CSC 111 — Spring 2026 Final Exam

Part I — Multiple Choice (1 point each)

Question 1 (Multiple Choice, 1 point)

For unsigned subtraction, when is the Carry flag (C) set to 0?

- ☐ A. When no borrow occurs
- ☑ **B. When a borrow occurs**
- ☐ C. When the result is negative
- ☐ D. When overflow occurs

Answer: B. When a borrow occurs

Question 2 (Multiple Choice, 1 point)

What ARM condition code suffix is used for a branch if the Zero flag is set after a comparison?

- ☐ A. NE
- ☐ B. GT
- ☑ **C. EQ**
- ☐ D. AL

Answer: C. EQ

Question 3 (Multiple Choice, 1 point)

The TST instruction performs which operation?

- ☐ A. R1 + R2
- ☐ B. R1 - R2
- ☑ **C. R1 & R2 (bitwise AND)**
- ☐ D. R1 ^ R2 (bitwise XOR)

Answer: C. R1 & R2 (bitwise AND)

Question 4 (Multiple Choice, 1 point)

What does the ARM instruction TST r1, #1 check?

- ☑ **A. If the least significant bit of r1 is set**
- ☐ B. If the least significant bit of r1 is cleared
- ☐ C. If the most significant bit of r1 is set
- ☐ D. If the most significant bit of r1 is cleared

Answer: A. If the least significant bit of r1 is set

Question 5 (Multiple Choice, 1 point)

Which instruction is used to return control to the caller from a subroutine?

- ☐ A. BL LR
- ☑ **B. BX LR**
- ☐ C. BL PC

☐ D. BX PC

Answer: B. BX LR

Question 6 (Multiple Choice, 1 point)

When the BL (Branch and Link) instruction is executed, what two operations occur?

☐ A. PC is set to target address; LR is incremented by 1

☐ B. LR holds the target address; PC is incremented by 2

☑ **C. LR = PC + 4 (return address); PC = target address**

☐ D. SP is decremented; PC is set to target address

Answer: C. LR = PC + 4 (return address); PC = target address

Question 7 (Multiple Choice, 1 point)

In a for loop implementation using 'SUBS r1, r1, #1' followed by 'BNE loop', what does this accomplish?

☐ A. Increments counter and loops if zero

☑ **B. Decrements counter and loops if not zero**

☐ C. Sets flags without changing register

☐ D. Unconditional branch

Answer: B. Decrements counter and loops if not zero

Question 8 (Multiple Choice, 1 point)

According to the ARM EABI, how is a 64-bit argument (such as a long int) passed to a subroutine?

☐ A. In a single 64-bit register

☐ B. Split across R0 and R2

☑ **C. In two consecutive 32-bit registers (e.g., R0:R1 or R2:R3)**

☐ D. Always passed on the stack

Answer: C. In two consecutive 32-bit registers (e.g., R0:R1 or R2:R3)

Question 9 (Multiple Choice, 1 point)

Where is a 32-bit return value placed by a subroutine?

☑ **A. In register R0**

☐ B. In register R7

☐ C. On the stack

☐ D. In the link register

Answer: A. In register R0

Question 10 (Multiple Choice, 1 point)

When you execute PUSH {R2, R1, R3}, in what memory address order are the registers stored in memory, from lowest to highest address?

☐ A. R2, R1, R3

☑ **B. R1, R2, R3**

☐ C. R3, R2, R1

☐ D. No specific order; implementation-dependent

Answer: B. R1, R2, R3

Question 11 (Multiple Choice, 1 point)

On a PUSH operation in Cortex-M (full descending stack), when is SP decremented?

☐ A. After storing each register

☐ B. Only once at the end

☑ **C. Before storing the first register**

☐ D. SP is incremented, not decremented

Answer: C. Before storing the first register

Question 12 (Multiple Choice, 1 point)

A timer runs from a 72 MHz clock. PSC = 71 and ARR = 999. In PWM Mode 1, CCR = 250. What is the duty cycle?

☐ A. 10%

☑ **B. 25%**

☐ C. 50%

☐ D. 75%

Answer: B. 25%

Question 13 (Multiple Choice, 1 point • ADD LSL)

What is the effect of the following instruction ADD r4, r4, r4, LSL #2?

☐ A. Multiply R4 by 2

☐ B. Multiply R4 by 3

☐ C. Multiply R4 by 4

☑ **D. Multiply R4 by 5**

Answer: D. Multiply R4 by 5

Question 14 (Multiple Choice, 1 point • ADD LSL)

What is the effect of the following instruction RSB r4, r4, r4, LSL #2?

☐ A. Multiply R4 by 2

☑ **B. Multiply R4 by 3**

☐ C. Multiply R4 by 4

☐ D. Multiply R4 by 5

Answer: B. Multiply R4 by 3

Question 15 (Multiple Choice, 1 point • ADD LSL)

What is the effect of the following instruction: SUB r4, r4, r4?

☐ A. Multiply R4 by 2

☐ B. Multiply R4 by 3

☑ **C. Set R4 to 0**

☐ D. Set R4 to 1

Answer: C. Set R4 to 0

Question 16 (Multiple Choice, 1 point • ADD LSL)

What is the effect of the following instruction: ADD r4, r4, r4?

☑ **A. Multiply R4 by 2**

☐ B. Multiply R4 by 3

☐ C. Set R4 to 0

☐ D. Set R4 to 1

Answer: A. Multiply R4 by 2

Question 17 (Multiple Choice, 1 point • MOV)

What is r2's value after executing the following instructions sequentially?

```
MOV r2, #0
MOVW r2, #0x1234
MOVT r2, #0xABCD
```

☑ **A. 0xABCD1234**

☐ B. 0x1234ABCD

☐ C. 0xABCD0000

☐ D. 0x00001234

Answer: A. 0xABCD1234

Question 18 (Multiple Choice, 1 point • MOV)

What is r2's value after executing the following instructions sequentially?

```
MOV r2, #0
MOVT r2, #0xABCD
MOVW r2, #0x1234
```

☐ A. 0xABCD1234

☐ B. 0x1234ABCD

☐ C. 0xABCD0000

☑ **D. 0x00001234**

Answer: D. 0x00001234

Question 19 (Multiple Choice, 1 point)

In PWM Mode 1 (Up-counting), if CNT < CCR, the output is:

☑ **A. Active (High)**

☐ B. Inactive (Low)

☐ C. Toggled

☐ D. Undefined

Answer: A. Active (High)

Question 20 (Multiple Choice, 1 point)

To increase the Duty Cycle in PWM Mode 1, you should:

- ☐ A. Decrease the CCR value.
- ☑ **B. Increase the CCR value.**
- ☐ C. Decrease the Prescaler (PSC).
- ☐ D. Increase the Prescaler (PSC).

Answer: B. Increase the CCR value.

Part II — Fill in the Blank

Question 21 (Fill in the Blank, 5 points • Calculate Register Values w/ Shifts I)

Find the register values (in hex) after finishing execution of the following program (sequentially, not individually).

```
MOV r0, 0x11
MOV r1, r0, LSL #1
MOV r2, r1, LSL #1
MOV r3, r1, LSR #1
ADD r4, r1, r1, LSL #2
```

Answers:

r0 = **0x11**
r1 = **0x22**
r2 = **0x44**
r3 = **0x11**
r4 = **0xAA**

Explanation: r0 is loaded with 0x11 (decimal 17). r1 = r0 << 1 = 0x11 × 2 = 0x22. r2 = r1 << 1 = 0x22 × 2 = 0x44. r3 = r1 >> 1 (logical) = 0x22 / 2 = 0x11. For r4, the second operand is r1 shifted left by 2 (r1 × 4 = 0x88), so r4 = r1 + (r1 << 2) = r1 × 5 = 0x22 × 5 = 0xAA. Key idea: a register operand with LSL #n in the same instruction multiplies that operand by 2^n at no extra cycle cost.

Question 22 (Fill in the Blank, 5 points • Calculate Register Values w/ Shifts II)

Find the register values after finishing execution of the following program (sequentially, not individually). Tip: CMP R0, R1 sets the Carry flag based on R0 − R1. Encode a negative number using two's complement.

```
MOV R0, #0x1234
MOV R1, #0x1111
ADD R2, R0, R1
SUB R3, R0, R1
RSB R4, R0, R1
CMP R0, R1
ADC R5, R0, R1
ADC R6, R0, #0x11
```

Answers:

R2 = **0x2345**
R3 = **0x0123**
R4 = **0xFFFFFEDD**
R5 = **0x2346**
R6 = **0x1246**

Explanation: R2 = R0 + R1 = 0x1234 + 0x1111 = 0x2345. R3 = R0 − R1 = 0x1234 − 0x1111 = 0x0123. RSB computes (operand2 − operand1): R4 = R1 − R0 = 0x1111 − 0x1234 = −0x123 = 0xFFFFFEDD in 32-bit two's complement. CMP R0, R1 performs R0 − R1 = 0x1234 − 0x1111, which produces no borrow, so the Carry flag = 1. ADC adds operands plus C: R5 = R0 + R1 + 1 = 0x2346, and R6 = R0 + 0x11 + 1 = 0x1234 + 0x12 = 0x1246. Key idea: on ARM, subtraction sets C = 1 when no borrow occurs, the opposite of the more familiar "borrow → carry" convention.

Question 23 (Fill in the Blank, 5 points • Shifts and Rotation)

Given R0 = 0x12345678, write the destination register value after executing each instruction (individually, not sequentially).

```
MOV R1, R0, ROR #8
```
R1 = **0x78123456**
```
MOV R2, R0, ROR #16
```
R2 = **0x56781234**
```
MOV R3, R0, ROR #24
```
R3 = **0x34567812**
```
MOV R7, R0, ASR #24
```
R7 = **0x00000012**
```
MOV R8, R0, ASR #31
```
R8 = **0x00000000**

Explanation: ROR rotates the 32-bit value right; bits that fall off the low end re-enter at the top.

Rotating 0x12345678 by 8 brings the low byte 0x78 to the top → 0x78123456.

ROR #16 swaps the halfwords → 0x56781234.

ROR #24 is equivalent to a left rotate of 8 → 0x34567812.

ASR is arithmetic shift right: it preserves the sign bit. Because the MSB of 0x12345678 is 0 (positive), ASR shifts in zeros. ASR #24 leaves only the top byte 0x12 at the bottom → 0x00000012.

ASR #31 shifts almost everything out, leaving the (zero) sign bit replicated → 0x00000000. Key idea: ROR preserves all bits, ASR loses them but preserves sign.

Question 24 (Fill in the Blank, 10 points • Binary Arithmetic and NZCV Flags)

Assume a 4-bit system. For each operation, compute the result and NZCV flags. The first row is given.

Operation	Result (binary)	Unsigned decimal	Signed decimal	NZCV
0100 + 0101	1001	4 + 5 = 9	4 + 5 = −7 (true = 9)	1001
0011 − 0101	1110	3 − 5 = 14 (true = −2 as unsigned not representable)	3 − 5 = −2	1000
1111 + 1111	1110	15 + 15 = 14 (true = 30)	−1 + (−1) = −2	1010
0000 − 1111	0001	0 − 15 = 1 (true = −15)	0 − (−1) = 1	0000
0100 − 1001	1011	4 − 9 = 11 (true = −5 as unsigned not representable)	4 − (−7) = −5 (true = 11)	1001
1110 + 0011	0001	14 + 3 = 1 (true = 17)	−2 + 3 = 1	0010

Explanation: In a 4-bit machine, NZCV flags are computed from the raw 4-bit result. N (Negative) = MSB of the result. Z (Zero) = 1 if all bits are zero. For addition, C = carry out of bit 3; for subtraction (ARM convention), C = 1 if no borrow (i.e., minuend ≥ subtrahend as unsigned). V (oVerflow) = 1 when the signed result cannot fit in 4

bits — practically, when two same-sign operands produce a result of the opposite sign in addition, or when minuend and subtrahend have different signs and the result's sign differs from the minuend in subtraction. Walking row 3 (1111 + 1111): unsigned 15 + 15 = 30 = 11110_2 → low 4 bits 1110, C = 1 (carry out). Signed −1 + (−1) = −2 = 1110, no signed overflow (V = 0). N = 1, Z = 0 → NZCV = 1010. Row 5 (0100 − 1001): unsigned 4 − 9 underflows (borrow), so C = 0. Signed 4 − (−7) = 11, which doesn't fit in a signed 4-bit range [−8, 7] → V = 1. Result bits 1011 → N = 1, Z = 0 → NZCV = 1001.

Question 25 (Fill in the Blank, 6 points • Swap register values)

Write a program to swap the contents of R0 and R1 using EOR (Exclusive OR). Hint: do it in three steps:

Step 1: Compute R0 ⊕ R1. Step 2: Compute (R0 ⊕ R1) ⊕ R1 = R0. Step 3: Compute (R0 ⊕ R1) ⊕ R0 = R1.

Answer:

```
Line 1:  EOR R0, R0, R1
Line 2:  EOR R1, R0, R1
Line 3:  EOR R0, R0, R1
```

Explanation: XOR has the property that x ⊕ x = 0 and x ⊕ 0 = x, so it is its own inverse. Let a, b be the original values. After Line 1, R0 = a ⊕ b (R1 still = b). Line 2: R1 = R0 ⊕ R1 = (a ⊕ b) ⊕ b = a — R1 now holds the original R0. Line 3: R0 = R0 ⊕ R1 = (a ⊕ b) ⊕ a = b — R0 now holds the original R1. The values are swapped using no temporary register and no memory. This is sometimes called the XOR-swap trick.

Question 26 (Fill in the Blank, 15 points • C to Assembly 2 (short-circuit ||))

Implement the short-circuit || in two assembly styles for:

```
if ((r1 == r3) || (r5 == r6))
    r7 = r7 + 10;
```

Version 1 — Using branches:

```
CMP   r1, r3        ; compare r1 and r3
BEQ   ADDIT         ; if r1 == r3, skip checking second condition
CMP   r5, r6        ; compare r5 and r6
BNE   SKIP          ; if r5 != r6, both tests failed → skip
ADDIT: ADD r7, r7, #10
SKIP:
```

Version 2 — Using conditional execution:

```
CMP    r1, r3        ; compare r1 and r3
CMPNE  r5, r6        ; only compare r5 and r6 if the first compare failed
ADDEQ  r7, r7, #10   ; add if either compare returned Equal
```

Explanation: Short-circuit || means: if the first condition is true, do not evaluate the second. Version 1 uses explicit branches: BEQ on the first CMP skips the second compare and jumps to the ADD. Otherwise the second CMP runs; if it fails (BNE), the whole thing is skipped. Version 2 leverages ARM's conditional execution: CMPNE r5, r6 only runs if the first CMP did not produce Equal (i.e., the first test failed), which is exactly the short-circuit behavior. After this, the Z flag is set iff either compare returned Equal — so ADDEQ r7, r7, #10 performs the add in exactly the right cases. Conditional execution avoids branches, keeping the pipeline full and the code smaller.

Question 27 (Fill in the Blank, 6 points • C to Assembly II — strcpy)

Fill in the blanks of the strcpy assembly. Recall that a C string is null-terminated (#0).

```
strcpy   PROC
   EXPORT strcpy
```

```
loop
   LDRB  r2, [r1]     ; load byte from src
   STRB  r2, [r0]     ; store byte to dst
   ADD   r1, r1, #1   ; advance src pointer
   ADD   r0, r0, #1   ; advance dst pointer
   CMP   r2, #0       ; was the byte the null terminator?
   BNE   loop
   BX    LR
   ENDP
```

Explanation: The function copies bytes from *src to *dst until it copies (and stops after) the null terminator. After each LDRB/STRB pair, both pointers must advance by one byte: ADD r1, r1, #1 and ADD r0, r0, #1. The loop exit test is CMP r2, #0 — comparing the byte just copied to the null terminator. BNE loop continues as long as the byte was non-zero. When r2 == 0, the null terminator has just been stored at the destination, so the loop falls through to BX LR. Note that arguments arrive via R0 (dst) and R1 (src) per the ARM EABI calling convention.

Question 28 (Fill in the Blank, 4 points • Timer)

A 16-bit timer has: CPU clock = 360 MHz; PSC = 35999; up-counting; desired timer frequency = 1 kHz. Calculate the timer clock frequency and ARR.

Answer:

Timer clock frequency **= CPU clock / (PSC + 1) = 360 MHz / 36000 = 10 kHz**

ARR **= (Timer clock / Desired frequency) − 1 = (10 kHz / 1 kHz) − 1 = 9**

Explanation: The prescaler divides the CPU clock by (PSC + 1) to produce the timer's counting clock: 360 MHz ÷ 36000 = 10 kHz, so the timer counter ticks 10 000 times per second. In up-counting mode, the counter counts 0, 1, 2, …, ARR and then rolls over — that's (ARR + 1) ticks per period. To achieve a 1 kHz update event, we need 10 ticks per period, so ARR + 1 = 10 → ARR = 9. Note the "+1" in both PSC and ARR formulas: the registers hold one less than the divider because a value of 0 means "divide by 1."

Question 29 (Fill in the Blank, 6 points • Timer PWM)

A 16-bit timer has: CPU clock = 400 MHz; PSC = 399; ARR = 999; CCR = 700; down-counting; PWM mode 1.

Answer:

Timer clock frequency **= 400 MHz / (399 + 1) = 1 MHz**

Timer (PWM) frequency **= 1 MHz / (ARR + 1) = 1 MHz / 1000 = 1 kHz**

PWM duty cycle **= CCR / (ARR + 1) = 700 / 1000 = 0.7 (i.e., 70%)**

Explanation: The prescaler divides the CPU clock: 400 MHz ÷ (399 + 1) = 1 MHz timer tick rate. One full PWM period takes (ARR + 1) ticks, so the PWM frequency is 1 MHz / 1000 = 1 kHz. For PWM Mode 1 in down-counting, the output is active while CNT > CCR and inactive while CNT ≤ CCR (the symmetric counterpart of up-counting Mode 1, which is active while CNT < CCR). In both directions, CCR / (ARR + 1) is the active fraction of the period — so duty cycle = 700 / 1000 = 0.7 = 70%. Note that changing PSC changes the PWM frequency without affecting duty cycle, while changing CCR changes duty cycle without affecting frequency.

Question 30 (Fill in the Blank, 18 points • Program Execution Analysis)

Trace the following program in little-endian, starting NZCV = 0000. (Tips: MVN inverts all bits; LSLS is the only flag-setting instruction here.)

```
LDR  R1, =0x10000010
LDR  R2, [R1, #4]
MVN  R3, R2
```

```
MOV  R4, #0x00F0
AND  R5, R3, R4
EOR  R6, R5, R2
STR  R6, [R1, #8]
LSLS R7, R6, #28
```

Initial memory at 0x10000010 (byte order, increasing address):

```
FF EF CD AB   00 00 CD AB   AA BB CC DD   ...
```

Final register values:

R1 = **0x10000010**
R2 = **0xABCD0000**
R3 = **0x5432FFFF**
R4 = **0x000000F0**
R5 = **0x000000F0**
R6 = **0xABCD00F0**
R7 = **0x00000000**

Final memory at 0x10000010 (bytes 8–11 overwritten):

```
FF EF CD AB   00 00 CD AB   F0 00 CD AB   ...
```

Final NZCV = 0110

Note on the last step:
LSLS (Logical Shift Left with flag update) shifts the bits of the register to the left by the specified immediate value.
Given:
R6 = 0xABCD00F0
LSLS R7, R6, #28
This shifts 0xABCD00F0 left by 28 bits.
Since each hexadecimal digit represents 4 bits, shifting left by 28 bits is equivalent to shifting left by 7 The lowest nibble (0) in R6 gets shifted to the highest nibble position, and all lower bits are filled with zeros.
Therefore, the result is: **R7 = 0x00000000**
Because the S suffix is used, the Zero (Z) flag will be set to 1, and the Carry (C) flag will capture the last bit shifted out, which is bit 3 of the original register—the highest bit of the F, which is 1.