22
23 loop bruh
24
25 mov ah,1
26 int 21h
27 sub al,48
28 add bh,al
29
30 mov al,0
31
32 add bh,48
33 mov ah,9
34 lea dx,c
35 int 21h
36 mov ah,2
37 mov dl,bh
38 int 21h
39
40 mov ah,0
41 int 16h
42
43 ret
44 a db 'tong 3 so : $'
45 b db '+$'
46 c db ',0ah,0dh,ket qua : $'
47
48
49
50
```

The instruction at address 41 (INT 16h) is highlighted in yellow.

## INSTRUCTION: MUL

### 1. Cho chương trình:

org 100h

MOV AX,1000

MUL X

ret

X DB 250

Y DW 250

- Hỏi phép nhân được thực hiện giữa 2 toán hạng gì? Kết quả bằng bao nhiêu? Và kết quả được lưu vào đâu? Chụp hình kết quả sau khi chạy lệnh MUL X.
- Thay lệnh MUL X thành lệnh MUL Y. Và trả lời lại các câu hỏi phía trên.

-Toán hạng: AL \* X

-Kết quả ax = E290h

-Lưu trong AX

```

original source code
01 org 100h
02 MOU AX,1000
03 MUL X
04 ret
05 X DB 250
06 Y DW 250
07
08
09
10

```

	H	L
D0	90	
00	00	
00	0B	
00	00	
F4 00		

-toán hạng : ax \* y

-Kết quả ax = D090h

-Lưu trong AX

```

original source code
01 org 100h
02 MOU AX,1000
03 MUL y
04 ret
05 X DB 250
06 Y DW 250
07
08
09
10

```

	H	L
D0	90	
00	00	
00	0B	
00	03	
F4 00		

2. Viết chương trình nhập vào 1 số nguyên (có 4 chữ số), với các yêu cầu sau:

- Xuất ra chuỗi "Moi ban nhap so: ",
- Số nguyên gồm 4 chữ số.
- Kết quả cần được lưu vào biến.
- Có dừng màn hình, đợi nhấn phím bất kì thì mới kết thúc chương trình.

\*Gợi ý:

- Ví dụ nhập 1234, thì trình tự nhập và gộp số:

o  $a = 1$

o  $a = a * 10 + 2$

o  $a = a * 10 + 3$

o  $a = a * 10 + 4$

- Phép nhân 10 là phép nhân với số 1 byte.

The screenshot shows a debugger interface with the following details:

- Assembly View:**

```

org 100h
MOU DX, OFFSET msg_input
MOU AH, 09h
INT 21h

MOU AX, 0
MOU CX, 4

input_loop:
MOU AH, 01h
INT 21h

SUB AL, '0'
MOU BL, AL

MOU DL, 10
MUL DL

ADD AL, BL
LOOP input_loop

MOU result, AX

MOU DL, 0Dh
MOU AH, 02h
INT 21h
MOU DL, 0Ah
MOU AH, 02h
INT 21h

MOU DX, OFFSET msg_press
MOU AH, 09h
INT 21h

MOU AH, 07h
INT 21h
ret

msg_input DB 'Moi ban nhap so: $'
msg_press DB 'Nhan phim bat ki de ket thuc...$'
result DW ?

```
- Registers View:**

H	L
07	24
00	04
00	00
01	4A
F400	
0200	
- Memory Dump View:**

F4200	FF	255	RES
F4201	FF	255	RES
F4202	CD	205	=
F4203	21	033	!
F4204	CF	207	_
F4205	00	000	NULL
F4206	00	000	NULL
F4207	00	000	NULL
F4208	00	000	NULL
F4209	00	000	NULL
F420A	00	000	NULL
F420B	00	000	NULL
- Output Window:**

Moi ban nhap so: 1234  
Nhan phim bat ki de ket thuc...

### 3. Viết chương trình tính hiệu 2 số nguyên dương (có 4 chữ số), với các yêu cầu sau:

- Xuất ra chuỗi "Nhập số 1: ", "Nhập số 2: ", "ket qua la: "
- Nhập vào 2 số nguyên dương (có 4 chữ số) từ bàn phím
- Bạn chủ động nhập Số bị trừ lớn hơn Số trừ
- Tính hiệu 2 số và lưu kết quả vào 1 biến (Hiệu lưu vào biến, không cần xuất ra)

- Có dừng màn hình, đợi nhấn phím bất kì thì mới kết thúc chương trình.
- Giá trị nhập vào và kết quả cần được lưu vào biến.

The screenshot shows the UTEXLMS emulator8086 interface. On the left, there's a terminal window displaying the following text:

```
Nhap so 1: 4112
Nhap so 2: 4002
ket qua la: <da luu vao bien>
Nhan phim bat ki de ket thuc...
```

On the right, the assembly code is displayed in the original source code window. The code implements a program that reads two numbers from the user, adds them, and prints the result. The assembly code is as follows:

```
01 org 100h
02 MOU DX, OFFSET msg_input1
03 MOU AH, 09h
04 INT 21h
05 CALL input_number
06 MOU num1, AX
07 CALL new_line
08 MOU DX, OFFSET msg_input2
09 MOU AH, 09h
10 INT 21h
11 CALL input_number
12 MOU num2, AX
13 CALL new_line
14 MOU AX, num1
15 SUB AX, num2
16 MOU result, AX
17 MOU DX, OFFSET msg_result
18 MOU AH, 09h
19 INT 21h
20 MOU DX, OFFSET msg_press
21 MOU AH, 09h
22 INT 21h
23 MOU AH, 07h
24 INT 21h
25 ret
26 input_number PROC
27 MOU AX, 0
28 MOU CX, 4
29 input_loop:
30 MOU AH, 01h
31 INT 21h
32 SUB AL, '0'
33 MOU BL, AL
34 MOU DL, 10
35 MUL DL
36 ADD AL, BL
37 LOOP input_loop
38 RET
39 input_number ENDP
40 new_line PROC
41 MOU DL, 0Dh
42 MOU AH, 02h
43 INT 21h
44 MOU DL, 0Ah
45 MOU AH, 02h
46 INT 21h
47 RET
48 new_line ENDP
49 msg_input1 DB 'Nhap so 1: $'
50 msg_input2 DB 'Nhap so 2: $'
51 msg_result DB 'ket qua la: <da luu vao bien>$'
52 msg_press DB 0Dh, 0Ah, 'Nhan phim bat ki de ket thuc'
53 num1 DW ?
54 num2 DW ?
55 result DW ?
56
```

#### 4. Làm lại bài 3, thay vì tính Hiệu bạn tính Tích 2 số.

```

        org 100h
        MOU DX, OFFSET msg_input1
        MOU AH, 09h
        INT 21h
        CALL input_number
        MOU num1, AX
        CALL new_line
        MOU DX, OFFSET msg_input2
        MOU AH, 09h
        INT 21h
        CALL input_number
        MOU num2, AX
        CALL new_line
        MOU AX, num1
        MOU BX, num2
        MUL BX
        MOU result, AX
        MOU result_high, DX
        MOU DX, OFFSET msg_result
        MOU AH, 09h
        INT 21h
        MOU DX, OFFSET msg_press
        MOU AH, 09h
        INT 21h
        MOU AH, 07h
        INT 21h
        ret
        input_number PROC
        MOU AX, 0
        MOU CX, 4
        input_loop:
        MOU AH, 01h
        INT 21h
        SUB AL, '0'
        MOU BL, AL
        MOU DL, 10
        MUL DL
        ADD AL, BL
        ADD AL, BL
        LOOP input_loop
        RET
        input_number ENDP
        new_line PROC
        MOU DL, 0Dh
        MOU AH, 02h
        INT 21h
        MOU DL, 0Ah
        MOU AH, 02h
        INT 21h
        RET
        new_line ENDP
        msg_input1 DB 'Nhập số 1: $'
        msg_input2 DB 'Nhập số 2: $'
        msg_result DB 'ket qua la: <da luu vao bien>$'
        msg_press DB 0Dh, 0Ah, 'Nhấn phím bat ki de ket thuc'
        num1 DW ?
        num2 DW ?
        result DW ?
        result_high DW ?

Claude Sonnet 4 ~

```

## INSTRUCTION: DIV

### 1. Cho đoạn chương trình:

```

org 100h
mov ax, 1000
DIV x
ret
x db 11
y db 3
z dw 3

```

Đây là phép chia 8bit hay 16bit? Kết quả phần thương lưu ở đâu và bằng bao nhiêu? Phần dư lưu ở đâu và bằng bao nhiêu? (Chụp hình minh họa)

- + Là phép chia 8 bit, kết quả được lưu vào AL và bằng 90 (5A)+ Phần dư được lưu vào AH và bằng 10 (A)

registers	H	L
AX	0A	5A
BX	00	00

- Thay lệnh “DIV x” bằng “DIV z”. Kết quả phần thương lưu ở đâu và bằng bao nhiêu?  
Phần dư lưu ở đâu và bằng bao nhiêu? (Chụp hình minh họa)  
Kết quả được lưu vào AX và bằng 333 (14D). Phần dư được lưu vào DX và bằng 1

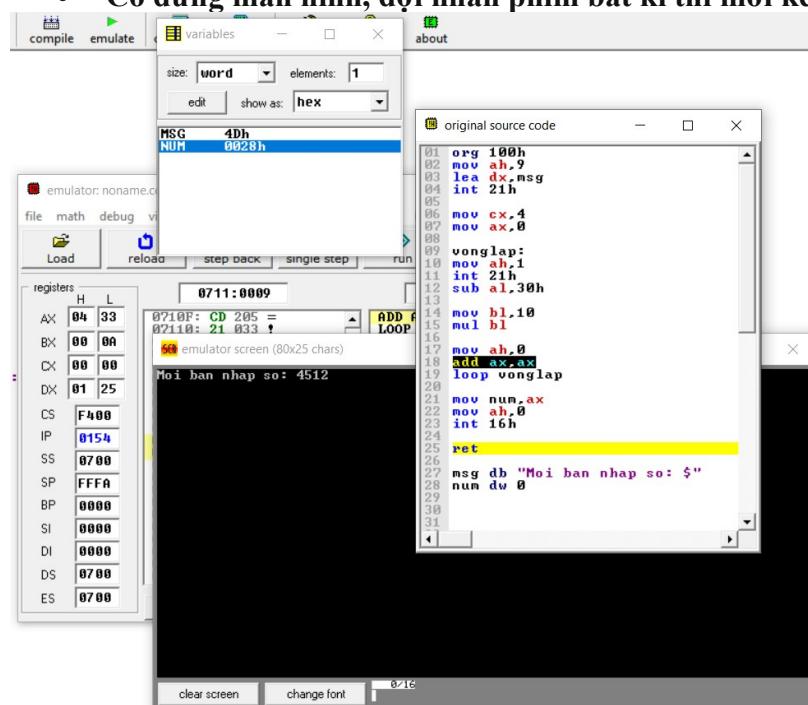
registers	H	L
AX	01	4D
BX	00	00
CX	00	0C
DX	00	01

- Thay lệnh “DIV x” bằng “DIV y” thì kết quả sẽ sao? Vì tại sao?  
Chương trình bị lỗi vì thương là 333 không lưu được vào thanh ghi AL 8 bit  
(max là 255)

## 2. Viết chương trình nhập vào 1 số nguyên (có 4 chữ số), với các yêu cầu sau:

- Xuất ra chuỗi "Moi ban nhap so: ",
- Số nguyên gồm 4 chữ số.
- Kết quả cần được lưu vào biến.
- Xuất số vừa nhập ra màn hình.

♣ Gợi ý: Sử dụng bộ nhớ stack để lưu từng số hạng, rồi xuất ngược ra màn hình.  
 • Có dừng màn hình, đợi nhấn phím bất kì thì mới kết thúc chương trình.



## 3. Viết chương trình chia 2 số nguyên dương cho nhau (có 4 chữ số), với các yêu cầu sau:

- Xuất ra chuỗi "Nhap so 1: ", "Nhap so 2: ", "Ket qua la: (Phan nguyen), (Phan du)"
- Nhập vào 2 số nguyên dương (có 4 chữ số) từ bàn phím và lưu vào 2 biến.
- Lấy 2 số chia cho nhau và lưu kết quả vào 2 biến (phần nguyên, phần dư)
- Có dừng màn hình, đợi nhấn phím bất kỳ thì xuất kết quả ra màn hình.

The screenshot shows a debugger interface with three main windows:

- Assembly Window:** Displays the assembly code for the program. The code involves input handling (INT 21H), variable initialization (MOV AX, 4, MOV BX, 10), calculations (DIV BX, ADD AX, Y), and output (CALL PrintNumber). It also includes loops for reading input and calculating the quotient and remainder.
- Registers Window:** Shows the current state of CPU registers. The AX register contains 00 7D, BX contains 02 26, CX contains 00 00, and DX contains 01 2C. Other registers like CS, IP, SS, SP, BP, SI, DI, DS, and ES have their standard values.
- Variables Window:** Displays memory variables. It lists strings MSG1, MSG2, MSG3, and variables X, Y, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, and R. The variable R is currently selected and highlighted in blue.

### [Buổi 6] Session 6 - Jump, Logic instructions

#### INSTRUCTION: JUMP

1. Xuất bảng chữ cái A-Z. Mỗi lần nhấn 1 phím bất kỳ thì mới xuất ký tự tiếp theo.

The screenshot shows a debugger interface with the following details:

- Registers:**

	H	L
AX	00	5A
BX	00	00
CX	00	01
DX	00	5A
CS	F400	
IP	01C0	
SS	0700	
SP	FFF8	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	
- Original source code:**

```

01 org 100h
02 mov cx, 26
03 mov dl, 'A'
04
05 print:
06 mov ah, 02h
07 int 21h
08 mov ah, 0
09 int 16h
10 inc dl
11 loop print
12 ret

```
- Emulator screen:** Displays the characters A through Z.

2. Xuất bảng cửu chương 2 trên nhiều dòng. Mỗi lần nhấn 1 phím bất kỳ thì mới xuất dòng tiếp theo.

The screenshot shows a debugger interface with the following details:

- Registers:**

	H	L
AX	00	0A
BX	00	0A
CX	00	6E
DX	00	0A
CS	F400	
IP	01C0	
SS	0700	
SP	FFF8	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	
- Original source code:**

```

01 org 100h
02 mov bl, 1
03 print:
04 cmp bl, 11
05 jg done
06 mov al, 2
07 call PrintNum
08 mov dl, 02h
09 int 21h
10 mov dl, 'x'
11 int 21h
12 mov dl, ','
13 int 21h
14 mov al, bl
15 call PrintNum
16 mov dl, ','
17 mov ah, 02h
18 int 21h
19 mov dl, '='
20 int 21h
21 int 21h
22 mov dl, ','
23 int 21h
24 mov al, 2
25 mul bl
26 call PrintNum
27 mov dl, 0Dh
28 mov ah, 02h
29 int 21h
30 mov dl, 0Ah
31 int 21h
32 mov ah, 0
33 int 16h
34 inc bl
35 jmp print
done:
37 ret
PrintNum:
39 push ax
40 push bx
41 push cx
42 push dx
43 mov ah, 0
44 mov cx, 0
45 mov bx, 10
46 divide:
47 mov dx, 0
48 div bx
49 push dx
50 inc cx
51 cmp ax, 0
52 jne divide
show:
54 pop dx
55 add dl, '0'
56 mov ah, 02h
57 int 21h
58 loop show
59 pop dx
60 pop cx
61 pop bx
62 pop ax
63 ret

```
- Emulator screen:** Displays the multiplication table of 2 from 1 to 10.

### INSTRUCTION: CMP

#### 1. Cờ ZF (Zero) bằng 0 khi nào, bằng 1 khi nào? (Khi thực hiện lệnh CMP)

- Cờ ZF = 1 khi kết quả phép so sánh 2 toán hạng là bằng nhau
- Cờ ZF = 0 khi kết quả phép so sánh 2 toán hạng không bằng nhau

#### 2. Lệnh CMP khác lệnh SUB ntn?

- CMP: so sánh hai toán hạng bằng cách thực hiện phép trừ (toán hạng 1 - toán hạng 2) nhưng không lưu kết quả, chỉ ảnh hưởng tới các cờ trạng thái (flags).
- SUB: thực hiện phép trừ thật sự, lưu kết quả vào toán hạng đích và cập nhật các cờ trạng thái.

#### 3. Khi toán hạng 1 lớn hơn toán hạng 2 thì những cờ nào lên 1? Ý nghĩa cờ đó là gì? (Không bắt buộc, siêng thì nên làm cho biết)

- Khi toán hạng 1 > toán hạng 2, thường thấy CF=0, ZF=0, SF=0, cho biết kết quả dương

#### 4. Khi toán hạng 1 nhỏ hơn toán hạng 2 thì những cờ nào lên 1? Ý nghĩa cờ đó là gì? (Không bắt buộc, siêng thì nên làm cho biết)

- Khi toán hạng 1 < toán hạng 2, thường CF=1 hoặc SF=1 để báo hiệu kết quả âm hoặc mượn.

### CONDITIONAL JUMP INSTRUCTION

#### 1. Viết chương trình nhập vào 2 số (số có 1 chữ số). Xuất ra số lớn hơn.

```

01 org 100h
02 lea dx, mgs1
03 mov ah, 9
04 int 21h
05 mov ah, 1
06 int 21h
07 sub al, '0'
08 mov num1, al
09 lea dx, mgs2
10 mov ah, 9
11 int 21h
12 mov ah, 1
13 int 21h
14 sub al, '0'
15 mov num2, al
16 mov al, num1
17 mov bl, num2
18 cmp al, bl
19 ja num1_is_greater ; Neu num1 > num2
20
21 lea dx, mgs3
22 mov ah, 9
23 int 21h
24 mov dl, num2
25 add dl, 48
26 mov ah, 2
27 int 21h
28 jmp exit
29 num1_is_greater:
30 lea dx, mgs3
31 mov ah, 9
32 int 21h
33 mov dl, num1
34 add dl, 48
35 mov ah, 2
36 int 21h
37 exit:
38 mov ah, 0
39 int 16h
40 mgs1 db "Nhập số thứ 1: $"
41 mgs2 db 0Dh,0Ah, "Nhập số thứ 2: $"
42 mgs3 db 0Dh,0Ah, "So lon hon la: $"
43 num1 db ?
44 num2 db ?
45

```

## 2. Viết chương trình nhập vào n số (số có 1 chữ số). Xuất ra số lớn nhất.

Gợi ý: Dùng mảng hoặc dùng Stack lưu các số đều được.

```

01 org 100h
02 lea DX, tb1
03 mov AH, 9
04 int 21h
05 mov AH, 1
06 int 21h
07 SUB AL, 30h
08 Mov n, AL
09 lea DX, tb2
10 mov AH, 9
11 int 21h
12 mov AL, n
13 mov AH, 0
14 mov CX, AX
15 mov SI, 0
16 nhapso:
17 mov AH, 1
18 int 21h
19 SUB AL, 30h
20 Mov ktu[SI], AL
21 inc SI
22 loop nhapso
23 mov SI, 0
24 mov AL, n
25 mov AH, 0
26 mov CX, AX
27 DEC CX
28 mov BL, ktu[SI]
29 mov max, BL
30 sosanh:
31 inc SI
32 mov BL, ktu[SI]
33 CMP BL, max
34 JBE labe12
35 mov max, BL
36 labe12:
37 loop sosanh
38 lea DX, tb3
39 mov AH, 9
40 int 21h
41 ADD max, 30h
42 MOU DL, max
43 mov AH, 2
44 int 21h
45 exit:
46 ret
47 tb1 DB 0Ah, 'so chu so muon nhap la n = $'
48 tb2 DB 0Ah, 0Dh, 'nhap vao n chu so: $'
49 tb3 DB 0Ah, 0Dh, 'so lon nhat la: $'
50 ktu DB 15 DUP(0)
51 n DB 0
52 max DB 0
53

```

### 3. Viết chương trình nhập vào 2 số bất kỳ, điều kiện để dừng nhập 1 số là khi bấm phím Enter (0Dh). Tính tổng 2 số đó.

The screenshot shows a debugger interface with the following details:

- Registers:** AX=00 95, BX=00 19, CX=00 0A, DX=01 A0, CS=F400, IP=01C0, SS=0700, SP=FFF8, BP=0000, SI=0000, DI=0000, DS=0700, ES=0700.
- Stack:** The stack contains assembly instructions and data. It shows inputs "Nhập số thứ 1: 124", "Nhập số thứ 2: 25", and the output "Tổng 2 số là: 149".
- Code View:** The assembly code is displayed in columns. The first column shows the original assembly code, and the second column shows the assembly code with comments explaining the logic.

```

01 org 100h
02 lea dx, msg1
03 mov ah, 9
04 int 21h
05 call InputNumber
06 mov num1, bx
07 lea dx, msg2
08 mov ah, 9
09 int 21h
10 call InputNumber
11 mov num2, bx
12 mov ax, num1
13 add ax, num2
14 mov result, ax
15 lea dx, msg3
16 mov ah, 9
17 int 21h
18 mov ax, result
19 call PrintNumber
20 jmp exit
21 InputNumber:
22 mov bx, 0
23 mov cx, 10
24 input_loop:
25 mov ah, 1
26 int 21h
27 cmp al, 0Dh
28 Je input_done
29 sub al, '0'
30 mov ah, 0
31 push ax
32 mov ax, bx
33 mul cx
34 mov bx, ax
35 pop ax
36 add bx, ax
37 jmp input_loop
38 input_done:
39 ret
40 PrintNumber:
41 push ax
42 push bx
43 push cx
44 push dx
45 mov cx, 0
46 mov bx, 10
47 divide_loop:
48 mov dx, 0
49 div bx
50 push dx
51 inc cx
52 cmp ax, 0
53 jne divide_loop
54 print_loop:
55 pop dx
56 add dl, '0'
57 mov ah, 2
58 int 21h
59 loop print_loop
60 pop dx
61 pop cx
62 pop bx
63 pop ax
64 ret
65 exit:
66 mov ah, 0
67 int 16h
68 ret
69 msg1 db "Nhập số thứ 1: $"
70 msg2 db 0Dh, 0Ah, "Nhập số thứ 2: $"
71 msg3 db 0Dh, 0Ah, "Tổng 2 số là: $"
72 num1 dw 0
73 num2 dw 0
74 result dw 0
    
```

### 4. Làm lại bài 3 nhưng tính hiệu.

The screenshot shows a debugger interface with the following components:

- Registers pane:** Displays CPU register values (AX, BX, CX, DX, CS, IP, SS, SP, BP, SI, DI, DS, ES) in both H (Hex) and L (Dec) formats.
- Memory dump pane:** Shows memory at address 0713:0000, displaying assembly instructions and their addresses.
- Stack pane:** Displays the stack content starting at address 0700.
- Code pane:** Shows the original source code in assembly language.
- Output pane:** Displays the emulator screen output, showing the results of the program execution.

```

    81 org 100h
    82 lea dx, msg1
    83 mov ah, 9
    84 int 21h
    85 call InputNumber
    86 mov num1, bx
    87 lea dx, msg2
    88 mov ah, 9
    89 int 21h
    90 call InputNumber
    91 mov num2, bx
    92 mov ax, num1
    93 sub ax, num2
    94 cmp ax, result
    95 jz result_ax
    96 lea dx, msg3
    97 mov ah, 9
    98 int 21h
    99 call PrintNumber
    100 mov ah, 4ch
    101 int 21h
    102 exit:
    103 mov ah, 0
    104 int 16h
    105 ret
    106 msg1 db "Nhập số thứ 1: $"
    107 msg2 db 0Dh, 0Ah, "Nhập số thứ 2: $"
    108 msg3 db 0Dh, 0Ah, "Hieu 2 so la: $"
    109 num1 dw 0
    110 num2 dw 0
    111 result dw 0
  
```

## 5. Làm lại bài 3 nhưng tính tích.

```

file edit bookmarks assembler emulator view external virtual de
new open examples save
file math debug view external virtual de
Load reload step back
registers
AX H L
BX
CX
DX
CS
IP
SS
SP
BP
SI
DI
DS
ES
0712:000F
0712F: 01 161 3
07130: B7 183 11
07131: 01 001 01
07132: EB 232 0
07133: 21 033 !
07134: 00 000 NULL
07135: EB 235 6
07136: 00 000 NULL
07137: BB 187 17
07138: 00 000 NULL
07139: 00 000 NULL
0713A: B9 185 19
0713B: 00 010 NEWI
0713C: 00 000 NULL
0713D: B4 180 21
0713E: 00 000 NULL
0713F: CD 995 24
07140: 21 033 !
07141: 3C 060 <
07142: BD 013 CRET
07143: 74 115 t
07144: 10 016 ▶
screen source reset
emulator screen (80x25 c)
Nhập số thu 1: 51
Nhập số thu 2: 3
Tích 2 số là: 153
input_done:
ret
PrintNumber:
push ax
push bx
push cx
push dx
pop ax
add dl, '0'
pop cx
pop bx
pop ax
ret
exit:
mov ah, 0
int 16h
ret
msg1 db "Nhập số thu 1: $"
msg2 db 0Ah, 0Ah, "Nhập số thu 2: $"
msg3 db 0Ah, 0Ah, "Tích 2 số là: $"
num1 dw 0
num2 dw 0
result dw 0

```

## 6. Làm lại bài 3 nhưng tính thương.

The screenshot shows a debugger interface with the following details:

- Registers:** AX=00 03, BX=00 04, CX=00 0A, DX=01 C2, CS=F400, IP=01C8, SS=0700, SP=FFF8, BP=0000, SI=0000, DI=0000, DS=0700, ES=0700.
- Stack:** The stack shows memory addresses from 07108 to 0711A, containing assembly instructions and data.
- Emulator Screen:** Displays the message "Nhập số thu 1: 63 Nhập số thu 2: 4 Thương la: 15, Du la: 3".
- Code Area:** Shows assembly code for various functions like PrintNumber, InputNumber, and a main loop.

## 7. Làm lại bài 3 nhưng chương trình cho phép nhập thêm phép toán +, -, \*, / để thực hiện phép toán tương ứng với 2 số vừa nhập.

Testcase 1 : cộng

The screenshot shows a debugger interface with the following components:

- Registers:** Shows CPU register values (AX, BX, CX, DX, SI, DI, DS, ES) and memory addresses.
- Stack:** Displays assembly instructions and their addresses (e.g., 071D: 0001, 071D: 0002, etc.).
- Emulator Screen:** Shows the output of the program, which includes input prompts and calculated results.
- Original Source Code:** Shows the assembly code being executed.

```

org 100h
lea dx, msg1
mov ah, 9
int 21h
call InputNumber
mov num1, bx
lea dx, msg2
mov ah, 9
int 21h
mov ah, 1
int 21h
mov operator, al
lea dx, msg3
mov ah, 9
int 21h
call InputNumber
mov num2, bx
mov al, operator
cmp al, '+'
je do_add
cmp al, '-'
je do_sub
cmp al, '*'
je do_mul
cmp al, '/'
je do_div
jmp exit
do_add:
    mov ax, num1
    add ax, num2
    mov result, ax
    lea dx, msg_result
    mov ah, 9
    int 21h
    mov ax, result
    call PrintNumber
    jmp exit
do_sub:
    mov ax, num1
    sub ax, num2
    mov result, ax
    lea dx, msg_result
    mov ah, 9
    int 21h
    mov ax, result
    call PrintNumber
    jmp exit
do_mul:
    mov ax, num1
    mul bx
    mov result, ax
    lea dx, msg_result
    mov ah, 9
    int 21h
    mov ax, result
    call PrintNumber
    jmp exit
do_div:
    mov ax, num1
    div bx
    mov result, ax
    lea dx, msg_result
    mov ah, 9
    int 21h
    mov ax, result
    call PrintNumber
    jmp exit
InputNumber:
    mov bx, 0
    mov cx, 10
    input_loop:
        mov ah, 1
        int 21h
        cmp al, 0Ah
        je input_done
        sub al, '0'
        push ax
        mov ax, bx
        mul cx
        mov bx, ax
        pop ax
        remainder dw 0
        operator db 0
    input_done:
        mov ah, 0
        int 16h
        ret
    msg1 db "Nhập số thu 1: $"
    msg2 db 0Dh, 0Ah, "Nhập phép toán (+, -, *, /): $"
    msg3 db 0Dh, 0Ah, "Nhập số thu 2: $"
    msg4 db 0Dh, 0Ah, "Kết quả: $"
    msg5 db "Nhập số thu 1: 56
Nhập phép toán (+, -, *, /): -
Nhập số thu 2: 20
Kết quả: 36_

```

Testcase 2: trù

The emulator screen displays the following output:

```

Nhập số thu 1: 56
Nhập phép toán (+, -, *, /): -
Nhập số thu 2: 20
Kết quả: 36_

```

Testcase 3: nhân

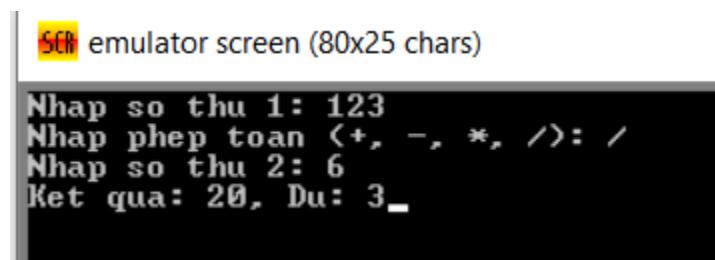
The emulator screen displays the following output:

```

Nhập số thu 1: 20
Nhập phép toán (+, -, *, /): *
Nhập số thu 2: 30
Kết quả: 600

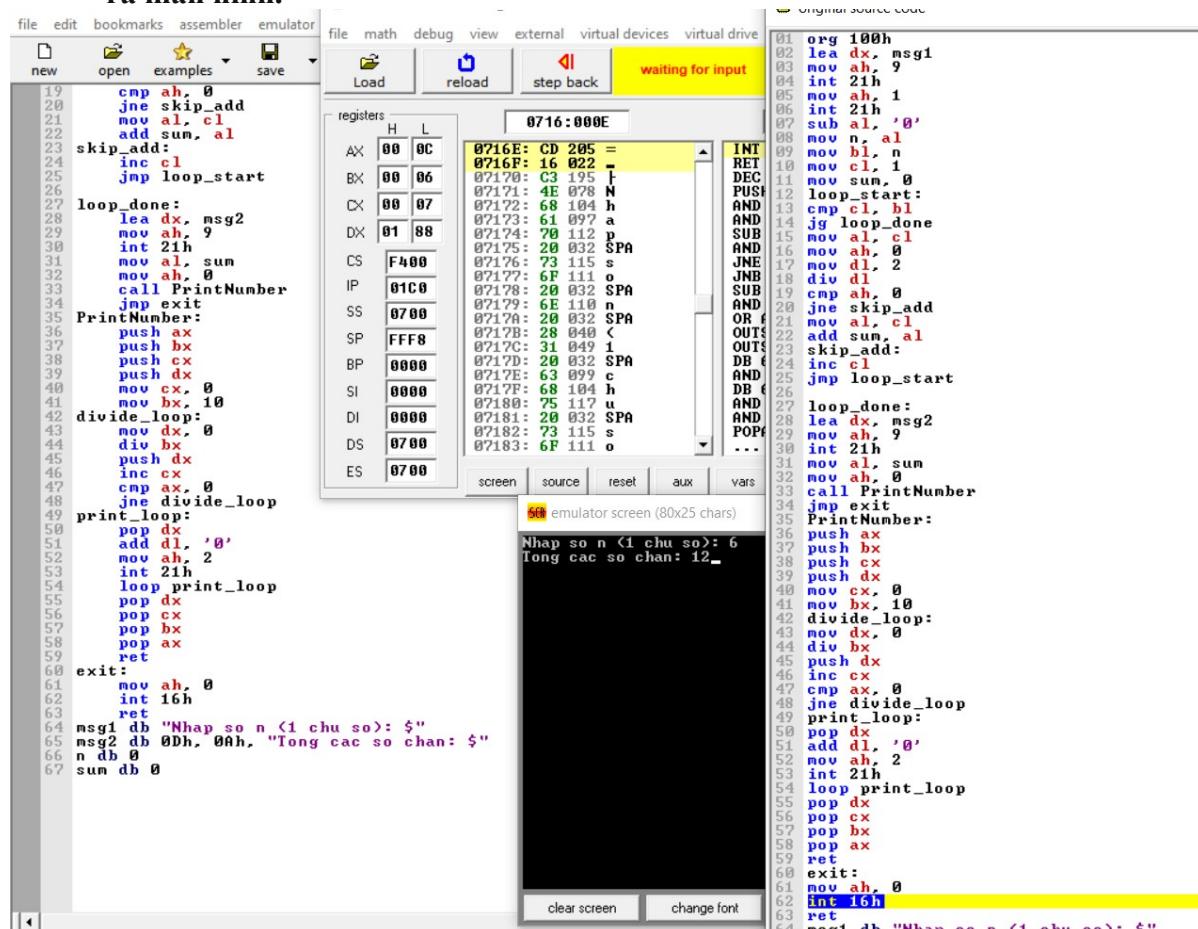
```

Testcase 4: chia



## VÍ DỤ TẠO VÒNG LẶP (TỪ LỆNH NHẤY CÓ ĐIỀU KIỆN)

- Nhập 1 số nguyên dương (1 chữ số). Tính tổng các số chẵn từ 1 đến n. Xuất kết quả ra màn hình.



## LOGIC OPERATIONS (CÁC PHÉP TOÁN LOGIC)

- Cho AL = 1110 1111B. Viết chương trình xóa (clear) bit thứ 5 của AL, giữ nguyên các bit còn lại.

The screenshot shows a debugger window with the following assembly code:

```

01 org 100h
02 mov al, 11101111b
03 and al, 11011111b
04 ; now AL = 11001111b
05 ret

```

The registers pane shows:

	H	L
AX	00	CF

The extended value viewer for AX shows:

watch: AX	<input type="radio"/> word <input checked="" type="radio"/> byte
hex:	00 CF
bin:	00000000 11001111
oct:	000 317

2. Cho AL = 0010 1001B. Viết chương trình set bit thứ 6 của AL lên 1, giữ nguyên các bit còn lại.

The screenshot shows a debugger window with the following assembly code:

```

01 org 100h
02 mov al, 00101001b
03 or al, 01000000b
04 ; now AL = 01101001b
05 ret

```

The registers pane shows:

	H	L
AX	00	69

The extended value viewer for AX shows:

watch: AX	<input type="radio"/> word <input checked="" type="radio"/> byte
hex:	00 69
bin:	00000000 01101001
oct:	000 151

3. Giả sử thanh ghi AL chứa một byte dữ liệu cảm biến, trong đó:

- Bit 7: Báo lỗi (Error)
- Bit 6: Nhiệt độ cao

- Bit 5: Quạt đang chạy

- Các bit khác: không quan trọng

Viết chương trình lấy ra giá trị của 3 cờ trạng thái này, lưu vào 3 biến, rồi xuất 3 biến này ra màn hình.

```

81 org 100h
82 mov al, 10100100b
83 mov bl, 5
84 and bl, 10000000b
85 shr bl, 7
86 mov err, bl
87 mov bl, al
88 and bl, 01000000b
89 shr bl, 6
90 mov temp, bl
91 mov bl, al
92 and bl, 00100000b
93 shr bl, 5
94 mov fan, bl
95 lea dx, msg1
96 mov ah, 9
97 int 21h
98 mov al, err
99 add al, '0'
100 mov dl, al
101 mov ah, 2
102 int 21h
103 lea dx, msg2
104 mov ah, 9
105 int 21h
106 mov al, temp
107 add al, '0'
108 mov dl, al
109 mov ah, 2
110 int 21h
111 lea dx, msg3
112 mov ah, 9
113 int 21h
114 mov al, fan
115 add al, '0'
116 mov dl, al
117 mov ah, 2
118 int 21h
119 mov ah, 0
120 int 16h
121 ret
122 msg1 db 0Dh,0Ah,"Error = $"
123 msg2 db 0Dh,0Ah,"Temp High = $"
124 msg3 db 0Dh,0Ah,"Fan On = $"
125 err db ?
126 temp db ?
127 fan db ?
128

```

### [Buổi 7] Bài 06 – Tổng quan dòng vi điều khiển 8051

#### 1. Trình bày cấu trúc cơ bản của dòng vi điều khiển 8051.

Vì điều khiển 8051 gồm các khối chức năng:

- + Bộ xử lý trung tâm (CPU)
- + Bộ nhớ chương trình (ROM 4kB) và bộ nhớ dữ liệu (RAM 128B)
- + Các thanh ghi có chức năng đặc biệt (SFR)
- + Các cổng vào ra (P0,P1,P2,P3)
- + Bộ định thời/bộ đếm (Timer0, Timer1)
- + Bộ giao tiếp nối tiếp (Serial)

#### 2. Liệt kê các ngoại vi cơ bản của dòng vi điều khiển 8051.

- + Các cổng vào ra (P0,P1,P2,P3)
- + Bộ định thời/bộ đếm (Timer0, Timer1)
- + Bộ giao tiếp nối tiếp (Serial)

### 3. Trình bày các vùng bộ nhớ và ý nghĩa của dòng vi điều khiển 8051.

- Bộ nhớ chương trình (Program Memory): Dùng để lưu trữ mã lệnh (code) của chương trình.

ROM nội

Dung lượng: 4KB. Khi chạy, CPU sẽ đọc lệnh từ đây để thực thi.

Nếu ROM nội không đủ, có thể mở rộng thêm ROM ngoài đến tối đa 64KB.

- Bộ nhớ dữ liệu (Data Memory): Dùng để lưu dữ liệu tạm thời

RAM nội

Dung lượng: 128 byte

Chia thành các vùng nhỏ:

- 00h – 1Fh: 32 byte cho 4 bank thanh ghi (R0–R7).
- 20h – 2Fh: 16 byte RAM định địa chỉ từng bit (128 bit).
- 30h – 7Fh: RAM đa dụng (biên tạm, dữ liệu người dùng).

Ngoài ra còn có RAM đặc biệt (SFR, 80h – FFh): chứa các thanh ghi chức năng đặc biệt như ACC, B, PSW, SP, DPTR, các thanh ghi điều khiển Timer, Serial, ngắt, các cổng P0–P3... Có thể mở rộng thêm RAM ngoài thêm 64KB

### 4. Tạo Project ASM và chạy mô phỏng Proteus với đoạn chương trình sau. Hỏi: mục đích của đoạn chương trình là gì? (Chạy mô phỏng xem kết quả)

The screenshot shows a code editor window with three tabs: '4nutnhan.asm', 'main.c\*', and 'main.asm'. The 'main.asm' tab is active, displaying the following assembly code:

```
1 //6. Cau hoi va bai tap ap dung
2 LED1 EQU P1.1
3 ORG 00H
4 SETB LED1
5 LOOP:
6 CPL LED1
7 CALL DELAY200ms
8 JMP LOOP
9 DELAY200ms:
10 MOV R2, #200
11 LAP2:
12 MOV R1, #200
13 LAP1:
14 NOP
15 NOP
16 NOP
17 DJNZ R1, LAP1
18 DJNZ R2, LAP2
19 RET
20 END
```

Mục đích : Làm nháy LED nối tại chân P1.1, với chu kỳ khoảng 200ms.

## 5. Tạo Project C và chạy mô phỏng Proteus với đoạn chương trình sau. Hỏi: mục đích của đoạn chương trình là gì? (Chạy mô phỏng xem kết quả)

The screenshot shows a code editor window with three tabs: '4nutnhan.asm', 'main.c\*', and 'main.asm'. The 'main.c\*' tab is active, displaying the following C code:

```
1 //6. Cau hoi va bai tap ap dung
2 #include <REGX52.H>
3
4 sbit LED = P1^1;
5
6 void delay_ms(unsigned int ms) {
7     unsigned int i,j;
8     for (i = 0; i < ms; i++)
9         for (j = 0; j < 123; j++);
10 }
11
12 void main() {
13     while (1) {
14         LED = 1;
15         delay_ms(200);
16         LED = 0;
17         delay_ms(200);
18     }
19 }
```

Mục đích: Đây là chương trình nháy LED . LED sẽ sáng trong 200ms, tắt trong 200ms, rồi lặp lại.

\*Tổng hợp code trong bài :

The image shows two side-by-side code editors. The left editor contains assembly language code (4nutnhan.asm) with the following content:

```

1 //6. Cau hoi va bai tap ap dung
2 LED1 EQU P1.1
3 ORG 00H
4 SETB LED1
5 LOOP:
6 CPL LED1
7 CALL DELAY200ms
8 JMP LOOP
9 DELAY200ms:
10 MOV R2, #200
11 LAP2:
12 MOV R1, #200
13 LAP1:
14 NOP
15 NOP
16 NOP
17 DJNZ R1, LAP1
18 DJNZ R2, LAP2
19 RET
20 END

```

The right editor contains C language code (main.c) with the following content:

```

1 //6. Cau hoi va bai tap ap dung
2 #include <REGX52.H>
3
4 sbit LED = P1^1;
5
6 void delay_ms(unsigned int ms) {
7     unsigned int i,j;
8     for (i = 0; i < ms; i++)
9         for (j = 0; j < 123; j++);
10 }
11
12 void main() {
13     while (1) {
14         LED = 1;
15         delay_ms(200);
16         LED = 0;
17         delay_ms(200);
18     }
19 }

```

### [Buổi 8] Bài 07 - Tập lệnh 8051

#### 1. Chạy lại 5 chương trình mẫu, xem và giải thích

The image shows three code editors side-by-side, each containing assembly language code:

- Editor 1 (main.asm):** Contains code for a simple arithmetic program (addition, subtraction, multiplication). It includes a table read loop and output to port 1.
- Editor 2 (main2.asm):** Contains code for a table read program, reading values from memory and outputting them to port 1.
- Editor 3 (main3.asm):** Contains code for a table read program, reading values from ROM and outputting them to port 1.

```

1 //4.4 CHUONG TRINH 4
2 ORG 0000H
3
4     MOV R0, #30H
5     MOV A, #20
6     MOV @R0, A
7
8     MOV R1, #31H
9     MOV A, #6
10    MOV @R1, A
11
12    CALL DIVIDE
13
14    JMP $
15
16    DIVIDE:
17    MOV R0, #30H
18    MOV A, @R0
19
20    MOV R1, #31H
21    MOV B, @R1
22
23    DIV AB
24
25    MOV P1, A
26    MOV P2, B
27
28    RET
29
30    END
31
32 //4.5 chuong trinh 05
33 // [sram1] 32kb (0000h - 7ffffh = 0 - 32767 )
34 // [sram2] 32kb (8000h - fffffh = 32768 - 65535)
35 ORG 00H
36
37 MAIN:
38     RAMLOC EQU 07FFAH
39     COUNT EQU 8
40
41     MOV DPTR, #RAMLOC
42     MOV R3, #COUNT
43     MOV A, #01H
44
45     repl:
46     MOVX @DPTR, A
47     RL A
48     INC DPTR
49     DJNZ R3, repl
50
51     MOV DPTR, #RAMLOC
52     MOV R3, #COUNT
53
54     rep2:
55     MOVX A, @DPTR
56     MOV P1, A
57     CALL DELAY
58     INC DPTR
59     DJNZ R3, rep2
60
61     JMP MAIN
62
63     DELAY:
64     MOV R1, #200
65
66     LAP1:
67     MOV R0, #200
68
69     LAP:
70     NOP
71     NOP
72     DJNZ R0, LAP
73     DJNZ R1, LAP1
74
75     RET
76
77     END

```

- Chương trình 01: Gán 2 giá trị không dấu vào 2 thanh ghi R0,R2. Tính tổng, hiệu, tích và xuất ra các Port 0,1,2,3 (dạng nhị phân).
- Chương trình 02: Lưu 2 hằng số vào vùng nhớ ROM, đọc và tính hiệu và xuất ra các Port1.
- Chương trình 03 : Tính tổng các số từ 1 đến N (trong trường hợp này N = 5).
- Chương trình 04 : Thực hiện phép chia hai số và trả về thương và số dư.
- Chương trình 05 : Tạo hiệu ứng LED chạy (LED running/shifting) bằng cách dịch bit sang trái và hiển thị tuần tự ra cổng P1.

## 2. Tính chính xác thời gian delay khi gọi đoạn chương trình con DELAY ( chương trình 5)

Vòng ngoài chạy 200 lần ( $R1 = 200 \rightarrow 1$ ).

Trong mỗi vòng ngoài:

$MOV R0, #200 \rightarrow 1\text{ mc}$

Vòng trong chạy 200 lần ( $R0 = 200 \rightarrow 1$ ):

$NOP \rightarrow 1\text{ mc}$

$NOP \rightarrow 1\text{ mc}$

$DJNZ R0, LAP \rightarrow 2\text{ mc}$

$\rightarrow$  Tổng cho 1 vòng trong:  $1 + 1 + 2 = 4\text{ mc}$

$\rightarrow 200\text{ lần} \rightarrow 200 \times 4 = 800\text{ mc}$

Sau khi kết thúc vòng trong: DJNZ R1, LAP1 → 2 mc

⇒ Mỗi vòng ngoài tốn: 1 (MOV R0) + 800 (vòng trong) + 2 (DJNZ R1) = 803 mc

⇒ Toàn bộ 200 vòng ngoài:  $200 \times 803 = 160600$  mc

Cộng thêm lệnh khởi tạo R1 ban đầu: 1 mc

→ Tổng cộng: 160601 mc

Với tần số 12 MHz → 1 mc = 1  $\mu$ s

⇒ Thời gian delay  $\approx 160601 \mu\text{s} \approx 160,6$  ms

### 3. Viết mã machine code + kèm địa chỉ ô nhớ + kèm thời gian thực thi từng lệnh cho đoạn chương trình số 1.

0000, MOV R0, #0DH, Mã máy: 78 0D, 2 byte, 1 mc, 1  $\mu$ s, Gán 0Dh vào R0

0002, MOV R1, #06H, Mã máy: 79 06, 2 byte, 1 mc, 1  $\mu$ s, Gán 06h vào R1

0004, MOV A, R0, Mã máy: E8, 1 byte, 1 mc, 1  $\mu$ s, Chuyển R0 → A

0005, ADD A, R1, Mã máy: 28, 1 byte, 1 mc, 1  $\mu$ s, A = A + R1

0006, MOV P0, A, Mã máy: F5 80, 2 byte, 1 mc, 1  $\mu$ s, Xuất tổng ra P0

0008, MOV A, R0, Mã máy: E8, 1 byte, 1 mc, 1  $\mu$ s, Đưa lại R0 vào A

0009, CLR C, Mã máy: C3, 1 byte, 1 mc, 1  $\mu$ s, Xóa cờ Carry

000A, SUBB A, R1, Mã máy: 98, 1 byte, 1 mc, 1  $\mu$ s, A = A - R1 - Carry

000B, MOV P1, A, Mã máy: F5 90, 2 byte, 1 mc, 1  $\mu$ s, Xuất hiệu ra P1

000D, MOV A, R0, Mã máy: E8, 1 byte, 1 mc, 1  $\mu$ s, Đưa R0 vào A để nhân

000E, MOV B, R1, Mã máy: F9, 1 byte, 1 mc, 1  $\mu$ s, Đưa R1 vào B

000F, MUL AB, Mã máy: A4, 1 byte, 4 mc, 4  $\mu$ s, Nhân A×B → A (LSB), B (MSB)

0010, MOV P2, A, Mã máy: F5 A0, 2 byte, 1 mc, 1  $\mu$ s, Xuất phần thấp ra P2

0012, MOV P3, B, Mã máy: F5 B0, 2 byte, 1 mc, 1  $\mu$ s, Xuất phần cao ra P3

0014, JMP \$, Mã máy: 80 FE, 2 byte, 2 mc, 2  $\mu$ s, Lặp vô hạn

END, , , , Kết thúc chương trình

\*Tổng hợp code trong bài:

```

main.asm          main2.asm          main3.asm          main4.asm
1 //4.1 chuong trinh 1      1 //4.2 chuong trinh 2      1 //4.3 chuong TRINH 3      1 //4.5 chuong trinh 05
2 ORG 0000H          2 ORG 0000H          2 ORG 0000H          2 //sram1] 32kb (0000h - 7ffffh = 0 - 32767 )
3 MOV R0, #0AH          3 MOV DPTR, #TABLE          3 MOV 30H, #05H          3 //sram2] 32kb (8000h - fffffh = 32768 - 65535)
4 MOV R1, #5           4 MOV A, #00H          4 MOV R1, 30H          4 ORG 00H
5 //ADD             5 MOVC A, @A+DPTR          5 MOV R2, #00H          5 MAIN:
6 MOV A, R0           6 MOV R0, a          6 CLR A              6 RAMLOC EQU 07FFAH
7 ADD A, R1           7 MOVC A, @A+DPTR          7 MOV R1, 30H          7 COUNT EQU 8
8 MOV P0, A           8 MOV R1, A          8 CLR C              8 MOV DPTR, #RAMLOC
9 //SUB             9 MOVC A, @A+DPTR          9 SUBB A, R1          9 MOV R2, A
10 MOV A, R0          10 MOV R0, a          10 SUBB A, R1          10 DJNZ R1, LOOP
11 CLR C             11 MOV A, R0          11 MOV A, R0          11 MOV A, R1
12 SUBB A, R1          12 MOV A, #01H          12 ADD A, R2          12 ADD A, R2
13 MOV P1, A           13 MOVC A, @A+DPTR          13 MOV R2, A          13 MOV R2, A
14 //MUL             14 MOV R1, A          14 DJNZ R1, LOOP          14 DJNZ R1, LOOP
15 MOV A, R0           15 //SUB R0 - R1          15 MOV 40H, R2          15 MOV P1, R2
16 MUL AB             16 MOV A, R0          16 MOV P1, A          16 JMP $
17 MOV B, R1           17 CLR C              17 ORG 0100H          17 JMP $
18 MUL AB             18 SUBB A, R1          18 TABLE:           18 DB 0AH, 03H
19 MOV P2, A           19 MOV A, #01H          19 END               19 END
20 MOV P3, B           20 //XUATKETQUA RA PORT 1
21 //TABLE HANG SO TRONG ROM
22 //ORG 0100H
23 //TABLE:
24 //DB 0AH, 03H
25 //END
26 //DJNZ R1, LOOP
27 //MOV 40H, R2
28 //MOV P1, R2
29 //JMP $
30 //END
31 //END

main2.asm          main3.asm          main4.asm
1 //4.4 CHUONG TRINH 4      1 //4.5 chuong trinh 05      1 //4.5 chuong trinh 05
2 ORG 0000H          2 //sram1] 32kb (0000h - 7ffffh = 0 - 32767 )      2 //sram1] 32kb (0000h - 7ffffh = 0 - 32767 )
3 //sram2] 32kb (8000h - fffffh = 32768 - 65535)      3 //sram2] 32kb (8000h - fffffh = 32768 - 65535)
4 ORG 00H          4 ORG 00H          4 ORG 00H
5 MAIN:           5 MAIN:           5 MAIN:
6 RAMLOC EQU 07FFAH      6 RAMLOC EQU 07FFAH      6 RAMLOC EQU 07FFAH
7 COUNT EQU 8          7 COUNT EQU 8          7 COUNT EQU 8
8 MOV DPTR, #RAMLOC      8 MOV DPTR, #RAMLOC      8 MOV DPTR, #RAMLOC
9 MOV R3, #COUNT          9 MOV R3, #COUNT          9 MOV R3, #COUNT
10 MOV A, #01H          10 MOV A, #01H          10 MOV A, #01H
11 DJNZ R3, repl          11 DJNZ R3, repl          11 DJNZ R3, repl
12 repl:             12 repl:             12 repl:
13 MOVX @DPTR, A          13 MOVX @DPTR, A          13 MOVX @DPTR, A
14 RL A               14 RL A               14 RL A
15 INC DPTR            15 INC DPTR            15 INC DPTR
16 DJNZ R3, repl          16 DJNZ R3, repl          16 DJNZ R3, repl
17 MOV DPTR, #RAMLOC      17 MOV DPTR, #RAMLOC      17 MOV DPTR, #RAMLOC
18 MOV R3, #COUNT          18 MOV R3, #COUNT          18 MOV R3, #COUNT
19 rep2:             19 rep2:             19 rep2:
20 MOVX A, @DPTR          20 MOVX A, @DPTR          20 MOVX A, @DPTR
21 MOV P1, A             21 MOV P1, A             21 MOV P1, A
22 CALL DELAY            22 CALL DELAY            22 CALL DELAY
23 INC DPTR            23 INC DPTR            23 INC DPTR
24 DJNZ R3, rep2          24 DJNZ R3, rep2          24 DJNZ R3, rep2
25 MOV DPTR, #RAMLOC      25 MOV DPTR, #RAMLOC      25 MOV DPTR, #RAMLOC
26 MOV R3, #COUNT          26 MOV R3, #COUNT          26 MOV R3, #COUNT
27 JMP MAIN             27 JMP MAIN             27 JMP MAIN
28 DELAY:             28 DELAY:             28 DELAY:
29 MOV R1, #200           29 MOV R1, #200           29 MOV R1, #200
30 LAP1:             30 LAP1:             30 LAP1:
31 MOV R0, #200           31 MOV R0, #200           31 MOV R0, #200
32 LAP:               32 LAP:               32 LAP:
33 NOP                 33 NOP                 33 NOP
34 NOP                 34 NOP                 34 NOP
35 NOP                 35 NOP                 35 NOP
36 NOP                 36 NOP                 36 NOP
37 DJNZ R0, LAP           37 DJNZ R0, LAP           37 DJNZ R0, LAP
38 DJNZ R1, LAP1           38 DJNZ R1, LAP1           38 DJNZ R1, LAP1
39 RET                 39 RET                 39 RET
40 END               40 END               40 END

```

### [Buổi 9] Bài 08 - Nhập xuất GPIO

#### 1. Viết chương trình tạo 5 hiệu ứng led đơn (dịch, sáng dần, tắt dần, chớp tắt, sáng dần).

```

ORG 0000H
JMP MAIN
ORG 0003H
JMP INT_0
ORG 0013H

```

JMP INT\_1

ORG 0030H

MAIN:

MOV P2, #0FFH

SETB EA

SETB EX0

SETB IT0

MAIN\_LOOP:

MOV A, #0FEH

MOV R0, #8

DICH\_TRAI:

MOV P2, A

CALL DELAY1S

RL A

DJNZ R0, DICH\_TRAI

MOV A, #0FFH

MOV R0, #8

SANG\_DON:

CLR C

RLC A

MOV P2, A

CALL DELAY1S

DJNZ R0, SANG\_DON

MOV A, #00H

MOV R0, #8

TAT\_DAN:

SETB C

RRC A

MOV P2, A

CALL DELAY1S

DJNZ R0, TAT\_DAN

MOV R0, #5

CHOP\_TAT:

MOV P2, #00H

CALL DELAY1S

MOV P2, #0FFH

CALL DELAY1S

DJNZ R0, CHOP\_TAT

MOV A, #0FEH

MOV R0, #8

SANG\_DAN:

MOV P2, A

CALL DELAY1S

ANL A, #0FEH

RL A

DJNZ R0, SANG\_DAN

JMP MAIN\_LOOP

DELAY1S:

MOV R3, #10

DELAY1S\_LOOP:

CALL DELAY100ms

DJNZ R3, DELAY1S\_LOOP

RET

DELAY100ms:

MOV R2, #100

D2\_LOOP:

MOV R1, #200

D1\_LOOP:

NOP

NOP

DJNZ R1, D1\_LOOP

DJNZ R2, D2\_LOOP

RET

INT\_0:

CPL P2.0

RETI

INT\_1:

CPL P2.7

RETI

END

### 3.Chạy đoạn code mẫu, xem kết quả hiển thị LED 7 đoạn, giải thích ý nghĩa

```
main.asm    main2.asm    main3.asm*
1 //code bo sung
2 SEGMENT_PORT EQU P0
3 SELECT_PORT EQU P2
4 LED7SEG DATA 30H
5 VALUE DATA 31H
6 ORG 0000H
7     JMP MAIN
8 ORG 0030H
9 MAIN:
10    MOV LED7SEG, #0
11    MOV VALUE, #0
12 LOOP:
13    MOV A, LED7SEG
14    RL A
15    RL A
16    MOV SELECT_PORT, A
17    MOV DPTR, #DIGIT_PATTERNS
18    MOV A, VALUE
19    MOVC A, @A+DPTR
20    MOV SEGMENT_PORT, A
21    CALL DELAY_2MS
22    INC LED7SEG
23    MOV A, LED7SEG
24    ANL A, #07H
25    MOV LED7SEG, A
26    INC VALUE
27    MOV A, VALUE
28    ANL A, #07H
29    MOV VALUE, A
30    JMP LOOP
31 DELAY_2MS:
32    MOV R2, #4
33 D2_OUTER:
34    MOV R1, #250
35 D2_INNER:
36    DJNZ R1, D2_INNER
37    DJNZ R2, D2_OUTER
38    RET
39 DIGIT_PATTERNS:
40    DB 03FH, 006H, 05BH, 04FH, 066H, 06DH, 07DH, 007H, 07FH, 06FH
41    FEND
```

\*\*\*Kết quả hiển thị LED 7 đoạn.

Hiển thị tuần tự các giá trị 0 → 7 trên 8 LED 7 đoạn.

Các LED sáng luân phiên (dạng quét), mỗi LED hiển thị một giá trị.

Vì điều khiển sử dụng:

P0 để xuất dữ liệu 7 đoạn (a–g).

P2 để chọn LED ( thông qua 74HC138).

**Tổng hợp code trong bài :**

```

main.asm
1 //5.1 chuong trinh 1
2 ORG 00H
3 BEGIN_ENTRY:
4 SETB P2.3
5 MAIN_LOOP:
6 CPL P2.3
7 CALL DELAYS
8 JMP MAIN_LOOP
9
10 DELAYS:
11 MOV R3, #5
12 S1BACK:
13 CALL DELAY200ms
14 DJNZ R3, S1BACK
15 RET
16
17 DELAY200ms:
18 MOV R2, #200
19 LAP2:
20 MOV R1, #200
21 LAP1:
22 NOP
23 NOP
24 NOP
25 NOP
26 NOP
27 DJNZ R1, LAP1
28 DJNZ R2, LAP2
29 RET
30
31 END

main2.asm
1 //5.2 chuong trinh 2
2 ORG 00H
3
4 START:
5 SETB P2.0
6 SETB P3.0
7 SETB P3.1
8
9 MAIN_LOOP:
10 JB P3.0, CHECK_OFF
11 CLR P2.0
12 SJMP MAIN_LOOP
13
14 CHECK_OFF:
15 JB P3.1, MAIN_LOOP
16 SETB P2.0
17 SJMP MAIN_LOOP
18
19 END

main3.asm*
1 //code bo sung
2 SEGMENT_PORT EQU P0
3 SELECT_PORT EQU P2
4 LED7SEG DATA 30H
5 VALUE DATA 31H
6 ORG 0000H
7 JMP MAIN
8 ORG 0030H
9
10 MOV LED7SEG, #0
11 MOV VALUE, #0
12 LOOP:
13 MOV A, LED7SEG
14 RL A
15 RL A
16 MOV SELECT_PORT, A
17 MOV DPTR, #DIGIT_PATTERNS
18 MOV A, VALUE
19 MOVC A, @+DPTR
20 MOV SEGMENT_PORT, A
21 CALL DELAY_2MS
22 INC LED7SEG
23 MOV A, LED7SEG
24 ANL A, #07H
25 MOV LED7SEG, A
26 INC VALUE
27 MOV A, VALUE
28 ANL A, #07H
29 MOV VALUE, A
30 JMP LOOP
31 DELAY_2MS:
32 MOV R2, #4
33 D2_OUTER:
34 MOV R1, #250
35 D2_INNER:
36 DJNZ R1, D2_INNER
37 DJNZ R2, D2_OUTER
38 RET
39 DIGIT_PATTERNS:
40 DB 03FH, 06H, 05BH, 04FH, 06EH, 06DH, 07DH, 007H, 07FH, 06FH
41 END

```

### [Buổi 10] Bài 09 - Timer/Counter

#### 1. Timer khác Counter như thế nào trong 8051?

- Trong vi điều khiển 8051, Timer dùng xung nội (fosc/12) để đo thời gian, còn Counter dùng xung ngoài (từ chân T0/T1) để đếm các sự kiện xảy ra bên ngoài.

#### 2.Trong Mode 1, Timer hoạt động như thế nào ? Độ dài đếm là bao nhiêu bit?

- Trong Mode 1, Timer của 8051 hoạt động ở chế độ đếm 16 bit, nghĩa là bộ đếm gồm THx (8 bit cao) và TLx (8 bit thấp) ghép lại thành bộ đếm 16 bit có khả năng đếm từ 0000H đến FFFFH (0–65535) trước khi tràn.

#### 3.Bit nào trong thanh ghi TCON được dung để khởi động Timer 0 ?

- Bit TR0 trong thanh ghi TCON được dùng để khởi động hoặc dừng Timer 0.

#### 4.Tạo trễ 1ms bằng Timer 0 ở chế độ Mode 1, tần số thạch anh 12Mhz

;CHƯƠNG TRÌNH TẠO TRỄ 1ms - DÙNG TIMER0, MODE 1

Thạch anh: 12MHz → 1 chu kỳ máy = 1μs

Tính toán:

$$65536 - (1\text{ms} \times 1\text{MHz}) = 65536 - 1000 = 64536 = 0xFC18$$

\*\*\*chương trình \*\*\*

DELAY\_1MS:

```
MOV TMOD, #01H      ; GATE0=0, chọn Timer0 Mode 1 (16-bit)
MOV TH0, #0FCH      ; Nạp giá trị ban đầu: 0xFC18
MOV TL0, #018H
SETB TR0           ; Bắt đầu chạy Timer 0
```

WAIT:

```
JNB TF0, WAIT      ; Chờ khi TF0 = 1 (tràn)
CLR TR0            ; Dừng Timer
CLR TF0            ; Xóa cờ tràn
RET
```

END

#### 5.Viết chương trình nhấp nháy LED trên P1.0, với chu kỳ 500ms sử dụng Timer 1.

ORG 0000H

MAIN:

SETB P2.5

LOOP:

CPL P2.5

CALL DELAY\_250MS

JMP LOOP

DELAY\_250MS:

MOV R7, #25

DELAY\_LOOP:

```
CALL DELAY_10MS  
DJNZ R7, DELAY_LOOP  
RET
```

; chương trình tạo độ trễ 10ms sử dụng timer0, mode 1

; tính toán:  $65536 - (10\text{ms} \times 1\text{Mhz}) = 65536 - 10000 = 55536 = 0xD8F0$

DELAY\_10MS:

```
MOV TMOD, #01H ; GATE0 = 0, timer 0, mode 1  
MOV TH0, #0D8H  
MOV TL0, #0F0H  
SETB TR0 ; bắt đầu chạy timer 0
```

WAIT:

JNB TF0, WAIT ; TF=0 nháy đèn nhǎn.

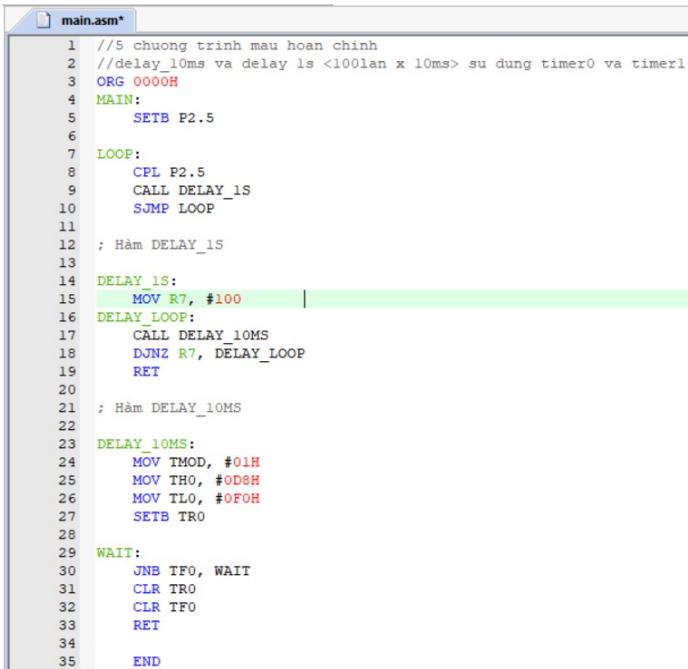
CLR TR0

CLR TF0

RET

END

**Tổng hợp code trong bài :**



```
1 //5 chương trình mua hoan chinh
2 //delay_10ms va delay ls <100lan x 10ms> su dung timer0 va timer1
3 ORG 0000H
4 MAIN:
5     SETB P2.5
6
7 LOOP:
8     CPL P2.5
9     CALL DELAY_1S
10    SJMP LOOP
11
12 ; Hàm DELAY_1S
13
14 DELAY_1S:
15     MOV R7, #100
16 DELAY_LOOP:
17     CAL DELAY_10MS
18     DJNZ R7, DELAY_LOOP
19     RET
20
21 ; Hàm DELAY_10MS
22
23 DELAY_10MS:
24     MOV TMOD, #01H
25     MOV TH0, #0D8H
26     MOV TL0, #0FOH
27     SETB TR0
28
29 WAIT:
30     JNB TF0, WAIT
31     CLR TR0
32     CLR TF0
33     RET
34
35 END
```

### [Buổi 11] Bài 10 - Ngắt

**1.Ngắt ngoài (External Interrupt) INT0/INT1 hoạt động theo cạnh hay mức? Cấu hình như thế nào?**

- Hoạt động theo cạnh hoặc mức.
- Bit IT0 (TCON.0) dùng để chọn chế độ hoạt động của ngắt INT0: nếu IT0 = 0 thì kích theo mức thấp, còn IT0 = 1 thì kích theo cạnh xuống.
- Tương tự, bit IT1 (TCON.2) điều khiển ngắt INT1: nếu IT1 = 0 thì kích theo mức thấp, còn IT1 = 1 thì kích theo cạnh xuống.

**2.Khi xảy ra ngắt, trình xử lý sẽ làm gì? Lệnh nào giúp trả lại chương trình chính?**

- Khi xảy ra ngắt, vi điều khiển tự động lưu địa chỉ lệnh kế tiếp (PC) và nhảy đến chương trình phục vụ ngắt (ISR) tương ứng để thực hiện.

- Sau khi hoàn thành ISR, lệnh RETI được dùng để trả về chương trình chính tại vị trí đã bị tạm dừng trước khi ngắt xảy ra.

**3.Bit nào trong thanh ghi IE ( Interrupt Enable) và TCON dung để bật ngắt ngoài INT0?**  
Bit EX0 (IE.0) trong thanh ghi IE = 1 → cho phép ngắt ngoài 0 hoạt động.

**4.Làm sao để cấu hình và sử dụng ngắt Timer?**

1. Cấu hình chế độ hoạt động của timer qua thanh ghi TMOD  
Mỗi Timer có 4 bit điều khiển riêng gồm các bit GATE, C/T, và hai bit chọn chế độ M1, M0.  
Bit C/T xác định Timer hoạt động như bộ định thời (0) hay bộ đếm xung ngoài (1).

Hai bit M1–M0 chọn chế độ: 13 bit (Mode 0), 16 bit (Mode 1), 8 bit tự nạp lại (Mode 2), hoặc chia Timer0 thành 2 Timer 8 bit (Mode 3).

2. Tính toán và nạp giá trị khởi tạo vào THx và TLX
3. Cho phép ngắt timer qua thanh ghi IE
4. Khởi động timer bằng cách set bit TRx
5. Viết chương trình xử lý ngắt timer

**5. Viết chương trình sử dụng cả INT0 và INT1 để điều khiển 2**

- INT0 bật LED tại P2.0
- INT0 tắt LED tại P2.0

ORG 0000H  
LJMP MAIN

ORG 0003H  
LJMP ISR\_INT0

ORG 0013H  
LJMP ISR\_INT1  
ORG 0030H  
MAIN:  
MOV P2, #0FFH  
MOV IE, #10000101B  
MOV TCON, #00000101B

HERE:  
SJMP HERE  
ISR\_INT0:  
CLR P2.0  
RETI  
ISR\_INT1:  
SETB P2.0  
RETI  
END

**6. Viết chương trình dung ngắt ngoài INT0 để tang biến đếm và hiển thị trên port P2.**

ORG 0000H  
LJMP MAIN  
ORG 0003H  
LJMP ISR\_INT0  
ORG 0030H  
MAIN:  
MOV P2, #0FFH  
MOV IE, #10000001B  
MOV TCON, #00000001B

MOV A, #00H

HERE:

MOV B, A

CPL A

MOV P2, A

MOV A, B

SJMP HERE

ISR\_INT0:

INC A

RETI

END

## 7. Viết chương trình dung ngắn Timer 1 để nháy LED P2.0 mỗi 3s.

ORG 0000H

LJMP MAIN

ORG 001BH

LJMP ISR\_TIMER1

MAIN:

SETB P2.0

MOV TMOD, #10H

MOV TH1, #0B1H

MOV TL1, #0E0H

MOV IE, #88H

SETB TR1

MOV R7, #60

LOOP:

SJMP LOOP

ISR\_TIMER1:

CLR TR1

MOV TH1, #0B1H

MOV TL1, #0E0H

CLR TF1

DJNZ R7, EXIT\_ISR

CPL P2.0

MOV R7, #60

EXIT\_ISR:

SETB TR1

RETI

END

Tổng hợp code trong bài :

```

main2.asm
1 //6.1 chuong trinh so 01
2 //chuong trinh dao trang thai led don pin p2.7, use button at p3.2
3 ORG 0000H
4 JMP MAIN
5
6 ORG 0003H
7 JMP INT0_ISR
8
9 ORG 0030H
10 MAIN:
11 SETB IT0
12 SETB EX0
13 SETB EA
14
15 SETB P2.7
16
17 LOOP:
18 JMP LOOP
19
20 INTO_ISR:
21 CPL P2.7
22 RETI
23
24 END

```

```

main3.asm
1 //6.2 chuong trinh so 2
2 //nhap nhay led p2.1 tan so 1Hz(chu ky 1s). su dung ngat timer0, mode 1 <16bit>
3 ORG 0000H
4 JMP MAIN
5
6 ORG 000BH
7 JMP TIMERO_ISR
8
9 ORG 0030H
10 MAIN:
11 MOV TMOD, #01H
12
13 MOV TH0, #3CH
14 MOV TL0, #0B0H
15
16 MOV R7, #10
17
18 SETB ET0
19 SETB EA
20
21 SETB P2.1
22
23 SETB TR0
24
25 LOOP:
26 JMP LOOP
27
28 TIMERO_ISR:
29 MOV TH0, #3CH
30 MOV TL0, #0B0H
31
32 DJNZ R7, EXIT_ISR
33 MOV R7, #10
34 CPL P2.1
35
36 EXIT_ISR:
37 RETI
38
39 END

```

```

main4.asm*
1 //6.3 chuong trinh so 3
2 //tao xung PWM su dung timer 1, mode 2(8 bit auto-reload)
3 //tan so PWM 100hz <chuky 10ms>. do phan giao 100 buoc
4 ORG 0000H
5 JMP MAIN
6 ORG 001BH
7 JMP TIMER1_ISR
8 ORG 0030H
9 MAIN:
10 MOV TMOD, #20H
11 MOV TL1, #156
12 MOV TH1, #156
13 MOV R2, #50
14 SETB ET1
15 SETB EA
16 SETB P2.0
17 SETB TRI
18 LOOP:
19 JMP LOOP
20 TIMER1_ISR:
21 INC R1
22 CLR C
23 MOV A, R1
24 SUBB A, R2
25 JC PWM_HIGH
26 CLR P2.0
27 JMP CHECK_CYCLE_END
28 PWM_HIGH:
29 SETB P2.0
30 CHECK_CYCLE_END:
31 CJNE R1, #100, EXIT_ISR
32 MOV R1, #0
33 SETB P2.0
34 EXIT_ISR:
35 RETI
36 END

```

### [Buổi 12] Bài 11 - Lập trình GPIO sử dụng ngôn ngữ C

#### 1. Ngắt là gì? Ưu điểm của việc sử dụng ngắt hay vì polling là gì?

- Phương pháp ngắt (interrupt) cho phép vi điều khiển phản ứng ngay lập tức khi có sự kiện xảy ra, như nút nhấn được nhấn mà không cần phải liên tục kiểm tra.
- Ngắt giúp vi điều khiển phản ứng nhanh và tiết kiệm thời gian CPU hơn so với phương pháp polling.

**2.Trong lập trình C cho 8051, hàm ISR (Interrupt Service Routine), được khai báo như thế nào?**

- Khai báo ISR bằng cú pháp void ten\_ham(void) interrupt n. Số n xác định loại ngắt mà hàm đó phục vụ

Ngắt	Giá trị n trong khai báo ISR
Ngắt ngoài 0 (INT0)	0
Ngắt Timer 0	1
Ngắt ngoài 1 (INT1)	2
Ngắt Timer 1	3
Ngắt nối tiếp (Serial)	4

**3.Viết chương trình sử dụng ngắt ngoài INT1 (P3.3) để tăng biến đếm mỗi khi nhấn nút, hiển thị ra cổng P2.**

```
#include <REGX51.H>
unsigned char count = 0;
void ISR_INT1(void) interrupt 2
{
    count++; // Tăng biến đếm mỗi khi nhấn nút
    P2 = ~count; // Hiển thị giá trị ra Port 2
}
void main(void)
{
    IE = 0x84; // EA=1 cho phép ngắt toàn cục, EX1=1 cho phép ngắt ngoài 1
    TCON = 0x04; // IT1=1: kích ngắt INT1 theo sườn xuống
    P2 = 0xFF; // Khởi tạo port P2 = 0
    while(1); // Chương trình chính rảnh, chờ ngắt
}
```

**Tổng hợp code trong bài :**

```

main.c //3.2 chuong trinh 1
#include <REGX51.H>
sbit LED0 = P2^0;
void delay_ms(unsigned int ms) {
    unsigned int i, j;
    for (i = 0; i < ms; i++)
        for (j = 0; j < 123; j++);
}
void main() {
    while (1) {
        LED0 = 0;
        delay_ms(500);
        LED0 = 1;
        delay_ms(500);
    }
}

main2.c //3.3 chuong trinh 2 dich 8 led don
#include <REGX51.H>
void delay_ms(unsigned int ms) {
    unsigned int i, j;
    for (i = 0; i < ms; i++)
        for (j = 0; j < 123; j++);
}
void main() {
    while (1) {
        P2 = 0xFE;
        delay_ms(200);
        P2 = 0xFD;
        delay_ms(200);
        P2 = 0xFB;
        delay_ms(200);
        P2 = 0xF7;
        delay_ms(200);
        P2 = 0xEF;
        delay_ms(200);
        P2 = 0xDF;
        delay_ms(200);
        P2 = 0xBF;
        delay_ms(200);
        P2 = 0x7F;
        delay_ms(200);
    }
}

main3.c //3.4 chuong trinh 3 dieu khien do sang led p2.2 = pwm
#include <REGX51.H>
#include <intrinsics.h>
sbit LED = P2^2;
void delay_us(unsigned int us) {
    while (us--) {
        _nop_();
    }
}
void main() {
    unsigned char i, duty_cycle;
    while (1) {
        for (duty_cycle=0; duty_cycle<100; duty_cycle++) {
            for (i = 0; i < 100; i++) {
                if (i < duty_cycle) {
                    LED = 0;
                } else {
                    LED = 1;
                }
                delay_us(10);
            }
        }
        for (duty_cycle=100; duty_cycle>0; duty_cycle--) {
            for (i = 0; i < 100; i++) {
                if (i < duty_cycle) {
                    LED = 0;
                } else {
                    LED = 1;
                }
                delay_us(10);
            }
        }
    }
}

main4.c //4.2 chuong trinh bat/tat led don bang 2 nut nhan
#include <REGX51.H>
sbit LED = P2^7;
sbit BTN_ON = P3^0;
sbit BTN_OFF = P3^1;
void delay_ms(unsigned int ms) {
    unsigned int i, j;
    for (i = 0; i < ms; i++)
        for (j = 0; j < 123; j++);
}
void main() {
    LED = 1;
    BTN_ON = 1;
    BTN_OFF = 1;
    while (1) {
        if (BTN_ON == 0) {
            delay_ms(20);
            if (BTN_ON == 0) {
                LED = 0;
                while (BTN_ON == 0);
            }
        }
        if (BTN_OFF == 0) {
            delay_ms(20);
            if (BTN_OFF == 0) {
                LED = 1;
                while (BTN_OFF == 0);
            }
        }
    }
}

```

```

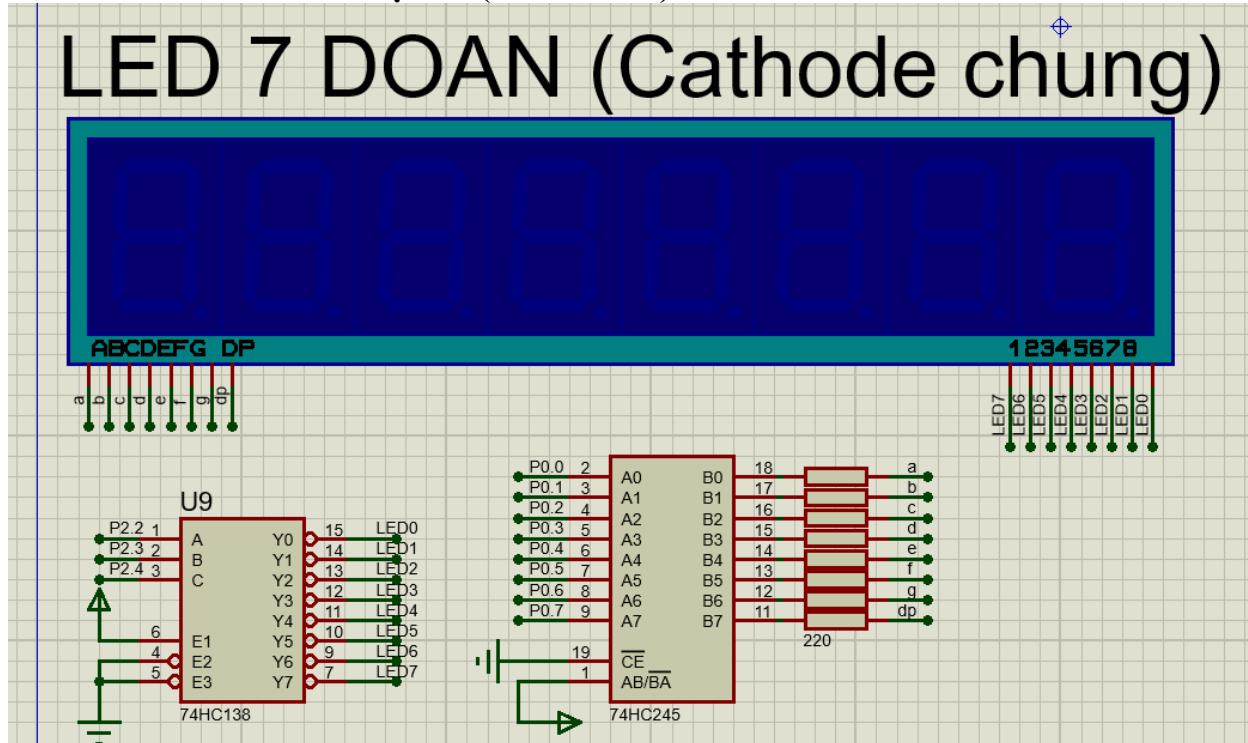
1 //5.4 ham xu ly ngat trong c
2 #include <REGX51.H>
3
4 sbit LED = P2^0;
5
6 void external0_isr(void) interrupt 0 {
7 }
8
9 void external1_isr(void) interrupt 2 {
10 }
11
12 void main(void) {
13     IT0 = 1;
14     IT1 = 1;
15
16     EX0 = 1;
17     EX1 = 1;
18     EA = 1;
19
20     while (1) {
21
22 }
23 }
24
25 }

1 //5.5 chuong trinh mau - bat/tat led don = 2 nut nhan = ngat
2 #include <REGX51.H>
3
4 sbit LED = P2^7;
5
6 void external0_isr(void) interrupt 0 {
7     LED = 1;
8 }
9 void external1_isr(void) interrupt 2 {
10     LED = 0;
11 }
12
13 void main(void) {
14     IT0 = 1;
15     IT1 = 1;
16
17     EX0 = 1;
18     EX1 = 1;
19     EA = 1;
20
21     LED = 1;
22     P3_2 = 1;
23     P3_3 = 1;
24
25     while (1) {
26
27 }
28 }

```

### [Buổi 13] Bài 12 - Lập trình Led 7 đoạn

1.Vẽ sơ đồ chi tiết 8 led 7 đoạn rời (trên Proteus)



2.Viết mã giả (pseudocode) trình bày quá trình hiển thị số 1 -> 8 lên 8 led 7 đoạn bằng phương pháp dịch.

## BẮT ĐẦU

Khai báo:

```
SEGMENT_PORT ← cổng điều khiển các đoạn LED (P0)
SELECT_PORT ← cổng chọn LED (P2)
digit_patterns[] ← mảng mã hiển thị cho các số 1 đến 8
led7seg ← 0      // chỉ vị trí LED hiện tại (0→7)
value ← 0        // giá trị số hiển thị (1→8)
```

## THIẾT LẬP BAN ĐẦU:

Tắt tất cả các LED 7 đoạn (SEGMENT\_PORT = 0x00)

## LẬP VÔ HẠN:

1. CHỌN LED cần hiển thị:

chọn LED thứ (led7seg) bằng cách xuất mã chọn ra SELECT\_PORT

2. HIỂN THỊ GIÁ TRỊ TƯƠNG ỨNG:

xuất mã hiển thị tương ứng với số (value) ra SEGMENT\_PORT

3. TRÌ HOÃN (delay):

chờ 500 mili giây để người dùng thấy rõ LED sáng

4. DỊCH (QUÉT) SANG LED TIẾP THEO:

led7seg ← led7seg + 1

nếu led7seg > 7 thì led7seg ← 0

5. TĂNG GIÁ TRỊ HIỂN THỊ:

value ← value + 1

nếu value > 7 thì value ← 0

QUAY LẠI bước 1

KẾT THÚC

**3. Viết chương trình đếm từ 000 đến 999 và lặp lại, hiển thị lên 3 led 7 đoạn.**

```
#include <REGX51.H>
#define SEGMENT_PORT P0
#define SELECT_PORT P2
unsigned char digit_patterns[] = {
    0x3F, // 0
    0x06, // 1
    0x5B, // 2
    0x4F, // 3
    0x66, // 4
    0x6D, // 5
    0x7D, // 6
    0x07, // 7
    0x7F, // 8
    0x6F // 9
```

```
};  
void delay_ms(unsigned int ms) {  
    unsigned int i, j;  
    for (i = 0; i < ms; i++) {  
        for (j = 0; j < 123; j++);  
    }  
}  
void display_digit(unsigned char led, unsigned char value) {  
    SELECT_PORT = (led & 0x07) << 2;  
    SEGMENT_PORT = digit_patterns[value];  
    delay_ms(2);  
}  
void main(void) {  
    unsigned char i, j, k;  
    while (1) {  
        for (i = 0; i < 10; i++) {  
            for (j = 0; j < 10; j++) {  
                for (k = 0; k < 10; k++) {  
                    unsigned int t;  
                    for (t = 0; t < 50; t++) {  
                        display_digit(0, k);  
                        display_digit(1, j);  
                        display_digit(2, i);  
                    }  
                }  
            }  
        }  
    }  
}
```

**Tổng hợp code trong bài :**

```

1 //5.5 chuong trinh hien thi so 3 len led 7 doan thu 6
2 #include <REGX51.H>
3
4 #define SEGMENT_PORT P0
5 #define SELECT_PORT P2
6
7 unsigned char digit_patterns[] = {
8     0x3F, // 0
9     0x06, // 1
10    0x5B, // 2
11    0x4F, // 3
12    0x66, // 4
13    0x6D, // 5
14    0x7D, // 6
15    0x07, // 7
16    0x7F, // 8
17    0x6F // 9
18 };
19
20 void main(void)
21 {
22     unsigned char led_index = 6;
23     SEGMENT_PORT = 0x00;
24     SELECT_PORT = (led_index)<<2;
25     SEGMENT_PORT = digit_patterns[3]; |
26     while (1)
27     {
28     }
29 }

```

```

1 //5.2 chuong trinh so 2 - xuat cac so tu 0 -> 7 len 8 led 7 doan <quet led>
2 #include <REGX51.H>
3 #define SEGMENT_PORT P0
4 #define SELECT_PORT P2
5 unsigned char digit_patterns[] = {
6     0x3F, // 0
7     0x06, // 1
8     0x5B, // 2
9     0x4F, // 3
10    0x66, // 4
11    0x6D, // 5
12    0x7D, // 6
13    0x07, // 7
14    0x7F, // 8
15    0x6F // 9
16 };
17 unsigned char led7seg = 0;
18 unsigned char value = 0;
19 void delay_ms(unsigned int ms) {
20     unsigned int i, j;
21     for (i = 0; i < ms; i++)
22         for (j = 0; j < 123; j++);
23 }
24 void main(void)
25 {
26     SEGMENT_PORT = 0x00;
27     while (1)
28     {
29         SELECT_PORT = ((led7seg & 0x07) << 2);
30         SEGMENT_PORT = digit_patterns[value & 0x07];
31         delay_ms(2);
32         led7seg++;
33         if (led7seg > 7) led7seg = 0; //hoac viet la led7seg &= 0x07;
34         value++;
35         if (value > 7) value = 0; //hoac viet la value &= 0x07;
36     }
37 }

```

### [Buổi 14] Bài 13 - Lập trình UART

#### 1. UART là gì? Tại sao cần UART trong hệ thống nhúng?

- UART (Universal Asynchronous Receiver Transmitter) là giao thức truyền thông nối tiếp không đồng bộ, được sử dụng rộng rãi trong các hệ thống nhúng. UART cho phép giao tiếp hai chiều, song công toàn phần (full duplex) giữa hai thiết bị chỉ với hai đường truyền dữ liệu chính:

- TX (Transmitter): đường truyền dữ liệu.

- RX (Receiver): đường nhận dữ liệu.

\*\*\*UART cần thiết trong hệ thống nhúng vì:

- Truyền dữ liệu dễ dàng: UART cho phép vi điều khiển giao tiếp với máy tính hoặc thiết bị khác chỉ bằng hai dây:

TXD (Transmit Data) → gửi dữ liệu

RXD (Receive Data) → nhận dữ liệu

- Tiết kiệm chân và tài nguyên: chỉ cần 2 dây thay vì 8 dây như giao tiếp song song.

- Hoạt động không đồng bộ: không cần tín hiệu clock chung giữa hai thiết bị → đơn giản và linh hoạt.

- Chuẩn phô biến: được dùng trong RS-232, TTL Serial, Bluetooth, GPS, GSM, USB–Serial, v.v.

## **2.UART trên 8051 sử dụng Timer nào để tạo baud rate?**

- UART trên 8051 sử dụng Timer 1 để tạo baud rate.

## **3.Các thanh ghi liên quan đến UART trong 8051 là gì? Chức năng từng thanh ghi?**

- SCON (Serial Control): Cấu hình và điều khiển UART, xác định chế độ truyền (mode 0–3), bật nhận và kiểm tra cờ ngắt truyền/nhận.
- SBUF (Serial Buffer): Thanh ghi đệm dữ liệu, ghi vào để gửi qua TX, đọc ra để nhận từ RX.
- PCON (Power Control): Điều khiển nguồn và tốc độ Baud, bit SMOD dùng để nhân đôi Baud Rate.
- TMOD (Timer Mode): Cấu hình chế độ hoạt động của Timer1, thường chọn Mode 2 để tạo Baud Rate ổn định.
- TH1 / TL1: Thanh ghi của Timer1, chứa giá trị nạp ban đầu tạo tần số Baud Rate.
- TCON (Timer Control): Điều khiển bật/tắt và theo dõi cờ tràn của Timer1 (TR1, TF1).
- IE (Interrupt Enable): Cho phép ngắt UART, bit ES bật ngắt nối tiếp và EA bật ngắt toàn cục.

## **4.Viết chương trình tạo menu hiện trên Terminal để điều khiển 4 led đơn. Viết ứng dụng gửi ký tự xuống để điều khiển bật/tắt 4 led. Menu cập nhật lại trạng thái 4 LED mỗi lần điều khiển.**

```
#include <REGX51.H>
#include <stdio.h>

sbit LED1 = P2^0;
sbit LED2 = P2^1;
sbit LED3 = P2^2;
sbit LED4 = P2^3;
// Khoi tao UART, Baud 9600bps, thach anh 11.0592MHz
void UART_Init(void) {
    TMOD = 0x20; // Timer1, mode 2 (8-bit auto reload)
    TH1 = 0xFD; // Toc do baud 9600
    SCON = 0x50; // UART mode 1, 8-bit, REN=1
    TR1 = 1; // Bat dau Timer1
```

```
}

void UART_TxChar(char c) {
    SBUF = c;
    while (TI == 0);
    TI = 0;
}

void UART_SendString(char *s) {
    while (*s) {
        UART_TxChar(*s++);
    }
}

void Display_Menu(void) {
    UART_SendString("\r\n===== MENU DIEU KHIEN LED =====\r\n");
    UART_SendString("1. BAT LED 1\r\n");
    UART_SendString("2. TAT LED 1\r\n");
    UART_SendString("3. BAT LED 2\r\n");
    UART_SendString("4. TAT LED 2\r\n");
    UART_SendString("5. BAT LED 3\r\n");
    UART_SendString("6. TAT LED 3\r\n");
    UART_SendString("7. BAT LED 4\r\n");
    UART_SendString("8. TAT LED 4\r\n");
    UART_SendString("=====*\r\n");
    UART_SendString("Trang thai LED (0 = sang, 1 = tat): ");
    UART_TxChar(LED1 ? '1' : '0');
    UART_TxChar(' ');
    UART_TxChar(LED2 ? '1' : '0');
    UART_TxChar(' ');
    UART_TxChar(LED3 ? '1' : '0');
    UART_TxChar(' ');
    UART_TxChar(LED4 ? '1' : '0');
    UART_SendString("\r\nNhap lenh: ");
}

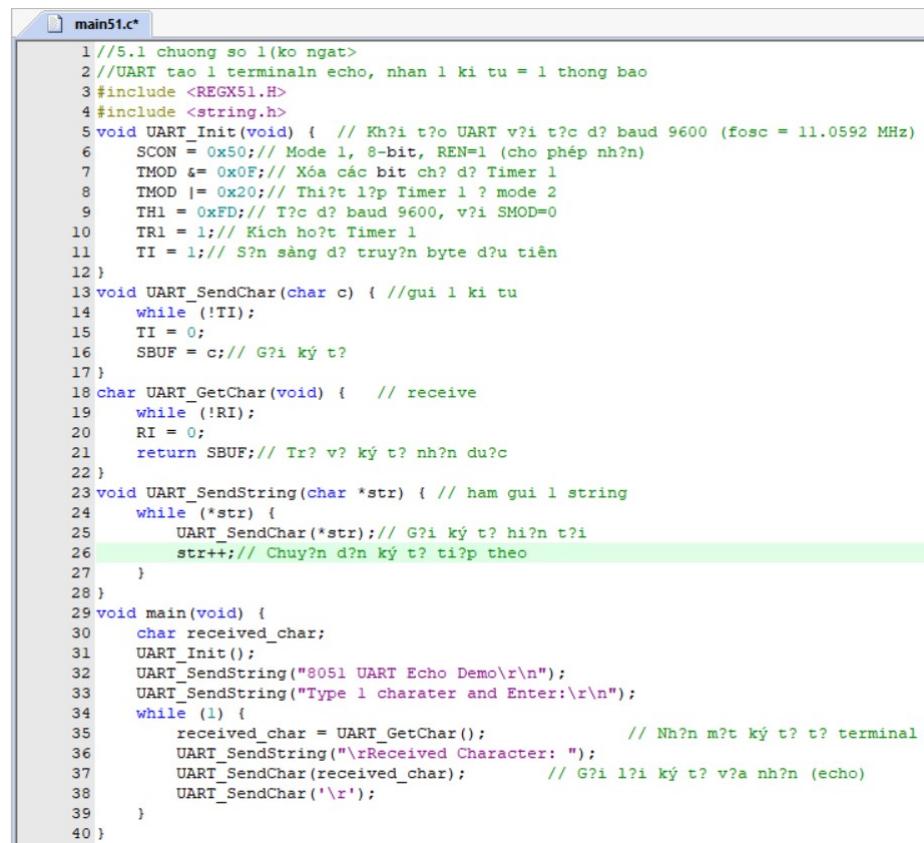
void main(void) {
    char ch;
    UART_Init();
    LED1 = LED2 = LED3 = LED4 = 1;
    Display_Menu();
    while (1) {
        while (RI == 0);
        ch = SBUF;
```

```

RI = 0;
switch (ch) {
    case '1': LED1 = 0; break;
    case '2': LED1 = 1; break;
    case '3': LED2 = 0; break;
    case '4': LED2 = 1; break;
    case '5': LED3 = 0; break;
    case '6': LED3 = 1; break;
    case '7': LED4 = 0; break;
    case '8': LED4 = 1; break;
    default:
        UART_SendString("\r\nLenh khong hop le!\r\n");
        break;
}
Display_Menu();
}
}

```

### Tổng hợp code trong bài :



```

main51.c*
1 //5.1 chuông sò 1(ko ngắt)
2 //UART tạo 1 terminalln echo, nhận 1 ký tự = 1 thông báo
3 #include <REGX51.H>
4 #include <string.h>
5 void UART_Init(void) { // Khởi tạo UART với tần số 9600 (fosc = 11.0592 MHz)
6     SCON = 0x50;// Mode 1, 8-bit,REN=1 (cho phép nhận)
7     TMOD &= 0x0F;// Xóa các bit ch? d? Timer 1
8     TMOD |= 0x20;// Thiết lập Timer 1 ? mode 2
9     TH1 = 0xFD;// T?c d? baud 9600, v?i SMOD=0
10    TR1 = 1;// Kích hoạt Timer 1
11    TI = 1;// S?n sẵn d? truy?n byte đầu tiên
12 }
13 void UART_SendChar(char c) { // Gửi 1 ký tự
14     while (!TI);
15     TI = 0;
16     SBUF = c;// Gửi ký tự
17 }
18 char UART_GetChar(void) { // Nhận
19     while (!RI);
20     RI = 0;
21     return SBUF;// Trả về ký tự nhận được
22 }
23 void UART_SendString(char *str) { // Hàm gửi 1 string
24     while (*str) {
25         UART_SendChar(*str);// Gửi ký tự hiện tại
26         str++; // Chuẩn bị ký tự tiếp theo
27     }
28 }
29 void main(void) {
30     char received_char;
31     UART_Init();
32     UART_SendString("8051 UART Echo Demo\r\n");
33     UART_SendString("Type 1 character and Enter:\r\n");
34     while (1) {
35         received_char = UART_GetChar(); // Nhận ký tự từ terminal
36         UART_SendString("\rReceived Character: ");
37         UART_SendChar(received_char); // Gửi lại ký tự vừa nhận (echo)
38         UART_SendChar('\r');
39     }
40 }

```

```
main51.c main52.c
1 //5.2 chương trình số 2 <co ngat receive>, UART g?i ki tu terminal, '1' = led on, khac= led off
2 #include <REGX51.H>
3 #include <stdio.h>
4 #include <string.h>
5 sbit LED = P2^0;
6 volatile bit flag = 0;           // C? b?o c? d? li?u m?i
7 volatile unsigned char recvData; // D? li?u nh?n du?c
8 void UART_Init(void) { // H?m kh?i t?o UART v?i t?c d? baud 9600// (fosc = 11.0592 MHz)
9     SCON = 0x50; TMOD |= 0x0F; TMOD |= 0x20; TH1 = 0xFD; TR1 = 1; TI = 1;
10 }
11 void UART_SendChar(char c) {
12     while (!TI); TI = 0; SBUF = c;
13 }
14 void UART_SendString(char *str) {
15     while (*str) { // L?p cho d?n khi g?p k? t? NULL
16         UART_SendChar(*str); str++;
17     }
18 }
19 // ES = 1; // Cho phép ng?t UART EA = 1; // Cho phép ng?t toàn c?c
20 void UART_EnableInterrupt(void) {
21     ES = 1; EA = 1;
22 }
23 void UART_ISR(void) interrupt 4 {
24     if(RI) {
25         RI = 0; recvData = SBUF; flag = 1;
26     }
27 void main(void) {
28     LED = 1; UART_Init(); UART_EnableInterrupt();
29     while(1)
30     {
31         if(flag==1)
32         {
33             if(recvData=='1')
34                 LED = 0;           // B?t LED
35             else
36                 LED = 1;           // T?t LED
37             UART_SendString("\rOK\r");
38             flag=0;
39         } //Do some other codes...
40 }}
```