## CHAPTER 1: INTRODUCTION

1. C data types
   a. Width
      i. Usually: char (8), short (16), long (32), long long (64)
      ii. Always: int8_t (8), int16_t (16), int32_t (32), int64_t (64)
      iii. Always: uint8_t (8), uint16_t (16), uint32_t (32), uint64_t (64)
   b. Range of N-bit binary numbers:
      i. Unsigned: 0 to $2^{N}$-1
      ii. 2's complement: -$2^{N-1}$ to +$2^{N-1}$-1

## CHAPTER 2: BINARY NUMBER SYSTEMS

1. Unsigned
   a. Converting Base R to Base 10: Polynomial Evaluation
   b. Converting Base 10 to Base R: Repeated Multiplication/Division (by R)
   c. Converting between two bases, neither of which is decimal
   d. Representation errors (e.g., representing $0.1_{10}$ in binary)

2. 2’s complement
   a. Changing sign (two methods):
      i. Invert all the bits and add 1
      ii. Work right to left, copy through first 1, then copy inverse
   b. Converting to decimal (two methods):
      i. If negative, expand to N bits, form 2’s complement, polynomial evaluation, add minus sign
      ii. Polynomial evaluation, but change the sign of the most significant term
3. Conversion by inspection when bases are power-related (e.g., $2^4 = 16$)
   a. Be able to create and use (for example) a hex to binary table

## CHAPTFR 3: WRITING FUNCTIONS IN ASSEMBLY

1. Function call (BL) and return (BX LR)
2. Functions that call other functions must preserve and restore Link Register (LR)
3. Accessing parameters inside the function
   a. parameters assigned to R0-R3 in left-to-right order
   b. 8, 16, and 32-bit parameters allocated a single register
   c. 64-bit parameters allocated a sequential register pair
      (LS Half in 1st register, MS Half in 2nd)
4. Preparing the return value
   a. 8, 16, 32-bit results returned as 32-bit values in R0
   b. 64-bit values returned in R1.R0 (R1=MS Half, R0=MS Half)
5. Register usage conventions
   a. Functions may modify R0-R3 without preserving their content
   b. Functions must preserve content of R4-R12

## CHAPTER 4: COPYING DATA

1. Copying constants into registers (e.g., LDR R0,=0)
2. Copying data from memory to registers
   a. 8-bit unsigned operands: LDRB
   b. 8-bit signed operands: LDRSB
   c. 16-bit unsigned operands: LDRH
   d. 16-bit signed operands: LDRSH
   e. 32-bit operands: LDR

- f. 64-bit operands: LDRD

3. Copying data from one register to another (MOV)
4. Copying data from registers to memory
    - a. 8-bit operands: STRB
    - b. 16-bit operands: STRH
    - c. 32-bit operands: STR
    - d. 64-bit operands: STRD
5. Addressing modes
    - a. Immediate Offset (E.g., [R0] or [R0,4])
    - b. Register Offset (E.g., [R0,R1] or [R0,R1,LSL 2]
    - c. Pre-Indexed (e.g., [R0,4]!)
    - d. Post-Indexed (e.g., [R0],4)
6. Pointers and arrays
    - a. Relationship between address calculation and pointer arithmetic
        - i. pointer + constant → Use Immediate Offset
        - ii. pointer + integer variable → use register offset (need LSL #bytes/object)
    - b. Relationship between address calculation and subscripting with constant subscript
        - i. Use Immediate Offset (e.g., [R0,4], where
        - ii. Register holds starting address of the array
        - iii. Constant is subscript x #bytes/element
    - c. Relationship between address calculation and subscripting with variable subscript
        - i. Use Register Offset (e.g., [R0,R1 LSL 2], where
        - ii. First register holds starting address of the array
        - iii. Second register holds the subscript
        - iv. LSL constant = #bytes per array element
7. Pointers and structures
    - a. Use Immediate Offset (e.g., [R0,4], where
    - b. register holds starting address of the structure
    - c. Constant is address offset within structure

## CHAPTER 5: INTEGER ARITHMETIC

1. Binary Addition and Subtraction: Same for unsigned and 2's complement.
2. Binary Addition (be able to create and use the addition table, resulting in carries and sum bits)
3. Binary Subtraction (two methods):
    - a. Form 2's complement of subtrahend (on bottom) and add, or
    - b. Invert the subtrahend, add operands, add 1 (I.e., a - b = a + ~b + 1)
4. Rollover (all 0's ←→ all 1's) versus overflow (arithmetic behavior: result exceeds range)
5. Unsigned Overflow during Addition: Carry-Out = 1
6. Unsigned Overflow during Subtraction: Borrow-Out=1 (Carry-Out=0)
7. 2's complement Overflow during addition or subtraction
    - a. Overflow impossible if adding (subtracting) operands w/different (same) signs
    - b. Overflow if carry/borrow-in and carry/borrow-out of MS bit differ
8. Condition flags: Unaffected unless "S" appended to (most) instruction
9. Multiplication by $2^N$: Shift left by N bits
10. Multiplication of two N-bit operands produces 2N bits of product
11. 2's complement vs. Unsigned multiplication (Only MS half of double-length product differs)
    - a. All multiply instructions: Operands may ONLY be registers
    - b. Single-length products: Same instruction works for unsigned and 2's complement (MUL)
    - c. Double-length Unsigned product: UMULL
    - d. Double-length Signed product: SMULL
    - e. (Most) flags unaffected by multiply instructions
    - f. Overflow
        - i. Multiply instructions do not affect the overflow (V) flag
        - ii. Not possible if ***<u>using</u>*** the entire double-length product
        - iii. Producing a single-length product: Detection is effectively impossible

iv. Producing a double-length product, but using only LS Half:
    1. Unsigned: MS half is non-zero
    2. dSigned: MS half is not a sign-extension of the LS half.

12. Divide Instructions:
    a. All divide instructions: Operands may ONLY be registers
    b. Single-length unsigned quotient: UDIV
    c. Single-length signed quotient: SDIV
    d. (Most) flags unaffected by divide instructions
    e. There is no divide instruction that uses a double-length dividend
13. Computing the remainder (MLS) and modulus
14. Saturating Arithmetic (I won't test you on this)

# CHAPTER 6: MAKING DECISIONS AND WRITING LOOPS

1. Compare and test instructions (focus on CMP)
    a. Computes (but discards) the difference of first less second operand
    b. Characteristics of difference recorded in the flags (NVCZ)
    c. First operand must be a register
    d. Second operand may be a small constant, a register, or a shifted register
    e. Subsequent conditional branch condition is a left-to-right reference to CMP operands
2. Conditional branch instructions
    a. "B" followed by a two-character condition code.
    b. Condition TRUE: Leaves current instruction sequence and branches to target label
    c. Condition FALSE: Continues sequentially to the following instruction if condition is NOT satisfied
3. if-then-else sequences
    a. Conversion to assembly requires using testing for opposite condition
    b. "Then statements" must be immediately followed by an unconditional branch around the "Else statements".
4. comparing 64-bit integers
    a. Best way: Compute 64-bit difference, setting flags ("S") and discarding difference.
5. IT blocks
    a. Controls 1-4 instructions (the "IT Block") immediately following the IT instruction
    b. IT operand is a condition code; determines whether next instruction is executed.
    c. 2nd, 3rd, and 4th instructions determined by appending one or more "T" (Then) or "E" (Else) to "IT"
    d. Each instruction in the IT Block must append the condition that enables it.
    e. Can't branch into an IT Block.
    f. Only last instruction of an IT Block can be a branch
    g. IT instructions that affect the flags do not affect instructions in the IT Block
6. Compound conditionals: Please see document, "Converting from C to Assembly", found on our Camino page under "Learning Resources" for Chapter 6.
7. Writing loops: See the Chapter 6 slides, available as videos at www.cse.scu.edu/~dlewis/book3