İŞLEMCİ YAPILARININ HATALARA KARŞI HASSASİYETİNİ KARŞILAŞTIRMAK İÇİN YENİ BİR BİT ETKİ KATSAYISI TANIMLANMASI VE KULLANILMASI

SERDAR ZAFER CAN

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ

TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

AĞUSTOS 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

Serdar Zafer CAN tarafından hazırlanan İŞLEMCİ YAPILARININ HATALARA KARŞI HASSASİYETİNİ KARŞILAŞTIRMAK İÇİN YENİ BİR BİT ETKİ KATSAYISI TANIMLANMASI VE KULLANILMASI adlı bu tezin Yüksek Lisans tezi olarak uygun olduğunu onaylarım.

_____

Doç. Dr. Oğuz ERGİN

Tez Danışmanı

Tez Jüri Üyeleri

Başkan :Doç. Dr. Özcan ÖZTÜRK _____

Üye : Doç. Dr. Ali Bozbey _____

Üye : Doç. Dr. Oğuz ERGİN _____

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.

Serdar Zafer CAN

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 – Ağustos 2015**

SERDAR ZAFER CAN

İŞLEMCİ YAPILARININ HATALARA KARŞI HASSASİYETİNİ KARŞILAŞTIRMAK İÇİN YENİ BİR BİT ETKİ KATSAYISI TANIMLANMASI VE KULLANILMASI

ÖZET

Üretim teknolojisinin ilerlemesi ile birlikte mikroişlemcilerde kullanılan transistörlerin boyutları da küçülmektedir. Bu küçülme sayesinde hem çalışma gerilimleri daha düşük seviyelere çekilebilmekte, hem de tümleşik devrelere daha fazla sayıda transistör yerleştirilerek, mikroişlemcilerin işlem ve hafıza kapasiteleri artırılabilmektedir. Ancak transistör boyutlarındaki bu küçülme, mikroişlemcilerin geçici hatalara karşı hassasiyetini arttırmıştır. Günümüzde işlemci mimarlarının oluşturdukları yapıların hassasiyetlerini tasarım aşamasında ölçebilmeleri için en çok kullandıkları katsayı Mimari Hassasiyet Katsayısıdır (MHK). Mimarlar bu katsayıya bakarak sistemlerinin güvenilirliği konusunda üretimden önce fikir edinirler.

MHK sistemdeki bitleri hataya karşı hassas veya dayanıklı gibi iki farklı kategoride inceler. Ancak hassas bir bitteki değişimin sistemde ne kadar etki yarattığı konusunda herhangi bir bilgi vermez. Bu tezde, TÜBİTAK'ın 112E004 numaralı "Geçici Hatalara Karşı Dayanıklı Mikroişlemciler" projesi dâhilinde MHK da kullanılarak bir bitte oluşabilecek hatanın sisteme ne kadar etki ettiğini belirlemek amacıyla yeni bir katsayı tanımı yapılmıştır ve analiz sonuçları diğer katsayılarla karşılaştırılmıştır.

Anahtar Kelimeler: Mikroişlemciler, geçici hatalar, hataya karşı dayanıklılık.

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

SERDAR ZAFER CAN

BIT IMPACT FACTOR: TOWARDS MAKING FAIR VULNERABILITY COMPARISON

ABSTRACT

Transistors used in microprocessors are shrinking with the advancements in the manufacturing technology. With the help of this scaling on the transistors, operating voltages can be lowered and both computational power and memory capacity can be increased by placing transistors denser. However, smaller transistors lead growth of the soft error vulnerability of the microprocessors. Today, architects generally use Architectural Vulnerability Factor (AVF) as vulnerability metric for their products at the design time. It gives them an idea about how much reliable their system is.

AVF considers that the value of a bit is either required for Architecturally Correct Execution (ACE-bit) or not (unACE-bit). Therefore, AVF cannot distinguish the vulnerability impact level of an Architecturally Correct Execution - ACE bit. In this thesis, a new metric is introduced by extending AVF to provide more accurate vulnerability analysis of the bits in the system and it is compared with the other metrics. This thesis was supported by a TUBITAK project titled "Reliable Microprocessor Design" under grant number 112E004.

Keywords: Microprocessors, soft errors, vulnerability, bit impact.

TEŞEKKÜR

Bu çalışmayı tamamlamamda ve yüksek lisans eğitimim boyunca emeği geçen değerli danışmanım Doç. Dr. Oğuz Ergin'e; dâhil olduğum projeyi destekleyen TÜBİTAK'a; tez konumun makale olmasında büyük katkıları bulunan Gülay Yalçın'a; çalışma arkadaşlarım Emrah İşlek ve Mustafa Çavuş başta olmak üzere tüm Kasırga Ailesine; her türlü imkânı ve eğitim bursumu sağlayan TOBB ETÜ Mühendislik Fakültesi ve Fen Bilimleri Enstitüsü'ne ve son olarak da her zaman beni karşılıksız destekleyen aileme teşekkür ederim.

İÇİNDEKİLER

ÖZET ..... **iv**

ABSTRACT ..... **v**

TEŞEKKÜR ..... **vi**

ŞEKİL LİSTESİ ..... **ix**

TABLO LİSTESİ ..... **xi**

1 GİRİŞ **1**

2 TEMEL KONULAR **4**

2.1 Mikroişlemci Mimarisi 4

2.1.1 Boru Hattı Yapısı 6

2.1.2 Sırasız Yürütüm 9

2.1.2.1 Yeniden Adlandırma 10

2.1.2.2 Yeniden Sıralama Arabelleği 13

2.1.2.3 Rezervasyon İstasyonu 14

2.2 Mimari Hassasiyet Katsayısı 16

2.3 Hamming Uzaklığı 16

3 İLİŞKİLİ ÇALIŞMALAR **17**

4 ÖNERİLEN YÖNTEM **20**

4.1 Bitlerin Sınıflandırılması 20
4.1.1 Sıradan Veri 23
4.1.2 İşlem Kodu 23
4.1.3 Kaynak Belirteçleri 24
4.1.4 Sonuç Belirteçleri 25
4.1.5 Kontrol Bilgileri 26
4.1.6 Sistem Bitleri 27
4.2 Bit Etki Katsayısı (BEK) 28
4.3 MHK ile BEK Arasındaki İlişki 29
5 DENEYSEL SONUÇLAR 31
6 SONUÇ 36
KAYNAKLAR 37
ÖZGEÇMİŞ 41

ŞEKİL LİSTESİ

2.1 Bir bilgisayarın beş temel bileşeni [6]. 4

2.2 MIPS işlemcisinin beş aşamalı boru hattı tasarımı 7

2.3 Beş aşamalı MIPS işlemcisinde boru hattı kullanıldığında ve kullanılmadığında buyrukların zaman içinde geçtiği aşamalar 7

2.4 Sırasız yürütüme sahip işlemcinin aşamaları [7]. 10

2.5 Veri sorunları [7]. 11

2.6 Yeniden sıralama arabelleğine veri girişleri ve çıkışları [7]. 13

2.7 Örnek bir rezervasyon istasyonu girdisi [7]. 14

2.8 Sırasız yürütüme sahip bir işlemcinin boru hattı yapısı. 15

4.1 Örnek bir MDDY buyruk. Gri alanlar o anda MDDY değildir. 21

4.2 Bir işlem biriminin giriş ve çıkışları 22

4.3 Sıradan bir verideki hatanın etkisi 23

4.4 Kaynak yazmacı belirtecindeki bir hatanın etkisi 24

4.5 Sonuç yazmacı belirtecindeki bir hatanın etkisi 26

5.1 Sınama programlarının yazmaç dosyası için ortalama Hamming uzaklık değerleri 33

5.2 Sınama programlarının işlem kodları için ortalama Hamming uzaklık değerleri 33

5.3 Sınama programlarının kaynak yazmacı belirteçleri için ortalama Hamming uzaklık değerleri 34

5.4 Sınama programlarının sonuç yazmacı belirteçleri için ortalama Hamming uzaklık değerleri 34

5.5 Sınama programlarının farklı tipteki veriler için ortalama MHK, $MHK_{ağırlıklı}$ ve hata verme yöntemi ile elde edilen hassasiyet seviyeleri 35

TABLO LİSTESİ

1.1 Intel Pentium 4 ve Intel i7 - 4790 işlemcilerinin karşılaştırması. 1

2.1 Örnek bir yazmaç yeniden adlandırma. $0 yazmacının değeri sıfırdır 12

2.2 Hamming uzaklık örnekleri 16

5.1 Benzetimci ortamının özellikleri.....31

1. GİRİŞ

Günümüzde, çeşitli ışımalar sonucunda açığa çıkan yüklü parçacıkların sayısal sistemlere çarpması sonucunda, devrelerde bulunan bitlerin değerleri değişerek hatalar oluşmaktadır [1] [2]. Bu parçacık çarpmaları kalıcı sorunlara neden olmaz ancak sistemin çökmesine kadar varabilecek çeşitli sorunlar oluşturabilir. Bu tarz kalıcı olmayan hatalara literatürde geçici hata (İngilizce: soft error) ismi verilmiştir.

Üretim teknolojisinin ilerlemesiyle birlikte her yeni nesilde daha küçük boyutlarda transistörler kullanılmaktadır. Tablo Tablo *1.1*'de Intel firmasının ürettiği iki adet işlemcinin boyutları ve transistör sayıları karşılaştırılmıştır. Tabloda da görüldüğü üzere daha küçük alanlara daha çok transistör yerleştirilebilmektedir. Intel i7 işlemcisi, Intel Pentium 4 (P4) işlemcisinin boyutunun %80'i kadardır ve P4'ten 33 kat daha fazla transistör bulundurmaktadır.

Tablo 1.1. Intel Pentium 4 (P4) ve Intel i7 4790 işlemcilerinin karşılaştırması.

İşlemci	**Alanı**	**Üretim Teknolojisi**	**Transistör Sayısı**
Intel Pentium 4 (2000)	217 mm^2	180 nm	42 milyon
Intel i7 – 4790 (2014)	177 mm^2	22 nm	1.4 milyar

Transistörlerin boyutlarının küçülmesi ile çalışma gerilimi de düşürülebilmekte ve birim alana daha çok sayıda transistör koyularak sistemlerin hesaplama ve hafıza kapasiteleri arttırılabilmektedir. Ancak düşük gerilimde çalışan ve çok sayıda transistör içeren sistemlere yüklü parçacıkların çarpma olasılığı ve etkisi de artmaktadır. Bu durum sonucunda sayısal sistemlerin geçici hatalara karşı duyarlılığının daha da artacağı yönünde tahminler yapılmaktadır [3]. Geçici hataların gerçekleşme olasılığındaki artış, günümüz sayısal sistemlerin güvenilirliği üzerindeki önemi de arttırmıştır.

Bir sistemin geçici hatalara karşı hassasiyeti, tasarım aşamasında mümkün olduğunca erken vakitte belirlenmelidir ve gerekli önlemler alınmalıdır. Sistemin hassasiyet öngörüsü yapılırken, hem aşırıya kaçınılmamalı hem de göz ardı edinilmemelidir. Aşırı fazla hassasiyet öngörüsü, alınacak önlemlerle sistemin gereğinden fazla güç harcamasına ve alan kaplamasına neden olacaktır. Göz ardı edilen hassas yapılar da sistemin güvenilirliğinin düşük olmasıyla sonuçlanacaktır.

Mimari Hassasiyet Katsayısı (MHK) [4], günümüzde işlemci yapılarının hassasiyet tahmini konusunda en çok kullanılan ölçüdür. MHK analizi temel olarak bir bitte oluşan hatanın, programın sonucuna etki edip etmediğini tespit ederek sistemin hassasiyetini ölçmektedir. Eğer sistem çalışırken bir bitte hata oluşmuş ve çeşitli nedenlerden dolayı (hatanın oluştuğu yapı program çalışırken kullanılmıyorsa, hatalı bit okunmadan üzerine yeni bir değer yazılıyorsa vb.) sonuç doğru elde edilmiş ise bu hata maskelenmiş demektir ve hatalı biti korumaya ihtiyaç yoktur. Bu maskelenme etkisi incelenerek sistemin ne kadar korunması gerektiği ayarlanabilir.

MHK, bir sistemde programın doğru sonuç vermesi için hatasız olması gereken bit (Mimari Düzeyde Doğru Yürütüm - MDDY biti) sayısının sistemdeki toplam bit sayısına oranı ile hesaplanır. Dolayısıyla MHK analizi, bir bitteki değişimin hata oluşturma süreciyle değil de sonuca etkisi ile ilgilenir. Eğer bir bitteki değişim sonuca etki etmiş ise o bit MDDY bitidir.

Genel sistemin MHK'sı hesaplanırken, sistemdeki her bir bileşenin kendi MHK'ları hesaplanır ve bileşenin kapladığı alanın genel sistemdeki oranıyla çarpılarak toplanır. Elde edilen toplam, sistemin MHK'sı olarak kabul edilir. MHK'nın bu şekilde hesaplanması çok sayıda bileşen içeren ve çalıştırılması çok uzun zaman alan yüklerde bazı uyuşmazlıklara neden olmaktadır [5]. Bu durumun iki temel sebebi vardır: (1) MHK, bütün MDDY bitlerinin (bir bileşende veya buyrukta) toplam hassasiyete aynı etkiyi yaptığını varsayar. (2) MHK, farklı bileşenlerde bulunan MDDY bitlerinin tüm sistemin hassasiyetine aynı etkiyi yaptığını varsayar. Bu varsayımlar sonucunda MHK ile farklı bileşenler arasında ve bir bileşenin farklı bölgeleri arasında gerçekçi bir hassasiyet karşılaştırması yapmak mümkün değildir.

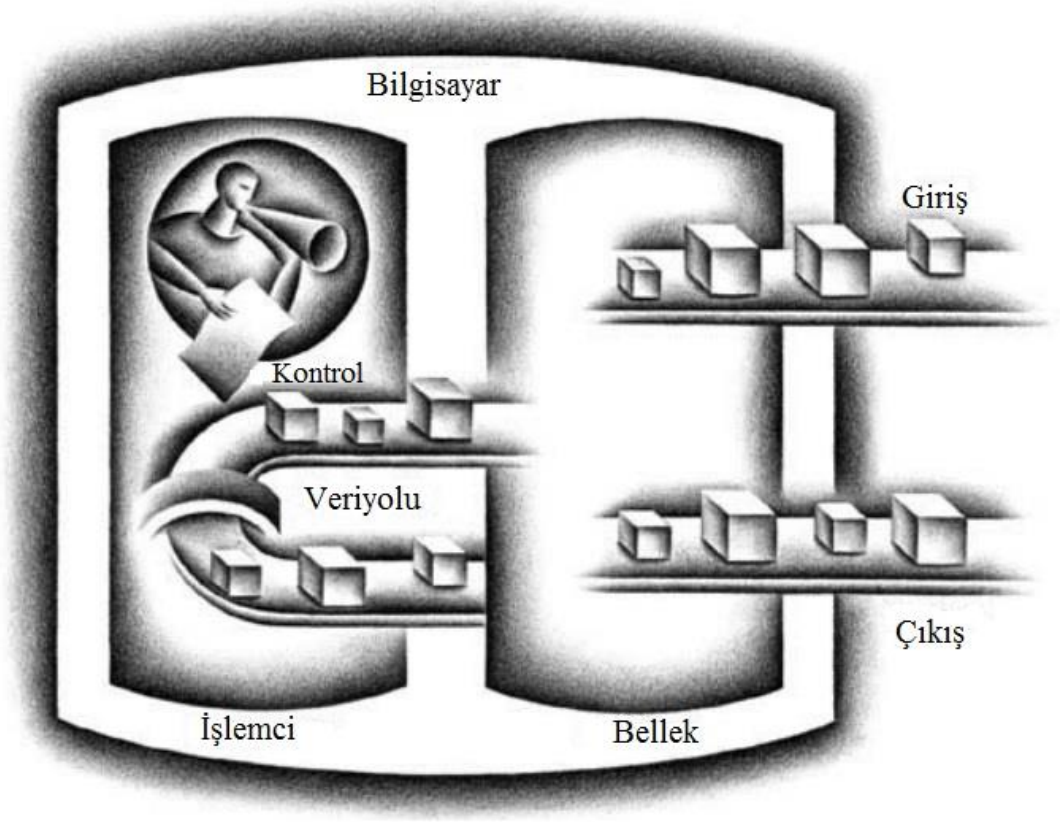
Bu tezin amacı, bir işlemcinin farklı bileşenlerini ve bir bileşenin farklı tipteki bitlerini karşılaştırmayı sağlayan yeni bir ölçüt sunmaktır. Çalışmanın devamında 2. bölümde işlemcilerin mimari yapısı ile ilgili temel bilgiler verilmiştir. 3. bölümde bu konuda yapılan ilişkili çalışmalar sunulmuştur. 4. ve 5. bölümlerde önerilen yöntemin detayları açıklanmış ve elde edilen sonuçlar gösterilmiştir. Son olarak 6. bölümde de çalışma ile ilgili nihai değerlendirmeler yapılmıştır.

2. TEMEL KONULAR

Bu bölümde mikroişlemciler hakkında temel bilgiler verilmiştir. Bu bilgiler, tez kapsamında yapılan çalışmaların anlaşılabilir olması için gereklilik arz etmektedir.

2.1 Mikroişlemci Mimarisi

Genel olarak bir bilgisayarı oluşturan parçalar beşe ayrılır: Giriş, çıkış, bellek, veri yolu ve kontrol birimi. Çoğu zaman son iki bileşen birleştirilir ve işlemci adıyla anılır. İşlemci buyrukları ve verileri bellekten alır. Veriler belleğe girişten gelir ve bellekten çıkışa aktarılarak dışarıdan okunabilir. Kontrol birimi ise veri yolu, bellek, giriş ve çıkışlarda işlemlere karar veren sinyalleri bu birimlere gönderir. Bir bilgisayarın beş bileşene ayrılmış genel düzeni Şekil 2.1'de gösterilmiştir.



Şekil 2.1. Bir bilgisayarın beş temel bileşeni [6].

İşlemcilerin temel görevi, hangi yapıda ve mimaride olursa olsun, bellekte tutulan ve buyruk dizilerinden oluşan programları çalıştırmaktır. Buyruklar, bellekte yazıldıkları sıra ile okunurlar ve çalıştırılmak için işlemciye getirilirler. Çalıştırılırken de belli aşamalardan geçerler ve sonlandırılırlar. Bu şekilde çalışan sistemler sıralı yürütme mantığı ile çalışan sistemlerdir.

Her işlemcide, bir sonraki yürütülecek buyruğun bellekteki adresini gösteren, Program Sayacı (İngilizce: Program Counter - PC) adı verilen yazmaçlar bulunur. İşlemci bu yazmacın belirttiği adresten buyruğu okur ve yürütülmesi için gerekli adımlar uygulanır. Daha sonra Program Sayacının (PS) değeri sıradaki buyruğun getirileceği şekilde güncellenerek programın çalışması sağlanır. Bir sonraki buyruğa geçileceğinde PS'nin değeri çoğu zaman bir buyruğun büyüklüğü kadar arttırılır. Örneğin, 32 bit uzunluğunda buyruklara sahip bayt adresleme sistemini kullanan bir işlemcide, sonraki buyruğun adresine erişmek için PS değeri 4 arttırılır (32 bit = 4 bayt). Ancak atlama buyrukları olarak adlandırılan bazı buyruklar, PS değerini doğrudan güncelleyebilir. Bu buyrukların yürütülmesi sonucunda sonraki adres hesaplanmış olur ve PS yazmacı hesaplanan değer ile güncellenir.

İşlemcide bulunacak aşama sayısına tasarım sırasında karar verilir. Aşama sayısının yüksek olması donanımın karmaşıklaşmasına, az olması da saat sıklığının düşük kalmasına neden olabilir. Buradaki dengenin iyi bir şekilde oluşturulması gerekmektedir.

Örnek olarak, MIPS (Microprocessor without Interlocked Pipeline Stages) mimarisinde buyruklar beş aşamada çalıştırılıp sonlandırılmaktadır. Bu aşamalar ve aşamalarda yapılan işlemler aşağıdaki gibidir:

Getir: PS'nin gösterdiği adresteki buyruk bellekten getirilir.

Çöz: Buyruğun hangi işlemi yapacağı ve işleme girecek olan değerler yazmaçlardan okunur.

Yürüt: İşlem veya adres hesaplanması gerçekleştirilir.

Bellek: Buyruğa göre (okuma veya yazma) bellek erişimi yapılır.

Geri Yaz: Hesaplanan veya bellekten okunan veri, yazmaç öbeğine yazılır.

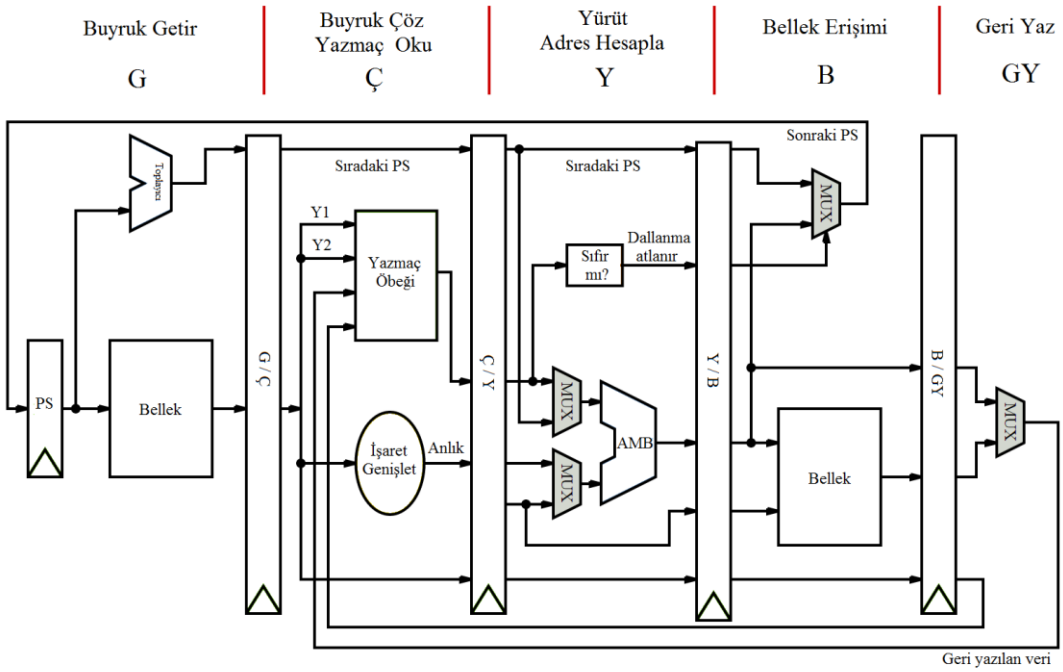
İlk tasarlanan işlemcilerde aynı anda sadece bir buyruk işlenebiliyordu. MIPS mimarisi düşünüldüğünde, bir buyruk geri yaz aşamasını bitirmeden sonraki buyruk getir aşamasına gelemiyordu. Bu durum işlemcideki birimlerde verimsizliğe yol açmaktaydı. Bir buyruk yürütülürken, başka bir buyruk çözülebilir ve bir başkası da getirilebilir. İşlemcinin daha verimli çalışması için önerilen yöntem ise boru hattı sistemidir. Bölüm 2.1.1'de boru hattı yöntemi hakkında temel bilgiler aktarılmıştır.

İşlemci performansını arttırmak için boru hattı yönteminden farklı olarak başka teknikler de geliştirilmiştir. Bunlardan en temel yöntem sırasız yürütmedir (İngilizce: out-of-order execution). Bu yöntem hakkındaki açıklamalar Bölüm 2.1.2'de bulunmaktadır.

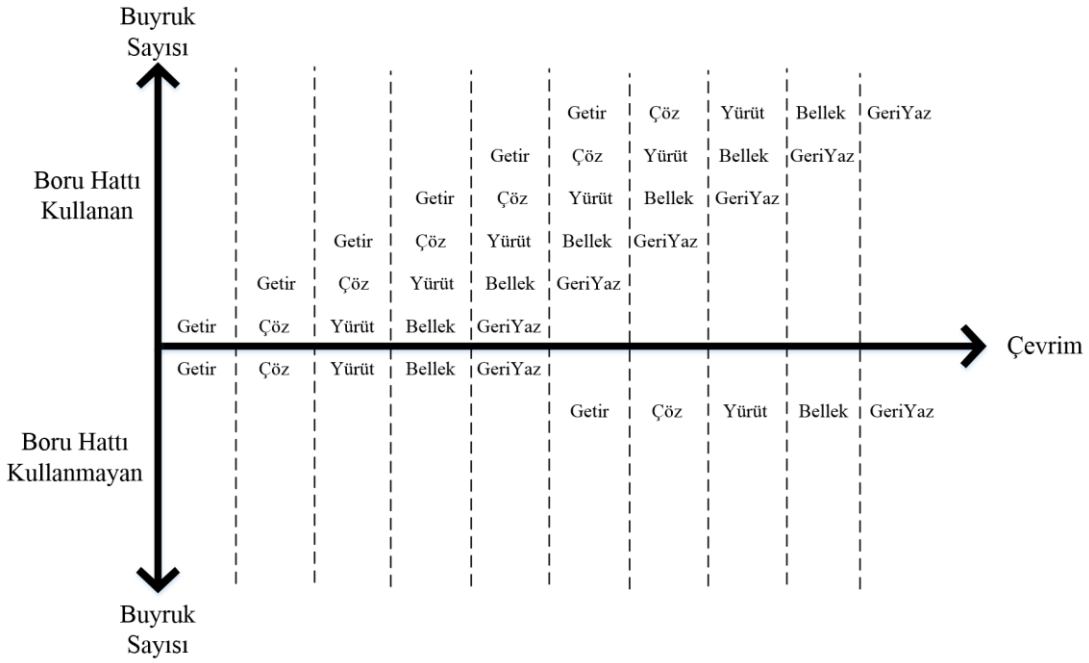
2.1.1 Boru Hattı Yapısı

Bir buyruğun işlenmesi için aşamalardan geçmesi, bir sıvının boru hattından geçmesine benzetilmiştir. Sıvının bir yerden başka bir yere aktarılması için tüm hat boyunca ilerlemesi gerekmektedir. Sıvı, akışını gerçekleştirirken boru hattı tamamen dolu olur, sıvı belli miktar gruplarında aktarılmamaktadır. Aynı şekilde bir buyruk işlemcinin boru hattında ilerlerken, diğer buyruk öncekini beklemeden hatta girebilir. Böylece her aşamada sürekli bir buyruk bulunur ve verimlilik arttırılmış olur. Güncel işlemciler boru hattı sistemini uygulamaktadırlar. Tasarımlara göre aşama sayısı değişkenlik göstermektedir. Şekil 2.2'de beş aşamalı boru hattına sahip MIPS mimarisi verilmiştir.

Şekil 2.3'te MIPS mimarisi için boru hattı olan ve olmayan iki durum için, zaman içinde buyruklar işlenirken geçtikleri aşamalar gösterilmiştir. Aynı sürede boru hattı kullanıldığında işlenen buyruk sayısının daha fazla olduğu açıkça görülmektedir.



Şekil 2.2. MIPS işlemcisinin beş aşamalı boru hattı tasarımı



Şekil 2.3. Beş aşamalı MIPS işlemcisinde boru hattı kullanıldığında ve kullanılmadığında buyrukların zaman içinde geçtiği aşamalar

Boru hattı kullanımının faydaları ile birlikte getirdiği bazı sorunlar bulunmaktadır. Bu sorunlar üç grupta incelenebilir: Yapı sorunu (İngilizce: structural hazard), veri sorunu (İngilizce: data hazard) ve denetim sorunu (İngilizce: control hazard).

Yapı sorununun nedeni, işlemcide bulunan birimlerin fiziksel olarak aynı anda birden fazla buyruk için çalışamamasından kaynaklanmaktadır. Örnek olarak yazmaç dosyası düşünülürse, boru hattındaki bir buyruk çöz aşamasındayken yazmaç dosyasından okuma yapmak isteyecek, bir başka buyruk da geri yaz aşamasında hesaplanan sonucu yazmaç dosyasına yazmak isteyecektir. Eğer yazmaç dosyasında fiziksel olarak yeterli sayıda giriş çıkış yolu yok ise bu anda bir sorun oluşacaktır. Yapı sorunlarının çözümü, bu tarz sorunlara neden olabilecek birimler için yeterli sayıda giriş – çıkış yolu oluşturmaktır.

Bir buyruk, kendisinden bir önceki buyruğun hesapladığı sonucu kullanıyorsa, yani veri bağımlılığı varsa, bu durumda veri sorunu oluşur. Önceki buyruk geri yaz aşamasını bitirmeden sonraki buyruk çöz aşamasında doğru değeri yazmaç öbeğinden okuyamayacaktır. Çözüm ise veri yönlendirmesidir (İngilizce: data forwarding). Yürüt aşamasından ve bellek aşamasından sonra çöz aşamasına bir yol bağlanırsa, bağımlı buyruk çöz aşamasında iken yazmaç öbeğinden okumak yerine bağlanan yoldaki veriyi okuyarak normal akışına devam edebilir.

Denetim sorununun sebebi ise dallanma buyruklarının boru hattına girmesidir. Dallanma buyrukları PS yazmacını farklı bir adres hesabı ile güncelleyebilir. Yeni adres hesabı en erken çöz aşamasında yapılacağından, dallanma buyruğundan sonra hangi buyruğun getirileceği denetim sorununu oluşturmaktadır. Çözüm olarak en yaygın kullanılan yöntem dallanma öngörüsü (İngilizce: branch prediction) yapmaktır. Dallanma buyruğundan sonra hangi buyruğun getirileceği tahmin edilir ve buyruk işlenmek için boru hattına girer. Eğer tahmin doğru olursa herhangi bir sorun olmadan program çalışmaya devam eder. Tahmin yanlış olduğunda ise, tahmin edilen buyruklar geçersiz olarak işaretlenir ve sonuçları dikkate alınmaz. Günümüzdeki dallanma öngörücüleri %99'a varan doğrulukta öngörü yapabilmektedirler.

Aslında her üç sorun için ortak çözüm olarak buyrukların bekletilmesi (İngilizce: stall) düşünülebilir. Gereken miktarda saat çevrimi beklendiğinde, donanım üzerinde değişiklik yapılmadan bu sorunlar giderilebilir. Ancak günümüzde bir

programın yürütme zamanı mümkün olduğunca en aza indirgenmek istenildiği için beklemek verimli bir çözüm olmaz. Ayrıca teknolojinin ilerlemesi ile donanımda yapılacak değişikliklerin maliyeti, buyrukların bekletilmesinden kaynaklanan zaman maliyetine göre göz ardı edilebilir düzeydedir.

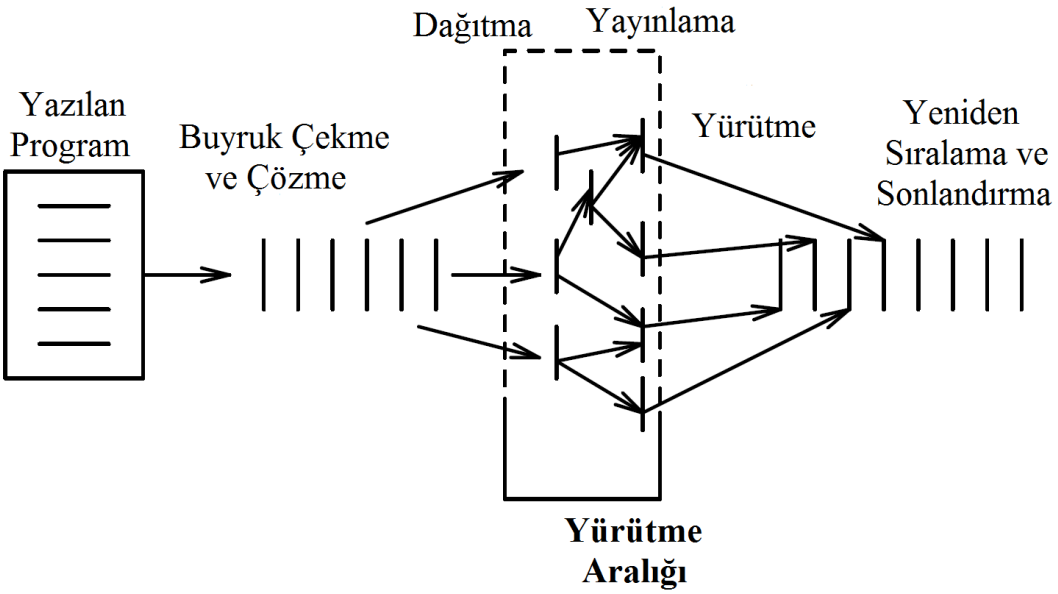
2.1.2 Sırasız Yürütüm

Sırasız yürütüm tekniği günümüz işlemcilerinde performansı arttırmak için kullanılan yöntemlerden biridir. Temel amaç, buyrukları programda bulundukları sıra yerine, işleme girecek verilerin hazır olup olmamasını kontrol ederek yürütmektir. Böylece bir buyruğun kullanacağı veriler hazırken, kendinden önceki buyrukların yürütülmesini beklemeden hazır olan bir işlem biriminde yürütülerek, hem zaman yönünden hem de işlem birimlerinin kullanımı yönünden verimlilik sağlanmış olur.

Sıralı yürütüm mantığında çalışan bir işlemcide bir buyruk öncelikle işlemciye getirilir; işleme girecek değerler hazırsa (örneğin yazmaç dosyasında) ilgili yerden okunur, değilse beklenir; buyruk ilgili işlem biriminde yürütülür ve sonucu yazılır. Sırasız yürütümde ise buyruk getirilir; buyruk bir arabelleğe alınarak kullanacağı verilerin hazır olmasını beklenir; verileri hazır olan buyruk kendinden öncekileri beklemeden yürütülmek için uygun bir işlem birimine gönderilir; sonuçlar sıralanır ve en eski buyruğun sonucu ilk yazılacak şekilde buyruklar sonlandırılır.

Sırasız yürütüm kullanan işlemcilerde buyruklar yürütüldükleri aşama dışında programda yazıldıkları sırayla ilerlemektedirler. Sırayla getirilirler, sırayla çözülürler, sırasız yürütülürler ve sırayla yazılırlar. Konsept olarak sırasız yürütüm Şekil 2.4'te belirtilmiştir.

Bir işlemcide sırasız yürütüm yöntemini gerçekleştirebilmek için bazı yeni yapılara ve kabiliyetlere ihtiyaç vardır. Bunların en temel olanları ise Yazmaçların Yeniden Adlandırılması (İngilizce: Register Renaming) ve Yeniden Adlandırma Tablosu (İngilizce: Renaming Table), Yeniden Sıralama Arabelleği (İngilizce: Reorder Buffer) ve Rezervasyon İstasyonu'dur (İngilizce: Reservation Station).



Şekil 2.4. Sırasız yürütüme sahip işlemcinin aşamaları [7].

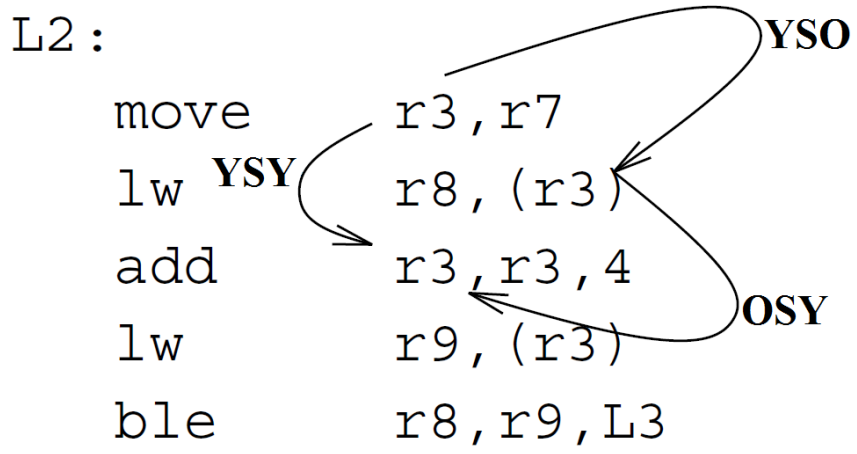
2.1.2.1 Yeniden Adlandırma

Çoğu zaman programların buyrukları arasında veri bağımlılıkları bulunur. Birden fazla buyruk belli bir zaman içerisinde aynı saklama alanına (yazmaç, bellek vb.) erişmek (yazma veya okuma) isteyebilir. Bu durumlar sorun olarak görülür ve gerekli önlemler alınmadığı takdirde, erişimlerin doğru sırada yapılması sağlanmadığında, programın sonucu yanlış elde edilebilir. Buyruklar arası oluşabilecek sorunlar üç gruba ayrılmıştır. Bunlar Yazma Sonrası Yazma (YSY), Okuma Sonrası Yazma (OSY) ve Yazma Sonrası Okuma'dır (YSO). Her sorun için örnek içeren bir kod parçası Şekil 2.5'te verilmiştir.

YSY (İngilizce: Write-After-Write – WAW) sorunu birden fazla buyruğun aynı saklama alanına veri yazmak istemesi durumudur. Bu soruna çıktı bağımlılığı (İngilizce: output dependency) da denir. Verilerin programdaki sırada yazılması gerekiyormuş gibi gözükür.

OSY (İngilizce: Write-After-Read – WAR) sorunu, bir buyruğun güncelleyeceği saklama alanının, bu buyruktan önce o alanı okuyacak başka buyruklar varsa ortaya çıkar. Diğer adı karşıt bağımlılığıdır (İngilizce: anti dependency). Bir

buyruk, ilgili saklama alanını güncellemeden, önceki buyrukların bu alandaki değeri zamanında okuması sağlanmalıdır.



Şekil 2.5. Veri sorunları [7].

OSY (İngilizce: Write-After-Read – WAR) sorunu, bir buyruğun güncelleyeceği saklama alanının, bu buyruktan önce o alanı okuyacak başka buyruklar varsa ortaya çıkar. Diğer adı karşıt bağımlılığıdır (İngilizce: anti dependency). Bir buyruk, ilgili saklama alanını güncellemeden, önceki buyrukların bu alandaki değeri zamanında okuması sağlanmalıdır.

YSO (İngilizce: Read-After-Write – RAW) sorunu ise bir buyruğun ürettiği sonucun, kendinden sonraki buyruklar tarafından kullanılacağı zaman meydana gelir. Bu soruna gerçek bağımlılık (İngilizce: true dependency) denir. Bir buyrukta kullanılacak değerlerin zamanında güncellenmesi gerekmektedir.

YSY ve OSY çoğu zaman yapay bağımlılıklar (İngilizce: artificial dependencies) olarak anılır. Başlıca nedenleri yeteri kadar optimize edilmemiş kodlar, sınırlı sayıda yazmaç sahibi olma, belleği mümkün olduğunca ekonomik kullanma isteği ve bazı döngü kullanımlarıdır. Bu sorunların giderilmesi için Yazmaçların Yeniden Adlandırılması (YYA) kullanılmaktadır.

Sırasız yürütüm yapan işlemcilerde genelde iki tip yazmaç öbeği bulunmaktadır. Bunlar mimari yazmaç dosyası (İngilizce: Architectural register file) ve fiziksel yazmaç dosyasıdır (İngilizce: Physical register file). Fiziksel yazmaç dosyasının (FYD) büyüklüğü, mimari yazmaç dosyasının (MYD) büyüklüğünde ya da daha

büyük (daha çok yazmaç) olacak şekilde tasarlanır. MYD, derleyiciler tarafından kullanılır. Üst seviye dilde yazılan program makine diline dönüştürülürken, MYD'de bulunan yazmaçlar ile buyruklar yazılır. İşlemci içerisinde ise FYD kullanılır. Boru hattına giren buyruklar, çöz aşamasından sonra fiziksel yazmaçlar ile işleme sokulurlar.

Yeniden adlandırma işlemini sağlıklı yapabilmek için iki yapıya ihtiyaç vardır: Yeniden Adlandırma Tablosu (YAT) ve Serbest Yazmaç Listesi (SYL). Mimari bir yazmaca atanacak fiziksel yazmaç SYL'den seçilir. Burada bulunan yazmaçlar eşleştirmeye uygun olanlardır. Yapılan eşleştirme YAT'ye kaydedilir ve ilgili yazmaç numaraları buradan takip edilir.

Buyruklar çözüldükten sonra yeniden adlandırma aşamasına gelirler. Burada buyruğun önceki buyruklara bağımlılığı incelenir ve varsa mimari sonuç yazmacı numarasına karşılık uygun bir fiziksel yazmaç numarası atanır. Mimari kaynak yazmacı numaraları da YAT incelenerek fiziksel olanlar ile güncellenir. Sonrasında buyruk boru hattında ilerlemeye uygun hale gelir ve Yeniden Sıralama Arabelleğinde kendisine yer ayrılır. Tablo 2.1'de örnek bir MIPS kodunun yeniden adlandırma sonrasında nasıl çalıştırılacağı bulunmaktadır. Büyük harfle yazılan yazmaçlar mimari, küçük harfle yazılanlar ise fiziksel yazmaçları göstermektedir.

Tablo 2.1. Örnek bir yazmaç yeniden adlandırma. $0 yazmacının değeri her zaman sıfırdır.

	YA Öncesi	**YA Sonrası**	**Yapılan İşlem**
1	lw R1, 0[r26]	lw r6, 0[r26]	r6 = Bellek[r26 + 0]
2	addi R1, R1, 8	addi r6, r6, 8	r6 = r6 + 8
3	sw R1, 0[r26]	sw r6, 0[r26]	Bellek[r26 + 0] = r6
4	and R1, r13, $0	and r10, r13, $0	r10 = r13 & $0
5	add R1, R1, r18	add r10, r10, r18	r10 = r10 + r18
6	sw R1, 0[r21]	sw r10, 0[r21]	Bellek[r21 + 0] = r10

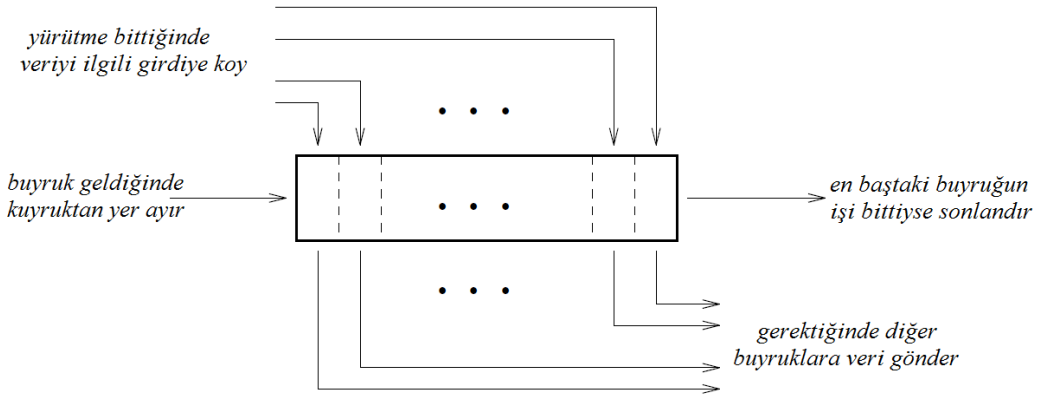
Fiziksel bir yazmacın ömrü, yeniden adlandırma aşamasında bir buyruğun, mimari yazmaç numarasına sahip sonuç yazmacına atanarak başlar ve boru hattı içerisinde bu fiziksel yazmacı okuyacak başka bir buyruk kalmayana kadar devam eder. Aradaki sürede birden çok kez okunabilir. Okuyacak buyruk kalmadığında ise ilgili satır YAT'den silinir ve fiziksel yazmaç numarası SYL'ye eklenir.

Geçen bu süre boyunca yazmacın hassas durumda olduğu kabul edilir ve korunmaya ihtiyacı vardır. Günümüzdeki çoğu işlemcide fiziksel yazmaç öbekleri hata düzeltme kodları ile korunmaktadır [8] [9].

2.1.2.2 Yeniden Sıralama Arabelleği

Yeniden Sıralama Arabelleğinin (İngilizce: Reorder Buffer – ROB) temel görevi sırasız yürütülen buyrukların program sırasında sonlandırılmasını sağlamaktır. Basit bir kuyruk yapısındadır. Kuyruktaki en eski buyruk kuyruğun en başındadır ve yeni gelen buyruklar kuyruğun en sonuna eklenir. Yeniden Sıralama Arabelleğindeki (YSA) bir girdide bulunması gereken en önemli alanlar buyruğun tipi (atlama, belleğe yazma, yazmaca yazma), sonuç yeri (bellek adresi veya yazmaç numarası), sonuç ve girdinin geçerliliğini bildiren bit.

Çöz aşamasından sonra yeniden adlandırılan buyruklar için yeni girdi oluşturulur ve arabelleğe eklenir. Gerekli bilgiler ilgili işlem biriminin rezervasyon istasyonuna gönderilir ve yürütülene kadar istasyonda bekler. Buyruk yürütüldükten sonra YSA'da en başa gelene kadar burada kalır. Kendisinden önce buyruk kalmayınca sonlandırılması yapılır ve YSA'dan çıkarılır. Şekil 2.6'da YSA'ya giren ve çıkan verilerin neler olduğu belirtilmiştir.

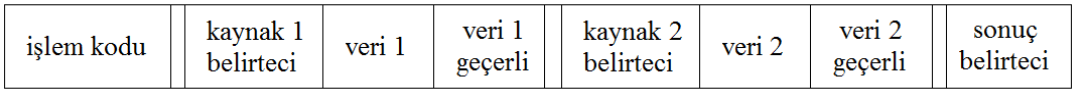


Şekil 2.6. Yeniden Sıralama Arabelleğine veri girişleri ve çıkışları [7].

2.1.2.3 Rezervasyon İstasyonu

İlk olarak Tomasulo'nun algoritmasında önerilmiştir [10]. Rezervasyon İstasyonları'nın (Rİ) sağladığı en önemli kolaylık, buyrukların sırasız bir şekilde yayınlanmasını (İngilizce: issue) sağlamaktır. Bütün işlem birimlerine özel veya hepsine ortak olacak şekilde tasarlanabilir.

Örnek bir Rİ girdisi Şekil 2.7'de sunulmuştur. Burada işlem kodu, işlemde kullanılacak işlenenler (İngilizce: operand), kaynak belirteçleri, kaynakların geçerliliğini gösteren bitler ve sonuç belirteci (bellek veya yazmaç numarası) bulunmaktadır.



Şekil 2.7. Örnek bir Rezervasyon İstasyonu girdisi [7].

İşlemcide bir buyruk yürütüldüğünde ve sonuç üretildiğinde, sonuç belirteci ile birlikte bütün Rİ'ler bilgilendirilir. Eğer herhangi bir Rİ hesaplanan veriyi bekliyorsa (belirteçler eşleşiyorsa), veriyi alır ve geçerlilik bitini günceller. Bir buyruğun bütün işlenenleri hazırsa, buyruk uygun bir işlem birimine gönderilir ve yürütülür. Yürütme sonrası diğer Rİ'lerin de kullanabilmesi için sonuç belirteci de buyrukla birlikte gönderilir.

Sırasız yürütüm yapan işlemcinin, bahsedilen yapıların kullanımıyla boru hattındaki aşamaları artmaktadır. En basit sırasız yürütüm yapan işlemci için gerekli aşamalar aşağıda belirtilmiştir. Şekil 2.8'de örnek bir sırasız yürütüm yapan boru hattı yapısı gösterilmiştir.

Getir: PS'nin gösterdiği adresteki buyruk bellekten getirilir.

Çöz: Buyruğun hangi işlemi yapacağı anlaşılır.

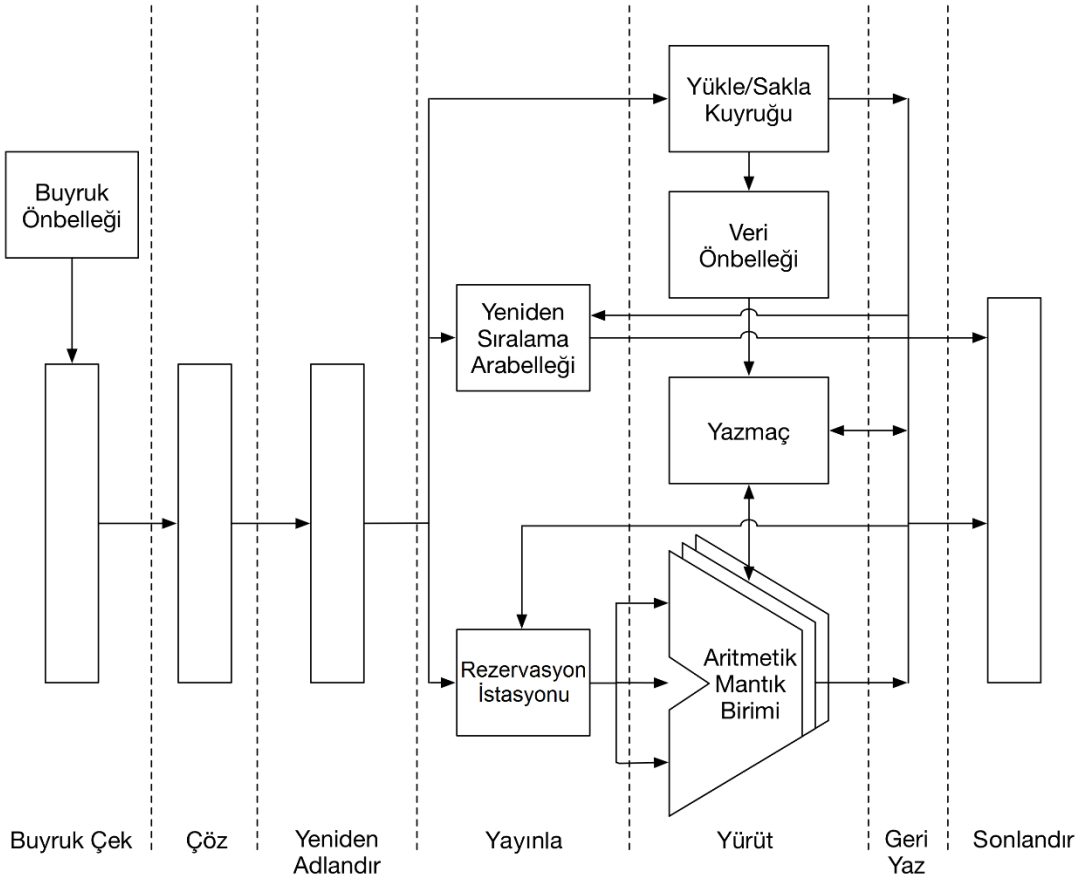
Yeniden Adlandır: Yazmaçlar yeniden adlandırılarak gerçek olmayan bağımlılıklar ortadan kaldırılır ve yeniden adlandırma tablosu güncellenir.

Yayınla: Yeniden adlandırılan buyruklara YSA'da yer ayrılır. YSA içerisinde kaynak yazmaçları hazır olan bir buyruk varsa önceki hazır olmayan buyruklar beklenmeden ilgili işlem biriminin rezervasyon istasyonuna gönderilir.

Yürüt: Boşta olan işlem birimleri rezervasyon istasyonlarında bekleyen buyruk varsa onları okur ve yürütür. Yürütme sonrası hangi yazmaç güncellenecekse onun bilgisi ilgili tüm birimlere bildirilir.

Geri Yaz: Hesaplanan veya okunacak veriler, ilgili yerlere yazılır.

Sonlandır: Bütün işlemleri biten buyruklar YSA'dan ilk girdikleri sıra ile çıkarılır.



Şekil 2.8. Sırasız yürütüme sahip bir işlemcinin boru hattı yapısı.

2.2 Mimari Hassasiyet Katsayısı

İşlemci çalışırken, bir bölümüne yüklü parçacık çarpması sonucu o bölgede bulunan mantık devrelerindeki bit veya bitlerin değeri değişebilir. Bu değişim çalışma sonunda elde edilen sonuca etki etmeyebilir. Bu durumun sebebi, bir program çalışırken doğru sonuca ulaşmak için işlemci içerisinde bulunan bitlerin kullanılmasına gerek yoktur. Çalışan programın doğru sonuç vermesi için hatasız çalışması gereken bitlere Mimari Düzeyde Doğru Yürütüm (MDDY) bitleri denir [4]. Çalışma esnasında işlemcinin her bileşeninde farklı sayılarda MDDY bitleri bulunmaktadır. Bu bitlerin sayısı sistemin geçici hatalara karşı hassasiyet seviyesini belirlemektedir. MDDY bitlerinin sayısı fazla ise, bir bitlik hatanın sonuca etki etme ihtimali de o kadar fazla olur. Bileşenlerin geçici hatalara karşı hassasiyetleri Mimari Hassasiyet Katsayısı (MHK) terimi ile ifade edilir. MHK, tasarım aşamasında işlemcinin hassasiyet düzeyini belirlemek için kullanışlıdır. Genel olarak MHK hesabı Denklem 2.1'de verilmiştir.

$$MHK = \frac{Donanımdaki\ Ortalama\ MDDY\ Bitlerinin\ Sayısı}{Donanımdaki\ Toplam\ Bit\ Sayısı} \tag{2.1}$$

2.3 Hamming Uzaklığı

Bilgi kuramında, aynı uzunluktaki iki dizinin (sayı, harf, sembol vb.) aynı sıradaki farklı elemanlarının sayısı, bu iki dizinin birbirlerine olan Hamming uzaklığını verir. Başka bir deyişle, bir diziden başka bir diziyi elde etmek için gerekli olan eleman değişikliği sayısıdır. Adını Richard Hamming'den almıştır. Tablo 2.2'de farklı tipteki ve uzunluktaki dizilerin, birbirlerine olan Hamming uzaklıkları verilmiştir.

Tablo 2.2. Hamming uzaklık örnekleri

Dizi 1	**Dizi 2**	**Hamming Uzaklığı**
Ser**c**an	Ser**k**an	1
T**eke**r**l**ek	T**opa**r**l**ak	4
66872**36**	66872**63**	2
11**0**1**0**1001**0**	11**1**0**0**0011	4

3. İLİŞKİLİ ÇALIŞMALAR

Bu bölümde bir sistemin bileşenlerinin hassasiyetinin hesaplanması konusunda yapılan ilişkili çalışmalar sunulmuştur.

İşlemcinin başarısızlık (yanlış sonuç üretme veya sonuç üretememe) oranı, MHK ve MDDY analizi yapılarak tasarım aşamasında ölçülmüştür [4]. Bitler MDDY ve MDDY olmayan şeklinde ikiye ayrılmıştır. MDDY bitleri, programın sonucunun doğru olması için hatasız bir şekilde çalışması gereken bitler olarak belirlenmiştir. MDDY olmayanlar ise bitler ise donanımsal (boşta olan, geçersiz ya da yanlış tahmin edilen durumlar, öngörücü yapıları vs.) veya mimari (NOP buyrukları, performans arttırıcı buyruklar, dinamik olarak ölü buyruklar veya maskeleme) sebeplerden dolayı sonuca etki etmeyen bitler olarak belirlenmiştir.

Çalışma süresi uzun olan programların çok bileşenli ve büyük sistemlerde yürütülmesi durumunda MHK hesaplandığında hassasiyetin doğru bir şekilde belirlenemediği gösterilmiştir [5]. İşlemcinin hassasiyetini netleştirebilmek için MDDY olmayan bitlere yenileri eklenmiştir [11]. Y-bitleri, kontrol buyrukları ve kullanılmayan alanlar da MDDY olmayan bitler olarak değerlendirilmiştir. Ayrıca MHK'nın daha net bir şekilde hesaplanabilmesi için detaylı MHK analizinin nasıl yapılması gerektiği sunulmuştur [12].

Farklı bileşenlerin MHK'larının hesaplanması konusunda çeşitli çalışmalar yapılmıştır. Adres tabanlı yapıların (önbellekler, Etkin Sayfalar Önbellekleri (İngilizce: Translation Lookaside Buffer – TLB) ve saklama arabellekleri) hem veri alanının hem de etiket alanının MHK'ları hesaplanmıştır [13]. Ayrıca saklama alanlarındaki adres etiketlerinin, Hamming uzaklığı 1 olan başka bir adres etiketinin olup olmadığı da incelenmiştir. Bu şekilde herhangi bir hata oluşumunda farklı bir adres satırının seçilip seçilemeyeceği konusunda tahminler yapılmıştır. Aynı yöntemle ikinci seviye (L2) önbelleklerin boyutundaki doğrusal artışın, önbelleğin hataya karşı hassasiyetini süper-doğrusal olarak arttığı gösterilmiştir [14].

MHK, genellikle bitlerin hassasiyetlerini mikro mimari düzeyde belirler. Bu tanımın buyruk düzeyinde genişletilmiş haline Program Hassasiyet Katsayısı (PHK) (İngilizce: Program Vulnerability Factor - PVF) denilmiştir [15]. PHK hassasiyeti sadece Buyruk Kümesi Mimarisi (İngilizce: Instruction Set Architecture – ISA) seviyesinde inceler. Benzer şekilde yazmaç öbeğinin MHK değerinin durağan olarak hesaplanması da önerilmiştir [16]. Bu yöntemle birlikte mikro mimariden bağımsız olarak, bir programın farklı aşamalarının da hassasiyetlerinin hesaplanması mümkün kılınmıştır. Bir programın daha hassas olan aşamaları belirlenebildiği için, sadece o aşamalara özel koruma yöntemleri geliştirilebilir.

MHK ve PHK arasında bağlantı kurabilmek için Donanım Hassasiyet Katsayısı (DHK) (İngilizce: Hardware Vulnerability Factor - HVF) tanımı yapılmıştır [17]. DHK, sadece donanımın hassasiyetini hesaplar ve bunu yaparken mimari ve mikro mimari birimlerinin farklı inceler. DHK program seviyesindeki maskelemelerden etkilenmediği için donanım tasarımcılarına bileşenlerin hassasiyeti konusunda daha sağlıklı sonuç verir. Ayrıca DHK, PHK değerinden MHK değerini kolayca hesaplanmasına olanak sağlar.

Mimari ve programların yanı sıra, işlemcilerin üretim teknolojilerinin de hassasiyeti etkilediği belirlenmiştir [18]. Üretim çeşitliliğinin hassasiyete etkisini belirtebilmek için, çeşitli cihaz özellikleri (eşik gerilimi ve transistör uzunluğu gibi) analiz hesabına parametre olarak eklenmiştir.

MHK hesaplamalarda bütün bitlerin eşit derecede önemli olduğunu varsayar. Ancak bu varsayım özellikle grafik uygulamalarında doğru değildir. Geçici hataların grafik işlemcilerindeki etkisini incelemek için Görsel Hassasiyet İzgesi (İngilizce: Visual Vulnerability Spectrum – VVS) önerilmiştir [19].

Benzetimciler üzerinde yapılan MHK analizi dışında, bazı çalışmalarda [20] [21] [22] [23] tasarımdan sonra kullanımda olan işlemciler üzerinde MHK tahmini çalışmaları yapılmıştır. Bu çalışmalarda sistemin dinamik olarak hassasiyeti arttırmaya yönelik olan koruma yöntemlerine nasıl adapte olabileceği üzerine odaklanılmıştır.

İşlemcilerin geçici hatalara karşı hassasiyetini etkileyen en büyük etmenlerden birinin çalıştırılan programlar olması sebebiyle, gözlemlenebilecek en yüksek geçici hata olasılığı tespiti üzerine çalışmalar yapılmıştır [24].

Bahsedilen bu çalışmaların hiçbirisi farklı MDDY bitlerinin, sistemin hassasiyetine farklı yönde etki ettiği konusunu işlememişlerdir. Bu tez kapsamında ilgilenilen konu, MHK'yı bu yönde geliştirmektir.

4. ÖNERİLEN YÖNTEM

Bu çalışmadaki amaç, farklı işlemci bileşenlerinin ve bir bileşendeki farklı bitlerin hangisinin geçici hatalara karşı daha hassas olduğunu belirtebilmektir. Bu sonuca ulaşabilmek için öncelikle bir bileşenin bütün bitlerinin aynı tipte olmadığını ve farklı tipteki bitlerde oluşan hataların etkisinin de farklı olduğunu göstermek gerekir.

Örnek olarak, bir buyruğun anlık (İngilizce: immediate) kısmındaki bir bitin değişmesi, buyruğun çalışması sonucunda bir bitlik hataya neden olabilir. Ancak buyruğun kaynak yazmacı kısmında gerçekleşen bir bitlik değişim, buyruk çalıştırıldıktan sonra çok sayıda bitin hatalı olarak elde edilmesine yol açabilir.

Öncelikle farklı işlemci bileşenlerinin (yazmaç dosyası, yeniden sıralama arabelleği ve rezervasyon istasyonu) içerdiği bitler hassasiyet düzeyine göre sınıflandırılacaktır. Sonrasında her bit sınıfında oluşabilecek bir bitlik hatanın etkisi incelenecektir. Bir bitin hassasiyet düzeyini belirten Bit Etki Katsayısı (BEK) tanımlanacak ve işlemci bileşenleri arasında veya bir bileşenin bitleri arasında hassasiyet karşılaştırması yapılmasına olanak sağlanacaktır.

4.1 Bitlerin Sınıflandırılması

Bütün hassasiyet katsayısı tanımlarında esas amaç bir bitteki değişimin belirlenen bir süre içerisinde gözle görülür bir hataya neden olup olmayacağını belirlemektir. Bu yaklaşım bitler arasında herhangi bir hassasiyet seviyesinin olmadığını varsayarak yapılır; bir bit, hataya karşı hassastır ya da değildir. Gerçekte ise işlemci bileşenlerindeki hatalar farklılık gösterir. Kimi bir bitlik hatalar, çok sayıda yanlış bit içeren sonuçlar elde edilmesine sebep olabilir ve sistemin çökme sürecini hızlandırabilir. Kimi bir bitlik hatalar ise sadece bir bit hata olarak kalabilir.

Şekil 4.1'de örnek olarak bir MDDY buyruk sunulmuştur. Buyrukta bazı bitlerde hata oluşması, diğerlerine nazaran daha büyük bir hataya neden olabilmektedir.

Verilen örnek programda bir sayı 3 ile çarpılmaktadır ve sonuç 8'den küçük ise sayı 2 arttırılmaktadır. Programın doğru yürütülmesi sonucunda R1 yazmacı 4 değerini içerir. Karşılaştırma işlemi sonrası (I3), R0 yazmacı MDDY olmaktan çıkar, çünkü üzerine 0 yazılacaktır.

```
I1: mov R1, 0010b       // R1=2
I2: mul R0, R1, 0011b  // R0=R1*3
I3: cmpa 1000b          // Eğer R0<8 ise
I4: bhi end             //
I5: add R1, 0010b       // R1=R1+2
  end:                  //
I6: mov R0,0000b        //  R0=0
```

Hatasız Yürütüm	R0 0000	**0110**	0110
	R1 0000	**0010**	**0100**
Anlıkta Hata (0011 -> 0010)	R0 0000	**0100**	0100
	R1 0000	**0010**	**0100**
Sonuç Belirtecinde Hata (R0 -> R1)	R0 0000	**0000**	0000
	R1 0000	**0110**	**1000**

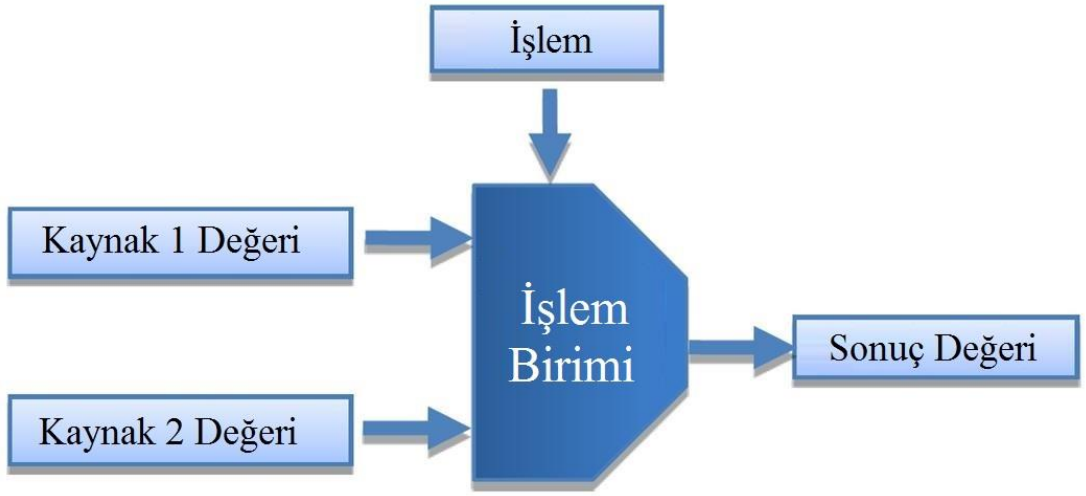
Şekil 4.1. Örnek bir MDDY buyruk. Gri alanlar o anda MDDY değildir.

Şekilde çarpma buyruğunda (I2) iki farklı hata durumu oluşturulmuştur. Birincisinde hata anlık değerdedir ve sayı 3 yerine 2 ile çarpılmıştır. Bu durumda anlık değerdeki değişim dallanma buyruğunun sonucunu değiştirmemiştir ve sonuç olarak yine R1 yazmacı 4 değerini tutar. İkinci durumda ise hata sonuç yazmacını tutan bölgededir ve elde edilen sonuçlar R0 yazmacı yerine R1 yazmacına yazılmaktadır. Bu değişim yine karşılaştırma buyruğunun sonucunu etkilememiştir ve dallanma atlanmamıştır ama R1 yazmacına yazılan değer, yazılması gereken değerden oldukça farklıdır. Dolayısıyla, yürütme sonucunda, örnekte de görüldüğü üzere sonuç yazmacı alanındaki bitlerde oluşabilecek bir bitlik hata, anlık değeri tutan bitlerde oluşabilecek bir bitlik hatadan daha yüksek bir etki oluşturacaktır.

Bir bitteki değişimin neden olacağı hatalı bit sayısı, bir hatanın etkisinin maskelenebilirliğini nicelemek için kullanılabilir. Önceki örnekteki gibi, anlık değerdeki bir hatanın maskelenme olasılığı, sonuç yazmacı belirteci alanındaki bir hatanın maskelenme olasılığından daha fazladır. Buna dayanarak bir bitteki hata ile sonraki bağımlı bileşende oluşan ortalama hatalı bit sayısını belirten BEK tanımı yapılacaktır.

Hatalı bir bit, bulunduğu yerden okunmadığı ve herhangi bit hesaplamada kullanılmadığı sürece sisteme etki etmez. Bütün okunan veriler elbet işlem

birimlerine gelecektir ve bu veriler üzerinde bir işlem yapılacaktır. Dolayısıyla Şekil 4.2'de gösterildiği gibi işlem birimleri maskelemenin derecesini belirleyen bileşenlerdir.



Şekil 4.2. Bir işlem biriminin giriş ve çıkışları

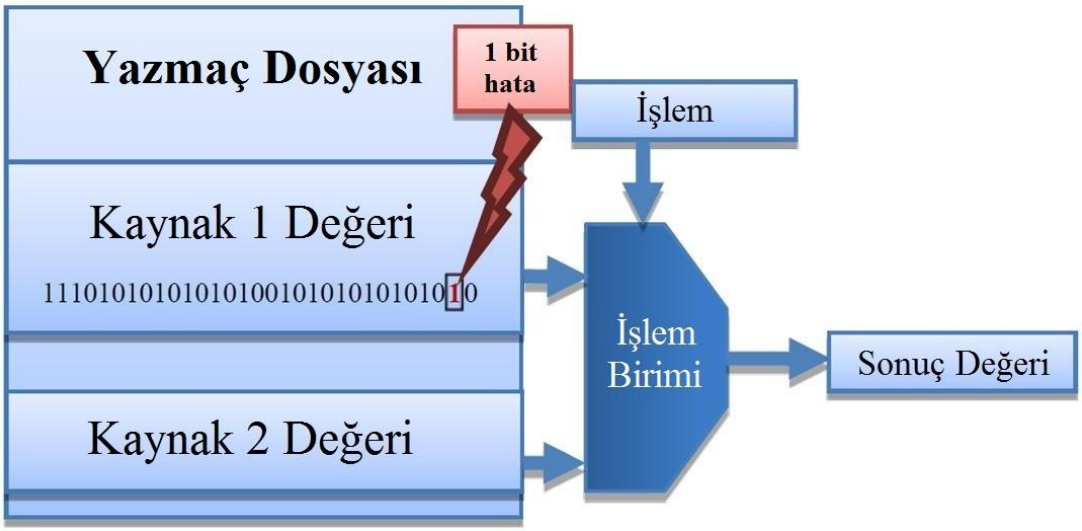
İşlem birime gelen girdilere ve işleme göre oluşacak hatalı bit sayısı hesaplanabilir. Öncesinde işlem birimlerine girdi olarak gelen, işlemci bileşenlerinde bulunan bilgi çeşitlerinin sınıflandırılması yapılmıştır. Bu sınıflandırma aşağıdaki gibidir:

1. Sıradan Veri (yazmaçlar, anlık değerler vb.)
2. İşlem Kodları (Aritmetik Mantık Birimi işlemi, fonksiyon bitleri)
3. Kaynak Belirteçleri (Kaynak fiziksel yazmaç etiketleri)
4. Sonuç Belirteçleri (sonuç etiketleri, YSA numarası, Yükle/Sakla Kuyruğu numarası gibi yapı girdi numaraları)
5. Kontrol Bilgileri (Saklanan verinin durumunu belirten bitler, ör: hazır, geçerli vb.)
6. Sistem Bitleri

Bu bit sınıflarının açıklamaları Tek Bitlik Değişim (TBD) (İngilizce: Single Event Upset – SEU) modeli ile yapılmıştır.

4.1.1 Sıradan Veri

Anlık değerler ile birlikte yazmaç dosyasının kontrol bitleri dışındaki bütün bitleri sıradan veri grubuna dâhildir. Bu bitlerin herhangi birisindeki değişim ile birlikte oluşan yeni değerin eski değerden Hamming uzaklığı her zaman 1 olur. Bu verilerden birisi, Şekil 4.3'teki gibi işlem birimine giden kaynak değerlerinden biri olursa, bir bit hatalı olarak yürütülürler.



Şekil 4.3. Sıradan bir verideki hatanın etkisi

Birçok işlem tek bitlik hatayı maskeleyebilir ve hata buyruğa bağlı olarak hesaplanan sonuca etki etmeyebilir. Eğer işlem biriminin sonucunda bir hata gözlenmişse, bu hata, yazmaç dosyasına yazılacak ve bu yazmacı kaynak olarak kullanan diğer buyruklara sıçrayacaktır.

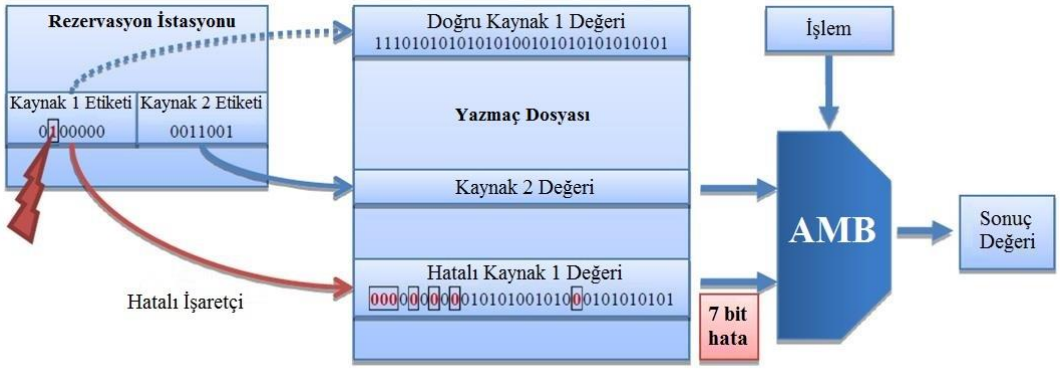
4.1.2 İşlem Kodu

İşlem kodu işlemciye derleyici tarafından gönderilir ve işlemci işlem koduna bakarak gerekli buyrukları oluşturur. Buyruk kümesi tasarımına bağlı olarak, bazı işlem kodları benzer operasyonları gerçekleştiriyor olabilir. Örneğin Alpha buyruk kümesinde, 14.02A ile 14.7EB arasındaki işlem kodları karekök alma işlemini farklı şekillerde yapmaktadır. Dolayısıyla işlem kodundaki bir bitlik değişim bazen tamamen maskelenmekte ve hataya neden olmamaktadır.

4.1.3 Kaynak Belirteçleri

İşlemci bileşenlerinin bazılarında direk veri tutmak yerine verinin saklandığı yer tutularak işaretçi görevini gören alanlar olabilir. Örneğin Rezervasyon İstasyonları'nda (Rİ), yürütme öncesinde girdilerin okunabilmesi için fiziksel yazmaç belirteçleri saklanır. Ayrıca Yeniden Adlandırma Tablosu da buyruklar arasındaki bağımlılığı takip edebilmek için bu belirteçleri kullanır.

Şekil *4.4*'te Rİ'de bulunan 7 bitlik fiziksel yazmaç belirtecinde (toplam 128 yazmacı adresleyebilir) oluşabilecek bir bitlik hatanın etkisi gösterilmiştir. Hatanın yerine bağlı olarak, kaynak etiketi 1 yedi farklı hatalı yazmaca işaret edebilir. Kaynak değer hatalı bir yazmaçtan okunacağı için doğru kaynak değeri ile okunan kaynak değeri arasındaki fark yüksek olacaktır. Yazmaç öbeği Hata Düzeltme Kodu (İngilizce: Error Correction Code – ECC) ile korunuyor olsa bile, yazmaçtaki değer kendi içerisinde tutarlı olacaktır ancak belirteçte bir hata olduğunu öngöremeyeceğinden herhangi bir düzeltme yapılamayacaktır.



Şekil 4.4. Kaynak yazmacı belirtecindeki bir hatanın etkisi

Görüldüğü üzere, kaynak belirteçlerinin hassasiyete etkisi, sıradan verilerin etkisinden daha yüksektir. Bu durum önbelleklerin etiket kısmının güvenilirlik açısından veri alanına göre daha önemli olmasına benzemektedir. Adres etiketindeki bir hata sonucu yanlış bir veri okunabilir veya ıska durumu oluşabilir. Yazmaç belirteçleri içinse bu durum önbelleklere göre daha vahim

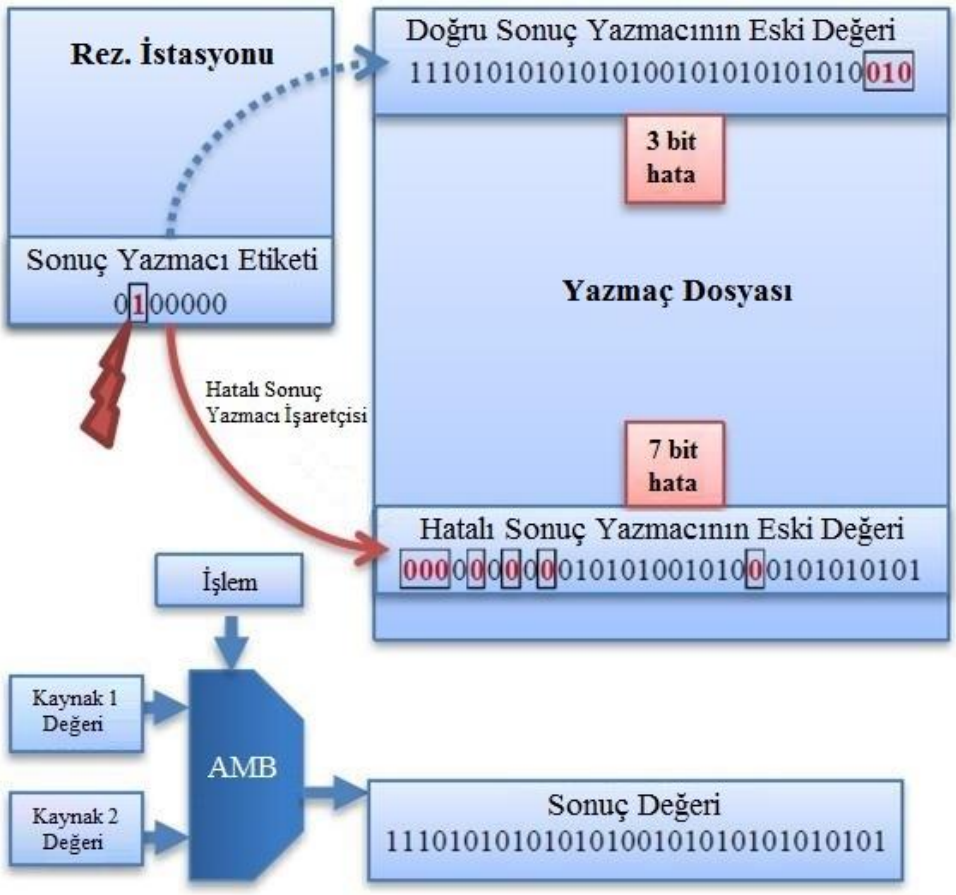
sonuçlar doğurabilir çünkü belirteçte oluşan hata sonrası yazmaç dosyasına erişim mutlaka olacaktır ve yanlış yazmaç okunacaktır.

4.1.4 Sonuç Belirteçleri

Bir buyruğun yürütülmesi sonrasında yeni bir değer hesaplanmışsa, bu değer her zaman sonuç belirteci ile gösterilen konuma yazılır. Ayrıca veri saklayan her yapının dizin değerleri de birer sonuç belirtecidir. Örneğin çöz aşamasında her bir buyruğa YSA'daki dizinini belirten bir numara atanır ve buyruk çalıştıktan sonra bu numara kullanılarak YSA dizinine erişilip gerekli güncellemeler yapılır.

Bir sonuç belirtecinde oluşan hata, aynı anda iki farklı konumda birden hata oluşmasına neden olur: (1) Asıl güncellenmesi gereken konumun değeri değiştirilmediğinden hatalı olur ve (2) yanlış bir konumun değeri değiştirildiğinden olası okuma durumlarında hatalı değer üzerinden işlem yapılır.

Şekil 4.5'te yüklü bir parçacığın, rastgele bir sonuç yazmacı belirtecine çarptığında oluşabilecek bir durum örnek olarak verilmiştir.



Şekil 4.5. Sonuç yazmacı belirtecindeki bir hatanın etkisi

Buyruğun yürütülmesi sonucunda elde edilen sonuç değeri doğru yere yazılmamıştır. Dolayısıyla güncellenmesi gereken en anlamsız üç bit değişmemiştir ve üç bitlik bir hataya neden olmuştur. Aynı zamanda değiştirilmemesi gereken başka bir yazmaç üzerine sonuç değeri yazılmış ve 7 bitlik bir hataya yol açmıştır. Sonuç olarak sonuç yazmacı belirtecinde oluşan bir bitlik hata, yazmaç dosyasında toplam 10 bitlik hataya dönüşmüştür. Sonuç belirteçlerinde oluşabilecek bir hata iki farklı konumu etkilediğinden, bu alanlar sıradan veri ve kaynak belirteçlerine göre daha hassastırlar.

4.1.5 Kontrol Bilgileri

Sıradan veriler ve belirteçler dışında, birçok işlemci bileşeninde bulunan bazı bitlerin durumlarını göstermek için başka bit değerleri tutulur. Örneğin, bir konumda tutulan değerin geçerliliğini belirtmek için geçerli biti (İngilizce: valid bit) kullanılır. Bu bit üzerine çarpan bir parçacık, o satırdaki bütün bilgilerin

kaybolmasına neden olabilir. Kontrol bitinin atandığı konuma göre ne kadar veri kaybı olacağı değişkenlik gösterir. Örnek olarak 64 bitlik değer tutan bir yazmacın geçerli bitinde oluşacak bir hata, yazmaçta bulunan bütün 64 bitin de geçersiz olmasına neden olacaktır ve hatanın etkisi hemen görülecektir. Kontrol bitleri mikro mimariye göre değişkenlik göstereceğinden ve kullanıldığı yapıya göre farklı işlevlere sahip olacağından bu tez kapsamında kontrol bitleri ile ilgili herhangi bir çalışma yapılmamıştır.

4.1.6 Sistem Bitleri

İşlemciler, üzerlerinde çalıştırılan işletim sistemi ile iletişimde olabilmek için çeşitli mimari durumlar tutar. İşlemcide olağandışı bir durum olduğunda kontrolü işletim sistemine devretmek için gerekli bilgiler bazı bitler içerisinde tutulur. Bu bitler üzerine yüklü bir parçacık çarparsa, aslında olmayan bir durum gerçekleşecek ve işletim sistemi gerek olmadığı halde devreye girecektir. Sistem bitleri, bir grup veri yerine çalışan tüm sistemi etkilediğinden, bahsedilen bütün bit sınıflarından daha önemlidir. Benzer şekilde, mikro mimariye göre değişkenlik gösterdiğinden bu çalışmanın kapsamı dışındadır.

4.2 Bit Etki Katsayısı (BEK)

Bir bitin, işlem biriminin sonucunu ne kadar hassasiyette etkilediğini belirlemek için Bit Etki Katsayısı (BEK) tanımı önerilmiştir. BEK hesaplamak için öncelikle bir bit hata ile elde edilen sonuç ile hatasız sonuç arasındaki Hamming uzaklığı hesaplanır. Sonrasında bu uzaklık sonucun toplam bit sayısına bölünür. BEK, bitte meydana gelecek hatanın maskelenebilme olasılığını belirtir (düşük BEK, yüksek maskelenebilme olasılığı). Dolayısıyla BEK, MHK hesabında çeşitli ayarlamalar yapmak için kullanılabilir.

Sıradan verilerin BEK değerini hesaplarken, sıradan veri içeren bir buyruk, her seferinde sadece bir bite hata verilerek bütün olası ihtimaller oluşturularak çalıştırılır (64 bitlik bir makinede 64 farklı olası hata için 64 kere çalıştırılır). Daha sonra elde edilen tüm hatalı sonuçların doğru sonuca ortalama ne kadar Hamming uzaklığında olduğu hesaplanır.

Kaynak yazmacı belirteçleri için BEK hesabı da sıradan veriler için kullanılan yönteme benzemektedir. Belirteçte oluşabilecek olası tüm tek bitlik hatalarda okunacak yazmaçlar ile elde edilen hatalı sonuçların, doğru sonuçtan ortalama Hamming uzaklığı, o kaynak yazmacı belirtecinin BEK değerini verir. 128 yazmaçlı bir işlemcide yazmaç belirteçleri 7 bitle gösterilir, dolayısıyla yedi farklı hatalı belirteç ihtimali vardır. Örneğin bir buyruk R1 yazmacını okuyorsa, bu buyruk için R0, R3, R5, R9, R17, R33 ve R65 yazmaçlarından okuma yapılarak ayrı ayrı sonuçlar hesaplanır. Sonrasında bu hatalı sonuçların, doğru sonuçtan ortalama kaç Hamming uzaklığında olduğu ölçülür.

Sonuç yazmacı belirteçleri için iki farklı Hamming uzaklığı hesaplamak gerekir: (1) buyruğun yürütülmesinden elde edilen sonucun, hatasızken yazılması gereken yazmacın eski değerine olan Hamming uzaklığı ve (2) olası hatada sonucun yazılabileceği yazmaçlardaki değerlerin doğru sonuca olan ortalama Hamming uzaklığı. Bu iki uzaklık değerleri toplandığında sonuç yazmacı belirteçleri için BEK değeri elde edilir.

4.3 MHK ile BEK Arasındaki İlişki

Bir biti MHK gibi sadece hassas veya değil şeklinde sınıflandırmak yerine $MHK_{ağırlıklı}$ tanımı yapılarak bitin hassasiyet seviyesi belirtilebilir. Bu bölümde $MHK_{ağırlıklı}$ hesaplanması gösterilmektedir. MHK, herhangi bir yapıdaki bir bit hata oluşması durumunda sistemin çökme ihtimalini vermekteydi. Benzer şekilde, bir yapının bir bit hata oluşsa bile düzgün sonuç verebilme ihtimali de Mimari Güvenilirlik Katsayısı (MGK) olarak tanımlanabilir. MGK Denklem 4.1'deki gibi hesaplanabilir.

$$MGK = 1 - MHK \tag{4.1}$$

BEK, bir bit hatanın buyruğun sonucuna ne kadar etki ettiğini belirmekte ve Denklem 4.2'deki gibi hesaplanmaktadır.

$$BEK = \frac{Hatalı\ Sonuçların\ Doğru\ Sonuca\ Olan\ Ortalama\ Hamming\ Uzaklığı}{Yürütme\ Sonrasında\ Sonucun\ Toplam\ Bit\ Sayısı} \tag{4.2}$$

BEK, bir hatanın yürütme sonucunda hata üretme ihtimalini belirtmekteydi. Buradan BEK ile hesaplanan güvenilirlik katsayısı $MGK_{ağırlıklı}$ Denklem 4.3'teki gibi hesaplanabilir.

$$MGK_{ağırlıklı} = MGK\ x\ (1 - BEK) \tag{4.3}$$

Son olarak;

$$MHK_{ağırlıklı} = 1 - \ MGK_{ağırlıklı} \tag{4.4}$$

Buradan;

$$MHK_{ağırlıklı} = 1 - (1 - MHK)x\ (1 - BEK) \tag{4.5}$$

Sistemde oluşan bir bitlik hata, sonraki buyruklar tarafından maskelenebilir. Bu çalışmada önerilen temel kavram ise bir hatanın yürütülen buyruğun sonucuna etkisi ne kadar büyük olursa, maskelenme ihtimali de o kadar az olur. Dolayısıyla MHK hesabına BEK değerini de ekleyerek $MHK_{ağırlıklı}$ farklı bitler arasında daha dengeli ve adil bir karşılaştırma yapılabilir.

5. DENEYSEL SONUÇLAR

SPEC 2006 sınama programlarının [25] yazmaç değerleri, kaynak/sonuç yazmacı belirteçleri ve işlem kodlarının Bit Etki Katsayısı (BEK) değerlerini gözlemlemek için MSIM mikro mimari benzetimcisi [26] kullanılmıştır. Benzetimler esnasında Tablo 5.1'de verilen değerler ile benzetimci ortamı oluşturulmuştur. Programların MHK değerleri de ölçülerek BEK ile birlikte $MHK_{ağırlıklı}$ değerleri hesaplanmıştır. Sonrasında MHK ve $MHK_{ağırlıklı}$ değerleri, hata verme (İngilizce: fault injection) yöntemi ile elde edilen hassasiyet değeri ile karşılaştırılmıştır.

Tablo 5.1. Benzetimci ortamının özellikleri

Değişken	**Belirlenen Değerler**
Makine genişliği	4 buyruk getirme, yayınlama, sonlandırma
Aralık boyutu	80 girdili Yükle/Sakla kuyruğu, 128 girdili YSA, 128 yazmaçlı FYD
İşlem birimleri	2 Aritmetik birimi, 2 kayan nokta birimi, 1'er adet tam sayı ve kayan nokta çarpıcı
Seviye 1 Buyruk ve Veri Önbellekleri	128 KB, 4 yollu kümeli ilişkili, 64 bayt satır, 1 çevrimde erişim
Seviye 2 Birleştirilmiş Önbellek	256 KB, 16 yollu kümeli ilişkili, 64 bayt satır, 6 çevrimde erişim

Hata verme yöntemi uygulanırken, her program için işlemcinin farklı yapılarına 100'er hata verilmiştir. Bu yöntem için de MSIM kullanılmıştır. Program düzeyinde hata verme yönteminin kapı seviyesinde (İngilizce: Register Transfer Level – RTL) hata verme yöntemine nazaran düşük kesinlikte olduğu ispatlanmıştır [27]. Bu çalışmada kapı seviyesi yerine mikro mimari benzetimci düzeyinde hata verme yöntemi uygulanmıştır. Bunun sebebi MHK değerinin hesaplanmasında mikro mimari benzetimci kullanıldığından adil bir karşılaştırma yapabilmektir.

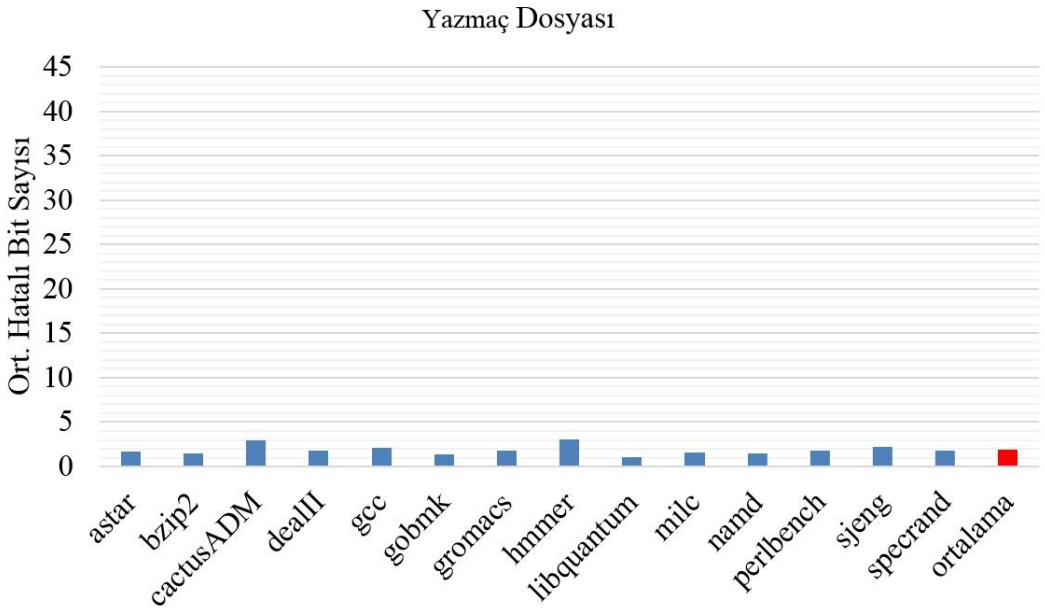
Deneylerde oluşturulan her bir hata için ilgili yapılarda tüm olası hatalar da oluşturulup çalıştırıldı. Daha sonra yürütme sonucunda oluşan tüm hatalı sonuçlar ile hatasız sonuç arasındaki ortalama farklı bit sayıları hesaplandı. Bu değer Tek Bitlik Değişim (TBD) durumunda ilgili bit tipinin sistemin hassasiyetine yaptığı etkiyi göstermektedir.

Şekil 5.1, yazmaç verilerinde oluşabilecek bir bitlik hatanın işlem birimine girdiğinde, sonuç üzerinde ortalama 2 bitlik hataya neden olduğunu göstermektedir. Bu sonuçlar [28] çalışmasında elde edilen, bir bitlik hatanın tamsayı işlem birimlerinin sonucunda yaklaşık %90 oranında (hatanın tipine göre değişebilir) yine bir bitlik farklılık oluşturduğu sonucuna yakındır.

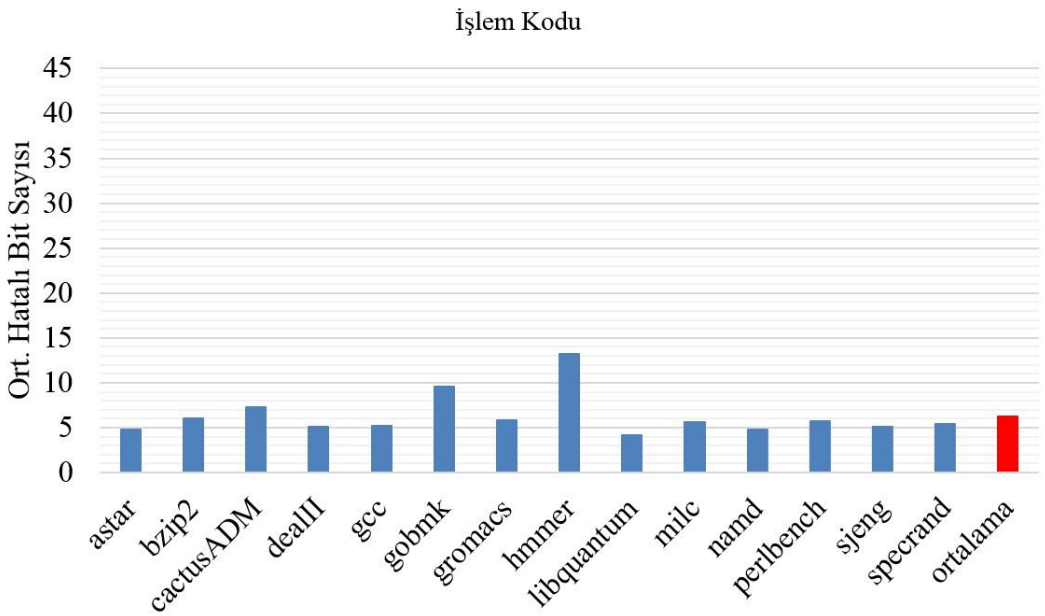
İşlem kodları için elde edilen sonuçlar Şekil 5.2'de verilmiştir. Bir bitlik hatanın sonuca etkisi ortalamada yaklaşık olarak 6 bitlik farklılığa neden olmaktadır. Bu sonucun mikro mimari tasarımına ve buyruk kümesine bağlılığının yüksek olduğu unutulmamalıdır.

Şekil 5.3'te kaynak yazmacı belirteçlerinin ortalama Hamming uzaklığı değerinin 16 bit olduğu (64 bitlik Aritmetik Mantık Birimi - AMB kullanan bir işlemcide) görülmektedir. Kaynak yazmacı belirteçlerinin sistemin hassasiyetine sıradan verilere göre 8 kat daha fazla etki ettiği anlaşılmaktadır.

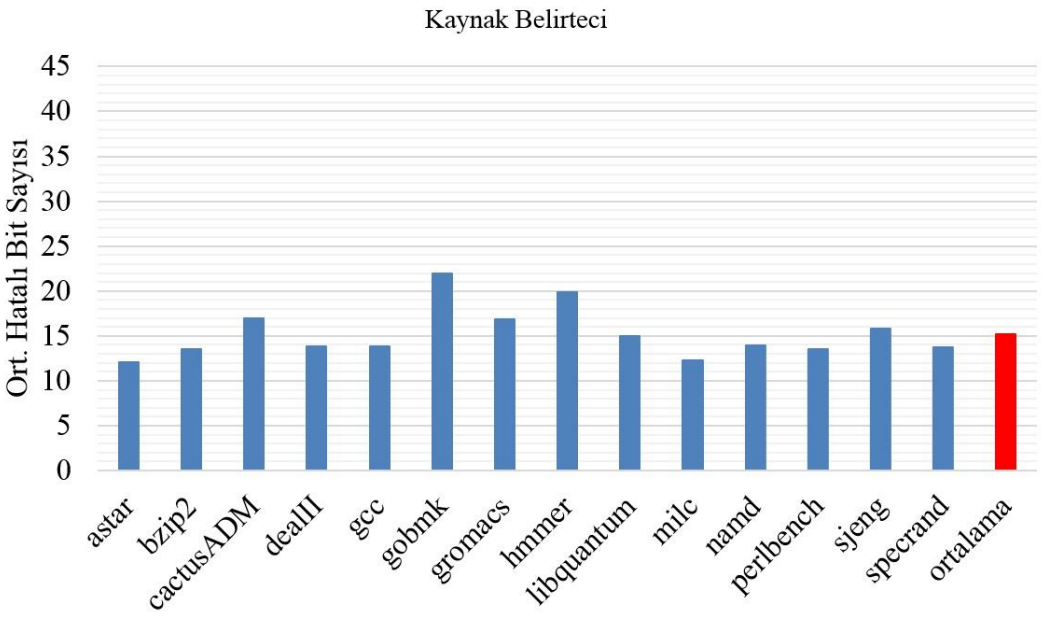
Sonuç yazmacı belirteçleri için iki farklı uzaklık ölçülmelidir: (1) Doğru sonuç yazmacındaki eski değer ve yeni üretilen sonuç arasındaki uzaklık ve (2) üretilen sonuç ile olası diğer yanlış sonuç yazmaçlarındaki değerler arasındaki ortalama uzaklık. Bu iki uzaklığın toplamı sonuç yazmacı belirteçleri için BEK değerini vermektedir. Bu değer Şekil 5.4'de görüldüğü üzere sıradan verilere nazaran 14 kat daha fazladır ve hassasiyete etkisi de diğer tipteki bitlerden daha yüksektir.



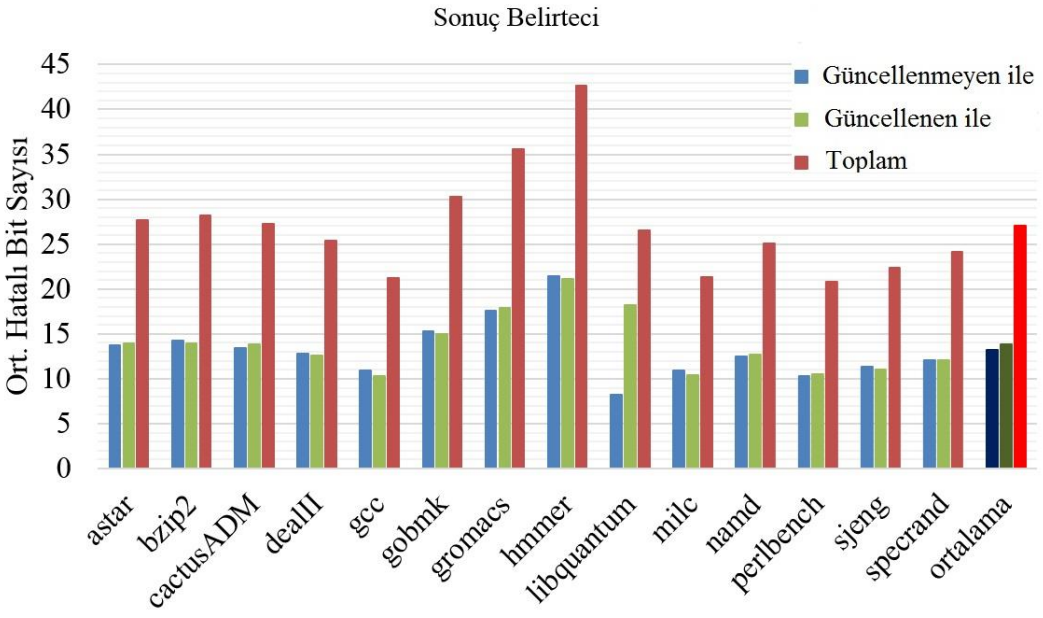
Şekil 5.1. Sınama programlarının yazmaç dosyası için ortalama Hamming uzaklık değerleri



Şekil 5.2. Sınama programlarının işlem kodları için ortalama Hamming uzaklık değerleri

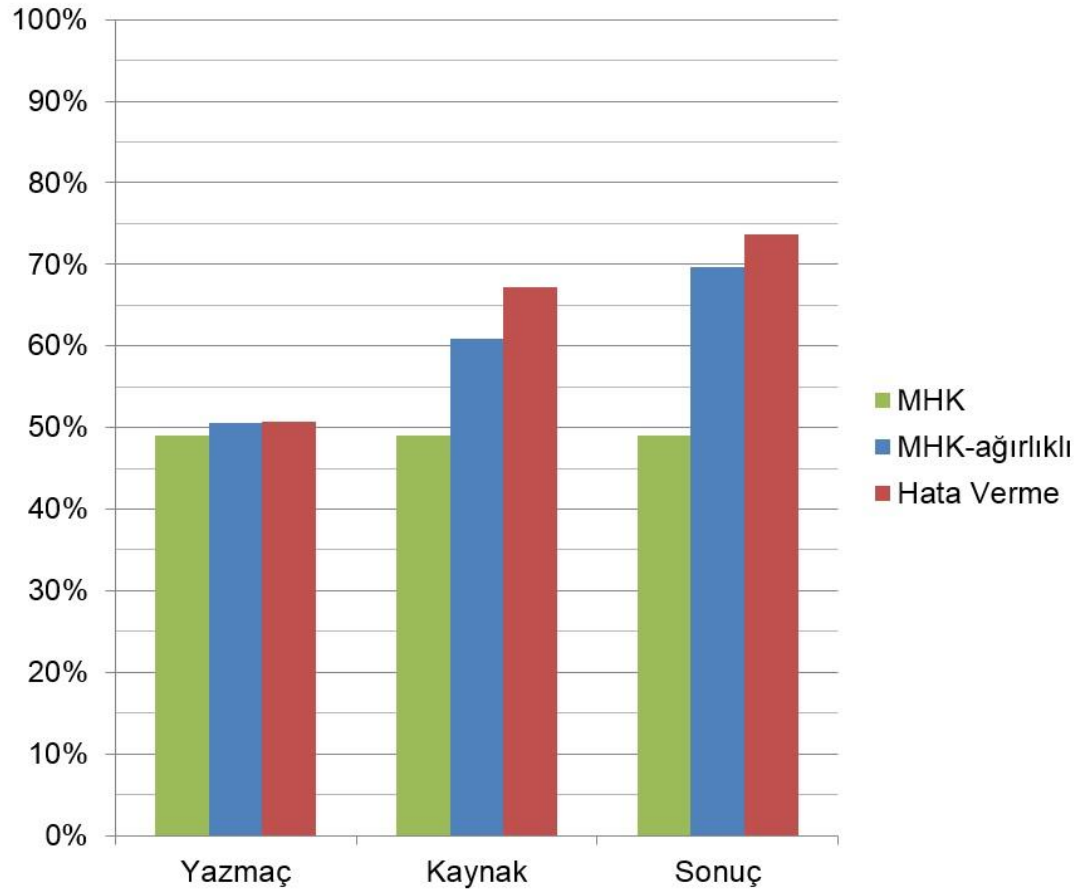


Şekil 5.3. Sınama programlarının kaynak yazmacı belirteçleri için ortalama Hamming uzaklık değerleri



Şekil 5.4. Sınama programlarının sonuç yazmacı belirteçleri için ortalama Hamming uzaklık değerleri

Şekil 5.5'de kaynak belirteçleri, sonuç belirteçleri ve yazmaç değerleri (sıradan veri) için ortalama MHK ve $MHK_{ağırlıklı}$ değerleri sunulmuştur. Ayrıca bu yapılar için hata verme yöntemiyle elde edilen hassasiyet seviyeleri de gösterilmiştir. Hata verme yöntemi donanım birimlerinin hassasiyet seviyesini diğer yöntemlere nazaran daha isabetli vermektedir ancak benzetim süreleri oldukça uzundur. MHK değeri ölçümleri, çalıştırılan buyruklar boru hattının yürütme aşamasındayken hata verilerek alınmıştır. Şekil 5.5 göstermiştir ki yazmaç belirteçleri (kaynak veya sonuç) yazmaçlarda saklanan verilerden daha hassastır. Ayrıca BEK değerini MHK hesabına eklediğimizde ($MHK_{ağırlıklı}$) elde edilen sonuçlar hata verme yönteminden elde edilenlere yakınlaşmaktadır.



Şekil 5.5. Sınama programlarının farklı tipteki veriler için ortalama MHK, $MHK_{ağırlıklı}$ ve hata verme yöntemi ile elde edilen hassasiyet seviyeleri

6. SONUÇ

Teknolojinin ilerlemesi ile birlikte tümleşik devrelere daha küçük boyutlarda ve daha çok sayıda transistör koymak mümkün hale geldi. Hem transistör sayısındaki artış, hem de yüksek saat sıklığında çalışmaları nedeniyle işlemcilerin tükettiği enerji de artmaktadır. Bu enerji tüketimini azaltmak için transistörlerin çalışma gerilimleri düşürülmektedir. Ancak bu durum, çeşitli yüklü parçacıkların çarpmasıyla meydana gelen geçici hataların oluşma ihtimalini yükseltmiştir. Günümüzde analog dünyada sayısal dünyaya geçiş arttığından, sayısal devrelerin güvenilirliği önem kazanmıştır. Sayısal sistem tasarımcıları, olası hatalar için önlem almalı ve tasarımlarını güvenilir hale getirmelidirler.

Sayısal sistemlerin güvenilirliğini belirleme konusunda yaygın olarak Mimari Hassasiyet Katsayısı (MHK) kullanılmaktadır. Bu değer, sistemlerin geçici hatalara karşı ne kadar hassas olduğunu belirtmektedir. MHK analiz yöntemine göre sistemdeki bitler sonuca etki eder veya etmez olarak iki sınıfa ayrılmaktadır. Bir bitin hassasiyet düzeyi konusunda herhangi bir bilgi vermemektedir.

Bu tez çalışmasında, sistemde bulunun bütün bitlerin, sistemin hassasiyetine aynı oranda etki etmediği savunulmuştur. İşlemcinin bir bileşeninde oluşabilecek tek bitlik hatanın, yürütme biriminden geçtikten sonra sonuçta ne kadarlık bir hata oluşturduğunu belirten Bit Etki Katsayısı (BEK) tanımlanmıştır. Tanımlanan BEK değeri, MHK hesaplanmasında kullanılarak $MHK_{ağırlıklı}$ değeri üretilmiştir. $MHK_{ağırlıklı}$ ile farklı veri türleri için hassasiyet ölçümleri yapılmıştır. Yapılan ölçümler MHK ve hata verme yöntemi ile edilenler ile karşılaştırılmıştır. Elde edilen sonuçlarda $MHK_{ağırlıklı}$ değerlerinin, genelde üst hassasiyet sınırı olarak kabul edilen hata verme yöntemi ile elde edilen değerlere yaklaştığı görülmektedir. Ayrıca, $MHK_{ağırlıklı}$ ile yapılan hassasiyet karşılaştırmalarının farklı veri türleri için daha adil olduğu düşünülmektedir.

KAYNAKLAR

[1] R. Baumann, Soft errors in advanced computer systems, *IEEE Design and Test of Computers*, 22(3): 258 - 266, 2005.

[2] J.F. Ziegler, H.W. Curtis, H.P. Muhlfeld, C.J. Montrose, B. Chin, IBM experiments in soft fails in computer electronics (1978 - 1994). *IBM Journal of Research and Development*, 40(1): 3 - 18, 1996.

[3] S. Borkar, Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *IEEE Micro*, 25(6): 10 - 16, 2005.

[4] S.S. Mukherjee, C. Weaver, J. Emer, S.K. Reinhardt, T. Austin, A systematic methodology to compute the architectural vulnerability factors for a highperformance microprocessor. *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, 29 - 41, 2003.

[5] X. Li, S.V. Adve, P. Bose, J.A. Rivers, Architecture-level soft error analysis: examining the limits of common assumptions. *Proceedings of International Conference on Dependable Systems and Networks*, 266 - 275, 2007.

[6] D.A. Patterson, J.L. Hennessy, Computer Organization and Design The Hardware/Software Interface, *Morgan Kaufmann*, 2014.

[7] J.E. Smith, G.S. Sohi, The microarchitecture of superscalar processors. *Proceedings of the IEEE*, 83(12): 1609 - 1624, 1995.

[8] J. Kahle, B. Sinharoy, W. Starke, S. Taylor, S. Weitzel, S.G. Chu, S. Islam, V. Zyuban, The implementation of POWER7TM: A highly parallel and scalable multi-core high-end server processor. *Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 7(11): 102 - 103, 2010.

[9] S.D. KNaffziger, G. Hammond, The implementation of the next- generation 64 b ItaniumTM microprocessor. *Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 1 : 344 - 472, 2002.

[10] R. M. Tomasulo, An Efficient Algorithm for Exploiting Multiple Arithmetic Units. *IBM Journal of Research and Development*, 25 - 33, 1967.

[11] N.J. Wang, A. Mahesri, S.J. Patel, Examining ace analysis reliability estimates using fault-injection. *Proceedings of the 34th Annual International Symposium on Computer Architecture*, 460 - 469, 2007.

[12] A. Biswas, P. Racunas, J. Emer, S. Mukherjee, Computing accurate AVFs using ACE analysis on performance models: a rebuttal. *IEEE Computer Architecture Letters*, 7(1): 21 - 24, 2008.

[13] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S.S. Mukherjee, Computing architectural vulnerability factors for address-based structures. *Proceedings of the International Symposium on Computer Architecture - ISCA*, 532 - 543, 2008.

[14] A. Biswas, C. Recchia, S.S. Mukherjee, V. Ambrose, L. Chan, A. Jaleel, et al. Explaining cache SER anomaly using DUE AVF measurement. *International Symposium on High Performance Computer Architecture*, 1 - 12, 2010.

[15] V. Sridharan, D.R. Kaeli, Eliminating microarchitectural dependency from architectural vulnerability. *International Symposium on High Performance Computer Architecture*, 117 - 128, 2009.

[16] J. Lee, A. Shrivastava, Static analysis to mitigate soft errors in register files. *Proceedings of the Conference on Design, Automation and Test in Europe*, 1367 - 1372, 2009.

[17] V. Sridharan, D.R. Kaeli, Using hardware vulnerability factors to enhance AVF analysis. *Proceedings of the 37th Annual International Symposium on Computer Architecture*, 461 - 472, 2010.

[18] X. Fu, T. Li, J.A.B. Fortes, Soft error vulnerability aware process variation mitigation. *International Conference on High-Performance Computer Architecture*, 93 - 104, 2009.

[19] J.W. Sheaffer, D.P. Luebke, K. Skadron, The visual vulnerability spectrum: characterizing architectural vulnerability for graphics hardware. *Proceedings of the 21st ACM SIGGRAPH/Eurographics Symposium on Graphics Hardware*, 9 - 16, 2006.

[20] L. Duan, B. Li, L. Peng, Versatile prediction and fast estimation of architectural vulnerability factor from processor performance metrics. *International Conference on High-Performance Computer Architecture*, 129 - 140, 2009.

[21] X. Li, S.V. Adve, P. Bose, J.A. Rivers, Online estimation of architectural vulnerability factor for soft errors. *Proceedings of the 35th Annual International Symposium on Computer Architecture*, 341 - 352, 2008.

[22] K.R. Walcott, G. Humphreys, S. Gurumurthi, Dynamic prediction of architectural vulnerability from microarchitectural state. *Proceedings of the 34th Annual International Symposium on Computer Architecture*, 516 - 527, 2007.

[23] A.A. Nair, S. Eyerman, L. Eeckhout, L.K. John, A first-order mechanistic model for architectural vulnerability factor. *Proceedings of the 39th Annual International Symposium on Computer Architecture*, 273 - 284, 2012.

[24] A.A. Nair, L.K. John, L. Eeckhout, AVF stressmark: towards an automated methodology for bounding the worst-case vulnerability to soft errors. *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, 125 - 136, 2010.

[25] J.L. Henning, SPEC CPU2006 benchmark descriptions. *SIGARCH Computer Architecture News*, 34 , 1 - 17, 2006.

[26] J.J. Sharkey, D. Ponomarev, K. Ghose, M-sim: A Flexible, Multithreaded Architectural Simulation Environment. *Technical Report* CS-TR-05-DP01, Department of CS, SUNY – Binghamton, 2005.

[27] H. Cho, S. Mirkhani, C.Y. Cher, J.A. Abraham, S. Mitra, Quantitative evaluation of soft error injection techniques for robust system design. *Proceedings of the 50th Annual Design Automation Conference*, 101:1 - 101:10, 2013.

[28] M.L. Li, P. Ramachandran, U.R. Karpuzcu, S.K.S. Hari, S.V. Adve, Accurate microarchitecture-level fault modeling for studying hardware faults. *HPCA*, 105 - 116, 2009.

ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, adı : CAN, Serdar Zafer

Uyruğu : T.C.

Doğum tarihi ve yeri : 30.08.1990 Yenimahalle

Medeni hali : Bekar

Telefon : +90 (538) 316 00 26

e-mail : serzafcan@gmail.com

Eğitim

Derece	Eğitim Birimi	Mezuniyet tarihi
Lisans	TOBB Ekonomi ve Teknoloji Üniversitesi	2012

İş Deneyimi

Yıl	Yer	Görev
2012-2015	TOBB Ekonomi ve Teknoloji Üniversitesi	Burslu Yüksek Lisans Öğrencisi

Yabancı Dil

İngilizce (Çok iyi)

Yayınlar

Serdar Zafer Can, Gülay Yalçın, Oğuz Ergin, Osman Sabri Ünsal, Adrián Cristal, "Bit Impact Factor: Towards making fair vulnerability comparison". *Microprocessors and Microsystems - Embedded Hardware Design*, 38(6): 598-604, 2014

Pedro Reviriego, Serdar Zafer Can, Çağrı Eryılmaz, Juan Antonio Maestro, Oğuz Ergin, “Exploiting processor features to implement error detection in reduced precision matrix multiplications”. *Microprocessors and Microsystems – Embedded Hardware Design*, 38(6): 581-584, 2014