# 7CSC111 Assembly Language Programming

# Fall 2025 Midterm Exam ANS

Student Name: _____ID：_____

| Total Points | |
|---|---|

**Note: This exam paper should be kept confidential and any dissemination violates copyright.**

| Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 |
|---|---|---|---|---|---|---|---|
| /10 | /10 | /20 | /10 | /10 | /10 | /20 | /10 |

**(In all the problems, the system is assumed to be 32-bit unless specified otherwise.)**

**Q1 Multiple-choice questions: enter your answer keys here:**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| A | C | B | C | B | C | C | C | B | A |

For the following multiple-choice questions, each question has exactly one correct answer key. If multiple choices are correct, choose the option "All of the above". **Fill in the answer keys in the table above. (Answer keys written in the question area will not be counted.)**

1. Which registers are general-purpose on ARM Cortex-M?
   A. R0–R12
   B. R13–R15
   C. CONTROL, BASEPRI, PRIMASK
   D. xPSR only
   ANS: A

2. What are the roles of R13, R14, and R15 on ARM Cortex-M?
   A. General purpose
   B. Program counter, stack pointer, and link register in that order
   C. Stack pointer (R13), link register (R14), program counter (R15)
   D. Interrupt mask registers
   ANS: C

3. On ARM Cortex-M3, the borrow and carry flags relation is:
   A. Carry = Borrow
   B. Carry = NOT Borrow
   C. Borrow always 0
   D. Carry always 0
   ANS: B

4. In two's complement, TC(x) can be obtained by:
   A. Invert bits
   B. Invert bits and subtract one
   C. Invert bits and add one
   D. Add one then invert bits
   ANS: C

5. In a 5-bit system, which statement is true about −16 ($10000_2$)?
   A. Its two's complement is $00000_2$
   B. Its two's complement is itself
   C. It cannot be represented
   D. It equals +16
   ANS: B (most negative number maps to itself)

6. What is the bit width of each register in ARM Cortex-M processors?
   A) 16 bits
   B) 24 bits
   C) 32 bits
   D) 64 bits
   ANS: C) 32 bits

7. In ARM assembly instruction format, what is typically the first operand (operand1)?
   A) Source register
   B) Immediate value
   C) Destination register
   D) Memory address
   ANS: C) Destination register

8. When adding two 64-bit integers split across two registers each, which instruction pair correctly handles the low and high halves?
   A. ADC for high halves, then ADDS for low halves to set carry
   B. ADC for low halves, then ADDS for high halves
   C. ADDS for low halves to set carry, then ADC for high halves
   D. ADD for low halves, then ADD for high halves
   ANS: C (the order cannot be reversed)

9. In ARM subtraction, what does the carry flag C indicate when a borrow occurs in SUBS?
   A. C = 1 when there is a borrow
   B. C = 0 when there is a borrow
   C. C toggles regardless of borrow
   D. C is always preserved from the previous instruction
   ANS: B (Carry equals not Borrow)

10. Which single instruction multiplies a register by 17 using the barrel shifter on the second operand?
    A. ADD r4, r4, r4, LSL #4
    B. RSB r5, r5, r5, LSL #5
    C. ADD r1, r0, r0, ASR #3
    D. MUL r1, r0, #17
    ANS: A

**Q2. (10 points)** Assuming a 4-bit system. Show the equivalent decimal values when the data is interpreted as unsigned binary or signed binary.

| Binary Value | Signed Decimal Value | Unsigned Decimal Value |
|---|---|---|
| 1111 | | |
| 1011 | | |
| 0110 | | |
| 1001 | | |
| 0011 | | |

ANS:

| Binary Value | Signed Decimal Value | Unsigned Decimal Value |
|---|---|---|
| 1111 | -1 | 15 |
| 1011 | -5 | 11 |
| 0110 | 6 | 6 |
| 1001 | -7 | 9 |
| 0011 | 3 | 3 |

**Q3. (20 points)**
(a) (5 points) Assume an 8-bit system. Copy bits 5..3 in R4 to be the rightmost bits of R5, and set the other R5 bits to 0.
ANS:

```
MOV  R0, #0x38      @ build mask 00111000
Or:
MOV  R0, #7 << 3    @ build mask 00111000
Or:
MOV  R0, #7         @ R0 = 00000111
LSL  R0, R0, #3     @ R0 = 00111000  (now mask is 0x38)

AND  R5, R4, R0     @ isolate bits 5..3. Or: AND  R5, R4, #0x38
LSR  R5, R5, #3     @ shift them to bits 2..0
```

Alternatively, compute R5 = (R4 >> 3) & 0x7 (shift right then isolate the lower 3 bits)

```
MOV  R5, R4, LSR #3  @ R5 = R4 >> 3
Or:
LSR  R5, R4, #3      @ R5 = R4 >> 3

AND  R5, R5, #0x7    @ keep only lower 3 bits
```

(b) (5 points) Assume an 8-bit system. Copy the bottom four bits 3..0 in R0 to bits 6..3 in R1, and keep the other bits in R1 unchanged.

```
    AND  R2, R0, #0x0F   ; take bits 3..0 from R0 into R2
    LSL  R2, R2, #3      ; move into positions 6..3 in R2
    AND  R1, R1, #0x87   ; clear bits 6..3 in R1 with mask 0x87 = 10000111
                         ; (preserve other bits of R1)
    Or:
    BIC  R1, R1, #0x78   ; clear bits 6..3 in R1 with mask 0x78 = 01111000

    ORR  R1, R1, R2      ; insert new bits
```

ANS:
Or: use instruction BFI <Rd>, <Rn>, #<lsb>, #<width>
Semantics: Takes the lowest <width> bits from <Rn>. Inserts them into <Rd>, starting at bit position <lsb>. Bits outside the range [lsb, lsb+width−1] of <Rd> remain unchanged.

```
BFI R1, R0, #3, #4   ; copy 4 bits from R0 (starting at bit 0) into R1 at bit
position 3
```

(c) (10 points) Assume R0 and R1 are initialized with the values below. Write a sequence of assembly instructions that would produce the values in registers R2–R5, from R0 and R1.

Given:

| | |
|---|---|
| R0 | 0xCAFEFADE |
| R1 | 0x00FF00FF |

Produce:

| | |
|---|---|
| R2 | 0xCAFFFAFF |
| R3 | 0xCAFFFA8F |
| R4 | 0xFFFFFFCA |
| R5 | 0xDEFF00FF |

ANS:

```
ORR R2, R0, R1       @ R2 = 0xCAFEFADE | 0x00FF00FF = 0xCAFFFAFF
BIC R3, R2, 0x70     @ R3 = 0xCAFFFAFF & not 01110000 = 0xCAFFFA8F
ASR R4, R0, #24      @ R4 = 0xCAFEFADE >> 24 = 0xFFFFFFCA (sign-extended)
```

For R5, option 1:

```
LSL R5, R0, #24      @ R5 = 0xCAFEFADE << 24 = 0xDE000000
ORR R5, R5, R1       @ R5 = 0xDE000000 | 0x00FF00FF = 0xDEFF00FF
```

For R5, option 2:

```
ORR  R5, R1, R0, LSL #24
```

For R5, option 3:

```
MOV  R5, R1            @ R5 = 0x00FF00FF
BFI  R5, R0, #24, #8   @ Bit Field Insert – insert the lowest 8 bits of R0
bits [7:0] (0xDE) into bits [31:24] of R5
```

Note that this instruction is invalid due to constraints on the immediate value (c.f., pp. 78-80 in Ch4 ARM Arithmetic Logic):

```
AND R3, R2, #0xFFFFFF8F
```

**Q4. (10 points)** Assume a 4-bit system. For each of the following operations, compute the result and NZCV flags, based on the first row that has been given to you.

| Operation | Result in binary | Equivalent unsigned arithmetic in decimal | Equivalent signed arithmetic in decimal | NZCV |
|---|---|---|---|---|
| **1001 + 0010** | **1011** | **9 + 2 = 11** | **-7 + 2 = -5** | **1000** |
| 1101 + 1100 | | | | |
| 1101 - 1100 | | | | |
| 1100 + 1010 | | | | |
| 0100 - 0110 | | | | |
| 0100 + 0010 | | | | |

ANS: (Unsigned 4-bit range: 0 to 15, Signed 4-bit range: −8 to +7. In case of overflow, please write the actual result after truncation to 4 bits. In this exam, you are also given credit for the ground truth result that exceeds the range.)

| Operation | Result in binary | Equivalent unsigned arithmetic in decimal | Equivalent signed arithmetic in decimal | NZCV |
|---|---|---|---|---|
| **1001 + 0010** | **1011** | **9 + 2 = 11** | **-7 + 2 = -5** | **1000** |
| 1101 + 1100 | 1001 | 13 + 12 = 9 (or 25) | -3 + (-4) = -7 | 1010 |
| 1101 - 1100 | 0001 | 13 - 12 = 1 | -3 - (-4) = 1 | 0010 |
| 1100 + 1010 | 0110 | 12 + 10 = 6 | -4 + (-6) = 6 (or -10) | 0011 |
| 0100 - 0110 | 1110 | 4 - 6 = 14 (or -2) | 4 - 6 = -2 | 1000 |
| 0100 + 0010 | 0110 | 4 + 2 = 6 | 4 + 2 = 6 | 0000 |

**Key Explanations:**
1101 + 1100: Result wraps to 1001. Carry out occurs (C=1), but no signed overflow since -3 + (-4) = -7 is valid in 4-bit signed range.
1101 - 1100: Straightforward subtraction giving 1. No borrow needed (C=1).
1100 + 1010: Adding two negative numbers (-4 + -6 = -10) produces a positive result (+6) because -10 is outside the 4-bit signed range [-8, 7]. This sets overflow V=1.
0100 - 0110: Subtracting larger from smaller requires a borrow (C=0). Result is 1110 = -2 in signed, 14 in unsigned.
0100 + 0010: Clean addition with no flags set - result is positive, non-zero, no carry, no overflow.

**Q5. (10 points)** For each of the following instructions (executed individually, not sequentially), compute the result and NZCV flags. Assume initially R0=0xFFFFFFFF, R1=0x00000001, NZCV=0000.

| Instruction | Result in dest. register R0 | NZCV |
|---|---|---|
| MOVS R0, #0 | | |
| ANDS R0, R0, #0 | | |
| ORRS R0, R0, R1 | | |

| | | |
|---|---|---|
| ANDS R0, R0, R1 | | |
| ADDS R0, R0, R1 | | |
| SUBS R0, R0, R1 | | |
| ADDS R0, R0, R1, LSL #31 | | s |
| BICS R0, R0, R1, LSL #31 | | |
| EORS R0, R0, R1, LSL #31 | | |
| LSLS R0, R0, #31 | | |

ANS: (Assume initially R0=0xFFFFFFFF, R1=0x00000001.)

| Instruction | Result in R0 | NZCV | Explanation |
|---|---|---|---|
| MOVS R0, #0 | 0x00000000 | 0100 | Result is zero (Z=1), bit 31=0 (N=0), MOVS doesn't affect C/V |
| ANDS R0, R0, #0 | 0x00000000 | 0100 | Any value AND 0 = 0, so Z=1, N=0, logical ops don't affect C/V |
| ORRS R0, R0, R1 | 0xFFFFFFFF | 1000 | 0xFFFFFFFF \| 0x00000001 = 0xFFFFFFFF, bit 31=1 (N=1), non-zero (Z=0) |
| ANDS R0, R0, R1 | 0x00000001 | 0000 | 0xFFFFFFFF & 0x00000001 = 0x00000001, bit 31=0 (N=0), non-zero (Z=0) |
| ADDS R0, R0, R1 | 0x00000000 | 0110 | 0xFFFFFFFF + 0x00000001 = 0 with carry out (C=1), Z=1, no overflow |
| SUBS R0, R0, R1 | 0xFFFFFFFE | 1010 | 0xFFFFFFFF - 0x00000001 = 0xFFFFFFFE, N=1, no borrow (C=1) |
| ADDS R0, R0, R1, LSL #31 | 0x7FFFFFFF | 0011 | R1<<31=0x80000000; 0xFFFFFFFF+0x80000000=0x7FFFFFFF; Overflow! (V=1), Carry (C=1) |
| BICS R0, R0, R1, LSL #31 | 0x7FFFFFFF | 0000 | R0 & ~(0x80000000) = 0x7FFFFFFF, clears bit 31, N=0, Z=0 |
| EORS R0, R0, R1, LSL #31 | 0x7FFFFFFF | 0000 | 0xFFFFFFFF ^ 0x80000000 = 0x7FFFFFFF, flips bit 31, N=0, Z=0 |
| LSLS R0, R0, #31 | 0x80000000 | 1010 | 0xFFFFFFFF<<31=0x80000000, N=1, last bit out was 1 (C=1) |

**Key Explanations:**

1) ADDS R0, R0, R1, LSL #31 causes **signed overflow** (V=1) when adding two negative numbers that produce a positive result
2) BICS R0, R0, R1, LSL #31 and EORS R0, R0, R1, LSL #31 both produce 0x7FFFFFFF by clearing/flipping bit 31
3) LSLS R0, R0, #31 he carry flag C receives the last bit shifted out, which for a shift of 31 is the original bit 1, which is 1.

**Q6. (10 points)**

(a) Suppose r0 = 0x00008000, and the following memory layout:

| Address | Data |
|---|---|
| 0x00008007 | 0x79 |

| | |
|---|---|
| 0x00008006 | 0xCD |
| 0x00008005 | 0xA3 |
| 0x00008004 | 0xFD |
| 0x00008003 | 0x0D |
| 0x00008002 | 0xEB |
| 0x00008001 | 0x2C |
| 0x00008000 | 0x1A |

(a1) (3 points) ARM processors can be configured as big-endian or little-endian. What is the value of r1 after running LDR r1, [r0]?

a. If little-endian, r1 = _____

b. If big-endian, r1= _____

ANS:

a. If little-endian, r1 = 0x0DEB2C1A
b. If big-endian, r1 = 0x1A2CEB0D

(a2) (3 points) Suppose the system is little-endian. What are the values of r1 and r0? Each instruction is executed individually, not sequentially.

a. LDR r1,[r0,#4]
b. LDR r1,[r0],#4
c. LDR r1,[r0,#4]!

After LDR r1,[r0,#4]

r0 = _____

r1 = _____

After LDR r1,[r0],#4

r0 = _____

r1 = _____

After LDR r1,[r0,#4]!

r0 = _____

r1 = _____

ANS:
After LDR r1,[r0,#4]
r0 = 0x00008000
r1 = 0x79CDA3FD
After LDR r1,[r0],#4
r0 = 0x00008004

r1 = 0x0DEB2C1A
After LDR r1,[r0,#4]!
r0 = 0x00008004
r1 = 0x79CDA3FD

**Explanations**:
1) `LDR r1, [r0, #4]` — loads from `r0+4` but **does not** change `r0`.
2) `LDR r1, [r0], #4` — post-index; load from `r0`, then `r0 += 4`.
3) `LDR r1, [r0, #4]!` — pre-index; `r0 += 4` then load from new `r0`

(b) (4 points) Assume little-endian memory ordering. Suppose r0 = 0x2000,0000 and r1 = 0x12345678. All bytes in memory are initialized to 0x00. Fill the following table after the assembly program has finished execution.

```
STR r1, [r0], #4
STR r1, [r0, #4]!
STR r1, [r0]
```

ANS:

| Memory Address | Initial Memory Content | Final Memory Content |
|---|---|---|
| 0x2000,0013 | 00 | 00 |
| 0x2000,0012 | 00 | 00 |
| 0x2000,0011 | 00 | 00 |
| 0x2000,0010 | 00 | 00 |
| 0x2000,000F | 00 | 00 |
| 0x2000,000E | 00 | 00 |
| 0x2000,000D | 00 | 00 |
| 0x2000,000C | 00 | 00 |
| 0x2000,000B | 00 | 12 |
| 0x2000,000A | 00 | 34 |
| 0x2000,0009 | 00 | 56 |
| 0x2000,0008 | 00 | 78 |
| 0x2000,0007 | 00 | 00 |
| 0x2000,0006 | 00 | 00 |
| 0x2000,0005 | 00 | 00 |
| 0x2000,0004 | 00 | 00 |
| 0x2000,0003 | 00 | 12 |
| 0x2000,0002 | 00 | 34 |
| 0x2000,0001 | 00 | 56 |
| 0x2000,0000 | 00 | 78 |

**Q7. (20 points)** Show all updates to registers as the assembly code shown below runs, assuming **big-endian** ordering. (Memory addresses increase from left to right in the table.) Fill in the tables of final register values and memory contents. Show the NZCV flags after execution, assuming NZCV=0000 initially.

| Assembly program |
|---|
| LDR R1, =0x10000010<br>LDR R2, [R1]<br>LDR R3, [R1, #4]!<br>ASR R4, R2, #12<br>MOVW R5, #14<br>LSL R6, R5, #9<br>BIC R7, R2, R6<br>AND R8, R7, R6<br>STR R7, [R1, #8] |

Initial memory contents:

| 0x10000010 | FF | EF | CD | AB | 00 | 00 | CD | AB | AA | BB | CC | DD | EE | FF | 11 | 22 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Final register values:

| R1 | | R2 | | R3 | | R4 | |
|---|---|---|---|---|---|---|---|
| R5 | | R6 | | R7 | | R8 | |

Final memory contents:

| 0x10000010 | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Final NZCV = _____

ANS:
Registers

| R1 | 0x10000014 | R2 | 0xFFEFCDAB | R3 | 0x0000CDAB | R4 | 0xFFFFFEFC |
|---|---|---|---|---|---|---|---|
| R5 | 0x0000000E | R6 | 0x00001C00 | R7 | 0xFFEFC1AB | R8 | 0x00000000 |

Final memory contents:

| 0x10000010 | FF | EF | CD | AB | 00 | 00 | CD | AB | AA | BB | CC | DD | **FF** | **EF** | **C1** | **AB** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Final NZCV = 0000

Explanations (not required for exam):

1. `LDR R1, =0x10000010` → `R1 = 0x10000010`.
2. `LDR R2, [R1]` → load bytes `FF EF CD AB` → `R2 = 0xFFEFCDAB`.
3. `LDR R3, [R1, #4]!` → **pre-index**: it increments `R1` to `0x10000014` before the load. Load `00 00 CD AB` → `R3 = 0x0000CDAB`.
4. ASR R4, R2, #12 → arithmetic shift right 12 of `0xFFEFCDAB` → `R4 = 0xFFFFFEFC`. NZCV unchanged (assume initial NZCV = 0000). (Please refer to p. 60 "Notes on Shifts and Flags" in Ch4 ARM Arithmetic Logic.)
    1. Hypothetically, if we had ASRS instead of ASR, then `ASRS R4, R2, #12` → arithmetic shift right 12 of `0xFFEFCDAB` → `R4 = 0xFFFFFEFC`. Flags: N=1, Z=0, C=bit11(R2)=1 (last bit shifted out is the leftmost bit of DAB), V unchanged → NZCV = 1010.
5. `MOVW R5, #14` → `R5 = 0x0000000E`.
6. `LSL R6, R5, #9` → `R6 = 0x00001C00`.
7. `BIC R7, R2, R6` → `R7 = R2 & ~R6 = 0xFFEFC1AB`.
8. `AND R8, R7, R6` → `R8 = 0x00000000`.
9. `STR R7, [R1, #8]` → stores at address `R1 + 8` = `0x10000014 + 8 = 0x1000001C`. Write `0xFFEFC1AB` (big-endian bytes `FF EF C1 AB`) to `0x1000001C..0x1000001F`.

Final memory (address order from `0x10000010`):

- `0x10000010..0x10000013` = `FF EF CD AB`
- `0x10000014..0x10000017` = `00 00 CD AB`
- `0x10000018..0x1000001B` = `AA BB CC DD` (unchanged)
- `0x1000001C..0x1000001F` = `FF EF C1 AB` (stored value)

**Q8. (10 points)** Consider an array[] of 25 integers (each integer is 4 bytes). Assume register R2 holds the base memory address of array, i.e., array[0]. A compiler associates variables x and y with registers R0 and R1, respectively. Translate this C program into ARM assembly language based on the provided comments.. (The original problem did not specify if x is unsigned or signed integer, so both LSR and ASR are graded as correct for computing x/4.)

| C program | Assembly program |
|---|---|
| uint32_t x = array[5] + y;<br>array[6] = x * 4;<br>array[7] = x / 4;<br>array[8] = x - 10;<br>array[9] = x * (x - 1); | _____; load array[5] (with 5*4 = 20 bytes offset) into<br>; temporary register R3<br><br>_____; x = array[5] + y<br><br>_____; R3 = x * 4<br><br>_____; store into array[6]<br><br>_____; R3 = x / 4<br><br>_____; store into array[7]<br><br>_____; R3 = x – 10<br><br>_____; store into array[8]<br><br>_____; R4 = x – 1<br><br>_____; R3 = x * (x - 1)<br><br>_____; store into array[9] |

ANS:

| C program | Assembly program |
|---|---|
| uint32_t x = array[5] + y;<br>array[6] = x * 4;<br>array[7] = x / 4;<br>array[8] = x - 10;<br>array[9] = x * (x - 1); | LDR R3, [R2, #20] ; load array[5] (with 5*4 = 20 bytes offset) into a temp register<br>ADD R0, R3, R1 ; x = array[5] + y<br>LSL R3, R0, #2 ; R3 = x * 4.<br>Or:<br>MOV R1, #4<br>MUL R3, R0, R1 ; Note that MUL R3, R0, #4 is not valid syntax<br><br>STR R3, [R2, #24] ; store into array[6]<br>LSR R3, R0, #2 ; R3 = x / 4. Note that UDIV R3, R0, #2 is not valid syntax<br>STR R3, [R2, #28] ; store into array[7]<br>SUB R3, R0, #10 ; R3 = x - 10<br>STR R3, [R2, #32] ; store into array[8]<br>SUB R4, R0, #1 ; R4 = x - 1 |

| | |
|---|---|
| | MUL R3, R0, R4 ; R3 = x * (x - 1)<br>STR R3, [R2, #36] ; store into array[9] |

Or:

| C program | Assembly program |
|---|---|
| uint32_t x = array[5] + y;<br>array[6] = x * 4;<br>array[7] = x / 4;<br>array[8] = x - 10;<br>array[9] = x * (x - 1); | LDR R3, [R2], #20 ; load array[5] (with 5*4 = 20 bytes offset) into a temp register<br>ADD R0, R3, R1 ; x = array[5] + y<br>LSL R3, R0, #2 ; R3 = x * 4.<br>STR R3, [R2], #4 ; store into array[6]<br>LSR R3, R0, #2 ; R3 = x / 4. Note that UDIV R3, R0, #2 is not valid syntax<br>STR R3, [R2], #4 ; store into array[7]<br>SUB R3, R0, #10 ; R3 = x - 10<br>STR R3, [R2], #4 ; store into array[8]<br>SUB R4, R0, #1 ; R4 = x - 1<br>MUL R3, R0, R4 ; R3 = x * (x - 1)<br>STR R3, [R2], #4 ; store into array[9] |

Appendix; Conversion table. (You may tear off and discard this page.)

| Decimal | Binary | Hex |
|---|---|---|
| 0 | 0000 | 0x0 |
| 1 | 0001 | 0x1 |
| 2 | 0010 | 0x2 |
| 3 | 0011 | 0x3 |
| 4 | 0100 | 0x4 |
| 5 | 0101 | 0x5 |
| 6 | 0110 | 0x6 |
| 7 | 0111 | 0x7 |
| 8 | 1000 | 0x8 |
| 9 | 1001 | 0x9 |
| 10 | 1010 | 0xA |
| 11 | 1011 | 0xB |
| 12 | 1100 | 0xC |
| 13 | 1101 | 0xD |
| 14 | 1110 | 0xE |
| 15 | 1111 | 0xF |