1. (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).

2. (8) Describe in detail the sequence of events that occurs when interrupts are enabled and a keyboard interrupt occurs.

3. (3) What is the parameter passing convention used in CUSP? What are the other options?

4. (3) Suppose the following code is executed:

```
LDA#    $C0
OUTB    $10
INB     $10
HLT
```

Assuming the device is on-line and ready, what does the Accumulator contain after the HLT instruction?

5. (13) Given the following code sequence:

```
        .EQU    @,$000
L1:     LDA#    $10
        TAX
        LDC     L1
L2:     ADA#    $40
        STC     L1
        LDX#    L2
L3:     STC     STRNG1
        HLT
STRNG1: .WORD   $610713
STRNG2: .WORD   $000007
STRNG3: .WORD   $000008
        .EQU    @,$10
        .BLKW   5,$07
        .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. (No, there is no typo in this problem.) (Well, there wasn't the *first* time I gave it, anyway ...)

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

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

6. (29) 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 $106).

Assembly Language Program:

```
        .EQU    @,$100
L1:     .WORD   $EE0102
L2:     .WORD   $101
        .EQU    D,$FFF
L3:     .WORD   $2EA047
L4:     .WORD   $390EFF
L5:     .EQU    L,$123
L6:     .WORD   $100
L7:     .WORD   $0FD

        LDA#    L
        LDA     L1
        POPA
        LDS+    ! $002
        LDC*    ! $006
        LDA&    ! $006
        LDA+    L3-3
        LDX*    L6
        TAF
        .END
```

Symbol Table	

Before LDA# Inst.	**ACC=$2**	**SP=$101**	**XR=$3**	**FP=$0FE**	**PC=$106**
After LDA# Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After LDA Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After POPA Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After LDS+ Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After LDC* Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After LDA& Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After LDA+ Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After LDX* Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After TAF Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**

7. (25) The following is a Tape drive to Display program. This program will not work as shown - there are at least 5 lines that have been changed. Your task is to find and fix the lines so that the program will work.

```
                .EQU    TDR_CNTL,$030
                .EQU    TDR_STAT,$020
                .EQU    TDR_LEN,$021
                .EQU    TDR_ADDR,$022
                .EQU    READ_BITS,$10
                .EQU    MOUNTED_BIT,$20
                .EQU    EOT_BIT,$08
                .EQU    CRT_CNTL,$316
                .EQU    CRT_DATA,$317
                .EQU    NEXT_LINE_VALUE,$05
MAIN:           INB     TDR_STAT        ; check for tape drive ready
                JGE     STOP
                LDA     BUFFER
                OUTW    TDR_ADDR
NEXT_REC:       LDA#    B_SIZE
                OUTB    TDR_LEN
                LDA#    READ_BITS       ; initiate read
                OUTB    TDR_CNTL
POLL:           INB     TDR_STAT        ; wait for operation complete
                JGT     COMPLETE
                AND#    MOUNTED_BIT     ; check that tape still mounted
                JEQ     STOP
                JMP     POLL
COMPLETE:       AND#    EOT_BIT
                JNE     STOP
                INB     TDR_LEN
                JEQ     NEW_LINE        ; if zero, skip data output
                STA     ! COUNT
                LDX#    0
COPY_CHAR:      LDC     BUFFER
                OUTB    CRT_DATA        ; copy to screen
                AOC     COUNT
                JLT     COPY_CHAR
NEW_LINE:       LDA#    NEXT_LINE_VALUE ; move cursor to next line
                OUTB    CRT_DATA
                JMP     NEXT_REC
STOP:           HLT
                .EQU    B_SIZE,30       ; max characters per record
BUFFER:         .BLKW   (B_SIZE+2)/3    ; buffer length is rounded up
COUNT:          .BLKW   1
                .END
```

8. (15) 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
                LDC     STRING
                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        ; print 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
INDEX:          .BLKW   1
STRING:         .CHAR   'This data will be printed. ',LENGTH

                .EQU    @,$FF9
                .WORD   ISR
```

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
Frame Immediate	1
Direct	2
Frame Direct	3
Indexed	4
Frame Indexed	5
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 (to begin new 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