Department of Electronics and Telecommunication Engineering

University of Moratuwa

EN2031 Fundamentals of Computer Organization and Design
Processor Design

Final Submission

Gammune D.J.T.	210179R
Vidumini H.M.V.	210669U
Wathudura T.R.	210682D

Contents

1 Part A: Instruction Set Architecture (ISA) 2
- 1.1 Instruction Set . . . 2
  - 1.1.1 ALU Instructions . . . 2
  - 1.1.2 Data Transfer Instructions . . . 2
  - 1.1.3 Stack Operations . . . 2
  - 1.1.4 Load Operations . . . 2
  - 1.1.5 Store Operation . . . 2
  - 1.1.6 Load Immediate Operations . . . 2
  - 1.1.7 Branching Instructions . . . 2
- 1.2 Instructions Format . . . 2
  - 1.2.1 R Type (Register Type) . . . 2
  - 1.2.2 I Type (Immediate Type) . . . 2
  - 1.2.3 Load Type . . . 2
  - 1.2.4 Store Type . . . 2

2 Part B: Opcode Encoding 2
- 2.1 R Type (Register Type) . . . 3
- 2.2 I Type (Immediate Type) . . . 3
  - 2.2.1 IL Type (Immediate Load Type) . . . 3
  - 2.2.2 IJ (Branch Operation) . . . 3
- 2.3 Load Type . . . 3
- 2.4 Store Type . . . 3

3 Part C: Datapath Diagram 3

4 Part D: Microarchitecture Design and Controller 3
- 4.1 Microarchitecture Design Approach . . . 3
- 4.2 Controller Design . . . 4
  - 4.2.1 Controller Overview . . . 4
  - 4.2.2 Controller Inputs . . . 4
  - 4.2.3 Controller Outputs . . . 4
  - 4.2.4 Control Structures . . . 4
  - 4.2.5 Control Signal Diagram . . . 5

List of Tables

1 ALU Instructions . . . 2
2 Data Transfer Instructions . . . 2
3 Stack Operations . . . 2
4 Load Operations . . . 2
5 Store Operation . . . 2
6 Load Immediate Operations . . . 2
7 Branching Instructions . . . 2
8 R Type Opcode Encoding . . . 3
9 IL Type Opcode Encoding . . . 3
10 IJ Type Opcode Encoding . . . 3
11 Load Type Opcode Encoding . . . 3
12 Store Type Opcode Encoding . . . 3
13 ALU Control Signals . . . 4
14 Branch Control Signals . . . 4
15 MemtoReg Controls . . . 4
16 ALUSrc Controls . . . 4
17 Control Signals . . . 5

List of Figures

1 Datapath Diagram . . . 3

1 Part A: Instruction Set Architecture (ISA)

1.1 Instruction Set

1.1.1 ALU Instructions

Opcode	Operands	Operation
ADD	RA, RB	$RA \leftarrow RA + RB$
SUB	RA, RB	$RA \leftarrow RA - RB$
OR	RA, RB	$RA \leftarrow RA \vee RB$
AND	RA, RB	$RA \leftarrow RA \& RB$

Table 1: ALU Instructions

1.1.2 Data Transfer Instructions

Opcode	Operands	Operation
MOV	RA, RB	Move the value in RB to RA.

Table 2: Data Transfer Instructions

1.1.3 Stack Operations

Opcode	Operands	Operation
POP	RA	Pop the value from the top of the stack and store it in RA.
PUSH	RB	Push the value in RB to the top of the stack.

Table 3: Stack Operations

1.1.4 Load Operations

Opcode	Operands	Operation
LOAD	RA, Ra	Load the value from the memory address in Ra to RA. $RA \leftarrow DM[Ra]$
PLOAD	RA, Ra	Load the value from the memory address in Ra to RA and increment RB by 1. $RA \leftarrow DM[Ra], Ra \leftarrow Ra + 1$
ILDR	RA, RB	Load the value from the memory address specified in RB to RA $RA \leftarrow DM[RB]$
ILDRP	RA, Ra	Load the value from the memory address specified in RB to RA, and then increment the value in RB by 1 $RA \leftarrow DM[RB], RB \leftarrow RB + 1$

Table 4: Load Operations

1.1.5 Store Operation

Opcode	Operands	Operation
STORE	RA, RB	Store the value in RB in the memory address specified by register RA. RA can be any register from R0 to R6. $DM[RA] \leftarrow RB$

Table 5: Store Operation

1.1.6 Load Immediate Operations

Opcode	Operands	Operation
ILD	RA, imm[7:0]	Load 8-bit immediate value to RA.
UILD	RA, imm[15:8]	OR 8-bit immediate value with upper 8 bits of RA.

Table 6: Load Immediate Operations

1.1.7 Branching Instructions

Opcode	Operands	Operation
JMP	imm[7:0]	Update PC relative to the current PC value to a given offset.
JEQ	RB, imm[7:0]	Compare values in R0 and RA. If equal, update PC relative to the current PC value to a given offset.
JLT	RB, imm[7:0]	Compare values in R0 and RA. If less than or equal, update PC relative to the current PC value to a given offset.
JGT	RB, imm[7:0]	Compare values in R0 and RA. If greater than or equal, update PC relative to the current PC value to a given offset.

Table 7: Branching Instructions

1.2 Instructions Format

1.2.1 R Type (Register Type)

15:11	10:8	7:5	4:0	Instruction Format
Function	RA	RB	Opcode	Opcode (RA, RB)

1.2.2 I Type (Immediate Type)

15:8	7:5	4:0	Instruction Format
Immediate Operand	RD	Opcode	Opcode (RA, imm)

1.2.3 Load Type

15:8	7:5	4:0	Instruction Format
-	RA	Opcode	LOAD RA, RB or PLOAD RA, RB

1.2.4 Store Type

15:11	10:8	7:5	4:0	Instruction Format
Function	RA	RB	Opcode	STORE RA, RB

2 Part B: Opcode Encoding

- An opcode consists of 5 bits.
- The first two bits decide the type of the instruction:
  - R Type - 00
  - I Type - 01
  - Load Type - 10
  - Store Type - 11
- The remaining 3 bits of the opcode determine the specific instruction.

2.1 R Type (Register Type)

R-type instructions include ADD, SUB, OR, AND, and MOV. The remaining 3 bits of the opcode directly control the ALU functions based on the operation.

Instruction	Opcode
ADD	00000
SUB	00001
OR	00010
AND	00011
MOV	00110

Table 8: R Type Opcode Encoding

2.2 I Type (Immediate Type)

The I-type instruction category comprises IL (Immediate Operand Load) and IJ (Branch Operation).

2.2.1 IL Type (Immediate Load Type)

Instruction	Opcode
ILD	01000
UILD	01011

Table 9: IL Type Opcode Encoding

2.2.2 IJ (Branch Operation)

Instruction	Opcode
JMP	01100
JEQ	01101
JLT	01110
JGT	01111

Table 10: IJ Type Opcode Encoding

In jump instructions, the last 3 bits of the opcode determine the branch control signals.

- For both the third bit of the opcode indicates the instruction type:
  - IL (Immediate Operand Load): 0
  - IJ (Branch Operation): 1

2.3 Load Type

The LOAD-type instructions include POP, LOAD, PLOAD, ILDR, and ILDRP. Similar to R-type instructions, the remaining 3 bits in the opcode differentiate these instructions.

Instruction	Opcode
POP	10001
LOAD	10010
PLOAD	10011
ILDR	10100
ILDRP	10101

Table 11: Load Type Opcode Encoding

2.4 Store Type

The LOAD-type instructions include POP, LOAD, and PLOAD. Similar to R-type instructions, the remaining 3 bits in the opcode differentiate these instructions.

Instruction	Opcode
STORE	11001
PUSH	11010

Table 12: Store Type Opcode Encoding

3 Part C: Datapath Diagram

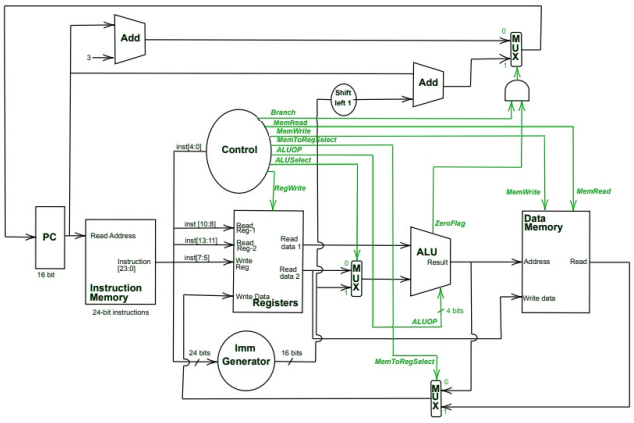

Figure 1: Datapath Diagram

4 Part D: Microarchitecture Design and Controller

4.1 Microarchitecture Design Approach

For our processor, we have opted for the **hardwired microarchitecture design approach**. This decision is justified as follows:

- **Simplicity**: Hardwired microarchitecture is relatively simple to design and comprehend, which suits the scope and requirements of our processor.
- **Efficiency**: Hardwired microarchitecture generally offers faster execution and efficiency

since control signals are generated using combinational logic.

- **Real-time Operation**: The hardwired approach is well-suited for real-time applications, which is essential for our processor's intended use.
- **Ease of Implementation**: With a well-defined and limited instruction set, a hardwired approach is easier to implement and debug.

4.2 Controller Design

4.2.1 Controller Overview

The controller plays a vital role in the datapath, responsible for generating control signals to coordinate various components. Here, we present a simplified example of the controller design for illustration. A complete controller would be more complex and tailored to the specific datapath.

4.2.2 Controller Inputs

The controller takes the following inputs:

- **Opcode (5 bits)**: The opcode from the instruction to determine the operation type.
- **Function (3 bits)**: Additional bits for specific operations.
- **RA (3 bits)**: Register A identifier.
- **RB (3 bits)**: Register B identifier.
- **Imm (8 bits)**: Immediate value for load-type instructions.

4.2.3 Controller Outputs

The controller generates various control signals, including:

- **ALU Control Signals**: Signals such as ALU_Add, ALU_Subtract, ALU_OR, ALU_AND, ALU_MOV to control the ALU's operation based on the opcode.
- **Register Write Control**: Signals like RegWrite_RA, RegWrite_RB to enable writing to registers RA and RB.
- **Memory Control Signals**: If the processor has a memory component, signals for memory access.
- **Jump Control Signals**: Signals like Jump_JMP, Jump_JEQ, Jump_JLT, Jump_JGT to control branching.
- **Stack Control Signals**: If stack operations are supported, signals for pushing and popping from the stack.

4.2.4 Control Structures

ALU Control Signals

Instruction	Opcode
ADD	000
SUB	001
OR	010
AND	011
Pass B	110

Table 13: ALU Control Signals

Branch Control Signals

Instruction	Opcode
Don't branch	000
Unconditional	100
Jump if equal	101
Jump if greater than	111
Jump if less than	110

Table 14: Branch Control Signals

MemtoReg Signals

Decides what will be written to the write register.

Instruction	Opcode
ALU output	0
Data Read From	1

Table 15: MemtoReg Controls

ALUSrc Control

This control decides what is the second input to the ALU

Instruction	Opcode
Register B	00
Immediate	01
RO	10

Table 16: ALUSrc Controls

4.2.5 Control Signal Diagram

Control signal	**MemRead**	**MemtoReg**	**ALUOP**	**MemWrite**	**ALUSrc**	**RegDataWrite**	**RegAddrWrite**	**Branch**	**AddrALUOP**	**AddrSel**	**Upper Sel**
Instruction											
ADD	0	0	000	0	00	1	0	000	0	0	0
SUB	0	0	001	0	00	1	0	000	0	0	0
OR	0	0	010	0	00	1	0	000	0	0	0
AND	0	0	011	0	00	1	0	000	0	0	0
MOV	0	0	100	0	00	1	0	000	0	0	0
LOAD	1	1	000	0	00	1	0	000	0	0	0
PLOAD	1	1	000	0	00	1	1	000	1	0	0
POP	1	1	000	0	00	1	1	000	0	0	0
STORE	0	0	000	1	00	0	0	000	0	0	0
PUSH	0	0	000	1	00	0	1	000	1	1	0
ILD	0	0	100	0	01	1	0	000	0	0	0
UILD	0	0	010	0	01	1	0	000	0	0	1
JMP	0	0	000	0	00	0	0	100	0	0	0
JEQ	0	0	001	0	10	0	0	101	0	0	0
JLT	0	0	001	0	10	0	0	110	0	0	0
JGT	0	0	001	0	10	0	0	111	0	0	0

Table 17: Control Signals