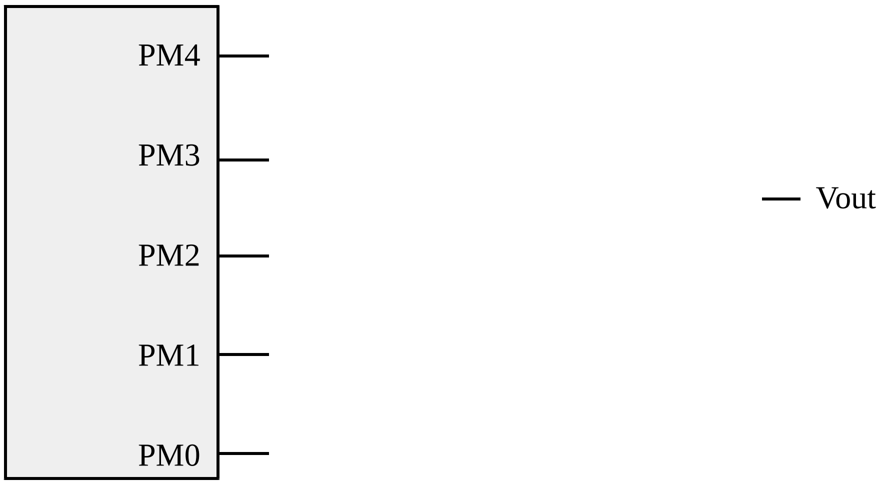
First:________________ Last:____________________

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.	
(15) Question 6.	
(5) Question 7.	
(5) Question 8.	
(5) Question 9.	
(5) Question 10.	

(5) Question 11.

(5) Question 12.

(30) Question 13.

```
       org  $3800  ; RAM
Second rmb  1      ; increment this every second

       org  $4000  ; EEPROM
main   lds  #$4000 ; initialize stack
; initialize output compare 7









loop    bra loop  ; main program does nothing

;output compare 7 interrupt service routine
OC7han









; set the output compare interrupt vector



       org $FFFE
       fdb main   ; reset vector
```

(5) Question 1. Specify the proper order of events occurring during the context switch from foreground (main program) to background (interrupt service routine, ISR). *Push registers* means Push PC, Y, X, A, B, and CCR on the stack.

A) Finish instruction, push registers, I=1, PC=vector, execute ISR.
B) Finish instruction, push registers, clear trigger flag, I=1, PC=vector, execute ISR.
C) Finish instruction, push registers, I=0, PC=vector, execute ISR.
D) Finish instruction, I=1, push registers, PC=vector, execute ISR.
E) Finish instruction, I=1, push registers, clear trigger flag, PC=vector, execute ISR.
F) Finish instruction, I=0, push registers, PC=vector, execute ISR.
G) None of the above

(5) Question 2. Consider a 12-bit ADC with a range of 0 to +5V. What is the approximate resolution of this ADC? Give units.

(5) Question 3. There are three decimal fixed-point numbers. The height, **H**, and the width, **W**, have a resolution of 0.01 cm. The area, **A**, has a fixed-point resolution of 0.01 cm^2. Let **IH** be the integer part of **H**, let **IW** be the integer part of **W**, and let **IA** be the integer part of **A**.. The goal is to calculate area, **A** = **H*****W**. Show the mathematical equation needed to calculate **IA** in terms of **IH**, **IW** and numerical constants. Your answer will look something similar to **IA** = **IH**+**IW*2**

Questions 4 and 5 involve the following assembly program involving one 16-bit parameter passed on the stack and one 8-bit local variable, also on the stack.

```
main lds  #$4000
     movw #1000,2,-sp  ; pass 16-bit in parameter on stack
     jsr  sub2
     leas 2,s    ; balance stack, discarding the in parameter
here bra  here
in   set  xxx    ; binding of 16-bit input parameter
cnt  set  yyy    ; binding of 8-bit local variable
sub2 des         ; allocate 8-bit local variable called cnt
     psha        ; save register A
     pshx        ; save register X
;****body of the subroutine
; .......other stuff.......
     ldx  in,sp  ; get a copy of in parameter
     staa cnt,sp ; store into local variable cnt
; .......other stuff.......
;****end of body
     pulx        ; restore register X
     pula        ; restore register A
     ins         ; 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`**?

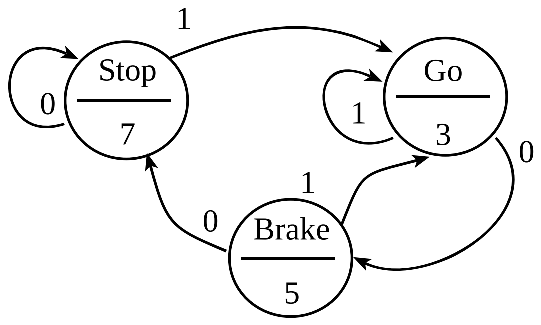
(15) Question 6. Design a 5-bit DAC converter interfaced to PM4, PM3, PM2, PM1, and PM0. The range should be 0 to +5V. You may use any resistance value from 1kΩ to 1MΩ, but please specify the resistor values.

(5) Question 7. What event triggers the start of an ADC conversion on the 6812?
A) The software writes to the ATDCTL3 register.
B) The software writes to the ATDCTL4 register.
C) The software writes to the ATDCTL5 register.
D) The ADC is automatically started by hardware.
E) Software sets the ADPU bit in the ATDCTL2 register.
F) Software read ATDSTAT0 with SCF set, followed by reading the result register.
G) None of these choices is correct.

The following 6812 assembly program implements a one-input four-output finite state machine. The input is on Port M bit 0 and the output is on Port T bits 3,2,1,0.

```
     org  $4000  Put in ROM

Stop  fcb  7          ;Output
      fdb  Stop,Go    ;Next
Go    fcb  3
      fdb  sss,ttt
Brake fcb  5
      fdb  Stop,Go
Main lds  #$4000
     bset DDRT,#$0F  ; PT3-0 outputs
     bclr DDRM,#$01  ; PM0 is input
     ldy  #Stop      ; RegY is the State pointer
FSM  yyy             ; RegB is Output value for this state
     stab PTT        ; Perform the output
     ldab PTM        ; 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?

(5) Question 11. Why do we use a *Fifo* queue in a producer/consumer system to pass data between the foreground and the background? Choose the best answer.
A) Because the *Fifo* queue data is stored on the stack, making the system reentrant.
B) Because the *Fifo* queue has an unlimited amount of storage, allowing both the producer and consumer to operate at full speed.
C) Because the *Fifo* queue temporarily stores data, decoupling the execution of the producer and the consumer. If there is room in the *Fifo*, the producer can run. If there is data in the *Fifo* the consumer can run.
D) All of A, B, and C are correct.
E) None of A, B, and C are correct.

(5) Question 12. Why do we use local variables instead of globals? Choose the best answer.
A) Local variables have a more limited scope than global variables, simplifying the interaction between modules in a modular design.
B) If storage is only needed temporarily, then local variables allow RAM memory to be reused.
C) When a subroutine is called both by the main program and by an interrupt service routine, local variables will not create a critical section (i.e., it will be reentrant).
D) All of A, B, and C are correct.
E) None of A, B, and C are correct.

(30) Question 13. Assume the PLL is not active, and the E clock is 4 MHz. Design a system that increments an 8-bit global variable called **Second**, every 1 second. Show the main program (including ritual), the output compare 7 interrupt service routine, and the output compare interrupt vector. The main program initializes **Second**=0, activates output compare 7 interrupts, and then performs a do-nothing loop. The output compare 7 interrupt service routine increments the variable **Second**. Additional global variables are allowed, but not needed.

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	stab	8-bit store memory from RegB
psha	push 8-bit RegA onto stack	std	16-bit store memory from RegD
pshb	push 8-bit RegB onto stack	sts	16-bit store memory from SP
pshc	push 8-bit RegCC onto stack	stx	16-bit store memory from RegX
pshd	push 16-bit RegD onto stack	sty	16-bit store memory from RegY
pshx	push 16-bit RegX onto stack	suba	8-bit sub from RegA
pshy	push 16-bit RegY onto stack	subb	8-bit sub from RegB
pula	pop 8 bits off stack into RegA	subd	16-bit sub from RegD
pulb	pop 8 bits off stack into RegB	swi	software interrupt, trap
pulc	pop 8 bits off stack into RegCC	tab	transfer A to B
puld	pop 16 bits off stack into RegD	tap	transfer A to CC
pulx	pop 16 bits off stack into RegX	tba	transfer B to A
puly	pop 16 bits off stack into RegY	tbeq	test and branch if result=0
rev	Fuzzy logic rule evaluation		tbeq Y,loop
revw	weighted Fuzzy rule evaluation	tbl	8-bit look up and interpolation
rol	8-bit roll shift left Memory	tbne	test and branch if result≠0
rola	8-bit roll shift left RegA		tbne A,loop
rolb	8-bit roll shift left RegB	tfr	transfer register to register
ror	8-bit roll shift right Memory		tfr X,Y
rora	8-bit roll shift right RegA	tpa	transfer CC to A
rorb	8-bit roll shift right RegB	trap	illegal instruction interrupt
rtc	return sub in expanded memory	trap	illegal op code, or software trap
rti	return from interrupt	tst	8-bit compare memory with zero
rts	return from subroutine	tsta	8-bit compare RegA with zero
sba	8-bit subtract RegA-RegB	tstb	8-bit compare RegB with zero
sbca	8-bit sub with carry from RegA	tsx	transfer S to X
sbcb	8-bit sub with carry from RegB	tsy	transfer S to Y
sec	set carry bit, C=1	txs	transfer X to S
sei	set I=1, disable interrupts	tys	transfer Y to S
sev	set overflow bit, V=1	wai	wait for interrupt
sex	sign extend 8-bit to 16-bit reg	wav	weighted Fuzzy logic average
	sex B,D	xgdx	exchange RegD with RegX
staa	8-bit store memory from RegA	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

Address	Bit 7	6	5	4	3	2	1	Bit 0	Name
$0082	ADPU	AFFC	AWAI	ETRIGLE	ETRIGP	ETRIG	ASCIE	ASCIF	ATDCTL2
$0083	0	S8C	S4C	S2C	S1C	FIFO	FRZ1	FRZ0	ATDCTL3
$0084	SRES8	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0	ATDCTL4
$0085	DJM	DSGN	SCAN	MULT	0	CC	CB	CA	ATDCTL5
$0086	SCF	0	ETORF	FIFOR	0	CC2	CC1	CC0	ATDSTAT0
$008B	CCF7	CCF6	CCF5	CCF4	CCF3	CCF2	CCF1	CCF0	ATDSTAT1
$008D	Bit 7	6	5	4	3	2	1	Bit 0	ATDDIEN
$0270	PTAD7	PTAD6	PTAD5	PTAD4	PTAD3	PTAD2	PTAD1	PTAD0	PTAD
$0272	DDRAD7	DDRAD6	DDRAD5	DDRAD4	DDRAD3	DDRAD2	DDRAD1	DDRAD0	DDRAD

address	msb															lsb	Name
$0090	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ATDDR0
$0092	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ATDDR1
$0094	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ATDDR2
$0096	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ATDDR3
$0098	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ATDDR4
$009A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ATDDR5
$009C	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ATDDR6
$009E	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ATDDR7

address	msb															lsb	Name
$0044	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TCNT
$0050	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC0
$0052	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC1
$0054	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC2
$0056	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC3
$0058	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC4
$005A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC5
$005C	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC6
$005E	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC7

Address	Bit 7	6	5	4	3	2	1	Bit 0	Name
$0046	TEN	TSWAI	TSBCK	TFFCA	0	0	0	0	TSCR1
$004D	TOI	0	0	0	TCRE	PR2	PR1	PR0	TSCR2
$0040	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0	TIOS
$004C	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I	TIE
$004E	C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F	TFLG1
$004F	TOF	0	0	0	0	0	0	0	TFLG2

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/$2^{\mathbf{n}}$, 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)

Vector Address	Interrupt Source or Trigger flag	Enable	Local Arm
$FFFE	Reset	none	none
$FFEE	Timer Channel 0, C0F	I bit	TIE.C0I
$FFEC	Timer Channel 1, C1F	I bit	TIE.C1I
$FFEA	Timer Channel 2, C2F	I bit	TIE.C2I
$FFE8	Timer Channel 3, C3F	I bit	TIE.C3I
$FFE6	Timer Channel 4, C4F	I bit	TIE.C4I
$FFE4	Timer Channel 5, C5F	I bit	TIE.C5I
$FFE2	Timer Channel 6, C6F	I bit	TIE.C6I
$FFE0	Timer Channel 7, C7F	I bit	TIE.C7I