This is a closed book exam. You have 50 minutes, so allocate your time accordingly.

Please read the entire quiz before starting.

Please read and affirm our honor code:
“The core values of The University of Texas at Austin are learning, discovery, freedom, leadership, individual opportunity, and responsibility. Each member of the university is expected to uphold these values through integrity, honesty, trust, fairness, and respect toward peers and community.”

Signed: ____________________________________ November 21, 2008

(5) Question 1. List the three conditions that must be true for an RDRF interrupt to occur?

(5) Question 2. Consider a right-justified unsigned 10-bit ADC with a range of 0 to +10V. What ADC value would you get if the analog input were 1.00V?

(10) Question 3. Consider a serial port operating with a baud rate of 1000 bits per second. Draw the waveform occurring at the PS1 output (voltage levels are +5 and 0) when the ASCII ‘5’ ($35) is transmitted on SCI1. The protocol is 1 start, 8 data and 1 stop bit.

| PS3 | t -1 0 1 2 3 4 5 6 7 8 9 10 11 12 ms |
Questions 4 and 5 involve the following assembly program involving one 8-bit parameter passed on the stack and one 16-bit local variable, also on the stack.

```assembly
main lds #$4000
    movb #100,1,-sp  ; pass 8-bit in parameter on stack
    jsr sub2
    ins             ; balance stack, discarding the in parameter
here bra here
in set xxx      ; binding of 8-bit input parameter
cnt set yyy      ; binding of 16-bit local variable
sub2 leas -2,sp  ; allocate 16-bit local variable called cnt
    pshx        ; save register X
****body of the subroutine
; ........other stuff....... 
    ldab in,sp   ; get a copy of in parameter
    stx cnt,sp ; store into local variable cnt
; .........other stuff....... 
****end of body
    pulx        ; restore register X
    leas 2,sp ; deallocate cnt
    rts         ; return
```

(5) **Question 4.** What value should you use in the **xxx** position to implement the binding of the parameter, **in**?

(5) **Question 5.** What value should you use in the **yyy** position to implement the binding of the local variable, **cnt**?

(10) **Question 6.** Assuming the SCI0 has been previously initialized, write a busy-wait subroutine that outputs one 8-bit byte. The parameter is passed call by reference using RegX.

```assembly
;input: RegX points to data,  Output: none
SCI_OutChar
```
(15) Question 7. This Fifo queue has 8 allocated locations and can hold up to seven 8-bit data values. The picture shows it currently holding three values (shaded). The Fifo and its two pointers are defined in RAM. When the pointers are equal the Fifo is empty.

Fifo rmb 8 ; allocates 8 holding up to 7 values
GetPt rmb 2 ; points to oldest data
PutPt rmb 2 ; points to place to put next

This function initializes the Fifo

Fifo_Init ldx #Fifo
    stx GetPt
    stx PutPt ; Fifo is empty
    rts

Write an assembly subroutine, Fifo_Get, that implements the Get operation. A result code is returned in RegA. If RegA=1, then a byte was successfully removed, and the data is returned in RegB. If RegA=0, no data could be removed from the fifo because it was previously empty at the time of the call.

; input: none,   Output: RegA=success, RegB=data removed
Fifo_Get
The following 9S12 assembly program implements a one-input four-output finite state machine. The input is on Port J bit 0 and the outputs are on Port H bits 3,2,1,0 and Port T bits 3,2,1,0.

```
org $4000 ; Put in ROM
Stop fcb 1,2 ; Output
fdb Stop,Go ; Next
Go fcb 3,4
fdb sss,ttt
Turn fcb 5,6
fdb Stop,Go
Main lds #$4000
  bset DDRH,#$0F ; PH3-0 first out
  bset DDRT,#$0F ; PT3-0 second out
  bclr DDRJ,#$01 ; PJ0 is input
  ldy #Stop ; RegY is the State pointer
FSM yyy ; RegA, RegB are output values for this state
  staa PTH ; Perform the first output to PTH
  stab PTT ; Perform the second output to PTT
  ldab PTJ ; Read input
  andb #$01 ; just interested in bit 0
  lslb ; 2 bytes per 16 bit address
  aby ; add 0,2 depending on input
  zzz ; Next state depending on input
bra FSM
```

(5) **Question 8.** What should you put in the *sss, ttt* positions?

(5) **Question 9.** What instruction (op code and operand) goes in the *yyy* position?

(5) **Question 10.** What instruction (op code and operand) goes in the *zzz* position?

(30) **Question 11.** Assume the PLL is not active, and the E clock is 8 MHz. Write software using output compare 5 interrupts to produce a 1 kHz squarewave on PH0. PH0 is low for 500 μsec, then high for 500 μsec, repeated over and over. Include ALL software for this system: main, initialization, output compare 5 interrupt service routine, the output compare 5 interrupt vector, and reset vector. The main program initializes the stack, initializes PH0, activates output compare 5 interrupts, and then performs a do-nothing loop. The output compare 5 interrupt service routine performs the output to PH0. Global variables are allowed, but not needed.
org $4000 ; EEPROM
main lds #$4000 ; initialize stack
; initialize PH0, and output compare 5

loop bra loop ; main program does nothing

; output compare 5 interrupt service routine
OC5han

; set the output compare interrupt vector
ab OUT 8-bit decrement RegB
abx OUT unsigned add RegX=RegX+RegB
aby OUT unsigned add RegY=RegY+RegB
adca 8-bit add with carry to RegA
adcb 8-bit add with carry to RegB
adda 8-bit add to RegA
addb 8-bit add to RegB
addd 16-bit add to RegD
anda 8-bit logical and to RegA
andb 8-bit logical and to RegB
andc 8-bit logical and to RegC
asl/lsl 8-bit left shift Memory
aslb/lslb 8-bit arith left shift RegB
asl/dlsl 16-bit left shift RegD
asr 8-bit arith right shift Memory
asra 8-bit arith right shift to RegA
asrb 8-bit arith right shift to RegB
bclr branch if carry clear
bclr bit clear in memory  bclr PTT, #001
bcs branch if carry set
beq branch if result is zero (Z=1)
bege branch if signed >
bgnd enter background debug mode
bgt branch if signed >
bi branch if unsigned >
bn branch if unsigned ≥
bita 8-bit and with RegA, sets CCR
bitb 8-bit and with RegB, sets CCR
ble branch if signed ≤
blb branch if unsigned <
blsi branch if signed <
blt branch if signed <
blmi branch if result is negative (N=1)
br branch if result is nonzero (Z=0)
bpl branch if result is positive (N=0)
brs branch always
brcr branch if bits are clear  brcr PTT, #001,loop
brn branch never
brset branch if bits are set  brset PTT, #001,loop
bset bit set clear in memory  bset PTT, #004
bsr branch to subroutine
bvc branch if overflow clear
bsv branch if overflow set
call subroutine in expanded memory
cba 8-bit compare RegA with RegB
clc clear carry bit, C=0
clc clear I=0, enable interrupts
clr 8-bit memory clear
clra RegA clear
clrb RegB clear
clv clear overflow bit, V=0
cmpa 8-bit compare RegA with memory
cmpb 8-bit compare RegB with memory
com 8-bit logical complement to memory
coma 8-bit logical complement to RegA
comb 8-bit logical complement to RegB
cpd 16-bit compare RegD with Memory
cpx 16-bit compare RegX with Memory
cpy 16-bit compare RegY with Memory
da 8-bit decrement memory
daa 8-bit decimal adjust accumulator
dbdec decrement and branch if result=0  dbdec Y,loop
dbne decrement and branch if result≠0  dbne A,loop
dec 8-bit decrement memory
decb 8-bit decrement RegB
decd 8-bit decrement RegD
den 16-bit decrement RegX
dey 16-bit decrement RegY
div RegY=RegY/RegX, unsigned divide
divs RegY=RegY/RegX, signed divide
emac 16 by 16 signed multiply, 32-bit add
emax 16-bit unsigned maximum in RegD
emax 16-bit unsigned maximum in memory
en 16-bit unsigned minimum in RegD
emim 16-bit unsigned minimum in memory
emul RegY=RegY*RegD unsigned multiply
emuls RegY=RegY*RegD signed multiply
eror 8-bit logical exclusive or to RegA
etbl 16-bit look up and interpolation
exg exchange register contents  exg X,Y
fddiv unsigned fdiv, X=(65536*D)/X
fdom 16-bit increment and branch if result=0 fdom Y,loop
fdom increment and branch if result≠0  fdom A,loop
fdiv 16-bit divided, X=D/X, D=remainder
fdivs 16-bit signed divide, X= D/X, D= remainder
inc 8-bit increment memory
incb 8-bit increment RegB
incd 8-bit increment RegD
ins 16-bit increment RegSP
inx 16-bit increment RegX
iny 16-bit increment RegY
jmp jump always
jcb jump to subroutine
jcs long branch if carry clear
jcsb long branch if carry set
jeq long branch if result is zero
jeqe long branch if result is zero
jbege long branch if signed >
jbg long branch if signed >
jbhi long branch if unsigned >
jbhs long branch if unsigned ≥
jbhi long branch if signed ≤
jblo long branch if unsigned <
jbls long branch if unsigned <
jbbit long branch if signed <
jbmi long branch if result is negative
jbne long branch if result is nonzero
jbp long branch if result is positive
jbra long branch always
jlbr long branch never
jlbc long branch if overflow clear
jlbs long branch if overflow set
jdax 8-bit load memory into RegA
jdb 8-bit load memory into RegB
jdd 16-bit load memory into RegD
jdbs 8-bit load memory into RegSP
jdx 16-bit load memory into RegX
jdy 16-bit load memory into RegY
jsw load effective addr to SP jsw 2,sp
jswa 16-bit load effective addr to X jswa 2,x
jswb 16-bit load effective addr to Y jswb 2,y
jsr 8-bit logical right shift memory
jsra 8-bit logical right shift RegA
jsrb 8-bit logical right shift RegB
jsrd 16-bit logical right shift RegD
maxa 8-bit unsigned maximum in RegA
maxm 8-bit unsigned maximum in memory
mem determine the membership grade
min 8-bit unsigned minimum in RegA
minm 8-bit unsigned minimum in memory
movb 8-bit move memory to memory movb #100,PTT
movw 16-bit move memory to memory  movw #13, SCIBD
mul RegD=RegA*RegB
neg 8-bit 2's complement negate memory
nega 8-bit 2's complement negate RegA
negb 8-bit 2's complement negate RegB
ora 8-bit logical or to RegA
orab 8-bit logical or to RegB
orcc 8-bit logical or to RegCC
psha push 8-bit RegA onto stack
pshb push 8-bit RegB onto stack
pshc push 8-bit RegCC onto stack
pshd push 16-bit RegD onto stack
pshx push 16-bit RegX onto stack
pshy push 16-bit RegY onto stack
pula pop 8 bits off stack into RegA
pulb pop 8 bits off stack into RegB
pulc pop 8 bits off stack into RegCC
puld pop 16 bits off stack into RegD
pulx pop 16 bits off stack into RegX
puly pop 16 bits off stack into RegY
rev Fuzzy logic rule evaluation
revv weighted Fuzzy logic rule evaluation
rol 8-bit roll shift left Memory
rola 8-bit roll shift left RegA
rolb 8-bit roll shift left RegB
ror 8-bit roll shift right Memory
rorb 8-bit roll shift right RegA
rorc 8-bit roll shift right RegB
rtc return sub in expanded memory
rti return from interrupt
rts return from subroutine
sba 8-bit subtract RegA-RegB
sbca 8-bit subtract with carry from RegA
sec set carry bit, C=1
sei set I=1, disable interrupts
sev set overflow bit, V=1
sex sign extend 8-bit to 16-bit reg sex B, D
staa 8-bit store memory from RegA
stab 8-bit store memory from RegB
std 16-bit store memory from RegD
ststa 16-bit store memory from SP
stx 16-bit store memory from RegX
styx 16-bit store memory from RegY
suba 8-bit subtract from RegA
subb 8-bit subtract from RegB
subd 16-bit subtract from RegD
swi software interrupt, trap
tab transfer A to B
tap transfer A to CC
tba transfer B to A
tbeq test and branch if result=0 tbeq Y,loop
tbl 8-bit look up and interpolation
tbne test and branch if result≠0 tbne A,loop
tfr transfer register to register tfr X, Y
tpa transfer CC to A
trap illegal instruction interrupt
trap illegal op code, or software trap
tst 8-bit compare memory with zero
tsta 8-bit compare RegA with zero
tstb 8-bit compare RegB with zero
tsx transfer S to X
tsy transfer S to Y
tsx transfer X to S
tsys transfer Y to S

Example | Mode | Effective Address |
---------|------|------------------|
ldaa #u  | immediate | No EA |
ldaa u   | direct   | EA is 8-bit address |
ldaa U   | extended | EA is a 16-bit address |
ldaa m,r | 5-bit index | EA=r+m (-16 to 15) |
ldaa v,r+ | pre-incr | r=r+v, EA=r (1 to 8) |
ldaa v,r- | pre-dec  | r=r-v, EA=r (1 to 8) |
ldaa v,r+ | post-inc  | EA=r, r=r+v (1 to 8) |
ldaa A,r  | Reg A offset | EA=r+A, zero padded |
ldaa B,r  | Reg B offset | EA=r+B, zero padded |
ldaa D,r  | Reg D offset | EA=r+D |
ldaa q,r  | 9-bit index | EA=r+q |
ldaa W,r  | 16-bit index | EA=r+W |
ldaa [D,r] | D indirect | EA=[r+D] |
ldaa [W,r] | indirect | EA=[r+W] |

Freescale 6812 addressing modes r is X, Y, SP, or PC

Pseudo op | Meaning |
-----------|---------|
= equ set  | Where to put subsequent code |
de.c db fcb .byte | Allocate byte(s) with values |
dcc cc    | Create an ASCII string |
dc.w dw fdb .word | Allocate word(s) with values |
dc.l dl .long | Allocate 32-bit with values |
ds ds.b rmb .blkb | Allocate byte(s) without init |
ds w .blkw | Allocate word(s) without init |

Vector | Interrupt Source | Arm |
--------|-----------------|-----|
FFFF    | Reset           | None |
FFFF8   | Trap            | None |
FFFF6   | SWI             | None |
FFFF0   | Real time interrupt | CRGINT.RTIE |
SFFEE   | Timer channel 0 | TIE.C0I |
SFFEC   | Timer channel 1 | TIE.C1I |
SFFEA   | Timer channel 2 | TIE.C2I |
SFFED   | Timer channel 3 | TIE.C3I |
SFFEE   | Timer channel 4 | TIE.C4I |
SFFE4   | Timer channel 5 | TIE.C5I |
SFFE2   | Timer channel 6 | TIE.C6I |
SFFE0   | Timer channel 7 | TIE.C7I |
SFFD    | Timer overflow  | TSCR2.TO1 |
SFFD6   | SCI0 TDRE, RDRF | SCI0CR2.TIE.RIE |
SFFD4   | SCI1 TDRE, RDRF | SCI1CR2.TIE.RIE |
SFFCE   | Key Wakeup J    | PIEJ.[7,6,1,0] |
SFFCC   | Key Wakeup H    | PIEH.[7:0] |
SFFKE   | Key Wakeup P    | PIEP.[7:0] |

Interrupt Vectors.
### Bit 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 Bit 0

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>Bit 0</th>
<th>TCNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEN</td>
<td>TSWAI</td>
<td>TSFRZ</td>
<td>TFFCA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C7I</td>
<td>C6I</td>
<td>C5I</td>
<td>C4I</td>
<td>C3I</td>
<td>C2I</td>
<td>C1I</td>
<td>C0I</td>
</tr>
<tr>
<td>TOI</td>
<td>0</td>
<td>PUPT</td>
<td>RDRF</td>
<td>TCRE</td>
<td>PR2</td>
<td>PR1</td>
<td>PR0</td>
</tr>
<tr>
<td>C7F</td>
<td>C6F</td>
<td>C5F</td>
<td>C4F</td>
<td>C3F</td>
<td>C2F</td>
<td>C1F</td>
<td>C0F</td>
</tr>
<tr>
<td>TOF</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>Bit 0</th>
<th>TC0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>Bit 0</td>
<td>TC1</td>
</tr>
<tr>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>Bit 0</td>
<td>TC2</td>
</tr>
<tr>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>Bit 0</td>
<td>TC3</td>
</tr>
<tr>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>Bit 0</td>
<td>TC4</td>
</tr>
<tr>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>Bit 0</td>
<td>TC5</td>
</tr>
<tr>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>Bit 0</td>
<td>TC6</td>
</tr>
<tr>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>Bit 0</td>
<td>TC7</td>
</tr>
</tbody>
</table>

### Additional Registers

- **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 8MHz/2^n, with PLL TCNT is 24MHz/2^n, n ranges from 0 to 7.
- **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).

---

Jonathan W. Valvano  
November 21, 2008  
10:00am-10:50am