Midterm Exam 1

- Fill in your name:
- This exam is open book and open notes.
- The exam is 80 minutes and worth 100 points.
- Show all your work.

<table>
<thead>
<tr>
<th>Question</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
</tr>
</tbody>
</table>
1. **Short Answer** (18 points)

   (a) List two reasons why one would implement a variable using a register.

   (b) List two reasons why one would implement a variable on the stack.

   (c) List two reasons why one would implement a variable in RAM as a global variable.

   (d) What is the difference between regular interrupts and periodic polling?

   (e) What are the interrupt related bits in the CCR?

   (f) In all programs using a 68HC11 with ROM, they start with lines similar to the following:

       org $FFFE
       fdb Start

       where Start is the starting address of the program. What is the purpose of this code?

   (g) Which bit in the PIOC is not writable? What operation sets this bit? What operation clears this bit?
2. **Addressing Modes** (12 points)

Assume that each instruction begins in the state given below:

- C = 1
- RegA = $97
- RegB = $CA
- RegX = $3000
- RegY = $2000
- [$0020] = $B6
- [$3000] = $1A
- [$3001] = $2B
- [$3002] = $3C
- [$3003] = $4D
- [$00FB] = $F1
- [$00FC] = $E2
- [$00FD] = $D3
- [$00FE] = $C4
- [$00FF] = $B5
- SP = $00FD
- PC = $0000

For each of the following instructions, fill in the table below indicating the addressing mode, effective address if any, operand if any, and new contents of relevant registers and/or memory locations in the comment. Note: do not treat the instructions like a program. In other words, each instruction starts anew in the state given above.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Addressing Mode</th>
<th>Address</th>
<th>Operand</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldx $20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ldy #$20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ldx 2,x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bmi 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pula</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stab $3003</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. **Stack** (10 points)

Hand execute the following program. After each instruction is executed, indicate the condition of the stack and the values of any registers that changed. The original condition is RegA=$66, RegB=$55, RegX=$A6B5, and RegY=$A2B1. Assume the stack is initially empty, with RegSP=$00FF.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>psha</td>
<td></td>
</tr>
<tr>
<td>pshy</td>
<td></td>
</tr>
<tr>
<td>pshx</td>
<td></td>
</tr>
<tr>
<td>pula</td>
<td></td>
</tr>
<tr>
<td>pulb</td>
<td></td>
</tr>
<tr>
<td>pulx</td>
<td></td>
</tr>
<tr>
<td>pula</td>
<td></td>
</tr>
</tbody>
</table>
4. **FSM** (10 points)
   Write the pseudo instructions that would start the assembly implementation for the state machine below.

```
S1
Out=00
Wait=10
00,10,11
S2
Out=01
Wait=20
01 11
00,10
S3
Out=11
Wait=30
10 01
00,10,11
```
5. **Gadfly Synchronization** (25 points)

In this problem, you will interface with gadfly synchronization the input device to Port C of a 6811 as shown above. The timing to input 1 byte is shown also shown above. The sequence to input an ASCII character from the input device is as follows. When a new character is available, the input device puts its ASCII code on the 8-bit data bus, then the input device makes \texttt{Ready}=1. Next, your software should read the 8-bit ASCII code, then acknowledge receipt of the data by pulsing \texttt{Acknowledge}. After the pulse the input device will make \texttt{Ready}=0 again.

(a) Show bit by bit your choice for the parallel I/O control register.
(b) Show the ritual that initializes the microcomputer.
(c) Show the subroutine that inputs one data byte. Use gadfly synchronization. Return the new data by reference using \texttt{RegY}. \texttt{RegY} points to the place to store the next data byte.
6. **Interrupts** (25 points)
   In this problem, you redo your interface from the last problem using interrupts. You can assume the existence of the subroutines InitFifo, PutFifo, and GetFifo. InitFifo has no parameters and you can assume that it will save and restore any registers that it uses. PutFifo puts the 8-bit data in RegA into the FIFO. GetFifo takes a pointer in RegX to the place to put the 8-bit data taken from the FIFO.

   (a) Show bit by bit your choice for the parallel I/O control register.
   (b) Show the ritual that initializes the microcomputer.
   (c) Show the interrupt handler that inputs one byte of data. Ignore FIFO full errors. No polling is required. You do not need to write the main program that processes the data.