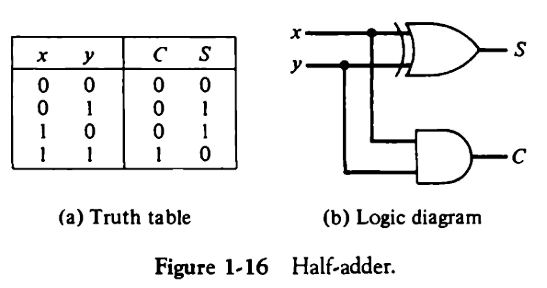
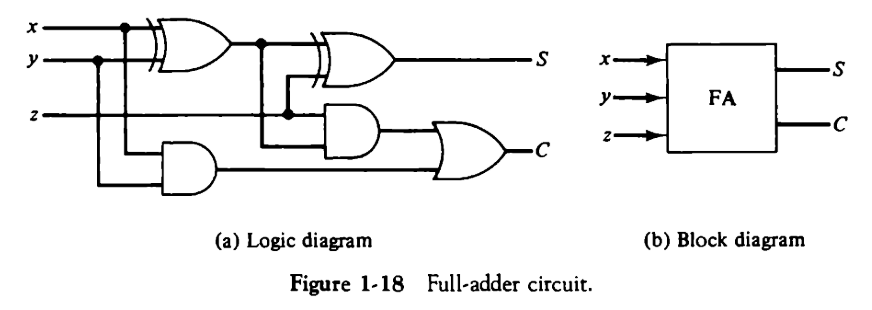
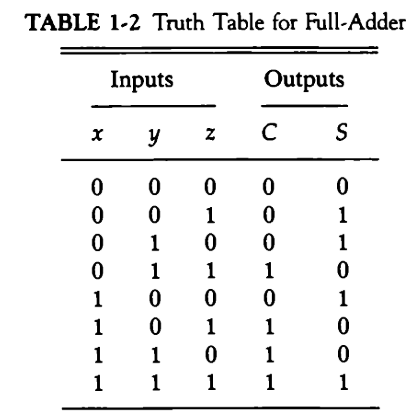
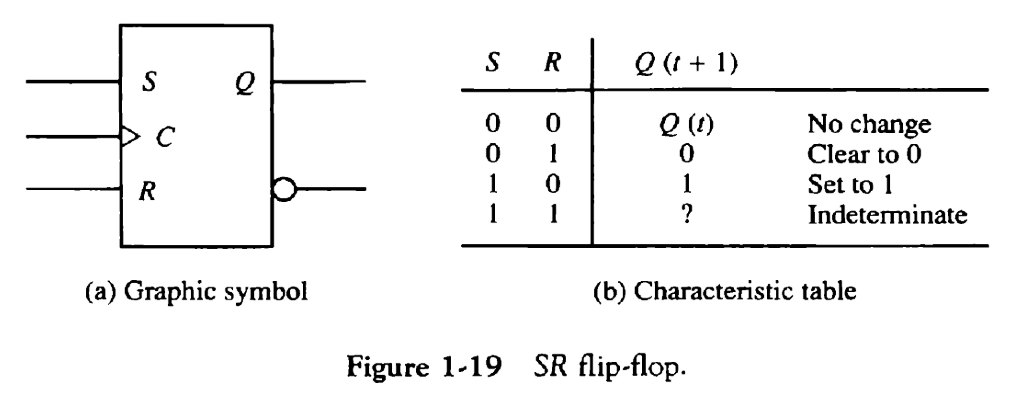
**CHAPTER 1**

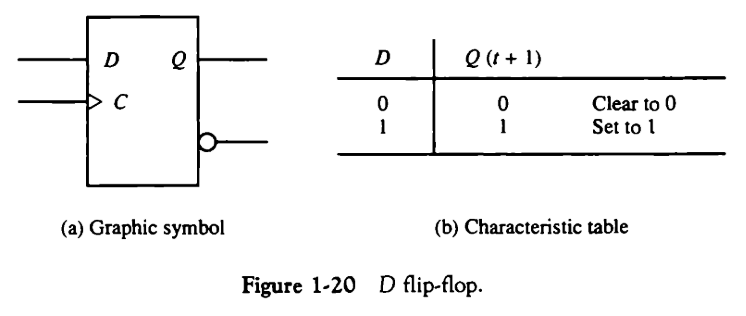
**Adders**

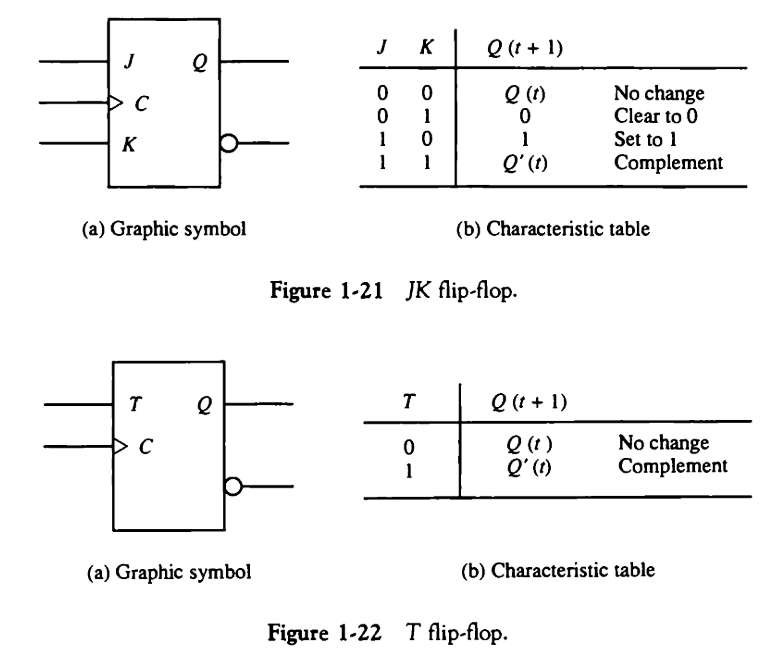


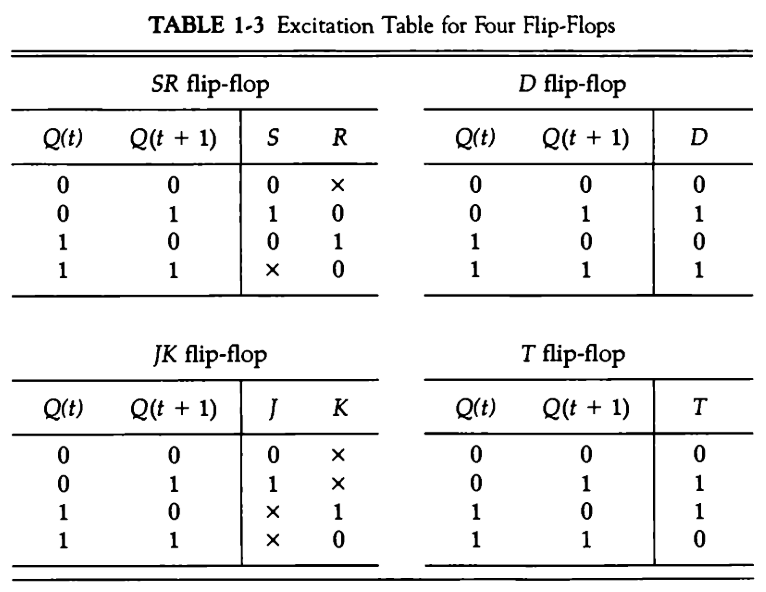
 

**Flip-flops**

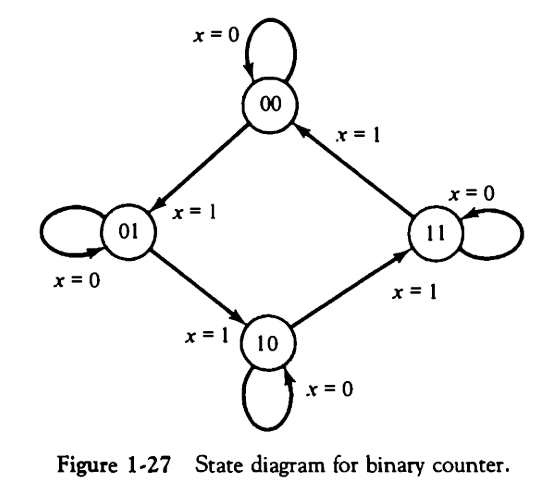
****

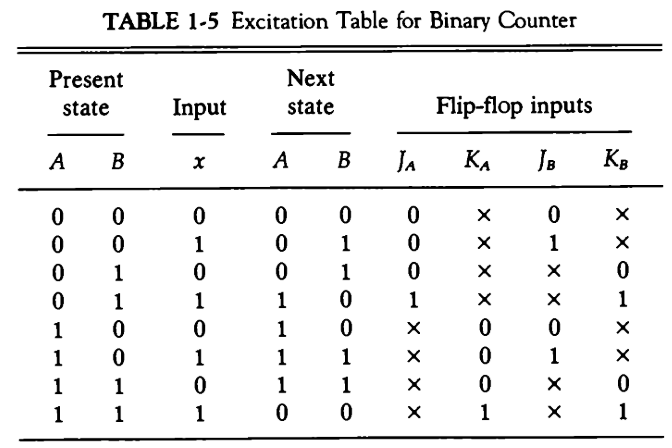
****

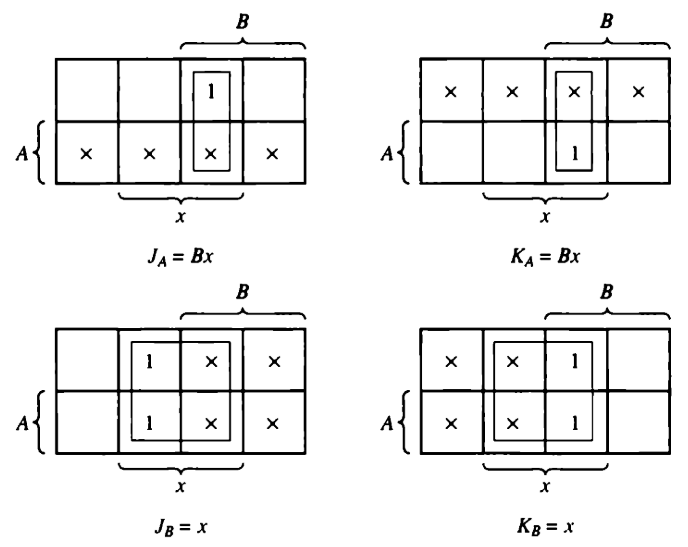
****

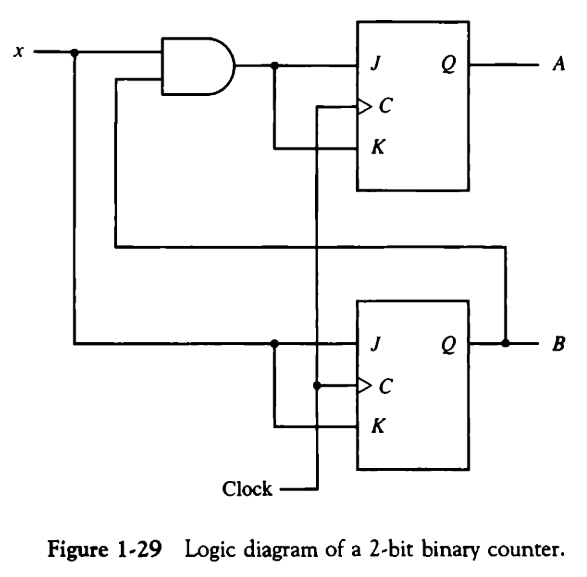
****

**Binary Counter Design Example**



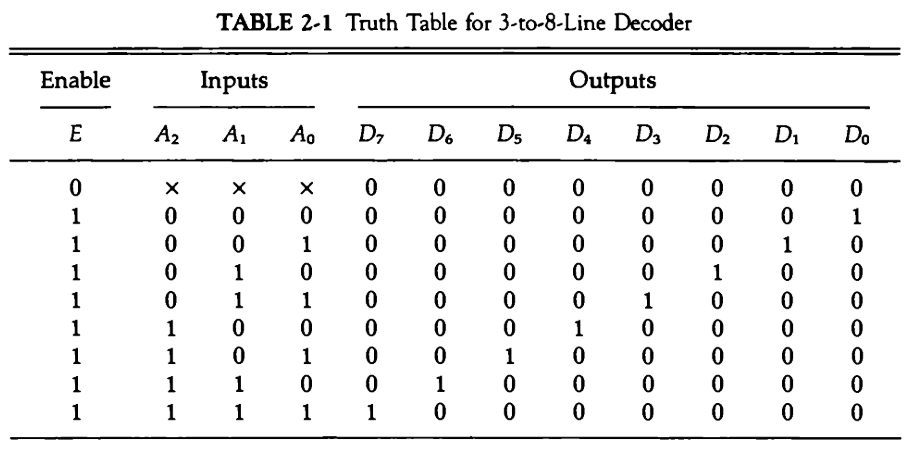
****

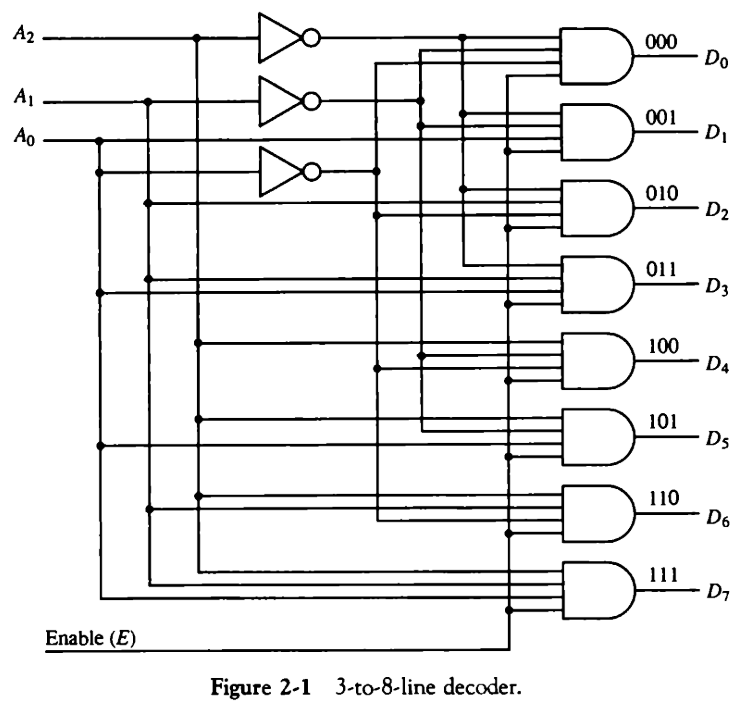
****

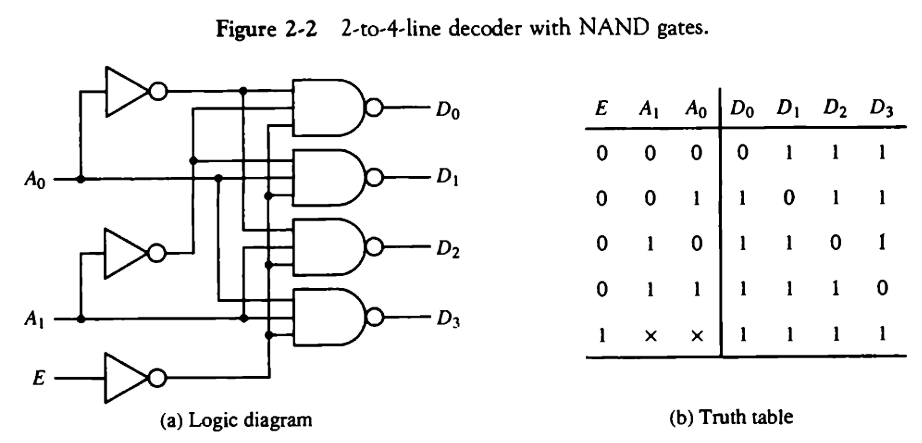
****

**CHAPTER 2**

**Decoder**

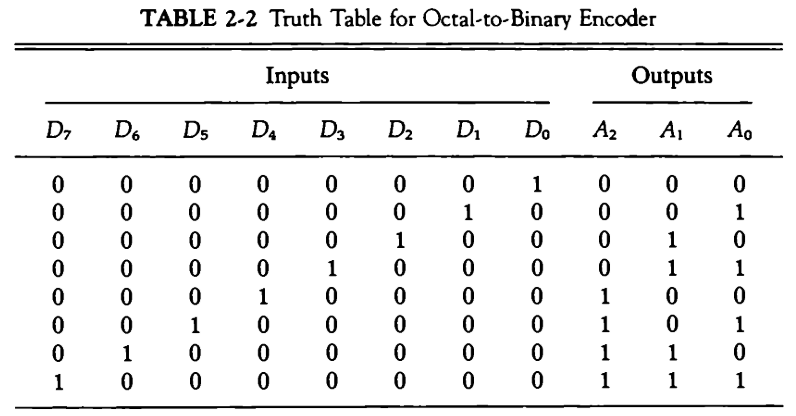
****

****

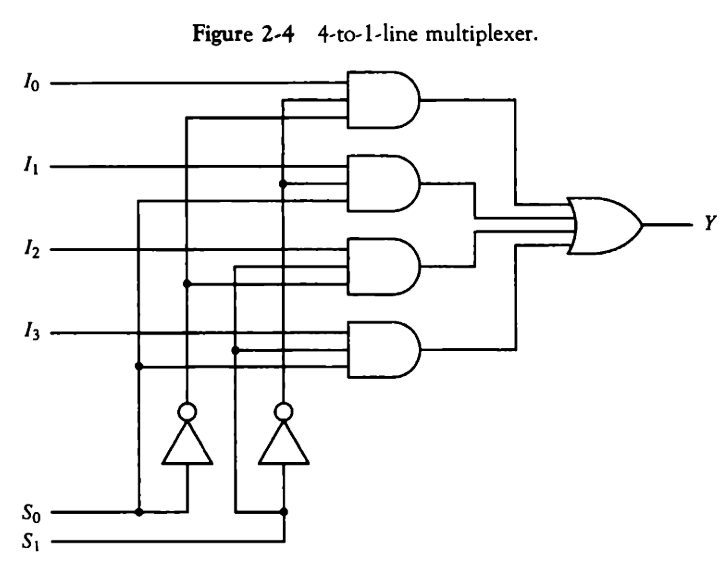
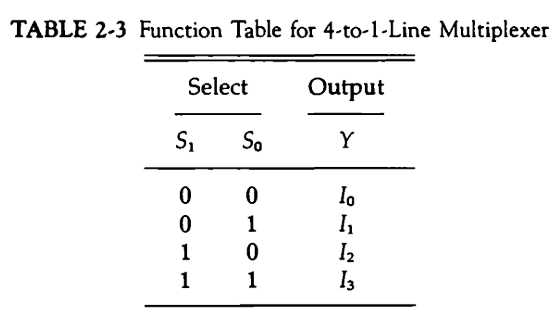
****

****

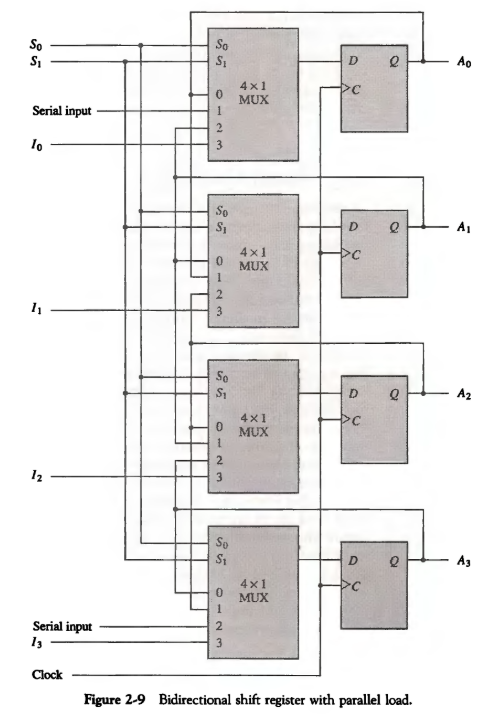
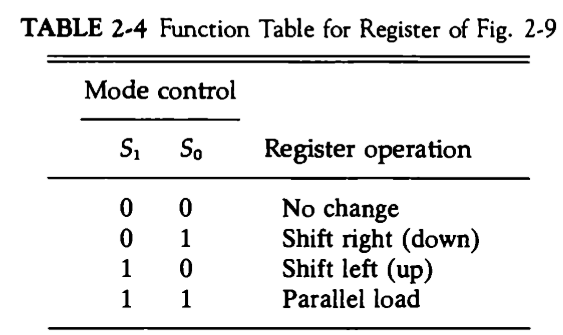
**Encoder**

****

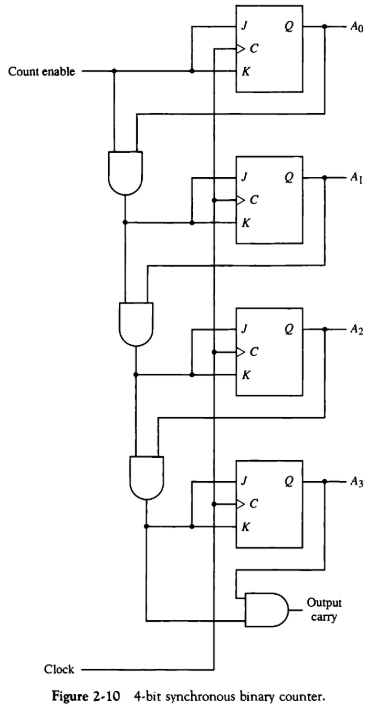
**Multiplexer**

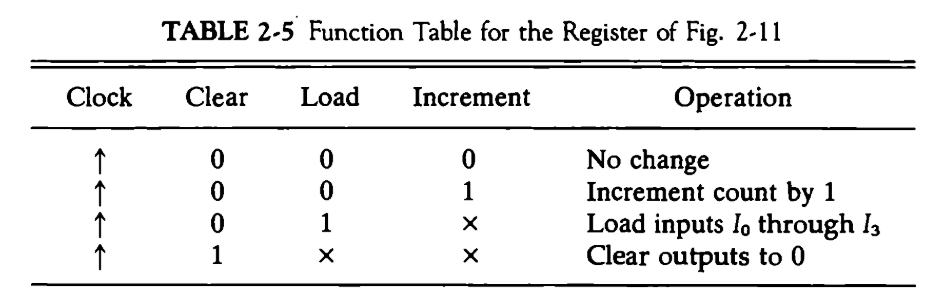
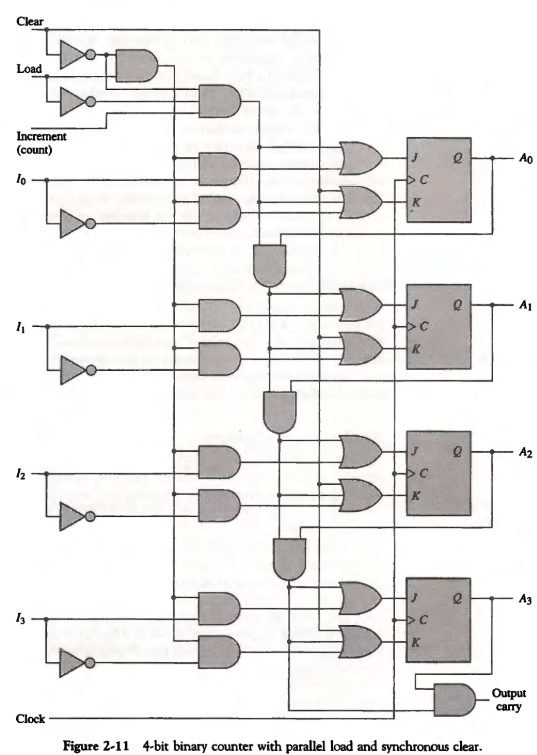
**** ****

**Register**

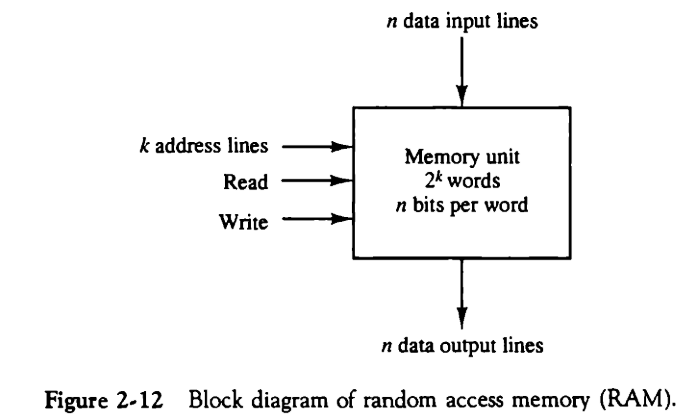
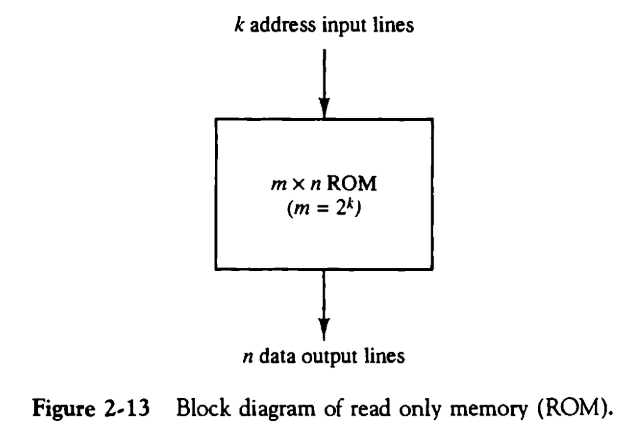
**** ****

**Counter**

****

****

**Memory Unit**

**** ****

* Random Access Memory (RAM)’de, memorydeki herhangi bir worde erişmek, konumundan bağımsız olarak eşit zaman alıyor.
* Read Only Memory (ROM), RAM’den farklı olarak read kontrol bitine sahip değil, çünkü içine yazma yapılmadığı için çıktısı herhangi bir zamanda erişilebilir ve sadece mevcut duruma bağlı. ROM sadece combinational circuit’lerden oluşuyor.
* Mask Programlama:

ROM'un yolları yarı iletken şirketi tarafından son fabrikasyon sürecinde programlanır.

ROM'un üretilmesi için müşterinin belirlediği doğruluk tablosunu doldurması gerekir.

Mask programlama, müşterinin doğruluk tablosuna göre 1'leri ve 0'ları üretmek için uygun bir maske oluşturulmasını içerir.

Bu yöntem maliyetlidir ve büyük miktarlarda aynı ROM yapılandırması sipariş edildiğinde ekonomiktir.

* PROM (Programmable Read-Only Memory):

PROM, tüm sigortaların sağlam olduğu şekilde sipariş edilir, yani depolanan kelimelerin tüm bitlerinde 1'ler bulunur.

PROM'daki sigortalar, her bir adres için çıkış terminalleri üzerinden uygulanan akım darbeleriyle patlatılır.

Patlamış bir sigorta bir ikili 0 durumunu tanımlar, sağlam bir sigorta ise ikili 1 durumunu verir.

Bu sayede kullanıcılar laboratuvarlarında PROM'ları istedikleri giriş adresleri ve depolanan kelimeler arasındaki ilişkiyi sağlamak için programlayabilirler.

* EPROM (Erasable Programmable Read-Only Memory):

EPROM, önceden patlatılmış sigortalara sahip olmasına rağmen başlangıç ​​değerine yeniden yapılandırılabilir.

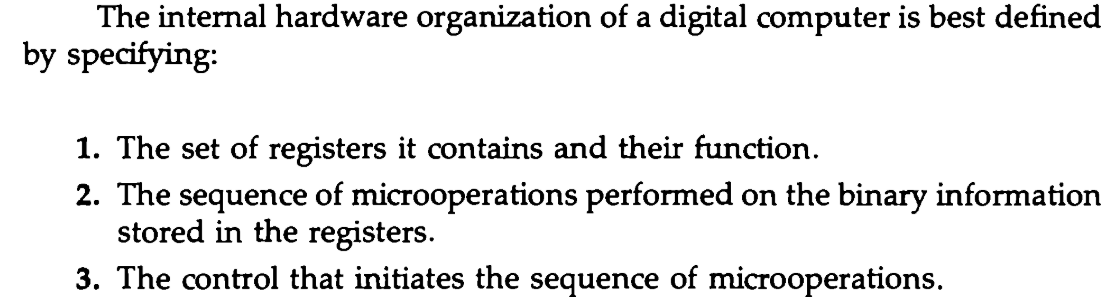
EPROM, özel bir ultraviyole ışık altında belirli bir süre tutulduğunda iç kapıları patlatan kısa dalga radyasyon ile silinebilir.

Silindikten sonra EPROM, başlangıç durumuna geri döner ve yeni bir kelime kümesine yeniden programlanabilir.

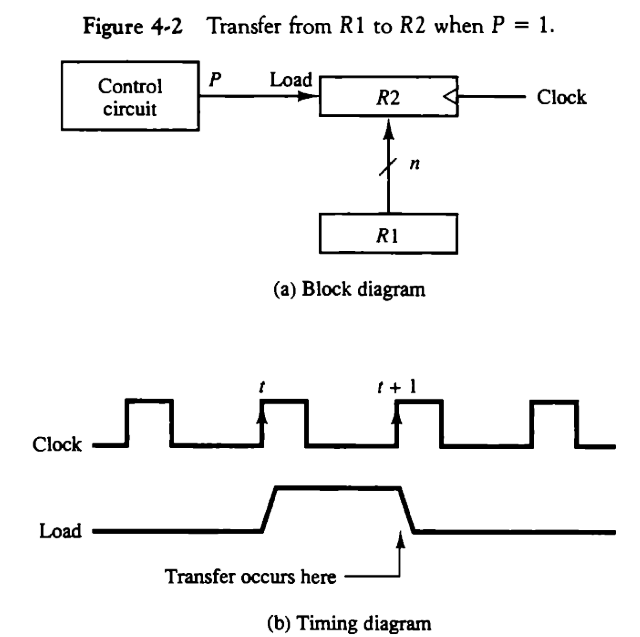
Bazı PROM'lar ultraviyole ışık yerine elektrik sinyalleri ile silinebilir. Bu PROM'lara EEPROM (Electrically Erasable PROM) denir.

**CHAPTER 4**

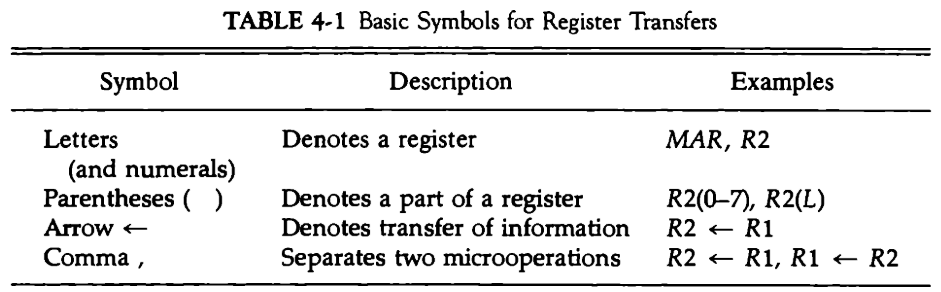
* Register'larda saklanan veriler üzerinde gerçekleştirilen işlemlere microoperation denir. Bir microoperation, bir veya daha fazla register'da saklanan bilgi üzerinde gerçekleştirilen temel bir işlemdir. Bir clock pulse’da tamamlanır.



* P: R2 <- R1’in block diyagramı:

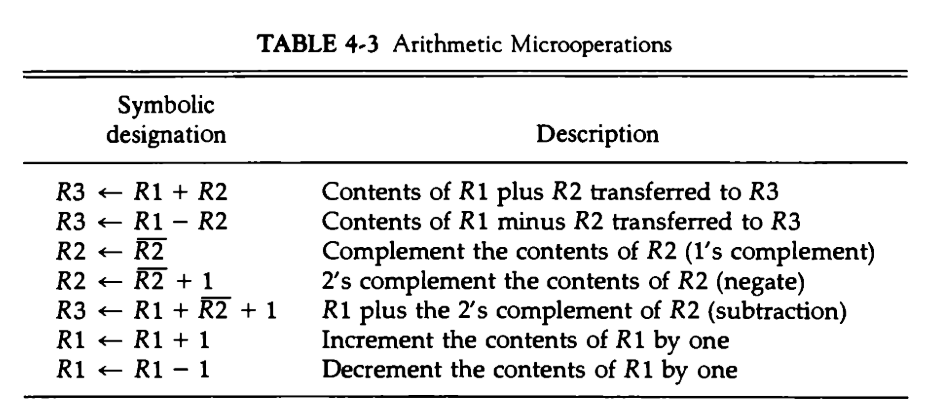


* P, t’de aktif hale gelir. t + 1’e gelindiğinde clock pozitif transitiona geçerken P’yi aktif bulur ve load işlemini gerçekleştirir.
* P, t’den hemen sonra aktif olsa bile transfer clock’un bir sonraki pozitif transitionuna kadar gerçekleşmez.
* T: R2 <- R1, R1 <- R2 denildiğinde, virgül bu iki mikrooperasyonun aynı anda gerçekleştiğini (aynı clock darbesi içerisinde) belirtir.

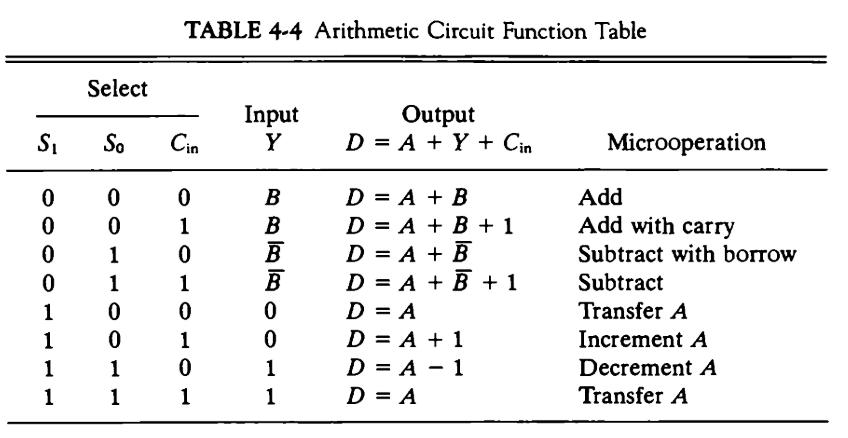
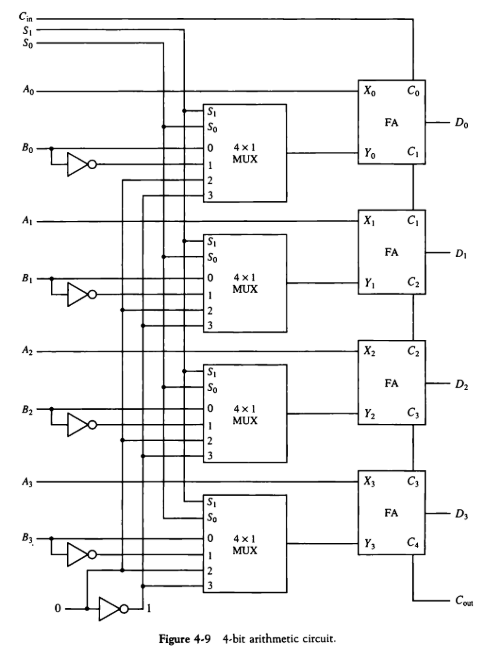


* R2’ işlemi 1’s complement of R2, R2’ + 1 işlemi 2’s complement of R2’dir.

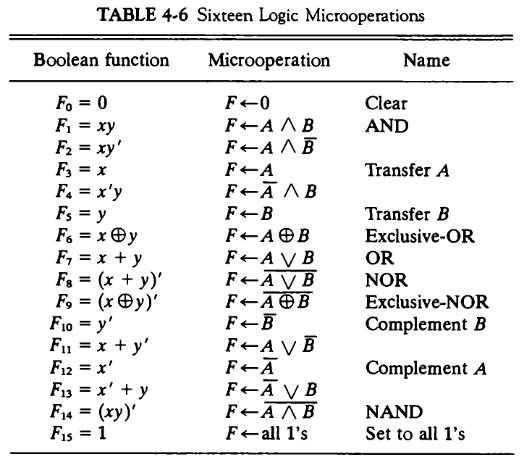
**Arithmetic Microoperations**

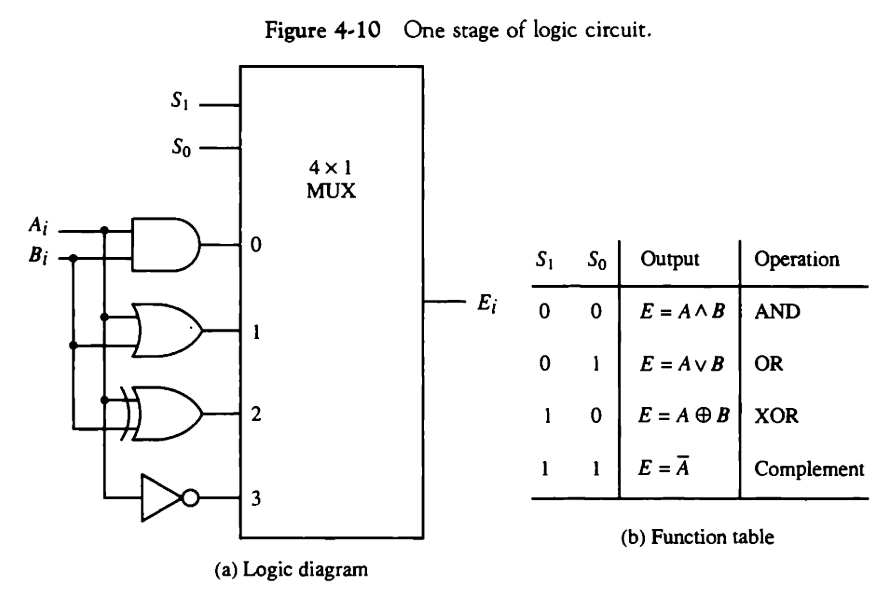


* Çarpma ve bölme microoperation değil.



**Logic Microoperations**

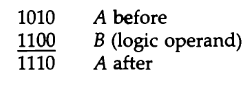
****

****

**Logic Microoperation Applications**

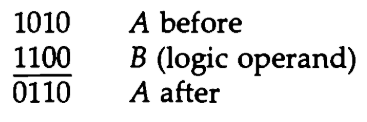
1. **Selective Set**

B logic operandındaki 1’leri, A registerine yerleştirir. Kısaca A <- A OR B



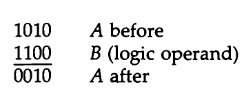
1. **Selective Complement**

B logic operandındaki 1 olan bitlerin indexlerinin A’daki karşılıkları complement edilir. Kısaca A <- A XOR B



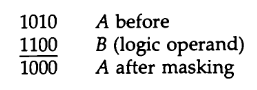
1. **Selective Clear**

B logic operandındaki 1 olan bitlerin indexlerinin A’daki karşılıkları 0’lanır. Kısaca A <- A AND B’



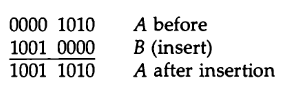
1. **Mask**

B logic operandındki 0 olan bitlerin indexlerinin A’daki karşılıkları 0’lanır. Kısaca A <- A AND B



1. **Insert**

A’ya bit eklenir. Önce eklenecek bit grubu mask edilir, sonra OR’lanarak eklenir.

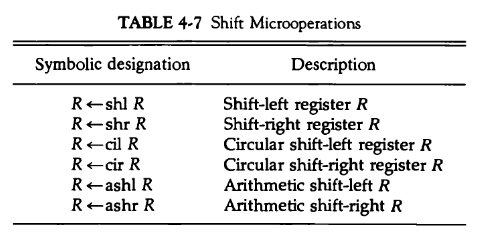


1. **Clear**

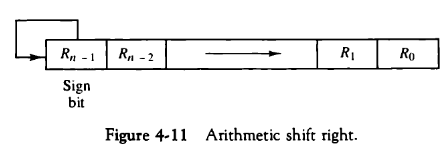
A ve B karşılaştırılır, aynılarsa 0 elde edilir. Kısaca A <- A XOR B



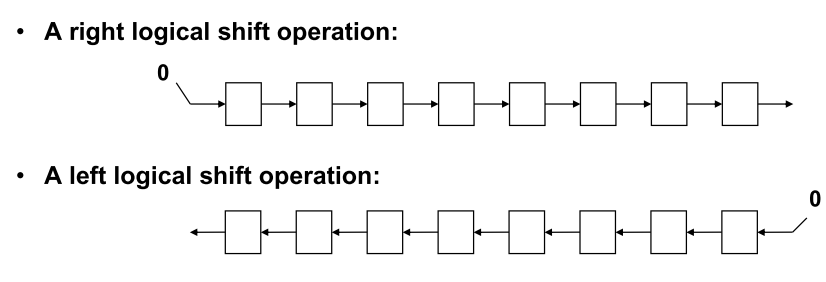
**Shift Microoperations**

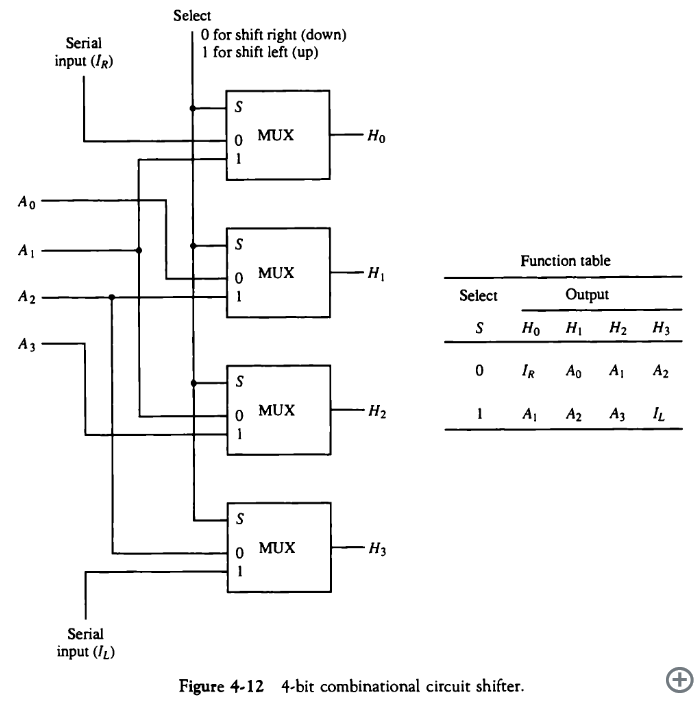


* Arithmetic shift operasyonu signed sayıyı kaydırmak için kullanılır. Kaydırdıkça 2’ye bölünür ya da çarpılır. Sign bitinin aynı kalması gerekir.

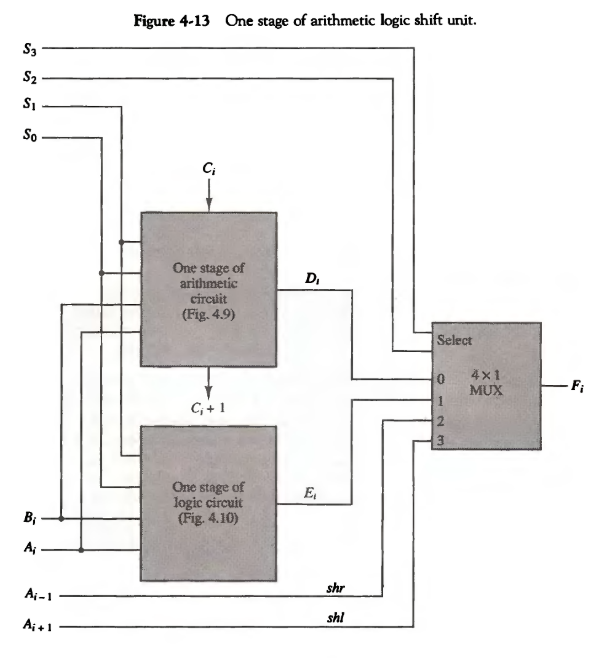


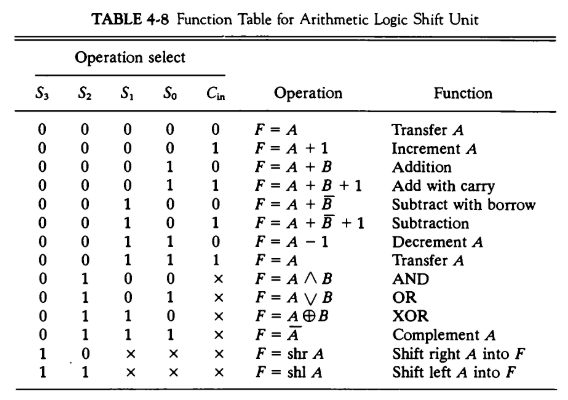
* Arithmetic shift left operasyonunda R0’a 0 set edilir. Diğer bitler sola kaydırılır. Sign biti değişirse overflow olmuş olur. Vs = Rn-1 XOR Rn-2 ile overflow kontrol edilir, Vs = 1 ise overflow var. Vs, overflow flip-flopuna, shift operasyonu ile aynı clock darbesinde aktarılır.





**Arithmetic Logic Shift Unit**





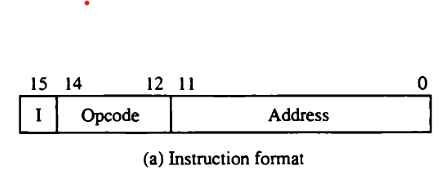
**CHAPTER 5**

**Instruction Codes**

* Instructiondan elde edilen operand ve Accumulator (AC) registerindeki değer kullanılarak, opcode un temsil ettiği operasyon gerçekleştirilir.

1. Eğer instruction, operandın adresi yerine direkt operandı içeriyorsa instructiona immediate operanda sahip denir.
2. Eğer operandın adresini taşıyorsa direct addresse sahip denir.
3. Eğer operandın bulunduğu adresin bulunduğu adresi içeriyorsa indirect addressing.

* Direct yada indirect adres olması, opcode’un başındaki 1 bit ile belirlenir. 0 direct, 1 indirect.
* **Operandın bulunduğu adres,** indirectte de directte de effective addres’tir!



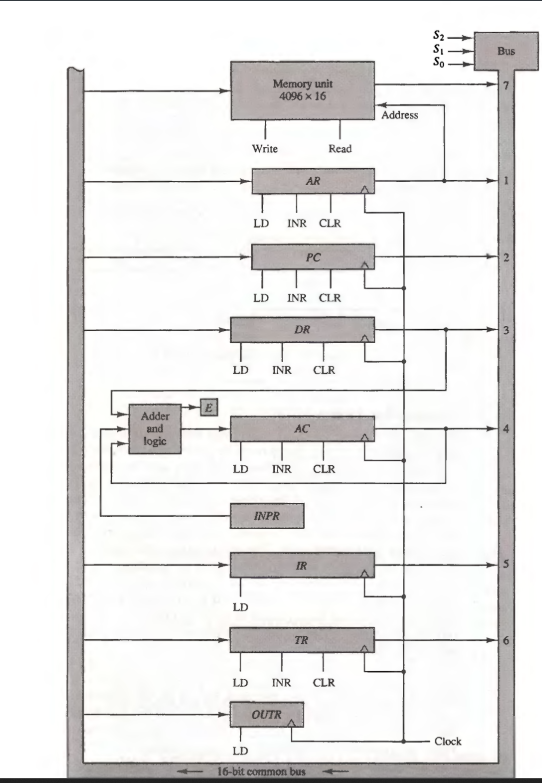
**Register**

* Memory 4096 worde sahip. Her word 16 bitten oluşuyor.



* Data Register (DR): memoryden okunan operandı tutar.
* Program Counter (PC): mevcut instruction execute edildikten sonra memoryden okunacak bir sonraki addresi tutar.
* Branch instructionu geldiğinde instructiondaki adres PC’ye yazılır, böylece program akışı verilen adrese atlamış olur.
* Memoryden instruction okumak için PC’nin değeri address olarak kullanılır. Sonrasında PC = PC + 1 olur ve bir sonraki instructionu tutmaya devam eder.

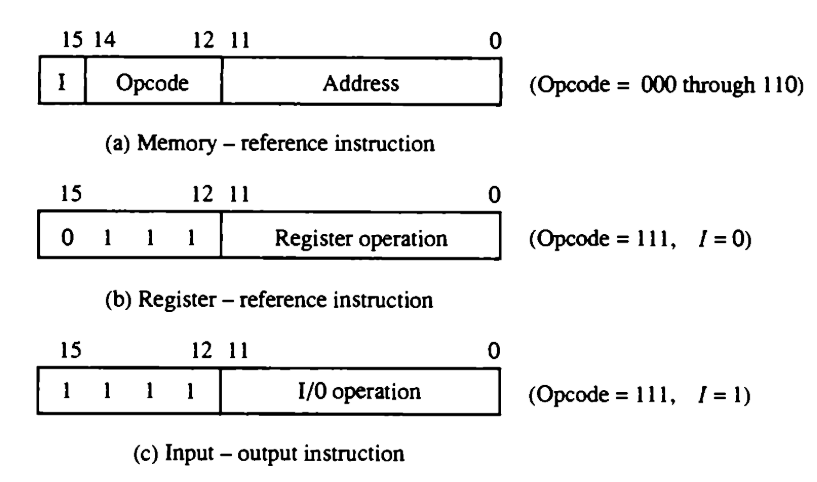
BUS YAPISI:



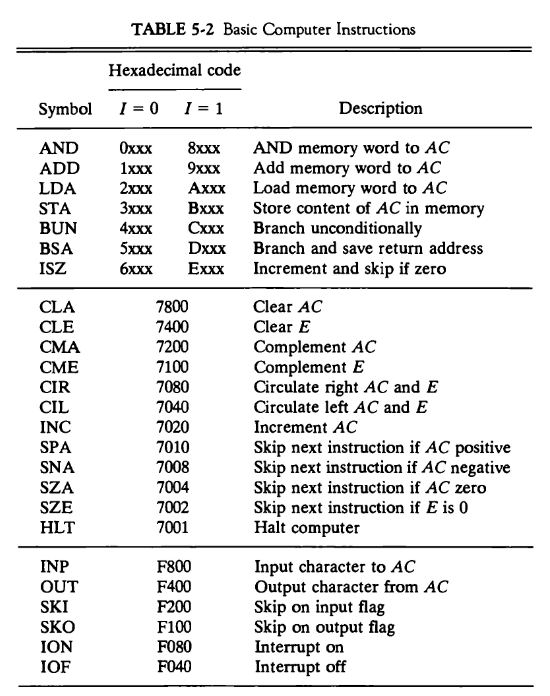
* s2s1s0 busa hangi registerin/memory adresinin değer göndereceğini seçer. Her donanımın kendi load (LD) inputu var. Bu inputlar enable olduğunda bustaki veriler kendilerine yüklenebilir.
* 12 bitlik PC ve AR gibi register’lar MSB bitlerine 0 yazılarak 16 bite çıkarılıp öyle yüklenir.
* Toplama işleminin sonucu AC'ye, elde edilen carry E flip-flopuna yüklenir (E, extended AC bit olarak geçiyor)
* DR ile AC içindeki değerler opcode a göre operasyonu gerçekleştirdikten sonra sonuç tekrar AC’de depolanır. AC <- DR + AC mesela.
* Herhangi bir registerin bus’a load edilmesi ile adder and logic circuit’te bir operasyonun gerçekleşmesi, aynı clock cycle içerisinde gerçekleşebilir!

**Instructions**

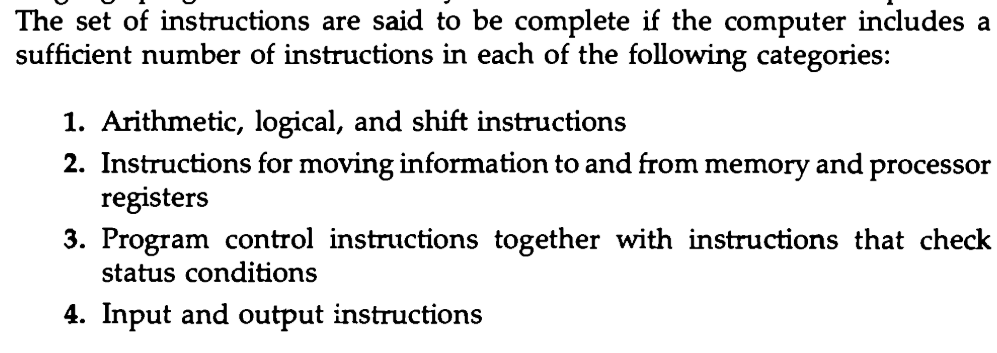
* 3 tip instruction var:



* Eğer IR(14-12) bitler 111 ise 15. bite bakılır. 15. bit 0 ise register-reference instruction, 1 ise input-output instruction olur. 111 değilse direkt memory reference instruction.
* Opcode için kullanılan 14-12 bitlerden dolayı operasyon sayısı 8 ile sınırlandırılmış gibi görünebilir, fakat register-reference ve I/O instructionlarında IR(11-0) operand için kullanılmadığından dolayı bu bitler de operasyonlar için kullanılır ve toplam operasyon sayısı 8’i geçebilir.

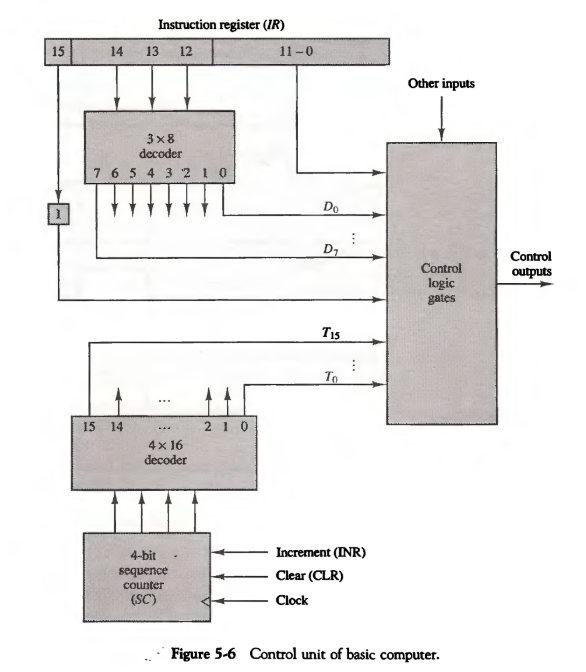


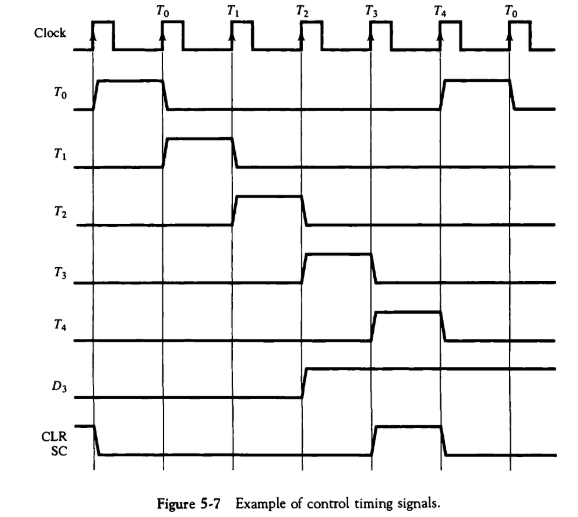
* Yukarıdakiler sadece minimum set. CMA ve AND kullanarak NAND yapabiliyosun, bunu yapınca zaten direkt tüm logic ve aritmetik işlemleri yapabilir hale geliyorsun, fakat sık kullanılan OR, SUBTRACT gibi işlemler ayrıca programlanabilir.
* Instructionların complete olması:



**Control and Timing**

* Control unit’deki registerlar da clocka bağlı. Sistemdeki tüm flip-flop ve registerlara clock bağlı.
* Control ünitesi; common bus multiplexerı, processor registerlarının kontrol inputları ve AC registerinin mikrooperasyonları için kontrol inputları üretir.
* 2 türlü kontrol var: hardwired and microprogrammed. Hardwired’da devrelerdeki component değiştikçe controldeki devreler de değişmeli, microprogrammed olunca sadece programı güncellemek yeterli.
* Control unit; 2 decoder, 4-bitlik sequence counter (15’e kadar sayabiliyor) ve birkaç tane kontrol logic gate içeriyor.

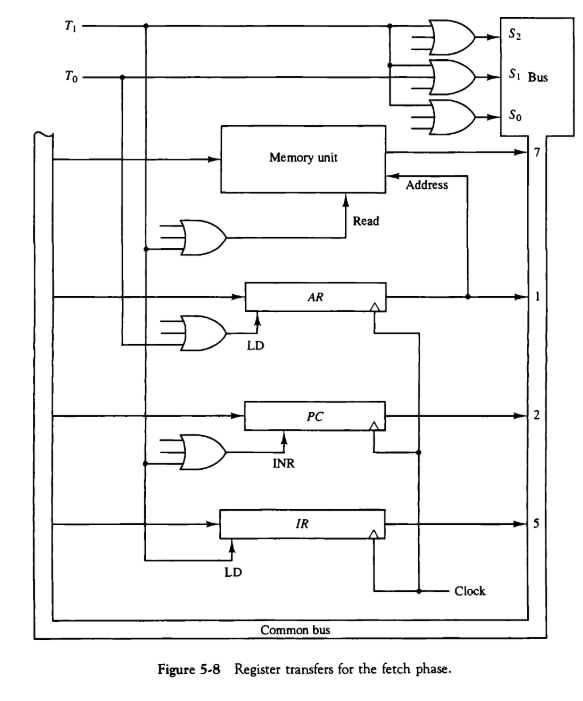




*  denildiğinde T0 anında bus’ın selectleri PC’nin outputunun busa alındığı duruma getirilir, AR’nin load enable’ı da aktif edilir. Asıl veri aktarımı T0’da değil, T0’ın label edildiği (yukarıdaki timing diagramda görülebilir) clock cycle geldiğinde sağlanır, aynı clock darbesi sequence counteri de 1 arttırır.
* Ti’nin label edildiği clock darbesi geldiğinde darbe, sadece Ti’nin kontrol input olarak kullanıldığı registerlara etki eder.
* Basic computer’da, timing signal ile başlatılan memory read-write işlemlerinin, bir sonraki rising clock gelene kadar bittiği varsayılır gerçekte olmasa da. Dolayısıyla memory cycle süresinin clock cycle süresinden kısa olduğu varsayılır. Memory cycle önce bitmeli ki memorydeki okunan memory word, registere load edilebilsin gerektiğinde.
* Memory’nin clock’a bağlı olmadığı fark edilmeli!

**Instruction Cycle**

1. Instructionu memoryden fetch et.
2. Instructionu decode et.
3. Indirect adresse effective adresini oku.
4. Insturctionu çalıştır.

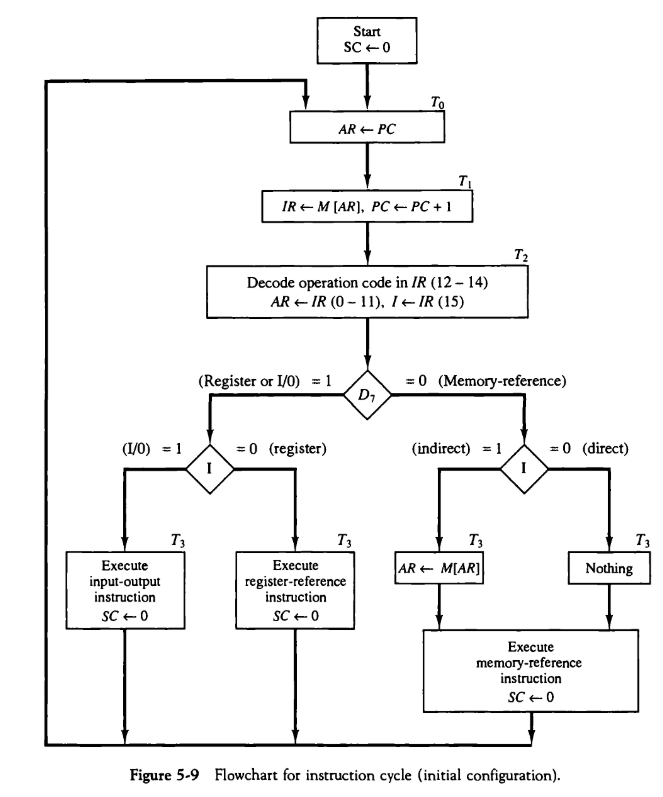


**Fetch and Decode**

1. T0: AR <- PC çünkü memory’nin address inputu sadece AR’ye bağlı.
2. T1: IR <- M[AR], PC <- PC + 1
3. T2: D0,...,D7 <- Decode IR(12-14), AR <- IR(0-11), I <- IR(15) (I indirect addressingi gösteren flip-flop).

**Determining Instruction Type**

* T3 boyunca instructionun tipi belirlenir.

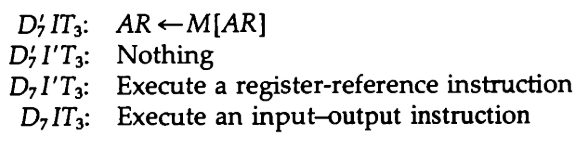


* Eğer D7’T3 ise yani IR(12:14) 111 değilse, yani memory reference instruction ise:

1. T3 zamanında indirect ise gider effective adresi bulur.
2. Değilse hiçbir şey yapmaz.

SC arttırılıp T4’le devam ederek instructionu çalıştırmaya devam eder.

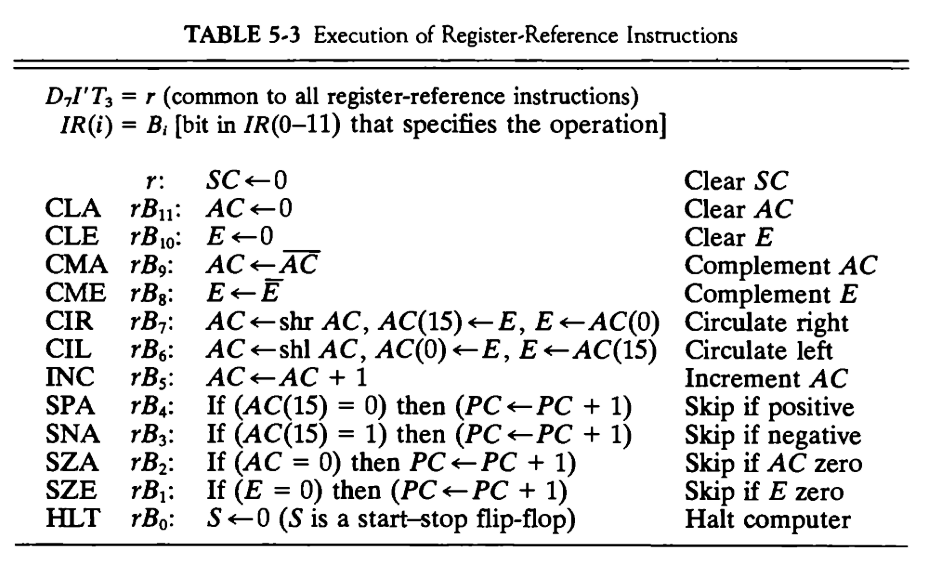
* Eğer D7T3 ise, T3 zamanında instruction execute edilir, sonra SC <- 0 olur ve başa dönülür. (T0 zamanına)



* AR <- M[AR] yapılırken M[AR]’nin LSB ilk 12 bit alınır.

**Register Reference Instructions**

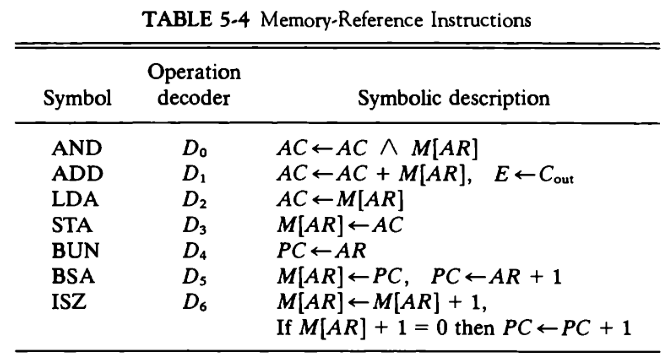
* “A register-reference instruction specifies an operation on or a test of the AC register.” Memoryden operand çekmesi gerekmiyor, memorye reference etme durumu yok.
* **T3 de execute** edilirler.
* D7I’T3 kontrol fonksiyonuna sahip. Bu r olarak kısaltılır.
* IR(11-0)’ın her bir biti Bi olarak kısaltılır. Yani 11.biti 1 ise B11 olur. Yani yukarıdaki tabloya da bakarak CLA instructionunun 7800 olduğunu ve binary açılımının 0111 1000 0000 0000 olduğu görülürse bu rB11 olmuş olur.



* HLT instructionu, start-stop flip-flopu olan S’yi sıfırlar ve sequence counterin saymasını durdurur. Devam ettirmek için S yi manuel olarak 1 e almak gerekir.

**Memory Reference Instructions**

* Executionu **T4 ile** başlar, T3’de addressing türü belirlenir, indirect ise gidip effective bulunur vs.



* Yukardaki **AR’ler effective address**. Her işlemin micrrooperasyonları aşağıda.

1. **AND**

D0T4: DR <- M[AR]

D0T5: AC <- AC AND DR, SC <- 0

1. **ADD**

D1T4: DR <- M[AR]

D1T5: AC <- AC + DR, E <- Carry out, SC <- 0

1. **LDA (LOAD TO AC)**

D2T4: DR <- M[AR]

D2T5: AC <- DR, SC <- 0

Bellekten verinin okunması ve aludan geçmesi, 1 clock cycle’ı aşabileceğinden (adder and logic circuitteki gecikmelerden dolayı), DR olmadan direkt girişi AC’ye yönlendirirsek, işlem tam olarak bitmeden bir sonraki cycle’a geçilebilme durumu olabilir. Dolayısıyla araya tampon olarak DR’yi alıp mikrooperasyonun 1 clock cycle’da bitmesini garantileriz.

1. **STA (Store AC)**

D3T4: M[AR] <- AC, SC <- 0

Adder and logic circuitteki gecikme gibi bir gecikme olmadığından tek cycle’da halledilebilir.

1. **BUN (Branch Unconditionally)**

BUN ile program akışının başka bir instructiona atlaması sağlanır.

PC, bir sonraki cycle’da memoryden okunacak adresi içeriyordu. T1 zamanında zaten 1 kere arttırılmıştı.

D4T4: PC <- AR, SC <- 0

1. **BSA (Branch and Save Return Address)**

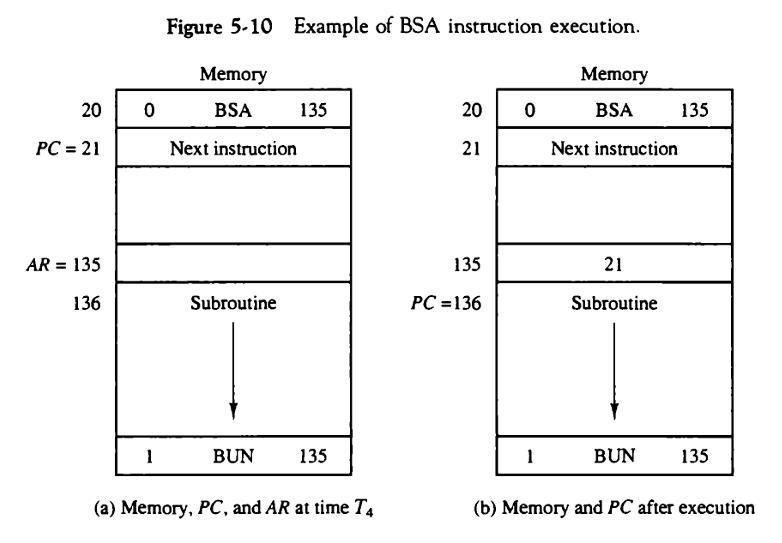
Subroutine call’larda (fonksiyon) kullanılır. Bittikten sonra da return adresi bulunabilmesi için kaydedilir.

Özünde yaptığı işlem M[AR] <- PC, PC <- AR + 1, fakat bu tek clock cycle’da yapılamaz bus ve memorynin doğru ayarlanabilmesi için.

D5T4: M[AR] <- PC, AR <- AR + 1

D5T5: PC <- AR, SC <- 0

Gibi 2 cycle da yapılması gerekir. Aşağıda örneği var. Fonksiyon sonuna geldiğinde return edilebilmesi için **indirect** BUN kullanılır.



1. **ISZ (Increment and Skip if Zero)**

Adı üstünde verilen bellek kelimesi +1 edilir, eğer sıfıra eşitse PC <- PC + 1 olur. Kullanımıyla ilgili daha ayrınıtlı bir bilgi yok. PC’nin arttırılması için önce DR’ye alınması lazım, sonra geri yerine yazılıyor.

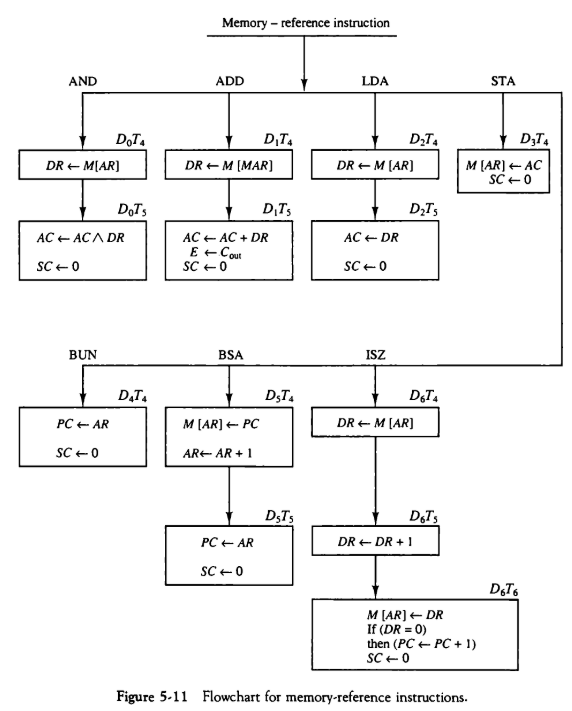
D6T4: DR <- M[AR]

D6T5: DR <- DR + 1

D6T6: M[AR] <- DR, if (DR = 0) then (PC <- PC + 1), SC <- 0

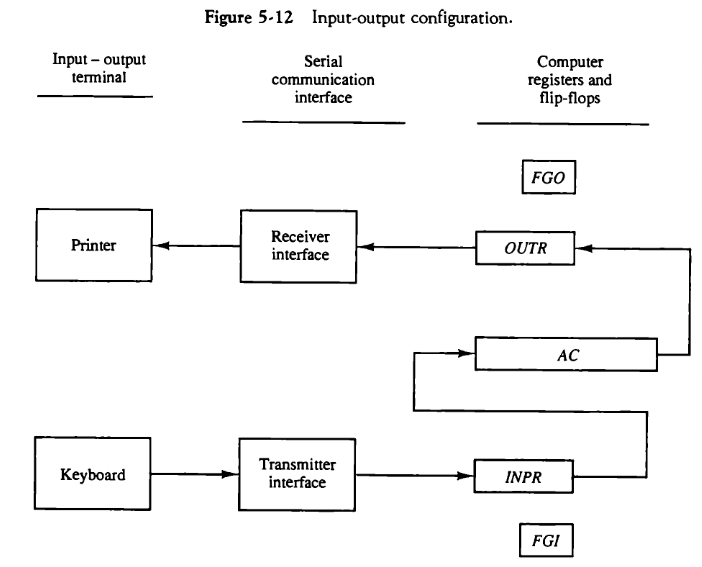
* Aslında en uzun süren ISZ’den de görülebileceği gibi (sequence counter 0’dan 6’ye kadar saydı) 3 bitlik sequence counter’da yeterli olabilirdi. En başta 4-bit alınmasının nedeni karmaşık işlemlerin de yapılabilmesi için.

ÖZET:

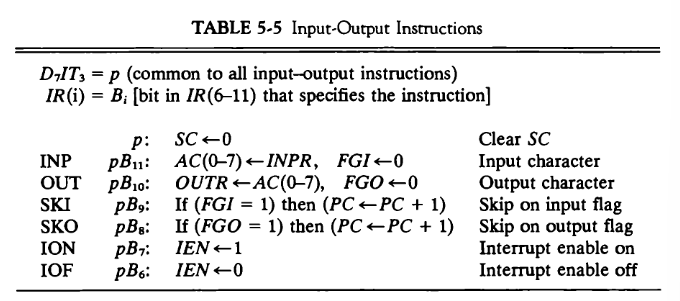


**Input-Output and Interrupts**

* Instructionlar ve memoryde saklanan veriler, input cihazından gelmeli.
* INPR ve OUTR 8-bit registerlar. Keyboarddan alınan seri input, transmitter interface aracılığıyla seri olarak INPR’ ye yazılır, aynı şekilde OUTR de seri olarak receiver interface’e gönderilir, sonra yine seri olarak printera gönderilir. INPR ve OUTR, AC’ye paralel olarak veri yazıp okur.



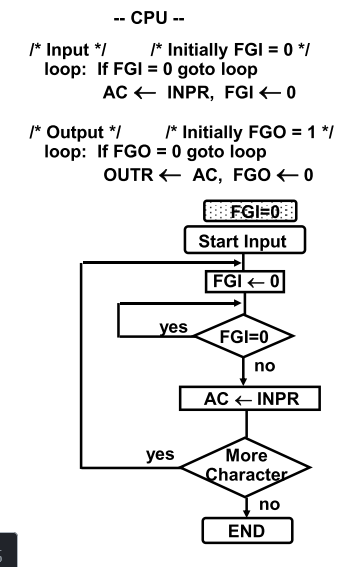
* FGI en başta 0 olarak başlar, veri girişi olduğunda flag 1’e ayarlanır ve veri INPR’e shift edilir. Flag 1 olduğu sürece INPR’in içeriği değiştirilemez. Bilgisayar flag bite bakar, eğer 1 ise INPR, paralel olarak AC’ye alınır, ve FGI 0’lanır.
* OUTR için işler biraz daha farklı. En başta outputun flip-flopu olan FGO, 1 e set edilir, sonra bilgisayar bakar, eğer bu flip-flop 1 ise AC’deki veri paralel olarak OUTR’ye alınır ve FGO 0’a set edilir. Sonra output cihazı bunu alır, yazdırır vs. Kendi işlemlerini tamamladığı zaman FGO tekrar 1’e set edilir. FGO 0 olduğu sürece AC’den OUTR’ye veri aktarılamaz.
* FGI ve FGO flag’leri, input device ve bilgisayarın zaman farkını senkronize etmek için kullanılır.



* Register reference’daki gibi, hepsinde D7IT3 ortak olduğu için kısaltılarak p denir. Yine register reference instructionlardaki gibi IR(6-11)’de hangi bit aktifse o bit Bi olarak alınır.
* INPR ve OUTR, 8 bit olduğundan AC’ye aktarılırken ilk 8 bit aktarılır.
* SKI ve SKO’daki skip edilen instructionlar, genelde flag’ın yeniden kontrol edilmesini sağlayan branch instructionları oluyor. (IEN, interrupt enable flip-flopu)

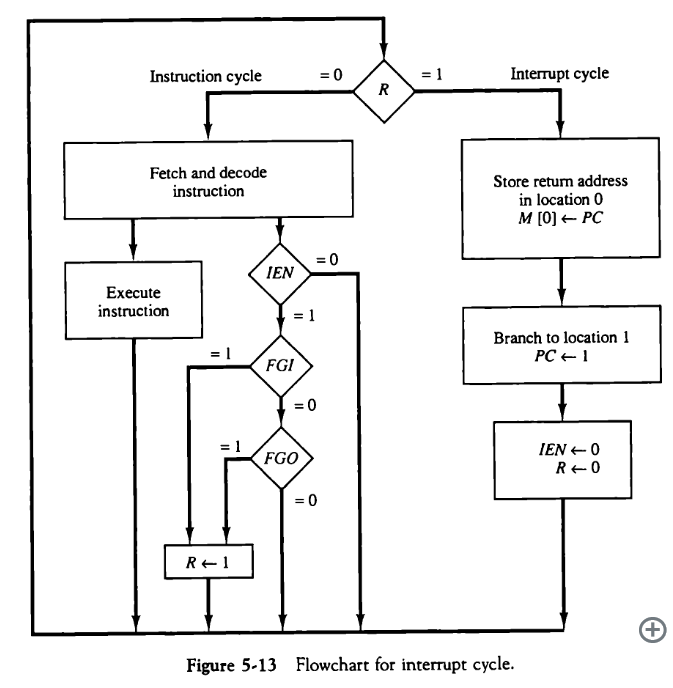
**Program Interrupt**

* Programlanmış Kontrol Transferi: Bu yöntemde, bilgisayar düzenli aralıklarla bir bayrak bitini kontrol eder. Bayrak biti ayarlandığında, bilgisayar bilgi aktarımını başlatır. Ancak, bilgisayar ve giriş/çıkış cihazı arasındaki bilgi akış hızı farkı nedeniyle bu tür bir transfer verimsizdir. Bilgisayar, bayrak bitini kontrol ederken boşa zaman harcar ve bu süre içinde başka işlemler yapabilir.



* Interrupt’lerin Kullanımı: Interrupt’ler, harici bir cihazın bilgisayara bilgi aktarımı için daha etkili bir yöntem sağlar. Harici cihaz, bilgisayara bilgi aktarımı için hazır olduğunda, interrupt sinyali gönderir. Bu sinyal, bilgisayarın mevcut programını geçici olarak keser, giriş/çıkış transferini gerçekleştirir ve ardından programın kaldığı yerden devam etmesine izin verir.

I/O device’ını CPU yerine I/O arayüzü gözlemler.

****

* Bir interrupt varsa 2. interrupt gelemez, IEN e bakılıyor çünkü. Interrupt enable 1 ise flag’lerin kontrolüne devam ediliyor.
* R, interrupt flip-flopu eğer 0 ise instruction cycle’ına girilir, IEN (interrupt enable) kontrol edilir, eğer 0 ise bu programcının interruptu kullanmak istemediği anlamına gelir, program bir sonraki cycle ile devam eder. Eğer IEN 1 ise, FGI ve FGO’ya bakılır, eğer ikisi de 0 ise INPR ve OUTR registerlarının hazır olmadığı anlamına gelir, bir sonraki cycle’a geçilir. Eğer FGO ya da FGI 1 ise, R 1’e set edilir ve execution bitişinde başa dönülüp R tekrar kontrol edilir. Eğer 1 ise interrupt cycle’a geçlir.
* Burada PC, önceden belirlenen bir konuma, örnekte 0. bellek konumuna yazılır. Sonra PC, 1’e getirilir. 1. Konumda da, I/O’nun handle edileceği programın konumuna BUN ile atlanır. Buradaki I/O programı flag’leri kontrol eder, gereken input-output informationları transfer edilir. Bu işlem bittiğinde, ION instructionu çalıştırılır ve IEN 1’e set edilir, yani bilgisayar interrupt uygulanabilir hale getirilir. I/O programının sonunda PC’yi 0’a atlatacak **indirect** instruction bulunur, 0’da da ilk programın kaldığı yerin adresi tutuluyordu, böylece program kaldığı yerden devam eder.

**Interrupt** **Cycle**

* Interrupt cycle, BSA operasyonunun hardware implementasyonudur.
* R, interrupt enable (IEN) ve; FGO ve FGI’dan en az biri aktif olduğunda 1 e set edilecek, ayrıca T0, T1 ve T2 dışında herhangi bir yerde set edilebilir. Yani:

T0’T1’T2’IEN(FGI + FGO): R <- 1 olur. (Buradaki + logic OR)

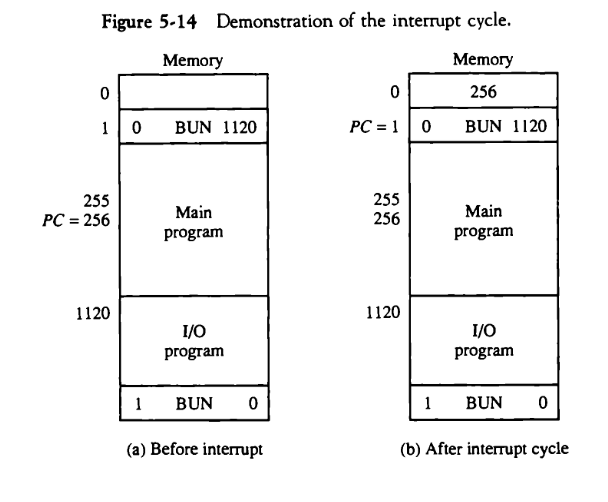
* Sonra, eğer interrupt cycle’a girilmişse,

RT0: AR <- 0, TR <- PC

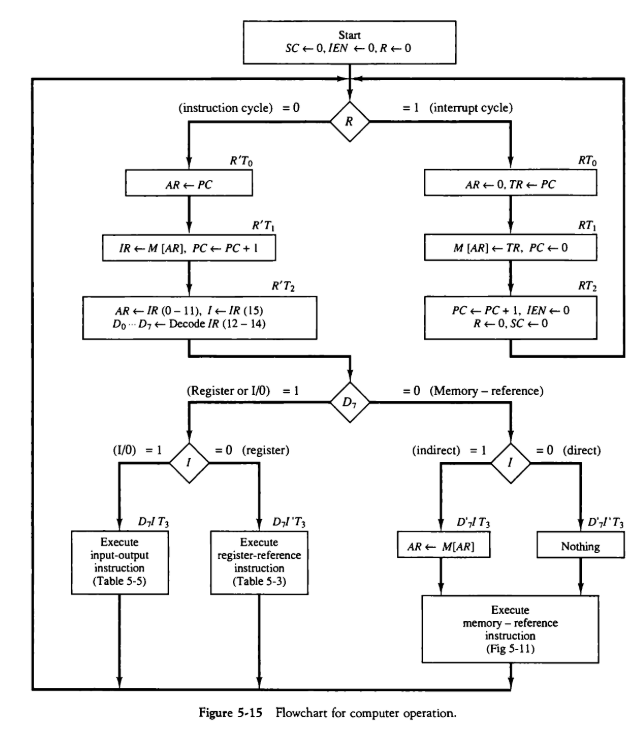
RT1: M[AR] <- TR, PC <- 0

RT2: PC <- PC + 1, IEN <- 0, R <-0 SC <- 0 olacak ve PC, I/O service programına yönlendirilmiş olacak.

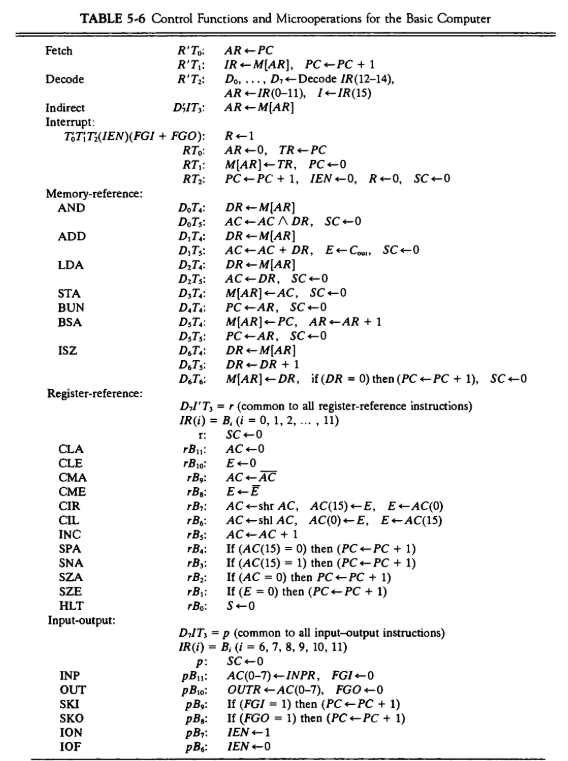
* Özetle, eğer FGI ya da FGO’dan herhangi biri, execution aşamasında 1 olursa (T0 T1 T2’den sonra ve interruptlar enable), instruction akışı ramde bulunan I/O service programına yönlendirilir. Interrupt cycle ile:



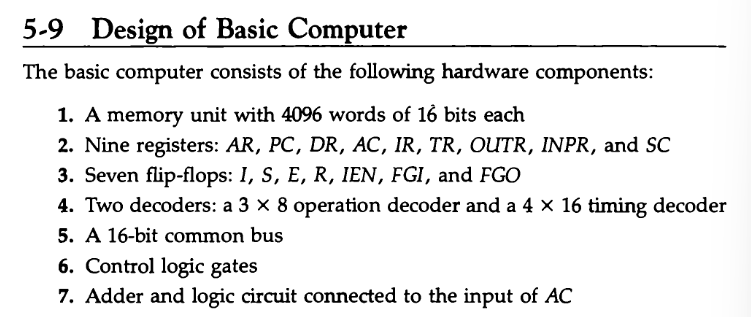
ÖZET:



* Tüm kontrol fonksiyonu ve mikrooperasyonların toplanmış hali:

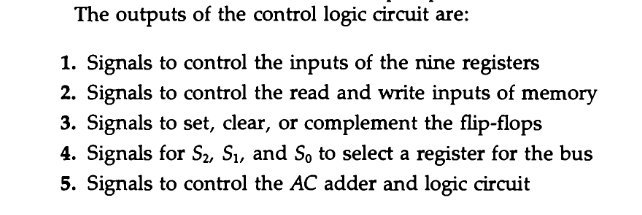
,

**Design of Basic Computer**

****

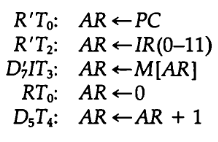
**Control Logic Gates**

* Figür 5-6’daki T0, T... ve D0, D... inputlarına ek olarak, AC = 0 kontrolü ve AC’nin sign bit kontrolü için AC(0-15) ve DR = 0 kontrolü için DR(0-15) ve yukarıdaki I, S, E gibi 7 flip-flop; control logicin inputu olarak yer alır.

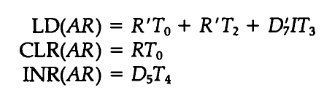


**Control of registers and Memory**

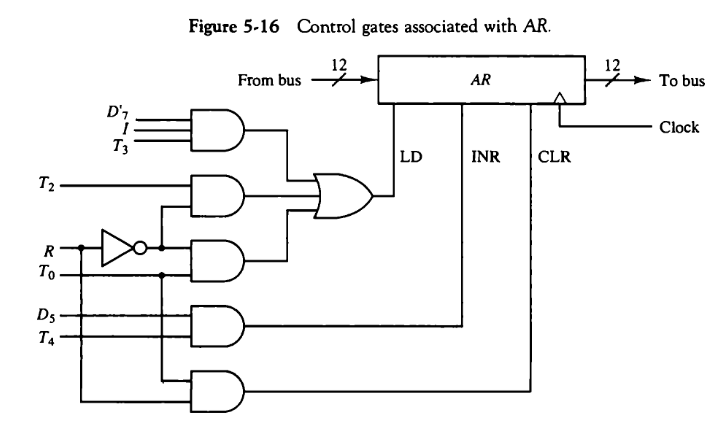
* Register ve memorylerin kontrol inputlarını sağlamak için, tablo 5-6’dan yararlanabiliriz. Mesela AR’nin inputlarının kontrolleri bulunmak isteniyorsa, AR’nin inputlarını etkileyen operasyonlar seçilir:



sonra input türüne göre birleştirilir:



devrelerin her biri gerekli inputlara bağlanır:



* Memory için de aynı yöntemle bulunuyor.

**Control of Single Flip-Flops**

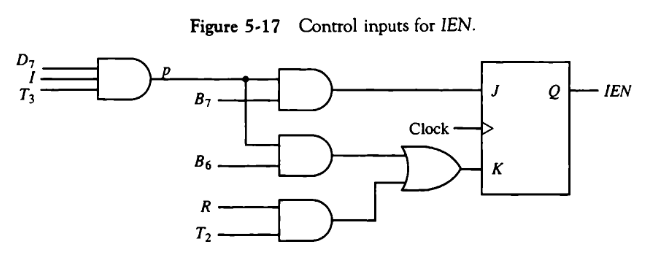
* Yine 5-6’dan, ilgili flipflopu etkileyen operasyonlar tablo 5-6’dan bulunur.

pB7: IEN <- 1

pb6: IEN <- 0

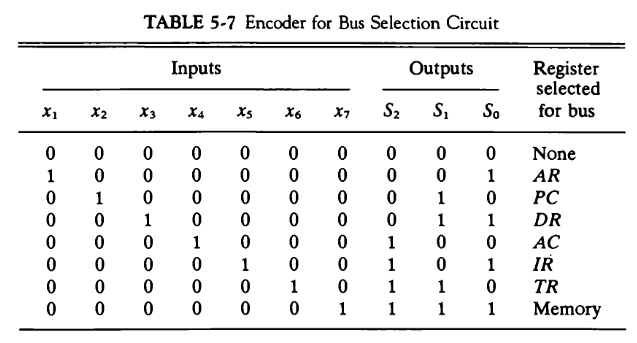
RT2: IEN <- 0

Flipflopun türü jk seçilirse mesela, set reset aşağıdaki gibi kolayca ayarlanır.



**Control of Common Bus**

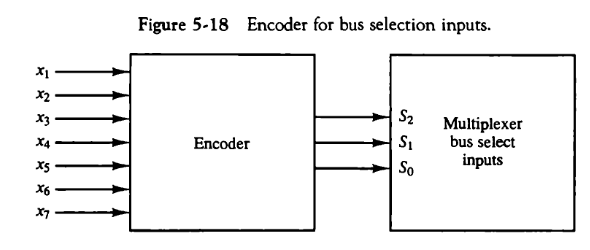
* 5-7’deki tablodan yararlanılır:

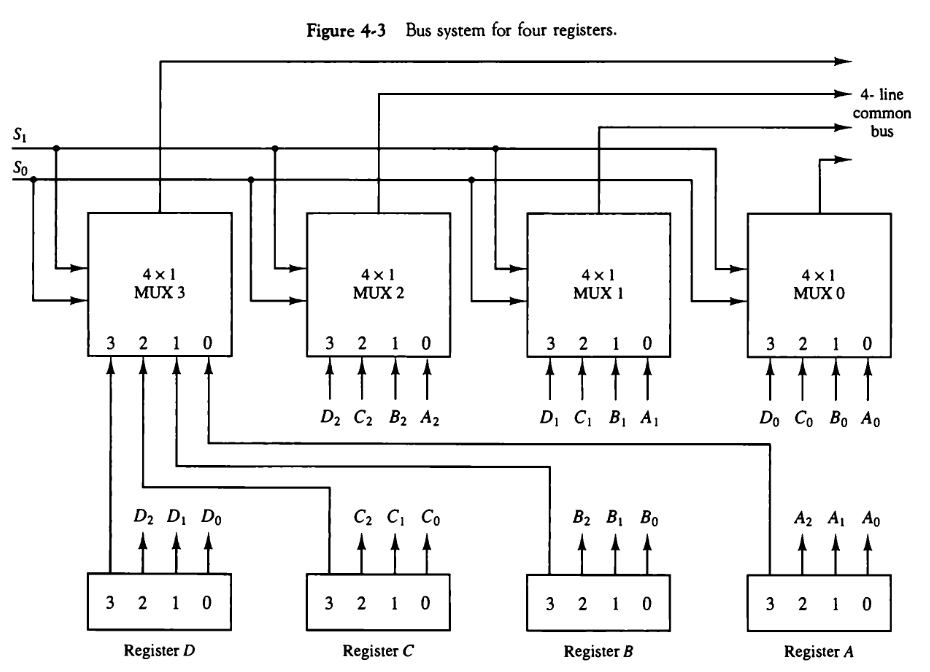


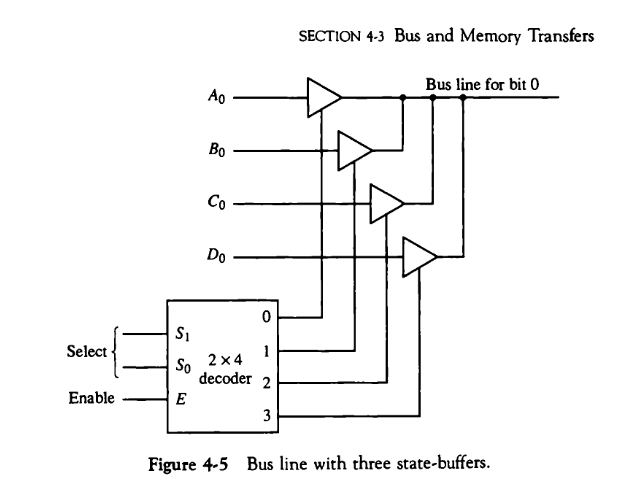
* xi’lerin girdi olduğu bir encoder var, encoderin outputu s2s1s0 ı üretiyor. Her xi’nin boolean functionu bulunmalı. Mesela tabloya göre x1, AR’nin bus’a geçişini sağlamalı. Dolayısıyla tablo 5-6’dan, AR’yi bus’ta geçirecek statementler bulunur:



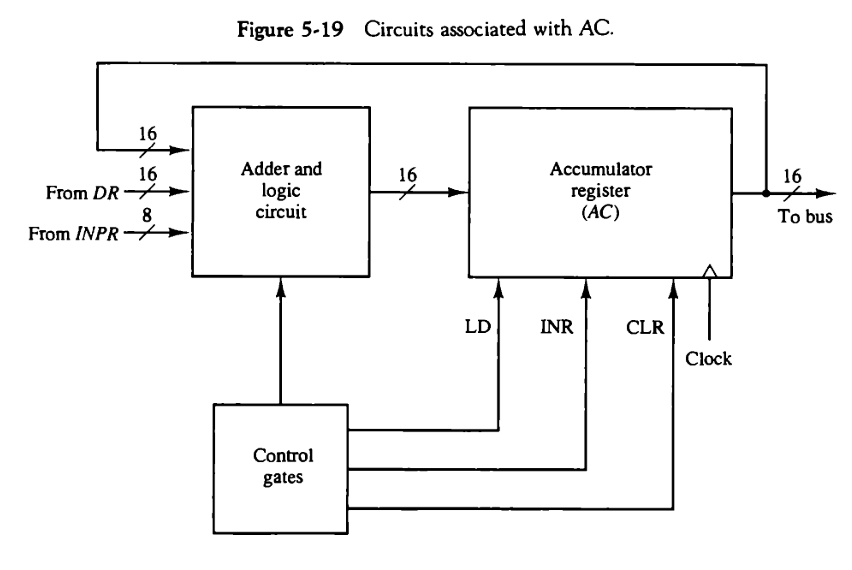
Yani x1 = D4T4 + D5T5 olmuş olur. Bu tüm xi’ler için yapılır ve x’li girişlere bağlanır.



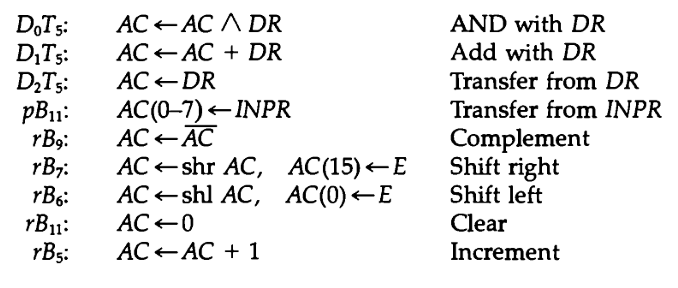




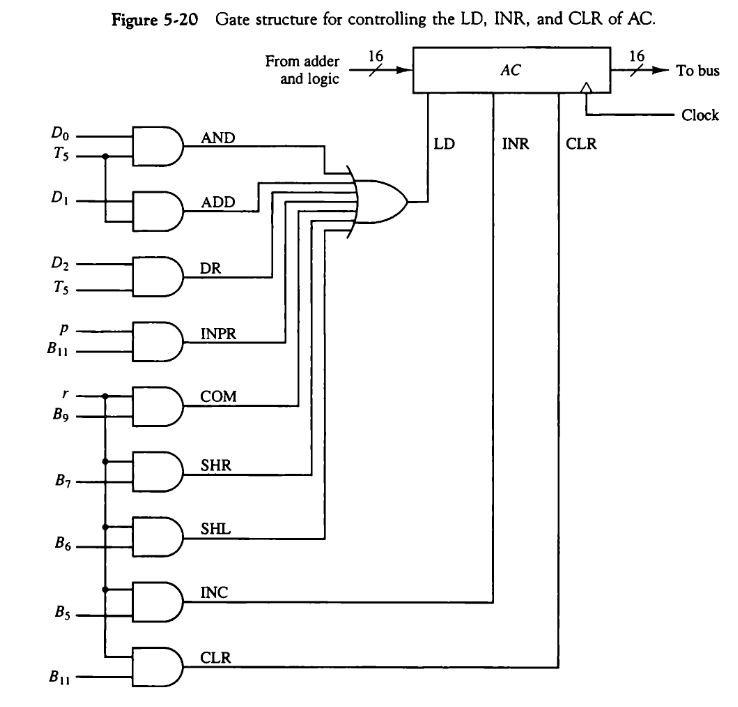
**,Design of Accumulator Logic**

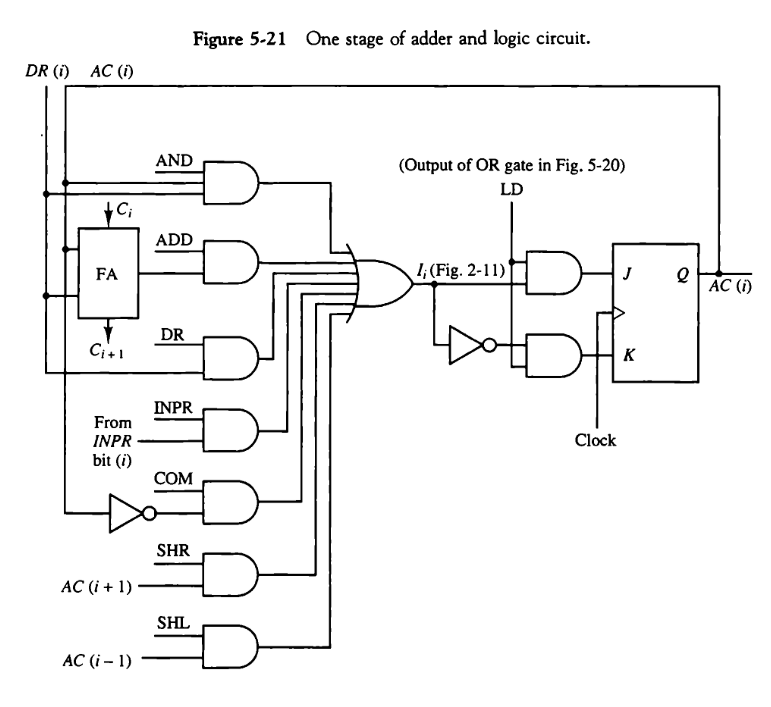


* AC nin giriş çıkışları yukarıdaki gibi. Kontrol inputlarını bulmak için yine tablo 5-6’da, AC’nin içeriğini değiştiren kontrol fonksiyonları bulunur:



Sonra yine LD, INR ve CLR’ı tetikleyenler ayrılıp birleştirilir ve en son bu devre elde edilir:

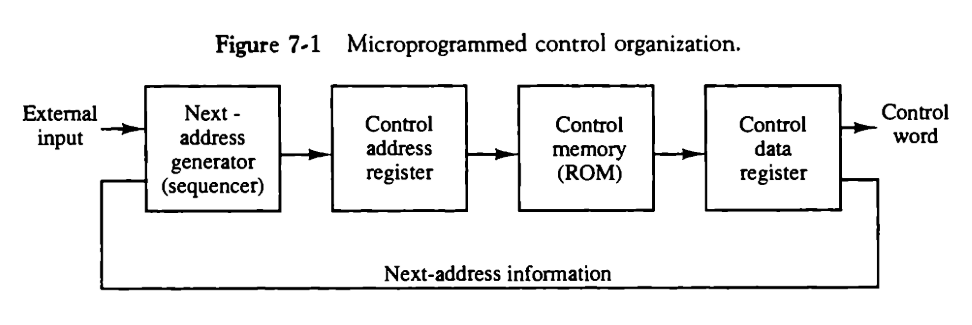




* Adder and logic circuit’in iç yapısı yukarıda verilmiştir. Buradaki AND, ADD gibi labellenmiş kablolar bir üst resimde, kendileriyle aynı labeldeki output kablolarına bağlanmış olacaklar. Bu sadece 1 bitin temsili gösterimi, bundan 16 tane var AC’nin her bir biti için.
* JK flipflopun da, NOT kapısıyla birlikte düşünüldüğünde D flipflopa dönüştüğü gözlemlenebilir.

**CHAPTER 7**

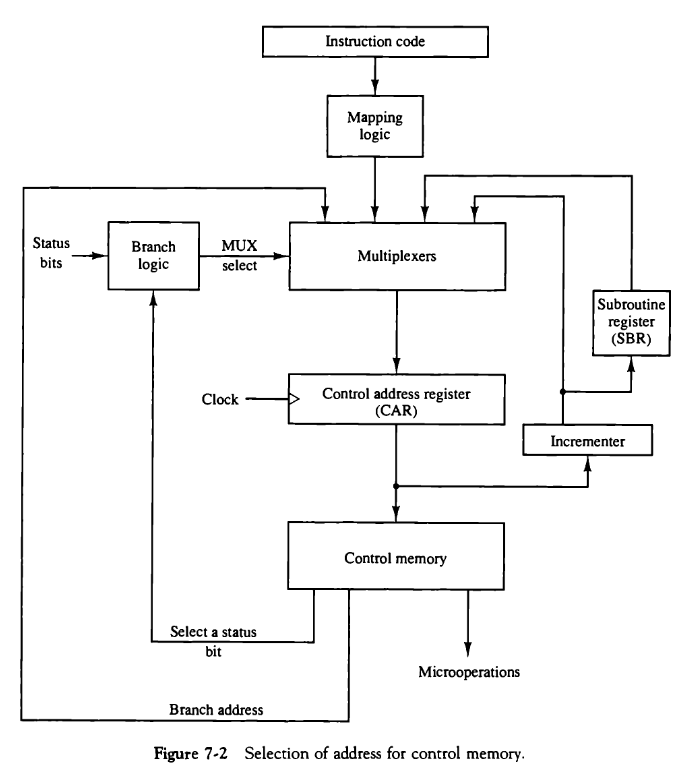
* A control unit whose binary control variables are stored in memory is called a microprogrammed control unit. Each word in control memory contains within it a microinstruction.
* The microinstruction specifies one or more microoperations for the system. A sequence of microinstructions constitutes a microprogram.
* Control memory ROM, çünkü kontrol birimi çalışırken değiştirilmeye ihtiyaç duymuyor.
* Dinamik mikroprogramlama, bir mikroprogramın başlangıçta manyetik disk gibi yardımcı bir bellekten yüklenmesine olanak tanıyan gelişmiş bir tekniktir. Dinamik mikroprogramlama kullanan kontrol birimleri, yazılabilir bir kontrol belleği (Writable Control Memory) kullanır. Bu bellek, mikroprogramı değiştirmek için yazma işlemleri yapabilen, ancak çoğunlukla okuma işlemleri için kullanılan bir bellek türüdür.



* The next address generator is sometimes called a microprogram sequencer, as it determines the address sequence that is read from control memory.
* Control memory address register specifies the address of the microinstruction, and the control data register holds the microinstruction read from memory.
* The control data register is sometimes called a pipeline register. It allows the execution of the microoperations specified by the control word simultaneously with the generation of the next microinstruction. Bu işlem 2 fazlı clock gerektirir, birisi address register için, diğeri data register için.
* Control data register olmadan, control address register’e tek clock bağlanarak sistem oluşturulabilir. In this way the address register is the only component in the control system that receives clock pulses. (The other two components: the sequencer and the control memory are combinational circuits and do not need a clock.)
* Reduced instruction set (RISC) mimarisi üzerine kurulu çoğu bilgisayar, microprogrammed yerine hardwired control kullanır.

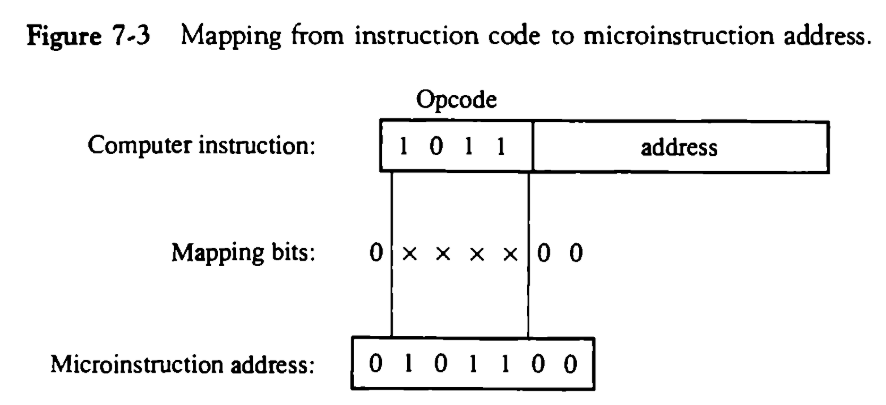
**Address Sequencing**

* Kontrol memoryde, microinstructionlar gruplar halinde bulunur, her grup routine belirtir.
* Bilgisayardaki her instructionun kontrol memoryde microprogram routine i var.
* A mapping procedure is a rule that transforms the instruction code into a control memory address.
* Microprogram sequencer görevleri:
  + Incrementing of the control address register.
  + Unconditional branch or conditional branch, depending on status bit conditions.
  + A mapping process from the bits of the instruction to an address for control memory.
  + A facility for subroutine call and return.



* Status bit’leri (adder’in carry outu, bir sayının işaret biti vs.) conditional branch durumunu etkiler.
* Branch logic kendisine gelen status bitleri değerlendirir, eğer koşul sağlanıyorsa belirtilen adrese branch eder, sağlanmıyorsa address register increment edilir.

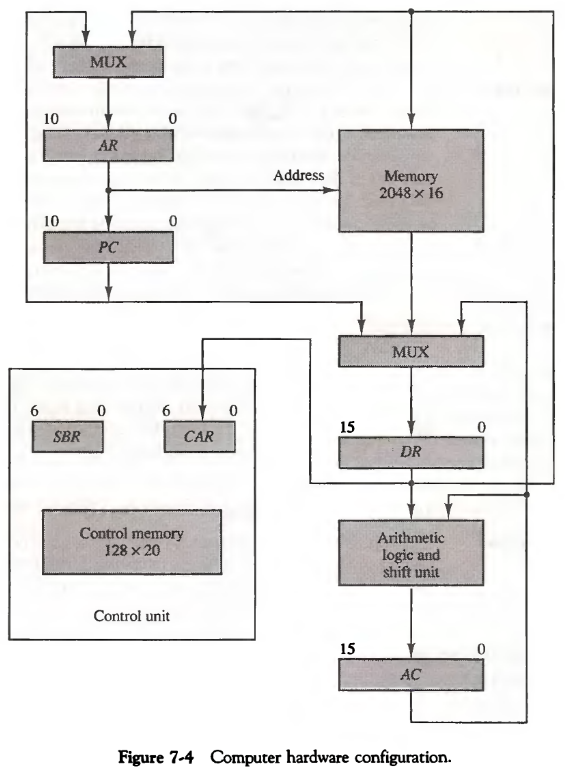
**Instruction Mapping**

****

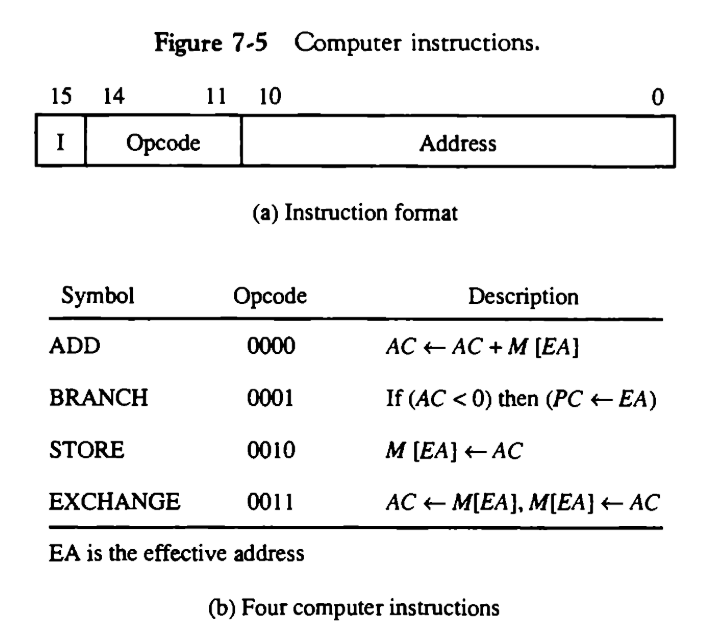
**Subroutines**

* Subroutinler, belirli bir görevi yapmak için diğer routine’ler tarafından kullanılan programlardır.
* Subroutinlerin geri döneceği addresler kaydedilmeli. (LIFO)

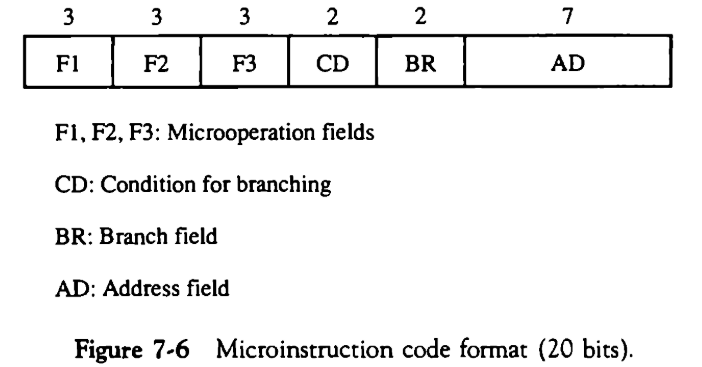
**Microprogrammed Digital Computer**



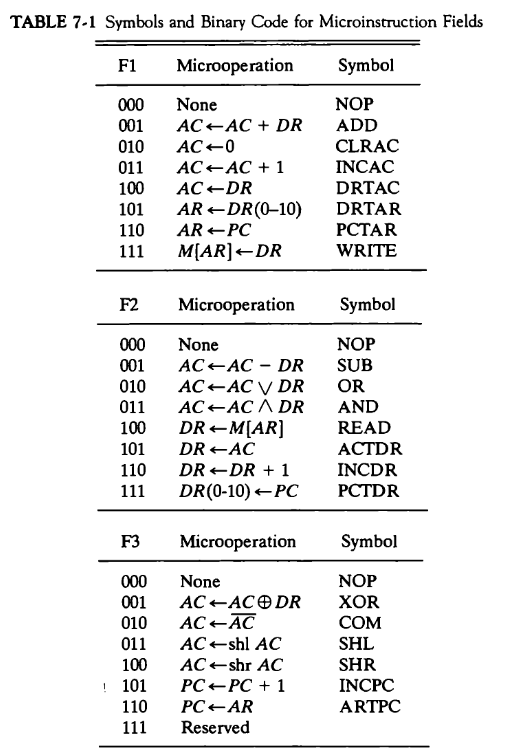
* Processor register’ları Chapter 5’teki gibi çalışıyor. Control unit, figür 7-2’deki organizasyona sahip.
* Common bus yapısı yerine MUX’lar kullanılıyor.



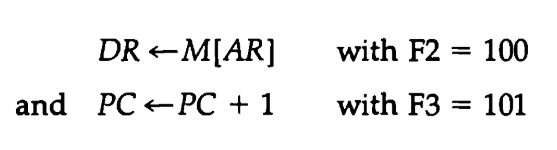
* Her instruction mikroprogramlanmalı. Karmaşıklığı azaltmak için sadece birkaç instruction verildi yukarıda.



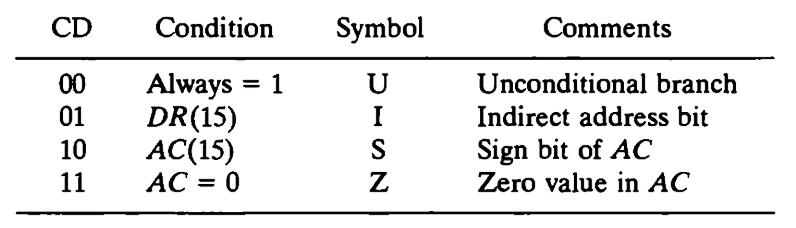
* Figür 7-6’da microinstruction kod formatı var. F1, F2 ve F3 mikrooperasyonu belirtiyor. CD alanı status bit conditionlarını seçiyor, BR branch türünü belirtiyor, AD branch adresini belirtiyor. AD 7 bit, çünkü control memory 128 kelimeye sahip.



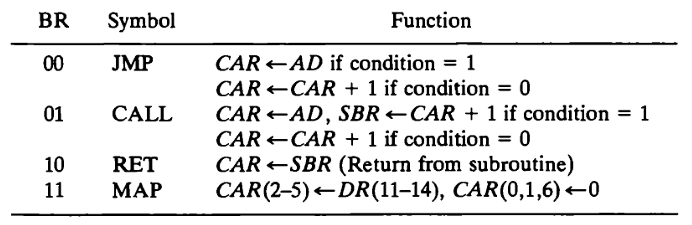
* Microinstructionlarda F1, F2 ve F3 field’ları eş zamanlı olarak çalışacak mikrooperasyonları belirtir. Mesela sırasıyla 000, 100, 101 olduğunda aşağıdaki gibi olur:



* Birbiriyle çakışan (mesela hem AC’yi sıfırlayan, hem de AC’ye atama yapan) 2 microoperation aynı anda belirtilemez.
* 00’ın always 1 olmasının sebebi, BR ile logic operasyonlara sokulduğunda unconditional branch sağlaması.



* CALL ile JMP farkı, CALL’da return adresini SBR’ye kaydetmesi. CALL ve JMP’nin condition’larını CD field belirliyor. RET ve MAP, CD ve AD field’larından bağımsız.

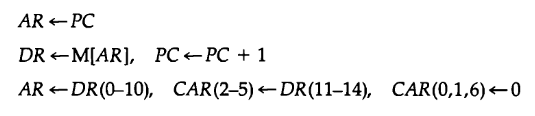


**Symbolic Microinstructions:**

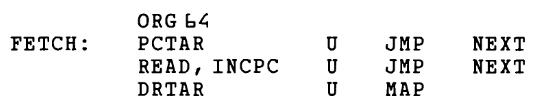
* Microprogram’lar Assembly dili kullanılarak symbolic microinstructionlar biçiminde yazılabilir. Her microinstruction 5 bölüme ayrılır: label, microoperations, CD, BR, AD. Genel gösterim:
  + Label kısmı boş olabilir, : ile biter.
  + Microoperationlar 1, 2 ya da 3 symbolden oluşabilir. Microoperasyon yoksa NOP symbolü kullanılır. (assembler tarafından 9 tane 0’a çevrilir)
  + CD alanı yukarıdaki CD tablosundaki sembollerden biri.
  + BR fieldi yukarıdaki BR tablosundaki sembollerden biri.
  + AD fieldi address field için 3 şekilde değer belirtir:
    - Symbolic adress olabilir (bu labelde de görünür olmalı)
    - NEXT olabilir, bir sonraki adresi gösterir.
    - BR fieldi RET yada MAP ise boş bırakılır (assembler tarafından 7 tane 0’a çevrilir)
* ORG psödoinstructionu, microprogram’ın başlangıç adresini belirtir.

**The Fetch Routine**

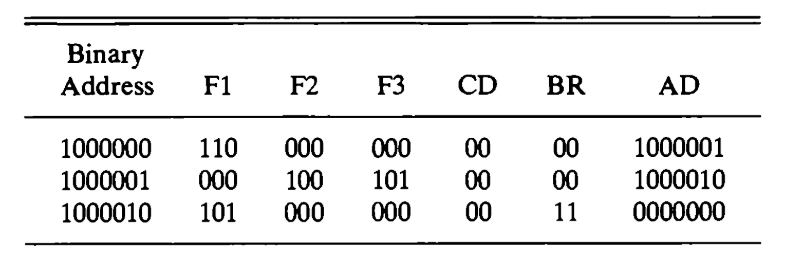
* Fetch aşaması için microinstructionlar üretilmesi gerekiyor.

****

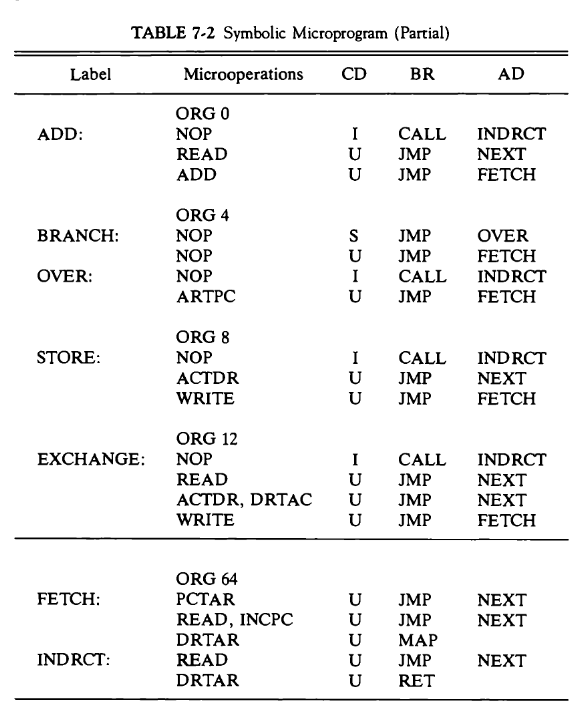
* Control memory’de fetch routininin başlangıç konumu 64 olarak belirlenirse:



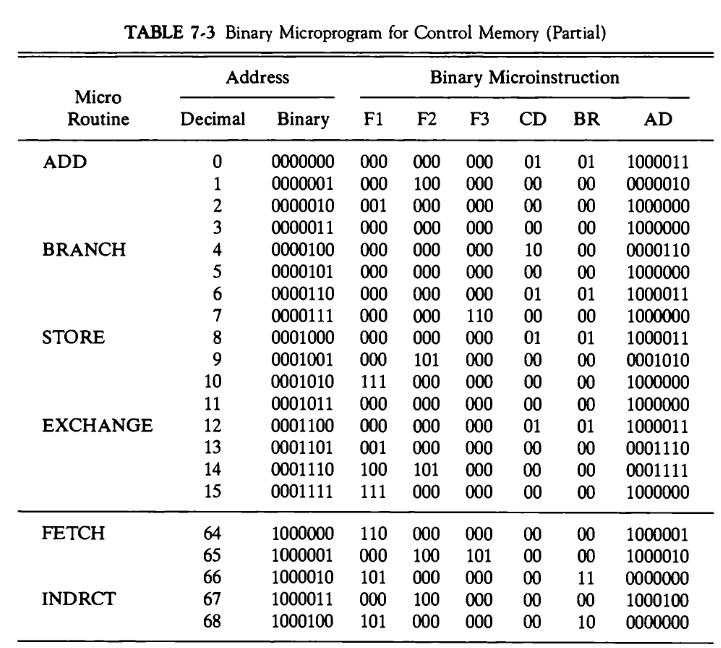
* Gereken 3 microinstruction control memory’de sırasıyla 64, 65 ve 66’da tutuluyor. Assembler, bunları aşağıdaki formata çevirir:



* Dikkat edilirse fetch routine’de son microinstruction 0xxxx00’a branch etmiş. Buradaki xxxx’ler AND operasyonunun opcode’undan geliyor. Map’leme işlemi yukarıda gösterildi.
* Instructiondaki I indirect bitine göre memory reference instruction routine’leri INDRCT subroutine’ini çağırabilir. Eğer subroutine çağırılırsa, return addresi SBR’e kaydedilir.



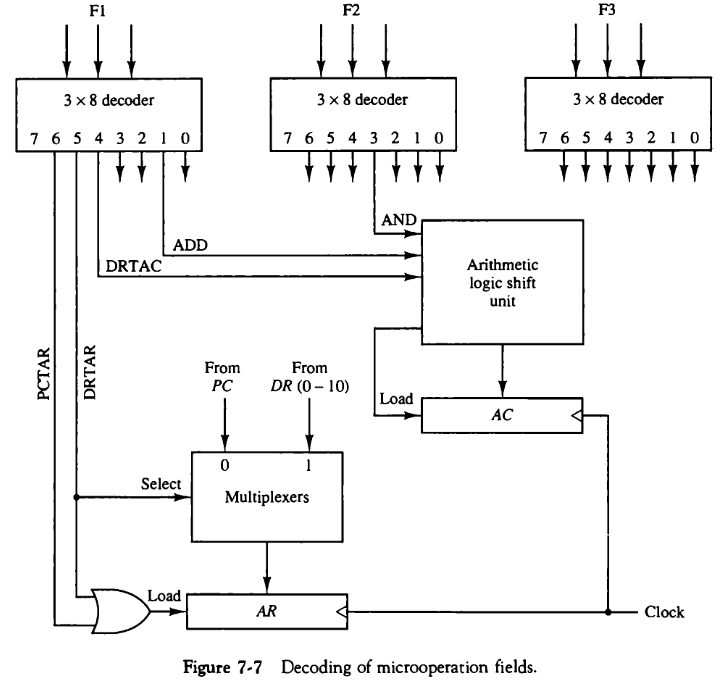
* Yukarıdaki symbolic microprogramın assembler tarafından dönüştürülmüş hali aşağıda. Burada 3. adres, AND routini tarafından kullanılmıyor, ADD routini sadece 0, 1 ve 2’nci adreslerdeki microinstructionları kullanıyor, bir sonraki BRANCH rouitini de 4’ncü adreste başlıyor. Burada 3’ncü adresteki tüm değerler sıfırlanabilirdi, fakat herhangi bir hata oluşup bir şekilde 3’e branch edilmesi durumunda sorun yaşamamak için FETCH routinine geri branch edecek şekilde ayarlandı.



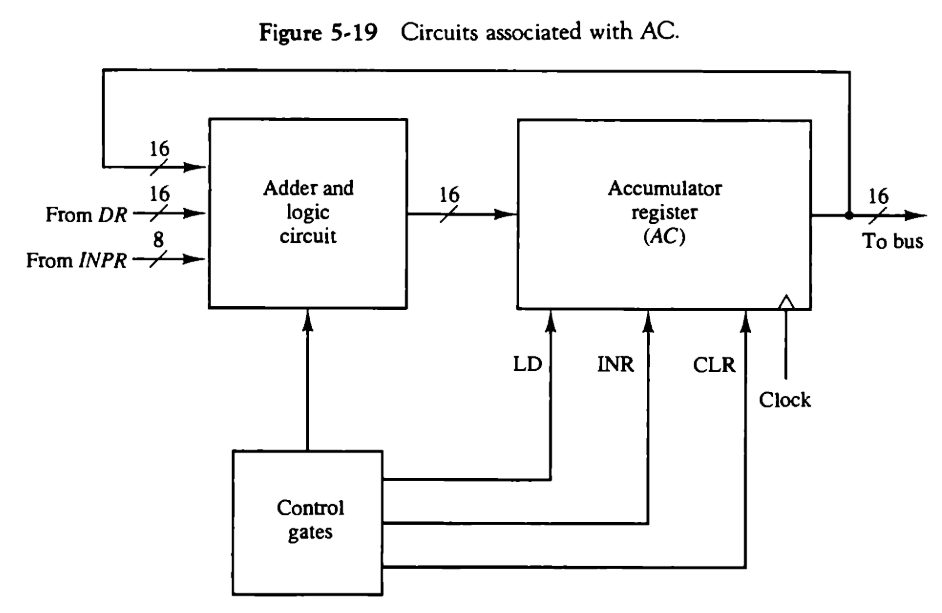
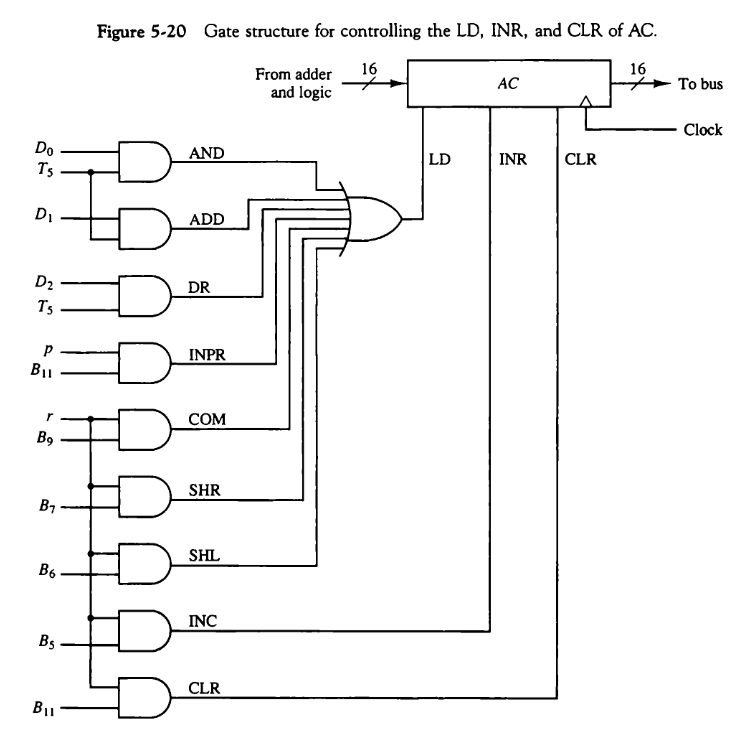
* ROM, RAM’e göre daha hızlı ve daha ucuz olduğu için çoğu microprogrammed sistemde daha çok kullanılıyor.

**Design of Control Unit**

* F1, F2 ve F3 field’ları aşağıdaki gibi decode edilir.

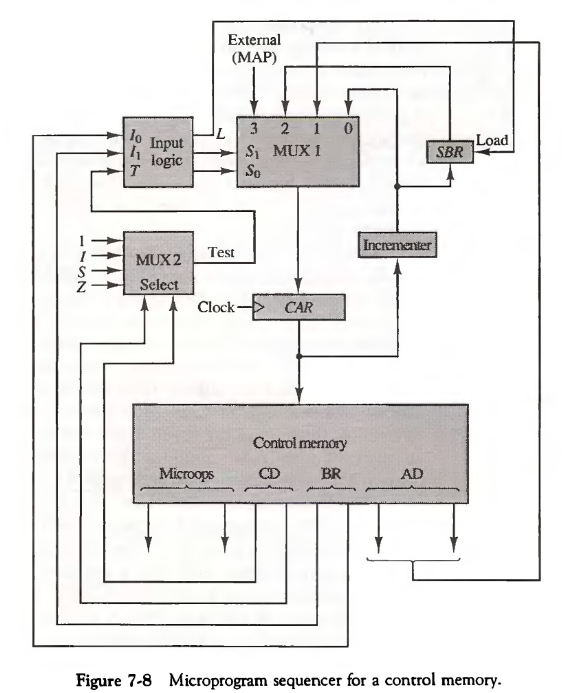


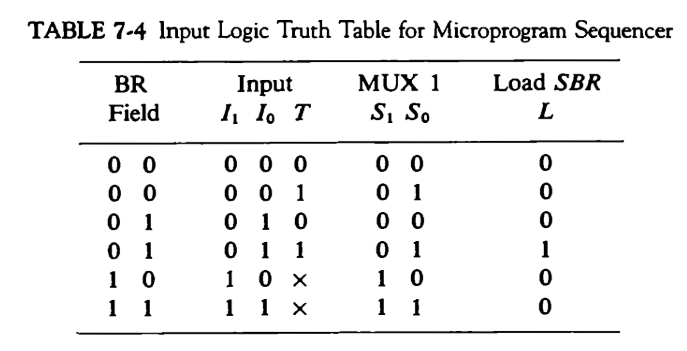
* Arithmetic logic unit’in dizaynı chapter 5’den figür 5-20 ve 5-19 gibi yapılabilir, tek fark orada yapıldığı gibi T ve D’leri gate’lerden geçirmek yerine direkt figür 7-7’deki decoder çıktıları figür 5-20’deki aynı isimli girişlere bağlanabilir.



**Microprogram Sequencer**

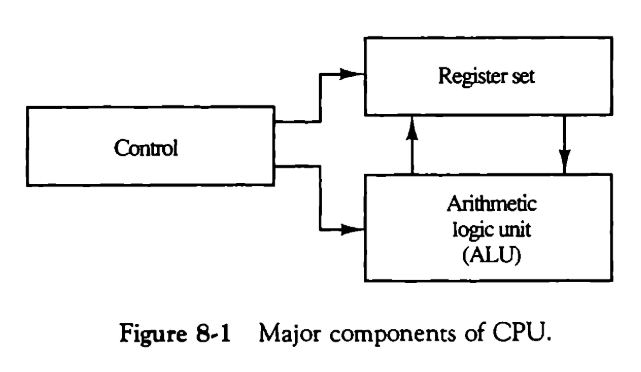
* Address seçme bölümü microprogram sequencer olarak adlandırılır.

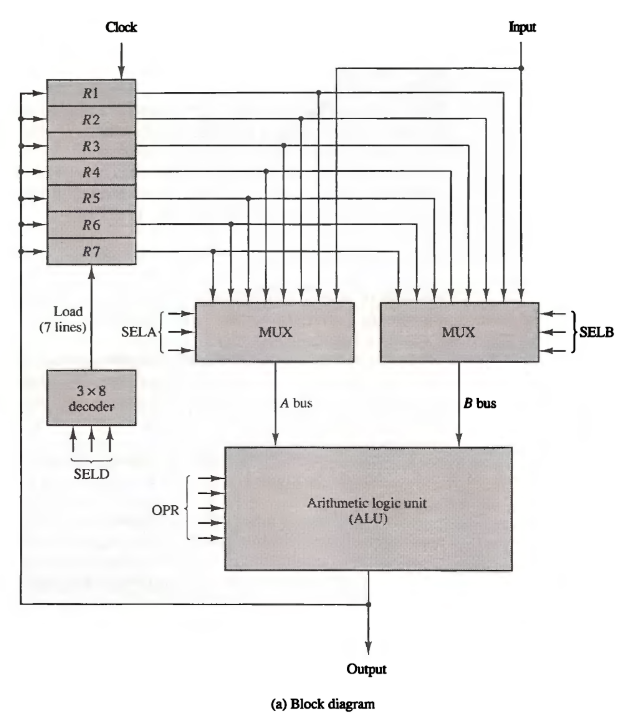


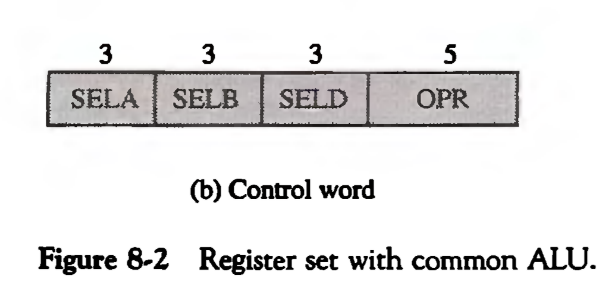


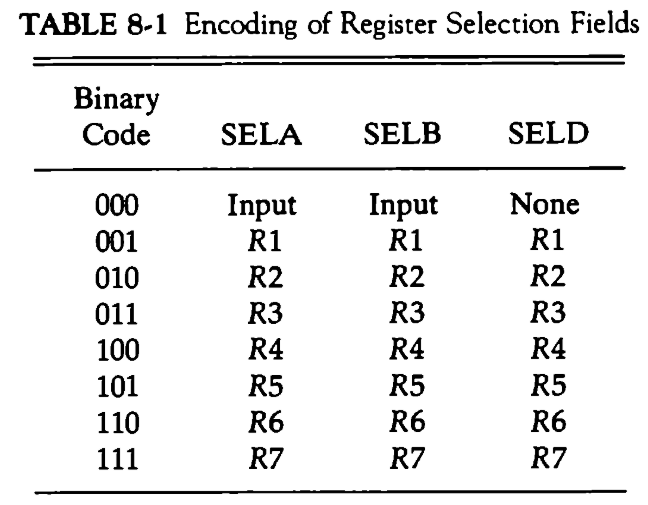
* Incrementer, sequence counter gibi flipflop içermez, combinational (half adder’lardan oluşur) circuittir.

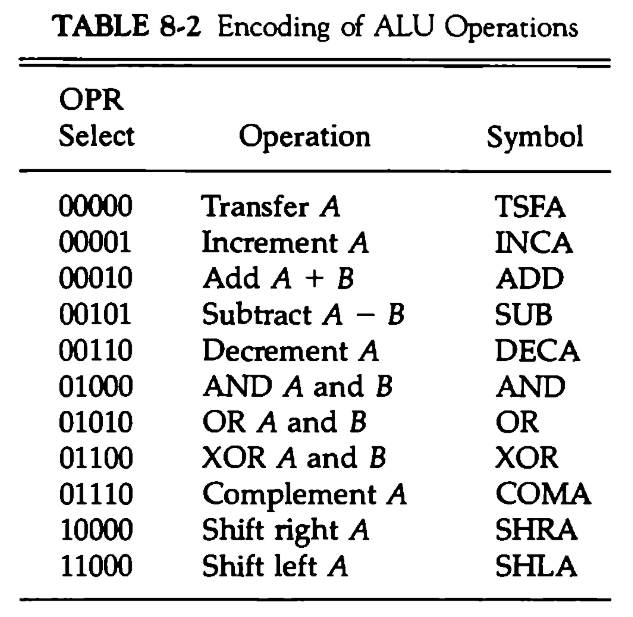
**CHAPTER 8**







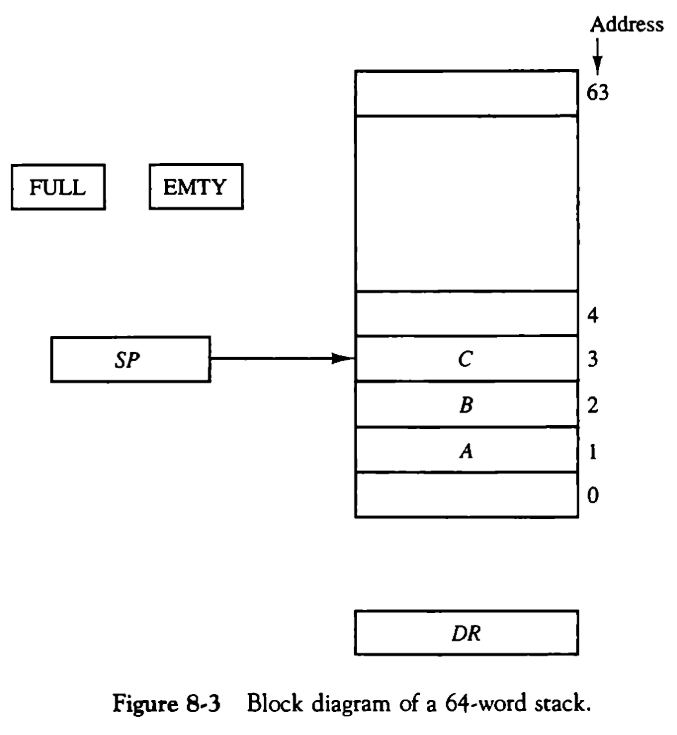




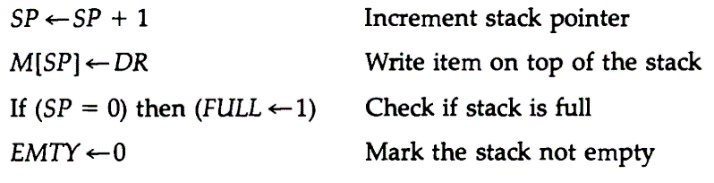


**Register Stack**

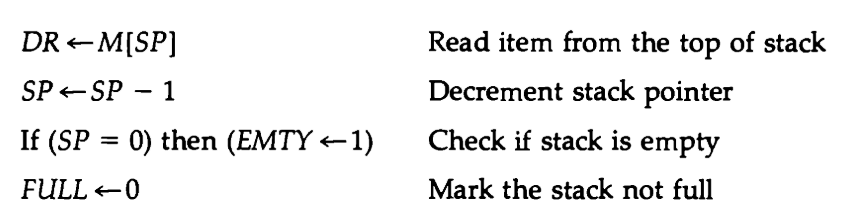
* Dijital bilgisayarlardaki stack, esasen bir adres kaydedicisiyle birlikte yalnızca sayma işlemi yapabilen (ilk değer yüklendikten sonra) bir bellek birimi.
* The register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top item in the stack.
* Nothing is pushed or popped in a computer stack. These operations are simulated by incrementing or decrementing the stack pointer register.
* A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers:



* Stack’e veri eklenmesi/silinmesi, SP’nin arttırılması/azaltılması anlamına geliyor.
* EMPTY, stack 1, FULL stack doluyken 1. DR stack’e eklenen, ya da stackten çıkarılan datayı temsil ediyor. Push operasyonunda gerçekleşenler:

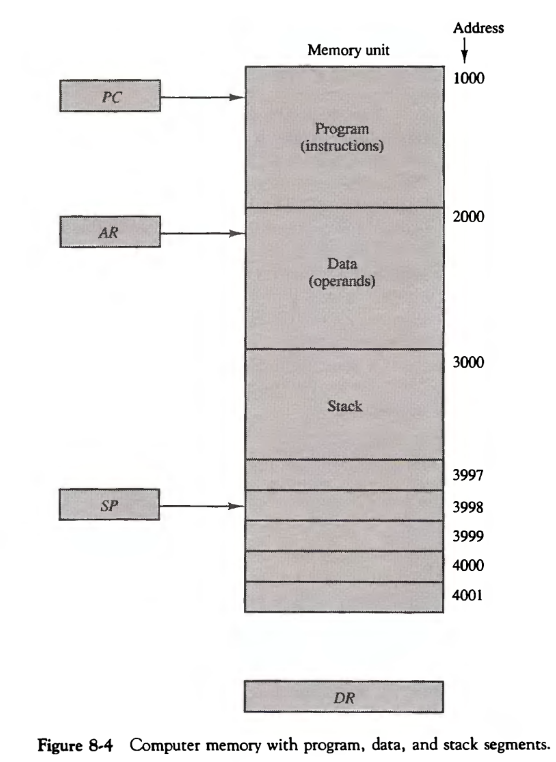


* Eklerken de aşağıdaki gibi:



**Memory Stack**

* Memory stack aşağıdaki gibi modellenebilir. SP azaldıkça (mimariye göre değişebilir!) stack’e ekleme yapılıyor, arttıkça pop. Stack’in tamamen boş ya da dolu olması için ayrı 2 processor register kullanılabilir, biri alt limiti diğeri üst limiti tutar, eleman eklenip çıkarılırken karşılaştırılır.



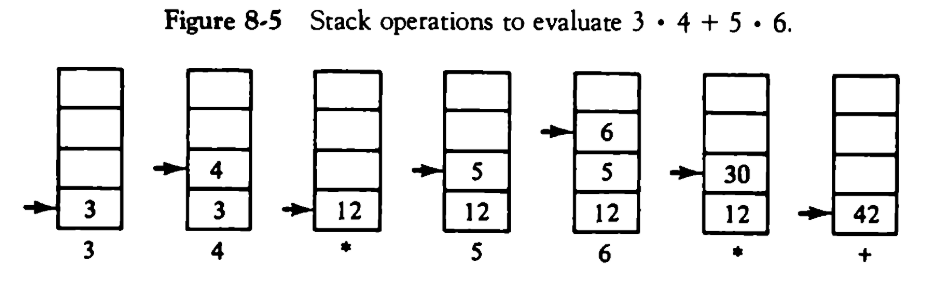
**Reverse Polish Notation**

* Infix notation: operatörler, operandların arasında: a \* b + c \* d. Bu hesaplamayı yapabilmek için \* operatörlerinin sonuçları kaydedilmeli, sonra gerektiğinde kullanılacak.
* Polish (prefix) notation: Operatörler, operandlardan önce geliyor: +AB
* Postfix (reverse polish [RPN]) notation: Operatörler, operandlardan sonra gelir: AB+
* Reverse polish notation stack manipulasyonu için uygun:

Expression: a \* b + c \* d,

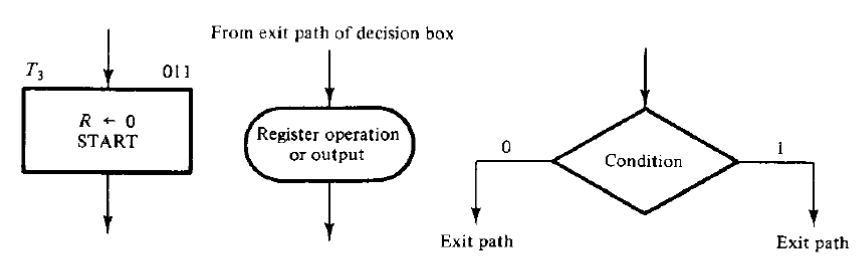
Reverse polish hali: ab\*cd\*+

* + Operatör bulana kadar soldan sağa git.
  + Operatör bulduğunda, operatörün solundaki 2 operandla işlemi yap. Operandları ve operatörü çıkar, yerlerine elde ettiğin sonucu yaz.
  + Operatör kalmayana kadar devam et.
* Infixten reverse polish’e dönüşüm esnasında infix’teki işlem önceliği dikkate alınmalıdır. Örneğin: (A + B)\*[C\*(D + E) + F] ise dönüşüm AB + DE + C \* F + \*
* Reverse Polish notation, combined with a stack arrangement of registers, is the most efficient way known for evaluating arithmetic expressions. The fact that any arithmetic expression can be expressed in parentheses-free Polish notation.
* Most compilers, irrespective of their CPU organization, convert all arithmetic expressions into Polish notation anyway because this is the most efficient method for translating arithmetic expressions into machine language instructions. So in essence, a stack-organized CPU may be more efficient in some applications than a CPU without a stack.
* Reverse polish notation’un stack ile kullanımı aşağıdaki gibi. İşlem öncelikle reverse polish’e dönüştürülüyor: 34\*56+



**Algorithmic State Machines (ASM)**

* Bir akış diyagramı türü. Dijital donanım algoritmalarını tanımlamak için kullanılır.
* 3 temel elementi var:
  + State box: Dikdörtgenle gösterilir, içinde register operasyonları ya da çıkış sinyalleri bulunur.
  + Decision box: Elmas şeklinde. Verilen girdinin etkisine göre yönlendirir.
  + Conditional box: Oval şekilli. Eğer kendisine input olarak verilen koşul sağlanmışsa içindeki register operasyonları ya da output sinyalleri generate edilir.
* ASM ile flowchart farkı, flowchartta işlemler sequentially bir şekilde gerçekleşir, ASM’de ise bloklar içindeki operasyonlar eşzamanlı olarak gerçekleşir.

****

**CISC**

* CISC makinelerinin bir hedefi, her yüksek seviyeli dil ifade türüne karşılık gelen bir makine dili talimatına sahip olmaktır.
* Karmaşık Talimat Seti: CISC makineleri çok sayıda talimat ve adresleme moduna sahiptir. Bu, her yüksek seviyeli dil ifadesi türü için bir makine dili talimatına sahip olma amacı taşır.
* Değişken Uzunlukta Talimatlar: Talimatların ve adresleme modlarının çokluğu, değişken uzunlukta talimat formatlarına yol açar.
* Bellek Üzerinde Doğrudan İşlem Yapma: Bellek adresleri üzerinde doğrudan işlem yapabilen talimatlara sahiptir.
* Mikroprogramlı Kontrol: Karmaşık talimatlar ve adresleme modları nedeniyle mikroprogramlı kontrol gereklidir.
* Yüksek Seviye Dil Desteği: Her yüksek seviyeli dil ifadesi türü için bir makine dili talimatı sağlayarak programlamayı kolaylaştırır.
* Bellek Kullanımı: Değişken uzunlukta talimatlar, sık kullanılan talimatlar için kısa opkodlar kullanılarak bellek verimli bir şekilde kullanılabilir.
* Karmaşık Tasarım: Talimat setinin ve adresleme modlarının karmaşıklığı, işlemcinin tasarımını ve uygulamasını zorlaştırabilir ve hata yapma olasılığını artırabilir.
* Yavaşlama: Karmaşık talimatlar ve adresleme modları nedeniyle işlemci yavaşlayabilir.
* Bellek Erişim Döngüleri: Talimatların bellek üzerinde doğrudan işlem yapması, bir talimatın yürütme süresini uzatabilir.
* Düşük Verim: Birçok talimat ve adresleme modu nadiren veya hiç kullanılmaz, bu da kaynakların verimsiz kullanılmasına yol açar.
* Yüksek Kontrol Alanı: Mikroprogramlı kontrol belleği, CPU yongasının önemli bir bölümünü kaplar.

**RISC**

* RISC (Reduced Instruction Set Computer) mimarileri, az sayıda ve basit talimat setine sahiptir.
* Az Sayıda Talimat: RISC, az sayıda ve basit talimat setine sahiptir.
* Az Sayıda Adresleme Modu: Sınırlı sayıda adresleme modu kullanılır.
* Bellek Erişimi: Yalnızca yükleme ve saklama talimatları bellek erişimi gerektirir, diğer tüm işlemler CPU'nun içindeki kaydediciler üzerinde gerçekleştirilir.
* Sabit Uzunlukta Talimatlar: Tüm talimatlar sabit uzunluktadır ve kolayca kodlanabilir.
* Tek Döngüde Talimat Yürütme: Talimatlar genellikle tek bir döngüde yürütülür.
* Mikroprogramsız Kontrol: Kontrol ünitesi mikroprogramsız, doğrudan donanımla gerçekleştirilir (hardwired).
* Hız: Basit ve küçük kontrol ünitesi daha hızlı çalışır. Bellek erişimi yerine kaydedici işlemleri daha hızlıdır ve sabit uzunluktaki talimatlar sayesinde kod çözme süresi kısadır. Ayrıca, boruhattı (pipelining) işlemleri hızlandırır.
* VLSI (Çok Büyük Ölçekli Entegrasyon): Daha küçük kontrol alanı, daha fazla sayıda kaydedici eklenmesine olanak tanır ve performansı artırır.
* Düşük Tasarım Maliyeti ve Yüksek Güvenilirlik: Daha kısa tasarım süresi ve daha basit bir kontrol ünitesi, daha düşük tasarım maliyeti ve daha yüksek güvenilirlik sağlar.
* Yüksek Seviyeli Dil Desteği: Daha az sayıda talimat ve adresleme modu, derleyici yazma yükünü azaltır ve kodun daha verimli olmasını sağlar.
* Yüksek Bellek Tüketimi: Sabit uzunluktaki talimatlar, bellek tüketimini artırabilir.
* Yüksek Kaydedici Kullanımı: Daha fazla kaydediciye ihtiyaç duyulması, kaydedici alanının etkin kullanılmasını gerektirir.
* Gelişmiş Derleyici Gereksinimi: Verimli çalışması için daha gelişmiş ve optimize edilmiş derleyicilere ihtiyaç duyar.
* Yüksek Donanım Maliyeti: Geniş kaydedici setleri ve gelişmiş donanım gereksinimleri, donanım maliyetlerini artırabilir.
* Instructionların ve addressing mode’ların sadeleştirilmesiyle chipte, RISC işlemcilerde CISC’e göre daha çok boşluk kaldı. Bu boşluğa register’lar atılabilir.
* Instruction executionlarının hızlandırılması için pipeline instruction execution için kullanılabilir.
* Farklı instructionların farklı fazları (execution, fetch, decode vs.). Eğer üç instruction aynı anda çalışıyorsa ve bir instruction’ın çalışması üç döngü alıyorsa, CPU ortalama olarak her döngüde bir instruction çalıştırmaktadır.
* Pipelining, işlemcinin farklı fazlarda birden fazla instruction üzerinde paralel olarak çalışmasını sağlar.
* Genel amaçlı kayıtların sayısının fazla olması, bir işlemcinin belleğe yükleme veya saklama işlemleri için erişim sayısını en aza indirmesini sağlar.
* Bellek erişimlerinin kayıt erişimlerinden çok daha yavaş olması nedeniyle bu durum önemli bir hız artışı sağlar.
* Kayıt erişimleri hızlıdır, çünkü sadece CPU üzerindeki veri yolu kullanılır ve her transfer bir saat döngüsünde gerçekleştirilebilir.
* Belleğin dışında işlem yapmak, çok daha yavaş olan bellek (veya sistem) veriyolunu kullanmayı gerektirir.
* Bellek veriyolu üzerinden belleğe okuma veya yazma işlemleri yapmak birçok saat döngüsü alabilir.
* Bellek veriyolu donanımı genellikle işlemciden daha yavaştır.
* Diğer cihazların (örneğin, disk sürücüleri) bellek veriyoluna erişim için rekabet etmesi bile söz konusu olabilir.
* Bu nedenle, sadece bu sebep bile bir RISC işlemcisinin benzer bir CISC işlemcisine göre avantajlı olmasını sağlayabilir, çünkü belleğe sadece talimatları için ve ara sıra bir bellek değerini yüklemek veya saklamak için erişmesi gerekmektedir.

**Pipelining**

* A very important feature of many RISC processors is the ability to execute an instruction each clock cycle. Farklı instructionların farklı fazları (execution, fetch, decode vs.). If we’re running three instructions at once, and it takes an average instruction three cycles to run, the CPU is executing an average of an instruction a clock cycle
* Pipelining is the use of the processor to work on different phases of multiple instructions in parallel.
* Genel amaçlı kayıtların sayısının fazla olması, bir işlemcinin belleğe yükleme veya saklama işlemleri için erişim sayısını en aza indirmesini sağlar.
* Bu durum, bellek erişimlerinin kayıt erişimlerinden çok daha yavaş olması nedeniyle önemli bir hız artışı sağlar. Kayıt erişimleri hızlıdır, çünkü sadece CPU üzerindeki veri yolu kullanılır ve her transfer bir saat döngüsünde gerçekleştirilebilir.
* Belleğin dışında işlem yapmak, çok daha yavaş olan bellek (veya sistem) veriyolunu kullanmayı gerektirir.
* Bellek veriyolu üzerinden belleğe okuma veya yazma işlemleri yapmak birçok saat döngüsü alabilir – Bellek veriyolu donanımı genellikle işlemciden daha yavaştır – Diğer cihazların (örneğin, disk sürücüleri) bellek veriyoluna erişim için rekabet etmesi bile söz konusu olabilir.
* Bu nedenle, sadece bu sebep bile bir RISC işlemcisinin benzer bir CISC işlemcisine göre avantajlı olmasını sağlayabilir, çünkü belleğe sadece – talimatları için ve – ara sıra bir bellek değerini yüklemek veya saklamak için erişmesi gerekmektedir.

**Window Register**

* Window register sistemi, RISC (Reduced Instruction Set Computer) mimarilerinde kullanılan bir tekniktir. RISC işlemcilerde, bellek erişimlerinin sayısını en aza indirmek ve işlemleri hızlandırmak için geniş bir dizi kayıt (register) kullanılır. Window register sistemi, bu kayıtların nasıl yönetileceğini ve kullanılacağını optimize eden bir yöntemdir.
* Window register, özellikle RISC (Reduced Instruction Set Computer) mimarilerinde kullanılan bir tekniktir. Bu teknik, fonksiyon çağrıları ve dönüşleri sırasında bellek erişimlerini minimize etmek için kullanılır. İşlevi:
* Fonksiyon Çağrıları: Fonksiyonlar çağrıldığında, CPU yeni bir register penceresi aktif hale getirir. Bu, fonksiyonun çalışması için gereken parametrelerin ve yerel değişkenlerin saklanacağı yeni bir register setinin kullanılmasını sağlar.
* Fonksiyon Dönüşleri: Fonksiyonlar döndüğünde, önceki register penceresine geri dönülür ve böylece önceki fonksiyonun çalışması kaldığı yerden devam eder.
* Çeşitli Register Türleri:
  + Global Registerlar: Tüm fonksiyonlar tarafından erişilebilir.
  + Yerel (Local) Registerlar: Sadece ilgili fonksiyon tarafından kullanılan değişkenler için ayrılmıştır.
  + Paylaşılan (Shared) Registerlar: Fonksiyonlar arası veri paylaşımı için kullanılır, bu sayede veri kopyalamaya gerek kalmaz.
* Aktif Register Penceresi:
  + Sadece bir register penceresi aktif olabilir ve bu aktif pencere bir işaretçi tarafından belirlenir.
* Pencere Geçişi:
  + Fonksiyon çağrıldığında, işaretçi arttırılır ve yeni bir register penceresi aktif hale gelir.
  + Fonksiyon yeni bir fonksiyon çağırdığında, çağıran fonksiyonun yüksek numaralı registerları, çağrılan fonksiyonun düşük numaralı registerları ile paylaşılır. Bu, parametrelerin ve sonuçların iletilmesini sağlar.
* Performans ve Verimlilik:
  + Bellek Erişimlerini Azaltma: Fonksiyon çağrıları sırasında registerların yığma (push) ve yığından çıkarma (pop) işlemlerinin önüne geçilerek bellek erişimleri minimize edilir.
  + Hız Kazanımı: Bellek erişimlerinden kaçınmak, genel işlem hızını artırır, çünkü bellek erişimleri genellikle register erişimlerinden daha yavaştır.
  + Daha Az Kopyalama: Parametre geçişi ve sonuç dönüşü sırasında veri kopyalama ihtiyacı ortadan kalkar, bu da ek performans kazancı sağlar.
* Üç Sınıf Register:
  + Global Registerlar: Tüm fonksiyonlar tarafından erişilebilir.
  + Yerel Registerlar: Fonksiyonlara özel değişkenler.
  + Paylaşılan Registerlar: Fonksiyonlar arasında veri paylaşımı için kullanılır.