(25) **Question 1.** The status of the FIFO is strategic to understanding the performance of a producer/consumer system. This is the FIFO that links the foreground SCI_OutChar to the background SCI interrupt handler. This FIFO can hold up to 9 characters.

```c
char static volatile *TxPutPt;  
char static volatile *TxGetPt;  
char static TxFifo[10];
void TxFifo_Init(void){
    TxPutPt=TxGetPt=&TxFifo[0];  }
int TxFifo_Put(char data){
    char volatile *tempPt;
    tempPt = TxPutPt;
    *(tempPt++) = data;
    if (tempPt==&TxFifo[10]){  
        tempPt = &TxFifo[0];  }
    if (tempPt == TxGetPt ){  
        return(0);  }
    else{
        TxPutPt = tempPt;
        return(1); }}
int TxFifo_Get(char *datapt){
    if(TxPutPt == TxGetPt)  
        return(0);
    else{
        *datapt = *(TxGetPt++);
        if(TxGetPt==&TxFifo[10]){  
            TxGetPt = &TxFifo[0];  }
        return(1);  }
    }
void SCI_OutChar(char data){
    // debugging instrument inserted here
    while(TxFifo_Put(data) == 0){};  
    SC0CR2 = 0xAC;
}
```

Part a) Why are the **TxPutPt** and **TxGetPt** defined as static?

Part b) Write a function that returns the number of elements currently stored in the TxFifo. You may not modify the existing functions or global variables. If the FIFO is empty return a zero. If the FIFO is contains 5 elements return a 5. A 9 means the FIFO is full.
Part c) Write a minimally-intrusive debugging instrument, which will be inserted at the beginning of SCI_OutChar. This instrument will collect data concerning the number of elements in the TxFifo. A histogram is a frequency plot containing the number of occurrences versus TxFifo size. In particular, Count[n] will contain the number of times SCI_OutChar was called with n elements in the TxFifo.

unsigned short Count[10]; // histogram

Part d) What would the histogram look like if the system were I/O bound?
Section 9.7.1.2 shows a 32K PROM interface to a MC68HC812A4. The goal is to find a fast-enough memory so that cycle stretching is not required.

Part a) What are the maximum values of $t_{\text{ACC}}$, $t_{\text{CE}}$, $t_{\text{OE}}$ allowed to operate without cycle stretching?

Part b) Draw the read-cycle timing diagram for the new interface.

$CE^* = \text{CSP0}$

R/W

A14-A0

Read Data Available

Read Data Required
(50) Question 3. The following is from Lab17os.c. In this problem, you will modify the OS so the memory for the TCB is dynamically allocated on the heap. In particular, there will be no NumThread MAX_THREADS or SystemTCB. Instead, you should define a pointer TCBPtr pt; and execute the code pt=malloc(sizeof(TCBType)); to create a new TCB. malloc returns a NULL (0) pointer when the heap is full. You may assume RunPt is defined and initialized to NULL (0). To return the memory back to the heap, execute free(pt);

//******** OS_AddThread ***************
// add a foreground thread to the scheduler
// Inputs: pointer to a void/void foreground function
// Outputs: 1 if successful, 0 if this thread cannot be added
int OS_AddThread(void(*fp)(void)){
    if(NumThread >= MAX_THREADS){
        return 0; // structure is full
    }
    if(NumThread){
        SystemTCB[ NumThread-1 ].Next=&SystemTCB[ NumThread ];
    }
    SystemTCB[ NumThread ].StackPt = &SystemTCB[ NumThread ].InitialCCR;
    SystemTCB[ NumThread ].Id = 1<<NumThread;
    SystemTCB[ NumThread ].InitialCCR = 0x40;
    SystemTCB[ NumThread ].InitialPC = fp;
    SystemTCB[ NumThread ].Next = &SystemTCB[ 0 ];
    NumThread++;
    return 1;
}

Part a) Write a new OS_AddThread that dynamically allocates space and links it into the circular active TCB list. If RunPt is NULL (0), this is the first thread created. Additional global variables are allowed.
Part b) Write a new function `OS_Kill` that a thread can call to kill itself. Return the TCB to the heap, reconstruct the circular TCB list, and launch another thread. You may assume there is at least one active thread (all the threads do not die.)