UNIVERSITETI POLITEKNIK I TIRANËS
FAKULTETI I TEKNOLOGJISË SË INFORMACIONIT
Departamenti i Inxhinierisë Informatike

Relacion Laboratori

Lënda: Arkitekturë e kompjuterave

Punoi: Piro Gjikdhima **Pranoi:** MSc. Klark Ahmeti

Bachelor “ Inxhinieri Informatike”

Tiranë

Maj 2024

Punë laboratori 1

Ushtrimi 1:

Të ndërtohet një full adder me një bit. Te ndërtohet një full adder me 8 bit duke përdorur 8 full adder me 1 bit.

Ushtrimi 2:

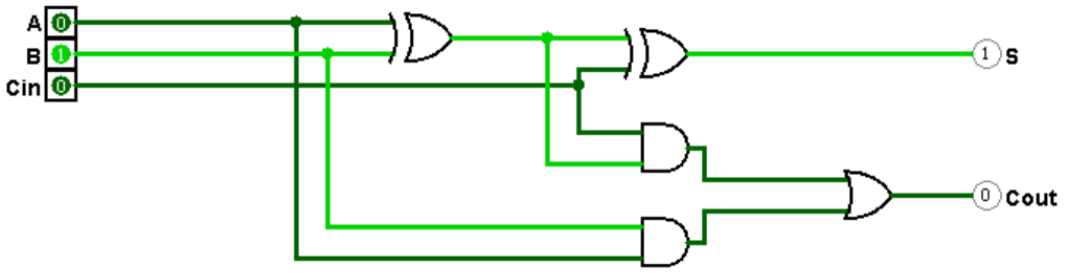
a) Të ndërtohet një multiplexer 8:1. Përdorni multiplexerin e ndërtuar për të implementuar qarkun që kryen funksionin: **Out = AB + BC + AC**

b) Ndërtoni një dekoder 2/4. Përdorni dekoderin për të implementuar funksionin e dhënë në pikën a.

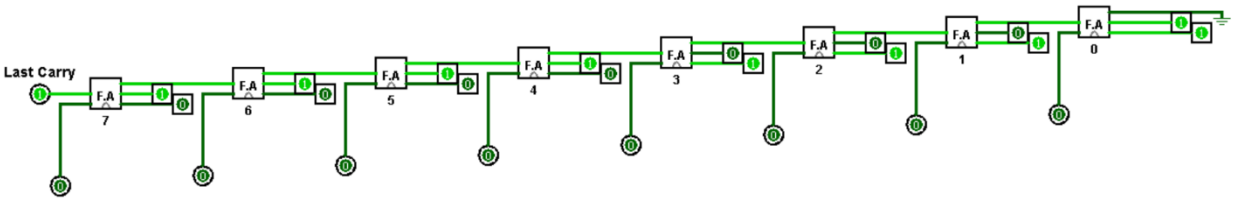
Për zgjidhjen e ushtrimeve është përdorur programi Logisim për ndërtimin e skemave.

Ushtrimi 1

Skema e Full-Adder me 1-bit



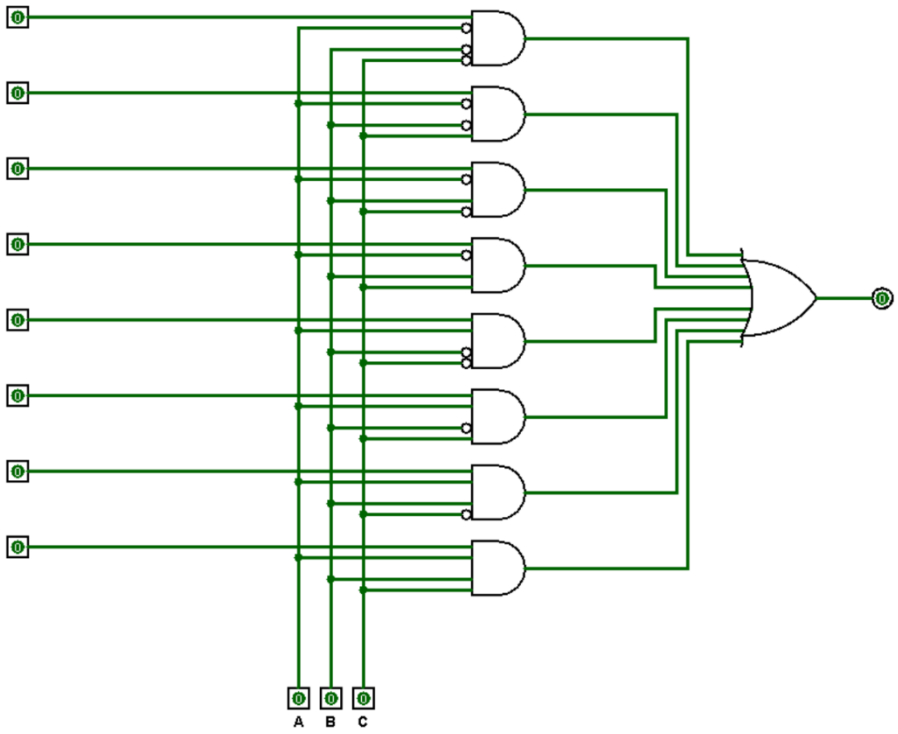
Skema e Full-Adder me 8-bit



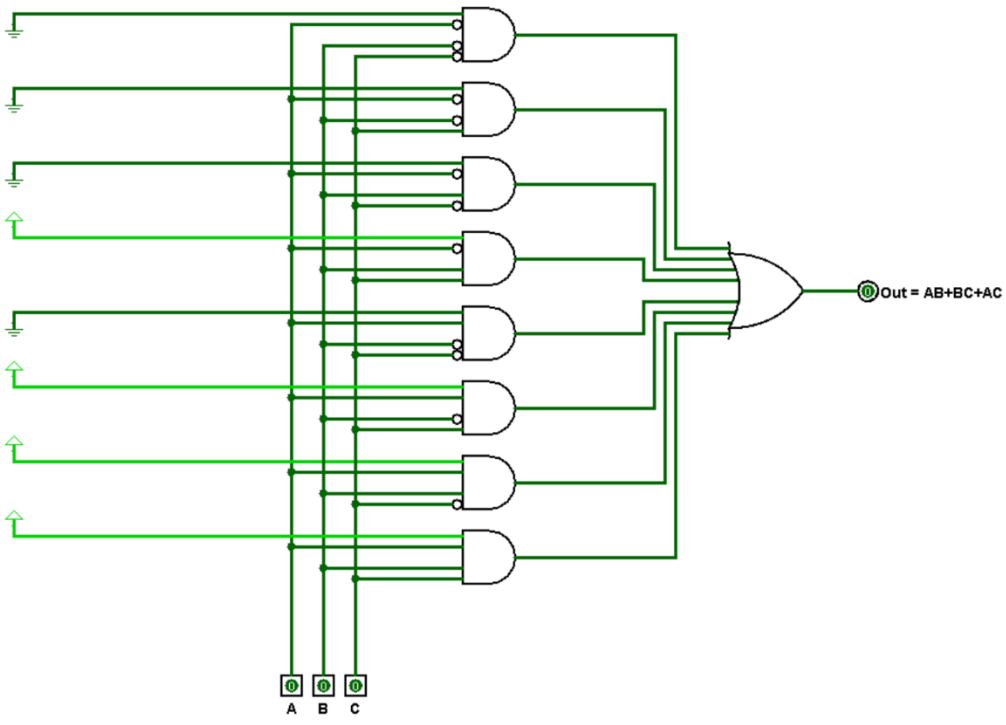
Skema për Full-Adder me 8-bit është ndërtuar duke përdorur qarkun e Full-Adder me 1-bit të ndërtuar më lart, duke ripërdorur skemën në Logisim.

Ushtrimi 2

Skema e Multiplekserit 8×1



Shprehja **Out = AB + BC + AC** me multiplexer



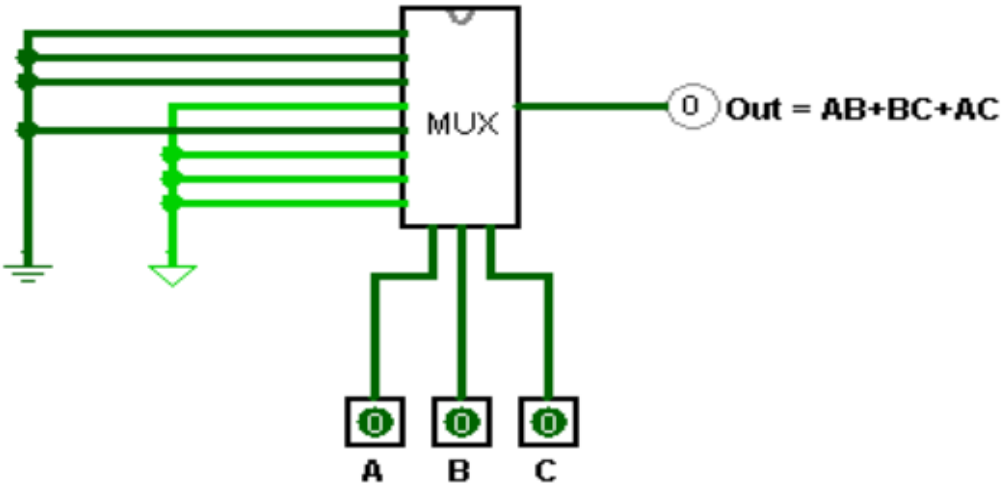
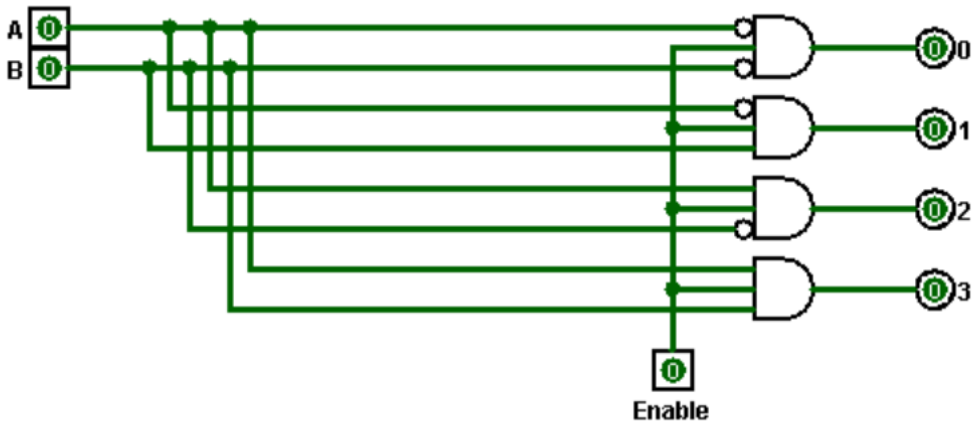


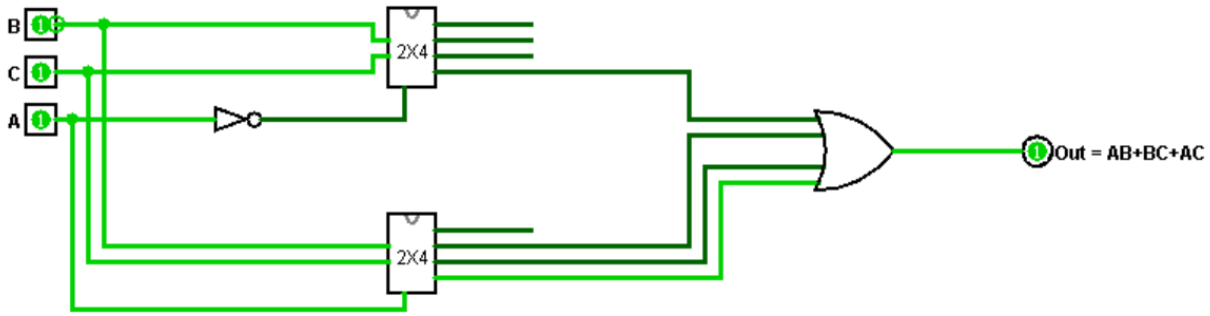
Tabela e Vërtetësisë

A	B	C	AB	BC	AC	OUT
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	1
1	0	0	0	0	0	0
1	0	1	0	0	1	1
1	1	0	1	0	0	1
1	1	1	1	1	1	1

Skema e Dekoderit 2×4



Shprehja **Out = AB + BC + AC** me dekoder



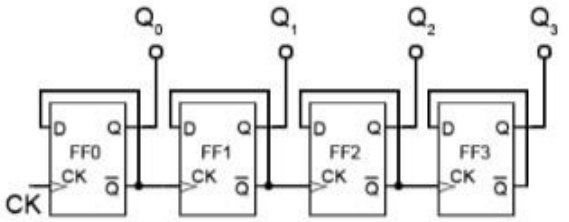
Punë Laboratori 2

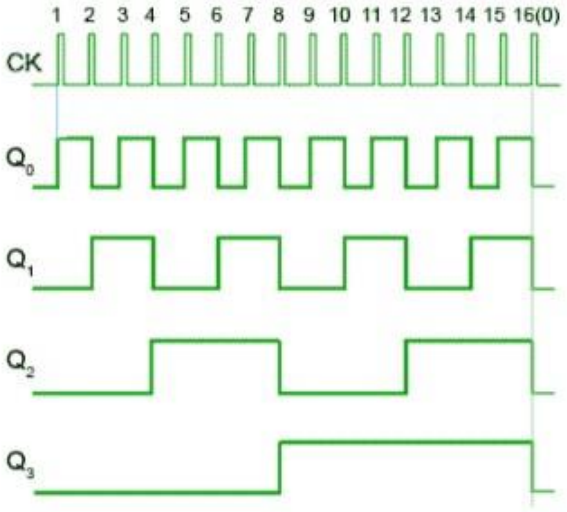
Ushtrimi 1:

Të ndërtohet një flip flop D (FF-D).

Ushtrimi 2:

Një numërues me 4 bit mund të ndërtohet duke përdorur FF-D si në figurë:





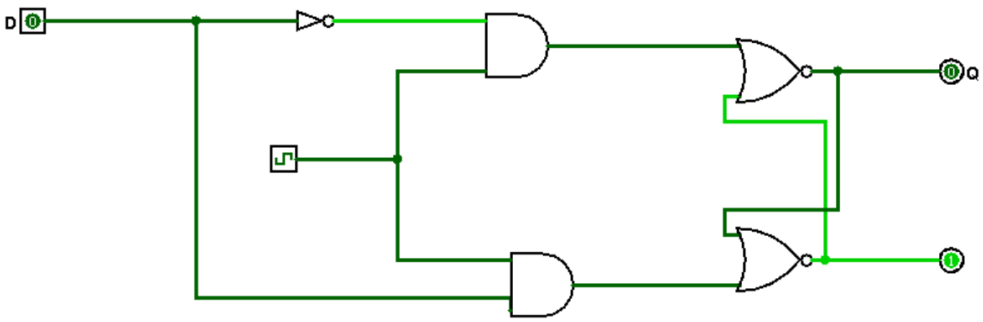
Ndërtoni një numërues me 8 bit duke përdorur FF-D.

Ushtrim 3:

Të ndërtohet një flip flop T (FF-T) duke përdorur flip flop D. Të ndërtohet numeruesi me 8 bit duke përdorur FF-T.

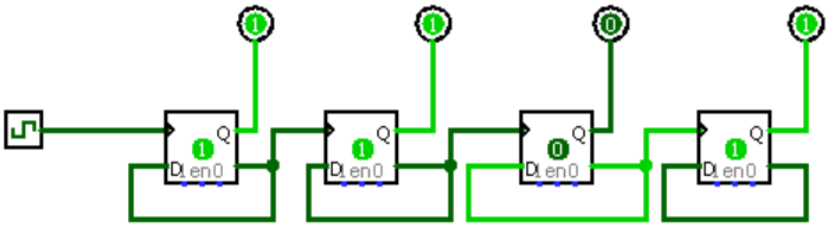
Ushtrim 1

Skema e FF – D



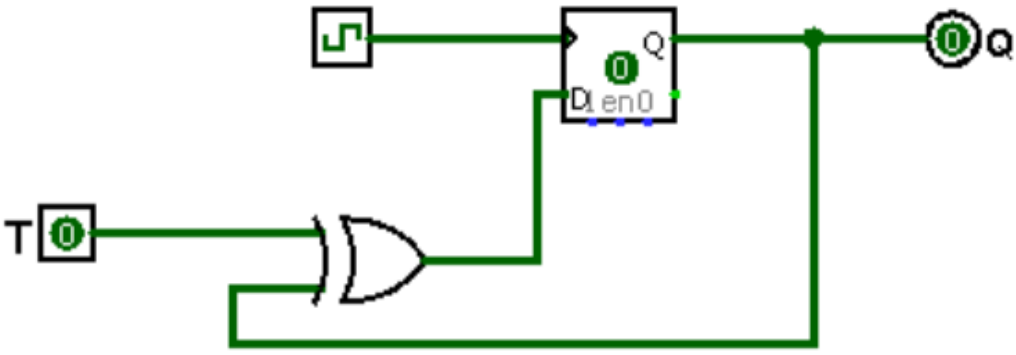
Ushtrimi 2

Skema e Numëruesit me 4-bit duke përdorur FF-D

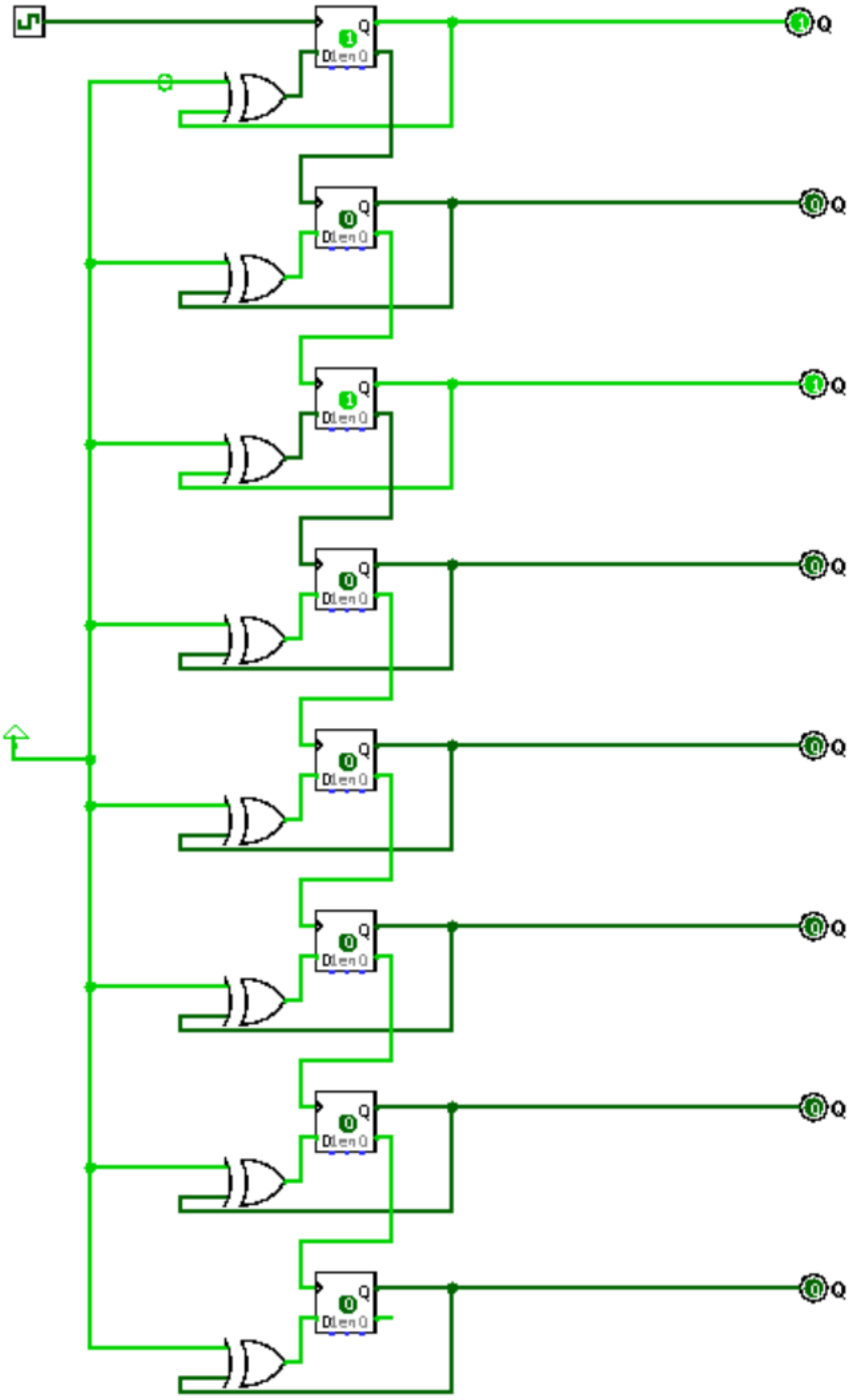


Ushtrimi 3

Skema e FF-T duke përdoru FF-D



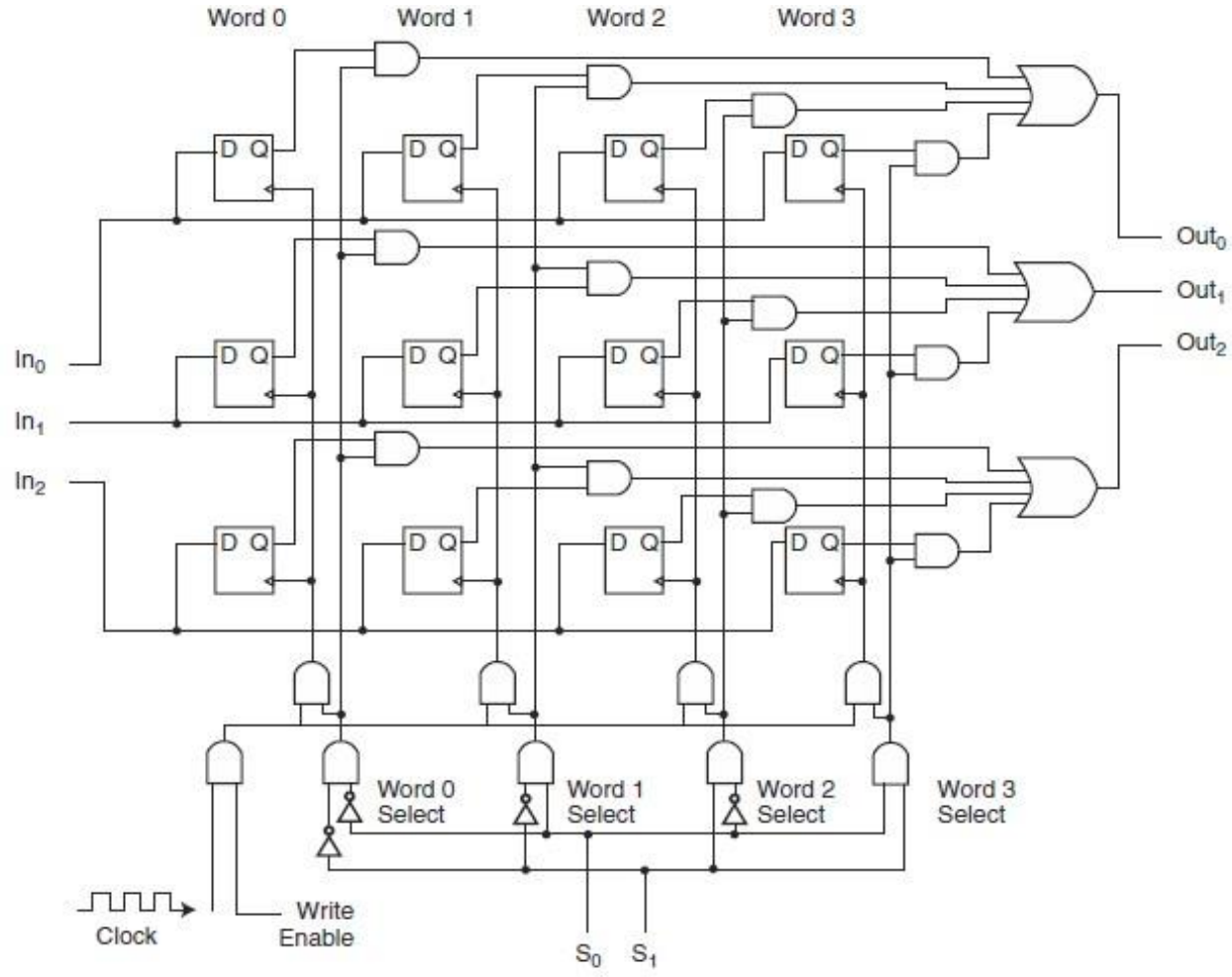
Skema e Numëruesit me 8-bit duke përdorur FF-T



Punë Laboratori 3

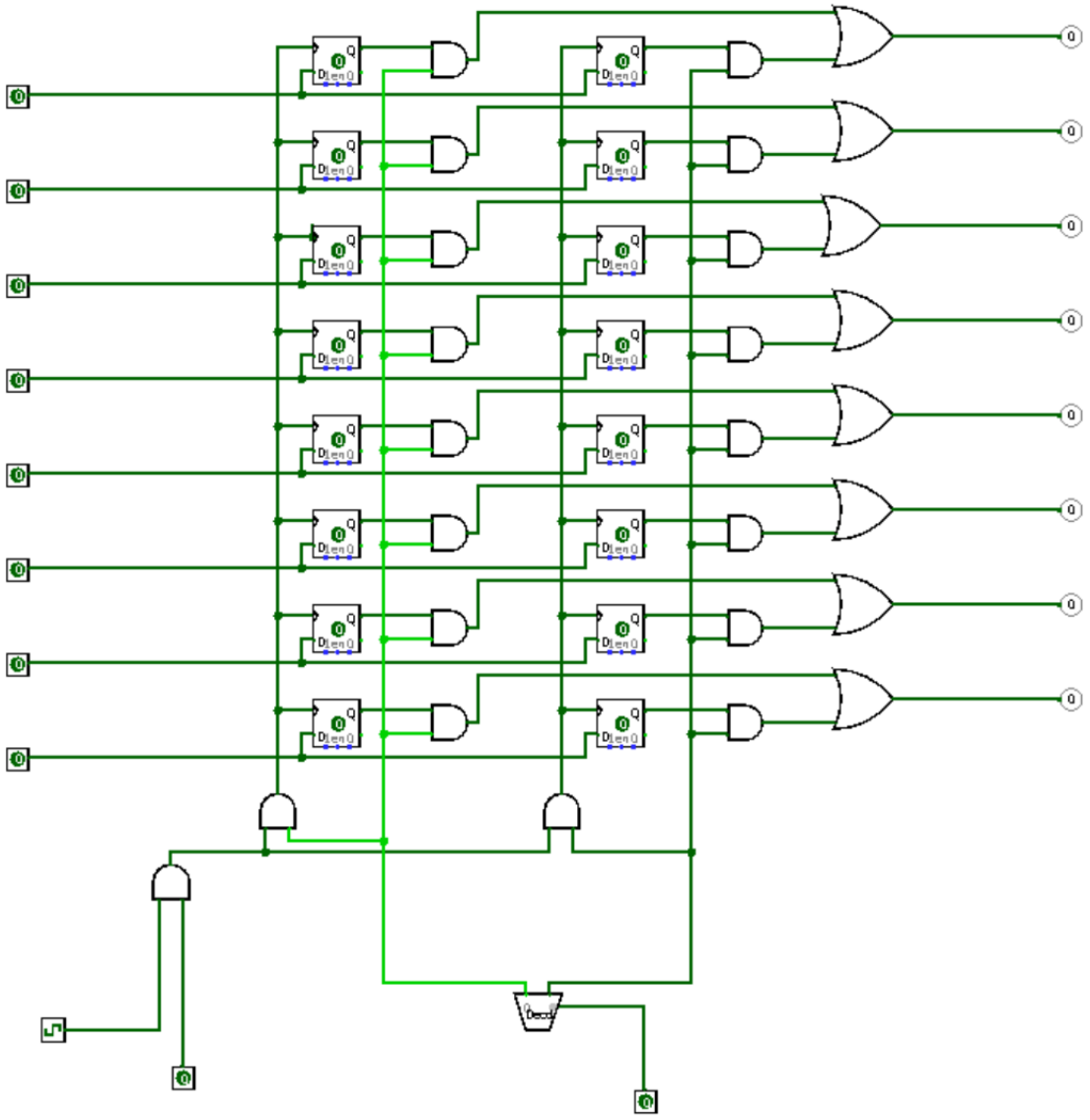
Ushtrim 1:

Skema ne figuren e meposhtme paraqet nje memorje 4x3. Ndertoni nje memorje 2x8. Perdorni dekoderine e gatshem te simulatorit per te selektuar fjalen.



Ushtrim 1

Skema e memories 2x8



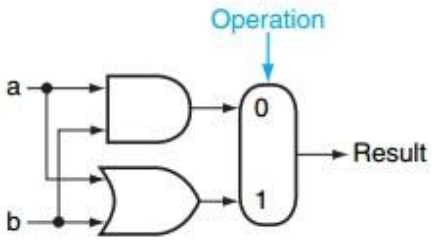
Memoria ndërtohet prej një dekoderi 1x2 sepse kemi për të përzgjedh midis 2 word me nga 8 bit , dhe përdorim gjithsej 16 FF-D , 8 për secilin word për të zgjedhur bitet e word-it.

Punë Laboratori 4

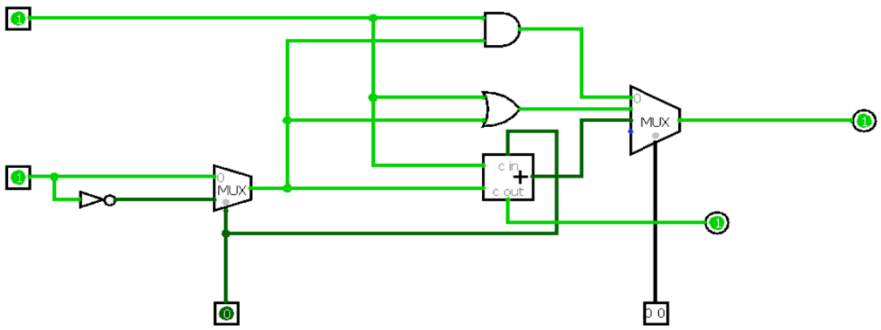
Ushtrim 1:

Duke përdorur simulatorin LogiSim, të ndërtohet një qark i cili kryen funksionin e mbledhjes, zbritjes, AND dhe OR mbi dy bit-e që merr si input-e. Disa bit-e të tjera në hyrje tregojnë cili funksion duhet te kryhet.

P.sh. qarku në figurën e mëposhtme, kryen funksionin AND edhe OR mbi bit-et a dhe b, në varësi te input-it *Opreation* (nëse ky bit është 0, kryhet funksioni AND, në të kundërt kryhet funksioni OR).

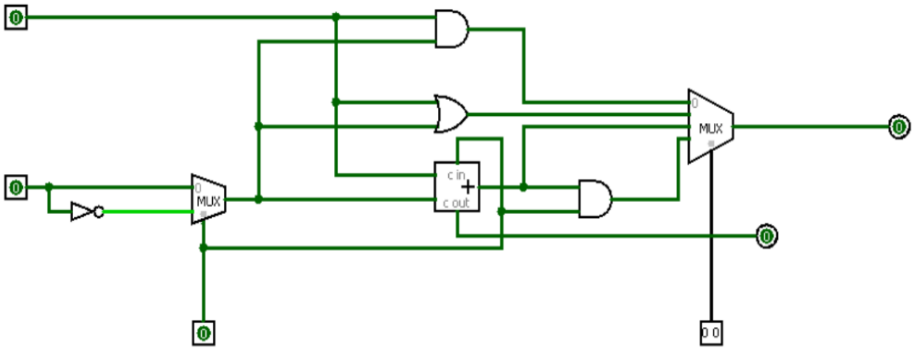


Ushtrimi 1



ALU-ja përbehet prej një porte AND , një porte OR , një Full-Adder-i, dy Multipexer-a (një 2x1 dhe një 4x1) dhe 4 sinjale (dy input-et, sinjali për C_{in}, dhe përzgjedhësi i MUX 4x1 për veprimin që kërkojmë të kryejmë) .

Tek skema e mëposhtme kam shtuar një portë AND me qëllim që të përdorim sinjalin e fundit të kontrollit për zbritjen, megjithëse ky sinjal do të përdorej për veprim tjetër.



Punë Laboratori 5

Ushtrimi 1:

Çfarë bën programi i mëposhtëm? Shpjegoni përgjigjen tuaj.

```
#sample2.asm
.data 0x10000100
msg: .asciiz “Hello”
     .text main:
li $v0, 4   la
$a0, msg   syscall
li $v0, 10
syscall
```

Klikoni në tab-in **Data** ne kornizën *Text Segment* për të shfaqur *Data Segment*. Ku dhe si ruhet stringa “Hello” në kujtesë? Shkruaj vlerat në ASCII të karaktereve përbërëse të stringës së lartpërmëndur.

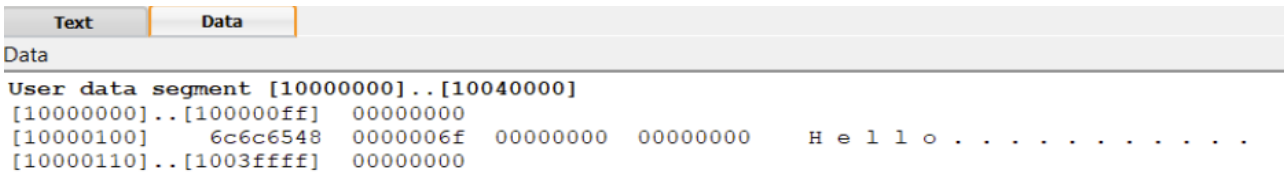
Ushtrimi 2:

Duke u bazuar tek skedari i ushtrimit 1, modifikoje atë për të ndryshuar stringën në ekran nga “Hello” në “HellO”. Ndryshimet e bëra duhet te pasqyrohen në kujtesën e kompjuterit. (Për këtë qëllim duhet të punohet mbi stringën fillestare duke përdorur tabelën e kodit ASCII)

Ushtrimi 1

Programi bën këto veprime:

1. Krijon një String "Hello" në segmentin e data.
2. Ngarkon procedurën print_string në $v0 dhe adresën e String-ës msg në $a0.
3. Bën një syscall për të printuar vargun "Hello" në ekran.
4. Ngarkon procedurën exit në $v0.
5. Bën një syscall për të përfunduar programin.
1. Me pak fjalë, ky program printon "Hello" në ekran dhe më pas përfundon.



‘H’ – 48_{hex}; ‘e’ – 65_{hex}; ‘l’ – $6c_{hex}$; ‘l’ – $6c_{hex}$; ‘o’ – $6f_{hex}$ Këto vlera duken dhe në data segment.

Ushtrimi 2

```
      .data 0x10000100
msg:  .asciiz "Hello"
      .text

main:
      li $v0, 4          # Kodi i thirrjes për print_string
      la $a0, msg        # Vendos adresen e stringes msg ne $a0

      lb $t2, 4($a0)     # Ngarkon nje bajt nga adresa ($a0 + 4) ne $t2
                         # ($a0 + 4) do te jete karakteri "o" ne stringen "Hello"

      addi $t2, $t2, -32 # Zbrit 32 nga vlera e $t2 dhe ruaj rezultatin ne $t2
                         # Kjo efektivisht e konverton karakterin "o" ne "O"

      sb $t2, 4($a0)     # Vendos karakterin e modifikuar ne adresen ($a0 + 4)
                         # Kjo do te ndryshoje "Hello" ne "HellO"
      syscall            # Thirrja për print_string

      li $v0, 10         # Kodi i thirrjes për dalje
      syscall            # Dalje nga programi
```

Punë Laboratori 6

Ushtrim i 1:

Jepet kodi në Assembly si më poshtë:

```
# messages.asm
.data
str: .asciiz "the answer = "
.text
main:
     li $v0, 4    # kodi i thirrjes së sistemit për print_string
     la $a0, str  # venia ne regjister e adreses se stringës
     syscall      # printimi i stringës
     li $v0, 1    # kodi i thirrjes së sistemit për
     print_int li $a0, 5      # numri i plotë për tu
     printuar syscall   # printimi i numrit të plotë li $v0,
     10 # kodi i thirrjes së sistemit për dalje syscall
     # mbarimi i programit
```

Bëni ndryshime për të krijuar një program i cili merr një numër nga përdoruesi, shfaq stringën e dhënë në kod dhe më pas printon numrin

Ushtrim i 2:

Jepet skedari arrayCount.asm :

```
# arrayCount.asm
.data
arrayA: .word 1, 0, 2, 0, 3
count: .word 999
     .text
main:

add $zero, $zero, $zero
……..
# kodi për të lexuar vlerën e ndryshores X
# kodi për të numëruar shumëfishat e X në vektorin arrayA
# kodi për të printuar rezultatin
# kodi për të mbaruar programin
```

a) Tregoni adresat e kujtesës ku ruhen ***arrayA*** dhe ***count.***
b) Zmadhoni vektorin duke vendosur 8 elemente nga 1 deri te 8.
c) Printoni numrin e elementëve të vektorit të cilët plotëpjestohen me një numër të dhënë nga përdoruesi, *X*, ku *X* është një numër i plotë fuqi e 2-shit.
d) Shtoni rreshta në kodin e pikës C në mënyrë që elementet e vektorit ti japë vetë përdoruesi.

Ushtrimi 1

```
# messages.asm
     .data
str: .asciiz "Vendosni një numër: "  # stringa për t'u shfaqur
     .text
main:
    li $v0, 4            # kodi i thirrjes për print_string
    la $a0, str          # adresimi i stringës për t'u printuar
    syscall              # thirrja për print_string

    li $v0, 5            # kodi i thirrjes për leximin e një numri të plotë
    syscall              # thirrja për leximin e një numri të plotë
    move $s0, $v0        # ruaj numrin e lexuar në $s0

    li $v0, 1            # kodi i thirrjes për print_int
    move $a0, $s0        # vendos numrin e lexuar në regjistrin për printim
    syscall              # thirrja për print_int

    li $v0, 10           # kodi i thirrjes për dalje
    syscall              # dalje nga programi
```

Ushtrimi 2

a)

Adresa e arrayA ruhet në $s0 i cili ka vlerën 10010040
Adresa e count ruhet në $s1 i cili ka vlerën 10010088
Vlerat janë në formatin Hexadecimal.

b) Madhësinë e vektorit e rrisim duke inicializuar 8 vlera në vend të 5.

c)

```
# arrayCount
        .data
arrayA: .word 1, 2, 3, 4, 5, 6, 7 ,8
prompt1:.asciiz "Numri për të cilin gjejmë shumëfishat në vektor (fuqia e dytë): "
prompt2:.asciiz "Rezultati: "
count:  .word 999
        .text

main:
   la $s0, arrayA      # Ngarko adresën bazë të vargut në $s0
   la $s1, count       # Ngarko adresën e variablës së numrit në $s1
   li $s3, 0           # Inicializo numrin për shumëfishat

   li $v0, 4           # Printo kërkesën për hyrjen e përdoruesit
   la $a0, prompt1
   syscall
   li $v0, 5           # Lexo numrin nga hyrja e përdoruesit
   syscall
   addi $t2, $v0, -1   # Ruaj numrin për të gjetur shumëfishat
   li $t1, 0           # Inicializo numrin për iteracionet

LOOP:
   slti $t0, $t1, 8     # Kontrollo nëse numri i iteracioneve është më pak se
   madhësia e vargut
```

```
   beq $t0, $zero, FUND # Nëse jo, del nga cikli

   lw $t3, 0($s0)       # Ngarko vlerën nga vargu
   beq $t3, $zero, SKIP # Nëse elementi i vargut është zero, shko tek iterimi
   tjetër
   and $t0, $t3, $t2    # Kontrollo nëse elementi i vargut është shumëfish i numrit
   të dhënë
   bne $t0, $zero, SKIP # Nëse nuk është shumëfish, shko tek iterimi tjetër
   addi $s3, $s3, 1     # Inkremento numrin për shumëfishat

SKIP:
   addi $s0, $s0, 4   # Shko te elementi tjetër në varg
   addi $t1, $t1, 1   # Inkremento numrin për përsëritje
   j LOOP             # Përsërit pjesën e shumëfishave

FUND:
   li $v0, 4             # Printo kërkesën për rezultatin
   la $a0, prompt2
   syscall
   sw $s3, 0($s1)        # Ruaj numrin e shumëfishave në memorie
   li $v0, 1             # Printo numrin e shumëfishave
   move $a0, $s3
   syscall
   li $v0, 10            # Dal nga programi
   syscall
```

d)

```
# arrayCount
         .data
arrayA:  .word 1, 2, 3, 4, 5, 6, 7, 8
prompt:  .asciiz "Vendos vleren ne vektor: "
prompt1: .asciiz "Numri për të cilin gjejmë shumëfishat në vektor (fuqia e dytë): "
prompt2: .asciiz "Rezultati: "
count:   .word 999
         .text

main:
    la $s0, arrayA      # Ngarko adresën bazë të vektorit në $s0
    la $s1, count       # Ngarko adresën e variablës së numrit në $s1
    li $s3, 0           # Inicializo një numër për shumëfishat

    # Përshkrimi për të futur vlerat në varg
    li $t1, 0           # Inicializo numrin për iteracionin
INSERT:
    slti $t0, $t1, 8    # Kontrollo nëse numri i iteracionit është më pak se
madhësia e vargut
    beq $t0, $zero, DIL # Nëse jo, dal nga përsëritja
    li $v0, 4           # Printo kërkesën për vendosjen e vlerave prej përdoruesit
    la $a0, prompt
    syscall
    li $v0, 5           # Lexo vleren nga  përdoruesi
    syscall
    sw $v0, 0($s0)      # Ruaj vleren në varg
    addi $s0, $s0, 4    # Shko te elementi tjetër në varg
    addi $t1, $t1, 1    # Inkremento numrin për iteracion
    j INSERT            # Përsërit
```

```
# Pas futjes së vlerave, kërko numrin për të gjetur shumëfishat
DIL:
    li $v0, 4             # Printo kërkesën për hyrjen e përdoruesit
    la $a0, prompt1
    syscall
    li $v0, 5             # Lexo numrin nga hyrja e përdoruesit
    syscall
    addi $t2, $v0, -1     # Ruaj numrin për të gjetur shumëfishat

    # Përshkrimi për të numëruar shumëfishat e numrit të dhënë në varg
    li $t1, 0             # Rifillo numrin për iteracionet
LOOP:
    slti $t0, $t1, 8      # Kontrollo nëse numri i përsëritjeve është më pak se
madhësia e vargut
    beq $t0, $zero, FUND  # Nëse jo, dal nga përsëritja

    lw $t3, 0($s0)        # Ngarko vlerën nga vargu
    beq $t3, $zero, SKIP  # Nëse elementi i vargut është zero, shko tek iterimi
tjetër
    and $t0, $t3, $t2     # Kontrollo nëse elementi i vargut është shumëfish i
numrit të dhënë
    bne $t0, $zero, SKIP  # Nëse nuk është shumëfish, shko tek iterimi tjetër
    addi $s3, $s3, 1      # Inkremento numrin për shumëfishat

SKIP:
    addi $s0, $s0, 4      # Shko te elementi tjetër në varg
    addi $t1, $t1, 1      # Inkremento numrin për përsëritje
    j LOOP                # Përsërit

# Printo numrin e shumëfishave
FUND:
    li $v0, 4             # Printo kërkesën për rezultatin
    la $a0, prompt2
    syscall
    sw $s3, 0($s1)        # Ruaj numrin e shumëfishave në memorie
    li $v0, 1             # Printo numrin e shumëfishave
    move $a0, $s3
    syscall
    li $v0, 10            # Dal nga programi
    syscall
```

Punë Laboratori 7

Ushtrim 1:

Çfarë bën programi i mëposhtëm? Shpjegoni përgjigjen tuaj.

```
 .data
array: .word 6, 2, -33, 10, 4, 12, 4
length: .word 7 .text
.globl main
 main:
     la $a0, array lw
     $a1, length jal
     max add $a0, $v0,
     $zero li $v0, 1
     syscall li $v0, 10
     syscall
 max:
     add $t0, $zero, $zero
     add $v0, $zero, $zero
     add $v1, $zero, $zero
cikel:
     sltu $t2, $t0, $a1
     beq $t2, $zero,
     fund lw $t1, 0($a0)
     sltu $t2, $t1, $v0
     bne $t2, $zero,
     kalo add $v0, $t1,
     $zero add $v1, $t0,
     $zero
kalo:
     addi $t0, $t0, 1
     addi $a0, $a0, 4
     j cikel
fund:
      jr $ra
```

Për secilin instruksion vendosni komentin korrespondues.

Ku vendoset në memorje e dhëna **`array`**? Po e dhëna **`length`**? Shpjegoni përgjigjen tuaj.

Tregoni përmbajtjen e regjistrave në fund të ekzekutimit të programit. Shpjegoni përmbajtjen e secilit prej tyre.

Ushtrim 2:

Ndërtoni një program që shfaq në ekran elementët e një vektori (me 8 elementë) që janë shumfisha të numrit pesë. (Vektori të jetë variabël global). Programi të përmbajë proçedurën **`main`** edhe një proçedurë tjetër. Numri i elementëve që do të kontrollohen të merret si input nga përdoruesi.

Të supozohet që inputi i dhënë nga përdoruesi nuk tejkalon përmasën e vektorit.

(p.sh. nëse vektori ka elementët: 7 15 14 25 5 115 16 16, edhe përdoruesi jep si input vlerën 4, atëherë në ekran do të shfaqen vetëm numrat 15, 25).

Ushtrimi 1

```
.data
array: .word 6, 2, -33, 10, 4, 12, 4
length: .word 7
.text
.globl main

# Funksioni max gjen vlerën maksimale dhe pozicionin e saj në vektor

main:
    # Vendos adresën e fillimit të vektorit në $a0
    la $a0, array
    # Ngarko gjatësinë e vektorit në $a1
    lw $a1, length
    # Thirr funksionin max për të gjetur vlerën maksimale dhe pozicionin e saj
    jal max

    # Shfaq vlerën maksimale në konsolë
    li $v0, 1
    syscall

    # Dalje nga programi
    li $v0, 10
    syscall

max:
    # Inicializo variablat $v0 dhe $v1 me vlerat fillestare
    add $v0, $zero, $zero   # $v0 = 0 (vlera maksimale fillestare)
    add $v1, $zero, $zero   # $v1 = 0 (pozicioni fillestar i vlerës maksimale)

    # Loop për të gjetur vlerën maksimale dhe pozicionin e saj
    # i: iteratori për vektorin
cikel:
    sltu $t2, $t0, $a1      # Kontrollon nëse iteratori është më i vogël se gjatësia
e vektorit
    beq $t2, $zero, fund    # Nëse nuk, shko në fund të funksionit

    # Ngarko vlerën në indeksin aktual të vektorit
    lw $t1, 0($a0)

    # Kontrollo nëse vlera aktuale është më e vogël se vlera maksimale
    sltu $t2, $t1, $v0
    bne $t2, $zero, kalo   # Nëse po, kalo në iteracionin tjetër

    # Nëse vlera aktuale është më e madhe, ruaj vlerën dhe pozicionin e saj
    add $v0, $t1, $zero   # Vlera maksimale aktualizohet
    add $v1, $t0, $zero   # Pozicioni i vlerës maksimale aktualizohet

kalo:
    addi $t0, $t0, 1      # Rrit iteratorin për të kaluar në elementin tjetër
    addi $a0, $a0, 4      # Leviz në adresën e elementit tjetër të vektorit
    j cikel                # Kthehu në fillim të ciklit

fund:
    jr $ra                 # Kthehu në thirrësin e shkëputur
```

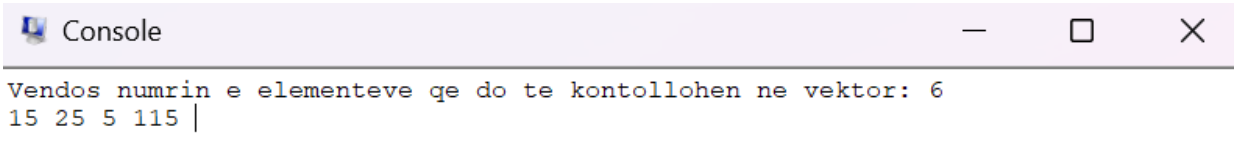
Vlera maksimale në fund do të jetë më e madhja në vektor në vlerë absolute.

$a0: adresa e fillimit të vektorit
$a1: gjatësia e vektorit
$v0: vlera maksimale
$v1: pozicioni i vlerës maksimale në vektor

```
PC        = 400094
EPC       = 0
Cause     = 0
BadVAddr  = 0
Status    = 3000ff10

HI        = 0
LO        = 0

R0  [r0] = 0
R1  [at] = 10010000
R2  [v0] = a
R3  [v1] = 0
R4  [a0] = 1001005c
R5  [a1] = 5
R6  [a2] = 7fffee64
R7  [a3] = 0
R8  [t0] = 0
R9  [t1] = 6
R10 [t2] = 73
R11 [t3] = 5
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 10010018
R17 [s1] = 6
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
R28 [gp] = 10008000
R29 [sp] = 7fffee58
R30 [s8] = 0
R31 [ra] = 400064
```

Ushtrimi 2

```
# arrayCount
        .data
arrayA: .word : 7, 15, 14, 25,  5,  115, 16, 16
prompt: .asciiz "Vendos numrin e elementëve që do të kontollohen në vektor: "
space: .asciiz " "
        .text

main:

    la $s0, arrayA
    li $v0, 4
    la $a0, prompt
    syscall
    li $v0, 5
    syscall
    move $s1, $v0
    li $t1, 0           # Inicializo numrin për iteracionet
    li $t3, 5           # Inicializo numrin për shumefishat

LOOP:
    slt $t0, $t1, $s1
    beq $t0, $zero, FUND
    lw $t2, 0($s0)

    move $a0,$t2
    move $a1,$t3
    jal modulus

    bne $v0, $zero, SKIP

    move $a0,$t2
    li $v0, 1
    syscall
    la $a0, space
    li $v0, 4
    syscall

SKIP:
    addi $s0, $s0, 4
    addi $t1, $t1, 1
    j LOOP
FUND:
    li $v0, 10
    syscall
modulus:
mod_loop:
    sub $a0, $a0, $a1
    bgez $a0, mod_loop
    add $a0,$a0,$a1
    move $v0, $a0
    jr $ra
```



Punë Laboratori 8

Ushtrim 1:

Ndërtoni një program në assembler që ekzekuton kodin e dhënë më poshtë në gjuhën C. (Vektori të jetë variabël global).

```
for (k = 0; k < count; k++) {
       for (index = 0; index < length; index ++) {
if(index < 15)
          array[index] = 0;
else
          array[index] = array[index] + 1;
      }
    }
```

Programi fillon si më poshtë (madhësia e matricës dhe vlera e count do të varet nga skenarët që paraqiten në vijim).

```
.data
array: .word 0 : 256 #matrica array ka vend per 256 fjale
length: .word 256 count: .word 4 .text .globl main
```

Si fillim vendosni me anë të një cikli gjithë elementët e matricës në vlerën 10. Më pas shkruani kodin për ciklin e dhënë më sipër.

Përdorni simulatorin Mars. Shpjegimin për përdorimin e tij e gjeni tek dokumenti ***Mars Tutorial.pdf*** (tek faqja e laboratoreve)

Ushtrimi 1.1

```
.data
array: .word 0: 256      #matrica array ka vend per 256 fjale
length: .word 256
count: .word 4

.text
.globl main

main:
          # Inicializo të gjithë elementët e array në vlerën 10
          la $t0, array        # Ngarko adresën e fillimit të array në $t0
          li $t1, 10           # Vendos vlerën për inicializimin e vektorit (10) në
      $t1
          li $t2, 256          # Ngarko gjatësinë e vektorit (256 elementë) në $t2

loop_init:
          sw $t1, 0($t0)       # Ruaj vlerën 10 në pozicionin aktual të treguar nga
      $t0
          addi $t0, $t0, 4     # Kalo te pozicioni tjetër në vektor (rrit adresën
      me 4 byte)
          addi $t2, $t2, -1    # Zvogëlo numrin e iteracioneve
          bnez $t2, loop_init  # Nëse $t2 nuk është zero, vazhdo ciklin

          # Fundi i programit
          li $v0, 10           # Ngarko kodin e daljes për syscall për të
      përfunduar programin
          syscall              # Kryej thirrjen e sistemit për të dalë
```

Ushtrimi 1.2

```
.data
vektor: .word 0: 256      # Matrica vektor ka vend për 256 fjale
length: .word 256
count: .word 4

.text
.globl main

main:
    # Inicializo të gjithë elementët e vektorit në vlerën 10
    la $t0, vektor        # Ngarko adresën e fillimit të vektorit në $t0
    li $t1, 10            # Vendos vlerën për inicializimin e vektorit (10) në $t1
    li $t2, 256           # Ngarko gjatësinë e vektorit (256 elementë) në $t2

loop_init:
    sw $t1, 0($t0)        # Ruaj vlerën 10 në pozicionin aktual të treguar nga $t0
```

```
    addi $t0, $t0, 4      # Kalo te pozicioni tjetër në vektor (rrit adresën me 4
byte)
    addi $t2, $t2, -1     # Zvogëlo numrin e iteracioneve
    bnez $t2, loop_init   # Nëse $t2 nuk është zero, vazhdo ciklin

    # Marrim vlerat e length dhe count
    lw $t3, length        # $t3 = length
    lw $t4, count         # $t4 = count
    # Cikli i jashtëm për k
    li $t5, 0             # $t5 = k

outer_loop:
    beq $t5, $t4, end_program  # Nëse k == count, përfundo programin
    # Cikli i brendshëm për index
    li $t6, 0             # $t6 = index
    la $t0, vektor        # $t0 = adresa bazë e vektorit

inner_loop:
    beq $t6, $t3, outer_loop_end  # Nëse index == length, përfundo ciklin e
brendshëm
    # Kontrolli nëse index < 15
    li $t7, 15            # $t7 = 15
    bge $t6, $t7, else_part  # Nëse index >= 15, shko te else_part
    # Pjesa if: array[index] = 0
    sw $zero, 0($t0)      # Vendos 0 në array[index]
    j skip_else

else_part:
    # Pjesa else: array[index] = array[index] + 1
    lw $t8, 0($t0)        # Lexo array[index] në $t8
    addi $t8, $t8, 1      # $t8 = array[index] + 1
    sw $t8, 0($t0)        # Ruaj vlerën e re në vektor[index]

skip_else:
    addi $t6, $t6, 1      # index++
    addi $t0, $t0, 4      # Lëviz te elementi tjetër në vektor
    j inner_loop

outer_loop_end:
    addi $t5, $t5, 1      # k++
    j outer_loop

end_program:
    # Përfundimi i programit
    li $v0, 10           # Ngarko kodin e daljes për syscall për të përfunduar
programin
    syscall               # Kryej thirrjen e sistemit për të dalë
```

Skenari 1:

Cache Parameters:

- **Placement Policy:** Direct Mapping
- **Block Replacement Policy:** LRU
- **Set size (blocks):** 1 (Mund të jetë më e madhe se 1? Pse?)
- **Number of blocks:** 4
- **Cache block size (words):** 2

Program Parameters:

- **Array Size:** 128 words
- **Count:** 4

Pyetje:

1. Shpjegoni hit rate që përftohet.
2. Po nëse rritet count, çfarë do të ndodhte me hit rate? Pse?
3. Si mund të modifikohen parametrat e programit që të maksimizohet hit rate? Shpjegoni përgjigjen tuaj.
4. Si mund të modifikohen parametrat e cache-së që të maksimizohet hit rate? Shpjegoni përgjigjen tuaj.

Pyetja 1

Direct Mapped Cache ka çdo bllok të cache të caktuar në një vend specifik në kujtesë. Në këtë rast, ne kemi një cache me 4 blloqe dhe madhësia e bllokut është 2 fjalë.

Array ka 128 fjalë që do të thotë 64 blloqe (128 / 2 = 64). Meqenëse cache ka vetëm 4 blloqe, çdo bllok në kujtesë do të caktohet në një nga këto 4 blloqe të cache, duke përdorur index % 4.

Nëse count = 4, Programi do të kalojë aksesojë elementët e array 4 herë. Meqenëse array ka 64 blloqe dhe cache ka vetëm 4 blloqe, çdo bllok i ri që kërkohet ne cache do të rezultojë në një miss pas herës së parë që ky bllok ka qenë në cache.Meqë aksesimi është sekuencial ne kod dhe në cache blloqet që kanë ardhur do të zëvendësohen dhe do të vijnë në cache përsëri në iteracionin pasardhës.

Pra, meqë politika e vendosjes është Direct Mapping dhe sepse aksesimi është sekuencial do të kemi miss për çdo bllok që kërkohet në cache.Pra fjala e parë në bllok është miss, kurse e dyta është hit.

Pra Total Access: 128 * 4 = 512 access dhe Hit Rate = Miss Rate = (64×4)/(128×4) = 0.5 = 50 %

Pyetja 2

Nëse count rritet, meqë politika është Direct Mapping , nuk do të kemi ndryshim të hit rate apo miss rate. Do të rriten herët që ndodh aksesimi i blloqeve, por nuk dryshon fakti që word-i i parë do të jetë gjithmonë miss dhe i dyti hit.

Pyetja 3

- Madhësia e Array: Duke zvogëluar madhësinë e array, do të ketë më pak blloqe për të akesesuar, dhe mundësia për hit në cache do të rritet.Psh nëse marrim Array size = 8. Herën e parë do të kemi miss dhe hit pra, 0.5 hit dhe miss.Herët e tjera blloqet kanë ardhur, pra do të kemi vetëm hit. Që të kemi përmirësim duhet që të paktën Array Size të jetë më e vogël se 16(8 blloqe).

Pyetja 4

- Rritja e Numrit të Blloqeve: Pra rritja e madhësisë së cache.Duke rritur numrin e blloqeve në cache, më shumë blloqe të array mund të mbahen në cache njëkohësisht, duke reduktuar miss rate.
- Madhësia e Bllokut: Rritja e madhësisë së bllokut nga 2 fjalë në më shumë do të thotë që çdo bllok i cache do të përmbajë më shumë fjalë të array. Kjo do të reduktojë numrin e aksesimeve në kujtesë kryesore dhe do të rrisë hit rate.