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.**

<table>
<thead>
<tr>
<th>(5) Question 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(5) Question 2.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(5) Question 3.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(5) Question 4.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(5) Question 5.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(15) Question 6.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM4</td>
</tr>
<tr>
<td>PM3</td>
</tr>
<tr>
<td>PM2</td>
</tr>
<tr>
<td>PM1</td>
</tr>
<tr>
<td>PM0</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Vout</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(5) Question 7.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(5) Question 8.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(5) Question 9.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(5) Question 10.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
(5) Question 11.

(5) Question 12.

(30) Question 13.

```assembly
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 \( \text{IH} \) be the integer part of \( H \), let \( \text{IW} \) be the integer part of \( W \), and let \( \text{IA} \) be the integer part of \( A \). The goal is to calculate area, \( A = \text{IH} \cdot \text{IW} \). Show the mathematical equation needed to calculate \( \text{IA} \) in terms of \( \text{IH} \), \( \text{IW} \) and numerical constants. Your answer will look something similar to \( \text{IA} = \text{IH} + \text{IW} \times 2 \)

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.

```assembly
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.
dey 16-bit decrement RegY
dex 16-bit decrement RegX
des 16-bit decrement RegSP
decb 8-bit decrement RegB
deca 8-bit decrement RegA
dec 8-bit decrement memory
dbne A,loop
bls branch if unsigned
blo branch if unsigned <
bhs branch if unsigned ≥
bhi branch if unsigned >
bgt branch if signed >
bgnd enter background debug mode
bgt branch if signed >
bbh branch if unsigned ≥
bs branch if unsigned ≥
bits 8-bit and with RegA, sets CCR
bith 8-bit and with RegB, sets CCR
ble branch if signed ≤
bl branch if unsigned <
bls branch if unsigned ≤
bit 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
BCLR branch if bits are clear
BCLR 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
BVS branch if overflow clear
BVS branch if overflow set
call subroutine in expanded memory
cba 8-bit compare RegA with RegB
clear carry bit, C=0
cle clear I=0, enable interrupts
cle 8-bit memory clear
cr 8-bit memory clear
cr 8-bit memory clear
cr 8-bit memory clear
cr 8-bit memory clear
cr rega clear
cr 8-bit memory clear, 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 rega,loop
dec 8-bit decrement memory
deca 8-bit decrement RegA
decb 8-bit decrement RegB
dec 8-bit decrement RegSP
dex 16-bit decrement RegX
dey 16-bit decrement RegY
orc 8-bit logical or to RegCC
psha push 8-bit RegA onto stack
psb push 8-bit RegB onto stack
psch push 8-bit RegCC onto stack
psdh push 16-bit RegD onto stack
pshx push 16-bit RegX 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
rolb 8-bit roll shift left RegB
ror 8-bit roll shift right Memory
rorr 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
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
tbel test and branch if result=0
tbel test and branch if result#0
tble test and branch if result<=0
trong transfer register to register
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
tra transfer S to X
tsy transfer Y to S
trb transfer S to Y
trx transfer X to S
tsf transfer S to Reg
wai wait for interrupt
wav weighted Fuzzy logic average
xgdx exchange RegD with RegX
xgdy exchange RegD with RegY

<table>
<thead>
<tr>
<th>example</th>
<th>addressing mode</th>
<th>Effective Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldaa #u</td>
<td>immediate</td>
<td>none</td>
</tr>
<tr>
<td>ldaa u</td>
<td>direct</td>
<td>EA is 8-bit address (0 to 255)</td>
</tr>
<tr>
<td>ldaa U</td>
<td>extended</td>
<td>EA is a 16-bit address</td>
</tr>
<tr>
<td>ldaa m,r</td>
<td>5-bit index</td>
<td>EA=r+m (-16 to 15)</td>
</tr>
<tr>
<td>ldaa v,r</td>
<td>pre-increment</td>
<td>r=r+v, EA=r (1 to 8)</td>
</tr>
<tr>
<td>ldaa v,-r</td>
<td>pre-decrement</td>
<td>r=r-v, EA=r (1 to 8)</td>
</tr>
<tr>
<td>ldaa v, r</td>
<td>post-increment</td>
<td>EA=r, r=r+v (1 to 8)</td>
</tr>
<tr>
<td>ldaa v,-r</td>
<td>post-decrement</td>
<td>EA=r, r=r-v (1 to 8)</td>
</tr>
<tr>
<td>ldaa A, r</td>
<td>Reg A offset</td>
<td>EA=r+A, zero padded</td>
</tr>
<tr>
<td>ldaa B, r</td>
<td>Reg B offset</td>
<td>EA=r+B, zero padded</td>
</tr>
<tr>
<td>ldaa D, r</td>
<td>Reg D offset</td>
<td>EA=r+D</td>
</tr>
<tr>
<td>ldaa q, r</td>
<td>9-bit index</td>
<td>EA=r+q (-256 to 255)</td>
</tr>
<tr>
<td>ldaa W, r</td>
<td>16-bit index</td>
<td>EA=r+W (-32768 to 65535)</td>
</tr>
<tr>
<td>ldaa [D,r]</td>
<td>D indirect</td>
<td>EA=(r+D)</td>
</tr>
<tr>
<td>ldaa [W,r]</td>
<td>indirect</td>
<td>EA=(r+W) (-32768 to 65535)</td>
</tr>
</tbody>
</table>

**Freescale 6812 addressing modes**

<table>
<thead>
<tr>
<th>Pseudo op</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>org</td>
<td>Specific absolute address to put subsequent object code</td>
</tr>
<tr>
<td>=</td>
<td>equ</td>
</tr>
<tr>
<td>set</td>
<td>Define or redefine a constant symbol</td>
</tr>
<tr>
<td>dc.b</td>
<td>db</td>
</tr>
<tr>
<td>fcb</td>
<td>.byte</td>
</tr>
<tr>
<td>dcc</td>
<td>dw</td>
</tr>
<tr>
<td>fcc</td>
<td>fdb</td>
</tr>
<tr>
<td>dc.l</td>
<td>dl</td>
</tr>
<tr>
<td>ds</td>
<td>rmb</td>
</tr>
<tr>
<td>.blkb</td>
<td></td>
</tr>
<tr>
<td>ds .w</td>
<td>.blkw</td>
</tr>
<tr>
<td>Address</td>
<td>Bit 7</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>$0082</td>
<td>ADPU</td>
</tr>
<tr>
<td>$0083</td>
<td>0</td>
</tr>
<tr>
<td>$0084</td>
<td>SRES8</td>
</tr>
<tr>
<td>$0085</td>
<td>DJM</td>
</tr>
<tr>
<td>$0086</td>
<td>SCF</td>
</tr>
<tr>
<td>$008B</td>
<td>CCF7</td>
</tr>
<tr>
<td>$008D</td>
<td>Bit 7</td>
</tr>
<tr>
<td>$0270</td>
<td>PTAD7</td>
</tr>
<tr>
<td>$0272</td>
<td>DDRAD7</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>$0090</td>
<td></td>
</tr>
<tr>
<td>$0092</td>
<td></td>
</tr>
<tr>
<td>$0094</td>
<td></td>
</tr>
<tr>
<td>$0096</td>
<td></td>
</tr>
<tr>
<td>$0098</td>
<td></td>
</tr>
<tr>
<td>$009A</td>
<td></td>
</tr>
<tr>
<td>$009C</td>
<td></td>
</tr>
<tr>
<td>$009E</td>
<td></td>
</tr>
</tbody>
</table>

**TSCR1** is the first 8-bit timer control register

**TSCR2** is the second 8-bit timer control register

**TCNT** is 4MHz/2^n, with PLL TCNT is 24MHz/2^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)

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

Jonathan W. Valvano   November 28, 2007   10:00am-10:50am