Name:

Dawood Sarfraz

Roll no:

P20-0153

Batch:

FALL-2020

Session:

BSCS-3B

Lab:

#07

Report:

#07

Submitted to:

Dr. Usman Abbasi

FAST NUCES Peshawer Campus

Part 1:

Bitwise Logical operations

The four Basic operations:

* AND
* OR
* XOR
* NOT

AND (and):

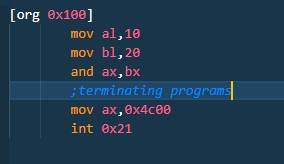
In and when our both bits are 1 our result will be 1

And taken two operands Syntax:

and operand1,operand2

our result will store in operand2

e.g :



Output:

When we use and operations

10 = 00001010

20 = 00010100

Result in ax register

ax = 00000000

After and operations

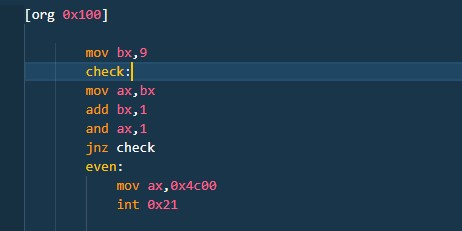


2nd Examples

Here we check the number is even or odd A simple is that if least significant bit is 1 the.

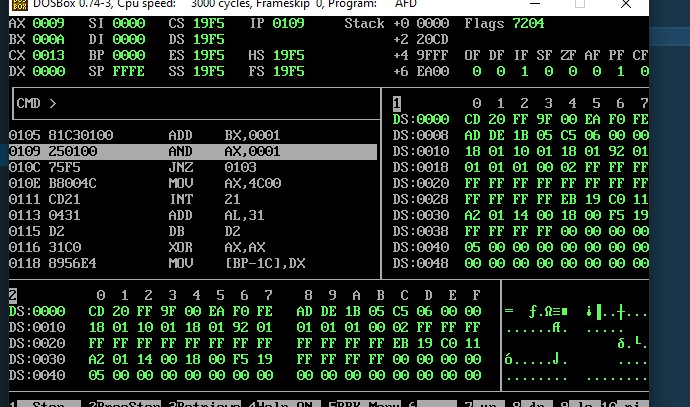
If this is 1, the number is odd, else the number is even.

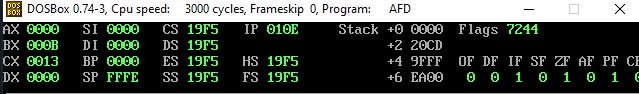
Code



Output:

Here our value is 9 which is odd we first check it is odd again move add it 1 and which will become 10 that is even now our and check then gave output it.



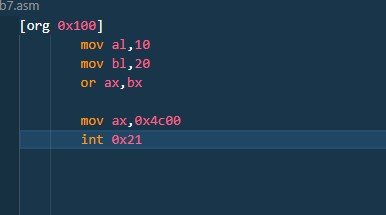


Here when we and ax, 1 the result we will get is 0000 so our ZF(flag) will set .

2: OR operations

The bitwise OR operator returns 1, if the matching bits from either or both operands are one. It returns 0, if both the bits are zero.

Code:

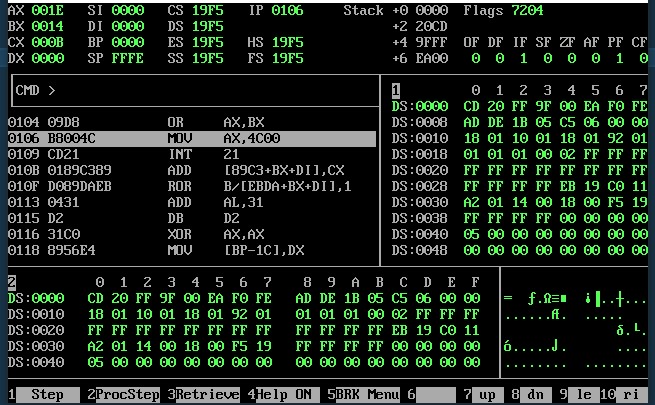


al=00001010

bl=00010100

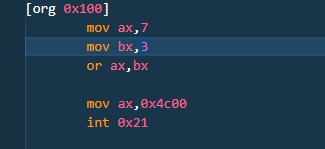
Result:

ax=00011110



Code 2:

In this when we mov ax, 7 after or with bx again with 7….why!



Output:

7 = 0111

3 = 0011

Result

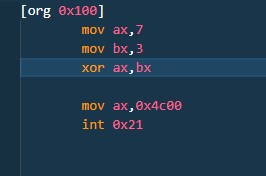
Or=0111



3: XOR

The XOR instruction implements the bitwise XOR operation. The XOR operation sets the resultant bit to 1, if and only if the bits from the operands are different. If the bits from the operands are same, the resultant bit is cleared to 0.

Code:



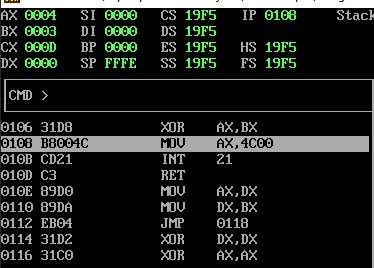
7 = 0111

3 = 0011

Result

xor = 0100

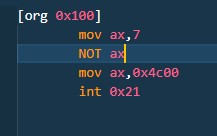
Output:



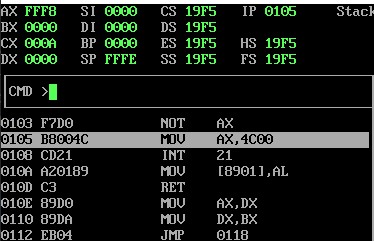
NOT

NOT operation reverses the bits in an operand. The operand could be either in a register or in the memory.

Code:

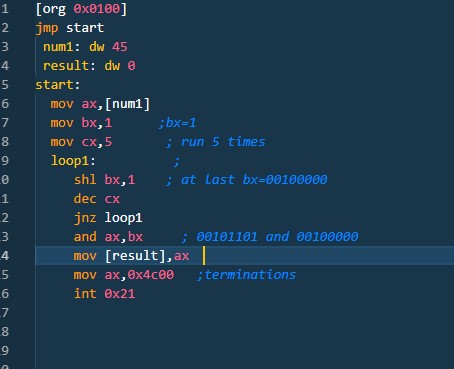


Output:



**Bit Manipulation**

1. **- Find ith Bit**



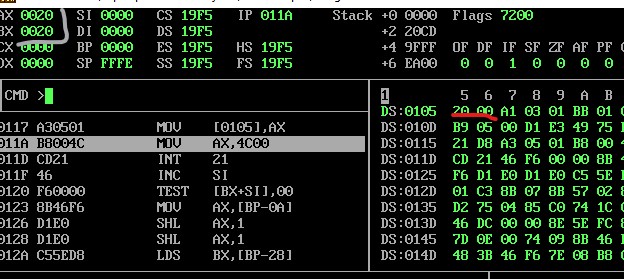
45 = 00101101

After making bx register to 32 we apply and operations on it

32 = 00100000

Result :

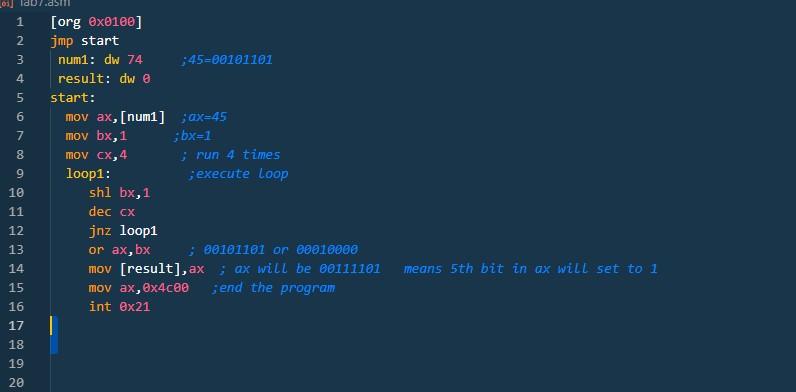
OutPut:



1. – Set the ith Bit:

Code:

74d = 01001010

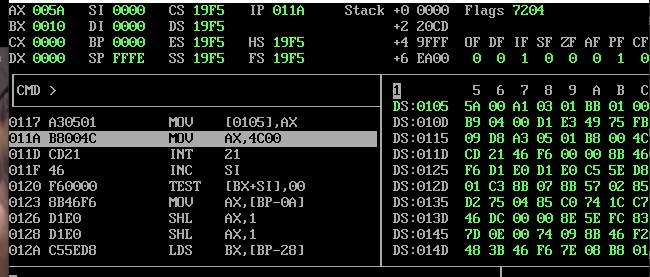


16d = 00010000

After or we will get some of this result

90d=01011010

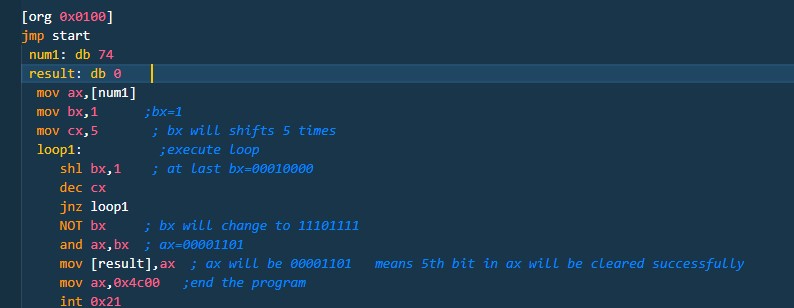
Output:



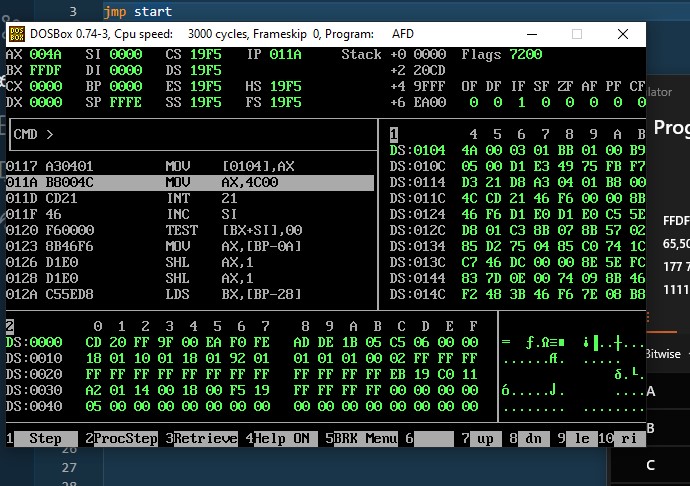
1. – Clear the ith Bit:

Code:

OutPut:



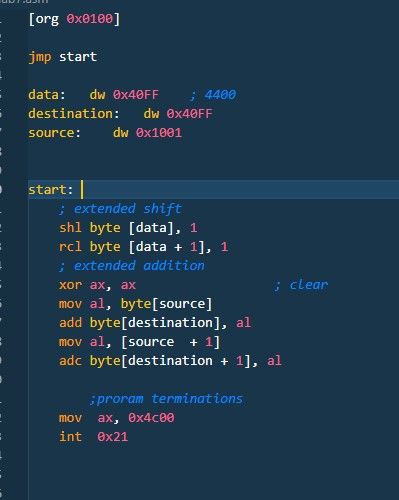
Extended Registers:



**Addition:**

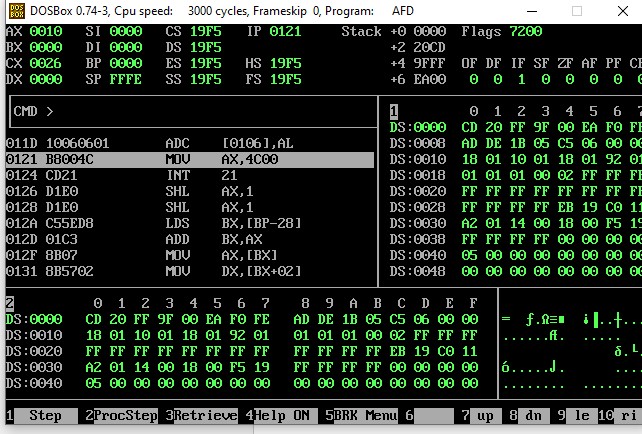
we adds lower bits of destination and source and then we add higher bits of destination and source and if carry generated that will be added in it we have taken destination and source as word but we take the higher part of source register and add into the higher part of destination register and carry also added during in it by keyword adc(add with carry). Similarly we take the lower part of source register and add into the lower part of destination register.

code:



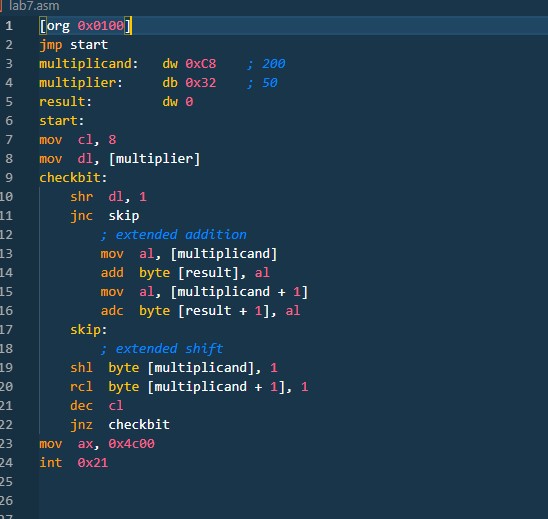
Output:

Multiplications:

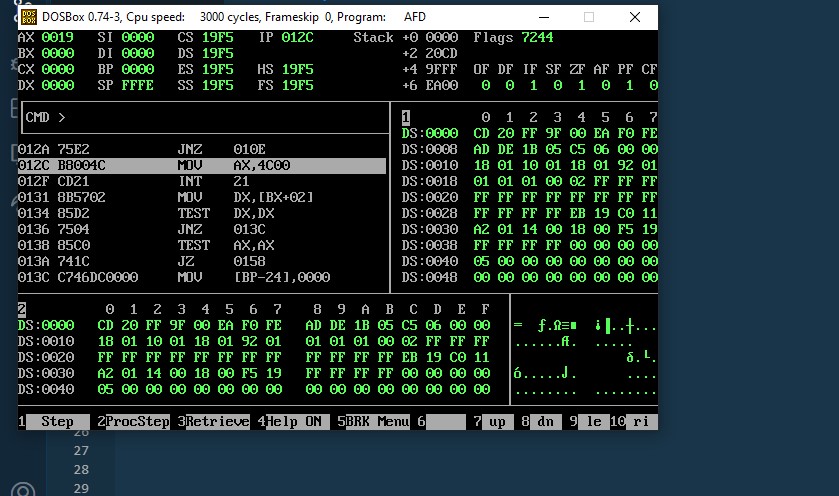


we adds lower bits of destination and source and then we add higher bits of destination and source and if carry generated that will be added in it we have taken destination and source as word but we take the higher part of source register and add into the higher part of destination register and carry also added during in it by keyword adc. Similarly we take the lower part of source register and add into the lower part of destination register.

Code:



Output:



**COMPLETE**