
**T.C.**

**GEBZE TEKNİK ÜNİVERSİTESİ**

**Bilgisayar Mühendisliği Bölümü**

# BİL -331

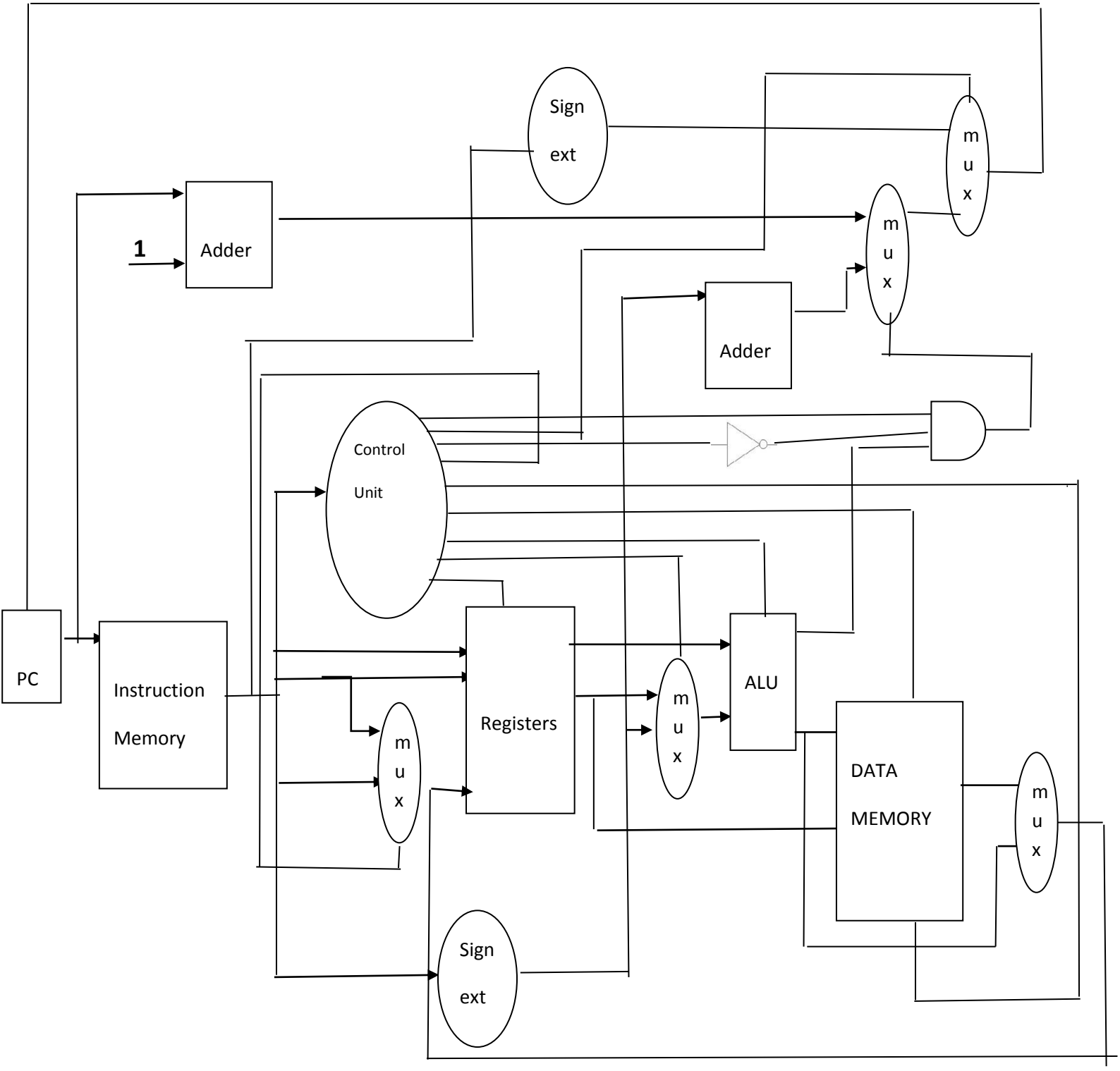
# MIPS Single Cycle Processor

**Efkan Duraklı**

**Aralık, 2017**
**Gebze,KOCAELİ**

# 1. INTRODUCTION

## 1.1 Big Picture



Tasarım genel olarak yukarıdaki gibi çalışmaktadır.

Kullanılan kontrol sinyalleri **branch**, **bneq**, **MemRead**, **MemWrite**, **ALUSrc**, **RegWrite**, **RegDest**, **jump**, **jal**, **ALUOp**.

ALUOp sinyali 3 bitlik bir sinyaldir ve ALU'nun yapacağı işlemi belirten sinyaldir.

Processorun desteklediği Instructionlar.

**R-Type :** add, addu, and, jr, nor, or, slt, sltu, sll, srl, sub, subu, sra

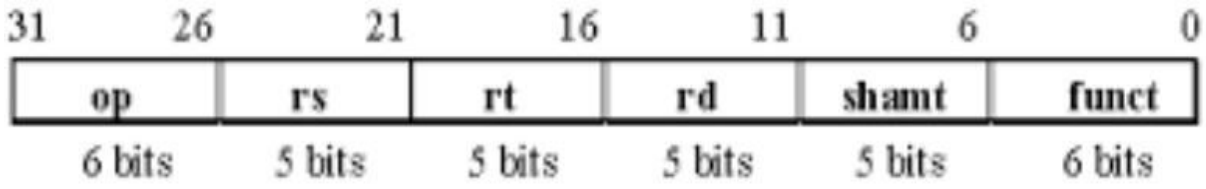
**I-Type:** addi, addiu, andi, beq, bneq, ll, lui, lw, ori, slti, sltui, sw

**J-Type:** j, jal

## 1.2 Life Cycle of 1 Instruction

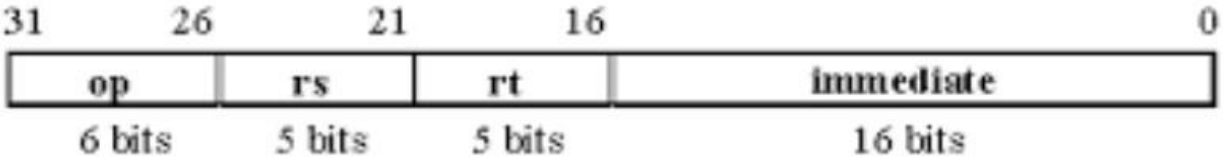
Bütün Instructionlar Insruction Fetch aşamasını gerçekleştirmektedir.

### 1.2.1 Life Cycle of R-Type Instruction




R-Type Instructionlar yukarıdaki gibi fetch edilir.5 bitlik rs ve rt sinyallerinin gösterdiği register adreslerinden 32 bitlik datalar okunur.Bu okunan datalar ALU modülüne input olarak gelir.ALU modülünde function koduna göre işlem gerçekleştirilir.ALU modülünün çıkışındaki değer cycle sonunda 5 bitlik rd sinyalinin gösterdiği registera yazılır.Program counter cycle sonunda eğer instruction jr değilse 1 artrılır.Jr instructionunda program countera rs sinyalinin gösterdiği register adresindeki data yazılır.

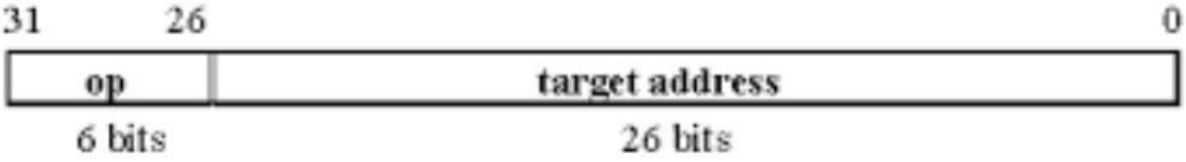
### 1.2.2 Life Cycle of I-Type Instruction




I-Type Instructionlar yukarıdaki gibi fetch edilir.5 bitlik rs ve rt sinyallerinin gösterdiği register adreslerinden 32 bitlik datalar okunur.Instructionda bulunan 16 bitlik immediate alanı 32 bite sign biti dikkate alınarak extend edilir.ALU nun girişine ALUSrc sinyalinin 1 olmasından dolayı extend edilmiş data ve rs sinyalinim gösterdiği registerdan okunan data gelir.ALU modülü input olarak aldığı datalarla ALUOp sinyaline göre işlemi yapar.Memoryden okuma

ve yazma yapan lw ve sw gibi instruvctionlarda ALU'nun çıkışındaki değer memoryden okunacak datanın adresini ya da memorye yazılcak datanın adresini gösterir.Kontrol sinyaline göre memoryden okunan data ya da ALU'da hesaplanan data 5 bitlik rt sinyalinin gösterdiği register adresine yazılır.ALU'nun çıkışındaki 1 bitlik zero biti, beq sinyali ve bneq sinyalinin değili and'lenerek branch edilip edilmeyeceğine karar verilir.Eğer branch edilecekse program countera instructionun immediate değerinin sign extend edilmiş hali yazlır.Diğer durumlarda program counter 1 artırılır.

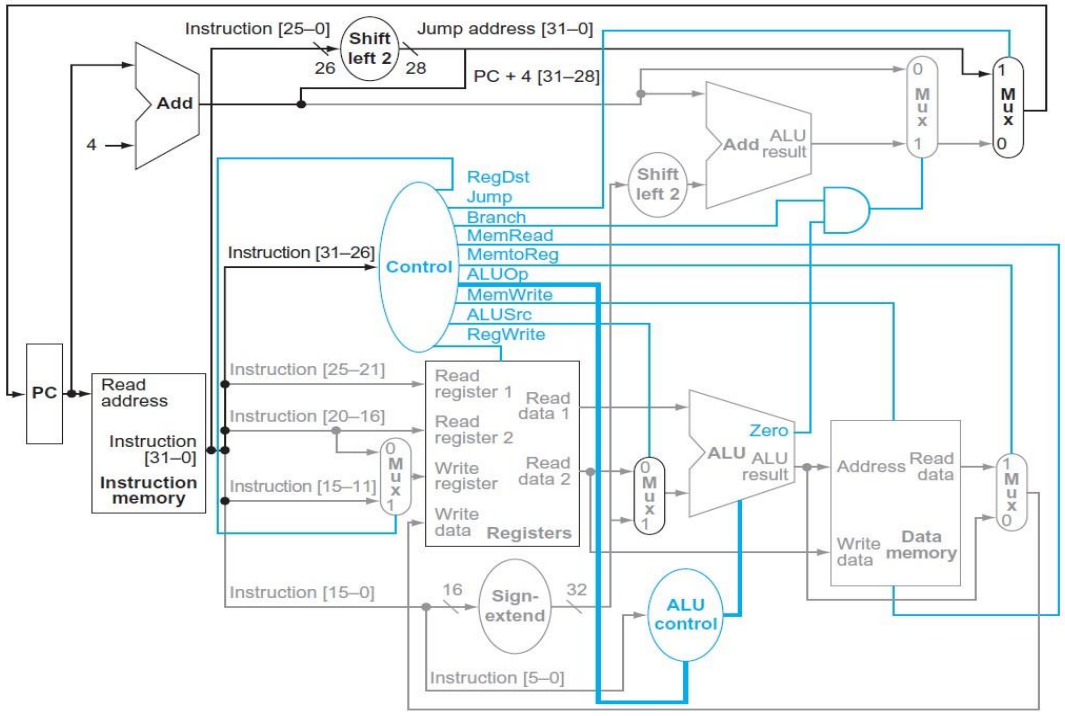
### 1.2.3 Life Cycle of J-Type Instruction




Bu instructionlar Jump ve jal instructionlarıdır.Jump instructionunda 26 bitlik target adres sign extender modülünde 32 bite extend edilir.Cycle sonunda program countera extend edilmiş data yazılır.Jal instructionunda da aynı şeyler yapılır.Jump instructionuna ek olarak jal instructionunda Register 31 ' e program counter + 2 yazılır.

## Important Informations

Bu projede shift left modülü bulunmamaktadır.Ayrıca proram counter 4'er 4'er artırılmak yerine 1'er 1'er artırlır.Bunun sebebi bizim memory olarak kullandığımız dosyalarda byte adressing yerine Word adressing olmasıdır.Aynı sebepten dolayı lw ve sw instructionlarında adress hesaplanırken address değerleri 2 shift right yapılmıştır.



# 2. METHOD

Bu proje behavioral verilog kullanılarak yapılmıştır.Toplamda 9 modül bulunmaktadır.

**control_unit modülü:** Bu mudül input olarak opcode ve function kodu alır.Bu inputlara göre instructionın doğru çalışması için gereken kontrol sinyalirini üretir.

**mips instr mem:** Bu modülde instruction.mem dosyasındaki instructionlar instr_mem iki boyutlu arrayine okunur.Bu modüle input olarak gelen program_counter değerinin gösterdiği yerdeki instruction değeri bu modülün çıkışına verilir.

**mux for destination register modülü:** Bu modül rd veya rs registerlarından hangisine yazılacağını seçmek için kullanıldı.RegDest sinyali bu modüle input olarak gelir.Bu sinyale göre rd veya rs sinyalleri çıkışa verilir.

**sign extender 16 to 32 modülü:** Bu modül instructionun 16 bitlik immediate kısmını 32 bite extend etmek için kullanıldı.

**sign extender 26 to 32 modülü:** Bu modül j type instructionlarda bulunan 26 bitlik adress değerini 32 bite extend etmek için kullanıldı.

**mips registers modülü:** Bu modülde registers.mem dosyasındaki 32 bitlik 32 adet data registers iki boyutlu arrayine okunur.Rs ve rt registerlarının içerikleri okunur.Cyle sonunda eğer signal_reg_write sinyali 1 ise registera yazma yapılır.

**mux_2_1_32bit:** Bu modül iki yerde kullanılır.ALU modülünün girişini seçmek için ALUSrc sinyali input olarak gelir.ALUSrc 1 ise sign extend edilmiş immediate değeri bu modülün çıkışına verilir.Eğer 0 ise rt değerinin içeriği bu modülün çıkışına verilir.

Bu modül bir de registera yazılacak datayı seçmek için kullanılır.Eğer MemToReg sinyali 1 ise bu modülün çıkışına memoryden okunan değer verilir.Eğer 0 ise ALUnun çıkışı bu modülün çıkışına verilir.

**ALU modülü:** Bu modül aritmetik işlemleri gerçekleştirmek için kullanılan modüldür.Bu modüle input olarak gelen ALUOp kontrolüne göre yapılacak işlem seçilir ve o işlemin sonucu bu modülün çıkışına verilir.Ayrıca bu modülde zero ve overflow biti de bu modülün outputuna verilir.Zero biti branch kararını vermek için kullanılır.

**mips data mem:** Bu modülde ilk önce data.mem dosyasından okunan 32bitlik datalar data_mem iki boyutlu arrayine okunur.Eğer sig_mem_read sinyali 1 ise mem_adress değerinin gösterdiği adresten okuma yapılır.Cycle sonunda eğer sig_mem_write sinyali 1 ise memorye yazma yapılır.

**mips core modülü:** Bu modül top-level modüldür. Bu modülde bütün modüller birbirine bağlanır.Cycle sonunda program countera uygun değer yazılır.

Registerların ve memorynin güncel değerlerini yazabilmek için res_regissters.mem ve res_data.mem dosyaları oluşturulur ve güncel değerler bu dosyalara yazılır.

## 3. RESULT

Bu projede top modülü test etmek için bir adet testbench bulunmaktadır.Bu testbenchde clock 0 ‘dan başatılır.Her 100 nanosaniyede clock değişitirilir.Clock’un her bir olduğu zaman cyle sonudur.

Bu projeyi test etmek için aşağıdaki assembly kodu kullanılmıştır.

sw $31, 256($15)

add $3, $2, $1

addu $5, $3, $4

and $7, $5, $6

or $9, $7, $8

nor $11, $10, $9

$13 = $12 << 4

$15 = $14 >> 2

sub $17, $31, $16

subu $18, $30, 17

slt  $20, $19, 29

sltu $21, $22, $21

addi $23, $21, 18

andi $25, $23, 27

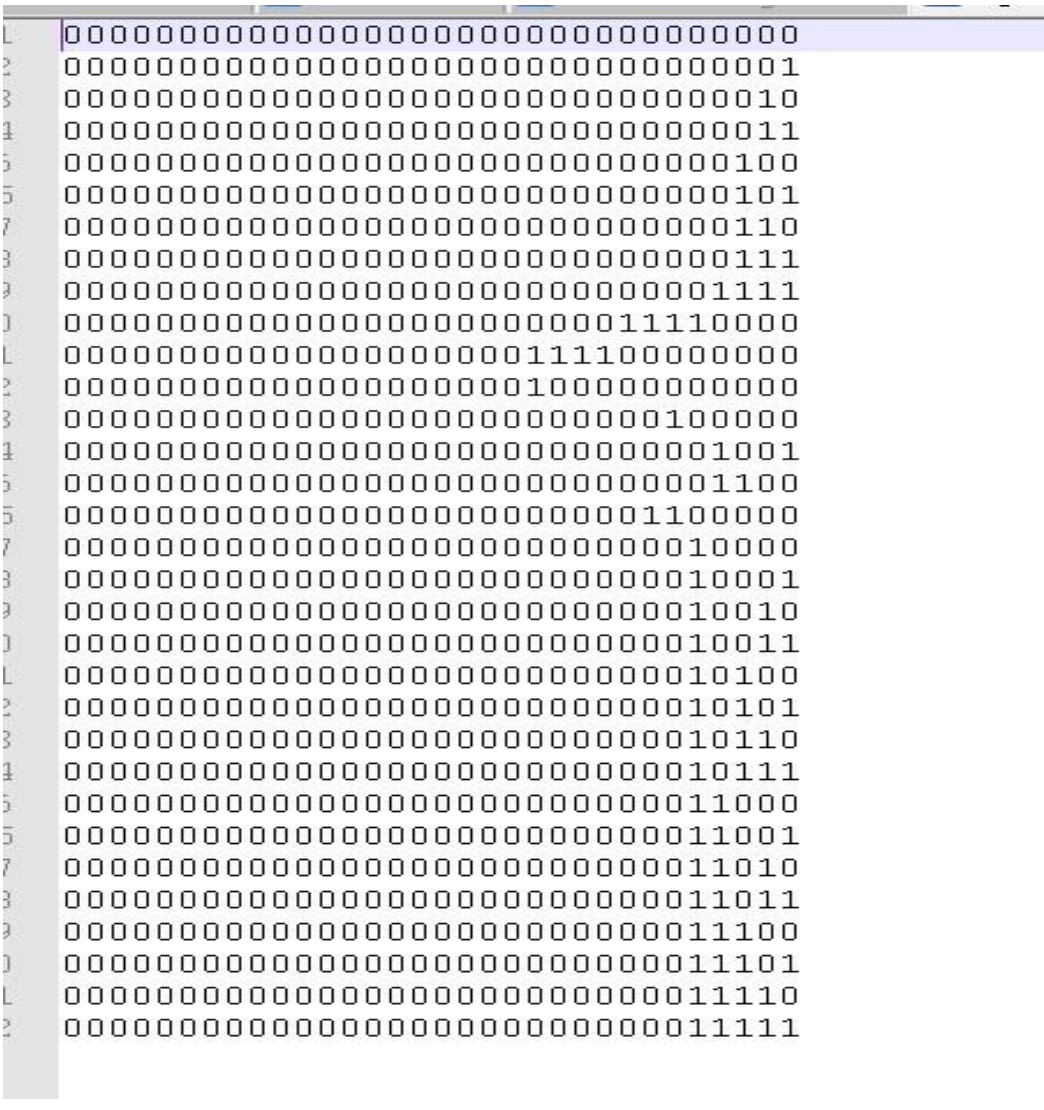
addiu $26, $25, 35

ori $27, $26, 25

slti $28, $29, 29

lw $31, 128($30)

ll $256($31)




Çalıştırmadan önce registerların içeriği.

lui $29, 255

beq $3, $7, 22

addi $3, $3, 1

j 19

bneq $23, $25, 25

addi $23, $23, 1

j 22

beq $3, $5, 28

addi $3, $3, 1

jr $2

bneq $30, $30, 31

addi $30, $30, 1

jal 28

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips_testbench/test/registers/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000110
00000000000000000000000000000100
00000000000000000000000000000111
00000000000000000000000000000110
00000000000000000000000000000110
00000000000000000000000000001111
00000000000000000000000000001111
00000000000000000000111100000000
11111111111111111111000011110000
00000000000000000000000000100000
00000000000000000000000111110000
00000000000000000000000000001100
00000000000000000000001111000000
00000000000000000000000000010000
11111111111111111111111111110001
11111111111111111111111111010011
00000000000000000000000000010011
00000000000000000000000000000001
00000000000000000000000000000001
00000000000000000000000000010110
00000000000000000000000000101001
00000000000000000000000000011000
00000000000000000000000000011001
00000000000000000000000000111101
00000000000000000000000000011011
00000000000000000000000000000001
00000000000000000000000000000000
00000000000000000000000000100000
00000000000000000000000000010111
```

Çalıştırdıktan sonra registerların içeriği.