1. (5) What does the IRTN instruction do, and why is it necessary?

2. (5) Describe the difference between I/O-mapped and Memory-mapped I/O, and give one advantage and one disadvantage of each.

3. (5) 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).

4. (5) What is a "null-terminated" string, what are its advantages and disadvantages, and why we are interested in it? (Is this the approach used in CUSP?)

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

6. (20) Here is the Timer beep program using interrupts we looked at in class. This program will not work as it appears here - there are at least 5 lines missing that are necessary in order to make the program function correctly. (There are several other lines that have been removed that have no impact on the functionality). Your task is to fill in the missing lines, so that the program will work.

```
            .EQU    CRT_DATA,$317
            .EQU    TIM_CNTL,$030
            .EQU    TIM_VALUE,$031
            .EQU    ENABLE_RESET_AND_START,$D0
            .EQU    ENABLE_AND_RESET,$C0
MAIN:       LDS#    STACKTOP
            LDA#    1000
            OUTW    TIM_VALUE
            LDA#    ENABLE_RESET_AND_START
            OUTB    TIM_CNTL
FIDDLE:     NOP
            JMP     FIDDLE
ISR:        PSHA
            LDA#    ENABLE_AND_RESET
            OUTB    TIM_CNTL
            LDA#    $07
            OUTB    CRT_DATA
            IRTN
```

7. (30) 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 $10D).

Assembly Language Program:

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

        XOR&    ! $009
        LDC*    ! $003
        LDA+    F
        LDS+    ! $006
        LDF&    E
        LDX*    B
        LDA#    K
        LDA     G
        TAX
        POPA
        .END
```

Symbol Table	

Before XOR Inst.	**ACC=$2**	**SP=$106**	**XR=$5**	**FP=$100**	**PC=$10D**
After XOR Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After LDC Inst.	**ACC=**	**SP=**	**XR=**	**FP=**	**PC=**
After LDA 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. (25) Below is the "sample.csp" program which adds a pair of numbers. I have changed 5 lines of the program so that the program no longer works the way it should. You are to figure out which lines are wrong, and fix the program so that it will work again.

```
;  This sample CHASM program adds pairs of numbers.
; It terminates when the first value entered is zero.
            .EQU    PUT_NUM,$E00    ; MINI_OS write number routine
            .EQU    GET_NUM,$E01    ; MINI_OS read number routine
            .EQU    PUT_STR,$E05    ; MINI_OS write string routine
            .EQU    PUT_NL,$E06     ; MINI_OS write new line routine
            .EQU    StackTop,$E00   ; top of stack
START:      LDS     StackTop        ; initialize stack pointer
READ1:      PSH#    P1_Length       ; prompt to obtain first value
            PSH#    Prompt1
            JSR     PUT_STR         ; call PUT_STR(P1_Length,Prompt1)
            ADS#    2
            JSR     GET_NUM         ; read first value
            CMA#    0               ; if zero we're all done
            JNE     DONE
            STA     Value1          ; store first value
READ2:      PSH#    Prompt2         ; prompt to obtain second value
            PSH#    P2_Length
            JSR     PUT_STR         ; call PUT_STR(P2_Length,Prompt2)
            ADS#    2
            JSR     GET_NUM         ; read second value
            ADA     Value1          ; add values together
            STA     Result          ; and store result
            PSH#    A_Length        ; output answer message
            PSH#    Answer
            JSR     PUT_STR         ; call PUT_STR(A_Length,Answer)
            ADS#    3
            LDA     Result          ; output actual answer
            INT     PUT_NUM
            JSR     PUT_NL          ; output blank line
            JMP     READ1           ; go back for next pair of values
DONE:       HLT
Prompt1:    .CHAR   'Enter first value (0 halts)',P1_Length
Prompt2:    .CHAR   'Enter second value',P2_Length
Answer:     .CHAR   'The sum of the values is ',A_Length
Value1:     .BLKW   1
Result:     .BLKW   1
            .END
```