First:____________________ Last:______________________

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:

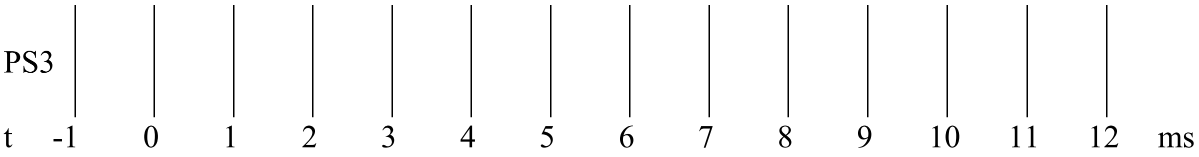
"The core values of The University of Texas at Austin are learning, discovery, freedom, leadership, individual opportunity, and responsibility. Each member of the university is expected to uphold these values through integrity, honesty, trust, fairness, and respect toward peers and community."

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.



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.

```
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.......
     ldaa 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.

```
;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.

Address	Contents	
$3900	**$00**	
$3901	**$01**	
$3902	**$02**	
$3903	**$12**	
$3904	**$56**	← **GetPt**
$3905	**$78**	
$3906	**$34**	
$3907	**$66**	← **PutPt**

```
      org  $3900
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
```

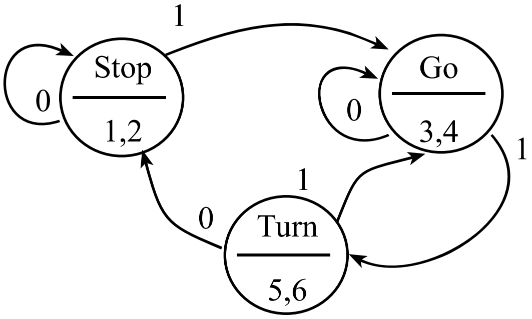
Write an assembly subroutine, `Fifo_Get`, that implements the Get operation. A result code is returned in RegA. If RegA=1, then a byte was successfully removed, and the data is returned in RegB. If RegA=0, no data could be removed from the fifo because it was previously empty at the time of the call.

```
;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
```

aba	8-bit add RegA=RegA+RegB	
abx	unsigned add RegX=RegX+RegB	
aby	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	
andcc	8-bit logical and to RegCC	
asl/lsl	8-bit left shift Memory	
asla/lsla	8-bit left shift RegA	
aslb/lslb	8-bit arith left shift RegB	
asld/lsld	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	
bcc	branch if carry clear	
bclr	bit clear in memory	bclr PTT,#$01
bcs	branch if carry set	
beq	branch if result is zero (Z=1)	
bge	branch if signed ≥	
bgnd	enter background debug mode	
bgt	branch if signed >	
bhi	branch if unsigned >	
bhs	branch if unsigned ≥	
bita	8-bit and with RegA, sets CCR	
bitb	8-bit and with RegB, sets CCR	
ble	branch if signed ≤	
blo	branch if unsigned <	
bls	branch if unsigned ≤	
blt	branch if signed <	
bmi	branch if result is negative (N=1)	
bne	branch if result is nonzero (Z=0)	
bpl	branch if result is positive (N=0)	
bra	branch always	
brclr	branch if bits are clear	brclr PTT,#$01,loop
brn	branch never	
brset	branch if bits are set	brset PTT,#$01,loop
bset	bit set clear in memory	bset PTT,#$04
bsr	branch to subroutine	
bvc	branch if overflow clear	
bvs	branch if overflow set	
call	subroutine in expanded memory	
cba	8-bit compare RegA with RegB	
clc	clear carry bit, C=0	
cli	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	
daa	8-bit decimal adjust accumulator	
dbeq	decrement and branch if result=0	dbeq Y,loop
dbne	decrement and branch if result≠0	dbne A,loop
dec	8-bit decrement memory	
deca	8-bit decrement RegA	
decb	8-bit decrement RegB	
des	16-bit decrement RegSP	
dex	16-bit decrement RegX	
dey	16-bit decrement RegY	
ediv	RegY=(Y:D)/RegX, unsigned divide	
edivs	RegY=(Y:D)/RegX, signed divide	
emacs	16 by 16 signed multiply, 32-bit add	
emaxd	16-bit unsigned maximum in RegD	
emaxm	16-bit unsigned maximum in memory	
emind	16-bit unsigned minimum in RegD	
eminm	16-bit unsigned minimum in memory	
emul	RegY:D=RegY*RegD unsigned multiply	
emuls	RegY:D=RegY*RegD signed multiply	
eora	8-bit logical exclusive or to RegA	
eorb	8-bit logical exclusive or to RegB	
etbl	16-bit look up and interpolation	
exg	exchange register contents	exg X,Y
fdiv	unsigned fract div, X=(65536*D)/X	
ibeq	increment and branch if result=0	ibeq Y,loop
ibne	increment and branch if result≠0	ibne A,loop
idiv	16-bit unsigned div, X=D/X, D=remainder	
idivs	16-bit signed divide, X=D/X, D= remainder	
inc	8-bit increment memory	
inca	8-bit increment RegA	
incb	8-bit increment RegB	
ins	16-bit increment RegSP	
inx	16-bit increment RegX	
iny	16-bit increment RegY	
jmp	jump always	
jsr	jump to subroutine	
lbcc	long branch if carry clear	
lbcs	long branch if carry set	
lbeq	long branch if result is zero	
lbge	long branch if signed ≥	
lbgt	long branch if signed >	
lbhi	long branch if unsigned >	
lbhs	long branch if unsigned ≥	
lble	long branch if signed ≤	
lblo	long branch if unsigned <	
lbls	long branch if unsigned ≤	
lblt	long branch if signed <	
lbmi	long branch if result is negative	
lbne	long branch if result is nonzero	
lbpl	long branch if result is positive	
lbra	long branch always	
lbrn	long branch never	
lbvc	long branch if overflow clear	
lbvs	long branch if overflow set	
ldaa	8-bit load memory into RegA	
ldab	8-bit load memory into RegB	
ldd	16-bit load memory into RegD	
lds	16-bit load memory into RegSP	
ldx	16-bit load memory into RegX	
ldy	16-bit load memory into RegY	
leas	16-bit load effective addr to SP	leas 2,sp
leax	16-bit load effective addr to X	leax 2,x
leay	16-bit load effective addr to Y	leay 2,y
lsr	8-bit logical right shift memory	
lsra	8-bit logical right shift RegA	
lsrb	8-bit logical right shift RegB	
lsrd	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	
mina	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	
`oraa`	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	
`revw`	weighted Fuzzy 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	
`rora`	8-bit roll shift right RegA	
`rorb`	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 sub with carry from RegA	
`sbcb`	8-bit sub with carry from RegB	
`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	
`sts`	16-bit store memory from SP	
`stx`	16-bit store memory from RegX	
`sty`	16-bit store memory from RegY	
`suba`	8-bit sub from RegA	
`subb`	8-bit sub from RegB	
`subd`	16-bit sub 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	
`txs`	transfer X to S	
`tys`	transfer Y to S	

`wai`	wait for interrupt
`wav`	weighted Fuzzy logic average
`xgdx`	exchange RegD with RegX
`xgdy`	exchange RegD with RegY

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 v,r-`	post-dec	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
`org`	Where to put subsequent code
`= equ set`	Define a constant symbol
`dc.b db fcb .byte`	Allocate byte(s) with values
`fcc`	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 bytes without init
`ds.w .blkw`	Allocate word(s) without init

Vector	Interrupt Source	Arm
$FFFE	**Reset**	None
$FFF8	**Trap**	None
$FFF6	**SWI**	None
$FFF0	**Real time interrupt**	CRGINT.RTIE
$FFEE	**Timer channel 0**	TIE.C0I
$FFEC	**Timer channel 1**	TIE.C1I
$FFEA	**Timer channel 2**	TIE.C2I
$FFE8	**Timer channel 3**	TIE.C3I
$FFE6	**Timer channel 4**	TIE.C4I
$FFE4	**Timer channel 5**	TIE.C5I
$FFE2	**Timer channel 6**	TIE.C6I
$FFE0	**Timer channel 7**	TIE.C7I
$FFDE	**Timer overflow**	TSCR2.TOI
$FFD6	**SCI0 TDRE, RDRF**	SCI0CR2.TIE,RIE
$FFD4	**SCI1 TDRE, RDRF**	SCI1CR2.TIE,RIE
$FFCE	**Key Wakeup J**	PIEJ.[7,6,1,0]
$FFCC	**Key Wakeup H**	PIEH.[7:0]
$FF8E	**Key Wakeup P**	PIEP.[7:0]

Interrupt Vectors.

Address	Bit 7	6	5	4	3	2	1	Bit 0	Name
$0040	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0	TIOS

$0044-5	Bit 15	14	13	12	11	10		Bit 0	TCNT
$0046	TEN	TSWAI	TSFRZ	TFFCA	0	0	0	0	TSCR1
$004C	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I	TIE
$004D	TOI	0	PUPT	RDPT	TCRE	PR2	PR1	PR0	TSCR2
$004E	C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F	TFLG1
$004F	TOF	0	0	0	0	0	0	0	TFLG2
$0050-1	Bit 15	14	13	12	11	10		Bit 0	TC0
$0052-3	Bit 15	14	13	12	11	10		Bit 0	TC1
$0054-5	Bit 15	14	13	12	11	10		Bit 0	TC2
$0056-7	Bit 15	14	13	12	11	10		Bit 0	TC3
$0058-9	Bit 15	14	13	12	11	10		Bit 0	TC4
$005A-B	Bit 15	14	13	12	11	10		Bit 0	TC5
$005C-D	Bit 15	14	13	12	11	10		Bit 0	TC6
$005E-F	Bit 15	14	13	12	11	10		Bit 0	TC7
$0082	ADPU	AFFC	ASWAI	ETRIGLE	ETRIGP	ETRIG	ASCIE	ASCIF	ATD0CTL2
$0083	0	S8C	S4C	S2C	S1C	FIFO	FRZ1	FRZ0	ATD0CTL3
$0084	SRES8	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0	ATD0CTL4
$0085	DJM	DSGN	SCAN	MULT	0	CC	CB	CA	ATD0CTL5
$0086	SCF	0	ETORF	FIFOR	0	CC2	CC1	CC0	ATD0STAT0
$008B	CCF7	CCF6	CCF5	CCF4	CCF3	CCF2	CCF1	CCF0	ATD0STAT1
$008D	Bit 7	6	5	4	3	2	1	Bit 0	ATD0DIEN
$008F	PAD07	PAD06	PAD05	PAD04	PAD03	PAD02	PAD01	PAD00	PORTAD0
$0090-1	Bit 15	14	13	12	11	10		Bit 0	ATD0DR0
$0092-3	Bit 15	14	13	12	11	10		Bit 0	ATD0DR1
$0094-5	Bit 15	14	13	12	11	10		Bit 0	ATD0DR2
$0096-7	Bit 15	14	13	12	11	10		Bit 0	ATD0DR3
$0098-9	Bit 15	14	13	12	11	10		Bit 0	ATD0DR4
$009A-B	Bit 15	14	13	12	11	10		Bit 0	ATD0DR5
$009C-D	Bit 15	14	13	12	11	10		Bit 0	ATD0DR6
$009E-F	Bit 15	14	13	12	11	10		Bit 0	ATD0DR7
$00C9	0	0	0	SBR12	SBR11	SBR10		SBR0	SCI0BD
$00CA	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT	SCI0CR1
$00CB	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	SCI0CR2
$00CC	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF	SCI0SR1
$00CD	0	0	0	0	0	BRK13	TXDIR	RAF	SCI0SR2
$00CF	R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0	SCI0DRL
$00D0-1	0	0	0	SBR12	SBR11	SBR10		SBR0	SCI1BD
$00D2	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT	SCI1CR1
$00D3	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	SCI1CR2
$00D4	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF	SCI1SR1
$00D5	0	0	0	0	0	BRK13	TXDIR	RAF	SCI1SR2
$00D7	R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0	SCI1DRL
$0240	PT7	PT6	PT5	PT4	PT3	PT2	PT1	PT0	PTT
$0242	DDRT7	DDRT6	DDRT5	DDRT4	DDRT3	DDRT2	DDRT1	DDRT0	DDRT
$0248	PS7	PS6	PS5	PS4	PS3	PS2	PS1	PS0	PTS
$024A	DDRS7	DDRS6	DDRS5	DDRS4	DDRS3	DDRS2	DDRS1	DDRS0	DDRS
$0250	PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0	PTM
$0252	DDRM7	DDRM6	DDRM5	DDRM4	DDRM3	DDRM2	DDRM1	DDRM0	DDRM
$0258	PP7	PP6	PP5	PP4	PP3	PP2	PP1	PP0	PTP
$025A	DDRP7	DDRP6	DDRP5	DDRP4	DDRP3	DDRP2	DDRP1	DDRP0	DDRP
$0260	PH7	PH6	PH5	PH4	PH3	PH2	PH1	PH0	PTH
$0262	DDRH7	DDRH6	DDRH5	DDRH4	DDRH3	DDRH2	DDRH1	DDRH0	DDRH
$0268	PJ7	PJ6	0	0	0	0	PJ1	PJ0	PTJ
$026A	DDRJ7	DDRJ6	0	0	0	0	DDRJ1	DDRJ0	DDRJ

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/$2^{\mathbf{n}}$, with PLL **TCNT** is 24MHz/$2^{\mathbf{n}}$, **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)