This is a closed book exam. You must put your answers in these boxes only. You have 3 hours, so allocate your time accordingly. *Please read the entire exam before starting.*
(5) **Question 1.** Precision in decimal digits

(2) **Part 2a.** Choose A-E

(2) **Part 2b.** Choose A-E

(2) **Part 2c.** Choose A-E

(2) **Part 2d.** Choose A-E

(2) **Part 2e.** Choose A-E

(2) **Part 3a.** Specify **RegB**

(2) **Part 3b.** Specify **0 or 1**

(1) **Part 3c.** Specify **0 or 1**

(2) **Part 4a.** Give the value for **xxx**

(3) **Part 4b.** Give the value for **yyy**

(5) **Question 5.** Give the value for **zzz**

(5) **Question 6.** Give the value

(5) **Question 7.** Give the hexadecimal value

(5) **Question 8.** Give the op code
(5) **Question 9.** Simplified memory cycles (you may or may not need all 5 entries)

<table>
<thead>
<tr>
<th>R/W</th>
<th>Addr</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(5) **Question 10.** Show the assembly code

```assembly
org $3900
aa rmb 2
bb rmb 1
TCNT equ $0044
org $4300
ldx TCNT
loop ldx #cc
ldaa 2,x
staa bb
bra loop
cc fcb 0,1,2,3,4
```

(10) **Question 11.** Hand assemble this program

<table>
<thead>
<tr>
<th>Address</th>
<th>Machine code</th>
<th>Source code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>org $3900</td>
</tr>
<tr>
<td></td>
<td></td>
<td>aa rmb 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bb rmb 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TCNT equ $0044</td>
</tr>
<tr>
<td></td>
<td></td>
<td>org $4300</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ldx TCNT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>loop ldx #cc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ld aa 2,x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>staa bb</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bra loop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cc fcb 0,1,2,3,4</td>
</tr>
</tbody>
</table>
(15) **Question 12.** Show the assembly code

(20) **Question 13.** Show the assembly code
(5) Question 1. The measurement system range is 0 to 19.9 and a resolution of 0.1. What is the precision in decimal digits?

(10) Question 2. Place letter code for the best answer in the answer boxes

(2) Part a) Which direction does data flow on the data bus during a read cycle?
   A) From 6812 to memory, or from 6812 to output device
   B) From memory to 6812, or from input device to 6812
   C) From input device to memory
   D) From memory to output device
   E) None of these answers is correct

(2) Part b) Which term best describes a computer system with a response time to external events that is short and bounded?
   A) Real time
   B) Dynamic
   C) Busy-waiting
   D) Nonintrusive
   E) None of these answers is correct

(2) Part c) Which data structure has the following features? It can hold a variable number of fixed-size elements. It has two main operations, one to store data into itself, and a second operation to remove data. The data is removed in a “first come first served” order.
   A) String
   B) Binary tree
   C) FIFO queue
   D) MACQ
   E) None of these answers is correct

(2) Part d) What is the difference between busy-waiting and gadfly synchronization?
   A) Busy-waiting is used for I/O devices and gadfly is used for software events.
   B) Busy-waiting allows you to perform other operations (so you can be busy) while waiting for I/O devices and gadfly simply waits for the I/O device to be ready.
   C) The two terms describe exactly the same synchronization method.
   D) Gadfly involves software loops, while busy-waiting involves hardware interrupts.
   E) None of these answers is correct.

(2) Part e) What is drop out?
   A) Drop out is the error that occurs when the result of a calculation exceeds the range of the number system.
   B) Drop out is the error that occurs after a right shift or a divide, and the consequence is that an intermediate result loses its ability to represent all of the values.
   C) Drop out is when both the Carry and the Overflow bits are set.
   D) Drop out is data is lost when the software does not respond fast to an I/O event.
   E) None of these answers is correct.

(5) Question 3. Consider the result of executing the following two 6812 assembly instructions.
   l dab #170
   add b #125

(2) Part a) What will be the unsigned decimal value in Register B (0 to 255)?
(2) Part b) What will be the value of the carry (C) bit?
(1) Part c) What will be the value of the overflow (V) bit?
(5) Question 4. Consider the following assembly subroutine that creates two local variables, called \( p \) and \( q \). The variable \( p \) is 8 bits and initialized to 50. The variable \( q \) is 16 bits and initialized to 500. The local variable binding is created using the \texttt{set} pseudo-ops.

\begin{verbatim}
p    set xxx ; binding
q    set yyy ; binding

subl pshx ; save register X
    movb #50,1,-sp ; allocate and initialize p
    movb #500,2,-sp ; allocate and initialize q

;... stuff
    ldaa P,sp ; read from p
    ldx Q,sp ; read from q

;... more stuff
    leas 3,s ; deallocate p,q
    pulx ; restore register X
rts ; return
\end{verbatim}

(2) Part a) What value should you use in the \texttt{xxx} position to implement the binding of \( p \)?

(3) Part b) What value should you use in the \texttt{yyy} position to implement the binding of \( q \)?

(5) Question 5. Consider the following main program that calls an assembly subroutine using call by reference parameter passing on the stack. An address to an output port is pushed on the stack, and this subroutine will set bit 0 of that port. The subroutine uses Register X stack frame.

\begin{verbatim}
port set zzz ; binding
main lds #$4000
    ldd #PTT ; address to PTT
    pshd ; pass 16-bit \texttt{port} parameter on stack
    jsr Set0
    leas 2,sp
stop
Set0 pshx ; save register X
tsx ; create Register X stack frame
    leas -4,sp ; allocate locals
    ldy \texttt{port},x ; get \texttt{port} parameter
    bset 0,y,#$01 ; set bit0 of the port
    leas 4,sp ; deallocate locals
    pulx ; restore register X
rts ; return
\end{verbatim}

What value should you use in the \texttt{zzz} position to implement the binding of this parameter?

(5) Question 6. A signed 16-bit decimal \texttt{fixed-point} number system has a \( \Delta \) resolution of 1/100. What is the corresponding value of the number if the integer part stored in memory is \$C000? 

(5) Question 7. Assume RegX is initially \$4321, RegD is initially \$8765. What is resulting hexadecimal value in RegD after these instructions execute?

\begin{verbatim}
psha
pshb
pshx
pula
leas 2,s
pulb
\end{verbatim}
(5) Question 8. Assume PTT is a signed 8-bit input and PTM is an output. The goal of this code is to clear PTM if PPT is larger than 100. Which op code should be used in the ??? position?

```
lda PTT
cmpa #100
??? skip
clr PTM
```

(5) Question 9. Give the simplified memory cycles produced when the following one instruction is executed. Assume the PC contains $4000, Register Y contains $3900, each memory location from $3800 to $3FFF contains a value equal to the least significant byte of its address. I.e., $3800 contains $00, $3801 contains $01, etc. Just show R/W=Read or Write, Address, and Data for each cycle.

```
$4000 EE71  ldx 2,y+
```

(5) Question 10. Consider a table with 25 entries and 4 fields. Each entry has different values, but the same size and format. The assembly code for table entry number 5 is shown below

```
Entry5  fcb 100  8-bit count field
       fdb 1000 16-bit time field
       fcb "happy,"0,0,0 8-byte name field
       fdb -50,300,-200 three 16-bit yaw,pitch,roll fields
```

Assume Register X is pointing to this entry

```
ldx #Entry5
```

Using Register X, write assembly code that reads the pitch value of Entry5 into a register

(10) Question 11. Hand assemble the program shown on the answer pages

(15) Question 12. The SCISR1 register contains the TDRE and RDRF flags

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

TDRE — Transmit Data Register Empty Flag

- TDRE is set when the transmit shift register receives a byte from the SCI data register. When TDRE is 1, the transmit data register (SCIDRH/L) is empty and can receive a new value to transmit. Clear TDRE by reading SCI status register 1 (SCISR1), with TDRE set and then writing to SCI data register low (SCIDRL).
- 1 = Byte transferred to transmit shift register; transmit data register empty
- 0 = No byte transferred to transmit shift register

RDRF — Receive Data Register Full Flag

- RDRF is set when the data in the receive shift register transfers to the SCI data register. Clear RDRF by reading SCI status register 1 (SCISR1) with RDRF set and then reading SCI data register low (SCIDRL).
- 1 = Received data available in SCI data register
- 0 = Data not available in SCI data register

The SCIDRL register serial input/output data. Write a subroutine that inputs one ASCII character from the serial port. Your subroutine should wait for new input using gadfly synchronization. The subroutine uses return by value parameter passing with RegB. You don’t need to write the initialization ritual. COMMENTS WILL BE GRADED.

```
SCISR1  equ $00CC ; SCI Status Register 1
SCIDRL  equ $00CF ; SCI Data Register Low
```
**Question 13.** The objective of this problem is to implement the following two-input two-output Mealy finite state machine. The state graph and ROM-based data structure are given. Your job is to write the entire main program, including reset vector, to run this machine. The FSM repeats over and over the following sequence

1) Input from PTT;
2) Output to PTM (depends on input and state);
3) Change states (depends on input and state).

The initial state is **Stop**.

You will use the following linked data structure (without copying it to the answer sheet)

```assembly
org $4000       ; Put in ROM
Stop fcb 0,0,0,1 ; Outputs
           fdb Stop,Stop,Wait,Error ; Next states
Wait  fcb 0,1,0,2
       fdb Stop,Error,Wait,Go
Go    fcb 1,0,0,0
       fdb Error,Stop,Wait,Go
Error fcb 0,0,0,0
         fdb Error,Error,Error,Error
```

The inputs are connected to PTT bits 1,0 and the outputs are connected to PTM bits 1,0.

You may use the following I/O port definitions (without copying them to the answer sheet)

```assembly
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
```
<table>
<thead>
<tr>
<th>postbyte,xb</th>
<th>syntax</th>
<th>mode</th>
<th>explanations</th>
<th>rr register</th>
</tr>
</thead>
<tbody>
<tr>
<td>rr0000000</td>
<td>,r</td>
<td>IDX</td>
<td>5-bit constant, n=0</td>
<td>00 X</td>
</tr>
<tr>
<td>rr00nnnn</td>
<td>n,r</td>
<td>IDX</td>
<td>5-bit constant, n=0 to +15</td>
<td>01 Y</td>
</tr>
<tr>
<td>rr01nnn</td>
<td>-n,r</td>
<td>IDX</td>
<td>post-decrement, n=1 to 8</td>
<td>10 SP</td>
</tr>
<tr>
<td>rr10n00n</td>
<td>n,r+</td>
<td>IDX</td>
<td>post-increment, n=1 to 8</td>
<td>11 PC</td>
</tr>
<tr>
<td>rr101n0n</td>
<td>n,r-</td>
<td>IDX</td>
<td>pre-decrement, n=1 to 8</td>
<td></td>
</tr>
<tr>
<td>rr110nnn</td>
<td>n,</td>
<td>IDX</td>
<td>pre-increment, n=1 to 8</td>
<td></td>
</tr>
<tr>
<td>rr111nnn</td>
<td>+r</td>
<td>IDX</td>
<td>Reg A accumulator offset</td>
<td></td>
</tr>
<tr>
<td>rr100000</td>
<td>A,r</td>
<td>IDX</td>
<td>Reg B accumulator offset</td>
<td></td>
</tr>
<tr>
<td>rr10110</td>
<td>D,r</td>
<td>IDX</td>
<td>Reg D accumulator offset</td>
<td></td>
</tr>
<tr>
<td>rr0000ff</td>
<td>n,r</td>
<td>IDX1</td>
<td>9-bit const, n 16 to 255</td>
<td></td>
</tr>
<tr>
<td>rr001ff</td>
<td>-n,r</td>
<td>IDX1</td>
<td>9-bit const, n -256 to -16</td>
<td></td>
</tr>
<tr>
<td>rr00ll00</td>
<td>eff</td>
<td>IDX2</td>
<td>16-bit const, any 16-bit n</td>
<td></td>
</tr>
<tr>
<td>rr00ll01</td>
<td>[D,r]</td>
<td>[IDX2]</td>
<td>Reg D offset, indirect</td>
<td></td>
</tr>
<tr>
<td>rr00ll11</td>
<td>[D,IDX]</td>
<td>Reg D offset, indirect</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**BRA**

**Operation:** \((P) + 0002 + Rel \Rightarrow P\)

Unconditional branch to an address calculated as shown in the expression. Rel is a relative offset stored as a two’s complement number in the second byte of the branch instruction.

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRA ne8</td>
<td>REL</td>
<td>20 rr</td>
</tr>
</tbody>
</table>

**LDX**

**Operation:** \((M : M+1) \Rightarrow X\)

Loads the most significant byte of index register X with the contents of memory location M, and loads the least significant byte of X with the contents of the next byte of memory at M+1.

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDX #opr8a</td>
<td>IMM</td>
<td>C0 jj xx</td>
</tr>
<tr>
<td>LDX opr8a</td>
<td>DIR</td>
<td>DE dd</td>
</tr>
<tr>
<td>LDX opr16a</td>
<td>EXT</td>
<td>F0 hh 11</td>
</tr>
<tr>
<td>LDX opr0x, xysp</td>
<td>IDX</td>
<td>F0 x0 fi</td>
</tr>
<tr>
<td>LDX opr16, xysp</td>
<td>IDX2</td>
<td>F0 x0 ee ff</td>
</tr>
<tr>
<td>LDX [D, xysp]</td>
<td>[IDX]</td>
<td>FF xe ee ff</td>
</tr>
<tr>
<td>LDX [opr16, xysp]</td>
<td>[IDX2]</td>
<td>FF xe ee ff</td>
</tr>
</tbody>
</table>

**LDAA**

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

Loads the content of memory location M into accumulator A. The condition codes are set according to the data.

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAA #opr8i</td>
<td>IMM</td>
<td>B0 11</td>
</tr>
<tr>
<td>LDAA opr8a</td>
<td>DIR</td>
<td>96 dd</td>
</tr>
<tr>
<td>LDAA opr16a</td>
<td>EXT</td>
<td>B6 hh 11</td>
</tr>
<tr>
<td>LDAA opr0x, xysp</td>
<td>IDX</td>
<td>A6 x0 fi</td>
</tr>
<tr>
<td>LDAA opr16, xysp</td>
<td>IDX2</td>
<td>A6 x0 ee ff</td>
</tr>
<tr>
<td>LDAA [D, xysp]</td>
<td>[IDX]</td>
<td>A6 x0 ee ff</td>
</tr>
<tr>
<td>LDAA [opr16, xysp]</td>
<td>[IDX2]</td>
<td>A6 x0 ee ff</td>
</tr>
</tbody>
</table>

**STAA**

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

Stores the content of accumulator A in memory location M. The content of A is unchanged.

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>STAA opr8a</td>
<td>DIR</td>
<td>5A 01</td>
</tr>
<tr>
<td>STAA opr16a</td>
<td>EXT</td>
<td>7A hh 11</td>
</tr>
<tr>
<td>STAA opr0x, xysp</td>
<td>IDX</td>
<td>5A x0 fi</td>
</tr>
<tr>
<td>STAA opr16, xysp</td>
<td>IDX2</td>
<td>5A x0 ee ff</td>
</tr>
<tr>
<td>STAA [D, xysp]</td>
<td>[IDX]</td>
<td>5A x0 ee ff</td>
</tr>
<tr>
<td>STAA [opr16, xysp]</td>
<td>[IDX2]</td>
<td>5A x0 ee ff</td>
</tr>
</tbody>
</table>
aba 8-bit add RegY=RegX+RegB
abx unsigned add RegY=RegX+RegB
aby unsigned add RegY=RegX+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/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 ≥
bgt branch if signed >
bhi branch if unsigned >
bhs branch if unsigned ≥
bits 8-bit and with RegA, sets CCR
bitb 8-bit and with RegB, sets CCR
ble branch if signed ≤
bleq branch if result is zero
bge branch if signed ≥
ligt branch if signed >
blhi branch if unsigned >
blhs branch if unsigned ≥
blt branch if signed <
bltu branch if unsigned <
bltz branch if unsigned ≤
bit branch if signed <
bitm branch if result is negative (N=1)
bne branch if result is nonzero (Z=0)
bpl branch if result is positive
ble branch if result is positive (N=0)
br branch always
brclr branch if bits are clear,
brn branch never
brset branch if bits are set
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
cle clear carry bit, C=0
cleq clear carry bit, C=0, enable interrupts
cir 8-bit Memory clear
clra RegA clear
clrb RegB clear
cliv clear overflow bit, V=0
cmpa 8-bit compare RegA with memory
cmpb 8-bit compare RegB with memory
cmpc 8-bit compare RegA with carry set
com 8-bit logical complement to Memory
coml 8-bit logical complement to RegB
combine 8-bit logical complement to RegB
cmpeq 16-bit compare RegD with memory
cmpeq 16-bit compare RegX with memory
cpy 16-bit compare RegY with memory
cda 8-bit divide memory
cee 8-bit decrement memory
ccda 8-bit decrement RegA
cdcb 16-bit decrement RegB
ceği 8-bit decrement RegX
cke 16-bit decrement RegY
cdiv 16-bit divide RegY
lce 8-bit compare RegY with RegX, unsigned divide
lceq 8-bit compare RegY with RegX, unsigned divide
lsb 16-bit signed minimum in memory
lsbx 16-bit signed minimum in RegX
lsby 16-bit signed minimum in RegY
ema 16 by 16 signed mult, 32-bit add
ema1 8-bit unsigned maximum in RegD
emax 16-bit unsigned maximum in RegD
emind 16-bit unsigned minimum in RegD

### 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>equ     Define a constant symbol</td>
</tr>
<tr>
<td>dc.b</td>
<td>Allocates byte(s) of storage with initialized values</td>
</tr>
<tr>
<td>db</td>
<td>fcb     Allocate word(s) of storage with initialized values</td>
</tr>
<tr>
<td>byte</td>
<td>fcc     Create an ASCII string (no termination character)</td>
</tr>
<tr>
<td>dfc</td>
<td>dw      Allocate 32-bit long word(s) of storage with initialized values</td>
</tr>
<tr>
<td>.word</td>
<td>ds      Allocate 32-bit long word(s) of storage without initialization</td>
</tr>
<tr>
<td>.b</td>
<td>ds.b    Allocate bytes of storage without initialization</td>
</tr>
<tr>
<td>.w</td>
<td>ds.w    Allocate bytes of storage without initialization</td>
</tr>
<tr>
<td>.blkb</td>
<td>ds.l    Allocate 32-bit words of storage without initialization</td>
</tr>
<tr>
<td>.blkw</td>
<td></td>
</tr>
</tbody>
</table>