Name___________________________________

ECE 785
TOPICS IN ADVANCED COMPUTER DESIGN
SPRING 2019 FINAL EXAM

Closed book, closed notes, one 8.5”x11” page of notes allowed. Calculator allowed. You are not permitted to have a computer or other electronic assistance. Show your work for each problem for full credit. Assume that C code is compiled for the ARM Cortex-A8 architecture and the TI AM335x processor on the Beaglebone Black Wireles using gcc according to the ARM Architecture procedure calling standard (AAPCS).

Question	Maximum Score	Points Off	Score
1	5		
2	10		
3	20		
4	20		
5	20		
6	25		
Total	100		

Please read and sign this statement: I have not received assistance from anyone nor assisted others while taking this test. I have also notified the test proctor of any violations of the above conditions.

Signature ____

Name___________________________________

Consider the following source code and corresponding object code, generated with –O3 optimization. The code encrypts input data from din and stores it in dout. Encryption is performed by exclusive-oring the input data with a series of pseudo-random values. These values are generated using a linear-feedback shift register which starts with the value of the seed argument.

Source Code:

```
void LFSR(uint16_t * din, uint16_t * dout, uint16_t seed) {
  uint16_t lfsr, n;
  uint8_t bit;

  lfsr = seed;
  for (n = 0; n<N; n++) {
    // source code from http://en.wikipedia.org/wiki/Linear_feedback_shift_register
    /* taps: 16 14 13 11; characteristic polynomial: x^16 + x^14 + x^13 + x^11 + 1 */
    bit = ((lfsr >> 0) ^ (lfsr >> 2) ^ (lfsr >> 3) ^ (lfsr >> 5) ) & 1;
    lfsr = lfsr >> 1;
    if (bit)
      lfsr |= 0x8000;
    *dout++ = *(din++) ^ lfsr;
  }
}
```

Object Code:

```
        push    {r4, r5, r6, r7}
        subs    r1, r1, #2
        add     r7, r0, #2048
        mov     r3, r2
.L38:
        lsrs    r2, r3, #3
        lsrs    r5, r3, #1
        eor     r2, r2, r3, lsr #2
        eors    r2, r2, r3
        orr     r6, r5, #32768
        eor     r2, r2, r3, lsr #5
        tst     r2, #1
        ite     eq
        moveq   r3, r5
        movne   r3, r6
        ldrh    r4, [r0], #2
        eor     r2, r3, r4
        strh    r2, [r1, #2]!       @ movhi
        cmp     r0, r7
        bne     .L38
        pop     {r4, r5, r6, r7}
        bx      lr
```

Name___________________________________

1. Circle each basic block in the object code above. Label each basic block.

2. Draw the control-flow graph for the object code. Label each basic block. You do not need to include the instructions in the basic blocks.

3. The source code has a statement: **bit = ((lfsr >> 0) ^ (lfsr >> 2) ^ (lfsr >> 3) ^ (lfsr >> 5)) & 1**
   a. Which object code instructions implement this source code? Mark them above with a 3.
   b. Explain how that object code works.

Name___________________________________

4. The source code has a statement: **if (bit) lfsr |= 0x8000**
    a. Which object code instructions implement this source code? Mark them above with a 4.
    b. Explain how that object code works.

5. Imagine that you profile the code using perf.
    a. Which object code instruction do you expect to dominate execution time, and why?
    b. How can you change the code to minimize this problem? Explain why the change helps, and under what conditions it completely eliminates the problem.

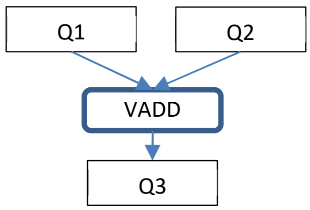
Name_________________________________

6. We would like to make **LFSR** run faster by unrolling its loop and then using the Advanced SIMD instructions for vectorization.
   a. Explain how to unroll the loop in the function **LFSR** by a factor of eight. Write pseudocode or C code to show your approach.

Name___________________________________

b. Vectorize the unrolled loop. Sketch out a diagram showing the dataflow with Advanced SIMD registers and instructions (please see the example below). Hint: rather than compute **bit** with so many shifts and exclusive-ors, you may wish to use the VCNT instruction, which counts the number of set bits (equal to 1) in each element.

Example dataflow notation for VADDQ Q3, Q1, Q2:



Name______________________________________

Prefixes for Parallel Instructions	
S	Signed arithmetic modulo 2^8 or 2^{16}, sets CPSR GE bits
Q	Signed saturating arithmetic
SH	Signed arithmetic, halving results
U	Unsigned arithmetic modulo 2^8 or 2^{16}, sets CPSR GE bits
UQ	Unsigned saturating arithmetic
UH	Unsigned arithmetic, halving results

Condition Field		
Mnemonic	**Description**	**Description (VFP)**
EQ	Equal	Equal
NE	Not equal	Not equal, or unordered
CS / HS	Carry Set / Unsigned higher or same	Greater than or equal, or unordered
CC / LO	Carry Clear / Unsigned lower	Less than
MI	Negative	Less than
PL	Positive or zero	Greater than or equal, or unordered
VS	Overflow	Unordered (at least one NaN operand)
VC	No overflow	Not unordered
HI	Unsigned higher	Greater than, or unordered
LS	Unsigned lower or same	Less than or equal
GE	Signed greater than or equal	Greater than or equal
LT	Signed less than	Less than, or unordered
GT	Signed greater than	Greater than
LE	Signed less than or equal	Less than or equal, or unordered
AL	Always (normally omitted)	Always (normally omitted)

All ARM instructions (except those with Note C or Note U) can have any one of these condition codes after the instruction mnemonic (that is, before the first space in the instruction as shown on this card). This condition is encoded in the instruction.

All Thumb-2 instructions (except those with Note U) can have any one of these condition codes after the instruction mnemonic. This condition is encoded in a preceding IT instruction (except in the case of conditional Branch instructions). Condition codes in instructions must match those in the preceding IT instruction.

On processors without Thumb-2, the only Thumb instruction that can have a condition code is `B <label>`.

Operation mnemonic

Destination, source1, source2 registers

V {QHDR} XXX.td.t1.t2   Dd, Dn, Dm
                                              Qd, Qn, Qm

Optional features:
- Q = saturating
- H = halving of data width
- D = doubling of data width
- R = rounding

Data type specifiers (dst, src1, src2):

8, 16, 32 or 64	size only
I8, I16, I32, 64	integer
F32	float
P8, P16	polynominal
S8, S16, S32, S64	signed
U8, U16, U32, U64	unsigned

For example:

Training-figure03.vsd

VRSBH.I16.I32   d2, q0, q1   ; subtract returning high half with rounding

Name_______________________________

Mnemonic	Brief description	Mnemonic	Brief description
VABA, VABD	Absolute difference and Accumulate, Absolute Difference	**VQDMULL**	Saturating Doubling Multiply
VABS	Absolute value	**VQDMULH**	Saturating Doubling Multiply returning High half
VACGE, VACGT	Absolute Compare Greater than or Equal, Greater Than	**VQMOV{U}N**	Saturating Move (register)
VACLE, VACLT	Absolute Compare Less than or Equal, Less Than (pseudo-instructions)	**VQNEG**	Negate, saturate
VADD	Add	**VQRDMULH**	Saturating Doubling Multiply returning High half
VADDHN	Add, select High half	**VQRSHL**	Shift Left, Round, saturate (by signed variable)
VAND	Bitwise AND	**VQRSHR{U}N**	Shift Right, Round, saturate (by immediate)
VBIC	Bitwise Bit Clear (register, immediate)	**VQSHL**	Shift Left, saturate (by immediate, signed variable)
VBIF, VBIT, VBSL	Bitwise Insert if False, Insert if True, Select	**VQSHR{U}N**	Shift Right, saturate (by immediate)
VCMPE, VCEQ, VCMPLE, VCLE, VCMPLT, VCLT	Compare Equal, Less than or Equal, Compare Less Than	**VQSUB**	Subtract, saturate
VCMPGE, VCGE, VCMPGT, VCGT	Compare Greater than or Equal, Greater Than	**VRADDHN**	Add, select High half, Round
VCMPLE, VCLE, VCMPLT, VCLT	Compare Less than or Equal, Compare Less Than (pseudo-instruction)	**VRECPE**	Reciprocal Estimate
VCLS, VCLZ, VCNT	Count Leading Sign bits, Count Leading Zeros, and Count set bits	**VRECPS**	Reciprocal Step
VCVT	Convert fixed-point or integer to/from floating-point	**VREV**	Reverse elements
VDUP	Duplicate scalar to all lanes of vector	**VRHADD**	Halving Add, Round
VEOR	Bitwise Exclusive OR	**VRINT**	Round to integer
VEXT	Extract	**VRSHR**	Shift Right and Round (by immediate)
VFMA, VFMS	Fused Multiply Accumulate, Fused Multiply Subtract	**VRSHRN**	Shift Right, Round, Narrow (by immediate)
VHADD, VHSUB	Halving Add, Halving Subtract	**VRSQRTE**	Reciprocal Square Root Estimate
VLD	Vector Load	**VRSQRTS**	Reciprocal Square Root Step
VMAX, VMIN	Maximum, Minimum	**VRSRA**	Shift Right, Round, and Accumulate (by immediate)
VMAXNM, VMINNM	Maximum, Minimum, consistent with IEEE 754-2008	**VRSUBHN**	Subtract, select High half, Round
VMLA, VMLS	Multiply Accumulate, Multiply Subtract (vector, by scalar)	**VSHL**	Shift Left (by immediate)
VMOV	Move (immediate, register)	**VSHR**	Shift Right (by immediate)
VMOVL, VMOV{U}N	Move Long, Move Narrow (register)	**VSHRN**	Shift Right, Narrow (by immediate)
VMRS, VMSR	Move from/to special	**VSLI**	Shift Left and Insert
VMUL	Multiply (vector, scalar)	**VSRA**	Shift Right, Accumulate (by immediate)
VMVN	Move Negative (immediate)	**VSRI**	Shift Right and Insert
VNEG	Negate	**VST**	Vector Store
VORN	Bitwise OR NOT	**VSUB**	Subtract
VORR	Bitwise OR (register, immediate)	**VSUBHN**	Subtract, select High half
VPADD, VPADAL	Pairwise Add, Pairwise Add and Accumulate	**VSWP**	Swap vectors
VPMAX, VPMIN	Pairwise Maximum, Pairwise Minimum	**VTBL, VTBX**	Vector table look-up
VQABS	Absolute value, saturate	**VTRN**	Vector transpose
VQADD	Add, saturate	**VTST**	Test bits
VQDMLAL, VQDMLSL	Saturating Doubling Multiply Accumulate, and Multiply Subtract	**VUZP, VZIP**	Vector interleave and de-interleave