1. INTRODUCTION

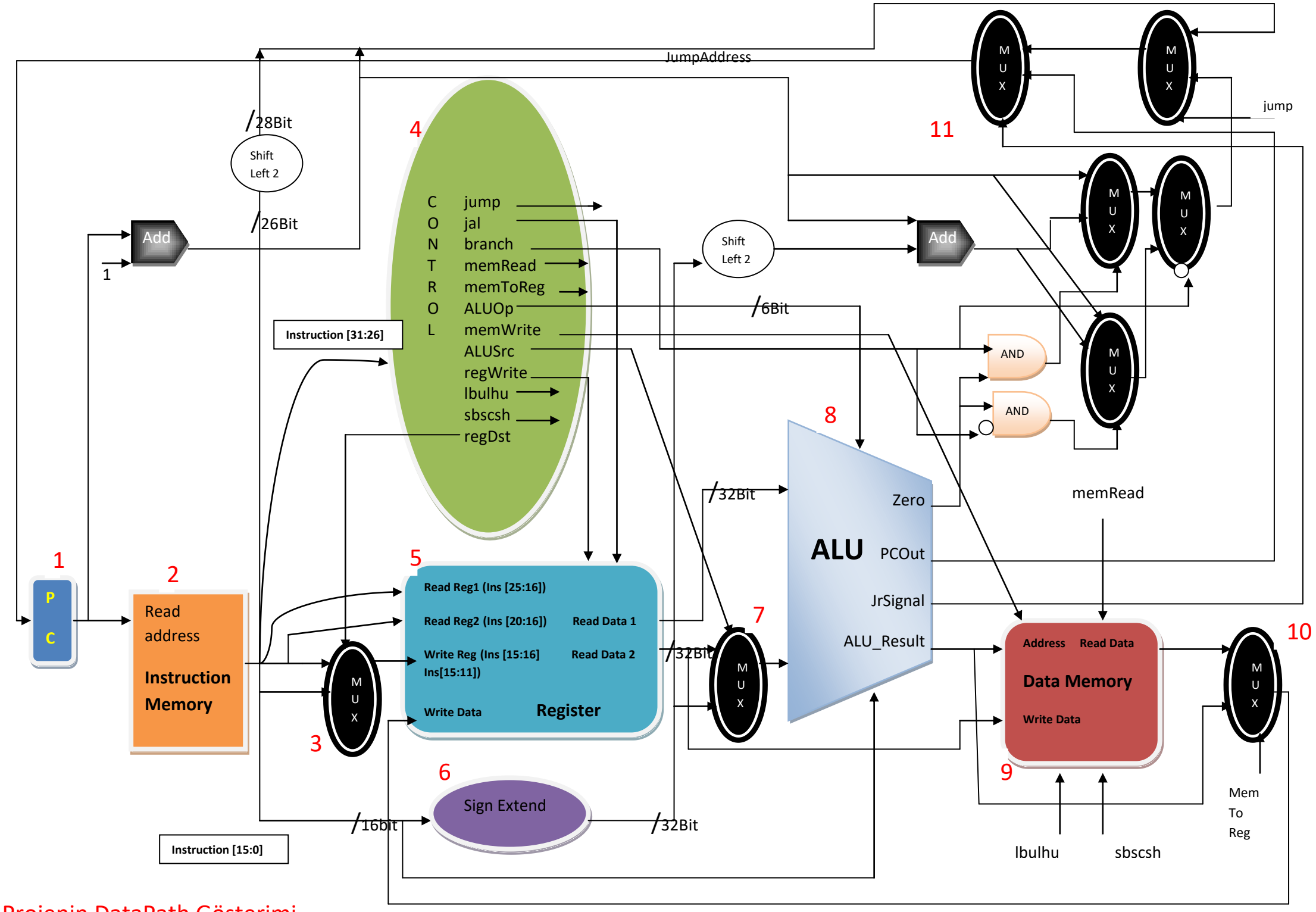
Verilog ile R-Type, I-Type ve J-Type instructionlarının bir kısmını çalıştıracak şekilde tasarlanmış olan bu projede Resim 1.1 Instruction Listesinde belirtilenler implement edilmiştir.

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		0 / 08_{hex}
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27_{hex}
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25_{hex}
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / $2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00_{hex}
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		0 / 02_{hex}
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22_{hex}
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23_{hex}

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

Resim 1.1 Instruction Listesi



Projenin DataPath Gösterimi

1.1 DataPath View of Project

Yukarıda belirtilen DataPath modelinde Verilog programında yazılan modüllerin nasıl bir birine bağlandığı şekillerle gösterilmektedir. Ayrıca yanlarında belirtilen "kırmızı renkli sayılar" aşağıda belirtilecektir ve daha sonra açıklamalı olarak anlatılacaktır.

1) PC: mips_core.v dosyası içerisinde modül olmadan yazılmıştır ve clock sinyaline bağlı çalışır.
2) Instruction Memory: mips_instr_mem.v dosyasında modül olarak yazılmıştır.
3) MUX: mux_2x1_out5bit.v dosyası içerisinde modül olarak yazılmıştır. 5bitlik verileri alıp seçim sonucu 5bitlik veriyi output olarak vermektedir.
4) Control: control.v dosyasında modül olarak yazılmıştır.
5) Register: mips_registers.v dosyası içerisinde modül olarak yazılmıştır.
6) Sign Extend: signExtend.v dosyası içerisinde modül olarak yazılmıştır.

7-10-11) MUX: mux_2x1_out32bit.v dosyası içerisinde modül olarak yazılmıştır. 32bitlik verileri alıp seçim sonucu 32bitlik veriyi output olarak vermektedir.

8) ALU: ALU.v dosyası içerisinde modül olarak yazılmıştır ve artirmatik işlemleri yapmaktadır.
9) Data Memory: mips_data_mem.v dosyası içerisinde modül olarak yazılmıştır.

1.2 Life Cycle of 1 Instruction

Not: Kendi yaptığım DataPath e göre çalışma şekilleri belirtilmektedir.

R-Type: PC -> Instruction Memory -> [(opCode ->CONTROL) + (RS-RT-RD-> Registers) + (immediate -> Sign Extend)] -> (RT -> ALU) -> Registers

I-Type:

LW – LB - LH : PC -> Instruction Memory -> [(opCode ->CONTROL) + (RS-RT-RD-> Registers) + (immediate -> Sign Extend)] -> (SignExtend -> ALU) -> Data Memory -> Registers

SW – SB – SH : PC -> Instruction Memory -> [(opCode ->CONTROL) + (RS-RT-RD-> Registers) + (immediate -> Sign Extend)] -> (SignExtend -> ALU) -> Data Memory

BEQ – BNE : PC -> Instruction Memory -> [(opCode ->CONTROL) + (RS-RT-RD-> Registers) + (immediate -> Sign Extend)] -> (SignExtend -> ALU(zero)) -> (BranchAddress-PC+1) -> PC

OTHER I-TYPE INSTRUCTIONS : PC -> Instruction Memory -> [(opCode ->CONTROL) + (RS-RT-RD-> Registers) + (immediate -> Sign Extend)] -> (SignExtend -> ALU) -> Registers

J-Type:

JUMP: PC -> Instruction Memory -> (opCode -> CONTROL) -> JUMPADDRESS -> PC

JAL: PC -> Instruction Memory -> (opCode -> CONTROL) -> ALU -> [(JUMPADDRESS -> PC) + (PC+2 -> Register[31])]

2. METHOD

Instruction Memory (mips_instr_mem.v)

input [31:0] program_counter;

output [31:0] instruction;

".\\instruction.mem" dosyasını okur ve instr_mem arrayine aktarır. Daha sonra hangi instructionın çalışacağını program_counter a göre belirler. instr_mem arrayinden belirlenen instruction değeri Output olan instruction modül dışına aktarılır.

MUX (mux_2x1_out5bit.v)

output reg[4:0] out;

input [4:0] rd,rt;

input select;

Input olarak gelen 5bitlik rd ya da rt değeri select değerine göre seçme işlemi yapar ve out a aktarılır. Böylece modül dışına aktarılmış olur.

MUX (mux_2x1_out32bit.v)

output reg[31:0] out;

input [31:0] val1, val2;

input select;

Input olarak gelen 32bitlik val1 ya da val2 değeri select değerine göre seçme işlemi yapar ve out a aktarılır. Böylece modül dışına aktarılmış olur. Bu modül DataPath üzerinde gösterilen 3 nolu MUX haricinde bulunan tüm MUXlarda kullanılmaktır.

Sign Extend (signExtend.v)

output reg[31:0] signExtOut;

input [15:0] immediate;

Input olarak alınan immediate değerinin 16. bitine göre [31:16] bit arasına o değerler aktarılır ve 32bite tamamlanır.

CONTROL (control.v)

input[5:0] opCode;
output reg[5:0] ALUOp;
output reg regDst, jump, jal, branch, memRead, memToReg, memWrite, ALUSrc, regWrite;
output reg [1:0] lbulhu, sbscsh;

SİGNALS	R-Type	Other I-Type	LW-LL	LBU	LHU	SW	SB	SH	BEQ - BNE	JUMP	JAL
regDst	1	0	0	0	0	0	0	0	0	0	0
jump	0	0	0	0	0	0	0	0	0	1	0
jal	0	0	0	0	0	0	0	0	0	0	1
branch	0	0	0	0	0	0	0	0	1	0	0
memRead	0	0	1	1	1	0	0	0	0	0	0
memToReg	0	0	1	1	1	0	0	0	0	0	0
regWrite	1	1	1	1	1	0	0	0	0	0	0
memWrite	0	0	0	0	0	1	1	1	0	0	0
ALUSrc	0	1	1	1	1	1	1	1	0	0	0
lbulhu	00	00	00	01	10	00	00	00	00	00	00
sbscsh	00	00	00	00	00	10	01	11	00	00	00

CONTROL modülü opCode değerini alıp instructionlara signal üretmektedir. Ayrıca yapmış olduğum DataPath de ALUOp çıkışı direkt olarak opCode dur 6bitliktir.

Registers (mips_registers.v)

output reg [31:0] read_data_1, read_data_2;
input [31:0] write_data;
input [4:0] read_reg_1, read_reg_2, write_reg;
input signal_reg_write, clk, jalSignal;
input [31:0]jalPc;

".\\registers.mem" dosyasından okunan verileri registers arraye aktarır ve rs ve rt değerlerinin contentlerini output olarak verir. Ayrıca R-Type ve other I-Type lar için register arrayine yazma işlemleri vardır. Ayrıca ”jal” instructionı çalıştığında register[31] a PC değerini aktarmaktadır.

ALU (ALU.v)

output reg [31:0] ALU_result;
output reg Zero;
output reg [31:0]PCOut;
output reg jrSignal;
input [31:0] rs, rtOrSingExtend;
input [15:0] immediate;
input [5:0] ALUop;

Input olarak alınan değerlerden ALUop değeri ile ile hangi instruction olduğu belirlenir ve aritmetik işlemler yapılarak Outputlara sonuçlar verilir.

Data Memory (mips_data_mem.v)

output reg[31:0] read_data;
input [31:0] mem_address;
input [31:0] write_data;
input sig_mem_read;
input sig_mem_write;
input [1:0]lbulhu,sbscsh;

".\\data.mem" dosyasından okunan verileri data_mem arrayine aktarır ve sig_mem_read ve sig_mem_write sinyallerine göre "memorye yazılacak mı?" yoksa "memoryden okunacak mı ? " işlemleri belirlenir. Okuma işlemi yapıldığı taktirde read_data Outputu ile modül dışına aktarılır.

Instruction Memory (mips_instr_mem.v)

input clock;
$monitor("instruction: %32b\nread_data_1: %32b\nread_data_2: %32b\nwrite_data : %32b\nsingextend : %32b\nPC : %d\n",instruction,read_data_1,read_data_2,write_data,signExtOut,PC);

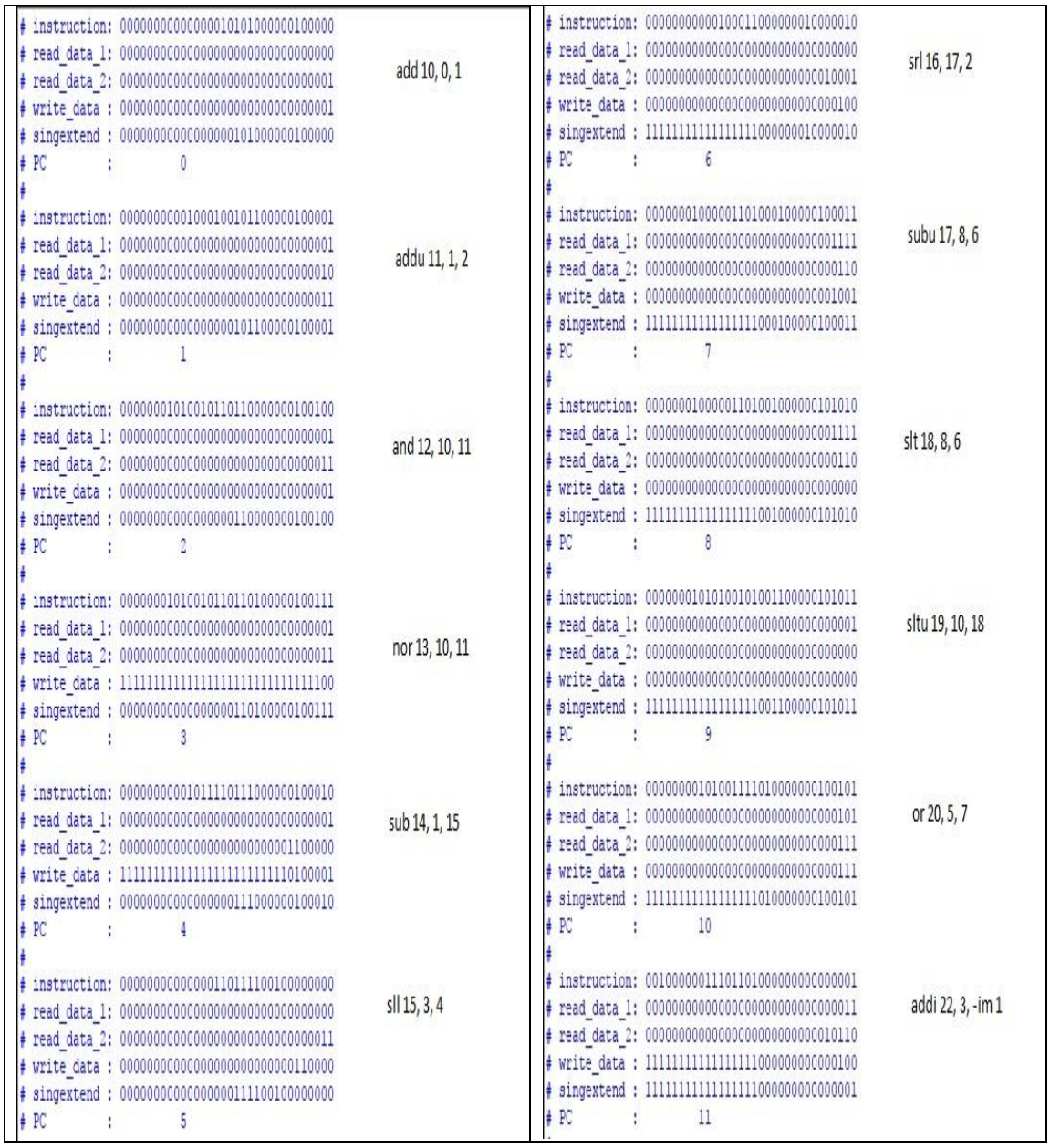
Bütün modüllerin bir arada çağrılarak tutulduğu modüldür. Sadece Clock sinyalini alır. Bütün modüllere işlemler gitmeye başlar CONTROL' ünden çıkan sinyal değerleriyle birlikte tüm modüllerin tam olarak ne yapacağı belirlenmiş olur. Sonuç olarak "monitor" işlemiyle ekrana çıktı aktarılmış olur. Bu çıktıyı görmenin yöntemi "ModelSIM" programını çalıştırmaktır.

3. METHOD

Testbench Results

Aşağıda bulunan Testbench çıktılarının hepsi mevcut değildir ModelSIM de denendiği taktirde tüm sonuçlar görünecektir. PC değiştiği yerlerden 25 – 28 jump, 35 – 40 jal, 45 – 49 beq, 60 – 64 bne, 70 – 2 jr işlemlerini yapmaktadır.

NOT: “res_register.mem” ve “res_data.mem” dosyalarına erişim vardır proje klasörü \ simulation\modelsim altında mevcuttur. İçeriği değişmektedir.



```
# instruction: 00100100100101110000000000000010
# read_data_1: 00000000000000000000000000000100
# read_data_2: 00000000000000000000000000010111
# write_data : 00000000000000000000000000000110
# singextend : 00000000000000000000000000000010
# PC         :         12
#
# instruction: 00110000101110000000000000000100
# read_data_1: 00000000000000000000000000000101
# read_data_2: 00000000000000000000000000011000
# write_data : 00000000000000000000000000000100
# singextend : 00000000000000000000000000000100
# PC         :         13
#
# instruction: 10010001111111001000000000000010
# read_data_1: 00000000000000000000000000110000
# read_data_2: 00000000000000000000000000011001
# write_data : 00000000000000000000000000010010
# singextend : 00000000000000000000000000000010
# PC         :         14
#
# instruction: 10010101111111010000000000000011
# read_data_1: 00000000000000000000000000110000
# read_data_2: 00000000000000000000000000011010
# write_data : 00000000000000000000000000010011
# singextend : 00000000000000000000000000000011
# PC         :         15
#
# instruction: 11000000010110110000000000100010
# read_data_1: 00000000000000000000000000000010
# read_data_2: 00000000000000000000000000011011
# write_data : 00000000000000000000000000000100
# singextend : 00000000000000000000000000100010
# PC         :         16
#
# instruction: 00111100000111000000000000100000
# read_data_1: 00000000001000000000000000000000
# read_data_2: 00000000000000000000000000011100
# write_data : 00000000001000000000000000000000
# singextend : 00000000000000000000000000100000
# PC         :         17
```

addiu 23, 4, im 2

andi 24, 5, zero

lbu

lhu

ll

lui

```
# instruction: 10001100010111010000000000100000
# read_data_1: 00000000000000000000000000000010
# read_data_2: 00000000000000000000000000011101
# write_data : 00000000000000000000000000000010
# singextend : 00000000000000000000000000100000
# PC         :         18
#
# instruction: 00110100101111100000000000100000
# read_data_1: 00000000000000000000000000000101
# read_data_2: 00000000000000000000000000011110
# write_data : 00000000000000000000000000100101
# singextend : 00000000000000000000000000100000
# PC         :         19
#
# instruction: 00101000101000000000000000001110
# read_data_1: 00000000000000000000000000000101
# read_data_2: 00000000000000000000000000000001
# write_data : 00000000000000000000000000000001
# singextend : 00000000000000000000000000001110
# PC         :         20
#
# instruction: 00101100101000010000000000001110
# read_data_1: 00000000000000000000000000000101
# read_data_2: 00000000000000000000000000000001
# write_data : 00000000000000000000000000000001
# singextend : 00000000000000000000000000001110
# PC         :         21
#
# instruction: 10100000111000110000000000000100
# read_data_1: 00000000000000000000000000000111
# read_data_2: 00000000000000000000000000000011
# write_data : 00000000000000000000000000001011
# singextend : 00000000000000000000000000000100
# PC         :         22
#
# instruction: 10100100111000110000000000101010
# read_data_1: 00000000000000000000000000000111
# read_data_2: 00000000000000000000000000000011
# write_data : 00000000000000000000000000110001
# singextend : 00000000000000000000000000101010
# PC         :         23
```

```
# instruction: 10101100111000110000000000010000
# read_data_1: 00000000000000000000000000000111
# read_data_2: 00000000000000000000000000000011
# write_data : 00000000000000000000000000010111
# singextend : 00000000000000000000000000010000
# PC         :        24
#
# instruction: 00001000000000000000000000000111
# read_data_1: 00000000000000000000000000001001
# read_data_2: 00000000000000000000000000001001
# write_data : 00000000000000000000000000010111
# singextend : 00000000000000000000000000000111
# PC         :        25
#
# instruction: 00000001010010110110000000100100
# read_data_1: 00000000000000000000000000000001
# read_data_2: 00000000000000000000000000000011
# write_data : 00000000000000000000000000000001
# singextend : 00000000000000000110000000100100
# PC         :        28
#
# instruction: 00000001010010110110100000100111
# read_data_1: 00000000000000000000000000000001
# read_data_2: 00000000000000000000000000000011
# write_data : 11111111111111111111111111111100
# singextend : 00000000000000000110100000100111
# PC         :        29
#
# instruction: 00000000001011110111000000100010
# read_data_1: 00000000000000000000000000000001
# read_data_2: 00000000000000000000000000110000
# write_data : 11111111111111111111111111010001
# singextend : 00000000000000000111000000100010
# PC         :        30
#
# instruction: 00000000000000011011110010000000
# read_data_1: 00000000000000000000000000001001
# read_data_2: 00000000000000000000000000000011
# write_data : 00000000000000000000000000110000
# singextend : 00000000000000000111100100000000
# PC         :        31
```

```
# instruction: 00000000000010001100000010000010
# read_data_1: 00000000000000000000000000001001
# read_data_2: 00000000000000000000000000001001
# write_data : 00000000000000000000000000000010
# singextend : 11111111111111111000000010000010
# PC         :        32
#
# instruction: 00000001000001101000100000100011
# read_data_1: 00000000000000000000000000001111
# read_data_2: 00000000000000000000000000000110
# write_data : 00000000000000000000000000001001
# singextend : 11111111111111111000100000100011
# PC         :        33
#
# instruction: 00000001000001101001000000101010
# read_data_1: 00000000000000000000000000001111
# read_data_2: 00000000000000000000000000000110
# write_data : 00000000000000000000000000000000
# singextend : 11111111111111111001000000101010
# PC         :        34
#
# instruction: 00001100000000000000000000001010
# read_data_1: 00000000000000000000000000000000
# read_data_2: 00000000000000000000000000000000
# write_data : 00000000000000000000000000000000
# singextend : 00000000000000000000000000001010
# PC         :        35
#
# instruction: 00000000001011110111000000100010
# read_data_1: 00000000000000000000000000000001
# read_data_2: 00000000000000000000000000110000
# write_data : 11111111111111111111111111010001
# singextend : 00000000000000000111000000100010
# PC         :        40
#
# instruction: 00000000000000110111100100000000
# read_data_1: 00000000000000000000000000000000
# read_data_2: 00000000000000000000000000000011
# write_data : 00000000000000000000000000110000
# singextend : 00000000000000000111100100000000
# PC         :        41
```