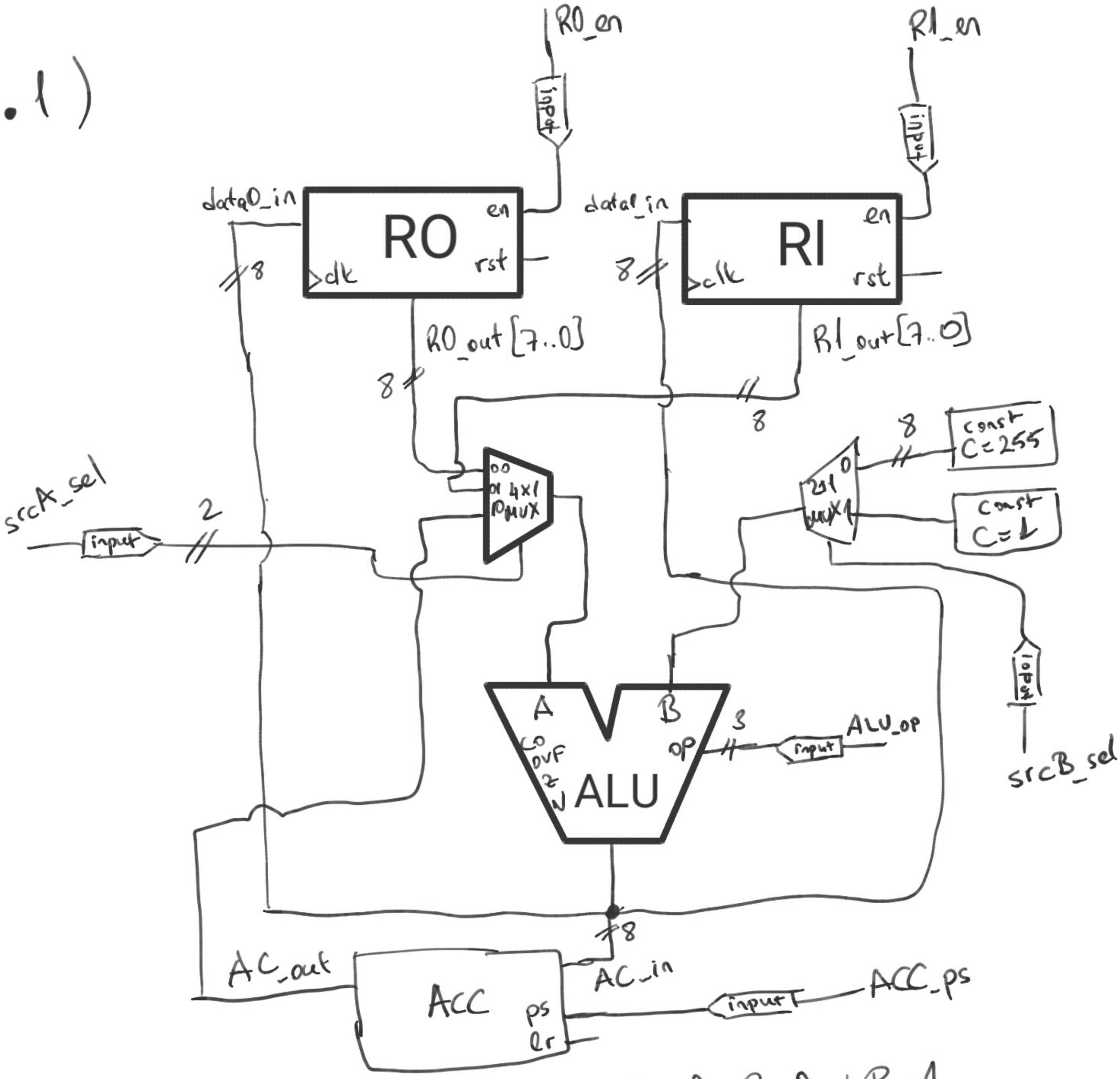
Ahmet Salih Aksakal
2030005

EE446 - Experiment 2
Preliminary Work

1.1)

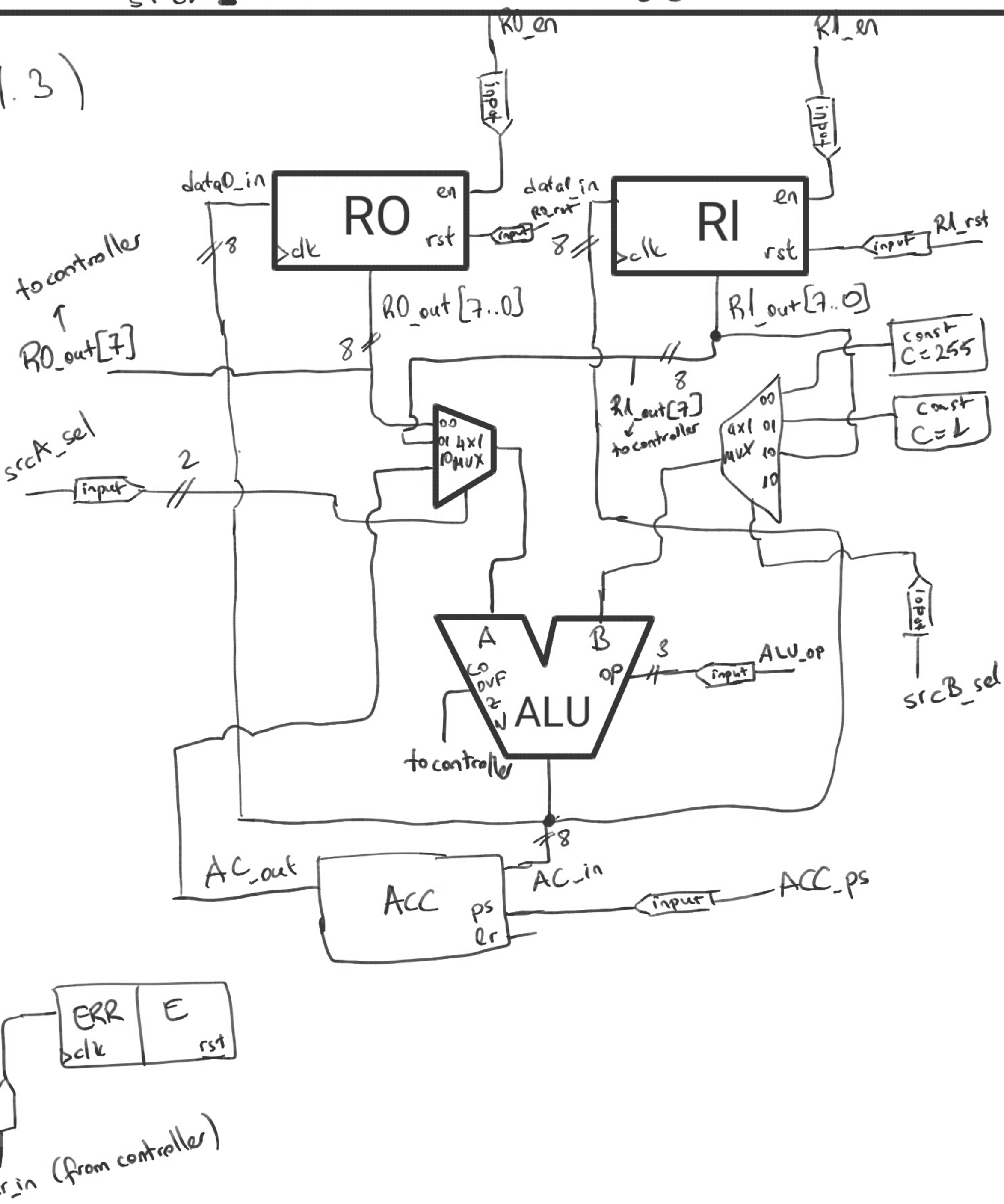


→ 1st cycle:
- srcA_sel = 00 or 01 for Reg0 and Reg1
- ALU_op = 110 for EXOR with 1111_1111
  ↳ This takes 1's comp.
- ACC_ps = 1, • srcB_sel = 0

→ 2nd cycle:
- srcA_sel = 10 from loaded temporary result
- ALU_op = 000 for adding 1
- we of Reg0 or Reg1 = 1, • srcB_sel = 1

1.2) MSB of Reg0 & Reg1 goes into controller.
srcA_sel will be set accordingly by the controller.

1.3)



1.4) For signed multiplication the best way is to implement Booth's Algorithm.

→ The idea is to make the multiplication by only adding and subtracting power of 2's of the number. For example say we need to find 7x12

→ We can represent 12 as 16-4

that makes $7 \times 12 = 7 \times (16 - 4)$

$= 7 \cdot 2^4 - 7 \cdot 2^2$

= LSL #7, 4

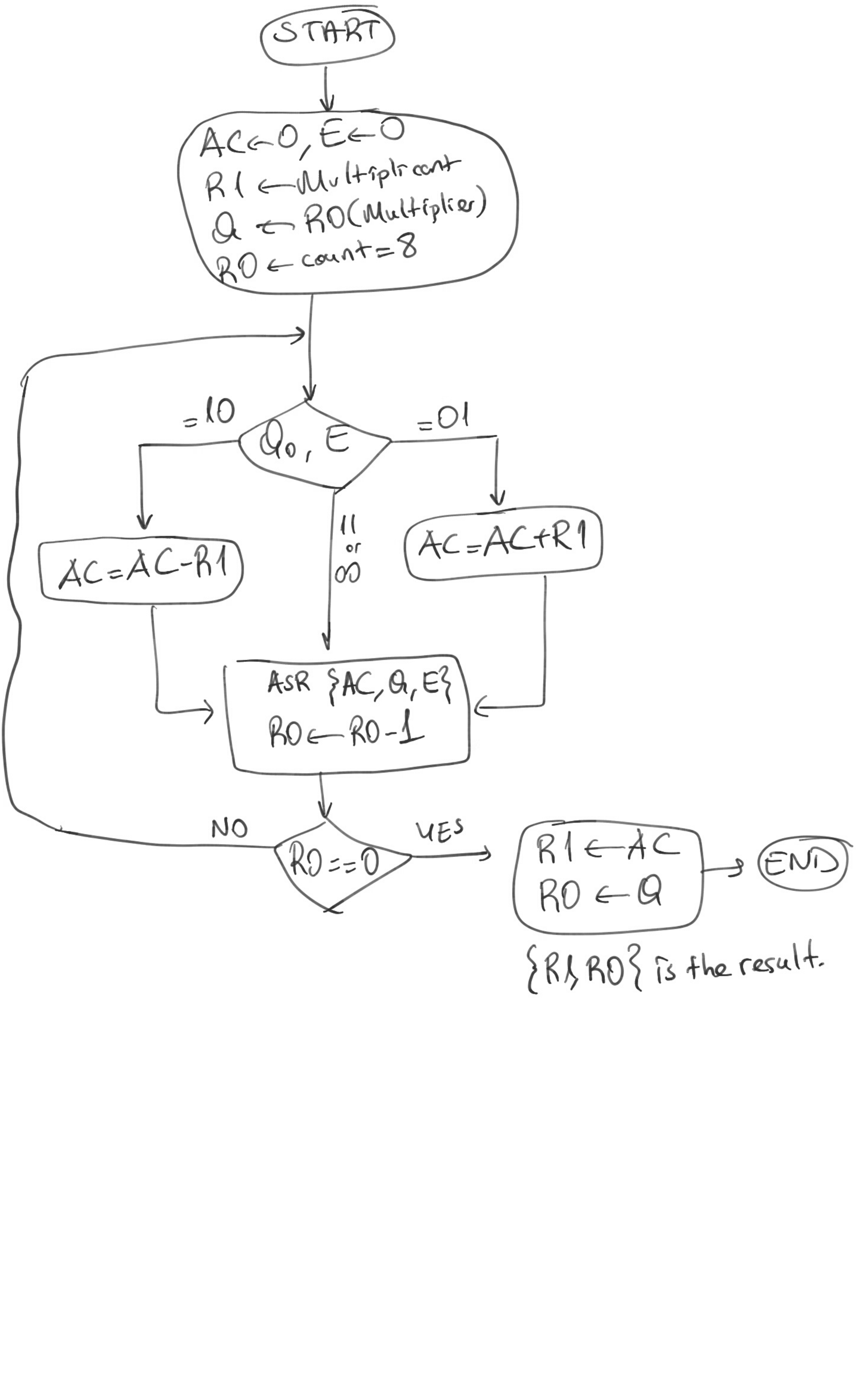
− LSL #7, 2

RESULT

→ To implement this on an algorithm, Booth says that we need to check pairs of the multiplier. By using an extra bit we can implement it on h.w.

Extra bit → E
Multiplicant → R1
Multiplier → Q

and AC is temp. reg. and at final the MSB of the result.



{R1, R0} is the result.

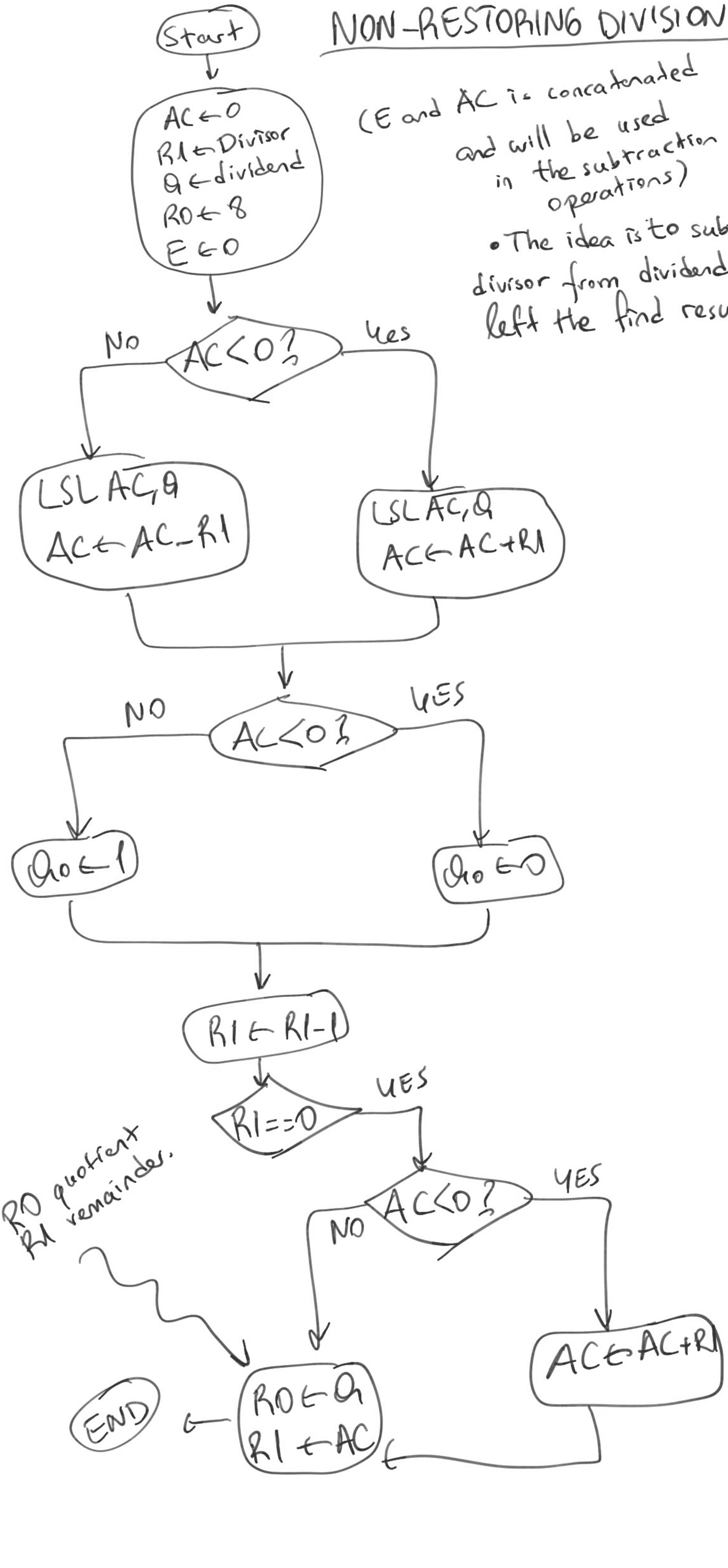
RO_en
R1_en
comes from 2x1 MUX below.
comes from 2x1 MUX below.
data0_in
data1_in
RO_rst
R1_rst
RO
R1
clk
en
rst
RO_out[7..0]
R1_out[7..0]
to controller
RO_out[7]
8
const C=255
const C=1
srcA_sel
2
4x1 MUX
R1_out[7]
to controller
2
srcB_sel
A
B
CO
OVF
Z
N
op
3
ALU_op
ALU
to controller (for ALP op.)
to controller (for MUL op)
ALU_out
8
RO_out[7..0] comes here
AC_out
AC_in
AC
out
in_left
in_right
ps
lr
AC_out[7]
E_out to controller
Q_out
Q_in
Q
Q[7]
ERR
E
clk
in
rst
AC_out[7]
AC_out[0]
Q_out[0]
err_in (from controller)
AC_lr
AC_ps
Q_out
Q_lr
Q_ps
8
AC_out
ALU_out
R1 data1_in pins
R1 src_sel
2x1 mux
ALU_out (comes from ALU)
RO data0_in pins.
RO src_sel.

(.5)

NON-RESTORING DIVISION ALG.

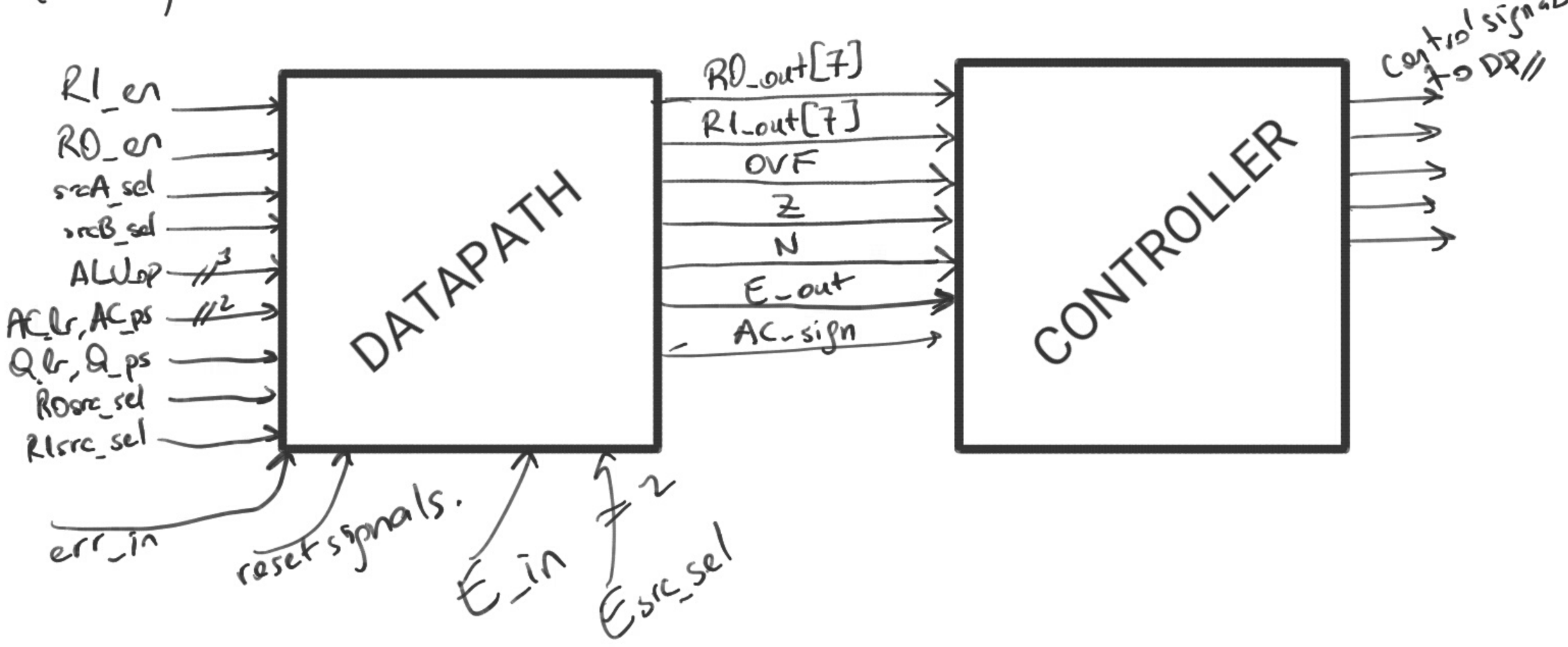
(E and AC is concatenated and will be used in the subtraction operations)

- The idea is to subtract divisor from dividend from left the find result.

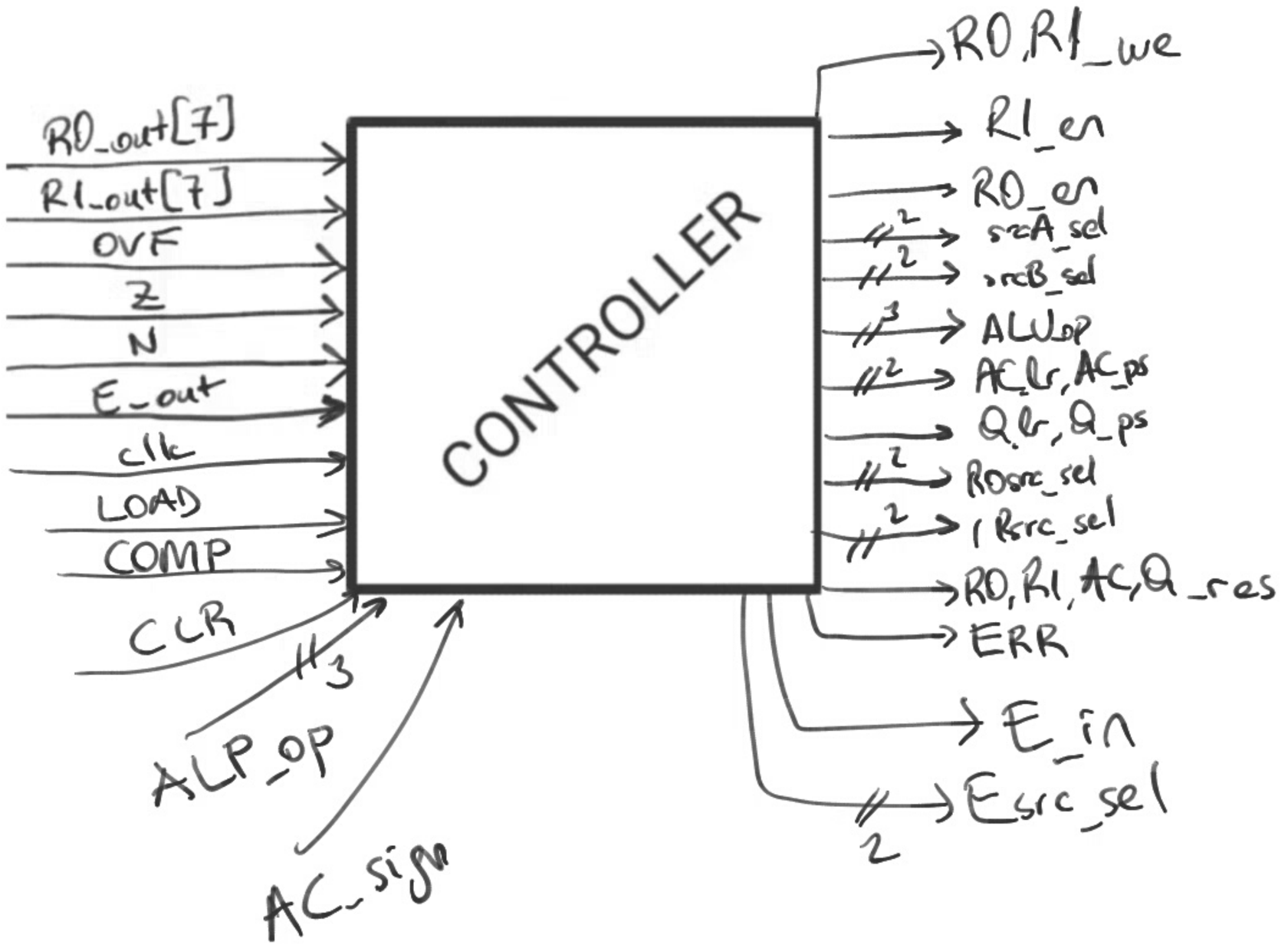


- Changes made on the datapath with blue ink.
  - → E_out sent to controller
  - → AC_out[7] is connected to E_in
  - → Q[7] is connected to AC's input_right port. For shift left operation
  - → Q's input left is connected to ground.

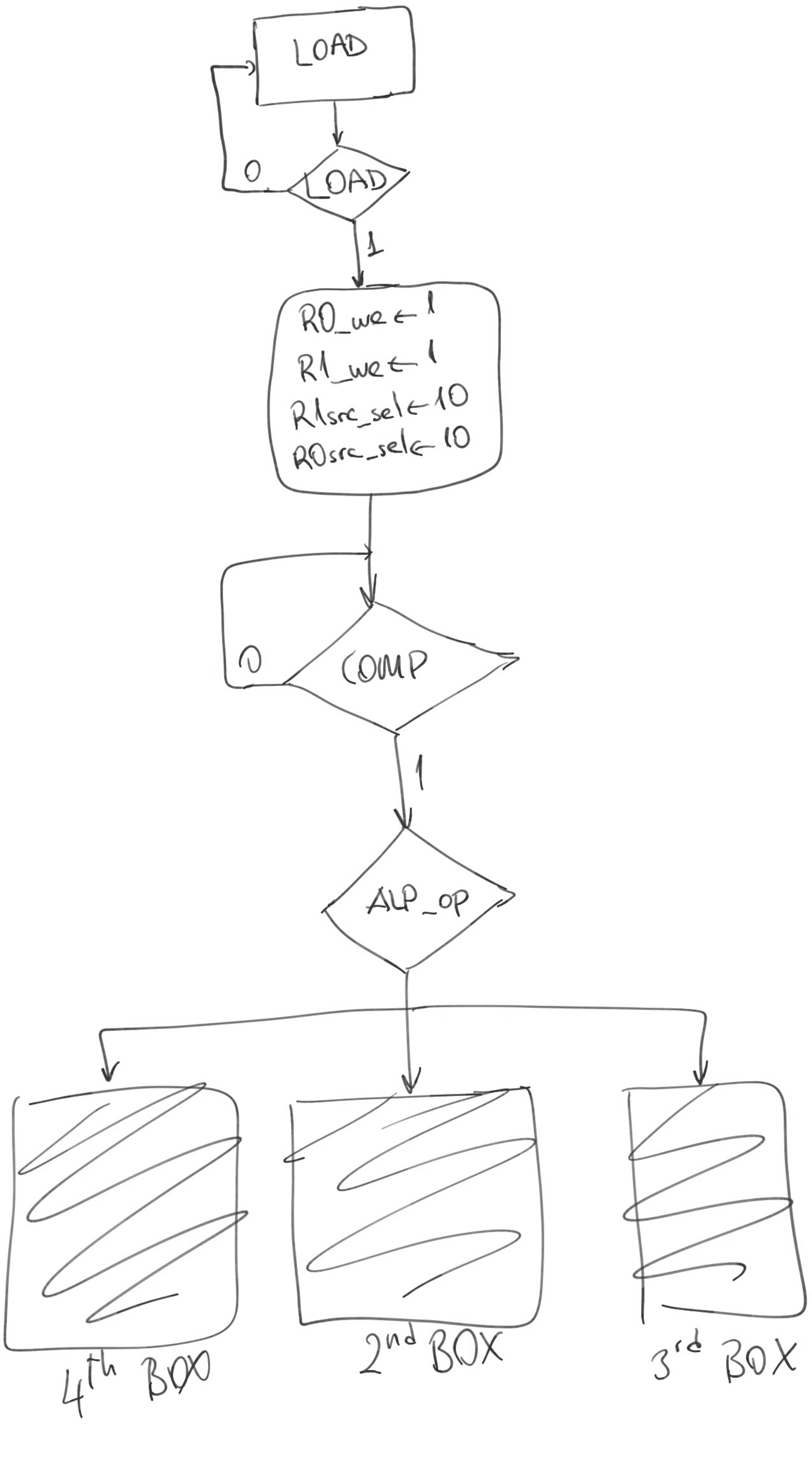
1.6)



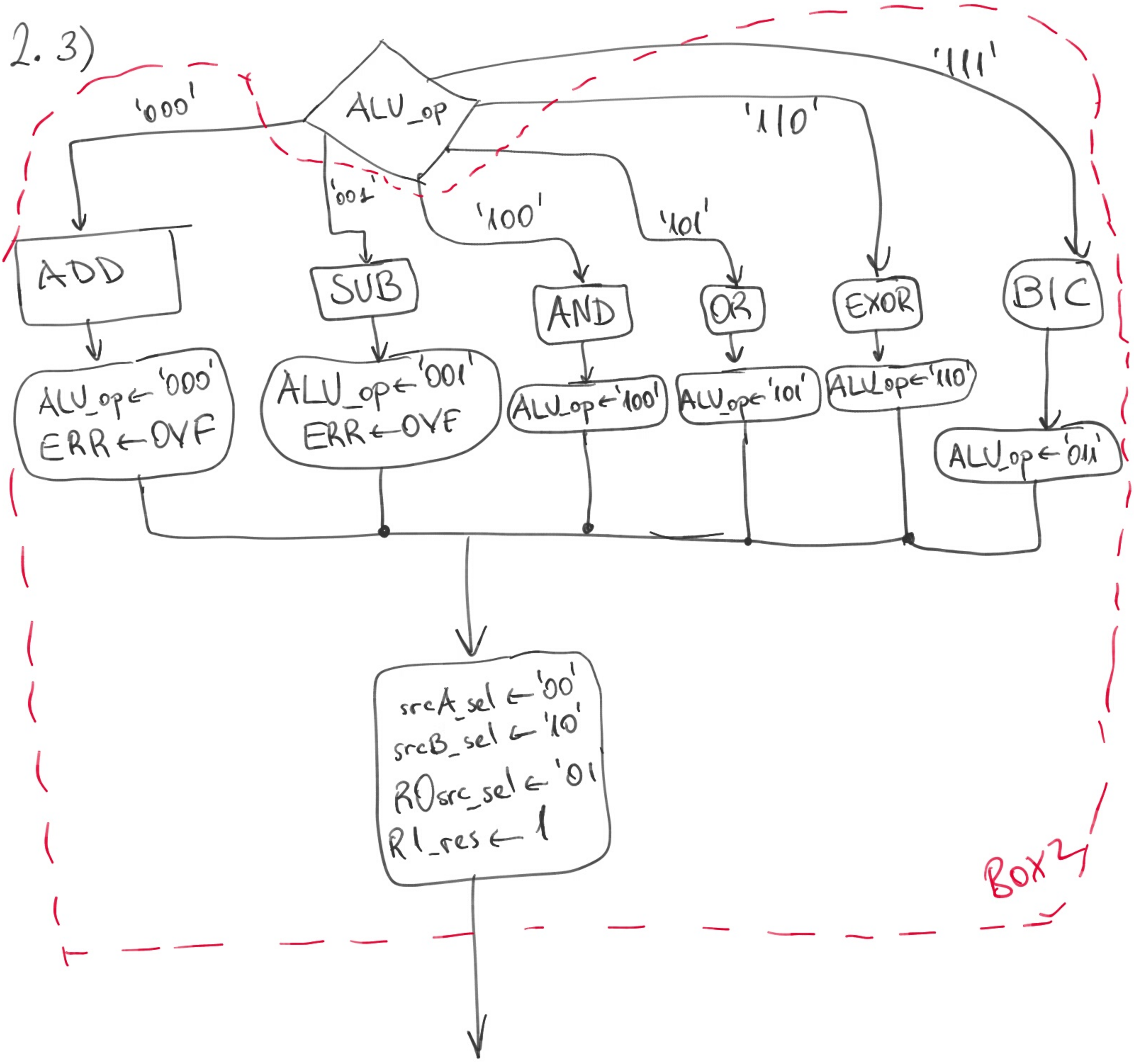
2.1)



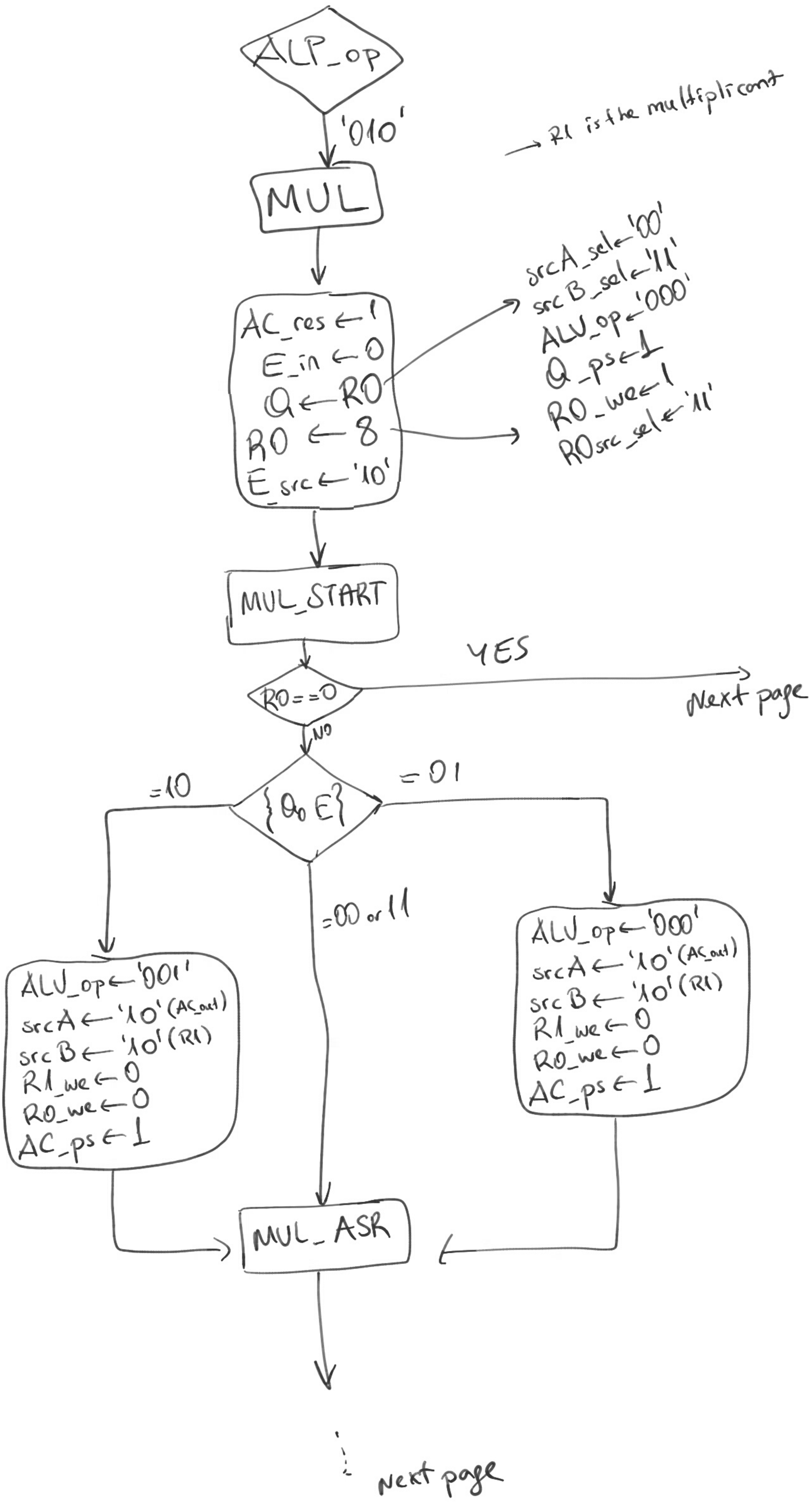
2.2) 1st Box : LOAD operation

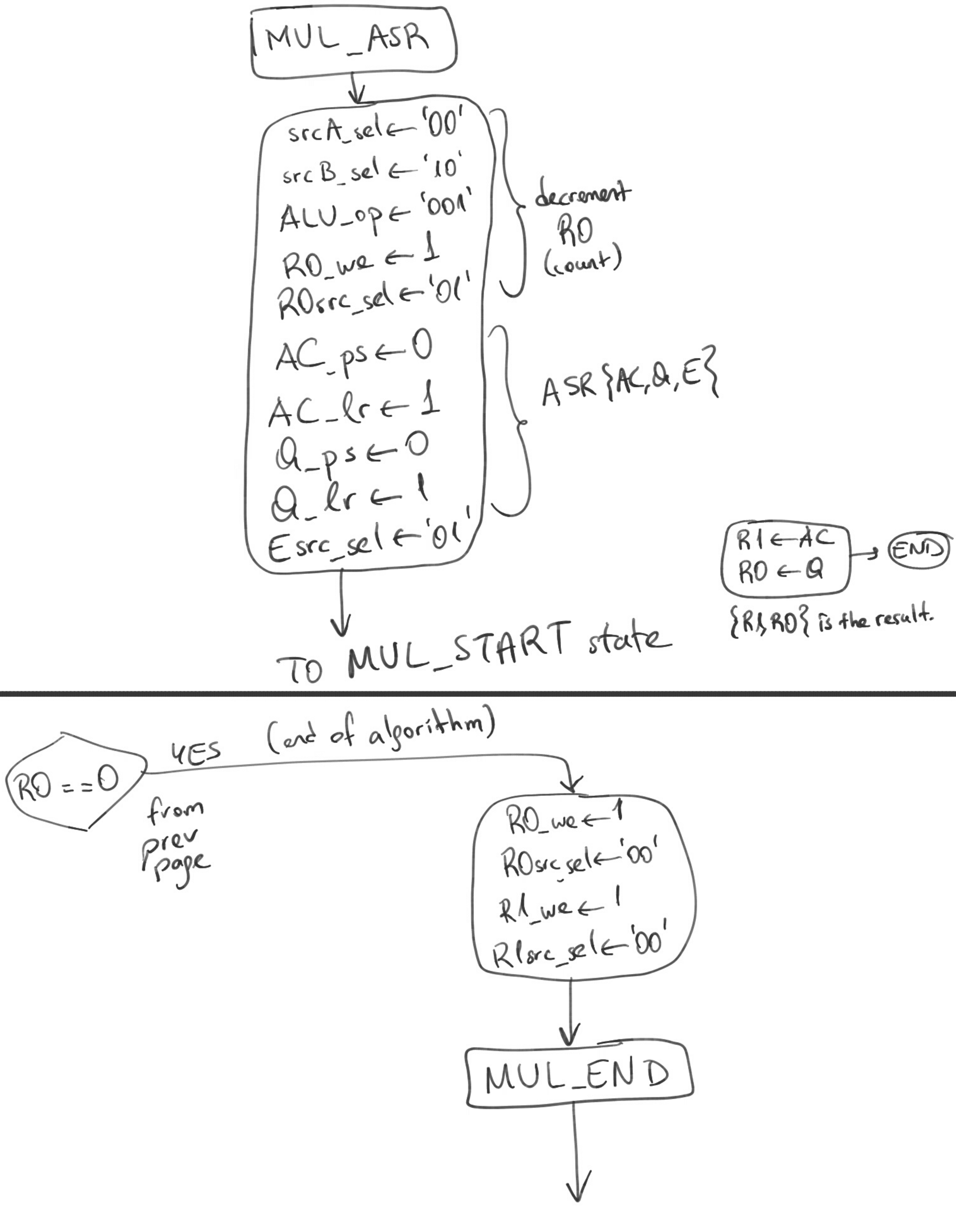


2. 3)

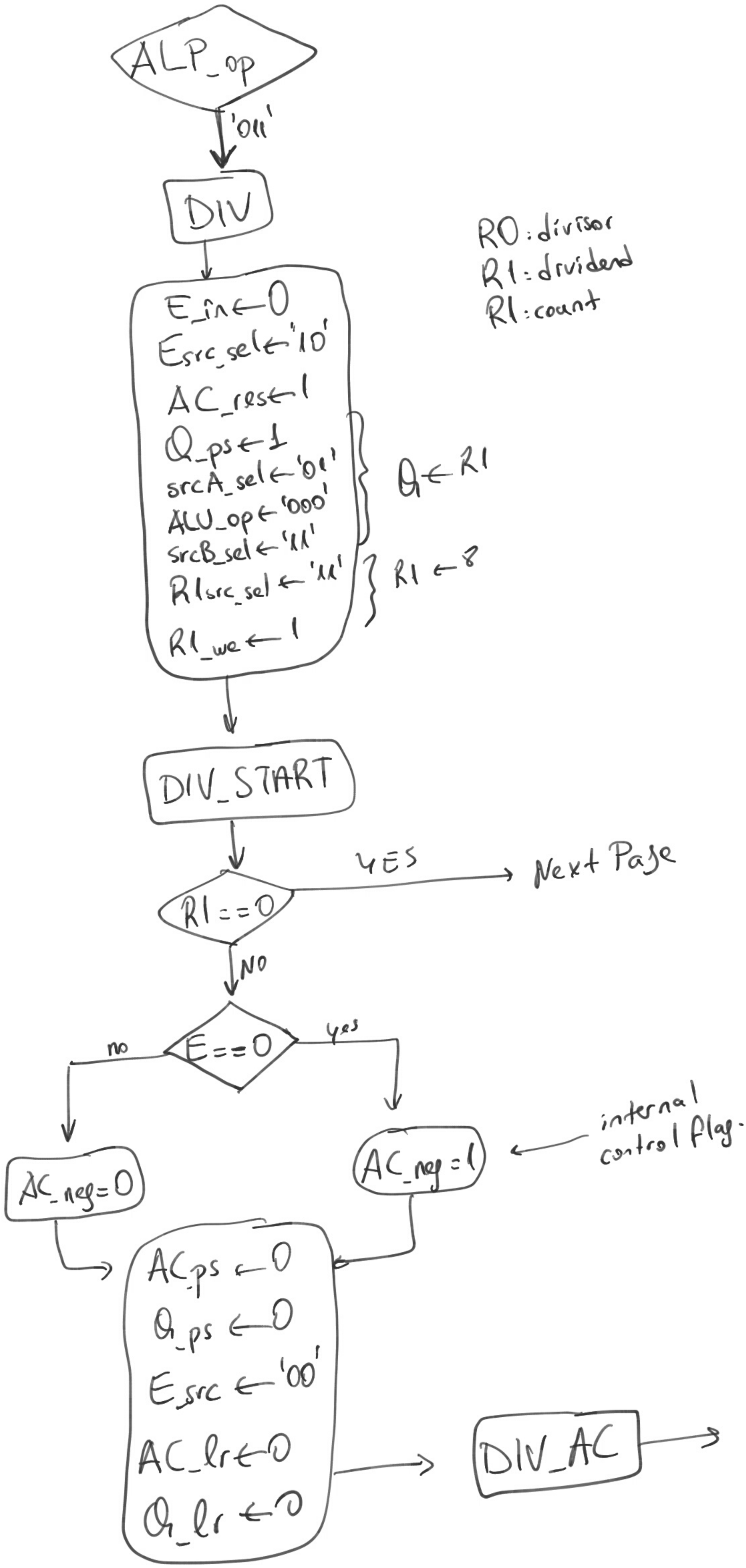

ALU_OP
'000'
'001'
'100'
'101'
'110'
'111'
ADD
SUB
AND
OR
EXOR
BIC
ALU_op ← '000'
ERR ← OVF
ALU_op ← '001'
ERR ← OVF
ALU_op ← '100'
ALU_op ← '101'
ALU_op ← '110'
ALU_op ← '011'
srcA_sel ← '00'
srcB_sel ← '10'
R0src_sel ← '01'
R1_res ← 1
Box4

2.4)


ALP_OP
'010'
MUL
AC_res ← 1
E_in ← 0
Q ← R0
R0 ← 8
E_src ← '10'
→ R1 is the multiplicand
srcA_sel ← '00'
srcB_sel ← '11'
ALU_op ← '000'
Q_ps ← 1
R0_we ← 1
R0src_sel ← '11'
MUL_START
R0 == 0
YES
Next page
NO
{0, E}
=10
=01
=00 or 11
ALU_op ← '001'
srcA ← '10' (AC_out)
srcB ← '10' (R1)
R1_we ← 0
R0_we ← 0
AC_ps ← 1
ALU_op ← '000'
srcA ← '10' (AC_out)
srcB ← '10' (R1)
R1_we ← 0
R0_we ← 0
AC_ps ← 1
MUL_ASR
Next page


MUL_ASR
srcA_sel ← '00'
srcB_sel ← '10'
ALU_op ← '001'
R0_we ← 1
R0src_sel ← '01'
decrement R0 (count)
AC_ps ← 0
AC_lr ← 1
Q_ps ← 0
Q_lr ← 1
Esrc_sel ← '01'
ASR {AC, Q, E}
TO MUL_START state
R1 ← AC
R0 ← Q
END
{R1, R0} is the result.
R0 == 0
YES
(end of algorithm)
from prev page
R0_we ← 1
R0src_sel ← '00'
R1_we ← 1
R1src_sel ← '00'
MUL_END

2.5)



R0 : divisor
R1 : dividend
R1 : count

DIV-AC

AC_neg

0

1

ALU_op ← '000'
srcA ← '10' (AC_out)
srcB ← '10' (R1)
R1_we ← 0
R0_we ← 0
AC_ps ← 1

ALU_op ← '010'
srcA ← '10' (AC_out)
srcB ← '10' (R1)
R1_we ← 0
R0_we ← 0
AC_ps ← 1

AC_sign

0

1

ACC0

$Q_0 \leftarrow 1$

$Q_{res} \leftarrow 1$

srcA_sel ← '00'
srcB_sel ← '10'
ALU_op ← '001'
R0_we ← 1
R0src_sel ← '01'

Decrement count

TO DIV_START state

count=0

AC_sign

0

AC>0

1

ALU_op ← '000'
srcA ← '10' (AC_out)
srcB ← '10' (R1)
R1_we ← 0
R0_we ← 0
AC_ps ← 1

A+M state

$R0 \leftarrow Q_1$
$R1 \leftarrow AC$

END

Algorithm ends.

2.6) For normal operations 1 clock cycle

For multiplication 17 clock cycle . 16 for + 1 end op. AC add/sub and shift

For division 18 clock cycle. +1 for restore if needed.

2.7) controller has 16 states. //

```
module Controller(R0_sign,R1_sign,OVF,Z,N,E_out,clk,LOAD,COMP,CLR,Q0,ALP_op,
                      R0_we,R1_we,R0_en,R1_en,srcA_sel,srcB_sel,ALU_op,AC_lr,AC_ps,
                      Q_lr,Q_ps,R0src_sel,R1src_sel,R0_res,R1_res,AC_res,Q_res,ERR,E_in,Esrc_sel
);

                      input wire R0_sign,R1_sign,OVF,Z,N,E_out,clk,LOAD,COMP,CLR,Q0;
                      input wire [2:0] ALP_op;

                      output reg R0_we,R1_we,R0_en,R1_en,AC_lr,AC_ps,Q_lr,Q_ps,R0_res,R1_res,AC_
res,Q_res,ERR,E_in;

                      output reg [1:0] srcA_sel,srcB_sel,R0src_sel,R1src_sel,Esrc_sel;
                      output reg [2:0] ALU_op;

                      parameter SIZE = 4;
                      parameter LOADING = 4'b0000,
                                  ADD = 4'b0001,
                                  SUB = 4'b0010,
                                  AND = 4'b0011,
                                  OR = 4'b0100,
                                  EXOR = 4'b0101,
                                  BIC = 4'b0110,
                                  MUL = 4'b0111,
                                  MUL_START = 4'b1000,
                                  MUL_ASR = 4'b1001,
                                  CLEAR = 4'b1011,
                                  THE_END = 4'b1100;

                      reg [SIZE-1:0] state,next_state;

                      always@(state)
                      begin : FSM_Comb
                          next_state = LOADING;
                          case(state)
                              LOADING: begin
                                  if(LOAD==1) begin
                                      R0_we=1;
                                      R1_we=1;
                                      R1src_sel=2'b10;
                                      R0src_sel=2'b10;
                                      if(COMP==1) begin
                                          case(ALP_op)
                                              3'b000: next_state = ADD;
                                              3'b001: next_state = SUB;
                                              3'b010: next_state = MUL;
                                              3'b011: next_state = CLEAR;
                                              3'b100: next_state = AND;
                                              3'b101: next_state = OR;
                                              3'b110: next_state = EXOR;
                                              3'b111: next_state = BIC;
                                          endcase
                                      end
                                      else next_state = LOADING;
                                  end
                              end
                              ADD: begin
                                  ALU_op=3'b000;
                                  ERR=OVF;
                                  srcA_sel=2'b00;
                                  srcB_sel=2'b10;
                                  R0src_sel=2'b01;
                                  R1_res=1;
                                  next_state = THE_END;
                              end
                              SUB: begin
                                  ALU_op=3'b001;
                                  ERR=OVF;
                                  srcA_sel=2'b00;
                                  srcB_sel=2'b10;
                                  R0src_sel=2'b01;
                                  R1_res=1;
                                  next_state = THE_END;
                              end
```

```
                                AND: begin
                                    ALU_op=3'b100;
                                    srcA_sel=2'b00;
                                    srcB_sel=2'b10;
                                    R0src_sel=2'b01;
                                    R1_res=1;
                                    next_state = THE_END;
                                end
                                OR: begin
                                    ALU_op=3'b101;
                                    srcA_sel=2'b00;
                                    srcB_sel=2'b10;
                                    R0src_sel=2'b01;
                                    R1_res=1;
                                    next_state = THE_END;
                                end
                                EXOR: begin
                                    ALU_op=3'b110;
                                    srcA_sel=2'b00;
                                    srcB_sel=2'b10;
                                    R0src_sel=2'b01;
                                    R1_res=1;
                                    next_state = THE_END;
                                end
                                BIC: begin
                                    ALU_op=3'b111;
                                    srcA_sel=2'b00;
                                    srcB_sel=2'b10;
                                    R0src_sel=2'b01;
                                    R1_res=1;
                                    next_state = THE_END;
                                end
                                MUL: begin
                                    AC_res=1;
                                    E_in=0;
                                    srcA_sel=2'b00;
                                    srcB_sel=2'b11;
                                    ALU_op=3'b000;
                                    Q_ps=1;
                                    R0_we=1;
                                    R0src_sel=2'b11;
                                    Esrc_sel=2'b10;
                                    next_state=MUL_START;
                                end
                                MUL_START: begin
                                    if(Z==1) begin
                                        R0_we=1;
                                        R1_we=1;
                                        R0src_sel=2'b00;
                                        R1src_sel=2'b00;
                                        next_state = THE_END;
                                    end
                                    else begin
                                        if({Q0,E_out} == 2'b10) begin
                                            ALU_op=3'b001;
                                            srcA_sel=2'b10;
                                            srcB_sel=2'b10;
                                            R1_we=0;
                                            R0_we=0;
                                            AC_ps=1;
                                            next_state=MUL_ASR;
                                        end
                                        else if ({Q0,E_out} == 2'b01) begin
                                            ALU_op=3'b000;
                                            srcA_sel=2'b10;
                                            srcB_sel=2'b10;
                                            R1_we=0;
                                            R0_we=0;
                                            AC_ps=1;
                                            next_state=MUL_ASR;
                                        end
                                        else next_state = MUL_ASR;
                                    end
```

```
                                end
                                MUL_ASR: begin
                                    srcA_sel=2'b00;
                                    srcB_sel=2'b10;
                                    ALU_op=3'b001;
                                    R0_we=1;
                                    R0src_sel=2'b01;
                                    AC_ps=0;
                                    AC_lr=1;
                                    Q_ps=0;
                                    Q_lr=1;
                                    Esrc_sel=2'b01;
                                    next_state=MUL_START;
                                end
                                CLEAR: begin
                                    R0_res=1;
                                    R1_res=1;
                                    AC_res=1;
                                    Q_res=1;
                                    next_state=LOADING;
                                end
                                THE_END: begin
                                    if(LOAD==1) next_state=LOADING;
                                    else next_state=THE_END;
                                end
                                default: next_state = LOAD;
                            endcase
                        end

                        always@(posedge clk)
                        begin : FSM_Seq
                            if(CLR == 1) state <= CLEAR;
                            else state <= next_state;
                        end


endmodule
```