

Gebze Technical University

Computer Organization

CSE 331

Homework 4

Report

Elif Akgün

1801042251

Ödev için yazdığım modüller ve testleri aşağıdaki gibidir.

1) ALU1Bit

İşlemler için 32 bit ALU gerekiyor. 1 bitlik ALU'lar ile 32 bitlik ALU oluşturdum.

```
VSIM 7> step -current
# Time= 0      ALUOp=000      Ai=0      Bi=1      Cin=0      Result=0      Cout=0
# Time=20      ALUOp=000      Ai=0      Bi=1      Cin=1      Result=0      Cout=1
# Time=40      ALUOp=000      Ai=1      Bi=1      Cin=0      Result=1      Cout=1
# Time=60      ALUOp=001      Ai=0      Bi=0      Cin=0      Result=0      Cout=0
# Time=80      ALUOp=001      Ai=1      Bi=0      Cin=0      Result=1      Cout=0
# Time=100     ALUOp=001      Ai=1      Bi=1      Cin=0      Result=1      Cout=1
# Time=120     ALUOp=010      Ai=0      Bi=1      Cin=0      Result=1      Cout=0
# Time=140     ALUOp=010      Ai=0      Bi=1      Cin=1      Result=0      Cout=1
# Time=160     ALUOp=010      Ai=0      Bi=0      Cin=0      Result=0      Cout=0
# Time=180     ALUOp=110      Ai=0      Bi=1      Cin=0      Result=0      Cout=0
# Time=220     ALUOp=110      Ai=1      Bi=1      Cin=0      Result=1      Cout=0
```

2) MUX4To1

1'er bitlik girişleri olan bir MUX'tur. ALU'dan çıkacak sonucu belirlemek için kullandım.

```
VSIM 5> step -current
# time = 0, A= 0, B= 1, C= 0, D= 0, S1= 1, S0= 0, Output= 0
# time =20, A= 1, B= 1, C= 0, D= 0, S1= 1, S0= 0, Output= 0
# time =40, A= 1, B= 0, C= 1, D= 0, S1= 0, S0= 1, Output= 0
# time =60, A= 1, B= 0, C= 1, D= 0, S1= 1, S0= 0, Output= 1
# time =80, A= 0, B= 1, C= 1, D= 0, S1= 0, S0= 1, Output= 1
# time =100, A= 0, B= 1, C= 0, D= 1, S1= 1, S0= 0, Output= 0
# time =120, A= 1, B= 1, C= 0, D= 1, S1= 0, S0= 0, Output= 1
# time =140, A= 1, B= 0, C= 1, D= 1, S1= 1, S0= 1, Output= 1
```

3) ALU32bit

[illegible]

[illegible][illegible]

Bu modül register block' u temsil eder. Input olarak verilen adreslerden registerların içeriğini okur ve output olarak bu içerikleri verir. Yeni instructionlardan biri geldiyse hem rs'e hem de rd'ye yazarken l type instruction gelince rt'ye yazar. Aşağıdaki ekran fotoğrafında da görüldüğü gibi sigJal inputu 1 ise yani jal instructionı çalışıyorsa 31. register'a PC+4 yazar.

```

VSIM 12> step -current
# Time= 0
# regWrite1=1 regWrite2=1 sigJal=1
# read_reg1=00001
# read_data1=10000010001000100010000101001010
# read_reg2=00010
# read_data2=10000001001000010000011010101010
# write_reg1=00011
# write_data1=000000000000000000000000000011110
# write_reg2=00100
# write_data2=000000000000000000000000000011111
# PC=00000000000000000000000000001000
# registers[31]=00000000000000000000000000001100
#
# Time=120
# regWrite1=1 regWrite2=1 sigJal=0
# read_reg1=00101
# read_data1=011001010010100000000000010111011
# read_reg2=00110
# read_data2=0000000000000000000000000000011
# write_reg1=00111
# write_data1=000000000000000000000000000011110
# write_reg2=01000
# write_data2=000000000000000000000000000011111
# PC=000000000000000000000000000010000
# registers[31]=00000000000000000000000000001100
#
# Time=240
# regWrite1=1 regWrite2=1 sigJal=1
# read_reg1=01001
# read_data1=0000000000000000000000000000011
# read_reg2=01010
# read_data2=10110010111110111111001011110110
# write_reg1=01011
# write_data1=000000000000000000000000000011110
# write_reg2=01100
# write_data2=000000000000000000000000000011111
# PC=0000000000000000000000000000100
# registers[31]=00000000000000000000000000001000
.

```

6) dataMemory

Memory'ye yazma ve memory'den okuma işlemleri bu blokta yapılır. memRead biti 1 ise verilen adresteki veriyi okurken memWrite biti 1 ise girilen adrese veriyi yazar.

11) signExtender

I-type instructionlarda immediate partı 32 bite tamamlıyoruz. Sign extend yapmak için bu modülü yazdım.

```
sim:/signExtender_testbench/imm16bit
VSIM 10> step -current
# Time= 0
# Input=1100110100000000
# Output=111111111111111100110100000000
#
#
# Time=20
# Input=0000001100110000
# Output=00000000000000000000001100110000
#
```

12) MUX2to1_32Bit

Datapath üzerinde 32 bitlik kablolar arasında seçim yapabilmek için bu modülü yazdım.

```
VSIM 5> step -current
# Time= 0
# A=000000000000000000000010101110000
# B=00001110000011100000000000000001
# select=0
# Result=00001110000011100000000000000001
# Time=20
# A=000000000000000000000010101110000
# B=00001110000011100000000000000001
# select=1
# Result=000000000000000000000010101110000
# Time=40
# A=11110001101000001111111010100010
# B=00100011000010000000000010101110
# select=0
# Result=00100011000010000000000010101110
# Time=60
# A=11110001101000001111111010100010
# B=00100011000010000000000010101110
# select=1
# Result=11110001101000001111111010100010
```

13) MUX2to1_1bit

Datapath üzerinde 1 bitlik kablolar arasında seçim yapabilmek için bu modülü yazdım.

```
VSIM 5> step -current
# Time= 0      A=0      B=0      select=0 Result=0
# Time=20      A=0      B=0      select=1 Result=0
# Time=40      A=1      B=0      select=0 Result=0
# Time=60      A=1      B=0      select=1 Result=1
# Time=80      A=0      B=1      select=0 Result=1
# Time=100     A=0      B=1      select=1 Result=0
# Time=120     A=1      B=1      select=0 Result=1
# Time=140     A=1      B=1      select=1 Result=1
```

14) MUX2to1_5Bit

Datapath üzerinde 5 bitlik kablolar arasında seçim yapabilmek için bu modülü yazdım.

```
sim:/MUX2to1_5Bit_testbench/R
VSIM 10> step -current
# Time= 0
# A=10000
# B=00001
# select=0
# Result=00001
#
#
# Time=20
# A=10000
# B=00001
# select=1
# Result=10000
#
#
# Time=40
# A=00010
# B=01110
# select=0
# Result=01110
#
#
# Time=60
# A=00010
# B=01110
# select=1
# Result=00010
```


15) leftShift2Bit_32bit

Branch instructionlarında sign extend işleminden sonra 2 bit sola kaydırmak gerekiyor.
Bunun için bu modülü yazdım.

```
VSIM 6> step -current
# Time= 0
# Input=      11100000000000000000010101110000
# Output=     1000000000000000000001010111000000
#
#
# Time=20
# Input=      00001110000011100000000000000001
# Output=     00111000001110000000000000000100
#
#
# Time=40
# Input=      001000111110100000000000010101110
# Output=     100011111010000000000001010111000
```

16) bitExtender1to32

1 bitlik sinyalleri 32 bit yapmak için bu modülü yazdım.

```
sim:/bitExtender1to32_testbench/X
VSIM 5> step -current
# Time= 0
# A=0
# X=00000000000000000000000000000000
#
#
# Time=20
# A=1
# X=00000000000000000000000000000001
#
```

17) signExtender1to32

1 bitlik sinyallerde sign extend işlemi yapabilmek için bu modülü yazdım

```

sim:/signExtender1to32_testbench/X
VSIM 6> step -current
# Time= 0
# A=0
# X=00000000000000000000000000000000
#
#
# Time=20
# A=1
# X=11111111111111111111111111111111

```

18) branchAddress

Branch instructionlarda $PC = PC + 4 + \text{BranchAddress}$ oluyor. BranchAddress'ı belirlemek için bu modülü yazdım.

```

sim:/branchAddress_testbench/output32bit
VSIM 5> step -current
# Time= 0
# Input=1100110100000000
# Output=11111111111111110011010000000000
#
#
# Time=20
# Input=0000001100110000
# Output=00000000000000000000110011000000
.

```

19) jumpAddress

j ve jal instructionlarında $PC = \text{JumpAddress}$ oluyor. JumpAddress'ı belirlemek için bu modülü yazdım.

```

sim:/jumpAddress_testbench/jumpAddr
VSIM 5> step -current
# Time= 0
# PC=00000000000000000000000000000100
# Address=0000000000000111111111111111
# Jump Address=000000000000000011111111111100
#
#
# Time=20
# PC=10100000000000000000000000011100
# Address=1111111111110000000000000000
# Jump Address=101011111111111100000000000000
.

```

20) myXOR32bit

xorn instructionu için 32 bitlik XOR alan bu modülü yazdım.

```
sim:/myXOR32bit_testbench/R
VSIM 16> step -current
# Time= 0
# A=00000100001000000000010101110000
# B=00001110000011100000000001100001
# R=00001010001011100000010100010001
#
#
# Time=20
# A=00100000110010000000000010101110
# B=00100001100000000110000010010010
# R=00000001010010000110000000111100
```

21) MIPS32BitProcessor

Ana modüldür. Tüm alt modüller (instMemory, registerBlock, dataMemory vs.) bu modülün altında çağırılır ve üretilen sinyallere göre gerekli işlemler yapılır. Jr instructionu bu modülde kontrol edilir. Jr sinyali 1 ise PC = R[rs] (= Read data 1) işlemi yapılır. Bu modülü test etmek için instructionların yazıldığı instructions.txt, registerların okunduğu registers_first.mem, registerların yazıldığı registers_last.mem, memory'nin okunduğu data_firts.mem, ve memory'e yazmak için data_last.mem dosyalarını kullandım. İlk olarak sırayla xorn, orn, andn, subn, addn, ori, lw, sw, ve lui instructionlarını test ettim. Dosyalar aşağıdaki gibidir.

```
data_first.mem
1 0010000111111000000000000000000100
2 0010010000110001000000000000000000
3 0011000000000001000000000000000000
4 0011010000000001111111111111111111
5 0000000000000000000000000000000000
6 0000000000000000000000000000000010
7 0000000000000000000000000000000010
8 000000000000000000000000000000001000
9 000000000000000000000000000000001000
10 00000000000000000000000000000000100000
11 00000000000000000000000000000000100000
```

```
data_last.mem
1 // memory data file (do not edit
2 // instance=/MIPS32BitProcessor_t
3 // format=bin addressradix=h data
4 0010000111111000000000000000000100
5 0010010000110001000000000000000000
6 1111111111111111111111111111111111
7 0011010000000001111111111111111111
8 0000000000000000000000000000000000
9 0000000000000000000000000000000010
10 0000000000000000000000000000000010
11 000000000000000000000000000000001000
12 00000000000000000000000000000000100000
13 00000000000000000000000000000000100000
14 00000000000000000000000000000000100000
```

```
registers_first.mem x registers_last.mem x
1 0000000000000000000000000000100
2 00000011000000110010011111100000
3 10000001001000010000011010101010
4 11000000011000001000001001010101
5 01000100001010001100101000111100
6 01100101001010000000000010111011
7 00100001001010001110000011110000
8 00000001001010001010010100100010
9 11100101011011000010110110111011
10 01001101000001000000110111101110
11 00000000000000000000000000001110
12 01001101000001000000110100011000
13 11100101011011000010110111001010
14 01001101000001000000110110100110
15 00000011000000000010000001101101
16 11000000011000001000001001010101
17 10000001001000010000011010101010
18 0000000000000000000000000000111
19 01100101001010000000000010111011
20 0000000000000000000000000000001
21 11111111111111111111111111111111
22 11100101011011000010110110111011
23 01001101000001000000110111101110
24 11111111111111111001111111100000
25 11111111111111111111111111100010
26 11100101011011000010110111001010
27 01001101000001000000110110100110
28 00000100001000001000110010000111
29 01110101000000010000000010000110
30 01011001011101000001101010100000
31 00010110000111100000000010000111
32 10101010110000011111101010011001
```

```
registers_last.mem x registers_first.mem x
1 // memory data file (do not edit the
2 // instance=/MIPS32BitProcessor_test
3 // format=bin addressradix=h dataaddress
4 000000000000000000000000000000100
5 10000010001000100010000101001010
6 10000001001000010000011010101010
7 00000000000000000000000000000011
8 01100101001010001100101010111111
9 01100101001010000000000010111011
10 00000000000000000000000000000011
11 00000001001010000010010100100010
12 11100101011011000010110110111011
13 00000000000000000000000000000011
14 10110010111110111111001011110110
15 01001101000001000000110100011000
16 00000000000000000000000000000010
17 01010000000001000010111000010011
18 0000001100000000001000001101101
19 00000000000000000000000000000011
20 10110010111110111111001011110110
21 000000000000000000000000000000111
22 00000000000000000000000000001000
23 000000000000000000000000000000001
24 11111111111111111111111111111111
25 11001100110011000000000000000000
26 01001101000001000000110111101110
27 111111111111111111001111111100000
28 111111111111111111111111111100010
29 11100101011011000010110111001010
30 01001101000001000000110110100110
31 00000100001000001000110010000111
32 01110101000000010000000010000110
33 01011001011101000001101010100000
34 00010110000111100000000010000111
35 10101010110000011111101010011001
```

```
instructions.txt - Not Defteri
Dosya Düzen Biçim Görünüm Yardım
000000_00001_00010_00011_00000_100110
000000_00100_00101_00110_00000_100101
000000_00111_01000_01001_00000_100100
000000_01010_01011_01100_00000_100010
000000_01101_01110_01111_00000_100000
001101_01010_10000_01100_00000_100010
100011_10001_10010_0000000000000000
101011_10011_10100_0000000000000001
001111_01101_10101_1100110011001100
```


- XORN

```
# Time= 0
# PC=00000000000000000000000000000000
# Instruction=0000000001000100001100000100110
# Opcode=000000
# rs=00001
# rt=00010
# rd=00011
# shamt=00000
# Function Code=100110
# Immediate=0001100000100110
# Mem_read=0
# Reg_Write1=1
# Reg_Write2=1
# Mem_Write=0
# RegDst=1
# Jump=0
# Jal=0
# Jr=0
# Branch=0
# BranchNot=0
# MemToReg=0
# ALUOp2bit=10,
# ALUOp3bit=000
# ALUSrc=0
# write_reg1=00001
# read_data1=0000001100000011001001111100000
# read_data2=10000001001000010000011010101010
# write_data1=10000010001000100010000101001010
# write_data2=00000000000000000000000000000011
# registers[31]=10101010110000011111101010011001
```

Ekran çıktısında da görüldüğü gibi dosyadan instruction okunup parçalandı. Rs'in ve Rt'nin içeriği okunarak XOR işlemi yapıldı. XORN, yeni tip instruction olduğu için hem RegWrite1 hem de RegWrite2 sinyali 1'dir. Bu yüzden sonuç Rs'in adresine yazıldı. Sonuç 0'dan büyük olduğu için Rd'nin adresine 3 yazıldı. (registers_last.mem'de 5. ve 7. satır). Diğer yeni tip instructionlar aynı şekilde işlemektedir. Raporun uzamaması için sadece ekran çıktılarını ekledim.

- ORN

```
# Time=50  
# PC=00000000000000000000000000000100  
# Instruction=00000000100001010011000000100101  
# Opcode=000000  
# rs=00100  
# rt=00101  
# rd=00110  
# shamt=00000  
# Function Code=100101  
# Immediate=0011000000100101  
# Mem_read=0  
# Reg_Write1=1  
# Reg_Write2=1  
# Mem_Write=0  
# RegDst=1  
# Jump=0  
# Jal=0  
# Jr=0  
# Branch=0  
# BranchNot=0  
# MemToReg=0  
# ALUOp2bit=10,  
# ALUOp3bit=001  
# ALUSrc=0  
# write_reg1=00100  
# read_data1=01000100001010001100101000111100  
# read_data2=01100101001010000000000010111011  
# write_data1=01100101001010001100101010111111  
# write_data2=00000000000000000000000000000011  
# registers[31]=101010101100000011111101010011001
```

- ANDN ve SUBN

```
# Time=150
# PC=000000000000000000000000000000001000
# Instruction=00000000111010000100100000100100
# Opcode=000000
# rs=00111
# rt=01000
# rd=01001
# shamt=00000
# Function Code=100100
# Immediate=0100100000100100
# Mem_read=0
# Reg_Write1=1
# Reg_Write2=1
# Mem_Write=0
# RegDst=1
# Jump=0
# Jal=0
# Jr=0
# Branch=0
# BranchNot=0
# MemToReg=0
# ALUOp2bit=10,
# ALUOp3bit=000
# ALUSrc=0
# write_reg1=00111
# read_data1=00000001001010001010010100100010
# read_data2=11100101011011000010110110111011
# write_data1=00000001001010000010010100100010
# write_data2=00000000000000000000000000000011
# registers[31]=10101010110000011111101010011001
```

[illegible]

[illegible]

- LW

[illegible]

lw l-type bir instructiondır. Bu yüzden immediate part için sign extend işlemi yapılır ve Register bloktaki Read data 1 ile add işlemi yapılarak sonuç Data Memory'nin Address'ine verilir. Belirtilen adresteki veri okunarak Rt'nin adresine yazılır. (Bknz data_first.mem'de 8. Satırdaki veri, register_last.mem'de 22. satırdadır.)

- SW

[illegible]

sw l-type bir instructiondır. Bu yüzden immediate part için sign extend işlemi yapılır ve Register bloktaki Read data 1 ile add işlemi yapılarak sonuç Data Memory'nin Address'ine verilir. Belirtilen adrese Read data 2 yazılır. (Bknkz registers_first.mem'de 21. Satırdaki veri data_last.mem'de 6. Satıra yazılmış.)

- LUI

[illegible]

Lui instructionı gelince immediate part MSB 16 bit olur ve LSB 16 bit 0 olur. Elde edilen 32 bitlik sayı Rt'nin adresine yazılır. (Bknz register_last.mem'de 25. satır)

- BEQ ve BNE

```

instructions.txt - Not Defteri
Dosya Düzen Biçim Görünüm Yardım
000100_00000_11111_0000000000000001
000000_00001_00010_00011_00000_100110
000000_00100_00101_00110_00000_100101
000101_00000_00000_0000000000000001
000000_00111_01000_01001_00000_100100
000000_01010_01011_01100_00000_100010
000000_01101_01110_01111_00000_100000
001101_01010_10000_01100_00000_100010
100011_10001_10010_0000000000000000
101011_10011_10100_0000000000000001
001111_01101_10101_1100110011001100

```

bu şekilde çalışmaya devam eder. Ekran çıktıları aşağıdaki gibidir.

Bu instructionları test etmek için kullandığım test dosyası yandaki gibidir. Ekran görüntüsünde de görüldüğü gibi 1. instructionı beq ve 4. instruction bne instructionıdır. Program başladığında PC=0 iken bne instructionı okununca Branch sinyali 1 olur. $PC = PC + 4 + \text{BranchAddres}$ olacak. BranchAddres = 4 oluyor. Bu durumda PC = 8 olacak. Buna göre daha sonra 3. instruction çalışır. Daha sonra 4. instruction olan beq okunur. BranchNot sinyali 1 olur ve $PC = PC + 4 + \text{BranchAddres}$ eşit olur. Program

```

# Time= 0
# PC=00000000000000000000000000000000
# Instruction=00010000000111110000000000000001
# Opcode=000100
# rs=00000
# rt=11111
# rd=00000
# shamt=00000
# Function Code=000001
# Immediate=0000000000000001
# Mem_read=0
# Reg_Write1=0
# Reg_Write2=0
# Mem_Write=0
# RegDst=0
# Jump=0
# Jal=0
# Jr=0
# Branch=1
# BranchNot=0
# MemToReg=0
# ALUOp2bit=01,
# ALUOp3bit=110
# ALUSrc=0
# write_reg1=11111
# read_data1=000000000000000000000000000100
# read_data2=1010101011000001111101010011001
# write_data1=01010101001111100000010101101011
# write_data2=00000000000000000000000000000000
# registers[31]=1010101011000001111101010011001

```

```

# Time=50
# PC=000000000000000000000000000001000
# Instruction=000000001000010100110000000100101
# Opcode=000000
# rs=00100
# rt=00101
# rd=00110
# shamt=00000
# Function Code=100101
# Immediate=0011000000100101
# Mem_read=0
# Reg_Write1=1
# Reg_Write2=1
# Mem_Write=0
# RegDst=1
# Jump=0
# Jal=0
# Jr=0
# Branch=0
# BranchNot=0
# MemToReg=0
# ALUOp2bit=10,
# ALUOp3bit=001
# ALUSrc=0
# write_reg1=00100
# read_data1=01000100001010001100101000111100
# read_data2=01100101001010000000000001011011
# write_data1=01100101001010001100101010111111
# write_data2=0000000000000000000000000000011
# registers[31]=1010101011000001111101010011001

```

[illegible]

- Aşağıda bu test için kullanılan instruction dosyası verilmiştir. Görüldüğü üzere ilk instruction j instructionıdır. Bu instruction çalıştıktan sonra PC adresi JumpAddress olur. $\text{JumpAddr} = \{ \text{PC}+4[31:28], \text{address}, 2'b0 \}$ şeklinde hesaplanır. PC adresi 3. Instructionı gösterdiği için daha sonra 3. Instruction çalışır. Sonrasında 4. Instruction olan jal instructionı çalışır. $\text{PC}=\text{JumpAddress}$ olurken register[31]'e de PC+4 yazılır. Adımlar aşağıdaki ekran görüntülerin de gösterilmiştir.

instructions.txt - Not Defteri

Dosya	Düzen	Bişim	Görünüm	Yardım
000010_00000_00000_000000000000000010				
000000_00001_00010_00011_00000_100110				
000000_00100_00101_00110_00000_100101				
000011_00000_00000_0000000000000000101				
000000_00111_01000_01001_00000_100100				
000000_01010_01011_01100_00000_100010				
000000_00000_00001_00001_00000_001000				

```

# Time= 0
# PC=00000000000000000000000000000000
# Instruction=00001000000000000000000000000010
# Opcode=000010
# rs=000000
# rt=000000
# rd=000000
# shamt=000000
# Function Code=000010
# Immediate=0000000000000000
# Mem_read=0
# Reg_Write1=0
# Reg_Write2=0
# Mem_Write=0
# RegDst=0
# Jump=1
# Jal=0
# Jr=0
# Branch=0
# BranchNot=0
# MemToReg=0
# ALUOp2bit=00,
# ALUOp3bit=010
# ALUSrc=0
# write_reg1=000000
# read_data1=0000000000000000000000000000100
# read_data2=00000000000000000000000000000100
# write_data1=000000000000000000000000000001000
# write_data2=000000000000000000000000000000000
# registers[31]=10101010110000011111101010011001

```

```

# Time=50
# PC=000000000000000000000000000001000
# Instruction=00000000100001010011000000100101
# Opcode=000000
# rs=00100
# rt=00101
# rd=00110
# shamt=00000
# Function Code=100101
# Immediate=0011000000100101
# Mem_read=0
# Reg_Write1=1
# Reg_Write2=1
# Mem_Write=0
# RegDst=1
# Jump=0
# Jal=0
# Jr=0
# Branch=0
# BranchNot=0
# MemToReg=0
# ALUOp2bit=10,
# ALUOp3bit=001
# ALUSrc=0
# write_reg1=00100
# read_data1=01000100001010001100101000111100
# read_data2=01100101001010000000000010111011
# write_data1=01100101001010001100101010111111
# write_data2=0000000000000000000000000000011
# registers[31]=10101010110000011111101010011001

```

```

# Time=150
# PC=000000000000000000000000000001100
# Instruction=00001100000000000000000000000101
# Opcode=000011
# rs=000000
# rt=000000
# rd=000000
# shamt=00000
# Function Code=000101
# Immediate=0000000000000101
# Mem_read=0
# Reg_Write1=0
# Reg_Write2=0
# Mem_Write=0
# RegDst=0
# Jump=1
# Jal=1
# Jr=0
# Branch=0
# BranchNot=0
# MemToReg=0
# ALUOp2bit=00,
# ALUOp3bit=010
# ALUSrc=0
# write_reg1=000000
# read_data1=00000000000000000000000000000100
# read_data2=00000000000000000000000000000100
# write_data1=000000000000000000000000000001000
# write_data2=000000000000000000000000000000000
# registers[31]=10101010110000011111101010011001

```

(jump, j ve jal inst.ları
gelince 1 olur)

```

# Time=250
# PC=0000000000000000000000000000010100
# Instruction=00000001010010110110000000100010
# Opcode=000000
# rs=01010
# rt=01011
# rd=01100
# shamt=00000
# Function Code=100010
# Immediate=0110000000100010
# Mem_read=0
# Reg_Write1=1
# Reg_Write2=1
# Mem_Write=0
# RegDst=1
# Jump=0
# Jal=0
# Jr=0
# Branch=0
# BranchNot=0
# MemToReg=0
# ALUOp2bit=10,
# ALUOp3bit=110
# ALUSrc=0
# write_reg1=01010
# read_data1=00000000000000000000000000000110
# read_data2=01001101000001000000110100011000
# write_data1=10110010111110111111001011110110
# write_data2=000000000000000000000000000000010
# registers[31]=00000000000000000000000000000000

```

Not

- myAND32Bit ve myOR32Bit modülleri var. Bunları aralarda bazı işlemler için kullandım, ALU'nun işini yaptırmadım. ALU ile and ve or işlemlerini yapabileceğim o zaman aklıma gelmemiş ve bu modülleri yazmışım. Vakit olmadığı için bu modülleri silip işlemleri ALU'ya aktaramadım.
- Register ve dataların okunduğu dosyaların uzantısı txt olunca sorun oluşabiliyor. Dosyaların uzantısı mem olmalıdır.

