BÖLÜM 3
PIC Mikrodenetleyiciye Giriş

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

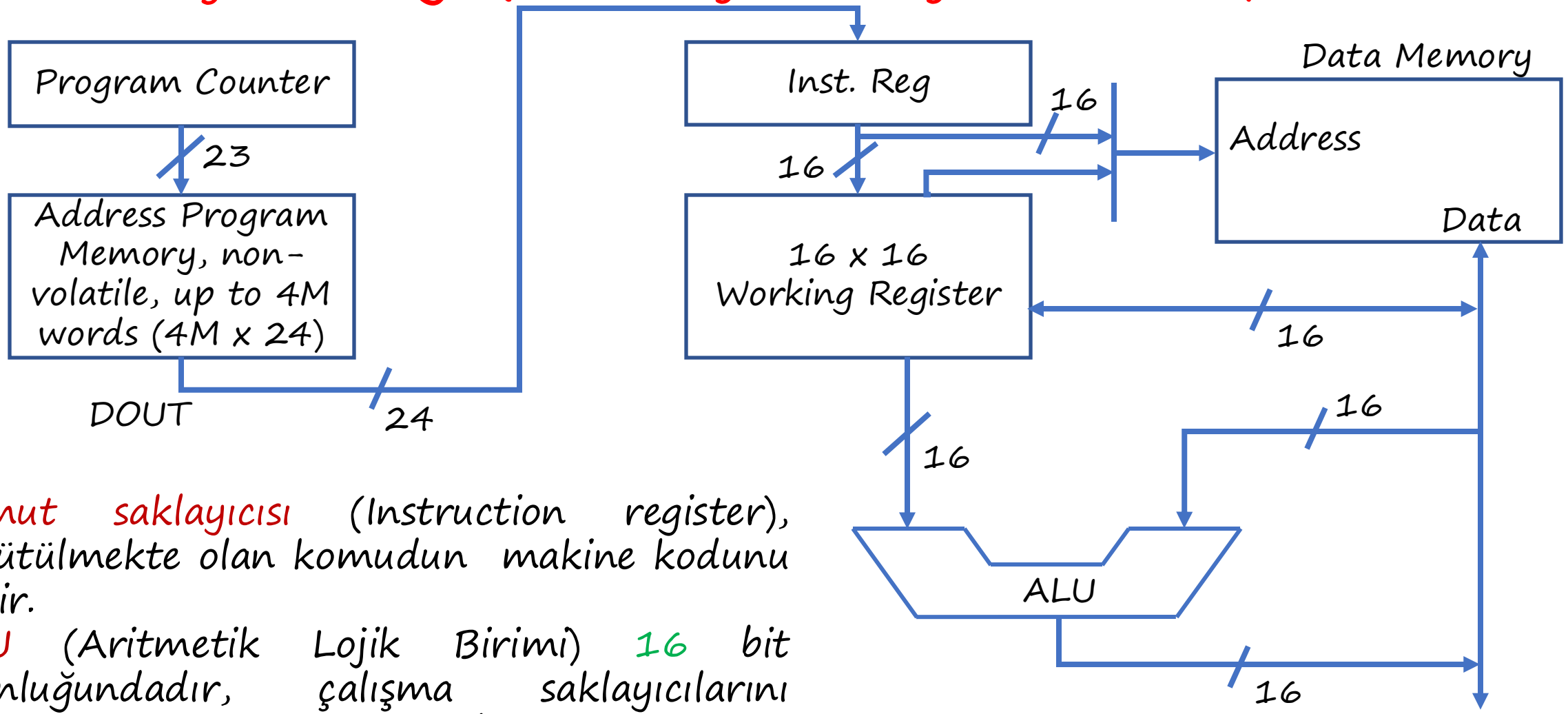
Mikrodenetleyici (µC) ve Mikroişlemci (µP)

- µC tek bir çip çözümü olarak tasarlanmıştır, µP harici çip (bellek, arabirim) gerektirir.
- µC program depolama için çip üzerinde uçucu olmayan belleğe sahiptir, µP' de çip üzeri bellek yoktur.
- µC, µP'den daha fazla çevresel arabirime (seri arabirimler, ADC, zamanlayıcılar gibi) sahiptir.
- µC'nin sanal bellek desteği yoktur (yani Linux'u çalıştıramaz), µP'nin ise vardır.
- Genel amaçlı µP'ler tipik olarak µC'lerden daha yüksek performansa sahiptir (saat hızı, veri genişliği, komut seti, önbellek gibi).
- µP'ler ve µC'ler arasındaki farklılık giderek bulanıklaşıyor.

Microchip PIC24 Ailesi

Özellikler	Açıklama
Instruction width (Komut uzunluğu)	24 bit
Çip üzeri program belleği (uçucu değildir, elektrikle silinebilir)	dsPIC33EP128GP502 32K byte program belleğine sahiptir, mimari 4M komut saklayabilir.
Çip üzeri RAM belleği (uçucudur)	dsPIC33EP128GP502, 16K (16384) byte veri hafızaya sahiptir, mimari 65536 byte hafızayı destekler
Saat freakansı	70 MHz
16-bit Mimari	Genel amaçlı saklayıcı, 84 komut (alt komutlar dahil değildir)
Çevresel arabirimler	Çeşitli seri haberleşme birimleri (USART, SPI, I2C),), timer devreler, ADC gibi

PIC24 Çekirdeği (Basitleştirilmiş Gösterim)



Komut saklayıcısı (Instruction register), yürütülmekte olan komudun makine kodunu içerir.

ALU (Aritmetik Lojik Birimi) 16 bit uzunluğundadır, çalışma saklayıcılarını (Working register) veya veri belleğini operand olarak kabul eder.

17 X 17 çarpım birimi gösterilmemiştir.

BELLEK ORGANİZYONU

Hatırlatma: 2^{10}=1K=1024

- PIC33/PIC24 µC üzerindeki bellek iki türe ayrılmıştır:
- Program Belleği:
  A. Program belleği komutları depolar
  B. Program belleği uçucu değildir (güç kesildiğinde içeriği korunur)
  C. Kapasite: 4Mx24-bit. Her bir komut 24 bit (3 byte) uzunluğundadır ve bellek 4M komutu saklayabilir
  D. PC (program counter, program sayıcı) 23 bit uzunluğundadır, ancak komutlar çift kelime sınırlarında başlar (yani PC'nin LSB biti her zaman sıfırdır), böylece PC 4M komut adresleyebilir.

İşlem kodu (opcode): 8-bit
Veri ya da bellek adresi: 16-bit

- Veri Belleği:
  A. Veri belleği veriyi depolar
  B. Veri belleği uçudur (güç kesildiğinde içeriği kaybolur)
  C. Kapasite : 65536x8-bit. Her bir veri 8 bit (1 byte) uzunluğundadır ve bellek 65536 byte (64K byte) veri saklayabilir —> Dolayısıyla veri belleğin adres yolu genişliği 16 bit uzunluğundadır

1 Byte Uzunluktan Fazla Olan Verilerin Saklanması (1)

Veri belleği kapasitesi: 65536 x 8 —> Her bellek adresi yalnızca 1 byte uzunluğunda veri saklayabilir

Peki, 1 byte uzunluktan fazla olan verileri nasıl saklayabiliriz?

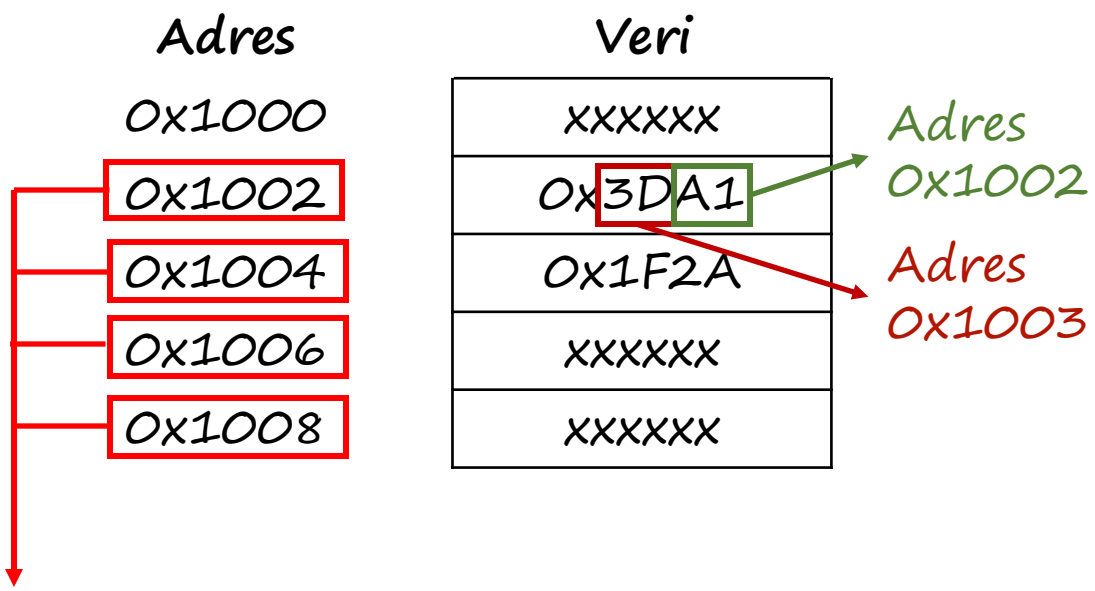
Veri, artan veri adreslerinde, adresin düşük anlamlı byte (least significant byte, LSB) değerinden adresin yüksek anlamlı byte (most significant byte, MSB) değerine doğru olacak biçimde bellekte sıralanır.

1 Byte Uzunluktan Fazla Olan Verilerin Saklanması (2)

Örnek: iki adet 16 bitlik değerler:
0x3DA1 ve 0x1F2A

Yüksek anlamlı Byte (MSB)

Düşük anlamlı Byte (LSB)



Örnek: iki adet 32 bitlik değerler:
0x3DA1 135A ve 0x9E6C 25E3

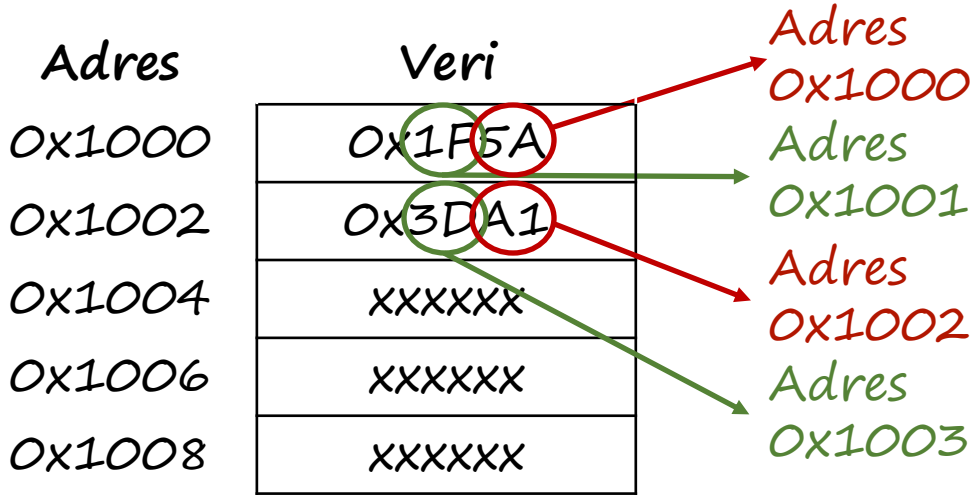
Adres	Veri
0x1000	0x135A
0x1002	0x3DA1
0x1004	0x25E3
0x1006	0x9E6C
0x1008	xxxxxx

Önemli: Örnekteki adres dizini yalnızca Düşük anlamlı byte adresini göstermektedir. Yüksek anlamlı byte adresi ise LSB adresi + 1 olmaktadır.

1 Byte Uzunluktan Fazla Olan Verilerin Saklanması (3)

- 16 bitlik veya 32 bitlik bir değerin düşük anlamlı byte kısmına ait adres çift olmalıdır.
- Birden fazla 8 bitlik değer varsa, bunlar nasıl saklayabiliriz?

Örnek: 4 adet 8 bitlik değerler:
0x3D, 0xA1, 0x1F, ve 0x5A



MOV Komutu (Saklayıcılar Arasında Transfer) (1)

- MOV komutu kaynak saklayıcının içeriğini hedef saklayıcıya kopyalamaktadır.
  A. Kaynak saklayıcı W0 ila W15 arasında olan 16 çalışma saklayıcısından biri olabilir.
  B. Hedef saklayıcı W0 ila W15 arasında olan 16 çalışma saklayıcısından biri olabilir.
  C. mov.b —> byte mod mov —> word mod

Word mod: mov W2, W1 W2'deki tüm 16 bitlik değeri W1'e kopyalar

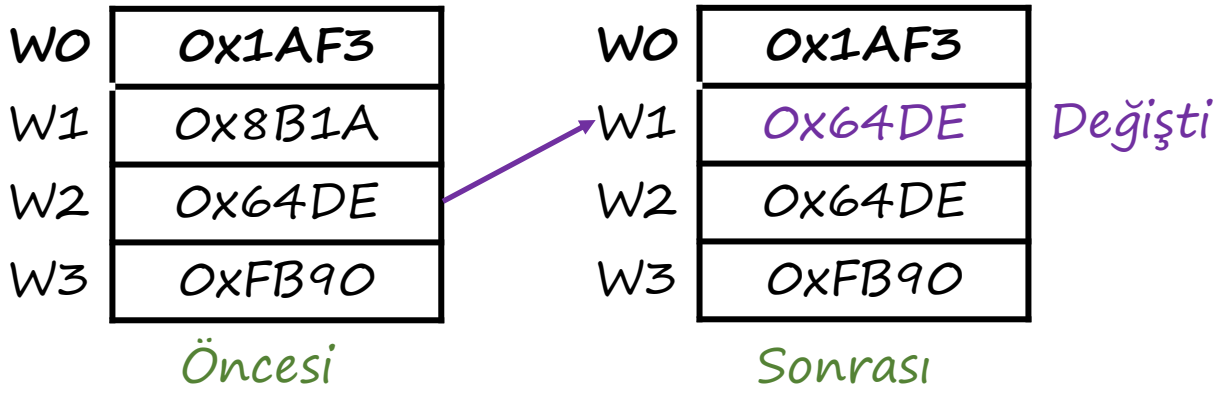
Byte mod: mov.b W2, W1 W2 saklayıcısının düşük anlamlı byte kısmını W1 saklayıcısının düşük anlamlı byte kısmına kopyalar. W1' deki yüksek anlamlı byte (MSB) ise değişmeden kalır

MOV Komutu (Saklayıcılar Arasında Transfer) (2)

Word mod: mov W2, W1 — W2'deki tüm 16 bitlik değeri W1'e kopyalar

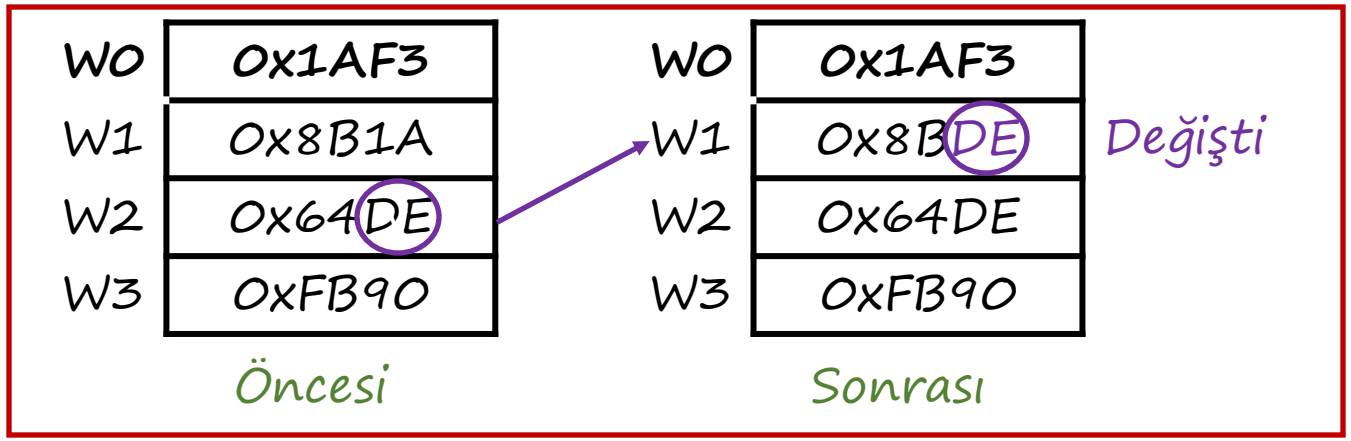
Byte mod: mov.b W2, W1. — W2 saklayıcısının düşük anlamlı byte kısmını W1 saklayıcısının düşük anlamlı byte kısmına kopyalar. W1'deki yüksek anlamlı byte (MSB) ise değişmeden kalır

a) mov W2, W1 (Word mod)



b) mov.b W2, W1 (Byte mod)

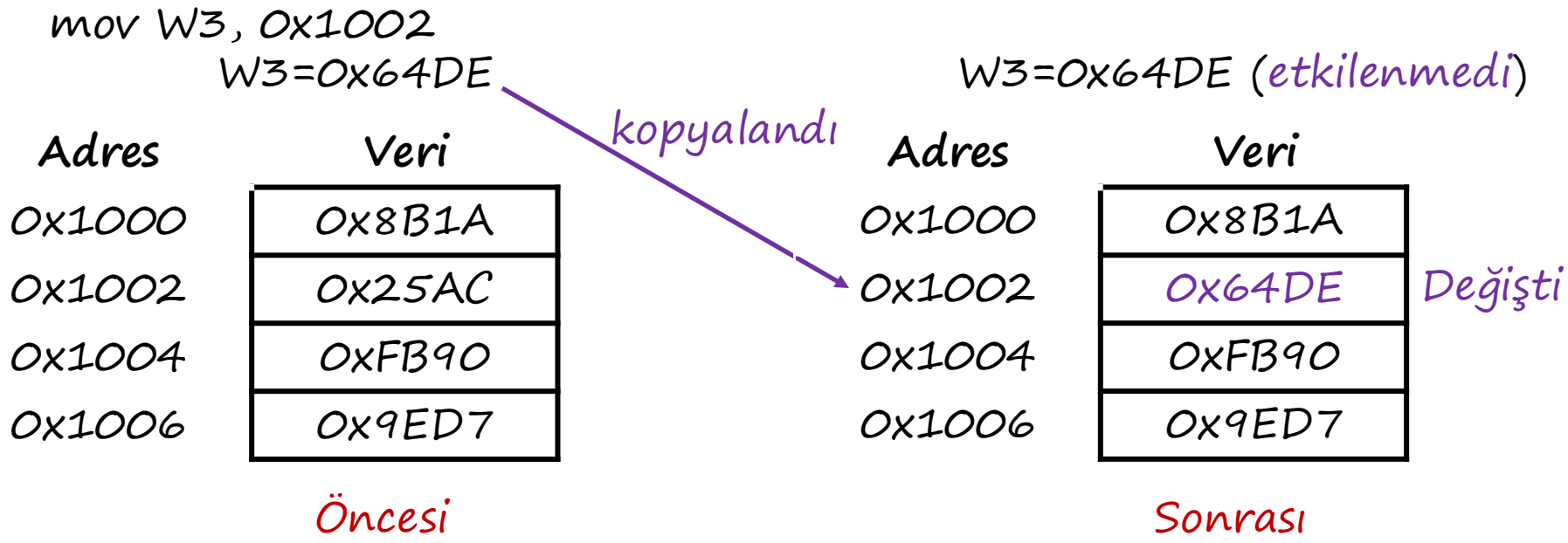
Önemli!



MOV Komutu (Saklayıcıdan Veri Belleğine Transfer)

- Mov komutu kaynak saklayıcı içeriğini veri bellek adresine kopyalar.
  A. Kaynak saklayıcı W1 ila W15 arasında olan 15 çalışma saklayıcısından biri olabilir
  B. Hedef veri bellekte bir bellek adresidir
  C. Yalnızca WORD modunu destekler, bu nedenle bellek adresi çift olmalıdır.

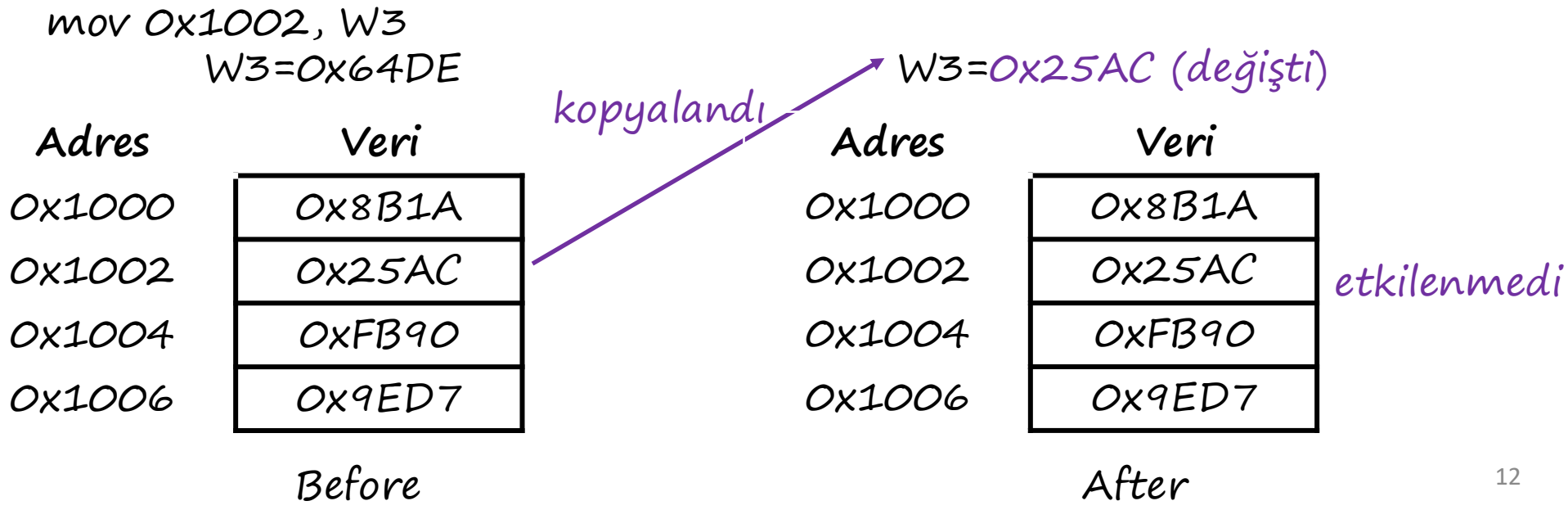
Örnek: mov W3, 0x1002 W3'teki tüm 16 bitlik verileri 0x1002 ile belirtilen veri belleği adresine kopyalar



MOV Komutu (Veri Belleğinden Saklayıcıya Transfer)

- Mov komutu veri belleği adresinin içeriğini hedef saklayıcıya kopyalar.
  A. Kaynak veri bellekte bir bellek adresidir.
  B. Hedef saklayıcı W1 ile W15 arasında olan 15 çalışma saklayıcısından biri olabilir.
  C. Yalnızca WORD modunu destekler, bu nedenle bellek adresi çift olmalıdır.

Örnek: mov 0x1002, W3 0x1002 ile belirtilen veri belleği adresindeki tüm 16 bitlik verileri W3' e kopyalar



Temel Saklayıcı (W0) Kullanan MOV Komutu

- W0 PIC' deki temel saklayıcının ismidir. Bu saklayıcının diğer ismi ise: **WREG**
- W0 ve **WREG** aynı saklayıcı olmasına rağmen bu iki saklayıcının isimleri değiş tokuş edilemez
- Veri belleğine doğrudan erişimde WREG ismini kullanmalıyız, diğer durumlarda ise W0 ismini kullanmalıyız

Örneğin:

A. W0 saklayıcının içeriğini veri belleği adresine kopyalama: mov WREG, 0x1002

B. Veri belleği adresinden W0 saklayıcısına kopyalama: mov 0x1002, WREG

Temel Saklayıcı (W0) Kullanan MOV Komutu (2)

- Veri belleğinin içeriğini W1 ila W15 arasında olan 15 saklayıcıdan birine kopyalama durumunda sadece WORD mod (16-bit) kullanılır

  Örnek: mov 0x1002, W3

- Veri belleğinin içeriğini W0 saklayıcına kopyalama durumunda, ya WORD mod (16-bit) ya da BYTE mod (8-bit) kullanılır.

  Örnek: mov.b 0x1001, WREG

0x1001 bellek adresindeki 1 byte uzunluktaki veri W0 saklayıcısının düşük anlamlı byte kısmına kopyalanır. W0 saklayıcısındaki yüksek anlamlı byte kısmı ise değişmeden kalır

mov.b 0x1001, WREG

W0=0x64DE

Öncesi:

Location	Contents
0x1000	0x8B1A
0x1002	0x25AC
0x1004	0xFB90
0x1006	0x9ED7

kopyalandı

W0=0x648B (değişti)

Sonrası:

Location	Contents	
0x1000	0x8B1A	etkilenmedi
0x1002	0x25AC	
0x1004	0xFB90	
0x1006	0x9ED7	

Temel Saklayıcı (WO) Kullanan MOV Komutu (3)

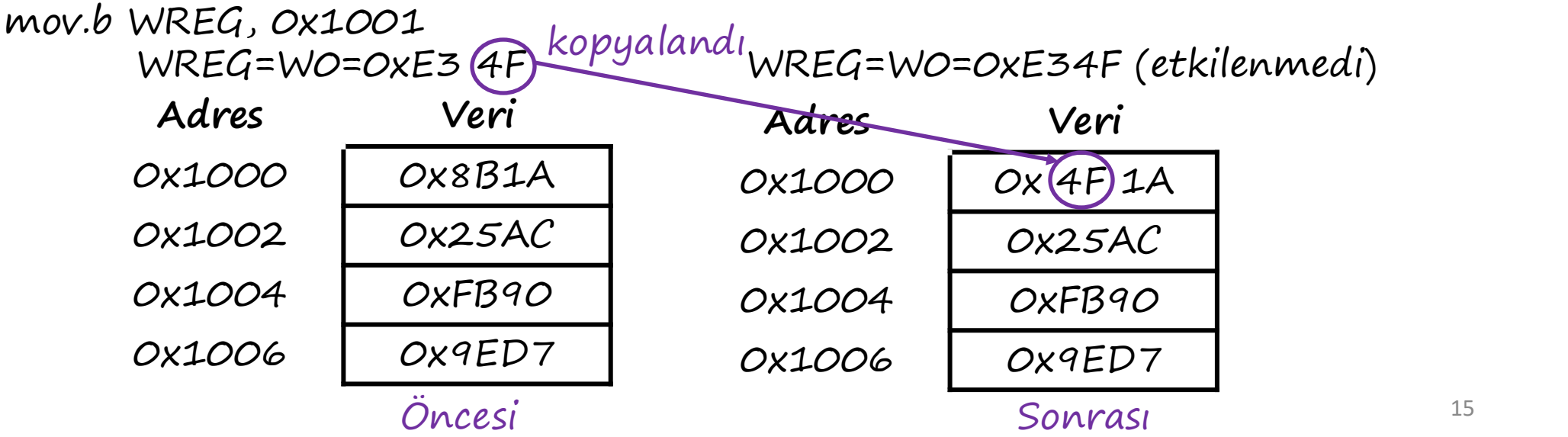
- W1 ila W15 arasında olan 15 saklayıcının birinden veri belleğine kopyalama durumunda sadece WORD mod (16-bit) kullanılır

  Örnek: mov W3, 0x1002

- WO saklayıcısından veri belleğine kopyalama durumunda, ya WORD mod (16-bit) ya da BYTE mod (8-bit) kullanılır.

  Örnek: mov.b WREG, 0x1001

WO saklayıcısının düşük anlamlı kısmını 0x1001 ile belirtilen veri belleğinin adresine kopyalar



Çalışma Saklayısına Sabit bir Sayının Taşınması (1)

- Sabit bir sayıyı çalışma saklayıcısına taşımanın genel biçimi:

A. Word mod: mov #lit16, Wn

16 bitlik bir sabit sayısını Wn saklayıcına taşır, burada Wn, W0 ile W15 arasında bir saklayıcı olabilir.

B. Byte mod: mov.b #lit8, Wn

8 bitlik bir sabit sayısını Wn saklayıcısının düşük anlamlı byte kısmına taşır, burada Wn, W0 ile W15 arasında olabilir. Wn saklayıcısının yüksek anlamlı byte kısmı ise değişmeden kalır.

Örnekler:

A. mov #0x1000, W2 — 0x1000 sayısını W2 saklayıcısına taşınmaktadır.

B. mov.b #0xA3, W3 — 0xA3 sayısını W3 saklayıcısının düşük anlamlı byte kısmına taşır. Wn saklayıcısının yüksek anlamlı byte kısmı değişmeden kalır.

Sayı formatı binary (#0b1001), hex (#0xAC14) ya da dec (#32) olabilir

Çalışma Saklayısına Sabit bir Sayının Taşınması (2)

- mov #0xxxxx, Wn ve mov 0xxxxx, Wn arasındaki farklılık:

A. mov #0x1000, W2 W2= 0x1000

B. mov #32, W2 W2= 32

C. mov.b #0xAD, W2 0xAD sayısı W2 saklayıcısının düşük anlamlı kısmına taşınır. W2 saklayıcısının yüksek anlamlı kısmı ise değişmez

D. mov 0x1000, W2 0x1000 ile belirtilen adresin içeriği W2 saklayıcısına kopyalanır

Örnek:

ADD Komutu (1)

- Toplama komutunun genel biçimi:

  A. Word mod: add Wa, Wb, Wc

  Wa + Wb->Wc

  Wa, Wb, Wc saklayıcıları W0 ila W15 arasında olan 16 çalışma saklayıcısından biri olabilir.

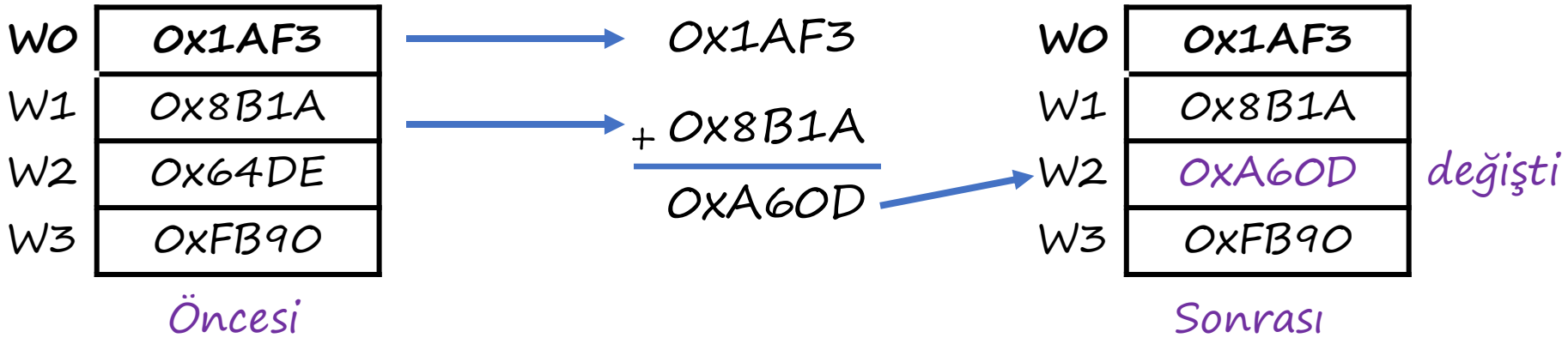
  B. Byte mod: add.b Wa, Wb, Wc

  Wc.lsb = (Wa.lsb) + (Wb.lsb) , Wc.msb=değişmez

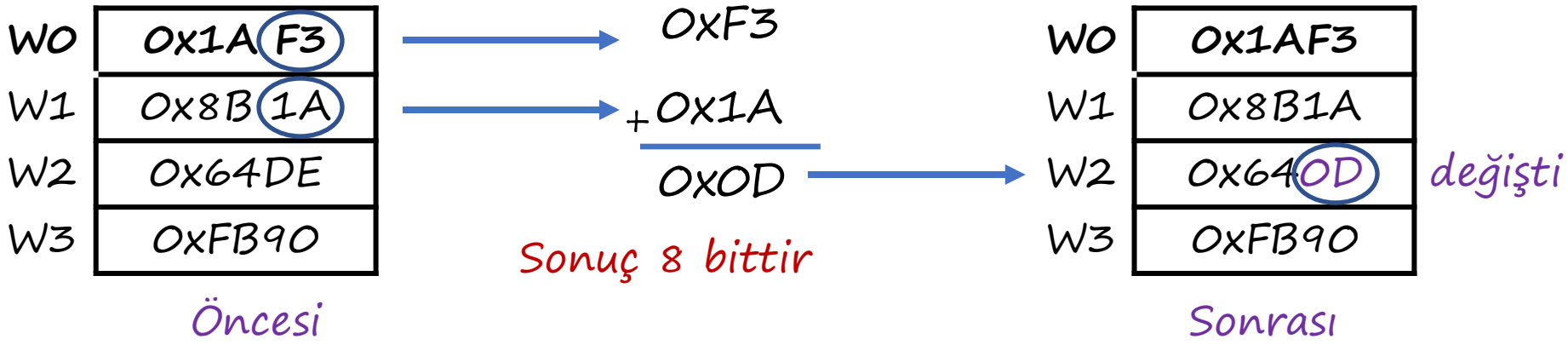
  Wa, Wb, Wc saklayıcıları W0 ila W15 arasında olan 16 çalışma saklayıcısından biri olabilir.

ADD Komutu (2)

(a) add W0, W1, W2

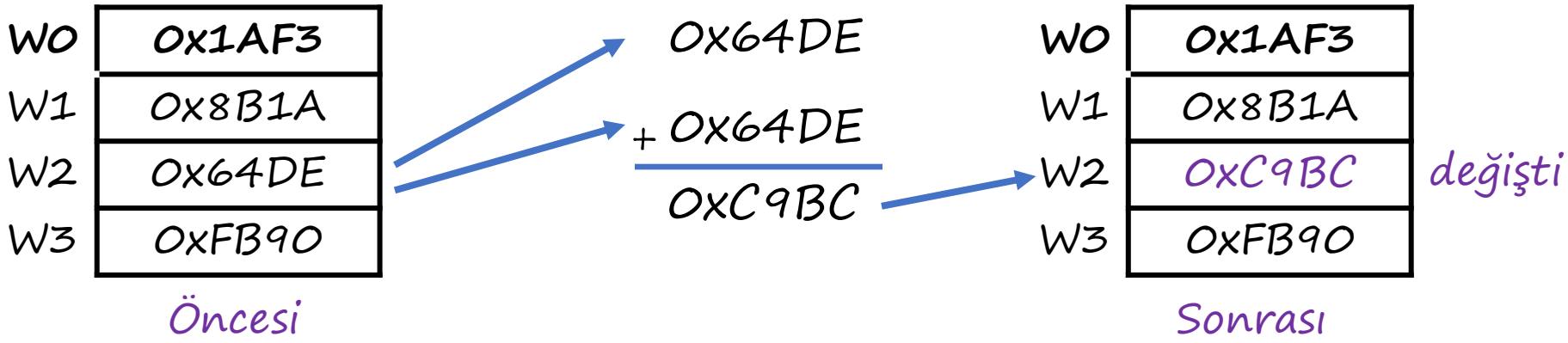


(b) add.b W0, W1, W2



ADD Komutu (3)

(c) add W2, W2, W2



ADD Komutunu Kullanarak Sabit Bir Sayı (Literal) ile Toplama

- Sabit bir sayı ile toplama:

A. Word mod: add Wa, #lit5, Wb

Wa saklayıcısındaki değer ile **5-bitlik** bir sabit sayı toplanır. Sonuç Wb saklayıcısına yerleştirilir. Wa ve Wb, W0 ile W15 arasında herhangi bir saklayıcı olabilir

(Wa) + #lit5 -> Wb

B. Byte mod: add.b Wa, #lit5, Wb

Wa saklayıcısının düşük anlamlı byte kısmı ile **5-bitlik** bir sabit sayı toplanır. Sonuç Wb saklayıcısının düşük anlamlı byte kısmına yerleştirilir. Wb saklayıcısının yüksek anlamlı byte kısmı değişmez. Wa ve Wb, W0 ile W15 arasında herhangi bir saklayıcı olabilir.

(Wa.lsb) + #lit5 -> Wb.lsb

Önemli: #lit5 5 bitlik sabit bir sayıdır. Yani sayı aralığı: 0 — ($2^5 - 1$): 0 - 31

add W0, #4, W2

add W0, #60, W2

60 > 31

ADD Komutunu Kullanarak Veri Belleğindeki Değer ile Toplama

- Veri belleğindeki değer ile toplama komutu:

A. Word mod: add f, WREG

WO saklayıcısındaki değer ile f ile belirtilen bellek adresindeki değer toplanır. Sonuç WO saklayıcısına yerleştirilir. Veri belleğine doğrudan erişim olduğu için WREG ismini kullanmalıyız. (f) + (WREG) -> WREG

B. Byte mod: add.b f, WREG

WO saklayıcısının düşük anlamlı byte kısmı ile f ile belirtilen bellek adresindeki 1 byte uzunluğundaki değer toplanır. Sonuç WO saklayıcısının düşük anlamlı byte kısmına yerleştirilir. WO saklayıcısının yüksek anlamlı byte kısmı değişmez. (f) + (WREG.lsb) → WREG.lsb

C. Bir operandlı word mod: add f

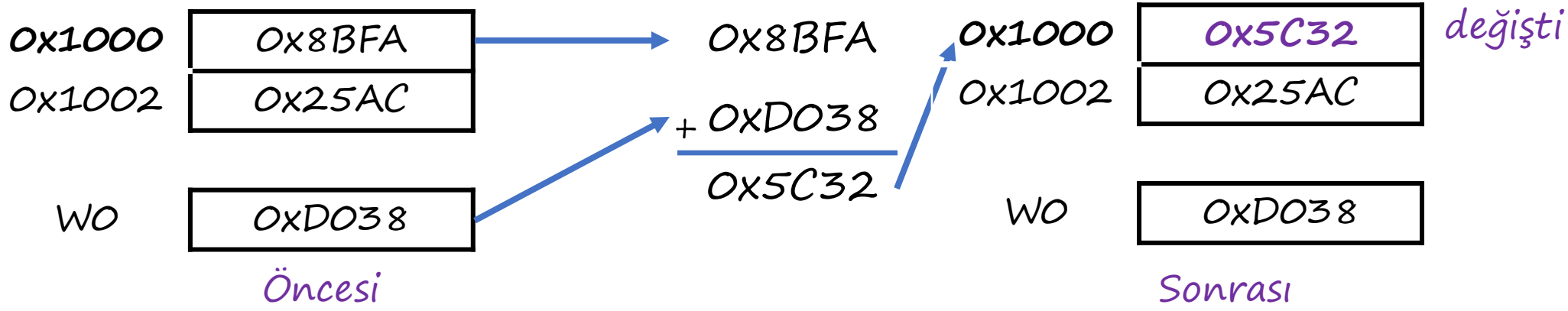
WO saklayıcısındaki değer ile f ile belirtilen bellek adresindeki değer toplanır. Sonuç f ile belirtilen bellek adresine yerleştirilir. (f) + (WREG) → f

D. Bir operandlı byte mod: add.b f

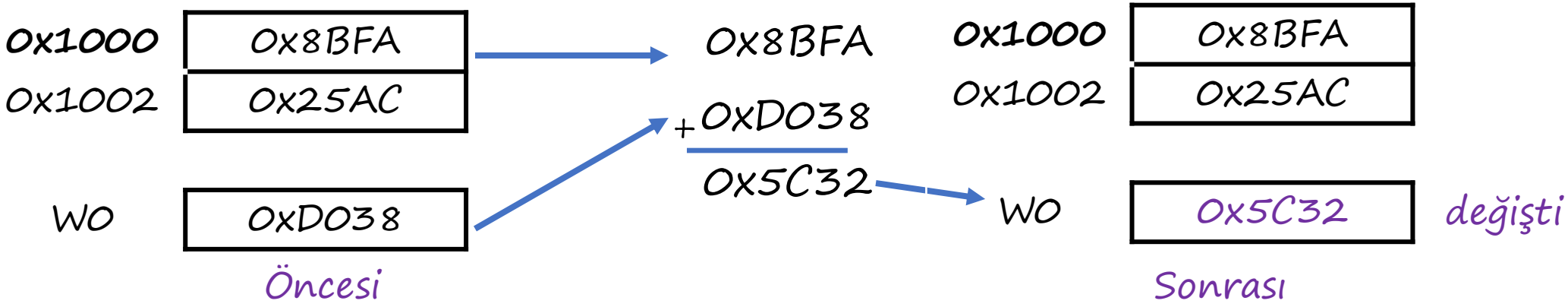
WO saklayıcısının düşük anlamlı byte kısmı ile f ile belirtilen bellek adresindeki 1 byte uzunluğundaki değer toplanır. Sonuç (1 byte) f ile belirtilen bellek adresine yerleştirilir. (f) + (WREG.lsb) → f

ADD Komutunu Kullanarak Veri Belleğindeki Değer ile Toplama

(a) add 0x1000

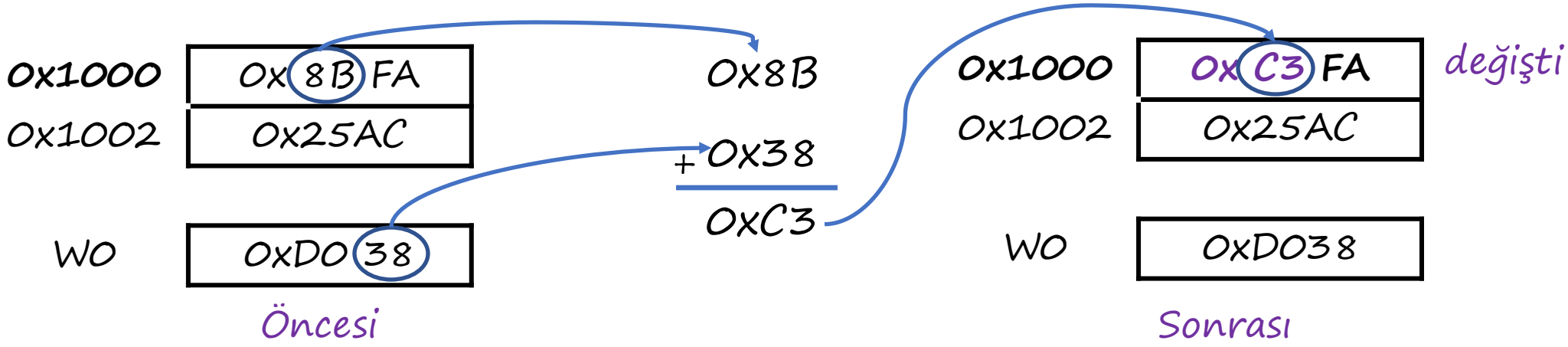


(b) add 0x1000, WREG



ADD Komutunu Kullanarak Veri Belleğindeki Değer ile Toplama

(a) ADD.B 0x1001



SUB Komutu (1)

- Çıkarma komutunun genel biçimi:

  A. Word mod: sub Wa, Wb, Wc

  (Wa) – (Wb) → Wc. Wa, Wb, Wc saklayıcıları W0 ila W15 arasında olan 16 çalışma saklayıcısından biri olabilir.

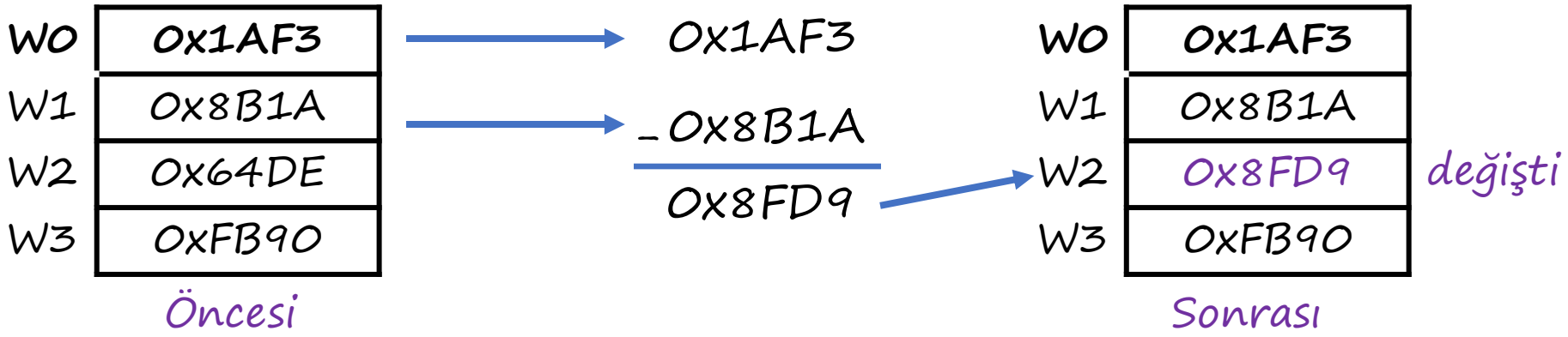
  B. Byte mod: sub.b Wa, Wb, Wc

  Wa ve Wb saklayıcılarının düşük anlamlı byte kısmı çıkarılır. Sonuç Wc saklayıcısının düşük anlamlı kısmına yerleştirilir. Wc saklayıcısının yüksek anlamlı kısmı ise değişmez. Wa, Wb, Wc saklayıcıları W0 ila W15 arasında olan 16 çalışma saklayıcısından biri olabilir.

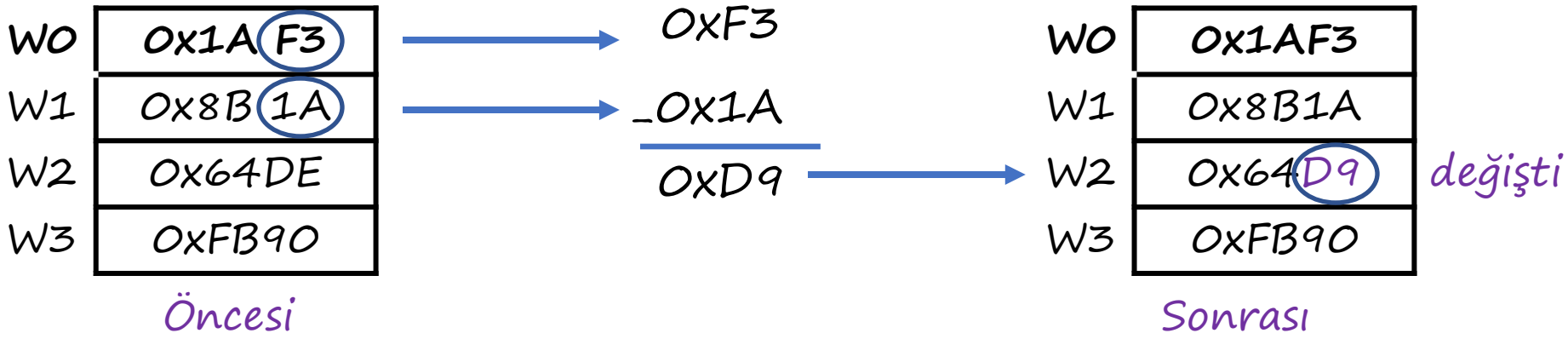
  (Wa.lsb) – (Wb.lsb) → Wc.lsb

SUB Komutu (2)

(a) sub W0, W1, W2



(b) sub.b W0, W1, W2



SUB Komutunu Kullanarak Sabit Bir Sayı (Literal) ile Çıkarma

- Sabit bir sayı ile çıkarma:

A. Word mod: sub Wa, #lit5, Wb

Wa saklayıcısındaki değerden **5-bitlik** bir sabit sayı çıkarılır. Sonuç Wb saklayıcısına yerleştirilir. Wa ve Wb, W0 ile W15 arasında herhangi bir saklayıcı olabilir. (Wa) – #lit5 -> Wb

B. Byte mod: sub.b Wa, #lit5, Wb

Wa saklayıcısının düşük anlamlı byte kısmından **5-bitlik** bir sabit sayı çıkarılır. Sonuç Wb saklayıcısının düşük anlamlı byte kısmına yerleştirilir. Wb saklayıcısının yüksek anlamlı byte kısmı ise değişmez. Wa ve Wb, W0 ile W15 arasında herhangi bir saklayıcı olabilir. (Wa.lsb) – #lit5 -> Wb.lsb

Önemli: #lit5 5 bitlik sabit bir sayıdır. Yani sayı aralığı: 0 — $(2^5 - 1)$: 0 - 31

sub W0, #4, W2

4 < 31

sub W0, #60, W2

60 > 31

SUB Komutunu Kullanarak Veri Belleğindeki Değer ile Çıkarma (1)

- Veri belleğindeki değer ile çıkarma:
  A. Word mod: sub f, WREG

     f ile belirtilen bellek adresindeki değerden WO saklayıcısındaki değer çıkarılır. Sonuç WO saklayıcısına yerleştirilir. WO veri belleğine doğrudan eriştiği için WREG ismini kullanmalıyız. (f) – (WREG) -> WREG

  B. Byte mode: sub.b f, WREG

     f ile belirtilen bellek adresindeki 1 byte uzunluğundaki değerden WO saklayıcısının düşük anlamlı byte kısmı çıkarılır. Sonuç WO saklayıcısının düşük anlamlı byte kısmına yerleştirilir. WO saklayıcısının yüksek anlamlı byte kısmı değişmez. (f) – (WREG.lsb) → WREG.lsb

  C. Bir operandlı word mod: sub f

     f ile belirtilen bellek adresindeki değerden WO saklayıcısındaki değer çıkarılır. Sonuç f ile belirtilen bellek adresine yerleştirilir. (f) – (WREG) → f

  D. Bir operandlı byte mod: sub.b f

     f ile belirtilen bellek adresindeki 1 byte uzunluğundaki değerden WO saklayıcısının düşük anlamlı byte kısmı çıkarılır. Sonuç (1 byte) f ile belirtilen bellek adresine yerleştirilir. (f) – (WREG.lsb) → f

Artırım Komutu (1)

- Saklayıcılar arası artırım komutu:
  A. Word mod: inc Wa, Wb
     Wa saklayıcısındaki değere 1 eklenir. Sonuç Wb saklayıcında saklanır.
     (Wa) + 1 -> Wb
  B. Byte mod: inc.b Wa, Wb
     Wa saklayıcısının düşük anlamlı byte kısmına 1 eklenir ve Sonuç Wb saklayıcısının düşük anlamlı byte kısmına yerleştirilir. Wb saklayıcısının yüksek anlamlı byte kısmı değişmez. (Wa.lsb) + 1 -> Wb.lsb
- Bellek üzeri artırım komutu:
  A. Word mod: inc f
     f ile belirtilen bellek adresindeki değere 1 eklenir ve sonuç f bellek adresinde saklanır. (f) + 1 → f
  B. Byte mod: inc.b f
     f ile belirtilen bellek adresindeki değerin düşük anlamlı byte kısmına 1 eklenir. Sonuç f ile belirtilen bellek adresinin düşük anlamlı byte kısmına yerleştirilir. f ile belirtilen bellek adresindeki değerin yüksek anlamlı byte kısmı değişmez . (f.lsb) + 1 → f.lsb

Artırım Komutu (2)

- Bellek adresinden temel saklayıcıya (WO) doğru artırım komutu:
  A. Word mod: inc f, WREG

     f ile belirtilen bellek adresindeki değere 1 eklenir ve sonuç WO saklayıcısında saklanır. (f) + 1 → WREG

  B. Byte mode: inc.b f, WREG

     f ile belirtilen bellek adresindeki değerin düşük anlamlı byte kısmına 1 eklenir. Sonuç WO saklayıcısının düşük anlamlı byte kısmına yerleştirilir. WO saklayıcısının yüksek anlamlı byte kısmı ise değişmez. (f.lsb) + 1 → WREG.lsb

Örnek:
W2 = 0x40FF, W4 = 0xAD12
inc W2, W4 —> W4 = 0x4100
inc.b W2, W4 —> W4 = 0xAD00

Byte modunda, yüksek anlamlı byte kısmı değişmez.

Azaltım Komutu (1)

- Saklayıcılar arası azaltım komutu:
  A. Word mod: dec Wa, Wb
     Wa saklayıcısındaki değerden 1 çıkarılır. Sonuç Wb saklayıcında saklanır.
     (Wa) – 1 -> Wb
  B. Byte mod: dec.b Wa, Wb
     Wa saklayıcısının düşük anlamlı byte kısmından 1 çıkarılır ve sonuç Wb saklayıcısının düşük anlamlı byte kısmına yerleştirilir. Wb saklayıcısının yüksek anlamlı byte kısmı değişmez. (Wa.lsb) – 1 -> Wb.lsb
- Bellek üzeri azaltım komutu:
  A. Word mod: dec f
     f ile belirtilen bellek adresindeki değerden 1 çıkarılır ve sonuç f bellek adresinde saklanır. (f) – 1 → f
  B. Byte mode: dec.b f
     f ile belirtilen bellek adresindeki değerin düşük anlamlı byte kısmından 1 çıkarılır. Sonuç f ile belirtilen bellek adresinin düşük anlamlı byte kısmına yerleştirilir. f ile belirtilen bellek adresindeki değerin yüksek anlamlı byte kısmı değişmez . (f.lsb) – 1 → f.lsb

Azaltım Komutu (2)

- Bellek adresinden temel saklayıcıya (WO) doğru azaltım komutu: :
  A. Word mod: dec f, WREG
     f ile belirtilen bellek adresindeki değerden 1 çıkarılı ve sonuç WO saklayıcısında saklanır. (f) – 1 → WREG
  B. Byte mode: dec.b f, WREG
     f ile belirtilen bellek adresindeki değerin düşük anlamlı byte kısmından 1 çıkarılır. Sonuç WO saklayıcısının düşük anlamlı byte kısmına yerleştirilir. WO saklayıcısının yüksek anlamlı byte kısmı ise değişmez. (f.lsb) – 1 → WREG.lsb

Örnek:
W2 = 0x1100, W4 = 0xAD12
dec W2, W4 —> W4 = 0x10FF
dec.b W2, W4 —> W4 = 0xADFF

Byte modunda, yüksek anlamlı byte kısmı değişmez.

C dilinden Assembly diline — İsimlendirme Kuralları (1)

A. u8_a: İşaretsiz 8-bitlik a değişkeni

B. u16_b: İşaretsiz 16-bitlik b değişkeni

C. u32_c: İşaretsiz 16-bitlik c değişkeni

D. i8_a: İşaretli 8-bitlik a değişkeni

E. i16_b: İşaretli 16-bitlik b değişkeni

F. i32_c: İşaretli 32-bitlik c değişkeni

G. au8_a: a dizisi, dizideki her bir eleman işaretsiz 8-bitlik değerindedir

H. au16_b: b dizisi, dizideki her bir eleman işaretsiz 16-bitlik değerindedir

I. au32_c: c dizisi, dizideki her bir eleman işaretsiz 32-bitlik değerindedir

C dilinden Assembly diline — İsimlendirme Kuralları (2)

H. ai8_a: a dizisi, dizideki her bir eleman işaretli 8-bitlik değerindedir

I. ai16_b: b dizisi, dizideki her bir eleman işaretli 16-bitlik değerindedir

J. ai32_c: c dizisi, dizideki her bir eleman işaretli 32-bitlik değerindedir

K. pu8_a: işaretsiz 8 bitlik bir değere işaret eden a işaretçisi (pointer)

L. pu16_b: işaretsiz 16 bitlik bir değere işaret eden b işaretçisi

M. pu32_c: işaretsiz 32 bitlik bir değere işaret eden c işaretçisi

N. pi8_a: işaretli 8 bitlik bir değere işaret eden a işaretçisi

O. pi16_b: işaretli 16 bitlik bir değere işaret eden b işaretçisi

P. pi32_c: işaretli 32 bitlik bir değere işaret eden c işaretçisi

C dilinden Assembly diline— Basit bir Program

C kodu	Assembly kodu
`u8_i = 100;`	`mov.b #100, WO` `mov.b WREG, _u8_i`
`u8_i = u8_i + 1;`	`inc.b _u8_i`
`u8_j = u8_i;`	`mov.b _u8_i, WREG` `mov.b WREG, _u8_j`
`u8_j = u8_j - 1;`	`dec.b _u8_j`

`u8_k = u8_i + u8_j;`	`mov.b _u8_i, WREG`
	`add.b _u8_j, WREG`
	`mov.b WREG, _u8_k`

- Sabit sayılar (literal) doğrudan veri belleğine taşınamaz
- WO veri belleğine erişmesi durumda WREG ismini kullanmalıyız

- mov.b komutunda, veri belleğine erişebilmek için yalnızca WO kullanılabilir
- Verileri bir bellek adresinden diğerine doğrudan taşıyamazsınız:
  mov _u8_i, _u8_j

C dilinden Assembly diline– Bellek Tahsisi

```
.include pic24_all.h
.global _reset
```

Giriş

```
.bss
_u8_i:          .space 1
_u8_j:          .space 1
_u8_k:          .space 1
```

Her 8 bitlik değişken için 1 byte bellek ayırılır.

```
.text
._reset:
        mov #__SP_init, W15
        mov #__SPLIM_init, W0
        mov W0, SPLIM
```

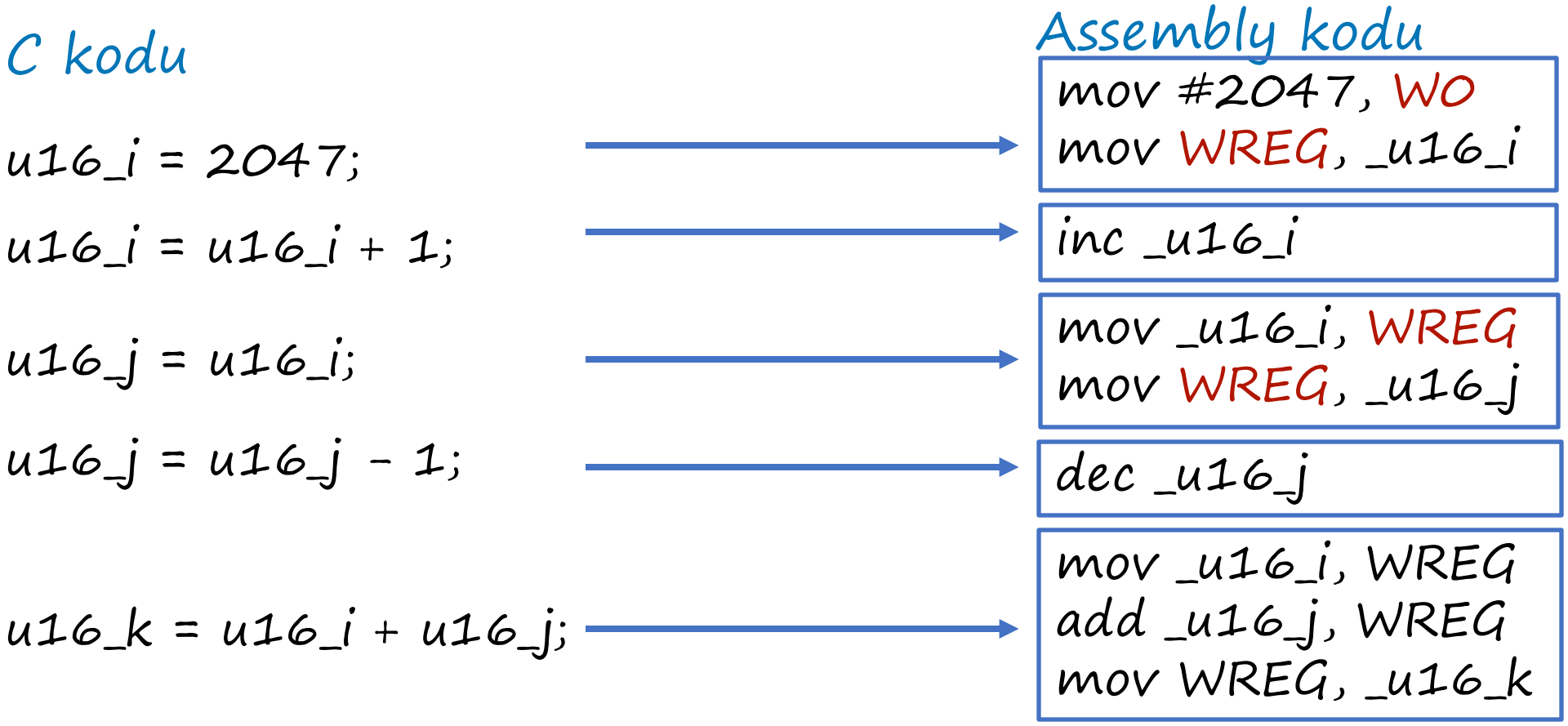
Yığın yapılandırma

_reset etiketinin altında ilk komut bulunur. Bu, derleyicinin "_reset" etiketi altındaki komutları satır satır makine koduna dönüştüreceği anlamına gelir.

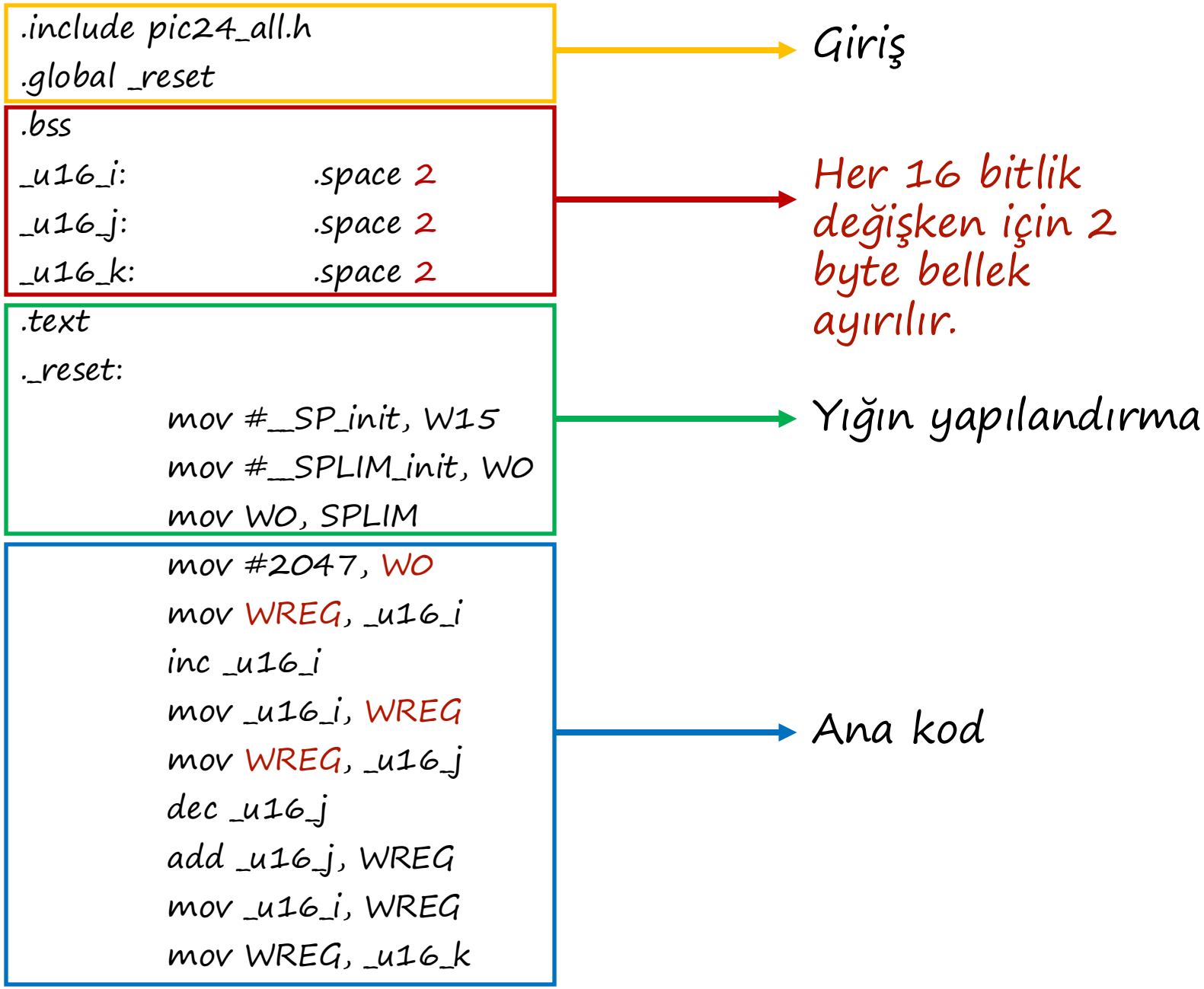
```
        mov.b #100, W0
        mov.b WREG, _u8_i
        inc.b _u8_i
        mov.b _u8_i, WREG
        mov.b WREG, _u8_j
        dec.b _u8_j
        add.b _u8_j, WREG
        mov.b _u8_i, WREG
        mov.b WREG, _u8_k
```

Ana kod

C dilinden Assembly diline— 16 bit değişken kullanımı (1)



C dilinden Assembly diline — 16 bit değişken kullanımı (2)



Saat Çevrimi (Clock Cycle) ve Komut Çevrimi (Instruction Cycle) Kavramları

- PIC' deki farklı donanımları senkronize etmek için saat sinyali kullanılır.
- Saat sinyal kaynakları:
  A. External crystal oscillator (Harici kristal osilatörü)
  B. External RC circuit (Harici RC osilatörü)
  C. Internal RC circuit (Dahili RC osilatörü)
- Çoğu PIC33/PIC24 µC için maksimum saat frekansı (Fosc) = 80 MHz
- Önemli: Bir komut çevrimi= 2 x saat çevrimi (saat periyodu) !!!
- PIC komutlarının çoğu yürütülmesi için sadece 1 komut çevrimine ihtiyaç duyar, ancak bazı komutlar için birden fazla komut çevrimi gerekebilir.
- Genel olarak, bir komut program sayıcının girişten yeni adres yüklemesini sağlıyorsa, bir komut çevriminden fazla sürecektir.

Toplam Komut Çevrimlerinin Hesaplanması

```
.include pic24_all.h
.global _reset

.bss
_u8_i:          .space 1
_u8_j:          .space 1
_u8_k:          .space 1

.text
._reset:
                mov #__SP_init, W15
                mov #__SPLIM_init, WO
                mov WO, SPLIM
                mov.b #100, WO
                mov.b WREG, _u8_i
                inc.b _u8_i
                mov.b _u8_i, WREG
                mov.b WREG, _u8_j
                dec.b _u8_j
                add.b _u8_j, WREG
                mov.b _u8_i, WREG
                mov.b WREG, _u8_k
```

Komut Çevrim
1
1
1
1
1
1
1
1
1
1
1
1

_reset etiketinin altında ilk komut bulunur. Bu, derleyicinin "_reset" etiketi altındaki komutları satır satır makine koduna dönüştüreceği anlamına gelir.

Toplam Zaman Tüketimi

Komut (Instruction)	Komut çevrimi (Instruction Cycle)
mov #__SP_init, W15	1
mov #__SPLIM_init, W0	1
mov W0, SPLIM	1
mov.b #100, W0	1
mov.b WREG, _u8_i	1
inc.b _u8_i	1
mov.b _u8_i, WREG	1
mov.b WREG, _u8_j	1
dec.b _u8_j	1
add.b _u8_j, WREG	1
mov.b _u8_i, WREG	1
mov.b WREG, _u8_k	1
Toplam:	12

Soru: Saat frekansı Fosc = 80 MHz ise, soldaki tablodaki komutların uygulanması ne kadar sürer?

Adım 1: Toplam saat çevrim sayısı = 2 x komut çevrimi = 2 x 12 = 24

Adım 2: Toplam zaman tüketimi= toplam saat çevrim sayısı / saat frekansı
= 24 / 80 MHz = 0.3 µs