First:________________ Middle Initial: _____ Last:____________________

This is a closed book exam. You must put your answers on this piece of paper only. You have 50 minutes, so allocate your time accordingly. ***Please read the entire quiz before starting.***

(5) Question 1.

(5) Question 2.

(5) Question 3.

(5) Question 4.

(5) Question 5.

(5) Question 6.

(5) Question 7.

(5) Question 8.

(5) Question 9.

(10) Question 10.

```
Pulse bset PTT,#1  ; send pulse on PT0
      bclr PTT,#1




      rts
```

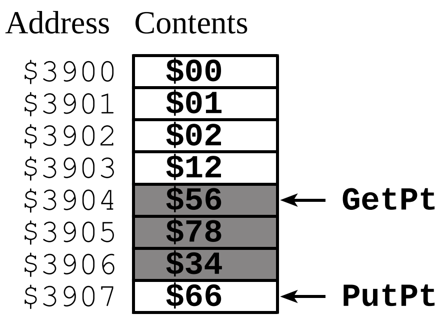
(15) Question 11.

```
Convert
```

(5) Question 12.

(15) Question 13.

```
calc                                             ;short calc(void){

                                                 ;short sum,n;

                                                 ;n = 100;

                                                 ;sum = 0;

                                                 ;do{ sum = sum+n;

                                                 ;} while(--n);

                                                 ;return(sum);}
```



This Fifo queue can hold up to eight 8-bit data values, and the picture shows it currently is holding three values (shaded).

(5) Question 1. What value is returned if we were to call **`Fifo_Get`** at this point?

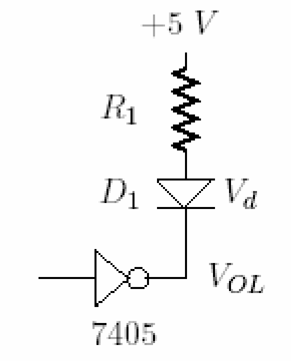
(5) Question 2. Next, assume we call **`Fifo_Put`**. What will be the new **`PutPt`** after we call **`Fifo_Put`**?

Questions 3 and 4 involve the following assembly program involving a stack frame.

```
main lds  #$4000
     ldaa #100
     psha        ; pass 8-bit in parameter on stack
     jsr  sub2
     leas 1,s    ; balance stack
here bra  here
data set  xxx    ; binding of 8-bit local variable
in   set  yyy    ; binding of 8-bit input parameter
sub2 pshx        ; save register X
     des         ; allocate 8-bit local variable called data
     tsx         ; RegX stack frame
;****body of the subroutine
     ldab in,x   ; get a copy of in parameter
     stab data,x ; store into local variable data
;****end of body
     ins         ; deallocate data
     pulx        ; restore register X
     rts         ; return
```

(5) Question 3. What value should you use in the **`xxx`** position to implement the binding of the local variable, **`data`**?

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



(5) Question 5. Specify the resistor value for R_1, assuming LED current I_d is 1 mA, the LED voltage V_d is 2.5 V, and the gate output voltage V_{OL} is 0.5V.

(5) Question 6. Which three events cause an interrupt to occur? Specify three letters in any order.
A) The software disarms the interrupt (e.g., RTIE=0)
B) The I bit in the CCR is set
C) The I bit in the CCR is clear
D) The software arms the interrupt (e.g., RTIE=1)
E) The software acknowledges the interrupt, clearing the flag (e.g., RTIF=0)
F) The software sets the flag bit (e.g., RTIF=1)
G) The hardware sets the flag bit (e.g., RTIF=1)
H) The hardware acknowledges the interrupt, clearing the flag (e.g., RTIF=0)

(5) Question 7. Consider a 10-bit ADC with a range of -10 to +10V. What is the approximate resolution of this ADC? Give units.

(5) Question 8. The 0 to 5V 10-bit ADC on the 9S12C32 uses the successive approximation conversion technique. This technique involves a series of guesses. Which will be the first guess?

A) 0 V
B) 5V/1024 = 5mV
C) 5V/256 = 20mV
D) 1.25 V
E) 2.5 V
F) 3.75 V
G) 5 V

(5) Question 9. The following ISR has a bug (I know it doesn't do anything):

```
;*****called when RTIF is set ********************
RTIhandler
      ldaa PTT
      sei
      rti

      org  $FFF0
      fdb  RTIhandler
```

A) Register A is altered by the ISR, so the main program will be confused.
B) The `sei` instruction disables interrupts, so no more interrupts will occur
C) This ISR did not acknowledge the interrupt (clear RTIF), so it will interrupt over and over continuously.
D) This ISR did not acknowledge the interrupt (clear RTIF) so no more interrupts will occur.
E) The stack is unbalanced, so it will crash.
F) The ISR didn't need to set the I bit with the `sei`, because the `rti` instruction will automatically set the I bit when the handler returns.

(15) Question 10. A complicated software system includes this subroutine

```
Pulse bset PTT,#1  ; send pulse on PT0
      bclr PTT,#1
      rts
```

Design a minimally intrusive debugging instrument that will allow you to measure how many times this subroutine has been called. You may assume that this subroutine is called less than 1000 times. Include global variable definitions, an initialization subroutine and the instrument added to `Pulse` that counts. No comments required for this question.

(15) Question 11. Write a subroutine that converts a 10-bit ADC sample into a position. The input parameter is passed call by reference in Reg Y, meaning Reg Y contains a pointer to the input 16-bit input data. The output parameter should be returned by value in RegD, meaning Reg D itself contains the result. The conversion is a linear function (Output = 2.5*Input+100). The range of input values is 0 to 1023. The output range is 100 to 2556. The following main program illustrates how data is passed into and out of your subroutine. *Comments are required.*

```
       org  $3800
Result rmb  2
       org  $4000
Data   fdb  1000
main   lds  #$4000
       ldy  #Data   ; RegY points to data
       jsr  Convert ; your subroutine
       std  Result  ; save result
```

2.5*65536= 163840
2.5/65536= 0.00003814697
65536/2.5= 26214
3.5*65536= 229376
3.5/65536= 0.00005340576
65536/3.5= 18725

The following table shows some example data. (don’t worry about rounding the LSB)

Input	2.5*Input	Output	Meaning
0	0	100	1.00 cm
100	250	350	3.50 cm
1000	2500	2600	26.00 cm
1022	2555	2655	26.55 cm

(hint: you can solve this problem one of two ways. First, you could use the **fdiv** instruction. Second, you could rewrite the formula as **Output**= 2***Input**+**Input**/2+100.

(5) Question 12. A serial port will be used to transfer 2000 bytes of information per second. The protocol is 1 start bit, 8 data bits, and 2 stop bits. What is the slowest baud rate that can handle this serial transfer?

(20) Question 13. Translate explicitly (line by line) the following C program to assembly. Both variables (**n sum**) must be stored on the stack, including symbolic binding. For each line of C, fetch necessary values off the stack, operate, and store back to the stack as appropriate. For example, **sum = sum+n;** should be implemented as read **sum** from stack, read **n** from stack, add, write result back to stack. The output parameter should be returned by value in Reg D. No additional comments required for this question.

```
short calc(void){
short sum,n;  // two 16-bit signed variables
  n = 100;
  sum = 0;
  do{ sum = sum+n;
  } while(--n);  // means decrement and branch back if not zero
  return(sum);}  // return a 16-bit result in Reg D
```

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 mult, 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 mult
emuls	RegY:D=RegY*RegD signed mult
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=rem
idivs	16-bit signed divide, X=D/X, D=rem
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
leax	16-bit load effective addr to X
leay	16-bit load effective addr to 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	addressing mode	Effective Address
ldaa #u	immediate	none
ldaa u	direct	EA is 8-bit address (0 to 255)
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-increment	r=r+v, EA=r (1 to 8)
ldaa v,-r	pre-decrement	r=r-v, EA=r (1 to 8)
ldaa v,r+	post-increment	EA=r, r=r+v (1 to 8)
ldaa v,r-	post-decrement	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 (-256 to 255)
ldaa W,r	16-bit index	EA=r+W (-32768 to 65535)
ldaa [D,r]	D indirect	EA={r+D}
ldaa [W,r]	indirect	EA={r+W} (-32768 to 65535)

Freescale 6812 addressing modes

Pseudo op				meaning
org				Specific absolute address to put subsequent object code
=	**equ**			Define a constant symbol
set				Define or redefine a constant symbol
dc.b	**db**	**fcb**	**.byte**	Allocate byte(s) of storage with initialized values
fcc				Create an ASCII string (no termination character)
dc.w	**dw**	**fdb**	**.word**	Allocate word(s) of storage with initialized values
dc.l	**dl**		**.long**	Allocate 32-bit long word(s) of storage with initialized values
ds	**ds.b**	**rmb**	**.blkb**	Allocate bytes of storage without initialization
ds.w			**.blkw**	Allocate bytes of storage without initialization
ds.l			**.blkl**	Allocate 32-bit words of storage without initialization

FDIV Fractional Divide FDIV

Operation: (D) ÷ (X) ⇒ X; Remainder ⇒ D

Description: Divides an unsigned 16-bit numerator in double accumulator D by an unsigned 16-bit denominator in index register X, producing an unsigned 16-bit quotient in X and an unsigned 16-bit remainder in D. If both the numerator and the denominator are assumed to have radix points in the same positions, the radix point of the quotient is to the left of bit 15. The numerator must be less than the denominator. In the case of overflow (denominator is less than or equal to the numerator) or division by zero, the quotient is set to $FFFF, and the remainder is indeterminate.

FDIV is equivalent to multiplying the numerator by 2^{16} and then performing 32 by 16-bit integer division. The result is interpreted as a binary-weighted fraction, which resulted from the division of a 16-bit integer by a larger 16-bit integer. A result of $0001 corresponds to 0.000015, and $FFFF corresponds to 0.9998. The remainder of an IDIV instruction can be resolved into a binary-weighted fraction by an FDIV instruction. The remainder of an FDIV instruction can be resolved into the next 16 bits of binary-weighted fraction by another FDIV instruction.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	Δ	Δ	Δ

Z: Set if quotient is $0000; cleared otherwise

V: 1 if X ≤ D
Set if the denominator was less than or equal to the numerator; cleared otherwise

C: $\overline{X15} \bullet \overline{X14} \bullet \overline{X13} \bullet \overline{X12} \bullet \ldots \bullet \overline{X3} \bullet \overline{X2} \bullet \overline{X1} \bullet \overline{X0}$
Set if denominator was $0000; cleared otherwise