IITB-RISC

Project 1

Multi-Cycle Implementation

OV Shashank; Avineil Jain; Pratik Brahma; Yogesh Mahajan

IIT Bombay

Contents

[Microprocessor Design 2](#_Toc465807127)

[State Elaboration 2](#_Toc465807128)

[State Flow Diagram 3](#_Toc465807129)

[Datapath Design 5](#_Toc465807130)

[Instruction Set 6](#_Toc465807131)

[Instructions that affect Flag Settings 6](#_Toc465807132)

[Instruction Set and Addressing Modes 6](#_Toc465807133)

[Instruction Encoding 6](#_Toc465807134)

[Instruction Timing Reference 7](#_Toc465807135)

[Custom Instructions 7](#_Toc465807136)

[VHDL Codes 8](#_Toc465807137)

[Components 8](#_Toc465807138)

[Microprocessor Blocks 8](#_Toc465807139)

[Testing 8](#_Toc465807140)

[Assembler 8](#_Toc465807141)

[Quartus Project 8](#_Toc465807142)

# Microprocessor Design

## State Elaboration

|  |  |
| --- | --- |
| S1 | R7 MEM (A)  MEM (D) IR  R7 ALU  +1 ALU  ALU PC |
| S2 | I6 – 8  A1RF  I9 – 11  A2RF  D1 E1  D2 E2,T1  I0-7  PEINPUT |
| S3 | E1 ALU  E2 ALU  ALU T1 |
| S4 | I3 – 5 /I6 - 8 A3RF  T1 D3RF |
| S5 | PC D3RF  “111” A3RF  T2 ALU  0 ALU |
| S6 | I0 – 8  SE9 – 16  LS7  LS7  D3RF  I9 – 11  A3RF |
| S7 | E1 ALU  I0 – 5  SE6 – 16  ALU  ALU T1 , MEM(A) |
| S8 | MEM (DO) T2, D3RF  I9 – 11  A3RF |

|  |  |
| --- | --- |
| S9 | D2 MEM10(DI)  PC D3RF  “111” A3RF |
| S10 | do {  MEMDAT (DO) T2} |
| S11 | T2 D3RF  PEOUTPUT A3RF  T1 ALU  +1 ALU  ALU T1,MEM(DI)}  while (! invalid\_next); |
| S12 | PEOUTPUT A2RF  T1 MEM(A) |
| S13 | T1 ALU  +1 ALU  ALU T1  while (! invalid\_next); |
| S14 | R7 ALU  I0 – 5  SE6 – 16  ALU  ALU PC |
| S15 | PC D3RF  I9-11 A3RF  R7 ALU  I0 – 8  SE9 – 16  ALU  ALU PC |
| S16 | D1RF PC  I9 – 11  A3RF  PC D3RF |

## State Flow Diagram

|  |  |  |
| --- | --- | --- |
| **ADD/ADC/ADZ/NDU/NDC**  **/NDZ/ADO/NDO** | **ADI** | **LHI** |
|  |  |  |
| **LW** | **SW** | **LM** |
|  |  |  |

|  |  |
| --- | --- |
| SM | BEQ |
|  |  |
| JAL | JLR |
|  |  |

# Datapath Design



# Instruction Set

## Instructions that affect Flag Settings

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Flags** | | | **Instruction** | **Flags** | | |
| **C** | **Z** | **OV** | **C** | **Z** | **OV** |
| ADD | X | X | X | NDU |  | X |  |
| ADC | X | X | X | NDC |  | X |  |
| ADZ | X | X | X | NDZ |  | X |  |
| ADO | X | X | X | NDO |  | X |  |
| ADI | X | X | X | LW |  | X |  |

## Instruction Set and Addressing Modes

|  |  |
| --- | --- |
| Rn | Registers R6-R0 of the Register Bank |
| data 6 | Signed 6-bit constant included in instruction |
| data 9 | Signed 9-bit constant included in instruction |
| rel 6 | Signed (two’s complement) 6-bit offset byte. Used by BEQ. Range is -32 to +31 short words relative to the present instruction |
| rel 9 | Signed (two’s complement) 9-bit offset byte. Used by JAL. Range is -256 to +255 short words relative to the present instruction |
| reg | 8-bits representing registers R7-R0. Used by LM and SM |
| R7 | Program Counter |
| R | Registers R7-R0 of the Register Bank |

## Instruction Encoding

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ADD: | 00\_00 | RA | RB | RC | 0 | 00 |
| ADC: | 00\_00 | RA | RB | RC | 0 | 10 |
| ADZ: | 00\_00 | RA | RB | RC | 0 | 01 |
| ADO: | 00\_00 | RA | RB | RC | 0 | 11 |
| ADI: | 00\_01 | RA | RB | data 6 | | |
| NDU: | 00\_10 | RA | RB | RC | 0 | 00 |
| NDC: | 00\_10 | RA | RB | RC | 0 | 10 |
| NDZ: | 00\_10 | RA | RB | RC | 0 | 01 |
| NDO: | 00\_10 | RA | RB | RC | 0 | 11 |
| LHI: | 00\_11 | RA | data 9 | | | |
| LW: | 01\_00 | RA | RB | rel 6 | | |
| SW: | 01\_01 | RA | RB | rel 6 | | |
| LM: | 01\_10 | RA | reg | | | |
| SM: | 01\_11 | RA | reg | | | |
| LLI: | 10\_11 | RA | data 9 | | | |
| BEQ: | 11\_00 | RA | RB | rel 6 | | |
| JAL: | 10\_00 | RA | rel 9 | | | |
| JLR: | 10\_01 | RA | RB | 000\_000 | | |

## Instruction Timing Reference

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Mnemonic | | Flags | | | Oscillator Period |  | Mnemonic | | Flags/  Condition | Oscillator Period |
| C | Z | OV |
| ADD | Rn, R, R | X | X | X | 5 | ADI | Rn, R, data6 | X | 5 |
| ADD | R7, R, R | X | X | X | 4 | ADI | R7, R, data6 | X | 4 |
| ADC | Rn, R, R | 0 | X | X | 5 | LHI | Rn, data9 | X | 3 |
| ADC | R7, R, R | 0 | X | X | 4 | LHI | R7, data9 | X | 2 |
| ADC | Rn, R, R | 1 | X | X | 3 | LLI | Rn, data9 | X | 3 |
| ADZ | Rn, R, R | X | 0 | X | 5 | LLI | R7, data9 | X | 2 |
| ADZ | R7, R, R | X | 0 | X | 4 | LW | R, R, rel6 | X | 5 |
| ADZ | Rn, R, R | X | 1 | X | 3 | SW | R, R, rel6 | X | 4 |
| ADO | Rn, R, R | X | X | 0 | 5 | LM | R, reg | X | 3+2\*reg\_count |
| ADO | R7, R, R | X | X | 0 | 4 | SM | R, reg | X | 3+2\*reg\_count |
| ADO | Rn, R, R | X | X | 1 | 3 | JAL | Rn, rel9 | X | 3 |
| NDU | Rn, R, R | X | X | X | 5 | JAL | R7, rel9 | X | 2 |
| NDU | Rn, R, R | X | X | X | 4 | JLR | Rn, R | X | 3 |
| NDC | Rn, R, R | 0 | X | X | 5 | JLR | R7, R | X | 2 |
| NDC | R7, R, R | 0 | X | X | 4 | BEQ |  | Not Equal | 3 |
| NDC | Rn, R, R | 1 | X | X | 3 | BEQ |  | Equal | 4 |
| NDZ | Rn, R, R | X | 0 | X | 5 |  |  |  |  |
| NDZ | R7, R, R | X | 0 | X | 4 |  |  |  |  |
| NDZ | Rn, R, R | X | 1 | X | 3 |  |  |  |  |
| NDO | Rn, R, R | X | X | 0 | 5 |  |  |  |  |
| NDO | R7, R, R | X | X | 0 | 4 |  |  |  |  |
| NDO | Rn, R, R | X | X | 1 | 3 |  |  |  |  |

## Custom Instructions

* ADO: ADD if the overflow flag is set. The rest of the instruction format is the same as the other conditional execution instructions
* NDO: NAND if the overflow flag is set. The rest of the instruction format is the same as the other conditional execution instructions
* LLI: Load lower immediate. Load the immediate 9 bits into the register as mentioned in the instruction without sign extension by padding zeros in the upper 7 bits.

NOTE: The microprocessor is supposed to be a signed microprocessor and hence the overflow bit was introduced. It may be considered as the signed equivalent of the carry bit. The carry bit here still represents the unsigned version and hence the two must be used carefully.

# VHDL Codes

(All the codes have been hyperlinked here for easy viewing; Notepad++ or gedit is recommended)

## Components

* [ALU](Codes/Components/alu.vhd)
* [Priority Encoder](Codes/Components/p_encoder.vhd)
* [Load/Store Multiple Logic Block](Codes/Components/ls_multiple.vhd)
* [Registers](Codes/Components/basic.vhd)
* [Memory](Codes/Components/ram.vhd) (Generated using Altera’s MegaWizard)
* [Register File](Codes/Components/register_file.vhd)
* [Sign Extender](Codes/Components/sign_extend.vhd)

## Microprocessor Blocks

* [Data Path](Codes/IITB_RISC/data_path.vhd): This consists of the entire data path along with all the transfers and the predicates corresponding to an RTL layout of the microprocessor. All the T and S signals are pretty accurately detailed as comments at the beginning of the architecture.
* [Control Path](Codes/IITB_RISC/control_path.vhd): This consists of the controller Moore FSM. It has been decomposed into three processes; The first one describes the flip flops which control the states, the second one describes the next state logic and finally the third process controls the output logic based on the present state. (The order in the code might not exactly follow this order). All the T signals have been accompanied by the expected result they are to cause in the data path.
* [IITB\_RISC](Codes/IITB_RISC/IITB_RISC.vhd): The top-level entity than combines the data path and the controller FSM. The register 0 has been shown as an output so that the processes of the microprocessor can be displayed outside in hardware.

## Testing

* [Test\_final](Codes/IITB_RISC/test_final.vhd): This is a test bench which can be used to simulate the entire microprocessor in Altera’s modelsim. It basically functions to provide the clock and reset signals to the microprocessor.

Testing results have been stored as [images](Screenshots.zip) and are also available along with this document.

## Assembler

The code [assembler.py](Codes/Assembler/assembler.py) was developed as a combined effort of Kalpesh and Shashank. It basically works by taking as its input a text file containing code written in an assembly like language, and creates an INTEL HEX file which can be directly provided to Quartus which uses it to preload memory. A [sample hex](Codes/Assembler/mem.hex) file produced by the code is provided along with this document. Also, a [sample text](Codes/Assembler/fpga_test.txt) input file has also been included.

## Quartus Project

The [quartus project folder](IITB_RISC_Project.zip) used for testing is also included along with this submission. This can be used for quick compilation and viewing in Quartus. Necessary settings have been implemented to allow for timing analysis and RTL and gate level simulation