CS-280 Detailed Lesson Objectives

Dr. E. Durant <durant@msoe.edu> – Updated March 31, 2005

1. Lesson 1: Introduction and number systems
    a. Perform unsigned and signed (2's complement) binary arithmetic.
    b. Convert between binary, decimal, and hexadecimal numbers.
    c. List several examples of embedded computer systems.
    d. Define "microcontroller" and contrast it with "microprocessor".
2. Lesson 2: Microcontroller components, tool introduction
    a. State the purpose of the following computer parts: clock circuit, control unit, ALU, memory (RAM, ROM, EEPROM/flash/etc.), I/O.
    b. Describe the key components of a memory interface (address, R/~W, ~enable, data)
    c. Describe the main assembly language development tools and their position in the toolchain (compiler, assembler, linker, simulator, downloader, and debugger).
3. Lesson 3: Tool chain and programmer's model (Registers)
    a. Describe the HC11 register set (A, B, (D), X, Y, SP, PC, CCR:{NZVC})
    b. Use a memory map to describe the location of code, mutable data, and constant data.
    c. State the basic memory map for the HC11 system used in CS-280.
4. Lesson 4: Addressing, CCR
    a. Apply each of the HC11 addressing modes with various instructions (immediate, direct, extended, indexed (X and Y), inherent, and relative).
    b. Translate short instruction sequences to machine code using the Motorola Reference Guide (MRG).
    c. Calculate the cycle count and execution time for short instruction sequences using the MRG.
5. Lesson 5: Tools: Assembler, linker, simulator (Flashin' LED Demo)
    a. Use the appropriate section for code and various types of data.
    b. Allocate memory (initialized and uninitialized) of byte and word size.
    c. Allocate strings.
6. Lesson 6: Instruction set introduction: transfers, load/store, add/subtract (Example program)
    a. Use the transfer (register/register, stack, clear), load/store, increment/decrement, and add/subtract instructions to implement short programs.
    b. Contrast big endian with little endian representations.
    c. Analyze (determine memory and register (including CCR) contents) short assembly language programs.
7. Lesson 7: Data members and member functions in assembly
    a. Lay out an object data structure.
    b. Allocate memory for an array of objects.
    c. Access objects & data members in code.
    d. Pass 1 or 2 arguments to functions.

8. Lesson 8: Display / Instruction set: flags, multiply/divide, shift/rotate
    a. Send lines of text to the 16×2 display.
    b. Explain the use of the 3 basic display subroutines.
    c. Test (compare with 0) values using the TST* instructions.
    d. Set/clear the carry (C) and overflow (V) flags directly.
    e. Multiply and divide numbers with an awareness of execution time and the difference between integer and fractional division.
    f. Apply logical and arithmetic shifts to numbers and recognize the equivalence of arithmetic shifts to multiplication and division by powers of 2.
    g. Apply the rotate (through carry) instructions.
9. Lesson 9: Instruction set; bit/logic, branches
    a. Be able to use bit test operations (BIT*) in assembly language programs.
    b. Use AND, OR, and EOR instructions to implement bit manipulations.
    c. State the basic CCR manipulations performed by bit, comparison, and arithmetic operations and be able to determine all such manipulations by referring to the MRG.
    d. Use the JMP and BRA instructions and explain the primary purpose of each.
    e. Make appropriate use of conditional (signed, unsigned, etc.) branches.
10. Lesson 10: Decimal arithmetic / keypad
    a. Explain the use of binary-coded decimal (BCD) notation.
    b. Use the DAA instruction to perform BCD addition and subtraction.
    c. Output 1-4 character BCD and hex numbers to the display.
    d. Explain the use of pull-up and pull-down resistors on binary inputs.
    e. Describe the operation of the matrix keypad.
    f. Explain the operation of an algorithm to detect a key press from the matrix keypad and determine its value.
    g. Wire output ports to either source or sink current, allowing LEDs to be used in both active high and active low configurations.
11. Lesson 11: Instruction set: subroutines, stacks
    a. Explain the use of the stack for temporary local storage, general parameter passing, and subroutine context.
    b. Use the PSH/PUL A/B/X/Y instructions for temporary local storage, copying local variables, swapping values, etc.
    c. Describe the initialization of the SP and how the stack grows and shrinks.
    d. Use the INS/DES instructions for stack deallocation/allocation.
    e. Use the TSX/Y instructions to address data relative to a stack frame and explain the use of the T[XY]S instructions.
    f. Demonstrate the SP and PC effects of the BSR, JSR, and RTS instructions.
    g. Recognize scopes over which the stack must be balanced (usually and primarily function scope).

12. Lesson 12: C++ review/compiler (Demo)
    a. Define cross compiler (cross assembler, etc.) and related terms (target, host).
    b. Map the inherent C++ data types to their implementation with GCC for the HC11.
    c. Use the C++ bitwise operators and the assignment versions to efficiently perform bit operations from the high level language (~, |, &, ^, <<, >>).
    d. Explain the necessity and operation of the C++ volatile qualifier for memory mapped I/O, etc.
    e. Access I/O ports using one of the two standard methods (individual variables or I/O array), but recognize the use of either method.
    f. Explain the basic operation of gcc as a compile/assemble/link driver and as a proper compiler (general operation, not command line argument details).
13. Lesson 13: ASM vs. C++ structure, argument passing, inline asm
    a. Pass any number of 8- and 16-bit arguments to functions with C++ interfaces that are implemented in assembly language.
    b. Return 8- and 16-bit values from ASM functions with C++ interfaces.
    c. Explain situations in which mixing C++ and ASM is the preferred implementation method.
    d. Apply the 2 key register save disciplines for functions (caller stacking of needed state and function stacking of used registers).
    e. Use the register save discipline of gcc for the HC11 to correctly and most efficiently use registers in ASM function implementations with C++ interfaces.
    f. Identify and use, when appropriate, the 3 methods of assembly argument passing: globals, register, and stack. State the key limitations of each method.
    g. Pass arguments by reference (pointer) (general use, not compiler specifics).
    h. Use __asm for short in-line assembly sequences in C++ code.
    i. Explain the meaning and use of "extern" on declarations (location of .global definition and label) of both functions and variables.
14. Lesson 14: Parallel I/O
    a. Using the RG, note which HC11 pins are available for general purpose input, output, and I/O when not being used for their special functions.
    b. Describe the problem that the MC68HC24 port replacement unit (PRU) solves and give a brief overview of the solution.
    c. Use and describe simple strobbed input on the HC11, making proper use of DDRC, PORTCL, all related PIOC bits (except STAI – covered later), and the STRB pin.
    d. Use and describe simple strobbed output on the HC11, making proper use of PORTB, all related PIOC bits, and the STRA pin.
    e. State the main benefit of strobbed I/O over non-strobbed I/O.
15. Lesson 15: Parallel I/O: Handshaking (ASM example: input handshaking with ADC0808)
    a. Use and describe input and output handshaking on the HC11 using DDRC, PORTCL, all related PIOC bits (except STAI – covered later), and the STRA and STRB pins.
    b. Explain the different uses of the 2 methods of accessing data on port C: PORTC and PORTCL.
    c. State the main benefit of handshaking I/O over strobbed I/O.

16. Lesson 16: A/D
    a. Set up and use the A/D system to acquire digital representations of analog voltages connected to PORTE pins. Specifically, use OPTION:ADPU and all ADCTL fields.
    b. State the basic timing issues of the A/D system and describe the operation of ADCTL:CCF.
    c. Calculate the correspondence between A/D bits and a particular analog voltage.
17. Lesson 17: A/D internals and integer scaling
    a. Use shifts and IDIV to scale the sum of *N* of 8-bit A/D conversions to an integer in an arbitrary range (e.g., representing centivolts) keeping maximum accuracy within the limits of an 8x8 multiply and a 16x16 divide.
18. Lesson 18: Interrupts
    a. State the advantages of interrupts over polling.
    b. Define interrupt latency.
    c. Explain the role of CCR:I.
    d. Describe what happens when an HC11 interrupt occurs, especially regarding the stack, CCR:I, and PC.
    e. Explain the use of the interrupt vector and set up RAM vectors for (both special and normal mode) locating ISRs.
    f. Set up jump vectors (a.k.a., pseudovector) in conjunction with a ROM vector for locating ISRs.
    g. Write assembly and C++ code to enable and handle general interrupts.
    h. Use the external interrupt (~IRQ) source.
    i. Explain the use of the SWI instruction in debugging.
19. Lesson 19: Interrupts: real-time interrupt, input capture/output compare
    a. Explain the basic use of the free running counter (TCNT) and the use of the timer prescaler bits (TMSK2:PR).
    b. Perform a "16-bit atomic read" of TCNT using both C++ and ASM and explain why atomic reads/write must be used to read and/or write various 16-bit registers.
    c. Use the real-time interrupt (RTI) to perform a task at certain regular, fixed time intervals using the PACTL:RTR bits.
    d. Use the output compare interrupts to perform tasks at time intervals that are accurately and flexibly defined by the programmer.
    e. Explain the role of input capture for accurately recording when an external event occurred asynchronously to the program execution.
20. Lesson 20: Interrupts: pulse accumulator
    a. Use the pulse accumulator function to count and respond to external events (either each one or a predefined number) using interrupts.
    b. Explain the role of the gated timer function of the pulse accumulator system.
    c. Use STAI (interrupt, source = IRQ) as an alternative to STAF polling for strobed input and handshaking.