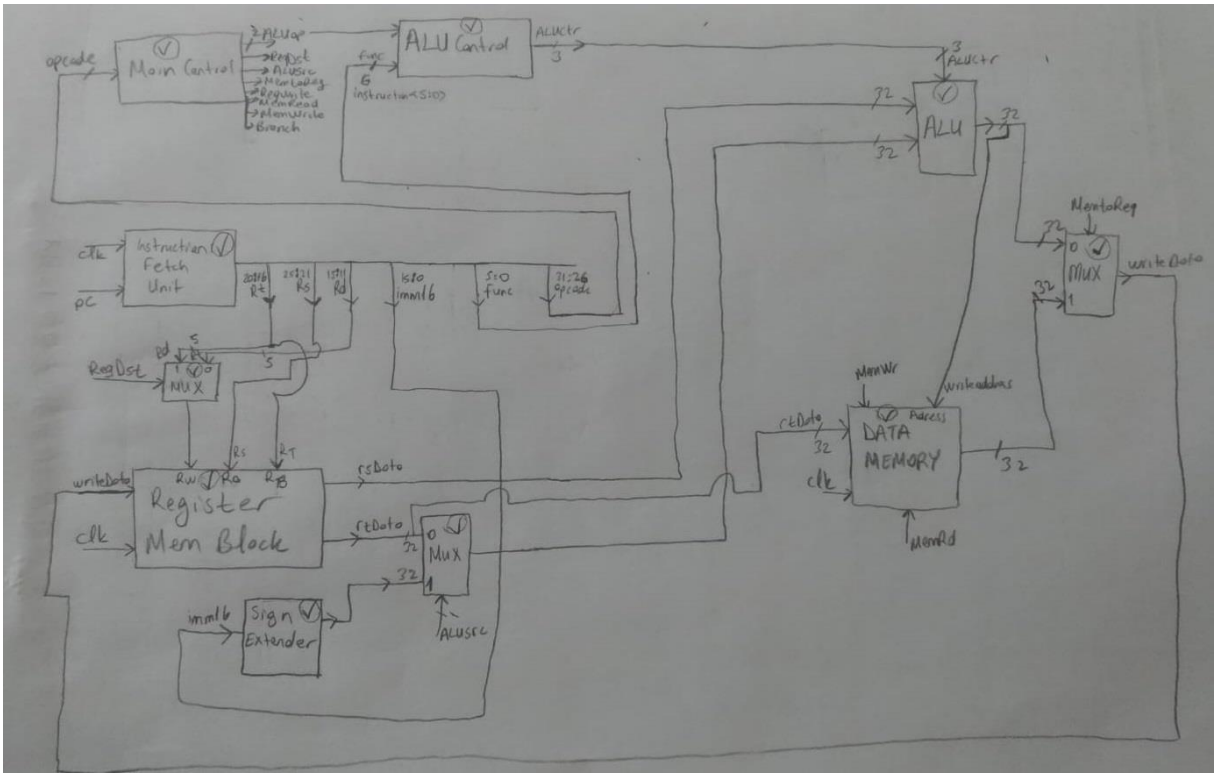
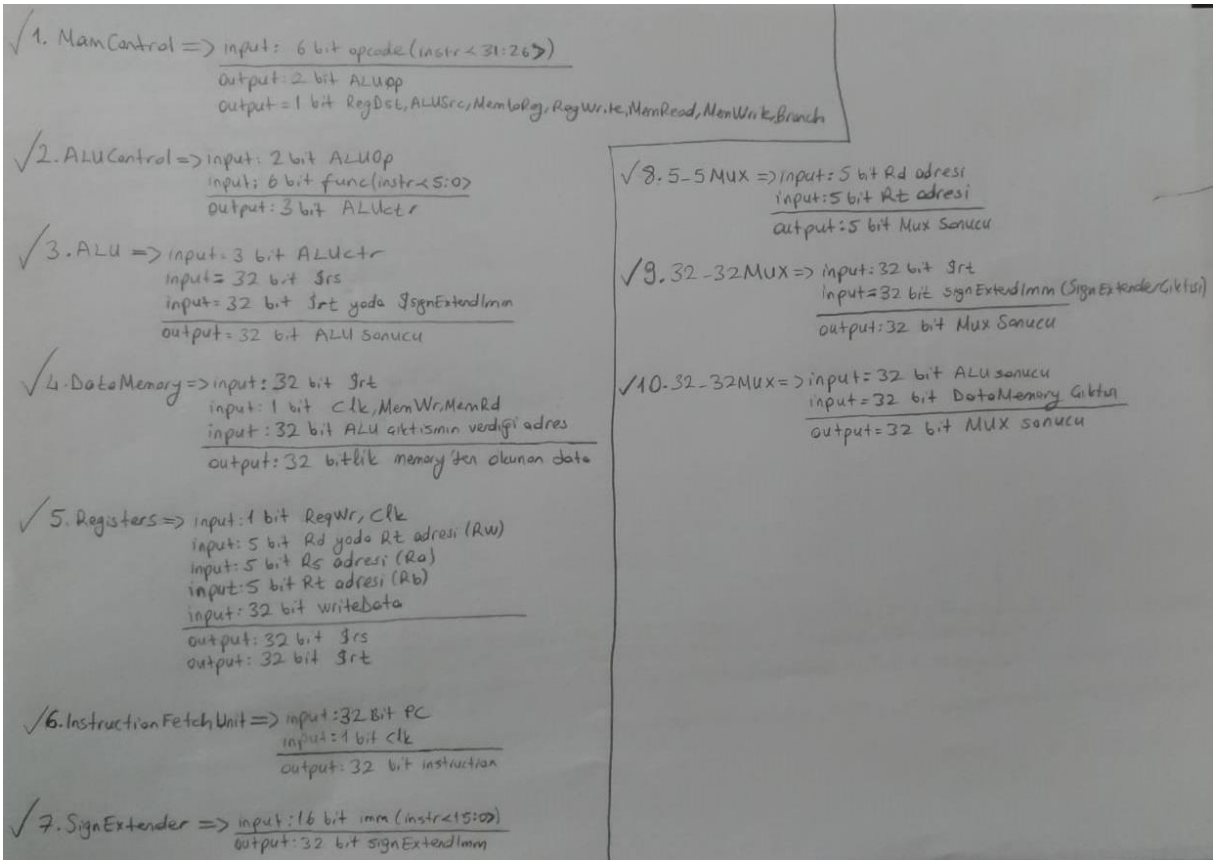
1.Schematic designs for all modules





ALUControl Truth Table

	INPUTS								OUTPUT			
	ALUOp			Function Field						Operation		
	ALUOp2	ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	ALUCtr 2	ALUCtr 1	ALUCtr 0	
ANDi	1	0	1	X	X	X	X	X	X	0	0	0	AND
ORi	1	1	1	X	X	X	X	X	X	0	0	1	OR
addiu beq	0	0	0	X	X	X	X	X	X	0	1	0	ADD
	0	1	1	X	X	X	X	X	X	1	0	0	SUBTRACT
J	X	X	X	X	X	X	X	X	X	X	X	X	—
add	1	0	0	1	0	0	0	0	0	0	1	0	add
addu	1	0	0	1	0	0	0	0	1	0	1	0	addu
and	1	0	0	1	0	0	1	0	0	0	0	0	and
nor	1	0	0	1	0	0	1	1	1	1	1	1	nor
or	1	0	0	1	0	0	1	0	1	0	0	1	or
sltu	1	0	0	1	0	1	0	1	1	1	0	0	sltu
sll	1	0	0	0	0	0	0	0	0	0	1	0	sll
srl	1	0	0	0	0	0	0	1	0	1	0	1	srl
sub	1	0	0	1	0	0	0	1	0	1	0	0	sub
subu	1	0	0	1	0	0	0	1	1	1	0	0	subu

$ALUCtr[2] = \overline{ALUOP2}.ALUOP1.ALUOP0 + ALUOP2.\overline{ALUOP1}.\overline{ALUOP0} \bullet F5.F4.\overline{F3}.F2.F1.F0 + ALUOP2.\overline{ALUOP1}.ALUOP0.F5.F4.F3.\overline{F2}.F1.F0 + ALUOP2.\overline{ALUOP1}.ALUOP0.F5.F4.F3.F2.F1.F0 + ALUOP2.ALUOP1.\overline{ALUOP0}.F5.F4.F3.\overline{F2}.\overline{F1}.F0 + ALUOP2.ALUOP1.\overline{ALUOP0}.F5.\overline{F4}.\overline{F3}.\overline{F2}.F1.\overline{F0} + ALUOP2.\overline{ALUOP1}.\overline{ALUOP0}.F5.\overline{F4}.\overline{F3}.\overline{F2}.F1.F0$

$ALUCtr[1] = \overline{ALUOP2}.\overline{ALUOP1}.\overline{ALUOP0} + ALUOP2.\overline{ALUOP1}.\overline{ALUOP0}.F5.F4.F3.F2.\overline{F1}.\overline{F0} + ALUOP2.\overline{ALUOP1}.ALUOP0.F5.\overline{F4}.\overline{F3}.F2.F1.\overline{F0} + ALUOP2.ALUOP1.\overline{ALUOP0}.F5.\overline{F4}.\overline{F3}.\overline{F2}.F1.F0$

$ALUCtr[0] = ALUOP2.\overline{ALUOP1}.ALUOP0 + ALUOP2.\overline{ALUOP1}.\overline{ALUOP0}.F5.\overline{F4}.\overline{F3}.F2.\overline{F1}.F0 + ALUOP2.\overline{ALUOP1}.ALUOP0.F5.F4.\overline{F3}.\overline{F2}.F2.\overline{F1}.F0 + ALUOP2.ALUOP1.\overline{ALUOP0}.F5.F4.F3.F2.F1.\overline{F0}$

1. add
2. addu
3. and
4. nor
5. or
6. sltu
7. sll
8. srl
9. sub
10. subu

} R-Type (opcode = 000000)

11. andi
12. ori
13. addiu
14. lw
15. sw
16. beq

} I-Type

17. j) J-Type

Main Control Truth Table

	INPUTS						OUTPUTS										
	OP5	OP4	OP3	OP2	OP1	OP0	RegDst	RegWrite	ALUSrc	MemRead	MemWrite	MemtoReg	Branch	Jump	ALUOp[2]	ALUOp[1]	ALUOp[0]
1 add	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0
2. addu	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0
3 and	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0
4 nor	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0
5 or	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0
6 sltu	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0
7 sll	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0
8 srl	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0
9 sub	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0
10 subu	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0
11 andi	0	0	1	1	0	0	0	1	1	0	0	0	0	0	1	0	1
12 ori	0	0	1	1	0	1	0	1	1	0	0	0	0	0	1	1	1
13 addiu	0	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0
14 lw	1	0	0	0	1	1	0	1	1	1	0	1	0	0	0	0	0
15 sw	1	0	1	0	1	1	X	0	1	0	1	X	0	0	0	0	0
16 beq	0	0	0	1	0	0	X	0	0	0	0	X	1	0	0	1	1
17 j	0	0	0	0	1	0	X	0	X	0	0	X	0	1	X	X	X

$RegDst = \overline{OP5}.\overline{OP3}$

$RegWrite = \overline{OP2}.\overline{OP1} + \overline{OP3}.OP0 + \overline{OP5}.OP3$

$ALUSrc = OP0 + OP3$

$MemRead = OP5.\overline{OP3}$

$MemWrite = OP5.OP3$

$MemtoReg = OP5.\overline{OP3}$

$Branch = \overline{OP3}.OP2$

$Jump = OP1.\overline{OP0}$

$ALUOp[2] = OP3.OP2 + \overline{OP2}.OP0$

$ALUOp[1] = \overline{OP3}.OP2 + OP2.OP0$

$ALUOp[0] = OP2$

2.Verilog modules and their description

nextPC: **Bir sonraki instructiona geçmek için PC yi alır ve 4 ekler next pc olarak return eder.**

_5_5_MUX: **İki tane 5 bitlik veri alır ve 1 bitlik select biri alır ve 5 bitlik verilerden birini output olarak dışarı verir.**

_32_32_MUX: **İki tane 32 bitlik veri alır ve 1 bitlik select biri alır ve 32 bitlik verilerden birini output olarak dışarı verir.**

SignExtender: **Gelen 16 bitlik verinin sign bitini alır ve 32 bitlik signextendimm haline getirip output olrak dışarı verir.**

ZeroExtender: **Gelen 16 bitlik verinin başına 16 tane 0 koyup output olarak dışarı verir.**

InstructionFetchUnit: **Gelen 32 bitlik program counter a ve clock a göre instruction.mem dosyasından okunan instructionu output olarak dışarı verir.**

DataMemory: **rt contenti, clk,memWr,memRd ve alu dan çıkan adresi alır. Eğer yazma ise rt contenti verilen adrese yazılır , eğer okuma ise verilen adresten rt ye okuma gerçekleştirilir.**

ALUControl: **MainControl unitinin ürettiği ALUOp'u ve func field i alır ve ALUCtr olarak return eder.**

Trurth table ile sadeleştirmesi yapılmıştır yukardaki fotoğrafta.

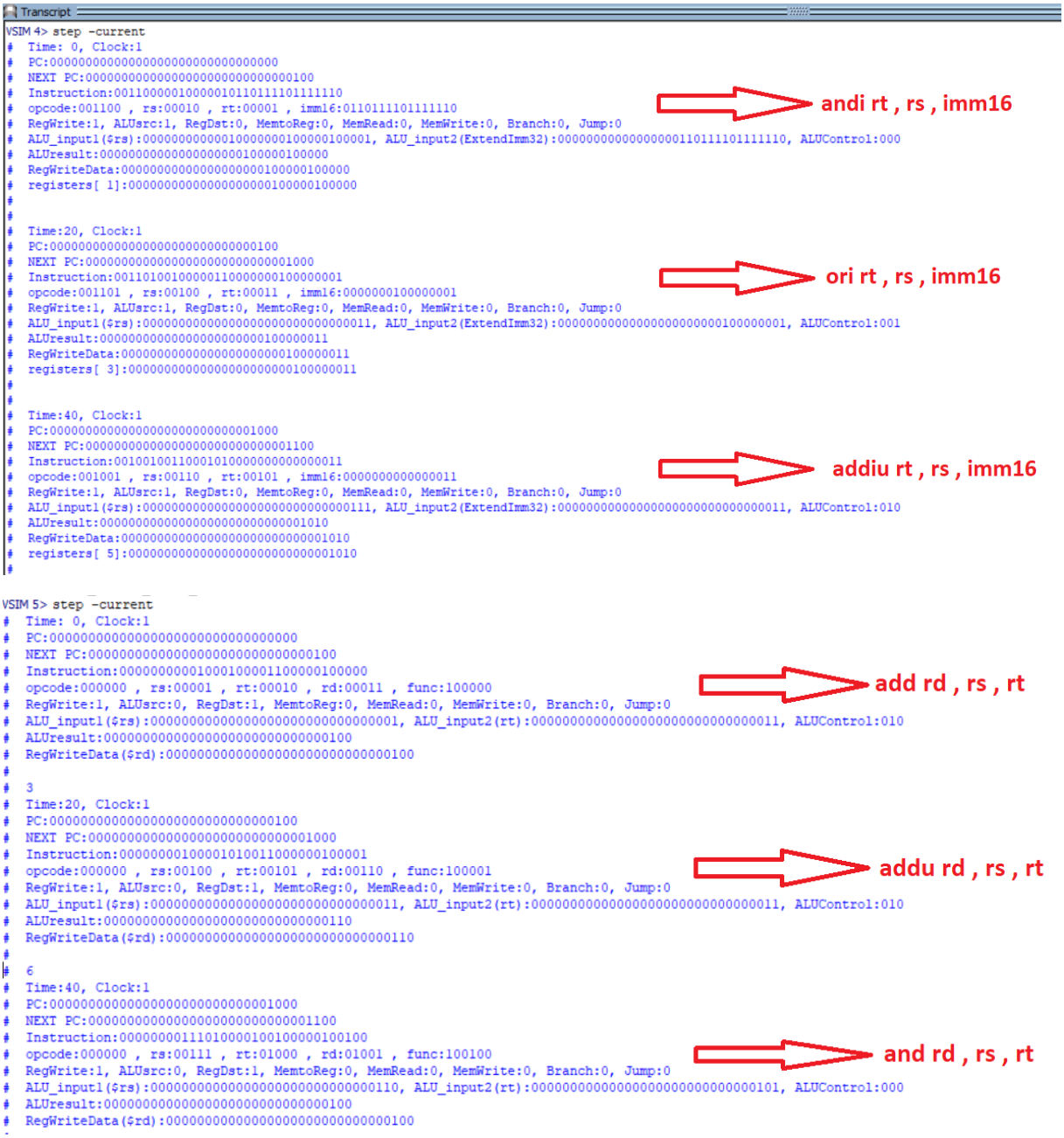
MainControl: **Input olarak gelen opcode alınır ve opcode a göre RegDst,AluSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch sinyanlleri üretilir ve ayrıca ALUop üretilir ALUControl a gönderilmek üzere.Truth table ile sadeleştirmesi yukarıdaki fotoğrafta gösterilmiştir.**

alu32: **Bir önceki assigment te kullanılan ALU nun aynısıdır.**

mips_registers: **Registere yazma yada registerden okuma amacıyla kullanılmaktadır. Rs ve rt adresleri verilrir ve dataları alınır yada write data gönderilir ve registera yazılması sağlanır.**

mips32_single_cycle: **Genel işleyişi sağlayan moduldur. Yukardi modulleri kullanır. Instructionu fetch eder.rs ve rd contenlerini alır, sign extend eder, maincontrole opcode u gönderir çıktısını alucontrole gönderir ve onunda çıktısını aluya göderir ,registerdan okunan datalar aluya gönderilir ve çıkan sonuç memory ye yada registera tekrar yazılır, her instruction bittiğinde program counter 4 arttırılır.**

3.Modelsim Simulation results



```
#  Time:60, Clock:1
#  PC:00000000000000000000000000001100
#  NEXT PC:00000000000000000000000000010000
#  Instruction:00000001010010110110000000100111
#  opcode:000000 , rs:01010 , rt:01011 , rd:01100 , func:100111
#  RegWrite:1, ALUsrc:0, RegDst:1, MemtoReg:0, MemRead:0, MemWrite:0, Branch:0, Jump:0
#  ALU_input1($rs):00000000000000000000000000000110, ALU_input2(rt):00000000000000000000000000000101, ALUControl:111
#  ALUresult:11111111111111111111111111111000
#  RegWriteData($rd):11111111111111111111111111111000
#
# 12
#  Time:80, Clock:1
#  PC:00000000000000000000000000010000
#  NEXT PC:00000000000000000000000000010100
#  Instruction:00000001101011100111100000100101
#  opcode:000000 , rs:01101 , rt:01110 , rd:01111 , func:100101
#  RegWrite:1, ALUsrc:0, RegDst:1, MemtoReg:0, MemRead:0, MemWrite:0, Branch:0, Jump:0
#  ALU_input1($rs):00000000000000000000000000000110, ALU_input2(rt):00000000000000000000000000000101, ALUControl:001
#  ALUresult:00000000000000000000000000000111
#  RegWriteData($rd):00000000000000000000000000000111
#
# 15
#  Time:100, Clock:1
#  PC:00000000000000000000000000010100
#  NEXT PC:00000000000000000000000000011000
#  Instruction:00000010000100011001000000101011
#  opcode:000000 , rs:10000 , rt:10001 , rd:10010 , func:101011
#  RegWrite:1, ALUsrc:0, RegDst:1, MemtoReg:0, MemRead:0, MemWrite:0, Branch:0, Jump:0
#  ALU_input1($rs):00000000000000000000000000000011, ALU_input2(rt):00000000000000000000000000000001, ALUControl:100
#  ALUresult:00000000000000000000000000000010
#  RegWriteData($rd):00000000000000000000000000000010
#

#  Time:120, Clock:1
#  PC:00000000000000000000000000011000
#  NEXT PC:00000000000000000000000000011100
#  Instruction:00000010011101001010100011000000
#  opcode:000000 , rs:10011 , rt:10100 , rd:10101 , func:000000
#  RegWrite:1, ALUsrc:0, RegDst:1, MemtoReg:0, MemRead:0, MemWrite:0, Branch:0, Jump:0
#  ALU_input1($rs):11000000000000000000000000010100, ALU_input2(rt):00000000000000000000000000001110, ALUControl:110
#  ALUresult:00000000000001010000000000000000
#  RegWriteData($rd):00000000000001010000000000000000
#
# 21
#  Time:140, Clock:1
#  PC:00000000000000000000000000011100
#  NEXT PC:00000000000000000000000000100000
#  Instruction:00000010110101111100000010000010
#  opcode:000000 , rs:10110 , rt:10111 , rd:11000 , func:000010
#  RegWrite:1, ALUsrc:0, RegDst:1, MemtoReg:0, MemRead:0, MemWrite:0, Branch:0, Jump:0
#  ALU_input1($rs):11100000000000000000000000010111, ALU_input2(rt):00000000000000000000000011000000, ALUControl:101
#  ALUresult:11100000000000000000000000010111
#  RegWriteData($rd):11100000000000000000000000010111
#
# 24
#  Time:160, Clock:1
#  PC:00000000000000000000000000100000
#  NEXT PC:00000000000000000000000000100100
#  Instruction:00000011001110101101100000100010
#  opcode:000000 , rs:11001 , rt:11010 , rd:11011 , func:100010
#  RegWrite:1, ALUsrc:0, RegDst:1, MemtoReg:0, MemRead:0, MemWrite:0, Branch:0, Jump:0
#  ALU_input1($rs):00000000000000000000000000000111, ALU_input2(rt):00000000000000000000000000000001, ALUControl:100
#  ALUresult:00000000000000000000000000000110
#  RegWriteData($rd):00000000000000000000000000000110
#
# 27
#  Time:180, Clock:1
#  PC:00000000000000000000000000100100
#  NEXT PC:00000000000000000000000000101000
#  Instruction:00000011100111011111100000100011
#  opcode:000000 , rs:11100 , rt:11101 , rd:11110 , func:100011
#  RegWrite:1, ALUsrc:0, RegDst:1, MemtoReg:0, MemRead:0, MemWrite:0, Branch:0, Jump:0
#  ALU_input1($rs):00000000000000000000000000000111, ALU_input2(rt):00000000000000000000000000000011, ALUControl:100
#  ALUresult:00000000000000000000000000000100
#  RegWriteData($rd):00000000000000000000000000000100
```

nor rd , rs , rt

or rd , rs , rt

sltu rd , rs , rt

sll rd , rt , shamt

srl rd , rt , shamt

sub rd , rs , rt

subu rd , rd , rt

4. If not compiling or partial working the explanation of which parts work which parts do not

Toplamda 17 instructionun çalışması gerekiyor. DataMemory unitini yazdım fakat lw,sw,beq ve j instructionlarını test edemedim o yüzden çalışıp çalışmadıklarını bilmiyorum.

Diğer I type(andi, ori , addiu) ve R(add,addu,and,nor,or,sltu,sll,srl,sub,subu) type tüm instructionları test ettim ve çalışıyor.(13 instruction)(13 ü de yukardaki test screens shotlarında çalıştığı detaylı şekilde gösterilmiştir)