University of Toronto
Faculty of Applied Science and Engineering
Department of Electrical and Computer Engineering

Midterm Test
March 3, 2022

ECE243 – Computer Organization

Examiners – Prof. Stephen Brown and Prof. Jonathan Rose

SOLUTIONS

1. There are **5** questions and **20** pages. Do **all** questions. The test duration is **1 hour and 45 minutes**.
2. **ALL WORK IS TO BE DONE ON THESE SHEETS.** You can use the blank pages included on Pages 17 – 20 if you need more space for any question. Be sure to indicate clearly where your work continues on those pages.
3. Closed book. An **Aid Sheet** is included for your reference starting on Page 15.
4. No calculators are permitted.

[15 marks] 1. Short answers:

[4 marks] (a) Consider the sequence of ARM instructions shown below. Assume that this code is stored in the memory starting at address 0. Also, assume that when the program starts register R0 = 0, and R1 = 0. Fill in the comment field to the right of each instruction to show the contents of both registers R0 and R1 **after** the instruction on that line of code is executed. Show your answers in hexadecimal.

```
        .global _start

_start: LDR     R0, =X          // R0 = 0x10, R1 = 0

        LDR     R1, [R0]        // R0 = 0x10, R1 = 0xA

        LDR     R1, [R0, #4]    // R0 = 0x10, R1 = 0x14

        LDR     R1, [R0], #4    // R0 = 0x14, R1 = 0xA

X:      .word   10, 20
```

[4 marks] (b) The sequence of ARM instructions below loads some data values from memory into register R1. Fill in the comment field to the right of each instruction to show the contents of register R1 **after** the instruction on that line of code is executed. Show your answers in hexadecimal.

```
        .global _start

_start: LDR     R0, =DATA

        LDR     R1, [R0]        // R1 = 0xAABBCCDD

        LDRB    R1, [R0]        // R1 = 0xDD

        LDRB    R1, [R0, #1]    // R1 = 0xCC

        LDRB    R1, [R0, #3]    // R1 = 0xAA

DATA:   .word   0xAABBCCDD
```

Question 1 continued ...

[7 marks] (c) The ARM code below performs a simple calculation and puts the final result into register `R1`. When performing its calculation, the program calls a subroutine named `DOUBLE`.

```
         .global _start

_start:  MOV     R0, #3
         MOV     R1, #3
CALL:    BL      DOUBLE
BACK:    ADD     R1, R0
HERE:    B       HERE

DOUBLE:  ADD     R0, R0
         MOV     PC, LR
```

Assume that this program is stored in the memory starting at the address 0. Also assume that an ARM processor has executed this code up to, but *not including* the instruction at label `HERE`. Fill in the values of the registers shown below. Give your answers in hexadecimal.

PC 0x10 LR 0xC R1 0x9

Assume now that we want to call this same subroutine, `DOUBLE`, but for some particular reason we do not want to execute the `BL` instruction to call the subroutine. On the next page there are two alternative versions of the program that attempt to use a normal branch instruction `B` to "call" the subroutine. For each of these alternative versions, draw a circle around the correct assessment: `YES`, or `NO`. Circle the word `YES` if the code produces the same result in register `R1` as the original, and circle `NO` if the code does not produce the same result in register `R1` as the original.

Question 1 continued ...

```
// This is the first alternative version
        .global _start

_start: MOV     R0, #3
        MOV     R1, #3
        LDR     LR, =BACK
CALL:   B       DOUBLE
BACK:   ADD     R1, R0
HERE:   B       HERE

DOUBLE: ADD     R0, R0
        MOV     PC, LR
```

Assessment: YES

```
// This is the second alternative version
        .global _start

_start: MOV     R0, #3
        MOV     R1, #3
        LDR     LR, =CALL
CALL:   B       DOUBLE
BACK:   ADD     R1, R0
HERE:   B       HERE

DOUBLE: ADD     R0, R0
        ADD     LR, #4
        MOV     PC, LR
```

Assessment: YES

[15 marks] 2. This question is based on the ARM assembly language program below. You will need to understand how the program works, answer some questions about the program's behaviour, and add some new features to the program.

```
        .global _start

_start: LDR     R9, =0xFF200000
MAIN:   LDR     R1, =HEX3_0
        LDR     R2, =HEX5_4
        MOV     R10, #16

LOOP:   LDR     R3, [R1]
        STR     R3, [R9, #0x20]
        LDR     R3, [R2]
        STR     R3, [R9, #0x30]

        SUBS    R10, #1
        BLT     MAIN
PRESS:  LDR     R0, [R9, #0x50]
        CMP     R0, #0
        BEQ     NEXT
WAIT:   LDR     R0, [R9, #0x50]
        CMP     R0, #0
        BNE     WAIT

NEXT:   ADD     R1, #4
        ADD     R2, #4

        LDR     R8, =250000      // delay value
DELAY:  SUBS    R8, #1           // software delay
        BNE     DELAY            // loop
        B       LOOP

HEX3_0: .word   0x00000001, 0x00000002, 0x00000004, 0x00000008
        .word   0x00000800, 0x00080000, 0x08000000
        .word   0x0, 0x0, 0x0, 0x0, 0x0, 0x0
        .word   0x01000000, 0x00010000, 0x00000100
HEX5_4: .word   0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
        .word   0x00000008, 0x00000800, 0x00001000, 0x00002000
        .word   0x00000100, 0x00000001
        .word   0x0, 0x0, 0x0
```

Question 2 continued ...

[2 marks] (a) Assume that the program's machine code starts in memory at address 0. When the program reaches the label `PRESS` in the code for the *first* time, what will be the value of the condition code flags listed below.

Z 0 N 0

[2 marks] (b) The first time the code reaches the label `PRESS` describe the appearance of the HEX5-0 displays. That is, which segment(s) would be illuminated at this time? You can draw a diagram to help with your explanation, if you like.

ANSWER:

The top segment on HEX0 will be illuminated. All other segments will be off.

[2 marks] (c) Describe what you would see on the HEX5-0 displays at the point in time when the code reaches the label `PRESS` and register R10 has the value 4. That is, which segment(s) would be illuminated at this time? You can draw a diagram to help with your explanation, if you like.

ANSWER:

The top segment on HEX5 will be illuminated. All other segments will be off.

[3 marks] (d) Describe briefly what you would observe on the HEX5-0 displays if the code were running continuously. You can draw a diagram to help with your explanation, if you like.

ANSWER:

One segment will be lit at a time, and it will appear to rotate in a clockwise direction around the outside of the six 7-segment displays.

Question 2 continued ...

[6 marks] (e) For this part you are to modify the program so that it uses the ARM Private Timer to implement a delay, rather than using the software delay loop. You have to fill in two sections of code in the spaces given below: 1. At the label `_start`, configure the ARM Private Timer to provide 0.1 second timeouts, 2. At the label `DELAY`, use polled-I/O to wait for each timeout from the timer. Recall that the timer uses a clock frequency of 200 MHz.

```
        .global _start
_start:
        LDR     R8, =0xFFFEC600     // ARM A9 Private Timer
        LDR     R0, =20000000       // 1/(200 MHz) x 2x10**7
                                    //    = 0.1 sec
        STR     R0, [R8]            // write to load reg
        MOV     R0, #0b011          // auto mode, enable
        STR     R0, [R8, #0x8]      // control reg

        LDR     R9, =0xFF200000
MAIN:   LDR     R1, =HEX3_0
        LDR     R2, =HEX5_4
        MOV     R10, #16

LOOP:   LDR     R3, [R1]
PRESS:  ... some of the code is not shown here
NEXT:   ADD     R1, #4
        ADD     R2, #4

DELAY:  LDR     R0, [R8, #0xC]      // read timer status
        CMP     R0, #0
        BEQ     DELAY
        STR     R0, [R8, #0xC]      // reset timer flag bit

        B       LOOP
... the data is not shown here
```

[10 marks] 3. For this question you are given an assembly language program and asked to describe its behaviour, and then to translate it into an equivalent C program. The assembly language program is shown below:

```
          LDR   R5, =0xFF200000
MAIN:     MOV   R0, #0
          LDR   R4, [R5, #0x40]

LOOP:     MOV   R2, R4
          ANDS  R2, #1
          BEQ   NOT_ONE
          ADD   R0, #1
NOT_ONE:  LSR   R4, #1
CONT:     SUBS  R4, #0
          BNE   LOOP

          LDR   R1, =SEG7
          ADD   R1, R0
          LDRB  R2, [R1]
          STR   R2, [R5, #0x20]
          B     MAIN

SEG7:     .byte 0b00111111       // '0'
          .byte 0b00000110       // '1'
          .byte 0b01011011       // '2'
          .byte 0b01001111       // '3'
          .byte 0b01100110       // '4'
          .byte 0b01101101       // '5'
          .byte 0b01111101       // '6'
          .byte 0b00000111       // '7'
          .byte 0b01111111       // '8'
          .byte 0b01100111       // '9'
          .byte 0b01110111       // 'A'
```

[4 marks] (a) What does this program "do"? Explain briefly, in the space below, what you would observe on a DE1-SoC board if you were to execute this program. **Answer:**

This program shows on HEX0 a count of the number of SW bits that are set to 1. The number is displayed in hexadecimal.

Question 3 continued ...

[6 marks] (b) In the space below, write a program in the C language that has the same functionality as the assembly language program given in this question. For convenience, a character (byte) array is provided, which you should make use of in your solution. As a reminder, the right-shift operator in C code is >>.

SOLUTION

```
char seg7[] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07,
    0x7f, 0x67, 0x77};

int main(void)
{
    volatile int *SW_ptr      = 0xFF200040;     // SW port
    volatile int *HEX3_0_ptr = 0xFF200020;     // HEX3_HEX0 port
    int value;
    int count;

    while(1) {
        value = *SW_ptr;                 // read SW
        count = 0;
        while (value != 0){
            if (value & 1)
                count = count + 1;
            value = value >> 1;
        }
        *HEX3_0_ptr = seg7[count];
    }
}
```


```
char seg7[] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07,
    0x7f, 0x67, 0x77};

int main(void)
{
```

[10 marks] 4. The C code shown below implements a bubble sort algorithm. It uses the LIST array to obtain the data to be sorted. The first element in the array gives the number of data elements, and the rest of the array provides the data itself. In the example below, there are 10 items to be sorted, and they are originally in a random order. The bubble sort algorithm sorts this data *in-place* (meaning that it changes the LIST array in memory into a sorted list) in descending order.

```
int LIST[] = {10, 5, 1, 7, 3, 4, 0, 6, 2, 9, 8};

int main(void)
{
    int i, tmp, flag, len;

    len = LIST[0];     // number of items to be sorted
    do {
        flag = 0;      // if flag remains 0, the list is sorted
        for (i = 1; i < len; i = i + 1) {
            if (LIST[i] < LIST[i + 1]) {
                tmp = LIST[i];
                LIST[i] = LIST[i + 1];
                LIST[i + 1] = tmp;
                flag = 1;
            }
        }
        len = len - 1; // the list is partially sorted
    } while (flag);
}
```

On the following page there is an assembly language version of this C code. This assembly language version contains some logical errors, which you are to identify and fix.

Question 4 continued ...

```
1                .text
2                .global _start
3
4  _start:       LDR      R4, =LIST
5                LDR      R10, [R4]
6
7  DO_WHILE:     MOV      R8, #0
8                MOV      R9, #1
9
10 FOR:          CMP      R10, R9
11               BEQ      END_FOR
12               LSL      R9, #2
13               LDR      R5, [R4, R9]
14               ADD      R11, R9, #1
15               LDR      R6, [R4, R11]
16               CMP      R5, R6
17               BGT      CONT
18
19               STR      R5, [R4, R11]
20               STR      R6, [R4, R9]
21               MOV      R8, #1
22
23 NO_SWAP:      LSR      R9, #2
24 CONT:         ADD      R9, #1
25               B        FOR
26
27 END_FOR:      SUB      R10, #4
28               CMP      R8, #1
29               BNE      DO_WHILE
30
31 END:          B        END
32
33 LIST:         .word  10, 5, 1, 7, 3, 4, 0, 6, 2, 9, 8
```

The above program contains a number of logical errors. In the space on the following page, provide a corrected version of the code. You can either show all of the code, or else show only the lines of code that you corrected. Either way, indicate clearly where you have made changes to the code, for example by using the line numbers shown in the code, or circling/underlining your corrections. Do *not add any additional lines of code* to fix the errors; just correct the errors in the code that is there.

Question 4 continued ...

PROVIDE YOUR CORRECTED CODE IN THE SPACE BELOW:

SOLUTION

```
                .text
                .global _start

_start:         LDR     R4, =LIST
                LDR     R10, [R4]     // len = LIST[0]

DO_WHILE:       MOV     R8, #0        // flag = 0
                MOV     R9, #1        // i = 1

FOR:            CMP     R10, R9
                BEQ     END_FOR
                LSL     R9, #2
                LDR     R5, [R4, R9]
                ADD     R11, R9, #4
                LDR     R6, [R4, R11]
                CMP     R5, R6
                BGT     NO_SWAP

                STR     R5, [R4, R11]
                STR     R6, [R4, R9]
                MOV     R8, #1

NO_SWAP:        LSR     R9, #2
CONT:           ADD     R9, #1
                B       FOR

END_FOR:        SUB     R10, #1
                CMP     R8, #0
                BNE     DO_WHILE

END:            B       END

LIST:           .word  10, 5, 1, 7, 3, 4, 0, 6, 2, 9, 8
```

[10 marks] 5. Trace an ARM Program:

Consider the ARM code shown below. This code calls a subroutine, `FUNC`, which is recursive (it calls itself). You are to *trace* the execution of this program. Note that the address that each instruction would have in the memory is shown to the left of the code.

```
                    .text
                    .global _start
          _start:
00000000            LDR     SP, =0x20000

00000004            LDR     R4, =X
00000008            LDR     R0, [R4], #4
0000000C            LDR     R1, [R4], #4
00000010            BL      FUNC
00000014            STR     R0, [R4]

00000018  END:      B       END

0000001C  FUNC:     PUSH    {LR}
00000020            PUSH    {R3}
00000024            MOV     R3, R0
00000028            CMP     R1, R0
0000002C            BLT     RETURN
00000030  AGAIN:    SUB     R1, R3
00000034            BL      FUNC
00000038            MOV     R1, R0
0000003C  RETURN:   MOV     R0, R1
00000040            POP     {R3}
00000044            POP     {PC}

          X:        .word   2
          Y:        .word   5
          M:        .space  4
```

(a) If this program is executed on the ARM processor, what would be the values of the ARM registers listed on the next page the **first** time the code reaches, but has not yet executed, the instruction at address 0x38. Also, show in the space provided the contents of the stack in memory at this point in time (fill in the memory addresses on the left, and show the data stored in each location). Give your answers in hexadecimal. For memory values that are not known, if any, write N/A in the corresponding box.

Question 5 continued ...

R0 1	R1 1	R3 2
R13 0x1FFF0	R14 0x38	R15 0x38

SOLUTIONs

Memory Address	Content
1FFE8	2
1FFEC	0x38
1FFF0	2
1FFF4	0x38
1FFF8	N/A
1FFFC	0x14
20000	N/A

(b) **IMPORTANT**: this final part of the question is worth **2 bonus marks**, which means that it is **optional**. Since the `FUNC` subroutine is recursive, it is not easy to understand what it produces as a result. To earn bonus marks, in the space below describe what the program "produces". That is, what is the relationship between X, Y and M?

Answer

SOLUTION:
M = Y mod X

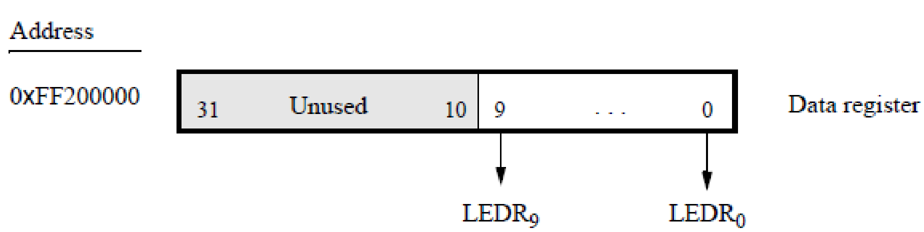
Answer ________________________________

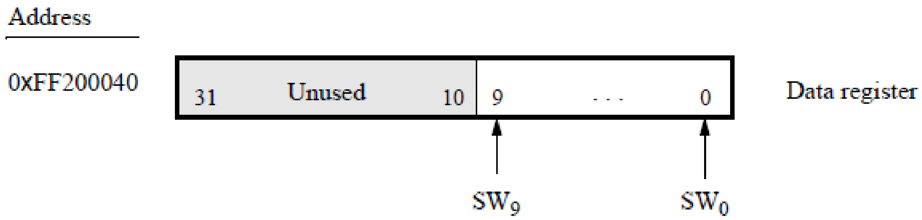
Aid Sheet

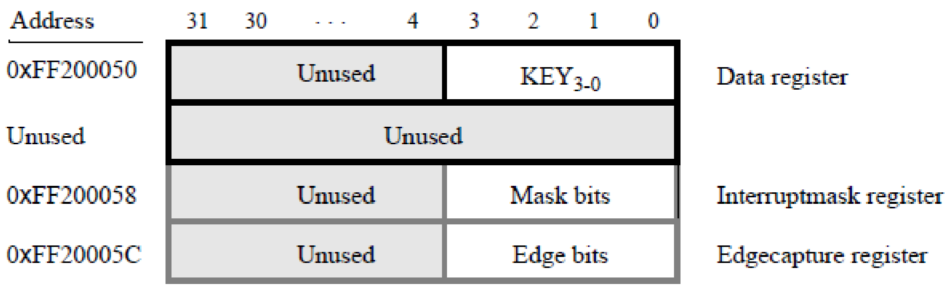
ARM Addressing Modes

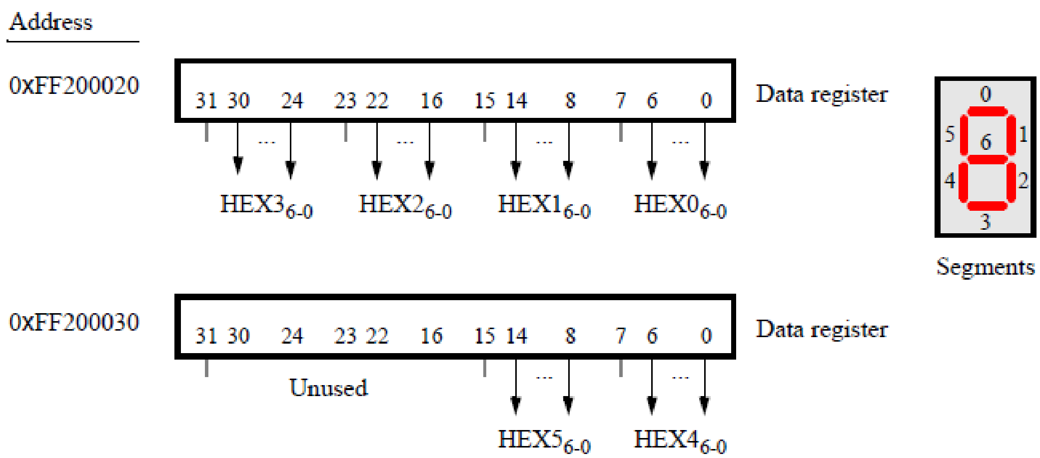
Name	Assembler syntax	Address generation
Offset:		
immediate offset	[Rn, #offset]	Address = Rn + offset
offset in Rm	[Rn, ±Rm, shift]	Address = Rn ± Rm shifted
Pre-indexed:		
immediate offset	[Rn, #offset]!	Address = Rn + offset; Rn ← address
offset in Rm	[Rn, ±Rm, shift]!	Address = Rn ± Rm shifted; Rn ← address
Post-indexed:		
immediate offset	[Rn], #offset	Address = Rn; Rn ← Rn + offset
offset in Rm	[Rn], ±Rm, shift	Address = Rn; Rn ← Rn ± Rm shifted

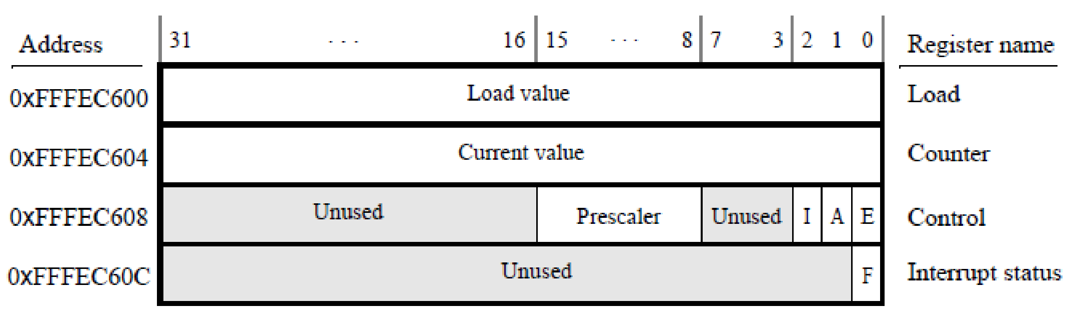
I/O Ports in the DE1-SoC Computer




Address
0xFF200040
31
Unused
10
9
. . .
0
Data register
SW9
SW0


Address
31
30
. . .
4
3
2
1
0
0xFF200050
Unused
KEY3-0
Data register
Unused
Unused
0xFF200058
Unused
Mask bits
Interruptmask register
0xFF20005C
Unused
Edge bits
Edgecapture register


Address
0xFF200020
31 30 24 23 22 16 15 14 8 7 6 0
Data register
HEX36-0
HEX26-0
HEX16-0
HEX06-0
0
5
6
1
4
2
3
Segments
0xFF200030
31 30 24 23 22 16 15 14 8 7 6 0
Data register
Unused
HEX56-0
HEX46-0


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

Extra answer space for any question on the test, if needed:

Extra answer space for any question on the test, if needed:

Extra answer space for any question on the test, if needed:

Extra answer space for any question on the test, if needed: