Gebze Technical University
Computer Engineering

CSE 331

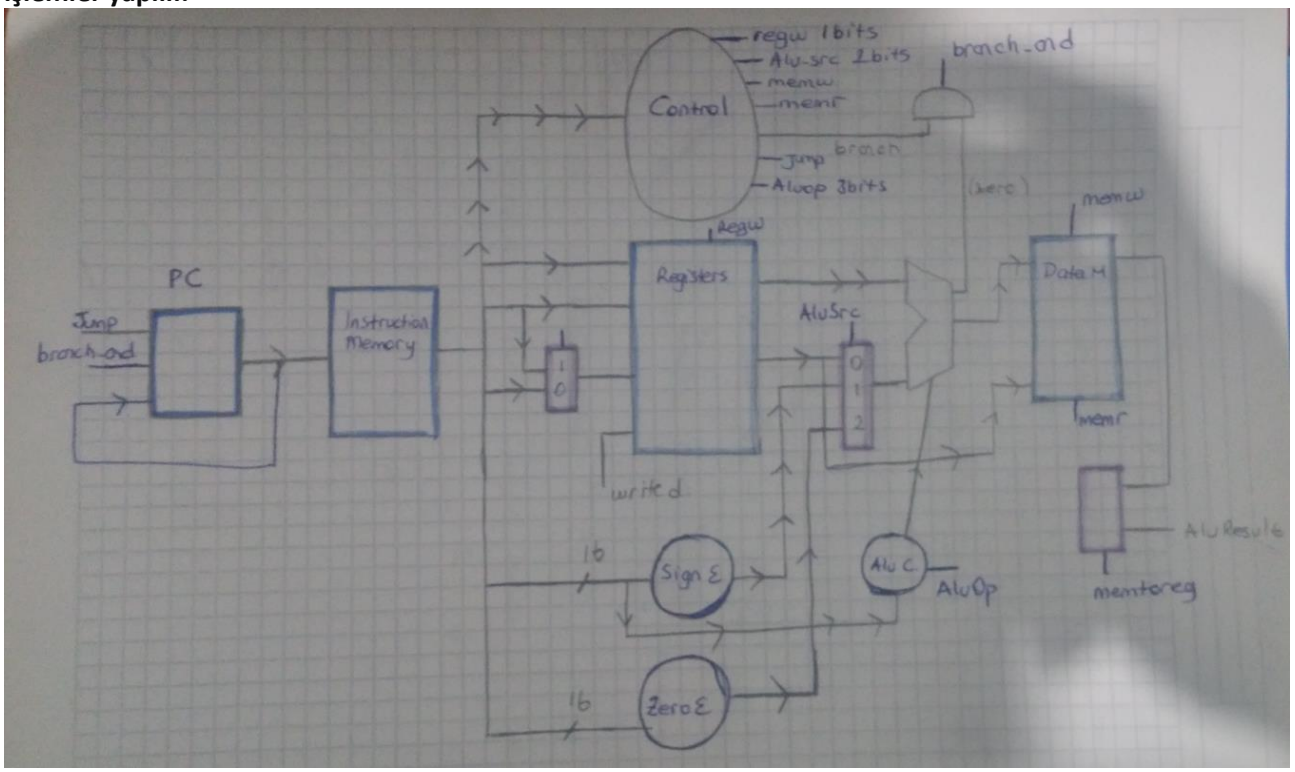
ASSIGNMENT 4 REPORT

AKIN ÇAM
151044007

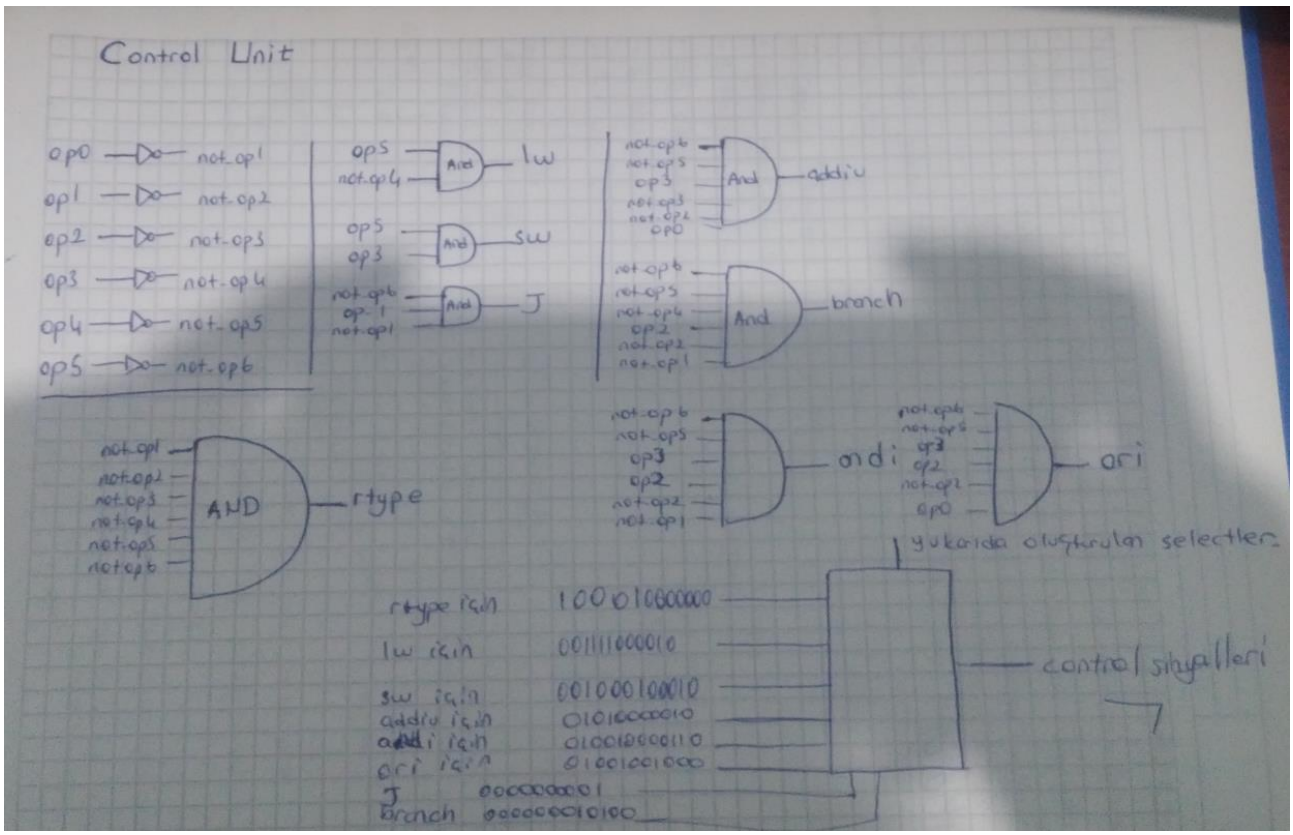
Course Assistant:

1. Schematic Design for All Modules.

R-type I-type jump ve branch instructionları için çalışan single cycle processor tasarlanmıştır.
Program counter adresine göre instruction memoryden instruction okunur.
Opcode kullanılarak control unit sinyalleri belirlenmiştir.
Bölünen instruction ile birlikte register contentleri register blocktan okunur.
Sign extend ve Zero extend işlemleri gerçekleştirilmiştir.
Alu src sinyaline göre al unun ikinci inputu belirlenmiştir.(0 r type-1 signextend-2 zero extend).
Hesaplanan işlem data memory e ve çıkıştaki mux a bağlanmıştır. Load Word ve store Word için data memory kullanılır ve memtoreg bunu seçer(load Word için). R type ya da load Word için content register a yazılır.
Registerların son halleri test açısından farklı bir dosyaya yazılmıştır.(registers_result).
Program counter a jump bir önceki instruction ve alunun zero sonucu ile branch sinyalinin and işleminin sonucu gönderilmiştir.
2 tane clock kullanılmıştır. Birinci clock un posedge kısmında pc hesaplanır negedge kısmında instruction memory den okunur. Bu clock süresi diğerine göre daha uzundur. İkinci clock un posedge kısmında register dan contentler okunur ve negedge kısmında data memory işlemleri var ise bu işlemler yapılır.



Control_unit:



Sinyaller:

	Reg Dest	Alu Src	Mem to Reg	Reg write	Mem Rd	Mem Wr	Branch	AluOp	AluOp	AluOp	J	Opcode
Add	1	0	0	1	0	0	0	0	0	0	0	000000
Addu	1	0	0	1	0	0	0	0	0	0	0	000000
And	1	0	0	1	0	0	0	0	0	0	0	0
Nor	1	0	0	1	0	0	0	0	0	0	0	0
Or	1	0	0	1	0	0	0	0	0	0	0	0
sltu	1	0	0	1	0	0	0	0	0	0	0	0
sll	1	0	0	1	0	0	0	0	0	0	0	0
srl	1	0	0	1	0	0	0	0	0	0	0	0
sub	1	0	0	1	0	0	0	0	0	0	0	0
subu	1	0	0	1	0	0	0	0	0	0	0	0
Addiu	0	1	0	1	0	0	0	0	0	1	0	001001
lw	0	1	1	1	1	0	0	0	0	1	0	100011
sw	X	1	X	0	0	1	0	0	0	1	0	101011
branch	X	0	X	0	0	0	1	0	1	0	0	000100
andi	0	2	0	1	0	0	0	0	1	1		001100
ori	0	2	0	1	0	0	0	1	0	0		001101
J	—	—	—	0	—	0	—	—	—	—		000001

Zero_extend:

Andi ve ori instrucitonları için zero extend yapılır. 32 bit haline getirme işlemi most significant 16 bitine 0 eklenmesiyle oluşur. Alu src bu durumda 10 sinyali gönderilir ve zero extend seçilir.

Sign_extend:

I type instructionlar için sign extend yapılır. 32 bit haline getirme işlemi most significant 16. Bitin genişletilmesi ile oluşur. Alu src bu durumda 01 sinyali gönderilir.

Register_block:

Behavioral olarak oluşturulmuştur.
Test açısından yazma işlemi farklı bir dosyaya gerçekleştirilmektedir.
Zero bitine yazılmama durumu kontrol edilmiştir.(zero biti hep 0 kalacaktır.)
Clock(ikinci) un posedge kısmında register contentleri okunmaktadır.
Eğer işlem shift ise rt registerinin least significant 5 biti alınır 32 bit extend edilir. Çünkü rs rt kadar shift yapılır ve rs 32 bittir.
Eğer regw sinyali 1 ise register_results dosyasına register contentlerinin yazma işlemi gerçekleştirilir.

Next_PC:

1. **Clock un posedge kısmında instruction adresi belirlenmektedir.**

Program başlangıcı için ilk adres 0 olarak belirlenmiştir.
Jump branch ve bir önceki instruction ı input olarak almaktadır.
Eğer işlem normal ise bir sonraki index te bulunan adres gönderilir.
Eğer işlem jump ise

```
temp1= res+32'b00000000000000000000000000000001;
        res[31:28]=temp1[31:28];
        res[27:2]=inst[25:0];
        res[1]=1'b0;
        res[0]=1'b0; işlemi gerçekleştirilir.
```

Eğer işlem branch ise

```
temp2[31:15]=inst[15];
        temp2[14:0]=inst[14:0];
        temp3[31:2]=temp2[29:0];
        temp3[1]=1'b0;
        temp3[0]=1'b0;
        res=temp3; işlemi gerçekleştirilir.
```

İnstruction_memory:

1. **Clock un negedge kısmında instruction pc den verilen adrese göre okuma işlemi gerçekleştirilir.**

Data_memory:

2. **Clock un negedge kısmında bu işlemler gerçekleştirilir.**

Eğer mem read ise verilen adresin contenti okunur.
Eğer mem write ise verilen içerik adrese yazılır.

Control_unit:

Opcode a göre kontrol sinyalleri belirlenmiştir.
Regdest,alu_src(2bit),mem_to_reg,regw,memread,memwrite,branch, aluop(3 bits), jump.
r type için 1 00 0 1 0 0 0 000 0
lw 0 01 1 1 1 0 0 001 0

sw x 01 x 0 0 1 0 001 0
addiu 0 01 0 1 0 0 0 001 0
branch x 00 x 0 0 0 1 010 0
andi 0 10 0 1 0 0 0 011 0
ori 0 10 0 1 0 0 0 100 0
j 0 00 0 0 0 0 0 000 1
opcode a göre sinyaller belirlenmiştir.
alu src r type için mux a 00
alu src i type için mux a 01
alu src andi ori için mux 10 olmaktadır.

Alu_control:
rtype için result 000 ile 111 e kadar bir önceki ödevdeki gibi aynıdır.
lw sw ve addiuu da toplama yapıldığı için 001
branch için 100(çıkarma).
andi için and kodu 000
ori için or kodu 001
işlemler function code a göre belirlenmiştir.

mips32_single_cycle:
ilk önce birinci clk un posedge kısmında adress hesaplanır.
birinci clk un negedge kısmında instruction okunur.
verilen instruction hem r type hem de i type için parçalanır. Uygun parçalar işleme göre kullanılacaktır.
control unit sinyalleri belirlenir
alunun select bitleri belirlenir.(alu control).
i type için instruciton ın least significant 16 biti 16. biti ile extend edilir.
andi ve ori için instruciton ın least significant 16 biti 0 ile extend edilir.
register da rd nin rt mi rd mi olduğur reg_dest sinyali ile belirlenir.(mux ile).
registerdan register contentleri okunur.
aluya girecek ikinci parametre belirlenir. rtype-itype-(andi ori).(rt,signed_extend,zero_extend)(mux)
işlem alu da hesaplanır.
alunun zero sonucu kontrol edilir ve işlem branch ise branch sinyali belirlenir.
işlemin set less than unsigned olup olmadığına bakılır.
sltu ise eşit olup olmamasına göre 0 ve ya 1 extend edilir. değil ise normal sonuç seçilir.
data memory sw lw için kullanılacaktır. 2. clock un negatif edge kısmında çalışır.
data memoryden gelen değer mi yoksa alu sonucu mu rd ye yazılmak isteniyorsa yazılacak bu belirlenir.(mux).
write signal 1 ise 2.clk un positive edge kısmında yazma gerçekleşir.

**********TEST SONUÇLARI**********
step -current

#
#
start program counter operation--> 00000000000000000000000000000000
#
instruction is--->00000000001000010001000000100000
reg_dest--->1
alu_src--->00
mem_to_reg--->0
reg_write--->1
memr--->0 nmemw--->0

```
#  aluop--->000
#  branch--->0
#  jump--->0
# alu result is---->00000000000000000000000001111110
# ------------------------------------------
#
#
# normal program counter operation--> 00000000000000000000000000000001
#
#  instruction is--->00000000001000010001100000100001
#  reg_dest--->1
#  alu_src--->00
#  mem_to_reg--->0
#  reg_write--->1
#  memr--->0 nmemw--->0
#  aluop--->000
#  branch--->0
#  jump--->0
# alu result is---->00000000000000000000000001111110
# ------------------------------------------
#
#
# normal program counter operation--> 00000000000000000000000000000010
#
#  instruction is--->00000000001000010010000000100100
#  reg_dest--->1
#  alu_src--->00
#  mem_to_reg--->0
#  reg_write--->1
#  memr--->0 nmemw--->0
#  aluop--->000
#  branch--->0
#  jump--->0
# alu result is---->00000000000000000000000000111111
# ------------------------------------------
#
#
# normal program counter operation--> 00000000000000000000000000000011
#
#  instruction is--->00000000001000010010100000100111
#  reg_dest--->1
#  alu_src--->00
#  mem_to_reg--->0
#  reg_write--->1
#  memr--->0 nmemw--->0
#  aluop--->000
#  branch--->0
#  jump--->0
# alu result is---->11111111111111111111111111000000
```

```
# ----------------------------------------
#
#
# normal program counter operation--> 00000000000000000000000000000100
#
#  instruction is--->00000000001000010011000000100101
#  reg_dest--->1
#  alu_src--->00
#  mem_to_reg--->0
#  reg_write--->1
#  memr--->0 nmemw--->0
#  aluop--->000
#  branch--->0
#  jump--->0
# alu result is---->00000000000000000000000000111111
# ----------------------------------------
#
#
# normal program counter operation--> 00000000000000000000000000000101
#
#  instruction is--->00000000000000010011100000101011
#  reg_dest--->1
#  alu_src--->00
#  mem_to_reg--->0
#  reg_write--->1
#  memr--->0 nmemw--->0
#  aluop--->000
#  branch--->0
#  jump--->0
# alu result is---->00000000000000000000000000000001
# ----------------------------------------
#
#
# normal program counter operation--> 00000000000000000000000000000110
#
#  instruction is--->00000000001000010100000011000000
#  reg_dest--->1
#  alu_src--->00
#  mem_to_reg--->0
#  reg_write--->1
#  memr--->0 nmemw--->0
#  aluop--->000
#  branch--->0
#  jump--->0
# alu result is---->00000000001111110000000000000000
# ----------------------------------------
#
#
# normal program counter operation--> 00000000000000000000000000000111
```

```
#
#  instruction is--->00000000001000010100100011000010
#  reg_dest--->1
#  alu_src--->00
#  mem_to_reg--->0
#  reg_write--->1
#  memr--->0 nmemw--->0
#  aluop--->000
#  branch--->0
#  jump--->0
# alu result is---->00000000000000000000000000000000
# ------------------------------------------
#
#
# normal program counter operation--> 00000000000000000000000000001000
#
#  instruction is--->00000000000000010101000000100010
#  reg_dest--->1
#  alu_src--->00
#  mem_to_reg--->0
#  reg_write--->1
#  memr--->0 nmemw--->0
#  aluop--->000
#  branch--->0
#  jump--->0
# alu result is---->11111111111111111111111111000001
# ------------------------------------------
#
#
# normal program counter operation--> 00000000000000000000000000001001
#
#  instruction is--->00000000001000010101100000100011
#  reg_dest--->1
#  alu_src--->00
#  mem_to_reg--->0
#  reg_write--->1
#  memr--->0 nmemw--->0
#  aluop--->000
#  branch--->0
#  jump--->0
# alu result is---->00000000000000000000000000000000
# ------------------------------------------
#
#
# normal program counter operation--> 00000000000000000000000000001010
#
#  instruction is--->00100100001011000000000000000001
#  reg_dest--->0
#  alu_src--->01
```

```
#  mem_to_reg--->0
#  reg_write--->1
#  memr--->0 nmemw--->0
#  aluop--->001
#  branch--->0
#  jump--->0
# alu result is---->00000000000000000000000001000000
# ------------------------------------------
#
#
# normal program counter operation--> 00000000000000000000000000001011
#
#  instruction is--->10001100001011010000000000000001
#  reg_dest--->0
#  alu_src--->01
#  mem_to_reg--->1
#  reg_write--->1
#  memr--->1 nmemw--->0
#  aluop--->001
#  branch--->0
#  jump--->0
# data memory read---->00000000000000000000000000000000
# alu result is---->00000000000000000000000001000000
# ------------------------------------------
#
#
# normal program counter operation--> 00000000000000000000000000001100
# data memory read---->00000000000000000000000000000000
# data memory read---->00000000000000000000000000000000
#
#  instruction is--->10101100001000010000000000000001
#  reg_dest--->0
#  alu_src--->01
#  mem_to_reg--->0
#  reg_write--->0
#  memr--->0 nmemw--->1
#  aluop--->001
#  branch--->0
#  jump--->0
# data memory write---->00000000000000000000000000111111
# alu result is---->00000000000000000000000001000000
# ------------------------------------------
#
#
# normal program counter operation--> 00000000000000000000000000001101
# data memory write---->00000000000000000000000000111111
# data memory write---->00000000000000000000000000111111
#
#  instruction is--->00010000001000010000000000000100
```

```
#  reg_dest--->0
#  alu_src--->00
#  mem_to_reg--->0
#  reg_write--->0
#  memr--->0 nmemw--->0
#  aluop--->010
#  branch--->1
#  jump--->0
# alu result is---->00000000000000000000000000000000
# ------------------------------------------
#
#
# branch address operation--> 00000000000000000000000000010000
#
#  instruction is--->00110000010000100000000000000001
#  reg_dest--->0
#  alu_src--->10
#  mem_to_reg--->0
#  reg_write--->1
#  memr--->0 nmemw--->0
#  aluop--->011
#  branch--->0
#  jump--->0
# alu result is---->00000000000000000000000000000001
# ------------------------------------------
#
#
# normal program counter operation--> 00000000000000000000000000010001
#
#  instruction is--->00110100001011110000000000000111
#  reg_dest--->0
#  alu_src--->10
#  mem_to_reg--->0
#  reg_write--->1
#  memr--->0 nmemw--->0
#  aluop--->100
#  branch--->0
#  jump--->0
# alu result is---->00000000000000000000000000111111
# ------------------------------------------
#
#
# normal program counter operation--> 00000000000000000000000000010010
#
#  instruction is--->00001000000000000000000000000001
#  reg_dest--->0
#  alu_src--->00
#  mem_to_reg--->0
#  reg_write--->0
```

```
#  memr--->0 nmemw--->0
#  aluop--->000
#  branch--->0
#  jump--->1
# alu result is---->00000000000000000000000000000000
# ------------------------------------------
#
#
# jump address operation--> 00000000000000000000000000000100
#
#  instruction is--->00000000001000010011000000100101
#  reg_dest--->1
#  alu_src--->00
#  mem_to_reg--->0
#  reg_write--->1
#  memr--->0 nmemw--->0
#  aluop--->000
#  branch--->0
#  jump--->0
# alu result is---->00000000000000000000000000111111
# ------------------------------------------
#
#
# normal program counter operation--> 00000000000000000000000000000101
#
#  instruction is--->00000000000000010011100000101011
#  reg_dest--->1
#  alu_src--->00
#  mem_to_reg--->0
#  reg_write--->1
#  memr--->0 nmemw--->0
#  aluop--->000
#  branch--->0
#  jump--->0
# alu result is---->00000000000000000000000000000001
# ------------------------------------------
#
#
# normal program counter operation--> 00000000000000000000000000000110
```

İnstructions:

```
00000000001000010001000000100000
00000000001000010001100000100001
00000000001000010010000000100100
00000000001000010010100000100111
00000000001000010011000000100101
00000000000000010011100000101011
00000000001000010100000011000000
00000000001000010100100011000010
00000000000000010101000000100010
00000000001000010101100000100011
00100100001011000000000000000001
10001100001011010000000000000001
10101100001000010000000000000001
00010000001000010000000000000100
00110000001000010000000000000001
00110100001011110000000000000111
00110000001000010000000000000001
00110100001011110000000000000111
00001000000000000000000000000001
```

Data Memory before instructions:

```
 4   00000000000000000000000000000000
 5   00000000000000000000000000000000
 6   00000000000000000000000000000000
 7   00000000000000000000000000000000
 8   00000000000000000000000000000000
 9   00000000000000000000000000000000
10   00000000000000000000000000000000
11   00000000000000000000000000000000
12   00000000000000000000000000000000
13   00000000000000000000000000000000
14   00000000000000000000000000000000
15   00000000000000000000000000000000
16   00000000000000000000000000000000
17   00000000000000000000000000000000
18   00000000000000000000000000000000
19   00000000000000000000000000000000
20   00000000000000000000000000000000
21   00000000000000000000000000000000
22   00000000000000000000000000000000
23   00000000000000000000000000000000
24   00000000000000000000000000000000
25   00000000000000000000000000000000
26   00000000000000000000000000000000
27   00000000000000000000000000000000
28   00000000000000000000000000000000
29   00000000000000000000000000000000
30   00000000000000000000000000000000
31   00000000000000000000000000000000
32   00000000000000000000000000000000
33   00000000000000000000000000000000
34   00000000000000000000000000000000
35   00000000000000000000000000000000
```

Data memory after instructions:

```
00000000000000000000000000111111
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

Registers before instructions:

```
00000000000000000000000000000000
00000000000000000000000000111111
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

Registers after instructions:

00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000001111110
00000000000000000000000001111110
00000000000000000000000000111111
11111111111111111111111111000000
00000000000000000000000000111111
00000000000000000000000000000001
00000000001111110000000000000000
00000000000000000000000000000000
11111111111111111111111111000001
00000000000000000000000000000000
00000000000000000000000001000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000111111
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000