Gebze Technical University
Computer Engineering

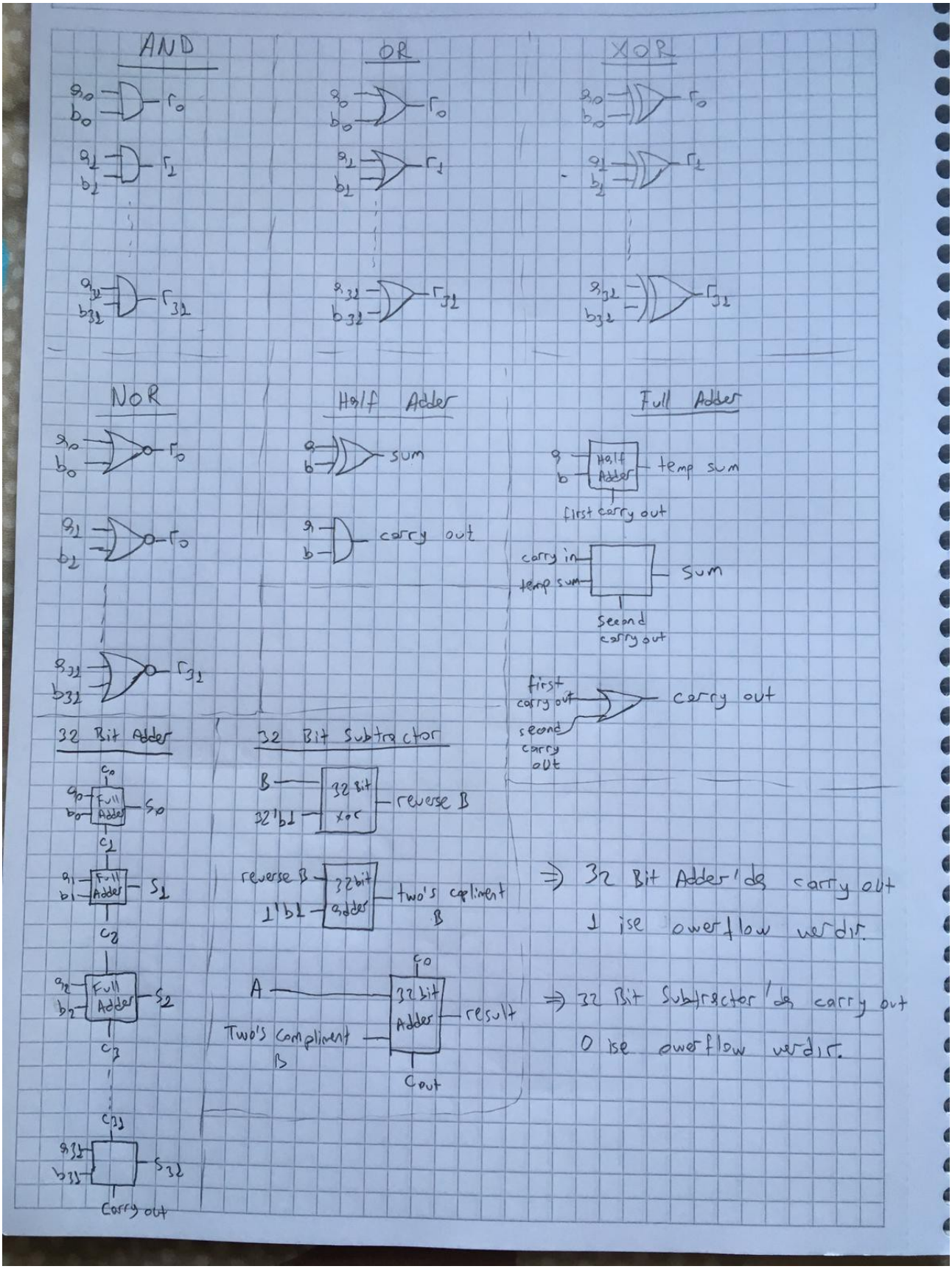
CSE 331 - 2018 Fall

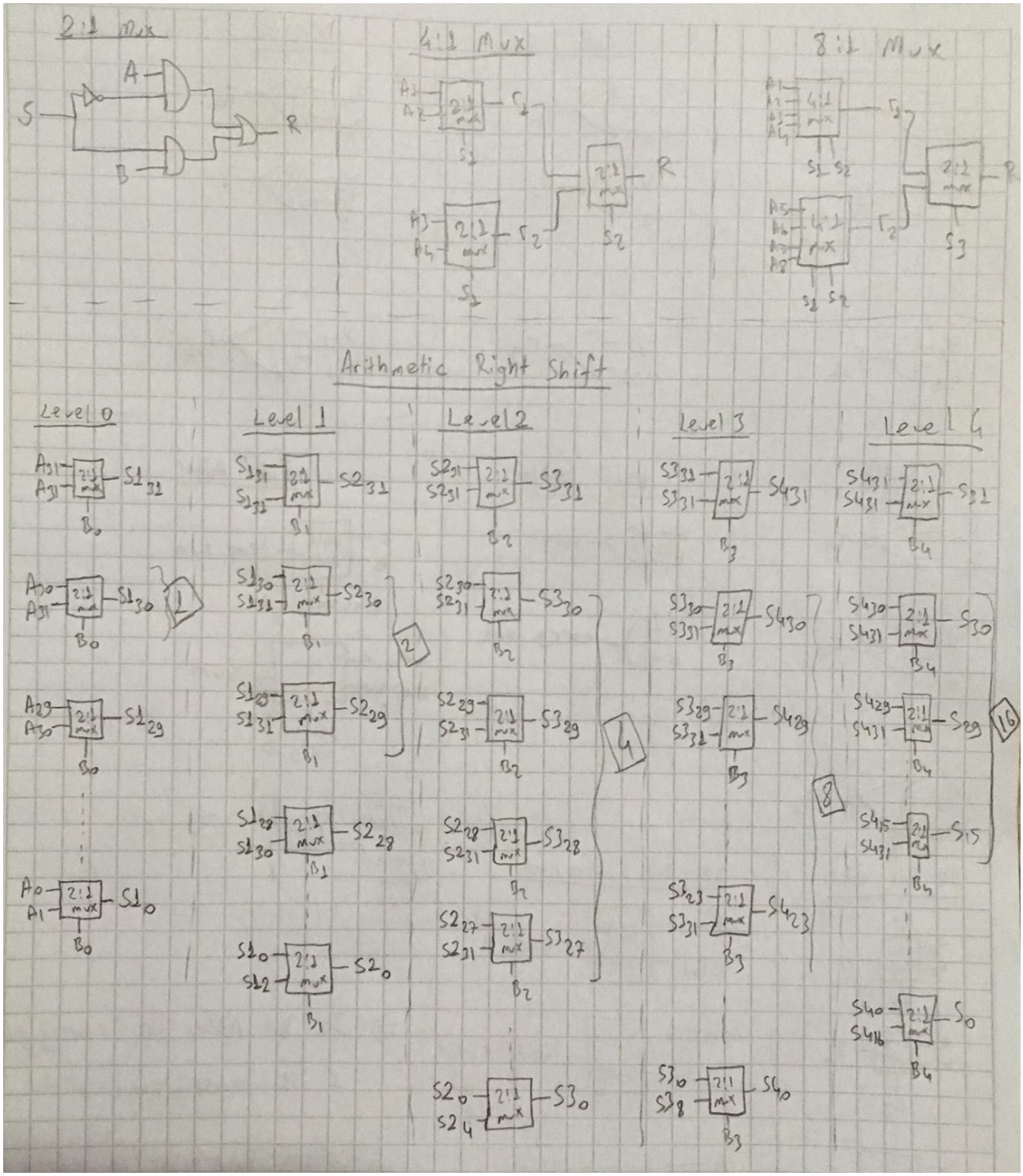
HOMEWORK 4 REPORT

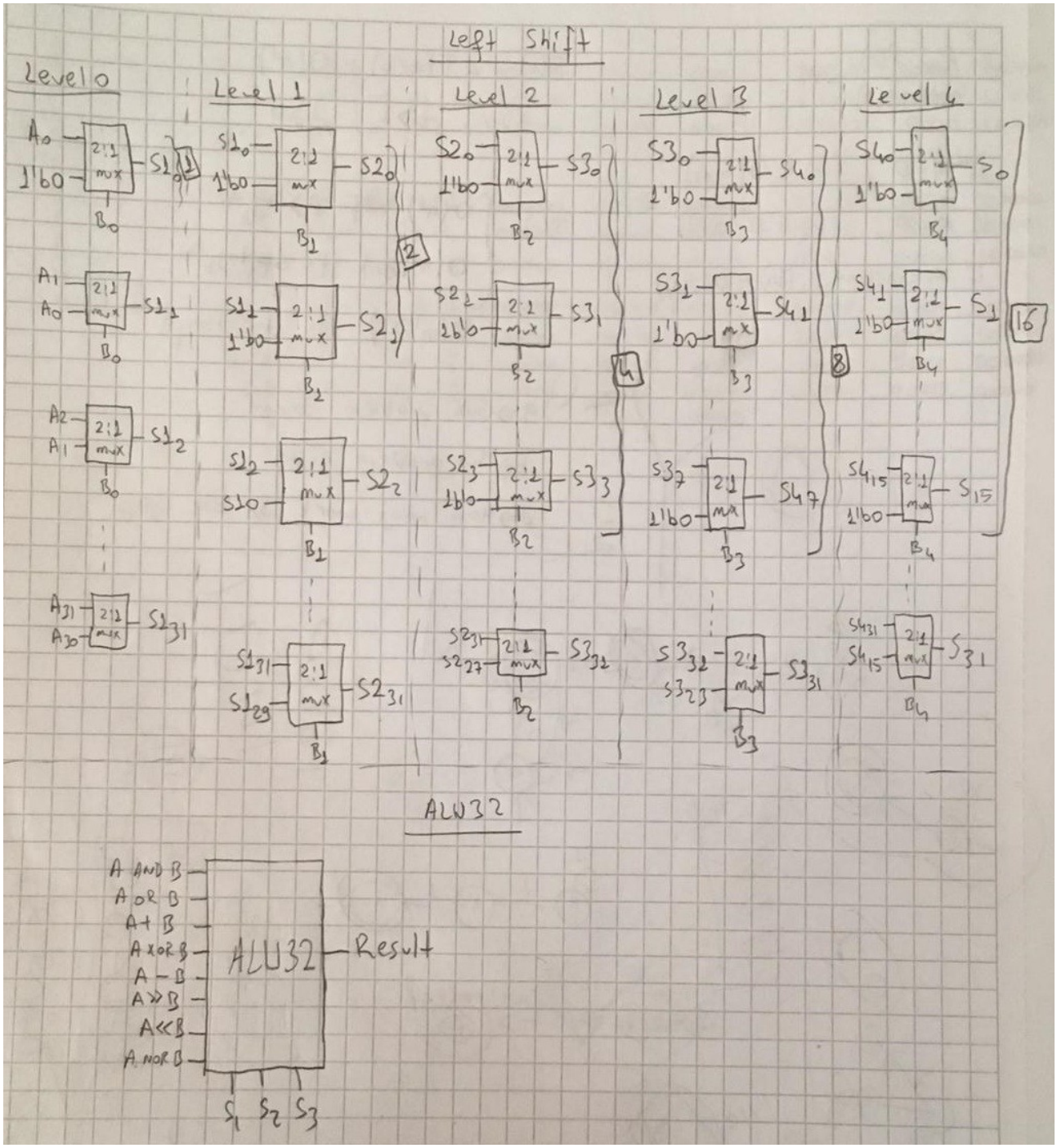
EMİRHAN KARAGÖZOĞLU
151044052

Course Assistant: Fatma Nur Esirci

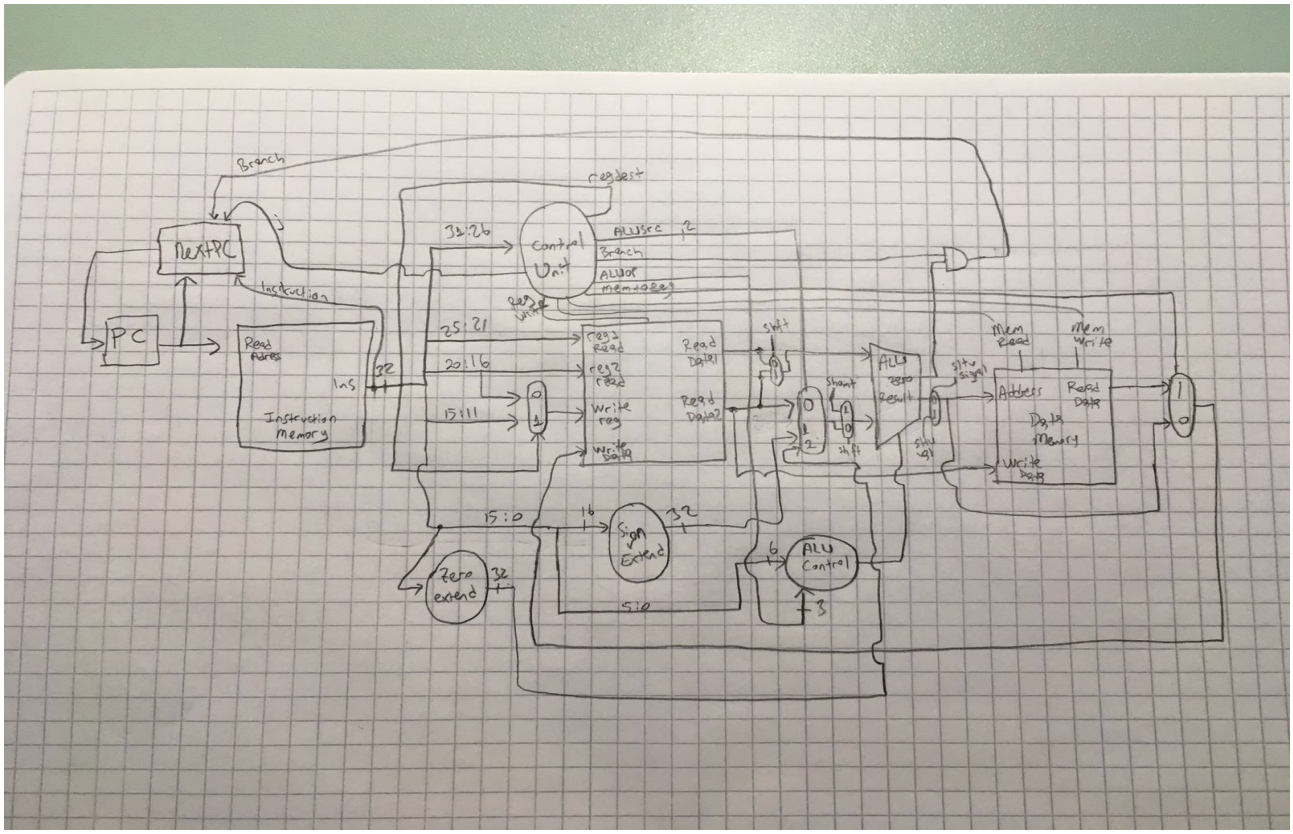
Schematic Desings for all Modules




2:1 mux
A
S
R
B
4:1 MUX
8:1 MUX
Arithmetic Right Shift
Level 0
Level 1
Level 2
Level 3
Level 4
1
2
4
8
16


Left Shift
Level 0
Level 1
Level 2
Level 3
Level 4
ALU32
A AND B
A OR B
A + B
A XOR B
A − B
A >> B
A << B
A NOR B
ALU32
Result
S1
S2
S3

Mips 32 Single Cycle Datapath



Control Unit

opcode		RegDst	ALUSrc 1	ALUSrc 0	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUop2	ALUop1	ALUop0	Jump
000000	R TYPE	1	0	0	0	1	0	0	0	0	1	0	0
000010	j	X	X	X	X	0	0	0	0	X	X	X	1
001100	andi	0	1	0	0	1	0	0	0	0	1	1	0
001101	ori	0	1	0	0	1	0	0	0	1	0	0	0
001001	addiu	0	0	1	0	1	0	0	0	0	0	0	0
100011	lw	0	0	1	1	1	1	0	0	0	0	0	0
101011	sw	X	0	1	X	0	0	1	0	0	0	0	0
000100	beq	X	0	0	X	0	0	0	1	0	0	1	0

0 → rt
1 → singextend immed
2 → zeroextend immed

lw, sw, addiu → 0
beq → 1
R Type → 2
andi → 3
ori → 4

RegDst = R-Type

MemtoReg = lw

RegWrite = R-Type + andi + ori + addiu + lw

MemRead = lw

MemWrite = sw

Branch = beq

Jump = j

ALUSrc0 = addiu + lw + sw
ALUSrc1 = andi + ori

ALUop0 = andi + beq
ALUop1 = R-Type + andi
ALUop2 = ori

Alu Control

Instruction opcode	ALUop	Instruction Operation	Function Field	Desired ALU action	ALU control
LW	000	load word	xxxxxx	add	010
SW	000	store word	xxxxxx	add	010
beq	001	branch equal	xxxxxx	subtract	100
R-Type	010	add	100000	add	010
R-Type	010	subtract	100010	subtract	100
R-Type	010	and	100100	and	000
R-Type	010	or	100101	or	001
R-Type	010	srl	000010	srl	101
R-Type	010	sll	000000	sll	110
R-Type	010	nor	100111	nor	111
andi	011	andi	xxxxxx	and	000
ori	100	ori	xxxxxx	or	001
addiu	000	addiu	xxxxxx	add	010
R-Type	010	sltu	101011	subtract	100
R-Type	010	addu	100001	add	010
R-Type	010	subu	100011	sub	100

$$ALU\ control[2] = (ALUop[0] \text{ xor } ALUop[1]) \cdot (-Function[5] + Function[1] + ALUop[0])$$

$$ALU\ control[1] = (ALUop[0] \text{ xnor } ALUop[2]) \cdot ((Function[2] \text{ xnor } Function[1]) + (-ALUop[1]))$$

$$ALU\ control[0] = (ALUop[2] \text{ xor } ALUop[1]) \cdot (-ALUop[0]) \cdot \left(((-Function[5] \cdot Function[1]) + (Function[2] \cdot Function[0])) + ALUop[2]\right)$$

Verilog Modules and Their Description

AND : 32 bit 2 adet sayı alır AND işlemi yapar ve 32 bit sonuç return eder.
OR : 32 bit 2 adet sayı alır OR işlemi yapar ve 32 bit sonuç return eder.
NOR : 32 bit 2 adet sayı alır NOR işlemi yapar ve 32 bit sonuç return eder.
XOR : 32 bit 2 adet sayı alır XOR işlemi yapar ve 32 bit sonuç return eder.
+ operator : 32 bit 2 adet sayı alır toplama işlemi yapar ve 32 bit sonuç return eder.
- operator : 32 bit 2 adet sayı alır çıkarma işlemi yapar ve 32 bit sonuç return eder.
>> operator : 32 bit 2 adet sayı alır ilk sayıyı ikinci sayı kadar aritmetik olarak sağa kaydırır.
<< operator : 32 bit 2 adet sayı alır ilk sayıyı ikinci sayı kadar sola kaydırır.
Extenders: sign extend ve zero extend işlemlerini gerçekleştirir.
Alu32: 32 bit 2 sayı ve 3 bit seçici alır, seçiciye göre gerekli işlemin sonucunu döndürür.
RegisterBlock: 5'er bit rs, rt ve rd adresleri alır. Bu adreslerideki rs ve rt contentlerini okuyup döndürür. Ayrıca 32 bit write data ve 1 bit write sinyal alır. Write sinyaline göre rd veya rt nin adresine write datayı yazar.
InstructionBlock: 32 bit pc alır ve instruction memorydeki o pc adresinden ilgili instructionı okuyup döndürür.
MemoryBlock: 32 bit write data ve address alır. 1'er bit write ve read sinyali alır. Bu sinyallere göre gelen address'e write datayı yazar yada address'ten datayıp okuyup döndürür.
NextPC: Branch, Jump ve normal durumlar için pc değerini hesaplar.
Control: 6bit opcode alır ve gerekli tüm control sinyallerini üretir. (RegDst,ALUSrc,Branch,MemRead,MemWrite,MemtoReg,ALUop,Jump, RegWrite)
AluControl: 6 bit function field ve 3 bit ALUop alır 3 bit alu select döndürür.
Mips32_single_cycle: 2 clock alır ve tüm datapathi yönetir.

Modelsim Simulation Results

```
VSIM 5> step -current
# PC = 00000000
# PC = 00000001
# PC = 00000002
# PC = 00000003
# PC = 00000004
# PC = 00000005
# PC = 00000006
# PC = 00000007
# PC = 00000008
# PC = 00000009
# PC = 0000000a
# PC = 00000010
# PC = 0000000b
# PC = 0000000c
# PC = 0000000d
# PC = 0000000e
# PC = 0000000f
# PC = 00000010
# ** Note: $finish    : C:/Users/Emir/Desktop/co4/151044052/151044052_restored/mips32_single_cycle_testbench.v(32)
#    Time: 510 ps  Iteration: 1  Instance: /mips32_single_cycle_testbench
```

Tasarladığımız Mips32 single cycle datapath R-Type(and, or, add, sub, srl, sll, nor, sltu, addu, subu) , I-Type (andi, ori, addiu, lw, sw, beq) ve J-Type (jump) instructioları desteklemektedir. Result olarak tüm bu instructionlar için ortak nokta olan PC değerlerini bastırdım. Bu sayede instruction memoryden hangi sıra ile instructionların datapath' e alındığını görebilirsiniz. Instruction memory içerinde her instruction'ın yanına ne işlem yaptığı ve işlemin beklenen sonucu yorum olarak yazılmıştır. Register blogunda 1-9 arası registerlar kendi değerlerini göstermektedir (r3=3, r9=9 gibi). Geriye kalan tüm registerların değerleri datapath çalıştıktan sonraki değişiklikleri rahat gözlemleyebilmek için 0 olarak verilmiştir. Register bloguna yazma yapan tüm instructionlar 10-23 arasındaki registerları kullanmışlardır.(her bir instructionun sonucu bu aralıkta farklı bir register'a yazılmıştır ve bunlar yorum olarak instruction memory dosyasında belirtilmiştir.) . Instruction blogu 32 instruction tutabiliyor. Register blogu 32 adet register tutabiliyor. Memory blogu 256 adet content tutabiliyor. Memory blogu da ilk başta 256 adet 0 contenti barındırmaktadır.
Testi çalıştırmadan önceki ve sonraki memory ve register contentlerine bu bilgiler ışığında bakarak test sonuçlarını gözlemleyebilirsiniz.

Extre Point:
Register blogunda r0-r26-r27-r28-r29-r30-r31 nolu register'lara yazılmasını engelledim.