CSUS
SCHOOL OF ENGINEERING AND COMPUTER SCIENCE
Department of Computer Science
CSC 35
Spring 2021
Dr. Ghansah

Lab #6: Direct Addressing, Input/Output, Intro to PROCEDURES

Purpose: The main purpose of this lab assignment is to demonstrate understanding of how to use procedures (ie. Subroutines, subprograms, functions, methods) in x86 Assembly. It requires modification of the previous lab.

Introduction: In x86 architecture, a *procedure* is used to define and execute a block of instructions. A procedure is used in situations where the block of instructions will be used multiple times. In that case the procedure is *defined* once and subsequently *called* whenever it is needed. If the caller of the procedure (eg. Main) wants to send *parameters* to the called procedure, those parameters are passed before the procedure is called. These parameters are *input parameters*. Additionally, if the caller of the procedure needs values from the procedure, those values are returned via *output parameters*. In Intel x86 (and most Architectures), parameters can be passed through registers or through the stack. This assignment focuses on defining and using procedures as well as parameter passing *via registers*.

Defining a Procedure: The syntax for declaring procedure in Intel architecture is as follows (words in bold are keywords).
Procedure_Name **proc**

…
Block of Instructions
…
Ret ; Return from procedure. THIS IS REQUIRED.

Procedure_Name **endp**

Calling a Procedure
The syntax for calling a defined procedure is simply:

Call Procedure_Name

Notice that “Call” is the keyword.

Example:
The following intel assembly snippet of a program is an example we will use to illustrate the above concepts. The main procedure is used to implement the High Level language statement C=A-B. Notice that A, B, and C are variables. In the example below the main program calls procedure *subtract* to compute A-B. The *subtract* procedure will return output parameter which the main program will place in variable C. I have provided relevant keywords in **bold** for emphasis. In the example below, note how the *main* procedure passes inputs A and B as parameters through registers before calling *subtract* procedure. Let me take this opportunity to point out that, even though *main* and *subtract* are both defined as **proc**edures, *main* does not need a **Ret**urn statement. This is because *main* is a special type of procedure. In fact, it is the first procedure called by the Operating System when the latter is called upon to run your program. Strictly speaking, the EXIT process you have already seen at the end of *main* is partly used to implement **Ret**urn to the Operating System.

```
.data
A WORD 500
B WORD 200
C WORD ?
.code
Main proc
…
; input parameters A=ax, B=bx, output parameter C=ax
Mov ax, A
Mov bx, B
Call subtract
Mov C, ax
…
…
Main endp

Subtract proc
;input parameters in ax and dx. Output of subtraction in ax
Sub ax,bx
Ret
Subtract endp
```

End main

Procedure: Organize the previous lab assignment to utilize a procedure for each step of the sessions below. This means you should create four (actually three. See example below) *procedures*, one for each of the steps of the session. Then the *main* program will call each of the procedures. In this assignment, you will use parameters in the four procedures you will create. Parameter passing will be through registers. I have provided some hints below to help you.

A typical session on the screen will look like the following

Input X= 0040
Input Y= 0024
Calculating W…
Output W= ?

Input X= 0001
Input Y= 0001
Calculating W…
Output W= ?

Input X= 1234
Input Y= 0016
Calculating W…
Output W= ?

NOTE1: AS MENTIONED ABOVE EACH OF THE LINES IN THE ABOVE SESSION MUST BE IMPLEMENTED IN A PROCEDURE. THUS, EXACTLY FOUR CALLS ARE EXPECTED IN THE MAIN PROGRAM (EXCEPT CALLS TO IMPLEMENT NEWLINES), ONE FOR EACH OF THE LINES IN A SESSION.

NOTE2: THE ONLY GLOBAL VARIABLES THAT CAN BE ACCESSED BY PROCEDURES ARE: A) VARIABLES FOR PROMPTING SUCH AS INPUTX, … B) VARIABLES LOC1, LOC2, LOC3, AND SUM.
NOTICE IN THE EXAMPLE BELOW HOW VARIABLES, X, Y, AND W PASSED AS PARAMETERS THROUGH REGISTERS. THUS, THEY SHOULD NOT BE ACCESSED BY ANY PROCEDURE.

HINTS:
1) Declare additional variable for strings in the data area. For instance CalcW BYTE "Calculating W…", '0'. This can be used to display "Calculating W…" either within the main program or within CalcW procedure.

2) Example below provides hints on parameter passing specific to this assignment. You can feel free to use it as a starting point. In this example, a single Procedure called *GetInput* is used for inputting numbers X and Y from keyboard. It is called two times, each time using input parameters for the prompt used to display message for X or Y. Output parameter from a register is placed in the corresponding variable. Similarly, X and Y are passed through registers as input parameters to *CalculateW* procedure which returns the result through ax register. The final procedure *OutputW* has one input parameter but no output parameter since it simply displays the output results on the screen, etc.

```
…
.data
…
.code
Main PROC
…

Mov si, offset InputX          ;input parameter passing
Call  GetInput
Mov X,ax                       ;output parameter passing

Mov si, offset InputY
Call GetInput
Mov Y,ax
```

```
Mov si,X                ;input parameter passing
Mov di,Y                ; input parameter passing
Call calculateW
Mov W, ax               ;output parameter passing

Mov ax,W                ;input parameter passing
Call OutW
…
Main ENDP
```

3) Feel free to use Irvine Library *CrLf* Procedure for creating Newlines. This one has no parameters and is used as needed in the above session.
You may also use Irvine Libraries *ReadChar*, *WriteChar*, and *WriteString*.
You may not use any library functions other than the ones specified above.

Testing: Be sure to test your program and make sure it works before you submit it to your lab instructor on CANVAS as specified below.

Demonstration: Demonstrate your program to the instructor with screen shots similar to the session above. INSTRUCTOR WILL CHECK your source code 1) Whether parameters are passed before calls are made to procedures, 2) Single procedure for Multiple digit inputs, X and Y. 3) Whether global variables are avoided except for the exceptions specified above, 4) Whether global variables are used in procedures illegally. 5) The lab instructor will also assemble and run the documented source code you upload to CANVAS as specified below. The instructor will check whether your program adheres to the session above as specified by your screen shots.

Submission: Submit electronic copy of your program to CANVAS including a well *documented program (source code)* and output (screen shots). **Filenames must be according to the format specified in the syllabus**