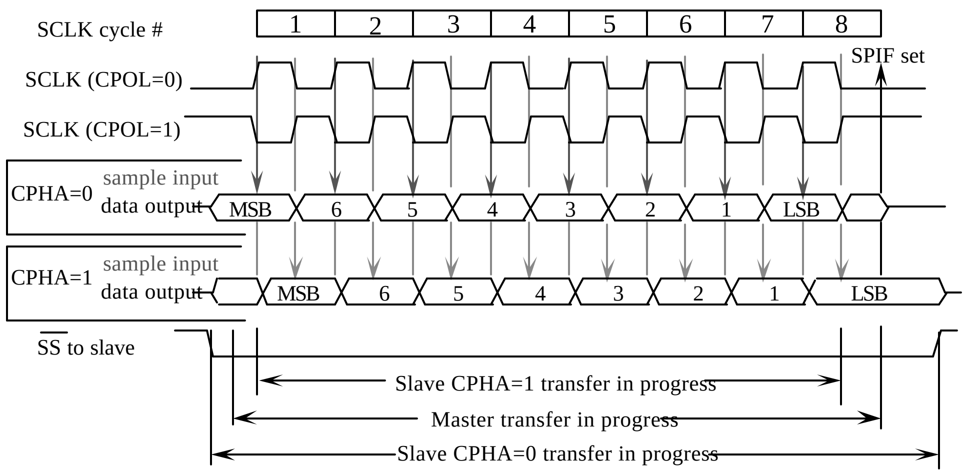
TCNT is 16-bit up counter (see **TSCR2**)
TCn are 16-bit input capture/output compare latch registers, n=0 to 7
PTT is 8-bit bi-directional I/O port
DDRT is the associated direction register for Port T (0 means input, 1 means output)
ATDDIEN ADC digital enable register, 1 to make corresponding pin digital, 0 to make corresponding pin analog
DDRAD is the associated direction register for digital pins of Port AD (0 means input, 1 means output)
TIOS is 8-bit input/output select (0 means input capture, 1 means output compare)
TSCR1 is a timer control register
    bit 7 **TEN**, 1 means allow timer to function normally, 0 means disable timer including **TCNT**
TFLG1 is 8-bit timer flag register
    set by input capture or output compare event
    cleared by write to this register with bit set
TFLG2 is 8-bit timer flag register
    bit 7 **TOF** timer overflow interrupt flag, set on **TCNT** overflow, cleared by write to this register with bit set
TIE is 8-bit timer arm register
    1 means corresponding bit in **TFLG1** will request an interrupt
    0 means corresponding bit in **TFLG1** will not request an interrupt
TSCR2 is 8-bit timer control register
    bit 7 **TOI** timer overflow interrupt enable, 1 = interrupt on TOF, 0 = TOF interrupts will not occur
    bits 2,1,0 **PR2**, **PR1**, **PR0**, select rate, let **n** be the 3-bit number
    without PLL TCNT is 4MHz/$2^{\mathbf{n}}$, with PLL TCNT is 24MHz/$2^{\mathbf{n}}$, **n** ranges from 0 to 7
TCTL3 is 8-bit timer control register, input capture mode (00=off, 01=rise, 10=fall, 11=both rise and fall)
    bits 7-6 **EDG7B,EDG7A** input capture 7 edge , bits 5-4 **EDG6B,EDG6A** input capture 6 edge
    bits 3-2 **EDG5B,EDG5A** input capture 5 edge, bits 1-0 **EDG4B,EDG4A** input capture 4 edge
TCTL4 is 8-bit timer control register, input capture mode (00=off, 01=rise, 10=fall, 11=both rise and fall)
    bits 7-6 **EDG3B,EDG3A** input capture 3 edge, bits 5-4 **EDG2B,EDG2A** input capture 2 edge
    bits 3-2 **EDG1B,EDG1A** input capture 1 edge, bits 1-0 **EDG0B,EDG0A** input capture 0 edge
CRGFLG real time interrupt flag register
    bit 7 **RTIF** real time interrupt flag, set on RTI timeout, cleared by write to this register with bit set
CRGINT real time interrupt control register
    bit 7 **RTIE** real time interrupt enable, 1 means interrupt on RTIF, 0 means RTI interrupts will not occur
RTICTL real time interrupt control register, M clock is 4 MHz
    bits 6-4 **RTR6**, **RTR5**, **RTR4**, select rate, let **n** be the 3-bit number, **n** ranges from 1 to 7
    bits 3-0 **RTR3**, **RTR2**, **RTR1**, **RTR0**, select rate, let **m** be the 4-bit number, **m** ranges from 0 to 7
        interrupt period is 128µs*(**m**+1)*$2^{\mathbf{n}}$
SPICR1 SPI control register
    bit 6 **SPE** — SPI System Enable
        0 = SPI internal hardware is initialized and SPI system is in a low-power disabled state.
        1 = SPI function enabled
    bit 4 **MSTR** — SPI Master/Slave Mode Select
        0 = Slave mode 1 = Master mode
    bits 3-2 **CPOL**, **CPHA** — SPI Clock Polarity, Clock Phase
    bit 1 **SSOE** — Slave Select Output Enable
SPIBR SPI baud rate
    bits 6-4 **SPPR2**, **SPPR1**, **SPPR0**, select rate, let **m** be the 3-bit number, **m** ranges from 0 to 7
    bits 2,1,0 **SPR2**, **SPR1**, **SPR0**, select rate, let **n** be the 3-bit number, **n** ranges from 0 to 7
        SPI clock frequency is 12MHz/(**m**+1)/$2^{\mathbf{n}}$
SPISR SPI control register
    bit 7 **SPIF** set after the eighth SCK cycle in a data transfer
        cleared by reading status register (with SPIF set) followed by read or write to SPI data register.
SPIDR is 8-bit SPI data register



ATDCTL2 ADC control register, bit 7 **ADPU**, set to enable ADC
ATDCTL4 ADC control register, bit 7 **SRES8**, 1 for 8-bit ADC, 0 for 10-bit ADC
ATDCTL5 ADC control register
    bit 7, DJM Justification, 1=right justified, 0=left justified
    bit 6, DSGN Signed Representation, 1=signed, 0=unsigned
    bit 5, SCAN, 0 = single sequence of conversions then stop, 1 = continuous conversion
    bit 4, MULT, 0 = sequence of conversions on a single channel, 1 = sequence of conversions on multiple channels
    bits 2-0, write channel number to start ADC, channel number 0 to 7
ATDSTAT 16-bit ADC status register, bit 15 **SCF**, cleared by write to **ATDCTL5**, set when ADC finished
ATDDR0 first 10-bit ADC result
SCIDRL 8 bit data serial data register
SCIBD is 16-bit SCI baud rate register, let **n** be the 16-bit number Baud rate is 12MHz/**n**
SCICR1 is 8-bit SCI control register
    bit 4 M, Mode, 0 = One start, eight data, one stop bit, 1 = One start, eight data, ninth data, one stop bit
SCICR2 is 8-bit SCI control register
    bit 7 TIE, Transmit Interrupt Enable, 0 = TDRE interrupts disabled, 1 = interrupt whenever TDRE set
    bit 5 RIE, Receiver Interrupt Enable, 0 = RDRF interrupts disabled, 1 = interrupt whenever RDRF set
    bit 3 TE, Transmitter Enable, 0 = Transmitter disabled, 1 = SCI transmit logic is enabled
    bit 2 RE, Receiver Enable, 0 = Receiver disabled, 1 = Enables the SCI receive circuitry.
SCISR1 is 8-bit SCI status register
    bit 7 TDRE, Transmit Data Register Empty Flag
        Set if transmit data can be written to SCDR
        Cleared by **SCISR1** read with TDRE set followed by **SCIDRL** write.
    bit 5 RDRF, Receive Data Register Full
        set if a received character is ready to be read from **SCIDRL**
        Clear the RDRF flag by reading **SCISR1** with RDRF set and then reading **SCIDRL** .

```
0xFFD6    interrupt 20 SCI0/SCI
0xFFDE    interrupt 16 timer overflow
0xFFE0    interrupt 15 timer channel 7
0xFFE2    interrupt 14 timer channel 6
0xFFE4    interrupt 13 timer channel 5
0xFFE6    interrupt 12 timer channel 4
0xFFE8    interrupt 11 timer channel 3
0xFFEA    interrupt 10 timer channel 2
0xFFEC    interrupt 9  timer channel 1
0xFFEE    interrupt 8  timer channel 0
0xFFF0    interrupt 7  real time interrupt
```

Place your answers on pages 5 and 6.

(10) Question 1. List the events in proper order as a **`RDRF`** interrupt causes the computer to switch from foreground to background? Do not include events explicitly caused by executing software in either the foreground or in the background, just the hardware events.

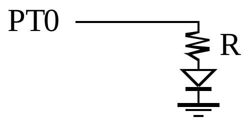
A) The **`RDRF`** interrupts are armed by setting the **`RIE`** bit in the **`SCICR2`** register
B) The PC, Y, X, B, A, CC registers are pulled from the stack
C) The CC, A, B, X, Y, PC registers are pushed on the stack
D) The **`I`** bit is set to one (disable)
E) The PC is loaded with the 16-bit contents of **`$FFD6`**
F) The **`SCISR1`** is read with **`RDRF`** set, followed the reading of **`SCIDRL`**, clearing **`RDRF`**
G) Incoming data is moved from the shift register to the data register, setting **`RDRF`**
H) The **`I`** bit is cleared to zero (enable)

PLACE THE CORRECT SEQUENCE OF LETTERS ON THE ANSWER PAGE.

(15) Question 2. The objective of this question is to interface a solenoid to PT7 using one NPN transistor. You may add simple components like capacitors, diodes and resistors, but no additional transistor devices other than the one NPN transistor. Show all connections between the 6812 PT7 and the solenoid. The solenoid requires a voltage between 4V and 6V at 50mA to activate. The solenoid has a 0.01mH inductance. You do not have to specify values of the resistors, capacitors, and diodes, just give the circuit.

(10) Question 3. Let **x** be the precision of the ADC **in bits**. Let **y** and **z** be the minimum and maximum analog voltages for the ADC in volts respectively. Let $\mathbf{f_s}$ be the sampling rate in Hz. Let $\mathbf{t_c}$ be the ADC conversion time (how long each conversion takes) in µs. In terms of the variables **x y z** $\mathbf{t_c}$ $\mathbf{f_s}$, give the equation for the ADC resolution in volts. You can double-check your equation by considering the 9S12C32 example of **x**=10 bits, **y**=0V, **z**=5V, $\mathbf{f_s}$ =1kHz, and $\mathbf{t_c}$=5 µs.

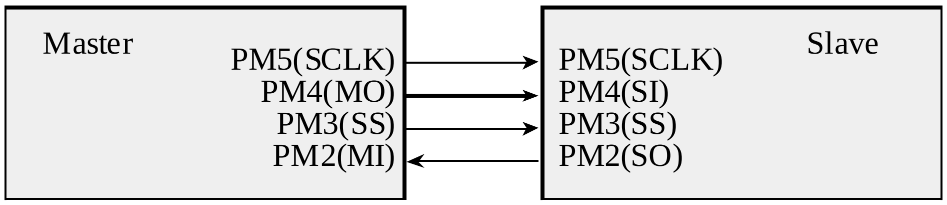
(5) Question 4. If the LED current is small enough the LED can be interfaced directly to the 6812 as shown below



Assume the LED voltage is 2V and its current is 1 mA. For this interface to work, which inequality must be true? *PLACE THE CORRECT LETTER ON THE ANSWER PAGE.*

A) $I_{IH} > 1mA$	**E)** $I_{IH} < 1mA$
B) $I_{IL} > 1mA$	**F)** $I_{IL} < 1mA$
C) $I_{OH} > 1mA$	**G)** $I_{OH} < 1mA$
D) $I_{OL} > 1mA$	**H)** $I_{OL} < 1mA$

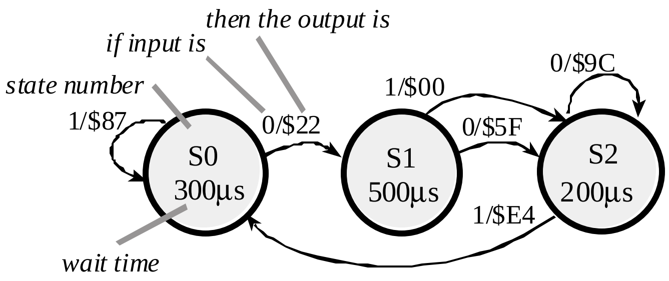
(5) Question 5. Two 6812's are to be interfaced together using their SPI ports. If the master 6812 uses CPHA=CPOL=0 mode, what values should the slave 6812 use for CPHA and CPOL?



(5) Question 6. The software outputs to a 5V stepper motor the sequence 5,6,10,9,5,6,10,9… such that one new output occurs every 1 ms. With this pattern, the stepper motor spins clockwise at 100 RPM. What will be the consequence of increasing the stepper motor voltage from +5V to +6V?

A) The motor will spin faster.
B) The motor will spin slower.
C) The motor will spin at the same speed, but with increased torque.
D) There will be no change in speed or torque.
E) The motor will spin counterclockwise at the same speed and torque.

(50) Question 7. You will implement this Mealy finite state machine. There is one digital input signal (connected to PTM bit 0) and eight digital output signals (connected to PTT). The sequence is wait, input, output, go to next state, wait, input, output, go to next state, wait… where the waiting occurs using **output compare interrupt 0**. You may assume the system is running at 4 MHz, i.e., the PLL was not activated. State **`S0`** is the initial state.



For this question, don't worry about being friendly. You will write the entire software system to run this FSM. You must use the following data structure that defines the FSM. After initialization, all input, output, and waiting occur in the output compare interrupt service routine. You cannot call any functions, unless you explicitly define those functions in your solution. The main program and FSM data structure will be as follows, and these cannot be changed.

```
const struct State{
  unsigned short Time;          // Time in usec to wait
  unsigned char Out[2];         // Output to Port T
  const struct State *Next[2];}; // Next if input=0,1
typedef const struct State StateType;
#define S0 &fsm[0]
#define S1 &fsm[1]
#define S2 &fsm[2]
StateType fsm[3]={
     {300,{0x22,0x87},{S1,S0}},  // S0
     {500,{0x5F,0x00},{S2,S2}},  // S1
     {200,{0x9C,0xE4},{S2,S0}}   // S2
};
StateType *Pt;  // Current State
```

```
void main(void){
  InitFSM();    // initialize the FSM and OC0 interrupts
  for(;;) {};
}
```

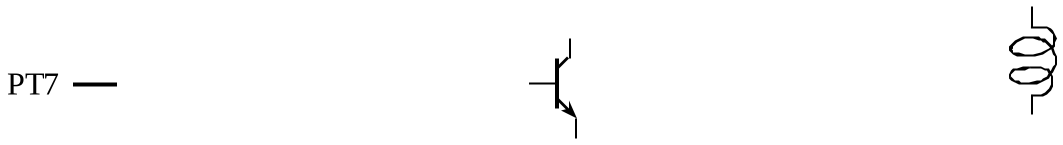
Other than the **`for(;;)`** *statement in this main program, there can be NO backward jumps in this solution.*

Jonathan W. Valvano First:_________________ Last:_____________________

April 11, 2005, 1:00pm-1:50pm. This is a closed book exam. You have 50 minutes, so please allocate your time accordingly. ***Please read the entire quiz before starting***. Only this piece of paper (pages 5 and 6) will be turned in.

(10) Question 1. Give the letters in time-sequence order (not all letters will be used)

(15) Question 2. Show the interface circuit.



(10) Question 3. Give the equation

(5) Question 4. Give the letter A, B, C, D, E, F, G or H

(5) Question 5. Give the values of CPHA and CPOL required in the slave.

(5) Question 6. Give the letter A, B, C, D or E

(25) Question 7a. Show the **`InitFSM()`** function that initializes output compare 0 and the FSM.

(25) Question 7b. Show the **`output compare 0 ISR`** that executes the finite state machine.