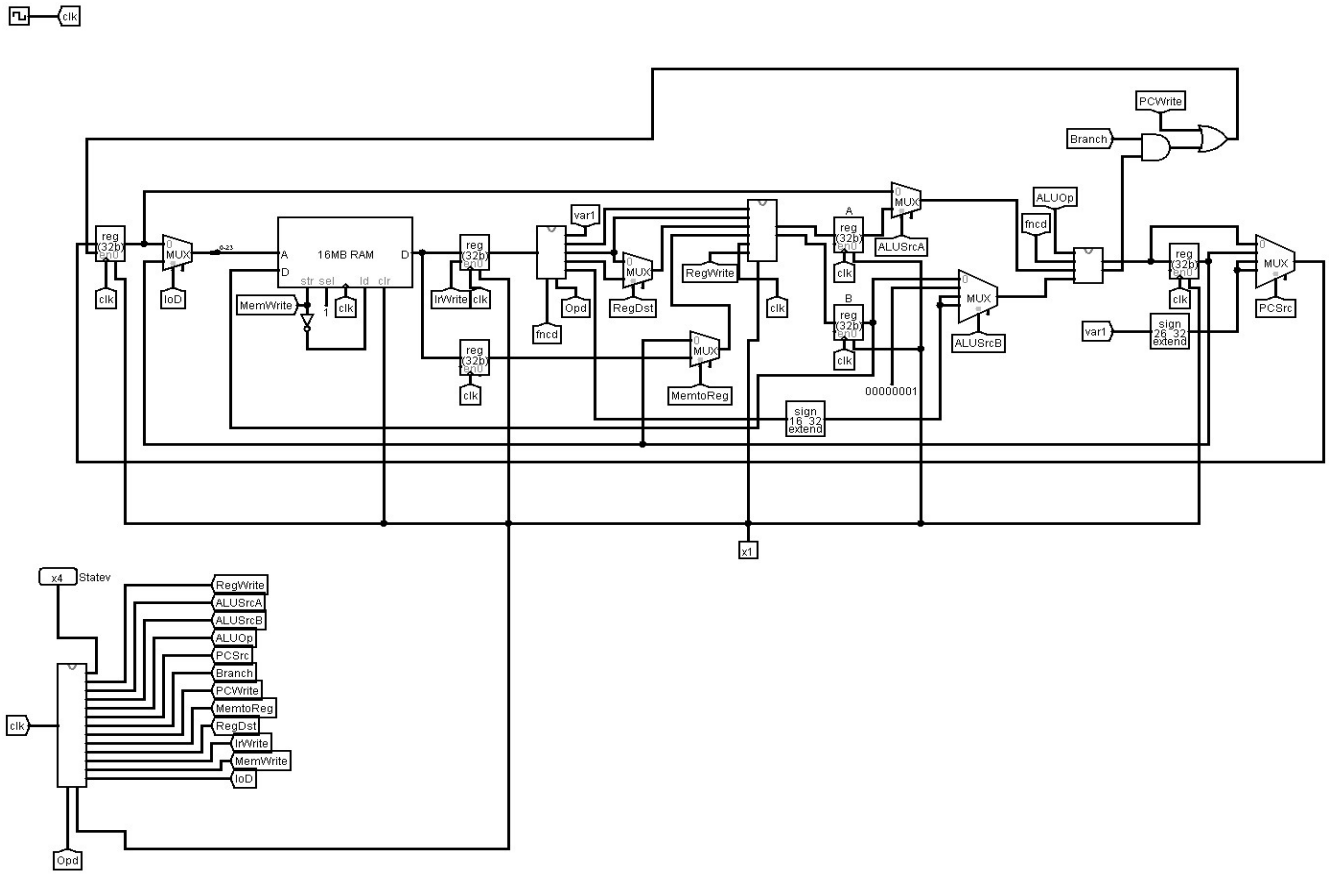
MULTICYCLE MIPS IN VERILOG



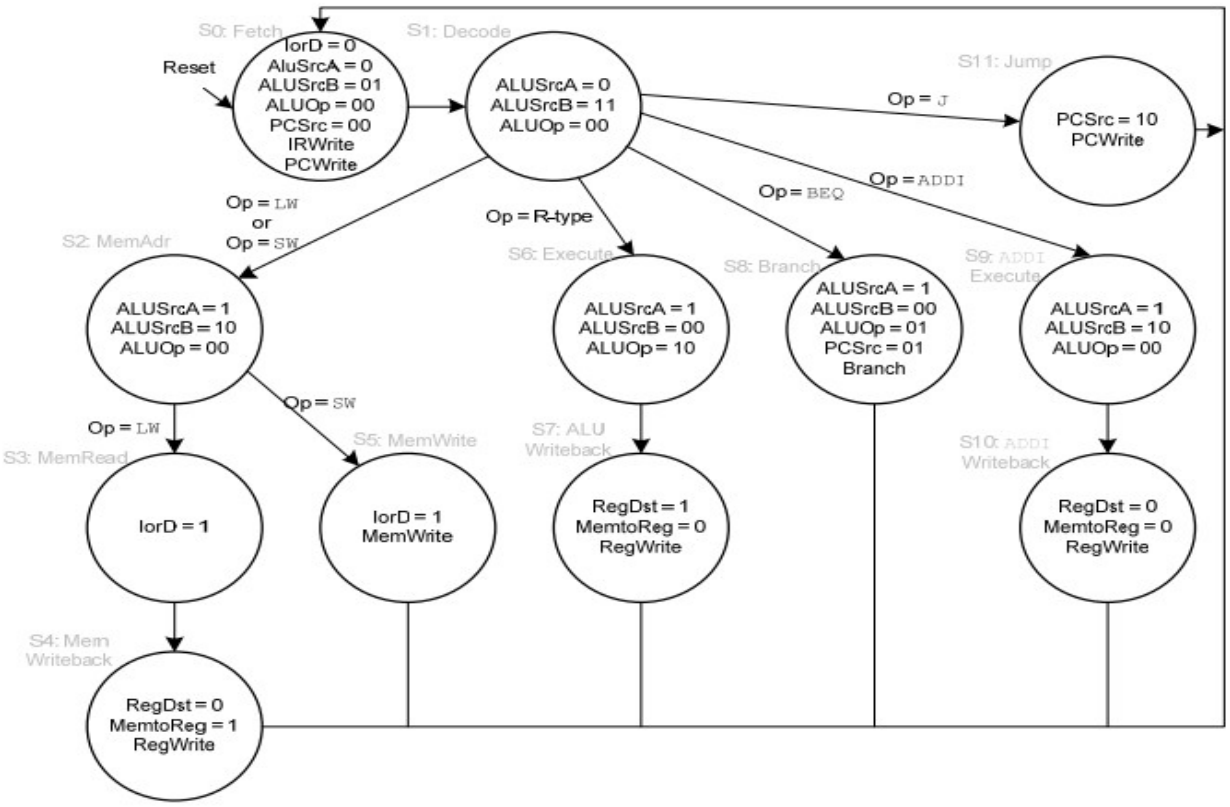
The above diagram is the logicsim representation of the Multi- cycle mips, I have implemented.

mips.v File

```
module mips(clk,reset);

        input clk,reset;
        wire [5:0] Op;
        wire Zero;
        wire IorD;
        wire MemRead;
```

```
        wire MemWrite;
        wire MemToReg;
        wire IRWrite;
        wire [1:0] PCSource;
        wire [1:0] ALUSrcB;
        wire ALUSrcA;
        wire RegWrite;
        wire RegDst;
        wire PCSel;
        wire [1:0] ALUOp;
        wire [3:0] ALUCtrl;
        wire [5:0] Function;

control control_D(clk, reset,Op, Zero, IorD, MemRead, MemWrite, MemtoReg, IRWrite,
PCSource, ALUSrcB, ALUSrcA, RegWrite, RegDst, PCSel, ALUOp);
alucontrol alucontrol_D(ALUOp, Function, ALUCtrl);
datapath  datapath_D(clk, reset,IorD, MemRead, MemWrite, MemtoReg, IRWrite, PCSource,
ALUSrcB, ALUSrcA, RegWrite, RegDst, PCSel, ALUCtrl, Op, Zero, Function);

endmodule
```

FSM



This is the representation of the control unit. The states are numbered as shown above. Showing a partial code of control.v which implements the above state machine.

```
        parameter FETCH = 4'b0000;
        parameter DECODE = 4'b0001;
        parameter MEMADRCOMP = 4'b0010;
        parameter MEMACCESSL = 4'b0011;//L1
        parameter MEMREADEND = 4'b0100;//L2
        parameter MEMACCESSS = 4'b0101;//S
        parameter EXECUTION = 4'b0110;
        parameter RTYPEEND = 4'b0111;
        parameter ADDI_EXECUTE = 4'b1001;
        parameter JUMP = 4'b1011;
        parameter ADDI_END = 4'b1010;
        parameter BEQ = 4'b1000;
```

```
always@(state or Op) begin
        case (state)
    FETCH:  nextstate = DECODE;
    DECODE:  case(Op)
            6'b100011: nextstate = MEMADRCOMP;//lw
            6'b101011: nextstate = MEMADRCOMP;//sw
            6'b000000: nextstate = EXECUTION;//r
            6'b000100: nextstate = BEQ;//beq
            6'b001000: nextstate= ADDI_EXECUTE;//ADDI_execute
            6'b000010: nextstate= JUMP;
            default: nextstate = FETCH;
           endcase
    MEMADRCOMP:  case(Op)
            6'b100011:     nextstate = MEMACCESSL;//lw
            6'b101011:     nextstate = MEMACCESSS;//sw
            default: nextstate = FETCH;
           endcase
    MEMACCESSL:    nextstate = MEMREADEND;
    MEMREADEND:    nextstate = FETCH;
    MEMACCESSS:    nextstate = FETCH;
    EXECUTION: nextstate = RTYPEEND;
    ADDI_EXECUTE: nextstate = ADDI_END;
    ADDI_END: nextstate =FETCH;
    RTYPEEND: nextstate = FETCH;
    BEQ:   nextstate = FETCH;
    JUMP: nextstate = FETCH;
    default: nextstate = FETCH;
  endcase
 end
```

tb_mux.v

```
module tb_mips;
	reg clk;
	reg reset;
	mips mips_DUT(clk,reset);
	initial
		forever #5 clk=~clk;
	initial begin
	clk=0;
	reset=1;
	#10 reset=0;
	#6000 $finish;
	end
endmodule
```

Mem.dat

This file is loaded into the memory of the mips in datapath.v. We must declare instructions in the given format.

@ADDRESS

Content

EX:

@48
0000_0001
@60
0000_0101
0000_0011
0000_0010
0000_0100

MY TESTED INSTRUCTIONS

To add all the numbers from 5 to 1 (Sigma 5)

```
80:	ADDI $t1 $zero 0x0005		// s1 = 5
	ADDI $t2 $zero 0x0001		// t2 = 1
	ADDI $t3 $zero 0x0000		// t3 =0
83:	ADD $t3 $t3 $t1			// t3 = t3+t1
	SUB $t1 $t1 $t2			//t1=t1-t2
	BEQ $t1 $zero 0x0001		// Jump to done on t1= 0
	J 0x0000083			// Jump to location 83
```

Mem.dat for this program

```
@80
20090005
200A0001
200B0000
01695820
012A4822
11200001
08000083
```

Infinite Fibonacci series

```
80:   ADDI $t1 $zero 0x0000      // t1 =0
      ADDI $t2 $zero 0x0001      // t2=1
      ADDI $t3 $zero 0x0000      // t3=0
      ADDI $t4 $zero 0x0000      // t4 =1
84:   ADD $t3 $t2 $t1            // t3 = t2+t1
      sw t3 0x20(t4)             // store t3 at the address t4 + 0x20
      ADD $t1 $t3 $zero          //  t1 = t3
      addi t4 t4 1               //  t4++
      ADD $t3 $t2 $t1            // t3 = t2+t1
      sw t3 0x20(t4)             // store t3 at the address t4 +0x20
      ADD $t2 $t3 $zero          // t2 = t3
      addi t4 t4 1               // t4++
      J 0x0000084                // jump to 0x84
```

Mem.dat

```
@80
20090000
200A0001
200B0000
200C0000
01495820
AD8B0020
01604820
218C0001
01495820
AD8B0020
01605020
218C0001
08000084
```

Sort the numbers decreasing order (Bubble Sort) :

```
80:   addi s7 zero 60
      addi $s0,zero 0          // i
      addi $s6,zero 3          // store s6 as N-1 where n is the size of the array
      addi $s1,zero 0          // j
84:   lw $t0, 60($s1)          //$t0 is A[j]. The data starts at address 60
      lw $t1, 61($s1)          //$t1 is A[j+1]
      slt t3 t0 t1             //  set t3 if t0 is less than t1
      beq t3 zero 0x2          // go to address 90 when equal to skip swap
      sw $t1, 60($s1)          //$t1 is A[j]
      sw $t0, 61($s1)          //$t0 is A[j+1]
90:   addi $s1, $s1, 1         // j++
      sub $s5, $s6, $s0        //$s5 is N-i-1
      beq s1 s5 0x1            // if j reaches n-i-1 jump to next i
      J 0x84
      addi $s0, $s0, 0x1       // i++
      addi $s1, zero,0x0       // j
      beq s0 s6 0x1            // if I = n-1 end
      J 0x84
```

Mem.dat

```
@60
0000_0101
0000_0011
0000_0010
0000_0100
@80 //starting address of instruction memory
20090000
200A0001
200B0000
200C0000
01495820
AD8B0020
01604820
218C0001
01495820
AD8B0020
01605020
218C0001
08000084
```

Sort in ascending order bubble sort:

```
80:   addi s7 zero 60
      addi $s0,zero 0          // i
      addi $s6,zero 3          // store s6 as N-1 where n is the size of the array
```

```
        addi $s1,zero 0          // j
84:     lw $t0, 60($s1)          //$t0 is A[j]. The data starts at address 60
        lw $t1, 61($s1)          //$t1 is A[j+1]
        slt t3 t1 t0             //  set t3 if t1 is less than t0
        beq t3 zero 0x2          // go to address 90 when equal to skip swap
        sw $t1, 60($s1)          //$t1 is A[j]
        sw $t0, 61($s1)          //$t0 is A[j+1]
90:     addi $s1, $s1, 1         // j++
        sub $s5, $s6, $s0        //$s5 is N-i-1
        beq s1 s5 0x1            // if j reaches n-i-1 jump to next i
        J 0x84
        addi $s0, $s0, 0x1       // i++

        addi $s1, zero,0x0       // j

        beq s0 s6 0x1            // if I = n-1 end

        J 0x84
```

Mem.dat

```
@60
0000_0101
0000_0011
0000_0010
0000_0100
@80 //starting address of instruction memory
20170060
20100000
20160003
20110000
8E280060
8E290061
0128582A
11600002
AE290060
AE280061
22310001
02D0A822
12350001
08000084
22100001
20110000
12160001
08000084
```

Reverse an array of size 4 store at address 60:

20100003	addi s0,zero,3
20130002	addi s3,zero,2
20110000	addi s1,zero,0
20140001	addi s4,zero,1
02309020	add s2,s1,s0
8e280060	lw t0,96(s1)
8e490060	lw t1,96(s2)
ae290060	sw t1,96(s1)
ae480060	sw t0,96(s2)
02749822	sub s3,s3,s4
02348820	add s1,s1,s4
02549022	sub s2,s2,s4
12600001	BEQ $s3 $zero 0x0001
08000085	jump to 0x85

Mem.dat
@60
0000_0101
0000_0011
0000_0010
0000_0100
@80
20100003
20130002
20110000
20140001
02309020
8E280060
8E490060
AE290060
AE480060
02749822
02348820
02549022
12600001
08000085

Presented By

v. sai sujeeth