T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MİMARİSİ SİMÜLATÖRÜ TASARIMI

YÜKSEK LİSANS TEZİ

Bilg.Müh. Halit ÖZTEKİN

Enstitü Anabilim Dalı : BİLGİSAYAR VE BİLİŞİM MÜH.

Tez Danışmanı : Doç. Dr. Feyzullah TEMURTAŞ

Ocak 2009

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MİMARİSİ SİMÜLATÖRÜ TASARIMI

YÜKSEK LİSANS TEZİ

Bilg.Müh. Halit ÖZTEKİN

Enstitü Anabilim Dalı : BİLGİSAYAR VE BİLİŞİM MÜH.

Bu tez 08/ 01 /2009 tarihinde aşağıdaki jüri tarafından oybirliği ile kabul edilmiştir.

Doç.Dr Feyzullah TEMURTAŞ Jüri Başkanı	Doç.Dr Ayhan ÖZDEMİR Üye	Yrd.Doç.Dr Ali GÜLBAĞ Üye

TEŞEKKÜR

Tezimin bu aşamaya gelmesinde hiçbir vakit beni unutmayan, bana sağlık ve afiyet veren yüce rabbime sonsuz teşekkürlerimi evvela bir borç bilirim. Tezimin hazırlık aşaması olsun, yapım aşaması olsun bana en büyük desteği gösteren danışman hocam Doç.Dr Feyzullah TEMURTAŞ'a, takıldığım noktalarda yardımlarını esirgemeyen Yrd.Doç.Dr Ali GÜLBAĞ'a, maddi ve manevi desteklerini esirgemeyen çalışma arkadaşlarım ve bölümdeki diğer hocalarıma, tezimde yaşadığım sıkıntıları benimle beraber yaşayan eşime ve beni bugünlere gelmemde vesile olan aileme teşekkür ederim.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

AC : Akümülatör
ALU : Aritmetik mantık birimi
AR : Adres kaydedicisi
CCR : Durum kod kaydedicisi
CISC : Karmaşık komut seti
CPU : Merkezi işlemci birimi
DR : Veri kaydedici
FF : Flip-flop
HSA : Donanım system mimarisi
InREG : Giriş kaydedicisi
IR : Komut kaydedicisi
ISA : Komut seti mimarisi
IX : İndis kaydedici
MAR : Bellek adres kaydedicisi
MBR : Bellek tampon kaydedicisi
MML : MultiMedia Logic
MUX : Veri seçici
OutREG : Çıkış kaydedicisi
PC : Program sayıcısı
RAM : Rastgele erişimli bellek
RISC : Azaltılmış buyruk kümesi
ROM : Yalnızca okunabilir bellek
SP : Yığın kaydedicisi
TR : Geçici kaydedici

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

ÖZET

Anahtar kelimeler: Bilgisayar mimarisi, mikro işlemci tasarımı, simülasyon

Öğrenciler dinleme ve uygulamanın her ikisininde olduğu bir dersi daha iyi öğrenirler. Bilgisayar mühendisliği, elektrik mühendisliği gibi benzer bölümlerde bulunan donanım laboratuarları olmayan küçük okullarda bilgisayar mimarisi dersinde uygulama yapmak zordur. Bu tez, bilgisayar mimarisi ve CPU tasarımı konularında çalışmak isteyen öğrencilere yön gösterici bir çalışmadır. Bu tezde tasarlanan mikro işlemci ticari mikro işlemcilerde ortak olarak kullanılan 59 komutu simüle eder. Altı farklı adresleme moduna sahiptir. Kesme ve giriş-çıkış işlemleri yapabilmektedir. Genel ve özel amaçlı olmak üzere 11 kaydedicisi vardır. 64 KB lık bir belleğe sahip olan bu mikro işlemci, kaydediciler ve bellek arasındaki iletişimin gerçekleşebilmesi için 16 bitlik bir ortak veri yolu tasarlanmıştır. Yazılan bir programın makine koduna dönüştürülebilmesi için Visual Studio .NET'te bir derleyici program yazılmıştır. Sonuç olarak, yazılan bu program vasıtasıyla öğrenci yazdığı bir programın akışını izleyebilmektedir.

COMPUTER ARCHITECTURE SIMULATOR DESIGN

SUMMARY

Key Words: Computer Architecture, microprocessor design, simulation

Students learn better when they both hear and do. In computer architecture courses “doing” can be difficult in small schools without hardware labs hosted by computer engineering, electrical engineering, or similar departments. This thesis is an instructional aid for students studying computer architecture and CPU design. The microprocessor that designed in this thesis simulates 59 instructions that used common in commercial microprocessor. It has six different adressing modes. It has Interrupt and input-output instructions. It has 11 registers that are general and private registers. It is designed 16-bit common data bus to realize communication between this microprocessor has a 64 KB memory, registers, and memory. It is wrote a program that is compiling to machine code in Visual Studio .NET. consequently, can watch the flow of written program by means of this program

BÖLÜM 1. GİRİŞ

Bilgisayar mühendisliğinin temel derslerinden bir tanesi olan Bilgisayar Organizasyonu ve Mimarisi dersi, işlemci, bellek, giriş-çıkış ve ortak yol gibi bir bilgisayarda ortak olarak bulunan temel kavramları inceler[1]. Ayrıca bu derste CISC(Karmaşık komut seti) ve RISC(Azaltılmış Komut seti) işlemcilerinin mimarisi ve organizasyonu, pipeline işlemciler, saklama ve bellek sistemi gibi konuları da ele alır[2]. Bilgisayar Mimarisi ve Organizasyonu alanındaki herhangi bir dersi anlatmak büyük bir problemdir. Problemi tanımlamak gerekirse, öğrencinin teorik bilgisini pratik uygulamayla birleştirerek bilgisayarın programlanması konusunda kullanımını artıracak bir bilgisayar mimarisi ve organizasyonu kavramını sadece kara tahta ile verilen öğretimden uzaklaştırmaktır[3, 4]. Bilgisayar bilimi öğrencileri günümüz bilgisayar sistemlerini etkili bir şekilde kullanabilmek için, kendilerine sunulan kara kutu yaklaşımından ziyade daha detaylı bilgilere ihtiyaç duyarlar. Bellek hiyerarşisi, pipeline tekniği veri yolu hakkında detaylı bilgileri saklamak günümüzde baş döndürücü bir şekilde artış gösteren bilgisayar dünyasında yaşanan gelişmeleri etkili bir şekilde kullanan öğrenciler ortaya çıkarmaz[5]. Bu dersin öğrencilere kazandıracağı en önemli davranışlardan bir tanesi, bir donanım yapısının nasıl tasarlanacağı; tasarlanan bu donanımın çevre birimlerle iletişimini sağlayacak yapıyı oluşturmadır. Bilgisayarda yer alan donanımların işletilmesini sağlayan ve birbirleriyle haberleşmesini sağlayan birim merkezi işlemci birimi(CPU)dir. Bu iletişimi gerçekleştirirken kullanmış olduğu komut setinin tasarlanması ve merkezi işlemci birimi tasarımı bu dersin en önemli konusunu teşkil etmektedir. Yalnız derste gösterilen merkezi işlemci tasarımı ve bunun çevre birimlerle iletişimini sağlayacak yapı öğrenciler tarafından, uygulama yapma eksikliği nedeniyle tam olarak anlaşılamamaktadır. Laboratuar uygulamaları için eldeki elektromekanik deney setleri, sınıfların kalabalık olması, zaman ve mekanın uygun olmayışı, daha fazla temin edilmesi durumunda ekonomik olmaması ve bakım-onarım gerektirdiğinden günümüzde uygun olmamaktadır. Bilgisayar kaynakları ve çoklu-ortam araçlarındaki

gelişimler, pahalı ve hantal sistemler yerine kişi ve kuruluşları sanal öğretim araçlarına yöneltmiştir[6]. Yapılan bu tezde tasarlanan bilgisayar ile bu bahsedilen eksiklerin giderilmesi amaçlanmıştır. Buna benzer uluslar arası literatürde yapılan çalışmalara bakılacak olursa bu alanda birçok yayın yapılmıştır.

Timothy D. Stanley[7] ve arkadaşları tarafından yapılan çalışmaların ilki, "The Essentials of Computer Organization and Architecture"[8] adlı kitapta bahsedilen Java programlama dilinde yazılmış 16 bitlik Marie adlı bilgisayarı Multimedia Logic[3] adlı simülasyon programında gerçekleştirilmesidir. Marie bilgisayar Tablo 1.1'de verilen 13 adet komutu icra edebilen ve bu komutları yerine getirirken direkt ve dolaylı adresleme modunu kullanan bir tasarımdır. Marie bilgisayarda işlem kodu olarak 4 bit ayrılmış, kalan 12 bit ise adres bitleri olarak tahsis edilmiştir.

Kullanabileceği bellek alanı 512 byte dır. Kullanılan genel amaçlı ve işlem yazaçları MAR, MBR, PC, AC, InREG, OutREG ve IR olmak üzere 7 adettir. Bu yapıda kullanılan ortak veri yolu 16 bit olup veri seçicilerle(MUX) oluşturulmuştur.

Timothy D. Stanley[9] ve arkadaşları tarafından yapılan çalışmaların ikincisinde ise, ilk çalışmada olduğu gibi hazır bir yapıyı kullanmak yerine tamamen kendi tasarımları olan iki adet tasarım yapılmıştır. Bunlardan birincisi, sekiz komutu tek saat çevriminde çalıştırabilen 8 bitlik Harvard mimarisi, ikincisi ise 4 komutu üç saat çevriminde yerine getiren 8 bitlik von Neumann mimarisidir.

Von Neumann mimarisinde tasarlanan bilgisayarın yerine getirdiği komutlar: bellekteki bilgiyi akümülatöre kaydeden "Load" komutu, akümülatördeki bilgiyi belleğe kaydeden "Save" komutu, bellekteki veri ile akümülatörün içeriğini toplayan "Add" komutu ve son toplama işleminin sonucu sıfır ise dallanma yapan "Jump" komutudur. 8 bitlik bir program sayıcı(PC), 8 bitlik bir Komut kaydedicisi(IR), 8 bitlik bir akümülatör(AC) ve 8 byte lık bir bellek kullanılmıştır. Komut kaydedicisini yüksek anlamlı iki biti işlem koduna ayrılmış olup kalan 6 bit ise adrese tahsis edilmiştir. Veri yolu 8 bit olup MUX lar ile oluşturulmuştur.

Harvard mimarisinde tasarlanan komutlar von Neuman yapısında kullanılan komutların aynısı olup, birinci tasarımın okunabilirliğini geliştirmek için tasarlanmış bir yapıdır. Bu tasarım bir önceki tasarıma göre iki adet Aritmetik ve mantık birimi kullanır. Bunlardan birincisi program sayıcısını artırma, ikincisi ise aritmetik işlemleri yapmak için kullanılmıştır.

Tablo 1.1. Marie bilgisayar komut seti[2]

İşlem Kodu	Komut	Mikro İşlem Adımları
0000	JnS X	MBR←PC MAR←X M[MAR]←MBR MBR←X AC←1 AC←AC+MBR PC←AC
0001	Load X	MAR←X MBR←M[MAR], AC←MBR
0010	Store X	MAR←X, MBR←AC M[MAR]←MBR
0011	Add X	MAR←X MBR←M[MAR] AC←AC+MBR
0100	Subt X	MAR←X MBR←M[MAR] AC←AC-MBR
0101	Input	AC←InREG
0110	Output	OutREG←AC
0111	Halt	

Tablo 1.1.(Devam) Marie bilgisayar komut seti[2]

İşlem Kodu	Komut	Mikro İşlem Adımları

1000	Skipcond	If IR[11-10]=00 then
		If AC<0 then
		PC←PC+1
		Else If IR[11-10]=01 then
		If AC=0 then
		PC←PC+1
		Else If IR[11-10]=10 then
		If AC>0 then
		PC←PC+1

1001	Jump X	PC← IR[11-0]
1010	Clear	AC←0
1011	AddI X	MAR←X MBR←M[MAR] MAR←MBR MBR←M[MAR] AC←AC+MBR
1100	JumpI X	MAR←X MBR←M[MAR] PC←MBR

Timothy D. Stanley[10] ve arkadaşları tarafından yapılan çalışmaların sonuncusunda, ikinci çalışmada tasarlanan yapı biraz daha geliştirilmiştir. İki adet tasarım yapılmıştır. Bunlardan birincisi, 8 komutu icra edebilen ve bu komutları tek bir saat çevriminde yerine getiren 8 bitlik Harvard mimarisi; ikincisi ise 4 komuta sahip ve her bir komutu 3 saat çevriminde icra edebilen 16 bitlik von Neumann mimarisidir.

Bu mimarilerde gerçekleştirilen komutların yerine getirilme süresi 1 saat çevriminden 12 saat çevrime kadar değişmektedir. Bu tasarımlardan birincisi, 16 bitlik von Neuman tasarımıdır. Bu tasarımda 13 adet komut ve yedi adet

kaydedici(register) kullanılmıştır. Bu kaydediciler; akümülatör(AC), bellek adres kaydedicisi(MAR), bellek tampon kaydedicisi(MBR), program sayıcısı(PC), komut kaydedicisi(IR), giriş kaydedicisi(InREG) ve çıkış kaydedicisinden(OutREG) oluşmaktadır. 13 adet komuttan her biri 16 bit uzunluğuna sahip olup en anlamlı dört biti işlem kodunu(opcode), kalan en düşük anlamlı 12 bitte ise adres yer almaktadır. Bu komutlar; Load, Store, Add, Subtract, Input, Output, Halt, Skip conditional, Jump, Store and Jump, Clear, Add Indirect ve Jump Indirect olmak üzere on üç adettir. Bu tasarımda Multimedia Logic adlı simülasyon programında yer alan iki adet aritmetik lojik birimi(ALU) kullanılmıştır. Bunlardan birincisi, program kaydedicisini artırma, diğeri ise toplama, çıkarma ve mantıksal işlemler için kullanılmıştır. D. Stanley ve arkadaşları tarafından yapılmış olan tasarımlardan ikincisi ise, 16 bitlik Harvard yapısıdır. Bu yapıda kullanılan komutlar; Add, Sub, Multi, BranchNE, BranchLT, Move, Jump, Load, Print, Input, Halt, JumpI, RegI, Write, Div ve Mod olmak üzere 16 adet komut kullanılmaktadır. Bu tasarımda 16 adet onaltı bitlik kaydedici kullanılmaktadır. Bu tasarımda kullanılan komut yapısı bir önceki tasarımda olduğu gibi en anlamlı dört bit işlem koduna ayrılmaktadır.

Timothy D. Stanley ve arkadaşları tarafından yapılan çalışmalar incelendiğinde komut setinin yetersizliği ve adresleme modlarının azlığı ilk görülen eksikliklerdir. Günümüz bilgisayarlarında komut setinin genişliği ve bu komut setinin kullanmış olduğu adresleme modları göz önünde bulundurulacak olursak önemli bir eksikliktir. Diğer bir eksiklik ise kullanılan yapının kara kutu halinde verilmiş olup bu kutu içerisinde meydana gelen olayları kullanıcı görememektedir.

Takao Kawawura[11] ve arkadaşları tarafından SQUEAK[12] yazılım ortamında SIMPLE(Sixteen bit MicroProcessor for Labolatry Experiments) adı verilen bir bilgisayar tasarlamışlardır. Yapılan bu bilgisayarın özelliklerine değinecek olursak; 16 bitlik bir işlemci, 64k kelime bellek alanı, kaydedici adresleme modunu kullanan, her bir komutun bir kelime(word) uzunluğuna sahip bilgisayar 28 komut işleyebilmektedir. Bu komutlar; Load, Store, Immediate Load, şartsız olarak dallanabilen 3 komut, şarta bağlı olarak dallanabilen 8 komut, aritmetik ve lojik işlemleri yapabilen 11 komut ve kontrol amaçlı 3 komuttan oluşmaktadır. Her bir komutun yürütülme aşaması beş aşamada gerçekleştirilmektedir:

a) Komutun çözülme aşaması
b) Kaydedici yükleme aşaması
c) İşlem aşaması
d) Ana bellek erişim aşaması
e) Kaydedici saklama aşaması

Takao Kawawura ve arkadaşları tarafından yapılan çalışma incelendiğinde, Stanley ve arkadaşları tarafından yapılan çalışmada kullanılan komut setinin azlığı burada giderilmiştir. Fakat komut setinde dallanma komutlarının yetersiz oluşu, lojik komutlarının olmayışı, giriş-çıkış komutlarının olmayışı tam bir bilgisayar denebilmesinin şartlarını taşımamaktadır.

John D.Carpinelli[13] ve arkadaşları tarafından yapılan çalışmada dört komutu işleyebilen bir yapı tasarlanmıştır. Bu bilgisayarda kullanılan bellek her biri 8 bit genişliğinde 64 adet gözden oluşur. Bu yüzden merkezi işlemci birimi(CPU)nin 6 adet adres ucu, 8 adet veri ucu vardır. Tasarlanan bu bilgisayarda aritmetik ve lojik işlemlerin tutulduğu genel amaçlı bir adet akümülatör(AC) kullanılmıştır. Yapılan çalışmada kullanılan kaydediciler; 6 bitlik adres kaydedicisi(AR), 6 bitlik program sayıcısı(PC), 2 bitlik komut kaydedicisi(IR) ve 8 bitlik veri kaydedicisi(DR)dir. ADD, AND, JMP ve INC olmak üzere dört adet komut tasarlanmıştır. Komut seti tasarımında kullanılan bir byte'lık bellek kelimesinin en önemli 2 biti işlem kodu için, geri kalan 6 bit ise adres bilgisi için yer ayrılmıştır. Yapılan bu çalışmada kontrol devresi donanım ve mikro programlı olmak üzere iki farklı yöntemle yapılmıştır.

John D. Carpinelli[14] ve arkadaşları tarafından yapılan diğer bir çalışmada Relatively Simple CPU adını verdikleri bir mikro işlemci birimi tasarlanmıştır. Bu merkezi işlemci birimi her bir gözü 8 bitten oluşan 64K lık bir bellek kullanmaktadır. Bu birim 16 bit adres yolu ve 8 bit veri yolu kullanmaktadır. Bu birim toplam 16 adet komutu gerçekleştirebilmektedir. Bu komutları gerçekleştirirken kullanmış olduğu kaydediciler 16 bitlik adres kaydedicisi(AR), 16 bitlik program sayıcısı(PC), 8 bitlik veri kaydedicisi(DR), 8 bitlik komut kaydedicisi(IR) ve 8 bitlik geçici kaydediciden(TR) oluşmaktadır. 3 byte'lık bir komut tasarımı yapılmıştır. İlk byte

işlem kodunu temsil ederken kalan iki byte yani 16 bit adresi teşkil etmektedir. Bu merkezi işlemci birimin yapabildiği komutlar aşağıda Tablo 1.2'de gösterilmiştir.

Yapılan çalışmalarda komut setinin azlığı, adresleme modlarının yetersizliği öne çıkan eksikliklere ilave olarak giriş-çıkış işlemleri gibi bir bilgisayarın çevre birimlerle iletişimi yapmaması ve kesme işlemlerini icra edememesi başlıca göze çarpan eksikliklerdir.

José R. Arias ve Daniel F. García[15] tarafından yapılan makalede Simple CPU adını verdikleri 16 bitlik bir merkezi işlem birimi tasarlanmıştır. Bu birimin 8 adet genel amaçlı kaydedicisi(R0-R7), bir adet program sayıcısı(PC), durum kaydedicisi(SR), ALU geçici giriş kaydedicisi(TMPI), ALU geçici çıkış kaydedicisi(TMPO), bellek veri kaydedicisi(MDR), bellek adres kaydedicisi(MAR) ve komut kaydedicisinden(IR) oluşmaktadır. Simple CPU bilgisayarı derhal adresleme, kaydedici adresleme ve dolaylı bellek adresleme olmak üzere üç adet adresleme kipini kullanmaktadır. Yapılan bu çalışmada da kontrol devresi donanım ve mikro programlı olmak üzere tasarlanmıştır.

Tablo 1.2. "Relatively Simple CPU" için Komut Seti

Komut	İşlem
NOP	İşlem yok
LDAC #adres#	AC=M[#adres#]
STAC #adres#	M[#adres#]=AC
MVAC	R=AC
MOVR	AC=R
JUMP #adres#	Git #adres#
JUMPZ #adres#	Eğer Z=1 ise git #adres#
JPNZ #adres#	Eğer Z=0 ise git #adres#
ADD	AC=AC+R, Z←1
SUB	AC=AC-R, Z←1
INAC	AC=AC+1, Z←1
CLRAC	AC=0, Z←1

Tablo 1.2. (Devam)"Relatively Simple CPU" için Komut Seti

Komut	İşlem
AND	AC=AC∧R, Z←1
OR	AC=ACvR, Z←1
XOR	AC=AC⊕R, Z←1
NOT	AC=AC', Z←1

Clark, Czewoski ve Strazdins tarafından Sparc V9[16] isimli bir simülatör programı geliştirilmiştir. Bu simülasyon programı grafiksel arayüze sahip olmasına karşın kompleks bir yapıya sahip olması bilgisayar mimarisini yeni öğrenen için zor olacaktır.

Wainer ve arkadaşları tarafından geliştirilen Alfa-1 simülatör programı[17] Sparc işlemci ile birlikte çalışır. Bu program bütün bir mimari hakkında bilgi vermesine karşın sınırlı kullanıcı ara yüzü olması bu programın dezavantajıdır.

Moure ve arkadaşları tarafından yapılan çalışma olan Kscalar Simulator[18] sadece mikroişlemcilerin çalışması hakkında bilgi verdiğinden tam bir bilgisayar mimarisi hakkında kullanıcının bilgilendirilmesi konusunda yetersiz kalacaktır.

Holland ve arkadaşları tarafından yapılan çalışma[19] bir önceki yapılan çalışmada[18] değinilen çalışmaya benzerdir. Bu çalışmada tamamlanmamış bir işlemciyi öğrencilerinden tamamlanmasını beklerler. Bu öğrenciler için mikroişlemcilerin çalışması hakkında detaylı bilgi öğrenilmesi konusunda iyi bir fırsattır. Ancak buna karşın kayan noktalı işlem yapmayan tek bir işlemci ile sınırlandırılmıştır.

Yine son iki çalışmada değinilen simülatör programlarına benzer olarak Tao ve arkadaşları tarafından SIMT[20] isimli bir simülatör programı geliştirilmiştir. Bu simülatör programı paylaşımlı bir bellek sisteminin işleyişini kullanıcıya göstermektedir. Yapılan bu son üç çalışmada sadece tek bir amaca yönelik olması belli başlı bilgisayar mimarisi konularında ilerlemek isteyen kullanıcılar için güzel

çalışmalardır. Ancak genel bilgisayar mimarisi hakkında detaylı bilgi öğrenmek isteyenler için ekstra bir çaba sarf etmesine sebep olacaktır.

Yapılan çalışmalara bakıldığında yapılan çalışmaların hepsi bilgisayar bilimlerinin temel derslerinden olan Bilgisayar Mimarisi ve Organizasyonu dersindeki uygulama eksikliğinden bahseden ve bu yönde yapılan çalışmalardır. Ancak bu yönde yapılan çalışmalar komut setinin yeterince kapsamlı olmaması, kullanılan adresleme modlarının yetersiz olması görülen eksiklerin başında gelmektedir. Yapılan hiçbir çalışmada kesme ve yığın işlemlerine değinilmediğinden kullanıcının bu alandaki uygulama eksikliği giderilmemiştir. Bunlara ilave olarak yapılan çalışmalarda, temel lojik elemanlarının içyapısını ayrıntılı olarak kullanıcıya vermektense kapalı bir blok olarak kullanıcının kullanımına sunulmuştur.

Bu eksiklerin giderilmesi bu tezin hazırlanmasında başlıca etken olmuştur. Günümüzde kullanılan ticari mikro işlemcilerin çok sık kullandığı 59 adetten oluşan komut seti oluşturulmuş olup yine çok sık kullanılan adresleme modlarından 6 farklı adresleme modu kullanılmıştır. Kesme ve yığın işlemleri bu çalışmada kullanıcıların kullanımına sunulmuştur. Temel aritmetik işlemlerden olan çarpma ve bölme işlemleri simülasyon programında bulunan hazır eleman kullanmak yerine kullanıcının iç yapısını rahatlıkla inceleyebileceği temel lojik elemanlarından oluşmuştur. Yapılan çalışmada özellikle hazır tool kullanmak yerine, örneğin bir kod çözücü eleman kullanmak yerine bu eleman oluşturulup kullanılmış, böylelikle kullanıcının temel lojik elemanlarının iç yapısını görme imkanına kavuşması sağlanmıştır.

Bölüm 2'de özellikle bu çalışmada kullanılan temel lojik elemanlarının çalışma prensibi, mikro işlemciler hakkında genel bilgiler verilmiş olup Bölüm 4'de anlatılacak olan çalışmayı daha anlamlı kılacaktır.

Bölüm 3'te çalışmanın gerçekleştirildiği Multimedia Logic(MML) simülasyon programı tanıtılacak ve barındırmış olduğu özellikler ayrıntılı olarak ele alınacaktır.

Bölüm 4'de yapılan çalışma detaylı olarak tanıtılacak ve Bölüm 5'te yapılan çalışma üzerinde örnekler çalıştırılacak ve çalışması boyunca sahip olmuş olduğu genel ve özel amaçlı kaydedicilerdeki durum adım adım gösterilecektir.

Bölüm 5'te ise yapılan tez ile ilgili tartışma ve öneriler kısmı yer alacaktır.

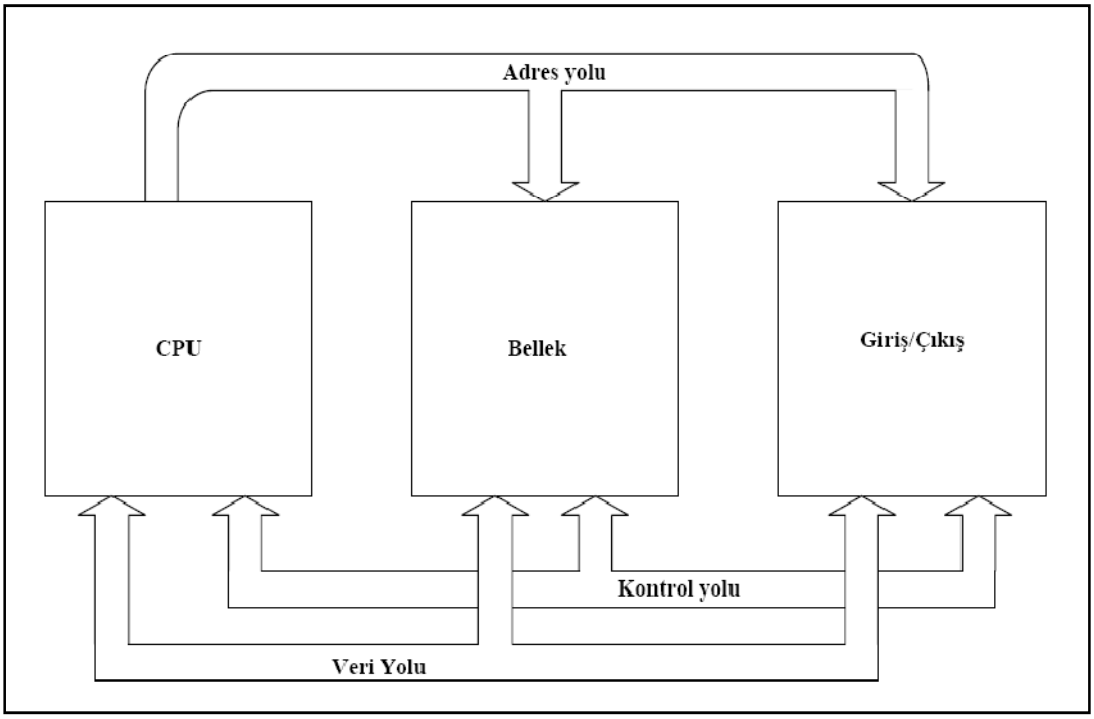
BÖLÜM 2. BİLGİSAYAR MİMARİ TASARIMI VE BİLEŞENLERİ

Bilgisayar tanımı basitçe açıklamak gerekirse, sayısal hale getirilmiş giriş bilgilerini kabul edip sahip olmuş olduğu kayıtlı komut listesi yardımıyla işleyip çıkış bilgisi üreten hızlı bir elektronik hesaplama birimidir[21].

Genel olarak bilgisayar ile iki şekilde ilgilenilir[22]:
a) Yazılım (Software) : Bilgisayarın fiziksel parçalarını işler hale getiren bileşenlerdir.
b) Donanım (Hardware) : Bilgisayarı oluşturan fiziksel parçaların tümüdür.

Her ikisi de birbirinin tamamlayıcısıdır. Birisi olmazsa diğeri de olmaz. Sistem öncelikli olarak tasarlanırken önce sistemi meydana getirecek elemanlar, yani donanım parçaları göz önüne alınır. Daha sonra yazılım bu yapıya bakılarak yazılır. Yazılım, donanımın hangi yönteme göre nasıl çalışacağını gösteren bir sanal uygulamadır. Hangi zamanda hangi elemanın devreye girerek üzerindeki bilgiyi işlemesini sağlamaktadır. Basit bir bilgisayarın ana elemanları Şekil 2.1.'de görülmektedir.

Tüm sayısal bilgisayarlar şekilde gösterilen elemanlara sahiptirler. Bunların dışındaki elemanlar ya da cihazlar seçimliktir. Bilgisayarı oluşturan bu sistemdeki elamanlar; mikroişlemci(CPU), bellek ve giriş/çıkış(G/Ç) birimleridir. Mikroişlemcinin işleyeceği komutlar ve veriler geçici veya kalıcı belleklerde tutulmaktadır. Bilgiyi oluşturan komut ve veriler bellekte karmaşık veya farklı alanlarda tutulabilir. Yazan kişinin karakterini veya seçtiği yolu gösteren çeşitli algoritmalardan meydana gelen program işlemciyi kullanarak verilerin işlenmesini sağlar.



Şekil 2.1. Genel Bilgisayar yapısı

Bilginin işlenmesi sırasında ortaya çıkabilecek ara değerler, en sonunda sonuçlar bellekte bir yerde depolanmak zorundadır. Bütün bu yapılan işlemler bir hesaba dayanmaktadır. Bilgisayarın bilgiyi işlemedeki ana karar vericisi sistemin kalbi sayılan mikroişlemcidir. Mikroişlemci, saklı bir komut dizisini ardışıl olarak yerine getirerek veri kabul edebilen ve bunları işleyebilen sayısal bir elektronik eleman olarak tanımlanabilir. Mikroişlemci temelde mantık kapıları, flip-floplar, sayıcı ve saklayıcılar gibi standart sayısal devrelerden oluşur.

Mikro işlemci tarafından gerçekleştirilen iki temel işlem vardır. Birincisi komutların yorumlanarak doğru bir sırada gerçekleşmesini sağlayan kontrol işlevi, diğeri toplama, çıkarma vb. özel matematik ve mantık işlemlerinin gerçekleştirilmesini sağlayan icra işlevidir. Bilgisayarda çalıştırılan yazılımlar kendi aralarında ikiye ayrılır. Bunlar, programcı tarafından yüksek düzeyde yazılan programlardır ki insanlar tarafından anlaşılabilir düzeydedir ve bu yazılan programların makine tarafından anlaşılmasını sağlayan bağdaştırıcı (interface) yazılımlardır ki işletim sistemi(OS) olarak anılırlar. Mikroişlemci, mantıksal 0 ve 1 esasına göre çalıştığından, verilen komutların da bu esasa dayanması gerekmektedir. Kısaca sayısal bilgisayarların kullandığı doğal dile makine dili denir. Programcı tarafından

yüksek düzeyde yazılan programlar ancak yine insanlar tarafından anlaşılabilir. Bu programların makine tarafından anlaşılabilmesi için derleyici, yorumlayıcı ve assembler gibi aracı programların kullanılması gerekir. Demek ki, yazılım denildiğinde akla, işletim sistemi, üst düzey diller vasıtasıyla yazılan çeşitli uygulama programları gelir. Bu diller;

– Yüksek seviyeli diller
– Orta seviyeli diller
– Düşük seviyeli diller

olmak üzere üç sınıfa ayrılabilir. Bu yüksek, orta, düşük kelimelerinin anlamı donanımın yazılıma ne kadar yakın olduğunu gösterir. Yüksek seviyeli dillerin kontrol sistemlerinde kullanımı zordur. Yüksek seviyeli bir dilde yazılan program derleyici tarafından derlendiğinde bilgisayar bunu düşük seviyeli dile (makina diline) çevirerek anlar. Orta seviyeli dillerin (assembly) kontrol sistemlerinde kullanımı daha uygundur.

Assembly dilini kullanırken donanımı bilmeniz zorunludur. Örneğin Intel 8085 ve Motorola 6800 mikroişlemcilerinin assembly dilleri farklıdır. Çünkü donanımları farklıdır. Orta seviyeli diller kullanılarak program yazma zor ve zahmetli bir iştir. Bunun için makine dilinin komutlar şeklinde verilmesini sağlayan assembly diller geliştirilmiştir. Assembly dilinde program yazmak makine diline göre daha kolay ve anlaşılırdır. Fakat fazla miktarda komut içerir. Bunun için anlama ve kullanımı belli bir zaman alır. Assembly makineye yönelik dillerdir. Programcı kullandığı bilgisayarın donanımını ve adresleme tekniklerini çok iyi bilmelidir. Assembly programları standart değildir. Aynı model olmayan her mikroişlemcinin kendine özgü assembly dili vardır**.** Programcı bu dille makineyle en basit şekilde iletişim kurar. Assembly dilinde yazılan her program bellekte saklanırken veya işlenirken 0 veya 1'ler formuna çevrilmeye gerek duyar. Bu çevirme işi programcı tarafından üretici firmanın databook kitabına bakılarak elle veya bir assembler (Assembly derleyicisi) yardımıyla yapılır. Tek tek komut kodu karşılığına bakılarak ikili komut kodları bulunuyorsa ve eğer program çok uzun veya tekrarlamalı ise, kaynak programı amaç programa çevirmek çok zor ve hata yapma payı yüksek olacaktır. Bu gibi durumlarda iyi bir assembler programı kullanılmalıdır.

Bazen programcılar Assembly dili ile assembleri karıştırmaktadırlar. Assembly dili, konuşma dilinde emir şeklindeki cümleden özenle seçilerek alınmış ve sayısı genelde üç en fazla dört olabilen harflerden meydana gelen ve bir komut anlam ifade eden hatırlatıcıları içerir. Assembly dilinde program yazmak makine dilinde yazmaktan daha kolay ve takibi daha basittir. Fakat bu programın belleğe konulmadan önce makine diline çevrilmesi gereklidir, işte bu işi assembler denilen (bir nevi paket programda denilen) çevirici program yapar. Bu çevirme işlemine kaynak programın amaç programa çevrilmesi denir. Assembly dilinde yazılmış bir programın amaç programa çevirmede en çok kullanılan yöntem elle yapılan işlemdir. Bu yöntemde her satırdaki hatırlatıcıya karşılık gelen kodlar üretici firma tarafından yayınlanan databook'a bakılarak bulunur. Böylece amaç program bulunmuş olur.

Assembly dilinin dezavantajlarını şu şekilde sıralayabiliriz:

1) Assembly dilinde bir program yazmak için üzerinde çalışılan bilgisayarın özellikleri hakkında detaylı bilgi sahibi olunmalıdır. Mesela bunlar, bilgisayar mikroişlemcisinde bulunan kaydediciler ve sayısı, komut kümesi ve adresleme türleri gibi değişik özelliklerdir.

2) Assembly dilinin diğer bir mahsuru elastiki olmamasıdır. Değişik firmalarca üretilen her mikroişlemcinin kendisine has bir programlama dili olmasıdır. Bundan dolayı bir mikroişlemci için yazılan bir assembly dilindeki program diğer bir mikroişlemcide çalışmayabilir.

Assembly Dilinin avantajlarını ise şu şekilde sıralayabiliriz:

1) Assembly dilinde program yazanlar, donanımın çalışmasını çok iyi anlamak ve ona göre iyi programlar geliştirmek zorunda olduklarından kendilerine birçok kazanımlar sağlarlar. Yüksek düzeyli dillerde program yazarken bilgisayar donanımının görünmeyen bazı yanlarına assembly dilinde sahip olunur.

2) Assembly dilinde yazılan programlar yüksek düzeyli dillerle yazılan programlara nazaran daha hızlı ve küçük boyutludur. Assembly dili, program büyüklüğünde ve çalışma hızında ideal optimizasyon sağlar.

Düşük seviyeli diller ise, makine dilleridir. Yine makineye özgü bir dildir. Bu dilde programlama çok zor, hata yapma oranı çok yüksek ve programı kontrol etme imkanı nerede ise yoktur. Assembly ve makine diline uygun uygulamalar şu şekilde sıralanabilir.

– Hesaplamalardan daha çok giriş/çıkış gerektiren uygulamalar
– Gerçek zaman denetimi ve uygulamaları
– Fazla veri işlemesi gerekmeyen uygulamalar
– Hızlılık istenen uygulamalar

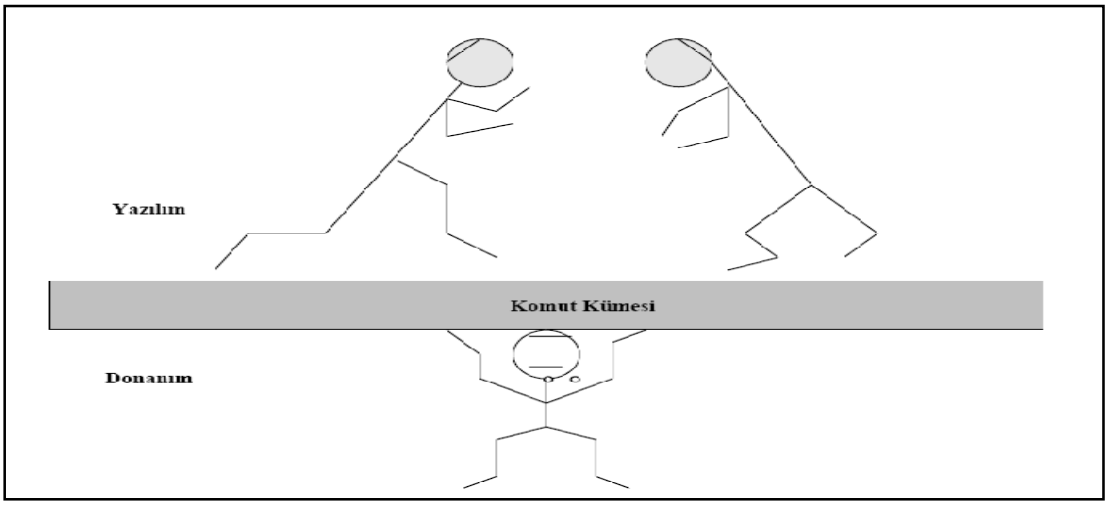
2.1. Mikro İşlemci ve Mikrobilgisayarlar

2.1.1. Bilgisayar mimarisi

Bilgisayar mimarisi, komut kümesinin, donanım elamanlarının ve sistem organizasyonunun dahil olduğu bir bilgisayarın tasarımıdır. Mimari iki farklı yaklaşımla tanımlanmaktadır:

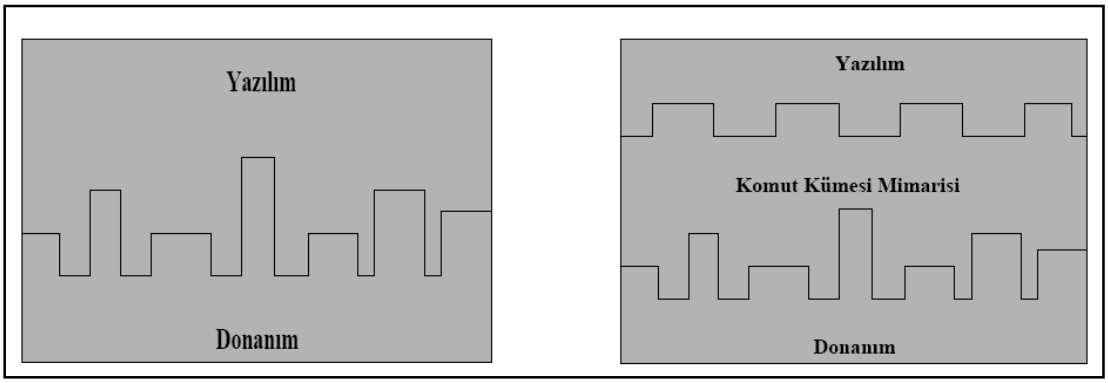
– ISA(Komut kümesi mimarisi)
– HSA(Donanım sistem mimarisi)

ISA; bir bilgisayarın hesaplama karakteristiklerini belirleyen komut kümesinin tasarımıdır. HSA; mikro işlemci, depolama ve Giriş/Çıkış sistemlerinin dâhil olduğu alt sistem ve bunların bağlantı şekilleridir. Komut kümesinin yazılım ve donanımla ilişkisi Şekil 2.2'de görülmektedir.



Şekil 2.2. Komut kümesinin yazılım ve donanımla ilişkisi

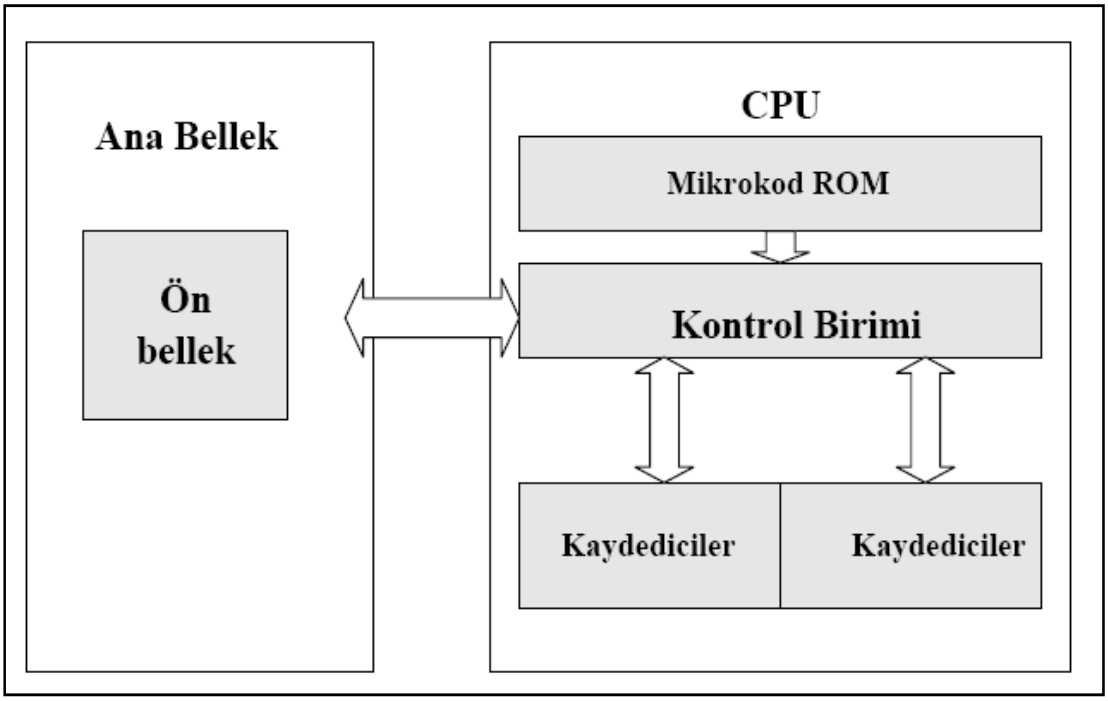
Bilgisayar sistemlerinde bütün mesele, bu iki kavramı yerli yerine oturtmaktır. Mimari bir kavram olarak HSA'nın ne olduğu ve hangi elemanlardan meydana geldiği yukarıda açıklanmıştır. ISA ise, programcının bu elemanlara yön verecek programı yazması durumunda nasıl bir kabul göreceğidir. Farklı şirketler tarafından üretilen farklı bilgisayarların fiyat/performans açısından elbette farklı mimarileri olabilir. Özel bilgisayar sistemleri(günümüzde bir çeşit oyun konsolları)için programcı kodlarını makinenin doğrudan özel donanımına göre yazmaktaydı. Böylece bir makine için yazılan program aynı firma tarafından üretilse bile, ne rekabet ettiği bir makinesinde ne de diğer makinesinde çalışabilmekteydi. Mesela, A makinesi için yazılan bir oyun B makinesinde veya C makinesinde çalışmayacaktır. Programcı tarafından yazılan kodlar donanımı açma anahtarı olarak düşünülebilir (Şekil 2.3.).



Şekil 2.3. Komut Kümesi mimarisinin yazılım ve donanımla ilişkisi

Programsal yaklaşım:

Bilgisayar sistemlerinde bütün mesele sistemi meydana getiren tüm elemanların bir komutla nasıl devreye sokulacağıdır. Ufak tefek farklılıkları olsa da birbirine benzer yapıdaki bilgisayarlar için farklı programlar yazmak oldukça maliyetli olduğundan, programcının yazdığı komutların her bilgisayar tarafından algılanarak yürütülmesi esas hedeftir. Ortaya atılan ilk çözüm mikrokod yaklaşımı, daha sonraları iki standarttan biri olmuştur. Donanımı devreye sokacak öz bilgilerin yani komut kümesinin yer aldığı bu yere (bölgeye) mikrokod motoru denilmektedir**.**(Şekil 2.4.).



Şekil 2.4. Bir mikrokod ROM'un sistemdeki yeri

Burası, mikro işlemci içinde mikro işlemci olarak da ifade edilebilir. Programcının yazdığı kodları işlemcinin daha çabuk anlayabileceği veya çalıştırabileceği küçük mikrokodlara dönüştüren bu mikrokod motoru, işlemci ROM(Yalnız okunabilir bellek) bellek vasıtasıyla yerleştirilmiştir. Mikroprogram ve icra birimi tarafından meydana gelen mikrokod ROM'un görevi, özel komutların bir dizi kontrol sinyallerine çevirerek sistem elemanlarının denetlenmesini sağlar. Aynı zamanda, mikrokod CISC(Karmaşık komut seti) tipi işlemcilerdeki temel işlevi, alt düzey komut kümesiyle programcının çalıştığı üst düzey komutlar arasında soyutlama düzeyi oluşturmaktır.

Mikroişlemci üreticileri, sistem tasarımında iki yönlü düşünmek zorundadırlar. Birincisi, mimariyi meydana getiren elemanların işlevleri, ikincisi bu elemanların nasıl devreye sokulacağıdır. Elemanları devreye sokmak için program yazmak gerekecektir. Bu işin bir yanı; diğer yanı ise donanımdır. Donanımla tasarım mühendisleri ilgilenir. Fakat programcı öyle bir program yazmalı ki, sistem tarafından algılanarak doğru zamanda doğru eleman devreye sokulabilsin. Donanım mimarisini programcıya aktaracak en iyi yol ona kullanabileceği komut kümesini hazır vermektir. Bilgisayar sisteminin donanımsal tüm özelliklerini içeren sisteme

komut kümesi mimarisi denildiğine göre, programcı bu kümeye bakarak veya bu kümeyi kullanabilen derleyicileri kullanarak hiçbir endişeye gerek duymaz. Programcının yazdığı bir komut işletildiğinde, mikrokod ROM bu komutu okur ve sonra o komuta karşılık gelen uygun mikrokodları yükler ve çalıştır.

Donanımsal Yaklaşım:

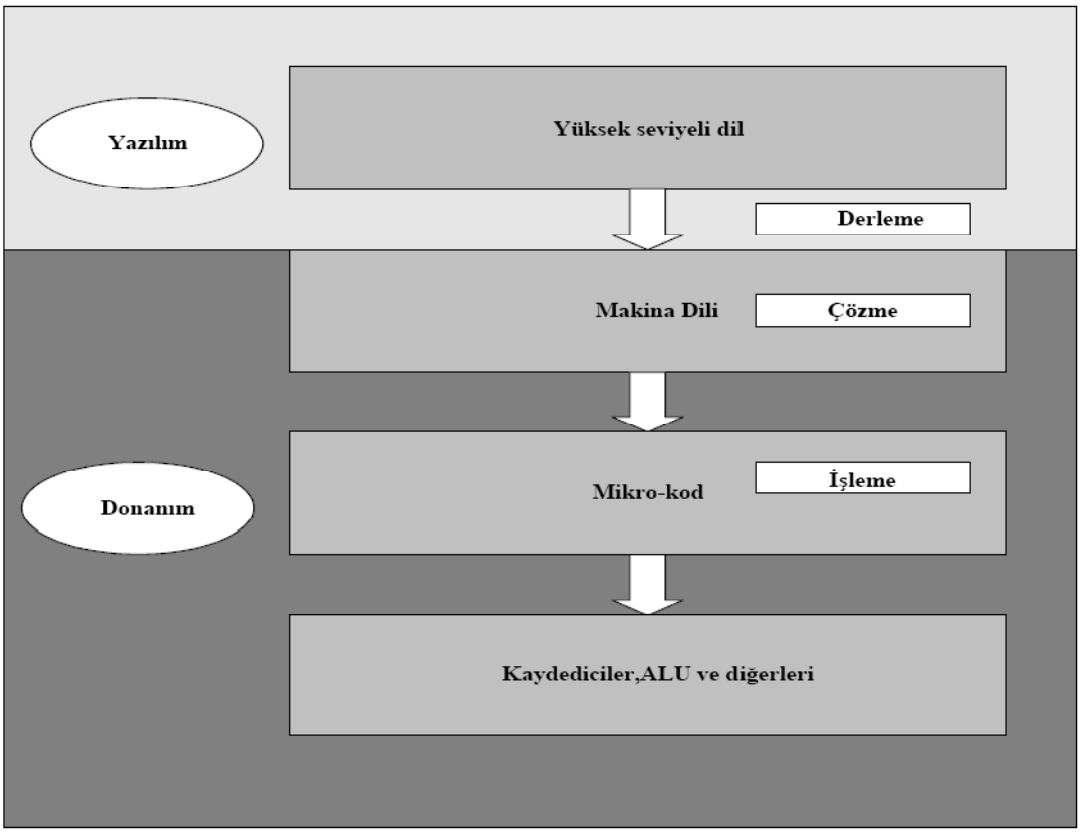
Mikrokod kullanılarak ISA sisteminin yürütülmesinin başlıca sakıncası başlangıçta komutların doğrudan çalıştıran sisteme göre yavaş olmasıdır. Mikrokod, ISA tasarımcılarına programcının ara sıra kullandığı her çeşit komutların komut kümesine eklenmesini ister. Daha çok komut demek daha fazla mikrokod, çekirdek büyüklüğü ve güç demektir. ISA mimarisinin yaşanan aksaklıklarından dolayı daha sonraları, komutların doğrudan donanım elemanları tarafından yorumlanarak sistemin denetlendiği diğer bir mimari yaklaşımda donanımsal çalışma modelidir. Komutların anlaşılır standart bir boyuta getirilerek çalışıldığı sisteme RISC(Azaltılmış komut seti) modeli denilmektedir. Yani komutların donanımsal çalışma modeline sahip RISC tipi bilgisayarlarda, komut kümesindeki komutların sayısı azaltılmış ve her bir özel komutun boyutu düşürülmüştür.

2.1.2. Mikro işlemci mimarisi

2.1.2.1. CISC

Bu mimari, programlanması kolay ve etkin bellek kullanımını sağlayan tasarım felsefesinin bir ürünüdür. Her ne kadar performans düşüklüğüne sebep olsa ve işlemciyi daha karmaşık hale getirse de yazılımı basitleştirmektedir. CISC mimarisinin karakteristik iki özelliğinden birisi, değişken uzunluktaki komutlar, diğeri ise karmaşık komutlardır. Değişken ve karmaşık uzunluktaki komutlar bellek tasarrufu sağlar. Karmaşık komutlar iki ya da daha fazla komutu tek bir komut haline getirdikleri için hem bellekten hem de programda yer alması gereken komut sayısından tasarruf sağlar. Karmaşık komut karmaşık mimariyi de beraberinde getirir. Mimarideki karmaşıklığın artması, işlemci performansında istenmeyen

durumların ortaya çıkmasına neden olur. Ancak programların yüklenmesinde ve çalıştırılmasındaki düşük bellek kullanımı bu sorunu ortadan kaldırabilir.



Şekil 2.5. CISC tabanlı bir işlemcinin çalışma biçimi

CISC mimarisi çok kademeli işleme modeline dayanmaktadır. İlk kademe yüksek düzeyli dilin yazıldığı yerdir. Sonraki kademeyi makine dili oluşturur ki yüksek düzeyli dilin derlenmesi sonucu bir dizi komutlar makine diline çevrilir. Bir sonraki kademede makine diline çevrilen komutların kodları çözülerek, mikroişlemcinin donanım birimlerini kontrol edebilen en basit işlenebilir kodlara (mikrokod) dönüştürülür. En alt kademede ise işlenebilir kodları alan donanım aracılığıyla gerekli görevler yerine getirilir.

CISC Tasarımının Özellikleri:

80'li yıllara kadar çıkarılan çipler kendine has tasarım yollarını takip ettiler. Bunlardan çoğu "CISC tasarım kararları" denilen kurallara uydular. Bu çiplerin

hepsinin benzer komut kümeleri ve donanım mimarileri vardır. Komut kümeleri, assembly dili programcılarının rahatlığı için tasarlanırlar ve donanım tasarımları oldukça karışıktır.

CISC Mimarisinin Üstünlükleri:

CISC makineler ilk gelişim sıralarında bilgisayar performansını yükseltmek için mevcut teknolojileri kullandılar.

– Mikroprogramlama, assembly dilinin yürütülmesi kadar kolaydır ve sistemdeki kontrol biriminden daha ucuzdur.

– Yeni komutlar ve mikrokod ROM'a eklemenin kolaylığı tasarımcılara CISC makinelerini geriye doğru uyumlu yapmalarına izin verir. Yeni bir bilgisayar aynı programları ilk bilgisayarlar gibi çalıştırabilir çünkü yeni bilgisayar önceki bilgisayarların komut kümelerini de içerecektir.

– Her bir komut daha yetenekli olmaya başladığından, verilen bir görevi yürütmek için daha az komut kullanılır. Bu, nispeten yavaş ana belleğin daha etkili kullanımını sağlar.

– Mikroprogram komut kümeleri, yüksek seviyeli dillerin yapılarına benzer biçimde yazılabildiğinden, derleyici karmaşık olmak zorunda değildir.

CISC Mimarisin Mahsurları:

– İşlemci ailesinin ilk kuşakları genelde her yeni versiyon tarafından kabullenilmiştir. Böylece komut kodu ve çip donanımı bilgisayarların her kuşağıyla birlikte daha karmaşık hale gelmiştir.

– Mümkün olduğu kadar çok komut, mümkün olan en az zaman kaybıyla belleğe depolanabiliyor ve komutlar neredeyse her uzunlukta olabiliyor. Bunun anlamı farklı komutlar farklı miktarda saat çevrimi tutacaktır(makinenin performansını düşürecektir).

– Çoğu özel güçlü komutlar geçerliliklerini doğrulamak için yeteri kadar sık kullanılmıyor. Tipik bir programda mevcut komutların yaklaşık %20'sini kullanıyor.

– Komutlar genellikle bayrak (durum) kodunu komuta bir yan etki olarak kurar. Bu ise ek çevrimler yani bekleme demektir. Aynı zamanda, sıradaki komutlar işlem

yapmadan önceden bayrak bitlerinin mevcut durumunu bilmek durumundadır. Bu da yine ek çevrim demektir. Bayrakları kurmak zaman gibi, programlar takip eden komutun bayrağın durumunu değiştirmeden önce bayrak bitlerini incelemek zorundadır**.**

2.1.2.2. RISC

RICS Mimarisi, CISC mimarili işlemcilerin kötü yanlarını piyasanın tepkisi ve ona bir alternatif olarak, işlemci mimari tasarımlarında söz sahibi olan IBM, Apple ve Motorola gibi firmalarca sistematik bir şekilde geliştirilmiştir. (CISC, piyasa şartları doğrultusunda şekillenen ve kendiliğinden oluşan bir sistemdir.) RISC felsefesinin taraftarları, bilgisayar mimarisinin tam anlamıyla bir elden geçirmeye ihtiyacı olduğunu ve neredeyse bütün geleneksel bilgisayarların mimari bakımdan birtakım eksikliklere sahip olduğu ve eskidiğini düşünüyorlardı. Bilgisayarların gittikçe daha karmaşık hale getirildiği ve hepsinin bir kenara bırakılıp en baştan yeniden başlamak gerektiği fikrindeydiler.

RISC Mimarisinin Özellikleri:

RISC mimarisi aynı anda birden çok komutun birden fazla birimde işlendiği iş-hattı (pipelining) tekniği ve süperskalar yapılarının kullanımıyla yüksek bir performans sağlamıştır. Daha önce değinildiği gibi, bu tasarım tekniği yüksek bellek ve gelişmiş derleme teknolojisi gerektirmektedir. Bu mimari, küçültülen komut kümesi ve azaltılan adresleme modları sayısı yanında aşağıdaki özelliklere sahiptir:

- Bir çevrimlik zamanda bir komut işleyebilme
- Aynı uzunluk ve sabit formatta komut kümesine sahip olma
- Ana belleğe sadece “load” ve “store” komutlarıyla erişim; operasyonların sadece kaydedici üzerinde yapılması
- Bütün icra birimlerinin mikrokod kullanılmadan donanımsal çalışması
- Yüksek seviyeli dilleri destekleme
- Çok sayıda kaydediciye sahip olması

RISC Mimarisinin Üstünlükleri:

RISC tasarımı olan bir işlemciyi kullanmak, karşılaştırılabilir bir CISC tasarımını kullanmaya göre pek çok avantaj sağlar**.**

Hız: Azaltılmış komut kümesi, kanal ve süperskalar tasarıma izin verdiğinden RISC işlemciler genellikle karşılaştırılabilir yani yarı iletken teknolojisi ve aynı saat oranları kullanılan CISC işlemcilerinin performansının 2 veya 4 katı daha yüksek performans gösterirler.

Basit Donanım: RISC işlemcinin komut kümesi çok basit olduğundan çok az çip uzayı kullanırlar. Ekstra fonksiyonlar, bellek kontrol birimleri veya kayan noktalı aritmetik birimleri de aynı çip üzerine yerleştirilir.

Kısa Tasarım Zamanı: RICS işlemciler CISC işlemcilere göre daha basit olduğundan daha çabuk tasarlanabilirler ve diğer teknolojik gelişmelerin avantajlarını CISC tasarımlarına göre daha çabuk kabul edebilirler.

RICS Mimarisinin Sınırlılıkları:

CISC tasarım stratejisinden RICS tasarım stratejisine yapılan geçiş kendi problemlerini de beraberinde getirmiştir. Donanım mühendisleri kodları CISC işlemcisinden RISC işlemcisine aktarırken anahtar işlemleri göz önünde bulundurmak zorundadır.

CISC ve RISC Tabanlı İşlemcilerin Karşılaştırılması:

CISC ve RISC tabanlı işlemcilerin karşılaştırılmasında iki önemli faktör farklılıklarını ortaya çıkarmada yeterlidir.

Hız: Genelde RISC çipleri kanal tekniği kullanarak eşit uzunlukta segmentlere bölünmüş komutları çalıştırmaktadır. Kanal tekniği komutları kademeli olarak işler ki bu RISC'in bilgi işlemini CISC'den daha hızlı yapmasını sağlar RISC işlemcisinde tüm komutlar 1 birim uzunlukta olup kanal tekniği ile işlenmektedir. Bu teknikte bazıları hariç komutlar, her bir basamağında aynı işlemin uygulandığı

birimlerden geçerler. Kanal teknolojisini açıklamak için herhangi bir komutun işlenmesindeki adımlar ele alınırsa:

Komut kodu ve işlenecek veriler dahil bütün bilgilerin mikro işlemcideki kaydedicilerde olduğu düşünülürse, birinci adımda yapılacak işin kaydedicide bulunan komut kodu çözülür, ikinci adımda üzerinde çalışılacak veri (işlenen) kaydediciden alınıp getirilir, üçüncü adımda veri, komuta göre ALU(Aritmetik ve Mantık Birimi)'da işleme tabi tutulur ve dördüncü adımda da sonuç kaydediciye yazılacaktır. Böylece bir komutun işlemesi için her bir basamak bir saat çevrimi gerektirirse, dört çevrimle (adımda) gerçekleşmiş olmakta ve bir adım bitmeden diğeri başlayamamaktadır. Kanal tekniği ile çalışan işlemcilerde birinci adımda komut kodu çözülür, ikinci adımda birinci komutun üzerinde çalışacağı veri (işlenen) kaydediciden alınırken, sıradaki ikinci işlenecek olan komutun kodu çözülür. Üçüncü adımda ilk komutun görevi ALU'da yerine getirilirken, ikinci komutun işleyeceği işlenen alınıp getirilir. Bu anda sıradaki üçüncü komutun kodu çözülür ve işlem böylece devam eder. Kanal (Pipeline) tekniğinde çevrim zamanın düşmesi için komut kodlarının hızlı çözülmesi gereklidir. RISC mimarisinde tüm komutlar 1 birim uzunlukta oldukları için komut kodunu çözme işlemi kolaylaşır. Sistemde kullanılan kaydedicilerin simetrik bir yapıda olması, derleme işlemini kolaylaştırmaktadır. RISC işlemcilerde belleğe yalnız yükle ve depola komutlarıyla ulaşılır. Bazı eski CISC mimarisinde de olmasına rağmen RISC mimarisinin sabit uzunluktaki basit komutlarla çalışması pipeline sistemini daha iyi kullanmasına sebep olmaktadır. Bu yüzden hesaplama oranlarının birinci öncelik arz ettiği yerlerde iş-istasyonları ve dağıtıcılarda çok tercih edilmektedir.

Transistör sayısı: CISC mimarisinde kullanılan transistor sayısı RISC'e nazaran daha fazladır. Transistör sayısının bir yerde çok olması fazla yerleşim alanı ve ayrıca fazla ısı demektir. Bundan dolayı da fazla ısı üretimi soğutma olayını gündeme getirmektedir. CISC tabanlı Pentium işlemcilerde karışık ısı dağıtıcısı veya soğutma fanları kullanılmaktadır.

RISC mimarisindeki önemli üstünlüklere karşı bazı mahzurları ortaya çıkmaktadır. RISC mimarisi, CISC'in güçlü komutlarından yoksundur ve aynı işlemi yapmak için

daha fazla komut işlenmesini gerektirir. Bundan dolayı da RISC'in bant genişliği artar. Bu sistemde güçlü komutların yokluğu ikinci bir yardımcı işlemciyle ya da işlemci içinde oluşturulacak ayrı bir pipeline bölümüyle giderilebilir. Komut ön-belleğinin kullanılması yüksek komut alıp getirme işlemini azaltmaktadır. RISC mimarisi diğerine nazaran daha kompleks yazılımlara ihtiyaç duyar. RISC ve CISC mimarilerinin karşılaştırılması özet olarak aşağıda tablo olarak verilmiştir.

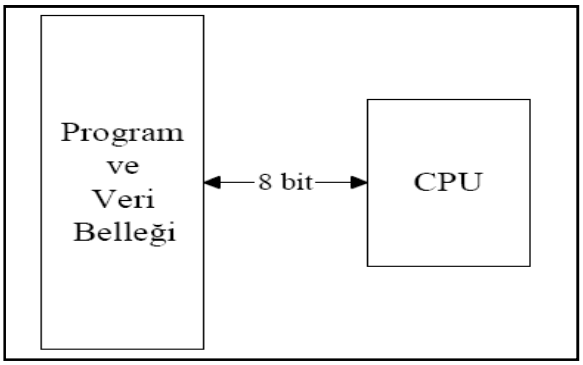
Tablo 2.1 RISC ve CISC mimarilerinin karşılaştırılması

RISC (Hard-wired Control Unit)	**CISC (Microprogrammed Control Unit)**
Hızlı	Nispeten yavaş
Ucuz	Pahalı
Yeniden dizayn zor	Esnek
Daha az komut (instruction)	Daha fazla komut (instruction)
Daha fazla saklayıcı bellek (register)	Daha az saklayıcı bellek (register)

2.1.3. Mikrobilgisayar mimarisi

2.1.3.1. Princeton (Von Neumann) yapısı

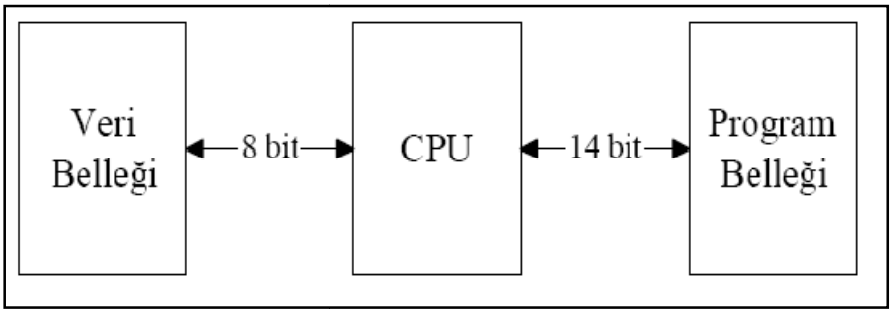
– Program ve veri için ortak hafıza kullanır.
– Yonga tasarımı basittir.
– Bir komutun yürütülmesi çok sayıda saykıl (cp) gerektirebilir.



Şekil 2.6. Von-Neumann yapısı

2.1.3.2. Harvard yapısı

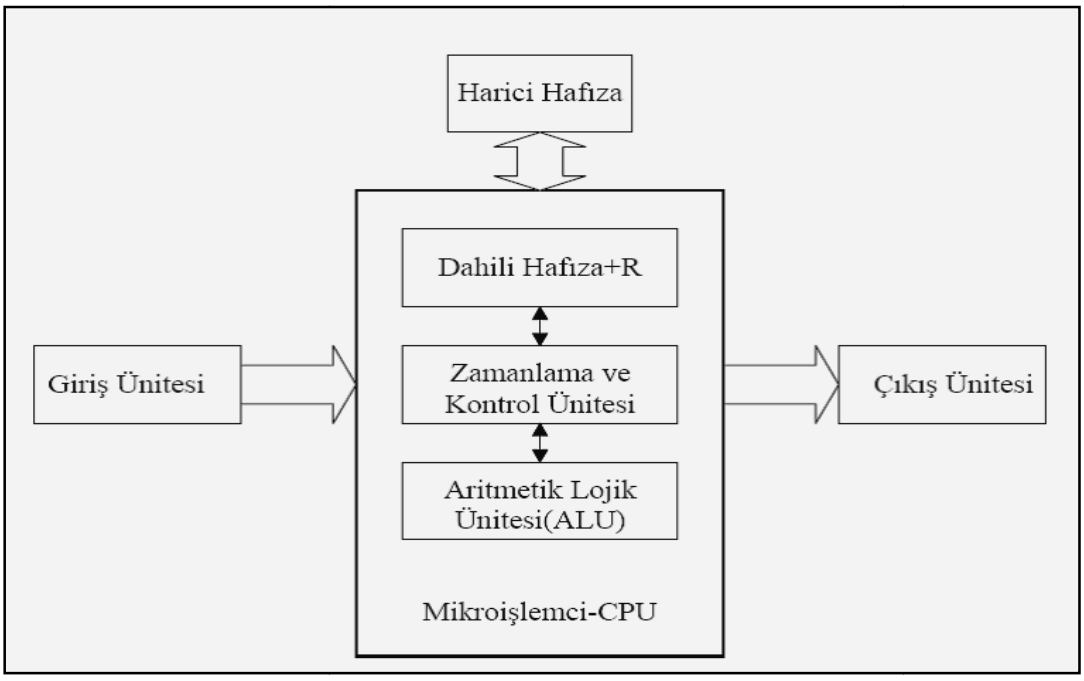
– Program ve veri için ayrı ayrı hafıza kullanılır.
– Komutlar bir saykıl da (cp) yürütülür.
– Döngüler ve gecikmeler daha kolay ayarlanır.



Şekil 2.7. Harvard yapısı

2.1.4. Mikro işlemci ve mikrobilgisayar yapısı

Bir mikroişlemci; dahili hafıza, kaydediciler, kontrol ünitesi ve aritmetik lojik biriminden oluşmaktadır. Bir mikrobilgisayar ise; mikroişlemci, harici hafıza, giriş ve çıkış ünitelerinden oluşur.



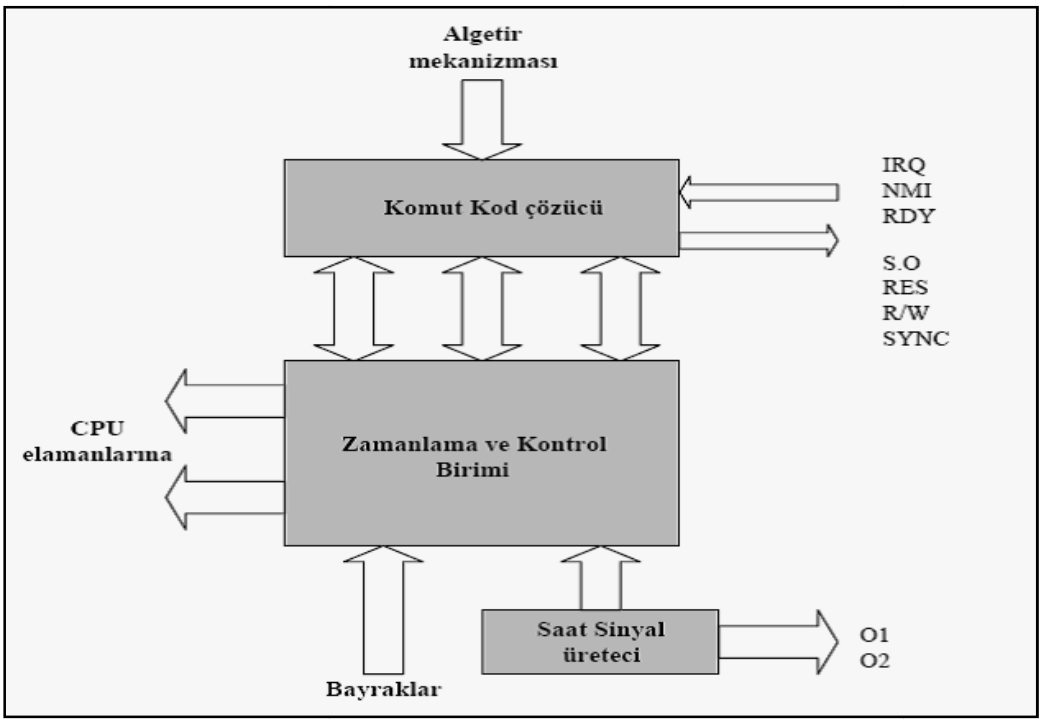
Şekil 2.8. Bir Mikrobilgisayarın yapısı

2.1.4.1. Mikrobilgisayarın elemanları

a) Giriş Üniteleri: Klavye (tuş takımı), programlar, sensörler.v.s.
b) Giriş/Çıkış Üniteleri: Hem giriş hem de çıkış olarak kullanılan ünitelerdir. Modemler, seri ve paralel portlar bunlara örnektir.
c) Harici (program) Hafızası: Programların yazıldığı yerdir. Küçük bilgisayarlarda (mikrobilgisayar) bu ünite, EPROM, ROM, PROM veya EEPROM dur. Büyük bilgisayarlarda hard disk ve disk, optik diskler (CD-ROM) kullanılmaktadır. Optik diskler hard disk ve disklere göre daha hızlıdır.

2.1.4.2. Mikroişlemcinin elemanları

1-) Zamanlama ve Kontrol Ünitesi: Bu kısım sistemin tüm işleyişinden ve işlemin zamanında yapılmasından sorumludur. Zamanlama ve kontrol birimi, bellekte program bölümünde bulunan komut kodunun alınıp getirilmesi, kodun çözülmesi, ALU tarafından işlenmesi ve sonucun alınıp belleğe geri konulması için gerekli olan kontrol sinyallerini üretir. Bilgisayar sisteminde bulunan dahili ve harici bütün elemanlar bu kontrol sinyalleri ile denetlenir.



Şekil 2.9. Zamanlama ve kontrol birimin giriş ve çıkış sinyalleri

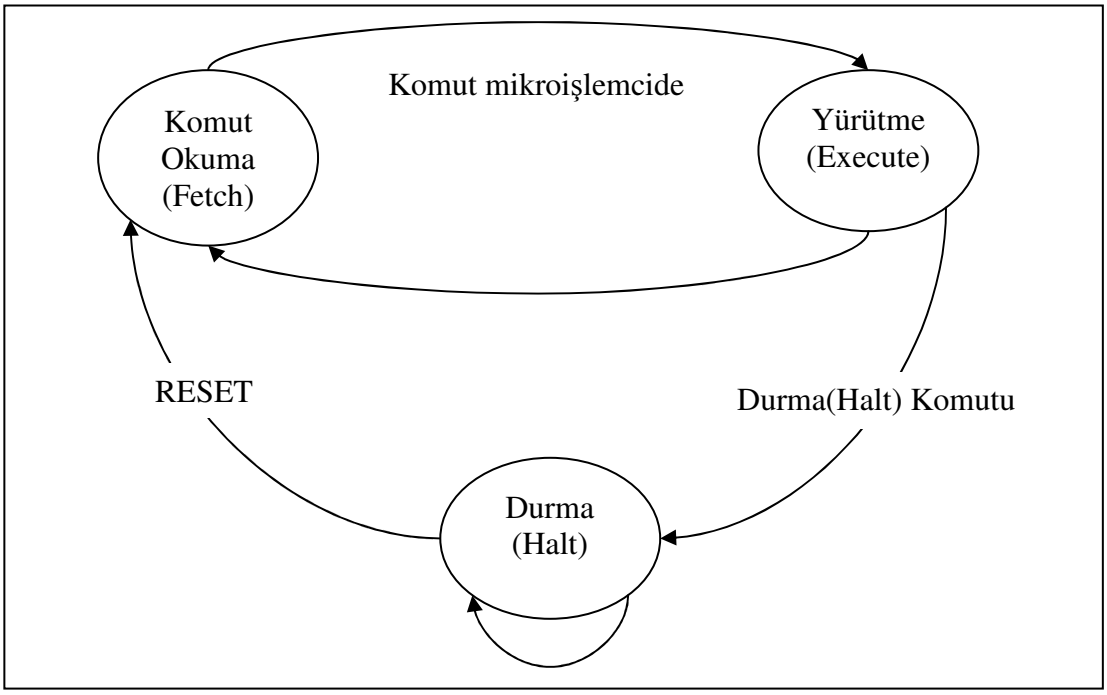
Basit bir mikroişlemcide bu bölüm üç değişik işlevi yerine getirir:

a) Zamanlama kontrolü: İşlemci harici saat sinyali üreten bir birimden giriş alan iç-saat devresine sahiptir. Bu sinyal alınarak talebe göre zamanlama sinyallerine çevrilerek sisteme dağıtılır.
b) Komut kod çözücü: Bu devre komut kaydedicisinde (IR) tutulan komutları yorumlar ve ALU'ya kaydedicilerle çalışması için uygun sinyaller gönderir.(kastedilen zamanlama ve kesme sinyalleridir)
c) Kesme mantık birimi: Bu birimde diğer kontrol elemanlarına benzer. Gerekli durumlarda kesme sinyallerini alarak işlemciyi uyarır.

2) Dahili Hafıza: Geçici bilgilerin saklandığı yerdir. Çok hızlı çalışırlar.
3) Kaydedici(Register): Temel depolama elemanları olan FF(Flip-flop)'lardan oluşurlar.
4) Aritmetik Lojik Ünitesi: Toplama, çıkarma, artırma, azaltma, elde bitiyle toplama, çarpma, bölme ve AND, OR, XOR, NOT, shift (kaydırıcı) mantık birimlerinden oluşur. Bu işlemler assembly dilinin komutları ile yapılır.

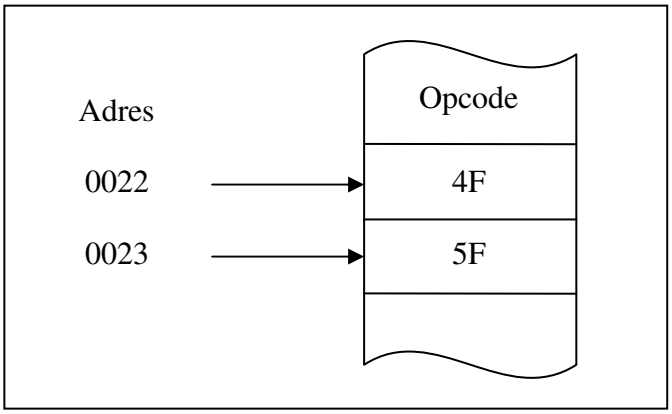
2.2. Mikro İşlemcinin Çalışması

Bir mikroişlemcinin çalışmasında, kontrol birimi tarafından yerine getirilen ve Şekil 2.10'da gösterilen temel iki işlem vardır. Komut okuma (fetch) ve komut yürütme(execute). Komut okuma, mikroişlemcinin hafızadan bir işlem kodu (opcode) alıp komut saklayıcısına(Instruction Register –IR) getirme işlemine denir. Komut saklayıcısına gelen komut ile hangi işlemin yapılacağı komut kod çözücüsü tarafından belirlenir. Gereken sinyaller kontrol birimi tarafından üretilir. Eğer komut ile belirlenen işlem için, bazı işlem verisine (operand) gerek var ise, bu veriler hafızadan okunur. Son olarak komutun yürütülmesi gerçekleştirilir. Bir komutun yürütülmesi bittikten sonra, benzeri şekilde; tekrar komut okuma ve yürütme işlemleri, bir durma (halt) komutu yürütülünceye kadar yapılır. Mikroişlemcinin çalışmasının durduran bu komut ile işlemci bir üçüncü duruma girer ve bu durumdan çıkabilmesi için bir donanım sıfırlaması (reset) gerekir.



Şekil 2.10. Bir mikro işlemcideki komut okuma ve yürütme çevrimleri

Bir mikroişlemcinin Şekil 2.11'de yer alan bellekte bulunan örnek bir programın işletmesini adım adım inceleyelim:



Şekil 2.11. Örnek bir program

1. Program sayacı o anda çalışan komutun adresini üzerine alır. PC =[0022]
2. Komut kayıtçısı, o anda çalışan komutu üzerinde bulundurur. IR= 4F
3. Kontrol ünitesi bu komuta göre uygun elektronik sinyalleri uygun yere göndererek komutu çalıştırır. 4F için A akümülatörünü sıfırlayıcı işaret gönderir.
4. Daha sonra PC bir sonraki adrese geçer ve aynı işlemler tekrarlanır.

2.3. Kullanılan Sayısal Elemanlar

2.3.1. Kod çözücüler

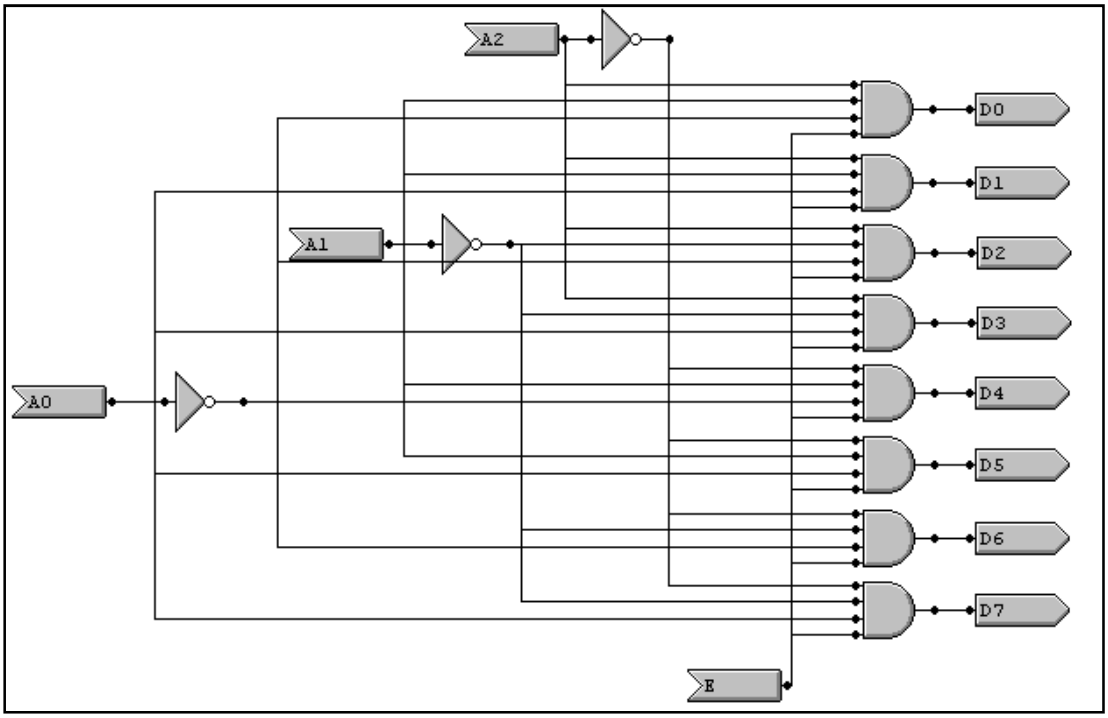
Sayısal bilgisayarlarda, bilgilerin farklı değerli ikili kodlarla belirlenmiştir. Çözücü devre ile birlikte kodlu ikili girişlerden 2^n lik farklı çıkışlar elde edilir. Eğer n bit kod bilgisi, bit kombinezonu ile kullanılmasaydı çıkış 2^n den daha az olabilirdi. Çözücülerin amaçları n girişli değişkenler ile 2^n lik ikili birleşimler meydana getirmektir[23].

$m \leq 2^n$ olduğunda kod çözücüye n x m hatlı kod çözücü denir. Bu kod çözücünün amacı n farklı girişten 2^n ikili çıkış kombinezonu elde etmektir. Burada n giriş ve m çıkış vardır.

Şekil 2.12 de 3x8 hatlı kod çözücünün mantık şeması gösterilmektedir. A_0, A_1 ve A_2 olan 3 veri girişinden sekiz çıkış kodlanmaktadır. 3 tane tersleyen, girişlerin tümleyenini almaktadır. 8 ve kapısının her biri ikili kombinezonlardan birini üretir. Bu kod çözücünün en temel uygulaması ikiliden sekizliye çevirmedir. Giriş değişkenleri ikilik değer alır ve çıkış olarak sekizlik sayı sisteminin 8 rakamından biri alınır. Bununla birlikte 3x8 lik kod çözücü herhangi bir üç bit koddan 8 farklı çıkış elde etmek için kullanılır.

Ticari olarak üretilen kod çözücüler devrenin çalışmasını denetleyen bir veya daha fazla izin girişine sahiptir. Şekil 2.12 deki devrede bir izin girişi vardır. E=1 olduğunda kod çözücü çalışır, E=0 olduğunda çalışmaz.

Kod çözücünün çalışması Tablo 2.2 deki doğruluk tablosu kullanılarak açıklanabilir. İzin girişi sıfır olduğunda(E=0), üç veri girişinin değerine bakılmaksızın tüm çıkışlar sıfır olur. E=1 olduğunda kod çözücü normal kipte çalışır. Her bir giriş için çıkışlardan sadece bir tanesi 1, diğerleri 0 dır.



Şekil 2.12. 3x8 hatlı kod çözücü

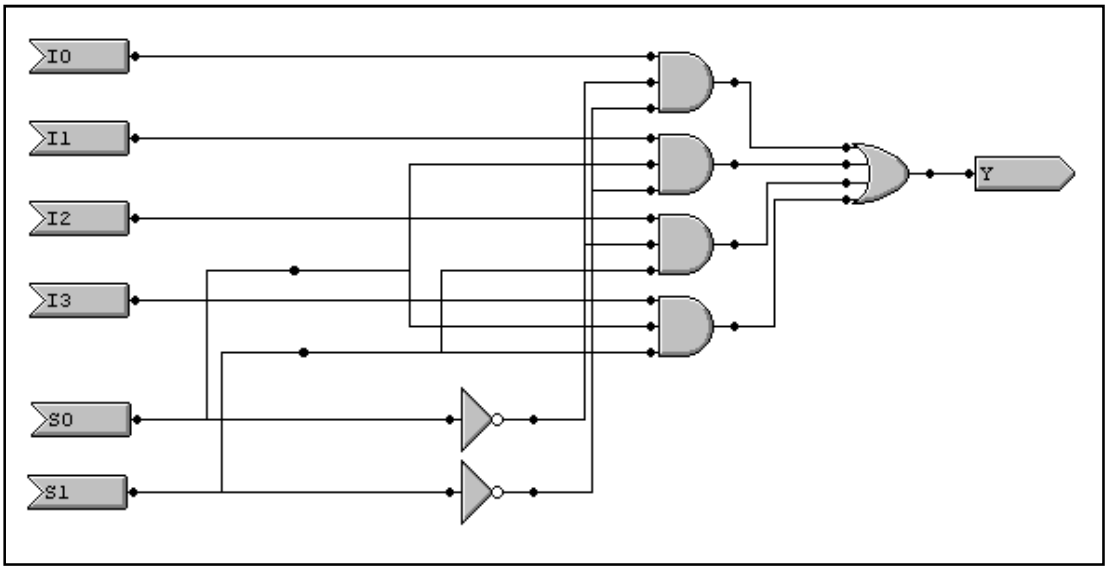
Tablo 2.2. 3x8 lik kod çözücünün doğruluk tablosu

Enable	**Girişler**			**Çıkışlar**							
E	A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

2.3.2. Seçiciler

Bir seçici 2^n tane giriş hattının bir tanesinden aldığı ikili bilgiyi, tek bir çıkışa yollayan birleşik bir devredir. Seçim girişlerinin ayarlanması ile belirli bir veri giriş hattının değeri tek çıkışa gönderilir. 2^n den 1 e seçici, 2^n giriş hattına sahiptir. n seçim giriş hatları hangi giriş verisinin çıkışa yollanacağını belirtir[23].

4x1 seçici Şekil 2.13'de gösterilmiştir. 4 veri giriş hattından her biri I_0 dan I_3 e kadar "ve" kapılarının girişlerine uygulanır. "ve" kapısının çıkışları bir "veya" kapısına uygulanarak tek bir çıkış elde edilir. S_0 ve S_1 seçim girişleri hangi girişin çıkışa yönlendirileceğini belirler.



Şekil 2.13. 4x1 hatlı seçici

Devrenin çalışmasını görmek için S_1S_0=10 olduğu durum göz önüne alınsın. I_2 girişinin bağlı olduğu "ve" kapısının diğer girişleri 1 dir. Diğer 3 "ve" kapısından en az bir tanesinin girişi diğerlerinin çıkışını 0 yapacak şekilde 0 a eşit olur. Böylece "veya" kapısının çıkışı I_2 değerine eşit olur. Seçilen girişten çıkışa doğru bir yol elde edilir. 4x1 seçicinin fonksiyon tablosu Tablo 2.3'de verilmiştir.

Tablo 2.3, 4 veri girişi arasındaki bağlantı ve çıkış hattını, seçilmiş S_0 ve S_1 giriş hatlarına bağlı olarak göstermektedir. Seçim girişleri 00 iken Y çıkışı I_0 girişine eşittir. Seçim girişleri 01 iken Y çıkışı I_1 girişine eşittir, diğerleri ikinin

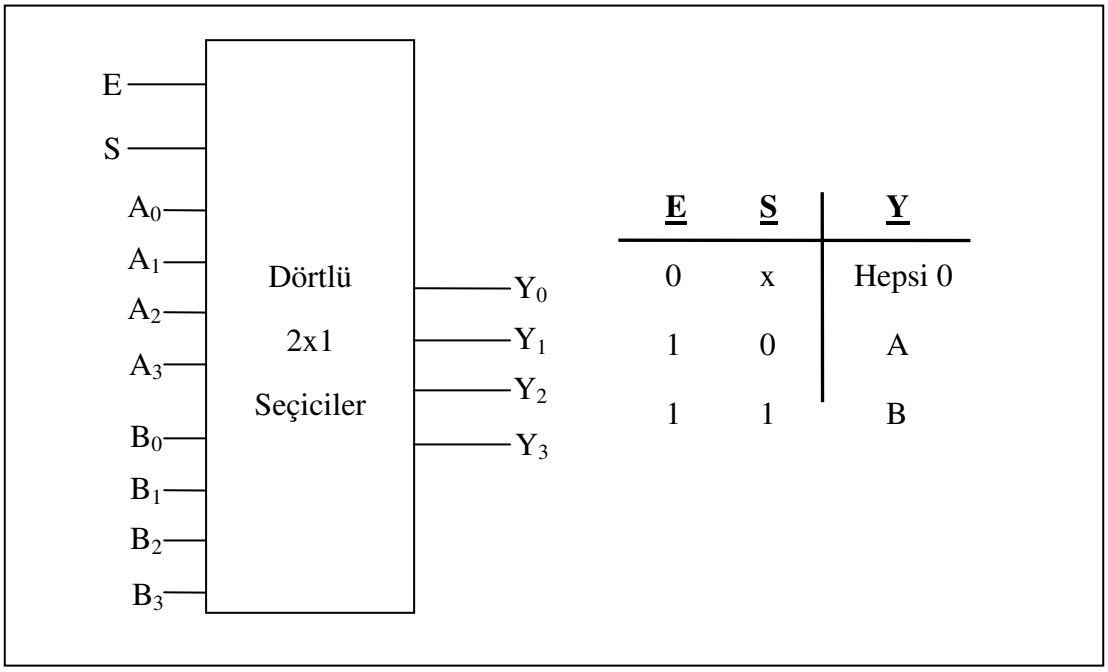
kombinezonları şeklindedir. Seçici aynı zamanda veri seçici olarak adlandırılır. Çok sayıda veri girişinden biri seçilerek ikili biçimde çıkışa yönlendirilir.

Tablo 2.3. 4x1 seçici için fonksiyon tablosu

Seçim		**Çıkış**
S_1	**S_0**	**Y**
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

"ve" kapıları ve "tersleyen" ler bir kod çözücü devre oluştururlar ve gerçekte seçilmiş giriş hatlarını çözerler. Genelde bir 2^n den 1 e seçicinin boyutu 2^n sayıda veri girişi ve tek çıkış hattı ile belirtilir. Seçiciler MUX olarak isimlendirilir.

Bazı durumlarda iki veya daha fazla MUX bir entegre devrede birleştirilir. Çoklu birim yapılarında MUX ların hepsi için seçim veya izin girişleri ortaktır. 4 adet 2x1 seçicinin blok diyagramı Şekil 2.14'de verilmiştir. Devre 4 MUX'a sahiptir ve bunlardan her biri iki giriş hattından birini seçebilir. Y_0 çıkışı A_0 veya B_0 girişinin herhangi birinden olabilir. Benzer şekilde Y_1 çıkışı A_1 veya B_1 girişinin herhangi birinin değerini alabilir ve böylece devam edilir. S giriş hattı 4 adet MUX'tan herhangi birinin bir hattını seçer. E izin girişi normal çalışmada aktif olmak zorundadır. Her ne kadar devre 4 MUX'a sahipse de bu 4 bitlik veri hattından birini seçen devre olarak düşünülebilir. Fonksiyon tablosundanda görüleceği üzere E=1 olduğunda birim aktif hale geçer. Daha sonra S=0 ise A'nın 4 girişi 4 çıkışa yönlendirilir. Diğer taraftan S=1 ise 4 B girişi 4 çıkışa yönlendirilir. S'in değeri ne olursa olsun eğer E=0 ise çıkışlar 0 olur.



Şekil 2.14. Dörtlü 2x1 seçiciler

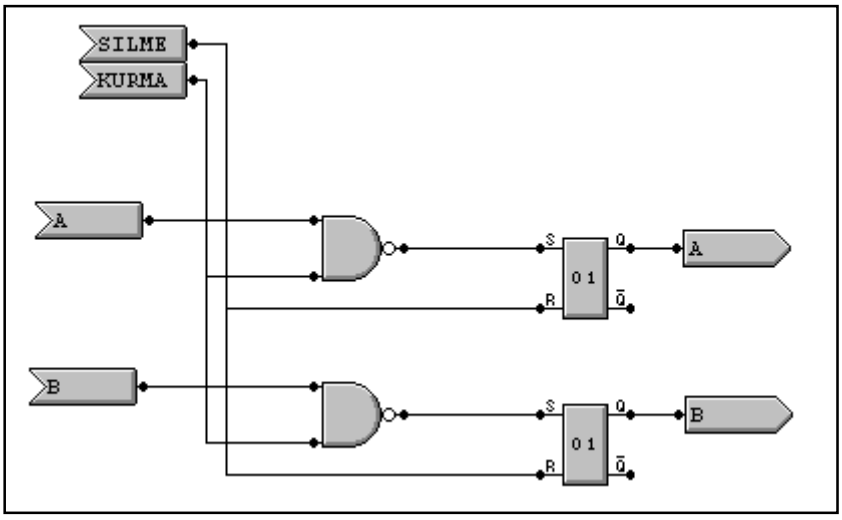
2.3.3. Kaydediciler

Lojik kapılar ve FF'lerden oluşan, ikili bilgileri geçici olarak saklamak için kullanılan devreler, 'kaydediciler' (registers) olarak adlandırılır. Kaydedicilerin sayıcılardan tek farkı; kaydedicilerin sayıcılarda olduğu gibi belirli bir sayma dizisini devamlı tekrarlamamasıdır. Kaydediciler; bilgisayarlarda bilgi depolama, ikili toplayıcı-çıkarıcı devrelerde bilgi tutma ve bilgi transferi gibi işlemlerde, sayıcılarda bellek birimlerinde, vb. yerlerde kullanılırlar[23].

2.3.3.1. Paralel yüklemeli kaydediciler

Bütün bilgilerin aynı anda FF'lere yüklendiği kaydediciler, 'paralel kaydediciler' olarak adlandırılır. Paralel kaydedicilere bilgi yükleme işlemi, senkron veya asenkron olarak gerçekleştirilir. Tetikleme girişi kullanılmadan, bilgilerin kurma girişi yardımı ile yüklendiği kaydedici devreleri 'asenkron paralel kaydedici' olarak isimlendirilirken, bilgilerin yüklenmesi için tetikleme girişlerinin kullanıldığı devreler 'senkron paralel kaydedici' olarak adlandırılır[23].

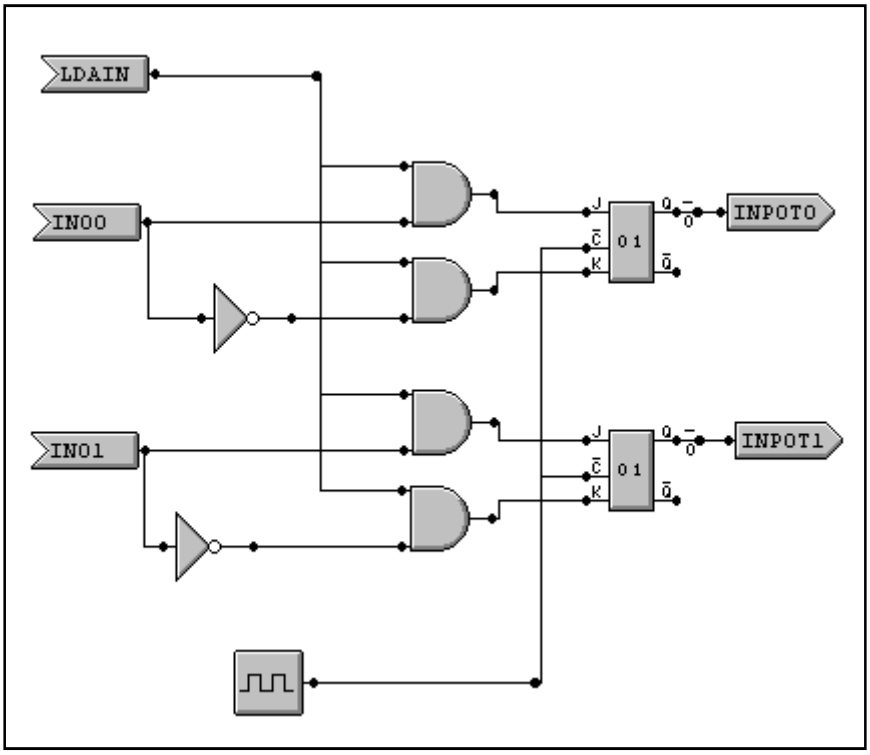
Yeni bilgi girişi için tetikleme girişinin kullanılmadığı asenkron paralel kaydedici devresinde, her yeni bilginin yüklenmesinden önce FF'lerin sıfırlanması gerekir (Şekil 2.15).



Şekil 2.15. Asenkron paralel kaydedici prensip şeması

Bilgi yüklenmeden önce yapılması gerekli sıfırlama işlemi, sıfırlama girişinin (R) aktif yapılmasıyla gerçekleştirilir. FF'lerin enerjilenmesinden sonra, yüklenecek bilgiler bilgi girişleri olarak kullanılan 'VEDEĞİL' kapılarına uygulanır. Kurma girişinin '1' yapılması ile istenen bilgiler kaydediciye aktarılır. Bunun anlamı; bilgi girişlerine uygulanan bilginin kaydediciye yüklenmesini istediğimiz anda, kurma girişinin '1' yapılmasının yeterli olmasıdır. Kaydediciye yüklenen bilgiler, bir sonraki sıfırlama işlemine kadar FF'lerde saklanır.

Giriş bilgilerinin kaydedicilere yüklenmesi için gerekli tetikleme sinyallerinin aynı anda aynı kaynaktan uygulandığı paralel kaydediciler, 'senkron paralel kaydediciler' olarak isimlendirilir. D tipi FF'lerin kullanıldığı bu tip kaydedicilerde, 'Clk' girişinin '1' yapılması ile girişlerden uygulanan bilgiler FF'lere yüklenebilir ve çıkıştan alınabilir. 'Clk' girişinin '0' yapılması durumunda ise, girişten uygulanan bilgi çıkıştan alınamaz (Şekil 2.16).



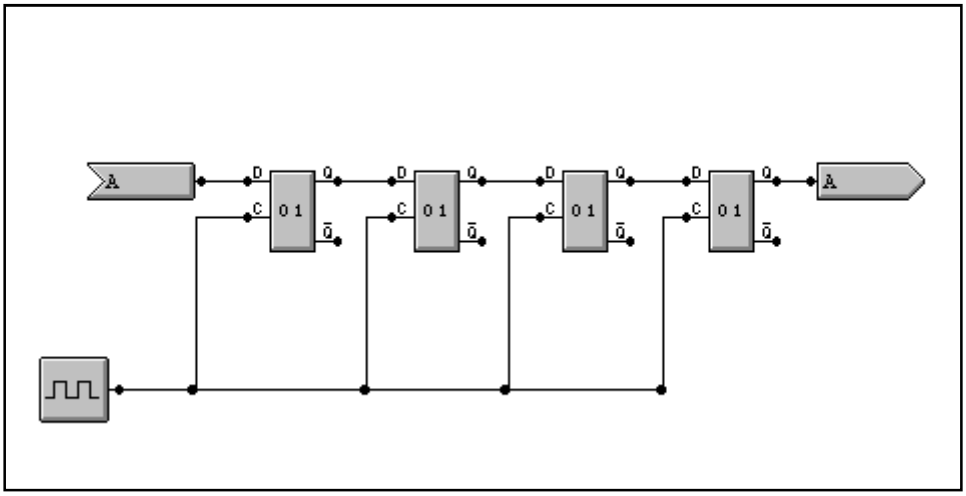
Şekil 2.16. 2 bitlik senkron paralel kaydedici

2.3.3.2. Kaydırmalı kaydediciler

Bilgilerin kaydedici devresine seri olarak yüklendiği ve her tetikleme sinyali ile önceki bilgilerin sağa veya sola kaydırıldığı devre, 'seri kaydedici' veya 'kaymalı kaydedici' olarak adlandırılır. Kayma işlemi uygulanan tetikleme sinyali ile gerçekleştirildiği için, kaydedicilerde kullanılan tetikleme sinyali 'kaydırma sinyali' olarak da adlandırılabilir. Bilginin seri olarak gönderilmesi tercih edilen yerlerde, kaymalı kaydedici (shift register) devrelerden faydalanılır[23].

Seri kaydedicilerde, ilk FF haricindeki FF'lerin çalışma konumu bir önceki FF'nin çalışmasına göre belirlenir. Diğer bir ifade ile her tetikleme palsı ile bir önceki FF'deki bilgiler sonraki FF'lere aktarılır. Girişte bulunan seri bilginin tetikleme sinyali ile senkronizeli olarak çıkışa aktarıldığı kaydedicilerde istenen sayıda FF kullanılabilir. Kullanılacak FF sayısına uygun entegreler seçilerek, oluşturulmak istenen devre gerçekleştirilir.

Kaymalı kaydediciler, bilginin kaydırılma yönüne göre isimlendirilirler. Kaydediciler bilgi kaydırılma yönüne göre sağa kaymalı, sola kaymalı ve her iki yöne kaymalı kaydediciler olmak üzere üç çeşittir. Şekil 2.17'de 4 bitlik bir kaydırma kaydedicisi görülmektedir.



Şekil 2.17. 4 bitlik sağa kaydırmalı kaydedici

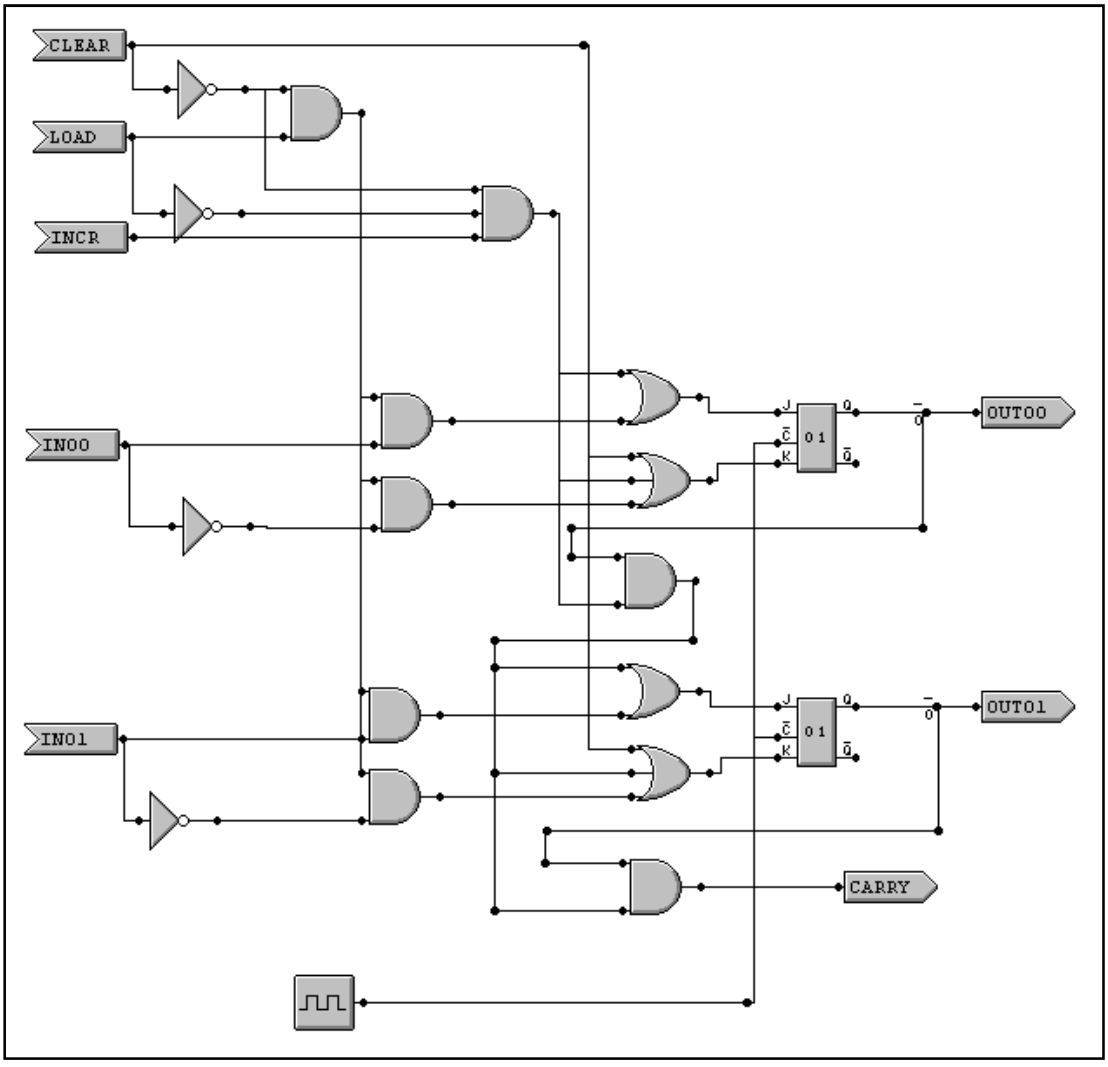
2.3.4. İkili sayıcılar

Giriş vuruşlarının uygulanmasıyla önceden belirlenmiş değerler alan kaydedicilere sayıcı denir. Giriş vuruşları saat vuruşları olabileceği gibi başka dış girişler de olabilir. Bunlar belli zamanlarda ve aralıklarda olabileceği gibi rastgele de olabilir. Sayıcılar sayısal mantık bulunan hemen hemen tüm cihazlarda bulunur. Bunlar bir olayın kaç defa tekrar ettiğini saymakta kullanılır ve zaman sinyalleri üretirler. Böylece sayısal bilgisayarlarda işlemlerin sırasını denetlerler[23].

Bir sayaç birçok farklı sırayı izleyebilir. En basit olanları en düz sırayı izleyenlerdir. İkili sayım sırasını izleyen bir sayaca ikili sayıcı denir. n bit ikili sayıcı n bit kaydedicidir ve n tane flip-flopu vardır. sahip olduğu uygun birleşik devresi ile ikili sayım yapar. n bit için sayım 0 dan 2^n-1'e kadar sürer. İkili sayıların sayım sırası 0000, 0001, 0010, 0011 vb. gibi olduğunda düşük mertebeli bitlerin her sayma işleminden sonra tümleyeni alınır. Diğer bitlerin ise, bir sayımdan diğerine ve düşük mertebedekilerin hepsi 1 ise tümleyenleri alınır. Örneğin 0111 den 1000 elde etmek için düşük mertebedeki bitlerin tümleyeni alınır, ikinci sıradaki düşük mertebeli bitin

tümleyeni alınır. 0111 in ilk biti 1 dir, üçüncü sıradaki düşük mertebeli bitin tümleyeni alınır. 1,2 ve 3. bitler 1 olduğundan tümleyeni 0'dır. Dördüncü sırdaki bit 0 olduğundan tümleyeni 1 dir.

Bir sayıcı devrede genellikle tümleyen alabilme özelliğine sahip flip-floplar kullanılır. JK ve T tipi flip-flopların her ikisi de bu özelliğe sahiptir. JK flip-flopunun J ve K girişlerinin her ikisi de 1 ise ve saat işareti pozitif geçişe sahipse tümleyen alınır. J=K=0 ise flip-flopun çıkışı değişmez. Bununla birlikte sayıcı flip-floplar saat sinyali ile kaldırılmaksızın izin girişi ile aktif ya da pasif yapılabilir. Şekil 2.18 2 bitlik bir sayıcı görülmektedir.



Şekil 2.18. Paralel yüklemeli saatli silmeli 2 bitlik ikili sayıcı

2.3.5. Bellek birimi

Bir bellek birimi saklama yapabilen flip-floplardan oluşmuş bir topluluktur. Bellek, kelime diye tanımlanan ikili bilgileri bit grupları halinde saklar. Bellek kelimesi 1 lerin ve 0 ların oluşturduğu bir gruptur ve bir sayıyı, işlem kodunu, bir veya daha fazla alfasayısal karakteri veya diğer ikili kodları içerir. 8 bitten oluşan gruplara byte denir. bir çok bilgisayar belleğinde kelimeler kullanılır ki, bitlerin 8 katı byte dır. Böylece 16 bit kelime 2 byte, 32 bit kelime 4 byte içerir. Ticari bilgisayarların hafızasında saklayabileceği byte sayısı genelde bellidir[23].

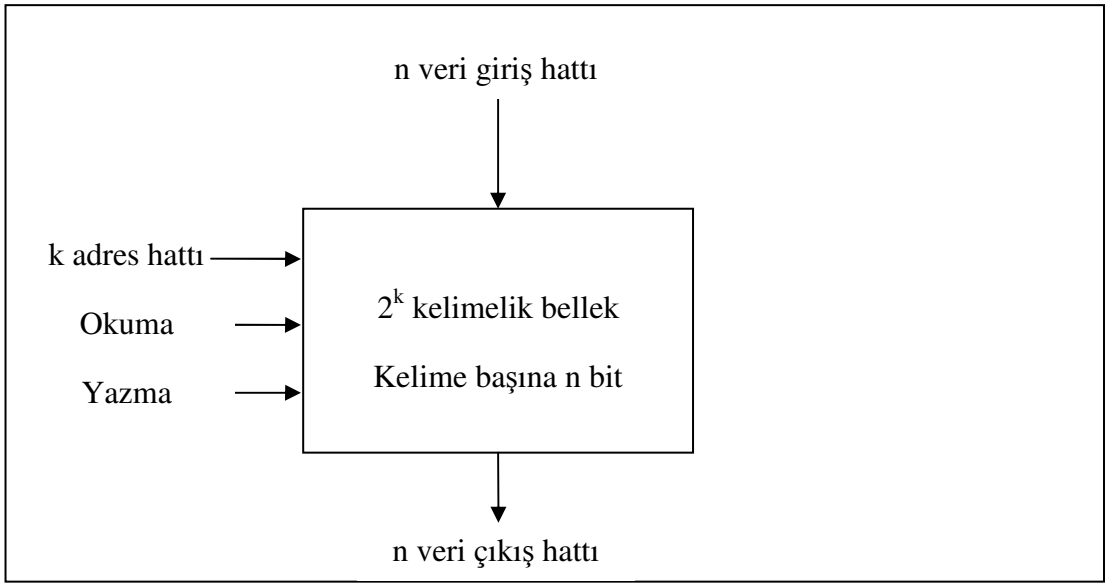
Bellek biriminin iç yapısını kelimelerin sayısı belirtir ve her kelime bir çok bit içerir. Özel giriş hatları bir tek kelime seçen adres yolları olarak adlandırılır. Bellekte her kelime belirtilen bir sayıya atanır ve adres diye isimlendirilir. 0 dan başlayıp bir bir artarak 2^k-1 e kadar devam eder. k adres hatlarının sayısıdır. Belirtilen kelimenin seçimi bellek içinde adres çizgileriyle k bitlik ikili adrese başvurularak yapılır. Bellek içinde bir kod çözücü bu adresi kabul eder ve belirtilen kelimenin bitlerini seçerek yolu açmasını sağlar. Bilgisayar hafızası 1024 kelime olarak düzenlenirse bu 10 bit adres gerektirir. 2^{32} kelime 32 bit adres gerektirir. Kelimelerin sayısı hafızada K (kilo), M (mega) veya G (giga) harflerinden biriyle işaretlidir. Bilgisayar sistemlerinde hafıza, RAM (adreslenebilir bellek) veya ROM (yalnız okunabilir bellek) şeklindedir.

2.3.5.1. Adreslenebilir bellek

Rastgele erişilebilir bellekte bellek hücrelerine bilgi aktarımı için hangi adreste olurlarsa olsunlar erişilebilirler. Yani bellekte bir kelimeye erişim süreci, hücreler belleğin neresinde olursa olsun aynı zaman sürecini gerektirir. Dolayısıyla buna "rastgele erişilebilir" adı verilir.

Bellekle çevresi arasındaki iletişim giriş ve çıkış hatları ile olur. Ayrıca adres seçim hatlarına ve aktarımın yönünün bilinmesi için denetim hatlarına gerek vardır. RAM biriminin blok şeması Şekil 2.19 da gösterilmiştir. n tane veri giriş hattı bilgilerin belleğe saklanmasını, n tane veri çıkış hattı, bellekten istenen bilgilerin alınmasını

sağlar. k tane adres hattı, k bitlik ikili sayı oluşturur. Bu bellekte mevcut 2^k tane bitlerin k tanesinden oluşan kelimedir. Aktarımın yönünü belirlemek için iki denetim girişi gerekir.



Şekil 2.19. RAM'in blok şeması

Rastgele erişilebilir veya adreslenebilir belleğe uygulanabilecek iki işlem oku ve yaz işlemleridir. Yaz işlemi belleğe girişi, oku işlemi ise bellekten dışarı aktarımı gösterir. Bu denetim sinyallerinden birini kabul ederek, belleğin içindeki devrelerin istenen işlemi yapması sağlanır. Belleğe yeni bir kelime aktarıp, belleğe yazma işlemini yapması için aşağıdaki adımlar gerçekleştirilmelidir.

1. İstenen kelimenin ikili adresini adres yoluna yaz.
2. Belleğe yazılmasını istediğin veri bitlerini veri giriş yoluna yaz.
3. Yaz girişini çalıştır.

Bellek birimi, veri giriş hatlarında halen bulunmakta olan veri bitlerini alır ve adres hattı tarafından belirlenen adrese yazar.

Halen bellekte bulunan bir kelimenin bellekten dışarı taşınması için gerekli adımlar şunlardır.

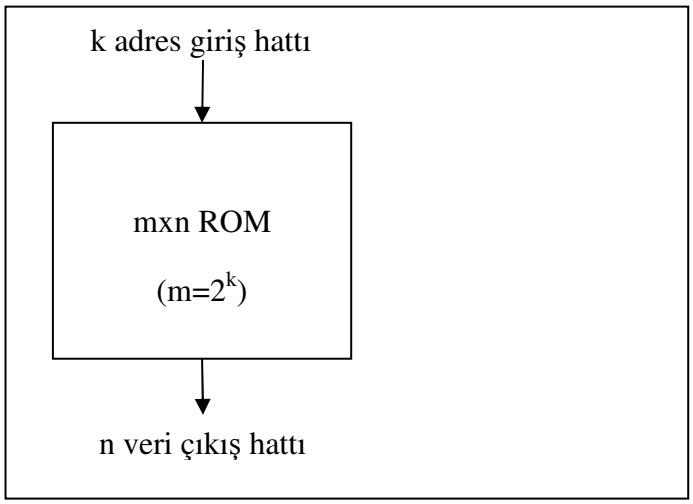
1. İstenen kelimenin adres yoluna yaz.
2. Oku girişini çalıştır.

Bellek birimi adres tarafından belirlenen kelimenin bitlerini alır ve veri çıkış yoluna aktarır. Okumadan sonra okunan kelimenin içeriği değişmez.

2.3.5.2. Yalnız okunabilir bellek(ROM)

Adından anlaşılacağı gibi yalnız okunabilir bellek, sadece okuma işlemi yapan bellek birimidir. Yazma yeteneği yoktur. Bundan şu anlaşılır ki, ROM içindeki ikili bilgiler, birimin üretimi sırasında sabit olarak yazılır ve farklı kelimeler yazarak değiştirilemez. Hâlbuki RAM, genel amaçlı bir birim olup, bilgisayar işlemleri sırasında içeriği değiştirilebilir. ROM ise kısıtlı bir birim olup sabit olarak yerleştirilmiş bulunan kelimeleri okumaya yarar. ROM'un içine yazılacak ikili bilgiler, tasarımcı tarafından belirlenir. Sonra birimin içine, istenen iç bağlantı biçimini gerçekleştirecek biçimde yerleştirilir. ROM'lar içlerinde bazı elektronik sigortalarla yapılırlar. Böylece özel bir işi için programlanabilirler. Bir kere program yapılınca, bu program bilgisayar tekrar tekrar kapatılsa ve yeniden açılsa bile silinmez.

mxn bir ROM, m kelimeden oluşur. Her kelime n bittir. Şekil 2.20 de görüldüğü gibi ROM k bitle adreslenir. Yani 2^k = m kelimelik bir bellek olup n tane çıkışı vardır. Tümleşik devre şeklinde bir ROM'un bir veya daha fazla izin girişi vardır. Böylece birkaç tane ROM bir araya getirilebilir ve daha büyük bellekler elde edilebilir.



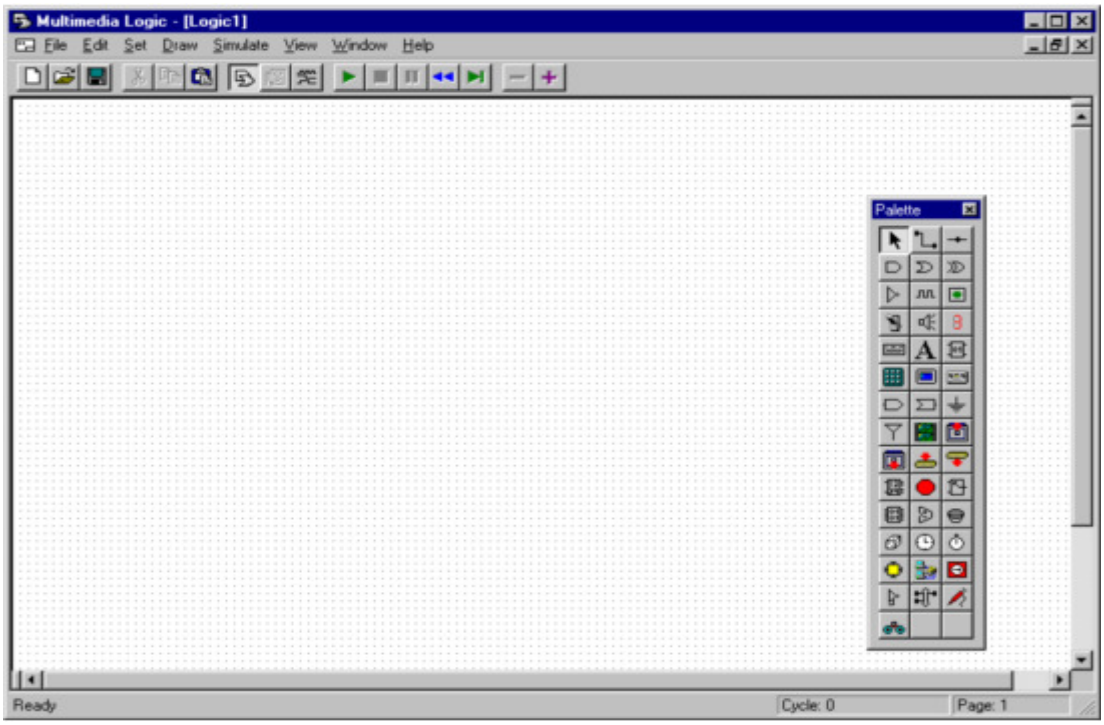
Şekil 2.20. ROM'un bellek yapısı

Bir ROM'da, oku denetimine gerek yoktur. Çünkü herhangi bir anda, adres tarafından seçilen kelimenin n biti çıkışa gider. ROM'da tüm çıkışlar girişlerin fonksiyonu olduğundan tanım olarak ROM birleşik devredir. Gerçekte, ROM kod çözücüler ve "veya" kapılarından oluşur. Daha büyük saklama hacmine gerek yoktur. Çünkü RAM'in aksine ROM daki kelimelerin hepsi değişmez değerlerdir. ROM'lar sayısal sistemlerin tasarımında geniş bir biçimde kullanılırlar. Temel olarak ROM doğruluk tablosu ile belirlenen bir giriş-çıkış bağıntısı üretir. Bunun anlamı k girişi n çıkışı olan herhangi bir birleşik devre kurulabilir. Bilgisayarda bellek birimi olarak kullanıldığında, değişmeyen programlar veya sabitler çizelgesi gibi işler için kullanılır. ROM bilgisayarlarda denetim biriminin tasarım ve kurulumunda da kullanılır. Böylece, bilgisayarda muhtelif işlerin yapılması ve denetimi için gerekli, iç denetim değişkenlerinin sırasının ve değerlerini saklamakta kullanılır. Denetim biriminde, ikili denetim bilgilerinin saklanmasında ROM kullanılan bir bilgisayarın denetim birimine mikro programlanmış denetim denir. Bu çalışmada denetim birimi donanım kontrollü olarak tasarlanmıştır.

BÖLÜM 3. MULTIMEDIA LOGIC SİMÜLASYON PROGRAMI

3.1. Giriş

Multimedia Logic programı dijital lojikte kullanılan temel elemanların işleyişini ve bu elemanlarla oluşturulmuş devrelerin çalışma şeklini gerçek dünya şartları, maliyet v.b göz önünde bulundurulmadan görsel olarak sunan bir simülatör programıdır[24]. Bu program tamamen açık kaynak kodlu olup www.softronix.com\logic.html adresinden indirilebilir. Program çalıştırıldığında devre tasarımı yapmamıza izin veren Şekil 3.1 deki gibi çalışma ekranı gelir. Ekranda yer alan Palet araç çubuğundan eklemek istenilen bir elemanın üzerine gelinerek, farenin sol tuşuna basılı tutup çalışma alanında istenilen yere sürükle-bırak prensibi ile istenilen kadar eleman çalışma alanına eklenebilir.



Şekil 3.1. Multimedia Logic program arayüzü

Multimedia Logic programında bir node(bağlantı noktası)'un 3 değişik durumu vardır. Bağlantı noktalarındaki değeri öğrenmek için ilk alternatif bağlantı notasına bir led(gösterge) bağlanarak öğrenilir ve led'de beliren durum aşağıdakilerden biri olabilir. Diğer bir alternatif ise programın araç çubuğunda yer alan bu simgeye  tıklanarak bağlantı noktasına gelindiğinde farenin sol tuşuna basılı tutulduğunda toggle probe adlı bu simgenin göstergesinde bağlantı noktasının değeri belirir.

a) LO(off veya False) 
b) HI(on veya True) 
c) UNKNOWN(Ne True ne de False) 

3.2. Çalışma Modları

Multimedia Logic Simülatör programı farklı durumlarda olabilir.

–  Bu buton yardımıyla program tasarım moduna geçer. Bu moddayken program üzerinde devre tasarımları yapmanıza imkan verir.

–  Tasarladığınız devreyi çalıştırmaya yarar.

–  Çalıştırdığınız devreyi durdurarak başlangıç konumuna getirmenize yarar.

–  Çalıştırdığınız devreyi herhangi bir anda durdurarak devrenizde yer alan elemanların o anki durumlarını görmenize sebep olur.

–  Çalışan devrenizi başlangıç konumuna döndürerek devrenizin yeniden başlamasına neden olur.

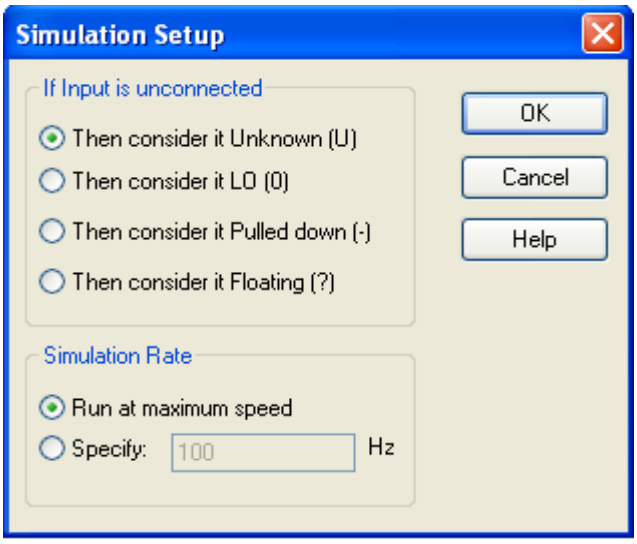
–  Tasarladığınız devreyi adım adım işletmenize olanak sağlayarak devrenizin durumunu an ve an görmenize sebep olur.

3.3. Multimedia Logic Program Menüleri

Menüler tanıtılmaya çalışırken herkesin çok aşina olduğu menüleri değil de simülasyon ortamına hazırlık ve simülasyon esnasında gerekli olabilecek menüler tanıtılmaya çalışılacak.

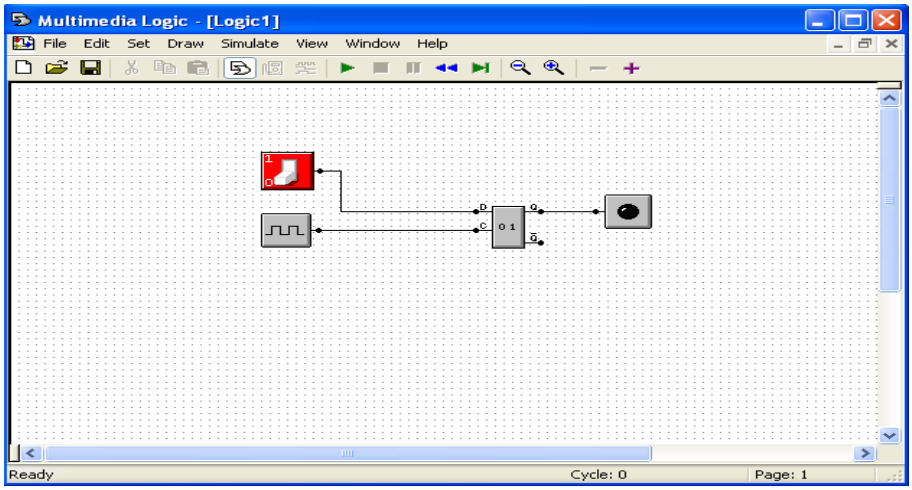
3.3.1. Simulate menüsü

Setup: Simülasyon hızının ayarlanabildiği bir yerdir. Aşağıda şekilde görüldüğü gibi iki seçenek mevcuttur. Ya maksimum makinenizin müsaade ettiği maksimum hızda çalışacaktır ya da sizin belirlediğiniz bir hızda çalışacaktır.



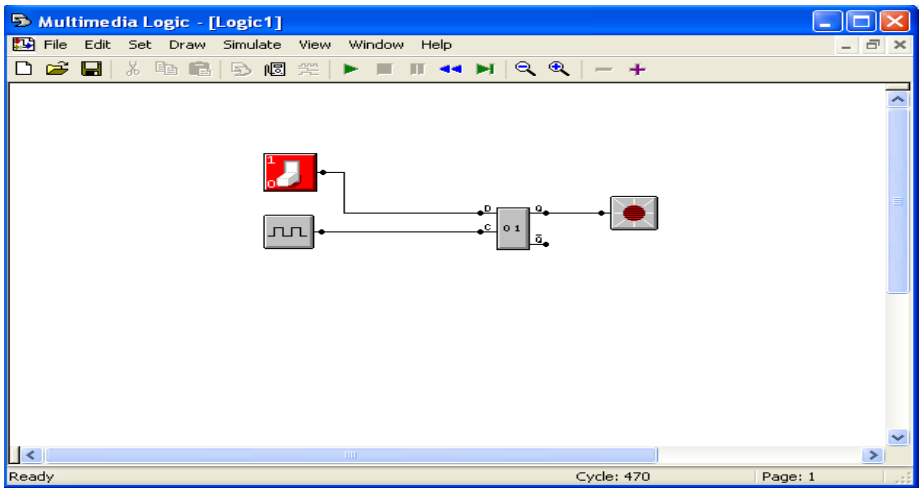
Şekil 3.2. Simülasyon hızını değiştirebilme

Bu konuyu biraz daha açmak gerekirse, örneğin sizin belirlediğiniz hız 100 Hz olsun. Bunun anlamı tasarladığınız devre çalışmaya başladığında saniye 100 cycle olarak ilerleyecektir. Flip floplar çeşidine bağlı olarak saat darbesinin ya "0" seviyesinde ya da "1" seviyesinde girişine gelen bilgileri değerlendirerek çıkışa yansıtırlar.

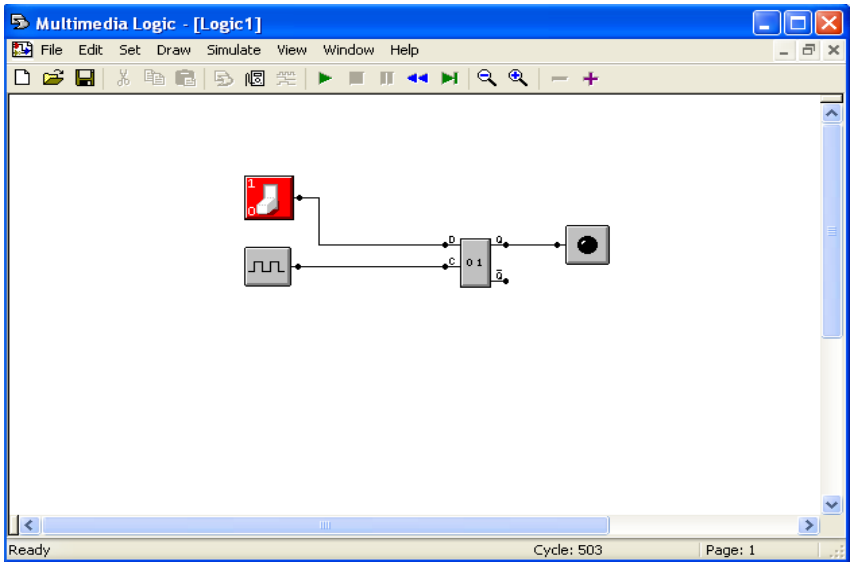


Şekil 3.3. Simulasyon hızına bir örnek

Biraz daha anlamlandırmak için yukarıdaki örneğe bakacak olursak yukarıdaki D tipi flip-flop saatin “1” seviyesindeyken aktif hale gelir. Yani saat darbesi 0 ürettiği müddetçe flip-flop girişindeki değeri çıkışa yansıtmaz. Yukarıdaki örnekte, ilerleyen konularda anlatılacağı üzere saat darbesinin lojik “0” da kalma süresi 500 cycle lojik “1” de kalma süresi 500 cycle olarak ayarlanmıştır. Simülasyon hızı yukarıda belirtildiği gibi 100 Hz, yani saniyede 100 çevrim(cycle)dir. Aşağıda şekillerin birincisinde simülasyonun 470. çevrim(cycle)deki anındaki durumu, ikinci şekilde ise simülasyonun anındaki durumu gösterilmiştir.



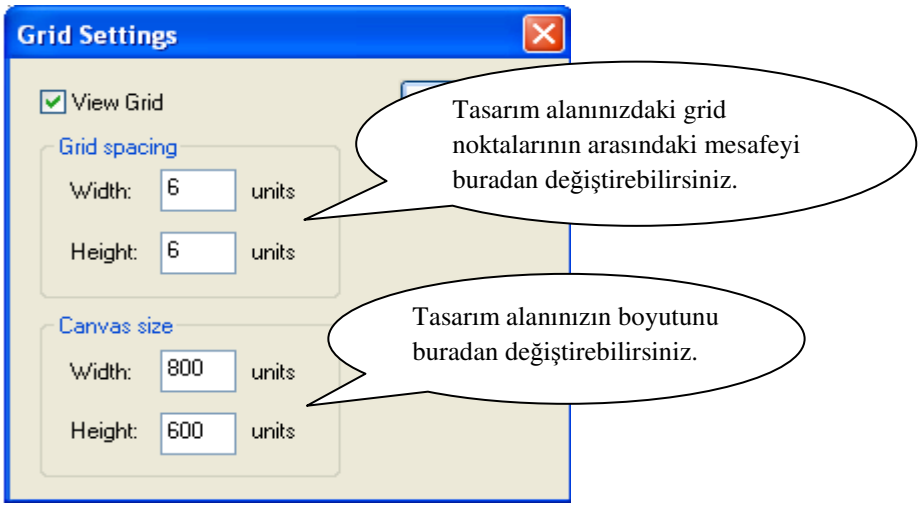
Şekil 3.4. Simulasyonun 470 çevrim geçtikten sonraki durumu



Şekil 3.5. Simulasyonun 503 çevrim geçtikten sonraki durumu

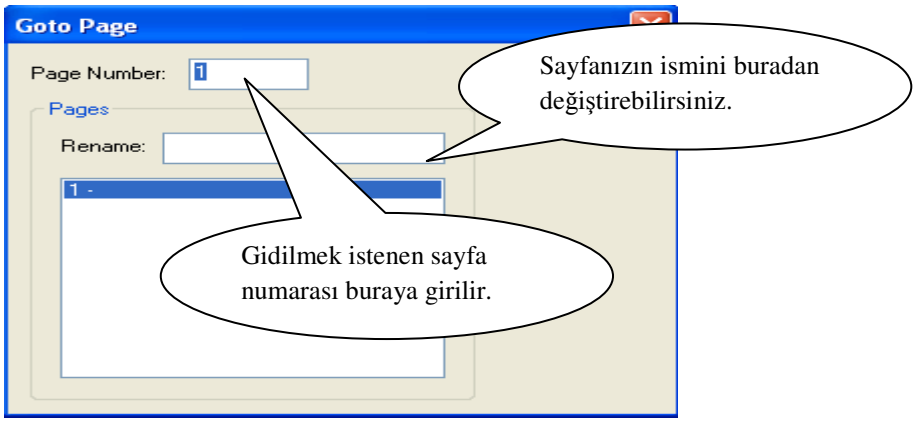
3.3.2. View menüsü

Grid Settings: Aşağıdaki ekran çıktısında da görüldüğü gibi devreyi tasarladığınız alandaki grid noktalarının genişliğini, yüksekliğini ve en önemlisi tasarım yaptığınız alanın boyutunu değiştirebilirsiniz. Dikkat edilmesi gereken nokta ise, tasarım alanınızın boyutunu en fazla 4096 değeri girebilirsizniz.



Şekil 3.6. Tasarım alanı ayarları

Go to Page: Eğer devre tasarımınız birden fazla sayfadan oluşuyor ise istenilen sayfaya gidilmesine olanak verir. Ayrıca sayfanızın ismini adlandırdığınız takdirde çok sayıda sayfadan oluşan devrenizde istenilen sayfaya gitmeniz daha kolay olacaktır. Bununla ilgili iletişim kutusu aşağıdaki gibidir.



Şekil 3.7. İstenilen sayfanın ismini değiştirme

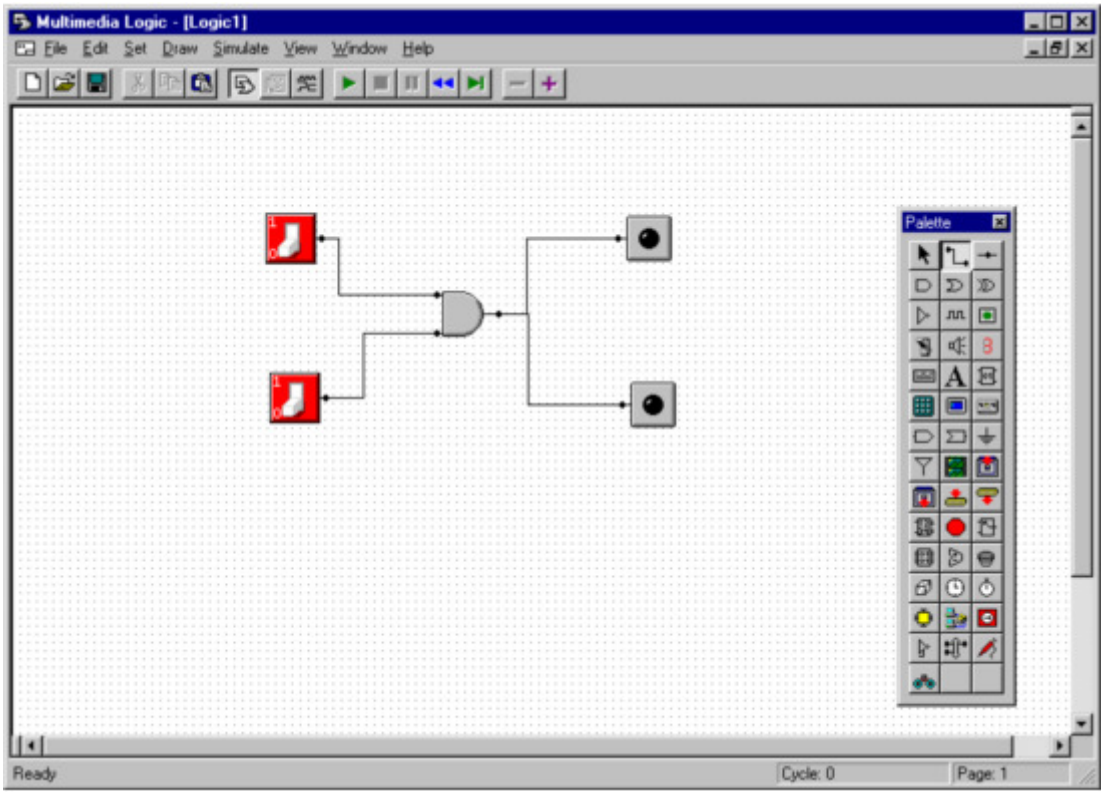
3.4. Devre Çizimi

Lojik elemanlar içerisinde en çok kullanılan elemanlardan biri olan AND kapısının tasarımını Multimedia Logic programı kullanılarak gerçekleştirelim. Aşağıdaki Tablo 3.1 de iki girişli AND kapısının doğruluk tablosu yer almıştır.

Tablo 3.1 İki girişli AND kapısının doğruluk tablosu

Q_1	Q_2	F
0	0	0
0	1	0
1	0	0
1	1	1

Çalışma ekranında yer alan palet araç çubuğunun ikinci sütunun ikinci elemanı olan AND kapısı simgesine tıklanılarak fare artı görünümlü hale gelir ve çalışma ekranında istenilen yere farenin sol tuşuna basılarak AND kapısı eklenmiş olur. AND kapısına lojik 0 veya 1 gerilimi sağlayacak olan iki adet anahtarımızı yine paletten AND kapısında yapılan işlemleri tekrar ederek çalışma alanına dahil edilir. Şimdi de AND kapısı ile anahtarlar arasındaki bağlantı ise şu şekilde yapılır. Anahtarın bağlantı noktasına fare imleci getirilerek farenin artı şekline gelmesi sağlanır ve sol tuşa basılı tutarak bağlantını diğer ucu olan AND kapısının girişlerinden birinin bağlantı ucuna gelinerek sol tuşa basılması bırakılır. Diğer bütün elemanlardaki bağlantı aynı şekilde olur. İşlem bittikten sonra oluşan devre aşağıdaki Şekil 3.8 deki gibi olur.



Şekil 3.8. Basit AND kapısı devresi

3.5. Palet Bileşenleri

Bu bölümde tasarlanan sanal mikro işlemcide kullanılan elemanları ayrıntılı olarak tanıtılmaya çalışılacak. Diğer elemanlar hakkında bilgi almak isteyenler programın web adresinden[24] programı indirerek yardım menüsünü inceleyebilirler.

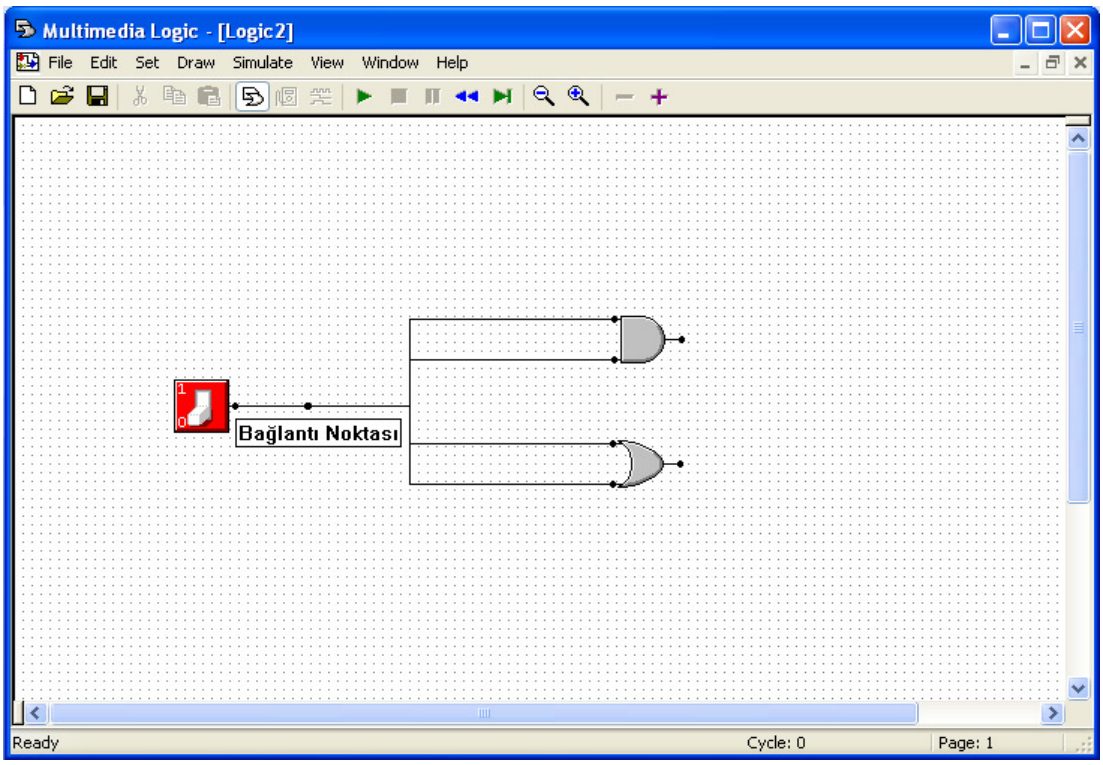
3.5.1. Bağlantı aracı 

İki elemanı birbirine bağlamaya yarayan elemandır. İki eleman arasında bağlantı yapılmaya çalışılırken dikkat edilmesi gereken nokta; bir elemanın çıkış noktasından diğer elemanın giriş noktasına bağlantı yapılması gerektiğidir. Ayrıca şu da unutulmamalıdır ki bir elemanın çıkışının birden fazla elemanın girişine bağlantı yapılabileceğini; tam tersi olan bir elemanın girişine birden fazla elemanın çıkışının bağlanamayacağı göz önünde bulundurulmalıdır. İki eleman arasında bağlantıyı gerçekleştirmek için bağlantısı yapılacak olan elemanın bağlantı noktasına farenin

imleci getirilip artı şeklini aldıktan sonra sol tuşa basılı tutarak istediğimiz elemanın bağlantı noktasının üzerine gelip bırakmaktır.

3.5.2. Bağlantı noktası

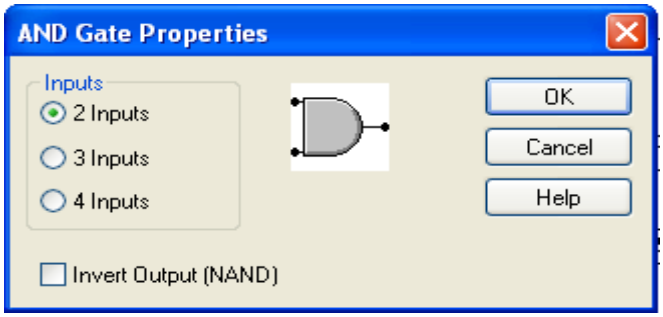
Devrenizde daha estetik kablolama yapmanıza imkan tanır. Bu elemana sadece bir elemanın çıkışı bağlanabilirken, kendisinden birden sayıda elemanın girişine bağlanabilme özelliğine sahiptir. Bu aşağıdaki şekil 3.9 de görülebilir.



Şekil 3.9. Bağlantı Noktası

3.5.3. AND(ve) kapısı

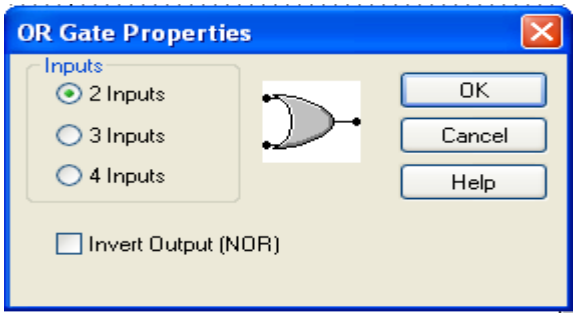
Çeşitli elemanların çıkışlarından gelen verileri lojik ve işlemine tabi tutar. AND kapısının Şekil 3.10'da görüleceği gibi iki, üç ve dört girişli AND kapısı ekleme imkanı vardır. Ayrıca iki, üç ve dört girişli NAND(vedeğil) kapısı da yine buradan devreye ekleme imkanı vardır.



Şekil 3.10. AND kapısı iletişim kutusu

3.5.4. OR(veya) kapısı 

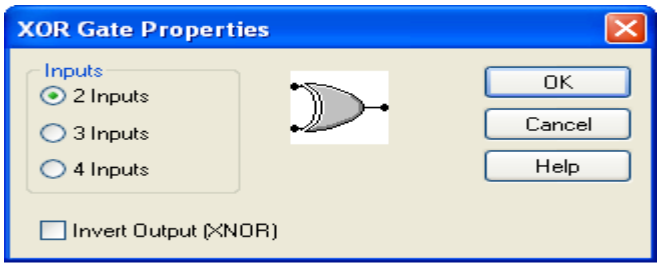
Girişlerine gelen verileri lojik veya işlemine tabi tutar. AND kapısının sahip olduğu özelliklerin aynısına sahiptir. Şekil 3.11 de OR kapısı iletişim kutusu özellikleri görülmektedir.



Şekil 3.11. OR kapısı iletişim kutusu

3.5.5. XOR kapısı 

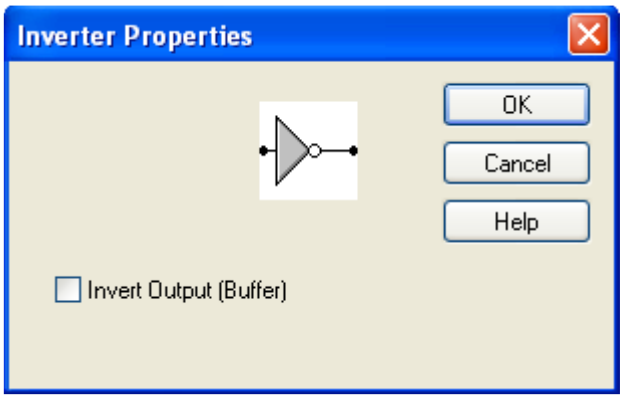
Girişlerine gelen verileri lojik xor işlemine tabi tutar. AND ve OR kapısının sahip olduğu özelliklere sahiptir. Şekil 3.12 de XOR kapısı iletişim kutusu görülmektedir.



Şekil 3.12. XOR kapısı iletişim kutusu

3.5.6. NOT(değil) kapısı

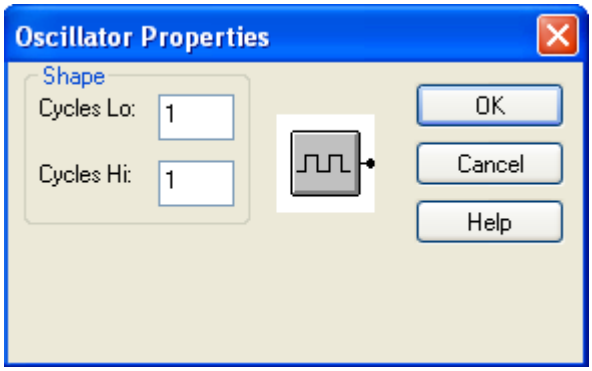
Girişine gelen veriyi lojik değilini alarak çıkışına verir. Aynı zamanda Şekil 3.13 de iletişim kutusunda görüldüğü gibi bu kapıyı buffer olarak ta kullanma özelliği vardır. Bir başka deyişle, girişi olduğu geçirir. Bu ilk bakışta anlamsız gelse de devrelerdeki senkronizeyi sağlamak için gerekli bir elemandır.



Şekil 3.13. NOT kapısı iletişim kutusu

3.5.7. Osilatör aracı

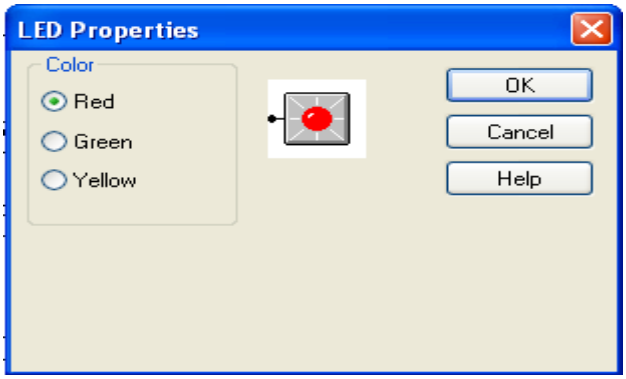
Devrelerde saat sinyali üretmeye yarayan elemandır. Şekil 3.14 deki iletişim kutusundan da görüleceği gibi ne kadar süre ile lojik 0 ve ne kadar süre ile lojik 1 seviyesinde kalacağı buradan ayarlanır.



Şekil 3.14 Osilatör Elemanı

3.5.8. Led aracı

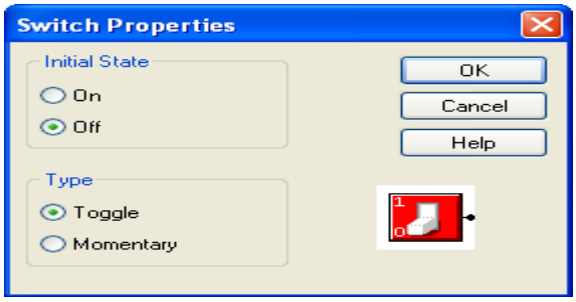
Devrede yer alan herhangi bir bağlantı noktasına bağlanarak devrenizin o kısmındaki durumu hakkında bilgi verir. Çıkış olarak 0(off),1(on) ve U(unknown) durumlarına sahiptir. Şekil 3.15'deki iletişim kutusundan kırmızı, yeşil ve sarı olarak ayarlanabilir.



Şekil 3.15. LED aracı iletişim kutusu

3.5.9. Anahtar

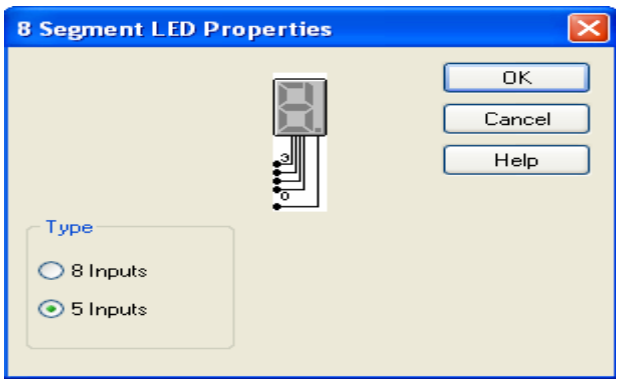
Anahtar elemanı 2 durumlu bir giriş elemanıdır. Simülasyon çalışmaya başladığında bağlı olduğu elemana lojik 0 veya lojik 1 sinyalini gönderir. Şekil 3.16 daki iletişim kutusunda ilk durum için ya lojik 0 ya da lojik 1 olarak ayarlanarak gerekli elemanların simülasyon başladığında ilk durumu verilmiş olur. Toggle ve Momentary olmak üzere iki çeşit anahtar vardır. Toggle, yani bağlanmış olduğu elemana ya 1 yada 0 sinyalini gönderir, Momentary ise anlık olarak devreye ilk durumuna bağlı olarak 0 veya 1 sinyalini gönderir ve ilk durumuna geri döner.



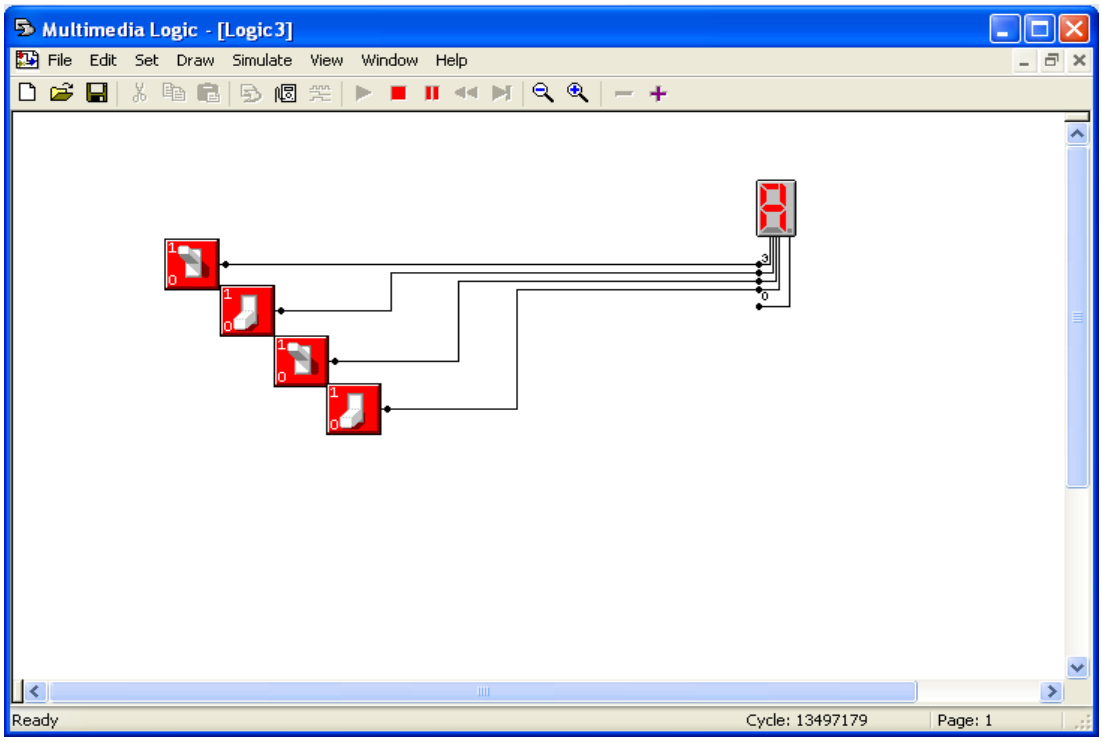
Şekil 3.16. Anahtar iletişim kutusu

3.5.10. Sekiz parçalı display

Bu aracın giriş sinyallerine bağlı olarak 0 ila F arasında hexadecimal sayı üretir. Bu elaman vasıtasıyla devrenin elemanlarının çıkışları hakkında bilgiyi kullanıcıya iletir. Beş ve sekiz girişli olmak üzere iki çeşidi olmakla beraber sekiz girişli olanında hangi segmentin on ya da off olduğunu kullanıcıya bildirir. Bu eleman ile ilgili iletişim kutusu aşağıda Şekil 3.17 de görülmektedir. Bununla ilgili bir örnek Şekil 3.18 de görülmektedir.



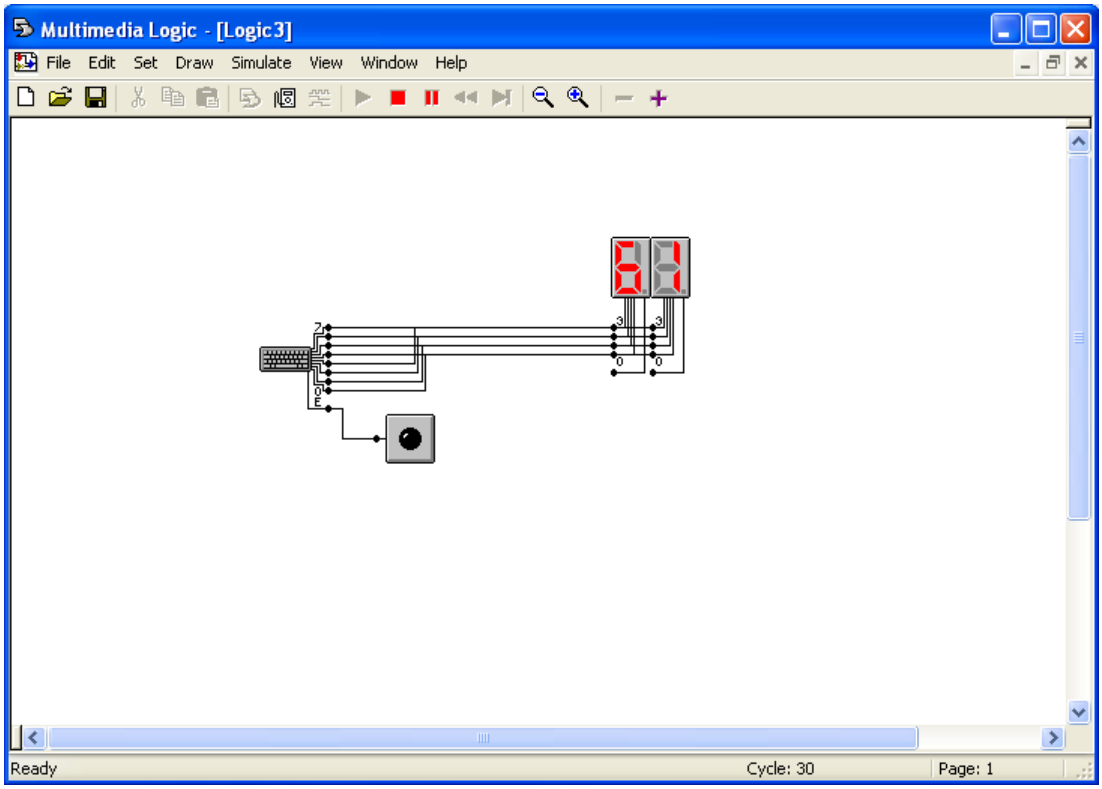
Şekil 3.17. Sekiz Segmentli LED iletişim kutusu



Şekil 3.18. Sekiz segmentli LED'e bir örnek

3.5.11. Klavye

Bilgisayarın klavyesini kullanarak devreye veri girilmesine imkân tanır. Devreye birden fazla klavye eklenebilir, ancak bir anda sadece bir tanesi kullanılabilir. Simülasyon başladığında klavyeyi kullanabilmek için klavye simgesi üzerine tıklanarak klavye aktif hale getirilir. Klavyede yer alan E çıkışı bir tuşa basıldığında sinyal üretir. Bu sinyal ile klavyeden bir tuşa basılıp basılmadığını kontrol eden devreler tasarlamak için kullanılır. Şekil 3.19 da klavye ile ilgili bir örnek görülmektedir.



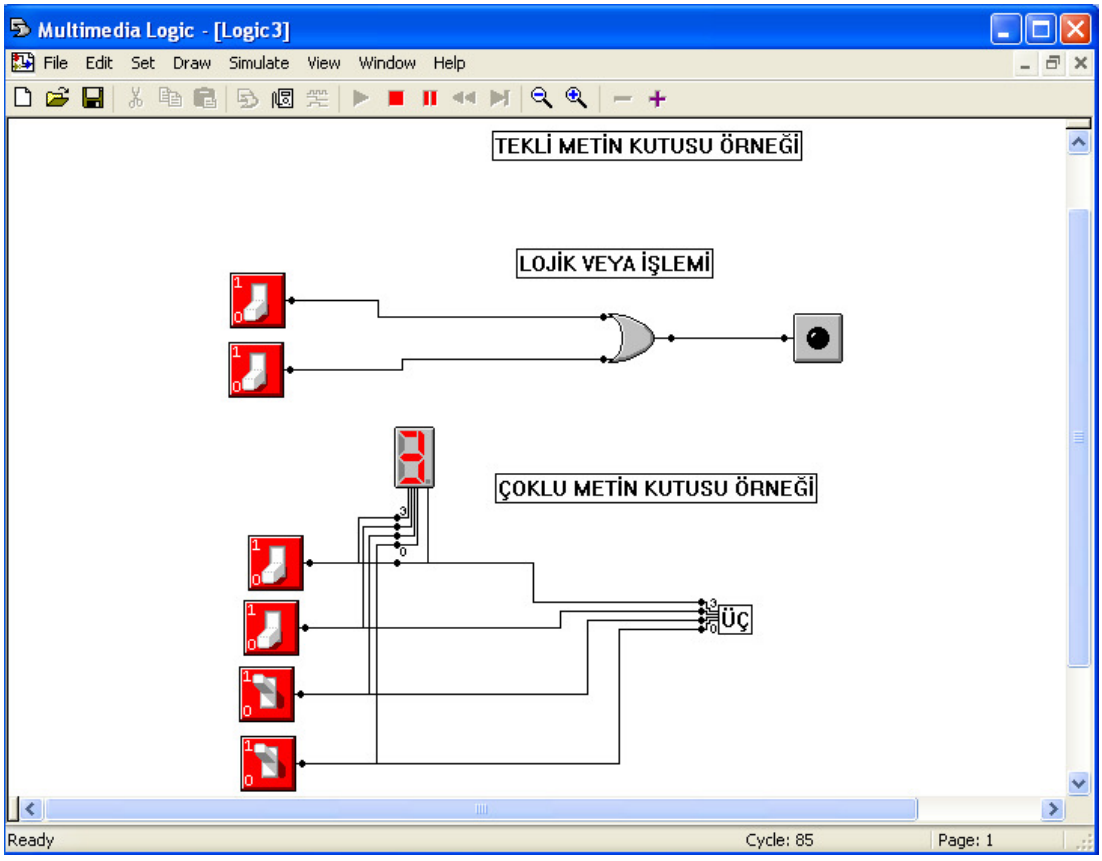
Şekil 3.19. Klavyeden "a" tuşuna basıldığında üretilen ASCII kod

3.5.12. Metin kutusu

Tekli ve çoklu olmak üzere iki çeşidi vardır. Tekli metin kutusu devrenizin gerekli yerlerine açıklama ihtiyacı olduğunda faydalı bir elemandır. Çoklu metin kutusu ise devrenizin durumuna göre 16 farklı şekilde çıktı üretebilen bir elemandır. Aşağıdaki Şekil 3.20'de hem tekli hem de çoklu metin kutusu örneği verilmiştir.

3.5.13. Flip-Flop

Flip-flop en temel bir bit saklama aracıdır. Bir flip-flop'un iki çıkışı vardır. Biri elemanın içindeki değeri, diğeri bunun tümleyenini verir. Bir flip-flop'un içindeki değeri değiştirmesi için bir saat darbesi gereklidir. Multimedia Logic programında kullanılan flip-floplarda kenar geçişli(edge triggered) ve seviye geçişli(level triggerred) olmak üzere her flip-flopun iki çeşidi mevcuttur. Kenar geçişli flip-floplar, eğer yükselen kenar geçişli ise saat darbesi lojik 0'dan lojik 1'e geçişte flip-flopun çıkışı tetiklenir; değilse saat darbesi lojik 1'den lojik 0'a geçişte flip-flopun çıkışı tetiklenir. Seviye geçişli flip-floplar ise, saat darbesi lojik 0 veya lojik 1 olduğunda flip-flopun çıkışı değişir. Çok farklı flip-floplar mevcuttur. Bu programda kullanılanlara bazı flip-floplar aşağıda anlatılmıştır.



Şekil 3.20. Metin Kutusu tiplerine bir örnek

3.5.13.1. RS tipi flip-flop

Bu programda saatli saatsiz olmak üzere iki çeşidi vardır. RS flip-flopun iki girişi ve iki çıkışı vardır. Bu iki girişten birincisi R, diğeri ise S girişidir. Çıkışları ise Q ve Q' nun tümleyeni $\bar{Q}$ dir. Saatli ve saatsiz RS flip-flopun doğruluk tablosu Şekil 3.21(a) ve Şekil 3.21(b)'de görülmektedir. Arasındaki en temel fark, saatsiz RS flip-flopu her türlü girişe tepki vererek bir çıkış üretir. Sadece girişlerin ikisi de R=0, S=0 olduğunda çıkış değişmez. Saatli RS flip-flopu ise R=0 ve S=0 durumlarının haricinde saat 1 bilgisi üretmediği müddetçe çıkışı değişmez. RS tipi flip-flopun saat=1 iken R ve S ikisi birden 1 olamaz, çünkü flip-flopun bundan sonraki durumu bilinemez. Bu kararsızlık durumundan dolayı RS tipi flip-floplar pratikte nadir olarak kullanılırlar.

R	S	Q(t+1)
0	0	Değişiklik yok
0	1	0
1	0	1
1	1	?

(a)

C(Clock)	R	S	Q(t+1)
0	x	x	Değişiklik yok
1	0	0	Değişiklik yok
1	0	1	0
1	1	0	1
1	1	1	?

(b)

Şekil 3.21. RS tipi flip-flop doğruluk tabloları a)Saatsiz b) Saatli

3.5.13.2. D tipi flip-flop

Bu flip-flopun sadece bir girişi vardır. Saatin lojik 0'dan lojik 1'e geçişi esnasında D girişindeki değeri alır. Eğer D girişi 1 ise çıkış 1, D girişi 0 ise çıkış 0 olur. Dikkat edilecek olursa D flip-flopunun durumunun değişmeden kalmasını sağlayacak herhangi bir giriş koşulu bulunmamaktadır. D flip-flopunun yalnızca bir girişe sahip olması avantajına karşı durumunu koruyamaması önemli bir dezavantajdır. Durumunu koruyabilmesi için saat sinyalinin iptal edilmesi gerekir. Bu durumda D flip-flopu durumunu değiştirmez.

3.5.13.3. JK tipi flip-flop

RS flip-flopunun belirsiz durumunun ortadan kaldırılmasıyla meydana gelmiş bir flip-floptur. J ve K girişleri, S ve R girişleri gibi flip-flopu 1 ve 0 yapacak şekilde davranırlar. J ve K girişlerinin her ikisi de 1 olması durumunda belirsizlik yerine flip-flopun çıkışının bir önceki durumun tümleyeni flip-flopun yeni çıkışı olur. Özellikle bu tip flip-floplar sayaç ve zamanlayıcı tasarımında kullanılır. JK flip-flopunun doğruluk tablosu Tablo 3.2 de gösterilmiştir. Tabloda yer alan Q(t+1) flip-flopun bir sonraki durumunu temsil etmektedir. Q(t) ise flip-flopun o anki durumuna işaret etmektedir.

Tablo 3.2. JK flip-flop doğruluk tablosu

Clock	J	K	Q(t+1)
0	X	X	Q(t)
1	0	0	Q(t)
1	0	1	0
1	1	0	1
1	1	1	$\overline{Q(t)}$

3.5.14. Keypad(Program klavyesi)

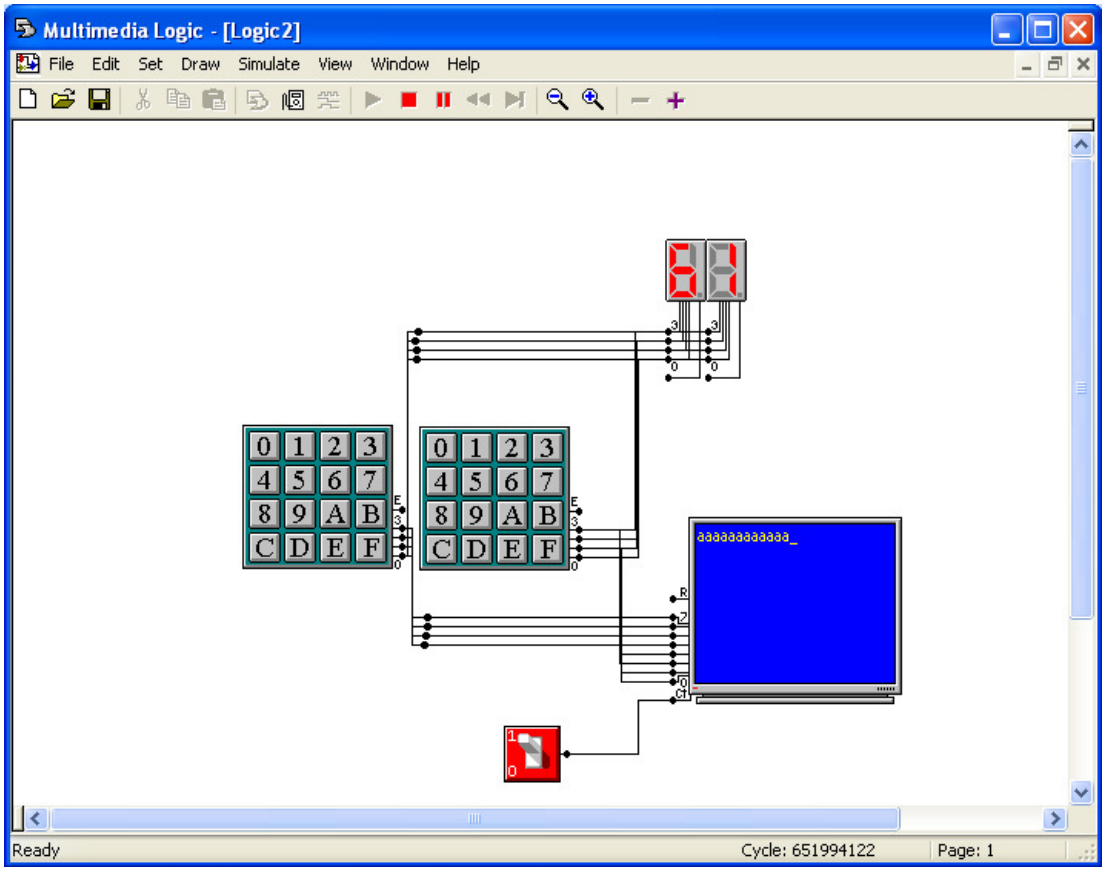
Bu eleman programın kullanıcıya sunmuş olduğu bilgisayar klavyesinin küçük bir versiyonudur. Üzerinde 0'dan F' ye kadar karakter setini ihtiva eden ve üzerine basıldığı herhangi bir karakterin hexadecimal(onaltılık) karşılığı çıkışa veren bir elemandır. Bu aracın çıkışında yer alan E çıkışı ise klavyede bir tuşa basılıp basılmadığını kontrol eder. Basıldığında E çıkışı 1 değilse 0 sinyali gönderir.

3.5.15. ASCII display(Monitör)

Çıkış aracı olan bu monitör girişine gelen hexadecimal sayıyı, o sayıya karşılık gelen karakteri ekrana yansıtır. Gelen değerin ekranda görünebilmesi için monitörün girişindeki "Ct" girişinin 0'dan 1'e geçiş yapması gerekmektedir. Örneğin, "a"

karakteri hexadecimal olarak 61 değerine sahiptir. Girişine gelen 61 hexadecimal değeri ekranda "a" karakterinin görünmesine neden olur. Bununla ilgili bir örnek aşağıda Şekil 3.22'de görülmektedir.

Monitör ekranında bir satırda toplam 16 karakter olmak üzere 8 satırlık değeri gösterebilir. Girişe gelen değer, eğer hexadecimal 0D(Enter) değeri ise monitör imleci bir satır aşağıya indirecek ve yine eğer hexadecimal 08(BackSpace) eğeri gelir ise imleç bir önceki karakteri silecektir. Monitörün girişinde yer alan R girişi ise monitörde yer alan bütün bilgileri silecektir.



Şekil 3.22. ASCII Display örneği

3.5.16. Signal sender ve signal receiver

Sinyal gönderici ve sinyal alıcı elemanlar özellikle devreniz birden fazla sayfaya yayıldığında çok gerekli elemandır. Örneğin devrenizin bir parçasını çıkışları diğer sayfada yer alan başka bir parçanın girişleri olabilir. İşte bu elemanlar sayesinde

sayfalar arasında herhangi bir gecikme olmadan sinyallerin taşınmasına sebep olan elemanlardır. Aynı sayfa içinde de bu elemanlar özellikle devrenin daha okunabilir olması için kablolama yerine bu elemanlar tercih edilebilir. Dikkat edilmesi gereken nokta ise, sinyali gönderen eleman ile sinyali alan elemanın aynı isimli olması gerekir. Bir sinyal göndericinin isminden ikinci aynı isimli bir sinyal gönderici eleman olamazken, aynı isimli birden fazla sinyal alıcı olabilir. Burada şu örneği verirsek çok daha anlaşılır olacaktır. TÜRKSAT uydusu bir adettir ama bu uydudan yayın alan milyonlarca sinyal alıcı(receiver) vardır.

3.5.17. Ground  ve plus 

Bu elemanlar bağlandıkları girişlere her zaman 0 veya 1 değeri veren elemanlardır.

3.5.18. Write file  ve read file 

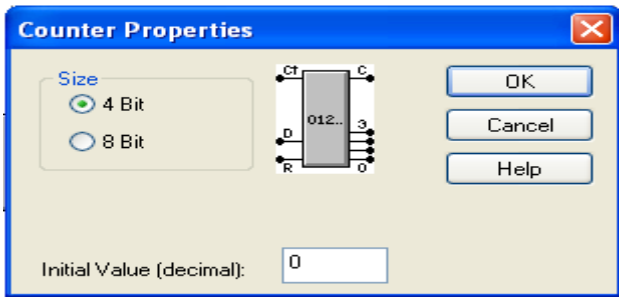
Write File elemanı büyük miktardaki verilerinizi dosyaya yazmanıza izin verir. Girişinde yer alan 8 giriş hexadecimal olarak iki basamaklı sayıya çevrilerek kullanıcının herhangi bir dosyaya satır satır yazılır. Yazılabilmesi için gerekli sinyalin Write File elemanın girişinde yer alan “Ct” girişinin 0’dan 1’e aktif edilmesi gerekir.

Read File elemanı ise, kullanıcının saklamış olduğu bir dosyadaki verileri her satırda hexadecimal olarak iki basamaklı sayı olmak koşuluyla dosyadan okur ve çıkışına iletir. Burada da dosyadan okunabilmesi için “Ct” sinyalinin aktif olması gerekir. “Ct” sinyali aktif olduğu müddetçe dosyadan sırayla okumaya devam eder ve dosya sonuna geldiğinde elemanın çıkışındaki “E” sinyali 1 olur. Herhangi bir zamanda dosyayı baştan okutmak istenildiğinde girişte yer alan “R” sinyali aktif edilmelidir.

3.5.19. Counter 

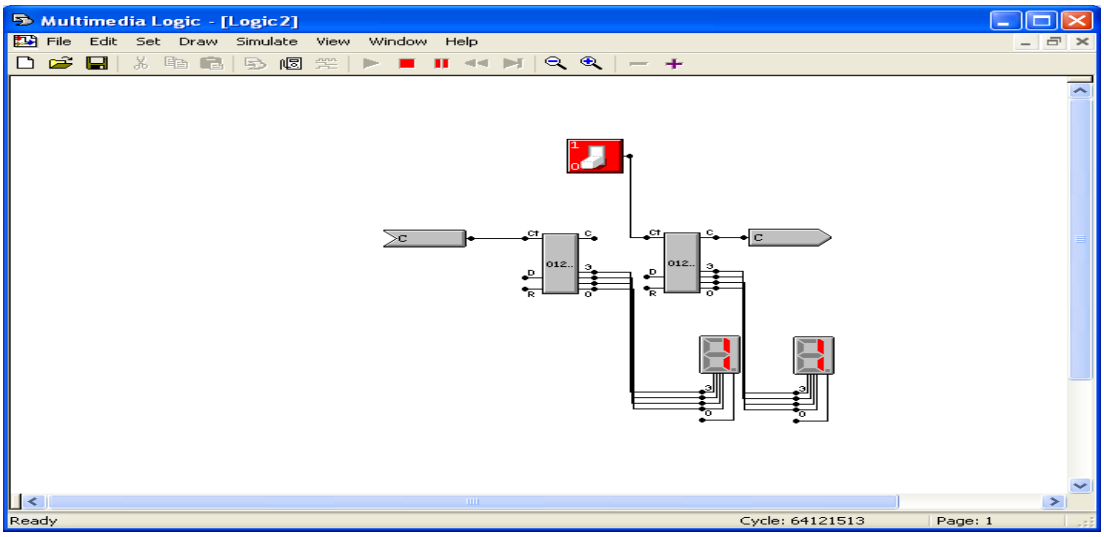
Bu eleman adından da anlaşılacağı üzere, yukarı veya aşağı yönde birer birer sayar. Sayma işlemini yapabilmesi için “Ct” girişinin 1 olması gerekir. 4 bitlik ve 8 bitlik

olmak üzere iki çeşidi mevcuttur. Bu Counter elemanıyla ilgili iletişim kutusu Şekil 3.23'de görülmektedir.



Şekil 3.23. Counter iletişim kutusu

Başlangıçta herhangi bir sayıdan itibaren saydırılmak istenirse Şekil 3.23'de görülen iletişim kutusunda Initial Value adlı Textbox kutusuna değer girilir. Sayıcının aşağı veya yukarı yönde saymasına imkan veren, sayıcının girişindeki "D" girişidir. Eğer bu D girişi 1 ise aşağı yönde değilse yukarı yönde sayar. Herhangi bir anda sayıcının tekrar baştan itibaren sayabilmesi için "R" girişine 1 değeri verilmelidir. Çıkışında yer alan "C" sinyali ise sayıcı son sayıya geldiğinde 1 olur, diğer durumlarda sıfırdır. Bu çıkış özellikle birden fazla sayıcıyı birleştirerek çok daha büyük sayıları sayabilen sayıcıları birbirine bağlamak için kullanılır. Bu çıkış diğer sayıcının "Ct" girişine bağlanarak istenilen sayıları sayabilen sayıcılar elde edilebilir. Bununla ilgili bir örnek aşağıda Şekil 3.24'de görülmektedir. Aşağıdaki örnekte iki adet 4 bitlik Counter birleştirilerek 8 bitlik Counter elde edilmektedir.



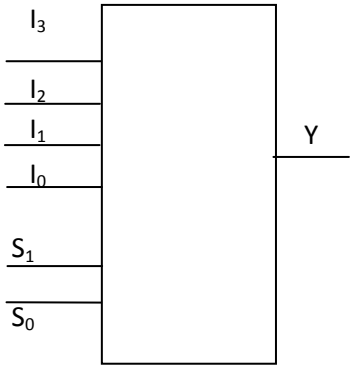
Şekil 3.24. 4 bitlik iki Counterla 8 bitlik Counter oluşturulması

3.5.20. Pause

Tasarlanan devre simüle edilirken oluşan bir şarta bağlı olarak simulasyonu durdurur. Örneğin tasarladığınız devrede belirlenen bir şarta ulaştığında devredeki elemanların o anki durumlarını kontrol etme fırsatı verir.

3.5.21. Multiplexer ve demultiplexer

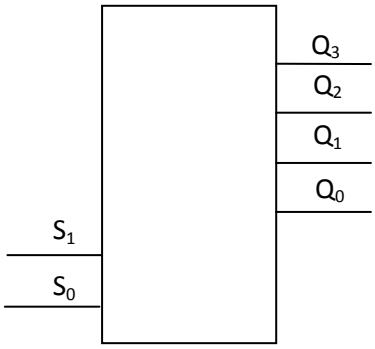
Temel olarak bir multiplexer girişine gelen 2^n adet girişten sadece bir tanesini çıkışa gönderir. Bu işlemi yapabilmesi için n adet seçici girişe sahip olması gerekir. Örneğin 4x1 lik bir multiplexer'in blok diyagram ve doğruluk tablosu Şekil 3.25.a ve Şekil 3.25.b'de aşağıda verilmiştir. Demultiplexer ise n seçme hattına bağlı olarak 2^n çıkış hattına sahiptir. Seçme girişlerine bağlı olarak 2^n çıkış hattından bir tanesi lojik 0 diğerleri ise lojik 1 dir. Demultiplexerin blok diyagramı ve doğruluk tablosu Şekil 1.26.a ve Şekil 1.26.b de verilmiştir. Multiplexeri bir veri seçici, demultiplexer ise bir decoder olarak kabul edilir. Dikkat edilecek olursa decoder de bu durum tam tersidir. Yani, 2^n çıkış hattından sadece bir tanesi lojik 0 diğerleri ise lojik 1 dir.



Şekil 3.25.a 4x1 Mux Blok Diyagramı

SEÇİM		ÇIKIŞ
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Şekil 3.25.b 4x1 Mux Doğruluk Tablosu

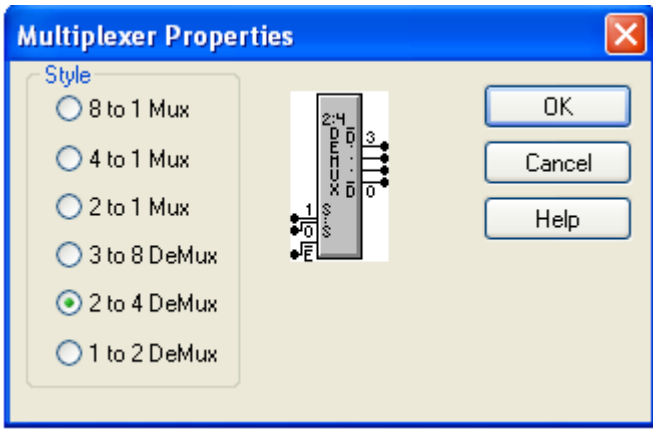


SEÇİM		ÇIKIŞ			
S_1	S_0	Q_3	Q_2	Q_1	Q_0
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1

Şekil 3.26.a. 2X4 DeMultiplexer Blok Diyagramı

Şekil 3.26.b. 2x4 Demultiplexer D. Tablosu

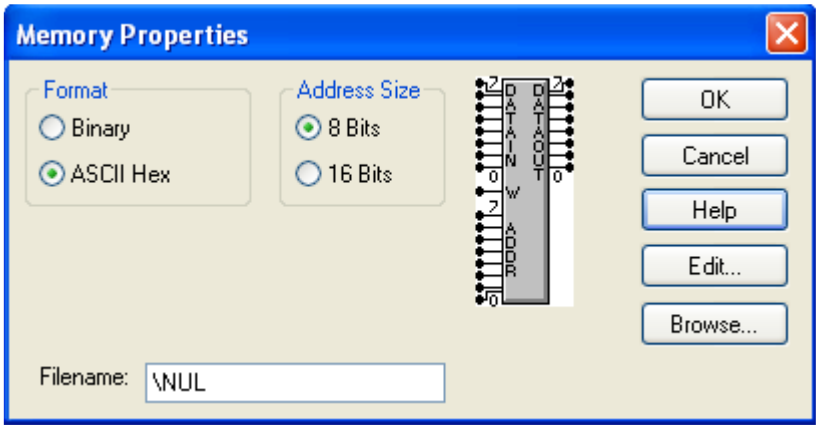
Multimedia Logic programında 3 adet multiplexer ve 3 adet demultiplexer kullanıcıların hizmetine sunulmuştur. Kullanıcı aşağıda Şekil 3.27 da görülen iletişim kutusunda istediği elemanı seçerek devresine dahil edebilir. Aşağıdaki şekilde görüldüğü gibi Multiplexer veya Demultiplexer elemanlarının E girişi lojik "0" yapılmadığı müddetçe elemanları kullanmak mümkün değildir.



Şekil 3.27. Multiplexer veya Demultiplexer iletişim kutusu

3.5.22. Memory(Bellek)

Bellek elemanındaki her bir sözcük veya satır 8 bit olup adresleme olarak hem 8 bit hem de 16 bit adreslemeyi desteklemektedir. 8 bit adresleme kullandığında 256 byte veriyi saklama kapasitesine sahiptir. 16 bit adresleme kullandığında 64KByte veriyi saklama yeteneğine sahiptir. Bellek elamanıyla ilgili iletişim kutusu Şekil 3.28'de gösterilmiştir.

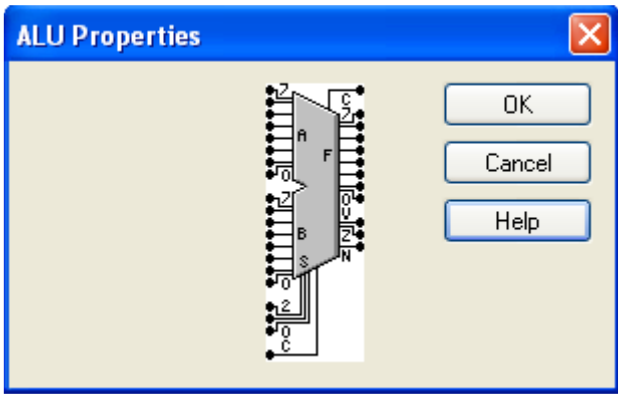


Şekil 3.28. Memory(Bellek(iletişim kutusu)

Şekil 3.28'de görüldüğü gibi adresleme olarak 8 bit veya 16 bit olarak tercih imkanı vardır. Bellek elamanına simülasyon ilk başladığında bir dosyadan bilgileri okuyup belleğine yerleştirme yerleştirebilir. Bellek elemanında yer alan W girişi lojik "1" olduğunda belleğin adres girişlerinde hangi adres varsa belleğin o adresine DATAIN girişlerinden gelen bilgi yazılır. Eğer W girişi lojik "0" ise yine belleğin adres girişlerine bağlı olarak belleğin o adresindeki bilgi DATAOUT çıkışında görünür. Belleğin adres uçlarından herhangi bir tanesi U(unknown) ise belleğin çıkış uçları U(unknown) olur.

3.5.23. Aritmetik lojik birimi(ALU)

Aritmetik lojik birimi toplama, çıkarma, bölme, çarpma, kaydırma ve karşılaştırma yapabilen temel kontrol devresidir. Bu eleman ile ilgili iletişim kutusu Şekil 3.29'da görülmektedir.



Şekil 3.29. Aritmetik Lojik birimi(ALU) iletişim kutusu

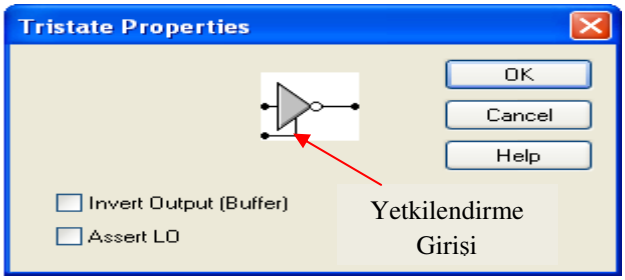
Yukarıdaki şekilden de görüldüğü gibi seçme uçlarının durumuna bağlı olarak 8 bitlik iki sayı ile yukarıda bahsedilen işlemleri yapar. Yalnız sadece tek bir Aritmetik lojik birimi ile 4 bitlik iki sayıyı çarpabilir. Çünkü iki 4 bitlik sayının çarpımı 8 bitlik bir sonuç doğurabilir. Bu problemi gerektiği sayıda ALU kullanarak 8 bitlik hatta daha yüksek sayıdaki bitli sayılar çarpılabilir. ALU'nın çıkışında yer alan Z çıkışı sonuç sıfır ise "1" değilse "0"; V çıkışı ise taşma olayı gerçekleştiğinde "1" değilse "0" dır. Tablo 3.3'de ALU'nun seçim uçlarına bağlı olarak yapacağı işlemler gösterilmiştir.

Tablo 3.3. Aritmetik Lojik Birimi(ALU) işlem Tablosu

Seçim Ucu(S)	**İşlem**	**Sonuç**
000	Toplama	A+B+(C_{in})
001	Çıkarma	A-(B+C_{in})
010	Çarpma	A*B
011	Bölme	A÷B
100	A=B	Eşit ise sonuç 1 değilse 0
101	A<B	Küçükse sonuç 1 değilse 0
110	A'yı B kadar sola kaydırma	A'nın en son biti C_{out}'a aktarılır.
111	A'yı B kadar sağa kaydırma	A'nın ilk biti C_{out}'a aktarılır.

3.5.24. TRI-STATE(Üç durumlu buffer)

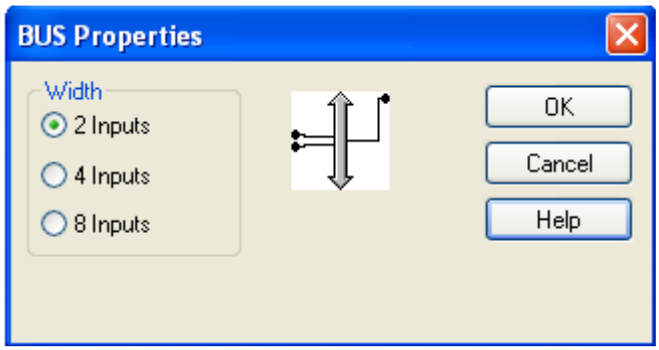
Özellikle bu eleman ortak veri yolu tasarımında kullanılır. Bu elemanın sahip olmuş olduğu Şekil 3.30'de gösterilen yetkilendirme girişi sayesinde girişine gelen veriyi ya geçirir ya da geçirmez.



Şekil 3.30. TRISTATE Buffer

3.5.25. BUS(Veri yolu)

Birden çok elemanın aynı yolu kullanmak istediğinde bu eleman kullanılır. Veri yolu kullanımında muhakkak üç durumlu bufferlardan yararlanılır. Bu elemanla ilgili iletişim kutusu Şekil 3.31'deki gibidir. Aşağıdaki şekilden de görüldüğü gibi iki, dört ve 8 girişli BUS seçenekleri mevcuttur.



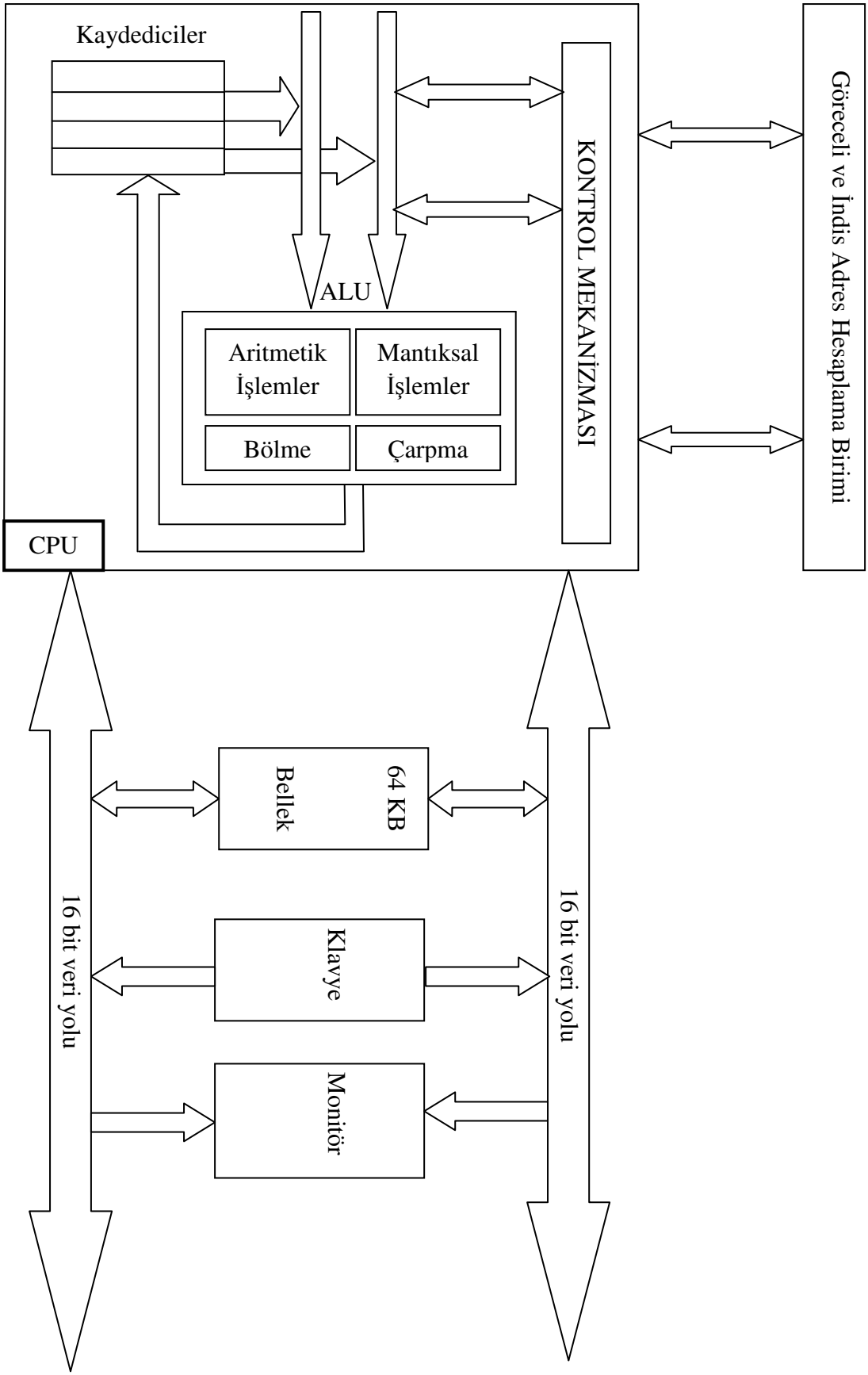
Şekil 3.31. BUS iletişim kutusu

BÖLÜM 4. TASARLANAN BİLGİSAYAR MİMARİSİ SİMÜLATÖRÜ

Bu bölümde tez çalışmasına konu olan bilgisayarın tasarımı, genel ve özel amaçlı kaydedicileri, zamanlama ve denetim yapısı ve kullandığı komut kümesi detaylı olarak anlatılacak ve daha sonra bu bilgisayarın içyapısı, kaydedicilerdeki veriler üzerine işlem yapacak mikro işlemlere değinilecektir.

4.1. Genel Özellikler

Bu tez çalışmasında tasarlanan bilgisayar, bellek ve akümülatör üzerinde işlem yapan 21 adet, indeks ve yığın üzerinde işlem yapan 8 adet, dallanma işlemleri yapabilen 22 adet, durum kod kaydedicisi üzerinde işlem yapabilen 6 adet ve 2 adet giriş-çıkış komutları olmak üzere toplam 59 adet komut icra edebilen 8 bitlik bir bilgisayardır. Bu komutları yerine getirirken kullanmış olduğu altı farklı adresleme modu vardır. Bunlar; derhal(immediate), direkt(direct), dolaylı(indirect), indis(index), göreceli(relative) ve doğal(inherent) adresleme modlarıdır. Bu bilgisayar bu komutları icra ederken 11 adet kaydedici(register) kullanmaktadır. Kullanılan kaydedicilerin bir kısmı 16 bit ve bir kısmı ise 8 bit olarak tasarlanmıştır. Bu elemanların birbirleriyle haberleşmesini sağlayacak veri yolu 16 bitten oluşmaktadır. Adres kaydedicisi 16 bit olduğundan tasarlanan bilgisayarın kullanabileceği bellek alanı 64KB'dır. Tasarlanan bu bilgisayarda göreceli adresleme ve indis adresleme modu kullanan bir komut ile karşılaşıldığında etkin adresi hesaplamak için Aritmetik ve lojik biriminin(ALU) yerine özel bir devre tasarlanmış olup, bu moddaki komutları yerine getirilme zamanı ALU'yu kullanarak etkin adresi hesaplama yöntemine göre altı çevrim(cycle) daha az zaman harcanması sağlanmaktadır. Bu tasarım ilerleyen bölümlerde ayrıntılarıyla ele alınacaktır. Bilgisayar mimarisi simülatörünün blok diyagramı Şekil 4.1'de görülmektedir.



Şekil 4.1. Tasarlanan Bilgisayar Mimarisi Simülatörü blok diyagramı

4.2. Sahip Olunan Kaydediciler(Register)

Bilgisayar buyrukları normal olarak ardışık bellek adreslerinde tutulurlar ve bir zamanda sadece bir tanesi icra edilir[25]. Denetim mekanizması, belleğin belirli bir bölgesinden bir buyruk okur ve bunu icra eder ve bu şekilde ardışık olarak devam eder. Böyle sırayla yapılan buyruk sıralamasında bir buyruğun icrasından sonra getirilecek ve icra edilecek buyrukların adreslerini hesaplamak için bir sayaca gerek vardır. Ayrıca denetim birimi içinde bellekten okunan buyruğun saklanması için gerekli bir yazaç bulunması gerekir. Bilgisayarın veriyi işlemek için işlemci yazaçlarına ve bellek adreslerini tutmak için bir yazaca ihtiyaç vardır. Bu gereksinimler Tablo 4.1 de verilmiştir.

Tablo 4.1. Kullanılan kaydedici isimleri ve bit uzunlukları

Register	Bit uzunluğu
Data Register(DR)	16 bit
Program Counter(PC)	16 bit
Adres Register(AR)	16 bit
Accumulator(AC)	16 bit
Instruction Register(IR)	8 bit
Index Register(IX)	16 bit
Stack Pointer(SP)	16 bit
Output Register(OUTR)	8 bit
Input Register(INR)	8 bit
Temporary Register(TR)	16 bit
Condition Code Register(CCR)	8 bit

Veri Kaydedicisi: Bellekten okunan verilerin tutulduğu kaydedicidir. Buradaki veri aritmetik mantık birimi tarafından değerlendirilerek veri yoluna iletilir.

Program Sayıcı: Adres ucu kadar bite sahiptir. O anda çalışacak olan komutun adresini üzerinde bulundurur. Komut çalıştırıldıktan sonra bir arttırılır.

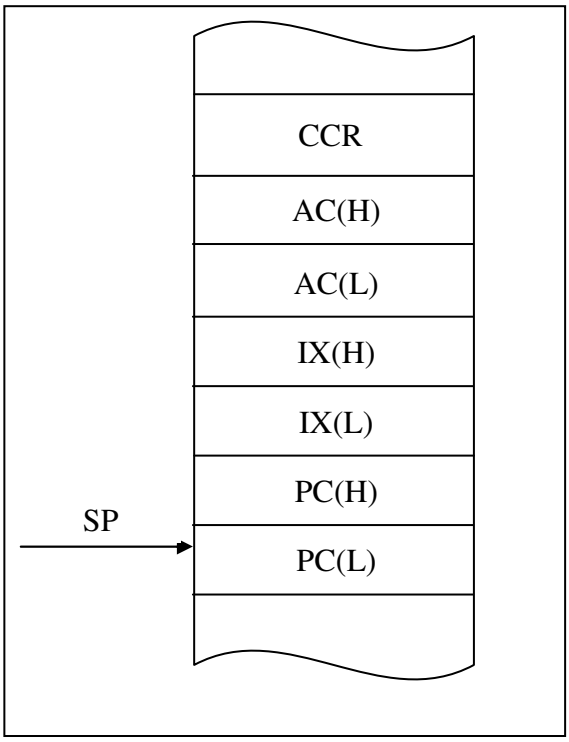
Adres Kaydedicisi: Belleğin herhangi bir bölgesindeki veriye erişimde kullanılır.

Komut Kaydedicisi: O anda çalışan komutu üzerinde bulundurur.

Akümülatör: Veri kaydedici kadar uca sahiptir. ALU tarafından kullanılır. Genelde o andaki verileri veya işlem sonuçlarını üzerinde bulundurur.

İndis Kaydedicisi: Kullanılacak gerçek hafıza yerini belirlemek için bu yazaç içindeki değer, özellikle bu bilgisayarda indisli adresleme modu kullanan komutların ikinci byte'ında yer alan offset değeriyle toplanır.

Yığın göstergeci: Hafızadaki herhangi bir hücre adresini bulundurur. Herhangi bir dallanma ile alt programlara gitme ve kesme istekleri anında mikroişlemcinin o anda gerekli olan bilgilerini dönüş anında kullanmak üzere saklamak gerekir[26]. Bunun içinde geçici olarak yığın göstergecinin RAM üzerinde göstermiş olduğu adresten geriye doğru bir veri yığını oluşturulur. İşte yığın göstergeci, bu veri yığınının oluşturulacağı başlangıç adresini üzerinde tutar. Yığına atılan son bilgi ilk alınır. Tasarlanan bu bilgisayarda yığına atılan bilgiler sırasıyla Program sayıcı(PC), indis yazacı, akümülatör ve durum yazacı içindeki bitlerdir. Şekil 4.2'de bu durum gösterilmiştir.



Şekil 4.2. Yığın yapısı

– Yığından bir okuma/yazma yapılacaksa, yığın göstergecinin işaret etmiş olduğu hafıza hücresinden okunur/yazılır.

– Yığın göstergecinin değeri mikro işlemci tarafından otomatik olarak ya bir arttırılır ya da bir azaltılır.

– Yığın türleri:

  – LIFO(Last-in First-Out): Yığına atılan son bilgi ilk alınır.

  – FIFO(First-ib First Out): Yığına atılan ilk bilgi ilk alınır.

  Bu bilgisayarda kullanılan yığın türü LIFO'dur.

– Bir PUSH komutu ile veri yığına atılırken, PULL komutu ile veri yığından alınır.

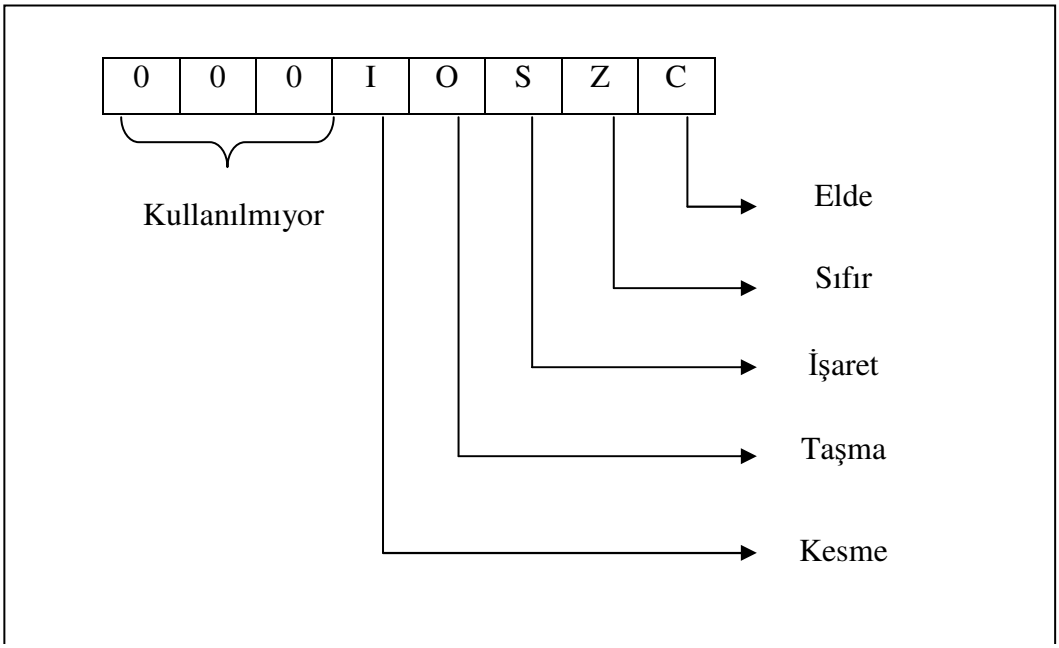
– Yığın çok düzeyli kesmelerin kolayca gerçeklenmesini, sınırsız sayıda alt programın iç içe geçirebilmesini ve birçok veri işleme türlerinin basitleştirilmesini sağlar.

– Mikroişlemcinin ana programdan alt programa gittiği zaman ana programa döneceği adresi sakladığı adres gözünün adresini içerir.

Giriş Kaydedicisi: Dış birimler tarafından girilen bilgileri saklayarak aritmetik mantık birimine gönderir. Bu birimin veri yoluyla bağlantısı yoktur.

Çıkış Kaydedicisi: Bilgisayar işlene verilerin dış dünyaya aktarımında kullanılan birimdir. Girişleri veri yoluna bağlanmış olup çıkışları dış dünyadaki bir cihazın girişlerine bağlıdır.

Durum kod Kaydedicisi: ALU ile birlikte çalışır. Bayrak kaydedicisi, bütün mikro işlemcilerde olduğu gibi, bir işlemin sonucunda sonucun ne olduğunu kaydedici bitlerine yansıtan bir kaydedicidir. Bu kaydediciye bayrak denmesinin sebebi, karar vermeye dayalı komutların yürütülmesinde sonuca göre daha sonra ne yapılacağını bit değişimiyle bu kaydedicinin bir bitlik hücrelerine yansıtmasıdır[27]. Kaydedicideki bitlerin mantıksal 1 olması bayrak kalktı, mantıksal 0 olması bayrak indi anlamına gelmektedir. Karşılaştırma ve aritmetik komutların çoğu bayraklara etki eder. Bilgisayarın sahip olduğu durum kod kaydedicisi Şekil 4.3 de görülmektedir.



Şekil 4.3. Durum kod kaydedicisi

4.2.1. Kaydecilerin özellikleri

Tablo 4.1'den de görüleceği üzere bazı kaydediciler 16 bitlik bazıları ise 8 bitliktir. 16 bit olarak tasarlanan kaydediciler düşük anlamlı sekiz biti bir kaydedici, yüksek anlamlı sekiz biti bir kaydedici olmak üzere iki adet kaydedici olarak tasarlanmıştır. Kaydedicilerin girişleri tasarlanan 16 bitlik veri yoluna bağlıdır. Düşük anlamlı bitleri veri yolunun düşük anlamlı kısmına, yüksek anlamlı bitleri ise veri yolunun yüksek anlamlı kısmına bağlıdır. Çıkışları ise yine aynı şekilde veri yoluna bağlıdır. Şekil 4.4'de verilen veri kaydedicisinin bir bitlik kısmında yer alan giriş ve çıkışları açıklaması Tablo 4.2'de verilmiştir. Yapılan bu açıklamalar diğer kaydediciler için ortak bir özellik taşıyacaktır. Çünkü diğer kaydediciler içinde ortak bir isim kodlaması yapılmıştır.

Tablo 4.2 Veri kaydedicisine ait giriş/çıkış kontrolleri

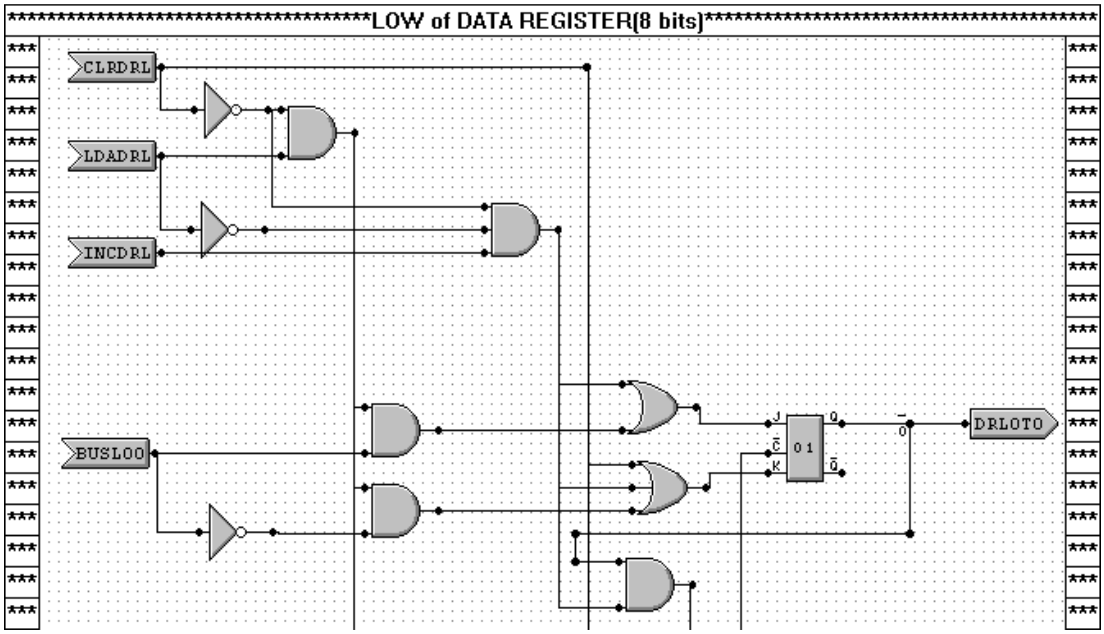
Giriş/Çıkış İsmi	Açıklaması
CLRDRL	Veri Kaydedicisinin(DR) Düşük anlamlı(L) kısmını temizle(CLR).
LDADRL	Veri Kaydedicisinin(DR) Düşük anlamlı(L) kısmına yükle(LDA).
INCDRL	Veri Kaydedicisinin(DR) Düşük anlamlı(L) kısmını arttır(INC).
BUSL00	Düşük anlamlı veri yolunun(BUS) ilk biti(00)
DRLOT0	Veri Kaydedicisinin(DR) Düşük anlamlı(L) kısmının çıkışının(OuT) ilk biti(0).

Kaydedicilerin büyük bir çoğunluğunda yükleme, bir artırma, silme kontrol girişleri vardır. Bazı kaydedicilerde bu kontrollere ilave olarak bir azaltma kontrol girişi ilave edilmiştir. Bazı kaydedicilerde ise sadece yükleme kontrol girişi vardır. Tasarlanan kaydedicilerin sahip oldukları kontrol girişleri Tablo 4.2'de verilmiştir.

Tablo 4.3 Kaydedicilerin sahip olduğu kontrol girişleri

	Sahip Olunan Kontrol Girişleri			
Kaydedici İsmi	Yükle	Temizle	Arttır	Azalt
Veri Kaydedici(DR)	Var	Var	Var	Yok
Geçici Kaydedici(TR)	Var	Var	Var	Yok
Program Sayıcı(PC)	Var	Var	Var	Yok
Adres Kaydedici(AR)	Var	Var	Var	Var
Akümülatör(AC)	Var	Var	Var	Yok
Komut Kaydedici(IR)	Var	Yok	Yok	Yok
İndis Kaydedici(IX)	Var	Var	Var	Var
Yığın Göstergeci(SP)	Var	Var	Var	Var
Giriş Kaydedici(INPR)	Var	Yok	Yok	Yok
Çıkış Kaydedici(OUTR)	Var	Yok	Yok	Yok
Durum Kod Kaydedici(CCR)	Var	Yok	Yok	Yok

Şekil 4.4'de MML programında tasarlanan kaydedicilerden veri kaydedicinin düşük anlamlı kısmının bir bitinin ekran görüntüsü görülmektedir.



Şekil 4.4. Veri kaydedicinin bir bitlik kısmı

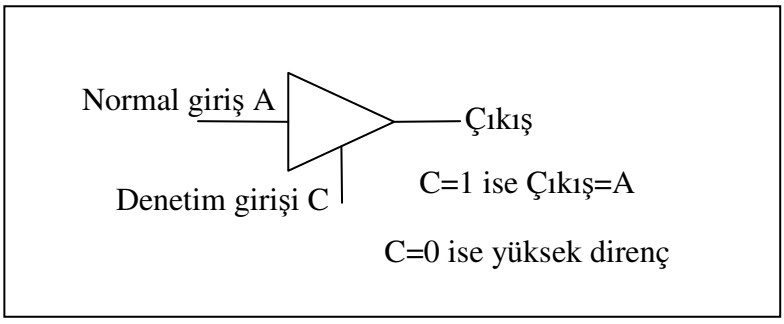
4.3. Ortak Veri Yolu(Common Data Bus)

Genel bir sayısal bilgisayar birçok kaydediciye sahiptir ve bir yol ile aktarım bilgilerini bir yazaçtan diğerine aktarmayı sağlamaktadır. Sistemdeki tüm yazaçlar arasında yollar kullanıldığından sayıları çoktur. Bir ortak veri yolu sistemi, çoklu kaydedici konfigürasyonları için kaydediciler arasındaki bilgilerin birçok etkin şekilde aktarılmasını sağlar. Bir veri yolu yapısı kaydedicideki her bir bit için ortak yolların bir kümesini oluşturmaktır. Çoklu kaydedici konfigürasyonlarında bilgilerin kaydediciler arasında aktarımı için daha etkin bir yöntem ortak veri yolu kullanmaktır[28]. Tasarlanan bilgisayarda 10 adet kaydedici ve bellek elemanını düşünecek olursak bunların çıkışlarını diğer elemanların girişlerine bağladığımızda karmakarışık bir teller ağı meydana gelir. Bu karmaşıklığı ortadan kaldırmanın yolu elemanlar arasında ortak bir veri yolu kullanmaktır.

Bu tasarlanan bilgisayarın çoğunluğunu 16 bit kaydediciler(register) oluşturduğundan veri yolu 16 bit olmakla beraber iki kısma ayrılmıştır. Bu 16 bitlik veri yolunun 8 bitlik kısmı kaydedicilerin yüksek değerlikli bitleri haberleşme için

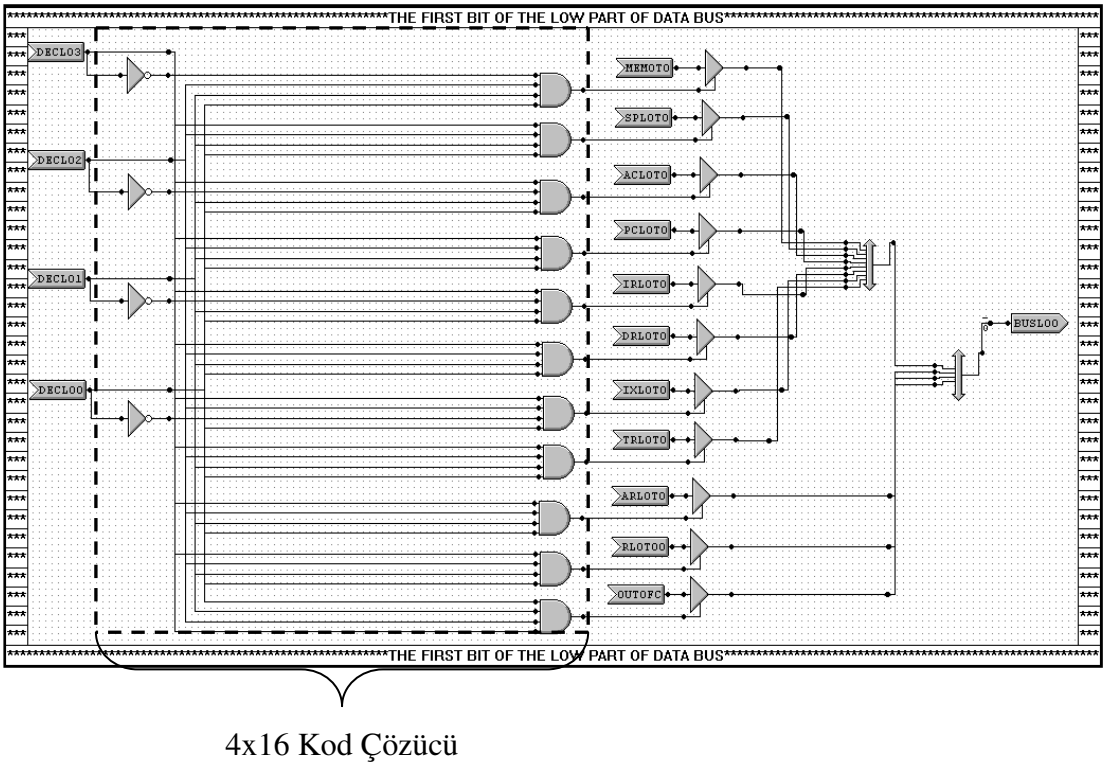
kullanırken, diğer 8 bitlik kısmı ise kaydedicilerin düşük öncelikli bitleri tarafından kullanılmaktadır. Bu şartlar göz önünde bulundurulduğunda her bir ayrı kaydedicinin bu iki adet 8 bitlik yola verilerini bırakabilmeleri için veri yolu kontrolü gerekmektedir. Bu kontrol ya seçicilerle(multiplexer) ya da üç durumlu bufferlar(tri-state buffer) ile oluşturulabilir. Bu bilgisayarda veri yolu kontrolü üç durumlu bufferlar ile sağlanmıştır. Bir üç durumlu kapı, üç durumu gösteren sayısal devredir. Bu durumlardan ikisi diğer kapılar gibi "1" ve "0" sinyallerine denktir. Üçüncü durum yüksek direnç durumudur. Yüksek direnç durumu bir açık devre gibi davranmaktadır. Bunun anlamı, çıkışlar ayrılmış durumdadır ve bir mantıksal anlamı yoktur.

Üç durumlu buffer kapısının grafik sembolü Şekil 4.5'te gösterilmektedir. Normal giriş ve denetim girişinin her ikisini de içerdiğinden, normal bir bufferdan ayrılmaktadır[29]. Denetim girişi, çıkış durumunu tanımlamaktadır. Denetim girişi "1" olduğunda normal bir buffer gibi işlem yapmakta ve normal girişi çıkışa aktarmaktadır. Denetim girişi "0" olduğunda çıkış devre dışıdır ve normal girişin değeri ne olursa olsun kapı yüksek direnç durumuna geçer. Üç durumlu kapının yüksek direnç olma durumu, diğer kapılarda var olmayan özel bir nitelik taşımaktadır.



Şekil 4.5. Üç durumlu buffer kapısı

Üç durumlu kapılarla tasarlanan 16 bitlik veri yolunun bir bitlik kısmı Şekil 4.6'da gösterilmektedir. 11 adet bufferın çıkışları ise tek bir veri yolu hattına bağlanmaktadır. Bufferların denetim girişleri tanımlıdır. Herhangi bir zamanda sadece bir buffer aktif durumdadır.



Şekil 4.6. Üç durumlu bufferlar ile oluşturulmuş veri yolunun bir bitlik kısmı

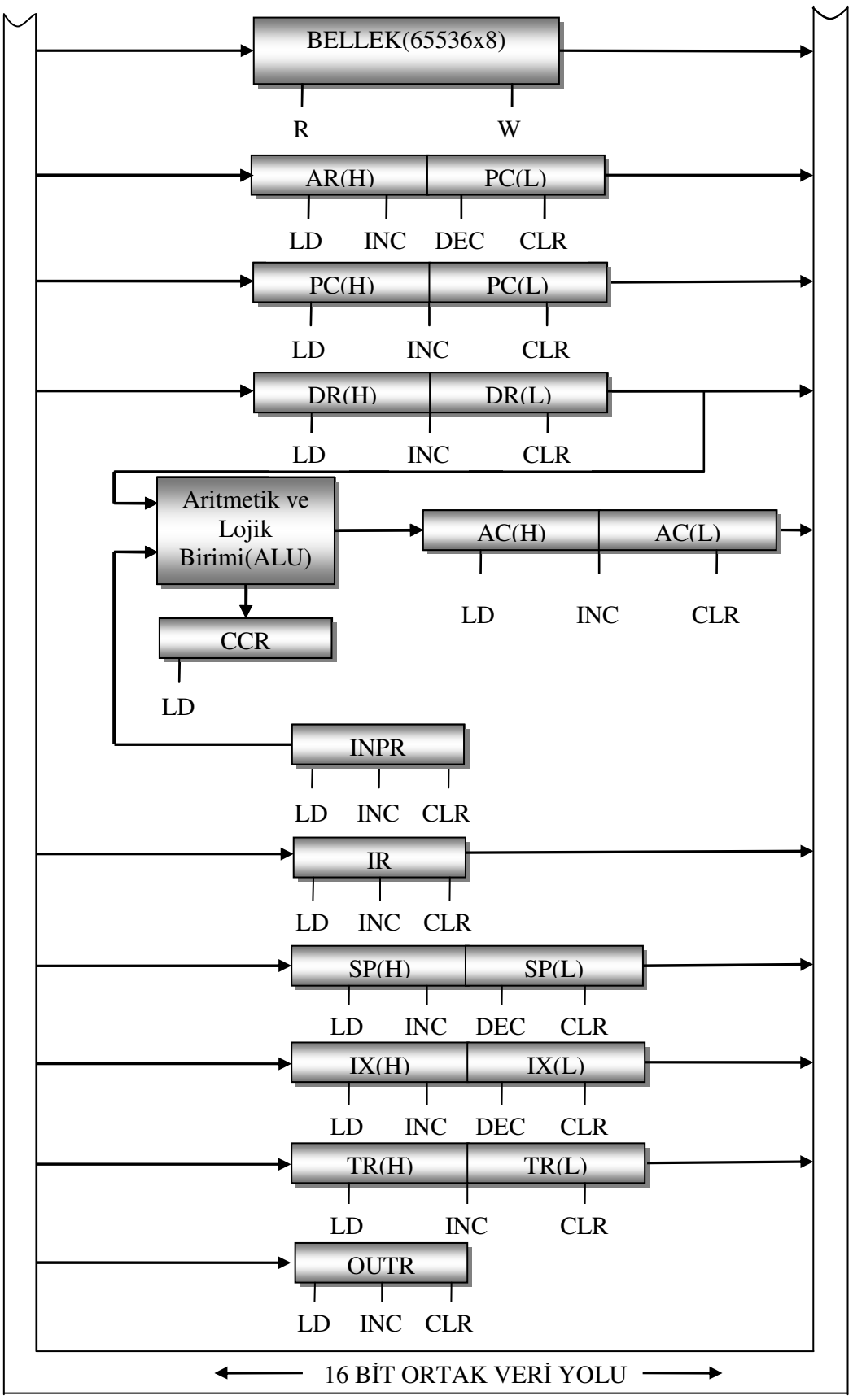
Ortak veri yoluna aynı anda sadece bir kaydedici veya belleğin bilgilerini veri yoluna aktarabilmesine olanak sağlayan yapı 4x16'lık iki adet kod çözücülerle tasarlanarak elde edilir. Birinci kod çözücü yüksek öncelikli bitlerin ortak veri yoluna aktarılmasına izin verirken, diğer kod çözücü ise düşük öncelikli bitlerin ortak veri yoluna aktarılmasına izin verir. Tablo 4.4 ve Tablo 4.5'de kod çözücülerin girişlerinin hangi durumlarında hangi elemanın veri yoluna bilgilerini bırakacağı bilgisi verilmektedir.

Komut kaydedicisi(IR) elemanı 8 bitlik olduğundan yüksek değerli veri yoluna erişimi bulunmamaktadır. Yüksek değerli veri yolu seçicinin girişlerine bu elemanın düşük değerli yola erişebilmesini sağlayan değer geldiğinde yola lojik 0 bilgisi verilir. Bunun yapılmasındaki neden, bu eleman veri yolunun sadece düşük değerli kısmını kullandıklarından o anda veri yolunun yüksek değerli kısmında daha önceki işlemlerden kalma veri olabileceğinden yola lojik 0 bilgisi verilir. Bir başka anlatımla, bu elemanlar sanki yüksek değerli 8 biti lojik 0 olan 16 bitlik eleman gibi davranırlar.

Bellek elemanı da 8 bitlik olmasına rağmen hem yüksek değerlikli hem de düşük değerlikli veri yoluna erişimi vardır. Çünkü bellek elemanı bazı durumlarda kaydedicilerin yüksek değerli kısmına bazen de düşük değerlikli kısmına bilgi göndermek durumunda olduğundan ortak veri yolunun hem düşük hem de yüksek değerlikli kısmına erişmek zorundadır. Bu bilgisayardaki bellek ve yazaçlar arasındaki ortak veri yolu sistemine bağlanmaları Şekil 4.7'de görülmektedir.

Şekil 4.7'de görüleceği üzere 11 adet kaydedici ve bellek ortak veri yoluna bağlanmaktadırlar. Hangi elemanın hangi durumda veri yolunu kullanacağı 4x16 lık kod çözücülerin girişlerine gelen veri karar verir. Örneğin, kaydedicilerin düşük anlamlı 8 bitinin hangisinin yola serbest kalacağına karar veren eleman olan kod çözücünün girişleri olan DECL03, DECL02, DECL01 ve DECL01 girişlerine 1000 bilgisi geldiğinde adres kaydedicinin düşük anlamlı sekiz biti yola serbest bırakılır. Aynı şekilde yüksek anlamlı 8 bitin hangisinin yola serbest bırakılacağına karar veren eleman olan 4x16 lık kod çözücünün girişleri olan DECH03, DECH02, DECH01 ve DECH00 girişlerine de 1000 geldiğinde adres kaydedicinin yüksek anlamlı sekiz biti yola serbest bırakılır.

Bu bilgi yolda serbest kaldıktan sonra kaydedicilerden herhangi birisinin bu bilgiyi alabilmesi için bu bilgisayarda tasarım gereği kontrol girişi "LDAXXX" olan kaydedici yoldaki bilgiyi alacaktır. Bellek elemanı ise bu bilgiyi alabilmesi için tasarımda kullanılan belleğin "R\W" kontrol girişi aktif edilmelidir. Tablo 4.4 ve Tablo 4.5'de kod çözücülerin girişlerine bağlı olarak hangi birimin verilerini yola serbest bırakılacağı gösterilmiştir.



Şekil 4.7. Temel bilgisayar kaydedicilerinin ve belleğin ortak veri yoluna bağlanması[22]

Tablo 4.4. Düşük anlamlı veri yolunun tahsisi

Kod Çözücü Girişi	Yolu Kullanan Birim
0001	Yığın göstergeci(SP)
0010	Akümülatör(AC)
0011	Program sayıcı(PC)
0100	Komut kaydedici(IR)
0101	Veri kaydedici(DR)
0110	İndis kaydedici(IX)
0111	Geçici Kaydedici(TR)
1000	Adres Kaydedici(AR)
1001	Bellek(MEM)
1010	Indis ve Göreceli Adresleme
1100	Durum Kod Kaydedici(CCR)

Tablo 4.5. Yüksek anlamlı veri yolunun tahsisi

Kod Çözücü Girişi	Yolu Kullanan Birim
0001	Yığın göstergeci(SP)
0010	Akümülatör(AC)
0011	Program sayıcı(PC)
0101	Veri kaydedici(DR)
0110	İndis kaydedici(IX)
0111	Geçici Kaydedici(TR)
1000	Adres Kaydedici(AR)
1001	Bellek(MEM)
1010	Indis ve Göreceli Adresleme

Tablo 4.4 ve Tablo 4.4'e dikkatlice bakılacak olursa her iki kod çözücünün girişlerine lojik "1010" bilgisi geldiğinde, kodu çözülen komutun adresleme modu eğer indis veya göreceli adresleme modu ise ileride anlatılacağı üzere bu modlu komutların etkin adresinin hesaplanabilmesi için tasarlanan birim tarafından elde edilen etkin adres veri yoluna bırakılır. Bırakılan bu bilgiyi adres kaydedici "LDAXXX" kontrol girişini aktif ederek komutu gerekli adresi alır.

Yine tablolardan görüleceği üzere düşük anlamlı yol tahsisi yapan kod çözücünün girişlerine lojik “1100” bilgisi geldiğinde durum kod kaydedicisi yolu kullanmaktadır. Bu özellikle kesme meydana geldiğinde durum kod kaydedicisindeki veri belleğe atılırken veri yolunu kullanma gereksiniminden doğmaktadır.

Şekil 4.7’de görüleceği üzere, aritmetik ve mantık biriminin(ALU) veri yoluna bağlantısı direkt olarak bulunmamaktadır. Bu birim veri kaydedicisi, akümülatör ve giriş kaydedicinden gelen verileri değerlendirerek akümülatöre gönderir. Bu şekilde dolaylı olarak veri yoluna erişmesi sağlanmış olur.

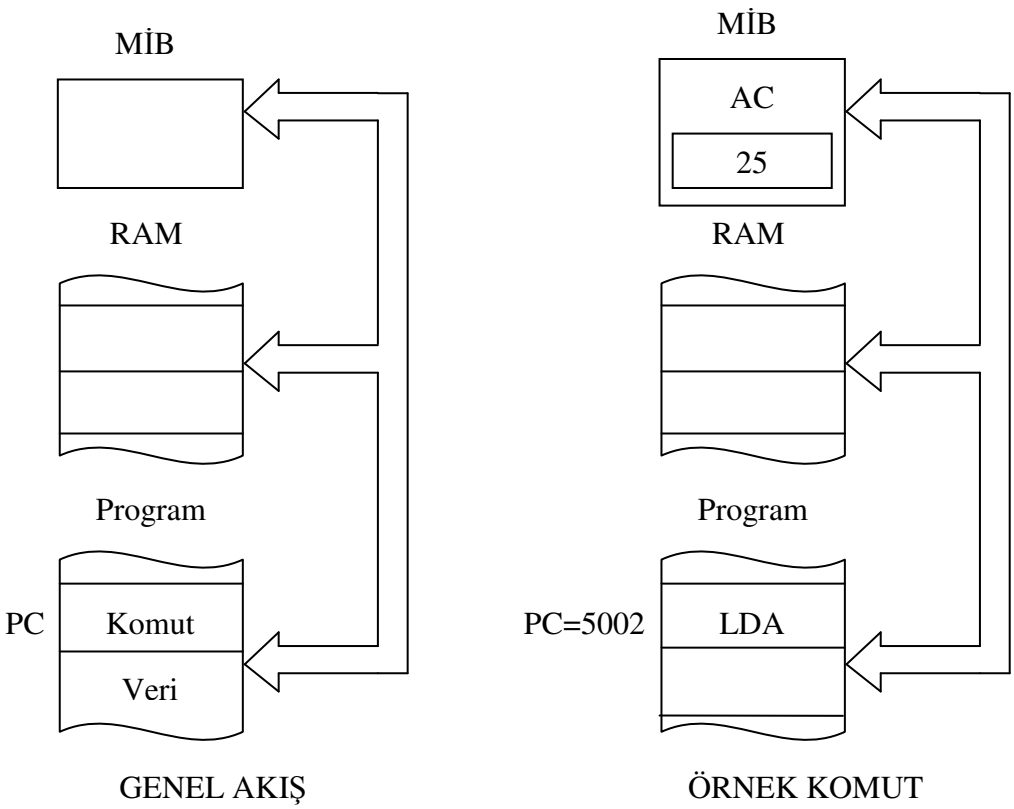
Çıkış kaydedicisi sadece veri yolundan bilgi alabilirken, giriş kaydedicisinin ise ALU’da olduğu gibi direkt olarak veri yoluyla bağlantısı yoktur. Üzerindeki veriyi ALU’ya ileterek gerekli işlemleri yapıldıktan sonra akümülatöre gönderilir.

4.4. Adresleme Modları

Programları oluşturan kodlar ve veriler hafızaya yüklendikten sonra işlemci tarafından satır-satır icra edilirler. Ayrıca merkezi işlemci birimi(CPU) tüm giriş çıkış işlemlerini de hafızaya erişerek yapar. Bazen hafızadan doğrudan bir kod ya da veri alır ve işler. Bazen hafızaya bir veri gönderdiğinizde birde bakmışsınız bu bir yazıcıdan belge olarak çıkmış vs. İşte bilgisayarın donanım ve yazılım düzeyinde yaptığı bunca çeşitli iş için CPU hafızaya değişik yollardan erişme ihtiyacı duyar. Sizlerde programlarınızı yazarken CPU’nun hafızaya nasıl erişeceğini yazdığınız kodlarla belirtmek zorundasınız. İşte bunları belirtmenin yolu adresleme modlarını kullanmaktan geçiyor[30]. Yukarıda bahsedildiği üzere tasarlanan bu bilgisayarda ticari işlemcilerde ortak ve çok sık kullanılan adresleme modlarından altı tanesi kullanıldı.

4.4.1. Derhal adresleme modu

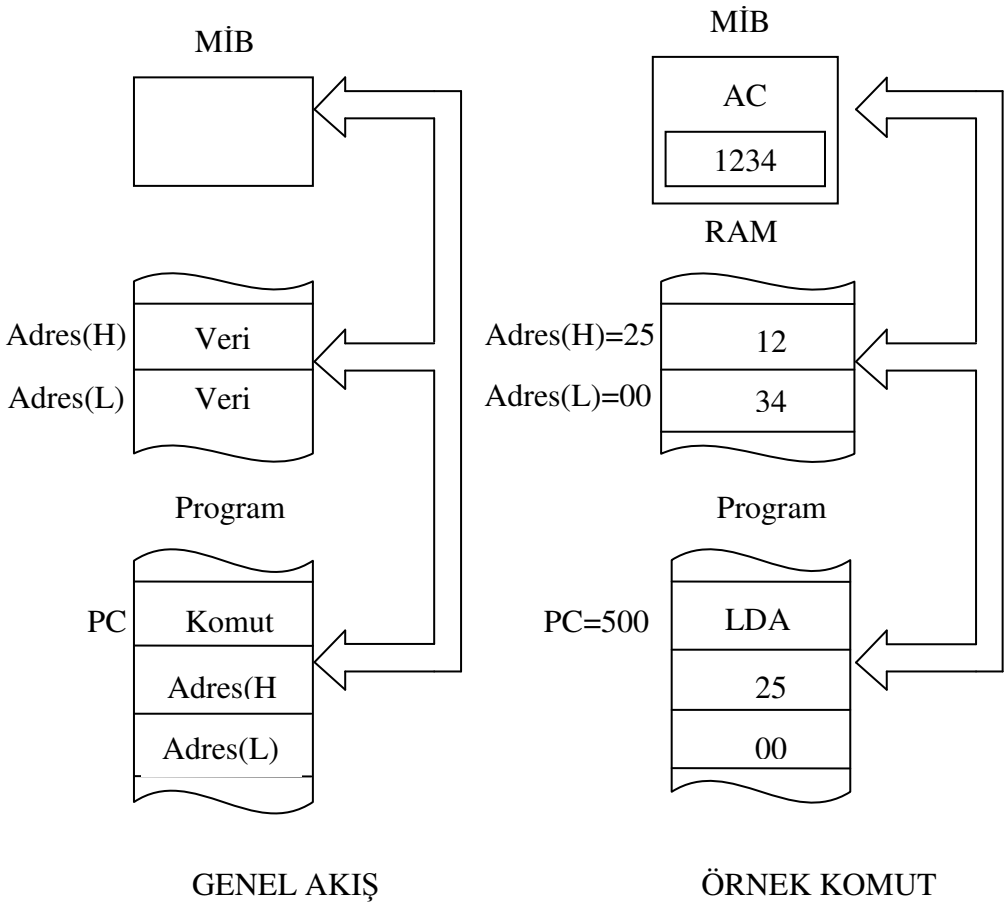
Bu modda veri komutun içinde verilmiştir[30]. Bu moddaki komutlar kaydedicilere başlangıç değerlerin verilmesi ve sabit değerler yazılması açısından uygundur. Şekil 4.8’de bu modun çalışması görülmektedir.



Şekil 4.8. Derhal adresleme modunun çalışmasına bir örnek

4.4.2. Direk adresleme modu

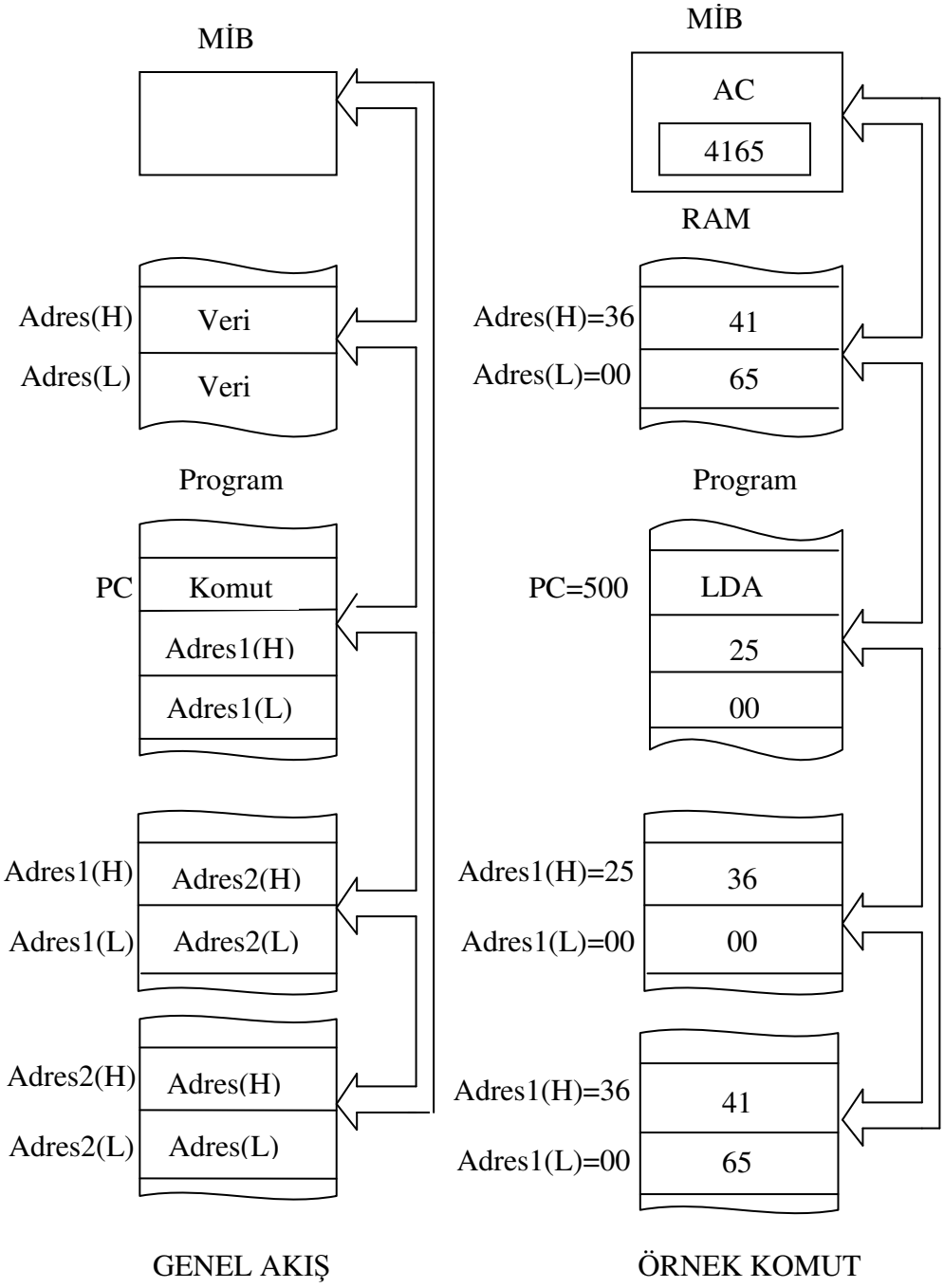
Bu modda etkin adres komutun içinde yer almaktadır. Veri bellekte, adresi komutun içindedir. Dallanma tipi bir komutta ise komutun içinde yer alan adres dallanılacak olan adresi gösterir[30].



Şekil 4.9. Direk adresleme modunun çalışmasına bir örnek

4.4.3. Dolaylı adresleme modu

Bu mod ise, buyruğun içindeki adres verinin bellekteki gerçek adresinin bulunduğu yerin adresidir. Denetim mekanizması, buyruğu bellekten alır getirir ve burada yazılı adresi alarak bu adreste yer alan veriyi okur[30].

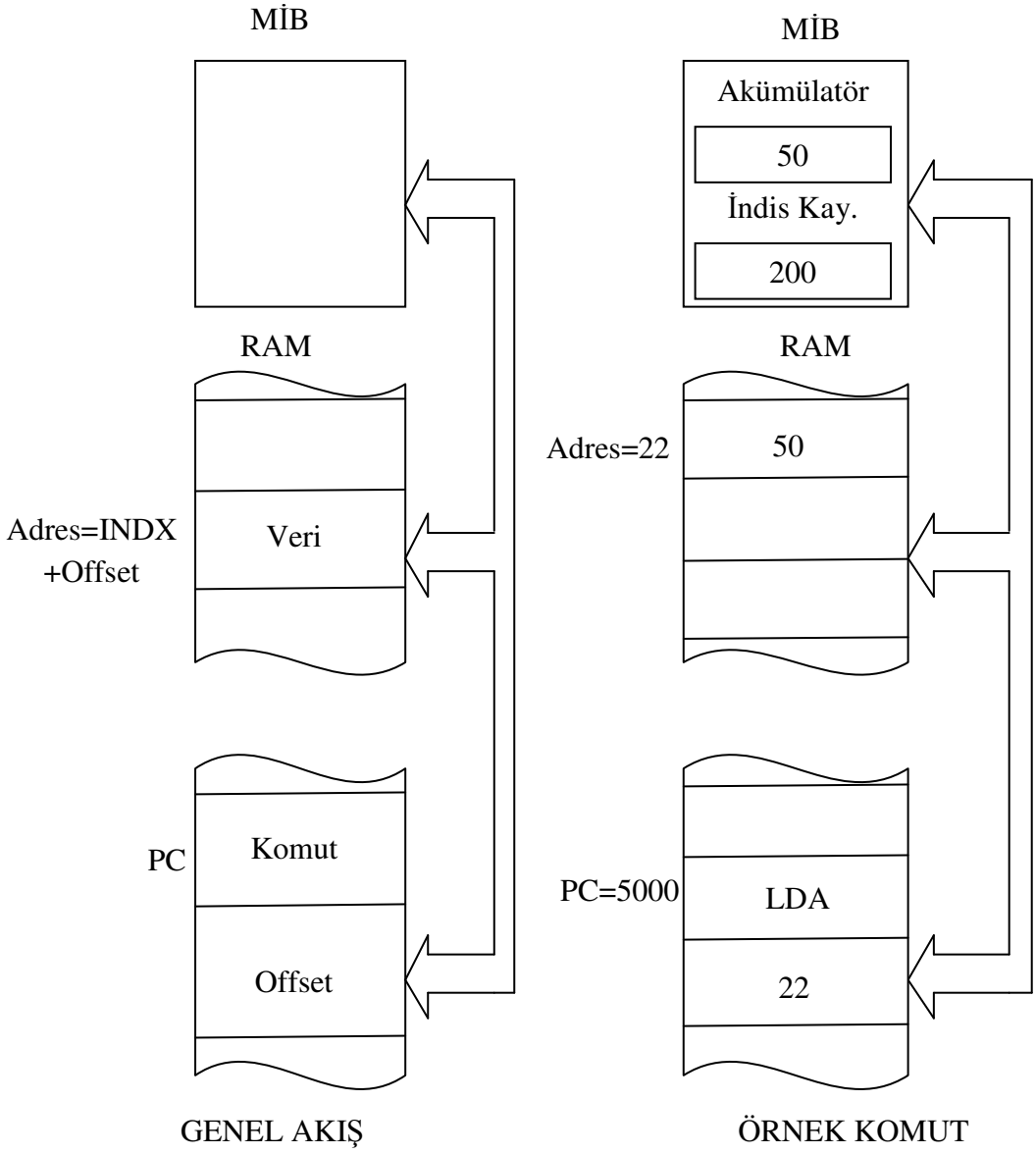


Şekil 4.10. Dolaylı adresleme modunun işleyişi

4.4.4. İndis adresleme modu

İndis kaydedicisinin içeriği buyruğun adres değerine eklenir. İndis kaydedicisi özel bir mikro işlemci birimi kaydedicisi olup bir indis değeri içerir. Buyruk içindeki adres değeri çizelgenin başlangıç adresidir[30]. Dizideki her veri bellekle başlangıç adresine göreceli bir konumda bulunur. Başlangıç adresi ile verinin adresi arasındaki fark indis kaydedicisinde bulunan değerdir. Dizideki her veri aynı komut ile erişilir.

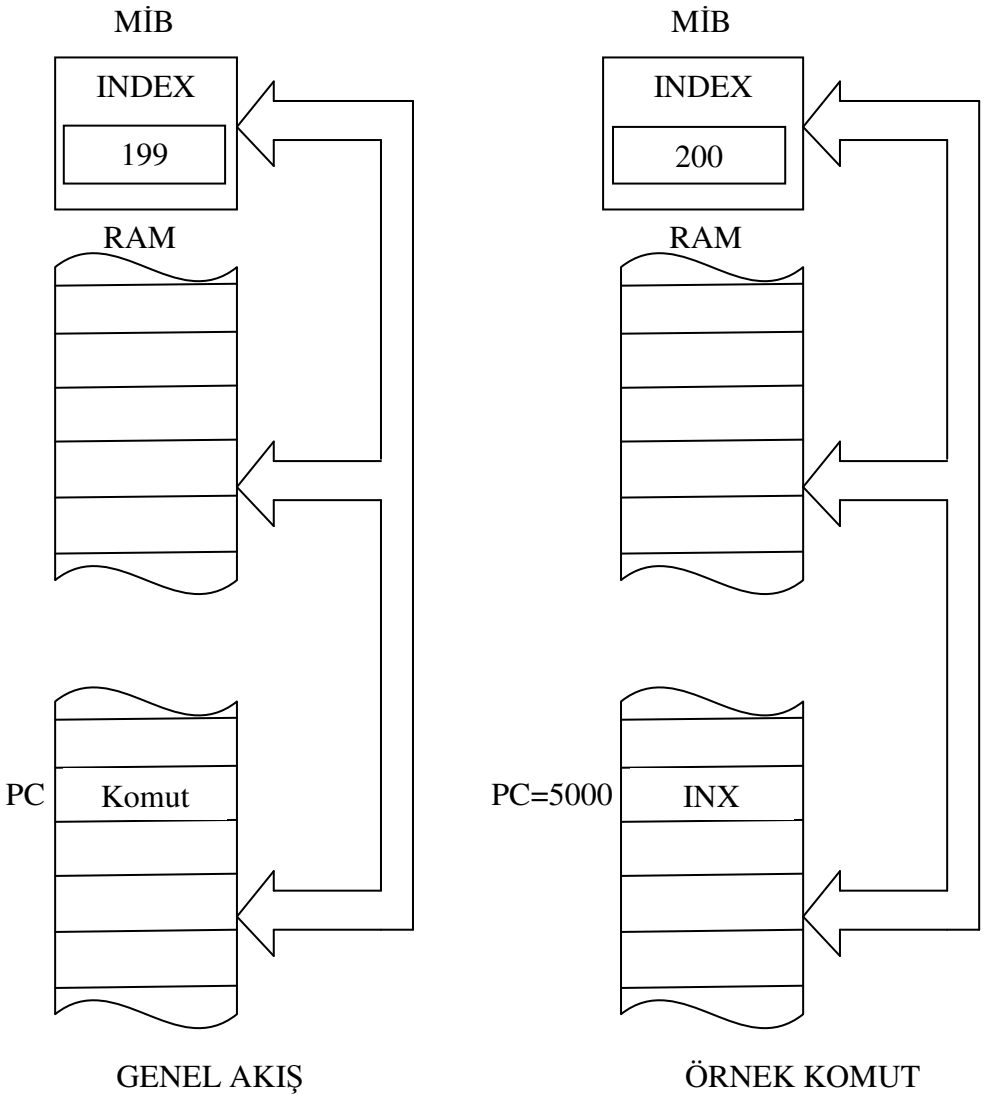
İndis kaydedicisinin değeri ardışık veriler ulaşmak için kullanılabilir. İndis adresleme metodunun işleyişi Şekil 4.11'de gösterilmektedir.



Şekil 4.11. İndisli adresleme modunun işleyişi

4.4.5. Doğal adresleme modu

Mikroişlemcilerde, özellikle kaydediciler arasında veri transferi için uygulanan hızlı bir adresleme türüdür[30]. Kaydediciler ile akümülatör ve bellek arasında da aynı uygulama yapılır. Bu moddaki komutlar için program akışı Şekil 4.12'de gösterilmiştir.

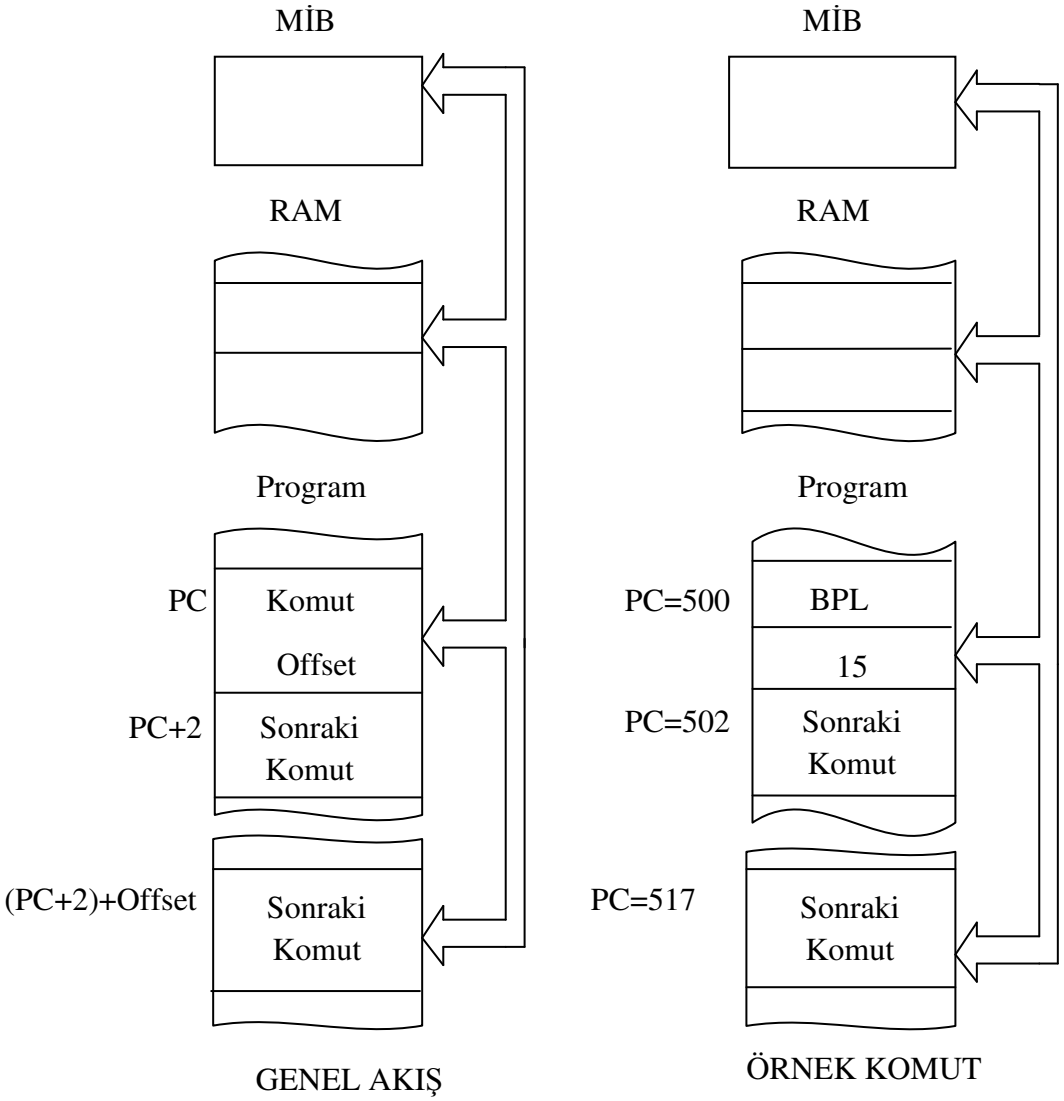


Şekil 4.12. Doğal adresleme metodunun işleyişi

4.4.6. Göreceli adresleme modu

Bu modda program sayıcının içeriği buyruğun adres kısmına eklenerek etkin adres elde edilir[30]. Komutun içinde yer alan veri bu bilgisayarda 8 bitlik bir işaretli sayıdan oluşmaktadır. 8 bitlik işaretli sayıyı düşünecek olursak; -128 ile +127 arasında değer alacağından bu adresleme modunda elde edilen etkin adres bir sonraki komutu gösteren program sayıcının 128 eksiğine veya 127 fazlasına konumlanmış olacaktır. Bir örnek ile açıklamak gerekirse; Program sayıcının içeriği 800 olsun. Komutun adres kısmında yer alan değerde 32 olduğunu varsayalım. Bu durumda

etkin adres 830+32=862 olup bir sonraki buyruktan 30 bellek birimi uzaklıktadır. Bu mod, eğer dallanılacak yer, komut civarında ise dallanma için kullanılır. Faydası uzun adres yerine buyruğun içine kısa adres yazılmasıdır.

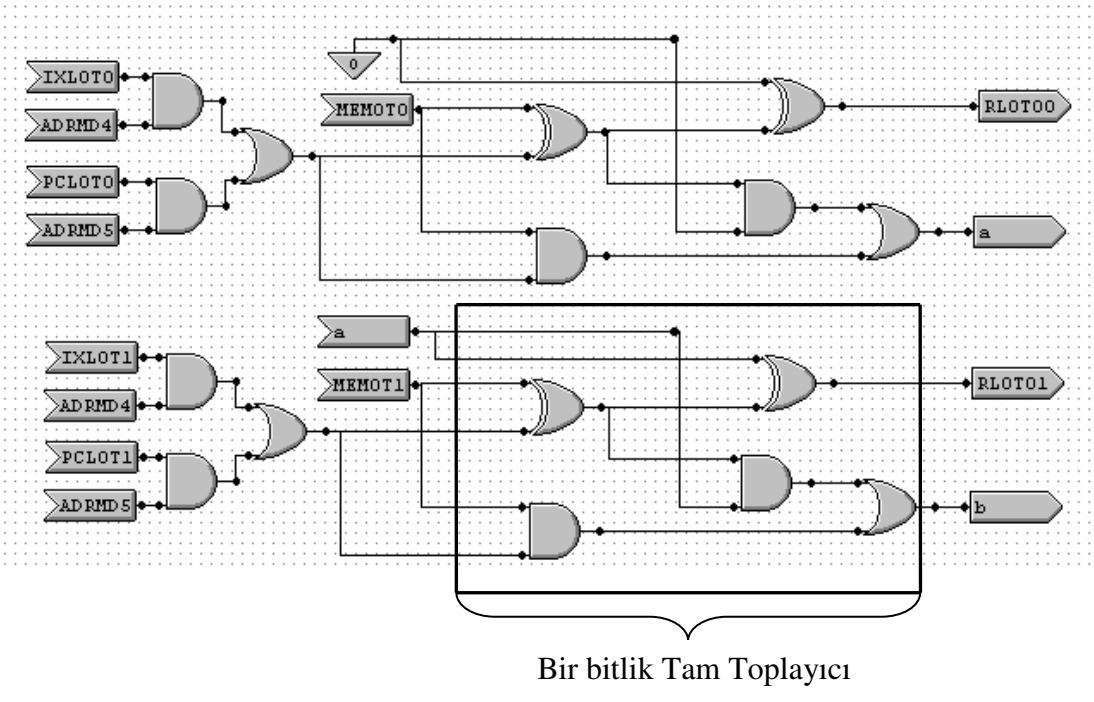


Şekil 4.13. Göreceli adresleme modunun işleyişi

4.5. Göreceli ve İndis Adresleme Modları için Adres Hesaplama Birimi

Adresleme modlarında değinilen göreceli ve indis adresleme modlarına sahip komutlar geldiğinde komuttan sonra gelen bir bytelık veri program sayıcıya veya indeks kaydedicisindeki veri ile toplanması gerektiğine adresleme modları bölümünde değinildi. Bu toplama işlemini gerçekleşebilmesi için elimizde bulunan aritmetik mantık birimini kullanmak gerekir. Ancak bu toplama işlemi yapılmadan

önce akümülatör ve veri kaydedicisindeki veriyi kaybetmemek için bir yığına atmak ve etkin adresi hesapladıktan sonra tekrar yığından bu bilgilerin alınıp tekrar yerlerine konulması gerekir. Bu işlemler bu moda sahip bir komutun icra edilme süresini uzatmaktadır. Bu etkin adresi aritmetik ve mantık biriminde yapmaktan ziyade ayrı bir birim üzerinde hesaplanması bu ekstra süreyi kaldıracaktır. Bunun için tasarlanan birimin iki bitlik kısmı Şekil 4.14'de görülmektedir. Şekildeki ADRMD4 indis adresleme modunu, ADRMD5 ise göreceli adresleme modunu temsil etmektedir. Bu iki modan hangisi aktif ise indis kaydedicisindeki veri veya program sayıcındaki veri ile adres kaydedicisinin bellekte o an göstermiş olduğu offset değeri bit tam toplayıcı devreden geçirilmektedir. Çıkan sonuç yukarıda bahsedildiği üzere 16 bitlik veri yoluna aktarılarak ilgili birim veri yolundan kendisine transfer etmektedir.



Şekil 4.14. Etkin adres Hesaplama biriminin iki bitlik kısmı

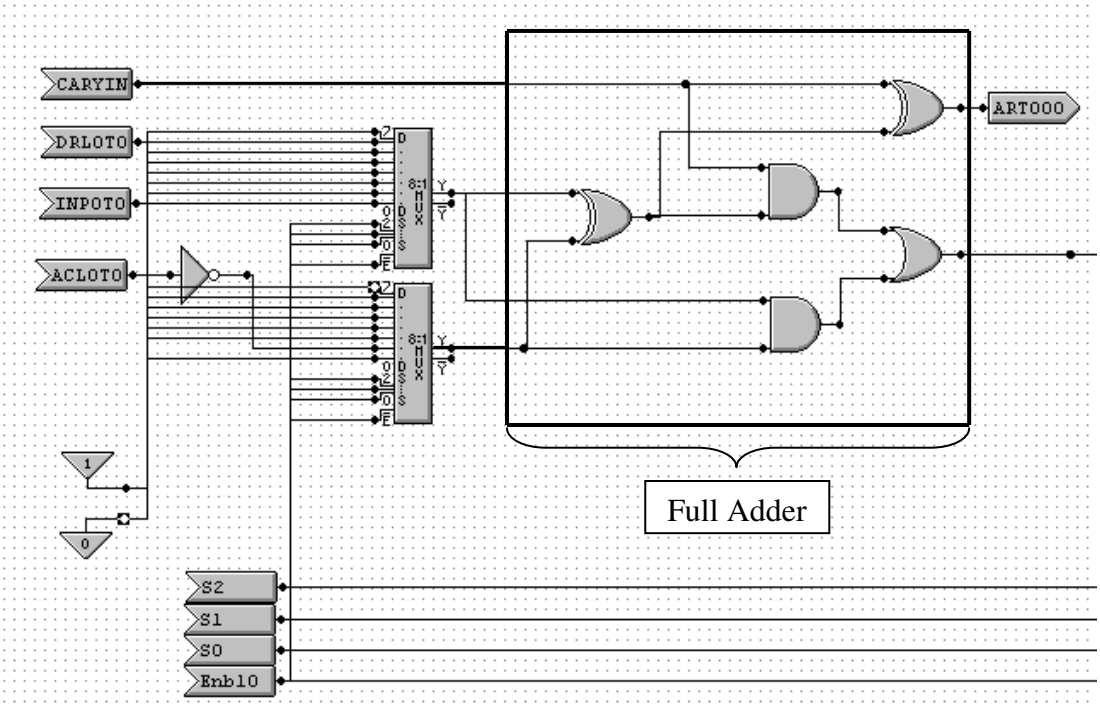
4.6. Aritmetik Mantık Birimi(ALU)

Bir mikroişlemcide bulunan ALU, bütün aritmetik ve mantık işlemlerinin yapıldığı yerdir[28]. Bu bilgisayarda tasarlanan ALU, çarpma işlemi ve giriş kaydedicisinden gelen veri hariç 16 bitlik veriler ile işlem yapabilen bir birim olarak tasarlanmıştır.

Bu bilgisayarda tasarlanan kaydediciler 16 bitlik olduğundan çarpma işlemi 8 bitlik olarak tasarlandı. Tasarlanan ALU dört bölüme ayrılmaktadır. Bunlardan birincisi çarpma ve bölme işlemleri hariç aritmetik işlemlerin yapıldığı bölüm; ikincisi, mantık işlemlerinin yapıldığı bölüm; üçüncü ve dördüncü bölümler ise çarpma ve bölme işlemlerinin yapıldığı bölümlerdir.

4.6.1. Aritmetik işlemler kısmı

MML programında tasarlanan ALU'nun aritmetik işlemlerin yapıldığı bölümde gerçekleşen işlemler Tablo 4.6'da verilmektedir. Şekil 4.15'te ise MML programında gerçekleştirilen aritmetik işlemler bölümünün bir bitlik kısmı görülmektedir.



Şekil 4.15. ALU'nun aritmetik işlemler yapan bölümün bir bitlik kısmı

Tablo 4.6. ALU'nun aritmetik işlem bölümü

Seçim				Girişler		Çıkış	Açıklaması
S_2	S_1	S_0	CARRYIN	Y_0	Y_1	Q	
0	0	0	0	DR	AC	AC←DR+AC	Toplama
0	0	0	1	DR	AC	AC←DR+AC+1	Elde ile Toplama
0	0	1	0	DR	AC	AC←DR+$\overline{AC}$	Borç ile Çıkarma
0	0	1	1	DR	AC	AC←DR+$\overline{AC}$+1	Çıkarma
0	1	0	0	DR	00	AC←DR	DR'in aktarımı
0	1	0	1	DR	00	AC←DR+1	DR 1 arttırma
0	1	1	0	00	AC	AC←AC	AC'nin aktarımı
0	1	1	1	00	AC	AC←AC+1	AC'yi 1 artırma
1	0	0	0	DR	11	AC←DR-1	DR'yi 1 azaltma
1	0	0	1	DR	11	AC←DR	DR'nin aktarımı
1	0	1	0	11	AC	AC←AC-1	AC'yi 1 azaltma
1	0	1	1	11	AC	AC←AC	AC'nin aktarımı
1	1	0	0	DR	11	AC←DR-1	DR'yi 1 azaltma
1	1	0	1	DR	11	AC←DR	DR'nin aktarımı
1	1	1	0	INPR	00	AC←INP	Giriş Kaydedicisinin aktarımı
1	1	1	1	INPR	00	AC←INP+1	Giriş Kaydedicisini 1 artırma

Şekil 4.15'te bir bitlik kısmı gösterilen aritmetik işlemler bölümü 16 adet tam toplayıcı ve işlemlerin seçimi için 3 seçim girişine sahip 32 adet MUX'dan oluşmaktadır. Bu kısım Tablo 4.6'da verilen 16 adet işlemi yerine getirebilmektedir. Örneğin; seçim uçları $S_2S_1S_0$=001 ve elde girişi de CARRYIN=1 olduğunda veri kaydedicisindeki değerden akümülatördeki veri çıkartılır. CARYIN=0 geldiğinde ise veri kaydedicisindeki değerden akümülatördeki değer borçlu olarak çıkartılır. Başka bir deyişle DR-AC-1 işlemi gerçekleştirilir. $S_2S_1S_0$=010 ve CARYIN=0 olduğunda

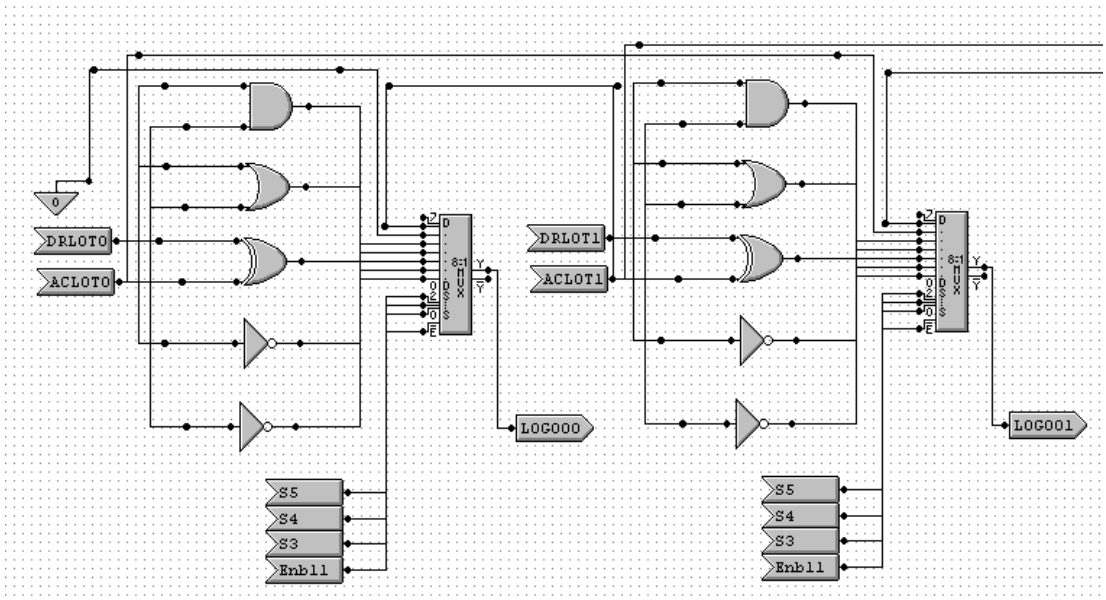
veri kaydedicisindeki veri hiçbir değişikliğe uğramadan aritmetik işlem biriminin çıkışına iletilir. CARYIN=1 olduğunda ise veri kaydedicindeki verinin bir fazlası alınarak çıkışa verilir.

4.6.2. Mantıksal işlemler kısmı

ALU' nun diğer bir bölümü olan mantıksal işlemlerin yapıldığı bölüm ise 6 farklı işlemi yerine getirebilmektedir. Tablo 4.7'da yerine getirilen işlemler verilmiş olup MML programında tasarlanan bu bölümün iki bitlik kısmı Şekil 4.16'da verilmiştir.

Tablo 4.7. ALU'nun mantıksal işlem bölümünde gerçekleştirilen işlemler

S_5	S_4	S_3	Q	Açıklaması
0	0	0	AC←AC∧DR	Lojik "VE" İşlemi
0	0	1	AC←AC∨DR	Lojik "VEYA" İşlemi
0	1	0	AC←AC⊕DR	Lojik "XOR" İşlemi
0	1	1	AC← $\overline{DR}$	DR'in tümleyeni
1	0	0	AC← $\overline{AC}$	AC'in tümleyeni
1	0	1	AC←shlAC	AC'yi sola kaydırma
1	1	0	AC←shrAC	AC'yi sağa kaydırma



Şekil 4.16. Mantıksal işlemlerin yapıldığı bölümün iki bitlik kısmı

ALU'nun bu bölümünde yapılan işlemlerin yerine getirilebilmesi için 16 adet 8x1 lik MUX kullanılmıştır. Örneğin; seçim uçları uçları $S_5S_4S_3$=001 olduğunda veri kaydedicisindeki veri ile akümülatördeki veri lojik "veya" işlemine tabi tutularak çıkışa verilmektedir. $S_5S_4S_3$=101 ise akümülatördeki veri bir bit sola kaydırılır. Şekil 4.16'ya dikkat edilirse ilk MUX elemanının girişine lojik "0" bilgisi verilmektedir.

4.6.3. Çarpma bölümü

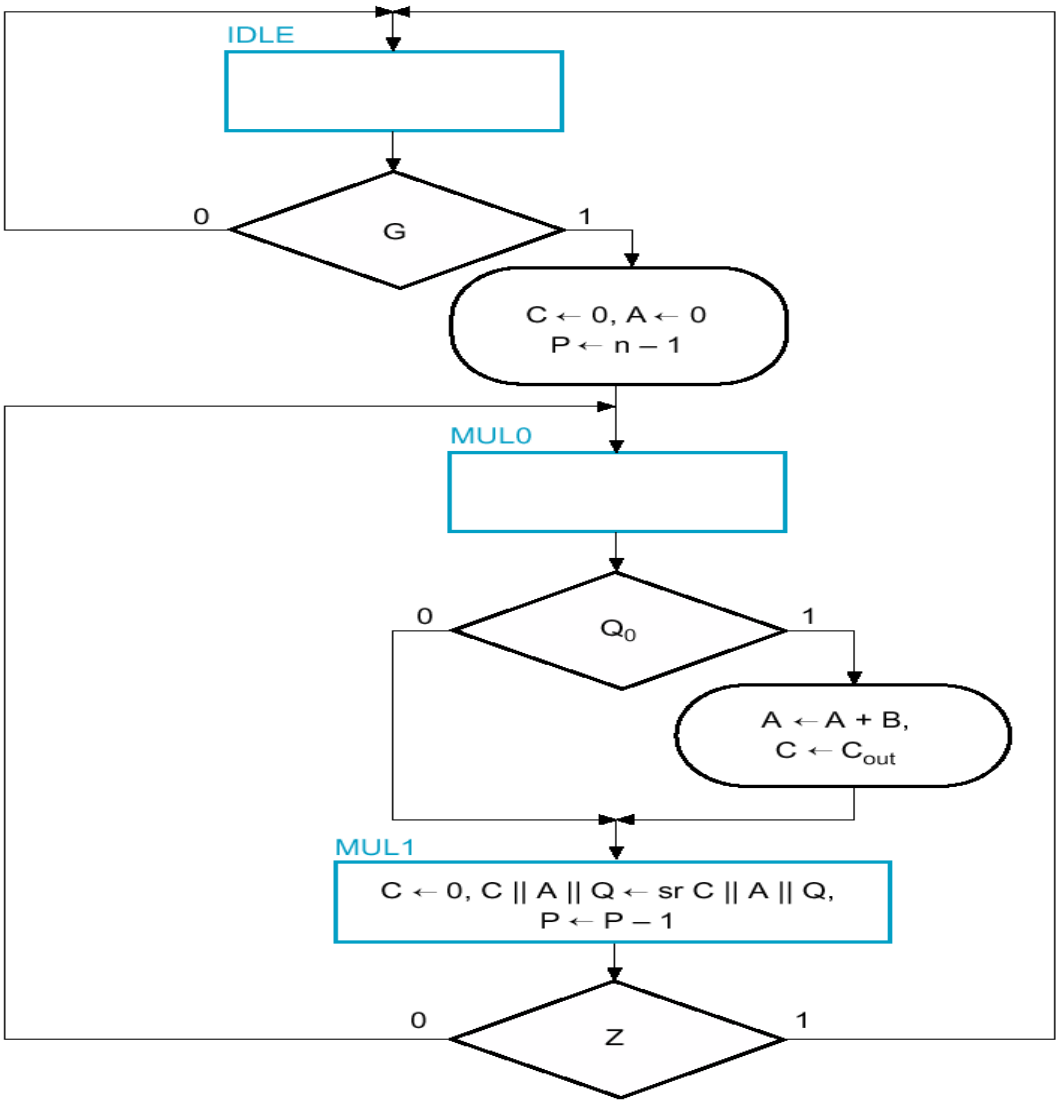
İki adet ikili sayının çarpımı kalem kâğıtla yapılırsa bir dizi ardışık kaydırma ve toplama işlemleriyle gerçekleşir[31]. İşlem çarpmanın en önemsiz bitinden başlayarak sırayla bütün bitlerine bakılarak yapılır. Eğer çarpanın biti lojik "1" ise çarpılan aşağıya kopya edilir. Aksi halde aşağıya sıfır kopya edilir. Her kopya işleminden sola bir kaydırma yapılır. Son bit içinde kaydırma ve yazma yapıldıktan sonra bunlar toplanarak çarpım elde edilmiş olur. Çarpımın işareti çarpanla çarpılanın işaretlerinden bulunur. İşaretler aynı ise çarpım pozitif, farklı ise negatiftir.

Sayısal bir bilgisayarda çarpma işlemi yapılacağı zaman işlemi biraz değiştirmek uygun olur. Kalem kâğıtla yapılan örnekte, bütün ikili sayıları kaydedicilere yazmak yerine sadece iki sayının toplamını almak daha uygun olur.[31] Daha sonra her sayı bu toplam eklenerek sonuca gidilir. İkincisi çarpımı sola kaydırmak yerine, kısmi toplam sağa kaydırılır. Böylece çarpılan daima yerinde kalmış olur. Üçüncüsü ise doğal olarak çarpanın biti "0" ise sıfırları toplamaya gerek yoktur.

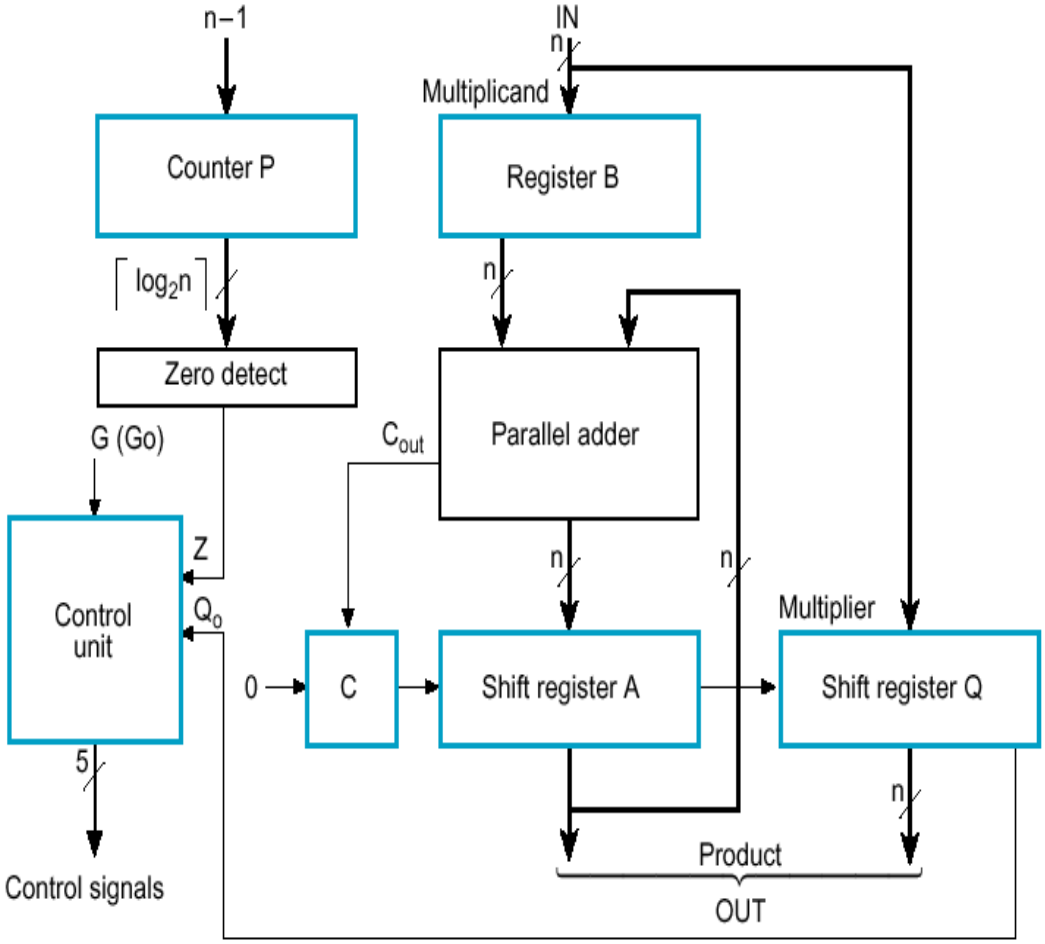
Çarpım için donanım Şekil 4.18'de gösterilmiştir. Çarpan Q kaydedicisindedir. Sıra sayıcıya çarpandaki bit sayısı daha önceden yazılır. Her kısmi toplam eldesinde bu sayı 1 azaltılır. Sayaç sıfır olunca çarpma işlemi bitmiştir, işlem durur.

İşlemin başında çarpılan B kaydedicisinde, çarpan Q dadır. A ve B bin toplamı kısmi çarpımı oluşturur. Kısmi çarpım CA kaydedicisine aktarılır. Kısmin çarpım ve çarpan sağa kaydırılır. Bu işlem shrCAQ ile gösterilmektedir. Anın en önemsiz biti Q nun en önemli bitine kaydırılır. C biti A nın en önemli bitine ve C ye de "0" kaydırılır. Kaydırma işleminden sonra kısmi çarpımın bir biti Q ya kaymış ve çarpanın bitlerini bir bit sağa doğru itmiştir. Bu durumda çarpanın en sağ biti Q_0 a

kayar. Kayma işleminden sonra bu bitin “0” mı “1” mi olduğuna bakılır. Eğer bu bit “1” ise çarpılan sayı A daki kısmi çarpıma eklenir. Eğer “0” ise hiç bir şey yapılmaz. Her iki durumdan sonra CAQ yazacı bir bit sağa kaydırılır. Sıra sayıcı bir azaltılır, sonra sıfır olup olmadığına bakılır. Eğer “0” değilse işlem tekrarlanır. Eğer SC=0 ise işlem durur. Kısmi çarpan her defasında Q nun içine bir bit bir bit girer ve çarpanın en önemsiz biti kaybolur. En sonunda çarpan kaybolur. Sonuç A ve Q dadır. Bu kısımda anlatılan işlemler için akış şeması Şekil 4.17’de görülmektedir.



Şekil 4.17. Çarpma işlemi için akış şaması

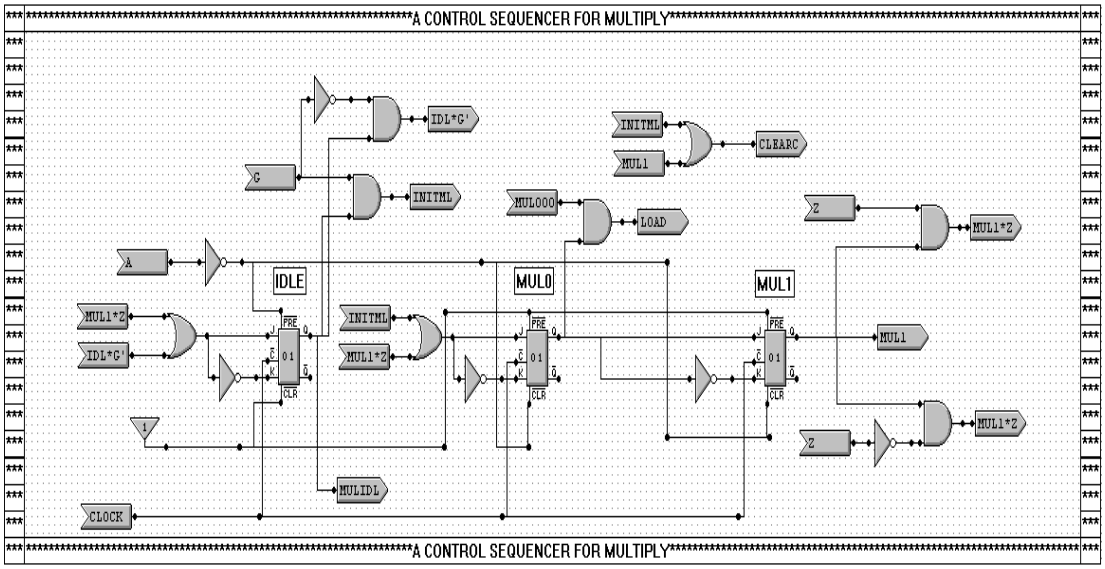


Şekil 4.18. Çarpma işlemi için donanım

Çarpma işlemi anlatılan işlemleri kontrol edecek bir yapıya ihtiyaç vardır. Şekil 4.17'de verilen akış şemasına bakarak çarpma işleminde kullanılan sıra sayıcı ve kaydediciler için kontrol işaretlerini ve bu kontrol işaretlerinin MML programında karşılıklarını belirleyelim. Burada IDLE etiketiyle belirlenen kutucuk çarpma işlemi başlangıç konumunu teşkil etmektedir. Başka bir deyişle, çarpan ve çarpılanın değerlerin girilmesi aşamasıdır. Eğer bu aşamadayken çarpma işlemi başlama sinyali olan "G" aktif olursa çarpma kontrol devresi MUL0 durumuna geçer. Bu durumdayken C flip-flopunun ve A kaydedicisinin "0" ile ve sıra sayıcının çarpma işlemi kaç bitlik ise bit sayısı ile yüklenmesi işlemleri gerçekleşir. "G" aktif değilse çarpma kontrol devresi "IDLE" konumunda durmaya devam edecektir. Kontrol devresi MUL0 durumundayken çarpanın bulunduğu Q kaydedicisinin en düşük anlamlı bitine bakarak ya birazdan bahsedeceğimiz işlemleri gerçekleştirecektir ya da hiçbir işlem yapmadan MUL1 durumuna geçecektir. Eğer Q'nun en düşük anlamlı

biti, yani Q_0 "1" ise A kaydedicisi ile çarpılan kaydedicisi yani B kaydedicisi toplama devresine gönderilerek toplama işlemi gerçekleştirilerek sonuç tekrar A kaydedicisine yazılacaktır. Ve bu toplama işleminden çıkacak olan çıkış elde biti "C_{out}" C flip-flopuna yazılacaktır. Q_0 "0" ise hiçbir işlem yapmadan MUL1 durumuna geçecektir. MUL1 durumundayken hiçbir koşul bulunmaksızın kısmi çarpım bir bit sağa kaydırılarak P sıra sayıcı bir azaltılır. Daha sonra sıra sayıcın sıfır olup olmadığına bakılarak, eğer "0" ise çarpma işlemi bitirilir değilse, kontrol devresi MUL0 durumuna dallanarak yukarıda bahsedilen işlemleri sıra sayıcı "0" olana kadar devam eder.

Çarpma kontrol devresinde bahsedilen işlemlerde IDLE, MUL0 ve MUL1 olmak üzere üç durumdan bahsedildi. Bu üç durumu 3 adet flip-flop kullanarak ifade edebiliriz. Bir anda sadece bir flip-flop yani bir durum aktif olması sağlanmış olur. İşte anlatılan bu işlemleri kontrol işaretleriyle ifade olursak karşımıza Tablo 4.8 çıkacaktır. Bu kontrol devresinin MML programında gerçeklenmiş hali Şekil 4.19'da görülmektedir. Tablo 4.8 incelenirken Şekil 4.19 ile beraber incelenmesi tablonun okunabilirliğini artıracaktır.



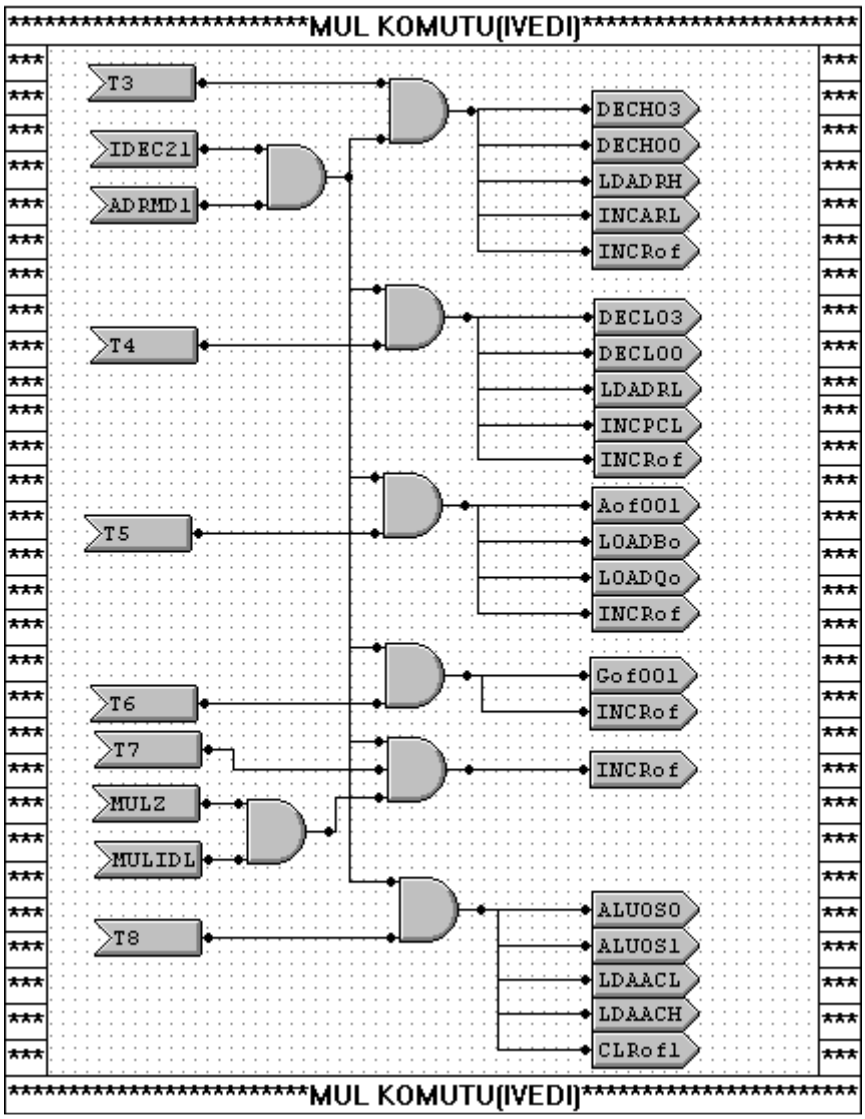
Şekil 4.19. Çarpma işlemine ait kontrol devresinin MML programında gerçeklenmesi

Tablo 4.8. Çarpma işlemi için kontrol işaretleri

Blok Diyagram Modülü	MikroİŞlem	Kontrol İşareti İsmi	MML'deki Kontrol İfadesi
A kaydedicisi	A←0	Initialize	MULIDL*G
	A←A+B	Load	MULO00*MUL00
	C‖A‖Q←shrC‖A‖Q	Shift Right	MUL1
B kaydedicisi	B←IN	Load_B	LOADB
C Flip-Flopu	C←0	Clear_C	MULIDL*G+MUL1
	$C \leftarrow C_{out}$	Load	MULO00*MUL00
Q kaydedicisi	Q←IN	LoadQ	LOADQ
	C‖A‖Q←shrC‖A‖Q	Shift Right	MUL1
P sıra sayıcı	P←n-1	Initialize	MULIDL*G
	P←P-1	Shift Right	MUL1

Çarpma işlemi için kontrol devresi hazırlandıktan sonra sıra çarpma için gerekli olan mikro işlem adımlarını oluşturmaya geldi. Bu mikro işlem adımlarını MML programından alınmış ekran görüntüsü olan Şekil 4.20 üzerinde açıklayalım.

İlk iki adımda çarpılacak olan değer bellekten alınarak veri kaydedicisine transfer ediliyor. Üçüncü adım, yani T5 zaman diliminde ilk önce flip-flopları başlangıç durumunda olması sağlanıyor. Yani IDLE durumu aktif, diğer durumlar pasif olabilmesi için kontrol devresine bir "A" sinyali gönderiliyor. Zaten bu sinyal geldikten sonra çarpan ve çarpılana değerleri yükleniyor. Burada şu belirtilmelidir ki MML programında çarpma devresi tasarlanırken işaretli sayıların çarpımı göz önüne alındı. Bu durumdan dolayı çarpma işlemi başlamadan önce sayılar negatif ise 2'ye tümleyeni alınır o şekilde çarpma devresine sokulur. T6 zaman diliminde çarpma kontrol devresine başlama sinyali "G" gönderilerek yukarıda anlatılan işlemler gerçekleşmeye başlıyor. T7 zaman diliminde sıra sayıcı ve IDLE flip-flopunun çıkışlarının "1" olması durumunda çarpma işleminin bitmiş olduğu anlaşılıp sonuç akümülatöre atılıyor. Sonuç akümülatöre atmadan önce eğer çarpan ile çarpılanı işaretleri farklı ise 2'ye tümleyeni alınıp o şekilde akümülatöre atılır değilse sonuç aynen akümülatöre atılır.



Şekil 4.20. Çarpma işlemi için mikro işlemler

4.6.4. Bölme kısmı

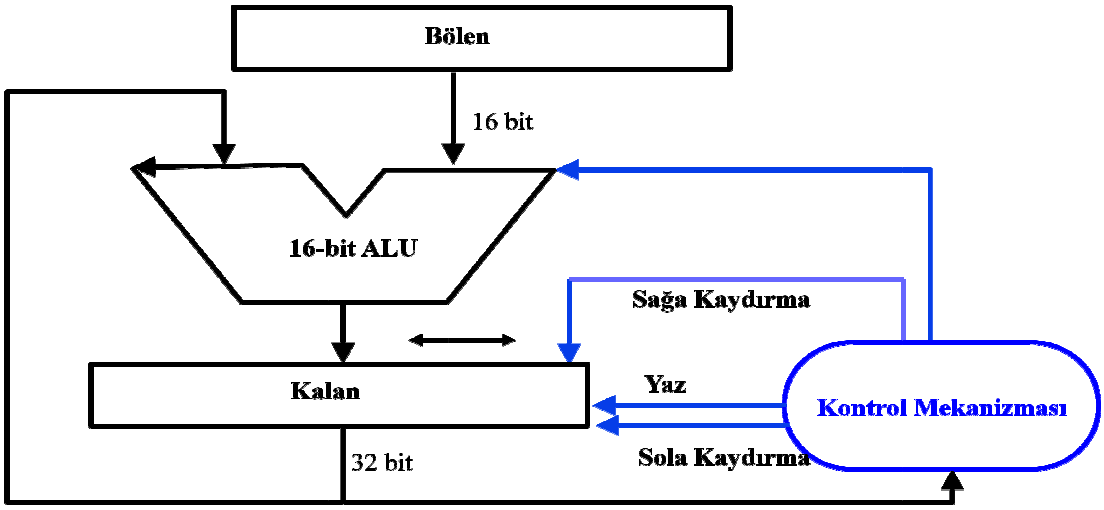
İki adet ikili sayının bölme işlemini kağıt ve kalemle yapmak, bir dizi karşılaştırma, kaydırma ve çıkarma işlemlerini gerektirir[31]. İkili bölme ondalık bölmeden daha basittir, işleme giren sayıların basamaklarında sadece “0” ve “1” vardır. Ayrıca bölenin kalan içinde kaç defa olduğunu bulmak kolaydır. Bölme işlemi Şekil 4.21’de sayısal olarak gösterilmektedir. Örneğin, bölüm 5 bit, bölünen 10 bit olsun. Bölünenin en anlamlı 5 biti bölen ile karşılaştırılır. Bu 5 bit bölenden küçük olduğunda, bölünenin önemli bitlerinden altı tanesi alınır. 6 bit bölenden büyük olduğundan bölüm hanesine “1” yazılır. Bölümün yazılmasında “1” bitini bölünenin

altıncı biti hizasına yazarak hangi bitin kalan olarak gönderileceği daha kolay bulunur. Bölen 1 bit sağa kaydırılarak yazılır ve bölünen sayıdan çıkartılır. Eğer kısmi kalan bölenden küçük ise bölüme "0" yazılır ve bölen tekrar sağa kaydırılır. Her durumda bölen sağa kaydırılır hem de kalan elde edilir.

Bölen:	11010	Bölüm=C
M=10001	0111000000	Bölünen=Q
	01110	Q<M nin 5 biti, bölünenin 5 biti
	011100	Q≥M nin 6 biti
	-10001	M yi mantıksal sağa kaydır ve çıkart; Bölüme "1" yaz
	-010110	Kalan ≥M nin 7 biti
	--10001	M yi mantıksal sağa kaydır ve çıkart; Bölüme "1" yaz
	--001010	Kalan<M; Bölüme "0" yaz ve M yi mantıksal sağa kaydır
	---010100	Kalan≥M
	----10001	M yi mantıksal sağa kaydır ve çıkart; Bölüme "1" yaz
	----000110	Kalan<M; Bölüme "0" yaz
	-----00110	Son kalan

Şekil 4.21. İkili bölme işlemi için örnek

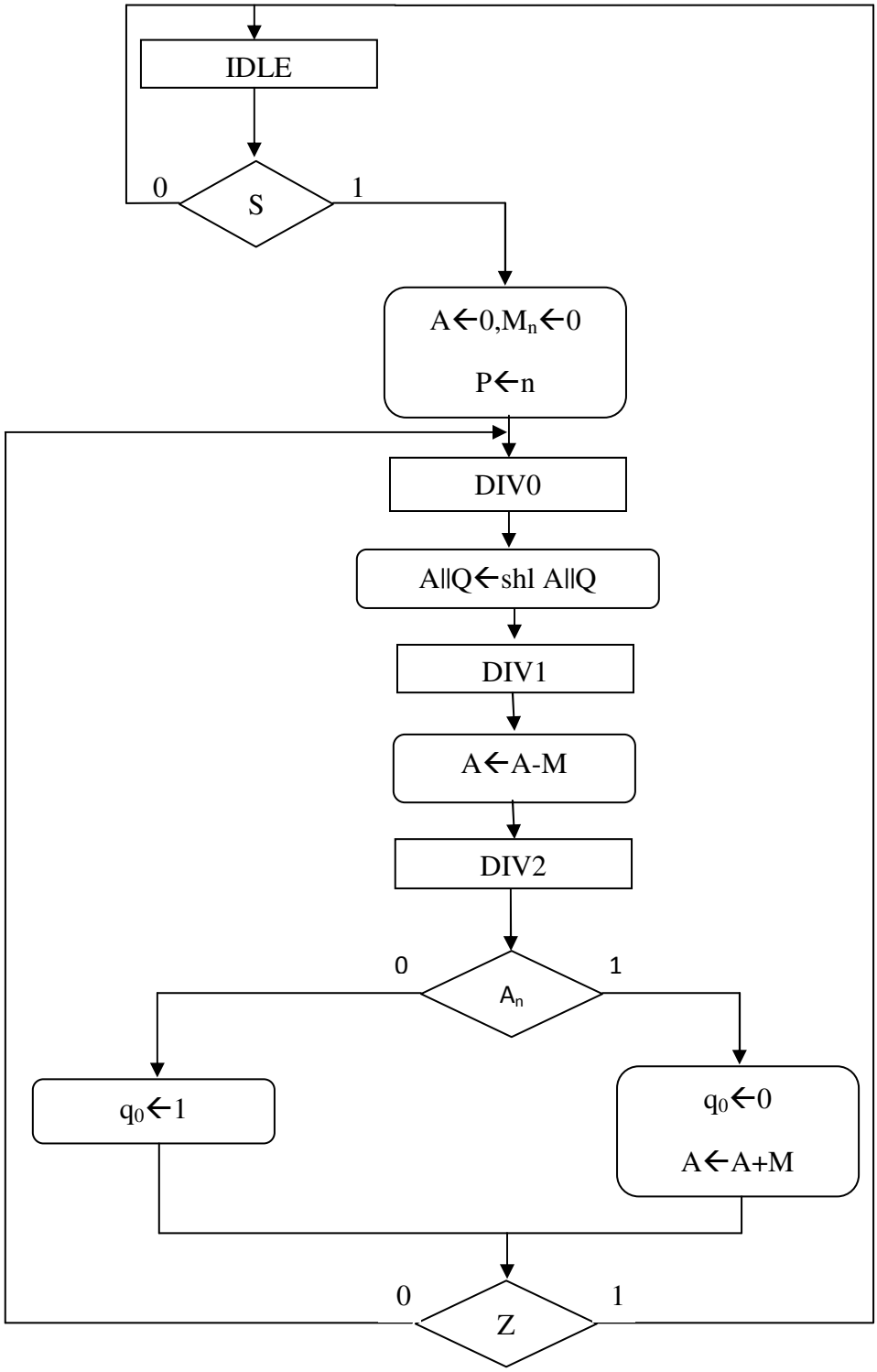
Sayısal bir bilgisayara bölme işlemi için donanım konulacaksa, işlemler biraz değişecektir. Böleni sağa kaydırmak yerine bölünen veya kısmi kalan sola kaydırılır. Böylece sayılar göreceli konumlarına gelmiş olurlar. Çıkarma işlemi bölünene bölenin 2 ye göre tümleyeni eklenmesiyle yapılır. Bölme işlemi için gerekli donanım Şekil 4.22'de görülmektedir.



Şekil 4.22. Bölme İşlemi için donanım mekanizması

Bölen M ve bölünen A ve Q'dadır. Sıra sayıcıya başlangıçta bölenin bit sayısı atanır. Bölünen sola 1 kaydırılır ve bölenin 2 ye göre tümleyeni kaydırmaya eklenir. Sayıların birbirine göre durumu(A≥M veya A<M) A nın en anlamlı bitinden bellidir. Eğer "1" ise bölüm olarak "1" sayısı Q nun en önemsiz bite yazılır. Kısmi kalan sola 1 kaydırılır. Eğer "0" ise bölüm olarak "0" sayısı Q nun en düşük anlamlı bitine yazılır ve M nin değeri kısmi kalana eklenir. Böylece A daki kısmi kalanın eski değeri toplanana kadar devam eder. Kısmi kalan sola kaydırıldığında bölüm bitleri yani Q larda sola kaydırılır. Sıra sayıcı bölenin bit sayısına ulaşınca bölme işlemi biter ve sonuç Q da son kalan ise A dadır. Bölme işlemi için açıklananlar Şekil 4.23'deki akış şemasında açıkça görülmektedir.

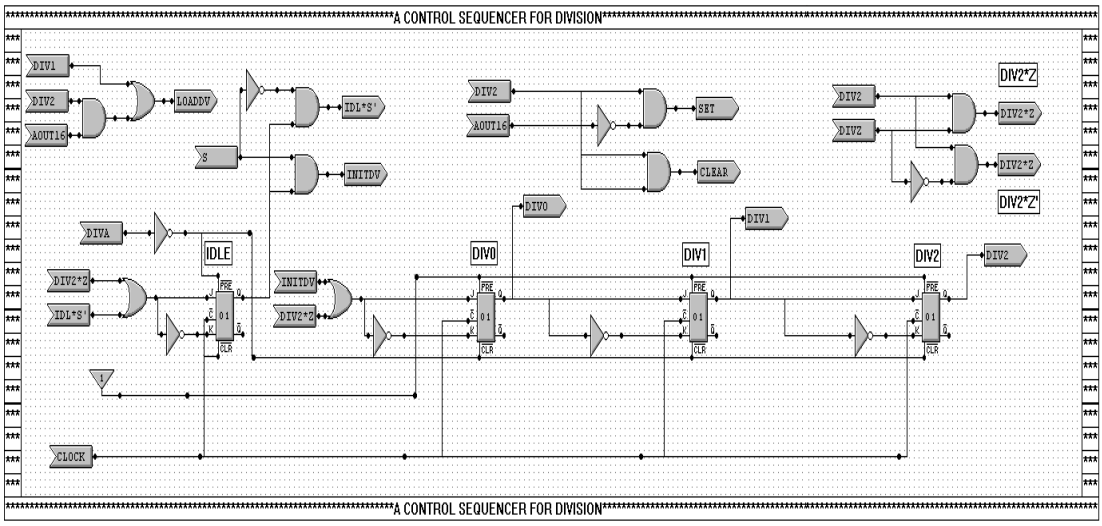
Bölme işlemini akış şemasını verdikten sonra yukarıda anlatılan işleri düzenleyecek bir kontrol mekanizmasına ihtiyaç vardır. IDLE durumu bölme işleminin başlangıcında bölenin ve bölünenin değerlerinin hazır olduğu andır. Başka bir deyişle, bölme işlemi henüz başlamamıştır. Bölme işlemini başlatan "S" sinyali "1" geldiğinde bölme işlemi sonucunda kalanı içerecek olan A kaydedicisine 0 bilgisi transfer ediliyor. Bölen, yani M kaydedicisine bir bit daha eklenerek bu bite "0" bilgisi veriliyor ve en son olarak bölenin bit sayıcı sıra sayıcı P ye yüklendikten sonra kontrol mekanizması DIV0 durumuna geçiyor.



Şekil 4.23. Bölme İşlemi için akış şeması

Bölen, yani M kaydedicisine bir bit daha eklenerek bu bite "0" bilgisi veriliyor ve en son olarak bölenin bit sayıcı sıra sayıcı P ye yüklendikten sonra kontrol mekanizması DIV0 durumuna geçiyor. Bu durumdayken kalanın ve bölünenin bulunduğu AQ kaydedicisi sola bir kaydırılıyor ve kontrol DIV1 durumuna geçiyor. DIV1 durumunda kalandan bölünen çıkartılıp sonuç tekrar kalan kaydedicisine yükleniyor. Bu işlemin ardından kontrol mekanizması DIV2 durumuna geçiyor. Bu durumda kalan kaydedicisinin en anlamlı biti kontrol edilip "0" veya "1" olması durumuna göre kontrol farklı işlemler yerine getiriyor. Eğer "0" ise bölüm biti "1" olarak bölünenin en düşük anlamlı bitine transfer ediliyor. Eğer "1" ise, yani başka bir deyişle kalan bölenden küçük olduğu anlaşılırsa bölüm biti "0" olarak Q yani bölünenin en düşük anlamlı bitine transfer edilip kalana bölen eklenip kalan kaydedicisi yani A ya transfer ediliyor. Bu işlemlerin ardından sıra sayıcının sıfır olup olmadığına bakılıp eğer sıfır ise kontrol bölme işlemini DIV0 konumuna dallandırıp yukarıda anlatılan işlemlerin tekrar edilmesini sağlıyor. Eğer sıra sayıcı sıfır durumuna gelmişse bölme işlemi sonlandırılıp sonuç Q, kalan A kaydedicisine kaydediliyor.

Bölme kontrol devresinde bahsedilen işlemlerde IDLE, DIV0, DIV1 ve DIV2 olmak üzere dört durumdan bahsedildi. Bu dört durumu 4 adet flip-flop kullanarak ifade edebiliriz. Bir anda sadece bir flip-flop yani bir durum aktif olması sağlanmış olur. İşte anlatılan bu işlemleri kontrol işaretleriyle ifade edersek karşımıza Tablo 4.9 çıkacaktır. Bu kontrol devresinin MML programında gerçeklenmiş hali Şekil 4.24'te görülmektedir. Tablo 4.9 incelenirken Şekil 4.23 ile beraber incelenmesi tablonun okunabilirliğini artıracaktır.



Şekil 4.24. Bölme işlemine ait kontrol mekanizmasının MML'de gerçeklenmesi

Tablo 4.9. Bölme işlemine ait kontrol işaretleri

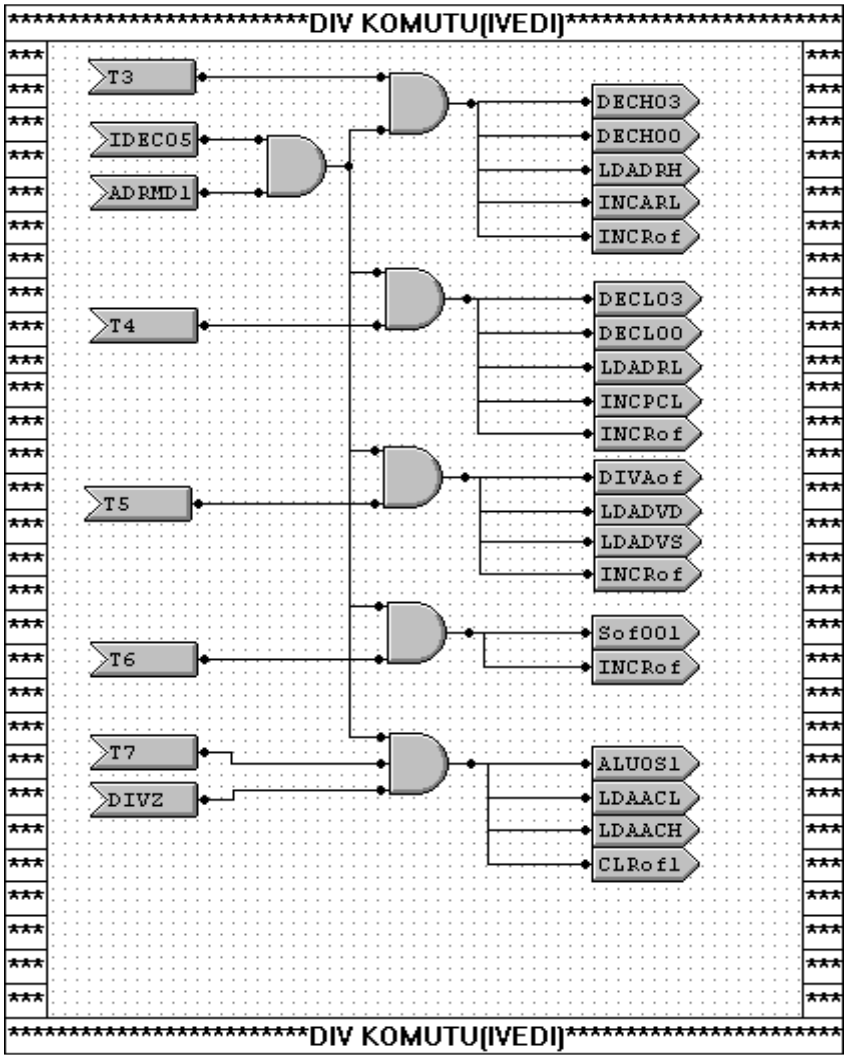
Blok Diyagram Modülü	MikroİŞlem	Kontrol İşareti İsmi	MML'deki Kontrol İfadesi
Register A	A←0	Initialize	IDLE*S
	A←A-M	Load	DIV1
	A←A+M	Load	DIV1*Q_n
	A‖Q←shl A‖ Q	Shift	DIV0
Register Q	Q←IN	Load_DVD	LOADDVD
	A‖Q←shl A‖ Q	Shift	DIV0
	Q_0←0	Clear	DIV1*Q_n
	Q_0←1	Set	DIV1*$\overline{Q_n}$
Counter P	P←n	LoadCt	IDLE*S
	P←P-1	Load	DIV1
Register M	M←IN	Load_DVS	LOADDVS

Bölme için kontrol devresi hazırlandıktan sonra sıra bölme için gerekli olan mikro işlem adımlarını oluşturmaya geldi. Bu mikro işlem adımlarını MML programından alınmış ekran görüntüsü olan Şekil 4.25 üzerinde açıklayalım.

İlk iki adımda bellekteki veri, veri kaydedicine aktarılma işlemi gerçekleştiriliyor. Bu adımlardan sonra akümülatördeki ve veri kaydedicisindeki veri kontrol

mekanizmasının devreye girmesiyle T4 zaman diliminde Q ve M kaydedicilerine yükleme işlemi gerçekleştiriliyor. T6 zaman diliminde bölme işlemi başlatılabilmesi gerekli olan "S" sinyali tek bir çevrim zaman dilimi boyunca aktif edilerek bölme işlemi başlatılıyor. Bu zaman diliminden sonra kontrol bölme devresine geçmiş oluyor. Bölme devresindeki sıra sayıcı sıfır ve T7 zaman diliminde bölme işleminin sonucu akümülatöre transfer edilerek bölme işlemi bitirilir.

Yukarıda anlatılan aritmetik ve lojik birimin(ALU) parçaları olan; aritmetik, mantıksal, bölme ve çarpma işlemlerin sonuçları 16 adet 4x1 lik MUX lar tarafından ALU nun çıkışına yönlendirilir. Burada kullanılan MUX ların seçim uçlarını alacağı değerler bağlı olarak çıkışa hangi kısmın kullanacağı aşağıda Tablo 4.10'da verilmiştir.

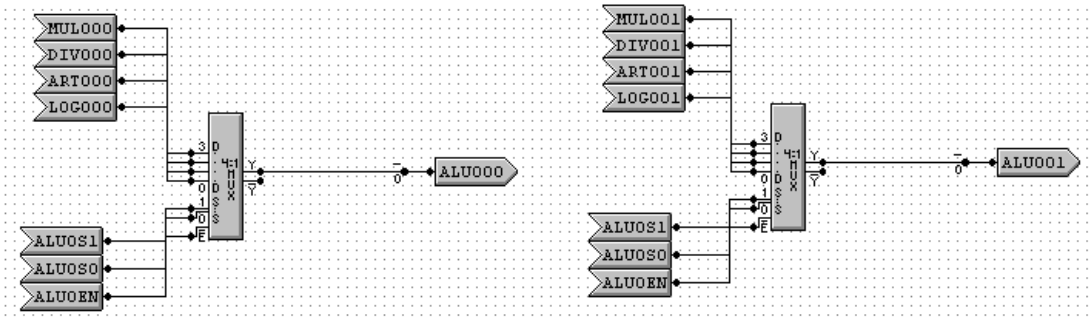


Şekil 4.25. Bölme İşlemine ait mikro işlem adımlarının MML'de gerçeklenmesi

Tablo 4.10. ALU nun çıkışını seçim uçlarına bağlı olarak kullanacak kısımlar

Seçim Uçları		Çıkış
ALUOS1	ALUOS0	Y
0	0	Mantıksal Kısım
0	1	Aritmetik Kısım
1	0	Bölme
1	1	Çarpma

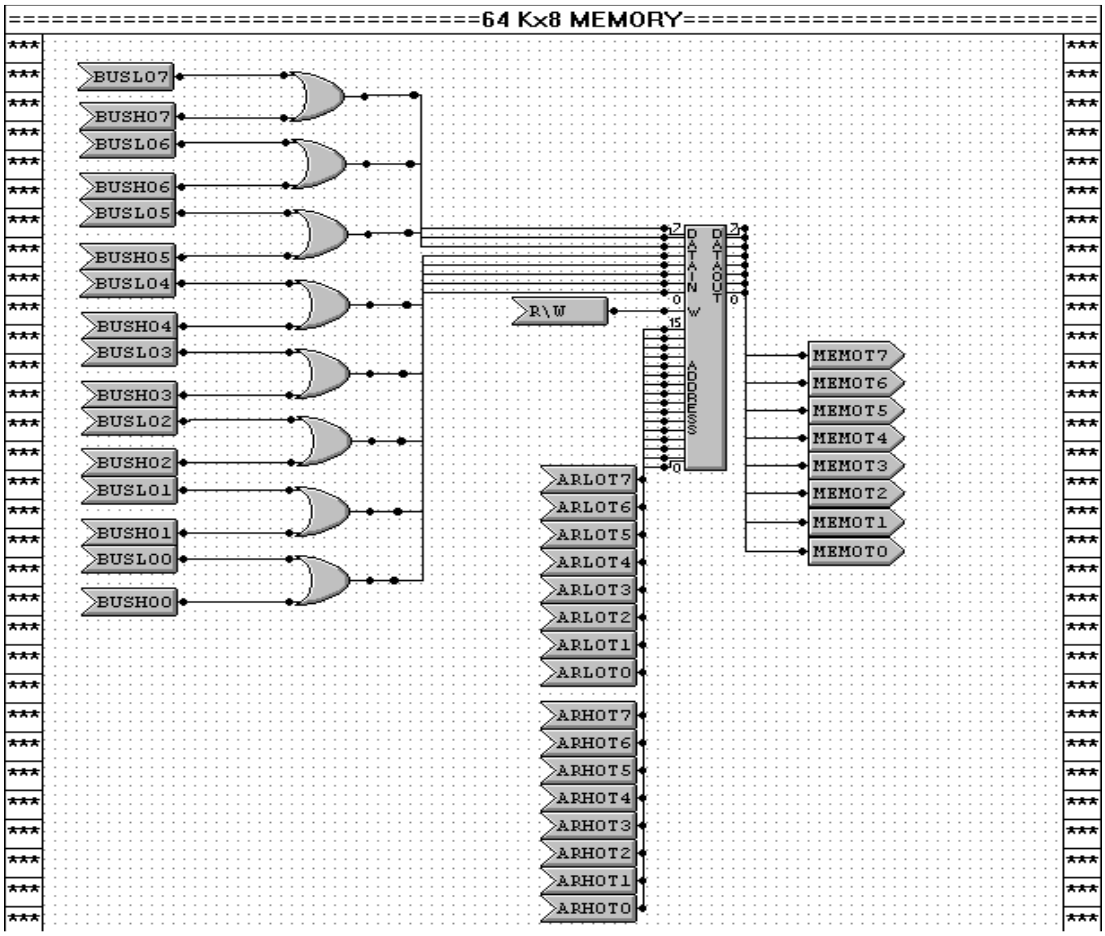
Tablodan görüleceği ALUOS1||ALUOS0=10 olduğunda ALU nun çıkışını bölme devresinden gelecek olan sonuç kullanacaktır. Aynı şekilde "00" olduğunda ise mantıksal kısımdan gelecek olan kısım kullanacaktır. Burada anlatılan işlemlerim MML programında gerçeklenişinin bir bitlik kısmı Şekil 4.26'da görülmektedir.



Şekil 4.26. ALU nun çıkışını kullanacak bölümün belirlendiği kısım

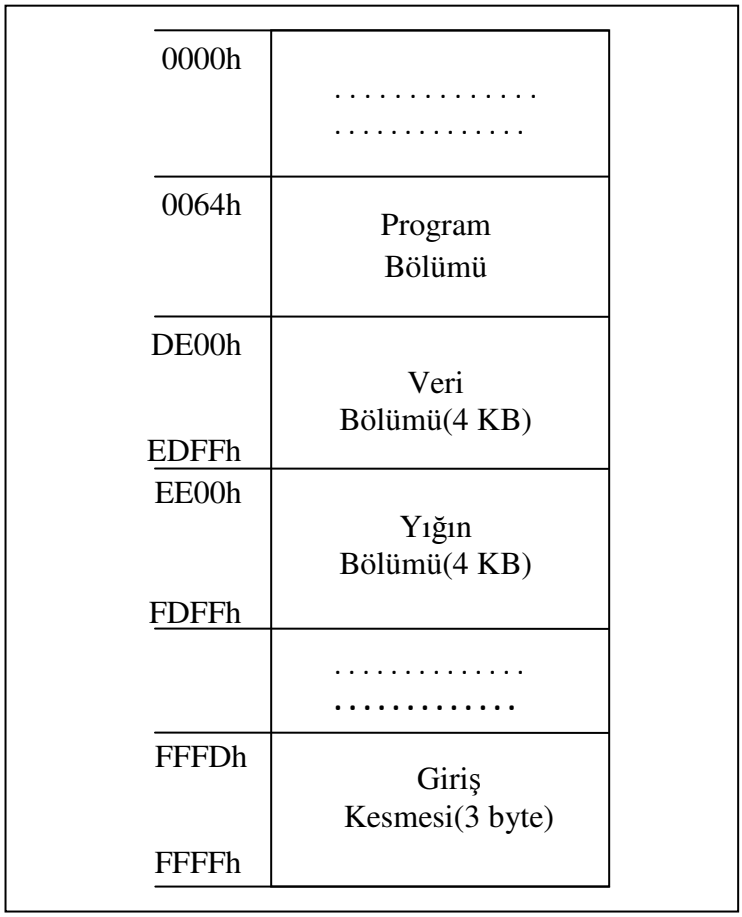
4.7. Bellek Birimi

Tasarlanan bilgisayarda kullanılan bellek elemanı 16 bit adres ucuna sahip olduğundan 64 KB lık bir kapasiteye sahiptir. Kullanılan bellek elemanı Şekil 4.27'de görülmektedir.



Şekil 4.27. Tasarlanan Bellek elemanı

Şekil 4.27'den görüleceği üzere bellek elamanını, hem düşük anlamlı veri yolundan hem de yüksek anlamlı veri yolundan bilgi alabilecek şekilde tasarlanmış 8 bitlik veri girişi, 16 bitlik adres kaydedicisinden gelen adres bilgisini alacak adres uçları ve 8 bitlik çıkışını düşük anlamlı veri yoluna iletecek şekilde tasarlandı. "R\W" kontrol girişi ise bellekten okuma veya yazma yapılacağı zaman gerekli bir giriştir. Bu kontrol girişinin, bellekten okuma yapılacağı lojik "0" , belleğe bilgi yazılacağı zaman lojik "1" yapılması gereklidir. Tasarlanan bilgisayarın kullanacağı 64 KB lık bellek alanı Şekil 4.28'deki gibi bölümlere ayrıldı.



Şekil 4.28. Tasarlanan Belleğin bölümlere ayrılmış hali

4.8. Zamanlama ve Denetim

Tasarlanan bilgisayarda bütün kaydedicilerin zamanı bir ana saat tarafından kontrol edilmektedir[32]. Saat vuruşları sistemdeki tüm flip-floplara ve kaydedicilere uygulanmaktadır. Saat vuruşu kaydedicilerin içeriğini denetim sinyali tarafından izinlendirilmediği sürece değiştirmez. Diğer bir deyişle, saat vuruşu sadece izin verilmiş kaydedicileri ve flip-flopları değiştirir. Denetim sinyalleri denetim birimi tarafından üretilir ve ortak veri yolundaki üç durumlu bufferların girişlerine, işlemci kaydedicilerine ve mikro işlemler için sıra sayıcıya gönderilir.

İki tip denetim yapısı vardır. Donanımsal denetim ve mikro programlanmış denetim. Donanımsal denetim yapısının mantığı kapılar, flip-floplar, kod çözücüler ve diğer sayısal devrelerle sağlanır. Mikro programlanmış denetimde değişiklik mikro

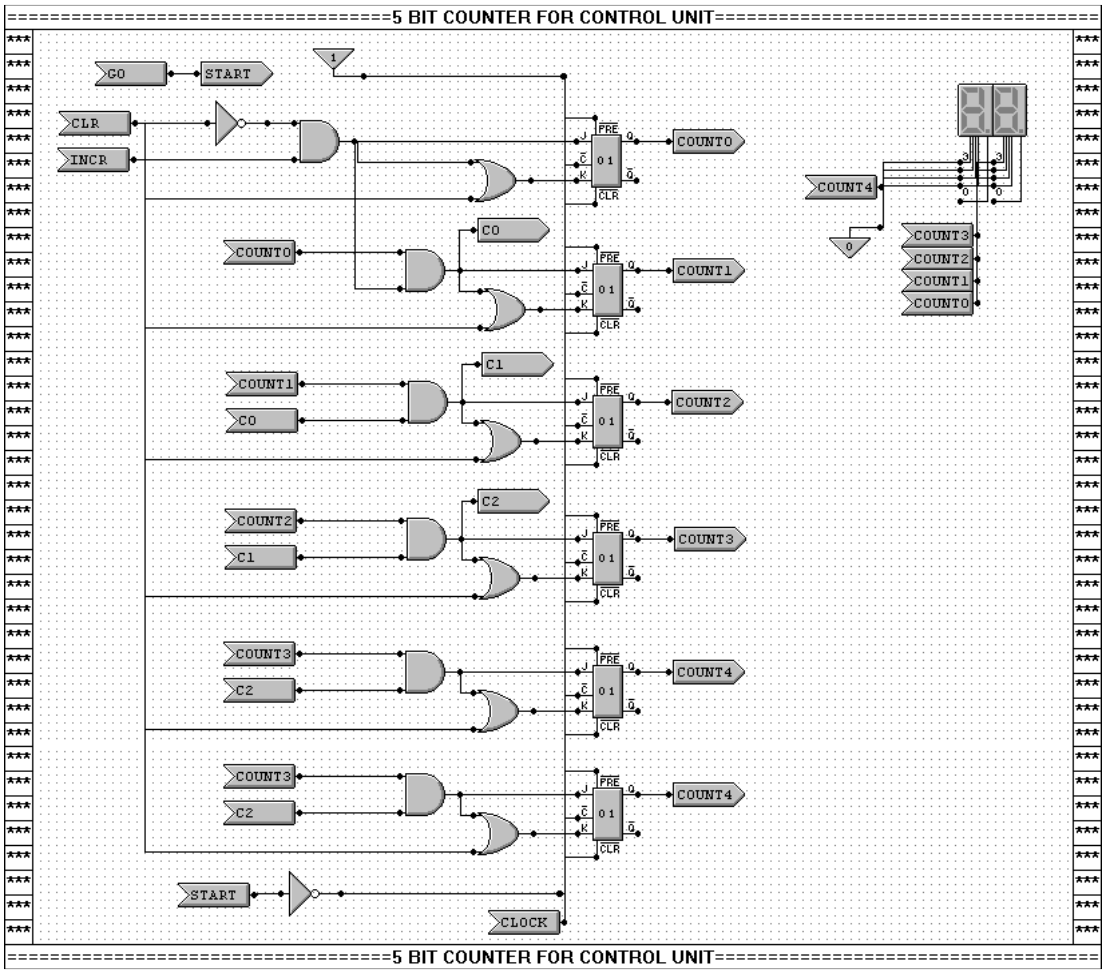
programın güncellenmesi ile sağlanır. Tasarlanan bu bilgisayarda denetim donanımsal olarak tasarlanmıştır.

Denetim birimi 5 bitlik bir sıra sayıcı ve bu sayıcının çıkışları ise 5x32 lik bir kod çözücüden geçirilerek T0'dan T31'e kadar zaman dilimleri elde edilir. Komut kaydedicisinin(IR) dört, beş ve altıncı bitleri 3x8 lik bir kod çözücüden geçirilerek adresleme modlarını belirleyen bir sinyal elde edilir. Bunlara ilaveten komut kaydedicisinin 0, 1, 2, 3 ve 7. Bitleri 5x32 lik bir kod çözücüden geçirilerek yürütülecek olan komutun tipi belirlenir. İşte denetim biriminde yer alan bu üç kod çözücüden gelen sinyaller komutların icra edilebilmesi için gerekli sinyallerin çoğunu oluşturur. Denetim biriminin MML programında tasarlanmış hali Şekil 4.29a, Şekil 4.29b, Şekil 4.29c ve Şekil 4.29d'de verilmiştir.

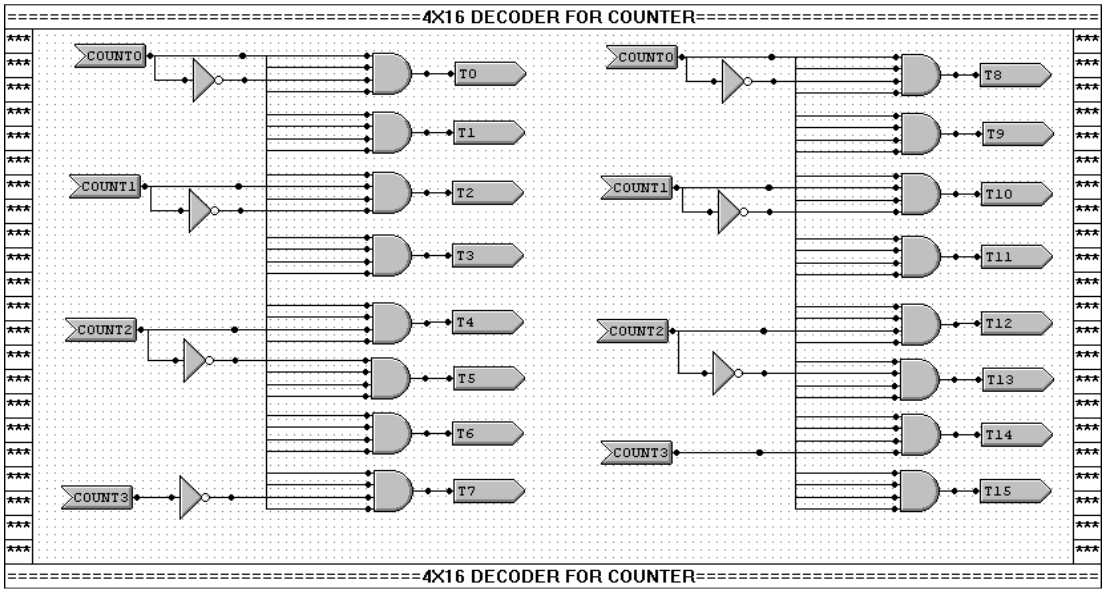
Komut kaydedicisinin 4,5 ve 6. bitleri komut tasarımı adlı bölümde bahsedildiği gibi adresleme modunu tayin etmeye yarayan bitlerdir. Bu üç bit 3x8 lik bir kod çözücüden geçerek 8 adet çıkış elde edilmektedir. Bu bilgisayarda bunlardan 6 âdeti kullanıldı. Diğer iki adresleme modu ise sonradan bilgisayara ilave adresleme modları eklenebilmesine olanak sağlar. Tablo 4.11'de çıkışların hangi adresleme modlarına karşılık geldiği görülmektedir.

Tablo 4.11. Denetim birimi tarafından üretilen adresleme mod sinyalleri

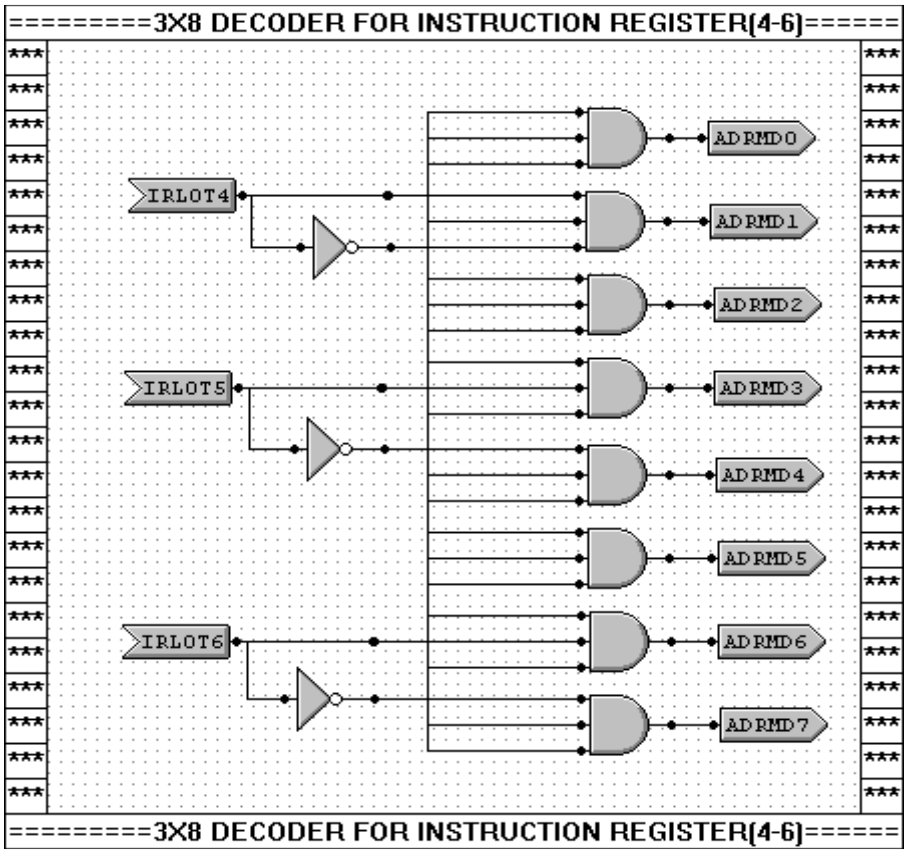
Kontrol Adı	**Adresleme Modu Tipi**	**Kontrol Adı**	**Adresleme Modu Tipi**
ADRMD0	Doğal Mod	ADRMD3	Dolaylı Mod
ADRMD1	Derhal Mod	ADRMD4	İndis Mod
ADRMD2	Direkt Mod	ADRMD5	Göreceli Mod



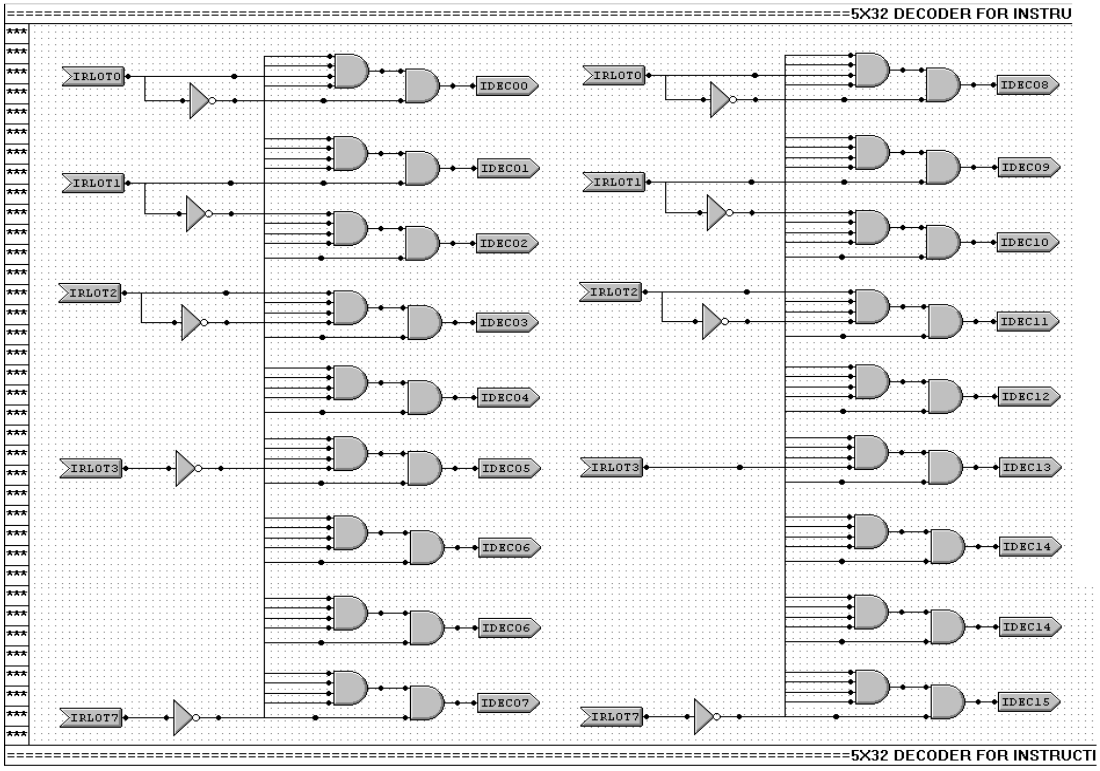
Şekil 4.29a. Denetim biriminde kullanılan 5 bitlik sıra sayıcı



Şekil 4.29b. Sıra sayıcı için 5x32 lik kod çözücü



Şekil 4.29c. Adresleme Modları için 3x8 lik kod çözücü



Şekil 4.29d. Komut tipini belirlemeye yarayan 5x32 lik kod çözücünün bir kısmı

Komut kaydedicisinin 5 biti Şekil 4.29.d'de görüleceği üzere 5x32 lik bir kod çözücüden geçirilerek komutun tipini belirlemeye yarayan sinyaller üretilmektedir. Örneğin; komut kaydedicisinden gelen 5 bitin 00101 olduğunu varsayalım. Bu durumda bu kod çözücünün 5 numaralı ucu yani IDEC05 sinyali aktif olacaktır. Bu sinyal tek başına komutun tipini belirlemeye yaramayacaktır. Çünkü aynı sinyali üreten başka komutlar olabilir. Sadece bu sinyal göz önünde bulundurulursa hem DIV hem de NEG komutunun çözülmesinde elde edilen sinyaldir. İşte tam bu anda adresleme modalarından gelen sinyal devreye girerek hangi komutun olacağına karar verir. Adresleme modlarını çözümleyen kod çözücüden de ADRMD1 geldiğini varsayarsak bu sinyallerin Ek-M incelendiğinde DIV komutuna karşılık geldiği görülecektir. Bu sinyal bölme işlemi için tasarlanan mikro işlemlerde yer aldığından, buradaki sinyalleri aktif edecek ve gerekli zaman dilimlerinde tanımlanan işleri yerine getirecektir.

4.9. Komut Tasarımı

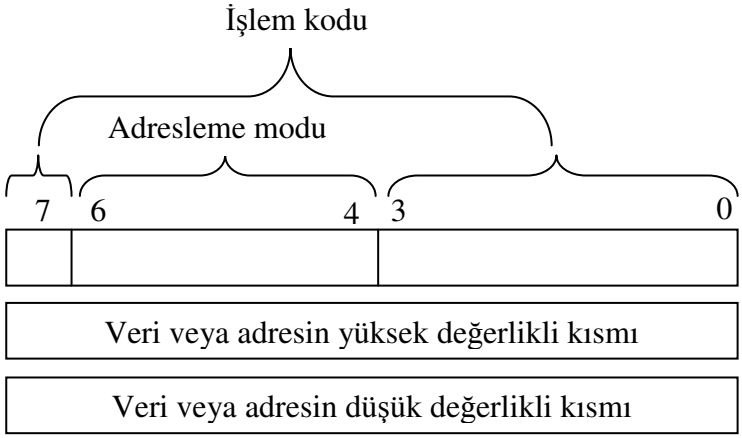
Bir bilgisayarın içyapısı, kaydedicilerdeki veriler üzerine işlem yapacak mikro işlemler sırasıyla tanımlanmıştır. Bir sayısal bilgisayar verilerle birçok mikro işlem icra etme yeteneğine sahip olduğu gibi, ayrıca bu mikro işlemlerin hangi sırada icra edileceği konusunda da önceden programlama yeteneğine sahiptir[32]. Bilgisayarın kullanıcısı, bilgisayarın işlemlerini bir program ile denetler. Program bir dizi buyruklar olup bunlar işlemleri belirtirler. Ayrıca verileri ve hangi işlemin sırasının geldiğini de belirtirler.

Bir bilgisayar buyruğu ikili bir koddur. Bir dizi mikro işlemleri tanımlar. Buyruklar verilerle beraber bellekte bulunurlar. Bilgisayar her buyruğu bellekten okur bunu denetim kaydedicisine yazar. Denetim bu kodu anlar ve onun gereğini mikro işlemleri sırasıyla icra ederek yerine getirir. Her bilgisayarın kendine özel buyruk kümesi vardır. Buyrukların bellekte saklanabilmesi ve sonra sırasıyla icra edilmesi, bir bilgisayarın en önemli özelliğidir.

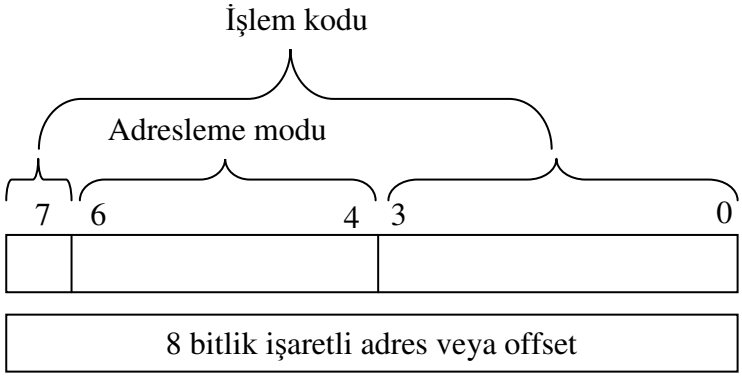
Komut kodu bir grup bitten ibarettir. Bunlar bilgisayara belli bir işlemi yapmasını söylerler. Genelde buyruk birkaç kısma ayrılır. Buyruğun en önemli parçası işlem

parçasıdır. Burada bir kısım bitler toplama, çıkarma, çarpma, kaydırma ve tümleme gibi bazı işlemleri tanımlarlar. Buyrukların içindeki işlem biti sayısı bilgisayar içinde kullanılabilen işlem sayısına bağlıdır. n tane bit ile 2^n(veya daha az) işlem gösterilebilir. Buyruk kodunda yer alan diğer önemli kısım ise adres modunu belirleyen kısımdır. Burada anlatılacak olan bilgisayarda buyruk kodunun 5 biti işlem koduna ayrılmıştır. Beş bitlik bir işlem koduyla 32 adet işlem yerine getirilirken bu bilgisayar 60 adet işlemi icra edebilmektedir. Buradaki farklılık komut kodundaki kalan üç bitin adresleme modu için kullanılarak daha az sayıdaki işlem kodu ile daha fazla komut tasarlanabilmesine olanak sağlamaktadır.

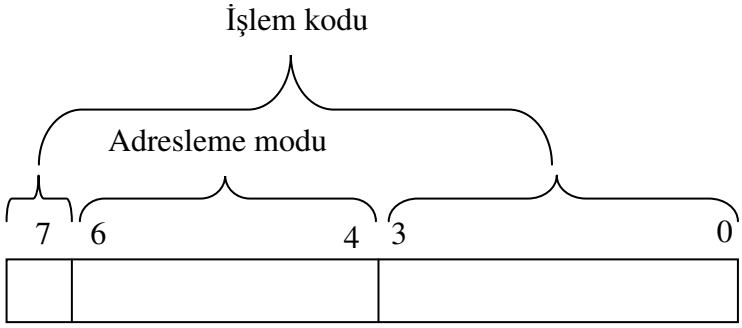
Bu bilgisayarda derhal adresleme modu, direk adresleme modu ve dolaylı adresleme modlarına sahip komutlar 3 byte, indis ve göreceli adresleme modlarına sahip komutlar 2 byte, doğal adresleme moduna sahip komutlar ise 1 byte olarak tasarlanmıştır. Bununla ilgili yapı aşağıda Şekil 4.30a, Şekil 4.30b ve Şekil 4.30c'de görülmektedir.



Şekil 4.30a. Derhal, direkt ve dolaylı adresleme modu kullanan komut kodu tasarımı



Şekil 4.30b. İndis ve göreceli adresleme modu kullanan komut kodu tasarımı



Şekil 4.30c. Doğal adresleme modu kullanan komut kodu tasarımı

4.9.1. İşlemci komutları

Bir komut, komut seti mimarisi olarak tanımlanan işlemcinin tek bir işlemidir. Daha yaygın bir ifadeyle, yürütülen bir programın bir parçası olarak tanımlanır[32]. Tasarımı yapılan bu bilgisayarda komutlar akümülatör ve bellek üzerinde işlem yapan komutlar, indis ve yığın göstergeci ile ilgili komutlar, dallanma ile ilgili komutlar, durum kod kaydedicisi üzerinde işlem yapan komutlar ile giriş-çıkış komutları olmak üzere dört ana bölümde toplanabilir. Komut listesi Ek-F, Ek-G, Ek-H, Ek-I ve Ek-J'de verilmiştir.

İndis ve yığın göstergeci ile ilgili komutlar 8 adettir. Bu komutlar da derhal, direkt, dolaylı, indis ve doğal adresleme modlarını kullanmaktadır. Derhal, direkt ve dolaylı adresleme modlarını kullanan komutların uzunluğu 3 byte olup, indis adresleme modunu kullanan komutların uzunluğu ise 2 byte'dır. Doğal adresleme modunu

kullanan komutun uzunluğu akümülatör ve bellek üzerine işlem yapan komutlar da olduğu gibi tek byte'tır.

Dallanma komutları 22 adet olup göreceli, direkt, indis ve doğal adresleme modlarını kullanmaktadırlar. Göreceli ve indis adresleme modunu kullanan komutların uzunluğu 2 byte olup direkt adresleme modunu kullanan komutların uzunluğu 3 byte'tır. Doğal adresleme modunu kullanan komutlar ise diğer komut çeşitlerinde olduğu gibi tek byte'tır.

Durum kod kaydedicisi üzerinde işlem yapan komutlar 6 adet olup bu komutların hepsi doğal adresleme modu kullanmaktadır. Giriş ve çıkış komutları 3 adet olup bu komutlar da doğal adresleme modunu kullanırlar.

4.9.1.1. Akümülatör ve bellek işlem komutları

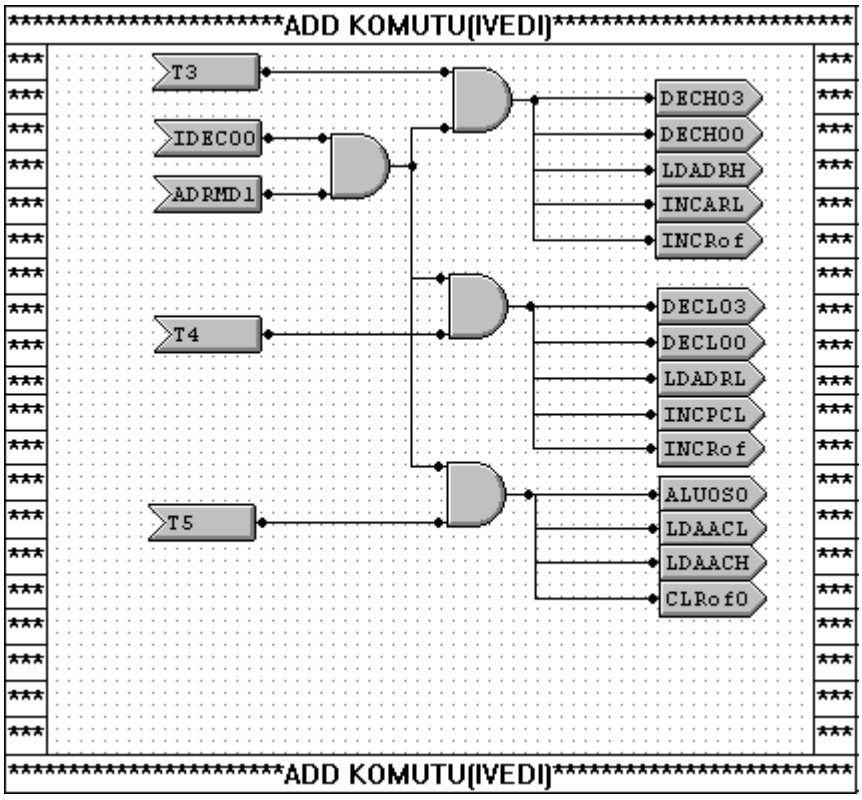
Akümülatör ve bellek üzerine işlem yapan komutlar 21 adettir. Bu komutlardan bellek üzerine işlem yapanları derhal, direkt, dolaylı ve indis adresleme modlarını kullanmaktadır ve bu komutların uzunluğu indis adresleme modu hariç 3 byte, indis adresleme modunu kullanan komutlar ise 2 byte uzunluğuna sahiptirler. Akümülatör üzerine işlem yapan komutlar ise tek byte uzunluğuna sahip olup doğal adresleme modunu kullanmaktadırlar. Şimdi bu bölümde bellek ve akümülatör üzerine işlem yapan komutlardan ADD komutunun derhal, direkt, dolaylı ve indis adresleme modlarının çalışmasını adım adım inceleyeceğiz. Diğer komutların mikro işlem adımları Ek-A'da verilmiştir.

a) ADD(Derhal Mode): Komutun al-getir ve kodunu çöz evresinden sonraki mikro işlem asımları Tablo 4.12'de görülmektedir.

Tablo 4.12. ADD(Derhal Mod) Komutunun Mikro İşlem Adımları

ADD(Derhal Mod) Komutunun Mikro İşlem Adımları		
Adım-1	T3* IDEC00*ADRMD1	$DR_H \leftarrow M[AR]$, $AR \leftarrow AR+1$
Adım-2	T4* IDEC00*ADRMD1	$DR_L \leftarrow M[AR]$, $PC \leftarrow PC+1$
Adım-3	T5* IDEC00*ADRMD1	$AC \leftarrow AC+DR$, $C \leftarrow C_{out}$, $SC \leftarrow 0$

Al-getir ve kodunu çöz evresinde yapılan işlemleri hatırlayacak olursak şu an adres kaydedicisinde(AR) ADD komutundan sonra gelen operandın yüksek anlamlı kısmının bulunduğu bellek bölgesini işaret etmektedir. İşte Adım-1'de bellek bölgesinde işaret edilen bu veri, veri kaydedicisinin(DR) yüksek anlamlı kısmına transfer edilerek adres kaydedicisinin değeri bir artırılarak bellekteki bir sonraki alan işaret edilmiş oluyor. Adım-2' de ise bellekte işaret edilen veri, veri kaydedicisinin düşük anlamlı kısmına transfer edilerek program sayıcısı bir artırılarak program sayıcısının bir sonraki komutu işaret etmesi sağlanmış oluyor. Son olarak Adım-3'de aritmetik ve mantık biriminin(ALU) girişlerine gelmiş olan veri kaydedicisindeki veri ile daha önceden hazır bulunan akümülatör içeriği toplama işlemine tabi tutularak tekrar akümülatör üzerine kaydedilip sayıcı sıfırlanır. Sayıcının sıfırlanmasından maksat bir sonraki komutun al-getir ve kod çözme aşamasına girmesi temin etmektir. ADD komutunun derhal modu ile ilgili Multimedia Logic simülasyon programında yapılan işlemler Şekil 4.31'de görülmektedir.



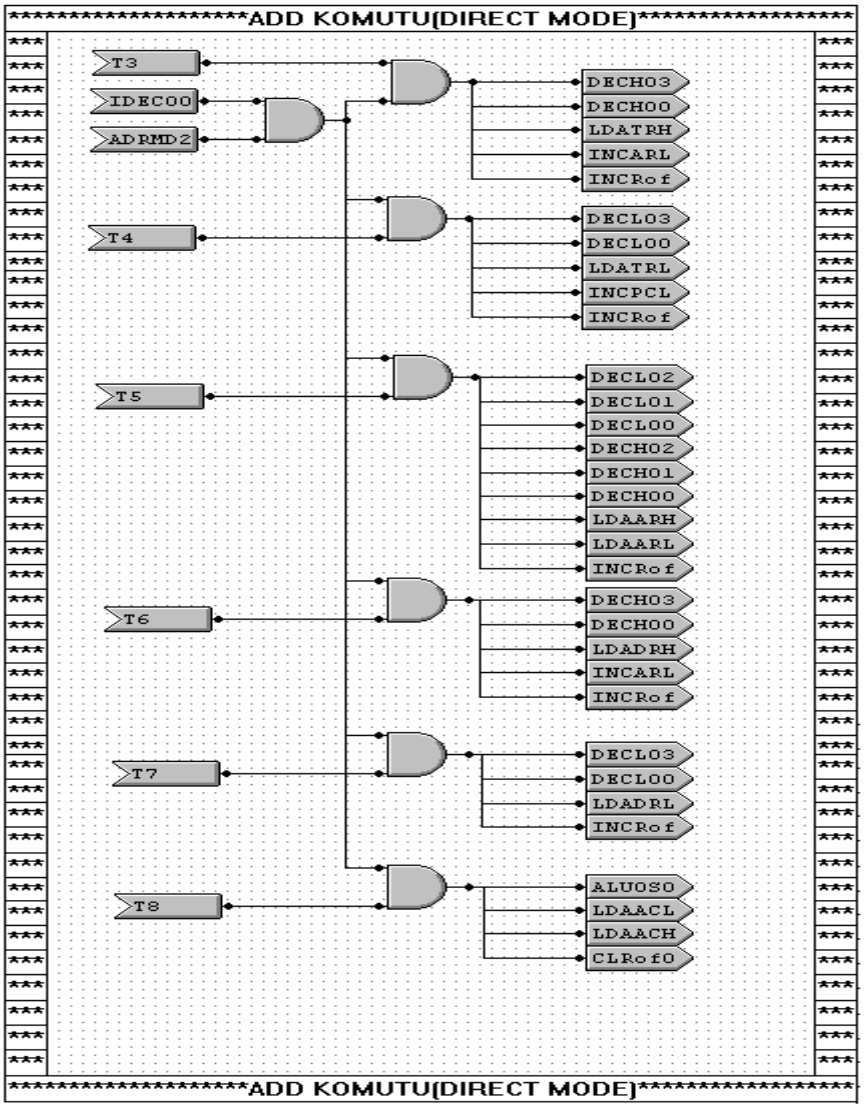
Şekil 4.31. ADD(Derhal Mod) komutunun Multimedia Logic'teki gerçeklenişi

b) ADD(Direkt Mod): Komutun adım adım mikro işlem adımları Tablo 4.13'de gösterilmiştir.

Tablo 4.13. ADD(Direkt Mod) Komutunun Mikro İşlem Adımları

ADD(Direkt Mod) Komutunun Mikro İşlem Adımları		
Adım-1	T3* IDEC00*ADRMD2	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
Adım-2	T4* IDEC00*ADRMD2	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
Adım-3	T5* IDEC00*ADRMD2	$AR \leftarrow TR$
Adım-4	T6* IDEC00*ADRMD2	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
Adım-5	T7* IDEC00*ADRMD2	$DR_L \leftarrow M[AR]$
Adım-6	T8* IDEC00*ADRMD2	$AC \leftarrow AC+DR, C \leftarrow C_{out}, SC \leftarrow 0$

Direkt adresleme modunda komutun işlem kodu kısmından sonra gelen ikinci ve üçüncü byte veriler, verinin bellekte bulunduğu adresi göstermektedir. Tablo 4.13'de görüldüğü üzere başlangıçtaki üç adım bu byte'larda yer alan adres bilgisini adres kaydedicisine yerleştirmeği göstermektedir. Burada geçici kaydedici(TR) kullanmadaki maksat şu şekilde açıklanabilir: Bilgisayarda kullanılan bellek 8 bitlik bir bellek ve 16 bitle adreslenebildiğinden, operand kısmında yer alan adres bilgisi iki parçada alınabilir. Bu parçalar ilk önce geçici kaydedicisine, daha sonra buradan adres kaydedicisine transfer edilir. Dördüncü, beşinci ve altıncı adımlar derhal adresleme modunu kullanan ADD komutunda olduğu gibi yerine getirilir. ADD komutunun direkt modu ile ilgili Multimedia Logic simülasyon programında yapılan işlemler Şekil 4.32'den de görüleceği üzere 9 adımda tamamlanmaktadır.



Şekil 4.32. ADD(Direkt Mode) komutunun Multimedia Logic'te gerçeklenişi

c) ADD(Dolaylı Mod): Komutun adım adım mikro işlem adımları Tablo 4.14'de gösterilmiştir.

Tablo 4.14. ADD(DolaylıMod) Komutunun mikro işlem adımları

ADD(Dolaylı Mod) Komutunun Mikro İşlem Adımları		
Adım-1	T3* IDEC00*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
Adım-2	T4* IDEC00*ADRMD3	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
Adım-3	T5* IDEC00*ADRMD3	$AR \leftarrow TR$
Adım-4	T6* IDEC00*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
Adım-5	T7* IDEC00*ADRMD3	$TR_L \leftarrow M[AR]$

Tablo 4.14.(Devam) ADD(DolaylıMod) Komutunun mikro işlem adımları

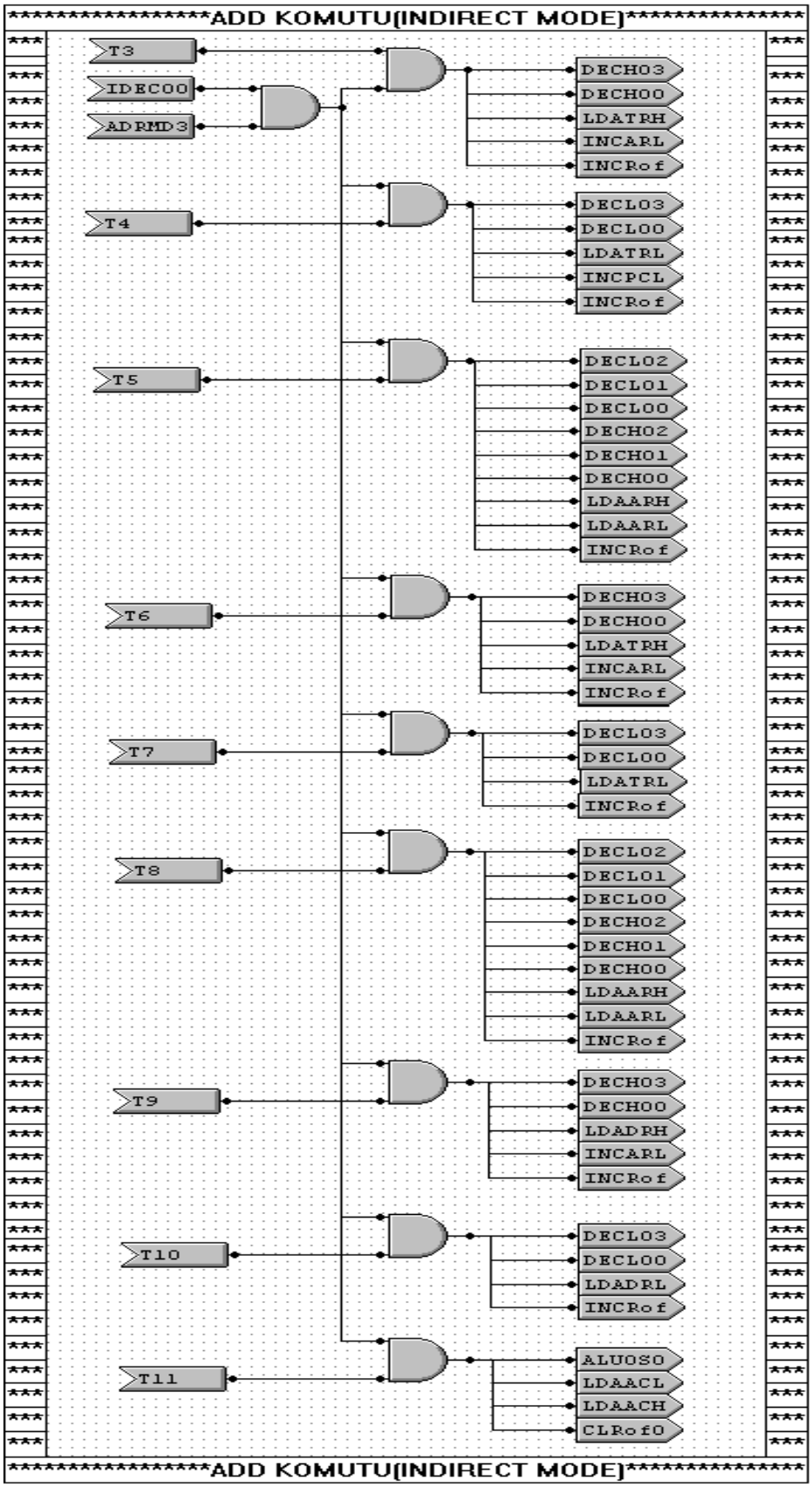
ADD(Dolaylı Mod) Komutunun Mikro İşlem Adımları		
Adım-6	T8* IDEC00*ADRMD3	AR←TR
Adım-7	T9* IDEC00*ADRMD3	DR_H←M[AR],AR←AR+1
Adım-8	T10* IDEC00*ADRMD3	DR_L←M[AR]
Adım-9	T11* IDEC00*ADRMD3	AC←AC+DR, C←C_{out} ,SC←0

Tablo 4.14'den görüleceği üzere ilk altı adım direkt adresleme modunda anlatılan işlerin tekrarı şeklindedir. İki defa tekrar etmesinin sebebi operand kısmında yer alan adresin verinin adresi değil, verinin adresinin nerede bulunduğunu belirten adres bulunmasından kaynaklanmaktadır. Son üç adım yukarıda anlatılan adresleme modlarında kullanılan işlerin tekrarı niteliğindedir. Bu moda sahip ADD komutu aşağıda Şekil 4.33'de görüleceği üzere 12 çevrimde bitmektedir.

d) ADD(İndis Mod): Komutun adım adım mikroişlemleri Tablo 4.15'de gösterilmiştir.

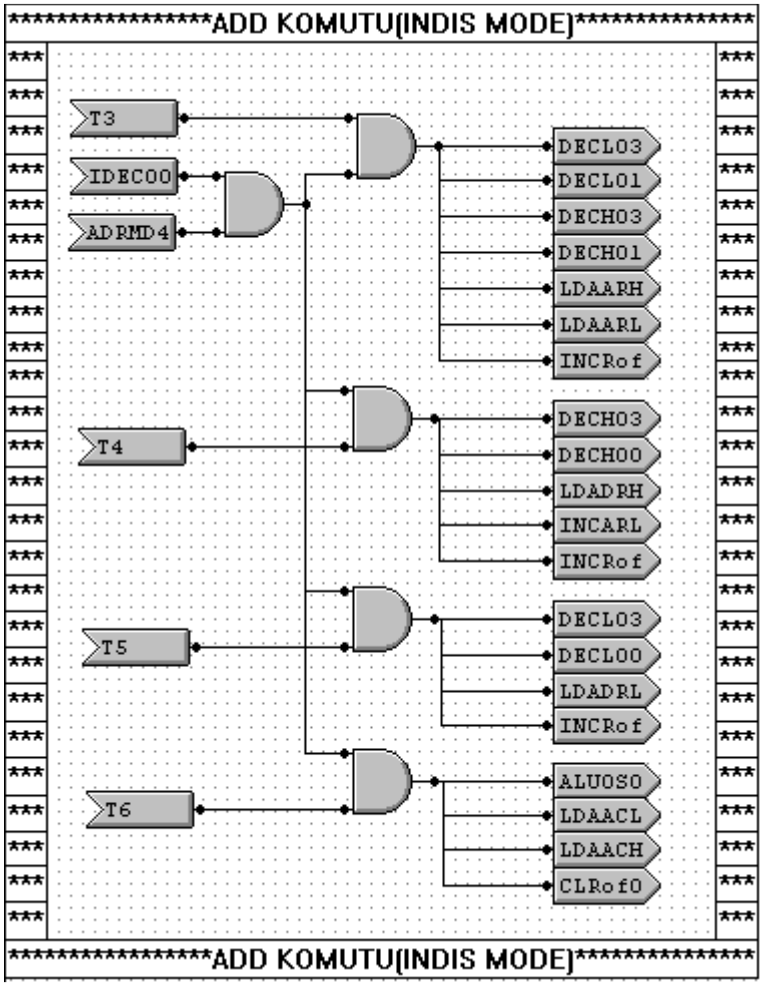
Tablo 4.15. ADD(İndis mod) Komutunun mikroişlem adımları

ADD(İndis Mod) Komutunun Mikro İşlem Adımları		
Adım-1	T3* IDEC00*ADRMD4	AR←Etkin Adres
Adım-2	T4* IDEC00*ADRMD4	DR_L←M[AR], AR←AR+1
Adım-3	T5* IDEC00*ADRMD4	DR_H←M[AR]
Adım-4	T6* IDEC00*ADRMD4	AC←AC+DR, C←C_{out} ,SC←0



Şekil 4.33. ADD(Dolaylı Mod) komutunun Multimedia Logic programındaki mikro işlem adımları

Tablo 4.15'deki adım-1'de bu bilgisayar için önceki bölümlerde bahsedildiği üzere bir göreceli ve indis adresleme modları için özel olarak hazırlanmış adres hesaplama devresinden gelen etkin adres bilgisi adres kaydedicisine aktarılır. Adım-2 ve Adım-3'de ise adres kaydedicisine atanan adresin bellekte gösterdiği kısımdaki veri sırasıyla veri kaydedicisinin yüksek anlamlı ve düşük anlamlı kısımlarına transfer edilerek Adım-3 toplama işlemi gerçekleştirilir. Toplama işleminin ardından sayıcı sıfırlanarak mikroişlemcinin yeni bir komutu al-getir ve kodunu çöz evresine getirmesine olanak sağlar. Tablodan da görüleceği üzere bu adresleme modunu kullanan ADD komutu 7 adımda icra edilmektedir. Bu yedi adımın Multimedia Logic programında tasarlanmış hali Şekil 4.34'de görülmektedir.



Şekil 4.34. ADD(İndis Mod) komutunun Multimedia Logic programındaki mikro işlem adımları

4.9.1.2. İndis ve yığın işlem komutları

İndis ve yığın üzerine komutlar 8 adet olup bu komutlar derhal, direkt, dolaylı, indis ve doğal adresleme modlarını kullanmaktadırlar. Derhal, direkt ve dolaylı adresleme modlarını kullanan komutların bellekte işgal ettiği byte sayısı üç, indis adresleme modunu kullanan komutların işgal ettiği byte sayısı iki ve doğal adresleme modunu kullanan komutların ise bir byte'dır.

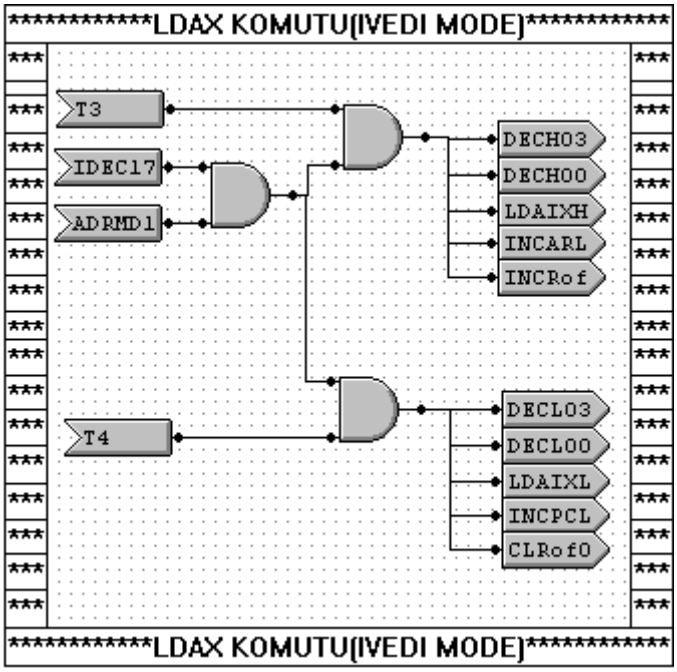
İndis ve yığın işlem komutlarından Load Index Register(LDAX) komutunun bahsedilen adresleme modlarındaki mikro işlem adımlarını inceleyelim. Diğer komutların mikro işlem adımları Ek-B'de verilmiştir.

a) LDAX(Derhal Mod): Bu komutun bellekte göstermiş olduğu işlem kodu 91h olup bellekte 3 byte yer işgal etmektedir. İşlem kodundan sonra gelen iki byte'lık kısım, bu komut ile indis kaydedicisine yüklenecek olan değeri göstermektedir. Bu komut ile mikro işlem adımları Tablo 4.16'de gösterilmiştir.

Tablo 4.16. LDAX(Derhal Mod) komutunun Mikro İşlem adımları

LDAX(Derhal Mod) Komutunun Mikro işlem adımları		
Adım-1	T3* IDEC17*ADRMD1	$IX_H \leftarrow M[AR], AR \leftarrow AR+1$
Adım-2	T4* IDEC17*ADRMD1	$IX_L \leftarrow M[AR], PC \leftarrow PC+1, SC \leftarrow 0$

Tablo 4.16 incelendiğinde Adım-1'de, sayıcının çıkışı T3, komut kodu çözücünün çıkışı IDEC17 ve Adresleme Modu kod çözücünün çıkışı ADRMD1 olduğunda, adres kaydedicinin bellekte göstermiş olduğu kısımdaki veri indis kaydedicisinin yüksek anlamlı kısmına transfer edilerek adres kaydedicisinin değeri bir artırılarak bir sonraki kısmı göstermesi sağlanmış olur. Adım-2'de ise sayıcının çıkışı T4 olduğunda adres kaydedicisinin göstermiş olduğu yerdeki veri indis kaydedicisinin düşük anlamlı kısmına transfer edilir. Her komutta olduğu gibi program sayıcısı bir artırılıp sayıcı sıfırlandığında işlemcinin bir sonraki komutu alıp işlemesi sağlanmış olur. Bu komutun icra edilmesi al-getir ve kodunu çöz evresi hariç iki çevrim zamanı almaktadır. Bu komutun Multimedia Logic programında mikro işlem adımlarının tasarlanması Şekil 4.35'de gösterilmiştir.



Şekil 4.35. MML simülasyonunda LDAX(Derhal Mod) Komutunun Mikroişlemleri

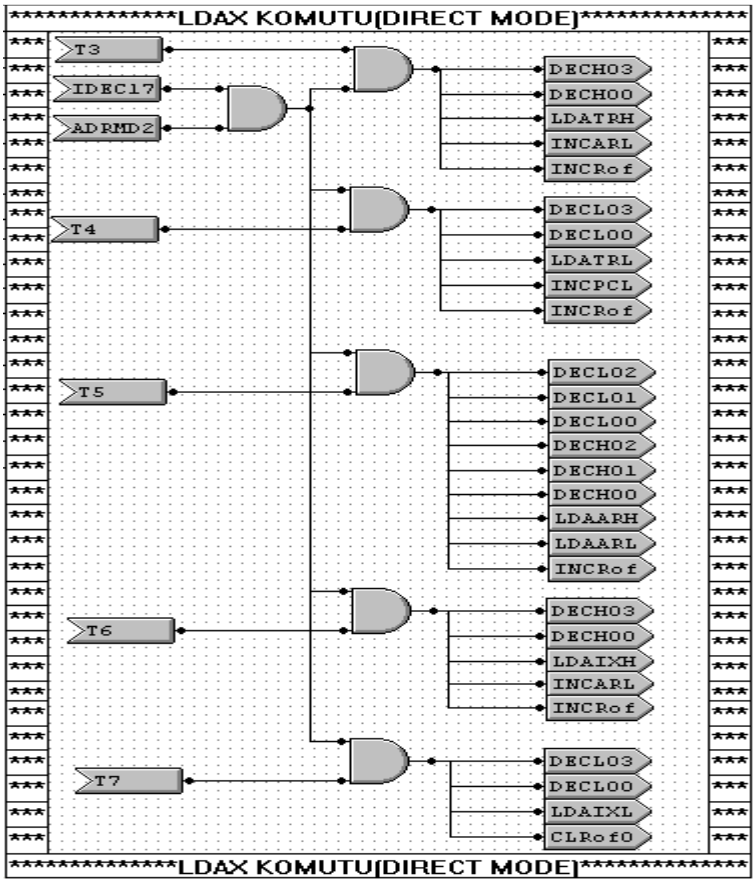
b) LDAX(Direkt Mod): Komutun bellekte göstermiş olduğu işlem kodu A1h olup bellekte 3 byte'lık bir yer işgal etmektedir. Bu komutun mikro işlem adımları Tablo 4.17'de gösterilmiştir.

Tablo 4.17. LDAX(Direkt Mod) Komutunun Mikro İşlem Adımları

LDAX(Direkt Mod) Komutunun Mikro İşlemleri		
Adım-1	T3* IDEC17*ADRMD2	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
Adım-2	T4* IDEC17*ADRMD2	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
Adım-3	T5* IDEC17*ADRMD2	$AR \leftarrow TR$
Adım-4	T6* IDEC17*ADRMD2	$IX_H \leftarrow M[AR], AR \leftarrow AR+1$
Adım-5	T7* IDEC17*ADRMD2	$IX_L \leftarrow M[AR], SC \leftarrow 0$

Adım-1'de sayıcının çıkışı T3, komut kod çözücünün çıkışı IDEC17 ve adresleme modu kod çözücünün çıkışı ADRMD2 olduğunda adres kaydedicinin bellekte göstermiş olduğu veri geçici kaydedicinin yüksek anlamlı kısmına atılır ve adres kaydedicinin değeri bir arttırılarak sayıcının T4 çıkışı üretmesi sağlanır. Adım-2'de ise Adım-1'de değinilen olayın aynısı gerçekleştirilir. Aradaki fark, adres kaydedicinin bellekte göstermiş olduğu veri geçici kaydedicinin yüksek anlamlı

kısmına değil düşük anlamlı kısmına transfer edilir. Bu iki adımda geçici kaydediciye atılmasının sebebi şu şekilde açıklanabilir: Eğer birinci adımda adres kaydedicinin bellekte göstermiş olduğu değer adres kaydedicinin yüksek öncelikli kısmına atılmış olsaydı, Adım-1'de yer alan AR←AR+1 mikro işleminden dolayı adres kaydedicideki değer bir artırılmış olacak ve orijinal değer bozulmuş olacaktır. Bu bozulmayı engellemek için ara bir eleman olan geçici kaydedici kullanıldı. Adım-3'de bu geçici kaydedicideki adres bilgisi adres kaydediciye transfer edilerek komutun bellekte göstermiş olduğu veriye ulaşılmış olunur. Geçici kaydediciden adres kaydedicisine atılmasındaki sebep, belleğin adresleyen birimin adres kaydedici olmasından kaynaklanmaktadır. Adım-4 ve Adım-5'de ise adres kaydedicinin bellekte göstermiş olduğu veri indis kaydedicisine atılarak komutun icra safhası bitirilerek mikroişlemcinin yeni bir komut işlemesine zemin hazırlanmıştır. Bu komut al-getir ve kodunu çöz evresi hariç toplam beş adımda yerine getirilmektedir. Bu komutun MML simülasyon programındaki mikro işlem adımlarının tasarlanması Şekil 4.36'da gösterilmiştir.



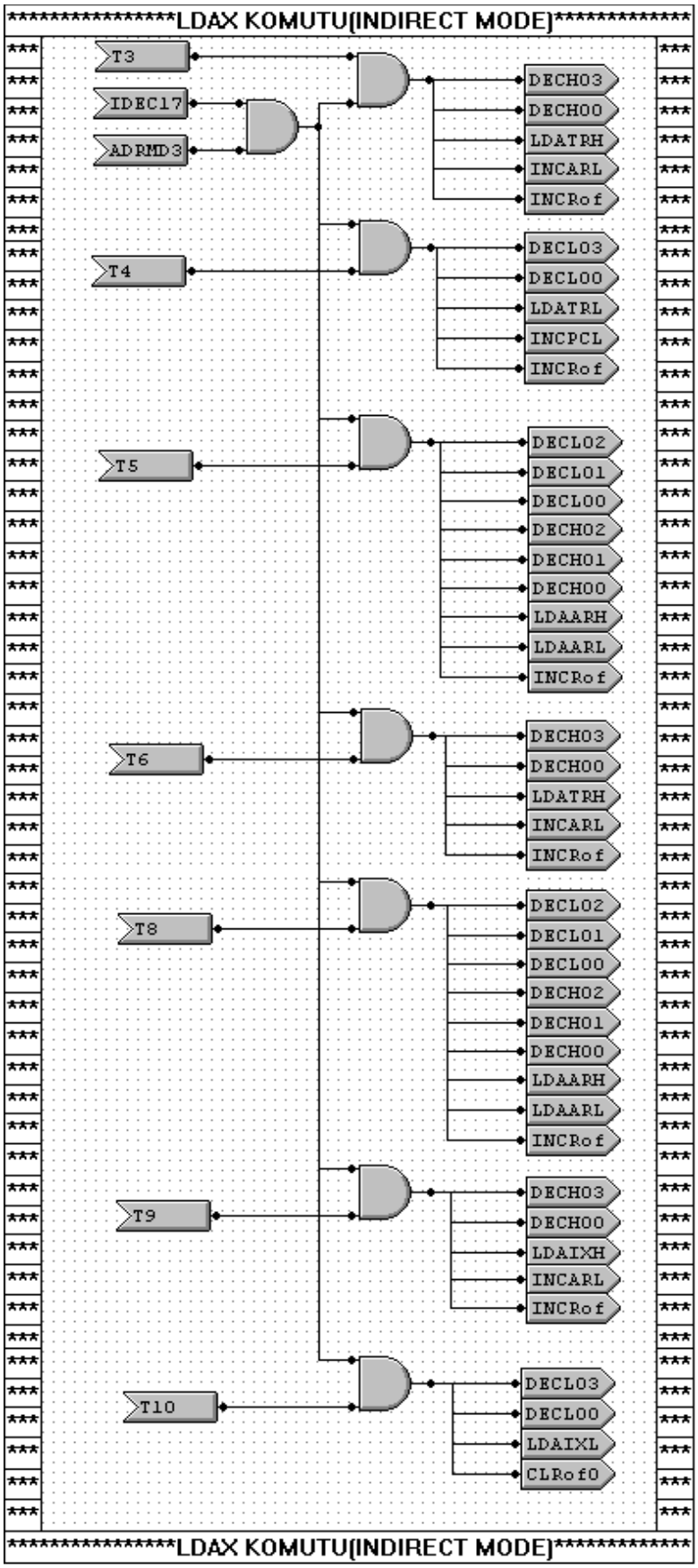
Şekil 4.36. LDAX(Direkt Mod) komutunun MML programındaki mikro işlem adımları

c) LDAX(Dolaylı Mod): İşlem kodu B1h olan bu komutun bellekte işgal ettiği byte sayısı üçtür. İşlem kodundan sonra gelen iki byte, LDAX komutuyla indis kaydedicisine yüklenecek değerin adresinin adresi temsil etmektedir. Bu komutun mikro işlem adımları Tablo 4.18'de gösterilmiştir.

Tablo 4.18. LDAX(Dolaylı Mod) mikro işlem adımları

Adım-1	T3* IDEC17*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
Adım-2	T4* IDEC17*ADRMD3	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
Adım-3	T5* IDEC17*ADRMD3	$AR \leftarrow TR$
Adım-4	T6* IDEC17*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
Adım-5	T7* IDEC17*ADRMD3	$TR_L \leftarrow M[AR]$
Adım-6	T8* IDEC17*ADRMD3	$AR \leftarrow TR$
Adım-7	T9* IDEC17*ADRMD3	$IX_H \leftarrow M[AR], AR \leftarrow AR+1$
Adım-8	T10* IDEC17*ADRMD3	$IX_L \leftarrow M[AR], SC \leftarrow 0$

İlk üç adım direkt adresleme modu kullanan LDAX komutuyla aynı işlemlerdir. Bundan sonra gelen üç adım da ilk üç adımın aynısıdır. Burada iki defa tekrar edilmesinin sebebi komutun işlem kodundan sonra gelen iki byte'ın verinin adresini değil, verinin adresinin hangi adreste olduğunu belirten verilerdir. Son iki adım ise hem derhal hem de direkt adresleme modunda olduğu gibi belirtilen adresteki verinin indis kaydedicisine aktarılma işlemleridir. Dolaylı moda sahip LDAX komutu al-getir ve kodunu çöz evresinden itibaren 8 çevrimde gerçekleştirilmektedir. Bu mikro işlem adımlarının MML programında tasarlanışı Şekil 4.37'de görülmektedir.



Şekil 4.37. LDAX(Dolaylı Mod) komutunun mikro işlemlerinin MML programında gerçeklenmesi

d) LDAX(İndis Mod): Bu komut C1h işlem koduna sahip olup bellekte 2 byte yer işgal etmektedir. İşlem kodundan sonra gelen ikinci byte veri, offset değeri olup adresleme modu indis veya göreceli olduğu zaman etkin adresi hesaplayan devreye iletilerek etkin adres hesaplanır. Bu moda sahip LDAX komutunun mikro işlem adımları Tablo 4.19'da verilmiştir.

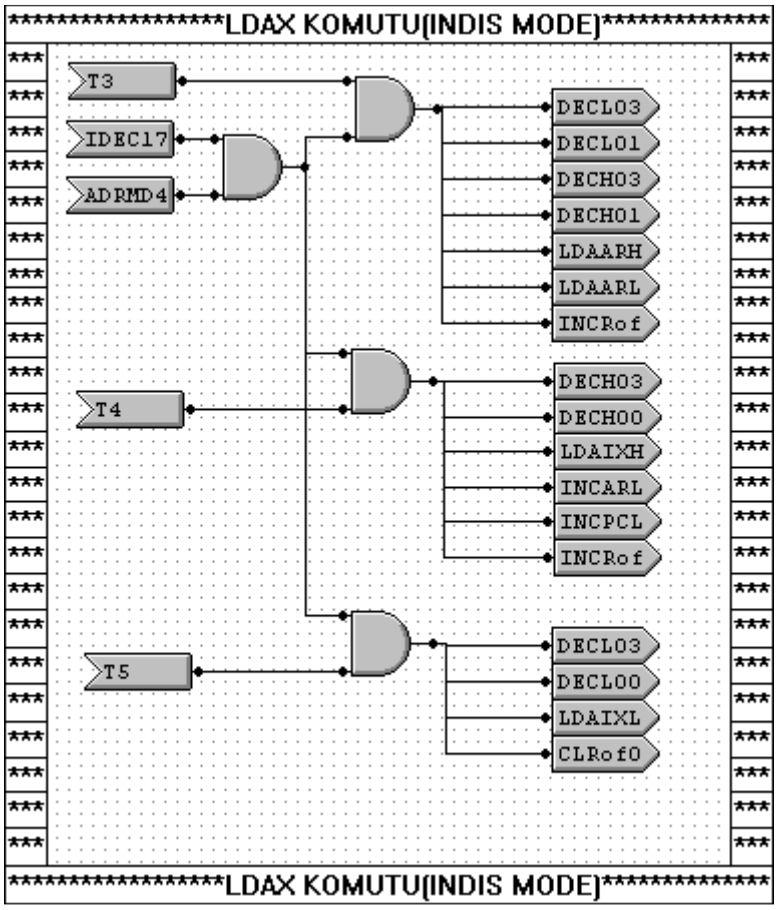
Tablo 4.19. LDAX(İndis Mod) komutunun mikro işlem adımları

Adım-1	T3* IDEC17*ADRMD4	AR←Etkin Adres
Adım-2	T4* IDEC17*ADRMD4	IX_L←M[AR], AR←AR+1
Adım-3	T5* IDEC17*ADRMD4	IX_H←M[AR], SC←0

Tablodan da açıkça görüldüğü üzere, ilk adımda hesaplanan etkin adres, adres kaydedicisine transfer edilir. Son iki adımda ise LDAX komutunun diğer adresleme mod türlerinde olduğu gibi adres kaydedicisinin bellekte göstermiş olduğu veri indis kaydedicisine transfer edilerek sayıcı sıfırlanır ve mikroişlemcinin bundan sonraki komuta konumlanmasına neden olur. Tablodan da görüleceği üzere indis moddaki LDAX komutunun yerine getirilmesi al-getir ve kodunu çöz evresinden sonra 3 çevrim zaman almaktadır. Bu moddaki LDAX komutunun MML programında tasarlanması Şekil 4.38'de görülmektedir.

4.9.1.3. Sıçrama ve dallanma komutları

Sıçrama ve Dallanma komutları 22 adet olup; bazı komutları şartsız olarak dallanabilen, bazıları ise bir şarta bağlı olarak dallanarak programın akışını değiştiren komutlardır. Bu komutların büyük bir çoğunluğu göreceli adresleme modunu kullanırken, ana programa dönüş komutları ise doğal adresleme modunu kullanmaktadır. Göreceli adresleme modunu kullanan komutların bellekte 2 byte'lık bir yer işgal ederken, doğal adresleme modunu kullanan dallanma komutları ise 1 byte'lık yer işgal etmektedir.



Şekil 4.38. LDAX(İndis Mod) komutunun mikro işlem adımlarının MML programında gerçeklenmesi

Göreceli adresleme modunu kullanan komutlardan şartsız dallanma komutu olan BSR(Branch to Subroutine) komutunu ve doğal adresleme modunu kullanan komutlardan da RTS(Return from Subroutine) komutunun mikro işlem adımları ayrıntılı olarak incelenecektir. Diğer sıçrama ve dallanma komutlarının mikro işlem adımları ise EK-G' de verilmiş olup, kullanıcı burada örnek olarak verilen komutların mikro işlem adımlarını gördükten sonra benzer şekilde EK-G'deki komutların da mikro işlem analizini yapabilir.

Tablo 4.20'de BSR komutunun mikro işlem adımları verilmiştir. Diğer komutların mikro işlem adımları Ek-C'de verilmiştir.

Tablo 4.20. BSR komutunun mikro işlem adımları

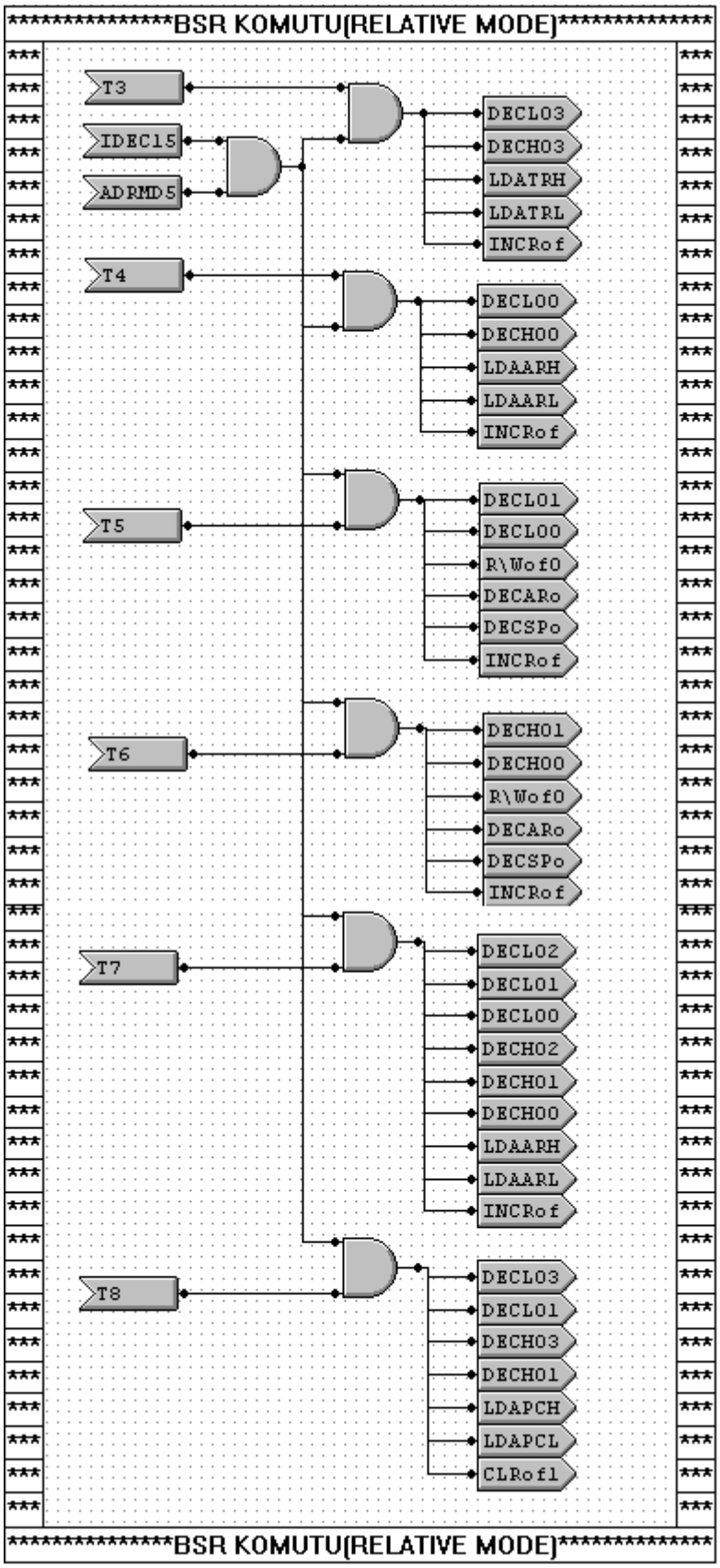
Adım-1	T3* IDEC15*ADRMD5	TR←AR
Adım-2	T4* IDEC15*ADRMD5	AR←SP
Adım-3	T5* IDEC15*ADRMD5	M_{SP}←PC_L, AR←AR-1, SP←SP-1
Adım-4	T6* IDEC15*ADRMD5	M_{SP}←PC_H, AR←AR-1, SP←SP-1
Adım-5	T7* IDEC15*ADRMD5	AR←TR
Adım-6	T8* IDEC15*ADRMD5	PC←Etkin adres, SC←0

İlk adımda al-getir ve kodunu çöz evresini göz önünde bulundurursak adres kaydedicisinde bulunan değer, BSR komutunun işlem kodundan sonraki kısmı işaret etmektedir. İlerleyen adımlarda yığın ile ilgili işlemler yapılacağından adres kaydedicinin içeriğindeki adres değeri geçici bir elemana transfer ediliyor. İkinci adımda ise bellekte yığın işlemleri için ayrılmış olan bölgenin başlangıç adresi adres kaydedicisine transfer edilir. Sonraki iki adımda ise dallanma komutu bittikten sonra dallanmadan sonraki komutun bellekte yerini gösteren program sayıcının değeri bellekte yığın denilen yere atılıyor. Dikkat edilirse program sayıcının ilk önce düşük anlamlı kısmı, daha sonra yüksek anlamlı kısmı yığına atılıyor. Bu tamamen tasarlanan bilgisayarın yığın işlemlerini Last in First Out(LIFO) prensibine göre yapılmasından kaynaklanmaktadır. Bu komut al-getir ve kodunu çöz evresi hariç toplam 6 çevrim zamanında icra edilmektedir. Bu mikro işlem adımlarının MML programında tasarlanmış hali Şekil 4.39'da görülmektedir.

Bir diğer sıçrama ve dallanma komutu olan RTS komutunun mikro işlem adımları Tablo 4.21'de verilmiştir.

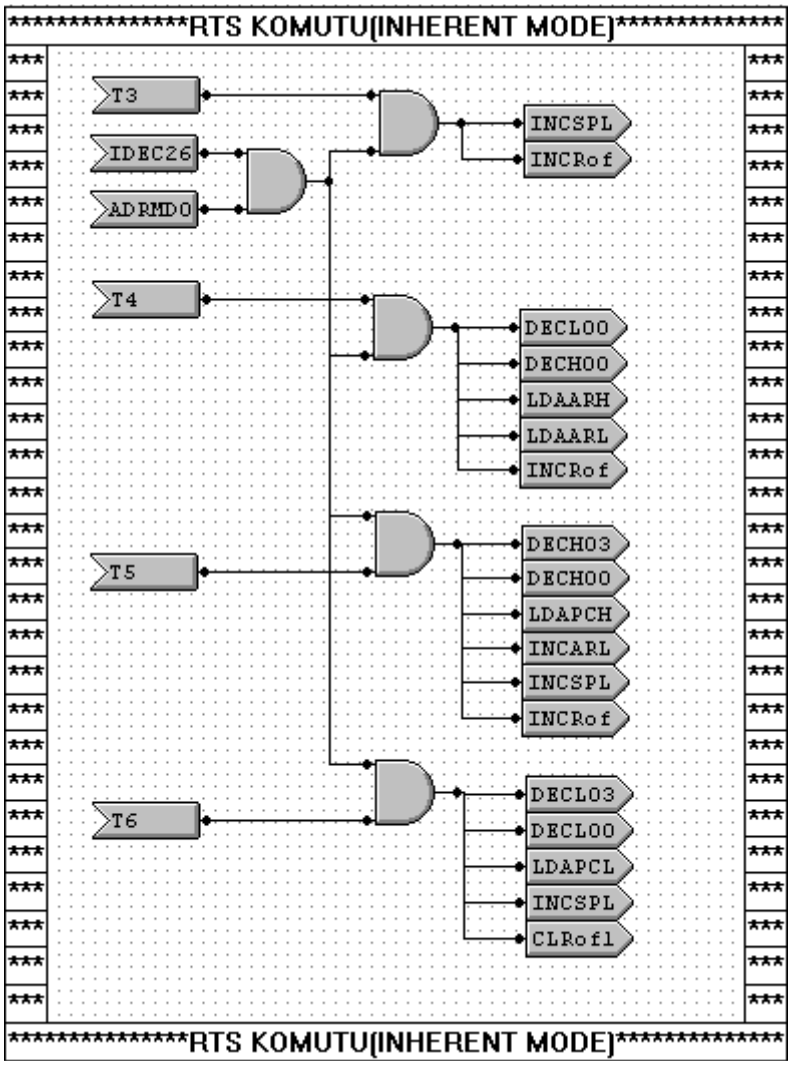
Tablo 4.21. RTS komutunun mikro işlem adımları

Adım-1	T3* IDEC26*ADRMD5	SP←SP+1
Adım-2	T4* IDEC26*ADRMD5	AR←SP
Adım-3	T5* IDEC26*ADRMD5	PC_H←M_{SP}, AR←AR+1, SP←SP+1
Adım-4	T6* IDEC26*ADRMD5	PC_L←M_{SP}, AR←AR+1, SP←SP+1, SC←0



Şekil 4.39. BSR komutunun mikro işlemlerinin MML programında tasarlanışı

İlk adımda yığın göstergeci bir artırılarak yığında kaydedilen ilk elamanın bulunduğu adrese ulaşılmış olur. Daha sonra bu yığın göstergecindeki değer adres kaydedicisine atılarak bellekle iletişim sağlanmış olur. BSR komutunda yapılan işlemleri hatırlayacak olursak yığına ilk önce program sayıcının düşük anlamlı kısmı atılmış daha sonra yüksek anlamlı kısmı atılmıştı. LIFO prensibine göre son atılan değer ilk alınacağından program sayıcının yüksek anlamlı kısmı ilk önce daha sonra düşük anlamlı kısmı alınarak komut icra edilmiş olur. Bu komut toplam 4 çevrim zamanında yerine getirilmektedir. Bu komutun MML programındaki mikro işlem adımları Şekil 4.40'da görülmektedir.



Şekil 4.40. RTS komutunun mikro işlemlerinin MML'de gerçeklenişi

4.9.1.4. Durum kod kaydedicisi komutları

Bu komutlar toplam 6 adet olup özellikle dallanma komutlarının gerçekleşmesinde önemli rol oynamaktadırlar. Bu komutların hepsi doğal adresleme modunu kullanıp al-getir ve kodunu çöz evresi hariç tek çevrim zamanında icra edilmektedirler. Örneğin CLC(Clear Carry) komutunun mikro işlem adımı şu şekilde gerçekleşmektedir. Diğer komutların mikro işlem adımları ise EK-D'de verilmiştir.

T3* IDEC19*ADRMD0 C←0, SC←0

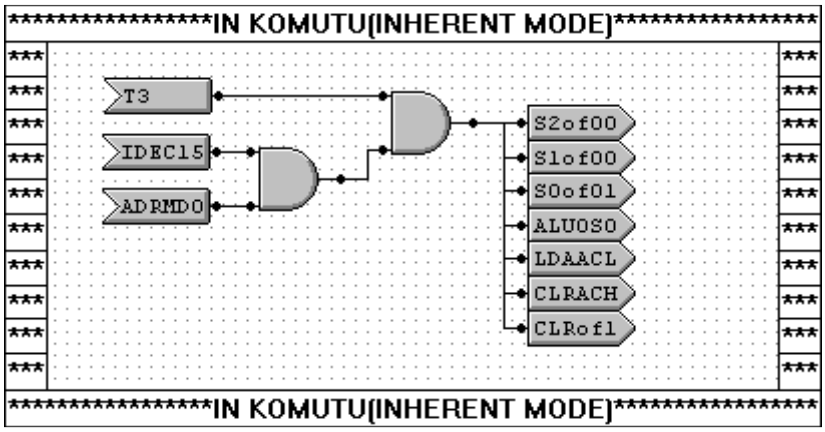
4.9.1.5. Giriş-Çıkış buyrukları ve kesmeler

Bir bilgisayarın çalışmasında dış çevre birimleriyle haberleşmesi zorunludur. Buyruklar ve veriler bir giriş biriminden gelmelidir. İşlem sonuçları ise çıkış birimi vasıtasıyla kullanıcıya iletilmelidir. Tasarlanan bu bilgisayarda giriş birimi olarak bir klavye, çıkış birimi olarak da ASCII display kullanılmıştır. Klavyeden gelen seri bilgiler INPR giriş kaydedicisine aktarılır. Çıkış birimine aktarılacak veriler ise OUTR çıkış yazacında tutulur.

Giriş-çıkış buyrukları bilginin AC nin içine ve AC den dışarıya aktarımı için gereklidir. Ayrıca bayrak bitleri de denetlenir. Ayrıca kesmelerde bu buyruklarla denetim yapılır. Tasarlanan bilgisayarda kullanılan giriş-çıkış buyrukları ve onların mikro işlemlerini sırasıyla inceleyelim.

IN : 0F h işlem koduna sahip tek bytelık bir giriş komutudur. Bu komut bilgiyi INPR kaydedicisinden bilgiyi alarak, AC nin düşük anlamlı 8 bitine aktarır. Ayrıca giriş bayrağını sıfırlar. IN giriş komutuna ait mikro işlem, tek adımdan oluşmaktadır. Bu mikro işlem adımının MML programında tasarlanmış hali Şekil 4.41'de verilmiştir.

T_3.IDEC15.ADRMD0: AC←INPR, SC←0



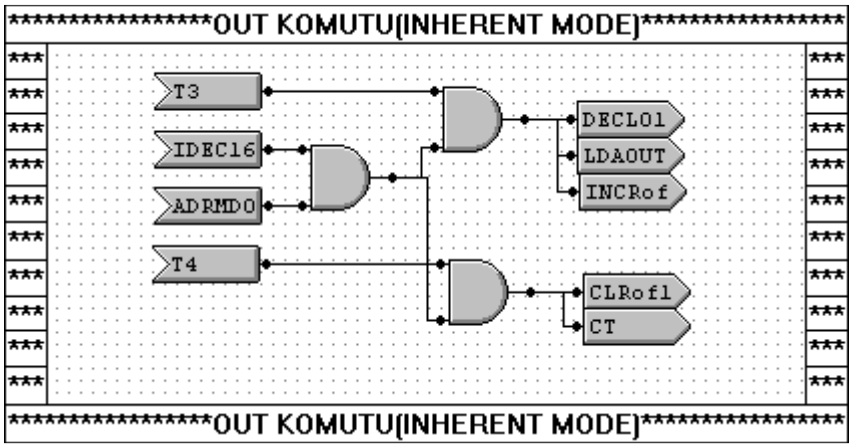
Şekil 4.41. IN komutunun MML programındaki mikro işlem adımları

Şekil 4.40 incelendiğinde, ilk adım olarak INPR yazacındaki verinin ALU üzerinden AC ye aktarılması için ALU'nun aritmetik işlem biriminin S_2, S_1, S_0 kontrol girişleri aktif edilir. Daha sonra ALU nun çıkışını, aritmetik işlem biriminin kullanabilmesi için gerekli izni verecek olan ALUOS0 kontrol girişi aktif edilerek INPR kaydedicisindeki verinin çıkışa, oradan da AC nin girişine gelerek veri hazır hale gelir. LDAACL kontrol girişi aktif edilerek girişteki veriler AC nin çıkışına iletilir. Son olarak sayıcı sıfırlanarak mikroişlemcinin bir sonraki komutu icra edebilmesi için sistemi al-getir ve kodunu çöz evresine gönderir.

OUT : 80 h işlem koduna sahip tek bytelık bir çıkış komutudur. Bu komut AC nin düşük anlamlı sekiz bitini alarak OUTR kaydedicisine aktarır ve çıkış bayrağını sıfırlar. OUT çıkış komutuna ait mikro işlem adımları Tablo 4.22'de ve bu adımların MML programında tasarlanmış hali Şekil 4.42'de verilmiştir. Buralardaki CT kontrol girişi ASCII display'in girişine gelen veriyi ekrana yazdırması için gerekli bir sinyaldir.

Tablo 4.22. OUT komutuna ait mikro işlem adımları

Adım-1	T_3.IDEC16.ADRMD0	OUTR←AC
Adım-2	T_4.IDEC16.ADRMD0	CT=1, SC←0



Şekil 4.42. OUT komutunun mikro işlem adımlarının MML de tasarlanması

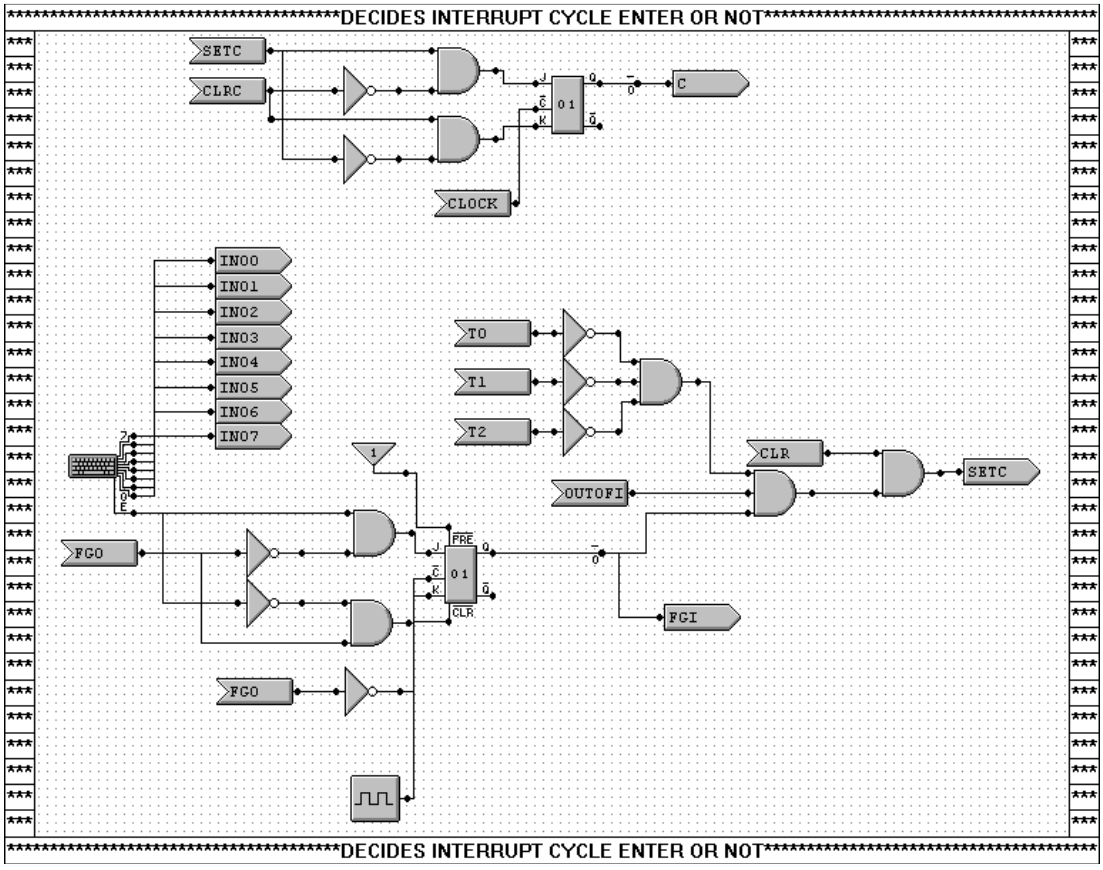
Program Kesme:

Giriş-çıkış buyrukları anlatılırken bayrak bitinden bahsedildi. Bilgisayar bu bayrak bitini kontrol ederek, bitin 1 yapıldığını görünce bilgi aktarımına başlar. Bu tip aktarım yeterince etkin değildir. Özellikle çevre birimi ile bilgisayar arasındaki aktarma oranı çok düşüktür. Bunun nedeni incelenirse; buyruk sürecini 1 µs de bitiren bir bilgisayar düşünelim. Bir de saniyede 10 karakter aktarabilen bir giriş-çıkış birimi düşünelim. Bu 100.000 µs de bir karakter eder. Bilgisayar her bir aktarımda bayrağı 50.000 defa kontrol eder. Faydalı bir iş yapılacağına, bayrak kontrolü ile zaman kaybedilmektedir[8].

Programlama ile denetim için bir alternatif yol, dış birimin aktarıma hazır olduğunu bilgisayara hazır vermesidir. Bu arada bilgisayar başka işlerle uğraşabilir. Bu tip aktarım kesme yöntemini kullanır. Bilgisayar bir programı çalıştırırken, bayrakları kontrol etmez. Bir bayrak 1 yapıldığında, icra etmekte olduğu programın çalışması durdurulur ve bayrağın 1 yapıldığı haber verilir. Bilgisayar yapmakta olduğu işi anında bırakır ve giriş-çıkış aktarımı için gerekli önlemleri alır. Giriş-çıkış işini bitirince kesme gelmeden önceki programa geri döner ve kaldığı yerden devam eder.

Durum kod kaydedicisinde(CCR) bulunan kesme flip-flopu iki komut ile 1 veya 0 yapılabilir. STI(87 h) komutu ile bilgisayar kesme yapabilir duruma geçer. CLI(86 h) komutu ise bilgisayarın kesme durumuna geçmesini engeller.

Tasarlanan bilgisayar ya kesme çevrimindedir ya da buyruk çevrimindedir. Bu bilgisayarda C flip-flopu kullanılmıştır. C=1 ise bilgisayar kesme çevriminde, C=0 ise normal buyruk çevrimindedir. Buyruk çevrimi boyunca, kesme flip-flopu denetim birimi tarafından gözlenir. Eğer sıfır ise programcı kesmeleri kullanmak istemiyordur. Dolayısı ile denetim bir sonraki buyruk sürecine geçer. Eğer kesme flip-flopunun çıkışı(OUTOFI) 1 ise denetim bayrak bitlerine bakar. Eğer her iki bayrak ta 0 ise giriş ve çıkış birimlerinin hazır olmadıkları anlaşılarak denetim bir sonraki buyruk çevrimine geçer. Eğer bayraklardan bir tanesi 1 ise denetim kesme sürecine geçer. Bu sürecin MML programında tasarlanışı Şekil 4.43'de görülmektedir.



Şekil 4.43. Kesme çevrimine karar verildiği bölüm

Bilgisayar kesme sürecine girmeden önce, işlemci kaydedicilerin içeriğini kaydetmek zorundadır. Kesme çevriminden dönüşte kaldığı yerden devam edebilmesi için bu işlemci yazaçlarının içeriklerini bellekte yığın denilen bölgeye kaydeder. Bu yığın atma işlemini şu sırada gerçekleştirir.

1. Program sayıcının(PC) düşük anlamlı 8 biti
2. Program sayıcının(PC) yüksek anlamlı 8 biti
3. İndis kaydedicisinin(IX) düşük anlamlı 8 biti
4. İndis kaydedicisinin(IX) yüksek anlamlı 8 biti
5. Akümülatörün(AC) düşük anlamlı 8 biti
6. Akümülatörün(AC) yüksek anlamlı 8 biti
7. Durum kod kaydedicisi(CCR)

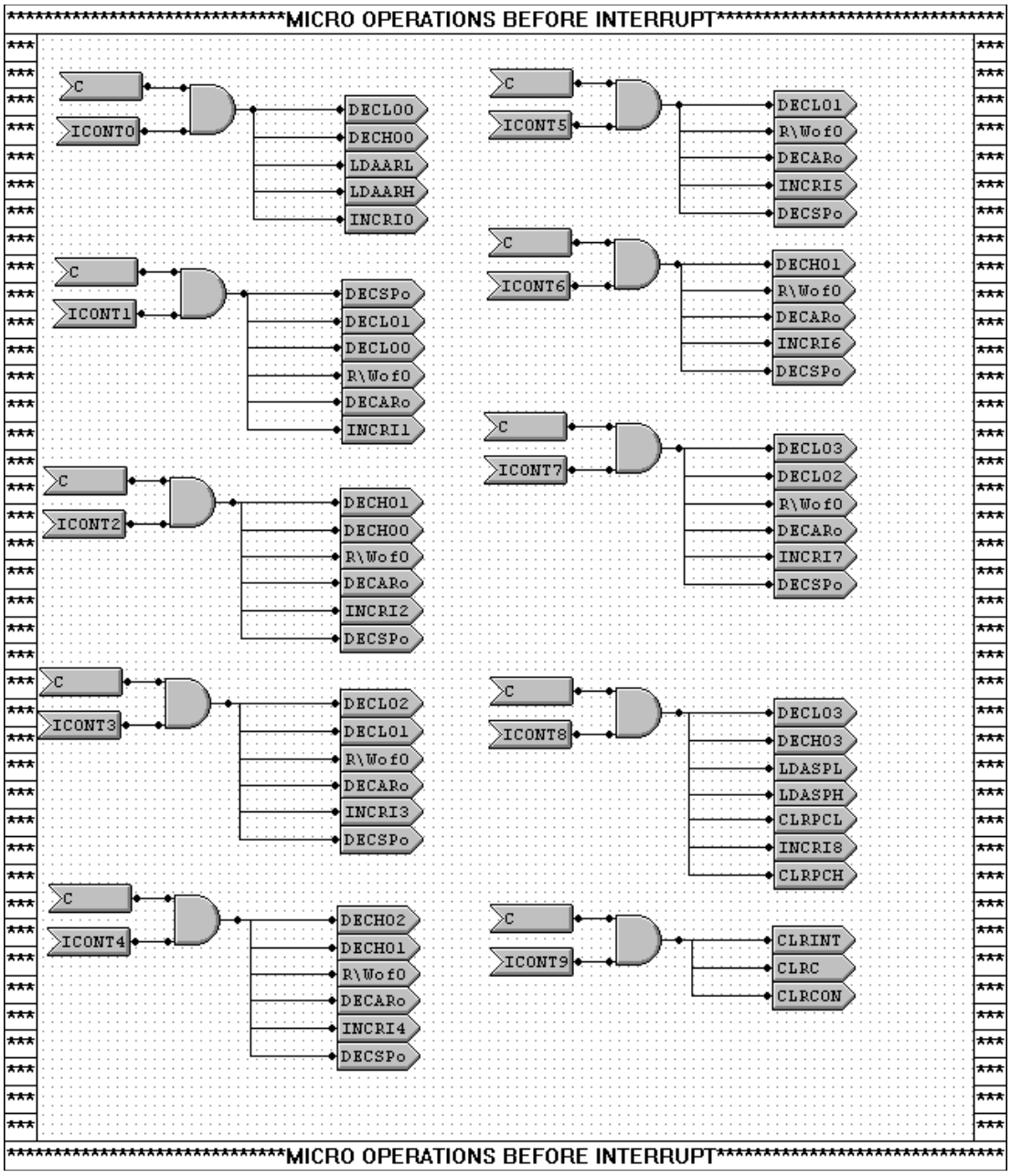
Kesme sürecine girmeden yapılan işlemlerin MML programında tasarlanışı Şekil 4.44'de verilmiştir. Bu işlemleri yaptıktan sonra belleğin ilk üç gözünde yer alan BRA FFFDh komutunu işleyerek belleğin son üç gözünde yer alan giriş kesme programını icra ederek programa kaldığı yerden devam eder.

4.10. Buyruk Süreci

Bu bölüme kadar komutun tasarımı, komutun icra edilebilmesi için gerekli mikro işlem adımlarından bahsedildi. Bellekte bulunan program bir dizi buyruktan oluşmaktadır[32]. Bu programın icrası, her buyruk için bir sürecin sonunda mümkün olur. Her bir buyruk süreci bir takım alt parçalardan oluşur. Bunlara alt süreçler veya süreçler denir. Tasarlanan bu bilgisayarda her bir buyruk aşağıdaki süreçlerden geçmektedir.

1. Bir buyruğun bellekten alınıp getirilmesi(FETCH)
2. Buyruk kodunun çözülmesi(DECODE)
3. Buyruğun icrası

Üçüncü adımın tamamlanmasından sonra denetim tekrar birinci adım olan al-getir evresine döner. Bir sonraki buyruğu alır getirir, kodunu çözer ve buyruğu icra eder. Bu işlem HALT komutu gelene kadar devam eder.



Şekil 4.44. Kesme sürecine girmeden önce yapılan işlemlerim MML de gerçeklenmesi

4.10.1. Al-Getir ve kodunu çöz evresi

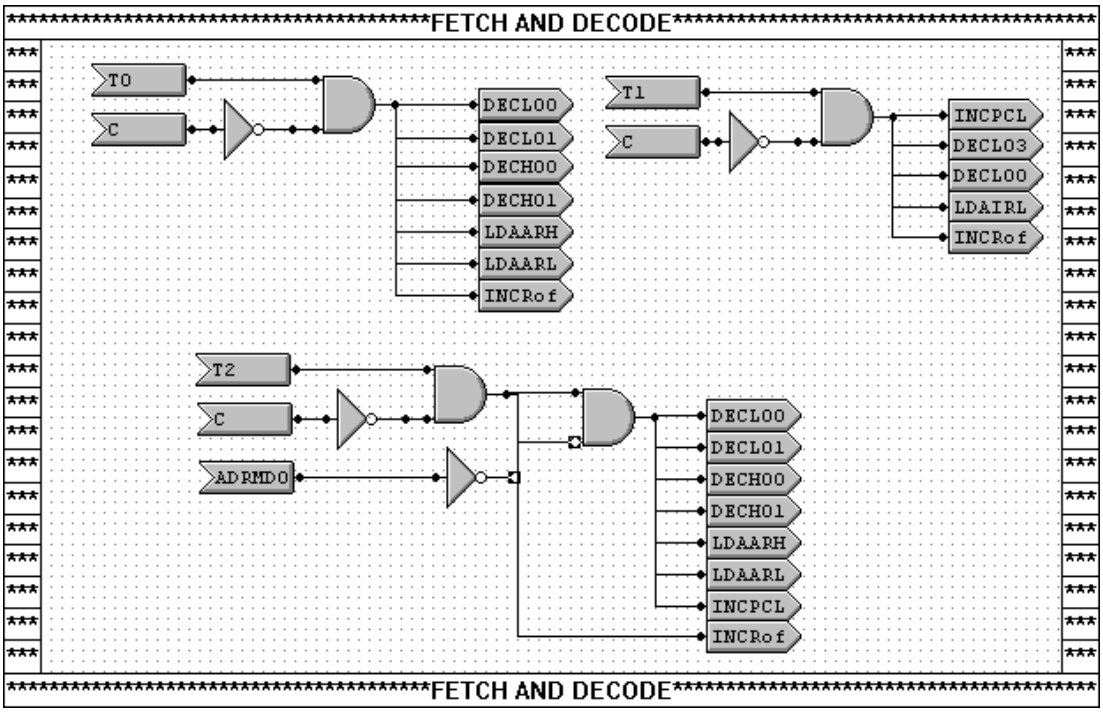
İlk olarak program sayıcı ilk buyruğun adresi ile yüklenir. Sıra sayıcı sıfırlanır. Böylece T_0 zaman sinyali elde edilir. Her bir saat vuruşundan sonra sıra sayıcı bir arttırılır. Böylece zaman sinyalleri T_0, T_1, T_2 …… diye gider. Al-getir ve kodunu çöz evrelerine ait mikro işlemler aşağıdaki mikro işlem adımlarıyla belirlenir.

T_0 : AR←PC

T_1 : IR←M[AR], PC←PC+1

$T_2.\overline{ADRMD0}$:AR←PC, PC←PC+1, IDEC00…..IDEC31←kodu çöz IR(0-3,7), ADRMD0…..ADRMD5←IR(4-6)

Belleğin adres girişlerine sadece AR bağlı olduğundan adresin önce PC den AR ye aktarılması gerekir. Bu iş T_0 zamanında olur. Bellekten okunan buyruk komut kaydedicisine(IR) yerleştirilir. Bu iş T_1 zamanında olur. Bu zaman diliminde PC bir artırılarak, eğer adresleme modu ADRMD0 ise yani doğal adresleme moduna sahip bir komut ise, program sayıcının(PC) bir sonraki komuta konumlanması sağlanmış olur. T_2 zaman diliminde ise komut kaydedicisine yerleştirilen komutun çözülmesi aşaması gerçekleşmiş olur. Komut kaydedicindeki(IR) gerekli bitler zamanlama ve denetim bölümünde anlatıldığı üzere gerekli kod çözücülerden geçirilerek adresleme modu çeşidi ve komut tipi öğrenilmiş olur. Eğer adresleme modu ADRMD0 değilse PC bir artırılarak program sayıcının bir sonraki komutu konumlanması sağlanmış olur. Al-getir ve kodunu çöz evresinin MML programında tasarlanmış hali Şekil 4.45'de görülmektedir.



Şekil 4.45. Al-getir ve kodunu çöz evresi

Şekil 4.44 de, al-getir ve kodunu çöz evresinde yukarıda anlatılanlardan farklı olarak bir "C sinyali görülmektedir. Buradaki bu "C" sinyali Giriş-Çıkış komutları bölümünde anlatılmış olmakla beraber kesme evresinde olup olmadığını gösteren bir sinyaldir. Şekil 4.45 de görüleceği üzere T_0 zaman diliminde 8 bitlik iki parçadan oluşan 16 bitlik veri yolunu, program sayıcının kullanabilmesi için gerekli izinlerin verilmiş olması gerekir. Bunun için Ortak veri yolu bölümünde anlatıldığı üzere PC nın veri yolunu kullanabilmesi için hem düşük anlamlı veri yolunu seçecek hem de yüksek anlamlı veri yolunu seçecek kod çözücüye 0011 sinyalini göndermesi gerekir. Bu sinyal gönderildikten sonra veri yolu PC nın emrine verilerek, PC daki veri, veri yolunda dolaşmaya başlayacaktır. Dolaşan bilgiyi AR nin alabilmesi için LDAARH ve LDAARL kontrol girişlerini aktif etmesi gerekir. Bu işler tamamlandıktan sonra INCR sinyali ile sıra sayıcı bir artırılarak T_1 zaman dilimine geçmesi sağlanmış olur.

T_1 zaman diliminde, PC bir artırılarak eğer komutun adresleme modu ADRMD0 ise bir sonraki komutu göstermesi sağlanır. Bunun için PC nın INCPCL kontrol girişine sinyal gönderilerek bu işlem sağlanmış olur. T_0 zaman diliminde PC daki veri AR ye aktarıldığından dolayı, belleğin o adresteki komut bilgisinin IR ye aktarılması gerekir. Bu aktarma işlemleri gerçekleştirilirken veri yolunu kullanacak olan belleğin gerekli izinleri alması gerekir. Bunun için veri yolu iznini gerçekleştiren kod çözücüye 1001 bilgisini göndererek bu izni alır. Bu izni aldıktan sonra AR nin bellekte göstermiş olduğu komut, veri yoluna serbest bırakılmış olur. Veri yolunda dolaşan bu bilgi LDAIRL kontrol girişi aktif edilerek komut kaydedicine alınmış olur. Ve tekrar INCR sinyali aktif edilerek sıra sayıcının bir sonraki zaman dilimine geçmesi sağlanmış olur.

Son olarak T_2 zaman diliminde ise, komutun adresleme modu ve tipi çözüldükten sonra, eğer adresleme modu doğal adresleme modu değilse INCPCL kontrol sinyali aktif edilerek PC nın bir sonraki komutu göstermesi sağlanmış olur. Eğer doğal adresleme modu ise zaten bir önceki zaman diliminde PC bir sonraki komutu göstermesi sağlanmıştı. Bu zaman diliminde komuttan sonra gelen verinin alınabilmesi için PC nın AR ye aktarılması işlemleri gerçekleşir. Bu işlemlerin gerçekleşebilmesi için gerekli izinler T_0 zaman diliminde anlatılmıştı. Ve sıra sayıcı bir artırılarak komutun icra safhasına geçilmesi sağlanmış oluyor.

BÖLÜM 5. UYGULAMA VE SONUÇLAR

Bilindiği gibi bir bilgisayarın komut kümesinin tamlığı aşağıda maddeler halinde verilen özellikleri taşıyorsa komut kümesi tamdır denir[32].

- Aritmetik, mantık ve kaydırma buyrukları
- Verinin bellek ve yazaçlar arasında aktarılmasını sağlayacak komutlar
- Durumu belirleyen komutlarla program denetim komutları
- Giriş-çıkış komutları

Tasarlanan bilgisayar bir bilgisayarın tam olma koşulunu sağlamaktadır. Bu nedenle bu bölümde yukarıdaki dört maddede yer alan örnekler yapılacak ve her bir örnekte bilgisayarın genel ve özel amaçlı kaydedicilerin durumu adım adım gözlenecektir..

Örnek-1

Belleğin verilerin saklandığı bölge olan veri segmentinde yer alan bir veriyi $f(x) = x^2 + x + 1$ fonksiyonuna tabi tutarak yine aynı yere kaydeden bir örneği bilgisayarda yapmaya çalışalım.

Bölüm 4.7'de bellek organizasyonu yapılırken belleğin de00 h adresinden itibaren 4 KB lık bellek alanı veriler için ayrılmıştı. Bu segmentte yer alan verilerden ilki 0002 h verisi olsun.

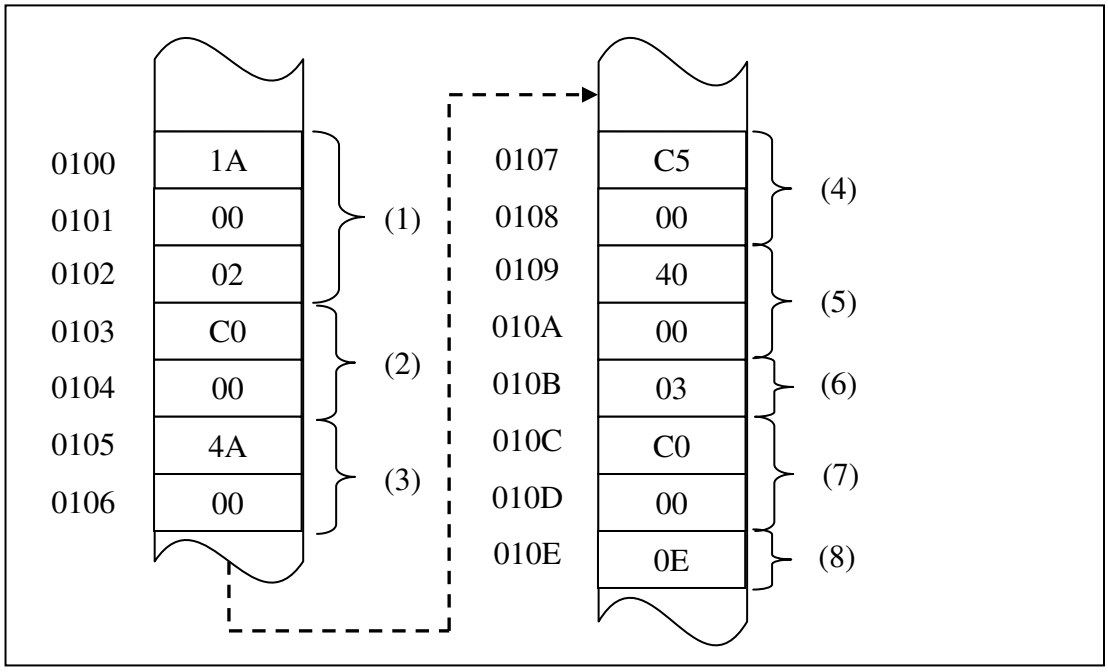
İstenilen program şu şekilde karşımıza çıkacaktır:

```
LDA #0002 h     \\ Veri ilk önce akümülatöre atılıyor.(1)
STA *00 h       \\ Veri segmentin ilk gözüne saklanıyor.(2)
\\=====Programın asıl başlangıç yeri==========\\
LDA *00 h       \\ Veri bellekten alınıp akümülatöre atılıyor.(3)
MUL *00 h       \\ Kendisiyle çarpılıyor(x^2)(4)
```

Veri segmentinde veri olmadığından ilk olarak veri atıldı.

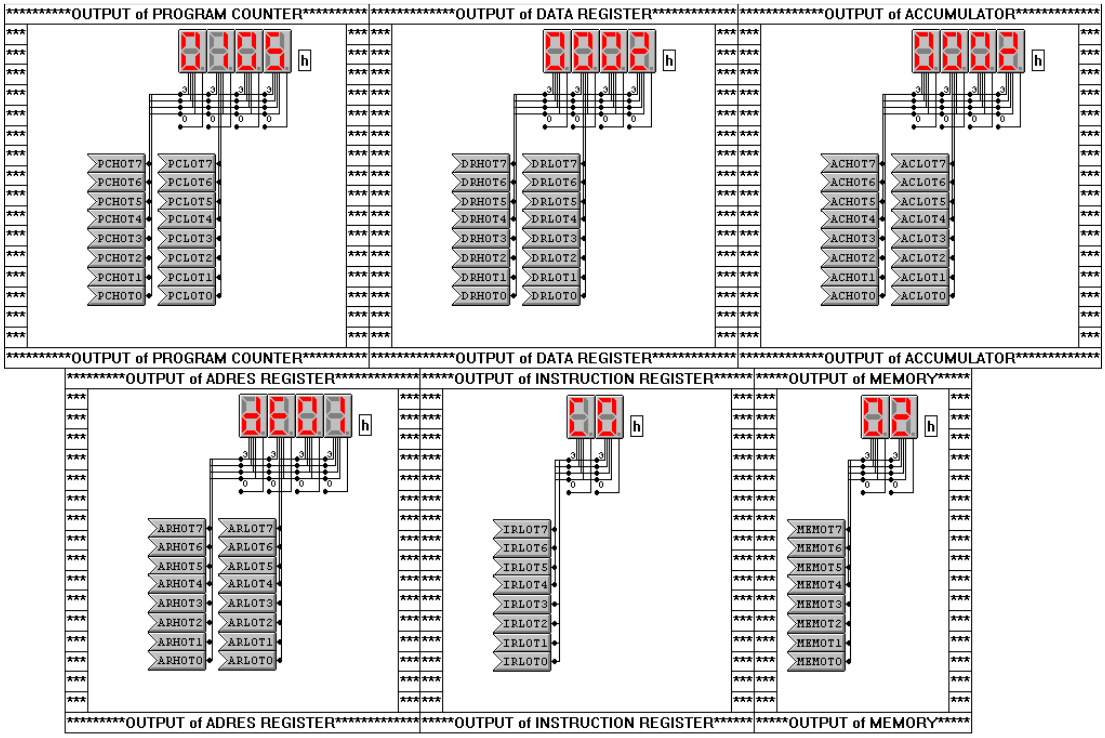
ADD *00 h \\ Kendisiyle toplanıyor(x^2+x)(5)
INCR \\ Bir fazlası alınıyor.(x^2+x+1)(6)
STA *00 h \\ Fonksiyondan geçirilen veri aynı yere saklanıyor.(7)
HLT \\ Programdan çıkılıyor.(8)

Bu yazılan program, bilgisayar için tasarlanan ve Ek-K'da tanıtılan assembler programında yazıldıktan sonra "Yükle" butonuna basıldığında ikili koda çevrilecektir. Çevrilen programın bellekteki görüntüsü Şekil 5.1'de verilmiştir.

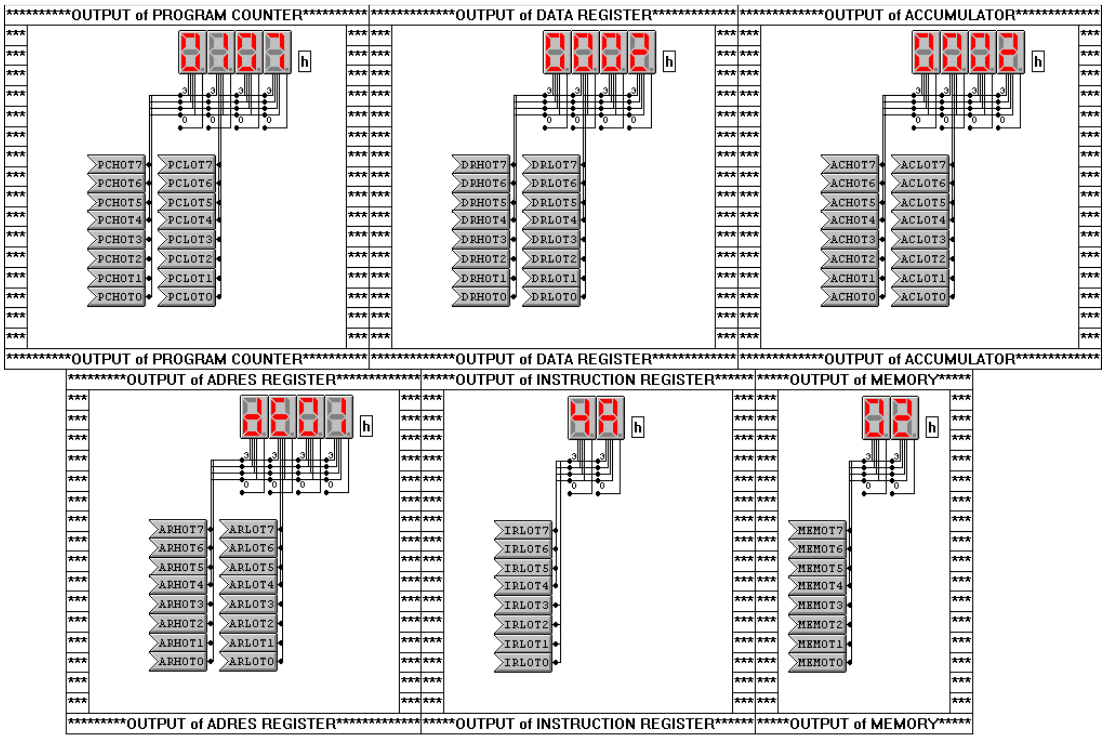


Şekil 5.1. Programın ikili kodun belleğe yerleşmiş hali

Program çalışmaya başladığında yani 3. adımdayken ilgili kaydedicilerin durumu Şekil 5.2'de görülmektedir. Programın 3.adımında LDA *00h komutu indis adresleme moduna sahip olduğundan, işlem kodundan sonra gelen 00h değeri, indis kaydedicisinin değerine eklenerek etkin adres hesaplanır. Bu hesaplanan etkin adresteki veri akümülatöre atılır. Bu komutun işletilmesinden sonraki kaydedicilerin durumu Şekil 5.3'de verilmiştir

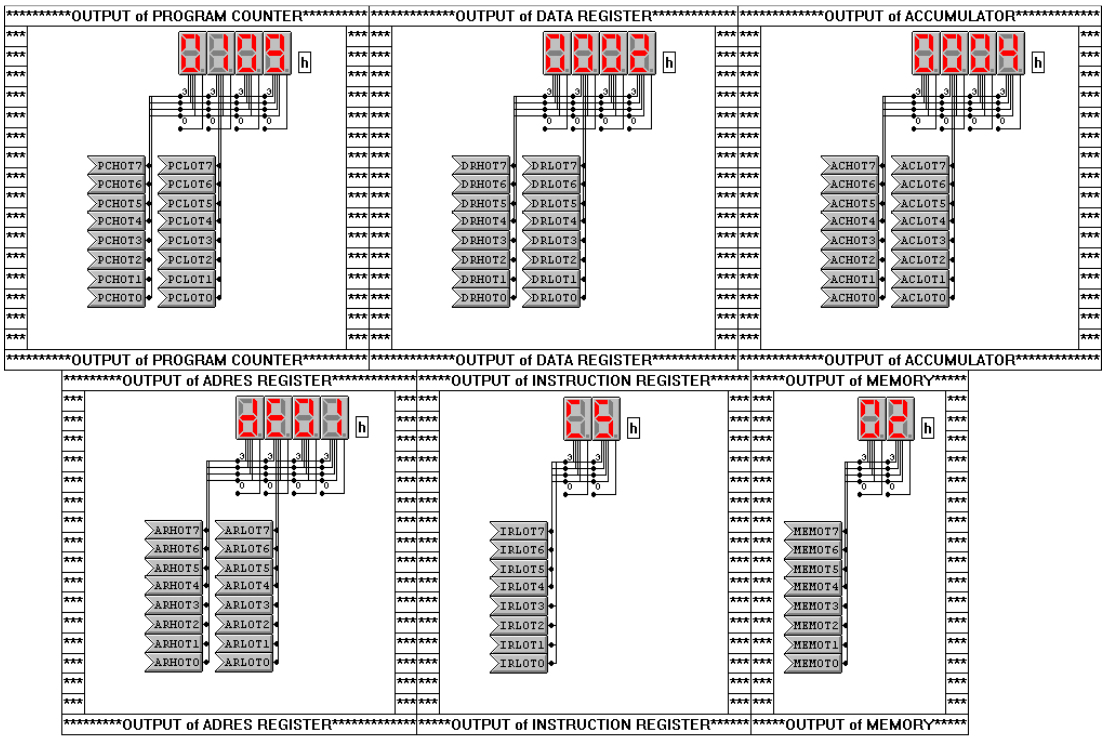


Şekil 5.2. Programın işletilmesi aşamasındaki ilgili kaydedicilerin durumu



Şekil 5.3. Programın 3.adım bitiminde kaydedicilerin durumu

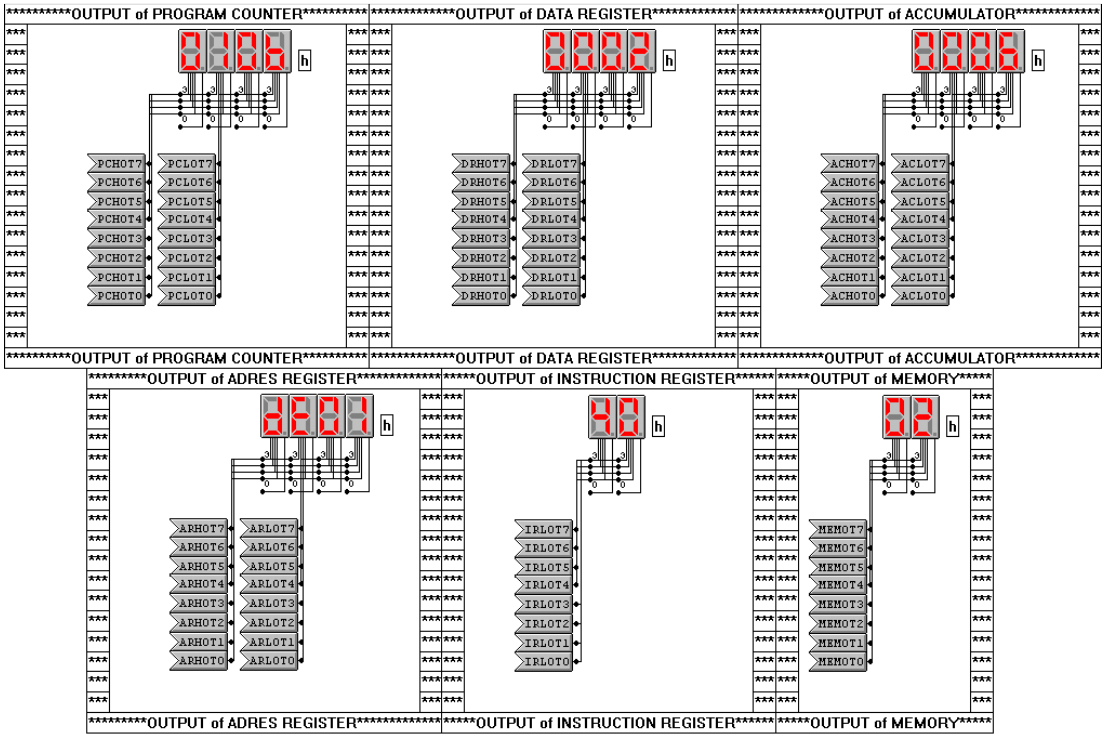
Dördüncü adımda yer alan MUL *00h komutu bir önceki adımda akümülatöre atılan değer ile indis kaydedicisinin göstermiş olduğu ilk değer ile çarpılarak akümülatöre atılır. Bu işlem adımı bittikten sonra oluşan kaydedici durumları Şekil 5.4'te verilmiştir.



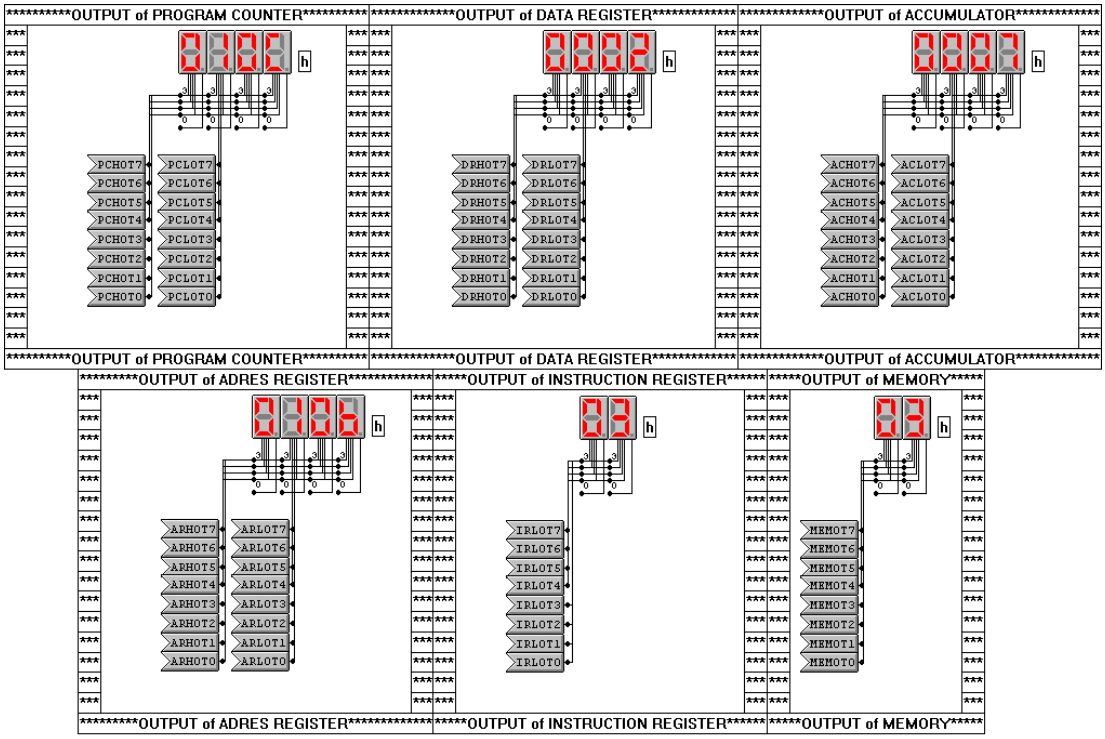
Şekil 5.4. Programın 4.adım bitiminde kaydedicilerin durumu

Programın 5.adımında ADD *00h komutu indis adresleme moduna sahip olduğundan, işlem kodundan sonra gelen 00h değeri, indis kaydedicisinin değerine eklenerek etkin adres hesaplanır. Bu hesaplanan etkin adresteki veri ile akümülatördeki veri toplama işlemine tabi tutularak sonuç akümülatöre atılır. Bu komutun işletilmesinden sonraki kaydedicilerin durumu Şekil 5.5'de verilmiştir.

Altıncı adımda ise INCR komutu ile akümülatördeki verinin bir fazlası alınarak tekrar akümülatöre atılır. Bu komutun işletilmesinden sonraki kaydedicilerin durumu Şekil 5.6'de verilmiştir.



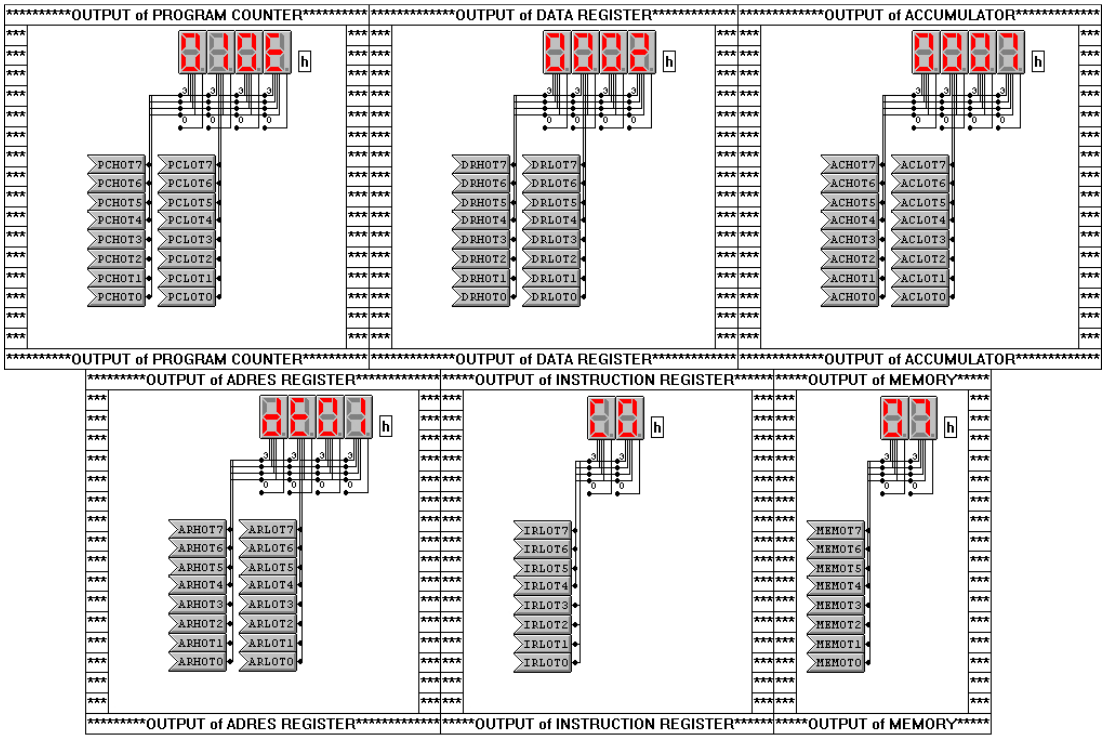
Şekil 5.5. Programın 5.adım bitiminde kaydedicilerin durumu



Şekil 5.6. Programın 6.adım bitiminde kaydedicilerin durumu

Programın yedinci adımında, artık istenen değerin $f(x) = x^2 + x + 1$ fonksiyonundan geçirildikten sonra çıkan değer tekrar aynı yere kaydediliyor. Bu adım bitiminde kaydedicilerin içerikleri Şekil 5.7'de görülmektedir. Son adımda ise

HLT komutu ile simülasyon durarak programın akışı bitirilir. Son adımda ise HLT komutu ile simülasyon durarak programın akışı bitirilir.



Şekil 5.7. Programın 7.adım bitiminde kaydedicilerin durumu

Örnek-2

İki elemanlı bir dizinin ilk elemanı ikinci elemanından büyük veya eşitse; ekrana 10 defa “B” karakteri yazdıracak, değilse 5 defa “K” karakteri yazdıran bir programı düşünelim.

Bu örnekte; tasarlanan bilgisayarın çıkış, döngü ve karşılaştırma komutlarını yerine getirebildiğini gösterelim. Bu bilgisayar için yazılan assembler programında(Bkz. Ek-K), bahsedilen örneğin programını yazarsak şu şekilde olacaktır.

Takdir edilmelidir ki dizinin elemanları uzun bir program dizisinde bir yerlerde elde edilmiş olup daha önceden veri segmentinde hazır bulunacaklardır. Bu sebepten dolayı, bir önceki örnekte yer alan veri segmentine veri transfer etme olayını burada göz ardı ettiğimizi varsayalım. Programın ana kısmı şöyle olacaktır.

```
LDA #000A h
STA %04 h
LDA #0005 h
STA %06 h
```

Döngü değişkenleri olan 10 ve 5 veri segmentinin gerekli gözlerine atılıyor.

```
LDA %00 h   \\ Veri segmentinde yer alan ilk eleman akümülatöre atılıyor.(1)
CMP %02 h   \\ Veri segmentindeki ikinci veri ile akümülatör kıyaslanıyor.(2)
BGE *0A h   \\ Eğer sıfırdan büyük veya eşit ise BUYUK alt programına
            dallanılıyor.(3)
KUCUK:      LDA #004B h \\ "K" Karakteri akümülatöre gönderiliyor.(4)
            OUT         \\ "K" karakteri ekrana basılıyor.(5)
            LDA %06 h
            DECR
            STA %06 h
```

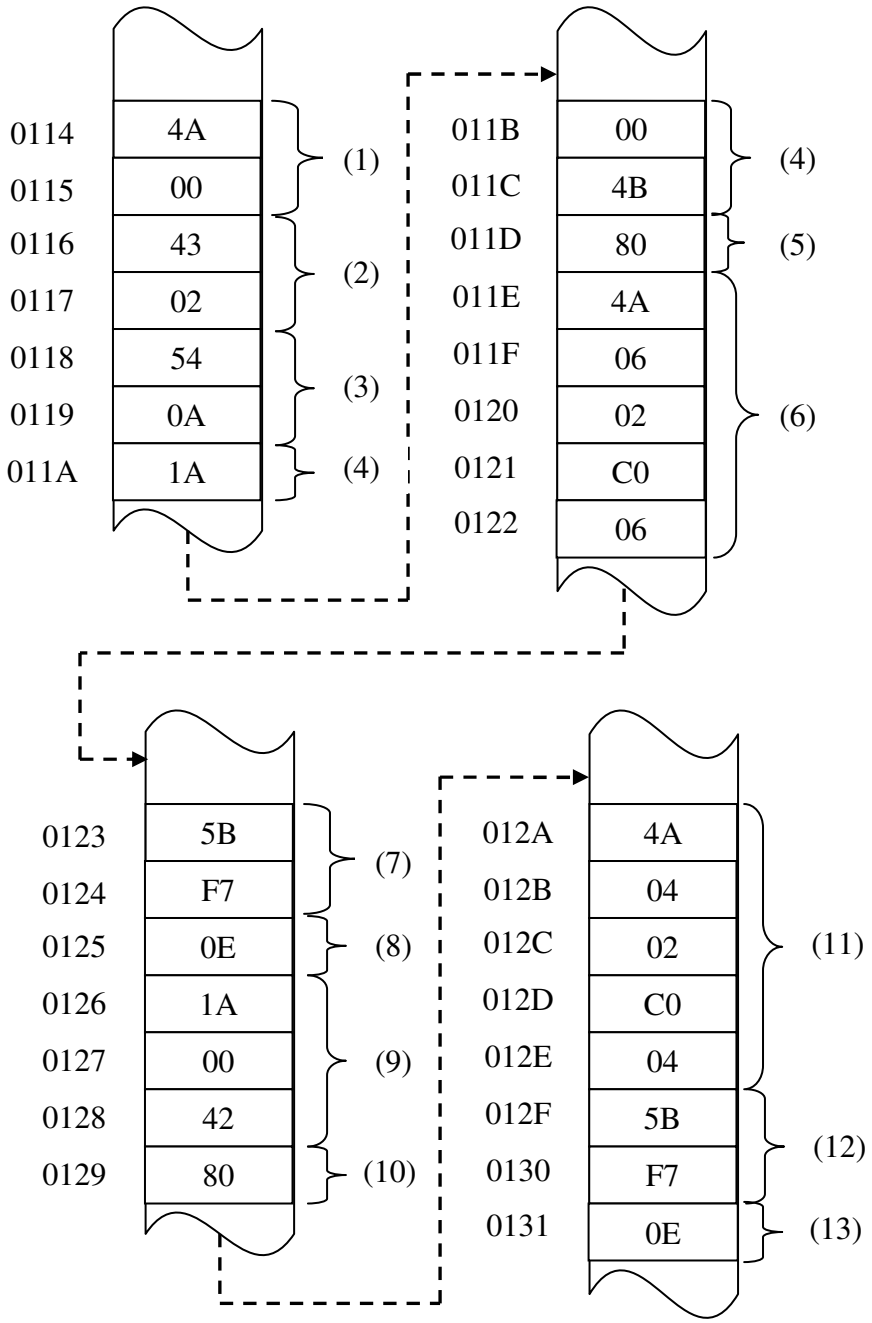
Döngü değişkeni olan 5 verisi bir azaltılıyor.(6)

```
            BNE *F7   \\ Döngünün sıfıra ulaşıp ulaşmadığı kontrol ediliyor.(7)
            HLT       \\ Programdan çıkılıyor.(8)
BUYUK:      LDA #0042 h  \\ "B" Karakteri akümülatöre gönderiliyor.(9)
            OUT          \\ "B" karakteri ekrana basılıyor.(10)
            LDA %04 h
            DECR
            STA %04 h
```

Döngü değişkeni olan 10 verisi bir azaltılıyor.(11)

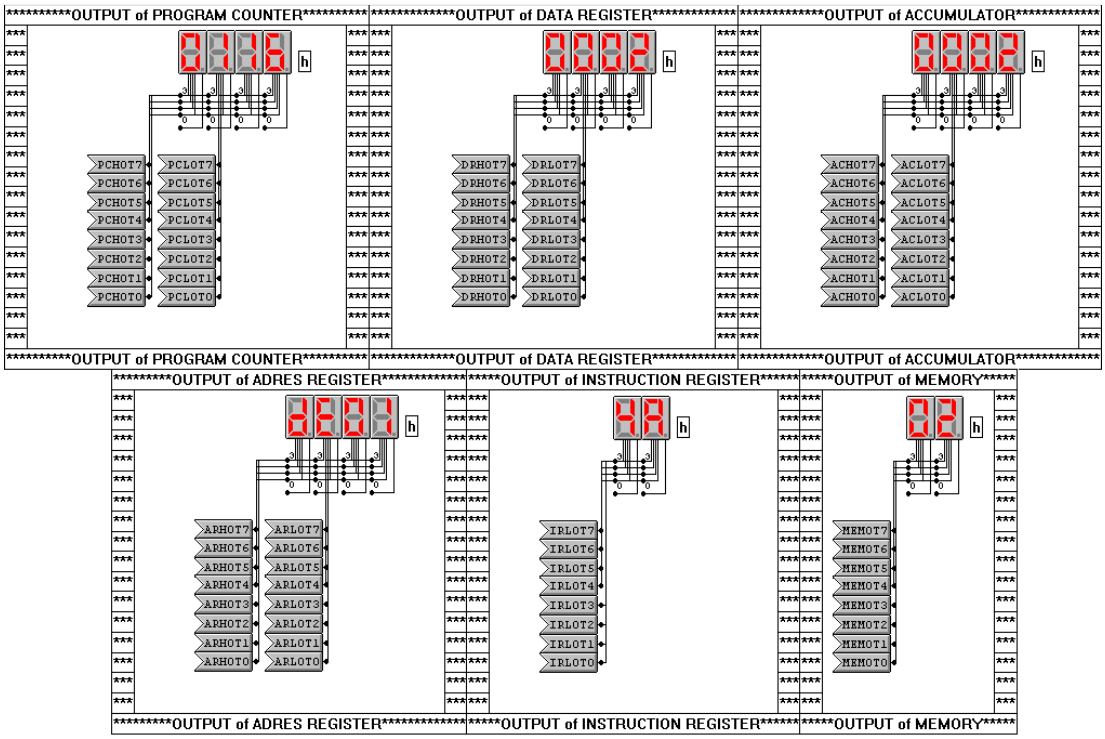
```
            BNE *F7 h  \\ Döngünün sıfıra ulaşıp ulaşmadığı kontrol ediliyor.(12)
            HLT        \\ Programdan çıkılıyor.(13)
```

Bu yazılan program ikili koda çevrildikten sonra bellekte alacağı düzen Şekil 5.8'de görülmektedir.



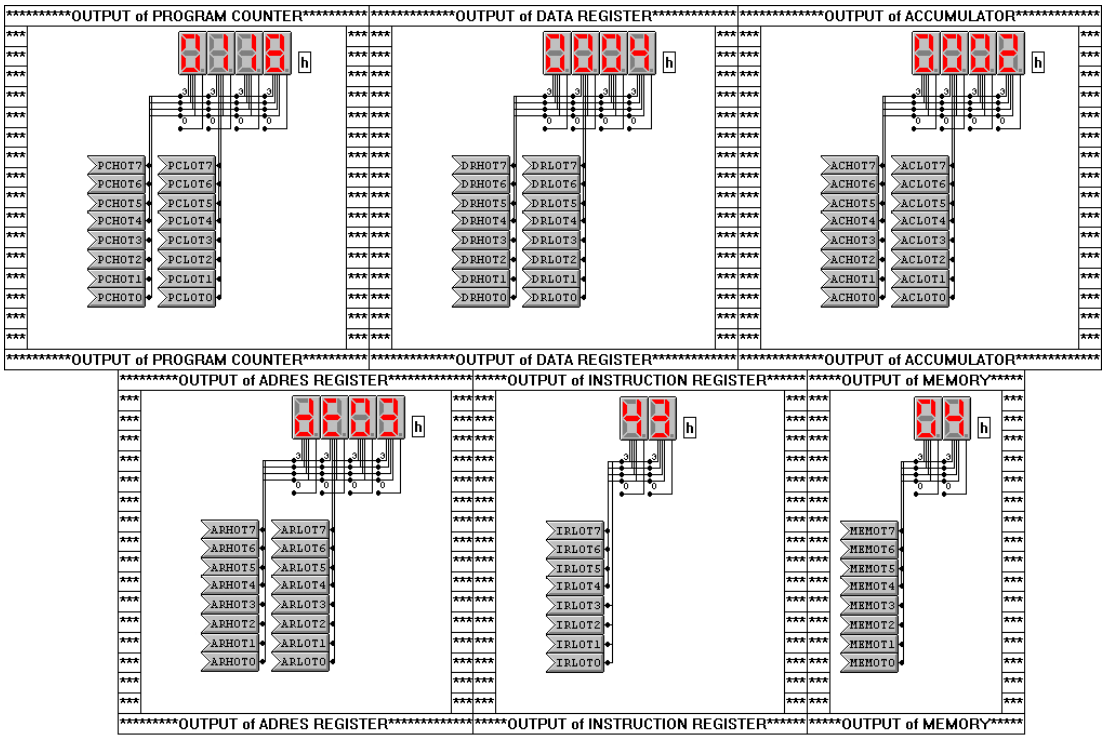
Şekil 5.8. Programın ikili kodunun belleğe yerleşmiş hali

Programı A=[2, 4] olan bir dizi için işletelim. Programın birinci adımında veri segmentinde yer alan dizinin birinci elemanı bellekten alınıp akümülatöre atılıyor. Birinci adım sonunda kaydedicilerde oluşan durum Şekil 5.9'da görülmektedir.

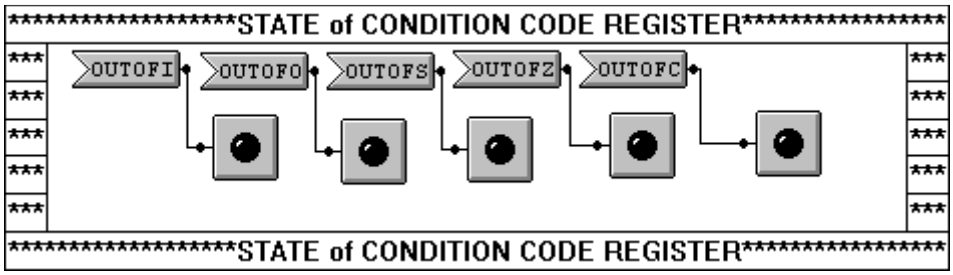


Şekil 5.9. Programın 1.adım bitiminde kaydedicilerin durumu

İkinci adımda, veri segmentinde yer alan ikinci veri yani, dizinin ikinci elemanı olan “4” verisi akümülatördeki veri ile karşılaştırılıyor. Bu adım bittikten sonra kaydedicilerde oluşan durum Şekil 5.10’da görülmektedir.

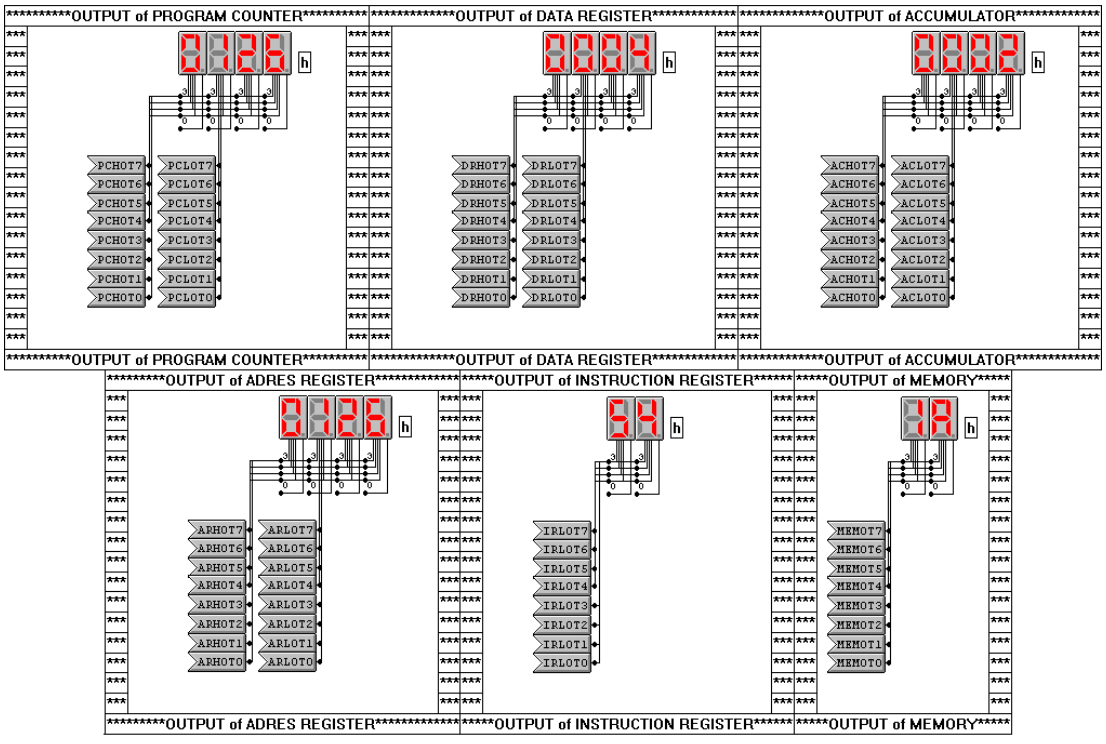


Şekil 5.10. Programın 2.adım bitiminde kaydedicilerin durumu



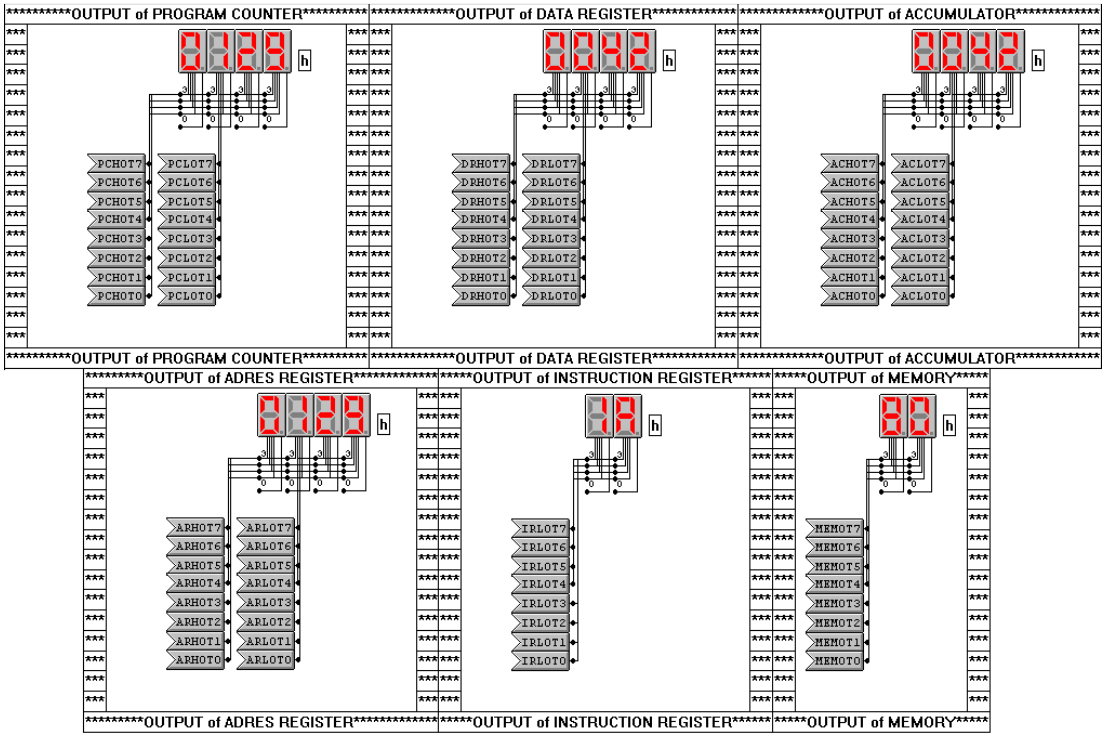
Şekil 5.11. Durum Kod Kaydedicisinin 2. Adım bitimindeki durumu.

Üçüncü adımda dizinin iki elemanı arasındaki karşılaştırma işleminden sonra Şekil 5.11'dende görüleceği üzere bayrak kontrolü yaparak sıfırdan büyük veya eşit mi olduğuna karar veriyor. Eğer bu karşılaştırma işlemi sonucunda işlem doğru ise "BUYUK" alt programına; değil ise, programın bir sonraki satırı olan "KUCUK" alt programına dallanıyor. Bu örneğimizde karşılaştırma işlemi doğru olacağından 012A h adresinden başlayan "BUYUK" alt programına dallanacaktır. Dallandığı andaki kaydedicilerin durumu Şekil 5.12'de görülmektedir.



Şekil 5.12. Dallanma gerçekleştiği andaki kaydedicilerin durumu.

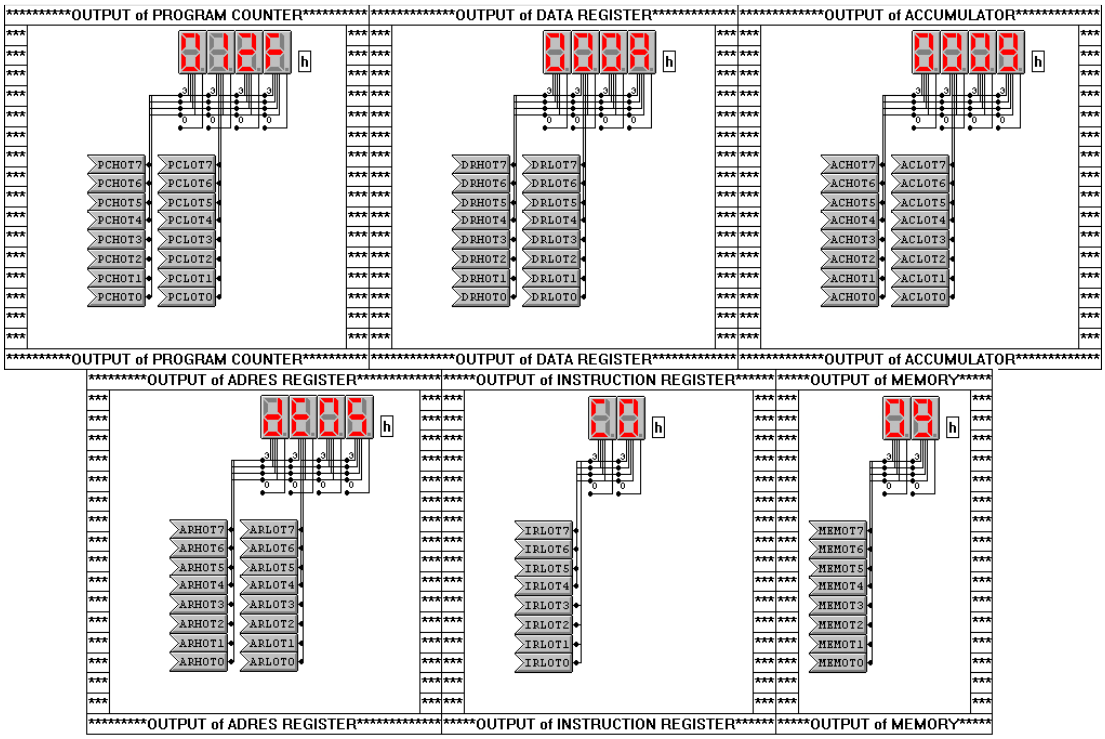
Dokuzuncu adımda yani "BUYUK" adlı programın ilk adımında "B" karakteri akümülatöre atılıyor ve hemen ardından "B" karakteri ekrana bastırılıyor. Bu iki adım bittiğinde kaydedicilerdeki durum Şekil 5.13 ve Şekil 5.14'de görülmektedir.



Şekil 5.13. Dokuzuncu adım bitiminde kaydedicilerin durumu.

Şekil 5.14. Onuncu adım bitiminde kaydedicilerin durumu.

Onbirinci adımda ise döngü sayısının bir azaltılıp tekrar aynı yere kaydedilmesi işlemleri gerçekleşiyor. Bu adımlar sonucunda kaydedicilerde Şekil 5.15'deki gibi bir durum oluşur.



Şekil 5.15. Onbirinci adım sonunda, kaydedicilerin durumu

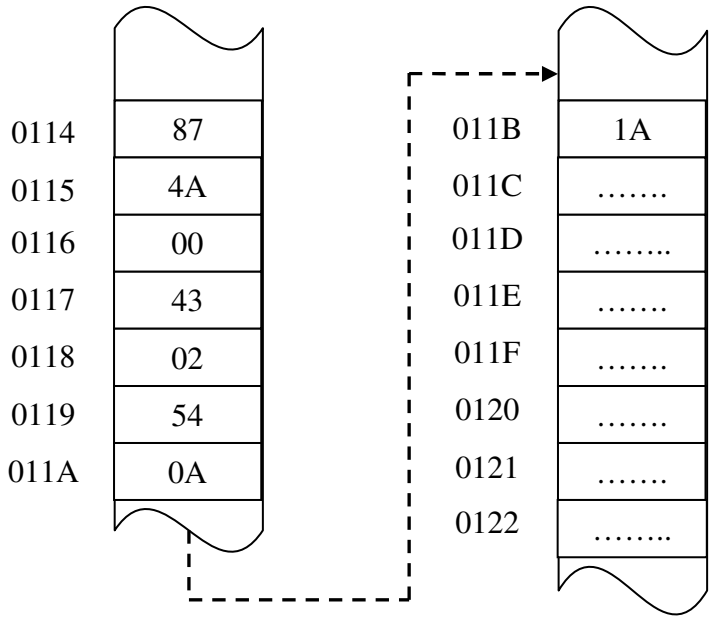
Onikinci adımda döngü sayısının sıfıra ulaşıp ulaşmadığı kontrol ediliyor. Eğer sıfıra ulaşmadıysa “BUYUK” adlı alt programın başına dallanacak ve döngü sayısı sıfır olana kadar yukarıda anlatılan işlemleri tekrar edecektir. Eğer sıfır ise, bir sonraki komut olan HLT komutunu işletmeye başlayacak ve programı sonlandıracaktır. Programın sonucunda ise Şekil 5.16’daki görüntü elde edilecektir.

Şekil 5.16. Program bitiminde elde edilen görüntü.

Örnek-3

Bu örnekte ise bilgisayarlarda kesmelerin nasıl kullanıldığına bakalım. Tasarlanan bilgisayarda sadece bir adet giriş kesmesi kullanılmış ve çıkış aygıtının her zaman müsait durumda olduğu varsayılmıştır. Giriş kesmesinde yapılan işlemler ise, klavyeden girilen karakteri akümülatöre, oradan da çıkış aygıtına iletmek şeklinde

tanımlanmıştır. Örnek-2 deki programın bellekte olduğunu varsayalım. Kesme vasıtasıyla ekrana "HOS GELDINIZ" metnini yazdıralım. Bu program çalışırken gelen bir kesme dolayısıyla program akışında meydana gelen durumları gösterelim. Şekil 5.17'ye dikkat edilecek olursa programın başına STI komutunun hexadecimal karşılığı olan 87 h yazılarak bilgisayarın kesmelere açık olduğu belirtilmiştir. Kesme geldiğinde program akışı belleğin ilk gözüne konumlandırılır. Burada yer alan şartsız dallanma komutu ile belleğin son 3 byte'ında yer alan giriş kesmesini yerine getirerek programa kaldığı yerden devam eder. Kesme geldiğinde yapılan işlemlere Bölüm 4.8.1.5'de değinilmişti. Diyelim ki, kesme dördüncü döngü yapılırken meydana geldiğini varsayalım. Tam bu anda kaydedicilerin durumu Şekil 5.18'de görülmektedir.

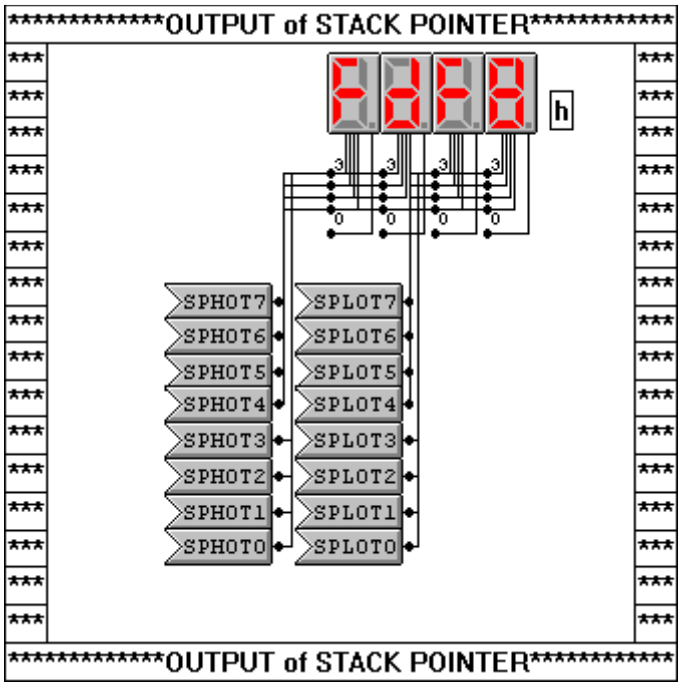


Şekil 5.17. Kesmenin dahil edildiği program

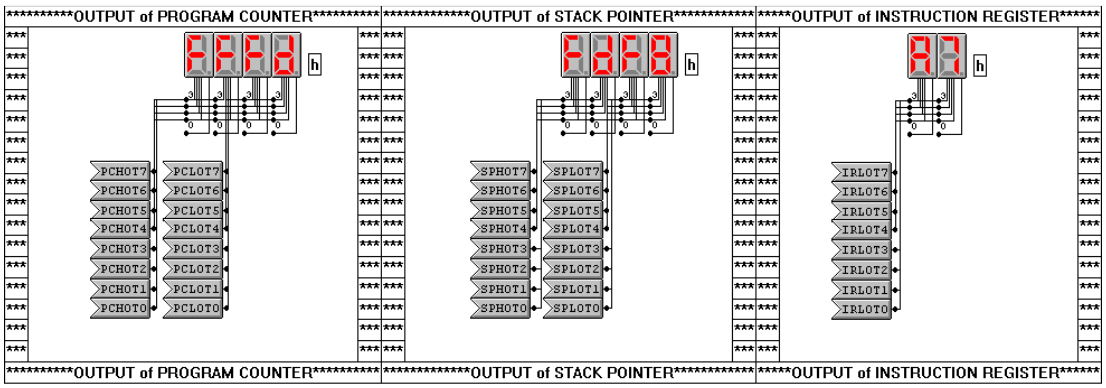
Şekil 5.18'den de görüleceği üzere kesme gelmeden önce program "BUYUK adlı alt programın döngü değişkenini bir azaltmadan önce komutu işletmiş ve kesmeye dallanmıştır. Kesme bitiminde programın bu komut ile işine devam etmesi gerekecektir.

Şekil 5.18. Kesme geldiğinde kaydedicilerin durumu.

Kesme geldiğinde yapılan ilk iş, program sayıcının içeriğini yığına atmaktır. Yığının bellekte göstermiş olduğu ilk göz FDFF h adresli bellek bölgesidir. Belleğimiz 8 bitlik bir bellek olduğundan program sayıcının ilk önce düşük anlamlı kısmı, daha sonra yüksek anlamlı kısmı yığına atılacaktır. Kesmede yapılacak ikinci işlem indis kaydedicisini yığına atmak ve ardından akümülatörün içeriğini yığına saklamaktır. Son olarak durum kod kaydedicisinin bayrak bitleri yığında saklanır. Kesme işlemi geldiğinde yapılacak olan bu işlemlerin ardından bilgisayar giriş kesmesini işlemeye hazırdır. Giriş kesmesini işletmeden önceki yığın kaydedicinin durumu Şekil 5.19'da görülmektedir. Yığın göstergecinin değeri FDFF h adresinden FFD8 h adresini gösterir olmuştur. İlk iki byte, program sayıcısına, sonraki iki byte indis kaydedicisine, daha sonraki iki byte akümülatöre ve son byte ise durum kod kaydedicinin içeriği belleğe atılması için kullanılmıştır. Bundan sonra bilgisayarın yapacağı ilk belleğin sıfır numaralı adresine giderek burada giriş kesmesinin olduğu alt program adresini alarak program sayıcısına yüklemektir. Giriş kesmesine dallanma yapacağı anda program sayıcısının durumu Şekil 5.20'de görülmektedir.



Şekil 5.19. Giriş kesmesini işletmeden önce yığın göstergecinin durumu

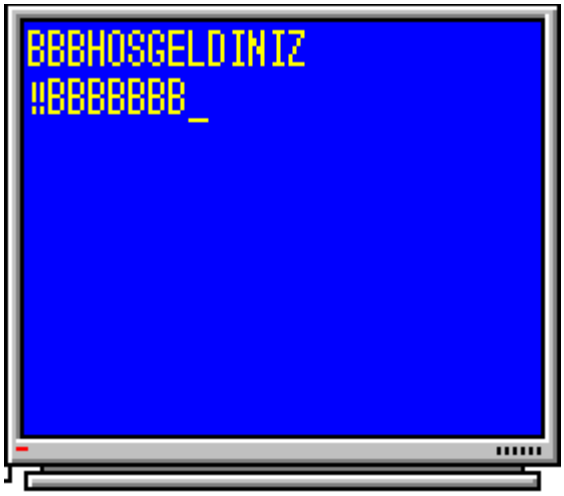


Şekil 5.20. Giriş kesmesine dallanmadan önceki bazı kaydedicilerin durumları

Giriş kesmesine ait programı işlettikten sonra kesmeye girmeden önce yapılan işlemlerin tam tersini, yığında yer alan durum kaydedicinin bayrak bitlerini tekrar eski durumlarına, akümülatörü eski durumuna, indis kaydedicisini eski durumuna ve son olarak program sayıcısının eski durumuna getirerek kesme işlemini bitirir ve program akışı kaldığı yerden devam eder. kesme bittiğinde kaydedicilerdeki durum Şekil 5.21'de görülmektedir.

Şekil 5.21. Kesme işlemi sona erdiğinde kaydedicilerin durumu

Program, yukarıda değinildiği üzere dördüncü döngüde kalmıştı. Kesmeden sonra program altı döngü daha işletilerek aşağıdaki ekran görüntüsü elde edilir.



Şekil 5.22. Program bitiminde elde edilen ekran görüntüsü

BÖLÜM 6. TARTIŞMA VE ÖNERİLER

Bu tez çalışmasında, bilgisayar mühendisliğinin temel derslerinden olan "Bilgisayar Mimarisi ve Organizasyonu" dersinin daha iyi anlaşılabilmesi için, ders materyali olarak kullanılabilecek bir bilgisayar geliştirildi. Geliştirilen bu bilgisayar ile öğrencinin temel lojik elemanlarının işleyişini görme, çalışma prensibini anlama bir mikro işlemcinin çalışma şekli ve çevre birimlerle olan iletişimin nasıl oluşturulması gerektiği ve tasarlanırken nelere dikkat edilmesi gerektiği bu tezin başlıca konularını oluşturmaktadır. Multimedia Logic simülasyon programında tasarlanan bilgisayarın işleyişini görme adına Visual Studio .NET ortamında geliştirilen ve Ek-K'da tanıtılan assembler programı yazılan programı makine diline çevirerek bilgisayardaki belleğe atar ve simülasyon hızını ayarlayarak çalışmasını adım adım görmesi kullanıcıya bir imkan olarak verilmektedir.

Bu tezin birinci aşamasında, uluslar arası bu alanda yapılmış olan çalışmalar incelendi. İncelenen bu çalışmalara bakıldığında, yapılan çalışmaların ortak amacının bilgisayar mimarisi derslerindeki uygulama eksikliğini kapatmak gayesiyle yapılmış olduğu görüldü. Ancak yapılan çalışmalarda komut listesinin yetersiz oluşu, adresleme modalarının sayısının az oluşu ve tasarım aşamasında kullanılan elemanların hazır bloklar halinde verilmiş olması tespit edilen eksikliklerdir. Görülen eksiklikler bu tez çalışmasında kapatılmıştır.

İkinci aşamada, temel mikro işlemcilerin çalışması, sahip olduğu mimari yapılar, temel lojik elemanlar anlatılmış olup tasarlanan mikro işlemcinin tasarımında kullanılan yapılar incelenme aşamasında bu bilgilerden kullanıcının faydalanması gözetildi.

Üçüncü aşamada, tasarlanan mikro işlemcinin uygulaması yapılacak olan Multimedia Logic adlı simülasyon programının tanıtımı yapılarak tasarlanan mikro işlemcide kullanılan elemanlar hakkında ön bilgi olması sağlandı.

Bu tezin dördüncü aşamasında, özellikle ticari mikro işlemcilerin ortak olarak kullanmış olduğu komut seti oluşturulmuş bunun sonucunda 59 adet komut ortaya çıkmıştır. Bu komut listesi oluşturulurken özellikle çok sık kullanılan komutlar bu tez çalışmasında gerçekleştirilmiştir.

Beşinci aşamada, hangi adresleme modlarının kullanılacağına karar verildi. Bu karar verilirken yine çok sık kullanılan adresleme modlarından 6 tanesi kullanıldı. Bu adresleme modları Bölüm 4.4'de tanıtılmıştır.

Komut listesi ve adresleme modları oluşturulduktan sonra gerekli genel ve özel amaçlı kaydediciler, kullanılacak bellek yapısı ve bu bellek ile kaydediciler arasındaki iletişimi sağlayacak ortak veri yolu tasarımı ele alındı.

Komut listesi, adresleme modları ve ortak veri yolu oluşturulduktan sonra işletilecek olan komutların mikro işlem adımları tanımlanarak bu mikro işlem adımlarının uygun bir şekilde çalışmasını sağlayacak denetim mekanizması geliştirildi. Geliştirilen denetim mekanizması donanımsal olarak gerçekleştirilmiş olup tasarlanan komut ve adresleme modu sayısının artmasına olanak verilmiştir.

Son aşamada ise, tasarlanan mikro işlemcinin doğru şekilde çalışmasını göstermek amacıyla örnekler verilmiş olup, yapılan örneklerin sonuçları adım adım gösterilerek kullanıcının istifadesine sunulmuştur. Bu şekilde kullanıcı bir mikro işlemcinin, yazılan bir programı nasıl işlettiğini ve işletirken yaptığı işlemleri gözlemlediği için bilgisayarın araka planında neler olup neler bittiğini çok iyi bir şekilde düşünmesine olanak sağlamaktadır.

İleriye yönelik çalışmalar olarak aşağıdaki konular önerilebilir:

Tasarlanan mikro işlemcinin komut kümesi ve sahip olmuş olduğu adresleme mod sayısı genişletilebilir.

Çarpma devresi Bölüm 4'de bahsedildiği üzere 8 bitlik sayıları çarpabilen bir yapıya sahiptir. Bu yapı genişletilebilir.

Bu tez çalışmasında denetim mekanizması donanımsal olarak gerçekleştirildi. Denetim mekanizması mikro program yaklaşımıyla yapılırsa kullanıcının ikisi arasındaki farkı görmesi sağlanmış olur. Bu şekilde derslerde anlatılan mikro programla denetim mekanizmasının da uygulaması yapılmış olur.

Bu mikro işlemciye pipeline yapısı dahil edilerek kullanıcının hem bu yapının nasıl oluşturulduğunu görmesi hem de komutların daha kısa sürede yerine getirileceğini ve bu şekilde mikro işlemciden daha fazla performans elde edileceğini görmüş olacaktır.

Bu mikro işlemci için yazılan derleyici programı geliştirilerek tasarlanan mikro işlemcinin gömülü sistem haline getirilerek bilgisayar mimarisi derslerinde faydalanabilecek bir mikro işlemci olacaktır.

KAYNAKLAR

[1] STALLINGS, W., Computer Organization and Architecture, Prentice Hall, New Jersey, 1996.

[2] PATTERSON, D. A., HENNESY, J. L., Computer Architecture: A Quantitative Approach, Morgan Kaufmann, San Francisco, California, 1996.

[3] DJORDJEVIC, J., MILENKOVIC, A., GRBANOVIC, N., BOJOVIC, M., An Educational Environment for Teaching a Course in Computer Architecture and Organization, Proceeding of the IEEE/ACM HPCA-98 Workshop on Computer Architecture Education, Las Vegas, NV, January 1998.

[4] DJORDJEVIC, J., MILENKOVIC, A., PRODANOVIC, S., A Hierarchical Memory System Environment, Proceeding of the IEEE/ACM ISCA-98 Workshop on Computer Architecture Education, Barcelona, Spain, June 1998.

[5] THEYS, M. D., TROY, P. A., Lessons Learned from Teaching Computer Architecture to Computer Science Students, 33rd ASEE/IEEE Frontiers in Education Conference, Chicago, Vol. 2, pp. 7-12, 2003.

[6] http://w3.gazi.edu.tr/~nurettin/visual/index.htm, Eylül 2008.

[7] STANLEY, T. D., PRIGMORE, D., MIKOLYSKI, S., EMBREY, G., FIFE, L., COLTON, D., Pegagogic Value in Understanding Computer Architecture of Implementing the Marie Computer from Null and Lobur in the Logic Emulation Software, multimedia Logic, San Diego, California, pp. 66-71, 2007.

[8] NULL, L., LOBUR, J., The essentials of Computer Organization and Architecture, Jones and Barlett Publishers, Sudbury, 1997.

[9] STANLEY, T. D., PRIGMORE, D., MIKOLYSKI, S., EMBREY, G., FIFE, L., COLTON, D., From Archi Torture to Architecture: Undergraduate Students Design and Implement Computers Using the Multimedia Logic Emulator, Computer Science Education, Vol. 17, pp. 141-152, 2007.

[10] STANLEY, T. D., XUAN, T. Q., FIFE, L., COLTON, D., Simple Eight Bit, Emulated Computers for Illustrating Computer Architecture Concepts and Providing a Starting Point for Student Design, Proceedings of the ninth Australasian conference on Computing education, Ballarat, Victoria, Australia, pp. 141-146, 2007.

[11] KAWAMURA, T., KAWAGUCHI, Y., NAKANISHI, S., SUGAHARA, K., Machine Cycle CPU Simulator for Educational Use based on Squeok Environment, Creating, Connecting and Collaborating Through Computing, 2003. C5 2003. Proceedings. First Conference on**,** Tottori Univ. Japan, pp. 120-121, 2003.

[12] KORIENEK, G., WRENSCH, T., DECHOW, D., SQUEAK- A Quick Trip to Object Land, Addision Wesley, 2002.

[13] CARPINELLI, J., The very Simple CPU Simulator, 32nd ASEE/IEEE Frontiers in Education Conference, New Jersey Inst. of Technol., Newark, NJ, USA, pp. 11-14, 2002.

[14] CARPINELLI, J., ZAMAN, T., Instructional tools for designing and analyzing a very simple CPU, International Journal of Electrical Engineering Education, Vol. 43, pp. 261-270, 2006.

[15] ARIAS, J. R., GARCIA, D. F., Introducing computer Architecture education in the first course of computer science career, Proceedings of the 1998 workshop on Computer architecture education, New York, NY, USA, 1998.

[16] CLARK, B., CZEZOWOSKI, A., STRAZDINS, P., Imlementation Aspects of a SPARC V9 Complete Machine Simulator, Australian Computer Science Communications, Vol. 24, pp. 23-32, 2002.

[17] WAINER, G., DAICZ, S., SIMONI, L., WASSERMAN, D., Using the Alfa-1 Simulated Processor for Educational Purposes, Journal on Educational Resources in Computing (JERIC), Vol. 1, pp. 111-151, 2001.

[18] MOURE, J., REXEACHS, D., LUQUE, E., KScalar Simulator, Journal on Educational Resources in Computing (JERIC), Vol.2, pp. 73-116, 2002.

[19] HOLLAND, M., HARRIS, J., HAUCK, S., Harnessing FPGAs for Computer Architecture Education, Proceeding 2003 IEEE International Conference on Microelectronic Systems Education, Washington, DC, USA, pp. 12-13, 2003.

[20] Tao, J., SCHULZ, M., KARL, W., A Simulation Tool for Evaluating Shared Memory Systems, Proceeding 36th Annual Simulation Symposium, pp. 335-342, 2003.

[21] HAMACHER, C., VRANESIC, Z., ZAKY, S., Computer Organization, 5th Edition, McGraw-Hill, pp. 6, New York, 2002.

[22] TÜRKOĞLU, İ., Mikrobilgisayar Mimarisi ve Programlama, Ders Notları, Fırat Üniversitesi, sf. 10-26, 2006.

[23] MANO, M. M., Digital Design, pp. 87-95, NJ: Prentice Hall, 1991

[24] GEORGE, M., Multimedia Logic Help, www.softronix.com, Haziran 2008.

[25] MANO, M. M., Bilgisayar Sistemleri Mimarisi, MARŞOĞLU, A., 3. Basım, SUÇSUZ, N., Literatür Yayıncılık, pp. 121, İstanbul, 2002.

[26] HAMACHER, C., VRANESIC, Z., ZAKY, S., Computer Organization, 5th Edition, McGraw-Hill, pp. 68-70, New York, 2002.

[27] HAMACHER, C., VRANESIC, Z., ZAKY, S., Computer Organization, 5th Edition, McGraw-Hill, pp. 46, New York, 2002.

[28] MANO, M. M., Bilgisayar Sistemleri Mimarisi, MARŞOĞLU, A., 3. Basım, SUÇSUZ, N., Literatür Yayıncılık, pp. 93-115, İstanbul, 2002.

[29] MANO, M. M., Bilgisayar Sistemleri Mimarisi, MARŞOĞLU, A., 3. Basım, SUÇSUZ, N., Literatür Yayıncılık, pp. 9, İstanbul, 2002.

[30] MANO, M. M., Bilgisayar Sistemleri Mimarisi, MARŞOĞLU, A., 3. Basım, SUÇSUZ, N., Literatür Yayıncılık, pp. 244-247, İstanbul, 2002.

[31] MANO, M. M., Bilgisayar Sistemleri Mimarisi, MARŞOĞLU, A., 3. Basım, SUÇSUZ, N., Literatür Yayıncılık, pp. 319-333, İstanbul, 2002.

[32] MANO, M. M., Bilgisayar Sistemleri Mimarisi, MARŞOĞLU, A., 3. Basım, SUÇSUZ, N., Literatür Yayıncılık, pp. 129-159, İstanbul, 2002.

EKLER

EK-A. Bellek ve Akümülatör Komutlarının Mikro İşlem Adımları

1. ADD: Bellekteki veri ile Akümülatördeki veriyi toplayarak akümülatöre kaydeden bir komuttur. Dört farklı adresleme moduna sahiptir.

a) Derhal Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC00*ADRMD1	$DR_H \leftarrow M[AR]$, $AR \leftarrow AR+1$
T4* IDEC00*ADRMD1	$DR_L \leftarrow M[AR]$, $PC \leftarrow PC+1$
T5* IDEC00*ADRMD1	$AC \leftarrow AC+DR$, $C \leftarrow C_{out}$, $SC \leftarrow 0$

b) Direkt Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC00*ADRMD2	$TR_H \leftarrow M[AR]$,$AR \leftarrow AR+1$
T4* IDEC00*ADRMD2	$TR_L \leftarrow M[AR]$,$PC \leftarrow PC+1$
T5* IDEC00*ADRMD2	$AR \leftarrow TR$
T6* IDEC00*ADRMD2	$DR_H \leftarrow M[AR]$,$AR \leftarrow AR+1$
T7* IDEC00*ADRMD2	$DR_L \leftarrow M[AR]$
T8* IDEC00*ADRMD2	$AC \leftarrow AC+DR$, $C \leftarrow C_{out}$,$SC \leftarrow 0$

c) Dolaylı Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC00*ADRMD3	$TR_H \leftarrow M[AR]$,$AR \leftarrow AR+1$
T4* IDEC00*ADRMD3	$TR_L \leftarrow M[AR]$,$PC \leftarrow PC+1$
T5* IDEC00*ADRMD3	$AR \leftarrow TR$
T6* IDEC00*ADRMD3	$TR_H \leftarrow M[AR]$,$AR \leftarrow AR+1$

KOMUTUN MİKRO İŞLEM ADIMLARI	
T7* IDEC00*ADRMD3	$TR_L \leftarrow M[AR]$
T8* IDEC00*ADRMD3	AR←TR
T9* IDEC00*ADRMD3	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T10* IDEC00*ADRMD3	$DR_L \leftarrow M[AR]$
T11* IDEC00*ADRMD3	$AC \leftarrow AC+DR, C \leftarrow C_{out}, SC \leftarrow 0$

d) İndis Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC00*ADRMD4	AR←Etkin Adres
T4* IDEC00*ADRMD4	$DR_L \leftarrow M[AR], AR \leftarrow AR+1$
T5* IDEC00*ADRMD4	$DR_H \leftarrow M[AR]$
T6* IDEC00*ADRMD4	$AC \leftarrow AC+DR, C \leftarrow C_{out}, SC \leftarrow 0$

2. ADDC: Bellekteki veri, Akümülatördeki veri ve eldeyi toplayarak akümülatöre kaydeden bir komuttur. Dört farklı adresleme moduna sahiptir.

a) Derhal Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC01*ADRMD1	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC01*ADRMD1	$DR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC01*ADRMD1	$AC \leftarrow AC+DR+C, C \leftarrow C_{out}, SC \leftarrow 0$

b) Direkt Mode

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC01*ADRMD2	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC01*ADRMD2	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC01*ADRMD2	AR←TR
T6* IDEC01*ADRMD2	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC01*ADRMD2	$DR_L \leftarrow M[AR]$
T8* IDEC01*ADRMD2	$AC \leftarrow AC+DR+C, C \leftarrow C_{out}, SC \leftarrow 0$

c) Dolaylı Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC01*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC01*ADRMD3	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC01*ADRMD3	$AR \leftarrow TR$
T6* IDEC01*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC01*ADRMD3	$TR_L \leftarrow M[AR]$
T8* IDEC01*ADRMD3	$AR \leftarrow TR$
T9* IDEC01*ADRMD3	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T10* IDEC01*ADRMD3	$DR_L \leftarrow M[AR]$
T11* IDEC01*ADRMD3	$AC \leftarrow AC+DR+C, C \leftarrow C_{out}, SC \leftarrow 0$

d) İndis Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC01*ADRMD4	$AR \leftarrow$ Etkin Adres
T4* IDEC01*ADRMD4	$DR_L \leftarrow M[AR], AR \leftarrow AR+1$
T5* IDEC01*ADRMD4	$DR_H \leftarrow M[AR]$
T6* IDEC01*ADRMD4	$AC \leftarrow AC+DR+C, C \leftarrow C_{out}, SC \leftarrow 0$

3. AND: Bellekteki veri ile Akümülatördeki veriyi lojik AND işlemine tabi tutarak sonucu akümülatöre kaydeden bir komuttur. Dört farklı adresleme moduna sahiptir

a) Derhal Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC02*ADRMD1	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC02*ADRMD1	$DR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC02*ADRMD1	$AC \leftarrow AC \wedge DR, C \leftarrow C_{out}, SC \leftarrow 0$

b) Direkt Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC02*ADRMD2	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC02*ADRMD2	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC02*ADRMD2	$AR \leftarrow TR$
T6* IDEC02*ADRMD2	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC02*ADRMD2	$DR_L \leftarrow M[AR]$
T8* IDEC02*ADRMD2	$AC \leftarrow AC \wedge DR, C \leftarrow C_{out}, SC \leftarrow 0$

c) Dolaylı Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC02*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC02*ADRMD3	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC02*ADRMD3	$AR \leftarrow TR$
T6* IDEC02*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC02*ADRMD3	$TR_L \leftarrow M[AR]$
T8* IDEC02*ADRMD3	$AR \leftarrow TR$
T9* IDEC02*ADRMD3	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T10* IDEC02*ADRMD3	$DR_L \leftarrow M[AR]$
T11* IDEC02*ADRMD3	$AC \leftarrow AC \wedge DR, C \leftarrow C_{out}, SC \leftarrow 0$

d) İndis Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC02*ADRMD4	$AR \leftarrow$ Etkin Adres
T4* IDEC02*ADRMD4	$DR_L \leftarrow M[AR], AR \leftarrow AR+1$
T5* IDEC02*ADRMD4	$DR_H \leftarrow M[AR]$
T6* IDEC02*ADRMD4	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$

4. CMP: Bellekteki veri ile Akümülatördeki veriyi karşılaştırarak sonucu akümülatöre kaydeden bir komuttur. Dört farklı adresleme moduna sahiptir.

a) Derhal Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC03*ADRMD1	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC03*ADRMD1	$DR_L \leftarrow M[AR]$, $PC \leftarrow PC+1$
T5* IDEC03*ADRMD1	AC-DR,$SC \leftarrow 0$

b) Direkt Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC03*ADRMD2	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC03*ADRMD2	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC03*ADRMD2	$AR \leftarrow TR$
T6* IDEC03*ADRMD2	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC03*ADRMD2	$DR_L \leftarrow M[AR]$
T8* IDEC03*ADRMD2	AC-DR ,$SC \leftarrow 0$

c) Dolaylı Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC03*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC03*ADRMD3	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC03*ADRMD3	$AR \leftarrow TR$
T6* IDEC03*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC03*ADRMD3	$TR_L \leftarrow M[AR]$
T8* IDEC03*ADRMD3	$AR \leftarrow TR$
T9* IDEC03*ADRMD3	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T10* IDEC03*ADRMD3	$DR_L \leftarrow M[AR]$
T11* IDEC03*ADRMD3	AC-DR,$SC \leftarrow 0$

d) İndis Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC03*ADRMD4	AR←Etkin Adres
T4* IDEC03*ADRMD4	DR_L←M[AR], AR←AR+1
T5* IDEC03*ADRMD4	DR_H←M[AR]
T6* IDEC03*ADRMD4	AC←AC-DR, SC←0

5. DIV: Bellekteki veriyi ile Akümülatördeki veriye bölerek sonucu akümülatöre kaydeden bir komuttur. Dört farklı adresleme moduna sahiptir.

a) Derhal Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC05*ADRMD1	DR_H←M[AR],AR←AR+1
T4* IDEC05*ADRMD1	DR_L←M[AR]. PC←PC+1
T5* IDEC05*ADRMD1	AC←AC÷DR, SC←0

b) Direkt Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC05*ADRMD2	TR_H←M[AR],AR←AR+1
T4* IDEC05*ADRMD2	TR_L←M[AR],PC←PC+1
T5* IDEC05*ADRMD2	AR←TR
T6* IDEC05*ADRMD2	DR_H←M[AR],AR←AR+1
T7* IDEC05*ADRMD2	DR_L←M[AR]
T8* IDEC05*ADRMD2	AC←AC÷DR, SC←0

c) Dolaylı Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC05*ADRMD3	TR_H←M[AR],AR←AR+1
T4* IDEC05*ADRMD3	TR_L←M[AR],PC←PC+1
T5* IDEC05*ADRMD3	AR←TR
T6* IDEC05*ADRMD3	TR_H←M[AR],AR←AR+1

KOMUTUN MİKRO İŞLEM ADIMLARI	
T7* IDEC05*ADRMD3	$TR_L \leftarrow M[AR]$
T8* IDEC05*ADRMD3	$AR \leftarrow TR$
T9* IDEC05*ADRMD3	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T10* IDEC05*ADRMD3	$DR_L \leftarrow M[AR]$
T11* IDEC05*ADRMD3	$AC \leftarrow AC \div DR, SC \leftarrow 0$

d) İndis Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC05*ADRMD4	$AR \leftarrow$ Etkin Adres
T4* IDEC05*ADRMD4	$DR_L \leftarrow M[AR], AR \leftarrow AR+1$
T5* IDEC05*ADRMD4	$DR_H \leftarrow M[AR]$
T6* IDEC05*ADRMD4	$AC \leftarrow AC \div DR, SC \leftarrow 0$

6. XOR: Bellekteki veriyi ile Akümülatördeki veriyi lojik XOR işlemine tabi tutarak sonucu akümülatöre kaydeden bir komuttur. Dört farklı adresleme moduna sahiptir.

a) Ivedi Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC06*ADRMD1	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC06*ADRMD1	$DR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC06*ADRMD1	$AC \leftarrow AC \oplus DR, C \leftarrow C_{out}, SC \leftarrow 0$

b) Direkt Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC06*ADRMD2	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC06*ADRMD2	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC06*ADRMD2	$AR \leftarrow TR$
T6* IDEC06*ADRMD2	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC06*ADRMD2	$DR_L \leftarrow M[AR]$

KOMUTUN MİKRO İŞLEM ADIMLARI	
T8* IDEC06*ADRMD2	AC←AC⊕ DR, C←C_{out} ,SC←0

c) Dolaylı Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC06*ADRMD3	TR_H←M[AR],AR←AR+1
T4* IDEC06*ADRMD3	TR_L←M[AR],PC←PC+1
T5* IDEC06*ADRMD3	AR←TR
T6* IDEC06*ADRMD3	TR_H←M[AR],AR←AR+1
T7* IDEC06*ADRMD3	TR_L←M[AR]
T8* IDEC06*ADRMD3	AR←TR
T9* IDEC06*ADRMD3	DR_H←M[AR],AR←AR+1
T10* IDEC06*ADRMD3	DR_L←M[AR]
T11* IDEC06*ADRMD3	AC←AC⊕ DR, C←C_{out} ,SC←0

d) İndis Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC05*ADRMD4	AR←Etkin Adres
T4* IDEC05*ADRMD4	DR_L←M[AR],AR←AR+1
T5* IDEC05*ADRMD4	DR_H←M[AR]
T6* IDEC05*ADRMD4	AC←AC⊕DR, SC←0

7. OR: Bellekteki veriyi ile Akümülatördeki veriyi lojik OR işlemine tabi tutarak sonucu akümülatöre kaydeden bir komuttur. Dört farklı adresleme moduna sahiptir.

a) Derhal Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC11*ADRMD1	DR_H←M[AR],AR←AR+1
T4* IDEC11*ADRMD1	DR_L←M[AR], PC←PC+1
T5* IDEC11*ADRMD1	AC←AC∨DR, SC←0

b) Direkt Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC11*ADRMD2	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC11*ADRMD2	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC11*ADRMD2	$AR \leftarrow TR$
T6* IDEC11*ADRMD2	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC11*ADRMD2	$DR_L \leftarrow M[AR]$
T8* IDEC11*ADRMD2	$AC \leftarrow AC \vee DR, C \leftarrow C_{out}, SC \leftarrow 0$

c) Dolaylı Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC11*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC11*ADRMD3	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC11*ADRMD3	$AR \leftarrow TR$
T6* IDEC11*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC11*ADRMD3	$TR_L \leftarrow M[AR]$
T8* IDEC11*ADRMD3	$AR \leftarrow TR$
T9* IDEC11*ADRMD3	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T10* IDEC11*ADRMD3	$DR_L \leftarrow M[AR]$
T11* IDEC11*ADRMD3	$AC \leftarrow AC \vee DR, C \leftarrow C_{out}, SC \leftarrow 0$

d) İndis Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC11*ADRMD4	$AR \leftarrow$ Etkin Adres
T4* IDEC11*ADRMD4	$DR_L \leftarrow M[AR], AR \leftarrow AR+1$
T5* IDEC11*ADRMD4	$DR_H \leftarrow M[AR]$
T6* IDEC11*ADRMD4	$AC \leftarrow AC \vee DR, SC \leftarrow 0$

8. SUB: Bellekteki veriyi ile Akümülatördeki veriden çıkartma işlemine tabi tutarak sonucu akümülatöre kaydeden bir komuttur. Dört farklı adresleme moduna sahiptir.

a) Derhal Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC14*ADRMD1	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC14*ADRMD1	$DR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC14*ADRMD1	$AC \leftarrow AC-DR, C \leftarrow C_{out}, SC \leftarrow 0$

b) Direkt Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC14*ADRMD2	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC14*ADRMD2	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC14*ADRMD2	$AR \leftarrow TR$
T6* IDEC14*ADRMD2	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC14*ADRMD2	$DR_L \leftarrow M[AR]$
T8* IDEC14*ADRMD2	$AC \leftarrow AC-DR, C \leftarrow C_{out}, SC \leftarrow 0$

c) Dolaylı Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC14*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC14*ADRMD3	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC14*ADRMD3	$AR \leftarrow TR$
T6* IDEC14*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC14*ADRMD3	$TR_L \leftarrow M[AR]$
T8* IDEC14*ADRMD3	$AR \leftarrow TR$
T9* IDEC14*ADRMD3	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T10* IDEC14*ADRMD3	$DR_L \leftarrow M[AR]$
T11* IDEC14*ADRMD3	$AC \leftarrow AC-DR, C \leftarrow C_{out}, SC \leftarrow 0$

d) İndis Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC14*ADRMD4	$AR \leftarrow$ Etkin Adres
T4* IDEC14*ADRMD4	$DR_L \leftarrow M[AR], AR \leftarrow AR+1$
T5* IDEC14*ADRMD4	$DR_H \leftarrow M[AR]$

KOMUTUN MİKRO İŞLEM ADIMLARI	
T6* IDEC14*ADRMD4	AC←AC-DR, C←$C_{out,}$ SC←0

9. SUBC: Bellekteki veriyi ile Akümülatördeki veriden borç biti ile çıkartma işlemine tabi tutarak sonucu akümülatöre kaydeden bir komuttur. Dört farklı adresleme moduna sahiptir.

a) Derhal Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC15*ADRMD1	DR_H←M[AR],AR←AR+1
T4* IDEC15*ADRMD1	DR_L←M[AR], PC←PC+1
T5* IDEC15*ADRMD1	AC←AC-DR-C, C←C_{out} ,SC←0

b) Direkt Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC15*ADRMD2	TR_H←M[AR],AR←AR+1
T4* IDEC15*ADRMD2	TR_L←M[AR],PC←PC+1
T5* IDEC15*ADRMD2	AR←TR
T6* IDEC15*ADRMD2	DR_H←M[AR],AR←AR+1
T7* IDEC15*ADRMD2	DR_L←M[AR]
T8* IDEC15*ADRMD2	AC←AC-DR-C, C←C_{out} ,SC←0

c) Dolaylı Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC15*ADRMD3	TR_H←M[AR],AR←AR+1
T4* IDEC15*ADRMD3	TR_L←M[AR],PC←PC+1
T5* IDEC15*ADRMD3	AR←TR
T6* IDEC15*ADRMD3	TR_H←M[AR],AR←AR+1
T7* IDEC15*ADRMD3	TR_L←M[AR]
T8* IDEC15*ADRMD3	AR←TR
T9* IDEC15*ADRMD3	DR_H←M[AR],AR←AR+1

KOMUTUN MİKRO İŞLEM ADIMLARI	
T10* IDEC15*ADRMD3	$DR_L \leftarrow M[AR]$
T11* IDEC15*ADRMD3	$AC \leftarrow AC-DR-C, C \leftarrow C_{out}, SC \leftarrow 0$

d) İndis Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC15*ADRMD4	AR←Etkin Adres
T4* IDEC15*ADRMD4	$DR_L \leftarrow M[AR], AR \leftarrow AR+1$
T5* IDEC15*ADRMD4	$DR_H \leftarrow M[AR]$
T6* IDEC15*ADRMD4	$AC \leftarrow AC-DR-C, C \leftarrow C_{out}, SC \leftarrow 0$

10. MUL: Bellekteki veriyi ile Akümülatördeki veriyi çarpma işlemine tabi tutarak sonucu akümülatöre kaydeden bir komuttur. Dört farklı adresleme moduna sahiptir.

a) Derhal Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC21*ADRMD1	$DR_L \leftarrow M[AR]$
T4* IDEC21*ADRMD1	$AC \leftarrow AC*DR, C \leftarrow C_{out}, SC \leftarrow 0$

b) Direkt Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC21*ADRMD2	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC21*ADRMD2	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC21*ADRMD2	$AR \leftarrow TR$
T6* IDEC21*ADRMD2	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC21*ADRMD2	$DR_L \leftarrow M[AR]$
T8* IDEC21*ADRMD2	$AC \leftarrow AC*DR, C \leftarrow C_{out}, SC \leftarrow 0$

c) Dolaylı Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC21*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC21*ADRMD3	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC21*ADRMD3	$AR \leftarrow TR$
T6* IDEC21*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC21*ADRMD3	$TR_L \leftarrow M[AR]$
T8* IDEC21*ADRMD3	$AR \leftarrow TR$
T9* IDEC21*ADRMD3	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T10* IDEC21*ADRMD3	$DR_L \leftarrow M[AR]$
T11* IDEC21*ADRMD3	$AC \leftarrow AC*DR, C \leftarrow C_{out}, SC \leftarrow 0$

d) İndis Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC21*ADRMD4	$AR \leftarrow$ Etkin Adres
T4* IDEC21*ADRMD4	$DR_L \leftarrow M[AR], AR \leftarrow AR+1$
T5* IDEC21*ADRMD4	$DR_H \leftarrow M[AR]$
T6* IDEC21*ADRMD4	$AC \leftarrow AC*DR, C \leftarrow C_{out}, SC \leftarrow 0$

11. LDA: Bellekteki veriyi akümülatöre yükler. Dört farklı adresleme moduna sahiptir.

a) Derhal Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC10*ADRMD1	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC10*ADRMD1	$DR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC10*ADRMD1	$AC \leftarrow DR, SC \leftarrow 0$

b) Direkt Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC10*ADRMD2	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$

KOMUTUN MİKRO İŞLEM ADIMLARI	
T4* IDEC10*ADRMD2	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC10*ADRMD2	$AR \leftarrow TR$
T6* IDEC10*ADRMD2	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC10*ADRMD2	$DR_L \leftarrow M[AR]$
T8* IDEC10*ADRMD2	$AC \leftarrow DR, SC \leftarrow 0$

c) Dolaylı Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC10*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC10*ADRMD3	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC10*ADRMD3	$AR \leftarrow TR$
T6* IDEC10*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC10*ADRMD3	$TR_L \leftarrow M[AR]$
T8* IDEC10*ADRMD3	$AR \leftarrow TR$
T9* IDEC10*ADRMD3	$DR_H \leftarrow M[AR], AR \leftarrow AR+1$
T10* IDEC10*ADRMD3	$DR_L \leftarrow M[AR]$
T11* IDEC10*ADRMD3	$AC \leftarrow DR, SC \leftarrow 0$

d) İndis Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC10*ADRMD4	$AR \leftarrow$ Etkin Adres
T4* IDEC10*ADRMD4	$DR_L \leftarrow M[AR], AR \leftarrow AR+1$
T5* IDEC10*ADRMD4	$DR_H \leftarrow M[AR]$
T6* IDEC10*ADRMD4	$AC \leftarrow DR, SC \leftarrow 0$

12. STA: Akümülatördeki veriyi belleğe kaydeder. Üç farklı adresleme moduna sahiptir.

a) Direkt Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC16*ADRMD2	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC16*ADRMD2	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC16*ADRMD2	$AR \leftarrow TR$
T6* IDEC16*ADRMD2	$M[AR] \leftarrow AC_H, AR \leftarrow AR+1$
T7* IDEC16*ADRMD2	$M[AR] \leftarrow AC_L, SC \leftarrow 0$

b) Dolaylı Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC16*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC16*ADRMD3	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC16*ADRMD3	$AR \leftarrow TR$
T6* IDEC16*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC16*ADRMD3	$TR_L \leftarrow M[AR]$
T8* IDEC16*ADRMD3	$AR \leftarrow TR$
T9* IDEC16*ADRMD3	$M[AR] \leftarrow AC_H, AR \leftarrow AR+1$
T10* IDEC16*ADRMD3	$M[AR] \leftarrow AC_L, SC \leftarrow 0$

c) İndis Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC16*ADRMD4	$AR \leftarrow$ Etkin Adres
T4* IDEC16*ADRMD4	$M[AR] \leftarrow AC_H, AR \leftarrow AR+1$
T5* IDEC16*ADRMD4	$M[AR] \leftarrow AC_L, SC \leftarrow 0$

13. CLR: Akümülatördeki veriyi temizler ve yine akümülatöre kaydeder. Sadece doğal adresleme moduna sahiptir.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC01*ADRMD0	$AC \leftarrow 00h, SC \leftarrow 0$

14. DECR: Akümülatördeki verinin bir eksiğini alarak akümülatöre kaydeder. Doğal adresleme moduna sahiptir.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC02*ADRMD0	AC←AC-1,SC←0

15. INCR: Akümülatördeki verinin bir fazlasını alarak akümülatöre kaydeder. Doğal adresleme moduna sahiptir.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC03*ADRMD0	AC←AC+1,SC←0

16. COMR: Akümülatördeki verinin 1'e tümleyenini alır ve akümülatöre kaydeder. Doğal adresleme moduna sahiptir.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC04*ADRMD0	AC←$\overline{AC}$,SC←0

17. NEGR: Akümülatördeki verinin 2'e tümleyenini alır ve akümülatöre kaydeder. Doğal adresleme moduna sahiptir.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC05*ADRMD0	AC←00-AC,SC←0

18. PSH: Akümülatördeki veriyi yığın göstergecinin bellekte göstermiş olduğu yere kaydeder. Doğal adresleme moduna sahiptir.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC06*ADRMD0	AR←SP
T4* IDEC06*ADRMD0	M[AR]←AC_L ,SP←SP-1
T5* IDEC06*ADRMD0	AR←SP
T6* IDEC06*ADRMD0	M[AR]←AC_H ,SP←SP-1,SC←0

19. PUL: Yığın göstergecinin bellekte göstermiş olduğu veriyi akümülatöre kaydeder. Doğal adresleme moduna sahiptir.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC07*ADRMD0	SP←SP+1
T4* IDEC07*ADRMD0	AR←SP
T5* IDEC07*ADRMD0	DR_H← M[AR] ,SP←SP+1
T6* IDEC07*ADRMD0	AR←SP
T7* IDEC07*ADRMD0	DR_L← M[AR]
T8* IDEC07*ADRMD0	AC←DR, SC←0

20. SAR: Akümülatördeki veriyi aritmetik olarak sağa kaydırır ve akümülatöre kaydeder. Doğal adresleme moduna sahiptir.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC08*ADRMD0	AC←shr AC, SC←0

21. SAL: Akümülatördeki veriyi aritmetik olarak sola kaydırır ve akümülatöre kaydeder. Doğal adresleme moduna sahiptir.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC09*ADRMD0	AC←shl AC, SC←0

EK-B. Yığın ve İndeks Kaydedicisi Üzerine İşlem Yapan Komutlarının Mikro İşlem Adımları

1. LDAX: Bellekteki veriyi indeks kaydedicisine yükler. Dört farklı adresleme moduna sahiptir.

a) Derhal Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC17*ADRMD1	$IX_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC17*ADRMD1	$IX_L \leftarrow M[AR], PC \leftarrow PC+1, SC \leftarrow 0$

b) Direkt Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC17*ADRMD2	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC17*ADRMD2	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC17*ADRMD2	$AR \leftarrow TR$
T6* IDEC17*ADRMD2	$IX_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC17*ADRMD2	$IX_L \leftarrow M[AR], SC \leftarrow 0$

c) Dolaylı Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC17*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC17*ADRMD3	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC17*ADRMD3	$AR \leftarrow TR$
T6* IDEC17*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC17*ADRMD3	$TR_L \leftarrow M[AR]$
T8* IDEC17*ADRMD3	$AR \leftarrow TR$
T9* IDEC17*ADRMD3	$IX_H \leftarrow M[AR], AR \leftarrow AR+1$
T10* IDEC17*ADRMD3	$IX_L \leftarrow M[AR], SC \leftarrow 0$

d) İndis Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC17*ADRMD4	AR←Etkin Adres
T4* IDEC17*ADRMD4	IX_H←M[AR], AR←AR+1
T5* IDEC17*ADRMD4	IX_L←M[AR], SC←0

2. LDAS: Bellekteki veriyi yığın göstergecine yükler. Dört farklı adresleme moduna sahiptir.

a) Derhal Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC18*ADRMD1	SP_H←M[AR],AR←AR+1
T4* IDEC18*ADRMD1	SP_L←M[AR], PC←PC+1,SC←0

b) Direkt Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC18*ADRMD2	TR_H←M[AR],AR←AR+1
T4* IDEC18*ADRMD2	TR_L←M[AR],PC←PC+1
T5* IDEC18*ADRMD2	AR←TR
T6* IDEC18*ADRMD2	SP_H←M[AR],AR←AR+1
T7* IDEC18*ADRMD2	SP_L←M[AR],SC←0

c) Dolaylı Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC18*ADRMD3	TR_H←M[AR],AR←AR+1
T4* IDEC18*ADRMD3	TR_L←M[AR],PC←PC+1
T5* IDEC18*ADRMD3	AR←TR
T6* IDEC18*ADRMD3	TR_H←M[AR],AR←AR+1
T7* IDEC18*ADRMD3	TR_L←M[AR]
T8* IDEC18*ADRMD3	AR←TR
T9* IDEC18*ADRMD3	SP_H←M[AR],AR←AR+1

KOMUTUN MİKRO İŞLEM ADIMLARI	
T10* IDEC18*ADRMD3	$SP_L \leftarrow M[AR], SC \leftarrow 0$

d) İndis Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC18*ADRMD4	AR←Etkin Adres
T4* IDEC18*ADRMD4	$SP_H \leftarrow M[AR]$, AR←AR+1
T5* IDEC18*ADRMD4	$SP_L \leftarrow M[AR]$, SC←0

3. STAX: İndeks kaydedicisindeki değeri belleğe kaydeder. İki farklı adresleme moduna sahiptir.

a) Direkt Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC19*ADRMD2	$TR_H \leftarrow M[AR]$,AR←AR+1
T4* IDEC19*ADRMD2	$TR_L \leftarrow M[AR]$,PC←PC+1
T5* IDEC19*ADRMD2	AR←TR
T6* IDEC19*ADRMD2	$M[AR] \leftarrow IX_H$,AR←AR+1
T7* IDEC19*ADRMD2	$M[AR] \leftarrow IX_L$,SC←0

b) Dolaylı Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC19*ADRMD3	$TR_H \leftarrow M[AR]$,AR←AR+1
T4* IDEC19*ADRMD3	$TR_L \leftarrow M[AR]$,PC←PC+1
T5* IDEC19*ADRMD3	AR←TR
T6* IDEC19*ADRMD3	$TR_H \leftarrow M[AR]$,AR←AR+1
T7* IDEC19*ADRMD3	$TR_L \leftarrow M[AR]$
T8* IDEC19*ADRMD3	AR←TR
T9* IDEC19*ADRMD3	$M[AR] \leftarrow IX_H$,AR←AR+1
T10* IDEC19*ADRMD3	$M[AR] \leftarrow IX_L \leftarrow$,SC←0

4. STAS: Yığın göstergecindeki değeri belleğe kaydeder. İki farklı adresleme moduna sahiptir.

a) Direkt Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC20*ADRMD2	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC20*ADRMD2	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC20*ADRMD2	$AR \leftarrow TR$
T6* IDEC20*ADRMD2	$M[AR] \leftarrow SP_H, AR \leftarrow AR+1$
T7* IDEC20*ADRMD2	$M[AR] \leftarrow SP_L, SC \leftarrow 0$

b) Dolaylı Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC20*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T4* IDEC20*ADRMD3	$TR_L \leftarrow M[AR], PC \leftarrow PC+1$
T5* IDEC20*ADRMD3	$AR \leftarrow TR$
T6* IDEC20*ADRMD3	$TR_H \leftarrow M[AR], AR \leftarrow AR+1$
T7* IDEC20*ADRMD3	$TR_L \leftarrow M[AR]$
T8* IDEC20*ADRMD3	$AR \leftarrow TR$
T9* IDEC20*ADRMD3	$M[AR] \leftarrow SP_H, AR \leftarrow AR+1$
T10* IDEC20*ADRMD3	$M[AR] \leftarrow SP_L, SC \leftarrow 0$

5. DECX: İndeks kaydedicisindeki veriyi bir azaltarak yine indeks kaydedicisine kaydeder. Sadece doğal adresleme moduna sahiptir.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC10*ADRMD0	$IX \leftarrow IX-1, SC \leftarrow 0$

6. INCX: İndeks kaydedicisindeki veriyi bir artırarak yine indeks kaydedicisine kaydeder. Sadece doğal adresleme moduna sahiptir.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC11*ADRMD0	IX←IX+1, SC←0

7. DECS: Yığın göstergecindeki veriyi bir azaltarak yine yığın göstergecine kaydeder. Sadece doğal adresleme moduna sahiptir.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC12*ADRMD0	SP←SP-1, SC←0

8. INCS: Yığın göstergecindeki veriyi bir artırarak yine yığın göstergecine kaydeder. Sadece doğal adresleme moduna sahiptir.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC13*ADRMD0	SP←SP+1, SC←0

EK-C. Sıçrama ve Dallanma Komutlarının Mikro İşlem Adımları

1. BRA: İşlem kodundan sonra gelen byte'da yer alan veriden etkin adres hesaplanıp, bu adrese şartsız olarak dallanılır.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC00*ADRMD5	PC←Etkin adres, SC←0

2. BCC: Eğer elde biti sıfır ise hesaplanan etkin adrese dallanılır. Elde biti sıfır değilse bir sonraki komuttan program akışına devam edilir.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC01*ADRMD5	Eğer C=0 ise PC←Etkin adres, SC←0

3. BCS: Eğer elde biti 1 ise hesaplanan etkin adrese dallanılır. Elde biti 1 değilse bir sonraki komuttan program akışına devam edilir.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC02*ADRMD5	Eğer C=1 ise PC←Etkin adres, SC←0

4. BZR: Eğer sıfır biti 1 ise hesaplanan etkin adrese dallanılır. Sıfır biti 1 değilse bir sonraki komuttan program akışına devam edilir.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC03*ADRMD5	Eğer Z=1 ise PC←Etkin adres, SC←0

5. BGE: Eğer sıfırdan büyük veya eşitse etkin adrese dallanılır.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC04*ADRMD5	Eğer N⊕V=0 ise PC←Etkin adres, SC←0

6. BGR: Eğer sıfırdan büyükse etkin adrese dallanılır.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC05*ADRMD5	Eğer Z+ (N⊕V)=0 ise PC←Etkin adres, SC←0

7. BHI: Eğer daha büyük ise etkin adrese dallanılır.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC06*ADRMD5	Eğer C+Z=0 ise PC←Etkin adres, SC←0

8. BLE: Eğer sıfırdan küçük veya eşitse etkin adrese dallanılır.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC07*ADRMD5	Eğer Z+(N⊕V)=1 ise PC←Etkin adres, SC←0

9. BLS: Eğer daha küçük veya eşitse etkin adrese dallanılır.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC08*ADRMD5	Eğer C+Z=1 ise PC←Etkin adres, SC←0

10. BLT: Eğer sıfırdan küçükse etkin adrese dallanılır.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC09*ADRMD5	Eğer N⊕V=1 ise PC←Etkin adres, SC←0

11. BMI: Eğer işaret biti 1 ise etkin adrese dallanılır.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC10*ADRMD5	Eğer N =1 ise PC←Etkin adres, SC←0

12. BNE: Eğer sıfır biti 0 ise etkin adrese dallanılır.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC11*ADRMD5	Eğer Z=0 ise PC←Etkin adres, SC←0

13. BVC: Eğer taşma biti 0 ise etkin adrese dallanılır.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC12*ADRMD5	Eğer V=0 ise PC←Etkin adres, SC←0

14. BVS: Eğer taşma biti 1 ise etkin adrese dallanılır.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC13*ADRMD5	Eğer V=1 ise PC←Etkin adres, SC←0

15. BPL: Eğer işaret biti 0 ise etkin adrese dallanılır.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC14*ADRMD5	Eğer N=0 ise PC←Etkin adres, SC←0

16. BSR: Şartsız olarak hesaplanan etkin adresteki alt programa dallanılır.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC15*ADRMD5	TR←AR
T4* IDEC15*ADRMD5	AR←SP
T5* IDEC15*ADRMD5	M_{SP}←PC_L, AR←AR-1, SP←SP-1
T6* IDEC15*ADRMD5	M_{SP}←PC_H, AR←AR-1, SP←SP-1
T7* IDEC15*ADRMD5	AR←TR
T8* IDEC15*ADRMD5	PC←Etkin adres, SC←0

17. RTS: Alt programdan, program akışının kaldığı yere döner.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC26*ADRMD0	SP←SP+1
T4* IDEC26*ADRMD0	AR←SP
T5* IDEC26*ADRMD0	PC_H← M_{AR}, AR←AR+1, SP←SP+1
T6* IDEC26*ADRMD0	PC_L← M_{AR}, SP←SP+1, SC←0

18. JMP: Verilen adrese program akışı kaydırılır. Direkt ve indis adresleme modlarına sahiptir.

a) Direkt Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC23*ADRMD2	PC_H←M[AR, AR←AR+1
T4* IDEC23*ADRMD2	PC_L←M[AR], SC←0

b) İndis Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC12*ADRMD4	PC←etkin adres, SC←0

19. JSR: Alt programdan program akışının kaldığı yere dönülmesini sağlar. Direkt ve indis adresleme modlarına sahiptir.

a) Direkt Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC24*ADRMD2	TR←AR, PC←PC+1
T4* IDEC24*ADRMD2	AR←SP
T5* IDEC24*ADRMD2	M_{SP}←PC_L, AR←AR-1, SP←SP-1
T6* IDEC24*ADRMD2	M_{SP}←PC_H, AR←AR-1, SP←SP-1
T7* IDEC24*ADRMD2	AR←TR
T8* IDEC24*ADRMD2	TR_H←M[AR], AR←AR+1

KOMUTUN MİKRO İŞLEM ADIMLARI	
T9* IDEC24*ADRMD2	$TR_L \leftarrow M[AR]$
T10* IDEC24*ADRMD2	PC←TR, SC←0

b) İndis Mod

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC13*ADRMD4	TR←AR, PC←PC+1
T4* IDEC13*ADRMD4	AR←SP
T5* IDEC13*ADRMD4	$M_{SP} \leftarrow PC_L$, AR←AR-1, SP←SP-1
T6* IDEC13*ADRMD4	$M_{SP} \leftarrow PC_H$, AR←AR-1, SP←SP-1
T7* IDEC13*ADRMD4	AR←TR
T8* IDEC13*ADRMD4	PC←etkin adres, SC←0

20. RTI: Kesme işlemi bittikten sonra, program akışı kesilmeden önceki durumuna geri döndürür. Doğal adresleme moduna sahiptir.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC17*ADRMD0	SP←SP+1
T4* IDEC17*ADRMD0	AR←SP
T5* IDEC17*ADRMD0	CCR←M[AR], AR←AR+1, SP←SP+1
T6* IDEC17*ADRMD0	$DR_H \leftarrow M[AR]$, AR←AR+1, SP←SP+1
T7* IDEC17*ADRMD0	$DR_L \leftarrow M[AR]$, AR←AR+1, SP←SP+1
T8* IDEC17*ADRMD0	$IX_H \leftarrow M[AR]$, AC←DR, AR←AR+1, SP←SP+1
T9* IDEC17*ADRMD0	$IX_L \leftarrow M[AR]$, AR←AR+1, SP←SP+1
T10* IDEC17*ADRMD0	$PC_H \leftarrow M[AR]$, AR←AR+1, SP←SP+1
T11* IDEC17*ADRMD0	$PC_L \leftarrow M[AR]$, I←0, SC←0

21. NOP: Sadece program sayıcını bir artırır.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC25*ADRMD0	PC←PC+1, SC←0

22. HLT: Simülasyonu durdurur.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC14*ADRMD0	Durdur

EK-D. Durum Kod Kaydedicisi Komutlarının Mikro İşlem Adımları

1. CLC: Elde bitini sıfırlar.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC19*ADRMD0	C←0, SC←0

2. CLI: Kesme bitini sıfırlar.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC20*ADRMD0	I←0, SC←0

3. CLO: Taşma bitini sıfırlar.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC21*ADRMD0	O←0, SC←0

4. STC: Elde bitini 1 yapar.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC22*ADRMD0	C←1, SC←0

5. STI: Kesme bitini 1 yapar.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC23*ADRMD0	I←1, SC←0

6. STO: Taşma bitini 1 yapar.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC24*ADRMD0	O←1, SC←0

EK-E. Giriş-Çıkış Komutlarının Mikro İşlem Adımları

1. **IN:** Giriş kaydedicisindeki veriyi akümülatöre kaydeder.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC15*ADRMD0	$AC_L \leftarrow INPR$, $SC \leftarrow 0$

2. **OUT:** Akümülatördeki veriyi çıkış kaydedicisine kaydeder.

KOMUTUN MİKRO İŞLEM ADIMLARI	
T3* IDEC16*ADRMD0	$OUTR \leftarrow AC_L$, $SC \leftarrow 0$

AKÜMÜLATÖR VE BELLEK KOMUTLARI																
		Derhal			Direkt			Dolaylı			İndis			Doğal		
İşlem Adı	**Kısaltması**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**
Topla	ADD	10	3	3	20	6	3	30	9	3	40	4	2			
Elde ile Topla	ADDC	11	3	3	21	6	3	31	9	3	41	4	2			
Lojik "VE"	AND	12	3	3	22	6	3	32	9	3	42	4	2			
Temizle	CLR													01	1	1
Karşılaştır	CMP	13	3	3	23	6	3	33	9	3	43	4	2			
Azalt	DECR													02	1	1
Böl	DIV	15	3	3	25	6	3	35	9	3	45	4	2			
Lojik "XOR"	XOR	16	3	3	26	6	3	36	9	3	46	4	2			
Arttır	INCR													03	1	1
1'e Tümleme	COM													04	1	1
2'e Tümleme	NEG													05	1	1
Yükle	LDA	1A		3	2A		3	3A		3						
Lojik "VEYA"	OR	1B	3	3	2B	6	3	3B	9	3	4B	4	2			
İt	PSH													06	4	1
Çek	PUL													07	6	1
Aritmetik sağa Kaydırma	SAR													08	1	1

AKÜMÜLATÖR VE BELLEK KOMUTLARI																
		Derhal			**Direkt**			**Dolaylı**			**İndis**			**Doğal**		
İşlem Adı	**Kısaltması**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**
Aritmetik sola Kaydırma	SAL													09	1	1
Çıkarma	SUB	1E	3	3	2E	6	3	3E	9	3	4E	4	2			
Borç ile Çıkarma	SUBC	1F	3	3	2F	6	3	3F	9	3	4F	4	2			
Sakla	STA				A0	5	3	B0	8	3	C0	3	2			
Çarpma	MUL	95	2	3	A5	6	3	B5	9	3	C5	4	2			

OP : İşlem Kodu

~ : İcra edilme süresi

: Bellekte kapladığı byte sayısı

YIĞIN VE İNDİS KOMUTLARI																
		Derhal			**Direkt**			**Dolaylı**			**İndis**			**Doğal**		
İşlem Adı	**Kısaltması**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**
İndis K. yükle	LDAX	91	2	3	A1	5	3	B1	8	3	C1	3	2			
Yığın K. yükle	LDAS	92	2	3	A2	5	3	B2	8	3	C2	3	2			
İndis K. sakla	STAX				A3	5	3	B3	8	3						
Yığın K.sakla	STAS				A4	5	3	B4	8	3						
İndis K.azalt	DECX													0A	1	1
İndis K.arttır	INCX													0B	1	1
Yığın K.azalt	DECS													0C	1	1
Yığın K.arttır	INCS													0D	1	1

OP : İşlem Kodu

~ : İcra edilme süresi

: Bellekte kapladığı byte sayısı

SIÇRAMA VE DALLANMA KOMUTLARI																
		Göreceli			Direkt			Dolaylı			İndis			Doğal		
İşlem Adı	**Kısaltması**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**
Şartsız dallan	BRA	50	1	2												
C=0 ise dallan	BCC	51	1	2												
C=1 ise dallan	BCS	52	1	2												
Z=1 ise dallan	BZR	53	1	2												
N⊕V=0 ise dallan	BGE	54	1	2												
Z+(N⊕V)=0 ise dallan	BGR	55	1	2												
C+Z=0 ise dallan	BHI	56	1	2												
Z+(N⊕V)=1 ise dallan	BLE	57	1	2												
C+Z=1 ise dallan	BLS	58	1	2												
N⊕V=1 ise dallan	BLT	59	1	2												
N=1 ise dallan	BMI	5A	1	2												
Z=0 ise dallan	BNE	5B	1	2												
V=0 ise dallan	BVC	5C	1	2												
V=1 ise dallan	BVS	5D	1	2												

SIÇRAMA VE DALLANMA KOMUTLARI																
		Göreceli			**Direkt**			**Dolaylı**			**İndis**			**Doğal**		
İşlem Adı	**Kısaltması**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**
N=0 ise dallan	BPL	5E	1	2												
Alt programa dallan	BSR	5F	6	2												
Alt programdan dön	RTS													8A	4	1
Atla	JMP				A7	2	3				4C	1	2			
Alt programa atla	JSR				A8	8	3				4D	6	2			
Kesmeden dönüş	RTI													81	9	1
İşlem yok	NOP													89	1	1
Programı durdur	HLT													0E	1	1

OP : İşlem Kodu

~ : İcra edilme süresi

: Bellekte kapladığı byte sayısı

SIÇRAMA VE DALLANMA KOMUTLARI																
		Göreceli			Direkt			Dolaylı			İndis			Doğal		
İşlem Adı	**Kısaltması**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**
Elde bitini sıfırla	CLC													83	1	1
Kesme bitini sıfırla	CLI													84	1	1
Taşma bitini sıfırla	CLV													85	1	1
Elde bitini bir yap	STC													86	1	1
Kesme bitini bir yap	STI													87	1	1
Taşma bitini bir yap	STV													88	1	1

OP : İşlem Kodu

~ : İcra edilme süresi

: Bellekte kapladığı byte sayısı

GİRİŞ-ÇIKIŞ KOMUTLARI																
		Göreceli			Direkt			Dolaylı			İndis			Doğal		
İşlem Adı	**Kısaltması**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**	**OP**	**~**	**#**
Girişteki bilgiyi alır	IN													0F	1	1
Veriyi çıkışa iletir	OUT													80	1	1

OP : İşlem Kodu

~ : İcra edilme süresi

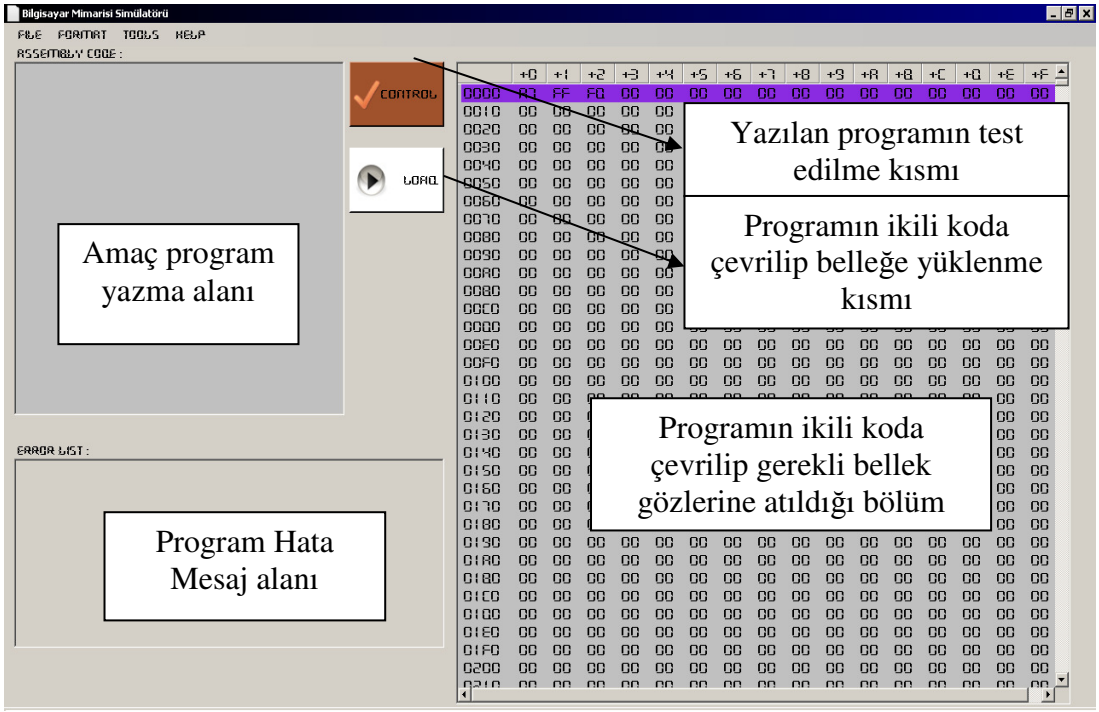
: Bellekte kapladığı byte sayısı

EK-K

Bilgisayar Mimarisi Simülatör programı için yazılan Assembler programı:

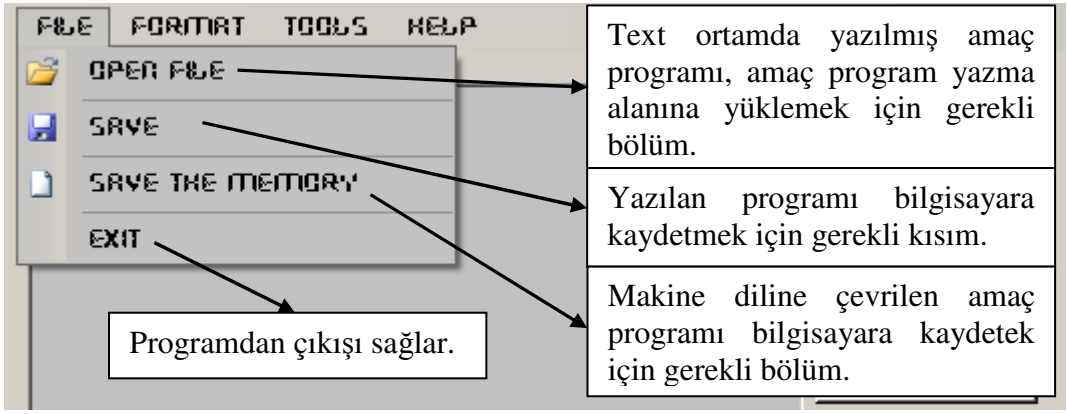
Visual Studio.NET ortamında geliştirilen program, Bilgisayar Mimarisi simülatörü içinde yer alan 64KB lık belleğe ikili kodları gönderen bir programdır. Bu program vasıtasıyla kullanıcı Ek-F ile başlayıp Ek-J ile biten komut tablosu listesini kullanıp amaç programı yazdıktan sonra programın "Load" butonu ile amaç program makine diline çevrilip belleğe yüklemektedir.

Program bilgisayara kurulup çalıştırıldığında aşağıdaki ana ekran görüntüsü elde edilmektedir.

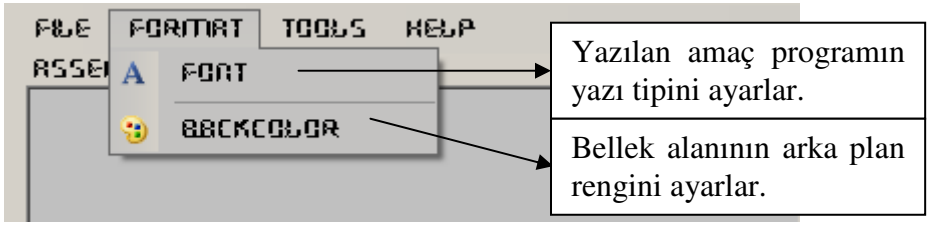


Programın “FILE” menüsünde dört kısım bulunmaktadır. Bu kısımlarla ilgili açıklamalar aşağıda ekran görüntüsü üzerinde verilmektedir.

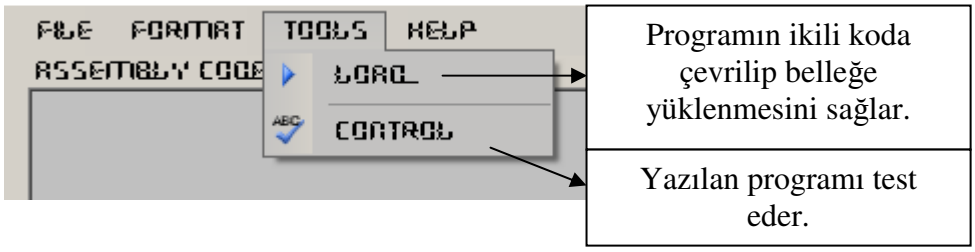
File Menüsü:



Format Menüsü:



Tools Menüsü:



Help Menüsü:

Yazılan programın kullanıcının istifade edebilmesi ve programı daha ileri seviyelere götürebilmesi açısından kullanılan kodlar aşağıda verilmiştir.

AnaEkran.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Collections;

namespace Bellek
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         #region tanımlamalar
         LstBellek.Columns.Add("", -2, HorizontalAlignment.Left);
         LstBellek.Columns[0].Width = 65;
```

```
LstBellek.Columns.Add("+0", -2, HorizontalAlignment.Left);
LstBellek.Columns[1].Width = 40;
LstBellek.Columns.Add("+1", -2, HorizontalAlignment.Left);
LstBellek.Columns[1].Width = 40;
LstBellek.Columns.Add("+2", -2, HorizontalAlignment.Left);
LstBellek.Columns[2].Width = 40;
LstBellek.Columns.Add("+3", -2, HorizontalAlignment.Left);
LstBellek.Columns[3].Width = 40;
LstBellek.Columns.Add("+4", -2, HorizontalAlignment.Left);
LstBellek.Columns[4].Width = 40;
LstBellek.Columns.Add("+5", -2, HorizontalAlignment.Left);
LstBellek.Columns[5].Width = 40;
LstBellek.Columns.Add("+6", -2, HorizontalAlignment.Left);
LstBellek.Columns[6].Width = 40;
LstBellek.Columns.Add("+7", -2, HorizontalAlignment.Left);
LstBellek.Columns[7].Width = 40;
LstBellek.Columns.Add("+8", -2, HorizontalAlignment.Left);
LstBellek.Columns[8].Width = 40;
LstBellek.Columns.Add("+9", -2, HorizontalAlignment.Left);
LstBellek.Columns[9].Width = 40;
LstBellek.Columns.Add("+A", -2, HorizontalAlignment.Left);
LstBellek.Columns[10].Width = 40;
LstBellek.Columns.Add("+B", -2, HorizontalAlignment.Left);
LstBellek.Columns[11].Width = 40;
LstBellek.Columns.Add("+C", -2, HorizontalAlignment.Left);
LstBellek.Columns[12].Width = 40;
LstBellek.Columns.Add("+D", -2, HorizontalAlignment.Left);
LstBellek.Columns[13].Width = 40;
LstBellek.Columns.Add("+E", -2, HorizontalAlignment.Left);
LstBellek.Columns[14].Width = 40;
LstBellek.Columns.Add("+F", -2, HorizontalAlignment.Left);
LstBellek.Columns[15].Width = 40;
#endregion
```

```
            LstBellek.Width = 720;
            LstBellek.Height = 680;
            LstBellek.Left = this.Width - 290;
            LstBellek.MultiSelect = false;
            geriAlToolStripMenuItem.Enabled = false;
            yineleToolStripMenuItem.Enabled = false;
            openFileDialog1.Title = "Choose File";
            openFileDialog1.FileName = "";
            openFileDialog1.Filter = "Text Files|*.txt|" + "All Files|*.*";
            saveFileDialog1.Filter = "Text Files|*.txt|" + "RTF Files|*.rtf|" + "All
Files|*.*";
            saveFileDialog1.Title = "Save File";
            colorDialog1.FullOpen = true;
            fontDialog1.ShowColor = true;
            fontDialog1.ShowEffects = true;
        }

        ArrayList MyList;
        ArrayList ObcodeList = new ArrayList();
        ObCode MyObCode = new ObCode();
        bool kontrol = true;

        private void Form1_Load(object sender, EventArgs e)
        {
            LstBellek.Items.Clear();
            double son = 65536;
            string cevirici = "";
            MyList = new ArrayList();
            uint hexa = 0;
            for (int i = 0; i < son; i = i + 16)
            {
                hexa = Convert.ToUInt32(i);
                cevirici = String.Format("{0:x4}", hexa).ToUpper();
```

```
            MyList.Add(cevirici);
        }

        for (int i = 0; i < MyList.Count; i++)
        {
            LstBellek.Items.Add(MyList[i].ToString());
            LstBellek.Items[i].SubItems.Add("00");
            LstBellek.Items[i].SubItems.Add("00");
            LstBellek.Items[i].SubItems.Add("00");
            LstBellek.Items[i].SubItems.Add("00");
            LstBellek.Items[i].SubItems.Add("00");
            LstBellek.Items[i].SubItems.Add("00");
            LstBellek.Items[i].SubItems.Add("00");
            LstBellek.Items[i].SubItems.Add("00");
            LstBellek.Items[i].SubItems.Add("00");
            LstBellek.Items[i].SubItems.Add("00");
            LstBellek.Items[i].SubItems.Add("00");
            LstBellek.Items[i].SubItems.Add("00");
            LstBellek.Items[i].SubItems.Add("00");
            LstBellek.Items[i].SubItems.Add("00");
            LstBellek.Items[i].SubItems.Add("00");
            LstBellek.Items[i].SubItems.Add("00");
            LstBellek.Items[i].SubItems.Add("00");
            LstBellek.Items[i].SubItems.Add("00");
            LstBellek.Items[i].SubItems.Add("00");
        }
        GirisKesme();

    }

    private void btnBellegeYukle_Click(object sender, EventArgs e)
    {
        kontrol = true;
        string gecerli_adres = "0100";
        int adres = 0;
        for (int c = 0; c < LstBellek.Items.Count; c++)
```

```
        {
            if (LstBellek.Items[c].Text == gecerli_adres)
            {
                adres = c;
            }
        }

        ObcodeList.Clear();
        string kesme = "";
        for (int a = 0; a < RchtxtAssembly.Lines.Length; a++)
        {
            if (RchtxtAssembly.Lines[a].Contains(';'))
            {
                if (RchtxtAssembly.Lines[a].Trim().StartsWith(";") == true)
                {
                    continue;
                }
                else
                {
                    kesme = RchtxtAssembly.Lines[a].Substring(0,
RchtxtAssembly.Lines[a].IndexOf(';')).ToUpper().Trim().ToString();
                }
            }
            else
            {
                kesme = RchtxtAssembly.Lines[a].ToUpper().Trim().ToString();
            }
            ObcodeList.Add(MyObCode.deneme(kesme));
            AyrilacakAlanSayisiBelirleme(kesme);
        }

        for (int b = 0; b < ObcodeList.Count; b++)
        {
```

```
            if (ObcodeList[b].ToString() == "")
            {
                MessageBox.Show("Error was handled", "ERROR MESSAGE",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                btnKontrol_Click(sender, e);
                kontrol = false;
                break;
            }
        }

        if (kontrol == true)
        {
            int yer = 0;
            Random r = new Random();
            try
            {
                Form1_Load(sender, e);
                do
                {
                    LstBellek.Items[adres].BackColor =
Color.FromArgb(r.Next(150,255), r.Next(150,255), r.Next(150,255));
                    LstBellek.Items[adres].SubItems[yer % 16 + 1].Text =
ObcodeList[yer].ToString();
                    yer += 1;
                    if (yer % 16 == 0)
                        adres += 1;
                } while (yer != ObcodeList.Count);
                rchtHataKonsol.Text = "Assembly Code(s) converted to Obcode
successfully - " + DateTime.Now.ToLongTimeString();
            }
            catch
            {
                foreach(var item in RchtxtAssembly.Lines)
```

```
                {
                    if(item.Contains(';')==true)
                    {
                        continue;
                    }
                }
                MessageBox.Show("There's no code to convert!", "ERROR
MESSAGE", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
        else
        {
            ;
        }
    }

    private void btnKontrol_Click(object sender, EventArgs e)
    {
        kontrol = true;
        string assemblycode = RchtxtAssembly.Text.ToUpper().Trim();
        ObcodeList.Clear();
        if (RchtxtAssembly.Text.Trim() == "")
        {
            MessageBox.Show("Assembly Code could not be found", "ERROR
MESSAGE", MessageBoxButtons.OK, MessageBoxIcon.Error);
            rchtHataKonsol.Text = "Assembly Code could not be found";
        }
        else
        {
            ObCodeKontrol(MyObCode);
            rchtHataKonsol.Text = "";
            for (int b = 0; b < ObcodeList.Count;b++)
            {
```

```
                if (ObcodeList[b].ToString() == "")
                {
                    if (RchtxtAssembly.Lines[b].Trim().StartsWith(";") == true)
                    {
                        continue;
                    }
                    kontrol = false;
                    rchtHataKonsol.Text = rchtHataKonsol.Text + "Incorrect syntax : " +
"Line : " + (b + 1).ToString() + " \r\n";
                }
                else
                {
                    ;
                }
            }
            if (kontrol == true)
            {
                rchtHataKonsol.Text = rchtHataKonsol.Text + "Assembly Code(s)
converted successfully - " + DateTime.Now.ToLongTimeString();
            }
        }
    }

    private ArrayList ObCodeKontrol(ObCode MyObCode)
    {
        for (int a = 0; a < RchtxtAssembly.Lines.Length; a++)
        {

ObcodeList.Add(MyObCode.deneme(RchtxtAssembly.Lines[a].ToUpper().Trim().T
oString()));
        }
        return ObcodeList;
    }
```

```
private void AyrilacakAlanSayisiBelirleme(string metin)
{
    int ayrilacakAlanSayisi = 0;
    string son_metin="";
    bool sorgu = true;
    if (metin.Contains('#') == true)
    {
        sorgu = true;
        son_metin = metin.Substring(metin.Length - 5);
        son_metin = son_metin.Substring(0, 4);
        if (sorgu == true)
        {
            ayrilacakAlanSayisi = 3;
            ObcodeList.Add(son_metin.Substring(0, 2));
            ObcodeList.Add(son_metin.Substring(son_metin.Length - 2));
        }
    }
    else if (metin.Contains('$') == true)
    {
        son_metin = metin.Substring(metin.Length - 5);
        son_metin = son_metin.Substring(0, 4);
        if (sorgu == true)
        {
            ayrilacakAlanSayisi = 3;
            ObcodeList.Add(son_metin.Substring(0, 2));
            ObcodeList.Add(son_metin.Substring(son_metin.Length - 2));
        }
    }
    else if (metin.Contains('@') == true)
    {
        son_metin = metin.Substring(metin.Length - 5);
        son_metin = son_metin.Substring(0, 4);
```

```
        if (sorgu == true)
        {
            ayrilacakAlanSayisi = 3;
            ObcodeList.Add(son_metin.Substring(0, 2));
            ObcodeList.Add(son_metin.Substring(son_metin.Length - 2));
        }
    }
    else if (metin.Contains('%') == true)
    {
        son_metin = metin.Substring(metin.Length - 3);
        ayrilacakAlanSayisi = 2;
        son_metin = son_metin.Substring(0, 2);
        ObcodeList.Add(son_metin);
    }
    else if (metin.Contains('*') == true)
    {
        son_metin = metin.Substring(metin.Length - 3);
        ayrilacakAlanSayisi = 2;
        son_metin = son_metin.Substring(0, 2);
        ObcodeList.Add(son_metin);
    }
    else
    {
        ayrilacakAlanSayisi = 1;
    }
}

private void CikToolStripMenuItem_Click(object sender, EventArgs e)
{
    Environment.Exit(0);
}

private void KesToolStripMenuItem_Click(object sender, EventArgs e)
```

```
{
   RchtxtAssembly.Cut();
}

private void KopyalaToolStripMenuItem_Click(object sender, EventArgs e)
{
   RchtxtAssembly.Copy();
}

private void YapistirToolStripMenuItem_Click(object sender, EventArgs e)
{
   RchtxtAssembly.Paste();
}

private void silToolStripMenuItem_Click(object sender, EventArgs e)
{
   RchtxtAssembly.Clear();
}

private void TumunuSecStripMenuItem1_Click(object sender, EventArgs e)
{
   RchtxtAssembly.SelectAll();
}

private void geriAlToolStripMenuItem_Click(object sender, EventArgs e)
{
   if (RchtxtAssembly.CanUndo == true)
   {
      RchtxtAssembly.Undo();
   }
   if (RchtxtAssembly.CanUndo == false)
   {
      geriAlToolStripMenuItem.Enabled = false;
```

```
        }
    }

    private void RchtxtAssembly_TextChanged(object sender, EventArgs e)
    {
        if (RchtxtAssembly.CanUndo == true)
            geriAlToolStripMenuItem.Enabled = true;
        if (RchtxtAssembly.CanRedo == true)
            yineleToolStripMenuItem.Enabled = true;
    }

    private void yineleToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if (RchtxtAssembly.CanRedo == true)
        {
            RchtxtAssembly.Redo();
        }
        else
        {
            yineleToolStripMenuItem.Enabled = false;
        }
    }
    private void programHakkındaToolStripMenuItem_Click(object sender,
EventArgs e)
    {
        Hakkında frm = new Hakkında();
        frm.ShowDialog();
    }

    private void bellekArkaPlanToolStripMenuItem_Click(object sender,
EventArgs e)
    {
        if (colorDialog1.ShowDialog() == DialogResult.OK)
```

```
        {
            LstBellek.BackColor = colorDialog1.Color;
        }
    }

    private void yazıTipiToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if (fontDialog1.ShowDialog() == DialogResult.OK)
        {
            RchtxtAssembly.Font = fontDialog1.Font;
            RchtxtAssembly.ForeColor = fontDialog1.Color;
        }
    }

    private void bellegeYukleToolStripMenuItem_Click(object sender, EventArgs
e)
    {
        btnBellegeYukle_Click(sender, e);
    }

    private void koduKontrolEtToolStripMenuItem_Click(object sender, EventArgs
e)
    {
        btnKontrol_Click(sender, e);
    }

    private void Program_Kullanımı(object sender, EventArgs e)
    {
        ProgramKullanimi myfrm = new ProgramKullanimi();
        myfrm.ShowDialog();
    }

    private void Txt_Ac(object sender, EventArgs e)
```

```
    {
        if (openFileDialog1.ShowDialog() == DialogResult.OK)
        {
            try
            {
                RchtxtAssembly.LoadFile(openFileDialog1.FileName,
RichTextBoxStreamType.PlainText);
            }
            catch
            {
                MessageBox.Show("File cannot be opened");
            }
        }
    }

    private void Txt_Kaydet(object sender, EventArgs e)
    {
        if (saveFileDialog1.ShowDialog() == DialogResult.OK)
        {
            try
            {
                RchtxtAssembly.SaveFile(saveFileDialog1.FileName,
RichTextBoxStreamType.PlainText);
            }
            catch
            {
                MessageBox.Show("File cannot be saved");
            }
        }
    }
    private void BizeUlasin_Click(object sender, EventArgs e)
    {
        BizeUlasin yenifrm = new BizeUlasin();
```

```
        yenifrm.ShowDialog();
    }

    private void openFileDialog1_FileOk(object sender, CancelEventArgs e)
    {
        try
        {
            RchtxtAssembly.LoadFile(openFileDialog1.FileName,
RichTextBoxStreamType.PlainText);
        }
        catch
        {
            e.Cancel = true;
            MessageBox.Show("This file is not suitable for this program");
        }
    }

    private void GirisKesme()
    {
        LstBellek.Items[0].BackColor = Color.BlueViolet;
        LstBellek.Items[LstBellek.Items.Count - 1].BackColor = Color.BlueViolet;
        LstBellek.Items[0].SubItems[1].Text = "A7";
        LstBellek.Items[0].SubItems[2].Text = "FF";
        LstBellek.Items[0].SubItems[3].Text = "FD";
        LstBellek.Items[LstBellek.Items.Count-1].SubItems[14].Text = "0F";
        LstBellek.Items[LstBellek.Items.Count-1].SubItems[15].Text = "80";
        LstBellek.Items[LstBellek.Items.Count-1].SubItems[16].Text = "81";
    }

    private void BellegiKaydet_Click(object sender, EventArgs e)
    {
        if (saveFileDialog1.ShowDialog() == DialogResult.OK)
        {
```

```
            try
            {
               int b=0;
               System.IO.TextWriter dosya =
System.IO.File.CreateText(saveFileDialog1.FileName);
               for (int a = 0; a < LstBellek.Items.Count; a++)
               {
                  for ( b = 1; b < 17; b++)
                  {
                     if (a == LstBellek.Items.Count - 1 && b == 16)
                     {
                        dosya.Write(LstBellek.Items[a].SubItems[b].Text);
                     }
                     else
                        dosya.WriteLine(LstBellek.Items[a].SubItems[b].Text);
                  }
               }
               dosya.Close();
            }
            catch
            {
               MessageBox.Show("File cannot be saved");
            }
         }
      }
   }
}
```

Obcode.cs

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;
namespace Bellek
{
   public class ObCode
   {
      ArrayList Komutlar = new ArrayList();
      ArrayList ObCodeList = new ArrayList();
      public string deneme(string metin)
      {
         #region Tüm Komutlar
         Komutlar.Add("ADD #"); ObCodeList.Add("10");
         Komutlar.Add("ADD $"); ObCodeList.Add("20");
         Komutlar.Add("ADD @"); ObCodeList.Add("30");
         Komutlar.Add("ADD %"); ObCodeList.Add("40");
         Komutlar.Add("ADC #"); ObCodeList.Add("11");
         Komutlar.Add("ADC $"); ObCodeList.Add("21");
         Komutlar.Add("ADC @"); ObCodeList.Add("31");
         Komutlar.Add("ADC %"); ObCodeList.Add("41");
         Komutlar.Add("AND #"); ObCodeList.Add("12");
         Komutlar.Add("AND $"); ObCodeList.Add("22");
         Komutlar.Add("AND @"); ObCodeList.Add("32");
         Komutlar.Add("AND %"); ObCodeList.Add("42");
         Komutlar.Add("CMP #"); ObCodeList.Add("13");
         Komutlar.Add("CMP $"); ObCodeList.Add("23");
         Komutlar.Add("CMP @"); ObCodeList.Add("33");
         Komutlar.Add("CMP %"); ObCodeList.Add("43");
         Komutlar.Add("DIV #"); ObCodeList.Add("15");
         Komutlar.Add("DIV $"); ObCodeList.Add("25");
         Komutlar.Add("DIV @"); ObCodeList.Add("35");
         Komutlar.Add("DIV %"); ObCodeList.Add("45");
```

```
Komutlar.Add("XOR #"); ObCodeList.Add("16");
Komutlar.Add("XOR $"); ObCodeList.Add("26");
Komutlar.Add("XOR @"); ObCodeList.Add("36");
Komutlar.Add("XOR %"); ObCodeList.Add("46");
Komutlar.Add("LDA #"); ObCodeList.Add("1A");
Komutlar.Add("LDA $"); ObCodeList.Add("2A");
Komutlar.Add("LDA @"); ObCodeList.Add("3A");
Komutlar.Add("LDA %"); ObCodeList.Add("4A");
Komutlar.Add("ORR #"); ObCodeList.Add("1B");
Komutlar.Add("ORR $"); ObCodeList.Add("2B");
Komutlar.Add("ORR @"); ObCodeList.Add("3B");
Komutlar.Add("ORR %"); ObCodeList.Add("4B");
Komutlar.Add("SUB #"); ObCodeList.Add("1E");
Komutlar.Add("SUB $"); ObCodeList.Add("2E");
Komutlar.Add("SUB @"); ObCodeList.Add("3E");
Komutlar.Add("SUB %"); ObCodeList.Add("4E");
Komutlar.Add("SUBC #"); ObCodeList.Add("1F");
Komutlar.Add("SUBC $"); ObCodeList.Add("2F");
Komutlar.Add("SUBC @"); ObCodeList.Add("3F");
Komutlar.Add("SUBC %"); ObCodeList.Add("4F");
Komutlar.Add("LDAX #"); ObCodeList.Add("91");
Komutlar.Add("LDAX $"); ObCodeList.Add("A1");
Komutlar.Add("LDAX @"); ObCodeList.Add("B1");
Komutlar.Add("LDAX %"); ObCodeList.Add("C1");
Komutlar.Add("LDAS #"); ObCodeList.Add("92");
Komutlar.Add("LDAS $"); ObCodeList.Add("A2");
Komutlar.Add("LDAS @"); ObCodeList.Add("B2");
Komutlar.Add("LDAS %"); ObCodeList.Add("C2");
Komutlar.Add("MUL #"); ObCodeList.Add("95");
Komutlar.Add("MUL $"); ObCodeList.Add("A5");
Komutlar.Add("MUL @"); ObCodeList.Add("B5");
Komutlar.Add("MUL %"); ObCodeList.Add("C5");
Komutlar.Add("CLR"); ObCodeList.Add("01");
```

```
Komutlar.Add("DECR"); ObCodeList.Add("02");
Komutlar.Add("INCR"); ObCodeList.Add("03");
Komutlar.Add("COMR"); ObCodeList.Add("04");
Komutlar.Add("NEGR"); ObCodeList.Add("05");
Komutlar.Add("PSH"); ObCodeList.Add("06");
Komutlar.Add("PUL"); ObCodeList.Add("07");
Komutlar.Add("SAR"); ObCodeList.Add("08");
Komutlar.Add("SAL"); ObCodeList.Add("09");
Komutlar.Add("DECX"); ObCodeList.Add("0A");
Komutlar.Add("INCX"); ObCodeList.Add("0B");
Komutlar.Add("DECS"); ObCodeList.Add("0C");
Komutlar.Add("INCS"); ObCodeList.Add("0D");
Komutlar.Add("HLT"); ObCodeList.Add("0E");
Komutlar.Add("IN"); ObCodeList.Add("0F");
Komutlar.Add("OUT"); ObCodeList.Add("80");
Komutlar.Add("RTI"); ObCodeList.Add("81");
Komutlar.Add("RTS"); ObCodeList.Add("8A");
Komutlar.Add("CLC"); ObCodeList.Add("83");
Komutlar.Add("CLI"); ObCodeList.Add("84");
Komutlar.Add("CLO"); ObCodeList.Add("85");
Komutlar.Add("STC"); ObCodeList.Add("86");
Komutlar.Add("STI"); ObCodeList.Add("87");
Komutlar.Add("STO"); ObCodeList.Add("88");
Komutlar.Add("NOP"); ObCodeList.Add("89");
Komutlar.Add("BRA *"); ObCodeList.Add("50");
Komutlar.Add("BCC *"); ObCodeList.Add("51");
Komutlar.Add("BCS *"); ObCodeList.Add("52");
Komutlar.Add("BZR *"); ObCodeList.Add("53");
Komutlar.Add("BGE *"); ObCodeList.Add("54");
Komutlar.Add("BGR *"); ObCodeList.Add("55");
Komutlar.Add("BHI *"); ObCodeList.Add("56");
Komutlar.Add("BLE *"); ObCodeList.Add("57");
Komutlar.Add("BLS *"); ObCodeList.Add("58");
```

```
            Komutlar.Add("BLT *"); ObCodeList.Add("59");
            Komutlar.Add("BMI *"); ObCodeList.Add("5A");
            Komutlar.Add("BNE *"); ObCodeList.Add("5B");
            Komutlar.Add("BVC *"); ObCodeList.Add("5C");
            Komutlar.Add("BVS *"); ObCodeList.Add("5D");
            Komutlar.Add("BPL *"); ObCodeList.Add("5E");
            Komutlar.Add("BSR *"); ObCodeList.Add("5F");
            Komutlar.Add("STA $"); ObCodeList.Add("A0");
            Komutlar.Add("STA @"); ObCodeList.Add("B0");
            Komutlar.Add("STA %"); ObCodeList.Add("C0");
            Komutlar.Add("STAX $"); ObCodeList.Add("A3");
            Komutlar.Add("STAX @"); ObCodeList.Add("B3");
            Komutlar.Add("STAS $"); ObCodeList.Add("A4");
            Komutlar.Add("STAS @"); ObCodeList.Add("B4");
            Komutlar.Add("JMP $"); ObCodeList.Add("A7");
            Komutlar.Add("JMP %"); ObCodeList.Add("4C");
            Komutlar.Add("JSR $"); ObCodeList.Add("A8");
            Komutlar.Add("JSR %"); ObCodeList.Add("4D");
            #endregion
            string obcode = "";
            string aradeger = "";
            for (int i = 0; i < Komutlar.Count; i++)
            {
               aradeger = Komutlar[i].ToString();
               if (metin.IndexOf(aradeger)>=0)
               {
                  obcode = ObCodeList[i].ToString();
               }
            }
            return obcode;
         }
      }
   }
```

ÖZGEÇMİŞ

Halit ÖZTEKİN, 16.03.1979 da Kocaeli'nin Gölcük ilçesine bağlı Halıdere beldesinde doğdu. İlkokulu Halıdere beldesinde tamamladıktan sonra, orta ve lise eğitimini Gölcük'te tamamladı. 1995 yılında Barboros Hayrettin Lisesinden mezun oldu. 1997 yılında başladığı SAÜ Bilgisayar Mühendisliği bölümünü 2002 yılında bölüm birincisi olarak bitirdi. 2002-2005 yılları arasında İzmit Halk Eğitim Merkezi, Bimser gibi kuruluşlarda eğitmen olarak görev yaptı. 2007 yılında bir süre Kocaeli Büyükşehir Belediyesinin Strateji ve Geliştirme daire başkanlığında donanım şefi olarak görev yaptı. 2007 Eylül döneminde SAÜ Bilgisayar ve Bilişim Mühendisliği bölümünde yüksek lisansa başladı. 2008 yılının başından itibaren çok arzuladığı akademisyenlik hayatının başlangıcı olan Araştırma Görevlisi olarak Sakarya Üniversitesi Bilgisayar Mühendisliği bölümünde göreve başladı ve halen bu görevde akademisyenlik hayatını sürdürmektedir.