1. (25 points) Arrays are usually passed using CBA instead of CBV.

   a) (10 points) In two sentences, describe two disadvantages of passing arrays using CBV?

   b) (5 points) What addressing mode do you use to access an array that is passed using CBA?

   c) (5 points) What addressing mode would you use to access an array that was passed using CBV?

   d) (5 points) In one sentence, describe a circumstance that would make passing an array using CBV preferable to passing the array using CBA?

2. (32 points) Given the following series of instructions, fill in the hexadecimal values of the registers when the program halts. Also fill in the 20 chars contained in the ARR4, including the hyphens.

   ```
   .EQU @,0
   LDS# $E00
   PSH# ARR1
   PSH# ARR2
   PSH# ARR3
   JSR TEST
   HLT
   TEST:  BGN# 2
   LDX# 3
   LAST:  LDA& ! 5
          STC+ ARR4
          SOJ LAST
          ADS# 3
          RTN
   .EQU @,$100
   ARR1:  .CHAR 'ABCDEFGHIJKLMNOPQRST'
          .EQU @, $200
   ARR2:  .CHAR '12345678901234567890'
          .EQU @, $300
   ARR3:  .CHAR 'abcdefghijklmnopqrstuvwxyz'
          .EQU @, $400
   ARR4:  .CHAR '--------------------'
   ```

   ACC ____________
   XR ____________
   SP ____________
   FP ____________
   PC ____________
   IR ____________

   ARR4 (8 points)

3. (10 points) Our crack CPU research team is going to add to a register to CUSP. Of the ACC, XR, FP, SP, which should not be considered for duplication, and why?
4. (15 points) For this question you are given the following program fragment, timer ISR, and correct declarations. The program fragment is suppose to initialize and then start the timer so that the timer causes an interrupt every $238$ cycles. The timer ISR is suppose to cause the monitor bell to ring once and then return safely to the program.

```
.EQU CRT_CNTRL, $316
.EQU BELL, $06
.EQU TIMER_CNTRL, $030
.EQU TIMER_RELOAD, $031
.EQU TIMER_VECTOR, $FFB
.EQU START_AFTER_LOAD, $10
.EQU RESET_READY_BIT, $40
.EQU ENABLE_INT, $80
.EQU @, 0

PROGRAM:  LDA# $238
STA TIMER_RELOAD
LDA# TIMER_ISR
STA TIMER_VECTOR
LDA# START_AFTER_LOAD + RESET_READY_BIT + ENABLE_INT
STA TIMER_CNTRL

.EQU @, $200

TIMER_ISR: PSHA
LDA# BELL
OUTB CRT_CNTRL
POPA
IRTN
```

Assuming the timer ISR is correct, the program fragment fails to achieve its purpose because of three errors in the PROGRAM fragment. What are the three errors:

1) ____________________________________________________________________________________
2) ____________________________________________________________________________________
3) ____________________________________________________________________________________

5. (68 points) For this question you will be implementing the C string function strncpy(), which copies a given number of characters from s2 into s1. The declaration is: char* strncpy(char *s1, char *s2, int n). The arguments s1 and s2 point to strings (arrays of characters terminated by a null character). The strncpy subroutine copies up to n characters from s2 into s1, and then returns s1. If s2 is shorter than n, then s2 and its terminating null character are copied to s1. If the length of s2 is n or greater, then the first n characters of s2 are copied to s1 and no null character is appended. Here are some examples (with the implicit null characters shown):

Assuming s1 initially contains “Hi\0”
If strncpy(s1, "there\0", 6), then after the call s1 is "there\0".
If strncpy(s1, "there\0", 7), then after the call s1 is "there\0".
If strncpy(s1, "there\0", 1), then after the call s1 is "ti\0". The null character is that of the original s1
If strncpy(s1, "there\0", 4), then after the call s1 is "ther". There is no way to tell what is after the 'r'

a) (60 points) Write a CUSP implementation strncpy(). Remember to make your function preserve the appropriate registers.

b) (8 points) Given the following CUSP declarations write a CUSP call to the strncpy subroutine that implements strncpy(Arr1, Arr2, x). For this I don't want to know the result. Just write the CUSP code to put the parameters on the stack and call strncpy.

```
Arr1: .CHAR 'This is just some garbage\$00'
Arr2: .CHAR 'This some more garbage\$00'
x:   .WORD 7
```