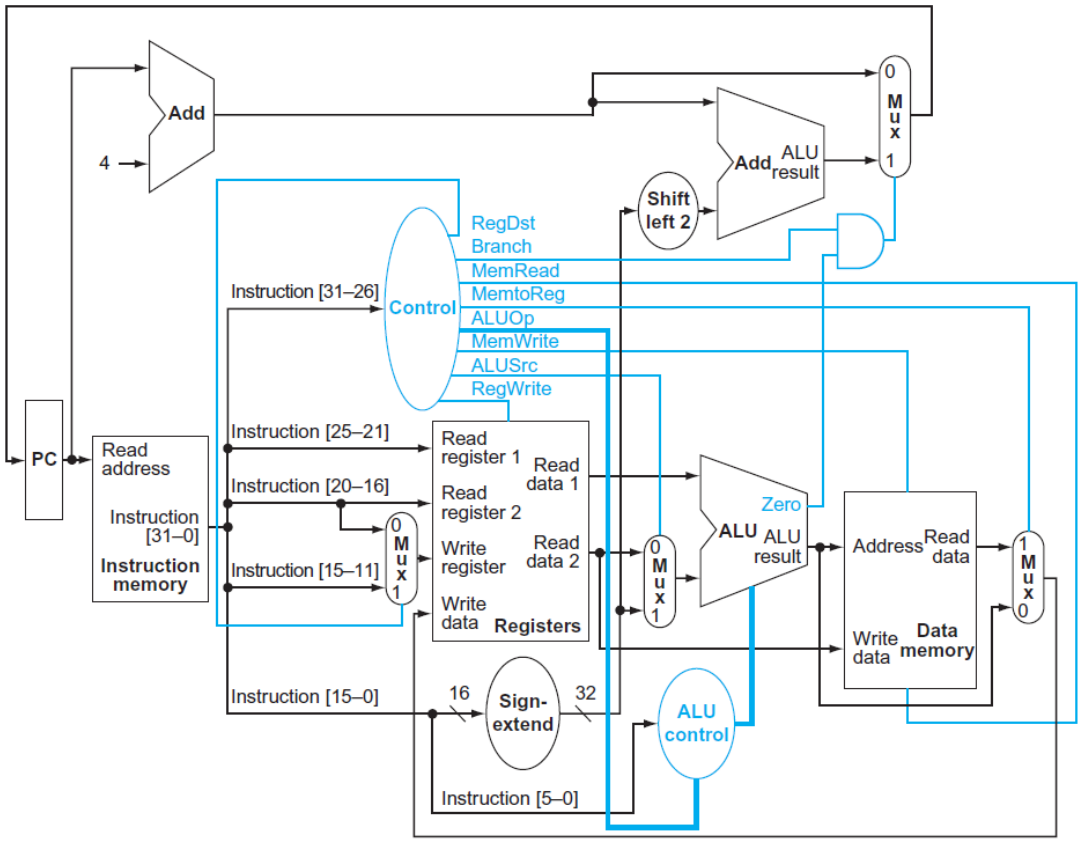
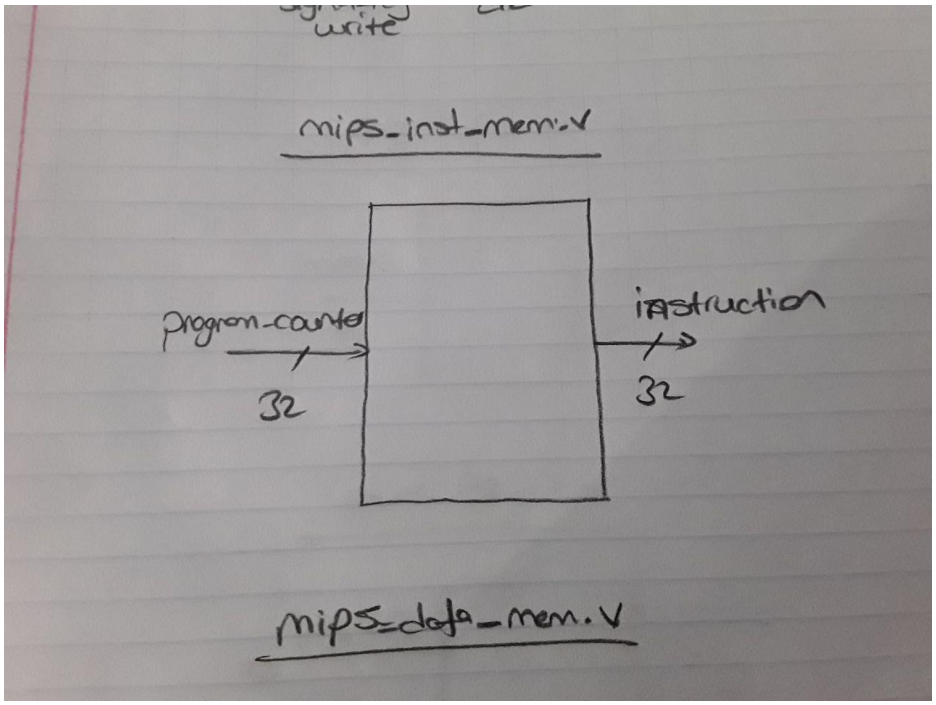
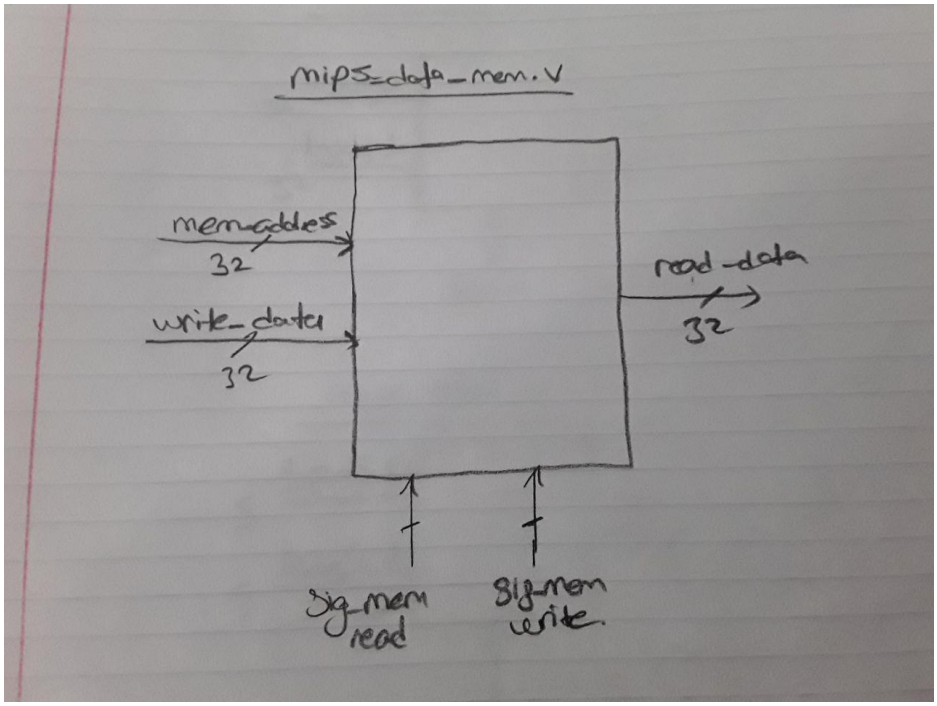
Bilgisayar Organizasyonu Final Projesi Raporu



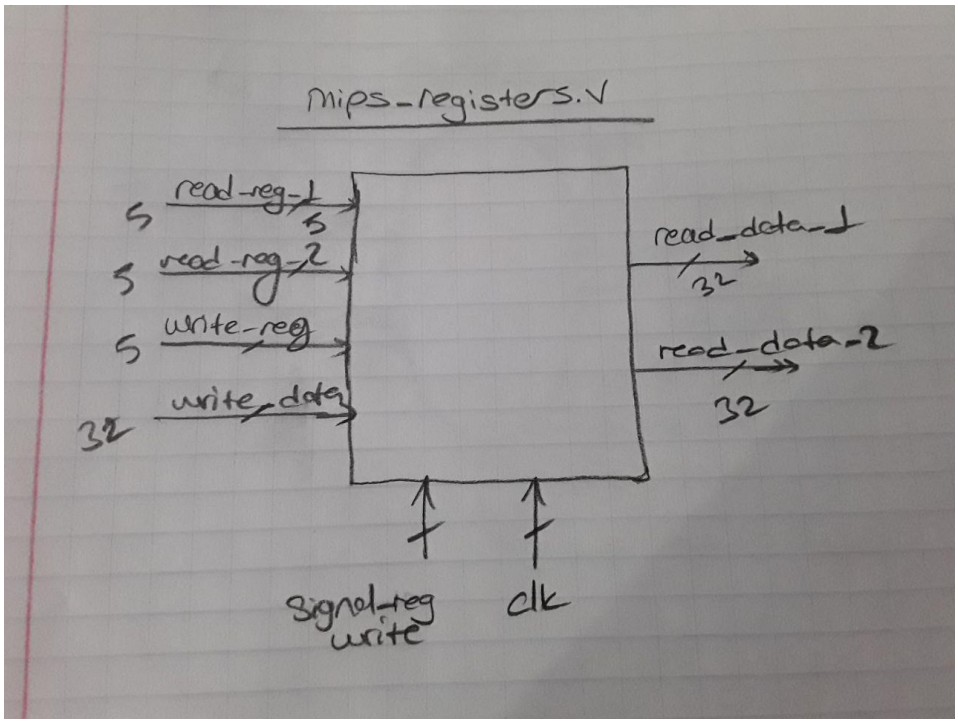
Uyguladığım single cycle datapath



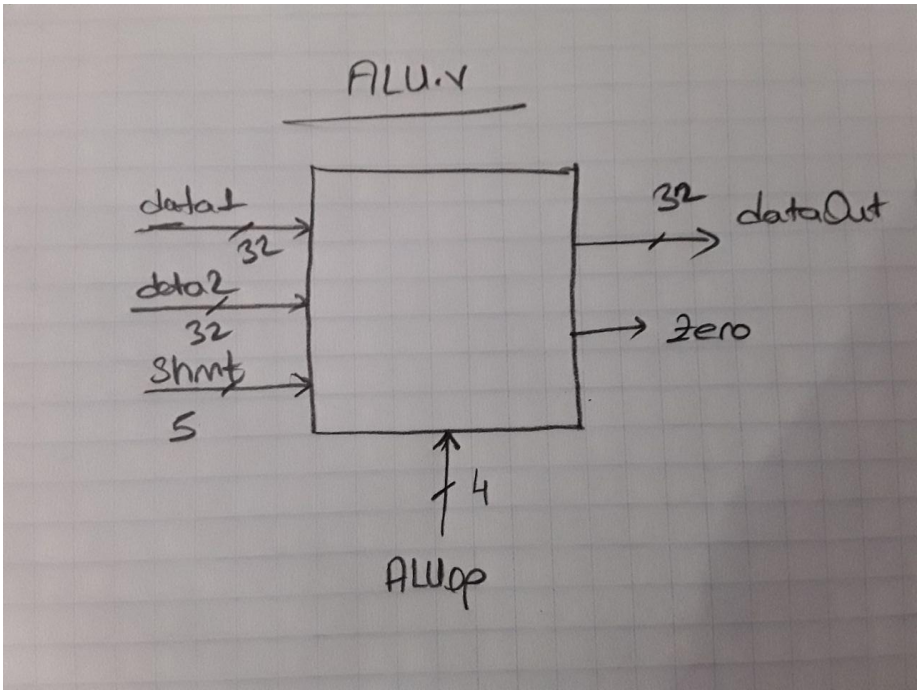
Mips_inst_mem.v: instruction.mem dosyasında tüm instructionları okur. program_counter girdisi ile instruction lar üzerinde ilerler. Ve instruction çıktısını üretir.



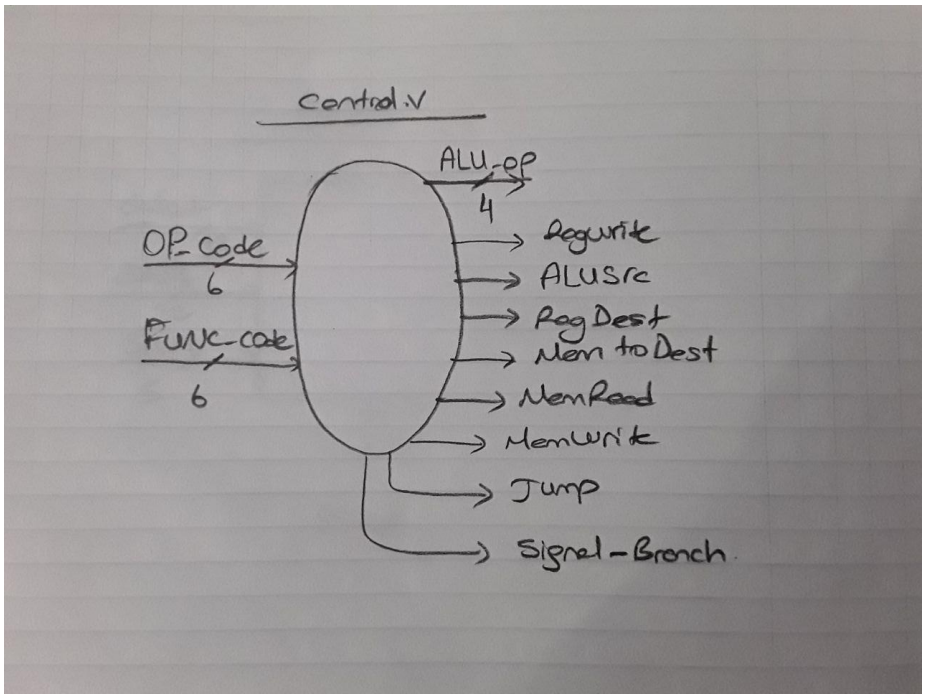
mips_data_mem.v: Data memory den dataların okunduğu modül. Mem_address, write_data sig_mem_read ve sig_mem_write verilerini alır, ve sinyal durumlarına göre eğer sig_mem_read ise verilen mem_address deki veriyi okur read_data ya assign eder. Eğer sig_mem_write ise verilen mem_address verisine write_data verisini yazar.



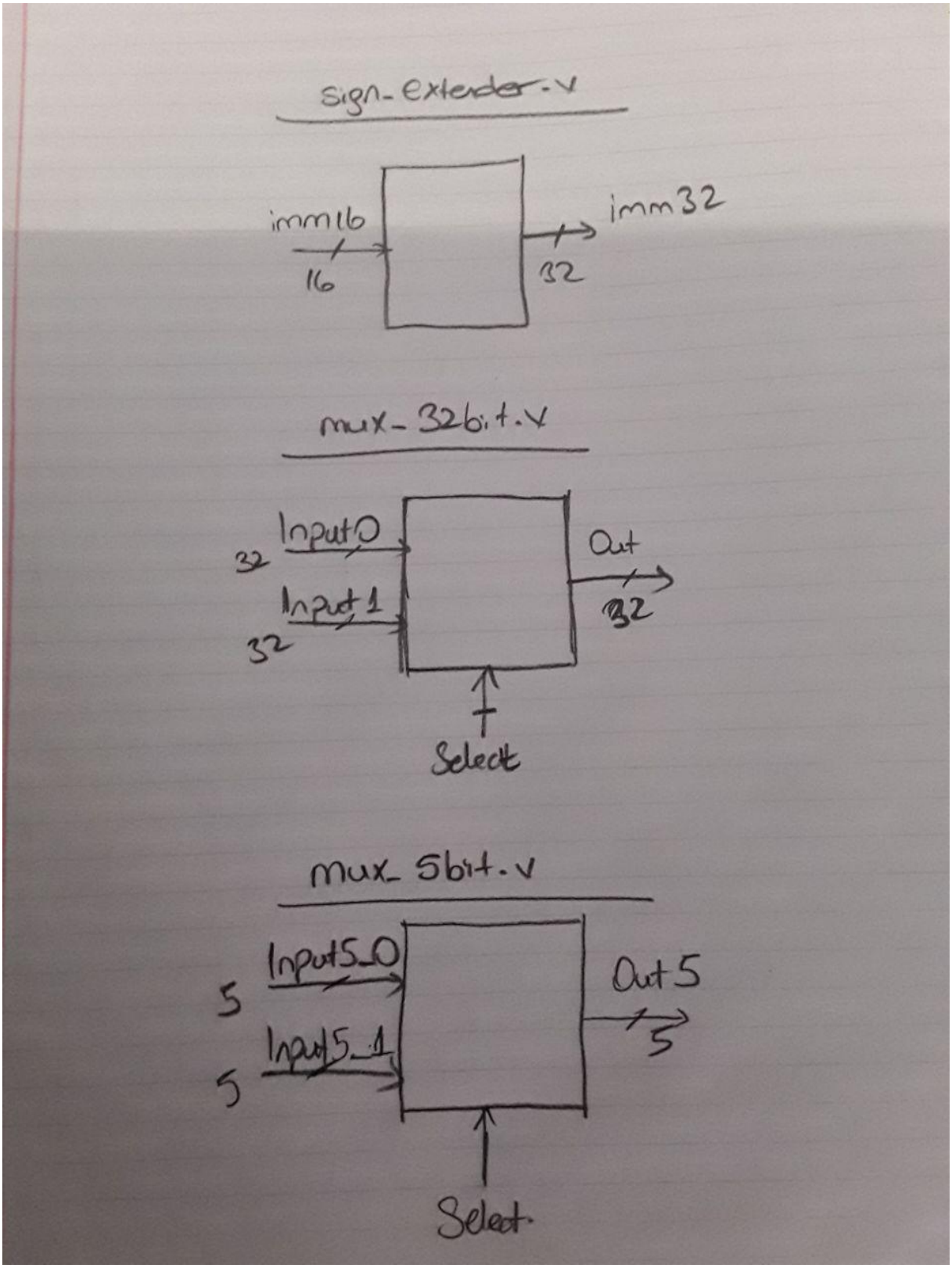
mips_registers.v: Okunan instructiona göre instruction parçalanır ve içindeki veriler ile register bloğundaki verilere ulaşılır.
Read_reg_1(rs),read_reg_2(rt),write_reg(rt/rd),write_data,signal_reg_write ve clk girdilerini alır ve read_data_1 ve read_data_2 çıktılarını verir. Signal_reg_write olması register bloğuna verinin write_reg adresine yazılır.



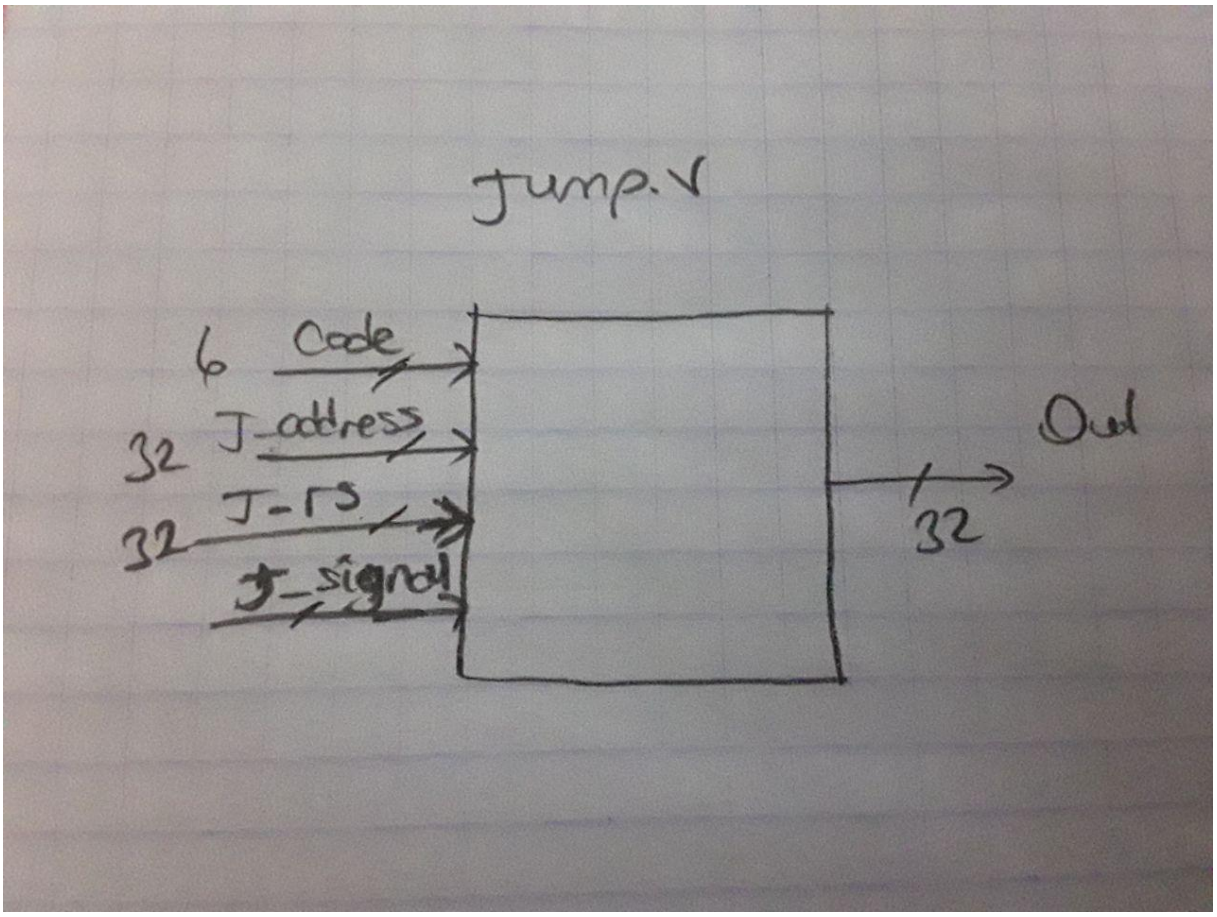
ALU.v: Datapath in tüm aritmetik ve lojik işlemlerini gerçekleştirdiği yerdir.
Data1,data2,shmt ve ALUop girdilerini alır ve dataOut ve zero çıktılarını üretir. ALUop bilgisine göre işlemini gerçekleştirir.



control.v : Datapath de işlenecek olan instructionun tüm sinyallerini bu birim üretir. OP_code ve FUNC_code bilgisini alır. Bu iki girdiye göre ALU nun kullanacağı ALU_op bilgisini ve datapath üzerinde ki diğer birimlerin kullanacağı gerekli sinyalleri üretir.



sign_Extender.v: 16 bitlik veriyi 32 bit e genişletir. imm16 girdisini 32 bite genişletip imm32 e atar.
mux_32bit.v: 32 bitlik iki girdiden select bitine göre seçim yapar.
mux_5bit.v: 5 bitlik iki girdiden select bitine göre seçim yapar.



jump.v: gelen code ve jump signaline göre instructionun jump yada jump register olup olmadığını anlar. Ona göre program counter ayarlanır.

TESTBENCH

add $15, $2, $3	-> 000000 00010 00011 01111 00000 100000
sub $16, $5, $4	-> 000000 00101 00100 10000 00000 100010
sll $17, $10, 3	-> 000000 00000 01010 10001 00011 000000
srl $18, $11, 3	-> 000000 00000 01011 10010 00011 000010
addi $19, $12, 11	-> 001000 01100 10011 00000 00000 001011
xor $20, $2, $3	-> 000000 00010 00011 10100 00000 100110
beq $8, $9, 2	-> 000100 01000 01001 00000 00000 000010
and $22, $2, $3	-> 000000 00010 00011 10110 00000 100100
or $23, $2, $3	-> 000000 00010 00011 10111 00000 100101
lw $24, $3($2)	-> 100011 00010 11000 00000 00000 000011
sw $25, $2($4)	-> 101011 00010 11001 00000 00000 000100

res_registers.mem | res_data.mem

```
1  00000000000000000000000000000000
2  00000000000000000000000000000001
3  00000000000000000000000000000010
4  00000000000000000000000000000011
5  00000000000000000000000000000100
6  00000000000000000000000000000101
7  00000000000000000000000000000110
8  00000000000000000000000000000111
9  00000000000000000000000000001000
10 00000000000000000000000000001000
11 00000000000000000000000000001010
12 00000000000000000000000000001011
13 00000000000000000000000000001100
14 00000000000000000000000000001101
15 00000000000000000000000000001110
16 00000000000000000000000000000101
17 00000000000000000000000000000001
18 00000000000000000000000001010000
19 00000000000000000000000000000001
20 00000000000000000000000000010111
21 00000000000000000000000000000001
22 00000000000000000000000000010101
23 00000000000000000000000000010110
24 00000000000000000000000000010111
25 00000000000000000000000000000101
26 00000000000000000000000000011001
27 00000000000000000000000000011010
28 00000000000000000000000000011011
29 00000000000000000000000000011100
30 00000000000000000000000000011101
31 00000000000000000000000000011110
32 00000000000000000000000000011111
33
```

res_register.mem

File Edit Selection Find View Goto Tools Projec

res_registers.mem | res_data.mem

```
1  00000000000000000000000000000000
2  00000000000000000000000000000001
3  00000000000000000000000000000010
4  00000000000000000000000000000011
5  00000000000000000000000000000100
6  00000000000000000000000000000101
7  00000000000000000000000000011001
8  00000000000000000000000000000111
9  00000000000000000000000000001000
10 00000000000000000000000000001001
11 00000000000000000000000000001010
12 00000000000000000000000000001011
13 00000000000000000000000000001100
14 00000000000000000000000000001101
15 00000000000000000000000000001110
16 00000000000000000000000000001111
17 00000000000000000000000000010000
18 00000000000000000000000000010001
19 00000000000000000000000000010010
```

res_data.mem

BEQ:

beq instructionunda $8 ve $9 registerlarının içerikleri eşit iken gösterilen adrese atlıyor. Sonrasında iki instruction çalışıyor.

```
# Loading work.jump
VSIM 169> run -all
# /INSTRUCTION: 00000000010000110111100000100000/OP_code 000000/FUNC_code 100000/Branch 0/ALU_op 0000
00000000000000000000000000000011/RD adress: 01111/RESULT 00000000000000000000000000000101
# /INSTRUCTION: 00000000101001001000000000100010/OP_code 000000/FUNC_code 100010/Branch 0/ALU_op 0110
00000000000000000000000000000100/RD adress: 10000/RESULT 00000000000000000000000000000001
# /INSTRUCTION: 00000000000010101000100011000000/OP_code 000000/FUNC_code 000000/Branch 0/ALU_op 0011
00000000000000000000000000001010/RD adress: 10001/RESULT 00000000000000000000000001010000
# /INSTRUCTION: 00000000000010111001000011000010/OP_code 000000/FUNC_code 000010/Branch 0/ALU_op 0100
00000000000000000000000000001011/RD adress: 10010/RESULT 00000000000000000000000000000001
# /INSTRUCTION: 00100001100100110000000000001011/OP_code 001000/FUNC_code 001011/Branch 0/ALU_op 0000
00000000000000000000000000010011/RD adress: 00000/RESULT 00000000000000000000000000010111
# /INSTRUCTION: 00000000010000111010000000100110/OP_code 000000/FUNC_code 100110/Branch 0/ALU_op 1011
00000000000000000000000000000011/RD adress: 10100/RESULT 00000000000000000000000000000001
# /INSTRUCTION: 00010001000010010000000000000010/OP_code 000100/FUNC_code 000010/Branch 1/ALU_op 1001
00000000000000000000000000001000/RD adress: 00000/RESULT 00000000000000000000000000000000
# /INSTRUCTION: 10001100010110000000000000000011/OP_code 100011/FUNC_code 000011/Branch 0/ALU_op 0000
00000000000000000000000000011000/RD adress: 00000/RESULT 00000000000000000000000000000101
# /INSTRUCTION: 10101100010110010000000000000100/OP_code 101011/FUNC_code 000100/Branch 0/ALU_op 0000
00000000000000000000000000011001/RD adress: 00000/RESULT 00000000000000000000000000000110
# /INSTRUCTION: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx/OP_code xxxxxx/FUNC_code xxxxxx/Branch x/ALU_op 0000
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx/RD adress: xxxxx/RESULT 00000000000000000000000000000110
```

$8 ve $9 un içeriği eşit olmadığında ise verilen adrese geçmiyor. Tüm instructionlar çalışmış oluyor.

```
# Loading work.mips_data_mem
# Loading work.jump
VSIM 171> run -all
# /INSTRUCTION: 00000000010000110111100000100000/OP_code 000000/FUNC_code 100000/Branch 0/ALU_
00000000000000000000000000000011/RD adress: 01111/RESULT 00000000000000000000000000000101
# /INSTRUCTION: 00000000101001001000000000100010/OP_code 000000/FUNC_code 100010/Branch 0/ALU_
00000000000000000000000000000100/RD adress: 10000/RESULT 00000000000000000000000000000001
# /INSTRUCTION: 00000000000010101000100011000000/OP_code 000000/FUNC_code 000000/Branch 0/ALU_
00000000000000000000000000001010/RD adress: 10001/RESULT 00000000000000000000000001010000
# /INSTRUCTION: 00000000000010111001000011000010/OP_code 000000/FUNC_code 000010/Branch 0/ALU_
00000000000000000000000000001011/RD adress: 10010/RESULT 00000000000000000000000000000001
# /INSTRUCTION: 00100001100100110000000000001011/OP_code 001000/FUNC_code 001011/Branch 0/ALU_
00000000000000000000000000010011/RD adress: 00000/RESULT 00000000000000000000000000010111
# /INSTRUCTION: 00000000010000111010000000100110/OP_code 000000/FUNC_code 100110/Branch 0/ALU_
00000000000000000000000000000011/RD adress: 10100/RESULT 00000000000000000000000000000001
# /INSTRUCTION: 00010001000010010000000000000010/OP_code 000100/FUNC_code 000010/Branch 1/ALU_
00000000000000000000000000001001/RD adress: 00000/RESULT 11111111111111111111111111111111
# /INSTRUCTION: 00000000010000111011000000100100/OP_code 000000/FUNC_code 100100/Branch 0/ALU_
00000000000000000000000000000011/RD adress: 10110/RESULT 00000000000000000000000000000010
# /INSTRUCTION: 00000000010000111011100000100101/OP_code 000000/FUNC_code 100101/Branch 0/ALU_
00000000000000000000000000000011/RD adress: 10111/RESULT 00000000000000000000000000000011
# /INSTRUCTION: 10001100010110000000000000000011/OP_code 100011/FUNC_code 000011/Branch 0/ALU_
00000000000000000000000000011000/RD adress: 00000/RESULT 00000000000000000000000000000101
# /INSTRUCTION: 10101100010110010000000000000100/OP_code 101011/FUNC_code 000100/Branch 0/ALU_
00000000000000000000000000011001/RD adress: 00000/RESULT 00000000000000000000000000000110
# /INSTRUCTION: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx/OP_code xxxxxx/FUNC_code xxxxxx/Branch x/ALU_
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx/RD adress: xxxxx/RESULT 00000000000000000000000000000110

VSIM 172>
```

NOT: İmplement ettiğim instructionların hepsini testbench de yazmadım ama çalışıyorlar. Ayrıca (JAL) instructionunu implement etmedim. Ama ödevde istenen instructionların dışında (XOR ve XORI) instructionlarını implement ettim.
Birde register.mem ve data.mem için okuma işlemi için kendi bilgisayarımda çalıştığım path dizinini verdim. Sizin ayarladığınız şekilde okuma işlemini yapamıyordum hocam.

Mustafa BİNGÜL
141044077