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 $1210$? 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 the type, direction and amount of data
being transferred.

Part b) The subset of elements from which the entire set can be created.

Part c) A computer system where the I/O devices are accessed in a similar way as memory is accessed (i.e., using the same instructions).

(6) Question 3. Consider the following two instructions

```plaintext
ldaa #-100
adda #90
```

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

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

(5) Question 4. A 30-bit number is approximately how many decimal digits?
(10) Question 5. Interface the switch to PT0 using positive logic (pressed is high, not pressed is low). 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 {1Ω, 10Ω, 100Ω, 1kΩ, 10kΩ, 100kΩ and 1MΩ}. Pick the best resistors to use (you will not need them all.) Use the 7406 only if it is absolutely needed. Assume V_{OL} of the 7406 is 0.5 V.

(5) Question 6. You are given an LED with a desired operating point of 2.5V at 10 mA. Sketch the approximate voltage versus current relationship for this diode.
(5) Question 7. Assume PC is $5000, and Register B is $34. You may assume location $0001 contains $12. 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.

$5000 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

$4114 org $4114
$4114 CE03E8 Test ldx #1000 $3FFB
$4117 0708 bsr Delay $3FFC
$4119 3D rts $3FFD
$411A CF4000 main lds #$4000 $3FFE
$411D 07F5 loop bsr Test $3FFF
$411F 20FC bra loop
$4121 09 Delay dex
$4122 26FD bne Delay
$4124 3D rts
$FFFE org $FFFE
$FFFE 411A 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 -128 to +127?

(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 PT2 and PT0, and one positive logic LED is connected to PT5. 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

```
PTT   equ  $0240
DDRT  equ  $0242
```
(20) Question 12. Write a C program that controls a kidney dialysis pump. Port P is an 8-bit output that adjusts power to the pump. The range is 0 (no power) to 255 (full power). Port T 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 150 ml/min. If the measured flow is less than 150 ml/min, increase the power by 1 unit. If the measured flow is more than 150 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 PTP. To measure the flow, read 8 bits from PTT.
aba 8-bit add RegA=RegA+RegB
dey 16-bit decrement RegY
dex 16-bit decrement RegX
abx unsigned add RegX=RegX+RegB (unsigned)
ediv RegY=(Y:D)/RegX, unsigned divide
aby unsigned add RegY=RegY+RegB (unsigned)
edivs RegY=(Y:D)/RegX, signed divide
adca 8-bit add with carry to RegA
emac 16 by 16 signed mult, 32-bit add
adcb 8-bit add with carry to RegB
emaxd 16-bit unsigned maximum in RegD
add 8-bit add to RegA
emaxm 16-bit unsigned maximum in memory
addb 8-bit add to RegB
emin 16-bit unsigned minimum in RegD
addb 8-bit add to RegB
emul RegY=RegY*RegD unsigned mult
addc 8-bit logical and to RegCC
eora 8-bit logical exclusive or to RegA
asr 8-bit arith right shift to RegB
bsr branch to subroutine
asra 8-bit arith right shift to RegA (signed)
bset PTT,#$01,loop
asl 8-bit arith right shift Memory (signed)
bset PTT,#$84 ;set bits 7 and 2
asld/lsld 16-bit left shift RegD
bset PTT,#$05 ;clear bits 2 and 0
aslb/lslb 8-bit left shift RegB
bclr PTT,#$05 ;clear bits 2 and 0
asla/lsla 8-bit left shift RegA
bclr clear bits in memory
aslb/lslb 8-bit left shift RegB
bls branch if signed
asld/lsld 16-bit left shift RegD
blo branch if unsigned <
bls branch if unsigned \\ge
blt branch if signed <
bset PTT,#$01,loop
ble branch if signed \le
bhi branch if unsigned >
bset PTT,#$84 ;set bits 7 and 2
bhs branch if unsigned \\le
bset set bits in memory
bsb branch if carry set
bset PTT,#$584 ;set bits 7 and 2
bset branch if carry set
bsr branch to subroutine
bvc branch if overflow clear
bvc branch if overflow clear
bvs branch if overflow set
ldaa 8-bit decimal adjust accumulator
call subroutine in expanded memory
call subroutine in expanded memory
ldeq increment branch if result=0
clrc branch if bits are clear
ldex 16-bit load effective addr to X
clear bits in memory
ldex 16-bit load effective addr to X
ldiy 16-bit load memory into RegY
ldix 16-bit load memory into RegX
ldx 16-bit load memory into RegX
ldy 16-bit load memory into RegY
ldz 16-bit load memory into RegZ
leas 16-bit load effective addr to SP
lea 16-bit load effective addr to X
lei 16-bit load memory to memory
mov 8-bit move memory to memory
movb #100,PTT
movh #13,SCIBD
mul RegD=RegA*RegB, 8 by 8 into 16 bits
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
psy 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
rew 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
sha 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
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
styx 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
tbne test and branch if result≠0

---

### Example Addressing Modes

<table>
<thead>
<tr>
<th>Addressing Mode</th>
<th>Effective Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld #u</td>
<td>Immediate, No EA</td>
</tr>
<tr>
<td>ld u</td>
<td>Direct, EA is 8-bit address (0 to 255)</td>
</tr>
<tr>
<td>ld U</td>
<td>Extended, EA is a 16-bit address</td>
</tr>
<tr>
<td>ld m,r</td>
<td>5-bit index, EA=r+m (-16 to 15)</td>
</tr>
<tr>
<td>ld v,+r</td>
<td>Pre-increment, r=r+v, EA=r (1 to 8)</td>
</tr>
<tr>
<td>ld v,-r</td>
<td>Pre-decrement, r=r-v, EA=r (1 to 8)</td>
</tr>
<tr>
<td>ld v,+r</td>
<td>Post-increment, EA=r, r=r+v (1 to 8)</td>
</tr>
<tr>
<td>ld v,-r</td>
<td>Post-decrement, EA=r, r=r-v (1 to 8)</td>
</tr>
<tr>
<td>ld A,r</td>
<td>Reg A offset, EA=r+A, zero padded</td>
</tr>
<tr>
<td>ld B,r</td>
<td>Reg B offset, EA=r+B, zero padded</td>
</tr>
<tr>
<td>ld D,r</td>
<td>Reg D offset, EA=r+D</td>
</tr>
<tr>
<td>ld q,r</td>
<td>9-bit index, EA=r+q (-256 to 255)</td>
</tr>
<tr>
<td>ld W,r</td>
<td>16-bit index, EA=r+W (-32768 to 65535)</td>
</tr>
<tr>
<td>ld [D,r]</td>
<td>Direct, EA=(r+D)</td>
</tr>
<tr>
<td>ld [W,r]</td>
<td>Indirect, EA=(r+W) (-32768 to 65535)</td>
</tr>
</tbody>
</table>

### Freescale 9S12 Addressing Modes

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Pseudo Op</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specific absolute address to put subsequent object code</td>
<td>org</td>
</tr>
<tr>
<td>Define a constant symbol</td>
<td>= equ</td>
</tr>
<tr>
<td>Define or redefine a constant symbol</td>
<td>set</td>
</tr>
<tr>
<td>Allocate byte(s) of storage with initialized values</td>
<td>dc.b, db, fcb .byte</td>
</tr>
<tr>
<td>Create an ASCII string (no termination character)</td>
<td>fcc</td>
</tr>
<tr>
<td>Allocate word(s) of storage with initialized values</td>
<td>dc.w, dw, fdb .word</td>
</tr>
<tr>
<td>Allocate 32-bit long word(s) of storage with initialized values</td>
<td>dc.l, dl .long</td>
</tr>
<tr>
<td>Allocate bytes of storage without initialization</td>
<td>ds, ds.b, rmb .blkb</td>
</tr>
<tr>
<td>Allocate bytes of storage without initialization</td>
<td>ds.w .blkw</td>
</tr>
<tr>
<td>Allocate 32-bit words of storage without initialization</td>
<td>ds.l .blkl</td>
</tr>
</tbody>
</table>
RAM is $2000 to $3FFF, ROM is $4000-$FFFF, and the reset vector is at $FFFE

### 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,xyspp</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:**

\[(SP) - \$0002 \Rightarrow SP\]

\[RTNH : RTNL \Rightarrow M(SP) : M(SP+1)\]

\[(PC) + \text{Rel} \Rightarrow PC\]

**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 HCS12</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 HCS12</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTS</td>
<td>INH</td>
<td>3D</td>
<td>UPPP</td>
</tr>
</tbody>
</table>
**SUBB**  Subtract B  **SUBB**

**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>SUBB #opr8i</td>
<td>IMM</td>
<td>C0 ii</td>
<td>P</td>
</tr>
<tr>
<td>SUBB opr8a</td>
<td>DIR</td>
<td>D0 dd</td>
<td>rF</td>
</tr>
<tr>
<td>SUBB opr16a</td>
<td>EXT</td>
<td>F0 hh ll</td>
<td>rP</td>
</tr>
<tr>
<td>SUBB oprx0_xySp</td>
<td>IDX</td>
<td>E0 xb</td>
<td>rP</td>
</tr>
<tr>
<td>SUBB oprx9_xySp</td>
<td>IDX1</td>
<td>E0 xb ff</td>
<td>rP</td>
</tr>
<tr>
<td>SUBB oprx16_xySp</td>
<td>IDX2</td>
<td>E0 xb ee ff</td>
<td>frP</td>
</tr>
<tr>
<td>SUBB [D,xySp]</td>
<td>[D,IDX]</td>
<td>E0 xb</td>
<td>flFrP</td>
</tr>
<tr>
<td>SUBB [opr16,xySp]</td>
<td>[IDX2]</td>
<td>E0 xb ee ff</td>
<td>flFrP</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