Department of Electrical and Computer Engineering
University of Alabama in Huntsville

CPE 323 – Introduction to Embedded Computer Systems
Midterm Exam Keys

Instructor: Dr. Aleksandar Milenkovic

Date: February 20, 2013

Place: EB 207

Time: 3:55 PM – 5:15 PM

Note: Work should be performed systematically and neatly. This exam is closed books and closed neighbour(s). Allowable items include exam, pencils, straight edge, calculator, and materials distributed by the instructor. Bonus questions are optional. Best wishes.

Question	Points	Score
1	10+2	
2	30	
3	20+3	
4	20	
5	20	
Sum	100+5	

Please print in capitals:

Last name:______________________________

First name: ______________________________

1. (10 points + 2 bonus points) Misc, MSP430

Circle the correct answer for A-E and type in number for F.

1.A. ([✓ True] | False) (2 points) Assembly language directive “DS16 3” allocates 3 words in memory.

1.B. ([✓ True] | False) (2 points) Register R1 serves as the stack pointer (SP).

1.C. (True | [✓ False]) (2 points) The stack grows toward higher addresses in memory.

1.D. ([✓ True] | False) (2 points) The address range of a 2 KB block of data placed in memory at the address 0x0400 is [0x0400 – 0x0BFF].

1.E. (True | [✓ False]) (2 points) Instruction ADD R7, 0(R8) requires one 16-bit word to be encoded.

1.F. (bonus, 2 points) How many memory operations (read from memory and write to memory) will be performed during execution of the instruction ADD.W 0(R7), &0x430.

3 to fetch instruction, 2 to fetch operands, and 1 to write the result => 6 memory operations

2. (30 points) Assembler (Directives, Instructions, Addressing Modes)

2.A. (10 points) Show the word-wide HEXADECIMAL content of known memory locations that correspond to the following sequence of assembler directives. ASCII code for character ‘A’ is 65 (decimal), and for character ‘0’ is 48 decimal. Note: suffix q denotes an octal number.

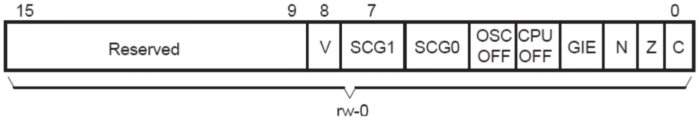
```
            ORG 0x0400
MR          DS8 4
            ORG 0xC000
CAB         DC8 011q, -8, '3', '1'
            EVEN
CBS         DC8 "CBD"
            EVEN
CCL         DC32 -2
```

Label	Address [hex]	Memory[15:0] [hex]
MR	0x0400	0x????
	0x0402	0x????
	...	
CAB	0xC000	0xF809
	0xC002	0x3133
CBS	0xC004	0x4243
	0xC006	0x0044
CCL	0xC008	0xFFFE
	0xC00A	0xFFFF

011q = 00.001.011 => 0x09

2.B. (20 points) Consider the following instructions given in the table below. For each instruction determine addressing modes of the source and destination operands and the result of the operation. Fill in the empty cells in the table. The initial content of memory is given in the table. The initial value of the registers R2, R5, and R6 are as follows: SR=R2=0x0003 (V=0, N=0, Z=1, C=1), R5=0x0803, R6=0x0808. Assume that the starting conditions are the same for each instruction (i.e., always start from the initial conditions in memory and given register values).

Note: Format of the status register (R2) is as follows.



Label	Address [hex]	Memory[15:0] [hex]
	0x0800	0x0504
	0x0802	0xFEEE
TONI	0x0804	0xA87E
	0x0806	0x33F4
	0x0808	0xF014
	0x080A	0x2244
EDE	0x080C	0xCDDA
	0x080E	0xEFDD

	Instruction	**Source Addressing Mode**	**Destination Operand Addressing Mode**	**Source Address**	**Dest. Address**	**Result (content of memory location or register)**
(a)	ADD.B &TONI, R5	Absolute	Register	0x0804	-	byte operation, src: M[0804]=0x7E dst/src: 03; dst: 0x7E+03 => R5 = 0x0081 C=0, Z=0, N=1, V=1
(b)	CMP.B #0x44, 2(R6)	Immediate	Indexed	-	0x080A	byte operation (dst – src = dst +not src+1); (do not store result) dst – src = 0x44-0x44 C=1, V=0, N=0, Z=1.
(c)	SWPB TONI	-	Symbolic	-	0x0804	Swap bytes: M[0x0804] = 0x7EA8 Flags are unchanged. (C=1, Z=1, N=0, V=0)
(d)	BIS @R6+, -3(R5)	Autoincrement	Indexed	0x0808	0x0800	src: M[0x0808]=0xF014, dst: M[0x0800] = 0x0504, bitwise logical OR M[0800]=0xF514; Flags are unchanged. V=0, N=0, Z=1. C=1

Notes of setting flags: Instructions that set flags, set N and Z flags as usual. CMP sets the flags in the same manner as the SUB instruction.

3. Analyze assembly program (20 points + 3 bonus points) Consider the following segment of an assembly program.

```
1               MOV     #arr1, R5
2               CLR.B   R7
3               CLR.B   R8
4       lnext:  ADD.B   @R5+, R7
5               CMP     #aend, R5
6               JZ      lexit
7               ADD.B   @R5+, R8
8               CMP     #aend, R5
9               JZ      lexit
                JMP     lnext
10      lexit:  MOV.B   R7, P1OUT
11              MOV.B   R8, P2OUT
12              JMP $
13      arr1    DC8   4, 3, 2, 1, 5, 6, 8, 7, 9
14      aend
15             END
```

3.A. (2 points) How many bytes is used to store the array at the label **arr1**?
9 bytes

3.B. (2 points) What is the value of the symbol aend if we know that the arr1 symbol is 0xC020?
0xC029.

3.C. (3 points) What does the instruction in line 1 do?
R5 is loaded with the value of label arr1, which corresponds to the starting address of the byte array.

3.D. (10 points) What does this program do? Add code comments (lines 1-12).

This program finds the sums of even and odd elements of the byte array. The sum of even elements is displayed on P1OUT and the sum of odd elements is displayed on P2OUT.

3.E. (3 points) What is the value on P1OUT at the end of the program?
0x1C (28)

3.F. (bonus, 3 points) Estimate execution time of the code segment until statement in line 11 is reached (including that instruction). Assume the following: on average each instruction executed takes 1.8 clock cycles and the clock frequency is 1 MHz. Show your work.
*IC = 3+{4*7 + 3)+2 = 36*
*ExTime=IC*CPI*1us = 3.6*1.8= 64.8 us*

4. Design assembly program (20 points) Design and write an MSP430 assembly language subroutine is_palindrome(char *a, int n) that processes an alphabetical word (a word that consists of only upper- and lower-case letters) of *n* characters to determine whether the word is a palindrome or not. A palindrome word is a word that can be read the same way in either direction (e.g., “Anna” or “racecar” or “civic”). The subroutine returns 1 if the given word is palindrome, otherwise it returns 0. The main program that calls the subroutine is shown below. What does the main program do when the word is a palindrome? How does the main program pass the parameters to the subroutine?

```
#include "msp430.h"                      ; #define controlled include file
        NAME    main                     ; module name
        EXTERN  is_palindrome
        PUBLIC  main                     ; make the main label visible
                                         ; outside this module
        ORG     0FFFEh
        DC16    init                     ; set reset vector to 'init' label
        RSEG    CSTACK                   ; pre-declaration of segment
        RSEG    CODE                     ; place program in 'CODE' segment
init:   MOV     #SFE(CSTACK), SP         ; set up stack
main:   NOP                              ; main program
        MOV.W   #WDTPW+WDTHOLD,&WDTCTL   ; Stop watchdog timer
        BIS.B   #0x01, P1DIR             ; set bit 0 of port 1 as output
        MOV     #myW, R4                 ; load the starting address of the myW into the register R4
        PUSH    R4                       ; push starting address on the stack
        MOV     #myNW, R5                ; the next address in R5
        DEC     R5                       ;
        SUB     R4, R5                   ; R5 = the number of characters in the word
        PUSH    R5                       ; push the number of characters to the stack
        CALL    #is_palindrome           ; call subroutine
        BIT     #0x01, R12               ; test R12
        JZ      skip
        MOV.B   #0x01, P1OUT             ; palindrome found
skip:   JMP $

myW     DC8     "civic"
myNW
        END
// parameters are passed through the stack (the word starting address, # characters);
// if palindrome set bit 1 of port1;
// the subroutine can be extended to compare upper case and lower case letters
#include "msp430.h"                      ; #define controlled include file
        PUBLIC is_palindrome
       RSEG CODE
is_palindrome:
        PUSH  R6                       ; save R6
        PUSH  R7
        PUSH  R8
        PUSH  R9
        MOV   0x0C(SP), R6
        MOV   0x0A(SP), R7             ; the number of characters
        ADD   R6, R7
        DEC   R7                       ; R7 points to the last character
        CLR   R12                      ; no match
lcheck: MOV.B @R6+, R8
        MOV.B @R7, R9
        DEC   R7
        CMP.B R8, R9
        JNE   lfin
        CMP   R6, R7
        JGE   lcheck
        INC   R12
lfin:   POP   R9
        POP   R8
        POP   R7
        POP   R6
        RET
        END
```

5. (20 points, C language) Consider the following C program. Assume that the register SP at the beginning points to 0x0900. Assume all variables are allocated on the stack, and in the order as they appear in the program. Answer the following questions.

5.A. (10 points) Illustrate the content of the stack at the moment before the statement at line 7 is executed. What is the content of the stack pointer?

5.B. (10 points) Comment the code (lines 7 – 11) indicating the result of each statement. Illustrate the content of the stack at the end of execution of the statement in line 11. What are values of the elements of the integer array `a` if you print them at the end of the program?

```
1   int main( void )  {
2     volatile int a[3] = {-3,4,0};
3     volatile long int c = -8;
4     volatile int *p = a;
5     volatile long int *lp;
6
7     p = p + 2;       //p points to a[2], 0x08F2
8     *p = - a[1];     // a[2] = -4;
9     lp = &c;         //lp points to c, 0x08F6
10    lp++;            // lp points to 0x08FA
11    *lp = c + 10;    // M[0x08FA] = 0x000_0002
12  }
```

A.

Address	**M[15..0]**	**Comment**
0x0900		OTOS
0x08FE	0x0000	a[2]
0x08FC	0x0004	a[1]
0x08FA	0xFFFD	a[0]
0x08F8	0xFFFF	c, upper
0x08F6	0xFFF8	c, lower
0x08F4	0x08FA	p
0x08F2	0x????	lp

SP = 0x08F2

B.

Address	**M[15..0]**	**Comment**
0x0900		OTOS
0x08FE	0xFFFC	a[2]
0x08FC	0x0000	a[1]
0x08FA	0x0002	a[0]
0x08F8	0xFFFF	c, upper
0x08F6	0xFFF8	c, lower
0x08F4	0x08FE	p
0x08F2	0x08FA	lp

a[0] = 2, a[1] = 0, a[2] = -4;