Midterm Exam Statistics

• Mean Score = 62
• Median Score = 62
• High Score = 86
• Distribution
  80s = 5
  70s = 3
  60s = 11
  50s = 9
  <50 = 8

Note: A solution will be posted on the class web site.

Final Project

• Conducted during the last three weeks of class
• Assignment: Design and implement an embedded application of your choosing
• Constraint: Your system must include at least one of the following:
  – Use of a PIC feature not used in labs—e.g. CCP unit
  – Use of a protocol not used in labs—e.g. I2C
  – Use of a peripheral chip not used in lab
• Scope/complexity of your application must be at least comparable to that of lab 5 and lab 6.
• Stretch yourselves--more points will be awarded to more ambitious projects
• Project proposals will be due on **Tues. April 10**

Designing an Embedded Application—Lab 5 as an Example

55:036
Embedded Systems and Systems Software

Lab 5—Major Elements

• Read User-IDs from Mag Stripe reader
• Read PINs from Keypad
• Compare the entered USER-IDs and PINs to values pre-stored in on-board EEPROM
• Display text on LCD
• Check for pushbutton presses
• Update stored PINs (in EEPROM)
Where Do We Start?

• Central issues: use of timers, other devices, interrupts
• Usually a good idea to consider timing constraints first
• This will dictate the overall structure of the application

Lab 5—Timing Constraints

• Mag Stripe Reader
  
  Data must be sampled within approx. 0.75 msec. after leading edge of strobe

Lab 5—Timing Constraints

• Keypad:
  – Switch debounce requires sampling keypad at approx. 10 msec. intervals
  – Variation of a few msec. in either direction can be tolerated

• Pushbutton switch:
  – Debounce shouldn’t be an issue since we are just checking for evidence a button push.
  – Only interested in button pushes that occur within 5 seconds of authorization

Lab 5—Timing Constraints

• LCD
  – Writing a character to the LCD requires approx. 45 microseconds
  – Need to insure that programmed delays don’t interfere with servicing of other devices
    • e.g. Writing 16 characters to the LCD requires approx. 0.7 milliseconds
So, How Should Lab 5 be Structured?

- Main Program Loop:
  - Authorize Users
    - Check entered User-ID/PIN against values stored in EEPROM
  - Handle PIN updates
  - Service LCD
- Mag. Stripe Reader
  - Service via external interrupt
  - Should be OK to trigger interrupts on leading edge of strobe.
- KeyPad
  - Service via periodic timer interrupt
  - 10 msec. period should work OK

Interrupt Priorities

- It should be OK to configure both external interrupts and timer interrupts as low priority.
- Timing constraints for both devices are fairly flexible
- No particular advantage to giving one priority over the other
- Both ISRs will be short
- Shouldn’t interfere too much with each other
- Will need to use polling to determine which interrupt has occurred
- Alternatively, can assign one device high priority and the other low priority.

Lab 5 Program Structure

- Main program loop
  - (services LCD, implements logic for validating User IDs and PINs to authorize access, and changing PINs)
- Timer ISR
  - (for reading Keypad)
- External Interrupt ISR
  - (for servicing mag-stripe reader)

Designing the Mag-Stripe Reader

- ISR
  - This ISR will be invoked at leading-edge of each strobe cycle (approx the middle of each data bit period) from the mag-stripe reader
  - ISR needs to sample the data line from the mag-stripe reader and assemble and store digits read from track 2
Track 2 Format

<table>
<thead>
<tr>
<th>Odd Parity</th>
<th>b3</th>
<th>b2</th>
<th>b1</th>
<th>b0</th>
<th>Meaning of Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Start sentinel</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Print data char</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Second data char</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Third data char</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Fourth data char</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>End sentinel</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>LRC char</td>
</tr>
</tbody>
</table>

Characters are recorded LSB first with parity at the end—i.e. right to left in this diagram.

Track 2 data characters are 4 bits + parity. (Must be numeric: 0x0 – 0x9)

Designing the Mag-Stripe Reader ISR

- General Idea
  - Identify the beginning of the “start sentinel”
  - Assemble next four bits into a digit and store in an array
  - Read the parity bit and discard (or verify correct parity)
  - Continue to assemble and store digits as above until the 10-digit User-ID, stop sentinel, and LRC character have been read

A Useful Structuring Tool—Finite State Machine

- S0: Waiting for Start Sentinel
- S1: First (l.o.) bit of char read
- S2: Second bit of char read
- S3: Third bit of char read
- S4: Fourth (h.o.) bit of char read
- S5: Parity bit read

Note: Structure the Mag-reader ISR as a FSM—one state transition per invocation.
Implementing the FSM

```c
#define S0 0
#define S1 1
#define S2 2
#define S3 3
#define S4 4
#define S5 5
char next_state = S0;
char current_bit;
char next_bit;
char ch;
char User_ID[12];

Mag_Stripe_ISR() {
    current_bit = next_bit
    next_bit = /* data value read from Mag stripe rdr */
    switch (next_state) {
        S0:
            if (next_bit == 0)
                next_state = S0;
            else
                next_state = S1;
            break;
        S1:
            /* do S1 stuff here */
            next_state = S2;
            break;
        S2:
            /* do S2 stuff here */
            next_state = S3;
            break;
        S3:
            /* do S3 stuff here */
            next_state = S4;
            break;
        S4:
            /* do S4 stuff here */
            next_state = S5;
            break;
        S5:
            /* do S5 stuff here */
            if (/* 12 chars have been read */)  
                next_state = S0;
            else
                next_state = S1;
    }  // end switch
    } // end of ISR
```

Possible Lab 5 Program Structure

Possible Lab 5 Program Structure (Ignoring RS-232 I/O)

ISR sets Card_Swiped to indicate that a complete User-ID has been read from a card. The User-ID is placed by the ISR into the array ID.
Main Program as a FSM

A_IDLE

Main Program as a FSM (Ignoring PIN Change Logic)

A_IDLE

CHECK_ID

!(Card_Swiped)

Card_Swiped

GET_PIN

invalid ID

valid ID

CHECK_PIN

PIN_Entered
Main Program as a FSM (Ignoring PIN Change Logic)

Pseudo-Code for Main FSM states

```c
main () {
    //Initialization Goes Here
    while(1)
    switch (Auth_Next_State) {
        case A_IDLE:
            /* A_IDLE State Stuff Goes Here */ break;
        case CHECK_ID:
            /* CHECK_ID State Stuff Goes Here */ break;
        case GET_PIN:
            /* GET_PIN State Stuff Goes Here */ break;
        case CHECK_PIN:
            /* CHECK_PIN State Stuff Goes Here */ break;
        case AUTHORIZE:
            /* AUTHORIZE State Stuff Goes Here */ break;
    }
}
```

Initialization:
- Display "Swipe Card" on LCD;
- Turn off Authorization LED;
- Card_Swiped = false;
- PIN_Entered = false;
- Auth_Next_State = A_IDLE;

Pseudo-Code for Main FSM states

**Initialization:**
- Display "Swipe Card" on LCD;
- Turn off Authorization LED;
- Card_Swiped = false;
- PIN_Entered = false;
- Auth_Next_State = A_IDLE;

**Pseudo-Code for Main FSM states**

**IDLE:**
- if (Card_Swiped)
  - Auth_Next_State = CHECK_ID;
- else Auth_Next_State = A_IDLE;
- break;
Pseudo-Code for Main FSM states

CHECK_ID:
Reset Card_Swiped;
If (ID matches a stored User-ID) {
    Stored_PIN = PIN of matching stored User-ID;
    Display "Enter PIN" on LCD;
    Auth_Next_State = GET_PIN;
} else {
    Display "Invalid ID" on LCD
    Auth_Next_State = A_IDLE;
} break;

GET_PIN:
if (PIN_Entered)
    Auth_Next_State = CHECK_PIN;
else
    Auth_Next_State = GET_PIN;
break;

CHECK_PIN:
If (PIN == Stored_PIN) {
    Turn on Authorization LED;
    Display "Door is Unlocked" on LCD;
    Auth_Next_State = Authorize;
} else {
    Display "Re-enter PIN" on LCD;
    Auth_Next_State = GET_PIN;
} break;

AUTHORIZE:
If (less than five seconds have elapsed since entering this state)
    Auth_Next_State = AUTHORIZE;
else {
    Turn off authorization LED;
    Display "Swipe Card" on LCD;
    Auth_Next_State = A_IDLE;
} break;
Discussion

• You will need to add in the “Change PIN” functionality
• This will require extensions to the Main FSM
• Need to carefully review timing issues to make sure that there is no interference among various parts of the system.