Quiz 1

Date: February 23, 2012

UT EID: ________________

Printed Name: ____________________________________________________________

Last,                                First

Your signature is your promise that you have not cheated and will not cheat on this exam, nor will you help others to cheat on this exam:

Signature: __________________________________________________________________

Instructions:

• Closed book and closed notes.
• No calculators or any electronic devices (turn cell phones off).
• You must put your answers on pages 2-6 only.
• You have 75 minutes, so allocate your time accordingly.
• Show your work, and put your answers in the boxes.
• Please read the entire quiz before starting.
(5) Question 1. What is the value of the unsigned four-digit hexadecimal number $1120$? Give your answer as a decimal number.

(6) Question 2. For each of the following statements fill in the word or phrase that matches best

Part a) A drawing with circles (programs) and rectangles (hardware) where the arrows illustrate which module accesses/controls/calls which other modules.

Part b) A property of a number system that specifies the total number of possible values.

Part c) A computer system where the I/O devices are accessed differently from the way memory is accessed (i.e., using the special I/O instructions).

(6) Question 3. Consider the following two instructions

\begin{verbatim}
ldab #100
subb #-90
\end{verbatim}

What will be the value of the overflow (V) bit?

What will be the value of the carry (C) bit?

(5) Question 4. A 20-bit number is approximately how many decimal digits?
(10) **Question 5.** Interface the switch to PT0 using negative logic (pressed is low, not pressed is high). No software is required in this question, and you may assume PT0 is an input. Your bag of parts includes the switch, the 7406, and one resistor each of the values \{1\, \Omega, 10\, \Omega, 100\, \Omega, 1\, k\, \Omega, 10\, k\, \Omega, 100\, k\, \Omega and 1\, M\, \Omega\}. Pick the best resistors to use (you will not need them all.) Use the 7406 only if it is absolutely needed. Assume \(V_{\text{OL}}\) of the 7406 is 0.5 V.

![Diagram of 7406 and switch connection]

(5) **Question 6.** You are given an LED with a desired operating point of 2 V at 1 mA. Sketch the approximate voltage versus current relationship for this diode.

![LED response graph]

LED response

<table>
<thead>
<tr>
<th>Voltage (V)</th>
<th>Current (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1.5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2.5</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

Jonathan W. Valvano      February 23, 2012       5:00pm-6:15pm
(5) Question 7. Assume PC is $6000, and Register B is $45. You may assume location $0001 contains $03. Show the simplified bus cycles occurring when the subb instruction is executed. In the “changes” column, specify which registers get modified during that cycle, and the corresponding new values. Do not worry about changes to the CCR. Just show the one instruction.

$6000 D001 subb $0001

<table>
<thead>
<tr>
<th>R/W</th>
<th>Addr</th>
<th>Data</th>
<th>Changes to A,B,X,Y,S,PC,IR,EAR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(4) Question 8. Consider the following piece of code that starts at main

$4110 org $4110
$4110 CE03E8 Test ldx #1000 $3FFB
$4113 0708 bsr Delay $3FFC
$4115 3D rts $3FDF
$4116 CF4000 main lds #$4000 $3FFE
$4119 07F5 loop bsr Test $3FFF
$411B 20FC bra loop
$411D 09 Delay dex
$411E 26FD bne Delay
$4120 3D rts
$FFFE org $FFFE
$FFFE 4116 fdb main

Think about how this program executes up to and including the first execution of dex

Fill in specific hexadecimal bytes that are pushed on the stack.
Using an arrow, label to which box the SP points.
Your solution may or may not use all the boxes.

(4) Question 9. Show the C code to create a variable named Position with range 0 to 65535?

(10) Question 10. Write assembly code to swap D, X, and Y (D goes to X, X goes to Y, and Y goes to D). You must use the stack and cannot use any global variables. You do not need to set the reset vector or initialize the stack in this question.
(20) **Question 11.** Assume two positive logic switches are connected to PT5 and PT2, and one positive logic LED is connected to PT0. Write an assembly language program (main, initialization, loop, and reset vector) that turns on the LED if exactly one of the two switches is on. Turn off the LED if neither or both switches are pressed. After initializing the port, the input from switches and output to LED will be performed over and over continuously. Your code must have comments and be written in a **friendly** manner. You may use the following definitions

\[
\begin{align*}
PTT & \quad \text{equ} \quad \$0240 \\
DDRT & \quad \text{equ} \quad \$0242
\end{align*}
\]
(20) **Question 12.** Write a C program that controls a kidney dialysis pump. Port T is an 8-bit output that adjusts power to the pump. The range is 0 (no power) to 255 (full power). Port P is an 8-bit input that contains the measured blood flow in ml/min. The range is 0 (no flow) to 255 ml/min. The goal is to pump blood at 100 ml/min. If the measured flow is less than 100 ml/min, increase the power by 1 unit. If the measured flow is more than 100 ml/min, decrease the power by 1. Implement ceiling and floor (do not let the power go above 255 or below 0). First initialize Port T and Port P, then run the pump controller over and over continuously. You may use the symbols DDRP, DDRT, PTP and PTT. To adjust power to the pump, write 8 bits to PTT. To measure the flow, read 8 bits from PTP.
aba 8-bit add RegA=RegA+RegB
abx unsigned add RegX=RegX+RegB (unsigned)
aby unsigned add RegY=RegY+RegB (unsigned)
adc 8-bit add with carry to RegA
add 8-bit add with carry to RegB
addr 8-bit add to RegA
addr+ 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 left shift RegB
asld/lslld 16-bit left shift RegD
asrb 8-bit arith right shift to RegB (signed)
asra 8-bit arith right shift to RegA (signed)
asr 8-bit arith right shift Memory (signed)
asld/lslld 16-bit left shift Memory (signed)
ascb 8-bit arith right shift to RegB (signed)
bcc branch if carry clear
bcld clear bits in memory
bcc 8-bit arith right shift to RegB (signed)
bcs branch if carry set
beq branch if result is zero (Z=1)
beq 8-bit arith right shift to RegA
bgt branch if signed >
blt branch if signed <
ble branch if signed ≤
blge branch if signed ≥
blle branch if signed ≤
blgt 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 set bits in memory
bset PTT,#$84 ;set bits 7 and 0
bshr branch to subroutine
bvc branch if overflow clear
bvs branch if overflow set
call subroutine in expanded memory
cbas 8-bit compare RegA with RegB (A-B)
clc clear carry bit, C=0
cli clear T=0, enable interrupts
clr 8-bit memory clear
clea RegA clear
clrB RegB clear
clv clear overflow bit, V=0
cmpeq 8-bit compare RegA with memory
cmpeqb 8-bit compare RegB with memory
cmpeqw 8-bit compare RegW with memory
cmpg 8-bit logical complement to RegA
cmpgb 8-bit logical complement to RegB
cmpgw 8-bit logical complement to RegW
cmpeq 8-bit compare RegX with memory
cmpgt 16-bit compare RegX with memory
cmpg 8-bit logical complement to RegX
cmpgt 8-bit logical complement to RegA
cmpl 8-bit logical complement to RegB
cmpeq 8-bit compare RegD with memory
cmpl 8-bit logical complement to RegCC
cmpl 8-bit logical complement to RegG
cmpeq 8-bit compare RegY with memory
daa 8-bit decimal adjust accumulator
dec 8-bit decrement memory
dec 8-bit decrement RegA
dec 8-bit decrement RegB
dec 8-bit decrement RegX
dec 8-bit decrement RegSP
dec 8-bit decrement RegW
dec 8-bit decrement RegCC
dec 8-bit decrement RegG
dec 8-bit decrement RegY
dec 8-bit decrement RegD
dec 8-bit decrement RegX
dec 8-bit decrement RegY
dec 8-bit decrement RegSP
dec 8-bit decrement RegW
dec 8-bit decrement RegCC
dec 8-bit decrement RegG
dec 8-bit decrement RegY
dec 8-bit decrement RegD
dec 8-bit decrement RegX
dec 8-bit decrement RegY
dec 8-bit decrement RegSP
dec 8-bit decrement RegW
dec 8-bit decrement RegCC
dec 8-bit decrement RegG
dec 8-bit decrement RegY
dec 8-bit decrement RegD
dec 8-bit decrement RegX
dec 8-bit decrement RegY
dec 8-bit decrement RegSP
dec 8-bit decrement RegW
dec 8-bit decrement RegCC
dec 8-bit decrement RegG
dec 8-bit decrement RegY
dec 8-bit decrement RegD
dec 8-bit decrement RegX
dec 8-bit decrement RegY
dec 8-bit decrement RegSP
dec 8-bit decrement RegW
dec 8-bit decrement RegCC
dec 8-bit decrement RegG
}
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
psbh push 8-bit RegB onto stack
pshc push 8-bit RegCC onto stack
psld push 16-bit RegD onto stack
pslh 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
ror a 8-bit roll shift right RegA
rorb 8-bit roll shift right RegB
rtc return subroutine in expanded memory
rti return from interrupt
rts return from subroutine
sh a 8-bit subtract RegA - RegB
shb 8-bit subtract RegB - RegA
shca 8-bit subtract with carry from RegA
shcb 8-bit subtract with carry from RegB
sec set carry bit, C=1
sei set I=1, disable interrupts
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
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
sex sign extend 8-bit to 16-bit reg
sex B, D
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
tbne test and branch if result≠0
tfr 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
tsb 8-bit compare RegB with zero
tax transfer S to X
tay transfer S to Y
tsx transfer S to X
tsy transfer Y to X
wai wait for interrupt
wav weighted Fuzzy logic average
xgdx exchange RegD with RegX
xgdy exchange RegD with RegY

### Example Addressing Mode

<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>Define a constant symbol</td>
</tr>
<tr>
<td>set</td>
<td>Define or redefine a constant symbol</td>
</tr>
<tr>
<td>dc b</td>
<td>Allocate byte(s) of storage with initialized values</td>
</tr>
<tr>
<td>db</td>
<td>Create an ASCII string (no termination character)</td>
</tr>
<tr>
<td>fcb</td>
<td>Allocate word(s) of storage with initialized values</td>
</tr>
<tr>
<td>.byte</td>
<td>Allocate 32-bit word(s) of storage with initialized values</td>
</tr>
<tr>
<td>.word</td>
<td>Allocate 32-bit long word(s) of storage with initialized values</td>
</tr>
<tr>
<td>.blkb</td>
<td>Allocate bytes of storage without initialization</td>
</tr>
<tr>
<td>.blkw</td>
<td>Allocate bytes of storage without initialization</td>
</tr>
<tr>
<td>.blkl</td>
<td>Allocate bytes of storage without initialization</td>
</tr>
</tbody>
</table>

### Freescale 9S12 Addressing Modes

<table>
<thead>
<tr>
<th>Example</th>
<th>Addressing Mode</th>
<th>Effective Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld a # u</td>
<td>immediate</td>
<td>No EA</td>
</tr>
<tr>
<td>ld a u</td>
<td>direct</td>
<td>EA is 8-bit address (0 to 255)</td>
</tr>
<tr>
<td>ld a U</td>
<td>extended</td>
<td>EA is a 16-bit address</td>
</tr>
<tr>
<td>ld a m, r</td>
<td>5-bit index</td>
<td>EA=r+m (-16 to 15)</td>
</tr>
<tr>
<td>ld a v, + r</td>
<td>pre-increment</td>
<td>r=r+v, EA=r (1 to 8)</td>
</tr>
<tr>
<td>ld a v, - r</td>
<td>pre-decrement</td>
<td>r=r-v, EA=r (1 to 8)</td>
</tr>
<tr>
<td>ld a v, r</td>
<td>post-increment</td>
<td>EA=r, r=r+v (1 to 8)</td>
</tr>
<tr>
<td>ld a v, r</td>
<td>post-decrement</td>
<td>EA=r, r=r-v (1 to 8)</td>
</tr>
<tr>
<td>ld a A, r</td>
<td>Reg A offset</td>
<td>EA=r+A, zero padded</td>
</tr>
<tr>
<td>ld a B, r</td>
<td>Reg B offset</td>
<td>EA=r+B, zero padded</td>
</tr>
<tr>
<td>ld a D, r</td>
<td>Reg D offset</td>
<td>EA=r+D</td>
</tr>
<tr>
<td>ld a q, r</td>
<td>9-bit index</td>
<td>EA=r+q (-256 to 255)</td>
</tr>
<tr>
<td>ld a W, r</td>
<td>16-bit index</td>
<td>EA=r+W (-32768 to 65535)</td>
</tr>
<tr>
<td>ld a [D, r]</td>
<td>D indirect</td>
<td>EA=(r+D)</td>
</tr>
<tr>
<td>ld a [W, r]</td>
<td>indirect</td>
<td>EA=(r+W) (-32768 to 65535)</td>
</tr>
</tbody>
</table>
RAM is $2000$ to $3FFF$, ROM is $4000$-$FFFF$, and the reset vector is at $SFFFE$

### STX

**Store Index Register X**

**Operation:**  
$$(X_H : X_L) \Rightarrow M : M + 1$$

**Description:** Stores the content of index register $X$ in memory. The most significant byte of $X$ is stored at the specified address, and the least significant byte of $X$ is stored at the next higher byte address (the specified address plus one).

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
<th>HCS12 Access Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>STX opr8a</td>
<td>DIR</td>
<td>5E dd</td>
<td>PW</td>
</tr>
<tr>
<td>STX opr16a</td>
<td>EXT</td>
<td>7E hh ll</td>
<td>PWO</td>
</tr>
<tr>
<td>STX oprx0_xysp</td>
<td>IDX</td>
<td>6E xb</td>
<td>PW</td>
</tr>
<tr>
<td>STX oprx9,xyssp</td>
<td>IDX1</td>
<td>6E xb ff</td>
<td>PWO</td>
</tr>
<tr>
<td>STX oprx16,xysp</td>
<td>IDX2</td>
<td>6E xb ee ff</td>
<td>PWP</td>
</tr>
</tbody>
</table>

### BSR

**Branch to Subroutine**

**Operation:**  
$$(SP) \Rightarrow SP$$  
$$RTNH : RTNL \Rightarrow M(SP) : M(SP+1)$$  
$$(PC) + \text{Rel} \Rightarrow PC$$

**Description:** Sets up conditions to return to normal program flow, then transfers control to a subroutine. Uses the address of the instruction after the BSR as a return address. Decrements the SP by two, to allow the two bytes of the return address to be stacked. Stacks the return address (the SP points to the high-order byte of the return address). Branches to a location determined by the branch offset. Subroutines are normally terminated with an RTS instruction, which restores the return address from the stack.

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
<th>Access Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSR rel8</td>
<td>REL</td>
<td>07 rr</td>
<td>SPPP</td>
</tr>
</tbody>
</table>

### RTS

**Return from Subroutine**

**Operation:**  
$$(M(SP) : M(SP+1)) \Rightarrow PCH : PCL; (SP) + 0002 \Rightarrow SP$$

**Description:** Restores context at the end of a subroutine. Loads the program counter with a 16-bit value pulled from the stack and increments the stack pointer by two. Program execution continues at the address restored from the stack.

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
<th>Access Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTS</td>
<td>INH</td>
<td>3D</td>
<td>UfPPP</td>
</tr>
</tbody>
</table>
**SUBBB**  Subtract B  **SUBBB**

**Operation:**  \((B) - (M) \Rightarrow B\)

**Description:** Subtracts the content of memory location \(M\) from the content of accumulator \(B\) and places the result in \(B\). For subtraction instructions, the \(C\) status bit represents a borrow.

- **N:** Set if MSB of result is set; cleared otherwise
- **Z:** Set if result is \(00\); cleared otherwise
- **V:** \(B_7 \cdot M_7 \cdot R_7 + B_7 \cdot M_7 \cdot R_7\)
  - Set if a two’s complement overflow resulted from the operation; cleared otherwise
- **C:** \(B_7 \cdot M_7 + M_7 \cdot R_7 + R_7 \cdot B_7\)
  - Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
<th>Access Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{SUBB} #opr8i</td>
<td>IMM</td>
<td>C0 ii</td>
<td>P</td>
</tr>
<tr>
<td>\texttt{SUBB} opr8a</td>
<td>DIR</td>
<td>D0 dd</td>
<td>rPf</td>
</tr>
<tr>
<td>\texttt{SUBB} opr16a</td>
<td>EXT</td>
<td>F0 hh ll</td>
<td>rPO</td>
</tr>
<tr>
<td>\texttt{SUBB} oprx0.xysp</td>
<td>IDX</td>
<td>E0 xb</td>
<td>rPf</td>
</tr>
<tr>
<td>\texttt{SUBB} oprx9.xysp</td>
<td>IDX1</td>
<td>E0 xb ff</td>
<td>rPO</td>
</tr>
<tr>
<td>\texttt{SUBB} oprx16.xysp</td>
<td>IDX2</td>
<td>E0 xb ee ff</td>
<td>frFP</td>
</tr>
<tr>
<td>\texttt{SUBB} [D.xysp]</td>
<td>[D,IDX]</td>
<td>E0 xb</td>
<td>fFrPf</td>
</tr>
<tr>
<td>\texttt{SUBB} [opr16.xysp]</td>
<td>[IDX2]</td>
<td>E0 xb ee ff</td>
<td>fFPrPf</td>
</tr>
</tbody>
</table>

**ADDA**  Add without Carry to A  **ADDA**

**Operation:** \((A) + (M) \Rightarrow A\)

**Description:** Adds the content of memory location \(M\) to accumulator \(A\) and places the result in \(A\).

- **N:** Set if MSB of result is set; cleared otherwise
- **Z:** Set if result is \(00\); cleared otherwise
- **V:** \(A_7 \cdot M_7 \cdot R_7 + A_7 \cdot M_7 \cdot R_7\)
  - Set if two’s complement overflow resulted from the operation; cleared otherwise
- **C:** \(A_7 \cdot M_7 + M_7 \cdot R_7 + R_7 \cdot A_7\)
  - Set if there was a carry from the MSB of the result; cleared otherwise