

INCLUDE Dosyalar

Assembly programlarını yazarken kullanılacak register adreslerini (EQU) komutu ile tanımlamak hem kolaylık sağlamak hem de anlaşılabilirliği arttırmakta idi. Ancak her sefer bu tanımları tekrar tekrar yapmak gereksiz gibidir.

Bunun yerine **Include Dosya** kullanarak aynı isimli her PIC için sabit olan bu tanımları her programda yeniden yapmaktan kurtulmuş oluruz.

Mesela; **P16F84.INC** dosyası PIC16F84 için gerekli tanımları içerir ve MPASM klasörü içinde (.asm) dosyamızla aynı yerde bulunmalıdır. Include dosyası içinde Register adres tanımlarının yanı sıra RAM bölgesinin tanımı ve **Konfigürasyon Bitleri** denilen bazı bitlerin tanımı da yapılmaktadır.

Programımız içinde Include Dosyası kullanabilmemiz için (PIC16F84 de) INCLUDE “P16F84.INC” satırını yazmamız yeterlidir.

Include dosya sayesinde (örneğin) STATUS registerinin her bir bitine isim verildiği için:

BCF STATUS, 5 yerine

BCF STATUS, RP0 yazmak mümkündür. Aynı şeyler INTCON, OPTION gibi diğer registerler içinde geçerlidir. Artık her File Register için kullanılan (EQU) komutları kalktığı için kaynak (assembly) programımız önemli ölçüde kısaldı sadeleşecektir. Sadece Genel Amaçlı RAM bölgesinde kullanacağımız bir değişken için (SAYAC gibi) EQU komutu yazmamız gerekebilecektir.

KONFIGÜRASYON BİTLERİ

Bu bitler PIC’ e gerilim verildiği anda geçerli kuralları belirlemek içindir. Mesela; PIC devremizin saatini (osilatörünü) RC tipi olarak kullanacaksak bunu programda bildirmemiz lazımdır.

Benzer şekilde;

Watchdog timer’ i devreye sokmak veya çıkarmak,

Power-on Reset özelliğini devreye sokmak ya da çıkarmak,

Kod korumayı devreye almak veya almamak için bu konfigürasyon bitleri kullanılır.

Aslında konfigürasyon bitleri programa yazılmadan programlama esnasında da doğrudan belirlenebilir. Zira bütün programlayıcı programları bu imkanı vermektedir. Eğer program içersinde bir satır olarak Konfigürasyon Bitlerini vermek istersek bu komutun yazılışına ait bir örnek şöyle verilebilir:

_CONFIG _CP_OFF & WDT_OFF & PWRTE_OFF & RC_OSC
 └──┬──┘ └──┬──┘ └──┬──┘ └──┬──┘
 Kod Komut Yok Watchdog Timer OFF Power-on Reset Yok Osilatör Tipi (RC)

& : “ve” demektir.

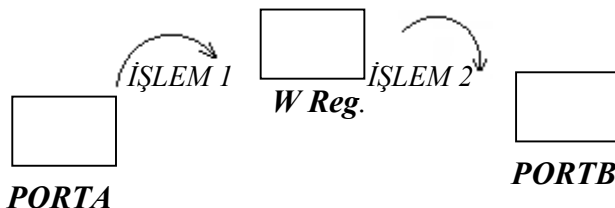
_ : Alt çizgi boşluk yerine kullanılıyor.

OFF : Yok, devre dışı anlamına gelir.

VERİ AKTARMA İŞLEMLERİ

PIC’ de veri aktarma (transfer) işlemleri W registeri üzerinden yapılmaktadır.

Örnek:



Bu işlemi yapmak için Gerekli Komutlar:

MOVF PORT A , W ; (İŞLEM 1) Port A ' yı W' ye aktar.

MOVWF PORT B ; (İŞLEM 2) W 'yi Port B' ye aktar.

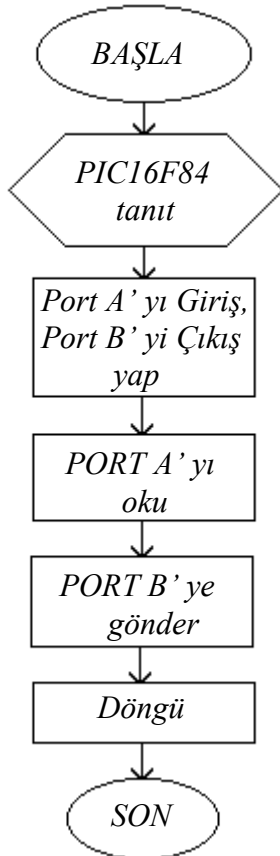
NOT: PORTA asla doğrudan Port B' ye aktarılmaz.

Örnek: '0F' sayısını PORT B' ye yazmak için

MOVLW h'0F' }
MOVWF PORT B } yazılmalıdır.

Program Örneği 1: PIC' e enerji verildiği anda PORTA da basılı butona karşılık gelen B Portundaki LED' leri söndüren bir program yazalım.

Cözüm: Önce akış diyagramını çizelim:



NOT: Akış Diyagramında Bank değiştirme işlemi ayrıntıları gösterilmemiştir.

Programda INCLUDE komutu kullanalım.

```
; PROGRAM1.ASM -----09 / 11 / 2003
LIST      P = 16F84
INCLUDE   "P16F84.INC"
CLRF      PORTB
```

BSF	STATUS,5	; BANK 1' e geç
CLRF	TRISB	; B portu tümü çıkış
MOVLW	h'FF'	
MOVWF	TRISA	; TRISA' yı FF yükle, PORTA tamamen Giriş .
BCF	STATUS,5	; BANK 0' a geç
OKU		
MOVF	PORTA,W	; PORTA' yı oku sonucu W' ye yaz
MOVWF	PORTB	; W ' yi PORT B' ye yaz (kopyala)
DONGU		
GOTO	DONGU	; Sonsuz Döngü, GOTO OKU da denebilir.
END		; Son

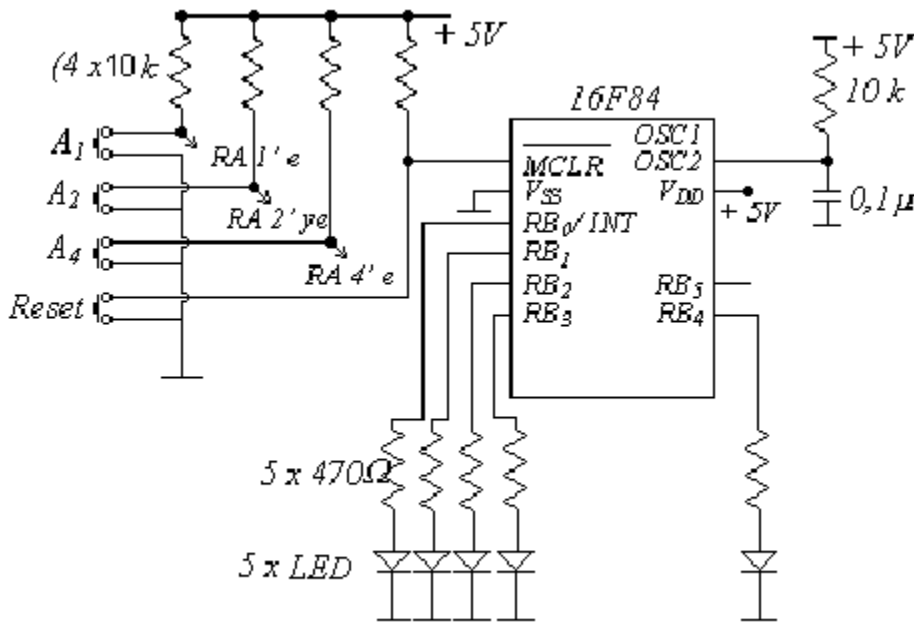
Pratik Gerçekleştirme İçin Gerekli Adımlar:

- 1) Programı yazıp PROGRAM1.ASM olarak kayıt (save) ediniz.
- 2) MPASM ile derleme yapıp (PROGRAM1.hex) dosyasını elde ediniz. Konfigürasyon bitlerine dikkat ederek programlayıcıya PIC' i yerleştirip programı yükleyiniz. Konfigürasyon bitleri için

Osilatör tipi : RC	PWRT : ON	} yazılabilir.
WDT : OFF	CP : OFF	

Not: INCLUDE (.INC) dosya kullanıldığı için 16F84.INC dosyasının da aynı klasör (directory) içinde bulunmasını sağlamayı ihmal etmeyin.

- 3) Devreyi aşağıdaki gibi kurun.



NOT 1: RA0, RA3 girişleri de (+) beslemeye bağlıdır. Yani Lojik 1 almaktadır.

NOT 2: PIC devrelerinde besleme gerilimi (+ 5V) en az 0,1 µF bir kondansatör ile şaseye bağlanmalıdır.

- 4) Devreye enerji vermeden A1, A2, A4 butonlarına tek yada birlikte basınız.

- 5) Basılan buton ve yana LED ilişkisini inceleyin.

NOT: Hiçbir butona basılmazsa PortA bitlerinin harici pull-up dirençleri (10k) dolayısıyla LED' leri yaktığı görülecektir. Zira A portu normalde (1) olup basınca (0) olmaktadır.

6) Durum deęiřtirirken önce istedięiniz butonlara basın sonra RESET yapın.

Sonsuz Döngü: PIC16F84 de duraklama komutu olmadığı için Döngü kullanılmaktadır. Programda bekleme yerine yeni butonların durumu okunmak isterse,

$\left\{ \begin{array}{l} \text{DONGU} \\ \text{GOTO DONGU} \end{array} \right\}$ komutu yerine

GOTO OKU yazabiliriz. Artık RESET 'e gerek kalmadan tuřlara bastıkça LED deęiřmelerini göreceęiz.

KARAR İÇİN BİT TEST ETME İřLEMİ

Herhangi bir register içindeki bir bit **BTFS** ve **BTFS** komutları ile test edilip sonuca göre ya bir sonraki (sıradaki) satırdan devam edebilir yada bir satır atlanabilir (Program Yönlendirilir).

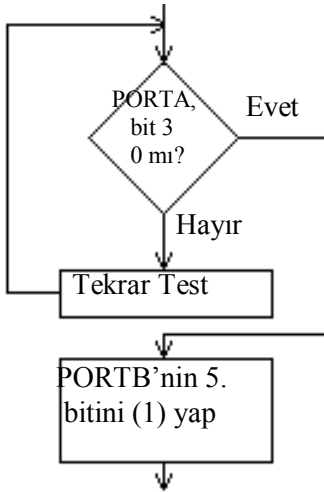
BTFS Komutu: Bu komut ingilizcede “File Register bitini test et, bu bit (0) ise bir sonraki komuta atla” anlamına gelmektedir.

Komut Formatı:

BTFS $\begin{array}{c} 7, 6, 5, 4, 3, 2, 1, 0 \\ \text{File Register} \end{array}$, $\begin{array}{c} \square \\ b \end{array}$ (0,.....,7 arası bir sayı)

řeklinde. Bir örnek program parçası için akıř diyagramı ile birlikte komutun çalışmasını inceleyelim:

Bu program parçası PORTA' nın 3. bitini (RA3) test etmekte bu bit (0) ise PORTB' nin 5. bitini (1) yapmakta ; aksi halde test etmeye devam etmektedir.



TEST

BTFS PORTA, 3
GOTO TEST
BSF PORTB, 5

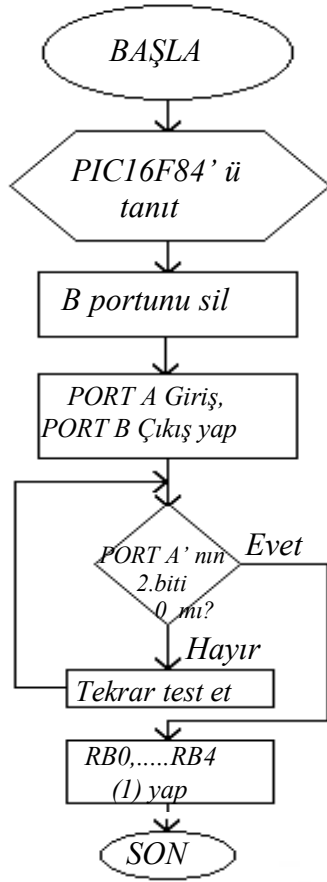
NOT: PORTA ve PORTB' nin önceden tanımlandığı kabul edilmiştir.

BTFS Komutu: Bu komut ise BTFS komutuna benzerdir. Ancak, file register bitinin (0) yerine (1) olduğunu test etmektedir. Bu bit eęer (1) ise komuttan sonraki (sıradaki) komut atlanmakta; deęilse [(0) ise] sıradaki komuttan devam edilmektedir.

Program Örneęi 2: (Yukarıdaki aynı devre için)

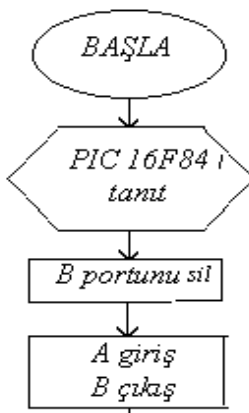
A portunun 2. bitindeki butona basınca B portunun (RB0,.....,RB4) LED' lerini yakan bir program yazalım.

Cözüm:



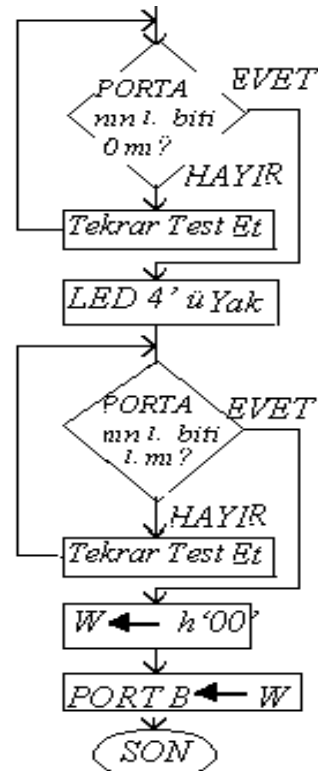
```
; PROGRAM 2.ASM-----08 / 12 / 2003
LIST      P = 16F84
INCLUDE   "P16F84.INC"
CLRF      PORTB
BSF       STATUS, 5
CLRF      TRISA
MOVLW     h'FF'
MOVWF     TRISA
BCF       STATUS, 5
TEST
BTFSC     PORTA, 2
GOTO      TEST
MOVLW     h'0F'
MOVWF     PORTB
END
```

Program Örneği 3: PORTA' nın 1. bitine (RA1) bağlı butonuna basılı olarak enerji verildikten sonra PORTB' deki RB4' e bağlı (LED4) yakacak, daha sonra butondan el çekilince PORTB' nin tüm LED' lerini yakan bir program yapınız. [Butona basılı değilken PORTA 'nın girişlerinin (1) kabul edilmektedir.]



(Devamı Yan Tarafda)

```
-;PROGRAM3.ASM-----08/12/ 2003
LIST      P = 16F84
INCLUDE   "P16F84.INC"
```



```

BSF      STATUS, 5
CLRF     TRISB      ; B Portu Çıkış
MOVLW    h'FF'
MOVWF    TRISA      ;PortA Giriş
BCF      STATUS, 5  ; Bank 0 ' a geç
BSF      PORTB, 4   ; RB4 'ü 1 yap.

```

TEST2

```

BTFSS    PORTA, 1   ;RA1 =1 mi?
GOTO     TEST2      ;Değilse TEST2 ye git
MOVLW    h'FF'
MOVWF    PORTB      ;PortB 'nin tümünü 1 yap

```

DONGU

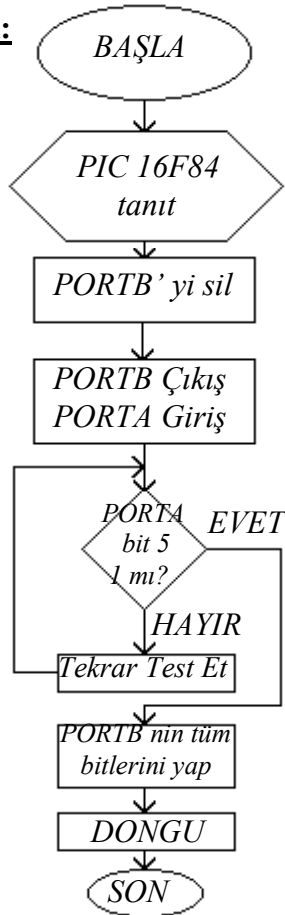
```

GOTO     DONGU
END

```

Program Örneği 4: PORTA' nın 5. bitine bağlı buton normalde (0) durumunda olup butona basılınca (1) gönderilmektedir. PORTA' nın 5. bitine 1 (RA5 = 1) gelince B portuna bağlı 8 LED' in tamamının yanması istenmektedir. Akış diyagramı ile birlikte programı PIC16F84 için yazınız.

Cözüm:



```

;PROGRAM 4.ASM---11 / 12 / 2003
LIST      P = 16F84
INCLUDE   "P16F84.INC"
CLRF      PORTB      ;B Portunu sil
BSF       STATUS, 5  ; Bank 1 'e geç
CLRF      TRISB      ; B Portu Çıkış
MOVLW     h'FF'
MOVWF     TRISA      ; A Portu Giriş
BCF       STATUS, 5  ; Bank 0 ' a geç

TEST
BTFSS     PORTA, 5   ;RA5=1 mi?
GOTO      TEST       ;Değilse TEST 'e
MOVLW     h'FF'
MOVWF     PORTB      ;PortB 'ye FF yaz.

DONGU
GOTO      DONGU
END

```

Program Örneği 5: Enerji verildiği anda A portunun 3. ve 4. bitleri (RA3, RA4) Lojik 1 olduğuna göre, her ikisine de (0) verildiğinde B portunun 5. , 6. , 7. bitlerini (1), diğerlerini (0) yapan aksi takdirde test etmeye devam eden bir assembly programını PIC16F84 için yazınız.

Cözüm:

```

; PROGRAM 5.ASM----11 / 12 / 2003
LIST      P = 16F84
INCLUDE   "P16F84.INC"
CLRF      PORTB

```

```

BSF      STATUS, 5      ; Bank 1' e geç
CLRF     TRISB          ; PORTB tamamı Çıkış
MOVLW    h'FF'
MOVWF    TRISA          ; PORTA Giriş
BCF      STATUS        ; BANK 0' a geç

TEST
    BTFSC PORTA, 3      ; PORTA' nın 3. biti 0 mı?
    GOTO  TEST
    BTFSC PORTA, 4      ; PORTA' nın 4. biti 0 mı?
    GOTO  TEST
    MOVLW b'11100000'   ; W 'ye h'E0' yükle.
    MOVWF PORTB         ; PortB 'ye (E0) yükle

DONGU
    GOTO  DONGU
END

```

DÖNGÜ KULLANMAK

Bazen belli işlem ya da işlemlerin **belirli sayıda tekrarlanması** istenebilir. Böyle bir durumda bir register SAYAÇ olarak kullanılır. Genellikle tekrar sayısı (SAYAÇ)' a yüklendikten sonra her seferinde (SAYAÇ) 1 azaltılarak (0)' a ulaşılan kadar Döngü devamı ettirilir.

Bu amaçla

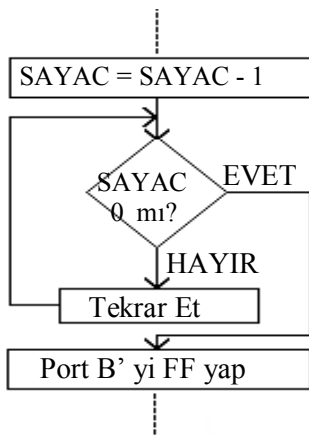
DECFSZ komutu kullanılması gerekir.

Bu komut her icra edildiğinde bu register (burada SAYAC) 1 azaltılır ve SAYAC =0 olunca bir sonraki komuta atlanır. Aksi halde (SAYAC $\neq$ 0 ise) DECFSZ komutundan hemen sonraki (sıradaki) komut icra edilir.

Bu komutun formatı:

DECFSZ Sayaç , d $\leftarrow$ W yada F şeklindedir.

Aşağıdaki program parçasında SAYAC' ın daha önce tanımladığı ve belli bir değerle (döngü sayısı) yüklendiği farz edilmiştir. Döngü bitince PORTB' ye (FF) yüklenmektedir.



```

TEKRAR
    DECFSZ SAYAC, F
    GOTO  TEKRAR
    MOVLW h'FF'
    MOVWF PORTB

```

Program Örneği 6: PORTA' nın 2. bitine bağlı butona 10 kere basıldıktan sonra PORTB' nin 0. bitine bağlı LED' in yakılması isteniyor (Her butona basılması arasında 25 peryot beklenmesi arzu edilmektedir. Böylelikle butondan doğan parazitler engellenmiş ve bir basışta 10 kereyi birden saymasının önüne geçilmiş olacaktır). Gerekli programı PIC 16F84 için assembly dili ile yazınız.

Cözüm: : ; PROGRAM 6.ASM-----20 / 12 / 2003

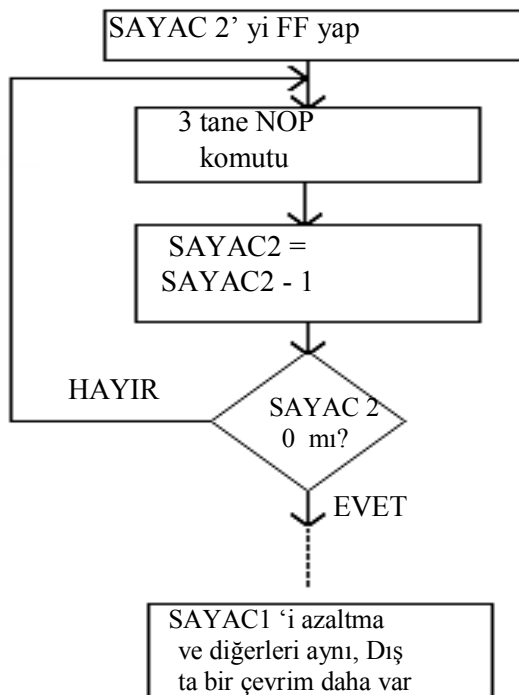
```

LIST      P = 16F84
INCLUDE   "P16F84.INC"
SAYAC     EQU    h'0C'      ; Genel amaçlı RAM ilk adresi
CLRF      PORTB
BSF       STATUS, 5
CLRF      TRISB
MOVLW     h'FF'

MOVWF     TRISA
BCF       STATUS, 5
BASLA
MOVLW     d'10'
MOVWF     SAYAC
TEST
BTFSC     PORTA, 2      ; RA2= 0 mı?
GOTO      TEST          ; Değilse TEST etmeye devam et
NOP        ; İşlem yapma
.
.
.
.
NOP        ; Toplam 25 adet NOP komutu var.
DECFSZ    SAYAC, F      ; Sayacı 1 azalt sonuç 0 mı?
GOTO      TEST          ; Değilse TEST 'e git
BSF       PORTB, 0      ; Evetse, PORTB nin 0. bitini 1 yap.
DONGU
GOTO      DONGU
END

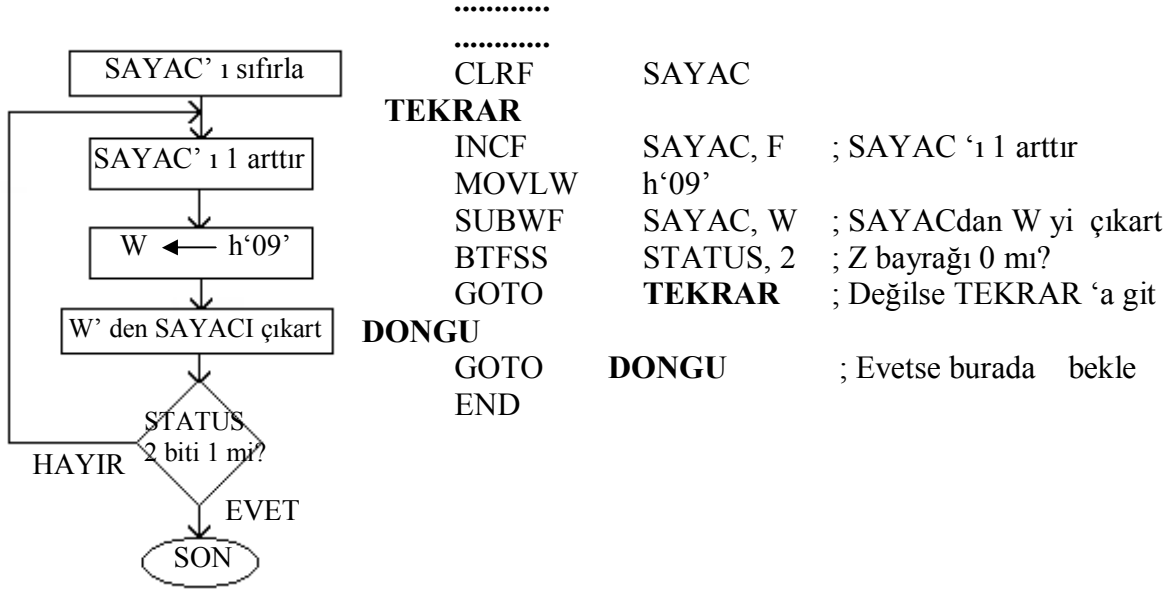
```

Yukarıdaki örnekte 25 adet NOP kullanılmıştır. Böyle bir sistemde iki basma arasında bekleme süresi arttırılmak istenirse bu sayı arttırılır. Ancak bu durumda hafızanın gereksiz dolması gibi bir sorunla karşılaşırız. Bunun yerine 2. bir sayaç tanımlayarak az sayıda (aşağıdaki örnekte 3 tane) NOP' un defalarca icrası (mesela 255 kere) sağlanabilir. Bunun için STATUS 'un 2. bitini olan

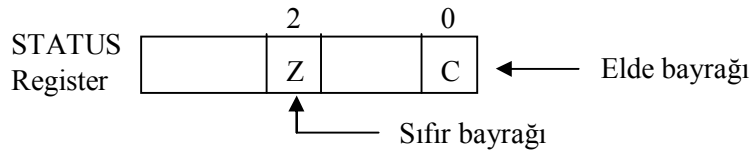


KARŞILAŞTIRMA İŞLEMİ VE DÖNGÜ

SAYAC her zaman azalan yönde çalışmaz. Her döngüde SAYAC' ın arttığı durumlarda SAYAC değeri istenen bir sayı ile karşılaştırılmalıdır. Eşitlik sağlanınca döngü sona erdirilir. Örnek olarak (09) kere tekrarlanan tekrar etiketli bir döngü (çevrim) düzenleyelim: (Program parçası)



Programda SAYAC o ana kadar ki tekrar sayısını belirtir. Çıkarma komutu SUBWF ile STATUS registerinde bulunan Z (Sıfır bayrağı) ve C (Elde bayrağı) etkilenir.



SUBWF Komutu:

Bu komut o andaki File registerinin içeriğini W registerinden çıkarır ve sonucu W yada file registerine yazar. Komut Formatı;

SUBWF

File registeri

 , W yada f

Çıkarma sonucunda Z ve C bayraklarının değeri çıkartılan iki sayı arasındaki ilişki bilgisini taşır.

File registeri > W ise Z = 0 C = 1

File registeri = W ise Z = 1 C = 0

File registeri < W ise Z = 0 C = 0 olur.

Bu bayrakların değerine göre dallanma gerçekleştirilir.

SUBLW Komutu:

SUBLW komutuna benzer olup bu komutla sabit sayıdan W nin içeriği çıkarılır. Sonuç W registerine yazılır. Komut formatı aşağıda verilmiştir.

SUBLW k ; Burada k sabit sayıyı temsil etmektedir.

Örnek: Önce HAFIZA adı verilen ve daha önce tanımlanmış bir adreste h'35' sayısını yazalım. Bu sayı W' ye yüklenen sayıdan küçük yada büyük ise PORTB' nin 0. bitini (1) yapalım. Aksi takdirde (GIT) adresine gönderelim. Gerekli program parçası:

Cözüm:

```
MOVLW h'35'
MOVWF HAFIZA
MOVLW (SAYI)
SUBWF HAFIZA, W
BTFSC STATUS, 2
GOTO GIT
BSF PORTB, 0
```

HAFIZA ≥ SAYI ise

HAFIZA = SAYI ise

NOT

- (SAYI) yerine h'28', h'35', h'4A' gibi bir baytlık sayı yazılacaktır.
- GIT adresi programda başka yerdedir.

Örnek: W registerine h'60' yazdıktan sonra, h'50' sayısından W 'de bulunan sayıyı çıkartalım. C bayrağı (0) ise PORTB' nin 3., 4., ve 5. bitlerine (1) yüklensin. Aksi halde GIT adresine dallanmak üzere bir program parçası yazalım.

Cözüm:

```
MOVLW h'60'
SUBWF h'50'
BTFSC STATUS, 0
GOTO GIT
MOVLW b'00111000'
MOVWF PORTB
```

Program Örneği 7: PIC 16F84 için bir program yazarak h'1A' sayısından h'09' sayısını çıkarınız ve farkı PORTB' ye yazınız. STATUS registerinin içeriğini de W' ye aktarınız.

Cözüm:

```
;PROGRAM 7.ASM---08 / 12 / 2003
LIST P = 16F84
INCLUDE "P16F84.INC"
CLRF PORTB
BSF STATUS, 5 ; Bank 1' e geç
CLRF TRISB ; B portu çıkış
BCF STATUS, 5 ; Bank 1' e geç

BASLA
MOVLW h'09'
SUBLW h'1A' ; 1A sayısından W deki 09 u çıkar
MOVWF PORTB ; Fark Port B' ye yazıldı
MOVF STATUS, W ; STATUS Reg.' i W' ye aktarıldı

DONGU
GOTO DONGU
END
```

STATUS REGISTER

Bu register PIC 16F84 için 0x03 ve 0x83 adreslerinde bulunur. İçerdiği bitler şunlardır:

- C (bit 0)** :Elde Bayrağı denir. (En soldaki 7. bitte taşma varsa 1 olur).
- DC (bit 1)** : **Dijit Elde Bayrağı** adını alır. Alt 4 bit (nibble)' den 5. bite elde ya da borç geçişinde (1) olur.
- Z (bit 2)** : **Sıfır bayrağı** denir. Bir aritmetik yada mantık komutun sonucu (0) ise bu bayrak (1) olur.
- PD (bit 3)** :Enerji kesilme biti.
- PIC' e enerji verilince yada CLRWDT komutu çalışınca (1) olur.
 - SLEEP komutu ile (0) olur.
- TD (bit 4)** : Zaman doldu biti.
- PIC' e enerji verildiğinde CLR WDT ve SLEEP komutu ile (1) olur.
 - WDT zamanlayıcısı zamanı dolduğu vakit (0) olur.
- RP1 – RPO (bit 6 – bit 5)** : Bank seçme bitleridir. (00) yapılırsa Bank 0, (01) yapılırsa Bank 1 seçilir (16F84' de daima RP1 = 0 kalmalıdır).
- IRP (bit 7)** : PIC 16F84' de kullanılmaz ve daima (0) kalmalıdır.

Program Örneği 8: PORT B' ye (00)₁₆' dan başlayarak (08)₁₆' e kadar sayılan (00, 01, 02, ..., 08) (sayıcı gibi) yazan bir program PIC16F84 için assembly dilinde yazınız.

Cözüm: ; PROGRAM8.ASM----08 / 12 / 2003

```
LIST      P = 16F84
INCLUDE   "P16F84.INC"
SAYAC     EQU      h'0E'      ; Sayaç genel amaçlı RAM de ( 0E ) de olsun.
          CLRF      PORTB      ; Bank 1 e geç
          BSF        STATUS, 5
          CLRF      TRISB      ; Port B tamamı çıkış
          BCF        STATUS, 5  ; Bank 0 a geç

          BASLA
          CLRF      SAYAC

          TEKRAR
          MOVF       SAYAC, W    ; SAYAC 'yı W ' ye yükle
          MOVWF      PORT B      ; Port B den yolla
          INCF       SAYAC, F    ; SAYAC' ı 1 arttır
          MOVLW      h'09'
          SUBWF      SAYAC, W    ; ( SAYAC – W ) yi W ye yaz
          BTFSS      STATUS, Z   ; Z bayrağı 1 mı?
          GOTO       TEKRAR      ; Değilse TEKRAR' a

          DONGU
          GOTO       DONGU      ; Bekle
          END
```

ZAMAN GECİKTİRME DÖNGÜLERİ

Bazı programlarda GECİKME bloğu kullanıldığı yani hiçbir şey yapmadan bir süre beklenmesi gerektiği daha önce belirtilmişti.

Zaman geciktirme için ya yazılım ya da donanım imkanlarından yararlanmak gerekir. Yazılım kullanıldığında her komutun icra süresi bilinmeli ve ona göre toplam GECİKME hesaplanmalıdır. RC tipi osilatör kullanıldığında bu elemanların sıcaklığa bağlı değer değiştirilmelerinden dolayı hassas bir gecikme süresi elde etmek mümkün olmaz. Böyle bir durumda kristal / rezonatör kullanılması şarttır.

Komut Peryodu: PIC16F84 genellikle 4 MHz’ de çalıştırılır ve bu frekans içinde 4’ e bölünerek $f = 1 \text{ MHz}$ ’ lik **dahili frekans** elde edilir. Bu da

$$T = \frac{1}{f} = \frac{1}{1 \text{ MHz}} = 1 \mu\text{s} \quad \text{bir komut icra süresidir.}$$

Bazı komutlar ise 2 peryotta icra edilir. Bunlar :

GOTO	RETFIE
CALL	INCFSZ (Sayaç $\neq 0$ ise 1 peryot)
RETURN	BTFSC (Bit 1 ise 1 peryot)
DECFSZ (Sayaç $\neq 0$ ise 1 peryot)	BTFSS (Bit 1 ise 1 peryot)
RETLW	PC’ye (Prog. Sayacına) veri yazan komutla

Tek Döngü İle Maksimum ve Minimum Gecikme

Bir sayaç için “FF” yazarsak maksimum, “01” yazarsak minimum gecikme elde edilir. Aşağıdaki programda maksimum gecikme ($N = \text{FF} = 255$) sağlanmaktadır.

Komut	Peryot Sayısı
MOVLW h'FF'	1
MOVWF SAYAC	1
TEKRAR	
DECFSZ SAYAC, F	1 x 254 + 2
GOTO TEKRAR	2 x 254
	<u>Toplam : 766 peryot</u>

Örnek : 4 MHz bir osilatörden beslenen PIC 16F84’ de 250 μs gecikme sağlamak için bir GECİKME rutini (program parçası) yazalım.

$$\frac{4 \text{ MHz}}{4} = 1 \text{ MHz} \rightarrow 1 \mu\text{s} \quad \left\{ \begin{array}{l} \text{Bir dahili komut peryodu} \end{array} \right.$$

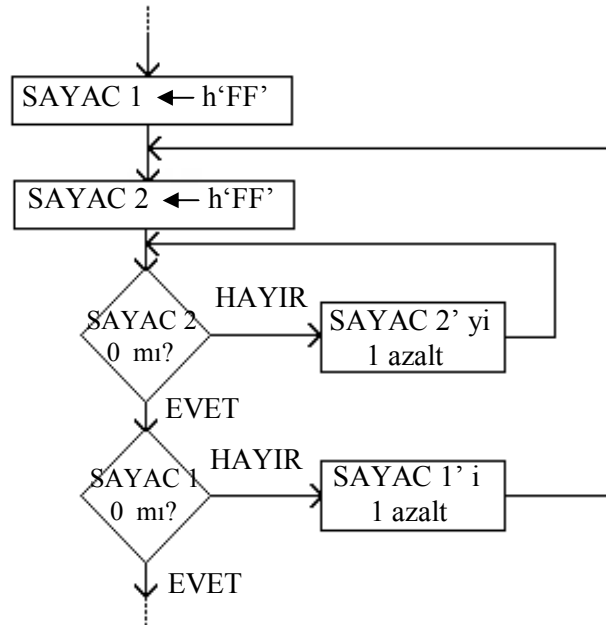
Komut	Peryot Sayısı	
MOVLW h'52'	1	MOVWF
SAYAC	1	
NOP	1	(56) ₁₆ = (82) ₁₆
NOP	1	
NOP	1	
TEKRAR		
DECFSZ SAYAC, F	1 x 81 + 2	
GOTO TEKRAR	2 x 81	
	<u>250 peryot</u>	

$$1 \times 250 = 250 \mu\text{s} \text{ gecikme süresi}$$

İçice İki Döngü İle Gecikme

SAYAC (FF) ile yüklense bile 766 peryot gecikme sağlanmakta idi. Bu süre az gelebilir. Bu durumda içice 2 yada daha fazla döngü kullanılabilir. Bu amaçla kullanılacak bir çift döngü rutin için akış diyagramı ve programı yapalım (süre maximum olsun).

İçice iki Döngü kullanan Program Algoritması ve Programı aşağıda verilmiştir. Böyle bir program için elde edilebilecek yaklaşık Gecikme süresi de tabloda verilmiştir.



Komut			Peryot Sayısı
	MOVLW	h'FF'	1
	MOVWF	SAYAC 1	1
TEKRAR1			
	MOVLW	h'FF'	1 x 255
	MOVWF	SAYAC 2	1 x 255
TEKRAR2			
	DECFSZ	SAYAC2, F	1 x 255 x 255
	GOTO	TEKRAR2	2 x 255 x 255
	DECFSZ	SAYAC1, F	1 x 255
	GOTO	TEKRAR1	2 x 255

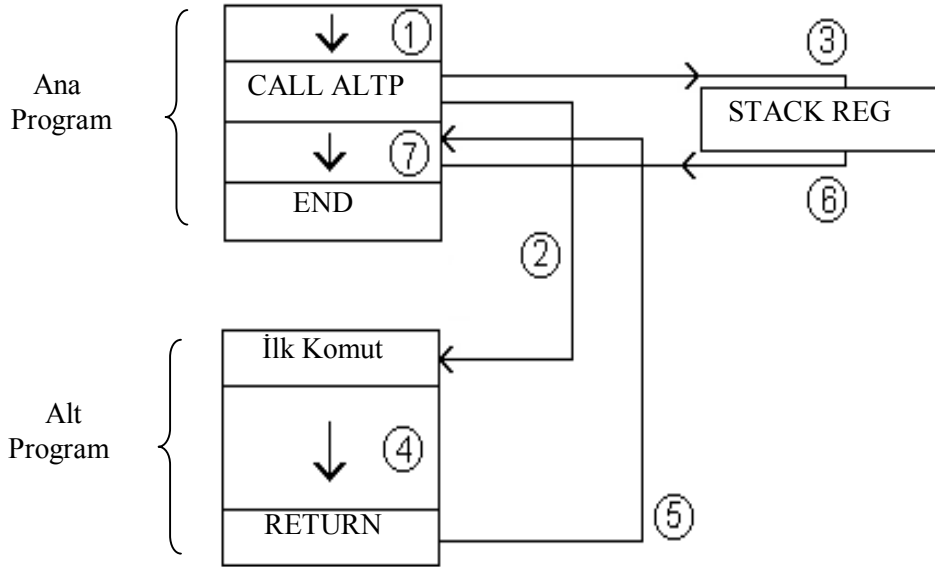
Toplam : 196606 peryot

1 μ s (mikro saniye)dahili komut peryodu için 196606. μ s = 0,196 saniye yaklaşık gecikme elde edilir. SAYAC 1 ile SAYAC 2 farklı da olabilir. Bu durumda (255) yerine o sayının karşılığı olan desimal sayı hesaba katılmalıdır.

ALT PROGRAMLAR

Program içerisinde birden fazla kullanılacak rutinler (program parçaları) varsa bunlar tekrar yazılmaz (Hafıza tasarrufu !). Bu durumda tekrar kullanılacak program parçaları **ALTPROGRAM** olarak düzenlenir . Ana (asıl) Programdan Alt Programa CALL komutu ile gidilir, RETURN komutu ile geri dönlür. CALL komutuyla atlanırken geri dönüş adresinin saklanması gerekir. Bunun için STACK REGISTER denilen özel bir registerden faydalanılır. Bu işlem otomatik yapıldığı için programcının bununla ilgilenmesi gerekmez.

Bir alt programın çalışmasını şematik çizersek ve numaralanmış olarak gerçekleştiren işlemleri gösterirsek :



Bu işlemleri sırayla yazarsak;

1. Ana program başlangıçtan itibaren çalışır.
2. CALL _ ALTP komutu ile Alt programın ilk komutuna atlanır.
3. Ana programdan ayrılma adresi STACK Registere otomatik olarak saklanır.
4. Alt program ilk komuttan itibaren icra edilir.
5. RETURN komutu ile karşılaşınca Alt programın bittiği anlaşılır ve Ana programda kaldığımız yere geri dönülür.
6. STACK Registerde saklanmış ayrılma adresi alınır.
7. Ana program devam eder ve normal olarak END komutu ile son bulur.

Şeklinde Alt Programa ait aşamalar elde edilir. Burada Programcı sadece CALL komutunu ve alt programı yazmak ve bunların isimlerinin (etiketin) aynı olmasını sağlamakla yükümlüdür. Alt Program sonuna RETURN yazılması, tüm programın (ana ve alt Programın) en altında END komutu bulunması zorunludur. Diğer işlemler otomatik olarak yapılmaktadır.