

**Department of Electrical and Computer Engineering**

**COMSATS University Islamabad, Wah campus**

# 16-Bit Single Cycle Processor

Waleed Niaz

REG. NO: FA16-BEE-054  
subject: Digital System Designs

**Instructor Remarks and Grade:**

sa

d

asd

**Contents**

[16-Bit Single Cycle Processor 1](#_Toc28081484)

[Chapter 1 3](#_Toc28081485)

[Introduction 3](#_Toc28081486)

[Chapter 2 4](#_Toc28081487)

[Project description 4](#_Toc28081488)

[Rformat 5](#_Toc28081489)

[Load word 5](#_Toc28081490)

[Store word 6](#_Toc28081491)

[Branch equal 6](#_Toc28081492)

[Jump 6](#_Toc28081493)

[Functions of different control signals: 6](#_Toc28081494)

[Functions of Alu Controller: 7](#_Toc28081495)

[Chapter 3 8](#_Toc28081496)

[Modules: 8](#_Toc28081497)

[Chapter 4 13](#_Toc28081498)

[Implementation 13](#_Toc28081499)

[Chapter 5 19](#_Toc28081500)

[Results 19](#_Toc28081501)

[Chapter 6 21](#_Toc28081502)

[Summary 21](#_Toc28081503)

[References 21](#_Toc28081504)

# Chapter 1

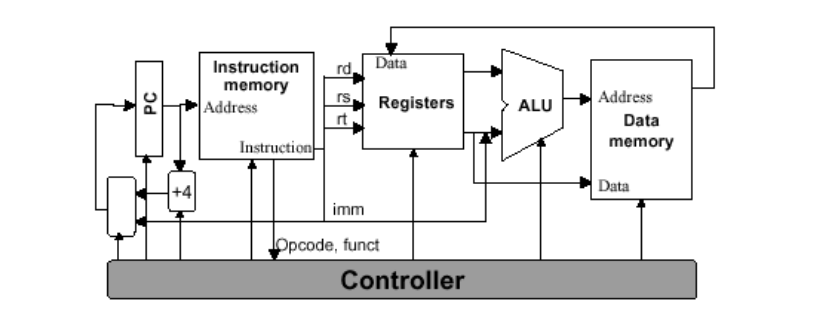
## Introduction

A microprocessor is a [computer processor](https://en.wikipedia.org/wiki/Processor_(computing)) that incorporates the functions of a central processing unit on a single [integrated circuit](https://en.wikipedia.org/wiki/Integrated_circuit) (IC). The microprocessor is a multipurpose, [clock](https://en.wikipedia.org/wiki/Clock_signal) driven, [register](https://en.wikipedia.org/wiki/Processor_register) based, [digital integrated circuit](https://en.wikipedia.org/wiki/Digital_integrated_circuit) that accepts [binary](https://en.wikipedia.org/wiki/Binary_code) data as input, processes it according to [instructions](https://en.wikipedia.org/wiki/Instruction_(computing)) stored in its [memory](https://en.wikipedia.org/wiki/Memory_(computing)) and provides results (also in binary form) as output. Microprocessors contain both [combinational logic](https://en.wikipedia.org/wiki/Combinational_logic) and [sequential digital logic](https://en.wikipedia.org/wiki/Sequential_logic). Microprocessors operate on numbers and symbols represented in the [binary number system](https://en.wikipedia.org/wiki/Binary_number_system). [1]

We had designed the Datapath and control for a 16 bit, single-cycle (nonpipelined) processor. This processor will be able to execute a set of instructions given (similar to MIPS). A Verilog description of microprocessor has been implemented by integrating individual modules.

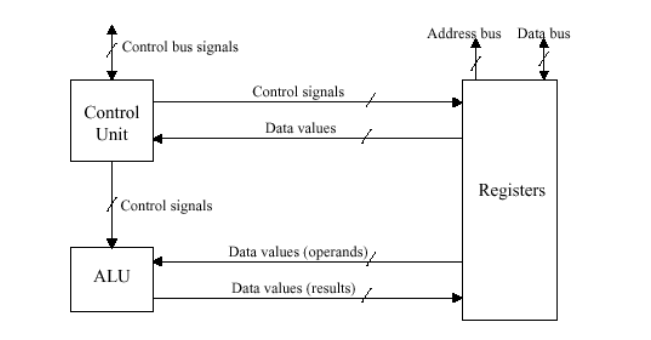
# Chapter 2

## Project description



**Figure1: Schematic diagram** [2]

* Processor (CPU) is the active part of the computer, which does all the work of data manipulation and decision making.
* Datapath is the hardware that performs all the required operations, for example, ALU, registers, and internal buses.
* Control is the hardware that tells the data path what to do, in terms of switching, operation selection, data movement between ALU components, etc.



**Figure2: Schematic diagram of Data** [2]

The implemented processor processes five type of commands differentiated with 3-bit opcode.

### Rformat

Rformat is the type of instruction that takes 2 registers and perform function according to the given combination of bits.

The 16-bit distribution of Rformat instruction is given below.

Opcode

FunF

Sft

Rt

Rs

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

### Load word

Load word takes 16-bit data from data memory to register file destination Rt. Physical address of memory is computed by adding offset and16 bit data stored in register file at address Rs.

The 16-bit distribution of Load word instruction is given below.

Offset

Rt

Opcode

Rs

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

### Store word

store word takes 16-bit data from register file source Rt to data memory. Physical address of memory is computed by adding offset and16 bit data stored in register file at address Rs.

The 16-bit distribution of Store word instruction is given below.

Rt

Offset

Rs

Opcode

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

### Branch equal

Branch equal is conditional jump. The processor will jump to the given target offset is data at Rs and Rt is equal.

The 16-bit distribution of Load word instruction is given below.

Rt

Target

Rs

Opcode

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

### Jump

Jump format is unconditional jump. The processor will jump to the given target offset.

The 16-bit distribution of jump instruction is given below.

Target Instruction

Opcode

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### Functions of different control signals:

|  |  |
| --- | --- |
| **Opcode** | **Instruction Format** |
| 000 | Rformat |
| 001 | LW |
| 010 | SW |
| 011 | Beq |
| 100 | Jmp |
| 1xx | Rformat |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Signals** | **Rformat** | **LW** | **SW** | **Beq** | **Jmp** |
| **J** | 0 | 0 | 0 | 0 | 1 |
| **Beq** | 0 | 0 | 0 | 1 | 0 |
| **RegWrite** | 1 | 1 | 0 | 0 | 0 |
| **MemRead** | 0 | 1 | 0 | 0 | 0 |
| **MemWrite** | 0 | 0 | 1 | 0 | 0 |
| **MtoR** | 0 | 1 | 1 | 0 | 0 |
| **AluSource** | 0 | 1 | 1 | 0 | 0 |
| **AluOp** | 00 | 01 | 01 | 10 | 11 |

### Functions of Alu Controller:

|  |  |  |
| --- | --- | --- |
| **AluControl** | **Output** | **Instructions** |
| **00** | FunctionField | Rformat |
| **01** | Add | LW,SW |
| **10** | Sub | Beq |
| **11** | Xx | Xx |

# Chapter 3

## Modules:

#### Mux4to1:

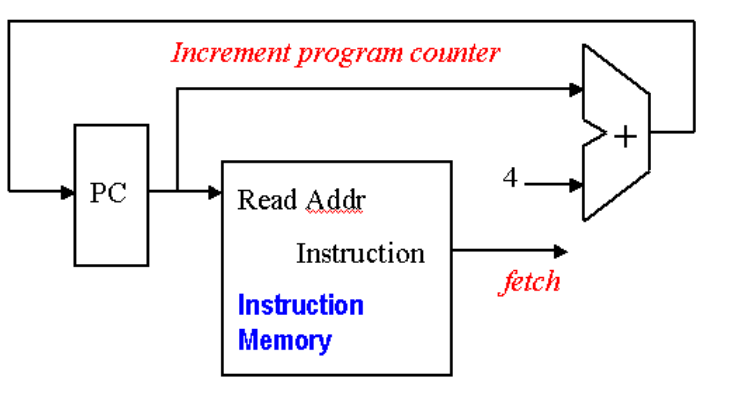
Mux 4 to 1 takes 4 16-bit inputs and using selection lines from control unit gives required output to the instruction memory.

#### 16bitAdder

16-bit adder adds two 16-bit inputs and give required result.

#### Instruction Memory

Instruction memory takes input from program counter which points to the next instruction with the posedge of the clock. It is 16\*16-bit ROM.



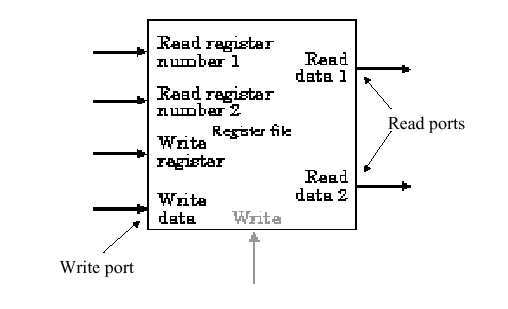
**Figure3: Increment program counter diagram** [3]

#### Control Unit

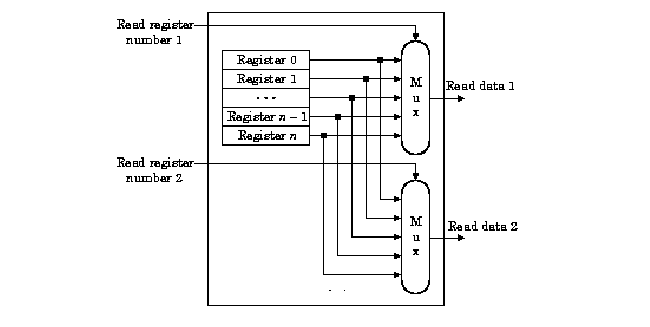
Controls whole circuit according to the 3-bit opcode in instruction memory.

#### Register File

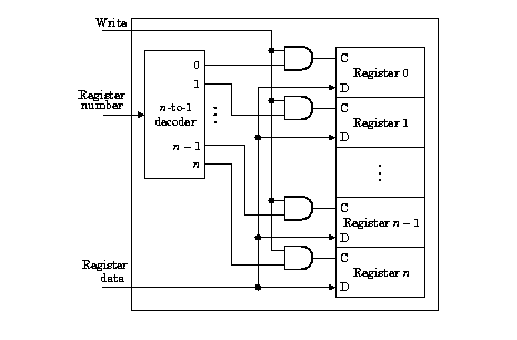
Register file takes 4-bit register input as address and points to the required location.



**Figure4: Register file block diagram**



**Figure5: Register file read data diagram**



**Figure5: Register file write data diagram**

#### Sign Extend 5 to 16

5-bit number is extended by extending most significant bit up to 16-bit number.

#### Mux2to1

2 16-bit inputs are selected according to selection line from control unit.

#### ALU

Alu is operated by AluOp signal from Alu Controller.

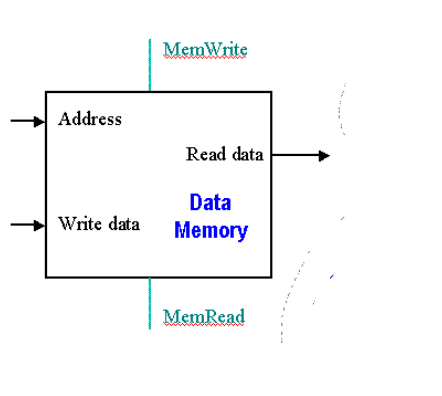
A screenshot of a cell phone

Description automatically generated

**Figure6: Register file to ALU data diagram**

#### Data Memory

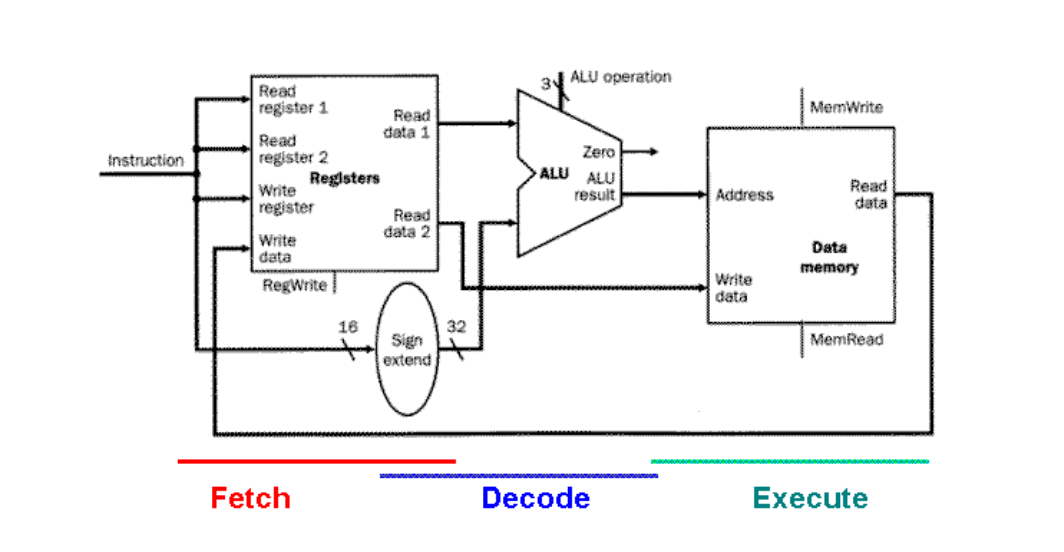
Data Memory is 216-1\*16 RAM. Which is accessed in LW and SW format Instruction.



**Figure7: Data memory block diagram**

#### Sign Ext 13 to 16

Sign of 13-bit number is extended by taking the start 3 bits from the output of program counter in case of jump Instruction.



**Figure8: Register file to ALU to data memory diagram**

A picture containing text, whiteboard

Description automatically generated

**Figure 8: 16-bit single cycle processor**

# Chapter 4

## Implementation

Verilog description of each module is given below.

|  |  |
| --- | --- |
| Mux4to1 | 16bitAdder |
| module Mux4to1(  output reg [15:0] Out,  input [15:0] In0,  input [15:0] In1,  input [15:0] In2,  input [15:0] In3,  input Sel1,  input Sel0  );  wire [1:0]Sel;  assign Sel[1]=Sel1;  assign Sel[0]=Sel0;  //initial Out=16'b0;  always@(Sel)  begin  case(Sel)  2'b00:assign Out=In0;  2'b01:assign Out=In1;  2'b10:assign Out=In2;  2'b11:assign Out=In3;  default:assign Out=16'b0;  endcase  end  endmodule | module Adder(  output reg[15:0] Out,  input [15:0] In  );  always@(In)  assign Out=In+16'h0001;  endmodule |

|  |  |
| --- | --- |
| Instruction Memory | Control Unit |
| module InstructionMem(  output reg [15:0] Instruction,  input [15:0] InstructionAddress  );  initial assign Instruction=16'b000\_0000\_0000\_00\_000;  always@(InstructionAddress)  begin  case(InstructionAddress)    16'h0000:  assign Instruction=16'b000\_1111\_1110\_00\_000;//rformat add  16'h0001:  assign Instruction=16'b000\_1111\_1110\_00\_001;//rformat sub  16'h0002:  assign Instruction=16'b000\_1111\_1110\_00\_011;//rformat and  16'h0003:  assign Instruction=16'b011\_1111\_1110\_00010;//  16'h0004:  assign Instruction=16'b001\_0000\_1000\_01000;//  16'h0005:  assign Instruction=16'b010\_0000\_1000\_01001;//SW  16'h0006:  assign Instruction=16'b100\_0000\_0000\_00000;//Sw    default:assign Instruction=16'b000\_0000\_0000\_00\_000;  endcase  end  endmodule | module ControlUnit(  output reg J,  output reg Beq,  output reg JmpG,  output reg RegWrite,  output reg MemRead,  output reg MemWrite,  output reg MtoR,  output reg AluSource,  output reg [1:0] AluOp,  input [2:0] OpCode  );  parameter Rformat=3'b000,LoadW=3'b001,StoreW=3'b010,BranchEq=3'b011,Jump=3'b100,JmpG1=3'b111;  always@(OpCode)  begin  case(OpCode)  Rformat:  begin  assign J=1'b0;  assign Beq=1'b0;  assign JmpG=1'b0;  assign RegWrite=1'b1;  assign MemRead=1'b0;  assign MemWrite=1'b0;  assign MtoR=1'b0;  assign AluSource=1'b0;  assign AluOp=2'b00;  end  LoadW:  begin  assign J=1'b0;  assign Beq=1'b0;  assign JmpG=1'b0;  assign RegWrite=1'b1;  assign MemRead=1'b1;  assign MemWrite=1'b0;  assign MtoR=1'b1;  assign AluSource=1'b1;  assign AluOp=2'b01;//Add  end  StoreW:  begin  assign J=1'b0;  assign Beq=1'b0;  assign JmpG=1'b0;  assign RegWrite=1'b0;  assign MemRead=1'b0;  assign MemWrite=1'b1;  assign MtoR=1'b1;  assign AluSource=1'b1;  assign AluOp=2'b01;//Add  end  BranchEq:  begin  assign J=1'b0;  assign Beq=1'b1;  assign JmpG=1'b0;  assign RegWrite=1'b0;  assign MemRead=1'b0;  assign MemWrite=1'b0;  assign MtoR=1'b0;  assign AluSource=1'b0;  assign AluOp=2'b10;//Sub  end  Jump:  begin  assign J=1'b1;  assign Beq=1'b0;  assign JmpG=1'b0;  assign RegWrite=1'b0;  assign MemRead=1'b0;  assign MemWrite=1'b0;  assign MtoR=1'b0;  assign AluSource=1'b0;  assign AluOp=2'b11;//nothing  end  JmpG1:  begin  assign J=1'b0;  assign Beq=1'b0;  assign JmpG=1'b1;  assign RegWrite=1'b0;  assign MemRead=1'b0;  assign MemWrite=1'b0;  assign MtoR=1'b0;  assign AluSource=1'b0;  assign AluOp=2'b11;//Sub  end  default:  begin  assign J=1'b0;  assign Beq=1'b0;  assign JmpG=1'b0;  assign RegWrite=1'b0;  assign MemRead=1'b0;  assign MemWrite=1'b0;  assign MtoR=1'b0;//0  assign AluSource=1'b0;  assign AluOp=2'b00;//nothing  end  endcase  end  endmodule |

|  |  |
| --- | --- |
| **Register** File | Sign Extend 5 to 16 |
| module RegisterFile(  output [15:0] R1,  output [15:0] R2,  input [3:0] Rs,  input [3:0] Rt,  input [15:0] DataIn,  input RegWrite,  input reset ,input clk );  reg [15:0]Rmem[0:15];  integer i;  if (reset)  begin  for(i=0;i<14;i=i+1)  Rmem[i]<=16'h0000;  Rmem[14]<=16'b1111\_1111\_1111\_1111;  Rmem[15]<=16'b1111\_1111\_1111\_1111; end  always @( RegWrite,negedge clk)  begin  if(RegWrite)  #50 Rmem[Rt]=DataIn;  end  assign R1=Rmem[Rs];  assign R2=Rmem[Rt];  endmodule | module SignExt5to16(  output [15:0] Out,  input [4:0] In  );  reg [10:0]Temp;  always@(In)  begin  if (In[4]==1'b1)  assign Temp=11'b11111111111;  else if (In[4]==1'b0)  assign Temp=11'b00000000000;  end  assign Out[15:5]=Temp;  assign Out[4:0]=In;  endmodule |

|  |  |
| --- | --- |
| Mux2to1 | ALU Control |
| module Mux2to1(  output reg [15:0] Out,  input [15:0] In0,  input [15:0] In1,  input Sel  );  //initial assign Out=16'b0;  always@(Sel)  begin  case(Sel)  1'b0:assign Out=In0;  1'b1: assign Out=In1;  //default:assign Out=16'b0;  endcase  end  endmodule | module AluControl(  output reg [2:0] AluCon,  input [1:0] AluOp,  input [2:0] FunF  );  always @(AluOp,FunF)  begin  case (AluOp)  2'b00: AluCon[2:0]=FunF[2:0];  2'b01: AluCon[2:0]=3'b000;//LW&SW  2'b10: AluCon[2:0]=3'b001;//Beq  2'b11: AluCon[2:0]=3'b111;//nothing  default: AluCon[2:0]=3'b111;//nothing  endcase  end  endmodule |

|  |  |
| --- | --- |
| Data Memory | Sign Ext 13 to 16 |
| module DataMem(  output reg [15:0] DataOut,  input [15:0] DataIn,  input [15:0] BaseAdd,  input ReadMem,  input WriteMem,  input reset,  input clk,  output [15:0] W8,  output [15:0] W9,  output [15:0] W10,  output [15:0] W11  );  reg [15:0]Mem[0:65535];  integer i;  always@(reset)  begin  if (reset)  begin  for(i=0;i<65536;i=i+1)  if(i==8|i==9|i==10|i==11)  begin Mem[8]=16'hFFFF;  Mem[9]=16'h000F;  Mem[10]=16'h00FF;  Mem[11]=16'h0FFF;  end  else  Mem[i]=16'h0000;  DataOut=16'h0000;  end  end  always @(posedge clk,BaseAdd)  begin  if (ReadMem)  DataOut=Mem[BaseAdd];  if (WriteMem==1)  begin  Mem[BaseAdd]=DataIn;  // DataOut=16'h0000;  end  end  assign W8=Mem[8];  assign W9=Mem[9];  assign W10=Mem[10];  assign W11=Mem[11];  endmodule | module SignExt(  output [15:0] Out,  input [12:0] OffsetJ,  input [2:0] StartPC  );    assign Out[12:0]=OffsetJ[12:0];  assign Out[15:13]=StartPC;  endmodule |

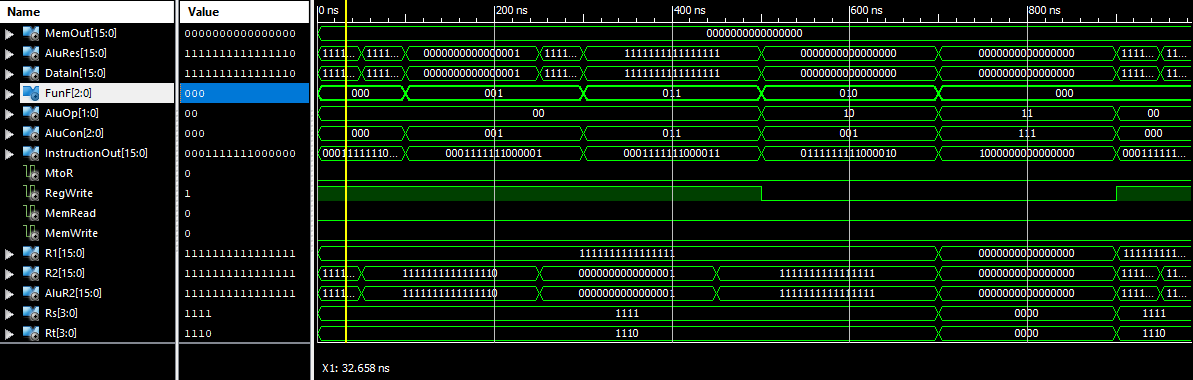
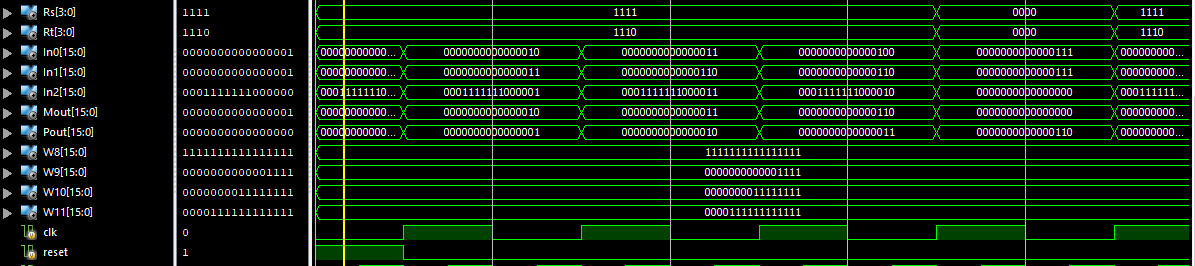
|  |
| --- |
| Alu |
| module ALU(  output reg [15:0] DataOut,  output reg Zero,  output reg AG,  input [15:0] A,  input [15:0] B,  input [2:0] AluC  );  initial begin assign DataOut=16'b0;  Zero=0;  AG=0;  end  always@(AluC)  begin  case (AluC)    3'b000:assign DataOut=A+B;  3'b001: assign DataOut=A-B;  3'b010: assign DataOut=A&B;  3'b011: assign DataOut=A|B;  3'b100: assign DataOut=A^B;  3'b101: assign DataOut=A+B;  3'b110: assign DataOut=A+B;  3'b111: assign DataOut=16'h0000;  default assign DataOut=16'h0000;  endcase  if(DataOut==0)  assign Zero=1;  else assign Zero=0;  if (A[15:0]>B[15:0])  assign AG=1;  else AG=0;  end  endmodule |

|  |
| --- |
| Main |
| module Main(  output [15:0] MemOut,  output [15:0] AluRes,  input clk,  input reset,  output [15:0]DataIn,  output [2:0]FunF,  output [1:0]AluOp,  output [2:0]AluCon,  output [15:0]InstructionOut,  output MtoR,  output RegWrite,  output MemRead,  output MemWrite,  output [15:0]R1,  output [15:0]R2,  output [15:0]AluR2,  output [3:0]Rs,  output [3:0]Rt,  output [15:0]In0,  output [15:0]In1,  output [15:0]In2,  output [15:0]Mout,  output reg [15:0]Pout,  output [15:0] W8,  output [15:0] W9,  output [15:0] W10,  output [15:0] W11,  input clk1    );  wire J;  wire Beq;  wire JmpG;  wire AluSource;  wire [15:0]ExtendedOff;  wire Zero,AG;  wire [12:0]OffJ;    //Program Counter  Mux4to1 Mux1(Mout,In0,In1,In2,In2,Sel1,Sel0);  always@(posedge clk, reset)  begin  if (reset)  Pout=16'h0000;  else  Pout=Mout;  end  Adder Adder1(In0,Pout);  InstructionMem Instruction1(InstructionOut,Pout);  ControlUnit Control1(J,Beq,JmpG,RegWrite,MemRead,MemWrite,MtoR,AluSource,AluOp,InstructionOut[15:13]);  assign Rs=InstructionOut[12:9];  assign Rt=InstructionOut[8:5];  RegisterFile RegF1(R1,R2,Rs,Rt,DataIn,RegWrite,reset,clk);  SignExt5to16 Sign1(ExtendedOff,InstructionOut[4:0]);  Mux2to1 Mux2(AluR2,R2,ExtendedOff,AluSource);  assign FunF=InstructionOut[2:0];  AluControl AluCon1(AluCon,AluOp,FunF);  ALU ALU1(AluRes,Zero,AG,R1,AluR2,AluCon);  DataMem DataMem1(MemOut,R2,AluRes,MemRead,MemWrite,reset,clk,W8,W9,W10,W11);  Mux2to1 Mux3(DataIn,AluRes,MemOut,MtoR);  AddOffB Addoff1(In1,ExtendedOff,In0);  assign OffJ=InstructionOut[12:0];  SignExt Sign2(In2,OffJ,In0[15:13]);  assign Sel1=J;  assign Sel0= Beq&Zero;  //wire S1;  //assign S1=JmpG&&AG;  //assign Sel1=J|S1;  //assign Sel0= Beq&Zero|S1;  endmodule |

# Chapter 5

## Results

|  |
| --- |
| Test Bench |
| module t\_new;  // Inputs  reg clk;  reg reset;  // Outputs  wire [15:0] MemOut;  wire [15:0] AluRes;  wire [15:0]DataIn;  wire [2:0]FunF;  wire [1:0]AluOp;  wire [2:0] AluCon;  wire [15:0] InstructionOut;  wire MtoR;  wire RegWrite;  wire MemRead;  wire MemWrite;  wire [15:0] R1;  wire [15:0] R2;  wire [15:0]AluR2;  wire [3:0] Rs;  wire [3:0] Rt;  wire [15:0] In0;  wire [15:0] In1;    wire [15:0] In2;  wire [15:0] Mout;  wire [15:0] Pout;    wire [15:0] W8;  wire [15:0] W9;  wire [15:0] W10;  wire [15:0] W11;  // Instantiate the Unit Under Test (UUT  Main uut (  .MemOut(MemOut),  .AluRes(AluRes),  .clk(clk),  .reset(reset),  .DataIn(DataIn),  .FunF(FunF),  .AluOp(AluOp),  .AluCon(AluCon),  .InstructionOut(InstructionOut),  .MtoR(MtoR),  .RegWrite(RegWrite),  .MemRead(MemRead),  .MemWrite(MemWrite),  .R1(R1),  .R2(R2),  .AluR2(AluR2),  .Rs(Rs),  .Rt(Rt),  .In0(In0),  .In1(In1),  .In2(In2),  .Mout(Mout),  .Pout(Pout),  .W8(W8),  .W9(W9),  .W10(W10),  .W11(W11),  .clk1(clk1)  );  initial begin  clk1=0;  forever #50 clk1=~clk1;  end  initial begin  clk=0;  forever #100 clk=~clk;  end  initial begin  // Initialize Input  // Wait 100 ns for global reset to finish  reset=1;  // Add stimulus here  #100  reset=0; end    endmodule |

**Figure 9: Timing diagram of 16-bit processor**

# Chapter 6

## Summary

A data path contains all the functional units and connections necessary to implement an instruction set architecture. For our single-cycle implementation, we use two separate memories, an ALU, some extra adders, and lots of multiplexers. Our processor is a 16-bit machine, so most of the buses are 16-bits wide. Control unit tells the data path what to do, based on the instruction that’s currently being executed.

Performance is one of the most important criteria in judging systems. Our main performance equation explains how performance depends on several factors related to both hardware and software.

A single-cycle CPU has two main disadvantages.

* The cycle time is limited by the worst-case latency.
* It isn’t efficiently using its hardware

# References

|  |  |
| --- | --- |
| [1] | "Microprocessor," wikipedia, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Microprocessor. [Accessed 22 12 2019]. |
| [2] | "CompOrg32bitprocessor," [Online]. Available: https://www.cise.ufl.edu/~mssz/CompOrg/CDA-proc.html. |
| [3] | berkeley, "32bitprocessor," [Online]. Available: https://inst.eecs.berkeley.edu/~cs152/fa06/handouts/lab\_2.html. |