Homework #1

Out: April 11, 2023
Due: April 22, 2023
Value:200 points

For this assignment there is **no collaboration allowed**, except on problem 4. You also may **not** use a debugger or simulator to find the answers to the problems 1, 2, and 3.

1) General Instructions (20 points)

For each of the following instructions assume that the registers contain the following values (in hexadecimal) before each instruction is executed. That is, use the register values below for every instruction.

R0 = 3E	R1 = 00	R2 = 00	R16 = C3	R17 = C5	R18 = CD

Fill in the following table with the contents of R16 and the flags after the specified instruction is executed. Note that for the flags you will need to indicate that some are unchanged after the instruction (use --).

Instruction	R16 Contents After Instruction	Flags After Instruction			
		Z	N	C	V
`MOV R16, R1`					
`NEG R1`					
`DEC R2`					
`SWAP R16`					
`TST R16`					
`CPI R16, 0F0H`					
`ADD R16, R0`					
`LDI R16, 0`					
`SUB R16, R17`					

2) CMP Instruction and Flags (20 points)

Complete the following table. Fill in the flag settings for a `CP R0, R1` instruction and the actual relationship between the values in `R0` and `R1` when they are interpreted as signed or unsigned numbers.

Register		Flags after CP R0, R1					Unsigned Relationship (< = >)	Signed Relationship (< = >)
R0	R1	Z	N	S	C	V		
00	00							
7F	00							
80	00							
FF	00							
00	7F							
80	7F							
FF	7F							
00	80							
7F	80							
FF	80							
00	FF							
7F	FF							
80	FF							
FF	FF							

3) Stack Operations (15 points)

Given the following instructions and initial register contents, give the values stored in memory at locations 0370 to 037F when the instruction at location 6002 is reached. Assume the code begins executing at the instruction at location 2000. List the **bytes** on the stack (at the specified memory locations), not the words. (Note: all stack locations do **not** contain known values.)

R0 = 56	R1 = 78	R2 = A9	R3 = ED	SP = 037F

```
2000        PUSH  R0
2001        PUSH  R1
2002        CALL  FNC1
2004        ...

3300  FNC1: PUSH  R2
3301        IN    R24, SPL
3302        IN    R25, SPH
3303        SBIW  R24, 4
3304        OUT   SPL, R24
3305        OUT   SPH, R25
3306        PUSH  R0
3307        RCALL FNC2
3308        ...

6000  FNC2: PUSH  R1
6001        PUSH  R3
6002        ...
```

Memory	
Address	**Data (Byte**
0370	
0371	
0372	
0373	
0374	
0375	
0376	
0377	
0378	
0379	
037A	
037B	
037C	
037D	
037E	
037F	

4) Hexer Game Functional Specification (80 points)

Write the functional specification for the Hexer game. You should include all the parts of a functional specification as discussed in class. There are many aspects of the design that are left open to you. For example, how game completion is indicated, is it possible to select one out of many games and how, what is stored on the SD card, etc. You should decide how you want the Hexer game to work and then document those decisions in the functional specification.

5) Pseudo-Code Introduction (20 points)

Below is the functional specification and pseudo-code for a program that computes the number of combinations of *n* items taken *m* at a time. There are bugs in the pseudo-code (the functional specification is correct). Find the bugs and fix them. You do not need to fix the "Known Bugs."

Functional Specification

Description: Finds the number of combinations of *n* items taken *m* at a time. The formula for this is:
n! / ((*n* - *m*)! * *m*!).
A value of 0 (zero) for either *n* or *m* terminates the program.

Input:	Integer values of *n* and *m*.
Output:	The number of combinations of *n* items taken *m* at a time.
User Interface:	The user is prompted for the values of *n* and *m* and then the number of combinations of *n* items taken *m* at a time or an error message is output. If a zero is input for either *n* or *m* the program terminates without outputting an error message.
Error Handling:	An error message is output if not exiting and *n* or *m* is negative or $n < m$.
Algorithms:	Uses the above formula.
Data Structures:	None.
Known Bugs:	There is no overflow checking and only integer input is allowed.
Limitations:	None.

Pseudo-Code

```
REPEAT
    output(prompt)
    n, m = get_input()
    IF ((n >= 0) AND (m >= 0) AND (n > m)) THEN
        result = fact(n) / (fact(n-m) * fact(m))
        output(result)
    ELSE
        IF (n < m) THEN
            output(error message n < m)
        ELSE
            output(error message n or m negative)
        ENDIF
    ENDIF
UNTIL ((n <= 0) AND (m <= 0))
exit
```

Procedure: fact

Functional Specification

Description:	Computes and returns the factorial of the number passed as an argument.
Arguments:	Non-negative integer for which to find the factorial (*n*).
Return Value:	Factorial of the argument.
Input:	None.
Output:	None.
Error Handling:	No error checking is done.
Algorithms:	$n! = 1 * 2 * 3 * ... * (n-2) * (n-1) * n$
Data Structures:	None.

Pseudo-Code

```
FOR i = 1 TO n UPDATE +1
    result = result * i
ENDFOR
RETURN result
```

Procedure: get_input (supplied)

Description: Gets input from the user.

Procedure: output (supplied)

Description: Outputs its argument(s) to the user.

6) Switch Hardware (15 points) (due 4/29/23)

Using the debugger, determine the status of the switches and rotary encoders requested by the TA.

7) LED Hardware (15 points) (due 5/13/23)

Using the debugger, turn on the LEDs (one at a time) requested by the TA.

8) Timer/Tone Hardware (15 points) (due 5/27/23)

Using the debugger, generate the tones (one at a time) requested by the TA.

Homework #1 Resources

- Hexer Game - System Requirements and Description
- Homework Q&A

Last updated April 12, 2023 05:18 PM by glen@caltech.edu