

Lab 6

Program Flow Control

Objectives

After completing this lab,

- Students will be able to transfer control to any label or address unconditionally.
- Students will be able to transfer control to any label or address based on some conditions.
- Students will be able to implement customized loops instead of using built-in loops.
- Students will be able to use opposite conditional jumps to reduce number of jump instructions inside loop.
- Students will be able to make a long conditional jump using short conditional jump instructions.

Control in a program can be unconditionally or conditionally transferred to any location based on status flags. There are instructions for transferring control.

Unconditional Jump

The basic syntax of JMP instruction:
JMP label

The "JMP" instruction transfers control to the address or label followed by it. This instruction can move control within or outside of the current code segment. In the following table, example 1 shows the jump inside the current code segment, called a "near jump," and example 2 shows the jump outside the current code segment, called a "long jump." For a short jump, the address that comes after the "JMP" instruction will be an offset. For a long jump, the address will be a logical address made up of a segment address and an offset.

Example-1: Transferring control to label within same code segment.	Example-2: Transferring control to address outside current code segment.
<pre>.model small .data .code Mov ax,@data Mov ds,ax Jmp Label1 Mov AX, BX Mov CX, DX Lable1: Mov AX,1 Mov BX,2 .exit</pre>	<pre>.model small .data .code Mov ax,@data Mov ds,ax Jmp 0x07C0:0x0000 Mov AX, BX Mov CX, DX .exit</pre>

Conditional Jumps

Unlike JMP instruction that does an unconditional jump, there are instructions that do a conditional jump (jump only when some conditions are in act). These instructions are divided in three groups, first group just test single flag, second compares numbers as signed, and third compares numbers as unsigned. All these three groups use status flags to jump.

We know that the status flags are modified because of ALU instructions. Therefore, to compare two numbers, we usually perform a subtraction operation on these two numbers. We are not concerned with the result of the subtract operation but with the status flags that are updated. Therefore, instead of the "SUB" instruction, the "CMP" instruction is used, which discards the result and keeps the status of flags.

The basic syntax of CMP instruction:

CMP Destination Operand, Source Operand

Jump instructions that test single flag

The following table shows the conditional jump instructions that jump based on the value of a single flag. In the instruction column, some rows contain more than one instruction that does the same thing. They are even assembled into the same machine code. **JE** will be assembled as **JZ**, **JC** will be assembled as **JB**, and so on. These names are used to make programs easier to understand, code, and remember.

Instruction	Description	Condition	Opposite Instruction
JZ , JE	Jump if Zero (Equal).	ZF = 1	JNZ, JNE
JC , JB, JNAE	Jump if Carry (Below, Not Above Equal).	CF = 1	JNC, JNB, JAE
JS	Jump if Sign.	SF = 1	JNS
JO	Jump if Overflow.	OF = 1	JNO
JPE, JP	Jump if Parity Even.	PF = 1	JPO
JNZ , JNE	Jump if Not Zero (Not Equal).	ZF = 0	JZ, JE
JNC , JNB, JAE	Jump if Not Carry (Not Below, Above Equal).	CF = 0	JC, JB, JNAE
JNS	Jump if Not Sign.	SF = 0	JS
JNO	Jump if Not Overflow.	OF = 0	JO

JPO, JNP	Jump if Parity Odd (No Parity).	PF = 0	JPE, JP
----------	------------------------------------	--------	---------

Jump instructions for signed numbers

To compare two signed numbers, the instructions shown in the following table are used right after the 'CMP' instruction.

Instruction	Description	Condition	Opposite Instruction
JE, JZ	Jump if Equal (=). Jump if Zero.	ZF = 1	JNE, JNZ
JNE, JNZ	Jump if Not Equal (<>). Jump if Not Zero.	ZF = 0	JE, JZ
JG, JNLE	Jump if Greater (>). Jump if Not Less or Equal (not <=).	ZF = 0 and SF = OF	JNG, JLE
JL, JNGE	Jump if Less (<). Jump if Not Greater or Equal (not >=).	SF <> OF	JNL, JGE
JGE, JNL	Jump if Greater or Equal (>=). Jump if Not Less (not <).	SF = OF	JNGE, JL
JLE, JNG	Jump if Less or Equal (<=). Jump if Not Greater (not >).	ZF = 1 or SF <> OF	JNLE, JG

Jump instructions for unsigned numbers

To compare two unsigned numbers, the instructions shown in the following table are used right after the 'CMP' instruction.

Instruction	Description	Condition	Opposite Instruction
JE, JZ	Jump if Equal (=). Jump if Zero.	ZF = 1	JNE, JNZ
JNE, JNZ	Jump if Not Equal (<>). Jump if Not Zero.	ZF = 0	JE, JZ
JA, JNBE	Jump if Above (>). Jump if Not Below or Equal (not <=).	CF = 0 and ZF = 0	JNA, JBE

JB, JNAE, JC	Jump if Below (<). Jump if Not Above or Equal (not >=). Jump if Carry.	CF = 1	JNB, JAE, JNC
JAЕ, JNB, JNC	Jump if Above or Equal (>=). Jump if Not Below (not <). Jump if Not Carry.	CF = 0	JNAE, JB
JBE, JNA	Jump if Below or Equal (<=). Jump if Not Above (not >).	CF = 1 or ZF = 1	JNBE, JA

Limitation of Conditional Jump instructions

- All conditional jumps have one big limitation, unlike **JMP** instructions, they are short (one-byte jumps having a range from -128 to 127 bytes). However, we can easily avoid this limitation using a cute trick:
 - Get conditional jump instruction from the tables above and make it jump to *label_X*.
 - Under that *label_X*, Use **JMP** instructions to jump to the desired location.
- Emu8086 uses this trick implicitly for conditional jumps.

Program 1: To swap values of AX and BX registers if AX < BX.

1a: Program with straight conditional jumps.	1b: Program with opposite conditional jump to reduce jumps.
<pre>.model small .data .code Mov ax,@data Mov ds,ax Mov ax,5 Mov bx,10 Cmp ax,bx JS swap Jmp exit_cmp Swap: XCHG ax,bx exit_cmp: .exit</pre>	<pre>.model small .data .code Mov ax,@data Mov ds,ax Mov ax,5 Mov bx,10 Cmp ax,bx JNS exit_cmp Swap: XCHG ax,bx exit_cmp: .exit</pre>

Program 2: To iterate a loop while AX < BX using conditional jump for unsigned numbers.

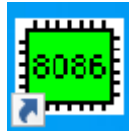
1a: Program with straight conditional jumps.	1b: Program with opposite conditional jump to reduce jumps.
<pre>.model small .data .code Mov ax,@data Mov ds,ax Mov ax,5 Mov bx,10 compare: cmp ax,bx JB iterate jmp exit_loop iterate: inc ax jmp compare exit_loop: .exit</pre>	<pre>.model small .data .code Mov ax,@data Mov ds,ax Mov ax,5 Mov bx,10 compare: cmp ax,bx JAE exit_loop iterate: inc ax jmp compare exit_loop: .exit</pre>

Program 3: To iterate a loop while AX < BX using conditional jump for signed numbers.

1a: Program with straight conditional jumps.	1b: Program with opposite conditional jump to reduce jumps.
<pre>.model small .data .code Mov ax,@data Mov ds,ax Mov ax,-5 Mov bx,0 compare: cmp ax,bx JL iterate jmp exit_loop iterate: inc ax jmp compare exit_loop: .exit</pre>	<pre>.model small .data .code Mov ax,@data Mov ds,ax Mov ax,-5 Mov bx,0 compare: cmp ax,bx JGE exit_loop iterate: inc ax jmp compare exit_loop: .exit</pre>

Emu8086 Tutorial Step by Step

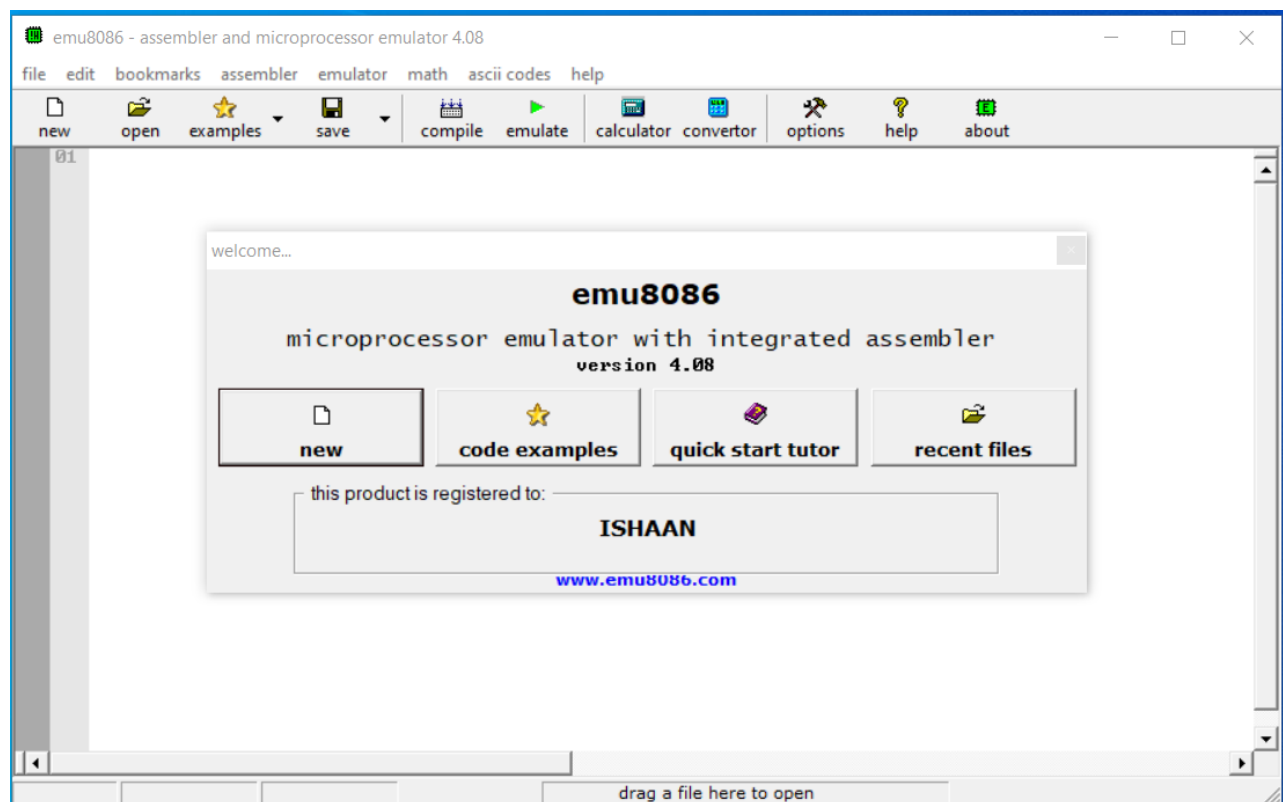
Step-1:



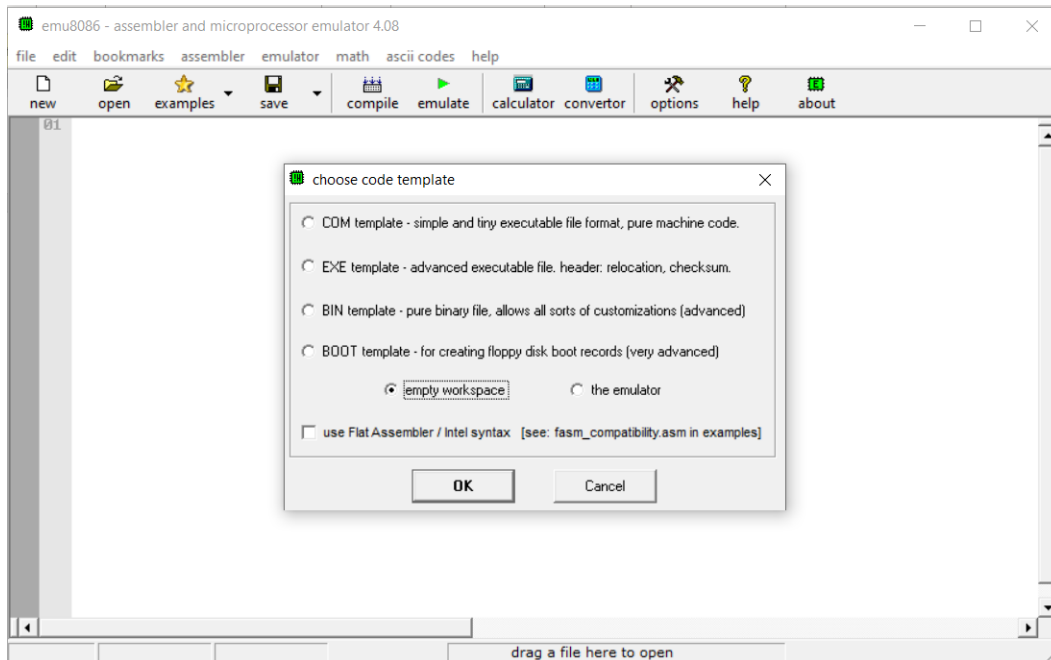
Double click on the icon on the desktop

Step-2:

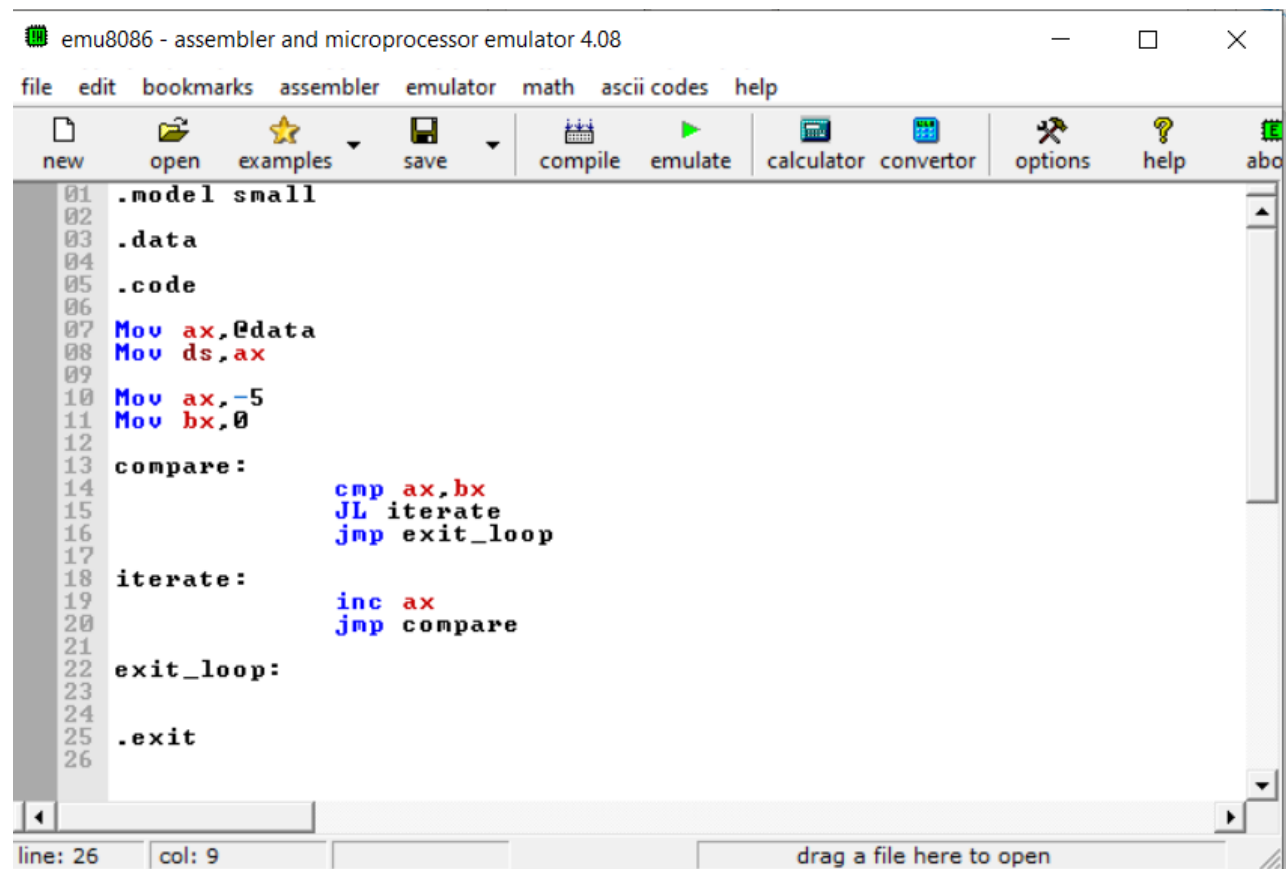
The following window will appear. Click on new.



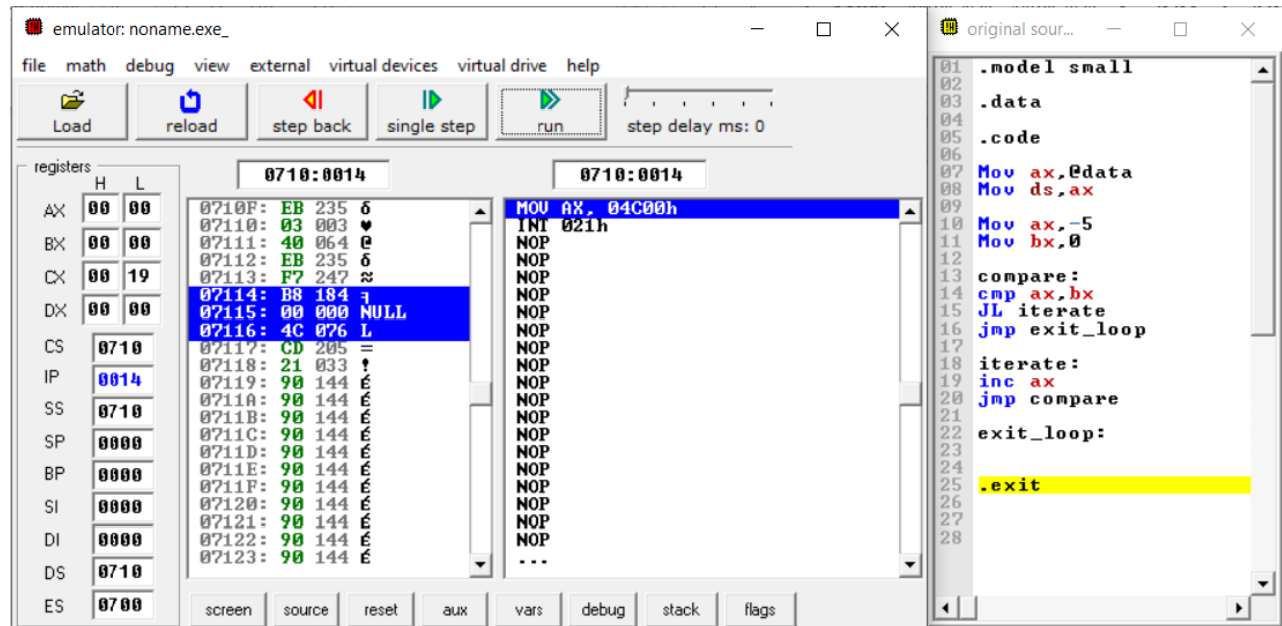
Step-3: Click on empty workspace and press OK.



Step-4: Type the code given in program-3a above and click on emulate.



Step-5: Keep clicking on "Single Step" to execute program instructions one by one and observe the register values side by side and the number of times the loop is iterated. Stop clicking "Single Step" when the ".exit" is highlighted to observe the final value of the AX register.



Practice Exercise

Task-1

Write a program that declares and initializes an array of 20 elements and then calculates the number of occurrences of a specific number in the array.

Task-2

Write a program that declares and initializes a word-type array of 20 elements and sorts it using any sorting algorithm of your choice.