EEE095/6S 2022 Class Test 2

ANSWER SHEET

Student Number:
Name:

11 October 2022 [100]

Solutions!

Instructions:

Please use this attached answer sheets for answering the questions for this test. NB: write you student number and name on the box at the top of each answer page.

Section 1: Multiple Choice [7 x 5marks = 35 marks]

Mark your answer option with a tick (✔) or a cross (**X**), to delete an entry scratch it out (♒) thoroughly so that it looks like neither a tick or a cross otherwise you'd get 0 for the question. Assume you are only to select one answer option for question unless stated otherwise.

Question	A	B	C	D
1.1		X		
1.2			X	
1.3				X
1.4		X		
1.5	X			
1.6				X
1.7		X		

Section 2: YES/NO [5 x 5marks = 25 marks]

Mark your answer option with a tick (✔) or a cross (**X**), to delete an entry scratch it out (♒) thoroughly so that it looks like neither a tick or a cross otherwise you'd get 0 for the question. Assume you are only to select one answer option, Yes or No, unless it is stated that short circuiting your answer is permissible.

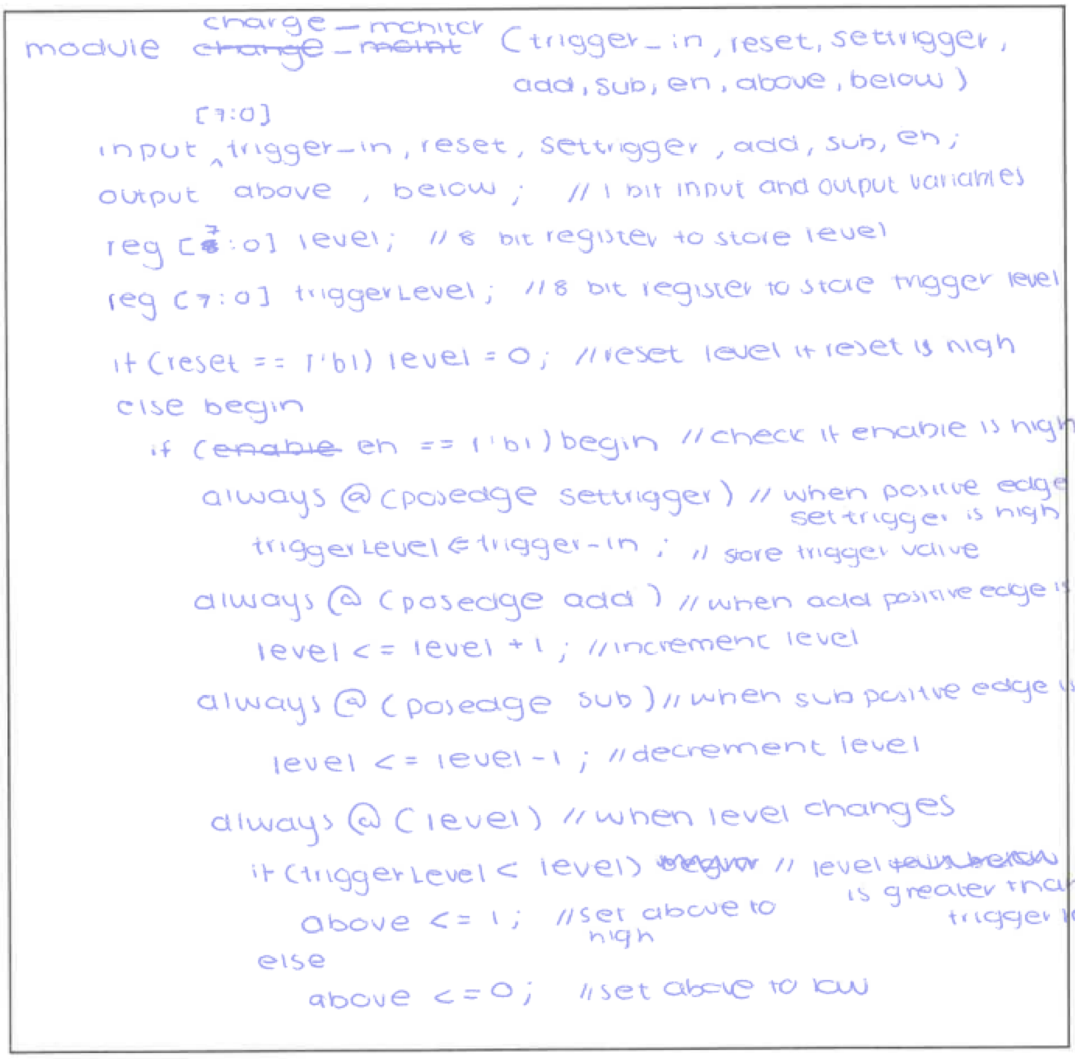
Question	YES	NO

2.1		X
2.2		X
2.3		X
2.4		X
2.5	X	

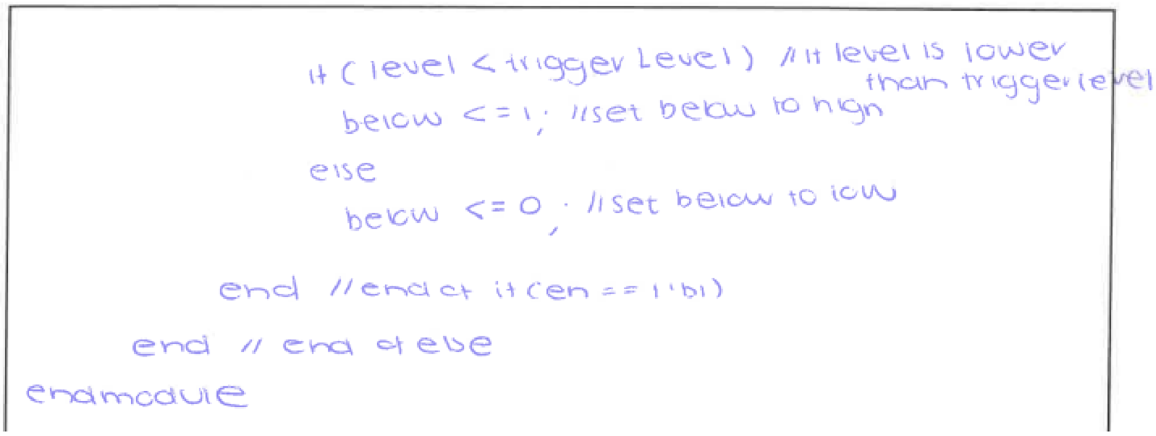
Section 3: Short Answers [40 marks]

Solutions!

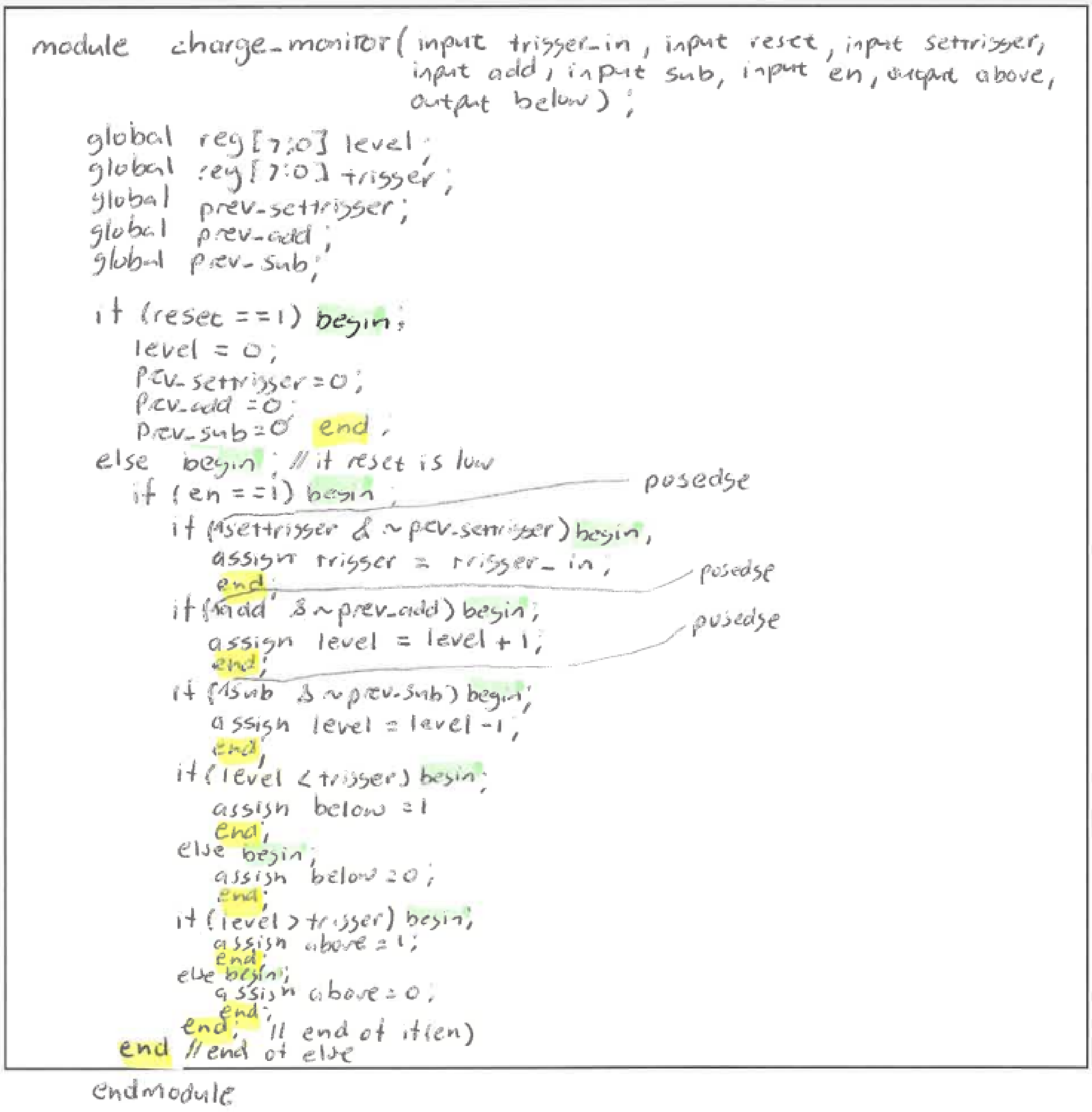
Question 3.1 Answer [25 marks]



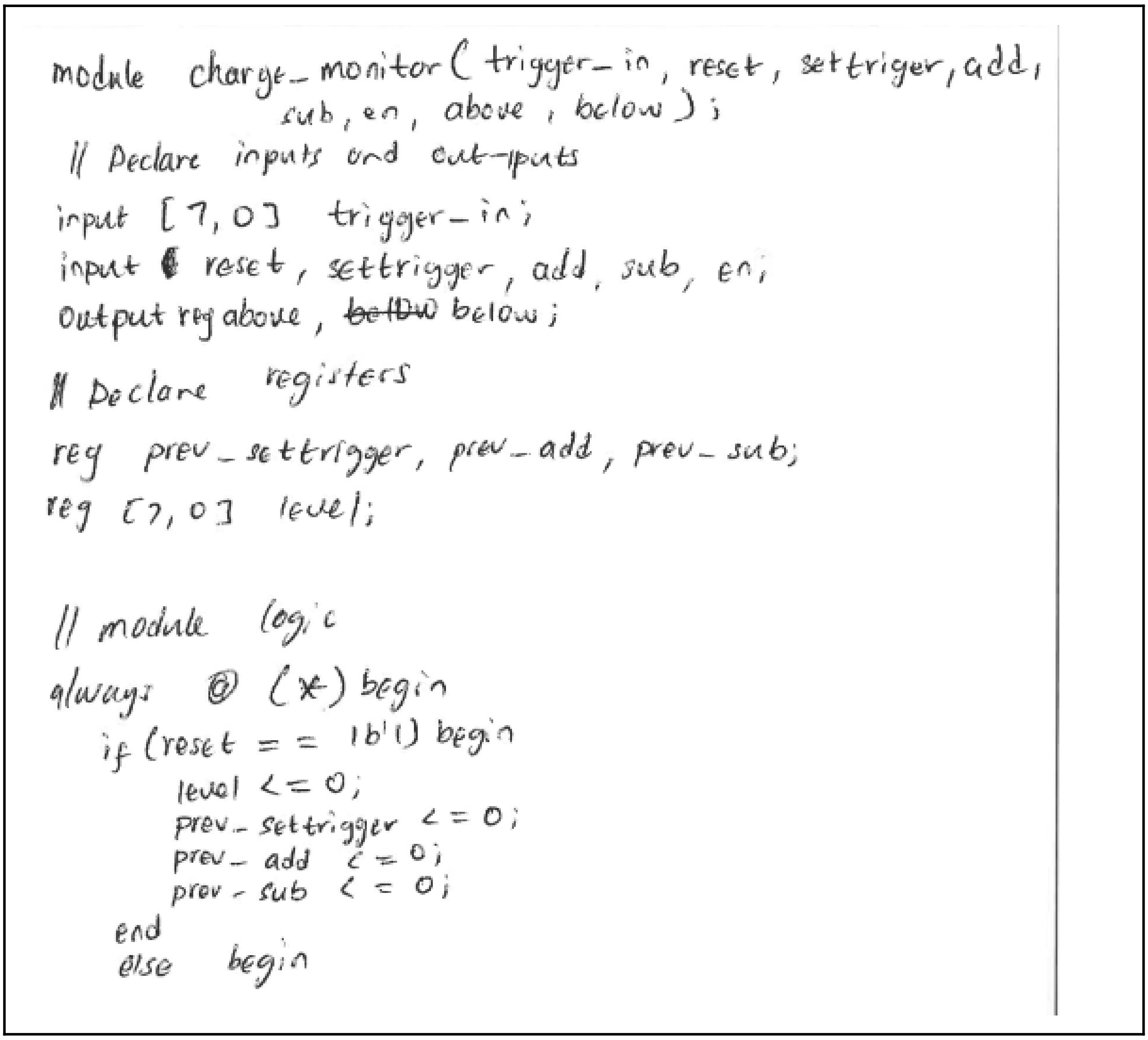
```
module charge_monitor ~~change_maint~~ (trigger_in, reset, settrigger,
                                add, sub, en, above, below)
      [7:0]
   input ^trigger_in, reset, settrigger, add, sub, en;
   output above , below ;   // 1 bit input and output variables
   reg [7:0] level;  // 8 bit register to store level
   reg [7:0] triggerLevel;  //8 bit register to store trigger level

   if (reset == 1'b1) level = 0;  //reset level if reset is high
   else begin
      if (~~enable~~ en == 1'b1) begin  //check if enable is high
         always @(posedge settrigger) // when positive edge of
                                            settrigger is high
            triggerLevel <= trigger_in ;  // store trigger value
         always @ (posedge add ) // when add positive edge is high
            level <= level + 1 ; //increment level
         always @ (posedge sub )// when sub positive edge is high
            level <= level - 1 ; //decrement level

         always @ (level) // when level changes
            if (triggerLevel < level) ~~begin~~ // level ~~falls below~~
                                                   is greater than
               above <= 1;  //set above to                  trigger level
                              high
            else
               above <= 0;   //set above to low
```



```
                 if ( level < trigger Level ) //if level is lower
                                                than triggerlevel
                   below <= 1; //set below to high
                 else
                   below <= 0; //set below to low

            end //end of if(en == 1'b1)

        end // end of else
endmodule
```



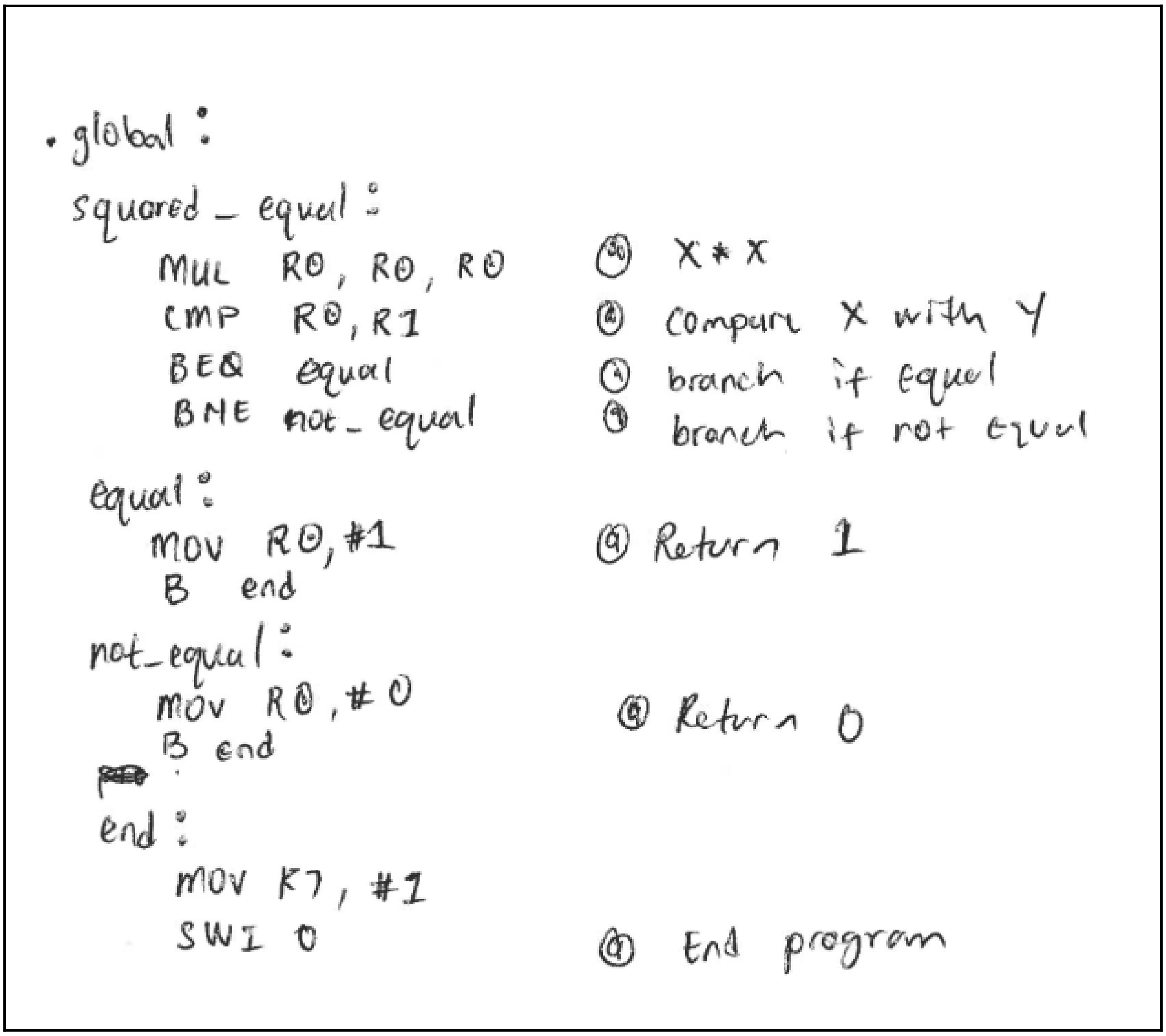
```
module  charge_monitor( input trigger_in , input reset , input settrigger,
                        input add , input sub, input en, output above,
                        output below ) ;

    global reg[7:0] level;
    global reg[7:0] trigger;
    global  prev_settrigger;
    global  prev_add;
    global  prev_sub;

    if (reset ==1) begin;
       level = 0;
       prev_settrigger=0;
       prev_add =0;
       prev_sub=0  end ,
    else  begin; //if reset is low
      if (en ==1) begin ,                                   posedge
         if (1settrigger & ~prev_settrigger) begin,
            assign trigger = trigger_in ,
            end;                                              posedge
         if (1add  & ~prev_add) begin;
            assign level = level + 1;                         posedge
            end;
         if (1sub & ~prev_sub) begin;
            assign level = level -1,
            end,
         if(level < trigger) begin;
            assign below =1
            end,
         else begin;
            assign below =0;
            end;
         if (level > trigger) begin;
            assign above =1;
            end;
         else begin;
            assign above =0;
            end;
        end; // end of if(en)
    end //end of else
endmodule
```



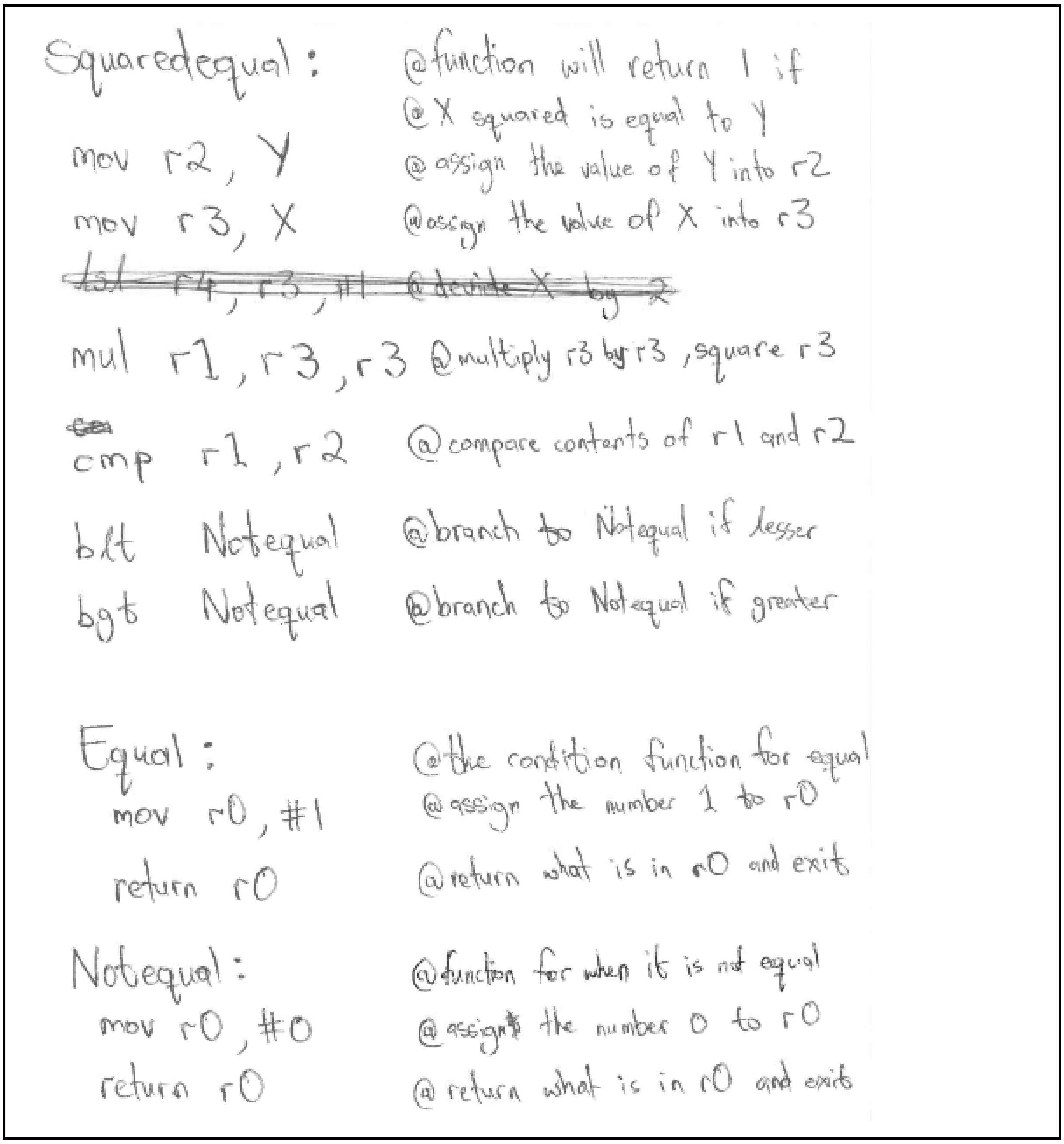
```
module  charge_monitor( trigger_in, reset, settriger, add,
                  sub, en, above, below );
 // Declare inputs and out-puts

input [7,0]  trigger_in;
input  reset, settrigger, add, sub, en;
Output reg above, ~~below~~ below;

// Declare  registers

reg  prev_settrigger, prev_add, prev_sub;
reg [7,0]  level;


// module  logic
always  @ (*) begin
   if (reset == 1b'1) begin
        level <= 0;
        prev_settrigger <= 0;
        prev_add  <= 0;
        prev_sub  <= 0;
   end
   else  begin
```

```
if (en = = 1b'1) begin
      if(settrigger &&(~prev_settriged) begin
            trigger<= trigger_in;
        end
        if ( add && ~prev_sub) begin
             level <= level + 1b'1;
         end
         if(sub && ~prev_sub) begin
             level <= level - 1b'1;
         end

         if ( level < trigger) begin
              qbove <= 1b'1;
          end
          else begin
               ~~abo~~
               below <=1b'0;
            end

         if(level > trigger) begin
              above <= 1b'1;
          else begin
              above <= 1b'0;
          end
        end

   prev_settrigger <= settrigger;
   prev_add <= add;
   prev_sub < = sub;
 End
endmodule
```

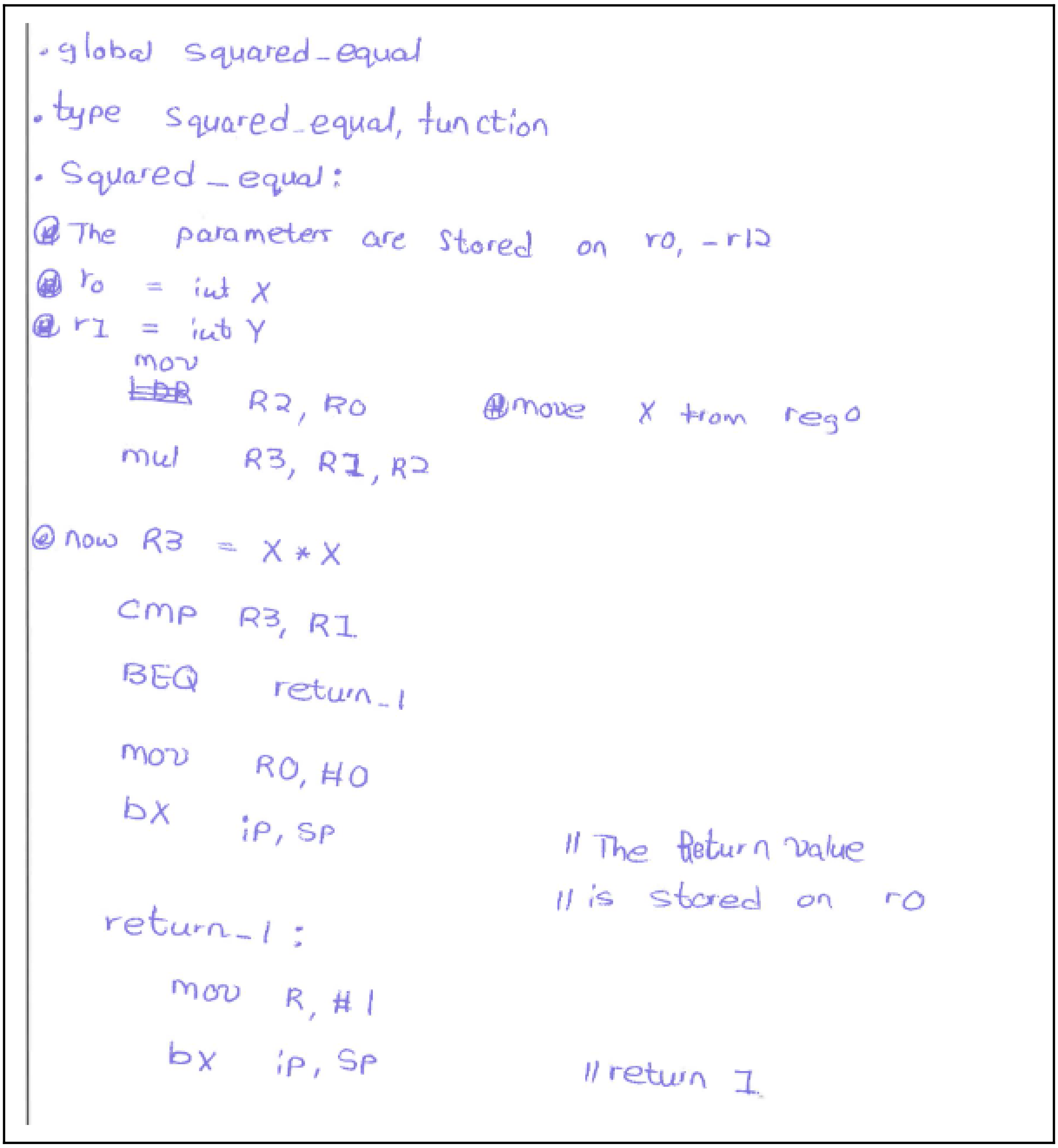
Question 3.2 Answer [15 marks]



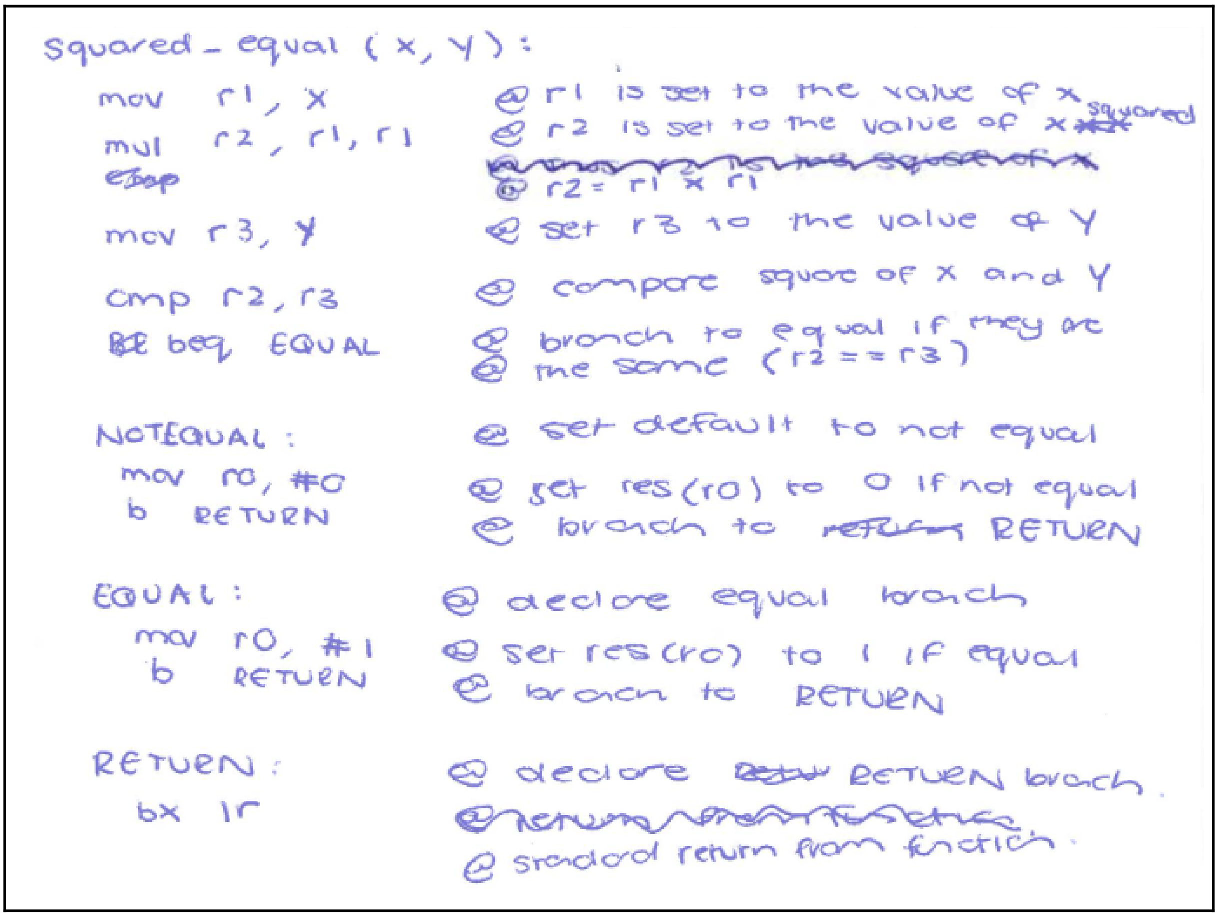
```
.global :
squared_equal :
    MUL  R0, R0, R0        @  X * X
    CMP  R0, R1            @  Compare X with Y
    BEQ  equal             @  branch if equal
    BNE  not_equal         @  branch if not equal

 equal :
    MOV  R0, #1            @ Return 1
    B  end
 not_equal :
    MOV  R0, #0            @ Return 0
    B end

 end :
    MOV R7, #1
    SWI 0                  @  End program
```



```
Squaredequal:          @function will return 1 if
                       @X squared is equal to Y
 mov r2, Y             @assign the value of Y into r2
 mov r3, X             @osign the value of X into r3
 ~~lsl r4, r3, #1      @devide X by 2~~
 mul r1, r3, r3        @multiply r3 by r3 ,square r3
 cmp r1 ,r2            @compare contents of r1 and r2
 blt Notequal          @branch to Notequal if lesser
 bgt Notequal          @branch to Notequal if greater


 Equal:                @the condition function for equal
  mov r0, #1           @assign the number 1 to r0
  return r0            @return what is in r0 and exit

 Notequal:             @function for when it is not equal
  mov r0 , #0          @assign the number 0 to r0
  return r0            @return what is in r0 and exits
```



```
.global squared_equal
.type squared_equal, function
. Squared_equal:
@The parameters are Stored on r0, -r12
@ r0 = int X
@ r1 = int Y
      mov
      ~~LDR~~   R2, R0          @move X from reg0
      mul   R3, R1, R2

@now R3 = X*X
      cmp   R3, R1
      BEQ   return_1
      mov   R0, #0
      bx    ip, sp
                                // The Return value
                                // is stored on r0
   return_1:
        mov   R, #1
        bx    ip, sp            //return 1.
```



```
squared_equal ( x, y ):
    mov   r1, x           @ r1 is set to the value of x
    mul   r2, r1, r1      @ r2 is set to the value of x squared
                          ~~@ r2 is the square of x~~
                          @ r2 = r1 x r1
    mov r3, y             @ set r3 to the value of y

    cmp r2, r3            @ compare square of x and y
    ~~BE~~ beq  EQUAL     @ branch to equal if they are
                          @ the same (r2 == r3)

    NOTEQUAL :            @ set default to not equal
     mov  r0, #0          @ set res(r0) to 0 if not equal
     b  RETURN            @ branch to ~~return~~ RETURN

    EQUAL :              @ declare equal brach
      mov  r0, #1        @ set res(r0) to 1 if equal
      b   RETURN          @ branch to  RETURN

    RETURN :             @ declare ~~Retu~~ RETURN brach.
      bx  lr             ~~@ return from function~~
                          @ standard return from function.
```

```
Squared_equal:
   mov   r1 , x        ;    @ r1 = x
   MUL   r1 , r1 , r1  ;    @ r1 = r1 * r1 = x * x
   mov   r2 , Y        ;    @ r2 = Y
   CMP   r1 , r2       ;
   BEQ   Equal ;            @ If r1 == r2 , branch to Equal
   B     NotEqual;          @ Branch to NotEqual - will happen if r1 != r2
                                                   ~~Reg weren't equal~~

 Equal:
      mov   r0, #1 ;   @ r0=1;
        b    Return ;
NotEqual:
       mov   r0 , #0;   @ r0=0;
        b    Return;

Return:
     bx   lr ;        @ retun value stored at default location r0
```

end of test solutions