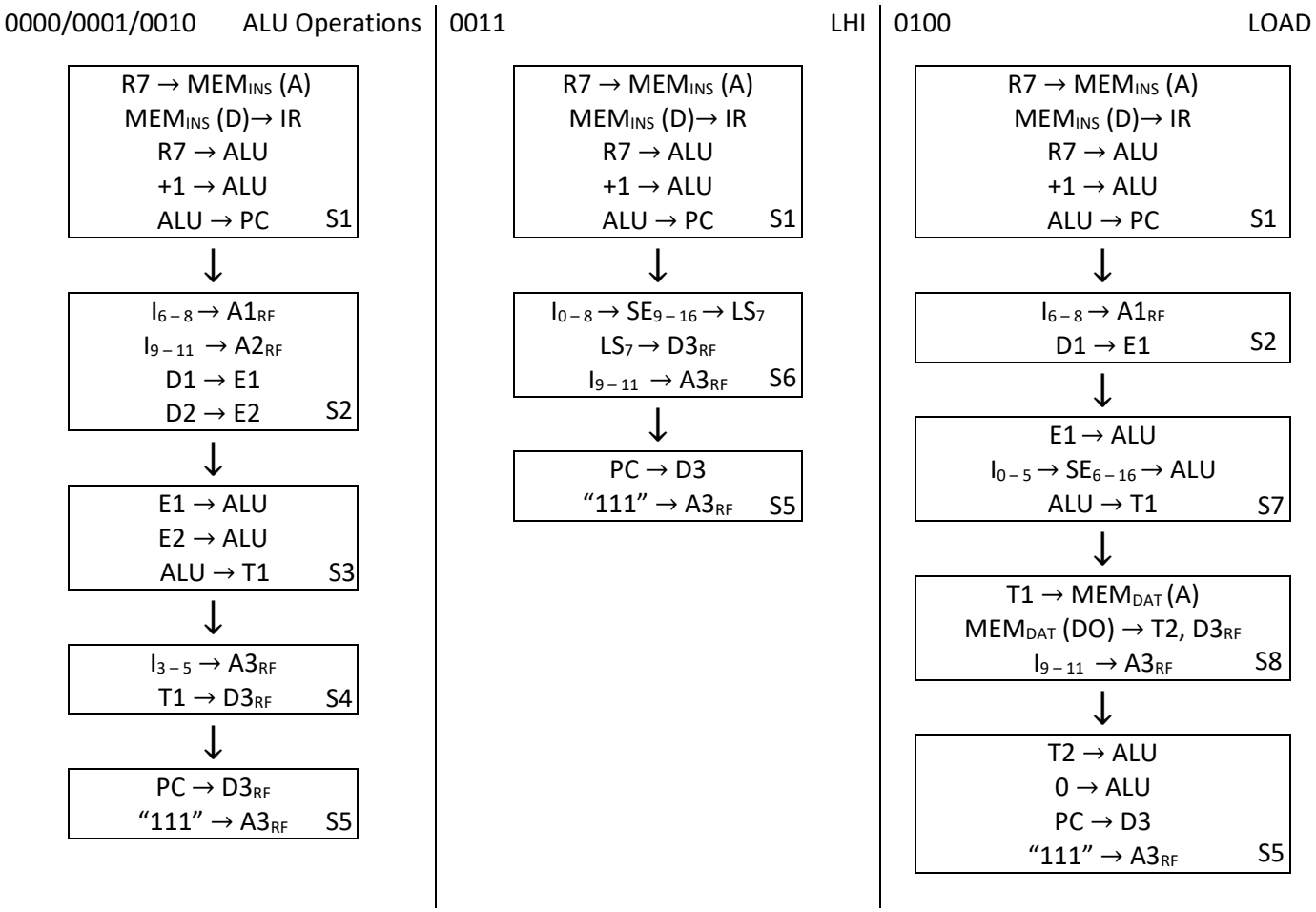
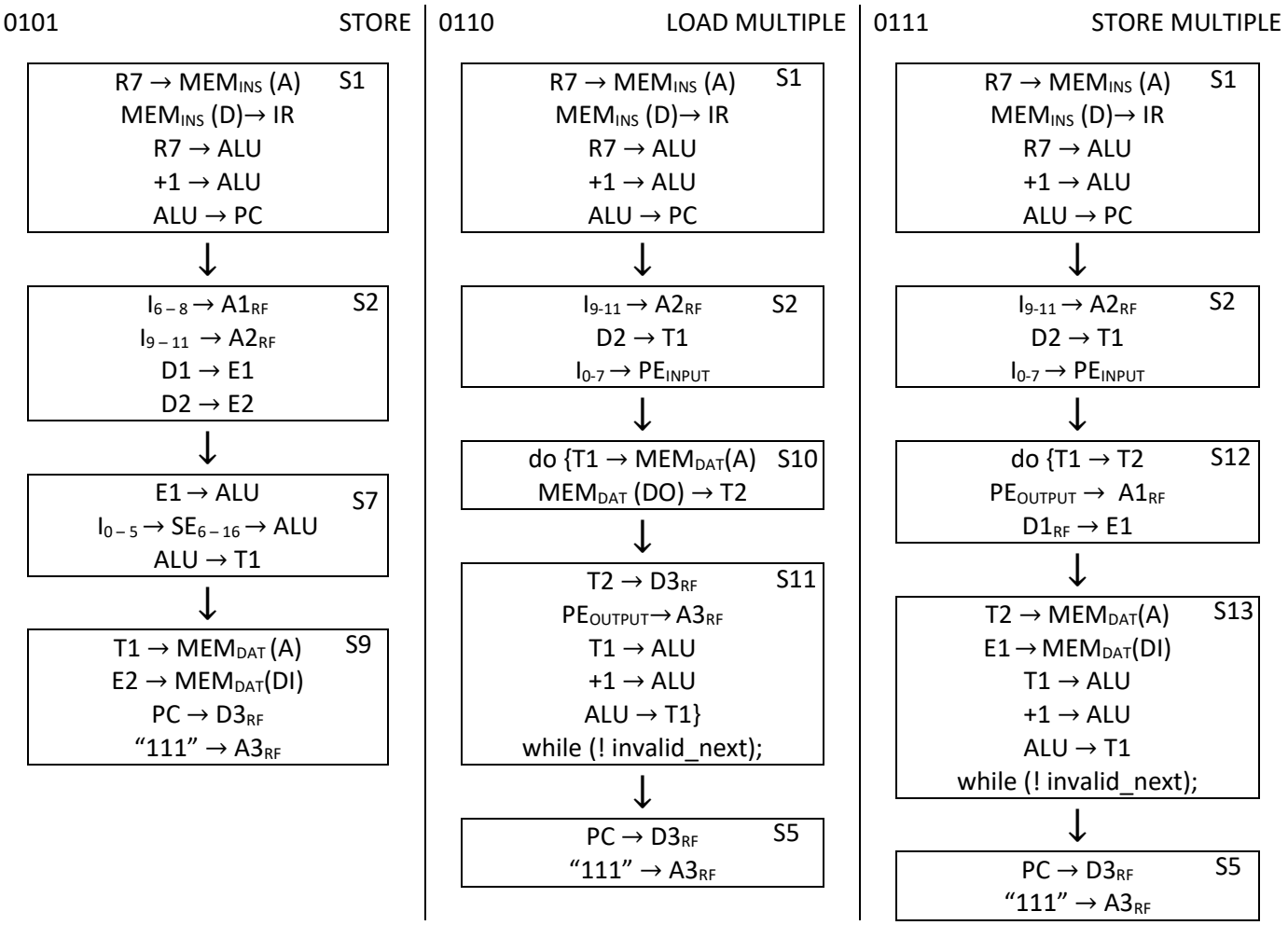
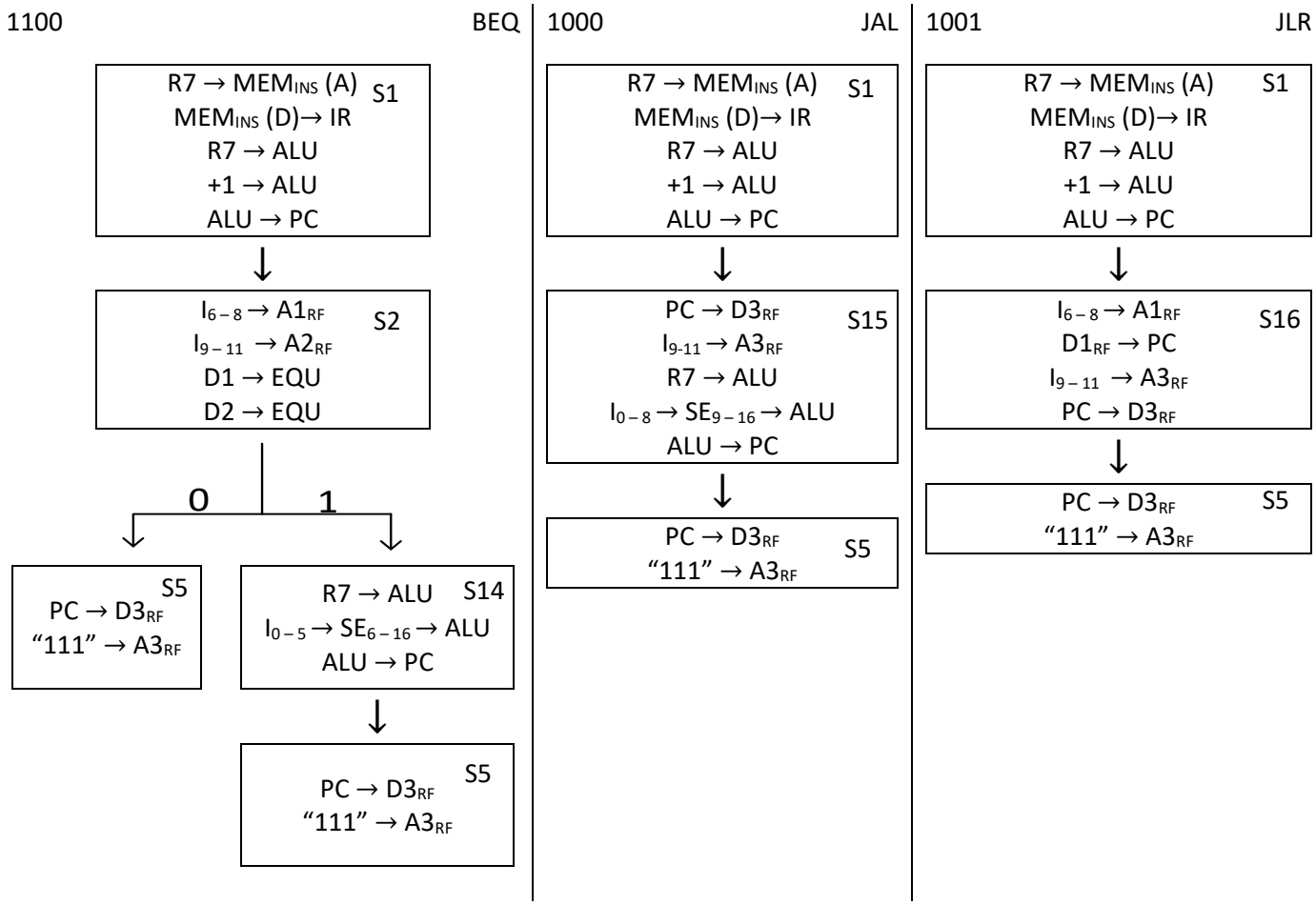
EE309 Microprocessors - Project 1
Design Document

CONTENTS

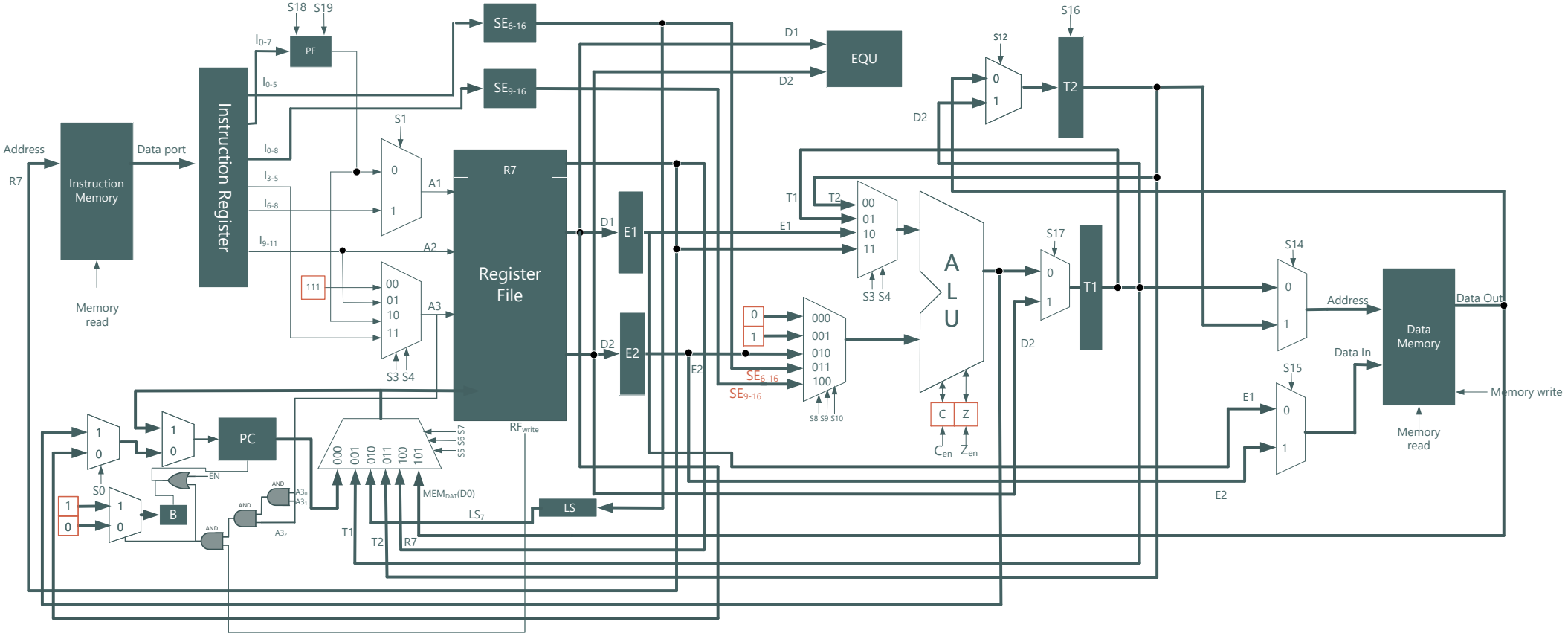
FLOWCHARTS




0101
STORE
R7 → MEMINS (A)
MEMINS (D)→ IR
R7 → ALU
+1 → ALU
ALU → PC
S1
I6 – 8 → A1RF
I9 – 11 → A2RF
D1 → E1
D2 → E2
S2
E1 → ALU
I0 – 5 → SE6 – 16 → ALU
ALU → T1
S7
T1 → MEMDAT (A)
E2 → MEMDAT(DI)
PC → D3RF
"111" → A3RF
S9
0110
LOAD MULTIPLE
R7 → MEMINS (A)
MEMINS (D)→ IR
R7 → ALU
+1 → ALU
ALU → PC
S1
I9-11 → A2RF
D2 → T1
I0-7 → PEINPUT
S2
do {T1 → MEMDAT(A)
MEMDAT (DO) → T2
S10
T2 → D3RF
PEOUTPUT→ A3RF
T1 → ALU
+1 → ALU
ALU → T1}
while (! invalid_next);
S11
PC → D3RF
"111" → A3RF
S5
0111
STORE MULTIPLE
R7 → MEMINS (A)
MEMINS (D)→ IR
R7 → ALU
+1 → ALU
ALU → PC
S1
I9-11 → A2RF
D2 → T1
I0-7 → PEINPUT
S2
do {T1 → T2
PEOUTPUT → A1RF
D1RF → E1
S12
T2 → MEMDAT(A)
E1 → MEMDAT(DI)
T1 → ALU
+1 → ALU
ALU → T1
while (! invalid_next);
S13
PC → D3RF
"111" → A3RF
S5

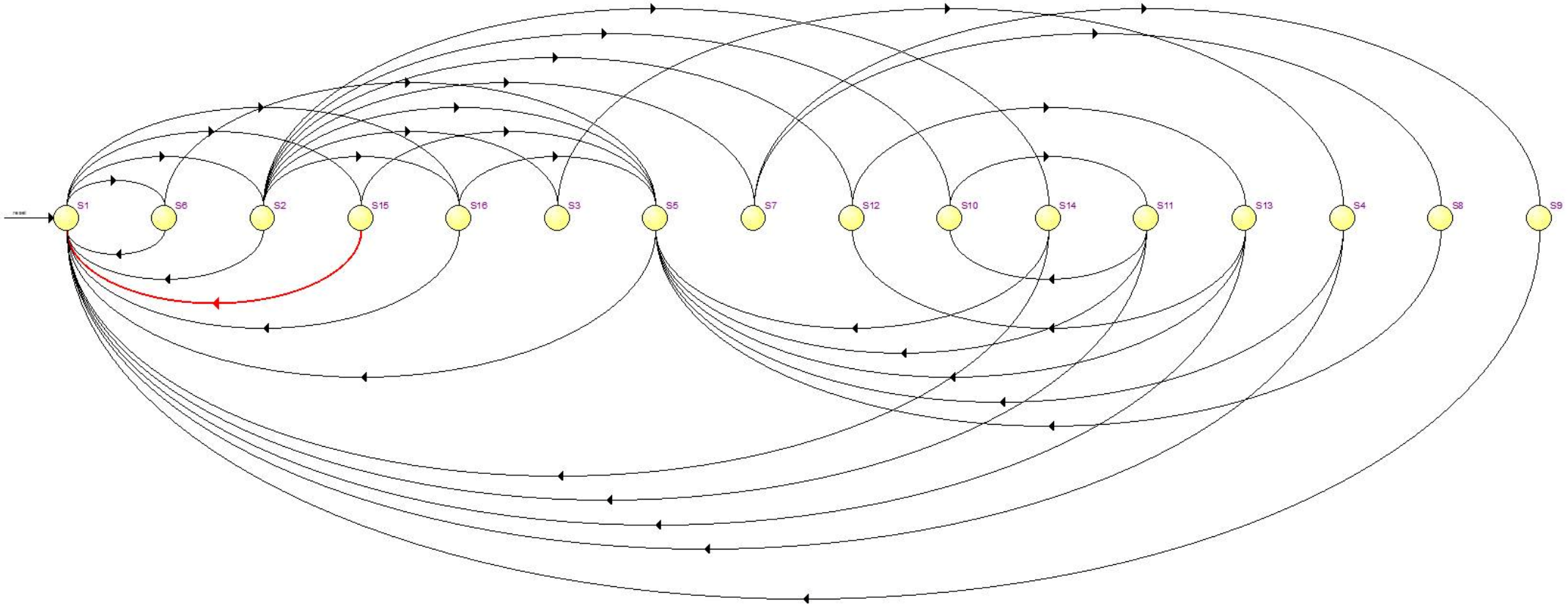

1100
BEQ
R7 → MEMINS (A)
MEMINS (D)→ IR
R7 → ALU
+1 → ALU
ALU → PC
S1
I6 – 8 → A1RF
I9 – 11 → A2RF
D1 → EQU
D2 → EQU
S2
0
1
PC → D3RF
"111" → A3RF
S5
R7 → ALU
I0 – 5 → SE6 – 16 → ALU
ALU → PC
S14
PC → D3RF
"111" → A3RF
S5
1000
JAL
R7 → MEMINS (A)
MEMINS (D)→ IR
R7 → ALU
+1 → ALU
ALU → PC
S1
PC → D3RF
I9-11 → A3RF
R7 → ALU
I0 – 8 → SE9 – 16 → ALU
ALU → PC
S15
PC → D3RF
"111" → A3RF
S5
1001
JLR
R7 → MEMINS (A)
MEMINS (D)→ IR
R7 → ALU
+1 → ALU
ALU → PC
S1
I6 – 8 → A1RF
D1RF → PC
I9 – 11 → A3RF
PC → D3RF
S16
PC → D3RF
"111" → A3RF
S5

DATAPATH DESIGN



[(hyperlinked)]

STATE TRANSITION DIAGRAM



VHDL Code for state transitions (Hyperlinked)

The logic bits that decide the next state are the 4 bits of op-code, condition[0], condition[1], C, Z , special bit B, and the invalid_next bit.

STATE TRANSITION TABLE

Current_State	Next State	Condition
S1	S16	(op_code[0]).(!op_code[1]).(!op_code[2]).(op_code[3])
S1	S15	(!op_code[0]).(!op_code[1]).(!op_code[2]).(op_code[3])
S1	S2	(!op_code[0]).(!op_code[1]).(!op_code[2]).(!op_code[3]) + (!op_code[0]).(!op_code[1]).(op_code[2]) + (!op_code[0]).(op_code[1]) + (op_code[0]).(!op_code[1]).(!op_code[2]).(!op_code[3]) + (op_code[0]).(!op_code[1]).(op_code[2]) + (op_code[0]).(op_code[1]).(!op_code[2]).(op_code[3]) + (op_code[0]).(op_code[1]).(op_code[2])
S1	S6	(op_code[0]).(op_code[1]).(!op_code[2]).(!op_code[3])
S2	S16	(op_code[0]).(!op_code[1]).(!op_code[2]).(op_code[3])
S2	S14	(!op_code[0]).(!op_code[1]).(op_code[2]).(op_code[3]).(eq)
S2	S12	(op_code[0]).(op_code[1]).(op_code[2]).(!op_code[3])
S2	S10	(!op_code[0]).(op_code[1]).(op_code[2]).(!op_code[3])
S2	S7	(!op_code[0]).(!op_code[1]).(op_code[2]).(!op_code[3]) + (op_code[0]).(!op_code[1]).(!op_code[3])
S2	S3	(!op_code[0]).(!op_code[2]).(!op_code[3]).(!condition[0]).(!condition[1]) + (!op_code[0]).(!op_code[2]).(!op_code[3]).(!condition[0]).(condition[1]).(C) + (!op_code[0]).(!op_code[2]).(!op_code[3]).(condition[0]).(!condition[1]).(Z)
S2	S5	(!op_code[0]).(!op_code[1]).(!op_code[2]).(!op_code[3]).(!condition[0]).(condition[1]).(!C) + (!op_code[0]).(!op_code[1]).(!op_code[2]).(!op_code[3]).(condition[0]).(!condition[1]).(!Z) + (!op_code[0]).(!op_code[1]).(!op_code[2]).(!op_code[3]).(condition[0]).(condition[1]) + (!op_code[0]).(!op_code[1]).(op_code[2]).(op_code[3]).(!eq) + (!op_code[0]).(op_code[1]).(!op_code[2]).(!op_code[3]).(!condition[0]).(condition[1]).(!C) + (!op_code[0]).(op_code[1]).(!op_code[2]).(!op_code[3]).(condition[0]).(!condition[1]).(!Z) + (!op_code[0]).(op_code[1]).(!op_code[2]).(!op_code[3]).(condition[0]).(condition[1])
S2	S1	(!op_code[0]).(!op_code[1]).(!op_code[2]).(op_code[3]) + (!op_code[0]).(op_code[1]).(op_code[3]) + (op_code[0]).(!op_code[1]).(op_code[2]).(op_code[3]) + (op_code[0]).(op_code[1]).(!op_code[2]) + (op_code[0]).(op_code[1]).(op_code[2]).(op_code[3])
S3	S4	Unconditional

Current_State	Next State	Condition
S4	S5	(!B)
S4	S1	(B)
S5	S1	Unconditional
S6	S5	(!B)
S6	S1	(B)
S7	S9	(op_code[0])
S7	S8	(!op_code[0])
S8	S5	Unconditional
S9	S1	Unconditional
S10	S11	Unconditional
S11	S10	(invalid_next)
S11	S5	(!B).(!invalid_next)
S11	S1	(B).(!invalid_next)
S12	S13	Unconditional
S13	S12	(invalid_next)
S13	S5	(!B).(!invalid_next)
S13	S1	(B).(!invalid_next)
S14	S5	(!B)
S14	S1	(B)
S15	S5	(!B)
S15	S1	(B)
S16	S5	(!B)
S16	S1	(B)

COMPONENTS

Register File

(Hyperlinked)

```
entity register_file is
    generic(
        word_length: integer := 16;
        num_words: integer := 8);

    port(
        data_in: in std_logic_vector(word_length-1 downto 0);
        data_out1, data_out2, R7: out std_logic_vector(word_length-1 downto 0);
        sel_in,                  sel_out1,                  sel_out2:                  in
std_logic_vector(integer(ceil(log2(real(num_words))))-1 downto 0);
        clk, wr_ena: in std_logic);

end entity;
```

Register file is an array of 8 registers with each register being 16 bit long.

Port Name	Port Type	Length	Function
data_in (D3)	Input	16	Data to be written in the register file
data_out1,data_out2	Output	16	Output port for data retrieved from the register file
R7	Output	16	Dedicated output for Register 7 (PC)
sel_in	Input	3	Address for register to be written
sel_out1, sel_out2	Input	3	Address for data to be retrieved
wr_ena	Input	1	Enable pin for writing data
clk	Input	1	Clock

Register

(Hyperlinked)

```
entity my_reg is
    generic ( data_width : integer);
    port(
        clk, ena: in std_logic;
        Din: in std_logic_vector(data_width-1 downto 0);
        Dout: out std_logic_vector(data_width-1 downto 0));
end entity;
```

Port Name	Port Type	Length	Function
Din	Input	16	Data to be written in the register
Dout	Output	16	Output of the register
clk	Output	1	Clock
ena	Input	1	Enable pin for writing data

Sign Extend

(Hyperlinked)

```
entity sign_extend is
    generic(input_width: integer := 6;
        output_width: integer := 16);
    port(
        input: in std_logic_vector(input_width-1 downto 0);
        output: out std_logic_vector(output_width-1 downto 0));
end entity;
```

Component extends the given bit string into another bit string of specified length, prefixing the required number of sign bits

Port Name	Port Type	Length	Function
input	Input	6 , 9	Data to be extended
output	Output	16	Required 16 bit string

Load_Store Multiple Hardware

(Hyperlinked)

```
entity ls_multiple is
    generic(input_width: integer := 8);
    port(
        input: in std_logic_vector(input_width-1 downto 0);
        ena, clk, set_zero: in std_logic;
        valid, invalid_next: out std_logic;
        address:  out  std_logic_vector(integer(ceil(log2(real(input_width))))-1  downto
0));
end entity;
```

The top level hardware implemented for load multiple (LM) and store multiple (SM) instructions. Outputs the address from the priority encoder, based on the input and also sets the bit at output address to 0, based on the set_zero signal.

Port Name	Port Type	Length	Function
input	Input	8	8 bit data from the instruction
address	Output	3	Output of the Priority Encoder
valid	Output	1	Bit to specify if a valid input is given
invalid_next	Output	1	Indicator bit to the FSM to indicate the penultimate valid state
set_zero	Input	1	Input from the FSM, to set the bit at the output address to 0
ena	Input	1	Enable Pin to accept a byte from instruction, comes from the FSM
clk	Input	1	Clock

Priority Encoder

(Hyperlinked)

```
component p_encoder is
        generic(input_width: integer := 16);
        port(
            input: in std_logic_vector(input_width-1 downto 0);
            output:    out    std_logic_vector(integer(ceil(log2(real(input_width))))-1
downto 0);
            valid: out std_logic);
end component;
```

Port Name	Port Type	Length	Function
input	Input	8	8 bit data from the instruction
output	Output	3	The address of the highest priority bit
valid	Output	1	Bit to specify if a valid input is given

Arithmetic Logical Unit

(Hyperlinked)

```
entity alu is
    generic(word_length: integer := 16);
    port(
        input1, input2: in std_logic_vector(word_length-1 downto 0);
        output: out std_logic_vector(word_length-1 downto 0);
        cin, sel: in std_logic;
        CY, Z: out std_logic);
end entity;
```

The ALU supports the following operations – ADD, NAND

Port Name	Port Type	Length	Function
input1,input2	Input	16	Inputs to the ALU
output	Output	16	Output of the ALU
cin	Input	1	Carry input bit to the adder
sel	Input	1	Select bit for the type of instruction
CY,Z	Output	1	Output Carry and Zero Flags

Memory

(Hyperlinked)

```
entity RAM_SIM is
    generic(
        word_length: integer := 16;
        num_words: integer := 65536);

    port(
        data_in: in std_logic_vector(word_length-1 downto 0);
        data_out : out std_logic_vector(word_length-1 downto 0);
        address: in std_logic_vector(integer(ceil(log2(real(num_words))))-1 downto 0);
        clk, wr_ena, rd_ena: in std_logic);

end entity;
```

Consists of 65536 words, with each word being 16 bit long.

Port Name	Port Type	Length	Function
data_in	Input	16	Data to be written in the memory
data_out	Input	16	Data obtained from the memory
addresss	Input	16	Address given to the memory to obtain data
wr_ena	Input	1	Enable to write the data in the memory
rd_ena	Input	1	Enable to read data from the memory

TEAM MEMBERS

1. OV Shashank
2. Pratik Brahma
3. Avineil Jain
4. Yogesh Mahajan