BÖLÜM 6
İşaretçi, Yığın, ve Altprogram

Doç. Dr. Fatih Evran
Elektrik-Elektronik Mühendisliği
Ofis:421, Mühendislik Binası
Düzce Üniversitesi
Email: fatihevran@duzce.edu.tr

Doğrudan ve Dolaylı Adresleme Modları (1)

- Doğrudan adresleme modu: Komuttaki adres alanı operandın etkin adresini içerir.

A. mov f, Wa

f bir etkin adrestir: f bellek konumundan okunan değeri Wa' ya kopyala

B. mov Wa, Wb

Wa is bir etkin adrestir: Wa' dan değeri oku ve Wb' ye kopyala

---

- Dolaylı adresleme modu: Komuttaki adres alanı, operandın etkin adresinin bulunduğu bellek konumunu içerir.

A. mov [Wa], Wb

"[ ]" kullanırsak, dolaylı adres kullanıyoruz demektir. Bu durumda Wa, ara bellek adresini içerir

Adım 1: Wa' deki değer oku. Bu değer ara bellek adresi olarak kullanılır

Adım 2: Yukarıdaki ara bellek adresinden değeri okuyun ve ardından değeri Wb' ye kopyalayın

Doğrudan ve Dolaylı Adresleme Modları (2)

Register	Veri
W0	0x1000
W1	0x1002
W2	0x1004

Bellek Adresi	Veri
0x1000	0xA13D
0x1002	0x34D9
0x1004	0x226E

Doğrudan adresleme:

"mov W0, W1" yürütüldükten sonra, W1' deki yeni değer nedir?

W0' daki değeri W1' e kopyalayın. Yani W1' deki yeni değer 0x1000 olmaktadır

Dolaylı adresleme

"mov [W0], W1" yürütüldükten sonra, W1' deki yeni değer nedir?

Adım 1: W0' daki değeri okuyun. Bu değer, 0x1000, yani ara bellek adresi olmaktadır.

Adım 2: 0x1000 ara bellek adresinden değeri okuyun, ardından değeri W1' e kopyalayın. Yani W1'deki yeni değer 0xA13D olmaktadır.

Dolaylı Adresleme Modları (1)

A. mov [++Wa], Wb Adım 1: (Wa) = (Wa) + 2 Adım 2: Ara bellek adresi olarak Wa' deki değeri oku. Adım 3: Ara bellek adresinden değeri okuyun, ardından değeri Wb' ye kopyalayın.	B. mov [Wa++], Wb Adım 1: Ara bellek adresi olarak Wa' deki değeri oku. Adım 2: Ara bellek adresinden değeri okuyun, ardından değeri Wb' ye kopyalayın. Adım 3: (Wa) = (Wa) + 2
C. mov.b [++Wa], Wb Adım 1: (Wa) = (Wa) + 1 Adım 2: Ara bellek adresi olarak Wa' deki değeri oku. Adım 3: Ara bellek adresinden değeri okuyun, ardından değeri Wb' nin LSB kısmına kopyalayın.	D. mov.b [Wa++], Wb Adım 1: Ara bellek adresi olarak Wa' deki değeri oku. Adım 2: Ara bellek adresinden değeri okuyun, ardından değeri Wb' nin LSB kısmına kopyalayın. Adım 3: (Wa) = (Wa) + 1

Dolaylı Adresleme Modları (2)

E. mov [Wa + Wb], Wc Adım 1: Ara bellek adresi olarak (Wa+Wb)' deki değeri oku. Adım 2: Ara bellek adresinden değeri okuyun, ardından değeri Wc' ye kopyalayın
F. mov.b [Wa + Wb], Wc Adım 1: Ara bellek adresi olarak (Wa+Wb)' deki değeri oku. Adım 2: Ara bellek adresinden değeri okuyun, ardından değeri Wc' nin LSB kısmına kopyalayın

Dolaylı Adresleme Mod Örnekleri (1)

Register	Veri
W0	0x1000
W1	0x1002
W2	0x1004

Bellek Adresi	Veri
0x1000	0xA13D
0x1002	0x34D9
0x1004	0x226E

"mov [++W0], W1" yürütüldükten sonra, W0 ve W1' deki yeni değer nedir?

Adım 1: (W0) = (W0) + 2 → W0 = 0x1000 + 2 = 0x1002

Adım 2: Ara bellek adresi → Ara bellek adresi = 0x1002
olarak W0' daki değeri oku.

Adım 3: Ara bellek adresinden → Ara bellek adresindeki (0x1002) değeri W1' e kopyala
değeri okuyun, ardından değeri
W1' e kopyalayın.

↓

W1 = 0x34D9

Dolaylı Adresleme Mod Örnekleri (2)

Register	Veri
W0	0x1000
W1	0x1002
W2	0x1004

Bellek Adresi	Veri
0x1000	0xA13D
0x1002	0x34D9
0x1004	0x226E

"mov [W0++], W1" yürütüldükten sonra, W0 ve W1' deki yeni değer nedir?

Adım 1: Ara bellek adresi olarak W0' daki değeri oku → Ara bellek adresi = 0x1000

Adım 2: Ara bellek adresinden değeri okuyun, ardından değeri W1' ye kopyalayın → Ara bellek adresindeki (0x1000) değeri W1' e kopyala

↓

W1 = 0xA13D

Adım 3: (W0) = (W0) + 2 → W0 = 0x1000 + 2 = 0x1002

Dolaylı Adresleme Mod Örnekleri (3)

Register	Value
W0	0x1000
W1	0x1002
W2	0x1004

Memory	Value
0x1000	0x A1 3D
0x1002	0x34D9
0x1004	0x226E

"mov.b [++W0], W1" yürütüldükten sonra, W0 ve W1' deki yeni değer nedir?

Adım 1: (W0) = (W0) + 1 → W0 = 0x1000 + 1 = 0x1001

Adım 2: Ara bellek adresi → Ara bellek adresi = 0x1001
olarak W0' daki değeri oku

Adım 3: Ara bellek adresinden değeri okuyun, ardından değeri W1' nin LSB kısmına kopyalayın → Ara bellek adresindeki (0x1001) değeri W1' nin LSB kısmına kopyala

W1 = 0x10A1

Dolaylı Adresleme Mod Örnekleri (4)

Register	Value
W0	0x1000
W1	0x1002
W2	0x1004

Memory	Value
0x1000	0xA13D
0x1002	0x34D9
0x1004	0x226E

"mov.b [W0++], W1" yürütüldükten sonra, W0 ve W1' deki yeni değer nedir?

Adım 1: Ara bellek adresi olarak W0' daki değeri oku → Ara bellek adresi = 0x1000

Adım 2: Ara bellek adresinden değeri okuyun, ardından değeri W1' nin LSB kısmına kopyalayın → Ara bellek adresindeki (0x1000) değeri W1' nin LSB kısmına kopyala

W1 = 0x103D

Adım 3: (W0) = (W0) + 1 → W0 = 0x1000 + 1 = 0x1001

Değişkenlerin Adresi

- C' de bir değişkenin adresini almak için "&" kullanırız.

Değişken ismi	Bellek adresi	Değer
u16_i	0x1000	0xA13D

&u16_i = u16_i adresi=0x1000

Assembly dilinde, bir değişkenin adresini almak için "#" kullanırız, neden?

↓

Aslında bir değişkenin adı, bu değişkenin başlangıç adresidir.

↓

Assembly dilinde, "#" ile başlayan sayı değişmez olarak kabul edilir.

↓

#değişken_ismi = bu değişkenin başlangıç adresidir.

↓

C dilindeki &u16_i assembly dilindeki #u18_i ile eşdeğerdir

C Dilinde İşaretçiler (Pointers) (1)

- Bir işaretçi değişkeni, başka bir değişkenin adresini içerir
- İşaretçi, dizileri alt programlara geçirebilir
  - Adım 1: İşaretçi, dizinin ilk adresini alt programa geçirir
  - Adım 2: Dizideki herhangi bir elemanın adresi= Başlangıç adresi + ofset
- PIC' de bellek adresi 16 bittir. Bu nedenle, bir işaretçi değişkeni 8/16/32 bit veriye işaret etse de 16 bitlik bir değere sahiptir.
  - Örneğin pu8_i, pu16_j, pu32_k, pi8_o, pi16_p, pi32_q farklı veri türlerine işaret eder, ancak bu işaretçilerin tümü 16 bitlik bir değere sahiptir.

C Dilinde İşaretçiler (2)

Değişken ismi	Bellek	Veri
u16_i	0x1000	0xA13D
u32_j	0x1002	0x31EF
	0x1004	0x1D2B

```
uint16 u16_i, u16_x;
uint32 u32_j, u32_y;
uint16* pu16_a;
uint32* pu32_b;
```

`pu16_a = &u16_i;` → pu16_a = u16_i' nin başlangıç adresi= 0x1000

`u16_x = *pu16_a;` → u16_x = pu16_a ile gösterilen değer = 0x1000 adresindeki 16 bitlik değer = 0xA13D

`pu32_b = &u32_j;` → pu32_b = u32_j' nin başlangıç adresi = 0x1002

`u32_y = *pu32_b;` → u32_y = pu32_b ile gösterilen değer = 0x1002 adresinden itibaren 32 bitlik sayı = 0x1D2B 31EF

Assembly Dilinde İşaretçi Kullanımı

C kodu	Assembly kodu
uint16 u16_i, u16_j; uint16* pu16_a	u16_i: .space 2 u16_j: .space 2
u16_i = 0xA245; →	mov #0xA245, W0 mov WREG, u16_i
pu16_a = &u16_j; →	mov #u16_j, W1
u16_k = *pu16_a; →	mov [W1], W0, mov WREG, u16_k

Assembly dili işaretçileri yerleştirmek için dolaylı adreslemeyi kullanır

İşaretçi Aritmetiği (1)

- İşaretçi aritmetiği, işaretçiye bir değerini eklemek veya işaretçiden bir değerini çıkarmak anlamına gelir
- İşaretçi artırma: pointer++, pointer = pointer + 1 ile eşdeğerdir
- İşaretçi azaltma: pointer--, pointer = pointer – 1 ile eşdeğerdir
- İşaretçiye eklenen veya işaretçiden çıkarılan değer, işaretçinin gösterdiği veri türünün byte cinsinden boyutuyla çarpılır

Örnek:	Değişken ismi	Bellek	Veri
	u8_i	0x1000	0xA13D

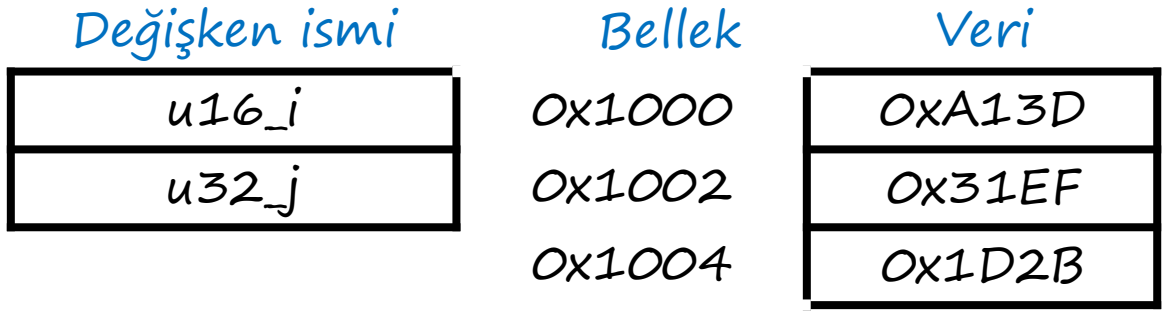
pu8_a = &u8_i; → pu8_a = u8_i' nin adresi = 0x1000

pu8_a = pu8_a +1; →

pu8_a tarafından ulaşılan değişken, 1 byte olan u8_i' dir.

pu8_a +1 = 0x1000 + 1 x 1 byte = 0x1001

İşaretçi Aritmetiği (2)



`pu16_a = &u16_i;` pu16_a = u16_i' nin başlangıç adresi = 0x1000

`pu16_a = pu16_a + 1;` pu16_a tarafından uluşılan değişken is 2 byte olan u16_i olmaktadır

pu16_a + 1 = 0x1000 + 1 x 2 byte = 0x1002

`pu32_b = &u32_j;` pu32_b = u32_j ' nin başlangıç adresi = 0x1002

`pu32_b = pu32_b + 3;` pu32_b tarafından ulaşılan değişken 4 byte olan u32_j olmaktadır

pu32_b + 3 = 0x1002 + 3 x 4 byte = 0x100E

C Dilinde Dizi

- Bir dizi, bir tamsayı dizisi, karakter dizisi vb. gibi benzer veri türlerinden meydana gelmektedir.
- Bir dizinin tanımlanması: veri türü dizi ismi [eleman sayısı]

Örnek 1: uint8 au8_x[4]

Bu dizinin ismi au8_x olmaktadır. Bu dizide toplam 4 eleman bulunmaktadır. Her bir eleman 8-bit işaretsiz bir tamsayıdır.

Örnek 2: uint32 au32_y[5]

Bu dizinin ismi au32_y olmaktadır. Bu dizide toplam 5 eleman bulunmaktadır. Her bir eleman 32-bit işaretli bir tamsayıdır.

Bir Dizinin Hafızada Gösterilmesi

uint8 au8_x[4] = {0x3E, 0xAB, 0x72, 0x1F}	uint16 au16_y[2] = {0xAE24, 0x25C3}
uint32 au16_z[2] = {0x3A5B 43CF, 0x67AC 9D9E}	

Bellek	Veri
0x1000	0xAB3E
0x1002	0x1F72
0x1004	0xAE24
0x1006	0x25C3
0x1008	0x43CF
0x100A	0x3A5B
0x100C	0x9D9E
0x100E	0x67AC

Değişkenler	
au8_x[1]	au8_x[0]
au8_x[3]	au8_x[2]
au16_y[0]	
au16_y[1]	
au32_z[0].LSW	
au32_z[0].MSW	
au32_z[1].LSW	
au32_z[1].MSW	

Dizi Adresinin Bulunması

- uint8 au8_x[4] = {0X3E, 0xAB, 0X72, 0X1F}

Bellek	Veri	Değişkenler	
0X1000	0xAB3E	au8_x[1]	au8_x[0]
0X1002	0X1F72	au8_x[3]	au8_x[2]

Örnek:

au8_x, 0X1000 değerinde olan dizideki başlangıç adresidir

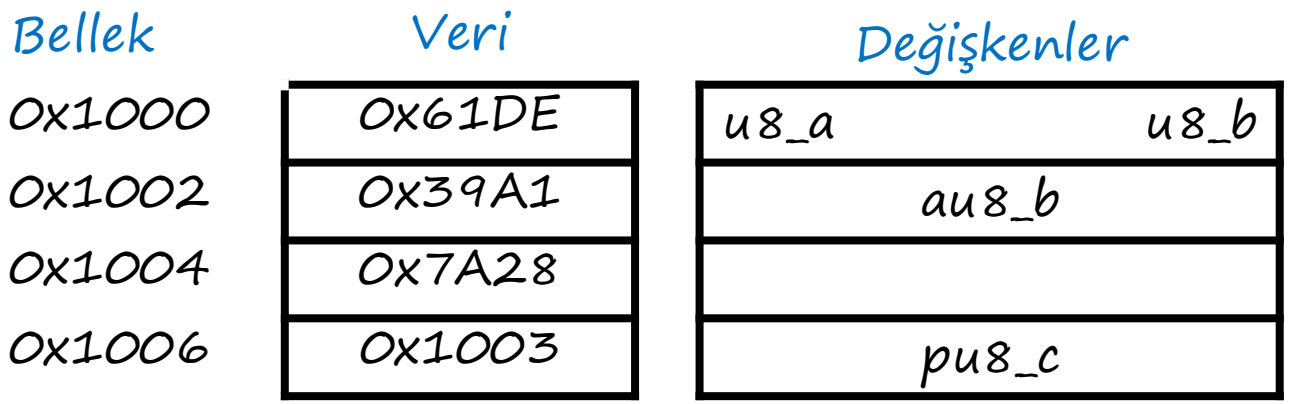
au8_x + 1, 0X1001 değerinde olan dizideki ikinci elemanın adresidir

au8_x + 2, 0X1002 değerinde olan dizideki üçüncü elemanın adresidir

au8_x[2], dizideki ikinci elemanın değeridir. Bu elemanın değeri 0X72 olmaktadır

&au8_x[2], 0X1002 değerinde olan dizideki üçüncü elemanın adresidir

Örnekler-Devam (1)



u8_a = 0x61

&u8_a = 0x1001

au8_b = 0x1002

au8_b + 1 = 0x1003

au8_b [1] = 0x39

pu8_c = 0x1003

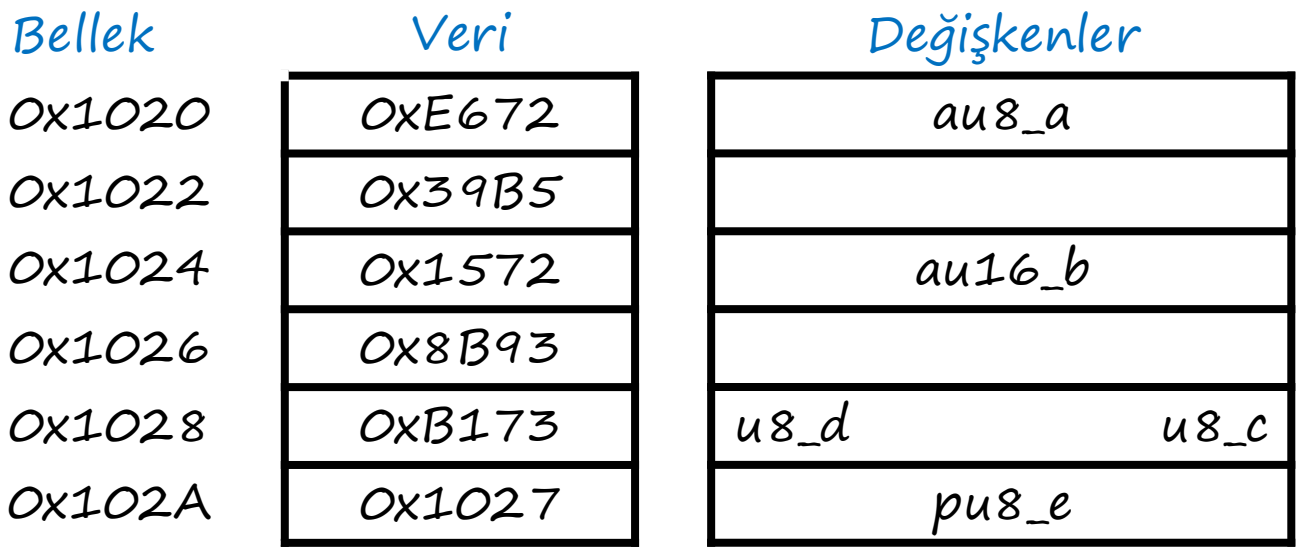
&pu8_c = 0x1006

*pu8_c = 0x39

pu8_c[1] = 0x28

pu8_c + 2 = 0x1005

Örnekler-Devam (2)



u8_d = 0xB1

&u8_d = 0x1029

au16_b = 0x1024

&au16_b [1] = 0x1026

au8_a +1 = 0x1021

au8_a[3]= 0x39

pu8_e = 0x1027

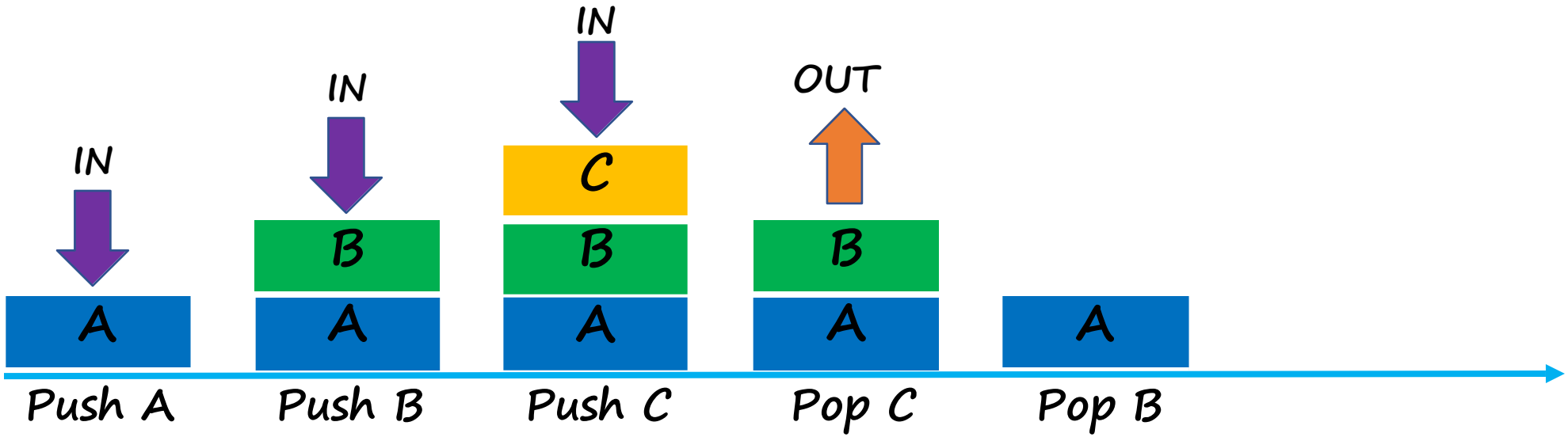
&pu8_e = 0x102A

*pu8_e = 0x8B

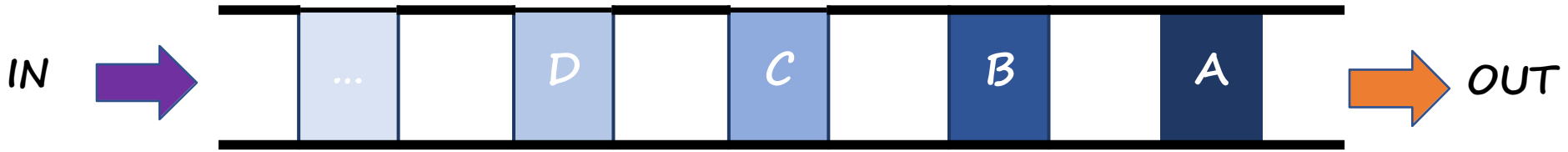
pu8_e + 2 = 0x1029

Yığın (Stack)

- Veri depolamak için kullanılabilecek birkaç yapı vardır
  - A. Yığın (Stack): Son Giren İlk Çıkar (Last In, First Out - LIFO)
  - B. Kuyruk (Queue): İlk Giren İlk Çıkar (First In, First Out- FIFO)



Yığın: Veri çıkışı ve girişi aynı yerde olmaktadır



Kuyruk: Veri çıkışı ve girişi farklı yerde olmaktadır

Yığın Hafızada Push ve Pop Komutlarının Çalışması

- W15, yığın işaretçisidir (stack pointer – SP)
- Push işlemi: push src (burada src saklayıcı ya da bellek adresi olabilir)

Önemli !

Adım 1: src' deki veri W15' in gösterdiği yığına itilir.
Adım 2: W15 = W15 + 2 (SP yığının bir sonraki konumunu işaret eder)

- Pop işlemi: pop dest (burada src saklayıcı ya da bellek adresi olabilir)

Adım 1: W15 = W15 – 2 (SP yığının bir önceki konumunu işaret eder)
Adım 2: W15' in işaret ettiği yığındaki veriyi hedefe (dst) çıkar (aktar)

Önemli !

Örnek – Push Komutu

Push W1

Register	Veri
W1	0x33AB
W15	0x1002

Bellek	Veri
0x1000	0xAB3E
0x1002	0x1F72

Adım 1: W1' deki değer W15' in gösterdiği yığına itilmektedir.

↓

W15, 0x1002 bellek adresine işaret etmektedir.

↓

W1'deki veri 0x1002 belleğine itilmektedir.

↓

0x1002 bellekteki değer 0x33AB olarak değiştirilmektedir.

↓

Adım 2: W15 = W15 + 2 (SP yığının bir sonraki konumunu işaret eder)

↓

W15' deki değer = 0x1002 + 2 = 0x1004

Örnek – Pop Komutu

Pop W1

Register	Veri
W1	0x33AB
W15	0x1002

Bellek	Veri
0x1000	0xAB3E
0x1002	0x1F72

Adım 1: W15 = W15 – 2, (SP yığının bir önceki konumunu işaret eder)

↓

W15 = 0x1002 – 2 = 0x1000

↓

Adım 2: W15'in işaret ettiği yığındaki veriyi W1'ye aktar.

↓

W15'in işaret ettiği hafıza adresi = 0x1000

↓

0x1000 bellek adresindeki veriyi W1'e aktar.

↓

W1 = 0xAB3E

C Dilinde Altprogramlar

- C dilinde altprogramların genel biçimi:

```
(Veri Tipi) altprogram ismi (parametreler)
{
   local variable declaim;
   subroutine stmt1;
   subroutine stmt2;
   ...
   return value;
}
```

Write a subroutine to count how many ones in a variable

```
uint8 countOnes (uint16 u16_v)
{
   uint8 u8_cnt, u8_i;
   u8_cnt = 0;
   for (u8_i = 0; u8_cnt < 16; u8_i++)
     {
       if (u16_v & 0x0001) u8_cnt++;
       u16_v = u16_v >> 1;
     }
   return u8_cnt;
}
main (void)
{
    uint16 u16_k;
    u16_k = 0xA501;
    u8_j = countOnes (u16_k);
}
```

Assembly Dilinde Altprogramlar

- Bir alt programı çağırmak için hem "call" hem de "rcall" komutları kullanılabilir.

  Adım 1: Bu komutlar yığına dönüş adresini gönderilir. Geri dönüş adresi, call/recall komutundan sonraki komutun adresidir.

  Adım 2: Altprograma şartsız bir dallanma gerçekleştirilir

- "return" komutu bir altprogramdan dönüş için kullanılır.

  Step 1: Bu komut yığının en üstünde bulunan dönüş adresini program sayıcına aktarır.

rcall ve call (1)

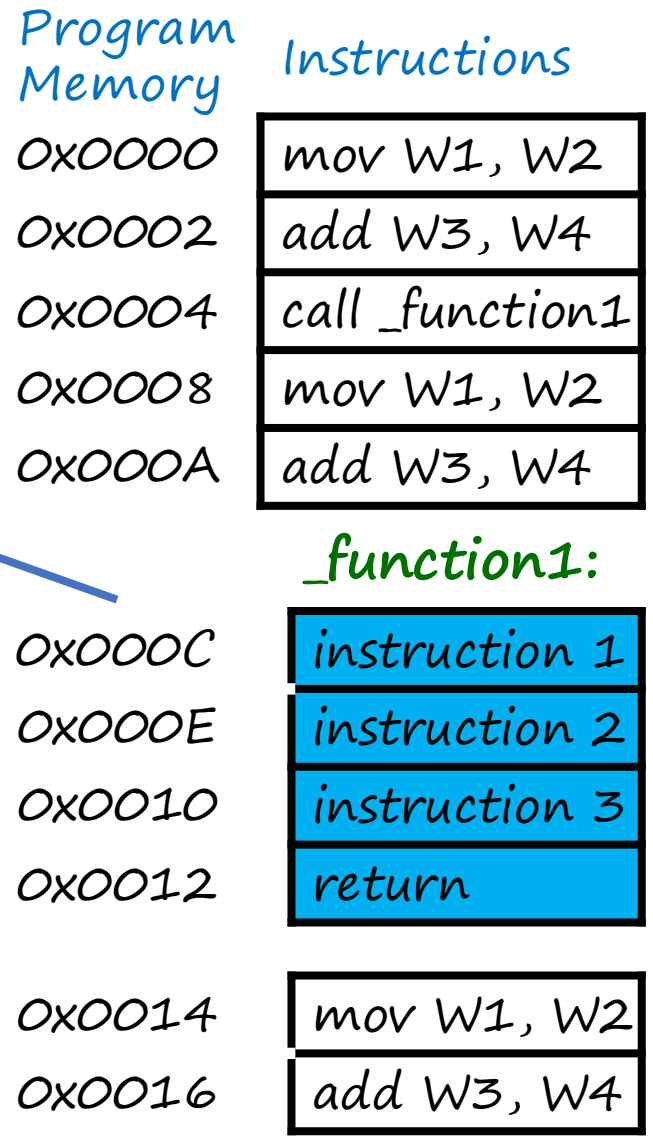
- call: PC, fonksiyondaki (altprogram) ilk komutun mutlak adresini (24 bit) yükler

Altprogram çağrısından sonra: PC = 0x000C

- rcall: PC = "PC + ofset (16 bits)"

  - fonksiyondaki ilk komut ile mevcut adres arasındaki adres ofseti

Relative address (ofset)= (0x000C -0x0004)



rcall ve call (2)

- The call komutu 2 kelime (word) uzunluğundadır
- The rcall komut 1 kelime uzunluğundadır
- Rcall komutunda ofset sadece 16-bittir.

rcall komutu call komutundan daha hızlıdır

Alt program geçerli adresten uzaktaysa

Fonksiyondaki ilk komut ile mevcut adres arasındaki ofset $2^{16} - 1$' den büyük olabilir

Yanlış bir komut yükler

Altprograma Parametre Geçişi

- Soldan sağa sırayla yani artan saklayıcı numaralarında, W0-W7 arasındaki saklayıcılar parametreleri geçirmek için kullanılır
- Bir altprogramı çalıştırmadan önce, altprogram W0-W14 arasındaki saklayıcıları kullanması gerekiyorsa, bu saklayıcılar yığına kaydedilmelidir.
  A. W0-W7 çağıran (caller) tarafından kaydedilir.
  B. W8-W15 çağrılan (callee) tarafından kaydedilir
- Fonksiyonun dönen değerleri: W0-W3
  A. 8-bitlik dönen değer: W0
  B. 16-bitlik dönen değer: W0
  C. 32-bitik dönen değer: W0-W1
  D. 64-bitlik dönen değer: W0-W3
  E. Birden fazla dönen değer:
     İşaretçi üzerinden başlangıç adresi döner.

Örnek

```
mov W1, W2
add W1, W2
push W1              ;save W1
push  W2             ;save W2
mov i, WREG          ;pass i to W0
call _function1      ;call function
mov.b WREG, u8_j     ;save return value
pop W2               ;restore W2
pop W1               ;restore W1
```

Altprogram Örneği

```
;; main
mov #0xA501, W0
mov WREG, _u16_k
;; Subroutine call
mov _u16_k, WREG    ;pass parameter
rcall countOnes
mov.b WREG, _u8_j   ;get return value
done bra done       ;do not return
countOnes:
   clr.b W1         ;use W1 to count ones
   clr.b W2         ;use W2 as loop counter u8_i
   loop_top:
     cp.b W2, #16.  ;test u8_i >=16
     bra GEU, end_loop
     and #0x0001, W0 ;test last bit of u16_k is 1 or not
     brz Z, end_if
     inc.b W1, W1 ; ones counter++
   end_if:
     lsr W0, W0     ;16_k >> 1, to test next bit
     inc.b W2, W2   ;Loop counter++
     bra loop_top
   end_loop:
   mov.b W1, W0
   return
```

Write a subroutine to count how many ones in a variable

```
uint8 countOnes (uint16 u16_v)
{
   uint8 u8_cnt, u8_i;
   u8_cnt = 0;
   for (u8_i = 0; u8_cnt < 16; u8_i++)
     {
       if (u16_v & 0x0001) u8_cnt++;
       u16_v = u16_v >> 1;
     }
   return u8_cnt;
}
main (void)
{
    uint16 u16_k;
    u16_k = 0xA501;
    u8_j = countOnes (u16_k);
}
```