CPE 233 Software Assignment 5

Stack Operations

Report by:
Ethan Vosburg (evosburg@calpoly.edu)
Mateo Vang (mkvang@calpoly.edu)

February 17, 2024

Table of Contents

1 Flow Charts

1.1 Division Flowchart

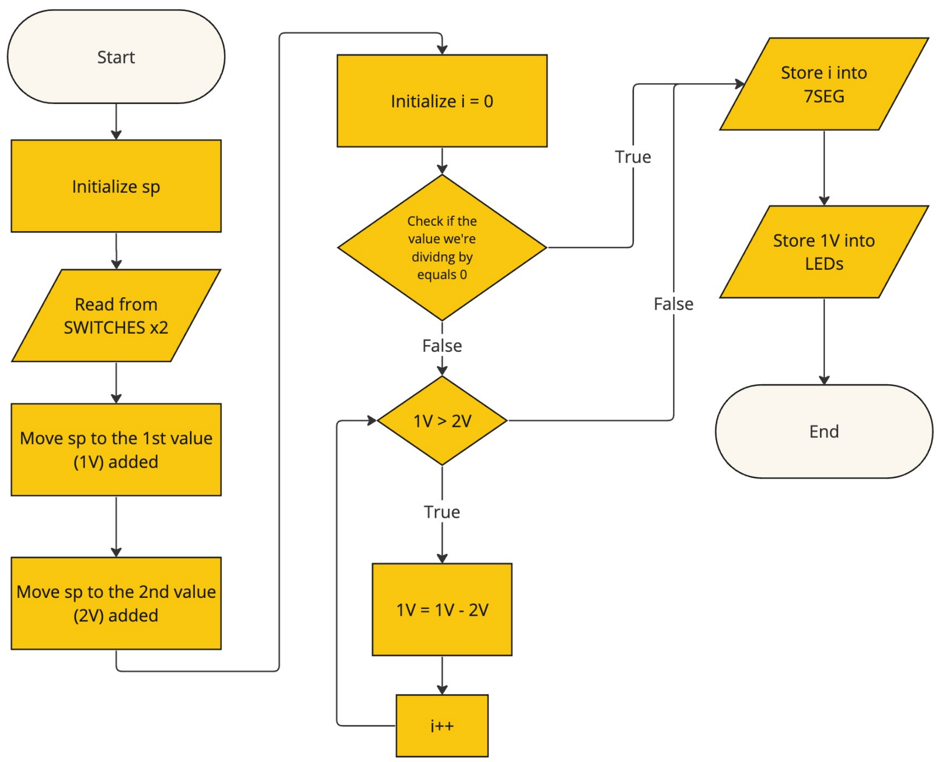

Figure 1: Stack Division Flowchart

1.2 Stack Reading and Writing Flowchart

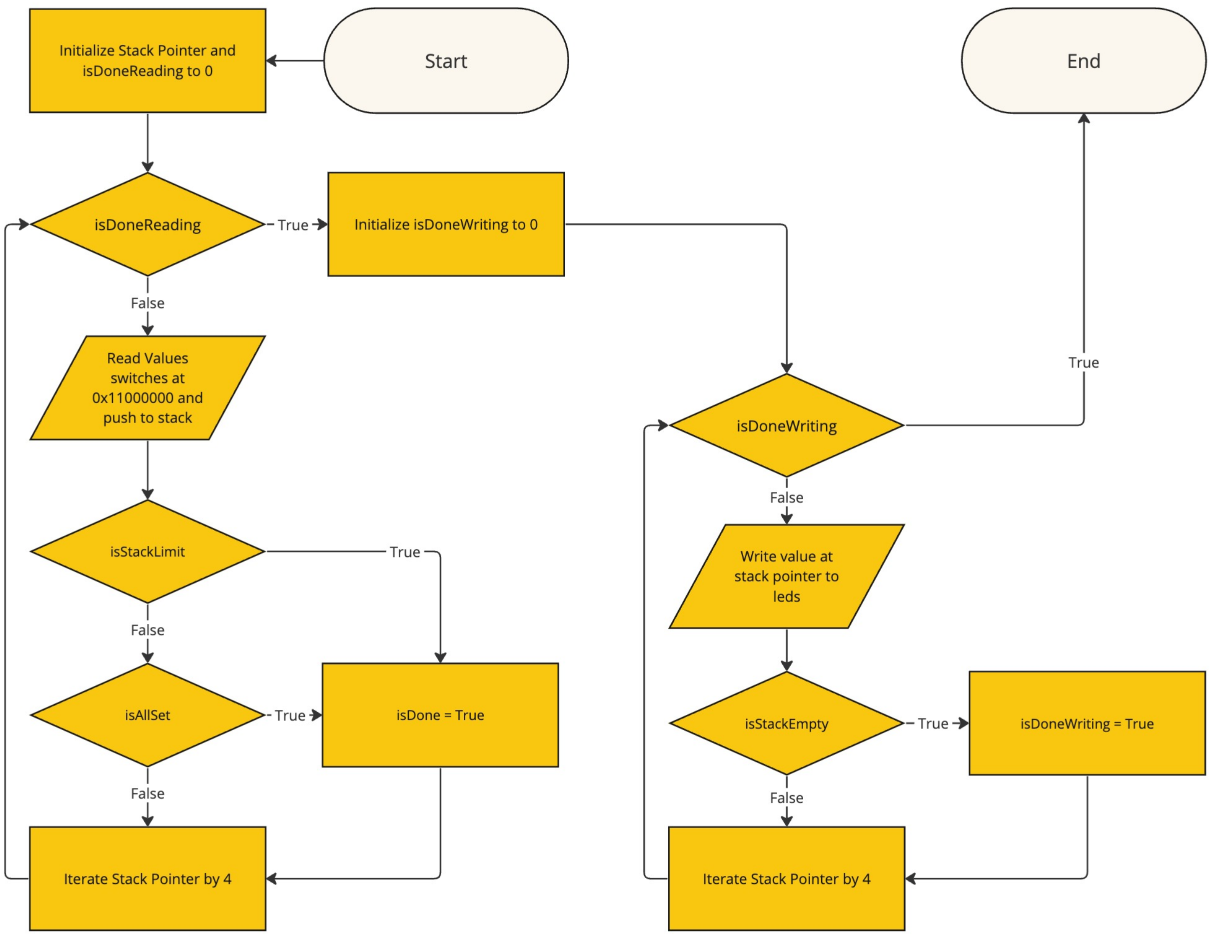

Figure 2: Loading and Unloading Stack

2 Assembly Instructions

2.1 Division

```
# file: StackDivisionSW5.asm
# brief: Using the stack pointer to store values in the STACK memory addresses
#    then dividing the first value stored by the second value stored.
# author: Mateo Vang
# date: 2/16/2024
lui sp, 0x10 # stack pointer
lui s3, 0x11000 # address of SWITCHES
# stores the 1st value read(1V) into the stack
lh t0, (s3)
addi sp, sp, -4
sh t0, 0(sp)
# stores the 2nd value read(2V) into the stack
lh t0, (s3)
addi sp, sp, -4
sh t0, 0(sp)

lh t0, 4(sp) # 1V
lh t1, 0(sp) # 2V
li t2, 0 # our counter which acts like our result which we will store into 7SEG
beqz t1, endSUB # just jump straight to the end of the program tries to divide by zero

    SUB: blt t0, t1, endSUB # Subs t0 by t1 if t0 > t1
         sub t0, t0, t1 # At the end, t0 will be our remainder
         addi t2, t2, 1 # This will be our result
    j SUB

    endSUB: sh t2, 0x40(s3) # stores result into 7SEG
            sh t0, 0x20(s3) # stores remainder into LEDs
```

Listing 1: Assembly Code for Division in Figure 1

2.2 Stack Reading and Writing

```
# file: SW5-StackOperation.asm
# brief: Assembly code for stack operations
#
# This file contains the assembly code for loading the stack with values and
# then popping them off and displaying them on the LED's.
#
# author: Ethan Vosburg
# date: 02-16-2024

    lui s0, 0x11000             # Declare MMIO base address
    li  s1, 0xffffffff          # Declare comparison value
    li  sp, 0x10000             # Declare stack pointer
    li  s2, 0x10000             # Declare end of stack

valueLoad:
    lw  t0, 0(s0)               # Get values from switches
    beq t0, s1, valueDisplay    # Branch if value is 0xffffffff
    addi sp, sp, -4             # Increment the stack pointer
    sw  t0, 0(sp)               # Push the value onto the stack
    j valueLoad


valueDisplay:
    beq sp, s2, END             # Branch if the stack is empty
    lw t0, 0(sp)                # Get the value from the stack
    addi sp, sp, 4              # Pop the value off the stack
    sw t0, 0x20(s0)             # Display the value on the LED
    j valueDisplay

END:
```

Listing 2: Reading and Writing to Stack in Figure 2

3 RARS Verification

3.1 Division Verification

The test cases below demonstrate the code performs the desired outputs.

Table 1: Flow Chart 1 Test Cases

Test Cases	Stored to SevenSeg	Stored to LEDs
24/2	0xC	0x0
12/5	0x2	0x2
6/10	0x0	0x6
20,000/0	0x0	0x4eE20
4/4	0x1	0x0
0/10	0x0	0x0

1. Test case 1 shows that the program knows to stop the division process once it's complete.
2. Test case 2 shows that the program knows how to store both the result and remainder into their respective memory addresses.
3. Test case 3 shows that the program knows how to divide bigger numbers by smaller numbers.
4. Test case 4 shows that the program knows what to do if something is divided by zero.
5. Test case 5 shows that the program can correctly divide a number by itself.
6. Test case 6 shows that the program can correctly divide zero by any number.

3.2 Stack Reading and Writing Verification

Table 2: Flow Chart 2 Test Cases

Test Cases	Expected Output	Actual Output
0x0, 0x0, 0x0, 0xffffffff	0x0, 0x0, 0x0,	0x0, 0x0, 0x0,
0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0xffffffff	0x5, 0x4, 0x3, 0x2, 0x1, 0x0	0x5, 0x4, 0x3, 0x2, 0x1, 0x0
0xae00bc09, 0xbbeeffaa, 0x09793638, 0xeb78c0de, 0xffffffff	0xeb78c0de, 0x09793638, 0xbbeeffaa, 0xae00bc09	0xeb78c0de, 0x09793638, 0xbbeeffaa, 0xae00bc09
0xfffffffa, 0xfffffffb, 0xfffffffc, 0xffffffff	0xfffffffc, 0xfffffffb, 0xfffffffa	0xfffffffc, 0xfffffffb, 0xfffffffa

The test cases above demonstrate the code produces the desired outputs.

1. Test case 1 test all zeros.
2. Test case 2 test incremental input.
3. Test case 3 test random large numbers.
4. Test case 4 test numbers nearing the limit of the word.