İŞLEMCİLERDE YAZMAÇ NUMARASI SAKLAMA ALANLARINI, ORTAKLAŞA GÖSTERİM YÖNTEMİ İLE, GEÇİCİ HATALARA KARŞI DAYANIKLI HALE GETİRME

EMRAH İŞLEK

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ

TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

NİSAN 2015

ANKARA

Fen Bilimleri Enstitü onayı

Prof. Dr. Osman EROĞUL
Müdür

Bu tezin Yüksek Lisans derecesinin tüm gereksinimlerini sağladığını onaylarım.

Doç. Dr. Erdoğan DOĞDU
Anabilim Dalı Başkanı

EMRAH İŞLEK tarafından hazırlanan İŞLEMCİLERDE YAZMAÇ NUMARASI SAKLAMA ALANLARINI, ORTAKLAŞA GÖSTERİM YÖNTEMİ İLE, GEÇİCİ HATALARA KARŞI DAYANIKLI HALE GETİRME adlı bu tezin Yüksek Lisans tezi olarak uygun olduğunu onaylarım.

Doç. Dr. Oğuz ERGİN
Tez Danışmanı

Tez Jüri Üyeleri

Başkan : Prof. Dr. Mehmet Önder EFE

Üye : Doç. Dr. Oğuz ERGİN

Üye : Doç. Dr. Ali BOZBEY

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, ayrıca tez yazım kurallarına uygun olarak hazırlanan bu çalışmada orijinal olmayan her türlü kaynağa eksiksiz atıf yapıldığını bildiririm.

Emrah İşlek

Üniversitesi : **TOBB Ekonomi ve Teknoloji Üniversitesi**
Enstitüsü : **Fen Bilimleri**
Anabilim Dalı : **Bilgisayar Mühendisliği**
Tez Danışmanı : **Doç. Dr. Oğuz ERGİN**
Tez Türü ve Tarihi : **Yüksek Lisans – Nisan 2015**

Emrah İşlek

İŞLEMCİLERDE YAZMAÇ NUMARASI SAKLAMA ALANLARINI, ORTAKLAŞA GÖSTERİM YÖNTEMİ İLE, GEÇİCİ HATALARA KARŞI DAYANIKLI HALE GETİRME

ÖZET

Günümüz teknolojisi, çok düşük kaynak gerilimlerinde çalışabilen, gittikçe ufalan boyutlarda transistör üretilebilmesine ve içerisinde çok daha sık transistör olan işlemcilerin üretilebilmesine olanak sağlamıştır. Fakat bu işlemciler enerji yüklü parçacık çarpması sonucu oluşan geçici hatalara karşı daha hassas hale gelmiştir. Bu durum özellikle elektromanyetik gürültünün fazla olduğu uzay gibi yerlerde önemli bir güvenilirlik problemine dönüşmektedir. Günümüzde işlemcileri bu tarz hatalara karşı korumak için çeşitli hata düzeltme algoritmaları ve koruyucu donanımlar geliştirilmiştir.

İşlemcilerde yazmaç numarasını saklayan alanlarda oluşabilecek geçici hatalar, işlemci üzerinde çalışmakta olan programın farklı yazmaç numarasındaki veriyi kullanması sonucu çok büyük hatalara sebep olabilmektedir. Bu tez çalışmasında, TÜBİTAK'ın 112E004 nolu "Geçici Hatalara Karşı Dayanıklı Mikroişlemciler" projesi kapsamında işlemcilerde yazmaç numarasını saklayan alanları korunaklı hale getirmek için tasarladığımız bir tekniğe ve analiz sonuçlarına yer verilmiştir.

Anahtar Kelimeler: İşlemciler, geçici hatalara karşı koruma, yazmaç numarası, ortaklaşa gösterim.

University	:	**TOBB University of Economics and Technology**
Institute	:	**Institute of Natural and Applied Sciences**
Science Programme	:	**Computer Engineering**
Supervisor	:	**Assoc. Prof. Oğuz ERGİN**
Degree Awarded and Date	:	**M.Sc. – April 2015**

Emrah İşlek

COLLECTIVE POINTING: PROTECTING POINTER VALUES AGAINST SOFT ERRORS ON PROCESSORS

ABSTRACT

Today's technology provides us with the capability of producing low-voltage, increased transistor density processors. However these advancements come with side effects as well, they make processors more susceptible to high energy particles. These susceptibilities turn into security issues and make the processors' vulnerability to errors when these processors' operating environment is changed to places like high altitude atmosphere or space due to increasing electromagnetic interference. Luckily, there are also technologies developed for safe-guarding against these sort of errors, such as error correction algorithms and shielding hardware.

The errors occurring at places storing the register id number used for addressing other registers can cause fatal errors by making the processor use wrong data coming from the wrong register. In this thesis, we have designed a technique for making these places store the register indices in a safe-guarded way against these errors and finally we also have shown our findings. This thesis was supported by a TUBITAK project titled "Reliable Microprocessors Design" under grant number 112E004.

Keywords: Processors, protection against transient errors, register number, collective pointing.

TEŞEKKÜR

Bu çalışmayı tamamlamamda emeği geçen değerli danışman hocam Doç. Dr. Oğuz Ergin'e; çalıştığım projeme destek veren TÜBİTAK'a; kıymetli çalışma arkadaşlarım Vehbi Eşref Bayraktar, Serdar Zafer Can, Mustafa Çavuş ve Abdullah Giray Yağlıkçı başta olmak üzere bütün Kasırga Ailesine; iş arkadaşım Kamil Yavuz Kapusuz'a; çalışma ortamımızı ve burs sağladığı için TOBB ETÜ Mühendislik Fakültesi ve Fen Bilimleri Enstitüsüne ve tez çalışmam sırasında beni destekleyen aileme teşekkür ederim.

İçindekiler

Şekil Listesi

Tablo Listesi

1. GİRİŞ

Günümüzde, teknolojinin gelişmesi ile beraber işlenmesi gereken veri miktari günden güne artmaktadır. İnsanoğlu artan bu verilerin kısa sürede işlenebilmesi için ise daha hızlı, güvenilebilir bir işlemciye gereksinim duyulmaktadır. Örneğin, eskiden standart bir kamera, standart çözünürlük olan 640x480 piksellik bir görüntü alıp işlerken, şu anda 4k denilen 4096x2160 piksel çözünürlükte görüntü alabilir hale gelmiştir. İşlenmesi gereken veri miktarı kat ve kat artmıştır.

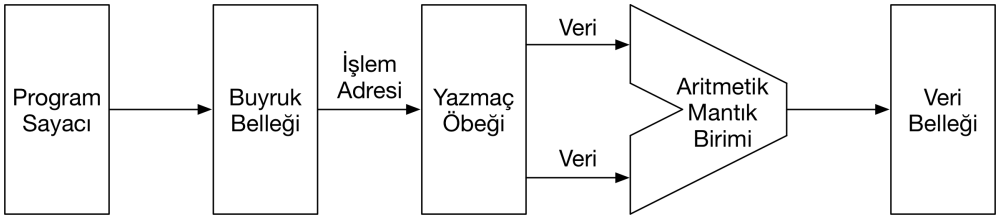
Yarı iletken üretim teknolojisinin gelişmesi, git gide daha küçük boyutlardaki transistörlerin, daha uygun fiyatlarda üretilebilmesine olanak sağlamıştır. Bu durum Moore Yasası'nda da belirtildiği üzere, daha ufak işlemcilere daha fazla transistör sığdırılmasını sağlamıştır [1]. Transistörlerin ufalarak daha küçük alanlara daha fazla transistör sığması durumu, birim alanda güç artışını ve ısı yayılımı gibi problemleri de beraberinde getirmiştir.

Enerji yüklü taneciklerin neden olduğu geçici hatalar, özellikle elektromanyetik gürültünün fazla olduğu ortamlarda çalışan işlemciler için büyük bir güvenlik sorunu haline gelmiştir. Buna ek olarak, yarı iletken teknolojisinin gelişmesi ile beraber performans için artırılan frekans, düşük güç tüketimi ve ısı yayılımı sorunları gibi bazı olaylar bu sorunu daha şiddetli bir duruma getirmektedir.

Geçici hatalar sürekli olmadığından dolayı, işlemcilerde performans azalması olmaması için, bu tip hataların denetimlerinin seri bir şekilde yapılabilmesi gerekmektedir. Aksi durumlarda, işlemcilerdeki boru hattında bir darboğaz oluşturmakta ve işlemlerin fazladan beklemesine, bunun sonucu olarak ise ciddi performans azalmalarına sebebiyet verebilmektedir. Geçici hata denetimi için kullanılan en temel teknikler hata düzeltme kodu ve eşlik biti denetimi olmuştur.

İşlemcilerde hesaplama sırasında kullanılan veriler, tekrar kullanılacağı zamana kadar, geçici olarak işlemcinin yazmaç öbeğinde saklanmaktadır. Şekil 1.1'deki basit bir işlemci tasarımında da gösterildiği gibi, yazmaç öbeği işlemci üzerinde yürütülen herhangi bir işlem sırasında sürekli kullanılan bir birimdir. Bu nedenle

işlemcinin en önemli bileşenlerinden biri haline gelen yazmaç öbeği, geçici hatalar konusunda işlemci için en kritik bölgelerden biridir. Burada oluşabilecek hatalar, işlemcinin diğer bütün bileşenlerine rahatlıkla aktarılabilmekte ve daha büyük hatalara, hatta çalışan programların çökmesine sebep olabilmektedir. Bu yüzden işlemcilerde yazmaç öbeklerinin geçici hatalara karşı korunması hayati bir önem taşımaktadır.



Şekil 1.1: Basit bir işlemci tasarımı

Yazmaç öbeklerinde saklanan verilerin okunabilmesi için, istenilen adresteki bilginin satır numarası yazmaç öbeğine verilmekte ve ilgili satırdaki bilgi okunabilmektedir. Yamaç öbeğine verilen yazmaç numarası hatalı ise, yazmaç öbeğinden yanlış bir satır okunması sonucunda, istenen veriden farklı bir veri okunacaktır. Bu durumda ise işlemci yanlış çalışacaktır. Bu yüzden yazmaç numaralarının hatasız olması, işlemcinin doğru çalışması için hayati bir öneme sahiptir. Bu tez çalışmasında, yazmaç numaralarının saklandığı alanların, daha güvenli hale getirilebilmesi için tasarlanan yeni bir yönteme yer verilmiştir.

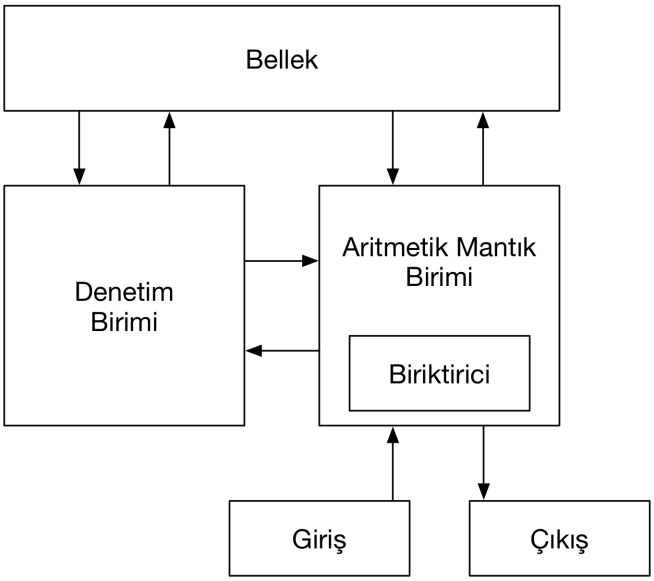
2. TEMEL KONULAR

Bu bölüm, işlemcilerle ilgili bazı temel bilgiler ve hata düzeltme teknikleri anlatılarak, tez çalışmasında bahsedilen konuların daha iyi anlaşılması için sunulmuştur.

2.1 Basit Bir İşlemcinin Tasarımı

İşlemciler, milyonlarca transistörün bir araya gelmesi ile oluşturulmuş, bir bilgisayarı oluşturan en temel ve en önemli bileşenlerindendir. İşlemciler, bilgisayarın hafızasına yüklenmiş olan program buyruklarını belirli bir sıra ile içerisine alıp, basit ya da karmaşık matematiksel işlemler ile işlemektedirler. İşlenilen buyrukların sonucunda hesaplanan sonuçlara göre, gereken işlemleri yapmaktadırlar. Başka işlemleri tetikleme, giriş kapılarını okuma ya da çıkış kapılarına veri yazma gibi.

Bir işlemciyi genel olarak, denetim birimi ve aritmetik mantık birimi olmak üzere iki kısımda inceleyebiliriz. 1945'te tanımlanmış Von Neumann işlemci modelini Şekil 2.1'de görebilirsiniz. Bu tasarımda, bellek veri saklanmasında, aritmetik mantık birimi matematik ve mantık işlemlerinin yapılmasında, denetim birimi ise zamanlama ve denetim işlemlerinde kullanılmaktadır.



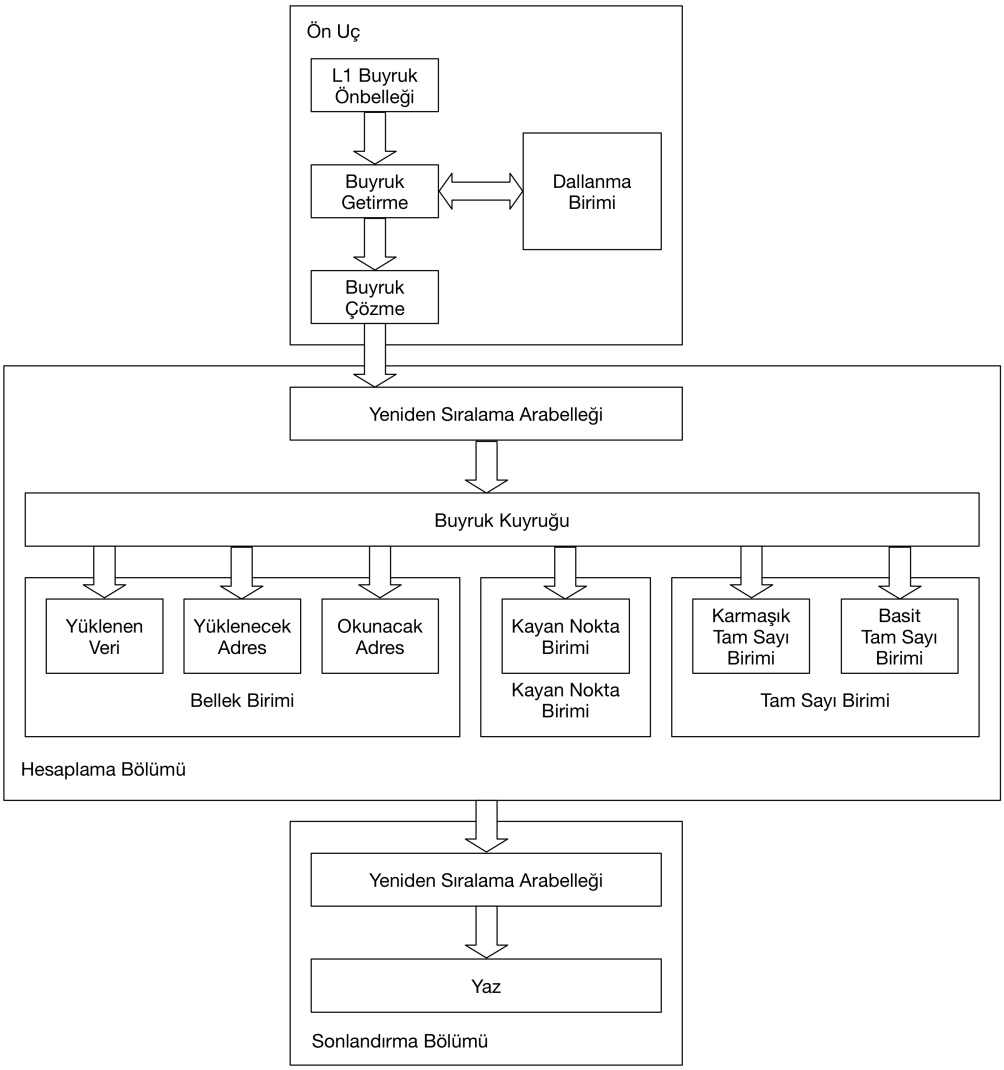
Şekil 2.1: Von Neumann Mimarisi

Von Neumann Mimarisi, 1945 yılında John Von Neumann tarafından tanımlanmış bir mimari olup, bütün modern bilgisayarlar için bir şablon olmuştur [2].

İlkel işlemcilerde sıralı işleme yöntemi kullanılmıştır. Bu tasarıma göre bir programın çalışması için, öncelikle program buyrukları sırayla ana bellekten işlemciye getirilmekte, ardından yürütme birimlerinde işlenmektedirler. Buyrukların yürütülmesi sırasında buyruğun türüne göre ana bellekten veri çekme, yazma ya da yazmaç kullanma tarzında işlemler olabilmektedir. Buyruk yürütülmesi bittiğinde, getirilmesi gereken buyruğun adresini tutan program sayacı artar ve ardından yeni buyruk işleme alınmaktadır. Bu şekilde, sırası gelen bütün buyruklar sırayla işlenerek program tamamlanmaktadır.

Sıralı işleme tasarımı basit bir tasarım olsada performans olarak zayıf kalmaktadır. Zaman içerisinde performansı artırabilmek için yeni tasarımlar düşünülmüş ve uygulamaya konmuştur. Bunlardan birisi yeniden adlandırma yöntemidir. Bu yöntem, buyruklarda geçen yazmaçlar arasında birbirleri ile bağımlılığı olmayanların isimleri değiştirilerek, birbirinin işlemlerini beklemesini engellemeye

dayanmaktadır. Böylelikle, getirilen buyruklar önce yeniden adlandırmaya tabi tutulur, ardından bağımsız olanlar kendisinden önceki buyrukların sonuçlanmasını beklemesine gerek kalmadan yürütme işlemi yapılabilmektedir. Son olarak, sonuçlar yazılmadan önce ise karışık olan sonuçlar yeniden sıralanır ve çıktı olarak verilir. Bu mimariye örnek olarak Pentium 3 işlemcisinin basitleştirilmiş tasarımı Şekil 2.2'de verilmiştir [3]. Bu şekilde Pentium 3 işlemcisi, yaptığı işleme göre üç temel parçaya bölünmüştür. Ön uç olarak adlandırılmış ilk bölümde, işlemcinin işlemesi için buyruk getirme, dallanma işlemleri ve getirilen buyruğu çözme işlemleri yapılmaktadır. Hesaplama bölümü olarak adlandırılmış olan ikinci bölümde ise, yeniden adlandırma ve ardından hesaplama işlemleri yapılmaktadır. Sonlandırma bölümü olarak belirtilen yerde ise, sırası karışık olarak tamamlanmış buyrukların tekrar sıralı hale sokulması ve gerekli yerlere yazılma işlemleri yapılmaktadır.



Şekil 2.2: Pentium 3 işlemcisinin bölümleri

Intel'in tasarladığı bu işlemci 32 bitlik işlem yapma yeteneğine sahip olup, 9.5 milyon transistörden oluşmaktadır. Çalışma frekansı ise maksimum 1.4 Ghz'dir.

2.1.1 Boru Hattı Mimarisi

Boru hattı, günümüz işlemcilerinin başarımını artırmak için kullanılan en temel yöntemlerden biri olmuştur. Amacı, buyrukların işlenmesi sırasında boş duran kaynakları kullanılabilir hale getirebilmektir. Bu tasarım sayesinde birim sürede yürütülen ve biten buyruk sayısı artmaktadır.

Bu tasarıma göre bir işlemci işini beş ana aşamada yapabilmektedir. Bu aşamalar ve yapılan işlemler şöyle özetlenebilir:

Yakala : Buyruklar, buyruk belleğinden getirilirler.

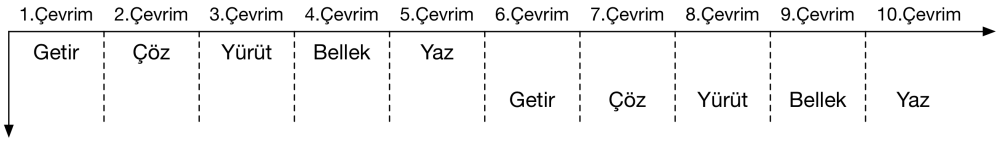
Çöz : Verilerin hangi yazmaçlarda olduğunu ve hangi işlemin yapılacağı çözülür.

Yürüt : Sonuç ya da adres hesaplamaları yapılır.

Bellek : Veriler bellekten alınır ya da yazılır.

Sonuç : Bulunan sonuçlar belleğe yazılır.

Boru hattı olmayan bir işlemcide 2 buyruğun geçtiği bu beş aşama Şekil 2.3'te gösterildiği gibi olmaktadır. Bu 2 buyruk toplamda 10 çevrimlik bir zamanında tamamlanmaktadır.



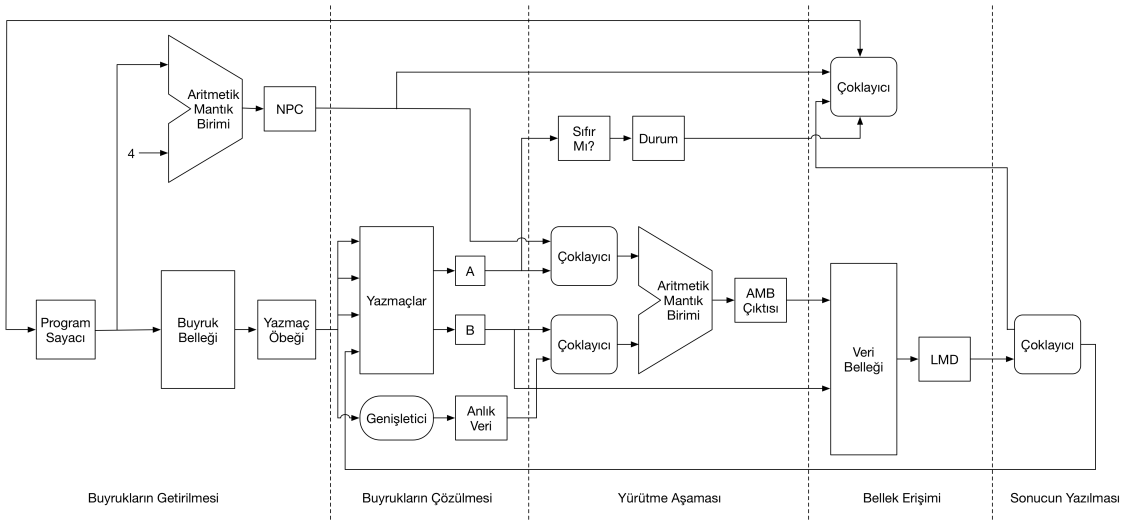
Şekil 2.3: Boru hattı kullanmayan bir işlemcinin aşamaları

Boru hattı mimarisini kullanan bir işlemcide ise bu aşamalar Şekil 2.4'te gösterildiği gibi olmaktadır. Bu çizelgeye göre 10 çevrimlik bir zamanda toplam 6 adet buyruk tamamlanmaktadır.

Şekil 2.4: Boru hattı kullanan bir işlemcinin aşamaları

Şekillerden de görüldüğü üzere, bu örneğe göre boru hattı tasarımına sahip bir işlemci, normal işlemciye göre 3 kat daha hızlı çalışmaktadır. Tasarımdan tasarıma, boru hattı aşamalarına ve bölündüğü işlem sayısına bağlı olarak bu artış farklılık gösterebilmektedir.

Boru hattı tasarımı kullanan MIPS (İngilizcesi: Microprocessor without Interlocked Pipeline Stages) mimarisinde buyrukların akış diyagramı Şekil 2.5'te gösterilmiştir. MIPS indirgenmiş komut kümesi (İngilizcesi: RISC) kullanmaktadır. Bu yüzden tasarımı oldukça sadedir. Bu mimaride, karmaşık işlemlerin uzun sürede yapılması yerine, basit işlemlerin daha hızlı yapılması yoluna gidilmiştir. Bu tasarımda kullanılan A, B, LMD, NPC gibi yapılar boru hattındaki zamanlamanın ayarlanması ve verilerin paralel olarak ilerleyebilmesi için kullanılan, verileri geçici olarak tutan yapılardır. Şekilden de görülebileceği üzere, buyrukların getirilmesi aşamasında bulunan aritmetik mantık birimi program sayacının düzgün ilerlemesi için toplama işlemi yaparken, yürütme aşamasındaki aritmetik mantık birimi ise buyruklardaki hesaplama işlemlerini gerçekleştirmektedir.



Şekil 2.5: MIPS mimarisi

Bu tasarımın çalışma aşamaları şu şekilde özetlenebilir:

Buyrukların getirilmesi : Program sayacının gösterdiği buyruk, buyruk belleğinden çekilir ve program sayacı çekilen buyruğun uzunluğuna, yapılan işleme göre değişir.

Buyrukların çözülmesi : Bu aşamada verilerin hangi yazmaçlarda olduğunu ve hangi işlemin yapılacağı çözülür.

Yürütme aşaması : İşlenecek olan buyruğa göre sonuç ya da adres hesaplamaları yapılır.

Bellek erişimi : Veriler veri belleğinden alınır ya da yazılır.

Sonucun yazılması : Bulunan sonuçlar duruma göre yazmaçlara yazılır.

2.1.2 Buyruk Seviyesi Paralelleştirme

İşlemcileri hızlandırmak için kullanılan başka bir yöntem ise buyrukların paralel çalıştırılmasıdır. Çok yollu işlemciler aynı anda birden fazla buyruğu eş zamanlı olarak çalıştırabilmektedirler. Bunun sonucu olarak da bir çevrimlik zamanda aynı anda birden fazla buyruğu bitirebilmektedirler. Bu işlem, buyruk seviyesindeki paralellik sayesinde olabilmektedir.

Buyruk seviyesinde paralelleşirme için boru hattındaki aşamalar biraz değişikliğe uğramış olup, şöyle özetlenebilir:

Buyruk çek : Buyruklar çekilir. Bu birimde buyruk önbelleği ve dallanma tahmin mantık birimi bulunmaktadır.

Çöz : Buyrukların türleri ve hangi kaynağı kullanacağı çözülür.

Yeniden adlandırma / Yer atama : Buyruklardaki bağımlılığı ortadan kaldırmak için, buyruklardaki yazmaçlar yeniden adlandırılır. Ardından yeniden sıralama arabelleğine, buyruk kuyruğuna ve yükle/sakla kuyruklarına iletir. Buralarda boş yer yoksa eğer, buyrukları kaynaklar tekrar boşalana kadar bekletir ve sonra yükler.

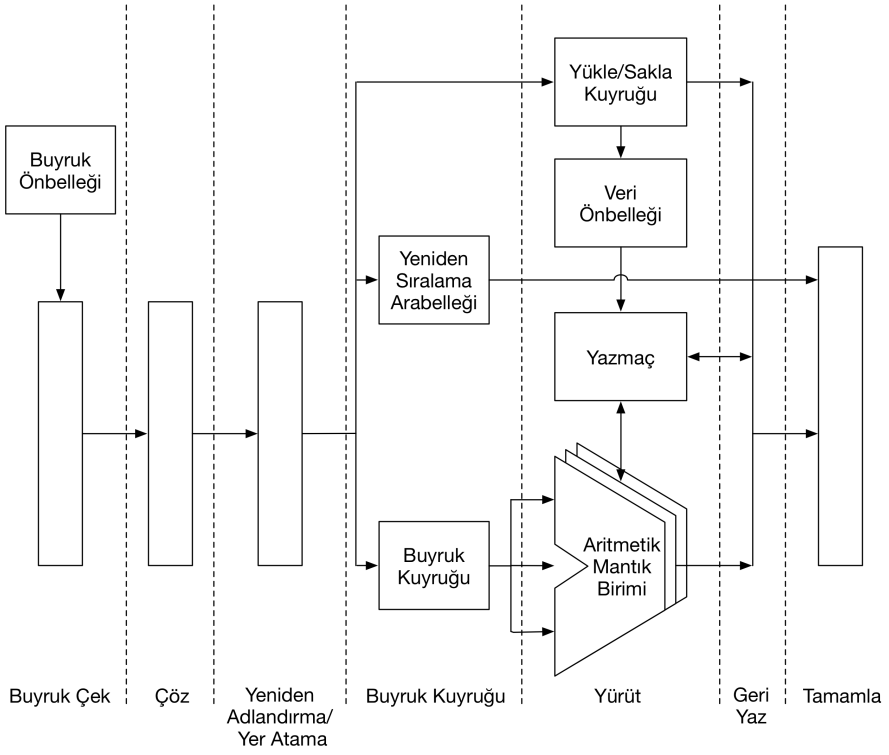
Buyruk kuyruğu : Buyruklar yürütme yapan birimler boşalana kadar burada bekletilir. Buyrukların kullanacağı kaynaklar boşa çıktığı zaman, yürütme birimlerine gönderilirler.

Yürüt : Buyruklar bu aşamada yürütülürler. Burada tam sayı, kayan noktalı sayı, mantıksal işlemler gibi farklı işlemleri yapan farklı birimler bulunmaktadır.

Geri yaz : Sonuçlar yazmaçlara ya da yükle/sakla kuyruklarına yazılırlar.

Tamamla : Tamamlanan buyruklar sıralanır. Eğer kendinden önceki buyruklar tamamlanmış ise o buyruk çıkabilir. Yanlış tahmin edilmiş bir buyruk ise de bunun etkisi burada yok edilir.

Çok yollu (İngilizcesi: superscalar) işlemcilerin çalışma modeli Şekil 2.6'da gösterilmiştir.



Şekil 2.6: Çok yollu işlemci çalışma modeli

Buyruk seviyesinde paralelleştirme yapısını kullanan, çok yollu işlemci olarak Pentium 4 işlemcisini örnek olarak verebiliriz [3]. Çok yollu işlemcilerde buyrukların paralel çalışabilmesi, yeniden sıralama arabelleği ve buyruk kuyruğu gibi bazı yapıların kullanılması ile mümkün hale gelmiştir. Bunlarla ilgili detaylı bilgi devam eden bölümlerde verilecektir.

2.2 Bazı Donanım Birimleri

İşlemciler, kendi içlerinde farklı işler için özelleştirilmiş farklı birimler bulundurmaktadırlar. Aritmetik mantık birimi, yazmaç öbeği, buyruk kuyruğu, yeniden sıralama arabelleği gibi bir çok birimleri vardır.

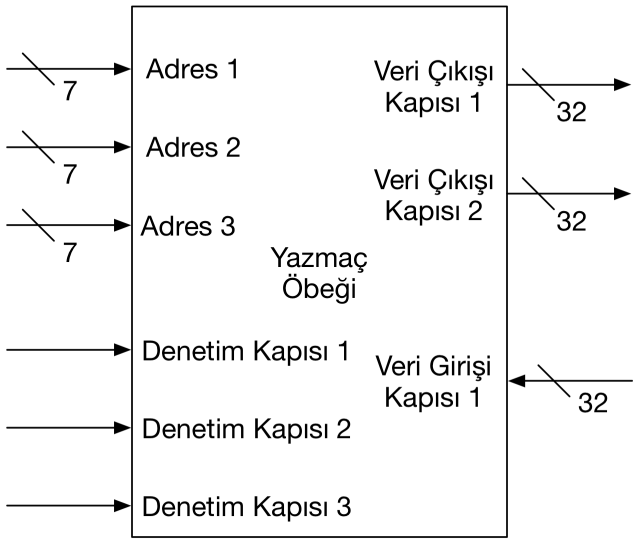
Her birim yapılan işlem türüne göre ve içerisinde uygulanan algoritmaya göre farklı zamanlarda işlemlerini gerçekleştirmektedirler. Örneğin, Intel Pentium 4 işlemcinin aritmetik mantık birimi, toplama işlemini 1 ile 3 saat vuruşu arasında yaparken, bölmeyi 17 ile 41, çarpmayı ise 11 saat vuruşunda gerçekleştirmektedir. Bu saat farkından dolayı farklı buyruklar farklı zamanlarda tamamlanabilmektedirler.

İşlemcilerin performanslarını artırmak için gün geçtikçe yeni birimler ve yeni teknikler ortaya çıkmaktadır. Çok yollu işlemcilerde, performans için boru hattı ve buyruk seviyesinde paralelleştirme yöntemi kullanılmaktadır. Buyruklar önceki bölümlerde de belirtildiği gibi yürütme zaman farklarından dolayı sırasız biçimde sonlanmakta ve birbirine bağımlı olan buyruklar kilitlenmeye yol açabilmektedirler. Bu yüzden bazı donanım birimlerine ihtiyaç duyulmuştur. Buyruk kuyruğu, yeniden adlandırma tablosu ve yeniden sıralama arabelleği bu donanım birimlerinden bazılarıdır.

2.2.1 Yazmaç Öbeği

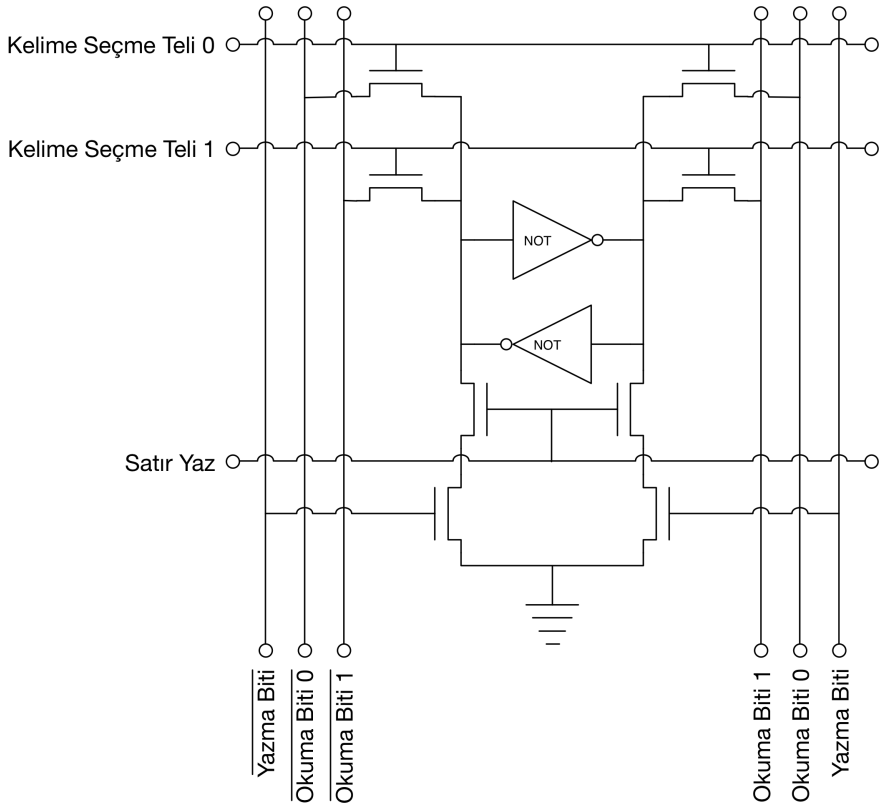
Yazmaç öbeği işlemci içindeki adres ve verileri tutmak için kullanılan geçici depolama alanıdır. İçerisindeki veriler belirli bir adresleme prensibi ile tutulmakta olup, yine bu adresler ile erişim sağlanmaktadır.

Şekil 2.7'de kelime uzunluğu 32 bit olan 128 satırlık basit bir yazmaç öbeği şeması görülmektedir. Buradaki yazmaç öbeği 7 bitlik 2'si okumak için 1'i ise yazmak için olan toplam 3 adet adres bilgisi, bu adresler için 1'er bitlik 3 adet denetim girişi ve veriler için ise 32 bitlik 2 adet çıkış, 1 adet ise giriş kapısı bulunmaktadır.



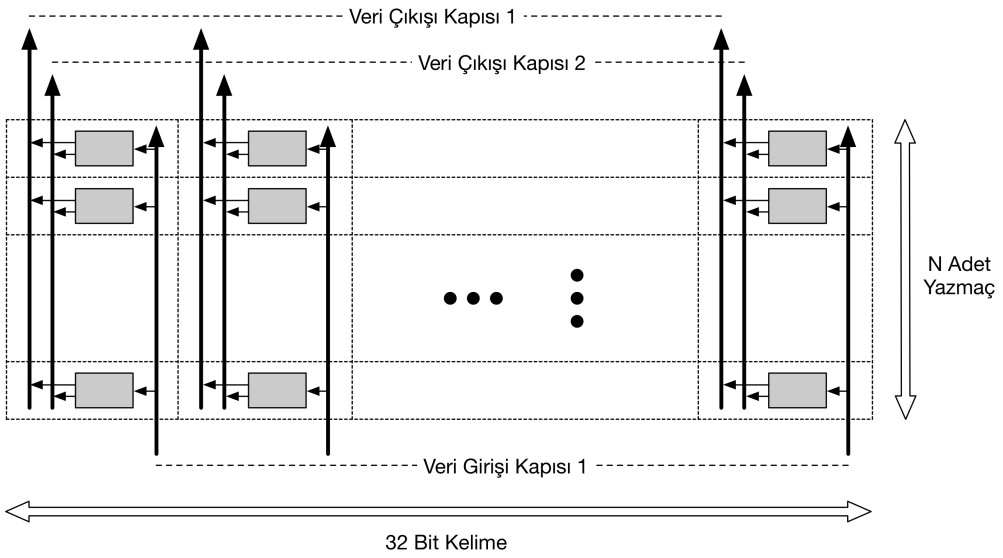
Şekil 2.7: Yazmaç öbeği

Yazmaç öbeğinin içerisinde verileri tutan çok fazla miktarda yazmaç öbeği hücreleri bulunmaktadır. Basit bir yazmaç öbeği hücresi Şekil 2.8'de görülmektedir. Bu hücreler, birbirine paralel olarak ve ters bağlanmış iki adet eviriciden ve bu eviricilere bağlı transistörlerden oluşmaktadır. Şekildeki bit hücresi 2 adet okuma ve 1 adet ise yazma için kapıya sahiptir. Hafızada tutulması gereken durum satır yaz teline bağlı transistörlerin aktif edilmesi ile yazma biti tellerinden eviricilere yüklenir ve tekrar satır yaz teline bağlı transistörlerin pasif yapılması ile eviciler üzerinde korunmaktadır. Okuma durumunda ise kelime seçme tellerine bağlı transistörler aktif hale getirilip, okuma bitleri telleri üzerinden içerisindeki veri okunabilmektedir.



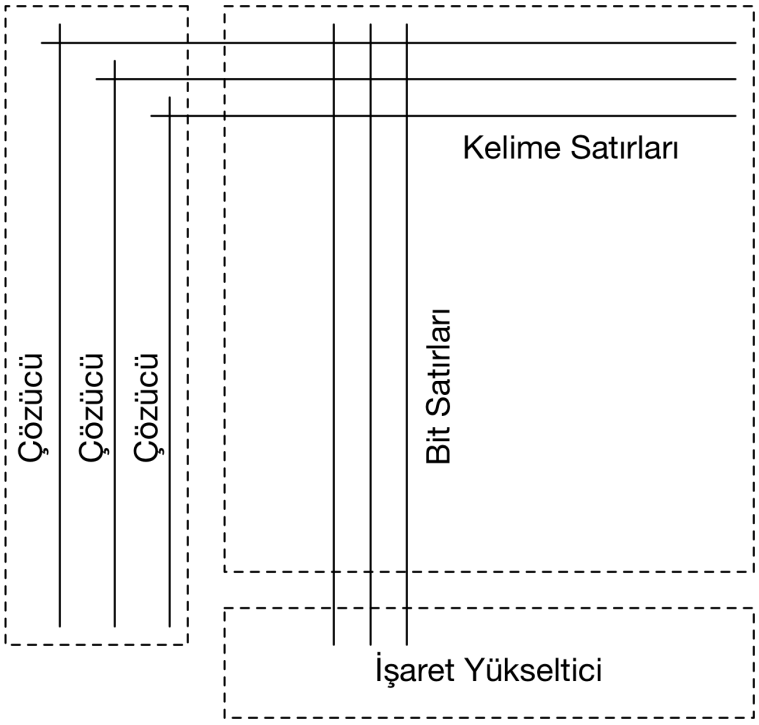
Şekil 2.8: Yazmaç öbeği hücresi

Bir yazmaç öbeği Şekil 2.9'da görüldüğü üzere 'kelime uzunluğu' x 'yazmaç adeti' kadar bit hücresinden oluşmaktadır. Bu bit hücrelerinin, şekilde de görüldüğü üzere aynı sütunda olanlarının veri okuma ve yazma telleri, aynı satırda olanlarının ise denetim telleri birbirine paralel bağlanmıştır.



Şekil 2.9: Yazmaç öbeği satır ve sütun yapısı

Son olarak, bir yazmaç öbeğinin genel bölümleri Şekil 2.10'da gösterildiği gibidir. Girişine verilen bir yazmaç numarası kod çözücü devre ile çözüldükten sonra okunacak ya da yazılacak olan kelime satırının seçilmesi sağlanır. Daha sonra kullanılacak olan kapıya göre kelime teli seçimi yapılarak okuma ya da yazma işlemleri yapılmaktadır. Okunan bitleri tutan eviricilerin yükleri zayıf olduğu için en son işaret yükselticilerden geçerek kesin veriler elde edilmektedir.



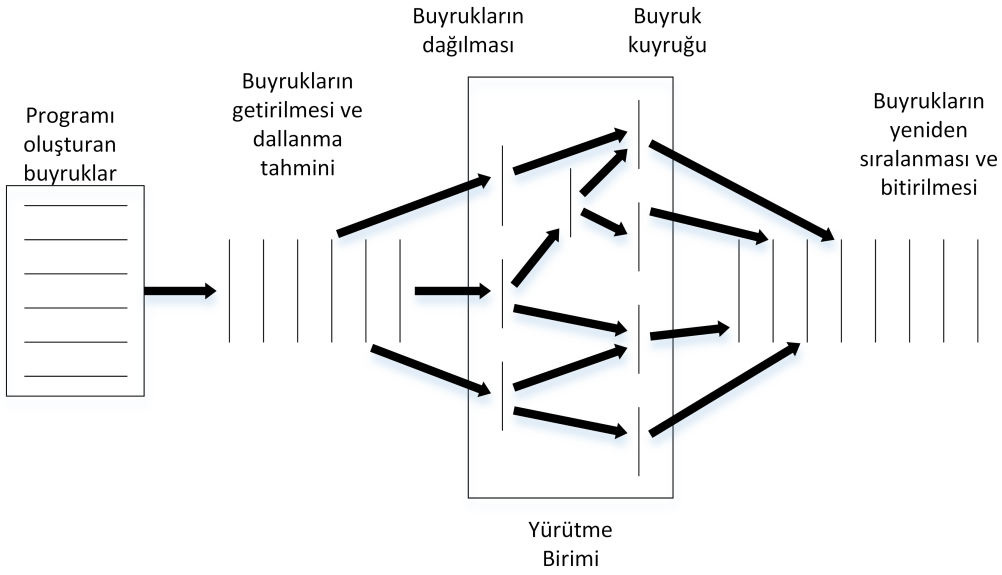
Şekil 2.10: Yazmaç öbeği genel yapısı

2.2.2 Buyruk Kuyruğu

Buyruk kuyruğu (İngilizcesi: instruction-issue queue), Şekil 2.6'da da görüldüğü gibi çok yollu işlemci modelinde sırasız olarak buyrukların çalışmasını sağlayan birimlerden biridir. Bu birimde, yeniden adlandırılması yapılmış buyruklar tutulmakta olup, yürütme birimlerine buradan dağıtılmaktadırlar. Bu birimde, buyrukların kullandıkları yazmaçların yanında ekstra bir bitlik geçerlilik bitleri de bulunmaktadır. Bağımlı olan yazmaçlar hazır olduğunda bu bitler ile gösterilir ve uygun olan birimlere gönderilmektedirler.

2.2.3 Yeniden Sıralama Arabelleği

Yeniden sıralama arabelleği (İngilizcesi: reorder buffer), sırasız olarak işlem yapan işlemcilerde (İngilizcesi: out-of-order execution), sırasız olarak tamamlanmış olan buyrukları tekrardan orjinal programın sırasına sokmak için tasarlanmış bir donanımdır. Yeniden sıralama arabelleği buyruklar için bir sıralı liste tutmaktadır. Yürütme işlemi bitmiş buyruklar bu listeye eklenir ve listenin başından itibaren tamamlanmış olan buyruklar sıralı olarak listeden çıkartılmaktadırlar. Böylelikle buyruklar işlemcide farklı zamanlarda tamamlansalarda, ilk girdikleri sırada çıkmaktadırlar. Bu durum Şekil 2.11'de de gösterilmiştir.



Şekil 2.11: Çok yollu işlemcilerde buyruk yürütülmesi

2.2.4 Yeniden Adlandırma Tablosu

Yeniden adlandırma tablosu (İngilizcesi: rename table), işlemcide buyrukları sıralı olarak almanın gereksiz olduğu yerlerde, sırasız olarak kullanabilmek için tasarlanmıştır. Bu birimde yapılan işlem, buyruklar arası aynı yazmaç kullanımından dolayı oluşan sahte bağımlılığı ortadan kaldırmak ve işlemcinin

sırasız olarak buyrukları yürütülebilmesine olanak vermektir.

Örneğin, Tablo 2.1'de gösterildiği gibi bir programımız olduğunu varsayabiliriz. Burada ilk olarak bellekten veri getirme, toplama ve sonrasında belleğe tekrardan sonuç verisini yazma işlemleri yapılmaktadır. Bellekten veri okuma ve yazma işlemleri zaman alan bir işlemdir. Bu aşamada işlemci boşta beklemek zorunda kalacak ve performanstan kayıp yaşayacaktır.

Tablo 2.1: Yeniden adlandırma öncesi durum

#	**Buyruk**
1	R1 = Bellek[1000]
2	R1 = R1 + 27
3	Bellek[1020] = R1
4	R1 = Bellek[1008]
5	R1 = R1 + 7
6	Bellek[1024] = R1

Tablo 2.1'e 4. buyruktan itibaren bakıldığında geri kalan işlemlerin önceki buyruklar ile bir ilişkisinin bulunmadığı görülmektedir. İşte buna benzer durumları çözebilmek amacıyla düşünülmüş olan yeniden adlandırma yöntemine göre buyruklar, yeniden adlandırma ile Tablo 2.2'de gösterildiği şekle dönüştürülüp, bu bağımlılıktan kurtulabilmektedirler. Bu sayede işlemcinin boşa beklemesinin de önüne geçilmiş olacaktır. Bunun sonucunda, 1-2-3 buyruklarının ve 4-5-6 buyruklarının farklı iki yürütme biriminde, paralel olarak yürütülebilmesine olanak verilebilmektedir.

Tablo 2.2: Yeniden adlandırma sonrası durum

#	**Buyruk**
1	R1 = Bellek[1000]
2	R1 = R1 + 27
3	Bellek[1020] = R1

#	**Buyruk**
4	R2 = Bellek[1008]
5	R2 = R2 + 7
6	Bellek[1024] = R2

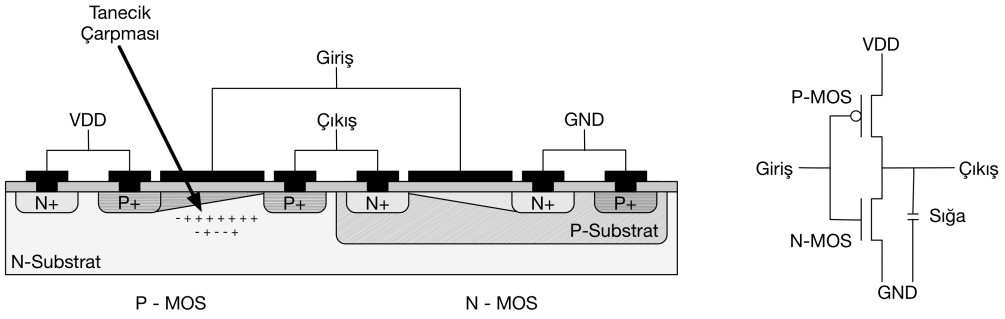
2.3 Geçici Hatalar ve Nedenleri

2.3.1 Geçici Hatalar

Geçici hatalar (İngilizcesi: soft errors), elektronikte ve hesaplamada sinyallerde ya da verilerde çeşitli nedenler sonucu ortaya çıkan bozulmalardır. Bu hatalar, sistemde yer alan bir sorundan dolayı oluşabileceği gibi, dış etkenlerden kaynaklı da oluşabilirler. Bu tip hatalar, donanıma kalıcı olarak herhangi bir zarar vermeseler bile, yürütülmekte olan işlemlerin yanlış çalışmasına ve programın sonlanmasına sebep olabilmektedirler.

Geçici hatalar meydana geldikleri yere göre iki tür olarak incelenebilir:

İşlemci seviyesinde oluşan geçici hatalar : Bu tip hatalar, genellikle ışımalar sonucu oluşan yüklü parçacıkların işlemciye çarparak, Şekil 2.12'de gösterildiği gibi işlemci içindeki transistörlerin hatalı çalışması sonucu oluşan hatalardır.



Şekil 2.12: Mosfet üzerine parçacık çarpması

Bu tarz hatalar sonucunda, işlemci içerisindeki yazmaçlarda tutulan bitlerdeki veriler değişebilmektedir. Bu durum tek bit değişimi (İngilizcesi: single event upset) olarak da bilinmektedir.

Sistem seviyesinde oluşan geçici hatalar : Bu tip hatalar, elektronik kart üzerinde veri aktarımı sırasında, verinin çeşitli nedenlerden dolayı hatalı iletilmesi sonucu oluşan hatalardır. Genellikle veri yollarındaki sorunlardan ve bozuk parçadan kaynaklı olabilmektedir.

2.3.2 Geçici Hataların Nedenleri

Geçici hataların oluştuğunun tespiti ve bu konu üzerindeki araştırmalar, IBM'in ilk zamanlarda yaptığı bilgisayarlarda oluşan sorunlar sonucu ortaya çıkmış ve daha sonra çeşitli bilim adamlarının yaptığı çalışmalar sonucunda ise netlik kazanmıştır [4].

1960'lı yıllarda ilk uydu denemeleri sırasında uzaydan gelen kozmik ışımalar sonucunda oluşan, enerji yüklü parçacıkların uydulardaki elektronik devrelerde hatalara sebep olduğu tespit edilmiş ve bunun üzerine devrelere güvenlik için çeşitli devrelerin eklenmesi ve MYLAR denilen koruyucu bir battaniye kullanılması yoluna gidilmiştir. Bu sorunun ardından yer yüzüne kadar kozmik akı ile gelen bu yüklü parçacıkların işlemciler üzerindeki etkileri incelenmiştir. Deniz seviyesine yakın yerlerde bu etkinin daha az olduğu, yüksek yerlerde ise daha fazla olduğu tespit edilmiştir.

1970'li yıllarda entegrelerin seramik paketlenmesi yaygınlaşırken, Green River üzerine inşaa edilen yeni fabrikanın kullandığı su ve seramikten kaynaklanan, işlemcilerde geçici çökme sorunu ile karşılaşılmış ve bu sorunun kaynağının uranyum yatakları üzerinden geçen Green River'ın taşıdığı alfa taneciklerinin sebep olduğu yine IBM'in çalışmaları sonucunda anlaşılmıştır. Daha sonra yapılan çalışmalarla işlemcilerin paketlenesinde kullanılan bazı malzemelerin yeterince ısındıklarında ışıma yaparak geçici hatalara sebep olabildikleri anlaşılmıştır [5].

Geçmişten günümüze insanoğlu, her geçen gün hep daha iyisini istemiştir. Bu beklenti, bilgisayar donanımlarında ise genellikle performans ve düşük güç tüketimli mobil cihazlar üzerine olmuştur. Gelişen üretim teknolojisi ile daha ufak

boyutlarda üretilebilen transistör altyapısı ile beraber, düşük güç tüketimi mümkün hale gelmiştir. Düşük eşik gerilimlerinde çalışabilen daha ufak boyutlardaki transistörler ile oluşturulan kapılardaki sığaların azalması sayesinde işlemciler daha az güç tüketir hale gelmiştir. Ayrıca gelişen teknoloji, işlemcinin büyüklüğünün artmadan birim alan içerisine daha sık ve daha fazla transistör sığdırılmasına olanak sağlamıştır. Fakat bu gelişmeler işlemcinin dışarıdan gelebilecek etkilere karşı daha hassas olmasına neden olmaktadır. İstenmeyen yüklü bir parçacık çarpması sonucunda aynı anda daha fazla transistör, daha kolay etkilenebilir hale gelmektedir.

2.4 Hata Denetimi ve Düzeltme Teknikleri

Hata denetimi ve düzeltme teknikleri, hem iletişimde, hem de donanım üzerinde okunup yazılan ya da iletilen verileri hatalardan korumak için kullanılan yöntemlerdir. İletişim sırasında oluşan parazitler ve bazı sorunlar verilerin yanlış aktarılmasına neden olmaktadır. Donanımda ise, verilerin işlendiği birimlerde, depolandığı hücrelerde ve iletiminde oluşabilecek geçici hatalar sonucu veriler bozulabilmektedirler. Bu tez çalışmasında donanımda oluşan hatalar ele alınmıştır. Genel olarak, hata denetimi ve düzeltme teknikleri, orjinal veriye denetim bitleri eklenmesi ve daha sonra bu bitlerin tekrar geri çözümlenmesi ile yapılmaktadır. Başlıca bazı hata denetimi ve düzeltme teknikleri şunlardır:

2.4.1 Veri Yenileme

Veri yenileme yöntemi ile hata düzeltmesi basit bir mantıktan oluşmaktadır. Bu yönteme göre, bir veri yazılacağı zaman en az 3 farklı yere yazılmaktadır. Bu veriler okunurken birbirleri ile karşılaştırılmakta ve birinde hata olması durumunda, diğerlerinde aynı ise buralardaki veri geçerli olmaktadır.

Örneğin, tutulan verimizin 1101 olduğunu varsayalım. Bu 4 bitlik veri hafızada saklanırken 3 farklı yerde saklanır. 3. tutulan yerde hata olduğunu farz edelim. Daha sonraki okumada veriler 1101, 1101, 1100 olarak okunur. Karşılaştırma işlemi sonucunda anlaşılır ki 3. verideki değer hatalıdır. Bunun sonucu, ilk iki verideki gibi, yani 1101 şeklinde olmalıdır. Bu yöntem basit bir yöntem olup, fazladan en az 2 kat bellek alanı işgal etmektedir.

2.4.2 Eşlik Biti

Basit bir hata denetim tekniği olan eşlik biti, kodlanan veri bitlerindeki 1 lerin sayısının tek ya da çift olmasına bakmaktadır. N bitlik bir kod, eşlik bitinin eklenmesi ile N+1 bit olmaktadır. Verideki 1 bitlerinin çift olması durumuna çift eşlilik, tek olmasına tek eşlilik denilmektedir.

Tablo 2.3'te çift ve tek eşliliğe göre verilerin sonuna eklenen bitler gösterilmiştir. Bu adım yapıldıktan sonra denetim biti eklenmiş veriler belleğe yazılır. Son adım olarak ise, okunan verilerde çift eşlilik ya da tek eşlilik yapıldığı bilinen kodlardaki 1 sayıları esas alınarak, verilerde 1 bitlik hata olup olmadığı anlaşılmaktadır. Bu mekanizma ayrı iki bitte oluşan hatayı ve 1 bitlik hata var ise de yerini tespit edememekte olup, fazladan 1 bitlik alana ihtiyaç duymaktadır.

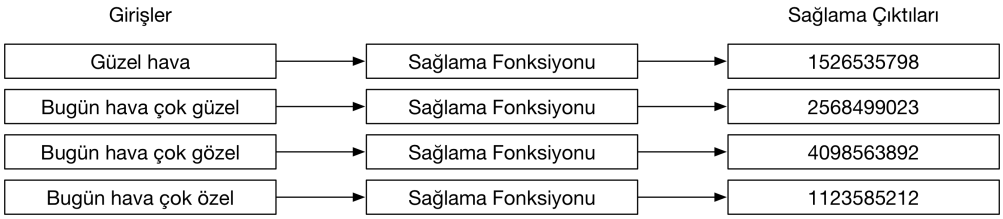
Tablo 2.3: Eşlik biti ile verilerin düzenlenmesi

Orjinal veri	Çift eşlik biti kullanılarak	Tek eşlik biti kullanılarak
0101010	01010101	01010100
1111111	11111111	11111110

2.4.3 Sağlama

Sağlama (İngilizcesi: checksum) genellikle yükleme dosyalarında kullanılmaktadır. Amacı, dosyalarda bozulma olup olmadığını anlamaktır. Sağlamanın mantığı dosyanın belirli yerlerinden rastgele değerler alarak, daha sonra tekrar aynı

değerler ile karşılaştırmaya dayanmaktadır. Sağlama algoritmaları farklılık göstermektermektedir. En iyi algoritma, dosya üzerindeki en ufak farklılıklarda büyük değişimlerde sonuç üretebilen algoritmadır. Şekil 2.13'te görüldüğü gibi girdilerki değişmeler, hesaplanan sağlama değerlerinin değişmesine neden olmaktadır. Böylelikle dosyalar üzerinde hata olup olmadığı anlaşılabilmektedir. Özellikle kurulum dosyalarının üzerinde bu denetimin yapılmasının sebebi, dosyalarda zararlı kişiler tarafından istenerek ya da istenmeyen bozulmalardan oluşabilecek herhangi bir değişim olup olmadığını tespit edebilmektir.



Şekil 2.13: Sağlama fonksiyonları

2.4.4 Döngüsel Artıklık Denetimi

Veri iletiminde ve saklanması sırasında kullanılan bir yöntemdir. Döngüsel artıklık denetimi yönteminde, bir blok içerisindeki veriler toplanır ve belirlenmiş bir sayıya bölünür. Kalan sayısı, iletilen verilerin sonunda gönderilir ya da saklanır. Bu işlem sonrasında, alıcı kısımda ya da okunurken bu işlemin tersi yapılır. Bütün veriler yine toplanır ve en sona eklenmiş olan kalan sayısı çıkarılıp, belirlenmiş olan sayıya tekrar bölünür. Eğer okunan verilerde hata olmamış ise kalanın 0 çıkması gerekmektedir. Aksi taktirde verinin hatalı olduğu tespit edilmiş olur.

2.4.5 Hash Fonksiyonları

Hash fonksiyonları farklı veriler için yüksek oranda benzersiz ve sabit uzunlukta kodlar üretebilen fonksiyonlardır. Bu fonksiyonlar, aynı olan veriler için her

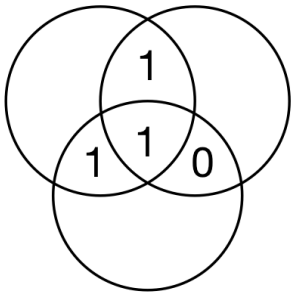
zaman aynı kodu üretip, ufak farklılıklarda ise büyük oranda farklı kodlar üretebilmektedirler. Genellikle veritabanlarındaki tablolarda aranan verileri hızlı bulmak, hızlı veri karşılaştırmaları, büyük boyutlardaki dosyaları karşılaştırmak için ve benzeri işlemlerde kullanılmaktadırlar.

2.4.6 Hamming Kod

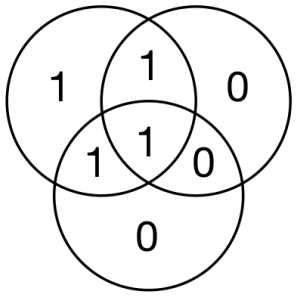
Richard Hamming tarafından bulunan, telekomünikasyonda çok kullanılan bir yöntemdir. Hamming kodun uygulandığı veri sonucunda 3 çeşit sonuç elde edilebilmektedir:

1. Hata algılanamadı.
2. Hata algılandı ve düzeltilebilir.
3. Hata algılandı fakat düzeltilemez.

Hamming (7,4)'ün çalışma şekli basitçe şu şekilde açıklanabilir. Elimizde 4 bitlik veri olsun ve bu verileri sırayla Şekil 2.14'teki gibi yerleştirelim. Daha sonra her daire içerisindeki boşluklara, çift sayıda 1 olacak biçimde Şekil 2.15'teki gibi 1 ya da 0 eklenir. Sonuç olarak 4 bitlik verimiz, 3 bitlik düzeltme bitleri ile 7 bitlik halde tutulur. Okunduğunda da bitler aynı şekilde geri yerlerine dizilerek, hatanın nerede olduğu tespit edilebilir ve duruma göre düzeltilebilmektedir.



Şekil 2.14: Veri bitleri



Şekil 2.15: Düzeltme bitlerinin eklenmiş hali

Bu işlemi mantık kapıları ile yapmak için ÖZEL VEYA kapısı kullanılmaktadır. Bu durumda 4 bitlik verimize d0, d1, d2, d3 dersek, eşlik bitlerimiz olan d4, d5 ve d6 bitleri Denklem 2.1'de görüldüğü şekilde hesaplanmaktadır.

$$\begin{aligned} d4 &= d0 \oplus d1 \oplus d2 \\ d5 &= d1 \oplus d2 \oplus d3 \\ d6 &= d2 \oplus d3 \oplus d0 \end{aligned} \tag{2.1}$$

Bu işlemden sonra d0, d1, d2, d3, d4, d5 ve d6 bitlerinden oluşan 7 bitlik sözcük olarak bellekte tutulur ve okunmasına ihtiyaç olduğu zaman hamming eşlik bitleri tekrar hesaplanmaktadır. Yeniden hesaplanan hamming bitlerine h4, h5, h6 dersek bu bitler Denklem 2.2'de görüldüğü gibi hesaplanmaktadır.

$$\begin{aligned} h4 &= d0 \oplus d1 \oplus d2 \\ h5 &= d1 \oplus d2 \oplus d3 \\ h6 &= d2 \oplus d3 \oplus d0 \end{aligned} \tag{2.2}$$

Bu işlemin ardından, yeni hesaplanan hamming eşlik bitleri ile bellekten okunmuş olan eski eşlik bitleri karşılaştırılır. Bu aşamada yine ÖZEL VEYA mantık kapısı kullanılmakta olup Denklem 2.3'te görüldüğü gibi yapılmaktadır.

$$
\begin{aligned}
c4 &= h4 \oplus h4 \\
c5 &= h5 \oplus d5 \\
c6 &= h6 \oplus d6
\end{aligned}
\tag{2.3}
$$

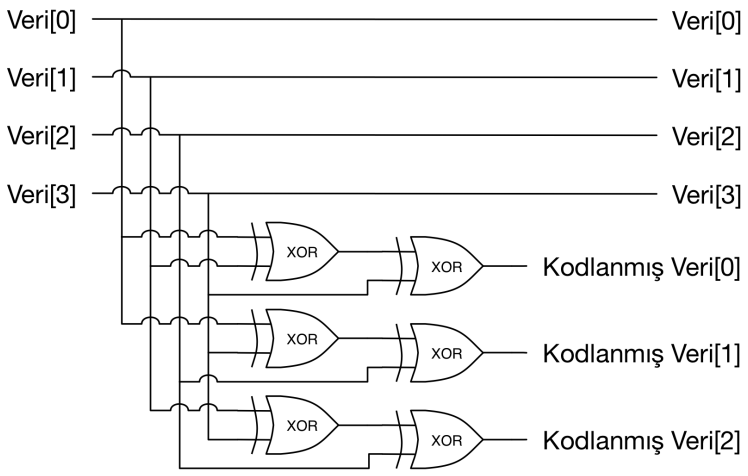
Son olarak c4, c5, c6 bitlerine göre hata olup olmadığı Tablo 2.4'teki gibi denetim yapılmakta ve hatalı bit varsa bu bit düzeltilmektedir.

Tablo 2.4: Hamming denetim bitlerine göre hatalı bit tespiti

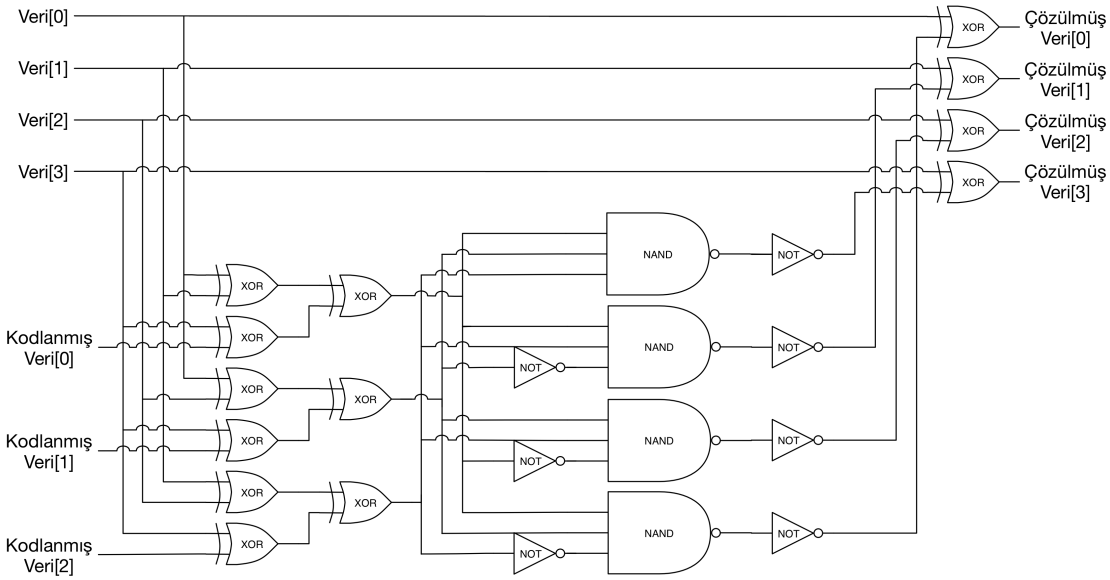
c4	**c5**	**c6**	**Hatalı Bit**
0	0	0	Hata yok
0	0	1	d6
0	1	0	d5
0	1	1	d3
1	0	0	d4
1	0	1	d0
1	1	0	d1
1	1	1	d2

Hamming kodda N adet düzelme biti var ise, bu bitler maksimum 2^N kadar bir durum belirtebilirler. Bu durumlardan biri hatasız olma durumu, 2k-1 durum ise hatalı olan bitleri ifade etmekte kullanılmaktadır. Bizim örneğimizde N=3 olarak alındığı için 2^3'ten 8 farklı durum söz konusu olmaktadır. Bu durumlardan biri hatasız durumu, geriye kalan 7'si ise hatanın olduğu biti göstermektedir. İki hata olduğu durumların anlaşılması için hata düzeltmeye ilave olarak 1 bit daha eklenir ve bu bit bütün bitler için eşlik biti olarak seçilir. Böyle bir tasarımda, iki bit hata olması durumunda ise sadece hata tespitinin yapılmasına izin vermektedir.

Bu kodlama mantıksal devre kapılarıyla yapılırsa Hamming (7,4) kodlayıcı devresi Şekil 2.16'daki gibi, kod çözücü devresi ise Şekil 2.17'deki gibi olacaktır.



Şekil 2.16: Hamming kodlayıcı



Şekil 2.17: Hamming kod çözücü

3. İLİŞKİLİ ÇALIŞMALAR

Geçmişten günümüze kadar, işlemciler üzerinde oluşabilen geçici hatalara karşı pek çok araştırma ve deneyler yapılmıştır. Bunlardan bazıları şu şekilde özetlenebilir:

Eşlik biti denetimi ve hata düzeltme kodu tasarımı, işlemcideki veri dizilerindeki geçici hataları farketmek ve düzeltmek için kullanılmaktadır [7]. Bu teknikler, hafızada saklanan verilerin kodlanması ile bulunan bilgi ve okuma aşamasında bu bilginin denetlenmesine dayanmaktadır. Eşlik ve hata düzeltme kodları, ön belleklerde sıkça kullanılmasına karşın, kodlama ve çözme aşamalarındaki fazla gecikmelerinden dolayı veri yolundaki bölümlerde kullanılamamaktadır. Örneğin, eşlik biti denetimi Intel'in 90nm titanyum işlemcisinde yazmaç öbeğini korumakta kullanılmış ve eşlik biti hesaplamak için fazladan çevrime ihtiyacı olmuştur [8].

Dar genişlikli değerler için hata düzeltme kodunda ve eşlik biti denetim devrelerinde, gecikmeyi azaltmak için bir yöntem önerilsede [9], bütün giriş değerleri düşünüldüğünde hata düzeltme kodu bilgisini hesaplama için gerekli gecikme cezası kaçınılmaz olmaktadır.

Birkaç araştırmacı, hem mimari seviyesinde [10] [11] [12] hemde kapı seviyesinde [13] işlemci boru hattındaki geçici hataların etkisini tespit etmeye çalışmıştır. Tek bit hata oranları (İngilizcesi: single error rate) üzerinde güç tasarufu tekniklerinin etkileri incelenmiştir [14].

Zamanda ve alanda tekrarlama, geçici hataları farketmek için bulunmuş en önemli tekniklerden biridir. Bu yöntemle, ya veri birden fazla hafıza alanına kopyalanır ve sonra okunurken basit bir oylama ile denetlenebilmekte [15] [16] [17] ya da değer tek bir kaynaktan bir kaç kez üretilerek denetlenebilmektedir [18].

Zamanda ve alanda tekrarlamaya örnek, hata tespiti ve düzeltmesi için çoklu iş parçacığı tekniği (İngilizcesi: multi threading) verilebilmektedir. Bu teknikte iş parçacıkları birbirinin kopyası olarak çalışmakta ve bir süre sonra bütün iş parçacıklarının sonuçları birbirleri ile karşılaştırılmaktadır. Böylelikle içlerinden

biri hataya maruz kalmış mı anlaşılmaktadır. Ardından gerekli düzeltme işlemi yapılarak tekrar işleme devam edilmektedir [19] [9] [20] [21] [22] [23].

İşlemci üzerinde, aynı anda çalışan seçilmiş iş parçacıklarının denetimi yapılırken performans düşmesini engellemek için bazı yöntemler bulunmaktadır. Bunlardan biri işlemcideki boştaki kaynakların kullanılmasıdır [24] [9] [25]. Ayrıca, kopyalanmış değerler için boştaki işlemci hafızasının kullanılmasıda önerilmiştir [26].

Buyruklar işlemci yapılarında yürütülürken fazla zaman harcamaktadırlar. Bu yüzden buyruklar üzerinde, hataya sebep olabilecek tanecik çarpması ihtimali fazla olmaktadır. Bu korunmasız durumu azaltmak için, işlemcinin uzun süre tıkandığı zamanlarda boru hattını boşaltma gibi bazı eski yöntemler bulunmaktadır [27]. Çoklu iş parçacığı olsa bile bir miktar performans azalması görülebilmektedir.

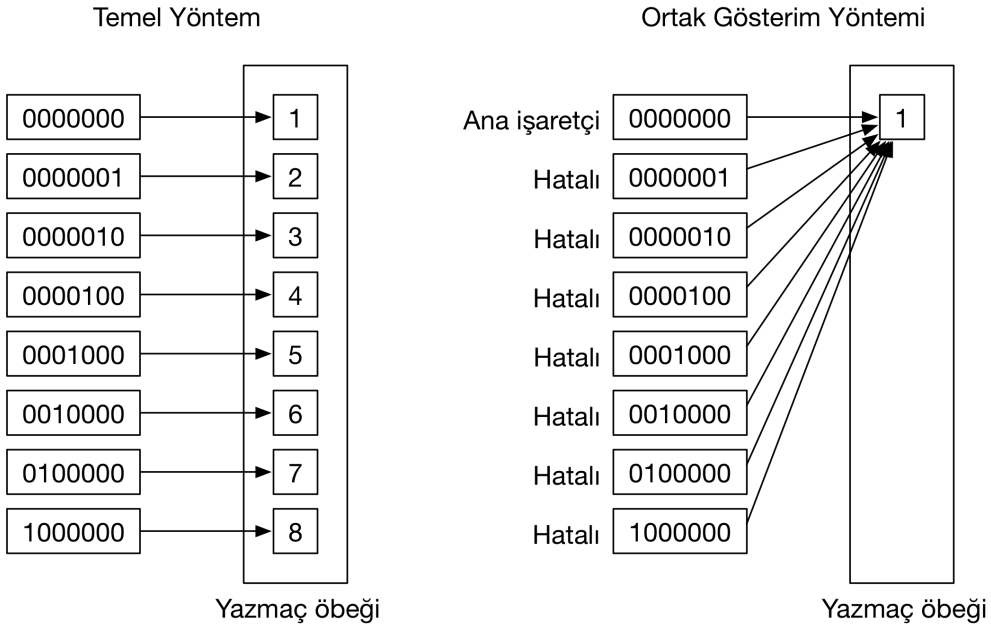
Literatürde, bunlardan başka, geçici hataları donanım seviyesinde belirtilerinden tespit etme [28], yazılım seviyesinde tespit etme [29] ve hem yazılım hem de donanımdan oluşmuş karma sistemlerde geçici hataların tespiti [30] gibi teknikler de bulunmaktadır.

Son zamanlarda hata düzeltme kodu bilgisini sanallaştırma yöntemi önerilmiştir [31]. Bu yöntem farklı kod algoritmalarını, donanımı modifiye etmeden kullanılabilmesine olanak sağlamaktadır. Sanallaştırılmış hata düzeltme kodu, sadece hata olduğunda verilerin denetlenmesine izin vermektedir. Son olarak, geçici olmayan hafızadaki güvenilirliği artırmak için adresleri yeniden haritalandırma yöntemi önerilmiştir [32].

4. ORTAK GÖSTERİM YÖNTEMİ

Bu tez çalışması, işlemcilerde yazmaç numaralarını saklayan alanları geçici hatalara karşı korumak için ortak gösterim yöntemini önermektedir. Ortak gösterim yöntemi, bütün tek bit hataları sıralı olarak düzeltmek yerine, ortak gösterim tekniğini kullanmaktadır.

Sıradan bir tasarımda bütün yazmaç numaraları farklı bir yazmacı gösterirken, ortak gösterim yönteminde birbirine hamming uzaklığı 1 olan bütün yazmaç numaraları, yazmaç numarası üzerinde bir bitlik hata oluşmuş olabilir diye başka bir yazmaç numarası yerine yine aynı yazmaç numarasını göstermektedirler. Bu yönteme göre işlemcide kullanılmak üzere ana yazmaç numaraları belirlenir ve işlemci hiçbir şekilde ana yazmaç numaralarının gösterdiği yerden hamming uzaklığı 1 olan yazmaç numaralarının gösterdiği yerlere atama yapmaz. 7 bitlik yazmaç numaraları için gösterimin nasıl olacağını Şekil 4.1'de açıkça gösterilmektedir.



Şekil 4.1: Ortak gösterim yöntemi

Ana yazmaç numaralarını seçilirken dikkat edilmesi gereken iki kural bulunmaktadır. Bunlar:

1. İki ana yazmaç numarası birbirinden 3 hamming uzaklıkta olmalıdır.
2. Hamming uzaklığı 1 olan yazmaç numaraları, sadece kendi ana yazmaç numaralarını göstermeli, başka ana yazmaç numaralarını da göstermemelidir.

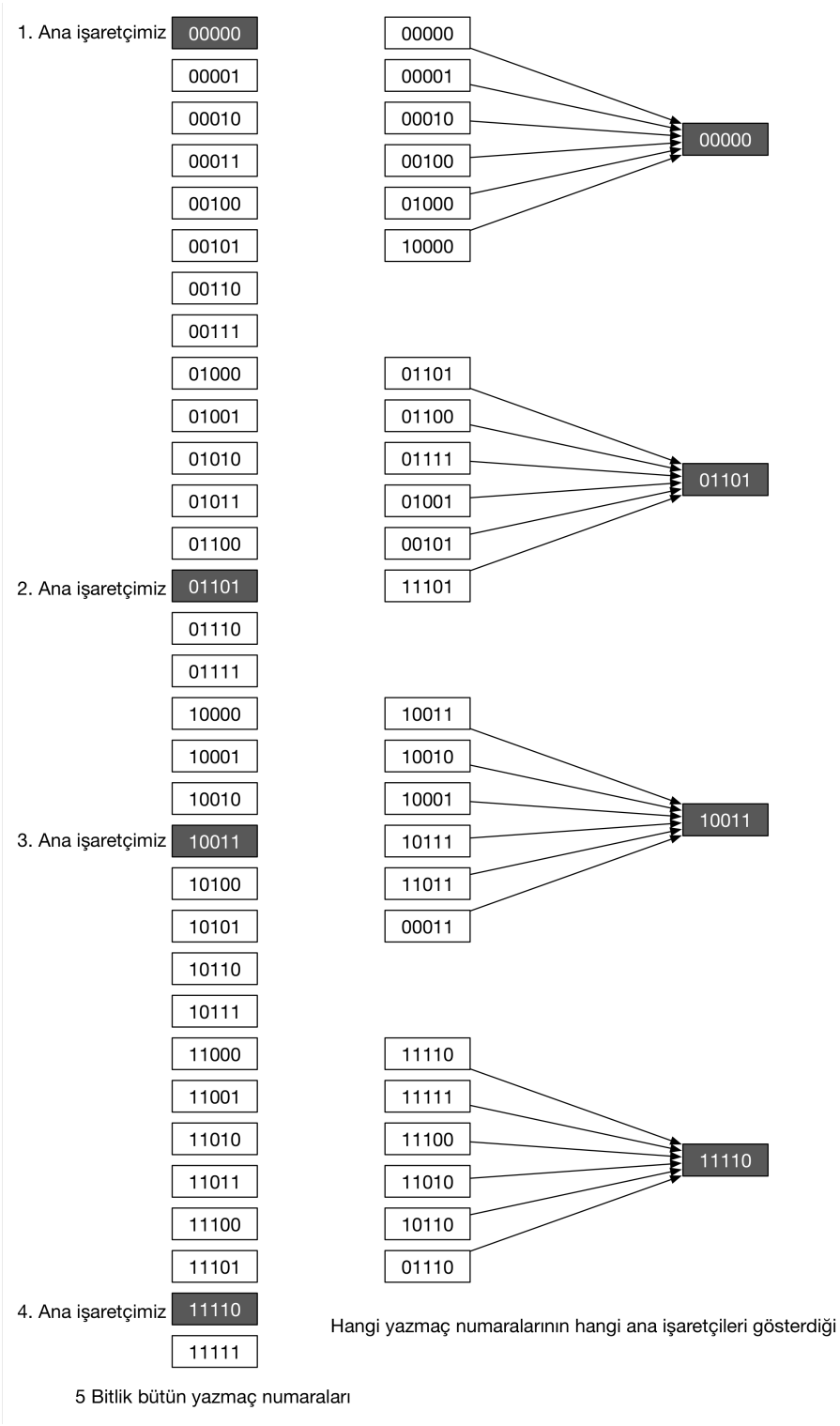
Ortak gösterim yöntemi, çözülmesi gereken üç problemi açığa çıkarmaktadır:

1. Aynı yeri birden fazla yazmaç numarasının göstermesi, saklama alanının verimsiz kullanılmasına sebep olmakta ve uzun yazmaç numaralarının kullanılmasını gerektirmektedir.
2. Saklama kısmında modifiye edilmiş yazmaç numaralarının, aynı alanı göstermesi için çözücü bir bloğa ihtiyaç vardır.
3. Yazmaç numarasının değerleri olarak artık modifiye edilmiş yeni ana yazmaç numaraları kullanılmalıdır.

4.1 Geniş Yazmaç Numaraları Kullanmak İçin Neye İhtiyaç Var?

Ortak gösterim yönteminde, her bir yazmaç numarasında oluşabilecek 1 bitlik hataların hepsini etkili olarak düzeltebilmek için geniş yazmaç numaralarına ihtiyaç vardır. Bu fazla alan gereksinimi, belleğe bir şey yazarken ve okurken hata düzeltme tekniği olarak kullanılan teklik çiftlik durumlarını (İngilizcesi: parity) karşılaştıran, ÖZEL VEYA kapılarından oluşan geniş ağaç yapılarında da benzerlik göstermektedir. Aslında, ortak gösterim yöntemindeki bu tek biti düzeltmek için gereken ek yük, Hamming kodu ile tek bitlik hatayı düzeltebilmek için gereken ek yük ile aynı miktardadır.

Ortak gösterim yöntemi kurallarına uygun olarak seçilmiş 5 bitlik ana işaretçiler ve hangi yazmaç numaralarının bu işaretçileri gösterdiğini Şekil 4.2'de gösterilmiştir. Şekilde de görüldüğü üzere, her bir ana işaretçiyi, kendisi ve kendisi ile hamming uzaklığı 1 olan diğer yazmaç numaraları, yani kendisinin bir bitinde hata oluşmuş versiyonu göstermektedir.



Şekil 4.2: 5 Bitlik ana işaretçi seçimi ve hangi numaraların hangi ana işaretçileri gösterdiği

Ortak gösterim yönteminde, ne kadar uzunluktaki yazmaç numarası için, ne kadar yazmaç kullanabileceğimizi bulmak için deneme yanıma yöntemi kullanılarak hazırlanmış olan, ana işaretçi bulma yazılımı ile Tablo 4.1'deki veriler elde edilmiştir.

Tablo 4.1: Bit sayıları ve gösterebildiği yazmaç sayıları

Bit sayısı	**Normalde gösterebildiği yazmaç sayısı**	**Ortak gösterim yöntemi ile gösterebildiği yazmaç sayısı**
1	2	1
2	4	1
3	8	2
4	16	2
5	32	4
6	64	8
7	128	16
8	256	16
9	512	32
10	1024	64
11	2048	128
12	4096	256
13	8192	512
14	16384	1024
15	32768	2048
16	65536	2048

Tablo 4.1'de açığa çıkan sonuçlara göre, Denklem 4.1'de gösterildiği şekilde formüle dökülebilmektedir.

$$Adreslenebilen\ yazmaç\ sayısı = 2^{n-[log_2 n]-1} \tag{4.1}$$

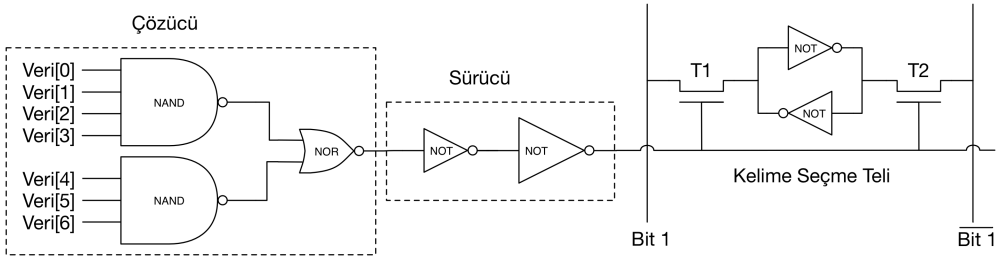
Hamming kodunda eşlik bitlerine "s" dersek kodlanmış kelimenin uzunluğu $2^s - 1$ olarak bulunmaktadır [6]. Aslında bu tasarım, Hamming kodunun farklı bir biçimde uygulanmış halidir. Hamming'te 1 bitlik hatayı düzelten bir grup eşlik biti yerine, bu tasarım bütün bir bitlik hatalı değerlerin aynı yazmacı göstermesi ile yapılmıştır. Bu yüzden, Hamming kodunda olduğu gibi, ortak gösterim tasarımında da ekstra devre yükü, bit sayısının artmasıyla azalmaktadır.

4.2 Çözücü Blok Yapısı

Adres bitlerinin evrilmiş versiyonlarının işlemci içerisinde hali hazırda var olduğunu varsayarsak eğer, sıradan durağan rastgele erişimli bellek (İngilizcesi: SRAM) bit hücresi dizisindeki tek bir girişin çözücü bloğu mantıksal olarak tek bir VE kapısı ile gerçeklenebilmektedir. Fakat, bu durum bit sayısına göre, giriş kapısı (İngilizcesi: fan-in) fazla olan bir VE kapısı gerektirmektedir.

Fiziksel olarak fazla giriş kapısı olan mantıksal kapılar, az olanlara göre daha yavaş çalışmaya eğilimlidirler. Yavaşlama eğilimi, kapı içerisindeki devrenin karmaşıklığının artması ve giriş sığasının artmasıyla ilgili olarak oluşmaktadır. Bu yüzden gerçekte fazla girişli VE kapıları kullanışsızdır. Bu sorunun çözümü olarak, VEYA kapısı ile iki ya da daha fazla VE kapılarının çıkışları birleştirilebilmektedir.

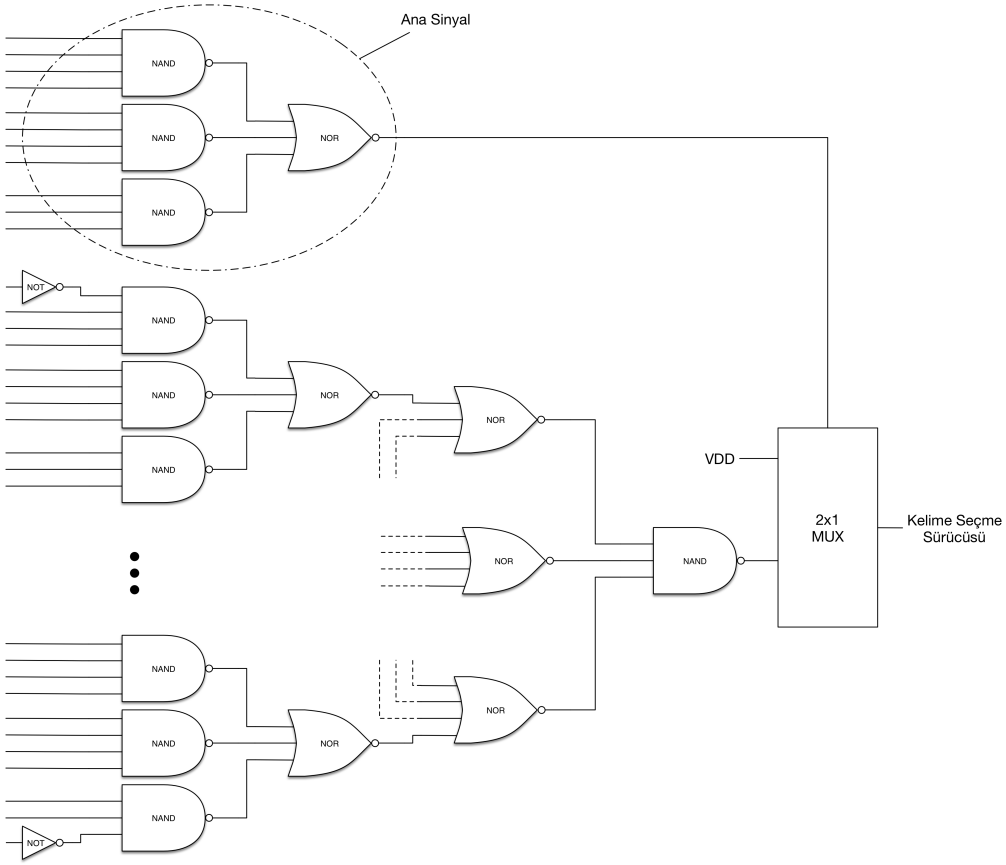
Temel çözücü devresinin blok diyagramı Şekil 4.3'te görüldüğü gibidir. Burada gösterilmiş çözücü bloğun çıkışı bütün bir satırda kelime seçme sürücüsünü aktif hale getirir ve böylelikle istediğimiz satıra erişmiş oluruz.



Şekil 4.3: Temel durağan rastgele erişimli bellek bit hücresi ve çözücü devresi

Önerilen ortak gösterim tasarımı, n bitli bir etiket kullanıldığı durum için, her bir giriş için Şekil 4.3'teki çözücü bloktan n+1 tanesinin paralel olarak kullanılmasını ve çıkışlarının VEYA kapısından geçirilmesini zorunlu kılmaktadır.

Gecikmedeki aşırı yükü azaltmak için Şekil 4.4'teki devre önerilmiştir. Bu devre, ana işaret değerlerinin çözümlenmesinin, hatalı olanlardan birine göre daha hızlı olmasına olanak vermektedir. Eğer ana sinyalin çözümlemesi 1 ise, seçicinin çıkışına Vdd verilmektedir. Bu da bütün bir çözücünün çıkışının 1 olduğu anlamına gelmektedir. Bu durumda çözücü bloğun gecikmesi ana sinyal için sadece seçicinin geçikmesi kadar olmaktadır. Seçiciye giren ikinci giriş gözlemlenerek hata anlaşılabilmekte ve sistem hata algılama mekanizması olarak kullanılabilmektedir.



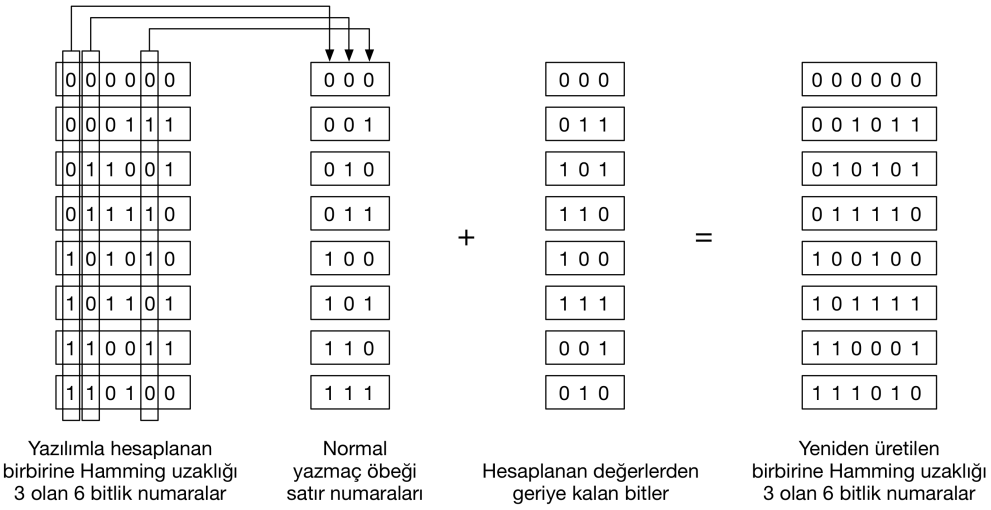
Şekil 4.4: Önerilen ortak gösterim yöntemi çözücü devresi

4.3 İşaretçilerin Yerleştirilme Mantığı

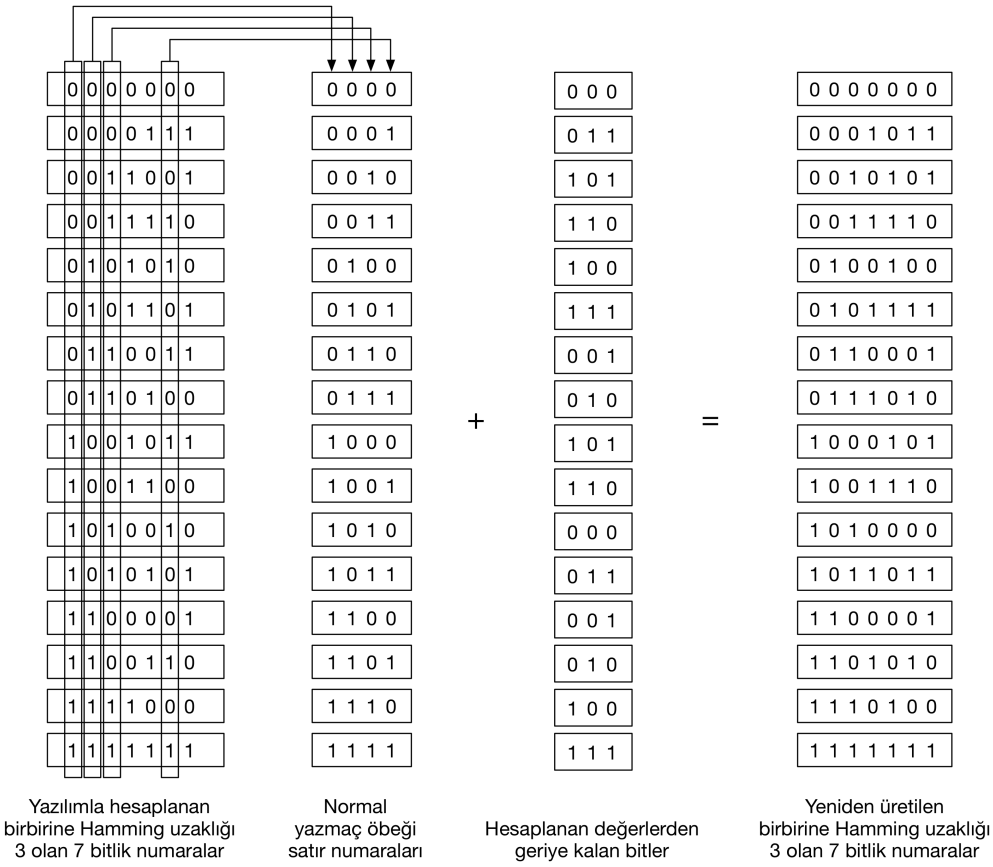
Ortak gösterim tasarımında, işaretçi değerlerini atanması mantığının modifiye edilmesine ihtiyaç vardır. Örneğin, boşta kalan bit dizileri ve yerleştirilmemiş yazmaç numaralarının donanımda toprağa çekilerek, yazmaçlardaki yerleşim mantığı ortak gösterim tasarımına göre düzenlenebilir. Bu yolla, yazmaç etiketlerindeki artan bit sayısı dışında, işlemcinin ön yüzünde değişen bir şey olmayacaktır. Fakat bu tasarım, yazmaç numaralarının çözümlenmesi sırasında fazladan gereksiz alan harcayacağından tavsiye edilmemektedir.

Aslında bu iş için en basit yöntem, normal bir işlemcide yazmaç numaralarının bit sayılarını artırmak ve hamming uzaklığı 1 olan yazmaç numaralarının da bu işaretçileri göstermelerini sağlamak olacaktır. Bu sayede bit sayıları dışında ve eklenecek çözücü devreler dışında tasarımda başka bir şey değişmemektedir.

Bu işlem için ilk olarak ana işaretçi bulan yazılım ile birbirine Hamming uzaklığı en az 3 olan numaralar bulunur. Ardından bu numaralardan sıralı bir liste çıkartılmak için bazı bitleri seçilir. Hamming uzaklığın mantığına göre satırların yerleri karıştırılmadan, sütunların yerleri değiştirilirse, numaraların birbirlerine olan uzaklığı değişmeyecektir. Bu yolla yazmaçlardaki sıranın bir kopyasını elde edip uzatılmış olan bitlere ise geriye kalan bitler ilave edilerek yeni yazmaç numaraları oluşturulur. Böylelikle yazmaç tarafında bitlerin uzatılması dışında hiçbir şey değişmemiş olacaktır. Bahsedilen yöntem Şekil 4.5'te 6 bitlik ve Şekil 4.6'da ise 7 bitlik ana işaretçiler kullanılarak oluşturulan tasarımlarda gösterilmiştir.



Şekil 4.5: 6 Bitlik ana işaretçi kullanılarak düşünülen tasarım



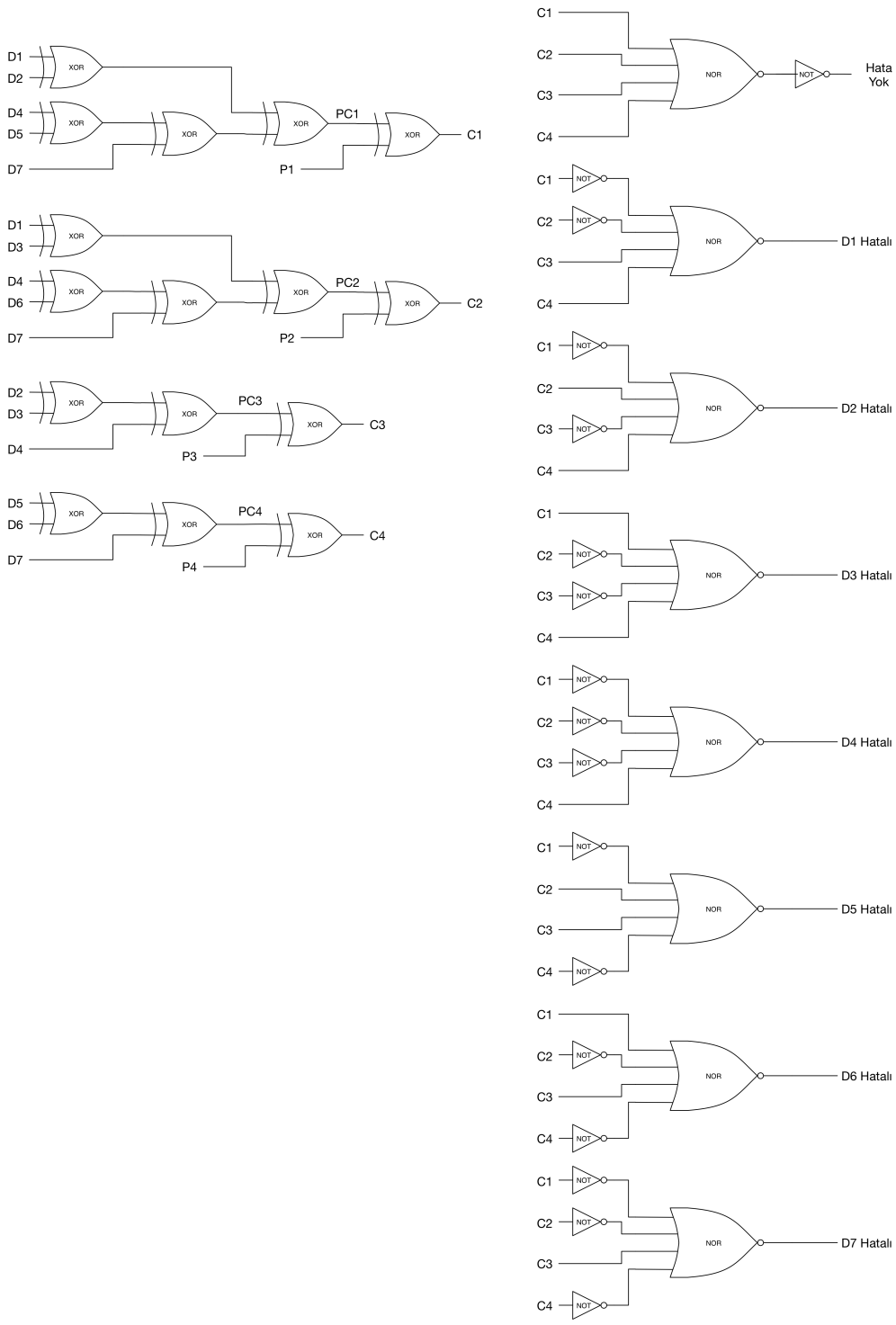
Şekil 4.6: 7 Bitlik ana işaretçi kullanılarak düşünülen tasarım

Tabi işlemcinin çalıştıracağı programdaki, buyruklardaki yazmaç numaralarının da değişmesi gerektiğini dikkate almıyoruz. Basit bir çevirici yazılımı kullanılarak bu işlem rahatlıkla yapılabilir.

5. DENEYLER VE SONUÇLAR

Bu tez çalışmasında bahsedilen ortak gösterim tasarımı, maliyet oranları bakımından geçici hataları düzeltmede sık kullanılan hata düzeltme kodu ile karşılaştırılmıştır. Bunun için Cadence Dizayn Araçları'ndan Virtuoso kullanılmıştır. 90nm CMOS UMC teknoloji kütüphanesi kullanılarak, her iki tasarımda devre seviyesinde yapılarak uygulanmıştır. Vdd gerilimi 1 volt alınarak analiz işlemleri yapılmış ve sonuçlar elde edilmiştir.

Yazmaç numarası uzunluğu 7 bit olan hata düzeltme kodunun kapı seviyesinde uygulanmış hali Şekil 5.1'de görülmektedir. Eşlik bitleri olan P1, P2, P3 ve P4 bitleri yazma işleminden önce PC1, PC2, PC3 ve PC4'ün hesaplandığı gibi hesaplanarak, gerçek veri ile birlikte bellekte tutulmaktadır. Okuma yapıldığında bellekteki verilerden eşlik bitleri (PC1, PC2, PC3 ve PC4) tekrar hesaplanmaktadır. Ardından okuma ve yazma işlemleri arasında herhangi bir hataya maruz kalınıp kalınmadığını tespit etmek için, önceden hesaplanmış olan eşlik bitleri ile karşılaştırılmaktadır. Karşılaştırma işlemi sonucu yani C1, C2, C3 ve C4 bitleri, eğer bitlerde bir hata yoksa 0 olmalıdır. Hatanın olup olmadığının tespiti ve varsa hangi veri bitinde hata olduğunun tespit edilmesi için Şekil 5.1'in sağ tarafında görülmekte olan devreler kullanılmaktadır. Hata olmaması durumunda ise Şekil 5.1'in sağ taraftaki en üstteki devreden 1 elde edilmektedir. Son olarak hata olması durumunda, hatalı veriler ile elde edilen sonuçlar ÖZEL VEYA kapılarından geçirilerek bu hatalı bit düzeltilmektedir.



Şekil 5.1: Hata düzeltme kodunun devresi

Hata düzeltme kodu tasarımı, Şekil 5.1'de de görüldüğü üzere yazmaçtaki tek bit hatayı düzeltebilmek için okuma ve yazma aşamalarında birkaç ÖZEL VEYA işleminden geçmesi gerekmektedir.

Bu tez çalışması kapsamında, hata düzeltme kodu tasarımı ile tek bir yazmaç numarasını korumak üzere tasarlanmış olan ortak gösterim tasarımının karşılaştırılması yapılmış ve Tablo 5.1'deki sonuçlar elde edilmiştir. Alınan sonuçlara göre, iki devre neredeyse aynı büyüklüğe sahiptir. Ortak gösterim tasarımı, harcadığı enerji ve geçikme sonuçları bakımından ÖZEL VEYA kapıları demetinden oluşan hata düzeltme kodu tasarımına göre daha iyidir. Fakat tablodaki ortak gösterim tasarımı sonuçları, tek bir kelime seçme sürücüsüne bağlanan devre için olduğundan, alan ve enerji masrafları yazmaç öbeğinin satır sayısı miktarınca doğru orantılı olarak artacaktır.

Tablo 5.1: Ortak gösterim yöntemi ile hata düzeltme kodunun karşılaştırılması

	Alan (μm^2)	Enerji (fJ)	Gecikme (ps)	
			Hata Var	Hata Yok
Ortak gösterim	444,402	109,2	176,36	104,5
Hata düzeltme kodu	469,238	387,9	358,787	

Ortak gösterim tasarımı, hata düzeltme kodu ile karşılaştırıldığında, en önemli avantajı düşük gecikme masrafının olmasıdır. Bu yüzden ortak gösterim tasarımı, yazmaçlarda tek bitlik hata düzeltme kodunun gecikmesinin problem olduğu yerlerde, tek bit hataları düzeltmeyi mümkün kılabilmektedir.

Ayrıca, önerilen tasarım işlemci boru hattının farklı aşamalarına hata düzeltme devrelerinin gecikmesini dağıtmaktadır. Örneğin, yazmaç numaralarını korumak için kullanılması durumunda, gecikme işlemcinin ön ucunda (İngilizcesi: frontend), buyruk belleği ve yazmaç öbeği üzerine dağıtılmıştır.

Bu sonuçlar dikkate alındığında ÖZEL VEYA ağaç yapısı ile oluşturulmuş olan hata düzeltme kodu tasarımının, fazla gecikmesinden dolayı, işlemci için zamanlamanın kritik olduğu yazmaç öbeği ve buyruk kuyruğu gibi yerlerde kullanılması mantıklı değildir. Önerdiğimiz ortak gösterim tasarımı toplamda daha fazla alan kaplamasına karşı, diğer tasarıma göre çok daha az geçikmeye sahiptir.

Kaynaklar

[1] Gordon, E.M., Cramming more components onto integrated circuits, Electronics, 38(8), 82 - 85, 1965

[2] "Von Neumann Architecture"
erişim adresi: http://en.wikipedia.org/wiki/Von_Neumann_architecture, erişim tarihi: 01 Nisan 2015.

[3] Hinton, G., Sager, D., Upton, M., Boggs, D., Carmean, D., Kyker, A., Roussel, P., The microarchitecture of the Pentium 4 processor, Intel Technology Journal, 1, 1 - 13, 2001.

[4] "IBM Experiments in Soft-Fails in Computer Electronics (1978-1994)"
erişim adresi: http://www.research.ibm.com/journal/rd/401/curtis.html, erişim tarihi: 01 Nisan 2015.

[5] Robert, C.B., Radiation-induced soft errors in advanced semiconductor technologies, IEEE Transactions on Device and Materials Reliability, 5(3), 2005

[6] Micheloni, R., Marelli, A., Ravasio, R., Error Correction Codes for Non-Volatile Memories, Springer, 2008, ISBN: 978-1-4020-8390-7.

[7] Phelan, R., Addressing soft errors in ARM core-based designs, White Paper, ARM, 2003.

[8] Fetzer, E., Dahle, D., Little, C., Safford, K., The parity protected, multithreaded register files on the 90-nm itanium microprocessor, Solid-State Circuits, IEEE Journal of, 41(1), 246 - 255, 2006.

[9] Koçberber, Y.O., Osmanlioglu Y., Ergin, O., Exploiting narrow values for faster parity generation, Microelectronics International, 26(3), 22 - 29, 2009.

[10] Sridharan, V., Asadi, H., Tahoori, M., Kaeli, D., Reducing cache susceptibility to soft errors, IEEE Transactions on Dependable and Secure Computing, 3(4), 353 - 364, 2006.

[11] Li, X., Adve, S., Bose, P., Rivers, J., SoftArch: an architecture-level tool for modeling and analyzing soft errors, International Conference on Dependable Systems and Networks, 496 - 505, 2005.

[12] Wang, N., Quek, J., Rafacz, T., Patel, S., Characterizing the effects of transient faults on a high-performance processor pipeline, International Conference on Dependable Systems and Networks, 61 - 70, 2004.

[13] Constantinides, K. ve diğerleri, Assessing SEU vulnerability, Workshop on Architectural Reliability, 2005.

[14] Lin, L., Degalahal, V., Vijaykrishnan, N., Kandemir, M., Irwin, M., Soft error and energy consumption interactions: a data cache perspective, International Symposium on Low Power Electronics and Design, 132 - 137, 2004.

[15] Ergin, O., Unsal, O., Vera, X., Gonzalez, A., Reducing soft errors through operand width aware policies, IEEE Transactions on Dependable and Secure Computing, 6(3), 217 - 230, 2009.

[16] Hu, J., Wang, S., Ziavras, S., In-register duplication: exploiting narrow-width value for improving register file reliability, International Conference on Dependable Systems and Networks, 281 - 290, 2006.

[17] Memik, G., Kandemir, M., Ozturk, O., Increasing register file immunity to transient errors, Design, Automation and Test in Europe, 1, 586 - 591, 2005.

[18] Qureshi, M., Mutlu, O., Patt, Y., Microarchitecture-based introspection: a technique for transient-fault tolerance in microprocessors, International Conference on Dependable Systems and Networks, 434 - 443, 2005.

[19] Gomaa, M., Scarbrough, C., Vijaykumar, T., Pomeranz, I., Transient-fault recovery for chip multiprocessors, 30th Annual International Symposium on Computer Architecture, 98 - 109, 2003.

[20] Mukherjee, S., Kontz, M., Reinhardt, S., Detailed design and evaluation of redundant multi-threading alternatives, 29th Annual International Symposium on Computer Architecture, 99 - 110, 2002.

[21] Reinhardt, S.K., Mukherjee, S.S., Transient fault detection via simultaneous multithreading, 27th Annual International Symposium on Computer Architecture, 25 - 36, 2000.

[22] Rotenberg, E., AR-SMT: a microarchitectural approach to fault tolerance in microprocessors, 29th Annual International Symposium on Fault-Tolerant Computing, 84 - 91, 1999.

[23] Vijaykumar, T. ve diğerleri, Transient-fault recovery using simultaneous multithreading, 29th Annual International Symposium on Computer Architecture, 87 - 98, 2002.

[24] Gomaa, M., Vijaykumar, T., Opportunistic transient-fault detection, 32th Annual International Symposium on Computer Architecture, 172 - 183, 2005.

[25] Smolens, J.C., Kim, J., Hoe, J.C., Falsafi, B., Efficient resource sharing in concurrent error detecting superscalar microarchitectures, 37th Annual IEEE/ACM International Symposium on Microarchitecture, 2004.

[26] Zhang, W. ve diğerleri, ICR: in-cache replication for enhancing data cache reliability, International Conferance on Dependable Systems and Networks, 291 - 300, 2003.

[27] Weaver, C., Emer, J., Mukherjee, S., Reinhardt, S., Techniques to reduce the soft error rate of a high-performance microprocessor, 31st Annual International Symposium on Computer Architecture, 264 - 275, 2004.

[28] Wang, N., Wang, N., ReStore: symptom-based soft error detection in microprocessors, IEEE Transactions on Dependable and Secure Computing, 3(3), 188 - 201, 2006.

[29] Reis, G.A. ve diğerleri, SWIFT: software implemented fault tolerance, International Symposium on Code Generation and Optimization, 234 - 254, 2005.

[30] Reis, G., Chang, J., Vachharajani, N., Mukherjee, S., Rangan, R., August, D., Design and evaluation of hybrid fault-detection systems, 32nd Annual International Symposium on Computer Architecture, 148 - 159, 2005.

[31] Yoon, D., Erez, M., Virtualized and flexible ECC for main memory, ACM SIGARCH Computer Architecture News - ASPLOS, 38(1), 397 - 408, 2010.

[32] Yoon, D.H., Muralimanohar, N., Chang, J., Ranganathan, P., Jouppi, N., Erez, M., FREE-p: protecting non-volatile memory against both hard and soft errors, 17th Annual International Symposium on High Performance Computer Architecture, 466 - 477, 2011.

ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, Adı	: İşlek, Emrah
Uyruğu	: T.C.
Doğum tarihi ve yeri	: 25.03.1988 Ankara
Medeni hali	: Bekar
E-mail	: emrahislek@gmail.com

Eğitim

Derece	**Eğitim Birimi**	**Mezuniyet Tarihi**
Y. Lisans	TOBB Ekonomi ve Teknoloji Üniversitesi	2015
Lisans	TOBB Ekonomi ve Teknoloji Üniversitesi	2012

İş Deneyimi

Yıl	**Yer**	**Görev**
2015-	RST Teknoloji	Sayısal Tasarım Mühendisi
2013-2014	TOBB Ekonomi ve Teknoloji Üniversitesi	Burslu Yüksek Lisans Öğrencisi
2012-2014	TOBB Ekonomi ve Teknoloji Üniversitesi	Proje Asistanı

Yabancı Dil

İngilizce
İspanyolca

Yayınlar

Exploiting a Fast and Simple ECC for Scaling Supply Voltage in Level-1 Caches
Gülay Yalçın, Emrah İşlek, Öykü Tozlu, Pedro Reviriego, Adrian Cristal, Osman S. Ünsal, Oğuz Ergin
On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International

Exploiting Existing Comparators for Fine-Grained Low-Cost Error Detection
Gülay Yalçın, Oğuz Ergin, Emrah İşlek, Osman S. Ünsal, Adrián Cristal
ACM Transactions on Architecture and Code Optimization (ACM TACO) - Oct 2014