ASSEMBLY İLE ARM PROGRAMLAMA-2

1. GİRİŞ

Önceki hafta ARM Assembly dilini tanıyarak, üzerinde bellek, dallanma ve aritmetik işlemler gerçekleştirdik. Bu hafta ise sadece ARM'ı değil kartımızın çevre elemanlarından biri olan ledleri kullanacağız. Assembly ile ledleri nasıl kullanacağımızı, neler yapabileceğimizi öğreneceğiz.

2. LDR ve MOV komutu

Geçen hafta MOV komutu ile herhangi bir yazmaca anlık değer atayabileceğimizi, LDR komutu ile de bellekten veri yükleyebileceğimizi öğrenmiştik.

ARM mimarisinde buyruklar 32-bit uzunluğundadır. Yani MOV R0, #2 yazdığımızda bu makine diline çevrildiğinde ("0" ve "1"lerden oluşan) uzunluğu 32-bit olur. Bazı bitleri hangi komutun verildiğini anlatmaya, bazı bitler hangi yazmacın kullanıldığına, bazıları ise anlık değer için bırakılmıştır.

Bu mimaride MOV komutunda anlık değer için bırakılan alan 8-bit uzunluğundadır. Yani biz 0-255 arasında değerler atayabilmekteyiz. Daha fazlasını atmaya çalıştığınızda programınız hata verip derlenmeyecektir. Bunu düzeltmek için ARM mimarisinde şöyle bir yol izlenmiştir. Tüm 0-255 arasında bir değer verip bunu sağa kaydırmak. Örnek verecek olursak;

"MOV R0, #0x40,26" komutu "0x40" değerini 26 kez sağa kaydırarak 0x1000 değerini elde etmenizi sağlar. Yani komut "MOV R0, #0x1000" şeklini almıştır. Bu yöntemi kullanmak uğraştırıcı olduğundan, burada devreye LDR komutu girmiştir.

LDR komutu ile istediğiniz 32-bit genişliğinde **sabit sayıyı** yazmaca yazabilirsiniz. Kullanımı ise şöyledir;

LDR <yazmacınız>, =<sabit sayınız>

Bu hafta ise kullanacağımız bazı sistem değerleri 255'den büyük olduğu için MOV komutu yerine bu komutu kullanacağız. Bu konu hakkında daha detaylı bilgiyi bu bağlantının sayfa 33 ile 35 arasında bulabilirsiniz.

LDR kullanmak için sabit sayıya ihtiyacınız var. Sabit bir sayıyı ARM Assembly dilinde şu şekilde yazabilirsiniz;

<vereceğiniz isim> EQU <vereceğiniz değer>

Bir sonraki başlıkta buna ait örnekleri inceleyebilirsiniz.

3. STM32F4 Kartı Üzerindeki Çevre Elemanları

Kartımızda kullanıcının programlamasına bırakılmış 4 adet LED bulunmaktadır. Bu pinler GPIO ("General Purpose Input Output") D portu pinleri olup sırasıyla "PD12", "PD13", "PD14" ve "PD15"tir.

Bu portun pinleri ledlerin herhangi bir harici elektrik devresi sürülmemiş bir direnç yoluyla ledlerin anotlarına bağlanır. Bu nedenle bu pinleri arasına zaman gecikmesi koyarak yükseltip alçaltmamız gerekmektedir. Pini yükseltirsek LED açılacak, alçaltırsak LED'i söndürecektir. Daha detaylı bilgiler için kartın dokümantasyonuna[syf.16] bakabilirsiniz.

Geçen hafta hatırlayacağınız üzere "SystemInit" fonksiyonunu, kodu çalıştırmadan önce herhangi bir ayarlama yapmayacağımız için, boş bırakmıştık. Bu hafta ise pinleri ve saati ayarlamak için o fonksiyonun içini dolduracağız.

3.1. GPIO'nun Programlanması

GPIO'nun programlanması için öncelikle bazı yazmaç tanımlamaları yapmamız gerekiyor.

```
1 RCC_AHB1ENR   EQU  0x40023830
2
3 GPIOD_MODER   EQU  0x40020C00
4 GPIOD_OTYPER  EQU  0x40020C04
5 GPIOD_OSPEEDR EQU  0x40020C08
6 GPIOD_PUPDR   EQU  0x40020C0C
7 GPIOD_ODR     EQU  0x40020C14
```

RCC_AHB1ENR (RCC AHB1 peripheral clock ENable Register): Birçok çevresel elemanın saat desteğini ya aktif eder ya da devre dışı bırakır. "AHB1" ise bu çevresel elemanların işlemci ile arasındaki yolun protokolünü temsil eder. GPIO'nun D portunu kullanmamız için o bölümün saatini aktif etmemiz gerekli. Saati aktif etmek için de bu yazmacı kullanacağız.

GPIOD_MODER (GPIO-D MODE Register): D portu için hangi modun kullanılacağını kontrol eden yazmaç. Her pini bağımsız olarak aşağıdaki modlarda kullanabiliriz:

- Dijital Input,
- Dijital Output,
- Analog Fonksiyon,
- Mikrodenetleyici içindeki alternatif fonksiyonlar

GPIOD_OTYPER (GPIO-D Output TYPE Register): Pinin push-pull modunda mı yoksa open-drain bir pin olarak mı çalışacağını belirler. Open-drain modu için ekstra olarak dışardan pull-up veya pull-down yapılması gerekiyor. Biz ise push-pull modunu kullanacağız.

GPIOD_OSPEEDR (GPIO-D Output SPEED Register): Portta kullanılan pinin maksimum değiştirilme hızını belirler. Maksimum hız aynı zamanda besleme voltajına, pinden çekilen akıma ve yük kapasitörüne bağlıdır. Hız düşük/orta/yüksek/çok yüksek şeklinde ayarlanabilir.

GPIOD_PUPDR (GPIO-D Pull-Up/Pull-Down Register): D portunda bulunan her bir pin için pull-up veya pull-down iç direncini aktif eder.

GPIOD_ODR (GPIO_D Output Data Register): D portundaki pinlerinden çıkışa veri yazmak için kullanılan yazmaçtır.

STM32F4'de bulunan yazmaçlar memory-mapped şeklindedir. Yani her yazmacın kendisiyle ilişkili bir adresi bulunmaktadır. Yukarıda verdiğimiz değerler bu yazmaçlara erişmek için kullandığımız adreslerdir. Örneğin RCC_AHB1ENR yazmacı belleğin 0x40023830 adresinde yer almaktadır.

Kartınızın STM32F4xx Reference Manual dokümanından bu yazmaç değerlerinin nasıl seçildiğine erişebilirsiniz. Örnek olarak GPIO-D için verilmiş aralık syf.65'de yer alıyor. Syf.287'de ise yazmacın içinde neler olduğunu nasıl yerleştirildiğini görebilirsiniz.

Ledleri kullanmak için gereken yazmaçları da gördükten sonra, nasıl programlayacağımıza bakalım.

RCC_AHB1ENR (RCC AHB1 peripheral clock ENable Register)

Öncelikle GPIO-D portundaki pinler için saati aktif etmemiz gerekiyor. Bunu yapmak için RCC_AHB1ENR yazmacının bulunduğu adrese gidip, GPIOD için gerekli biti "1" yapmalı, daha sonra da aynı adrese bunu geri yazmalıyız.

```
1 LDR    R1, =RCC_AHB1ENR
2 LDR    R0, [R1]
3 ORR.W  R0, #0x08
4 STR    R0, [R1]
```

İlk satırda gereken adrese gitmek için 32-bit genişliğindeki veriyi ldr kullanarak R1 yazmacına yazıyoruz.

İkinci satırda ise R0 yazmacına R1 yazmacındaki verinin karşılık geldiği bellek adresinde (yani RCC_AHB1ENR yazmacının adresini) bulunan veriyi yazıyoruz.

Üçüncü satırda ise R0 ile "1000" değerini bitwise şeklinde or işlemi gerçekleştirip R0'a yazıyoruz. Bunun nedeni ise GPIOD portunun, RCC_AHB1ENR yazmacının 4.bitinde bulunması.

Aşağıdaki görselde yazmacın sahip olduğu bitlerin ayrıldığı bayrakları görebilirsiniz.

RCC AHB1 peripheral clock register (RCC_AHB1ENR)
(STM32F407)

31	30	29	28	27	26	25	24
Reserved	OTGHSULPIEN	OTGHSEN	ETHMACPTPEN	ETHMACRXEN	ETHMACTXEN	ETHMACEN	Reserved

23	22	21	20	19	18	17	16
Reserved	DMA2EN	DMA1EN	CCMDATARAMEN	Reserved	BKPSRAMEN	Reserved	Reserved

15	14	13	12	11	10	9	8
Reserved	Reserved	Reserved	CRCEN	Reserved	GPIOKEN	GPIOJEN	GPIOIEN

7	6	5	4	3	2	1	0
GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN

Son satırda GPIODEN bitini "1" yaptıktan sonra yazmacın bulunduğu adrese geri yazıyoruz.

GPIOD_MODER (GPIO-D MODE Register)

Saati ayarladıktan sonra sıra belirleyeceğimiz GPIOD portundaki LED pinleri çıktı olarak ayarlamak.

```
1 LDR    R1,  =GPIOD_MODER
2 LDR    R0,  [R1]
3 ORR.W  R0,  #0x55000000
4 AND.W  R0,  #0x55FFFFFF
5 STR    R0,  [R1]
```

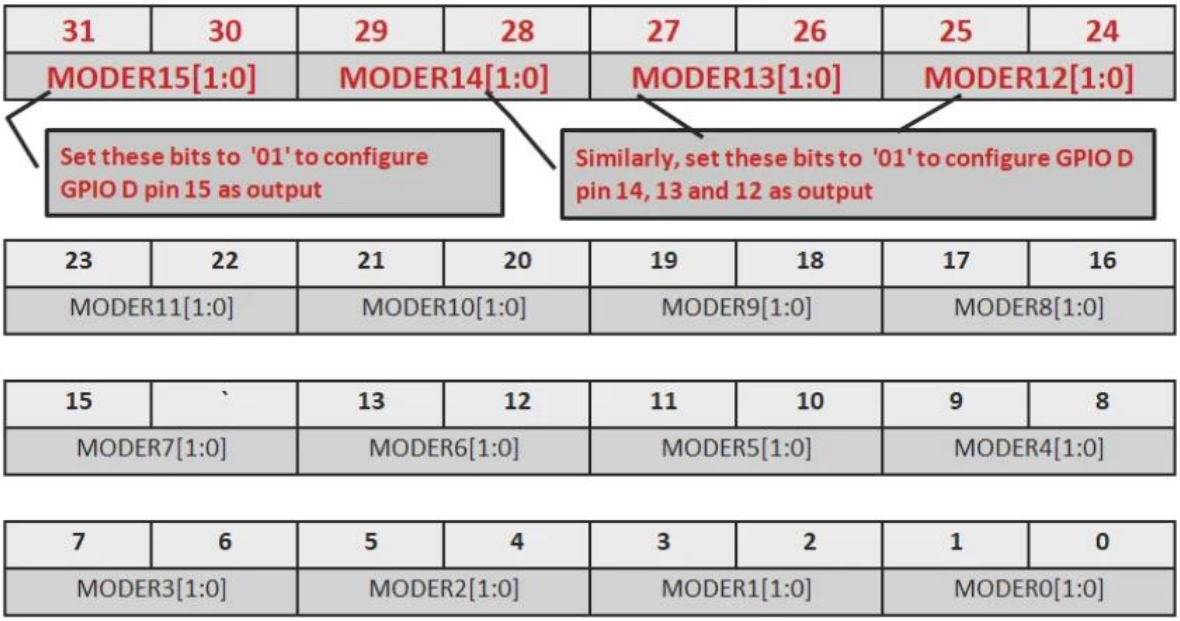
İlk iki satırda önceki yazmaca yaptığımız gibi adresten değeri okuyup R0 yazmacına yazıyoruz.

Üçüncü satırda ise "0x55000000" ile yazmacın değerini or işlemi yapmamızın nedeni GPIOD_MODER yazmacının 24. ve 25. bitlerinin "PD12"ye, 26. ve 27. bitlerininin "PD13", 28. ve 29. bitlerinin "PD14"e ve son olarak 30. ve 31. bitlerinin "PD15"e ayrılmasıdır.

Burada eğer bu bitlere "00" verirseniz input moduna, "01" verirseniz çıkış moduna, "10" verirseniz analog moduna, "11" verirseniz ise alternatif fonksiyon moduna sokmuş oluyorsunuz. Biz ledleri output olarak belirleyeceğimiz için o bitlere "01" değerini vermemiz gerekiyor. Bu nedenle "0x5500000" değerini binary olarak yazdığımızda "01010101000..00" olduğunu görüyorsunuz. Yani gerekli olan o 4 yer "01" değeri verilerek çıkış moduna alınmış oluyor.

Aşağıda ayrıntılı olarak yazmacın hangi bitte hangi bayrağa sahip olduğunu görebilirsiniz.

GPIO port D mode register GPIOD_MODER
(STM32F407)



Daha sonrasında 4.satırda AND kullanmamızın sebebi ise eğer 24-31 bitleri arasında "0" olması gereken ama "1" olarak kalan bitleri "0" olarak güncellemektir.

Son satırda da güncellediğimiz yazmaç değerini bir önceki yazmaç için yaptığımız gibi adresine geri yazıyoruz.

GPIOD_OTYPER (GPIO-D Output TYPE Register):

Pinlerimizi çıktı olarak ayarladıktan sonra "open-drain" olarak mı yoksa "push-pull" olarak mı seçeceğimizi ayarlayacağız. "Push-pull" herhangi bir dışardan kaynağa gerek duymadığı için onun seçilmesi işimizi kolaylaştıracaktır.

```
1 LDR    R1,  =GPIOD_OTYPER
2 LDR    R0,  [R1]
3 AND.W  R0,  #0xFFFF0FFF
4 STR    R0,  [R1]
```

Kodun ilk iki satırında daha önceki yazmaçlarda yaptığımız gibi o yazmacın adresindeki değeri R0 yazmacına atıyoruz. Push-pull "0" ile open-drain ise "1" ile ifade edilmektedir. Seçtiğimiz 12-15 arasındaki pinlerin yeri bu yazmaçta 12-15 bitleri arasında yer almaktadır.

Aşağıda ayrıntılı olarak yazmacın hangi bitte hangi bayrağa sahip olduğunu görebilirsiniz.

GPIO port D Output Type register GPIOD_OTYPER
(STM32F407)

31	30	29	28	27	26	25	24
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

23	22	21	20	19	18	17	16
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

15	14	13	12	11	10	9	8
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8

Clear these bits to configure push-pull output type on pins 12 to 15

7	6	5	4	4	2	1	0
OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0

Bitleri “0” yapabilmek ve geri kalan bitleri değiştirmemek için “0xFFFF0FFF” ile bitwise and işlemini gerçekleştiriyoruz.

Son satırda ise güncellediğimiz değeri bellekte yazmacın ifade edildiği adrese geri yazıyoruz.

GPIOD_OSPEEDR (GPIO-D Output SPEED Register)

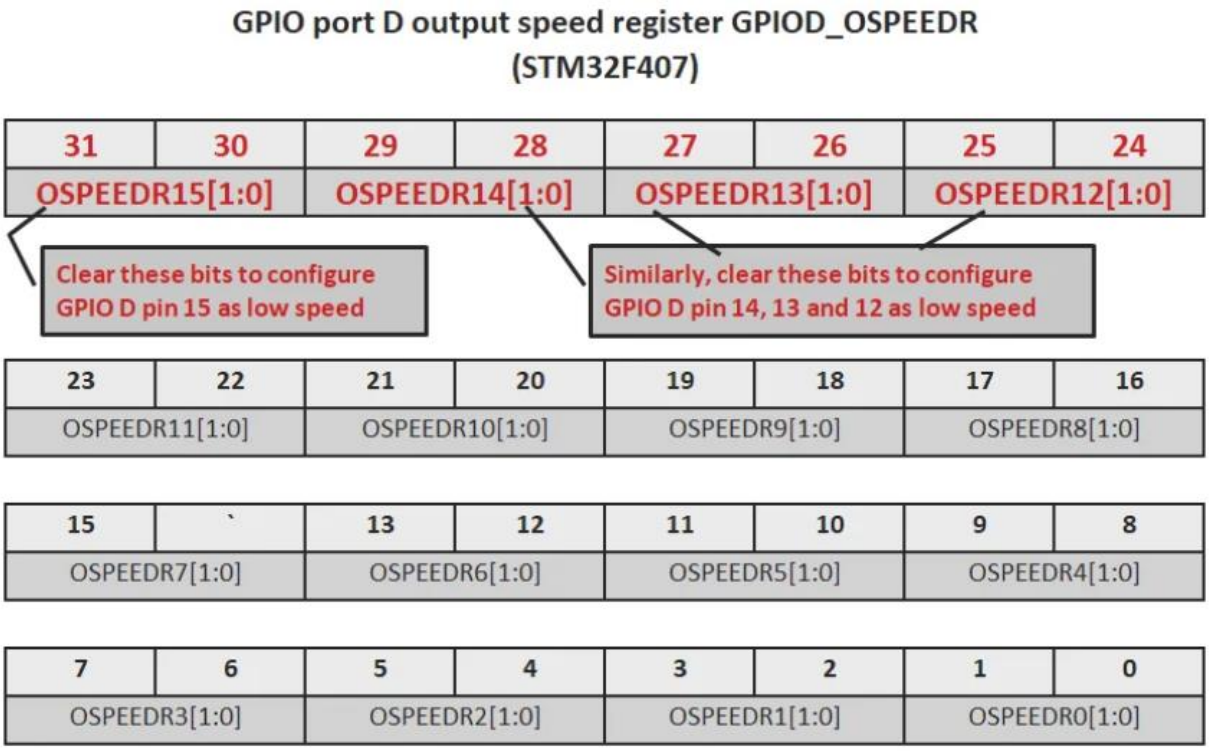
Çıkış pinlerinin hangi tip olacağını belirledikten sonra şimdi hızlarını belirleyeceğiz. Kartımızın dökümanına göre maksimum hız, yavaş hız seçeneği için, 2MHz ile 8MHz arasındadır. Gözle görebilmemiz için, örneğin 500ms’de bir yanmasını istiyorsak(2Hz), hız seçimimizi yavaş mod olarak belirleyeceğiz.

```
1 LDR    R1,  =GPIOD_OSPEEDR
2 LDR    R0,  [R1]
3 AND.W  R0,  #0x00FFFFFF
4 STR    R0,  [R1]
```

Önceki yazmaçlarda yaptığımız gibi ilk iki satırda yazmacımızın değerine erişip onu R0’a yazmacına yazıyoruz.

3.satırda ise “0x00FFFFFF” bitwise and işlemini yapmamızın sebebi 24-31 bitleri arasında seçtiğimiz pinlerin hız ayarları bulunmakta, yavaş hız olan “00” bitleriyle yazmacımızın 24-31 bitlerini güncelleyip, diğer bitleri değiştirmiyoruz.

Aşağıda ayrıntılı olarak yazmacın hangi bitte hangi bayrağa sahip olduğunu görebilirsiniz.



Son satırda ise güncellediğimiz değeri bellekte yazmacın ifade edildiği adrese geri yazıyoruz.

GPIOD_PUPDR (GPIO-D Pull-Up/Pull-Down Register)

Hızlarını da ayarladıktan sonra sıra iç dirençlerin kullanımına geldi. STM32F407 kartı her pin için ayarlanabilir pull-up/pull-down iç dirençleri sağlamaktadır. Bizim herhangi bir şekilde ledlere süreceğimiz pull-up veya pull-down olmadığı için, iç dirence ihtiyacımız bulunmamaktadır. Bu nedenle bu yazmacın karşılık gelen bitlerine "00" yazmalıyız.

```
1 LDR    R1,  =GPIOD_PUPDR
2 LDR    R0,  [R1]
3 AND.W  R0,  #0x00FFFFFF
4 STR    R0,  [R1]
```

İlk iki satırda daha önceki yazmaçlarda yaptığımız gibi bellekte o yazmaç için ayrılmış adrese gidip değerini R0 yazmacına yazıyoruz.

Bir sonraki satırda "0x00FFFFFF" ile bitwise and işlemi gerçekleştirmemizin nedeni ise 24-31 bitleri arasında bizim kullandığımız her pin için 2-bitlik bir bayrak alanı bulunmakta ve bunu "0x00FFFFFF" değeriyle "0" yazıyoruz ve geri kalan bitleri değiştirmiyoruz.

Aşağıda ayrıntılı olarak yazmacın hangi bitte hangi bayrağa sahip olduğunu görebilirsiniz.

GPIO port D Pull-up/Pull-down register GPIOD_PUPDR (STM32F407)

31	30	29	28	27	26	25	24
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]	

23	22	21	20	19	18	17	16
PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	

15	`	13	12	11	10	9	8
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]	

7	6	5	4	3	2	1	0
PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	

Bu sayede ana kodumuzu çalıştırmadan önce gerekli çevre elemanlarını programlamış olduk. Sırada programladığımız bu ledleri nasıl yakacağımızı ve söndüreceğimizi göreceğiz.

3.2 LED Yakma ve Söndürme

Ledleri yakmak veya söndürmek için şu ana kadar kullanmadığımız yazmaç olan GPIO_ODR yazmacını kullanacağız.

GPIOD_ODR (GPIO_D Output Data Register)

Yine önceki yazmaçlar için yapmış olduğumuz yazmacın bellekte karşılık gelen adresine ulaşıp yazmacın verisini R0 yazmacına yazıyoruz.

```
1  turnON
2      LDR     R1,   =GPIOD_ODR
3      LDR     R0,   [R1]
4      ORR.W   R0,   #0xF000
5      STR     R0,   [R1]
```

3.satırda değerimizi “0xF000” ile bitwise or işlemine sokmamızın nedeni ise GPIO_ODR yazmacının 12-15 bitlerinin PD12-PD15 pinlerine karşılık gelmesidir. Eğer karşılık gelen bite “1” değerini sürersek pin aktif hale gelecek, bizim durumumuzda led yanacak, eğer “0” sürersek de pin herhangi bir çıkış vermeyecektir, bizim durumumuzda ledin sönük olma durumu, bu nedenle 12. İle 15. bitlere 1 değerini vererek ledleri yakmış oluyoruz.

Eğer ledleri söndürmek istiyorsak bu durumda ORR.W yerine AND.W buyruğunu kullanıp R0 yazmacımızı "0xFFFF0FFF" ile bitwise and işlemi yapmamız gerekiyor.

Aşağıda ayrıntılı olarak yazmacın hangi bitte hangi bayrağa sahip olduğunu görebilirsiniz.

GPIO port output data register (GPIOx_ODR) (x = A..I/J/K)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy**: Port output data (y = 0..15)
These bits can be read and written by software.

3.3 Zaman Gecikmesi (Delay) Oluşturma

Zaman gecikmesi oluşturmak çoğu programlardaki kritik gereksinimlerden biridir. Bu haftaki uygulamamızda ledlerin belli aralıklarla yanıp sönmesini sağlamak için araya bir gecikme koymamız gerekiyor. Yani ledin pinini "1" yap, zaman gecikmesi koy, ledin pinini "0" yap şeklinde.

Zaman gecikmesi oluşturan aşağıdaki altyordamı inceleyelim.

```
    LDR   R2,  =DELAY_INTERVAL
delay1
    CBZ   R2,  turnOFF
    SUBS  R2,  R2, #1
    B     delay1
turnOFF
```

DELAY_INTERVAL: delay1 altyordamını kaç kere çalıştıracağımızı belirlediğimiz sabit sayı.

CBZ ise eğer yazmaç değeri "0" ise yazılan etikete atlayan değilse kodun bir sonraki satırdan devam etmesini sağlayan buyruk. Yani R2 eğer "0" olursa kod "turnOFF" etiketine atlayacak, değilse de SUBS buyruğundan devam edecektir.

B buyruğu ise herhangi bir koşul olmadan verilen etikete atlar. Yani kod o satıra geldiğinde tekrar delay1 etiketine geri dönecektir.

Kodun yapısını açıkladıktan sonra oluşturulan gecikmenin nasıl hesaplandığına bakalım. Öncelikle her bir buyruğun -derste görmüş olduğunuz konsept olan MC(Machine Cycle)- çevrim sayısını hesaplamamız gerekiyor. Daha sonra her bir çevrimin kaç ns sürdüğünü bildiğimizden her bir iterasyon için kodun ne kadar zaman gecikmesine neden olduğunu öğrenebiliriz.

Döngü 3 buyruktan oluşuyor. CBZ ve SUBS buyrukları tek çevrim, B buyruğu ise 3 çevrim sürmektedir. Buradan hangi buyruğun kaç çevrim sürdüğünü hesaplayabilirsiniz. Toplamda her bir iterasyonun 5 çevrim sürdüğünü öğrendiğimize göre her çevrim kaç ns sürmektedir onu görelim. Normal saat hızımız 16MHz olduğuna göre her bir çevrim 62,5ns sürmektedir. Yani her bir iterasyon (62,5ns*5) = 312,5ns ≅ 313ns sürmektedir.

Eğer biz 500ms'lik bir zaman gecikmesi yaratmak istiyorsak;

500ms/313ns = 1597444 olarak bulunur.

Yani DELAY_INTERVAL değişkenimizi sabit bir değer olarak,

```
DELAY_INTERVAL EQU 0x186004
```

bu şekilde tanımlamalıyız.