Assembly 2 Project

Thanapoom Phatthanaphan

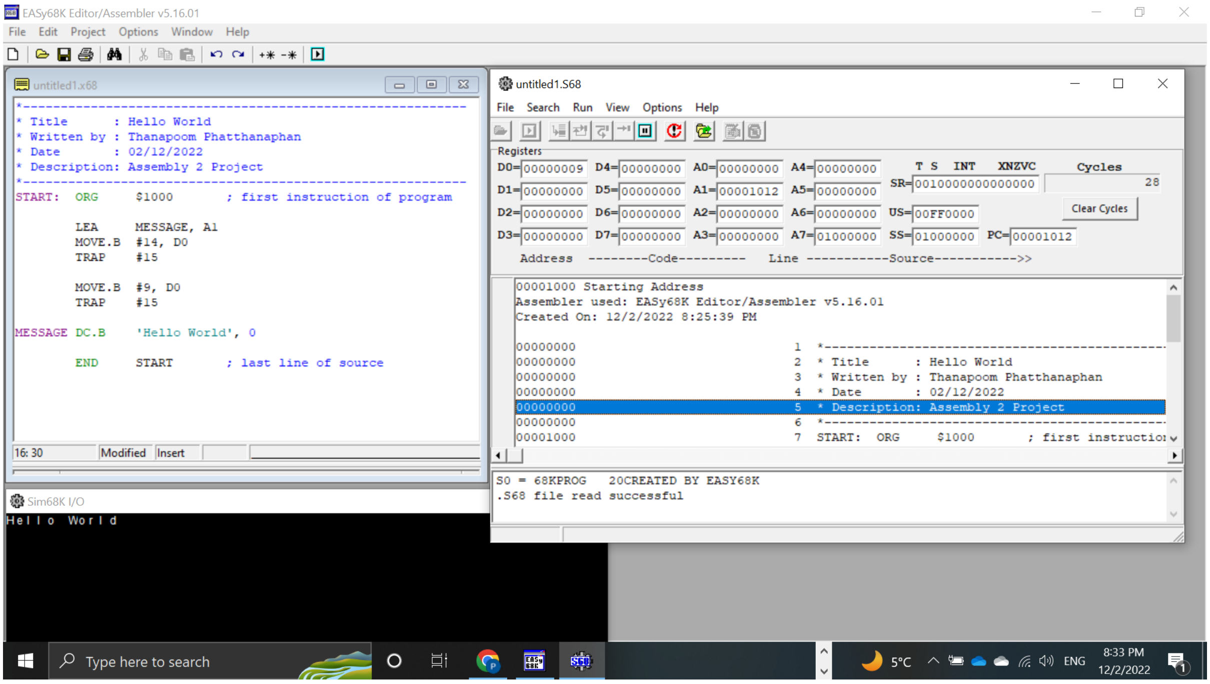
Department of Computer Science, Stevens Institute of Technology

CS 550: Computer Organization and Programming

Dr. Edward Banduk

December 2, 2022

Please use this program to compile your “Hello World” Assembly program. Please take a screenshot picture showing the compilation of your code by using EASY68K software.



```
*-----------------------------------------------------------
* Title      : Hello World
* Written by : Thanapoom Phatthanaphan
* Date       : 02/12/2022
* Description: Assembly 2 Project
*-----------------------------------------------------------
START:  ORG     $1000       ; first instruction of program

        LEA     MESSAGE, A1
        MOVE.B  #14, D0
        TRAP    #15

        MOVE.B  #9, D0
        TRAP    #15

MESSAGE DC.B    'Hello World', 0

        END     START       ; last line of source
```

```
D0=00000009  D4=00000000  A0=00000000  A4=00000000     T S  INT   XNZVC       Cycles
D1=00000000  D5=00000000  A1=00001012  A5=00000000  SR=0010000000000000                28
D2=00000000  D6=00000000  A2=00000000  A6=00000000  US=00FF0000
D3=00000000  D7=00000000  A3=00000000  A7=01000000  SS=01000000  PC=00001012
```

```
  Address  --------Code---------   Line -----------Source----------->>
00001000 Starting Address
Assembler used: EASy68K Editor/Assembler v5.16.01
Created On: 12/2/2022 8:25:39 PM

00000000                             1  *-----------------------------------------------
00000000                             2  * Title      : Hello World
00000000                             3  * Written by : Thanapoom Phatthanaphan
00000000                             4  * Date       : 02/12/2022
00000000                             5  * Description: Assembly 2 Project
00000000                             6  *-----------------------------------------------
00001000                             7  START:  ORG     $1000       ; first instructio
```

```
S0 = 68KPROG   20CREATED BY EASY68K
.S68 file read successful
```

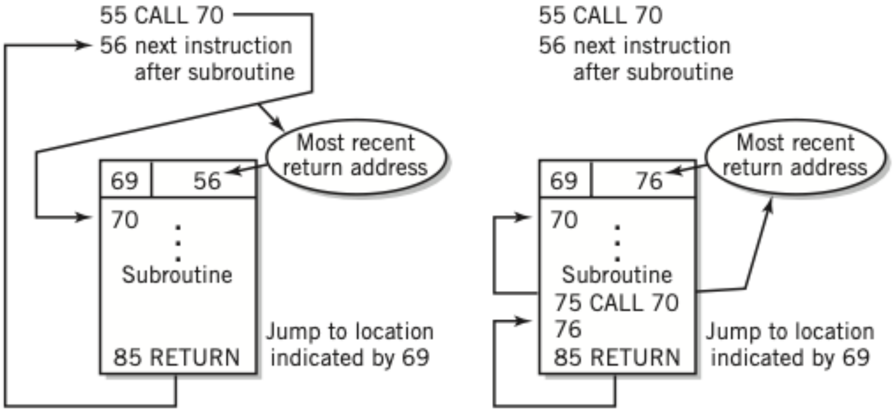
```
Hel l o  Wor l d
```

Part 2: Please explain how the stack is used for subroutine call and return

The stack is used to store data as LIFO (Last-in, First-out) structure. For subroutine CALL and RETURN, when the program uses subroutine CALL then the next instruction is stored in the top of the stack, the program jumps to execute the subroutine. After the subroutine is completed, the program returns to execute the instruction that is stored in the top of the stack and that instruction is popped out from the stack.

FIGURE 7.14

Fixed Location Subroutine Return Address Storage


55 CALL 70
56 next instruction after subroutine
Most recent return address
69
56
70
Subroutine
85 RETURN
Jump to location indicated by 69
55 CALL 70
56 next instruction after subroutine
Most recent return address
69
76
70
Subroutine
75 CALL 70
76
85 RETURN
Jump to location indicated by 69

(a) Subroutine called from loc.55

(b) Subroutine re-called from 75, within the subroutine

From figure 7.14a, the subroutine is called from address 55 to execute the instruction in address 70. The instruction of address 56 is then stored in the top of the stack, the program jumps to execute the instruction in address 70. From figure 7.14b, there is subroutine called from address 75 to execute the instruction in address 70. The instruction of address 76 is then stored in the top of the stack. When the subroutine is completed, the program executes the instruction in address 76 (the top stored instruction of the stack) and then the program returns to execute the instruction in address 76 because of the instruction RETURN in address 85.

Citation: Englander, I. (2014). The Architecture of Computer Hardware, Systems Software, and Networking: An Information Technology Approach (5th Ed., p. 221). John Wiley & Sons,

Part 3: Explain a computer's register-level architecture, including

a) CPU-memory interface

The CPU and Memory are connected together through the Memory Address Register (MAR) and the Memory Data Register (MDR). The MAR holds the address of the memory location. A decoder, which is connected to the MAR, is responsible to interpret the address and activates a single address line into the memory. The MDR holds a data value that is being stored to or retrieved from the memory location currently addressed by the memory address register. For the interaction between the CPU and the memory registers, the CPU is responsible to control memory transfers. It copies an address from some register in the CPU to the MAR in order to retrieve or store data at a specific memory location. When the MAR is loaded, the CPU communicates to the memory unit that the data is retrieved from memory or stored to memory which can be clarified by Read and Write operation. When the interaction being executing, the CPU transfers the data from register in the CPU to the MDR and then transfers it into memory. The previous data at that location will be replaced.

Citation: Englander, I. (2014). The Architecture of Computer Hardware, Systems Software, and Networking: An Information Technology Approach (5th Ed., pp. 198-203). John Wiley & Sons,

b) special-use registers

Registers are permanent storage locations in the CPU that hold binary value temporarily to be used for specific purposes. Registers are manipulated directly by the control unit during the execution of instructions. For general purpose, registers are usually considered to be a part of the arithmetic/logic unit. Registers are also capable to handle for special purposes such as they can hold data being processed, instructions being executed, memory or I/O address to be accessed, or even special binary codes used for some other purpose, such as codes that keep track of the status of the computer or the conditions of calculations.

Citation: Englander, I. (2014). The Architecture of Computer Hardware, Systems Software, and Networking: An Information Technology Approach (5th Ed., pp. 197-198). John Wiley & Sons,

c) addressing modes

Addressing modes are different ways to establish memory addresses within an instruction. Refer to a computer that can use a general-purpose register to hold an address, to find a memory location, the computer uses the value in that register as a pointer to the address. Instead of an address field, the computer uses the instruction to indicate which register holds the address. In order to use this technique, The size of the memory address register is normally required to be as large as the instruction address field, or much larger, because the addressing capabilities of the computer corresponds to the size of the registers.

Citation: Englander, I. (2014). The Architecture of Computer Hardware, Systems Software, and Networking: An Information Technology Approach (5th Ed., p. 204). John Wiley & Sons,