CPE 323: Introduction to Embedded Computer Systems
Homework I, Keys

1 (15)	2 (20)	3 (15)	4 (10)	Total (60)

Problem #1 (15 points)
(a) (5 points)
Fill in the following table specifying the ranges of the following data types.

Type	Minimum	Maximum
8-bit unsigned integers	0	255
16-bit unsigned integers	0	65,535
8-bit signed integers (2's complement)	-128	127
16-bit signed integers (2's complement)	-32,768	32,767
12-bit signed integers (2's complement)	-2,048	2,047

(b) (10 points)
Fill in the following table. For integers find their binary, octal, and hexadecimal representation in a 16-bit computer using 2's complement.

Number	Binary	Hex	Octal
12,212	0010 1111 1011 0100	2FB4	027664
-3,460	1111 0010 0111 1100	F27C	171174

12,212/2 = 6,106 | 0
6,106/2 = 3,053 | 0
3,053/2 = 1,526 | 1
1,526/2 = 761 | 0
761/2 = 380 | 1
380/2 = 190 | 0
190/2 = 95 | 0
95/2 = 47 | 1
47/2 = 23 | 1
23/2 = 11 | 1
11/2 = 5 | 1
5/2 = 2 | 1
2/2 = 1 | 0
1/2 = 0 | 1

$12,212 = 10111110110100_2 =$

Problem #2 (20 points)
Consider the following arithmetic operations. Find the results and set the flags C, V, N, and Z accordingly.
Show your work step-by-step: that means convert the input operands to binary, find the results of arithmetic operations in binary, find the flags, and convert the result back to decimal representation.

(a) 8-bit, two's complement
$20_{10} + 100_{10}$

$20 = 14_{16} = 0001\_0100_2$
$100 = 64_{16} = 0110\_0100_2$

```
Carry     00000_100
           0001_0100
          +0110_0100
--------------------
Result     0111_1000   (78₁₆ => R= 120₁₀)
```
C=0, V=0, N=0, Z=0

(b) 8-bit, two's complement (hint: implement subtraction through addition and set flags accordingly)
$(-25)_{10} - 69_{10}$

The operation is equivalent to (-25) + (-69) => Find 2's complement representation of (-25) and (-69)
$-25 = -19_{16}$
$=> 19_{16} = 0001\_1001$
$=>$ first complement of 19_{16} is $1110\_0110_2 => -25 = 1110\_0111 = E7_{16}$
Representation: $69 = 45_{16} = 0100\_0101$
Representation of -69 in 2's complement: $1011\_1011 => BB_{16}$

```
Carry     11111_111
           1110_0111
          +1011_1011
--------------------
Result     1010_0010   (A2₁₆ = 1010_0010, FC is 0101_1101, -R= 0101_1110 => R = -94)
```

C=1, V=0 N=1, Z=0

Problem #3 (15 points)
(a) (10 points) Convert the following number from decimal to the IEEE 32-bit floating point. Show your work.
-7.275_{10}

(b) (5 points) Could you convert the decimal number 1048576.125 into a single precision float without using any significant bits of the mantissa? Explain your answer.
Solutions:
(a)
7 / 2 = 3 | 1
3 / 2 = 1 | 1
1 /2 = 0 | 1

$7_{10} = 111_2$

0.275 * 2 = 0.55 | 0
0.55 * 2 = 0.1 | 1
0.1 * 2 = 0.2 | 0
0.2 * 2 = 0.4 | 0
0.4 * 2 = 0.8 | 0
0.8* 2 = 0.6 | 1
0.6*2 = 0.2 | 1
0.2*2 = 0.4 | 0
0.4*2 = 0.8 | 0
0.8*2 = 0.6 | 1
0.6*2 = 0.2 | 1

$0.275 = 0.01000110011001100110011001100110011_2$ (the last digit is 0 followed by 11, so it is rounded to 1)

$7.275_{10} = 111.01000110011001100110011_2 = 1.\mathbf{11010001100110011001100}\mathbf{1100} * 2^2$ (Normalized number, e = 2)
Due to rounding the last bit is set to 1 instead of 0.
S = 1
E = 127 + e = 129 = 1000.0001
M = 110_1000_1100_1100_1100_1100

31	30							23	22																				0
S	E	E	E	E	E	E	E	E	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
1	1	0	0	0	0	0	0	1	1	1	0	1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	1

$-5.275_{10} = C0E8\_CCCD_{16}$

(b)
Yes, you can.
1048576.125 => the integer portion is 2^20, the fraction is 0.001 =>
$1048576.125 = 1.00000000000000000000001*2^{20}$
The significand or mantissa has 23 bits, and the single-precision format has 23-bit of mantissa, thus, all is well.

Problem #4 (10 points)

A string variable is initialized to “Welcome to CPE 323, Fall 2020!”. The first character in the string is upper case letter ‘W’. Assume the string is terminated by a NULL ASCII character. How many bytes does this string take in memory (include the terminating NULL character)? Using the ASCII table, fill in the following table by entering hexadecimal. How many bytes are needed to store this string in memory?

String	HEX value
Welcome to CPE 323, Fall 2020!	57 65 6c 63 6f 6d 65 20 74 6f 20 43 50 45 20 33 32 33 2c 20 46 61 6C 6C 20 32 30 32 30 21 00

CPE/EE 323 Introduction to Embedded Computer Systems
Homework II, Fall 2020

1(25)	2(25)	3(25)	4 (25)	Total

Problem #1 (25 points) Address Space, Memory
Consider a hypothetical 24-bit processor called HYP24 with all registers, including PC and SP, being 24 bits long. The smallest addressable unit in memory is an 8-bit byte.
A. (4 points) What is the size of HYP24's address space in bytes and KB? How many address lines does HYP24 require?

Address space: ___16,777,216_____Bytes 2^n Address space: _____16,384_____ KB (KiloBytes). $2^n/1024$
Address bus lines: __24_____

B. (6 points) Assume that first quarter of the address space is dedicated for HYP24's RAM memory and the upper half of the address space is reserved for HYP24's Flash memory. Give address ranges for the RAM and Flash memories. Fill in the table below. What is the size of the RAM memory and the Flash memory?

	Start byte address	End byte address
RAM memory	0x00_0000 Lower	0x3F_FFFF $2^n/4 - 1$ or
Flash memory	0x80_0000 $2^n - 2^n/2$	0xFF_FFFF $2^n - 1$

$2^n/4$ $\frac{16384}{2} = 8192$ $\frac{16777216}{4} =$

RAM memory size [Bytes/KB]:__4,194,304 bytes/ 4,096 KB____
Flash memory size [bytes/KB]:__8,388,608 bytes/ 8,192 KB____
$2^n/2$ Last address not big enough $4194304-1 \rightarrow$ Hex = 3FFFFF

The MSP430F20x is a microcontroller with 64 KB of address space divided between code memory (flash), RAM memory, and input/output peripherals. It has 1,024 Bytes of RAM memory starting at the address 0x0200, and 256 Bytes of address space reserved for special purpose registers and 8-bit input/output peripherals (starting at the address 0x0000) followed by 256 Bytes reserved for 16-bit input/output peripherals. The flash memory of 8 KB resides at the top of address space (highest addresses in the address space).
C. (8 points) Determine the address map by filling in the following table.

64kB. 0x0000 – 0xFFFF

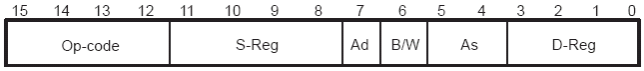
Address	Address [hexadecimal]	Sections in address space
Last Flash address	**0xFFFF**	Flash Memory
First Flash address	**0xE000**	
Last RAM address	**0x05FF**	RAM Memory
First RAM address	**0x0200**	
Last I/O address (16-bit per.)	**0x01FF**	I/O address space
First I/O address (16-bit per.)	**0x0100**	
Last I/O address (8-bit per.)	**0x00FF**	I/O address space
First I/O address (8-bit per.)	**0x0000**	

10000 – 8kB (1024)
0xE000
0x05FF
0x200 + 1024 – 1
0x01FF + 1
+1 = 0x0100 + 256 – 1
0x100 – 1 (00 included)
0x0000 + 256 bytes

D. (7 points) What is the program stack (what is it, where is it located, and how we deal with it)? What is the maximum stack size in the MSP430Fx described above? What should be the initial value of SP?
Program stack is Last-In-First-Out structure organized in RAM memory. In dealing with the stack we use a PUSH operation to put something onto the stack and a POP operation to take an item off of the stack. The SP register points to the top of the stack. In MSP430, SP points to the last full location on the stack, and stack grows toward lower addresses in memory.
The maximum size of the stack corresponds to the size of the RAM memory (512 words). Initial SP=0x0600.

Problem #2 (25 points) MSP430 Addressing Modes, Instruction Encoding

Consider the following instructions given in the table below. For each instruction determine its length (in words), the instruction words (in hexadecimal), source operand addressing mode, and the content of register R7 after execution of each instruction. Fill in the empty cells in the table. The initial content of memory is given below. Initial value of registers R5, R6, and R7 is as follows: R5=0xF002, R6=0xF00A, R7=0xFF88. Assume the starting conditions are the same for each question (i.e., always start from initial conditions in memory) and given register values. The format of the first word of double-operand instructions is shown below. (Note: Op-code for MOV is 0100).



	Instr. Address	Instruction	Instr. Length [words]	Instruction Word(s) [hex]	Source Operand Addressing Mode	R7=? [HEX]
(i)	**0x1116**	**MOV R5, R7**	**1**	**0x4507**	**Register**	**0xF002**
(ii)	**0x1116**	**MOV.B R5, R7**	**1**	**0x4547**	**Register**	**0x0002**
(a)	0x1116	MOV 6(R5), R7	2	0x4517_0x0006	Indexed	0xF014
(b)	0x1116	MOV.B 3(R5), R7	2	0x4557_0x0003	Indexed	0x00CC
(c)	0x1116	MOV.B -1(R6), R7	2	0x4657_0xFFFF	Indexed	0x00F0
(d)	0x1116	MOV EDE, R7	2	0x4017_0xDEF4	Symbolic	0xABBA
(e)	0x1116	MOV.B TONI, R7	2	0x4057_0xDEEC	Symbolic	0x0006
(f)	0x1116	MOV &EDE, R7	2	0x4217_0xF00C	Absolute	0xABBA
(g)	0x1116	MOV.B @R6, R7	1	0x4667	Register indirect	0x0044
(h)	0x1116	MOV @R6+, R7	1	0x4637	Autoincrement	0x2244
(i)	0x1116	MOV #41, R7	2	0x4037_0x0029	Immediate	0x0029
(j)	0x1116	MOV.B #27, R7	2	0x4077_0x001B	Immediate	0x001B

Label	Address [hex]	Memory[15:0] [hex]
	0xF000	0x0504
	0xF002	0xFFEE
TONI	0xF004	0xCC06
	0xF006	0x3304
	0xF008	0xF014
	0xF00A	0x2244
EDE	0xF00C	0xABBA
	0xF00E	0xEFDD

(d) MOV EDE, R7; => Displacement + PC = 0xF00C; The PC at the moment of address calculation for TONI is 0x1118h (points to the word with the displacement). Consequently, X=0xF00C – 0x1118 = 0xDEF4

(e) MOV.B TONI, R7; Displacement + PC = 0xF004 and PC=0x1118 => Displacement = 0xF00C – 0x1118 = 0xDEEC

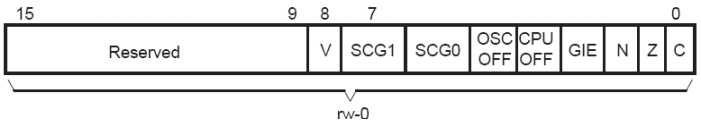
Note: consider an instruction MOV TONI, EDE. This instruction requires 3 words and the first word is followed by the source displacement (sD) and then the destination displacement (dD). We need to calculate the displacements sD and dD. The source operand resides at the address 0xF004 (TONI) and the destination operand resides at the address 0xF00C (EDE). We can write the following equations:

sD + PC = 0xF004; the PC=0x1118

dD + PC = 0xF00C; the PC=0x111A

Notice that we always take the current value of the PC into consideration. Thus, the PC is 0x1118 when we are calculating the sD and 0x111A when we are calculating the dD.

Problem #3 (25 points) MSP430 Instructions, Addressing Modes
Consider the following instructions given in the table below. For each instruction determine addressing modes of the source and destination operands, source and destination addresses, 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 registers R2, R5, and R6 is as follows: SR=R2=0x0000 (V=0, N=0, Z=0, C=0), R5=0x0403, R6=0xC006. Assume the starting conditions are the same for each question (i.e., always start from initial conditions in memory) and given register values.



Label	Address [hex]	Memory[15:0] [hex]
	0x0400	0xFEEE
	0x0402	0xA000
EDE	0x0404	0xA4BC

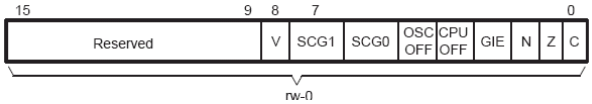
Label	Address [hex]	Memory[15:0] [hex]
	0xC000	0x0504
	0xC002	0xFEEE
TONI	0xC004	0xA8FA
	0xC006	0x33F4
	0xC008	0xF014
DEN	0xC00A	0x2244
	0xC00C	0xCDDA

	Instruction	**Instr. Size in Words**	**Source Operand Addressing Mode**	**Destination Operand Addressing Mode**	**Source Address**	**Dest. Address**	**Result (content of a memory location or a destination register; and new value of flags (C,V,Z, and N).**
(a)	ADD.B &TONI, R6	2	Absolute	Register	C004	-	byte operation: 0xFA + 0x06 += 0x00 => write the byte to R6 = 0x0000 R2=0x0003 (C=1, V=0, N=0, Z=1)
(b)	SUBC TONI, -3(R5)	3	Symbolic	Indexed	0xC004	0x0400	word operation (dst + not. src + C) src=M[0xC004] = 0xA8FA dst=M[0x0400] = 0xFEEE => result: FEEE+5705+0 = 0x55F3, M[0x0400]=0x55F3 Flags: R2=0x0001, C=1, Z=0, N=0, V=0
(c)	RRC.B @R5+	1	-	Register Indirect with Autoincrement	-	0x0403	Rotate right through carry src = 0xA0 (1010_0000), C = 0 dst = 0x50 (0101_0000), C = 0 M[0403] = 0x50 R5 = 0404 C=0, Z=0, N=0, V=0
(d)	AND.W #0xAA55, EDE	3	Immediate	Symbolic	-/(PC+2)	0x0404	src: 0xAA55 dst: 0xA4BC result: src AND dst = 0xA014 M[0404]=0x**A014;** **V=0, Z=0, N=1, C=1**

Notes of setting flags: All instructions set N and Z flags as usual. Specific details for C and V are as follows: RRC (V=0, C is loaded with the shifted out bit).

Problem #4 (25 points) MSP430 Instructions

Consider the following instructions given in the table below. For each instruction determine changes in registers after its execution. Fill in the empty cells in the table. Initial value of registers R2, R5, and R7 is as follows: R2=0x0007 (Status register), R6=0xBB66, R7=0x40A9. Assume the starting conditions are the same for each instruction in the table (i.e., always start from the initial conditions in registers). Note: Format of the register R2 is shown below. For a detailed description of the instructions use the 5xx family user guide.



Instruction		R7=0x????	V	N	Z	C
ADD.B	R6, R7	**0x000F**	**0**	**0**	**0**	**1**
ADD	R6, R7	**0xFC0F**	**0**	**1**	**0**	**0**
ADDC	R6, R7	**0xFC10**	**0**	**1**	**0**	**0**
SUB.B	R6, R7	**0x0043**	**1**	**0**	**0**	**1**
SUBC	R6, R7	**0x8543**	**1**	**1**	**0**	**0**
CMP.B	R6, R7	**0x40A9**	**1**	**0**	**0**	**1**
CMP	R6, R7	**0x40A9**	**1**	**0**	**0**	**0**
BIT	R6, R7	**0x40A9**	**0**	**0**	**0**	**1**
BIC	R6, R7	**0x4089**	**0**	**1**	**1**	**1**
BIS	R6, R7	**0xFBEF**	**0**	**1**	**1**	**1**
AND	R6, R7	**0x0020**	**0**	**0**	**0**	**1**
XOR.B	R6, R7	**0x00CF**	**0**	**1**	**0**	**1**
SWPB	R7	**0xA940**	**0**	**1**	**1**	**1**
RRC.B	R7	**0x00D4**	**0**	**1**	**0**	**1**
RRC	R7	**0xA054**	**0**	**1**	**0**	**1**
RRA.B	R7	**0x00D4**	**0**	**1**	**0**	**1**
RRA	R7	**0x2054**	**0**	**0**	**0**	**1**

CPE/EE 323 Introduction to Embedded Computer Systems
Homework III (MSP430 Assembly Language)

1(25)	2(25)	3(25)	4(25)	Total

Problem #1 (25 points) Assembly Language Directives

A. (15 points) Consider the assembly directives shown below. Show the content of relevant regions of the memory initialized by these directives. The memory is organized in words. The MSP430 ISA is a little-endian architecture. Fill in the table below. Assume that the assembler places the data segment in RAM memory starting from the address 0x1100. How many bytes is allocated and initialized by these directives?

```
            .data
lb1:        .byte 2, 4, -2, 'B'
lb2:        .byte 10011110b, 356q, 0x40
            .align 2
lw1:        .word -6, 4, 0x22AC
llw1:       .long 2
lf1:        .float 2.0
ls:         .cstring "ABC"
```

Label	Address [hex]	Memory[15:0] [hex]
lb1	0x1100	0x0402
	0x1102	0x42FE
lb2	0x1104	0xEE9E
	0x1106	0x??40
lw1	0x1108	0xFFFA
	0x110A	0x0004
	0x110C	0x22AC
llw1	0x110E	0x0002
	0x1110	0x0000
lf1	0x1112	0x0000
	0x1114	0x4000
ls	0x1116	0x4241
	0x1118	0x0043

26 bytes

B. (10 points) Consider the following sequence of instructions.

```
mov.b   lb1, R4
mov.b   lb1+2, R5
mov.w   lb2+2, R6
mov.w   llw1, R7
mov.w   lf1, R8
mov.w   ls, R9
mov.w   #ls, R10
mov.w   &lf1, R11
mov.b   &lb1+1, R12
mov.w   lf1+2, R13
```

What is the content of register R4-R9 after execution of this code snippet?

Register	Content [HEX]
R4	0x0002
R5	0x00FE
R6	0x??40
R7	0x0002
R8	0x0000
R9	0x4241
R10	0x1116
R11	0x0000
R12	0x0004
R13	0x4000

2. (25 points) Analyze assembly program

Consider the following code segment.

```
1          mov.w   #mya, R14
2          mov.w   #myaa, R13
3          sub.w   R14, R13
4          mov.b   #0x80, R7
5 MyLoop:  mov.b   @R14+, R15
6          cmp.b   R7, R15
7          jl      lskip
8          mov.b   R15, R7
9 lskip:   dec.w   R13
10         jnz     MyLoop
11         mov.b   R7, P1OUT
12
13 mya:    .byte   4, 5, 6, -10, 55, 64, -100
14 myaa:
```

A. (3 points) How many bytes is allocated by the assembly directive in line 13?
7 bytes

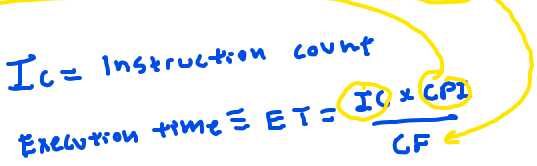
B. (3 points) What is the content of register R13 after the instruction in line 3 is completed?
7 (the number of elements of the array)

C. (10 points) What does this code segment do? Explain your answer. Hint: what does #0x80 represent?

This code segment traverses an array of signed 8-bit integers and finds its maximum (64 in this example). The maximum element is displayed on port P1.

D. (4 points) What is the value of P1OUT at the end of the program.
P1OUT = 0x40.

E. (5 points) Calculate the total execution time in seconds for the code sequence from above (line 1 – line 11). We know the following: the average CPI is 1.8 clocks per instruction. Assume the clock frequency is 1 MHz. What is MIPS rate for this code?
*IC = 4 + 5*6 (getting a new max) + 2*5 + 1 = 45 instructions*
*ExTime = IC*CPI/ClockFreq = 45*1.8/1*10^6= 81 μs*
MIPS = 1/1.8 = 0.55 MIPS



3. (25 points) Analyze assembly program

Consider the following code segment.

```
 1 RESET:      mov.w   #__STACK_END,SP         ; Initialize stack pointer
 2 StopWDT:    mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer
 3             sub.w   #12, SP
 4             mov.w   SP, R6
 5             mov.w   #myinput, R4
 6 gnext:      mov.b   @R4+, R5
 7             cmp.b   #0, R5
 8             jz      lend
 9             cmp.b   #'A', R5
10             jl      lcopy
11             cmp.b   #'Z'+1, R5
12             jl      lconv
13             jmp     lcopy
14 lconv:     sub.b   #'A', R5
15             add.b   #'a', R5
16 lcopy:     mov.b   R5, 0(R6)
17             inc.w   R6
18             jmp     gnext
19 lend:      mov.b   R5, 0(R6)
20             jmp     $               ; jump to current location '$' (endless loop)
21 myinput: .cstring "CPE325-lab"
```

A. (12 points) What does this program do? Add code comments (lines 1-19).

This program processes an input string myinput (CPE325-lab) and creates a new string on the stack. The program replaces all upper-case letters in the original string with corresponding lower-case letters in the new string.

B. (10 points) Sketch the content of the stack at the moment when the program executes the instruction at line 20. Assume that the original value of R1=0x4400 (initialized in line 1). ascii('A')=0x41, ascii('Z')=0x5A, ascii('0')=0x30.

Address	M[15..8]	M[7..0]
0x4400	0x??	0x??
0x43FE	0x??	NULL 0x00
0x43FC	'b' 0x62	'a' 0x61
0x43FA	'l' 0x6C	'-' 0x2D
0x43F8	5' 0x35	'2' 0x32
0x43F6	'3' 0x33	'e' 0x65
0x43F4	'p' 0x70	'c' 0x63

C. (3 points) Calculate the total execution time in seconds (time it takes for the program to reach statement in line 20). Assume the clock frequency is 8 MHz. How many instructions are executed in this program (before reaching the statement at line 20)?

*ExTime=N*ClockCycleTime = N/ClockRate = 188/8MHz = 23.5 μs;*

*IC = 5 + 12*3 (upper case to lower case) + 8*4 (numbers, '-') + 11*3 (lower case letters) + 3 (null) + 1 = 110*

4. (25 points) Write a subroutine

Design and write an MSP430 assembly language subroutine i2a_s(char *a, int myI) that converts a 16-bit integer, myI, into a character array with elements corresponding to the hexadecimal representation of the integer. For example, an integer myI=13,486=0x34AE is converted into an array with 4 elements as follows: a[0]='E', a[1]='A', a[2]='4', a[3]='3'. The main program that calls the subroutine is shown below. Ascii('A')=0x41, ascii('0')=0x30.

```
RESET:      mov.w   #__STACK_END,SP         ; Initialize stack pointer
StopWDT:    mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer

;-------------------------------------------------------------------------------
; Main code here
;-------------------------------------------------------------------------------
            sub.w   #4, SP                  ; allocate space for ascii chars
            mov.w   SP, R14                 ; R14 points to the allocated area
            mov.w   myI, R4                 ; integer is passed through R4
            push.w  R14                     ; push the starting address on the stack
            call    #i2a_s                  ; call subroutine
            add.w   #2, SP                  ; free space on the stack
lend:       jmp $

myI:        .word     0x34AE
;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
            .global __STACK_END
            .sect   .stack

;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
            .sect   ".reset"                ; MSP430 RESET Vector
            .short  RESET


i2a_s:
            push.w  R6                      ; save R6
            push.w  R8                      ; save R8
            push.w  R9                      ; save R9
            mov.w   0x08(SP), R6            ; R6 points at the destination
            mov.w   #4, R9                  ; R9 is counter
gnChar:     mov.w   R4, R8                  ; mov intput into R8
            and.w   #0x000F, R8             ; select the 4-bit digit
            cmp.w   #10, R8                 ; compare to 10
            jl      isDig                   ; the number is a digit
            sub.w   #10, R8                 ; for A-F, get offset
            add.w   #0x41, R8               ; ascii('A') = 0x41
            jmp     sChar
isDig:      add.b   #0x30, R8               ; for 0-9, ascii('0') = 0x30
sChar:      mov.b   R8, 0(R6)               ; store ascii character in memory
            inc.w   R6                      ; increment pointer R6
            rra.w   R4                      ; shift right (4 times)
            rra.w   R4
            rra.w   R4
            rra.w   R4
            dec.w   R9                      ; decrement step counter
            jnz     gnChar                  ; repeat if not done
            pop.w   R9
            pop.w   R8
            pop.w   R6
            ret        ; save the registers on the stack
```

PE 323 Introduction to Embedded Computer Systems
Homework IV

1 (25)	2 (20)	Total
__	__	____

Problem #1. (25 points) C language, Stack Data Placement, Pointers

Consider the following C program. Assume that the register SP at the beginning points to 0x0E00. Answer the following questions. Assume all variables are allocated on the stack, and in the order as they appear in the program.

A. (9 points) Illustrate the content of the stack at the moment before the statement at line 7 is executed.

B. (9 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 the contents of arrays x and carr?

C. (6 points) Show assembly language implementation of the statement at lines 5, 7 (think how would compiler translate these statements; it knows where the variables are placed relatively to the current top of the stack).

1	volatile unsigned int x[3]={1, 32768, 65533};
2	volatile char carr[4]={'C', '1', '0', 'a'};
3	volatile long int z = 65540;
4	volatile char *p_c = carr; // point to carr
5	volatile unsigned int *p_i = x; // point to x
6	
7	*(p_c + 2) += '6'; // carr[2]=M[0x0DF8]=M[0x0DF8] + 0x36 = 0x66 ('f')
8	p_i = p_i + 2; // p_i=0x0DFE
9	*p_i += *(p_i -6); // M[0x0DFE] <= M[0x0DFE] + M[0x0DF2] = 0FFFD+0x0004 = 0x0001
10	p_c = p_c + 3; // p_c= 0x0DF9
11	*p_c = 'A'; // carr[3]=M[0x0DF7]=0x41

Address	M[15..0]	Comment
0x0E00	0x????	OTOS
0x0DFE	0xFFFD	x[2]
0x0DFC	0x8000	x[1]
0x0DFA	0x0001	x[0]
0x0DF8	0x6130	carr[3], carr[2]
0x0DF6	0x3143	carr[1], carr[0]
0x0DF4	0x0001	z.upper
0x0DF2	0x0004	z.lower
0x0DF0	0x0DF6	p_c
0x0DEE	0x0DFA	p_i

B.

```
1   volatile unsigned int x[3]={1, 32768, 65533};
2   volatile char carr[4]={'C', '1', '0', 'a'};
3   volatile long int z = 65540;
4   volatile char *p_c = carr;      // point to carr
5   volatile unsigned int *p_i = x; // point to x
6
7   *(p_c + 2) += '6';      // carr[2]=M[0x0DF8]=M[0x0DF8] + 0x36 = 0x66 ('f')
8   p_i = p_i + 2;          // p_i=0x0DFE
9   *p_i += *(p_i -6);      // M[0x0DFE] <= M[0x0DFE] + M[0x0DF2] =
    0FFFD+0x0004 = 0x0001
10  p_c = p_c + 3;          // p_c= 0x0DF9
11  *p_c = 'A';             // carr[3]=M[0x0DF7]=0x41
```

Address	M[15..0]	Comment
0x0E00	0x????	OTOS
0x0DFE	**0x0001**	**x[2]**
0x0DFC	0x8000	x[1]
0x0DFA	0x0001	x[0]
0x0DF8	0x**4166**	carr[3], carr[2]
0x0DF6	0x3143	carr[1], carr[0]
0x0DF4	0x0001	z.upper
0x0DF2	0x0004	z.lower
0x0DF0	**0x0DF9**	p_c
0x0DEE	**0x0DFE**	p_i

*(p_c+2) += 6

0x0DF9 = M[0x0DF8] + 0x36 = 0x66

p_i = p_i + 2 = p_i + 4 (2 bytes)

*p_i = *(p_i - 6) = M[DFE] + DF2

0x0004 + 0xFFFD = 0x0001

DF2 DFE

p_c + p_c + 3 = DF9

* p_c = DF9 = 0x41

x[0-2]={1, 32768, 1}
carr[0-3]={'C', '1', 'f', 'A'}

C.

```
; volatile unsigned int *p_i = x
MOV  SP, R15
ADD  #12, R15 ; address of array x
MOV R15, 0(SP)
```

```
*(p_c + 2) += '6';
MOV.W 2(SP), R15      ; R15 is p_c
ADD.W #2, R15         ; p_c = p_c + 2
ADD.B #'6', 0(R15)    ; add #'6' to *(p_c+2)
```

Problem #2. (20 points) C language, Stack Data Placement, Pointers

Consider the following C program. Assume that the register SP at the beginning points to 0x0A00. Answer the following questions. Assume all variables are allocated on the stack, and in the order as they appear in the program. ASCII code for character '0' is 48.

```
1   int main( void )  {
2     volatile long int a = -67;
3     volatile int c = -6, d = -5;
4     volatile char mych[4] = {'0', '2', '4', '8'};
5     volatile long int *lpa = &a;
6     volatile int *pi = &d;
7     lpa = lpa - 2;     // lpa = lpa - 2*sizeof(long int)
8     *lpa = *lpa + 1026;//
9     pi += 2;            //
9     *pi = *pi + c;      //
11  }
```

long = 4
char = 1
int = 2

Fill in the following table by determining the values/addresses given below.

#	Question?	Value/Address
1	The number of bytes allocated on the stack for the variable declared in line 2.	4 bytes
2	The number of bytes allocated on the stack for the character array declared in line 4.	4 bytes
3	The number of bytes allocated on the stack for all variables declared in lines 2-6.	16 bytes
4	Value of mych[0] after initialization performed in line 4.	'0'/ 0x30
5	Address of variable a (&a).	0x09FC
6	Value of lpa at the moment after the statement in line 7 is executed.	0x09F4
7	Value of *lpa at the moment after the statement in line 8 is executed.	0x38343632
8	Value of mych[0] at the moment after the statement in line 8 is executed.	'2'/0x32
9	Value of pi at the moment after the statement in line 9 is executed.	0x09FC (points to a.low)
10	Value of *pi at the moment after the statement in line 10 is executed.	0xFFB7

(Note: The table below is not going to be graded; you can use it to sketch the stack if you want)

0x0A00	**TOS**	**Comments**
0x09FE	0xFFFF	a.upper
0x09FC	0xFFBD => 0xFFB7	a.lower
0x09FA	0xFFFA	c
0x09F8	0xFFFB	d
0x09F6	0x3834	mych[3],mych[2]
0x09F4	0x3230 => **0x3632**	mych[1],mych[0]
0x09F2	0x09FC => **0x09F4**	lpa
0x09F0	0x09F8 => 0x09FC	pi

0x38343230
0x00000401

0x38343632

CPE/EE 323 Introduction to Embedded Computer Systems
Homework V

1 (15)	2 (25)	3 (25)	4 (20)	5 (15)	Total

Problem #1 (15 points) Exception Processing
Answer the following questions.
A. (2 points) Illustrate how the top of the stack should look like at the moment just before RETI is executed.

PC	
SR	<= SP (TOP)

B. (5 points) The MSP430F5529 receives interrupt requests from the watchdog timer in the interval mode (WDT) and parallel ports P1.7 and P2.2 during execution of an instruction that takes 5 clock cycles to execute. P1 is received in the 2nd clock cycle of the instruction execution, P2 is received in the 3rd clock cycle, and WDT in the 4th clock cycle. Which ISR is accepted and processed first, assuming that WDTIE=1, P1.IE=0x80, P2.IE=0x04, and GIE=1? Hint: Inspect IVT for determining priorities.
Priority is determined by the entry number in the IVT: WDT is 57, P1 is 47 and P2 is 42. WDT is enabled, P1.7 and P2.2 are enabled, thus, the interrupt from WDT is accepted first. The clock cycle in which interrupt is received during an instruction execution is not important.

For the following questions assume that the interrupt vector table has 16 entries (0 – 15).
C. (3 points) What is the address range of the interrupt vector table (start-end)? __*FFE0-FFFE*_____

D. (3 points) What is the address of the interrupt vector with the entry number 3?
__*FFE6*_____

E. (2 points) How does the interrupt vector table get initialized and when?

Interrupt vector table is explicitly initialized by the assembly language programmer or the compiler. It is loaded at the same time the program is loaded into Flash memory.

Problem #2. (25 points) Interrupts
An MSP430-based system interfaces 4 external devices (ED0, ED1, ED2, ED3), each capable of generating an interrupt request. The external devices place a request by setting the request line (a transition from a logic one to a logic zero). The request lines are connected to port 1 pins P1.3 (ED0) and P1.4 (ED1), and port 2 pins P2.3 (ED2) and P2.4 (ED3). A request line is kept active as long as the interrupt request is pending, until the request is serviced. Answer the following questions.

A (5 points) Specify the registers that need to be initialized at the beginning to configure the system for accepting the interrupts from the devices ED0-ED3. Fill in the table below. Note: to specify interrupts active on the falling edge the edge-selection bits should be set to 1.

Register	Full Name	Content after initialization [binary contnet B7… B0], b – unknown (unchanged)
P1DIR	Port 1 Direction Register	0xbbb0.0bbb
P2DIR	Port 2 Direction Register	0xbbb0.0bbb
P1IES	Port 1 Interrupt Edge Selection	0x0001.1000
P2IES	Port 2 Interrupt Edge Selection	0x0001.1000
P1IE	Port 1 Interrupt Enable	0x0001.1000
P2IE	Port 2 Interrupt Enable	0x0001.1000
SR	Status register	GIE=1

B. (10 points) How many ISRs are needed to process interrupts from ED0-ED3. Outline the service routine (or routines if you need multiple ones) under the following conditions. If multiple requests occur at the same time, ED0 should have the highest priority and ED3 the lowest priority. Once in the service routine, you could service more than one pending request, but the highest priority one is serviced first.

Two service routines are used, P1 handling ED0 and ED1 and P2 handling ED2 and ED3.

P1_ISR (.int47 in MSP430F5529):
if (P1.IFG[3] is set [ED0 is pending]) then {Clear P1.IFG[3]; Process the interrupt for ED0};
if (P1.IFG[4] is set [ED1 is pending]) then {Clear P1.IFG[4]; Process the interrupt for ED1};
RETI

P2_ISR (.int47 in MSP430F5529):
if (P2.IFG[3] is set [ED2 is pending]) then {Clear P2.IFG[3]; Process the interrupt for ED2};
if (P2.IFG[4] is set [ED3 is pending]) then {Clear P2.IFG[4]; Process the interrupt for ED3};
RETI

C (10 points) Assume that processing request from each peripheral takes ~3 ms. The processor is in a low-power mode when not executing service routines. The following table describes a sequence of events in time. Fill in the table by answering what happens with the MSP430 on each relevant event if we know the following: ED2 raises an interrupt request at 13 ms, ED3 at 14 ms, and ED1 at 18 ms, and ED0 at 20 ms.

Time	**Event**	**MSP430 status**
0 ms	SW initialization	Active (run initialization software)
10 ms	Go to a low-power mode	Sleep
13 ms	Request from ED2 is received, pending	P2_ISR entered (ED2 portion executed, 3 ms to go)
14 ms	**Request from ED3 is received, pending**	**P2_ISR (ED2 portion executed, 2 ms to go)**
16 ms	**Finish ED2 portion of P2_ISR**	**P2_ISR (ED3 portion executed, 3 ms to go)**
18 ms	**Request from ED1 received, pending**	**P2_ISR (ED3 portion, 1 ms to go)**
19 ms	**Finish P2_ISR (ED3 portion)**	**P2_ISR exit P1_ISR entered (ED1 executed, 3 ms to go)**
20 ms	**Request from ED0 received, pending**	**P1_ISR (ED1, 2 ms to go)**
22 ms	**Finish P1_ISR**	**P1_ISR exit; Enter P1_ISR (ED0, 3 ms to go)**
25 ms	**Finish P1_ISR**	**P1_ISR exit, Enter LPM0**

Problem 3. (25 points, TimerB, Watchdog Timer)
Consider the following code segment that utilizes the watchdog timer in the interval mode with a period set in line 4 of the code.

```
1.  #include <msp430xG46x.h>
2.  void main(void) {
3.    int p = 0;
4.    WDTCTL = WDT_ADLY_250; // check the meaning of this constant in the include file
5.    P2DIR |= BIT2;         // Set P2.2 to output direction
6.    P2OUT &= ~BIT2;        // ?
7.    for (;;) {
8.      if ((IFG1 & WDTIFG) == 1) {
9.        p++;
10.       IFG1 &= ~WDTIFG;
11.       if (p == 7) P2OUT ^= BIT2;
12.       if (p == 15) { P2OUT ^= BIT2; p=0;}
13.     }
14.    }
15. }
```

A. (5 points) What does the code segment do? What does the code in line 10 do?

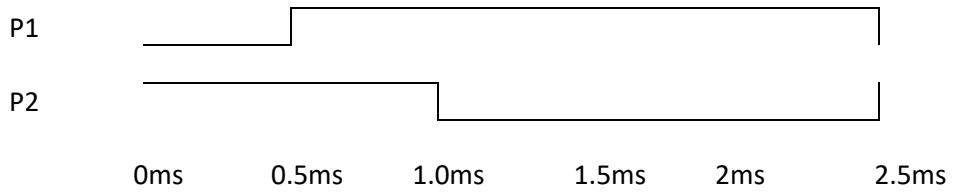
Using polling it checks when WDT counts down every 250 ms. The code drives P2.2 output – it is at a logic 0 for 7*250 ms (1.75 s), then it is toggled to a logic 1 where it remains for 8*250 ms or 2.00 s; the cycle repeats every 3.75 s (1.75 s off and 2.00 s on).

The code in line 10 clears the WDTIFG bit.

B. (10 points) How would you implement the given functionality using an interrupt service routine.

```
1.  #include <msp430xG46x.h>
2.  void main(void) {
3.    int p = 0;
4.    WDTCTL = WDT_ADLY_250; // 250 ms interval timer
5.    P2DIR |= BIT2;          // Set P2.2 to output direction
6.    P2OUT &= ~BIT2;
7.    IE1 |= WDTIE;
8.   _BIS_SR(LPM0_bits + GIE);
9.  }
10. #pragma vector = WDT_VECTOR
11. __interrupt void wdt(void) {
12.   static int p = 0;
13.   p++;
14.   if (p == 7) P2OUT ^= BIT2;
15.   if (p == 15) {P2OUT ^= BIT2; p = 0;}
16. }
```

C. (10 points) You would like to generate two periodic pulse-width modulated (PWM) signals P1 and P2, with frequency of 400 Hz (one period is 2.5 ms). Assume that an 2^20 Hz clock on SMCLK is used by TimerB. Can you do this using TimerB? If yes, describe a TimerB configuration (content of control and data registers) that will carry out signal generation? Note: use English and waveforms to describe your solution.



Yes, we can use the TimerB device to generate the pulse-width modulated signals. The signals are periodic and the period is 2.5 ms. The TimerB can be configured to work in the UP mode, and TBCCR0 should be initialized to ensure 2.5 ms period. We assume that capture and compare blocks TBCCR1 and TBCCR2 are used to create P1 and P2, respectively.

In the UP mode the TimerB counter is incremented until it reaches the value in TBCCR0 and then it goes back to 0. Since, the source clock period is 1/2^20 μs, one period has 2^20/400 clock cycles or 2621.44 clock cycles => TBCCR0=2620.

One period of P1 has the following shape: it is initially at logic 0 for 0.5 ms and then becomes a logic 1 for another 2.0 ms. Thus TBCCR1 should be 1/5th of the value in TBCCR0 or 524. The output mode for TBCCR1 block should be set/reset.

One period of P2 has the following shape: it is initially at a logic 1 for 1 ms and then becomes a logic 0 for another 1.5 ms. Thus, TBCCR2 should be 2/5th of the value in TBCCR0 or 1048. The output mode for TBCCR2 block should be reset/set.

Problem 4. (20 points), Clocks Time, Timers

Consider the following code segment. Assume that processor clock in the active mode is set to 2,000,000 Hz.

```
1.  while(1) {
2.    int i;
3.    for(i = 800; i>0; i--);          // one loop iteration takes 5 clock cycles
4.    P3OUT |= BIT5;                     // Set P3.5
5.    for (i = 800; i>0; i--);          // Delay
6.    P3OUT &= ~BIT5;                    // Clear P3.5
7.  }
```

A. (5 points) What does the code segment do assuming that P3.5 is configured as a digital output. You may ignore delay needed to execute instructions in lines 1, 4 and 6.

This code is an infinite loop. The code toggles the output port P3.5 every 800*5 = 4,000 clock cycles or 4,000*0.5 us = 2,000 us or 2.0 ms, resulting in a square wave at the output with period of 4 ms or 250 Hz.

B. (5 points) What will happen if you connect P3.5 to the buzzer?

The buzzer will play a tone at 250 Hz.

C. (10 points) How would you implement functionality achieved by the code segment above using TimerB. Port P3.5 is multiplexed with the output signal from the capture and compare block 4 of TimerB. Give details. How would you initialize the system? What would you do in the main loop? Assume the SMCLK is used which is 2^20 =1,048,576 Hz.

Setup TimerB to operate in UP mode. If we are using SMCLK=1,048,576 Hz, we should set the CCR0 to ensure a period of 4 ms. 4 ms is equivalent to 4,194.3 SMCLK clock ticks, thus, seting CCR0 to 4,193 will ensure roughly the desired period. The CCR4 register should be loaded with one half of the CCR0 or 2,096; It should operate in the compare mode, with output signal set in either set/reset or reset/set mode. The main loop would be unnecessary, and the processor may spend all the time in a low-power mode.

Problem 5. (15 points) UART Serial Communication
Consider the following C source code (assume that P5.BIT1 is connected to a LED).

```
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCIA0RX_ISR (void)
{
  P5OUT |= BIT1;
  while(!(IFG2&UCA0TXIFG));
  UCA0TXBUF = UCA0RXBUF;
  while(!(IFG2&UCA0TXIFG));
  UCA0TXBUF = UCA0RXBUF;
  while(!(IFG2&UCA0TXIFG));
  UCA0TXBUF = UCA0RXBUF;
  P5OUT &= ~BIT1;
}
```

A. (5 points) What does this code segment do? USCIA0 is configured in the UART mode.

It is an interrupt service routine associated with USCI device. When a new character is received it is sent back over UART (echo function) 3 times. The LED connected to P5.1 is set during execution of the interrupt service routine.

B. (5 points) USCI0 is configured in the UART mode to transfer 57,600 bits/second, 8-bit characters, odd parity, and two stop bits. How many processor clock cycles (MCLK = 2^20 Hz) expire during the time USCI needs to send one character over the UART?

1+8+1+2 = 12 bits per character; time to transmit a character is 12/57,600 = 0.2083333 ms
1 MCLK = 1 SMCLK= 1/2^20 s = 0.95367 us
Time for sending one character = 12*(2^20/57,600) = 218.45 MCLKs

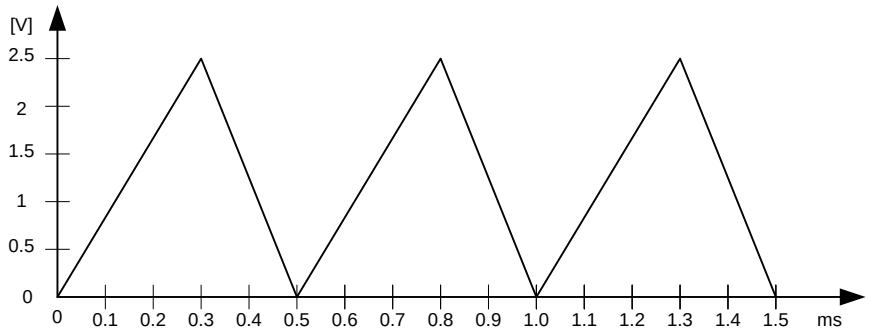
C. (5 points) Describe how to configure USCI to achieve the desired communication speed from B using the low frequency UART mode and SMCLK as the source clock, SMCLK=MCLK, (specify values of all important control registers and individual bit fields in these registers).

One bit period takes N=18.2 SMCLKs
=> Thus UCBR0 = INT(N)=18, UCBR1=0; UCBRSx=round((N – INT(N))*8)= round((18.2 – 18)*8) = round(1.6)=2
Thus, 6 out of every 8 bits sent by the USCI will take 18 SMCLKs and 2 out of every 8 bits will take 19 SMCLKs.

CPE 323 Introduction to Embedded Computer Systems
Homework VI

1 (30)	2 (20)	3 (20)		Total

Problem #1. (30 points) ADC.
Your task is to write a program that samples an analog signal *a0* as illustrated below using the MSP430's ADC12 device. The samples are then forwarded to the MSP430's DAC12 device. Answer the following questions.



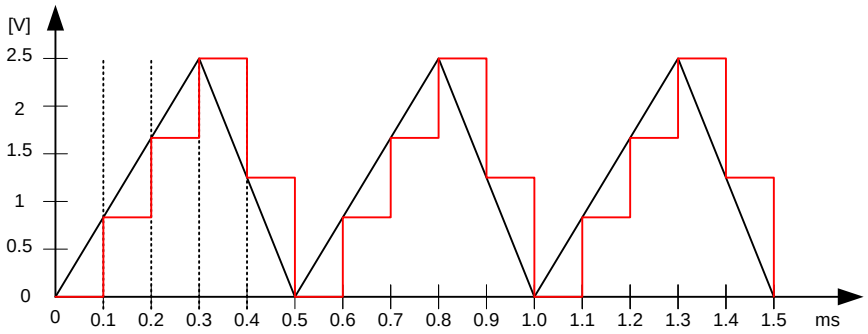
A. (2 points) What is the maximum and minimum input voltage of the input signal *a0*?
Max: 2.5 V
Max: 0 V

B. (2 points) What is duration of one period of the input signal *a0* in milliseconds? What is the frequency of *a0*?
*T = 0.5 ms => f = 1/0.5*10^-3 = 2000 Hz*

C. (10 points) Let us assume that we configure the MSP430's device to sample analog input *a0* with the sampling frequency f_{sample} = 10 KHz (10,000 samples are taken every second). How many samples do we have per one period of the input analog signal? Fill in the following table (please note that the number of rows does not reflect the number of samples per one period). Assume that our sampling is synchronized with *a0* (i.e., the first sample is taken at the very beginning of an *a0* period at t=0 s). Assume reference voltages V_{R+} = 2.5 V and V_{R-} = 0V.

Sample number	*a0* [V]	Sample value [decimal]
1	*0*	0
2	2.5/3 = 0.833	1,365
3	2*(2.5/3)=1.666	2,730
4	2.5 V	4,095
5	2.5/2 = 1.25 V	2,048 (or 2,047)

D. (4 points) Assume the ADC12 samples are immediately sent to MSP430 DAC12 device. Sketch the output analog signal created by the MSP430's DAC12 device (as you would see it on the oscilloscope).



E. (7 points) Give a short description of your program that performs the ADC and DAC conversions. We assume that clocks are initialized as follows: $f_{MCLK}=f_{SMCLK}=4$ MHz. What should be done to initialize devices, what is done in the main program loop, and what is done in corresponding interrupt service routines?

Initialization:

a) Clock subsystem (MCLK=4 MHz, SMCLK=4 MHz, ACLK = 32 KHz)

b) Configure corresponding ports for special functions (ADC12 input, DAC12 output)

*c) TimerA to control sampling; fsample = 5*2,000 = 10,000 Hz; Configure CCR0 in up mode, TACCR0= 4MHz/10KHz = 400.*

d) Configure ADC12 (trigger from TimerA CCR0), V_{R+} = 2.5V , V_{R-} = 0V

e) Configure DAC12: 12-bit, scale=1, V_R = 2.5 V

ADC12_ISR:

Read sample from channel A0.

DAC12_xDAT = sample;

F. (5 points) If you know that less than 80 clock cycles is spent for processing one sample (read the sample for the ADC12 and write the digital value of the sample to the DAC12 data register), what would be the maximum sampling frequency we could have without oversubscribing our processor time? Elaborate your answer.

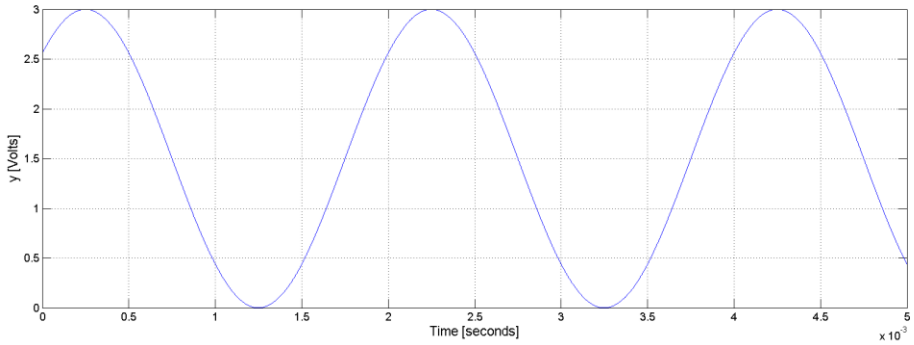
5 samples x 80 clock cycles = 400 clock cycles for 0.5 ms;

*The number of processor ticks in 0.5 ms is 0.5*10^-3/(1/4*10^6) = 2000 clock cycles.*

*So we could increase the number of samples to 2000/80 = 25 samples per period which corresponds to 2,000*25 Hz = 50,000 Hz = 50 KHz.*

Problem #2. (20 points) ADC, DAC.

To drive an external actuator we need to provide an analog periodic signal y, defined as follows, y=1.5*(1+sin(2*pi*f*t+pi/4)), f=500 Hz. Your task is to generate this signal using a **16-bit digital-to-analog** converter peripheral (DAC16). Answer the following questions.



A. (2 points) What is the maximum and minimum output voltage at the analog output? Fill in the table below by specifying min/max values and times when those values are achieved.

Min/Max	Value [Volts]	Time [milliseconds]
Min	0	1.25, 3.25
Max	3.0	0.25, 2.25, 4.25

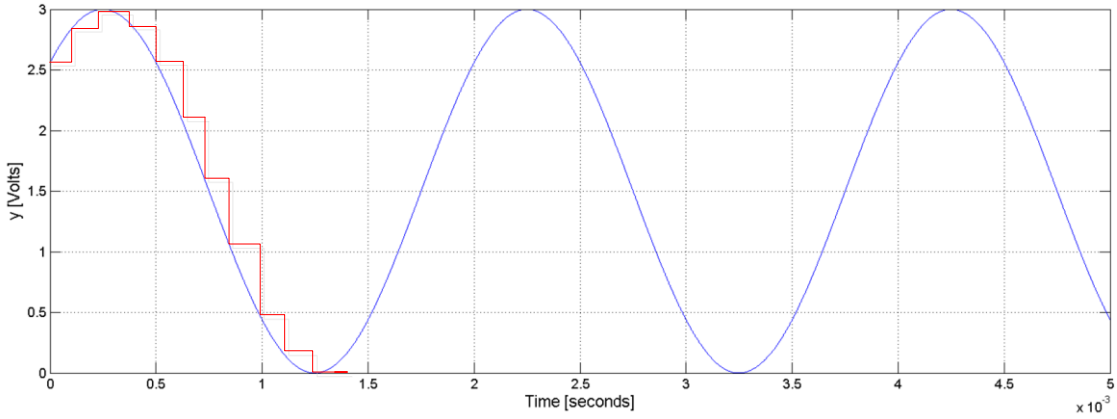
B. (2 points) What is duration of one period of the signal y in milliseconds?

T = 2 ms

C. (8 points) Let us assume that we provide a lookup table with only 16 samples for a single period of the signal y? Fill in the following table with the values for the first 4 samples (assume the first sample starts at t=0). Assume reference voltages V_{R+} = 3.0 V and V_{R-} = 0V

Sample	*t=? [ms]*	*sin(2*pi*f*t+pi/4)*	*y=1.5*(1+ sin(2*pi*f*t+pi/4))*	*Lookup table [16-bit unsigned value in decimal]*
1	**0.0**	**0.7071**	**2.5606**	**55,938**
2	**0.125**	**0.9238**	**2.8858**	**63,040**
3	**0.25**	**1**	**3**	**65,535**
4	**0.375**	**0.9238**	**2.8858**	**63,040**

E. (4 points) Sketch the output analog signal as you would see it on the oscilloscope. Use the lookup table from part C.



F. (4 points) If we use a TimerB ISR to control sending the next value to the DAC, how many interrupts per second TimerB should generate?

500*16 = 8,000

3. (20 points) I/O Peripherals

A. (20 points) You are designing an embedded application that interfaces a buzzer and a switch using MSP430's general-purpose input/output pins. The button is connected to P4.4 and the buzzer is connected to P4.2. Illustrate how the button and the buzzer connect to the MSP430. Your application should control the buzzer as follows: as long as we keep the button pressed (logic 0 at the input), the buzzer should be on. To activate the buzzer you should provide a square wave with frequency of 8 KHz.

Sketch the microcontroller and its connections to the button and the buzzer. Describe the application design. What needs to be done during initialization, what is done in the main program loop, and what is done in interrupt service routine(s)? You can use C/C++ code or English/pseudo-code to illustrate your design.

Initialization:
Configure basic clock module: e.g., MCLK=SMCLK=4MHz, ACLK=32KHz;
Configure P4.4 as input port (digital I/O); configure P4.2 as output; P4DIR |= BIT2;
Clear P4.2: P4OUT &= ~BIT2;
Configure TimerA: UP mode, 4MHz/16 KHz = 250 => TACCR0=249 (select SMCLK as input).
*The TimerA ISR is used to toggle the output of P4.2. As the toggle rate should be 2*8 KHz, the TACCR0 is set to 249.*
Main Program:
Loop:

```
    Disable TimerA interrupt;
    while (P4IN & BIT4); // wait for button to be pressed
    Debounce();
    if (!(P4IN & BIT4)) Enable TimerA interrupt;
    while !(P4IN & BIT4); // button is pressed
    // button is released
    Goto Loop

TimerA_ISR
    Toggle P4.2: P4OUT ^= BIT2;
```