UNIVERSITI MALAYSIA PERLIS

NMJ 31804 - Principles of Computer Architecture

Semester 2 (2021/2022)

Mini Project Report

Title: Design of 16-bit CPU

Prepared by:

NO	NAME	MATRIC NO	PROGRAM NAME & CODE
1.	SWETHA A/P SHANMUGAM STALIN	191022310	COMPUTER ENGINEERING RK20
2.	TAN YI JIE	191020976	COMPUTER ENGINEERING RK20
3.	IKMAL KHUZAIRI BIN KAMARUL BADRI	191020897	COMPUTER ENGINEERING RK20
4.	AZLIN NATASHA BINTI AZMAN FAUZI	191020873	COMPUTER ENGINEERING RK20

Lecturer: PM Dr. Phaklen Ehkan

1.0 INTRODUCTION

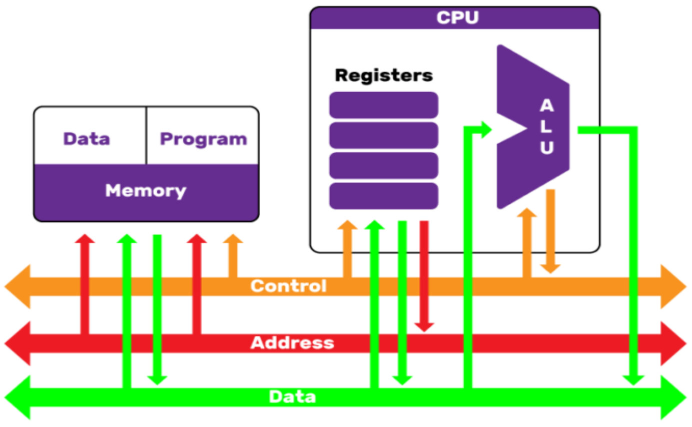
CPUs that process 16 bits instead of 8, 32, or 64 as a single unit. When the IBM PC debuted in 1981, the first personal computers, which had 8-bit CPUs in the late 1970s, switched to 16-bit processors. Although many goods do not need the increased performance, 16-bit CPUs are nevertheless employed as embedded processors in many different products. To carry out the operation and compute the output, a set of instructions or group of programmes (software) are written to the microprocessor. A physical set of hardware modules can be used to achieve this. The modules that are most often utilized are the Arithmetic Logic Unit (ALU), Control Unit, Registers, and Instruction Execution Unit.

The term "RISC," or Reduced Instruction Set Computer, refers to information processing using any of a series of microprocessors designed to do computing tasks with the fewest number of instructions in the shortest period of time. In contrast, computers designed with a full set of computer instructions were referred to as "CISC" (complex instruction set computers) in order to supply necessary capabilities in the most efficient way. RISC is the replacement for CISC (Complex Instruction Set Computer).

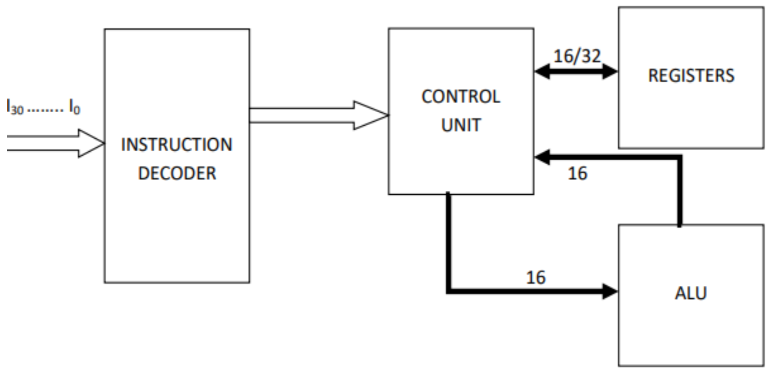
Later, it was discovered that the computer could perform more work in a shorter amount of time for the majority of applications by condensing the whole set of instructions to merely the most commonly used ones. Computers normally employ CISC, whereas smartphones, tablets, and other electronic devices use RISC. All modern microprocessors, including RISC-V, MIPS, PowerPC, Atmel's AVR, Microchip PIC processors, and Arm processors, incorporate at least a portion of the RISC architecture. The shift from 8-bit and 16-bit to 32-bit architectures, in particular, necessitated the need for RISC architectures. However, it took RISC architectures ten years to gain traction, mostly because there was no software designed to run on them. Intel also had an impact because it didn't see the need to completely redesign because it had the resources to keep using the CISC architecture. One of the first RISC ISAs was the MIPS architecture.

1.1 The Architecture of CPU

A CPU is composed of three major elements which are registers, arithmetic logic unit (ALU), and control unit (CU).



1.1.1 The Architecture of RISC



The above diagram shows the 16-bit architecture of RISC (Reduced Instruction Set Computer). The basic architecture of a RISC would have an instruction decoder, control unit, arithmetic control unit and registers.

2.0 METHODOLOGY

2.1 Instruction Set of the Processor

A. Memory Access Instructions

1. Load Word:

    LD ws, offset(rs1) ws:=Mem16[rs1 + offset]

2. Store Word:

    ST rs2, offset(rs1) Mem16[rs1 + offset]=rs2

B. Data Processing Instructions

1. Add:

    ADD ws, rs1, rs2 ws:=rs1 + rs2

2. Subtract:

    SUB ws, rs1, rs2 ws:=rs1 – rs2

3. Invert (1‘s complement):

    INV ws, rs1 ws:=!rs1

4. Logical Shift Left:

    LSL ws, rs1, rs2 ws:=rs1 << rs2

5. Logical Shift Right:

    LSR ws, rs1, rs2 ws:=rs1 >> rs2

6. Bitwise AND:

    AND ws, rs1, rs2 ws:=rs1 & rs2

7. Bitwise OR:

    OR ws, rs1, rs2 ws:=rs1 | rs2

C. Control Flow Instructions

1. Branch on Equal:

    BEQ rs1, rs2, offset

    Branch to (PC + 2 + (offset << 1)) when rs1 = rs2

2. Branch on Not Equal:

    BNE rs1, rs2, offset

    Branch to (PC + 2 + (offset << 1)) when rs1 != rs2

3. Jump: JMP offset Jump to {PC [15:13], (offset << 1)}

2.1.1 Instruction Format of the Processor

- Memory Access: Load

Op	Rs1	Ws	offset
4	3	3	6

- Memory Access: Store

Op	Rs1	Rs2	offset
4	3	3	6

- Data Processing

Op	Rs1	Rs2	Ws	useless
4	3	3	3	3

- Branch

Op	Rs1	Rs2	offset
4	3	3	6

- Jump

Op	offset
4	12

The above instruction format of the processor shows the number of bits used for each instructions.

2.1.2 Opcodes for the Operations of Processor

Decimal	Opcode (Binary)	Operation
0	0000	Load word
1	0001	Store word
2	0010	Add
3	0011	Subtract
4	0100	Invert (1's complement)
5	0101	Logical shift left
6	0110	Logical shift right
7	0111	Bitwise AND
8	1000	Bitwise OR
9	1001	Set on less than
10	1010	Add (Different register)
11	1011	Branch on equal
12	1100	Branch on not equal
13	1101	Jump

Table 2.1 shows all of the opcodes that have been used in the processor. If an opcode of 0000 has been used, the processor will run the operation of load word from one register to another register. The processor will run certain operations according to the use of opcode of instruction.

2.2 Control Signal of the Processor

Control Signal									
Instruction	Reg Dst	ALU Src	Mem to Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp	Jump
Data processing	1	0	0	1	0	0	0	00	0
LW	0	1	1	1	1	0	0	10	0
SW	0	1	0	0	0	1	0	10	0
BEQ, BNE	0	0	0	0	0	0	1	01	0
J	0	0	0	0	0	0	0	00	1

Table 2.2 shows the control signal that we designed for our processor. For data processing operation, the register destination and register write will be set at 1 while the other signals will be set at 0. The ALU opcode for the data processing will be 00. There are several more instructions being used which are load word, store word, branch on equal, branch on not equal and jump. All these instructions have different control signals.

2.2.1 ALU Control Unit

ALU control				
ALUOp	Opcode (Hex)	ALUcnt	ALUOperation	Instruction
10	xxxx	0000	ADD	LW, SW
01	xxxx	0001	SUB	BEQ, BNE
00	0002	0000	ADD	dType: ADD
00	0003	0001	SUB	dType: SUB
00	0004	0010	INVERT	dType: INVERT
00	0005	0011	LSL	dType: LSL
00	0006	0100	LSR	dType: LSR
00	0007	0101	AND	dType: AND
00	0008	0110	OR	dType: OR
00	0009	0111	SLT	dType: SLT
00	000A	1000	MUL	dType: MUL

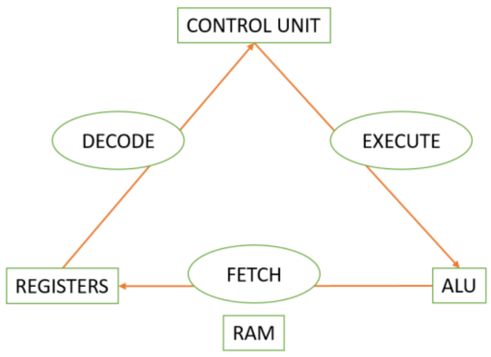
Table 2.3 shows the control unit of the arithmetic logic unit (ALU). The instructions of load and store will have the same ALUOp which is 10 while branch on equal and branch on not equal will have the value of 01. The other instruction will have the value of 00 for the ALUOp.

The ALU control consists of several instructions that are load word, store word, branch on equal, branch on not equal, addition, subtraction, invert, logical shift left, logical shift right, AND operation, OR operation, set on less than, and multiplication operation. Each of the instructions will have a different value of ALUcnt.

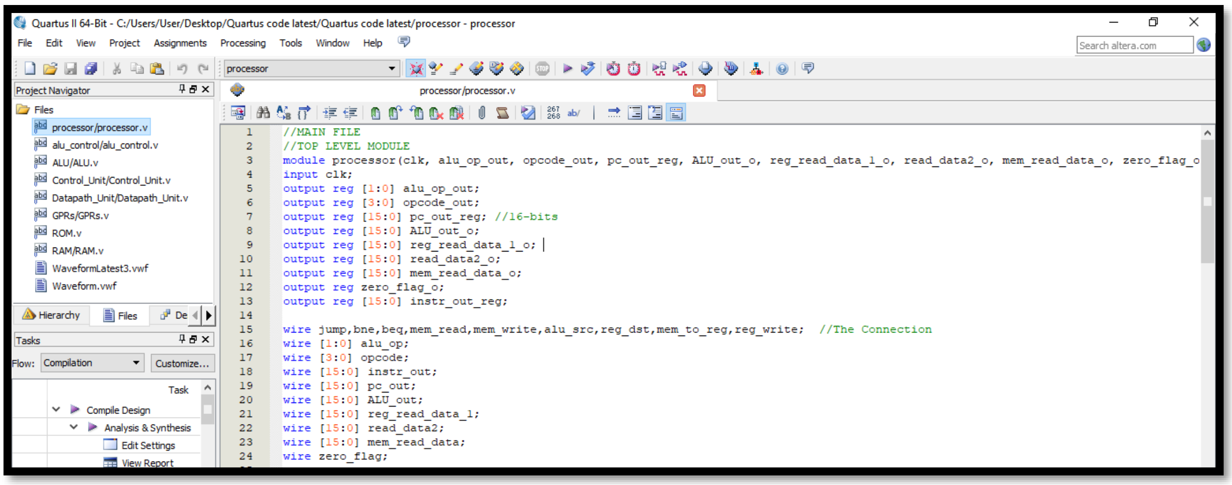
2.3 Fetch, decode and execution process of CPU

The fetch-decode-execute cycle is the CPU's main mechanism for running programs. The layout of registers, memory, and buses is included in our CPU design. We desire a CPU design that have the operation of fetch (CPU retrieves data and instructions from RAM), decode (instructions are decoded by the Control Unit), execute (ALU operates on data and carries out instructions) and store (RAM holds the processed data).

The programme's code is copied from secondary storage into the main memory in order to run a programme. The first instruction in the programme is put in memory, and then execution starts when the CPU's programme counter is set to that address. A memory address in the main memory is occupied by each machine code instruction in a program. A distinct memory address is assigned to each module. The programme counter, which also directs the CPU on their execution order, stores the address of each instruction. The CPU performs the fetch-decode-execute cycle when a program is being performed. This cycle is repeated until the STOP instruction is reached.

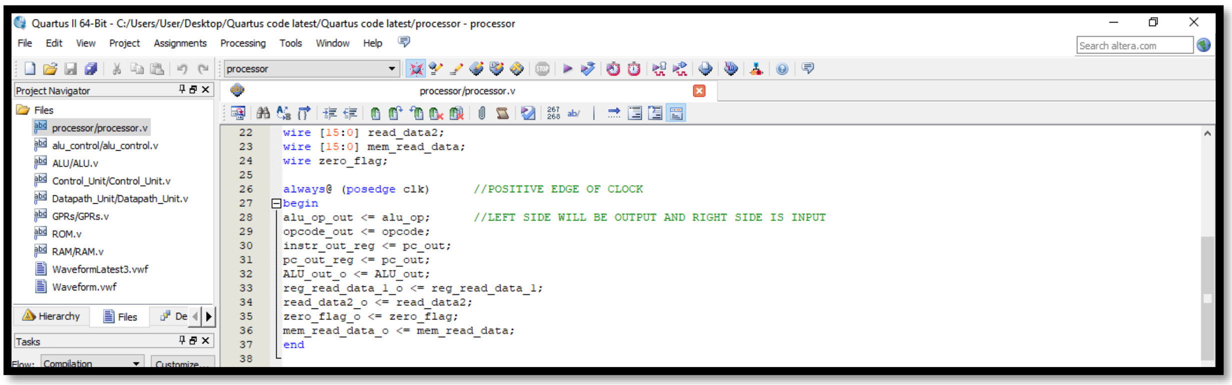


2.4 Top-level module of CPU design



```
//MAIN FILE
//TOP LEVEL MODULE
module processor(clk, alu_op_out, opcode_out, pc_out_reg, ALU_out_o, reg_read_data_1_o, read_data2_o, mem_read_data_o, zero_flag_o
input clk;
output reg [1:0] alu_op_out;
output reg [3:0] opcode_out;
output reg [15:0] pc_out_reg; //16-bits
output reg [15:0] ALU_out_o;
output reg [15:0] reg_read_data_1_o;
output reg [15:0] read_data2_o;
output reg [15:0] mem_read_data_o;
output reg zero_flag_o;
output reg [15:0] instr_out_reg;

wire jump,bne,beq,mem_read,mem_write,alu_src,reg_dst,mem_to_reg,reg_write;  //The Connection
wire [1:0] alu_op;
wire [3:0] opcode;
wire [15:0] instr_out;
wire [15:0] pc_out;
wire [15:0] ALU_out;
wire [15:0] reg_read_data_1;
wire [15:0] read_data2;
wire [15:0] mem_read_data;
wire zero_flag;
```


Quartus II 64-Bit - C:/Users/User/Desktop/Quartus code latest/Quartus code latest/processor - processor
Project Navigator
Files
processor/processor.v
alu_control/alu_control.v
ALU/ALU.v
Control_Unit/Control_Unit.v
Datapath_Unit/Datapath_Unit.v
GPRs/GPRs.v
ROM.v
RAM/RAM.v
WaveformLatest3.vwf
Waveform.vwf
Hierarchy
Files
Tasks
processor/processor.v

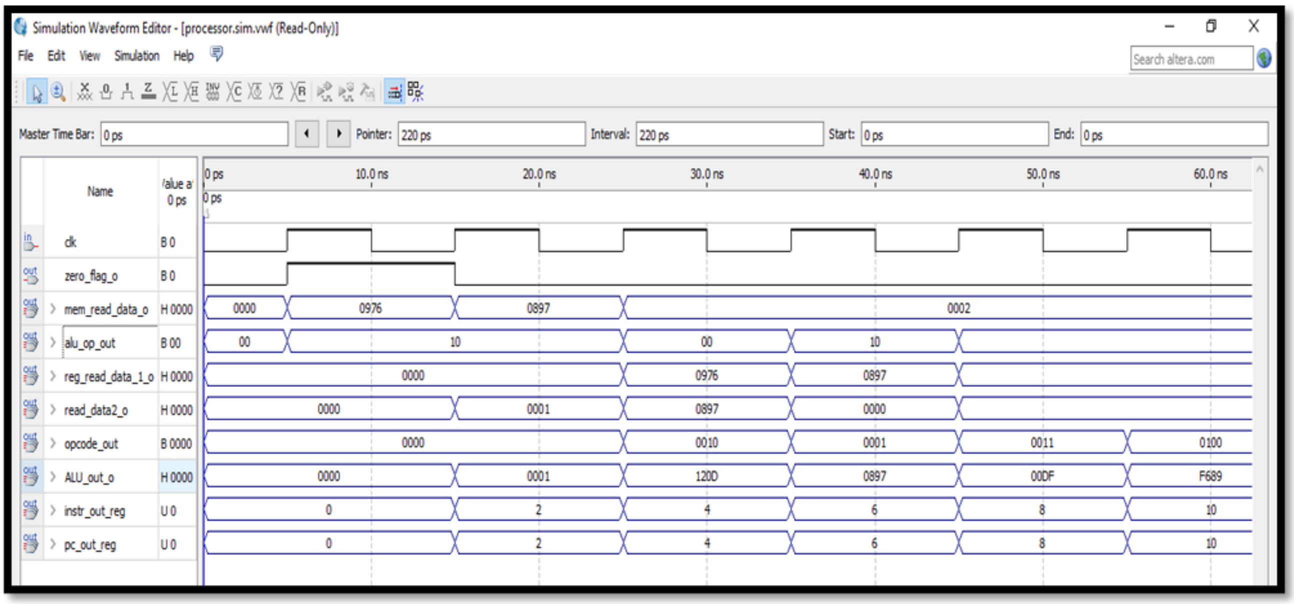
```
wire [15:0] read_data2;
wire [15:0] mem_read_data;
wire zero_flag;

always@ (posedge clk)      //POSITIVE EDGE OF CLOCK
begin
alu_op_out <= alu_op;      //LEFT SIDE WILL BE OUTPUT AND RIGHT SIDE IS INPUT
opcode_out <= opcode;
instr_out_reg <= pc_out;
pc_out_reg <= pc_out;
ALU_out_o <= ALU_out;
reg_read_data_1_o <= reg_read_data_1;
read_data2_o <= read_data2;
zero_flag_o <= zero_flag;
mem_read_data_o <= mem_read_data;
end
```

At top level module, we have create module named as processor and have many list of I/O signals(ports) such as clk, alu_op_out, opcode_out, pc_out_reg and etc. For clk which is clock cycle we set as input port direction declarations while others we put as output port direction declaration. Furthermore, As its sated there also have many wires for every submodule. Wire means as internal wire (net) declarations. The processor start its operation at each positive edge of the clock cycle and it will do all the operation in the submodule. The CPU is designed without any external device connections.

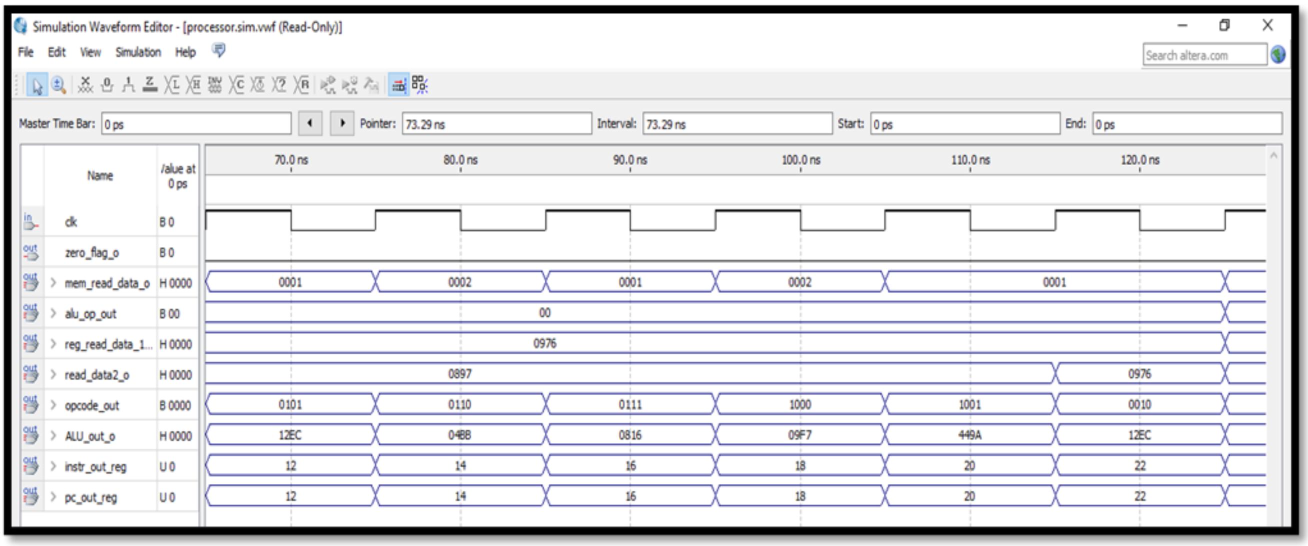
3.0 RESULT & DISCUSSION

3.1 Result


Simulation Waveform Editor - [processor.sim.vwf (Read-Only)]
Master Time Bar: 0 ps
Pointer: 220 ps
Interval: 220 ps
Start: 0 ps
End: 0 ps
Name
Value at 0 ps
0 ps
10.0 ns
20.0 ns
30.0 ns
40.0 ns
50.0 ns
60.0 ns
clk B 0
zero_flag_o B 0
mem_read_data_o H 0000: 0000 0976 0897 0002
alu_op_out B 00: 00 10 00 10
reg_read_data_1_o H 0000: 0000 0976 0897
read_data2_o H 0000: 0000 0001 0897 0000
opcode_out B 0000: 0000 0010 0001 0011 0100
ALU_out_o H 0000: 0000 0001 120D 0897 00DF F689
instr_out_reg U 0: 0 2 4 6 8 10
pc_out_reg U 0: 0 2 4 6 8 10

At the third positive edge of the clock cycle, the processor is doing an addition operation. At alu_op_out that takes 2 bits which is 00 means it’s doing data processing. The CPU takes register data A value which is 0976H and register data B value which is 0897H. The ALU control (at opcode_out) for addition operation is 0010 bit, So the result will be (0976H + 0897H = 120DH) and 120D will store at ALU_out operation. The ROM register and program counter will increment by two for every positive edge of duty cycle.

For the next positive edge of duty cycle, the alu_op_out doing load and store that being addressed by 2 bit (10). The value of register data 2, 0897H have been loaded and stored at ALU_out operation. Then, The opcode_out at 0010 stands for subtraction operation, So the result will be (0976H - 0897H = 00DFH) and 00DF will store at ALU_out operation. Same goes to the operation that beside it, it will do data processing which is inverting of A that the opcode_out will be 0100. The value of A, 0976H (0000 1001 0111 0110) been inverted into a result of F689H (1111 0110 1000 1001) and stored at ALU_out_operation.


Simulation Waveform Editor - [processor.sim.vwf (Read-Only)]
File Edit View Simulation Help
Master Time Bar: 0 ps
Pointer: 73.29 ns
Interval: 73.29 ns
Start: 0 ps
End: 0 ps
Name
Value at 0 ps
70.0 ns
80.0 ns
90.0 ns
100.0 ns
110.0 ns
120.0 ns
clk B 0
zero_flag_o B 0
mem_read_data_o H 0000 0001 0002 0001 0002 0001
alu_op_out B 00 00
reg_read_data_1... H 0000 0976
read_data2_o H 0000 0897 0976
opcode_out B 0000 0101 0110 0111 1000 1001 0010
ALU_out_o H 0000 12EC 04BB 0816 09F7 449A 12EC
instr_out_reg U 0 12 14 16 18 20 22
pc_out_reg U 0 12 14 16 18 20 22

Now, for the opcode_out at 0101 and 0110. Both will do data processing for shift left and shift right accordingly. Since it just take the value register A so the result will stored at ALU_out_operation.

The value of A	: 0976H (0000 1001 0111 0110)
Shift left (0101)	: 12ECH (0001 0010 1110 1100)
The value of A	: 0976H (0000 1001 0111 0110)
Shift right (0110)	: 04BBH (0000 0100 1011 1011)

Other than that, the value opcode_out for AND operation is 0111, the value of register A, 0976H and value of register B, 0997H will go through AND operation, then the result, 0816H will be stored at ALU. For OR operation the opcode_out is 1000, same goes to AND operation it will take value of A and B, the result of OR operation, 09F7H will be stored at ALU.For the last part which is opcode _out is 1001, the value of register A and value of register B will do the multiplication operation, then the result, 51449AH will be will be stored at ALU_out operation. Since our CPU can only hold for sixteen bits so the value that stored is just 449AH.

3.2 Discussion

Based on our observations throughout the 16-bit CPU design process, we found that there are some limitations on the 16-bit CPUs. The 16-bit processors are restricted to only a few registers. An n-bit register may hold 2n different values, as is common knowledge. As a result, a 16-bit register can only hold 65536 different values while a 32-bit processor's register can hold as many as 4294967296 values. A16-bit CPU can only access up to 64kByte of byte-addressable memory (128kByte for architectures with separate instruction and data spaces) while a 32-bit CPU can access up to 4GByte. This condition can be known from the result of multiplication operation.   Since our CPU can only hold for sixteen bits so the value that stored is just 449AH, not the original value, 51449AH.

Numerous changes can be made to the design. More elements can be added to the current design to create a more sophisticated one. The processor's instruction support capacity can be enhanced. Pipelining can be incorporated to the suggested design to boost performance.

A technique called "pipelining" overlaps many instructions as they are being carried out. By tying the various pipeline phases together, a structure resembling a pipe is produced. It makes it possible to store and carry out instructions in a methodical way. At one end, instructions enter, and at the other, they exit. The overall throughput of instructions is increased via pipelining. An input register and a combinational circuit make up each segment in a pipeline system. The combinational circuit performs operations on the data that is stored in the register by the register. The output of the combinational circuit is sent into the input register of the following segment.

4.0 CONCLUSION

Based on our simulation, we have successfully designed a CPU with the structure of 16 -bit RISC that consists of a Datapath unit and Control Unit. Our design was simulated successfully through Altera Quartus software. Verilog was being used to create the design, and Modelsim was used to simulate it. The majority of the objectives were met, and the resuts indicate that the processor is operating with no errors. By expanding the amount of instructions and creating a pipelined design with fewer clock cycles per instruction, future work will be able to bring further improvements.

REFERENCES

Architecture of a CPU. (n.d.). FutureLearn. https://www.futurelearn.com/info/courses/how-computers-work/0/steps/49283

Computer Organization RISC and CISC. (2022, Jan 13). GeeksforGeeks. https://www.geeksforgeeks.org/computer-organization-risc-and-cisc/

Thornton, S. (2018, Jan 9). RISC vs. CISC Architectures: Which one is better?. MICROCONTROLLERTiPS. https://www.microcontrollertips.com/risc-vs-cisc-architectures-one-better/

What is Pipelining in Computer Architecture?. (n.d.). tutorialspoint. https://www.tutorialspoint.com/what-is-pipelining-in-computer-architecture