DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING

UNIVERSITY OF MORATUWA

EN3021 - Digital System Design

Non-pipelined Single Stage (Cycle) CPU Design – Individual Project

Kishokkumar R. – 200306X

Project specifications

Need to implement:

1. All computational instructions covered by instruction types R and I.
2. All memory access instructions (load and store) - I and S type instructions
3. All Control Flow instructions: SB type

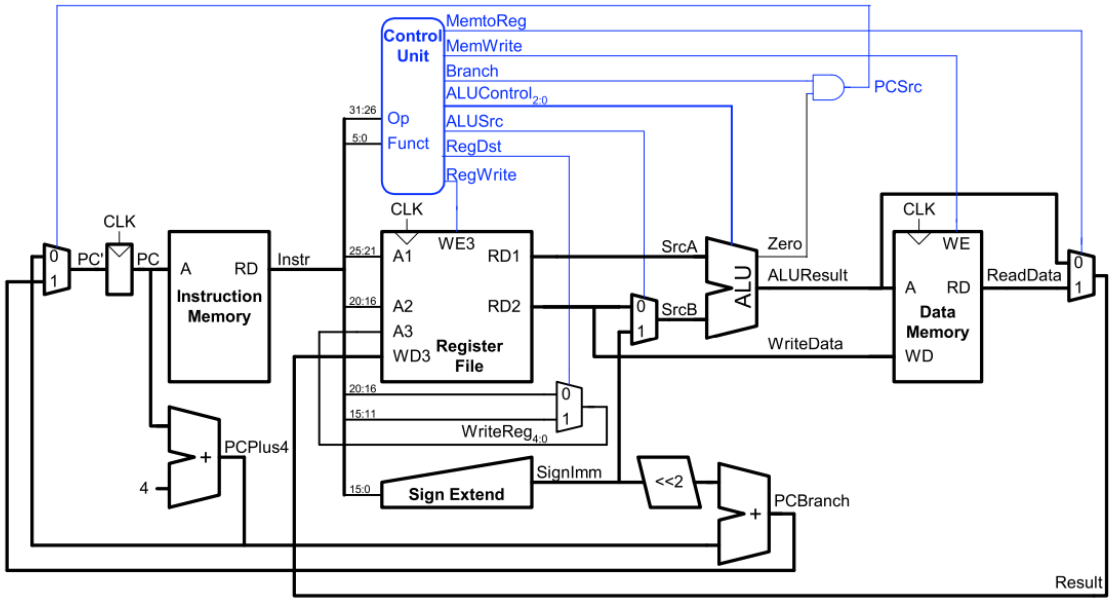
need to add support to two new instructions.
(a) MEMCOPY - Copies an array of size N from one location to another. N > 1

- Determine the max N that you can use for this instruction)

(b) MUL - Unsigned Multiplication (Note that RV32I does not include multiplication instructions)

- Identify the limits for operands of this instruction.

MIPS architecture complete data path



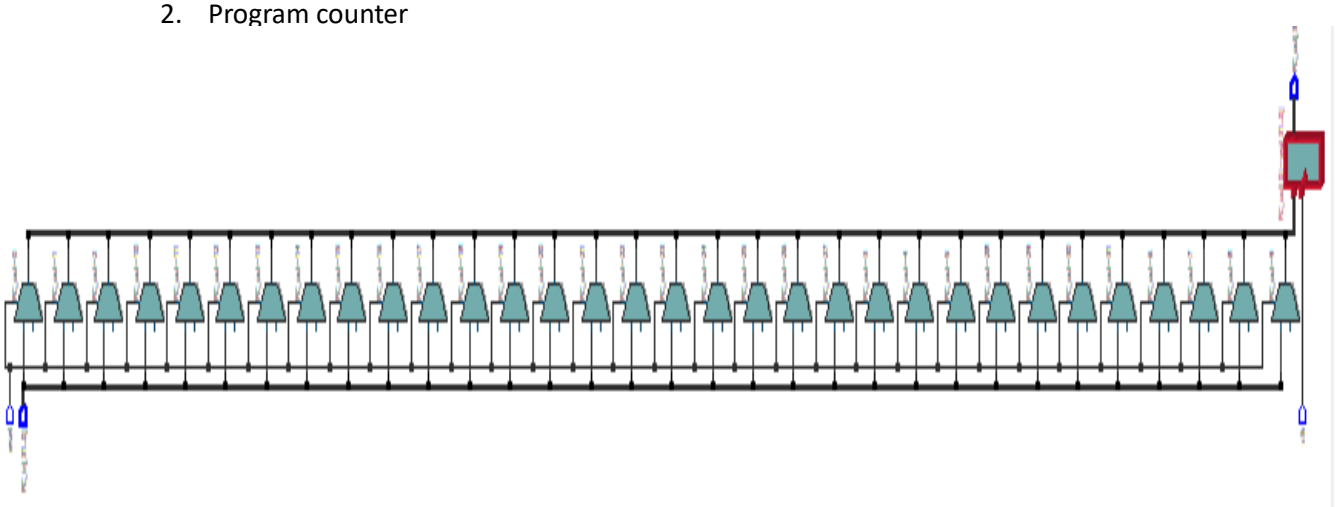
Designed modules.

1. ALU module

   It has arithmetic operations, and it will give required result for branch instructions.

operations	4 – Bit representations
AND	0000
OR	0001
XOR	0011
ADD	0010
SUB	0110
SLL	0100
SLT (set on less than)	0101
SLTU (set on less than unsigned)	1010
BRANCH (unsigned)	0111
SRL (logical right shift)	1000
SRA (Arithmetic right shift)	1100

2. Program counter



The PC_in input can be used to load a new value into the Pcounter module, which functions as a straightforward 32-bit counter. It functions on the clk signal's rising edge. PC_out is set to all zeros (32'h00000000), the counter is reset, and the reset signal is asserted (set to 1). The value from the PC_in input is copied to the PC_out output by the module when the reset signal is de-asserted or set to 0.

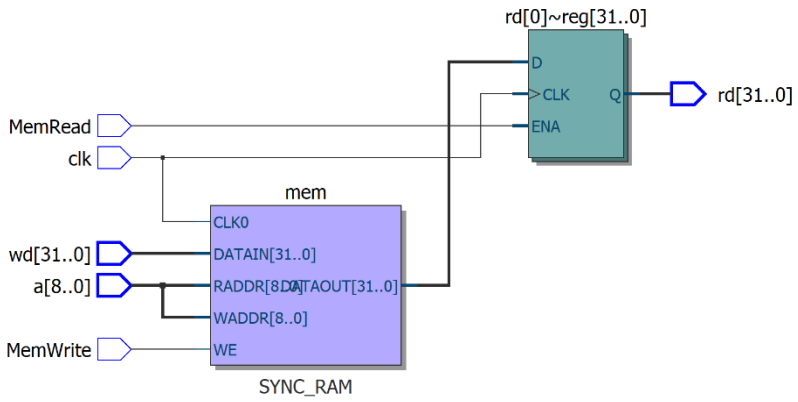
3. Register file.

NUM_REGS specifies the number of registers, and DATA_WIDTH bits determine the data width of each register. Register_file is the representation of the registers.

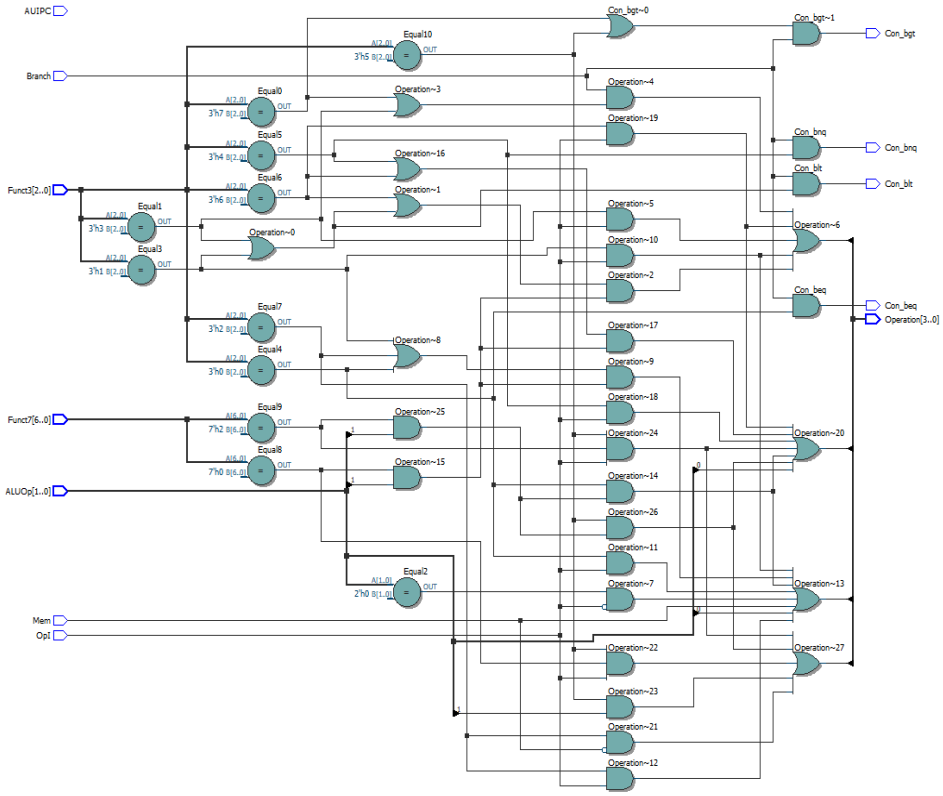
The following procedures are conducted by the module on each clock signal's negative edge:

The register file is cleaned if the first signal is asserted (set to 1). Zero is entered into each register.
A write operation takes place if rg_wrt_en is set to 1 and the rst signal is de-asserted, or set to 0. The register designated by rg_wrt_dest receives the data supplied by rg_wrt_data.
The module constantly modifies rg_rd_data1 and rg_rd_data2 based on the read addresses rg_rd_addr1 and rg_rd_addr2, regardless of the circumstances. The contents of the designated registers are provided by these outputs.

4. Data memory



5. ALU controller



ALU Operation	Inputs	Bit Representation
AND	ALUOp = 2'b00, Funct7 = 7'b0000000, Funct3 = 3'b111, OpI = 0	0001
OR	ALUOp = 2'b00, Funct7 = 7'b0000000, Funct3 = 3'b110, OpI = 0	0000
XOR	ALUOp = 2'b00, Funct7 = 7'b0000000, Funct3 = 3'b100, OpI = 0	0010

ADD	ALUOp = 2'b00, Funct7 = 7'b0000000, Funct3 = 3'b000, OpI = 0	0011
SUBTRACT	ALUOp = 2'b00, Funct7 = 7'b0100000, Funct3 = 3'b000, OpI = 0	0110
SHIFT LEFT	ALUOp = 2'b00, Funct7 = 7'b0000000, Funct3 = 3'b001, OpI = 0	0100
SET LESS THAN	ALUOp = 2'b00, Funct7 = 7'b0000000, Funct3 = 3'b110, OpI = 0	1000
SET LESS THAN UNSIGNED	ALUOp = 2'b00, Funct7 = 7'b0100000, Funct3 = 3'b110, OpI = 0	1100
SHIFT RIGHT LOGICAL	ALUOp = 2'b00, Funct7 = 7'b0000000, Funct3 = 3'b101, OpI = 0	0101
SHIFT RIGHT ARITHMETIC	ALUOp = 2'b00, Funct7 = 7'b0100000, Funct3 = 3'b101, OpI = 0	1101
SLT (Set Less Than)	ALUOp = 2'b00, Funct7 = 7'b0000000, Funct3 = 3'b010, OpI = 0	0111

6. Instruction memory

RV32I Base Instruction Set

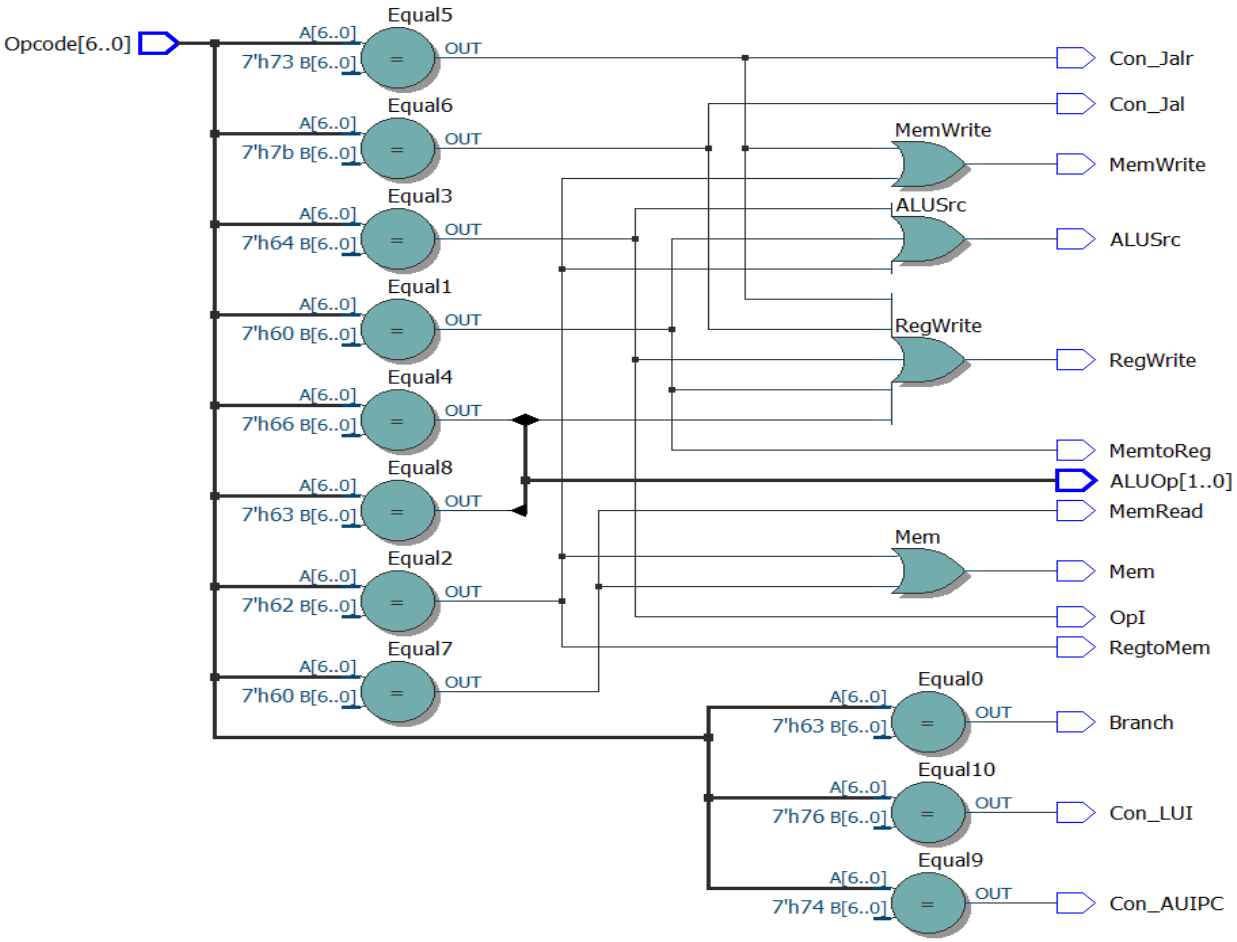
imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

All instructions are encoded according to the table, with the main considerations being the opcode, function 3, and function 7 parts. The exception to this encoding scheme is for the instructions LH, LB, LBU, LHU, SH, and SB.

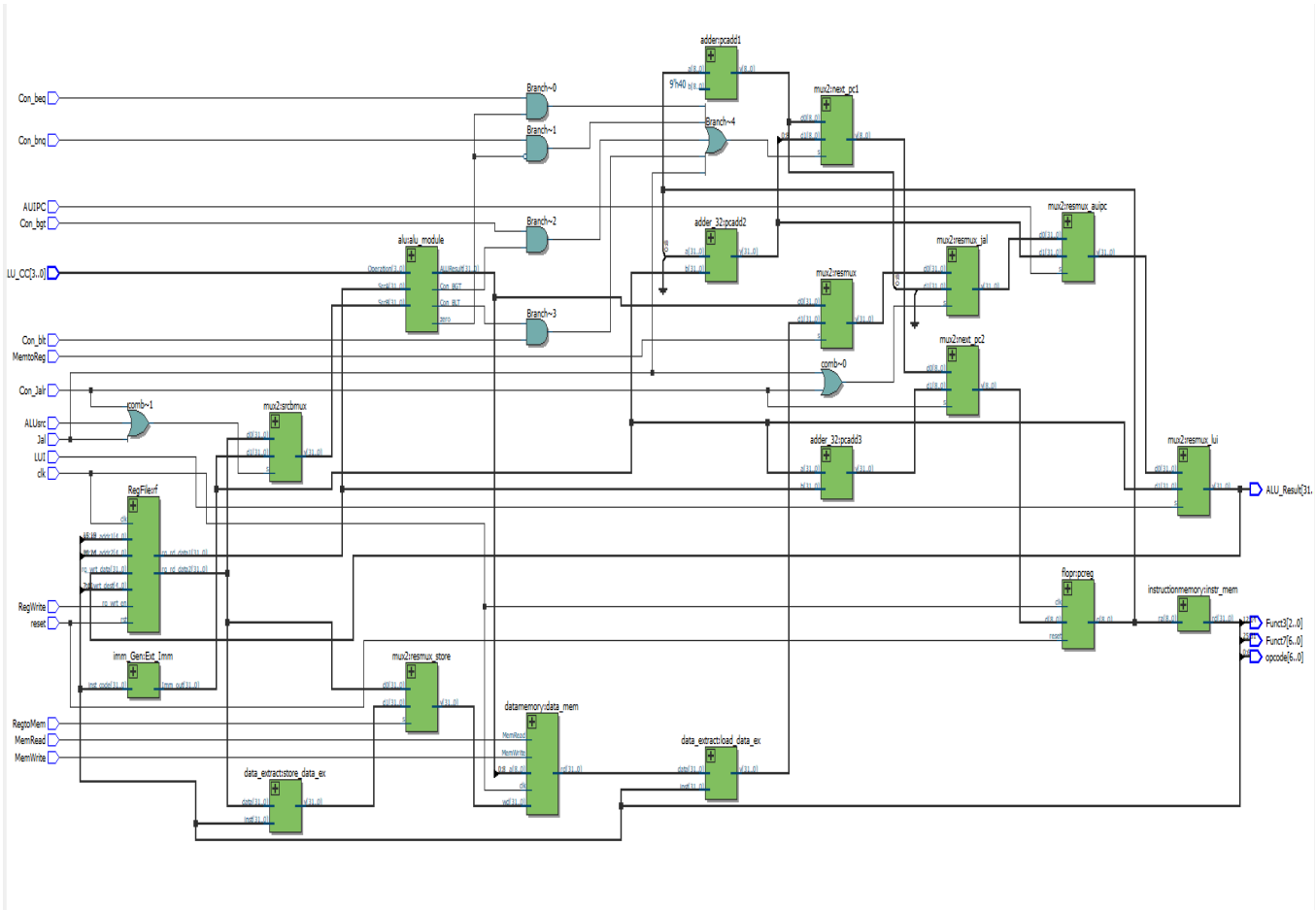
7. Contoller

Opcode	**Con_Jal**	**Con_Jalr**	**Branch**	**ALUSrc**	**MemtoReg**	**RegtoMem**	**RegWrite**	**Mem**	**MemRead**	**MemWrite**	**ALUOp[0]**	**OpI**	**ALUOp[1]**	**Con_AUIPC**	**Con_LUI**
JAL	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0

JALR	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0
BR	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
LW	0	0	0	1	1	0	1	1	1	0	0	0	0	0	0
SW	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0
RTypeI	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0
R_TYPE	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0
AUIPC	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0
LUI	0	0	0	1	0	0	1								


Opcode[6..0]
Equal5
A[6..0]
7'h73 B[6..0]
OUT
Equal6
7'h7b
Equal3
7'h64
Equal1
7'h60
Equal4
7'h66
Equal8
7'h63
Equal2
7'h62
Equal7
7'h60
Equal0
7'h63
Equal10
7'h76
Equal9
7'h74
MemWrite
ALUSrc
RegWrite
Mem
Con_Jalr
Con_Jal
MemWrite
ALUSrc
RegWrite
MemtoReg
ALUOp[1..0]
MemRead
Mem
OpI
RegtoMem
Branch
Con_LUI
Con_AUIPC

8. Data path


adder:pcadd1
Branch~0
Branch~1
Branch~4
mux2:next_pc1
Con_beq
Con_bne
AUIPC
Con_bgt
LU_CC[3..0]
Branch~2
adder_32:pcadd2
mux2:resmux_auipc
mux2:resmux_lal
alualu_module
mux2:resmux
Branch~3
Con_blt
MemtoReg
Con_Jalr
ALUsrc
Jal
LUI
clk
comb~0
mux2:next_pc2
comb~1
mux2:srcbmux
adder_32:pcadd3
mux2:resmux_lui
ALU_Result[31..
RegFile:rf
RegWrite
reset
flopr:pcreg
instructionmemory:instr_mem
Funct3[2..0]
Funct7[6..0]
opcode[6..0]
imm_Gen:Ext_Imm
mux2:resmux_store
RegtoMem
MemRead
MemWrite
datamemory:data_mem
data_extract:load_data_ex
data_extract:store_data_ex

Implementing new instruction

MEMcopy–

Memcpy uses CPU cycles for the following tasks:

1. Load/store of data.
2. Extra computations (like address alignment processing).
3. Branch forecasting.

Multiplication:

In our design, the ALU does not have a multiplication operation. If we need to implement it, it will require two instructions:

1. SRL - while doing this, we can multiply by 2^n.
2. ADD – it can be useful for multiplication other than 2^n.

Resource utilization

Analysis & Synthesis Resource Utilization by Entity

Compilation Hierarchy Node	LC Combinationals	LC Registers	Memory Bits	DSP Elements	DSP 9x9	DSP 18x18	Pins	Virtual Pins	Full Hierarchy Name
|riscv	2765 (0)	1033 (0)	16384	0	0	0	43	0	|riscv
|ALUController:ac|	22 (22)	0 (0)	0	0	0	0	0	0	|riscv|ALUController:ac
|Controller:c|	13 (13)	0 (0)	0	0	0	0	0	0	|riscv|Controller:c
|Datapath:dp|	2730 (7)	1033 (0)	16384	0	0	0	0	0	|riscv|Datapath:dp
|RegFile:rf|	1433 (1433)	1024 (1024)	0	0	0	0	0	0	|riscv|Datapath:dp|RegFile:rf
|adder:pcadd1|	7 (7)	0 (0)	0	0	0	0	0	0	|riscv|Datapath:dp|adder:pcadd1
|adder_32:pcadd2|	32 (32)	0 (0)	0	0	0	0	0	0	|riscv|Datapath:dp|adder_32:pcadd2
|adder_32:pcadd3|	11 (11)	0 (0)	0	0	0	0	0	0	|riscv|Datapath:dp|adder_32:pcadd3
|alu:alu_module|	563 (563)	0 (0)	0	0	0	0	0	0	|riscv|Datapath:dp|alu:alu_module
|data_extract:load_data_ex|	79 (79)	0 (0)	0	0	0	0	0	0	|riscv|Datapath:dp|data_extract:load_data
|data_extract:store_data_ex|	74 (74)	0 (0)	0	0	0	0	0	0	|riscv|Datapath:dp|data_extract:store_dat
|datamemory:data_mem|	0 (0)	0 (0)	16384	0	0	0	0	0	|riscv|Datapath:dp|datamemory:data_mem
|altsyncram:mem_rtl_0|	0 (0)	0 (0)	16384	0	0	0	0	0	|riscv|Datapath:dp|datamemory:data_mem|al
|altsyncram_isd1:auto_generated|	0 (0)	0 (0)	16384	0	0	0	0	0	|riscv|Datapath:dp|datamemory:data_mem|al
|flopr:pcreg|	9 (9)	9 (9)	0	0	0	0	0	0	|riscv|Datapath:dp|flopr:pcreg
|imm_Gen:Ext_Imm|	59 (59)	0 (0)	0	0	0	0	0	0	|riscv|Datapath:dp|imm_Gen:Ext_Imm
|instructionmemory:instr_mem|	136 (136)	0 (0)	0	0	0	0	0	0	|riscv|Datapath:dp|instructionmemory:inst
|mux2:resmux_lui|	253 (253)	0 (0)	0	0	0	0	0	0	|riscv|Datapath:dp|mux2:resmux_lui
|mux2:resmux_store|	32 (32)	0 (0)	0	0	0	0	0	0	|riscv|Datapath:dp|mux2:resmux_store
|mux2:srcbmux|	35 (35)	0 (0)	0	0	0	0	0	0	|riscv|Datapath:dp|mux2:srcbmux

Note: For table entries with two numbers listed, the numbers in parentheses indicate the number of resources of the given type used by the specific entity alone. The numbers listed outside of parent

Github link - https://github.com/kishokkumar200306X/EN3021-stage_1