TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

İŞLEMCİ YAZMAÇLARININ GÜÇ VE GÜVENİLİRLİK AÇISINDAN VERİMSİZLİĞİNİN ENGELLENMESİ

DOKTORA TEZİ

Abdulaziz EKER

Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Doç. Dr. Oğuz ERGİN

HAZİRAN 2016

Fen Bilimleri Enstitüsü Onayı

.....
Prof. Dr. Osman EROĞUL
Müdür

Bu tezin Doktora derecesinin tüm gereksinimlerini sağladığını onaylarım.

.....
Prof. Dr. Murat ALANYALI
Anabilimdalı Başkanı

TOBB ETÜ, Fen Bilimleri Enstitüsü'nün 111117704 numaralı Doktora öğrencisi **Abdulaziz EKER**'in ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"İŞLEMCİ YAZMAÇLARININ GÜÇ VE GÜVENİLİRLİK AÇISINDAN VERİMSİZLİĞİNİN ENGELLENMESİ"** başlıklı tezi 29.06.2016 tarihinde aşağıda imzaları olan jüri tarafından kabul edilmiştir.

Tez Danışmanı:	**Doç. Dr. Oğuz ERGİN** TOBB Ekonomi ve Teknoloji Üniversitesi	.....
Jüri Üyeleri:	**Doç. Dr. Özcan ÖZTÜRK (Başkan)** İ. Doğramacı Bilkent Üniversitesi	.....
	Yrd. Doç. Dr. A. Murat ÖZBAYOĞLU TOBB Ekonomi ve Teknoloji Üniversitesi	.....
	Doç. Dr. Ali BOZBEY TOBB Ekonomi ve Teknoloji Üniversitesi	.....
	Doç. Dr. Süleyman TOSUN Hacettepe Üniversitesi	.....

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, alıntı yapılan kaynaklara eksiksiz atıf yapıldığını, referansların tam olarak belirtildiğini ve ayrıca bu tezin TOBB ETÜ Fen Bilimleri Enstitüsü tez yazım kurallarına uygun olarak hazırlandığını bildiririm.

Abdulaziz EKER

ÖZET

Doktora Tezi

İŞLEMCİ YAZMAÇLARININ GÜÇ VE GÜVENİLİRLİK AÇISINDAN VERİMSİZLİĞİNİN ENGELLENMESİ

Abdulaziz EKER

TOBB Ekonomi ve Teknoloji Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Doç. Dr. Oğuz ERGİN

Tarih: HAZİRAN 2016

İşlemcinin yazmaç öbeği birçok bileşenle çeşitli aşamalarda etkileşim içinde olduğundan, işlemci veri-yolunun en aktif yapılarından birisidir. Buna rağmen yazmaç öbeği enerji ve veri yedekliliği bağlamında yeterince optimize edilmemiştir. Çalışmalarımızda hedefimiz ilk kısımda yazmaç öbeğinin enerji verimsizliğini optimize etmek ve ikinci kısımda gereksiz yazmaç yedekliliğinden daha yüksek güvenilirlik için istifade etmektir.

Çağdaş mimarilerde yazmaç öbeği işlemcinin en çok enerji harcayan ve en çok kullanılan bileşenlerinden biridir. Bu nedenle yazmaç öbeğinin güç tüketimini azaltmak son derece önemlidir. Bu tezde, değişmeyen bitlere yazmayarak yazmaç öbeğinin enerji sarfiyatını azaltan tasarımlar öneriyoruz. Tasarımlarımız, yazmaç bitlerinin her işlemde ortalama sadece % 10'unun değiştiği gözlemine dayanmaktadır. Çalışmamızda, yazmaç öbeğinin yazma gücünü güncelleme-tabanlı bir yapıyla azaltmak için bu verimsizliği asgariye indiren mimari ve devresel teknikler önermekteyiz. 64-bitlik bir veri-yolunda, yazmaç öbeğinin enerji kaybının bazı gösterge programlar için % 24,85'e kadar ve tümünün ortalamasında % 20,59 seviyesinde azaltılabilmesinin ihmal edilebilir bir başarım kaybıyla mümkün olduğunu göstermekteyiz.

Kozmik parçacıklar veya tümdevrenin paketleme malzemesinden kaynaklı radyasyon nedeniyle oluşan geçici hatalar, giderek önemi artan bir tasarım problemidir. Küçülen

transistör kapı uzunluklarıyla birlikte sırasız çok-yollu boru-hattındaki veri-yolu bileşenleri geçici hatalara karşı daha da yatkın hale gelmektedir. Çağdaş mikro-işlemcilerdeki veri saklayan başlıca yapı olan yazmaç öbeği, araştırmacıların geçici hatalara karşı korumak için çok sayıda tasarım önerdikleri önemli işlemci kısımlarından olmuştur.

Çalışmamıza yazmaç öbeğindeki kayıtlı değerlerin birçoğunun birbiriyle arasındaki Hamming uzaklıklarının çok az olduğu gözlemiyle başladık. Bu analiz sonuçlarını gösterdikten sonra sıfır Hamming uzaklığındaki yazmaç değerlerinin varlığından istifade eden bir geçici hata düzeltme yöntemi önerisi ortaya koyduk. Zaten mevcut olan bu yedekliliği parite korumasıyla birlikte kullanarak kaydedilmiş bir çok değeri düzeltme imkanını yakalamaktayız. Yöntemimizin kapsamasını genişletmek için aynı zamanda aralarındaki Hamming uzaklığı az olan değerler için de koruma sağlamayı öneriyoruz. Sonuçlarımız, yazmaç öbeğinde zaten mevcut olan kopyalardan istifade eden mekanizmaları kullanarak yazmaçların % 20,5'inin güç tüketiminde yalnızca % 2,8 artışla geçici hatalardan korunabildiğini göstermektedir. Bu yönteme aktif yazmaçları boşta olan yazmaçlara kopyalayan bir ilave mekanizmayı da eklersek, % 18,9 artan bir güç tüketimiyle yazmaç öbeğinin koruma kapsamı % 44,1'e yükselir. Kopyalama yerine en alt baytında sadece birkaç biti birbirinden farklı olan değerlerden istifade ederek kapsamayı önemsiz bir güç artışıyla % 39,8'e çıkarmak da mümkündür.

Anahtar Kelimeler: Yazmaç öbeği, Mikroişlemci, SRAM, Geçici hatalar.

ABSTRACT

Doctor of Philosophy

AVOIDING REGISTER FILE INEFFICIENCY IN TERMS OF POWER AND RELIABILITY

Abdulaziz EKER

TOBB University of Economics and Technology
Institute of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Doç. Dr. Oğuz ERGİN

Date: JUNE 2016

Processor register file is one of the most active structures of the processor datapath, interacting with many components in several pipeline stages. However register file has not been well-optimized in terms of energy and data redundancy. In our studies, our goal is to optimize its energy inefficency in the first part and to exploit the inefficient register redundancy for better reliability in the second part.

In modern architectures the register file is one of the most energy consuming and frequently used components of the processor. Therefore, reducing the register file power dissipation is critical. In this thesis, we propose schemes that reduce the energy dissipation of the register file by not writing the bits that are not changed. Our schemes rely on the observation that on the average only 10% of the register bits are changed by the instructions at each operation. In this study, we propose a combination of architectural and circuit level techniques that exploit this inefficiency for the register file's write power reduction using an update-based scheme. We show that for a 64-bit datapath it is possible to reduce the energy dissipation of the register file up to 24.85% for individual benchmark programs and by 20.59% on the average across all simulated benchmarks with a negligible performance compromise.

Soft errors caused by the cosmic particles or the radiation from the packaging material of the integrated circuits are an increasingly important design problem. With the shrinking feature sizes, the datapath components of the out-of-order superscalar

pipeline are becoming more prone to soft errors. Being the major data holding component in contemporary microprocessors, the register file has been an important part of the processor on which researchers offered many different schemes to protect against soft errors.

We start with the observation that many of the stored values inside the register file have very small Hamming distances when compared to each other. After showing this analysis results we propose a soft error correction scheme that makes use of the presence of multiple register values that have zero Hamming distance from each other. We use this already available redundancy along with parity protection to achieve error correction for many of the stored values. We also extend the coverage of our scheme to offer coverage for values that are small hamming distances apart from each other. Our results show that, by employing schemes that make use of the already available copies of the values inside the register file, it is possible to protect 20.5% of the registers from soft errors with an additional power consumption of 2.8%. If we include the extension which duplicates active registers to idle registers to increase redundancy, protection coverage increases to 44.1% of the register file, with an increased power dissipation of 18.9%. Instead of duplicating, with negligible power overhead, it is possible to extend the coverage to 39.8% by exploiting the values that differ only a few bits in their least significant byte.

Keywords: Register file, Microprocessors, SRAM, Soft errors.

TEŞEKKÜR

Çalışmalarım boyunca değerli yardım ve katkılarıyla beni yönlendiren, TOBB ETÜ'ye katılmamda bana destek olan, sayesinde sayısız şeyler öğrendiğim ve kararlarında hiçbir zaman beni unutmayan sayın hocam Dr. Oğuz Ergin'e çok teşekkür ederim.
Değerli hocam Dr. Özcan Öztürk'e ODTÜ'deki çalışmalarımda da TOBB ETÜ'deki çalışmalarımda da hiçbir zaman bana desteğini esirgemediği, zaman ayırdığı ve Tez İzleme Komitemle Tez Jürimde bulunmayı kabul ettiği için çok teşekkür ederim.
Değerli hocam Dr. A. Murat Özbayoğlu'na da hem tez aşamasında hem Tez İzleme Komitesinde bana destek olduğu ve cesaretlendirdiği için çok teşekkür ederim.
Değerli hocalarım Dr. Ali Bozbey ve Dr. Süleyman Tosun'a da zamanlarını ayırıp Tez Jürimde bulunmayı kabul ettikleri için çok teşekkür ederim.
Değerli eşim Züleyha'ya bunca yıldır çalışmalarımda hep yanımda olduğu ve en sıkıntılı zamanlarında bile beni düşündüğü için çok teşekkür ederim. Onun benim için ne çok şey ifade ettiğini anlatmam imkânsız. Canım oğlum Ömer Tarık'a da, henüz bunu okuyamıyor olsa da, hayatının ilk 4 yılında onunla geçirmek isteyip de geçiremediğim zamanlarımı telafi etmeye çalışacağımı söylemek isterim.
TÜBİTAK'ta birlikte çalıştığım ve yeterlilik sınavımı geçmeme benim kadar sevinen dostum Bilgin Vargün'e, ekip arkadaşlarıma ve son olarak da gelmiş geçmiş Kasırga Z10 ekibine çok teşekkür ederim.
TOBB Ekonomi ve Teknoloji Üniversitesi'ne sağladığı araştırma bursu için ayrıca teşekkür ederim.

İÇİNDEKİLER

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

KISALTMALAR

AHK : Aynı Hedef-Kaynak
ARM : Acorn RISC Machine
CPU : Merkezi işlem birimi (Central Processing Unit)
CRC : Döngüsel Artıklık Denetimi (Cyclic Redundancy Check)
DHTTB : Dallanma Hedef/Tahmin Tampon Belleği
EAB : En Alt Bayt
ECC : Hata Düzeltme Kodu (Error Correcting Code)
EDAC : Hata Algılama ve Düzeltme Kodu (Error Detecting And Correcting Code)
FHK : Farklı Hedef-Kaynak
FIFO : İlk Giren İlk Çıkar (First-in First-out)
FIT : 1 milyar saatte arızalanma sayısı (Failure-In-Time of 1 billion hours)
GÜNYET : GÜNcelleme-tabanlı Yazmaç öbEği Tasarımı
KOD : Kopyayla Düzeltme
MIPS : Microprocessor without Interlocked Pipeline Stages
MTTF : Hatalar Arası Ortalama Zaman (Mean Time To Failure)
Rİ : Rezervasyon İstasyonu
ROB : Yeniden Sıralama Belleği (Re-Order Buffer)
SEU : Tek Seferlik Bozulma (Single Event Upset)
SMT : Simultane çok iş-parçacıklı (Simultaneous Multithreading)
SPARC : Scalable Processor ARChitecture
SPEC : Standart Performans Ölçüm Kurumu (Standard Performance Evaluation Corporation)
SRAM : Durağan Rastgele Erişimli Bellek (Static Random Access Memory)
TMR : Üçlü Modüler Artıklık (Triple Modular Redundancy)
UMC : United Microelectronics Corporation
X86 : Intel'in 8086 işlemcisinden beri kullandığı buyruk kümesi mimarisidir
X86-64 : x86 buyruk kümesi mimarisinin 64-bit versiyonudur
XMM : x86 mimarisinin tek-buyruk çok-veri yapısı için ilavesidir
YADED : Yakın Değerlerle Düzeltme
YİM : Yazmaç İkileme Mekanizması
YSB : Yeniden Sıralama Belleği

SEMBOL LİSTESİ

Bu çalışmada kullanılmış olan simgeler açıklamaları ile birlikte aşağıda sunulmuştur.

Simgeler	**Açıklama**
E_R	Yazmaç öbeğinden bir yazmacı okumak için gerekli olan enerji
E_W	Yazmaç öbeğindeki bir yazmaca yazmak için gerekli olan enerji
$E_{W.GUNYET}$	GÜNYET mimarisindeki maskeli yazma mekanizması kullanıldığında harcanan yazma enerjisi
E_{yazma}	Yazma işlemi için harcanan enerji
$E_{yazma-surucusu}$	Yazma sürücüleri üzerinde harcanan enerji
E_{yazma}	Yazma işlemi için harcanan enerji
E_{cevre}	Yazmacın çevresindeki devre üzerinde harcanan enerji
$E_{bit-hatti}$	Yazma bit-hattındaki harcanan enerji
$E_{bit-hcresi}$	Bit-hücresinde harcanan enerji

1. GİRİŞ

İşlemcilerde en sık veri alışverişi yapılan ve işlemcinin her aşamasında aktif olan birim yazmaç öbeğidir. Bu nedenle yazmaç öbeği işlemcinin hem en çok güç tüketimine sebep olan, hem de üzerinde veri hatası olması durumunda işlemci birimlerinde silsile halinde en çok soruna yol açabilecek işlemci elemanlarındandır.

Yazmaç öbeği mevcut sistemlerde kullanılırken eğer etkin bir şekilde kullanılabilir ve belirgin olmayan verimsizlikleri yok edilebilirse güç tüketimi ve hatalara dayanıklılık bağlamında önemli faydalar elde edilebilir. Bu nedenle çalışmalarımızda yazmaç öbeğinin verimsizliğini kullanarak güç tüketimini düşürmeyi ve geçici hatalara karşı dayanıklılığını artırmayı amaçladık.

Mobil kullanımın yaygınlaştığı günümüzde işlemci sıcaklığını düşük tutmak ve güç kaybını azaltmak mühim bir sorundur. Yazmaç öbeği modern çok-yollu işlemcilerde güç kaybına sebep olan en önemli yapılardandır. Dolayısıyla, yazmaç öbeğinin güç tüketimini azaltmak işlemcinin güç tüketimine de önemli katkı sağlayacaktır. Buna karşın, işlemcilerdeki yazmaç öbeğinin enerjisi etkin bir şekilde kullanılmamaktadır.

Yazmaç öbeğindeki işlemlerden yazma için kullanılan enerji okuma için kullanılan enerjiden çok daha fazladır [1] [2] [3] ve ön-yükleme (precharging) ile bit-hattı için harcanan enerji de yazmaç öbeğinin yazma için harcadığı enerjinin neredeyse tamamına tekabül eder [1]. Ne var ki yazmaca yazılan bir değer yazmaç bitlerinin hepsinin değerini değiştirmese de, yazmacın bütün bit hatlarına her seferinde ön-yükleme ve boşaltma (discharging) yapılır. Bu davranıştan kaynaklı verimsizliğin boyutunu anlamak için yaptığımız deneylerde gördük ki bir buyruk yazmaç bitlerinin ortalamada yalnızca % 10'unu değiştirmekte, diğer bitler ise aynı değerde kalmaktadır. Buna karşın yazmaç öbeği her bit için bütün sütunlara ön-yükleme yaparak ve bütün bitler için bit-hatlarına enerji vererek yazmacın tamamında yazma işlemini her seferinde gerçekleştirmektedir.

Biz bu çalışmamızın ilk kısmında, 4. bölümde ayrıntılı anlatılacağı gibi, yazma işlemi sırasında aynı kalan bitlere yazma yapılmasını engelleyerek, yükü tamamen boşaltmanın ve bit-hatlarını sürmenin sebep olduğu enerjiyi azaltmaya çalışıyoruz. Bunun için mimari ve devre seviyesinde tekniklerin kombinasyonuyla güncelleme

temelli bir mekanizma kullanarak yazmaç öbeğinin yazma enerjisini azaltmayı önermekteyiz. Bu tasarımda iki adet yeni mekanizma ve bu mekanizmalarla ortaya çıkan yeni mimariden daha iyi faydalanan bir ilave iyileştirme kullanmaktayız. Bununla beraber, enerji tasarrufuna yönelik bu tasarım için Intel Atom [4] ve ARM Cortex [5] gibi mobil işlemciler için daha çok tercih edilen sıralı (in-order) işlemci mimarisini hedefledik.

Radyasyon veya elektriksel gürültü nedeniyle ortaya çıkan geçici hatalar donanımda kalıcı hasara sebep olmamakla birlikte sistem için kritik elemanlarda gerçekleştiğinde sistemin çökmesine sebep olabilmektedir [6] [7]. Transistör kapı uzunluğunun sürekli azalması, kaynak gerilimlerinin düşürülmesi, eşik gerilim değerinin indirilmesi ve saat frekanslarının sürekli artması sonucu geçici hatalar elektronik aygıtlar için giderek daha fazla sorun oluşturmaktadır.

Yazmaç öbeği, işlemcide en sık kullanılan yapılardan olduğu için, burada gerçekleşen bir hata hızlıca işlemcinin diğer kısımlarına sirayet edebilmekte ve sistem arızalarına sebep olmaktadır. Bu nedenle yazmaç öbeğinin hatalara karşı dayanıklı olması bütün işlemciler için son derece önem arz etmektedir. Buna rağmen işlemci tasarımlarının çoğu yazmaç öbeğinin geçici hatalara karşı korunmasına yönelik bir yöntem gerçekleştirmemişlerdir. Bunun asıl nedeni ise yazmaç öbeğine erişim hızında küçük miktarda bir düşüşün bile işlemci için çok ciddi başarım sorunu oluşturmasıdır.

Biz bu çalışmamızın ikinci kısmında da, 5. bölümde ayrıntılı anlatılacağı gibi, işlemcinin yazmaç öbeğindeki yazmaçları yedeklemesine gerek kalmaksızın, yazmaçlarda zaten bulunan benzerlikleri kullanarak yazmaç öbeği için etkin bir koruma sağlamayı önermekteyiz. Bu benzerliği hata dayanıklılığı için kullanmak üzere iki yöntem sunmaktayız. İlk yöntemimiz yazmaçlarda bulunan tıpatıp benzer değerleri kullanmak üzere tasarlanmış Kopyayla Düzeltme (KOD) ismindeki yöntemdir. KOD yöntemi ayrıca bundan önce başka araştırmacılar tarafından önerilmiş yöntemlerle bir arada kullanılabileceğinden, yöntemin başarımını artırmak için kopyası olmayan yazmaç değerlerini kullanılmayan yazmaçlara kopyalayan bir mekanizmayı [8] da yöntemimize ilave ettik. Yakın Değerlerle Düzeltme (YADED) diye adlandırdığımız ikinci yöntemimiz ise sadece en alt baytı farklı olan yazmaçları kullanarak dayanıklılık sağlamayı amaçlamaktadır.

Yazmaç öbeğinin güvenilirliğini sağlarken en önemli gereklilikler, yazmaç öbeği işlemcinin kritik yolu üzerinde olduğundan zamanda kayba sebebiyet vermemesi ve yazmaç öbeği işlemci gücünün önemli bir kısmına tekabül ettiğinden ekstra güç kaybına sebep olmamasıdır. Önerdiğimiz yöntemler mevcut değerlerin benzerliğini kullandığı için ihmal edilebilir kayıpla güvenilirliği ciddi oranda artırmaktadır.

2. İLGİLİ LİTERATÜR

İşlemcilerde güç tüketimi ve buna bağlı olarak oluşan aşırı ısınma işlemciler için son yıllarda ortaya çıkmış en önemli sorunlardandır [1]. İşlemcideki en aktif yapılardan olan yazmaç öbeğinin [9] güç tüketimini azaltmak da işlemcinin güç tüketimini önemli ölçüde düşüreceğinden, yazmaç öbeğinin etkin enerji kullanımı konusunda da çok sayıda çalışma yapılmıştır [10] [11] [2] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23].

Transistör boyutunun küçülmesi ve yonga üzerinde çok sayıda eleman bulunması nedeniyle etkisini daha çok hissettiren geçici hatalar da işlemci güvenilirliğini tehdit etmektedir [24] [25] [26]. İşlemci içinde oluşan geçici hataların etkisini artırmasına sebep olacak olan yapı da yine işlemcideki bir çok birimle iletişimde olan yazmaç öbeğidir [27] [28] [8]. Bu nedenle yazmaç öbeği üzerindeki geçici hataları düzeltmek konusunda çok sayıda çalışma yapılmıştır [29] [30] [31] [32] [28] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43].

İlgi alanımıza giren bu çalışmalar ilgili alt başlıklarda daha detaylı şekilde ele alınmıştır.

2.1. Yazmaç Öbeğinin Güç Tüketimini Azaltma Konusunda Yapılan Çalışmalar

Yazmaç öbeğinin güç tüketiminde tasarruf sağlamaya yönelik çok sayıda çalışma yapılmıştır. Bu çalışmalarda temel olarak belirli bir kaç yöntem kullanılmıştır. Güç tüketimini azaltmak için yazmaç öbeği devresine yönelik çalışan bazı araştırmacılar optimize olmayan gerilim harcamasını adaptif olarak optimize etmeyi [17] [18], SRAM'deki ön-doldurma aşamasını iyileştirmeyi [2], veya yazmaç öbeklerindeki port sayısını azaltmayı[13] [14] önermişlerdir.

Bazı araştırmacılar ise yazmaçların yazmaç öbeğine yazılmasının işlevsel ilişkisine göre bölümlenmesi [44], tahmin edilmesi [12] veya zamansal ilişkisine bağlı olarak bölümlenmesi [19] [20] ya da hızlı erişilebilir bir önbelleğe konulması [10] [11] için yöntemler önermişlerdir.

Bir diğer grup çalışmada ise yazmaçlarda bulunan verinin tamamının kıymetli olmadığı, yazmacın bazı bitlerinin veya tamamının tutulması/yazılması/okunması için harcanan enerjinin tasarruf edilebileceği iddia edilmiş ve bunlara yönelik yöntemler önerilmiştir [15] [16] [23] [21] [22].

Bizim önerdiğimiz ve tezin ilerideki kısımlarında anlatılacak yöntem ise modern mimarilerde zaten mevcut olan potansiyel güç tasarruf etme imkanını herhangi bir başarım düşüşüne sebep olmadan ve mimaride ihmal edilebilir miktarda değişiklikle kullanmaktadır.

Yazmaç öbeğindeki güç tüketimini düşürmek için devre tasarımına yönelik çalışmalardan bir kısmında yazmaç öbeği devrelerini şartlara göre uyum gösteren bir şekle getirmek önerilmiştir [17] [18]. Yazmaç öbeği tasarımlarında gerilim ve frekans, yongadaki parametre değişikliklerine, ani gerilim düşüşlerine ve sıcaklık değişikliklerine göre veya transistördeki yaşlanmaya bağlı okuma zamanlaması hatası olmaması için en kötü durum baz alınarak ayarlanır. Ancak okuma zamanlamasına dönük hataları oluşturabilecek bu şartlar nadiren oluşur. Bu nedenle [17] çalışmasında zamanlama sınırlarını ve hatalarını devre üstünde algılayan ve buna göre gerilim ve frekansı adapte eden bir tasarım önerilmiştir.

Gerilimi değişebilir kılarak yazmaç öbeğindeki güç tüketimini azaltmayı öneren diğer bir araştırma ise işlemci başarımı için kritik olan buyruklarla ilgili yazmaçları normal gerilim seviyesinde tutmayı, ancak zaman-kritik olmayan buyruklarla ilgili SRAM bit hücrelerini daha düşük gerilimde tutmayı önermişlerdir [18]. Kritik ve kritik olmayan buyrukları da bir gruptan diğerine geçiş gerekmesi halinde değiştirmek için tasarımı buyruk-dizisindeki en eski buyrukları sürekli gözlemlemek üzere yapmışlardır.

SRAM bit hücreleri şeklinde tasarlanan yazmaç öbeğindeki yazma enerjisini bit hücresi devrelerinin tasarımını değiştirerek azaltmayı amaçlayan bir çalışmada da araştırmacılar, yazmaç öbeğindeki ardışık yazmalardaki elektriksel yükü ön-doldurma aşamasını atlayarak yapmayı önermektedirler [2]. Yazmaç öbeğine yapılan ardışık yazmalarda eğer bir sütuna yazılan değer aynıysa ön-doldurma ve birbirinin her zaman tersi olan bit-tellerinden birini tekrar boşaltma yapmanın gereksizliğinden yola çıkarak, ön-doldurmayı iptal etmeyi önermişlerdir. Ardışık yazmalarda eğer satırlara farklı değerler yazılmaktaysa da bu durumda birbirinin tersi olan bit telleri arasında yük paylaşımı yaparak bir bit-telini sadece yarım doldurmaya yetecek kadar enerjiyle ön-doldurma yaparak, buradan yarı-yarıya tasarruf etmeyi önermişlerdir. Bununla birlikte bu yöntem yazmaç öbeğine yazmada önemli miktarda gecikmeye sebep olmaktadır.

Yazmaç öbeğinin fazla enerji harcamasının bir başka sebebi olan çok sayıda portu

olmasının önüne geçme iddiasında bulunan bir çalışmada araştırmacılar, okuma ve yazma işlemleri için ayrı ayrı port sayısını azaltan devresel yöntemler önermişlerdir [13]. Okuma işleminde işlemcilerin gereksiz yere, zaten doğrudan sonucunu aldıkları (bypass) yazmaçları, zamanlamayla alakalı sorunların oluşmaması için okuduklarını gözlemleyen araştırmacılar, bunu engelleyen doğrudan okuma uyarı biti eklenmesini ve ilgili mekanizmayı önermişlerdir. Ancak bu mekanizma bazı durumlarda boru hattının durmasına sebep olmakta, bu sebeple de işlemci başarımını ciddi şekilde düşürmektedir. Yazma işlemlerinde daha az port kullanmak içinse yazmaç öbeğini bölümlere ayırmayı ve bölüm başına da daha az port bulundurmayı önermektedirler. Port sayısını azaltarak yazmaç öbeği enerjisini düşürmeyi amaçlayan bir başka çalışmada ise araştırmacılar okuma portu sayısını azaltmak için kısa süreli geçici bir bellek eklemiş, yazmaç öbeği yerine bu küçük bellekten okuyarak yazmaç öbeği portlarına duyulan ihtiyacı azaltmışlar, aynı zamanda okuma başarımını da bir miktar artırmışlardır [14].

Yazmaç öbeğinin devresel yöntemlerle güç tüketimini azaltmaya dönük çalışmaların yanında yazmaçların işlevsel veya zamansal yakınlığına dönük de çeşitli çalışmalar vardır. Bunlardan biri olan bölümlenmiş yazmaç öbeği tekniğinde temel fikir büyük ve merkezi bir yazmaç öbeği yerine özel işlem birimleri olan daha küçük ve düşük güç tüketimli lokal yazmaç öbekleri bulunmasıdır [44]. Bu sayede her işlem birimi çoğunlukla kendi yazmaç öbeği bölümüyle ilişkili olur. Bunun yanında ilişki içindeki buyrukların aynı yazmaçların üstüne yazması bit değişimlerini de azaltarak bir miktar enerji verimliliği de sağlar. Bu yöntem enerji sarfiyatını dağıtmak konusunda başarılı olsa da, toplam güç tüketimini her zaman azaltamamaktadır.

Yazmaç verilerinin zamansal yakınlığını kullanmaya dönük bir çalışma olan öngörü-tahmin tekniği ise zaten kayıtlı verileri üreten buyrukları tahmin etmeyi önerir [12]. Bu yolla işlemcinin yazmaç öbeğini okuma ihtiyacı kalmaz ve böylece enerji tasarrufu elde edilir. Ancak bahsedilen teknikte, bunun ön-şartı değerlerin yakın zamanlarda oluşması ve elbette doğru tahmin edilebilmesidir. Aksi takdirde, yanlış tahminlerin çoğalması tersi etki yapabilir ve enerji sarfiyatını artırabilir. Tahmindeki doğruluk ise derleyici, yazmaç öbeği yapısı ve işlem karakteristiği gibi birçok parametreyle değişebilir.

Yazmaç öbeğindeki aynı değerlerin oluşmasındaki zamansal yakınlığı kullanan diğer bir çalışmada araştırmacılar yazmaca yazılan bir değerin aynısının kısa bir süre önce üretilen bir başka değerle aynı olduğu ve herhangi bir zamanda yazmaç öbeğinde çok sayıda benzer değerin bulunduğunu fark etmişler [19]. Benzer değerlerden de "0" ve "1" değerlerinin her uygulamada en sık görülen değerler olduğunu gözlemlemişler. Buradan yola çıkarak "0" ve "1" değerlerinin yazmaç öbeğinde yer işgal etmesini

önleyecek ve maliyeti çok düşük bir metod önermişler. Bu metod sayesinde yazma ve okuma miktarlarını azaltmış ve yazmaç öbeğindeki güç sarfiyatını düşürmüşlerdir. Zamansal yakınlığı yalnızca tıpatıp aynı yazmaç değerleri üzerinden değil, aynı zamanda en sağdaki birkaç biti farklı olan yazmaç değerleri üzerinden de kullanan bir çalışmada ise araştırmacılar yazmaç öbeğinin organizasyonunu değiştirerek hem okuma-yazma miktarını azaltmayı hem de yazmaç öbeği boyutunu küçültmeyi, ve bu sayede de güç tüketimini azaltmayı hedeflemişlerdir [20]. Buna mukabil saat döngüsü başına yapılan işlem sayısı, yani işlemci başarımı bu yöntem sebebiyle bir miktar düşmüştür.

Yazmaç öbeğindeki değerlerin zamansal yakınlığından faydalanan yazmaç öbeği önbelleği tasarımı, ancak birden çok saat döngüsünde erişilebilen yazmaç öbeğinin okuma ve yazma hızını artırmak için kullanılmaktaydı [45], [46]. Yazmaç öbeği önbelleğini güç tasarrufu amacıyla kullanan araştırmacılar ise çalışmalarında kısa süre içinde kullanılan yazmaçlar için FIFO şeklinde, nisbeten daha uzun süreli yazmaçlar içinse küçük bir kümeli-ilişkili önbellek şeklinde hafıza birimleri kullanmışlardır [10]. Bu sayede önemli oranda enerji tasarrufu sağlamışlar ancak bir miktar başarım kaybına sebep olmuşlardır. Yazmaç öbeği önbelleğini güç tüketimini azaltmak için kullanan bir başka tasarım ise derleyici seviyesinde bir optimizasyonla, yazmaçları okuyacak buyruk sayısını buyruk kodlarının içine yerleştirerek, yalnızca kullanılacak yazmaçları önbellekte tutmayı önerir [11]. Böylece güç tasarrufunun yanında başarım artışı da sağlar.

Bir grup çalışma ise yazmaçların bir kısmının ya da bir süre sonra tamamının aslında kıymetli olmadığı gözleminden yola çıkmıştır. Bu çalışmalardan biri yazmaçların büyük çoğunluğunun yalnızca alt baytları kullandığı, dolayısıyla dar yapılı olduğu gözlemine dayanarak, yazmaçları dar ve geniş olarak ayırmış ve bu sayede enerji tasarrufu sağlanmasını önermiştir [15]. Bu çalışmada en alt baytı kullanılan yazmaçlar tek bayta yazma okuma yapılması sayesinde önemli enerji tasarrufuna sebep olmuşlardır. Dar yazmaçların yapılan benzetim çalışmalarında oldukça sık görülmesi nedeniyle de yazmaç öbeği enerjisi düşürülmüştür. Benzer motivasyonla yapılan bir başka çalışmada ise 64-bitlik yazmaç öbeğini 32-bitlik yazmaçlar şeklinde gruplayıp, 32-bite sığabilen yazmaçları daha küçük alanda tutarak yazmaç öbeğinin daha etkin kullanılması önerilmiştir [16]. Bu sayede daha küçük yazmaç öbeği kullanılarak enerji tasarrufu sağlanabileceği ifade edilmiştir.

Sıralı işlemcilerde harcanan bit değişiminden kaynaklanan dinamik gücü azaltmaya yönelik bir çalışmada araştırmacılar, her bir yazmaç değerini kıymetli ve kıymetsiz baytlara bölerek ve kıymetsiz baytları okuma, yazma veya flip flopta tutma yapmayarak güçten tasarruf etmeyi önermektedir [23]. Aynı çalışmada yazmaç

verilerinin yanında her saat döngüsünde artan program sayacı için de en alt baytı kıymetli sayan ve eğer elde işlemi olur da diğer baytın da artırılması gerekirse bir sonraki döngüde bu artışı yapan bir yöntem de önerilmiştir. Bu durum başarımı bir miktar azaltsa da bit değiştirme sıklığını önemli ölçüde azaltmaktadır. Bu çalışma benzeri tasarrufları veri ve adres önbellekleri ile aritmetik mantık birimi için de önermiştir. Ancak tanecik düzeyinin (granularite) bayt seviyesinde olması yazmaç öbeğindeki enerji tasarrufu potansiyelini tümüyle kullanamamaya sebep olmaktadır.

Geleneksel yazmaç öbeklerinde, bir yazmaç ancak o yazmacı değiştiren buyruk emekli olduğunda, yani buyruk penceresinden çıktığında boşa çıkabilir. Ancak bazı araştırmacılar tarafından önerilen *yazmacı erken boşa çıkarma* yönteminde yazmacın bir başka buyruk tarafından kullanılmayacağı anlaşılır anlaşılmaz yazmacın tamamı artık kıymetsiz olduğundan yazmaç boşa çıkarılır. Bu yöntemin donanımsal olarak [21] veya derleyiciyi de kullanarak [22] gerçeklenen tasarımlarının hepsi yazmaç öbeğinin daha çok yazmaç tutabilmesini veya aynı miktar yazmaç ihtiyacını daha küçük boyutlu bir yazmaç öbeğiyle karşılamayı hedeflemektedir. Bu sayede yazmaç öbeği için harcanan enerji de etkin olarak kullanılmış olacaktır.

Bahsedilen bütün bu çalışmalarda yazmaç öbeğinde ya detaylı devresel değişiklik yapma, ya da çok sayıda ekstra yapı eklenmesi önerilmektedir. Bölüm 4 kapsamında ayrıntılı olarak anlatılacak olan bizim önerdiğimiz yöntem ise yazmaç öbeğinde zaten bulunan potansiyel verimliliği kullanmaya dönük devresel ve mimari bir tasarım olmakla birlikte minimal düzeyde değişikliğe sebep olmaktadır.

2.2. Yazmaç Öbeğindeki Geçici Hataları Yakalama ve Düzeltme Konusunda Yapılan Çalışmalar

Geçici hataların yazmaç öbeği üzerindeki etkisini ortadan kaldırmak için bir çok araştırma yapılmıştır. Hata düzeltme kodu (Error Correcting Code - ECC) yöntemi verinin içine verideki hatayı düzelten pariteler eklemeye dayanır ve güvenilirliği önceleyen birçok ticari sistemde tek-bit hata düzelten ve iki-bit hata algılayan hata düzeltme kodu kullanılmaktadır [42] [47] [48] [49] [50] [51]. Ancak korumasız bir işlemcinin yazmaç öbeğine hata düzeltme kodu eklemek enerji tüketimini 10 kat artırmaktadır [52]. Bu nedenle diğer bazı güvenilir işlemciler yazmaç öbeğinin hatalara karşı dayanıklılığını artırmak için sadece hata algılamaya yönelik tek-bitlik parite kullanmışlardır [53] [6] [43]. Hata düzeltme kodu, 64-bitlik veri için hatasız durumda 7 bit üretir ve tek-bitlik pariteye göre yaklaşık 4 kat daha fazla güç harcar. Benzer şekilde hata düzeltme kodunun hesaplama/çözme birimleri tek-bit pariteninkilerden 4 kat daha fazla alan kaplar.

Birçok araştırmacı da yazmaç öbeğindeki güvenilirliği artırmak için belli başlı beş çeşit yöntem önermiştir. Bunlardan biri derleyici kullanarak ya da ondan destek alarak yazmaç öbeğindeki geçici hatalara karşı koruma sağlamaktır [30] [33] [37]. Bu çalışmalar, donanım dahil edilmediği sürece düşük seviyelerde güvenilirlik getirmekte, bu nedenle de tek başına etkin olamamaktadırlar.

Diğer bir yöntem ise yazmaçların boyutunun genellikle kıymetli kısmından daha fazla olduğu bulgusundan yola çıkarak, yazmacın kıymetsiz kısmını kıymetli kısmı korumak için değişik yollarla kullanmayı içerir [28] [35] [38] [39]. Bu yöntemler de alan anlamında tasarruf sağlasa da enerji ve karmaşıklık düzeyini artırmalarından dolayı pek tercih edilmemektedir.

Bir kısım araştırmacılar ise daha önceden işlemci başarımının artırılması için önerilen yazmaç önbelleğini yazmaç öbeğinin hataya açıklığını azaltmak için kullanmayı önermişlerdir [31] [32] [41]. Ancak yazmaç önbelleği eklemek önemli bir enerji ve alan kaybına sebep olmaktadır ve bu nedenle gerçek sistemlerde geçici hatalara karşı kullanılmamaktadır.

Buyruk dizileriyle birlikte onların birer kopyalarını da aynı işlemcide sürdürerek, bu buyrukların sonuçlarını karşılaştırmaya ve bu şekilde hataları engellemeye dönük çalışmalar ise [34] [36] işlemciye neredeyse iki katı yük getirmekte ve hem hatasız durumdaki başarımı düşürmekte hem de günümüz işlemcilerindeki sorunlardan olan enerji ve sıcak bölge problemini daha da artırmaktadır.

Son olarak donanımsal ve mimari tasarımla koruma sağlayan mekanizmalar [29] [40] [54] [8] enerji ve alan anlamında daha etkin olmakla birlikte yazmacın dahili verimsizliğini kullanmayı içermediklerinden potansiyel kazancı maksimize edememişlerdir.

Bizim önerdiğimiz ve 5. bölümde ayrıntılı anlattığımız çözüm ise yazmaç öbeğinde yazmaçların çoğunu korumaya yetecek kadar tekrarlamanın olduğu bulgumuza dayanmaktadır. Çözümümüz yazmaç öbeğinin zaten içinde bulunan korumayı etkinleştirme üzerine olduğundan, bizden önce önerilen yöntemlerde bulunan enerji, zaman ve alanla ilgili dezavantajlar çok düşük seviyelerdedir.

Geçici hatalara karşı yazılım merkezli çözüm öneren bir çalışmada araştırmacılar geçici hataların kullanıcılar tarafından fark edilmemesini amaçlamışlardır [30]. Geçici hataların ticari sistemlerde kullanıcıyı rahatsız ettiği sürece önemsenmesi gerektiğini iddia eden çalışmada kod-yinelemesi denilen ve derleyici düzeyinde aynı buyruğun iki kez çağrılıp hatanın fark edilmesine dayanan yöntemi sadece kullanıcıya program/sistem hatası şeklinde görünme ihtimali olan durumlar için uygulanması önerilmiştir. Bu amaçla derleyici, programın ilgili karakteristiğini çıkarmak için 3 tur

daha çalıştırılır. Bu çalışma yazılımsal çözüm anlamında yükü azaltsa da, donanımda denetim-noktası kaydı (checkpointing) bulunduğuna dair varsayımı ve bu yolla engellenebilen hataları engellenmiş kabul etmesi mevcut işlemciler göz önüne alındığında geçerli bir varsayım olmamakta ve bu çalışmanın işlevselliğini zayıflatmaktadır.

Yazmaç öbeğinin hatadan kolay etkilenmesini nicel bir ölçüye dayandıran bir çalışmada araştırmacılar yazmacın öbekte aktif olarak bulunduğu süre boyunca ne kadar zaman hataya açık olduğunu *yazmacın hataya açıklık faktörü* olarak tanımlamışlardır [33]. Burada araştırmacılar yazma (Y) ve okuma (O) döngüsünde yazmacın hataya açık olduğu aralıkların sırasıyla Y $\rightarrow$ O veya O $\rightarrow$ O aralıkları olduğu, O $\rightarrow$ Y veya Y $\rightarrow$ Y aralığındaki hataların sonradan gelen yazmadan dolayı görünür olmadan yok olduğunu gözlemlemişlerdir. Buna bağlı olarak yazmaçların hataya açıklığının azaltılması için buyrukların sırasının, yazma operasyonlarının olabildiğince sonraya bırakılması ve okuma operasyonlarının da olabildiğince öne alınması için değiştirilmesini önermişlerdir. Böylece hatalara açık bulunulan süre azalacak ve yazmacın hataya açıklık faktörü düşecektir. Ayrıca alan ve güç kaybının büyük olmaması amacıyla yazmaç öbeğinin yalnızca küçük bir kısmını hata düzeltme koduyla korumalı tasarlayarak, yalnızca hataya açıklığı yüksek olan yazmaçları hata düzeltme koduyla korumalı olan bölüme koymak suretiyle dayanıklılık sağlanması önerilmektedir. Elbette buyrukların sıralamasının değiştirilmesi bir çok kısıttan dolayı her zaman mümkün olmamaktadır. Bu sebeple de bu çalışmada elde edilen fayda yeterince yüksek olmamıştır.

Yazmaç öbeğini korumak için yalnızca donanımsal yöntem kullanan çözümlerin enerji-etkin olmadığını ileri süren bir çalışmada ise donanımsal korumaya derleyici tarafından yardımcı olunduğunda yazmaç öbeğinin daha az enerjiyle korunabileceği belirtilmiştir [37]. Bu amaçla, donanımsal koruma ile yazmaç öbeğinin bir kısmının korunabildiğini varsaymış olan yazarlar, korunacak yazmaçların derleme sonrasında belirli algoritmalarla seçilebileceğini önermişlerdir. Bu şekilde hataya açık olan yazmaçlar bulunduktan sonra, bu yazmaçların donanıma bildirilmesi için de yazmaçların en üstteki yazmaç numaralarına konulmasını önermişlerdir. Böylece K tane yazmacı koruyabilen bir donanımın sadece yazmaç öbeğinin üstteki K yazmacını koruması yeterli olacaktır. Yazmaç yeniden adlandırma özelliği bulunan çok-yollu (superscalar) sistemlerde ise bu yöntem ufak değişikliklerle yine uygulanabilmektedir. Böylece donanımın yazmaç öbeğini korumak için enerji harcayacağı işlemler, hata düzeltme kodu kullanılıyorsa, yalnızca kodu üretmek ve okuma sırasında kontrol etmekten ibaret olacaktır. Bu çalışmada önerilen yöntemle, yazmaç öbeğini donanımsal olarak koruma için harcanan enerji bir miktar düşmektedir.

Yazmaçların önemli bir kısmının yalnızca alt baytlarını kullanması, dolayısıyla dar yapılı olması durumunu yazmaçların güvenilirliği için kullanmayı öneren bir çalışmada yazarlar fonksiyonel birimlere bir *dar yapı algılayıcısı* ekleyerek, 64 bit olan yazmacın alt yarısıyla ifade edilebilen dar yapılı yazmaç değerinin otomatik olarak kendini üst yarısına da kopyalamasını önerir [28]. *Yazmaç-İçi Çoklama* adını verdikleri bu tasarımda araştırmacılar yazmaçlara veri iletmek için kullanılan mevcut hattın üst 32-bitlik kısmını kullanarak yazmayı önermektedirler. Hata algılaması içinse alt yarıya sığabilen değere parite biti eklemeyi ve üst yarıdaki kopyayı yalnızca hata düzeltme için kullanmayı önermektedirler. Dar yapılı yazmaç değerlerini kullanmayı öneren yöntemlerin genelinde olduğu gibi bu çalışmada da dar yapılı olmayan yazmaçların korunması hakkında herhangi bir çözüm önerilmemektedir.

Yazmaçların dar yapılı olmasını yazmaç öbeğinin güvenilirliği için kullanmayı öneren bir başka çalışma, 32-bit yazmaçların çoğunun dar yapısının kalan kısma hata düzeltme kodunu sıkıştırmaya yeteceğini öngörmektedir [35]. 26-bitlik değerin tek bitlik hatalarını düzeltecek Hamming kodu 5-bit olacağından, 32-bitlik bir yazmacın 31 bitine sığacaktır. Ancak bunun ön-koşulu yazmacın kıymetli bitlerinin 26 bitten çok olmamasıdır. Yazarlar benzetimlerde 26 bitten fazla kıymetli biti olan yazmaçların çok az olduğunu, bu nedenle de yazmaç öbeğinin büyük kısmının fazladan alan kullanmadan korunabileceğini bulmuşlardır. 32 bitin kalan 1 biti de yazmacın kıymetli bitlerinin 26 bitten fazla olup olmadığının belirtilmesi ve fazlaysa hata düzeltme koduyla korumasının olmadığının işlemci tarafından anlaşılması için kullanılır. Bu çalışma, alan anlamında bir tasrruf sağlamakta ise de, her okuma-yazma işleminde hata düzeltme kodu çözme-hesaplama yapılması gerektiğinden harcanan gücü önemli miktarda artırmaktadır.

Dar yapılı verilerin kıymetsiz kısımlarını değişik şekillerde kullanmayı öneren bir başka çalışmada ise yazarlar bu kez işlemci önbelleğindeki yapılara dönük bir araştırma yapmışlardır [38]. Öncelikle kıymetsiz kısımlarda meydana gelen hataların da kıymetsiz olduğunu ve bunların fark edilmesinin hataya açıklık faktöründe düşüşe, koruma durumunda ise -bu kısımlar korunmayacağından- tasarrufa sebep olacağını belirtmişlerdir. Bir başka kullanım olarak ise kıymetli kısımların kıymetsiz bitlerde kopyalanmasıyla koruma sağlamayı önermişlerdir. Bu çalışmada son olarak da kopya değerlerin kaydedileceği bölgeyi genişletmiş, sıfır değerini tutan bölgeleri de kıymetli kısımları kaydetmekte kullanmışlardır. Bu çalışmanın bir uyarlaması da yazmaç öbeğindeki tek bit hataları düzeltmek için kullanılmıştır [39].

Gömülü işlemciler için yazmaç önbelleği kullanarak geçici hata koruması öneren bir çalışmada araştırmacılar öncelikle yazılımda hata olarak ortaya çıkan durumların gömülü işlemcide bulunan hangi kısımlarda gerçekleşen geçici hatalardan

kaynaklandığını araştırmışlar ve bunların çoğunun yazmaç öbeğindeki kayıt ve mantık elemanlarından kaynaklandığını görmüşlerdir [31]. Bu nedenle hem mevcut hata algılama ve düzeltme kodlarına göre mantıksal devreleri de kapsayıcı koruma sağlayan (ama yazmaç öbeğinin tamamını korumayan), hem de onlardan sadece bir miktar fazla alan ve enerji artışına sebep olan yazmaç öbeği önbelleği tasarımını önermişlerdir. Bu önbellek son kullanılan bir kaç yazma değerini tutmakta ve yazmaç okumasında yazmaç öbeği ve önbelleği karşılaştırılmaktadır. Önbellekteki değerler için aynı zamanda bir Döngüsel Artıklık Denetimi (Cyclic Redundancy Check - CRC) değeri hesaplanıp tutulmaktadır. Hata görüldüğünde, eğer önbellekteki CRC değeri tutarlıysa bu değer doğru kabul edilmekte, aksi takdirde yazmaç öbeğindeki değer doğru kabul edilmektedir.

Gömülü işlemcilerde bulunan yazmaç öbeğini geçici hatalara karşı korumak için önbellek kullanımını öneren bir başka çalışmada ise önbelleği devresel tekniklerle hatalara karşı koruma detaylandırılmıştır [32]. Bu çalışmada öncelikle dayanıklı bir önbelleğe konulacak yazmaçların o anda yoğun olarak kullanılan yazmaçlardan seçilmesi, önbellekten yazmaç çıkarılması gerektiğindeyse en çok tüketilmiş, yani buyruklarca okunmuş olan yazmacın kurban seçilmeye en uygun eleman olacağı düşünülmüştür. Buna yönelik yapılan tasarımda önbelleği hatalara karşı korumak amacıyla da gecikmeli çoklama yaparak, hem çoklama sayesinde birden fazla kopya ile koruma, hem de gecikmeli olmasıyla kopyalama sırasındaki parçacık çarpmasından kaynaklı hatalara karşı bir dayanıklılık sağlama önerilmiştir. Ayrıca tasarlanan yazmaç önbelleğinde, aktif kullanılan yazmaçların çoğu mevcut olduğundan, büyük bir önbellek dolayısıyla kaybedilen enerjinin bir kısmını yazmaç öbeğinin saatini durdurarak yazmaç öbeğinin dinamik gücünden tasarruf edilmesi denenmiştir. Buna rağmen tasarımda yine de önemli oranda güç artışı oluşmuştur. Bu çalışmada kullanılan boyutta bir önbellek ciddi bir alan kaplamakla birlikte aynı zamanda kablo uzunluklarının artışından kaynaklı önemli bir güç sarfiyatına da sebep olmaktadır.

Yazmaç öbeğinin içinde çok sık değişen ve yazmaç öbeğine yazılan yığıt göstergesi (stack pointer) için ayrılan kısmın oldukça fazla olduğunu gözlemleyen bir çalışmada ise araştırmacılar küçük bir yazmaç önbelleğinde yığıt göstergesinin beklenen değerinin bir kısmını tutarak yığıt göstergesinin güvenilirliğini önemli miktarda artırmışlardır [41].

Ticari işlemcilerde neredeyse hiçbir donanımsal değişiklik yapmadan geçici hatalara dayanıklılık sağlamayı amaçlayan bir çalışma, sırasız-yürütüm yapan çok-yollu işlemcilerin içinde bulunan çoklu işleme imkanını kullanmayı önerir [34]. Bu çalışma gelen buyruk dizisini, sanki aynı buyruk dizisinden birden fazla sayıda geliyormuş

gibi ama kopya buyruk dizilerini birbirine bağımlı olmayan buyruklar olarak işlemci yollarına vererek, hatalı işlem sonucunu buyruklar emekli olmadan fark etmeyi önerir. Hata algılandığında ise zaten henüz emekli olmamış yazmaçları iptal edip hatalı buyruğun başına geri dönmek, bu sayede geçici hataya karşı koruma sağlamak amaçlanır. Bu çalışmanın işlemci birimlerini hataya dayanıklılık için aşırı kullanması ve dolayısıyla yaygın durum olan hatasızlık durumunda dahi verimliliğin aşırı derecede düşmesine sebep olması en büyük dezavantajıdır.

Simultane çok iş-parçacıklı (Simultaneous Multithreading - SMT) işlemcilerin aynı anda birden fazla işlemi götürebilmesini dayanıklılık için kullanmayı öneren bir başka çalışmada ise yalnızca yazmaç öbeğini değil bütün veri yolunu korumak hedeflenmiştir [36]. Bazı ticari sistemlerde iki ayrı işlemcide aynı işlemi yürütmek ve her aşamada karşılaştırma yapmak suretiyle koruma sağlanmaktadır. Ancak yazarlar aynı işlemin çok iş parçacığını yürütebilen tek bir işlemcide olmasının daha iyi olacağını iddia etmektedirler. Bunun sebepleri olarak da daha az donanımla korumayı başarması, kaynak paylaşımı dinamik olduğunda işlemci elemanlarından daha yoğun faydalanarak başarımı artırma imkanı ve zaten üreticilerin üretmekte olduğu SMT işlemcilerin daha çok güvenilirlik isteyen başka bir market için kullanılmasının daha ekonomik olması ileri sürülmektedir. Ancak bu yöntemle koruma sağlanması için her aşamada karşılaştırma yapmak gerekmektedir; yazarlar da bu nedenle belleğe iki iş parçacığı için bir kere erişilmesini ve böylece aynı veriyle ilerlemelerini önermişlerdir. Bununla beraber başarımı artırmak için gecikmeli yürütüm adını verdikleri ve önbellekle dallanma tahmincisinin çok az önde giden iş parçacığı sayesinde ilgili/doğru verilerle dolmasını önermişlerdir. Ayrıca arkadaki iş parçacığı öndekinin tahminlerinin sonucunu kullanır ve dallanmalarda hata yapmaz. Bu çalışmada önerilen yöntemler sayesinde simultane çok iş-parçacıklı yürütüm kullanılarak çok işlemcili yürütüme göre daha az donanım ve daha fazla başarım ile güvenilirlik sağlanmış olur.

Yazmaç öbekleri ve L1 önbelleklerinde, kapladığı alanın büyük olması ve zamanlamada gecikmelere sebep olmasından dolayı hata algılama ve düzeltme kodu (Error Detecting And Correcting Code -EDAC) kullanılamamasından yola çıkan bazı araştırmacılar daha küçük kapsamlı bir hata algılama ve düzeltme yöntemi önermişlerdir [29]. Bu çalışmada önerilen yöntem iki boyutlu parite hesaplaması ile her satırda bayt başına bir bit paritenin yanında her sütunda da yazmaç öbeğinin o sütunu için bir parite bulundurur. Bunun yanında parite hesaplama ve çözümleme işlemleri bir kaç saat döngüsünde ve işlemcinin boru hattından bağımsız gerçekleşmektedir. Bu şekilde tek bitlik hatanın yeri noktasal olarak tespit edilip düzeltilebilmekte ve zamansal olarak da işlemci başarımına hata olmayan durumda önemli bir yük getirmemektedir. Ancak önerilen yöntemin sebep olduğu harcanan

enerji ve alandaki artış, mevcut hata düzeltme kodlarından daha düşük olmamakla birlikte, işlemcide kullanılan diğer hata düzeltme kodları genellikle 2-bitlik hata algılama sağlarken bu yöntem sadece tek bit hata algılayabilmektedir.

Yazmaç öbeğinin tamamını değil de bir kısmını hata düzeltme koduyla koruyarak yazmaç öbeğinin hataya açıklık faktörünü düşürmeyi amaçlayan bir çalışmada yazarlar hataya daha açık olan yazmaçları sadece kullanılacakları zaman korumayı önermektedirler [54] . Koruma işlemi için eklenmiş bir hata düzeltme belleğinde hata düzeltme kodlarını tutarak, yazma işlemiyle paralel şekilde hata düzeltme kodu hesaplanıp belleğe aktarılması önerilmektedir. Tek bit hataların tamamını algılayabilmek için, önerdikleri sistemde mevcut olan hata düzeltme kodu devresini her yazmaç için parite hesaplamakta kullanmayı düşünmüşlerdir. Hata düzeltme kodunu hesaplama ve çözme işlemini yazma ve okumayla beraber yaptıklarından önerilen yöntemin başarıma etkisi olmamakta, ancak hata düzeltme devresinin alana ve güç tüketimine yazmaç öbeğinin tam korunması kadar olmasa da önemli etkisi olmaktadır. Çalışmada korumaya alınacak yazmacı seçmek için uzun yaşayacağını tahmin ettikleri yazmaçlara öncelik vermekte, yazmacın başka tüketicisi olmadığında Yeniden Sıralama Belleği'ne (Re-Order Buffer - ROB) yollanan yazmacı emekli etme sinyalinin aynısının hata düzeltme belleğine de yollanmasını, böylece hata düzeltme belleğinden atılmasını önermektedirler. Hata algılanması ve hata düzeltme imkanı olmaması durumunda, yanlış dallanma sonrasında yapılan işlemlere benzer olarak, ROB ve hata düzeltme belleği boşaltılır ve program sayacı hata olan buyruktan tekrar başlar. Bu çalışmada önerilen yöntemle yazmaç öbeğinin hataya açıklık faktörü önemli oranda düşürülmüştür.

Yazmaç öbeğinin güvenilirliği için yapılan bir başka araştırmada SRAM bit hücresine okuma ve yazma portlarıyla erişimin olmadığı iki adet DEĞİL kapısı eklenerek dahili bir hata algılama mekanizması kurulması önerilmiştir [40]. Böylece tek bir yazmacın içinde aynı değerin iki kopyasını tutma imkanı sağlanmıştır. Bu çalışmada alanda ve enerjide bir miktar artış olmuş ancak zamanlamada önemli bir gecikme görülmemiştir.

Bahsedilen çalışmaların önerdiği yöntemlerden farklı olarak bizim yöntemlerimiz yazmaç öbeğinde halihazırda bulunan ve etkin kullanılmayan yazmaç yedekliliğinden güvenilirlik için, yazmaçların az miktarda gerçekleşen değişiminden ise enerji verimliliği için istifade etmeyi önerir.

3. İŞLEMCİ YAPILARI

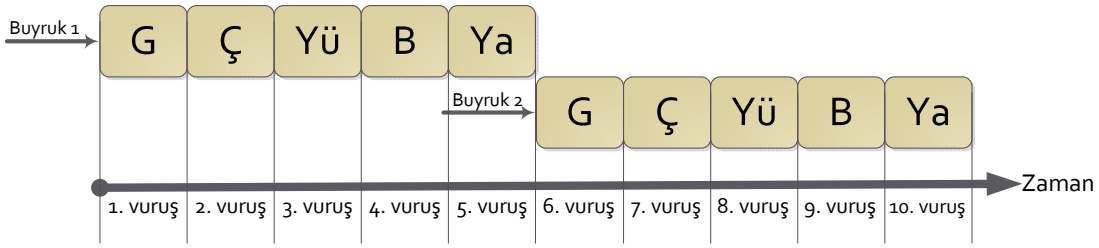
Basit mantıksal birimlerin uygulanmasıyla birlikte 1960'ların sonunda ortaya çıkan birkaç yüz transistörden oluşan mikro-işlemciler, hızla gelişerek günümüzde milyarlarca transistörden oluşan yoğun işlem kapasitesine sahip sistemler haline gelmişlerdir. Bu süreçte çok sayıda işlemci üreticisi ve yapıları ortaya çıkmış, bunlardan bir kısmı yok olmuş, bir kısmı ise günümüze kadar gelmiştir.

İşlemcilerin gelişiminde mihenk taşlarından biri boru hattı mimarisinin uygulanmasıdır. Boru hattı sayesinde her saat vuruşunda bir buyruk sonucunun ortaya çıkabilmesi, işlemci içindeki birimlerin üretim tesisindeki montaj hattı birimleri gibi çalıştırılmasıyla başarılmaktadır. 1990'larda uygulanmaya başlanan sırasız yürütüm ve çok yollu işlemci mimarisi de buyrukların aynı anda ilerlemesi ve birimlerin olabildiğince yüksek kapasitede çalışması sayesinde işlemci başarımını artırmıştır. Son olarak 2000'lerde uygulanmaya başlanan heterojen veya homojen çok çekirdekli işlemci yapıları da koşan bazı işlemlerin farklı çekirdeklerde birbirini etkilemeden devam etmesini ve işlemcideki sıcak bölgelerin dağıtılmasını amaçlayan bir mimari olarak ortaya çıkmıştır.

Modern işlemcilerde yüksek başarım için çoğunlukla sırasız yürütüm uygulanmakta, güç tasarrufu gerektiren uygulamalar içinse sıralı yürütüm tercih edilmektedir. Çok yollu ve çok çekirdekli tasarımlar her iki durumda da kullanılmaktadır. Alt bölümlerde sıralı yürütüm anlatılırken boru hattı ve işlemci elemanlarından da bahsedilecektir. Sırasız yürütüm ayrıntılandırılırken örnek olarak Intel'in Pentium 3 ve Core işlemcilerinde kullandığı P6 mimarisi anlatılacaktır. Anlatımlarda bilhassa işlemcilerin yazmaç öbeği tasarımı üzerinde durulacaktır.

3.1. Sıralı Yürütüm Yapan İşlemciler

Sıralı yürütüm yapan işlemciler genel işlemci elemanlarını ve boru hatlı yapıyı açıklamak için idealdir. Bu tür işlemciler daha çok gömülü işlemcilerde ve düşük güç tüketimi ihtiyacı olan işlemcilerde bulunur. İşlemcilerin verimini artıran en temel faktör boru hattı tekniğidir. Alt başlıklarda işlemcilerde boru hattı tekniği ve sıklıkla



Şekil 3.1: Buyrukların boru hattı yapısı olmaksızın 5 aşamalı süreci.

rastlanan aşamaları ile işlemci elemanları anlatılacaktır.

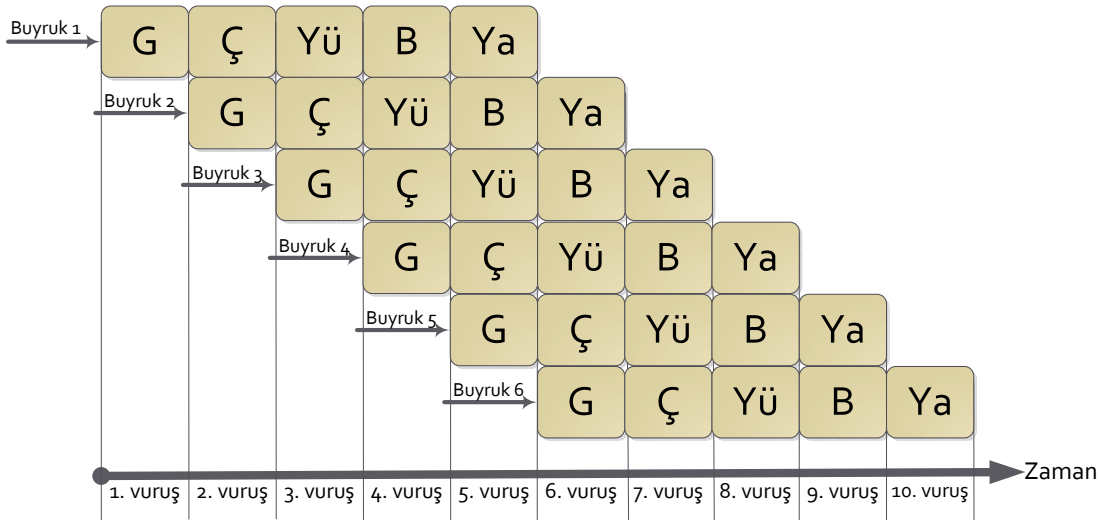
3.1.1 İşlemcilerde boru hattı

Boru hattı (pipeline) tekniği, işlemcideki kaynakların daha verimli kullanılabilmesi için fabrikalardaki üretim hattı gibi çalışmasına dayanır. İşlemcide bir buyruğun genel olarak geçtiği aşamalar şöyle sıralanabilir:

- **1. Getir (G)** : Buyruğun bir buyruk belleğinden (genellikle buyruk önbelleği) getirilmesi
- **2. Çöz (Ç)** : Buyruğun anlamının çözülmesi ve ihtiyaç duyduğu yazmaçlara erişilmesi
- **3. Yürüt (Yü)** : Veri belleğine erişilmesi gerekiyorsa gerekli adresin hesaplanması ve gereken aritmetik, mantıksal veya diğer işlemlerin gerçekleştirilmesi
- **4. Bellek (B)** : Adresi bildirilen verilerin bellekten okunması
- **5. Yaz (Ya)** : Okunan veya hesaplanan verilerin yazmaç öbeğine yazılması

Buyruklar için bahsedilen aşamaları gerçekleştirmek amacıyla iki tür yürütüm yapılabilir. Birincisi 3.1 şeklinde görüldüğü gibi, bir buyruk ancak bütün aşamalardan geçtikten sonra diğer buyruk işleme başlar. Bu durumda her 5 saat vuruşunda bir buyruk tamamlanmış olur. Bu nedenle 3.1 şeklinde 10 saat vuruşunda ancak 2 adet buyruk tamamlanmıştır.

Bunun yerine, işlemci kaynaklarının daha verimli kullanılması için boru hattı yapısı kullanılır (Şekil 3.2). Boru hattı yapısında, her aşamanın süresinin aynı olması ve buyrukların birbirini beklememesi şartıyla, ilk buyruk tamamlandıktan sonra her saat vuruşunda bir buyruk sonucu alınır. Yani işlemcinin uzun bir süre bu şekilde çalıştığı



Şekil 3.2: Buyrukların boru hattı yapısı içinde 5 aşamalı süreci.

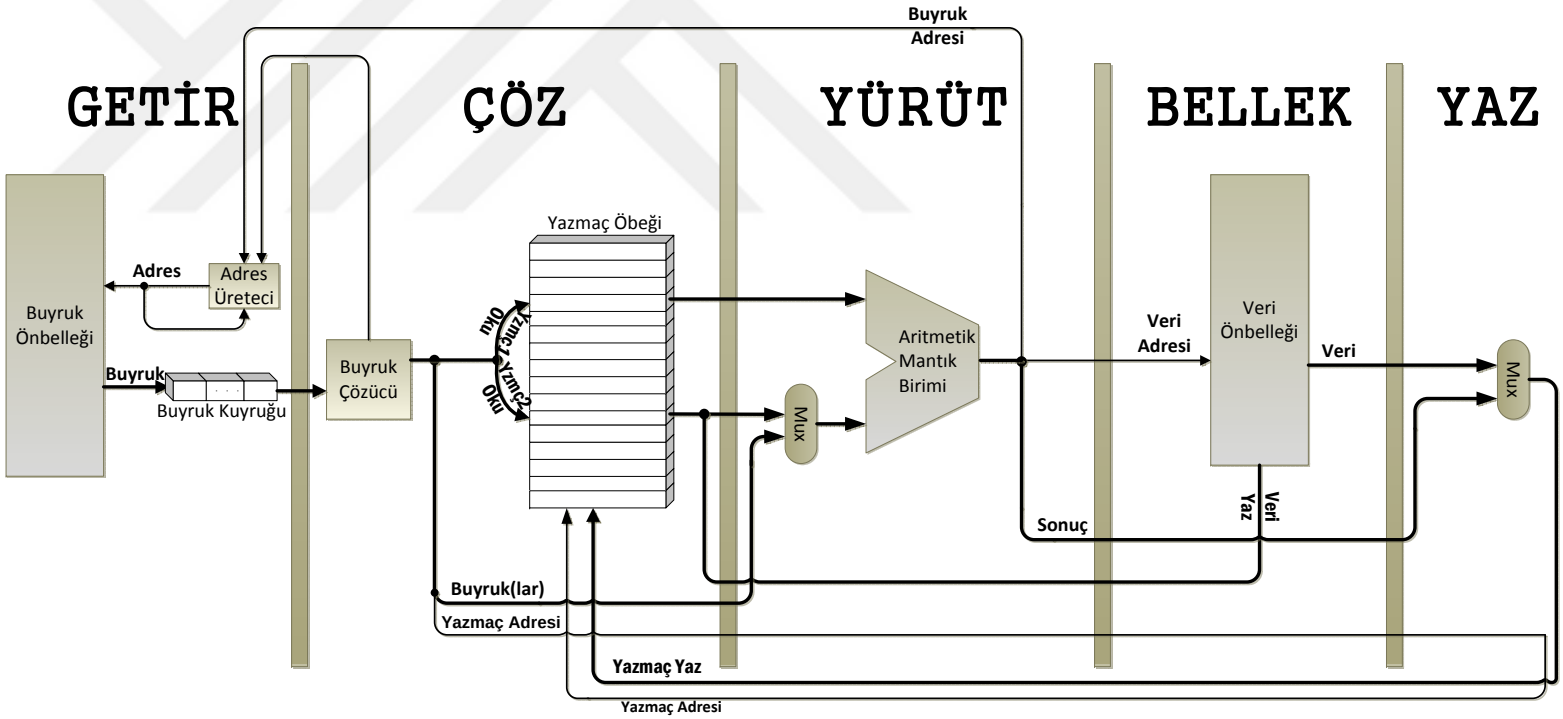
varsayılırsa, her bir vuruş başına bir buyruk tamamlanır. Bu da boru hatsız mimariye göre başarımı oldukça artırır.

Bahsedilen 5 aşamalı boru hattı yapısı MIPS, SPARC, Motorola 88000 ve Intel Pentium işlemcilerinde kullanılmıştır. Genel yapısı aynı olmakla birlikte boru hattı aşama sayısının arttığı sıralı mimariler de mevcuttur (Intel Atom, ARM Cortex gibi).

3.1.2 Sıralı yürütüm ve işlemci elemanları

Şekil 3.3 sıralı yürütüm yapan 5 aşamalı boru hatlı bir işlemci veri yolunu göstermektedir. Buyruklar ve veriler genellikle soldan sağa doğru ilerlemekle birlikte bazı durumlarda geri de dönebilmektedir. Her boru hattı aşamasının sonunda ilgili veriler saat kenarıyla tetiklenen flip-flop'larda bir sonraki saat kenarına kadar tutulur.

Buyruklar ana bellekte bulunmakla birlikte, işlemcinin buyruklara doğrudan ulaşabilmesi için işlemciye en yakın seviyedeki buyruk önbelleğine getirilir. **Getir** aşamasında programın buyrukları buradan genellikle sırayla alınır. Ayrıca birçok işlemci, buyrukları ihtiyaç duyulmadan önce çok sayıda getirip (öngetirme - prefetch) buyruk kuyruğuna koyarak verimliliğini artırmaya çalışır. Adres üretecinin görevi de buyrukların alınacağı adresi bulmaktır ve bunun için genellikle adresi sırayla artırır. Ancak koşulsuz/koşullu atlama veya dallanma durumunda adres sırayla artmayabilir. Koşulsuz atlama (JUMP buyruğu) durumu buyruk çözüldüğünde (**Çöz** aşaması) ortaya çıkar ve atlamanın yapılacağı adres adres üretecine verilir. Koşullu atlama ve dallanma (BRANCH buyruğu) durumunda ise iki ihtimal mevcuttur; ya buyruğun sonucu çıkana kadar bekleyip sonuç çıkınca atlama yapılacaktır, ya da sonuç tahmin edilmeye çalışılacak ve ancak yanlış tahmin durumunda yapılanlar telafi edilecektir.



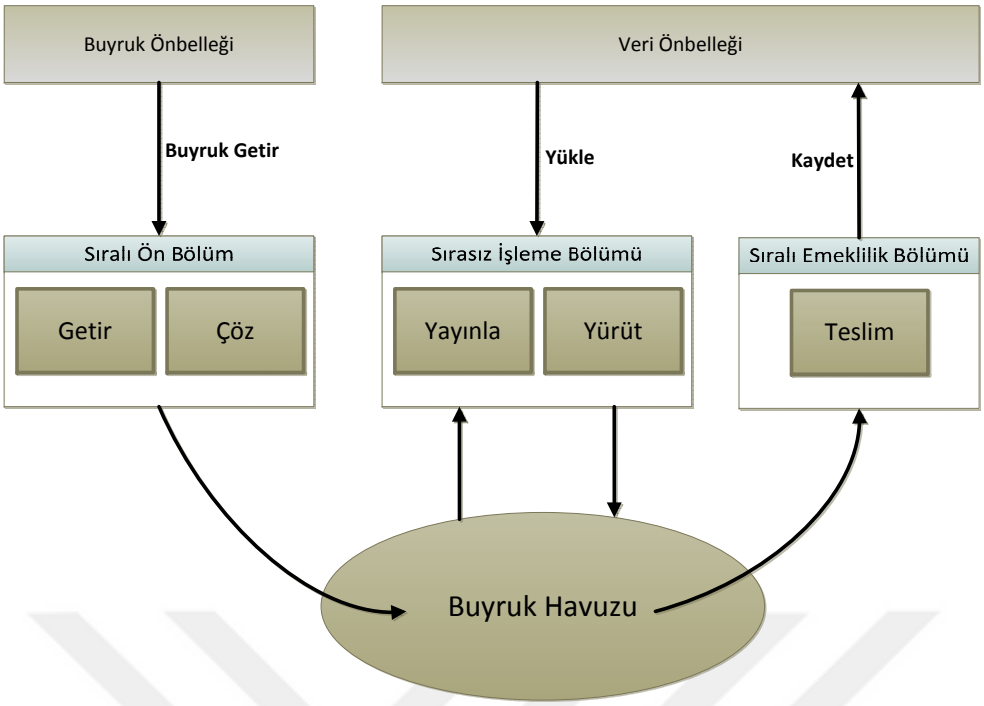
Şekil 3.3: Sıralı yürütüm yapan 5 aşamalı boru hatlı genel işlemci mimarisi.

Modern işlemcilerde çoğunlukla koşullu atlama veya dallanmanın sonucu **Çöz** aşamasında dallanma tahmin biriminde tahmin edilmekte ve tahmin edilen adres, adres üretecine verilmektedir. Tahmin sonucunun hatalı çıkması ise **Yürüt** aşamasının sonunda bulunmakta ve adres üretecine doğru adres bildirilmektedir. Hata durumunda dallanma buyruğundan sonra boru hattına girmiş olan bütün buyruklar iptal edilerek devam edilmesi gereken buyruktan tekrar başlanır.

5 aşamalı boru hattının **Çöz** aşamasında buyruk çözücü mantıksal devresi ve yazmaç öbeği vardır. Getirilen buyrukların çözülmesi işlemi buyrukların çeşidinin tanımlanması, adresleme modunun anlaşılması, yazmaç adresinin bulunması, diğer buyruklarla bağlılıklarının belirlenmesi aşamalarını içerir. Buyruk çözücü mantıksal devresi ayrıca buyruk boyutunun değişken olduğu işlemcilerde buyruk boyutunu da bulur ve karmaşık buyruklu işlemcilerde (Complex Instruction Set Computer) karmaşık buyruğu mikro-buyruklara dönüştürür. **Çöz** aşamasında, gelen buyruk yazmaçlara yazacak bir buyruksa yazmaç öbeğinde boş yazmaçlardan biri rezerve edilir. Eğer gelen buyruk yazmaç kullanan bir buyruksa yine bu aşamada yazmaç öbeğinden ilgili yazmaç veya yazmaçlar okunur.

Boru hattının **Yürüt** aşaması ise gereken hesaplamaların yapıldığı Aritmetik Mantık Birimini (AMB) içeren aşamadır. Diğer birimler aslında verileri tutup AMB'ye getirmek ve hesaplanan sonucu yine gerekli yerlere götürmek görevi yaparlar. AMB içinde aritmetik ve mantıksal operasyonları yapabilen sayısal devreler vardır. Bu operasyonlar toplama, çıkarma, artırma, azaltma, VE, VEYA, DEĞİL, ÖZELVEYA, bit kaydırma veya döndürme ve bazı işlemcilerde çarpma ve bölme işlemleridir. Bazı buyruklar bir operasyonu gerçekleştirmeyi belirli şartlara bağlarlar. Şartlı taşıma (CMOVcc) ve şartlı atlama (Jcc) gibi komutlar bunlara örnektir. Bu şartlar büyükse, sıfırsa, taşmışsa, eşitse, vs. şeklinde olabilir. İşlemcide bu durumlar bir yazmacın bitlerinde *bayrak* olarak tutulurlar. AMB işlemlerin sonucunda bu bayrakları da gerekiyorsa günceller. Boru hattının **Yürüt** aşamasında AMB sayesinde veri bellleğinden bir sonraki aşamada alınacak verinin adresi, yazmaçlar arasında yapılacak aritmetik/mantık operasyonların sonuçları veya atlama/dallanma yapılacaksa hedef buyruğun adresi hesaplanabilir. İlgili buyruk veri bellleğine erişimi gerektiriyorsa boru hattının **Bellek** aşamasına geçilir, yalnızca hesaplanan verinin yazmaç öbeğine yazılmasını içeriyorsa **Yaz** aşamasına geçilir.

Boru hattının **Bellek** aşamasında veri belleğine erişim sağlanır. Veri belleğine erişim gerektiren komutlar YÜKLE (LOAD) ve KAYDET (STORE) komutlarıdır. Bu komutların veri belleğine bildirilecek adresleri aritmetik mantık birimine girdikten sonra (çeşitli adres bulma metodlarına göre) hesaplanıp veri belleğine aktarılır. YÜKLE komutu veri belleğinden okuduğu veriyi yazmaç öbeğindeki ilgili yazmaca

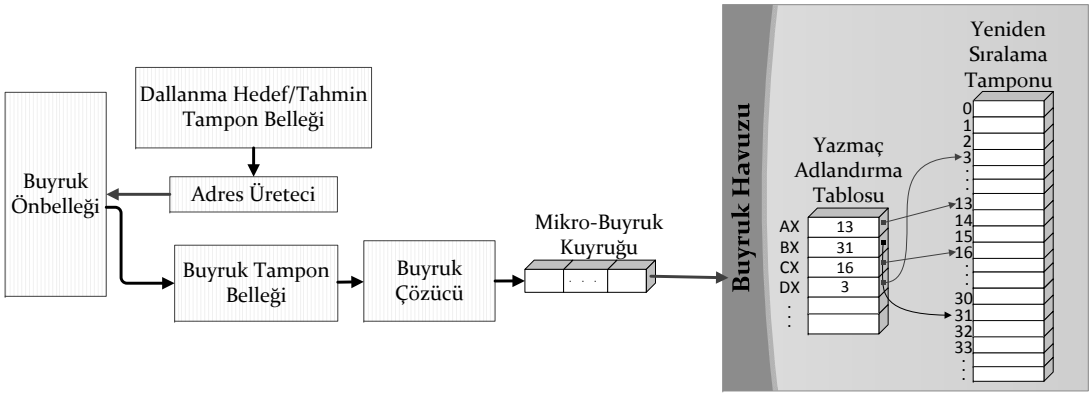


Şekil 3.4: Sırasız yürütüm yapan genel işlemci mimarisi blok diyagramı.

yazar. KAYDET komutu ise bir yazmacı veri belleğine yazar. Veri belleği genellikle hızlı erişim için ilk seviye veri önbelleği şeklindedir. Adreslenen veri önbellekte bulunamadığında, üst seviye bellekten sadece o veri değil o verinin etrafındaki veriler de önbelleğe alınır. Bunun sebebi veri erişiminin genellikle bellekteki yakın adreslere yakın zamanlarda gerçekleştiğinin gözlemlenmesi ve bu sayede gelecekte ihtiyaç duyulan verilere daha hızlı ulaşılmasının istenmesidir.

5 aşamalı boru hattının son aşaması olan **Yaz** aşamasında, hesaplanan veya veri belleğinden okunan değerler yazmaç öbeğine yazılır. Verinin yazmaç öbeğinde bulunan hangi yazmaca yazılacağı bilgisi buyruk içinde mevcuttur. Bu adres bilgisi buyrukla beraber boru hattında ilerler ve **Yaz** aşamasında yazılacak veriyle birlikte yazmaç öbeğine iletilir. Böylece gerekli veri yazmaç öbeğine konularak buyruk tamamlanmış olur.

Boru hattında en aktif eleman AMB olmakla birlikte, en aktif kayıt elemanı yazmaç öbeğidir. Zira neredeyse her buyruk yazmaç öbeğine yazma ve/veya yazmaç öbeğinden okuma(lar) yapmaktadır. Bu nedenle de yazmaç öbeği veri yolunun önemli miktarda enerji harcayan bir birimidir. Yine boru hattından görülebileceği üzere, yazmaç öbeğinin neredeyse her birimle ilişkili olması, yazmaç öbeğindeki herhangi bir hatanın diğer tüm birimlere sirayet edebileceğini göstermektedir.



Şekil 3.5: Sırasız işlemcilerdeki ön bölüm.

3.2. Sırasız Yürütüm Yapan İşlemciler

Sırasız yürütüm yapan işlemcilerde temel olarak üç aşama vardır (Şekil 3.4). Birinci aşama *sıralı ön bölüm*dür. Bu bölümde sıralı yürütümde olduğu gibi program kodunun buyruk akışı buyruk önbelleğinden alınır ve çözülür. Buradan gelen buyruklar bir havuza toplanır ve bu havuzdaki buyruklar ikinci aşama olan *sırasız işleme bölümü*nce kaynaklar uygun oldukça buyrukların bağımlılıkları gözetilerek alınır. Sırasız işleme bölümündeki sonuçlar geçici olarak yazmaçlarda tutulur. Bir buyruktan önceki tüm buyruklar tamamlanmış ve o buyruğa kadar olan tüm dallanmaların doğru olduğu ortaya çıkmışsa bu durumda buyruk *sıralı emeklilik bölümü*ne geçer. Emeklilik bölümü birinci aşama olan ön bölüm gibi sıralıdır ve buyrukları geliş sırasına göre alır. Bu bölümde buyrukla ilgili geçici olan bilgiler artık kalıcı duruma dönüşür ve bu şekilde tutulur. Sırasız yürütüm yapan işlemcilerde büyük ölçüde benzer yapı ve teknikler kullanıldığından, Şekil 3.4 üzerinde gösterilen bölümlerin ayrıntıları sırasız yürütüm yapan bir işlemci mimarisi olan Intel'in P6 mimarisi temel alınarak alt bölümlerde anlatılacaktır.

3.2.1 Sıralı ön bölüm

İşlemcinin ön bölümü buyrukları bellekten alıp işleme bölümüne aktarmakla sorumludur. Programlar genellikle kodun sırayla işleneceğini varsaydıklarından, ön bölüm sırasız işlemcilerde de sıralı olarak tasarlanmıştır. Ön bölüm birçok boru hattı aşamasından oluşsa da temel olarak getirme ve çözme işlevi vardır. 3.5 şeklinde Intel P6 mimarisinde kullanılan ön bölüm akışı ve bileşenleri gösterilmiştir. Adres üreteci buyruklara hızlı bir şekilde ulaşılabilmesi için işlemciye yakın bir konumda bulunan buyruk önbelleğinin adresini oluşturur. Buyruk, önbellekte bulunamadığında ise bellek hiyerarşisinin üst kısımlarında aranır. Ancak buyrukların buyruk önbelleğinde

bulunma oranı %90'ların üzerindedir.

Adres üretecinin genellikle önbelleğe adresleri sırayla vermesi beklenir, ancak dallanmalardan (branch) kaynaklanan adres atlamaları gerçekleşmektedir. Bu dallanmaların sonucunu beklemek boru hattının bir süre durmasına sebep olacağından, dallanmaların ne yönde gerçekleşeceğinin tahmin edilmesi gerekmektedir. Dallanmanın doğru tahmin edilememesi durumunda ise işlemcide ilerlemekte olan birçok buyruk geçersiz olacağından, işlemcilerde dallanmaları tahmin etmek ve buna dair hedef adresini tutmak için Dallanma Hedef/Tahmin Tampon Belleği (DHTTB) bulunur. DHTTB geçmişteki dallanmaların yönüne bakarak bir öngörüde bulunur ve buna göre de adres üretecine gidilecek olan adresi verir. Dallanma tahmini için P6 mimarisinde Yeh algoritması kullanılır [55]. Eğer karşılaşılan dallanmaya daha önce hiç rastlanmamışsa DHTTB'de bu dallanmaya ilişkin bilgi bulunmaz ve bu durumda varsayılan tahmin kullanılır. Boru hattının ilerleyen kısımlarında dallanma işleminin sonucu tahmin edilen sonuçla karşılaştırılır ve eğer sonuçlar uyuşmazsa geri döndürme işlemi başlatılır. Geri döndürme işlemi hatalı daldan sonra getirilmiş tüm buyrukları iptal etme ve yeni adresi doğru dala yönlendirme işleminden oluşur ve sık karşılaşılması durumunda işlemci başarımını önemli ölçüde düşürür.

Buyruk önbelleğine bildirilen adres ile ardındaki birkaç önbellek satırı (P6 mimarisinde 2 satır) alınır. Önbellek tasarımına göre değişse de her satırda 16 bayt kadar veri bulunur. Alınan buyrukların sınırları işaretlenerek Buyruk Tampon Belleğine aktarılır. Çok yollu işlemciler için işlemcinin yol sayısı kadar buyruk, buyruk çözücülere aktarılır. Bu bölümde, anlatımdaki sadeliği korumak için işlemcinin tek yollu olduğunu varsayıyoruz. Buyruk çözücü, buyrukları İndirgenmiş Buyruk İşlemcileri (Reduced Instruction Set Computer) gibi ve **mikro-buyruk** denilen bir forma sokar. Karmaşık buyruklar bir veya daha fazla mikro-buyruktan oluşabilirler. Buyruklar Intel mimarisinde ikili yapıdayken (bir tane aynı zamanda hedef olan kaynak ve ikinci kaynak) mikro-buyruklar üçlü yapıdadırlar (hedef, 1. kaynak, 2. kaynak). Buyruk çözücülerin aynı zamanda küçük birer tampon bellekleri olduğundan, bir buyruğun birden fazla mikro-buyruğu varsa bir sonraki aşamadan önce bir süre buyruk çözücüde tutulabilirler. Oluşan mikro-buyruklar Mikro-Buyruk Kuyruğundan buyruk havuzuna aktarılır. Buyruk havuzunda sırasız işlemcilerde toplam verimi artırmak için istihdam edilen yazmaç yeniden adlandırması yapılır.

3.2.1.1 Yazmaç yeniden adlandırma

Buyruk çözüldükten sonra yapılacak işlem buyrukla ilgili kaynakların okunmasıdır. Ancak buyruğun işleme alınabilmesi için kullanılacak işlem biriminin müsait olması

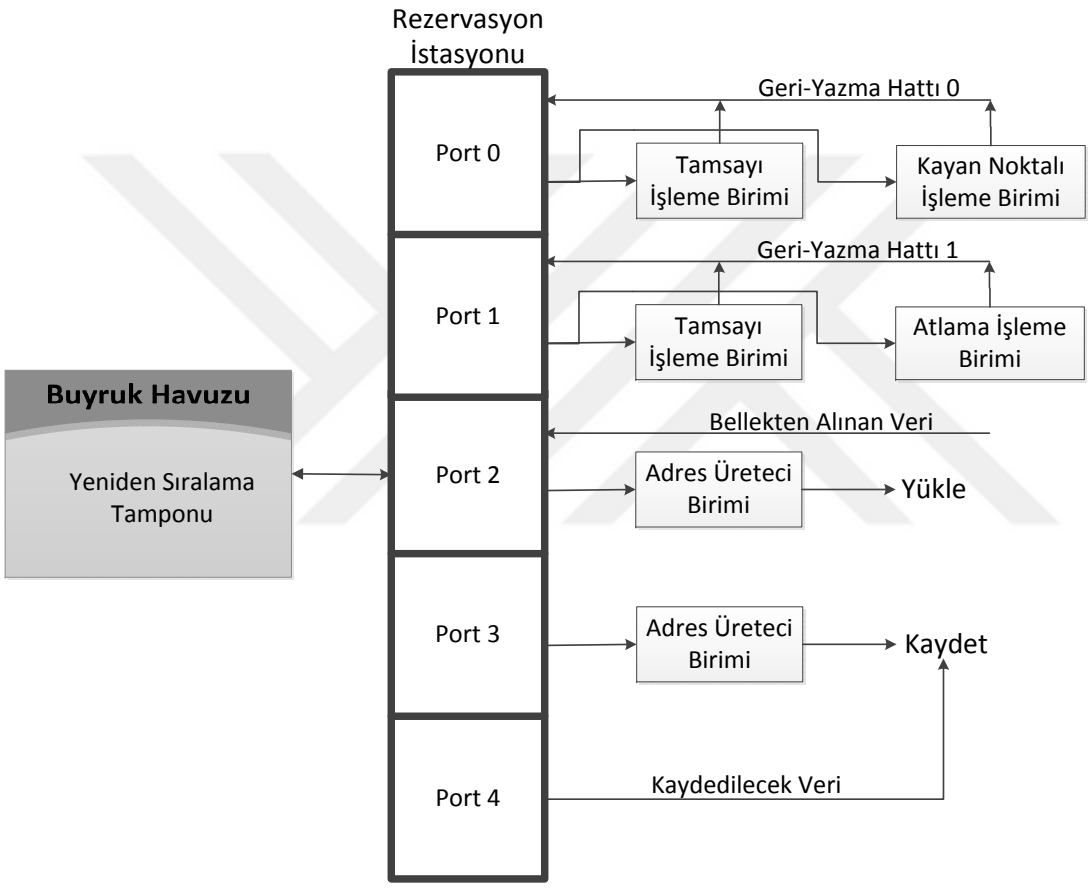
ve kaynakların hazır olması gerekmektedir. Eğer kaynaklar hazır değilse sıralama olarak daha önce gelen bir buyruğun sonucunu beklemek gerekmektedir ve buna *doğru veri bağımlılığı* denir. Eğer hedef hazır değilse, yani sıralama olarak daha önce gelen bir buyruk aynı hedefe yazacaksa, işlemi yapmak için bunun sonucunu beklemek şart değildir, zira işlemin sonucunu bulmak ve diğer işlem sonucunu yazana kadar geçici bir yerde tutmak yeterlidir. Bu şekilde olan bağımlılığa ise *çıktı bağımlılığı* veya *yanlış veri bağımlılığı* denir. Bu çeşit bağımlılıklar yazmaçların az sayıda olmasından ve bu nedenle tekrar kullanılmasından kaynaklanır. Eğer yeterli sayıda yazmaç bulunur ve bir yazmaç bir başka sonucu kaydetmek için tekrar kullanılmazsa, yanlış veri bağımlılığı hiç oluşmaz.

Yazmaç yeniden adlandırma yanlış veri bağımlılığını, aynı hedef yazmaca sahip her buyruğa farklı bir yazmaç tanımlayarak çözer. Böylece buyrukların hedefleri aynı görünse de, farklı yazmaçlara tanımlanmış olduklarından çıktı bağımlılıkları ortadan kalkar. Buyrukların görünen hedef ve kaynak yazmaçları buyruk setinin mimarisine bağlı olduğundan bu yazmaçların bulunduğu yazmaç öbeği *mimari yazmaç öbeği* diye adlandırılır. Buyrukların yazmaçlarının yeniden tanımlandığı ve mimari yazmaçlardan sayıca çok daha fazla olan yazmaçların bulunduğu belleğe ise *fiziksel yazmaç öbeği* denir. Bu durumu tutarlı olarak sürdürebilmek içinse mimari yazmaçların fiziksel yazmaçlara eşlendiği bir tabloya ihtiyaç vardır. Bu eşleme tablosuna *Yazmaç Adlandırma Tablosu* adı verilir. Bunun yanında yazmaç adlandırma tablosu bazen kaynak yazmacın mimari yazmaç öbeğinde bulunduğu bilgisini de verebilir, bu durumda buyruğun ihtiyaç duyduğu kaynak yazmacı fiziksel değil mimari yazmaç öbeğinden alması gerekir.

Mikro-buyruklar yazmaç adlandırma tablosundan en son eşleme bilgisine ulaşır ve mimari yazmaçlara referans veren kaynak yazmaç isimlerini gerekiyorsa fiziksel yazmaç isimlerine değiştirir ve bu noktadan sonra bu şekilde işlemci aşamalarına ilerler. Benzer şekilde, hedef yazmacı bulunan mikro-buyruklar da mimari yazmaca karşılık gelen bir fiziksel yazmaç rezerve eder ve yazmaç adlandırma tablosunu bu şekilde günceller. Intel P6 mimarisinde fiziksel yazmaç öbeği işlevini rezerve edilme sırasını da muhafaza ederek *Yeniden Sıralama Belleği* gerçekleştirir. Rezerve edilme sırası emekli etme aşamasında da buyrukların sırayla emekli edilemesini sağlar.

3.2.2 Sırasız işleme bölümü

Çözme aşaması tamamlandıktan sonra oluşan mikro-buyruklar için iki yapıda alan işgal edilir. Bunlardan biri fiziksel yazmaç öbeği görevini yapan ve mikro-buyruklarla ilgili işlemleri sırasıyla yapan *Yeniden Sıralama Belleği (YSB)*, diğeri ise



Şekil 3.6: Sırasız işlemcilerdeki işleme bölümü.

mikro-buyrukların işlemeye hazır hale gelip işleme alınana kadar beklediği *Rezervasyon İstasyonu (Rİ)* veya bazı işlemcilerde geçen adıyla *Yayınlama Kuyruğu*dur (Şekil 3.6). Sırasız işleme bölümünde bütün işlemler mikro-buyruklarla yapıldığından bu bölümde mikro-buyruk ve buyruk ifadesi aynı anlamda kullanılacaktır. Rİ'nin buyrukları tutmak için 20 tane yeri ve buyrukları 5 farklı kısma yayınlamak için 5 tane portu vardır. Her saat vuruşunda Rİ, içindeki buyrukların ihtiyaç duydukları verilerin oluşup oluşmadığını denetler ve hazır olan buyruklar için işleme birimlerinin uygunluğunu belirler. Rİ'nin ardında biri kayan noktalı ve tamsayılı işlemler biri atlama buyrukları ve tamsayılı işlemler için olan iki tane işleme birimi, biri yükleme biri kaydetme için kullanılan iki tane adres üreteci birimi ve bir de kaydedilecek veri için ayrılmış yol olan 5 kısım vardır. Ne zaman ki buyruk yayınlanır ve Rİ'nin ardındaki hatlara ilerlerse, Rİ'deki yeri boşalır. Tamsayı, kayan noktalı ve atlama işlemleriyle ilgili operasyonlarda Rİ'nin buyruğu işleme birimine yollaması için aşağıdakilerin gerçekleşmiş olması gerekmektedir:

- Buyruğun ihtiyaç duyduğu kaynak yazmaç veya verilerin hazır olması
- Buyruğun ihtiyaç duyduğu işleme biriminin müsait olması
- Buyruk işlendiğinde geri yazma hattının hazır olması

Bütün bu şartlar gerçekleşene kadar buyruk Rİ'de beklemek zorundadır ve ancak bu şartlar gerçekleştiğinde işleme birimlerine iletilir. İşlem tamamlanıp sonuç ortaya çıkınca geri yazma hattı tamamlanan sonucu alır ve yeniden sıralama belleğine yazar. Bellek işlemleri (yükle, kaydet) içinse adres üretecinde adres üretilir ve veri önbelleğine gidilir. İlgili veri adresi önbellekte ise operasyon yapılır. Eğer ilgili veri adresi önbellekte yoksa, ulaşana kadar hiyerarşideki bir üst belleğe gidilir ve bu zaman zarfında buyruk tampon bir bellekte tutulur. Veri önbelleğe geldiğinde buyruk tekrar işleme alınır ve buyruk tamamlanır ("yükle" buyruğu için önbellekten alınan veri yeniden sıralama belleğindeki ilgili fiziksel yazmaca yazılır, "kaydet" buyruğu için kaydedilecek veri önbelleğe yazılır).

3.2.3 Sıralı emeklilik bölümü

Sırasız işleme birimindeki ve fiziksel yazmaçlardaki veriler kesin değil tahmini verilerdir, çünkü sırasız işleme biriminde bir dallanmanın yanlış tahmin edildiği ortaya çıkarsa o dallanmadan sonraki buyruklarla ilgili bütün sonuçlar geçersiz olur. Emeklilik bölümü ise bu tahmini sonuçların kesin ve geri döndürülemez sonuçlar haline dönüştüğü bölümdür. Emeklilik bölümünde fiziksel yazmaçlar artık yazmaç

adlandırma tablosunda o fiziksel yazmaca karşılık gelen mimari yazmaca yazılır. Bir mikro-buyruğun emekli olması için şu şartların gerçekleşmesi zaruridir:

- Mikro-buyruğun *yeniden sıralama belleği*ndeki en eskisi olması
- Mikro-buyruğun parçası olduğu buyruğun, bütün mikro-buyruklarının sonuçlarının fiziksel yazmaçlara yazılmış olması (*yeniden sıralama belleği*ndaki yerlerine)
- Önce gelen buyrukların hiçbirinde yanlış tahmin edilmiş dallanma olmaması

Emekli etme işlemi, ilgili verinin *mimari yazmaç öbeği*ndeki ilgili yazmaca yazılmasıyla son bulur.

3.2.4 Yeniden sıralama belleği

Sırasız işlemciler, buyrukların sırasız bir şekilde işlenmesine ve sonuçlanmasına izin verirler. Ancak işlemcinin doğru şekilde çalışabilmesi için buyrukların belleği ve mimari yazmaçları programdaki sıralarıyla güncellemeleri gerekmektedir. Buyrukların sırasını koruyarak işlemcinin beklendiği gibi çalışmasını sağlayan yapı *yeniden sıralama belleği*dir (YSB). YSB sıralı işlemcilerdeki yazmaç öbeği görevini ifa etmesinin yanında, yazmaç yeniden adlandırma ve tahmini sonuçları kesinleştirme (emekli etme) işlevini de üstlenmiştir. YSB sırasız işlemcinin her üç bölümünede de kullanılır.

Yeniden sıralama belleği bir dairesel kuyruk şeklinde kullanılır (0'dan en üste, sonra tekrar 0'a). YSB tamamen dolu değilse *Çöz* aşamasındaki mikro-buyruk için YSB'de bir yer ayrılır. Her mikro-buyruk için bu işlem yapıldığından dolayı YSB'de kullanılan yerlerin sayısı işlemcide işlenen mikro-buyruk sayısını da verir. YSB'de ayrıca her mikro-buyruğun bağlı olduğu buyruğun ilk, ara veya son mikro-buyruğu olduğuna dair bilgi de tutulur. Bir buyrukla ilgili sonuçlar bulunur ve sırası gelirse YSB'den ayrılarak emekli edilir.

YSB, işleme birimlerinin sonuçlarını mimari yazmaçlara yazılmaksızın tutarak işlemenin kesin olmadan yapılabilmesini sağlar. YSB sayesinde işlemci, bütün dallanmaların doğru tahmin edildiğini varsayarak en yüksek hızda buyrukları işleyebilir. Eğer bir dallanma hatalı tahmin edilmiş olursa, işlemci birimi YSB'ye kaydedilmiş kesin olmayan sonuçları atarak hatayı kolayca telafi edebilir.

Görüldüğü üzere işlemcinin veri yolundaki neredeyse her aşamada önemli bir işlevi olan YSB aynı zamanda işlemcideki birçok yapıyla da doğrudan ilişkilidir. Bu

nedenle YSB (sırasız işlemcilerdeki genel adlandırmasıyla *fiziksel yazmaç öbeği*) hem veri yolunda harcanan enerjinin en büyük kullanıcılarından hem de herhangi bir hatanın en kolay yayılabileceği yapılardandır.

4. ENERJİ TASARRUFU İÇİN GÜNCELLEME TEMELLİ YAZMAÇ ÖBEĞİ MİMARİSİ

İçinde bulunduğumuz mobil bilgisayarlar çağında işlemcinin sıcaklığını düşük tutmak ve enerji kaybını azaltmak ciddi bir sorundur. Yazmaç öbeği ise modern çok yollu işlemcilerdeki en fazla enerji tüketen yapılardan biridir [56]. Bu nedenle yazmaç öbeğinin enerji sarfiyatını azaltmanın işlemcinin toplam güç tüketiminde önemli etkisi olacaktır. Bununla birlikte modern işlemciler yazmaç öbeğini yeterince verimli bir şekilde kullanmamaktadır.

Günümüz işlemci mimarilerde yazmaç öbeğinin enerji sarfiyatını azaltmak çeşitli nedenlerle önem arz etmektedir. Birinci olarak modern işlemci yapılarında yazmaç öbeği işlemci gücünün önemli bir bölümünü teşkil eder [9]. İkinci olarak, [57] çalışmasında belirtildiği üzere, yonga üzerinde artan çekirdek sayısıyla birlikte verilerle işlem yapmaktan çok veri taşımanın enerji maliyeti daha baskın olacaktır. Bu nedenle veriyi mümkün olduğunca yakında tutma çabası, yazmaç öbeği boyutunda büyük bir artış meydana getirecektir. Diğer taraftan, zaten işlemcinin enerji bütçesinin önemli bir bileşeni olan yazmaç öbeği, enerji verimliliği için dikkatli bir tasarım gerektirecektir. Üçüncü olarak, eş-zamanlı çoklu-işlemeyi (simultaneous multi-threading) kullanan mikro-işlemciler, mobil ve çok-çekirdekli işlemciler de dahil olmak üzere neredeyse bütün bilgisayar platformlarında bulunmaktadır [4] [58] [59]. Eş-zamanlı çoklu-işleme mimarisinde yazmaç öbeğinin yoğun kullanımı sebebiyle yazmaç öbeğinin enerji-etkin olması son derece önemlidir [60]. Son olarak, işlemcide ısınan noktaların önlenmesi modern işlemcilerin mühim bir zorluğudur ve yazmaç öbeği bu kapsamdaki en problemli yapıdır [1].

Ön-yükleme ve bit hattı enerjisi yazmaç öbeğine yazma operasyonu sırasında harcanan enerjinin neredeyse tamamına denk gelir [61]. Yazmaca yazılan bir değer bitlerin hepsini değiştirmese de, bit hatlarına her saat vuruşunda ön yükleme (precharging) ve yük boşaltma (discharging) yapılır. Çalışmalarımıza herhangi bir mimari yazmacın bitlerinin her buyrukta ortalama yalnızca % 10'unun değiştiği ve geri kalan bitlerinin aynı değeri tuttuğu gözlemiyle başladık. Bu gerçeğe rağmen yazmaç öbeği her bit için bütün sütunlara ön yükleme yaparak ve değeri yazmak için bit hattı enerjisi harcayarak bütün yazmaç değerini yazmaya çalışmaktadır. Biz bu

Çizelge 4.1: Yazmaç değerindeki değişimi gösteren kod bölümü.

Buyruk	*Açıklama*	Yürütmeden Önce İlk Yazmaç	Yürütmeden Önce İkinci Yazmaç	Yürütmeden Sonra Hedef Yazmaç	Değişen Bit Sayısı
SUB RDX,RSI	$RDX \Leftarrow RDX - RSI$	RDX : F5924E	RSI : F59240	RDX : E	**10**
MOV RCX,RDX	$RDX \Leftarrow RCX$	RCX : 0	RDX : E	RCX : E	**3**
SHR RCX,1	$RCX/2$	RCX : E	-	RCX : 7	**2**
XOR EDX,EDX	$EDX \Leftarrow EDX \oplus EDX$	RDX : E	-	RDX : 0	**3**

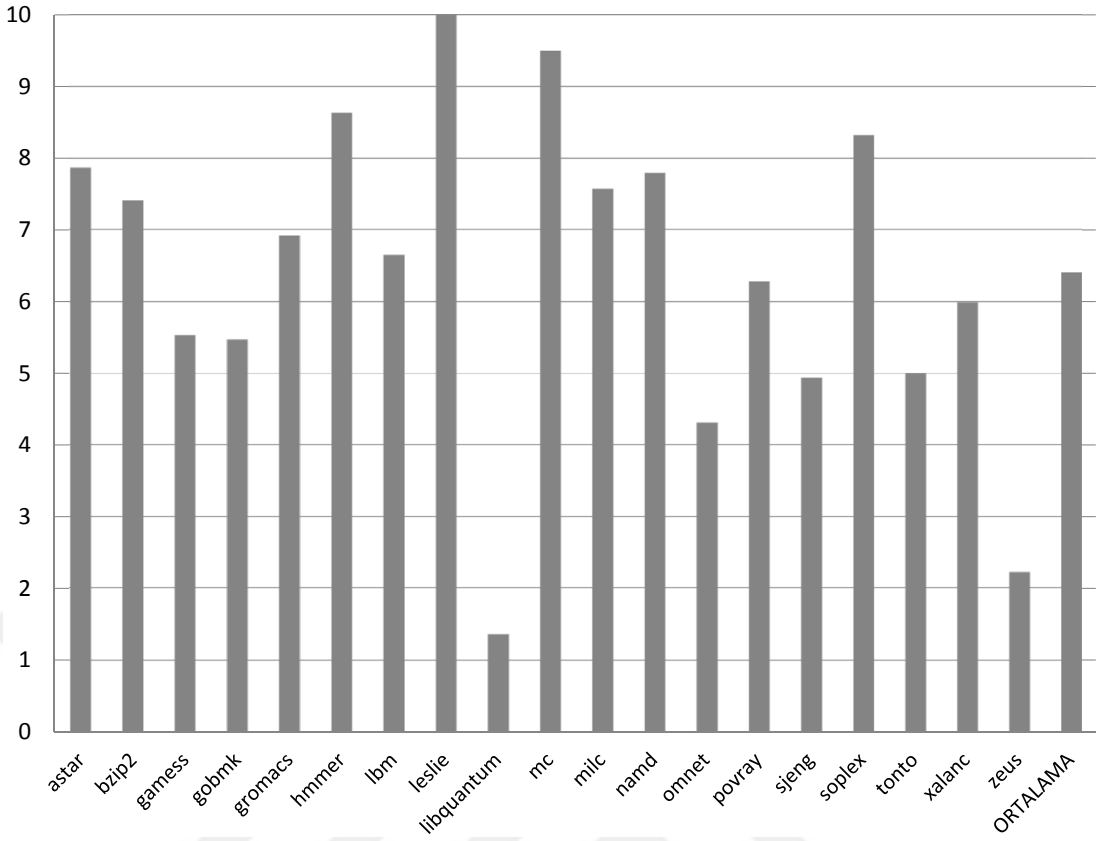
çalışmamızda, aynı kalan yazmaç bitlerine yazmayı engellemeyi, dolayısıyla da bit hücresinin tamamen boşalmasının önüne geçmeyi ve bit hattını sürme enerjisini azaltmayı öneriyoruz. Bu çalışmada mimari ve devre seviyesinde tekniklerin kombinasyonunu kullanan güncelleme-tabanlı bir tasarımla yazmaç öbeğinin yazma operasyonunda harcadığı gücü azaltmayı önermekteyiz. Bu tasarımı "Güncelleme-tabanlı Yazmaç öbeği Tasarımı (GÜNYET)" olarak adlandırmaktayız. GÜNYET mimarisi iki tane yeni enerji tasarruf mekanizmasını ve yeni mimari dolayısıyla elde ettiğimiz ekstra bir iyileştirmeyi kullanır. Tezin bu bölümündeki çalışmamız, çoğu mobil işlemcinin kullandığı sıralı yürütüm yapan işlemcileri hedeflemektedir [4] [5].

Yazmaç öbeği için enerji tasarrufu sağlamaya dönük çözümler öneren birçok çalışma yapılmıştır. Bu çalışmaların bazısı yazmaç önbelleği gibi fazladan kayıt yapıları kullanmak suretiyle [10], bazıları tahmin kullanma yoluyla [12], bir kısmı da yazma süresini artırmayı netice verecek yöntemlerle [2] yazmaç öbeği enerjisini azaltmayı önermişlerdir. Bizim önerdiğimiz yöntem ise enerji etkinliğini herhangi bir başarım azalmasına sebep olmaksızın ve mimaride ihmal edilebilir miktarda değişiklikle günümüz işlemcilerinde zaten mevcut olan güç tasarruf potansiyelini kullanmayı sağlar.

Bu bölümün kalan kısmındaki alt bölümler şu şekilde düzenlenmiştir. Bölüm 4.1 çalışmamızın motivasyonunu sunar. Bölüm 4.2 mimariyi açıklar. Bölüm 4.3 çalışmanın deneysel kurulumunu ve metodolojisini anlatır. Bölüm 4.4 sonuçları ve deneylerden elde edilen çıkarımları ayrıntılandırır. Bölüm 4.5 önerdiğimiz mimarinin sebep olduğu masrafları irdeler. Bölüm 4.6 ilgili çalışmalardan bahseder. Son olarak da bölüm 4.7 bulguları özetleyerek çalışmayı sonuçlandırır.

4.1. Motivasyon

GÜNYET tasarımı için bizi iki durum motive etti. Birincisi, yazmaç öbeğine yazmak için harcanan enerjinin yazmaç öbeğinden okumak için harcanan enerjiden çok daha fazla olması [62] [2] [3]. İkincisi de yazmaçlara yapılan yazma işlemlerinin çoğunun

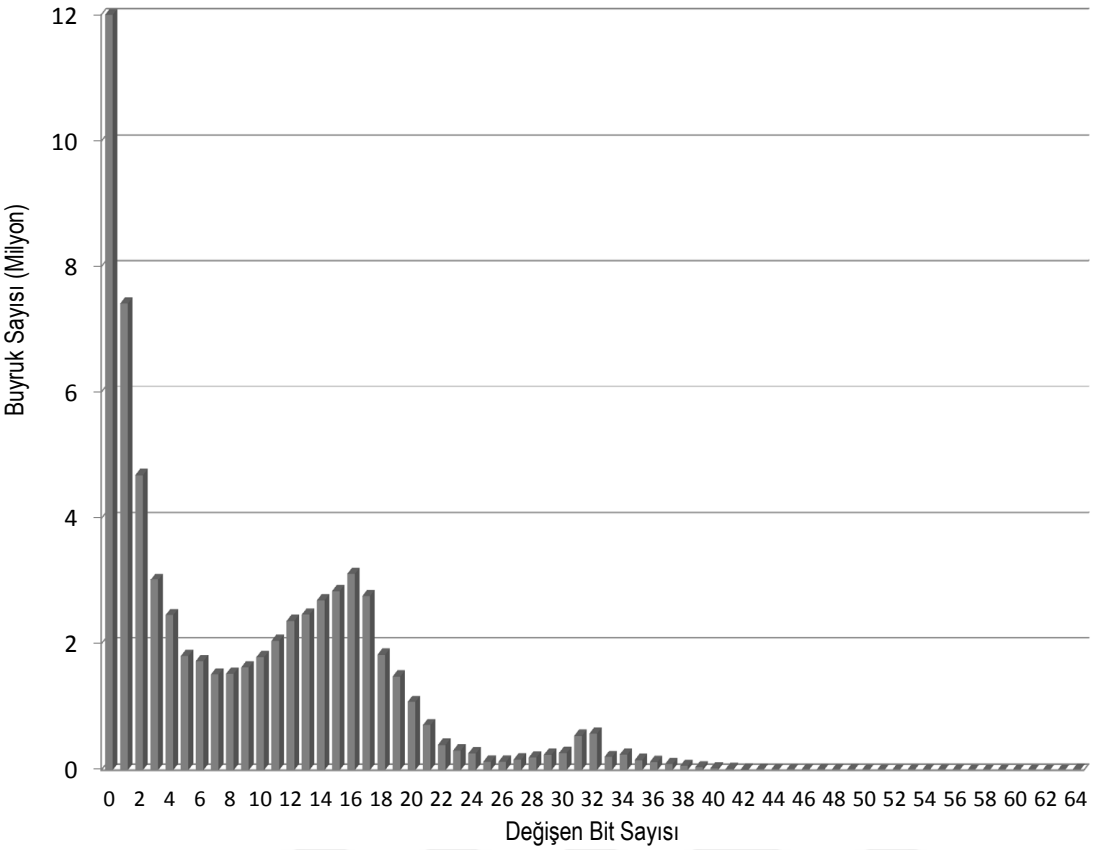


Şekil 4.1: 64-bit yazmaçların her gösterge program için yazmaç değiştiren buyruk başına ortalama değişen bit sayısı ve bütün gösterge programlar için ortalaması.

hedef yazmaçta sadece birkaç bitte değişime sebep olmasıdır. Bu ikinci durumun sebebi küçük sayılarla işlem yapan aritmetik buyruklarla FOR, WHILE gibi döngülerde sıkça kullanılan bir-artır (increment) veya bir-azalt (decerement) işlemlerinin yazmaç bitlerinin sadece küçük bir kısmında değişime sebep olmasıdır.

Örnek olarak, 4.1 çizelgesinde SPEC CPU2006 gösterge süitindeki programlardan *xalancbmk* programının buyruk seyrinden alınmış bir temel blok dizisi gösterilmiştir. Temel bloktaki ilk buyruk RDX yazmacının 64 bitinden 10 tanesini değiştirmiş, kalan üç buyruk ise hedef yazmaçlarının sadece 2 ya da 3 bitini değiştirmiştir.

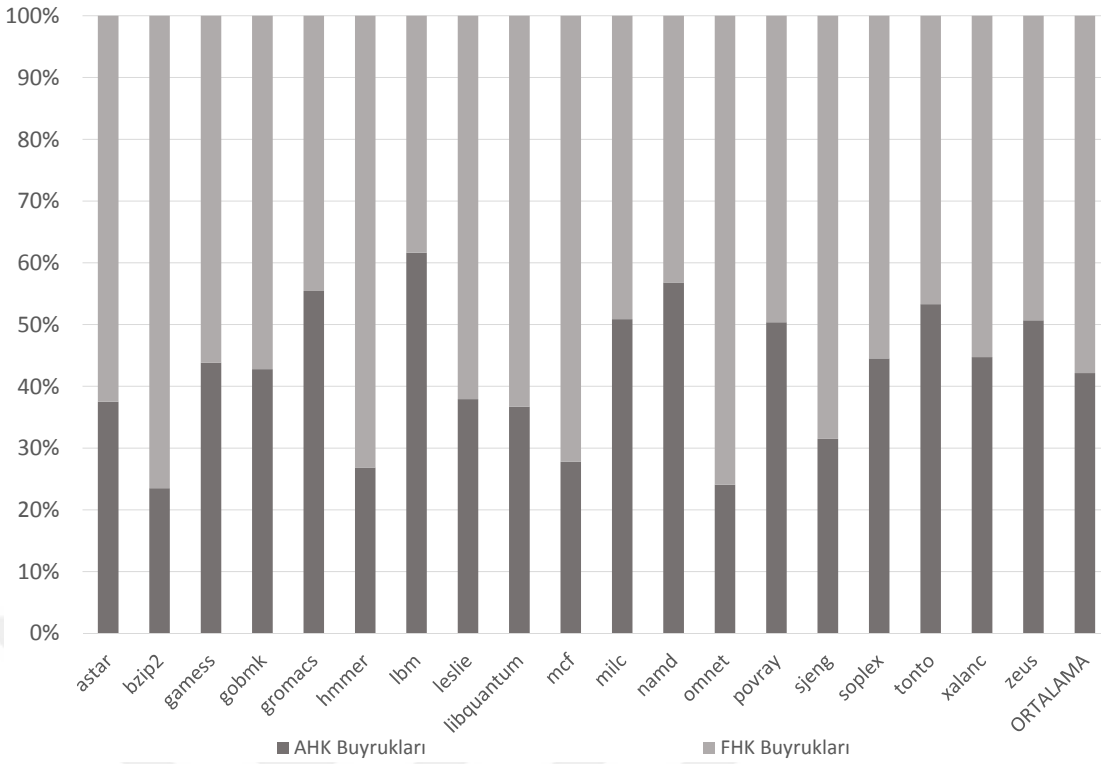
Önerilen fikrin potansiyelini değerlendirmek için SPEC 2006 gösterge programlarını 64-bit x86 mimarisinde benzetimledik. Şekil 4.1 buyruk başına 64-bitlik yazmaçtaki değişen bitlerin sayısını her bir gösterge program için göstermektedir. Şekilde de görüldüğü gibi, her buyrukta 64 bitin ortalama sadece 6.4 tanesinin değişmekte olduğunu bulduk. Bu sonuçlar düşünüldüğünde, mimari yazmaçların bitlerinin çoğu değişmiyorken yine de her bit için yazma işlemini gerçekleştirmek elbette enerji-etkin değildir. Bunun yanında belirli sayıda yazmaç bitini değiştiren buyrukların sıklığının da özelliklerini belirledik. Şekil 4.2 x-ekseninde buyrukların yazmaçta değiştirdiği bit



Şekil 4.2: Yürütüldüğünde belirli sayıda yazmaç bitini değiştiren buyrukların sayısı. Veriler bütün gösterge programları için ortalamayı gösterir.

sayılarını, y-ekseninde ise o miktarda bit değişikliğine sebep olan buyrukların kaç tane olduğunu göstermektedir. Bu şekilden görüleceği üzere, buyrukların önemli bir bölümü yazmaç bitlerini hiç değiştirmemekte, çoğunluğu ise sadece 7 tane veya daha az biti değiştirmektedir.

Bu çalışmadaki bir başka motivasyonumuz ise birçok aritmetik ve mantıksal buyruğun sonucunun yine o buyruğun kaynak yazmaçlarından birine yazılmasıdır, yani buyruğun bir yazmacı hem kaynak hem de hedef olarak kullanmasıdır. Bu şekilde olan buyrukları Aynı Hedef-Kaynak (AHK) buyruklar olarak adlandıracağız. Örneğin, `INC R1` buyruğu (`R1`'i bir artır) işlemcinin `R1` yazmacını okumasına, okunan değeri aritmetik-mantık biriminde (AMB) artırmasına ve sonucun yine `R1` yazmacına yazılmasına sebep olur. Böyle buyruklar genellikle hedef yazmacının bitlerinin çoğunu değiştirmez ve değişmeyen bitleri yakalamak oldukça kolaydır. Hedef yazmacın kaynak yazmaçla aynı olduğu buyrukların yüzdesi Şekil 4.3 ile gösterilmektedir. Yazmaç değiştiren buyrukların ortalama % 42'si AHK iken, geri kalan % 58'i ise hedef yazmacı ve kaynak yazmacı farklı olan buyruklardır. Bu çeşit buyruklar Farklı Hedef-Kaynak (FHK) buyruklar olarak adlandırılacaktır. Bu çalışmamızda FHK buyrukları için de devre seviyesinde sıfırlama mekanizması



Şekil 4.3: Aynı yazmacı hem kaynak hem hedef olarak kullanan buyrukların oranı çubukların alt kısmında, geri kalan hedef yazmacı kaynak yazmacından farklı olan buyrukların oranı ise çubukların üst kısmında gösterilmektedir.

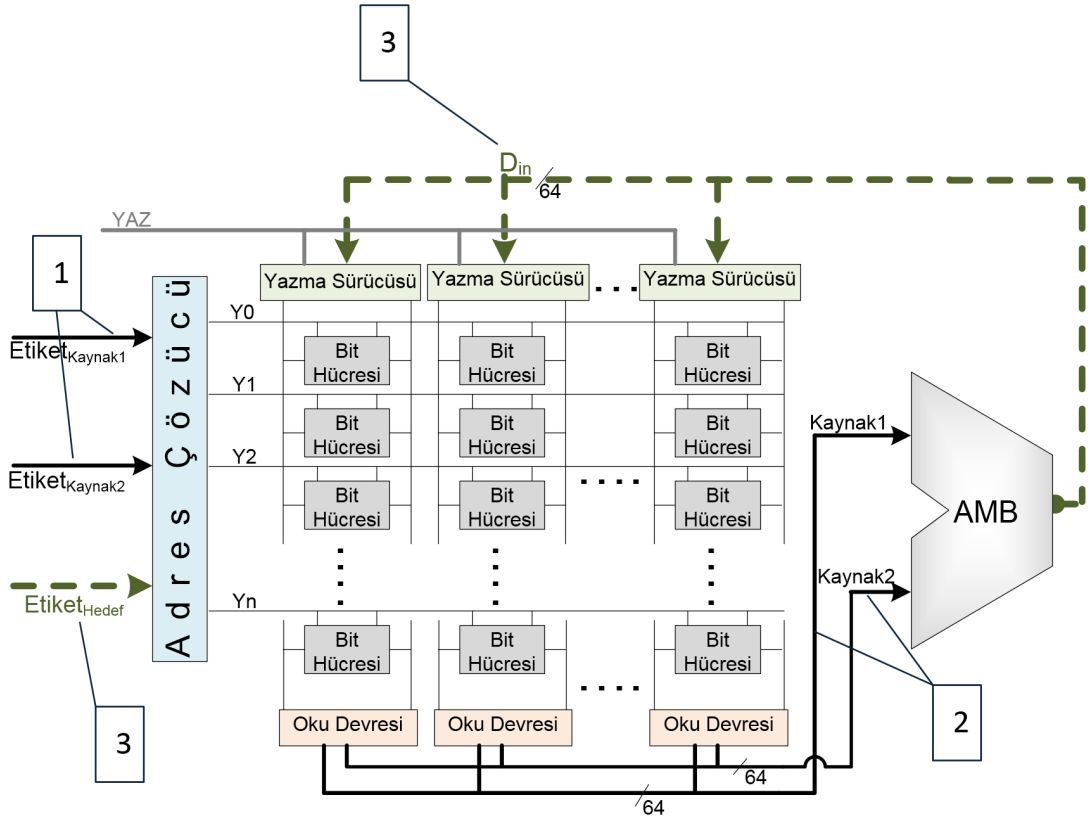
sağlayan ve oluşan sonucu hedef yazmaca bir fark vektörü olarak uygulayan bir güç-tasarruf metodu önermekteyiz.

4.2. Tasarım

Güncelleme temelli yazmaç öbeği için devresel ve mimari tekniklerin birleşiminden oluşan bir yöntem öneriyoruz. İlk alt-bölüm 4.2.1 içinde referans aldığımız ve iyileştirmelerimizi karşılaştırdığımız geleneksel sıralı mimarinin yazmaç öbeği yapısından bahsedeceğiz. Daha sonra gelen alt-bölümlerde ise önerdiğimiz tasarımı izah edeceğiz.

4.2.1 Referans mimari

Yazmaç öbekleri genellikle SRAM dizileri şeklinde gerçeklenir. Bir okuma işlemi için öncelikle bit-hatlarına ön-yükleme yapılır ve gerekli devrelerle yükleri eşitlenir. Yazmaçların adresleri buyrukların içine gömülü şekilde gelir ve yazmaç öbeğindeki adres-çözücü tarafından çözülür. Daha sonra seçilen hattın bitleri oku-devreleri (algılama-yükselteçleri) sayesinde algılanır ve AMB'ye okunan bitler iletilir.



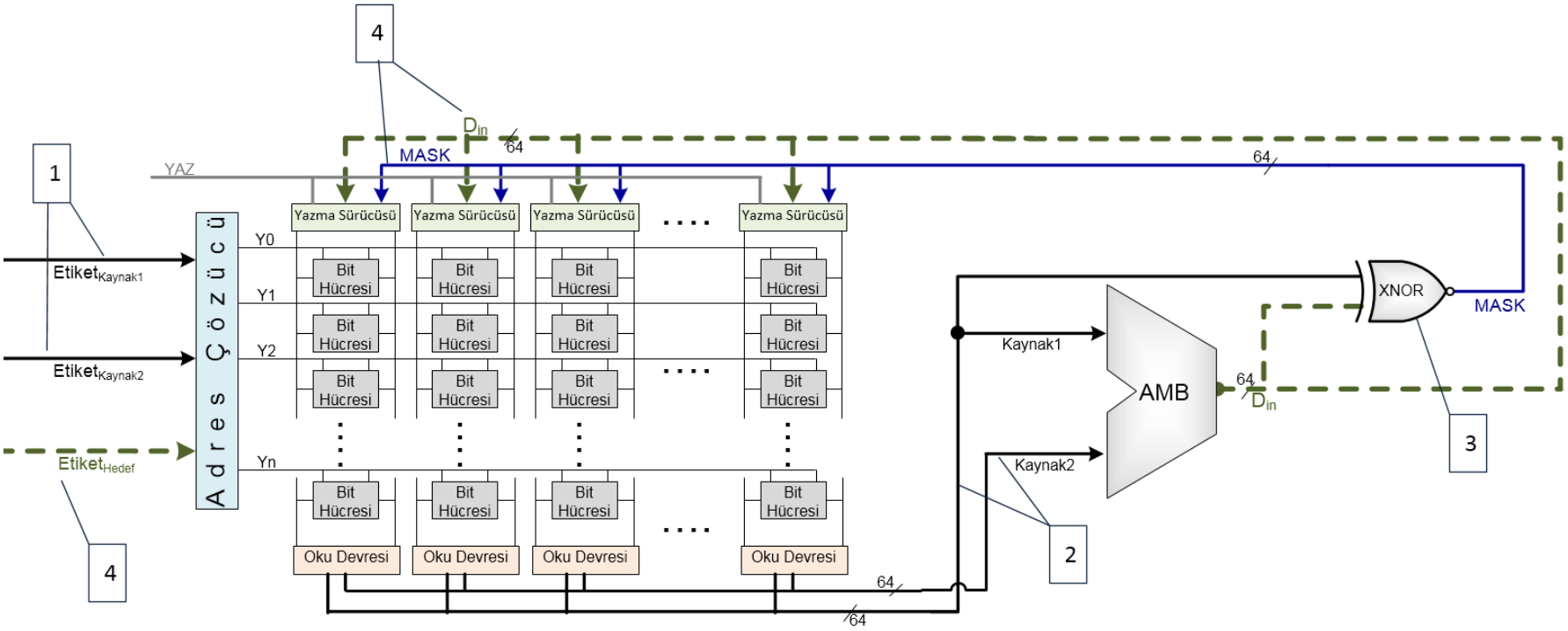
Şekil 4.4: Referans mimari olan sıralı bir işlemcinin kısmi bir diyagramı.

AMB'den çıkan sonucun geri yazılması gerektiğinde yazma sürücüleri üzerinden ilgili bit hatlarına yazılır. İlgili satıra yönelinmesi için hedef yazmacı yine adres-çözücü ile çözülür.

Şekil 4.4 yazmaç öbeği ve ilgili birimleri vurgulayacak biçimde sıralı bir işlemci mimarisinin bir kısmını göstermektedir. Bütün buyruklar önce adres üretecinin (veya diğer adıyla buyruk işaretçisinin) buyruk adresini buyruk önbelleğine iletmesiyle getirilir [63]. Buyruğun getirilmesinin ardından işlemde kullanılacak değerler kaynak yazmaçların etiketlerinin yazmaç indeksi olarak kullanılmasıyla yazmaç öbeğinden okunur (Şekil 4.4 içinde **1**). Aritmetik ve mantıksal buyruklar için, işlemde kullanılacak değerler AMB'ye iletilir (Şekil 4.4 içinde **2**) *YAZ* sinyalini kaldırarak ve yazmaç öbeğindeki hedef yazmaç adresini bildirerek yazma sürücüleriyle yazmaç öbeğine yazılır (Şekil 4.4 içinde **3**).

4.2.2 GÜNYET mimarisi

Şekil 4.5 GÜNYET mimarisinin ilk bölümünü göstermektedir. Bu kısım AHK buyruklar için enerji tasarrufu sağlamayı hedefler.



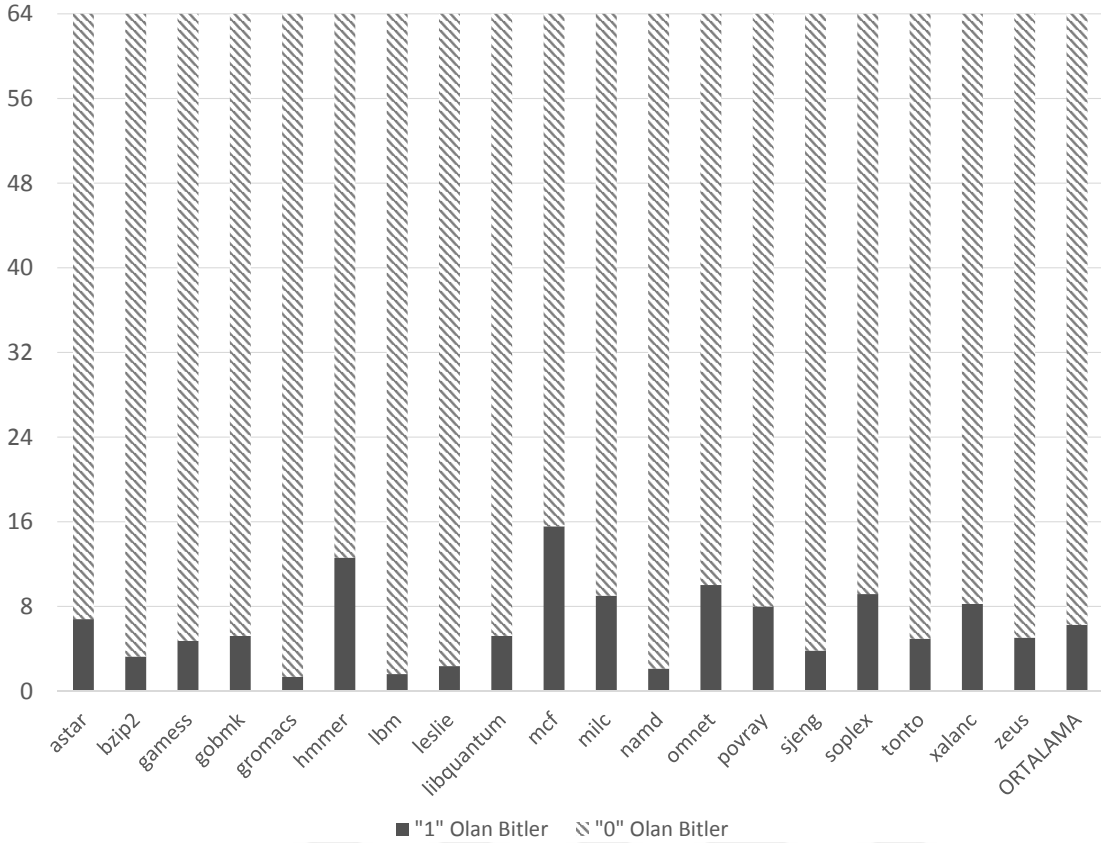
Şekil 4.5: GÜNYET AHK çözümü: `MASK` sinyali üretilen değerle önceki değerin farkını göstermek için oluşturulmuştur.

Kaynak yazmacın ve çıkan sonucun farkını bulmak için, referans mimariden farklı olarak, yazmaç değerinin ilk ve sonraki değeri arasında XNOR işlemini gerçekleştiren bir kombinasyonal mantık devresi vardır. Bu XNOR işleminin sonucu MASK olarak adlandırılır ve bu bilgi yazmaç öbeğine AMB sonucuyla birlikte iletilir. MASK sinyali işlem sonrasında hedef yazmacındaki değeri değişmemiş bitleri ifade eder. GÜNYET mimarisinde gerçekleşen operasyonların sırasını bir örnekle ifade etmek açısından tasarım *ADD* $R1, R1, R2$ (R1 yazmacının değerini R2 yazmacının değerine ekle, ve sonucu R1 yazmacına yaz) gibi bir buyruk üzerinden açıklanacaktır. Getirilen buyruk çözüldüğünde önce kaynak yazmaçların etiketleri yazmaç öbeğine verilir (Şekil 4.5 içinde **1**) ve R1 (Kaynak1) ile R2 (Kaynak2) yazmaçları AMB'ye yürütme aşaması için sürülür (Şekil 4.5 içinde **2**). Burada buyruğun AHK olduğuna dikkat edilmesi gerekir, zira R1 yazmacı aynı zamanda hedef yazmacıdır ($Etiket_{Kaynak1} = Etiket_{Hedef}$). Sonuç değeri ($D_{in}$) hazır olduğunda XNOR kapısına sürülür (Şekil 4.5 içinde **3**). XNOR kapısının diğer girdisi ise R1 yazmacının yürütmeden önceki değeridir ve bu değerin sonuç oluşana dek aradaki boru hattı aşamaları miktarınca flip-flop kullanılarak tutulması gerekir (bu flip-floplar karmaşıklığa sebep olmaması için şekilde gösterilmemiştir). XNOR işlemi ADD buyruğunun sonucunda R1 yazmacındaki değişmeyen bitleri "1" olarak çıktı verir. Daha sonra hedef yazmacın etiketi yazmaç öbeğine yazma indisi olarak yollanır, MASK çıktısı da SRAM dizisine toplama işlemi sonucuyla (D_{in}) birlikte iletilir (Şekil 4.5 içinde **4**). SRAM dizisi MASK sinyalinin "1" olduğu yazmaç bitlerinin elektrik yükünü tamamen boşaltmamak ve bu bitlere yazmamak, bu sayede de yazmaç öbeğinin enerjisinin büyük kısmını tasarruf etmek üzere tasarlanmıştır. GÜNYET mimarisinde dikkat edilmediği takdirde karışıklığa neden olabilecek bir özellik olarak belirtmek gerekir ki MASK bitleri yazmaç öbeğine kaydedilmemekte, sadece içeriği değişmeyen bitleri güncellemek amacıyla orada bulunan hatları hizmet dışı bırakmak için kullanılmaktadır.

Bu aşamaya kadar bahsettiğimiz yapı sadece AHK buyrukları için kullanılabilmektedir, bu nedenle de bu çözümü ***GÜNYET AHK Çözümü*** şeklinde adlandırmaktayız. Her ne kadar AHK buyruklarının miktarı farklı buyruk kümelerine sahip işlemciler arasında farklılık gösterse de bazı buyruk kümelerinin (Intel mimarisi [64] gibi) aritmetik ve mantıksal buyrukları çoğu buyruğun özellikle aynı yazmacı hem kaynak hem de hedef olarak kullanması şeklinde tasarlanmıştır. Intel mimarisi gibi çağdaş mimarilerde karmaşık buyruklar İndirgenmiş-Buyruk İşlemcilerdeki gibi buyruklar olan mikro-buyruklara dönüştürülmekte, mikro-buyruklarda ise hedef ve kaynak yazmaçlar farklı olabilmektedir. Buna rağmen sıralı işlemci mimarilerinde (örneğin Intel'in Atom işlemcilerinde) mikro-buyruk dönüşümü minimize edilmiştir ve bu mimariler çoğunlukla AHK olan orijinal buyrukları kullanırlar [4].

Önerilen güç tasarruflu yapıları kullanabilmek için FHK buyrukları daha farklı bir şekilde ele alınmalıdır. Bunun için basit bir çözüm şu şekilde olabilir: Yazmaç öbeğinden yazmacın eski değeri okunur ve yeni değerle `XNOR` işlemi yapılmak üzere mantıksal devreye sürülür, sonra da `XNOR` kapısının çıktısı, yani `MASK` sinyali yazmaç öbeğine GÜNYET AHK Çözümünde olduğu gibi iletilir. Fakat böyle bir çözüm yazmaç öbeğine ekstra bir okuma işlemi ve referans mimariye göre fazladan okuma enerjisi harcanması sonucunu doğurur. Referans mimaride yazmaçlarla işlem yapan FHK buyrukları için enerji sarfiyatının formülü $E_R + E_W$ şeklindedir. Bu formülde E_R yazmaç öbeğinden bir yazmacı okumak için gerekli olan enerjiyi, E_W de yazmaç öbeğindeki bir yazmaca yazmak için gerekli olan enerjiyi ifade eder. Eğer hedef yazmaç da kaynak yazmaçla birlikte okunur ve GÜNYET'in maskeli yazma mekanizmasıyla yazılırsa enerji sarfiyatının formülü bu kez $2E_R + E_{W.GUNYET}$ şeklinde olur. Bu formüldeki $E_{W.GUNYET}$ GÜNYET mimarisindeki maskeli yazma mekanizması kullanıldığında harcanan yazma enerjisidir. Bu durumda, bu çözümün kullanılması için ön-şart, eklenen fazladan okuma enerjisinin GÜNYET mimarisiyle tasarruf edilen enerji miktarından az olmasıdır. Aksi takdirde bu çözüm FHK buyrukları için fayda sağlamayacaktır. Bahsedilen bu çözümü ***Basit FHK Çözümü*** olarak adlandırmaktayız.

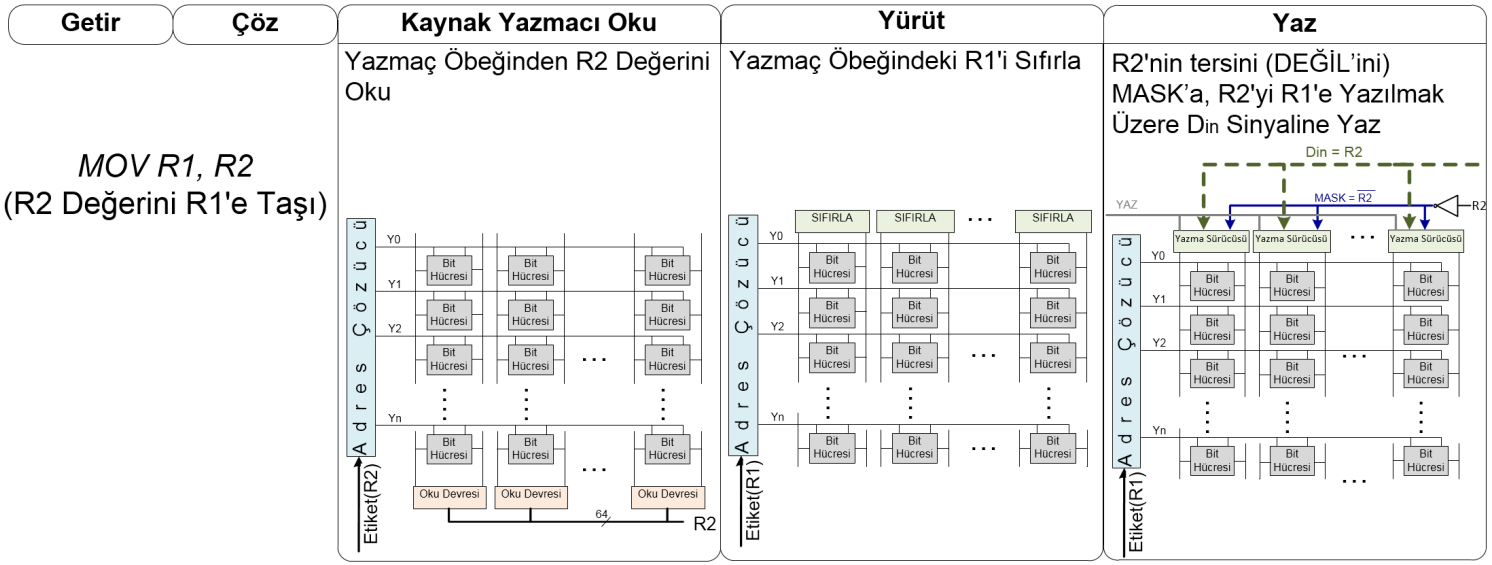
Basit FHK Çözümünün yanında, GÜNYET mimarisinde kullanılmak üzere orijinal bir başka mekanizma daha önermekteyiz. Bu yeni çözüm yazmaç öbeğine kaydedilen değerlerin bitlerinin çoğunun "0" değerini içermekte olduğu gerçeğine dayanmaktadır. Geçmiş araştırmalar da yazmaçlardaki bitlerin çoğunun "0" olduğunu göstermektedir [65]. Şekil 4.6 FHK buyrukları tarafından yazılan "0" ve "1" değerindeki bitlerin her bir SPEC 2006 gösterge programı için ortalama sayısını göstermektedir. Üstüste binmiş satırların alt segmenti 64 bitin "1" olan bitlerinin sayısını, üst segmenti ise "0" olan bitlerinin sayısını göstermektedir. Bütün gösterge programların ortalamasını aldığımızda, yazmaçtaki 64 bitin yalnızca 6,3 adet biti "1" olarak yazılmakta, geri kalan 57,7 biti ise "0" olmaktadır. Bu gözlem üzerine, çözümümüz yazmaca yeni değeri yazmadan önce tüm yazmaç içeriğini sıfırlayan bir devre tasarımına dayanmaktadır. `MOV` (taşı buyruğu) gibi FHK buyrukları için, hedef yazmaç sıfırlanır ve sonra sadece güncellenmesi gereken bitler (yani "1" yazılacak bitler) önerilen maskeli yazma mekanizmasıyla yazılabilir. Bu önerilen yöntemi tasvir etmek için Şekil 4.7 bir FHK buyruğu olan $MOV\ R1, R2$ (`R2` yazmacının değerini `R1` yazmacına taşı) buyruğunun sıralı işlemci boru-hattında ilerleyişini göstermektedir. $MOV\ R1, R2$ buyruğu boru-hattına geldiğinde `R2` değeri yazmaç öbeğinden okunur. Daha sonra, Yürüt aşamasında `R1` yazmacı yeni değeri bulunup yazmaç öbeğine yazılmadan önce, düşük güç tüketimi olan sıfırlama devresiyle sıfırlanır. Buyruğun Çöz aşamasında FHK buyruğu olduğu belirlendiğinden, hedef yazmacın kritik yolu



Şekil 4.6: FHK buyrukları tarafından yazılan 64-bitlik yazmaçların bit değerleri. Her sütunun alt kısmı 64 bit içinde "1" değeri olan bitlerin sayısını, üst kısmı da "0" değeri olan bitlerin sayısını gösterir.

etkilemeden Yürüt aşamasında sıfırlanması mümkün olmaktadır. Sıfırlama işleminin ardından, sonuç değerlerinin yazmaç öbeğine yazıldığı Yaz aşamasının başında `R1` yazmacı tamamen "0" bitlerinden oluşur, `R2`'nin DEĞİL kapısıyla ters çevrilmiş değeri `MASK` hattına sürülür ve yazılacak `R2` değeri de veri hattına (D_{in}'e) sürülür. D_{in} bitleri yazmaca ancak `MASK` bitleri "0" ise yazılacağından, `R2`'ye yazılacak değerin "0" olan bitleri maskelenecektir. Bahsedilen sıfırlama mekanizmasıyla FHK buyruklarını yazma yönteminde enerji harcanan kısımlar sadece Yürüt aşamasındaki sıfırlama ve değişen yani "1" olan bitlerin Yaz aşamasında yazılmasıdır. Önerilen bu çözümü ***GÜNYET FHK Çözümü*** olarak adlandırmaktayız.

Sıfırlama mekanizmasının yazmaç öbeğine eklenmesiyle, aynı yapıyı hedef yazmaca sıfır değerini yazan AHK buyrukları için de kullanma imkanımız oluşur. Geçmişte araştırmacılar tarafından ifade edildiği gibi bir çok uygulamada program yürütümü sırasında üretilen en yaygın değer sıfırdır [66] [19]. Önerilen hızlı ve enerji-etkin sıfırlama mekanizmasını kullanarak muhtemelen daha da fazla enerji tasarrufu sağlamak mümkün olacaktır.



Şekil 4.7: Sıfırlama mekanizmasıyla GÜNYET FHK çözümünün genel bir sıralı boru-hattındaki örneği. Getir aşaması `MOV R1,R2` buyruğunu buyruk belleğinden getirir ve Çöz aşaması da çözer.

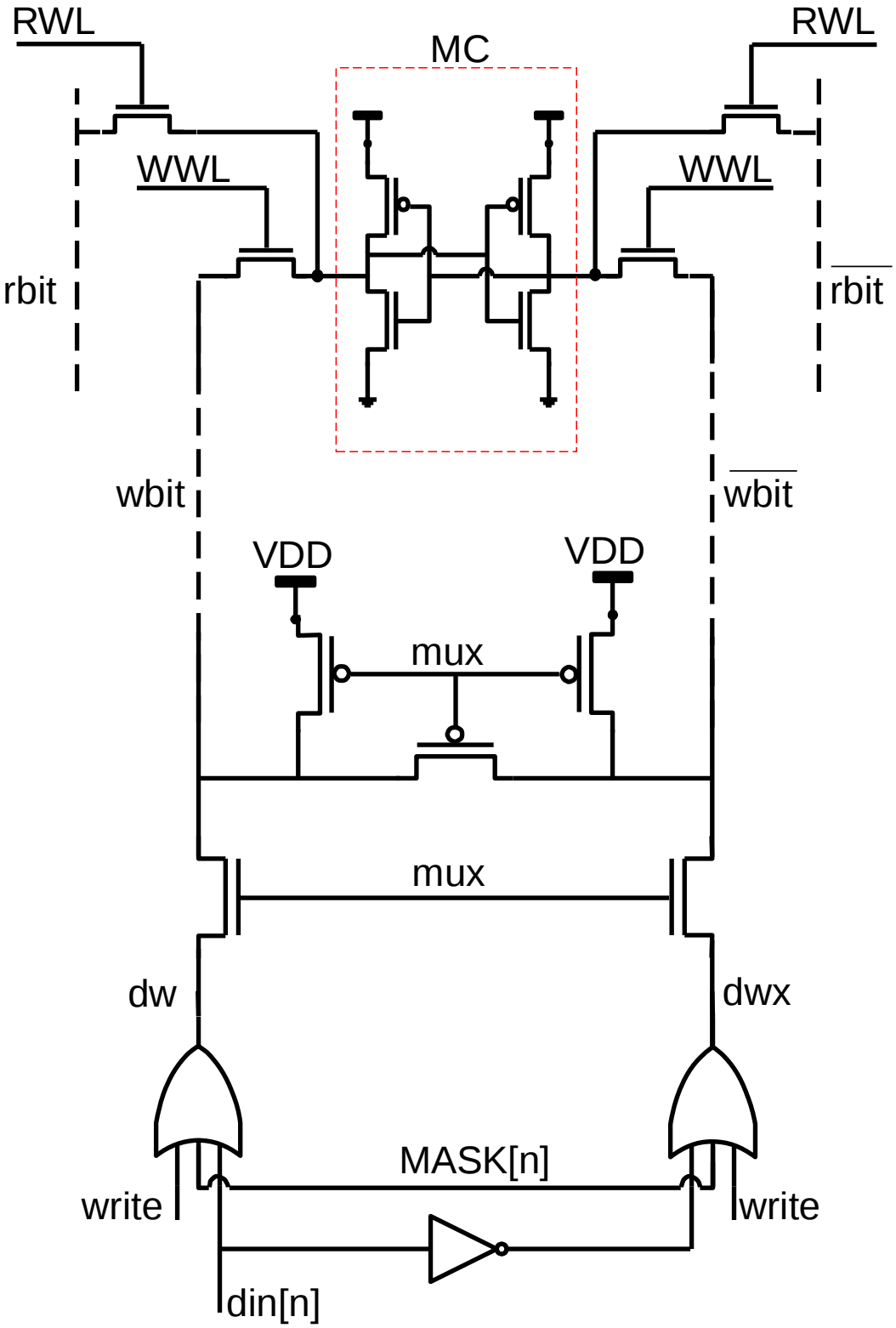
İşlemci mimarilerinin çoğunda sonucun sıfır olduğunu belirlemek ve sıfır-bayrağını kaldırmak için bir devre bulunmaktadır. Önerilen bu iyileştirme sıfır-bayrağı devresini ve sıfırlama devremizi kullanarak hayata geçirilebilir. AMB'den çıkan sonucun sıfır olduğu (mevcut sıfır-bayrağı devresini kullanarak) belirlendiğinde, FHK Çözümündeki sıfırlama devresi bu kez hedefi sıfırlamak için kullanılabilir. Burada sıfır-bayrağının kullanılmadığına, sadece sıfır-bayrağının kalkmasına sebep olan ve sıfır sonucunu algılayan devrenin kullanıldığına dikkat edilmeli, zira bazı buyruklar sıfır-bayrağını değiştirmemektedir.

4.2.3 Devre tasarımı

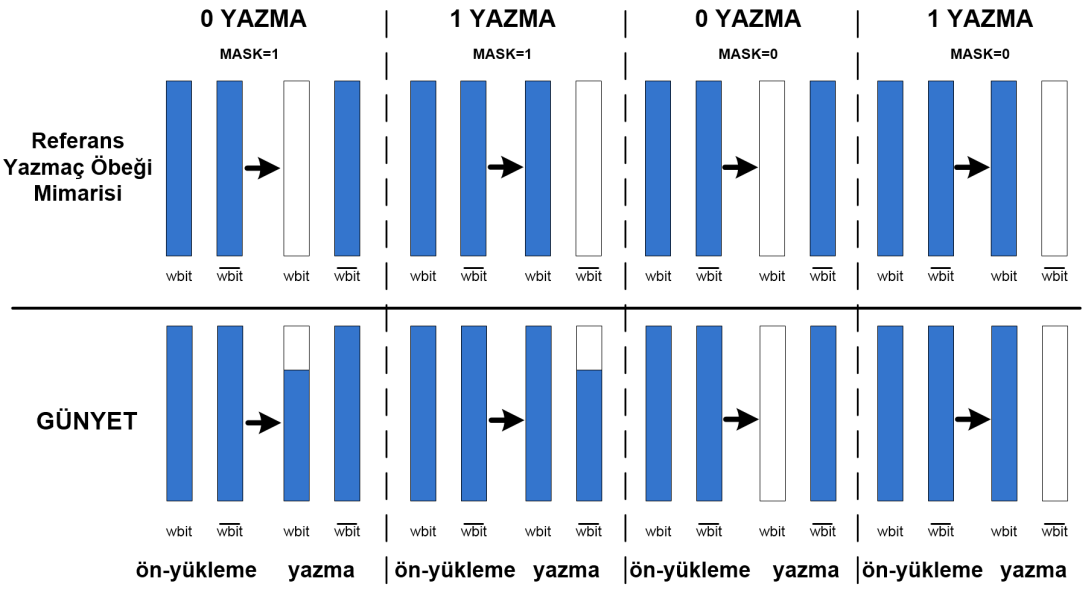
GÜNYET mimarisinde enerji tasarrufunun anahtar noktası hedef yazmacındaki değişmeyen bitlerin elektrik yükünü boşaltmasını engellemektir. Bu amaçla, Şekil 4.8 ile gösterildiği gibi, her yazma hattı sürücüsü MASK sinyali için fazladan bir girdiyle tasarlanmıştır. Şekildeki *write* sinyalinin aşağı çekilmesiyle yazma işlemi başlar ve bu da bit hattının (D_{in} verisinin ne olduğuna bağlı olarak) elektrik yükünün boşaltılmasına yol açar. Eğer bit değerinde değişiklik olmayacaksa MASK sinyali yazma hattının tamamen boşalmasına engeller. Bu sayede de seçilen sütunlardaki aynı değeri yazmaya çalışmaktan kaynaklanan güç tüketimi önlenmiş olur.

Şekil 4.8 devresindeki yazma işlemi kısaca şu şekilde gerçekleşir: Her yazma işleminden ($write = 0$) önce *wbit* ve $\overline{wbit}$ sinyallerine V_{DD} seviyesine kadar ön-yükleme yapılır. MASK sinyali "1" olduğunda, yani bit hücresinin aynı kalması durumunda hem *wbit* hem de $\overline{wbit}$ sinyalleri ön-yüklenmiş olarak kalır. WWL ve *mux* (sütun seç) sinyalleri "1" olduğunda ön-yükleme aşaması sona erer. Bu noktadan sonra yazma hatlarından birisi okuma işleminde olduğu gibi bit hücresi tarafından yavaş yavaş boşalacaktır. Normal şartlar altında, satır kapanmadan önce yazma hatlarından birisi V_{DD}'nin çok az bir kısmı kadar boşalacaktır.

Şekil 4.9 yazma hatlarının yük seviyesinin MASK sinyaline göre değişimini gösterir. MASK sinyali "0" olduğunda referans mimarideki yazmaç öbeği ile GÜNYET'in davranışı arasında fark yoktur. Ama MASK sinyali "1" olduğunda, GÜNYET mimarisindeki yazma hatlarından birisi kısmen boşalır, bu da bir sonraki ön-yükleme için harcanacak enerjiyi önemli miktarda azaltır. Boşalma seviyesi yazma hatlarındaki transistörlerin kesim durumunda (cutoff state) tutulmasını ($V_{DD} - V_{TH}$) garanti edecek seviyede tasarlanmalıdır; aksi takdirde transistörlerden *dw* veya *dwx* üzerinden fazladan akım geçebilecektir. MASK sinyalinden bağımsız olarak devrenin zamanlama davranışında herhangi bir değişiklik olmamaktadır. Diğer bir ifadeyle önerdiğimiz kısmi yazma yöntemi yazmaç öbeğine yazma zamanlamasını etkilemez.



Şekil 4.8: Sütun seçimi ve yazma enerjisini azaltmaya yönelik maskeli yazma mekanizmasının devre yapısı.



Şekil 4.9: GÜNYET AHK çözümü (Şekil 4.8 devresi için) ile referans mimarideki elektriksel yük durumlarının gösterimi. Mavi çubuklar tamamen yüklenmiş hatları, beyaz çubuklar ise tamamen boşalmış hatları ifade eder. GÜNYET mimarisi ilgili MASK biti "1" olduğunda tamamen boşalmayı engeller.

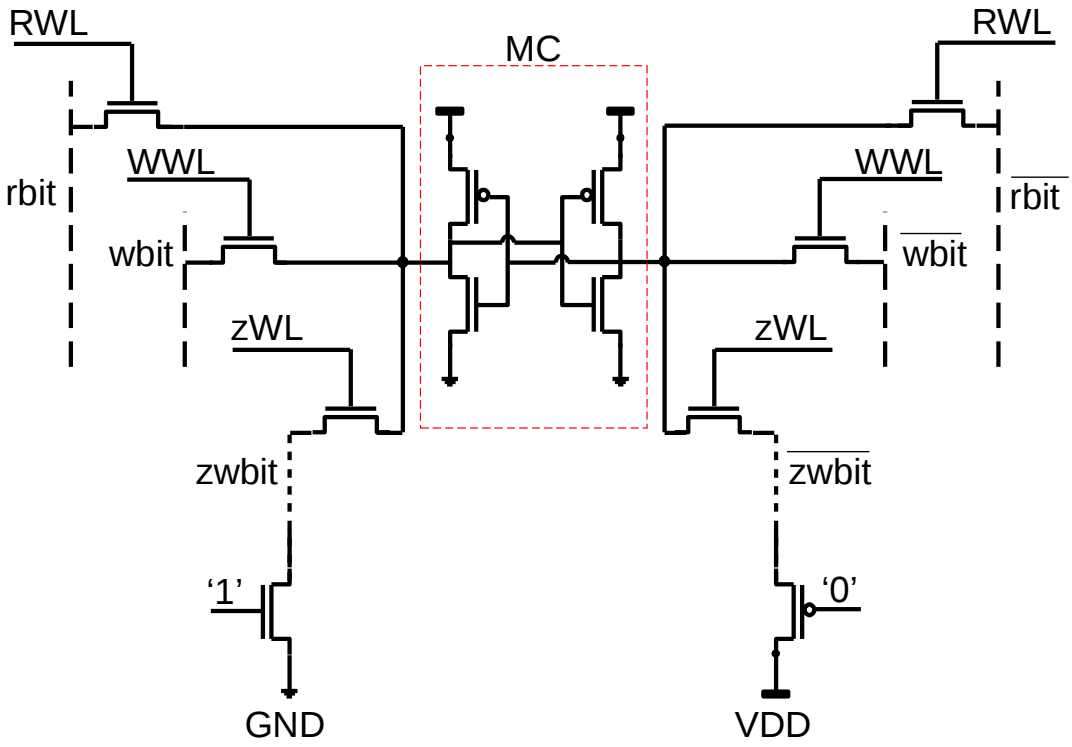
Yazma işleminden kaynaklı sarf edilen enerji yazma sürücüleri üzerinde ve çevresindeki devre üzerinde harcanan enerjinin toplamıdır (4.1). Yazma sürücülerinin enerjisinin ise iki bileşeni vardır: yazma bit-hattındaki harcanan enerji ve bit-hücresinde harcanan enerji (4.2).

$$E_{yazma} = E_{cevre} + E_{yazma-surucusu} \tag{4.1}$$

$$E_{yazma-surucusu} = E_{bit-hatti} + E_{bit-hucresi} \tag{4.2}$$

Önerilen tasarım 4.2 numaralı eşitlikteki her iki bileşeni de azaltır. Öncelikle Şekil 4.8 devresinde, yazma işlemi MASK sayesinde baskılandığında *mux* transistörleri (*dw*, *dwx*) ile ayrılmış yazma bit-hatlarının alt kısmı hiç boşalmaz ve üst kısım ($wbit$, $\overline{wbit}$) bit-hücresi tarafından bir seviyeye kadar kısmen boşalır. Bunun yanında bit-hücresinin durumunu muhafaza etmesi bit hücresinin değişimini önleyerek harcanan enerjiyi de azaltır.

FHK buyrukları için Şekil 4.8 ile gösterilen devre yazmaç içeriği bilinen bir değerle (örneğin sıfırlarla) değiştirilmedikçe herhangi bir enerji tasarrufu sağlamaz. Yazmacın içeriğini sıfırlamak MASK sinyalinin yine kullanılmasını sağlar. Bu amaçla "sıfır-yazma bit-hatları" diye adlandırdığımız ($zwbit$, $\overline{zwbit}$) ve hedef yazmacının tüm bitlerini sıfırlayan ikinci bir yazma portu öneriyoruz. Şekil 4.10 ile gösterildiği gibi bu hatlar sabit voltaj seviyelerine bağlıdır ve ancak yazmaç içeriğinin sıfırlanması

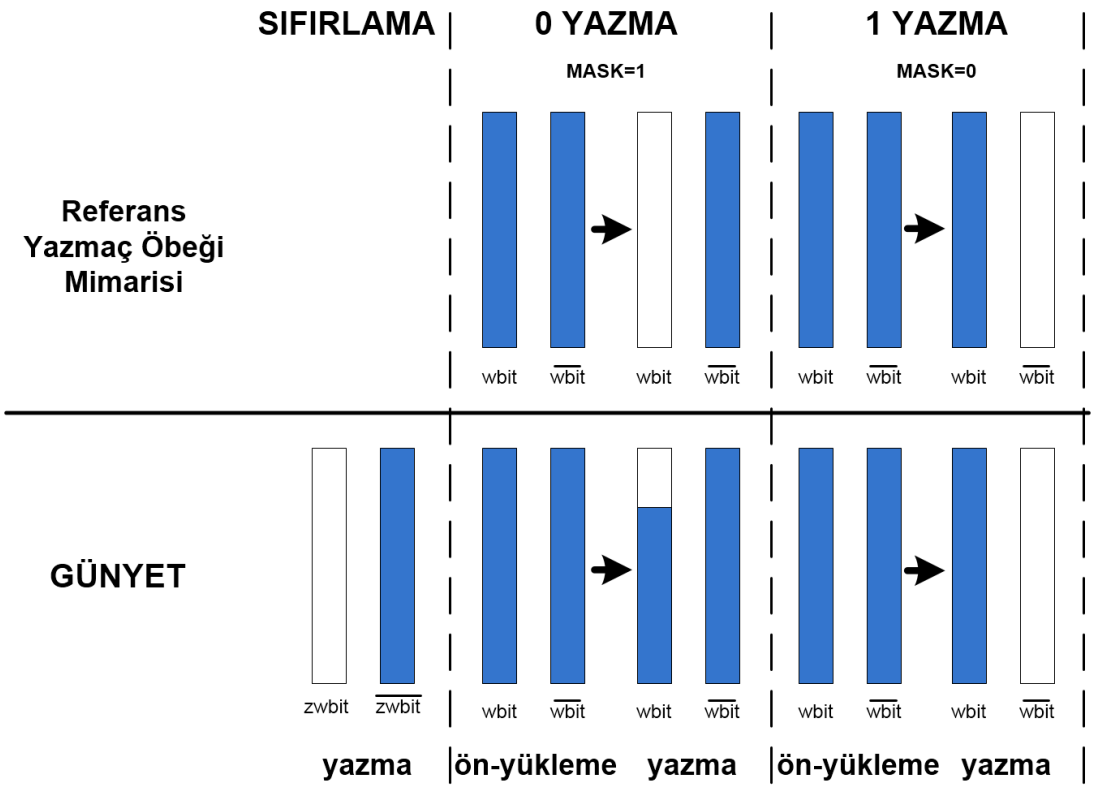


Şekil 4.10: Sıfırlama için değiştirilen bit-hücresi. *zwbit* ve $\overline{zwbit}$ yazma bit-hatları yazmacı sıfırlamak için eklenmiştir. Bu hedef yazmacın bitlerine çoğunlukla "0" yazılması yaygın olduğundan enerji tasarrufu sağlar.

gerektiği durumda kullanılırlar. Böylece anahtarlamadan kaynaklı güç harcaması tamamen elimine edilmektedir. Şekil 4.11 de GÜNYET FHK Çözümü için bit-hatlarının yük durumunu göstermektedir.

4.3. Benzetim Ortamı

GÜNYET'in etkisini karakterize etmek için bütün sistemin benzetimini yapabilen x86 işlemcilerine yönelik MARSSx86 mikro-mimari benzeticisinin değiştirdiğimiz bir versiyonunu kulandık [67]. MARSSx86 benzeticisi, değişik mimarileri taklit eden ve orijinal işletim sistemlerini çalıştırabilen bir sanal makine olan QEMU üzerine bina edilmiştir [68]. Deneyler için referans girdi kümesiyle x86-64 mimarisine göre O3 optimizasyon seviyesinde derlenmiş SPEC CPU2006 gösterge programları setini kullandık [69]. Benzetimi yapılan işlemcinin konfigürasyon parametreleri 4.2 numaralı çizelgede listelenmiştir. MARSSx86 benzeticisini, derlenen gösterge programları Linux Ubuntu çalışan 64-bit bir Intel Atom işlemci modelinde çalıştırmak için kullandık. QEMU'yu da her buyruk sonrasında işlemci durumunu elde etmek için kullandık. Deneylerimizde kernel modundaki buyrukları sonuçlarımıza dahil etmedik ve her gösterge programı 1 milyar buyruk ileri attıktan



Şekil 4.11: GÜNYET FHK çözümü (Şekil 4.10 devresi için) ile referans mimarideki yazma ve sıfırlama bit-hatlarındaki elektriksel yük durumlarının gösterimi. Mavi çubuklar tamamen yüklenmiş hatları, beyaz çubuklar ise tamamen boşalmış hatları ifade eder.

sonra 100 milyon kullanıcı buyruğu kadar çalıştırdık. Deneylerimizde hesaba kattığımız yazmaçlar genel amaçlı yazmaçlar, kayan-noktalı yazmaçlar ve XMM yazmaçlarıdır. Yazmaç sayıları, x86-64 mimarisine uygun olarak 16 genel amaçlı, 8 kayan noktalı ve 16 XMM yazmacı şeklindedir.

Devre seviyesinde yaptığımız benzetimler tasarlanan yazmaç öbeği modellerinde (32x64 ve 64x64) gerçekleştirildi. Bunun için Cadence Analog Design Environment programını UMC 90nm teknolojisiyle kullandık. Yazmaç öbeği benzetimleri ve ortalama güç tüketimi ölçümleri $1V$ V_{DD} geriliminde $27^{\circ}C$ çevresel sıcaklıkta gerçekleştirilmiştir.

4.4. Sonuçlar ve Değerlendirme

Deneylerimizde 32x64 ve 64x64 boyutlarında yazmaç öbeklerini modelledik ve benzetimini yaptık. Başarım karşılaştırması yapabilmek için de önerdiğimiz iki mekanizmayı, GÜNYET AHK ve FHK'yı ayrı ayrı uyguladık. Bir çok sıralı mobil işlemci 32 ve 64 yazmaçlı yazmaç öbeklerini kullandıkları için biz de deneylerimizde

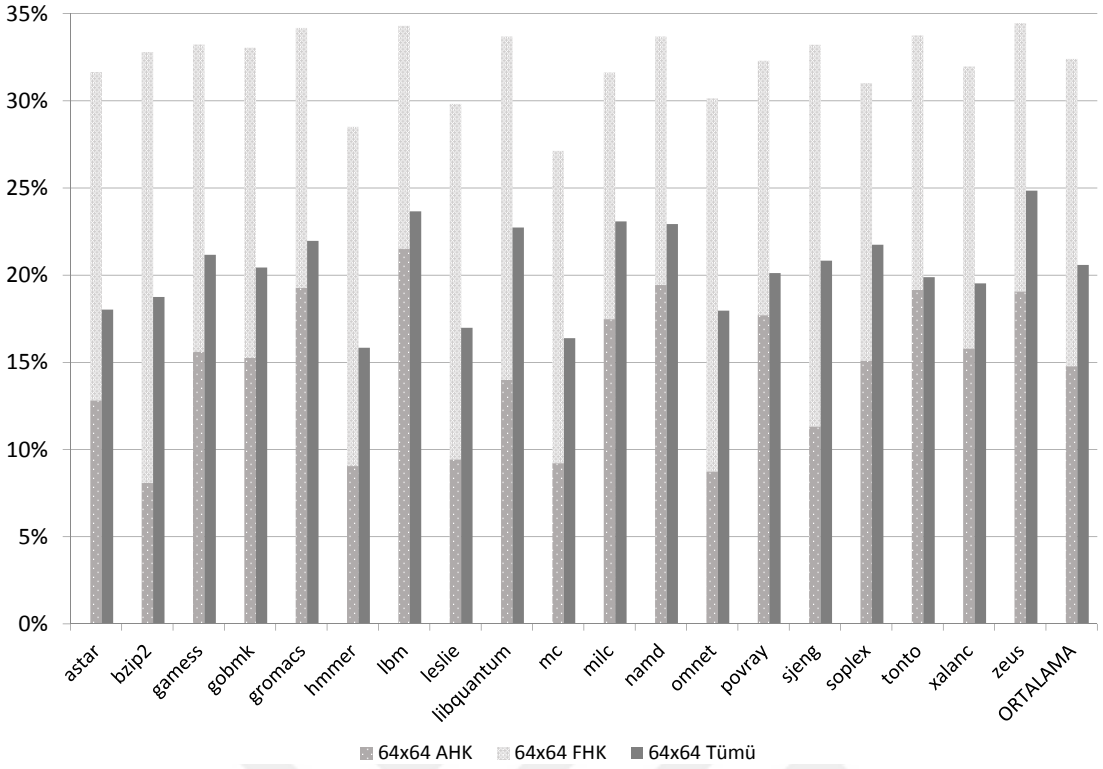
Çizelge 4.2: Benzetimi yapılan işlemcinin konfigürasyonu.

Parametre	**Konfigürasyon**
İşlemci	Sıralı, Atom gibi, x86 Buyruk Kümesi, 64 bit
Boru-Hattı Aşamaları Sayısı	16
Veri TLB / Buyruk TLB Boyutu	32 / 32
Dallanma Tahmincisi	Geçmiş ve Çift-modlu Birleşik (McFarling)
L1 Önbelleği	128KB Ayrık, geri-yazmalı, 8-yollu, 64B satır boyutu 2-vuruş gecikme
L2 Önbelleği	2MB, geri-yazmalı, 8-yollu, 64B satır boyutu, 5-vuruş gecikme

bu boyutlarla benzetim yaptık. Intel'in Atom işlemcisi simultane çok iş-parçacık yapısını 2 iş parçacığıyla kullanır ve bu iş-parçacıkları için ortak bir tamsayı yazmaç öbeği vardır. Ayrıca iş-parçacığı başına birer tane de kayan-noktalı yazmaç öbeği mevcuttur. Çeşitli MIPS işlemcilerinin [70] ve ARM'ın 64-bitlik sıralı işlemcilerinin [71] 32 genel amaçlı ve 32 kayan-noktalı yazmaçları vardır. Uzun veri yollarını önlemek için veriyi yakın tutma ihtiyacıyla birlikle ve bellekteki gecikmeyi önlemek amacıyla daha çok iş-parçacığı kullanmak suretiyle yazmaç öbeğinin boyutunun giderek büyüyeceğini tahmin ediyoruz.

Bölüm 4 kapsamında FHK çözümü için anlatılan ilk mekanizma, ancak ve ancak fazladan yazmaç okumak için harcanan gücün maskeli yazma sayesinde tasarruf edilen güçten düşük olması halinde faydalı olması beklenen **Basit FHK Çözümü**ydü. Devre seviyesinde yaptığımız benzetimlerde okumak için harcanan enerjisinin maskeli yazma sayesinde tasarruf edilen enerjiden 64 yazmaç için 2,18 ve 32 yazmaç için 4,95 kez daha fazla olduğunu gördük. Bu nedenle her iki yazmaç öbeği boyutunda da Basit FHK Çözümünü kullanmanın daha kötü bir netice verdiğini gözlemledik.

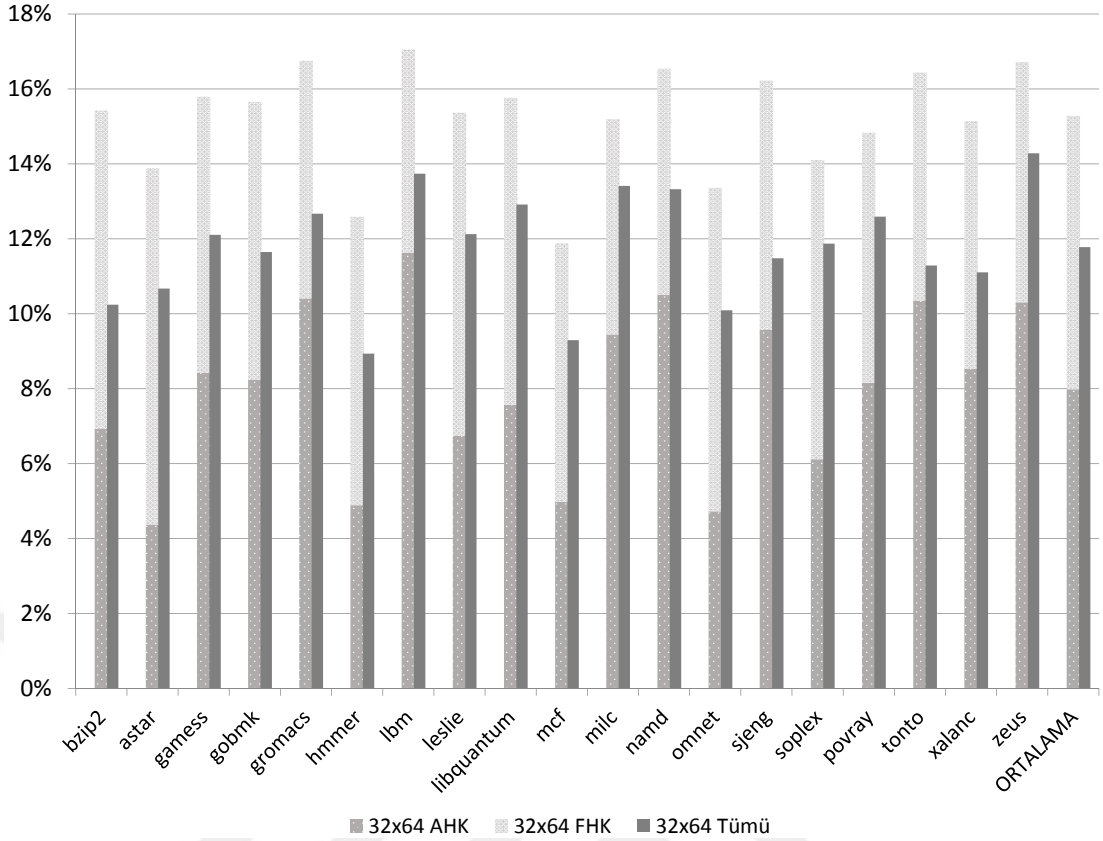
Şekil 4.12 ve Şekil 4.13 önerilen GÜNYET AHK ve FHK çözümlerinin referans mimariye göre sağladığı güç tasarrufunu oransal olarak ifade eder. Ortalama güç tasarrufu, gösterge programların benzetimlerinden elde edilen bit değişimi sonuçları temel alınarak hesaplanmıştır. GÜNYET FHK çözümünde yazma için harcanan güç



Şekil 4.12: 64 adet 64-bitlik yazmaçtan oluşan yazmaç öbeği için GÜNYET AHK ve FHK kullanımı sayesinde elde edilen güç tasarrufu. Her gösterge program için soldaki çubuk yazma operasyonundaki güç tasarrufunu gösterir. Soldaki çubuğun alt kısmı GÜNYET AHK sayesinde, üst kısmı da GÜNYET FHK sayesinde elde edilen güç tasarruflarını ifade eder. Sağdaki çubuk bütün yazmaç öbeğinde (okuma ve yazma operasyonlarının tamamında) GÜNYET sayesinde elde edilen toplam tasarrufu gösterir.

iki yazma operasyonundan oluşur: Sıfırlama ve maskeli yazma. Sıfırlama işleminde özel yazma hatları bulunduğundan harcanan güç sadece satır-çözme ve bit-hücreleri için harcanan güçten ibarettir - hatları yükleme boşaltma için güç harcanmaz. SPEC gösterge programlarından AHK buyruklarının tüm buyrukların % 42'sini, FHK buyruklarının da % 58'ini oluşturduğunu gözlemledik. Bütün deneylerin sonunda 64x64-bit yazmaç öbeğinde GÜNYET AHK ve FHK çözümlerinin ortalama yazma güç tasarrufunun **% 32,40**, ve 32x64-bit yazmaç öbeğinde yazma güç tasarrufunun **%15,28** olduğunu bulduk.

Bölüm 4 içinde bahsedilen en son iyileştirmenin güç tasarruf potansiyelini tahmin edebilmek için SPEC2006 gösterge programlarının benzetimlerini AHK buyruklarının ne kadarının yürütme sonucunda yazmaçlara sıfır yazdığını bulmak için tekrarladık. AHK buyruklarından yazmaç öbeğine sıfır yazanları bütün AHK buyruklarının % 4,54'ünü oluşturduğunu bulduk. Bu oran az olduğundan da bu iyileştirme yöntemi güç tasarrufunda çok büyük bir fark oluşturmamakta, GÜNYET



Şekil 4.13: 32 adet 64-bitlik yazmaçtan oluşan yazmaç öbeği için GÜNYET AHK ve FHK kullanımı sayesinde elde edilen güç tasarrufu. Her gösterge program için soldaki çubuk yazma operasyonundaki güç tasarrufunu gösterir. Soldaki çubuğun alt kısmı GÜNYET AHK sayesinde, üst kısmı da GÜNYET FHK sayesinde elde edilen güç tasarruflarını ifade eder. Sağdaki çubuk bütün yazmaç öbeğinde (okuma ve yazma operasyonlarının tamamında) GÜNYET sayesinde elde edilen toplam tasarrufu gösterir.

AHK ve FHK güç tasarrufunu 64 yazmaç için % 34,69'a, 32 yazmaç için % 18,21'e yükseltmektedir. Her ne kadar bu metot küçük bir avantaj sağlasa da, zaten yazmaç öbeğine FHK buyrukları için eklenmiş sıfırlama devresini bütün buyruklar için kullanmak fazladan bir yük getirmeyecektir.

Çalışmamız yazmaç öbeğindeki yazma enerjisini azaltmaya dönük olduğundan, GÜNYET mimarisiyle yazmaç öbeğinde elde edilen toplam enerji tasarrufunun miktarı yazmaç öbeğindeki bütün işlemlerin ne kadarının yazma işlemi olduğuna bağlıdır. Yazmaç öbeğinden yapılan okuma işlemlerini de yazma işlemleriyle birlikte dahil ederek GÜNYET mimarisinin toplam yazmaç öbeği enerji tasarrufunu hesapladık. SPEC 2006 gösterge programlarından elde ettiğimiz benzetim sonuçlarımız yazmaç öbeğindeki okuma işlemlerinin yazma işlemlerinden % 53 daha fazla olduğunu gösterdi. Bu farkın temel sebebi dolaylı bellek erişimi (bellek adresini yazmaçtan okumak) buyrukları ve sadece okuma yapan ama yazmaçları

değiştirmeyen karşılaştırma buyruklarıdır. Ayrıca aritmetik ve mantıksal buyruklar genellikle iki yazmacı okurken bir yazmaca yazmaktadır. Şekil 4.13 ve 4.12 ile gösterildiği gibi, GÜNYET mimarisini kullandığımızda yazmaç öbeğinin toplam güç tasarrufu 64 yazmaçlı yazmaç öbeği için % 20,59 iken 32 yazmaçlı yazmaç öbeği için % 11,78 çıkmaktadır. Gösterge programlarını tek tek incelediğimizde *zeus*, *lbm* ve *milc* gibi daha yüksek yazma oranı olan programlar 64 yazmaçlı yazmaç öbeği için % 25 seviyelerine kadar enerji tasarrufu sağlayabilmektedir.

4.5. Masraflar

4.5.1 Harcanan alandaki artış

Önerilen güncelleme tabanlı yapı yazmaç öbeğinde ve AMB'de bir miktar alan artışına sebep olur. Yazmaç öbeğindeki artış sabittir ve 2 tane NMOS transistörle MASK girdisi için eklenen transistörlerden oluşur. MASK girdisi için gerekli olan transistör sayısı yazma sürücüsü tasarımına bağlıdır ve mevcut çoğu tasarım için genellikle bir NMOS ve bir PMOS transistörden ibarettir. Sıfırlama devresi için bit-hücresinde yapılacak değişiklik her bit-hücresi için fazladan 2 tane NMOS geçiş transistörü ekler. Ancak bunun yazmaç öbeği mimarisinde sebep olacağı değişiklik çok azdır zira eklenen transistörler için ayrı bir satır hattı olacaktır.

AMB'de gerçekleşecek alan artışı AMB ile yazmaç öbeği arasındaki uzaklığa bağlıdır. Çünkü yazmaç öbeğine giden daha uzun metal hatlar MASK çıkışını düzgün şekilde iletmek için fazladan tamponlara ihtiyaç duyar, bu da bit başına alan masrafını belirler. Ayrıca sıfırlama mekanizması için eklenen basit DEĞİL kapıları ve çoklayıcılar da bir miktar alan artışına neden olur. Son olarak önerilen mimari, referans mimaride bulunmayan MASK hatlarını tanımladığından, doğal olarak yonga üzerinde hat sayısında bir miktar artış olur.

AMB ile yazmaç öbeği arasındaki uzaklık aynı zamanda önerilen güncelleme tabanlı mimarideki enerji tasarrufunu da belirler. MASK hatları uzadıkça veri iletimi için harcanan güç de artacaktır. Nitekim mevcut mobil işlemcilerde de bu birimler genellikle çok yakın tutulmuştur.

4.5.2 Gecikmedeki artış

Önerilen yöntem AMB'nin çıkışına XNOR kapıları eklediğinden bir miktar gecikme artışı oluşabilir. Bu gecikme MASK sinyalinin yazmaç öbeğine yazma işlemi için kritik hatta bulunmasından kaynaklanır. Fakat bu gecikme paralel hatlarda gerçekleştiğinden

yalnızca bir XNOR gecikmesi kadardır, bu da UMC 90nm kütüphanesinde yapılan deneylerde 50ps olarak bulunmuştur. GÜNYET mimarisinin sebep olduğu tek gecikme artışı da bu gecikmedir.

4.6. İlgili Çalışmalar

Yazmaç öbekleri için enerji tasarrufu çözümü öneren birçok çalışma yapılmıştır. Yazmaçlara hızlı erişim için genel kabul görmüş bir teknik olan yazmaç öbeği önbelleği aynı zamanda enerji etkinliği için de önerilmiştir [10] [11]. Bu teknikle yazmaç öbeğinin önüne iki adet küçük önbellek ekleyerek sık kullanılan yazmaç değerleri erişim için daha az enerji harcayarak okunur veya yazılır. Bizim tasarımımız depolama kapasitesini artırmayarak veya fazladan bellek eklemeyerek yazmaç önbelleği tasarımından farklıdır.

Bölümlenmiş yazmaç öbeği tekniğinde temel fikir büyük ve merkezi bir yazmaç öbeği yerine özel işlem birimleri olan daha küçük ve düşük güç tüketimli lokal yazmaç öbekleri bulunmasıdır [44]. Lokal yazmaç öbekleri enerji sarfiyatının yonga üzerinde dağılmasını sağlamaktaysa da toplam güç tüketimini azaltacağı kesin değildir. Bizim yötemimiz yazmaç öbeklerinde kullanılan enerjiyi modern mimarilerde zaten mevcut olan enerji tasarrufu potansiyelini kullanarak azaltır.

Yazmaç değerini öngörme-tahmin tekniği ise zaten kayıtlı verileri üreten buyrukları tahmin etmeyi önerir [12]. Bu metot sayesinde yazmaç verilerinin zamansal yakınlığı ve doğru tahmin durumunda işlemcinin yazmaç öbeğini okuma ihtiyacı kalmaz ve böylece enerji tasarrufu sağlanır. Bizim önerdiğimiz yöntemler ise yanlış tahmin durumunda fazladan zaman ve enerji kaybına neden olan öngörü-tahmin tekniklerini kullanmaz. Bunların yerine biz yazmaçlara yazma mekanizmasının özünde bulunan verimsizliği kaldırmaya çalışmaktayız.

Bir başka çalışma yazmaç öbeğindeki değerlerin benzerliğini kullanarak aynı sonuca sahip çok sayıda buyruğun aynı yazmacı göstermesini sağlamayı hedeflemiştir [19]. Fakat bunun bir çok arama ve karşılaştırma yüzünden çok fazla donanıma ihtiyaç duyması ve karmaşıklığa sebep olması nedeniyle yazarlar yazmaç öbeğinde sıklıkla bulunan "0" ve "1" için önerilerini sınırlamışlardır. Bahsedilen çalışma yazmaç öbeğini daha etkin kullanmayı amaçlıyorsa da, enerji-etkinlik bağlamında önerdiği faydalar GÜNYET mimarisiyle, aynı değeri aynı yazmaca yazmayarak ve sıfır yazan buyrukları sıfırlama devresiyle yazdırarak elde edilmektedir. Farklı olarak, GÜNYET mimarisi ayrıca "kısmen aynı" olan değerlerden kaynaklı da enerji tasarrufu yapmaktadır.

Aynı sütuna ardarda yapılan yazma işlemlerinde bit-hatlarına ön-yükleme yapmayı engelleyerek enerji tasarrufu sağlamaya çalışan bir mekanizma da bir çalışmada önerilmiştir [2]. Bu çalışmada yazarlar bit-hatlarının değişmesini algılayan mekanizmalar sunmuşlar ve aynı sütunda ardarda gelen yazmalarda bitlerde değişim yoksa ön-yüklemeyi engellemişler. Bitlerde değişim varsa, bu durumda bit-hattı çiftlerinde yük-paylaşımı yaparak ön-yüklemede harcanan enerjiyi azaltmayı önermişler. Bu çalışma da GÜNYET gibi yazmaç öbeğine yazma işlemindeki enerjiyi düşürmektedir. Fakat yazmaç öbeğinde en çok enerji tüketen kısım bit-hattı yazması olmasına rağmen bu çalışma yalnızca ön-yükleme enerjisini düşürmektedir. Bu çalışma aynı zamanda yazma gecikmesini % 16,2 kadar artırmaktadır. GÜNYET mimarisi minimal düzeyde (tek XNOR kapısı) bir gecikmeyle ön-yükleme enerjisini azaltmasının yanında, ayrıca değeri değişmeyen bitler için bit-hattı yazma enerjisini de azaltır.

Yazmaç değerlerini kıymetli ve kıymetsiz baytlara bölerek ve kıymetsiz baytların sayısını 2-bit eklemeyle tutan bir çalışma bu sayede yalnızca kıymetli baytları okumakta ve enerji tasarrufu sağlamaktadır [23]. Ancak bu çalışmada tanecik düzeyinin (granularite) bayt seviyesinde olması yazmaç öbeğinde bulunan enerji tasarrufu potansiyelini tam olarak kullanamamasına neden olmaktadır. Bunun yanında eklenen bitler ve karmaşık erişim mekanizmaları bu çalışmada elde edilebilecek azami enerji tasarrufunu daha da azaltmaktadır.

4.7. Sonuç

Bu tezin yazmaç öbeklerinin harcadığı güç bağlamındaki verimsizliğini engellemek üzerine olan bu kısmında, yazma işlemi için kullanılan enerjiyi azaltmak amacıyla yaptığımız mimari ve devre seviyesindeki tasarımlarımızı sunduk. Bu teknikler yazma işlemi sonucunda işlemci yazmaçlarının genellikle sadece bir kaç bitinin değiştiği gerçeğinden istifade etmektedir. AHK buyrukları için yazma bit-hatlarının tamamen boşalmasını azaltan GÜNYET AHK çözümünü önerdik. FHK buyrukları için de düşük güç harcayan sıfırlama devresini kullanarak hedef yazmacı sıfırlayan ve sadece "1" olan bitleri yazan bir yapı önerdik. Bu sıfırlama devresini aynı zamanda sonucu sıfır değeri olan buyruklar için de kullanmak üzere genişlettik. İşlemci üzerinde SPEC gösterge programlarını kullanarak yaptığımız deneylerde bu teknikleri kullanarak işlemci yazmaç öbeğinin yazma gücünü 64x64-bit boyut için % 32,40 ve 32x64-bit boyut için % 13,16 azalttık.

Bizim önerimiz her ne kadar işlemci yazmaç öbeğinin yazma enerjisini düşürmeyi amaçlasa da, yazmaç öbeğindeki okumaları da dahil ederek toplam enerjinin ne kadar

düştüğünü de ölçtük. Mimari ve devre seviyesinde benzetimler sonucunda topladığımız sonuçlardan gördük ki SPEC gösterge programlarını çalıştıran 64-bit işlemci üzerindeki GÜNYET mimarisi işlemci yazmaç öbeğinin gücünü 64 yazmaç için % 20,59 ve 32 yazmaç için % 11,78 azaltmıştır.

5. YAZMAÇLARDA MEVCUT BENZERLİKLERİ KULLANARAK YAZMAÇ ÖBEĞİNİ GEÇİCİ HATALARDAN KORUMAK

Radyasyon veya elektriksel gürültü nedeniyle oluşan geçici hatalar kalıcı bir hataya sebep olmaz, ancak sistemde önemli sorunlara yol açabilirler. Küçülen transistör kapı uzunlukları, daha düşük güç tüketimi için azalan kaynak ve eşik gerilim değerleri ve giderek artan işlemci frekansları elektronik sistemleri geçici hatalara karşı daha da yatkın hale getirir. Geçici hatalar sonucunda uygulamanın kapanmasına veya hatalı çıktılara sebep olan tek olaylık bozulma (single-event upset) ve sessiz veri çöküşü (silent data corruption) oluşur [6] [7].

Fiziksel yazmaç öbeği modern işlemcilerdeki en önemli bileşenlerden biridir. Fiziksel yazmaç öbeğine çok sık erişim gerçekleştiğinden dolayı, gerçekleşen herhangi bir hata kolaylıkla işlemcinin diğer kısımlarına yayılabilir, bu şekilde de sistemsel arızalara sebep olabilir. Bu nedenle fiziksel yazmaç öbeğinin etkin korunması her mimari için son derece kritik bir meseledir. Buna karşın fiziksel yazmaç öbeğinin başarımındaki az miktarlarda düşüş bile kabul edilemez görüldüğünden işlemci tasarımlarının çoğu fiziksel yazmaç öbeğini etkin bir koruma mekanizmasıyla donatmamıştır [72].

Yazmaç öbeğinde meydana gelen geçici hataların etkisini yok etmek için birçok çalışma yapılmıştır. Hata düzeltme kodu (ECC) yedekli bir şekilde veriyi kodlayıp okuma sırasında da kodlanan veriyi çözerek hatayı algılama ve düzeltme yapan geçici hatalara dayanıklılığı sağlamak için kullanılan bir metottur. Ancak ECC korumasının geleneksel (korumasız) bir yazmaç öbeğine göre enerji tüketimini 10 kat artırdığı raporlanmıştır [52]. ECC'nin önemli seviyede güç tüketiminden dolayı birçok yazmaç öbeği mimarisinde geçici hata algılamak için tek-bit parite koruması kullanılması tercih edilmiştir [53] [72]. Yaygın olan hatasız durumda 64-bit veri için ECC'de (7 adet parite biti hesaplanması gerektiğinden) tek-bit pariteye nazaran 4 kat daha fazla güç kaybı oluşur. Benzer şekilde, ECC'nin kodlayıcı/çözücü devresi de tek-bit pariteninkinden yaklaşık 4 kat daha büyüktür. Üçlü Modüler Artıklık (Triple Modular Redundancy - TMR) mikro-işlemcilerin geçici hatalardan korunması için sıkça kullanılan bir yöntemdi [73] ancak çok fazla alan ve güç masrafından dolayı artık modern mikro-işlemciler için tercih edilen bir yöntem değildir. Yazmaç öbeği ve veri önbelleği için koruma sağlamayı amaçlayan bir başka çalışma da dar yapılı

değerlerden arta kalan boşlukları kullanarak işlemcinin güvenilirliğini artırmayı önermiştir [38]. Bir başka araştırmacı grubunun yaptığı çalışmada ise geçici hataları gidermek için derleyici tarafından yönlendirilen bir yaklaşımla buyrukları ikileme ve yeniden dizme yöntemi kullanılmıştır [74]. Fakat bu yöntem kod boyutunu önemli ölçüde artırarak alttaki donanımı doldurmaktadır. Fiziksel yazmaç öbeğini koruyan nispeten daha az masraflı bir yöntem ise, her ne kadar daha fazla yazma operasyonu ve kullanılmayan yazmaçları seçen fazladan mantıksal devre eklemiş olsa da, donanım seviyesinde aktif yazmaçların kopyasını kullanılmayan yazmaçlarda oluşturur [8].

Bu tezin bu kısmında işlemcinin fiziksel yazmaç öbeğine, değerleri çoklamak suretiyle yedeklilik oluşturarak değil, içinde zaten bulunan benzerlikleri kullanarak etkin bir koruma sağlamayı öneriyoruz. Değerlerin benzerliklerini kullanan iki tane yöntem öneriyoruz: ilk yöntemimiz yazmaç öbeğindeki tıpatıp aynı değerleri kullanan "Kopyayla Düzeltme (KOD)" yöntemi, ikinci yöntemimiz ise sadece en alt baytta (EAB) farklılık gösteren değerleri kullanan "Yakın Değerlerle Düzeltme (YADED)" yöntemidir.

Fiziksel yazmaç öbeğinin güvenilirliğini sağlayan yöntemler için en temel gereklilikler işlemcinin kritik yolu üzerinde olduğundan gecikmenin ve yazmaç öbeği işlemci gücünün zaten % 16'sını kullandığından [9] harcadığı gücün neredeyse artmamasıdır. Bizim önerdiğimiz çözümler fiziksel yazmaç öbeğindeki değerlerin birçoğunun birbirinin benzeri olduğu ve birçok yazmaç değerinin zaten tıpatıp aynı değerlerinin başka yazmaçlarda bulunduğu gerçeğine dayanır. Yazmaç öbeğinde zaten mevcut olan bu yedekliliği, günümüz işlemcilerinde kabul edilemez gecikme ve güç masrafları ekleyen geçmişte yapılmış çalışmaların aksine, önemsiz miktarda masrafla çoklu bit hatalarını düzeltmek için kullanabiliriz. Yazmaç öbeğindeki geçici hataların çoğunu (tek-sayılı hataları) tek-bit parite kullanımıyla yakalayıp kopyalarına işaretçi tutarak yazmaçların kopyalarından elde ettiğimiz doğru değerlerle düzeltebiliriz. Yazmaç öbeğindeki kopyaların birçoğunun boru-hattındaki Yürüt aşamasında kaynak yazmaç değerini hedef yazmaca yazarak (MOV-taşı buyruğu gibi) oluşturulduğunu gözlemledik. Bu şekilde oluşan kopya değerler de kolayca algılanıp işaretlenebilir. Bu yöntem geçici hataların etkisini, tek-bit parite kadar kodlama masrafı ve ECC-paritesi kadar alan masrafıyla çok-bitli düzeltme yaparak azaltabilmektedir. Bu yöntem her ne kadar geçici hataları algılama ve düzeltme yaparak azaltsa da tamamen yok etmemektedir. Bununla birlikte kabul edilebilir Hatalar Arası Ortalama Zaman (Mean Time To Failure - MTTF) seviyelerine çok az güç ve gecikme masraflarıyla ulaşabilmesi için uygulanabilir bir alternatiftir.

Kopyası olan yazmaçların kapsamını artırmak amacıyla, bizim yaklaşımımız

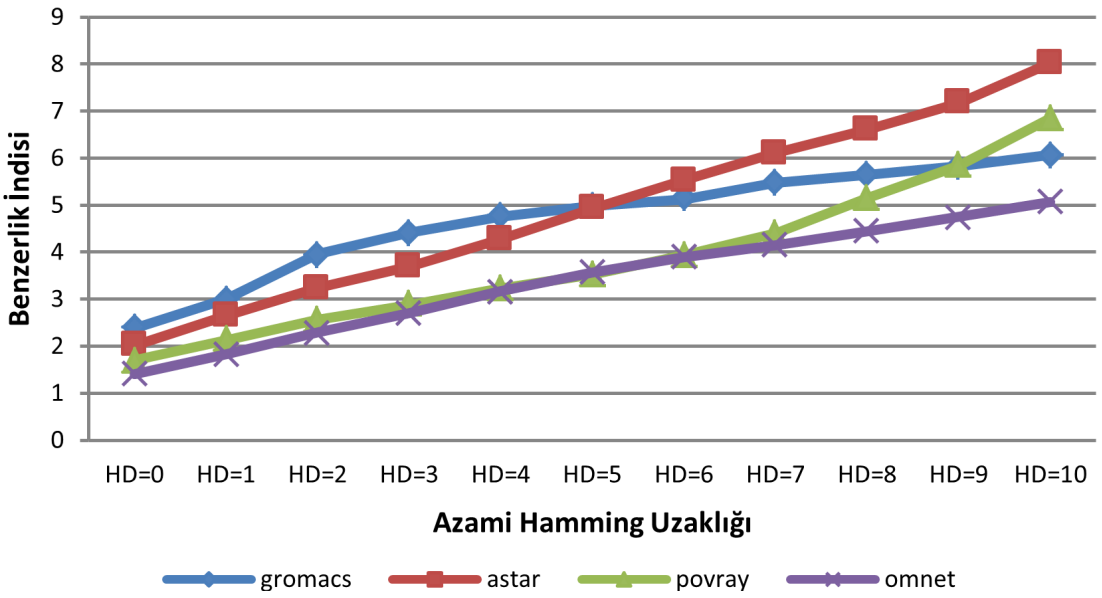
geçmişte önerilen ve yazmaç yedekliliğini değerleri çoklamak yoluyla yapay olarak oluşturan yöntemlerle birlikte de kullanılabilir. Bu nedenle, yöntemimize bir ilave olarak [8] çalışmasında anlatılan aktif kullanılan yazmaç değerlerini boşta duran yazmaçlara kopyalayarak, hata durumunda da kopya değerleri kullanan yöntemi eklemeyi önermekteyiz. Bu ilaveyi "Yazmaç İkileme Mekanizması" veya YİM diye adlandırıyoruz. Önerdiğimiz metodun kopya değerleri tutma ve geçici hata durumunda kullanma kabiliyeti YİM'in bizim yöntemimize adaptasyonunu kolaylaştırır. Yazmaç ikilemeyi bu şekilde kullanarak fiziksel yazmaç öbeğinde kopyası bulunan yazmaçların oranını artırmayı başardık.

Mimari benzetimler sonucunda yazmaç öbeğindeki benzer yazmaç değerlerin genellikle en alt baytının farklı olduğunu gözlemledik. Buna dayanarak KOD yöntemimizi benzerlikleri kullanmak üzere değiştirerek yazmaç öbeğini geçici hatalara karşı koruyan "Yakın Değerlerle Düzeltme" veya YADED yöntemini tasarladık. Bu yöntem tıpatıp aynı olmayan ama farkları sadece en alt baytlarında olan yazmaçları geçici hatalara karşı korumaktadır. Bu yakın değerlerle tıpatıp aynı olan değerler yazmaç öbeğinin büyük çoğunluğunu kapsamaktadır ve bize birçok hata düzeltme fırsatı sunmaktadır.

Bu kısmın kalanında bölümler şu şekilde düzenlenmiştir. Bölüm 5.1 çalışmamızın motivasyonunu sunar. Bölüm 5.2 önerdiğimiz tasarımı açıklar. Bölüm 5.3 çalışmada yaptığımız deneyleri anlatır. Bölüm 5.4 sonuçlardan ve çıkarımlarımızdan bahseder. Son olarak da bölüm 5.5 bulguları özetleyerek bu tezin bu kısmını sonuçlandırır.

5.1. Motivasyon

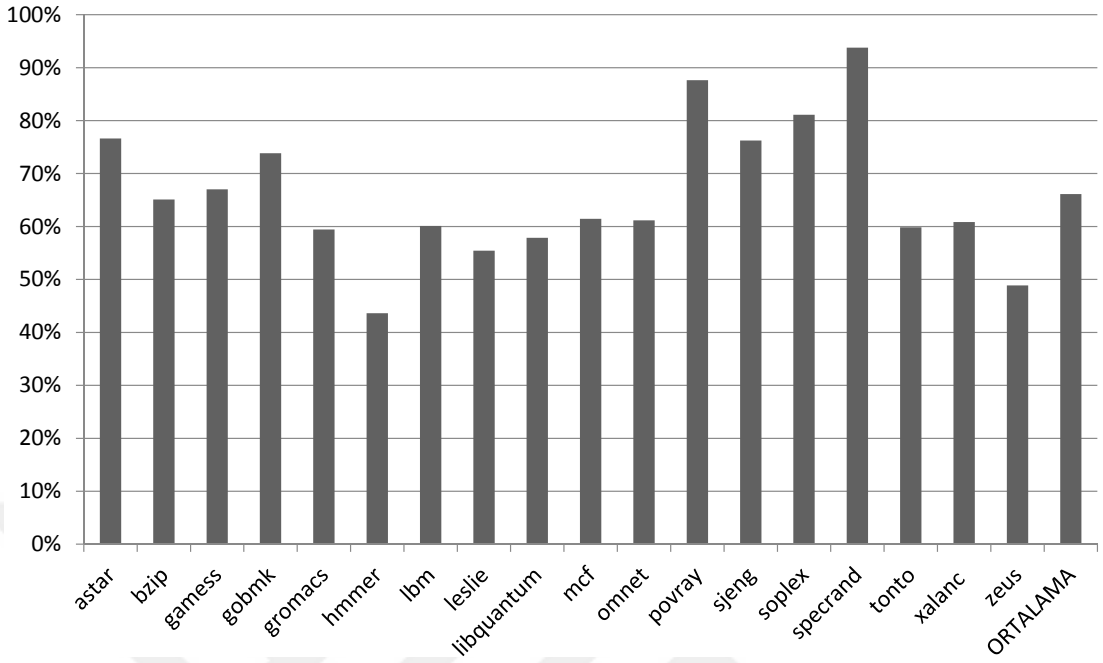
Yazmaçlara yazan birçok buyruğun yazmaçların sadece birkaç bitini değiştirdiğini bildiğimizden yazmaç öbeğinde bulunan yazmaçların çoğunun birbirine yakın değerler olduğunu düşündük. Yazmaç öbeğindeki değerlerin benzerliğini kontrol etmek için yazmaçlardaki değerlerin arasındaki Hamming uzaklığını [75] ölçmeye karar verdik. Yazmaçlar arası benzerliğin varlığını niceliksel olarak tanımlayabilmek için de "benzerlik indisi" diye bir metrik tanımladık. Bu metrik ortalama kaç tane aktif yazmacın aynı azami Hamming uzaklığına sahip grupta bulunduğunu ifade eder. Belirli bir azami Hamming uzaklığı değeri için, bir gösterge programın *benzerlik indisi* ne kadar yüksekse o gösterge programın çalışması sırasında yazmaç değerleri arasında o kadar benzerlik vardır. Şekil 5.1 4 adet SPEC2006 gösterge programının 0-10 arasındaki Hamming uzaklıkları için benzerlik indislerini gösterir (veri toplamak için 1 milyar buyruk atlanarak 10 milyon buyruk çalıştırılmıştır). Bir gruptaki ortalama yazmaç sayısı azami Hamming uzaklığı 0 olduğunda (değerlerin aynı



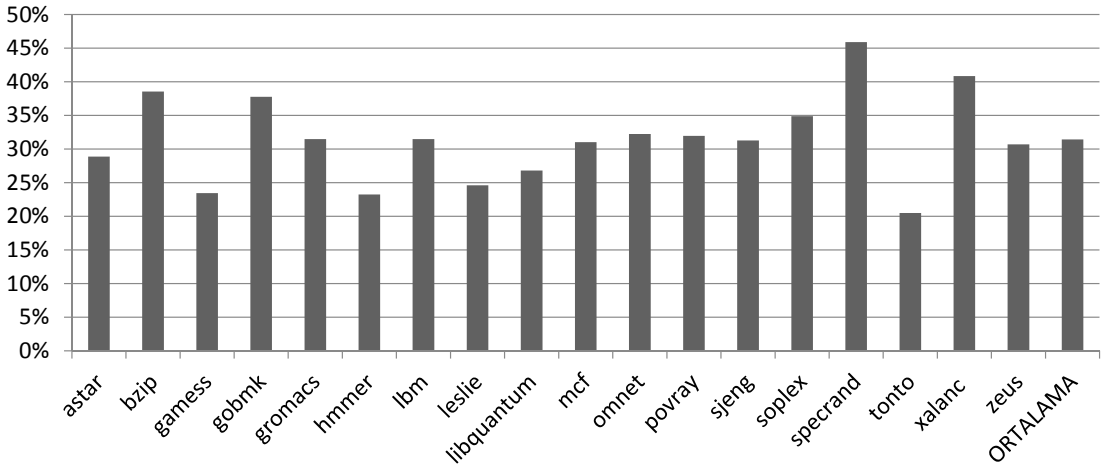
Şekil 5.1: 4 adet SPEC2006 gösterge programının 0-10 arası azami Hamming uzaklığı için benzerlik indisleri.

olduğu durum) 2.5'e, Hamming uzaklığı 5 olduğunda 5'e, Hamming uzaklığı 10 olduğunda da 8'e kadar çıkmaktadır. Hatalara karşı güvenilirlik için değerlerin aynı olduğu, yani Hamming uzaklığının 0 olduğu durumu kullanmaya karar verdik.

Yazmaç öbeğindeki aynı olan değerler üzerine yaptığımız çalışmada, her saat vuruşunda aktif yazmaçların ortalama % 66,1'inin fiziksel yazmaç öbeğinde zaten kopyalarının mevcut olduğunu gördük. Kaç tane yazmacın fiziksel yazmaç öbeğinde kopyasının bulunduğunu bulmak için SPEC2006 gösterge programlarının bir bölümünü, 160 yazmaçlı yazmaç öbeğine sahip (Intel'in kritik görevli sistemler için tasarladığı Itanyum Poulson işlemcisinde olduğu gibi) 64-bit bir x86 mimarisinde 1 milyar buyruk atlattıktan sonra 100 milyon buyruk kadar çalıştırdık. Şekil 5.2, değeri yazmaç öbeğindeki bir başka yazmaçta bulunan yazmaçların ortalama oranını göstermektedir. Yazmaç değerlerinin kopyalarının yüksek miktarda olduğu gözleminin ardından bunun sebebini analiz ettik. Her buyruğun yazmaç okuma ve yazma işlemlerini kontrol ettiğimizde, kopyaların % 31,4'ünün buyruğun hedef yazmacının değerinin kaynak yazmacıyla (ya da kaynak yazmaçlarından biriyle) aynı olmasından dolayı oluştuğunu gördük (Şekil 5.3); yani bir buyruk bir yazmacın değerini okur ve aynı değeri yazmaç öbeğindeki bir başka yazmaca yazar. Bu biçimdeki kopyaların yüksek oranı yazmaç taşıma ve sıfırla toplama veya çıkarma işlemlerinden kaynaklanmaktadır. Oluşmuş diğer kopyalarsa push-pop işlemleri (yığıt göstergesini bir artırıp bir azalttığı ve ardından yığıt göstergesini başka fiziksel yazmaçlara koyarak kopya değerleri oluşturduğundan) ve aynı bellek yerini farklı fiziksel yazmaçlara yükleyen dizi işlemeyle ilgili buyruklardan kaynaklanır.



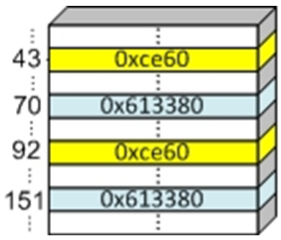
Şekil 5.2: 18 SPEC2006 gösterge program için 160-yazmaçlı bir tamsayı yazmaç öbeğinde kopyası olan aktif yazmaçların ortalama oranı. Her saat vuruşundaki oranın çalıştırılan 100 milyon buyruk için ortalaması alınmıştır.



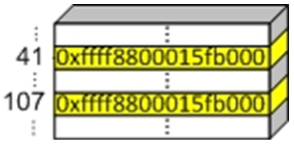
Şekil 5.3: Aynı değerli yazmaçlar arasında, bir yazmaçtan okuyup diğer yazmaca yazan bir buyruktan dolayı oluşmuş olanların oranı.

Çizelge 5.1: SPEC2006 programlarından yazmaç kopyası oluşturan mikro-kod bölümleri.

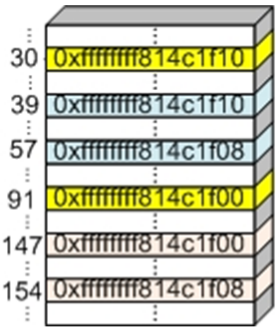
```
TAŞIMA (Astar'dan)
 mov rsp(pr151),rbp(pr70)
 ...
 mov rsi(pr92),0xce60
 ...
 mov rsi(pr43),0xce60
```



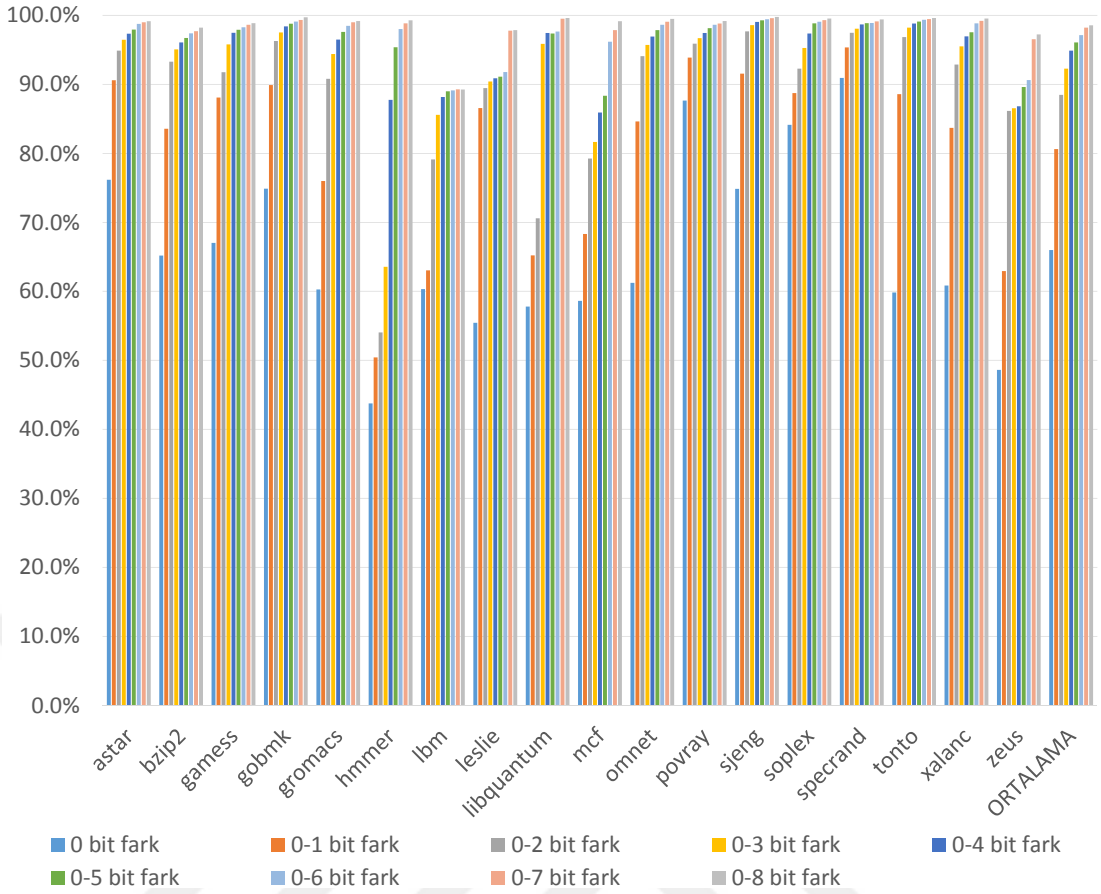

```
AYNI BELLEK ADRESİNDEN YÜKLEME (Specrand'dan)
 ld  r6(pr107),[r14,24]
 ...
 ld  r6(pr41),[r14,24]
```




```
PUSH-POP YIĞIN İŞLEMLERİ (Povray'dan)
 sub rsp(pr154),rsp(pr39),8
 ...
 sub rsp(pr91),rsp(pr154),8
 ...
 sub rsp(pr67),rsp(pr91),32
 ...
 ...
 add rsp(pr30),rsp(pr67),32
 ...
 add rsp(pr147),rsp(pr30),8
 ...
 add rsp(pr57),rsp(pr147),8
```



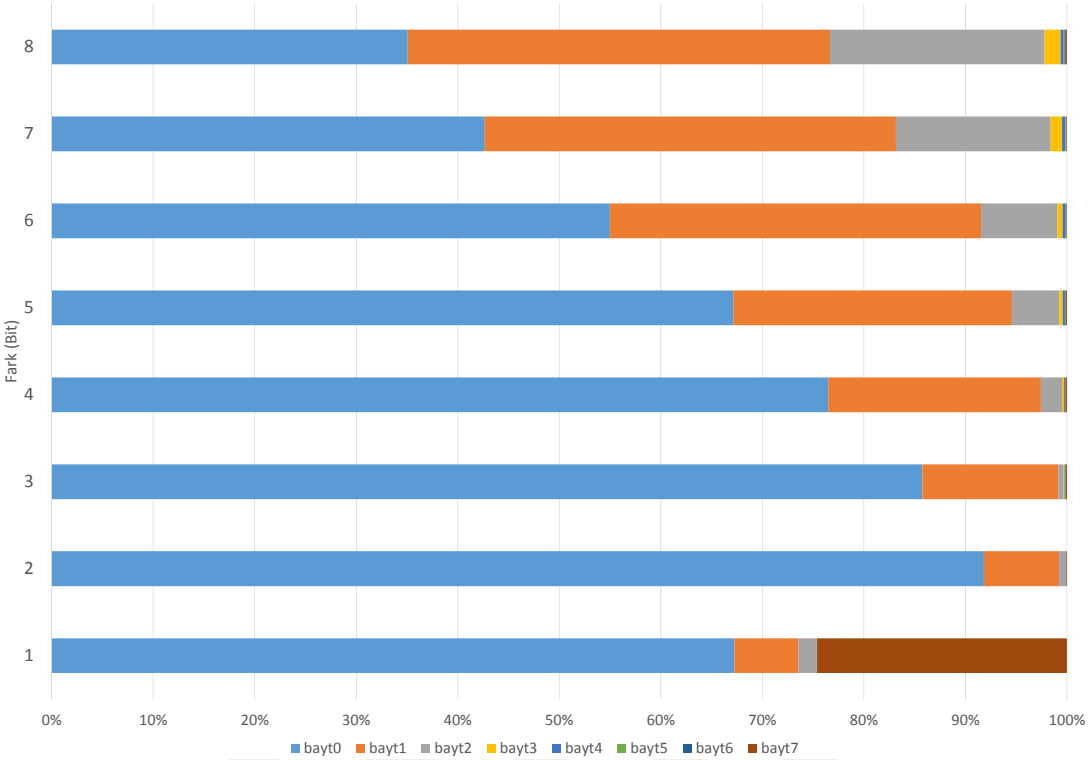
SPEC2006 gösterge programlarından bazılarından (*astar, specrand, povray*) alınan mikro-kod örnekleri Çizelge 5.1 ile gösterilmiştir. Yazmaçların yeniden adlandırılması tamamlandıktan sonra, mimari yazmaçlar fiziksel yazmaçlara eşlenmiştir. Fiziksel yazmaçlar Çizelge 5.1 içinde "*pr*" ön-ekiyle ve mimari yazmaçlardan sonra parantez içinde gösterilmiştir. Çizelgede her kod bölümünün sağında fiziksel yazmaç öbeğinin kod bölümü tamamlandığındaki son hali gösterilmiştir. Çizelgedeki ilk kod bölümü kopyaların taşıma (MOV) buyruklarıyla nasıl oluştuğunu gösterir. Çizelge 5.1 içindeki ikinci kod bölümü aynı bellek lokasyonundan R6 mimari yazmacına bir değer yükleyen iki tane yükle (LD-load) komutunu göstermektedir. Bu iki yükleme işlemi arada birkaç tane mikro-buyruk bulunan kısa bir zamanda gerçekleşmiştir. Yükle buyruklarının yürütüldüğü her iki zamanda da R6'nın yeniden adlandırıldığı fiziksel yazmaçlar aynı değerlere sahip



Şekil 5.4: 160 yazmaçlı bir tamsayı yazmaç öbeğinde 18 SPEC2006 gösterge programı için azami Hamming uzaklığı 0-8 arasında bir başka yazmaca sahip olan aktif yazmaçların ortalama oranı.

olmuşlardır. Çizelge 5.1 içindeki son kod bölümü *specrand* gösterge programından alınan mikro-kodlarda yığıt (stack) işlemlerini (push-pop kaynaklı işlemleri) gösterir. Yığıt göstergesi olarak kullanılan mimari yazmaç (RSP) güncellendikçe farklı fiziksel yazmaçlara yeniden adlandırılır, bu şekilde de yazmaç öbeğinde yığıt göstergesinin çok sayıda kopyaları oluşur.

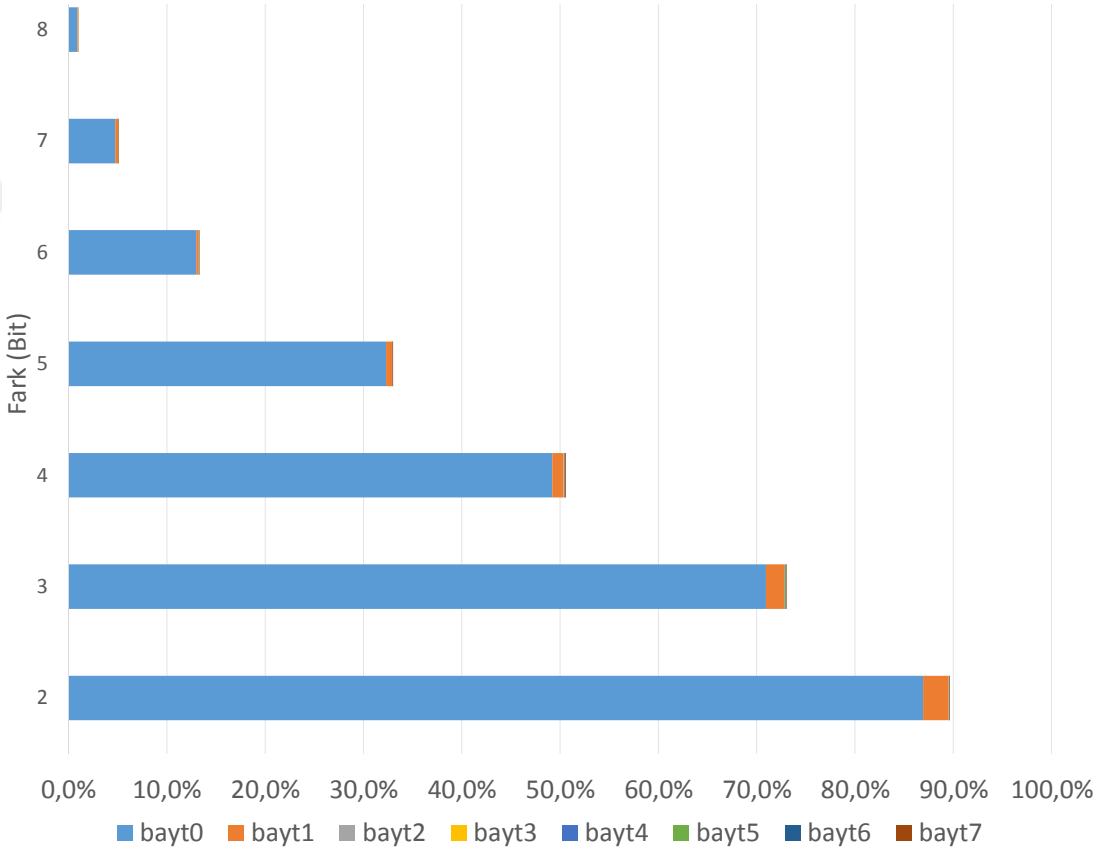
Kopya yazmaçların yanında sadece birkaç biti farklı olan yazmaçları da dahil edersek yazmaç öbeğindeki yazmaçların çok daha fazlası kapsanabilir. Şekil 5.4, 0-8 arasında Hamming uzaklığındaki benzer yazmaçların ortalama oranını gösterir. Tıpatıp aynı olan kopyaların ortalama oranı % 66,1 iken 1-bit Hamming uzaklığındaki yazmaçları dahil ettiğimizde oran ortalama % 80,6'ya çıkar. Benzerliği 8-bit Hamming uzaklığına çıkardığımızda ise benzer yazmaçların ortalama oranı %98,6'ya yükselir. *sjeng, gobmk* ve *libquantum* gibi gösterge programlar için ise bu değer % 99,5'in de üzerine çıkar. Bu bulgulardan yola çıktığımızda sadece aynı değerleri değil aynı zamanda benzer değerleri de kullanabilirsek, bu bize yazmaç öbeğini korumak için çok büyük bir fırsat sunacağı neticesine vardık. Bunun yanında birçok benzer yazmacın yalnızca en alt



Şekil 5.5: Her çubuk benzer yazmaçların farklı olan bit sayıları için (8 bite kadar) farklı bitlerin yazmaçtaki yerini gösterir.

baytında farklı olduğunu gözlemledik.

Şekil 5.5, ortalamada benzer yazmaçların farklı bitlerinin yazmacın neresinde olduğunu gösterir. Benzer yazmaçlar arasındaki fark 1-bit olduğunda fark en çok en alt baytta görülür (% 67,2), ondan sonra da en çok en üst baytta görülür (% 24,6). Fark 1-bitten fazla olduğunda ise farklı olan bitler en çok sadece en alt baytta görülür. Bu bulgudan istifade etmek için en alt bayt hariç bütün baytları hatalara karşı koruyan bir mekanizma tasarladık. Bu yöntemin kazancını bulmak için farklı bitlerinin tamamı en alt baytta olan benzer yazmaçları kontrol ettik. 1-bit fark elbette her zaman bir baytta olur ve baytlar arasında paylaştırılamaz. Ancak yazmaçlar arasında birden fazla bit fark varsa, bu bitlerin farklı baytlarda olması mümkündür. Diğer taraftan eğer, örneğin 2-bit fark olan benzer yazmaçların çoğunun farklı bitleri en alt baytta ise, yazmaçların benzerliğini kullanan bir mekanizma yazmacın diğer baytlarını koruyabilir. Şekil 5.6 farklı bitleri aynı baytta olan benzer yazmaçların ne kadar olduğunu göstermektedir. Hamming uzaklığı 2 olan benzer yazmaçların farklı bitlerinin % 91,8'i en alt baytta iken (Şekil 5.5), Şekil 5.6 ile gözlenebileceği gibi bu farklı bitlerin büyük çoğunluğu (% 86,8) aynı baytta bulunmaktadır. Farklı bitlerin sayısı arttıkça bütün farklı bitlerin aynı baytta olma yüzdesi azalır, ama yine de 5-bit Hamming uzaklığındaki yazmaçların bütün farklı bitlerinin en alt baytta olma oranı % 32,3 ve 7-bit uzaklıkta %4,75'tir.



Şekil 5.6: Her çubuk benzer yazmaçların birden fazla farklı bit sayıları için (8'e kadar) aynı baytta yer alan farklı bitlerin yerini ve benzer yazmaçlar arasındaki ortalama oranını gösterir.

Önerdiğimiz mekanizma için temel motivasyonumuz kopyaların ve benzer yazmaç değerlerinin yazmaç öbeğinde bulunmasıdır; yani yazmaç öbeği için geçici hataları düzeltme yöntemi uygulamanın masraflarını azaltmak için bu zaten mevcut olan yedekliliği kullanmaktayız.

5.2. Önerilen Tasarımlar

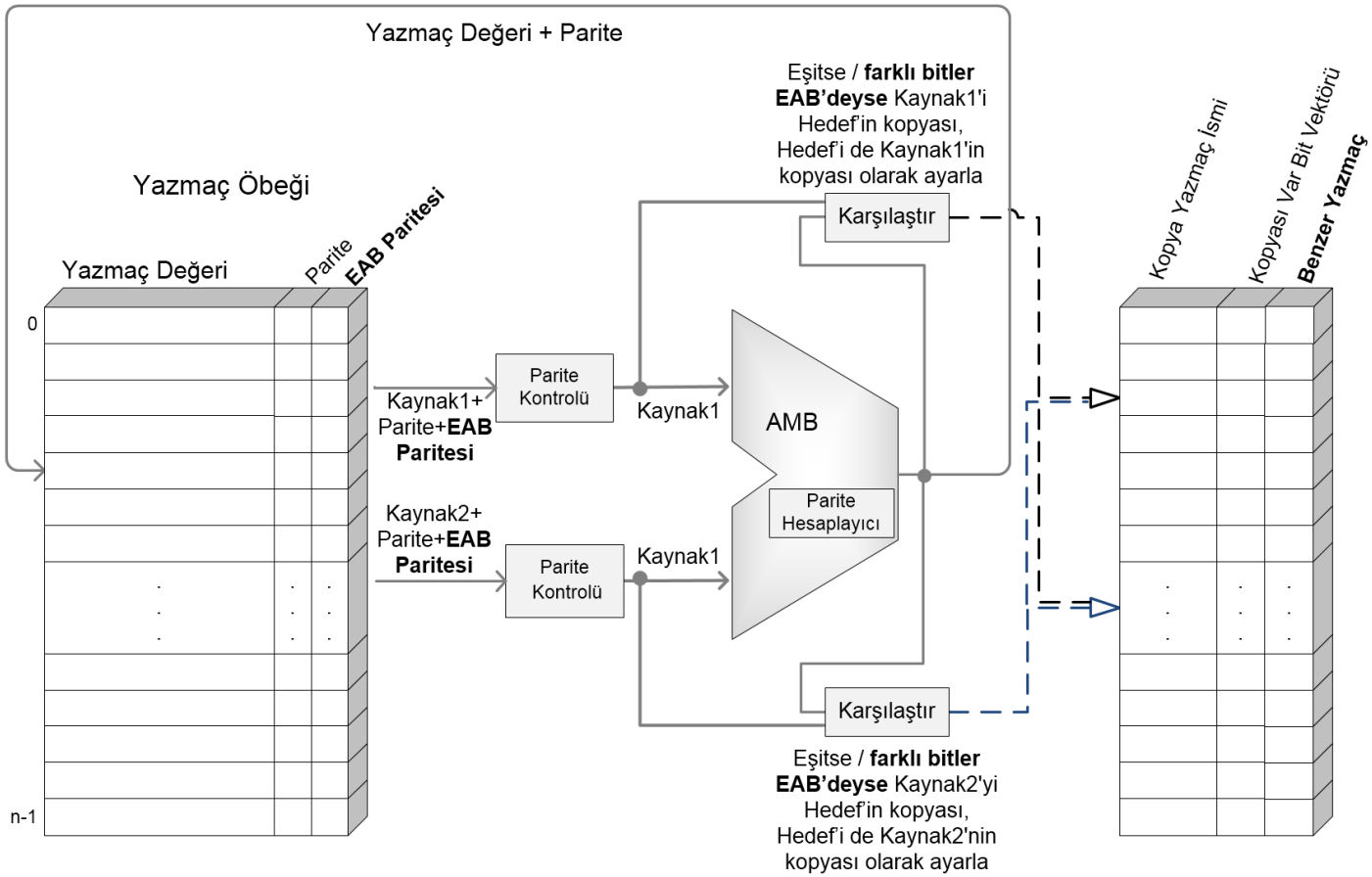
5.2.1 Kopyayla düzeltme (KOD)

Geçici hataları azaltmaya dönük çoğu güvenilirlik yöntemi bir çeşit çoklamaya veya yedekliliğe dayanır. Bu yöntemler mevcut alanı azaltarak veya kullanılmayan alanı kullanarak bu yedekliliği oluşturur. Biz ise bu bölümde, yazmaç öbeğinde zaten bulunan yedekliliği kullanmayı hedefliyoruz.

5.2.1.1 Tasarım detayları

Yazmaç öbeğinde tıpatıp aynı değerleri algılamak, aynı değerler herhangi bir yazmaçta olabileceğinden ve bir yazmacı güncelleyen her buyruğun diğer bütün yazmaçların değerlerini kontrol etmesi gerekeceğinden kolayca başarılabilecek bir durum değildir. Bu *kopya değer algılama* işlemini uygulanabilir yapabilmek için sonuç değeri en az bir kaynak yazmaç değerine eşit olan buyrukları kullanmayı öneriyoruz.

Önerdiğimiz çözümdeki bir tasarım sorunu aynı değerdeki yazmaçların nasıl algılanacağıdır. Birçok kopya yazmaç değeri kaynak yazmaçlarına değişiklik yapmayan bazı buyruklar tarafından oluşturulduğundan (örneğin bir değere sıfır eklemek), yazmaç kopyalarını boru-hattının yürüt aşamasında algılamak mümkündür. Kaynak yazmaçla sonuç arasında fark olmadığında bir kopya değer oluşur ve kopya değerin yeri kaydedilirse bu kopyayı daha sonra kullanmamız mümkün olur. Kopyaları iptal etme durumunda tutarlılıkla ilgili sorunlar oluşmaması için her yazmacın en fazla bir yazmacın kopyası olması gerekmektedir. Kopya değerleri algılamak amacıyla her iki kaynağın (iki kaynaklı buyruklar için) hedef yazmaçla eşitliğini tespit etmek için karşılaştırma devresi eklemeyi öneriyoruz. Eğer buyruğun yalnızca bir kaynak yazmacı varsa, karşılaştırma işlemlerinden sadece biri yapılır. Karşılaştırılan değerlerin aynı olduğu bulunursa ve yazmaçların isimleri aynı değilse (sırasız işlemcilerde yazmaç yeniden adlandırması nedeniyle genellikle böyledir), her kopya diğer yazmacın ismini Şekil 5.7 ile gösterildiği gibi "**Kopya Yazmaç İsmi (KYİ)**" adlı eklenmiş alana kaydeder.



Şekil 5.7: Geçici hatalara karşı koruma için yazmaç öbeği mimarisi. YADED ilavesi dolayısıyla eklenen alanlar veya değişiklikler koyu biçimde gösterilmiştir.

Eğer eşit-değerli kaynak yazmaç zaten bir kopya olarak kullanılmaktaydı ise bu kez hedef yazmacın kopyası olarak tahsis edilemez. Ayrıca eğer bir yazmaç aynı anda yürütülmekte olan birçok buyruğun kaynak yazmacı olarak kullanılıyorsa buyruklardan sadece birine kaynak yazmacı kopya olarak tayin etme izni verilir.

Kopyası olan bir değerde hata olması durumunda arama işlemini hızlandırmak amacıyla "**Kopyası Var Bit Vektörü (KVBV)**" isimli her yazmaç için bir bit bulunduran bir bit-vektörü öneriyoruz. KVBV sayesinde hata olduğunda buraya bakarak ve kopya yazmaca gitmeye gerek kalmadan kopyanın geçerli olup olmadığı anlaşılabilir. İki yazmacın birbirinin kopyası olduğu bulunduğunda, KVBV'nin ilgili bitleri "1" yapılarak güncellenir. Daha önceden kopya değer tutan bir yazmaç değeri değiştiğinde (üzerine yeni bir değer yazıldığında), bu yazmaca ve kopyasına denk gelen bitler KVBV'de "0" yapılır. Bu sayede daha sonra gelen yazma işlemi nedeniyle geçerliliği kalkan kopyaları saptamak mümkün olur ve geçici bir hatayı düzeltmek için artık kullanılmaz. KVBV'yi dizinlemek ve kopyanın ilgili bitini silmek için hedef yazmaca yazılmadan önce hedef yazmacın KYİ'sinin okunması gerekmektedir (Şekil 5.7 içinde gösterilmemiştir).

Önerdiğimiz mekanizma nedeniyle eklenen KYİ ve KVBV alanları ayrı bir küçük bellekte tutulur çünkü bu alanlara erişim yazmaç öbeğine erişimden bağımsızdır ve sadece bir kopya ya da hata yakalanınca gerçekleşir. Bu yüzden asıl veri kaydı alanına daha fazla port eklenmemekle birlikte eklenen alanın yazma portu sayısı kayıt biriminin yazma portu sayısının iki katıdır. Eklenen alanların okuma portlarının sayısı yazmaç öbeğininkiyle aynıdır. Bu suretle veri kayıt bölümündeki portların toplam sayısı değişmemiş ve eklenen alandaki portların sayısı ise kayıt bölümündekilere göre bir miktar fazla olmuştur. Bu alanlara erişim gecikme masrafını artırmaz çünkü ayrı olan bu bellek yazmaç öbeğinden çok daha küçüktür ve bu alanlara yapılacak güncellemeler yazmaç öbeğine geri yazma işlemiyle paraleldir.

Literatürde de ifade edildiği gibi tek-bitlik hatalar en yaygın olan hatalardır [8]. Dolayısıyla parite kontrolü hataların çoğunu yakalayacaktır ve biz de bu nedenle her fiziksel yazmaç için yazmaç öbeğine bir parite biti alanı ekledik. Yazmaç okuma sırasında parite kontrolü yapılır ve yazmaca yazma sırasında parite alanına konulmak üzere AMB aşamasında ilgili yazmacın paritesi hesaplanır. KOD sayesinde parite bitlerini kullanarak hata yakalamanın kapsamını hata düzeltmeye genişletmek mümkündür. Nitekim artık değerin bir kopyasına sahibiz ve bu kopyanın geçerli olup olmadığını da biliyoruz. Parite biti kullanarak hata yakalamak, kopya yazmaç bulunmasa dahi yararlıdır çünkü uygulamayı hatadan haberdar ederek uygulama seviyesinde bir çözüm bulunabilir.

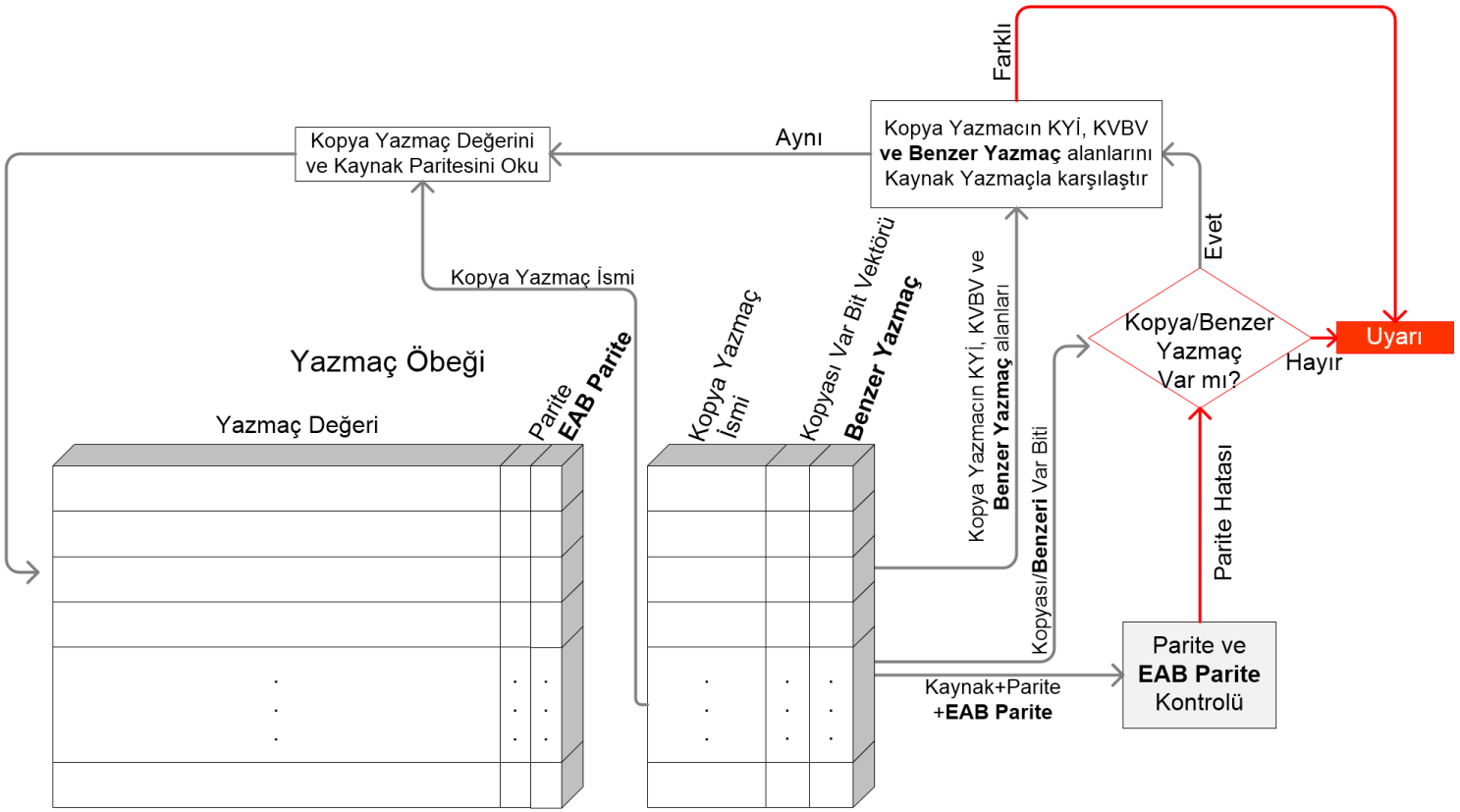
KOD'u tasvir etmek amacıyla `R4` değerini sıfır olan `R5` değerine ekleyen ve sonucu

R6'ya yazan bir buyruk Şekil 5.7 üzerinde açıklanacaktır. Buyruk belleğinden getirildikten sonra buyruk, kaynak ve hedef yazmaçlarının isimlerini içerir. R4 ve R5'in etiketleri yazmaç öbeğine yazmaçları dizinlemek üzere sağlanır. Şekil 5.7 ile de gösterildiği gibi kaynak yazmaç değerleri okunur ve parite kontrol devresine nakledilir. Eğer parite kontrol sonuçları doğruysa (yani R4 ve R5'te tek sayılı hata yoksa) değerleri AMB'ye götürülür. Sonuç değeri AMB'de hesaplanır ve paritesi tasarıma eklenen parite hesaplayıcı yapısında hesaplanır. Sonuç ve kaynak değerleri karşılaştırılır. Bu buyruk için sonuç değeri R4 değeriyle aynı olduğundan ve KVBV'de R4'e tekabül eden değer "0" olduğundan, R6'nın KYİ alanına R4 ve R4'ün KYİ alanına R6 yazılır. Ayrıca KVBV'deki ilgili bitler de "1" yapılır. Bu yolla her iki yazmaç da birbirine işaret edecek ve böylece kopyalarının yerini hata olması durumunda bulabileceklerdir.

KOD yönteminin parite kontrol ve düzeltme akış diyagramı Şekil 5.8 ile gösterilmiştir. Parite kontrolü kaynak yazmaç okunduğunda yapılır. KOD yönteminde kaynak yazmacın paritesi yazmaç değeriyle birlikte okunur. Eğer parite kontrolü yazmaç değerinde bir hata olduğunu gösterirse, bu durumda kopya yazmacın bulunup bulunmadığı KVBV içinde ilgili biti bularak anlaşılır. Eğer hata bulunan kaynak yazmacın yazmaç öbeğinde bir kopyası yoksa, o halde uygulamaya bir uyarı verilir ve bilgilendirilir. Aksi takdirde, kopya yazmaç, paritesi ve KYİ alanları hatayı düzeltmek için okunur. Kopya yazmacın bir hataya uğramış olmasına karşı önlem olarak kopya yazmacın paritesi de kontrol edilir. Kopya yazmacın KYİ alanını okuma işlemini de yöntemimize eklememizin sebebi, ekli alanlardan birinde (KYİ, KVBV) hata olması durumunda kopyanın KYİ alanını kaynak yazmacın ismiyle karşılaştırarak bu hatayı yakalayabilme imkanımızın olmasıdır. Ekli alanlardan birinde oluşan bir hatadan dolayı eğer karşılaştırma sonucu farklı çıkarsa, bu hata da böylece farkedilmiş olur.

5.2.1.2 Yazmaç ikileme mekanizması

KOD'un yazmaç öbeğindeki kapsamını genişletmek amacıyla, boşta olan yazmaçları aktif yazmaç değerlerinin yedek kopyalarını oluşturmak için kullanmak şeklinde bir ilave geliştirme öneriyoruz. Bu yöntem [8] çalışmasında önerildiği şekliyle uygulanabilir. Kullanılmayan yazmaçları, buyrukların sonucunu hedef yazmaçla birlikte yazmak ve bu şekilde ikinci bir kopya oluşturmak için kullanıyoruz. Bunu başarmak için işlemciye yeniden yazma aşamasında iki adet boş fiziksel yazmacı tahsis eden bir mekanizma ekledik. Bu iki yazmaçtan biri, normalde de olduğu gibi hedef yazmacı olarak, diğeri ise kopya yazmaç olarak kullanılır. Bu kopya yazmacın durumunu "suret" şeklinde adlandırdık.

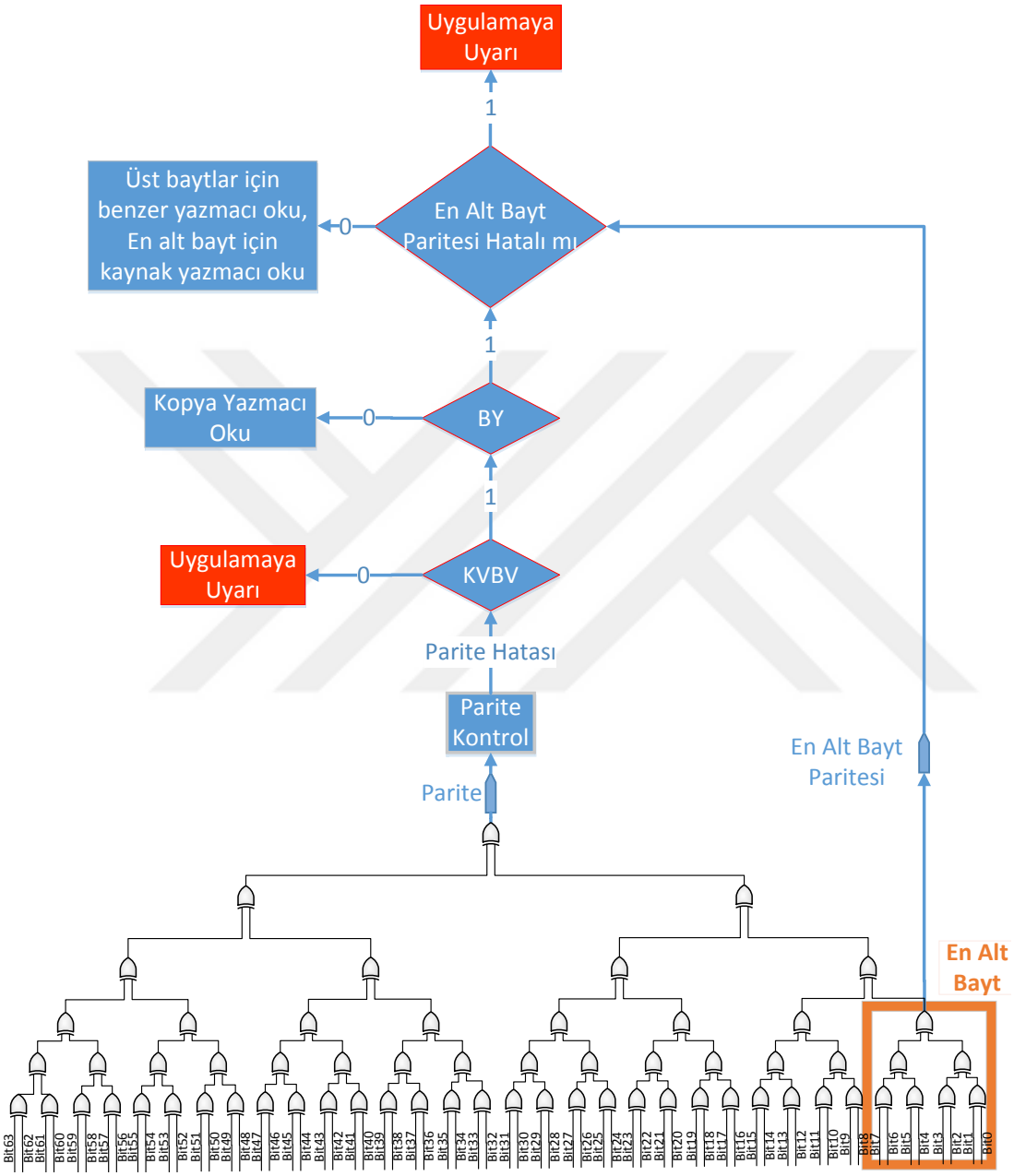


Şekil 5.8: KOD için yazmaç öbeği mimarisi üzerinde parite kontrol ve düzeltme akış diyagramı. YADED ilavesi dolayısıyla eklenen alanlar veya değişiklikler koyu biçimde gösterilmiştir.

"Suret" durumunda olan yazmaçlar yeniden yazma aşamasında , hatanın olmadığı yaygın durumda herhangi bir başarım azalmasını engellemek için, "boş" durumunda gibi ele alınır. Ancak bir başka "suret" yazmacı olarak tahsis edilmek istendiğinde "dolu (geçerli)" durumundaymış gibi davranır. Hedef yazmaçla birlikte bir de suret yazmaç tahsis edildiğinde, buyruğun sonucu her iki yazmaca da yazılır ve her iki yazmaç da diğer yazmacın adını KYİ alanına yazar. KVBV'nin de ilgili bitleri "1" yapılır. Bu noktadan sonra tasarım, parite kontrolü ve düzeltme açısından aynı şekilde çalışır. Bu ilave geliştirmeyi "Yazmaç İkileme Mekanizması" veya YİM olarak adlandırıyoruz.

5.2.2 Yakın değerlerle düzeltme (YADED)

KOD yönteminin kapsamı, sadece tıpatıp aynı olan kopyalar değil aynı zamanda benzer yazmaçları da yönteme dahil ederek genişletilebilir. Bu yolla farklı bitlerinin tümü en alt baytında (EAB) bulunan benzer yazmaçlar EAB hariç olmak üzere geçici hatalara karşı korunabilir. Bu ilaveyi "**Yakın Değerlerle Düzeltme (YADED)**" olarak adlandırıyoruz. YADED tasarımı uygulama seviyesine gitmeye gerek kalmadan yakalanan birçok hatanın düzeltmesiyle sonuçlanır. Eklenen alanlar ve mantıksal devreler 5.7 ve 5.8 şekillerinde koyu biçimde gösterilmiştir. YADED'in düzeltme akış diyagramı Şekil 5.9 ile gösterilmiştir. YADED mimarisindeki tek fark "Benzer Yazmaç (BY)" bit vektörü ve EAB parite alanıdır. BY alanı "1" olduğunda KYİ alanının, tıpatıp aynı bir kopyayı değil sadece EAB'si farklı olan benzer bir yazmacı gösterdiğini belirtir. EAB paritesi alanı yazmacın en alt baytının paritesidir. YADED yapısında KYİ ile KVBV alanlarını değiştiren devre KOD yapısına göre değişmiştir; karşılaştırma sonucunda farkın sadece en alt baytta olduğu görülürse KYİ, KVBV ve BY alanları kopya yerine benzer yazmacı gösterecek şekilde güncellenir. Eğer karşılaştırma sonucunda yazmaçların tıpatıp aynı olduğu görülürse, bu durumda BY alanı "0" yapılır. EAB için parite hesaplaması kısmi parite hesaplama konusunda yapılmış çalışmalardaki gibi, EAB'nin paritesi yazmacın paritesiyle birlikte eş-zamanlı olarak gerçekleştirilir [76] [77] (5.9 şeklinin alt kısmındaki devrede görüldüğü gibi). YADED yönteminde parite hatası yakalanırsa KVBV alanı kontrol edilir. Eğer KVBV alanı "0" ise parite hatası düzeltilemez ve uygulamaya uyarı verilir (exception), ancak KVBV alanı "1" ise yazmaç öbeğinde KYİ ile gösterilen bir kopya veya benzer yazmaç bulunmaktadır. Ardından Benzer Yazmaç (BY) alanı kontrol edilir; eğer "0" ise KYİ alanı kopya yazmacı göstermektedir, "1" ise sadece EAB'si farklı olan benzer bir yazmacı göstermektedir. İkinci durum için, bu kez aynı parite kodlama/çözme devresinden elde edilen EAB paritesi geçici hatanın en alt baytta olmadığından emin olmak için kontrol edilir. Çünkü benzer yazmaçlar için YADED



Şekil 5.9: Yazmacın tamamının ve en alt baytının (EAB) paritesinin kodlama/çözme devresi ile parite kontrol ve düzeltme akış diyagramı.

yöntemi hata ancak en alt baytta değilse yazmaç öbeğini koruyabilmektedir. Benzer yazmaçtan yazmaç değerindeki hatayı düzeltebilirsek bu durumda üst baytlar KYİ ile gösterilen benzer yazmaçtan okunur ve EAB da orijinal kaynak yazmaçtan alınır. Ancak EAB paritesi kontrolü burada hata olduğunu gösteriyorsa ve yazmacın sadece benzer yazmacı varsa (kopya yazmacı yoksa), bu durumda Şekil 5.9 ile gösterildiği gibi uygulamaya uyarı verilir.

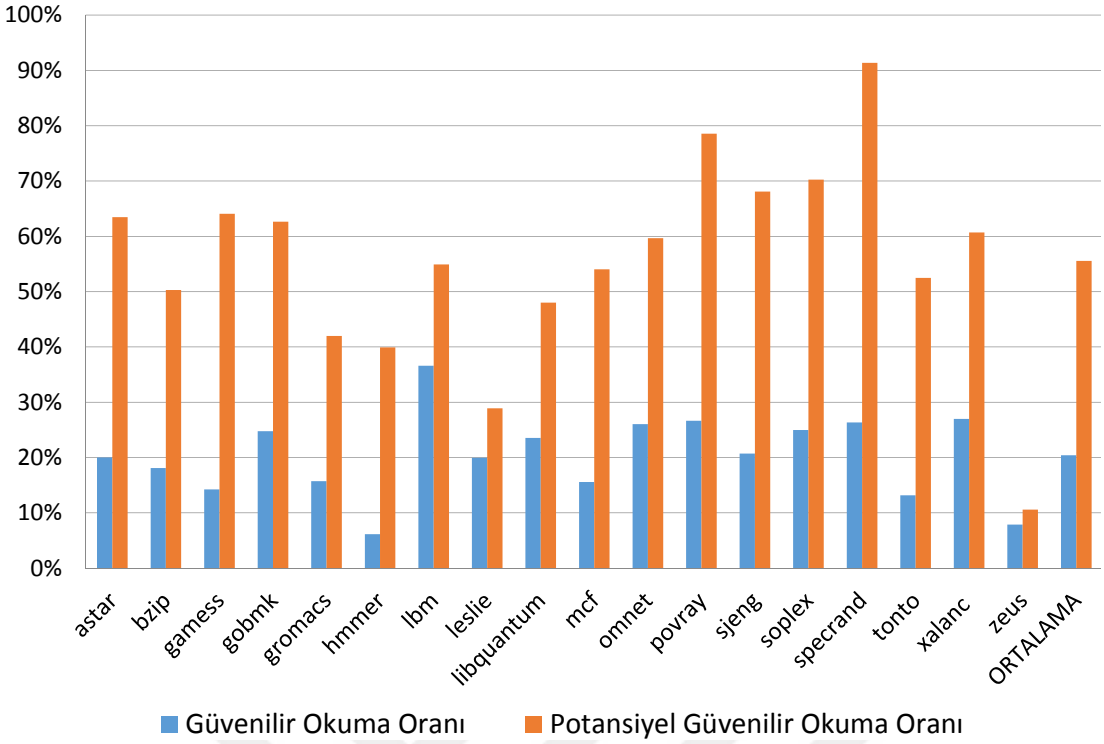
5.3. Benzetim Ortamı

Önerilen tasarımın etkisini karakterize etmek amacıyla mimari benzetimler için x86 işlemcilere yönelik MARSSx86 tüm-sistem mikro-mimari benzeticisinin değiştirdiğimiz bir versiyonunu kullandık [67]. x86 buyruk kümesi mimarisini kullanan MARSSx86 boru hattında yürütmek için İndirgenmiş Buyruk İşlemcileri'nin (Reduced Instruction Set Computer) buyrukları şeklinde olan mikro-buyrukları oluşturur. MARSSx86 benzeticisi, değişik mimarileri taklit eden ve orijinal işletim sistemlerini çalıştırabilen bir sanal makine olan QEMU üzerine bina edilmiştir [68]. Deneyler için referans girdi kümesiyle x86-64 mimarisine göre O3 optimizasyon seviyesinde derlenmiş SPEC CPU2006 gösterge programlarından 18 tanesini kullandık [69]. MARSSx86 benzeticisini, derlenen gösterge programları üzerinde Linux Ubuntu çalışan, 64-bit, sırasız ve 160 yazmaçlı bir tamsayı yazmaç öbeğine sahip Intel işlemci modelinde çalıştırmak için kullandık. Yazmaç öbeği boyutunu kritik görevli sunucular için tasarlanmış Intel İtanium Poulson işlemcisiyle benzer olacak şekilde seçtik [51]. Deneylerimizde her gösterge programı 1 milyar buyruk ileri attıktan sonra 100 milyon kullanıcı buyruğu emekli edilinceye kadar çalıştırdık. Sonuçları elde etmek için mikro-buyrukları ve mimari ile tamsayı fiziksel yazmaç öbeklerini kullandık.

Devre seviyesinde yaptığımız benzetimler tasarlanan yazmaç öbeği modelinde gerçekleştirildi. Bunun için Cadence Analog Design Environment programını UMC 90nm teknolojisiyle kullandık. Yazmaç öbeği benzetimleri ve ortalama güç tüketimi ölçümleri $1V$ V_{DD} geriliminde $27\,^{\circ}C$ çevresel sıcaklıkta gerçekleştirilmiştir.

5.4. Sonuçlar

Önerilen tasarımın hata düzeltme bağlamında ne denli başarılı olduğunu göstermek için iki metrik kullanabiliriz: ortalama kapsama ve güvenilir okuma oranı. Ortalama kapsama, kopyası (veya benzeri) olan yazmaçların bütün yazmaçlara oranıdır. Güvenilir okuma oranı ise, [8] çalışmasında tanımlandığı gibi, okunan ve kopyası

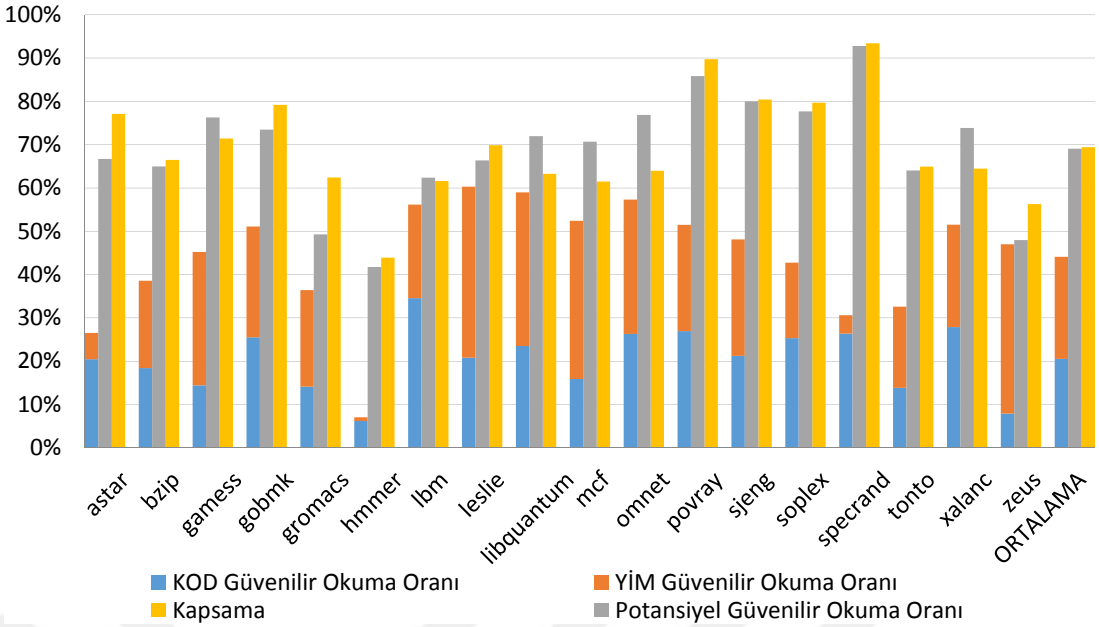


Şekil 5.10: KOD yöntemi kullanıldığında elde edilen güvenilir okuma oranı ile potansiyel güvenilir okuma oranı.

olan yazmaçların okunan yazmaçlara oranını ifade eder. Ortalama kapsamayı, kullandığımız bütün gösterge programlar ve bütün kopyalar için ölçtük. Kapsama istatistikleri için boşta olan yazmaçları da hesaba kattık zira eski bir değere sahip boş yazmaçların bazısı aktif yazmaçların kopyalarına sahip olabilmektedir ve tutarlılığı koruduktan sonra bunları kullanmanın herhangi bir sakıncası yoktur. Güvenilir okuma oranlarını bütün geçerli fiziksel yazmaçlar için ve kullandığımız bütün gösterge programlar için önerdiğimiz tasarım üzerinde ölçtük. Bunların yanında önerilen tasarımla yakalanamayan fakat başka şekillerde yakalanma ihtimali olan potansiyel kopya yazmaçlarını da değerlendirdik. Yöntemimizin etkinliğini değerlendirmek için yazmaçlara hata atma deneyleri de gerçekleştirdik.

Potansiyel kapsama istatistikleri Şekil 5.2 ile gösterilmişti. Bizim temel motivasyonumuz olan bu sonuç bütün aktif (dolu) yazmaçların %66,1'inin yazmaç öbeğinde kopyasının olduğunu gösteriyor. Bizim mekanizmamız yürütme sonucuyla kaynak yazmaç değerlerini karşılaştırmak suretiyle bu kopyaların ancak %31,4'ünü yakalayabilir (Şekil 5.3 ile gösterildiği gibi). Bu nedenle de algılanan ortalama kapsama % 20,8 olmaktadır.

KOD yöntemi için algılanan kopyaların güvenilir okuma oranları Şekil 5.10 içinde soldaki çubuklarla gösterilmiştir. Şeklin sağındaki sütunlar ise kopyaların tamamını algılayabilen bir mekanizma olsaydı güvenilir okuma oranının ne olacağını (yani

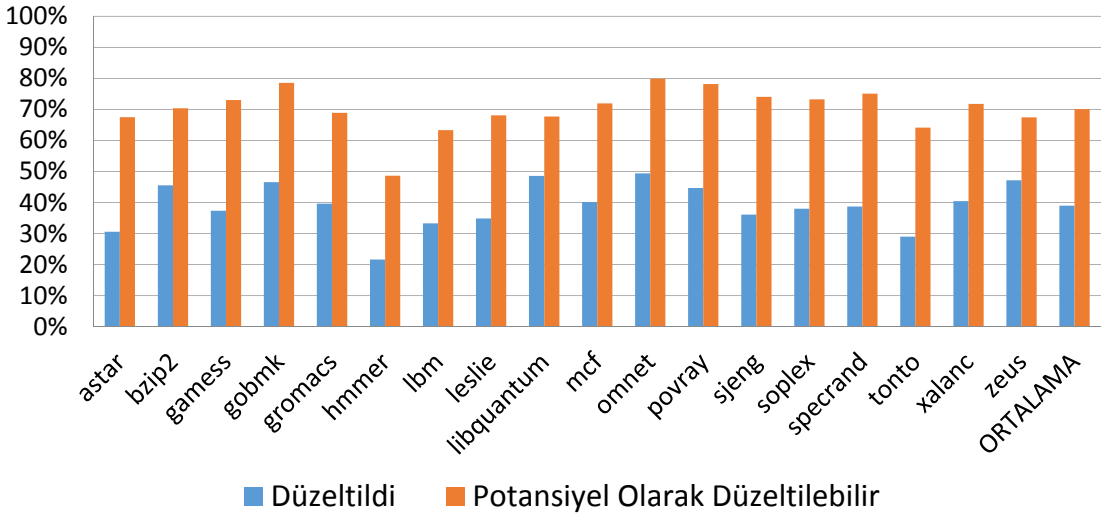


Şekil 5.11: KOD ile YİM sonuçları. Soldaki üst üste çubuklar algılanabilen kopyaların güvenilir okuma oranlarını gösterir; alttaki çubuk parçası KOD yöntemiyle algılanan kopyaları ve üstteki parça YİM yöntemiyle edinilen ve algılanan kopyaları ifade eder. Ortadaki sütun yazmaç öbeğindeki tüm kopyaları algılamak mümkün olsaydı elde edilecek potansiyel güvenilir okuma oranını gösterir. En sağdaki sütun ise bütün yazmaçların ortalama ne kadarının kopyalı olduğunu gösterir.

potansiyel güvenilir okuma oranını) gösterir. Algılanan kopyaların güvenilir okuma oranı ortalamada % 20,4'tür ve bazı gösterge programlar için % 36'ya kadar çıkabilmektedir. Potansiyel güvenilir okuma oranı da ortalama % 55,6'dır ve bazı gösterge programlar için % 70'e kadar çıkabilmektedir.

KOD ve YİM yöntemlerinin birleşiminden elde edilen sonuçlar Şekil 5.11 ile gösterilmiştir. YİM geliştirmesiyle birlikte ortalama kapsama % 69,4'e yükselir. Potansiyel güvenilir okuma oranı da önemli bir artış göstererek % 69,1'e ulaşır (5.11 şeklinde ortadaki sütun). Şekil 5.11 içindeki sütun gruplarının en solundaki çubuklar algılanabilen kopyalar nedeniyle oluşan güvenilir okuma oranını gösterir. Önerdiğimiz yapı aynı zamanda YİM tarafından oluşturulan kopya değerlerini de algıladığından, her gösterge programı için en soldaki üst üste olan çubuklardan alttaki KOD sayesinde elde edilen güvenilir okuma oranı, üstteki ise YİM tarafından oluşturulan kopyalar sayesinde elde edilen oranı gösterir. Tüm algılanabilen kopyalar sebebiyle oluşan güvenilir okuma oranı ortalamada % 44,1'dir.

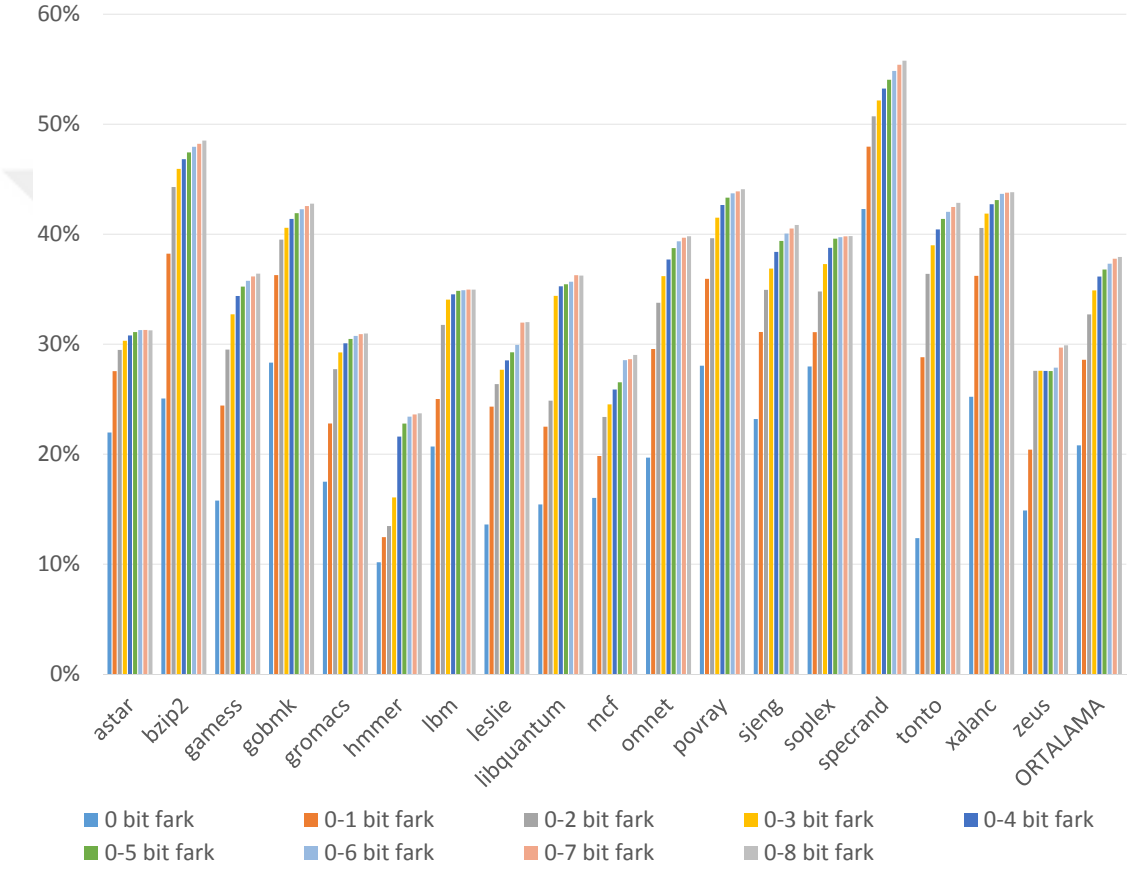
Bu sonuçlar önerilen mimarinin hata düzeltme başarımı hakkında fikir edinmemizi sağlasa da, geçici hataları düzeltmede önerilen mimarinin ne kadar başarılı olduğunu görmek için yazmaçlara hata atma deneyleri de yaptık. Gösterge programları 1 milyar



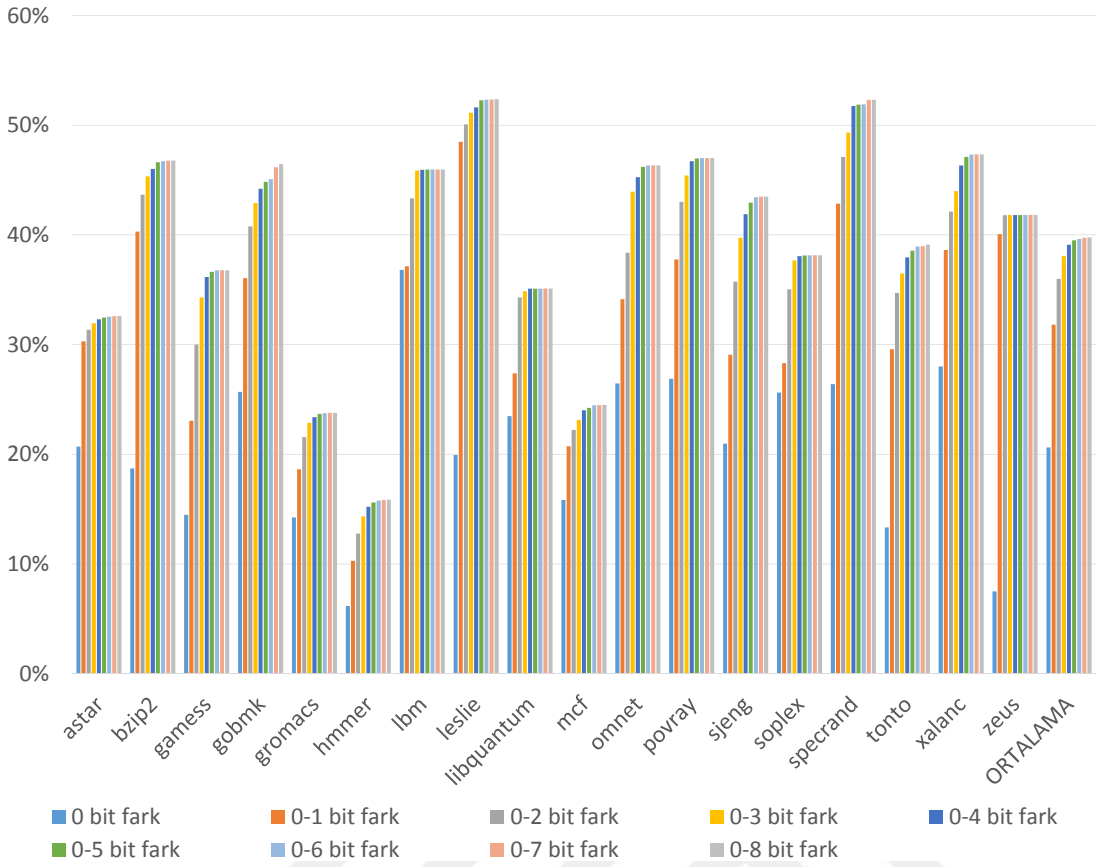
Şekil 5.12: Rastgele hata atma sonucunda bulunan hata düzeltme sonuçları. Soldaki sütun önerilen tasarımla düzeltilebilen hataların yüzdesini gösterir. Sağdaki sütun eğer tüm kopyalar algılanabilseydi o durumda düzeltilebilecek hataların yüzdesini gösterir.

buyruk ileri attıktan sonra yazmaç öbeğindeki rastgele bir yere rastgele bir zamanda hata attık. Hatanın bizim yöntemimizce düzeltilip düzeltilmediğini görmek için hatayı 10 milyon buyruk boyunca takip ettik. Toplamda gösterge program başına 200 hata attık ve benzetimi her yürütmede bir hata atma gerçekleştirdik. Uyguladığımız bu hata atma metodolojisi geçmiş çalışmalardaki hata atma yöntemlerine benzerdir [78] [79]. Şekil 5.12, 18 SPEC2006 gösterge programı için hata düzeltme oranlarını ve tüm gösterge programlarının ortalamasını gösterir. Şekil 5.12 içindeki sütun gruplarından soldaki sütun önerilen mimari ile elde edilen hata düzeltme oranlarını, sağdaki sütun ise yazmaç öbeğindeki bütün kopya yazmaçlar algılanabilseydi elde edilecek olan hata düzeltme oranını göstermektedir. Önerilen tasarımın hata düzeltme oranı ortalama % 39,0 ve potansiyel hata düzeltme oranı % 70,1 olarak gözlenmiştir. 40nm teknoloji için işlemci başına Tek Seferlik Bozulma (SEU) oranının 0,94 FIT/Kb olduğu raporlanmıştır [80]. Dolayısıyla, değerlendirdiğimiz 160x64 boyutundaki yazmaç öbeğinin 12,3 yıllık MTTF'si vardır. Önerdiğimiz metot kullanıldığında ise yazmaç öbeğinin MTTF'si 20,2 yıla çıkar.

Son olarak önerdiğimiz mimarinin güç tüketimiyle karşılaştırmak için geleneksel hatalara karşı korumasız yazmaç öbeğinin periferik devreleriyle birlikte güç tüketimini de bulduk. Önerilen mimarinin fazladan harcadığı güç hedef yazmacın KVBV ve KYİ alanlarını okumak, kopyasının KVBV alanına yazmak, parite hesaplama ve kontrolü ile karşılaştırma devresinin güçlerinden oluşur. KOD yapısının fazladan güç tüketimi YİM dahil olmadığında ortalama sadece % 2,8'dir. YİM'i dahil ettiğimizde ise ekstra güç tüketimi % 18,9 olur.



Şekil 5.13: 18 SPEC gösterge programı için ve 8-bit farka kadar benzer yazmaçların algılanabilen kapsama oranları ve bunların ortalaması.



Şekil 5.14: 18 SPEC gösterge programı için ve 8-bit farka kadar benzer yazmaçların güvenilir okuma oranları ve bunların ortalaması.

Bölüm 5.2.2 altında anlatıldığı şekilde YADED yöntemini kullanarak YİM'den kaynaklı eklenen güç tüketimini yaklaşık aynı miktarda güvenilirlik sağlanarak engelleyebiliriz. YADED kopya yazmaçlarının yanında sadece en alt baytı farklı olan benzer yazmaçları da geçici hatalara karşı korur. Bu mekanizmanın KOD yönteminde olmayan tek masrafı BY ve EAB paritesi alanlarıdır. En alt baytın paritesinin hesaplanması yazmacın tek-bit paritesini hesaplayan aynı devreyle tamamlandığından, YADED herhangi bir gecikme masrafı eklemez ve önemsiz bir güç masrafı ekler.

Benzer yazmaçları yakalamak için önerilen yöntem kopya yazmaçları bulmak için kullanılan yöntemle aynıdır; yürüt aşamasında kaynak ve hedef yazmaç değerlerini karşılaştırmak. Algılanabilen kapsama sonuçları Şekil 5.13 ile gösterilmiştir. Şekilde her gösterge program için YADED sayesinde algılanabilen kapsama oranını ifade eden 9 adet çubuk vardır; her grubun en soldaki çubuğu benzer yazmaçların en alt bitleri arasındaki farkın 0-bit olduğu durumu (kopya yazmaç), sola doğru her çubuk için fark artarak en sağdaki çubuk ise farkın 8-bit olduğu durumu gösterir. Ortalama algılanabilen kapsama kopya yazmaçlardaki (Hamming uzaklığı 0) % 20,8 olan durumdan, 8-bit azami Hamming uzaklığı dahil edildiğinde % 37,5'e çıkmaktadır.

YADED yönteminin güvenilirlik başarımı benzer yazmaçların güvenilir okuma oranlarını gösteren Şekil 5.14 ile görülebilir. Bütün gösterge programlar için genel eğilim, Hamming uzaklığı 0'dan 1 ve 2'ye çıktığında güvenilir okuma oranında hızlı bir yükselme ve uzaklıklarda daha fazla artış oldukça daha düşük yükselişler şeklindedir. Güvenilir okuma oranı kopya yazmaçlar için % 20,5'ten, Hamming uzaklığı 8 olduğunda % 39,8'e çıkmaktadır. YADED yönteminin güvenilir okuma oranı KOD mekanizmasının oranından oldukça iyi olmakla birlikte, KOD+YİM metotlarının birleşiminden çok az miktarda azdır. Bununla beraber YİM metodu KOD yöntemine göre yaklaşık 6 kat daha fazla güç harcadığından, YADED sayesinde YİM metodunu kullanmamakla elde edilen güç tasarrufu önemli orandadır. Diğer taraftan güç kaybını göze alabilecek bir durum söz konusuysa YİM'i YADED yöntemiyle birlikte de kullanabiliriz. Eğer YİM ve YADED birleşimi uygulanırsa, toplam güvenilir okuma oranı % 63,3 seviyesine ulaşır.

5.5. Sonuç

Bu tezin bu kısmında yazmaç öbeğini geçici hatalara karşı daha dayanıklı kılmak için yazmaç öbeğinde zaten mevcut olan kopyaları kullanan Kopyayla Düzeltme (KOD) tasarımını sunduk. Aynı zamanda en alt baytlarında farklılık olan benzer yazmaçları kullanarak geçici hatalara karşı güvenilirliği sağlayabildiğimiz Yakın Değerlerle Düzeltme (YADED) tasarımını da yaptık. Bu tasarımlarda sonuç üreten her buyruk için kopyaları ve benzer yazmaçları algılamak amacıyla bir karşılaştırma işlemi gerçekleştirilir. Ayrıca her yazmaç okumasında hatayı yakalamak için tek-sayılı parite kontrolü yapılır. Düzeltilebilecek hataları benzer yazmaç veya kopya yazmaçtan hangisi sayesinde düzeltebileceğimizi ayırt etmek için her yazmaç için en alt bayt (EAB) paritesi alanı eklenmiştir. Buyruğun yazmaç öbeğinde benzer mi yoksa kopya değer mi oluşturduğunu algılamak için yazmaç değerleri ve sonuçlar birbiriyle karşılaştırılır. Hata algılama için asıl yazmaç değeri yanında parite koruması kullanılmakta ve hata düzeltmek için kopyaya (veya benzer değere) bir işaretçi tutulmaktadır.

Hata düzeltme oranımızı artırmak için geçmiş bir çalışmada ([8]) açıklanan Yazmaç İkileme Mekanizmasının (YİM) önerilen tasarımla bir arada kullanılabileceğini de gösterdik. YİM ile birlikte kullanıldığında KOD yöntemimiz hataların % 39,0 kadarını düzeltebilir ve güvenilir okuma oranı olarak da % 44,1 güvenilir okuma oranı yakalar. KOD tasarımının güç harcama masrafı tek başına % 2,8 iken, ortalama kapsamayı artırmak için YİM'i kullandığımızda ise bu masraf %18,9'a çıkar. KOD yöntemiyle birlikte YİM kullanmak yerine YADED tasarımını kullanırsak, oldukça yakın bir güvenilirlik başarımı (% 39,8) elde etmekle beraber güç tüketimi KOD'un

güç tüketimi seviyesinde olur. Ayrıca, YADED yöntemini KOD+YİM yöntemine eklersek, toplam güvenilir okuma oranı ortalama % 63,6 seviyesine ulaşır.

Önerdiğimiz çözüm, kodlama zorluğu tek-bit parite seviyesinde, alan masrafı ise ECC seviyesinde olan bir hata düzeltme tasarımı sunar. ECC ile karşılaştırıldığında, bizim tasarımımız daha az donanım masrafıyla çok-bitli hata düzeltme mekanizması sunmakta ise de biz bu korumayı sadece kopyası veya (sadece en alt baytı farklı) benzeri olan yazmaç değerleri için sağlayabilmekteyiz.

Bizim önerdiğimiz metot geçici hata oranını azaltmaya dönük daha alt-seviye yaklaşımlardan (örneğin mantıksal devrelerin radyasyon dayanıklılığını artırmak için kapı boyutu değiştirme [81] veya en çok geçici hata etkisine sahip kapıları seçip onların dayanıklılığını alan masrafını düşük tutarak artıran istatistiki optimizasyon yöntemleri [82]) bağımsızdır ve bunlarla birlikte kullanılabilir.

6. SONUÇ VE ÖNERİLER

Bu tezde işlemci yazmaç öbeklerinde bulunan verimsizliklerden faydalanmak, bu yolla güç tüketimini azaltmak ve güvenilirliğini artırmak için mimari ve devresel yöntemler sunduk.

Yazmaçlara yazılan değerlerin, artırma-azaltma, küçük değerlerle aritmetik işlemler vb. yapan buyruklar sebebiyle genellikle az miktarda biti değiştirdiğini değerlendirdik ve bunu deneylerle doğruladık. Yazmaçtaki değişimin birkaç bitten ibaret olması, fakat buna rağmen SRAM yazma mekanizmasının değişmeyen bitlerde de aynı yazma enerjisini kullanması bizi bu enerji verimsizliğini engellemeye dönük yöntemler bulmaya itti.

Öncelikle mimari seviyede bu benzerlikleri yakalamaya dönük bir tasarım yaptık. Bu amaçla hedef yazmacı ve kaynak yazmacı aynı olan (AHK) buyruklar için okunan kaynak değeri AMB'den çıkan sonuçla karşılaştıran bir XNOR devresi ekledik. MASK diye adlandırdığımız karşılaştırma sonucu bize eski ve yeni değerleri aynı olan bitler için "1" farklı olanlar için "0" sonucunu verir. Bu sonucu SRAM devresine dahil ederek değişmeyen bitlere yazmak için harcanan enerjinin büyük kısımını oluşturan yazma bit-hattında ve bit-hücresindeki enerji sarfiyatını azalttık.

Bunun yanında hedef yazmacı ve kaynak yazmacı farklı olan (FHK) buyruklar için daha farklı bir mimari tasarım oluşturmamız gerekti, çünkü FHK buyrukları için hedef yazmacın eski değeri normalde okunmaz ve bu nedenle de eski ve yeni değerleri karşılaştırmak ancak fazladan bir okuma gerçekleştirirsek mümkün olur. Fakat fazladan bir okuma fazladan enerji tüketimine neden olur ve toplamda enerji tüketimini azaltamayız. FHK buyrukları için geçmişte yapılan araştırmalardan elde edilmiş bir başka bulgudan yola çıktık; buyruklar genellikle küçük değerlerle işlem yaparlar ve yazılan bitlerin çoğu aslında "0" olur. Bu bilgiyi deneylerle doğruladık ve gerçekten 64 bitlik yazmaçların yazılan bitlerinin yalnızca ortalama 6,3 tanesinin "1", diğerlerinin ise "0" olduğunu gördük. Bu bilgiye dayanarak yazmaçları çok az enerjiyle sıfırlayan devresel bir tasarım yaptık. Ardından yalnızca "1" olan bitleri yazan ve diğerlerini MASK sinyaliyle maskeleyen bir enerji tasarrufu mekanizması tasarladık.

Son olarak sıfırlama mekanizmasını AHK buyrukları için de genişleterek, sonucu "0" olan bütün buyrukları sıfırlama devresi sayesinde daha da düşük enerjiyle yazdık. Bu enerji tasarruf mekanizmalarının tamamını Güncelleme-tabanlı Yazmaç Öbeği Tasarımı (GÜNYET) olarak adlandırdık. GÜNYET mekanizması sayesinde 64 yazmaçlı bir yazmaç öbeğinin yazma enerjisi % 32,40 azaldı.

Yazmaçlar arasındaki değer taşıma, aynı bellek adresinden değer okuyup yazmaçlara yazma, push-pop işlemleri nedeniyle yığıt göstergesini farklı yazmaçlarda tutma gibi işlemlerden dolayı yazmaçlarda aynı anda aynı değerden birden fazla bulunabilir. Bunun yanında yazmaçlarda yapılan işlemler küçük değerler üzerinde gerçekleştiğinden sadece en alt baytı farklı olan ama geri kalanı aynı olan yazmaçlar da çok sayıda olabilir. Bu öngörülerimizi doğrulamak için çok sayıda gösterge programıyla birçok benzetimler yaptık ve her an aktif yazmaçların ortalama % 66,1'inin yazmaç öbeğinde kopya değerinin bulunduğunu bulduk. Benzer yazmaçları ele aldığımızda ise her an aktif yazmaçların ortalama % 98,6'sının 8-bit Hamming uzaklığında benzer bir yazmacı veya yazmaçları olduğunu gördük. Bu yedeklilik bir verimsizlik olsa da sistemin güvenilirliğini artırmak için faydalanılabilir bir durumdur.

Bu bulgulara dayanarak yazmaç öbeğinin güvenilirliğini artırmak için bir tasarım yaptık. Bu tasarımda öncelikle yazmaçlardaki aynılıkları (benzerlikleri) bulmak gerekti ve bunun için sonuç değeri en az bir kaynak yazmaç değeriyle aynı (benzer) olan yazmaçları kullandık. Burada *benzerlik* yalnızca en alt baytı birbirinden farklı geri kalan baytları aynı olan değerleri ifade eder. Bunu bir karşılaştırma devresiyle bulup, yeni eklenen bazı alanlarda kopya veya benzer değerlerin yerini tuttuk. Hatanın yakalanmasını ise tek-bit parite kullanarak bulmak suretiyle yakalanan hataların tutulan kopya veya benzer değerler sayesinde düzeltilmesini hedefledik. Kopya değerlerle düzeltme yaptığımız tasarıma Kopyayala Düzeltme (KOD), benzer değerlerle düzeltme yaptığımız tasarıma ise Yakın Değerlerle Düzeltme (YADED) adını verdik. KOD tasarımının ayrıca geçmiş bir çalışmada anlatılan ve aktif yazmaçların kullanılmayan yazmaçlara kopyalanarak yedekliliğin artırılmasını öneren bir hata düzeltme mekanizmasıyla (YİM) birlikte çalışabileceğini gösterdik. Bu şekilde KOD tasarımı sayesinde güvenilir bir şekilde okunabilen yazmaç oranı % 44,1 ve hata düzeltme oranı da % 39,0'dır. KOD tasarımının tek başına güç tüketimi % 2,8 artarken YİM ile birlikte kullanıldığında bu oran % 18,9'a çıkar. YADED tasarımı ise güç tüketimini KOD tasarımı seviyelerinde tutarken güvenilir okuma oranı % 39,8 seviyesindedir. Bununla birlikte YADED tasarımı da YİM yöntemiyle birlikte kullanılırsa güvenilir okuma oranı ortalamada % 63,3 seviyesine ulaşır.

Güç tüketimi ve güvenilirliğin önemi, sistem tasarımlarının çeşitli birimlerinde

giderek artmaktadır. Çalışmalarımızda yazmaç öbeğinin benzer veya aynı değerler nedeniyle sahip olduğu verimsizliği tespit ettik ve bundan güç tüketimini azaltma ve güvenilirlik için faydalanmaya çalıştık. Veri benzerliğine dayalı bu bulgudan önbellek, yığıt göstergesi ve diğer bellek elemanlarında da faydalanılabilmesi mümkündür. Bu nedenle önerdiğimiz yöntemler elektronik sistemlerin çeşitli birimlerinde kullanılabilir ve gözlemlediğimiz verimsizliklerden daha fazla yararlanılmasını sağlayacak yeni yöntemler gelecekte önerilebilir. Yeni önerilecek bu yöntemlerle bu tezde belirtilen güç tasarrufu ve güvenilirlik artışının daha da ilerletilmesi mümkündür.

KAYNAKLAR

[1] **D. Marculescu and E. Talpes**, "Energy awareness and uncertainty in microarchitecture-level design," *Micro, IEEE*, vol. 25, pp. 64 – 76, sept.-oct. 2005.

[2] **K. Patel, W. Lee**, **and M. Pedram**, "Minimizing power dissipation during write operation to register files," in *Proceedings of the 2007 international symposium on Low power electronics and design*, ISLPED '07, (New York, NY, USA), pp. 183–188, ACM, 2007.

[3] **K. Kanda, H. Sadaaki**, **and T. Sakurai**, "90% write power-saving sram using sense-amplifying memory cell," *Solid-State Circuits, IEEE Journal of*, vol. 39, pp. 927 – 933, june 2004.

[4] **T. R. Halfhill**, "Intel's tiny atom," *Microprocessor Report*, vol. 22, no. 4, p. 1, 2008.

[5] **P. Greenhalgh**, "Big. little processing with arm cortex-a15 & cortex-a7: Improving energy efficiency in high-performance mobile platforms," *white paper, ARM*, September 2011.

[6] **G. Saggese, N. J. Wang, Z. T. Kalbarczyk, S. J. Patel**, **and R. K. Iyer**, "An experimental study of soft errors in microprocessors," *IEEE Micro*, vol. 25, no. 6, pp. 30–39, 2005.

[7] **M. Rebaudengo, M. S. Reorda**, **and M. Violante**, "An accurate analysis of the effects of soft errors in the instruction and data caches of a pipelined microprocessor," in *Design, Automation and Test in Europe (DATE)*, 2003.

[8] **G. Memik, M. Kandemir**, **and O. Ozturk**, "Increasing register file immunity to transient errors," in *Design, Automation and Test in Europe, 2005. Proceedings*, pp. 586–591 Vol. 1, March 2005.

[9] **D. Gonzales**, "Micro-risc architecture for the wireless market," *Micro, IEEE*, vol. 19, pp. 30 –37, jul-aug 1999.

[10] **H. Zeng and K. Ghose**, "Register file caching for energy efficiency," in *Proceedings of the 2006 international symposium on Low power electronics and design*, ISLPED '06, (New York, NY, USA), pp. 244–249, ACM, 2006.

[11] **T. Jones, M. O'Boyle, J. Abella, A. Gonzalez**, **and O. Ergin**, "Energy-efficient register caching with compiler assistance," *ACM Trans. Archit. Code Optim.*, vol. 6, pp. 13:1–13:23, Oct. 2009.

[12] **M. H. Lipasti and J. P. Shen**, "Exceeding the dataflow limit via value prediction," in *Proceedings of the 29th annual ACM/IEEE international symposium on Microarchitecture*, MICRO 29, (Washington, DC, USA), pp. 226–237, IEEE Computer Society, 1996.

[13] **I. Park, M. Powell**, **and T. Vijaykumar**, "Reducing register ports for higher speed and lower energy," in *Microarchitecture, 2002. (MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on*, pp. 171–182, 2002.

[14] **N. S. Kim and T. Mudge**, "Reducing register ports using delayed write-back queues and operand pre-fetch," in *Proceedings of the 17th Annual International Conference on Supercomputing*, ICS '03, (New York, NY, USA), pp. 172–182, ACM, 2003.

[15] **O. Ergin**, "Exploiting narrow values for energy efficiency in the register files of superscalar microprocessors," *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, pp. 477–485, 2006.

[16] **M. Kondo and H. Nakamura**, "A small, fast and low-power register file by bit-partitioning," in *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, pp. 40–49, Feb 2005.

[17] **J. Kulkarni, C. Tokunaga, P. Aseron, T. Nguyen, C. Augustine, J. Tschanz**, **and V. De**, "4.7 a 409 gops/w adaptive and resilient domino register file in 22nm tri-gate cmos featuring in-situ timing margin and error detection for tolerance to within-die variation, voltage droop, temperature and aging," in *Solid- State Circuits Conference - (ISSCC), 2015 IEEE International*, pp. 1–3, Feb 2015.

[18] **K. Chen, E. Atoofian**, **and A. Manzak**, "Improving power of cache and register file through critical path instructions," in *Digital System Design (DSD), 2014 17th Euromicro Conference on*, pp. 349–355, Aug 2014.

[19] **S. Balakrishnan and G. S. Sohi**, "Exploiting value locality in physical register files," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, pp. 265 – 276, dec. 2003.

[20] **R. Gonzalez, A. Cristal, D. Ortega, A. Veidenbaum**, **and M. Valero**, "A content aware integer register file organization," in *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pp. 314–324, June 2004.

[21] **T. Monreal, V. Vinals, A. Gonzalez**, **and M. Valero**, "Hardware schemes for early register release," in *Parallel Processing, 2002. Proceedings. International Conference on*, pp. 5–13, 2002.

[22] **O. Ergin, D. Balkan, D. Ponomarev**, **and K. Ghose**, "Increasing processor performance through early register release," in *Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings. IEEE International Conference on*, pp. 480–487, Oct 2004.

[23] **R. Canal, A. González**, **and J. E. Smith**, "Very low power pipelines using significance compression," in *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, MICRO 33, (New York, NY, USA), pp. 181–190, ACM, 2000.

[24] **S. Mukherjee, J. Emer**, **and S. Reinhardt**, "The soft error problem: an architectural perspective," in *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, pp. 243–247, Feb 2005.

[25] **N. Seifert, X. Zhu**, **and L. Massengill**, "Impact of scaling on soft-error rates in commercial microprocessors," *Nuclear Science, IEEE Transactions on*, vol. 49, pp. 3100–3106, Dec 2002.

[26] **V. Chandra and R. Aitken**, "Impact of technology and voltage scaling on the soft error susceptibility in nanoscale cmos," in *Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS '08. IEEE International Symposium on*, pp. 114–122, Oct 2008.

[27] **R. Naseer, R. Bhatti**, **and J. Draper**, "Analysis of soft error mitigation techniques for register files in ibm cu-08 90nm technology," in *Circuits and Systems, 2006. MWSCAS '06. 49th IEEE International Midwest Symposium on*, vol. 1, pp. 515–519, Aug 2006.

[28] **J. Hu, S. Wang**, **and S. Ziavras**, "In-register duplication: Exploiting narrow-width value for improving register file reliability," in *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pp. 281–290, June 2006.

[29] **K. Mohr, G. Samson**, **and L. Clark**, "A radiation hardened by design register file with lightweight error detection and correction," *Nuclear Science, IEEE Transactions on*, vol. 54, pp. 1335–1342, Aug 2007.

[30] **S. Feng, S. Gupta, A. Ansari**, **and S. Mahlke**, "Shoestring: Probabilistic soft error reliability on the cheap," in *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, (New York, NY, USA), pp. 385–396, ACM, 2010.

[31] **J. A. Blome, S. Gupta, S. Feng**, **and S. Mahlke**, "Cost-efficient soft error protection for embedded microprocessors," in *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '06, (New York, NY, USA), pp. 421–431, ACM, 2006.

[32] **M. Fazeli, A. Namazi**, **and S. G. Miremadi**, "An energy efficient circuit level technique to protect register file from mbus and sets in embedded processors," in *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on*, pp. 195–204, June 2009.

[33] **J. Yan and W. Zhang**, "Compiler-guided register reliability improvement against soft errors," in *Proceedings of the 5th ACM International Conference on Embedded Software*, EMSOFT '05, (New York, NY, USA), pp. 203–209, ACM, 2005.

[34] **J. Ray, J. C. Hoe**, **and B. Falsafi**, "Dual use of superscalar datapath for transient-fault detection and recovery," in *Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 34, (Washington, DC, USA), pp. 214–224, IEEE Computer Society, 2001.

[35] **H. Amrouch and J. Henkel**, "Self-immunity technique to improve register file integrity against soft errors," in *VLSI Design (VLSI Design), 2011 24th International Conference on*, pp. 189–194, Jan 2011.

[36] **S. K. Reinhardt and S. S. Mukherjee**, "Transient fault detection via simultaneous multithreading," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ISCA '00, (New York, NY, USA), pp. 25–36, ACM, 2000.

[37] **J. Lee and A. Shrivastava**, "Compiler-managed register file protection for energy-efficient soft error reduction," in *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, ASP-DAC '09, (Piscataway, NJ, USA), pp. 618–623, IEEE Press, 2009.

[38] **O. Ergin, O. Unsal, X. Vera**, **and A. Gonzalez**, "Exploiting narrow values for soft error tolerance," *Computer Architecture Letters*, vol. 5, no. 2, pp. 12–12, 2006.

[39] **I. B. Karsli, P. Reviriego, M. F. Balli, O. Ergin**, **and J. A. Maestro**, "Enhanced duplication: a technique to correct soft errors in narrow values," *IEEE Computer Architecture Letters*, vol. 12, pp. 13–16, January 2013.

[40] **M. Kayaalp, F. Koc**, **and O. Ergin**, "Improving the soft error resilience of the register files using sram bitcells with built-in comparators," in *Digital System Design (DSD), 2012 15th Euromicro Conference on*, pp. 140–143, IEEE, 2012.

[41] **H. Ogur, D. D. Yavuz, G. Boztepe, S. Gesoglu, A. Eker, O. Ergin, G. Yalcin**, **and O. S. Unsal**, "Exploiting existing replicas of stack pointer in the register file for error detection," in *Workshop on Manufacturable and Dependable Multicore Architectures at Nanoscale*, 2015.

[42] **D. Wendel, R. Kalla, R. Cargoni, J. Clables, J. Friedrich, R. Frech, J. Kahle, B. Sinharoy, W. Starke, S. Taylor, S. Weitzel, S. G. Chu, S. Islam**, **and V. Zyuban**, "The implementation of power7tm: A highly parallel and scalable multi-core high-end server processor," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pp. 102–103, Feb 2010.

[43] **R. Haring, M. Ohmacht, T. Fox, M. Gschwind, D. Satterfield, K. Sugavanam, P. Coteus, P. Heidelberger, M. Blumrich, R. Wisniewski, a. gara, G. Chiu, P. Boyle, N. Chist**, **and C. Kim**, "The ibm blue gene/q compute chip," *IEEE Micro*, vol. 32, pp. 48–60, March 2012.

[44] **V. Zyuban and P. Kogge**, "Split register file architectures for inherently lower power microprocessors," in *Power-Driven Microarchitecture Workshop, in conjunction with 1SCA'98*, pp. 32–37, 1998.

[45] **J.-L. Cruz, A. Gonzalez, M. Valero**, **and N. Topham**, "Multiple-banked register file architectures," in *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, pp. 316–325, June 2000.

[46] **R. Balasubramonian, S. Dwarkadas**, **and D. Albonesi**, "Reducing the complexity of the register file in dynamic superscalar processors," in *Microarchitecture, 2001. MICRO-34. Proceedings. 34th ACM/IEEE International Symposium on*, pp. 237–248, Dec 2001.

[47] **NVIDIA**, *NVIDIA's Next Generation CUDATM Compute Architecture: Kepler TM GK110*. NVIDIA Corporation, 2012.

[48] **P. N. Glaskowsky**, *NVIDIA's Fermi: The First Complete GPU Computing Architecture*. NVIDIA Corporation, Sept. 2009.

[49] **NVIDIA**, *Tesla K20 GPU Accelerator Board Specification*. NVIDIA Corporation, July 2013.

[50] **NVIDIA**, *TESLA C2050 / C2070 GPU Computing Processor*. NVIDIA Corporation, July 2010.

[51] **R. Riedlinger, R. Arnold, L. Biro, B. Bowhill, J. Crop, K. Duda, E. S. Fetzer, O. Franza, T. Grutkowski, C. Little, C. Morganti, G. Moyer, A. Munch, M. Nagarajan, C. Parks, C. Poirier, B. Repasky, E. Roytman, T. Singh**, **and M. W. Stefaniw**, "A 32 nm, 3.1 billion transistor, 12 wide issue itanium processor for mission-critical servers," *IEEE Journal of Solid-State Circuits*, vol. 47, pp. 177–193, Jan 2012.

[52] **R. Phelan**, "Addressing soft errors in arm core-based soc," *ARM White Paper*, 2003.

[53] **E. Fetzer, D. Dahle, C. Little**, **and K. Safford**, "The parity protected, multithreaded register files on the 90-nm itanium microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 1, pp. 246–255, 2006.

[54] **P. Montesinos, W. Liu**, **and J. Torrellas**, "Using register lifetime predictions to protect register files against soft errors," in *Dependable Systems and Networks, 2007. DSN '07. 37th Annual IEEE/IFIP International Conference on*, pp. 286–296, June 2007.

[55] **T. Yeh and Y. Patt**, "Two-level adaptive training branch prediction," in *Proc. 24th Annual Int. Symposium on Microarchitecture*, pp. 51–61, 1991.

[56] **D. She, Y. He, B. Mesman**, **and H. Corporaal**, "Scheduling for register file energy minimization in explicit datapath architectures," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*, pp. 388–393, IEEE, 2012.

[57] **S. Borkar and A. A. Chien**, "The future of microprocessors," *Commun. ACM*, vol. 54, pp. 67–77, May 2011.

[58] **J. S. Gardner**, "Mips has a future with imagination," *Microprocessor Report*, february 2013.

[59] **M. Nemirovsky and D. M. Tullsen**, "Multithreading architecture," *Synthesis Lectures on Computer Architecture*, vol. 8, no. 1, pp. 1–109, 2013.

[60] **Y. Li, D. Brooks, Z. Hu**, **and K. Skadron**, "Performance, energy, and thermal considerations for smt and cmp," in *Proceedings of the 11th International Symposium on High-Performance Computer Architecture (HPCA)*, (San Francisco), pp. 71 – 82, 2005.

[61] **V. Zyubatn and P. Kogge**, "The energy complexity of register files," in *Proceedings of the 1998 international symposium on Low power electronics and design (ISLPED'98)*, (Monterey, CA), 1998.

[62] **V. Zyuban and P. Kogge**, "The energy complexity of register files," in *Low Power Electronics and Design, 1998. Proceedings. 1998 International Symposium on*, pp. 305 –310, aug. 1998.

[63] **D. Patterson and J. Hennessy**, *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, 5 ed., 2013.

[64] **Intel**, *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2: Instruction Set Reference A-Z*, January 2013. order no. 325383-045US.

[65] **F. Koc, O. Seckin**, **and O. Ergin**, "Using content-aware bitcells to reduce static energy dissipation," in *Proceedings of the IEEE 29th International Conference on Computer Design (ICCD)*, (Amherst, MA), 2011.

[66] **J. Jung, Y. Nakata, M. Yoshimoto**, **and H. Kawaguchi**, "Energy-efficient spin-transfer torque ram cache exploiting additional all-zero-data flags," *Quality Electronic Design, International Symposium on*, pp. 216–222, 2013.

[67] **A. Patel, F. Afram, S. Chen**, **and K. Ghose**, "Marssx86: A full system simulator for x86 cpus," in *Design Automation Conference 2011 (DAC'11)*, 2011.

[68] **F. Bellard**, "Qemu, a fast and portable dynamic translator," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, (Berkeley, CA, USA), pp. 41–41, USENIX Association, 2005.

[69] **Standard Performance Evaluation Corporation**, "Spec benchmarks," *http://www.spec.org*, 2006.

[70] **R. Rogenmoser, L. O'Donnell**, **and S. Nishimoto**, "A dual-issue floating-point coprocessor with simd architecture and fast 3d functions," in *Solid-State Circuits Conference, 2002. Digest of Technical Papers. ISSCC. 2002 IEEE International*, vol. 1, pp. 414–415 vol.1, 2002.

[71] **P. Greenhalgh**, "big.little processing with arm cortex-a15 & cortex-a7," tech. rep., ARM, 2011.

[72] **R. K. Iyer, N. M. Nakka, N. T. Kalbarczyk**, **and S. Mitra**, "Recent advances and new avenues in hardware-level reliability support," *Micro, IEEE*, vol. 25, no. 6, pp. 18–29, 2005.

[73] **R. E. Lyons and W. Vanderkulk**, "The use of triple-modular redundancy to improve computer reliability," *IBM Journal of Research and Development*, vol. 6, no. 2, pp. 200–209, 1962.

[74] **J. Hu, L. Feihul, V. Degalahal, M. Kandemir, N. Vijaykrishnan**, **and M. Irwin**, "Compiler-directed instruction duplication for soft error detection," in *DATE*, 2005.

[75] **R. W. Hamming**, "Error detecting and error correcting codes," *Bell System technical journal*, vol. 29, no. 2, pp. 147–160, 1950.

[76] **Y. Kocberber, Y. Osmanlioglu**, **and O. Ergin**, "Exploiting narrow values for faster parity generation," *Microelectronics International*, vol. 26, no. 3, pp. 22–29, 2009.

[77] **Y. Osmanlioglu, Y. Kocberber**, **and O. Ergin**, "Reducing parity generation latency through input value aware circuits," in *19th ACM Great Lakes Symposium on VLSI (GLSVLSI09)*, (Boston, MA), 2009.

[78] **G. Yalcin, O. Unsal, A. Cristal**, **and M. Valero**, "Fimsim: A fault injection infrastructure for microarchitectural simulators," in *ICCD*, 2011.

[79] **M. Li, P. Ramachandran, U. Karpuzcu, S. S. Hari**, **and S. Adve**, "Accurate microarchitecture-level fault modeling for studying hardware faults," in *HPCA*, (Raleigh, NC), 2009.

[80] **A. Dixit and A. Wood**, "The impact of new technology on soft error rates," in *Reliability Physics Symposium (IRPS), 2011 IEEE International*, pp. 5B.4.1–5B.4.7, April 2011.

[81] **Q. Zhou and K. Mohanram**, "Gate sizing to radiation harden combinational logic," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 1, pp. 155–166, 2006.

[82] **N. Miskov-Zivanov and D. Marculescu**, "Modeling and optimization for soft-error reliability of sequential circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 5, pp. 803–816, 2008.

ÖZGEÇMİŞ

Ad-Soyad : Abdulaziz EKER

Uyruğu : Türkiye Cumhuriyeti

Doğum Tarihi ve Yeri : 30.05.1982 - Samsun

E-posta : aeker@etu.edu.tr

ÖĞRENİM DURUMU:

- **Lisans** : 2005, Purdue University, Electrical and Computer Engineering Department, Computer Engineering
- **Yüksek Lisans** : 2007, North Carolina State University, Computer Engineering

MESLEKİ DENEYİM:

Yıl	Yer	Görev
2003-2004	Purdue University Computer Labs	Lab Assistant
2005-2006	North Carolina State Univesity	Teaching Assistant
2007-2012	TÜBİTAK Uzay Teknolojileri Araş. Enst.	Uzman Araştırmacı
2012-	TÜBİTAK BİLGEM İleri Teknolojiler Araş. Enst.	Uzman Araştırmacı

YABANCI DİL: İngilizce, İspanyolca

TEZDEN TÜRETİLEN YAYINLAR, SUNUMLAR VE PATENTLER:

- Eker, A., Ergin, O., 2015. Using value similarity of registers for soft error mitigation, Proceedings of 28th IEEE Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems , Amherst, MA, ABD.
- Ogur, H., Boztepe, G., Yavuz, D., Gesoglu, S., Eker, A., Yalcin, G., Unsal, O., Ergin, O., 2015. Exploiting Existing Replicas of Stack Pointer in Register File for Error Detection, Workshop on Manufacturable and Dependable Multicore Architectures at Nanoscale, Tallinn, Estonia.

- Eker, A., Ergin, O., 2016. Error Recovery Through Partial Value Similarity, Proceedings of 29th IEEE Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems , Amherst, MA, ABD.
- Eker, A., Ergin, O., 2016. Exploiting Existing Copies in Register File for Soft Error Correction, IEEE Computer Architecture Letters (CAL), vol. 15, no. 1, pp. 17-20.
- Eker, A., Mert, Y. M., Ergin, O., 2016. URFA - Update Based Register File Architecture with Partial Register Write for Energy Efficiency, Elsevier Microprocessors and Microsystems Journal. (Kabul edildi)

DİĞER YAYINLAR, SUNUMLAR VE PATENTLER:

- Liu, F., Guo, F., Solihin, Y., Kim, S., Eker, A., 2008. Characterizing and modeling the behavior of context switch misses. Proceedings of the 17th international conference on Parallel architectures and compilation techniques, Toronto, Kanada.