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. Digital value	
(5) Question 2. Baud rate in bits/sec	
(25) Question 3. Show subroutine	
(5) Question 4. Show the code	
(5) Question 5. List variables A,B,C,D,E,F,G	
(5) Question 6. List variables A,B,C,D,E,F,G	

(5) Question 7. Give value of `xxx`	
(5) Question 8. Give value of `xxx`	
(5) Question 9. Specify A, B, C, D, E	
(5) Question 10. Give the output sequence	
(5) Question 11. Give instruction for `yyy`	
(5) Question 12. Give value of `zzz`	
(5) Question 13. Specify A, B, C, D, E	
(5) Question 14. Give R_1 in ohms	
(5) Question 15. Specify A, B, C, D, E	

(5) Question 1. An analog voltage of 1.25 V is placed on the ADC input pin. What digital value results from the 10-bit unsigned right-justified ADC conversion?

(5) Question 2. A serial port is configured to run with a bandwidth of 1000 bytes/sec. The protocol is 8-bit data, 1 stop, and no parity. What is the baud rate of this port in bits/sec?

(15) Question 3. The **SCISR1** register contains the **TDRE** and **RDRF** flags

	7	6	5	4	3	2	1	0
R	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
W								
RESET:	1	1	0	0	0	0	0	0

TDRE — Transmit Data Register Empty Flag

TDRE is set when the transmit shift register receives a byte from the SCI data register. When TDRE is 1, the transmit data register (SCIDRH/L) is empty and can receive a new value to transmit.Clear TDRE by reading SCI status register 1 (SCISR1), with TDRE set and then writing to SCI data register low (SCIDRL).

1 = Byte transferred to transmit shift register; transmit data register empty
0 = No byte transferred to transmit shift register

RDRF — Receive Data Register Full Flag

RDRF is set when the data in the receive shift register transfers to the SCI data register. Clear RDRF by reading SCI status register 1 (SCISR1) with RDRF set and then reading SCI data register low (SCIDRL).

1 = Received data available in SCI data register
0 = Data not available in SCI data register

The **SCIDRL** register serial input/output data. Write a subroutine that inputs a CR-terminated string from the keyboard. For each character, it waits for new input using busy-wait (gadfly) synchronization. The subroutine uses call by reference parameter passing with RegY. When CR is typed, save the CR in the string and return. You don't need to write the initialization ritual.

```
CR       equ 13     ; return
SCISR1   equ $00CC  ; SCI Status Register 1
SCIDRL   equ $00CF  ; SCI Data Register Low
```

(5) Question 4. Write assembly code that allocates a 16-bit signed local variable with an initial value of 50?

Consider the following C program.

```
static short A=4;
const short B=5;
volatile short C=6;
void function(const short D, short E){
   static short F=7;
   short G=8;
}
```

(5) Question 5. List all the variables in the above C program are stored in ROM?

(5) Question 6. List all the variables in the above C program are local (stored temporarily on the stack or in a register)?

(5) Question 7. Consider the following assembly subroutine that creates a local variable called **buff** of size 25 bytes.

```
buff set xxx     ; binding
sub1 pshx        ; save register X
     leas -25,s  ; allocate buff
;****body of the subroutine
     staa buff,s ; store into buff[0]
;****end of body
     leas 25,s   ; deallocate buff
     pulx        ; restore register X
     rts         ; return
```

What value should you use in the **xxx** position to implement the proper binding of this local variable?

(5) Question 8. Consider the following main program that calls an assembly subroutine using call by value parameter passing on the stack. The subroutine uses Register X stack frame.

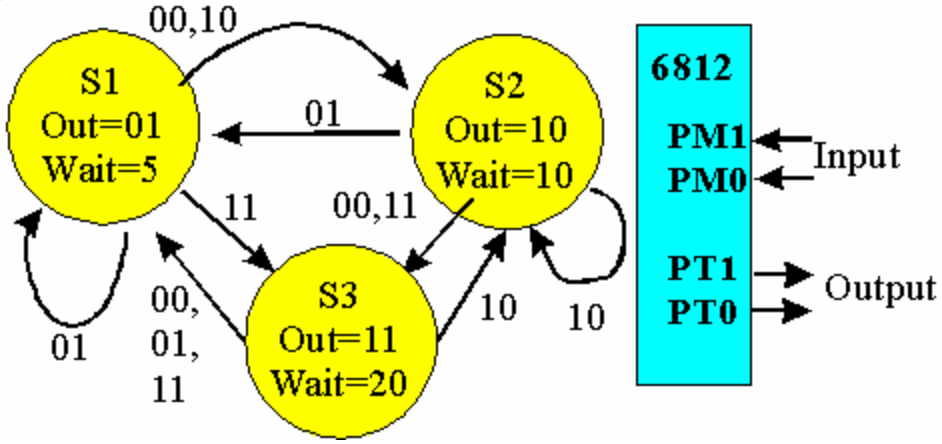
```
data set xxx     ; binding
main lds #$4000
     ldy #1000
     pshy        ; pass 16-bit data parameter on stack
     jsr sub2
     puly
     stop

sub2 pshx        ; save register X
     tsx         ; create Register X stack frame
     leas -10,sp ; allocate locals
;****body of the subroutine
     ldd data,x  ; get data parameter
;****end of body
     leas 10,sp  ; deallocate locals
     pulx        ; restore register X
     rts         ; return
```

What value should you use in the **xxx** position to implement the proper binding of this parameter?

(5) Question 9. When debugging a subroutine, we often call the subroutine with a specific set of inputs. If the outputs are incorrect, we make changes to the subroutine and call it again with the exact same inputs as before. What debugging process is this?

A) Stabilization,
B) Profiling
C) Desk check
D) Nonintrusiveness
E) Monitor

The following 6812 assembly program implements a two-input two-output finite state machine.




```
      org  $4000  Put in ROM
S1    fcb  %01    Output
      fcb  5      Wait Time
      fdb  S2,S1,S2,S3
S2    fcb  %10    Output
      fcb  10     Wait Time
      fdb  S3,S1,S2,S3
S3    fcb  %11    Output
      fcb  20     Wait Time
      fdb  S1,S1,S2,S1
Main  lds  #$4000
      bset DDRT,#$03  ; PTT1,PTT0 are LED outputs
      yyy  DDRM,#$03  ; PTM1,PTM0 are switch inputs
      ldx  #S1        ; Initial State pointer
FSM   ldab 0,x        ; Output value for this state in bits 1,0
      stab PTT
      ldaa 1,x        ; 8-bit Wait in this state
      bsr  WAIT
      ldab PTM        ; Read input
      andb #$03       ; just interested in bits 1,0
      lslb            ; 2 bytes per 16 bit address
      abx             ; add 0,2,4,6 depending on input
      ldx  zzz,x      ; Next state depending on input
      bra  FSM
```

(5) Question 10. If the input is 10, what will be the output sequence?

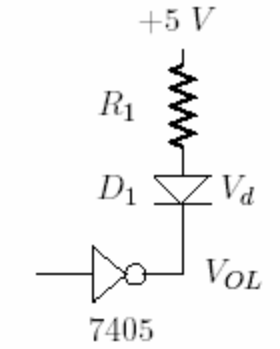
(5) Question 11. Which instruction goes in the **yyy** position?

(5) Question 12. Which number goes in the **zzz** position?

(5) Question 13. Consider a matrix with 4 rows and 6 columns, stored in column-major zero-index format. Each element is 1 byte or 8 bits. Which equation correctly calculates the address of the element at row I and column J?

A. base+I+J
B. base+4*I+J
C. base+I+4*J
D. base+6*I+J
E. base+I+6*J

(5) Question 14. Specify the resistor value for R_1, assuming LED current I_d is 1 mA, the LED voltage V_d is 1 V, and the gate output voltage V_{OL} is 0.5V.



(5) Question 15. Which registers are pushed on the stack by **`swi`** and pulled off by **`rti`**?

A. all registers but the SP
B. PC
C. PC and CCR
D. all registers including the SP
E. SP

```
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
asrb  8-bit arith right shift to RegB
bcc   branch if carry clear
bclr  clear bits in memory
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,
brn   branch never
brset branch if bits are set
bset  set bits in memory
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
dbne  decrement and branch if
result?0
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
fdiv  16-bit unsigned fractional
divide
ibeq  increment and branch if
result=0
ibne  increment and branch if
result?0
idiv  16-bit unsigned divide, X=D/X
idivs 16-bit signed divide, X=D/X
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
movw  16-bit move memory to memory
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=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
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
tbl   8-bit look up and interpolation
tbne  test and branch if result?0
tfr   transfer register to register
tpa   transfer CC to A
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+1 to X
tsy   transfer S+1 to Y
txs   transfer X-1 to S
tys   transfer Y-1 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)

Motorola 6812 addressing modes

Pseudo op	meaning
org 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