Abel Shefnie

CSC 337 Assembly Language

10/3/2020

Assignment 2

Question 1. Describe the relation between a basic assembly code and memory segmentation.

Ans: Basic assembly code can be divided into three sections:-

- The data section is used for declaring initialized data or constant,
- The bss section is used for declaring variables.
- The text section is used for keeping the actual code.

The memory segmentation divides the system memory into groups of independent segment referenced by pointer located in the segment registers. Each segment stores different type of data values like instruction code, data elements, and program stack. The memory in assembly code is categorized into three segments.

- Code segment- it is the segment of memory which stores the instruction code. It is represented by .text section.
- Data segment- it stores the data element of the code. It is represented by .data section and the .bss. The data section is used to declare the memory region, where data elements are stored for the program.
- Stack- this segment contains data values passed to functions and procedures within the program.

Question 2. If we compile the following piece of C++ code, explain how the registers will act?

```
String str = ”Welcome”

Std::cout<< str;
```

Ans: The std register will request to the complier that the specified variable string str is to be stored in a register of the processor instead of memory as a way to gain speed.

Question 3. When an instruction requires two operand what do the first operand and the second operand represents? What does the following addressing mode represents?

Ans: If the instruction requires two operand, then the first operand (left hand) represents the source operand the second operand (right hand) represents the destination operand.

Question 4. What is the syntax of the MOV instruction? What are the five forms a MOV instruction may have?

Ans: MOV instruction is used for moving data from one storage space to another, it takes two operands. The syntax of the MOV instruction are: - MOV destination, source.

The MOV instruction have five forms:-

1. MOV register, register

2. MOV register, immediate
3. MOV memory, immediate
4. MOV register, memory
5. MOV memory, register

Question 5. Find the name of the variable and variable-length from the following code

https://rextester.com/l/nasm_online_compiler

The variable name is "hello" and the variable length is "helloLen".

Question 6. Why the two codes differ from each other

code 1

code2

The first code output is "Hello world". It has "**helloLen: equ $-hello**", which implies the Length of the 'Hello world!' string. It also has "mov edx,helloLen", which is a constant that's been used to get its actual value, and it will be "Hello world".

The second code output is "Hell". It has "helloLen: equ $-hello", which implies the Length of the 'Hello world!' string. It also has "mov edx,4", which is being used to get the first 4 strings from the hello variable, and it will be "Hell".

Question 7. What are the change you have to make in the code Code_Multiple-Initialization to get the following output. (Make changes in code or re-write. Then press the button RUN(F8) . After running save it and name it. You will be given a link, which you will copy and paste it in your assignment.)

```
 # ^ ## ^ ## ^ ## ^ #
```

Ans: https://rextester.com/JVT78413

Question 8. Describe each line of the following code https://rextester.com/l/nasm_online_compiler.

Ans:

Line 3 – this is the syntax to declare data section which are used for declaring initialized data or constants.
Line 4 – 'hello' is a string of bytes which contains 'Hello world!'
Line 5 – determines the length of the string by subtraction the location of the start of the string from the location after the string
Line 7 – is for declaring text section which is used to keep the actual code
Line 8 – the global keyword is used to make an identifier accessible to the linker
Line 10 – defines the entry point of the program
Line 11 – this line is to let the system know, to show the output
Line 12 – register ebx calls the function to print output
Line 13 – ecx holds the string pointer
Line 14 – edx holds the string length
Line 16 – calls the kernel to show output
Line 18 – the system calls for exit
Line 19 – this line exits with return of 0, which means the program run successfully

Line 20 – this line is the interrupt with the system call