An IBM customer had trouble installing software and rang for support. "I put in the first disk, and that was OK. It said to put in the second disk, and had some problems with the disk. When it said to put in the third disk, I couldn't even fit it in..." The user hadn't realized that "Insert Disk 2" implied to remove Disk 1 first.

A confused caller to IBM was having trouble printing documents. He told the technician that the computer had said it couldn't find the printer. The user had also tried turning the computer screen to face the printer, but that his computer still couldn't "see" the printer.

**Quiz 2 review**

1) **subroutines**
   - understand what is passed in as input
     - 8-bit numbers in registers A, B
     - 16-bit numbers in registers D, X, Y
     - e.g., look at `SCI_OutUDec`
     - pointers to array/string in registers X, Y
     - e.g., look at `SCI_OutString`
   - understand what is returned as output
     - 8-bit numbers in registers A, B
     - 16-bit numbers in registers D, X, Y
     - e.g., look at `SCI_InUDec`
     - pointers to array/string in registers X, Y
     - e.g., look at `SCI_InString`
   - ends in `rts`

2) **arrays and strings**
   - if 8-bit then
     - read/write data using RegA or RegB
     - increment pointer by 1 after each access
   - if 16-bit then
     - read/write data using RegD
     - increment pointer by 2 after each access
   - how is the length determined?
     - Fixed length (known at design time)
     - First element is length
     - Last element is terminator code

3) **programming techniques**
   - if-then
   - if-then-else
   - for-loop
   - global variables

4) **test taking strategies**
   - understand the relationship between my program and yours
   - my program is the main program
   - my program outputs to the SCI, not yours
you write two subroutines that I will call
my program calls your first subroutine six times, then
my program calls your second subroutine six times
my program passes into yours some data, then
gives you a score based on your return parameters
understand the problem
look for test data my program feeds to yours
find it at the top of ROM
comments give results your program should return
don’t access this data directly, I will pass you a pointer
set a breakpoint at start of your program
add registers to ViewBox and single step
worry about special cases last

7.6. Profiling
Profiling is similar to performance debugging
Profiling collects
  when  time history, TCNT
  where PC
  what  strategic variables

7.6.1 Profiling using a software dump

unsigned short time[100];
unsigned short place[100];
unsigned short n;
void profile(unsigned short p){
    time[n]=TCNT; // record current time
    place[n]=p;
    n++;
}
SIZE equ 100
time ds.w SIZE
place ds.w SIZE
n ds 1
Debug_Init
    clr n
    rts
*RegY = p
Debug_profile
    pshb
    pshx
    ldab n
    cmpb #2*SIZE
    bhs ddone
ldx  #time
movw TCNT,B,X  record time
ldx  #place
sty  B,X       record place
addb #2
stab n
ddone pulx
pulb
rts

Change place parameter to capture PC instead of
Add profiling to Robot.asm for each function

7.6.2. Profiling using an Output Port

DDRT     equ 0x0242  ; Direction Register
PTT      equ 0x0240  ; I/O Register

add to SCI_Init
    movb  #$FF,DDRT
add to SCI_OutDec
    movb  #$01,PTT
    movb  #$02,PTT
    movb  #$04,PTT
    movb  #$08,PTT
    movb  #$10,PTT
    movb  #$20,PTT
    movb  #$40,PTT
    movb  #$80,PTT

attach a logic analyzer

8. Program structures
Chapter 8 objectives are to:
• Explain how to define local variables on the stack,
• C compilers local variables and pass parameters,
• call by value versus call by reference parameter passing,
• Use linked structures to implement finite state machines,
• Implement system calls using software interrupts,

8.7. Finite state machines with statically-allocated linked structures

8.7.1. Abstraction
Software abstraction
    define a problem with a set of basic abstract principles
    separate policies mechanisms

Jonathan W. Valvano
**Finite State Machine (FSM.)**

inputs, outputs, states, and state transitions  
state graph defines relationships of inputs and outputs

The three advantages of this abstraction are  
1) it can be faster to develop  
2) it is easier to debug (prove correct) and  
3) it is easier to change

**Moore FSM**

output value depends only on the current state, and  
inputs affect the state transitions  
significance is being in a state

input: when to change state  
output: how to be in that state

**Mealy FSM** depend both on the current state and the inputs.  
output value depends on input and current state  
inputs affect the state transitions.  
significance is the state transition

input: when to change state  
output: how to change state
linked structure
- multiple identically-structured nodes
- pointer (or link) to other nodes
- statically_allocated fixed-size linked structures
- used to define the FSM
- one-to-one mapping FSM state graph and linked structure
- one structure for each state

8.7.2. Stepper motor controller

This stepper motor FSM has two input signals four outputs.
Figure 8.18. A unipolar stepper motor interfaced.

// Port AD bits 7-6 are inputs
// PAD7 = CCW, PAD6=CW
// Port M bits 3-0 are outputs to stepper
const struct State {
    unsigned char Out;       // command
    const struct State *next[4];
} =
typedef const struct State StateType;
#define S5 &fsm[0]
#define S6 &fsm[1]
#define S10 &fsm[2]
#define S9 &fsm[3]
StateType fsm[4] = {
    { 5, {S5, S6, S9, S5} },
    { 6, {S6, S10, S5, S6} },
    {10, {S10, S9, S6, S10} },
    { 9, {S9, S5, S10, S9} } };

void main(void) {
    StateType *Pt;
    unsigned char Input;
    Timer_Init();
    DDRM = 0x03f;
    ATDDIEN = 0xC0;
    DDRAD = 0;
    PTM = 5;  /* initial output */
    Pt = S5;  /* initial state */
    while(1) { /* never quit */
Timer_Wait(8000);  /* 1ms wait */
Input = (PTAD&0xC0)>>6;
Pt=Pt->next[Input];
PTM=Pt->Out;   /* stepper drivers */
}
}

Write in assembly