Bilgisayar Mühendisliğine Giriş – BLM 101

Hafta 14: Kesme ve Yığınlar

Fenerbahçe Üniversitesi

14. Hafta İçeriği

- Trap Mekanizması
- Trap Komutları
- Altyordamlar (Subroutines)
- Yığın (Stack), Soyut Veri Tipi

Sistem Çağrıları (System Calls)

- Bazı operasyonları gerçekleştirmek derinlemesine bilgi ve çeşitli koruma mekanizmalarının kullanımını gerektirir
  - Haberleşilecek cihaz ile Giriş/Çıkış saklayıcıları ile haberleşirken gereken protokolün öğrenilmesi gerekir.
  - Bir giriş çıkış farklı bir uygulama tarafından da kullanılıyor olabilir. Kaynak kullanımının kontrol edilmesi gerekir.
- Her program geliştiren kişi bu detayları bilmek istemeyebilir.
- Sistem fonksiyonları veya sistem çağrıları, işletim sistemi tarafından güvenli bir biçimde operasyonları gerçekleştiren yapılardır.

Sistem Çağrıları (System Calls)

- 1. Programcı sistem çağrısını çağırır.
- 2. İşletim sistemi işlemi gerçekleştirir.
- 3. Programın akışı devam eder.

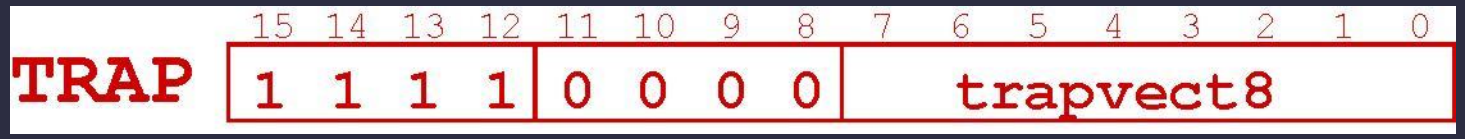
LC3 işlemcisinde bu mekanizma "TRAP" ile gerçekleştirilmektedir.

LC-3 TRAP Mekanizması

- *1. Çeşitli fonksiyonları gerçekleştirir*
  - Genellikle işletim sisteminin bir parçasıdır
- *2. Adresler.*
  - 0x0000'dan 0x00FF'e kadar adres aralığında tutulur
  - Bazı işlemcilerde "Sistem Kontrol Bloğu" olarak da geçer
- *3. TRAP komutu.*
  - Kontrolü işletim sistemi fonksiyonlarına bırakır
  - 8-bit trap vektörüne, 2^8 256 farklı fonksiyon tanımlanabilir
- *4. İşlem tamamlanınca, tekrar kullanıcı programına geri dönülür.*
  - TRAP komutu çağırıldıktan sonra, PC kaldığı yerden uygulamayı yürütmeye devam eder

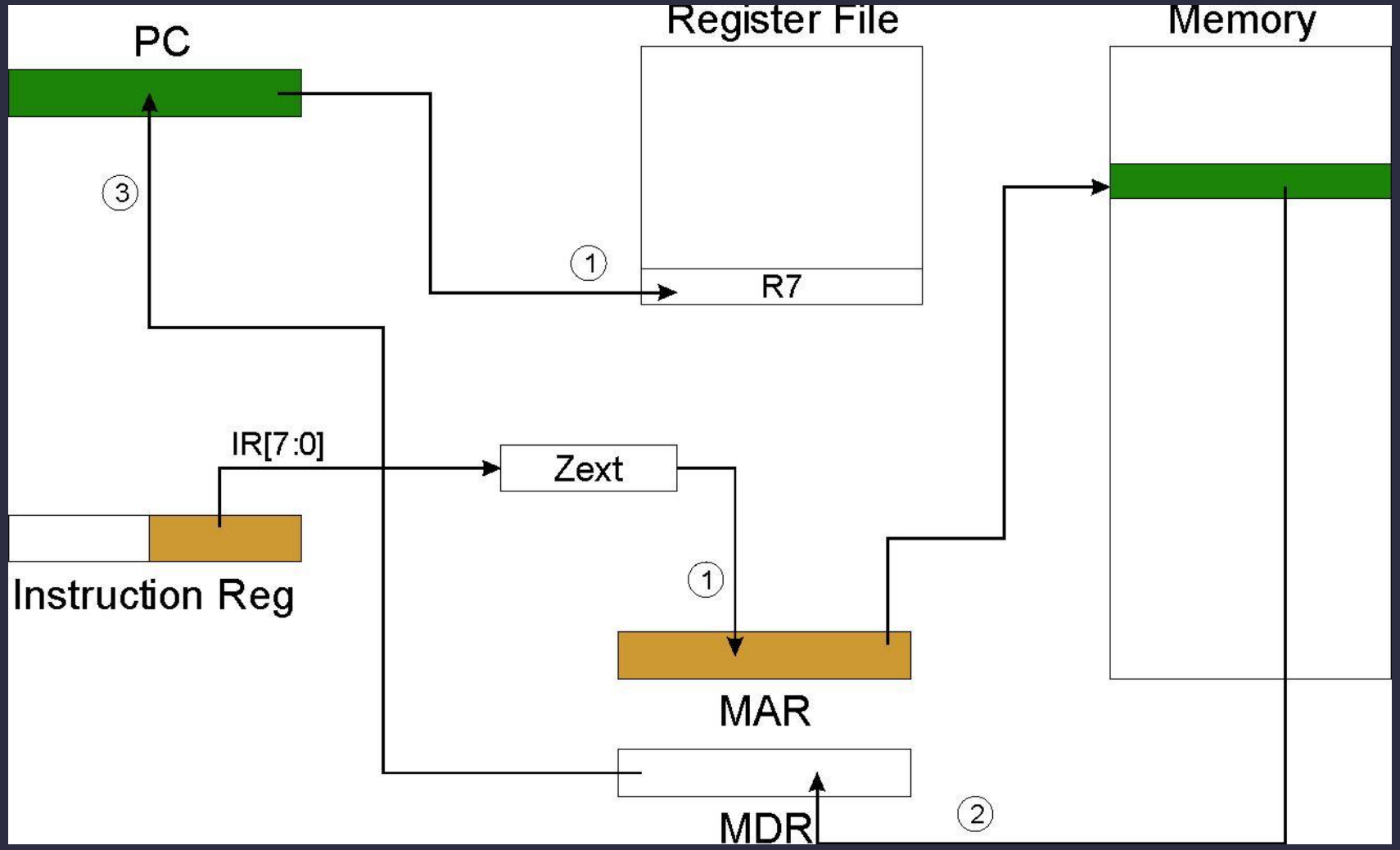
TRAP Komutu

- Trap Vektörü
  - Hangi sistem çağrısının (fonksiyonunun) başlatılacağını gösterir
  - 8-bit'lik sayı, fonksiyonun başlangıç adresini tutar
    - LC-3 işlemcisinde 8 bitlik bakılan adres aralığı: 0x0000 – 0x00FF



- İşlemcinin kodu devam ettireceği adres
  - 0x0000-0x00FF aralığındaki hangi adres verildiyse, o adresin içeriğindeki sayı, PC'e eşitlenerek o adresten işlem yapılmaya devam edilir.
- Geri dönüş
  - R7 saklayıcısına, PC (Program Counter)'in değeri saklanır, sistem çağrısı tamamlanınca R7 saklayıcındaki değer, tekrar PC'e aktarılır.

TRAP


PC
Register File
Memory
3
1
R7
IR[7:0]
Zext
Instruction Reg
1
MAR
MDR
2

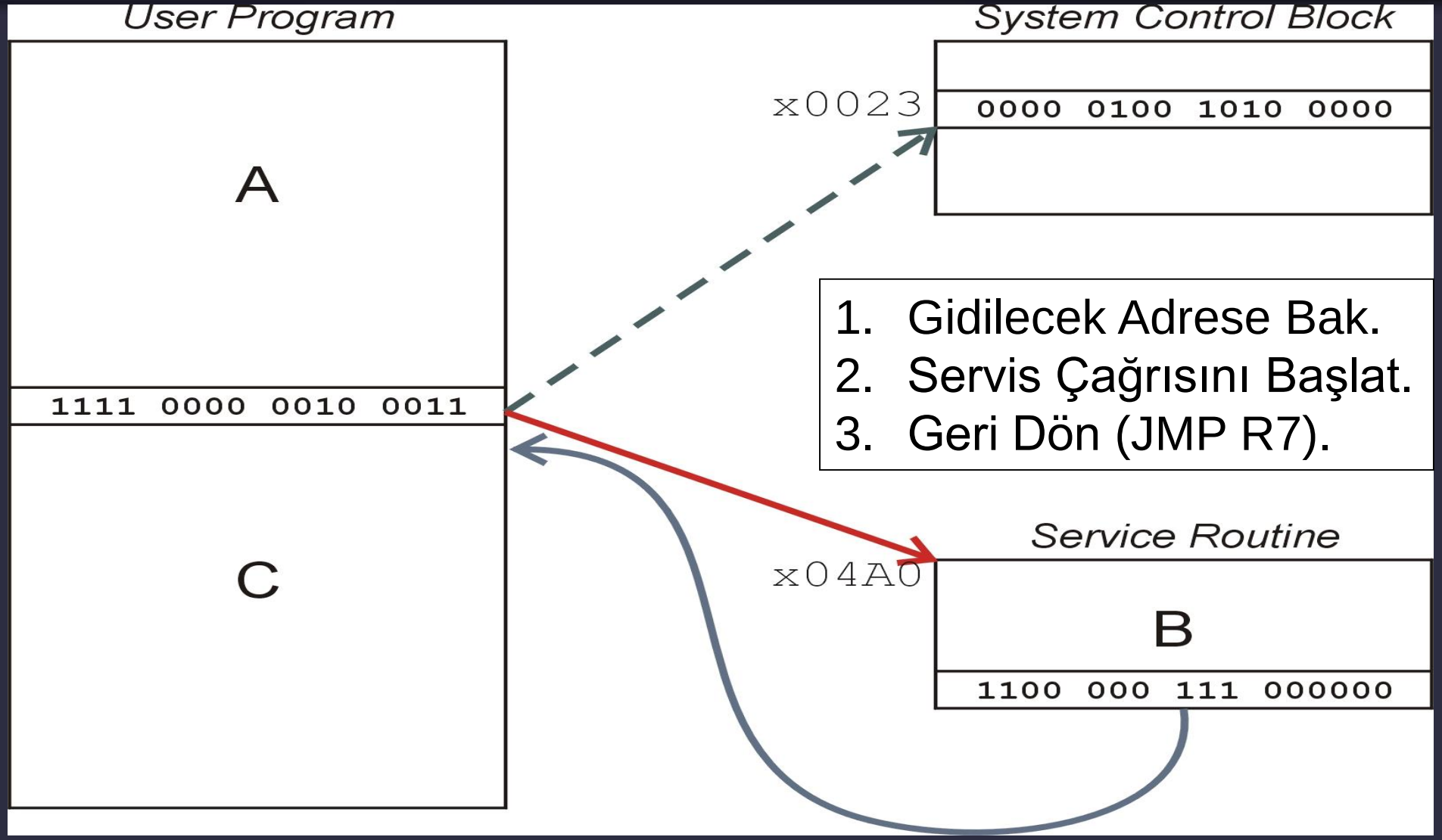
RET (JMP R7)

Geri Dönüş

- Eski PC’i R7 saklayıcısına kaydedildi
  - JMP R7, sistem çağrısı olmadan önceki adrese geri götürmektedir.
  - LC-3 assembly dilince "RET" ifadesi geri dönmeyi sağlamaktadır. Arka planda “JMP R7” kodu çalıştırılmaktadır.

- Sistem çağrısı esnasında R7 saklayıcısının değeri değiştirilmemelidir. Aksi taktirde geri dönülecek adres bilgisi kaybolacaktır.

TRAP Mechanism Operation



TRAP Komutu Kullanımı

```
          .ORIG x3000
          LD   R2, TERM    ; -55 sayısı
DONGU     TRAP x23         ; Giriş Karakter Alımı (R0)
          ADD  R1, R2, R0  ; R1 = R2 + R0, Alınan R0 == 55 ise programı bitir, ASCII 55 =7
          BRz  BITIR       ; 7 sayısı girildiyse bitir
          TRAP x21         ; Monitör'e aktarım (R0)
          BRnzp DONGU      ; döngü
TERM      .FILL    xFFC9   ; -'55'
BITIR     TRAP x25         ; halt
          .END
```

Örnek Çıktı Verme Fonksiyonu (Sistem Çağrısı)

```
            .ORIG x0430          ; Fonksiyon Başlangı Adresi
            ST    R7, R7KAYIT    ; R7 & R1 Kaydetme
            ST    R1, R1KAYIT
; ----- Karakter Yazdırma
KONTROL     LDI   R1, CRTSR      ; Durum Kontrol
            BRzp   KONTROL       ; 15. bit'in aktif olmasını bekle
YAZ         STI   R0, CRTDR      ; Karakteri Yaz
; ----- Çağrılan yere dön
DONUS       LD    R1, SaveR1     ; R1 & R7 Geri Kaydet
            LD    R7, SaveR7
            RET                  ; Geri dön

CRTSR       .FILL xF3FC ; durum saklayıcısı
CRTDR       .FILL xF3FF ; veri saklayıcısı
R1KAYIT     .FILL 0
R7KAYIT     .FILL 0
            .END
```

x21 adresinde bulunuyor

TRAP Fonksiyonları

vector	*Sembol*	*Açıklama*
x20	GETC	Tek bir karakteri okur
x21	OUT	Bir karakteri ekrana bastırır
x22	PUTS	Bir metni ekrana bastırır
x23	IN	Bir karakteri alır ve ekrana bastırır
x25	HALT	Programı bitirir

Saklayıcıların Değerlerinin Kaydedilmesi

- Bir saklayıcının değeri aşağıdaki durumlar için kaydedilmelidir:
  - Bir servis fonksiyonu esnasında değeri bozulacaksa
  - Ve bu saklayıcının değeri, fonksiyon bitiminden sonra kullanılıyorsa

- Kim Kaydetmeli?
  - Servis fonksiyonunu çağıran mı?
    - Daha sonra kullanılacağını bilir, ancak servis fonksiyonunun onun değerini bozabileceğini bilemez
  - Çağrılan servis fonksiyonu mu?
    - Neyi değiştirdiğini bilir, ancak daha sonra ihtiyaç duyulabileceğini bilemez

Örnek

```
          LEA  R3, Binary
          LD   R6, ASCII
          LD   R7, COUNT  ; 10 değeri atanıyor
AGAIN     TRAP x23        ; Klavyeden karakter al
          ADD  R0, R0, R6
          STR  R0, R3, #0
          ADD  R3, R3, #1
          ADD  R7, R7, -1       ; R7 Değiştiriliyor
          BRp  AGAIN            ; Başa dön
          BRnzp NEXT

ASCII     .FILL      xFFD0
COUNT     .FILL      #10
Binary    .BLKW #10
```

R7 saklayıcısı geri dönmeyi sağlar,
Ancak değeri değiştiriliyor

Saklayıcıların Değerlerinin Kaydedilmesi

- Çağrılan Fonksiyon
  - Çalışmadan saklayıcının değerini belleğe kaydeder.
  - Çağrıldığı yere dönmeden önce, bellekte saklanan değerleri alıp saklayıcılara geri yazar

- Çağıran Fonksiyon (Yada ana kod)
  - Bir sistem fonksiyonu çağırmadan önce saklayıcıları belleğe kaydeder.
  - Sistem çağrısı bittikten sonra saklayıcıların değerlerini bellekten alıp geri kaydeder.
    - R7 saklayıcısının değerinin TRAP çağrısından önce kaydedilmesi
    - R0 saklayıcının değerinin TRAP x23 (klavyeden giriş alma) çağrıdan önce kaydedilmesi

Servis Fonksiyonlarını neden Gerekli?

Servis fonksiyonları 3 nedenden ötürü gereklidir:

- 1. Programcıları sistem spesifik detaylara girmemesini sağlar
- 2. Sıklıkla yazılan aynı kodun, tek bir yerden çağrılmasını sağlar
- 3. Sistem kaynaklarını kötü niyetli yazılımcılardan koruyabilir

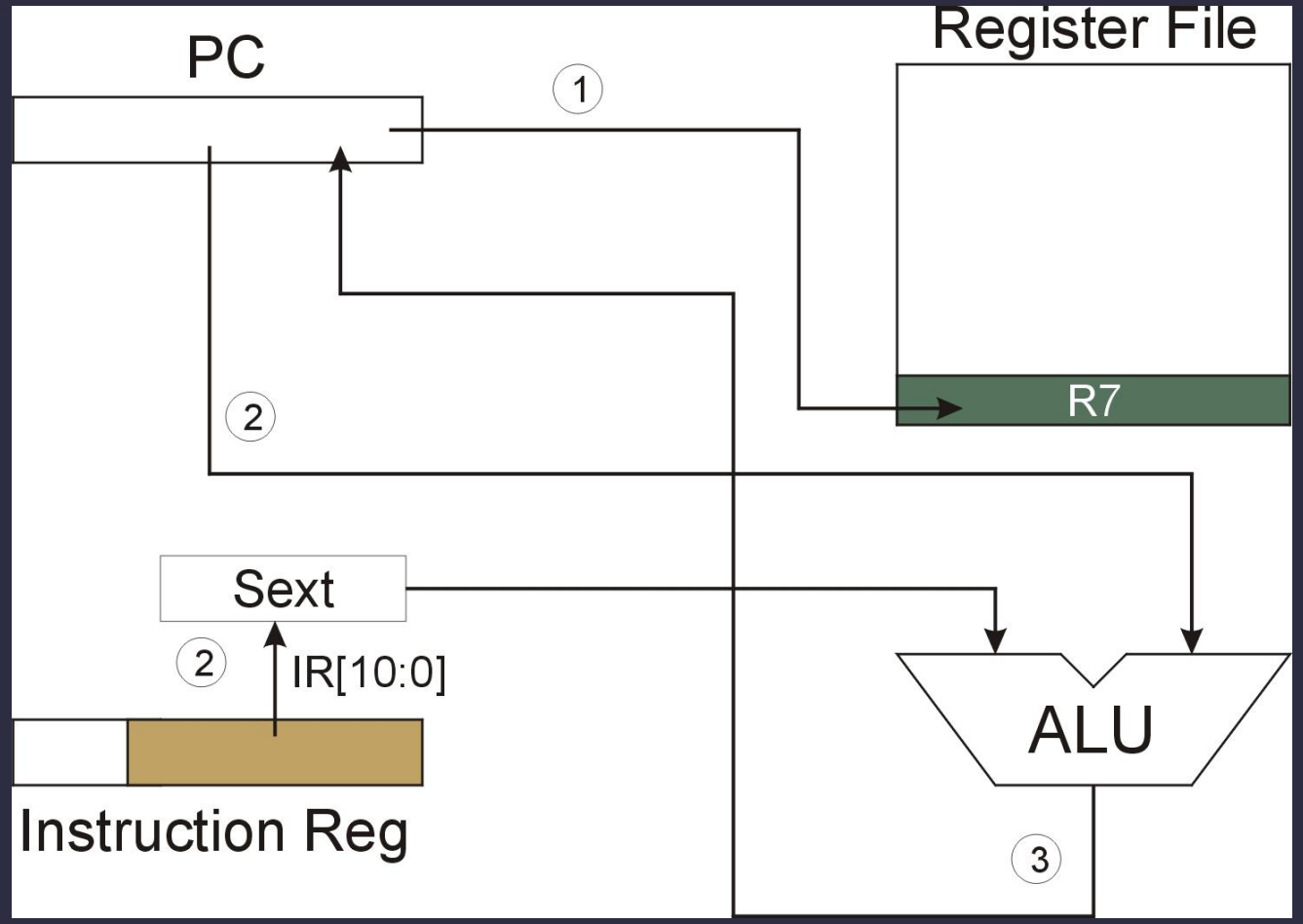
Sistem Fonksiyonu olmadan Fonksiyonlar

- Fonksiyonlar bir programın parçasıdır.
  - Kullanıcının program yazabildiği bellek adres aralığındadır.
  - Tanımlanmış bir görevi yapar
  - Diğer uygulamalar tarafından çağrılabilir
  - Fonksiyon tamamlandığında çağrıldığı yere geri döner
- Sistem fonksiyonları gibidirler, ancak işletim sisteminin bir parçası değillerdir.
  - Donanım kaynaklarının korunması gibi bir koruma önlemi alınmasına gerek yoktur.
  - Özel bir yetki gerek olmadan çalışabilirler
- Fonksiyon kullanımının gereklilikleri:
  - Tekrar kullanılabilir bir yapıdadır. Sürekli farklı yerlerden çağrılabilir
  - Büyük bir işi parçalarken, işin küçük parçaları fonksiyonlar olarak ifade edilebilir
  - Başka geliştiricilerin sağladığı kütüphane'lerin içerisindeki fonksiyonların kullanımı

JSR Komutu

- Koşulsuz olarak verilen adrese atlar ve şu anki PC'i R7 saklayıcısına yazar.
  - Geri dönülecek adresi kaydeder (R7 = PC)
  - PC'i komutun içerisindeki 11. bit olan adresleme moduna göre değiştirir.
    - Eğer 1 ise, PC-ilişkili: PC = PC + (IR[10:0])
    - Eğer 0 ise, PC = IR[8:6]

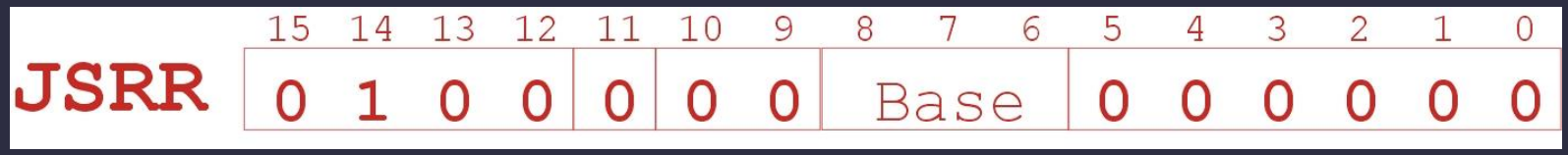
JSR



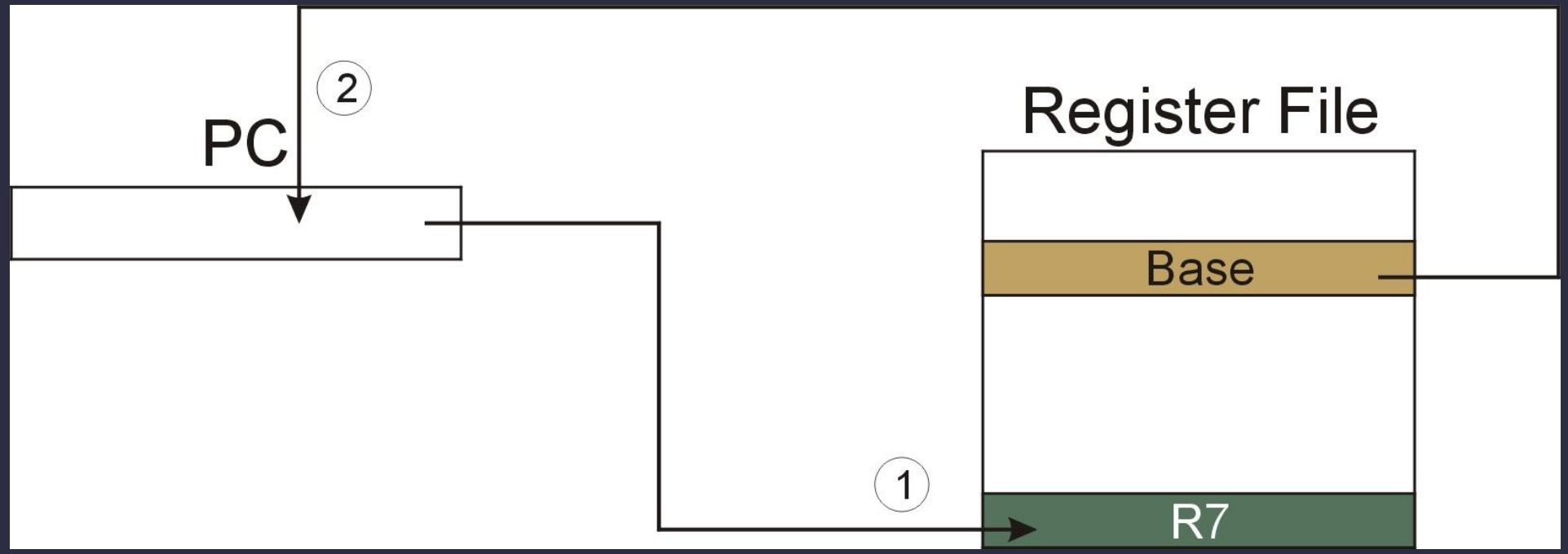
JSRR Komutu

- JSR komutu gibidir, adresleme modu farklıdır.
  - Hedef adres, saklayıcıdan gelmektedir

- Özelliği JSR'nin bit genlişliğinin, RAM'deki adresi ifade edemeyecek olduğu durumda işe yaramaktadır.



JSRR



Fonksiyondan Dönüş

- RET (JMP R7) fonksiyonun çağrıldığı yere döndürür.

Örnek: R0 içeriğinin tersinin alınması

```
2sComp  NOT  R0, R0      ;  Bitleri NOT'la
        ADD  R0, R0, #1      ;  1 ekle
        RET                  ;  çağrılan yere dön, RET için R7 saklayıcısı kullanıyor
```

```
; hesaplanmak istenen R4 = R1 - R3
        ADD  R0, R3, #0      ;  R0 = R3 + 0
        JSR  2sComp          ;  yukarıdaki fonksiyon'a atla, R7'e PC'i atadı
        ADD  R4, R1, R0      ;  R4 = R1 + R0 (aslında -R0)
        ...
```

Fonksiyonlara Argüman Gönderip, Değer Döndürme

- Argümanlar
  - Fonksiyona gönderilen değerlere argüman denmektedir.
  - Bunlar fonksiyonun hesap yapması için gerekli değerlerdir.
  - Örnek:
    - 2sComp fonksiyonunda, R0 giriş olarak alındı.
    - OUT servis fonksiyonunda, R0 giriş olarak alınıp, ekrana bastırılmaktadır.
    - PUTS servis fonksiyonunda, R0 giriş olarak alınıp, R0 adresindeki değerden başlanarak metin ekrana bastırılmaktadır.
- Dönüş Değeri
  - Fonksiyondan dönen değerledir.
  - Örnek:
    - 2sComp fonksiyonunda, R0 saklayıcısına hesaplamalar yapılmıştır.
    - GETC servis fonksiyonunda, klavyeden okunan değer, R0 saklayıcından döndürülmektedir.

Fonksiyonları Kullanma

- Bir programcı, fonksiyonu kullanabilmesi için;
  - Fonksiyonun adresini (Assembly dilinde, etiket (label)'ini)
  - İşlevini
  - Argümanlarını
  - Geri döndürdüğü değerleri (Hangi saklayıcıdan döndürecek ise)

  bilmesi gerekmektedir.

Saklayıcıları Kaydetme ve Geri Getirme

- Fonksiyonlar, aynı sistem fonksiyonları gibi olduğu için, saklayıcıların değerlerinin fonksiyon çağrılmadan kaydedilmesi gerekebilir.
- Genellikle, çağrılan fonksiyonda saklayıcıların değerlerinin kaydedilmesi uygulanır.
- Dolayısıyla programcı bu fonksiyonu kullanırken değer kaydetmek ile uğraşmaz
- R7 saklayıcısına ise fonksiyon çağrılmadan önce PC'nin değeri yazılmalıdır. Aksi taktirde geri dönülecek adres bulunamaz.

Kütüphane Fonksiyonları

- Diğer geliştiricilerin yazdığı fonksiyonların kullanımı
  - Kaynak kodunu (assembly) vermek istemiyor olabilirler.
  - Assembler/linker yazılımı, kütüphane fonksiyonunu sembol tablosuna bakarak, kullanılacak fonksiyonların etiketinin hangi satıra geldiğine bakar.

```
            ...
            .EXTERNAL FONKSIYON1
            ...
            LD   R2, SQAddr     ; Fonksiyon1 adresini yükler
            JSRR R2
            ...
SQAddr      .FILL     FONKSIYON1
```

Yığın (Stack): Soyut bir Veri Türü

- Soyutlama kavramı ile bir çok uygulamada karşılaşılmaktadır. Aslında olmayan (donanımda karşılığı olmayan) ancak var olan yapılar ile yeni bir kavram inşa edilmesidir.

- Yığınların genel olarak 3 kullanımı:

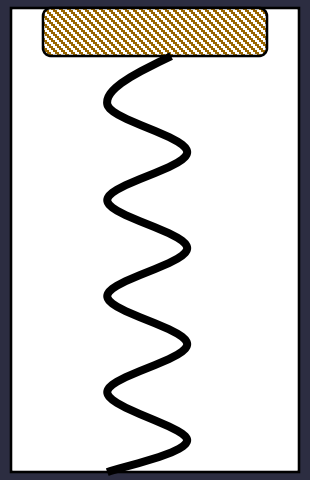
- Kesme (Interrupt) temelli giriş ve çıkışlar
- Aritmetik operasyonların saklanması
- Veri türü dönüşümleri (İkilik tümleyen'den ASCII dönüşümü)

Yığınlar

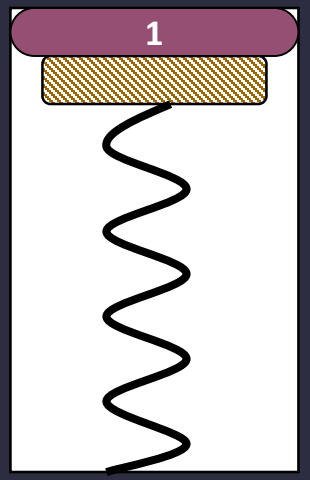
- LIFO (last-in first-out, Son Giren İlk Çıkar) depolama yapısıdır.
  - Kaydedilen ilk veri, en son çıkar.
  - Yığına kaydedilen son veri ise, ilk çıkar

- İki ana fonksiyonu vardır:
- PUSH: Yığına veri ekler
- POP: Yığından veri alır

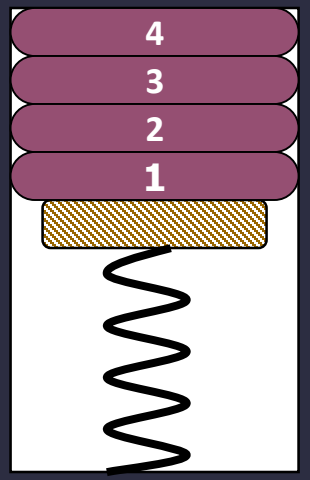
Yığın Örneği



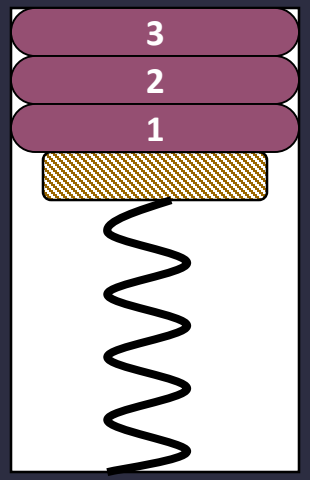
Başlangıç Durumu



Bir veri
eklendikten sonra



3 veri daha
eklendikten sonra



Bir veri
alındığında

Yazılım Gerçeklemesi

- Veriler hareket etmez, yığının başlangıç adresi değişir