First:________________ Last:____________________

This is a closed book exam. You must put your answers pages 1,2,3 only. You have 50 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. Consider the hexadecimal number $A5. If the format is 8-bit signed, what decimal number does this represent?

(5) Question 2. Consider the following display of a digital voltmeter.

Part a) The voltmeter reads from 0.0000 to 3.9999V. How many decimal digits is the voltmeter?

Part b) If you were asked to implement this voltmeter using a microcontroller, what is the minimum number of ADC bits you could use?

(5) Question 3. Consider the following two instructions

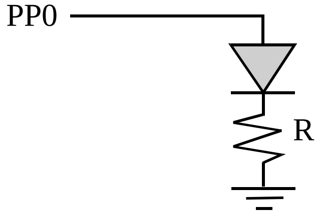
```
ldab #-1
addb #1
```

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

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

What will be the value of the zero (Z) bit?

(5) Question 4. What resistor value do you need to interface this LED to the 9S12? The desired operating point is 1.9V at 1 mA. At 1mA you can assume the V_{OH} of the 9S12 will be 4.9 V.



(10) Question 5. Consider the following piece of code that starts at **`main`**

```
$4035 CF4000          main   lds  #$4000
$4038 860A                   ldaa #10
$403A 0703            loop   bsr  Output
$403C 0430FB                 dbne A,loop
$403F 36              Output psha
```

Hand execute this program up to and including the first execution of **`psha`**

Fill in specific hexadecimal bytes that are pushed on the stack.
Using an arrow label to which box the SP points.
Not all the boxes will be used.

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

(10) Question 6. Assume PC is $5000, and Register D is initially $2233. You may assume location $0812 and $0813 are initially 0. Show the simplified bus cycles occurring when the **`subd`** 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 B30812            subd $0812
```

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

(10) Question 7. Draw the circuit diagram interfacing this switch using negative logic. I.e., if the switch is not pressed the PT1 input should be high, and if the switch is pressed, the input should be low. Please specify the resistance values for the resistors and the chip numbers (e.g., 7406) for any digital logic you use.

For questions 8 and 9, don't worry about establishing the reset vector, creating a main program, or initializing the stack pointer. You may use the following definitions

```
PTT   equ  $0240
DDRT  equ  $0242
PTP   equ  $0258
DDRP  equ  $025A
```

(20) Question 8. Assume **Sg** is a signed 16-bit global variable. Transfer the following C-code into assembly language. I.e., clear Port T bits 1 and 0 if **Sg** is greater than or equal to 100. You may assume PT1 and PT0 are already outputs.

```
          if (Sg >= 100) PTT &= ~0x03;
```

(30) Question 9. Assume bits 3-0 of both Port T and Port P are already inputs. You will write a subroutine that reads Port T and Port P, setting RegA bits 7-4 to Port T bits 3-0 and RegA bits 3-0 to Port P bits 3-0. Hint you can use a temporary global variable, or you could use the **`aba`** instruction.

```
;*****ReadPorts subroutine*********************
;Inputs:  none
;Outputs: RegA bits7-0 are PT3-0, RegA bits 3-0 are PP3-0
ReadPorts
```

```
aba    8-bit add RegA=RegA+RegB
abx    unsigned add RegX=RegX+RegB
aby    unsigned add RegY=RegY+RegB
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
asra   8-bit arith right shift to RegA
asrb   8-bit arith right shift to RegB
bcc    branch if carry clear
bclr   bit clear in memory
         bclr PTT,#$01
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   bit set clear in memory
         bset PTT,#$04
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
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 logical right shift memory
lsra   8-bit logical right shift RegA
lsrb   8-bit logical right shift RegB
lsrd   16-bit logical 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
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 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
          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 X,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

SUBD Subtract Double Accumulator SUBD

Operation: (A : B) – (M : M + 1) ⇒ A : B

Description: Subtracts the content of memory location M : M + 1 from the content of double accumulator D and places the result in D.

Source Form	Address Mode	Object Code	HCS12 Access Detail
SUBD #opr16i	IMM	83 jj kk	PO
SUBD opr8a	DIR	93 dd	RPf
SUBD opr16a	EXT	B3 hh ll	RPO
SUBD oprx0_xysp	IDX	A3 xb	RPf
SUBD oprx9,xyssp	IDX1	A3 xb ff	RPO
SUBD oprx16,xysp	IDX2	A3 xb ee ff	fRPP

BSR Branch to Subroutine BSR

Operation: (SP) – $0002 ⇒ SP
$RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$
(PC) + Rel ⇒ 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 ⇒ 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