A REPORT ON

DESIGN AND TESTING OF 16- BIT MULTICYCLE RISC PROCESSOR

Pertaining to the Area

COMPUTER ARCHITECTURE

Prepared by

Ansh Shah (2018A3PS0294P)

Abhinandan Sharma (2018A3PS0095P)

Tanvi Shewale (2018A3PS0298P)

Submitted to

Mr. Karri Babu Ravi Teja

In the partial fulfilment of the course

Computer Architecture

(CS F342)

Date: 20th April 2021

MULTICYCLE PROCESSOR DESIGN

The project aims at the implementation and testing of a 16-bit, multicycle RISC processor. Each instruction is divided into a number of cycles from the following.
IF: Instruction Fetch
ID: Instruction Decode
EX: Execute
MEM: Memory Operation
WB: Write Back

Each instruction is 16-bit long, with varying formats. Based on the arrangement and value of opcode, source registers, destination registers, immediate values and function fields, we divided the set of instructions into the following types. After the completion of ID cycle, the control unit FSM can branch into one of the 6 stages based on these instructions

REG type: includes add, subtract and logical instructions

4 bits	4 bits	4 bits	4 bits

SHIFT type: includes shift instructions

4 bits	4 bits	4 bits	4 bits

IMM (zero) type: includes add, subtract and logical instructions

4 bits	4 bits	8 bits

IMM (MSB) type: includes add and subtract instructions

4 bits	4 bits	8 bits

JMP type: includes the jump instruction

4 bits	12 bits

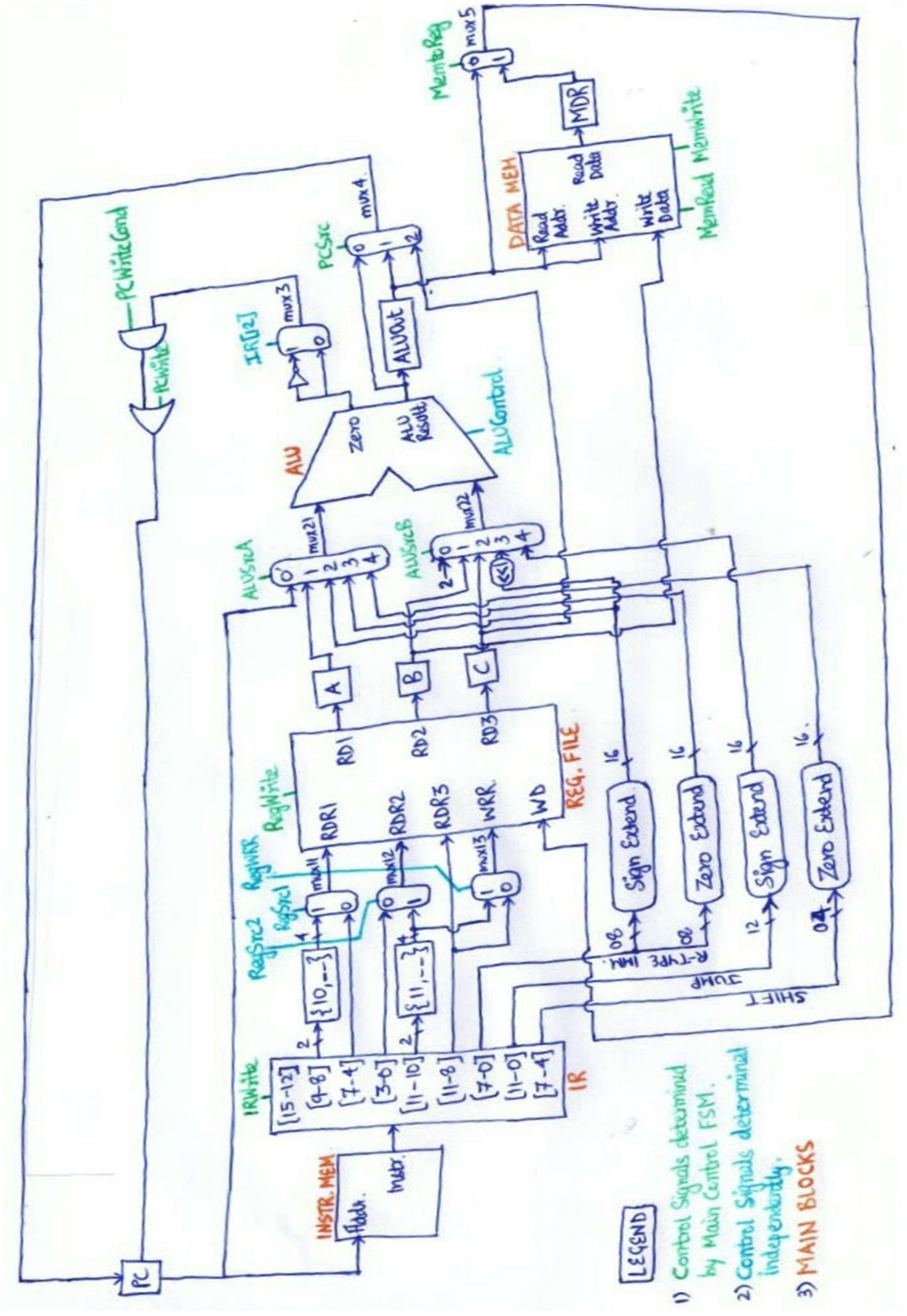
LW/SW type: includes load and store instructions

4 bits	2 bits	2 bits	8 bits

BRANCH type: includes BE and BNE instructions

4 bits	4 bits	4 bits	4 bits

DATAPATH:


PC
INSTR. MEM
Addr.
Instr.
IRWrite
[15-12]
[9-8]
[7-4]
[3-0]
[11-10]
[11-8]
[7-0]
[11-0]
[7-4]
IR
{10,--}
{11,--}
RegSrc2
RegSrc1
RegWRR
mux11
mux12
mux13
RegWrite
RDR1
RDR2
RDR3
WRR
WD
RD1
RD2
RD3
REG. FILE
A
B
C
Sign Extend
Zero Extend
Sign Extend
Zero Extend
16
08
12
04
IMM.
R-TYPE
JUMP
SHIFT
ALUSrcA
mux21
ALUSrcB
mux22
<<1
ALU
Zero
ALU Result
ALUControl
IR[12]
mux3
ALUOut
PCWrite
PCWriteCond
PCSrc
mux4
DATA MEM
Read Addr.
Write Addr.
Write Data
Read Data
MemRead
MemWrite
MDR
MemtoReg
mux5
LEGEND
1) Control Signals determined by Main Control FSM.
2) Control Signals determined independently.
3) MAIN BLOCKS

Main Module:
module processor (PC_init, clk, rst, error_log_1, error_log_2, error_log_3);

PC:
PC is a register that stores the address from where the next instruction is to be fetched.

Instruction Memory:
Instruction memory is a 64kB block and is byte organised. The instructions for the initialization of the memory are present in a data file "**inst_mem.dat**". The user can give the initial PC value of his/her choice from where the program will start executing, by giving a posedge to the reset signal. On every positive edge of reset, the value of PC becomes PC_init.

IR:
Wherever control signal IR_write is high, IR register is populated with the instruction.

Register File:
Read and write registers for the execution of the instruction are decoded based on the opcode from the appropriate locations in the instruction. There are 3 read register ports (RDR1, RDR2, RDR3) and one write register port (WRR). The data from the read register ports is stored in 3 internal registers A, B, C. There is also a write data (WD) port that will hold the data that is to be written in the register whose address is stored in WRR. This writing of data takes place only when the Reg_write signal is high. The 16 registers (each 16- bits) are initialized to zero each. Their modified contents can be verified using the data file "**reg_file.dat**". The number of read register ports has been increased (from 2 to 3) to reduce the number of states in the control unit FSM and simplify the design.

ALU:
The ALU performs addition, subtraction, NAND, OR and shift operations based on the input from ALU control. The ALU control unit is designed separately to simplify the main control unit FSM. The ALU control unit generates a 3-bit ALU_ctrl signal whose value is determined by the current stage (from the main control unit) and the opcode and function field (from the IR). The output of the ALU is stored in the ALU_out register. Whenever the ALU_result is zero, the "zero" port of ALU is made high.

Operation	**ALU_ctrl**
Add	000
Sub	001
NAND	010
OR	011
Shift left	100
Shift right logical	101
Shift right arithmetic	110

Data Memory:

Data memory is 64kB block which is Byte organised. The user can initialize and check the values using the data file "**data_mem.dat**". The address of the data to be read is provided at read_addr port and the address where the data is to be written is provided at write_addr port. The data to be written is provided at the write_data port. The read and write operations for the memory are performed according to the Mem_read and Mem_write signals. The data read from the memory is stored in MDR.

Multiplexers:
Mux 11, Mux 12, Mux13:
These are used to determine the data to be provided at RDR1, RDR2 and WRR. The control signals (Reg_src1, Reg_src2, Reg_wrr) are solely decided on the basis of opcode and are independent of the main control unit.

Mux 21, Mux 22:
These are used to determine the inputs to the ALU based on the control signals (ALU_src_A and ALU_src_B) generated by the main control unit.

Mux 3:
Based on the value of IR[12] (LSB of opcode), either value of zero port is passed or the negation of zero port is passed. To enable writing into the PC register, the signal at enable pin is as follows:

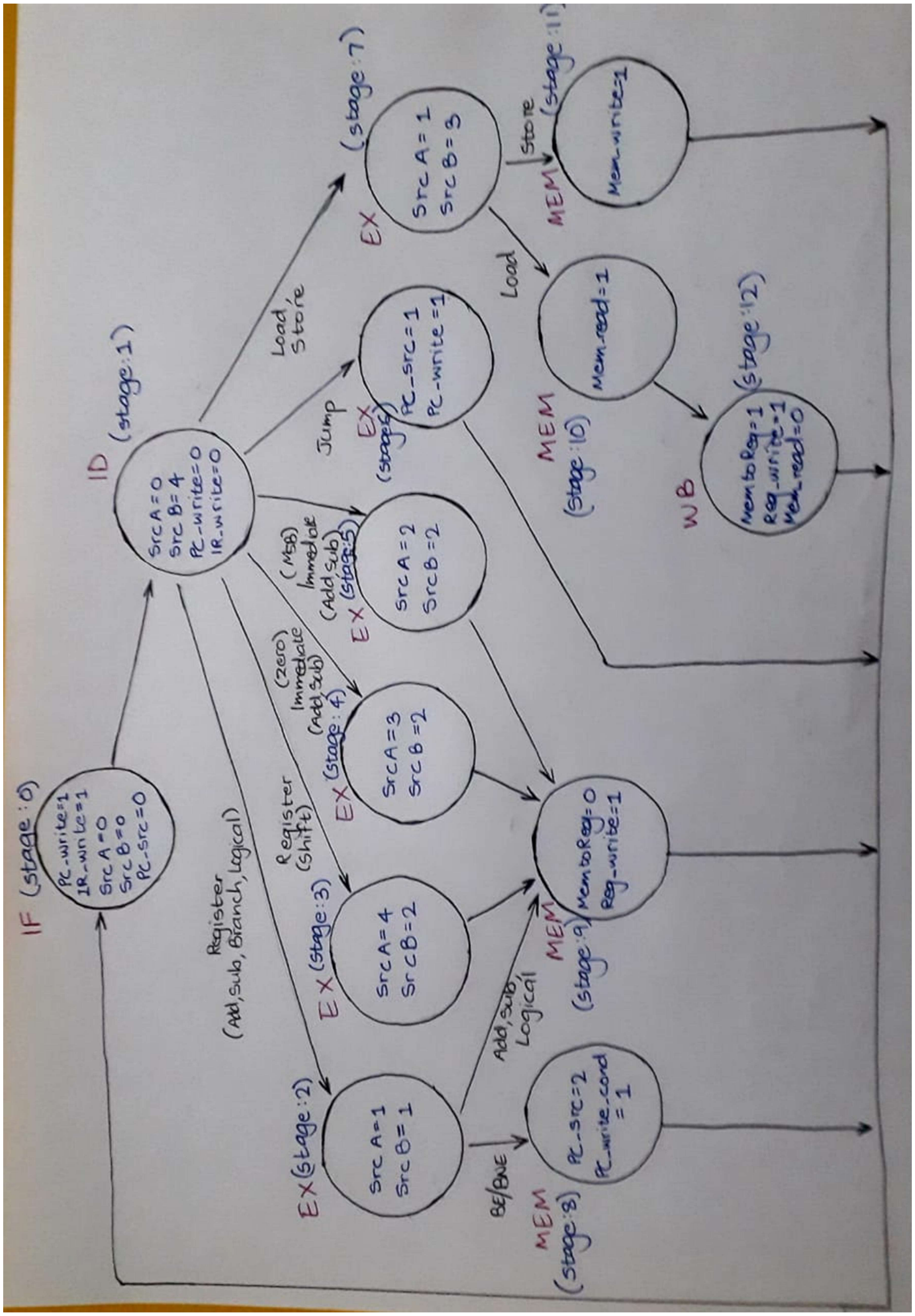
((zero * ~IR [12] | ~ zero * IR [12]) * PC_write_cond) | PC_write

This signal is used to enable the writing of branch address in PC for branch equal and branch not equal instructions.

Mux 4:
It determines the value to be stored in the PC register based on the PC_src control signal generated from the main control unit

Mux 5:
It determines the data to be provided at the WD port of Reg_file based on the Mem_to _Reg signal generated by the main control unit.

Main control unit:
The FSM for the main control unit is as follows



Control Signals:

Control Signal	Effect when asserted	Effect when deasserted
IR_write	IR is filled with the instruction	Value of IR is maintained
Reg_write	Data present in WD is written in register pointed by WRR	Value in the register pointed by WRR is maintained
PC_write	Value of PC is updated with the output of mux 4	Check the value of PC_write_cond to update PC
PC_write_cond	If the output of mux 3 is high, update the PC with the output of mux 4	Check PC write
Mem_read	Data present in the address pointed by read_addr is stored in MDR	Value of MDR is maintained
Mem_write	Data provided at WD port is written in the memory location pointed by write_addr	Contents of Data memory are maintained
Mem_to_Reg	Contents of MDR are made available at WD port of Reg file	Contents of ALU_out are made available at WD port of Reg file

Control Signal	0	1	2
PC_src	Value of ALU_result is passed as the output of mux 4	Value stored in ALU_out is passed as the output of mux 4 (occurs for jump instruction)	Value stored in register C is passed as the output of mux 4 (occurs for branch instructions)

Value	ALU_arc_A	ALU_src_B
0	mux21[15:0]<=PC	mux22[15:0]<=16’d2
1	mux21[15:0]<=A	mux22[15:0]<=B
2	mux21[15:0]<=sign extended IR[7:0]	mux22[15:0]<=C
3	mux21[15:0]<=zero extended IR[7:0]	mux22[15:0]<=(sign extend IR[7:0])<<1
4	mux21[15:0]<=zero extended IR[7:4]	mux22[15:0]<=sign extended IR[11:0]

ERROR LOG:
If an invalid instruction is given, three types of errors can be generated and displayed. The execution of the program is finished as soon as the error is detected.

ERROR_1: Contents of R0 cannot be modified
Register R0 is hardcoded to zero. If an attempt is made to change the contents of R0, the above error will be displayed

ERROR_2: Invalid function field for shift operation
In the case of shift instructions, if the func field of given instruction does not match any of the prespecified func fields (0001,0010,0011), the above error is displayed

ERROR_3: Cannot branch/ jump to an odd address in instruction memory
If the target address (for branch instructions) or the calculated address (for jump instruction) is odd, the above error is displayed because the instruction memory is even organized

TESTING:
The source code and the data files required for testing are also provided in the folder whose link is given below.
https://drive.google.com/drive/u/2/folders/1GOPT7QgDIg8Ek9EVHHmm6yFoA-tyGM37

Please refer to the README file for usage instructions.

When the code is simulated, the value of each control signal at every posedge will be displayed on the screen along with the timestamp. This can be found in the "**output.txt**" file in the drive

SAMPLE INSTRUCTIONS:

The following instructions were considered for testing the processor. All the 16 registers are initialized to 0000H each in the data file "**reg_file.dat**". At the end of execution, all the registers except R0 (which is hardcoded to 0000H) become FFFFH. This can be verified by checking the data file "**reg_file.dat**".

PC (HEX)	**Instr (HEX)**	**Equivalent Instr**	**OPERATION**	**RESULT**
0000	1000	load	R12 = data_mem[R8 + 0000H]	R12 = 1111H
0002	1501	load	R13 = data_mem[R9 + 0000H]	R13 = 0010H
0004	810C	add reg	R1 = R0 + R12	R1 = 1111H
0006	A222	add imm (zero)	R2 = R2 + 0022H	R2 = 0022H
0008	9388	add imm (sign)	R3 = R3 + FF88H	R3 = FF88H
000A	52C1	BNE	R12 and R1 are compared	false condition
000C	4DC1	BEQ	R12 and R1 are compared	PC = 0010H
000E	9000	add imm (sign)	invalid instr	ERROR: 1
0010	C432	sub reg	R4 = R3 - R2	R4 = FF66H
0012	D380	sub imm (sign)	R3 = R3 – FF80H	R3 = 0008H
0014	E466	sub imm (zero)	R4 = R4 – 0066H	R4 = FF00H
0016	B512	nand reg	R5 = R1 nand R2	R5 = FFFFH

0018	F632	or reg	R6 = R3 or R2	R6 = 002AH
001A	7477	nand imm (sign)	R4 = R4 nand 0077H	R4 = FFFFH
001C	6CAA	or imm (sign)	R12 = R12 or FFAAH	R12 = FFBBH
001E	3002	jump	PC = (PC + 2) + 0002H	PC = 0022H
0020	0438	shift	invalid instr	ERROR: 2
0022	0641	shl	R6 = R6 << 4 bits	R6 = 02A0H
0024	0182	shr	R1 = R1 >> 8 bits	R1 = 0011H
0026	0C83	sar	R12 = R12 >>> 8 bits	R12 = FFFFH
0028	2202	store	data_mem[R10 + 0004H] = R12	data_mem[4]=FFH data_mem[5]=FFH
002A	2703	store	data_mem[R11 + 0006H] = R13	data_mem[6]=10H data_mem[7]=00H
002C	8ED1	add reg	R14 = R1 + R13	R14 = 0021H
002E	AFFF	add imm (zero)	R15 = R15 + 00FFH	R15 = 00FFH
0030	9758	add imm (sign)	R7 = R7 + 0038H	R7 = 0038H
0032	4FC1	BEQ	R12 and R1 are compared	false condition
0034	57C1	BNE	R12 and R1 are compared	PC = 0038H
0036	0244	shift	invalid instr	ERROR: 2
0038	EE22	sub imm (zero)	R14 = R14 – 0022H	R14 = FFFFH
003A	CDD1	sub reg	R13 = R13 - R1	R13 = FFFFH
003C	D223	sub imm (sign)	R2 = R2 – 0023H	R2 = FFFFH
003E	B116	nand reg	R1 = R1 nand R6	R1 = FFFFH
0040	73F7	nand imm (sign)	R3 = R3 nand FFF7H	R3 = FFFFH
0042	67FF	or imm (sign)	RF7 = R7 or FFFFH	R7 = FFFFH
0044	0F81	shl	R15 = R15 << 8 bits	R15 = FF00H
0046	0F83	sar	R15 = R15 >>> 8 bits	R15 = FFFFH
0048	0682	shr	R6 = R6 >> 8 bits	R6 = 0002H
004A	F661	or reg	R6 = R6 or R1	R6 = FFFFH
004C	3002	jump	PC = (PC + 2) + 0004H	PC = 0052H
004E	0129	shift	invalid instr	ERROR: 2
0050	0128	shift	invalid instr	ERROR: 2
0052	F882	or reg	R8 = R8 or R2	R8 = FFFFH
0054	F993	or reg	R9 = R9 or R3	R9 = FFFFH
0056	FAA4	or reg	R10 = R10 or R4	R10 = FFFFH
0058	FBB5	or reg	R11 = R11 or R5	R11 = FFFFH

** Initial values for all 16 registers (R0 to R15) are 0000H each. After the execution, all registers except R0 (which is hardcoded to 0000H) become FFFFH