

# Tahmini Ders İeriđi

## (Tentative Course Schedule – Syllabus)



- 1. Hafta:** Sayı sistemleri, onluk/ikilik taban sayı gösterimleri, mantıksal kapılar, computer system overview, başarıml (performance)
- 2. Hafta:** 2'lik tabanda işaretli sayılar, mikroişlemci tarihi, benchmarking, başarıml,
- 3. Hafta:** Başarıml, Amdahl yasası, RISC-V development Environment, Verilog HDL ile Birleşik (Combinational) devreler
- 4. Hafta:**, Verilog HDL ile sıralı (sequential) mantıksal devre ve sonlu durum makinası tasarımı, timing analysis
- 5. Hafta:** Aritmetik devre tasarımları: Toplama, çıkarma, arpma, bölme
- 6. Hafta:** Fixed ve Floating-Point sayı gösterimleri
- 7. Hafta:** RISC-V buyruk kümesi mimarisi (ISA) ve buyrukların tanıtımı
- 8. Hafta:** RISC-V buyruk kümesi mimarisi (ISA) ve buyrukların tanıtımı
- 9. Hafta:** Tek-evrim işlemci tasarımı (single-cycle CPU)
- 10. Hafta:** ok-evrim işlemci tasarımı (multi-cycle CPU)
- 11. Hafta:** Boruhatlı işlemci tasarımı (pipelined CPU)
- 12. Hafta:** Bellek sistemi ve hiyerarşisi
- 13. Hafta:** İleri mimari konuları: Branch prediction, superscalar cpu, out-of-order execution, multi-core systems
- 14. Hafta:** Gömülü sistemler, mikrodenetleyiciler, SoCs

# BİLGİSAYAR BAŞARIMI (PERFORMANCE)

**SORU:** P1, P2 ve P3 adında 3 adet işlemci aynı ISA'yi çalıştırmaktadır.

P1 işlemcisi 3.0 GHz frekans ve 1.5 CPI değerine sahiptir.

P2 işlemcisi 2.5 GHz frekans ve 1.0 CPI değerine sahiptir.

P3 işlemcisi 4.0 GHz frekans ve 2.2 CPI değerine sahiptir.

- Bir saniyede işlenen buyruk sayısı baz alındığında işlemcilerin performansını kıyaslayın.
- Eğer 3 işlemci de bir program 10 saniyede yürüttüyse, 3 işlemci için de saat çevrim ve buyruk sayılarını hesaplayınız.
- Yürütme zamanını %30 azaltmak istiyoruz fakat tasarımdaki bu değişiklik CPI'da %20 artışa neden oluyor. Bu hızlanmayı sağlamak için saat frekansı ne oranda artmalıdır?

**SORU:** P1, P2 ve P3 adında 3 adet işlemci aynı ISA'yi çalıştırmaktadır.

P1 işlemcisi 3.0 GHz frekans ve 1.5 CPI değerine sahiptir.

P2 işlemcisi 2.5 GHz frekans ve 1.0 CPI değerine sahiptir.

P3 işlemcisi 4.0 GHz frekans ve 2.2 CPI değerine sahiptir.

a) Bir saniyede işlenen buyruk sayısı baz alındığında işlemcilerin performansını kıyaslayın.

$$\frac{\text{Buyruk}}{\text{Saniye}} = \left( \frac{\text{Buyruk}}{\text{Çevrim}} \right) * \left( \frac{\text{Çevrim}}{\text{Saniye}} \right) = \frac{1}{CPI} * Freq$$

$$P1 \rightarrow (2/3) * 3 * 10^9 = 2 \text{ Milyar}$$

$$P2 \rightarrow 1 * 2.5 * 10^9 = 2.5 \text{ Milyar}$$

$$P3 \rightarrow 1/2.2 * 4 * 10^9 = 1.81 \text{ Milyar}$$

$$P2 > P1 > P3$$

1/CPI → IPC (Instruction per cycle): Eski nesil işlemcilerde teorik olarak bir çevrimde en fazla 1 buyruk tamamlanabildiği için CPI pratikte her zaman 1'den büyük olmak zorundaydı. Yeni nesil superscalar out-of-order işlemcilerde aynı anda birden çok buyruk işlemcide işlenebildiği için toplamda bir çevrimde birden fazla buyruk tamamlanabiliyor. Bu yüzden CPI yerine IPC parametresi kullanılabiliyor.

**SORU:** P1, P2 ve P3 adında 3 adet işlemci aynı ISA'yi çalıştırmaktadır.

P1 işlemcisi 3.0 GHz frekans ve 1.5 CPI değerine sahiptir.

P2 işlemcisi 2.5 GHz frekans ve 1.0 CPI değerine sahiptir.

P3 işlemcisi 4.0 GHz frekans ve 2.2 CPI değerine sahiptir.

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

$$\text{CPU clock cycles} = \text{Instructions for a program} \times \frac{\text{Average clock cycles}}{\text{per instruction}}$$

b) Eğer 3 işlemci de bir program 10 saniyede yürüttüyse, 3 işlemci için de saat çevrim ve buyruk sayılarını hesaplayınız.

$$P1 \rightarrow 10 = \# \text{Buyruk}(P1) * 1.5 * (1 / (3 * 10^9)) \rightarrow \# \text{Buyruk}(P1) = 20.00 \text{ Milyar}$$

$$P2 \rightarrow 10 = \# \text{Buyruk}(P2) * 1.0 * (1 / (2.5 * 10^9)) \rightarrow \# \text{Buyruk}(P2) = 25.00 \text{ Milyar}$$

$$P3 \rightarrow 10 = \# \text{Buyruk}(P3) * 2.2 * (1 / (4 * 10^9)) \rightarrow \# \text{Buyruk}(P3) = 18.18 \text{ Milyar}$$

$$P1 \rightarrow \# \text{Çevrim}(P1) = \# \text{Buyruk}(P1) * \text{CPI}(P1) = 20 * 10^9 * 1.5 = 30 \text{ Milyar}$$

$$P2 \rightarrow \# \text{Çevrim}(P2) = \# \text{Buyruk}(P2) * \text{CPI}(P2) = 25 * 10^9 * 1.0 = 25 \text{ Milyar}$$

$$P3 \rightarrow \# \text{Çevrim}(P3) = \# \text{Buyruk}(P3) * \text{CPI}(P3) = 18.18 * 10^9 * 2.2 = 40 \text{ Milyar}$$

**SORU:** P1, P2 ve P3 adında 3 adet işlemci aynı ISA'yi çalıştırmaktadır.

P1 işlemcisi 3.0 GHz frekans ve 1.5 CPI değerine sahiptir.

P2 işlemcisi 2.5 GHz frekans ve 1.0 CPI değerine sahiptir.

P3 işlemcisi 4.0 GHz frekans ve 2.2 CPI değerine sahiptir.

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

$$\text{CPU clock cycles} = \text{Instructions for a program} \times \frac{\text{Average clock cycles per instruction}}$$

c) Yürütme zamanını %30 azaltmak istiyoruz fakat tasarımdaki bu değişiklik CPI'da %20 artışa neden oluyor. Bu hızlanmayı sağlamak için saat frekansı ne oranda artmalıdır?

$$\frac{\text{YürütmeZamanı}(Eski)}{\text{YürütmeZamanı}(Yeni)} = \left(\frac{10}{7}\right) = \left(\frac{\#Buyruk * CPI(Eski) * Freq(Eski)}{\#Buyruk * CPI(Yeni) * Freq(Yeni)}\right) = \left(\frac{1}{1.2}\right) \left(\frac{Freq(Eski)}{Freq(Yeni)}\right)$$

$$\left(\frac{Freq(Yeni)}{Freq(Eski)}\right) = \left(\frac{12}{7}\right) = 1.714$$

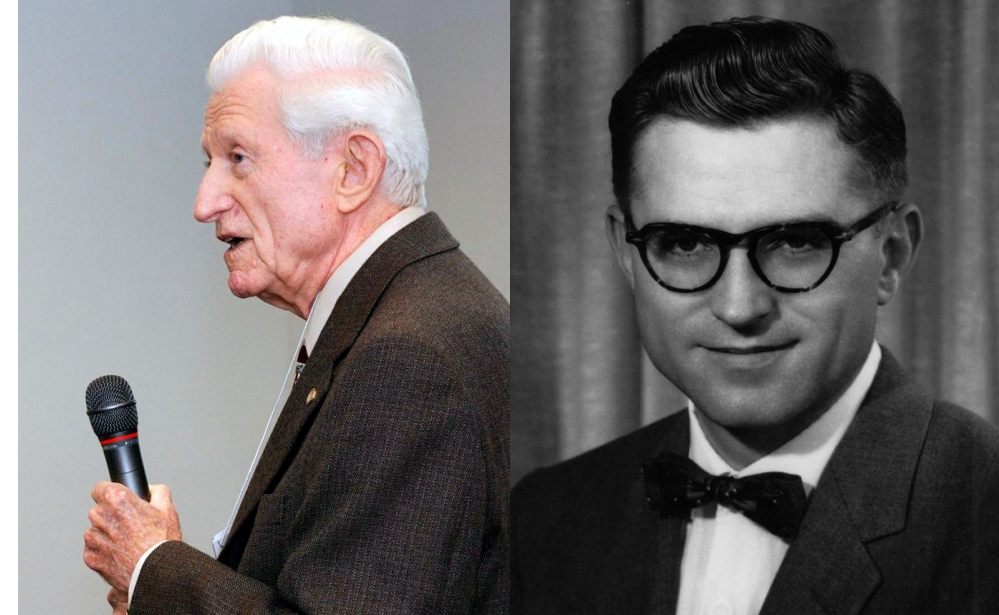
# AMDAHL YASASI

“The overall performance improvement gained by optimizing a single part of a system is limited by the fraction of time that the improved part is actually used” (Gene Amdahl, 1922 – 2015)

$$\text{Yürütme Zamanı (Geliştirme Sonrası)} = \left( \frac{\text{Geliştirmeden Etkilenen Kısımın Yürütme Zamanı}}{\text{Geliştirme Oranı}} \right) + \text{Geliştirmeden Etkilenmeyen Kısımın Yürütme Zamanı}$$

Örnek: 100 saniyede yürütülen bir programın, müşteri en çok 80 saniyede tamamlanmasını istiyor. Bu programda çarpma işlemleri toplam yürütme zamanının %10'u kadarıdır. Proje yöneticisi Amdahl yasasından bihaber olduğu için tasarım ekibine proje maliyetini sarsacak şekilde yüksek maaşla çok iyi bir çarpma donanım tasarımcısı katılıyor ve çarpma işleminin gecikmesini %50 kadar azaltmayı başarıyor. Yeni yürütme zamanı ne olur? Müşteri mutlu olur mu? Proje yöneticisi lisansta hangi derse önem göstermemiştir?

Yeni Yürütme Zamanı =  $10/2 + 90 = 95$  saniye  
Müşteri memnun değil. Proje yöneticisi bilgisayar mimarisi dersinde uyumuş.





# CREATING A RISC-V SW DEVELOPMENT ENVIRONMENT

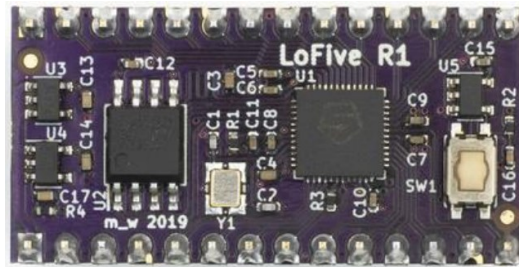
RISC-V ISA (Instruction Set Architecture) buyruklarını yürütebilmek ve debug edebilmek için çeşitli yöntemler mevcuttur.

1) RISC-V çekirdeğe sahip bir MCU ya da SoC içeren bir geliştirme kartı satın alıp ilgili IDE (Integrated Development Environment) üzerinde C ya da Assembly kod yazıp, compile edip, yürütülebilir (executable) bir dosya oluşturulur (.elf). Bu dosya IDE aracılığıyla geliştirme kartına yüklenir (JTAG). Bir debug protokolü ile (GDB) IDE üzerinde program çalıştırılır, debug edilir, registerlar ve memory gözlenir vs.

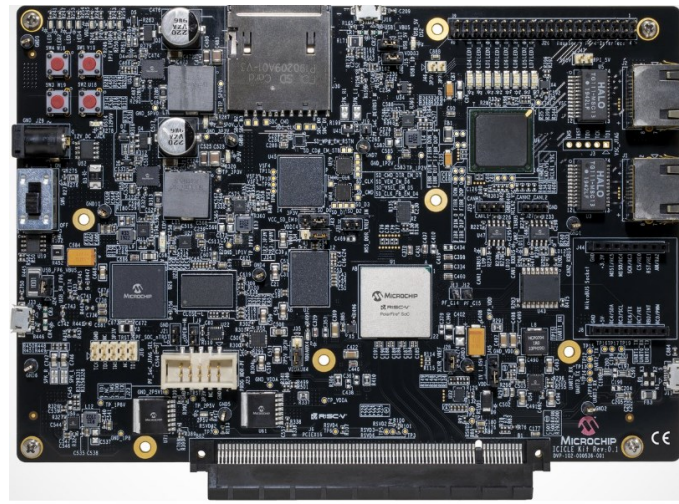
**SiFive – HiFive Unmatched**



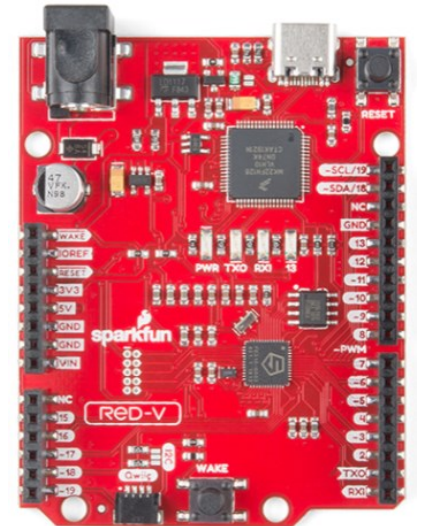
**GroupGets – LoFive R1**



**Microchip – Polarfire SoC**



**Sparkfun – RED-V Redboard**



# CREATING A RISC-V SW DEVELOPMENT ENVIRONMENT

2) Bir donanım emülatörü (hardware emulator) ile, RISC-V işlemcili bir donanım sistemi “emule edilir”. Emulasyon ortamında, Linux, FreeRTOS, Bare-Metal gibi OS ve kütüphaneler üzerinde çalışma yapılabilir. Örnek QEMU

[🏠](#) » [System Emulation](#) » [QEMU System Emulator Targets](#) » [RISC-V System emulator](#)

[🔗](#) [Edit on GitLab](#)

## RISC-V System emulator

QEMU can emulate both 32-bit and 64-bit RISC-V CPUs. Use the `qemu-system-riscv64` executable to simulate a 64-bit RISC-V machine, `qemu-system-riscv32` executable to simulate a 32-bit RISC-V machine.

QEMU has generally good support for RISC-V guests. It has support for several different machines. The reason we support so many is that RISC-V hardware is much more widely varying than x86 hardware. RISC-V CPUs are generally built into “system-on-chip” (SoC) designs created by many different companies with different devices, and these SoCs are then built into machines which can vary still further even if they use the same SoC.

For most boards the CPU type is fixed (matching what the hardware has), so typically you don’t need to specify the CPU type by hand, except for special cases like the virt board.

<https://www.qemu.org/docs/master/system/target-riscv.html>



# CREATING A RISC-V SW DEVELOPMENT ENVIRONMENT

## Board-specific documentation

Unfortunately many of the RISC-V boards QEMU supports are currently undocumented; you can get a complete list by running `qemu-system-riscv64 --machine help`, or `qemu-system-riscv32 --machine help`.

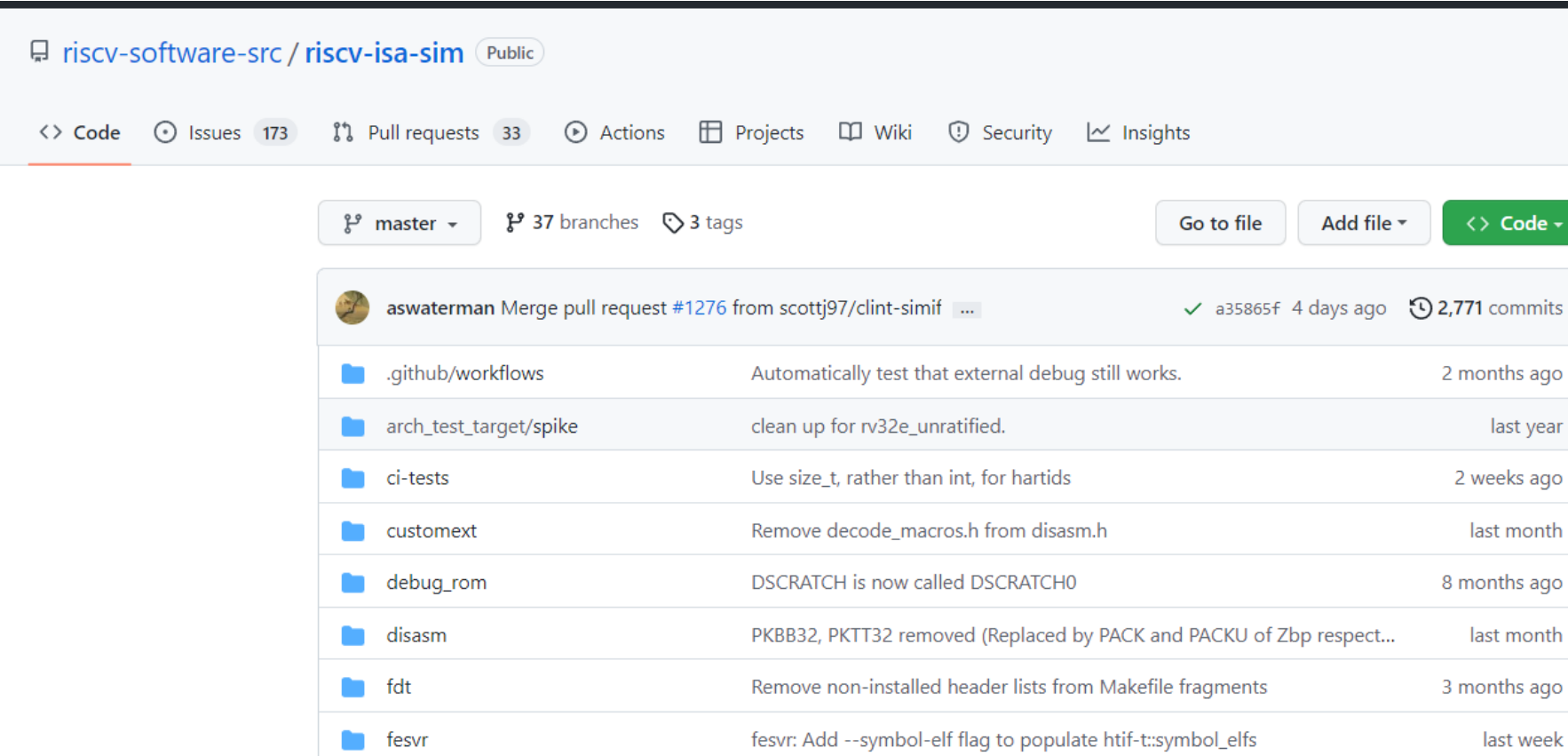
- Microchip PolarFire SoC Icicle Kit (`microchip-icicle-kit`)
- Shakti C Reference Platform (`shakti_c`)
- SiFive HiFive Unleashed (`sifive_u`)
- ‘virt’ Generic Virtual Platform (`virt`)

<https://risc-v-getting-started-guide.readthedocs.io/en/latest/linux-qemu.html>

<https://wiki.qemu.org/Documentation/Platforms/RISCV>

# CREATING A RISC-V SW DEVELOPMENT ENVIRONMENT

3) Bir instruction set simulator ile her bir buyruk adım adım çalıştırılıp yazmaç dosyası (register file), control and status registers (CSRs) ve bellek (memory) içeriği incelenebilir. Örnekler:



The screenshot shows the GitHub repository page for `riscv-software-src / riscv-isa-sim`. The repository is public and has 173 issues, 33 pull requests, and 2,771 commits. The current branch is `master`, with 37 branches and 3 tags. A recent pull request by `aswaterman` is highlighted, titled "Merge pull request #1276 from scottj97/clint-simif". Below the pull request, a list of recent commits is shown, each with a folder icon, a description, and a timestamp.

Folder	Description	Timestamp
<code>.github/workflows</code>	Automatically test that external debug still works.	2 months ago
<code>arch_test_target/spike</code>	clean up for rv32e_unratified.	last year
<code>ci-tests</code>	Use <code>size_t</code> , rather than <code>int</code> , for hartids	2 weeks ago
<code>customext</code>	Remove <code>decode_macros.h</code> from <code>disasm.h</code>	last month
<code>debug_rom</code>	<code>DSCRATCH</code> is now called <code>DSCRATCH0</code>	8 months ago
<code>disasm</code>	<code>PKBB32</code> , <code>PKTT32</code> removed (Replaced by <code>PACK</code> and <code>PACKU</code> of Zbp respect...	last month
<code>fdt</code>	Remove non-installed header lists from Makefile fragments	3 months ago
<code>fesvr</code>	<code>fesvr</code> : Add <code>--symbol-elf</code> flag to populate <code>htif-t::symbol_elfs</code>	last week

<https://github.com/riscv-software-src/riscv-isa-sim>

# CREATING A RISC-V SW DEVELOPMENT ENVIRONMENT

3) Bir instruction set simulator ile her bir buyruk adım adım çalıştırılıp yazmaç dosyası (register file), control and status registers (CSRs) ve bellek (memory) içeriği incelenebilir. Örnekler:

## riscvOVPsim - Free Imperas RISC-V Instruction Set Simulator

**riscvOVPsim** - RISC-V Instruction Set Simulator (ISS) - fast, simple, easy to use, cross software development for embedded systems

The **riscvOVPsim** ISS is an ideal starting point for an embedded software development project.

**riscvOVPsim** allows the development and debug of code for the target RISC-V processor on an x86 host PC with the minimum of setup and effort. It simply requires the cross compilation of your application and running riscvOVPsim with an argument to specify the name of the application object.

It is **FREE** to download from

GitHub here: [github.com/riscv-ovpsim](https://github.com/riscv-ovpsim) for FREE!

OVPworld here: [OVPworld.org/riscv-ovpsim-plus](https://www.ovpworld.org/riscv-ovpsim-plus) for FREE! enhanced riscvOVPsim including the new test suite for RISC-V vector extensions

<https://www.imperas.com/riscvovpsim-free-imperas-risc-v-instruction-set-simulator>

# CREATING A RISC-V SW DEVELOPMENT ENVIRONMENT

3) Bir instruction set simulator ile her bir buyruk adım adım çalıştırılıp yazmaç dosyası (register file), control and status registers (CSRs) ve bellek (memory) içeriği incelenebilir. Örnekler:

mortbopet / Ripes Public

<> Code Issues 34 Pull requests 3 Discussions Actions Projects 1 Wiki Security Insights

master 11 branches 15 tags Go to file Add file <> Code

mortbopet Fix auto clock timer not stopping when a breakpoint is hit #260 ✓ 2a03b38 on Jan 14 1,492 commits

.github	Add docker-build.yml for building docker image on PRs	last year
appdir/usr/share	AppStream metainfo Flathub compliance (#187)	2 years ago
docker	Add missing include and make MSVC and docker happy	last year
docs	Fix auto clock timer not stopping when a breakpoint is hit #260	2 months ago
examples	Format consoleReading example	last year
external	Update VSRTL version, release notes	7 months ago
resources	Updated documentation to reflect the change in cache terminology (#2...	5 months ago
src	Fix auto clock timer not stopping when a breakpoint is hit #260	2 months ago
test	Replace numStages with more elaborate processor structure info #203	last year

<https://github.com/mortbopet/Ripes>

100  
1010  
01  
Editor

Processor

Cache

Memory

I/O

Source code

Input type: ☒ Assembly ☐ C Executable codeView mode: ☐ Binary ☒ Disassembled

GPR

```
31      add    a3, a3, t6
32      slli   a3, a3, 8
33      and    t5, a3, a7
34      mv     a5, t2
35      mv     a3, a1
36 nextPixel:
37      divu   s0, a4, a1
38      add    s0, s0, t6
39      add    s1, t4, a4
40      divu   s1, s1, t3
41      add    s1, s1, t6
42      slli   s0, s0, 16
43      and    s0, s0, t0
44      or     s0, t5, s0
45      andi   s1, s1, 255
46      or     s0, s0, s1
47      sw     s0, 0(a5)
48      addi   a3, a3, -1
49      addi   a5, a5, 4
50      addi   a4, a4, 255
51      bnez   a3, nextPixel
52      addi   t6, t6, 1
```

```
4c:      01f686b3      add x13 x13 x31
50:      00869693      slli x13 x13 8
54:      0116ff33      and x30 x13 x17
58:      00038793      addi x15 x7 0
5c:      00058693      addi x13 x11 0

00000060 <nextPixel>:
60:      02b75433      divu x8 x14 x11
64:      01f40433      add x8 x8 x31
68:      00ee84b3      add x9 x29 x14
6c:      03c4d4b3      divu x9 x9 x28
70:      01f484b3      add x9 x9 x31
74:      01041413      slli x8 x8 16
78:      00547433      and x8 x8 x5
7c:      008f6433      or x8 x30 x8
80:      0ff4f493      andi x9 x9 255
84:      00946433      or x8 x8 x9
88:      0087a023      sw x8 0 x15
8c:      fff68693      addi x13 x13 -1
90:      00478793      addi x15 x15 4
94:      0ff70713      addi x14 x14 255
98:      fc0694e3      bne x13 x0 -56 <nextPixel>
9c:      001f8f93      addi x31 x31 1
```

Name	Alias	Value
x0	zero	0x00000000
x1	ra	0x00000000
x2	sp	0x7fffffe0
x3	gp	0x10000000
x4	tp	0x00000000
x5	t0	0x00ff0000
x6	t1	0x00000002
x7	t2	0xf000118
x8	s0	0x00851600
x9	s1	0x00000057
x10	a0	0xf0000000
x11	a1	0x00000023
x12	a2	0x00000019
x13	a3	0x00000011
x14	a4	0x000011ee
x15	a5	0xf000160
x16	a6	0x0000008c
x17	a7	0x0000ff00
x18	s2	0x00000000
x19	s3	0x00000000
x20	s4	0x00000000

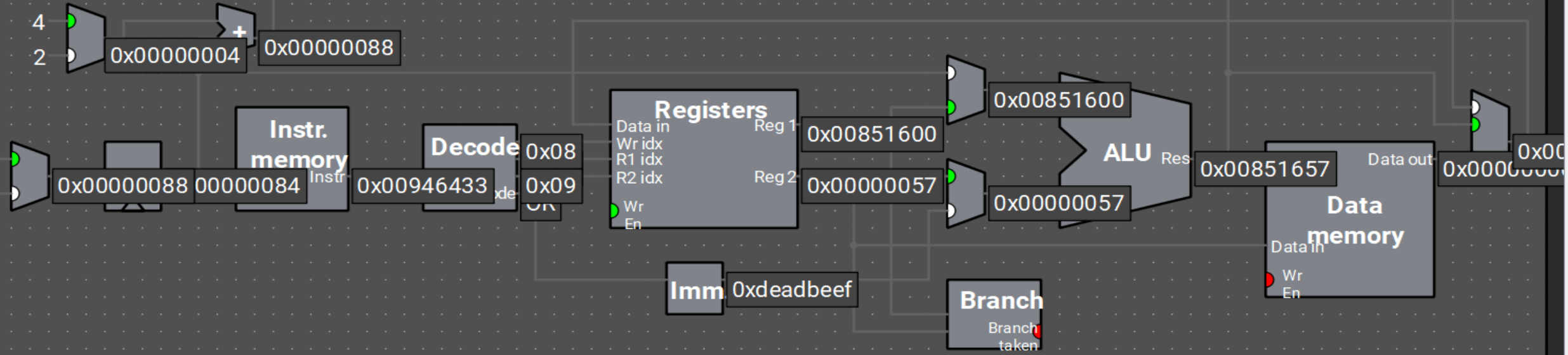
Display type:

Hex



# Single Cycle RISC-V Processor

or x8 x8 x9



Memory viewer

Address	Word	Byte 0	Byte 1	Byte 2	Byte 3
0x0000002c	0x00ff02b7	0xb7	0x02	0xff	0x00
0x00000028	0xf0068893	0x93	0x88	0x06	0xf0
0x00000024	0x000106b7	0xb7	0x06	0x01	0x00
0x00000020	0x00259813	0x13	0x98	0x25	0x00
0x0000001c	0x00b60e33	0x33	0x0e	0xb6	0x00
0x00000018	0x00000f93	0x93	0x0f	0x00	0x00
0x00000014	0x00912423	0x23	0x24	0x91	0x00
0x00000010	0x00812623	0x23	0x26	0x81	0x00
0x0000000c	0xff010113	0x13	0x01	0x01	0xff
0x00000008	0x01900613	0x13	0x06	0x90	0x01
0x00000004	0x02300593	0x93	0x05	0x30	0x02
0x00000000	0xf0000537	0x37	0x05	0x00	0xf0

Memory map

Name	Size	Range
.text	180	0x00000000 - 0x000000b3
.data	0	0x10000000 - 0x0fffffff
.bss	0	0x11000000 - 0x10ffffff
LED Matrix 0	3500	0xf0000000 - 0xf0000dab

Cache configuration:

Preset:

\_\_\_\_\_



$2^N$  Lines:

5

Repl. policy:

LRU

$2^N$  Ways:

0

Wr. hit:

Write-back

2<sup>N</sup> Words/Line:

2

Wr. miss:

Write allocate

Plot configuration:

Numerator

Hits

Denominator

Access count ☒ Ratio☒ Moving avg.50 cyc.

### Statistics:

Size (bits):

4896

Hit rate: 0.7444

0.7444

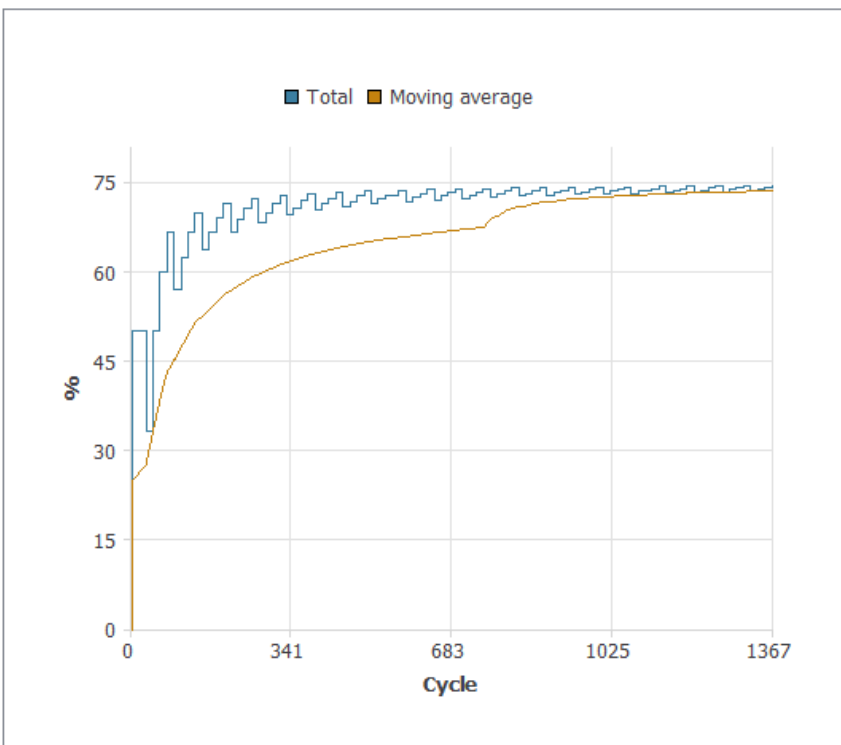
Writebacks:

0

Hits:

67

Misses:

23

0 1367



Access address

[illegible]

Index	V	D	Tag	Word 0	Word 1	Word 2	Word 3
0	1	1	0x00780000	0x00000000	0x00070004	0x000e0008	0x00000000
1	1	1	0x00780000	0x001d0011	0x00240015	0x002b0019	0x00000000
2	1	1	0x00780000	0x003a0022	0x00410026	0x0048002a	0x00000000
3	1	1	0x00780000	0x00570033	0x005e0037	0x0066003b	0x00000000
4	1	1	0x00780000	0x00740044	0x007b0048	0x0083004c	0x00000000
5	1	1	0x00780000	0x00910055	0x00990059	0x00a0005d	0x00000000
6	1	1	0x00780000	0x00ae0066	0x00b6006a	0x00bd006e	0x00000000
7	1	1	0x00780000	0x00cc0077	0x00d3007b	0x00da007f	0x00000000
8	1	1	0x00780000	0x00e90088	0x00f0008c	0x00f70090	0x00000000
9	1	1	0x00780000	0x00080b09	0x000f0b0d	0x00160b12	0x00000000
10	1	1	0x00780000	0x00250b1a	0x002c0b1e	0x00340b23	0x00000000
11	1	1	0x00780000	0x00420b2b	0x00490b2f	0x00510b34	0x00000000
12	1	1	0x00780000	0x005f0b3c	0x00670b40	0x006e0b45	0x00000000
13	1	1	0x00780000	0x007c0b4d	0x00840b51	0x008b0b56	0x00000000
14	1	1	0x00780000	0x009a0b5e	0x00a10b62	0x00a80b67	0x00000000
15	1	1	0x00780000	0x00b70b6f	0x00be0b73	0x00c50b78	0x00000000
16	1	1	0x00780000	0x00d40b80	0x00db0b84	0x00e20b89	0x00000000
17	1	1	0x00780000	0x00f10b91	0x00f80b95	0x0002160a	0x00000000
18	1	1	0x00780000	0x00101613	0x00171617	0x001f161b	0x00000000
19	1	1	0x00780000	0x002d1624	0x00351628	0x003c162c	0x00431630
20	1	1	0x00780000	0x004a1635	0x00521639	0x0059163d	0x00000000
21	1	1	0x00780000	0x00681646	0x006f164a	0x0076164e	0x00000000
22	0	0					
23	0	0					
24	0	0					
25	0	0					
26	0	0					
27	0	0					
28	0	0					
29	0	0					
30	1	1	0x003fffffff	0x00000000	0x00000000	0x00000000	0x00000000
31	0	0					

Devices

LED Matrix

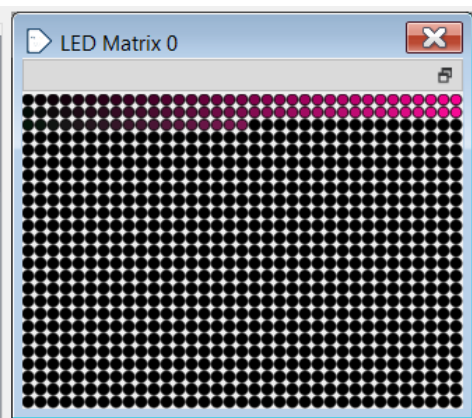
Switches

D-Pad

I/O exports

```
#ifndef RIPES_IO_HEADER
#define RIPES_IO_HEADER
// *****
// * LED_MATRIX_0
// *****
#define LED_MATRIX_0_BASE (0xf00000)
#define LED_MATRIX_0_SIZE (0xdac)
#define LED_MATRIX_0_WIDTH (0x23)
#define LED_MATRIX_0_HEIGHT

#endif // RIPES_IO_HEADER
```



Select Processor

RISC-V

32-bit

Single-cycle processor

5-stage processor w/o forwarding or hazard detection

5-stage processor w/o hazard detection

5-Stage processor w/o forwarding unit

5-stage processor

6-stage dual-issue processor

64-bit

Single-cycle processor

5-stage processor w/o forwarding or hazard detection

5-stage processor w/o hazard detection

5-Stage processor w/o forwarding unit

5-stage processor

6-stage dual-issue processor

Name: Single-cycle processor

ISA: RV32I

ISA Exts. ☒ M ☒ C

Layout Standard

A single cycle processor

Description:

Register initialization

x2 (sp) 0x7ffffff0

x3 (gp) 0x10000000

+

OK

Cancel

LED Matrix 0

Each LED maps to a 24-bit register storing an RGB color value, with B stored in the least significant byte.  
The byte offset of the LED at coordinates (x, y) is:  
$$\text{offset} = (y + x * N\_LEDS\_ROW) * 4$$

Parameters

Name	Value
Height	25
Width	35
LED size	8

Register map

Name	Address	R/W?	Size
LED_0	0x0	R/W	24
LED_1	0x4	R/W	24
LED_2	0x8	R/W	24
LED_3	0xc	R/W	24
LED_4	0x10	R/W	24

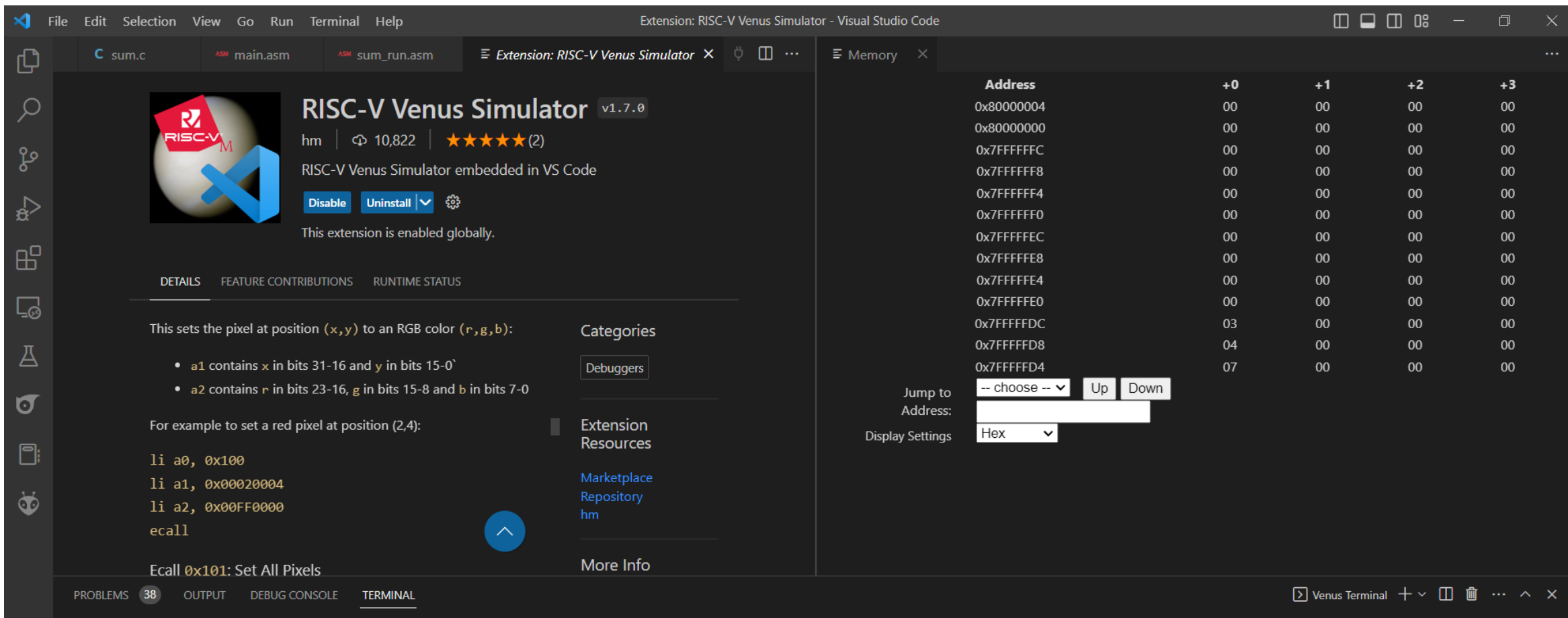
Exports

```
#define LED_MATRIX_0_BASE (0xf0000000)
#define LED_MATRIX_0_SIZE (0xdac)
#define LED_MATRIX_0_WIDTH (0x23)
#define LED_MATRIX_0_HEIGHT (0x19)
```

Processor: Single-cycle processor ISA: RV32IMC

# CREATING A RISC-V SW DEVELOPMENT ENVIRONMENT

3) Bir instruction set simulator ile her bir buyruk adım adım çalıştırılıp yazmaç dosyası (register file), control and status registers (CSRs) ve bellek (memory) içeriği incelenebilir. Örnekler:



The screenshot shows the Visual Studio Code interface with the RISC-V Venus Simulator extension installed. The extension page on the left displays the version (v1.7.0), a heart icon, 10,822 downloads, and a 5-star rating. It includes buttons for 'Disable', 'Uninstall', and a settings gear. Below this, the 'DETAILS' tab shows a description of the extension's functionality: setting a pixel at position (x,y) to an RGB color (r,g,b) using assembly instructions. The instructions provided are:

```
li a0, 0x100
li a1, 0x0020004
li a2, 0x00FF0000
ecall
```

The 'Ecall 0x101: Set All Pixels' instruction is also shown. The 'Memory' view on the right displays a table of memory addresses and their corresponding values:

Address	+0	+1	+2	+3
0x80000004	00	00	00	00
0x80000000	00	00	00	00
0x7FFFFFFC	00	00	00	00
0x7FFFFFF8	00	00	00	00
0x7FFFFFF4	00	00	00	00
0x7FFFFFF0	00	00	00	00
0x7FFFFFEC	00	00	00	00
0x7FFFFFE8	00	00	00	00
0x7FFFFFE4	00	00	00	00
0x7FFFFFE0	00	00	00	00
0x7FFFFFDC	03	00	00	00
0x7FFFFFD8	04	00	00	00
0x7FFFFFD4	07	00	00	00

Below the memory table, there are controls for 'Jump to Address' and 'Display Settings' (set to Hex). The bottom status bar shows 'Venus Terminal' and other window management icons.

```
1 addi    sp,sp,-32
```

```

2  sw    s0,28(sp)
3  addi   s0,sp,32
4  li     a5,3
5  sw     a5,-20(s0)
6  li     a5,4
7  sw     a5,-24(s0)
8  lw     a4,-20(s0)
9  lw     a5,-24(s0)
10 add    a5,a4,a5
11 sw     a5,-28(s0)
12 li     a5,0
13 mv     a0,a5
14 lw     s0,28(sp)
15 addi   sp,sp,32
16 ret

```

PRIV: 0x3

## Integer

```
x00 (zero): 0x00000000
x01 (ra)  : 0x00000000
x02 (sp)  : 0x7FFFFFF0
x03 (gp)  : 0x10000000
x04 (tp)  : 0x00000000
x05 (t0)  : 0x00000000
x06 (t1)  : 0x00000000
x07 (t2)  : 0x00000000
x08 (s0)  : 0x00000000
x09 (s1)  : 0x00000000
x10 (a0)  : 0x00000000
x11 (a1)  : 0x00000000
x12 (a2)  : 0x00000000
x13 (a3)  : 0x00000000
x14 (a4)  : 0x00000000
x15 (a5)  : 0x00000000
x16 (a6)  : 0x00000000
x17 (a7)  : 0x00000000
x18 (a7)  : 0x00000000
```

## > CALL STACK

## ✓ BREAKPOINTS

## > VENUS OPTIONS

## > PERIPHERALS

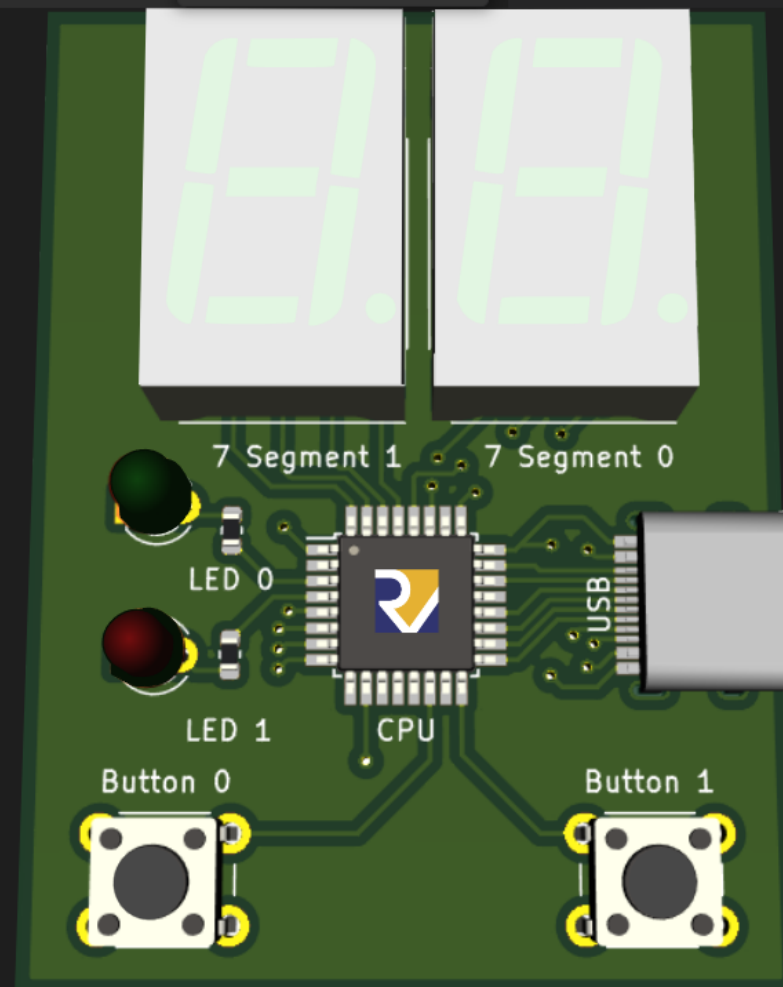
## > REGISTERS

## > MEMORY

## > DISASSEMBLY

PROBLEMS 38 OUTPUT DEBUG CONSOLE TERMINAL

Starting program C:\Users\ayken\OneDrive\Documents\Lectures\ERCIYES\2023 Spring Arch\riscv\_examples\sum\sum run.asm





# CREATING A RISC-V SW DEVELOPMENT ENVIRONMENT

4) RISC-V General Purpose host PC ??? (Bu sayede x86 gibi bir host üzerinde cross-compile ile target için binary üretmek yerine, doğrudan host üzerinde geliştirme yapılabilir)

## World's First Laptop with RISC-V Processor Now Available

By Ian Evenden published October 04, 2022


Alibaba gets the first RISC-V laptop on sale

       Comments (11)



Home / All Industries / Consumer Electronics / Computer Hardware & Software / Laptops / Personal & Home laptops



 View larger image



DC-ROMA RISC-V Development Laptop Premium Standard Basic Packages and Services

7 buyers

FOB Reference Price: [Get latest price](#)

**\$1,499.00 - \$4,999.00** / box | 1 box/boxes (Min. order)



Save up to US \$30 with PayPal



Benefits: Quick refunds on orders under US \$1,000

[Claim now](#)

Color



Hard Drive Capacity

256GB

Model Number

Premium-USD4,999

Standard-USD2,499

Basic-USD1,499

Customization: Premium-Engraving (Min. order 1 box)

[https://www.alibaba.com/product-detail/DC-ROMA-RISC-V-Development-Laptop\\_1600610157163.html](https://www.alibaba.com/product-detail/DC-ROMA-RISC-V-Development-Laptop_1600610157163.html)

# PEKİ RISC-V BUYRUKLARINDAN ASSEMBLY KOD NASIL OLUŞTURACAĞIZ?

- 1) Kendimiz yazabiliriz
- 2) Başkaları yazmış olabilir, onları kullanırız
- 3) Derlenen bir dille (C/C++) kod yazarız. RISC-V toolchain ile derleriz, oluşan binary dosyayı disassemble ederiz.

## 🔗 RISC-V GNU Compiler Toolchain

This is the RISC-V C and C++ cross-compiler. It supports two build modes: a generic ELF/Newlib toolchain and a more sophisticated Linux-ELF/glibc toolchain.

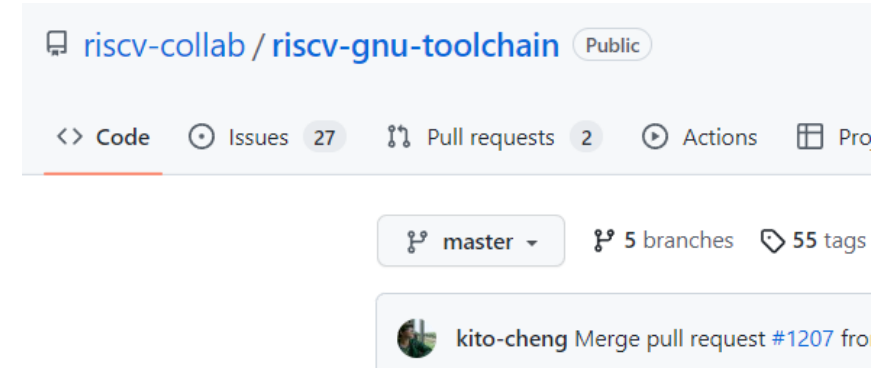
### Getting the sources

This repository uses submodules, but submodules will fetch automatically on demand, so `--recursive` or `git submodule update --init --recursive` is not needed.

```
$ git clone https://github.com/riscv/riscv-gnu-toolchain
```

Warning: git clone takes around 6.65 GB of disk and download size

<https://github.com/riscv-collab/riscv-gnu-toolchain>



# TOOLCHAIN: COMPILER, LINKER, LIBRARIES, DEBUGGER, ASSEMBLER

[18 languages](#) ▾[Article](#) [Talk](#)[Read](#) [Edit](#) [View history](#)

From Wikipedia, the free encyclopedia

In [software](#), a **toolchain** is a set of [programming tools](#) that is used to perform a complex [software development](#) task or to create a software product, which is typically another computer program or a set of related programs. In general, the tools forming a toolchain are executed consecutively so the output or resulting [environment state](#) of each tool becomes the input or starting environment for the next one, but the term is also used when referring to a set of related tools that are not necessarily executed consecutively.<sup>[1][2][3]</sup>




A simple software development toolchain may consist of a [compiler](#) and [linker](#) (which transform the source code into an [executable program](#)), [libraries](#) (which provide interfaces to the [operating system](#)), and a [debugger](#) (which is used to [test](#) and [debug](#) created programs). A complex software product such as a [video game](#) needs tools for preparing [sound effects](#), music, [textures](#), [3-dimensional models](#) and [animations](#), together with additional tools for combining these resources into the finished product.<sup>[1][2]</sup>

## What should a toolchain include?

Toolchains may vary depending on the team using them and the product being delivered.

However, basic toolchains often include the following components:

- **Assembler** -- converts assembly language into code.
- **Linker** -- links a set of program files together into a single program.
- **Debugger** -- used to test and debug programs.
- **[Compiler](#)** -- used to generate executable code from the source code of a program.
- **[Runtime libraries](#)** -- used to interface with an operating system (OS). It enables the program to reference external functions and resources. An API is an example of this.





 Test Toolchains **passing**
 license **GPL-2.0**
 Chat **on gitter**

- [Available Toolchains](#)
- [Download](#)
- [Installation](#)

The toolchains were built according to the instructions of the [official RISC-V GNU Compiler Toolchain repository](#) using **Ubuntu** on a **64-bit x86 machine** (actually on Ubuntu on Windows). Most of the provided toolchains (the non-multilib versions) also support standard ISA extensions like A, C and M. These prebuilt toolchains are part of the [NEORV32 RISC-V Processor](#) project.

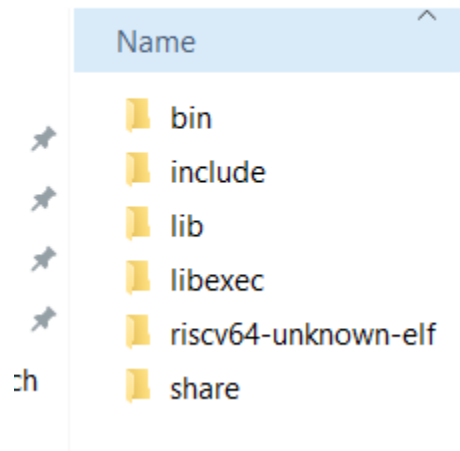
## Available Toolchains
































Toolchain prefix: `riscv32-unknown-elf` or `riscv64-unknown-elf` (see the individual releases)

Release (tag)	Download archive	GCC	binutils	<code>march</code>	<code>mabi</code>	c-lib
 <code>rv32i-4.0.0</code>	 <a href="#">download</a>	<code>12.1.0</code>	<code>2.39</code>	<code>rv32i</code>	<code>ilp32</code>	<code>newlib</code>
 <code>rv64imc-3.0.0</code>	 <a href="#">download</a>	<code>12.1.0</code>	<code>2.39</code>	multilib: <code>rv32i</code> <code>rv32ic</code> <code>rv32im</code> <code>rv32imc</code>	<code>ilp32</code>	<code>newlib</code>
 <code>rv32i-2.0.0</code>	 <a href="#">download</a>	<code>10.2.0</code>	<code>2.35</code>	<code>rv32i</code>	<code>ilp32</code>	<code>newlib</code>
 <code>rv32e-1.0.0</code>	 <a href="#">download</a>	<code>10.1.0</code>	<code>2.34</code>	<code>rv32e</code>	<code>ilp32e</code>	<code>newlib</code>

# TOOLCHAIN: COMPILER, LINKER, LIBRARIES, DEBUGGER, ASSEMBLER

Linux > Ubuntu > opt > riscv



- |  |   |
|--|---|
|  riscv64-unknown-elf-addr2line    |  riscv64-unknown-elf-gcov-dump     |
|  riscv64-unknown-elf-ar           |  riscv64-unknown-elf-gcov-tool     |
|  riscv64-unknown-elf-as           |  riscv64-unknown-elf-gdb           |
|  riscv64-unknown-elf-c++          |  riscv64-unknown-elf-gdb-add-index |
|  riscv64-unknown-elf-c++filt      |  riscv64-unknown-elf-gprof         |
|  riscv64-unknown-elf-cpp          |  riscv64-unknown-elf-ld            |
|  riscv64-unknown-elf-elfedit      |  riscv64-unknown-elf-ld.bfd        |
|  riscv64-unknown-elf-g++          |  riscv64-unknown-elf-lto-dump      |
|  riscv64-unknown-elf-gcc          |  riscv64-unknown-elf-nm            |
|  riscv64-unknown-elf-gcc-12.1.0   |  riscv64-unknown-elf-objcopy       |
|  riscv64-unknown-elf-gcc-ar      |  riscv64-unknown-elf-objdump       |
|  riscv64-unknown-elf-gcc-nm     |  riscv64-unknown-elf-ranlib        |
|  riscv64-unknown-elf-gcc-ranlib |  riscv64-unknown-elf-readelf      |
|  riscv64-unknown-elf-gcov       |  riscv64-unknown-elf-run         |
|  |  riscv64-unknown-elf-size        |
|  |  riscv64-unknown-elf-strings     |
|  |  riscv64-unknown-elf-strip       |



RISC-V Donanım ve Yazılım Geliştirme için Linux gerekebiliyor. Pek çok açık-kaynak yazılım ve donanım projesi de Linux tabanlı distro'lar için hazırlanıyor. Linux komutlarını nasıl çalıştırabilirim?

- 1) Herhangi bir Linux distro (Ubuntu, openSUSE, CentOS, Mint, etc.) bilgisayara kurabiliriz. Windows yanında dual-boot olarak çalışabilmektedir.
- 2) Bir Linux distro'yı Virtual Machine üzerinde (Oracle VM VirtualBox, VMware, etc)
- 3) WSL (Windows Subsystem for Linux) ile Windows üzerinde doğrudan bir distro çalıştırabiliriz.
- 4) Cygwin, Mingw gibi Linux komutlarını çalıştırabilen bir ortamı Windows üzerinde çalıştırabiliriz.

Ben genel olarak 1'i tercih ediyorum, bu sayede OS bütün aygıtlar üzerinde paylaşımsız tam performans çalışıyor. Basit gösterimler için WSL de yeterli oluyor (bu ders kapsamında RISC-V binary üretmek için mesela)

```
mbaykenar@DESKTOP-TSVKIOU:~/riscv_examples$ ls -l /
total 1976
lrwxrwxrwx 1 root root 7 Jan 4 00:40 bin -> usr/bin
drwxr-xr-x 2 root root 4096 Apr 18 2022 boot
drwxr-xr-x 11 root root 3040 Mar 20 22:35 dev
drwxr-xr-x 76 root root 4096 Mar 20 22:35 etc
drwxr-xr-x 3 root root 4096 Mar 13 23:45 home
-rwxrwxrwx 1 root root 1955960 Jan 1 1970 init
lrwxrwxrwx 1 root root 7 Jan 4 00:40 lib -> usr/lib
lrwxrwxrwx 1 root root 9 Jan 4 00:40 lib32 -> usr/lib32
lrwxrwxrwx 1 root root 9 Jan 4 00:40 lib64 -> usr/lib64
lrwxrwxrwx 1 root root 10 Jan 4 00:40 libx32 -> usr/libx32
drwx----- 2 root root 16384 Mar 13 23:33 lost+found
drwxr-xr-x 2 root root 4096 Jan 4 00:40 media
drwxr-xr-x 6 root root 4096 Mar 13 23:33 mnt
drwxr-xr-x 5 root root 4096 Mar 20 23:20 opt
dr-xr-xr-x 237 root root 0 Mar 20 22:35 proc
drwx----- 2 root root 4096 Jan 4 00:41 root
drwxr-xr-x 7 root root 140 Mar 20 22:37 run
lrwxrwxrwx 1 root root 8 Jan 4 00:40/sbin -> usr/sbin
drwxr-xr-x 8 root root 4096 Jan 4 00:41 snap
drwxr-xr-x 2 root root 4096 Jan 4 00:40 srv
dr-xr-xr-x 11 root root 0 Mar 20 22:35 sys
drwxrwxrwt 3 root root 4096 Mar 20 23:30 tmp
drwxr-xr-x 14 root root 4096 Jan 4 00:40 usr
drwxr-xr-x 13 root root 4096 Jan 4 00:41 var
mbaykenar@DESKTOP-TSVKIOU:~/riscv_examples$ pwd
/home/mbaykenar/riscv_examples
mbaykenar@DESKTOP-TSVKIOU:~/riscv_examples$ ls -l
total 12
drwxr-xr-x 2 mbaykenar mbaykenar 4096 Mar 16 16:12 hello_world
drwxr-xr-x 2 mbaykenar mbaykenar 4096 Mar 20 23:30 mult
drwxr-xr-x 2 mbaykenar mbaykenar 4096 Mar 14 10:29 sum
mbaykenar@DESKTOP-TSVKIOU:~/riscv_examples$
```

> Linux > Ubuntu > opt > riscv64i

Name

bin  
include  
lib  
libexec  
riscv64-unknown-elf  
share

.rch

# RISC-V COMPILATION & DISASSEMBLE

```
mbykenar@DESKTOP-TSVKIOU: ~/riscv_examples/hello_world
```

```
mbykenar@DESKTOP-TSVKIOU:~/riscv_examples/hello_world$ cat main.c
#include "stdio.h"

int main ()
{
    printf("Hello world!");
    return 0;
}mbykenar@DESKTOP-TSVKIOU:~/riscv_examples/hello_world$
```

export PATH=/opt/riscv64i/bin:\$PATH

riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -o main main.c

riscv64-unknown-elf-objdump --disassemble-all main > main.asm

```
mbykenar@DESKTOP-TSVKIOU: ~/riscv_examples/hello_world
```

```
mbykenar@DESKTOP-TSVKIOU:~/riscv_examples/hello_world$ ls -l
total 3484
-rwxr-xr-x 1 mbykenar mbykenar 225476 Mar 16 16:10 main
-rw-r--r-- 1 mbykenar mbykenar 3330486 Mar 14 10:23 main.asm
-rw-r--r-- 1 mbykenar mbykenar 83 Mar 13 23:00 main.c
mbykenar@DESKTOP-TSVKIOU:~/riscv_examples/hello_world$
```

```
main:      file format elf32-littleriscv
```

```
Disassembly of section .text:
```

```
00010094 <exit>:
```

```
10094: ff010113      addi sp,sp,-16
10098: 00000593      li a1,0
1009c: 00812423      sw s0,8(sp)
100a0: 00112623      sw ra,12(sp)
100a4: 00050413      mv s0,a0
100a8: 625020ef      jal ra,12ecc <__call_exitprocs>
100ac: 1b81a503      lw a0,440(gp) # 26a78 <_global_impure_ptr>
100b0: 03c52783      lw a5,60(a0)
100b4: 00078463      beqz a5,100bc <exit+0x28>
100b8: 000780e7      jalr a5
100bc: 00040513      mv a0,s0
100c0: 5e10f0ef      jal ra,1fea0 <_exit>
```

```
000100c4 <register_fini>:
```

```
100c4: 00000793      li a5,0
100c8: 00078863      beqz a5,100d8 <register_fini+0x14>
100cc: 00013537      lui a0,0x13
100d0: 6bc50513      addi a0,a0,1724 # 136bc <__libc_fini_array>
100d4: 7190206f      j 12fec <atexit>
100d8: 00008067      ret
```

```
000100dc <_start>:
```

```
100dc: 00016197      auipc gp,0x16
100e0: 7e418193      addi gp,gp,2020 # 268c0 <__global_pointer$>
100e4: 1cc18513      addi a0,gp,460 # 26a8c <__malloc_max_total_mem>
100e8: 22818613      addi a2,gp,552 # 26ae8 <__BSS_END__>
```

# WEEK3 ASSIGNMENT

Bilgisayarınızda yüklü değilse bir Linux distro yükleyin (Linux Environment başlığındaki 4 yöntemden birini tercih edebilirsiniz)

Aşağıdaki github linkindeki rv64imc pre-built risc-v toolchain binary'yi /opt/riscv64i/ klasörüne kaydedin

Aşağıdaki kodu ilk önce -march=rv32i, sonra da -march=rv32imc opsiyonlarıyla derleyip sonra disassemble edip iki <main> kod arasındaki farkı inceleyiniz

Yaptığınız işlemlerin sonucunu pdf formatında rapor haline getirip github hesabınızda private bir repo açıp o repoya beni contributor olarak ekleyin

<https://github.com/stnolting/riscv-gcc-prebuilt>

```
#include <stdio.h>

int main () {









int a = 3;
int b = 4;
int c;

c = a * b;

return 0;
}
```

## Available Toolchains

Toolchain prefix: `riscv32-unknown-elf` or `riscv64-unknown-elf` (see the individual releases)

Release (tag)	Download archive	GCC	binutils	march	mabi	c-lib
 rv32i-4.0.0	 download	12.1.0	2.39	rv32i	ilp32	newlib
 rv64imc-3.0.0	 download	12.1.0	2.39	multilib: rv32i rv32ic rv32im rv32imc	ilp32	newlib
 rv32i-2.0.0	 download	10.2.0	2.35	rv32i	ilp32	newlib
 rv32e-1.0.0	 download	10.1.0	2.34	rv32e	ilp32e	newlib

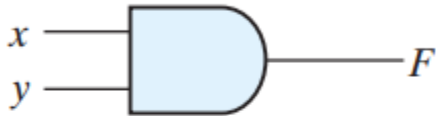

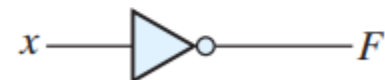

# COMBINATIONAL (BİRLEŞİK) DEVRELER

**Combinational (birleşik):** Devrenin çıkış sinyal(ler)i, yalnızca o anki giriş sinyal(ler)inin değerine bağlıdır. Devrede durum (state) belirten bir bellek (memory) bulunmamaktadır. Giriş sinyali değişir değişmez çıkış sinyaline etki eder.

**Sequential (sıralı):** Devrenin çıkış sinyal(ler)i, giriş sinyal(ler)inin ve devrede durum (state) belirten bellek (memory) değerine bağlıdır. Giriş sinyalinin değişimi her zaman çıkış sinyaline etki etmez, devrenin durumuna bağlıdır.



# MANTIKSAL KAPILAR

Name	Graphic symbol	Algebraic function	Truth table															
AND		$F = x \cdot y$	<table><tr><th><math>x</math></th><th><math>y</math></th><th><math>F</math></th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	$x$	$y$	$F$	0	0	0	0	1	0	1	0	0	1	1	1
$x$	$y$	$F$																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = x + y$	<table><tr><th><math>x</math></th><th><math>y</math></th><th><math>F</math></th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	$x$	$y$	$F$	0	0	0	0	1	1	1	0	1	1	1	1
$x$	$y$	$F$																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Inverter		$F = x'$	<table><tr><th><math>x</math></th><th><math>F</math></th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	$x$	$F$	0	1	1	0									
$x$	$F$																	
0	1																	
1	0																	
Buffer		$F = x$	<table><tr><th><math>x</math></th><th><math>F</math></th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	$x$	$F$	0	0	1	1									
$x$	$F$																	
0	0																	
1	1																	

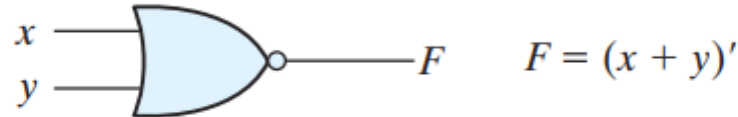
# MANTIKSAL KAPILAR

NAND



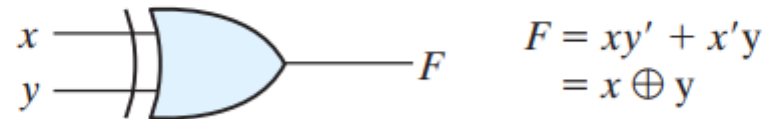
$x$	$y$	$F$
0	0	1
0	1	1
1	0	1
1	1	0

NOR



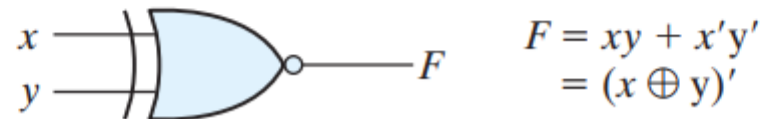
$x$	$y$	$F$
0	0	1
0	1	0
1	0	0
1	1	0

Exclusive-OR  
(XOR)



$x$	$y$	$F$
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-NOR  
or  
equivalence



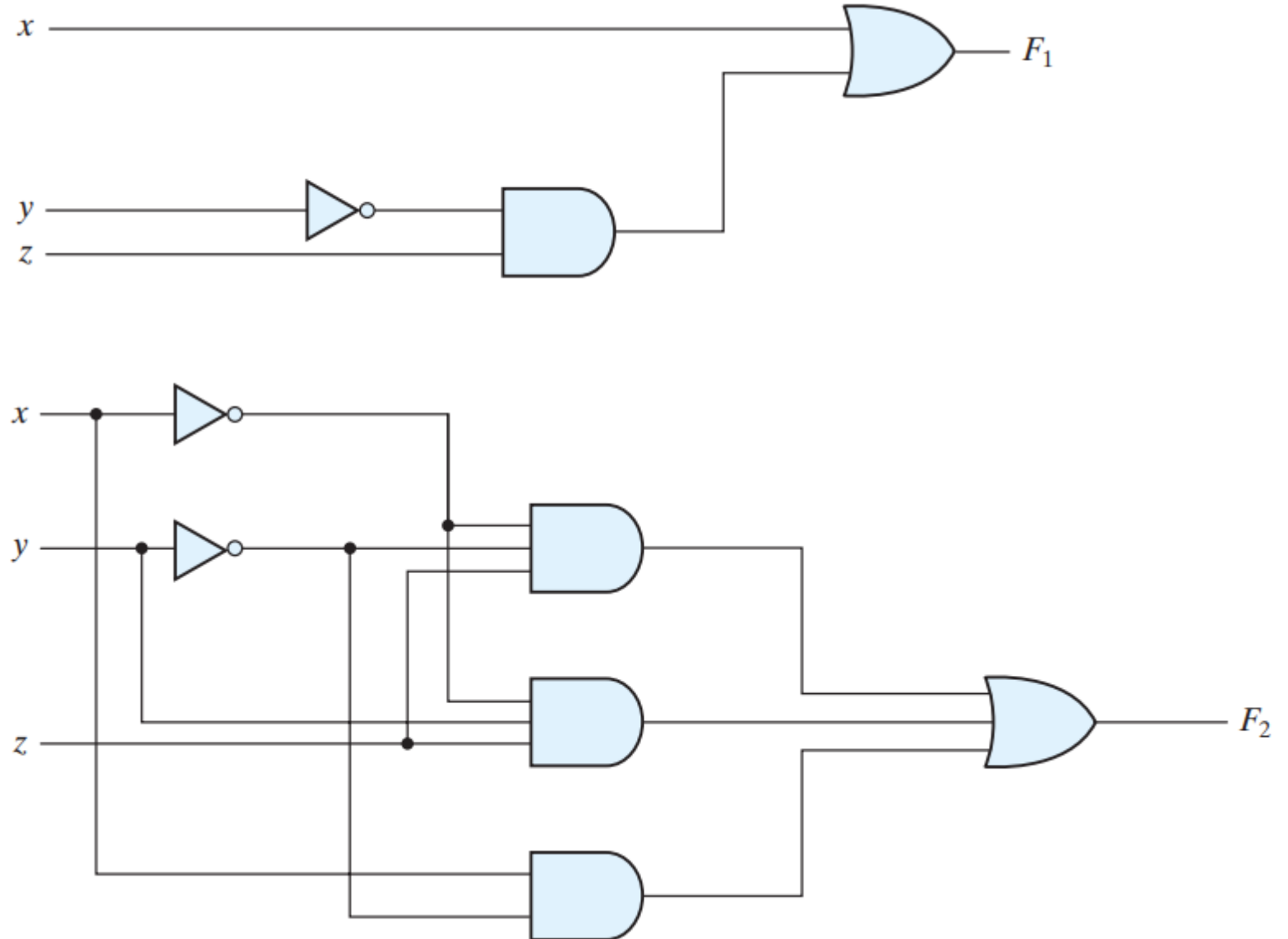
$x$	$y$	$F$
0	0	1
0	1	0
1	0	0
1	1	1

# Bool Fonksiyonları

$$F1 = x + y'z$$

$$F2 = x'y'z + x'yz + xy'$$

F1 & F2				
Giriş			Çıkış	
x	y	z	F1	F2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0



# Bool Fonksiyonları - Verilog

```
module bool_func
(
input x_i,
input y_i,
input z_i,
output f1_o,
output f2_o
);

assign f1_o = x_i | (!y_i & z_i);
assign f2_o = (!x_i & !y_i & z_i) |
               (!x_i & y_i & z_i) |
               (x_i & !y_i);

endmodule
```

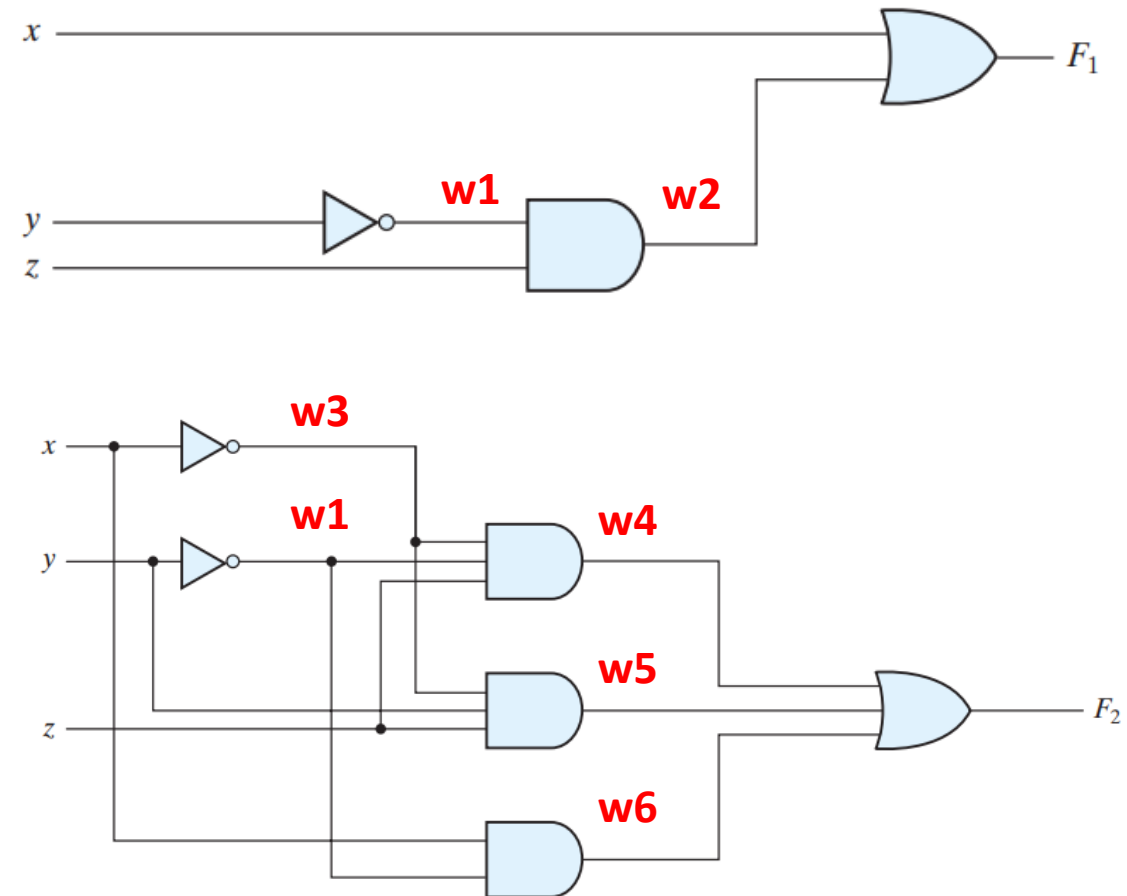
```
module bool_func_wire
(
input x_i,
input y_i,
input z_i,
output f1_o,
output f2_o
);

wire w1,w2,w3,w4,w5,w6;

assign w1 = !y_i;
assign w2 = w1 & z_i;
assign w3 = !x_i;
assign w4 = w3 & w1 & z_i;
assign w5 = w3 & y_i & z_i;
assign w6 = x_i & w1;

assign f1_o = x_i | w2;
assign f2_o = w4 | w5 | w6;

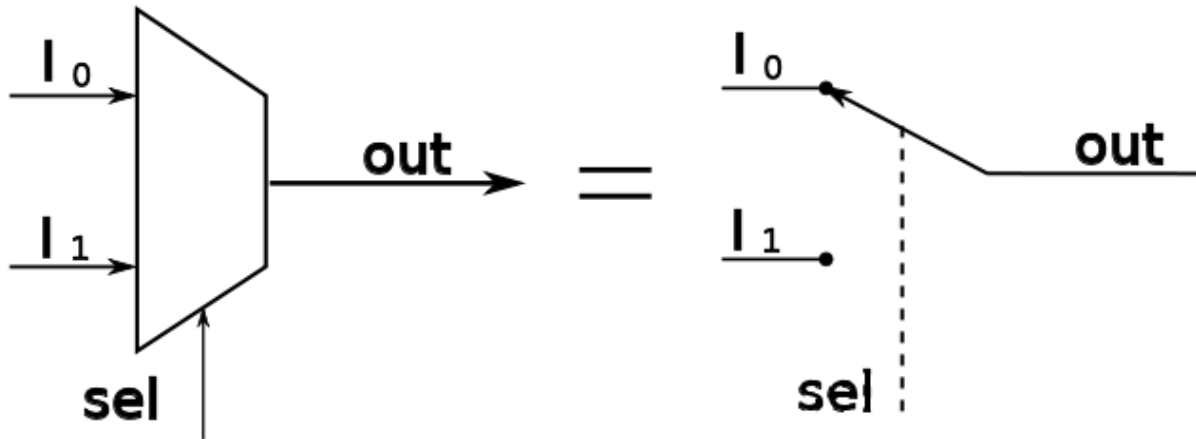
endmodule
```



# Multiplexer (Çoklayıcı)

*“a device that enables the simultaneous transmission of several messages or signals over one communications channel”*

*“a piece of electronic equipment that can send more than one electrical signal using only one connection”*

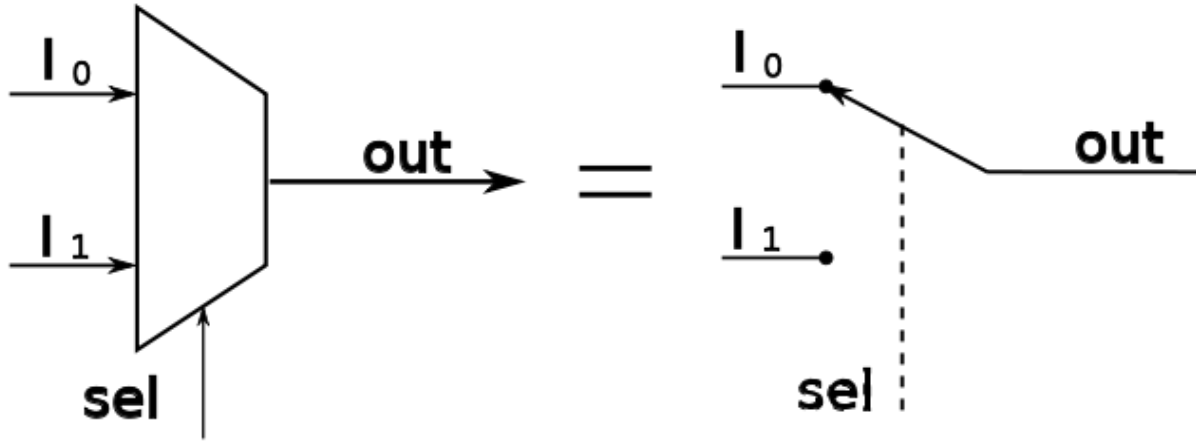


Giriş			Çıkış
sel	I1	I0	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

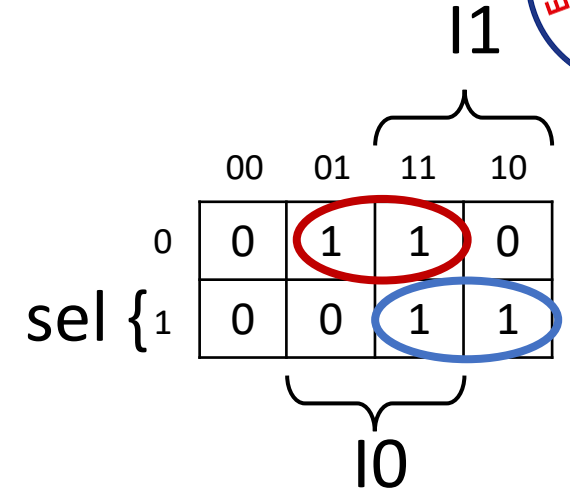
$$\begin{array}{c}
 \begin{array}{ccccc}
 & & & \text{I1} & \\
 & & & \{ & \\
 & 00 & 01 & 11 & 10 \\
 \text{sel} \begin{array}{l} 0 \\ 1 \end{array} & \begin{array}{|c|c|c|c|} \hline 0 & 1 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 \\ \hline \end{array} & \\
 & & \{ & & \\
 & & \text{I0} & & 
 \end{array}
 \end{array}$$

$$\text{out} = \text{sel}'\text{I0} + \text{I1sel}$$

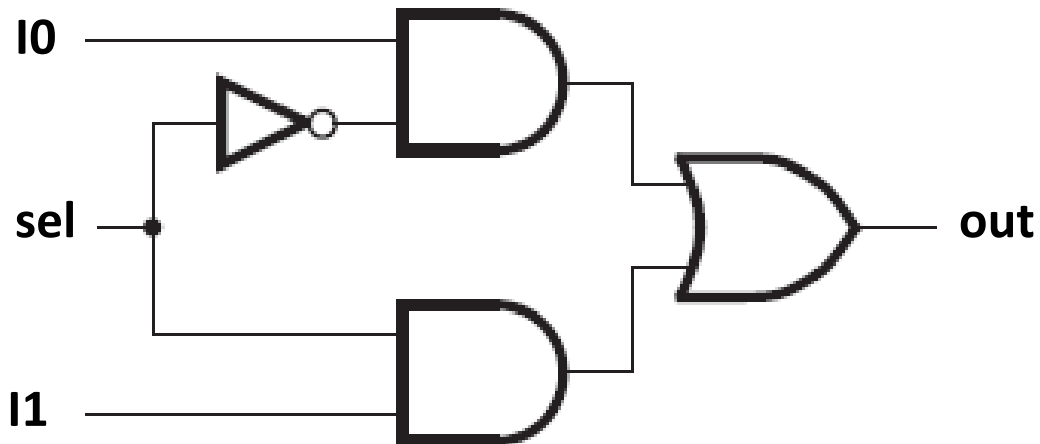
# 2x1 Multiplexer (Çoklayıcı)



Giriş			Çıkış
sel	I1	I0	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

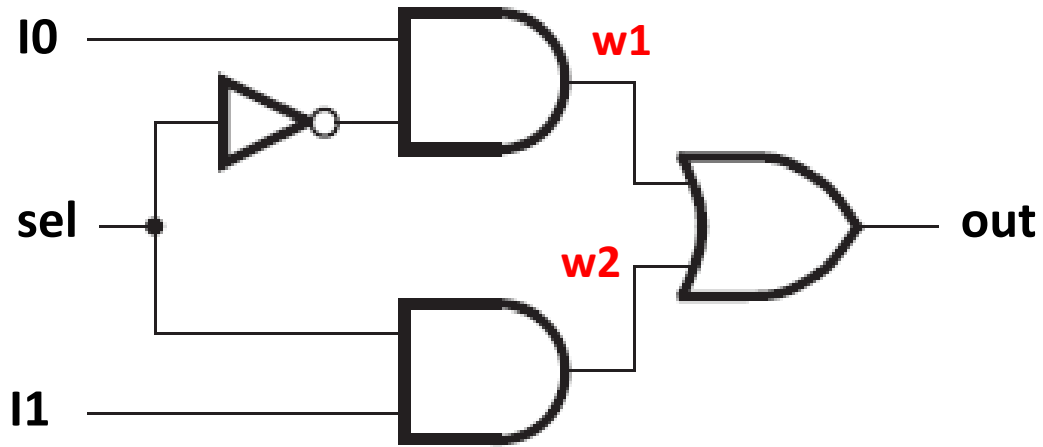


$$\text{out} = \text{sel}'I0 + I1\text{sel}$$





# 2x1 Multiplexer (Çoklayıcı) - Verilog



```
module mux_2_1
(
    input i0_i,
    input i1_i,
    input sel_i,
    output out_o
);
    wire w1,w2;

    assign w1      = i0_i & !sel_i;
    assign w2      = i1_i & sel_i;
    assign out_o   = w1 | w2;

    // conditional assignment
    // assign out_o = (sel_i == 1'b0) ? i0_i : i1_i;

endmodule
```

```
module mux_2_1_behav
(
    input i0_i,
    input i1_i,
    input sel_i,
    output out_o
);

    reg out;

    always@(i0_i or i1_i or sel_i) begin
        if (sel_i == 1'b0) begin
            out = i0_i;
        end
        else begin
            out = i1_i;
        end
    end

    assign out_o = out;

endmodule
```

# 2x1 Multiplexer (Çoklayıcı) Verilog Behavioral Modeling

```

module mux_2_1_behav
(
input i0_i,
input i1_i,
input sel_i,
output out_o
);

// reg data type must be used for assignments in a behavioral assignment block (always)
reg out;

always@(i0_i or i1_i or sel_i) begin
    if (sel_i == 1'b0) begin
        out = i0_i;
    end
    else begin
        out = i1_i;
    end
end

assign out_o = out;

endmodule

```

```

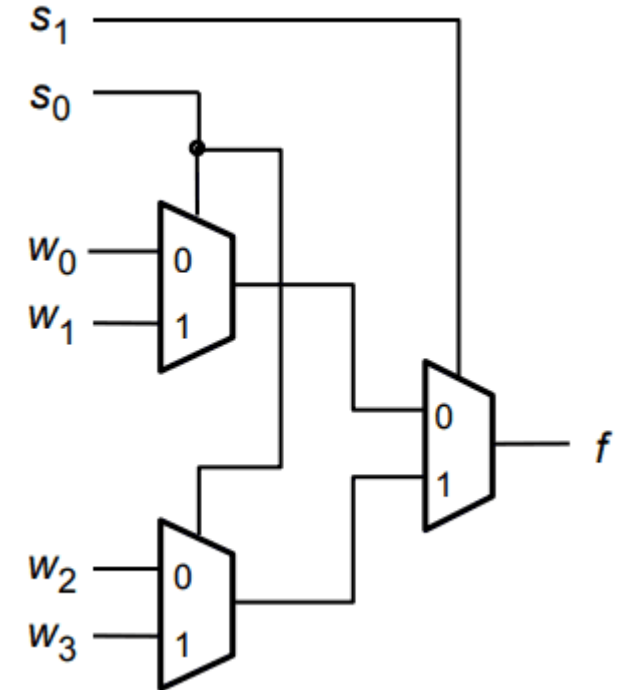
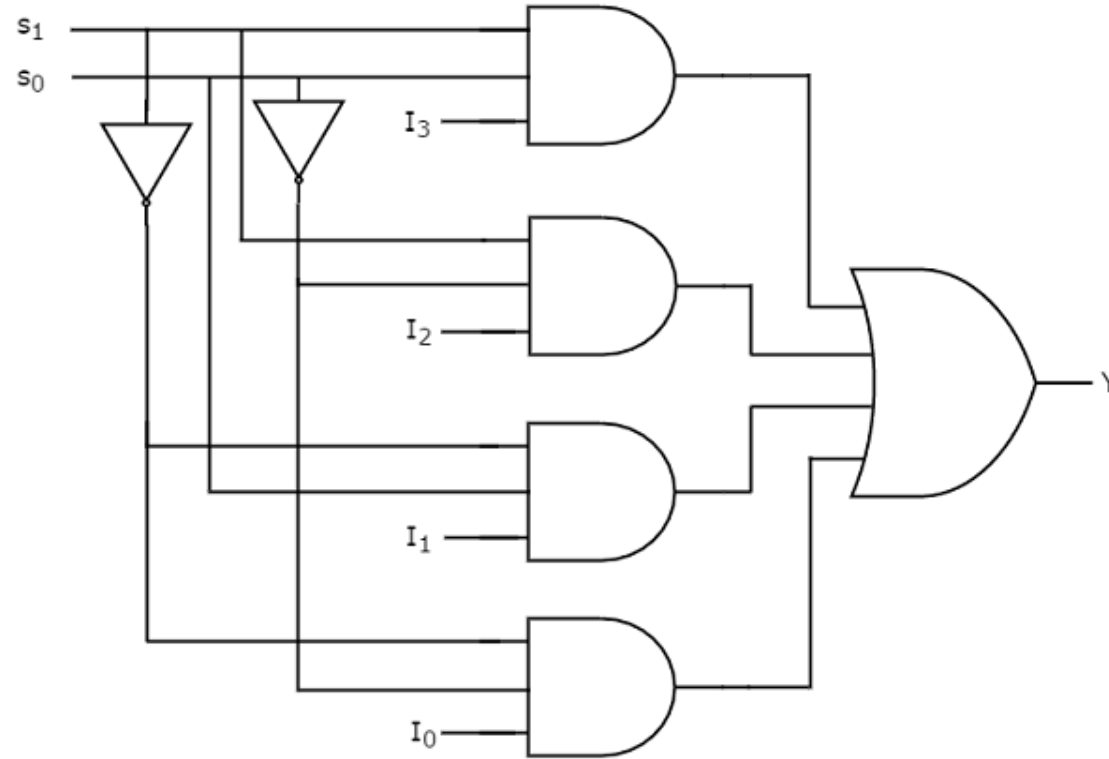
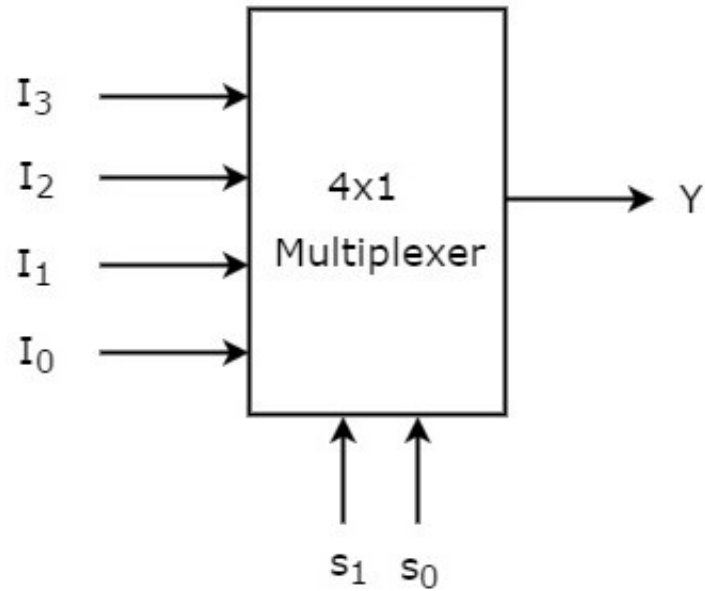
module mux_2_1_behav_reg
(
input i0_i,
input i1_i,
input sel_i,
output reg out_o
);

always@(i0_i or i1_i or sel_i) begin
    if (sel_i == 1'b0) begin
        out_o = i0_i;
    end
    else begin
        out_o = i1_i;
    end
end

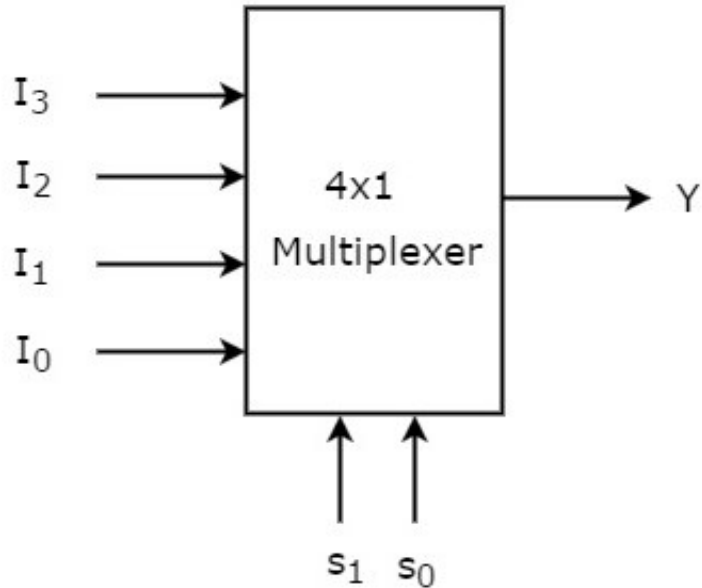
endmodule

```

# 4x1 Multiplexer (Çoklayıcı)



# 4x1 Multiplexer (Çoklayıcı) - Verilog



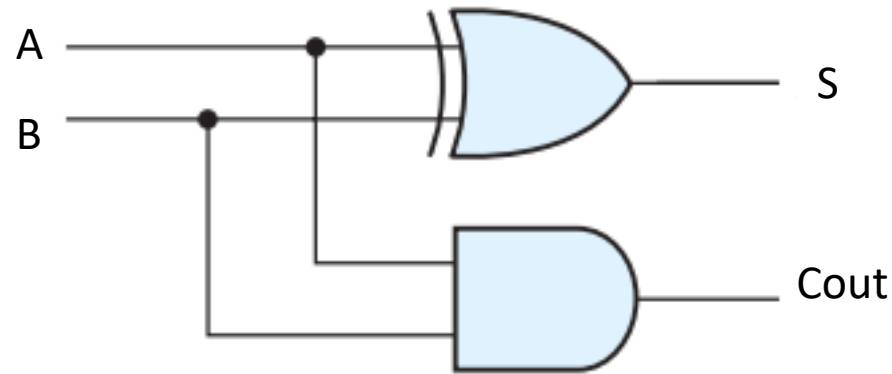
```
module mux_4_1
(
    input i0_i,
    input i1_i,
    input i2_i,
    input i3_i,
    input sel0_i,
    input sel1_i,
    output out_o
);

    assign out_o = (sel0_i == 1'b0) ?
                   (sel1_i == 1'b0) ? i0_i : i2_i :
                   (sel1_i == 1'b0) ? i1_i : i3_i;

endmodule
```

# Half Adder

Giriş		Çıkış	
A	B	S	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



$$S = A'B + AB' \rightarrow S = A \text{ xor } B$$

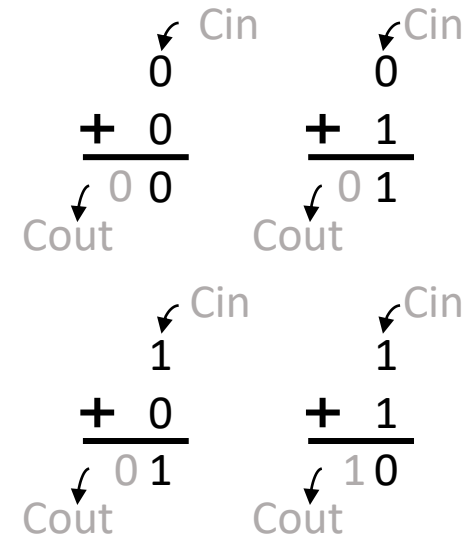
$$Cout = AB$$

```

module half_adder
(
  input a_i,
  input b_i,
  output s_o,
  output cout_o
);

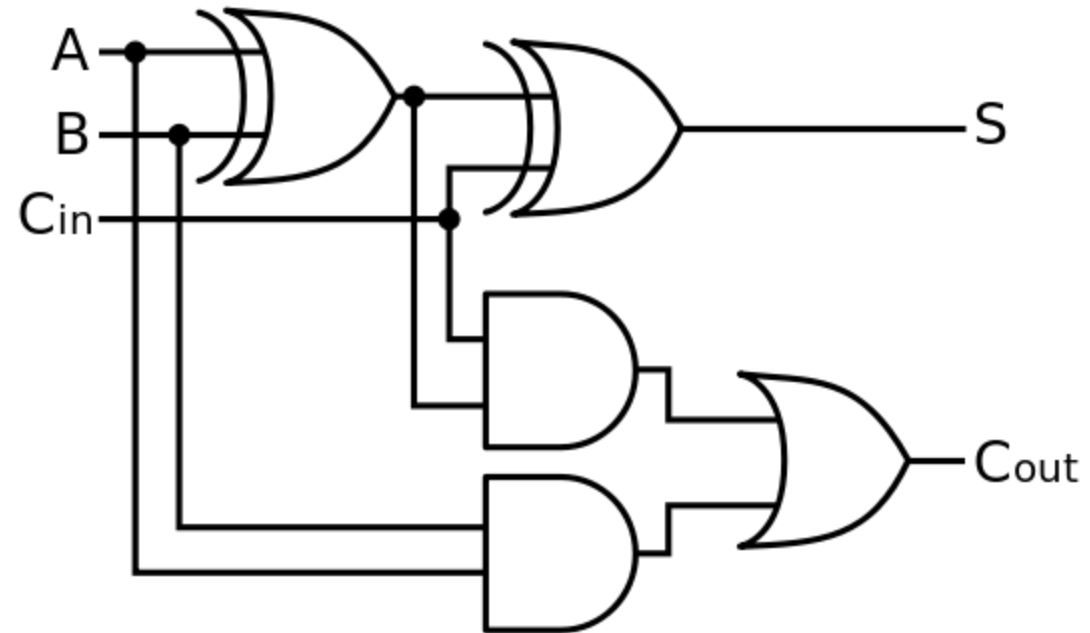
assign s_o    = a_i ^ b_i;
assign cout_o = a_i & b_i;

endmodule
  
```



# Full Adder

Giriş			Çıkış	
A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

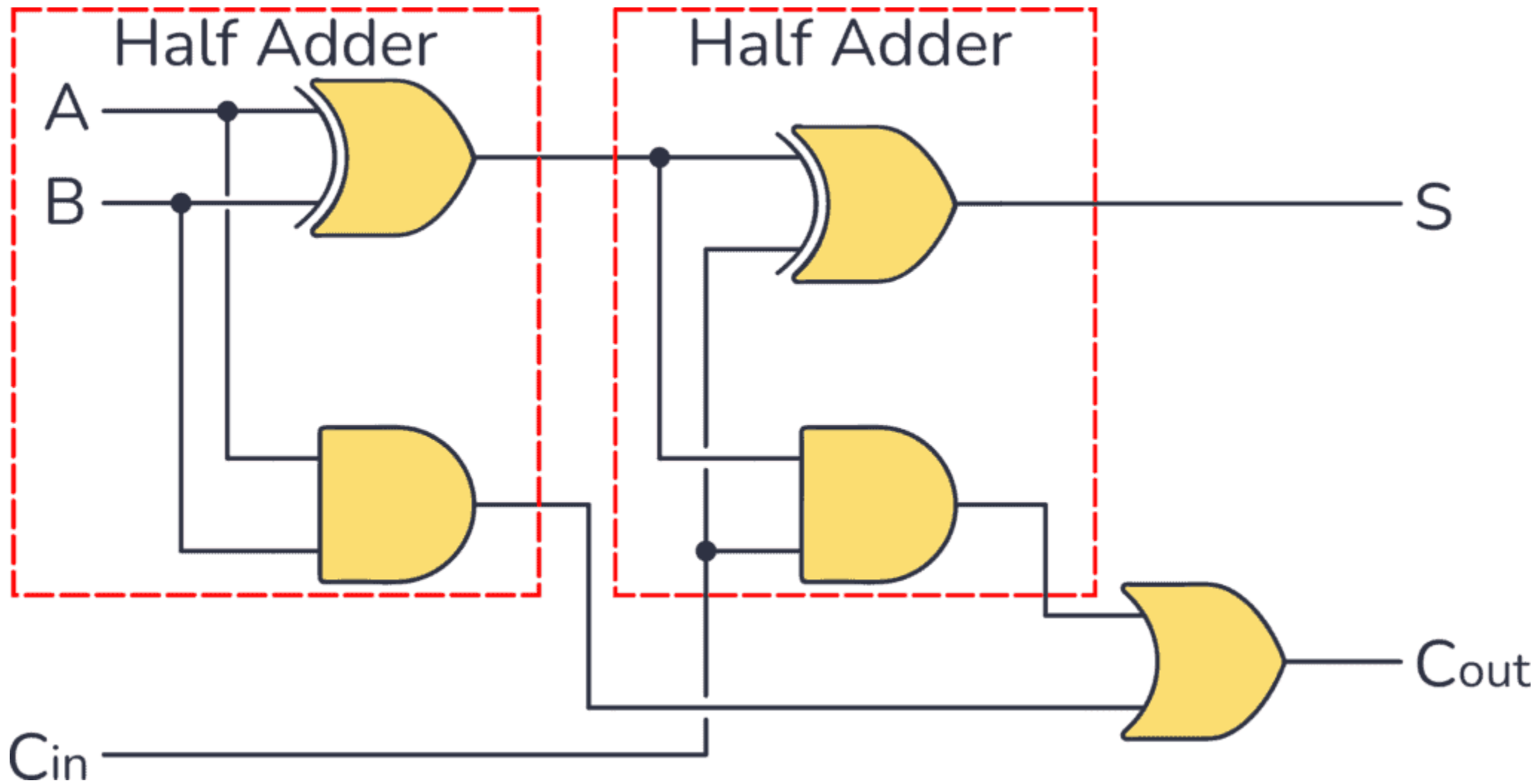


$$S = A \text{ xor } B \text{ xor } Cin$$

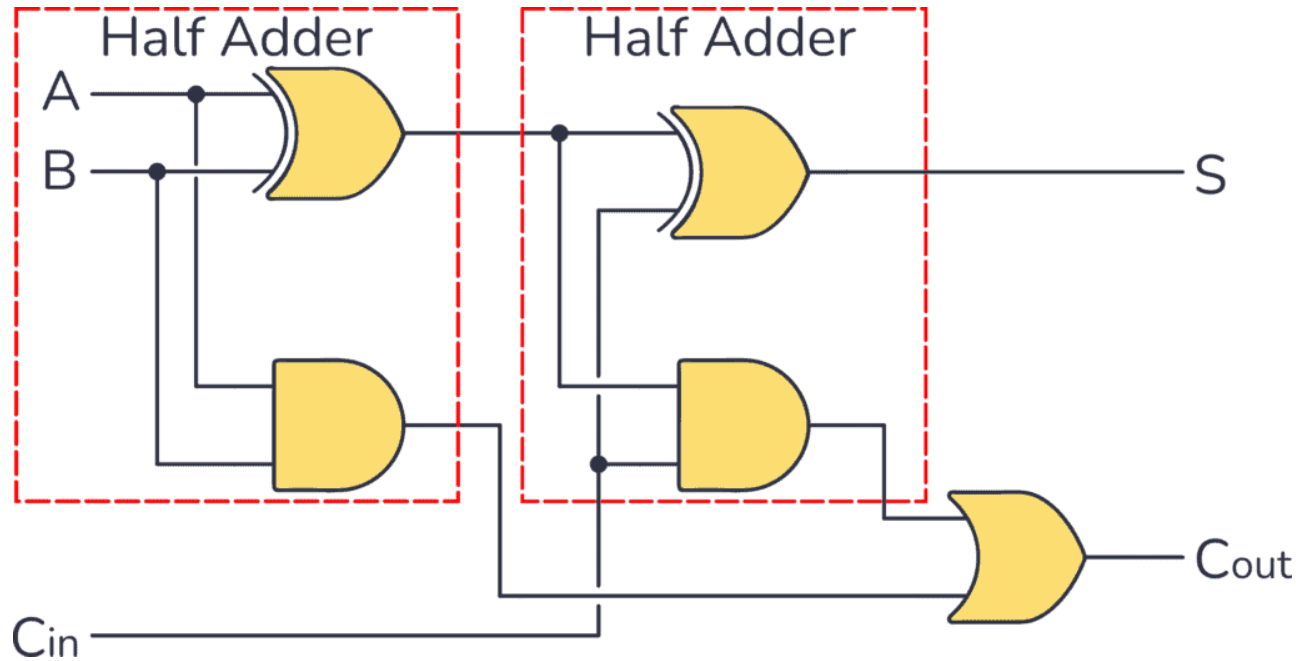
$$Cout = AB + ACin + BCin$$



# Full Adder



# Full Adder - Verilog



```
module full_adder
(
    input a_i,
    input b_i,
    input cin_i,
    output s_o,
    output cout_o
);

    assign {cout_o,s_o} = a_i + b_i + cin_i;

endmodule
```

```
module full_adder_hier
(
    input a_i,
    input b_i,
    input cin_i,
    output s_o,
    output cout_o
);

    wire ha1_s, ha1_cout, ha2_cout;

    half_adder HA1
    (
        .a_i    (a_i),
        .b_i    (b_i),
        .s_o    (ha1_s),
        .cout_o (ha1_cout)
    );

    half_adder HA2
    (
        .a_i    (ha1_s),
        .b_i    (cin_i),
        .s_o    (s_o),
        .cout_o (ha2_cout)
    );

    assign cout_o = ha2_cout | ha1_cout;

endmodule
```

# Combinational Devrelerde Gecikme (Latency) (Signal Propagation Delay)

Varsayımlar (25 derece sıcaklık için):

AND ve OR kapılarının gecikmeleri 0.3 ns

XOR kapısının gecikmesi 0.5 ns

Bağlantı (net) gecikmesleri 0.1 ns

