**CSCI 360-1 Midterm Exam Study Guide Fall 2017**

**The CSCI 360-1 Midterm Exam will consist of multiple choice and true/false questions, fill-in-the-blank questions, and short and long answer questions. It will be worth 100 points. The following is provided to you as a guide to assist you in your study. Although every attempt has been made to make this study guide fully comprehensive, there may be material on the exam that was covered in class or in an assignment but is not specifically listed here.**

**Binary and Hexadecimal representation and arithmetic**

Your understanding of the binary and hexadecimal systems is crucial when programming in Assembly language.

* It is advisable to know how to use the hexadecimal and decimal conversion chart on pages 39 and 40 of your yellow cards. **You will not be allowed to use calculators during the exam.**
* Know direct hexadecimal to binary conversion (X'A' = B'1010')
* Be able to compute the two's complement of binary and hexadecimal integers.
* Be able to perform addition and subtraction of binary and hexadecimal numbers.
* Be able to convert hexadecimal and binary numbers to decimal, and vice-versa.
* Be familiar with the concept of overflow. Know how to detect conditions that result in overflow.

**Memory Organization**

It is important to know how main memory is organized. In some cases you must organize and access your memory according to specific rules, otherwise your programs will not function correctly.

* Know the basic units of memory and addressibility: 1 Double word = 2 fullwords = 8 bytes = 64 bits.
* Know when data must align to a fullword boundary.
* Know how to use and the difference of the DC and DS statements. Know which data units will force alignment to which boundaries.
* Know what "slack bytes" are and how they might be created.
* Know what will cause, and how to debug a specification exception (S0C6).

**Dumps**

When programming and debugging on the machine level, the truth is always in the dump. Do not rely on what the assembler was 'supposed to do'. The only way to get to the root of a problem is to look in the dump. It is important then that you are familiar with the format of the dump and how to read it.

* Know the basic format. How to find the contents of memory locations and the registers.
* By using the listing, be able to locate instructions and data encoded by the assembler.
* Be able to interpret the Program Status Word (PSW).
* Know how to locate and compute the interrupt code, instruction length code, condition code, and the address of the NEXT instruction, all in the PSW.
* Given only the PSW, know how to find an ABENDing instruction.

**Addressing**One of the luxuries we get from the assembler is the use of labels, which greatly simplify the task of referencing memory throughout our program. However, it is important to know how the addresses are stored and determined in our instructions. There are no labels in the dumps, so we must be able to find an address given its base-index-displacement values.

* Given a base register, an index register, and a displacement, be able to compute the absolute address.
* Know how the assembler uses the USING statement to determine the register to use as a base register.
* Know what causes, and how to debug protection exceptions (S0C4) and addressing exceptions (S0C5).

**Instruction Encoding**

The basic job of the assembler is to take your instructions and encode them into their machine-language form. You should be familair with how the assembler does this, and especially how to do the reverse. That is, given the machine-language code, how to convert back to the explicit assembler. This is important when debugging from a dump.

* Know the basic RR and RX instruction formats.
* Know how to encode and decode the instructions we have covered.
* Know what causes, and how to debug, an operation exception (S0C1).

**Important Instructions**

Of course, all assembly language programmers must be familiar with the instruction set of the computer they are programming.

* Be familair with the operation of all the instructions we have covered: A, AR, S, SR, L, LR, ST, C, CR, BC, BCR, M, MR, D, DR, LA, LTR, LPR, LNR, LCR, MVI, MVC, CLC, STM, LM, BAL, BALR, BCT, BCTR
* Know how the multiply and divide instruction us an even/odd register pair to perform their functions.
* Do not forget the eXtended psuedo-instructions: XREAD, XPRNT, XDECO, XDECI.
* Know which instructions will set the condition code. Remember: the X-instructions will not be in your yellow cards!
* Be able to use the branch on condition instructions, including their equivalent extended mnemonics discussed in class.
* Know how to use literals. Do not forget an LTORG!
* Know how MVC can cause destructive overlap if used incorrectly.

**Basic Program Structure**

You are expected to know how to implement, in assembler, the basic structured programing constructs of higher-level languages. These include:

* Simple IF's (conditional statements).
* A top-driven loop.
* Know how to read and process an indefinite number of records from a file.

**EBCDIC**

If given a character, such as the capital letter A or a comma, find the equivalent EBCDIC encoding in the Yellow Card. And the vice versa. Given an EBCDIC code such as X'C1', find the character that it represents in storage. In this case, the capital letter A. Know the carriage control characters we use.

**Passing Parameters**

Know the Assembler standards that dictate how parameters are passed to a subroutine.

* Know how to use address constants, or "a-cons", such as DC A(TAXRATE), to create a parameter list.
* Know how to point the correct register (register 1) to the parameter list prior to branching to a subroutine.
* Know how to use BAL to branch to the subroutine.