STUDY GUIDE FOR EXAM 2

Exam 2 is designed to assess the material listed below. However, to answer some questions, you will need to know material from Exam 1 (including, but not limited to, the difference between ReadInt and ReadDec; when ADD/SUB will set the carry vs. overflow flags; how a .data section is laid out in memory, including endianness; what the current location counter $ is; etc.)

CHAPTER 6 – CONDITIONAL PROCESSING
(Lectures 8, 12, 18; Activity 10)

- *Section 6.2 (Boolean and Comparison Instructions)* – CMP: effects of subtraction on flags and use for comparison.

- *Section 6.3 (Conditional Jumps)* – Jumps based on specific flag values: JZ, JNZ, JC, JNC, JO, JNO, JS, JNS; jumps based on equality: JE, JNE, JECXZ; jumps based on unsigned comparisons: JA, JNA, JAE, JNAE, JB, JNB, JBE, JNBE; jumps based on signed comparisons: JG, JNG, JGE, JNGE, JL, JNL, JLE, JNLE.

- *Section 6.5 (Conditional Structures)* – Block-structured *if* statements. Compound expressions: logical AND, short-circuit evaluation; logical OR. *While* loops.
- Examples of questions and question types (not a comprehensive list):
  - Suppose AL = 255 and BL = 1. Will the instruction "cmp al, bl" set or clear the carry flag? overflow flag? sign flag? zero flag?
  - Suppose the status flags are: CF = 1, OF = 0, SF = 1, ZF = 0. Will the instruction JA jump or fall through? What about JG?
  - Which instruction should appear in the blank: JG or JA?
    ```
    call ReadDec
    cmp eax, 5
    ___ isLargerThanFive
    ```
  - Given pseudocode containing (possibly nested) *while* loops, *do-while* loops, and *if* statements, translate that code into assembly language. Or, given an alleged translation, describe why that translation is correct or incorrect.

MEMORY OPERANDS
(Lectures 19, 22, 23)

- *Section 4.3 (Data-Related Operators and Directives)* – OFFSET operator. PTR operator. LENGTHOF operator. SIZEOF operator.

- *Memory Operands (Sections 4.1, 4.4, 9.4, and supplemental)* – Direct memory operands. Indexed operands: label[reg] and [label+reg]; adding displacements; scale factors in indexed operands. Indirect operands; using PTR with indirect operands. Indirect operands and arrays. Base-index operands. Base-index-displacement operands. General form for memory operands: [base + (scale*index) + displacement]. LEA instruction.
- Examples of question types (not a comprehensive list):
  - Write values into a BYTE/WORD/DWORD array, from left to right, using *indexed* operands.
  - Write values into a BYTE/WORD/DWORD array, from left to right, using *indirect* operands.

- Suppose the following .data section starts at address 00405000h.
  ```
  .data
  arr DWORD 1, 2, 3, 4, 5
  ```
  What value will each of the following instructions place into EAX?
  ```
  mov eax, arr
  mov eax, [arr]
  movsz eax, BYTE PTR [arr]
  mov eax, OFFSET arr
  mov eax, LENGTHOF arr
  mov eax, [arr+2]
  lea eax, [arr+2]
  lea eax, [arr + 2*SIZEOF DWORD]
  ```

CHAPTERS 5 & 8 – PROCEDURES
(Lectures 20, 22, 23, 24, 25; Activities 11–12)

- *Section 5.4 (Stack Operations), Lecture 20* – Stack/LIFO data structure. Runtime stack; use of ESP register; push and pop operations; stack applications: saving registers, return address, arguments, local variables. PUSH, POP, PUSHFD, POPFD, PUSHAD, POPAD, PUSHA, POPA instructions and effects on stack.

- *Section 5.5 (Defining and Using Procedures)* – PROC and ENDP directives; labels in procedures vs. global labels; documenting procedures. CALL and RET instructions and effects on stack; nested procedure calls; passing register arguments to procedures. Saving and restoring registers; exception for returning values in EAX. Recursion.

- *Section 8.2 (Stack Frames)* – Stack frame/activation record; steps to create a stack frame. Use of EBP. Stack parameters: contrast with register parameters; passing by value vs. by reference; passing arrays. Accessing stack parameters: prologue and epilogue, use of base-offset addressing; explicit stack parameters; cleaning up the stack; C calling convention; STDCALL calling convention; passing 8- and 16-bit arguments on the stack; passing multiword arguments; saving and restoring registers. Local variables; using EQU to define symbolic constants for arguments and locals; accessing reference parameters; LEA instruction. ENTER and LEAVE instructions.
- Examples of question types (not a comprehensive list):
  - Write a simple procedure that uses register parameters or stack parameters.
  - Write a simple procedure that uses the STDCALL or C calling convention.
  - Create or destroy a stack frame using enter/leave.
  - Create or destroy a stack frame *without* using enter/leave.
  - Allocate space in a stack from for local variables, and store data there.
  - As in Activity 11 #1, given an "abuse" of call/ret/push/pop, identify what control flow or what output will result.
  - Given a simple program will push/pop/call/ret, identify what values will be on the stack at a particular point in the program's execution.
  - Given a procedure and a call to that procedure, draw the stack frame that results.
  - Given the value of ESP and/or EBP and a screenshot of the Visual Studio Memory window, identify some information about the stack frame (e.g., what was the value of EBP when it was pushed, or what is the return address to the calling function).
- You do **not** need to remember the order in which PUSHAD/POPAD pushes/pops registers.

INTEGER MULTIPLICATION & DIVISION
(Lecture 23)

- *Section 7.4 (Multiplication and Division Instructions)* – MUL (unsigned). IMUL (signed): one-, two-, and three-operand formats; use of two- and three-operand forms for unsigned multiplication. Necessity of sign extension for signed division; CDQ. IDIV (signed); divide overflow. Implementing arithmetic expressions. (32-bit forms only.)

BITWISE OPERATIONS
(Lectures 27, 28, 29, 30; Activities 13–15)

- *Boolean Instructions – Section 6.2* – Bitwise operations: AND, OR, XOR, and NOT instructions. Bit masks. Testing whether particular bits are set/clear; TEST instruction and effects on flags.

- *Bit Sets – Section 6.2.4 and Supplemental* – What is a bit set; when to use (and when to not use) a bit set. Unions, intersections, membership testing using Boolean instructions. Defining symbolic constants for members of a set; using the AND, OR, XOR, NOT, SHL, and SHR operators when defining symbolic constants. (Understand the difference between the AND/OR/etc. instructions and the AND/OR/etc. operators.)

- *Shift and Rotate Instructions – Section 7.2* – Logical vs. arithmetic shifts. Shifts vs. rotations. SHL, SHR, SHLD, SHRD, SAL, SAR, ROL, ROR instructions; effects on the carry flag.

- *Applications of Bitwise Operations – Supplemental and Sections 7.2–7.3* – Computing powers of 2. Right-shifting the sign bit. Isolating bit strings. Testing for odd integer values. Multiplying by powers of 2. Division by powers of 2, rounding toward $-\infty$. Our (non-standard) notations for representing bitwise and shift operations in formulas: $\&$ $|$ $\oplus$ $\neg$ $\ll$ $\gg^u$ $\gg^s$. Zeroing a register with XOR. XOR swap. Manipulating rightmost and trailing bits by adding/subtracting one. Arguments for correctness of formulas involving bitwise operations (examples given: computing minimum and absolute value).
- Examples of question types (not a comprehensive list):
  - Write an expression that will set or clear particular bits, and translate it into a sequence of assembly language instructions.
  - Suppose n is a 32-bit integer. Write an expression (with bitwise and arithmetic operators) that is equal to 0 if n is negative and 1 if n is nonnegative.
  - The sign function $sgn(x)$, which was defined in the lecture on fixed-point arithmetic, can be computed using bitwise and arithmetic operations without a conditional: $sgn(x) = (x \gg^s 31) \mid (-x \gg^u 31)$. Give an argument similar to the arguments given in lecture for the minimum and absolute value computations: describe what each subexpression computes, and then how these combine to compute $sgn(x)$.

FIXED-POINT REPRESENTATION & ARITHMETIC
(Lecture 30; Activity 16)

- *(Supplemental)* Binary numbers with fractional parts: place values, radix point/binary point, converting to/from decimal; what fractions can(not) be represented precisely in a finite number of bits. Rounding binary numbers. What is fixed-point representation; scaling factor, Q*f* and Q*m.f* notation (our definition of it, since it's not completely standard). Range and resolution of a fixed-bit data type. How to add, subtract, multiply, or divide numbers in a given Q*m.f* format; why the multiplication/division operations are more complex, and what the various subexpressions do.

Some exam questions will be similar to questions from the in-class activities and homework assignments.

An ASCII table will be provided if it is necessary for the exam. Calculators will **not** be allowed.