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 phase 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

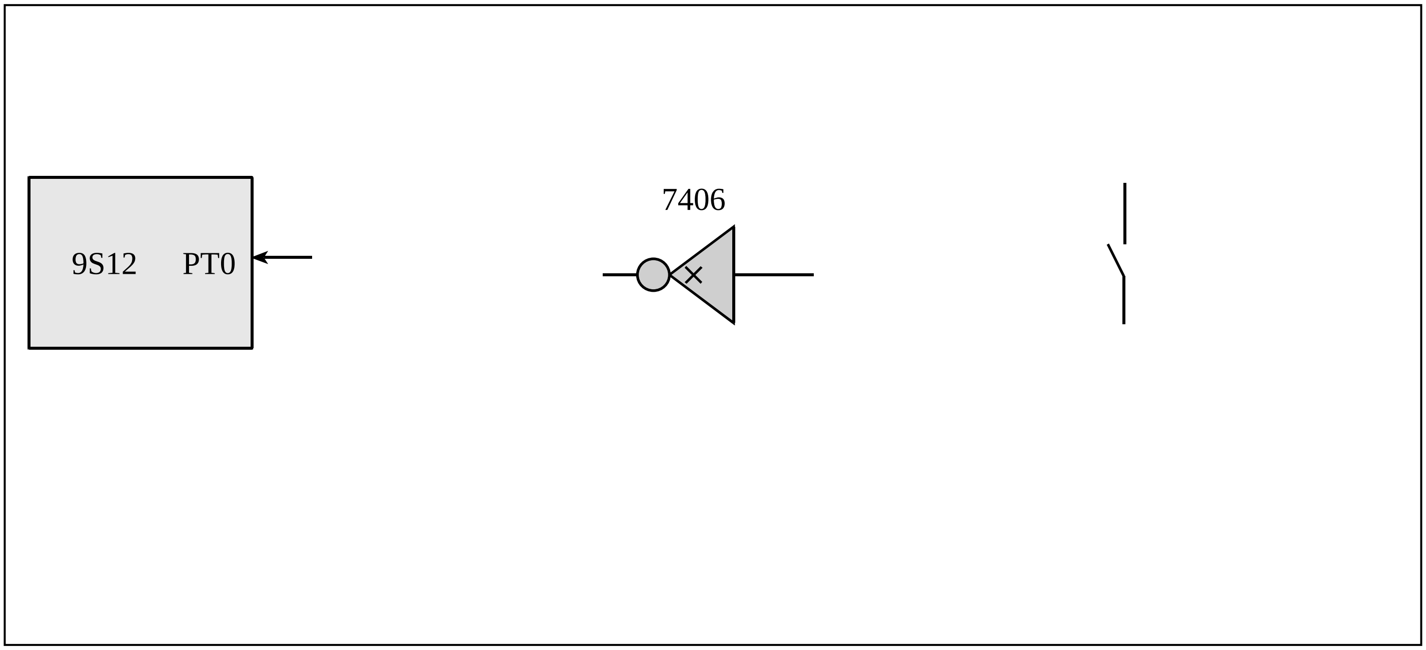
```
ldab #100
subb #-90
```

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

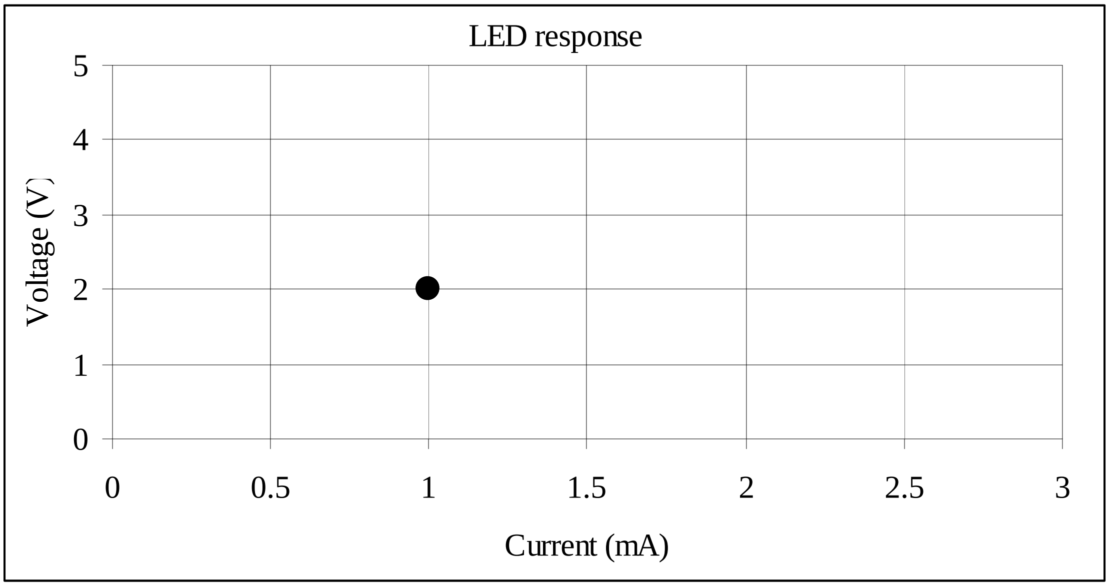
What will be the value of the carry (C) bit? --

(5) Question 4. A 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 {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 V at 1 mA. Sketch the approximate voltage versus current relationship for this diode.



(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
```

R/W	Addr	Data	Changes to A,B,X,Y,S,PC,IR,EAR

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

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

$3FFB	
$3FFC	
$3FFD	
$3FFE	
$3FFF	

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

```
PTT   equ  $0240
DDRT  equ  $0242
```

(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)
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 left shift RegB
asld/lsld 16-bit left shift RegD
asr    8-bit arith right shift Memory (signed)
asra   8-bit arith right shift to RegA (signed)
asrb   8-bit arith right shift to RegB (signed)
bcc    branch if carry clear
bclr   clear bits in memory
         bclr PTT,#$05 ;clear bits 2 and 0
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
         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 2
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 (A-B)
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
         dbeq Y,loop
dbne   decrement and branch if result≠0
         dbne A,loop
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
         exg X,Y
fdiv   unsigned fract div, X=(65536*D)/X
ibeq   increment and branch if result=0
         ibeq Y,loop
ibne   increment and branch if result≠0
         ibne A,loop
idiv   16-bit unsigned div, X=D/X, D=rem
idivs  16-bit signed divide, X=D/X, D=rem
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 unsigned right shift memory
lsra   8-bit unsigned right shift RegA
lsrb   8-bit unsigned right shift RegB
lsrd   16-bit unsigned 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
         movb #100,PTT
movw   16-bit move memory to memory
         movw #13,SCIBD
mul    RegD=RegA*RegB, 8 by 8 into 16 bits
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 subroutine 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
         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
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
         tbeq Y,loop
tbl    8-bit look up and interpolation
tbne   test and branch if result≠0
         tbne A,loop
tfr    transfer register to register
         tfr A,Y ;same as sex A,Y
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
tsx    transfer S to X
tsy    transfer S to Y
txs    transfer X to S
tys    transfer Y 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	No EA
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 9S12 addressing modes **r** is **X**, **Y**, **SP**, or **PC**

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
ds.w			**.blkw**	Allocate bytes of storage without initialization
ds.l			**.blkl**	Allocate 32-bit words of storage without initialization

RAM is \$2000 to \$3FFF, ROM is \$4000-\$FFFF, and the reset vector is at \$FFFE

STX Store Index Register X STX

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).

Source Form	Address Mode	Object Code	HCS12 Access Detail
STX opr8a	DIR	**5E dd**	**PW**
STX opr16a	EXT	**7E hh ll**	**PWO**
STX oprx0_xysp	IDX	**6E xb**	**PW**
STX oprx9,xyssp	IDX1	**6E xb ff**	**PWO**
STX oprx16,xysp	IDX2	**6E xb ee ff**	**PWP**

BSR Branch to Subroutine BSR

Operation: $(SP) - \$0002 \Rightarrow SP$

$RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$

$(PC) + Rel \Rightarrow PC$

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.

Source Form	Address Mode	Object Code	Access Detail HCS12
BSR rel8	REL	**07 rr**	**SPPP**

RTS Return from Subroutine RTS

Operation: $(M_{(SP)} : M_{(SP+1)}) \Rightarrow PC_H : PC_L$; $(SP) + \$0002 \Rightarrow SP$

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.

Source Form	Address Mode	Object Code	Access Detail HCS12
RTS	INH	**3D**	**UfPPP**

SUBB Subtract B SUBB

Operation: (B) – (M) ⇒ 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: $B7 \bullet \overline{M7} \bullet \overline{R7} + \overline{B7} \bullet M7 \bullet R7$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{B7} \bullet M7 + M7 \bullet R7 + R7 \bullet \overline{B7}$
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail HCS12
SUBB #*opr8i*	IMM	C0 ii	P
SUBB *opr8a*	DIR	D0 dd	rPf
SUBB *opr16a*	EXT	F0 hh ll	rPO
SUBB *oprx0_xysp*	IDX	E0 xb	rPf
SUBB *oprx9,xysp*	IDX1	E0 xb ff	rPO
SUBB *oprx16,xysp*	IDX2	E0 xb ee ff	frPP
SUBB [D,*xysp*]	[D,IDX]	E0 xb	fIfrPf
SUBB [*oprx16,xysp*]	[IDX2]	E0 xb ee ff	fIPrPf

ADDA Add without Carry to A ADDA

Operation: (A) + (M) ⇒ 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: $A7 \bullet M7 \bullet \overline{R7} + \overline{A7} \bullet \overline{M7} \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise

C: $A7 \bullet M7 + M7 \bullet \overline{R7} + \overline{R7} \bullet A7$
Set if there was a carry from the MSB of the result; cleared otherwise