COEN 20 **FINAL EXAM STUDY GUIDE** **FALL 2016**

IMPORTANT: The final exam will be comprehensive, so in addition to what is listed below, be sure to review the topics listed in the midterm study guide.

CHAPTER 7: MANIPULATING BITS

1. Basic Shift Operations (LSL, LSR, ROR, RRX, ASR)
   - 1.1. Difference between integer division by 2^K and ASR
   - 1.2. Implementing 64-bit shift operations (aka “Multiple Precision Shifts”)
   - 1.3. Three contexts where shift operations are used
     - 1.3.1. Address calculation expressions (e.g., “[R0,R1,LSL 2]”)
     - 1.3.2. Shift applied to third operand of an instruction (e.g., “ADD R0,R1,R2,ASR 31”)
     - 1.3.3.Regular Shift instruction (e.g., “LSR R0,R1,5”)
2. Bitwise Instructions (AND, ORR, EOR, BIC, ORN, MVN)
3. Bitfield Instructions (BFC, BFI, UBFX, SBFX)
4. Miscellaneous Bit Manipulation Instructions (I won’t test you on these)
5. Bit-Banding (I won't test you on this)

CHAPTER 8: MULTIPLICATION AND DIVISION REVISITED

1. Multiplication Using Left Shifts (Combinations of LSL, ADD and SUB)
2. Division by a Power of Two (Adding 2^K-1 prior to shift to get correct quotient)
3. Division by an Arbitrary Constant (aka “reciprocal multiplication”)
   - 3.1. Calculation of constant multiplier for a given constant divisor
   - 3.2. MS Half of double-length product contains the integer quotient
   - 3.3. Accuracy issues and strategies for improvement
4. Remainder When Dividing by 2^k
   - 4.1. Bitwise-AND of x and 2^K-1 produces x mod 2^k
   - 4.2. Difference between modulus and remainder
5. Calculating True Modulus with Arbitrary Divisor
   - 5.1. Add divisor to remainder when it’s negative

CHAPTER 9: GETTING STARTED WITH FLOATING POINT

1. Data Types for Real Numbers
   - 1.1. 32-bit IEEE float uses
     - 1.1.1. Sign + magnitude representation
     - 1.1.2. 8-bit excess-128 exponent
     - 1.1.3. 23-bit significand
     - 1.1.4. “Implied” MS bit of significand
2. Floating-Point Registers
   - 2.1. 32-bit registers S0-S31
   - 2.2. 64-bit registers D0-D15
3. Function Parameters and Return Values
   - 3.1. float parameters passed using S0-S15
   - 3.2. float result is returned in S0
4. Copying Data
   - 4.1. Copying float constants into float registers

4.1.1. VMOV: Only a small number of constants can be loaded
4.1.2. VLDR: Any float constant can be loaded from memory
4.2. Copying between float registers (VMOV)
4.3. Copying from memory to float register (VLDR)
4.3.1. VLDMIA, VLDMDB (I won't test you on these)
4.4. Copying from float register to memory (VSTR)
4.4.1. VSTMIA, VSTMDB (I won't test you on these)
5. Converting Between Integers and Real Numbers (four variations of VCVT)
5.1. (I won't test you on rounding)
6. Arithmetic with Real Numbers
6.1. VADD, VSUB, VNEG, VABS, VMUL, VDIV (all need .F32 suffix)
6.2. VFMA (or VLMA), VFMS (or VLMS): I won't test you on these
7. Comparing Real Numbers
7.1. VCMP.F32 followed by VMRS to copy FPU flags to Core flags

CHAPTER 10: WORKING WITH FIXED-POINT REAL NUMBERS

1. Q Format and The Imaginary Binary Point
2. Addition and Subtraction of Fixed-Point Reals
2.1. Use regular ADD and SUB instructions
3. Multiplication and Division of Fixed-Point Reals
3.1. Unsigned: $A_uB_u = 2^{64}A_{hi}B_{hi} + 2^{32}(A_{hi}B_{lo}+A_{lo}B_{hi}) + A_{lo}B_{lo}$
3.2. Signed: A_uB_u + correction terms
4. Fixed-Point Using a Universal Q16.16 Format
4.1. Fixed-Point product taken from middle of double-length integer product
4.2. Fixed-Point quotient requires placing fixed-point dividend in the middle of a double-length integer dividend
5. Multiplication of Q32.32 Fixed-Point Reals
5.1. Decompose each operand into its two 32-bit halves
5.2. Compute the 128-bit unsigned product by adding the 64-bit double-length products of the 32-bit halves.
5.3. Apply correction(s) to unsigned product to get a signed product:
5.3.1. If A < 0, subtract B from MS Half of product
5.3.2. If B < 0, subtract A from MS Half of product
6. Division of Q32.32 Fixed-Point Reals
6.1. Determine sign of result, divide using the unsigned magnitudes of the operands, change sign of result if necessary
6.2. Unsigned division: Use the "left shift and subtract" algorithm, but start with fixed-point dividend in the middle of the dividend bits.

CHAPTER 11: INLINE CODE

1. Inline Functions (requires "static inline" prefix on function header)
2. Inline Assembly
2.1. Basic asm
2.2. Extended asm

2.2.1. Template

2.2.2. OutputOperands

2.2.2.1. Must have a constraint modifier of "=" or "+"

2.2.2.2. A constraint modifier of "&" is optional.

2.2.3. InputOperands

2.2.3.1. Usually have a constraint of "r" or "I" or "ir"

2.2.3.2. Not needed if the same register is also used as an output operand

2.2.4. Clobbers ("cc" for flags; register name if its use is forced by programmer)

2.2.5. Constraints

2.2.5.1. "r" – allows use of a core register

2.2.5.2. "w" – allows use of a FP register

2.2.5.3. "i" – allows use of an integer constant

2.2.5.4. "X" – allows use of any kind of operand

2.2.6. Constraint Modifiers ("=" and "+" must be the 1st character)

2.2.6.1. "=" – Operand is used as an output; may be used later as an input

2.2.6.2. "+" – Operand is first used as an input, but is used later as an output

2.2.6.3. "&" – Do not use a register previously used as for an input operand

3. The Optional Volatile Keyword

3.1. Used to prevent optimizer from moving the code relative to surrounding code

CHAPTER 12: PROGRAMMING PERIPHERAL DEVICES

1. Blocking I/O (Example: Crc32 Peripheral)

1.1. Instructions that access I/O device "stall" until device is ready

1.2. Only used with devices that stall for only a couple of clock cycles

2. Polled Waiting Loop (Example: Random Number Generator)

2.1. An instruction loop continually tests the device status, waiting for "ready"

2.2. Then instructions are used to complete the next data byte transfer

2.3. Not used when response time is critical because program may not be executing the waiting loop when the device becomes ready.

3. Interrupt-Driven (Example: Timer Tick)

3.1. Allows regular program execution to be suspended so that the processor can execute code to transfer data to/from the device (significantly reduces response time)

3.2. Interrupts are a subset of "exceptions"

3.2.1. Interrupts are events caused by peripheral devices

3.2.2. Interrupts may be globally disabled/enabled (don't worry about how)

3.2.3. Interrupts may be individually disabled/enabled (don't worry about how)

3.3. All exceptions have:

3.3.1. An exception number (selects entry in vector table)

3.3.2. A priority level (determines which exception can pre-empt others)

3.3.3. An exception handler routine

3.3.4. An entry in the vector table (entry point address of handler)

3.4. Exception Response (performed automatically in hardware)

3.4.1. Stacking: Pushes PSR, return adrs, LR, R12, R3-R0

3.4.1.1. Current processor mode (Thread vs. Handler) recorded in pushed PSR.

3.4.1.2. Preserving R0-R3 allows Handler to use same ARM Procedure Call Standard

        3.4.2. Processor switched to Handler mode (changed in PSW)
        3.4.3. PC ← vector table[exception #]
    3.5. Exception Return (BX LR when in Handler mode):
        3.5.1. Unstacking: Pops R0-R3, R12, LR, return adrs (into PC), PSR
            3.5.1.1. Return address taken from stack instead of operand field of BX instruction
            3.5.1.2. Popping PSR restores previous processor mode.

4. Direct-Memory Access (Example: Memory-To-Memory Transfer)
    4.1. Instructions used to initialize DMA controller
        4.1.1. Number of bytes to transfer
        4.1.2. Direction of transfer
        4.1.3. Source and destination of transfer (memory or peripheral)
    4.2. Once started, DMA transfers and program execution continue independently
    4.3. Completion indicated by byte count going to zero
    4.4. An interrupt can be used to signal completion
5. The CPU Clock Cycle Counter (I won't test you on this)