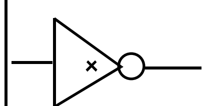
PTT is 8-bit bi-directional I/O port
DDRT is the associated direction register for Port T (0 means input, 1 means output)
PTM is 6-bit bi-directional I/O port
DDRM is the associated direction register for Port M (0 means input, 1 means output)

TSCR1 is the first 8-bit timer control register
    bit 7 **TEN**, 1 allows the timer to function normally, 0 means disable timer including **TCNT**
TSCR2 is the second 8-bit timer control register
    bits 2,1,0 are **PR2**, **PR1**, **PR0**, which select the rate, let **n** be the 3-bit number formed by **PR2**, **PR1**, **PR0**
    without PLL **TCNT** is $4MHz/2^n$, with PLL **TCNT** is $24MHz/2^n$, **n** ranges from 0 to 7
TCNT is 16-bit up counter
TIOS is the 8-bit output compare select register, one bit for each channel (1 = output compare, 0 = input capture)
TIE is the 8-bit output compare arm register, one bit for each channel (1 = armed, 0 = disarmed)
TC0 TC1 TC2… TC7 are the eight 16-bit output compare registers, one register for each channel
TFLG1 is the 8-bit flag register, one bit for each channel,
    (with output compare, flags are set when **TCNT** equals **TC0 TC1 TC2… TC7**)
    flags become zero when software writes a 1 to it (e.g., **TFLG1=0x08;** clears channel 3 flag)
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** .

ATDDIEN ADC digital enable register, 1 to make corresponding pin digital, 0 to make corresponding pin analog
PTAD is 8-bit bi-directional I/O port
DDRAD is the associated direction register for digital pins of Port AD (0 means input, 1 means output)

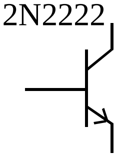
```
0xFFD6    interrupt 20 SCI
0xFFDE    interrupt 16 timer overflow
0xFFE0    interrupt 15 timer channel 7
0xFFE2    interrupt 14 timer channel 6
0xFFE4    interrupt 13 timer channel 5
0xFFE6    interrupt 12 timer channel 4
0xFFE8    interrupt 11 timer channel 3
0xFFEA    interrupt 10 timer channel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

7406



V_{OL} = 0.5V
I_{OL} = 40 mA
V_{IL} = 0.7V
I_{IL} = 1.6mA

2N2222



V_{ce} = 0.3V
V_{be} = 0.6V
h_{fe} = 100
I_{ce} =500mA max

9S12C32 parameters
I_{OL} = 10mA, I_{OH} = 10mA, I_{IL} = 1μA, I_{IH} = 1μA,
V_{OL} = 0.8V, V_{OH} = 4.2V, V_{IL} = 1.75V, V_{IH} = 3.25 V

Place your answers on pages 5 and 6.

For questions 1-5, classify each debugging technique *as A B or C.*
A) nonintrusive
B) minimally intrusive
C) highly intrusive

(5) Question 1. Adding this code to an interrupt service routine, then observing PM0 on an oscilloscope or logic analyzer.

```
PTM ^= 0x01;
```

(5) Question 2. Observing the four stepper motor control signals using a logic analyzer.

(5) Question 3. Adding this code to an interrupt service routine, then observing the output on Hyperterminal.

```
SCI_OutUDec(time); SCI_OutUDec(data); SCI_OutChar(CR);
```

(5) Question 4. Adding this code to an interrupt service routine, then observing **`Count`** using Periodical mode on the Metrowerks debugger.

```
Count++;
```

(5) Question 5. Adding a breakpoint, then single stepping the program.

---

(5) Question 6. A signed fixed point system has a range of values from -49.999 to +49.999 with a resolution of 10^{-3}. Note: 10^{-3} equals 0.001. With which of the following data types should the software variables be allocated? When more than one answer is possible choose the most space efficient type.

A) **`unsigned char`** **D)** **`char`** **G)** **`float`**
B) **`unsigned short`** **E)** **`short`** **H)** **`double`**
C) **`unsigned long`** **F)** **`long`**

(5) Question 7. Consider a situation where two periodic output compare interrupts both increment the same 16-bit global variable, **`Count`**. Do these read-modify-write sequences constitute a critical section?

```
unsigned short Count; // total number of interrupts
```

`void interrupt 8 OC0han(void){`
`  TFLG1 = 0x01;`
`  TC0 = TC0+1000;`
`  Count = Count+1;`
`}`
	`void interrupt 9 OC1han(void){`
	`  TFLG1 = 0x02;`
	`  TC1 = TC1+2000;`
	`  Count = Count+1;`
	`}`

Answer yes or no. If yes, specify how you would change the system to correct the error. If no, justify why there can be no error.

For question 8-12, consider the following simple C program.

```
const short aa=1000;
static short bb=1000;
short add3(short cc){
static short dd;
  dd = bb+cc;
  return(dd);
}
void main(void){ short ee;
  ee = add3(aa);
}
```

(2) Question 8. Where is **aa** allocated?

A) EEPROM B) global RAM
C) Reg D D) stack RAM

(2) Question 9. Where is **bb** allocated?

A) EEPROM B) global RAM
C) Reg D D) stack RAM

(2) Question 10. Where is **cc** allocated at the time of the function call (not while it is executing the body of the function, but rather when **jsr add3** is executed)

A) EEPROM B) global RAM
C) Reg D D) stack RAM

(2) Question 11. Where is **dd** allocated?

A) EEPROM B) global RAM
C) Reg D D) stack RAM

(2) Question 12. Where is **ee** allocated?

A) EEPROM B) global RAM
C) Reg D D) stack RAM

---

(5) Question 13. Assume output compare 7 interrupts are armed and enabled. Which event causes an interrupt to occur after the next instruction is executed?

A) The software executes **TFLG1 = 0x80;**
B) The software executes **TC7 = TC7+1000;**
C) The software executes **asm rti**
D) The software executes **asm sei**
E) The software executes **TFLG1 &= ~0x80;**
F) The hardware recognizes **TCNT** is equal to zero
G) The hardware recognizes **TCNT** is equal to **TC7**
H) The hardware recognizes **TCNT** is equal to **TC7+1000**
I) The hardware clears bit 7 in the **TFLG1** register

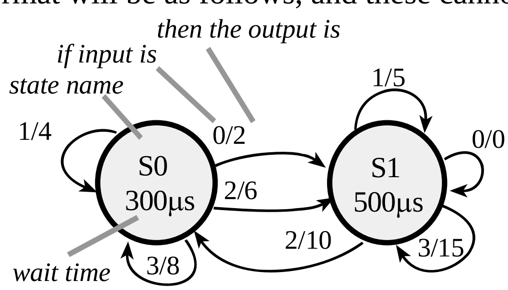
(5) Question 14. A high-efficiency red LED voltage requires 2 V at 1 mA to activate. Interface this LED to the 9S12C32 port pin PM0, such that when the software outputs a zero, the LED comes on, and when the software outputs a one, the LED goes off. If more than one possibility exists, choose the cheapest method. Label all interface components and resistor values. You can specify resistor values using an equation, rather than calculating the exact number.

(10) Question 15. Interface a 5V unipolar stepper to the 9S12C32. You need to show only one of the 4 coils. To activate, the coil needs 50 mA of current. Include protection against back EMF. Label all interface components and resistor values. You can specify resistor values using an equation, rather than calculating the exact number.

(35) Question 16. You will implement this Mealy finite state machine. There are two digital input signals (connected to Port M pins PM1, PM0) and four digital output signals (connected to Port M pins PM5, PM4, PM3, PM2). The controller sequence is

…input, output, go to next state, wait, input, output, go to next state, wait…

where the waiting occurs using **output compare interrupt 2**. You may assume the system is running at 4 MHz, i.e., the PLL was not activated. State **`S0`** is the initial state. You will write the entire software system to run this FSM. You must use the following structure that defines the format of 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 format will be as follows, and these cannot be changed.




```
const struct State{
  unsigned short Time;          // Time in usec to wait
  unsigned char Out[4];         // Output to Port M
  const struct State *Next[4];}; // Next if input=0,1,2,3
typedef const struct State StateType;
StateType *Pt;  // Current State
#define S0 &fsm[0]
#define S1 &fsm[1]
void main(void){
  InitFSM();    // initialize the FSM and OC2 interrupts
  for(;;) {};
}
```

Other than the **`for(;;)`** *statement in this main program, there can be NO backward jumps in this solution. You may not use* **`Timer_Wait()`**.

Jonathan W. Valvano First:__________________ Last:______________________

March 7, 2007, 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.

Question 1. A B or C		**Question 8.** A B C or D	
Question 2. A B or C		**Question 9.** A B C or D	
Question 3. A B or C		**Question 10.** A B C or D	
Question 4. A B or C		**Question 11.** A B C or D	
Question 5. A B or C		**Question 12.** A B C or D	
Question 6. A - H		**Question 13.** A - I	
Question 7. Yes/no (why)			

(5) Question 14. Show the LED interface.

(10) Question 15. Show the interface to one of the four stepper motor coils

(10) Question 16a. Show the FSM structure which is 1-1 to the state graph

```
StateType fsm[2]={
{[    ],{[    ],[    ],[    ],[    ]},{[    ],[    ],[    ],[    ]}},
{[    ],{[    ],[    ],[    ],[    ]},{[    ],[    ],[    ],[    ]}}
};
```

(12) Question 16b. Show the **`InitFSM()`** function that initializes output compare 2 and the FSM.

(13) Question 16c. Show the **`output compare 2 ISR`** that runs the finite state machine.