



**THE IMAGINATION UNIVERSITY PROGRAMME**

# **RVfpga Deney 4**

## **İşlev Çağrılar**

## 1. GİRİŞ

Modülerlik ile kodun yeniden kullanımını sağlayıp kod yazmayla ayıklamayı kolaylaştırdıklarından dolayı işlev çağrılarını bütün programların önemli bir parçasıdır. C programlama dili rastgele sayı oluşturucu ile yaygın matematik işlevleri gibi yaygın kullanılan C işlevlerinin standart kütüphanesini, işlemci/kart özelinde kütüphanelerin yanı sıra, içerir. Yüksek-düzeyle işlevler çeviriciye bir *Çağırma Uzlaşımını* izleyerek çevrilir. Bu deney C programlarında programlamacının yazdığı ya da C kütüphanelerinde bulunan işlevlerin nasıl yazılıp çağrılacağını gösterir. İşlevlerin çevirici dilinde nasıl gerçekleştirildiği de gösterilir. Deneyin sonunda işlevlerle kütüphane çağrılarını kullanan programlar yazma üzerine alıştırmalar sunuyoruz.

## 2. İşlevler kullanan bir C programı yazma

Bir işlev – altıyordam (subroutine) ya da işlem (procedure) de denir – tanımlı işlemleriyle arayüzü (girdilerle çıktıları) olan bir kod bloğuna paketlenmiş koddur. Bu modülerlik karmaşıklığı azaltıp kodun yeniden kullanımını destekleyerek verimliliği artırır. Bir işlev, programın rastgele bir noktasından, işlev bittiğinde program yürütmesi işlev çağrısından sonrasında sürdürülecek biçimde çağrılabilir. İşlevler başka bir işlevce (bunlara *içiçe geçmiş* ya da *yuvalanmış* işlev denir) ya da kendinde (*özyinelemeli* çağrılar denir) çağrılabilir.

İşlevleri olan bir RISC-V programı yazmak için Deneyler 2 ile 3'te tanımlanan adımların aynısını izlersin:

1. Bir RVfpga projesi oluştur
2. Bir C programı yaz
3. RVfpga'ı Nexys A7 FPGA kartına indir
4. Programı derle, indir, çalıştırıp/ayıkla

Bu adımlara detaylı yönergeler için Deney 2'ye bak. Aşağıda adımların özet tanımları vardır.

### Adım 1. Bir RVfpga projesi oluşturCreate an RVfpga project

project1 adlı projeni şu klasörde oluştur:

[RVfpgaPath]/RVfpga/Labs/Lab4

### Adım 2. Bir C programı yaz

Şimdi projeye bir C programı ekleyeceksin. Projede yeni bir dosya oluşturup şu C programını yaz ya da kopyala/yapıştır. Bu program şu dosyadan da erişilebilirdir:

[RVfpgaPath]/RVfpga/Labs/Lab4/LedsSwitches\_functions.c

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

#define READ_GPIO(dir) (*(volatile unsigned *)dir)
#define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) =
(value); }

void IOsetup();
unsigned int getSwitchVal();
void writeValtoLEDs(unsigned int val);

int main ( void )
```

```

{
    unsigned int switches_val;

    IOsetup();
    while (1) {
        switches_val = getSwitchVal();
        writeValtoLEDs(switches_val);
    }

    return(0);
}

void IOsetup()
{
    int En_Value=0xFFFF;
    WRITE_GPIO(GPIO_INOUT, En_Value);
}

unsigned int getSwitchVal()
{
    unsigned int val;

    val = READ_GPIO(GPIO_SWs);    // read value on switches
    val = val >> 16;    // shift into lower 16 bits

    return val;
}

void writeValtoLEDs(unsigned int val)
{
    WRITE_GPIO(GPIO_LEDs, val);    // display val on LEDs
}

```





Dosyayı projenin `src` dizinine kaydedip dosyayı `LedsSwitches_Functions.c` diye adlandır.

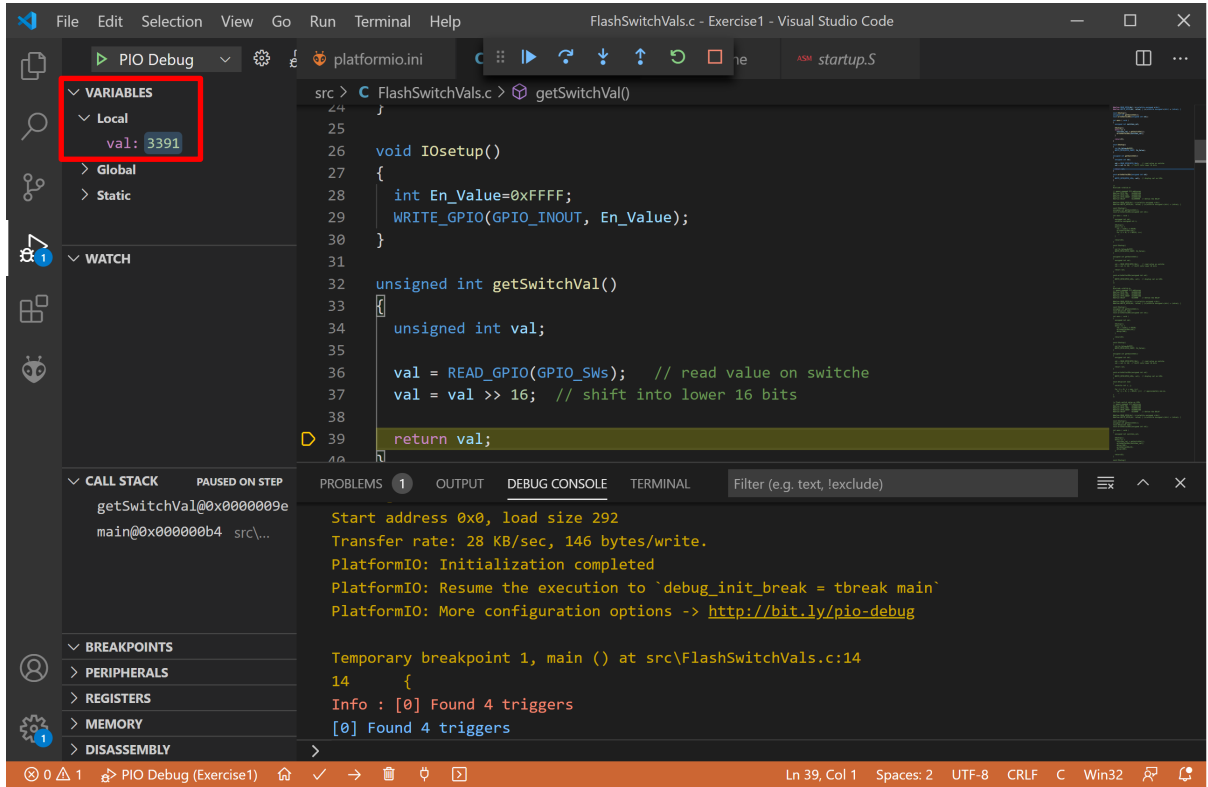
### Adım 3. RVfpga’i Nexys A7 FPGA kartına indir

RVfpga’i Nexys A7 kartına RVfpga Deneyler 2 ile 3’te yaptığın gibi indir.

### Adım 4. Programı derle, indir, çalıştır

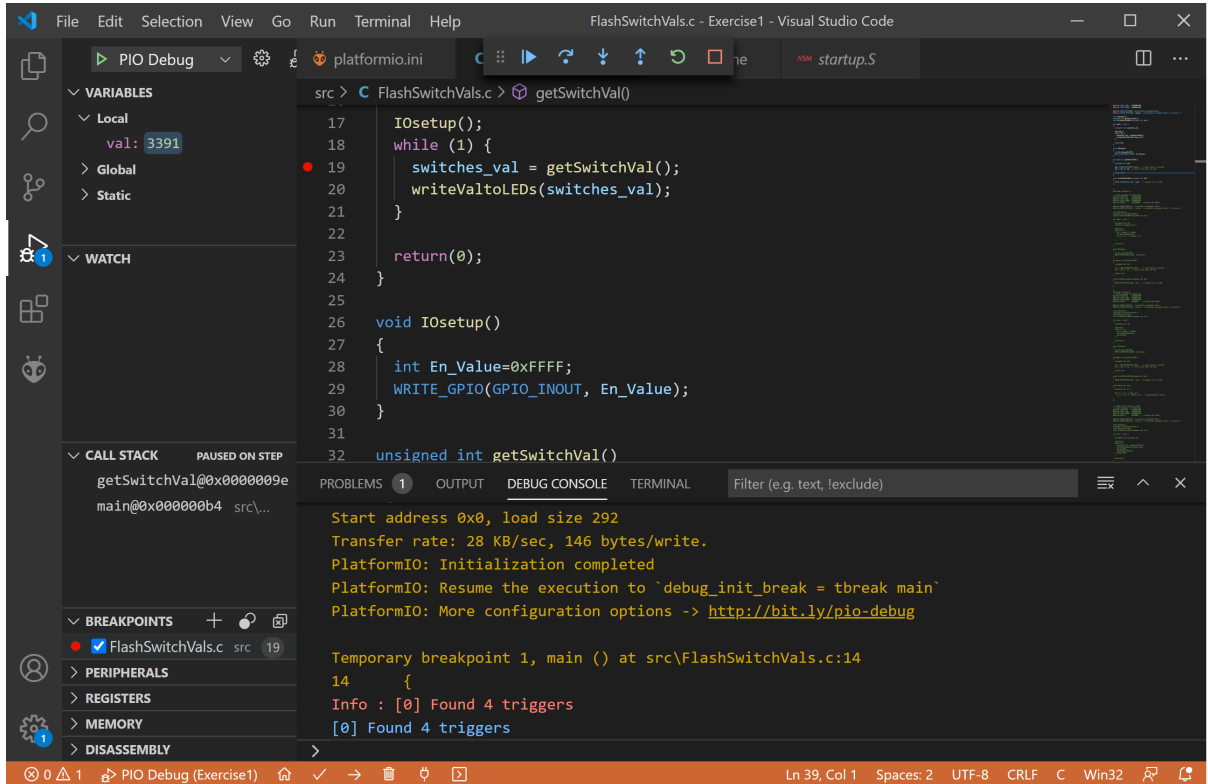
Şimdi programı derleyip, indirip, RVfpga’de çalıştırıp/ayıklayabilirsin.

Run (Çalıştır)  ardından Start Debugging (Ayıklamayı Başlat)  butonlarına bastıktan sonra `getSwitchVal()` işlevini çağıran satır 19’a gelene kadar Step Over butonuna  (üst araç çubuğunda konumlanmıştır) ya da F10’a iki kez tıkla. Ardından Step Into butonuna  (ya da F11) bas. Bu `getSwitchVal()` işlevinin içerisine adım atacaktır. Görülür değilse val değişkenini görmek için sol araç çubuğundaki VARIABLES → Local alanını genişlet. Programın burasında val değişkeni “optimized out” (“optimize edildi”) diye listelenebilir. Bir kez adım atıp (ya Step Over ya da Step Into) ardından val değişkeninin anahtarların değerine dönüştüğünü gör, Figür 1’de gösterildiği gibi.





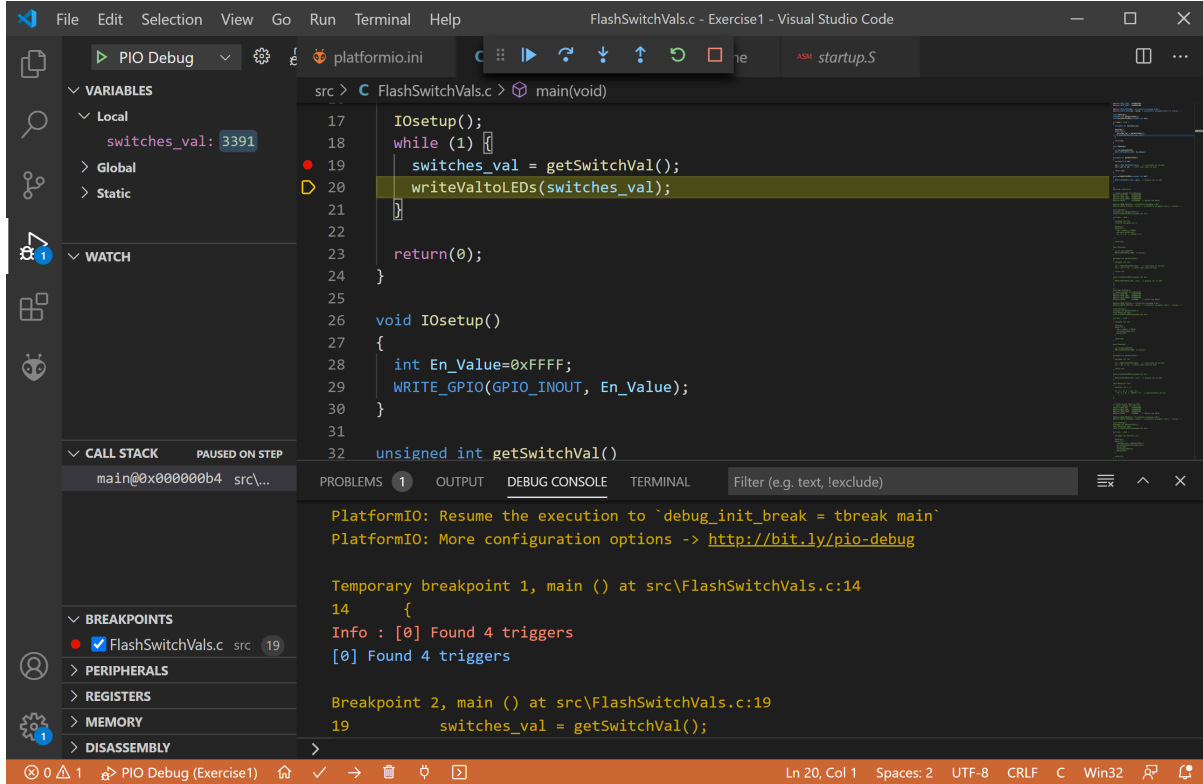
**Figür 1. getSwitchVal() işlevine içeri adım**

Şimdi satırın soluna tıklayarak satır 19'a bir kesme noktası koy. Solda kesme noktası olduğunu gösteren bir kırmızı noktanın gözüktüğünü göreceksin, Figür 2'de gösterildiği gibi.



**Figür 2. Bir kesme noktası ayarlama**

Continue (Sürdür) butonuna  (ya da F5) bas. Program satır 19'da kesme noktasına ulaşıldığında duracaktır. Bu kez Step Over butonuna  (ya da F10) bas. İşlev yürütülecek ancak ayıklayıcı işleve girmeyecektir. İşlevin yalnızca etkileri gösterilecektir. Özellikle switch\_val değişkeni anahtarın değerlerini alacaktır, Figür 3'te gösterildiği gibi.



**Figür 3. İşlevin üzeri adım**

### 3. Kütüphane işlevlerine çağrılar yapan bir C programı yazma

C gibi yüksek-düzeyle programlama dilleri programlamacıların yaygın kullandığı işlevlerin kütüphanelerini içerir. Yaygın kullanılan C kütüphanelerini bulmak için Google'da "C standard libraries" diye aratabilirsin. Bu işlev kütüphaneleri içeriğindeki işlevlerin tanımlarını veren başlık dosyasını içererek kullanılabilir. Şu satırı C program dosyasının başına ekleyerek yapılır:

```
#include <libraryname>
```

"libraryname" kütüphanenin adıyla değiştirilir. Örneğin, matematik kütüphanesi (math.h) şu tür yaygın işlevleri verir: fabs(), kayan noktalı sayının mutlak değerini işler, fmax(), iki kayan noktalı sayının büyüğünü verir, gibi gibi.

Bir diğer yaygın kütüphane C standart kütüphanesidir (stdlib.h). Bu kütüphanede rastgele sayı oluşturan işlevler vardır. Örneğin aşağıdaki program stdlib.h başlık dosyasını (#include <stdlib.h>) içerip rastgele bir sayı döndüren rand() işlevini çağırarak LEDlerde rastgele bir sayı gösterir. Aşağıdaki programı bir PlatformIO RVfpga projesine kopyalayıp yapıştır, sonra Nexys A7 FPGA kartı üzerinde RVfpga'de çalıştır.

```
#include <stdlib.h>

// memory-mapped I/O addresses
#define GPIO_SWs      0x80001400
#define GPIO_LEDs     0x80001404
#define GPIO_INOUT    0x80001408
#define DELAY         0x1000000 // Define the DELAY

#define READ_GPIO(dir) (*(volatile unsigned *)dir)
#define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) =
(value); }

void IOsetup();
unsigned int getSwitchVal();
void writeValtoLEDs(unsigned int val);

int main(void)
{
    unsigned int val;
    volatile unsigned int i;

    IOsetup();
    while (1) {
        val = rand() % 65536;
        writeValtoLEDs(val);
        for (i = 0; i < DELAY; i++)
            ;
    }
    return(0);
}

void IOsetup() {
    int En_Value=0xFFFF;
    WRITE_GPIO(GPIO_INOUT, En_Value);
}

unsigned int getSwitchVal() {
    unsigned int val;

    val = READ_GPIO(GPIO_SWs); // read value on switches
    val = val >> 16; // shift into lower 16 bits

    return val;
}

void writeValtoLEDs(unsigned int val) {
    WRITE_GPIO(GPIO_LEDs, val); // display val on LEDs
}
```

Bu programa şu dosyadan da erişilebilir:

[RVfpgaPath]/RVfpga/Labs/Lab4/RandomNumberLEDs.c

Bu C standart kütüphanelerine ek olarak Western Digital (WD), Firmware Package (<https://github.com/westerndigitalcorporation/riscv-fw-infrastructure>) içerisinde, SweRV EH1 işlemcisi (PSP, sisteminde şurada bulabilirsin `~/.platformio/packages/framework-wd-riscv-sdk/psp/`) ile Nexys A7 kartı (BSP, sisteminde şurada bulabilirsin `~/.platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_eh1/bsp/`) için belirli kütüphaneler sunar. İlk Kullanım Kılavuzunda açıkladığımız gibi (Bölüm 6.F – *HelloWorld\_C-Lang* programı), bu kütüphaneler `platform.ini` dosyasına düzgün satırı ekleyip C programının başlangıcında doğru dosyaları içererek projede içerilir.

Bu kütüphaneler programlamacıların kesintileri kullanmayı, karakter dizisi yazdırmayı, bireysel yazmaçları okumayı/yazmayı, daha birçoğunu yapmasına olanak sağlayan işlevlerle makroları sunar. RVfpga İlk Kullanım Kılavuzunda, deneylerde bu işlevlerin çoğunu örneklerde, alıştırılarda kullanacaksınız.

#### 4. RISC-V Çağırma Uzlaşımı

Bu bölüm yüksek-düzeyle işlevlerin RISC-V çevirici diline nasıl çevrildiğini tanımlayan RISC-V Çağırma Uzlaşımını tanımlar. Bu çağırma uzlaşımı **Application Binary Interface**'in (Uygulama İkili Arayüzü) (ABI) bir parçasıdır. Uzlaşım tanımlanmasıyla farklı programlamacıların yazdığı ya da kütüphanelerde barındırılan işlevler programlar arasında kullanılabilir. In RISC-V'da, **sıçrayıp bağla** yönergesi (`jal`) bir işleve çağrı çalıştırır. Örneğin şu kod `func1` işlevini çağırır:

```
jal func1
```

Bu yönerge `func1` etiketine sıçrarken `jal`'dan sonraki yönergenin adresini dönüş adresi yazmacına kaydeder (`ra = x1`). İşlev ardından `ra`'da depolanan adrese sıçrayan dönüş (`ret`) sözde-yönergesiyle (ya da yazmaca sıçra yönergesiyle: `jr ra`) dönüş yapar.

İşlevler girdi değerleriyle çağrılabilir, çağırılan işleve değer de döndürebilirler. RISC-V uzlaşımına göre girdi argümanları işleve `a0–a7` yazmaçlarında geçirilir. Eğer daha çok argüman gerekirse o argümanlar yığına yerleştirilir. Yine uzlaşımına göre dönüş değerleri `a0` ile `a1` yazmaçlarına yerleştirilir. Hangi yazmaçların **argüman geçirip değerler döndüreceği** RISC-V Çağırma Uzlaşımında tanımlanır.

İşlevin programın istenilen yerinden güvenli bir biçimde çalıştırılabilmesi için işlevin makinenin mimari durumunu koruması (bir diğer deyişle programlamacının görebildiği yazmaçların içeriği) önemlidir. `t0` yazmacını döngünün dizinini depolamak için kullanan `main` işlevli bir programımız olduğunu varsayalım. Döngünün gövdesinde `SortVector` adlı bir işlev çağrılır, `SortVector` işlevi `t0` yazmacını vektör A'nın adresini depolamak için kullanır (Figür 4'e göz at). Böylelikle `SortVector` işlevinde `t0` yazmacının üzerine yazılır, bunun da döngü dizinini değiştirip yürütmenin bozuk olmasına neden olma türünden yan etkisi vardır.

```

main:
    add t2, zero, M
    add t0, zero, zero
    ...
loop1:
    bge t0, t2, endloop1
    ...
    jal SortVector
    ...
    add t0, t0, 1
    j loop1
endloop1:
    ...
    ret

SortVector:
    ...
    la t0, A
    ...
    ret

```

**Figür 4. Bir yazmacın main (ana) program ile işlev arasındaki kullanımdaki çakışmasının örneği**

Açıkça görülüyor ki, eğer `main` programının programlamacı döngü dizinini gerçekleştirmek için başka bir yazmaç (örneğin, `t1`) kullansaydı bu olmazdı. Ancak programlamacıyı bir işlevi çağırmadan gerçekleştirmesinin bütün iç detaylarını bilmeye zorlayamayız (kimi yerde yapılamaz bile).

Bütün işlevler için daha uygun bir çözüm değiştirilecek yazmaçların bellekte geçici kopyasını oluşturup, *çağır*an programa dönmeye önce ilk değerleri yeniden depolamak olur. Bu çözüm LIFO (Son-Giren-İlk-Çıkar) politikasıyla erişilen bellek bölgesi **Çağrı Yığını** ile gerçekleştirilir. Bu bölge programın canlı işlevleriyle (bir diğer deyişle başlamış ancak yürütmesi bitmemiş işlevler) ilgili bütün bilgileri depolamak için kullanılır, erişilebilir belleğin sonundan başlayıp (bir diğer deyişle yüksek adreslerde), düşük adreslere doğru büyür.

Bir işlev normalde üç parçaya ayrılır:

- Giriş kodu (**Önsöz**)
- İşlev **Gövde**
- Çıkış kodu (**Sonsöz**)

**Önsöz** gerekiyorsa bir işlevin **yığın çerçevesini** oluşturup yazmaçları yığının depolamalıdır. **Yığın çerçevesi** işlevin yürütme sırasında kullandığı bellek bölgesidir. **Sonsöz çağır**an programın mimari durumunu yeniden depolayıp **yığın çerçevesinin** kapladığı yeri bırakır, böylelikle yığın **Önsöz** yürütmeden önceki gibi olur.

Yığının erişim, yığının son kaplanan yerinin adresini depolayan **yığın göstergesi** (`sp = x2`) adında, bir gösterge aracılığıyla yönetilir. Program başlamadan önce `sp` yığının tabanının adresiyle (bir diğer deyişle yığın bölgesinin en yüksek adresi) ilk değerini almalıdır. RVfpga'de `sp` yazmacı `_start` işlevinde ilk değerlendirilir, gerçekleştirmesi `~/platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_eh1/startup.S` dosyasındadır. İlk değerlendirmede yığın boştur. İkinci bir gösterge, **çerçeve göstergesi** (`fp = x8`), etkin işlevin **yığın çerçevesinin** taban adresini (bir diğer deyişle en yüksek adres) gösterir.



İşlevler **yığın çerçevesini** yalnızca kendilerinin erişilebildiği özel bellek bölgesi olarak kullanır. **Yığın çerçevesinin** bir parçası işlevin değiştirdiği yazmaçların kopyasını kaydetmeye ayrılmıştır, kimi durumlarda bellek yerleri üzerinden işleve parametre geçirmenin bir yolu olarak da kullanılabilir.

Tablo 1 RISC-V uzlaşmasının tam sayı yazmaçlarına atadığı tasarlanan rolleri tanımlar. Yine Tablo 1’de görselleştirildiği gibi, kimi yazmaçların işlev çağrılarının korunması gerekirken ötekilerinin üzerine yazılabilir (bir diğer deyişle korunmazlar).

- Bir işlevin bir yazmacın üzerine yazması gerekiyorsa, öncelikle yazmacın **yığın çerçevesinde** bir kopyasını yapması, ardından **çağırana** (bir diğer deyişle bu işlevi çağırın işleve) dönmeden değeri yeniden depolaması gerekir. Yığın göstergesiyle (**sp**) dönüş adres yazmacının (**ra**) yanı sıra on iki tam sayı yazmacı **s0–s11** çağrılar arası korunup, kullanılıyorsa **çağırılanın** kaydetmesi gerekir.
- Bir diğer yandan **çağırılanın** kimi yazmaçların **çağrılan** koruması altında olmadığını, dolayısıyla değerlerinin çağrıdan sonra yok olabileceğini bilmesi gerekir. Önemli olarak, argüman ile dönüş değeri yazmaçları (**a0–a7**), yedi tam sayı yazmaçları **t0–t6** uçucu yazmaçlar olup eğer işlev çalıştırmadan sonra yeniden kullanılıyorsa **çağırılanın** kaydetmesi gerekir.

**Tablo 1. RISC-V tam sayı yazmaçları**

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

Figür 4’teki örnekte bu uzlaşma göre iki çözüm olabilir:

- **main** programı döngü dizini için **SortVector** işlevinin koruyacağını garanti olan bir yazmacı (**s0** gibi) **t0** yerine kullanabilir.
- **main** işlevi **t0** kullanmayı sürdürebilir, ancak öyle yaparsa **SortVector** çağırmadan önce içeriğini yığına kaydedip **SortVector** dönüşü sonrasında yeniden depolaması gerekir.

Yığın işlevlerin yığın çerçeveleri daha çok bellek gerektirdikçe genişler, bu işlevler bittikçe küçülür. Yığın aşağı doğru genişlerken (düşük adreslere doğru) yığın göstergesi işlem girişinde bir 16-bayt sınırına hizalanmalıdır. Standart ABI’da yığın göstergesi işlem yürütmesi süresince hizalı kalmalıdır.

## Örnek

Bu örnek önce C’de (Figür 5) ardından RISC-V çevirici dilinde (Figür 6) bir sıralama algoritması gerçekleştirir. Girdi bütün öğelerin 0’dan büyük olduğu N öğeli bir A dizisidir. Çıktı ise A’nın öğelerini azalan sırada depolayan bir diğer B dizisidir.

C’de `main` işlevi `SortVector` işlevini çağırır, o da A ile B dizilerinin adresini, boyutlarını (N) alıp A’nın öğelerini B’ye öge-öge, azalan sırada depolar. Bu `SortVector` işlevi bir diğer işlev olan `MaxVector` işlevini çağırır, o da A dizisinin adresiyle boyutunu alıp A dizisinin en büyük değerinin döndürüp, değeri sıfırlar, böylelikle sonraki yinelemelerde değerlendirilmez

```
#define N 8

int MaxVector(int A[], int size)
{
    int max=0, ind=0, j;
    for(j=0; j<size; j++){
        if(A[j]>max){
            max=A[j];
            ind=j;
        }
    }
    A[ind]=0;
    return(max);
}

int SortVector(int A[], int B[], int size)
{
    int max, j;
    for(j=0; j<size; j++){
        max=MaxVector(A, size);
        B[j]=max;
    }
    return(0);
}

int main ( void )
{
    int A[N]={7,3,25,4,75,2,1,1}, B[N];
    SortVector(A, B, N);
    return(0);
}
```

**Figür 5. C dilinde sıralama algoritması**

Figür 6 aynı algoritmanın çeviricide yazımını görselleştirir. Önceki bölümlerde açıklanan konseptleri göz önünde bulundurarak bu programı çözümlüyoruz.

### - **main işlevi**

#### o Önsöz

- Öncelikle, korunan yazmaçları depolamak için yığında boşluk ayrılır: `add sp, sp, -16`. Önemli olarak, uzlaşmaya uygun olarak, `sp` yazmacı RISC-V’in 128-bit sürümüyle, RV128I, uyumluluğu korumak için hep 16-baytta hizalı olmalıdır.
- Bu işlevde kaydedilen yazmaçlar kullanılmadığı için `s0-s11` yazmaçlarının yığında depolanması gerekmez. Ancak yazmaç `ra` kaydedilmelidir, nedeni ise `main` işlevinin `ra` içerisinde depolanan değeri güncelleyen `SortVector` işlevini çağırıyor olmasıdır.

#### o İşlev Gövdesi

- `SortVector` işlevi `jal SortVector` yönergesiyle çalıştırılır. İşlev çağrılmadan önce, Çağırma Uzlaşımına uygun olarak, 3 girdi parametresi `a0` (A'nın adresi), `a1` (B'nin adresi), ile `a2` (A ile B dizilerinin boyutu) yazmaçlarına yerleştirilir.
- Sonsöz
  - Önsözde yığına kaydedilen yazmaç (`ra`) yeniden depolanır.
  - Yığın göstergesi (`sp`) ilk pozisyonuna yeniden depolanır: `add sp, sp, 16`.
- **SortVector işlevi**
  - Önsöz
    - İlk olarak işlevde kullanılan korunan yazmaçları depolamak için yığında boşluk ayrılır: `add sp, sp, -32`.
    - Ardından işlevde kullanılan kaydedilmiş yazmaçlar (`s1-s3`) bir bir yığında depolanır.
    - Yazmaç `ra` da kaydedilmelidir, nedeni ise `SortVector`'ün `ra` biçerisinde depolanan değerin üzerine yazan `MaxVector` işlevini çağırıyor olmasıdır.
  - İşlev Gövdesi
    - İlk olarak girdi parametreleri (`a0, a1, a2`) korunan yazmaçlara taşınır (`s1, s2 and s3`) ki `MaxVector` işlevi yürütüldükten sonra da kullanılabilirler.
    - B vektörünü hesaplamak için yinelemelerle A'nın maksimum değerini hesaplayıp B'ye depolayan bir döngünün gerçekleştirilmesi yapılır. A'nın maksimum değerini hesaplamak için `MaxVector` işlevi bütün yinelemelerde çalıştırılır: `jal MaxVector`. İşlev çağrılmadan önce, Çağırma Uzlaşımına uygun olarak, işlevin girdi parametreleri `a0` ile `a1` yazmaçlarına taşınır. İşlev yürütmeyi bitirdiğinde A'nın maksimum değeri `a0` yazmacında döndürülür.
    - Önemli olarak döngü, değişkenleri depolamak için çoğunlukla kaydedilmiş yazmaçları kullanır. Bu yazmaçların değerlerini `MaxVector` yürütmesinden sonra koruyacağının garantisi RISC-V Çağırma Uzlaşımında verilmiştir (bir diğer deyişle işlev değerlerini korumalıdır).
    - `a0` ile `a1` yazmaçlarını işlev değiştirebilir. Dolayısıyla çalıştırmadan önce hazırlanmalıdırlar.
    - `t1` yazmacının `MaxVector` dönüşünden sonra yeniden kullanılması gerekir. Dolayısıyla işlev çağrılmadan önce `SortVector`'ün yığında depolanıp (`sw t1, 16(sp)`) yürütme bitince yeniden depolanmalıdır (`lw t1, 16(sp)`).
  - Sonsöz
    - Önsözde kaydedilen yazmaçlar yeniden depolanır.
    - Yığın göstergesi (`sp`) de ilk pozisyonuna yeniden depolanır: `add sp, sp, 32`.
- **MaxVector işlevi**
  - Önsöz
    - Öncelikle yığında korunan yazmaçları depolamak için boşluk ayrılır: `add sp, sp, -16`.
    - Ardından işlevin kullandığı kaydedilmiş yazmaç (`s1`) yığında depolanır: `sw s1, 0(sp)`. Önemli olarak eğer yazmaç bu işlevde

korunmasaydı *çağırın* işlevin yürütmesi (`SortVector`) bozulurdu, nedeni ise vektör A'nın adresini depolamak için bu yazmacı kullanıyor olması.

- Bu işlev başka bir işlev çağırmayacağından (bir *yaprak* işlevdir), `ra` bu durumda kaydedilmesi gerekmez.

○ **İşlev Gövdesi**

- İşlev, A dizisindeki maksimum değeri hesaplamak için `s1` ile diğer geçici yazmaçları kullanıyor.

○ **Sonsöz**

- *Çağırana* dönmenden önce işlev dönüş değerini hazırlamalıdır: `mv a0, t2`.
- Önsözde yığına kaydedilen yazmaç (`s1`) yeniden depolanır.
- Yığın göstergesi (`sp`) de ilk pozisyonuna yeniden depolanır: `add sp, sp, 16`.

```
.globl main

.equ N, 8

.data
A: .word 7,3,25,4,75,2,1,1

.bss
B: .space 4*N

.text

MaxVector:
    add sp, sp, -16
    sw s1, 0(sp)

    mv s1, zero
    mv t2, zero
loop2:
    beq s1, a1, endloop2
    lw t1, (a0)
    ble t1, t2, else2
    mv t2, t1
    mv t3, a0
else2:
    add a0, a0, 4
    add s1, s1, 1
    j loop2
endloop2:
    sw zero, (t3)

    mv a0, t2
    lw s1, 0(sp)
    add sp, sp, 16
    ret

SortVector:
    add sp, sp, -32
    sw s1, 0(sp)
    sw s2, 4(sp)
    sw s3, 8(sp)
    sw ra, 12(sp)

    mv s1, a0                # Address of vector A
    mv s2, a1                # Address of vector B
    mv s3, a2                # Size of vectors A and B
```

```

mv t1, zero

loop1:
    beq t1, s3, endloop1
    mv a0, s1
    mv a1, s3
    sw t1, 16(sp)
    jal MaxVector
    lw t1, 16(sp)
    sw a0, (s2)
    add s2, s2, 4
    add t1, t1, 1
    j loop1
endloop1:

lw s1, 0(sp)
lw s2, 4(sp)
lw s3, 8(sp)
lw ra, 12(sp)
add sp, sp, 32
ret

main:
    add sp, sp, -16
    sw ra, 0(sp)

    la a0, A
    la a1, B
    add a2, zero, N
    jal SortVector

    lw ra, 0(sp)
    add sp, sp, 16
    ret

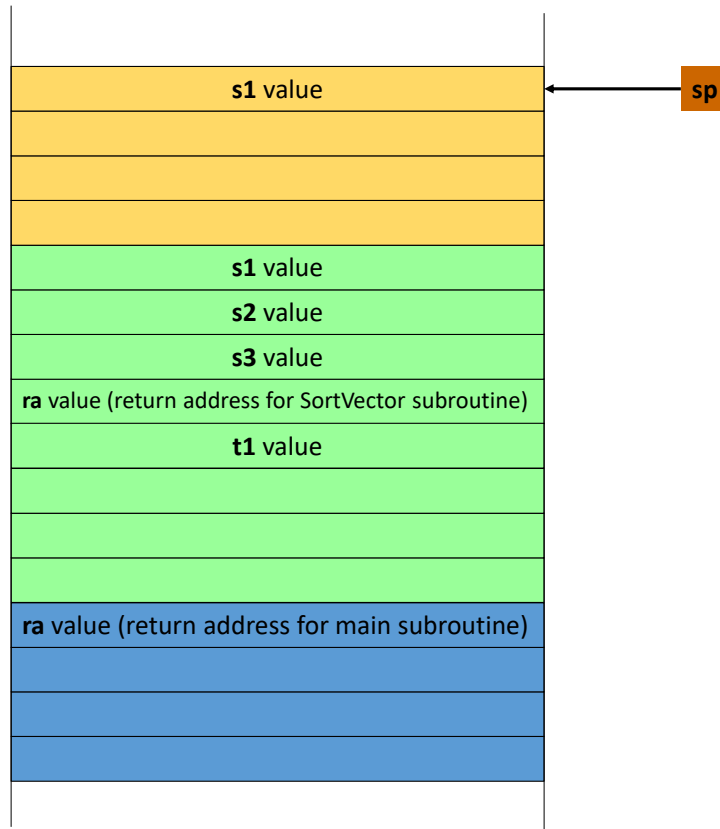
.end

```

**Figür 6. Çevirici dilinde sıralama algoritması**

**Figür 7** `MaxVector` işlevinin gövdesini yürütürken yığının durumunu görselleştirir.

- `main` işlevinin *yığın çerçevesi* maviyle gösterilirken işlev için dönüş adresini (`ra`) içerir.
- `SortVector` işlevinin *yığın çerçevesi* yeşille gösterilirken işlevin kullandığı kaydedilmiş yazmaçları (`s1-s3`), yazmaç `t1`, ile `ra` içerir.
- Son olarak, `MaxVector` işlevinin *yığın çerçevesi*, ki bu *etkin yığın çerçevesidir* (yürütülen işlevin *yığın çerçevesi*) sarıyla gösterilirken bu işlevin kullandığı kaydedilmiş yazmacı (`s1`) içerir.



**Figür 7. Figür 6'daki çevirici programı için MaxVector işlevinin gövdesindeyken yığın durumu.**

**GÖREV:** Figür 6'daki çevirici programı PlatformIO projesi olarak şurada sağlanmıştır: `[RVfpgaPath]/RVfpga/Labs/Lab4/SortingAlgorithm_Functions`. Programı kartta (ya da ISS simülatöründe) RSIC-V Çağırma Uzlaşımına uygun olarak yığında depolanmış değerle değişik yazmaçları (`s`, `ra`, `a`, gibi) çözümlemek için ayıklayıcıyla adım adım çalıştır.

- Dosya `.pio/build/rvfpga/firmware.dis`, PlatformIO program derlendikten sonra oluşturur bu dosyayı, programındaki yönergelerin adreslerini bilmek için kullanışlı olabilir.
- Yığının evrimiyle A ile B dizilerinin evrimini çözümlemek için Bellek Konsolunu (Memory Console) kullanabilirsin.
- Bu projede `sp` yazmacını 16-bayta hizalanmaya zorlayan ayarlanmış `link.lds` senaryosunu kullandık. Senaryoyu şurada bulabilirsin `[RVfpgaPath]/RVfpga/Labs/Lab4/SortingAlgorithm_Functions/ld/link.lds`. `sp` yazmacının hizası `ALIGN()` komutuyla zorlanır:

```
.stack :
{
    _heap_end = .;
    . = . + __stack_size;
    /* Force 16-B alignment of SP register */
    . = ALIGN(16);
    _sp = .;
} > ram : ram_load
```

## 5. Alıştırmalar

Şimdi bu alıştırmaları bitirerek işlev çağrılarını içeren kendi C/Çevirici programlarını oluşturacaksın.

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.** Anahtarların tersini LEDlerde gösteren bir C programı yaz. Programı **DisplayInverse\_Functions.c** olarak adlandır.

Örneğin anahtarlar (ikili sistemde): 01010101010101 ise, LEDler şunu göstermeli: 10101010101010; eğer anahtarlar: 1111000011110000 ise, LEDler şunu göstermeli: 0000111100001111; gibi gibi. Anahtarların değerinin tersini döndüren bir `getSwitchesInvert()` işlevini içer. İşlev bildirisi şudur:

```
unsigned int getSwitchesInvert();
```

**Alıştırma 2.** LEDlerin değerini anahtarlara yakıp söndüren bir C programı yaz. Programı **FlashSwitchesToLEDs\_Functions.c** olarak adlandır.

Değer iki saniyede bir yanıp sönmeli. num milisaniye kadar gecikmeye neden olan `delay()` işlevi içer. Bu gözlemsel, kesin olmayacak biçimde yapılabilir. İşlev bildirisi şöyle görünür:

```
void delay(int num);
```

**Alıştırma 3.** Tepki süresini ölçen bir C programı yaz. Programın bütün LEDler yandıktan sonra bir kişinin en sağdaki anahtarı (SW[0]) açmasına kadar geçen süreyi ölçmeli. Kullanıcının tepki süresini denemek için `stdlib.h` kütüphanesinden `rand()` işlevini rastgele bir gecikme oluşturmak için kullanacaksın. Programı **ReactionTime.c** olarak adlandır.

Program şöyle çalışmalıdır.

Program şöyle çalışmalıdır.

1. Kullanıcı başlamak istediklerini belirtmek için en sağdaki anahtarı kapatır (aşağı).
2. Program bütün LEDleri söndürür, rastgele bir süre için bekler (bu 3 saniyeyi geçmemeli). Alıştırma 2'den `delay()` işlevini kullanmak isteyebilirsin.
3. Ardından bütün LEDler açılır, program kullanıcı en sağdaki anahtarı açana kadar geçen süreyi milisaniye biriminde ölçer.
4. Kullanıcı en sağdaki anahtarı (SW[0]) açtığında, açana kadar geçen süre milisaniye biriminde, ikili tabanda LEDlerde, onlu tabanda dizesel konsolda gösterilir.
5. Oyun ardından kullanıcının en sağ anahtarı indirmesiyle (kapatmasıyla) tekrarlar.

**Alıştırma 4.** `rand()` işlevinin bir sıkıntısı kestirilebilir bir rastgele sayı dizisi kullanmasıdır. Yani bütün açılışlarında aynı rastgele sayıyla başlayıp, izlediği rastgele sayılar çalıştırmalar arasında aynı olacaktır. Alıştırma 3'teki programı birkaç kez çalıştırıp aynı rastgele sayıyla başlayıp, aynı rastgele sayı dizisini izlediğini gözlemle.

Ancak ilk olarak `srand()` işlevini kullanırsan bu `rand()` işlevini rastgele bir başlangıç noktasıyla tohum verecektir. Sıkıntı ise `srand()`'ın bir pozitif tam sayı argüman alıp bunun rastgele

olmasının gerekmesi. `srand()`'a bir rastgele sayı ver, örneğin, kullanıcının anahtarı kapatmasına kadar geçen süreyi milisaniye biriminde verebilirsin.

Alıştırma 3'ü LEDler yakılana kadar gerçekten rastgele bir sayı dizisi oluşturacak biçimde yeniden yaz. Elinden geldiğince işlevler kullan. Programı **ReactionTimeTrulyRandom.c** olarak adlandır.

**Alıştırma 5.** Alıştırma 4'ü tepki süresine orantılı olarak LEDlere büyüyen bir LED çubuğu gösterecek biçimde yeniden yaz. Böylelikle kullanıcı milisaniyenin ikili tabanda ne olduğunu anlamaya uğraşmadan hızlanıp hızlanmadıklarını kolayca görebilir. Hangi tepki süresinin kaç LEDi yakacağını aralığını kendin seçebilirsin. Örneğin hızlı tepki süreleri için yalnızca birkaç LED yakılmalı. Tepki süresi arttıkça sola doğru daha çok LED yakılmalı. Çok yavaş bir tepki bütün LEDleri yakmalı. Programı **ReactionTimeBar.c** olarak adlandır.

**Alıştırma 6.** "Simon says" oyununu gerçekleştiren bir C programı yaz. Şunlar olmalı:

1. Program en sağ üç LEDde bir örüntüyü yakıp söndürüp kullanıcının karşılık gelen anahtar dizisini en sağ üç anahtarda basmasını bekler. `Switches[2:0]`, `LED[2:0]`'a karşılık gelir, `LED[0]` en sağ LED iken `Switches[0]` en sağ anahtardır.
2. Rastgele örüntüler ilk 1 LED yakarak başlamalı, sonra 2 LED, sonra 3, gibi.
3. Ardından kullanıcı diziyi en sağ üç anahtarı kullanarak tekrarlamaya çalışır. Kullanıcı anahtarları açtıkça karşılık gelen LEDler yanmalı (kullanıcı anahtarı kapattığında da sönmeli).
4. Eğer kullanıcı doğru diziyi girerse biraz beklemenin ardından sonraki örüntü görüntülenmeli, dizide bir tane daha LED kullanarak.
5. Eğer kullanıcı yanlış diziyi girerse LEDler yakılı kalır, yeni dizi oynatılmaz.
6. Oyun en soldaki anahtar yukarı kaldırılıp (açıp) ardından indirilerek (kapatıp) sıfırlanır (`Switches[15]`).

Programı modülerleştirip, yazmayı, ayıklamayı, anlamayı kolaylaştırmak için istediğin gibi işlevler kullan. Programını yazmak için istediğin standart C kütüphanesini kullanabileceğini unutma. Programı **SimonSays.c** olarak adlandır.

**Alıştırma 7.**  $3 \times N$  ögeli bir A vektöründen, bütün ögelerinin üç ardışık A ögesinin toplamının mutlak değeri olan bir N ögeli B vektörü elde etmek istiyoruz. Örneğin:

$$B[0] = |A[0]+A[1]+A[2]|, \quad B[1] = |A[3]+A[4]+A[5]|, \quad \dots$$

**Triplets.S** (program RISC-V çağırma uzlaşımına uymalıdır) adında bir RISC-V çevirici programı yaz:

- `main` programı B'nin hesaplamasını şu yüksek düzeyli sözde koda göre gerçekleştir:

```
#define N 4

int A[3*N] = {a list of 3*N values};
int B[N];
int i, j=0;

void main (void)
{
```



```

for (i=0; i<N; i++){
    B[i] = res_triplet(A,j);
    j=j+3;
}

```

- `res_triplet` işlevi `V` vektörünün 3 ardışık ögesinin toplamının mutlak değerini `p` pozisyonundan başlayarak döndürür. Şu yüksek düzeyli sözde kodun sağladığı spesifikasyona göre gerçekleştirilir:

```

int res_triplet(int V[ ], int pos)
{
    int i, sum=0;
    for (i=0; i<3; i++)
        sum = sum + V[pos+i];
    sum=abs(sum);
    return sum;
}

```

- `abs(int x)` işlevi girdi argümanının mutlak değerini döndürür.

**Alıştırma 8. Filter.S** (program önceden üzerine çalıştığımız işlev yönetimi için standart uyumlu olmalıdır) adında bir RISC-V çevirici programı yaz. Şu sözde kodu kullanabilirsin:

```

#define N 6
int i, j=0, A[N]={48,64,56,80,96,48}, B[N];
for (i=0; i<(N-1); i++){
    if( (myFilter(A[i],A[i+1])) == 1){
        B[j]=A[i]+ A[i+1] + 2;
        j++;
    }
}

```

- Denk gelen RISC-V kodunu bellek boşluğu ayırma, karşılık gelen bölümleri bildirme (`.data`, `.bss`, `.text`) için gereken yönlendirmeleri içererek yaz. `myFilter` işlevi ilk argüman 16'nın katları, ikinci argüman da birinciden büyükse 1 değeri döndürür, bu doğru değilse 0 döndürür.
- `myFilter` işlevinin çevirici kodunu yaz.

**Alıştırma 9. (>0) tam sayı çiftleri listesi verildiğinde aralarında asal sayıları bulan, Coprimes.S** (program önceden üzerine çalıştığımız işlev yönetimi için standart uyumlu olmalıdır) adında bir RISC-V çevirici programı kurgulamak istiyoruz. Eğer iki sayının tek ortak böleni 1 ise aralarında asaldırlar.

Girdinin şu biçimdeki bir `D` dizisinde barındırıldığını varsayıyoruz:

$$D = (x_0, y_0, c_0, x_1, y_1, c_1, \dots, x_{N-1}, y_{N-1}, c_{N-1})$$

Bütün  $(x_i, y_i, c_i)$  üçlüleri şöyle yorumlanır:  $x_i$  ile  $y_i$  bir sayı çiftini gösterir,  $c_i$  ise ilk değer olarak 0'dır. Programı çalıştırdıktan sonra bütün  $c_i$ 'lerin değeri öyle değiştirilmelidir ki  $c_i = 2$ , eğer  $x_i$  ile  $y_i$  aralarında asal ise; öteki türlü, yani aralarında asal değiller ise  $c_i = 1$ .

Örneğin:

Şu girdi vektörü için:  $D = (3,5,0, 6,18,0, 15,45,0, 13,10,0, 24,3,0, 24,35,0)$

Sonuç şu olmalıdır:  $D = (3,5,2, 6,18,1, 15,45,1, 13,10,2, 24,3,1, 24,35,2)$

- D dizisini dolaşıp aşağıdaki sol kutuda verilen spesifikasyona göre sonuç oluşturan bir RISC-V çevirici programı yaz. Program, girdi argümanları D'nin başlangıç adresiyle denetlemek istediğimiz çift sayısını (0'dan M-1'e) olan `check_coprime (int D [], int i)` işlevini çağırır. İşlev dizinin i-nci çiftinin aralarında asal olup olmadığını denetleyip sonucu karşılık gelen bellek yerinde depolar.
- Sağ kutuda verilen spesifikasyona göre `check_coprime`, işlevleri için kodu yaz. `gcd(int a, int b)` işlevinin Deney 3'te gerçekleştirilip Öklit algoritmasına göre iki girdi argümanın en büyük ortak bölenini (`gcd`) verdiğini unutma. `gcd` 1 ise sayılar aralarında asaldır.

<pre>#define M 6 int D[] = {a list of M*3 int values} void main ( ) {     int i;     for (i=0; i&lt;M; i++)         check_coprime(D,i); }</pre>	<pre>void check_coprime (int A[ ], int pos) {     int res;     res = gcd( A[3*pos], A[(3*pos)+1] );     if (res == 1)         A[(3*pos)+2] = 2;     else         A[(3*pos)+2] = 1; }</pre>
---	--