**ECSE 324 Winter 2022 Midterm**

**Part A: Choose wisely**

Part A.1: *Multiple select*. Choose as many options are as appropriate for each question. *Select* **I don’t know** *for 1/4 credit.*

1. Which of the following bits or signals are needed to determine whether overflow has occurred when performing ADD R0, R1, R2?
   1. Most significant bit of R0
   2. Most significant bit of R1
   3. Most significant bit of R2
   4. Carry out
   5. I don’t know
2. Which of the following addresses are appropriate for half-word aligned memory accesses?
   1. 0x0DDBA115
   2. 0xB01DFACE
   3. 0xDEADBEA7
   4. 0xDABADABA
   5. I don’t know
3. Which sequences of instructions are equivalent to LDR R2, [R0, R1, LSL#3]
   1. MOV R3, #8  
      MUL R3, R1, R3  
      ADD R4, R0, R3  
      LDR R2, [R4]
   2. MOV R3, #3

MUL R3, R1, R3  
ADD R4, R0, R3  
LDR R2, [R4]

* 1. LSL R3, R1, #3  
     ADD R4, R0, R3  
     LDR R2, [R4]
  2. MOV R3, #8  
     MUL R3, R1, R3  
     LDR R2, [R0, R3]
  3. I don’t know

1. When is the offset for the BGT instruction below calculated?
   1. Compilation
   2. Assembler pass #1
   3. Assembler pass #2
   4. Linking
   5. I don’t know

dotpLoop:   
 **LDR**  V2, [A1], #4 // get vectorA[i] andpost-increment  
 **LDR** V3, [A2], #4 // get vectorB[i] andpost-increment  
 **MLA** V1, V2, V3, V1 // V1 += V2\*V3  
 **SUBS** A3, A3, #1 // i-- andset condition flags  
 **BGT** dotpLoop

1. Which of the following instructions set Z=1 in the CPSR, given R0=0xBEEFCAFE and R1=0x41103502?
   1. TST R0, #0xFC000
   2. TST R1, #0xFC000
   3. SUB R2, R0, R0
   4. ADDS R2, R0, R1
   5. I don’t know

Part A.2: *Matching*. Select the appropriate option for each part of each question, or *match any option to* **I don’t know** *for 1/4 credit.*

1. What is the value in each register after the following code is executed? (i) 0x4000, (ii) 0x4010, (iii) 0x1FF00, (iv) 0x1FF04.
   1. R0
   2. R1

0x1FF00 kbd: .word 0x00004000 // keyboard base address

0x1FF04 disp: .word 0x00004010 // display base address

0x1FF08 \_start:

LDR R0, kbd

0x1FF0C LDR R1, =disp

1. An interrupt service routine uses A2 and V1 and no other registers. When will each of the following registers be pushed onto the stack? (i) Before the ISR begins, during context saving initiated by interrupt hardware; (ii) During the ISR, in accordance with the ARM APCS; (iii) Never, because it is not necessary.
   1. PC
   2. A2
   3. V1
   4. R5

Part A.3: *Short answer*. *Write* **I don’t know** *for 1/4 credit.*

1. Assume a 32-bit *big endian* RISC computer, and memory contents defined as follows. If

* R0 = 0x0000 0004,
* R1 = 0x0000 0002
* R2 = 0x0010 0010

what is in R2 after: LDRSH R2, [R0, R1]! ? Give your answer in hexadecimal.

1. How many times is memory accessed during a single iteration of the following loop (the five instructions from dotpLoop to BGT, inclusive)?

dotpLoop:   
 LDR V2, [A1], #4 // get vectorA[i] andpost-increment  
 LDR V3, [A2], #4 // get vectorB[i] andpost-increment  
 MLAV1, V2, V3, V1 // V1 += V2\*V3  
 SUBSA3, A3, #1 // i-- andset condition flags  
 BGTdotpLoop

1. Why isn’t there a store instruction equivalent to LDRSH?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Addr** | **Data** |  | **Addr** | **Data** |
| 0x00 | 0xD1 |  | 0x08 | 0x78 |
| 0x01 | 0x4B |  | 0x09 | 0x91 |
| 0x02 | 0x45 |  | 0x0A | 0x03 |
| 0x03 | 0xC4 |  | 0x0B | 0x70 |
| 0x04 | 0x90 |  | 0x0C | 0xB3 |
| 0x05 | 0x12 |  | 0x0D | 0xDA |
| 0x06 | 0x4F |  | 0x0E | 0x7F |
| 0x07 | 0xEE |  | 0x0F | 0xE6 |

**Part B: It’s a bit RISC-y**

Assume an 8-bit RISC CPU with four general-purpose registers R0-R3, working scratch register WS, program counter PC, and current program status register CPSR.

The only available instructions are:

* LD Rm, Rs // WS <- Mem[Rm+Rs]
* ST Rm, Rs // Mem[Rm+Rs] <- WS
* MVW Rm // WS <- Rm
* MWV Rd // Rd <- WS
* MVI #Imm // WS <- #Imm
* ADD Rm // WS <- WS + Rm
* CMP Rm // If (WS – Rm == 0), set Z flag to 1
* B displacement // PC <- PC + displacement

Assume that when instruction *i* at address 0x*n* is executing, PC points to 0x*n*+1.

Each instruction can be conditionally executed (when Z==1), based on a 1-bit condition field in each instruction, indicated by adding the EQ suffix to an instruction.

*Select or write* **I don’t know** *below for 1/4 credit.*

What is the minimum number of bits required to encode a register operand?

* 1 bit
* 2 bits
* 3 bits
* 4 bits
* None of these
* I don’t know

What is the maximum number of bits available to encode the immediate value for MVI?

* 2 bits
* 3 bits
* 4 bits
* 5 bits
* None of these
* I don’t know

How many instructions can be stored in the memory addressable by this CPU?

* 32
* 64
* 128
* 256
* None of these
* I don’t know

Suppose we want to implement a pseudo-instruction, DBL Rd, which implements Rd <- WS\*2. Write a sequence of instructions to implement this pseudo-instruction.

Attempting to assemble the following code will result in an error. What is the problem?

Loop:

CMP R0

BEQ Done

MWV R1

MVI #1

ADD R1

MWV R1

LD R2, R1

ST R3, R1

MVW R1

B Loop

Done:

B Done

**Part C: They’re multiplying!**

Fill in the blanks below to complete the following ARMv7 assembly to implement a function that performs matrix-vector multiplication using a help function that performs vector-vector multiplication. *Write* **I don’t know** *for 1/4 credit.*

Assume the following C function prototypes:

// takes pointers to two vectors of size *length*, and saves their dot-  
// product at the address pointed to by *result*

void vvm(char\* vectorA, char\* vectorB, char\* result, int length);

// takes pointers to a square matrix and a vector of size *length*, and   
// saves their product starting at the address pointed to by *result*

void mmm(char\* vectorA, char\* vectorB, char\* result, int length);

Further assume:

* Functions must respect the ARM APCS.

// void vvm(char \*vectorA, char \*vectorB, char \*result, int length)

vvm:

PUSH {V1-V4}

MOV V1, #0

MOV V2, #0

vvmLoop:

CMP V1, A4

BEQ vvmRet

\_\_\_ V3, [A1], #1 // (1)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ // (2)

MLA V2, V3, V4, V2

ADD V1, V1, #1

\_\_\_\_\_\_\_\_\_\_\_\_\_ // (3)

vvmRet:

STRB V2, [A3], #1

POP \_\_\_\_\_\_ // (4)

\_\_\_ LR // (5)

// void mmm(char \*matrix, char \*vector, char \*result, int length)

mmm:

PUSH \_\_\_\_\_\_ // (6)

MOV V1, #0

mmmLoop:

\_\_\_ V1, A4 // (7)

\_\_\_ mmmRet // (8)

\_\_\_ vvm // (9)

ADD V1, V1, #1

SUB A2, A2, \_\_ // (10)

B mmmLoop

mmmRet:

POP {V1, PC}