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.

(5) Question 10.

(5) Question 11.

(5) Question 12.

(5) Question 13.

(5) Question 14.
(10) Question 15.

(10) Question 16.

(10) Question 17.
For questions 1 and 2, consider the following C program.

```c
xxxx short V=4;
void function(void){ yyy short W=4;
}
```

(5) Question 1. What qualifier should be used for `xxxx` so that `V` is allocated in ROM? Select from `signed unsigned volatile const static extern`

(5) Question 2. What qualifier should be used for `yyyy` so that `W` is permanently allocated in RAM? Select from `signed unsigned volatile const static extern`

This Fifo queue can hold up to eight 16-bit data values, and the picture shows it currently is holding three values (shaded).

(5) Question 3. What value is returned if we were to call `Fifo_Get` at this point?

(5) Question 4. Next, assume we call `Fifo_Put`. What will be the new `PutPt` after we call `Fifo_Put`?

Questions 5 and 6 involve the following assembly code.

```assembly
main lds #$4000
ldy #1000
pshy        ; pass 16-bit in parameter on stack
jsr sub1
puly        ; balance stack
stop

data set xxx ; binding of 16-bit local variable
in set yyy ; binding of 16-bit input parameter
sub1 pshx ; save register X
tsx         ; RegX stack frame
lea -2,s    ; allocate 16-bit local variable called data
;****body of the subroutine
ldd in,x    ; get a copy of in parameter
std data,x  ; store into local variable data
;****end of body
lea 2,s     ; deallocate data
pulx         ; restore register X
rts         ; return
```

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

(5) Question 6. What value should you use in the `yyy` position to implement the binding of the parameter, `in`?
(5) **Question 7.** Consider a 10-bit ADC with a range of 0 to +5V. What is the approximate resolution of this ADC? Give units.

(5) **Question 8.** What term do we use to describe it when the debugging code itself makes a small but acceptable change in the behavior of a software system?

(5) **Question 9.** To verify the proper functionality of a subroutine, we write a special main program that provides a known and repeatable sequence of inputs to the subroutine under test. In this way, each time the test subroutine is changed, we can be sure the change in output values is caused by the software modification and not due to a change in input values. What is this debugging procedure called?

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 1 ;Output
fdb Stop,Run ;Next
Run fcb 5
fdb Turn,Run
Turn fcb 10
fdb ***,***
Main lds #$4000
    bset DDRT,#$0F  ; PT3-0 outputs
    bclr DDRM,#$01  ; PM0 is input
    ldx #Stop ; RegX is the State pointer
    FSM
    yyy ; RegA is Output value for this state
    staa PTT ; Perform the output
    ldab PTM ; Read input
    andb #$01 ; just interested in bit 0
    lslb ; 2 bytes per 16 bit address
    abx ; add 0,2 depending on input
    zzz ; Next state depending on input
    bra FSM
```

(5) **Question 10.** What should you put in the ***,*** positions?

(5) **Question 11.** Which instruction causes an unfriendly operation?

(5) **Question 12.** What instruction (op code and operand) goes in the yyy position?

(5) **Question 13.** What instruction (op code and operand) goes in the zzz position?

(5) **Question 14.** Specify the resistor value for \( R_1 \), assuming LED current \( I_d \) is 2 mA, the LED voltage \( V_d \) is 1.5 V, and the gate output voltage \( V_{OL} \) is 0.5V.
(10) **Question 15.** Write a subroutine that generates one conversion on channel 4 of the 9S12C32 ADC and returns the 10-bit unsigned right-justified ADC conversion in Reg X (return by value). You may assume the ADC has already been enabled with the following function.

```assembly
ADC_Init  movb #$80,ATDCTL2  ;ADPU=1 enables A/D
         movb #$08,ATDCTL3  ;sequence length=1
         movb #$04,ATDCTL4  ;10-bit A/D
         rts
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Bit 7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>Bit 0</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0082</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ADPU</td>
</tr>
<tr>
<td>$0083</td>
<td></td>
<td>S8C</td>
<td>S4C</td>
<td>S2C</td>
<td>S1C</td>
<td>FIFO</td>
<td>FRZ1</td>
<td>FRZ0</td>
<td>ATDCTL2</td>
</tr>
<tr>
<td>$0084</td>
<td></td>
<td>S8C</td>
<td>SMP0</td>
<td>PRS4</td>
<td>PRS3</td>
<td>PRS2</td>
<td>PRS1</td>
<td>PRS0</td>
<td>ATDCTL4</td>
</tr>
<tr>
<td>$0085</td>
<td></td>
<td>DJM</td>
<td>DSGN</td>
<td>SCAN</td>
<td>MULT</td>
<td>0</td>
<td>CC</td>
<td>CB</td>
<td>ATDCTL5</td>
</tr>
<tr>
<td>$0086</td>
<td></td>
<td>SCF</td>
<td>0</td>
<td>ETORF</td>
<td>FIFO</td>
<td>0</td>
<td>CC2</td>
<td>CC1</td>
<td>CC0</td>
</tr>
<tr>
<td>$008B</td>
<td></td>
<td>CCF7</td>
<td>CCF6</td>
<td>CCF5</td>
<td>CCF4</td>
<td>CCF3</td>
<td>CCF2</td>
<td>CCF1</td>
<td>ATDSTAT0</td>
</tr>
<tr>
<td>$008D</td>
<td></td>
<td>Bit 7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>Bit 0</td>
</tr>
<tr>
<td>$0270</td>
<td></td>
<td>PTAD7</td>
<td>PTAD6</td>
<td>PTAD5</td>
<td>PTAD4</td>
<td>PTAD3</td>
<td>PTAD2</td>
<td>PTAD1</td>
<td>PTAD</td>
</tr>
<tr>
<td>$0272</td>
<td></td>
<td>DRRAD7</td>
<td>DDRAD6</td>
<td>DDRAD5</td>
<td>DDRAD4</td>
<td>DDRAD3</td>
<td>DDRAD2</td>
<td>DDRAD1</td>
<td>DDRAD</td>
</tr>
<tr>
<td>address</td>
<td></td>
<td>mbs</td>
<td>lsb</td>
<td>Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0090</td>
<td></td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>$0092</td>
<td></td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>$0094</td>
<td></td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>$0096</td>
<td></td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>$0098</td>
<td></td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>$009A</td>
<td></td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>$009C</td>
<td></td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>$009E</td>
<td></td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
</tbody>
</table>

(10) **Question 16.** Write a subroutine that outputs a null-terminated string to the SCI transmitter. An address to the string is pushed on the stack (call by reference). You may assume the SCI has already been enabled with the following function.

```assembly
SCI_Init  movb #$0c,SCICR2  ;enable SCI
         movw #$1,SCIBD  ;baud rate=250000
         rts
```

<table>
<thead>
<tr>
<th>Addr</th>
<th>Bit 7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>Bit 0</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$00C8</td>
<td>BTST</td>
<td>BSPL</td>
<td>BRLD</td>
<td>SBR12</td>
<td>SBR11</td>
<td>SBR10</td>
<td>SBR9</td>
<td>SBR8</td>
<td>SCIBD</td>
</tr>
<tr>
<td>$00C9</td>
<td>SBR7</td>
<td>SBR6</td>
<td>SBR5</td>
<td>SBR4</td>
<td>SBR3</td>
<td>SBR2</td>
<td>SBR1</td>
<td>SBR0</td>
<td>SCICR2</td>
</tr>
<tr>
<td>$00CB</td>
<td>TIE</td>
<td>TCIE</td>
<td>RIE</td>
<td>ILIE</td>
<td>TE</td>
<td>RE</td>
<td>RWU</td>
<td>S BK</td>
<td>SCISR1</td>
</tr>
<tr>
<td>$00CC</td>
<td>TDRE</td>
<td>TC</td>
<td>RDRF</td>
<td>IDLE</td>
<td>OR</td>
<td>NF</td>
<td>FE</td>
<td>PF</td>
<td>SCIDRL</td>
</tr>
<tr>
<td>$00CF</td>
<td>R7T7</td>
<td>R6T6</td>
<td>R5T5</td>
<td>R4T4</td>
<td>R3T3</td>
<td>R2T2</td>
<td>R1T1</td>
<td>R0T0</td>
<td></td>
</tr>
</tbody>
</table>

An example calling sequence is

```assembly
Message  fcb "Hello world",0
Main     lds #$4000
         jsr SCI_Init
         movw #Message,2,-sp  ;push address on stack
         jsr SCI_Out  ;your subroutine
         leas 2,s  ;balance stack
```

Jonathan W. Valvano      December 2, 2005      10:00am-10:50am
**(10) Question 17.** Assume 16-bit TCNT is enabled and running. Write an SWI handler that reads the value on TCNT and returns the result in Reg X (return by value). An example calling sequence

```
swi
```

After the software interrupt returns, RegX contains the value from TCNT. The SWI vector is located at $FFFF.$
aba 8-bit add RegY=RegY+RegB
abx unsigned add RegY=RegY+RegB
aby unsigned add RegY=RegY+RegB
add 8-bit add with carry to RegA
dacb 8-bit add with carry to RegB
adda 8-bit add to RegA
addb 8-bit add to RegB
adddd 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/lasl 8-bit left shift RegA
aslb/lslb 8-bit shift left RegB
asld/lsl 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 ≥
bgt branch if signed >
bbi branch if unsigned >
bbh branch if unsigned ≥
bits 8-bit and with RegA, sets CCR
bitb 8-bit and with RegB, sets CCR
ble branch if signed ≤
blb branch if unsigned ≤
blt branch if unsigned <
blt branch if result ≤
率达 branch if result is negative (N=1)
blne branch if result is negative
blne branch if result is not zero
br branch never
brset branch if bits are set
bssr branch to subroutine
bvs branch if overflow clear
bvs branch if overflow set
lcall subroutine in expanded memory
cba 8-bit compare RegA with RegB
clb clear carry bit, C=0
cli clear I=0, enable interrupts
clr 8-bit Memory clear
clda 8-bit logical or to RegB
clrb RegB clear
clv 8-bit 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
cpa 16-bit compare RegD with memory
cpb 16-bit compare RegB with memory
cpd 16-bit compare RegD with memory
cpd 16-bit compare RegD with memory
cpp 16-bit compare RegY with memory
cr 8-bit Memory clear
cra RegA clear
crb RegB clear
crv 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
cppw 16-bit move memory to memory
mul RegD=RegD*RegB
ncy 16-bit compare RegY with memory
daa 8-bit decimal adjust accumulator
db 8-bit decrement branch if result=0
dbs 8-bit decrement branch if result>0
dc 8-bit decrement memory
decn 8-bit decrement RegA
decb 8-bit decrement RegB
des 16-bit decrement RegD
dex 16-bit decrement RegX
dey 16-bit decrement RegY
div RegY=RegY/RegX, signed divide
divs RegY=RegY/RegX, signed divide
emod 16-bit unsigned maximum in RegD
emod 16-bit unsigned maximum in RegD
emains 16-bit unsigned minimum in RegD
emains 16-bit unsigned minimum in RegD
emains 16-bit unsigned minimum in RegD
emains 16-bit unsigned minimum in RegD
emains 16-bit unsigned minimum in RegD
puls pop 16 bits off stack into RegX
puly pop 16 bits off stack into RegY
rev Fuzzy logic rule evaluation
reww 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
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 subtract with carry from RegA
sbcb 8-bit subtract 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
tbl 8-bit look up and interpolation
tbne test and branch if result ≠ 0
tfr transfer register to register
tpa transfer CC to A
rtb return sub in expanded memory
rtne test and branch if result≠0
rev weighted Fuzzy rule evaluation
revw weighted Fuzzy rule evaluation

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
dsw .blkw Allocate bytes of storage without initialization
ds1 .blkl Allocate 32-bit words of storage without initialization