CSC 230 Assignment 2
Fall 2015

This assignment has two parts:

1) **Written Part:** Complete the Assignment 2 problem set on connex (in the Tests & Quizzes section). The problem set is to be completed online and, where calculations must be done, only requires input of the final answer. You are required to answer these questions without electronic tools. Treat these questions as if they were conducted during an exam. **Complete it on conneX before 9am on October 13, 2015.**

2) **Programming Part: Submit only your code (*.asm) file on conneX. Be sure to include your name and student number. Submit it before noon October 22, 2015 on conneX**

General notes.

- A word-addressable system (or portion of) implies that the smallest location which can be addressed and whose content can be transferred is exactly one word, whatever size the word is.
- A byte-addressable system (or portion of) implies that the smallest location which can be addressed and whose content can be transferred is exactly one byte.
- Reminder: 1K=1,024=2^{10}, 1M=1,024K=2^{20} and 1G=1,024M=2^{30}
- Where ever possible give answers power of 2.

Assignment 2: Written Part [35]

Question 1 [8]

A computer system has a RAM containing 16K bytes, each of which needs its own distinct address. In addition, it has 16 peripherals and they each require 2^4 distinct addresses in order to interface properly.

a) [1 mark] **(format flexible)** Assuming the I/O is also mapped as memory (i.e., peripherals are also accessed just like a memory, see Slide 55 of Introduction Class notes), how many distinct addresses in total are necessary in this system? Write the total number on the line in the expression below. (Express your answer as a sum of powers)

$$2^{\square} < \underline{16K+16*2^4=2^4 2^{10}+2^8} < 2^{\square}$$

total number of addresses here

b) [2 marks] **(machine marked, 2 decimal digits only for each of 2 answers)** Now complete the expression above writing the containing exponents in the boxes (for example, if the results were 18, you would write 4 in the left box and 5 in the right box.)

$2^{14} <$ _____ $< 2^{15}$

c) [1 mark] **(machine marked, 2 decimal digits only)** Given that an address bus for this system must be able to carry all the needed values for the addresses, how many lines (bus width) does the address bus require? **15**

d) [2 marks] Now let us assume that peripherals are mapped separately using a separate I/O bus. In this case what should be the I/O address bus width (lines) and what should be the memory bus width (lines)? **I/O 8 lines; memory 14 lines**

e) [2 marks] **(format flexible)** In your opinion, what would be one advantage and one disadvantage of these bus structures of connecting peripherals (i.e, memory mapped I/O and port I/O)? **advantage of memory mapped I/O: simple; advantage of separate I/O: independent pathways for data from CPU to memory & I/O, implies neither has to wait for the other.**

Question 2 [2] (format flexible)

A program compiled for a SPARC ISA (Instruction Set Architecture) writes the 32-bit unsigned integer 0xABCDEF01 to a file, and reads it back correctly. The same program compiled for a Pentium ISA also works correctly. However, when the file is transferred between machines, the program incorrectly reads the integer from the file as 0x 01EFCDAB. What is going wrong? **Sparc is big endean while Pentium is little. Internally within a machine it doesn't matter as the machine uses same mechanism but between machines the meaning of LSB and MSB must be the same. Otherwise machines will interpret them differently as shown above.**

Question 3 [5]

Let us assume that you are designing a brand new video card that displays full HD resolution of 1920x1080 pixels. Each pixel is represented by three colors (RGB) and that 8 bits are used to represent the intensity of each color (e.g., a 0 means the color is off and all 1s in a byte means maximum intensity). A frame buffer (memory) is generally used to store these pixels which in turn are displayed on a screen.

a) [1 mark] How much frame buffer (memory) in bytes would be needed to display one full resolution screen? **1920x1080x3x8=49766400 bits = 6220800 bytes = 6075KB**

b) [1 mark] If memory is byte addressable, how many address bits will be needed to address this memory? **23**

c) [1 mark] If memory is word addressable (assume word is 24 bits), how many address bits will be needed to address this buffer? **6075K Bytes / 3 bytes/word = 2025 KWords**

d) [1 mark] To give a depth effect, frame buffers also use one extra byte per pixel to represent a depth ("Z" axis) value, also referred to as "Z buffer". If we use one Z value for each pixel, then 32bits would be needed for each pixel (32bpp). How much frame buffer in bytes would be needed in this case? **1920x1080x32 bits = 66355200 bits, Thus 66355200/8 = 8294400 bytes**

e) [1 mark] If memory is byte addressable, how many address bits will be now needed to address this memory that includes Z value? **= 8294400 = 8100 Kbytes .. Thus 23 address bits**

Question 4 [5]

Recall from our class lectures that computer is a stored program processor which always fetches instructions from the memory, decodes and executes them that is known as "Fetch-Decode-Execute-Store" cycle. Memories are organized as a linear array of locations with each location having a specific address and each location storing specific content (a byte or word). To read or write to a specific location, CPU first issues the address of the location on address bus to access and then gets or puts the data into the location using the data bus along with other control signals on control bus (read or write). A specific timing (order has to be followed) to read or write using the address/data/control buses. Study the slides (6 to 14 of Class 3) to answer the following:

a) [1 mark] What operation are we performing on the memory during the fetch phase? **Get instruction from memory**
   What is the order of buses used? **1) address bus 2) control bus 3) data bus**
   What is the control signal used? **Read signal.**
b) [2 marks] "Store" phase is used in order to store any resultant data in the execution phase.
   What will be order of buses? **Address Bus and Data Bus; followed by Control Bus**
   Write the steps of detailed CPU activity (follow the steps of fetch phase and modify where necessary) **address → MAR; MAR → Address Bus; data → MDR; MDR → Data Bus; write signal → Control Bus; Wait for memory**
c) [1 mark] Would the decode phase involve any memory accesses? **No** Reason. **Docode unit is a hardware unit contained within the CPU. It simply converts the machine language instruction.**
d) [1 mark] In what phases of CPU activity is the memory access needed? **Fetch; Fetch further data (if needed); execute (when it stores).**

Question 5 [10]

Assembly language uses shorthand notation (mnemonics) to write instructions but ultimately the code will be all in binary. As a programmer, you would use an assembler to generate the binary (machine) code from the assembly language. However, at times you may only have access to the machine code and need to understand (decipher, reverse engineer) the sequence of instructions. For this question, our focus is a very small set of AVR instructions (**`mov, ldi, add, st`**) as shown on slide 27 and 29 of the Class 3 slide set.

a) [1 mark] What registers can we use for the **`mov`** instruction? **(any of the 32 registers)** What can be used for **`ldi`** instruction? **(only the upper 16 registers)**
b) [1 mark] What is the **X** used in **`st`** instruction? **(X is a pointer to an address.)** What does it comprise of? **X is the register pair R27:R26**

c) [1 mark] Is `ldi R0,310` valid instruction? **No** Reason. **(LDI takes only upper registers from R16 onwards. You cannot use R0)**

d) [1 mark] Encode `mov R11,R21` to machine code
`mov Rd,Rr  0010 11rd dddd rrrr`
`mov R11,R21  0010 1110 1011 0101`

e) [1 mark] Encode `ldi R17,129` to machine code **1110 1000 0001 0001**

f) [4 mark] Write detailed CPU activity steps (see slides 28 to 30) for the above instruction `ldi R17,129.` Assume that this instruction is at an address 0xF000.
Fetch: PC -> MAR; MAR -> address bus; control (read) -> control bus; wait for memory; data -> data bus; data bus -> MDR; MDR -> IR; PC -> ALU; ALU (inc (PC)); ALU (result) -> PC
Decode: IR ->Decoder hardware in control unit examines instruction, finds opcode & operands
Fetch further data, if required: MAR, MDR and PC are possibly in use
Execute: May involve ALU, plus reads and writes to memory

g) [1 mark] Translate `1001 0011 1100 1100` to Assembly instruction
`st X,Rr  1001 001r rrrr 1100 => st X, R28`

Question 6 [5]

Processors use various addressing modes to access program and data memory. Please refer to the class slides as well as the AVR Assembly Language document to answer the following questions:

a) What addressing mode is used by the `ldi` instruction? **(Immediate)**
b) What addressing mode is used by the `add` instruction? **(Register Direct)**
c) What addressing mode is used by the `ld` instruction? **(Data Indirect Addressing)**
d) What addressing mode is used by the `st` instruction? **(Data Indirect Addressing)**
e) What addressing mode is used by the `rjmp` instruction? **(Relative Addressing)**

Assignment 2: Programming Part [25]

Please note for all programing assignments, ***you must include your name and student number.*** Make sure the submitted work is yours and not someone else.

Question 7 [5]

You are given two inputs A=10 and B=25. Write a simple AVR assembly program to perform various arithmetic (subtract and add) and logic operations (AND, OR, shift left once A and shift right once B) as a sequence of operations. Assume that the inputs can be directly loaded into registers using `ldi` instruction (define R16 to be A and R17 to be B). Each of the results (e.g., A+B, A-B, A.B etc) needs to be temporarily stored and you can use other unused registers for this purpose. Your goal is to minimize number of

instructions. The sequence of operations and the registers where the results are stored is given by: A+B (=R4), A-B (=R5), A.B(=R6), A|B(=R7), A<<1(=R8) and B>>1(=R9).

Please submit your code for this question as "a2_question7.asm" on conneX.

Question 8 [20]

This question relates to matrix manipulations similar to the AVR assembly code discussed in Class 4. The following is an AVR assembly code to add two 2x2 matrices. The first one is initialized with values 1,2,3,4 and the second one with 10,11,12,13. They both are added and then put into the third matrix of size 2x2.

```
.device ATmega2560

;this program adds two matrices of size 2x2
;first we initialize matrices (matrix1 and matrix2) and then store the
result in matrix

;first get a pointer to matrix1 and put it in Register X: R27:R26
      ldi R26, low(matrix1); get the low order byte of the address
      ldi R27, high(matrix1); get the higher order byte of the address
      ldi R16, 1; starting value of matrix1
      ldi R17, 4; number of values to initialize is matrix size 2x2
;like for loop in C (do the following as long as R17>0)
loop:
      st X+, r16;
      inc R16; next value to initialize
      dec R17; decrement count R17
      breq nextinit; initialized first matrix. Go to next one if R17=0
      rjmp loop

;now initialize second matrix
nextinit:
      ldi R26, low(matrix2); get the low order byte of the address
      ldi r27, high(matrix2); get the higher order byte of the address
      ldi R16, 10; starting value of matrix2
      ldi R17, 4; number of values to initialize
;like for loop in C (do the following as long as R17>0)
loop1:
      st X+, r16;
      inc R16; next value to initialize
      dec R17;
      breq add_matrix; initialized second matrix. go to add them
      rjmp loop1

add_matrix:
      ;get the address of matrix1 into X (R27:R26) register
      ldi R26, low(matrix1); get the low order byte of the address
      ldi R27, high(matrix1); get the higher order byte of the address
      ;get the address of matrix2 into Y (R29:R28) register
      ldi R28, low(matrix2); get the low order byte of the address
```

```
      ldi R29, high(matrix2); get the higher order byte of the address
      ;get the address of matrix into Z (R31:R30) register
      ldi R30, low(matrix); get the low order byte of the address
      ldi R31, high(matrix); get the higher order byte of the address

      ;add matrices in a loop
      ldi R17, 4; number of values to initialize
loop2:
      ld R0,X+;get the element from matrix1 into R0
      ld R1,Y+;get the element from matrix2 into R1
      add R0, R1; R0=R0+R1 add the elements
      st Z+,R0; store the result in appropriate location
      dec R17;
      breq done; we are done adding matrices
      rjmp loop2; if not continue

done:
      rjmp done; infinite loop

.dseg
.org 0x000200 ; ATmega2560 data memory starts at 0x000200
matrix1: .byte 4; defines 4 bytes of storage for matrix1
matrix2: .byte 4; defines 4 bytes of storage for matrix2
matrix: .byte 4; defines 4 bytes of storage for matrix
```

Your goal is to understand the above program, modify and expand it:

a) Define four matrices **mymatrix1, mymatrix2, mymatrix3, mymatrix4**
b) All matrices are of same size 5x5
c) **mymatrix1** is initialized with integers in increasing order (in steps of 1) with values starting from 45
d) **mymatrix2** is initialized with integers in increasing order (in steps of 1) starting from 10
e) Write code to perform matrix addition of **mymatrix1** and **mymatrix2** with the result stored in **mymatrix3**
f) Write code to perform matrix subtraction of **mymatrix1** and **mymatrix2** with the result stored in **mymatrix4**

Please submit your code for this question as "a2_question8.asm"

Sample values are:

	50	51	52	53	54		1	2	3	4	5
	55	56	57	58	59		6	7	8	9	10
mymatrix1 =	60	61	62	63	64	mymatrix2 =	11	12	13	14	15
	65	66	67	68	69		16	17	18	19	20
	70	71	72	73	74		21	22	23	24	25