1. (6) Describe in detail the sequence of events that occurs when interrupts are enabled and an interrupt occurs (the keyboard generates an interrupt, for example).

2. (3) What are the 3 ways we talked about in class to monitor the status of an I/O device?

3. (2) What is DMA? Which CUSP I/O device uses DMA?

4. (4) In CUSP, the SP points to a certain item and grows in a certain way. In class, we talked about 3 other ways to manage a stack. Describe all 4 methods (include CUSP).

5. (5) Suppose the following code is executed:

```
LDA#    $C0
OUTB    $00
INB     $00
HLT
```

What does the Accumulator contain after the HLT instruction?

6. (10) Given the following code sequence:

```
        .EQU    @,$000
        LDX#    $1
        LDC     STRING
        ADA#    $000C
        ADX#    $01
        STC     STRING
        HLT
STRING: .WORD   $106123
        .END
```

What memory locations and registers are **altered** by the execution of this program? Show the values of the registers and memory locations that have **changed** when the program terminates.

ACC = **XR =** **PC =** **SP =**

MEM[ ] = **MEM[ ] =** **MEM[ ] =** **MEM[ ] =**

7. (25) Given the following assembly language program, fill out the symbol table, and write down the contents of the registers after the execution of each instruction. (Assume you have begun single-stepping with the instruction at location $10E).

Assembly Language Program:

```
        .EQU    @,$100
A:      .WORD   $103
B:      .WORD   $104
C:      .WORD   $7A4C21
        .EQU    D,$FFF
E:      .WORD   $102
F:      .WORD   $003
G:      .WORD   $2EA047
        .EQU    @,$107
H:      .WORD   $106
I:      .WORD   $634212
J:      .WORD   $453964
K:      .WORD   $107
L:      .EQU    P,$123
M:      .WORD   $005
N:      .WORD   $100
O:      .WORD   $549AE2

        AND&    ! $008
        LDA+    F
        LDC*    ! $001
        LDS+    ! $005
        LDF&    E
        LDX*    K
        LDA#    D
        LDA     O
        TAX
        POPA
        .END
```

Symbol Table	

Before AND Inst.	**ACC=$2**	**SP=$106**	**XR=$5**	**FP=$102**	**PC=$10E**
After AND Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After LDA Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After LDC Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After LDS Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After LDF Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After LDX Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After LDA Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After LDA Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After TAX Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After POPA Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**

8. (20) The following is an interrupt-driven print string program. This program will not work as shown - there are at least 5 lines missing that are necessary in order to make the program function correctly. Your task is to fill in the missing lines, so that the program will work.

```
                .EQU    @,$100
                .EQU    PRT_CNTL,$010
                .EQU    PRT_DATA,$011
                .EQU    INTERRUPT_ENA,$80
                .EQU    ENABLE_AND_RESET,$C0
                .EQU    STACKTOP,$E00
MAIN:           LDS#    STACKTOP
                LDA#    INTERRUPT_ENA
                OUTB    PRT_CNTL
                LDX#    0
                LDC     STRING
                OUTB    PRT_DATA
                STX     INDEX
DOW0RK:         NOP
                JMP     DOW0RK
ISR:            PSHA                            ; save registers
                LDX     INDEX                   ; advance to next string char.
                AOC#    LENGTH                  ; and test for output complete
                JEQ     DONE
                LDC     STRING                  ; output not complete -
                OUTB    PRT_DATA                ; pring character
                STX     INDEX                   ; store updated index
                JMP     EXIT
DONE:           LDA#    ENABLE_AND_RESET        ; output complete -
                OUTB    PRT_CNTL                ; clear interrupt pending bit
EXIT:           POPX                            ; restore regs
                POPA
                IRTN
STRING:         .CHAR   'This data will be printed. ',LENGTH
```

9. (25) You have already written a program to compare two null-terminated strings (problem 7.7). The code on the following page is most of the answer according to the authors of the book - you are to fill in the missing lines. For this test, the function is defined as follows:

FUNCTION STRCMP(VAR S1,S2:CHAR_STRING): INTEGER;

The program is to compare two null-terminated strings one character at a time, exiting when either the two characters do not match or when the null character has been encountered. On exit, the accumulator should contain either a 0 (strings matched), a negative number (string 1 was "smaller"), or a positive number (string 1 was "larger").

(Hint - figure out how you did this problem, and then look at this code and see what has to be added to make it work correctly.)

```
; Main Program


        .EQU    @,$000

        .EQU    STACKTOP,$E00

        LDS#    STACKTOP

        JSR     STRCMP

        ADS#    2

        HLT

SS1:            .CHAR  'ABCD\$00'

SS2:            .CHAR  'ABCDE\$00'
```

(Problem continues on following page)

```
; FUNCTION STRCMP(VAR S1,S2:CHAR_STRING): INTEGER;

STRCMP:   BGN#  0          ; init. call
          CLR   STOP       ; STOP := false
          LDX#  0
LOOP:     LDC*  ! S2       ; get and save S2[I]
          STA   S2TMP
          JNE   LAB        ; if EOS then set stop
          SET   STOP       ; STOP := true
LAB:      SBA   S2TMP      ; S1[I] - S2[I]
          JNE   DONE
          TST   STOP       ; EQ=1 if STOP=0; LT=1 if STOP<0
          JEQ   LOOP
DONE:     POPX             ; restore register
          RTN
S2TMP:    .BLKW 1
STOP:     .BLKW 1
          .END
```

ASCII Table (MSD = Most Significant Digit)																
MSD (Hex)	**Least Significant Digit (Hex)**															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
2	space	!	"	#	$	%	&	,	(	)	*	+	’	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	‘	a	b	c	d	e	f	g	h	i	h	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	del

Instructions:

Instruction	**Opcode (in Hex)**
LDA	00
LDX	01
LDS	02
LDF	03
STA	04
STX	05
STS	06
STF	07
ADA	10
ADX	11
ADS	12
ADF	13
SBA	14
SBX	15
SBS	16
SBF	17
CMA	20
CMX	21
CMS	22
CMF	23
HLT	FFFFFF

Addressing Modes:

Mode	**Opcode**
Immediate	0
Direct	2
Indexed	4
Indirect	6
Indirect Indexed	8

I/O Port Information:

I/O Port	Register
$000	Keyboard Control
$000	Keyboard Status
$010	Printer Control
$010	Printer Status
$020	Tape Drive Control
$020	Tape Drive Status
$030	Timer Control
$030	Timer Status
$316	CRT Control

Keyboard		
Register	Bit Number 7654 3210	Interpretation
Control	x--- ---- -x-- ---- --xx xxxx	1 = enable interrupts, 0 = disable interrupts 1 = flush buffer, 0 = no operation unused (no affect)
Status	x--- ---- -x-- ---- --xx xxxx	1 = ready (data available) 1 = interrupt enabled unused (always zero)
Interrupt Addr		$FF8

Tape Drive		
Register	Bit Number 7654 3210	Interpretation
Control	x--- ---- -x-- ---- --xx ---- ---- xxxx	1 = enable interrupts, 0 = disable interrupts 1 = clear interrupt request, 0 = no operation 00 = no operation 01 = read record 10 = write record 11 = rewind tape
Status	x--- ---- -x-- ---- --x- ---- ---x ---- ---- 1--- ---- -xxx	1 = ready (for an operation) 1 = interrupt enabled 1 = tape mounted 1 = interrupt pending 1 = end of tape encoutered on read unused (always zero)
Interrupt Addr		$FFA

Printer		
Register	Bit Number 7654 3210	Interpretation
Control	x--- ---- -x-- ---- --xx xxxx	1 = enable interrupts, 0 = disable interrupts 1 = clear interrupt request, 0 = no operation unused (no affect)
Status	x--- ---- -x-- ---- --x- ---- ---x ---- ---- xxxx	1 = ready (to receive character) 1 = interrupt enabled 1 = printer on-line 1 = interrupt pending unused (always zero)
Interrupt Addr		$FF9

Timer		
Register	Bit Number 7654 3210	Interpretation
Control	x--- ---- -x-- ---- --xx ---- ---- xxxx	1 = enable interrupts, 0 = disable interrupts 1 = clear ready bit, 0 = no operation 00 = no operation 01 = start timer (after loading counter) 10 = stop timer 11 = start timer (without loading counter) unused (no affect)
Status	x--- ---- -x-- ---- --xx xxxx	1 = ready (count complete) 1 = interrupt enabled unused (always zero)
Interrupt Addr		$FFB