



**THE IMAGINATION UNIVERSITY PROGRAMME**

# **RVfpga Deney 3**

## **RISC-V Çevirici Dili**

## 1. GİRİŞ

C, Java, Python gibi yüksek-düzeyle dillerde programlamak programlamacı için verimlidir. Bu yüksek-düzeyle diller yalın yönergeler grubu olan çevirici diline çevrilir. Yer yer performansı ya da zamanlaması önemli kod bölümleri belirli zamanlamanın garantisini vermek ya da işlem süresini azaltmak için çevirici dilinde yazılır. Bu deney sana PlatformIO'yu kullanarak RVfpga'de çalıştırabileceğin bir RISC-V çevirici dili programının nasıl oluşturulacağını gösterir. RISC-V çeviricisinin kısa bir tanıtımını yapıp ardından RVfpga'de nasıl bir çevirici programı oluşturulup çalıştırılır gösteriyoruz. Ardından kendi RISC-V çevirici programlarını yazmanın uygulamasını yapman için alıştırmalar veriyoruz.

## 2. RISC-V Çevirici Dili Tanıtımı

RISC-V çevirici dili yüksek-düzeyle dilleri gerçekleştirmek için kullanılan yalın yönergeleri içerir. Örneğin, yaygın RISC-V yönergeleri iki işleneni toplayan, çıkaran ya da çarpan `add`, `sub`, `mul` yönergelerini içerir.

RISC-V yönergelerinin basit türleri: sayısal (arithmetic, logical, and shift) yönergeler, bellek işlemleri, dallandırmalar/sıçramalardır. Tablo 1'de en yaygın RISC-V yönergeleri verilmiştir. Yönergeler yazmaçlarda ya da bellekte yerleşmiş ya da sabit (diğer bir deyişle *anlık*) olarak kodlanmış işlenenleri kullanır. RISC-V 32 tane 32-bit yazmaç içerir. Tablo 2 32 RISC-V yazmacının adlarını verir. Adlarıyla (örneğin, `zero`, `s0`, `t5`, gibi gibi.) ya da yazmaç numaralarıyla (örneğin, `x0`, `x8`, `x30`) tanımlanabilirler. Programlamacılar genelde yazmacın genel amacı üzerine bilgi barındıran yazmaç adlarını kullanır. Örneğin, kaydedilmiş yazmaçlar, `s0-s11`, genelde program değişkenleri için kullanılırken, geçici yazmaçlar, `t0-t6` geçici işlemler için kullanılır. `zero` yazmacı (`x0`), bu degree programlarda çok gerek duyulduğu için, hep 0 değerini barındırır. Tablo 2'de gösterildiği gibi diğer yazmaçların da belirli kullanımları var ancak bu deneyde yalnızca `zero` yazmacı, geçiciyle kaydedilmiş yazmaçları kullanman gerekecek.

**Tablo 1. Yaygın RISC-V çevirici yönergeleri**

	RISC-V Çeviricisi	Tanım	İşlem
Sayısal	<code>add s0, s1, s2</code>	Ekle	$s0 = s1 + s2$
	<code>sub s0, s1, s2</code>	Çıkar	$s0 = s1 - s2$
	<code>addi t3, t1, -10</code>	Anlıkla ekle	$t3 = t1 - 10$
	<code>mul t0, t2, t3</code>	32-bit çarp	$t0 = t2 * t3$
	<code>div s9, t5, t6</code>	Bölüm	$t9 = t5 / t6$
	<code>rem s4, s1, s2</code>	Kalan	$s4 = s1 \% s2$
	<code>and t0, t1, t2</code>	Bitsel AND yap	$t0 = t1 \& t2$
	<code>or t0, t1, t5</code>	Bitsel OR yap	$t0 = t1   t5$
	<code>xor s3, s4, s5</code>	Bitsel XOR yap	$s3 = s4 \wedge s5$
	<code>andi t1, t2, 0xFFB</code>	Anlıkla bitsel AND yap	$t1 = t2 \& 0xFFFFFFF B$
	<code>ori t0, t1, 0x2C</code>	Anlıkla bitsel OR yap	$t0 = t1   0x2C$
	<code>xori s3, s4, 0xABC</code>	Anlıkla bitsel XOR yap	$s3 = s4 \wedge 0xFFFFFABC$
	<code>sll t0, t1, t2</code>	Sola mantıksal kaydır	$t0 = t1 \ll t2$
	<code>srl t0, t1, t5</code>	Sağa mantıksal kaydır	$t0 = t1 \gg t5$
	<code>sra s3, s4, s5</code>	Sağa aritmetik kaydır	$s3 = s4 \ggg s5$
	<code>slli t1, t2, 30</code>	Anlıkla sola mantıksal kaydır	$t1 = t2 \ll 30$
	<code>srli t0, t1, 5</code>	Anlıkla sağa mantıksal kaydır	$t0 = t1 \gg 5$
	<code>srai s3, s4, 31</code>	Anlıkla sağa aritmetik kaydır	$s3 = s4 \ggg 31$

<b>Bellek</b>	lw s7, 0x2C(t1)	Sözcüğü yükle	s7 = memory[t1+0x2C]
	lh s5, 0x5A(s3)	Yarım-sözcüğü yükle	s5 = SignExt(memory[s3+0x5A] <sub>15:0</sub> )
	lb s1, -3(t4)	Baytı yükle	s1 = SignExt(memory[t4-3] <sub>7:0</sub> )
	sw t2, 0x7C(t1)	Sözcüğü depola	memory[t1+0x7C] = t2
	sh t3, 22(s3)	Yarım-sözcüğü depola	memory[s3+22] <sub>15:0</sub> = t3 <sub>15:0</sub>
	sb t4, 5(s4)	Baytı depola	memory[s4+5] <sub>7:0</sub> = t4 <sub>7:0</sub>
<b>Dallanma</b>	beq s1, s2, L1	Eşitse dallandır	if (s1==s2), PC = L1
	bne t3, t4, Loop	Eşit değilse dallandır	if (s1!=s2), PC = Loop
	blt t4, t5, L3	Azsa dallandır	if (t4 < t5), PC = L3
	bge s8, s9, Done	Çok ya da eşitse dallandır	if (s8>=s9), PC = Done
<b>Sözde Yönergeler</b>	li s1, 0xABCDEF12	Anlığı yükle	s1 = 0xABCDEF12
	la s1, A	Adresi yükle	s1 = A değişkeninin depolandığı bellek adresi
	nop	İşlem yok	işlem yok
	mv s3, s7	Taşı	s3 = s7
	not t1, t2	Değil (Tersine Döndür)	t1 = ~t2
	neg s1, s3	Olumsuzla	s1 = -s3
	j Label	Sıçra	PC = Label etiketi
	jal L7	Sıçrayıp bağla	PC = L7; ra = PC + 4
	jr s1	Yazmaca sıçra	PC = s1

Gerçek RISC-V yönergelerinin yanı sıra RISC-V, gerçek RISC-V yönergesi olmayan ancak programcıların sıkça kullandığı, sözde yönergeler de içerir (Tablo 1'in aşağısında gösterildiği gibi). Sözde yönergeler bir ya da daha çok gerçek RISC-V yönergesi kullanılarak gerçekleştirilir. Örneğin, taşıma sözde yönergesi (mv s1, s2) s2'nin içeriğini kopyalayıp s1'e koyar. Şu gerçek RISC-V yönergesiyle gerçekleştirilir: addi s1, s2, 0.

**Tablo 2. RISC-V yazmaçları**

Ad	Yazmaç Numarası	Kullanım
zero	x0	Sabit değer 0
ra	x1	Dönüş adresi
sp	x2	Yığın göstergesi
gp	x3	Genel gösterge
tp	x4	İş parçacığı göstergesi
t0-2	x5-7	Geçici değişkenler
s0/fp	x8	Kaydedilmiş değişken / Çerçeve göstergesi
s1	x9	Kaydedilmiş değişken
a0-1	x10-11	İşlev argümanları / Dönüş değerleri
a2-7	x12-17	İşlev argümanları
s2-11	x18-27	Kaydedilmiş değişkenler
t3-6	x28-31	Geçici değişkenler

Nokta ile başlayan komutlar çevirici program yönlendirmeleridir. Çevirici programca çevirilecek koddan çok çevirici programa komutlardır. Çevirici programa kodla verinin nereye konulacağını, programda kullanılmak üzere metin ile veri sabitlerini, gibi bilgileri söylerler. Tablo 3 RISC-V çevirici programının ana yönlendirmelerini gösterir (*The RISC-V Reader: An Open Architecture Atlas*, Patterson & Waterman, © 2017).

**Tablo 3. RISC-V ana yönlendirmeleri**

Yönlendirme	Tanım
<b>.text</b>	Arkasından gelen öğeler <code>text</code> bölümünde depolanır (makine kodu).
<b>.data</b>	Arkasından gelen öğeler <code>data</code> bölümünde depolanır (genel değişkenler).
<b>.bss</b>	Arkasından gelen öğeler <code>bss</code> bölümünde depolanır (0'la ilk değeri verilen genel değişkenler).
<b>.section .foo</b>	Arkasından gelen öğeler <code>.foo</code> adlı bölümde depolanır.
<b>.align n</b>	Sonraki veriyi $2^n$ -bayt sınırına hizalar. Örneğin, <code>.align 2</code> sonraki değeri sözcük sınırına hizalar.
<b>.balign n</b>	Sonraki veriyi n-bayt sınırına hizalar. Örneğin, <code>.balign 4</code> sonraki değeri sözcük sınırına hizalar.
<b>.globl sym</b>	<code>sym</code> etiketinin genel olup diğer dosyalardan referanslanabileceğini bildirir.
<b>.string "str"</b>	<code>str</code> karakter dizisini bellekte depolayıp boşla-sonlandır.
<b>.word w1,...,wn</b>	n 32-bit niceliği ardışık bellek sözcüklerinde depolar.
<b>.byte b1,...,bn</b>	n 8-bit niceliği belleğin ardışık baytlarında depolar.
<b>.space</b>	İlk değerleri verilmemiş değişken depolamak için bellek boşluğu ayırır. Yaygın olarak, girdi olarak da görev yapmadıklarında, çıktı değişkenlerini bildirmek için kullanılır. İstediğimiz boşluk hep bytelerin bir sayısı olarak ifade edilmelidir. Örneğin, <code>RES : .space 4</code> yönlendirmesi ilk değeri verilmemiş dört bayt (bir diğer deyişle bir sözcük) ayırır.
<b>.equ name, constant</b>	<code>name</code> sembolünü <code>constant</code> değeriyle tanımla. Örneğin, <code>.equ N, 12</code> , sembol N'yi 12 değeriyle tanımlar.
<b>.end</b>	<code>.end</code> yönlendirmesine ulaştığında çevirici program işini sonuçlandırır. Bu yönlendirmeden sonraki metinler yok sayılacaktır.

Aşağıdaki örnekler (Tablo 4 - Tablo 5'e göz at) RISC-V çeviricisindeki yaygın olarak kullanılan yüksek-düzeyle yapılan nasıl kodlanacağını gösterir. Önemli olarak dallandırma yönergeleri (`bge`, `bne`, `blt`, and `bge`) koşullu olarak bir etikete sıçrarken; sıçrama yönergesi (`j`) koşulsuz bir etikete sıçrar. Bir-satırlı yorumlar C'de `//` ile gösterilir, RISC-V çeviricisinde `#` ile gösterilir.

İlk örnekte (bir if/else deyimini gerçekleştirme, Tablo 4'e göz at), önemli olarak C koduyla RISC-V çevirici kodu karşıt durumları denetler: C kodu azdırı denetler (`<`) çevirici eşdeğeri çok ya da eşitseyi denetler (`>=`).

**Tablo 4. RISC-V Çevirici Örneği 1: if/else deyimi**

// C Kodu	# RISC-V Çeviricisi
<code>int a, b, c;</code>	<code># s0 = a, s1 = b, s2 = c</code>
<code>if (a &lt; b)</code>	<code>bge s0, s1, L1 # if (a &gt;= b) goto L1</code>
<code>    c = 5;</code>	<code>addi s2, zero, 5 # c = 5</code>
<code>else</code>	<code>j L2 # else bloğunu atla</code>
<code>    c = a + b;</code>	<code>L1: add s2, s0, s1 # c = a + b</code>
	<code>L2:</code>

İkinci örnekte (bir tam sayı dizisini işleme, Tablo 5'e göz at), RISC-V çevirici kodu geçici değişkenleri tutmak için geçici yazmaçları (t0-t3) kullanır, sabit 100 ile veri dizisinin taban adresi gibi. İlk üç yönergede yazmaçlara ilk değerlerini verdikten sonra RISC-V çevirici kodu  $i \geq 100$ 'ü bge (çok ya da eşitse dallandır) yönergesi ile denetler; yine C kodunun karşılığı. Eğer koşul sağlanmışsa for döngüsü biter. Eğer dala **geçilmemişse**,  $i$  100'den azdır dolayısıyla kalan kod yürütülür. Önemli olarak tam sayılar (32-bit ikiye tümler sayılar) bellekte 4 bayt kapladığından dolayı taban adrese eklenmeden önce  $i$  4'le çarpılır (slli t2, s0, 2 yönergesini kullanarak. RISC-V'da bellek baytla-adreslenebilir (bir diğer deyişle bir bayt başına bir adres vardır). Dizi bir karakterler dizisi olsaydı (i.e., char data[100];), dizideki öğeler yalnızca birer bayt kaplar,  $i$  sayısı  $i$  dizi dizini (array[i]) oluşturmak için direkt olarak taban adresine eklenebilirdi. Dizin ögesi okunup, on azaltılıp, yazıldıktan sonra (lw, addi, sw yönergeleriyle), dizi dizini  $i$  (s0) artırılıp program for döngüsünün başlangıcına sıçar (j L5 yönergesini kullanarak).

**Tablo 5. RISC-V Çevirici Örneği 2: tam sayı dizisini işleme**

// C Kodu	# RISC-V Çeviricisi
int i;	# s0 = i, t1 = verinin taban adresi
int data[100];	# (0x300'da olduğu varsayılır)
	addi s0, zero, 0 # i = 0
	addi t0, zero, 100 # t0 = 100
	li t1, 0x300 # dizinin taban adresi
for (i=0; i<100; i++)	L5: bge s0, t0, L7 # if (i>=100) döngüden çık
	slli t2, s0, 2 # t2 = i*4
	add t2, t1, t2 # data[i]'nin adresi
	lw t3, 0(t2) # t3 = array[i]
array[i] = array[i]-10;	addi t3, t3, -10 # t3 = array[i]-10
	sw t3, 0(t2) # array[i] = array[i]-10
	addi s0, s0, 1 # i++
	j L5 # döngü
	L7:

RISC-V çevirici dili üzerine daha çok detay için RISC-V Yönerge Kümesi El Kitabına (şuradan erişilebilir: <https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf>) ya da *Digital Design and Computer Architecture*, Harris & Harris, Elsevier, © 2021 (2021 yazında yayınlanacak) gibi bir ders kitabına ya da *The RISC-V Reader: An Open Architecture Atlas*, Patterson & Waterman, © 2017 bak.

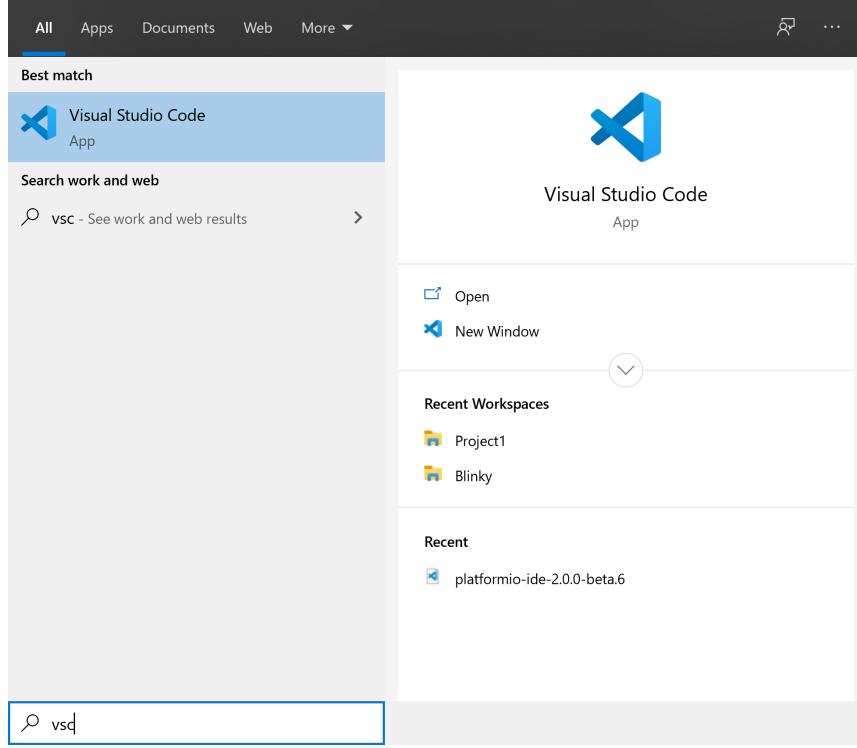
### 3. RVfpga için bir RISC-V Çevirici Programı Yazma

Şimdi RISC-V çevirici programları yazmayı kendin keşfedip alıştırma olarak uygulayabilirsin. Kendi programlarını yazmadan önce bir PlatformIO projesi kurup RVfpga'de çevirici programları oluşturup çalıştırmak için şu adımları izle:

1. Bir RVfpga projesi oluştur
2. RISC-V çevirici dili programı yaz
3. RVfpga'i Nexys A7 FPGA kartına indir
4. C programını derle, indir, çalıştır

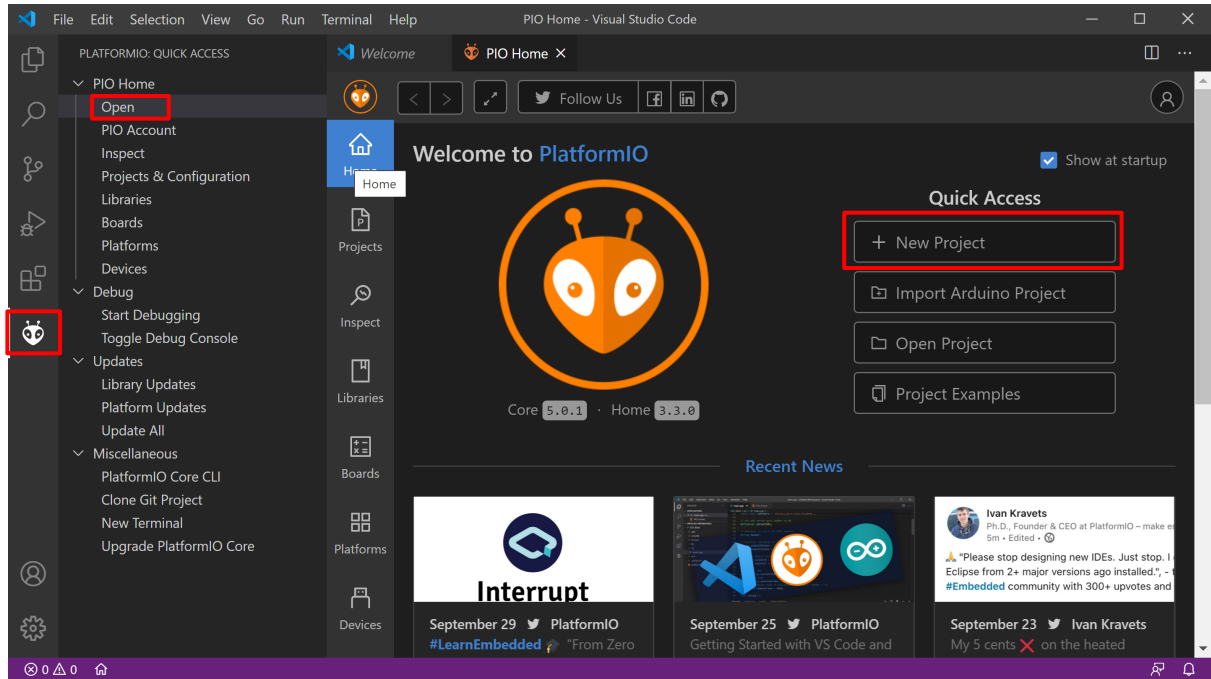
#### Adım 1. Bir RVfpga projesi oluştur

RVfpga Deney 2'nin Adım 1'ini izle – kolaylık olması için burada da yazılı. Start (Başlat) butonuna tıklayıp, VSCode yazıp ardından Visual Studio Code'a tıklayıp VSCode'u aç (Figür 1'e göz at).



**Figür 1. VSCode'u aç**

VSCode'u açtığında PlatformIO kendiliğinden açılmıyorsa sol menü şeridindeki PlatformIO ikonuna tıklayıp ardından PIO Home → Open (Aç) üzerine tıkla (Figür 2'ye göz at).

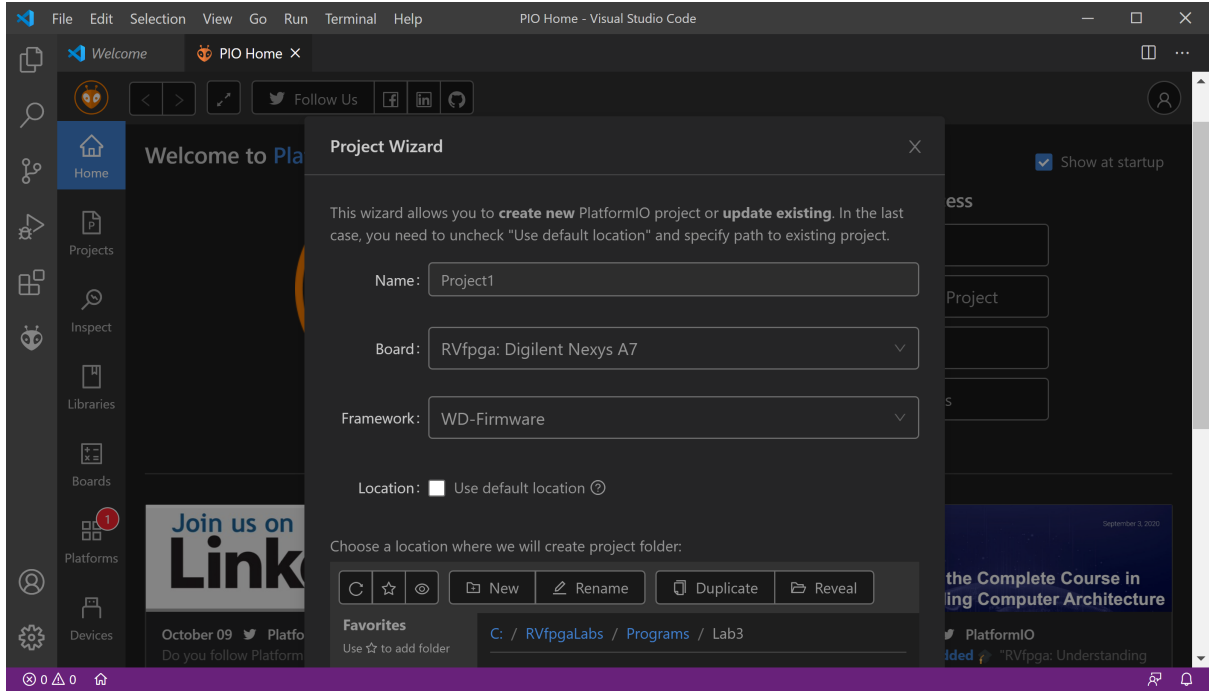


**Figür 2. PlatformIO'yu açıp yeni proje oluştur**

Şimdi, PIO Home hoşgeldin penceresinde New Project (Yeni Proje) üzerine tıkla (Figür 2'ye göz at).

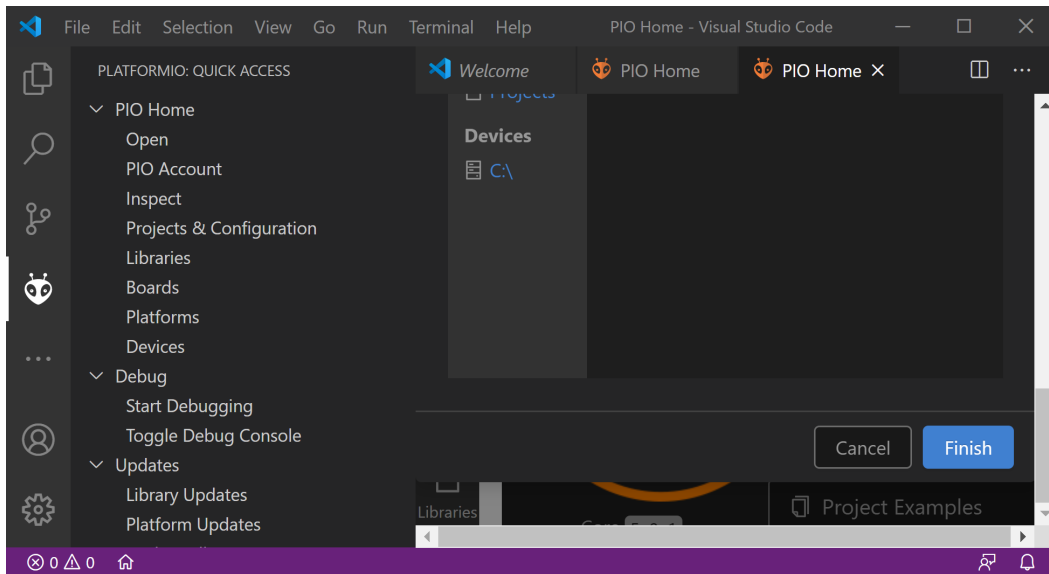
Figür 3'te gösterildiği gibi projeyi Project1 diye adlandırıp kartı RVfpga: Digilent Nexys A7 (RVfpga yazmaya başlarsan kart gözükecektir) olarak seç. Varsayılan çerçeveyi WD-framework (Western Digital framework –Freedom-E SDK gcc ile gdb içerir). Use default location (varsayılan konumu kullan) işaretini kaldırıp programını şuraya yerleştir:

[RVfpgaPath] /RVfpga/Labs/Lab3



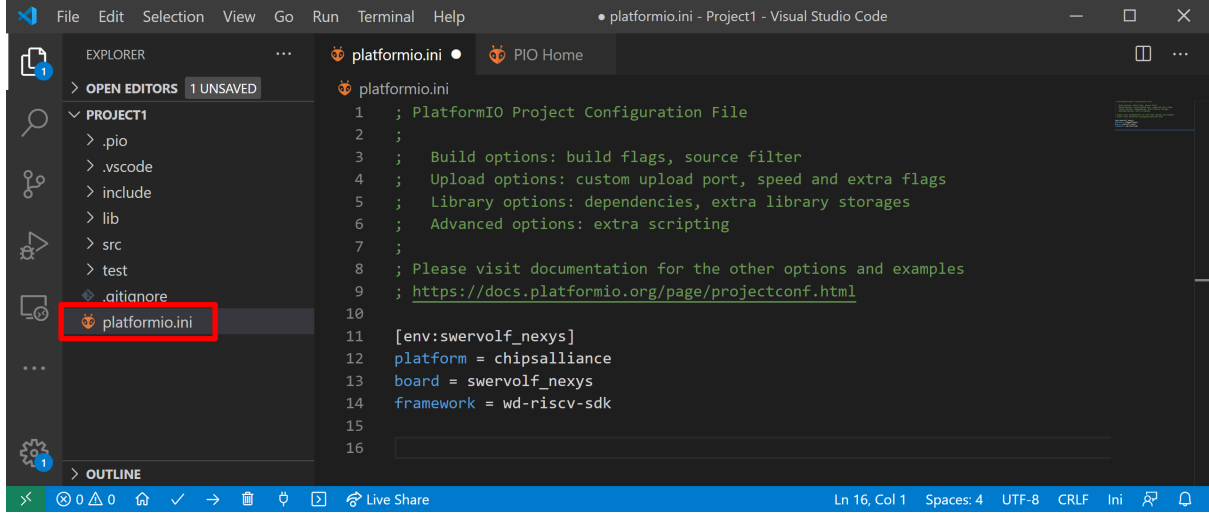
**Figür 3. Projeyi adlandırıp, kart ile proje klasörünü seç**

Pencerenin aşağısında Finish'e (Bitir) tıkla (Figür 4'e göz at).



**Figür 4. Proje oluşturmaya bitir**

Soldaki Explorer (Gezgin) panelinde, PROJECT1 altında (bunu genişletmen gerekebilir), platformio.ini'yi açmak için üzerine çift-tıkla (Figür 5'e göz at). Bu PlatformIO ilk değerlendirme dosyasıdır.

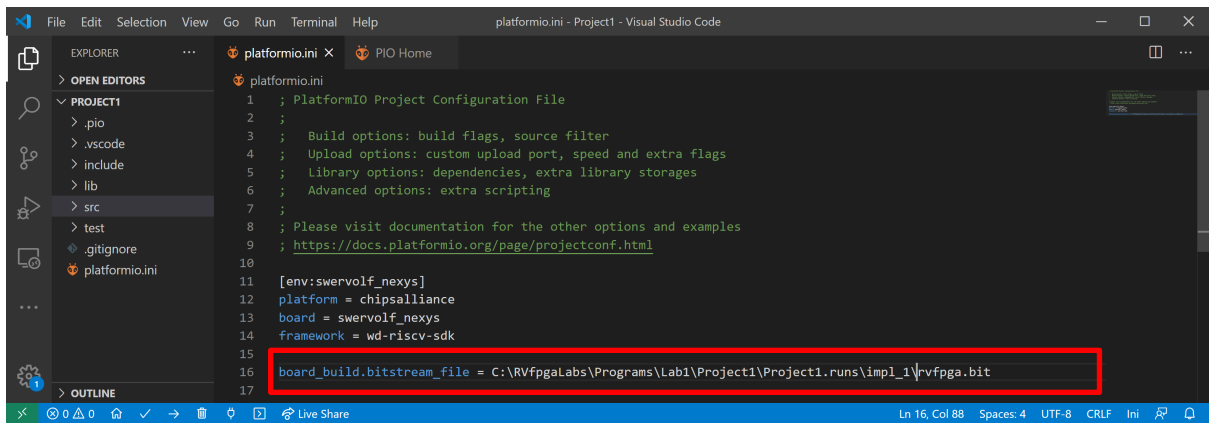


**Figür 5. PlatformIO ilk değerlendirme dosyası: platformio.ini**

Şu satırı platformio.ini dosyasına ekle, Figür 6'da gösterildiği gibi:

```
board_build.bitstream_file =  
[RVfpgaPath]/RVfpga/Labs/Lab1/Project1/Project1.runs/impl_1/rvfpga.bit
```

Bu satır FPGA'e yüklemek için PlatformIO'nun veri akışı dosyasını nerede bulması gerektiğini gösterir. Yukarıdaki yol Deney 1'de oluşturduğunuz veri akışının yeri. (Eğer Deney 1'i bitirmediyse, İlk Kullanım Kılavuzu ile dağıtılan şuradaki RVfpga veri akışını kullanabilirsiniz: [RVfpgaPath]/RVfpga/src/rvfpga.bit.) platformio.ini dosyasını kaydetmek için Ctrl-s'e bas.



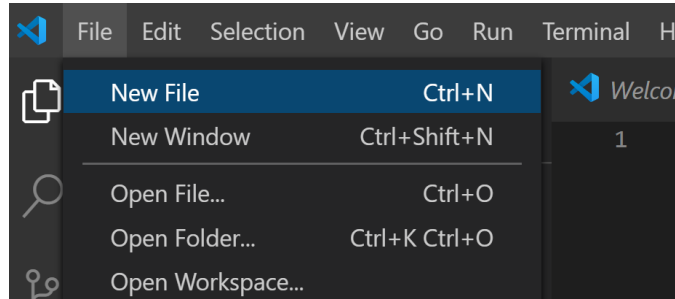
**Figür 6. RVfpga veri akışı (bitstream) dosyasının yerini ekle (rvfpga.bit)**

İlk Kullanım Kılavuzunda kullanılan örneklerde daha bütün bir *platformio.ini* dosyasının kullanıldığını unutma. Ek komutlar gerektiren bir işlevsellik kullanmak istersen (Verilator simütörüne giden yol, dizisel konsolun yapılandırılması, whisper ayıklama aracı, gibi gibi.), o örneklerdeki *platformio.ini*'yi kullanabilirsin.



## Adım 2. Bir RISC-V çevirici dili programı yaz

Şimdi bir RISC-V çevirici programı yazacaksın. File (Dosya) → New File (Yeni Dosya) üzerine tıkla (Figür 7'ye göz at).



**Figür 7. Projeye dosya ekle**

Boş bir pencere açılacaktır. Şu RISC-V çevirici programını o pencereye yaz (ya da kopyala/yapıştır) (Figür 8'e göz at). Bu programa şuradan da erişilebilir:

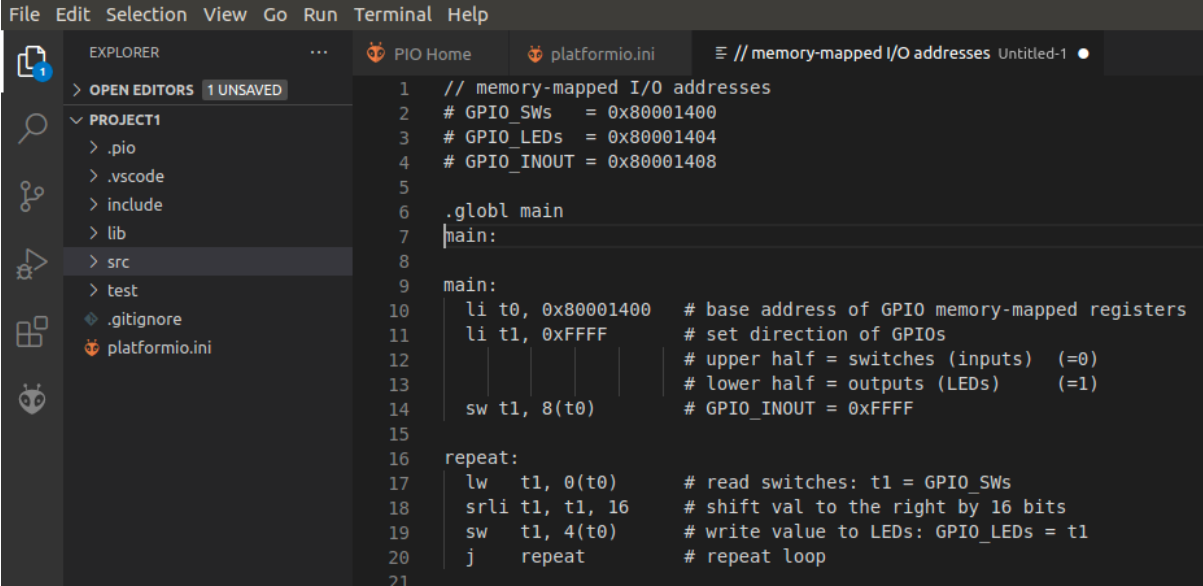
```
[RVfpgaPath]/RVfpga/Labs/Lab3/ReadSwitches.S

// memory-mapped I/O addresses
# GPIO_SWs    = 0x80001400
# GPIO_LEDs   = 0x80001404
# GPIO_INOUT  = 0x80001408

.globl main
main:

main:
    li t0, 0x80001400    # base address of GPIO memory-mapped registers
    li t1, 0xFFFF       # set direction of GPIOs
                        # upper half = switches (inputs)    (=0)
                        # lower half = outputs (LEDs)       (=1)
    sw t1, 8(t0)         # GPIO_INOUT = 0xFFFF

repeat:
    lw  t1, 0(t0)        # read switches: t1 = GPIO_SWs
    srli t1, t1, 16       # shift val to the right by 16 bits
    sw  t1, 4(t0)        # write value to LEDs: GPIO_LEDs = t1
    j   repeat           # repeat loop
```



```

1 // memory-mapped I/O addresses
2 # GPIO_SWs = 0x80001400
3 # GPIO_LEDS = 0x80001404
4 # GPIO_INOUT = 0x80001408
5
6 .globl main
7 main:
8
9 main:
10     li t0, 0x80001400 # base address of GPIO memory-mapped registers
11     li t1, 0xFFFF    # set direction of GPIOs
12                     # upper half = switches (inputs) (=0)
13                     # lower half = outputs (LEDs) (=1)
14     sw t1, 8(t0)      # GPIO_INOUT = 0xFFFF
15
16 repeat:
17     lw t1, 0(t0)      # read switches: t1 = GPIO_SWs
18     srli t1, t1, 16    # shift val to the right by 16 bits
19     sw t1, 4(t0)      # write value to LEDs: GPIO_LEDS = t1
20     j repeat          # repeat loop
21

```

**Figür 8. RISC-V çevirici programını gir**

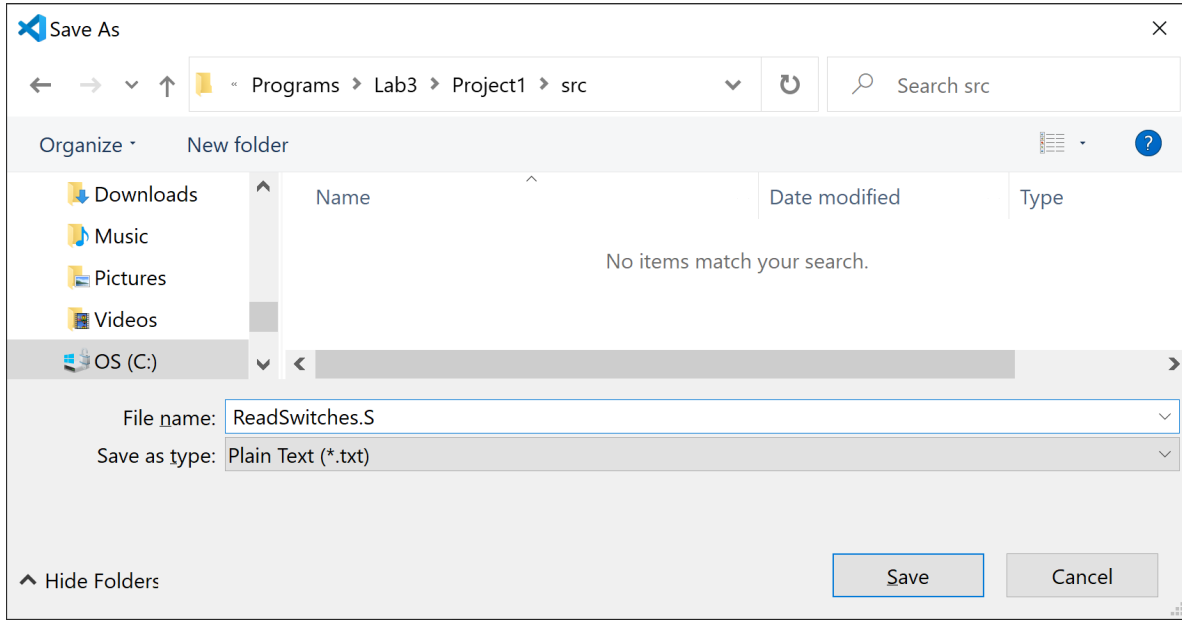
Çevirici kodu, kod başlangıcında şu satırları barındırmalıdır:

```
.globl main
main:
```

.globl çevirici yönlendirmesi etiketi bütün dosyalarda görünür kılar. Önyükleme kodu (~/.platformio/packages/framework-wd-riscv-sdk/board/nexys\_a7\_ah1/startup.S) sistemi yapılandırıp bu etikete sıçrayacaktır (*main*). Ayıklayıcı başladığında oraya geçici bir kesme noktası ayarlayacaktır.

Bu RISC-V çevirici programı Deney 2'deki örnekle aynıdır ancak RISC-V çeviricisiyle yazılmıştır. Genel-amaçlı I/O'nun (GPIO) yönünü ayarlayıp, anahtarların değerlerini durmaksızın okuyup LEDlere yazar..

Programı panele girdikten sonra dosyayı kaydetmek için Ctrl-s'e bas. ReadSwitches.S olarak adlandırıp Project1 dizininin src klasörüne kaydet (Figür 9'a göz at).




**Figür 9. Dosyayı ReadSwitches.S olarak kaydet**

### Adım 3. RVfpga'i Nexys A7 FPGA kartına indir

Şimdi RVfpga'i Nexys A7 FPGA kartına indireceksin. RVfpga'i indirme için yönergeleri GSG ile Deney 2'de tanımlandığı gibi izle – burada kolaylık olması için tekrarlanmıştır.

Sol menü şeridindeki PlatformIO ikonuna  tıklayarak RVfpga'i Nexys A7 kartına indir, ardından Project Tasks → env:swervolf\_nexys → Platform genişletip Upload Bitstream tıkla.

Alternatif olarak **Error! Reference source not found.**'de gösterildiği gibi RVfpga'i bir PlatformIO terminal pencersinden de indirebilirsin. PlatformIO penceresinin aşağısındaki

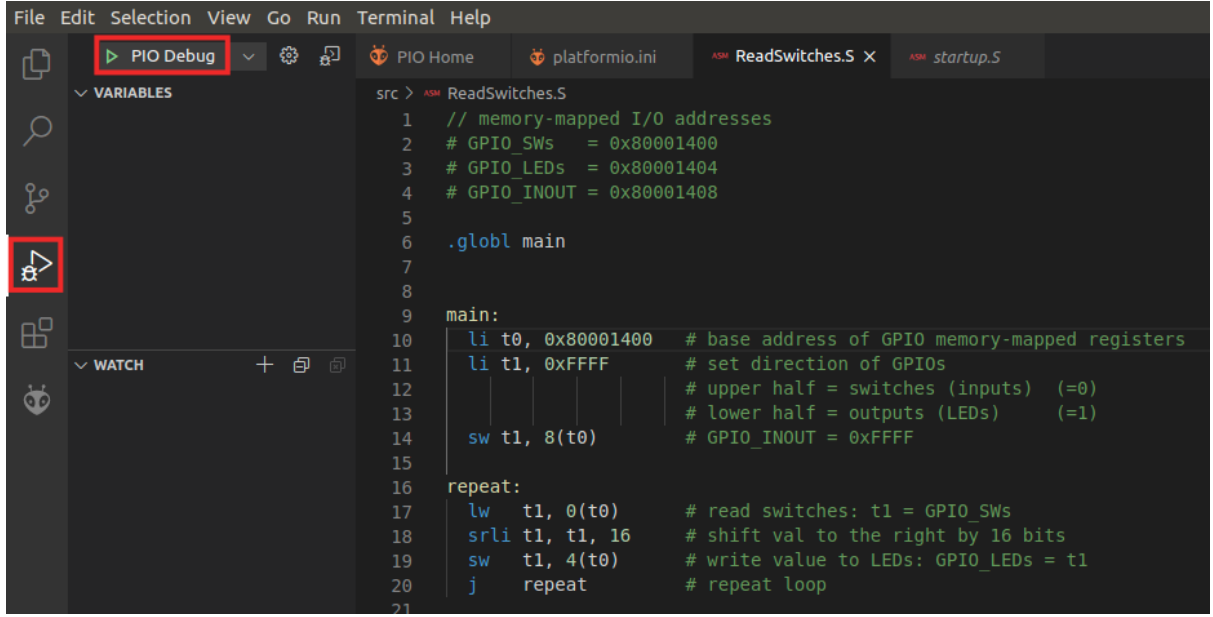
PlatformIO: New Terminal (PlatformIO: Yeni Terminal) butonuna  tıklayıp, ardından şunu PlatformIO terminaline yaz (ya da kopyala):

```
pio run -t program_fpga
```

### Adım 4. RISC-V çevirici programını derle, indir, çalıştır

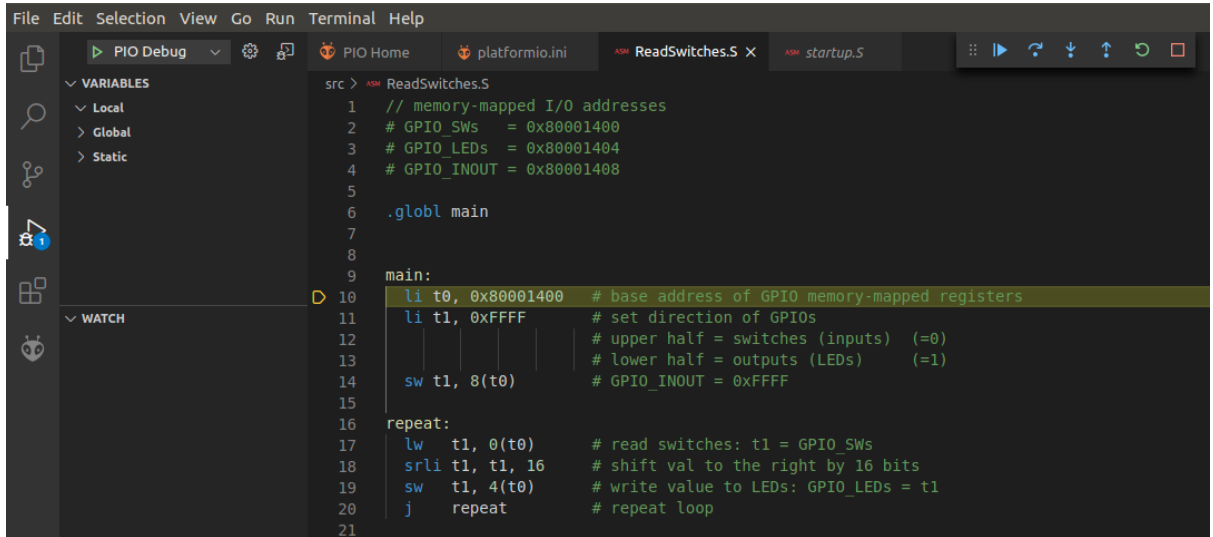
RVfpga kartta çalıştığına göre programını derleyip, RVfpga'e indirip, çalıştırıp/ayıklayacaksın. VSCode açıkta değilse aç. Son projen, Project1, kendiliğinden açılmalı. Açılmazsa PlatformIO eklentisinin açık olduğunun sağlamasını yapıp File (Dosya) → Open Folder (Klasörü Aç) üzerine tıklayıp bu deneyde oluşturduğun Project1'i seç (ancak açma).

Sol menü şeridindeki Run (Çalıştır) butonuna tıklayıp ardından Start Debugging (Ayıklamayı Başlat) butonuna tıkla (Figür 10'a göz at).



**Figür 10. Programı RVfpga'de çalıştır**

Program Nexys A7 kartındaki FPGA'de çalışan RVfpga'ye inecektir. Şimdi programı çalıştırıp ayıklamaya başlayabilirsin (Figür 11'e göz at).



**Figür 11. RVfpga'de program çalışıyor**

RVfpga İlk Kullanım Kılavuzu ile Deney 2'de tanımlandığı gibi ayıklama araç çubuğu ile Ayıklayıcı seçeneklerini programı çalıştırıp yönetmek için kullan. Örneğin, satır 17'ye bir kesme noktası ayarlayıp (satır numarasının soluna tıklayarak) ardından yazmaç t1'a anahtarların değerleri ona yüklendiği için bak. Stop (Durdur) butonuna



(ya da Shift - F5) basarak ayıklama oturumunu durdurduğunda ayıklama oturumu sona ermesine karşın program RVfpga'de çalışmayı sürdürür.

## 4. Alıştırmalar

Şimdi Deney 2'deki alıştırmalarda C yerine RISC-V çeviricisi kullanarak kendi RISC-V çevirici programlarını oluştur. Kolaylık olması için alıştırmaya tanımları aşağıda tekrarlanmıştır.

Unutma, RVfpga'i programlar arası karta yeniden yüklemeye gerek duymamak için Nexys A7 kartını bilgisayara bağlı bırakabilirsin. Ancak Nexys A7 kartını kapatırsan PlatformIO kullanarak RVfpga'i karta yeniden yüklemen gerekir.

**Alıştırma 1.** LEDlerin değerini anahtarlara yakıp söndüren bir RISC-V çeviricisi programı yaz. Darbe bir kişinin yanıp sönmeyi anlayabileceği yavaşlıkta olmalı. Programı **FlashSwitchesToLEDs.S** olarak adlandır.

**Alıştırma 2.** Anahtarların değerinin tersini LEDlerde gösteren bir RISC-V çeviricisi programı yaz. Örneğin anahtarlar şu ise (ikili sistemde): 0101010101010101, LEDler şunu göstermeli: 1010101010101010; anahtarlar şu ise: 1111000011110000, LEDler şunu göstermeli: 0000111100001111; gibi gibi. Programı **DisplayInverse.S** olarak adlandır.

**Alıştırma 3.** Bütün LEDler yakılana değin artan, kayan sayılar gösteren bir RISC-V çeviricisi programı yaz. Ardından örüntü kendini tekrarlamalı. Programı **ScrollLEDs.S** olarak adlandır.

Program şunları yapmalı:

1. İlk olarak bir yakılmış LED sağdan sola kayıp ardından soldan sağa kaymalı.
2. Ardından iki yakılmış LED sağdan sola kayıp ardından soldan sağa doğru kaymalı.
3. Ardından üç yakılmış LED sağdan sola kayıp ardından soldan sağa doğru kaymalı.
4. Gibi gibi, bütün LEDler yakılmış olana değin.
5. Ardından bu örüntü kendini tekrarlamalı.

**Alıştırma 4.** Anahtarların 4 düşük önemli bitiyle 4 yüksek önemli bitini pozitif 4-bit eklemeye gösteren bir RISC-V çeviricisi programı yaz. Sonucu LEDlerin 4 düşük önemli (en sağ) bitinde göster. Programı **4bitAdd.S** olarak adlandır. LEDlerin beşinci biti pozitif taşması (unsigned overflow) olunca yanmalı (yani taşınan (carry) 1 olunca).

**Alıştırma 5.** Öklit algoritmasıyla iki sayının, a ile b, en büyük ortak bölenini bulan bir RISC-V çeviricisi programı yaz. a ile b'nin değerleri programda statik tanımlı değişken olmalıdır.

Programı **GCD.S** olarak adlandır. Şurada Öklit algoritmasıyla ilgili ek bilgi vardır:

<https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm>. Google'da şunu da aratabilirsin: "Euclidean algorithm".

**Alıştırma 6.** Fibonacci dizisinin ilk 12 sayısını hesaplayıp, ardından bunu sonlu, 12 ögeli bir vektörde (bir diğer deyişle) depolayan bir RISC-V çeviricisi programı yaz. Fibonacci sayılarının sonsuz sekansı şöyle tanımlanır:

$$V(0)=0, V(1)=1, V(i)=V(i-1)+V(i-2) \quad (i=0,1,2,\dots)$$

Bir diğer deyişle i ögesinde denk gelen Fibonacci sayısı, dizinin önceki iki Fibonacci sayısının toplamına eşittir. **Error! Reference source not found.** i = 0'dan 8'e Fibonacci sayılarını gösterir.

**Tablo 6. Fibonacci dizisi**

i	0	1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---	---	---

<b>V</b>	0	1	1	2	3	5	8	13	21
----------	---	---	---	---	---	---	---	----	----

Vektör boyutu N programda bir sabit olarak tanımlanmalıdır. Programı **Fibonacci.S** olarak adlandır.

**Alıştırma 7.** N ögeli bir A vektöründen (dizisinden) yalnızca çift, sıfırdan büyük sayıları içeren bir başka B vektörü oluştur. Örneğin:  $N=12$ ,  $A = [0,1,2,7,-8,4,5,12,11,-2,6,3]$ , ise:  $B = [2,4,12,6]$ . Programı **EvenPositiveNumbers.S** olarak adlandır.

**Alıştırma 8.** İki N ögeli vektör (dizi) A ile B verilince şöyle bir C vektörü oluştur:

$$C(i) = |A[i] + B[N-i-1]|, i = 0, \dots, N-1.$$

Yeni vektörü hesaplayan programı RISC-V çeviricisinde yaz. Programında 12 ögeli diziler kullan. Programı **AddVectors.S** olarak adlandır.

**Alıştırma 9.** RISC-V çeviricisinde bubble sort algoritmasını gerçekleştir. Bu algoritma şu prosedürle bir vektörün bileşenlerini yükselen biçimde sıralar:

1. Vektörü bitene değin tekrar tekrar dolaş.
2.  $V(i) > V(i+1)$  ise komşu bileşenlere yer değiştir.
3. Ardışık bileşenlerin hepsi sıralı olunca algoritma sonlanır.

Programını denemek için 12 ögeli diziler kullan. Programı **BubbleSort.S** olarak adlandır.

**Alıştırma 10.** Negatif olmayan bir sayı n'in faktöriyelini yinelemeli çarpımlarla hesaplayan bir RISC-V çeviricisi programı yaz. Programı değişik n değerleriyle dene ancak son değer n=7 olmalıdır. n program içerisinde statik tanımlanmış bir değişken olmalı. Programı **Factorial.S** olarak adlandır.