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.***

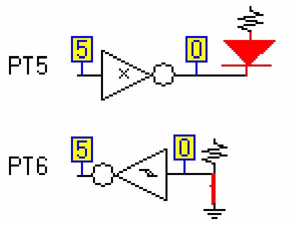
| | |
|---|---|
| **(5) Question 1.** Give the hex value……. | |
| **(5) Question 2.** Specify `0` or `1` …………… | |
| **(5) Question 3.** Specify `0` or `1` …………… | |
| **(5) Question 4.** Specify A-H ………..…. | |
| **(5) Question 5.** Show the equation…. | |
| **(5) Question 6.** Show the machine code…. | |
| **(5) Question 7.** How many binary bits?....... | |
| **(5) Question 8.** Specify values………..…. | RegA = <br> RegX = |

**(10) Question 9.** Simplified memory cycles (you may or may not need all 5 entries)

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

**(25) Part 10a)** Write the main program (figure shows the result after running your program)

```
DDRM equ $0252  ; Port M Direction
DDRT equ $0242  ; Port T Direction
PTM  equ $0250  ; Port M I/O Register
PTT  equ $0240  ; Port T I/O Register
     org $4000
main
```



**(25) Part 10b)** Write the assembly language subroutine.

```
;*******Check***********
; if PT6 is 1, then set PT5=1
; if PT6 is 0, then return without modifying PT5
Check
```

```
     org $FFFE
     fdb main
```

| | |
|---|---|
| aba | 8-bit add RegA+RegB |
| abx | unsigned add RegX+RegB |
| aby | unsigned add 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 arith left shift RegB |
| asld/lsld | 16-bit left shift RegD |
| asr | 8-bit arith right shift Memory |
| asra | 8-bit arith right shift |
| asrb | 8-bit arith right shift to RegB |
| bcc | branch if carry clear |
| bclr | clear bits in memory |
| 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, |
| brn | branch never |
| brset | branch if bits are set |
| bset | set bits in memory |
| 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 |
| dbne | decrement and branch if result≠0 |
| 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 |
| fdiv | unsigned fract div, X=(65536*D)/X |
| ibeq | increment and branch if result=0 |
| ibne | increment and branch if result≠0 |
| idiv | 16-bit unsigned divide, 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 |
| movw | 16-bit move memory to memory |
| 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 |
| 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 |
| tbl | 8-bit look up and interpolation |
| 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 |
| tstb | 8-bit compare RegB with zero |
| tsx | transfer S+1 to X |
| tsy | transfer S+1 to Y |
| txs | transfer X-1 to S |
| tys | transfer Y-1 to S |
| wai | wait for interrupt |
| wav | weighted Fuzzy logic average |
| xgdx | exchange RegD with RegX |
| xgdy | exchange RegD with RegY |

## INC Increment Memory INC

**Operation:** (M) + \$01 → M

| Source Form | Address Mode | Object Code | HCS12 Access Detail |
|---|---|---|---|
| INC opr16a | EXT | **72 hh ll** | **rPwO** |
| INC oprx0_xysp | IDX | **62 xb** | **rPw** |
| INC oprx9,xysp | IDX1 | **62 xb ff** | **rPwO** |
| INC oprx16,xysp | IDX2 | **62 xb ee ff** | **frPwP** |
| INC [D,xysp] | [D,IDX] | **62 xb** | **fIfrPw** |
| INC [oprx16,xysp] | [IDX2] | **62 xb ee ff** | **fIPrPw** |

| 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 6812 addressing modes* **r** is **X**, **Y**, **SP**, or **PC**

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 0,X 5b const | 10 −16,X 5b const | 20 1,+X pre-inc | 30 1,X+ post-inc | 40 0,Y 5b const | 50 −16,Y 5b const | 60 1,+Y pre-inc | 70 1,Y+ post-inc | 80 0,SP 5b const | 90 −16,SP 5b const | A0 1,+SP pre-inc | B0 1,SP+ post-inc | C0 0,PC 5b const | D0 −16,PC 5b const | E0 n,X 9b const | F0 n,SP 9b const |
| 01 1,X 5b const | 11 −15,X 5b const | 21 2,+X pre-inc | 31 2,X+ post-inc | 41 1,Y 5b const | 51 −15,Y 5b const | 61 2,+Y pre-inc | 71 2,Y+ post-inc | 81 1,SP 5b const | 91 −15,SP 5b const | A1 2,+SP pre-inc | B1 2,SP+ post-inc | C1 1,PC 5b const | D1 −15,PC 5b const | E1 −n,X 9b const | F1 −n,SP 9b const |
| 02 2,X 5b const | 12 −14,X 5b const | 22 3,+X pre-inc | 32 3,X+ post-inc | 42 2,Y 5b const | 52 −14,Y 5b const | 62 3,+Y pre-inc | 72 3,Y+ post-inc | 82 2,SP 5b const | 92 −14,SP 5b const | A2 3,+SP pre-inc | B2 3,SP+ post-inc | C2 2,PC 5b const | D2 −14,PC 5b const | E2 n,X 16b const | F2 n,SP 16b const |
| 03 3,X 5b const | 13 −13,X 5b const | 23 4,+X pre-inc | 33 4,X+ post-inc | 43 3,Y 5b const | 53 −13,Y 5b const | 63 4,+Y pre-inc | 73 4,Y+ post-inc | 83 3,SP 5b const | 93 −13,SP 5b const | A3 4,+SP pre-inc | B3 4,SP+ post-inc | C3 3,PC 5b const | D3 −13,PC 5b const | E3 [n,X] 16b indr | F3 [n,SP] 16b indr |
| 04 4,X 5b const | 14 −12,X 5b const | 24 5,+X pre-inc | 34 5,X+ post-inc | 44 4,Y 5b const | 54 −12,Y 5b const | 64 5,+Y pre-inc | 74 5,Y+ post-inc | 84 4,SP 5b const | 94 −12,SP 5b const | A4 5,+SP pre-inc | B4 5,SP+ post-inc | C4 4,PC 5b const | D4 −12,PC 5b const | E4 A,X A offset | F4 A,SP A offset |
| 05 5,X 5b const | 15 −11,X 5b const | 25 6,+X pre-inc | 35 6,X+ post-inc | 45 5,Y 5b const | 55 −11,Y 5b const | 65 6,+Y pre-inc | 75 6,Y+ post-inc | 85 5,SP 5b const | 95 −11,SP 5b const | A5 6,+SP pre-inc | B5 6,SP+ post-inc | C5 5,PC 5b const | D5 −11,PC 5b const | E5 B,X B offset | F5 B,SP B offset |
| 06 6,X 5b const | 16 −10,X 5b const | 26 7,+X pre-inc | 36 7,X+ post-inc | 46 6,Y 5b const | 56 −10,Y 5b const | 66 7,+Y pre-inc | 76 7,Y+ post-inc | 86 6,SP 5b const | 96 −10,SP 5b const | A6 7,+SP pre-inc | B6 7,SP+ post-inc | C6 6,PC 5b const | D6 −10,PC 5b const | E6 D,X D offset | F6 D,SP D offset |
| 07 7,X 5b const | 17 −9,X 5b const | 27 8,+X pre-inc | 37 8,X+ post-inc | 47 7,Y 5b const | 57 −9,Y 5b const | 67 8,+Y pre-inc | 77 8,Y+ post-inc | 87 7,SP 5b const | 97 −9,SP 5b const | A7 8,+SP pre-inc | B7 8,SP+ post-inc | C7 7,PC 5b const | D7 −9,PC 5b const | E7 [D,X] D indirect | F7 [D,SP] D indirect |
| 08 8,X 5b const | 18 −8,X 5b const | 28 8,−X pre-dec | 38 8,X− post-dec | 48 8,Y 5b const | 58 −8,Y 5b const | 68 8,−Y pre-dec | 78 8,Y− post-dec | 88 8,SP 5b const | 98 −8,SP 5b const | A8 8,−SP pre-dec | B8 8,SP− post-dec | C8 8,PC 5b const | D8 −8,PC 5b const | E8 n,Y 9b const | F8 n,PC 9b const |
| 09 9,X 5b const | 19 −7,X 5b const | 29 7,−X pre-dec | 39 7,X− post-dec | 49 9,Y 5b const | 59 −7,Y 5b const | 69 7,−Y pre-dec | 79 7,Y− post-dec | 89 9,SP 5b const | 99 −7,SP 5b const | A9 7,−SP pre-dec | B9 7,SP− post-dec | C9 9,PC 5b const | D9 −7,PC 5b const | E9 −n,Y 9b const | F9 −n,PC 9b const |
| 0A 10,X 5b const | 1A −6,X 5b const | 2A 6,−X pre-dec | 3A 6,X− post-dec | 4A 10,Y 5b const | 5A −6,Y 5b const | 6A 6,−Y pre-dec | 7A 6,Y− post-dec | 8A 10,SP 5b const | 9A −6,SP 5b const | AA 6,−SP pre-dec | BA 6,SP− post-dec | CA 10,PC 5b const | DA −6,PC 5b const | EA n,Y 16b const | FA n,PC 16b const |
| 0B 11,X 5b const | 1B −5,X 5b const | 2B 5,−X pre-dec | 3B 5,X− post-dec | 4B 11,Y 5b const | 5B −5,Y 5b const | 6B 5,−Y pre-dec | 7B 5,Y− post-dec | 8B 11,SP 5b const | 9B −5,SP 5b const | AB 5,−SP pre-dec | BB 5,SP− post-dec | CB 11,PC 5b const | DB −5,PC 5b const | EB [n,Y] 16b indr | FB [n,PC] 16b indr |
| 0C 12,X 5b const | 1C −4,X 5b const | 2C 4,−X pre-dec | 3C 4,X− post-dec | 4C 12,Y 5b const | 5C −4,Y 5b const | 6C 4,−Y pre-dec | 7C 4,Y− post-dec | 8C 12,SP 5b const | 9C −4,SP 5b const | AC 4,−SP pre-dec | BC 4,SP− post-dec | CC 12,PC 5b const | DC −4,PC 5b const | EC A,Y A offset | FC A,PC A offset |
| 0D 13,X 5b const | 1D −3,X 5b const | 2D 3,−X pre-dec | 3D 3,X− post-dec | 4D 13,Y 5b const | 5D −3,Y 5b const | 6D 3,−Y pre-dec | 7D 3,Y− post-dec | 8D 13,SP 5b const | 9D −3,SP 5b const | AD 3,−SP pre-dec | BD 3,SP− post-dec | CD 13,PC 5b const | DD −3,PC 5b const | ED B,Y B offset | FD B,PC B offset |
| 0E 14,X 5b const | 1E −2,X 5b const | 2E 2,−X pre-dec | 3E 2,X− post-dec | 4E 14,Y 5b const | 5E −2,Y 5b const | 6E 2,−Y pre-dec | 7E 2,Y− post-dec | 8E 14,SP 5b const | 9E −2,SP 5b const | AE 2,−SP pre-dec | BE 2,SP− post-dec | CE 14,PC 5b const | DE −2,PC 5b const | EE D,Y D offset | FE D,PC D offset |
| 0F 15,X 5b const | 1F −1,X 5b const | 2F 1,−X pre-dec | 3F 1,X− post-dec | 4F 15,Y 5b const | 5F −1,Y 5b const | 6F 1,−Y pre-dec | 7F 1,Y− post-dec | 8F 15,SP 5b const | 9F −1,SP 5b const | AF 1,−SP pre-dec | BF 1,SP− post-dec | CF 15,PC 5b const | DF −1,PC 5b const | EF [D,Y] D indirect | FF [D,PC] D indirect |

**(5) Question 1.** What is the 8-bit hexadecimal representation ($00 to $FF) for decimal value **`-90`**?

**(5) Question 2.** What will be the value of the carry (C) bit after executing the following?

```
ldab #50
subb #210
```

**(5) Question 3.** What will be the value of the overflow (V) bit after executing the following?

```
ldaa #50
adda #-60
```

**(5) Question 4.** A software variable can take on the following specific values -5.0, -4.9, -4.8, ..., 4.8, 4.9, 5.0. Which fixed-point format should be used for this variable? If more than one format could be used to solve the problem, choose the most space-efficient format. Enter the correct letter A-H.

| | |
|---|---|
| **A**) 8-bit signed fixed-point, Δ = 0.1 | **E**) 16-bit signed fixed-point, Δ = 0.1 |
| **B**) 16-bit signed fixed-point, Δ = 0.01 | **F**) 8-bit unsigned fixed-point, Δ = 0.1 |
| **C**) 8-bit signed fixed-point, Δ = 0.01 | **G**) 32-bit floating point |
| **D**) 16-bit unsigned fixed-point, Δ = 0.001 | **H**) fixed-point is not needed |

**(5) Question 5.** Rewrite the following equation using fixed-point math, assuming X,Y,Z are integers. Basically, your equation should perform this operation using integer add, subtract, multiply and divide, minimizing the error caused by dropout. Don't worry about overflow.

$Z = 0.123*X - 0.66*Y + 0.5$

**(5) Question 6.** Show the machine code generated by the instruction

```
inc 1,y+
```

**(5) Question 7.** You are asked to measure a parameter to 3½ decimal digits. What is the approximate precision in binary bits?

**(5) Question 8.** Assume the SP has been properly initialized. What will be the contents of RegA and RegX after the following six instructions are executed? Hint: draw stack pictures.

```
ldaa #$12
ldx  #$3456
pshx
psha
pulx
pula
```

**(10) Question 9.** Assume RegX is $3800, RegY is $3810, the PC is $4123, and Ram locations $3800 to $38FF are initially $00, $01,…$FF respectively. E.g., location $3856 contains $56. Show the simplified bus cycles occurring when the **`inc 10,x`** 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.*

```
$4123 620A           inc  10,x
```

**Question 10.** Write a **`main`** program and a subroutine **`Check`** that inputs from Port T bit 6 and outputs to Port T bit 5. The software system will continuously check the status of the input signal, setting the output high if the input ever becomes high. The system never sets the output low. Full credit will be given to the code that is friendly.

**(25) Part a)** Write the assembly language **`main`** program that first initializes the stack. Next it should make Port T bit 6 an input and Port T bit 5 an output. The body of the **`main`** program should call the part b) subroutine, **`Check`**, over and over without stopping.

**(25) Part b)** Write the assembly language subroutine called **`Check`**, which first reads Port T bit 6. If Port T bit 6 is 1, then set Port T bit 5 equal to 1. If Port T bit 6 is 0, then Port T bit 5 is not changed.