MİKROİŞLEMCİLERDE SANAL ADRESLERDEN YARARLANILARAK GEÇİCİ HATALARA KARŞI KORUMA SAĞLANMASI

YAMAN ÇAKMAKÇI

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜHENDİSLİGİ

TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

TEMMUZ 2012

ANKARA

Fen Bilimleri Enstitü onayı

_____

Prof. Dr. Ünver KAYNAK
Müdür

Bu tezin Yüksek Lisans derecesinin tüm gereksinimlerini sağladığını onaylarım.

_____

Doç. Dr. Erdoğan Doğdu
Anabilim Dalı Başkanı

Yaman ÇAKMAKÇI tarafından hazırlanan MİKROİŞLEMCİLERDE SANAL ADRESLERDEN YARARLANILARAK GEÇİCİ HATALARA KARŞI KORUMA SAĞLANMASI adlı bu tezin Yüksek Lisans tezi olarak uygun olduğunu onaylarım.

_____

Doç. Dr. Oğuz Ergin
Tez Danışmanı

Tez Jüri Üyeleri

Başkan : Yrd. Doç. Dr. Hüsrev Taha Sencar _____

Üye : Doç. Dr. Oğuz Ergin _____

Üye : Doç. Dr. Reşit Şendağ _____

TEZ BİLDİRİMİ

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

Yaman ÇAKMAKÇI

Üniversitesi : **TOBB Ekonomi ve Teknoloji Üniversitesi**
Enstitüsü : **Fen Bilimleri**
Anabilim Dalı : **Bilgisayar Mühendisliği**
Tez Danışmanı : **Doç. Dr. Oğuz Ergin**
Tez Türü ve Tarihi : **Yüksek Lisans – Temmuz 2012**

Yaman ÇAKMAKÇI

MİKROİŞLEMCİLERDE SANAL ADRESLERDEN YARARLANILARAK GEÇİCİ HATALARA KARŞI KORUMA SAĞLANMASI

ÖZET

Bu tez çalışmasında bir sistemdeki etkin sayfalar önbelleği, fiziksel yazmaç dosyası ve program sayacı gibi sanal adres saklayan yapıların oluşabilecek geçici hatalara karşı korunmasını sağlayan yeni bir yöntem önerilmektedir. Çalışma geçici hataların sanal adres saklayan yapılardaki etkilerini göstererek yola çıkmaktadır. Önerilen çözüm, bağlama zamanında sanal adreslerin atanmasında doğrusal blok kodlayıcı kulllanılmasıdır. Bahsi geçen kodlayıcının sanal adreslerin sayfa numarası kısmının atanmasında kullanılması ve bu kodlayıcının sayfa hatası kotarıcısına yerleştirilmiş bir çözücü ile desteklenmesi sayesinde sistemin hatalara karşı tahammüllünün artırıldığı gözlemlenmiştir. Ayrıca, donanımda gerçeklenmiş çözücüler kullanılarak program sayacı ve fiziksel yazmaç dosyasına kodlama yükü getirmeden hata düzeltici kodların sağladığı faydadan yararlanılabilinmesi öngörülmüştür.
Bu tezin temel katkısı incelenen işlemci yapılarının MHAE'lerinde elde edilen düşüştür. etkin veri sayfaları önbelleği 42.5%, etkin buyruk sayfaları önbelleği 40.3%, program sayacında 69.2% ve fiziksel yazmaç dosyasında 33.3% düzeyinde bir iyileştirme sağlanmıştır.

Anahtar Kelimeler: Hataya dayanıklılık, Sanal adresleme, Doğrusal blok kodlar, geçici hatalar.

University	:	**TOBB University of Economics and Technology**
Institute	:	**Institute of Natural and Applied Sciences**
Science Programme	:	**Computer Engineering**
Supervisor	:	**Associate Prof. Oğuz Ergin**
Degree Awarded and Date	:	**M.Sc. – July 2012**

Yaman ÇAKMAKÇI

EXPLOITING VIRTUAL ADDRESSES FOR ACHIEVING FAULT TOLERANCE IN MICROPROCESSORS

ABSTRACT

In this thesis a novel method to protect a system against errors resulting from soft errors occuring in the virtual address storing structures such as translation buffers, physical register file and the program counter. The work is motivated by showing how soft errors impact the structures that store virtual addresses. A solution is proposed by employing linear block encoding methods to be used as a virtual addressing scheme at link time. Using the encoding scheme to assign virtual page numbers for virtual addresses, it is shown that the system can tolerate soft errors using software with the help of the discussed decoding techniques applied to the page fault handler. Hardware solutions can also be applied at the program counter and the physical regiser file providing ECC protection without the burden of using circuit level encoding.

The main contribution of this thesis is the decreasing of AVF for data translation buffer by 42.5%, instruction translation buffer by 40.3%, program counter by 69.2% and physical register file by 33,3%.

Keywords: Fault tolerance, Virtual addressing, Linear block codes, Soft Errors.

TEŞEKKÜR

Bu çalışmayı gerçekleştirmemde emeği olan danışmanım Oğuz Ergin'e ve beni her zaman destekleyen aileme teşekkür ederim.

İÇİNDEKİLER

ŞEKİLLERİN LİSTESİ

ÇİZELGELERİN LİSTESİ

Kısaltma	**Kısaltma(İng)**	**Açıklama**	**İngilizce**
MHAE	MHAE	Mimari Hataya Açıklık Etkeni	Architectural Vulnerability Factor
MDDY	MDDY	Mimari Düzeyinde Doğru Yürütüm	Architecturally Correct Execution
PHAE	PVF	Program Hataya Açıklık Etkeni	Program Vulnerability Factor
DHAE	HVF	Donanım Hataya Açıklık Etkeni	Hardware Vulnerability Factor
SAU	VAS	Sanal Adres Uzayı	Virtual Address Space
YSY	WAW	Yazma Sonrası Yazma	Write After Write
OSY	WAR	Okuma Sonrası Yazma	Write After Read
FDH	DUE	Farkedilen Düzeltilemeyecek Hatalar	Detectable Unrecoverable Errors
GVY	SDC	Gizli Veri Yozlaşması	Silent Data Corruption
ESÖ	ESÖ	Etkin Sayfalar Önbelleği	Translation Lookaside Buffer
EVSÖ	DESÖ	Etkin Veri Sayfaları Önbelleği	Data Translation Lookaside Buffer
EBSÖ	IESÖ	Etkin Buyruk Sayfaları Önbelleği	Instruction Translation Lookaside Buffer
FSN	PPN	Fiziksel Sayfa Numarası	Physical Page Number
SSN	VPN	Sanal Sayfa Numarası	Virtual Page Number
TBÇV	SIMD	Tek Buyruk Çoklu Veri	Single Instruction Multiple Data

1. GİRİŞ

Artan saat vuruş sıklıklarıyla desteklenen kayda değer başarım artışları, daha ufak boyutlarda üretim yapılabilmesi ve düşük kaynak gerilimleri, geçici hatalar olarak bilinen sorunların yaygın olarak gözlemlenmesine sebep olmuştur. Geçici hatalar [39][18] alfa ışıması sonucu ortaya çıkan alfa parçacıklarının ve kozmik ışınlarda barınan yüksek enerjili nötrönlar ile protonların işlemci üzerindeki saklama alanlarına çarpması sonucu ortaya çıkmaktadırlar. Bu parçacıkların çarptıkları sığanın dolmasına veya boşalmasına sebep olmalarından ötürü yarattıkları hatalar kalıcı değildir. Geçici hataların işlemciler üzerinde yaygın olarak görülmeleriyle birlikte, güvenilirlik de başarım ve enerji tasarrufu ile beraber işlemci tasarımının temel tasarım ilkesi olarak kabul edilmeye başlanmıştır. Yonga üzerindeki kullanılabilir silikon alanın artışı ile birlikte geçici hata sorunun ileride artış göstereceği öngörülmektedir[11][1]. Geçici hatalara duyarlı sistemlerde bu hatalardan korunmak için yararlanılabilecek yöntemler kapsamlı bir şekilde [10] ile sunulmuştur.

Bilimsel yönteme göre incelenen olguların ölçülebilir olması büyük önem teşkil etmektedir. Bir işlemci yapısının geçici hatalara karşı dayanıklılığı Mimari Hataya Açıklık Etkeni (MHAE)[28] ile belirtilir. Donanımsal yapıların MHAE'lerinin ölçülmesi için Mimari Düzeyinde Doğru Yürütüm (MDDY) analizi yöntemi kullanılır[2]. MDDY bitleri saklanan bir değerin hata sonucu oluşan bir değişiminin programın doğruluğunu etkileyip etkilemediğini belirdir. Örneğin, öngörücü kullanmayan bir sistemin program sayacının bütün bitleri çalışma zamanının yaklaşık olarak %100'ünde MDDY bitleridir. Program sayacında oluşacak herhangi bir bit değişimi programın akışını tamamen değiştirecek ve

yanlış bir çıktıya sebep olacaktır.

Çeşitli araştırmacıların MHAE'yi temel alarak ortaya çıkardıkları farklı ölçüm yöntemleri de geliştirilmiştir. MHAE hata maskelemeyi de dikkate alacak şekilde Program Hataya Açıklık Faktörü (PHAE)[37] olarak genişletilmiştir. Ardından Donanım Hataya Açıklık Faktörü (DHAE)[38] MHAE ve PVF olgularını birleştirmek için geliştirilmiştir.

Derleyiciler, tanımlanmış bir dildeki kaynak programın farklı bir hedef dildeki dengine tercüme etmeye yarayan programlardır. Derleyiciler sayesinde yazılımcının üst seviye bir programlama dilinde kolayca ifade edebileceği kavramların işlemcinin çalıştırabileceği makine diline çevrilmesi ve makine dilinin karmaşıklığından uzak bir şekilde yapılacak işe odaklanması mümkün kılınmaktadır. Bu temel özelliklerinin yanında, oluşturdukları yürütülebilir programın başarımının yüksek, kod boyutunun küçük olması önemlidir. Fazla yaygın olmamakla birlikte derleyici üzerinden sistemin geçici hata toleransının arttırılmasının mümkün olduğu farklı çalışmalarda gösterilmiştir [3][25][22].

Bu tezde bir sanal adres uzayındaki (SAU) sanal adreslerin içindeki sayfa numaralarının doğrusal blok kodlar kullanılarak bağlama zamanında kodlanması ve daha sonra yazılım ve donanım üzerinde gerçeklenmiş sendrom çözücüler kullanılarak sistemin içinde bulunan farklı yapıların MHAE'sinin nasıl düşürülebileceği anlatılmaktadır.

Çalışmanın devamında 2. bölümde çalışma içinde kullanılacak kavramlar açıklanmıştır. 3. bölüm derleyici ve işlemci mimarisi etkileşimi ile ilişkili çalışmaların bir özetinden oluşmaktadır. Daha sonra, 4. bölümde önerilen yöntem anlatılmıştır. 5. bölümde deneysel sonuçların bir değerlendirilmesi ve 6. bölümde çalışmanın ileride nasıl şekillenebileceğiyle ilgili yorumlar yer almaktadır.

2. TEMEL KONULAR

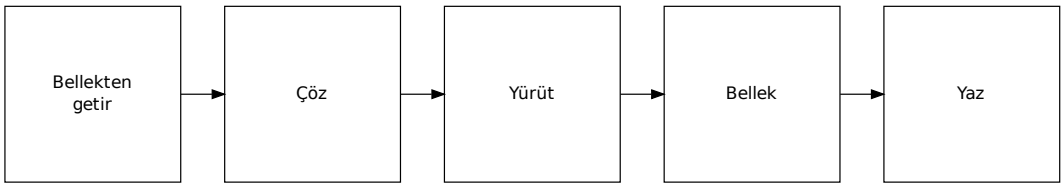
Bu bölümde yapılan tez çalışmasının kapsamında anlaşılması gereken konular ile ilgili temel bilgiler yer alacaktır.

2.1 Mikroişlemci Mimarisi

Bir bilgisayar programı buyruklar silsilesi olarak tanımlanır. Buyruklar mimariye göre farklılık göstermekle birlikte programın anlamı mimariler arasında değişiklik göstermemektedir. Mikroişlemcinin görevi verilen programın sırasını bozmadan buyrukları işletmektir. Bir mikroişlemci mimarisinin kendi içinde başarımını belirleyen en temel etken bir saat vuruşunda işletebildiği buyruk sayısıdır. Başarımı mikroişlemci mimarileri arasında değerlendirmek için aşağıdaki denklemden yararlanılabilir:

$$\text{Başarım} = \frac{1}{\text{buyruk sayısı}} \times \frac{\text{buyruk}}{\text{saat vuruşu}} \times \frac{1}{\text{saat vuruş sıklığı}}$$

Burada buyruk sayısını belirleyen etkenlerden ilki buyruk kümesi mimarisidir. Örneğin bazı mimarilerde karmaşık görevleri tek bir buyruk yerine getirebilirken, bazı mimariler bu görevi yerine getirmek için birden fazla buyruk kullanmak durumunda kalabilirler. İkinci etken ise derleyicinin ürettiği kod boyutudur. Derleyicinin programın anlamını bozmadan en az buyrukla ifade edebileceği kod, o program için başarım açısından ideal buyruk sayısını belirtir.



Şekil 2.1: Boru hattı evreleri

Başarımı arttırmanın bir diğer yolu da saat vuruş sıklığını arttırmaktır. Fakat bu yöntemin güç tüketimini istenmeyecek düzeylerde arttırmasının sonucu olarak yüksek saat vuruş sıklığına sahip tek çekirdekli işlemcilerden, daha basit yapıda ve daha düşük saat vuruş sıklığında çalışan çok çekirdekli işlemcilere bir yönelim oluşmuştur[29].

2.1.1 Boru Hattı

İlk işlemci tasarımları bir saat vuruşunda bir buyruk işlenecek şekilde yapılmıştır. Daha sonra bir buyruk işlenirken gerçekleşen aşamalar parçalara ayrılarak sistemin aynı anda farklı evrelerde birkaç buyruk işlemesine olanak sağlayabilen bir yöntem olarak boru hattı tekniği geliştirilmiştir.

Boru hattı evreleri Şekil 2.1 ile gösterilmiştir. Bu evreleri açıklamak gerekirse:

1. Çek: Program sayacının belirttiği adresteki buyruğu bellekten çeker ve program sayacını işlemci buyruk boyutu kadar arttırır.
2. Çöz: Bu evrede buyruğun kaynak işlenenleri yazmaç dosyasından okunurlar. Ayrıca, buyruk bir sonuç üretiyorsa bu sonucu yazabileceği bir yazmaç atanır. Buyruğun hangi yürütme birimine gönderileceği de bu evrede belirlenir.
3. Yürüt: Bu aşamada buyruğun ihtiyacı olan yürütme, ilgili yürütme biriminde gerçekleştirilir.

Çizelge 2.1: Boru hattının işleyişi

	Zaman(Saat vuruşu)								
Buyruk	1	2	3	4	5	6	7	8	9
i	Çek	Çöz	Yürüt	Bellek	Yaz	0	0	0	0
i+1	0	Çek	Çöz	Yürüt	Bellek	Yaz	0	0	0
i+2	0	0	Çek	Çöz	Yürüt	Bellek	Yaz	0	0
i+3	0	0	0	Çek	Çöz	Yürüt	Bellek	Yaz	0
i+4	0	0	0	0	Çek	Çöz	Yürüt	Bellek	Yaz

4. Bellek: Bellekten okuma ve belleğe yazma işlemleri bu aşamada gerçekleştirilir.

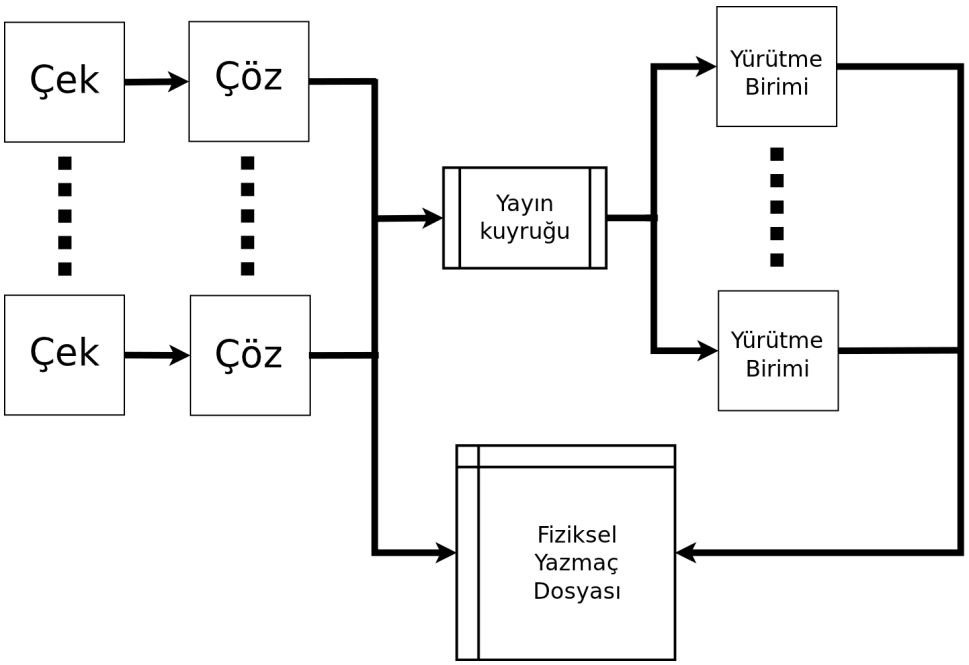
5. Yaz: Buyruk tarafından üretilen bir sonuç var ise bu sonuç çöz aşamasında atanmış olan yazmaça yazılır.

Buyrukların boru hattı içinde zamana göre nasıl ilerlediği Çizelge 2.1 ile gösterilmiştir. Çizelgede görüldüğü üzere 5nci saat vuruşuyla beraber boru hattında bulunan buyruk sayısı 5'e çıkmıştır. Bundan sonra, boru hattı duraklamaları ve boşaltmaları olmadığı varsayılarak, her saat vuruşu yeni gelen buyrukla beraber boru hattının içinde 5 buyruk olmaya devam edecektir.

Boru hattı evreleri ile ilgili daha fazla bilgi için [31] referansına başvurulabilir.

2.1.2 Çok Yollu İşlemciler

Çok yollu işlemciler bir saat vuruşunda birden fazla buyruk işletilebilmesine olanak sağlayan işlemcilerdir. Burada temel ilke buyruk düzeyindeki paralelliklerden(yani birbirinden bağımsız buyruklardan) yararlanarak başarımın arttırılmasıdır. Çok yollu işlemler boru hattına birden fazla buyruk çekebilir, aynı anda birden fazla buyruğu çözebilir ve bunları farklı yürütme birimlerine dağıtabilirler.



Şekil 2.2: Çok yollu işlemci mimarisi

Buyruk işlenmesi açısından üç ayrı akıştan söz etmek mümkündür. Bunlar:

- Buyruk akışı: Dallanma buyruklarının işlenmesinde izlenen yoldur. Uygulanan iyileştirme yöntemi dallanma öngörüsüdür.
- Yazmaç veri akışı: Aritmetik mantık buyruklarının işlenmesinde izlenen yoldur. Burada yazmaç yeniden kullanımı ve dinamik zamanlama yöntemleri kullanılmaktadır.
- Bellek veri akışı: Yükleme/saklama buyruklarının işlenmesinde izlenen yoldur. Yükleme atlatma ve yükleme iletme yöntemleri kullanılmaktadır.

Bu üç akış birbirlerinden temiz bir şekilde ayrılmamış olup, ayrı akışlar olarak ifade edilmelerine rağmen birinde oluşan bir değişiklik diğerine olumlu veya olumsuz bir şekilde sirayet edebilmektedir.

2.1.3 Dallanma öngörüsü

Dallanma öngörüsü kullanmayan sistemlerde, koşullu bir dallanma buyruğu yürütme birimine girdiğinde yürütme tamamlanana kadar boru hattının çalışmasına ara verilir. Dallanma buyruğu tamamlandıktan sonra boru hattına buyruk çekilmeye devam edilir. Bu durum başarımı düşürücü bir etki yaratmaktadır.

Dallanma öngörüsü, bir dallanma buyruğu boru hattında yürütülürken, boru hattının çalışmasına devam edebilmesini sağlamak için ortaya atılmış bir fikirdir. Belirlenmiş bir takım kriterlere göre dallanma buyruğunun nasıl sonuçlanacağı tahmin edilerek boru hattındaki duraksama engellenebilir. Dallanma öngörüsünde kullanılan iki temel kavram olan dallanma hedef tahmini ve dallanma koşul tahmini aşağıda açıklanmıştır.

Dallanma hedef tahmini, işletilen dallanma buyruğunun atlayacağı adresi kestirmek için kullanılan bir yöntemdir. Dallanmanın hangi adrese gideceğini hesaplamak uzun bir işlem olduğu için dallanma hedef belleği(BTB) adında bir önbellek kullanılır. Bu önbellek dallanmanın gideceği adresi doğrudan sağladığı için, bu adresin hesaplanması dolayısıyla oluşan saat vuruşu kayıpları engellenir.

Dallanma koşul tahmini ise işletilen dallanma buyruğunun hangi dalı seçeceğini kestirmek için kullanılmaktadır. Burada kullanılan temel yöntemler basit karara dayalı ve geçmişe dayalı olarak ikiye ayrılabilir.

Basit karara dayalı yöntemler belirlenmiş bir kritere göre dallanmanın ilerleyeceğini varsayarak buyruk çekilmesine devam edilmesini saplayan yöntemlerdir. Bu basit yöntemler bütün koşullu dallanmaların doğru sonuçlanacağı veya en kısa dallanmanın seçileceği gibi basit varsayımlara dayanan yöntemlerdir.

Geçmişe dayalı dallanma öngörüsünde dallanma buyruğunun daha önceden

izlediği yollar bir geçmiş tablosunda tutularak, dallanmanın hangi yolu seçeceğine dair kestirimler bu tablodaki veriler ışığında yapılır.

Bu konu ile ilgili daha ileri yöntemler bulunmkla beraber, bu çalışmada bunlara yer verilmeyecektir. İlgilenen okuyucu [35] [9][36] [14] kaynaklarından konuyla ilgili daha detaylı bilgi edinebilir.

2.1.4 Yazmaç Yeniden Adlandırması

Yazmaç yeniden adlandırılması buyruk düzeyi paralellikten yararlanmak için geliştirilmiş olan bir sırasız yürütme tekniğidir. Bu teknik Okuma Sonrası Yazma (OSY) ve yazma sonrası yazma (YSY) gibi hatalı bağımlılıkları ortadan kaldırır. Ardışık buyrukların sonuç işlenenleri ve/veya kaynak işlenenleri arasında olan bağımlılıklar veri bağımlılıkları olarak adlandırılır. Bunlar üç ayrı sınıfa ayrılmıştır:

1. Gerçek bağımlılık(Yazma sonrası okuma): Burada b_1 buyruğundan sonra gelen b_2 buyruğunun kaynak işlenenleri b_1 buyruğunun sonuç işleneniyle aynı ise buna gerçek bağımlılık denir.

2. Karşıt bağımlılık(Okuma sonrası yazma): b_1 buyruğunun kaynak işlenenlerinden biri b_2 buyruğunun sonuç işleneni ise bu durum karşıt bağımlılık olarak adlandırılır.

3. Çıktı bağımlılığı(Yazma sonrası yazma): b_1 buyruğunun sonuç işleneni ile b_2 buyruğunun sonuç işleneni aynı ise bu duruma çıktı bağımlılığı denir.

Yukarıda bahsi geçen veri bağımlılıkları Şekil 2.3 ile gösterilmiştir. Okuma sonrası okumanın herhangi bir zararı olmadığını ve bağımlılık olarak geçmediğini belirtmek gerekmektedir.

Paralel olarak yürütülebilecek buyrukların kullanacakları yazmaçlar için birden fazla yerin ayrıldığı yapıya fiziksel yazmaç dosyası adı verilir. Fiziksel yazmaç dosyası bellek hiyerarşisinin en tepesinde bulunur. En hızlı, küçük ve

Şekil 2.3: Veri bağımlılıkları

en pahalı bellek tipidir. Hedef işlenen içeren bir buyruk işlemci tarafından çözüldüğünde, buyruğun sonucunu yazmak için fiziksel yazmaç dosyasında boş bir yazmaç ayrılır. İşlemci buyruğu yürütmeyi tamamladığında sonucunu ayrılmış olan bu yazmaça yazar. Ardından gelen buyrukların bahsi geçen yazmaçtaki veriye erişmeleri gerekir ise, bu buyrukların girdi işlenenlerine fiziksel yazmaç dosyasından erişilir. Yazmaçtaki veriye ihtiyacı olan bütün tüketici buyruklar emekli olduktan sonra yazmaç içın ayrılan satır özgür bırakılır. Fiziksel yazmaç dosyası, veri bütünlüğünün korunması gereken en kritik işlemci yapılarından biridir. Bu yapı mimarinin mevcut durumunu sakladığından ötürü hatalara karşı dayanıklı olmalıdır. Çağdaş işlemci tasarımlarında fiziksel yazmaç dosyasını veri yozlaşmasına karşı korumak için hata düzeltici kodlar kullanılmaktadır[4][5]. Güvenilirliği sağlarken başarımın çok fazla düşürülmemesi de tasarım açısından önem taşımaktadır.

Yazmaç yeniden adlandırılması yapılırken bir fiziksel yazmacın ömrü aşağıda belirtildiği gibi geçer:

1. Yazmaç atanması: Bir buyruk çözüldüğünde, boş bir fiziksel yazmaç o buyruk sonucunu yazabilsin diye ayrılır.
2. Geri yazma: İşletilen buyruğun sonucu ayrılmış olan yazmaça yazılır.
3. Bağımlı okumalar: Yazmaçtaki değere girdi olarak ihtiyacı olan müteakip buyruklar veriyi yazmaçtan okurlar.
4. Yazmaç bırakılması: Fiziksel yazmaça boru hattı içinde başka bir referans kalmadığında yazmaç özgür bırakılır.

Fiziksel yazmaçların hataya açık oldukları zaman geri yazmanın tamamlanması ile bağımlı okuyucuların sonuncusunun okumayı tamamlaması arasında geçen zamandır. Yazmacın değerini üreten buyruğun da işlenmiş olması gerekmektedir.

Buyruklar üç ayrı sınıfa ayrılırlar; yükleme/saklama , aritmetik ve dallanma. Bu üç sınıf içinden yükleme/saklama ve dallanma buyruklarının ürettikleri sonuçlar mimariye göre değişmekle birlikte sanal veya fiziksel adreslerden oluşmaktadırlar.

Bu çalışmada kullanılan Alpha mimarisi 24 farklı yükleme/saklama ve 17 farklı dallanma buyruğunun sonuçlarını sanal adres olarak hesaplamaktadır[6].

2.1.5 Bellek Sıralama

Yükleme buyrukları bellekteki bir adreste bulunan veriyi yazmaç dosyasına kopyalamak için kullanılır. Saklama buyrukları ise yüklemenin tam tersi olan yazmaç dosyasındaki veriyi bellekte belirtilen adrese kopyalar. Belleğe doğrudan erişim ve önbellek erişimi fiziksel yazmaçlara erişmeye oranla daha yavaş yapılabilmektedir. Belleğe erişim için gerekli olan adres üretimi de tek bir buyrukta yapılamamaktadır. Bu sebeplerden ötürü yükleme/saklama buyrukları işlemci içinde çalışan buyruklara oranla daha fazla saat vuruşunda tamamlanabilen buyruklardır. Bu işlemlerin gerçekleşme zamanlarının nispeten daha uzun sürmesi boru hattı için bir darboğaz oluşturmaktadır. Bu bölümde bu darboğazı aşmak için önerilmiş olan yöntemler anlatılmaktadır.

Aynı adres üzerinde yükleme/saklama yapması gereken buyruklar arasında da yazmaç yeniden adlandırmada yaşanan bağımlılık tipleri oluşabilmektedir. Bir yükleme buyruğu bellek üzerinde bir adresteki veriyi okuduktan sonra, bu adrese bir saklama buyruğu yeni bir veri yazabilir(okuma sonrası yazma) veya da bunun tam tersi bir durum oluşabilir(yazma sonrası okuma). Ayrıca iki ayrı saklama buyruğu bu adresteki veriyi yakın zamanlarda değiştirebilir(yazma sonrası yazma).

Bellek sıralama ve bellek veri akışı teknikleri ile ilgili olarak [35][15][41][30] referanslarından ayrıntılı bilgi alınabilir.

2.1.6 Program Sayacı

Program sayacı, çekme evresinde önbellekten çekilecek olan sıradaki buyruğun adresini tutmakla sorumlu yazmaça verilen isimdir. Bir koşullu dallanma buyruğu görene kadar program sayacı buyrukları sıralı bir şekilde tutar.

2.2 Mimari Hataya Açıklık Etkeni

Bir sistemin Mimari Hataya Açıklık Etkeni(MHAE), incelenen sayısal donanım yapısının geçici hatalardan etkilenmeye uygunluğunu ölçer.

Donanım yapılarının MHAE'lerini hesaplamak için Mimarisel Doğru Yürütme(MDDY) analizi yöntemi kullanılır. İncelenen yapıdaki bir bitteki değişikliğin programın hatalı çıktı vermesine sebep olması, o bitin MDDY bit olduğu anlamına gelir. Burada dikkat edilmesi gereken bir nokta, bir bitin MDDY biti olup olmadığının zamana göre değişebileceğidir. Mesela, geçersiz bir önbellek satırının veri blokunda bulunan bitlerin değişimi programın çıktısana hiçbir şekilde etki etmeyecekken, bu satır geçerli olduğunda veri blokunda oluşabilecek değişikliklerin hatalı program çıktısına sebep olma ihtimali bulunmaktadır.

Bir donanım yapısının MHAE'si aşağıda belirtildiği gibi tanımlanır:

$$\text{MHAE} = \frac{\Sigma \text{ MDDY bitlerinin etkin olduğu çevrimler}}{\text{Toplam Çevrim Sayısı} \times \text{Donanım Yapısındaki Toplam Bit Sayısı}}$$

Hatalar farkedilen ve farkedilmeyen olarak sınıflandırılır. Gizli veri yozlaşması

(GVY), farkedilmeyen hatalar için kullanılan bir terimken, farkedilen düzeltilemeyecek hatalar(FDH) belirlenen hatalar için kullanılır.

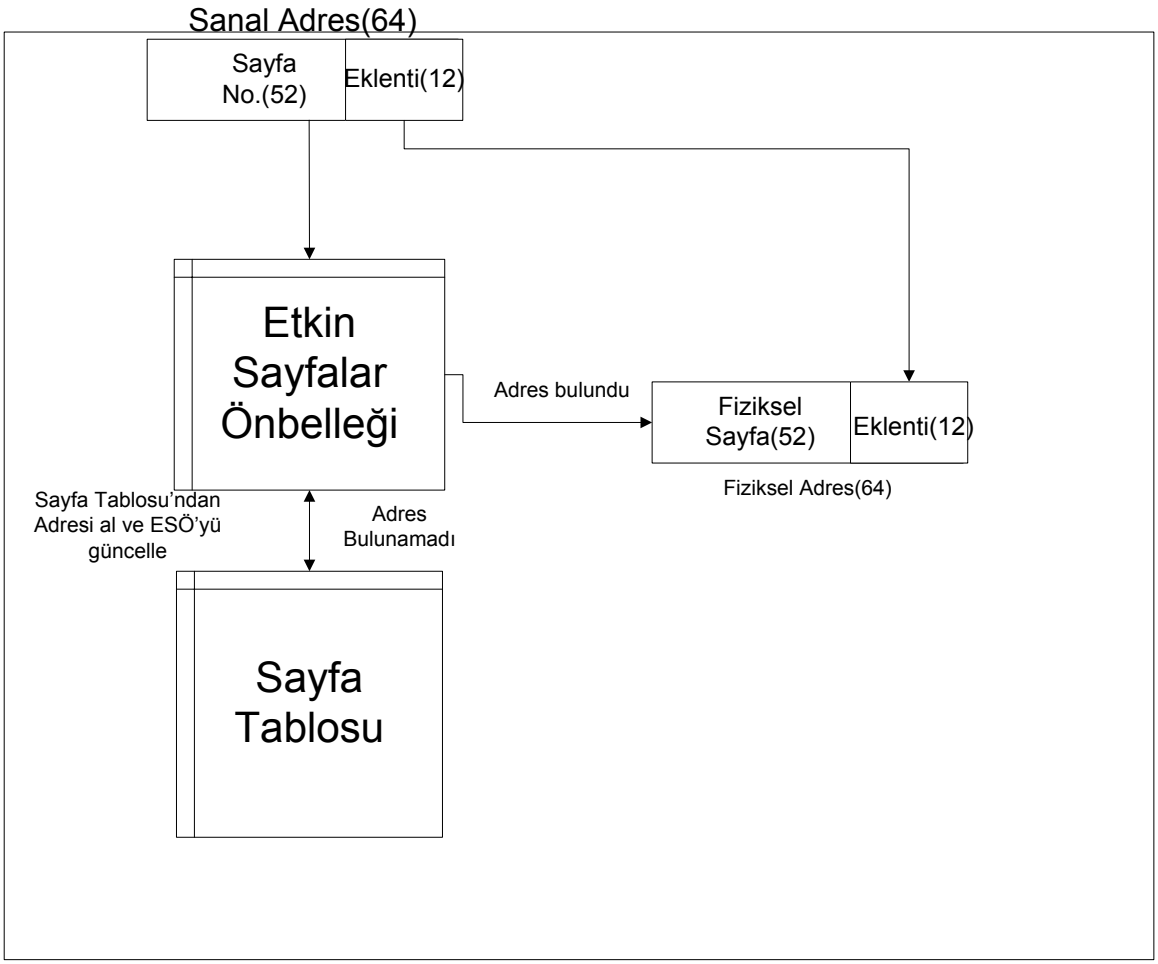
2.3 Sanal Adresleme

Sanal adresleme fikri belleğe sığamayacak kadar büyük programları desteklemek için ortaya atılmıştır. Ayrıca bir sanal adres uzayına bir diğer adres uzayından erişilmesini engelleyerek çok görevli işletimin güvenli bir biçimde yapılmasına yardımcı olur. Bu şekilde çalışan sistemlerde bir adres uzayında koşan programın bir başka adres uzayında koşan programın belleğine doğrudan erişimi yoktur.

Çağdaş sistemlerin sanal adreslemeyi gerçeklemek için kullandıkları yönteme sayfalama denir. Sayfa, sabit boyutlu bir bellek bloku olarak bağlayıcı tarafından atanan sanal sayfa numarası ile belirtilen bellek parçası olarak tanımlanır. Program kod ve veri bölütleri sayfaların içinde tutulur.

Bir program işletim sistemi tarafından çalıştırılacağı zaman, sayfaları fiziksel adres uzayına karşılık gelecek şekilde eşleştirilir. İşletim sistemi her bir program için bu eşleştirmeleri sayfa tablosu adında bir yapıda tutar. Sayfa tablosu sanal adresler ile fiziksel adresler arasındaki eşleşmeleri tutmak için uygun olmasına rağmen bellek üzerinde tutulması sebebiyle sanal adreslerin fiziksel karşılıklarının sorgulanması yavaş yürütülen bir süreçtir. Bu sebepten ötürü çağdaş işlemciler adres çeviri belleği(ESÖ) adında bir donanım yapısı barındırırlar. ESÖ adres eşleşmelerini tutmakla sorumlu ilişkili önbellektir. ESÖ'ler veri ve buyruk önbelleklerine benzer bir şekilde içerik adreslenebilir bellek(CAM) kullanarak gerçeklenmişlerdir. Bu şekilde eşleşmeler hızlıca sorgulanabilmektedirler.

Şekil 2.4 sanal adreslerin fiziksel adreslere çevirim sürecini göstermektedir. İşlemciden Bellek İşletme Birimi(MMU)'ne gelen bir bellek isteğinde istenilen çevirim ESÖ tarafından sağlanabiliyorsa buna ESÖ çarpması denir. Aranan çevirim ESÖ'de bulunamadıysa bu durum ESÖ ıskası olarak adlandırılır.



Şekil 2.4: Sanal adresten fiziksel adrese çevirim

ESÖ'nin önbelleklere göre konumu da önemli bir tasarım kararı oluşturmaktadır. Önbellekler fiziksel olarak dizinlenmiş sanal etiketli (PIVT), sanal olarak dizinlenmiş sanal etiketli(VIVT), veya sanal olarak dizinlenmiş fiziksel etiketli (VIPT) olabilirler.

Eşleştirilmemiş bir sayfaya bir buyruk tarafından erişilmeye çalışıldığında sayfa hatası meydana gelir. Sayfa hatasının olma sebebi sanal adresin bir fiziksel karşılığının henüz eşleştirilmemiş olması veya buyruğun erişmek istediği bellek alanına erişimi olmaması olabilir.

Çizelge 2.2: Hamming (7,4,3) kod kelimeleri

0000000	0100101	1000011	1100110
0001111	0101010	1001100	1101001
0010110	0110011	1010101	1110000
0011011	0111100	1011010	1111111

2.4 Doğrusal Blok Kodlar

Blok kodlar sabit boyutta mesajlara bölütlenmiş bilgi dizileridir. İki mesajın doğrusal kombinasyonu da doğrusal ise bu blok koda doğrusal blok kod denir. (n, k, d)'lik blok kod, k sembollük bir girdinin n boyutunda bir kod kelimesine gömüleceğini belirdir. Buradaki d ise iki kod kelimesi arasındaki asgari hamming mesafesini belirtir. k boyutunda bir bit dizisi kullanılarak 2^k farklı mesaj yaratılabilir. Kodlama süreci aşağıda belirtildiği gibidir:

1. $H \in \{0,1\}^{mxn}$ parite denetleme matrisi olsun, ve $m = n - k$ için $G \in \{0,1\}^{kxn}$ de C kodu için üreteç matrisi olsun.
2. Bir kodun sistematik olması için üreteç matrisinin $G = [I_k|P]$ formunda olması gerekir. Bu durumda parite denetleme matrisi $H = [P^T|I_k]$ olur.
3. Girdi mesajı $x = \{x_0, x_1, ..., x_{k-1}\}$ olan c kod kelimesi $c = mG$ olarak elde edilir.

Örnek olarak Hamming (7,4,3) kodlaması kullanılarak oluşturulabilecek kod kelimlere Çizelge 2.2'de gösterilmiştir. Görüldüğü üzere bütün kod kelimeleri arasındaki hamming mesafesi 3 veya 3'ün üzerindedir. Bu şekilde bu kod kullanılarak tek hata düzeltimi ve çift hataya kadar hata tespiti yapılabilmektedir.

3. İLİŞKİLİ ÇALIŞMALAR

Bu bölümde mikroişlemcileri geçici hatalardan korumak ve hataya müsaitliği ölçülebilir kılmak için yapılmış olan farklı çalışmalar anlatılmaktadır.

MHAE'ye alternatif olarak bir sistemin güvenilirliğinin ölçülebilmesi için [23] çalışmasında olasılık tabanlı bir kestirim modeli oluşturulmuştur.

[33] çalışmasında işlemcinin bellekten veri beklerken boşta kaldığı zamanlarda kullanılmayan işlem gücünün işletilmiş buyrukların bu bekleme esnasında tekrar işletilerek hata tespiti yapılmasını önermektedir.

Fiziksel yazmaç dosyasını korumak için farklı bir yöntem de [26] ile önerilmektedir. Çalışmada, veri saklayan fiziksel yazmaçların verilerinin kullanılmayan fiziksel yazmaçlara yedeklenerek hata tespit edildiğinde hataya düzeltmek için yedek kopyanın okunabileceği ve bu şekilde fiziksel yazmaç dosyasının korunabileceği anlatılmaktadır.

[34] çalışması ile sistemi hataya dayanıklı kılan mekanizmaların sadece yazılım üzerinde gerçeklenebileceği, hataya dayanıklı sistemlerin donanım yardımı olmadan, buyruk düzeyindeki paralellikten yararlanılarak sağlanabileceği gösterilmektedir.

[27] çalışmasında Alpha temelli bir işlemcinin yanında, aynı işlemcinin daha basitleştirilmiş bir türevi kullanılarak ana işlemcideki hatalar yardımcı işlemci

sayesinde tespit edilerek hata denetimi ve düzeltilmesi sağlanmaktadır.

[42] çalışması hata düzelten kodların sanallaştırılmasından yararlanılarak farklı kodlama biçimlerinin kullanılmasını ve hata düzelten kodların sadece hata oluşan durumlarda işletilmesini önermektededir.

İşlemci yazmaçlarında tutulan dar değerler üst bitleri 1 veya 0 dizisinden oluşan değerler olarak tanımlanmaktadır. Saklanan verilerin darlığından yararlanılarak başarım, güç tasarrufu ve güvenilirlik alanlarında olumlu etkileri olmuş çeşitli çalışmalar yapılmıştır[13][17] [24].

[20] çalışması fiziksel yazmaçta saklanan dar değerlerin aynı yazmaç üzerinde kopyalarının saklanması sayesinde fiziksel yazmaçın güvenilirliğinin arttırılabileceğini önermektedir.

Geçici hatalara karşı koruma sağlamak için, [16] çalışması dar değerleri tespit ederek, üzerinde önerilen farklı yöntemler ile başarımı etkilemeyecek bir şekilde işlemci içerisindeki yazmaçların güvenilirliğinin artırılmasının mümkün olduğunu belirtmektedir.

Literatürde derleyici mimari etkileşimini konu alan çalışmalar olmakla beraber, çalışmanın konusu olan sanal adreslerin kullanılarak güvenilirliğin arttırılmasına yönelik bir çalışmaya rastlanmamıştır .

[40] çalısması yazmaç dosyasının geçici hatalara karşı açıklığını ölçmek için yazmaç hataya açıklık etkenini geliştirmiştir ve sonrasında derleyiciyi kullanarak yazma ve okuma işlemlerinin konumlandırılmalarını değiştirerek kendi geliştirmiş olduğu hataya açıklık etkenindeki düşüş gösterilmiştir.

[3] ile MHAE'yi geliştirmenin sadece donanım üzerinde değil, derleyici kullanılarak da mümkün olduğu belirtilmiştir. Çalışmada MHAE kavramının başarım ile etkisini belirtilen yeni bir kavram geliştirilişmiş olup, derleyici bayraklarının kullanımlarına göre MHAE'ye etkileri gözlemlenmiştir.

[25] statik analiz yöntemiyle bir programdaki korunması önce likli olan program bölütlerini tespit ederek bunları kısmi olarak korunan bir belleğin korumalı kısmında saklamayı önermektedir. Bu şekilde bütün belleği ECC ile korumak yerine kritik program parçalarının koruduğu bir alan sağlanarak güçten ve alandan tasarruf edildiğini belirtmektedir.

[22] ise yazmaç dosyasının hataya açıklığını derleme zamanında kestirebilen bir statik analiz yöntemi önermekte ve derleyici en iyileştirmelerini kullanarak hatalara karşı koruma sağlanabildiğini belirtmektedir.

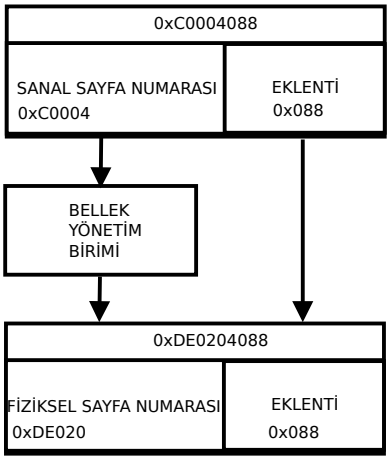
Uyuşmazlık ıskası, küme ilişkili veya doğrudan haritalı önbelleklerde farklı blokların aynı satırlara yerleştirilmesi sonucu oluşan bir durumdur. [32] çalışması çok çekirdekli sistemlerde oluşabilecek uyuşmazlık ıskalarının "derleyici yönetimli sayfa boyama" olarak isimlendirilmiş bir yöntemle engellenebileceğini göstermiştir. Bu yöntem derleyicinin erişim desenleri hakkında bilgi sahibi olmasından yararlanarak son seviye önbelleğin uygulamalar arasında daha düzgün şekilde paylaştırılmasını sağlamaktadır.

4. Önerilen Hata Düzeltme Yöntemi

Sanal adresler, sanal sayfa ve eklenti olmak üzere iki bölümden oluşmaktadırlar. Şekil 4.1 ile gösterilmiş olan fiziksel olarak dizinlenmiş sanal etiketli sanal adreslerde, adresin eklenti kısmı fiziksel adreslerde de sanal adreslerde de aynı yeri ifade eder. Fiziksel adres ile sanal adres arasındaki farklılık sadece sanal sayfa numarasını tutan kısımdadır. Bu tarz adreslemede etiket özgün bir şekilde veriyi tanımlayamadığından ötürü aynı sanal adresler farklı fiziksel adreslerle eşleştirilebilmektedir. Bunu engellemek için her bir adres satırına ayrıca adres uzayı belirteni eklenir. Bu şekilde yapılan adres isteklerinin hangi adres uzayından geldiği takip edilebilmekte ve farklı programların aynı sanal adrese sahip bölümlerinin birbirlerinin çevirimlerine karışması engellenebilmektedir.

Günümüz sistemleri ana belleği ve önbellekleri [21] [7] hata düzeltici kodlarla korumaktadırlar. Bellekler tamamen korumalı olsalar bile buralardaki veriler işlemci üzerindeki dizili yapıların kritik yol üzerinde olmaları sebebiyle bunlar üzerinde hata düzeltici kodların kullanılması başarımı olumsuz etkilemektedir. Bu sebepten ötürü parite denetlemesi gibi basit yöntemlerin kullanılması mümkünken daha karmaşık devreler gerektiren hata düzeltici kodlar tercih edilmemektedir [12][19].

Bir mikroişlemcinin üzerinde saklanan değerler adres değerleri ya da veri değerleri olarak ikiye ayrılabilir. Adres değerleri bir dallanma sonucunda gidilecek adresi, bellekten yüklenmesi gereken bir verinin hangi adresten yükleneceğini veya bellek üzerinde saklanması gereken bir verinin hangi bellek adresi üzerinde



Şekil 4.1: Fiziksel olarak dizinlenmiş sanal etiketli adres

Çizelge 4.1: Örnek program adreslemesi

Adres	Sembol
0000000000040060b	get_syndrome
0000000000400673	main
0000000000400b40	__libc_start_main
0000000000400df0	__libc_check_standard_fds
0000000000400e8e	check_one_fd.part.0
0000000000400f00	__libc_setup_tls
0000000000401130	_dl_tls_setup
00000000004011a0	__pthread_initialize_mini

saklanacağını tutarlar. Veri değerleri ise aritmetik işlem sonuçlarını tutarlar. Fiziksel olarak dizinlenmiş sanal etiketli önbellek kullanan sistemlerde program sayacında, fiziksel yazmaç dosyasında ve etkin sayfalar önbelleğinde sanal adres değerleri saklanmaktadır.

Program sayacı işlemcinin sıradaki buyruğu hangi adresten çekeceğini saklayan yapıdır. Program sayacının herhangi bir bitinde oluşacak bir hata program akışının değişmesine ve dolayısıyla programın hatalı bir şekilde sonlanmasına sebebiyet verecektir. Örneğin, Şekil 4.3'de SPARC tabanlı bir sistemin program sayacının bellekten bir sonraki getirmesi gereken adres olarak 0x403d2000 gözükmektedir. Bu adreste yazmaç üzerinde tutulan verinin 18 sayısı ile

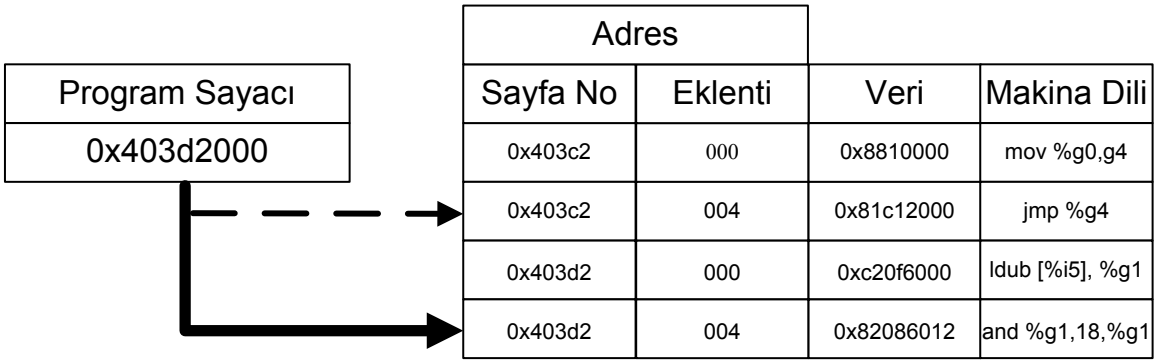
'mantıksal ve' işleminden geçirilerek sonucun aynı yazmaca yazılmasını belirten bir buyruk saklanmaktadır. Kesikli çizgi ise 0x403d2000 adresine hamming uzaklığı bir olan 0x403c2000 adresini göstermektedir. Bu adreste g4 yazmacında tutulan adrese dallanılmasını belirten bir buyruk bulunmaktadır. Program sayacının mevcut değerinin 17. bitinin 1 iken 0 değerine dönüşmesi program sayacının dallanma buyruğunu göstermesine sebep olacaktır. Bu dallanma buyruğunun hatalı bir şekilde bellekten getirilerek işlemci tarafından yürütülmesi program akışını bozacaktır ve program çıktısı hatalı olacaktır.

Fiziksel yazmaç dosyası yürütülen buyrukların sonuçlarının saklandığı donanım yapısıdır. Fiziksel yazmaç dosyası üzerinde sonuç olarak adres saklayan buyruklar yükleme/saklama buyrukları ve dallanma buyruklarıdır. Bu buyrukların sonuçlarının saklanmaları esnasında tutuldukları yazmacın sanal sayfa saklayan bit hücrelerinde meydana gelecek olan bir değişim belleğin yanlış adresinden veri çekilmesi, üretilen bir sonucun yanlış bellek adresine saklanması veya programın yanlış bir noktaya dallanmasıyla sonuçlanabilir. Bu durumlara ek olarak saklanan adres tanımsız bir adrese dönüşerek programın akışı geri dönülemeyen bir sayfa hatası ile son bulabilir. Sistem üzerinde işletilen buyrukların sınıflarına göre dağılımları Şekil 4.5 ile gösterilmiştir. Buradan da görüldüğü üzere işletilen bütün buyrukların %52'si bahsedilen durumdan etkilenmeye müsaitlerdir. Şekil 4.3 ile r0 yazmacındaki adrese koşulsuz dallanmak için kullanılan jmp buyruğunun değeri program sayacının bir sonraki saat vuruşunda göstereceği adresi saklamaktadır. Bu adres fiziksel yazmaç dosyasında tutulurken sayfa numarasının 2. biti 1'den 0'a değiştiği takdirde sonraki program sayacı yanlış adresten buyruk çekecek ve program akışı değiştiğinden ötürü program çıktısı hatalı olacaktır.

ESÖ ise sanal adreslerin fiziksel karşılıklarını sağlamakla sorumludur. ESÖ üzerinde etkin olan iki satırda saklanan sanal etiketlerin birbirleri arasındaki hamming mesafesi 1 olduğunda, bunlardan birinde oluşabilecek bir bitlik hata yanlış adres çevirimine yol açıp yanlış verinin çekilmesine sebep olabilecektir. Çizelge 4.2 ile üç adet basitleştirilmiş ESÖ satırı gösterilmektedir. Burada bütün satırların sanal sayfa numaralarının birbirlerine olan hamming uzaklığı 1'dir. Örneğin, Birinci satırın sanal sayfa numarasının 1. biti 1 olduğu zaman 2.

Çizelge 4.2: Örnek ESÖ satırları

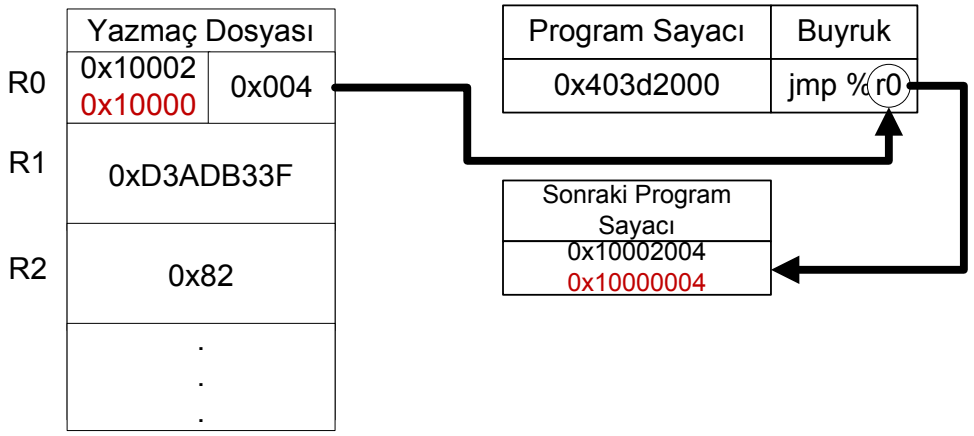
Sanal Sayfa Numarası Adres	Fiziksel Sayfa Numarası
0x400	0x180
0x401	0x243
0x402	0x854



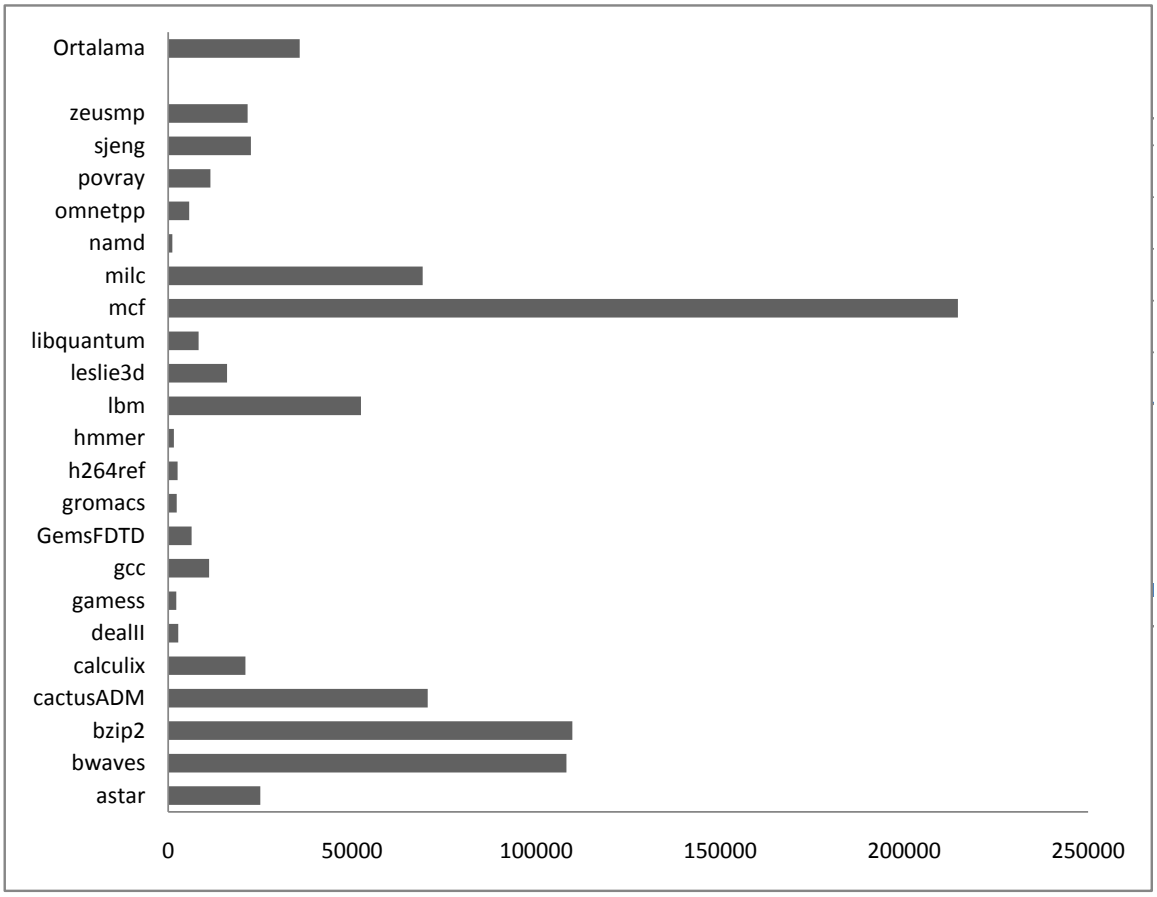
Şekil 4.2: Program sayacında bit hatası oluşması

satırla aynı sayfa numarası farklı adresleri belirtecektir veya 3. satırın sanal sayfa numarasının 2. biti 0 olduğu zaman 1. satırla birlikte aynı sanal sayfa numarasına sahip olup farklı adresi göstereceklerdir.

Kullanılabilecek sayfa numarası sayısı mimarinin kullandığı sayfa boyutuna göre değişiklik gösterir. Örneğin x86_64 mimarisi 4kB boyutunda sayfalar

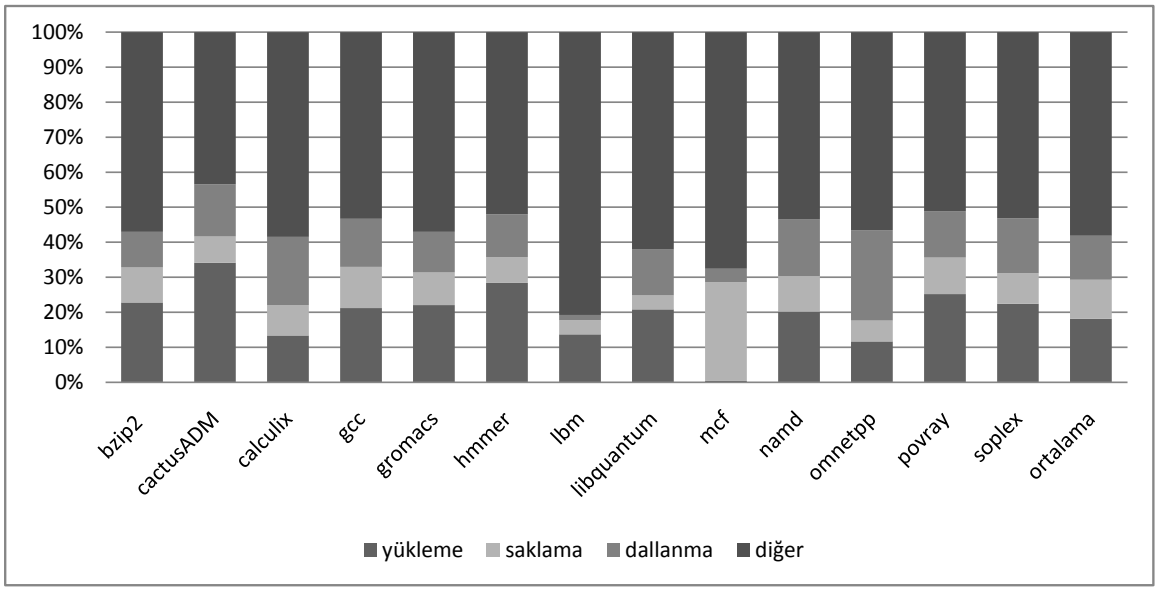


Şekil 4.3: Fiziksel yazmaçta bit hatası oluşması



Şekil 4.4: SPEC2006 ölçüm programlarının kullandıkları sayfa sayıları

kullandığı için 2^{52} sayfa tanımlamak mümkünken, Alpha mimarisinde 8kB boyutunda sayfalar kullanılması 2^{51} sayfa tanımlanmasını mümkün kılmaktadır. Önemli bir nokta mevcut masaüstü ve sunucu sistemlerinde koşan programların bu kadar sayfaya ihtiyaç duymamalarıdır. Örnek olarak SPEC2006 ölçüm programlarının kullandıkları sayfa sayıları Şekil 4.4 ile verilmiştir. Azami sayfa kullanımının 214611(2^{18}'den az) sayfa sayısıyla mcf ölçüm programına ait olduğu görülmektedir.

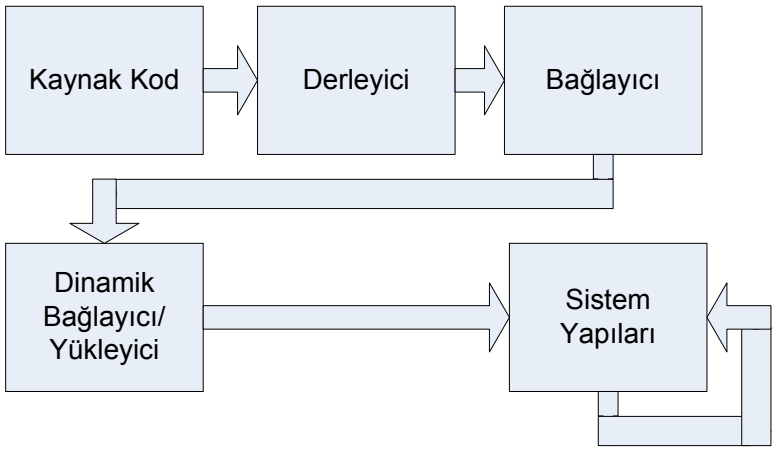


Şekil 4.5: Buyruk sınıflarının dağılımları

4.1 Bağlayıcı Üzerinde Gerçeklenmiş Döngüsel Kodlayıcı

Sanal adresleme destekleyen bir mikroişlemci üzerinde koşacak bir programın buyruk ve verilerinin saklanacakları sanal adreslerin bir şekilde atanması gerekmektedir. Bu süreç Şekil 4.6 ile gösterilmiştir. Günümüz işletim sistemlerinde programcı tarafından üst seviye bir programlama dilinde yazılmış olan bir programın adres atanması ortak kütüphane kullanılmadığı zamanlarda sadece bağlama zamanında gerçekleştirilir. Ortak kütüphane kullanılması durumunda işletim sistemi üzerindeki dinamik bağlayıcı/yükleyici ortak kütüphane ile ilgili sembolleri çözümler ve adreslemesini yapar. Derleyici farklı dosyalardan oluşan kodları makina diline çevirir. Bunun sonucu ortaya çıkan dosyalara nesne dosyaları adı verilir. Daha sonra bu nesne dosyaları bağlayıcı tarafından bir araya getirilir ve adres atamaları yapılır. Bağlayıcı adres atamalarını yaparken Bu süreç sonunda ortaya çıkan ikili dosya işlemci üzerinde çalışabilecek hale gelmiş olur.

Çizelge 4.1 ile 4kB boyutunda sayfalar kullanan, ortak kütüphane kullanmayan, Golay kodu üretmek için kullanılan örnek bir programın adreslemesinin bir



Şekil 4.6: Bir programın adreslenme yaşam döngüsü

parçası gösterilmiştir. 4kB sayfa kullanımı 16lık sistemde belirtilmiş adreslerin ilk 3 hanesinin eklentiden, adresin geri kalanının ise sayfa numarasından oluştuğunu belirtmektedir. Burada gösterilen program işlevleri iki ayrı sayfada tutulmaktadırlar. 0x400 numaralı sayfa get_syndrome, main, __libc_start_main, __libc_check_standard_fds, check_one_fd.part.0, __libc_setup_tls işlevlerine ait buyrukları tutmaktadır. _dl_tls_setup, __pthread_initialize_mini ve yer tasarrufu nedeniyle çizelge de gösterilmemiş işlevler ise 0x401 numaralı sayfada saklanmaktadır.

Bir önceki bölümdeki mevcut sistemlerin kullanabileceklerinden çok daha fazla sayfa adreslemesinin mümkün olduğu gözleminden yararlanarak sanal sayfa numaralarının döngüsel(51,43,3) kodlayıcı kullanılarak birbirleri arasında asgari hamming uzaklığı 3 olacak şekilde atandığı ve bu adreslerin tutulduğu noktalarda 1 bitlik hataların düzeltilebileceği bir yöntem bu bölümde önerilmektedir. Bu şekilde ESÖ'ler, yazmaç dosyası ve program sayacını hatalara karşı korumak mümkün olacaktır.

Döngüsel kodlar, c kod kelimesinin döngüsel kaydırılmasının da kod kelimesi oluşturduğu doğrusal blok kodlara denir. Gerçeklemesi diğer kodlara göre daha kolaydır. BCH kodu da bir çeşit döngüsel koddur. Bu kod Hamming kodunun çoklu bit hatalarını düzelten bir genellemesidir.

Çizelge 4.3: Kodlanmış adresler kullanan program

Adres	Sembol	Kodlanmış Adres
0x4001d8	__rela_iplt_start	0x200cc1d8
0x4002f8	_init	0x200cc2f8
0x4003e0	_start	0x200cc2f8
0x401130	_dl_tls_setup	0x2017d130
0x401690	exit	0x2017d690
0x401ae0	srand	0x2017dae0
0x4021d0	printf	0x2021f1d0
0x402450	__libc_message	0x2021f450
0x402840	__libc_fatal	0x2021f840

Bir sanal adres uzayı için sayfa numaralarını sıralı bir şekilde atamak yerine bağlayıcı üzerinde gerçeklenmiş bir döngüsel kodlayıcı kullanılarak sayfa numaraları atanabilir. Alpha mimarisinde sayfa eklentileri 13 bit, yani 8K boyutunda sayfalar, olduğu için $(51, 43, 3)$'lük bir döngüsel kodlayıcı bu iş için uygun düşmektedir. Alpha mimarisinde sayfa numaraları 43 veya 47 bit olarak ilklendirme zamanında ayarlanabilmektedir.

Bağlayıcı Çizelge 4.3 ile gösterilen doğrusal şekilde sayfa numarası atamak yerine, bu atadığı adresleri kodlayıcıdan geçirdikten sonra atayabilir. Bu şekilde bütün sayfa numaralarının birbirlerinden en az hamming uzaklığı üç ile kodlandığı garantilenmiş olur.

Sendrom çözme doğrusal blok kodların çözümünde yaygın olarak kullanılan bir yöntemdir. Sistemdeki sayfa numaraları doğrusal blok kodları ile kodlandığından ötürü sendrom çözücüyü sayfa hata kotarıcısının içinde gerçekleyerek geçici hatalar sonucu oluşan sayfa hatalarına karşı koruma sağlanabilir. Sendrom çözücüyü gerçeklemek için ilk önce çözme işleminde kullanılacak olan sendrom tablosunun oluşturulması gerekmektedir. Bu işlem için en düşük ağırlığa sahip olan $w = 0, 1^n$ kelimesi seçilir ve sendromu hesaplanır. Aynı sendroma sahip daha düşük ağırlıklı bir kelime bulunmazsa r tabloya koset lideri olarak eklenir.

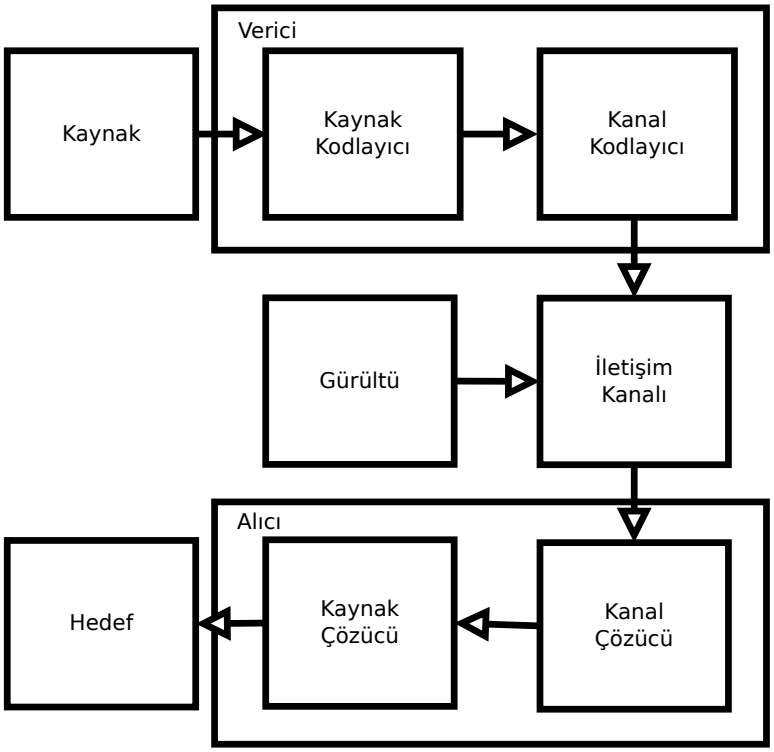
Sendrom $s = wH^T$ şeklinde hesaplanır.

Yukarıda bahsedilen çözme yöntemi, daha önceden bahsedilmiş olan kodlama yöntemi ile birlikte kullanılarak ESÖ, fiziksel yazmaç dosyası ve program sayacında saklanan sanal adreslerin sanal sayfa numaralarında oluşan hataların sayfa sorgusu sırasında sayfa hata kotarıcısında düzeltilebilmesine olanak sağlamaktadır.

4.2 Önerilen Yöntemin İletişim Sistemlerine Benzerliği

Bir iletişim sisteminde veri üzerinde kaynaktan hedefe ulaşana kadar izlenen yol Şekil 4.7 ile gösterilmiştir. İletilmek istenilen sinyal öncelikle kaynak kodlaması kullanılarak sıkıştırılır ve verimli bir şekilde iletilebilecek hale getirilir. Daha sonra kanal kodlaması kullanılarak bu sinyale fazladan birkaç bit eklenir ve bu fazladan bitler sayesinde sinyalin oluşabilecek hatalara karşı gürbüzlüğü sağlanır. Kanal kodlanması da tamamlanmış olan sinyal daha sonra verici tarafından kullanılan iletişim kanalına iletilir. Örnek olarak bu iletişim kanalı okyanus altındaki fiber optik bir kablo veya kablosuz iletişimde olduğu gibi hava olabilir. İletişim ortamındaki çevresel gürültüden ötürü iletilmekte olan sinyalde hatalar oluşabilir.napim kuzen ben de burokratik biseyler bek Bu sinyal alıcı tarafına ulaştığında alıcıda bulunan kanal çözücüsü bu hataları düzelterek kaynak çözücüsüne iletir. Kaynak çözücüsü sıkıştırılmış sinyali açtıktan sonra sinyal hedefe ulaşmış olur.

Bu tez çalışmasında önerilen süreç iletişim sistemlerinde kullanılmakta olan verinin kaynak tarafından gönderilirken hamming kodlanması ve alıcı tarafından çözülmesi sürecine büyük bir benzerlik göstermektedir ve aşağıda maddeler halinde açıklanmıştır. Terimlerin benzerliği açısından kaynak kodlama kavramının sinyalin temsil biçiminin değiştirilmesi olduğu, kaynak kodun ise



Şekil 4.7: İletilecek bilginin kodlanması ve çözülmesi

programı belirten ifadelerden oluşan yazılım parçası olduğuna dikkat edilmelidir.

1. Kaynak(Program kaynak kodu): Üst seviye bir programlama dilinde yazılmış program kaynak kodu iletilecek olan veriyi temsil eder.

2. Kodlayıcı: Üst seviye programın derleyici vasıtasıyla makina koduna çevrilmesi işlemi kaynak sinyalinin temsilinin değiştirilmesi olarak düşünülebilir. Bu noktada, yürütülmek istenilen algoritma üst düzey bir programlama dilinden işlemcinin çalıştırılabileceği ikili dilde bir temsile dönüştürülür. Makina diline çevrilmiş olan program bu noktada adresler ve bu adreslerde bulunan buyruklardan oluşmaktadır. Bu tez ile önerilmekte olan yöntem kanal kodlaması kullanarak programdaki buyrukları saklayan adreslerin sayfa numaralarının birbirlerinden hamming mesafesi üç uzaklığında olacak şekilde konumlandırılmalarıyla bu adreslerin saklandıkları yapılarda oluşabilecek bir bit hatasının programın akışında bir soruna yol açmamasını sağlamaktadır.

3. İletişim kanalı: Programın işleyişi sırasında sanal adreslerin saklandığı donanım yapılarında kanal kodlaması kullanılarak üretilmesi sayesinde hatalara karşı dayanıklı olmaları sağlanır. Örneğin program sayacında tutulan bir sanal adresin sayfa numarasında bir bit hata oluşması adreslerin sıralı bir şekilde saklanmaları durumunda yanlış bir program akışına sebep olacakken önerilen şekilde sayfa numaralandırılması yapıldığı için sistem oluşacak olan bir bitlik hatayı telafi edebilecektir.

4. Çözücü: Yapılacak olan her bir sanal adresten fiziksel adres tercümesi sistemde ilk olarak ESÖ üzerinde sorgulanır. Bu sorguyu yapan adres saklayan donanım birimi üzerinde bir bitlik hata oluştuğu durumlarda bütün adresler hamming mesafesi üç ile ayrıldığı için sorgulanan adres ESÖ üzerinde bulunamıyacaktır. ESÖ üzerinde bulunamayan adres sorgusu sayfa hatasına yol açar ve işletim sistemi tarafından saklanmakta olan sayfa tablosuna iletilir. Sayfa hata kotarıcısı üzerinde gerçeklenmiş olan kanal çözücüsü adres saklayan birimin ilettiği adresteki bir bitlik hatayı düzeltir ve düzeltilmiş haliyle ESÖ satırını günceller. Bu şekilde programın akışı oluşabilecek muhtemel bir sorundan arınmış bir şekilde devam eder.

5. Hedef: Kaynak tarafından gönderilen buyruk silsilesinin aradaki duraklarda oluşabilecek hatalara karşı alınan önlemler sayesinde MDDY'yi bozmadan işletilebilir halde sunulması sağlanılmış olur.

Yukarıda belirtilen süreçten de çıkarılabileceği üzere sanal adres saklayan bir donanım yapısında her hata oluştuğunda, sanal adresten fiziksel adrese yapılan sorguların nihai hedefi sayfa hata kotarıcısı olmaktadır. Bu tez ile incelenmiş olan donanım yapılarında oluşabilecek hata sürecleri aşağıda açıklanmıştır.

4.3 Donanım Üzerinde Gerçeklenmiş Sendrom Çözme

Tezde incelenen donanım yapıları için sendrom çözme için kullanılan standart hata düzeltici kod devreleri gerçeklenebilir. Buradaki durumun kullanılan ECC devrelerinden farkı kodlama kısmı bağlama zamanında yapıldığı için donanım

Çizelge 4.4: Yazma erişimlerinin toplam erişimlere oranı

Ölçüm programı	Oran	Ölçüm programı	Oran
bzip2	0,35	cactusADM	0,37
calculix	0,36	gcc	0,36
gromacs	0,64	hmmer	0,37
lbm	0,17	libquantum	0,35
mcf	0,12	namd	0,36
omnetpp	0,34	povray	0,32
soplex	0,35	ortalama	0,36

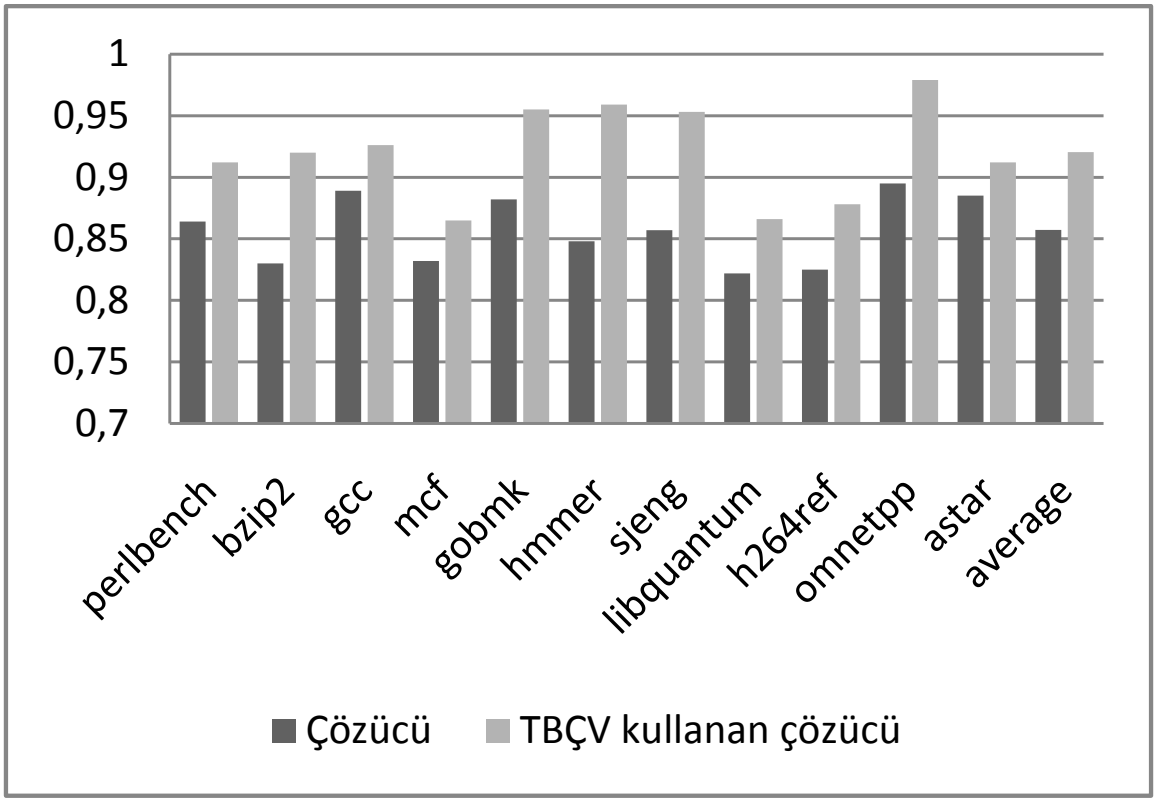
üzerinde kodlayıcı gerçeklemeye gerek olmayışıdır. Sadece yazmaç okumalarında ECC devreleri çalışacağından ötürü standart ECC yapısına göre güç tasarrufu sağlanabilecektir. Bu şekilde sayfa hata kotarıcısında gerçeklenmiş çözücünün ortaya çıkardığı başarım kaybı da engellenebilir. Fiziksel yazmaç dosyası için yazma erişimlerinin toplam erişimlere oranı Çizelge 4.4 ile gösterilmiştir.

5. DENEYSEL SONUÇLAR

Önerilen hata bulma ve düzeltme yönteminin başarımını ölçmek için Linux 3.0 çekirdeği sayfa hata kotarıcısı içinde gerçeklenmiş olan sendrom çözücü algoritması SPEC2006 ölçüm programlarıyla çalıştırılmıştır. Sendrom çözücünün başarıma etkisi Şekil 5.1 ile verilmiştir. Hata kotarıcısına eklenen blok çözücü %13 başarım kaybı yaratmıştır. Intel mimarisinde bulunan TBÇV buyrukları veri seviyesinde paralellikten yararlanarak aynı veri üzerinde paralel işlem yapılmasına olanak sağlayan buyruk kümesidir. Aynı çözücü TBÇV buyruk kümesindeki bir veri üzerinde 1 olan bitleri sayma buyruğu olan "popcnt" buyruğu kullanılarak gerçeklendiğinde başarım kaybı %8 olmuştur. Linux sayfa hata kotarıcısının içine gömülmüş olan çözücü kodları, TBÇV kullanan ve kullanmayan halleri ile birlikte Ekler kısmında verilmiştir.

Başarımda yaşanan düşüşe karşın, çalışmada kullanılan yöntem sayesinde buyruk ve veri ESÖ'lerinin MHAE'lerinde Şekil 5.2 ve 5.3 ile görüldüğü gibi kayda değer bir azalma sağlanmıştır. Önerilen yöntem sanal adreslerin sayfa numaraları kısmını koruduğu için bu azalma sağlanmıştır.

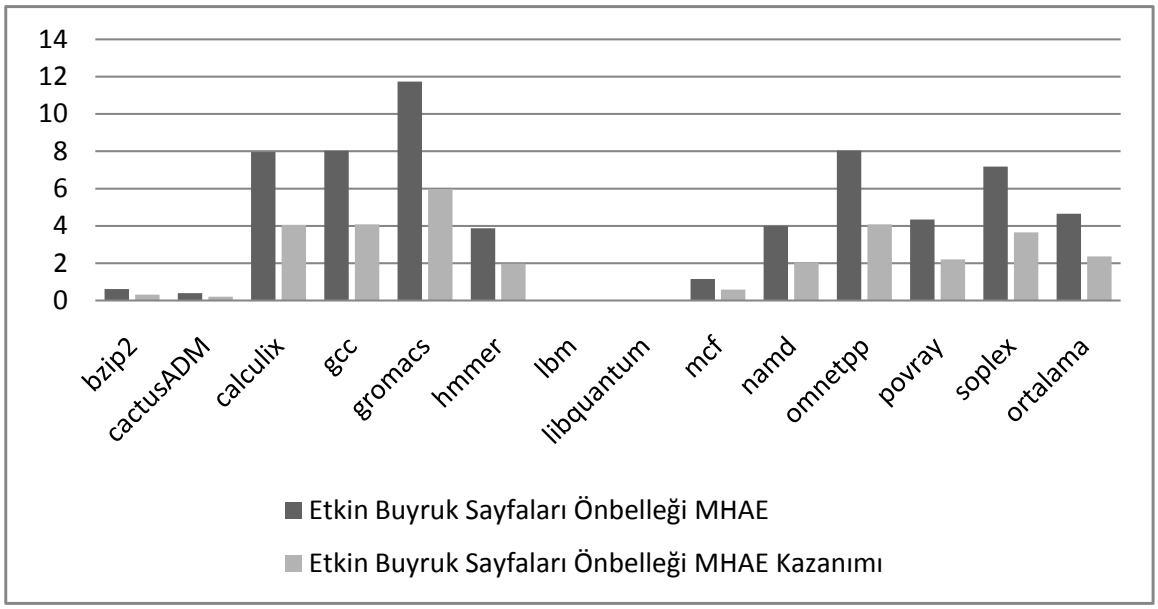
Alpha 21264 için bir ESÖ satırı sanal etiketler, fiziksel sayfa numarası, okuma/yazma izin bitleri, adres uzayı belirteni, okuma yazma için hata bitleri ve geçerli bitinden oluşmaktadır. ESÖ üzerinde sanal etiket yüzünden hata oluşabilmesinin tek yolu iki ayrı satırın geçerli oldukları durumda birbirlerine olan hamming uzaklıklarının 1 olması ve bu iki satırdan birinin sanal etiketinin bir geçici hata sebebiyle tek bitinin değişmesi sonucu aynı adresin farklı çevirilere sahip olması durumudur.



Şekil 5.1: Sayfa hata kotarıcısında gerçeklenmiş sendrom çözücünün başarıma etkisi

ESÖ üzerinde sanal adres sayfa numarası saklayan bit hücrelerinde oluşabilecek bir bit hatası hamming distance bir uzaklığında başka bir adres olmayacağı kodlama evresinde garanti edildiğinden ötürü sadece bir sayfa hatasına sebep olacaktır. ESÖ üzerinde sayfa numaraları birbirinden hamming uzaklığı bir ile ayrılmış farklı adresler bulunamayacağından dolayı oluşabilecek tek bitlik hatalara karşı koruma sağlanmış olur.

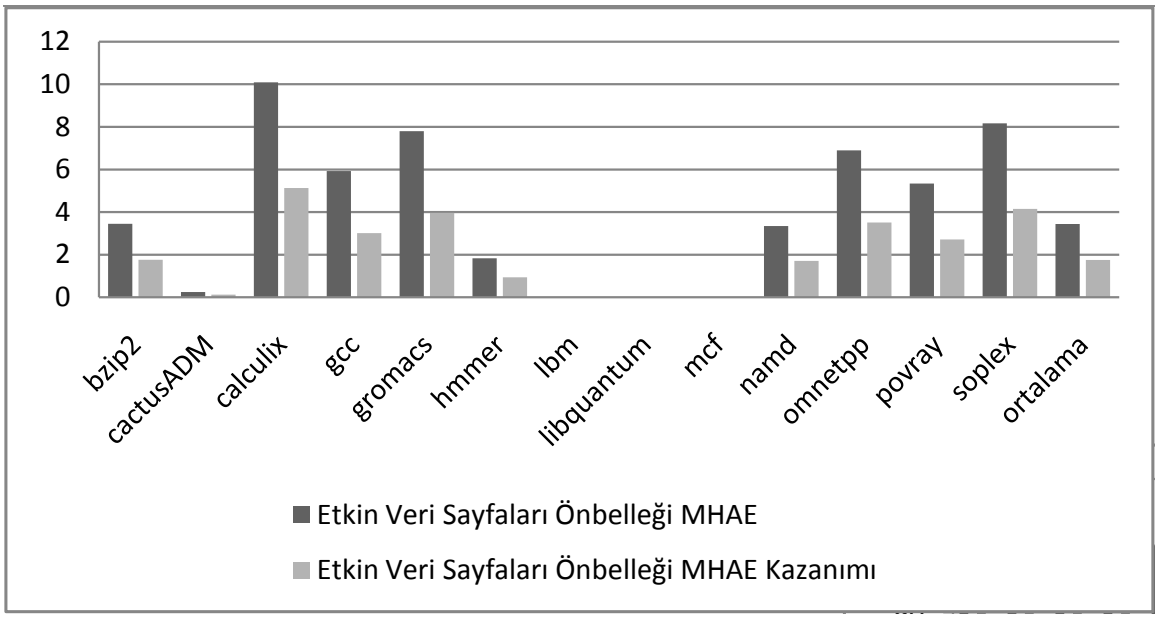
Bir fiziksel yazmaçın hataya açık olduğu an geri yazma ile tüketicilerin sonuncusunun okumasını bitirmesi aralığındadır. Alpha mimarisinde, yükleme/saklama ve dallanma buyrukları sonuçlarını sanal adre olarak üretmektedirler. Kullandığımız yöntem fiziksel yazmaç dosyasını hataya açık olduğu zamanlarda koruyabilmektedir. Fiziksel yazmaç dosyasında ortaya çıkan MHAE'deki azalma Şekil **??** ile gösterilmişir. Kullanılan üç ayrı buyruk sınıfından dallanma ve yükleme/saklama



Şekil 5.2: EBSÖ için MHAE'deki azalma

buyruklarının ürettikleri sonuçlar sanal adres olarak fiziksel yazmaçlarda saklanmaktadırlar. Bu sonuçlar bağımlı okuyucu buyrukların sonuncusunun saklanan değeri okumasına kadar atanmış olan yazmaçta tutulurlar. Saklanan sonuçlar kullanılacakları zaman ESÖ üzerinden sorgulanarak fiziksel adres karşılıkları öğrenilir. Bu duruma bağlı olarak sanal adres saklayan yazmaçlarda oluşacak bit hataları sayfa hata kotarıcısında adres karşılığının sorgulanmasına sebep olur. Bu durumda da aynı program sayacında olduğu gibi sayfa hata kotarıcısı hatayı düzelterek doğru

Program sayacında oluşabilecek bir geçici hata, boruhattının boşaltılmakta olmadığı bütün durumlar için hatalı sonuç üretecektir. Önerilen yöntem Şekil 5.5 program sayacının sanal sayfa numaraları tutulan kısmını koruyabilmektedir. Program sayacı üzerinde oluşan bir bitlik hata, program sayacında tutulan sanal adresin tercüme edilmesi isteği ile sonuçlanacaktır. Örneğin program sayacında 0x10000000 olarak saklanmakta olan bir adres 0x10080000 olarak değişebilir. Tezde önerilen yöntem doğrultusunda adresler hamming uzaklığı üç olacak şekilde kodlandıklarından, 0x10080000 adresi için sorgu sayfa hata kotarıcısına ulaştığında, sayfa hata kotarıcısı bu sorgunun aslında 0x10000000 adresine olduğunu farkederek sanal adresin doğru fiziksel karşılığını sağlayabilir. Bu şekilde program sayacı üzerinde oluşmuş olan bir bitlik hata savuşturulmuş



Şekil 5.3: EVSÖ için MHAE'deki azalma

olur.

5.1 Deney Ortamı

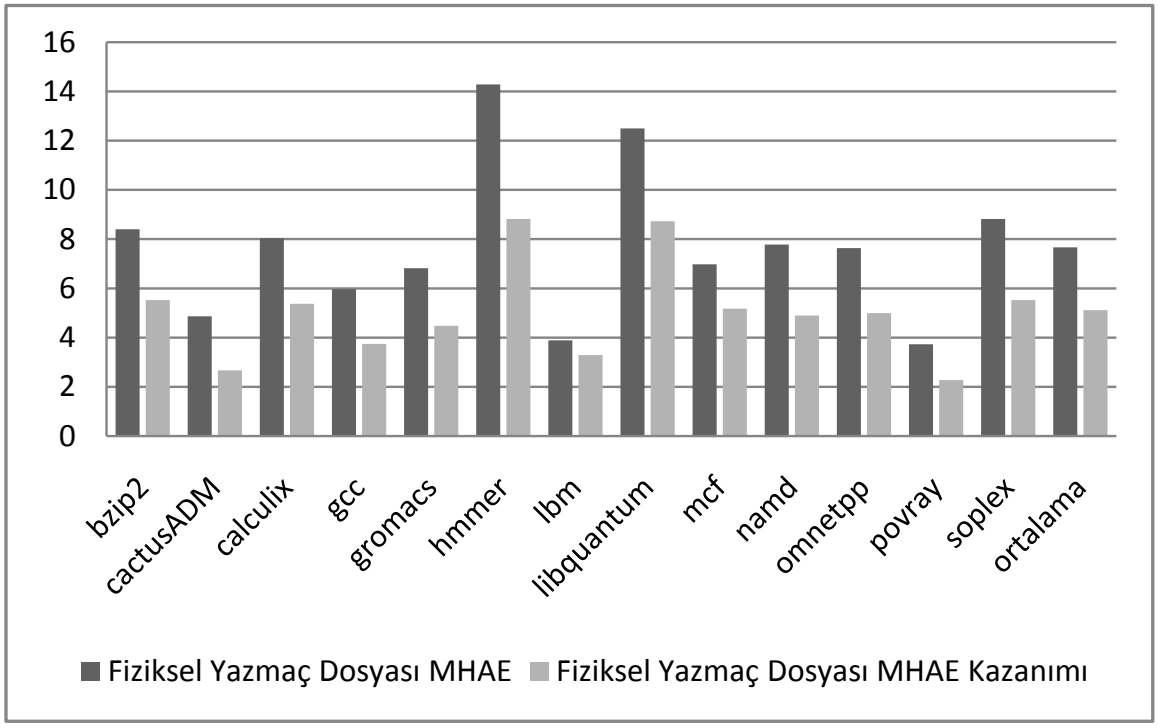
Çalışmada yapılan bütün MHAE ölçümleri gem5[8] benzetim ortamı üzerinde SPEC2006 ölçüm programlarını koşturarark yapılmıştır. Benzetim ortamı değişkenleri Tablo 5.2 ile gösterilmiştir. Sunulan sonuçlar 250 milyon buyrukluk ısınma zamanının ardından gem5'in 250 milyon buyruk koşturulmasıyla elde edilmiştir. Bütün ölçüm programlarının gem5 benzetim ortamı üzerinde sağlıklı bir şekilde çalıştırılması sağlanamadığından ötürü bazı ölçüm programlarıyla ölçüm alınamamıştır. Çalışmada kullanılmış olan SPEC2006 ölçüm programları ve açıklamaları Tablo 5.1 ile gösterilmiştir.

Ölçüm Programı	Dil	Uygulama Alanı	Açıklama

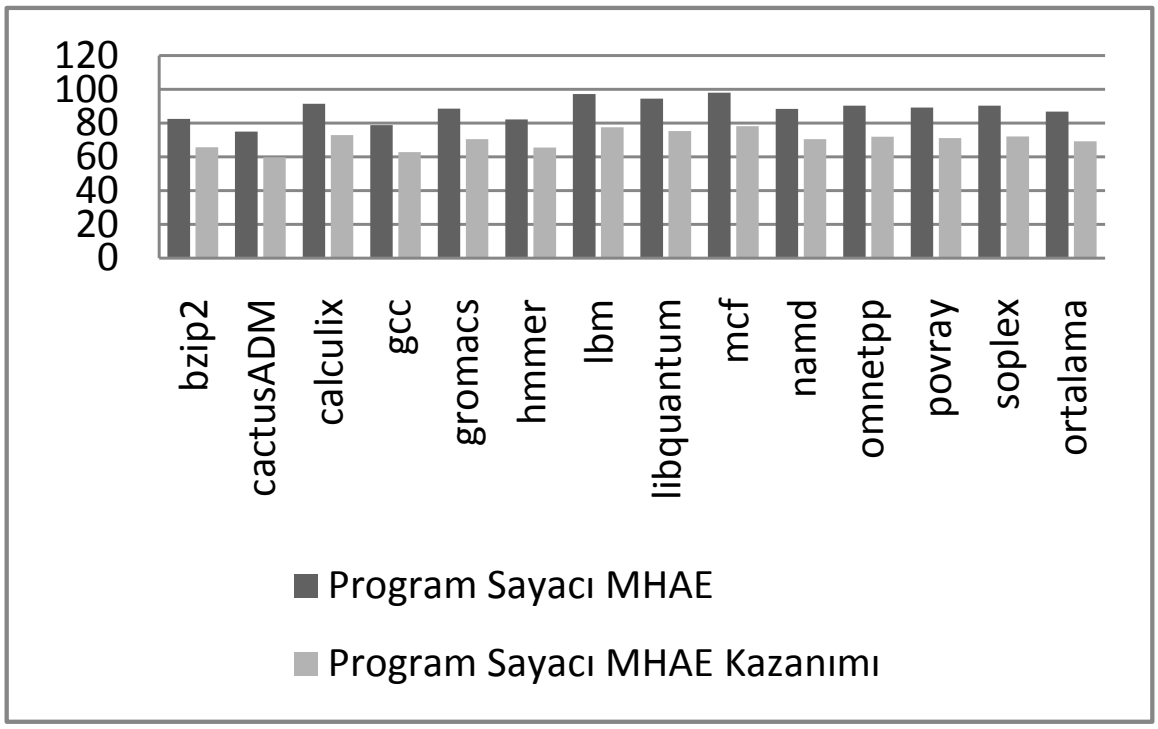
perlbench	C	Programlama dili	Perl 5.8.7'den türetilmiştir. İçinde SpamAssassin, MHonArc (e-posta düzenleyicisi), and specdiff (Ölçüm programları çıktılarını denetleyen SPEC aracı)programlarını barındırır.
bzip2	C	Sıkıştırma	Julian Seward'ın 1.0.3 sürümlü bellek-yoğun işlem yapan bzip2 kodu.
gcc	C	C derleyicisi	Opteron mimarisi için kod üreten gcc 3.2 tabanlı derleyici
mcf	C	Kombinatoryel Eniyileştirme	Aynı zamanda ticari ürünlerdede kullanılan bir ağ simpleks algoritması kullanarak araç takvimlesi yapar.
gobmk	C	Yapay Us	Go isimli kuralları basit ama çok karmaşık bir oyunu oynar.
hmmer	C	Gen dizisi araması	Profil gizli Markov modelleri kullanarak protein dizileri analizi
sjeng	C	Yapay Us	Farklı tarzlarda satranç oynayabilen üst düzey bir satranç programı

libquantum	C	Fizik / Kuvantum hesaplama	Shor'un polinom zaman faktorizasyon algoritmasını koşan bir kuvantum bilgisayarın benzetimini yapar
h264ref	C	Video Sıkıştırma	H.264/AVC kodlayıcısının referans gerçeklemesi.
omnetpp	C++	Ayrık olay benzetimi	OMNet++ ayrık olay benzetimini kullanarak büyük bir Ethernet ağının benzetimini yapar
astar	C++	Yol bulma algoritmaları	2 Boyutlu haritalar için yol bulma algoritmaları kütüphanesi. Bilindik A* algoritmasını da içerir.

Çizelge 5.1: Ölçüm programları açıklamaları



Şekil 5.4: Fiziksel yazmaç dosyası için MHAE'deki azalma



Şekil 5.5: Program sayacı için MHAE'deki azalma

Çizelge 5.2: Benzetim ortamı değişkenleri

Parametre	Yapılandırma
İşlemci	Alpha 21264
Boru Hattı Genişliği	8 çek, çöz, yürüt, yaz
Pencere Boyutu	32 satır yükleme/saklama kuyruğu, 192 satır yeniden sıralama belleği, 256 satır yazmaç dosyası
ESÖ	64 satır veri, 48 satır buyruk
L1 Buyruk Önbelleği	32KB, 2 yollu kümeli ilişkili, 64 bayt satır, 1 saat vuruşu isabet gecikmesi
L1 Veri Önbelleği	64 KB, 2 yollu kümeli ilişkili, 64 bayt satır, 1 saat vuruşu isabet gecikmesi
L2 Birleşik Önbellek	256 KB, 16 yollu kümeli ilişkili, 64 bayt satır, 10 saat vuruşu isabet gecikmesi

6. SONUÇ

Bu tez çalışmasında fiziksel olarak dizinlenmiş sanal etiketli sistemlerde sanal adres saklanan sistem yapıları olan etkin sayfalar önbelleği, fiziksel yazmaç dosyası ve program sayacı üzerinde oluşması muhtemel bit hatalarına karşı; sanal adres oluşturma zamanında kullanılan kodlama yöntemi ve sayfa hata kotarıcısı üzerinde gerçeklenmiş çözücüsünün birlikte kullanımı ile koruma sağlanmıştır. Önerilmiş olan yöntem sayesinde bahsi geçen donanım yapıları MHAE'lerinde etkin veri sayfaları önbelleği için 42.5%, etkin buyruk sayfaları önbelleği için 40.3%, program sayacı için 69.2% ve fiziksel yazmaç dosyası için 33.3% düzeyinde bir iyileştirme sağlanmıştır. Bu çalışmaya ek olarak ileride farklı kodlama ve çözme yöntemleri kullanılarak, veya donanım yardımı ile başarımda oluşan düşüşün önüne geçilebilir ve yöntemin sistem başarımına etkisi göz ardı edilebilecek şekilde hatalara karşı koruma sağlanabilir.

KAYNAKLAR

[1] In *Semiconductors Industry Association (SIA), International Technology Roadmap for Semiconductors 2005, http://www.itrs.net/Links/2005ITRS/Home2005.htm.*

[2] In *S. Mukherjee, "Architecture Design for Soft Errors", (Boston,Morgan-Kafumann),2008.*

[3] In *T. M. Jones, M. F. P. O'Boyle, and O. Ergin, "Evaluating the effect of compiler optimizations on MHAE," in Workshop on Interaction Between Compilers and Computer Architecture (INTERACT-12), 2008.*

[4] In *Kahle, J.; Sinharoy, B.; Starke, W.; Taylor, S.; Weitzel, S.; Chu, S.G.; Islam, S.; Zyuban, V.; , "The implementation of POWER7TM: A highly parallel and scalable multi-core high-end server processor," Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International , vol., no., pp.102-103, 7-11 Feb. 2010.*

[5] In *KNaffziger, S.D.; Hammond, G.; , "The implementation of the next- generation 64 b ItaniumTM microprocessor," Solid-State Circuits Conference, 2002. Digest of Technical Papers. ISSCC. 2002 IEEE International , vol.1, no., pp.344-472 vol.1, 2002.*

[6] In *"Alpha Architecture Reference Manual, Fourth Edition", Compaq, 2002.*

[7] In *"AMD eighth generation processor architecture. AMD white paper, Oct 2001.".*

[8] In *"N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell,*

M. Shoaib, N. Vaish, M. D.Hill, and D. A. Wood. The gem5 simulator. SIGARCHComput. Archit. News, 39:1–7, Aug. 2011.

[9] In A. Gordon, editor, *Programming Languages and Systems*, volume 6012 of *Lecture Notes in Computer Science*, pages 165–184. Springer Berlin Heidelberg, 2010.

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

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

[12] F. Bower, P. Shealy, S. Ozev, and D. Sorin. Tolerating hard faults in microprocessor array structures. In *Dependable Systems and Networks, 2004 International Conference on*, pages 51 – 60, june-1 july 2004.

[13] D. Brooks and M. Martonosi. Dynamically exploiting narrow width operands to improve processor power and performance. In *Proceedings of the 5th International Symposium on High-Performance Computing*, page 13. IEEE Computer Society, 1999.

[14] I. Burcea and A. Moshovos. Phantom-btb: a virtualized branch target buffer design. In *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems*, ASPLOS '09, pages 313–324, New York, NY, USA, 2009. ACM.

[15] H. Cain and M. Lipasti. Memory ordering: a value-based approach. In *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pages 90 – 101, june 2004.

[16] O. Ergin, O. Unsal, X. Vera, and A. Gonzales. Exploiting narrow values for soft error tolerance. *IEEE Computer Architecture Letters*, 5(2), 2006.

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

[18] J. F. Z. et al. Ibm experiments in soft fails in computer electronics(1978-1994). *IBM Journal of Research and Development*, 40(1), 1996.

[19] J. Friedrich, B. McCredie, N. James, B. Huott, B. Curran, E. Fluhr, G. Mittal, E. Chan, Y. Chan, D. Plass, S. Chu, H. Le, L. Clark, J. Ripley, S. Taylor, J. Dilullo, and M. Lanzerotti. Design of the power6 microprocessor. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pages 96 –97, feb. 2007.

[20] J. Hu, S. Wang, and S. G. Ziavras. In-register duplication: Exploiting narrow-width value for improving register file reliability. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 281–290. IEEE Computer Society, 2006.

[21] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe. Multi-bit error tolerant caches using two-dimensional error coding. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 40, pages 197–209, Washington, DC, USA, 2007. IEEE Computer Society.

[22] J. Lee and A. Shrivastava. Static analysis to mitigate soft errors in register files. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 1367–1372, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

[23] X. Li, S. Adve, P. Bose, and J. Rivers. Softarch: an architecture-level tool for modeling and analyzing soft errors. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 496 – 505, june-1 july 2005.

[24] M. H. Lipasti, B. R. Mestan, and E. Gunadi. Physical register inlining. In *Proceedings of the 31st annual international symposium on Computer architecture*, ISCA '04, pages 325–, Washington, DC, USA, 2004. IEEE Computer Society.

[25] M. Mehrara and T. Austin. Exploiting selective placement for low-cost memory protection. *ACM Trans. Archit. Code Optim.*, 5(3):14:1–14:24, Dec. 2008.

[26] G. Memik, M. Kandemir, and O. Ozturk. Increasing register file immunity to transient errors. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 586 – 591 Vol. 1, march 2005.

[27] F. Mesa-Martinez and J. Renau. Effective optimistic-checker tandem core design through architectural pruning. In *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, pages 236 –248, dec. 2007.

[28] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 29 – 40, dec. 2003.

[29] K. Olukotun and L. Hammond. The future of microprocessors. *Queue*, 3(7):26–29, Sept. 2005.

[30] S. Onder and R. Gupta. Dynamic memory disambiguation in the presence of out-of-order store issuing. In *Microarchitecture, 1999. MICRO-32. Proceedings. 32nd Annual International Symposium on*, pages 170 –176, 1999.

[31] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design, Fourth Edition, Fourth Edition: The Hardware/Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design)*. Morgan Kaufmann, 2008.

[32] S. H. Pugsley, J. B. Spjut, D. W. Nellans, and R. Balasubramonian. Swel: hardware cache coherence protocols to map shared data onto shared caches. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, PACT '10, pages 465–476, New York, NY, USA, 2010. ACM.

[33] M. Qureshi, O. Mutlu, and Y. Patt. Microarchitecture-based introspection: a technique for transient-fault tolerance in microprocessors. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 434 – 443, june-1 july 2005.

[34] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August. Swift: Software implemented fault tolerance. In *Proceedings of the international symposium on Code generation and optimization*, CGO '05, pages 243–254, Washington, DC, USA, 2005. IEEE Computer Society.

[35] J. Shen and M. Lipasti. *Modern Processor Design: Fundamentals of Superscalar Processors*. McGraw-Hill, 2005.

[36] J. E. Smith. A study of branch prediction strategies. In *25 years of the international symposia on Computer architecture (selected papers)*, ISCA '98, pages 202–215, New York, NY, USA, 1998. ACM.

[37] V. Sridharan and D. Kaeli. Eliminating microarchitectural dependency from architectural vulnerability. In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pages 117 –128, feb. 2009.

[38] V. Sridharan and D. R. Kaeli. Using hardware vulnerability factors to enhance mhae analysis. In *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10, pages 461–472, New York, NY, USA, 2010. ACM.

[39] C. Weaver, J. Emer, S. Mukherjee, and S. Reinhardt. Techniques to reduce the soft error rate of a high-performance microprocessor. In *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pages 264 – 275, june 2004.

[40] 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, pages 203–209, New York, NY, USA, 2005. ACM.

[41] Y. Yang, G. Gopalakrishnan, G. Lindstrom, and K. Slind. Analyzing the intel itanium memory ordering rules using logic programming and sat. In D. Geist and E. Tronci, editors, *Correct Hardware Design and Verification Methods*, volume 2860 of *Lecture Notes in Computer Science*, pages 81–95. Springer Berlin / Heidelberg, 2003.

[42] D. H. Yoon and M. Erez. Virtualized and flexible ecc for main memory. In *Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*, ASPLOS '10, pages 397–408, New York, NY, USA, 2010. ACM.

EKLER

A. Hata Sayfa Kotarıcısı Üzerinde Gerçeklenmiş Çözücünün Kaynak Kodu

Aşağıda verilmiş olan kod hata sayfa kotarıcısının girişinde çağırılan ve aranan sorguda bir hata olup olmadığını anlamak için kullanılan koddur. TBCV_ETKIN makrosu tanımlı olduğunda TBÇV buyruklarını kullanarak verinin bitlerini saymak için kullanılır, aksi takdirde daha düşük başarımlı olan bit sayma algoritması çalışır.

```
/* popcnt: x1 girdisinde bir olan bit sayisini dondurur */
uint64_t popcnt(uint64_t x1)
{
  #ifdef TBCV_ETKIN
  return _mm_popcnt_u64(x1);
  #else
  unsigned int c;
  for (c = 0; x1; c++)
  {
    x1 &= x1 - 1;
  }
  return c;
  #endif
}
```

```
/* decode_cw: recvd ile belirtilen sanal adreste hata var ise bunu d
              duzeltilmis halini dondurur */
uint64_t decode_cw(uint64_t recvd, uint64_t *synd_table)
{
  uint64_t result = 0;
  for(int i = 0; i < SYND_TABLE_SIZE; i++) {
    if((popcnt(recvd&synd_table[i])&1)) {
      __set_bit(i, &result);
    }
  }
  return result;
}
```

ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, Adı	: Yaman Çakmakçı
Uyruğu	: T.C.
Doğum tarihi ve yeri	: 24.12.1982 Ankara
Medeni hali	: Bekar
Telefon	: +90-533-7300888
Faks	:
e-mail	: ycakmakci@.etu.edu.tr

Eğitim

Derece	Eğitim Birimi	Mezuniyet Tarihi
Y. Lisans	TOBB Ekonomi ve Teknoloji Üniversitesi	2012
Lisans	Çankaya Üniversitesi	2007

İş Deneyimi

Yıl	Yer	Görev
2007-2011	TÜBİTAK UZAY	Araştırmacı

Yabancı Dil

İngilizce (Çok iyi)

Yayınlar

Mehmet Durna, Onur Atar, Mustafa Ceylan, Yaman Cakmakci, Mustafa Demirci, Omer Ali Kozal, Mehmet Oturak, Ali Ozdemir, Onur Turhan, "On Board Data Handling Subsytem Featuring BiLGE", Recent Advances in Space Technologies 2011