Ayesha Mehmood

ID # 5832

Degree # BSCS

Semester # 8th

Assignment # 3

Assembly Language

Assembly language

11) Create a definition for a doubleword that stored in memory in little endian formate.

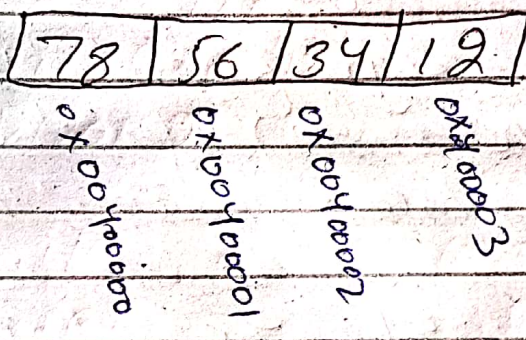
Ans):- Little Endian:-

The least significant byte (the "little end") of the data is placed at the byte with the lowest address. The rest of the data is placed in order in the next bytes in memory.

* In these definition, the data, a 32-bits patterns, in regarded as a 32-bit unsigned integer. The "most significant" byte is the one for the largest power of two: $2^{31}, \ldots, 2^{24}$. The "least significant" byte is the one for the smallest power of two: $2^{7}, \ldots, 2^{0}$.

* For example. Say that the 32-bit pattern 0x12345678 is stored at address 0x00400000.

Little Endian



12) Show the order of individual bytes in memory (lowest to highest) for the following doubleword.

Variable: val1 DWORD
87654321h

True or false: The following is a valid data definition statement:

VAR1 BYTE 0Ah, 255

Write an assembly program to ~~calculate~~ compute the following expression:

AL = BL + vall

vall is an 8-bit memory variable.

16) Declare unsigned 16-bit integer variable named wArray the uses three initializers.

Ans) wArray WORD 10, 20, 30

17) Declare a string variable containing the name of your favorite color. Initialize it as a null-terminated String.

Ans) myColor BYTE "blue", 0.

(b) Create an unitialized data declaration for 8 bits, 16-bit, 32 bits unsigned and signed integers.

Ans)
~~Var1 SWORD~~
Var2 BYTE
Var1 SWORD
SDWORD

19) Write a statement that causes the assembler to calculate the number of bytes in the following array, and assign the value to a symbolic constant named ArraySize:
myArray WORD 20 DUP(?)

Ans) ArraySize = ($ - myArray) / TYPE DWORD

Q23) Differentiate b/w equal-sign directive and EQU directive.

Equal-Sign Directive:-

- The equal-sign directive associates a symbol name with an integer expression. This syntax is

name = Expression

- Expression is a 32-bit integer
- may be redefined
- Name is called a symbolic constant.

EQU-directive:

- Define a symbol as either an integer or text expression
- Cannot be redefined.

Q28) Name the four basic parts of an assembly language instruction.

Ans) [label:] mnemonic [operands] [; comment].

Q27) How are data labels and code labels different?

Ans) Data labels exists in the data segment as variable offsets. Code labels are in the code segment, and are offsets for transfer of control instructions.

29) Show an example of a block comment.

COMMENT!
First line comment
second line comment

30) Why is it not a good idea to use numeric addresses when writing instructions that access variables?

Ans) You do not use numeric addresses (offsets) for variables because the addresses would change if new variables were inserted before the existing ones.

1) Using the value of -35 write it as an integer literal in decimal, hexadecimal, octal, and binary formats that are consistent with MASM syntax.

Ans) -35d, 0DDh, 3350, 11011101b

2) Write the real number -6.2×10^{-4} as a real number literal using MASM syntax.

Ans) -62E +04

5) which statement halts the assembly language program?

Ans) The exist statement (indirectly) call a predefined MS-window function that halts the program. The SNDP direct marks the end of the main procedure. The End main direct mark the last line of the ~~program~~ Program to be assembled. DATA type essential characteristics size in bit 8, 16, 48, 64 and 80.

7) What is a calling convention and how is in used in assembly language declaration.

Ans) A calling convention determines how parameters are passed to subordinates and how the stack is restored after the subroutine call.

(8) Why might use a Symbolic constant rather than an integer literal in your code?

Ans) An integer literal, such as 48, has no direct meaning to someone reading the program sources code. instead, a symbolic constant such a STUDENT-COUNT. Can be assignment an integer value and is self-documenting.

25) How is a source file different from a listing file?

Ans) A source file is given as input to a assembler. A listing file has additing text that will not assemble. it is a file that is created by the assembler and it is optionaly generated.

8) What type of files are produced by a assembler and linker?

Ans) The main output produced by assembler on input assembly language source file is the translation of the file into an object file in (ELF). ELF file produced by the assembler are relocatable file that hold code and/or data. They are input files for the linker.

9) which operation systems component reads and executes programs?

Ans) A monolithic kernel runs all the operating systems instruction in the same address space for speed. A microlessness run most process in user space

for modularity. This central component of a compiler system is responsible for running ; or executing program.

OR

The loader.

2- Create a single integer expression that uses all the operators. ~~from~~ Calculate the value of the expression.

$$(5+1)(-2+3)*2 \bmod 5 = 2$$

~~Q~~ What type of arguments must be passed to the ExitProcess procedure?

Ans)

- model small
- 386
- Stack 100h

- data
  Sunday = 0

Monday = 1
Tuesday = 2
wednesday = 3
Thursday = 4
Friday = 5
Saturday = 6

Days = DB Sunday, Monday,
Tuesday, wednesday,
Thursday, Friday, Saturday

- code
main:

```
mov ax, @data
mov ds, ax
;just for test; print the first
                value

mov ah, 02h
mov dl, days
add dl, 30h
int 21h

mov ah, 4Ch
int 21h

end main
```

6) What type of argument must be passed to ExitProcess procedure?

Ans) An integer, preferably 0.

20) Show how to calculate the number of elements in the following array, and assign the value to a symbolic constant named ArraySize:

```
myArray DWORD 30 DUP(?)

ArraySize = ($ - myArray)/4

OR ArraySize = ($ - myArray)
     TYPE DWORD.
```

14) Declare an array of 120 uninitalized unsigned doubleword values.

```
myArray DWORD 120 DUP(?)
```

15) Declare an array of byte and initilize it to the 5 letters of the alphabet.

myArray BYTE 'A', 'B', 'C', 'D', 'E'.

13) Find out if you can declare a variable of type DWORD and assign is a negative value.

val1 DWORD 12345678h; unsigned
val3 DWORD 20DUP(?); unsigned

4) Discuss the following MASM directives:

Include .386 .MODEL

.STACK PROTO .DATA

.CODE PROC ENDP END

```
.386
.model flat, stdcall. stack
4096 exit process PROTO,
dwExitCode : DWORD
```

The .384 directive identifies it as 32-bit program. Line 2 uses the flat memory model and windows requires the stdcall convention to be used. Line 3 sets aside 4096 bytes of storage, and line 4 declares a prototypes for the Exit process function. This prototype has a PROTO keyword, a ~~Comma~~ comma, and a list of input Parameters.

MODEL:

This tells the assembler which memory model to use. In 32-bit Programs, we use the flat memory model, which is associated with the processors Protected mode.

STACK

The Stack directive tells how many bytes of memory reserve for the runtime stack. 4096 happens to correspond to the size of a memory page in the processor's system for managing memory.

CODE

It's the boginig of the code area of the program (meaning what's afterwards is usually the main procedure).

DATA

The DATA directive creates a near data segment.

This DATA segment contain the frequency used data for your Program.

- Data segment can occupy
  - up to 64K in MS-Dos
  - or up to 512 megabyte under flat model in window NT.

PROTO

The PROTO directive by using the invoke directive.

Syntax:

Label proto [[distance]] [language-type]

[[parameter]] : tag---]]

Parameter:

distance (32-bit MASM only) (optional) used in 16-bit memory model to override the default and indicate NEAR or FAR calls.

Remarks:

See PROC

END

Directive for END of the command that is END of the. Here as you

using main you have to end it with.

- Simply know that the Assembler need END directives to end the file END can be written without main just end the file with END.

Now ENDP denotes END of PROCEDURE Like procedure (or "main") so before ending the file, End the "main" procedure. You have to end the procedure.

PROC

Mark start and end of a procedure block called label.

The statement in the block can be called with the call instruction or INVOKE directives.

Syntax:

label PROC [distance] [language-type]
[[PUBLIC | PRIVATE | EXPORT]].
[[< proto guearg >]]

* [[uses reglist,] [parameter [:tag]]...]
* [[FRAME [: ehandler-address]]]

Statement
Label end.

ENDP
Marks the end of Procedure name previously begun with PROC.

Syntax:
name ENDP.

22) write a program that defines a symbolic names for several string literals (character b/w quotes). Use each symbolic name in variable definition.

```
.386
.model flat. stdcall
.stack 4096
Exit ~~code~~ process PROTO,
dw ExitCode : DWORD
Str 1 EQU <" Anam", 0>
Str 2 EQU <" Zainab", 0>
Str 3 EQU <" Ifra", 0>
.data
```

```
First   BYTE  Str1
Second  BYTE  Str2
third   BYTE  Str3

.code
main PROC

INVOKE  ExitProcess, 0
main    ENDP
END     main.
```

26) write a Program that containg two Instruction: (1) add the number 5 to the EAX register, and (2) add 5 to the EDX register, Generate a listing file and examine the machine code generated by the assembler, what differences, if any, did you find b/w two instructions?

Ans) (1) . add the number 5 to the EAX register.

. add 8 to the EDX register.

* Submit the following:

- Lastname1.asm

- whatever the name. 1st
- answer. Pdf

2) EAX = -val2 + 7 - val3 + val1
Assume that val1, val2 and val3 are 16-bit number integer variables.

* Submit the following.

- Lastname2.asm.

32) write a program that calculates the following expression, using register;

A = (A + B) - (C + D)

Assign integer values to the EAX EBX, BCX, and EDX register.

```
.386
.model flat, stdcall
.stack, 4096
ExitProcess PROTO, dwExitCode: DWORD

.code
main PROC

movea x, 3h
```

```
mov ebx, 8h
mov ecx, 1h
mov edx, 8h

add eax, ebx
add ecx, edx
sub eax, ecx

INVOKE ExitProcess, 0
main ENDP
END main
```

31) Find, out by trial and error, if a program can have multiple code and data segments.

```
mov ebax, 8h
mov ecx, 1h
mov edx, 8h

add eax, ebx

add ecx, edx

sub eax, ecx
```