Computer Science Simplified

CSSimplified.com

An Assembly program in which a procedure converts Hexadecimal value to print its Decimal form on Screen

By Gangadhar Kopella | March 5, 2014 0 Comment

Now we will write another Assembly program in which a procedure converts Hexadecimal value to print its Decimal form on Screen

Let's identify variables needed for this program.
First variables will be the one which will hold the value present in the variable NUM converts Hexadecimal value to print its Decimal form on Console (Screen) and Other variable RES will be holding the Resultant Decimal equivalent printable form to be printed for the User on Screen, So in all Two variables.
The identified variables are **NUM** and **RES**.

First Line – DATA SEGMENT

DATA SEGMENT is the starting point of the Data Segment in a Program and DATA is the name given to this segment and SEGMENT is the keyword for defining Segments, Where we can declare our variables.

Next Line – NUM DW 1234H
RES DB 10 DUP ('$')

NUM DW 1234H We are initializing NUM to 1234H (H stands for **Hexadecimal value**). The number which we are using is of 4 digits and DB is capable of holding only 2 digits, Hence we use DW as it can hold 4 digits.

RES DB 10 DUP ('$') this line is a declaration of Array initialized with '$' which works as New Line Character. $ is used as (\n) NULL character in C program. (A Number Character is of a BYTE size Hence we have to used only DB Define Byte) as we don't know the lenght of the digits in the Resultant Decimal equivalent printable form, Therefore we take it approx size ten. Here 10 DUP ('$') stands for N i.e. Size of Array or Array Size. DUP stands for Duplicate i.e. it will duplicate the value in All the Array with the value present in Bracket (i.e. $).

Next Line – DATA ENDS

DATA ENDS is the End point of the Data Segment in a Program. We can write just ENDS But to differentiate the end of which segment it is of which we have to write the same name given to the Data Segment.

Now, Selection of data type is **DW** data type As the number which we are converting in the Register can be Max of size 2 bytes so DW is selected.

Source code

```
DATA SEGMENT
    NUM DW 1234H
    RES  DB 10 DUP ('$')
DATA ENDS
```

In Assembly programming, the variable are all defined by bytes only.

DB – Define Byte (Size – 1 Byte)

DW – Define Word (Size – 2 Byte)

DD – Define Double word (Size - 4 Bytes)

DQ – Define Quad word (Size – 8 Bytes)

DT – Define Ten Bytes (Size – 10 Bytes)

NUMBER SYSTEM in Assembly Programming is Decimal, Octal, Hexadecimal, Binary.

In the Program, We are entering the values for the variables and Do arithmetical Operations like Addition, Subtraction, Multiplication and Division So the Computer should understand which kind of Number is entered. Hence there is a different letters for different Number Systems. **O** or **o** stands for **Octal, H** or **h** stands for **Hexadecimal, B** or **b** stands for **Binary, D or d** stands for **Decimal. By** default type of numbering system is Decimal. If you do not specify any letter then the number is understood to be Decimal (By default).

Source code

```
DATA SEGMENT
    NUM DW 1234H
    RES  DB 10 DUP ('$')
DATA ENDS
CODE SEGMENT
       ASSUME DS:DATA,CS:CODE
START:
```

```
    MOV AX,DATA
    MOV DS,AX

    MOV AX,NUM

    LEA SI,RES
    CALL HEX2DEC

    LEA DX,RES
    MOV AH,9
    INT 21H

    MOV AH,4CH
    INT 21H
CODE ENDS
HEX2DEC PROC NEAR
    MOV CX,0
    MOV BX,10

LOOP1: MOV DX,0
       DIV BX
       ADD DL,30H
       PUSH DX
       INC CX
       CMP AX,9
       JG LOOP1
       ADD AL,30H
       MOV [SI],AL

LOOP2: POP AX
       INC SI
       MOV [SI],AL
       LOOP LOOP2
       RET
HEX2DEC ENDP

END START
```

Explanation :

In this Assembly Language Programming, A single program is divided into four Segments which are 1. Data Segment, 2. Code Segment, 3. Stack Segment, and 4. Extra Segment. Now, from these one is compulsory i.e. Code Segment if at all you don't need variable(s) for your program.if you need variable(s) for your program you will need two Segments i.e. Code Segment and Data Segment.

Next Line –CODE SEGMENT

CODE SEGMENT is the starting point of the Code Segment in a Program and CODE is the name given to this segment and SEGMENT is the keyword for defining Segments, Where we can write the coding of the program.

Next Line – ASSUME DS:DATA CS:CODE

In this Assembly Language Programming, their are Different Registers present for Different Purpose So we have to assume DATA is the name given to Data Segment register and CODE is the name given to Code Segment register (SS,ES are used in the same way as CS,DS)

Next Line – START:

START is the label used to show the starting point of the code which is written in the Code Segment. : is used to define a label as in C programming.

Next Line – MOV AX,DATA
MOV DS,AX

After Assuming DATA and CODE Segment, Still it is compulsory to initialize Data Segment to DS register. MOV is a keyword to move the second element into the first element. But we cannot move DATA Directly to DS due to MOV commands restriction, Hence we move DATA to AX and then from AX to DS. AX is the first and most important

register in the ALU unit. This part is also called INITIALIZATION OF DATA SEGMENT and It is important so that the Data elements or variables in the DATA Segment are made accessable. Other Segments are not needed to be initialized, Only assuming is enhalf.

Next Line – MOV AX,NUM

The above line code is used to Move NUM variable value to AX Register.

Next Line – LEA SI,RES
CALL HEX2DEC

The above Two line code is used to initialize RES to SI register and Call Procedure HEX2DEC

LEA SI,RES is used to Load Effective Address of RES variable to SI Register.

CALL HEX2DEC is used to Call a Procedure named HEX2DEC

Next Line – LEA DX,RES
MOV AH,9
INT 21H

The above three line code is used to print String or Message present in the character Array till $ symbol which tells the compiler to stop. As we have initialized all the values in an Array to $ you will think what will be printed. The procedure is going to change the Array to its Resultant Decimal equivalent printable form i.e. ASCII form of a digit number.

Now, lets understand line by line

LEA DX,RES in this LEA stands for LOAD EFFECTIVE ADDRESS and it loads the effective address of second element into the first element. This same code can be interchangably written as ***MOV DX, OFFSET RES*** where OFFSET means effective address and MOV means move second element into the first element.

MOV AH,9
INT 21H

The above two line code is used to PRINT the String or Message of the address present in DX register.

Standard Input and Standard Output related Interupts are found in INT 21H which is also called as DOS interrupt. It works with the value of AH register, If the Value is 9 or 9h, That means PRINT the String or Message of the address present in DX register.

Next Line – EXIT: MOV AH,4CH
INT 21H

The above two line code is used to exit to dos or exit to operating system. Standard Input and Standard Output related Interupts are found in INT 21H which is also called as DOS interrupt. It works with the value of AH register, If

the Value is 4ch, That means Return to Operating System or DOS which is the End of the program.

Next Line – CODE ENDS

CODE ENDS is the End point of the Code Segment in a Program. We can write just ENDS But to differentiate the end of which segment it is of which we have to write the same name given to the Code Segment.

PROCEDURE Code starts here:

Procedure is a part of code that can be called from a program in order to perform specific task.

Next Line – HEX2DEC PROC NEAR

This line of code is used to start a procedure code and we can make out the procedure by the keyword PROC which tells us the procedure is started. In assembly language we have two types of Procedures one is NEAR and other is FAR. NEAR is used to call the Procedure within the program whereas FAR is used to call the procedure outside the program. HEX2DEC is only the Name given to the Procedure Code.

Next Line – MOV CX,0
MOV BX,10

MOV CX,0 is used to move or assign value 0 (decimal value) to CX. The program which we are wishing to write is to covert HexaDecimal value to Decimal value, In which we will divide the number till the Quotient is going to be Zero. CX register (CX is also Called COUNTER). CX register will count the number digit generated by dividing the Hexadecimal number by Base value of Decimal i.e.Ten. ***MOV BX,10*** in this Base value 10 is moved to BX register, So that it is used to divide hexa number by 10.

Next Line – LOOP1: MOV DX,0

LOOP1: is a LABEL and all the words ending in colon (:) are Labels. MOV DX,0 is used to clear the unwanted value (garbage value) in DX register is removed by assigning ZERO to it. *First Loop starts here.*

Next Line – DIV BX
ADD DL,30H

DIV instruction only works with REG or MEMORY hence we cannot use DIV 10 where 10 is immediate, So we have to move 10 to BX register (we can take any register) this we have already done above and Then ***DIV BX*** Now DIV BX will Divide AX register with 10 which is passed to BX register and Result of division is present in AX register contains Quotientand DX register contains Remainder. Here we will not touch Quotient AX as it will be used for furture Division, But DX Remainder will be Decimal Digit and will always be less than Ten so the value will be in DL register only and to make it printable on Console (Screen) we have to add 30H So that it will become a ASCII character and will be saved in Charater Array and will be printed as String later So ***ADD DL,30H***.

Next Line – PUSH DX
INC CX

PUSH is a stack function. Stack is an area of memory for keeping temporary data. PUSH and POP are two stack operations which stores or gets 16 bits of data. ***PUSH DX*** stores 16 bit data inside DX register into Stack Area. INC is a instruction for Increment the present in Register or Memory. ***INC CX*** will increment the value present in CX register by One. Here we are using CX register as a counter and counting the numbers of digits in their ASCII form which are pushed into Stack. So that the same count will help to POP the values out of Stack.

Next Line – MOV CX,10

MOV CX,10 is used to move or assign value 10 (decimal value) to CX. The program which we are wishing to write is to input ten characters from console which will be entered by the user, Hence to do so we need a loop construct. In assembly programming language we have a LOOP instruction. This works with two other helpers which are Label and Counter. The Loop start with LABEL and ends with LOOP instruction with the same LABEL name with it. the execution of the Loop depends on the value in CX register (CX is also Called COUNTER).

Next Line – CMP AX,9
JG LOOP1

CMP AX,9 is used to compare AX register with 9 and jump if AX is greater to the respective LABEL LOOP1. The result of Comparision is not stored anywhere, but flags are set according to result. is Short Jump if first operand is Greater then second operand (as set by CMP instruction). Signed. SECOND is the label where the compiler will JUMP. *First Loop ends here. Note :- this loop is without LOOP keyword and depends upon the number to be converted.*

Next Line – ADD AL,30H
MOV [SI],AL

ADD AL,30H The Last Remainder will be Decimal Digit in AX register only as the number cannot be divided future and will always be less than Ten so the value will be in AL register only and to make it printable on Console (Screen) we have to add 30H So that it will become a ASCII character and will be saved in Charater Array and will be printed as String later. ***MOV [SI],AL*** Saving the Characters in Character Array (i.e. String) is done by Moving AL register to Address of SI register which is represented in Square Brackets i.e. [SI]. SI is assigned with the Character Array i.e. RES.

Next Line – LOOP2: POP AX
INC SI

LOOP2: is a LABEL and all the words ending in colon (:) are Labels. POP is a stack function. Stack is an area of memory for keeping temporary data. PUSH and POP are two stack operations which stores or gets 16 bits of data. ***POP AX*** *get*s 16 bit data to AX register from Top of Stack. ***INC CX*** will increment the value present in CX register by One. Here we are using CX register as a counter and counting the numbers of digits in their ASCII form which are pushed into Stack. So that the same count will help to POP the values out of Stack and save it in AX register. *Second Loop starts here.*

Next Line – MOV [SI],AL

The values out of Stack saved in AX register saved in string in this Loop. ***MOV [SI],AL*** Saving the Characters in Character Array (i.e. String) is done by Moving AL register to Address of SI register which is represented in Square Brackets i.e. [SI]. SI is assigned with the Character Array i.e. RES.

Next Line – LOOP LOOP2

This end of loop. In assembly programming language we have a LOOP instruction. This works with two other helpers which are Label and Counter. The Loop start with LABEL and ends with LOOP instruction with the same LABEL name with it. the execution of the Loop depends on the value in CX register (CX is also Called COUNTER).

Next Line – RET

RET is a return instruction. This instruction is used only if the control is been passed to the code outside Main like to Procedure. this return the control to the place where the Procudure was called.

Next Line – HEX2DEC ENDP

HEX2DEC ENDP is the End point of the **Procedure** in a Program.

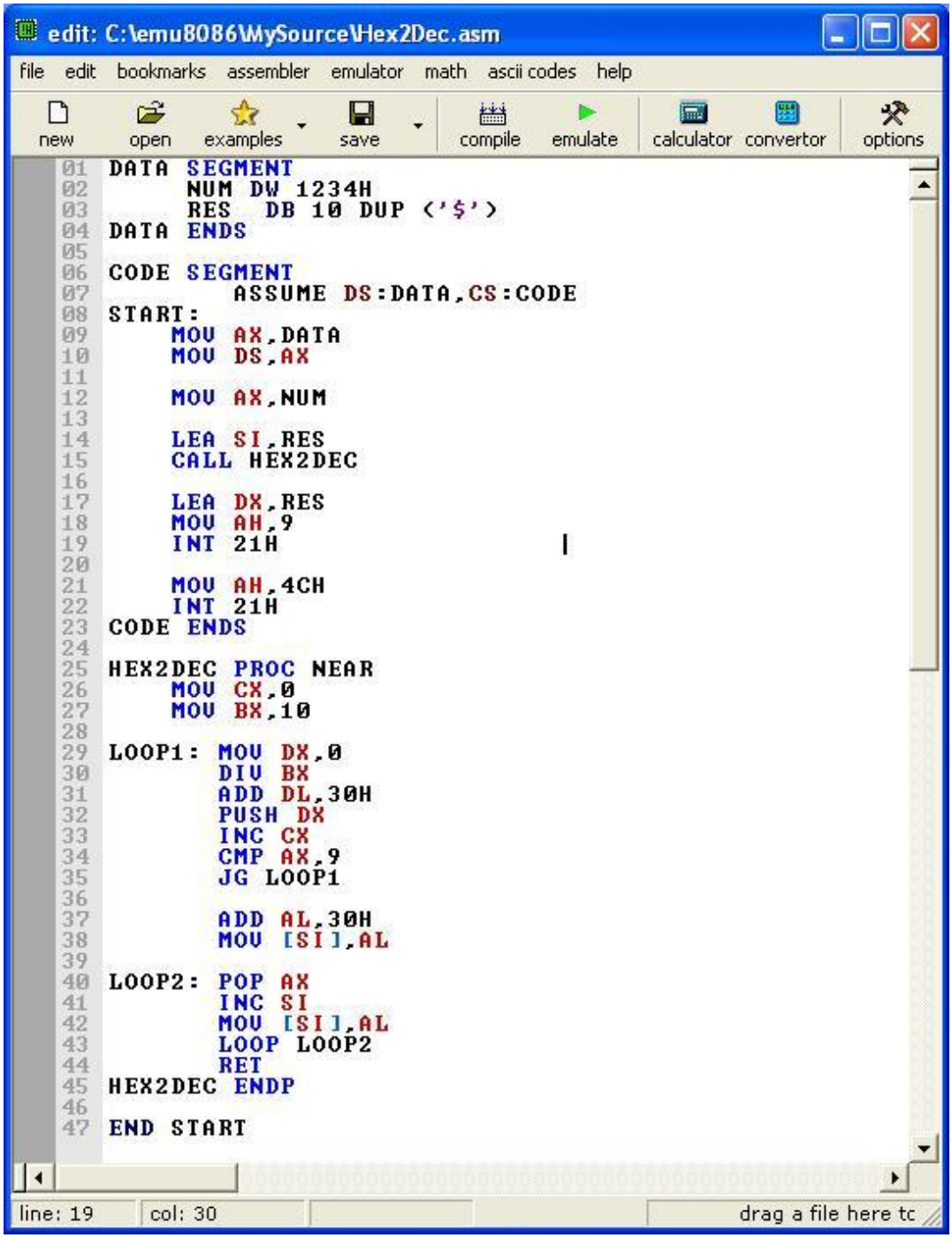
This line of code is used to end the procedure code and we can make out the procedure by the keyword ENDP which tells us the procedure is ended. In assembly language we have two types of Procedures one is NEAR and other is FAR. NEAR is used to call the Procedure within the program whereas FAR is used to call the procedure outside the program. HEX2DEC is only the Name given to the Procedure Code.

Last Line – END START

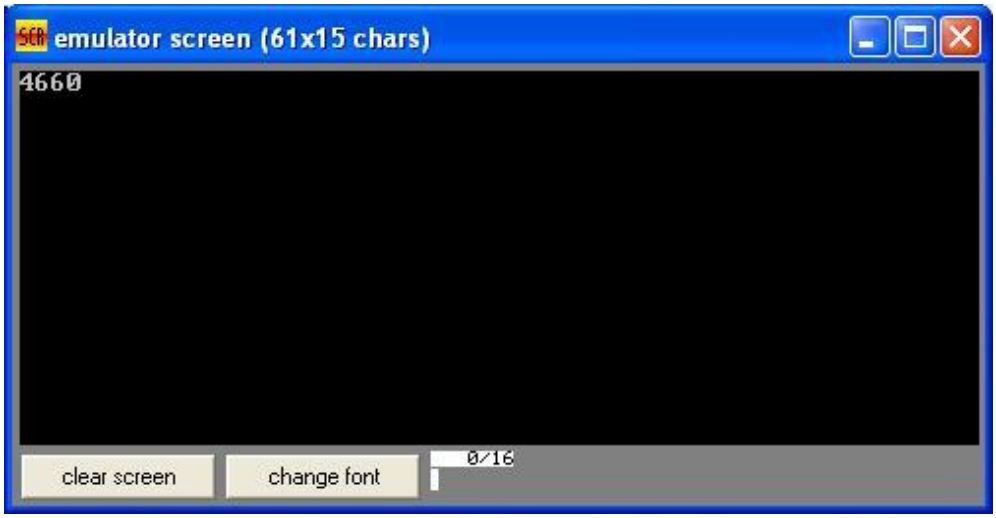
END START is the end of the label used to show the ending point of the code which is written in the Code Segment.

Note :- In this Assembly Language Programming, We have Com format and EXE format. We are Learning in EXE format only which simple then COM format to understand and Write. We can write the program in lower or upper case, But i prepare Upper Case.

Screen Shots :-



Output After Execution :-



Note :- To see the variable and its value you have to click **vars** button in the emulator.

Category: Assembly Language Programs Computer Organisation and Assembly Language Programming Tags: Assembly, convert, decimal, form, Hexadecimal, onscreen, print, procedure, program, value