Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

AY: 2024-25

Class:	**SE**	**Semester:**	**IV**
Course Code:	**CSL404**	**Course Name:**	**Microprocessor Lab**

Name of Student:	**Bhagyashri Kaleni Sutar**
Roll No. :	**75**
Experiment No.:	**5**
Title of the Experiment:	**Program to find maximum number from given array**
Date of Performance:	**10/02/2025**
Date of Submission:	**17/02/2025**

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty : Ms. Sweety Patil

Signature :

Date:

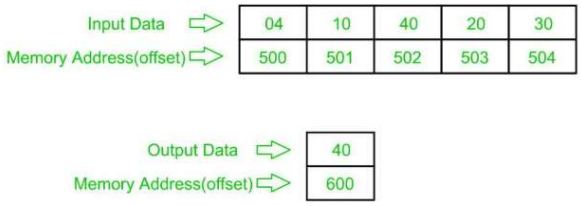
Aim: Assembly Language Program to find maximum number from a given array.

Theory: It is a simple and straightforward task that can be easily implemented in any programming language. It is a common operation used in many algorithms and applications, such as finding the maximum value in a data set or determining the winner of a game. It is a fast operation that can be completed in O(n) time complexity, where n is the number of elements in the array.

Algorithm:

1. Load data from offset 500 to register CL and set register CH to 00 (for count).
2. Load first number(value) from next offset (i.e 501) to register AL and decrease count by 1.
3. Now compare value of register AL from data(value) at next offset, if that data is greater than value of register AL then update value of register AL to that data else no change, and increase offset value for next comparison and decrease count by 1 and continue this till count (value of register CX) becomes 0.
4. Store the result (value of register AL) to memory address 2000 : 600.

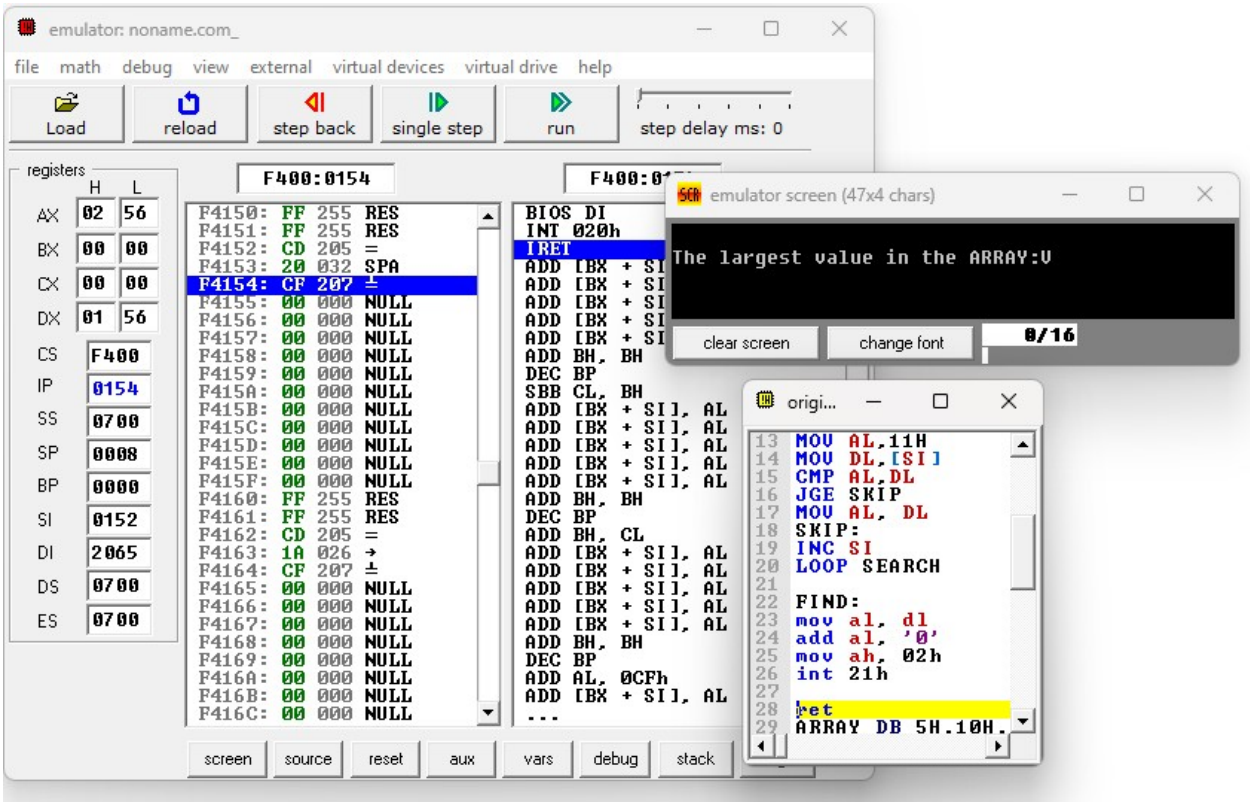
Example:



Code:

```
org 100h
M1 db 10,13, 'The largest value in the ARRAY:$'
buff db 88
LEA SI,ARRAY
MOV CX,10

mov al,00h
lea dx, M1
mov ah, 09h
int 21h

SEARCH:
MOV AL,11H
MOV DL,[SI]
CMP AL,DL
JGE SKIP
MOV AL, DL
SKIP:
INC SI
LOOP SEARCH

FIND:
mov al, dl
add al, '0'
mov ah, 02h
int 21h

ret
ARRAY DB 5H,10H,3H,9H, 15H,14H,12H,19H,20H,56H
```

Output:



Conclusion:

1. Explain the instructions AAS, IMUL, IDIV

Ans:--1. **AAS (ASCII Adjust After Subtraction**

Purpose: The `AAS` instruction is used to adjust the result of a subtraction in the context of ASCII-encoded decimal (BCD) numbers. It is used when working with **ASCII arithmetic** after a subtraction operation (usually involving the `SUB` or `SBB` instruction). It adjusts the result to ensure the resulting number is a valid ASCII decimal digit (0–9).

Example:-

```
; Subtraction of two ASCII values

mov al, '7'     ; AL = 0x37 (ASCII '7')

sub al, '5'     ; AL = 0x02 (invalid ASCII value)

aas             ; Adjust AL to get a valid ASCII result (AL = 0x02 - 0x06 =
0xF6)
```

2. IMUL (Integer Multiply)

Purpose: The `IMUL` instruction performs **signed integer multiplication**. It multiplies two signed integers and stores the result in the appropriate register. The `IMUL` instruction is used for multiplying two signed numbers, unlike `MUL`, which is used for unsigned integers.

Example:

```
; Example 1: IMUL with two operands

mov ax, -4      ; AX = -4

imul bx, ax, 3  ; BX = AX * 3 = -4 * 3 = -12

; Result: BX = 0xFFF4 (signed 16-bit value -12)



; Example 2: IMUL with a single operand

mov ax, 5       ; AX = 5

imul 3          ; AX = AX * 3 = 5 * 3 = 15

; Result: AX = 15
```

3. IDIV (Integer Divide)

Purpose: The `IDIV` instruction performs **signed integer division**. It divides the contents of the accumulator (`AX` for 16-bit operations or `EAX` for 32-bit operations) by a specified operand and produces both a quotient and a remainder.

Example

```
; Example 1: 16-bit signed division

mov ax, -15    ; AX = -15 (dividend)

mov bl, 4      ; BL = 4 (divisor)

idiv bl        ; AX / BL -> quotient in AL, remainder in AH

; Result: AL = -3 (quotient), AH = 1 (remainder)



; Example 2: 32-bit signed division
```

```
mov eax, -100  ; EAX = -100 (dividend)

mov ebx, 6     ; EBX = 6 (divisor)

idiv ebx       ; EAX / EBX -> quotient in AL, remainder in DX

; Result: AL = -16 (quotient), DX = 4 (remainder)
```

2. Explain the instructions AAM, AAD, CBW

Ans:-1. **AAM (ASCII Adjust AX After Multiply)**

Purpose: The `AAM` instruction is used in **ASCII arithmetic** after a multiplication operation. It adjusts the result in the `AX` register when multiplying two ASCII-encoded decimal (BCD) numbers to ensure the result is in a valid ASCII format.

Example

```
; Example of AAM usage:

mov al, '7'       ; AL = 0x37 (ASCII '7')

mov bl, '5'       ; BL = 0x35 (ASCII '5')

imul al, bl       ; AX = 0x111 (111 in decimal)

aam               ; AX = 0x11 (tens) and AL = 0x01 (ones)

; After AAM, AX = 0x11 (11 in hexadecimal, tens digit in AH and ones digit
in AL)
```

2. AAD (ASCII Adjust AX Before Division)

Purpose: The `AAD` instruction is used before performing **ASCII division**. It prepares the `AX` register for division by adjusting the contents so that it correctly represents a BCD (Binary-Coded Decimal) number.

Example

```
; Example of AAD usage:

mov al, '2'       ; AL = 0x32 (ASCII '2')

mov ah, '5'       ; AH = 0x35 (ASCII '5')

; AX = 0x5232 (combined value of '25' in BCD format)
```

```
aad                 ; AX = 0x52 (multiply by 10)

; Now AX = 0x52, which is correctly adjusted for BCD division
```

3. CBW (Convert Byte to Word)

Purpose: The `CBW` instruction is used to **extend a signed byte** (8-bit value) to a signed word (16-bit value). It takes the value in the `AL` register (which is an 8-bit value) and sign-extends it into the `AX` register (which is a 16-bit value), preserving the sign of the number.

Example

```
; Example of CBW usage:

mov al, -5          ; AL = 0xFB (signed byte -5)

cbw                 ; AX = 0xFFFB (sign-extended to 16 bits)

; After CBW, AX = 0xFFFB (the sign of AL is preserved in AX)
```