CSS422A Winter 2022, MW115-315pm Midterm Examination Rehearsal

In the real midterm exam,

You may:	**You must not:**
Open your textbook and your notebook. Refer to eTextbook and your memo in cloud. Refer to any CSS422 Canvas materials including this rehearsal and key answers. Use VisUAL for solving Q6. Use a calculator (no number conversion)	Use any Internet search engine. Use number-converting software tools such as: https://www.rapidtables.com/convert/number/decimal-to-hex.html https://www.exploringbinary.com/floating-point-converter/ Use ARM encoding/decoding software tools: https://armconverter.com/ Use Keil uVersion.

You need to write explanations in Q1-1, Q1-2, Q1-3, Q3-1, Q3-2 and Q5-4.
You need to write assembly code in Q5-1, Q5-2, Q5-3, and Q6.
Filling out blanks is sufficient in Q1-4, Q2 and Q4.
Point distribution may be Q1:16pts, Q2:16pts, Q3:16pts, Q4:20pts, Q5:16pts, and Q6:16pts but has not yet been decided

Q1. Computer Arithmetic and NZCV flags

Q1-1. Compute $-2147483647_{(10)}$ - $2_{(10)}$ in a 32-bit system like Cortex-M. Write the answer in decimal.

$-2147483647_{(10)}$ = **-0x7FFFFFFF**
2147483647 / 16 = 134217727 R **15 (F)**
134217727 / 16 = 8388607 R **15 (F)**
8388607 / 16 = 524287 R **15 (F)**
524287 / 16 = 32767 R **15 (F)**
32767 / 16 = 2047 R **15 (F)**
2047 / 16 = 127 R **15 (F)**
127 / 16 = 7 R **15 (F)**
7/ 16 = 0 R **7**

$2_{(10)}$ = $\mathbf{00000010_{(2)}}$

Two's Complement of -0x7FFFFFFF

```
   1000 0000 0000 0000 0000 0000 0000 0000
+  0000 0000 0000 0000 0000 0000 0000 0001
-------------------------------------------
   1000 0000 0000 0000 0000 0000 0000 0001
```

0x80000001 – 0x00000002

```
   1000 0000 0000 0000 0000 0000 0000 0001
-  0000 0000 0000 0000 0000 0000 0000 0010
-------------------------------------------
   0111 1111 1111 1111 1111 1111 1111 1111
```

0x7FFFFFFF = $\mathbf{2147483647_{(10)}}$

Q1-4. When Cortex-M computed Q1-1, what values would it have in its ASPR?

APSR flags	Values (0 or 1)	Your Explanations
N	0	The value is not negative; is positive.

Z	0	The value is not zero.
C	1	Borrow occurs into the MSB on subtraction; flag is set.
V	1	Overflow occurs subtracting beyond minimum range of the 32-bit system.

Q2. Memory Endianness and Alignment
Consider the following C program. Assuming that the variable x is allocated to the memory space starting at 0x20000000 in ARM, show the memory contents of this variable's all data members such as a, b, c, and d. Note that ASCII character 'A' is 0x41.

```
struct myData {
  char a;        // 1 byte
  short b;       // 2 bytes
  int c;         // 4 bytes
  float d;       // 4 bytes
};


int main( ) {
  struct myData x = {'A', 0154321, 0xAF12E24D, -20.3125};
  return 0;
}
```

Q2-1. Convert 0154321 into hexadecimal

$0154321_{(8)}$ = 000 001 101 100 011 010 001
1101 1000 1101 0001
D 8 D 1

$0154321_{(8)}$ = 0xD8D1

Q2-2. Convert –20.3125 into a 32-bit floating point representation. Show in hexadecimal

-20.3125
$20_{(10)}$ = 10100

$0.3125 \times 2 = 0.625$
$0.625 \times 2 = 1.25$
$0.25 \times 2 = 0.5$
$0.5 \times 2 = 1.0$
$0 \times 2 = 0.0$
$10100.01010 \rightarrow 1.010001010 \times 2^4$

Sign: 1
Exponent: 127 + 4 = 131

- 1000 0011

Mantissa: 010001010

11000001101000101000000000000000
1100 0001 1010 0010 1000 0000 0000 0000
C 1 A 2 8 0 0 0

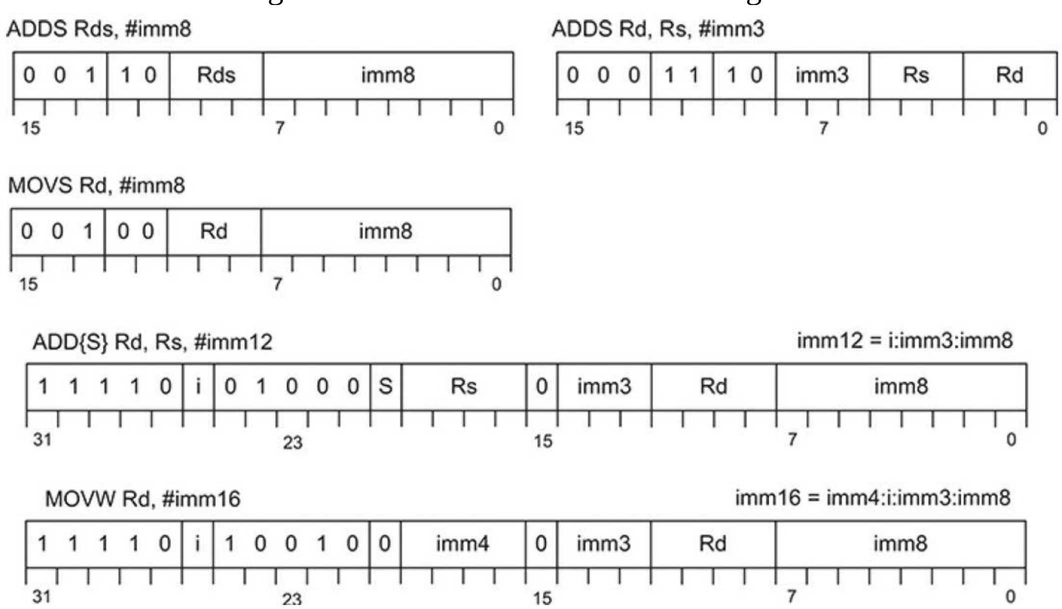
-20.3125 = C1A28000

Q2-3. Fill out the blanks of the following memory map.

Address	Contents in Hexadecimal
0x20000000	41
0x20000001	00
0x20000002	D1
0x20000003	D8
0x20000004	4D
0x20000005	E2
0x20000006	12
0x20000007	AF
0x20000008	00
0x20000009	80
0x2000000A	A2
0x2000000B	C1

Q3. Instruction Encoding and Decoding

Refer to the following ARM Thumb-2 instruction decoding chart:



Q3-1. Encode into hexadecimal:

ADDS R5, R3, #127

#127 = 0111 1111

ADD{S} Rd, Rs, #imm12: 11110 i 01000 S Rs 0 imm3 Rd imm8

- i: 0
- S: 1
- Rs: 0011

- imm3: 000
- Rd: 0101
- imm8: 01111111

11110 0 01000 1 0011 0 000 0101 01111111
1111 0001 0001 0011 0000 0101 0111 1111
F 1 1 3 0 5 7 F

ADDS R5, R3, #127 = F113057F

Q3-2. Decode:
0xF64A39CD

1111 0110 0100 1010 0011 1001 1100 1101
11110 1 10010 0 1010 0 011 1001 11001101

MOVW Rd, #imm16

- i: 1
- imm4: 1010
- imm3: 011
- Rd: 1001 (R9)
- imm8: 11001101
  - imm12 (i:imm3:imm8) = 1011 1100 1101 (0xBCD)
    - $2^0 + 2^2 + 2^3 + 2^6 + 2^7 + 2^8 + 2^9 + 2^{11} = 1 + 4 + 8 + 64 + 128 + 256 + 512 + 2048 = 3021$

0xF64A39CD = MOVW R9, #3021 *or* MOVW R9, #0xBCD

Q4. Cortex-M Memory Map

Let's assume that you're using Keil uVersion. Fill out the blanks of the memory map when running the following assembly program. No need to fill out all the blanks if they are not initialized. No need to convert instructions nor ASCII characters into hexadecimal numbers.

```
                THUMB
Stack_Size      EQU    0x00000200
                AREA   STACK, NOINIT, READWRITE, ALIGN=3
MyStackMem      SPACE  Stack_Size

                AREA   RESET, DATA, READONLY
                EXPORT __Vectors
__Vectors       DCD    MyStackMem + Stack_Size          ; Top of Stack
                DCD    Reset_Handler                    ; Reset Handler

                AREA   MYDATA, CODE, READONLY
src             DCB     "abcdefgh"
src_end         DCB     0

                ALIGN                                   ; Default Align=2
                ENTRY
Reset_Handler
```

```
EXPORT  Reset_Handler        [WEAK]
LDR     R0, =src_end
LDR     R5, =Reset_Handler
LDR     R1, [R0, -4]!
LDR     R2, [R0, -4]!
PUSH    {R1, R2, R5}
POP     {R3, R4, PC}
B       Reset_Handler
END
```

Address	Contents
	DRAM
0x60000000	Peripherals
0x40000000	SRAM
0x20000008	
0x20000200	
0x200001FC	00000015
0x200001F8	"dcba"
0x200001F4	"hgfe"
0x200001F0	
	...
0x20000000	
	...
0x00000018	LDR R1, [R0, -4]!
0x00000016	LDR R5, =Reset_Handler
0x00000014	LDR R0, =src_end
0x00000012	
0x00000010	\0
0x0000000C	"hgfe"
0x00000008	"dcba"
0x00000004	00000015
0x00000000	20000200

What value will each of R0, R1, R2, R3, R4, and R5 have? No need to convert ASCII characters into hexadecimal numbers.

Registers	Contents in Hex or ASCII characters
R0	0x0000000C
R1	"hgfe"
R2	"dcba"
R3	"hgfe"
R4	"dcba"
R5	0x00000015
PC	0x0000001C

(Yes/No) Is the instruction "B Reset_Handler" reachable, (i.e., executed)?

Q5. Data Processing Instructions

Q5-1. Write assembly code to compute R2 = R1 * 1000, using only SUB and LSL instructions.

```
LSL     R2, R1, #10                ; R2 = R1 * 1024
```

SUB R2, R2, #24 ; R2 = R2 – 24

Q5-2. Write assembly code to compute R2 = 63 % 8. May use any divide and multiply instructions.

MLS R0, #8, #7 SUB R2, #63, R0

Q5-3. Write assembly code to compute R4 = R5 - R6 / 128, using only SUB with a flexible operand 2 that includes ASR.

SUB R3, R5, R6, ASR #7

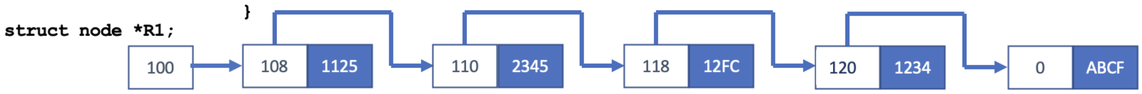
Q5-4. When R1 = 0x87654321 and R2 = 0x00000000, compute the R2 contents of SBFX R2, R1, #24, #8

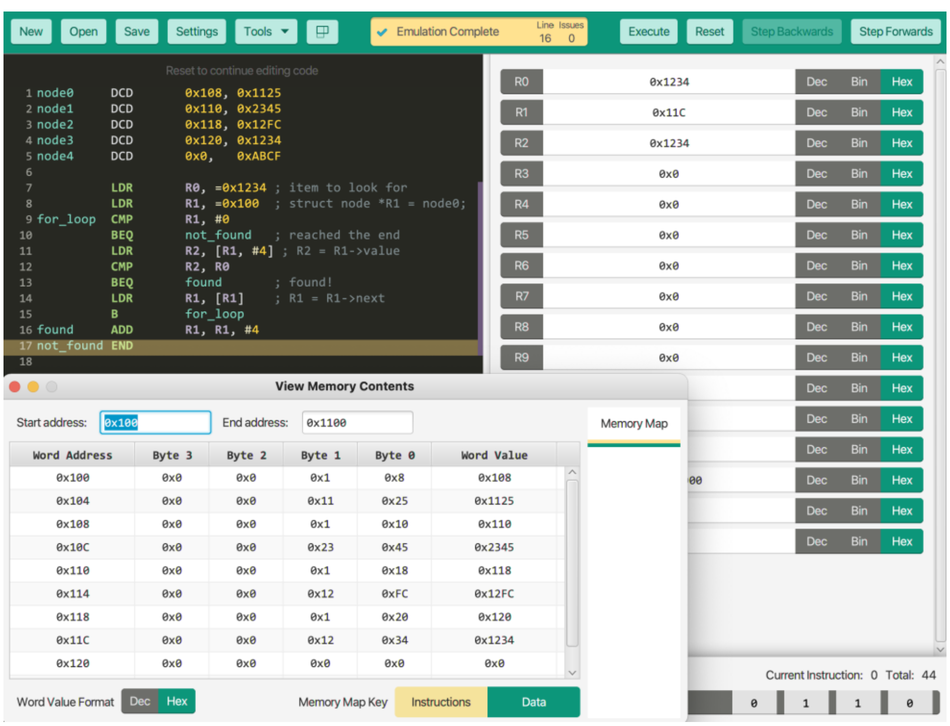
1000 0111 0110 0101 0100 0011 0010 0001 R2 = 1000 0111 (0x87) • Negative, R2 is 1-padded **R2 = 0xFFFFFF87**

Q6. Data Structure Manipulation using LDR/STR

The following code intends to travers a linked list in search for a given value in R0 and returns the address of this value into R1 (but not the address of the node). If the value was not found, it returns 0 in R1, (i.e., a null address).

```
struct node {
   struct node *next;
   int value;
}
```





Modify this code to insert a new node, named newItem with value 0xAAAA, before the node value 0x1234. Focus on the node insertion. No need to allocate a new space because the new node is defined as “**newItem DCD 0x0, 0xAAAA”** In this program, you don’t have to keep the address of value 0x1234 in R0.

```
node0  DCD           0x108, 0x1125
node1  DCD           0x110, 0x2345
node2  DCD           0x118, 0x12FC
node3  DCD           0x120, 0x1234
node4  DCD           0x0,   0xABCF
newItem DCD          0x0,   0xAAAA

             LDR            R0, =0x1234 ; item to find
             LDR            R1, =0x100  ; struct node *R1 = node0;
             LDR            R2, =0xAAAA ; item to add

for_loop     CMP            #0
             BEQ            not_found
             CMP            R0
             BEQ            found
found        STR            R0, [R2]
not_found
            END
```