EE309 Microprocessors - Project 1

Design Document

Contents

[Flowcharts 1](#_Toc464768963)

[Datapath Design 3](#_Toc464768964)

[State Transition Diagram 4](#_Toc464768965)

[State Transition Table 5](#_Toc464768966)

[Components 7](#_Toc464768967)

[Register File 7](#_Toc464768968)

[Register 7](#_Toc464768969)

[Sign Extend 7](#_Toc464768970)

[Load\_Store Multiple Hardware 8](#_Toc464768971)

[Priority Encoder 8](#_Toc464768972)

[Arithmetic Logical Unit 9](#_Toc464768973)

[Memory 9](#_Toc464768974)

[Team Members 9](#_Toc464768975)

# Flowcharts

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0000/0001/0010 | ALU Operations | 0011 | LHI | 0100 | LOAD |
| |  | | --- | | R7 MEMINS (A)  MEMINS (D) IR  R7 ALU  +1 ALU  S1  ALU PC | |  | | I6 – 8  A1RF  I9 – 11  A2RF  D1 E1  S2  D2 E2 | |  | | E1 ALU  E2 ALU  S3  ALU T1 | |  | | I3 – 5 A3RF  T1 D3RF | |  | | PC D3RF  S5  “111” A3RF |   S4 | | |  | | --- | | R7 MEMINS (A)  MEMINS (D) IR  R7 ALU  +1 ALU  ALU PC | |  | | I0 – 8  SE9 – 16  LS7  LS7  D3RF  S6  I9 – 11  A3RF | |  | | PC D3  S5  “111” A3RF |   S1 | | |  | | --- | | R7 MEMINS (A)  MEMINS (D) IR  R7 ALU  +1 ALU  S1  ALU PC | |  | | I6 – 8  A1RF  S2  D1 E1 | |  | | E1 ALU  I0 – 5  SE6 – 16  ALU  ALU T1 | |  | | T1 MEMDAT (A)  MEMDAT (DO) T2, D3RF  I9 – 11  A3RF | |  | | T2 ALU  0 ALU  PC D3  “111” A3RF |   S8  S7  S5 | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0101 | STORE  S1 | 0110 | LOAD MULTIPLE  S1 | 0111 | STORE MULTIPLE  S1 |
| |  | | --- | | R7 MEMINS (A)  MEMINS (D) IR  R7 ALU  +1 ALU  ALU PC | |  | | I6 – 8  A1RF  I9 – 11  A2RF  D1 E1  D2 E2 | |  | | E1 ALU  I0 – 5  SE6 – 16  ALU  ALU T1  S9 | |  | | T1 MEMDAT (A)  E2 MEMDAT(DI)  PC D3RF  “111” A3RF |   S7  S2 | | |  | | --- | | R7 MEMINS (A)  MEMINS (D) IR  R7 ALU  +1 ALU  ALU PC  S2 | |  | | I9-11  A2RF  D2 T1  I0-7  PEINPUT | |  | | do {T1 MEMDAT(A)  MEMDAT (DO) T2  S11 | |  | | T2 D3RF  PEOUTPUT A3RF  T1 ALU  +1 ALU  ALU T1}  while (! invalid\_next);  S5 | |  | | PC D3RF  “111” A3RF |   S10 | | |  | | --- | | R7 MEMINS (A)  MEMINS (D) IR  R7 ALU  +1 ALU  ALU PC  S2 | |  | | I9-11  A2RF  D2 T1  I0-7  PEINPUT  S12 | |  | | do {T1 T2  PEOUTPUT A1RF  D1RF E1  S13 | |  | | T2 MEMDAT(A)  E1MEMDAT(DI)  T1 ALU  +1 ALU  ALU T1  while (! invalid\_next);  S5 | |  | | PC D3RF  “111” A3RF | | |

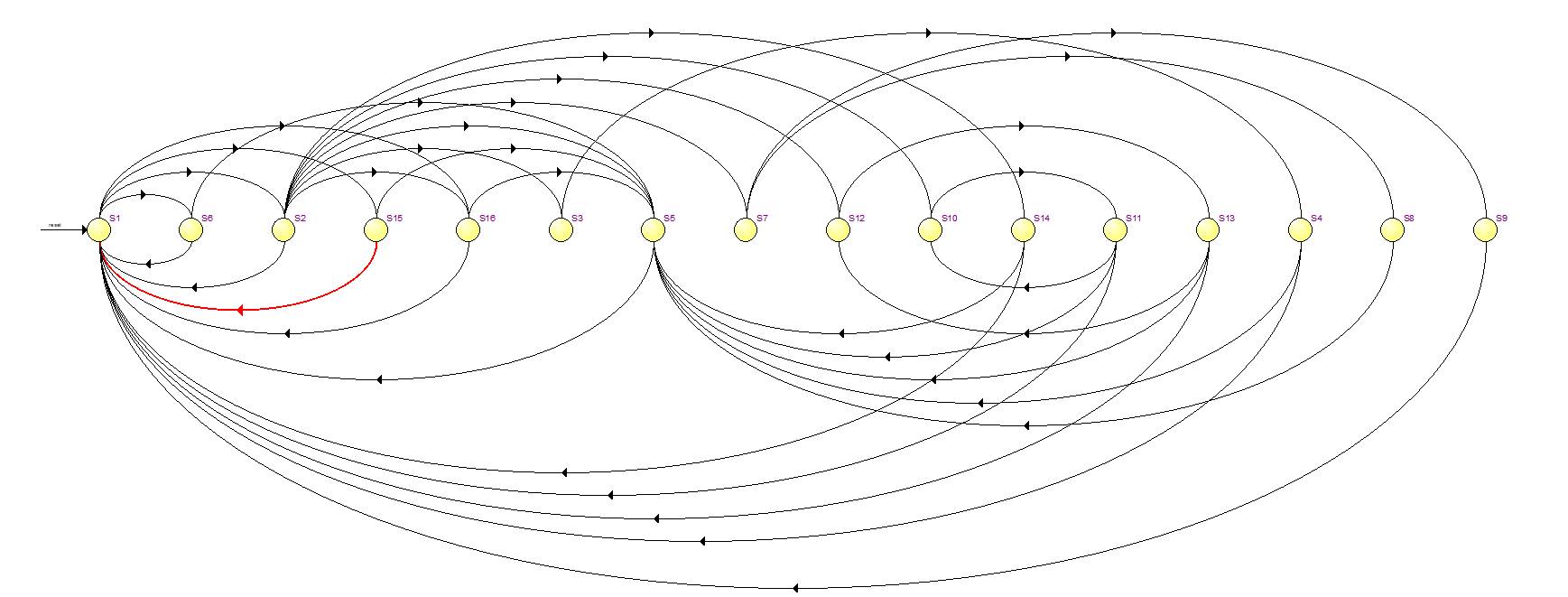
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1100 | BEQ  S1 | 1000 | JAL  S1 | 1001 | JLR  S1 |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | R7 MEMINS (A)  MEMINS (D) IR  R7 ALU  +1 ALU  ALU PC | | |  | |  | | | | | |  | I6 – 8  A1RF  S2  I9 – 11  A2RF  D1 EQU  D2 EQU | | |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

# Datapath Design



[(hyperlinked)](Data_Path.png)

# State Transition Diagram



VHDL Code for state transitions [(Hyperlinked)](VHDL_Codes/control_path.vhd)

**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 |
| 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)](VHDL_Codes/register_file.vhd)

**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)](VHDL_Codes/basic.vhd)

**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)](VHDL_Codes/sign_extend.vhd)

**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)](VHDL_Codes/ls_multiple.vhd)

**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)](VHDL_Codes/p_encoder.vhd)

**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)](VHDL_Codes/alu.vhd)

**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)](VHDL_Codes/RAM_sim.vhd)

**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