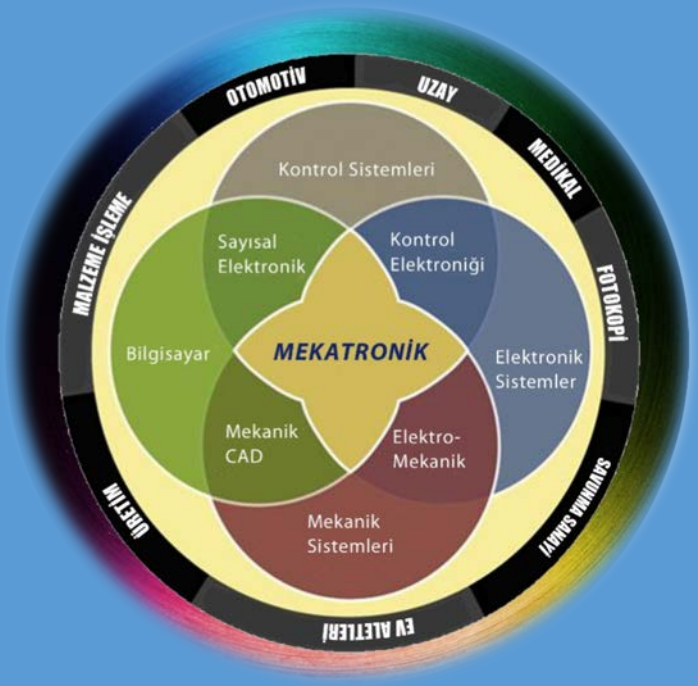
EGE ÜNİVERSİTESİ EGE MESLEK YÜKSEKOKULU

MEKATRONİK PROGRAMI

İLERİ MİKRODENETLEYİCİLER DENEY FÖYÜ



Mustafa Engin

2014

Contents

DENEY 1:
LCD GÖSTERGE KULLANIMI

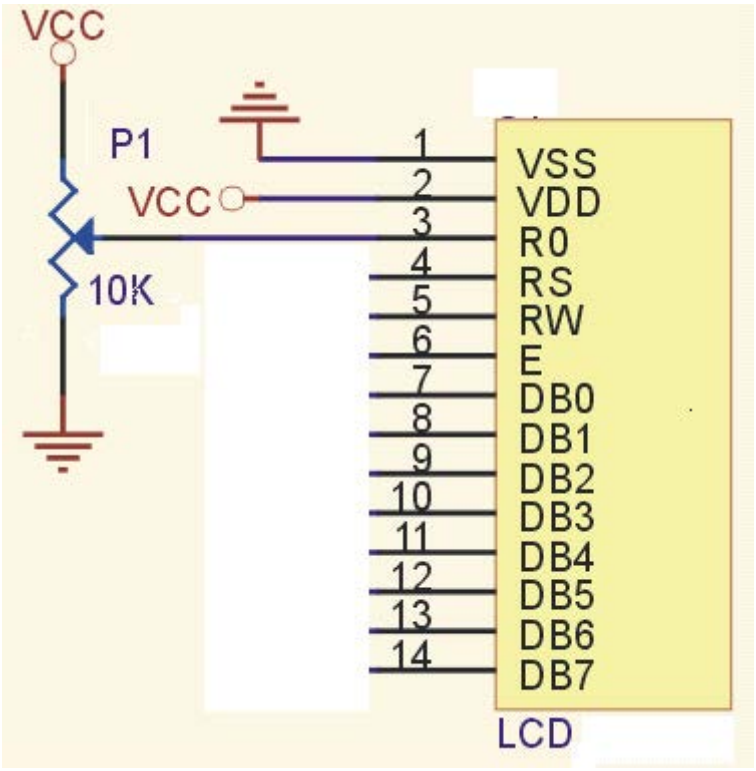
LED göstergelerin fazla akım çekmesi ve kullanım zorluğu, son yıllarda LCD göstergelerin kullanımını daha popüler hale getirmiştir. İstenilen karakterin daha iyi çözünürlükte elde edilmesi ve yeni üretilen LCD modüllerinin devreye bağlantısı ve programlanmasının kolay hale gelmesi diğer gösterge türlerine göre daha fazla kullanılmasını sağlamıştır. Özellikle karanlık ortamlarda kullanılamaması ilk zamanlarda bir sakınca olarak ortaya çıkmıştı, daha sonra arka plan aydınlatma ile bu sakınca ortadan kaldırılmıştır. Son yıllarda renkli LCD'lerde yaygın bir şekilde kullanılmaya başlamıştır. Yaygın kullanıma rağmen hala fiyatları yüksektir.

Bu deneyde kullanacağımız LCD aslında sadece bir göstergeden ibaret değildir. Akılı bir çevre birimidir, birden fazla karakter içermesi ve bunların belirli hızlarda taranma zorunluluğu üretici firmaları LCD'leri içerisinde osilatörü, sistem mikrodenetleyicisi, kod üreteci gibi birimleri içeren modül olarak üretmelerine neden olmuştur. LCD modüller en son yazılan bilgileri yenisi yazılana kadar göstergede görüntüledikleri için ana işlemciyi fazla meşgul etmezler.

LCD'ler 1 satırdan 4 satıra kadar, 16 karakterden 80 karaktere kadar ve 5X7, 5X10 nokta font gibi değişik ölçülerde üretilip satılmaktadır. Bazıları ise tüm ekran tek bir karakter gibi yapılandırılmıştır, bu türlerine grafik ekran adı verilmektedir. LCD modül ile iletişim TTL standardında 4 veya 8 veri hattı ile yapılır. 4 bit iletişim G/Ç hatlarının başka işler için kullanımını kolaylaştırırken iletişim süresini iki kat uzatmaktadır. LCD modüller veri hatlarının yanı sıra 3 ila 5 adet arası denetim hattına gereksinim duyarlar. Besleme mantık elemanları için 5 volt likit kristal sürücüler için ise farklı bir kaynağa gereksinim duyarlar. İkinci kaynak genellikle birinciden ayarlı potansiyometre yardımıyla elde edilir.

LCD Modül Bağlantı Uçları

LCD modüller üzerinde kullanılan denetleyiciler Hitachi firması tarafından üretilen HD44780U mikrodenetleyicisi standart oluşturmuştur. Bu standarttaki LCD modül bağlantı uçları tablo-8.1'de gösterilmiştir. Bağlantı uçlarını besleme, denetim ve veri olmak üzere üç grupta inceleyebiliriz.



Şekil-9.1 LCD gösterge bağlantısı.

Besleme Uçları

HD44780U standardında besleme ile ilgili üç uç yer almaktadır. Bunlar Vcc, Vee, Vss'dir. Vss ve Vcc standart TTL gerilimi 0 ve 5 Volttur. Vee ise likit kristal sürücü gerilimidir, değeri de Vcc'den daha küçük olur. 'dir. Vcc-Vee değerine V0 adı verilir, V0 5 Volttan büyük olabilir. Eğer Vee Vss'den daha negatif seçilirse V0 5 Volttan büyük olacaktır. Ortam sıcaklığı arttıkça Vee daha negatif yapılarak sıcaklık düzeltmesi yapılmalıdır. Laboratuar ortamında sıcaklık düzeltmesine gerek yoktur fakat dış ortamlarda üretici firmanın önerdiği sıcaklık düzetme bağlantısı yapılmalıdır.

Normal sıcaklık tipi ve güçlendirilmiş sıcaklık tipi olmak üzere iki tür LCD vardır. Normal sıcaklık türlerinde TN bileşimli sıvı, güçlendirilmiş sıcaklık türlerinde NTN bileşimli sıvı kullanılır. TN bileşimli sıvı kullanılan modüllerde görüş açısı 40 derece ile sınırlıdır. NTN bileşimli sıvılarda görüş açısı 70 derecedir. Normal sıcaklık modüllerde Vee gerilimi pozitiftir. Güçlendirilmiş sıcaklık modüllerde ise Vee gerilimi GND'ye göre –1 veya –2 Volt olduğunda daha iyi görüntü vermektedir.

Gösterge

Gösterge bir satırda 40 karaktere ve 4 satıra kadar geniş yapılabilir. Toplam HD44780U standardında 160 karakter denetlenebilir. Ticari olarak satılanlar 1X16, 2X16, 1X20, 2X40 gibi boyuttadırlar. Yazılan karakterler LCD denetleyicisinde yer alan 80 satırlık RAM bellekte tutulur. Bu RAM bellek display data RAM veya kısaca DDRAM

olarak adlandırılır. RAM 40 karakterlik iki satırdan oluşsa da gösterge penceresinde aynı anda her satırda 20 karakter görüntülenebilir. İkinci 20 karakter ise sanal gösterge olarak adland0ırılabilir. Sanal satırdaki karakterleri görüntülemek için gösterge sola doğru kaydırılmalıdır. Bu işlemin nasıl yapıldığı ileri ki bölümlerde anlatılacaktır.

UÇ No	Simge	Seviye	**İşlev**	**Açıklama**
1	V_{SS}	0	GND	
2	V_{CC}	+5 V	TTL Kaynak	Mantık devre elemanları için besleme kaynağı.
3	V_{ee}	Değişken	LCD Kaynak	LCD göstergenin besleme kaynağı.
4	RS	Y/D	Yazaç Seçme	Yüksek seviye olduğunda veri giriş çıkışı, düşük seviye olduğunda komut girişi yapılır.
5	R/W	Y/D	Oku / Yaz	Yüksek seviye okuma, düşük seviye yazma yapıldığını belirtir.
6	E	VURU	İzinleme	Modül içerisindeki işlemi başlatır. Mutlaka yüksek seviye vuru olmalıdır.
7	DB0	Y/D	Veri bit 0	
8	DB1	Y/D	Veri bit 1	
9	DB2	Y/D	Veri bit 2	
10	DB3	Y/D	Veri bit 3	
11	DB4	Y/D	Veri bit 4	
12	DB5	Y/D	Veri bit 5	
13	DB6	Y/D	Veri bit 6	
14	DB7	Y/D	Veri bit 7	

Tablo-9.1 LCD modülün bağlantı uçları ve görevleri.

Karakter Kümesi

LCD önceden belirlenmiş veya kullanıcı tarafından belirlenmiş karakterleri görüntüleyebilir. LCD denetleyicisinde bir ROM olan karakter üreteci bulunur, ve bu bellekte 192 karakter kayıtlıdır. Her karakterin tablo-8.2'de gösterildiği gibi bir kodu vardır ve bu karakterlere kodları kullanılarak ulaşılabilir. 96 adet ASCII karakter kendi

koduyla kullanılabilir. 64 adet Japon karakteri ve 32 adet özel karakter ve küçük harfler yer almaktadır. LCD denetleyicisi buna ek olarak kullanıcıların karakter tanımlayabilmeleri için 8 karakter saklayabilme kapasiteli karakter üretme RAM belleğe sahiptir. Bu belleğe kısaca CGRAM adı verilir. Kullanıcı tarafından belirlenen karakterler önce CGRAM'e kaydedildikten sonra buradan kullanılabilir.

Yüksek dörtlü													
Düşük dörtlü	0	2	3	4	5	6	7	A	B	C	D	E	F
0	(CGRAM1)		0	@	P	`	p						
1	(CGRAM2)	!	1	A	Q	a	q						
2	(CGRAM3)	"	2	B	R	b	r						
3	(CGRAM4)	#	3	C	S	c	s						
4	(CGRAM5)	$	4	D	T	d	t	Kana ve Latin alfabesi karakterleri					
5	(CGRAM6)	%	5	E	U	e	u						
6	(CGRAM7)	&	6	F	V	f	v						
7	(CGRAM8)	'	7	G	W	g	w						
8	(CGRAM1)	(	8	H	X	h	x						
9	(CGRAM2)	)	9	I	Y	ı	y						
A	(CGRAM3)	*	:	J	Z	j	z						
B	(CGRAM4)	+	;	K	[	k	{						
C	(CGRAM5)	,	<	L	¥	l	|						
D	(CGRAM6)	-	=	M	]	m	}						
E	(CGRAM7)	.	>	N	^	n	→						
F	(CGRAM8)	/	?	O	_	o	←						

Tablo-9.2 LCD karakter kümesi.

Denetim Hatları

EN, RW, RS olarak adlandırılan üç adet denetim hattı vardır. Enable (EN) hattı LCD'ye bilgi göndermek istediğimizi belirtmek için kullanılır. LCD ile iletişim kurmak isteyen program öncelikle bu hattın seviyesini yükseğe çekmelidir. Diğer iki denetim hattının seviyesi yapılacak işleme göre seçildikten sonra veri hatlarına veri yazılır ve EN hattı düşük seviye yapılır bu negatif geçiş LCD denetleyicisini işleme başlatır.Register select (RS) hattı gönderilen verinin komut mu yoksa veri mi olduğunu belirtmek için kullanılır. Bu hat DÜŞÜK seviye olduğunda veri hatlarındaki bilgi komut olarak alınır ve işletilir.

Eğer bu hat YÜKSEK seviye ise veri hatlarındaki bilgi karakter kodudur, bu kod alınır ve göstergede görüntülenir.

Read Write (RW) hattı oku yaz hattıdır. Bu hat DÜŞÜK seviye iken LCD'ye yazma yapılır. YÜKSEK seviye iken LCD yazaçları okunur. Okuma komutu birkaç tanedir, geri kalanı yazma komutudur. Genellikle uygulamalarda bu hat doğrudan GND'ye bağlanır.

Başlangıç Reseti

LCD modül gerilim uygulandığında kendisini kullanıma hazırlar. Kaynak LCD uyuşmazlığında bu işlem gerçekleşmeyebilir, bu durumda arka arkaya 3 adet 30h komutu ile ayarlama işlemi yazılım ile yapılabilir.

Function set:

Veri yolunun genişliği, karakter fontu ve göstergede kullanılacak satır sayısı belirlenir. Komutun açılışı şöyledir;

0 0 1 DL N F x x

bitlerin anlamları;

DL=0 ise 4 adet veri hattı, DL=1 ise 8 bit veri hattı kullanılacak.

N=0 1 satır, N=1 2 satır kullanılacak.

F=0 5X7 noktadan, F=1 5X10 noktadan oluşacak karakter fontu kullanılacak.

X; bu bitlerin değeri önemli değildir.

Entry Mode:

Okuma ve yazma işlemi sonrası imlecin ve göstergenin durumunu beliler. Genel kullanımı her yazma işlemi sonrası imlecin konumu bir arttırılırken gösterge sabit bırakılır. Bu kullanımda bir sonraki karakter bir sağ konuma yazılır. Komutun bitlerinin anlamı şöyledir;

0 0 0 0 0 1 I/D S

I/D=0 imlecin konumunu bir azalt (bir sola kaydır).

I/D=1 imlecin konumunu bir arttır (bir sağa kaydır).

S=0 gösterge sabit.

S=1 göstergeyi I/D bitine göre sağa veya sola doğru kaydır. Eğer I/D biti 1 ise gösterge sola doğru kayar, 0 ise sağa doğru kayar.

İmlec bir sonraki adımda karakterin yazılacağı konumu veya bir sonraki adımda okunacak karakterin konumunu gösterir. İmlecin işleyişi yukarıda entry mode ile belirlenmişti.

İmleci Evine Gönder (Home Curser)

İmlecin evi 0 adresli karakter konumudur. Bu tüm göstergelerde birinci satırın en soldaki karakterinin bulunduğu yerdir.

İmleci yerleştir (Move Curser)

İmleç DDRAM'ın her noktasına gönderilebilir. Gönderilen yer göstergenin görünen kısmında olmayabilir. İmleç istenilen yere adresi belirtilerek ile gönderilir. Komutun açılışı aşağıdaki gibidir.

1 A6 A5 A4 A3 A2 A1 A0

A0 –A6 DDRAM adresini belirtir. Birinci satırın adresi 00-27h aralığında ikinci satırın adresi 40h-67h aralığındadır.

İmleci Gizle/Göster/Kırpıştır (Hide/Show/Blink Curser)

İmleç görüntüden gizlenebilir, hala bir sonraki yazılacak karakter konumunu gösterir. İmleç başlangıçta alt çizgi olarak göstergede görünür, istenirse yanıp sönen karakter olarak ta görüntülenebilir. Yanıp sönen imleçte istenirse görünmez yapılabilir. Komut bunların dışında göstergeyi açıp kapatan bite de sahiptir. Komutun açılışı aşağıdaki gibidir.

0 0 0 0 1 D C B

D=1 gösterge açık, D=0 gösterge kapalı.

C=1 imleç açık, C=0 imleç kapalı.

B=1 bulunduğu konumdaki karakteri yakıp söndürür.

B=0 imleç sabit alt çizgi olarak görüntülenir.

İmleci Kaydır (Shift Curser)

İmleç veya gösterge sağa veya sola doğru kaydırılabilir. Komutun açılışı aşağıdaki gibidir.

0 0 0 1 S/C R/L x x

S/C=0 göstergeyi sabit tutar, S/C=1 göstergeyi kaydır.

R/L=0 sola, R/L=1 sağa doğru kaydır.

Göstergeyi Temizle (Clear Display):

Göstergeyi temizler. Gösterge temizlendiğinde tüm DDRAM satırlarına ASCII boşluk karakteri olan 20h yazılır. Komutun açılışı aşağıdaki gibidir.

0 0 0 0 0 0 0 1

display on/off (gösterge aç kapa) ve shift display (gösterge kaydır) komutları daha önce imleç komutlarında açıklanmıştı.

Karakter yaratma (character generator) RAM'i komutları kullanıcının kendine özgü karakteri yaratması için kullanacağı komutlardır.

Karakter Üretecinin Adresinin Ayarlanması (Set CGRAM address)

İmleç konumu belirle komutuna benzer şekilde çalışır. Adresi belirlenen konum bir sonraki yazmanın yapılacağı satırdır. CGRAM 64 satıra sahiptir. Komutun açılışı aşağıdaki gibidir.

0 1 A5 A4 A3 A2 A1 A0

A0-A5 CGRAM adresini temsil eder. CGRAM'ın kullanımı daha sonra tartışılacaktır.

Durum Gösterge Bitinin Okunması (Status Inquiry)

Durum denetlemesi olarak adlandıracağımız bu komut LCD denetleyicisinin yeni komut veya veri kabulüne hazır olup olmadığına bakmak için kullanılır. Aslında tek okuma yapan komuttur, eğer RW hattını GND'ye bağladığınızda bu komutu kullanamazsınız. Bu komutla okunan 8 bitten bir tanesi meşgul (busy) bayrağıdır diğer 7 bit ise en son işlem yapılan belleğin adres bilgisidir. Eğer DDRAM'den işlem yapıldı ise işlemi tamamlanan satırın adresi okunur. Bazen imlecin yeri bilinmediğinde bu komutla bulunabilir. Komutun açılışı aşağıdaki gibidir.

BF AC6 AC5 AC4 AC3 AC2 AC1 AC0

BF=1 ise denetleyici meşgul, BF=0 meşgul değil.

AC6-AC0 en son işlem yapılan belleğin adres bitleri.

Veri İşlemleri

Veri gönderme için RS hattı YÜKSEK seviyeye getirilmelidir. İlk karakter gönderilmeden önce adres ayarlanmalıdır. Veri CGRAM veya DDRAM belleklerine gönderilebilir.İmleç konumu işlem yapılacak DDRAM satırını gösterir. DDRAM'e veri gönderme işlemi move curser, shift curser, home curser veya reset display komutları sonrası yapılabilir. İmleç kapalı olsa bile imlecin konumu işlem yapılacak DDRAM satırını gösterir.

Entry mode göre imleç konumu her yazma işlemi sonrası bir azaltılacak veya arttırılacaktır. Eğer karakterler sağa doğru yazılmak isteniyorsa artırma seçilmiş olmalıdır. Sola doğru yazılmak isteniyorsa azaltma seçilmiş olmalıdır. Veri DDRAM'e RW hattı DÜŞÜK seviyeye çekilerek yazılır. DDRAM'den veri okumak için bu hat YÜKSEK seviyeye çekilmelidir. LCD ile veri ve komut iletimi 4 bit veya 8 bit ile yapılabilir. 4 bit veri iletimi seçildiğinde düşük değerli dört bit kullanılmaz. Komutlar veya veriler iki parçaya bölünür. Yüksek değerli dört bit önce sonrada düşük değerli dört bit gönderilir. Bir komut gönderme ancak iki adımda gerçekleştirilebilir. LCD'de reset sonrası 8 bit iletişim seçilmiş olur, 4 bit işleme geçmek için ayarlama sonrası hemen 20h bilgisi gönderilmelidir.

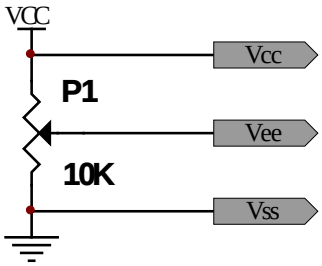
Zamanlama

LCD denetleyicisi normal işlemcilere göre oldukça yavaştır, verilen komutları yerine

getirmek için tablo-11.5'de belirtilen süreye ihtiyaç duyar. Bekleme yazılım zamanlayıcısı ile veya meşgul bayrağını kontrol ederek yapılabilir. Yazılım zamanlayıcısı kullanırsanız verilen sürelerin 1,5 katı kadar bekleme yaparsanız daha iyi olur.

Lcd Bağlantıları

LCD denetleyicisinin yavaş olmasında dolayı veri ve denetim hatları bellek haritalı mikroişlemcilerde veri yoluna ve RW hattına ve kod çözücüye bağlamak mümkün değildir. Eğer böyle bir bağlantı yapılıyor ise özellikle RW, EN ve RS işaretleri yavaşlatılarak LCD'ye uygulanmalıdır. 8051 bağlantısı portlar kullanılarak yapılacağından bu hatlar için gerekli gecikme yazılım ile yapılacaktır. Güçlendirilmiş sıcaklık LCD'ler için Vee geriliminin değeri ortam sıcaklığına bağlıdır. Şekil-11.9'da Vee geriliminin gerilim bölücü ile elde edilmesi gösterilmiştir. Vcc ile Vss arasına 5 Voltluk bir kaynak bağlanmalıdır. Vee'yi elde etmek için kullanılacak –7 Volt genellikle mikroişlemci kartlarında yer almaz. Fakat her mikroişlemci kartında 232C tümdevresi bulunur bu tümdevre +5 voltu kullanarak –9 Volt üretir. Mantık 1'i tanımlamak için bu gerilimi kullanır, T2X çıkışı genellikle kullanılmaz kullanılmadığında sürekli bu çıkış mantık 1 seviyesindedir. Dolayısıyla –9 Volt bu tümdevreden elde edilebilir. Uygun koyuluk trimpot ayarlanarak elde edilebilir. Sıcaklığa karşı değiştiğinde ayarlama işlem çalışma ortamında yapılmalıdır.



Şekil-9.1 normal sıcaklık LCD'lerin kontrast geriliminin elde edilmesi

Normal sıcaklık LCD'lerin negatif gerilime gereksinim duyulmadığından V0 gerilimin değeri 0<Vee<Vcc aralığında olmalıdır. Vcc ile GND arasına bağlanacak bir trimpotun orta ucunda V0 gerilimi elde edilebilir. Yine koyuluk ayarı bu trimpot ile çalışma ortamında yapılmalıdır. Şekil-8.2'de devresi gösterilmiştir.

Komut	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0	Açıklama
Göstergeyi temizle	0	0	0	0	0	0	0	0	0	1	Göstergeyi temizler imleci evine gönderir, kod üretecinin içeriğini değiştirmez.
İmleci evine gönder	0	0	0	0	0	0	0	0	1	x	İmleç birinci satırın en soldaki karakterin bulunduğu konuma gelir.

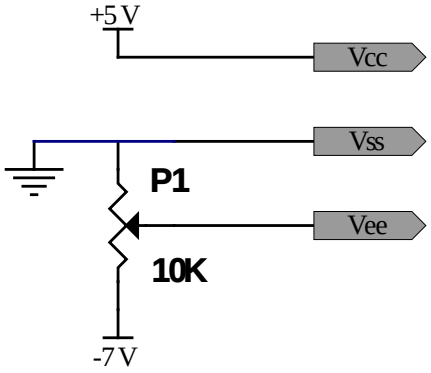
											Yazılacak yeni karakter imlecin bulunduğu yere yazılır.
Otomatik kaydır	0	0	0	0	0	0	0	1	ID	S	Karakter görüntülendikten sonra imlecin otomatik olarak kayma yönünü ve yazılı göstergenin içeriğinin kayıp kaymayacağını belirler.

Tablo-9.3'in devamı

Gösterge ve imleç aç kapa	0	0	0	0	0	0	1	D	C	B	D biti göstergeyi açar/kapatır, C biti imleci açar/kapatır, B biti imleci yakar söndürür.
Gösterge veya imleç kaydır	0	0	0	0	0	1	SC	RL	x	x	İmleç veya bulunduğu satırdaki karakterler belleğin içeriğini değiştirmeden kaydırılır.
İşlev seçme	0	0	0	0	1	DL	N	Z	x	x	DL biti veri hattı sayısının 4 veya 8 olarak seçerken, N biti gösterge birden fazla satıra sahipse hangi bir veya çok satır seçimini yapar.
Karakter üreteci adres belirleme	0	0	0	1							Oluşturulacak karakterin adresini belirler.
Veri belleği adres belirleme	0	0	1	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Veri belleği adresini belirler aynı zamanda imlecin konumunu da belirlemiş olur. Bir sonraki karakter bu adreste görüntülenir
Meşgul bayrağı ve imleç adresi oku	0	1	BF	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Meşgul bayrağı okunur. Bu bayrak 1 ise LCD yeni komuta daha hazır değildir. 0 olduğunda yeni komut veya veri kabul eder. Bu komut aynı zamanda imlecin bulunduğu konumun adresini de okur.
Veri yaz	1	0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Veri belleğine gösterilecek karakterin ASCII karşılığı veya karakter üretecine karakter bilgisi yazılır.
Veri oku	1	1	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Veri belleği veya karakter üretecinin içeriği okunur.

Tablo-9.3 LCD komutları.

R/S	R/W	D7	D6	D5	D4	D3	D2	D1	D0	
0	1	BF	*	*	*	*	*	*	*	Poll the "Busy Flag"



Şekil-9.2 Güçlendirilmiş sıcaklık LCD'ler için Vee geriliminin elde edilmesi

İşlem Sırası

1. Şekil-9.4'teki bağlantıları yapınız, LCD bağlantısını uç numaralarını kullandığınız LCD'nin veri yaprağına bakın. Eğer yoksa kart üzerinde 1 ve 14 nolu bacakların numaraları yazabilir, bu yazılara göre bağlantıyı yapın. 1 veya 14 numara belirlendiğinde sıra tüm LCD'lerde aynıdır. Emin olmak için LCD kılıfının metal kısmı ile her iki kenardaki bacaklara ölçü aletinin uçlarını ohm kademesinde değdirdiğinizde birinde mutlaka kısa devre gösterecektir. Bu uç 1 numara (GND) ucudur. Besleme uçları ters bağlanırsa veya gerilim var iken kartın arkasında yer alan işlemcinin bacaklarına çıplak elle dokunursanız LCD'nin denetleyicisi bozulacaktır. Bozulan LCD'de reset sonrası ya tüm karakterler dolu olarak görüntülenir veya hiç bir şey görüntülenmez. Bazılarında yarıya kadar tüm karakterler dolu diğer yarısı boş görünebilir.

Not: Tek satır ile çift satır LCD modüllerin 1 numaraları ters köşelerde olabilir.

2. Aşağıdaki tanımlamaları yapın. Bu tanımlamalar yapıldığında program içerisinde LCD uç isimlerini kullanabileceksiniz.

```
EN EQU P3.7
```

```
LRS EQU P3.6
RW EQU P3.5
veri EQU P1
```

3. Denetim hatların seviyelerini her komut öncesi belirlememiz gerekir.

RW hattı yazma yapılırken;

CLR RW

Komutu ile DÜŞÜK seviye yapılır. Okuma sırasında ise;

SETB RW

Komutu ile YÜKSEK seviye yapılır.

EN hattı kullanımı en karmaşık olan hattır. İşleme başlamadan önce bu hat;

SETB EN

Komutu ile YÜKSEK seviye çekilir. Diğer hatlar ve yazılacak veri hazırlandığında;

CLR EN

Komutu ile DÜŞÜK seviye yapılarak LCD mikrodenetleyicisin işlem yapmaya başlaması sağlanır.

RS hattı komut göndermede;

CLR RS

Komutu ile DÜŞÜK seviye yapılır. Veri göndermede;

SETB RS

Komutu ile YÜKSEK seviye yapılır.

4. LCD'ye komut yazıldığında işlenmesi belirli bir süre gerektirir. Yeni bir komut veya veri ancak bu sürenin bitiminde gönderilebilir. süre iki şekilde beklenir. Bu süre komutlara göre farklılık gösterir. Eğer bu süre zaman geciktirme döngüsü ile beklenecek olursa birden fazla zaman geciktirme döngüsü yazılacaktır. Diğer taraftan bu süre LCD modülün üzerine bağlanan kristal frekansı ile farklılık gösterebilmektedir. Bu farklılıklar göz önüne alındığında ikinci seçenek kullanılarak LCD meşgul bayrağının test edilmesi daha az hataya sebep olur. Bayrağı test eden programı yazalım.

```
bekle_LCD:
        SETB EN             ;LCD komutunu başlat.
        CLR RS              ;bu bir komut.
        SETB RW                     ;okuma komutu.
        MOV veri,#0FFh      ;tüm hatları giriş için 1 yap.
```

```
MOV A, veri                 ;LCD'yi oku.
JB ACC.7, bekle_LCD         ;eğer bit 7 yüksek ise, LCD hala meşgul.
CLR EN                      ;komutu bitir.
CLR RW                      ;RW hattını yaz konumuna getir.
RET
```

Yukarıdaki programın sakıncası donanım hatası yüzünden sonsuz döngüde kalmasıdır. Bunu engellemek için döngüye kısıtlama getirilebilir. Donanımda hata oluştuğunda zaten sistemin resetlenmesi gerekir ve döngüden çıkılır.

5. LCD'ye veri gönderilmeden önce gösterge, imleç ayarlamaları ve iletişim protokolü belirlenmelidir. İşe iletişim protokolü ayarlamak ile başlanır. Bazı ayarlama işlemleri birkaç komutun birleştirilmesi ile tek komut ile yapılabilir.

```
SETB EN
CLR LRS
CLR RW
MOV veri, #38h
CLR EN
ACALL bekle_LCD
```

Mov veri, #38h komutu function set 20h, 8 bit veri iletimi için 10h, ve iki satır kullanılacağını belirten 08h komutlarının birleştirilmiş halidir.

6. Ayarlama daha bitmemiştir yukarıdaki programı 0Eh komutunu ekleyelim.

```
SETB EN
CLR LRS
MOV veri, #0Eh
CLR EN
ACALL bekle_LCD
```

0Eh komutu 08h komutu tekrarı, 04h LCD aç komutu ile 02 imleç görünsün komutlarının birleşimidir.

7. Ayarlamaya devam edelim. İmleç her yazma işlemi sonrası bir sağa doğru kaysın gösterge sabit kalsın.

```
SETB EN
CLR LRS
MOV veri, #06h
CLR EN
ACALL bekle_LCD
```

06h komutu, 04h ile 02h komutlarının birleşimidir. İmleç konumu her yazma sonrası bir sağa otomatik olarak kaydırılacak şekilde ayarlanmış olur.

8. Tüm ayarlama programı aşağıdaki gibi olmalıdır.

```
ayar_LCD:
    SETB EN
    CLR LRS
    CLR RW
    MOV veri, #38h
    CLR EN
    ACALL bekle_LCD
    SETB EN
    CLR LRS
    MOV veri,#0Eh
    CLR EN
    ACALL bekle_LCD
    SETB EN
    CLR LRS
    MOV veri,#06h
    CLR EN
    ACALL bekle_LCD
    RET
```

9. Gösterge ayarlama işlemleri tamamlandığında temizlenir ve imleç birinci satırın en soldaki konumunda yer alır. Ayarlama sonrası gösterge temizlenmek istendiğinde aşağıdaki alt program kullanılabilir.

```
temiz_LCD:
    SETB EN
    CLR LRS
    MOV veri, #01h
    CLR EN
    ACALL bekle_LCD
    RET
```

10. Artık LCD'ye karakter gönderebiliriz. Aşağıdaki alt program akümülatörün içeriğini imlecin bulunduğu konuma yazar.

```
yaz_text:
```

```
SETB EN
SETB LRS
MOV veri, A
CLR EN
ACALL bekle_LCD
RET
```

Komut göndermeden farklı olarak sadece RS hattı YÜKSEK seviye yapıldı.

11. Şimdi LCD'ye "ILK DENEME" sözcüklerini yazdıralım.

```
LCD:
    ACALL ayar_LCD
    ACALL ayar_LCD
    ACALL ayar_LCD
    ACALL temiz_LCD
    MOV A,#'I'              ;I'nin ASCII kodunu yükle
    ACALL yaz_text
    MOV A,#'L'
    ACALL yaz_text
    MOV A,#'K'
    ACALL yaz_text
    MOV A,#' '
    ACALL yaz_text
    MOV A,#'D'
    ACALL yaz_text
    MOV A,#'E'
    ACALL yaz_text
    MOV A,#'N'
    ACALL yaz_text
    MOV A,#'E'
    ACALL yaz_text
    MOV A,#'M'
    ACALL yaz_text
    MOV A,#'E'
    ACALL yaz_text
    SJMP $
```

Bu programı çalıştırdığınızda göstergenin birinci satırının sol köşesinde "ILK

DENEME” sözcükleri görüntülenecektir.

12. Eğer elinizdeki LCD iki satır ise “DENEME” sözcüğünü ikinci satırın beşinci karakterine yazdıralım. 44780 denetleyicisinin DDRAM bellek haritası aşağıda gösterildiği gibidir. Beyaz olarak gösterilen kısım 2X16 karakter LCD modülün görünen kısmıdır. Bir karakterleri bu belleğe yazarız. Denetleyici ise onu aynı adres numaralı göstergede görüntüler. Birinci satır dolduğunda bir sonraki karakter otomatik olarak ikinci satıra geçmez. Bellek adresi bir satıra 40 karaktere kadar genişletme olanağı sağlayacak şekilde düzenlenmiştir. Birinci satırın on yedinci karakterin adresi 10h’dir fakat ikinci satırın ilk karakterinin adresi 40h’dir. İkinci satıra geçmek için imleç konumu ikinci satırın adresine konumlandırılmalıdır. İkinci satırın beşinci karakterinin adresi 44h’dir. İmleç konumu değiştirme komutu ise 80h’dir. Adres ve bu komutu birleştirirsek 11000100B sayıcı (C4h) elde edilir. Buna göre programı değiştirelim.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16					
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	54

DDRAM’in bellek haritası

Değerlendirme Soruları

1. Gruptaki kişilerin numaraları ve ad soyadları sağdan sola sürekli akan yazı olarak gösteren programı yazın.

2. Ğ, ğ, ç, ö, ü, ı, ş karakterlerini karakter RAM’inde tanımlayan programı yazın.

DENEY 2: KESMELERİN DENETİMİ

İşlem Basamakları

1. Dış kesme 1 geldiğinde LCD göstergede KESME 1, dış kesme 0 geldiğinde KESME 0 yazan programı yazın.

2. Seri portu 8 bit 4800 Baud rate hızında bilgisayar ile iletişim yapacak şekilde ayarlayın.

3. RI kesmesini izin vererek veri alan ve aldığı veriyi iç bellekte saklayan programı yazın.

4. TI kesmesini izin vererek veri gönderen programı yazın.

5. 2. 3. 4. adımdlardaki programları birleştirin. Seri kablo ile yandaki deney arkadaşlarınızla setlerinizi birbirine bağlayın. Seri porttan ‘A’ karakterini bekleyen geldiğinde karşı tarafa “B” karakteri göndersin.

6. Seri portu kayar yazaç modunda ayarlayarak 8 bit kayar yazaca bağlayın. DOT MATRIX göstergeye kayar yazaçtan sürerek bir karakter elde edin. (seri port kesmesini kullanın)

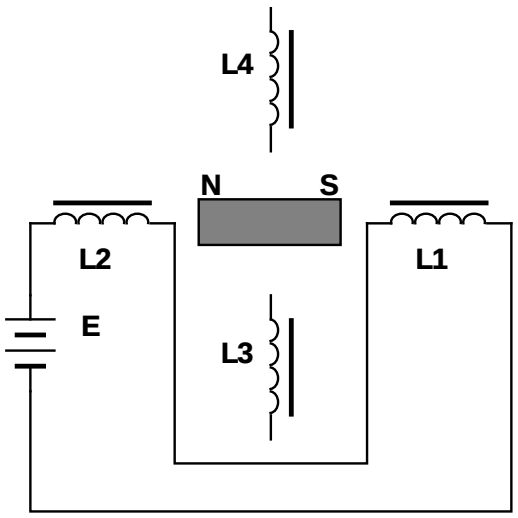
DENEY 3:
ADIM MOTORU DENETİMİ

Mikroişlemcinin işlediği verilerin sonuçları mekanik bir işarete dönüştürülmek istendiğinde motor kullanılır. Mikroişlemciden elde edilen denetim işareti basit bir devre yardımıyla adım motoruna uygulanabildiğinden bir çok uygulamada bu tür motorlar tercih edilir. Özellikle düşük gücün yeterli olduğu uygulamalarda adım motorunun kullanımı yaygındır. Adım motorunun dışında yüksek gücün gerektiği uygulamalarda, doğru akım motoru ve asenkron motor da kullanılır. Bu bölümde adım ve sabit mıknatıslı doğru akım motorlarının kontrolü yapılacaktır. bu motorların ileri yön, geri yön hareketleri, yarım ve tam adım çalışma kipleri, hızlanma ve pozisyon denetleme konuları incelenecektir.

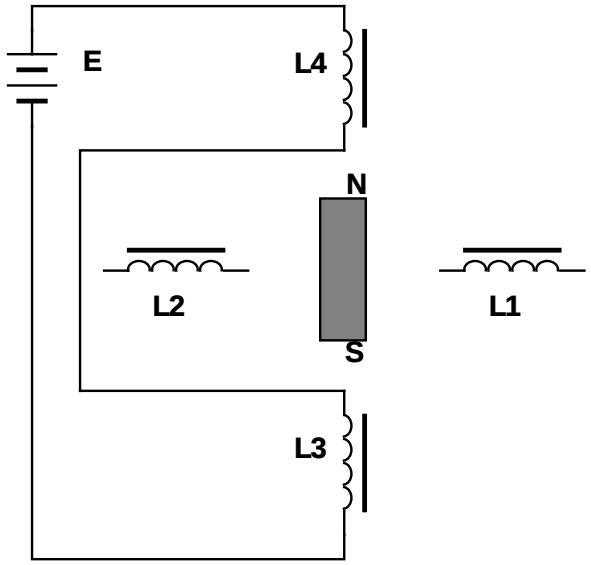
Adım Motorunun Yapısı

Adım motorları hassas konum denetiminin önemli olduğu uygulamalarda yaygın olarak kullanılır. Örnek olarak disk sürücüler, robot eklemleri, yazıcılar, tape sürücülerini verebiliriz. Motorun yapısı oldukça basittir, sabit mıknatıslı rotor ve iki sargısı olan statordan oluşur. Bobin üzerinden geçirilen akımın yönüne bağlı olarak stator sargılarında oluşan manyetik kutuplar ve rotorun manyetik kutupları motorun rotorunun pozisyonunu belirler. Stator sargılarından geçen akımın yönü değiştirilerek rotorun konumu değiştirilebilir. Eğer akım yönünün değiştirilmesi düzenli bir şekilde yapılırsa rotor istenen yönde haraket ettirilebilir.

Şekil-11.1'de C2 ve C1 sargılarına enerji verilmiştir, oluşan manyetik kutuplar sabit mıknatıs olan rotoru şekildeki pozisyona getirmiştir. şekil-9.2'de gösterildiği gibi C3 ve C4 bobinlerine enerji uygulanırsa oluşan manyetik kutuplar rotorun yeni konum almasına neden olur ve rotor 90 derece dönmüş olur.



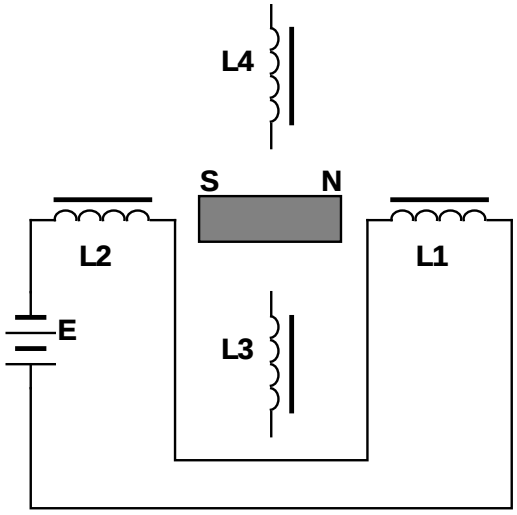
Şekil-11.1 Tam adım çalışmada 1. adımın konumu.



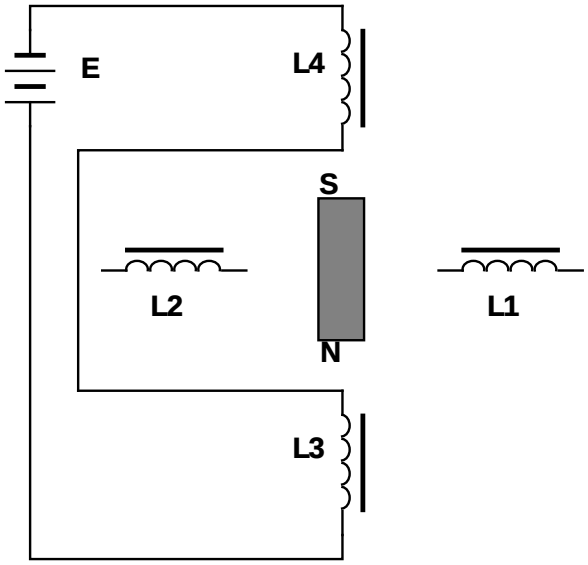
Şekil-11.2 Tam adım çalışmada 2. adımın konumu.

Kutuplardaki manyetik yönler kutup sargılarından geçen akımın yönü ile belirlenir. Sonuç olarak ta enerji uygulanan sargılar ve sargılar üzerinden geçen akımın yönü rotorun konumunu belirler. Şekil-11.3'te gösterildiği gibi tekrar C2 ve C1 bobinlerine bu kez ters yönde enerji uygulanırsa saat ibresi yönünde rotor bir 90 derece daha döner. Şekil-11.4'te gösterildiği gibi yine C3 ve C4 sargılarına fakat bu kez adım 2'dekinin tersi yönde enerji uygulanırsa rotor adım 3'e göre 90 derece daha saat ibresi yönünde döner. Böylece rotor tam bir devir atmış olur.

Yukarıda anlatılan adımlar sırayla istenilen hızda yapılarak hassas konum denetimi yapılabilir. Adım motorun sargıları doğrudan port hatlarından sürülemez, hatlar yeterli akımı sağlayamazlar. Şekil-11.5'te ki devrede görüldüğü gibi transistörlerle sürülür. Transistör yerine adım motoru sürücü tümdevreleri de kullanılabilir. Transistör seçimi yapılırken kollektör akımının sargıların çektiği akımdan daha yüksek olmasına dikkat edilmelidir. Devrede kullanılan diyotlara "fly back" diyodu adı verilir. Görevi motor sargılarında biriken enerji birikme yönüne ters yönde boşalmak isteyecektir, fakat transistör buna izin vermez. Bu enerjinin boşalması için alternatif yol sunmaktır. Normal çalışmada diyotlar ters yönde bağlı oldukları için üzerlerinden akım geçmesine izin vermeyeceklerdir.



Şekil-11.3 Tam adım çalışmada 3. adımın konumu.



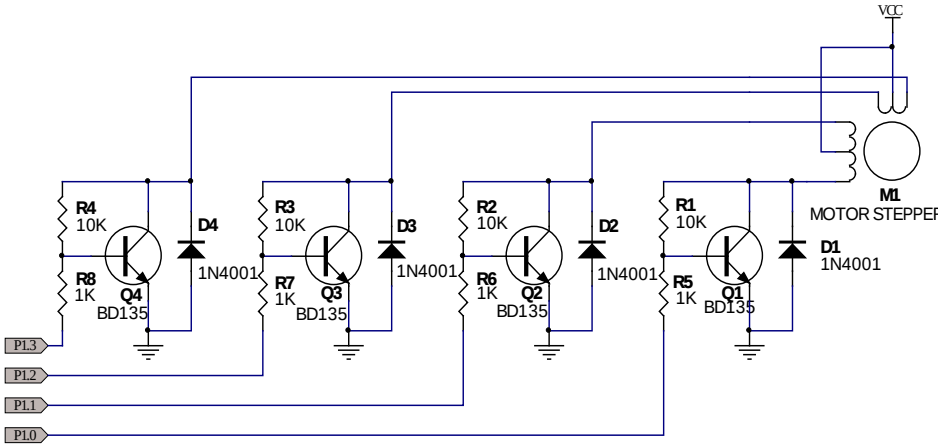
Şekil-11.4 Tam adım çalışmada 4. adımın konumu.

Adım	Q1	Q2	Q3	Q4	**DEĞR (HEX)**
1	İletimde	Kesimde	İletimde	Kesimde	A
2	İletimde	Kesimde	Kesimde	İletimde	9
3	Kesimde	İletimde	Kesimde	İletimde	5

4	Kesimde	İletimde	İletimde	Kesimde	6
1	İletimde	Kesimde	İletimde	Kesimde	A

Tablo-11.1 Tam adım çalışma için transistörlerin durumu.

Adım motorları tam ve yarım adım modlarında çalıştırılabilirler. Tam adım çalışmada bir adımda rotor 90 derece dönerken, yarım adım çalışmada 45 derece dönecektir. Tablo-11.1'de tam adım için sargılara uygulanması gerekirken gerilim seviyeleri gösterilmiştir. Sargının bağlı bulunduğu transistörün beyzine YÜKSEK seviye uygulandığında o sargıdan akım geçecektir. Tablo-11.2'de ise yarım adım için sargılara enerji uygulama sıraları verilmiştir. İletimde olan transistörün beyzine mantık 1 uygulanması gerekir. Transistör iletimde olduğunda bağlı bulunduğu sargının bir ucu Vcc'ye bağlı bir ucu toprağa bağlı olduğundan üzerinden bir akım geçecektir. Kesimde olan transistörün beyzine mantık 0 uygulanması gerekir. Kesimde olan transistörün bağlı olduğu sargının bir ucu açık devre olduğundan akım geçmeyecektir. 4 adet sargıyı denetlemek için 4 hat yeterlidir.



Şekil-11.5 Adım motorunun sargılarının sürülmesi.

Adım	Q1	Q2	Q3	Q4	**DEĞR (HEX)**
1	İletimde	Kesimde	İletimde	Kesimde	A
2	İletimde	Kesimde	Kesimde	Kesimde	8
3	İletimde	Kesimde	Kesimde	İletimde	9
4	Kesimde	Kesimde	Kesimde	İletimde	1
5	Kesimde	İletimde	Kesimde	İletimde	5
6	Kesimde	İletimde	Kesimde	Kesimde	4

7	Kesimde	İletimde	İletimde	Kesimde	6
8	Kesimde	Kesimde	İletimde	Kesimde	2
1	İletimde	Kesimde	İletimde	Kesimde	A

Tablo-11.2 Yarım adım çalışma için transistörlerin durumu.

İşlem Sırası

1. Şekil-11.5'deki devreyi kurun, aşağıdaki programı yazın.

```
;*********************************************************
;bu program adım motorunu saat ibresi dönüş yönünde döndürür.
;*********************************************************
;kullanılan portlar.
        Q1      equ     p1.0
        Q2      equ     p1.1
        Q3      equ     p1.2
        Q4      equ     p1.3
        Mot     equ     p1
;adım verileri
        adm1    equ     0FAh
        adm2    equ     0F8h
        adm3    equ     0F9h
        adm4    equ     0F1h
        adm5    equ     0F5h
        adm6    equ     0F4h
        adm7    equ     0F6h
        adm8    equ     0F2h
org 0h
        Ajmp adım_motor
org 100h
adim_motor:
        mov p1, #adm1
        Acall bekle
        mov p1, #adm2
```

```
        Acall bekle
        mov p1, #adm3
        Acall bekle
        mov p1, #adm4
        Acall bekle
        mov p1, #adm5
        Acall bekle
        mov p1, #adm6
        Acall bekle
        mov p1, #adm7
        Acall bekle
        mov p1, #adm8
        Acall bekle
        sjmp adim_motor
;gecikme alt programı.
Org 200h
bekle:
        mov r1, 80h
dongu2:
        mov r0, 0
dongu1:
        djnz r0, dongu1
        djnz r1, dongu2
        ret
```

2. Programı assembly edip simulatörde çalıştırın hataları varsa düzeltin. Doğru çalıştığından emin olduktan sonra işlemciyi programlayıp devreye takarak çalıştırın.

3. Bu program adım motorunu saat ibresi yönünde döndürecektir. Siz bu programı saat ibresinin tersi yönde döndürmek için gerekli programı yazın. Aynı gecikme programını kullanın.

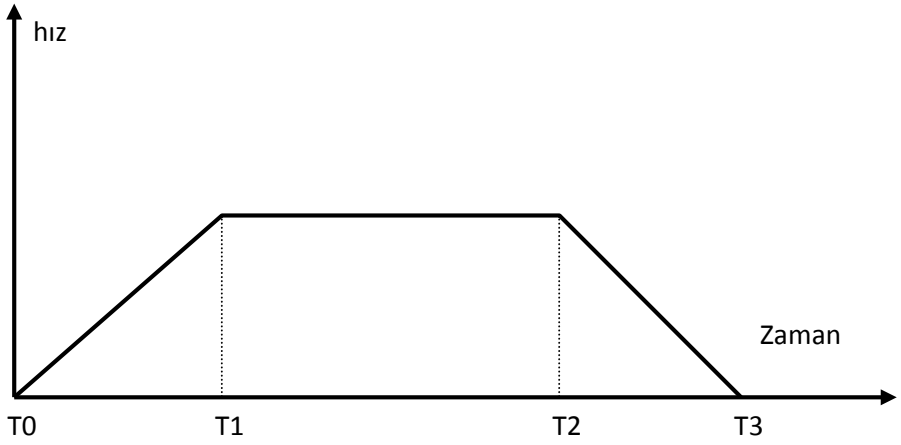
4. Adım motorunun hızını değiştirin. Zaman geciktirme döngüsündeki R1'in içeriğini değiştirerek yapabilirsiniz.

5. P1.4'de iki durumlu bir (dip switch) anahtar yerleştirin bu anahtarın durumu 1 ise motor saat ibresi yönünde, 0 ise saat ibresinin ters yönünde dönsün. Programı

aşağıda ayrılan yere yazın.

6. Zaman geciktirme alt programı kullanmak yerine, gecikmeyi T1 kesmesini kullanarak yapın. Programı yazıp çalıştırın.

7. Motorun hızı şekil-11.6'daki eğriyi takip edecek şekilde program yazın. Bu işlem için öncelikle motorun en yüksek hızını belirleyin. 0-T1 aralığında motor sabit ivme ile hızlansın, T1-T2 aralığında maksimum hızda çalışsın, T2-T3 aralığında ise 0-T1 aralığındaki ile aynı ivme ile yavaşlasın.

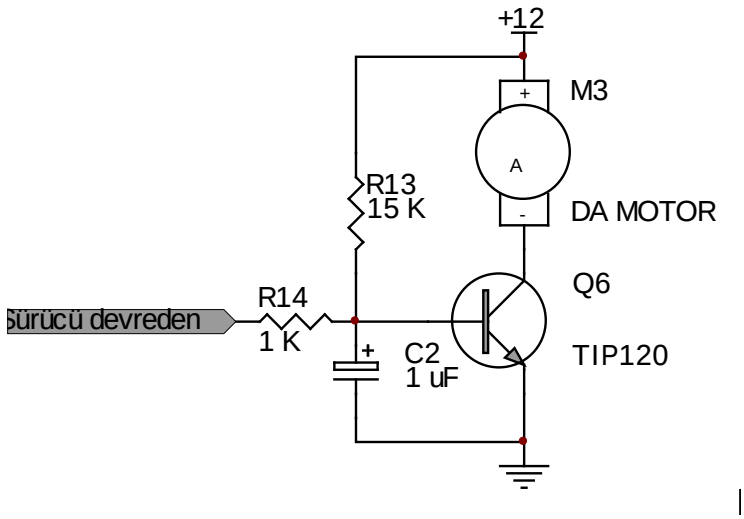


Şekil-11.6

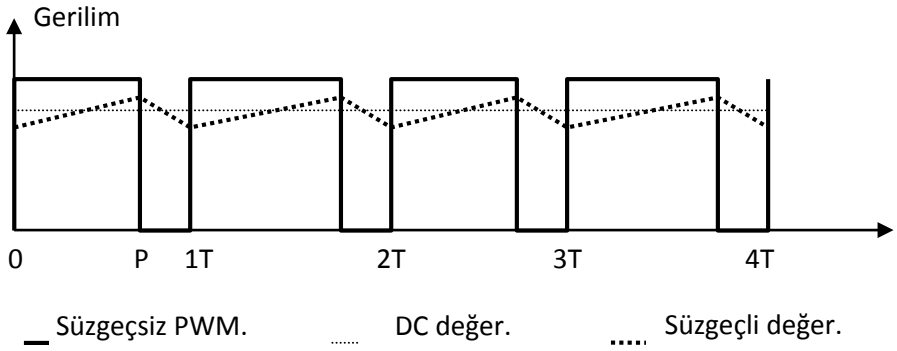
DENEY 4:
DA MOTOR DENETİMİ

DA Motorunun Yapısı

Sabit mıknatıslı doğru akım motorlarının hızları uçlarına uygulanan gerilim değeri ile denetlenebilir. Motorun yönü ise uygulanan gerilimin kutupları ile değiştirilir. Mikroişlemci ile sabit mıknatıslı doğru akım motorunun hızı iki türlü yapılır. Şekil-12.7'de gösterilen bağlantıda Q1 transistörünün kollektör akımı motor sargılarından da geçecektir. Eğer kollektör akımını ayarlarsak motorun hızını ayarlamış oluruz. Kollektör akımı ise bilindiği gibi beyz akımı ile ayarlanabilir. Beyz gerilimi değiştirilirse kollektör akımı ayarlanabilir. Beyz gerilimi değiştirmenin dışında transistör anahtar olarak kullanılarak transistörün iletimde kaldığı süre denetlenerek motor hız denetimi yapılabilir. Bu işlem için Vuru-genişlik modülasyonlu (PWM) işaret üretmek gerekir.



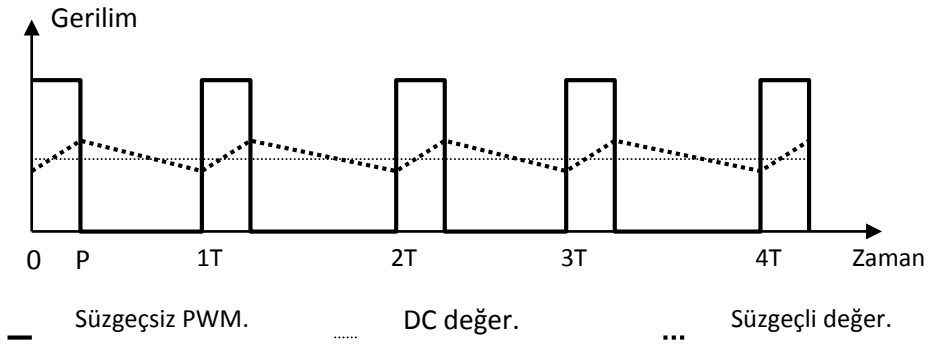
Şekil-12.1 DA motorun hız denetiminin PWM ile yapılması.



Şekil-12.3 Yüksek duty saykıllı PWM

Vuru Genişlik Modulasyonu

Kare dalga işareti düşük geçiren süzgeç devresinin girişine uyguladığımızda çıkışından kare dalganın rms değerinde bir analog işaret elde edilir. Bu noktadan yola çıkarak kare dalga işaretin duty saykılı değiştirilerek istenilen değerde analog işaret üretmek mümkündür.

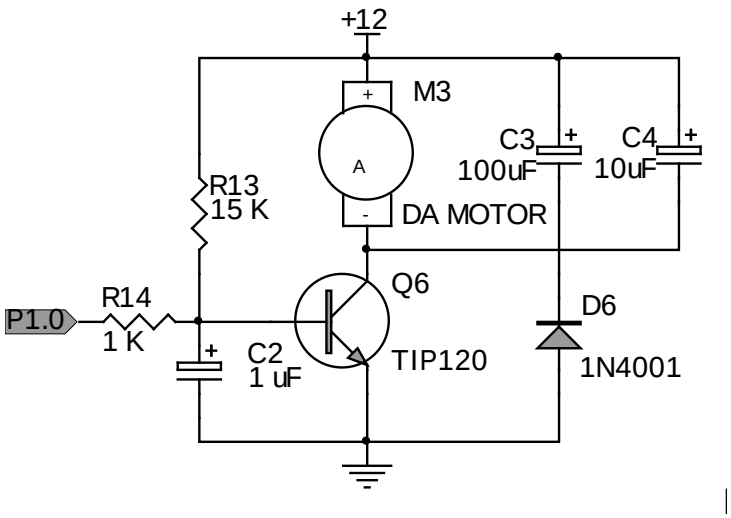


Şekil-12.2 Düşük duty saykıllı PWM

Duty saykılı değiştirilmiş kare dalga işarete vuru-genişlik modülasyonlu işaret adı verilir. Şekil-12.2'de düşük duty saykıllı, şekil-12.3'de yüksek duty saykıllı bir işaret ve etkin değerleri gösterilmiştir. PWM (puls-width modulation) anahtarlamalı güç kaynaklarında akım ve gerilimi ayarlamak için yaygın olarak kullanılır. Özellikle sadece bir g/ç hattı kullanarak bu işi yapması bu yöntemi en ucuz analog işaret üretme yöntemi yapmıştır. Şekil-12.2'deki mantık 1 seviyesindeki gerilim süzgeç devresinin depolayıcı elemanını çok az miktarda dolduracaktır ve sıfır seviyesi gerilimin süresi uzun olduğu için bu sürede dolan enerjide boşalacaktır. Şekil-12.3'da ise yüksek seviye gerilimin süresi uzun olduğu için süzgeç devresinin depolayıcı elemanı daha fazla dolacak ve sıfır seviye kısa süreli olduğu için boşalmayacaktır. Çıkıştan elde edilen gerilim yüksek olacaktır.

Pwm İle Motor Hız Denetimi

1. Şekil-12.4'daki devreyi kurun.



Şekil-10.4 PWM ile DA motorun hız denetim bağlantısı.

2. Aşağıdaki programı yazın.

```
pwm_motor:
        pwm             equ     p1.0
        duty_saykıl     equ     60h
        duty_saykılt    equ     61h
        motor_durum     equ     20h
org 0h
        Ajmp ana
org 000bh
        Ajmp zam0
org 100h
ana:
        clr p1.0
        clr motor_durum
        mov sp, #30h
        mov tmod, #02
        mov th0, duty_saykıl
        setb ea
        setb et0
        setb tr0
        mov p3, #0FFh
```

```
oku:
        mov a, p3
        mov duty_saykıl, a
        cpl a
        mov duty_saykılt,a
        sjmp oku
zam0:
        jb motor_durum, durdur
        setb pwm
        mov th0, duty_saykıl
        setb motor_durum
        reti
durdur:
        clr pwm
        mov th0, duty_saykılt
        clr motor_durum
        reti
```

3. Bu program p1.0'dan duty saykılı p3'e bağlanan anahtarlarla belirlenen PWM işaret üretir ve devreye bağlı sabit mıknatıs doğru akım motorunun hız denetimini yapar. Programın hatalarını simulatörde giderdikten sonra işlemciyi programlayıp çalıştırın.

4. Çalışma sırasında p3'e bağladığınız dip switch'leri kullanarak duty saykılı değiştirerek, motordaki hız değişimini gözlemleyin.

5. Eğer hız değişimini gözlemleyemediyseniz zamanlayıcı 0'ı kip 1'de çalıştırarak deneyin, p3 portundan okunan değeri duty_saykıl'ın yüksek değerli biti yapın. Düşük değerli baytı 00 yapın.

DENEY 5:
DAC BAĞLANTISI

Gerçek dünya ile iletişim kurmak için mikroişlemcinin analog işaret üretmesi veya analog işarete duyarlı hale gelmesi gerekir. Analog işaret üretmek için sayısal-analog çeviriciler kullanılırken analog işaretleri algılayabilmek için de analog-sayısal çeviriciler kullanılır. Kullanılan uygulamanın gerektirdiği hassaslığa bağlı olarak ADC veya DAC tümdevresi seçilerek kolaylıkla analog dünya ile iletişim kurulabilir.

SAYISAL-ANALOG ÇEVİRİCİLER

Sayısal-analog çeviriciler farklı hassasiyette tüm devre olarak üretilirler. Kullanım yerine göre bu tümdevreler arasından bir seçim yapılır. Bu deneyde genel kullanıma yönelik üretilen DAC0832 tümdevresi kullanılacaktır. 8 bit veri girişine sahip bu DAC'ın referans gerilimi ayarlanarak çıkış geriliminin en yüksek değeri belirlenir. çıkış geriliminin değeri şöyle hesaplanabilir;

$$V_{ÇIKIŞ} = \frac{S_G}{256} V_{REF}$$

S_G: Sayısal girişin onlu değeri.

V_{REF}:DAC referans girişine uygulanan gerilim.

Referans gerilimi arttırılarak çıkışın en yüksek değeri arttırılabilir. Fakat genellikle referans gerilimi 5 Volt seçilir DAC çıkışına bağlanan yükselteç yardımı ile çıkış gerilimi istenen değere yükseltilir. Kontrol hatları chip select (CS), internal latch enable (ILE) Xfer tranfer hattı sürekli izinli durumda kullanılabilir. Çevirme işlemi WR1 ve WR2 hatları ile tamamlanabilir. V_{REF} girişine bağlantı yapılmadığında iç referans gerilimi kullanılır, iç referans geriliminin değeri 5 volttur. DAC0832 tümdevresinin çıkışları diferansiyel akımdır, gerilime dönüştürmek için OP-AMP kullanılmalıdır. Şekil-13.1'de gösterilen bağlantıda LM324 kullanılarak çıkış akımı gerilime dönüştürülmüştür. Fakat çıkış negatif olduğu için ikinci bir OP-AMP kullanılarak çıkış terslenerek pozitif yapılmıştır. Sayısal girişe FFh yazıldığında analog çıkış birinci OP_AMP'ın çıkışında –5 Volt ikinci OP_AMP'ın çıkışında ise +5 Volt olacaktır. DAC'ın işleme başlatılması tablo-13.1'de gösterilmiştir.

İşlem	WR2	WR1
Etkin değil	1	1
Giriş tutucusuna	1	0
Çıkış tutucusuna	0	1

Tablo-13.1

Analog çıkışı elde edebilmek için tabloda verilen işlemle aşağıdaki sıra ile yapılmalıdır.

- Port 1'e sayısal girişi yaz.

- WR1 ve WR2 girişlerinin her ikisini YÜKSEK seviye yap.
- Sayısal girişteki veriyi giriş tutucusuna, WR1 DÜŞÜK seviye ve WR2 YÜKSEK seviye yaparak al.
- WR1 ve WR2 girişlerinin her ikisini YÜKSEK seviye yap.
- Çıkış tutucusuna aktarmak için, WR1 YÜKSEK seviye ve WR2 DÜŞÜK seviye yap.
- WR1 ve WR2 girişlerinin her ikisini YÜKSEK seviye yap.

Bu adımlar sıra ile yapılmalıdır.

İşlem Sırası

1. Şekil-11.1'deki devreyi kurun.

2. Aşağıdaki programı yazın.

```
;*******************************************************
;bu program DAC çıkışından testere dişi dalga üretir.
;*******************************************************
;port 1 sayısal çıkış, p3.7 WR2 ve P3.6 WR1
dacveri     equ   p1
WR1         equ   p3.6
WR2         equ   p3.7
Org 0h
        Ljmp dac
Org 100h
        Mov a, #0FFh
Testere:
        Dec a
        Mov dacveri, a
        Setb WR1
        Setb WR2
```

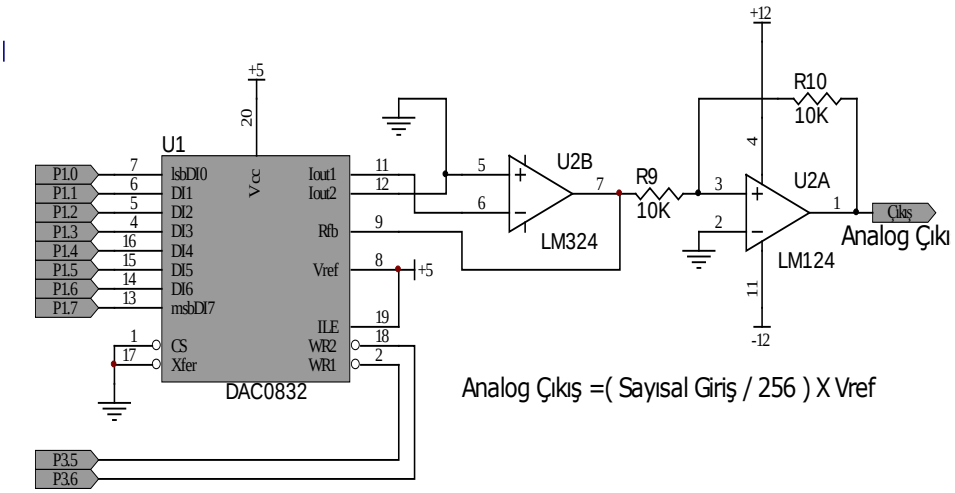
Clr WR1

Setb WR1

Clr WR2

Setb WR2

Sjmp testere



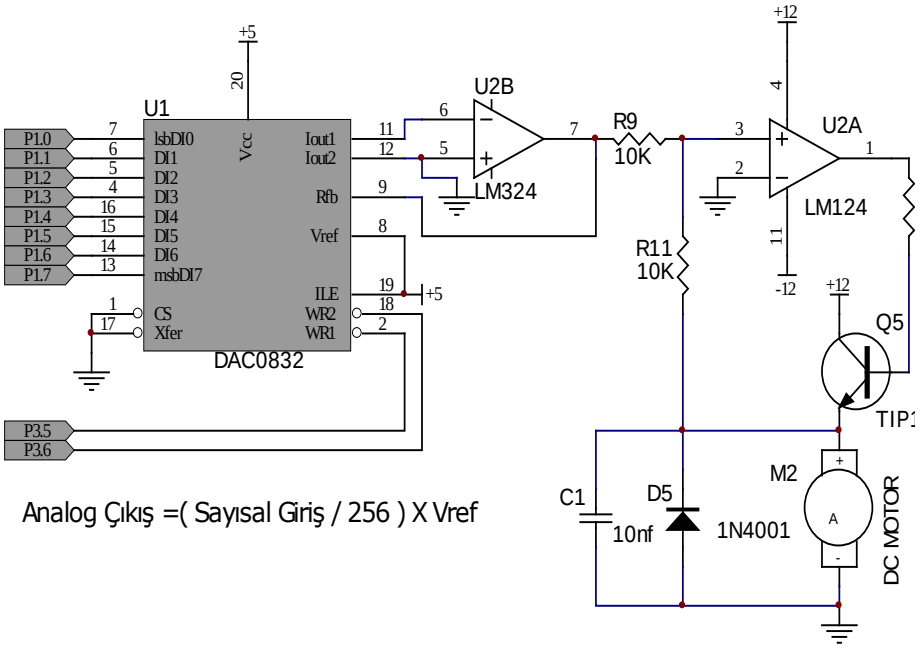
Şekil-13.1 DAC0832'nin 8051'e bağlanması.

3. Programı simulatörde çalıştırın varsa hatalarını düzeltin. İşlemciyi programlayıp devreye enerji verin $V_{çıkış}$ ucuna osilaskop bağlayıp testere dişi dalgayı gözlemleyin.

4. Şekil-13.2'deki devreyi kurun.

5. Motorun hızını P3.0, P3.1, P3.2, P3.3 girişlerine bağlanan anahtarlardan girilen değerlere göre ayarlayan programı yazın. 4 bit veri girişi olduğundan 16 kademeli bir hız denetini yapabilirsiniz. 0 bilgisi geldiğinde çıkışa 00h yazın motor çalışmasın. 1 bilgisi gelirse 1Fh yazın, 2 gelirse 2Fh, ... F gelirse FFh yazın. Böylece motorun hız denetimini 16 kademeli yapmış olursunuz.

6. Programı assembly edip simulatörde hatalarını düzelttikten sonra işlemciyi programlayıp devreye gerilim uygulayın. Anahtarların tüm olası durumlarına alarak motordaki hız değişimini gözleyin.



Analog Çıkış =(Sayısal Giriş / 256) X Vref

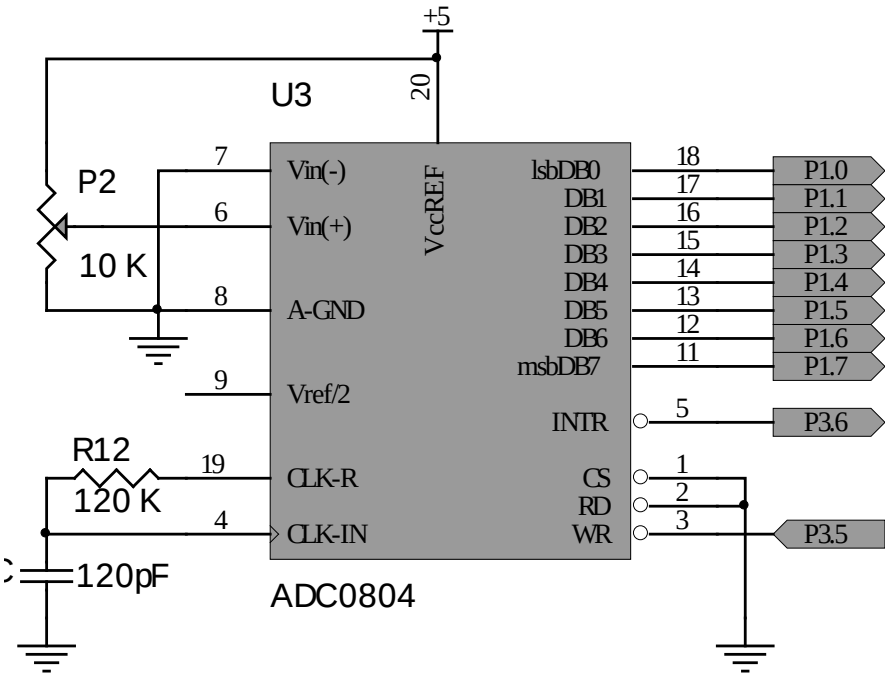
Şekil-13.2 DAC0832'yi kullanarak DA motorun hızının denetlenmesi.

DENEY 6:
ANALOG VERİ İŞLEME

Mikroişlemci dış dünyadaki analog değerleri okumak için analog-sayısal çeviricileri kullanır. Mikroişlemcinin kullanması gereken analog değişkenler ADC kullanılarak sayısal hale dönüştürülebilirler. Şekil-14.1'de en basit bir ADC'nin 8051'e bağlantısı gösterilmiştir. ADC0804 çok az ek devre elemanı gereksinim duyarak mikroişlemciye bağlanabilir. Şekilde analog değer bir trimpotun orta ucundan alınmıştır. Gerçek uygulamalarda genellikle sensör çıkışındaki küçük gerilim yükseltilerek analog girişe uygulanır.

ADC084'ün sayısal çıkışları 8051'in port 1 girişine bağlanmıştır. Veri hatlarının haricinde iki adet denetim hattı 8051'in port 3'ün 0 ve 1 numaralı hatlarına bağlanmıştır. WR hattı çevrimi başlatmak için 8051 tarafından kısa süreli DÜŞÜK seviye çekilir. Analog girişteki değerin karşılığı bulunur ve sayısal çıkışa yazıldıktan sonra INTR hattı ADC084 tarafından DÜŞÜK seviye çekilir. Doğru sayısal değer ancak INTR hattı 0 seviyesinde iken okunabilir.ADC0804 RC osilatörü tarafından üretilen saat işaretine göre çalışır. Analog girişe negatif veya pozitif gerilim uygulanabilir. Örnekte V(+) girişine 0-5 Volt arası bir gerilim uygulanacaktır.

ADC0804 gibi sayısal çıkışını paralel olarak veren ADC'ler olduğu gibi sayısal çıkışı seri olarak verebilen ve osilatör devreleri üzerinde olan ADC'lerde üretilmektedir. Kullanımı son yıllarda oldukça yaygınlaşmıştır. Çoğunlukla 10 kanaldan daha fazla girişleri ve 12 bitten fazla çıkışları vardır. Bu tür ADC'lerin çözünürlükleri yüksektir fakat erişim süreleri düşük olduğu için hızlı işlem gerçekleştirmek için kullanılamazlar. Bu ADC'leri denetleyen yazılımlar da paralel olanlara göre daha karmaşıktır. Bu eksilerine rağmen endüstriyel uygulamalarda tek bir tümdevre ile birden fazla analog değeri sayısala dönüştürmesinden ve daha az sayıda giriş/çıkış hattı kullanmasından dolayı yaygın bir şekilde kullanılmaktadır.



Şekil-14.1 DAC0832'yi kullanarak DA motorun hızının denetlenmesi.

İşlem Sırası

1. Şekil-14.1'teki devreyi kurun.

2. Okunan analog değeri port 2'ye bağlanan LCD göstergede veya 7-parçalı LED göstergede görüntüleyen programı yazın.

3. Programı assembly ederek simulatörde çalıştırın hataları varsa düzeltin. İşlemciyi programlayarak devreye gerilim uygulayın.

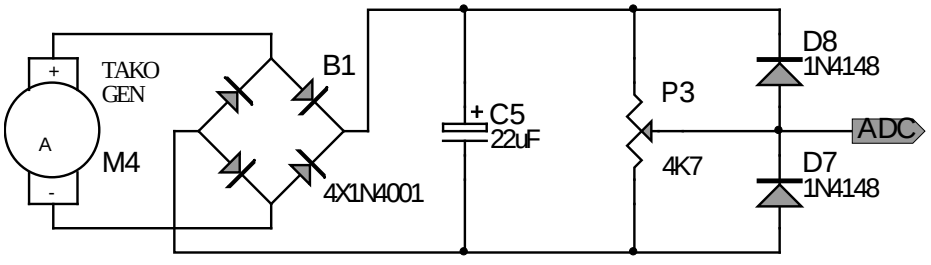
4. Trimpotu ayarlayarak göstergeyi gözleyin.

DENEY 7:

KAPALI ÇEVRİM MOTOR HIZ DENETİMİ

DA motorunun hızının uçlarına uygulanan akımla denetlenebildiğini daha önceki deneylerde öğrendiniz. Bu deneyde de aynı işlem yapılacaktır, fakat diğer deneylerde hız denetimi yapılırken yükün sabit olduğu varsayılmıştı. Bu deneyde ise değişken yük kullanılacaktır. Yükün artması motorun kaynaktan çektiği akımı arttırır, azalması ise akımı da azaltır. Açık çevrim denetleme kullandığınız daha önceki deneylerde motora uygulan akımı istediğimiz hıza göre arttırmış veya azaltmıştınız. Fakat motorun hızını ölçüp istediğimiz değere gelip gelmediğini denetlememiştiniz. Bu deneyde motorun hızını takometre veya enkoder kullanarak ölçeceksiniz, ölçtüğünüz değerle hedeflediğiniz değeri karşılaştırıp elde edilen farka göre motor akımını artırıp veya azaltarak hızın hedeflenen değerde kalmasını sağlayacaksınız.

Motorun hızını ölçmek için tako genaratörü kullanıldığında öncelikle bu üretecin çıkışının şekil-14.2'de gösterildiği gibi doğrultulup seviyesinin ADC' ye bağlanacak şekilde ayarlanmalıdır. Trimpot çıkışındaki gerilim 5 volttan küçük olmalıdır aksi halde diyot tarafından kırpılacaktır. Bu durumda motorun hızı yanlış ölçülmüş olacaktır. ADC' ye uygulanan analog işaret sayısala dönüştürülecektir.



Şekil-14.2 Tako genaratörden elde edilen işaretin düzenlenmesi.

Hız ölçümü için enkoder kullanıldığında elde edilen vurular seviye olarak düzenlendikten sonra zamanlayıcı veya kesme girişlerine uygulanır. Vurunun genişliği ölçülerek veya vuru adedi sayılarak hız belirlenir.Motorun hızını denetlemek için PWM veya DAC yöntemlerinden her ikisi de kullanılabilir.

İşlem Sırası

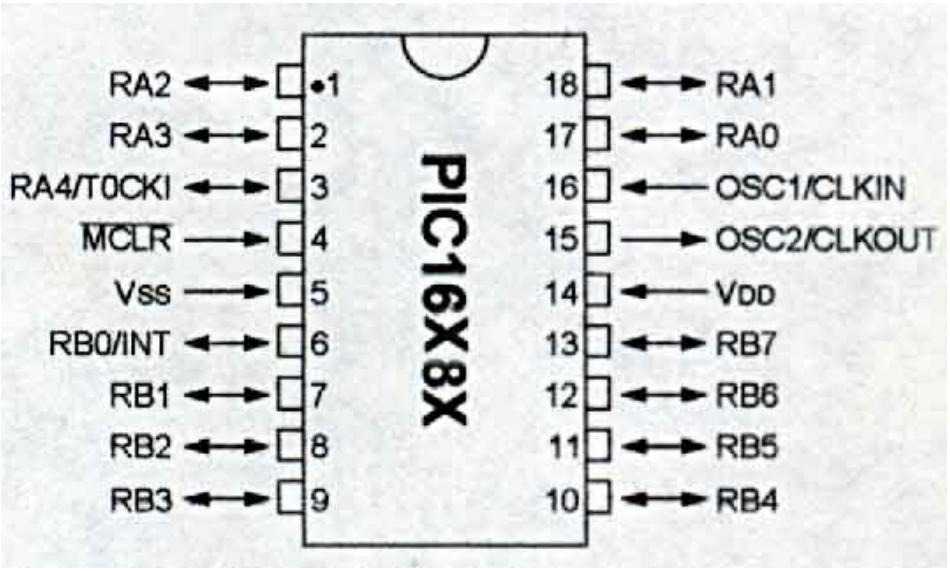
1. Şekil-12.2'deki devrenin çıkışını şekil-12.1'deki devrenin V_{in+} girişine bağlayın.

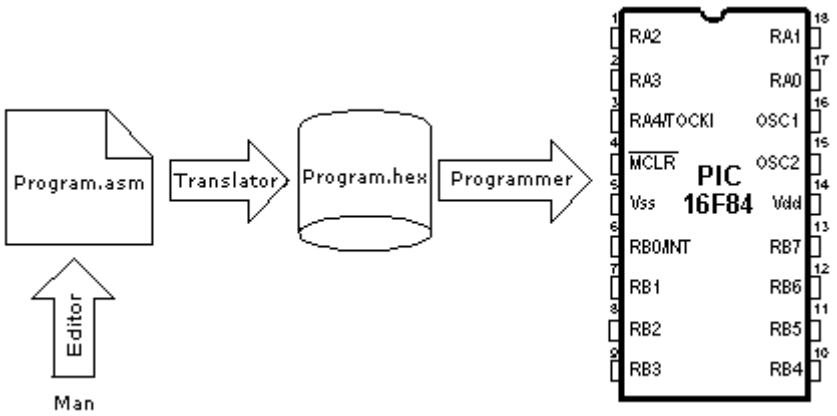
2. ADC'den 80h okunacak şekilde motorun hızını sabitleyen programı yazın. Programınıza ADC'nin çıkışını gösterecek şekilde gösterge programını da ekleyin.

3. Programı ve devreyi çalıştırın, motorun hızının sabitlenmesini bekleyin. Daha sonra motoru küçük miktarda yükleyerek hızı gözlemleyin.

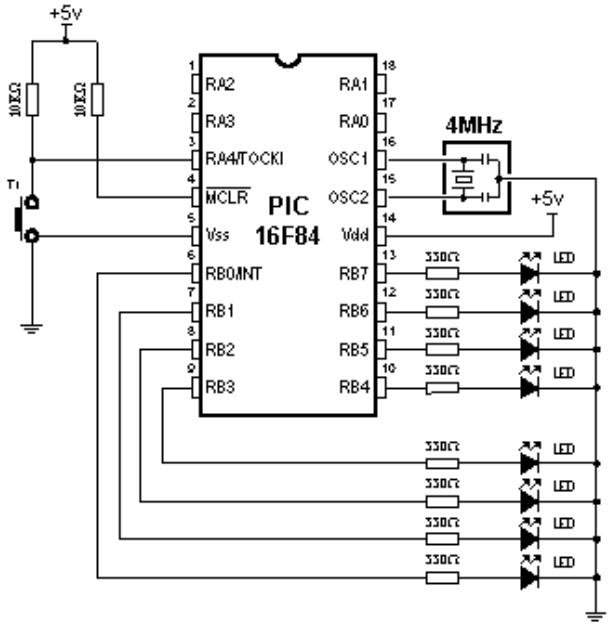
4. Hız değişene kadar yüklemeye devam edin. Hızın değiştiği nokta bu devrenin desteklediği aralıktır. Bundan daha fazla yükleme devre tarafından desteklenmeyecektir

DENEY 8:
PIC UGULAMALARI

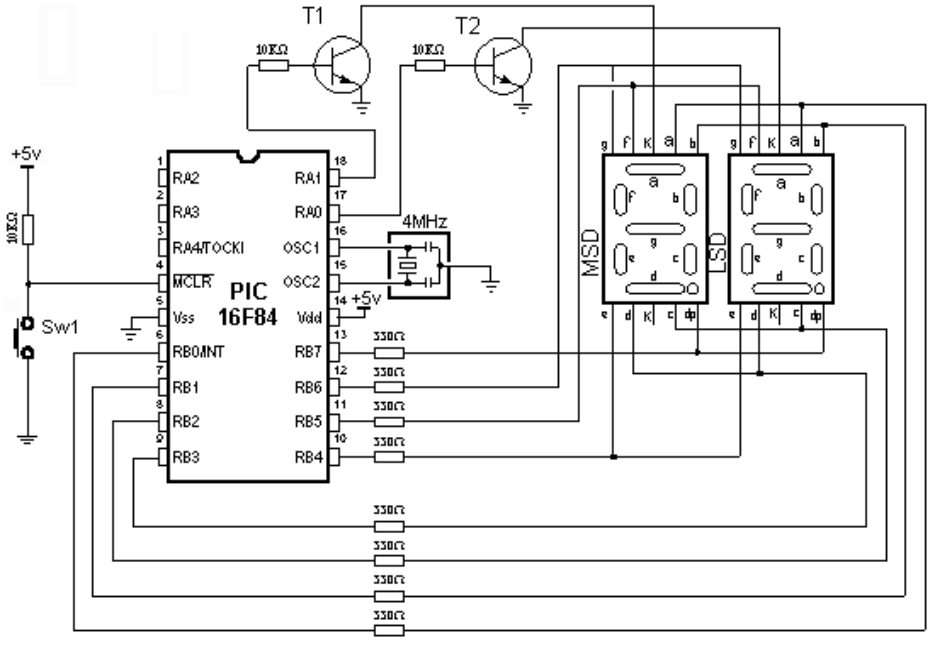




PIC16F84'e program yazmanın aşamaları.



PIC16F84'e LED bağlantısı



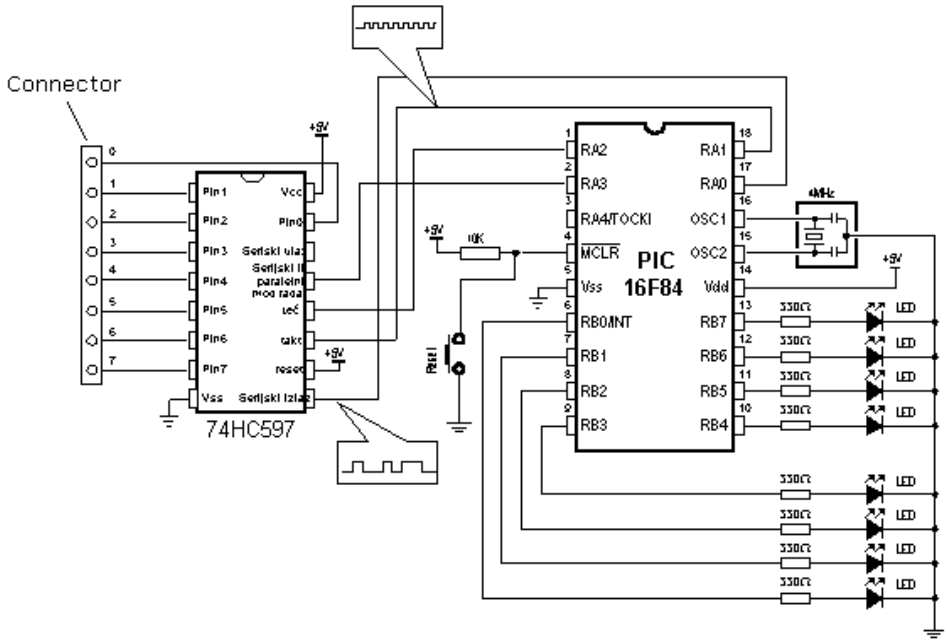
İki adet 7 elemanlı göstergenin PIC16F84'e bağlanması.

TEST.asm

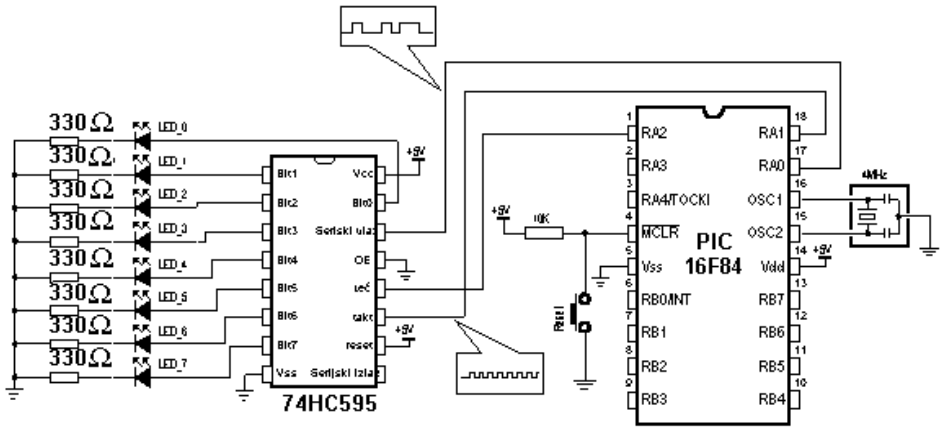
```
;***** Declaring and configuring a microcontroller *****

      PROCESSOR 16f84
      #include "p16f84.inc"

        __CONFIG   _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declaring variables *****

        Cblock 0x0C             ; Beginning of RAM
        WCYCLE                  ; Belongs to 'WAITX' macro
        PRESCwait
        endc

;***** Structure of program memory *****

        ORG    0x00             ; Reset vector
        goto    Main

        ORG    0x04             ; Interrupt vector
        goto    Main            ; No interrupt routine

        #include "bank.inc"     ; Assistant files


Main                            ; Beginning of the program

        BANK1
        movlw   0xff            ;Port A initialization
        mowwf   TRISA           ; TRISA <- 0xff  all input
        movlw   0x00            ; PORTB initialization
        mowwf   TRISB           ; TRISB <- 0xff
        movlw   0x00            ; PORTB initialization
        BANK0

        movlw   0xff
        mowwf   PORTB           ;Turn on all leds

Loop
        goto    Loop            ; Repeat loop

   End                          ; End of program
```

LED.asm

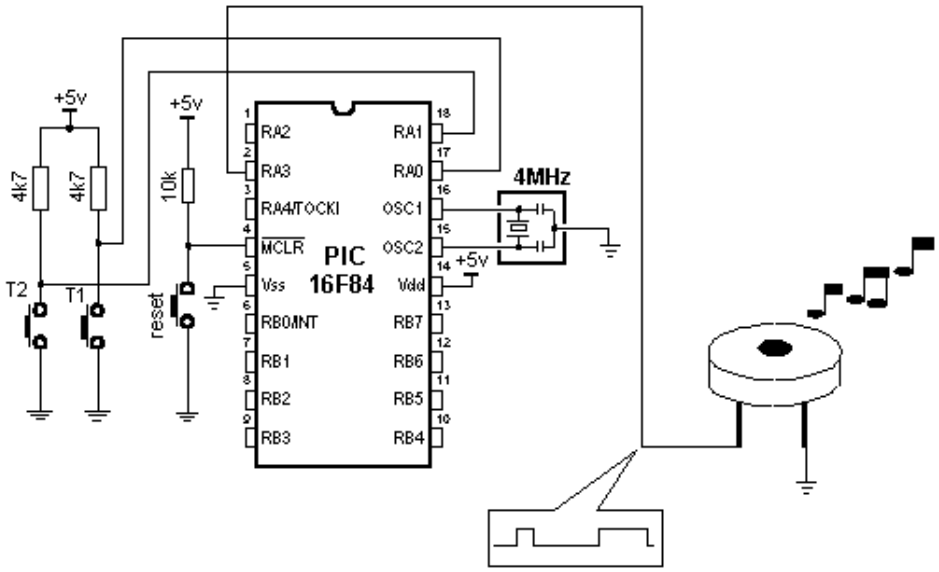
```
;***** Declaring and configuring a microcontroller *****

      PROCESSOR 16f84
      #include "p16f84.inc"

        __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC


;***** Declaring variables *****

        Cblock  0x0C            ; Beginning of RAM
        TempC                   ; Belongs to "LED_Disp2" macro
        LO
        endc

;***** Declaring the hardware *****

        LEDtrisA equ    TRISA
        LEDportA equ    PORTA

        LEDtrisB equ    TRISB
        LEDportB equ    PORTB

;***** Structure of program memory *****

        ORG     0x00            ; Reset vector
        goto    Main

        ORG     0x04            ; Interrupt vector
        goto    ISR             ; Interapt rutina is found
                                ; in 7-seg.inc file

        #include "bank.inc"     ; Assistant files
        #include "7-seg.inc"

Main                            ; Beginning of the program
        LED_Init

        LED_Disp2 0x21          ; Display on two 7-seg. displays
                                ; broj "21"
loop    goto    loop            ; Stay here

      End                       ; End of program
```



PIC16F84'ün giriş adedinin paralel girişli seri çıklı kayar yazaçla arttırılması



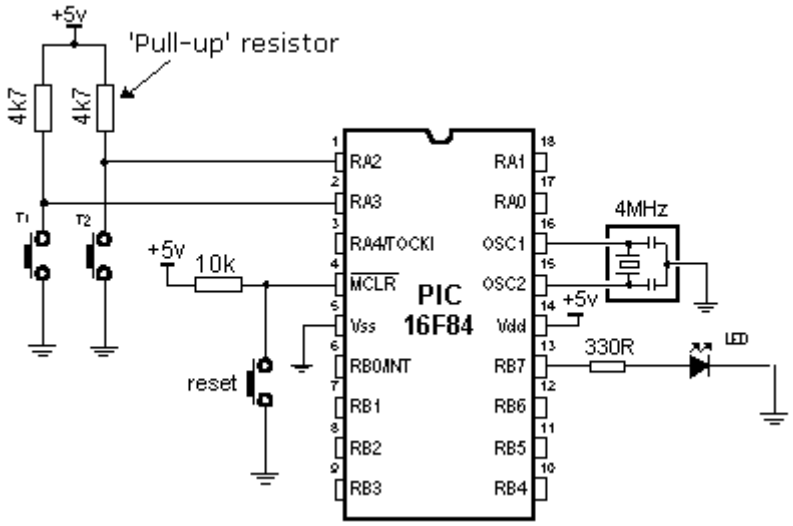
PIC16F84'ün çıkış hattı sayısının seri girişli paralel çıkışlı kayar yazaçla artırılması.



PIC16F84'ün çıkışına buzzer bağlama.

BEEP.asm

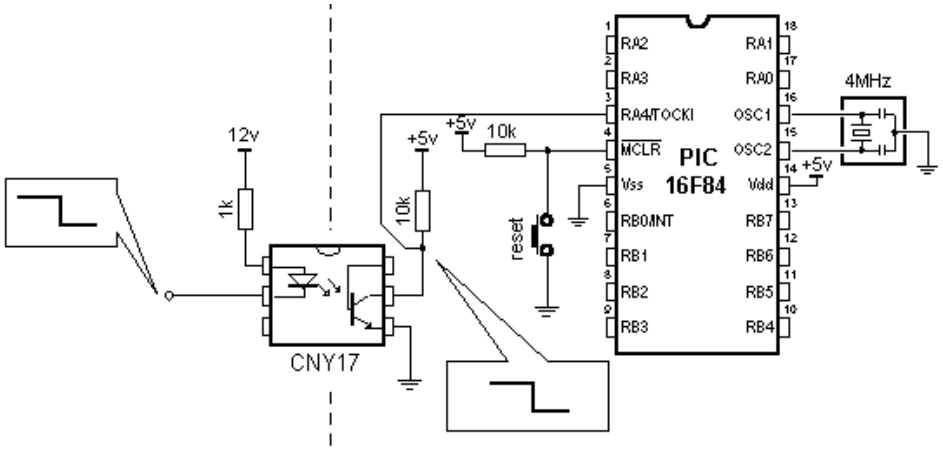
```
;***** Declaring and configuring a microcontroller *****

      PROCESSOR 16f84
      #include "p16f84.inc"

        __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declaring variables *****

        Cblock  0x0C            ; Beginning of RAM-a
        WCYCLE                  ; Belongs to 'WAITX' macro
        PRESCwait
        Beep_TEMP1              ; Belongs to 'BEEP' macro
        Beep_TEMP2
        Beep_TEMP3
        endc

;***** Declaring the hardware *****

        #define  BEEPport PORTA,3 ; Port and pin for mini speaker
        #define  BEEPtris TRISA,3

;***** Structure of program memory *****

        ORG     0x00            ; Reset vector
        goto    Main

        ORG     0x04            ; Interrupt vector
        goto    Main            ; No interrupt routine

       #include "bank.inc"      ; Assistant files
       #include "button.inc"
       #include "wait.inc"
       #include "beep.inc"

Main                            ; Beginning of the program
        BANK1
        movlw   0x17            ; Port A initialization
        mowwf   TRISA           ; TRISA <- 0x00
        movlw   0x00
        mowwf   TRISB
        BANK0

        BEEPinit                ; Mini speaker initialization

        clrf    PORTB

Loop
        Button 0, PORTA, 0, .100, Play1         ; Button 1
        Button 0, PORTA, 1, .100, Play2         ; Button 2
        goto    Loop

Play1
        BEEP 0xFF, 0x02
        BEEP 0x90, 0x05
        BEEP 0xC0, 0x03
        BEEP 0xFF, 0x03                         ; First melody
        return

Play2
        BEEP 0xbb, 0x02
        BEEP 0x87, 0x05
        BEEP 0xa2, 0x03
        BEEP 0x98, 0x03                         ; Second melody
        return

      End               ; End of program
```



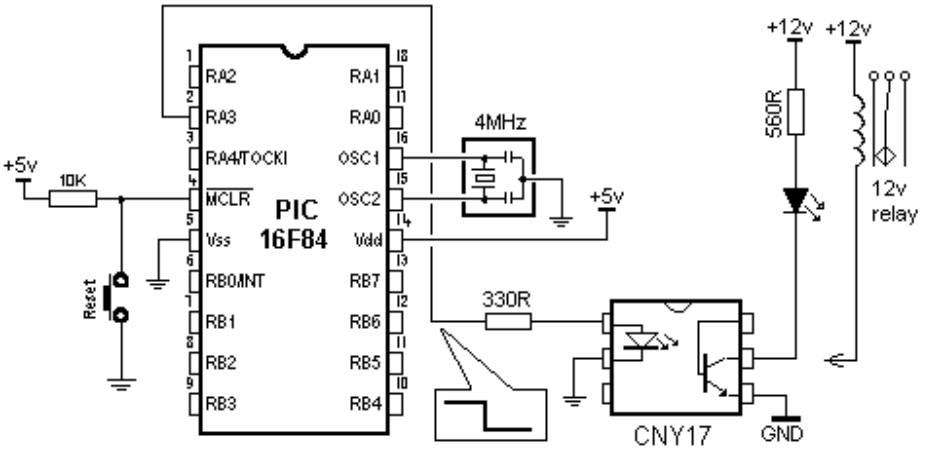
PIC16F84'e anahtar bağlama.

BUTTON.asm

```
;***** Declaring and configuring a microcontroller *****

      PROCESSOR 16f84
      #include "p16f84.inc"

        __CONFIG   _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declaring variables *****

        Cblock 0x0C             ; Beginning of RAM
        WCYCLE                  ; Belongs to 'WAITX' macro
        PRESCwait
        endc

;***** Structure of program memory *****

        ORG     0x00            ; Reset vector
        goto    Main

        ORG     0x04            ; Interrupt vector
        goto    Main            ; No interrupt routine

        #include "bank.inc"     ; Assistant files
        #include "button.inc"
        #include "wait.inc"

Main                            ; Beginning of the program
        BANK1
        movlw   0xff            ; PORTA initialization
        mowwf   TRISA           ; TRISA <- 0xff
        movlw   0x00            ; PORTB initialization
        mowwf   TRISB           ; TRISB <- 0x00
        BANK0

        clrf    PORTB           ; PORTB <- 0x00

Loop
        Button 0, PORTA, 2, .100, On    ; Button 1
        Button 0, PORTA, 3, .100, Off   ; Button 2
        goto    Loop

On
        bsf     PORTB,7         ; Turn on LED
        return

Off
        bcf     PORTB,7         ; Turn off LED
        return

   End          ; End of program
```



PIC16F84'ün girişine optocoupler bağlama.

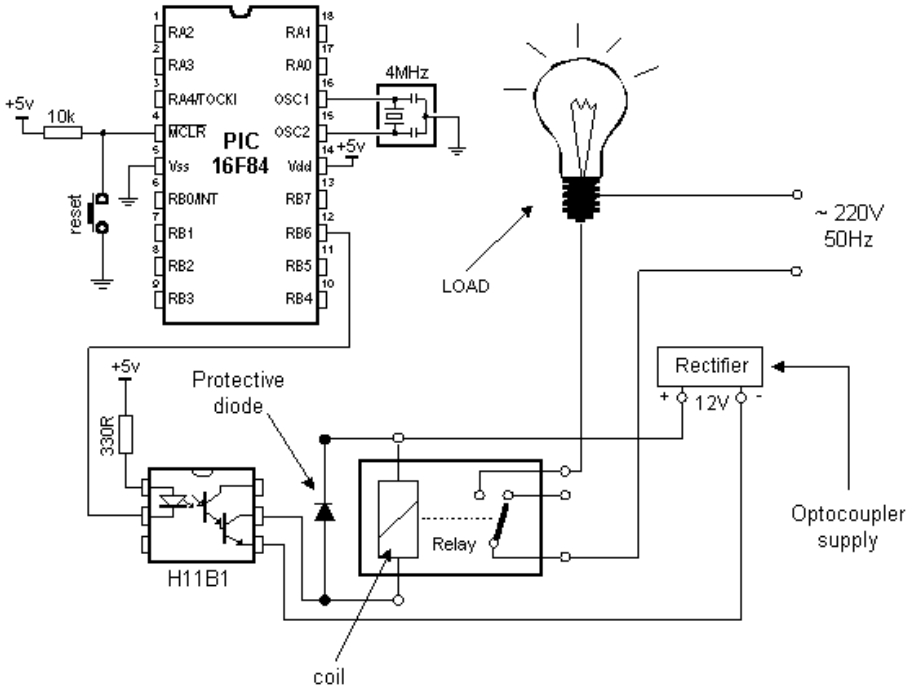


PIC16F84'ün çıkışına optocoupler bağlama.

Optik İzalosyyonlu Port Bağlantısı

OPTO_IN.asm

```
;***** Declaring and configuring a microcontroller *****

      PROCESSOR 16f84
      #include "p16f84.inc"

        __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Structure of program memory *****

        ORG    0x00            ; Reset vector
        goto   Main

        ORG    0x04            ; Interrupt vector
        goto   Main            ; No interrupt routine

         #include "bank.inc"   ; Assistant files

Main                           ; Beginning of the program
        BANK1
        movlw  0xff            ; PORTA initialization
        mowwf  TRISA           ; TRISA <- 0xff (input)
        movlw  0x00            ; PORTB initialization
        mowwf  TRISB           ; TRISB <- 0x00 (output)
        movlw  b'00110000'     ; RA4 -> TMR0,
        mowwf  OPTION_REG      ; Increment TMR0 on failing edge
        BANK0

        clrf   PORTB ; PORTB <- 0
        clrf   TMR0  ; TMR0 <- 0

Loop

        movf   TMR0,w          ; Copy TMR0 in W reg.
        mowwf  PORTB           ; send value of W reg. on PORTB
        goto   Loop            ; Repeat the loop

   End          ; End of program
```



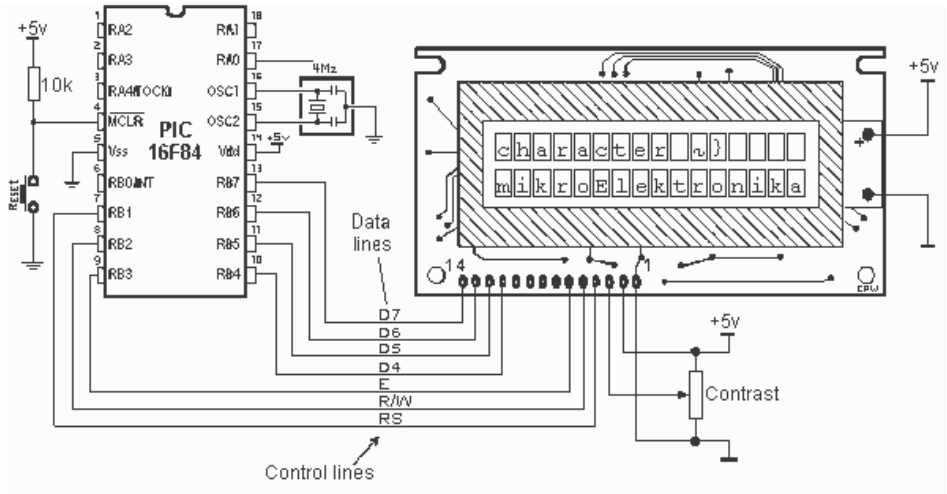
PIC16F84'ün çıkışına röle bağlama.

Port Hattı ile Röle Denetimi

RELAY.asm

```
;***** Declaring and configuring a microcontroller *****

        PROCESSOR 16f84
        #include "p16f84.inc"

          __CONFIG   _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declaring variables *****

          Cblock 0x0C               ; Beginning of RAM
          WCYCLE                    ; Belongs to 'WAITX' macro
          PRESCwait
          endc

;***** Declaring the hardware *****

          #define RELAY PORTA,3

;***** Structure of program memory *****

          ORG     0x00              ; Reset vector
          goto    Main

          ORG     0x04              ; Interrupt vector
          goto    Main              ; No interrupt routine

          #include "bank.inc"       ; Assistant files
          #include "button.inc"
          #include "wait.inc"

Main                                ; Beginning of the program
          BANK1
          movlw   0x17              ; PORTA initialization
          movwf   TRISA             ; TRISA <- 0x17
          movlw   0x00              ; PORTB initialization
          movwf   TRISB             ; TRISB <- 0x00
          BANK0

          clrf    PORTB             ; PORTB <- 0x00

Loop
          Button 0, PORTA, 0, .100, On    ; Button 0
          Button 0, PORTA, 1, .100, Off   ; Button 1

          goto    Loop              ; Repeat the loop

On
          bsf RELAY                 ; Turn on relay
          return

Off
          bcf RELAY                 ; Turn off relay
          return

    End             ; End of program
```

PIC ile Karakter LCD Denetimi

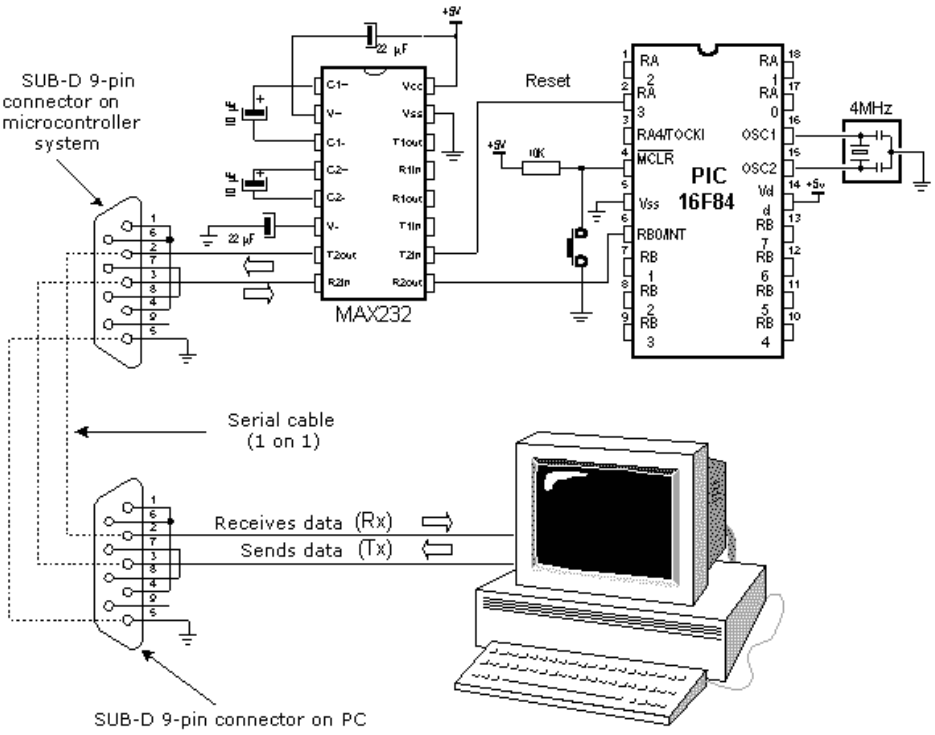


PIC16F84'e LCD gösterge bağlama.

LCD.asm

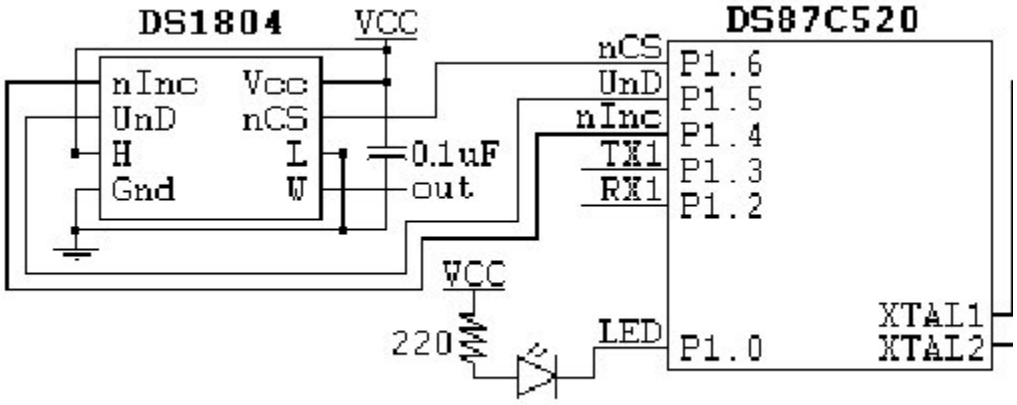
```
;***** Declaring and configuring a microcontroller *****

      PROCESSOR 16f84
      #include "p16f84.inc"

        __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declaring variables *****

        Cblock 0x0C             ; Beginning of RAM
        LCDbuf                  ; Belongs to 'LCDxxx' macros
        LCDtemp
        WCYCLE                  ; Belongs to 'WAITX' macro
        PRESCwait
        Pointer                 ; Pointer on characters in message
        endc

;***** Declaring the hardware *****

        LCDtris equ     TRISB
        LCDport equ     PORTB

;***** Structure of program memory *****

        ORG     0x00    ; Reset vector
        goto    Main

        ORG     0x04    ; Interrupt vector
        goto    Main    ; No interrupt routine

Messages                ; Beginning of the messages

      movwf   PCL

                        ; Display messages

Message1 dt "mikRoEleKtrOnlkA"
Message2 dt "bla, bla"
Message3 dt "example"

END_messages                            ; End of messages

        #include "bank.inc"      ; Assistant files
        #include "wait.inc"
        #include "lcd.inc"
        #include "print.inc"

Main                    ; Beginning of the program

        bcf PORTB,2

        LCDinit         ; LCD initialization

        LCDchar 'C'     ; Display character on LCD
        LCDchar 'a'
        LCDchar 'r'
        LCDchar 'a'
        LCDchar 'c'
        LCDchar 't'
        LCDchar 'e'
        LCDchar 'r'
        LCDchar 's'
        LCDchar ':'
        LCDchar ' '

        LCDchar         0x00    ; Display special characters
        LCDchar         0x01

        LCDline 2       ; Second line

; Display message1 on LCD

        PRINT   Messages, Message1, Message2, Pointer, LCDw

Loop    goto    Loop    ; Infinite loop

        End             ; End of program
```

PIC ile Seri Veri İletimi



PIC16F84'ün COM portuna bağlanması.

DENEY 9:
DS1804'ÜN 8051 İLE KULLANILMASI




```
;************************************************************
;* hardware description *
;* p1.0 - led p0.0 - sn74f373n *
;* p1.1 - p0.1 - " *
;* p1.2 - rxd1 – not used p0.2 - " *
;* p1.3 - txd1 – not used p0.3 - " *
;* p1.4 - nınc p0.4 - " *
;* p1.5 - und p0.5 - " *
;* p1.6 - ncs p0.6 - " *
;* p1.7 - p0.7 - " *
;* *
;* p3.0 - rxd0 - not used p2.0 - upper *
;* p3.1 - txd0 - not used p2.1 - address *
;* p3.2 - p2.2 - byte *
;* p3.3 - p2.3 - " *
;* p3.4 - p2.4 - " *
;* p3.5 - p2.5 - " *
;* p3.6 - wr\ p2.6 - " *
;* p3.7 - rd\ p2.7 - " *
;************************************************************
stack equ 02fh ; bottom of stack
; yığın 30h başlıyor
;** general sfr names **
smod_1 equ 0dfh ; buad rate doubler bit declared
;** port 1 signal names **
led equ 90h ; p1.0 is led
; p1.1 is not used
```

```
rx1 equ 92h ; p1.2 is serial port 1 rx
tx1 equ 93h ; p1.3 is serial port 1 tx
nınc equ 94h ; p1.4 is not ınc, - 1804
und equ 95h ; p1.5 is up not down - 1804
ncs equ 96h ; p1.6 is not chipselect - 1804
; p1.7 is not used
;* no ınterrupts are enabled in this code. ıf interrupts are to be *
;* enabled they need to have the label initialized here. *
;*********************************************************************
org 0000h     ; power up and reset
        ljmp start
org 0003h     ; external ınterrupt 0
        ljmp start
org 000bh     ; timer 0 ınterrupt
        ljmp start
org 0013h     ; external ınterrupt 1
        ljmp start
org 001bh     ; timer 1 ınterrupt
        ljmp start
org 0023h     ; serial port 0 ınterrupt
        ljmp start
org 002bh     ; timer 2 ınterrupt
        ljmp start
        ljmp start
;************************************************************
;* the main program demonstrates using a 8051 to ommunicate *
;* with a ds1804. the program increments and decrements the *
;* potentiometer without writing the nv register, and it  *
;*demonstrates writing to the nv register as well.                *
;************************************************************
org 0080h             ; location that hardware begins execution
start:
        clr ea                          ; disable ınterrupts
        cpl led                         ; complement led - ıdentifies that the
; program has started execution
        lcall init1804 ; initializes ds1804, must be done in first
                        ; 50ms to avoid the three inputs becoming
                        ; active without them being in a known
                        ; state.
        lcall wait3sec          ; give time to read the last nvreg value
                        ; on a multimeter
        cpl led                 ;toggle led
        mov r0,#5 ;increment the 1804 by 5 intervals, 5 times,
loop10:                 ; with 3 seconds between increments
```

```
        lcall ıncrement1804x5     ; increment the ds1804
        lcall wait3sec            ; waits 3 seconds
        cpl led                   ; toggle led
        djnz r0,loop10
        lcall decrement1804x5     ; decrement the 1804 5 intervals
        cpl led                   ; toggle led
        lcall writenvreg          ; write current value to nvreg. now 20 steps
                                  ; greater that the start of execution.
        lcall wait3sec            ; wait 3 seconds for multimeter use
        cpl led                   ; complement led
        lcall ıncrement1804       ; increment 1804 once, don't write nvreg
        lcall wait3sec            ; wait 3 seconds for multimeter use
        cpl led                   ; complement led
        lcall decrement1804       ; decrement 1804 once, don't write nvreg
        lcall wait3sec            ; wait 3 seconds for multimeter use
        cpl led         ; complement led
endofmain: ; wait forever
        sjmp endofmain
;************************************************************
;**** ıncrement ds1804x5 routine ****
;**** ıncrements the chip 5 times without de-selecting the ****
;**** the chip between increments. this does not write to the ****
;**** nonvolatile register. ****
;************************************************************************
;* uses no other routines, and destroys no registers *
;************************************************************************
ıncrement1804x5:
        setb und      ; select increment
        clr ncs; select chip
        clr nınc      ; clear inc - increase to next position
        setb nınc     ; set nınc back to inactive state.
        clr nınc      ; clear inc - increase to next position
        setb nınc     ; set nınc back to inactive state.
        clr nınc      ; clear inc - increase to next position
        setb nınc     ; set nınc back to inactive state.
        clr nınc      ; clear inc - increase to next position
        setb nınc     ; set nınc back to inactive state.
        clr nınc      ; clear inc - increase to next position
        setb ncs      ; set ncs before nınc to avoid writing to
; the nv eeprom register in the ds1804
        setb nınc     ; set nınc back to inactive state.
        ret
;*********************************************************
;**** ıncrement ds1804 routine                                    ****
```

```
;**** ınc rements the chip 1 time, and de-selects the chip      ****
;**** without writing to the nonvolatile register.              ****
;*********************************************************
;* uses no other routines, and destroys no registers *
;*********************************************************
increment1804:
        setb und        ; select increment
        clr ncs         ; select chip
        clr nınc        ; clear inc - increase to next position
        setb ncs        ; set ncs before nınc to avoid writing to
; the nv eeprom register in the ds1804
        setb nınc       ; set nınc back to inactive state.
        ret
;*********************************************************
;**** decrement ds1804x5 routine ****
;**** decrements the chip 5 times without de-selecting the ****
;**** the chip between decrements. this does not write to the ****
;**** nonvolatile register. ****
;*********************************************************
;* uses no other routines, and destroys no registers *
;*********************************************************
decrement1804x5:
        clr und         ; select increment
        clr ncs; select chip
        clr nınc        ; clear inc - increase to next position
        setb nınc       ; set nınc back to inactive state.
        clr nınc        ; clear inc - increase to next position
        setb nınc       ; set nınc back to inactive state.
        clr nınc        ; clear inc - increase to next position
        setb nınc       ; set nınc back to inactive state.
        clr nınc        ; clear inc - increase to next position
        setb nınc       ; set nınc back to inactive state.
        clr nınc        ; clear inc - increase to next position
        setb ncs        ; set ncs before nınc to avoid writing to
; the nv eeprom register in the ds1804
        setb nınc       ; set nınc back to inactive state.
        ret
;*********************************************************
;**** decrement ds1804 routine ****
;**** decrements the chip 1 time, and de-selects the chip ****
;**** without writing to the nonvolatile register. ****
;*********************************************************
;* uses no other routines, and destroys no registers *
;*********************************************************
```

```
decrement1804:
        clr und         ; select decrement
        clr ncs ; select chip
        clr nınc        ; clear inc - decrease to next position
        setb ncs        ; set ncs before nınc to avoid writing to
; the nv eeprom register in the ds1804
        setb nınc       ; set nınc back to inactive state.
        ret
;*********************************************************
;**** ds1804 write nonvolatile register routine ****
;**** writes the nonvolatile register to the current wiper ****
;**** value, and then waits 10ms for the write to occur. ****
;*********************************************************
;* uses wt routine, and destroys registers r5, r6, and r7 *
;*********************************************************
writenvreg:
        clr ncs         ; select chip
        setb ncs        ; deselect chip, nınc already high so the
; ds1804 will store to the nv register
        mov r5, #255 ; wait 10ms before continuing, the nv reg.
        mov r6, #3    ; storage time
        mov r7, #1    ;
        lcall wt        ;
ret
;*********************************************************
; ınitialize ds1804 routine  sets all three control signals to their inactive state,
; and waits 50ms for the inputs to become active before returning to the main program.
uses wt routine, and destroys registers r5, r6, and r7
;*********************************************************
init1804:
        setb ncs                ; de-select 1804
        setb nınc               ; de-activate ıncrement signal
        setb und                ; select increment
        mov r5, #255            ; wait 50ms, this is done so communication
        mov r6, #15  ; will not begin before the ds1804 is ready
        mov r7, #1               ; to accept input.
        lcall wt;
        ret
;*********************************************************
; 3 saniye gecikme alt programı
; r5, r6 ve r7'nin içeriklerini değiştirir.
;*********************************************************
wait3sec:
        mov r5, #255            ; wait 3 sec., this is done so advances can
```

```
        mov r6, #147        ; be watched on a multi-meter
        mov r7, #6
        lcall wt
        ret
;*************************************************************
; 14.42us ile 221 saniye arası istenilen gecikmeyi yaratır.
; waits r7 * 867.6 ms if r5 = r6 = 255 ****
; waits r6 * 3.4 ms if r5 = 255 and r7 = 1
; waits r5 * 13.34us if r6 = r7 = 1
;*************************************************************
; r5, r6, ve r7'nin içeriklerini değiştirir.
;*************************************************************
wt:
        lcall wait16us      ; 12.8us of waits
        lcall wait16us
        lcall wait16us
        lcall wait16us
        lcall wait16us
        lcall wait16us
        lcall wait16us
        lcall wait16us
        djnz r5, wt         ;wait = r5 * 13.34us + 1.1us if r6 = r7 = 1
        djnz r6, wt         ;wait = r6 * 3.4ms if r5 = 255, r7 = 1
        djnz r7, wt         ;wait = r7 * 867.6ms if r5 = r6 = 255
        ret
;*************************************************************
;1.6 usn gecikme yaratır. (22.22mhz clock)
;*************************************************************
wait16us:
        nop
        ret
end
```

DENEY 10:

SERİ GİRİŞ/ŞIKIŞLI EEPROM BELLEĞİN 8051 İLE KULLANILMASI

açıklama
bu progam 8051 ile xicor firması tarafından üretilen seri giriş çıkışlı eeprom belleğin haberleşmesini sağlamak için adı geçen firma trafından yazılmıştır. x24xx seisi eeprom'lar 2-wire adı verilen iki hatlı seri haberleşme protokolünü kullanır. bu hatlar saat işaretini taşıyan scl ve iki yönlü veri taşıyan sda'dır. örnek programda port 1 scl ve p11 sda hattı olarak görevlendirilmiştir.

```
;** tanımlamalar:
;** start: başlangıç ayarları
;** stop: sonlandırma ayarları
;** reset: elemanı resetlemek için gerekli işlemler.
;** progpage: ram tamponda bulunan veri dizisini seri belleğe taşır.
;** progbyte: ram'de yer alan baytı seri belleğe yazar.
;** seqread: seri bellekten adres göstericini gösterdiği adresten başlayarak veri
dizisini okur.
;** randomread: adresi belirtilen seri bellekten bir bayt veri okur.
;** ackpoll: yazma işleminin bitişini test eder.
;** outack: alındı işaretinin gedişini test eder.
;** getack: alındı işaretinin gelişini test eder.

;*********************************************************
;* ınternal ram
;*********************************************************
r00 equ 00h
r01 equ 01h
rambuff equ 40h ; ram buffer address
stack equ 60h
;*********************************************************
;* program constants
;*********************************************************
```

```
sdabit equ 01h              ; sda hattı olarak p1.0 görevlendirildi.
sclbit equ 00h              ; scl hattı olarak p1.1 görevlendirildi.
pageno equ 00h     ; seri belleğin sayfa numarası
bpx_0 equ 03h               ; bpx bıt 0 posıtıon ın wpr
bpx_1 equ 04h               ; bpx bıt 1 posıtıon ın wpr
wel equ 01h                 ; wel bıt posıtıon ın wpr
rwel equ 02h                      ; rwel bıt posıtıon ın wpr
wpen equ 07h                      ; wpen bıt posıtıon ın wpr
welon equ 00000010b       ; wel denetim baytı
rwelon equ 00000110b      ; rwel denetim baytı
maxdelay equ 0ffh                 ;alındı işaretinin beklenme süresi
bytedata equ 058h           ; changes the x to an x ın the test program
;* program constants for x24645
seqreadsize equ 16                ; byte counts to shıft out usıng seq. read
deviceıd equ 080h           ; devıce select
hiaddrmask equ 3fh                ; mask for upper address byte
wpr_addr equ 1fffh  ; wpr physıcal address locatıon (byte access)
pagesize equ 32             ; bytes per page
;*************************************************************
;* start of user code *
;*************************************************************
org 0000h
        ljmp maın                 ; reset vector
org 0200h
maın:
        mov sp,#stack             ; load stack poınter
                                  ; ınıtıalıze the buffer before
                                  ; programmıng the content to a  page
        mov r0,#rambuff     ; r0 = ram buffer address
        mov dptr,#teststring ; dptr = test strıng address
ınitram:
        clr a
        movc a,@a+dptr      ; copy the test strıng to
        mov @r0,a           ; ram buffer
        inc dptr
        inc r0
        jnz ınitram
        acall reset1        ; reset the ınterface state machıne
        mov dptr,#wpr_addr ; read the wpr content and fınd the
        acall randomread    ; blocks that are locked. ıf both
                                  ; wpen bıt and wp pın are hıgh then
        jnb acc.wpen,wpen_off     ; bpx bıts are protected (wrıtes ;are  ... warnıng ...
                                  permıtted when ;wp ıs brought low).  make sure that
                                  ;wp pın ıs low before attemptıng to ;wrıte new value
```

```
                                to the wpr when ;wpen bıt ıs set.
wpen_off:
        jb acc.bpx_0,clr_bpx
        jnb acc.bpx_1,no_bpx    ; skıp ıf bpx bıts are clear
clr_bpx:
        clr a                   ; clear the block lock bıts (unprotect
        acall progbp            ; the entıre devıce)
        acall ackpoll           ; waıt tıll the operatıon ıs completed
no_bpx:
        acall setwel            ; set the wrıte enable bıt
        mov dptr,#wpr_addr      ; read the wpr content and
        acall randomread        ; check that wel bıt
        jb acc.wel,wrıtes_en    ; ıs set hıgh, else ıts a faılure
        ajmp $                  ; check the devıce/connectıons*stop*
wrıtes_en:
        mov dptr,#pageno        ; dptr = page number
        mov r0,#rambuff         ; r0 = ram buffer address
        acall progpage          ; transfer buffer content to the page
                                ; ın serial memory ındıcated by dptr
        acall ackpoll           ; waıt tıll completıon of page prog.
        mov dptr,#00h           ; address 0 ıs the locatıon for the byte ;to be wrıtten
                                thıs wıll wrıte over the ;fırst byte ın ;the pageprog
        acall progbyte          ; wrıte byte to serial memory
        acall ackpoll
        acall clrwel            ; reset the wrıte enable bıt
        mov dptr,#pageno        ; dptr= page number
        mov r0,#rambuff         ; r0 = ram buffer address
        acall randomread        ; setup the address poınter and read
        mov @r0,a               ; the fırst byte ın the page, save
        inc r0                  ; ıt to the buffer
        mov a,# hıgh pageno ; load the upper byte of address
        mov r1,#1fh             ; byte count
        acall seqread           ; read/store the remaınıng data
        ajmp $                  ; end of maın
;*************************************************************
;** name: seqread
;** description: read sequentially from the serial memory.
;** function: this subroutine extracts contents of the serial memory and stores
;** them into the specified ram buffer. the total number of bytes to
;** read should be provided along with the buffer address. this
;** routine assumes that the address pointer has already been
;** initialized using the randomread routine.
;** calls: start, slavaddr, ınbyte, stopread
;** ınput: r0 = ram buffer base address, dph = high order address
```

```
;** r1 = number of bytes to read
;** output: none
;** register usage: a, r0, r1
;*********************************************************
seqread:
        acall start             ; start
        setb c          ; [c=1] read operatıon bıt
        acall slavaddr          ; send the slave address byte
seqreadloop:
        acall ınbyte    ; start readıng from the current address
        mov @r0,a       ; total number of bytes to read out of
        inc r0          ; serial memory
        djnz r01,seqreadnxt
        ajmp stopread           ; end of read operatıon
seqreadnxt:
        acall outack    ; send an acknowledge to the devıce
        ajmp seqreadloop
;*********************************************************
;** name: randomread
;** description: reads content of the serial memory at a specific location.
;** function: this subroutine sends out the command to read the content of
;** a memory location specified in the dptr.
;** calls: start, ınbyte, slavaddr, outbyte
;** ınput: dptr = address of the byte
;** output: a = read value
;** register usage: a
;*********************************************************
randomread:
        acall start             ; start
        clr c                   ; [c=0] wrıte operatıon bıt
        acall slavaddr          ; send the slave address byte
        mov a,dpl               ; load the lower byte of the page
        acall outbyte ; address and shıft out to the devıce
        mov a,dph
        acall start             ; start
        setb c          ; [c=1] read operatıon bıt
        acall slavaddr          ; send the slave address byte
        acall ınbyte    ; shıft ın a byte from the devıce
        ajmp stopread           ; end operatıon
;*********************************************************
;** name: stopread
;** description: terminate read operation.
;** function: this subroutine sends out the command to end reading content
;** of a serial memory.
```

```
;** calls: clockpulse, stop
;** ınput: none
;** output: none
;** register usage: none
;*********************************************************
stopread:
        setb p1.sdabit      ; make sure that the data lıne ıs hıgh ...
        acall clockpulse
        jmp stop            ; end operatıon
;*********************************************************
;** name: progpage
;** description: update a page of the serial memory.
;** function: this subroutine transfers the contents of the given buffer to the
;** serial memory. the caller program must supply the page number
;** of the serial memory to update and the base address of the
;** ram buffer.
;** calls: start, slavaddr, outbyte, stop
;** ınput: r0 = ram buffer base address, dptr = page number
;** output: none
;** register usage: a, r0, r1
;*********************************************************
progpage:
        acall start         ; start
        clr c               ; [c=0] wrıte operatıon bıt
        acall slavaddr      ; send the slave address byte
        mov a,dpl           ; load the lower byte of the page address
        anl a,#0e0h  ; mask out the unwanted lower bıts
        acall outbyte ; and shıft out to the devıce
        mov r1,#pagesize    ; transfer content of the ram buffer
progpagenxt:
        mov a,@r0    ; to the serial memory
        acall outbyte ; r0 should be poıntıng to the buffer
        mov a,#0ffh  ; cover up your tracks as buffer ıs
        mov @r0,a    ; read and wrıtten to the serial mem.
        inc r0       ; total number of bytes transferred
        djnz r01,progpagenxt      ; to the serial should not exceed the
                                  ; page sıze
        ajmp stop                 ; end of the operatıon
;*********************************************************
;** name: progbyte
;** description: write a byte to serial memory.
;** function: this subroutine transfers the contents of bytedata to the serial
;** memory. the address written to is contained in the slave address
;** (dph) and the lower byte is contained in (dpl). a two byte
```

```
;** address is required for both page and byte write commands.
;** calls: start, slavaddr, outbyte, stop
;** ınput: bytedata = byte to be written, dptr = address
;** output: none
;** register usage: a, dptr
;*********************************************************
progbyte:
        acall start             ; start
        clr c                   ; [c=0] wrıteoperatıon bıt
        acall slavaddr          ; send the slave address byte
        mov a,dpl               ; load the lower byte of the page address
        acall outbyte ; send out the lower byte of the address
        mov a,#bytedata     ; load the data to be sent ın the acc
        acall outbyte ; send out the data to the serıal memory
        ajmp stop               ; end of the operatıon
;*********************************************************
;** name: enprogwpr
;** description: enable updates to write protect register (wpr) of the serial ;** memory.
;** function: this subroutine writes the appropriate sequence to the serial ;** memory
;** to enable updating of the wpr. the progwpen and progbp routines
;** must call this subroutine before writes to the wpr are allowed.
;** once this sequence is activated, the only way to exit this mode
;** is by writing to the wpr or resetting the serial memory.
;** calls: randomread, setwel, setrwel
;** ınput: none
;** output: a = ınıtıal wpr value
;** register usage: a, b, dptr
;*********************************************************
enprogwpr:
        mov dptr,#wpr_addr          ;read the wpr content and
        acall randomread    ;test the status of
        mov b,a
        jnb b.wel,progwpr_1         ;the wel bıt and skıp ıf ıts set
        acall setwel            ;send set wel command
progwpr_1:
        jnb b.rwel,progwpr_2 ;check the rwel bıt and skıp ıf ıts  ; set
        acall setrwel           ;send set rwel command
progwpr_2:
        mov a,b
        ret
;*********************************************************
;** name: progbp
;** description: update block lock bits in wpr of the serial memory.
;** function: this subroutine writes to the wpr of the serial memory and
```

```
;** changes the bp1:0. the caller program must supply the new values
;** for the bp1:0 bits. this routine retains the original state of
;** the wpen bit.
;** calls: addrwpr, enprogwpr, outbyte, stop
;** ınput: a[1:0] = bl[1:0]
;** output: none
;** register usage: a, r0
;*********************************************************
progbp:
        anl a,#03h    ;mask out the unwanted bıts
        swap a              ;shıft the bpx bıts to the
        rr a          ;bıt posıtıons 4:3
        mov r0,a            ;save the bpx new values and
        acall enprogwpr     ;enable wrıtıng to the wpr
        anl a,#9ah    ;create the data pattern by maskıng
        orl a,#02h    ;ın the desıred bıt pattern and
        orl a,r0            ;savıng status of wpen bıt
        mov r0,a      ;set the bpx bıts per requested pattern
        acall addrwpr       ;generate wpr wrıte command
        mov a,r0            ;shıft out wpr pattern
        acall outbyte ;to the devıce
        ajmp stop
;*********************************************************
;** name: progwpen
;** description: update write protect enable bit in wpr of the serial ;**memory.
;** function: this subroutine writes to the wpr of the serial memory and
;** changes the wpen bit. the caller program must supply the new
;** value of the wpen bit. the state of the bp1:0 bits are preserved.
;** calls: addrwpr, enprogwpr, outbyte, stop
;** ınput: c
;** output: none
;** register usage: a, r0
;*********************************************************
progwpen:
        clr a         ; load the status flags
        rrc a         ; mask out the unwanted bıts
        mov r0,a ; save the wpen bıt new value and
        acall enprogwpr; enable wrıtıng to the wpr
        anl a,#9ah    ; create the data pattern by maskıng
        orl a,#02h    ; ın the desıred bıt pattern and
        orl a,r0            ; savıng status of wpen bıt
        mov r0,a ; set the wpen bıt per as requested
        acall addrwpr       ; generate wpr wrıte command
        mov a,r0      ; shıft out wpr pattern
```

```
        acall outbyte ; to the devıce
        ajmp stop
;*********************************************************
;** name:              setwel
;** description:       set the write enable latch (wel) bit in the wpr of the
;**                    serial memory.
;** function:   this subroutine writes to the wpr of the serial memory and
;** sets the wel bit.
;** calls:             addrwpr, outbyte, stop
;** ınput:             none
;** output:     none
;** register usage: a
;*********************************************************
setwel:
        acall addrwpr       ; generate wpr wrıte command
        mov a,#welon        ; shıft out wel-on pattern
        acall outbyte ; to the devıce
        ajmp stop
;*********************************************************
;** name:              clrwel
;** description:       reset the write enable latch (wel) bit in the wpr of the ;**
                serial memory.
;** function:   this subroutine writes to the wpr of the serial memory and
;**                    resets the wel bit.
;** calls:             addrwpr, outbyte, stop
;** ınput:             none
;** output:     none
;** register usage: a
;*********************************************************
clrwel:
        acall addrwpr       ; generate wpr wrıte command
        clr a           ; shıft out wel-off pattern
        acall outbyte ; to the devıce
        ajmp stop
;*********************************************************
;** name:      setrwel
;** description:set register write enable latch bit in the wpr of the serial
;**memory.
;** function: this subroutine writes to the wpr of the serial memory and
;**sets the rwel bit.
;** calls: addrwpr, outbyte, stop
;** ınput: none
;** output: none
;** register usage:   a
```

```
;*********************************************************
setrwel:
        acall addrwpr          ; generate wpr wrıte command
        mov a,#rwelon  ; shıft out rwel-on pattern
        acall outbyte ; to the devıce
        ajmp stop
;*********************************************************
;** name: addrwpr
;** description: ınitiate write operation to the wpr of the serial memory.
;** function: this subroutine issues the wpr address and write instruction
;** to the serial memory.
;** calls: start, slavaddr, outbyte
;** ınput: none
;** output: none
;** register usage: a
;*********************************************************
addrwpr:
        mov dptr,#wpr_addr
        acall start     ; start [ c = operatıon bıt ]
        clr c           ; [c=0] wrıte operatıon bıt
        acall slavaddr          ; send the slave address byte
        mov a,dpl     ; load the lower byte of address
        ajmp outbyte          ; and shıft out to the devıce
;*********************************************************
;** name: slavaddr
;** description: build the slave address for the serial memory.
;** function: this subroutine concatenates the bit fields for device ıd,
;** the high address bits and the command bit. the resultant
;** byte is then transmitted to the serial memory.
;** calls: outbyte
;** ınput: dph = page number high addr.
;** c = command bıt (=0 wrıte, =1 read)
;** output: none
;** register usage: a
;*********************************************************
slavaddr:
        mov a,dph
        rlc a           ; merge the command bıt
        xrl a,#deviceıd         ; and the devıce select bıts
        anl a,#hiaddrmask  ; wıth the upper byte of
        xrl a,#deviceıd                 ; page address
        ajmp outbyte                    ; send the slave address
;*********************************************************
;** name: outbyte
```

```
;** description: sends a byte to the serial memory
;** function: this subroutine shifts out a byte, msb first, through the
;** assigned sda/scl lines on port 1.
;** calls: clockpulse, getack
;** ınput: a = byte to be sent
;** return value: none
;** register usage: a
;*********************************************************
outbyte:
        setb c
outbyteloop:
        rlc a                   ; shıft out the byte, msb fırst
        jnc outbyte0
        setb p1.sdabit
        ajmp outbyte1
outbyte0:
        clr p1.sdabit
outbyte1:
        acall clockpulse            ; clock the data ınto the serial memory
        cjne a,#80h,outbytenxt      ; memory, skıp ıf not done
        jmp getack              ; check for an ack from the devıce
outbytenxt:
        clr c                   ; loop ıf all the bıts have
        ajmp outbyteloop    ; not been shıfted out
;*********************************************************
;** name: ınbyte
;** description: shifts in a byte from the serial memory
;** function: this subroutine shifts in a byte, msb first, through the
;** assigned sda/scl lines on port 1. after the byte is received
;** an ack bit is sent to the serial memory.
;** calls: clockpulse
;** ınput: none
;** return value: a = received byte
;** register usage: a
;*********************************************************
ınbyte:
        mov a,#00000001b
        setb p1.sdabit              ; sda lıne forced hıgh
ınbytenxt:
        acall clockpulse            ; clock the serial memory and shıft
        rlc a    ; ınto acc. the logıc level on the sda
        jnc ınbytenxt ;lıne. the devıce outputs data on sda
        ret             ; msb fırst
;*********************************************************
```

```
;** name: clockpulse
;** description: generate a clock pulse
;** function: this subroutine forces a high-low transition on the assigned ;**scl
;** pin of port 1. ıt also samples and returns the sda line state
;** during high clock period.
;** calls: none
;** ınput: none
;** return value: c = sda line status
;** register usage: none
;*************************************************************
clockpulse:
        setb p1.sclbit          ; based on a 12mhz system crystal the
        nop             ; bus cycle tıme ıs 1 mıcrosec.
        nop
        clr c
        jnb p1.sdabit,clockpulselo
        setb c
clockpulselo:
        clr p1.sclbit   ; lower the clock lıne
        ret
;*************************************************************
;** name: outack
;** description: send out an ack bit to the serial memory
;** function: this subroutine clocks an ack bit to the serial memory.
;** the ack cycle acknowledges a properly received data by lowering
;** the sda line during this period (9th clock cycle of a received
;** byte).
;** calls: clockpulse
;** ınput: none
;** return value: none
;** register usage: none
;*************************************************************
outack:
        clr p1.sdabit  ; make sure that the data lıne ıs low ...
        jmp clockpulse        ; clock out the ack cycle
;*************************************************************
;** name: getack
;** description: clock the serial memory for ack cycle
;** function: this subroutine returns the sampled logic level on the sda ;**during
;** high clock cycle. the serial memory holds the sda line hıgh if
;** it does not receive the correct number of clocks or it's stuck in
;** a previously initiated write command.
;** calls: clockpulse
;** ınput: none
```

```
;** return value: c = acknowledge bit
;** register usage: none
;*********************************************************
getack:
        setb p1.sdabit          ; force sda lıne hıgh
        acall clockpulse        ; generate one clock pulse
        ret
;*********************************************************
;** name: ackpoll
;** description: wait for an ack from the serial memory
;** function: this subroutine monitors the serial memory response
;** following a dummy write command. the program returns when it
;** either receives an ack bit or the maximum number of tries is
;** reached.
;** calls: start, slavaddr, stop
;** ınput: none
;** return value: c = acknowledge bit [=0 ack ,=1 no ack was received]
;** register usage: a, r1, dptr
;*********************************************************
ackpoll:
        mov r1,#maxdelay  ; load max no. of ack pollıng cycle
        mov dptr,#pageno  ; d = page number of the serial memory
ackpollnxt:
        acall start       ; start the ack poll cycle and
        clr c             ; [c=0] wrıte operatıon bıt
        acall slavaddr    ; send the slave address. then
                          ; monıtor the sda lıne for an ack from
                          ; the serial memory. termınate the
        acall stop        ; operatıon by a stop condıtıon.
        jnc ackpollexit   ; exıt ıf the ack was receıved
        djnz r1,ackpollnxt ; loop whıle the maxımum no. of cycles
                          ; have not expıred. else return wıth c=1
ackpollexit:   ret
;*********************************************************
;** name: start
;** description: send a start command to the serial memory
;** function: this subroutine generates a start condition on the bus. the
;** start condition is defined as a high-low transition on the sda
;** line while the scl is high. the start is used at the beginning
;** of all transactions.
;** calls: none
;** ınput: none
;** return value: none
;** register usage: none
```

```
;*********************************************************
start:
        setb p1.sdabit        ; force the sda lıne hıgh
        setb p1.sclbit        ; force the scl clock lıne hıgh
        clr p1.sdabit  ; before takıng the sda low
        nop
        nop
        nop
        nop
        clr p1.sclbit   ; force the scl low
        ret
;*********************************************************
;** name: stop
;** description: send stop command to the serial memory
;** function: this subroutine generates a stop condition on the bus. the stop
;** condition is defined as a low-high transition on the sda
;** line while the scl is high. the stop is used to indicate end
;** of current transaction.
;** calls: none
;** ınput: none
;** return value: none
;** register usage: none
;*********************************************************
stop:
        clr p1.sdabit  ; force the sda low before takıng
        setb p1.sclbit        ; the scl clock lıne hıgh
        nop
        nop
        nop
        nop
        setb p1.sdabit        ; force the sda hıgh (ıdle state)
        ret
;*********************************************************
;** name: reset
;** description: resets the serial memory
;** function: this subroutine is written for the worst case. system interruptions
;** caused by brownout or soft error conditions that reset the main
;** cpu may have no effect on the internal vcc sensor and reset
;** circuit of the serial memory. these are unpredictable and
;** random events that may leave the serial memory interface
;** logic in an unknown state. ıssuing a stop command may not be
;** sufficient to reset the serial memory.
;** calls: start, stop
;** ınput: none
```

```
;** return value: none
;** register usage: r0
;*************************************************************
reset1:
        mov r0,#0ah ;apply 10 clocks to the devıce. each
resetnxt:
        acall start     ;cycle consısts of a start/stop
        acall stop      ;thıs wıll termınate pendıng wrıte
        djnz r00,resetnxt     ;command and provıdes enough clocks
        ret             ; for remaınıng bıts of a read operatıon
teststring:
        db 'xıcor makes ıt memorable!'
        db 00
        end
```