**ECSE 324 Winter 2022 Final**

**Part A: Choose Wisely (34 pts)**

2pts each unless otherwise noted. These questions are a mix of matching (MA), multiple choice (MC), multiple select (MS), short answer (SA), and long answer (WR). Select or answer “I don’t know” for 1/4 credit.

*ISA*

1. Assume a 32-bit *little endian* RISC computer, and memory contents defined in the table below. If
   * R0 = 0x0000 0008
   * R1 = 0x0000 0002
   * R2 = 0x0000 0010

What is in R2 after: LDRSH R2, [R0], R1? Give your answer in hexadecimal, e.g., 0x0123 4567. (SA)

**0xFFFF 9178** – 1 pt for current address and bytes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

1. Which of following instruction sequences are equivalent to: POP {V1, LR}? (MS)
   * **LDMIA R13!, {R14, R4}**
   * **ADD SP, SP, #-4  
     LDMIB SP, {V1, LR}  
     ADD SP, SP, #12**
   * **LDR V1, [SP], #4  
     LDR LR, [SP], #4**
   * LDR V1, [SP]  
     LDR LR, [SP, #4]
2. How many times is memory accessed during the following code sequence? Assume that A1 contains 0xBEEF CAFE. (SA)

TST A1, #2

LDREQ V1, [A2], #4

LDRNE V2, [A2], #4

LDRLE V3, [A2], #4

LDRGE V4, [A2], #4

**7** – 1 pt for 6, 8, 9

*Input/Output*

1. Which of the following are true of (i) subroutines, (ii) interrupt service routines, (iii) both, or (iv) neither? (MA)
   * It must push LR if it calls a subroutine. **(iii)**
   * PC is pushed on the stack before it begins to execute. **(ii)**
   * It is called with the BL instruction. **(i)**
   * Use of LDM is prohibited. **(iv)**
2. Which of the following are most true of (i) serial communication, (ii) parallel communication, true of (iii) both, or true of (iv) neither? (MA)
   * Relatively less vulnerable to transceiver failure **(i)**
   * Relatively slower communication, due to e.g., electrical interference **(ii)**
   * Relatively less complex protocols **(ii)**
   * Requires a clock **(iv)**

*Memory*

1. Assume a 32-bit RISC CPU's 256 MB, byte-addressable, main memory is implemented using four (4) byte-addressable 64Mx8 asynchronous DRAM chips. I.e., the four (4) chips are interfaced and interconnected in such a way that only one is active for any given memory access. Each chip is structured such that their DRAM array rows are 32K bits wide. How many bits wide must the address port on the individual DRAM chips be? (MC)
   1. 10
   2. 12
   3. 14
   4. 16
   5. **None of the above (13)**
2. Assume you have a 32 KB cache consisting of 64 B cache blocks. How many comparators are needed if the cache is (a) 4-way set associative, or (b) fully associative? (SA)
   1. **4**
   2. **512**
3. Assume you have a 32 KB cache consisting of 64 B cache blocks, and 128 MB of main memory. If the cache is 8-way set associative, how many blocks in memory map to each set in cache? (MC)
   1. 214
   2. **215**
   3. 216
   4. 217
   5. None of the above
4. Select the steps that need to be taken in a direct-mapped write-back cache in the following situation. Memory block A (unmodified) is in set N. A store instruction accesses memory block B, not in cache, which also maps to set N.
   * Copy block A from cache to memory
   * **Copy block B from memory to cache**
   * **Update block B in cache**
   * Update block B in memory
5. The translation-lookaside-buffer (TLB) used to support page number translation in the memory management unit is one of the few places we find fully- associative memories in modern computer architecture. Why? (MS)
   * FA caches are fast, and virtual address translation must be as fast as possible.
   * **FA caches have low miss rates, and virtual address translation must miss as little as possible.**
   * **FA caches are slow, but this is okay because virtual address translation takes a long time, too.**
   * FA caches have high miss rates, but this is okay because page table lookup is fast.
6. Assume a 32-bit computer system with 1 GB of RAM. If pages are 4 KB each, and each page table entry requires 16 bits to track the status of the entry (valid, dirty, accessed, permissions, etc), how much memory is required for a complete page table? Give your answer in KB.

**5120** – 1 pt for 1280, 4352, 4456, or 5243

*Processor*

1. *4pts* Consider the following data path. Which of the following ARMv7 instructions would be supported? (MS)

Diagram, engineering drawing

Description automatically generated

* + **ADD R2, R3, #8**
  + MLA R1, R2, R3, R4
  + **TEQ A3, #0x12**
  + PUSH {V1, V2}
  + **PUSH {LR}**
  + POP {A1}
  + **LDR R1, [R2, R3]**
  + STR V2, [A1, A2]

1. Consider once again the example data path above. What is the contents of RB after the instruction ADD R1, R2, #4 is executed? Assume that R1 = 0xBEEF FEED and R2 = 0xFACE CAFE. Justify your answer for credit. (WR)

**RB will contain 0xBEEF FEED. Address B uses the same bits in the instruction that are used for Address C in instructions that use the immediate-operand format.**

1. The following design strategies can be used to reduce pipeline hazards. Match the strategy to the type of hazard that it best addresses: (i) Control hazard, (ii) Data hazard, (iii) Structural hazard due to contention for memory. (MA)
   1. Using conditional execution **(i)**
   2. Implementing split L1 caches **(iii)**
   3. Implementing forwarding **(ii)**
   4. Re-ordering instructions during assembly **(ii)**

*Arithmetic*

1. There are approximately 2*n* 32-bit floating point numbers between 1 and 4 when using the IEEE 754 standard. What is *n*?

**24** – 1 pt for 23 or 25

1. The IEEE 754 standard also includes a specification for 16-bit half precision floating-point numbers. This representation includes a sign bit, five bits of exponent, and 10 bits of mantissa. Like FP32, this representation is normalized, and uses a biased exponent; the bias in this case is 15. What is -2.375 in this representation? Give your answer in hexadecimal, e.g., 0xa1b2.

**0xC0C0**

**Part B: Some assembly required (16 pts)**

Complete the following assembly to implement a program that reads ASCII characters pressed on a keyboard and executes the JUMP subroutine every time the spacebar (ASCII code 32) is pressed.

The keyboard operates using interrupts. The SERVICE\_IRQ ISR is executed first when any interrupt occurs; there are no other IRQ besides \_reset in this simplified implementation. SERVICE\_IRQ calls KBD\_ISR when the keyboard needs to be serviced. Assume that SERVICE\_IRQ and JUMP are implemented elsewhere.

The keys pressed on the keyboard are stored in a circular buffer that can hold eight (8) characters (bytes). Characters are enqueued (stored at the tail of the queue) with a call to CENQ8, and dequeued (loaded from the head of the queue) with a call to CDEQ8. These functions are provided for you.

CENQ8:

push {v1}

ldrb v1, [a2] // load the tail index

strb a3, [a1, v1] // store the character at the tail

add v1, v1, #1 // increment the tail

cmp v1, #8 // check if we need to wrap

blt CENQ8\_ret // return if 7 or less

mov v1, #0 // reset tail index

CENQ8\_ret:

strb v1, [a2] // store new tail index value

pop {v1}

bx lr // return

CDEQ8:

push {v1}

ldrb v1, [a2] // load the head index

ldrb a1, [a1, v1] // load the character at the head

add v1, v1, #1 // increment the head

cmp v1, #8 // check if we need to wrap

blt CDEQ8\_ret // return if 7 or less

mov v1, #0 // reset head index

CDEQ8\_ret:

strb v1, [a2] // store new head index value

pop {v1}

bx lr

Complete the code that follows such that KBD\_ISR enqueues characters in the buffer, \_start dequeues characters in the buffer, compares them with the appropriate ASCII code, and calls JUMP as needed.

.section .vectors, "ax"

B \_start

B **SERVICE\_IRQ** // 1 1pt

.text

.global \_start

.equ dkbd, 0x4000

.equ ksp, **32** // 2 1pt

cbuf: .space 8 // circular buffer

cbufh: .byte 0 // head index

cbuft: .byte 0 // tail index

.space 2

KBD\_ISR:

push {lr}

ldr a1, **=cbuf** // 3 1pt

ldr a2, **=cbuft** // 4 1pt

ldr a3, **=dkbd** // 5 1pt

ldrb a3, [a3]

bl CENQ8

pop {lr}

bx lr

\_start:

ldr a1, =cbuf

ldr a2, =cbufh

ldr a3, =cbuft

wait:

**ldrb** v1, [a2]// 6 1pt

ldrb v2, [a3]

cmp **v1, v2** // 7 2pt

beq **wait** // 8 2pt

**bl** CDEQ8 // 9 1pt

cmp a1, #ksp

**bleq** JUMP// 10 2pts

ldr a1, **=cbuf** // 11 2pts

b **wait** // 12 1pt

**Part C: There are exactly two ways to cache a block (10 pts)**

Assume a 16-bit address space and byte-addressable, *big-endian*, main memory. The table below illustrates the state of a 16-set 2-way set associative cache. Each set consists of two blocks of 8B (with the least significant byte on the right). The state (*V* for Valid, and Tag) of each block in each set is indicated.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Set** | **V** | **Tag** | **Data** | **V** | **Tag** | **Data** |
| 7 6 5 4 3 2 1 0 |  |  | 7 6 5 4 3 2 1 0 |
| 0 | 0 | 134 | 83 ab 57 88 31 e7 a0 5e | 1 | 05f | 8f 4c e4 df 63 77 c6 5e |
| 1 | 1 | 01f | d2 ce aa 7c 82 23 d7 d7 | 1 | 1e6 | 1f 64 76 c7 a3 dc 19 78 |
| 2 | 1 | 0d3 | e6 65 8d 3b e3 53 f8 d7 | 0 | 09a | f8 40 3d f2 80 42 0a f0 |
| 3 | 1 | 02d | 0c b8 00 e4 d7 60 58 b0 | 0 | 087 | 71 4e 10 f7 b9 5f 58 c7 |
| 4 | 0 | 167 | 5f 91 85 64 de 45 ca e8 | 0 | 034 | 84 b7 c0 41 6e 5c d4 64 |
| 5 | 1 | 1d3 | be 8a 35 b0 9b 75 70 48 | 1 | 06c | 08 06 a0 bd 13 c6 69 a0 |
| 6 | 1 | 022 | 2a a4 7b bd 72 a1 f7 51 | 1 | 019 | 19 3b 55 fc d8 52 30 35 |
| 7 | 1 | 000 | 30 4b 12 c0 2f 51 8d 3a | 0 | 035 | 8e 57 c7 8b be 44 93 a2 |
| 8 | 0 | 1ab | df cd 5a f7 7f 4b cd b3 | 1 | 100 | 58 a5 4d 46 f5 30 91 e6 |
| 9 | 0 | 112 | ee f9 86 7d 24 dd 98 7c | 1 | 171 | 8c a7 de ec bc a5 06 cb |
| a | 1 | 0e3 | 2c 5b af c6 92 09 cf ca | 0 | 0f6 | a9 e5 f2 87 d2 91 9b 1d |
| b | 0 | 0df | e8 af 5b 2e 4a 77 e0 0c | 0 | 024 | 1d d0 ce 0b b8 ac a6 9c |
| c | 0 | 0ee | 9d b7 11 e9 68 7c a3 4f | 0 | 1c6 | 91 17 2f b3 3d 85 92 33 |
| d | 1 | 01d | cd 8b 2f 5d c8 de 16 90 | 0 | 0aa | d2 4e e3 20 12 83 f2 62 |
| e | 1 | 0a0 | d5 d0 a3 4b 9e 1a d4 55 | 1 | 1f0 | a3 9b 1d 98 26 38 a5 e0 |
| f | 0 | 155 | a5 65 2b 9f 55 20 72 e6 | 1 | 1be | 25 23 07 e1 bf 7f 72 1e |

2 pts each unless otherwise noted.

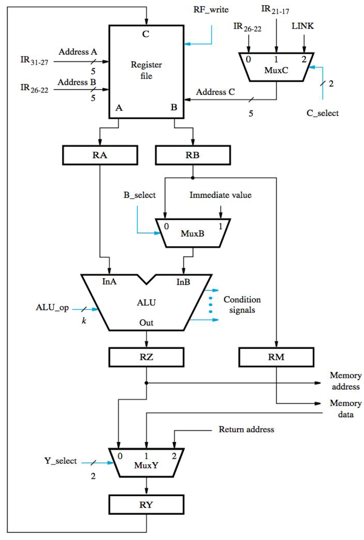
1. Does an access to address 0x5074 hit or miss? (**Hit**/Miss)
2. Does an access to address 0x4d10 hit or miss? (Hit/**Miss**)
3. What 4B word is returned by a load from 0xB8C8? (SA) **0xCB06 A5BC**
4. *4pts* Calculate the hit or miss status for each following sequence of memory accesses. Give your answer in decimal, e.g., 0.8. (WR)

0x169C, 0x169E, 0x16A0, 0x16A2, 0x16A4, 0xDB9A, 0xDB9C, 0x169C

**The first two accesses hit in set 3 (2/2). The next access, to set 4, misses (2/3). One of the two invalid blocks is replaced. The following two, however, are to the same block, and hit (4/5). The access to 0xDB9A misses, and the invalid block in set 3 is replaced; it is followed by a hit (5/7). The final access to a previously accessed block hits because this block was not replaced (6/8 = 0.75).**

**Part D: Timing is Everything (12 pts)**

Consider the following 5-stage RISC CPU datapath.



Use the following sequence of assembly instructions to answer the following questions.

ADD R2, R2, #4

LDR R1, [R0, #8]

STR R1, [R2]

1. *2pts* How many cycles does this sequence of instructions take to execute without pipelining? Assume that every memory access completes in a single cycle. (SA)

**15**

**5 cycles for each instruction; no overlapped execution.**

1. *4pts* When the ADD instruction is executing, what are the values of the following control signals? Give your answers in binary, e.g., 0b0110. (SA)
   1. B\_Select **(0b1)**
   2. Address A **(0b00010)**
   3. C\_Select **(0b00)**
   4. Address C **(0b00010)**
2. *3pts* How many cycles does this sequence of instructions require with pipelining, but without forwarding? Assume that every memory access completes in a single cycle, but that only a single memory access may be performed in each cycle. Give priority to instructions in the memory stage. Hazards are detected in the decode stage. (SA)

**10**

**1 2 3 4 5 6 7 8 9 0**

**ADD F D E M W**

**LDR F D E M W**

**STR F D D D D E M W**

1. *3pts* How many cycles does this sequence of instructions require with pipelining and forwarding? Assume that every memory access completes in a single cycle, but that only a single memory access may be performed in each cycle. Give priority to instructions in the memory stage. Hazards are detected in the decode stage. Forwarding is available from Memory to Execute, Write-back to Memory, and Write-back to Execute. (WR)

**7**

**1 2 3 4 5 6 7**

**ADD F D E M W**

**LDR F D E M W**

**STR F D E M W**

**Two paths are needed: Write-back to Execute (ADD->STR) to resolve the dependency on R2 for address calculation, and Write-back to Memory   
(LDR->STR) to resolve the dependency on R1 for the memory access itself.**

**Part Z: Bonus!**

1. *1pt* Write a haiku or other short poem about ECSE 324, Computer Organization. (WR)
2. *1pt* Do you grant Professor Meyer permission to share your poem (at his discretion) on Twitter @bretthmeyer? (Yes/No)