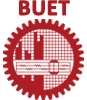
**BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY**



**Department of Electrical and Electronic Engineering**

**Course No. :** EEE 416

**Course Title:** Microprocessor and Interfacing Laboratory

**Logical Instructions and Jump Commands in Assembly Language**

**Name:** Mir Sayeed Mohammad

**ID:** 1606003

**Level:** 4

**Term:** 1

**Section:** A

**Submission Deadline:** 14 - 3 -2021

**Exercise Part 1 (a)**

**Assembly Code:**

CODE SEGMENT

ASSUME CS:CODE, DS:CODE

MOV BX, 3256H

MOV CX, 1554H

AND CX, BX

HLT

CODE ENDS

END

**Analysis:**

In the CX register, 1554H value is loaded. In the following instruction, the contents of CX and BX register are logically multiplied and the result put inside the CX register, giving the value of 1054H in CX register.

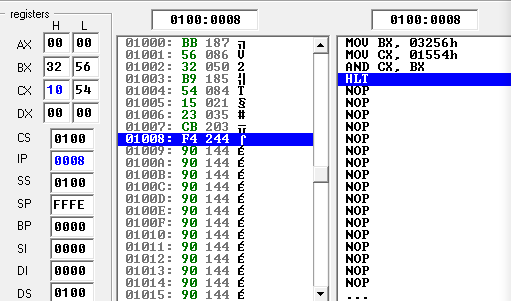


Fig: Execution of AND operation

**Exercise Part 1 (b)**

**Assembly Code:**

CODE SEGMENT

ASSUME CS:CODE, DS:CODE

MOV BX, 3256H

MOV CX, 1554H

XOR CX, BX

HLT

CODE ENDS

END

**Analysis:**

XOR operation is performed on the contents of CX and BX register, and the result = 2702H is stored in the CX register

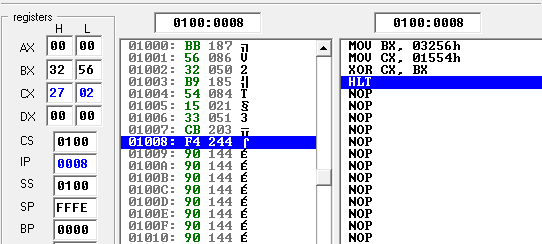


Fig: Execution of XOR operation

**Exercise Part 1 (c)**

**Assembly Code:**

CODE SEGMENT

ASSUME CS:CODE, DS:CODE

MOV AX, 1027H

MOV BX, 5A27H

MOV CX, 54A5H

OR AX, BX ; AX = AX | BX

XOR AX, CX ; AX = AX ^ CX

NOT AX ; AX = ~AX

TEST CX, BX ; check flags, result not stored

AND CX, AX ; CX = CX & AX

HLT

CODE ENDS

END

**Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| Code segment executed | Registers in Hex | Registers in Bin | Output |
| MOV AX, 1027H  MOV BX, 5A27H  MOV CX, 54A5H | AX = 1027  BX = 5A27  CX = 54A5 | 0001 0000 0010 0111  0101 1010 0010 0111  0101 0100 1010 0101 |  |
| OR AX, BX | AX = 5A27  BX = 5A27  CX = 54A5 | 0101 1010 0010 0111  0101 1010 0010 0111  0101 0100 1010 0101 |  |
| XOR AX, CX | AX = E82  BX = 5A27  CX = 54A5 | 0000 1110 1000 0010  0101 1010 0010 0111  0101 0100 1010 0101 |  |
| NOT AX | AX = F17D  BX = 5A27  CX = 54A5 | 1111 0001 0111 1101  0101 1010 0010 0111  0101 0100 1010 0101 |  |
| TEST CX, BX | AX = F17D  BX = 5A27  CX = 54A5 | 1111 0001 0111 1101  0101 1010 0010 0111  0101 0100 1010 0101 |  |
| AND CX, AX | AX = F17D  BX = 5A27  CX = 5025 | 1111 0001 0111 1101  0101 1010 0010 0111  0101 0000 0010 0101 |  |

**Exercise Part 2 (a)**

**Assembly Code:**

CODE SEGMENT

ASSUME CS:CODE, DS:CODE

MOV AX, 7A24H

MOV BX, 15A3H

SUB AX, BX ; AX = 6481H after subtraction

JMP L3T2

EEE316:

DIV BX ; Divide AX = 4481H by BX = 15A3H

; Quotient AX = 0003H

; Dividend DX = 0398H

JMP Last

L3T2: MOV CX, 45B1H

AND AX, CX ; AX = 4481H after AND

TEST AX, BX

JMP EEE316

Last: HLT

CODE ENDS

END

**Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| Code segment executed | Registers in Hex | Registers in Bin | Output |
| MOV AX, 7A24H  MOV BX, 15A3H | AX = 7A24  BX = 15A3  CX = 0000  DX = 0000 | 0111 1010 0010 0100  0001 0101 1010 0011  0000 0000 0000 0000  0000 0000 0000 0000 |  |
| SUB AX, BX | AX = 6481  BX = 15A3  CX = 0000  DX = 0000 | 0110 0100 1000 0001  0001 0101 1010 0011  0000 0000 0000 0000  0000 0000 0000 0000 |  |
| JMP L3T2 | AX = 6481  BX = 15A3  CX = 0000  DX = 0000 | 0110 0100 1000 0001  0001 0101 1010 0011  0000 0000 0000 0000  0000 0000 0000 0000 |  |
| L3T2: MOV CX, 45B1H | AX = 6481  BX = 15A3  CX = 45B1  DX = 0000 | 0110 0100 1000 0001  0001 0101 1010 0011  0100 0101 1011 0001  0000 0000 0000 0000 |  |
| AND AX, CX | AX = 4481  BX = 15A3  CX = 45B1  DX = 0000 | 0100 0100 1000 0001  0001 0101 1010 0011  0100 0101 1011 0001  0000 0000 0000 0000 |  |
| TEST AX, BX | AX = 4481  BX = 15A3  CX = 45B1  DX = 0000 | 0100 0100 1000 0001  0001 0101 1010 0011  0100 0101 1011 0001  0000 0000 0000 0000 |  |
| JMP EEE316 | AX = 4481  BX = 15A3  CX = 45B1  DX = 0000 | 0100 0100 1000 0001  0001 0101 1010 0011  0100 0101 1011 0001  0000 0000 0000 0000 |  |
| EEE316:  DIV BX | AX = 0003  BX = 15A3  CX = 45B1  DX = 0398 | 0000 0000 0000 0011  0001 0101 1010 0011  0100 0101 1011 0001  0000 0011 1001 1000 |  |
| JMP Last | AX = 0003  BX = 15A3  CX = 45B1  DX = 0398 | 0000 0000 0000 0011  0001 0101 1010 0011  0100 0101 1011 0001  0000 0011 1001 1000 |  |
| Last: HLT | AX = 0003  BX = 15A3  CX = 45B1  DX = 0398 | 0000 0000 0000 0011  0001 0101 1010 0011  0100 0101 1011 0001  0000 0011 1001 1000 |  |

The last label is required because the way the program is structured, L3T2 is jumped to first, and then L3T2 directs the pointer to EEE316 which is above itself in the instruction list. After EEE316, the program counter will automatically point to the location that is below EEE316, incidentally reaching L3T2 and looping here forever. This is why the pointer is required to jump to Last where the instruction register receives halt command and stops execution.

**Exercise Part 2 (b)**

**Assembly Code:**

CODE SEGMENT

ASSUME CS:CODE, DS:CODE

MOV AX, 7A24H

MOV BX, 95A3H

ADD AX, BX

JC L3T2 ; Jump to L3T2 if CF=1

EEE316: OR AX, 23H

JNZ Last ; Jump to Last if previous operation not zero

L3T2: MOV CX, 0FC7H

SUB AX, CX

TEST AX, BX

JZ EEE316 ; Jump to EEE316 if previous operation zero

Last: HLT

CODE ENDS

END

**Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| Code segment executed | Registers in Hex | Registers in Bin | Output |
| MOV AX, 7A24H  MOV BX, 95A3H | AX = 7A24  BX = 95A3  CX = 0000 | 0111 1010 0010 0100  1001 0101 1010 0011  0000 0000 0000 0000 |  |
| ADD AX, BX | AX = 0FC7  BX = 95A3  CX = 0000 | 0000 1111 1100 0111  1001 0101 1010 0011  0000 0000 0000 0000 |  |
| JC L3T2 | AX = 0FC7  BX = 95A3  CX = 0000 | 0000 1111 1100 0111  1001 0101 1010 0011  0000 0000 0000 0000 |  |
| L3T2: MOV CX, 0FC7H | AX = 0FC7  BX = 95A3  CX = 0FC7 | 0000 1111 1100 0111  1001 0101 1010 0011  0000 1111 1100 0111 |  |
| SUB AX, CX | AX = 0000  BX = 95A3  CX = 0FC7 | 0000 0000 0000 0000  1001 0101 1010 0011  0000 1111 1100 0111 |  |
| TEST AX, BX | AX = 0000  BX = 95A3  CX = 0FC7 | 0000 0000 0000 0000  1001 0101 1010 0011  0000 1111 1100 0111 |  |
| JZ EEE316 | AX = 0000  BX = 95A3  CX = 0FC7 | 0000 0000 0000 0000  1001 0101 1010 0011  0000 1111 1100 0111 |  |
| EEE316: OR AX, 23H | AX = 0023  BX = 95A3  CX = 0FC7 | 0000 0000 0010 0011  1001 0101 1010 0011  0000 1111 1100 0111 |  |
| JNZ Last | AX = 0023  BX = 95A3  CX = 0FC7 | 0000 0000 0010 0011  1001 0101 1010 0011  0000 1111 1100 0111 |  |

**Homework 1**

Write an assembly code that will determine whether a number is greater than 5, equal to or less, and put 0, 1 or 2 for the conditions in DX

**Assembly Code:**

CODE SEGMENT

ASSUME CS:CODE, DS:CODE

MOV AX, 06H ; input number

SUB AX, 05H

JZ zero ; if result was zero

JS negative ; if result was negative

MOV DX, 00H ; otherwise

JMP last

zero: MOV DX, 01H

JMP last

negative:

MOV DX, 02H

JMP last

last: HLT

CODE ENDS

END

**Output:**

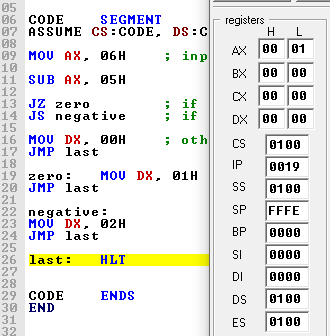
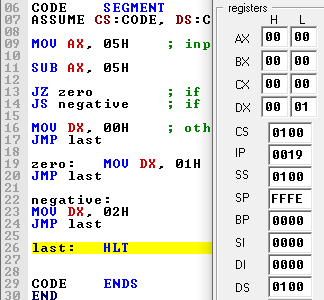
 

Fig: (a) AX = 6, DX = 0 after execution (b) AX = 5, DX = 1 after execution

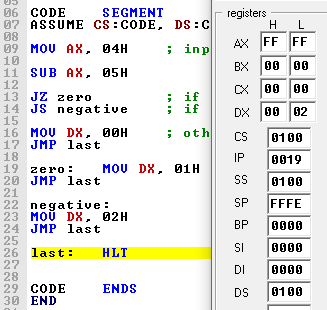


Fig: (c) AX = 4, DX = 2 after execution

**Homework 2**

Subtract 86B1H from 3F42H and store 0 in CX if overflow occurs and 1 if no overflow occurs

**Assembly Code:**

CODE SEGMENT

ASSUME CS:CODE, DS:CODE

MOV AX, 86B1H

MOV BX, 3F42H

SUB BX, AX

JO OF ; JMP if overflow

MOV CX, 01H ; otherwise

JMP last

OF: MOV CX, 0H

JMP last

last: HLT

CODE ENDS

END

**Output:**

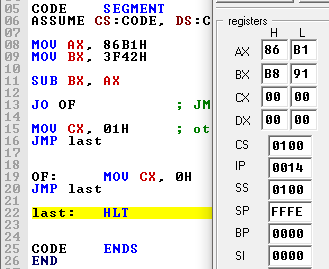
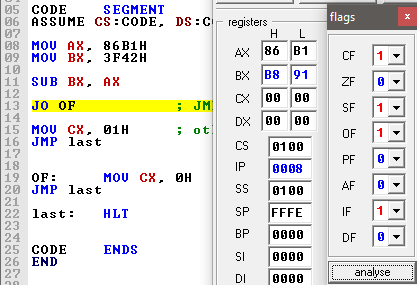


Fig: (a) Subtraction of AX from BX, overflow flag 1 (b) CX assumes value of 0

**Homework 3**

Take 2 arbitrary numbers x and y. If x>1000H perform x+y. If y<1000H perform x-y. If x>1000H and y<100H perform x = x’.

**Assembly Code:**

CODE SEGMENT

ASSUME CS:CODE, DS:CODE

MOV AX, 0250H

MOV BX, 0050H

CMP AX, 1000H ; Compares 1st val with 1000H

JS AX\_les\_1000 ; AX<1000

JZ AX\_eql\_1000 ; AX=1000

JNS AX\_grt\_1000 ; AX>1000

AX\_grt\_1000:

CMP BX, 100H ; Compares 2nd val with 100H

JS AX\_grt\_1000\_BX\_les\_100 ; AX > 1000 & BX < 100

JZ AX\_grt\_1000\_BX\_eql\_100 ; AX > 1000 & BX = 100

JNS AX\_grt\_1000\_BX\_grt\_100 ; AX > 1000 & BX > 100

AX\_les\_1000:

AX\_eql\_1000:

CMP BX, 1000H ; Compares 2nd val with 1000H

JS BX\_les\_1000 ; BX<1000

JNS last ; AX<=1000 & BX >=1000

; 1st CASE

AX\_grt\_1000\_BX\_eql\_100:

AX\_grt\_1000\_BX\_grt\_100:

ADD AX, BX

JMP last

; 2nd CASE

BX\_les\_1000:

SUB AX, BX

JMP last

; 3rd CASE

AX\_grt\_1000\_BX\_les\_100:

NOT AX

JMP last

last: HLT

CODE ENDS

END

**Output:**

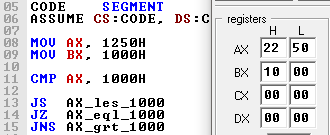


Fig: AX > 1000H, BX !<100H (1st condition applicable, x+y operation)

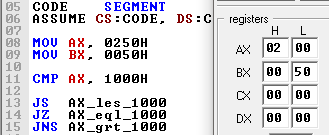


Fig: AX < 1000H, BX < 1000H (2nd Condition applicable, x-y operation)

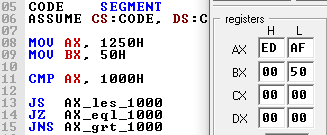


Fig: AX > 1000H, BX < 100H (3rd condition applicable, x = x’ operation)

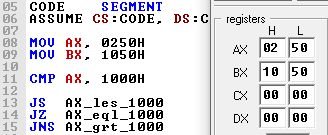


Fig: AX < 1000H, BX > 1000H (No condition applicable, no operation)

**Homework 4**

Write an assembly code that checks if a year is a leap year. Code template is shown below. If ‘YEAR’ is a leap year, put 1 in ‘LEAPYEAR’. Else put 0 in ‘LEAPYEAR’. You may observe value of LEAPYEAR by pressing the “var” button, beside the “flag” button.

**Assembly Code:**

CODE SEGMENT

ASSUME CS:CODE, DS:CODE

MOV AX, CS

MOV DS, AX

MOV AX, YEAR

MOV DX, 0

; \*\*\*\*\*\*\*\*\*\*\*\*\*SOLUTION CODE HERE\*\*\*\*\*\*\*\*\*\*\*\*\*

; DIVIDE BY 400

MOV CX, AX ; creates a copy of the main year

MOV BX, 0400D ; to divide by 400 first

DIV BX

CMP DX, 0H ; compare dividend with 0

JZ positive ; leap year if divided by 400

; DIVIDE BY 100

MOV AX, CX ; restore value of AX

MOV DX, 0 ; making sure dividend 0

MOV BX, 0100D ; to divide by 100

DIV BX

CMP DX, 0H ; compare dividend with 0

JZ negative ; not leap year if divided by 100

; DIVIDE BY 4

MOV AX, CX ; restore value of AX

MOV DX, 0 ; making sure dividend 0

MOV BX, 04D ; to divide by 4

DIV BX

CMP DX, 0H ; compare dividend with 0

JZ positive ;leap year if divided by 4

JNZ negative ; not leap year

positive:

MOV LEAPYEAR, 01H

JMP terminate

negative:

MOV LEAPYEAR, 0H

JMP terminate

terminate:

; \*\*\*\*\*\*\*\*\*\*\*\*\*\* END OF SOLUTION \*\*\*\*\*\*\*\*\*\*\*\*\*

HLT

YEAR DW 2021D ; DW = Data Word (16 bits)

LEAPYEAR DB ? ; DB = Data Byte (8 bits)

CODE ENDS

END

**Output:**

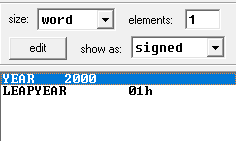
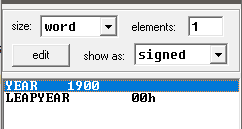


Fig: (a) 1900 not leap year (b) 2000 leap year

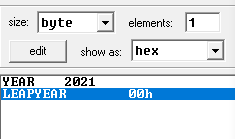
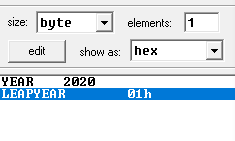


Fig: (c) 2020 leap year (d) 2021 not leap year