EE 306 Microprocessors
Spring 2023 – 2024
Midterm Exam
28 April 2024

Q1	Q2	Q3	Q4	Q5

Full Name: SOLUTIONS	
Student ID:	
Signature:	
# extra solution pages:	Bonus?☐

Submit this booklet, cheat sheet, bonus, and extra solution sheet(s). All must have your full name and signature. Note that there are 5 questions. In the code pieces, write inline comments. Good Luck!

1. (15 pnt) This program stores 0,1,2,3…n on the stack. Fill in the blanks according to the comments and then answer the questions.

```
      MOV___ R0, #0xC        // Write 12 into the register R0

      MOV___ R1, #5          // Write 5 into the register R1

Loop: SUBS__ R1, R1, #1      // Decrement R1 by 1. Flags may be affected

      BEQ Done

      PUSH__ {R0}            // Save the content of R0 on stack

      ADD___ R0, R0, #1      // Increment R0 by 1. Don't care the flags.

      B Loop

Done: B Done_________        // when you are done, stuck here forever.
```

a) You can see a for loop in the program. How many times does this Loop run? 4 (5 is accepted, if consisted with b)

b) Assume SP is pointing to address 0x600 before the program is executed. What is the value of SP after the program is executed, in hexadecimal? _0x5F0 or 0x5EC (if a is 5) _____

2. (10 pnt) Answer the short questions.

a) How many "select" bits does a 4x1 MUX have? ____2_____

b) Show A and B in 4-bit binary format. A = 12, B = 5.

$(A)_2$ = _____1100_____ $(B)_2$ = _____0101_____

c) R0: 12, R1: 32, R2: 16, R3: 0 **Calculate the effective address. All numbers are in decimal.**

i. STR R0, [R1, R2, LSR #1] EA = _____40_____

ii. STRB R1, [R0, R2, LSL #1] EA = _____44_____

iii. STR R2, [R0, R1]! EA = _____44_____

iv. STR R3, [R2], R1 EA = _____16_____

3. (50 pnt) Write a program for vote counting. The ballot reader is connected to the system as an input device. There are 3 parties in the elections and they are named: 0, 1, and 2.

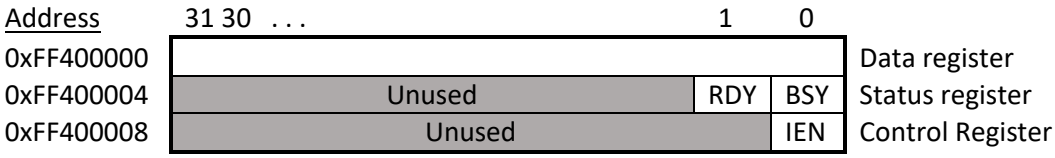

Figure 1: Ballot Reader

The device registers are shown in Figure 1.

Data register: The party name is delivered in the data register.

Status Register: When a new vote is entered to the reader, the RDY flag is set. Note that the device does not read a new ballot unless the RDY flag is acknowledged by the user. **Write '1' on the RDY flag to acknowledge the device**. BSY flag shows the counting is in process. The microprocessor should not change this flag: if you have to write on the status register, keep BSY value in the LSB.

Control Register: The device can send interrupt request if the IEN (interrupt enable) bit is 1.

a. Write a subroutine called READVOTE to read the vote when a new vote is available. This subroutine passes the party name to the main in the register R4.
b. The scores are stored in the memory, in an array called VOTES. The array is organized as: [score_party0, score_party1, score_party2, number_of_invalids]. Write a subroutine called COUNTVOTE. This subroutine reads the party name on R4. If the name is not one of the valid party names (if the name is greater 2) the subrotine increments the number of invalid votes by one. Else, the subrotines increments the number of votes for the party *i* (score_party0, score_party1, or score_party2).
c. Write a program to read and count 99 votes. Initialize your registers as necessary. Write a loop to complete 99 votes. Use READVOTE and COUNTVOTE subroutines.
d. Assume there is no tie, find the party with the highest votes. Write the party ID to the WINNER location in the memory.

```
READVOTE: PUSH {R0,R1,R2,LR}
          LDR R0, =0xFF400000
Wait:     LDR R1, [R0,#4]          //read status
          ANDS R2, R1, #0b10       //mask RDY bit
          BEQ Wait
          STR R1, [R0,#4]          // acknowledge
          LDR R4, [R0]             // read the vote
          POP {R0,R1,R2,PC}        // go back


COUNTVOTE: PUSH {R0,R1,R2,LR}
          CMP R4, #2
          MOVGT R1, #3             // invalids index
          MOVEQ R1, #2             // party2 index
          CMP R4,#1
          MOVEQ R1, #1             // party1 index
          CMP R4,#0
          MOVEQ R1, #0             // party0 index
          LDR R0, = VOTES
          LDR R2, [R0, R1, LSL #2] // read the voted party
          ADD R2, R2, #1           // increment its count
```

```
                STR R2, [R0, R1, LSL #2]
                POP {R0,R1,R2,PC}               // go back


                MOV R3, #100
Loop:           SUBS R3, R3, #1
                BEQ Findmax
                BL READVOTE
                BL COUNTVOTE
                B Loop

Findmax:        LDR R0, =VOTES
                MOV R4, #0              //initial winner party
                MOV R2, #0              //initial maximum number of votes
                MOV R1, #0              //index
Loopmax:        LDR R3, [R0, R1, LSL #2] //read party_i vote count
                CMP R3, R2
                MOVGT R2, R3            // new max vote count
                MOVGT R4, R1            // new winner index
                ADD R1, R1, #1          // update index
                CMP R1, #3              // run for r1= 0, 1, and 2
                BLT Loopmax

                LDR R0, =WINNER
                STR R4,[R0]
Done:           B Done
```

---- note that this is one correct answer. ---

BONUS

```
                LDR R0, =VOTES
                LDR R0, [R0,#0xC]               // number of invalid votes

                MOV R1, #0
Tens:           CMP R0,#10              // divide to 10 to find 10s digit.
                BLT Print
                ADD R1,R1,#1
                SUB R0,R0,#10
                B Tens
Print:          LDR R2, =HEXTABLE       //
                LDRB R0, [R2, R0]       // 7segment character of ones digit.
                LDRB R1, [R2, R1]       // 7segment character of tens digit.
                ORR R0, R0, R1, LSL #8  // combine tens and ones
                LDR R2, =0xFF200020
                STR R0, [R2]
```

4. (20 pnt) You have an 8-bit interrupt controller who can handle up to 16 interrupt requesting devices. You can enable the forwarding from *device_i* by setting the correct bit of the correct **Interrupt Set Enable Register**. You can assign priority to devices by wriring to the correct **Interrupt Priority Register**.

Address	7	6	5	4	3	2	1	0	Register name
0xF0									Interrupt Set Enable Register_0
0xF1	1					1			Interrupt Set Enable Register_1
0xF2									Interrupt Priority Register_0
…									…
…									…
0xF9									Interrupt Priority Register_7

Write assembly code piece to configure the controller to forward interrupts from 2 devices: Fire Alarm, and Door Bell. Their interrupt IDs are 10 and 15, respectively. Fire Alarm has higher priority than the Door Bell. Use ARM assembly language.

```
// enable bit 10 and bit 15. They are both on Interrupt Set Enable Register_1.
LDR R0, =0xF1  // pointer
MOV R1, #0b10000100 // byte to be written on Interrupt Set Enable Register_1.
STRB R1, [R0]  // STR is also accepted however, you cannot write a word to address 0xF1
// let priority of fire alarm and door bell be 1 and 5, respectively.
LDR R0, =0xF7  // pointer for door bell, lower order 4 bits.
MOV R1, #b0101
STR R1, [R0]
LDR R0, =0xF9  // pointer for door bell, higher order 4 bits.
MOV R1, #b00010000
STR R1, [R0]
```

5. (5 pnt) The private timer of ARM Cortex A9 has a 200 MHz clock. Write the program piece to configure the timer to generate ticks at every 100 ms. Do not check the ticks.

Address	31 … 16	15 … 8	7 … 3	2	1	0	Register name
0xFFFEC600	Load value						Load
0xFFFEC604	Current value						Counter
0xFFFEC608	Unused	Prescaler	Unused	I	A	E	Control
0xFFFEC60C	Unused					F	Interrupt status

```
LDR R0, =0xFFFEC600  // pointer to the device
LDR R1, =20000000    // count from 20 million to generate 0.1 second delay.
STR R1, [R0]
MOV R1, #0b011          // A = 1, E = 1
STR R1, [R0]
```