MİKROİŞLEMCİ KONTROLLU ÇOK KAVŞAKLI TRAFİK SİSTEMİ

Halil KARAKAŞ

Yüksek Lisans Tezi
Elektrik Elektronik Mühendisliği
Anabilim Dalı

1992

MIKROIŞLEMCI KONTROLLU ÇOK KAVŞAKLI TRAFIK SISTEMI

Halil KARAKAŞ

Anadolu Universitesi
Fen Bilimleri Enstitüsü
Lisansüstü Yönetmeliği Uyarınca
Elektrik - Elektronik Mühendisliği Anabilim Dalı
Elektronik Bilim Dalında
YUKSEK LISANS TEZI
Olarak Hazırlanmıştır.

Danışman: Doç. Dr. Hamdi ATMACA

ŞUBAT - 1992

Halil KARAKAŞ' ın 'Yüksek Lisans' tezi olarak hazırladığı 'MİKROİŞLEMCİ KONTROLLÜ ÇOK KAVŞAKLI TRAFİK SİSTEMİ' başlıklı bu çalışma, jürimizce lisansüstü yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

Üye: Prof. Dr. Atila BARKANA

Üye: Doç. Dr. Hamdi ATMACA

Üye: Y. Doç. Dr. Osman PARLAKTUNA

Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 27 MAYIS 1992 gün ve 314-5 sayılı kararıyla onaylanmıştır.

Prof. Dr. Rüstem KAYA
Enstitü Müdürü

İÇİNDEKİLER

İÇİNDEKİLER (devam)

ÖZET

Bu tez çalışmasında belirlenen yol kavşaklarındaki trafik ışıklarının bir merkezden kontrolü esas alınmış ve tasarlamıştır. Her bir kavşakta bulunan mikroişlemciler ile merkezi mikroişlemci arasında yıldız - çevre bağlantısı (star - loop connection) kurulmuş ve seri bir haberleşme hattı oluşturulmuştur. Merkezi mikroişlemciden her kavşakta bulunan mikroişlemciye veya her bir kavşaktaki mikroişlemciden merkezi mikroişlemciye bilgi transferi yapmak mümkün olmuştur. Merkezi kontrol sistemiyle kavşaklardaki trafik ışıklarının devre süreleri günün trafik yoğunluğuna göre hesaplanacak ve her kavşaktaki mikroişlemciye seri veri (data) hattından gönderilecektir. Bu şekilde yapılan düzenleme ile trafik akışındaki gecikmeler ve zaman kaybı ortadan kaldırılacaktır.

SUMMARY

In this thesis, the control of the traffic lights in road confluents from a central microprocessor is aimed, and is planned. Star - loop connection is constructed between central microprocessor and microprocessors that exist in each confluent, therefore a serial communication link is established between them. It is possible to make data transfer from central microprocessor to microprocessors that exist in each confluent or from microprocessors that exist in each confluent to central microprocessor. The time period of the traffic lights in confluents will be calculated according to the traffic density of the day, and will be sent from serial data link to microprocessors that exist in each confluent. In this case, delays in traffic flow and time loss will be avoided in rush hours.

TEŞEKKÜR

Çalışmalarımda her aşamada bana yardımcı olan ve beni yönlendiren değerli hocam Doç. Dr. Hamdi ATMACA 'ya, Yard. Doç. Dr. Osman PARLAKTUNA 'ya ve mesai arkadaşlarım Arş. Gör. Hakan TORA ,Arş. Gör. Salih EREN ve Arş. Gör Umit KUNKÇU 'ye ve benden manevi desteğini esirgemeyen sevgili arkadaşım Selda DURAN 'a teşekkür ederim.

ŞEKİLLLER DİZİNİ

1.GİRİŞ

Sinyaller, bir diğer deyişle ışıklı işaretler, yollar üzerinde ve özellikle kavşaklarda düzenli ve güvenli bir akım sağlamak için kullanılan trafik kontrol gereçleridir.

İlk olarak 1868 yılında Londra' da el ile yönetilen semaforlar biçiminde kullanılan trafik sinyalleri gece görünümlerini sağlamak amacı ile gaz lambaları ile aydınlatılmıştır. Kırmızı ve yeşil ışıklı ilk sinyalizasyon tesisi 1914 yılında A.B.D.'nde Cleveland'da kurulmuş, 1920 yılında Detroit'te sarı ışıklar da kullanılmıştır. 1924 yılından sonra Avrupa ülkelerinde de kullanılmaya başlanan ışıklı sinyaller özellikle 1950 yılından sonra büyük gelişme göstermiştir (1).

Herhangi bir yerde sinyalizasyon tesisi kurulması için aşağıdaki maddelerden en az birinin gerçekleştirilmesi gerekmektedir:

a) Kesişen akımlardan veya geometrik özelliklerden dolayı oluşan gecikmeleri, sıkışmaları ve tıkanıklıkları önlemek,

b) Taşıtların diğer taşıtlarla veya yaya geçitleri ile kesiştikleri noktalarda güvenli bir geçiş düzeni sağlamak ve kaza ihtimalini azaltmak,

c) Taşıt ve yaya yoğunluklarını göz önünde tutarak, akım yönlerine geçiş hakkı veya önceliği verirken uyumlu bir zaman dağıtımını yapmak,

d) Yüklü trafik yoğunluğu olan bir yol üzerindeki taşıtları zaman zaman durdurarak tali yollardaki trafiğe ve yayalara da geçiş olanağı sağlamak.

Trafik güvenliği ve kontrolü için kullanılan yatay ve düşey işaretlemelerde (Yol çizgilerinde ve trafik levhalarında) olduğu gibi, ışıklı işaretlerin de aşağıdaki dört niteliğe sahip olmaları gereklidir.

1. Sürücü ve yayaların dikkatini çekmelidir.

2. Basit ve kesin anlamları olmalıdır.

3. Sürücü ve yayaların saygı göstermeleri ve uymaları sağlanmalıdır.

4. Özellikle sürücülere intikal ve reaksiyon için yeterli zaman tanınmalıdır.

Trafik sinyalizasyonu sistemlerinin gerek projelendirilmesi gerekse uygulaması oldukça basit görünmekle birlikte küçümsenmemeli, kullanılacak cihaz ve gereçler ihtiyaca uygun biçimde titizlikle seçilmeli ve ne yetersiz nede fazla olmalıdır. Zaman dağıtımlarında taşıt ve yaya güvenliğine titizlik gösterilmeli, değişik akım yönlerine verilen geçiş hakkı süreleri yönlerin yoğunluklarının birbirlerine olan oranları ve sinyalize edilmiş tesisten geçiş süreleri ile uyumlu olmalıdır. Zaman dağıtımlarında özellikle akım değerlerinin saatlik, günlük, aylık, mevsimlik değişimleri göz önünde tutulmalı, ayrıca zaman aşımı nedeniyle akım özelliklerinin değişmesi halinde sürekli bir revizyon yapılmalıdır. Gereksiz olarak kurulmuş elemanları, yanlış yerleştirilmiş veya uyumsuz işletilen bir sinyalizasyon tesisi gecikmeleri büyük ölçüde arttırabilir ve bunun sonucu olarak sürücü ve yayaları ışıklara uymamaya alıştırabilir, hatta zorlayabilir (1,2).

Bu projede belli başlı kavşaklar göz önüne alınmış ve bu kavşaklardaki trafik akışını kontrol edebilecek bir sistem hazırlanmıştır. Bu sistemde dört ayrı kavşak bir merkezden bilgisayar aracılığı ile kontrol edilmektedir. Merkez bilgisayar ile her kavşakta bulunan yardımcı bilgisayarlar arasında bilgi alış verişini sağlamak için bir haberleşme hattı kurulmuştur. 3. Bölümde haberleşme ile ilgili açıklamalar yapılacaktır.

Kullanılan bu sistemde kavşakların kontrolü için gereken bilgiler günün belirli saatlerinde merkez bilgisayar aracılığı ile her kavşaktaki yardımcı bilgisayar'a gönderilecek ve sistemin düzgün bir şekilde çalışması sağlanacaktır. Kavşaklardaki taşıtların bekleme ve geçiş

süreleri günün değişen saatlerinde (Trafik yoğunluğuna göre) merkez bilgisayar tarafından kayıp zamanı en aza indirecek şekilde kontrol edilir.

2. SİNYALİZASYON PROJELERİNİN TEMEL İLKELERİ

2.1. Sinyal Devresi

Sinyalize bir tesiste birbirini izleyen değişik ışıklı sinyallerin bir devrine "Sinyal Devresi" veya kısaca "Devre" denir. Işıklı sinyallerin bir devreyi tamamlaması sırasında geçen toplam zamana da "devre süresi" veya "period" adı verilir.

Bir devre süresi iki bileşenden oluşur:

1. Taşıt akımları için ayrılan yeşil sürelerin toplamı
2. Yeşiller arasındaki sürelerin toplamı (kayıp zaman)

Bir sinyalize tesisin verimi büyük ölçüde devre süresinin uyumlu seçilmiş olup olmadığına bağlıdır. Bu nedenle devre süresinin saptanması sinyalizasyon projesinin hemen hemen en önemli bölümüdür. Pratikte uygulanabilecek minimum yeşil sürelerinin altına düşülmediği sürece, taşıt ve yayalara verilecek yeşil süreler devre süresinin uzunluğuna bağlıdır. Yeşiller arasında kalan süreler ise kabul edilen kıstaslara göre saptanır veya hesaplanır. Yeşiller arasında kalan sürelerin fazla olması bir devre içinde kayıp zamanı arttırır ve devre süresinin uzamasına yol açar.

Pratikte 30 sn'den daha kısa devre süresinin yeterli olacağı bir kavşak için "sinyalizasyon tesisinin kurulmasını gerektirmeyen bir kavşak" denebilir. Yayalara geçiş hakkı verecek olan yeşil süre 6 sn'den daha kısa olmamalıdır. 8 sn'den daha kısa olan taşıt yeşil süreleri ise pratikte uygulanabilme olanağı bulamamaktadır. Sinyalizasyon sistemlerinde 120 sn'den daha uzun devre süreleri uzun kuyrukların oluşmasına yol açar. Zorunlu şartlar altında 135-140 sn'lik devre süreleri maksimum olarak kabul edilebilir. Devre süresine bağlı olarak saptanan yeşil süreler için herhangi bir üst limit yoktur.(3)

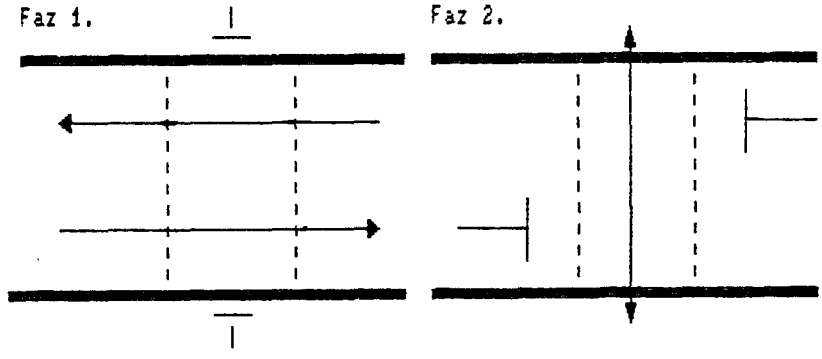
2.2. Faz Düzenleri

Bir sinyal devresi içinde belirli bir trafik akımını, veya birden fazla akımı aynı anda öngören kumanda şekline "faz" adı verilir. Sinyalizasyon Projesinde seçilecek faz sistemi kavşağa giriş olan yol sayısına ve kesişen trafik yoğunluğuna bağlıdır.

Sinyalize bir tesiste en az 2 ve en çok 4 fazlı sistemler uygulanır. Taşıt trafiği ile birlikte yaya trafiğinin çok yoğun olduğu bazı kavşaklarda, sinyalize yaya geçitlerinde olduğu gibi özel yaya fazları düzenlenebilir. Faz sayısının çok olması, her faz arasındaki yeşiller arası süreyi arttıracağından yeşiller arası sürelerin toplamı olan kayıp zamanın fazla olması ise devre süresinin uzamasını gerektireceğinden, proje hazırlanırken faz sayısının mümkün olduğu kadar azaltılmasına çalışılmalıdır.

2.2.1. Sinyalize yaya geçitleri

Kavşak olmayan yerlerdeki yaya geçitlerinin sinyalize edilmesinde bir taşıt, birde yayalar için olmak üzere iki faz kullanılır (Şekil 2.1).

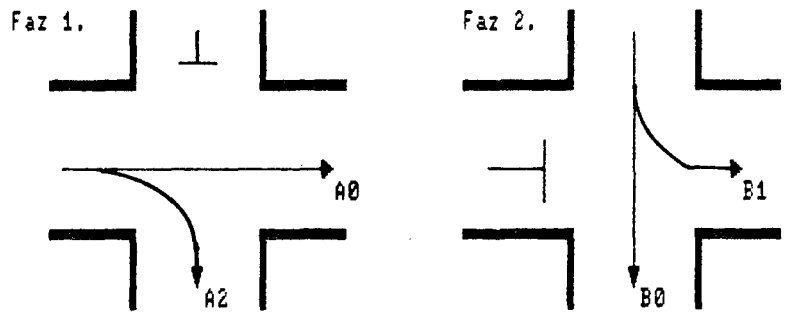


Şekil 2.1 Yaya Geçidi Fazları

2.2.2. Tek yönlü yolların kesiştiği kavşaklar

Tek yönlü yolların birleştiği üçlü kavşaklarda kavşağa doğru bir taşıt akımı olup, ana yola katılan trafik sola dönüş yapıyorsa sola dönüşün yoğun olması halinde sinyalizasyon gerekebilir. Bunun dışında bir kavşaktaki yönlerin tümü tek yönlü ise, sinyalizasyon tesisi kurulması için en az bir kesişme noktası öngörülür.

Dört beş kollu kavşaklarda Şekil 2.2 'de görüldüğü gibi bir fazlı bir düzen yeterlidir. Şekildeki örnekte B_1 sola dönüşü ile A_2 sağa dönüşünü yapan taşıtlar trafik kuralları uyarınca, aynı anda geçiş hakkı olan yayalara yol vereceklerdir.



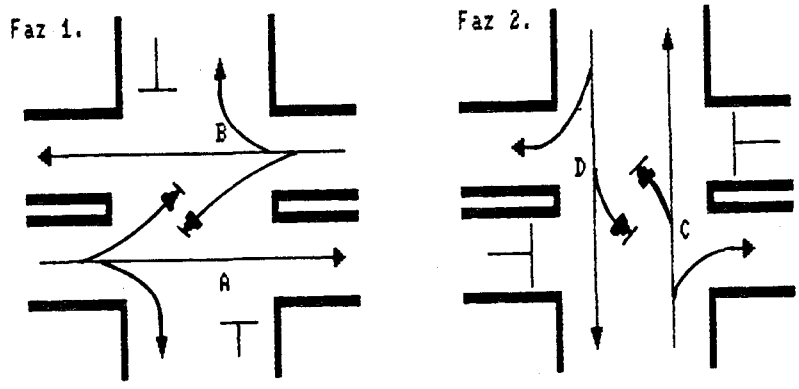
Şekil 2.2. Tek Yönlü Yol Kavşaklarında Fazlar

2.2.3. Sola dönüşlerin az olduğu kavşaklar

Çift yönde akımı bulunan yollarda sola dönüş yapan taşıt hacmi 60 taşıt/saat veya daha az ise, bunların sola dönüşleri için özel bir donanım yapmak gerekmez. Sola dönüş yapacak taşıtların keseceği trafik akımının arasında boşluk olmasa bile az sayıda taşıt fazların değişimi sırasındaki yeşiller arası süreden yararlanarak dönüş manevrasını rahatlıkla yapabilir. Ayrıca, dönüş yapacak taşıtların karşı yönünden kavşağa giren taşıtlar arasında zaman zaman boşluklar oluşuyorsa daha fazla sayıda taşıt bu boşluklardan yararlanarak sola dönüşünü tamamlayabilir. Kavşak alanının

geniş ve yeterli olması halinde, sola dönüş yapacak taşıtların kavşak içinde depo edilerek, yeşiller arası sürenin uzatılması suretiyle ek bir fazın uygulanmaması sağlanabilir.

Şekil 2.3 'te çift yönlü yolların kesiştiği dörtlü bir kavşakta sola dönüşlerin az olmasından yararlanılarak uygulanan iki fazlı bir düzen görülmektedir. Bu şekilde faz düzenine göre sağa dönüşler düz gidişler ile birlikte kesilmektedir. Kavşağın geometrik özellikleri uygun olursa sağa dönüşler bütün fazlarda serbest bırakılabilir, ancak bu durumda sağa dönüş yapacak taşıtların geçiş hakkı olan taşıtları sıkıştırmamasına ve sağa sapmadan önceki yaya geçidinin yayalara açık olmamasına dikkat edilmelidir. Aynı prensibler çok fazlı düzenlerde de söz konusudur.



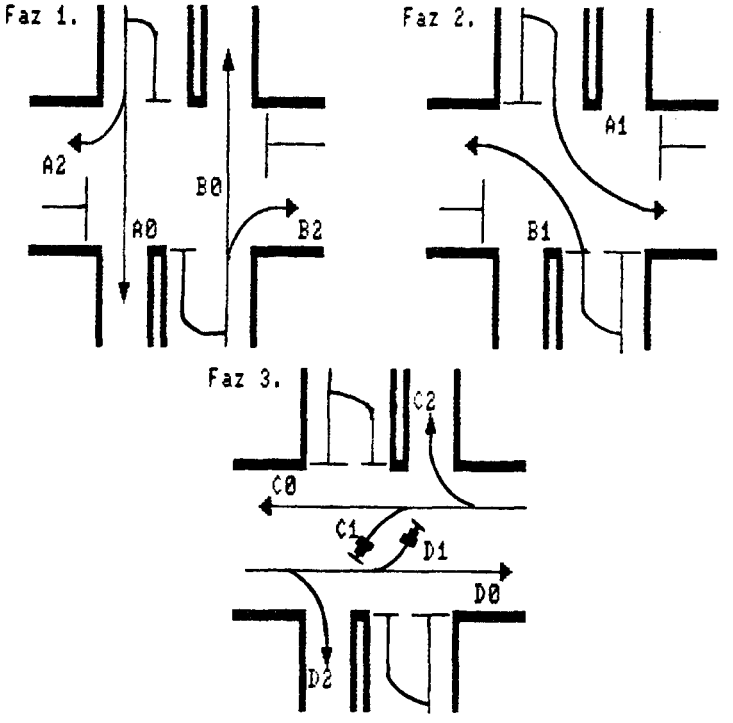
Şekil 2.3 Sola Dönüşün Az Olduğu Kavşakta Fazlar

2.2.4. Üç ve dört fazlı uygulamalar

Düz gidiş yapan taşıt yoğunluğunun yanı sıra sola dönüş yapan taşıt sayısının artması halinde çok fazlı düzen uygulaması gerekmektedir. Çok fazlı sinyalize düzen kurulması tasarlanan kavşakların faz sayısının azaltılması olanakları araştırılmalı, bu amaçla tek bir çözüm ile yetinilmeyip değişik alternatifler denenmelidir. Çok fazlı uygulamanın giderilemiyeceği kavşaklarda hangi durumlarda nasıl bir düzenin uyumlu olacağı aşağıda belirtilmiştir:

1) Karşılıklı İki Sola Dönüşün Yoğun Olduğu Kavşaklar:

a) Sola dönüşler için yeterli sığınma şeridi varsa şekil 2.4 te gösterilen üç fazlı düzenin uygulanması yerindedir. Bu düzende yoğun sola dönüşler için bir faz ayrılmaktadır.



Şekil 2.4. Sola Dönüşleri Ayrılan Üç Fazlı Düzen

b) Sola dönüşler için yeterli sığınma şeridi yoksa şekil 2.5 teki üç fazlı düzen kullanılmaktadır. Bu düzende yoğun sola dönüş akımlarına düz gidiş akımları ile birlikte geçiş hakkı verilmektedir.

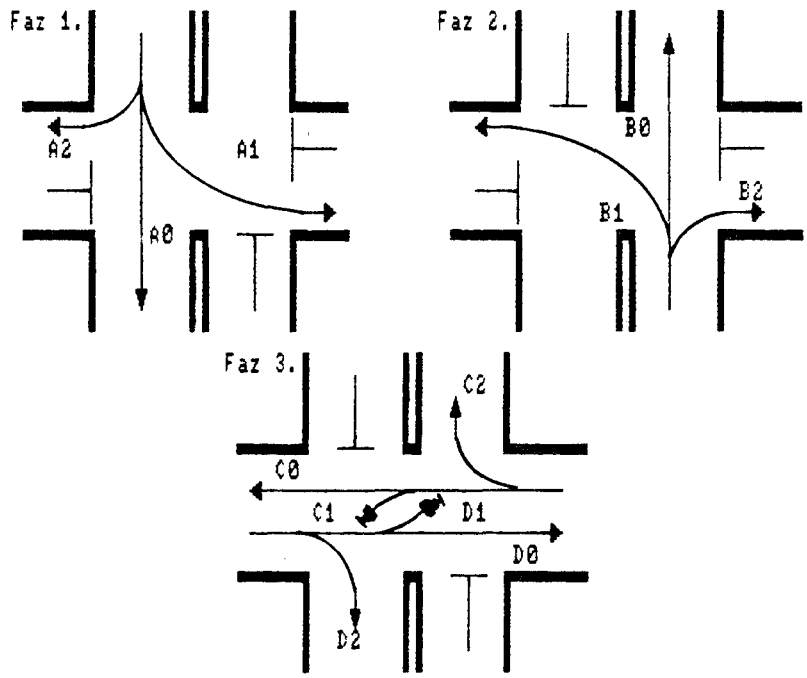
2) İkiden Fazla Sola Dönüşün Yoğun Olduğu Dörtlü Kavşaklar:

Bu kavşaklarda dört fazlı düzen uygulanmalıdır.

a) Sola dönüşler için yeterli sığınma şeridi bulunmayan kavşaklarda Şekil 2.5 deki 1. ve 2. fazlardan sonra C ve D yönlerine ayrı geçiş hakkı verilir. Bu biçimde tasarlanan dört fazlı uygulamalarda fazlar değişik sıralarda

düzenlenebilir, ancak kayıp zamanın minimum tutulması için birbirini izleyen fazların saatin akış yönünde olmamasına çalışılmalıdır. Bunun amacı, kavşağı boşaltmakta olan son taşıt ile kavşağa girecek ilk taşıtın arasındaki uzaklığın mümkün mertebe fazla olmasını sağlamaktır.

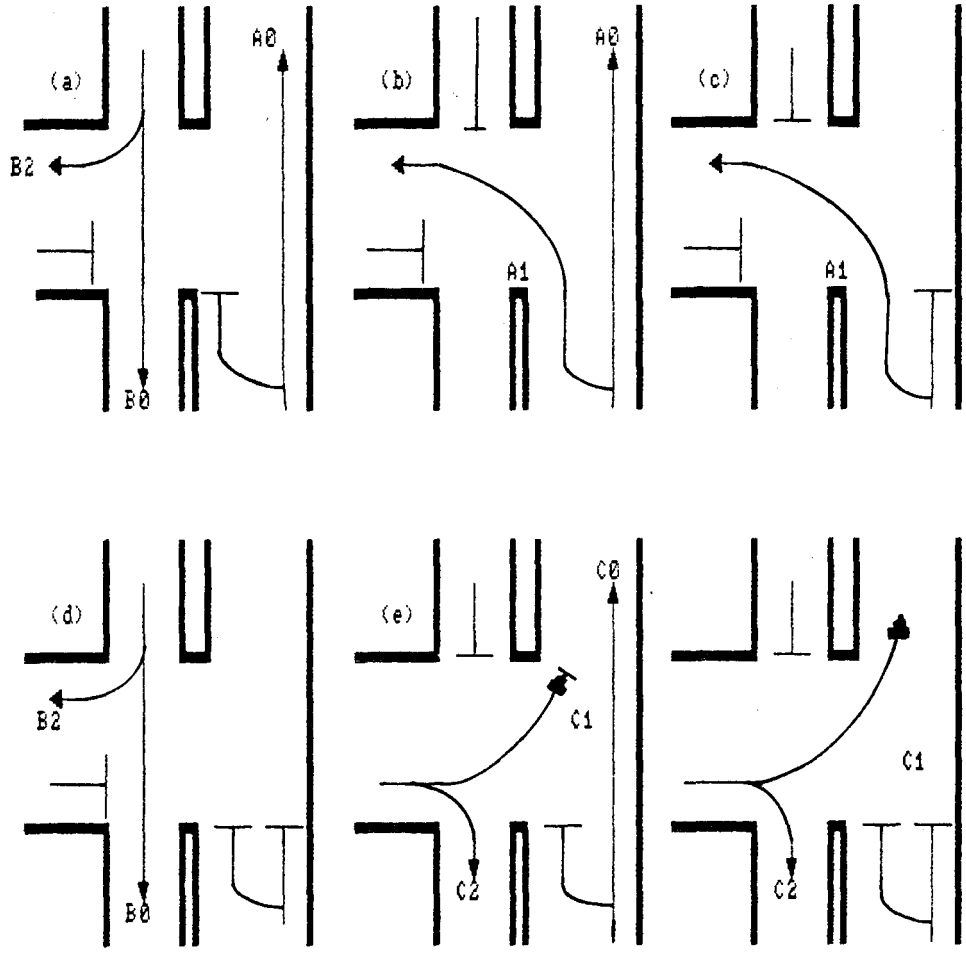
b) Sola dönüşler için yeterli sığınma şeritleri bulunan kavşaklarda yukarıda açıklanan dört fazlı düzen uygulanabildiği gibi Şekil 2.4 te gösterilen 1. ve 2. fazlardan sonra 3. fazda C_0 ve D_0 düz gidişlerine geçiş hakkı verilir, 4. fazda da C_1 ve D_1 sola dönüş akımları sığınma şeritlerini boşaltırlar.



Şekil 2.5. Sola Dönüşleri Birlikte Üç Fazlı Düzen

3) Sola dönüşlerin yoğun olduğu T-kavşaklar

Bu kavşaklarda genellikle 3 fazlı bir düzen uygulanır. Kavşak geometrisine ve taşıt yoğunluklarının birbirine oranına göre Şekil 2.6 daki fazlardan üçü seçilir.



Şekil 2.6. T - Kavşaklarda Faz Düzenleri

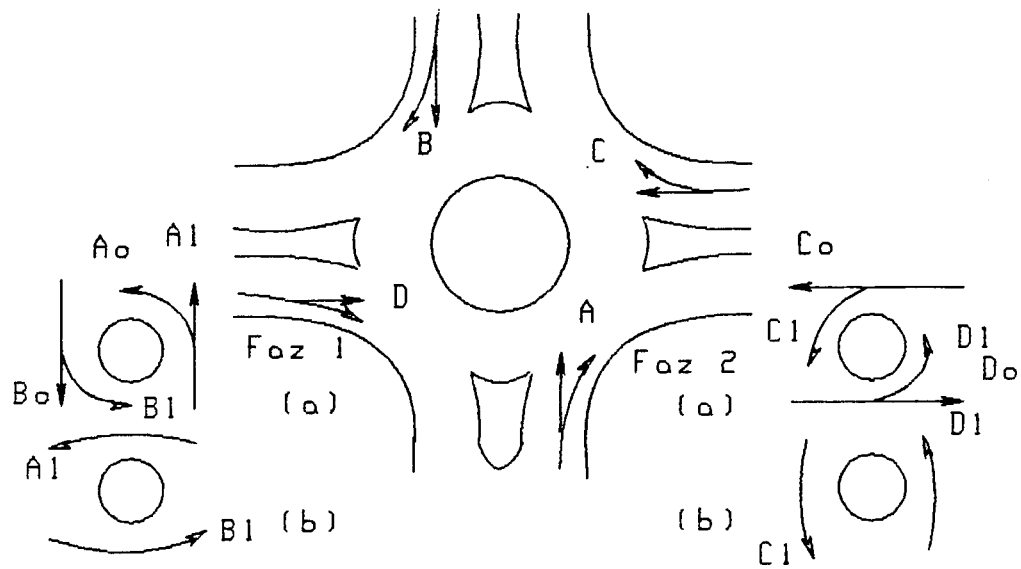
a) Ana yolda yeterli sığınma şeridi varsa 1. ve 2. faz olarak (a) ve (c) , sığınma şeridi yoksa (b) ve (d) kullanılır.

b) A_0 akımının B_0 akımından çok fazla olduğu durumlarda (a) hareketinden sonra 2. faz olarak (c) hareketi kullanılır.

c) 3. faz genellikle (f) hareketindeki gibi olmakla birlikte A_0 akımının çok yoğun olması veya ana yolu kesen yaya trafiği bulunmayıp B yönüne doğru kavşaktan açılan yolun A_0 ve C_1 akımlarının birbirine katılmaları halinde her ikisini birden taşıyabilecek nitelikte olması durumunda (e) hareketi uygulanabilir.

2.2.5 Ada çevresinde dönüş

Şekil 2.7 de sinyalize edilmiş bir dönel kavşak görülmektedir. İki fazlı düzenle yönetilen bu kavşağın her iki fazının (a) hareketinde düz gidiş ile birlikte kavşağa giren sola dönecek taşıtlar ortadaki adanın çevresinde dönerek depolanmakta, (b) hareketinde ise kavşağı boşaltmaktadırlar. Sola dönüş yapacak taşıtlar kavşak içine girmiş olduklarından fazların (b) hareketleri için verilecek süre çok kısa olabilmekte, bir sonraki fazda kavşağa girecek taşıtlar kavşağı boşaltmakta olan taşıtlarla aynı yönde seyrettiklerinden zaman kaybı önlendiği gibi trafik güvenliği de büyük ölçüde arttırılmaktadır. Burada yayalara yeşil yanmadığı durumlarda sağa dönüşler serbest bırakılmıştır (3,4).



Şekil 2.7. Ada Çevresinde Dönüş

2.3 Yeşiller Arası Süreler

2.3.1 Sarı süreler

Sarı ışıklı sinyalin amacı, taşıt sürücülerini geçiş hakkının sona ermiş olduğu hususunda uyararak, uzakta olanların duruşa geçmelerini ve duramayacak kadar yaklaşmış olanların kavşağı güvenle geçerek boşaltmalarını

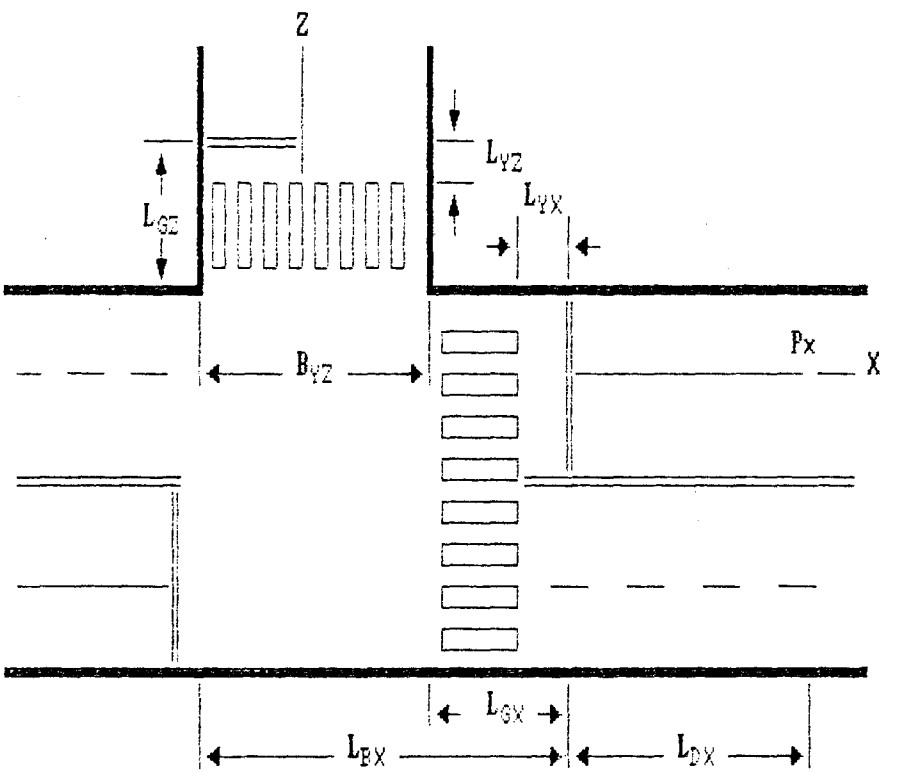
sağlamaktır. Bu nedenle, sinyalize bir tesiste devre süresi ve yeşil süreleri etkileyen en önemli faktörler taşıt yoğunluğu ve kompozisyonu olmasına rağmen, sarı süreler taşıtların sinyalize tesise yaklaşım hızlarına bağlıdır.

Şekil 2.8 de görülen basit T - tipi kavşakta, X yönüne geçiş hakkı verilirken kavşaktaki kritik ölçüler ve anlamları şunlardır:

L_{DX} = Duruş uzaklığı : X yönünden kavşağa yaklaşmakta olan taşıtın fren intikali ve reaksiyonu sırasında katettiği mesafe dahil olmak üzere güvenli duruş için kavşaktaki dur çizgisinden geride olması gereken uzaklık.

L_{BX} = Boşaltma uzaklığı : X yönünden kavşağa giren bir taşıt için dur çizgisinden kavşaktaki kesişme noktalarının sonuna kadar olan uzaklık.

L_{GX} = Giriş uzaklığı : X yönünden sonra geçiş hakkı elde edecek Z yönü için dur çizgisinden kavşaktaki ilk kesişme noktasına kadar uzaklık.



Şekil 2.8. Kritik Kavşak Ölçüleri

Şekil 2.8 deki Px noktası, X yönünden kavşağa giren taşıtlar için kritik bir noktadır. (Bu nokta aslında sabit olmayıp her taşıtın hızına göre değişik bir yerdedir, ancak proje hazırlanırken Px noktası taşıtların %85 hızına göre yaklaşık olarak alınır). Yeşil ışıktan sonra sarı ışık yandığında Px noktasına erişmemiş olan taşıt teorik olarak L_{DX} mesafesi içerisinde ve dur çizgisinden önce durabilecektir. Sarı ışık yandığı anda Px noktasını geçmiş olan bir taşıt ise dur çizgisine kadar duramayacağından, kırmızı ışık yanmadan kavşaktaki kesişme noktalarının bulunduğu alan içine girmiş olmalı, dolayısiyle (L_{DX} + L_{GX}) mesafesini katedebilmelidir. X yönünden kavşağa girmekte olan taşıtların %85 hızı Vx km/saat ise, gerekli sarı ışık süresi Yx şu şekilde bulunur.

$$Yx = \frac{3.6}{Vx} (L_{DX} + L_{gx}) \qquad (2.1)$$

Asfalt kaplaması olan bir yolda ıslak zemin ile lastikler arasındaki sürtünme katsayısı yaklaşık olarak 0.35 dir. İntikal ve reaksiyon süresi ise ortalama olarak 1.0 saniye kabul edilebilir. Bu şartlar için yaklaşım yolunun meyilsiz olduğu kabul edilerek gerekli sarı ışık süreleri Tablo 1. de gösterilmiştir.

Genellikle sarı ışık süreleri minimum 3 saniye olarak alınır. Pratik uygulamalarda yaklaşık olarak 70 km/saat %85 hız için 4 saniye, 90 km/st için 5 saniye sarı ışık süresi uyumlu kabul edilebilir.

Tablo 1. Sarı ışık Süreleri (Y)

%85 hız	Giriş uzaklığı(m)				
km/st	0	5	10	15	20
20	1.8	2.7	3.6	4.5	5.4
30	2.2	2.8	3.4	4.0	4.6
40	2.5	3.0	3.5	4.0	4.5
50	3.0	3.3	3.7	4.1	4.5
60	3.4	3.7	4.0	4.3	4.6
70	3.9	4.1	4.3	4.6	4.9
80	4.2	4.5	4.7	4.9	5.1
90	4.6	4.8	5.0	5.2	5.4
100	5.0	5.2	5.4	5.6	5.8

2.3.2 Kırmızı ve Sarı süreler

Herhangi bir yöne yeşil ışıkla geçiş izni verilmezden önce, harekete geçecek olan taşıtların hazırlanmaları ve zaman kaybetmemeleri için kırmızı ve sarı üniteler birlikte sinyal verirler. Bu ışıklı sinyal için metodik bir hesap uygulanmayarak duruma göre 2-3 saniyelik bir süre seçilir. Kırmızı ve Sarı ışıklı sinyalde taşıtların geçmemeleri ve beklemeleri gereklidir. Ancak yerel şartlara göre taşıt sürücüleri bu ışıklı sinyalde geçme eğilimi gösteriyorlarsa kırmızı ve sarı sürenin iki saniyeden uzun olmaması gerekir.

2.3.3 Koruma süreleri

Koruma süresi, geçiş hakkı sona eren bir yönden kavşağa girerek kavşağı boşaltan son taşıt ile, bundan sonraki fazda kavşağa girecek olan ilk taşıtın kesişme noktasında çarpışmamaları için fazlar arasında bırakılması gereken, ve yeşiller arası sürenin bir bölümünü oluşturan, kayıp bir zamandır. Güvenli bir koruma süresi hesaplamak için aşağıdaki kabulleri yapmak gerekir:

a) Kavşağı boşaltmakta olan taşıt daha önce kırmızı ışıkta beklemiş olan kuyruğun son elemanıdır ve kavşağı önündeki taşıtların arkasında ağır bir hızla terketmektedir.

b) Kavşağı terkedecek olan son taşıt sarı ışıklı sinyalde geçmektedir ve dur çizgisini geçtiği anda kırmızı ışıklı sinyal yanmaktadır.

c) Bir sonraki fazda kavşağa girmek üzere dur çizgisinde bekleyen taşıt yoktur. Kavşağa ilk girecek olan taşıt hızını düşürmeden yaklaşmakta olup (kırmızı ve sarı) sinyalden sonra yeşil ışığın yandığı anda dur çizgisini geçmektedir.

Şekil 2.8 de X yönünden gelen akım, Z yönünden gelen akım tarafından izleniyorsa standart bir otomobil boyu 5m. olarak alındığı taktirde, kavşağı terketmekte olan son taşıtın bir sonraki fazdaki ilk taşıt kendisine yetişmeden

katetmesi gerekli olan mesafe (L_{BX} + 5)m. dir. Z yönünden kavşağa girecek olan taşıtın ise, yeşil ışık yandığı andan sonra kesişme noktasına kadar katedeceği mesafe L_{GZ} dir. Her iki taşıt arasındaki koruma süresi aşağıdaki şekilde hesaplanır:

Boşaltma süresi :

$$t_{BX} = \frac{L_{BX} + 5}{V_B} \qquad (2.2)$$

Burada

V_B = Kavşağı boşaltmakta olan son taşıtın kavşak içindeki ortalama hızı (km/st),

L_{BX} = X yönünden kavşağa giren taşıtın kavşağı boşaltma mesafesi,

Giriş süresi ise

$$t_{GZ} = 3.6 \frac{L_{GZ}}{V_Z} \qquad (2.3)$$

Burada

V_Z = Geçiş hakkı açılacak olan yaklaşım yönünün %85 hızı,

L_{GZ} = Geçiş hakkı açılacak olan yaklaşım yönünün giriş uzaklığıdır.

Kesişme noktasındaki çarpışmayı önlemek için kullanılacak koruma süresi ise iki sürenin farkı olacaktır:

$$t_K = t_B - t_G \qquad (2.4)$$

Burada

t_K = Koruma süresi,

t_B = Boşaltma süresi,

t_G = Giriş süresidir.

Koruma süresinin hesaplanmasında en büyük zorluk V_B ve V_Z değerlerinin saptanmasıdır. Şehir içi kavşaklarının sinyalizasyon projelerinde %85 hız 50 km/saat şehir dışı yolları üzerindeki sinyalize kavşaklarda ise 80 km/saat olarak kabul edilmekle birlikte, gerçeğe yakın bir uygulama

yapılabilmesi için mevcut geometrik durum ve trafik şartları altındaki yaklaşık değerler gözlem yapılarak bulunmalıdır. Kavşakta beklemiş olan taşıtların kavşak içindeki ortalama hızları ise 25 km/st. mertebesinde kabul edilebilir.

2.3.4. Kayıp zaman

Kayıp zaman, bir devre içindeki bütün yeşiller arası sürelerin toplamıdır. Başka bir deyişle, bir devre içindeki sarı süreler ile koruma sürelerinin ve koruma sürelerinin dışında kalan hep - kırmızı sürelerin toplamıdır.

$$t_s = \sum_{i=1}^{n} Y_i + \sum_{i=1}^{n} t_{ki} + \sum_{i=1}^{n} t_R \qquad (2.5)$$

Burada

ts = Kayıp zaman,

tki=i fazı ile bir önceki faz arasındaki koruma süresi,

n = Faz sayısı,

Yi = i fazındaki akış yönü için verilen sarı süre,

ΣtR= Hep - kırmızı sürelerin toplamı (koruma süreleri hariç) dır.

2.4. Yaya Geçiş Süreleri

Yaya trafiğinin yoğun olmadığı kavşaklarda, yaya geçiş süreleri geçite dikey yöndeki taşıt trafiğinin kırmızı süre aralıklarında verilir. Yayalara geçiş hakkı verilmezden önce taşıtların kesinlikle durmuş olmaları gereklidir. Yayalara verilecek yeşil ışık süresinin 6 sn' den daha kısa olmamasına dikkat edilmelidir. Yayaların yeşil ışık süresi, yürüme hızı 1.2 m/sn kabul edilerek bulunur.

$$\text{Yaya Yeşili Süresi} \geq \text{Yaya Geçidi Uzunluğu } (B_Y) / 1.2 \qquad (2.6)$$

Yalnız yayalar için düzenlenmiş olan veya yaya trafiğinin yoğun olduğu geçitlerde ise her faz içinde karşıdan karşıya geçecek olan ortalama yaya sayısının göz önünde tutulması gerekir. Bunun için en uygun yöntem ne kadar sayıda yayanın geçidi ne kadar zamanda geçtiğini gözlemek suretiyle bulmaktır. Gözlem yapılmayan geçitlerde ise aşağıdaki

kıstaslar yaklaşık olarak uygulanabilir:

a) Yaya geçidinin her 1 m. genişliği için aynı anda karşılıklı birer yayanın geçebileceği düşünülmelidir.

b) Geçiti, önündekinin arkasından geçmesi gereken her yaya için yaya geçiti 1 m. den daha uzunmuş gibi düşünülmüştür.

Örneğin 5 m. genişliğinde ve 10 m. uzunluğunda bir yaya geçidinde her faz içinde bir taraftan diğerine 15 yaya geçmesi gerekiyorsa, yayalar her sırada 5 kişi olmak üzere 3 sıra halinde geçeceğinden By = 12 m. olarak alınmalıdır. Dolayısıyla yayanın yeşil süresi en az 10 sn, koruma süresi de en çok 5 sn olmalıdır.

2.5 Devre Süresi

Bir sinyalizasyon tesisinin projesinde, geometrik özellikleri, seçilen faz düzenini ve trafik şartlarını göz önüne alarak saptanan Devre süresinin hesaplanması projenin hemen hemen en önemli bölümüdür.

Devre süresinin hesaplanması için değişik kıstaslar uygulayan çeşitli yöntemler geliştirilmiştir. Sinyalize edilecek olan alandaki gecikmelerin minimuma indirilmesini ve bu alanın kapasitesinin de göz önünde tutulmasını öngören rasyonel yöntemler de, bazı parametrelerin saptanmasında deneysel sonuçların ya da ampirik denklemlerin kullanılmasından kaçınamamaktadır. Bunun nedeni, taşıt özellikleri ile yaya ve sürücü davranışlarının kesin teorilere bağlanamamış olmasıdır.

2.5.1 Etkili akım

Sinyalize bir tesisin herhangi bir fazında birden fazla yönde taşıt akımı aynı anda yer alabilir. Bir faz içindeki değişik taşıt akımlarından, otomobil birimi eşdeğeri olarak, şerit başına düşen yükü en yüksek olan, diğer bir deyimle otomobil birimi yükü en büyük olan yön o fazı kontrol eden yöndür.

Birbirini izleyen fazlardaki en yüksek yüklü olan yönlerin hepsine birden Etkili akım yönleri, veya daha kısa olarak Etkili akım adı verilir. Birbirini izleyen fazlardaki en yüksek trafik yüklerinin toplamına da Etkili akım yükü denir. Örneğin iki fazlı bir sinyalizasyon düzeni olan bir kavşağın her fazında iki ayrı yönde taşıt akımı varsa ve 1. fazdaki otomobil birimi yükleri **Ma** ve **Mb** , 2. fazdakiler de **Mc** ve **Md** ise, **Ma ≥ Mb** ve **Mc ≥ Md** olması halinde, etkili akım (**Ma, Mc**) ve etkili akım yükü de **ΣM = Ma + Mc** olacaktır.

2.5.2 Devre süresi bileşkeleri

Herhangi bir devre süresi, kullanılan yöntem ve uygulanan sistem ne olursa olsun, prensip bakımından iki bileşkeden oluşur.

a) Taşıt akımları için ayrılan yeşil süreler toplamı.
b) Kayıp zaman.

Devre Süresi,

$$C = \sum_{i=1}^{n} G_i + t_s \qquad (2.7)$$

ile ifade edilir. Burada

C : Devre süresi,
n : Faz sayısı,
G_i: i fazını kontrol eden akımın yeşil ışık süresi,
t_s: Kayıp zaman' dır.

Sinyalize yaya geçitlerinde, yayalar için verilen yeşil ışık süreleri taşıtlar için zaman kaybına yol açacağından, bu süreler hep-kırmızı süre kapsamına alınır ve dolayısıyla kayıp zaman olarak kabul edilir. Devre süresinin hesaplanmasında, kullanılan yöntemin özelliğine göre çeşitli faktörler göz önüne alınmakla birlikte, Etkili akım en önemli yeri tutar. Hemen hemen bütün yöntemlerde, etkili akım içindeki taşıtlara verilecek yeşil sürelerde devre süresinin ve aynı zamanda geçirecekleri akım yüklerinin fonksiyonu olurlar.

2.6 Optimum Devre Süresi

Uygulanan yöntem ne olursa olsun devre süresinin hesaplanmasında öncelikle ortalama akım değerleri kullanılır. Ortalama değerlere göre hesaplanan bir devre süresinin sakıncaları şu şekilde özetlenebilir:

a) Kavşağa ortalamadan daha düşük sayıda taşıt geldiği zaman yeşil sürelerde kayıplar oluşacaktır.

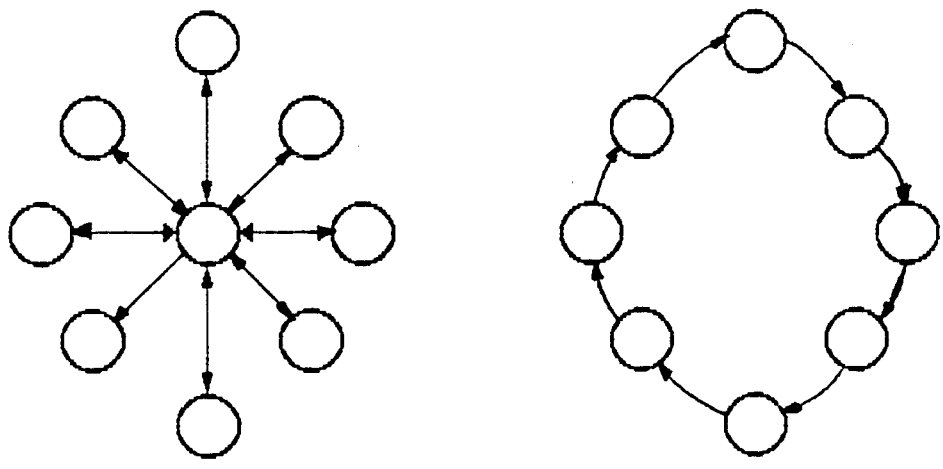
b) Kavşağa ortalamanın üstünde taşıt geldiği zaman kuyruklar uzayacak ve gecikmeler artacaktır.

Sinyalize edilen kavşağa ortalama değerlerden daha az taşıt geldiği zaman, kayıp yeşil süreler, ancak tesis uyarmalı olduğu taktirde önlenebilir. Kurulan tesis sabit zamanlı olursa devre süresinin kavşağa her yönden (veya isteğe göre yalnız bazı yönlerden) girecek olan taşıtların büyük bir oranının kendilerine verilecek yeşil süre içinde geçme imkanı elde etmesi, devre süresinin buna göre ayarlanması ile sağlanabilir. Bir diğer deyişle, etkili akımdaki bazı yönlerden veya hepsinden bir devre süresi içinde kavşağa giren taşıtların sayısı, ortalamanın üstünde olsa bile gelmekte olan taşıtların hepsini büyük bir ihtimalle (%90 - %95 gibi) geçirilmesi devre süresinin uzatılması ile gerçekleştirilebilir. Bu şekilde saptanan devre süresine Optimum Devre Süresi adı verilir (3,4).

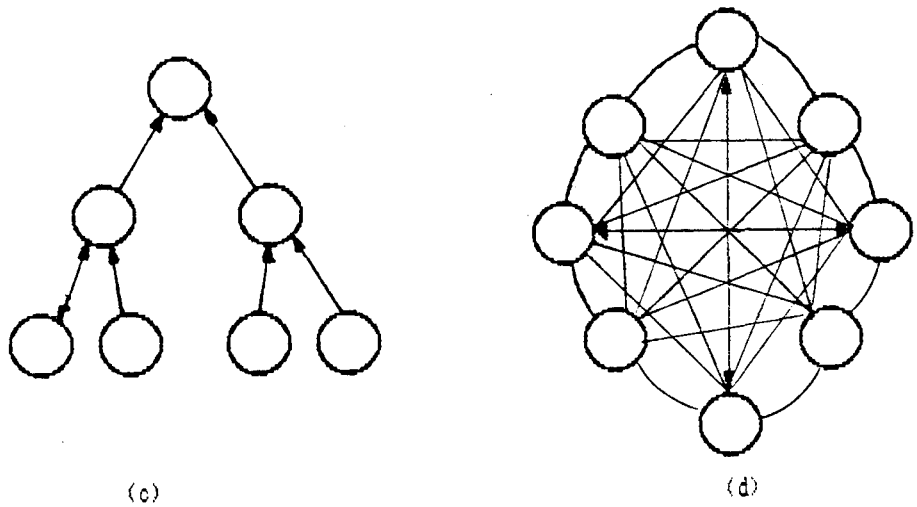
3. GENEL HABERLEŞME SİSTEMLERİ

Bu bölümde bilgisayarların birbirlerine değişik kombinasyonlarda bağlantı şekilleri üzerinde durulacaktır. Şekil 3.1 de dört değişik topolojik yapı görülmektedir. Şekil 3.1.a' da yıldız bağlantı (Star connection) görülmektedir. Burada yan ünitelerin çıkışları merkezdeki ana üniteye bağlanmıştır. Bilgi akışı çift yönlüdür. Yan ünitelerle merkez ünite arasında özel bir ayırıcı devre kullanılır. Şekil 3.1.b' de çevre bağlantı (Loop connection) görülmektedir. Bu bağlantı da üniteler kendi aralarında bir çevre oluşturur. Her bir ünitenin çıkışı bir sonrakinin girişine bağlanmıştır. Bu yapı tek olarak kullanıldığında merkezi ünite pozisyonu ortadan kalkar. İstenen ünite gerektiğinde merkezi ünite olarak kullanılabilir. Bilgi akışı tek yönlüdür. Şekil 3.1.c' de dallanma bağlantı (Tree connection) görülmektedir. Burada her bir ünite diğer bir üniteye oradanda merkezi üniteye bağlanmıştır. Bilgi akışı yan ünitelerden merkeze doğru tek yönlüdür. Şekil 3.1.d' de ise tam bağlantı (Completely connection) yapısı görülmektedir. Bu yapı, çok kalabalık üniteleri temel alarak çalışır. İstenen üniteden diğer üniteye bilgi transferi yapılabilir ve bilgi akışı çift yönlüdür (5,6,7).

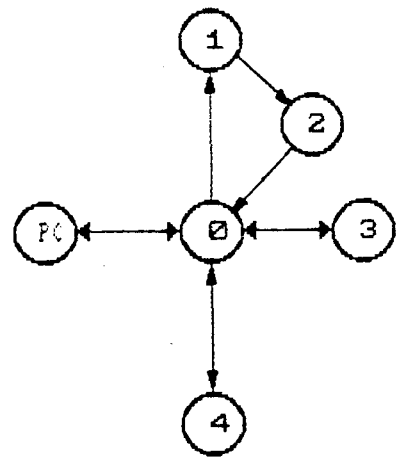
Gerçekleştirilen bu projede yıldız bağlantı (Star connection) ile çevre bağlantı (Loop connection) iç içe kullanılmıştır. Şekil 3.2 de bu bağlantı görülmektedir.



(a) (b)



Şekil 3.1 Haberleşmenin Topolojik Yapıları



Şekil 3.2. Oluşturulan Çok Mikroişlemcili Sistem Yapısı

3.1 Digital Haberleşmeye Giriş

Seri ve paralel veri (data) transferi bilgisayarlar arasında karşılıklı bilgi alış-veriş yöntemleridir. Her iki yöntemde geniş bir uygulama alanı bulmaktadır.

3.1.1 Paralel data transferi

Paralel veri (data) iletimi hızın önemli olduğu kısa mesafeli haberleşmelerde kullanılmaktadır. Bütün bir veri haberleşmesi çok hızlı bilgisayarlar arasında kurulmuştur ve bu haberleşmenin hızı saniyede milyon karaktere çıkabilmektedir.

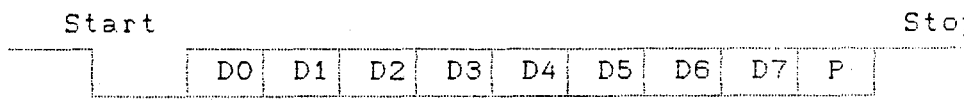
3.1.2 Seri veri transferi

Seri veri transferi genelde sayısal (digital) bilginin uzun mesafelere aktarılması gerektiği yerlerde kullanılır. Uzun mesafeli haberleşmede seri veri transferinin kullanılmasının tek sebebi bilgiyi taşımak için gerekli olan kablo sayısının az olmasından kaynaklanır. Fakat bu haberleşmede elde edilen transfer hızı bugünkü normal telefon hatları kullanıldığında ancak 4800 bit/sn (baud rate) dir.

İki tür seri haberleşme metodu mevcuttur. Bunlar sırası ile asenkron haberleşme ve senkron haberleşmedir.

a) Asenkron veri transferi

Asenkron veri haberleşmesinin bir diğer adı da başla - dur (start stop) haberleşmesidir. Çünkü senkronizasyonu sağlamak için gönderilen her 1 byte bilginin içinde başla ve dur bitleri bulunmaktadır. Asenkron haberleşmede clock (saat) sinyaline ihtiyaç yoktur. Çünkü senkronizasyon gönderilen başla ve dur biti ile sağlanır. Şekil 3.3 te asenkron haberleşmenin genel formatı verilmiştir. Gönderilecek her bilgiden önce lojik seviyesi 'low' olan bir sinyal gönderilir. Bunun ardından LSB ilk bilgi olacak şekilde veri gönderilir. Gönderilen verinin ardından istenirse veriye ait parite biti gönderilebilir. Stop biti her zaman veri bilgisinden sonra gelir ve lojik 'high' seviyesindedir (8).

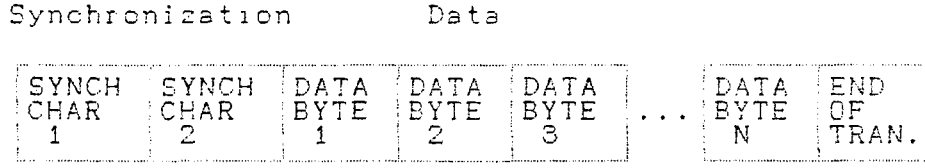


Şekil 3.3 Asenkron Data Haberleşme Formatı

b) Senkron haberleşme

Senkron haberleşme de senkronizasyon bir veya iki senkronizasyon karakteri gönderilerek sağlanır ve ardından

uzun bir veri bloğu gönderilir. Senkronizasyon karakterinin haricinde birde clock pulsler gönderilir. Senkron haberleşme değişik şekillerde gerçekleşebilir. Burada örnek olarak BISYNCH (Binary Synchronous communication) yöntemini kısaca açıklamak yeterli olacaktır. Şekil 3.4 'te görüldüğü gibi veri gönderecek olan ünite diğer üniteye haberleşmenin başında her biri tek byte (8 bitlik bilgi) olan iki adet senkronizasyon karakteri gönderir ve karakterlerden hemen sonra transfer edilecek olan veri bloğunu gönderir. Veri gönderme işlemi sonunda ise gönderme işleminin sona erdiğini belirten bir karakter gönderir ve işlemi bitirir (8).



Şekil 3.4 BISYNCH Haberleşme Formatı

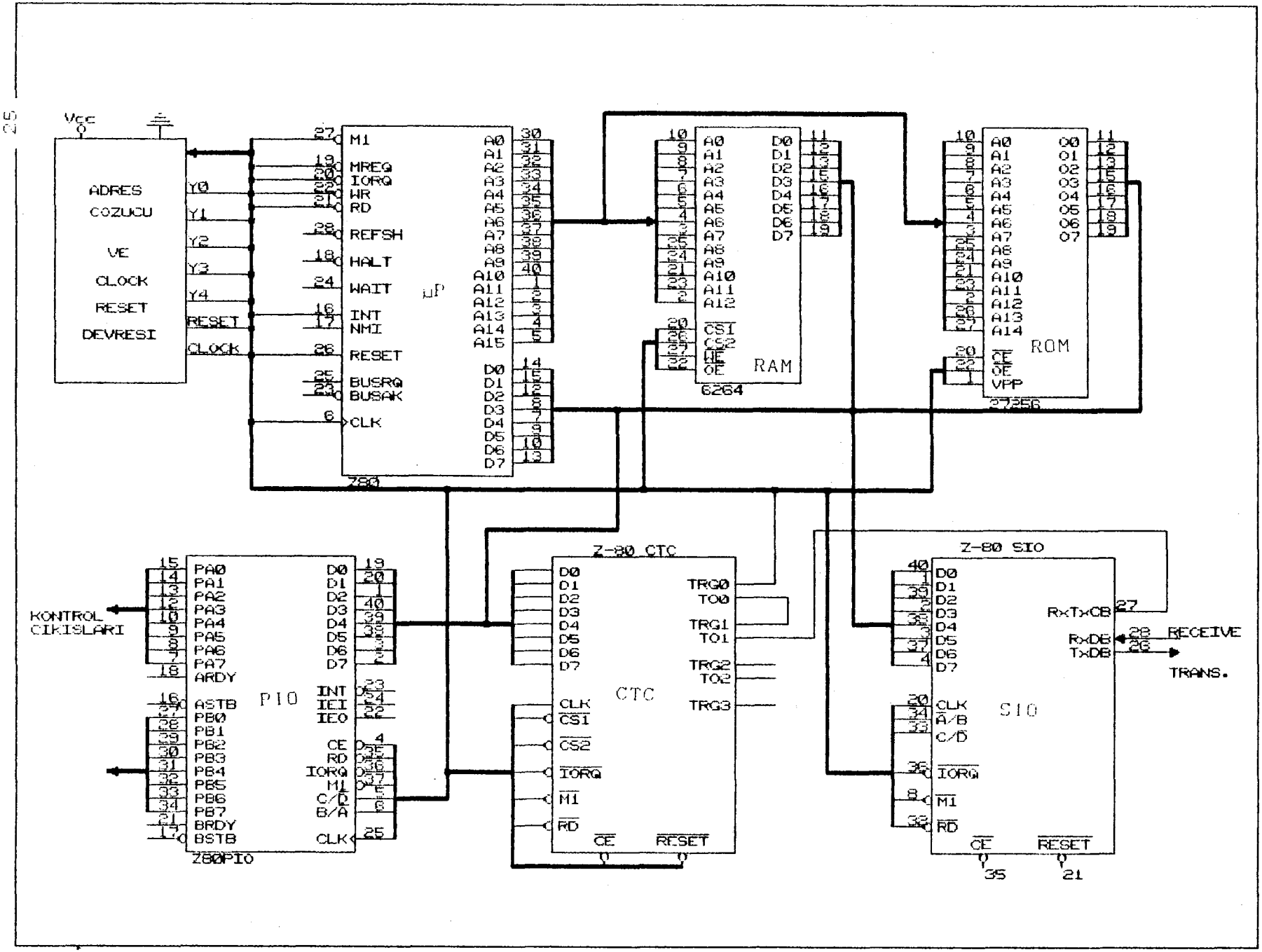
4. DONANIM

Bu projede kullanılan sistemin oluşturulması için 5 adet mikroişlemci kartı, 1 adet PC bilgisayar ve bu mikroişlemci kartlarının çok mikroişlemcili sistemi oluşturabilmeleri için bir ara devre hazırlanmıştır. Her mikroişlemci kartı üzerinde Z-80CPU, Z-80 PIO, Z-80 SIO, Z-80 CTC, 32Kbyte'lık ROM, 8Kbyte'lık RAM, gerekli adreslemeleri yapan adres çözücü devresi, saat sinyali devresi ve resetleme devresi mevcuttur. Ayrıca merkezi üniteyi PC bilgisayardan bağımsız olarak kullanabilmek için merkezi ünite için tuş ve gösterge devreleri ilave edilmiştir. Şekil 4.1 de bu mikroişlemci kartının bağlantısı görülmektedir.

4.1 Z-80 Mikroişlemcisi

Z-80 mikroişlemcisi Z-80, Z-80A, Z-80B, Z-80H olarak sırasıyla 2, 4, 6, 8 MHz 'lik mikroişlemciler olarak üretilir. Bu mikroişlemci, her ünitenin tüm işlevlerini yöneten, hafızalar ve giriş-çıkış uyum devreleri arasındaki bilgi işleme ve transfer işlemlerini gerçekleştiren birimdir. Z-80 CPU 8 bitlik mikroişlemciler içerisinde en popüler olanıdır. Z-80 NMOS teknolojisinin belirli bir şeklini kullanır ve bu sebepten dolayı da bir adet +5 V'luk kaynak ile çalışabilir. Bunun diğerlerinden farklı olan özellikleri ise bir tane saat sinyalinin oluşu, dinamik hafıza refresh lojiği ve interrupt modlarının fazla olmasıdır.

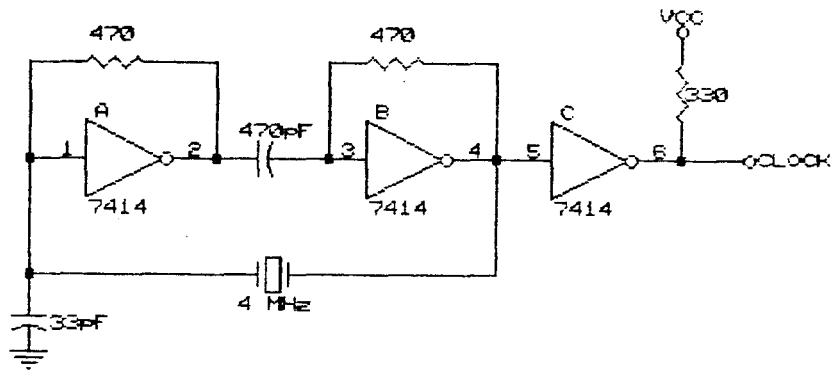
Donanım açısından da büyük üstünlüklere sahiptir. Mikroişlemci genel amaçlı yazaçlara (register), akümulatöre ve aritmetik mantık (ALU) ünitesine sahiptir. Diğer mikroişlemcilere kıyasla genel amaçlı yazaç sayısı daha fazladır. Komut sayısının ve yazaç sayısının fazlalığı programcıya büyük kolaylıklar sağlar. Hafıza ve giriş-çıkış (I/O) birimlerinin kontrolü için adres, bilgi ve kontrol giriş-çıkışlarına sahiptir.



Şekil 4.1 Mikroişlemci Kart

4.2 Saat (Clock) Devresi

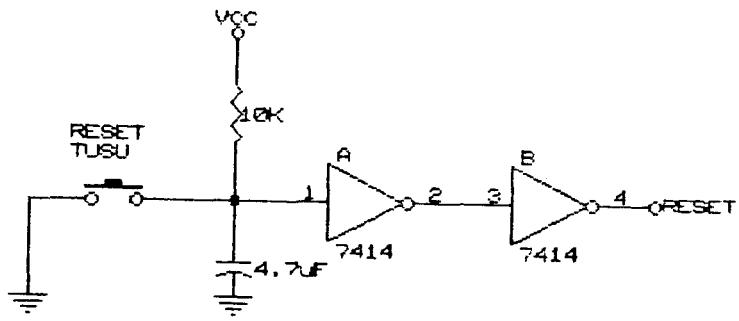
Z-80 mikroişlemci komutlarını fetch ve execute edebilmesi için gerekli kare dalgayı üreten kristalli osilatör devresidir. Saat devresi çıkışı sistemde saat sinyaline ihtiyaç duyulan diğer çevre birimlerine de bağlıdır. Saat devresi şekil 4.2 de verilmiştir.



Şekil 4.2 Saat (CLOCK) Devresi

4.3 Reset Devresi

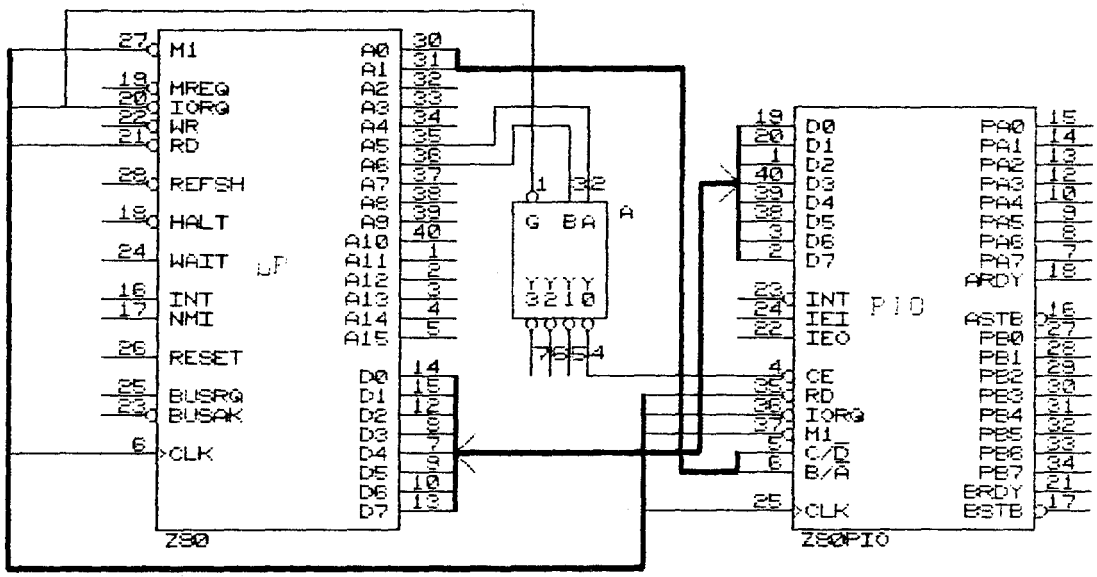
Bu devre Z-80 mikroişlemcisini ve bağlı olduğu diğer çevre elemanlarını resetlemek için kullanılır. Mikroişlemcinin resetlenmesinde program sayıcısına (program counter) 0000H bilgisi yüklenir ve böylece programın başlangıcına dönülür. Resetleme devresi şekil 4.3 de gösterilmiştir.



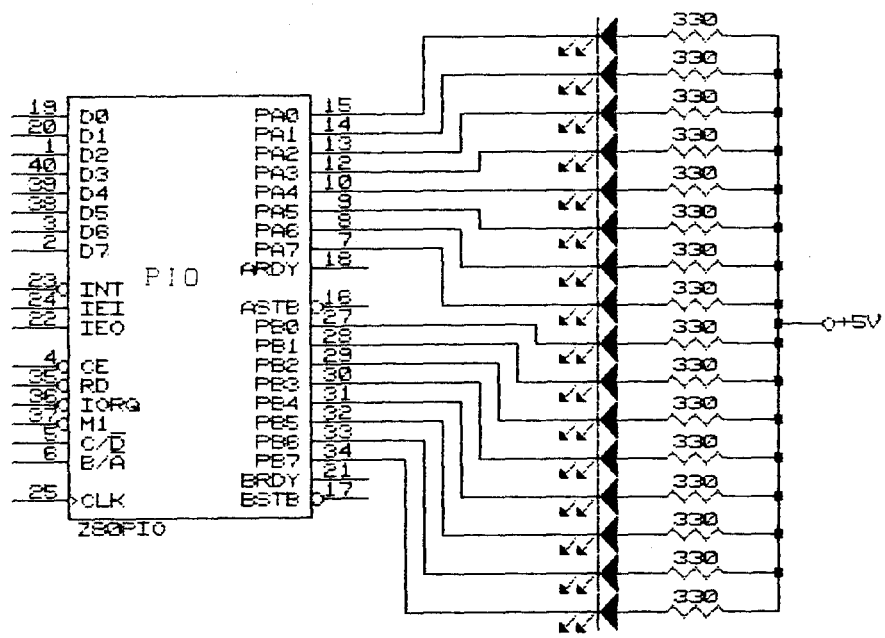
Şekil 4.3 Resetleme Devresi

4.4 Z-80 PIO (Paralel Giriş-Çıkış Birimi)

Giriş-çıkış ünitesi mikroişlemcinin dış dünya ile bilgi alış verişini sağlayan birimdir. PIO' nun Z-80 mikroişlemcisi ile bağlantısı şekil 4.4 te gösterilmiştir. Trafik ışıkları kontrolü için yapılan bağlantı şekil 4.5 te verilmiştir.



Şekil 4.4. Z - 80 PIO Bağlantısı



Şekil 4.5 PIO ile Işıkların Bağlantısı

4.5 Hafıza ve Giriş - Çıkış Üniteleri için Adres Çözücü

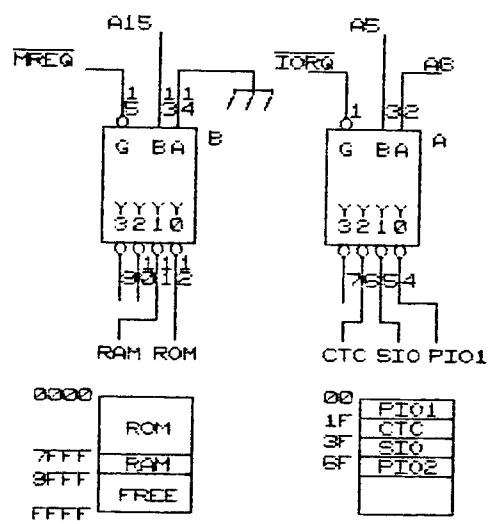
Sistemde 74LS139 adres kod çözücü tümleşik devresi kullanılmıştır. Bu entegrede iki adet 2x4 adres kod çözücü mevcuttur. Bunlardan bir tanesi hafıza üniteleri diğeri ise giriş-çıkış üniteleri için kullanılmaktadır.

4.5.1 Hafızaların adreslenmesi

Hafızalardan hangisinin kullanılacağının belirlenmesi için A_{15} adres hattı kullanılmıştır. A_{15} low ise EPROM, high ise RAM aktif olmaktadır. MEMREQ sinyali kod çözücünün E girişine bağlanarak kod çözücünün yalnızca hafıza işlemlerini yapması sağlanmıştır. Şekil 4.6.a da hafıza adres çözücü bağlantısı gösterilmiştir.

4.5.2 Giriş/Çıkış Ünitelerinin adreslenmesi

Çevre elemanlarının hangisinin kullanılacağını belirlemek için A_5 ve A_6 adres hatları kullanılmıştır. IORQ sinyali kod çözücünün E girişine bağlanarak sadece giriş/çıkış işlemleri sırasında aktif olması sağlanmıştır. Şekil 4.6.b de giriş/çıkış adres çözücü devresi verilmiştir.

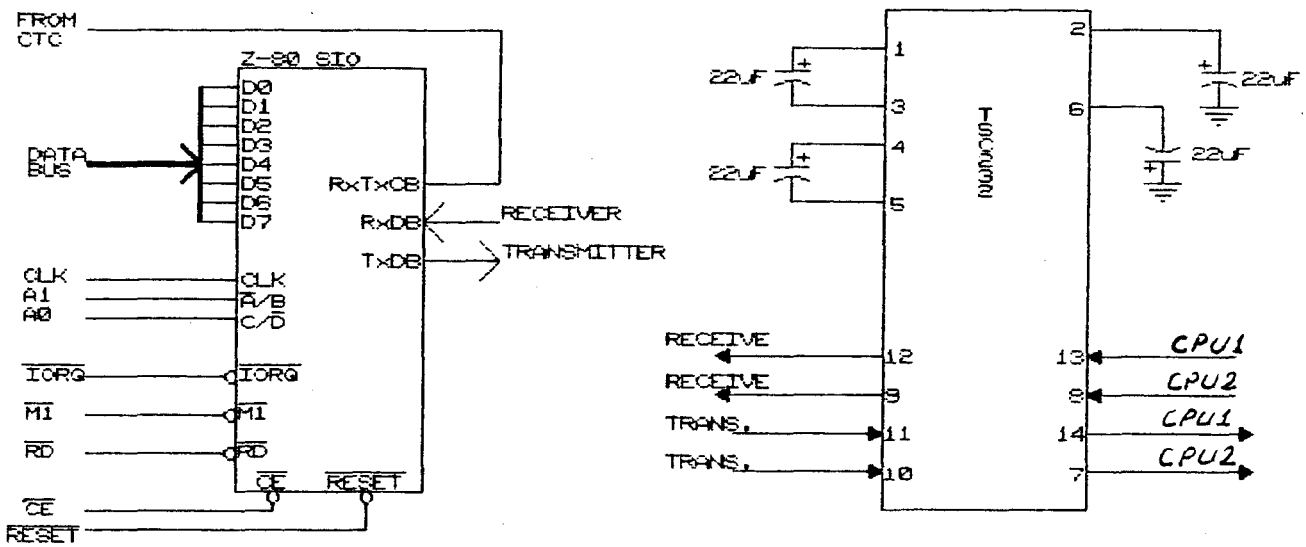


Şekil 4.6 Adres Çözücü ; (a) Hafıza, (b) Giriş/Çıkış

4.6 Z-80 SIO (Seri Giriş-Çıkış Ünitesi)

Seri haberleşme, Z-80 SIO ile yapılmaktadır. Z-80 SIO ile hem senkron hem de asenkron haberleşme yapmak mümkündür. Değişik status reqisterler ile SIO' nun önemli flaglerini ve hata durumlarını gözlemek mümkündür. Bu SIO devresinin mikroişlemci ile bağlantısı Şekil 4.7.a da gösterilmiştir.

Oluşturulan sistemin CPU' dan sonra en önemli ünitesidir. Bilgisayarlar arasındaki iletim SIO aracılığı ile sağlanmıştır. Bilgi iletiminde mesafenin biraz fazla olması nedeniyle SIO çıkışları TSC232 entegresi ile yükseltilmekte ve daha az kayıpla diğer SIO'ya iletilebilmektedir. TSC232 ile SIO arasındaki bağlantı şekil 4.7.b de gösterilmiştir.

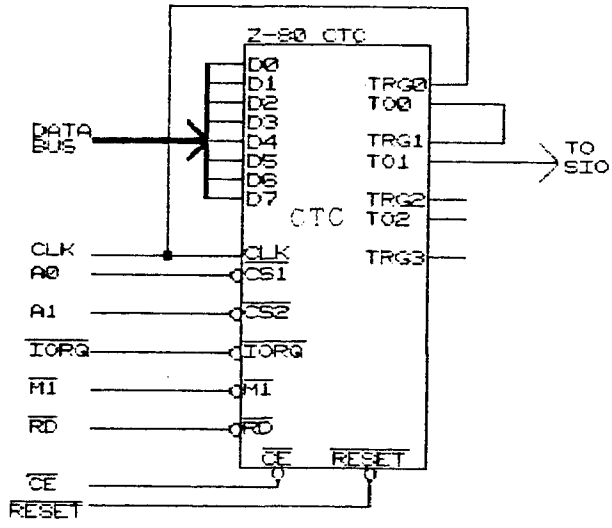


(a) (b)

Şekil 4.7. Z - 80 SIO ve TSC232 Bağlantısı

4.7 Z-80 CTC (Sayıcı, Zamanlayıcı Ünitesi)

Dört kanallı sayıcı, zamanlayıcı devresi bir çok uygulamada ve tasarımda rahatlıkla kullanılmaktadır. Z-80 CTC, Z-80 CPU' ya ve Z-80 SIO' ya hiç bir ara devre gerektirmeden bağlanabilmektedir. Z-80 CTC' nin sistem içerisindeki bağlantısı şekil 4.8 de gösterilmiştir.



Şekil 4.8. Z - 80 CTC Bağlantısı

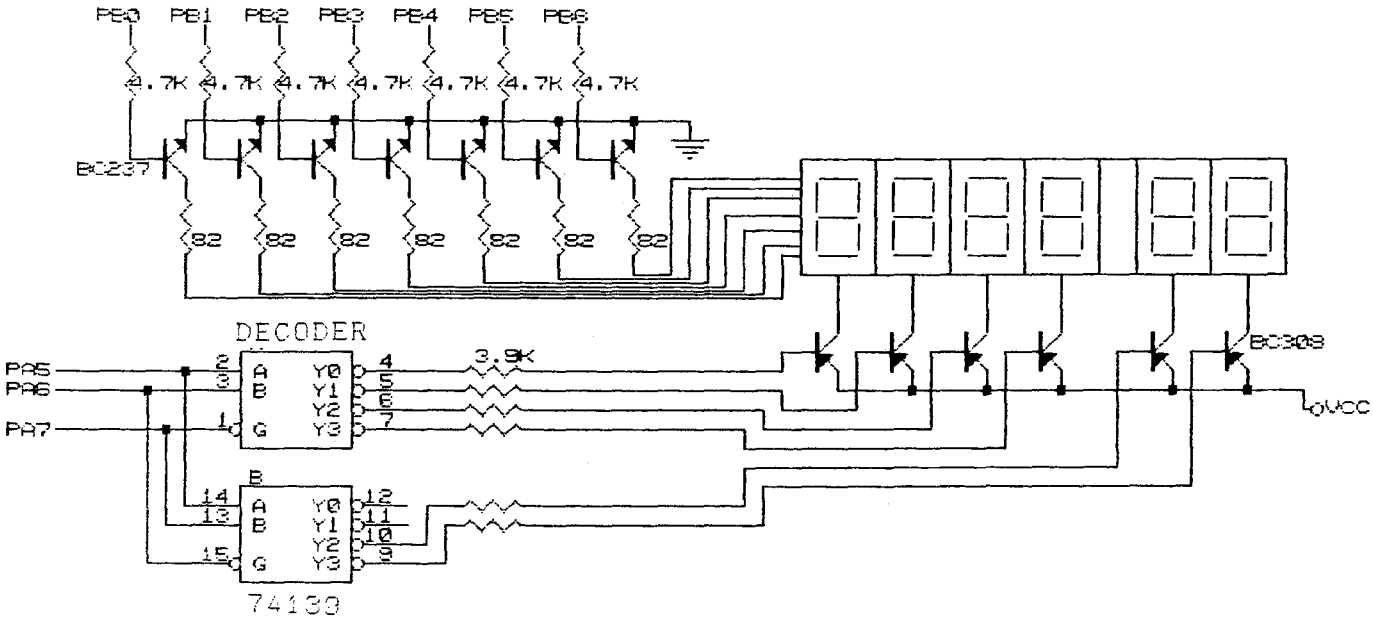
4.8 Gösterge

Gösterge altı adet ortak anodlu yedi parçalı displeylerden oluşmuştur. Displeylerin veri girişleri kendi aralarında paralel bağlanmıştır. Adres çözücü devre ile anod girişlerine transistörlerle besleme verilebilmektedir. Altı tanesinden yalnızca bir tanesi yanabilir. Çok hızlı bir tarama ile altı tanesininde yandığı görülebilir. Displeylerin bağlantısı şekil 4.9 da gösterilmiştir. Gösterge iki kısımdan oluşmuştur. İlk dört displey mikroişlemcinin adres hattı bilgisini, son iki displey ise data hattı bilgisini göstermektedir.

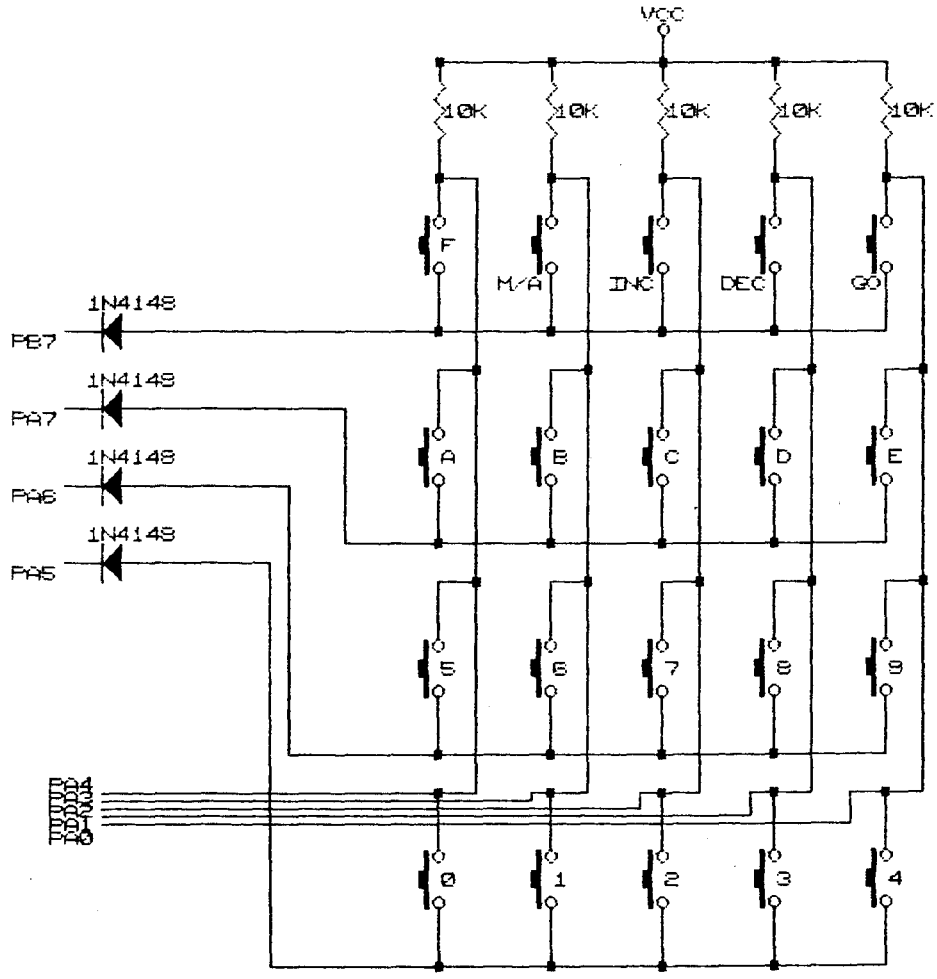
Adres kısmında 8011, data kısmında ise 3E yazılıysa bunun anlamı 8011H adresinde 3EH bulunmaktadır.

4.9 Tuş Takımı

On altı adet sayı tuşu ve dört adet fonksiyon tuşundan oluşan tuş takımı devresi şekil 4.10 da görüldüğü gibi beş satır ve dört sütundan oluşan bir matris formunda tasarlanmıştır. Devreye ilave edilen bu gösterge ve tuş takımı ile PC bilgisayardan bağımsız olarak merkez mikroişlemcisine program yazılıp bağlı ünitelere bilgi gönderilebilir.



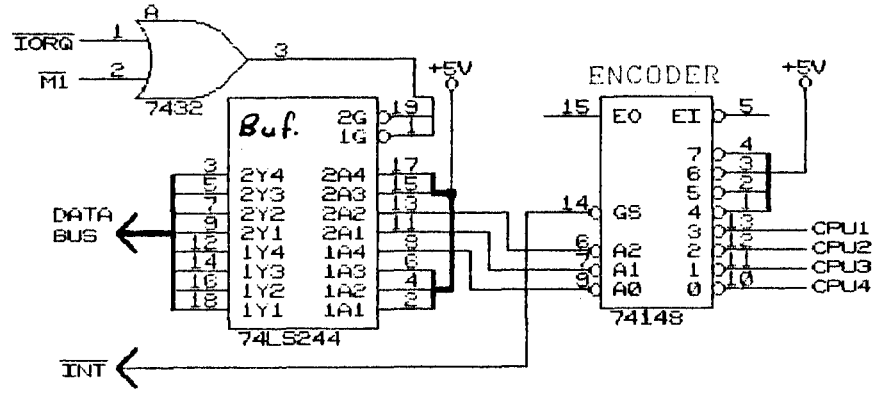
Şekil 4.9 Gösterge Bağlantısı



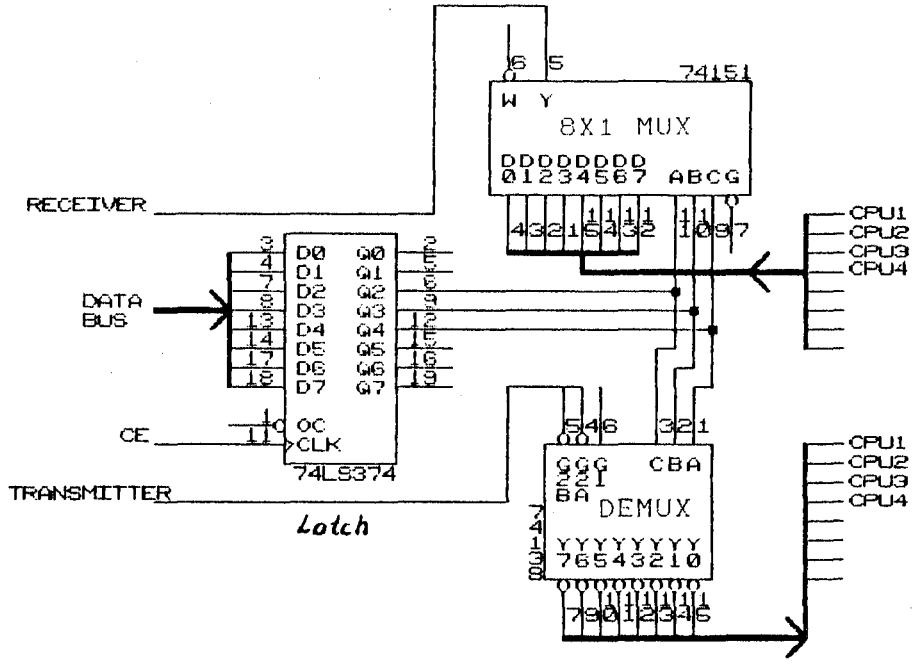
Şekil 4.10. Tuş Takımı

4.10 Interrupt Ara Devresi

Merkez bilgisayar ile yardımcı bilgisayarlar arasındaki haberleşmeyi uyum içinde yapabilmek için bir ara devre tasarlanmıştır. Bu ara devre ile yardımcı üniteler arasında bir öncelik sırası oluşturulmuştur. Aynı zamanda bu devre Z-80 CPU' nun interrupt mode 0 durumu için kullanılan ara devredir. Bu devrenin sisteme bağlanışı şekil 4.11 de gösterilmiştir.



Şekil 4.11. Interrupt ara Devresi



Şekil 4.12 Mux - Demux Bağlantısı

Şekil 4.12 de ise merkez ünite ile yardımcı üniteler arasında seri data transferi için bağlantıyı kurmakta kullanılan MUX ve DEMUX bağlantıları gösterilmiştir. Bu MUX ve DEMUX' ın seçici sinyalleri uygun konumlara ayarlanarak istenen ünite ile haberleşme yapılabilir.

4.11 Sistemin Çalışması

Her ünite açılışta RÖM'da bulunan programı çalıştırır ve bu programa göre kavşaklardaki trafik ışıkları kontrol edilir. Bu Sistem genel olarak üç yardımcı ünite, bir merkezi ünite ve bu merkezi üniteye bağlı bir PC bilgisayardan oluşmuştur. PC bilgisayarında yazılan programda saat sürekli olarak kontrol edilir ve saate göre uygulanacak program numarası belirlenir. Belirlenen programla ilgili bilgiler seri porttan merkezi üniteye gönderilir. Merkezi ünitede kullanılan 'interrupt' ara devresi ile bilginin hangi üniteden geldiği belirlenir ve ilgili alt programa atlanır. Alt programda ilk olarak yapılan belirlenen ünite ile merkezi ünite arasındaki bilgi hattını kontrol eden MUX - DEMUX yapısında uygun 'select'ler gönderilir ve merkezi ünite ile bilgi gönderen ünite arasındaki bilgi hattı birleştirilir. Birleştirilen bilgi hattından gelen veri alınır ve hafızaya yazılır. Merkezi Üniteye PC den gelen bilgi sıra ile diğer yardımcı ünitelere gönderilir. Bu işlem yapılırken her ünite ile merkezi ünite arasındaki bilgi hattı MUX - DEMUX 'ın 'select'leri aracılığı ile uygun konuma getirilir ve veri transfer işlemi yapılır. Yardımcı ünitelerde çalıştırılan programların numaraları üniteler tarafından her faz başlarında merkezi üniteye bildirilir. Merkezi üniteye gönderilen program numaraları merkezi ünitede kontrol edilir ve farklılık varsa çalışması gereken program verileri ilgili üniteye gönderilir. Hemen sonra da merkezi ünitedeki program numaraları bilgisayara gönderilir ve çalışması gereken program olup olmadığı kontrol edilir. Kontrol sonucunda, program numaraları aynı ise tüm üniteler normal çalışmasına devam edecektir, farklı ise işlenmesi gereken

program verileri bilgisayardan merkezi üniteye gönderilir. Merkezi üniteye gelen veriler diğer ünitelere sıra ile gönderilir ve kontrollü ve uygun şekilde tüm ünitelerin çalışması sağlanır (9,10,11,12).

Merkezi üniteden üç üniteye veri gönderme işlemi 'interrupt' ve MUX-DEMUX ara devreleri kullanılarak yapılmaktadır. Bunların yanında sistemde koordine çalışan iki ünite bulunmaktadır. Bu iki ünite ardı ardına gelen iki tane dört yollu kavşağı kontrol etmektedirler. Bu iki ünite ile merkezi ünite arasında bir çevre (loop connection) bağlantısı yapılmıştır. Bu şekilde yapılarak ard arda gelen iki kavşağın koordine çalışması sağlanmış, taşıtların ve yayaların kavşaklardaki beklemeleri önlenmiştir (11,13).

Bütün bunların yanında bağımsız çalışan ünitelerden birisine diğerlerinden farklı olarak bir yaya ışığı kontrol tuşu ilave edilmiştir. Bu tuşun görevi, taşıtlara yeşil ışık yanarken yayaları öncelikli duruma getirmektir. Tuşa basıldığında üniteyi yönlendiren (kontrol eden birim) mikroişlemciye bir NMI (Non Maskable Interrupt) 'interrupt' gitmekte ve bu 'interrupt' isteğinden bir müddet sonra tüm yönden gelen araçlara 3 sn süre ile sarı ışık yanmaktadır. Sarı ışıktan sonra taşıtlara kırmızı, yayalara ise yeşil ışık yanmaktadır. Yayalara belirli bir süre yeşil yandıktan sonra kırmızıya dönüşmektedir. Üç saniye koruma süresinden sonra en son geçiş hakkına sahip olan yöne sar ışık yanmakta ve sonra yeşile dönmektedir. Bundan sonra normal akış devam etmektedir. Kavşaklarda uygulanacak devre süreleri kavşaktaki trafik yoğunluğuna göre değiştirilmektedir. Bu şekilde düzenli ve geçikmelerin çok az olduğu bir trafik akışı sağlanmaktadır. Bu devre süreleri gözlemle yapılan istatistikler sonucunda alınan veriler dikkate alınarak ampirik denklemlerle bilgisayarla hesaplanır (14,15).

5. YAZILIM

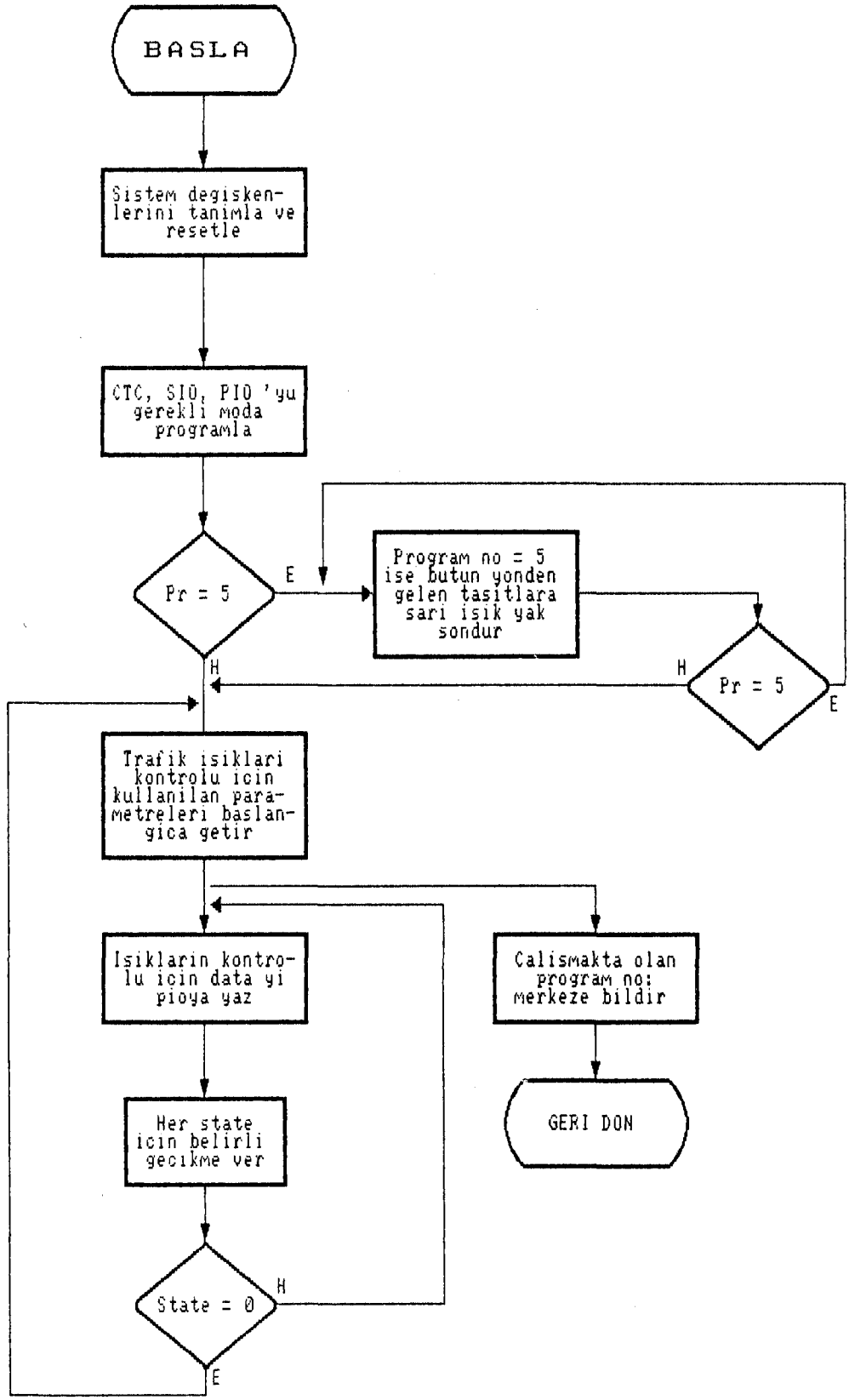
Sistem programları alt programlar halinde yazılmış olup istenen bir programdan diğer bir programa geçmek mümkündür. Alt programlara ulaşmada daha çok interrupt fonksiyonları kullanılmıştır. Bütün sistem PC Bilgisayarında yazılan program ile idare edilir. Yerine getirilecek fonksiyona göre ilgili alt program çalıştırılır ve fonksiyonun gerçekleştirilmesi sağlanır. Her işlemden sonra sistem programına dönülür.

5.1 Akış Şemaları

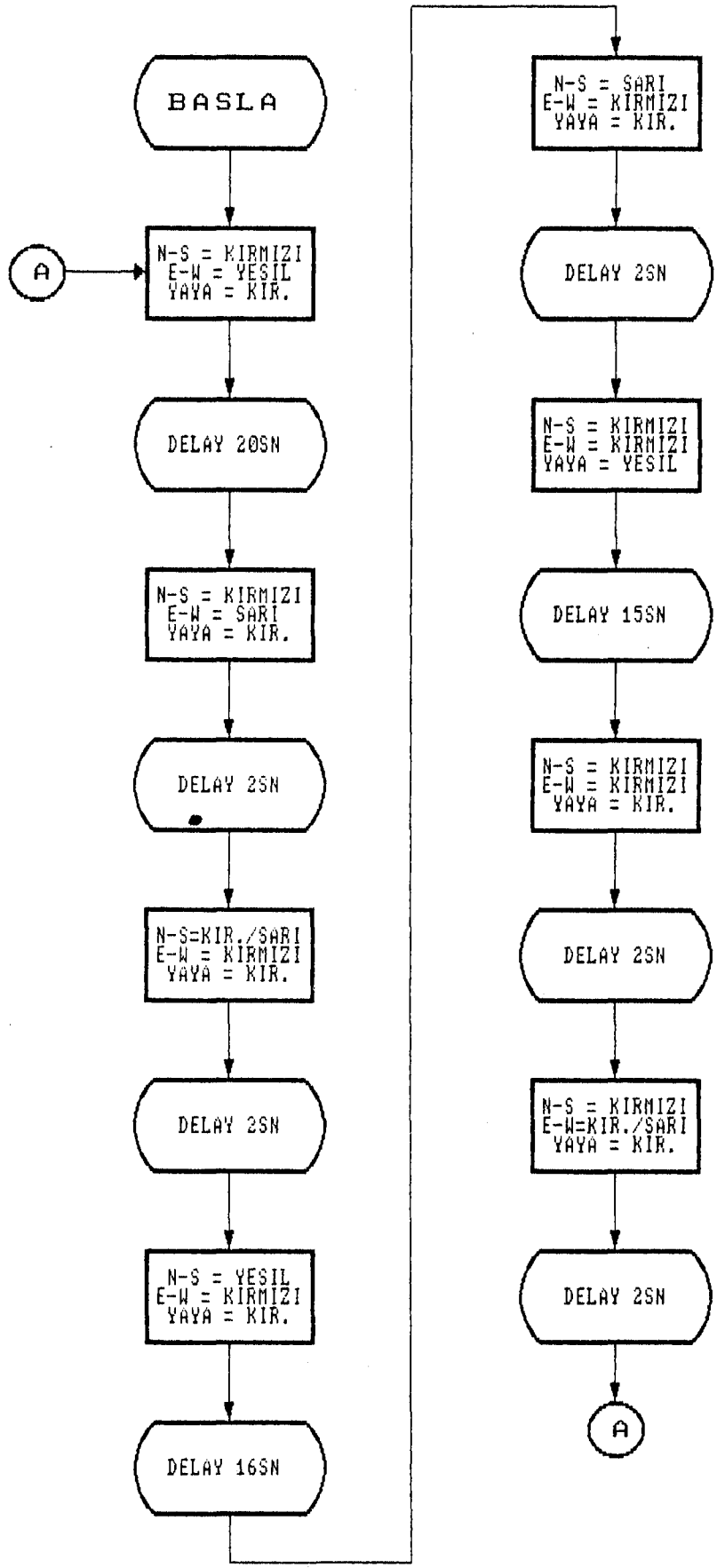
Sisteme enerji verildiğinde her kavşaktaki mikroişlemci merkezden bağımsız olarak ROM' unda bulunan programı çalıştırır ve bu şekilde kavşaktaki ışıkların merkezden bilgi gelinceye kadar kontrolü sağlanmış olur (Şekil 5.1).

Bir kavşakta uygulanan akış diyagramı şekil 5.2 de gösterilmiştir. PC Bilgisayarındaki sistem programı çalışmaya başladığı andan itibaren sistem trafik yoğunluğuna göre düzenli bir şekilde çalışmaya başlar. Bu da sistem programında saatin kontrol edilmesi ve belirlenen saatlere göre uygun programların çalıştırılması ile gerçekleştirilir. Günün değişik saatlerinde trafik yoğunluğu farklı olduğundan kavşaklarda uygulanan devre sürelerinin trafik yoğunluğuna göre düzenlenmesi gerekir. Bunun için belirlenen saatlerde uygulanması gereken devre süreleri bilgisayardan merkezi mikroişlemciye, oradan da kavşaklardaki mikroişlemcilere gönderilir. Bu şekilde de kavşaklardaki trafik akışı trafik yoğunluğuna göre değiştirilir. Akşam, belli bir saatten sonra ve sabah belli bir saate kadar olan sürede ışıkların sarı yanıp sönmesi sağlanır. Başlangıçta saat kontrol edilir ve bunun sonucunda saat 6:00 dan küçükse veya 19:00 dan büyükse bir numaralı program, 6:00 - 7:00 saatleri arasında ise iki numaralı program, 7:00 - 8:00 saatleri arasında ise üç numaralı program, 8:00 - 9:00 saatleri arasında ise dört numaralı program, 9:00 - 12:00 saatleri arasında ise beş numaralı program, 12:00 - 14:00 saatleri.

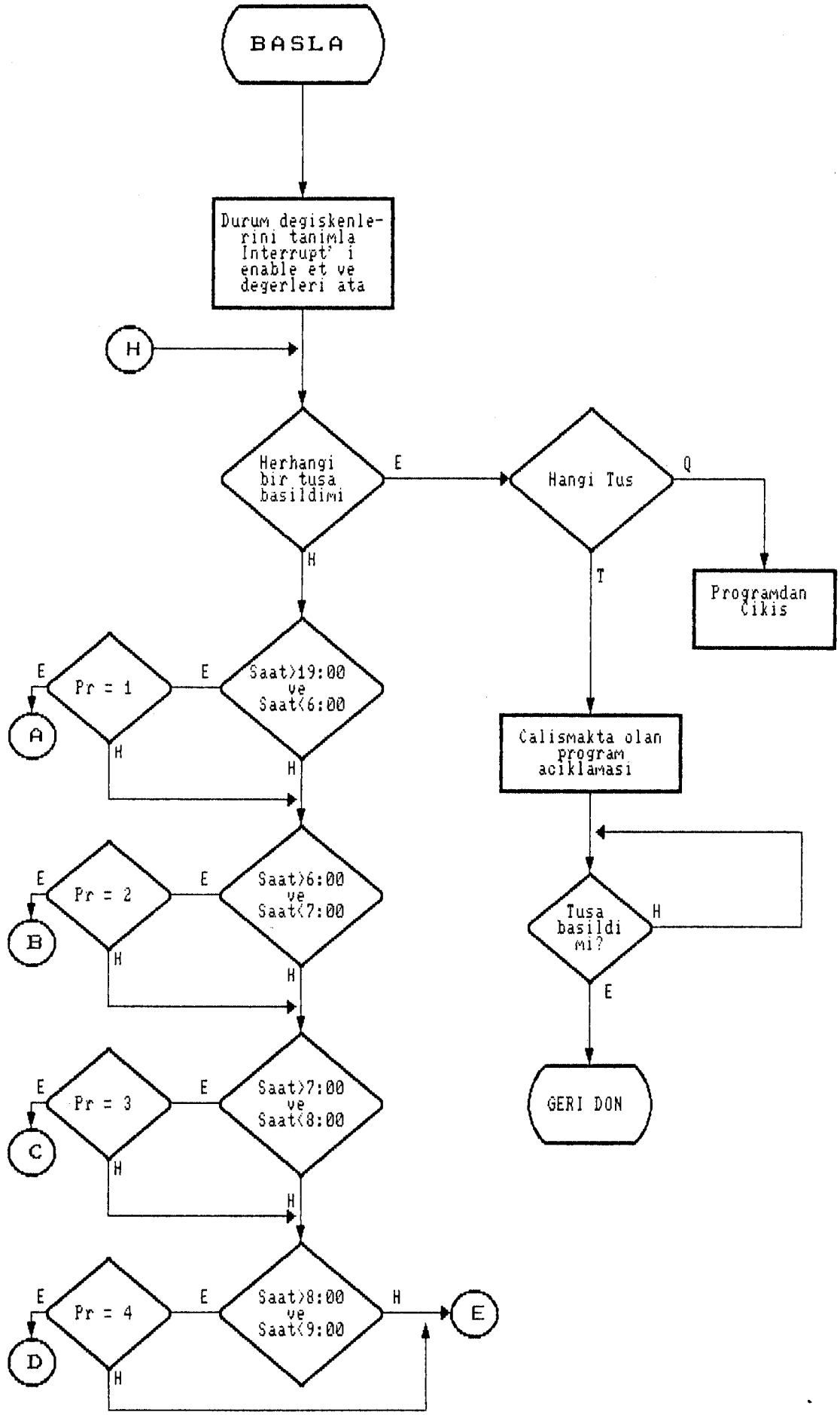
arasında ise altı numaralı program, 14:00 - 17:00 saatleri arasında ise beş numaralı program ve 17:00 - 19:00 saatleri arasında ise altı numaralı program çalıştırılacaktır. Sistem programının akış şeması Şekil 5.3 de, merkezi mikroişlemcideki yardımcı sistem programı akış şeması Şekil 5.4 da gösterilmiştir. Sistem programları Ek 5' te verilmiştir.



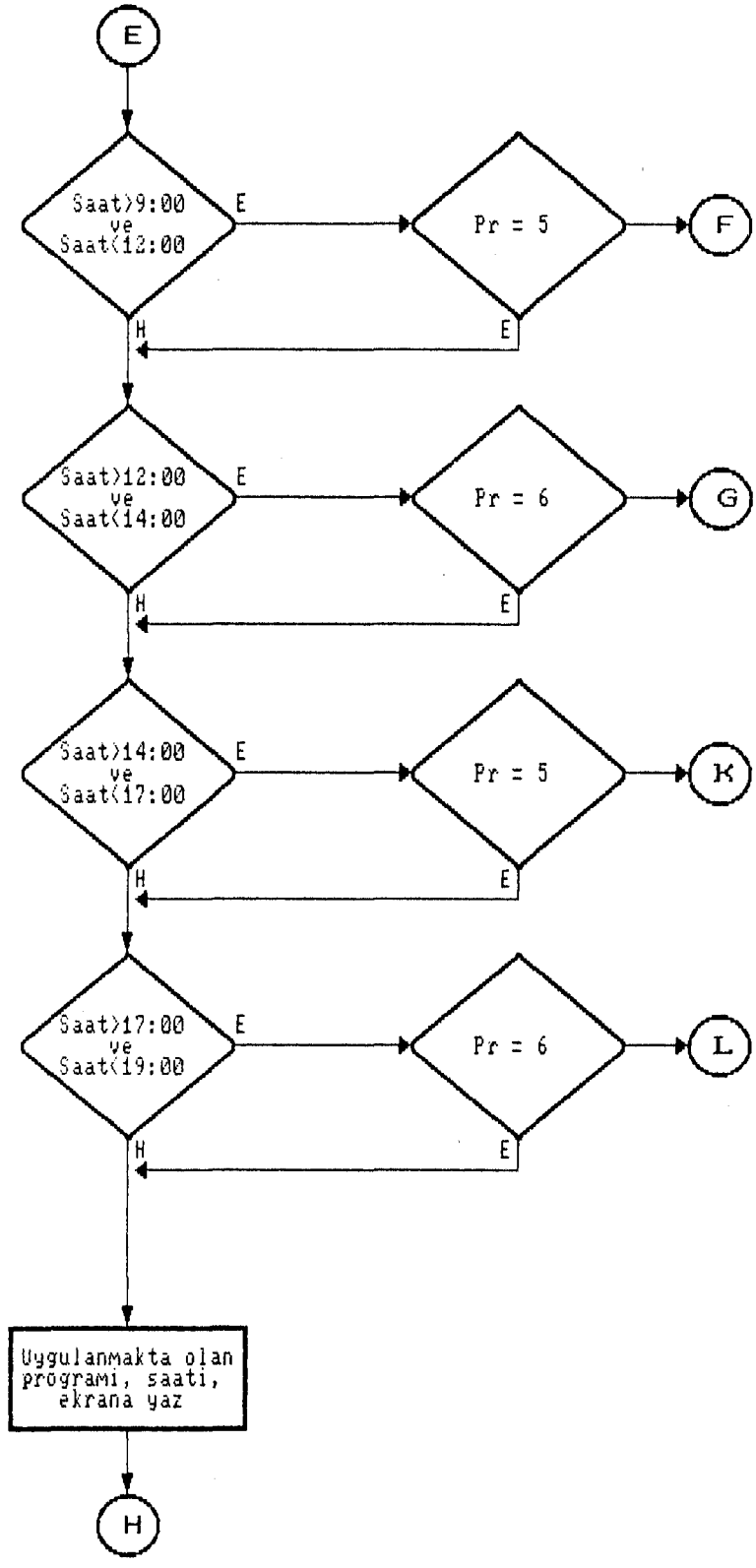
Şekil 5.1. Kavşaktaki Işık Kontrolü İçin Akış Şeması



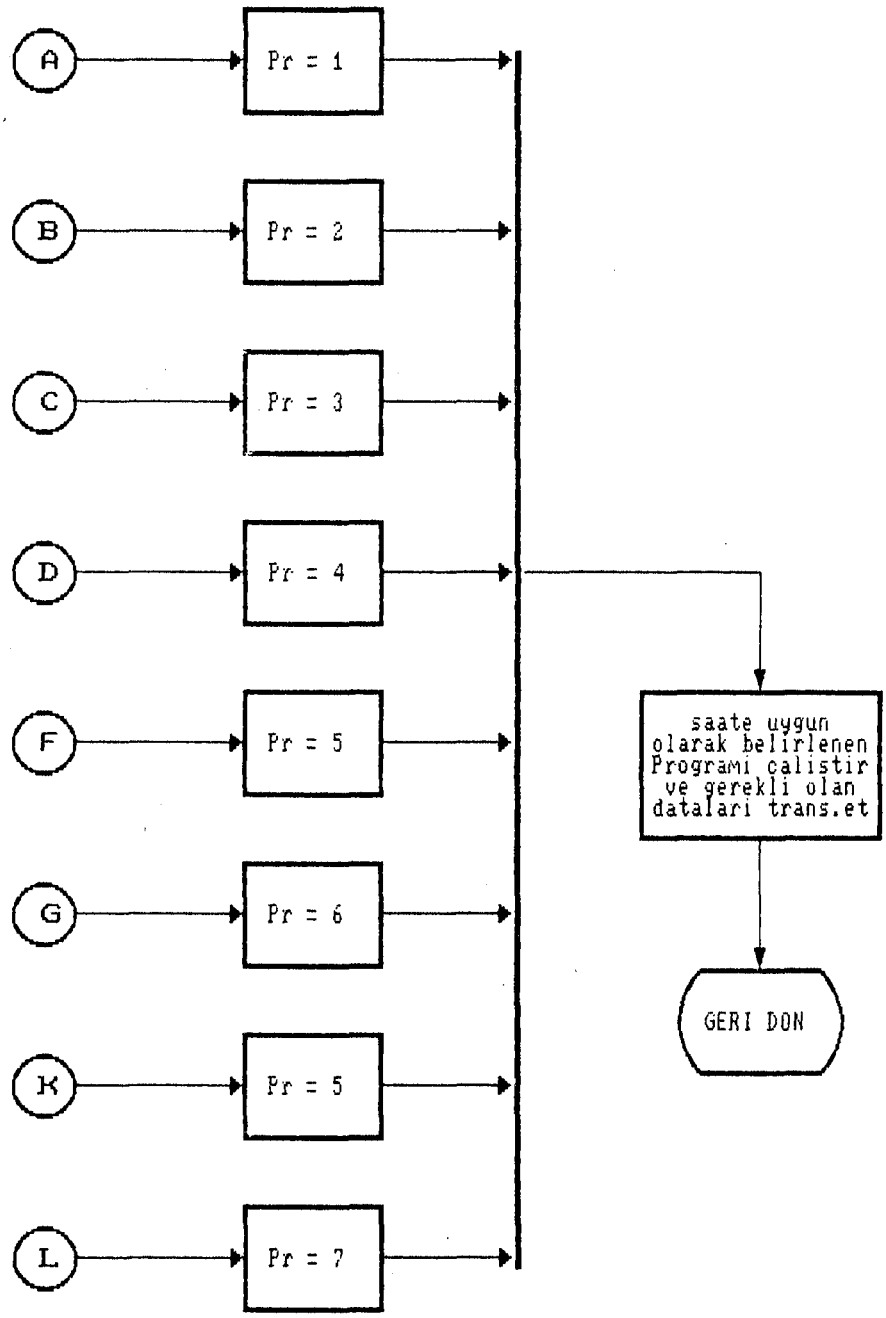
Şekil 5.2. Sinyalize Bir Tesisin Çalışmasının Akış Şeması



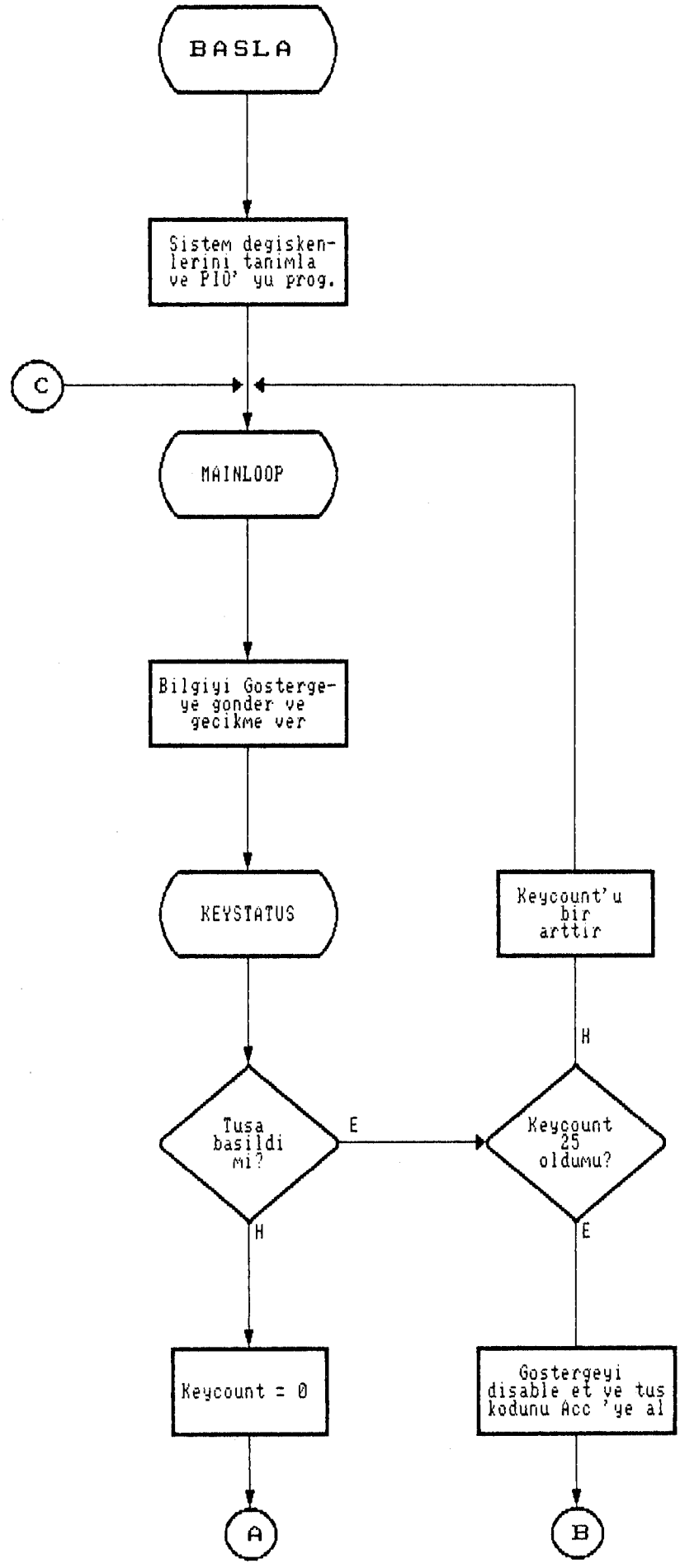
Şekil 5.3. Sistem Programı Akış Şeması



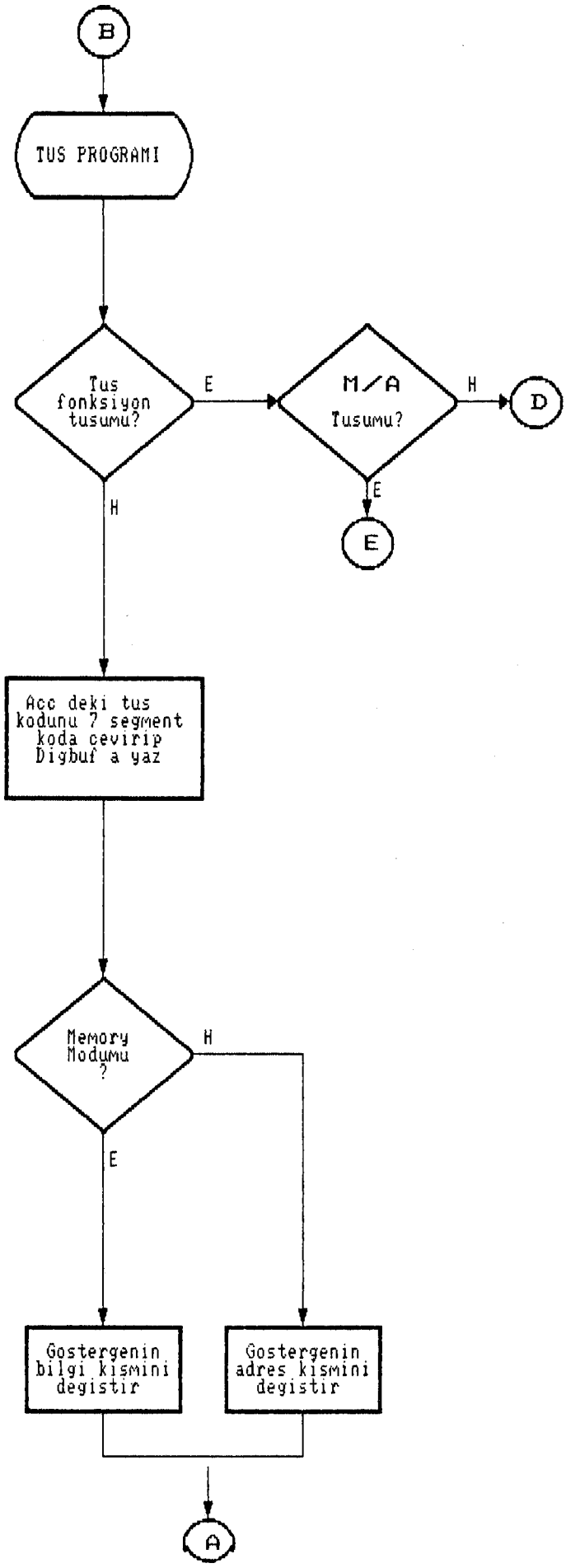
Şekil 5.3. Sistem Programı Akış Şeması (devam)

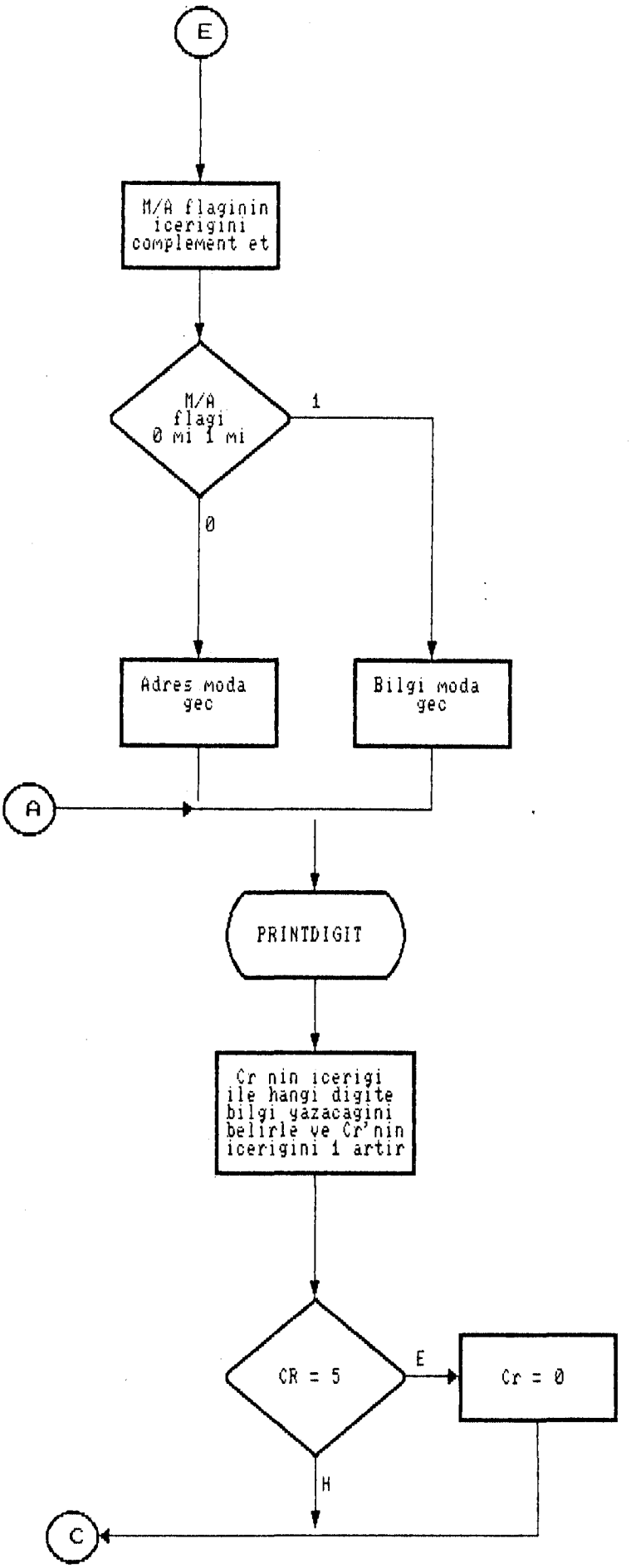


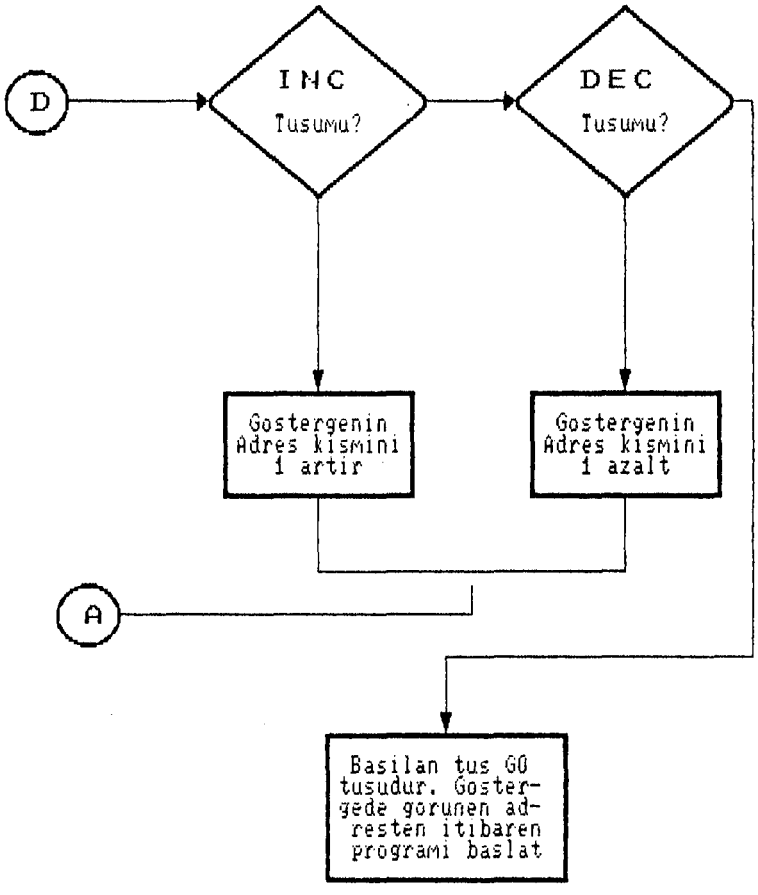
Şekil 5.3. Sistem Programı Akış Şeması (devam)



Şekil 5.4. Merkez Mikroişlemci Sistem Programı Akış Şeması


B
TUS PROGRAMI
Tus fonksiyon tusumu?
E
M/A Tusumu?
H
D
E
E
H
Acc deki tus kodunu 7 segment koda cevirip Digbuf a yaz
Memory Modumu ?
H
E
Gostergenin bilgi kismini degistir
Gostergenin adres kismini degistir
A


E
M/A flaginin icerigini complement et
M/A flagi 0 mi 1 mi
1
0
Adres moda gec
Bilgi moda gec
A
PRINTDIGIT
Cr nin icerigi ile hangi digite bilgi yazacagini belirle ve Cr'nin icerigini 1 artir
CR = 5
E
Cr = 0
H
C



Şekil 5.4. Merkez Mikroişlemci Sistem Programı Akış Şeması (devam)

6. SONUÇLAR ve ÖNERİLER

Bu çalışmada belirlenen bazı kavşaklardaki trafik ışıklarının bir merkezden mikroişlemciler yardımı ile kontrol edilmesi gerçekleştirilmiştir. Merkezi üniteden kavşaklardaki ünitelere bilgi iletimi seri hattan (seri data transferi metodu ile) yapılmaktadır. Hazırlanan yazılım ile de günün değişen trafik yoğunluğuna göre devre süreleri belirlenmiş ve buna göre ışıkların kontrolü yapılmıştır.

Trafik ışıklarının bir merkezden ve mikroişlemcilerle kontrolü ile kavşaklarda bekleme süreleri minimuma indirilecek, uzun kuyrukların oluşmasına izin verilmeyecektir. En önemlisi de kazaların ve beklemelerde boşa harcanan yakıtın asgari bir düzeye düşmesi sağlanacaktır.

Böyle bir çalışma daha ileri bir düzeye getirilip bütün bir şehrin trafik ışıkları kontrolü yapılabilir. Bunu gerçekleştirebilmek için de üniteler arasındaki veri transferi döşenen özel hatlar ile sağlanabilir. Bu hatlarda kullanılacak kaplolar düşük empedanslı ve gürültüsüz olarak seçilmelidir. Çünkü bu hatlarda ki gerilim düşümü bu tip kaplolarla minimum seviyeye düşürülebilir. Çok gelişmiş bir sistem kurmak istersek, kavşaklara trafik ışık akışını kontrol eden mikroişlemci tarafından kontrol edilebilen taşıt sayıcı sensörler yerleştirilmelidir. Bu sensörler ile kavşaklardan geçen araç sayısı hakkında sürekli istatiksel bilgi alınır. Bu bilgiler doğrultusunda sistem kendi parametrelerini hesaplayabilir. Parametreler tarafından bölüm 2' de verilen ampirik denklemlere uygulanır ve bunun sonucunda trafik akışına göre sürekli değişen Adapdif kontrol sistemi kurulmuş olur.

KAYNAKLAR DIZINI

1. Kutlu, K., 1964, Trafik etüdleri, Matbaa teknisyenleri basımevi, 138s.
2. Kutlu, K., 1967, Trafik tekniği, Teknik üniversite matbaası, 372s.
3. Ayfer, M.D., 1977, Trafik sinyalizasyonu, Karayolları genel müdürlüğü matbaası, 184s.
4. Hobbs, F.D., 1979, Traffic planning and engineering, Pergamon press, 544p.
5. Tanenbaum, A.S., 1981, Computer Networks, Prentice-Hall, Inc., 518p.
6. Gupta, A., 1987, Multi-Microprocessors, IEEE press, 268p,
7. Laver, M., 1975, Computers, Communications and Society, Oxford University press, 100p.
8. Uffenbeck, J., 1985, Microcomputers and microprocessors the 8080, 8085 and Z-80, Prentice-Hall, Inc., 670p
9. Zaks, R., 1982, Programming the Z-80, Sybex, Inc., 626p.
10. Greenfield, J.D., 1985, Microprocessor Handbook, John Wiley & Sons, Inc., 636p.
11. Ward, D.E., 1990, The American double ring system applied in London, London, viii+218p.
12. Antonini, C., 1981, Microcomputer programmed traffic light controls, Italy, 143 - 62 p.
13. Strawinski, T., 1983, Microcomputers in road traffic systems, Budapest, 126 - 137 p.
14. Pursula, M., 1989, Microprocessor and PC-based vehicle classification equipments using induction loops, UK, vii+199p.
15. Hawke, M.J., 1984, Application of microprocessor traffic controllers-adding pedestrian signals to existing signalled junctions, London, 60p.

EKLER

Chapter 7
ZILOG Z80

g Z80 microcomputer devices have been designed as 8080A enhancements. In fact, the same individuals ponsible for designing the 8080A CPU at Intel designed the Z80 devices at Zilog. The 8085, described in pter 5, is Intel's 8080A enhancement.

Z80 instruction set includes all 8080A instructions as a subset. In deference to rational necessity, vever, neither the Z80 CPU, nor any of its support devices attempt to maintain pin-for-pin compatibility with 0A counterparts. Compatibility is limited to instruction sets and general functional capabilities. A program has been written to drive an 8080A microcomputer system will also drive the Z80 system — within cer- limits; for example, a ROM device that has been created to implement object programs for an 8080A rocomputer system can be physically removed and used in a Z80 system.

Z80-8080A compatibility does extend somewhat further, since most support devices that have been gned for the 8080A CPU will also work with a Z80 CPU; therefore in many cases you will be able to upgrade 080A microcomputer system to a Z80, confining hardware modifications to the CPU and its immediate in- ace only.

interesting to note that the Z80 pins and signal interface is far closer than the 8085 to the three-chip 8080A figuration illustrated in 8080A chapter. Also, whereas the Z80 instruction set is greatly expanded as compared to 8080A, the 8085 instruction set contains just two new instructions. However, both the Z80 and the 8085 have lved the two most distressing problems associated with the 8080A — the three-chip 8080A CPU has in both cases reduced to one chip, and the three 8080A power supplies have in both cases been reduced to a single +5V power ply.

G, INC., manufacturers of the Z80, are located at:

10460 Bubb Road
Cupertino, California 95014

official second source for Zilog products is:

MOSTEK, INC.
1215 West Crosby Road
Carrollton, Texas 75006

annel MOS technology is used for all Z80 devices.

Z80 LSI TECHNOLOGY

THE Z80 CPU

tions implemented on the Z80 CPU are illustrated in Figure 7-1. They represent "typical" CPU logic, valent to the three devices: 8080A CPU, 8224 Clock and 8228 System Controller.

UMMARY OF Z80/8080A DIFFERENCES

are going to summarize Z80/8080A differences before describing differences in detail. If you know the)A well, read on; if you do not, come back to this summary after reading the rest of the Z80 CPU descrip- We will also contrast the Z80 and the 8085, where relevant.

he programmer, the Z80 provides more registers and addressing modes than the 8080A, plus a much larger uction set.

ificant hardware features are a single power supply (+5V), a single system clock signal, an additional inter- and logic to refresh dynamic memories.

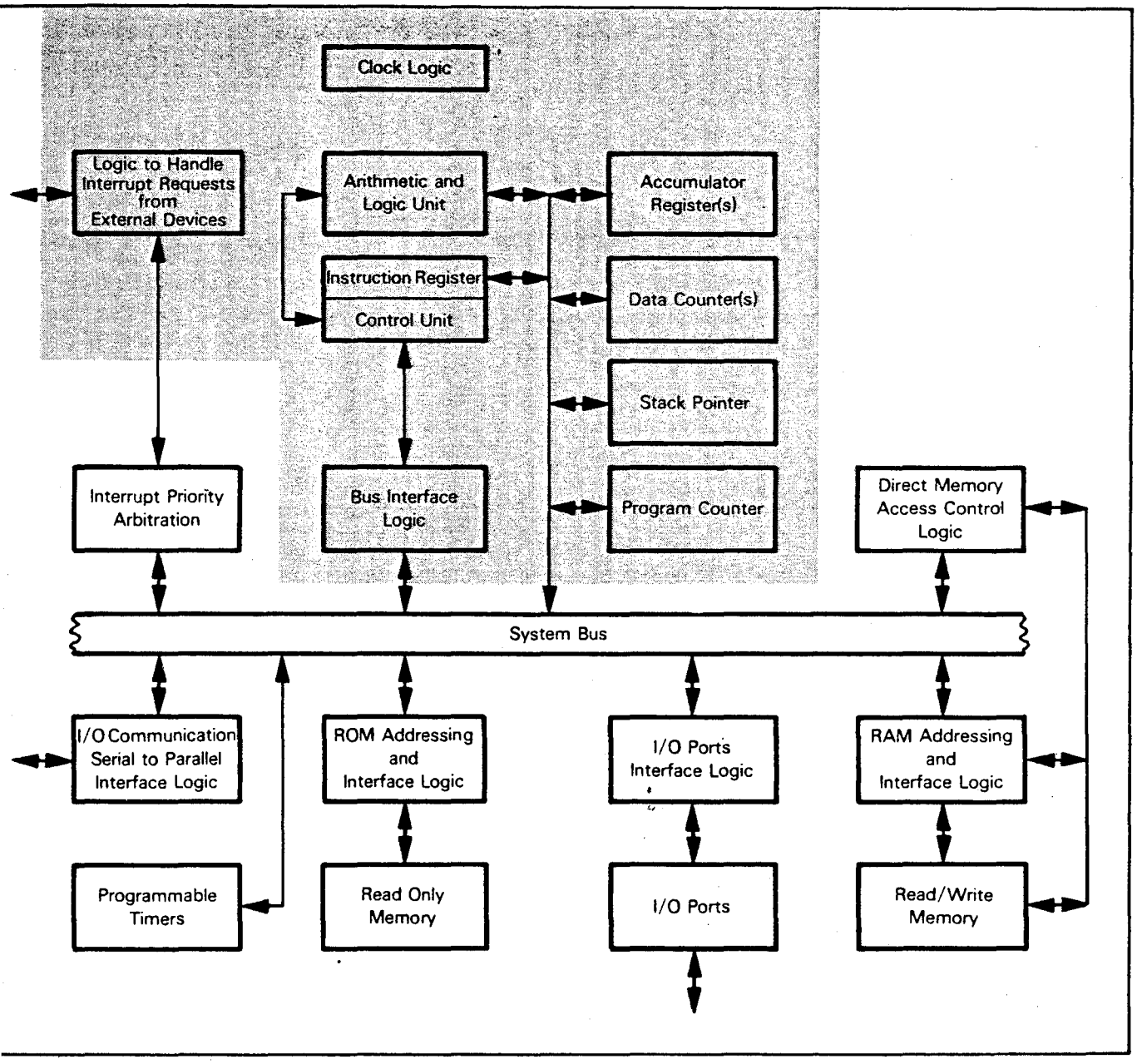


Figure 7-1. Logic Functions of the Z80 CPU

e 8085 also has a single power supply and a single system clock signal. The 8085 has three additional interrupts, but ks logic to refresh dynamic memories.

the Z80 CPU indeed the logical next 8080A evolution?

rdware aspects of the 8080A represent its weakest features, as compared to principal current competitors. ecifically, the fact that the 8080A is really a three-chip CPU is its biggest single problem; three chips are always ng to cost more than one. Next, the fact that the 8080A requires three power supplies (+5V, -5V and +12V) is a very gative feature for many users and the desirability of going to a single power supply is self-evident; the Z80 requires a gle +5V power supply. This is also true of the 8085.

e problems associated with condensing logic from three chips onto one chip are not so straightforward. Figure 7-2 il- trates the standard three-chip 8080A CPU. Let us assume that the three devices are to be condensed into a single p. Asterisks (*) have been placed by the signals which must be maintained if the single chip is to be hardware com- ible with the three chips it replaces. Forty-three signals are asterisked, therefore the standard 40-pin DIP cannot be d. The problem is compounded by the fact that not all 8080A systems use an 8228 System Controller. Some 8080A tems use an 8212 bidirectional I/O port to create control signals. A few of the earliest 8080 systems use neither the 28 System Controller, nor an 8212 I/O port; rather external logic decodes the Data Bus when SYNC is true in order to nerate control signals; for example, that is how the TMS5501 works. We must therefore conclude that any attempt

to reduce three chips to one will create a product that is not pin compatible with the 8080A; and, indeed, the Z80 is not pin compatible. What Zilog has done is include as many hardware enhancements as possible within the confines of a 40-pin DIP that must be philosophically similar to the 8080A, without attempting any form of pin compatibility. Figure 7-2 identifies the correlation between Z80 signals and 8080A signals. Notice that there is a significant similarity.

Figure 5-3 is equivalent to Figure 7-2, comparing 8085 and 8080A signals. Z80 signals are far closer to the 8080A three-chip set than the 8085.

Here is a summary of the hardware differences:

1) The Z80 has reduced three power supplies to a single +5V power supply.

2) Clock logic is entirely within the Z80.

3) The complex, two clock signals of the 8080A have been replaced by a single clock signal.

4) Automatic dynamic memory refresh logic has been included within the CPU.

5) Read and write control signal philosophy has changed. The 8080A uses separate memory read, memory write, I/O read and I/O write signals. The Z80 uses a general read and a general write, coupled with a memory select and an I/O select. This means that if a Z80 CPU is to replace an 8080A CPU then additional logic will be required beyond the Z80 CPU. You will either have to combine the four Z80 control signals to generate 8080A equivalents, or you will have to change the select and strobe logic for every I/O device. We will discuss this in more detail later.

6) Address and Data Bus float timing associated with DMA operations have changed. The 8080A floats these busses at the beginning of the third or fourth time period within the machine cycle during which a bus request occurs; this initiates a Hold state. The Z80 has a more straightforward scheme; a Bus Request input signal causes the Data and Address Busses to float at the beginning of the machine cycle; floating busses are acknowledged with a Bus Acknowledge output signal.

7) The Z80 has an additional interrupt request. In addition to the RESET and normal 8080A interrupt request, the Z80 has a nonmaskable interrupt which is typically used to execute a short program that prepares for power failure, once a power failure has been detected.

Now consider internal organization of the Z80 in terms of instruction set compatibility and enhancement.

As illustrated by Table 7-3 the 8080A instruction set is, indeed, a subset of the Z80 instruction set. Unfortunately, the Z80 uses completely new source program instruction mnemonics, therefore 8080A instructions cannot immediately be identified. Technical Design Labs, Inc., has an 8080-like Z80 assembly language.

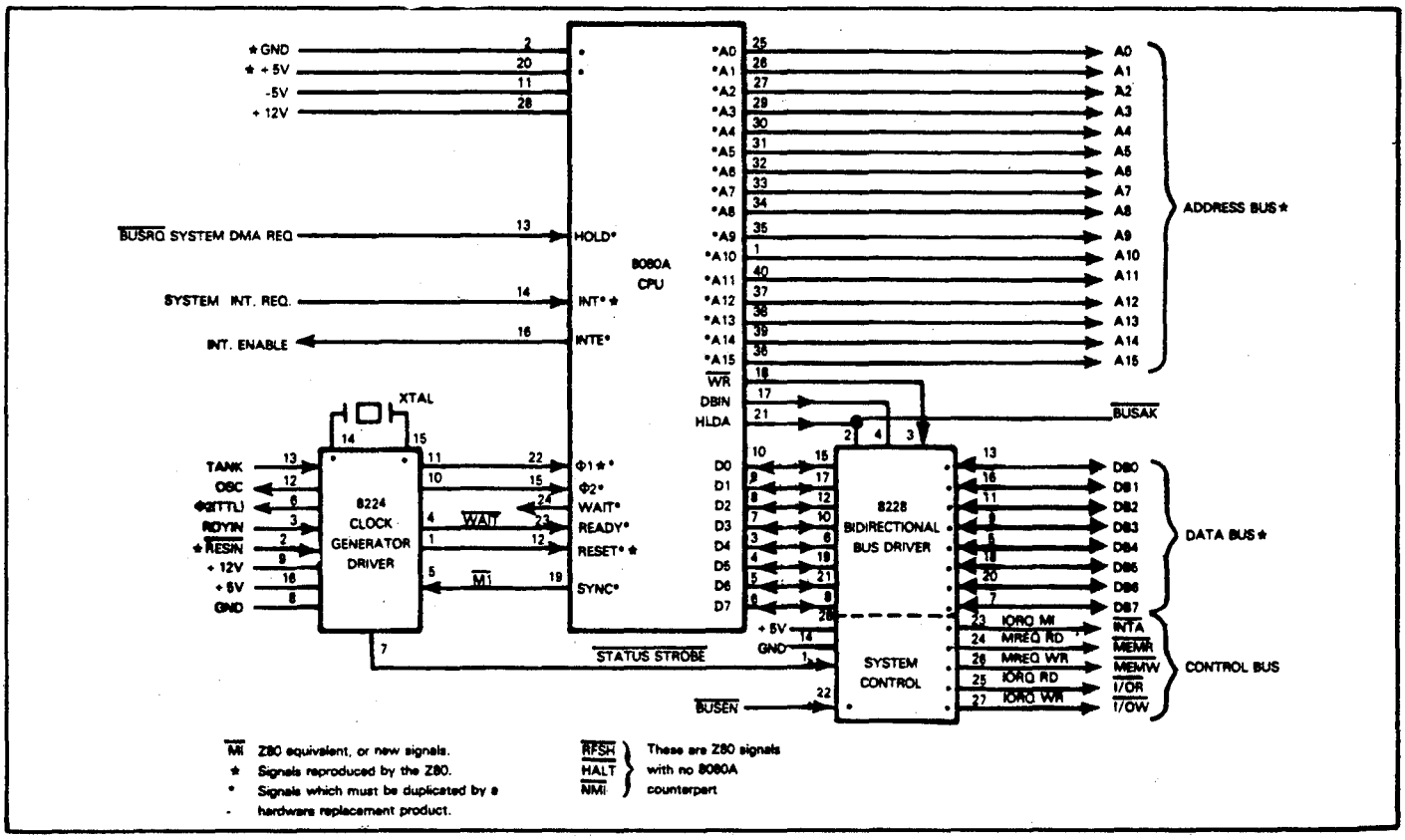


Figure 7-2. The Standard 8080A Three-Chip System and Z80 Signal Equivalents

There are very few unused object codes in the 8080A instruction set. The Z80 has therefore taken what few unused object codes there are, and used them to specify that an additional byte of object code follows:

11011101 ◄— Spare 8080A object code
◄— Specifies new Z80 object code follows

This results in most new Z80 instructions having 16-bit object codes; but simultaneously it means that a very large number of new instructions can be added.

Any enhancement of the 8080A can include major changes within the CPU; providing the 8080A registers and status flags remain as a subset of the new design, instruction compatibility remains. These are the principal enhancements made by the Z80:

1) The standard general purpose registers and status flags have been duplicated. This makes it very easy to handle single-level interrupts, since general purpose register and Accumulator contents no longer need to be saved on the Stack; instead, the program may simply switch to the alternate register set.

2) Two Index registers have been added. This means that additional Z80 instructions can use indexed memory addressing.

2) An Interrupt Vector register allows external logic the option of responding to an interrupt acknowledge by issuing the equivalent of a Call instruction — which vectors program execution to a memory address which is dedicated to the acknowledged external logic.

4) A single Block Move instruction allows the contents of any number of contiguous memory bytes to be moved from one area of memory to another, or between an area of memory and a single I/O port. You can also scan a block of memory for a defined value by executing a Block Compare instruction.

5) Instructions have been added to test or alter the condition of individual register and memory bits.

In contrast to the extensive enhancements of the Z80, the 8085 registers and status architecture are identical to the 8080A. There are only two additional instructions in the 8085 instruction set; however, the 8085, like the Z80, allows Call instructions to be used when acknowledging an interrupt — a particularly useful enhancement.

While on the surface the Z80 instruction set appears to be very powerful, note that instruction sets are very subjective; right and wrong, good and bad are not easily defined. Let us look at some nonobvious features of the Z80 instruction set.

First of all, the execution speed advantage that results from the new Z80 instructions is reduced by the fact that many of these instructions require two bytes of object code. Some examples of Z80 instructions and equivalent 8080A instruction sequences with equivalent cycle times are given in Table 7-1.

Table 7-1. Comparisons of Z80 and 8080A Instruction Execution Cycles

Z80			8080A		
Instructions		Cycles	Instructions		Cycles
LD	R,(IX + d)	19	LXI	H,d	10
			DAD	IX	10
			MOV	R,M	7
					27
LD	RP,ADDR	20	LHLD	ADDR	16
			MOV	C,L	5
			MOV	B,H	5
					26
SET	B,(HL)	15	MOV	A,M	7
			ORI	MASK	7
			MOV	M,A	7
					21

Also, a novice programmer may find the Z80 instruction set bewilderingly complex. At a time when the majority of potential microcomputer users are terrified by simple assembly language instruction sets, it is possible that users will react negatively to an instruction set whose complexity (if not power) rivals that of many large minicomputers.

Many of the new Z80 instructions use direct, indexed memory addressing to perform operations which are otherwise identical to existing 8080A instructions. Now the Z80 has two new 16-bit Index registers whose contents are added to

an 8-bit displacement provided by the instruction code; this is the scheme adopted by the Motorola MC6800. This scheme is inherently weaker than having a 16-bit, instruction-provided displacement, as implemented by the Signetics 2650. When the Index register is larger than the displacement, the Index register, in effect, becomes a base register. When the Index register has the same size, or is smaller than the displacement, it is truly an Index register as described in "Volume 1 — Basic Concepts". The Signetics 2650 implementation is more powerful.

Z80 PROGRAMMABLE REGISTERS

We will now start looking at the Z80 CPU in detail, beginning with its programmable registers.

The Z80 has two sets of 8-bit programmable registers, and two Program Status Words. At any time one set of programmable registers and one Program Status Word will be active and accessible.

In addition, the Z80 has a 16-bit Program Counter, a 16-bit Stack Pointer, two 16-bit Index registers, an 8-bit Interrupt Vector and an 8-bit Memory Refresh register.

Figure 7-3 illustrates the Z80 registers. Within this figure, the 8080A registers' subset is shaded.

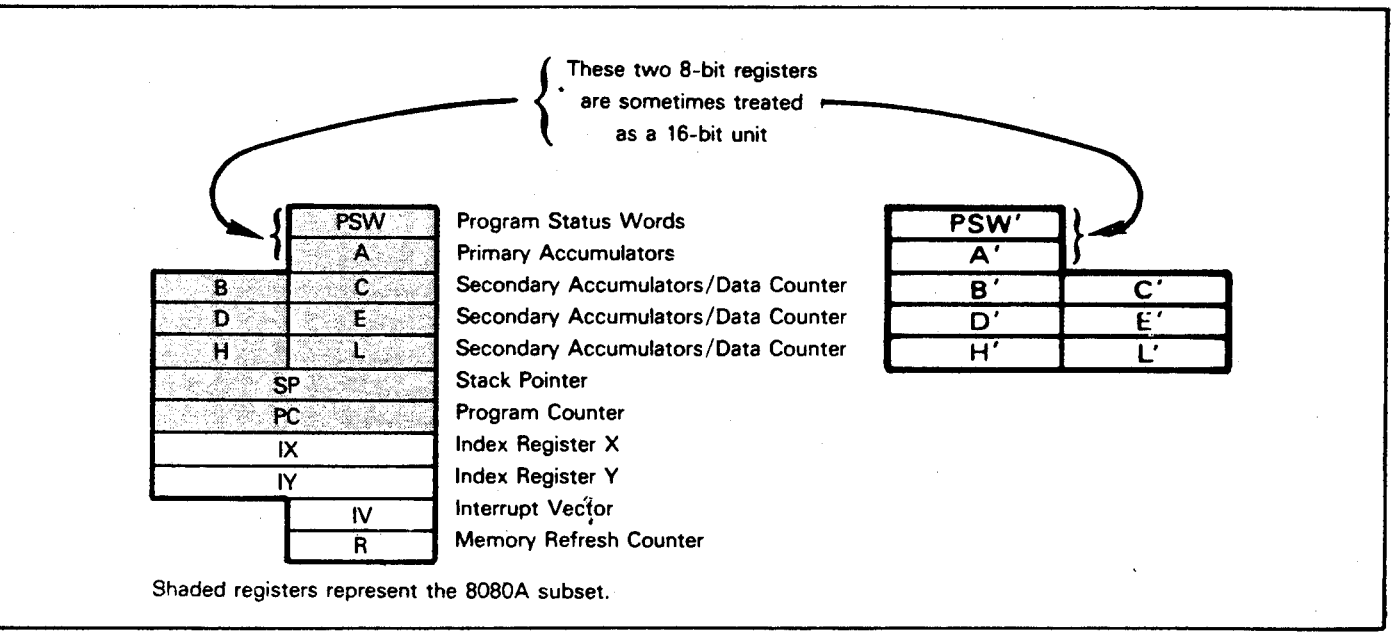


Figure 7-3. Z80 Programmable Registers

The Z80 uses its Program Status Word, its A, B, C, D, E, H, and L registers, plus the Stack Pointer and the Program Counter exactly as the 8080A uses these locations; therefore no additional discussion of these registers is needed.

The Program Status Word, plus registers A, B, C, D, E, H and L are duplicated. Single Z80 instructions allow you to switch access from one register set to another, or to exchange the contents of selected registers. At any time, one or the other set of registers, but not both, is accessible.

There are two 16-bit Index registers, marked IX and IY. These are more accurately looked upon as base registers, as will become apparent when we examine Z80 addressing modes.

The Interrupt Vector register performs a function similar to the ICW2 byte of the 8259 PICU device (described in the 8080A chapter). Z80 interrupt acknowledge logic gives you the option of initiating an interrupt service routine with a Call instruction, where the high order address byte for the call is provided by the Interrupt Vector register. The 8085 also provides this capability.

The Memory Refresh Counter register represents a feature of microcomputer systems which has been overlooked by everyone except Fairchild and Zilog. Dynamic memory devices will not hold their contents for very long, irrespective of whether power is off or on. A dynamic memory must therefore be accessed at millisecond intervals. Dynamic memory devices compensate for this short-coming by being very cheap — and dynamic refresh circuitry is very simple. Using a technique akin to direct memory access, dynamic refresh circuitry will periodically access dynamic memories, rewriting the contents of individual memory words on each access. About the only logic needed by dynamic refresh is a counter via which it keeps track of its progress through the dynamic memory; that is the purpose of the Z80 Memory Refresh Counter register. The Z80 also has a special DMA refresh control signal; therefore the Z80 provides much of the dynamic refresh logic needed by dynamic memory devices.

Z80 ADDRESSING MODES

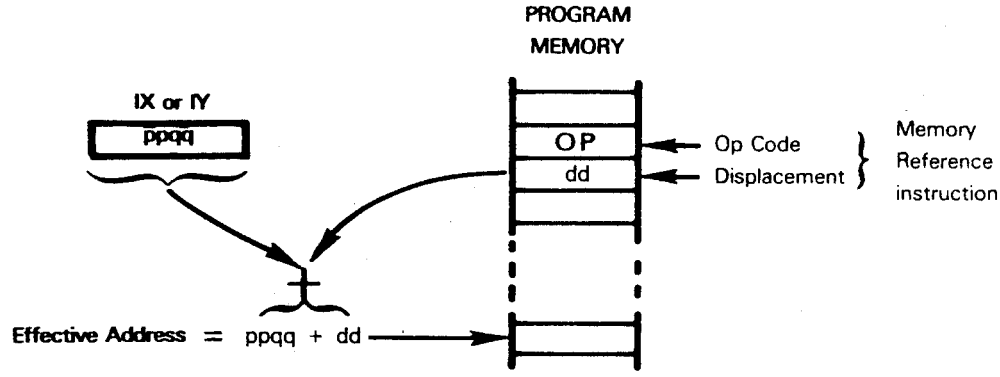
Z80 instructions use all of the 8080A addressing modes; the Z80 also has these two enhancements:

1) **A number of memory reference instructions use the IX and IY registers for indexed, or base relative addressing.**

2) **There are some two-byte program relative Jump instructions.**

Z80 INDEXED ADDRESSING

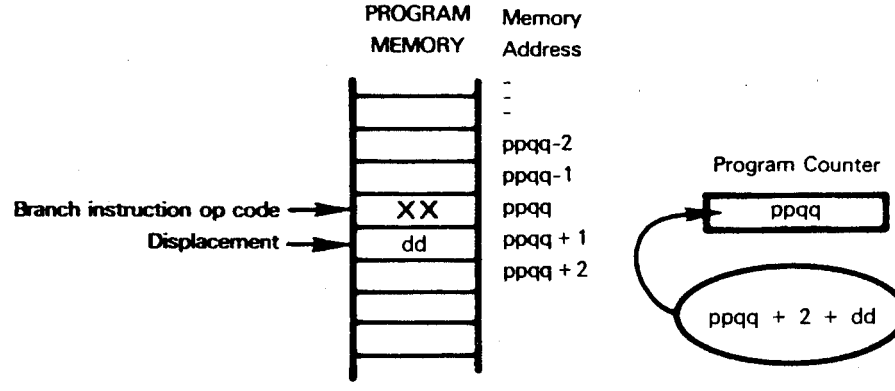
A memory reference instruction that uses the IX or IY register will include a single data displacement byte. The 8-bit value provided by the instruction object code is added to the 16-bit value provided by the identified Index register in order to compute the effective memory address:



p, q and d represent any hexadecimal digits;
dd represents an 8-bit, signed binary value.

This is standard microcomputer indexed addressing and is less powerful than having the memory reference instruction provide a 16-bit base address or displacement; for a discussion of these addressing modes see "Volume 1 — Basic Concepts", Chapter 6.

The program relative, two-byte Jump instructions provided by the Z80 provide standard two-byte, program relative addressing. A single, 8-bit displacement is provided by the Jump instruction's object code; this 8-bit displacement is added, as a signed binary value, to the contents of the Program Counter — after the Program Counter has been incremented to point to the sequential instruction:



The next instruction object code will be fetched from memory location ppqq+2+dd. p, q, and d represent any hexadecimal digits. dd represents a signed binary, 8-bit value.

For a discussion of program relative addressing, see "Volume 1 - Basic Concepts".

The Z80 addressing enhancements are of significant value when comparing the Z80 to the 8080A.

The value of the Index register comes not so much from having an additional addressing option, but rather IX and IY allow an efficient programmer to husband his CPU register space more effectively. Look upon IX and IY as performing memory addressing tasks which the 8080A would have to perform using the BC and DE registers. By freeing up the BC and DE registers for data manipulation, you can significantly reduce the number of memory reference instructions executed by the Z80.

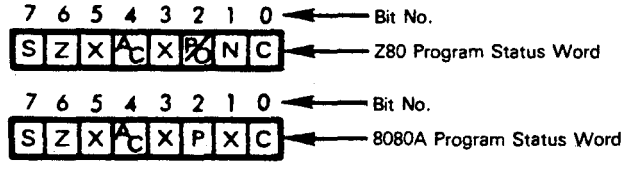
The two-byte program relative Jump instruction is useful because in most programs 80% of the Jump instructions branch to a memory location that is within 128 bytes of the Jump. That is the rationale for most microcomputers offering two-byte as well as three-byte Jump instructions.

Z80 STATUS

The Z80 and 8080A both use the Program Status Word in order to store status flags. These are the Z80 status flags:

Carry (C)
Zero (Z)
Sign (S)
Parity/Overflow (P/O)
Auxiliary Carry (A_C)
Subtract (N)

Statuses are recorded in the Program Status Word by the Z80, as compared to the 8080A, as follows:



The Parity/Overflow and Subtract statuses differ from the 8080A. All other statuses are the same. Note that the Z80, like the 8080A, uses borrow philosophy for the Carry status when performing subtract operations. That is to say, during a subtract operation, the Carry status takes the reciprocal value of any Carry out of the high-order bit. For details see the 8080A Carry status descriptions given in the 8080A chapter.

The 8080A has a Parity status but no Overflow status. The Z80 uses a single status flag for both operations, which makes a lot of sense. The Z80 Overflow status is absolutely standard, therefore only has meaning when signed binary arithmetic is being performed — at which time the Parity status has no meaning. Within the Z80, therefore, this single status is used by arithmetic operations to record overflow and by other operations to record parity. For a complete discussion of the Overflow status see "Volume 1 — Basic Concepts"

The Subtract status is used by the DAA instruction for BCD operations, to differentiate between decimal addition or subtraction. The Subtract and Auxiliary Carry statuses cannot be used as conditions for program branching (conditional Jump, Call or Return instructions).

Z80 CPU PINS AND SIGNALS

The Z80 CPU pins and signals are illustrated in Figure 7-4. Figure 7-2 provides the direct comparison between Z80 CPU signals and the standard 8080A, 8228, 8224 three-chip systems.

Let us first look at the Data and Address Busses.

The 16 address lines A0 - A15 output memory and I/O device addresses. The address lines are tristate; they may be floated by the Z80 CPU, giving external logic control of the Address Bus. **There is no difference between Z80 and 8080A Address Bus lines.**

The Data Bus lines D0 - D7 transmit bidirectional data into or out of the Z80 CPU. Like the Address Bus lines, the Data Bus lines are tristate. **The Z80 Data Bus lines do differ from the 8080A equivalent.** The 8080A Data Bus is multiplexed; status output on the Data Bus by the 8080A during the T2 clock period of very machine cycle is strobed by the SYNC pulse. The Z80 does not multiplex the Data Bus in this way. The Z80 Data Bus lines operate at normal TTL levels, whereas the 8080A Data Bus lines do not.

Z80 SYSTEM CONTROL SIGNALS

Control signals are described next; these may be divided into system control, CPU control and Bus control. First we will describe the System control signals.

$\overline{M1}$ identifies the instruction fetch machine cycle of an instruction's execution. Its function is similar, but not identical to the 8080A SYNC pulse. The Z80 PIO device uses the low $\overline{M1}$ pulse as a reset signal if it occurs without $\overline{IORQ}$ or $\overline{RD}$ simultaneously low.

$\overline{MREQ}$ identifies any memory access operation in progress; it is a tristate control signal.

$\overline{IORQ}$ identifies any I/O operation in progress. When $\overline{IORQ}$ is low, A0 - A7 contain a valid I/O port address. **$\overline{IORQ}$ is also used as an interrupt acknowledge; an interrupt is acknowledged by $\overline{M1}$ and $\overline{IORQ}$ being output low** — a unique combination, since $\overline{M1}$ is otherwise low only during an instruction fetch, which cannot address an I/O device.

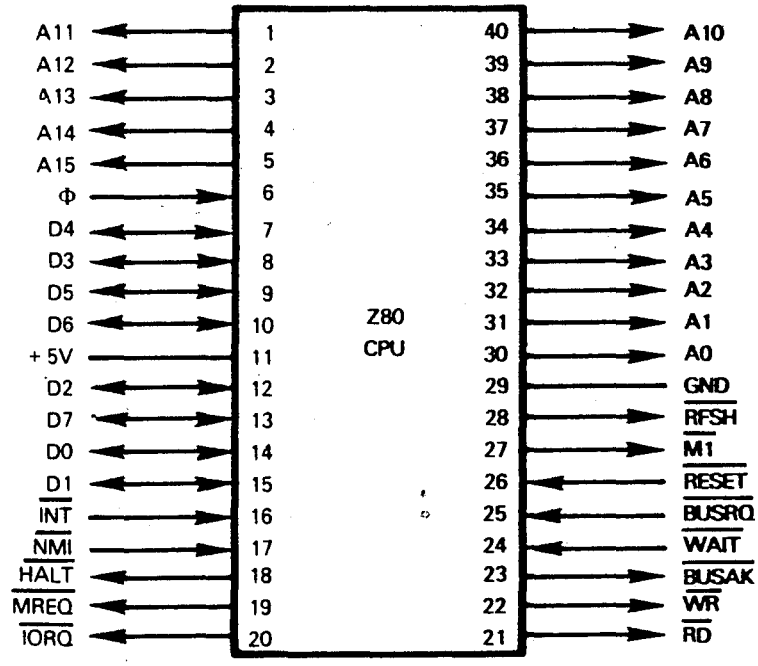
$\overline{RD}$ is a tristate signal which indicates that the CPU wishes to read data from either memory or an I/O device, as identified $\overline{MREQ}$ or $\overline{IORQ}$.

$\overline{WR}$ is a tristate control signal which indicates that the CPU wishes to write data to memory or an I/O device as indicated by $\overline{MREQ}$ and $\overline{IORQ}$. Some Z80 I/O devices have no $\overline{WR}$ input. These devices assume a Write operation when $\overline{IORQ}$ is low and $\overline{RD}$ is high. $\overline{RD}$ low specifies a Read operation.

The various ways in which the three control signals, $\overline{M1}$, $\overline{IORQ}$, and $\overline{RD}$, may be interpreted are summarized in Table 7-5, which occurs in the description of the Z80 PIO device.

$\overline{RFSH}$ is a control signal used to refresh dynamic memories. When $\overline{RFSH}$ is output low, the current $\overline{MREQ}$ signal should be used to refresh dynamic memory, as addressed by the lower seven bits of the Address Bus, A0 - A6.

Next we will describe CPU control signals.



PIN NAME	DESCRIPTION	TYPE
A0 - A15	Address Bus	Tristate, Output
D0 - D7	Data Bus	Tristate, Bidirectional
$\overline{M1}$	Identifies instruction fetch machine cycle	Output
$\overline{MREQ}$	Memory request — indicates that CPU is performing memory access	Tristate, Output
$\overline{IORQ}$	I/O request — indicates I/O operation in progress	Tristate, Output
$\overline{RD}$	CPU read from memory or I/O device	Tristate, Output
$\overline{WR}$	CPU write to memory or I/O device	Tristate, Output
$\overline{RFSH}$	Refresh dynamic memories	Output
$\overline{HALT}$	CPU Halt executed	Output
$\overline{WAIT}$	Wait state request	Input
$\overline{INT}$	Interrupt request	Input
$\overline{NMI}$	Nonmaskable interrupt request	Input
$\overline{RESET}$	Reset and initialize CPU	Input
$\overline{BUSRQ}$	Request for control of Address, Data and Control Busses	Input
$\overline{BUSAK}$	Bus acknowledge	Output
Φ	CPU clock	Input
+5V, GND	Power and Ground	

Figure 7-4. Z80 CPU Signals and Pin Assignments

Z80 CPU CONTROL SIGNALS

$\overline{\text{HALT}}$ is output low following execution of a Halt instruction. The CPU now enters a Halt state during which it continuously re-executes a NOP instruction in order to maintain memory refresh activity. A Halt can only be terminated with an interrupt.

$\overline{\text{WAIT}}$ is equivalent to the 8080A READY input. External logic which cannot respond to a CPU access request within the allowed time interval extends the time interval by pulling the $\overline{\text{WAIT}}$ input low. In response to $\overline{\text{WAIT}}$ low, the Z80 enters a Wait state during which the CPU inserts an integral number of clock periods; taken together, these clock periods constitute a Wait state.

$\overline{\text{INT}}$ and $\overline{\text{NMI}}$ are two interrupt request inputs. The difference between these two signals is that $\overline{\text{NMI}}$ has higher priority and cannot be disabled.

Z80 BUS CONTROL SIGNALS

There are two Bus control signals.

$\overline{\text{RESET}}$ is a standard reset control input. When the Z80 is reset, this is what happens:

The Program Counter, IV and R registers' contents are all set to zero.

Interrupt requests via $\overline{\text{INT}}$ are disabled.

All tristate bus signals are floated.

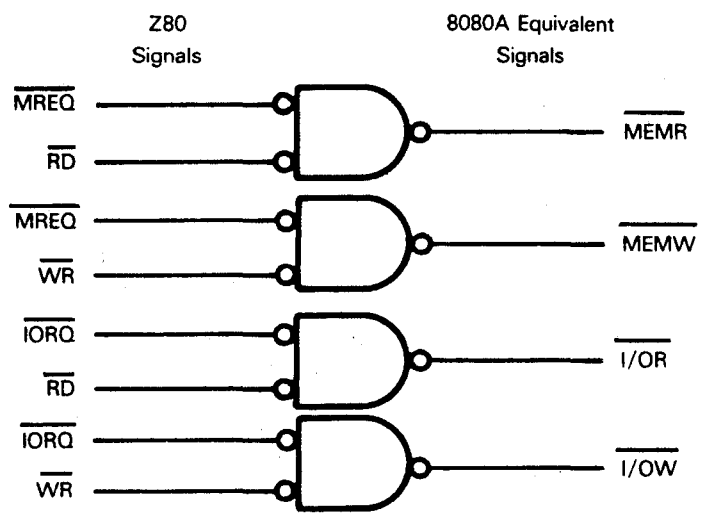
$\overline{\text{BUSRQ}}$ and $\overline{\text{BUSAK}}$ are bus request and acknowledge signals. In order to perform any kind of DMA operation, external logic must acquire control of the microcomputer System Bus. This is done by inputting $\overline{\text{BUSRQ}}$ low; at the conclusion of the current machine cycle, the Z80 CPU will float all tristate bus lines and will acknowledge the bus request by outputting $\overline{\text{BUSAK}}$ low.

Z80 - 8080A SIGNAL COMPATIBILITY

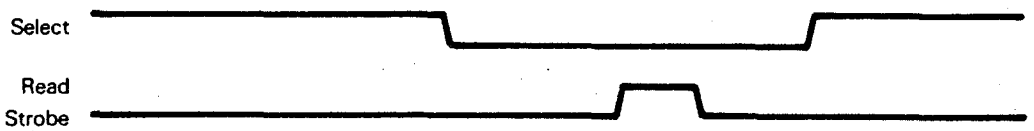
If you are designing a new product around the Z80 CPU, then questions of Z80 - 8080A signal compatibility are irrelevant; you will design for the CPU on hand.

If you are replacing an 8080A with a Z80, then it would be helpful to have some type of lookup table which directly relates 8080A signals to Z80 signals. Unfortunately, such a lookup table cannot easily be created. The problem is that the Z80 is an implementation of three devices; the 8080A CPU, the 8224 Clock, and 8228 System Controller; but there are very many 8080A configurations that do not include an 8228 System Controller.

Possibly the most important conceptual difference between the Z80 and 8080A involves read and write control signals. **The 8228 System Controller develops four discrete control signals for memory read, memory write, I/O read and I/O write. The Z80 has a general read and a general write, coupled with an I/O select and a memory select.** By adding logic, it would be easy enough to generate the four discrete 8080A signals from the two Z80 signal pairs; here is one elementary possibility:

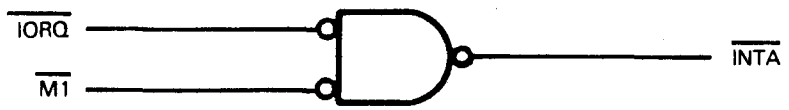


If your design allows it, however, it would be wiser to extend the Z80 philosophy to the various support devices surrounding the CPU. Recall from our discussion of 8080A support devices in Chapter 4 that every device requires separate device select and device access logic. For some arbitrary read operation, timing might be illustrated as follows:



With an 8080A scheme, select logic is decoded from Address Bus lines, while strobe logic depends on one of the four control lines $\overline{I/OR}$, $\overline{I/OW}$, $\overline{MEMR}$ or $\overline{MEMW}$. Using the Z80 philosophy, the memory select ($\overline{MREQ}$) or I/O select ($\overline{IORQ}$) control lines become part of the device select logic, while the read ($\overline{RD}$) or write ($\overline{WR}$) controls generate the strobe.

The Z80 has no interrupt acknowledge signal; rather it combines $\overline{IORQ}$ with $\overline{M1}$ as follows:



Some Z80 support devices also check for a "Return-from-Interrupt" instruction object code appearing on the Data Bus during an instruction fetch (when $\overline{M1}$ and $\overline{RD}$ will both be low). This condition is used to reset interrupt priorities among Z80 support devices.

The 8080A HOLD and HLDA signals are functionally reproduced by the Z80 $\overline{BUSRQ}$ and $\overline{BUSAK}$ signals.

The 8080A SYNC pulse has no direct Z80 equivalent. $\overline{M1}$ is pulsed low during an instruction fetch, or an interrupt acknowledge, but it is not pulsed low during the initial time periods of an instruction's second or subsequent machine cycles. **Frequently the complement of $\overline{M1}$ can be used instead of SYNC** to drive those 8080A peripheral devices that require the SYNC pulse.

The Z80 has no signals equivalent to 8080A INTE, WAIT or Φ2. There is also no signal equivalent to the 8228 $\overline{BUSEN}$.

If for any reason external logic must know when interrupts have been disabled internally by the CPU, then the Z80 will be at a loss to provide any signal equivalent to the 8080A control signals. Remember INTE in an 8080A system tells external logic when the CPU has enabled or disabled all interrupts;since external logic can do nothing about interrupts being disabled, and requesting an interrupt at this time does neither good nor harm, knowing that the condition exists is generally irrelevant.

The single Z80 $\overline{WAIT}$ input serves the function of the 8080A READY input. Irrespective of when the WAIT is requested, a Wait clock period will only be inserted between T_2 and T_3; moreover, as we will see shortly, there are certain Z80 instructions which automatically insert a Wait state, without waiting for external demand. You would need relatively complex logic to decode instruction object codes, clock signal and the $\overline{WAIT}$ input if your Z80 system is to generate the equivalent of an 8080A WAIT output. In all probability, it would be simpler to find an alternative scheme that did not require a signal equivalent to the 8080A WAIT output.

The Z80 simply has no second clock equivalent to 8080A Φ2. Any device that needs clock signal Φ2 cannot easily be used in Z80 configurations.

The 8228 $\overline{BUSEN}$ input is used by external logic to float the System Bus. In a Z80 system, CPU logic floats the System Bus; therefore $\overline{BUSEN}$ becomes irrelevant.

The 8080A CPU has no signals equivalent to Z80 $\overline{RFSH}$, $\overline{HALT}$ and $\overline{NMI}$.

$\overline{RFSH}$ applies to dynamic memory refresh only; it is irrelevant within the context of a Z80 - 8080A signal comparison.

$\overline{NMI}$, being a nonmaskable interrupt request, also has no 8080A equivalent logic.

The Z80 $\overline{HALT}$ output needs some discussion. One of the more confusing aspects of the 8080A is the interaction of Wait, Halt and Hold states. Let us look at these three states, comparing the Z80 and 8080A configurations and in the process we will see the purpose of the Z80 $\overline{HALT}$ output.

The purpose of the Wait state is to elongate a memory reference machine cycle in deference to slow external memory or I/O devices. The Wait state consists of one or more Wait clock periods inserted between T_2 and T_3 of a machine cycle. The 8080A and the Z80 handle Wait states in exactly the same way, except for the fact that the Z80 has no Wait acknowledge output and under certain circumstances will automatically insert Wait clock periods.

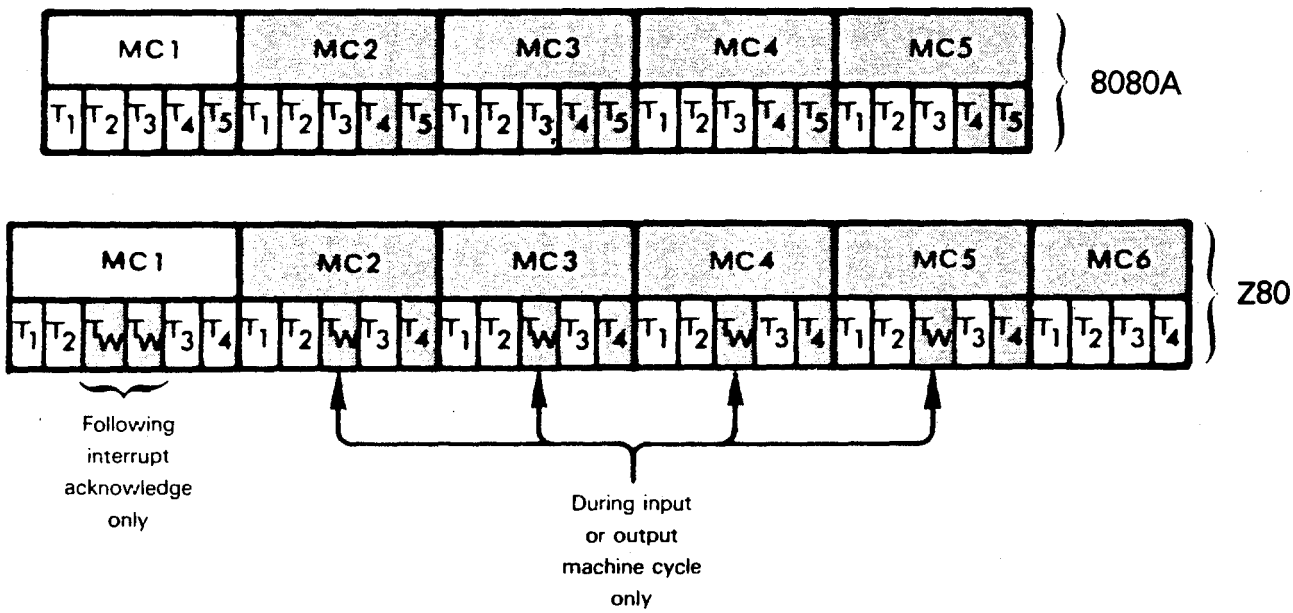
The purpose of the Hold condition is to allow external logic to acquire control of the System Bus and perform Direct Memory Access operations. Again both the Z80 and the 8080A have very similar Hold states. The only significant difference is that the Z80 initiates a Hold state at the conclusion of a machine cycle, whereas the 8080A initiates the Hold state during time period T_3 or T_4. The 8228 System Controller also needs a high BUSEN input in order to float its Data and Control Busses while the Z80 has no equivalent need.

The big difference between the Z80 and the 8080A comes within the Halt state. When the 8080A executes a Halt instruction, it goes into a Halt state, which differs from a Hold state. There are some complex interactions between Hold, Halt, Wait and interrupts within 8080A systems. None of these complications exists in the Z80 system, since the Z80 has no Halt state. After executing a Halt instruction, the Z80 outputs HALT low, then proceeds to continuously execute a NOP instruction. This allows dynamic memory refresh logic to continue operating. **If you are replacing an 8080A with a Z80, you must give careful attention to the Halt state. This is one condition where unexpected incompatibilities can arise.**

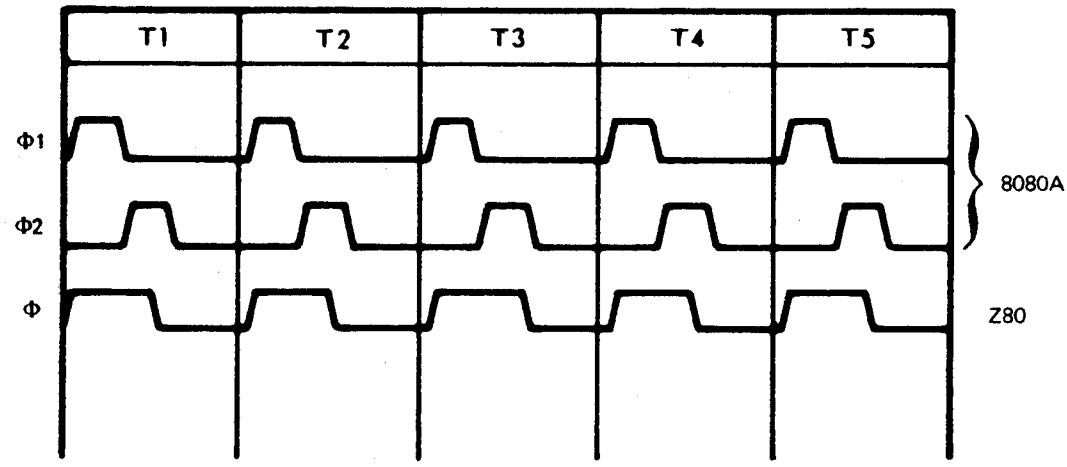
Z80 TIMING AND INSTRUCTION EXECUTION

Z80 timing is conceptually similar to, but far simpler than 8080A timing. Like the 8080A, the Z80 divides its instructions into machine cycles and clock periods. However, all Z80 machine cycles consist of either three or four clock periods. Some instructions always insert Wait clock periods, in which case five or six clock periods may be present in a machine cycle. Recall that 8080A machine cycles may have three, four or five clock periods.

The 8080A may require from one to five machine cycles in order to execute an instruction; Z80 instructions execute in one to six machine cycles. If we shade optional machine cycles and clock periods, Z80 and 8080A instruction time subdivisions may be compared and illustrated as follows:



Z80 clock signals are also far simpler than the 8080A equivalent. Where the 8080A uses two clock signals the Z80 uses one. Clock logic may be compared as follows:



INSTRUCTION FETCH EXECUTION SEQUENCES

As compared to the 8080A, Z80 instruction timing is marvelously simple. Gone is the SYNC pulse and the decoding of Data Bus for status. Every instruction's timing degenerates into an instruction fetch, optionally followed by memory or I/O read or write. Add to this a few variations for Wait state, interrupt acknowledge and bus floating and you are done.

Let us begin by looking at an instruction fetch. Timing is illustrated in Figure 7-5. Look at the instruction fetch timing in the 8080A chapter to obtain an immediate comparison of the Z80 and the 8080A.

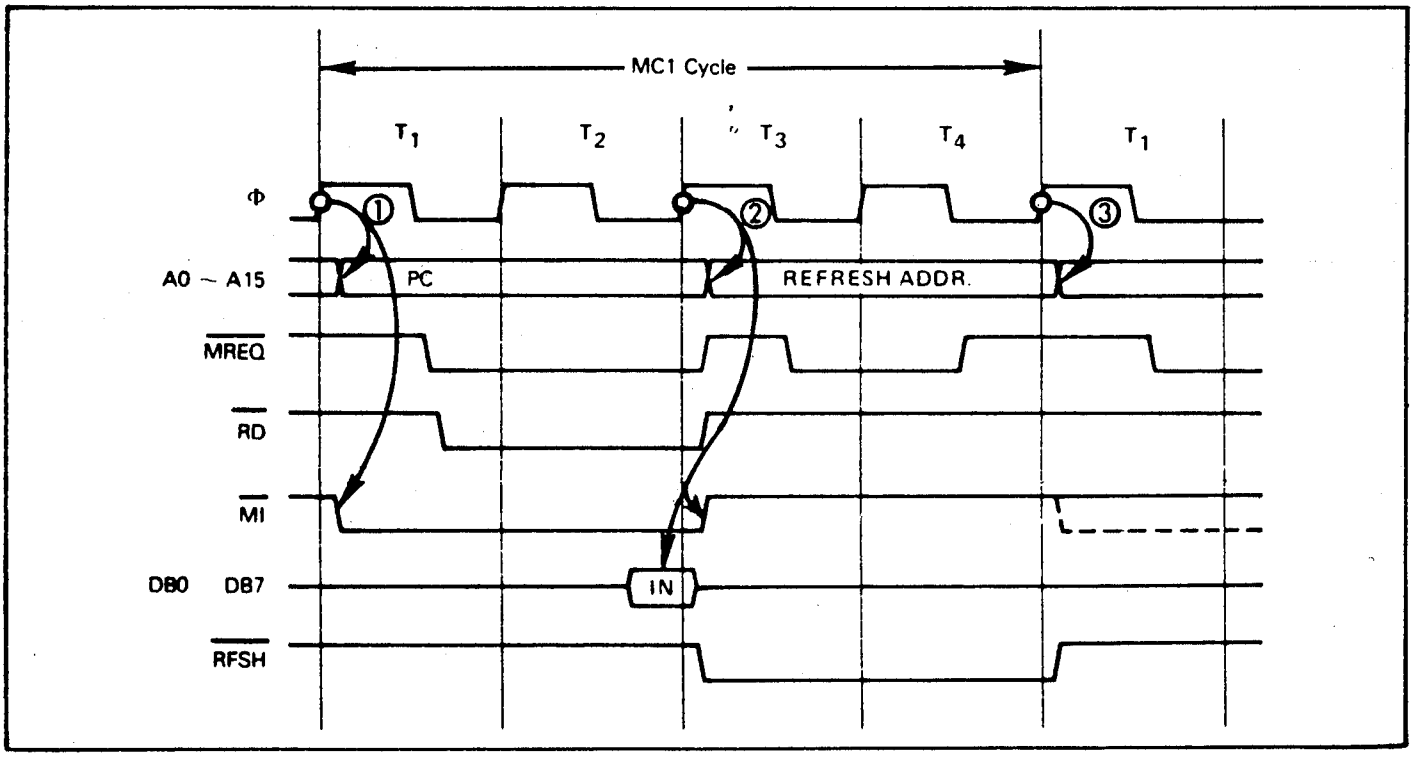


Figure 7-5. Z80 Instruction Fetch Sequence

Referring to Figure 7-5, note that the instruction fetch cycle is identified by $\overline{M1}$ output low during T_1 and T_2 (①). Since there is no status on the Data Bus to worry about, the Program Counter contents are output immediately on the Address Bus and stay stable for the duration of T_1 and T_2.

Since an instruction fetch is also a memory operation, $\overline{MREQ}$ and $\overline{RD}$ controls are both output low. This occurs half-way through T_1, at which time the Address Bus will stabilize. The falling edges of $\overline{MREQ}$ and $\overline{RD}$ can therefore be used to select a memory device and strobe data out. The CPU polls data on the Data Bus at the rising edge of the T_3 clock (②).

Clock perods T_3 and T_4 of the instruction fetch machine cycle are used by the Z80 CPU for internal operations. These clock periods are also used **to refresh dynamic memory.** As soon as the Program Counter contents are taken off the Address Bus (②), the refresh address from the Refresh register is output on lines A0 - A6 of the Address Bus. This address stays on the Address Bus until the conclusion of T_4 (③).

Since a memory refresh is a memory access operation, $\overline{MREQ}$ is again output low; however, it is accompanied by $\overline{RFSH}$ rather than $\overline{RD}$ low. Thus memory reference logic does not attempt to read data during a refresh cycle.

A MEMORY READ OPERATION

Memory interface logic responds to an instruction fetch and a memory read in exactly the same way. There are, however, a few differeces between memory read and instruction fetch timing. Memory read timing is illustrated in Figure 7-6. The principal difference to note is that during a memory read operation, the data is sampled on the falling edge of the T_3 clock pulse, whereas during an instruction fetch it is sampled on the rising edge of this clock pulse. Also a normal memory read machine cycle will consist of three clock periods, while the normal instruction fetch consists of four clock periods. Remember also that the Z80 identifies an instruction fetch machine cycle by outputting $\overline{M1}$ low during the first two clock periods of the instruction fetch machine cycle.

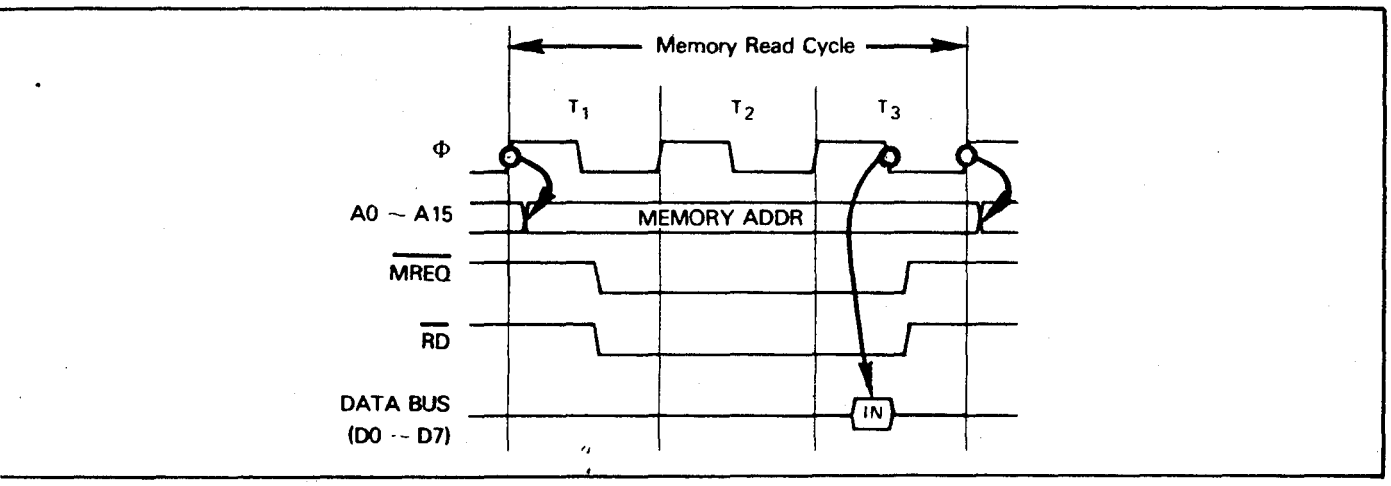


Figure 7-6. Z80 Memory Read Timing

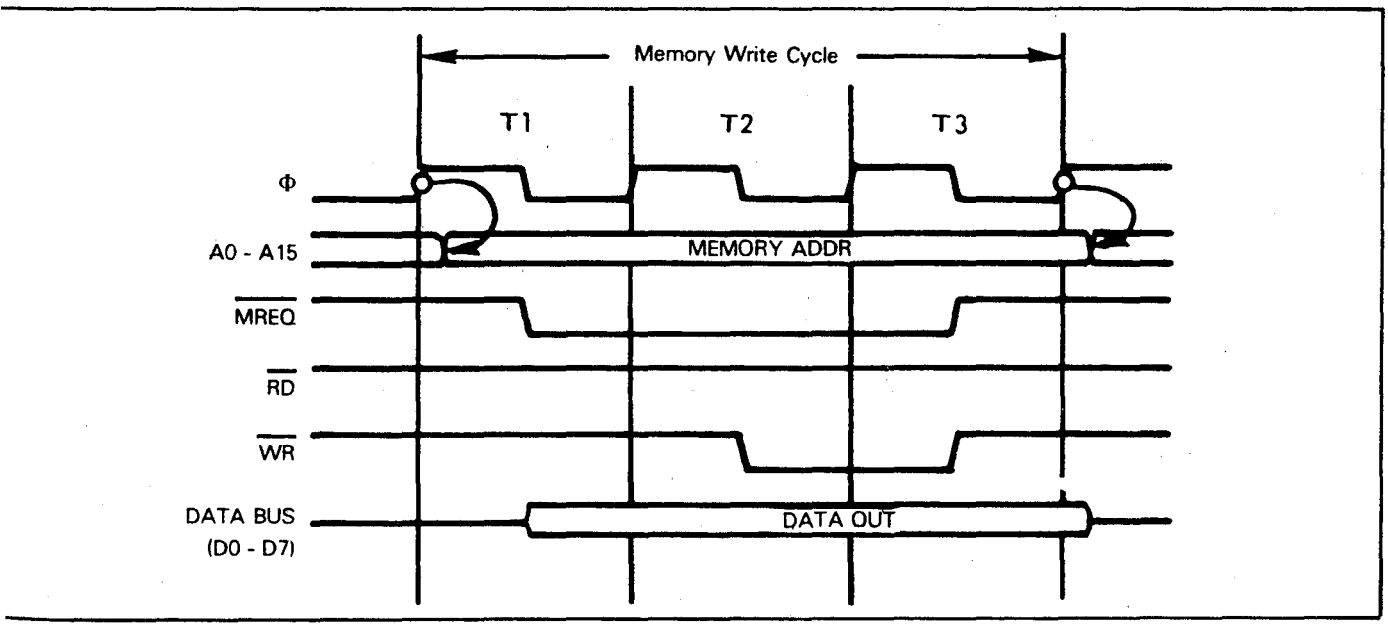


Figure 7-7. Z80 Memory Write Timing

MEMORY WRITE OPERATION

Figure 7-7 illustrates memory write timing for the Z80. The only differences between memory read and memory write timing are the obvious ones: $\overline{WR}$ is pulsed low for a write, and can be used as a strobe by memory interface logic to read data off the Data Bus.

THE WAIT STATE

Like the 8080A, **the Z80 allows a Wait state to occur between clock periods T_2 and T_3 of a machine cycle.** The Wait state frees external logic or memory from having to operate at CPU speed.

The Z80 CPU samples the $\overline{WAIT}$ input on the falling edge of Φ during T_2. Providing $\overline{WAIT}$ is low on the falling edge of Φ during T_2, Wait clock periods will be inserted. The number of Wait clock periods inserted depends strictly on how long the $\overline{WAIT}$ input is held low. As soon as the Z80 detects $\overline{WAIT}$ high on the falling edge of Φ, it will initiate T_3 on the next rising edge of Φ.

Note that the single Z80 $\overline{WAIT}$ signal replaces the READY and $\overline{WAIT}$ 8080A signals. As this would imply, no signal is output telling external logic the Z80 has entered the Wait state. **In the event that external logic needs to know whether or not a Wait state has been entered, these are the rules:**

1) The Z80 will sample $\overline{WAIT}$ on the falling edge of Φ in T_2.
2) If $\overline{WAIT}$ is low, then the Z80 will continue to sample the $\overline{WAIT}$ input for all subsequent Wait state clock periods.
3) The Z80 will not sample the $\overline{WAIT}$ input during any clock period other than T_2 or a Wait state.

Figure 7-8 illustrates Z80 Wait state timing.

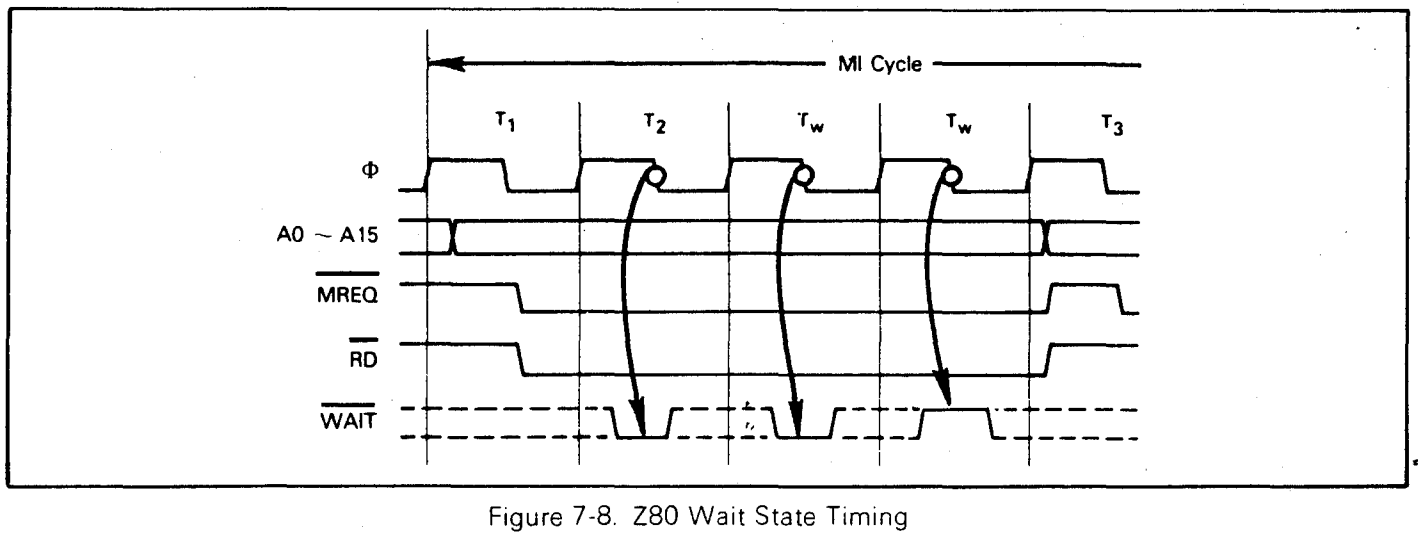


Figure 7-8. Z80 Wait State Timing

INPUT OR OUTPUT GENERATION

Timing for Z80 input and output generation is given in Figures 7-9 and 7-10.

The important point to note is that Zilog has acknowledged the infrequency with which typical I/O logic can operate at CPU speed. **One Wait clock period is therefore automatically inserted between T_2 and T_3 for all input or output machine cycles.** Otherwise timing differs from memory read and write operations only in that $\overline{IORQ}$ is output low rather than $\overline{MREQ}$.

Note that there is absolutely nothing to prevent you from selecting I/O devices within the memory space. This is something we did consistently in the 8080A chapter when describing 8080A support devices. But if you adopt this design policy, remember that your I/O logic must execute at CPU speed, unless you insert Wait states.

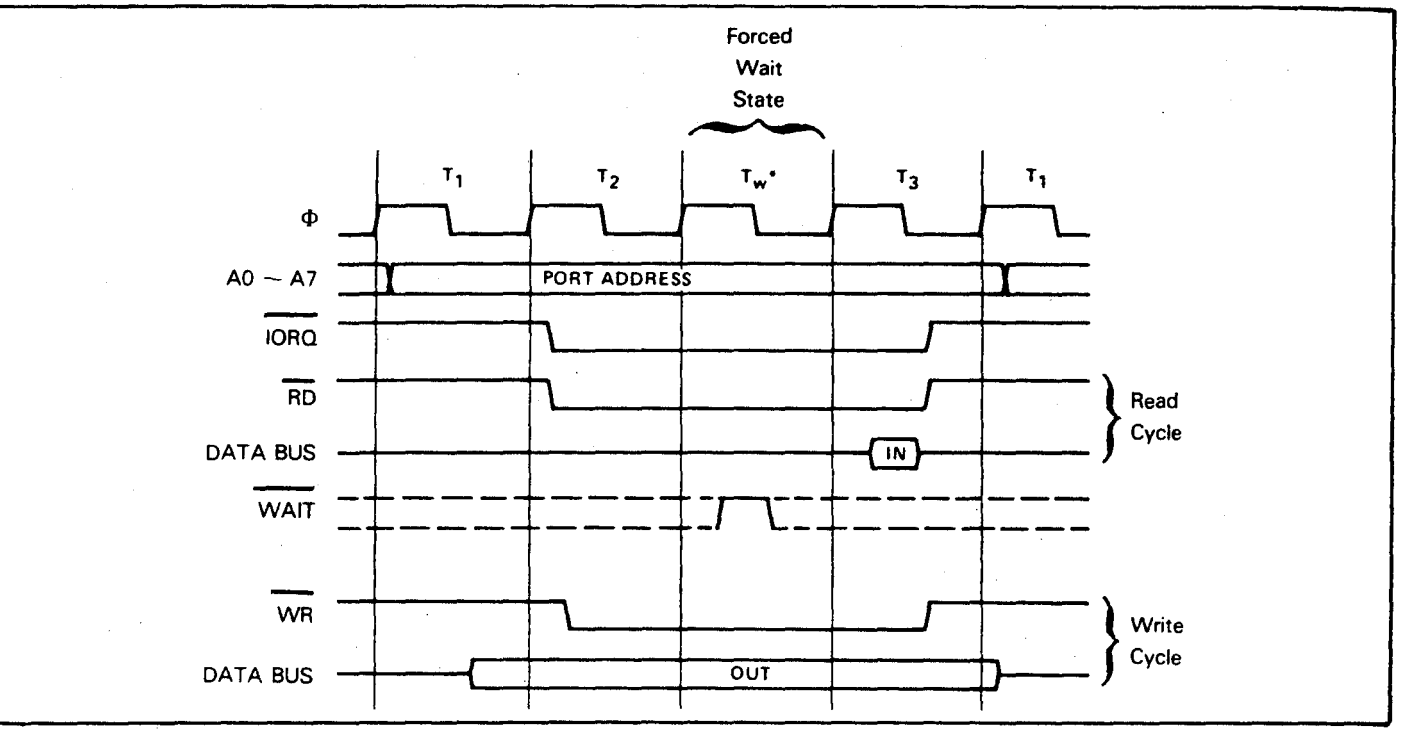


Figure 7-9. Z80 Input or Output Cycles

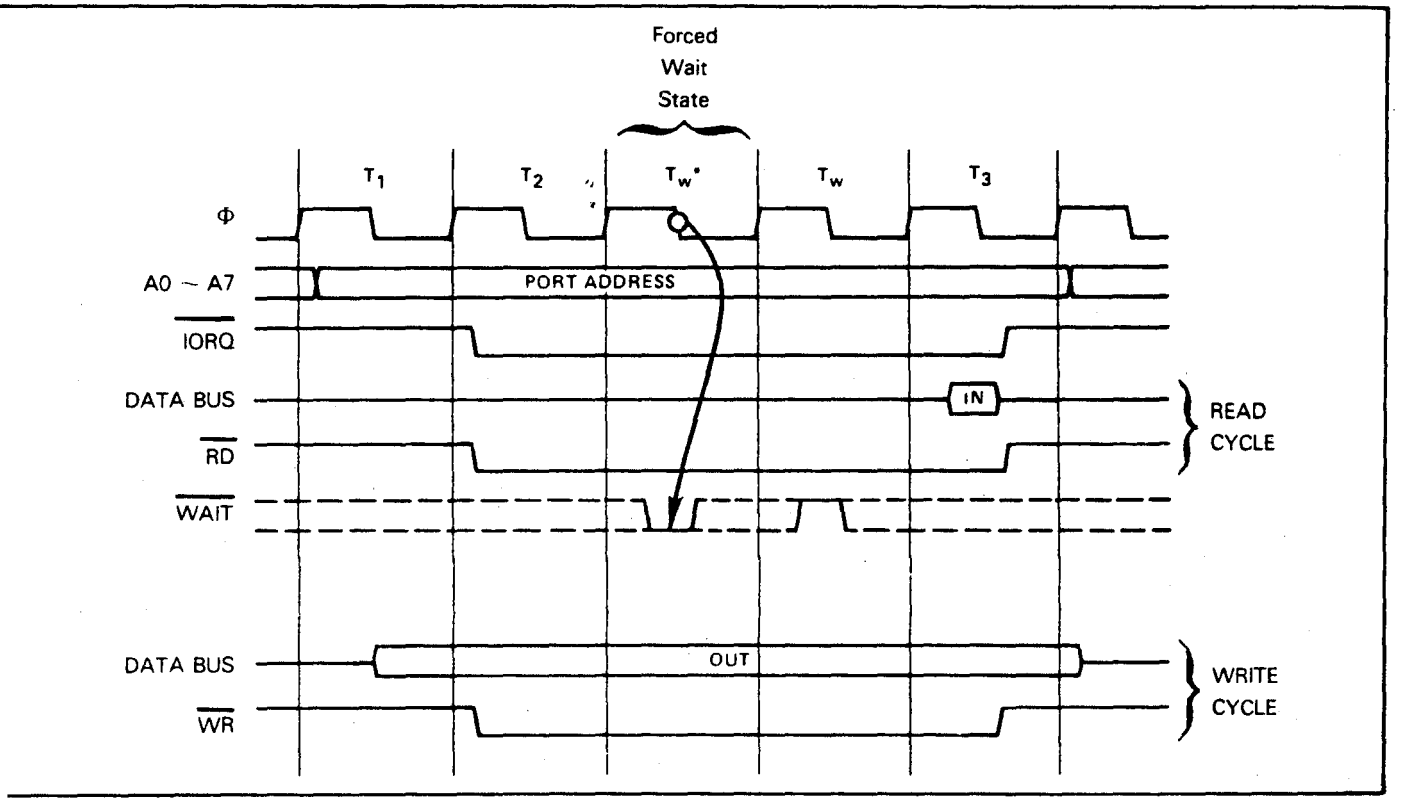


Figure 7-10. Z80 Input or Output Cycles with Wait States

US REQUESTS

ıe Z80 does not have a Hold state as described for the 8080A, but Z80 bus request logic is equivalent. **The Z80 will ɔat Address, Data and tristate Control Bus lines upon sensing a low BUSRQ signal.** BUSRQ is sampled by the 30 CPU on the rising edge of the last clock pulse of any machine cycle. If BUSRQ is sampled low, then tristate lines are ɔated by the CPU, which also outputs BUSAK low. The Z80 CPU continues to sample BUSRQ on the rising edge of ev- y clock pulse. As soon as BUSRQ is sensed high, floating will cease on the next clock pulse. This timing is illustrated Figure 7-11.

One significant difference between the Z80 and 8080A results from differences between the Hold and bus floating states. As the logic we have described for the Z80 would imply, it will only float the System Bus in between machine cycles. The 8080A, on the other hand, will enter a Hold state variably during T_3 or T_4 of the machine cycle, depending on the type of operation in progress. It is therefore possible for the Z80 to float its bus three clock periods later than an 8080A in a similar configuration.

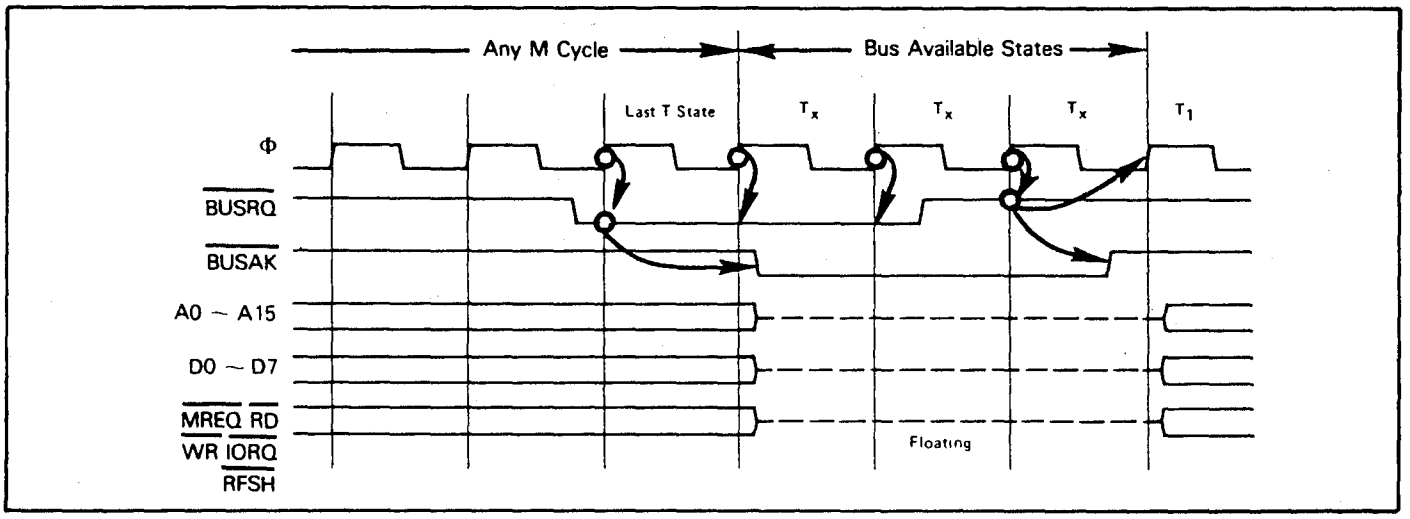


Figure 7-11. Z80 Bus Timing

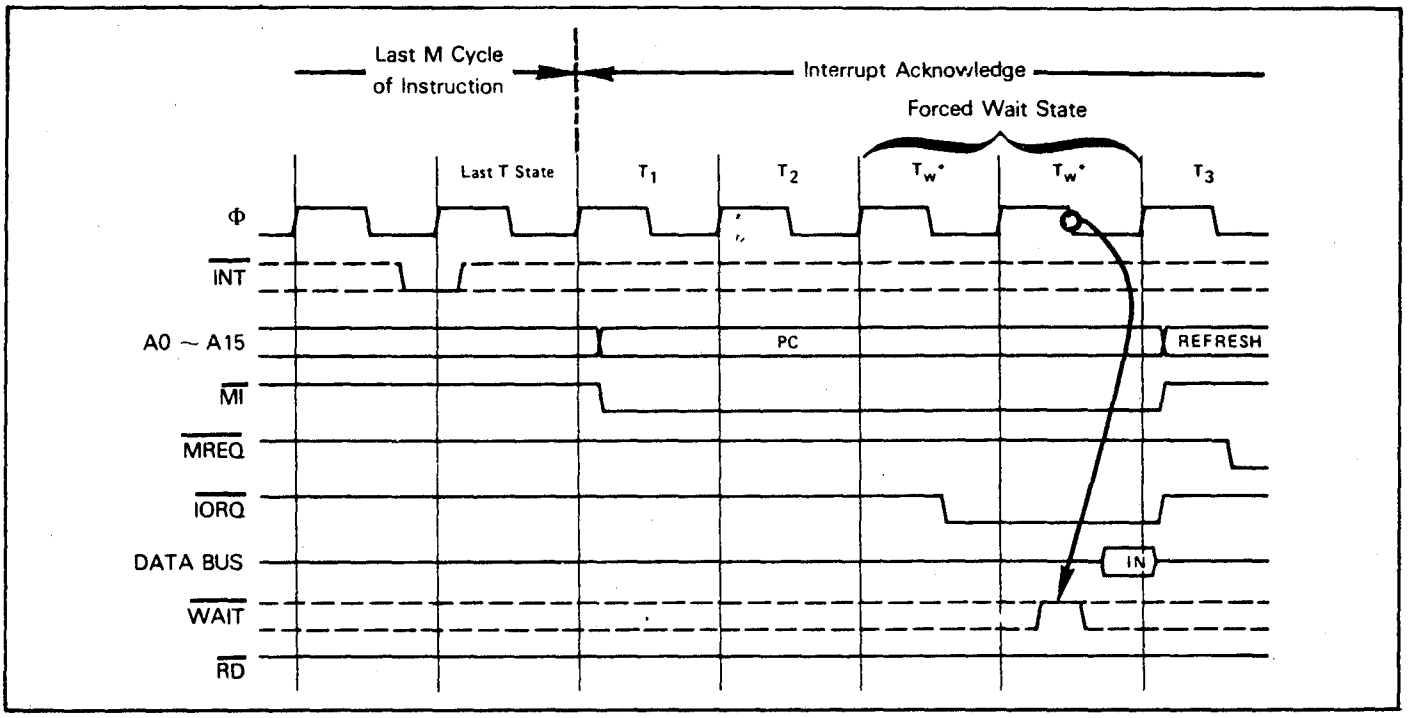


Figure 7-12. Z80 Response to a Maskable Interrupt Request

Note also that **if you are using the dynamic memory refresh logic of the Z80, then during long bus floats, external logic must refresh dynamic memory.** The simplest way around this problem in a Z80 system is to ensure that DMA operations acquire the System Bus for many short periods of time, rather than for a single long access.

EXTERNAL INTERRUPTS

The Z80 has two interrupt request input signals: $\overline{INT}$ and $\overline{NMI}$. The $\overline{NMI}$ (non-maskable interrupt) input cannot be disabled and has a higher priority than the $\overline{INT}$ interrupt input. There are three different operating or response modes for the $\overline{INT}$ input, while the response to $\overline{NMI}$ is simple and straightforward. Let us therefore begin by describing the $\overline{INT}$ interrupt request.

Timing for $\overline{INT}$ interrupt request and acknowledge sequence differs significantly from that of the 8080A interrupt request and is illustrated in Figure 7-12.

The interrupt request signal $\overline{INT}$ is sampled by the Z80 CPU on the rising edge of the last clock pulse of any instruction's execution. Note that there is an exception to this statement: during execution of block search and transfer instructions, the interrupt request signal is sampled after each byte of data is transferred/compared.

An interrupt request will be denied if interrupts have been disabled under program control, or if the $\overline{BUSRQ}$ signal is also low. Thus a DMA access will have priority over maskable interrupts.

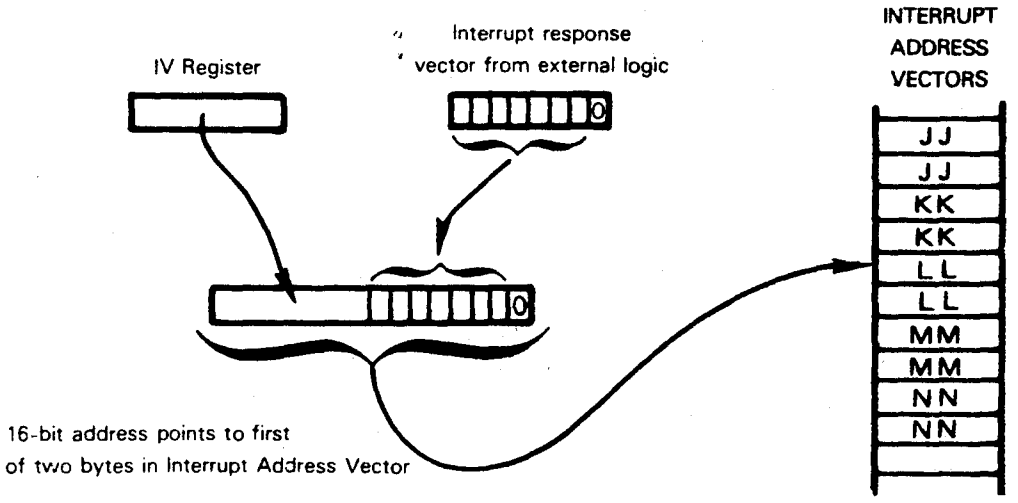
The Z80 CPU acknowledges an interrupt request by outputting $\overline{M1}$ and $\overline{IORQ}$ low. This occurs in a special interrupt acknowledge machine cycle, as illustrated in Figure 7-12. Note that this machine cycle has two Wait states inserted so that external logic will have time for any type of daisy chained priority interrupt scheme to be implemented.

When $\overline{IORQ}$ is output low while $\overline{M1}$ is low, external logic must interpret this signal combination as requiring an interrupt vector to be placed on the Data Bus by the acknowledged external interrupt requesting source. This interrupt vector can take one of three forms; the form depends on which of the three modes you have selected for the Z80 under program control.

In **Mode 0,** the interrupt vector will be interpreted as an object code, representing the first instruction to be executed following the interrupt acknowledge. If a multi-byte object code is supplied, then the bytes following the first must be supplied during subsequent machine cycles. This **is equivalent to the standard interrupt response of the 8080A.** Whenever you are replacing an 8080A with a Z80, therefore, the Z80 must operate in interrupt response Mode 0.

Z80 interrupt response logic in **Mode 1 automatically assumes that the first instruction executed following the interrupt response will be a Restart, branching to memory location 0038_{16}.** If the Z80 is in Mode 1, no interrupt vector is needed.

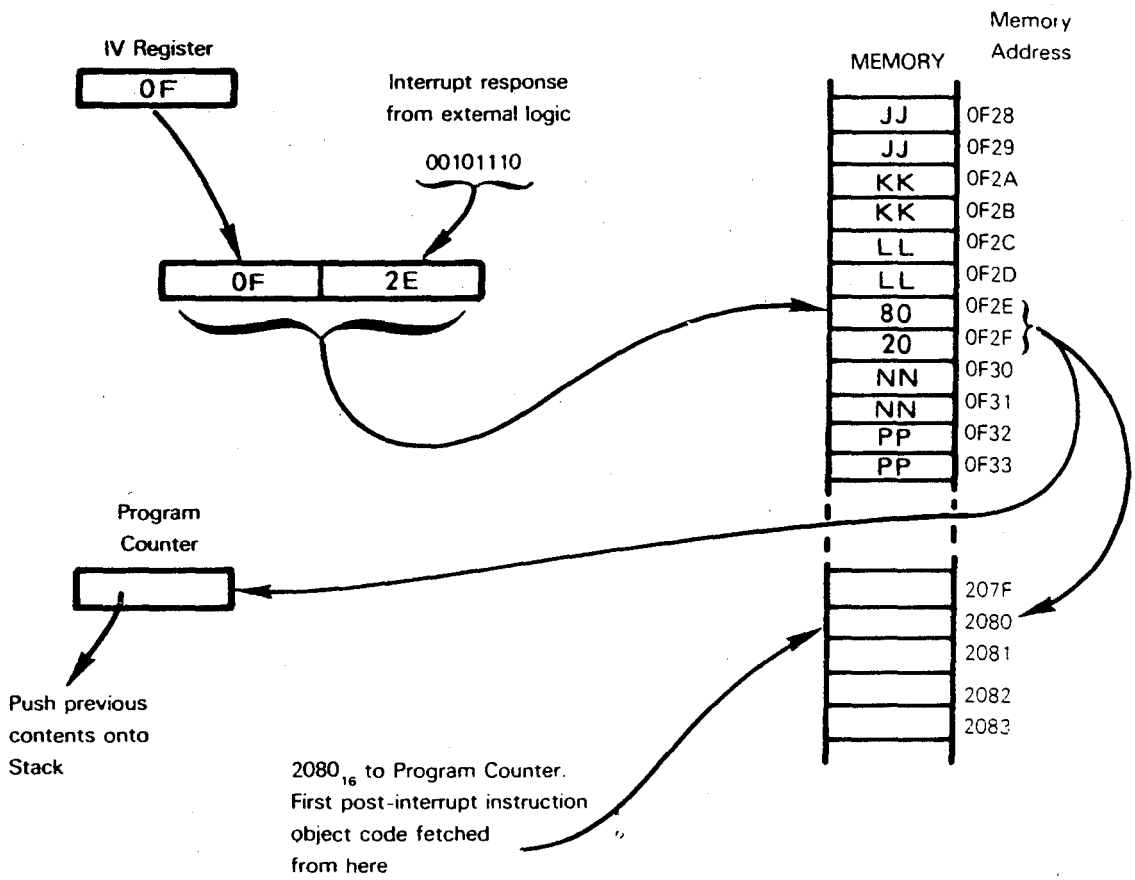
Z80 Mode 2 interrupt response has no 8080A equivalent. When you operate the Z80 **in Mode 2, you must create a table of 16-bit interrupt address vectors,** which can reside anywhere in addressable memory. These 16-bit addresses identify the first executable instruction of interrupt service routines. When an interrupt is acknowledged by the CPU in Mode 2, **the acknowledged external logic must place an interrupt response vector on the Data Bus. The Z80 CPU will combine the IV register contents with the interrupt acknowledge vector to form a 16-bit address, which accesses the interrupt address vector table.** Since 16-bit addresses must lie at even memory address boundaries, only seven of the eight bits provided by the acknowledged external logic will be used to create the table address; the low order bit will be set to 0. Thus the table of 16-bit interrupt address vectors will be accessed as follows:



The Z80 CPU will execute a Call to the memory location obtained from the interrupt address vector table.

Let us clarify this logic with a simple example. Suppose that you have 64 possible external interrupts; each interrupt has its own interrupt service routine, therefore 64 starting addresses will be stored in 128 bytes of memory. Let us arbitrarily assume that these 128 bytes are stored in a table with memory addresses $0F00_{16}$ through $0F7F_{16}$. Now in

order to use Mode 2, you must initially load the value $0F_{16}$ into the Z80 IV register. Subsequently an external interrupt request is acknowledged and the acknowledged external logic returns on the Data Bus the vector $2E_{16}$; this is what will happen:



If two Wait states are insufficient for external logic to arbitrate interrupt priorities and place the required vector on the Data Bus, **then additional Wait states can be inserted** in the usual way by inputting $\overline{WAIT}$ low. Timing is illustrated in Figure 7-13.

Z80 WAIT STATES DURING INTERRUPT ACKNOWLEDGE

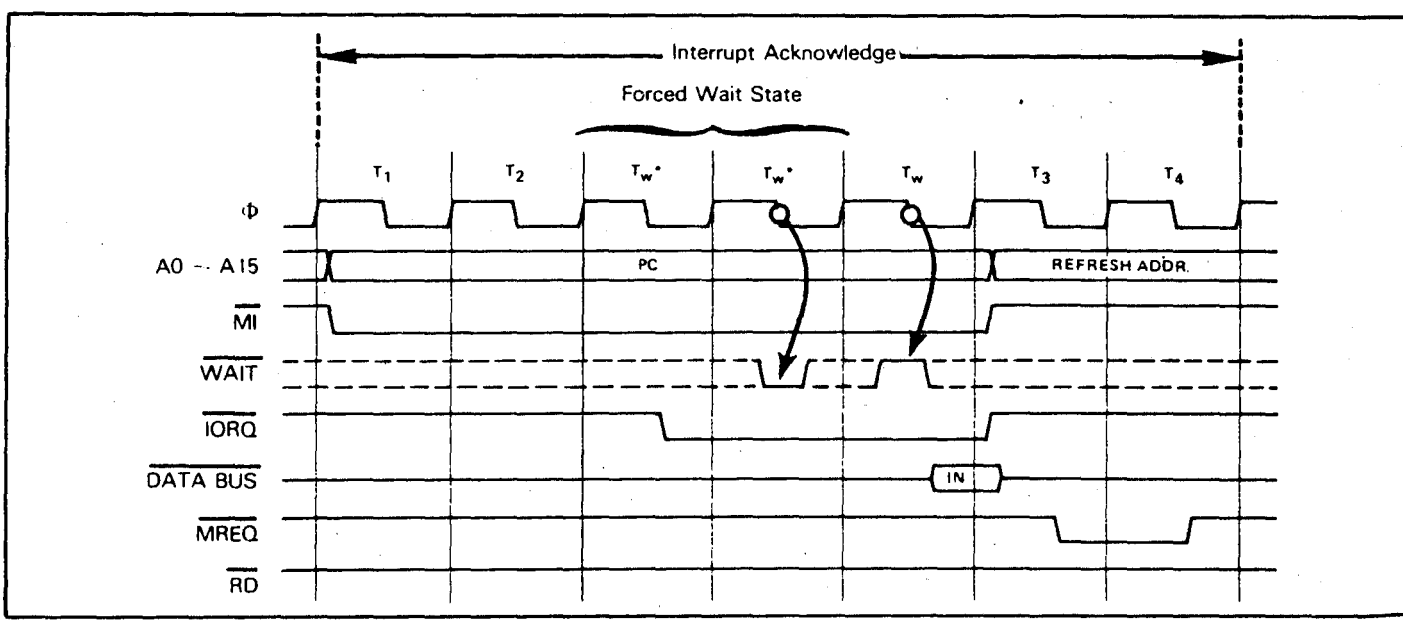


Figure 7-13. Wait States During Z80 Response to a Maskable Interrupt Request

Z80 NON-MASKABLE INTERRUPT

The response of the Z80 CPU to the non-maskable interrupt ($\overline{NMI}$) is quite similar to Mode 1 interrupt operation. There are a number of significant differences, however. First of all, the $\overline{NMI}$ interrupt cannot be disabled and has priority over the $\overline{INT}$ interrupt. (Remember that $\overline{BUSRQ}$ has priority over both interrupt inputs.)

Next, the non-maskable interrupt is an edge-sensitive (negative edge triggered) input. The Z80 reacts only to the edge of a pulse on the $\overline{NMI}$ line, rather than to a low level as is the case with the $\overline{INT}$ input. The negative edge of the $\overline{NMI}$ input causes an internal flip-flop to be set in the Z80, and this flip-flop is checked during the last cycle of an instruction execution. The CPU response to this interrupt is similar to a normal memory read operation except that the Data Bus is ignored on the next $\overline{M1}$ cycle. Timing for the interrupt response to the non-maskable interrupt request is illustrated in Figure 7-14.

The Z80 pushes the contents of the Program Counter onto the external stack and then automatically executes a Restart instruction to memory location 0066_{16}. Thus, this response is the same as the response to an $\overline{INT}$ interrupt in Mode 1 except that the Restart call is to a different memory location.

While the Z80 CPU is responding to the non-maskable interrupt, the internal flip-flop (IFF1) used to enable maskable interrupts is reset to prevent interrupts during the $\overline{NMI}$ service routine. Upon completion of the service routine, you do not simply want to once again set the IFF1 flip-flop, since maskable interrupts may not have been enabled prior to $\overline{NMI}$. This quandary is solved by using a second internal flip-flop (IFF2) for temporary storage. As the CPU begins its response to the $\overline{NMI}$ interrupt, it saves the state of the interrupt enable flip-flop (IFF1) by copying it into IFF2. At the end of the $\overline{NMI}$ service routine, you must execute a Return from Non-Maskable Interrupt (RETN) instruction which will copy the contents of IFF2 back into IFF1, thus automatically restoring the correction status for the maskable interrupt enable.

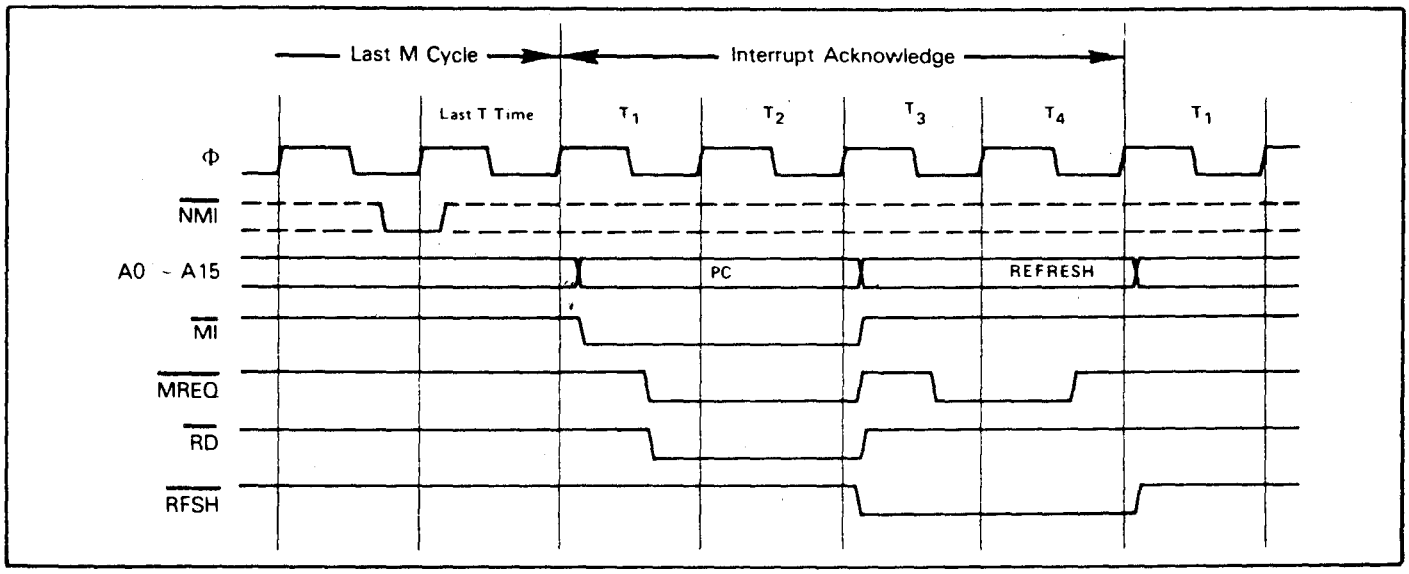


Figure 7-14. Z80 Response to a Nonmaskable Interrupt Request

THE HALT INSTRUCTION

When a Halt instruction is executed by the Z80 CPU, a sequence of NOP instructions is executed until an interrupt request is received. Both maskable and nonmaskable interrupt request lines are sampled on the rising edge of Φ during T_4 of every NOP instruction's machine cycle.

The Halt state will terminate when any interrupt request is detected, at which time the appropriate interrupt acknowledge sequence will be initiated, as illustrated in Figures 7-13 and 7-14.

Note that the Z80 executes the sequence of NOP instructions during a Halt so that it can continue to generate dynamic memory refresh signals.

Halt instruction timing is illustrated in Figure 7-15.

```
INIR        MOV     M,A
            INX     H
            DCR     B
            JNZ     LOOP
```

These instruction sequences input COUNT bytes from I/O port PORTN, and store the data in a memory buffer whose beginning address is START. COUNT and PORTN are symbols representing 8-bit numbers. START is an address label. The block transfer I/O instruction will continue executing until the B register has decremented to 0.

4) **Single Step Block Transfer I/O instructions.** These are identical to the block transfer I/O instructions described in category 3 above, except that instruction execution ceases after one iterative step. Referring to the INIR instruction example, if the INIR instruction were replaced by an INI instruction, a single byte of data would be transferred from PORTN to the memory location addressed by START. The address START would be incremented, Register B contents would be decremented, then instruction execution would cease.

When a block transfer or single step, block transfer I/O instruction is executed, C register contents, which identify the I/O port, are output on the lower eight Address Bus lines in the usual way; however, B register contents are output on the higher eight address lines A15 - A8. Therefore external logic can, if it wishes, determine the extent of the transfer.

Let us now look at the advantages gained by having the new Z80 I/O instructions.

The value of the Register Indirect I/O instructions is that programs stored in ROM can access any I/O port. If I/O port assignments change, then all you need to do is modify that small portion of program which loads the I/O port address into the C register.

The Block Transfer I/O instructions must be approached with an element of caution. In response to the execution of a single instruction's object code, up to 256 bytes of data may be transferred between memory and an I/O port. This data transfer occurs at CPU speed — which means external logic must input or output data at the same speed. If external logic cannot operate fast enough, it can insert Wait states in order to slow the CPU, but that takes additional logic; and one might argue that the traditional methods of polling on status to effect block I/O transfers is cheaper than adding extra Wait state logic.

Note that all Z80 enhanced I/O instructions require two bytes of object code.

PRIMARY MEMORY REFERENCE INSTRUCTIONS

Instructions that we classify as Primary Memory Reference constitute a subset of the Load instructions, as classifed by Zilog. **Within the Primary Memory Reference instructions category, as we define it, Zilog offers a single enhancement: base relative addressing.** Instructions that move data between a register and memory may specify the memory address as the contents of an Index register; plus an 8-bit displacement provided by the instruction object code. Here is a programming example of Zilog base relative addressing and the 8080A equivalent:

```
            Z80                 8080A
LD          IX,BASE     LXI     H,BASE
LD          C,(IX + DISP)   LXI     D,DISP
                        DAD     D
                        MOV     C,M
```

Observe that the two Z80 instructions do not use any CPU registers — other than the IX Index register. The 8080A uses the DE and HL registers. Here is an example of the true value that results from having Index registers. The Z80 can use the DE and HL registers to store temporary data, which the 8080A cannot do; the 8080A would have to store such temporary data in external read/write memory.

The biggest single advantage that accrues to the Z80 from having indexed addressing is the fact that well written Z80 programs will contain far fewer memory reference instructions than equivalent 8080A programs; therefore Z80 programs will execute faster.

Other primary memory reference instructions provided by the Z80, and not present in the 8080A, include instructions which load data into the Index registers and store Index registers' contents in memory. Since the 8080A does not have Index registers, it cannot have memory reference instructions for them. The Z80 also has instructions which transfer 16-bit data between directly addressed memory and any register pair, except AF. Recall that in the 8080A, HL is the only register pair which stores to memory and loads from memory using direct addressing.

BLOCK TRANSFER AND SEARCH INSTRUCTIONS

We classify the Zilog Block Transfer and Search instructions in a separate category, since our hypothetical computer, as described in Volume I, had no equivalent instructions.

A Block Transfer instruction allows you to move up to 65,536 bytes of data between two memory buffers which may be anywhere in memory. The H and L registers address the source buffer, the D and E registers address the destination buffer, and the B and C registers hold the byte count.

After every byte of data is transferred, the B and C registers' contents are decremented; instruction execution ceases after the B and C registers decrement to zero. You have the option of incrementing or decrementing the source and destination addresses following the transfer of each data byte. Thus you can transfer data from low to high memory, or from high to low memory. Here is a programming example of the Z80 Block Move instruction, along with the 8080A equivalent:

```
        Z80                      8080A
LD      BC,COUNT         LXI     B,COUNT
LD      DE,DEST          LXI     D,DEST
LD      HL,SRCE          LXI     H,SRCE
LDIR             LOOP:   MOV     A,M
                         STAX    D
                         INX     H
                         INX     D
                         DCX     B
                         MOV     A,B
                         ORA     C
                         JNZ     LOOP
```

The two instruction sequences illustrated above move a block of data, COUNT bytes long, from a buffer whose starting address is SRCE to another buffer whose starting address is DEST. SRCE and DEST are 16-bit address labels. COUNT is a symbol representing a 16-bit data value.

The Z80 - 8080A comparison above is one that makes the 8080A look particularly bad. This is because it emphasizes 8080A weaknesses; the 8080A requires memory addresses to be incremented as separate steps. Also, after decrementing the counter in Registers B and C, status is not set, therefore BC contents are tested by loading B into A and ORing with C.

You can use Block Move instructions in Z80 configurations that include dynamic memory. While the Block Move is being executed, dynamic memory is refreshed.

The Block Search instruction will search a block of data in memory, looking for a match with the Accumulator contents. The H and L registers address memory, while the B and C registers again act as a byte counter. When a match between Accumulator contents and a memory location is found, the Search instruction ceases executing. After every Compare, the B and C registers' contents are decremented; once again you have the option of either incrementing or decrementing H and L registers' contents. Thus you can search a block of memory from high address down, or from low address up.

The results of every step in a Block Search are reported in the Z and P/O statuses. If a match is found between Accumulator and memory contents, then Z is set to 1; otherwise Z will equal 0. When the B and C registers count out to zero, the P/O status will be reset to 0; otherwise the P/O status will equal 1.

Here is an example of a program using the Z80 Block Search instruction, along with 8080A program equivalent:

```
        Z80                      8080A
        LD      A,REFC           LXI     BC,COUNT
        LD      BC,COUNT         LXI     HL,SRCE
        LD      HL,SRCE  LOOP:   MVI     A,REFC
        CPDR                     CMP     M
        JR      Z,FOUND          JZ      FOUND
;NO MATCH FOUND                  DCX     H
        -                        DCX     B
        -                        MOV     A,B
;MATCH FOUND                     ORA     C
FOUND:                           JNZ     LOOP
        -                ;NO MATCH FOUND
        -                        -
        -                        -
                                 -
                         ;MATCH FOUND
                         FOUND:  -
                                 -
                                 -
```

Each of the above instruction sequences tries to match a character represented by the symbol REFC with the contents of bytes in a memory buffer. The memory buffer is origined at SRCE and is COUNT bytes long.

In the example illustrated above, SRCE is the highest memory address for the buffer, which is searched towards the low memory address. FOUND is the label for the first instruction in the sequence which is executed if a match is found. If no match is found, that is, the BC registers count out to 0, program execution continues with the next sequential instruction.

The Z80 Block Search instruction is particularly useful when searching a large memory buffer for a byte that may frequently occur. Suppose you have an ASCII text in which Control codes have been imbedded. For the sake of argument, let us assume that all Control codes are two bytes long, where the first byte has the hexadecimal value 02 and the second byte identifies the Control code. You can use one set of registers in order to search the text buffer for Control codes, while using the second set of registers to process the text buffer after each Control code has been located.

All you need to do in the Block Search instruction sequence illustrated above is follow the CPDR instruction with an EXX instruction; after executing the instruction sequence following MATCH FOUND, again execute an EXX instruction before returning to search for the next Control code.

Each of the Block Move and Block Search instructions has a single step equivalent. The single step instruction moves one byte of data, or compares the Accumulator contents with the next byte in a data buffer; addresses and counters are incremented and decremented as for the Block Move and Search instructions, however execution ceases after a single step has been completed.

SECONDARY MEMORY REFERENCE (MEMORY OPERATE) INSTRUCTIONS

Instructions that we classify as Secondary Memory Reference, or Memory Operate, constitute a portion of the arithmetic and logical instructions, as defined by the Z80. **Within the Memory Operate group of instructions, the single enhancement offered by the Z80 is a duplicate set of instructions that uses base relative addressing.** We have already discussed this enhancement in connection with Primary Memory Reference instructions. Here is a programming example with the 8080A equivalent:

Z80		8080A	
LD	IX,BASE	LXI	H,BASE
ADD	(IX + DISP)	LXI	D,DISP
		DAD	D
		ADD	M

The same comments we made regarding the use of indexed addressing in the Primary Memory Reference example apply to the instruction sequences above.

IMMEDIATE INSTRUCTIONS

Within the group of instructions that we classify as Immediate, the Z80 offers two enhancements:

1) Instructions are provided to load immediate data into the additional Z80 registers.

2) You can use base relative addressing to load a byte of data immediately into read/write memory.

JUMP INSTRUCTIONS

In addition to the standard Jump instruction offered by the 8080A, the Z80 has a two-byte, unconditional Branch instruction, and two instructions which allow you to jump to the memory location specified by an Index register.

The two indexed Jump instructions transfer the contents of the identified Index register to the Program Counter.

The two-byte Jump instruction interprets the second object code byte as an 8-bit signed binary number, which is added to the Program Counter, after the Program Counter has been incremented to point to the next instruction. This is a standard program relative branch, as described in Volume I.

Note that the Z80 uses many of the spare 8080A object codes to implement the two-byte Branch and Branch-on-Condition instructions. This makes sense; it would certainly not make much sense to have two bytes of object code followed by a single branch byte, since that would create a three-byte Branch instruction — offering no advantage over the three-byte Jump instructions which already exist.

SUBROUTINE CALL AND RETURN INSTRUCTIONS

The Z80 instructions in this group are identical to 8080A equivalents.

IMMEDIATE OPERATE INSTRUCTIONS

Z80 Immediate Operate instructions, as we define them, are identical to those in the 8080A instruction set.

JUMP-ON-CONDITION INSTRUCTION

The Z80 offers two significant Jump-on-Condition instruction enhancements over the 8080A:

1) **There are two-byte equivalents for four of the more commonly used Jump-on-Condition instructions.** The two-byte Jump-on-Condition instructions execute exactly as described for the two-byte Jump instruction.

2) **There is a decrement and Jump-on-Nonzero instruction** which is particularly useful in any kind of iterative loop. When this instruction is executed, the B register contents are decremented; if the B register contents, after being decremented, equal zero, the next sequential instruction is executed. If after being decremented the B register contents are not zero, then a Jump occurs. This is a two-byte instruction, where the Jump is specified by a single 8-bit signed binary value.

Here is an example of how the DJNZ instruction may be used along with the 8080A equivalent:

	Z80			8080A	
	AND	A		ANA	A
	LD	IX,VALA		LXI	D,VALA
	LD	IY,VALB		LXI	H,VALB
	LD	B,CNT		MVI	B,CNT
LOOP:	LD	A,(IX)	LOOP:	LDAX	D
	ADC	A,(IY)		ADC	M
	LD	(IX),A		STAX	D
	INC	IX		INX	D
	INC	IY		INX	H
	DJNZ	LOOP		DCR	B
				JNZ	LOOP

The two instruction sequences illustrated above perform simple multibyte binary addition. The contents of two buffers, origined at VALA and VALB, are summed; the results are stored in buffer VALA.

The first instruction in each sequence is executed in order to clear the Carry status. Like the 8080A, the Z80 does not have an instruction which sets the Carry status to 0, while performing no other operation.

REGISTER-REGISTER MOVE INSTRUCTIONS

Register-Register Move instructions, as we defined them in this book, constitute a subset of the Z80 Load instructions. All Z80 Exchange instructions, except those that exchange with the top of the Stack, are also classified as Register-Register Move instructions.

The Z80 enhancements within this instruction group apply strictly to the additional registers implemented within the Z80. That is to say, because the Z80 has registers which the 8080A does not have, the Z80 must also have instructions to move data in and out of these additional registers.

The instructions which exchange data between registers and their alternates need comment. Note that you can swap the entire set of duplicated registers, or you can swap selected register pairs. If you use these instructions following an interrupt acknowledge, you do not have to save the contents of the registers on the Stack. Of course, this will only work for a single interrupt level. There are also occasions when the alternate set of registers can be used effectively in normal programming logic, as we illustrated when describing the Block Search instruction.

REGISTER-REGISTER OPERATE INSTRUCTIONS

There are a few new Z80 Register-Register Operate instructions which do the following:

1) Add without Carry the contents of a register pair to an Index register.

2) Add with Carry to HL the contents of a register pair.

3) Subtract with Carry from HL the contents of a register pair.

REGISTER OPERATE INSTRUCTIONS

Within this category, the Z80 has two enhancements:

1) You can increment or decrement the contents of an Index register.

2) A rich variety of Shift and Rotate instructions have been added. These instructions are illustrated in Table 7-2. In particular, note the RLD and RRD instructions, which are very useful when performing multidigit BCD left and right shifts.

IT MANIPULATION INSTRUCTIONS

he 8080A has no equivalent for this set of Z80 instructions. We give these instructions a separate category in Table 7-2 because of their extreme importance in microprocessor applications.

it manipulation instructions are particularly important for signal processing. A single signal is a binary entity; it is not art of an 8-bit unit. One of the great oversights among microprocessor designers has been to ignore bit manipulation nstructions. **The Z80 has instructions that set to 1 (SET), reset to 0 (RES) or test (BIT) individual bits in memory r any general purpose register.** The result of a bit test is reported in the Zero status.

ere are some Z80 instructions with 8080A equivalents:

Z80		8080A	
BIT	4,A	MOV	B,A
		ANI	10H
		MOV	A,B

he 8080A tests Accumulator bits destructively — all untested bits are cleared; Accumulator contents must therefore e saved before testing. We can also contrive an example to emphasize the strengths of the Z80 bit instructions:

Z80		8080A	
LD	IY,BASE	LXI	H,BASE
SET	2,(IY + DISP)	LXI	D,DISP
		DAD	D
		MVI	A,4
		ORA	M

)nce again, note that the 8080A needs to use the D, E, H and L registers.

lote that all Z80 Bit instructions operate on memory or CPU registers. But in most microcomputer applications individual pins at I/O ports will most frequently be set, reset or tested. The Z80 has no I/O Bit instructions. If you wish, ou can interface I/O devices so that they are addressed as memory locations; however, in that case, you cannot use lock I/O instructions.

he 8080A can do anything that a Z80 Bit Manipulation instruction can do but an additional Mask instruction is eeded and the Accumulator is involved. On the surface these seem to be small penalties; but it is the frequency with vhich Bit Manipulation instructions are needed that escalates small penalties into major aggravations.

STACK INSTRUCTIONS

Additional Stack instructions provided by the Z80 allow the Z80 Index registers to be pushed onto the Stack, opped from the Stack, or exchanged with the top of the Stack.

NTERRUPT INSTRUCTIONS

n addition to the 8080A Interrupt instructions, the Z80 has two Return-from-Interrupt instructions. **RETI and RETN are sed to return from maskable and nonmaskable interrupt service routines, respectively.**

ETI and RETN are two-byte instructions. **Within the CPU these instructions enable interrupts, but otherwise execute exactly as a Return-from-Subroutine (RET) instruction. However, devices designed by Zilog to support he Z80 CPU use the RETI and RETN instructions in a unique way.** Any support device that has logic to request an nterrupt also includes logic which tests the Data Bus contents during the low $\overline{M1}$ pulse. Upon detecting the second yte of an RETI or RETN instruction's object code, a device which has had an interrupt request acknowledged determines that the interrupt has been serviced.

Vhy does a support device need to know that an interrupt service routine has completed execution? The reason is that ilog extends interrupt priority arbitration logic beyond the interrupt acknowledge process to the entire interrupt service routine.

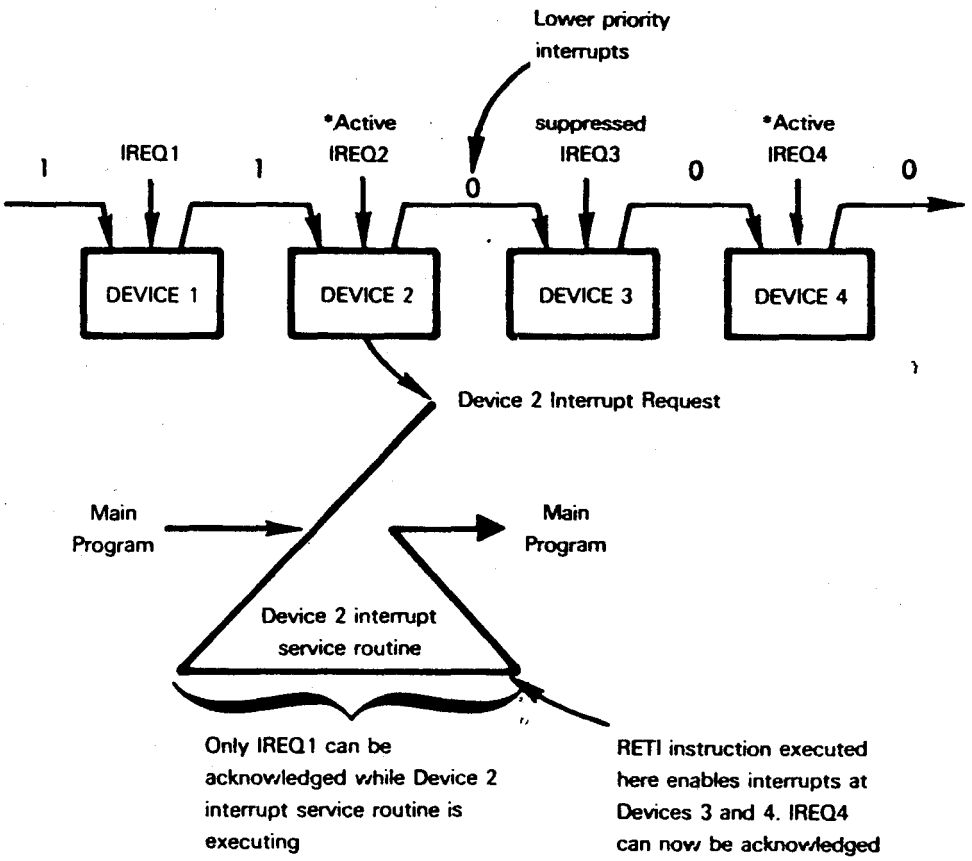
his is the scheme adopted by the 8259 PICU. After reading the next paragraph, if you are still unclear on concepts, efer to the 8259 PICU discussion in the 8080A chapter.

onsider the typical daisy chain scheme used to set interrupt priorities in a multiple interrupt microcomputer system. aisy chaining has been described in good detail in Volume 1. When more than one device is requesting an interrupt, n acknowledge ripples down the daisy chain until trapped by the interrupt requesting device electrically closest to the PU. As soon as the interrupt acknowledge process has ceased, an interrupt service routine is executed for the cknowledged interrupt; acknowledged external logic will now remove its interrupt request. Unless the CPU disables urther interrupts, a lower priority device can immediately interrupt the service routine of a higher priority device. With ne Zilog system, that is not the case. A device which has its interrupt request acknowledged continues to suppress interrupt requests from all lower priority devices in a daisy chain, until the second object code byte for an RETI or RETN

instruction is detected on the Data Bus. The acknowledged device responds to an RETI or RETN instruction's object code by re-enabling interrupts for devices with lower priority in the daisy chain.

Providing a Zilog microcomputer system has been designed to make correct use of the RETI and RETN instructions, interrupt priority arbitration logic will allow an interrupt service routine to be interrupted only by a high priority interrupt request.

Here is an illustration of the Zilog interrupt priority arbitration scheme:



The three IM instructions allow you to specify that the CPU will respond to maskable interrupts in Mode 0, 1 or 2. These three interrupt response modes have already been described.

STATUS AND MISCELLANEOUS INSTRUCTIONS

Z80 and 8080A instructions in these categories are identical.

THE BENCHMARK PROGRAM

Our benchmark program is coded for the Z80 as follows:

```
LD      BC,LENGTH     ;LOAD IO BUFFER LENGTH INTO BC
LD      DE,(TABLE)    ;LOAD ADDRESS OF FIRST FREE TABLE BYTE OUT OF FIRST TWO TABLE
                      ;BYTES
LD      HL,IOBUF      ;LOAD SOURCE ADDRESS INTO HL
LDIR                  ;EXECUTE BLOCK MOVE
```

The program above makes absolutely no assumptions. Both source and destination tables may have any length and may be located anywhere in memory.

Notice that there is no instruction execution loop, since the LDIR block move will not stop executing until the entire block of data has been moved.

e following abbreviations are used in this chapter:

',B,C,D,E,H,L	The 8-bit registers. A is the Accumulator and F is the Program Status Word.
',BC',DE',HL'	The alternative register pairs
dr	A 16-bit memory address
,)	Bit b of 8-bit register or memory location x
nd	Condition for program branching. Conditions are: NZ - Non-Zero (Z=0) Z - Zero (Z=1) NC - Non-carry (C=0) C - Carry (C=1) PO - Parity Odd (P=0) PE - Parity Even (P=1) P - Sign Positive (S=0) M - Sign Negative (S=1)
ta	An 8-bit binary data unit
ta16	A 16-bit binary data unit
;p	An 8-bit signed binary address displacement
(HI)	The high-order 8 bits of a 16-bit quantity xx
	Interrupt vector register (8 bits)
IY	The Index registers (16 bits each)
	Either one of the Index registers (IX or IY)
B	Least Significant Bit (Bit 0)
,el	A 16-bit instruction memory address
(LO)	The low-order 8 bits of a 16-bit quantity xx
SB	Most Significant Bit (Bit 7)
	Program Counter
rt	An 8-bit I/O port address
	Any of the following register pairs: BC DE HL AF
	The Refresh register (8 bits)
g	Any of the following registers: A B C D E H L
	Any of the following register pairs: BC DE HL SP
	Stack Pointer (16 bits)

Table 7-4. Z80 PIO Interpretation of Control Signals

SIGNALS			FUNCTIONAL INTERPRETATION *
$\overline{M1}$	$\overline{IORQ}$	$\overline{RD}$	
0	0	0	No function
0	0	1	Interrupt acknowledge
0	1	0	Check for end of interrupt service routine
0	1	1	Reset
1	0	0	Read from PIO to CPU
1	0	1	Write from CPU to PIO
1	1	0	No function
1	1	1	No function

* These interpretations only apply if the device has been selected

Z80 support devices also rely on exact Z80 CPU characteristics for interrupt processing. Specifically, Z80 support devices detect every instruction fetch, as identified by $\overline{M1}$ and $\overline{RD}$ simultaneously low; if a return from interrupt object code is fetched, then Z80 support devices respond to this object code by resetting internal interrupt priority logic. Accounting for this end of interrupt logic in a non-Z80 system could be difficult.

Because of the unique characteristics of the Z80 support devices, the Z80 PIO and CTC devices are described in this chapter. The Z80 DMA device is described in Volume 3, however, because this device is easily used in non-Z80 configurations; moreover, its unique capabilities make it a highly desirable part to include in any microcomputer system that has to move text or data strings. **The Z80 SIO device is also described in Volume 3** because it is an exceptionally powerful device; in many cases the power of the Z80 SIO device will compensate for the additional logic it will demand in a non-Z80 microcomputer system.

THE Z80 PARALLEL I/O INTERFACE (PIO)

The Z80 PIO is Zilog's parallel interface device; it may be looked upon as a replacement for the 8255 PPI, but it is equivalent to the PPI at a functional level only. No attempt has been made to make the Z80 PIO an upward compatible replacement for the 8255 PPI.

The Z80 PIO has 16 I/O pins, divided into two 8-bit I/O ports. Each I/O port has two associated control lines. This makes the Z80 PIO more like the Motorola MC6820 than the 8255 PPI.

The two Z80 PIO I/O ports may be separately specified as input, output or control ports. When specified as a control port, pins may be individually assigned to input or output. Port A may be used as a bidirectional I/O port.

The Z80 PIO also provides a significant interrupt handling capability. This includes:

The ability to define conditions which will initiate an interrupt.

Interrupt priority arbitration

Vectored response to an interrupt acknowledge

Figure 7-16 illustrates that part of our general microcomputer system logic which has been implemented on the Z80 PIO.

The Z80 PIO is packaged as a 40-pin DIP. It uses a single +5V power supply. All inputs and outputs are TTL-level compatible. The device is fabricated using N-channel silicon gate depletion load technology.

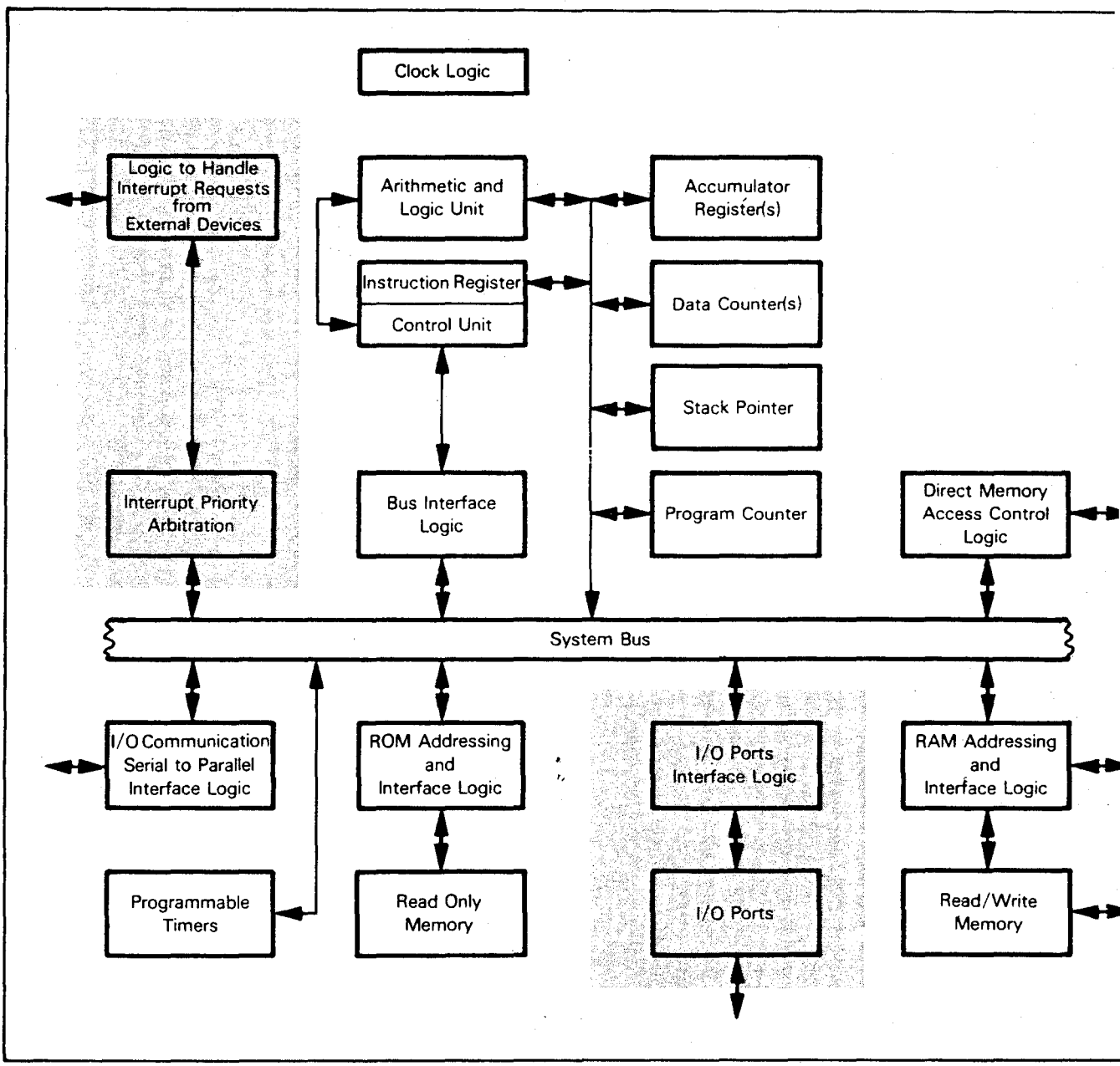


Figure 7-16. Logic Functions of the Z80 PIO

80 PIO PINS AND SIGNALS

80 PIO pins and signals are illustrated in Figure 7-17. Signals are very straightforward; therefore their functions ll be summarized before we discuss device characteristics and operation.

t us first consider the PIO CPU interface.

l data transfers between the PIO and the CPU occur via the Data Bus, which connects to pins D0 - D7.

r the PIO to be selected, a low input must be present at $\overline{CE}$. There are two additional address lines. **B/$\overline{A}$ SEL lects Port A if low and Port B if high. For the selected I/O port, C/$\overline{D}$ SEL selects a data buffer when low and a ntrol buffer when high.** Device select logic is summarized in Table 7-5.

Table 7-5. Z80 PIO Select Logic

SIGNAL			SELECTED LOCATION
$\overline{CE}$	B/$\overline{A}$ SEL	C/$\overline{D}$ SEL	
0	0	0	Port A data buffer
0	0	1	Port A control buffer
0	1	0	Port B data buffer
0	1	1	Port B control buffer
1	X	X	Device not selected

0 PIO device control logic is not straightforward. Of the control signals output by the Z80 CPU, three are input to the): $\overline{M1}$, $\overline{IORQ}$, and $\overline{RD}$. $\overline{WR}$ is not input to the PIO. **Table 7-5 illustrates the way in which Z80 PIO interprets $\overline{M1}$, $\overline{RQ}$ and $\overline{RD}$.** Observe that $\overline{RD}$ is being treated as a signal with two active states: low $\overline{RD}$ specifies a read operation, ereas high $\overline{RD}$ specifies a write operation. This does not conform to the CPU, which treats $\overline{RD}$ and $\overline{WR}$ as signals th a low active state only.

t us now look at the PIO external logic interface.

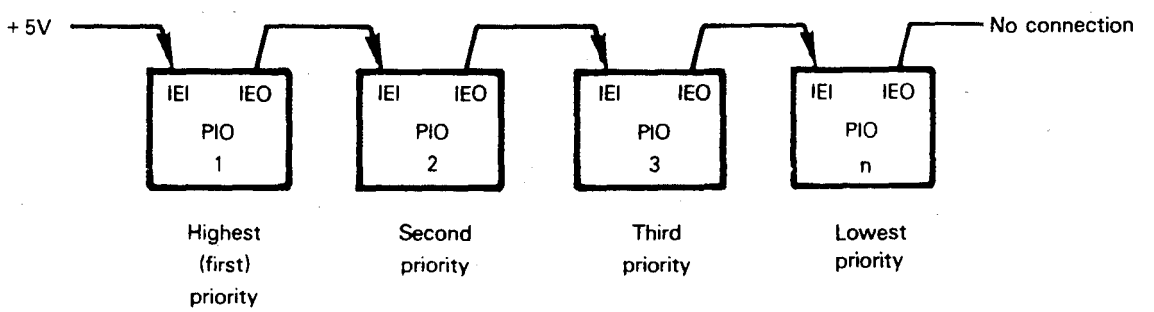
- A7 represent the eight bidirectional I/O Port A lines; I/O Port A is supported by two control signals, A RDY d $\overline{A\ STB}$.

nilarly, I/O Port B is implemented via the eight bidirectional lines B0 - B7 and the two associated control lines RDY and $\overline{B\ STB}$.

e I/O Port A and B control lines provide handshaking logic which we will describe shortly.

w consider interrupt control signals.

and IEO are standard daisy chain interrupt priority signals. When more than one PIO is present in a system, the hest priority PIO will have IEI tied to +5V and will connect its IEO to the IEI for the next highest priority PIO in the sy chain:

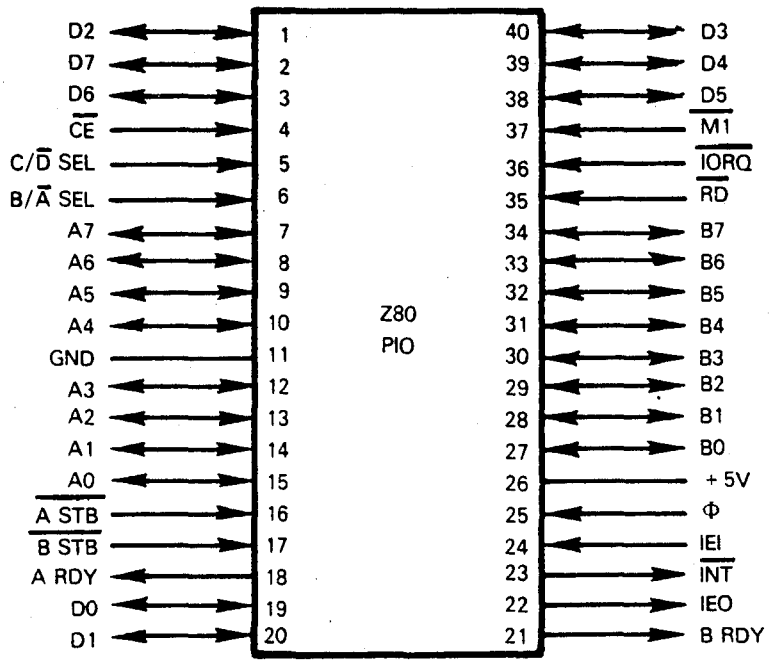


ou are unsure of daisy chain priority networks, refer to Volume 1 for clarification.

is a standard interrupt request signal which is output by the Z80 PIO and must be connected as an input to the CPU interrupt request. Observe that there is no interrupt acknowledge line, since $\overline{M1}$ and $\overline{IORQ}$ simultaneously low stitute an interrupt acknowledge and will thus be decoded by the Z80 PIO.

ck, power, and ground signals are absolutely standard. The same clock signal is used by the PIO and the Z80 J.

erve that there is no Reset signal to the PIO. $\overline{M1}$ low with both $\overline{RD}$ and $\overline{IORQ}$ high constitutes a reset. We will cribe the effect of a Z80 PIO reset after discussing operating modes.

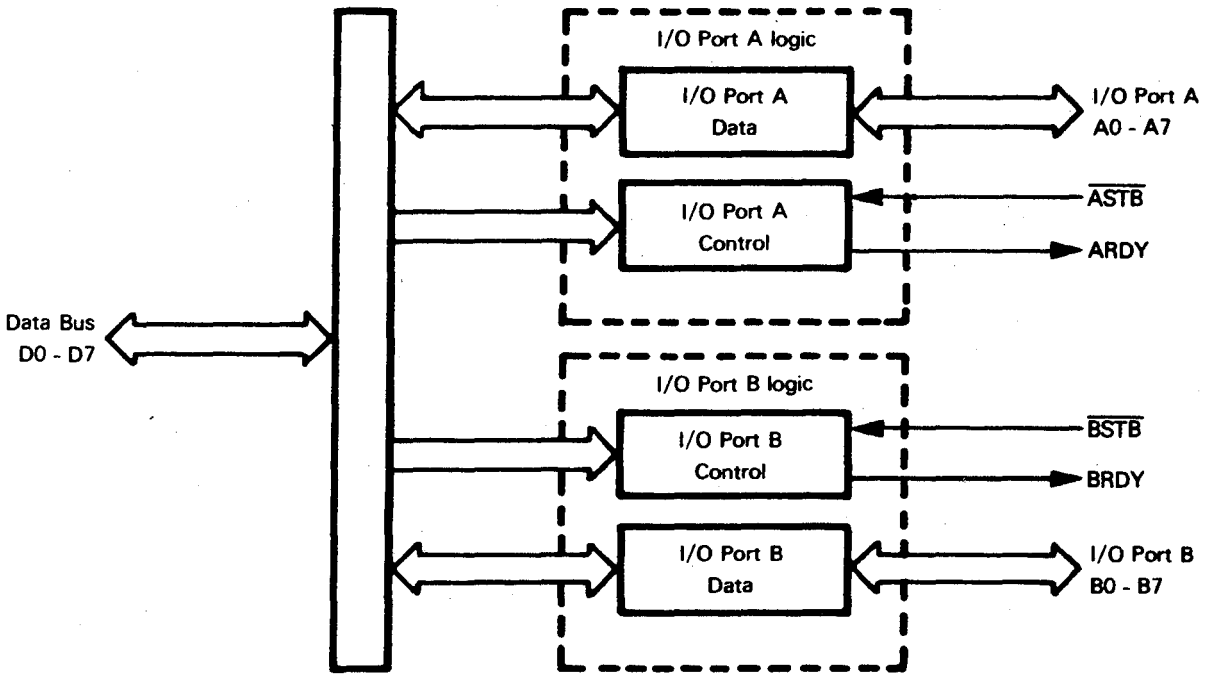


PIN NAME	DESCRIPTION	TYPE
D0 - D7	Data Bus	Tristate, Bidirectional
CE	Device Enable	Input
B/A SEL	Select Port A or Port B	Input
C/D SEL	Select Control or Data	Input
M1	Instruction fetch machine cycle signal from CPU	Input
IORQ	Input/Output request from CPU	Input
RD	Read cycle status from CPU	Input
A0 - A7	Port A Bus	Tristate, Bidirectional
A RDY	Register A Ready	Output
A STB	Port A strobe pulse	Input
B0 - B7	Port B Bus	Tristate, Bidirectional
B RDY	Register B Ready	Output
B STB	Port B strobe pulse	Input
IEI	Interrupt enable in	Input
IEO	Interrupt enable out	Output
INT	Interrupt request	Output, Open-drain
Φ, + 5V,GND	Clock, Power and Ground	

Figure 7-17. Z80 PIO Signals and Pin Assignments

Z80 PIO OPERATING MODES

To the programmer, a Z80 PIO will be accessed as four addressable locations:



By loading appropriate information into the Control register you determine the mode in which the I/O port is to operate.

The Z80 PIO has operating modes which are equivalent to those of the 8255 PPI, plus an additional mode which the 8255 PPI does not have. However, 8255 PPI Mode 0 provides 24 I/O lines, as against a maximum of 16 I/O lines available with the Z80 PIO.

Zilog literature uses Mode 0, Mode 1, Mode 2, and Mode 3 to describe the ways in which the Z80 PIO can operate; in order to avoid confusion between mode designations as used by the Z80 PIO and the 8255 PPI, mode equivalences are given in Table 7-6.

Table 7-6. Z80 PIO And 8255 Mode Equivalences

Z80 PIO	8255 PPI	INTERPRETATION
Mode 3*	Mode 0	Simple input or output
Mode 0	Mode 1	Output with handshaking
Mode 1	Mode 1	Input with handshaking
Mode 2	Mode 2	Bidirectional I/O with handshaking
Mode 3	None	Port pins individually assigned as controls

*Special case of Mode 3

Let us now look at the Z80 PIO modes in more detail.

Output mode (Mode 0) allows Port A and/or Port B to be used as a conduit for transferring data to external logic. Figure 7-18 illustrates timing for Mode 0. An output cycle is initiated when the CPU executes any Output instruction accessing the I/O port. The Z80 PIO does not receive the $\overline{WR}$ pulse from the CPU, therefore it derives an equivalent signal by ANDing $RD \cdot \overline{CE} \cdot \overline{C/D} \cdot \overline{IORQ}$.

This pseudo write pulse ($\overline{WR}$* in Figure 7-18) is used to strobe data off the Data Bus and into the addressed I/O port's Output register. After the pseudo write pulse goes high, on the next high-to-low transition of the clock pulse Φ, the RDY control signal is output high to external logic. RDY remains high until external logic returns a low pulse on the $\overline{STB}$ acknowledge. On the following high-to-low clock pulse Φ transition, RDY returns low. The low-to-high $\overline{STB}$ transition also generates an interrupt request.

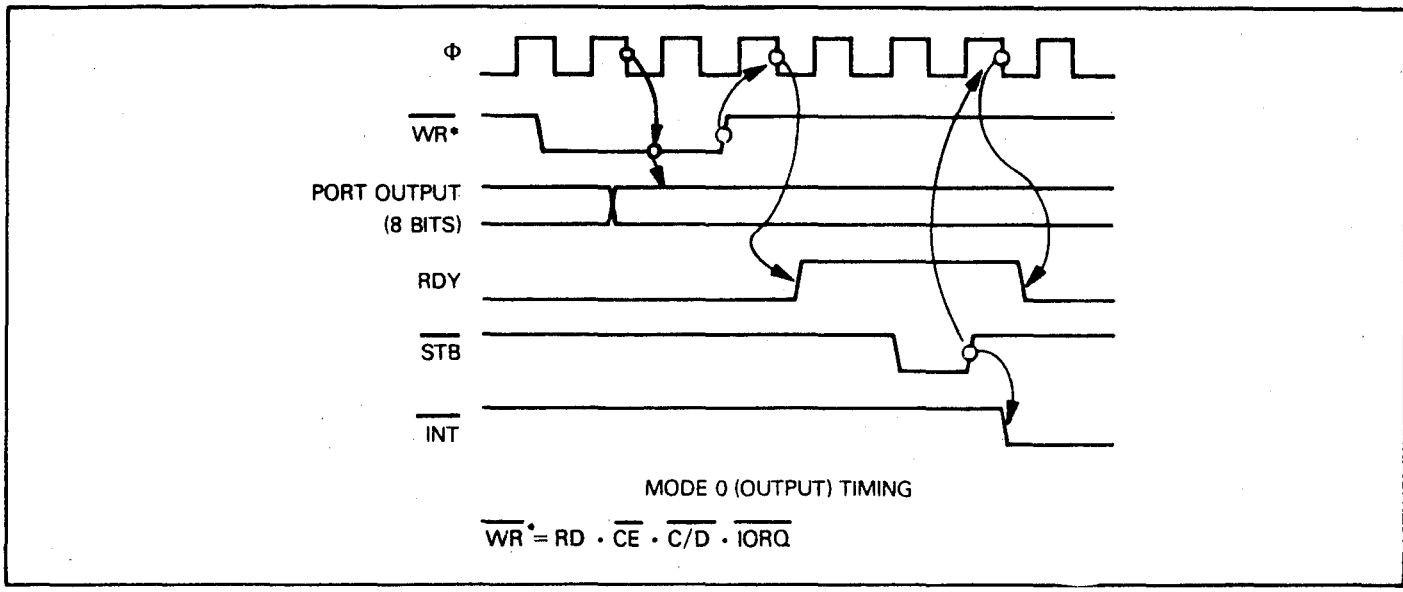
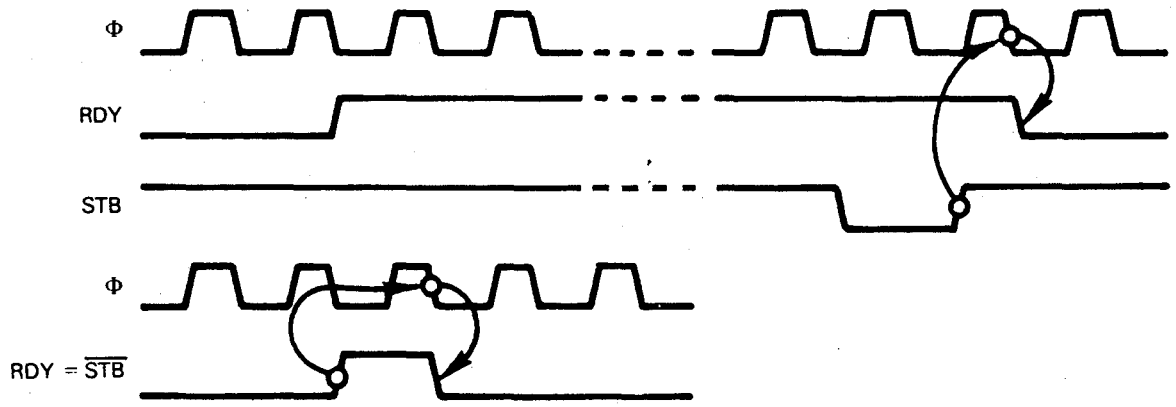


Figure 7-18. Mode 0 (Output) Timing

The RDY and STB signal transition logic has been designed to let RDY create STB. If you connect these two signals, the RDY low-to-high transition becomes the STB low-to-high transition and RDY is strobed high for one clock pulse only. This may be illustrated as follows:



Timing for input mode (Mode 1) is illustrated in Figure 7-19. External logic initiates an input cycle by pulsing STB low. This low pulse causes the Z80 PIO to load data from the I/O port pins into the port Input register. On the rising edge of the STB pulse an interrupt request will be triggered.

On the falling edge of the Φ clock pulse which follows STB input high, RDY will be output low informing external logic that its data has been received but has not yet been read. RDY will remain low until the CPU has read the data, at which time RDY will be returned high.

It is up to external logic to ensure that data is not input to the Z80 PIO while RDY is low. If external logic does input data to the Z80 PIO while RDY is low, then the previous data will be overwritten and lost — and no error status will be reported.

In bidirectional mode (Mode 2), the control lines supporting I/O Ports A and B are both applied to bidirectional data being transferred via Port A; Port B must be set to bit control (Mode 3).

Figure 7-20 illustrates timing for bidirectional data transfers. This figure is simply a combination of Figures 7-18 and 7-19 where the A control lines apply to data output while the B control lines apply to data input. The only unique feature of Figure 7-20 is that bidirectional data being output via Port A is stable only for the duration of the A STB low pulse. This is necessary in bidirectional mode since the Port A pins must be ready to receive input data as soon as the output operation has been completed.

Once again, it is up to external logic to make sure that it conforms with the timing requirements of bidirectional mode operation. External logic must read output data while A STB is low. If external logic does not read data at this time, the data will not be read and the Z80 PIO will not report an error status to the CPU; there is no signal that external logic sends back to the Z80 PIO following a successful read.

Also, it is up to external logic to make sure that it transmits data to Port A only while B RDY is high and A RDY is low. If external logic tries to input data while the Z80 PIO is outputting data, input data will not be accepted. If external logic tries to input data before previously input data has been read, the previously input data will be lost and no error status will be reported.

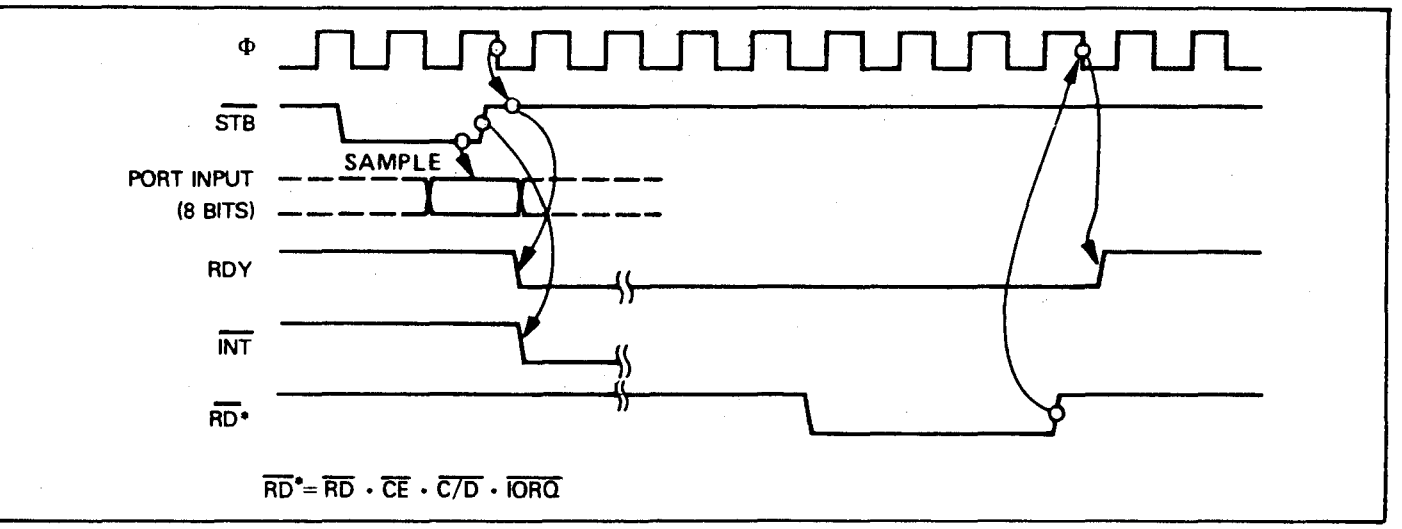


Figure 7-19. Mode 1 (Input) Timing

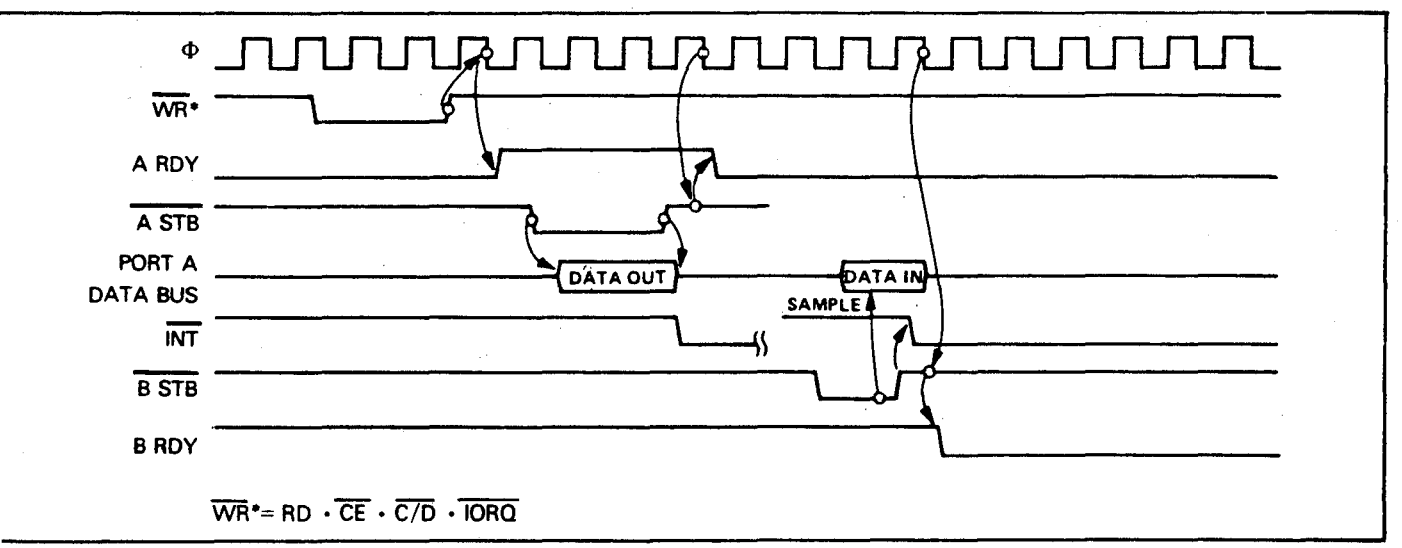


Figure 7-20. Port A, Mode 2 (Bidirectional) Timing

ontrol mode (Mode 3) does not use control signals. You must define every pin of an I/O port in Mode 3 as an in-ut or an output pin. The section on programming the Z80 PIO explains how to do this. Timing associated with the ac-al transfer of data at a single pin is as illustrated in Figures 7-18 and 7-19, ignoring the RDY and STB signals. If all the ns of a single port are defined in the same direction, then that port can be used for simple parallel input or output ithout handshaking).

80 PIO INTERRUPT SERVICING

ie Z80 PIO has a single interrupt request line via which it transmits interrupt requests to the CPU.

n interrupt request can originate from I/O Port A logic, or from I/O Port B logic. In the case of simultaneous in-rrupt requests, I/O Port A logic has higher priority.

n interrupt request may be created in one of two ways. We have already seen in our discussion of Modes 0, 1 and 2 at appropriate control signal transitions will activate the interrupt request line; that is the first way in which an inter-pt request may occur. In Mode 3 you can program either I/O port to generate an interrupt request based on the status signals at individual I/O port pins; you can specify which I/O port pins will contribute to interrupt request logic and nat the pin states must be for the interrupt request to occur. In a microcomputer system that has more than one Z80 D, interrupt priorities are arbitrated using daisy chain logic as we have already described. But there is a significant fference between priority arbitration within a Z80 system as compared to typical priority arbitration. Figure 7-21 il-strates interrupt acknowledge timing.

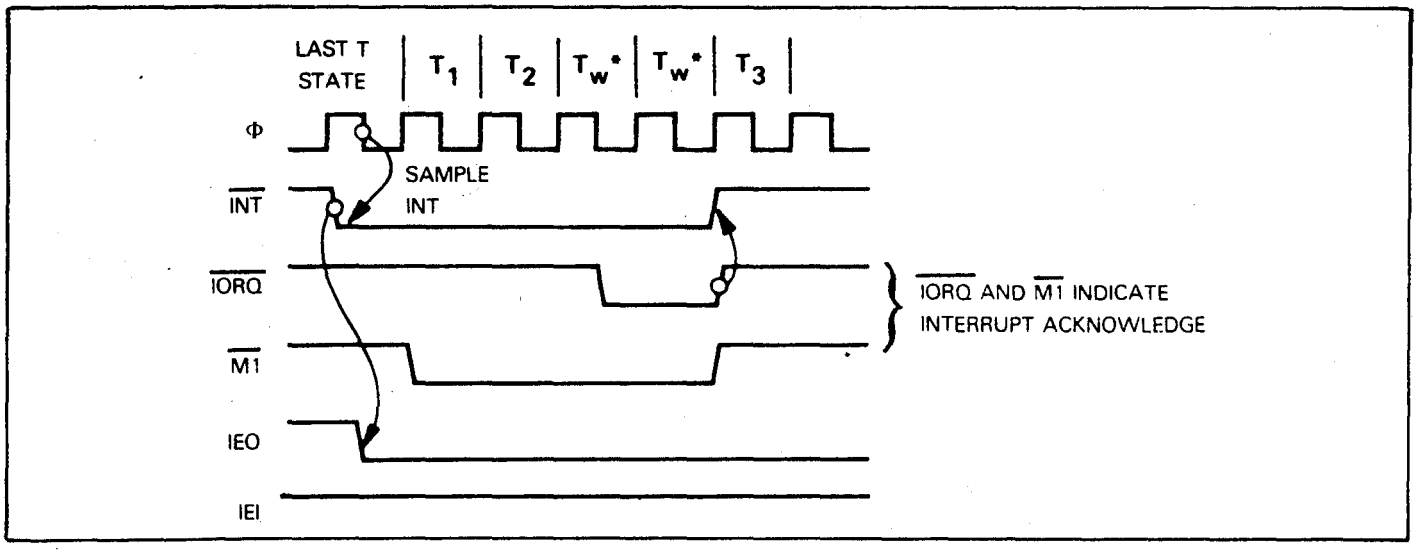


Figure 7-21. Interrupt Acknowledge Timing

The Z80 PIO requires the CPU to execute an RETI instruction upon concluding an interrupt service routine. Following an interrupt, an acknowledged Z80 PIO continously scans the Data Bus whenever $\overline{M1}$ is pulsed low. Until an RETI instruction's object code is detected, the acknowledged Z80 PIO will continuously output IEO low, thus disabling all lower priority Z80 PIOs. As soon as an RETI instruction's object code is detected on the Data Bus, the Z80 PIO will output IEO high, thus enabling lower priority Z80 PIOs. What this means is that interrupt priorities extend to the interrupt service routine as well as the interrupt request arbitration logic. Once an interrupt has been acknowledged, all lower priority interrupt requests will be denied until the acknowledged interrupt service routine has completed execution and has executed an RETI instruction. However, higher priority interrupts can be acknowledged and in turn interrupt an executing service routine. This is identical to the priority arbitration logic which we described for the 8259 PICU.

You can, if you wish, enable lower priority interrupts by executing an RETI instruction before an interrupt service routine has completed execution. But this requires that you execute an RETI instruction in order to return from a subroutine within the interrupted service routine. This instruction sequence may be illustrated as follows:

```
;START OF INTERRUPT SERVICE ROUTINE
        -
        -
        -
        CALL    ENABLE      ;ENABLE ALL INTERRUPTS AT PIO DEVICES
        -
        -
        -
        RET                 ;END OF INTERRUPT SERVICE ROUTINE
ENABLE  RETI
```

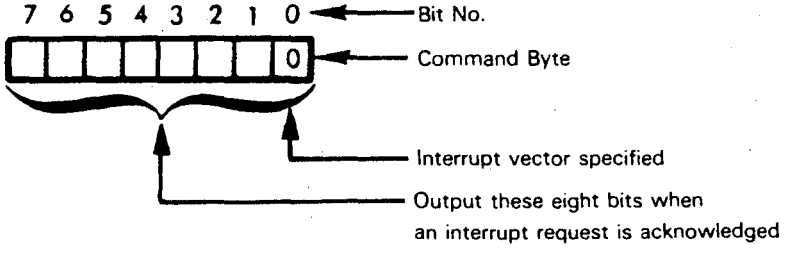
If you simply executed an RETI instruction shortly after entering an interrupt service routine, you would make a hasty exit from the routine — before completing the tasks that have to be performed in response to the acknowledged interrupt.

PROGRAMMING THE Z80 PIO

You program the Z80 PIO by outputting a series of commands.

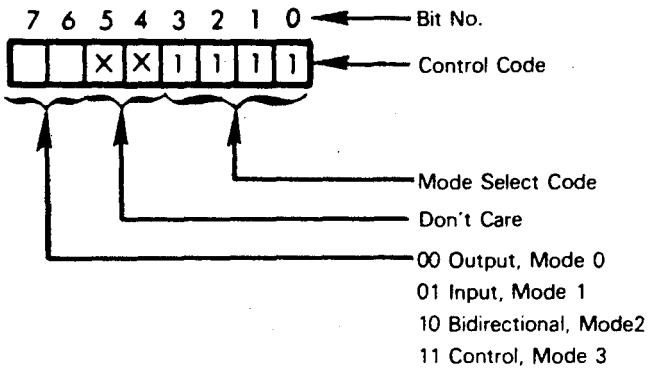
Let us start by identifying command format.

If the 0 bit of a command is low, then the receiving I/O port logic will interpret the command as an interrupt vector, with which it must respond to an interrupt acknowledge, assuming that the CPU is operating in interrupt Mode 2

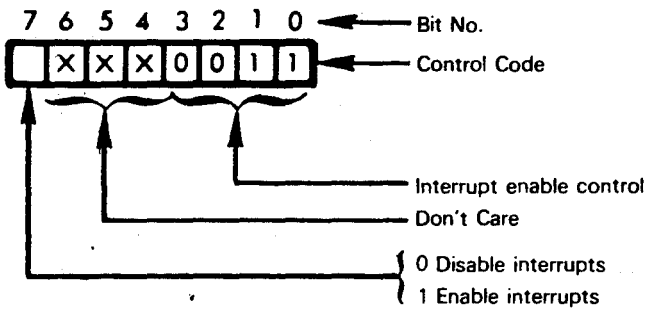


Do not confuse CPU interrupt modes with I/O port modes; they have nothing in common.

In order to define an I/O port's mode you must output a Control code to the I/O port's Control buffer. This is the Control code format:



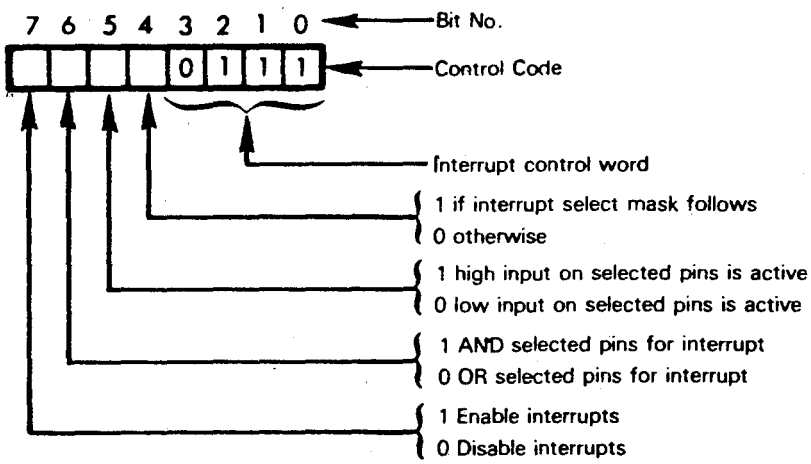
Observe that the same address, the I/O Port A or B Control buffer address, is used when outputting a Control code, an interrupt vector, or a mode select. The low-order four bits of the Control code determine the way in which the Control code will be interpreted. **The following Control code will enable or disable interrupts:**



If a Mode Select Control code is output specifying that an I/O port will operate in Mode 3, then the next byte output is assumed to be a pin direction mask. 1 identifies an input pin, whereas 0 identifies an output pin. Here is a sample instruction sequence:

```
LD      C,(PORTAC)      ;LOAD PORT A CONTROL ADDRESS INTO REGISTER C
LD      A,0CFH          ;LOAD MODE 3 SELECT INTO ACCUMULATOR
OUT     (C),A           ;OUTPUT TO PORT A CONTROL REGISTER
LD      A,3AH           ;DEFINE PINS 5, 4, 3 AND 1 AS INPUTS,
OUT     (C),A           ;PINS 7, 6, 2 AND 0 AS OUTPUTS
```

If you set an I/O port to Mode 3, then you can define the conditions which will cause an interrupt request; you do this by outputting the following interrupt Control code:



When you output an interrupt Control code, as illustrated above, if bit 4 is 1, Z80 PIO logic will assume that the next Control code output is an interrupt mask. An interrupt mask selects the pins that will contribute to interrupt request logic. A 0 bit selects a pin, while a 1 bit deselects the pin.

Combining the various Control codes that have been described we can now illustrate a typical sequence of instructions for accessing a Z80 PIO. Assume that PIO I/O port addresses are:

Port A data	4
Port A command	5
Port B data	6
Port B command	7

We are going to set I/O Port B to Mode 3, with an interrupt request triggered by either pin 6, 3 or 2 high. Pins 6, 3, 2 and 1 will be input pins, while pins 7, 5, 4 and 0 are outputs. The Port B interrupt vector will be 04. Port A will be a bidirectional I/O port with an interrupt vector of 02. Here is the initialization instruction sequence:

```
LD      A,8FH       ;SET PORT A TO MODE 2
OUT     (5),A
LD      A,2         ;OUTPUT INTERRUPT VECTOR
OUT     (5),A
LD      C,7         ;SET PORT B ADDRESS IN C
LD      A,0CFH      ;SET PORT B TO MODE 3
OUT     (C),A
LD      A,4EH       ;OUTPUT PIN DIRECTION MASK
OUT     (C),A
LD      A,4         ;OUTPUT INTERRUPT VECTOR
OUT     (C),A
LD      A,0B7H      ;OUTPUT INTERRUPT CONTROL WORD
OUT     (C),A
LD      A,0B3H      ;OUTPUT INTERRUPT MASK
OUT     (C),A
```

THE Z80 CLOCK TIMER CIRCUIT (CTC)

The Z80 Clock Timer Circuit is a programmable device which contains four sets of timing logic. Each set of timing logic can be programmed independently as an interval timer or an external event counter.

The master Z80 system clock is used by interval timer logic. A time out may be identified by an interrupt request.

An external signal is used to trigger decrement logic when the timer is functioning as an event counter. An interrupt may be requested when the predetermined number of events count out.

If you compare the Z80 CTC with the 8253 Counter/Timer described in Volume 3, **you will see that the Z80 CTC has four sets of counter/timer logic as compared to the three sets of the 8253;** however, the 8253 has more programmable options. In addition to functioning as an event counter or an interval timer, the 8253 can be programmed to generate a variety of square waves and pulse output signals.

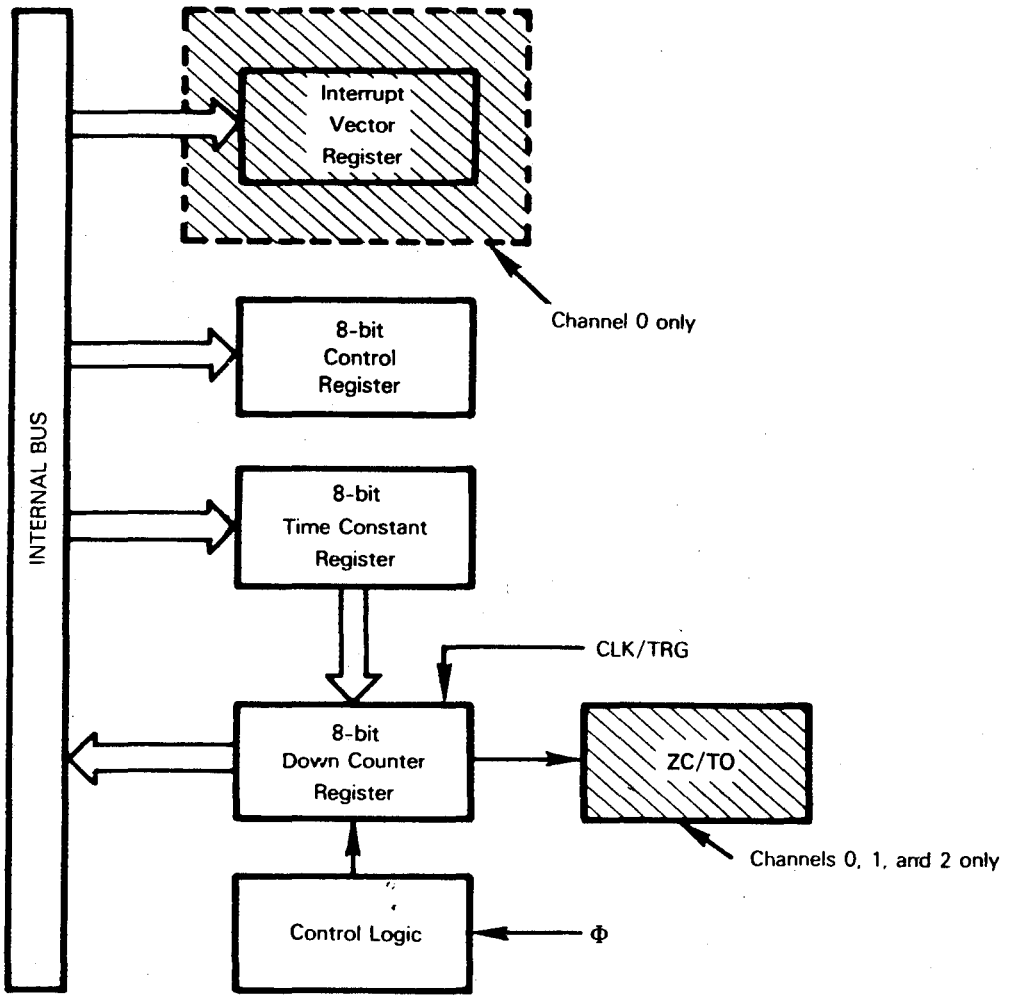
The Z80 CTC is fabricated using N-channel depletion load technology. It is packaged as a 28-pin DIP. All pins are TTL-level compatible.

Z80 CTC FUNCTIONAL ORGANIZATION

Before we examine pins, signals, and operating characterics of the Z80 CTC in detail, let us take an overall look at device logic.

There are four counter/timer logic elements in a Z80 CTC; each is referred to as a "channel".

Each of the four counter/timer channels may be visualized as consisting of three 8-bit registers and two control signals. This may be illustrated as follows:



n initial counter or timer constant is loaded into the Time Constant register. The value in the Time Constant gister is maintained unaltered until you write a new value into this register.

he initial Timer Constant is loaded into the Down Counter register at the beginning of a counter or timer operaon; the contents of the Down Counter register are decremented. You can at any time read the contents of the Down ounter register in order to determine how far a time interval or event counting sequence has progressed.

he Channel Control register contains a Control code which defines the channel's programmable options. There e four Control registers, one for each of the four channels. Thus one channel's operations in no way influence operaons for any other channel.

here is an Interrupt Vector register which is addressed as though it were part of channel 0 logic. This register ontains the address which is transmitted by the Z80 CTC upon receiving an interrupt acknowledge. The Z80 TC assumes that the Z80 CPU is operating in Interrupt mode 2 — in which mode the device requesting an interrupt sponds to an acknowledge by providing the second byte of a subroutine address which the CPU will Call. For details fer to our earlier discussion of the Z80 CPU.

80 CTC PINS AND SIGNALS

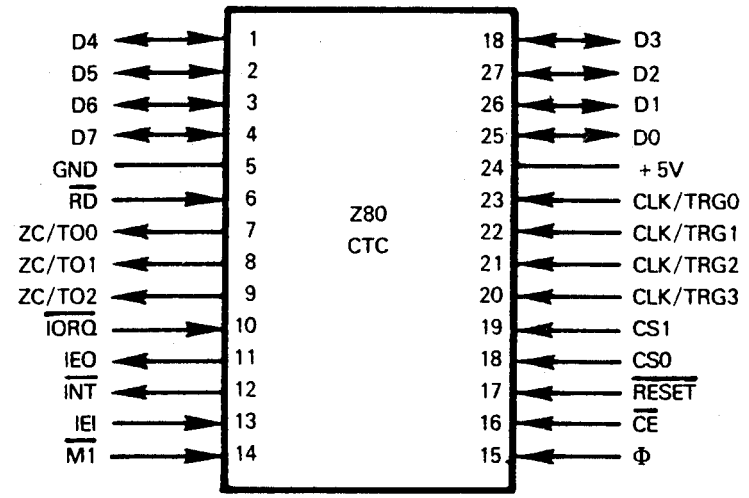
30 CTC pins and signals are illustrated in Figure 7-22.

0 - D7 is the bidirectional Data Bus via which parallel data is transferred between the CPU and any register of the 30 CTC.

$\overline{CE}$ is the master chip select signal for the Z80 CTC. This signal must be low for the device to be selected.

While $\overline{CE}$ is low, **CS0 and CS1 are used to select one of the four counter/timer logic channels** as follows:

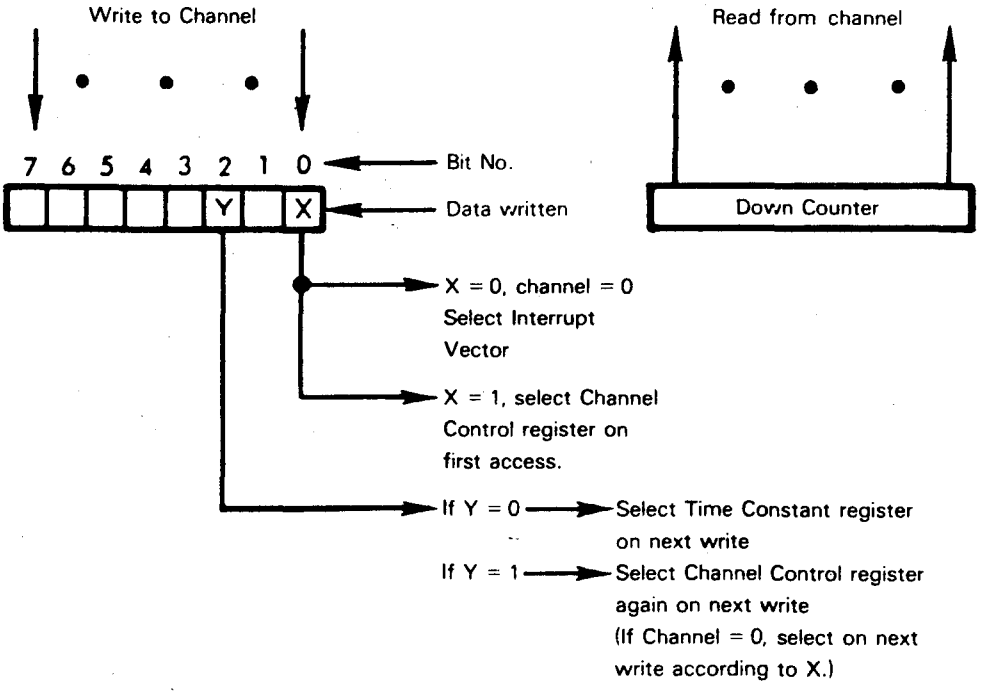
CS1	CS0	Channel
0	0	0
0	1	1
1	0	2
1	1	3



PIN NAME	DESCRIPTION	TYPE
D0-D7	Data Bus	Bidirectional, tristate
CLK/TRG0, CLK/TRG1, CLK/TRG2, CLK/TRG3,	External Clock or timer trigger	Input
ZC/TO0 ZC/TO1 ZC/TO2	Zero Count or timeout indicator	Output
$\overline{M1}$	Instruction fetch machine cycle signal from CPU	Input
$\overline{IORQ}$	Input/Output request from CPU	Input
$\overline{RD}$	Read cycle status from CPU	Input
$\overline{RESET}$	Device Reset	Input
IEI	Interrupt enable in	Input
IEO	Interrupt enable out	Output
$\overline{INT}$	Interrupt request	Output, Open-drain
$\overline{CE}$	Device enable	Input
CS0, CS1	Register select	Input
Φ, +5V, GND	Clock, power and ground	

Figure 7-22. Z80-CTC Signals and Pin Assignments

.0 and CS1 select registers associated with counter/timer logic, to be accessed by read and write operations. The ac-
al register which will be accessed is determined as follows:



the illustration above would imply, the Down Counter register is the only location of any channel whose contents
n be read. All other registers are write only locations.

hen you write to a channel, bits 0 and 2 of the data byte being written determine the data destination as follows:

If bit 0 is 0 and you are selecting channel 0, then the data is written to the Interrupt Vector register.

If bit 0 is 0 and you select channel 1, 2 or 3, the data destination is undefined.

If bit 0 is 1, then on the first access of any channel the data will be written to the Channel Control register.

If within the data byte written to a Channel Control register bit 0 is 1 and bit 2 is 0, then the next data byte written to this channel will be loaded into the Time Constant register, irrespective of whether bit 0 is 0 or 1. The data written will be interpreted as a time constant; select logic will immediately revert to selecting the Channel Control register or the Interrupt Vector register on the next write, depending on the condition of bit 0 of the next data byte.

$\overline{1}$, $\overline{IORQ}$ and $\overline{RD}$ are three control signals input to the Z80 CTC. Combinations of these three control signals **control
gic within the Z80 CTC, as described for the Z80 PIO. An exception is the device Reset.** The Z80 CTC has its
n $\overline{RESET}$ input. The PIO decodes a Reset when $\overline{M1}$ is low while $\overline{IORQ}$ and $\overline{RD}$ are high. With the exception of the
SET function, Table 7-4 defines the manner in which the Z80 CTC interprets $\overline{M1}$, $\overline{IORQ}$, and $\overline{RD}$ signals.

terrupt logic has three associated signals: IEI, IEO and $\overline{INT}$. These signals operate exactly as described for the
0 PIO.

e Z80 CTC requests an interrupt with a low $\overline{INT}$ output.

and IEO are used to implement daisy chain priority interrupt logic as described for the PIO.

ch of the four counter/timer channels has a CLK/TRG input control. This signal can be used to trigger timer logic;
s also used as a decrement control by counter logic.

unter/timer logic channels 0, 1 and 2 have a ZC/TO output. This signal is pulsed high on a time out or a count out.

hen a low input is applied to the $\overline{RESET}$ pin, the Z80 CTC is reset. At this time all counter/timer logic is stopped.
$\overline{T}$ is output high, IEO is output at the IEI level and the Data Bus is floated. Register contents are not cleared during a
et.

80 CTC OPERATING MODES

**e Z80 CTC is accessed by the CPU as four I/O ports or four memory locations. Timing for any CTC access con-
rms to descriptions given earlier in this chapter for the CPU.**

t us begin by looking at a counter/timer operating as a timer.

Using an appropriate Control code (described later) you select Timer mode for the channel and specify that an initial time constant is to follow.

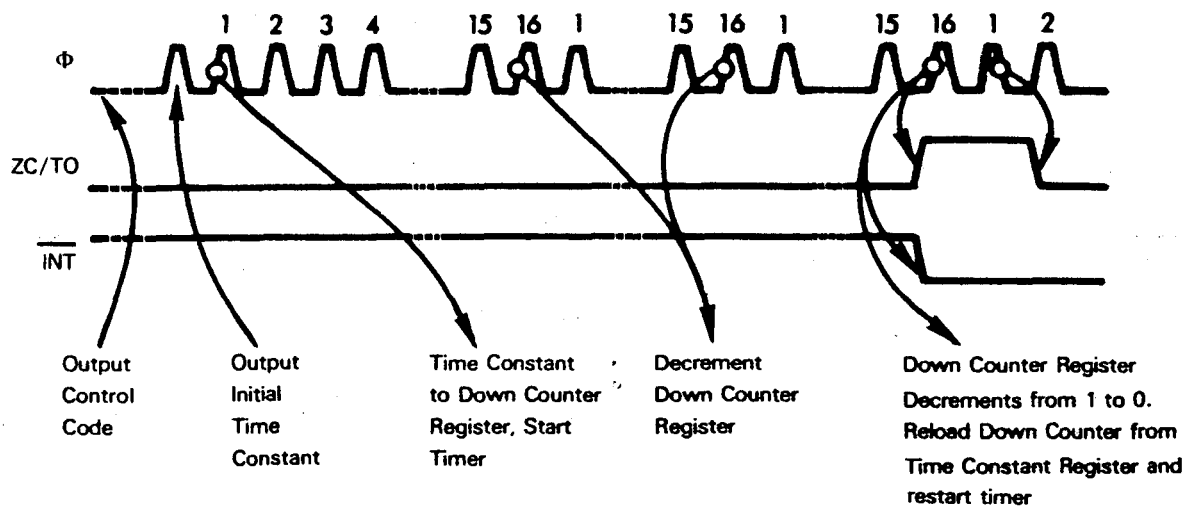
You load an initial constant into the Time Constant register, after which timer operations begin.

You have the option of using the CLK/TRG input to start the timer, in which case timer logic is initiated by external logic. The alternative is to initiate the timer under program control, in which case the timer starts on the clock pulse following the Time Constant register being loaded.

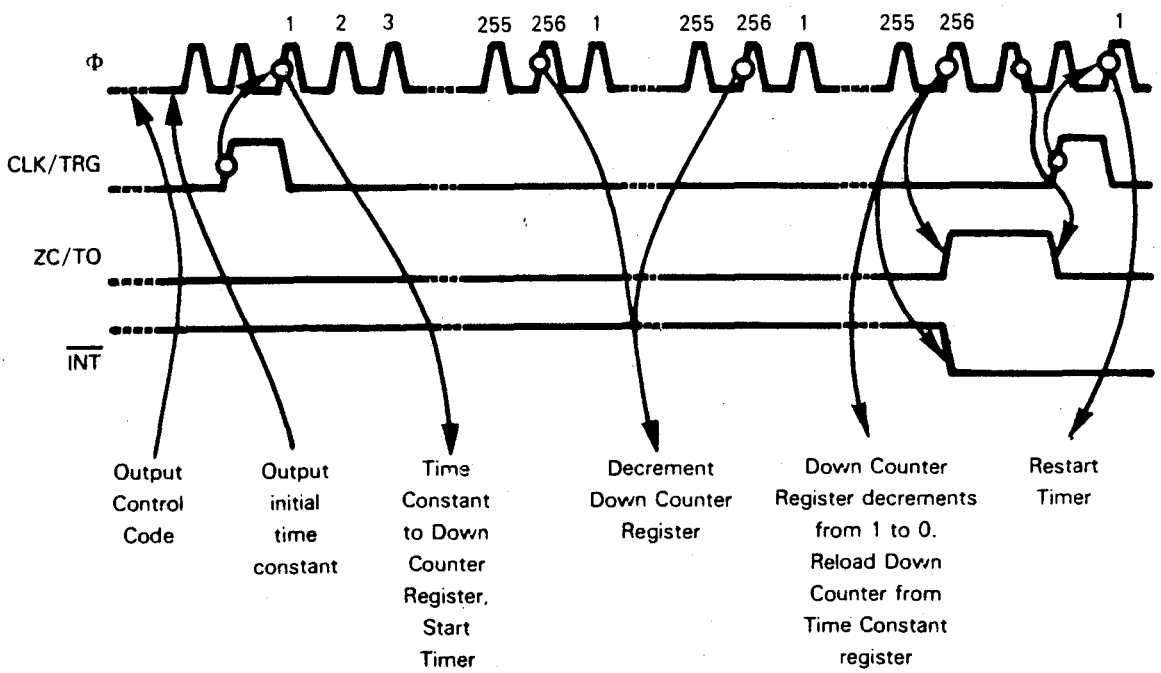
When timer operations begin, the Time Constant register contents are transmitted to the Down Counter register. The Down Counter register contents are decremented on every 16th system clock pulse, or on every 256th system clock pulse. You make the selection via the Control code. Assuming a 500 nanosecond clock, therefore, the timer will decrement the Down Counter register contents every 8 microseconds, or every 128 microseconds.

When timer logic decrements the Down Counter register contents from 1 to 0 a time out occurs. At this time ZC/TO is pulsed high, the Time Constant register contents are reloaded into the Down Counter register and timer logic starts again. Thus timer logic is free running; once started, the timer will run continuously until stopped by an appropriate Control code.

Here is a timing example for a timer started under program control and decrementing the Down Counter register on every 16th clock pulse:

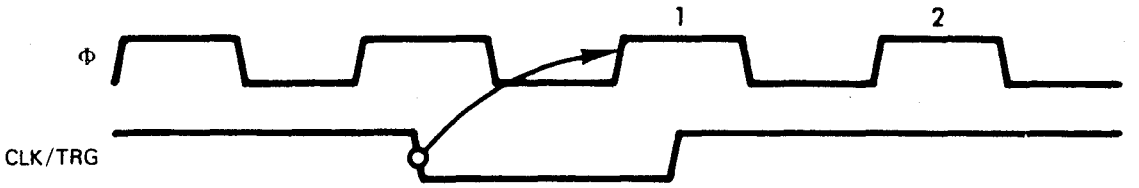


Here is a timing example for a timer whose operations are initiated by CLK/TRG, where the Down Counter register contents are decremented on every 256th clock pulse:



serve that every time out is marked by a ZC/TO high pulse. $\overline{INT}$ is also output low providing interrupt logic is enabled he channel.

he illustration above CLK/TRG is shown as a high true signal. You can specify CLK/TRG as a low true signal via the annel Control code; the timer will be initiated as follows:



exact timing requirements see the data sheets at the end of this chapter.

u can at any time write new data into the Time Constant register. If you do this while the timer is running, nothing pens until the next time out; at that time the new Time Constant register contents will be transferred to the Down unter register and subsequent time intervals will be computed based on the new Time Constant register contents.

ou are unfortunate enough to output data to the Time Constant register while a time out is in progress and the Time nstant register contents are being transferred to the Down Counter register, then an undefined value will be loaded the Down Counter register; however, following the next time out the new value in the Time Constant register will ly; that is to say, there will only be one undefined time interval.

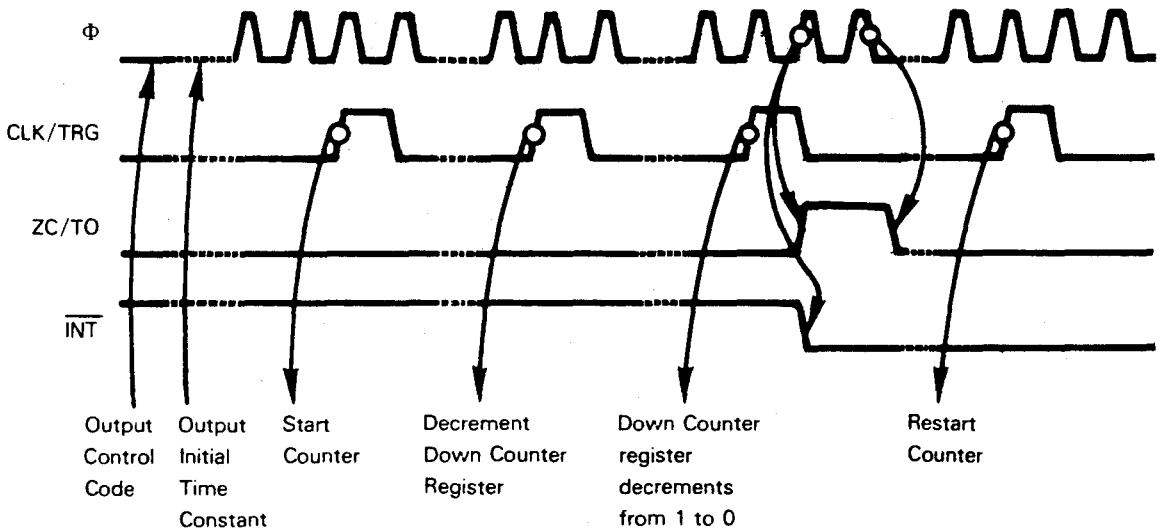
us now look at a counter/timer operating as a counter.

ng an appropriate Control code (described later) you select Counter mode for the channel and specify that an initial e constant is to follow.

load an initial constant into the Time Constant register, after which counter operations begin.

en counter operations begin, the Time Constant register contents are transmitted to the Down Counter register. The wn Counter register contents are decremented every time the CLK/TRG input makes an active transition. Counter c begins on the first active transition of CLK/TRG following data being loaded into the Time Constant register. The ve transition of CLK/TRG may be selected under program control as low-to-high or high-to-low.

en counter logic decrements the Down Counter register contents from 1 to 0, a count out occurs. At this time the TO signal is pulsed high; an interrupt request occurs, providing the channel's interrupt logic has been enabled. The e Constant register contents are reloaded into the Down Counter register and counter operations begin again. That say, counter logic is free running and will continue to re-execute until specifically stopped by an appropriate Con- code. Counter logic timing may be illustrated as follows:



Z80 CTC INTERRUPT LOGIC

Every Z80 CTC channel has its own interrupt logic. A channel's interrupt logic generates an interrupt request when the channel counts out or times out. All interrupt requests are transmitted to the CPU via the $\overline{INT}$ output. This is true if one, or more than one channel is requesting an interrupt. If more than one channel is requesting an interrupt, then priorities are arbitrated as follows:

Highest Priority	Channel 0
	Channel 1
	Channel 2
Lowest Priority	Channel 3

Every channel's interrupt logic can be individually enabled or disabled under program control.

The Z80 CTC device's overall interrupt logic is identical to that which we have already described for the Z80 PIO.

The interrupt request is transmitted to the CPU via a low $\overline{INT}$ signal.

The CPU acknowledges the interrupt by outputting $\overline{M1}$ and $\overline{IORQ}$ low as illustrated in the data sheets at the end of this chapter.

The device requesting an interrupt which is highest in the daisy chain acknowledges the interrupt. Presuming this is a Z80 CTC, the CTC places its interrupt vector on the Data Bus; it is assumed that the CPU is operating in Interrupt mode 2. The Z80 CTC immediately outputs IEO low, disabling all devices below it in the daisy chain.

When an RETI instruction is executed, Z80 CTC logic sets IEO high again.

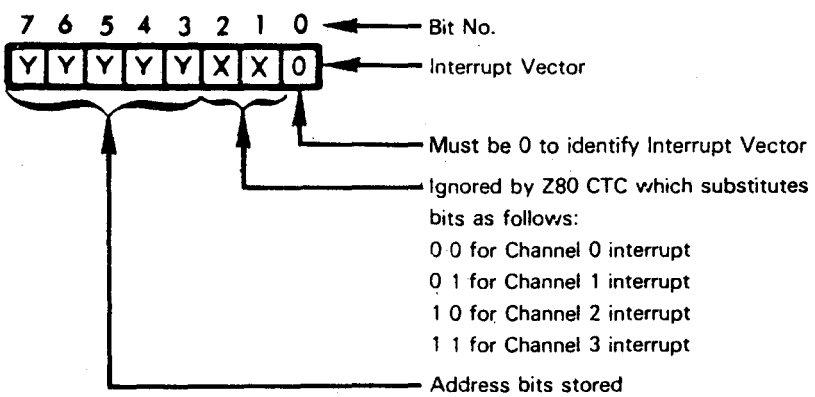
For more information on Z80 interrupt logic refer to discussions of this subject given earlier in the chapter for the Z80 CPU and the PIO.

PROGRAMMING THE Z80 CTC

These are the steps required to program a Z80 CTC:

1) **Output an interrupt vector once, when initializing the Z80 CTC.**
2) **For each active counter/timer channel, output one or more Control codes. Control codes are used initially to set counter/timer operating conditions and to load the Time Constant register. Subsequently Control codes are used to start and stop the counter/timer, or to change the initial time constant.**

The interrupt vector is written to a counter/timer by outputting a byte of data to counter/timer channel 0 with a 0 in the low order bit. The interrupt vector may be illustrated as follows:



The Control code which must be output to each active channel will be interpreted as illustrated in Figure 7-23.

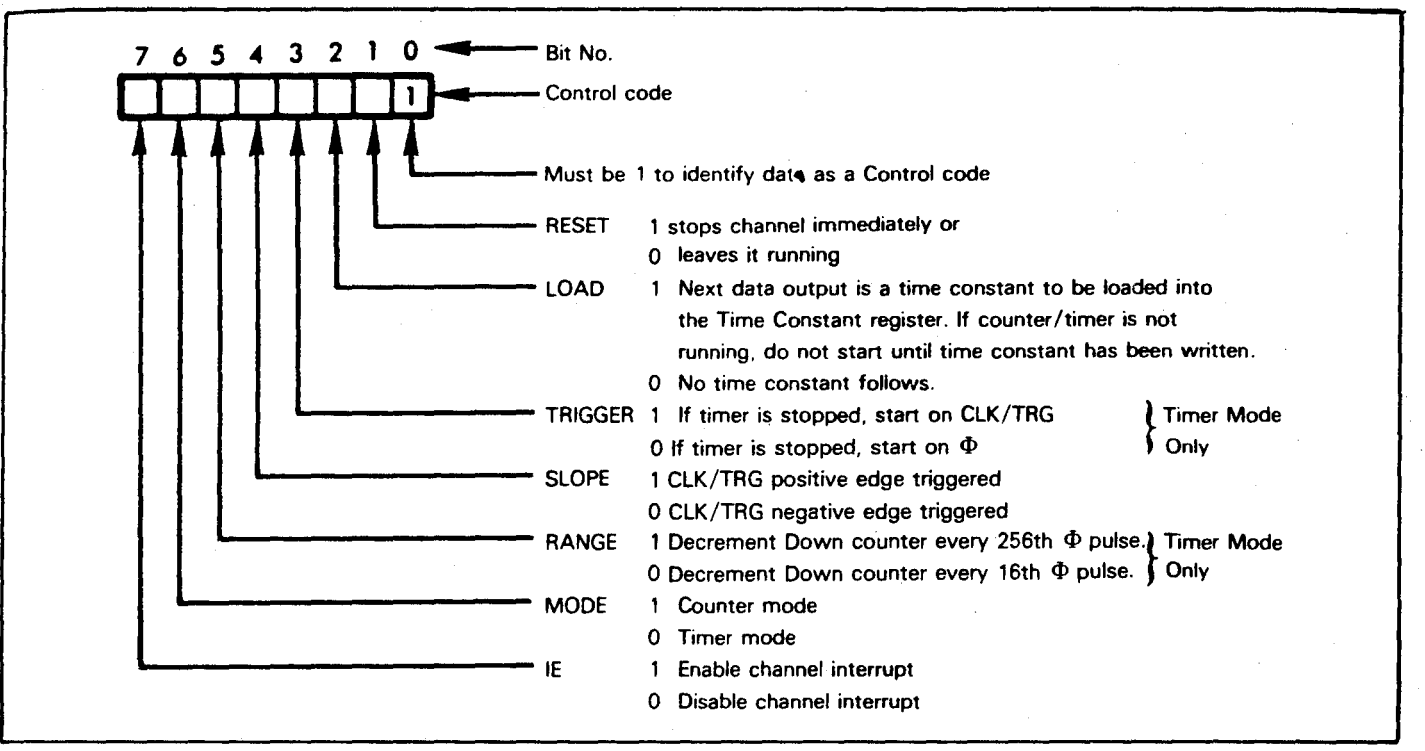


Figure 7-23. Z80 CTC Control Code Interpretation

Bit 0 must be 1 to identify the data as a Control code. If bit 0 is 0, then the data is interpreted as an interrupt vector — providing Channel 0 is addressed; the data is undefined otherwise.

Bit 1 is used to stop the channel when it is running. If bit 1 is 0, then every time the channel times out the Down Counter register is immediately reloaded from the Time Constant register contents and channel operations restart according to current options. If bit 1 is 1, the channel stops immediately; the ZC/TO output is inactive and channel interrupt logic is disabled. The channel must be restarted by outputting a new Control code.

Bit 2 is used to output time constants. If bit 2 is 1, then the next data output to the channel will be interpreted as a time constant. If bit 2 is 0, then the next data output to the channel will be interpreted as another Control code, or an interrupt vector, depending on the bit 0 value.

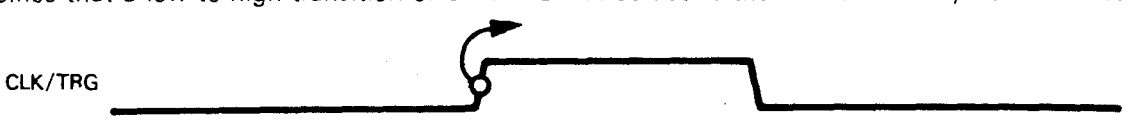
Bit 3 applies to Timer mode only; assuming that the timer is not running, it determines whether timer operations will be initiated by the system clock signal Φ, or by CLK/TRG.

If bit 3 is 0 then timer operations are initiated by system clock signal Φ; the timer will start on the next leading edge of Φ, unless the current Control code specifies (via bit 2) that a new time constant is to be output, in which case the timer will start on the rising edge of Φ which immediately follows output of the time constant. Timing for these two cases has been illustrated earlier.

If bit 3 is 1, then the active transition of the CLK/TRG signal initiates the timer. Once again, if bit 2 of the current Control code specifies that a new time constant is to be output then timer logic cannot be started until this new time constant has been output. Timing has been illustrated earlier.

Bit 4 determines whether the low-to-high or the high-to-low transition of CLK/TRG is active. Assuming that bit 6 has specified Timer mode and bit 3 has specified the timer will be triggered externally by CLK/TRG, the active transition of CLK/TRG starts the timer. If bit 6 is not 0 or bit 3 is not 1, then the active transition of CLK/TRG decrements the counter.

If bit 4 specifies that a low-to-high transition of CLK/TRG will be active then CLK/TRG may be illustrated as follows:



bit 4 specifies that the high-to-low transition of CLK/TRG will be active then CLK/TRG may be illustrated as follows:

Bit 5 applies to Timer mode only. If bit 5 is 0, Down Counter register contents will be decremented every 16th system clock pulse (Φ). If bit 5 is 1, the Down Counter register contents will be decremented every 256th system clock pulse (Φ).

Bit 6 determines whether the channel will be operated as a counter or a timer. If bit 6 is 0, Timer mode is selected; Counter mode is selected if bit 6 is 1.

Bit 7 is an interrupt enable/disable flag. If 0, the channel's interrupt logic is disabled; if 1, the channel's interrupt logic is enabled.

Let us now look at the programming example. Here are the assumed operating conditions for the Z80 CTC:

1) Channel 0 is operating as a counter with an initial time constant of 80_{16} and interrupt logic enabled.
2) Channel 1 is operating as a timer. It decrements on every 16th system clock pulse and has an initial time constant of 40_{16}; its interrupts are disabled and CLK/TRG starts the timer on its low-to-high transition.
3) Channel 2 is operating as a timer. It decrements every 256th system clock pulse and has an initial time constant of $C8_{16}$; its interrupts are enabled and the system clock starts the timer.
4) Channel 3 is inactive.

The CPU is operating with interrupt logic in Mode 2. CTC interrupt service routine starting addresses are stored at memory locations $2C40_{16}$, $2C42_{16}$ and $2C44_{16}$. The CTC is accessed as I/O ports $B8_{16}$, $B9_{16}$, BA_{16}, and BB_{16}.

Here is the appropriate CTC initiation instruction sequence:

```
        LD      A,2CH     ;LOAD INTERRUPT VECTOR REGISTER OF CPU
        LD      I,A
        IM      2         ;SELECT CPU INTERRUPT MODE 2
        LD      A,40H     ;OUTPUT INTERRUPT VECTOR TO
        OUT     (0B8H),A  ;CHANNEL 0
;START CHANNEL 0
        LD      A,0C5H    ;OUTPUT THE CONTROL CODE TO CHANNEL 0
        OUT     (0B8H),A
        LD      A,80H     ;OUTPUT THE INITIAL COUNT TO CHANNEL 0
        OUT     (0B8H),A  ;CHANNEL 0 BEGINS OPERATING.
;START CHANNEL1
        LD      A,1DH     ;OUTPUT THE CONTROL CODE TO CHANNEL 1
        OUT     (0B9H),A
        LD      A,40H     ;OUTPUT THE INITIAL TIMER CONSTANT TO CHANNEL1
        OUT     (0B9H),A  ;CHANNEL 1 BEGINS OPERATING. (IF TRANSITION OCCURS)
;START CHANNEL 2
        LD      A,0A5H    ;OUTPUT THE CONTROL CODE TO CHANNEL 2
        OUT     (0BAH),A
        LD      A,0C8H    ;OUTPUT THE INITIAL TIMER CONSTANT TO CHANNEL 2
        OUT     (0BAH),A  ;CHANNEL 2 BEGINS OPERATING
```

Serial Input/Output Controller

Z8440
Z80 SIO

Features

- Two independent full-duplex channels, with separate control and status lines for modems or other devices.
- Data rates of 0 to 500K bits/second in the x1 clock mode with a 2.5 MHz clock (Z80 SIO), or 0 to 800K bits/second with a 4.0 MHz clock (Z80A SIO).
- Asynchronous protocols: everything necessary for complete messages in 5, 6, 7 or 8 bits/character. Includes variable stop bits and several clock-rate multipliers; break generation and detection; parity; overrun and framing error detection.
- Synchronous protocols: everything necessary for complete bit- or byte-oriented messages in 5, 6, 7 or 8 bits/character, including IBM Bisync, SDLC, HDLC, CCITT-X.25 and others. Automatic CRC generation/checking, sync character and zero insertion/deletion, abort generation/detection and flag insertion.
- Receiver data registers quadruply buffered, transmitter registers doubly buffered.
- Highly sophisticated and flexible daisy-chain interrupt vectoring for interrupts without external logic.

General Description

The Z-80 SIO Serial Input/Output Controller is a dual-channel data communication interface with extraordinary versatility and capability. Its basic functions as a serial-to-parallel, parallel-to-serial converter/controller can be programmed by a CPU for a broad range of serial communication applications.

The device supports all common asynchronous and synchronous protocols, byte- or bit-oriented, and performs all of the functions traditionally done by UARTs, USARTs and synchronous communication controllers combined, plus additional functions traditionally performed by the CPU. Moreover, it does this on two fully-independent channels, with an exceptionally sophisticated interrupt structure that allows very fast transfers.

Full interfacing is provided for CPU or DMA

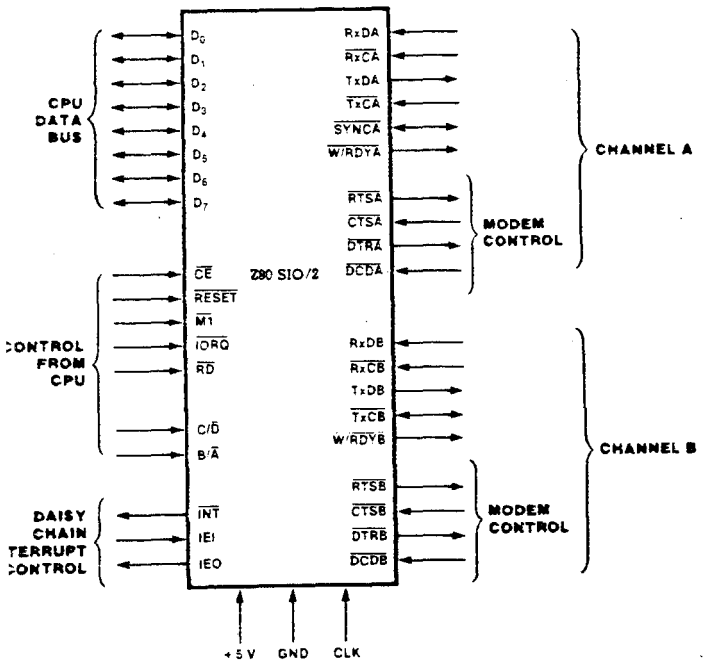


Figure 1. Z80 SIO-2 Logic Functions

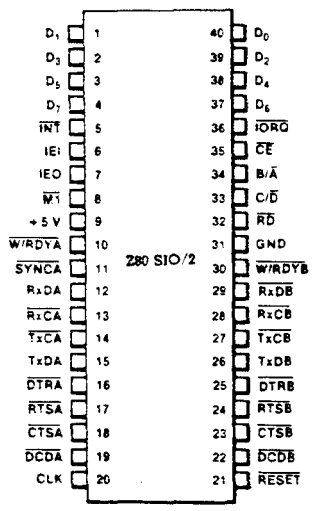


Figure 2. Z80 SIO-2 Pin Configuration

General Description (Continued)

control. In addition to data communication, the circuit can handle virtually all types of serial I/O with fast (or slow) peripheral devices. While designed primarily as a member of the Z80 family, its versatility makes it well suited to many other CPUs.

The Z80 SIO is an n-channel silicon-gate depletion-load device packaged in a 40-pin plastic or ceramic DIP. It uses a single +5 V power supply and the standard Z80 family single-phase clock.

Pin Description

Figures 1 through 6 illustrate the three pin configurations (bonding options) available in the SIO. The constraints of a 40-pin package make it impossible to bring out the Receive Clock ($\overline{RxC}$), Transmit Clock ($\overline{TxC}$), Data Terminal Ready ($\overline{DTR}$) and Sync ($\overline{SYNC}$) signals for both channels. Therefore, either Channel B lacks a signal or two signals are bonded together in the three bonding options offered:

- Z80 SIO-2 lacks SYNCB
- Z80 SIO-1 lacks DTRB
- Z80 SIO-0 has all four signals, but TxCB and RxCB are bonded together

The first bonding option above (SIO-2) is the preferred version for most applications. The pin descriptions are as follows:

B/$\overline{A}$. *Channel A Or B Select* (input, High selects Channel B). This input defines which channel is accessed during a data transfer between the CPU and the SIO. Address bit A_0 from the CPU is often used for the selection function.

C/$\overline{D}$. *Control Or Data Select* (input, High selects Control). This input defines the type of information transfer performed between the

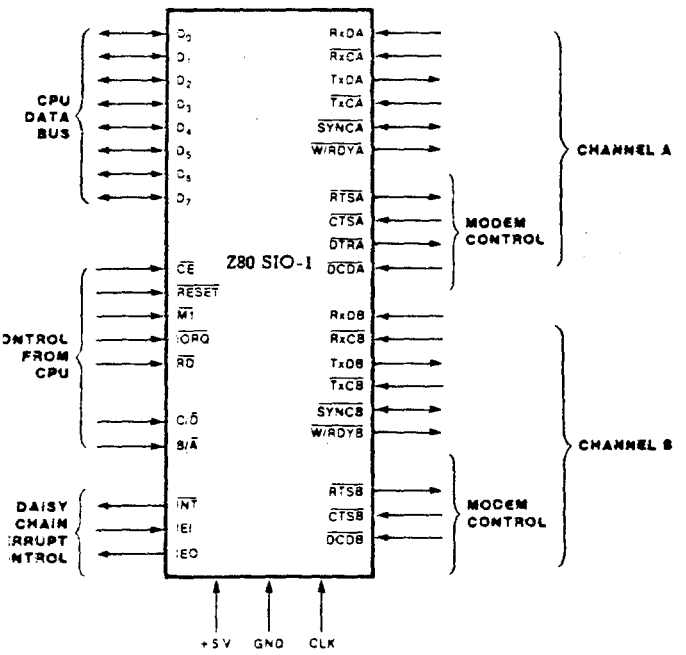


Figure 3. Z80 SIO-1 Logic Functions

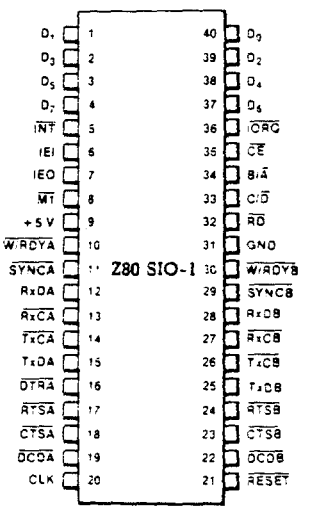


Figure 4. Z80 SIO-1 Pin Configuration

in Description (Continued)

PU and the SIO. A High at this input during CPU write to the SIO causes the information n the data bus to be interpreted as a com- and for the channel selected by B/$\overline{A}$. A Low C/$\overline{D}$ means that the information on the data us is data. Address bit A_1 is often used for is function.

E. *Chip Enable* (input, active Low). A Low vel at this input enables the SIO to accept ommand or data input from the CPU during a rite cycle or to transmit data to the CPU uring a read cycle.

LK. *System Clock* (input). The SIO uses the andard Z80 System Clock to synchronize ternal signals. This is a single-phase clock.

TSA, CTSB. *Clear To Send* (inputs, active ow). When programmed as Auto Enables, a ow on these inputs enables the respective ansmitter. If not programmed as Auto nables, these inputs may be programmed as eneral-purpose inputs. Both inputs are hmitt-trigger buffered to accommodate slow- etime signals. The SIO detects pulses on ese inputs and interrupts the CPU on both gic level transitions. The Schmitt-trigger buf- ing does not guarantee a specified noise- el margin.

-D_7. *System Data Bus* (bidirectional, state). The system data bus transfers data d commands between the CPU and the Z-80 D. D_0 is the least significant bit.

DA, DCDB. *Data Carrier Detect* (inputs, ive Low). These pins function as receiver ables if the SIO is programmed for Auto ables; otherwise they may be used as neral-purpose input pins. Both pins are mitt-trigger buffered to accommodate slow- time signals. The SIO detects pulses on se pins and interrupts the CPU on both ic level transitions. Schmitt-trigger buffer- does not guarantee a specific noise-level rgin.

RA, DTRB. *Data Terminal Ready* (outputs, ve Low). These outputs follow the state programmed into Z80 SIO. They can also be programmed as general-purpose outputs.

In the Z80 SIO-1 bonding option, DTRB is omitted.

IEI. *Interrupt Enable In* (input, active High). This signal is used with IEO to form a priority daisy chain when there is more than one interrupt-driven device. A High on this line indicates that no other device of higher priority is being serviced by a CPU interrupt service routine.

IEO. *Interrupt Enable Out* (output, active High). IEO is High only if IEI is High and the CPU is not servicing an interrupt from this SIO. Thus, this signal blocks lower priority devices from interrupting while a higher priority device is being serviced by its CPU interrupt service routine.

$\overline{INT}$. *Interrupt Request* (output, open drain, active Low). When the SIO is requesting an interrupt, it pulls $\overline{INT}$ Low.

$\overline{IORQ}$. *Input/Output Request* (input from CPU, active Low). $\overline{IORQ}$ is used in conjunction with B/$\overline{A}$, C/$\overline{D}$, $\overline{CE}$ and $\overline{RD}$ to transfer commands and data between the CPU and the SIO. When $\overline{CE}$, $\overline{RD}$ and $\overline{IORQ}$ are all active, the channel selected by B/$\overline{A}$ transfers data to the CPU (a read operation). When $\overline{CE}$ and $\overline{IORQ}$ are active but $\overline{RD}$ is inactive, the channel selected by B/$\overline{A}$ is written to by the CPU with either data or control information as specified by C/$\overline{D}$. If $\overline{IORQ}$ and $\overline{M1}$ are active simultaneously, the CPU is acknowledging an interrupt and the SIO automatically places its interrupt vector on the CPU data bus if it is the highest priority device requesting an interrupt.

$\overline{M1}$. *Machine Cycle* (input from Z80 CPU, active Low). When $\overline{M1}$ is active and $\overline{RD}$ is also active, the Z80 CPU is fetching an instruction from memory; when $\overline{M1}$ is active while $\overline{IORQ}$ is active, the SIO accepts $\overline{M1}$ and $\overline{IORQ}$ as an interrupt acknowledge if the SIO is the highest priority device that has interrupted the Z80 CPU.

$\overline{RxCA}$, $\overline{RxCB}$. *Receiver Clocks* (inputs). Receive data is sampled on the rising edge of

Pin Description (Continued)

RxC. The Receive Clocks may be 1, 16, 32 or 64 times the data rate in asynchronous modes. These clocks may be driven by the Z80 CTC Counter Timer Circuit for programmable baud rate generation. Both inputs are Schmitt-trigger buffered (no noise level margin is specified).

In the Z80 SIO-0 bonding option, RxCB is bonded together with TxCB.

RD. *Read Cycle Status* (input from CPU, active Low). If RD is active, a memory or I/O read operation is in progress. RD is used with B/Ā, CE and IORQ to transfer data from the SIO to the CPU.

RxDA, RxDB. *Receive Data* (inputs, active High). Serial data at TTL levels.

RESET. *Reset* (input, active Low). A Low RESET disables both receivers and transmitters, forces TxDA and TxDB marking, forces the modem controls High and disables all interrupts. The control registers must be rewritten after the SIO is reset and before data is transmitted or received.

RTSA, RTSB. *Request To Send* (outputs, active Low). When the RTS bit in Write Register 5 (Figure 14) is set, the RTS output goes Low. When the RTS bit is reset in the Asynchronous mode, the output goes High after the transmitter is empty. In Synchronous modes, the RTS pin strictly follows the state of the RTS bit. Both pins can be used as general-purpose outputs.

SYNCA, SYNCB. *Synchronization* (inputs/outputs, active Low). These pins can act either as inputs or outputs. In the asynchronous receive mode, they are inputs similar to CTS and DCD. In this mode, the transitions on these lines affect the state of the Sync/Hunt status bits in Read Register 0 (Figure 13), but have no other function. In the External Sync mode, these lines also act as inputs. When external synchronization is achieved, SYNC must be driven Low on the second rising edge of RxC after that rising edge of RxC on which the last bit of the sync character was received. In other words, after the sync pattern is detected, the external logic must wait for two full Receive Clock cycles to activate the SYNC input. Once SYNC is forced Low, it should be kept Low until the CPU informs the external synchronization detect logic that synchroniza-

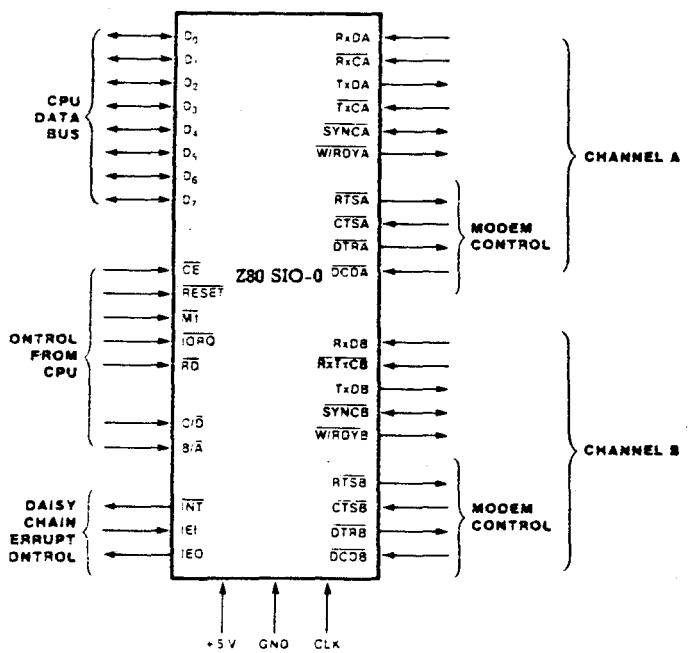


Figure 5. Z80 SIO-0 Logic Functions

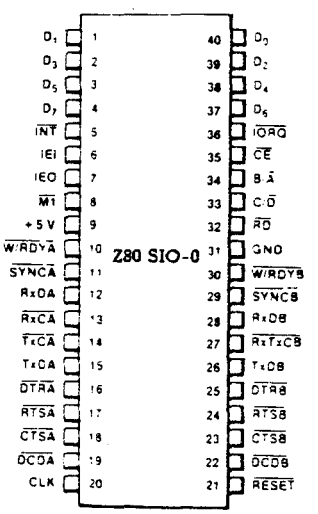


Figure 6. Z80 SIO-0 Pin Configuration

Pin Description (Continued)

tion has been lost or a new message is about to start. Character assembly begins on the rising edge of $\overline{RxC}$ that immediately precedes the falling edge of $\overline{SYNC}$ in the External Sync mode.

In the internal synchronization mode (Monosync and Bisync) these pins act as outputs that are active during the part of the receive clock ($\overline{RxC}$) cycle in which sync characters are recognized. The sync condition is not latched, so these outputs are active each time a sync pattern is recognized, regardless of character boundaries.

In the Z-80 SIO/2 bonding option, $\overline{SYNCB}$ is omitted.

$\overline{TxCA}$, $\overline{TxCB}$. *Transmitter Clocks* (inputs). In asynchronous modes, the Transmitter Clocks may be 1, 16, 32 or 64 times the data rate; however, the clock multiplier for the transmitter and the receiver must be the same. The Transmit Clock inputs are Schmitt-trigger buffered for relaxed rise- and fall-time requirements (no noise level margin is specified). Transmitter Clocks may be driven by the Z-80 CTC Counter Timer Circuit for programmable baud rate generation.

In the Z80 SIO-0 bonding option, TxCB is bonded together with $\overline{RxCB}$.

TxDA, TxDB. *Transmit Data* (outputs, active High). Serial data at TTL levels. TxD changes from the falling edge of $\overline{TxC}$.

$\overline{W}/\overline{RDYA}$, $\overline{W}/\overline{RDYB}$. *Wait/Ready A, Wait/Ready B* (outputs, open drain when programmed for Wait function, driven High and Low when programmed for Ready function). These dual-purpose outputs may be programmed as Ready lines for a DMA controller or as Wait lines that synchronize the CPU to the SIO data rate. The reset state is open drain.

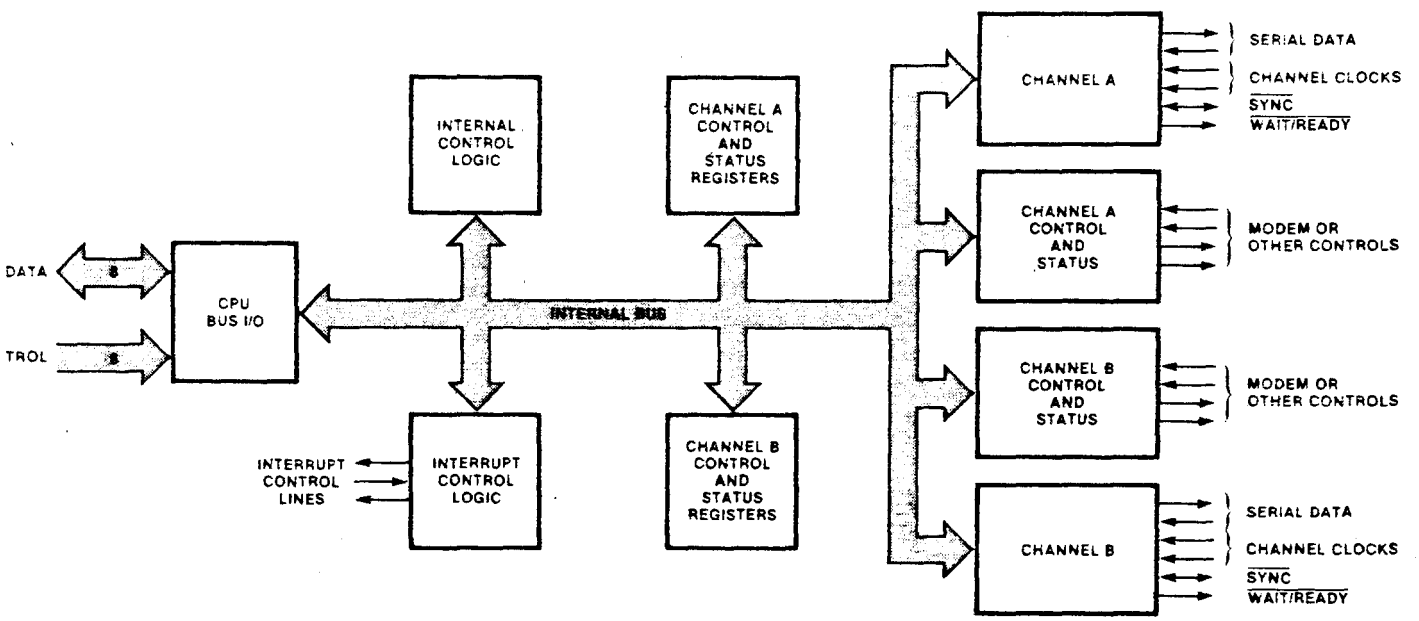


Figure 7. Block Diagram

Functional Description

The functional capabilities of the Z80 SIO can be described from two different points of view: as a data communications device, it transmits and receives serial data in a wide variety of data-communication protocols; as a Z80 family peripheral, it interacts with the Z80 CPU and other peripheral circuits, sharing the data, address and control buses, as well as being a part of the Z80 interrupt structure. As a peripheral to other microprocessors, the SIO offers valuable features such as non-vectored interrupts, polling and simple handshake capability.

Figure 8 illustrates the conventional devices that the SIO replaces.

The first part of the following discussion covers SIO data-communication capabilities; the second part describes interactions between the CPU and the SIO.

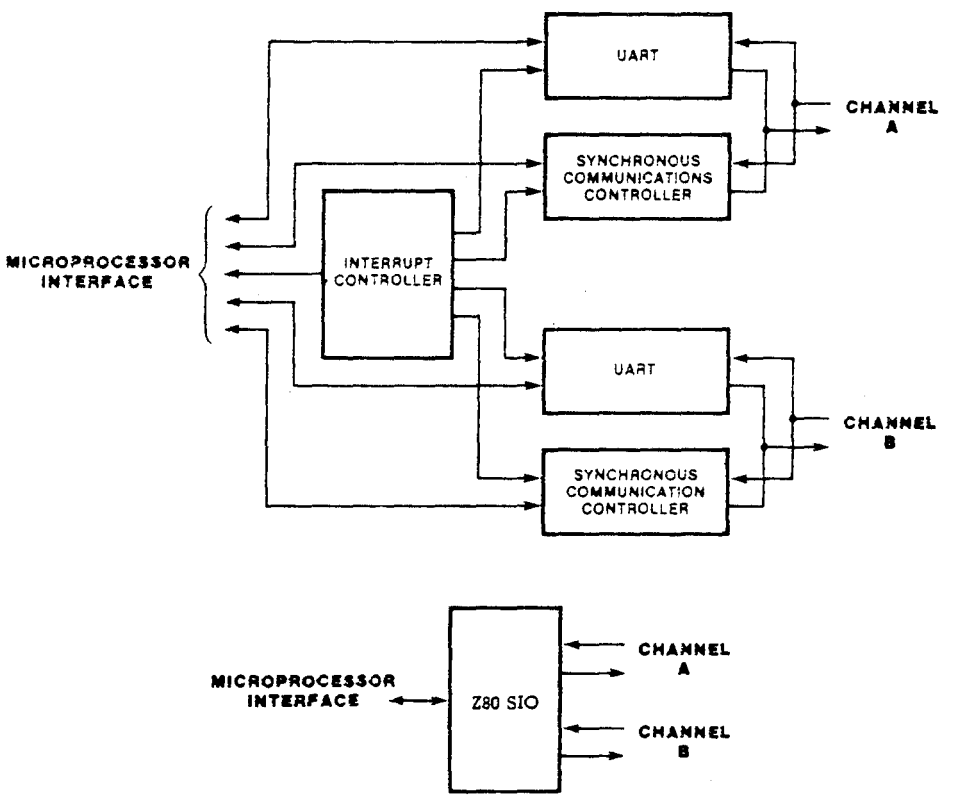


Figure 8. Conventional Devices Replaced by the Z80 SIO

ta Communication Capabilities

The SIO provides two independent full-plex channels that can be programmed for e in any common asynchronous or synchro-us data-communication protocol. Figure 9 ustrates some of these protocols. The follow-g is a short description of them. A more tailed explanation of these modes can be nd in the *Z80 Family Technical Manual.*

ynchronous Modes. Transmission and ception can be done independently on each annel with five to eight bits per character, us optional even or odd parity. The transmit-s can supply one, one-and-a-half or two stop s per character and can provide a break tput at any time. The receiver break-tection logic interrupts the CPU both at the rt and end of a received break. Reception is otected from spikes by a transient spike-ection mechanism that checks the signal e-half a bit time after a Low level is detected the receive data input (RxDA or RxDB in ure 5). If the Low does not persist—as in case of a transient—the character assembly ocess is not started.

Framing errors and overrun errors are ected and buffered together with the partial aracter on which they occurred. Vectored errupts allow fast servicing of error condi-is using dedicated routines. Furthermore, a lt-in checking process avoids interpreting a ning error as a new start bit: a framing or results in the addition of one-half a bit e to the point at which the search for the t start bit is begun.

he SIO does not require symmetric transmit receive clock signals—a feature that ws it to be used with a Z80 CTC or many er clock sources. The transmitter and eiver can handle data at a rate of 1, 1/16, 2 or 1/64 of the clock rate supplied to the eive and transmit clock inputs.

n asynchronous modes, the $\overline{\text{SYNC}}$ pin may programmed as an input that can be used functions such as monitoring a ring icator.

Synchronous Modes. The SIO supports both byte-oriented and bit-oriented synchronous communication.

Synchronous byte-oriented protocols can be handled in several modes that allow character synchronization with an 8-bit sync character (Monosync), any 16-bit sync pattern (Bisync), or with an external sync signal. Leading sync characters can be removed without interrupting the CPU.

Five-, six- or seven-bit sync characters are detected with 8- or 16-bit patterns in the SIO by overlapping the larger pattern across multiple in-coming sync characters, as shown in Figure 10.

CRC checking for synchronous byte-oriented modes is delayed by one character time so the CPU may disable CRC checking on specific characters. This permits implementation of protocols such as IBM Bisync.

Both CRC-16 ($X^{16} + X^{15} + X^2 + 1$) and CCITT ($X^{16} + X^{12} + X^5 + 1$) error checking polynomials are supported. In all non-SDLC modes, the CRC generator is initialized to 0's; in SDLC modes, it is initialized to 1's. The SIO can be used for interfacing to peripherals such as hard-sectored floppy disk, but it cannot generate or check CRC for IBM-compatible soft-sectored disks. The SIO also provides a feature that automatically transmits CRC data when no other data is available for transmission. This allows very high-speed transmissions under DMA control with no need for CPU intervention at the end of a message. When there is no data or CRC to send in synchronous modes, the transmitter inserts 8- or 16-bit sync characters regardless of the programmed character length.

The SIO supports synchronous bit-oriented protocols such as SDLC and HDLC by performing automatic flag sending, zero insertion and CRC generation. A special command can be used to abort a frame in transmission. At the end of a message the SIO automatically transmits the CRC and trailing flag when the transmit buffer becomes empty. If a transmit

Data Communication Capabilities (Continued)

underrun occurs in the middle of a message, an external/status interrupt warns the CPU of this status change so that an abort may be issued. One to eight bits per character can be sent, which allows reception of a message with no prior information about the character structure in the information field of a frame.

The receiver automatically synchronizes on the leading flag of a frame in SDLC or HDLC, and provides a synchronization signal on the $\overline{SYNC}$ pin; an interrupt can also be programmed. The receiver can be programmed to search for frames addressed by a single byte to only a specified user-selected address or to a global broadcast address. In this mode, frames that do not match either the user-selected or broadcast address are ignored. The number of address bytes can be extended under software control. For transmitting data, an interrupt on the first received character or on every character can be selected. The receiver automatically deletes all zeroes inserted by the transmitter during character assembly. It also calculates and automatically checks the CRC to validate frame transmission. At the end of transmission, the status of a received frame is available in the status registers.

The SIO can be conveniently used under DMA control to provide high-speed reception or transmission. In reception, for example, the SIO can interrupt the CPU when the first character of a message is received. The CPU then enables the DMA to transfer the message to memory. The SIO then issues an end-of-frame interrupt and the CPU can check the status of the received message. Thus, the CPU is freed for other service while the message is being received.

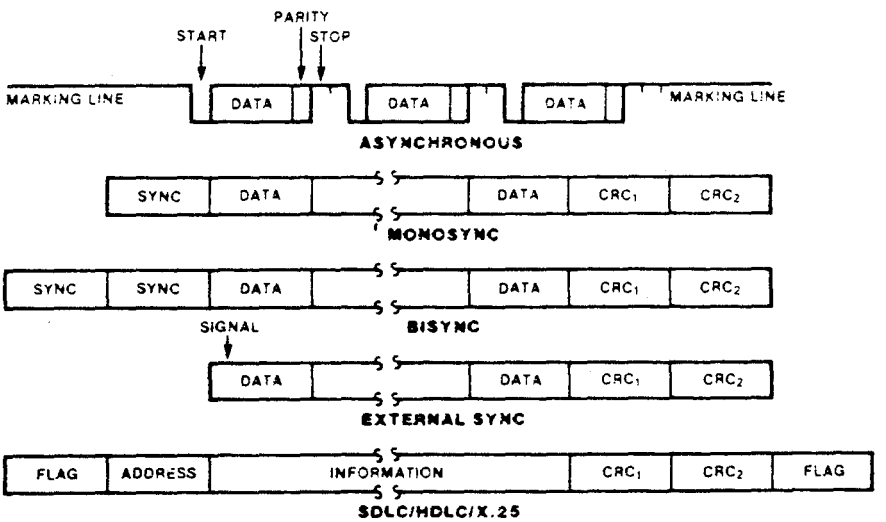


Figure 9. Some Z80 SIO Protocols

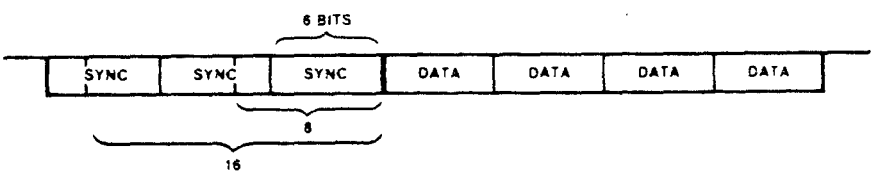


Figure 10.

'O Interface Capabilities

The SIO offers the choice of polling, interupt (vectored or non-vectored) and block-ansfer modes to transfer data, status and con-ol information to and from the CPU. The lock-transfer mode can also be implemented nder DMA control.

olling. Two status registers are updated at opropriate times for each function being per-rmed (for example, CRC error-status valid at e end of a message). When the CPU is oerated in a polling fashion, one of the SIO's o status registers is used to indicate whether e SIO has some data or needs some data. epending on the contents of this register, the PU will either write data, read data, or just o on. Two bits in the register indicate that a ata transfer is needed. In addition, error and her conditions are indicated. The second atus register (special receive conditions) does ot have to be read in a polling sequence, ntil a character has been received. All inter-pt modes are disabled when operating the evice in a polled environment.

terrupts. The SIO has an elaborate interrupt heme to provide fast interrupt service in al-time applications. A control register and a tus register in Channel B contain the inter-pt vector. When programmed to do so, the O can modify three bits of the interrupt vec-r in the status register so that it points direct-to one of eight interrupt service routines in emory, thereby servicing conditions in both annels and eliminating most of the needs for status-analysis routine.

Transmit interrupts, receive interrupts and ternal/status interrupts are the main sources interrupts. Each interrupt source is enabled der program control, with Channel A hav-a higher priority than Channel B, and with ceive, transmit and external/status interrupts oritized in that order within each channel. hen the transmit interrupt is enabled, the CPU is interrupted by the transmit buffer becoming empty. (This implies that the transmitter must have had a data character written into it so it can become empty.) The receiver can interrupt the CPU in one of two ways:

- Interrupt on first received character
- Interrupt on all received characters

Interrupt-on-first-received-character is typically used with the block-transfer mode. Interrupt-on-all-received-characters has the option of modifying the interrupt vector in the event of a parity error. Both of these interrupt modes will also interrupt under special receive conditions on a character or message basis (end-of-frame interrupt in SDLC, for example). This means that the special-receive condition can cause an interrupt only if the interrupt-on-first-received-character or interrupt-on-all-received-characters mode is selected. In interrupt-on-first-received-character, an interrupt can occur from special-receive conditions (except parity error) after the first-received-character interrupt (example: receive-overrun interrupt).

The main function of the external/status interrupt is to monitor the signal transitions of the Clear To Send ($\overline{CTS}$), Data Carrier Detect ($\overline{DCD}$) and Synchronization ($\overline{SYNC}$) pins (Figures 1 through 6). In addition, an external/status interrupt is also caused by a CRC-sending condition or by the detection of a break sequence (asynchronous mode) or abort sequence (SDLC mode) in the data stream. The interrupt caused by the break/abort sequence allows the SIO to interrupt when the break/abort sequence is detected or terminated. This feature facilitates the proper termination of the current message, correct initialization of the next message, and the accurate timing of the break/abort condition in external logic.

I/O Interface Capabilities (Continued)

In a Z80 CPU environment (Figure 11), SIO interrupt vectoring is "automatic": the SIO passes its internally-modifiable 8-bit interrupt vector to the CPU, which adds an additional 8 bits from its interrupt-vector (I) register to form the memory address of the interrupt-routine table. This table contains the address of the beginning of the interrupt routine itself. The process entails an indirect transfer of CPU control to the interrupt routine, so that the next instruction executed after an interrupt acknowledge by the CPU is the first instruction of the interrupt routine itself.

CPU/DMA Block Transfer. The SIO's block-transfer mode accommodates both CPU block transfers and DMA controllers (Z80 DMA or other designs). The block-transfer mode uses the Wait/Ready output signal, which is selected with three bits in an internal control register. The Wait/Ready output signal can be programmed as a $\overline{\text{WAIT}}$ line in the CPU block-transfer mode or as a $\overline{\text{READY}}$ line in the DMA block-transfer mode.

To a DMA controller, the SIO $\overline{\text{READY}}$ output indicates that the SIO is ready to transfer data to or from memory. To the CPU, the $\overline{\text{WAIT}}$ output indicates that the SIO is not ready to transfer data, thereby requesting the CPU to extend the I/O cycle.

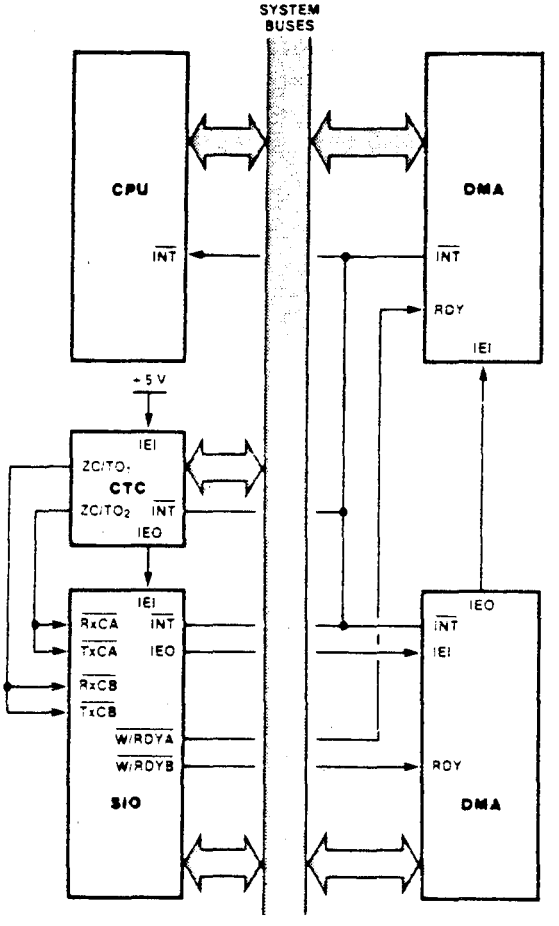


Figure 11. Typical Z80 Environment

ternal Structure

The internal structure of the device includes a 80 CPU interface, internal control and terrupt logic, and two full-duplex channels. ach channel contains its own set of control nd status (write and read) registers, and con- ol and status logic that provides the interface modems or other external devices.

The registers for each channel are desig- ated as follows:

WR0-WR7 — Write Registers 0 through 7
RR0-RR2 — Read Registers 0 through 2

The register group includes five 8-bit control gisters, two sync-character registers and two atus registers. The interrupt vector is written to an additional 8-bit register (Write Register in Channel B that may be read through other 8-bit register (Read Register 2) in hannel B. The bit assignment and functional ouping of each register is configured to nplify and organize the programming pro- ss. Table 1 lists the functions assigned to ch read or write register.

The logic for both channels provides for- ats, synchronization and validation for data nsferred to and from the channel interface. e modem control inputs, Clear To Send ($\overline{CTS}$) and Data Carrier Detect ($\overline{DCD}$), are monitored by the external control and status logic under program control. All external control-and-status-logic signals are general-purpose in nature and can be used for functions other than modem control.

Read Register Functions	
RR0	Transmit/Receive buffer status, interrupt status and external status
RR1	Special Receive Condition status
RR2	Modified interrupt vector (Channel B only)

Write Register Functions	
WR0	Register pointers, CRC initialize, initialization commands for the various modes, etc.
WR1	Transmit/Receive interrupt and data transfer mode definition.
WR2	Interrupt vector (Channel B only)
WR3	Receive parameters and control
WR4	Transmit/Receive miscellaneous parameters and modes
WR5	Transmit parameters and controls
WR6	Sync character or SDLC address field
WR7	Sync character or SDLC flag

Internal Structure (Continued)

Data Path. The transmit and receive data path illustrated for Channel A in Figure 12 is identical for both channels. The receiver has three 8-bit buffer registers in a FIFO arrangement, in addition to the 8-bit receive shift register. This scheme creates additional time for the CPU to service an interrupt at the beginning of a block of high-speed data. Incoming data is routed through one of several paths (data or CRC) depending on the selected mode and—in asynchronous modes—the character length.

The transmitter has an 8-bit transmit data buffer register that is loaded from the internal data bus, and a 20-bit transmit shift register that can be loaded from the sync-character buffers or from the transmit data register. Depending on the operational mode, outgoing data is routed through one of four main paths before it is transmitted from the Transmit Data output (TxD).

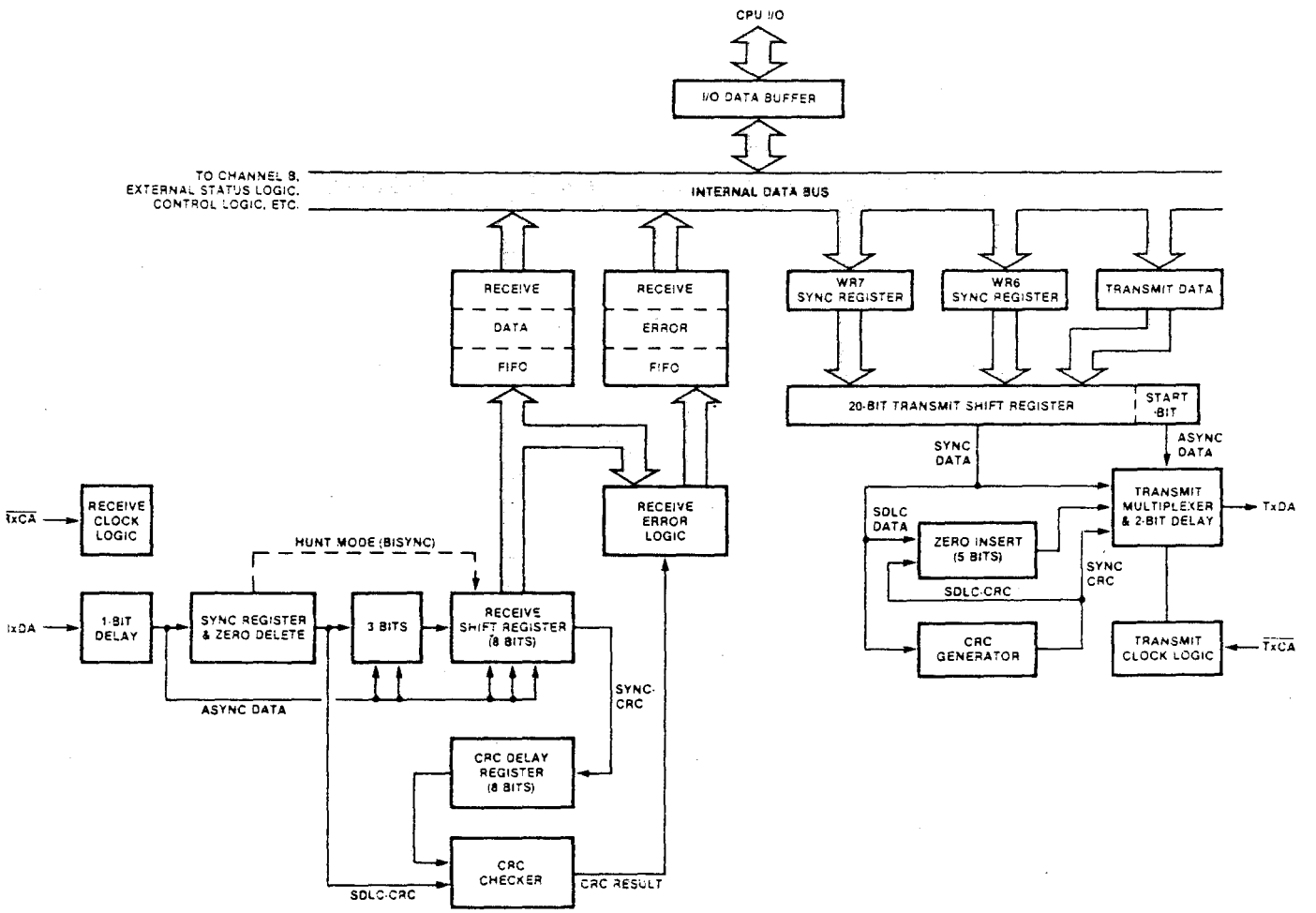


Figure 12. Transmit and Receive Data Path (Channel A)

rogramming

The system program first issues a series of ommands that initialize the basic mode of peration and then other commands that ualify conditions within the selected mode. or example, the asynchronous mode, haracter length, clock rate, number of stop ts, even or odd parity might be set first; then e interrupt mode; and finally, receiver or ansmitter enable.

Both channels contain registers that must be rogrammed via the system program prior to peration. The channel-select input (B/$\overline{A}$) and e control/data input (C/$\overline{D}$) are the command-ructure addressing controls, and are normal- controlled by the CPU address bus. Figures 5 and 16 illustrate the timing relationships for rogramming the write registers and transfer-ng data and status.

ead Registers. The SIO contains three read gisters for Channel B and two read registers r Channel A (RR0-RR2 in Figure 13) that can e read to obtain the status information; RR2 ntains the internally-modifiable interrupt ctor and is only in the Channel B register t. The status information includes error con-tions, interrupt vector and standard mmunications-interface signals.

To read the contents of a selected read gister other than RR0, the system program ust first write the pointer byte to WR0 in actly the same way as a write register opera-n. Then, by executing a read instruction, e contents of the addressed read register can read by the CPU.

The status bits of RR0 and RR1 are carefully ouped to simplify status monitoring. For ample, when the interrupt vector indicates it a Special Receive Condition interrupt has curred, all the appropriate error bits can be ad from a single register (RR1).

ite Registers. The SIO contains eight write gisters for Channel B and seven write gisters for Channel A (WR0-WR7 in Figure) that are programmed separately to con-ure the functional personality of the chan-ls; WR2 contains the interrupt vector for th channels and is only in the Channel B register set. With the exception of WR0, programming the write registers requires two bytes. The first byte is to WR0 and contains three bits (D_0-D_2) that point to the selected register; the second byte is the actual control word that is written into the register to configure the SIO.

WR0 is a special case in that all of the basic commands can be written to it with a single byte. Reset (internal or external) initializes the pointer bits D_0-D_2 to point to WR0. This implies that a channel reset must not be combined with the pointing to any register.

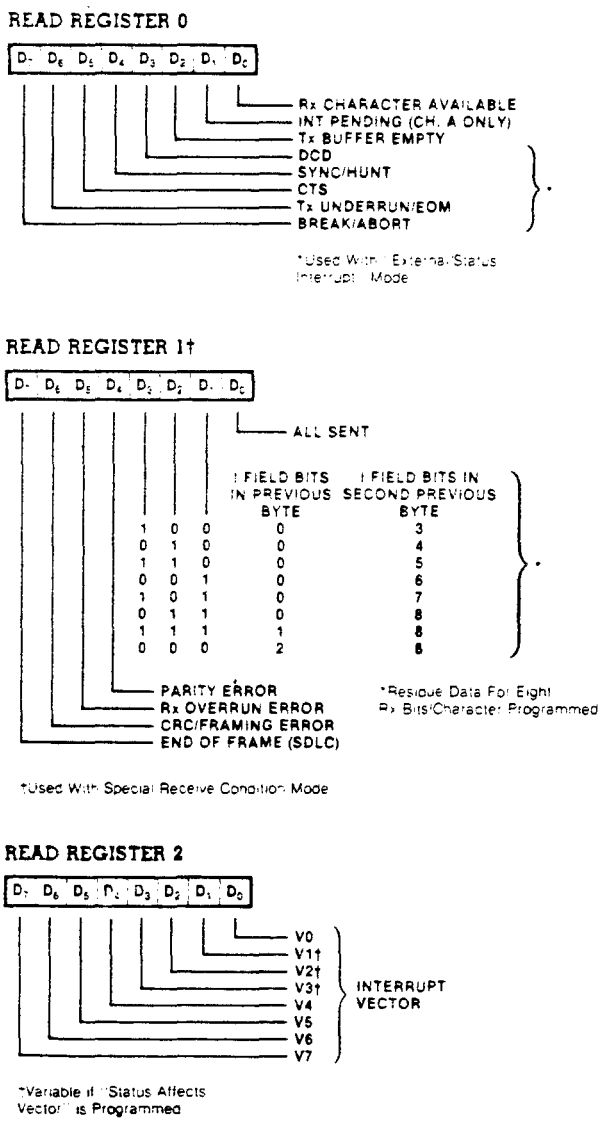
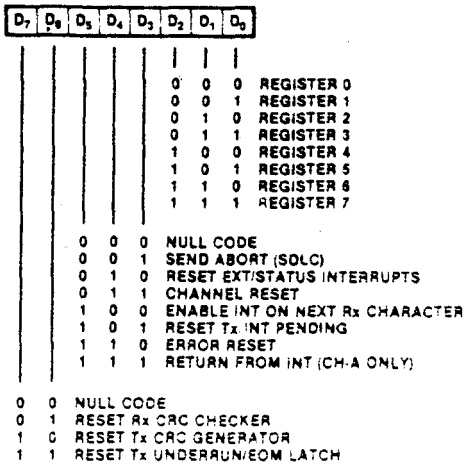


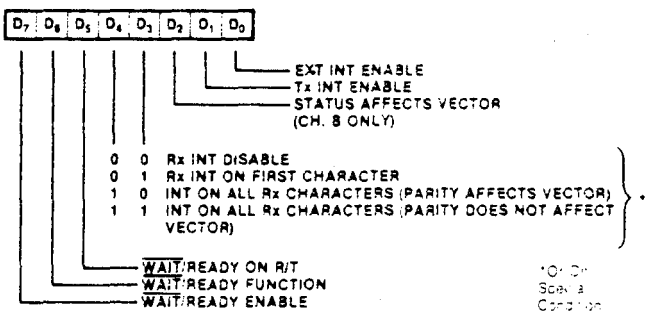
Figure 13. Read Register Bit Functions

Programming (Continued)

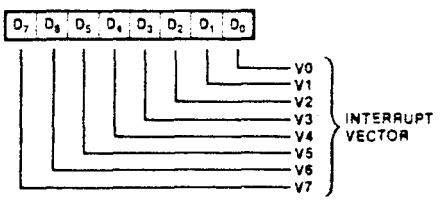
WRITE REGISTER 0



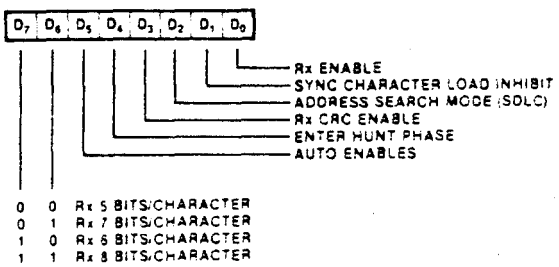
WRITE REGISTER 1



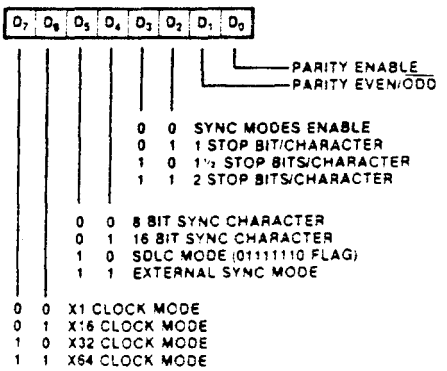
WRITE REGISTER 2 (CHANNEL B ONLY)



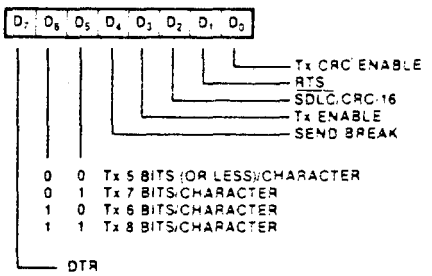
WRITE REGISTER 3



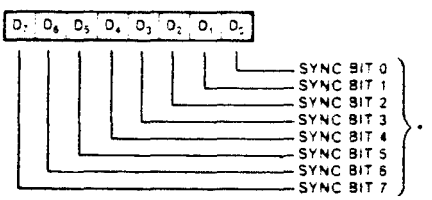
WRITE REGISTER 4



WRITE REGISTER 5

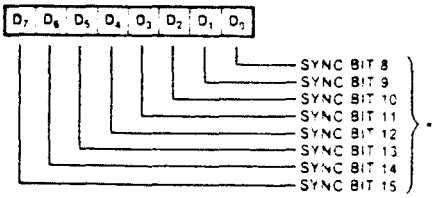


WRITE REGISTER 6



*Also SDLC Address Field

WRITE REGISTER 7



*For SDLC It Must Be Programmed to "01111110" For Flag Recognition

Figure 14. Write Register Bit Functions

ıming

The SIO must have the same clock as the PU (same phase and frequency relationship, ot necessarily the same driver).

ead Cycle. The timing signals generated by Z-80 CPU input instruction to read a data or atus byte from the SIO are illustrated in gure 15.

rite Cycle. Figure 16 illustrates the timing nd data signals generated by a Z-80 CPU out- ut instruction to write a data or control byte to the SIO.

terrupt-Acknowledge Cycle. After receiv- g an interrupt-request signal from an SIO NT pulled Low), the Z-80 CPU sends an terrupt-acknowledge sequence ($\overline{M1}$ Low, and $\overline{IORQ}$ Low a few cycles later) as in Figure 17. The SIO contains an internal daisy-chained terrupt structure for prioritizing nested inter- pts for the various functions of its two chan- els, and this structure can be used within n external user-defined daisy chain that rioritizes several peripheral circuits. The IEI of the highest-priority device is rminated High. A device that has an inter- pt pending or under service forces its IEO w. For devices with no interrupt pending or nder service, IEO = IEI. To insure stable conditions in the daisy ain, all interrupt status signals are pre- nted from changing while $\overline{M1}$ is Low. When $\overline{IORQ}$ is Low, the highest priority interrupt requestor (the one with IEI High) places its interrupt vector on the data bus and sets its internal interrupt-under-service latch.

Return From Interrupt Cycle. Figure 18 illustrates the return from interrupt cycle. Normally, the Z-80 CPU issues a RETI (Return From Interrupt) instruction at the end of an interrupt service routine. RETI is a 2-byte opcode (ED-4D) that resets the interrupt-under-service latch in the SIO to terminate the interrupt that has just been processed. This is accomplished by manipulating the daisy chain in the following way.

The normal daisy-chain operation can be used to detect a pending interrupt; however, it cannot distinguish between an interrupt under service and a pending unacknowledged interrupt of a higher priority. Whenever "ED" is decoded, the daisy chain is modified by forcing High the IEO of any interrupt that has not yet been acknowledged. Thus the daisy chain identifies the device presently under service as the only one with an IEI High and an IEO Low. If the next opcode byte is "4D," the interrupt-under-service latch is reset.

The ripple time of the interrupt daisy chain (both the High-to-Low and the Low-to-High transitions) limits the number of devices that can be placed in the daisy chain. Ripple time can be improved with carry-look-ahead, or by extending the interrupt-acknowledge cycle.

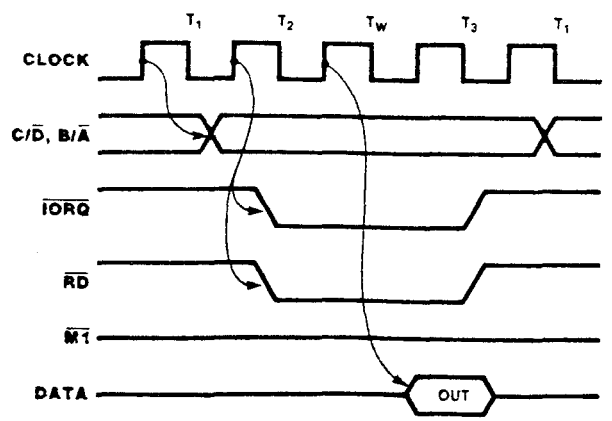


Figure 15. Read Cycle

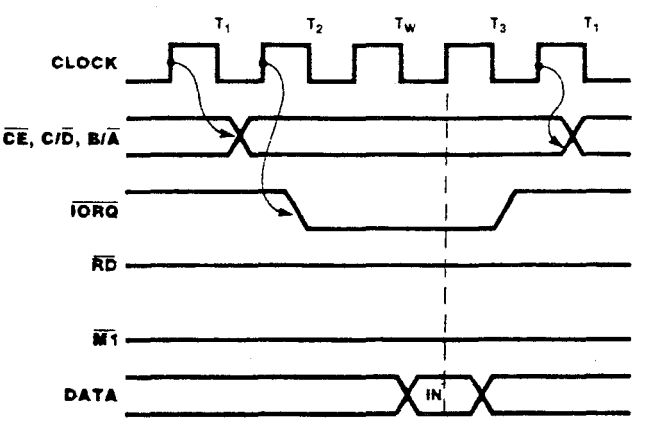


Figure 16. Write Cycle

Timing (Continued)

For further information about techniques for increasing the number of daisy-chained devices, refer to the *Z80 CPU Data Sheet.*

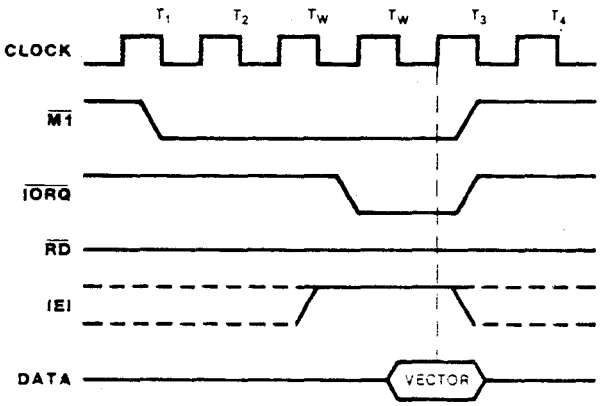


Figure 17. Interrupt Acknowledge Cycle

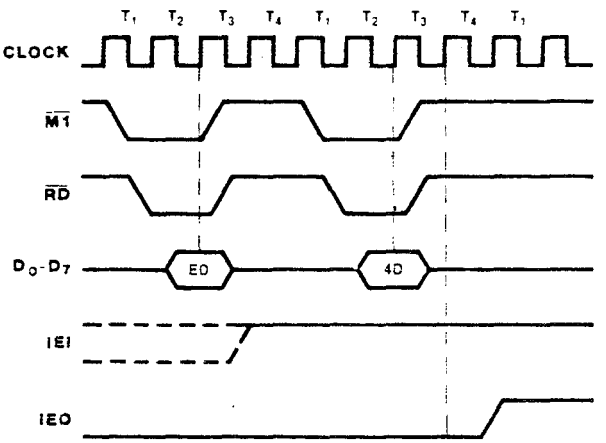


Figure 18. Return from Interrupt Cycle

Absolute Maximum Ratings

Voltages on all inputs and outputs with respect to GND . . . . . -0.3 V to +7.0 V

Operating Ambient Temperature . . . . . As Specified in Ordering Information

Storage Temperature . . . . . -65°C to +150°C

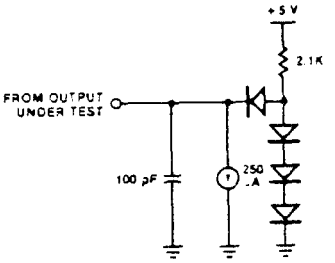
Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Test Conditions

The characteristics below apply for the following test conditions, unless otherwise noted. All voltages are referenced to GND (0 V). Positive current flows into the referenced pin. Available operating temperature ranges are:

- 0°C to +70°C, $+4.75\ V \leq V_{CC} \leq +5.25\ V$
- −40°C to +85°C, $+4.75\ V \leq V_{CC} \leq +5.25\ V$
- −55°C to +125°C, $+4.5\ V \leq V_{CC} \leq +5.5\ V$

The product number for each operating temperature range may be found in the ordering information section.



DC Characteristics

Symbol	Parameter	Min	Max	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3	+0.45	V	
V_{IHC}	Clock Input High Voltage	V_{CC}-0.6	+5.5	V	
V_{IL}	Input Low Voltage	-0.3	+0.8	V	
V_{IH}	Input High Voltage	+2.0	+5.5	V	
V_{OL}	Output Low Voltage		+0.4	V	I_{OL} = 2.0 mA
V_{OH}	Output High Voltage	+2.4		V	I_{OH} = -250 μA
I_{LI}	Input Leakage Current	-10	+10	μA	$0 < V_{IN} < V_{CC}$
I_Z	3-State Output/Data Bus Input Leakage Current	-10	+10	μA	$0 < V_{IN} < V_{CC}$
$I_{L(SY)}$	$\overline{SYNC}$ Pin Leakage Current	-40	+10	μA	$0 < V_{IN} < V_{CC}$
I_{CC}	Power Supply Current		100	mA	

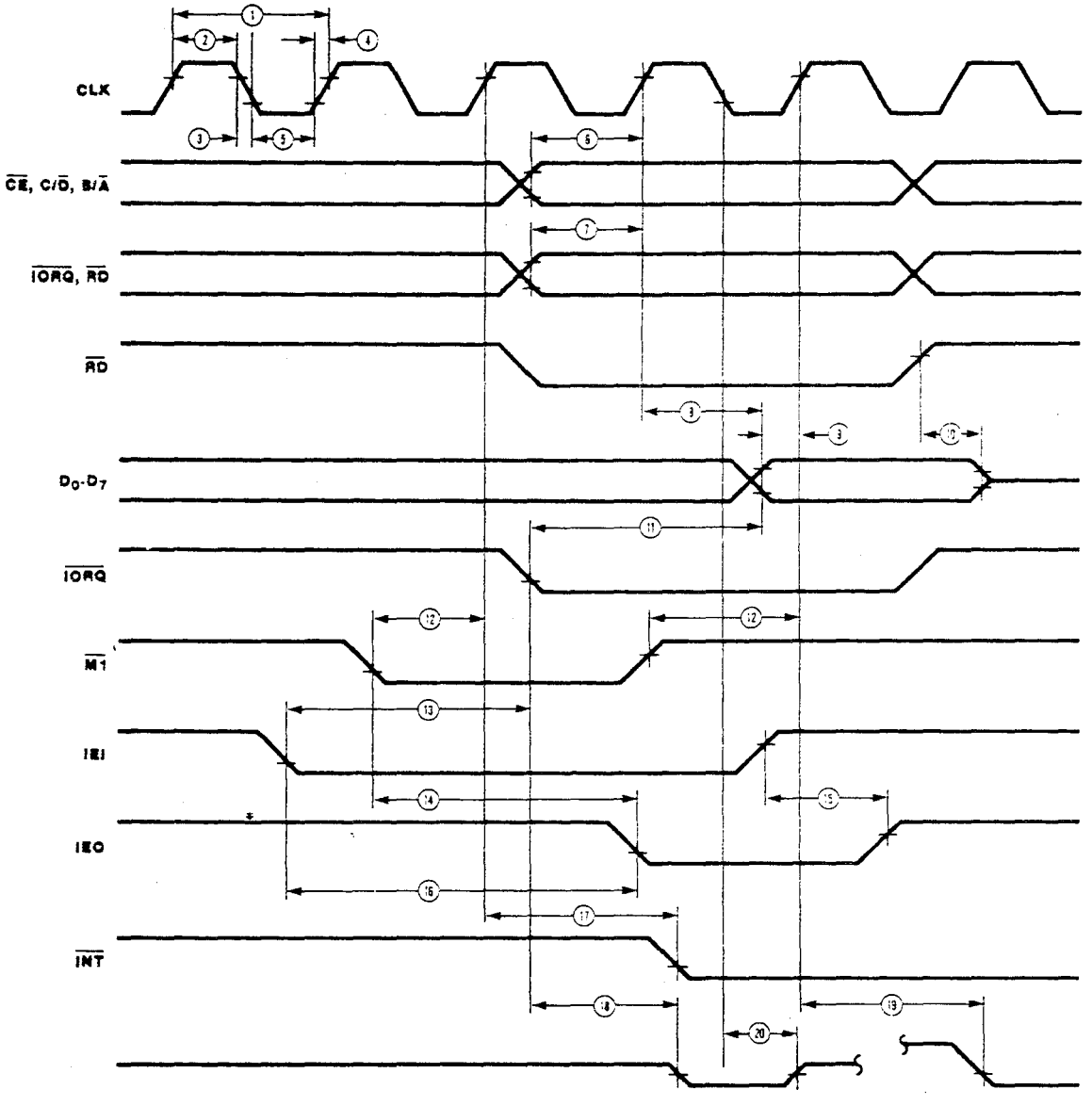
Over specified temperature and voltage range.

Capacitance

Symbol	Parameter	Max	Unit	Test Condition
C	Clock Capacitance	40	pF	Unmeasured
C_{IN}	Input Capacitance	5	pF	pins returned
C_{OUT}	Output Capacitance	10	pF	to ground

Over specified temperature range; f = 1MHz

AC Electrical Characteristics



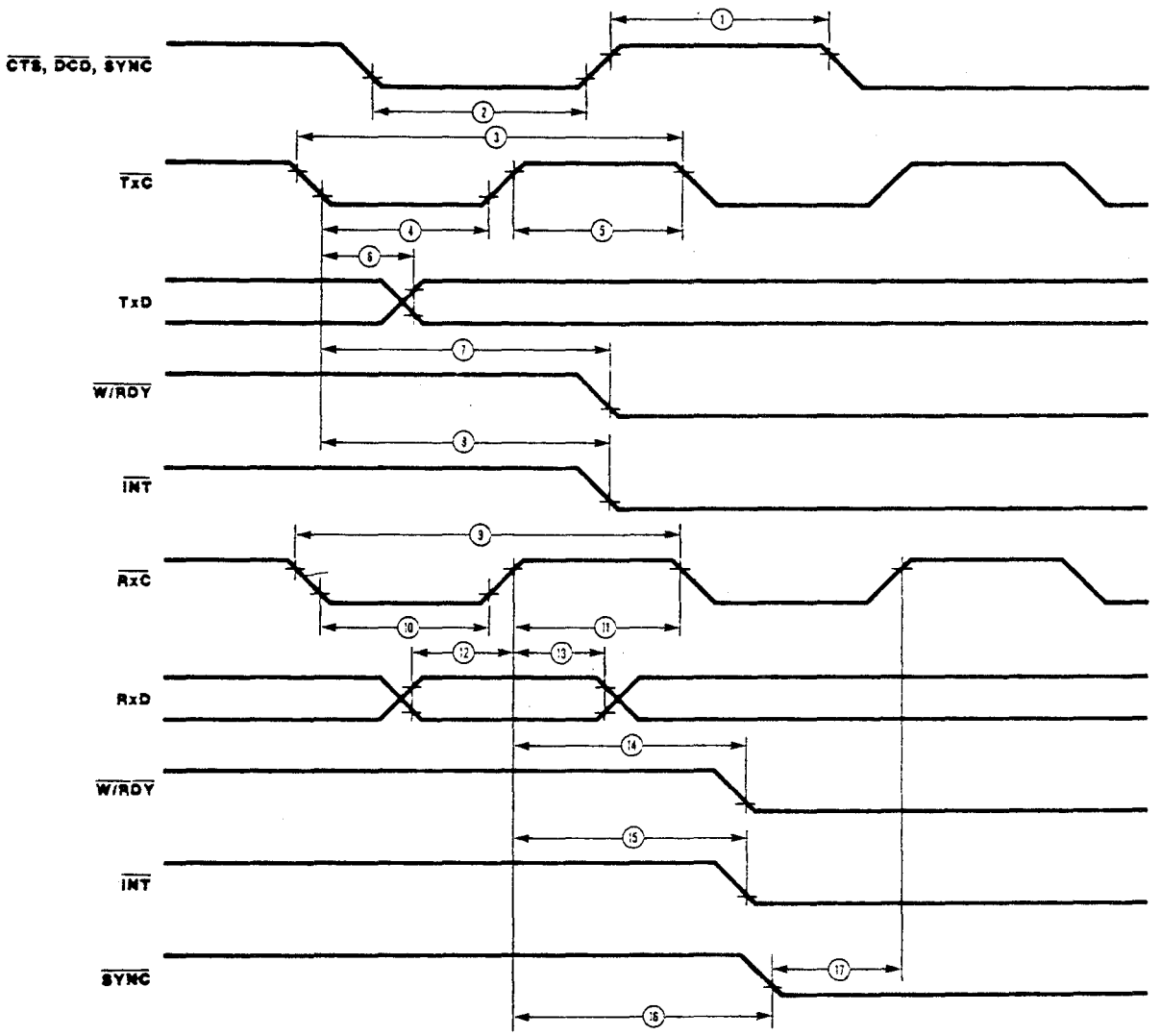
AC Electrical Characteristics (Continued)

Number	Symbol	Parameter	Z8440, 1, 2 Z80 SIO Min	Z8440, 1, 2 Z80 SIO Max	Z8440, 1, 2A Z80A SIO Min	Z8440, 1, 2A Z80A SIO Max	Z8440, 1, 2B Z80B SIO Min	Z8440, 1, 2B Z80B SIO Max	Unit
1	TcC	Clock Cycle Time	400	4000	250	4000	165	4000	ns
2	TwCh	Clock Width (High)	170	2000	105	2000	70	2000	ns
3	TfC	Clock Fall Time		30		30		15	ns
4	TrC	Clock Rise Time		30		30		15	ns
5	TwCl	Clock Width (Low)	170	2000	105	2000	70	2000	ns
6	TsAD(C)	$\overline{CE}$, C/$\overline{D}$, B/$\overline{A}$ to Clock ↑ Setup Time	160		145		60		ns
7	TsCS(C)	$\overline{IORQ}$, $\overline{RD}$ to Clock ↑ Setup Time	240		115		60		ns
8	TdC(DO)	Clock ↑ to Data Out Delay		240		220		150	ns
9	TsDI(C)	Data In to Clock ↑ Setup (Write or $\overline{M1}$ Cycle)	50		50		30		ns
10	TdRD(DOz)	$\overline{RD}$ ↑ to Data Out Float Delay		230		110		90	ns
11	TdIO(DOI)	$\overline{IORQ}$ ↓ to Data Out Delay (INTACK Cycle)		340		160		100	ns
12	TsM1(C)	$\overline{M1}$ to Clock ↑ Setup Time	210		90		75		ns
13	TsIEI(IO)	IEI to $\overline{IORQ}$ ↓ Setup Time (INTACK Cycle)	200		140		120		ns
14	TdM1(IEO)	$\overline{M1}$ ↓ to IEO ↓ Delay (interrupt before $\overline{M1}$)		300		190	160		ns
15	TdIEI(IEOr)	IEI ↑ to IEO ↑ Delay (after ED decode)		150		100		70	ns
16	TdIEI(IEOf)	IEI ↓ to IEO ↓ Delay		150		100		70	ns
17	TdC(INT)	Clock ↑ to $\overline{INT}$ ↓ Delay		200		200		150	ns
18	TdIO(W/RWf)	$\overline{IORQ}$ ↓ or $\overline{CE}$ ↓ to $\overline{W}$/$\overline{RDY}$ ↓ Delay Wait Mode)		300		210		175	ns
19	TdC(W/RR)	Clock ↑ to $\overline{W}$/$\overline{RDY}$ ↓ Delay (Ready Mode)		120		120		100	ns
20	TdC(W/RWz)	Clock ↓ to $\overline{W}$/$\overline{RDY}$ Float Delay (Wait Mode)		150		130		110	ns
21	Th	Any unspecified Hold when Setup is specified	0		0		0		ns

AC Electrical Characteristics (Continued)

Number	Symbol	Parameter	Z8440, 1, 2 Z80 SIO Min	Z8440, 1, 2 Z80 SIO Max	Z8440, 1, 2A Z80A SIO Min	Z8440, 1, 2A Z80A SIO Max	Z8440, 1, 2B Z80B SIO Min	Z8440, 1, 2B Z80B SIO Max	Unit
1	TcC	Clock Cycle Time	400	4000	250	4000	165	4000	ns
2	TwCh	Clock Width (High)	170	2000	105	2000	70	2000	ns
3	TfC	Clock Fall Time		30		30		15	ns
4	TrC	Clock Rise Time		30		30		15	ns
5	TwCl	Clock Width (Low)	170	2000	105	2000	70	2000	ns
6	TsAD(C)	$\overline{CE}$, C/$\overline{D}$, B/$\overline{A}$ to Clock ↑ Setup Time	160		145		60		ns
7	TsCS(C)	$\overline{IORQ}$, $\overline{RD}$ to Clock ↑ Setup Time	240		115		60		ns
8	TdC(DO)	Clock ↑ to Data Out Delay		240		220		150	ns
9	TsDI(C)	Data In to Clock ↑ Setup (Write or $\overline{M1}$ Cycle)	50		50		30		ns
10	TdRD(DOz)	$\overline{RD}$ ↑ to Data Out Float Delay		230		110		90	ns
11	TdIO(DOI)	$\overline{IORQ}$ ↓ to Data Out Delay (INTACK Cycle)		340		160		100	ns
12	TsM1(C)	$\overline{M1}$ to Clock ↑ Setup Time	210		90		75		ns
13	TsIEI(IO)	IEI to $\overline{IORQ}$ ↓ Setup Time (INTACK Cycle)	200		140		120		ns
14	TdM1(IEO)	$\overline{M1}$ ↓ to IEO ↓ Delay (interrupt before $\overline{M1}$)		300		190	160		ns
15	TdIEI(IEOr)	IEI ↑ to IEO ↑ Delay (after ED decode)		150		100		70	ns
16	TdIEI(IEOf)	IEI ↓ to IEO ↓ Delay		150		100		70	ns
17	TdC(INT)	Clock ↑ to $\overline{INT}$ ↓ Delay		200		200		150	ns
18	TdIO(W/RWf)	$\overline{IORQ}$ ↓ or $\overline{CE}$ ↓ to W/$\overline{RDY}$ ↓ Delay Wait Mode)		300		210		175	ns
19	TdC(W/RR)	Clock ↑ to W/$\overline{RDY}$ ↓ Delay (Ready Mode)		120		120		100	ns
20	TdC(W/RWz)	Clock ↓ to W/$\overline{RDY}$ Float Delay (Wait Mode)		150		130		110	ns
21	Th	Any unspecified Hold when Setup is specified	0		0		0		ns

AC Electrical Characteristics (Continued)



EK 5: SISTEM PROGRAMLARI

```
     CLS
     DIM A(26), B(26), C(26), D(26), E(26), F(26), H(26), MAT(26)
     COM(1) ON
     OPEN "COM1:1200,N,8,1,CS,DS,CD" FOR RANDOM AS #1
     ON COM(1) GOSUB 1500: GOTO 3
1500 XPR = INP(&H3F8): RETURN
3    CLS
     FOR I = 1 TO 26
       READ A(I), B(I), C(I), D(I), E(I), F(I), H(I)
     NEXT I
1    CLS
  PR = 0
  LOCATE 8, 15: PRINT "19:00 - 06:00 ARASI :    PR NO: 1 "
  LOCATE 9, 15: PRINT "06:00 - 07:00 ARASI :    PR NO: 2 "
  LOCATE 10, 15: PRINT "07:00 - 08:00 ARASI :    PR NO: 3 "
  LOCATE 11, 15: PRINT "08:00 - 09:00 ARASI :    PR NO: 4 "
  LOCATE 12, 15: PRINT "09:00 - 12:00 ARASI :    PR NO: 5"
  LOCATE 13, 15: PRINT "12:00 - 14:00 ARASI :    PR NO: 6"
  LOCATE 14, 15: PRINT "14:00 - 17:00 ARASI :    PR NO: 5"
  LOCATE 15, 15: PRINT "17:00 - 19:00 ARASI :    PR NO: 7"
  LOCATE 24, 8: PRINT " PROGRAM ACIKLAMASI ' T ' ";
                PRINT " CIKIS ICIN ' Q ' TUSUNA BASIN"
5 XA$ = INKEY$
  IF XA$ = "Q" OR XA$ = "q" THEN GOTO 2000
  IF XA$ = "T" OR XA$ = "t" THEN GOTO 3000
  IF MID$(TIME$, 1, 2) < "06" THEN GOSUB 100
  IF MID$(TIME$, 1, 2) >= "06" AND MID$(TIME$, 1, 2) < "07" THEN GOSUB 110
  IF MID$(TIME$, 1, 2) >= "07" AND MID$(TIME$, 1, 2) < "08" THEN GOSUB 120
  IF MID$(TIME$, 1, 2) >= "08" AND MID$(TIME$, 1, 2) < "09" THEN GOSUB 130
  IF MID$(TIME$, 1, 2) >= "09" AND MID$(TIME$, 1, 2) < "12" THEN GOSUB 140
  IF MID$(TIME$, 1, 2) >= "12" AND MID$(TIME$, 1, 2) < "14" THEN GOSUB 150
  IF MID$(TIME$, 1, 2) >= "14" AND MID$(TIME$, 1, 2) < "17" THEN GOSUB 140
  IF MID$(TIME$, 1, 2) >= "17" AND MID$(TIME$, 1, 2) < "19" THEN GOSUB 160
  IF MID$(TIME$, 1, 2) >= "19" THEN GOSUB 100
  LOCATE 1, 72: PRINT TIME$
  LOCATE 2, 72: PRINT "PR NO:"; PR
  GOTO 5
100 IF PR = 1 THEN RETURN
     PR = 1
      FOR I = 1 TO 26
        MAT(I) = F(I)
      NEXT I
      GOSUB 1000
      RETURN
110 IF PR = 2 THEN RETURN
     PR = 2
      FOR I = 1 TO 26
       MAT(I) = A(I)
      NEXT I
      GOSUB 1000
      RETURN
120 IF PR = 3 THEN RETURN
     PR = 3
      FOR I = 1 TO 26
       MAT(I) = B(I)
      NEXT I
      GOSUB 1000
      RETURN
```

```
130 IF PR = 4 THEN RETURN
    PR = 4
     FOR I = 1 TO 26
      MAT(I) = C(I)
     NEXT I
     GOSUB 1000
     RETURN
140 IF PR = 5 THEN RETURN
    PR = 5
     FOR I = 1 TO 26
      MAT(I) = D(I)
     NEXT I
     GOSUB 1000
     RETURN
150 IF PR = 6 THEN RETURN
    PR = 6
     FOR I = 1 TO 26
      MAT(I) = E(I)
     NEXT I
     GOSUB 1000
     RETURN
160 IF PR = 7 THEN RETURN
    PR = 7
     FOR I = 1 TO 26
      MAT(I) = H(I)
     NEXT I
     GOSUB 1000
     RETURN
1000 FOR I = 1 TO 26
     PRINT #1, CHR$(MAT(I));
    NEXT I
    RETURN
3000 CLS
     LOCATE 5, 17: PRINT "CALISMAKTA OLAN PROGRAMIN ACIKLAMASI"
     IF PR = 1 THEN 22 ELSE IF PR = 2 THEN 23 ELSE IF PR = 3 THEN 24
     IF PR = 4 THEN 25 ELSE IF PR = 5 THEN 26 ELSE IF PR = 6 THEN 27
       IF PR = 7 THEN 28 ELSE GOTO 7
22  LOCATE 10, 10: PRINT " SU ANDA LAMBALAR SARI YANIP SONMEKTEDIR "
    GOTO 7
23  LOCATE 10, 10: PRINT " Devre Suresi          : 26 sn "
    LOCATE 11, 10: PRINT " Ana Yol Gecis Suresi  :  8 sn "
    LOCATE 12, 10: PRINT " Tali Yol Gecis Suresi :  6 sn "
    LOCATE 13, 10: PRINT " Yaya Gecis Suresi     :  6 sn "
    LOCATE 14, 10: PRINT " Koruma Sureleri Top.  :  6 sn "
    GOTO 7
24  LOCATE 10, 10: PRINT " Devre Suresi          : 52 sn "
    LOCATE 11, 10: PRINT " Ana Yol Gecis Suresi  : 20 sn "
    LOCATE 12, 10: PRINT " Tali Yol Gecis Suresi : 18 sn "
    LOCATE 13, 10: PRINT " Yaya Gecis Suresi     :  8 sn "
    LOCATE 14, 10: PRINT " Koruma Sureleri Top.  :  6 sn "
    GOTO 7
25  LOCATE 10, 10: PRINT " Devre Suresi          : 42 sn "
    LOCATE 11, 10: PRINT " Ana Yol Gecis Suresi  : 15 sn "
    LOCATE 12, 10: PRINT " Tali Yol Gecis Suresi : 13 sn "
    LOCATE 13, 10: PRINT " Yaya Gecis Suresi     :  8 sn "
    LOCATE 14, 10: PRINT " Koruma Sureleri Top.  :  6 sn "
    GOTO 7
```

```
26  LOCATE 10, 10: PRINT " Devre Suresi          : 32 sn "
    LOCATE 11, 10: PRINT " Ana Yol Gecis Suresi  : 10 sn "
    LOCATE 12, 10: PRINT " Tali Yol Gecis Suresi :  8 sn "
    LOCATE 13, 10: PRINT " Yaya Gecis Suresi     :  8 sn "
    LOCATE 14, 10: PRINT " Koruma Sureleri Top.  :  6 sn "
    GOTO 7
27  LOCATE 10, 10: PRINT " Devre Sure            : 47 sn "
    LOCATE 11, 10: PRINT " Ana Yol Gecis Suresi  : 18 sn "
    LOCATE 12, 10: PRINT " Tali Yol Gecis Suresi : 15 sn "
    LOCATE 13, 10: PRINT " Yaya Gecis Suresi     :  8 sn "
    LOCATE 14, 10: PRINT " Koruma Sureleri Top.  :  6 sn "
    GOTO 7
28  LOCATE 10, 10: PRINT " Devre Suresi          : 60 sn "
    LOCATE 11, 10: PRINT " Ana Yol Gecis Suresi  : 24 sn "
    LOCATE 12, 10: PRINT " Tali Yol Gecis Suresi : 20 sn "
    LOCATE 13, 10: PRINT " Yaya Gecis Suresi     : 10 sn "
    LOCATE 14, 10: PRINT " Koruma Sureleri Top.  :  6 sn "
    GOTO 7
7    LOCATE 24, 8: PRINT "Press any key to continue"
6     A$ = INKEY$
     IF A$ = "" THEN 6 ELSE 1

2000 CLOSE #1
     END
        DATA 16,40,30,20,35,3,45
        DATA 3,3,3,3,3,3,3
        DATA 3,3,3,3,3,3,3
        DATA 12,36,26,16,30,3,40
        DATA 3,3,3,3,3,3,3
        DATA 12,16,16,16,16,3,20
        DATA 3,3,3,3,3,3,3
        DATA 3,3,3,3,3,3,3
        DATA 2,3,4,5,6,1,7
        DATA 16,40,30,20,35,3,45
        DATA 3,3,3,3,3,3,3
        DATA 16,40,30,20,35,3,45
        DATA 3,3,3,3,3,3,3
        DATA 12,20,20,20,20,3,30
        DATA 3,3,3,3,3,3,3
        DATA 3,3,3,3,3,3,3
        DATA 2,3,4,5,6,1,7
        DATA 16,40,30,20,35,3,45
        DATA 3,3,3,3,3,3,3
        DATA 3,3,3,3,3,3,3
        DATA 12,36,26,16,30,3,40
        DATA 3,3,3,3,3,3,3
        DATA 12,16,16,16,16,3,20
        DATA 3,3,3,3,3,3,3
        DATA 3,3,3,3,3,3,3
        DATA 2,3,4,5,6,1,7
```

```
;************************************
;
; Z-80 MICROCOMPUTER SYSTEM PROGRAM
; MASTER CARD PROGRAMI
; Halil KARAKAS
; EQUATE TABLES

MAIN:           EQU     0100H
STACKTOP:       EQU     9FEFH
DIGBUF:         EQU     9FF0H
HEXBUF:         EQU     9FF6H
MAFLAG:         EQU     9FF9H
MADR:           EQU     9FFAH
KEYCOUNT:       EQU     9FFCH
MESSTART:       EQU     9FFDH
CURRENTOR:      EQU     9FFFH

                ORG 00H
                JP MAIN
        CODES:  DB 18H,14H,04H,13H,0C1H,15H,68H
        ;************************************

                ORG 0020H
                DI
                JP 1000H

        ;************************************

                ORG 0028H
                DI
                JP 1100H

        ;*****************************************

                ORG 0030H
                DI
                JP 1200H

        ;*****************************************

                ORG 0038H
                DI
                JP 1300H

        ;*****************************************

                ORG 0100H
                LD SP,STACKTOP
                LD A,55H
                OUT (20H),A
                NOP
                NOP
                OUT (21H),A
                LD A,6FH
                OUT (20H),A
                LD A,0FH
```

```
                OUT (21H),A
                LD C,43H
                LD B,7
                LD HL,CODES
                OTIR
                IM 0
                EI
                LD A,0FFH
                OUT (01H),A
                LD A,1FH
                OUT (01H),A
                LD A,3FH
                OUT (03H),A
                LD A,01H
                LD (MAFLAG),A
                LD HL,DIGBUF
                LD (MESSTART),HL
                XOR A
                LD (CURRENTOR),A
                LD HL,0000H
                LD (MADR),HL
                LD (HEXBUF),HL
                CALL CONV7SEG
                LD HL,0000H
                LD (DIGBUF+4),HL
MAINLOOP:       LD HL,0100H
                CALL DELAYX
                CALL KEYSTATUS
                CALL Z,KEYPROG
                CALL PRINTDIG
                JP MAINLOOP
CONV7SEG:       LD DE,HEXBUF
                LD HL,DIGBUF
                LD C,03H
CNVLOOP:        PUSH HL
                LD A,(DE)
                AND 0F0H
                RRCA
                RRCA
                RRCA
                RRCA
                LD HL,SEGMTABLE
                CALL ADDAHL
                LD B,(HL)
                LD A,(DE)
                AND 0FH
                LD HL,SEGMTABLE
                CALL ADDAHL
                LD A,(HL)
                POP HL
                LD (HL),B
                INC HL
                LD (HL),A
                INC HL
                INC DE
                DEC C
```

```
                JP NZ,CNVLOOP
                RET
SEGMTABLE:      DB 3FH,11H,5EH,5BH,71H
                DB 6BH,6FH,39H,7FH,79H
                DB 7DH,67H,2EH,57H,6EH,6CH
ADDAHL:         ADD A,L
                LD L,A
                RET NC
                INC H
                RET
DELAYX:         DEC HL
                LD A,H
                OR L
                JP NZ,DELAYX
                RET
KEYSTATUS:      LD A,00H
                OUT (02H),A
                OUT (00H),A
                IN A,(00H)
                OR 0E0H
                CP 0FFH
                JP NZ,KEYWAIT
                LD A,00H
                LD (KEYCOUNT),A
                INC A
                RET
KEYWAIT:        LD A,(KEYCOUNT)
                CP 19H
                JP NZ,KEYW1
                INC A
                RET
KEYW1:          INC A
                LD (KEYCOUNT),A
                CP 19H
                RET NZ
FINDKEY:        LD A,00H
                OUT (02H),A
                OUT (00H),A
                IN A,(00H)
                OR 0E0H
                LD B,A
                LD C,04H
                LD A,0FEH
FINDLP:         CP B
                JP Z,FINDCOL
                RLCA
                DEC C
                JP NZ,FINDLP
FINDCOL:        LD A,80H
                OUT (02H),A
                LD A,0C0H
                OUT (00H),A
                IN A,(00H)
                OR 0E0H
                CP 0FFH
                LD B,00H
```

```
                JP NZ,ROWFOUND
                LD B,05H
                LD A,0A0H
                OUT (00H),A
                IN A,(00H)
                OR 0E0H
                CP 0FFH
                JP NZ,ROWFOUND
                LD B,0AH
                LD A,60H
                OUT (00H),A
                IN A,(00H)
                OR 0E0H
                CP 0FFH
                JP NZ,ROWFOUND
                LD B,0FH
ROWFOUND:       LD A,C
                ADD A,B
                LD B,A
                XOR A
                LD A,B
                RET
PRINTDIG:       LD HL,(MESSTART)
                LD A,(CURRENTOR)
                LD B,A
                CALL ADDAHL
                LD A,(HL)
                OUT (02H),A
                LD A,B
                ADD A,A
                ADD A,A
                ADD A,A
                ADD A,A
                ADD A,A
                OUT (00H),A
                LD A,B
                INC A
                CP 06H
                LD (CURRENTOR),A
                RET C
                XOR A
                LD (CURRENTOR),A
                RET
KEYPROG:        CP 10H
                JP NC,FUNKEY
                LD B,A
                LD A,(MAFLAG)
                CP 01H
                JP Z,ROLA4
                LD HL,(MADR)
                LD A,(HL)
                ADD A,A
                ADD A,A
                ADD A,A
                ADD A,A
                OR B
```

```
MEMMOD:     LD (HL),A
            LD HL,(MADR)
            LD A,(HL)
            LD (HEXBUF+2),A
            LD HL,(MADR)
            LD A,H
            LD (HEXBUF),A
            LD A,L
            LD (HEXBUF+1),A
            CALL CONV7SEG
            RET
ROLA4:      LD A,B
            LD HL,(MADR)
            ADD HL,HL
            ADD HL,HL
            ADD HL,HL
            ADD HL,HL
            OR L
            LD L,A
ADDRMOD:    LD (MADR),HL
            LD HL,(MADR)
            LD A,H
            LD H,L
            LD L,A
            LD (HEXBUF),HL
            CALL CONV7SEG
            LD HL,00H
            LD (DIGBUF+4),HL
            RET
FUNKEY:     AND 1FH
            CP 10H
            JP NZ,FUNC1
            LD A,(MAFLAG)
            AND 01H
            XOR 01H
            LD (MAFLAG),A
            JP Z,MEMMOD
            JP ADDRMOD
FUNC1:      CP 11H
            JP NZ,FUNC2
            LD HL,(MADR)
            INC HL
            LD (MADR),HL
            LD A,(MAFLAG)
            CP 00H
            JP Z,MEMMOD
            JP ADDRMOD
FUNC2:      CP 12H
            JP NZ,FUNC3
            LD HL,(MADR)
            DEC HL
            LD (MADR),HL
            LD A,(MAFLAG)
            CP 00H
            JP Z,MEMMOD
            JP ADDRMOD
```

```
FUNC3:          LD HL,(MADR)
                JP (HL)

        ;****************************************

                ORG 1000H
                PUSH AF
                PUSH HL
                PUSH BC
                PUSH DE
                LD A,03H
                OUT (62H),A
                LD C,43H
                LD DE,9A00H
                LD B,26
        POLL1:  LD A,0
                OUT (C),A
        POLL:   IN A,(C)
                BIT 0,A
                JP Z,POLL
                IN A,(42H)
                LD (DE),A
                LD A,30H
                OUT (C),A
                INC DE
                DJNZ POLL1
                CALL TRANS
                POP DE
                POP BC
                POP HL
                POP AF
                EI
                RET

        ;****************************************

                ORG 1100H
                PUSH AF
                PUSH HL
                PUSH BC
                PUSH DE
                LD A,02H
                OUT (62H),A
                LD C,43H
                LD A,0
                OUT (C),A
        PS1:    IN A,(C)
                BIT 0,A
                JP Z,PS1
                IN A,(42H)
                PUSH AF
                LD A,30H
                OUT (C),A
                POP AF
                LD HL,9008H
```

```
        CP (HL)
        CALL NZ,SC4TR
        CALL PCTRANS
        POP DE
        POP BC
        POP HL
        POP AF
        EI
        RET

;*************************************

        ORG 1200H
        PUSH AF
        PUSH HL
        PUSH BC
        PUSH DE
        LD A,01H
        OUT (62H),A
        LD C,43H
        LD A,0
        OUT (C),A
PS2:    IN A,(C)
        BIT 0,A
        JP Z,PS2
        IN A,(42H)
        PUSH AF
        LD A,30H
        OUT (C),A
        POP AF
        LD HL,9010H
        CP (HL)
        CALL NZ,SC3TR
        CALL PCTRANS
        POP DE
        POP BC
        POP HL
        POP AF
        EI
        RET

;*****************************************

        ORG 1300H
        PUSH AF
        PUSH HL
        PUSH BC
        PUSH DE
        LD A,00H
        OUT (62H),A
        LD C,43H
        LD A,0
        OUT (C),A
PS3:    IN A,(C)
        BIT 0,A
        JP Z,PS3
```

```
        IN A,(42H)
        PUSH AF
        LD A,30H
        OUT (C),A
        POP AF
        LD HL,9019H
        CP (HL)
        CALL NZ,SC1TR
        CALL PCTRANS
        POP DE
        POP BC
        POP HL
        POP AF
        EI
        RET

;**********************************

TRANS:  CALL SC1TR
        CALL SC3TR
        CALL SC4TR
        RET

;**********************************

SC1TR:  LD A,0
        OUT (62H),A
        LD HL,9000H
        LD B,9
        CALL SEND
        RET

;****************************************

SC3TR:  LD A,01
        OUT (62H),A
        LD HL,9009H
        LD B,8
        CALL SEND
        RET

;********************************************

SC4TR:  LD A,2
        OUT (62H),A
        LD HL,9011H
        LD B,9
        CALL SEND
        RET

;************************************************

SEND:   LD A,0
        OUT (43H),A
NSEND:   IN A,(43H)
        BIT 2,A
```

```
        JP Z,NSEND
        LD A,(HL)
        OUT (42H),A
        INC HL
        DJNZ SEND
        RET

;****************************************

PCTRANS:LD A,3
        OUT (62H),A
        LD HL,9008H
        LD B,1
        CALL SEND
        RET
```

```
;************************************
; SLAVE CARD 1 PROGRAMI
;************************************

        ORG 00H
        JP 0200H

;************************************

        ORG 0038H
        DI
        PUSH AF
        PUSH HL
        PUSH BC
        PUSH DE
        LD C,43H
        LD DE,9A00H
        LD B,9
POLL1:  LD A,0
        OUT (C),A
POLL:   IN A,(C)
        BIT 0,A
        JP Z,POLL
        IN A,(42H)
        LD (DE),A
        LD A,30H
        OUT (C),A
        INC DE
        DJNZ POLL1
        POP DE
        POP BC
        POP HL
        POP AF
        EI
        RET

;****************************************

CODES:  DB 18H,14H,04H,13H,0C1H,15H,68H
DATA:   DB 0CDH,0D5H,99H,79H,0B9H,0DAH,0D9H,0D1H
DELAYN: DB 20,3,3,16,3,16,3,3,2
YELLOW: DB 0B7H,0FFH,0B7H,0FFH,0B7H,0FFH,0B7H,0FFH

;****************************************

        ORG 200H
        LD SP,9FFFH
        LD A,55H
        OUT (20H),A
        NOP
        NOP
        OUT (21H),A
        LD A,6FH
        OUT (20H),A
        LD A,0FH
        OUT (21H),A
```

```
        LD C,43H
        LD B,7
        LD HL,CODES
        OTIR
        LD A,3FH
        OUT (01H),A
        LD IY,DELAYN
        LD B,9
        LD HL,9A00H
LOOP:   LD A,(IY)
        LD (HL),A
        INC HL
        INC IY
        DJNZ LOOP
        IM 1
        EI
AGAIN:  LD IX,DATA
        LD IY,9A00H
        LD B,8
        LD A,(IY+8)
        CP 1
        JP Z,ONLYYELLOW
        CALL TRANSMIT
CONT:   LD A,(IX)
        OUT (00H),A
        CALL DELAYX
        INC IX
        INC IY
        DJNZ CONT
        JP AGAIN

;*******************************

DELAYX: LD C,(IY)
DELAY1: LD DE,0FFFFH
DELAY2: DEC DE
        LD A,D
        OR E
        JP NZ,DELAY2
        DEC C
        JP NZ,DELAY1
        RET

;*****************************************

TRANSMIT:DI
        PUSH BC
        LD HL,9A00H
        LD B,9
TRANS:  LD A,0
        OUT (43H),A
INT1:   IN A,(43H)
        BIT 2,A
        JP Z,INT1
        LD A,(HL)
        OUT (42H),A
```

```
                INC HL
                DJNZ TRANS
                POP BC
                EI
                RET

        ;************************************

ONLYYELLOW:     LD IX,YELLOW
                LD A,(IX)
                OUT (00H),A
                CALL TRANSMIT
                CALL DELAY2SN
                INC IX
                LD A,(IX)
                OUT (00H),A
                CALL DELAY2SN
                LD A,(9A08H)
                CP 1
                JP Z,ONLYYELLOW
                JP AGAIN

        ;********************************

      DELAY2SN: LD C,5
        DEL1:   LD DE,0FFFFH
        DEL2:   DEC DE
                LD A,D
                OR E
                JP NZ,DEL2
                DEC C
                JP NZ,DEL1
                RET

        ;************************************
```

```
;************************************
; SLAVE CARD 2 PROGRAM1
;************************************

        ORG 00H
        JP 0200H

;**************************************

        ORG 0038H
        DI
        PUSH AF
        PUSH HL
        PUSH BC
        PUSH DE
        LD C,43H
        LD DE,9A00H
        LD B,9
POLL1:  LD A,0
        OUT (C),A
POLL:   IN A,(C)
        BIT 0,A
        JP Z,POLL
        IN A,(42H)
        LD (DE),A
        LD A,30H
        OUT (C),A
        INC DE
        DJNZ POLL1
        CALL DELAY5SN
        POP DE
        POP BC
        POP HL
        POP AF
        EI
        RET

;****************************************

CODES:  DB 18H,14H,04H,13H,0C1H,15H,68H
DATA:   DB 0CDH,0D5H,99H,79H,0B9H,0DAH,0D9H,0D1H
YELLOW: DB 0B7H,0FFH

;****************************************

DELAY5SN:LD C,10
DEL51:  LD DE,0FFFFH
DEL52:  DEC DE
        LD A,E
        OR D
        JP NZ,DEL52
        DEC C
        JP NZ,DEL51
        RET
```

```
;*****************************************

        ORG 200H
        LD SP,9FFFH
        LD A,55H
        OUT (20H),A
        NOP
        NOP
        OUT (21H),A
        LD A,6FH
        OUT (20H),A
        LD A,0FH
        OUT (21H),A
        LD C,43H
        LD B,7
        LD HL,CODES
        OTIR
        LD A,3FH
        OUT (01H),A
        NOP
        NOP
        OUT (03H),A
        IM 1
        EI
        LD A,1
        LD (9A08H),A
YELLOWY:LD IX,YELLOW
        CALL TRANSMIT
        LD A,(IX)
        OUT (00H),A
        INC IX
        CALL DELAY3
        LD A,(IX)
        OUT (00H),A
        CALL DELAY3
        LD A,(9A08H)
        CP 1
        JP Z,YELLOWY

AGAIN:  LD IX,DATA
        LD IY,9A00H
        LD B,8
        LD A,(IY+8)
        CP 1
        JP Z,YELLOWY
        CALL TRANSMIT
CONT:   LD A,(IX)
        OUT (00H),A
        CALL DELAYX
        INC IX
        INC IY
        DJNZ CONT
        JP AGAIN
```

```
;*********************************
DELAYX: LD C,(IY)
DELAY1: LD DE,0FFFFH
DELAY2: DEC DE
        LD A,D
        OR E
        JP NZ,DELAY2
        DEC C
        JP NZ,DELAY1
        RET

;*****************************************
DELAY3: LD C,5
DELA31: LD DE,0FFFFH
DELA32: DEC DE
        LD A,D
        OR E
        JP NZ,DELA32
        DEC C
        JP NZ,DELA31
        RET

;************************************

TRANSMIT:DI
        LD HL,9A08H
        LD A,0
        OUT (43H),A
 INT1:   IN A,(43H)
        BIT 2,A
        JP Z,INT1
        LD A,(HL)
        OUT (42H),A
        EI
        RET

;************************************
```

```
;**********************************
; SLAVE CARD 3 PROGRAM1
;**********************************

        ORG 00H
        JP 0100H

;**************************************

        ORG 0038H
        DI
        JP 570H

;**************************************

        ORG 0066H
        JP 500H
;**************************************

        ORG 0100H
        LD SP,9FFFH
        LD A,55H
        OUT (20H),A
        NOP
        NOP
        OUT (21H),A
        LD A,6FH
        OUT (20H),A
        LD A,0FH
        OUT (21H),A
        LD C,43H
        LD B,7
        LD HL,CODES
        OTIR
        LD A,3FH
        OUT (01H),A
        LD A,3FH
        OUT (03H),A
        LD IY,DELAYN
        LD B,8
        LD HL,9A00H
LOOP:   LD A,(IY)
        LD (HL),A
        INC HL
        INC IY
        DJNZ LOOP
        IM 1
        EI
AGAIN:  LD IX,DATA1
        LD HL,DATA2
        LD IY,9A00H
        LD B,7
        LD A,(IY+7)
        CP 1
        JP Z,YELLOWY
        CALL TRANSMIT
```

```
CONT:    LD A,(IX)
         OUT (00H),A
         LD A,(HL)
         OUT (02H),A
         CALL DELAYX
         INC IX
         INC HL
         INC IY
         DJNZ CONT
         JP AGAIN

;*********************************

DELAYX: LD C,(IY)
DELAY1: LD DE,0FFFFH
DELAY2: DEC DE
         LD A,D
         OR E
         JP NZ,DELAY2
         DEC C
         JP NZ,DELAY1
         RET

;*****************************************

TRANSMIT:DI
         PUSH HL
         LD HL,9A07H
         LD A,0
         OUT (43H),A
INT1:    IN A,(43H)
         BIT 2,A
         JP Z,INT1
         LD A,(HL)
         OUT (42H),A
         POP HL
         EI
         RET

;***********************************

YELLOWY:LD IX,YELLOW1
         LD IY,YELLOW2
         LD A,(IX)
         OUT (00H),A
         LD A,(IY)
         OUT (02H),A
         CALL TRANSMIT
         CALL DELY2
         INC IX
         INC IY
         LD A,(IX)
         OUT (00H),A
         LD A,(IY)
         OUT (02H),A
         CALL DELY2
```

```
        LD A,(9A07H)
        CP 1
        JP Z,YELLOWY
        JP AGAIN

;*******************************

DELY2:  LD C,5H
DEL1:   LD DE,0FFFFH
DEL2:   DEC DE
        LD A,D
        OR E
        JP NZ,DEL2
        DEC C
        JP NZ,DEL1
        RET

;************************************
        ORG 500H
        PUSH AF
        PUSH HL
        PUSH BC
        PUSH DE
        PUSH IY
        PUSH IX
        LD C,5
        CALL DEL1
        LD A,B
        CP 3
        JP Z,YESIL
        LD A,0DDH
        OUT (00H),A
        CALL DELY2
        LD A,0EEH
        OUT (00H),A
        LD A,2
        OUT (02H),A
        LD C,16
        CALL DEL1
        LD A,5
        OUT (02H),A
        CALL DELY2
YESIL:  POP IX
        POP IY
        POP DE
        POP BC
        POP HL
        POP AF
        LD A,(IX)
        OUT (00H),A
        RETN
```

```
        ORG 570H
        PUSH AF
        PUSH HL
        PUSH BC
        PUSH DE
        LD C,43H
        LD DE,9A00H
        LD B,8
POLL1:  LD A,0
        OUT (C),A
POLL:   IN A,(C)
        BIT 0,A
        JP Z,POLL
        IN A,(42H)
        LD (DE),A
        LD A,30H
        OUT (C),A
        INC DE
        DJNZ POLL1
        POP DE
        POP BC
        POP HL
        POP AF
        EI
        RET

;*****************************************

CODES:  DB 18H,14H,04H,13H,0C1H,15H,68H
DATA1:  DB 6BH,4DH,0B6H,0D6H,0EEH,0EEH,0ECH
DELAYN: DB 30,3,30,3,26,3,3,2
YELLOW1:DB 0DDH,0FFH
YELLOW2:DB 0FFH,0FFH
DATA2:  DB 05H,05H,05H,05H,02H,05H,05H
;*****************************************
```

```
;************************************
; SLAVE CARD 4 PROGRAMI
;************************************

        ORG 00H
        JP 0200H

;************************************

        ORG 0038H
        DI
        PUSH AF
        PUSH HL
        PUSH BC
        PUSH DE
        LD C,43H
        LD DE,9A00H
        LD B,9
POLL1:  LD A,0
        OUT (C),A
POLL:   IN A,(C)
        BIT 0,A
        JP Z,POLL
        IN A,(42H)
        LD (DE),A
        LD A,30H
        OUT (C),A
        INC DE
        DJNZ POLL1
        POP DE
        POP BC
        POP HL
        POP AF
        EI
        RET

;****************************************

CODES:  DB 18H,14H,04H,13H,0C1H,15H,68H
DATA:   DB 0CDH,0D5H,99H,79H,0B9H,0DAH,0D9H,0D1H
DELAYN: DB 20,3,3,16,3,16,3,3,2
YELLOW: DB 0B7H,0FFH,0B7H,0FFH,0B7H,0FFH,0B7H,0FFH

;****************************************

        ORG 200H
        LD SP,9FFFH
        LD A,55H
        OUT (20H),A
        NOP
        NOP
        OUT (21H),A
        LD A,6FH
        OUT (20H),A
        LD A,0FH
        OUT (21H),A
```

```
        LD C,43H
        LD B,7
        LD HL,CODES
        OTIR
        LD A,3FH
        OUT (01H),A
        LD IY,DELAYN
        LD B,9
        LD HL,9A00H
LOOP:   LD A,(IY)
        LD (HL),A
        INC HL
        INC IY
        DJNZ LOOP
        IM 1
        EI
AGAIN:  LD IX,DATA
        LD IY,9A00H
        LD B,6
        LD A,(IY+8)
        CP 1
        JP Z,ONLYYELLOW
        CALL TRANSMIT
CONT:   LD A,(IX)
        OUT (00H),A
        CALL DELAYX
        INC IX
        INC IY
        DJNZ CONT
        JP AGAIN

;*********************************

DELAYX: LD C,(IY)
DELAY1: LD DE,0FFFFH
DELAY2: DEC DE
        LD A,D
        OR E
        JP NZ,DELAY2
        DEC C
        JP NZ,DELAY1
        RET

;*****************************************

TRANSMIT:DI
        LD HL,9A08H
        LD A,0
        OUT (43H),A
INT1:   IN A,(43H)
        BIT 2,A
        JP Z,INT1
        LD A,(HL)
        OUT (42H),A
        EI
        RET
```

```
        ;************************************

ONLYYELLOW:     LD IX,YELLOW
                LD A,(IX)
                OUT (00H),A
                CALL TRANSMIT
                CALL DELAY2SN
                INC IX
                LD A,(IX)
                OUT (00H),A
                CALL DELAY2SN
                LD A,(9A08H)
                CP 1
                JP Z,ONLYYELLOW
                JP AGAIN

        ;********************************

       DELAY2SN: LD C,5
         DEL1:   LD DE,0FFFFH
         DEL2:   DEC DE
                 LD A,D
                 OR E
                 JP NZ,DEL2
                 DEC C
                 JP NZ,DEL1
                 RET

        ;************************************
```