MİKROİŞLEMCİLERDE ETİKET KARŞILAŞTIRICILARININ KULLANILMASIYLA GEÇİCİ HATALARIN TESPİTİ

GÜLAY YALÇIN

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ

TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

EYLÜL 2007

ANKARA

Fen Bilimleri Enstitü onayı

_____

Prof. Dr. Yücel ERCAN

Müdür

Bu tezin Yüksek Lisans derecesinin tüm gereksinimlerini sağladığını onaylarım.

_____

Prof. Dr. Ali YAZICI

Anabilim Dalı Başkanı

Gülay YALÇIN tarafından hazırlanan MİKROİŞLEMCİLERDE ETİKET KARŞILAŞTIRICILARININ KULLANILMASIYLA GEÇİCİ HATALARIN TESPİTİ adlı bu tezin Yüksek Lisans tezi olarak uygun olduğunu onaylarım.

_____

Yrd. Doç. Dr. Oğuz ERGİN

Tez Danışmanı

Tez Jüri Üyeleri

Başkan :Doç. Dr. Elif Derya ÜBEYLİ _____

Üye : Yrd. Doç. Dr. Oğuz ERGİN _____

Üye : Yrd. Doç. Dr. Y.Murat ERTEN _____

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.

Gülay YALÇIN

Üniversitesi : **TOBB Ekonomi ve Teknoloji Üniversitesi**
Enstitüsü : **Fen Bilimleri**
Anabilim Dalı : **Bilgisayar Mühendisliği**
Tez Danışmanı : **Yrd. Doç. Dr. Oğuz ERGİN**
Tez Türü ve Tarihi : **Yüksek Lisans – Eylül 2007**

Gülay YALÇIN

MİKROİŞLEMCİLERDE ETİKET KARŞILAŞTIRICILARININ KULLANILMASIYLA GEÇİCİ HATALARIN TESPİTİ

ÖZET

Yüksek enerjili parçacıkların çarpması sonucunda oluşan geçici hatalar, mikroişlemci tasarımında, gün geçtikçe önemi artan bir sorun haline gelmektedir. Transistör sığasının artmasına da bağlı olarak, geçici hataların yakın gelecekte daha da çok önem kazanması beklenmektedir. Eğer geçici hata tespit edilebilirse, programın bir kısmındaki kodların yeniden çalıştırılmasıyla, bu beklenmedik hatanın kurtarılması mümkün olabilir. Aksi takdirde tespit edilemeyen geçici hata, mikroişlemcinin sebebi anlaşılamadan çökmesine neden olabilmektedir. Bu sebeple, geçici hataların tespit edilmesi için yeni tekniklerin geliştirilmesi son derece önemlidir.

Çağdaş mikroişlemciler, komutların sırasız olarak işletilmesi ve dinamik olarak zamanlanması ilkelerine dayanmaktadır. Karşılaştırıcı devreler, sırasız işletim içerisinde, komutlar arasındaki veri bağımlılıklarını kontrol etmek için kullanılmaktadır. Bu karşılaştırıcı devreler büyük oranda işsiz durumdadırlar ve boş durdukları zamanda yayın kuyruğunda yer alan komutların yazmaç etiket bilgilerinde oluşabilecek geçici hataların tespiti için kullanmaları mümkündür. Bu çalışmada yonga üzerinde yer alan karşılaştırıcı devrelerin geçici hataların tespit edilmesi için kullanılmasına olanak sağlayan çeşitli yöntemler geliştirilmiştir.

Anahtar Kelimeler: Karşılaştırıcılar, bilgisayar mimarisi, hata tespit kodu, hata toleransı, mikroişlemciler.

University : **TOBB Economics and Technology University**
Institute : **Institute of Natural and Applied Sciences**
Science Programme : **Computer Engineering**
Supervisor : **Assistant Professor Dr. Oğuz ERGİN**
Degree Awarded and Date : **M.Sc. – September 2007**

Gülay YALÇIN

SOFT ERROR DETECTION BY USING TAG COMPERATOR IN MICROPROCESSORS

ABSTRACT

Soft errors caused by high energy particle strikes are becoming an increasingly important problem in microprocessor design. With increasing transistor density and die sizes soft errors are expected to be a larger problem in the near future. Recovering from these unexpected faults may be possible by reexecuting some part of the program if the error can be detected. Therefore it is important to come up with new techniques to detect soft errors.

Modern microprocessors employ out-of-order execution and dynamic scheduling logic. Comparator circuits used to keep track of data dependencies are usually idle and can be used to detect the soft errors on the stored tag information inside the issue queue. In this thesis, we propose various schemes to exploit on-chip comparators to detect transient faults.

Keywords: Comparators, computer architecture, error detection coding, fault tolerance, microprocessors

TEŞEKKÜR

Çalışmalarım boyunca değerli yardım ve katkılarıyla beni yönlendiren danışmanım Dr. Oğuz ERGİN'e, yine kıymetli tecrübelerinden faydalandığım TOBB Ekonomi ve Teknoloji Üniversitesi Bilgisayar Mühendisliği Bölümü öğretim üyelerine ve yüksek lisans öğrencilerine, son olarak desteğini esirgemeyen ailem ve arkadaşlarıma teşekkürü bir borç bilirim.

İÇİNDEKİLER

	Sayfa
TEZ BİLDİRİMİ	iii
ÖZET	iv
ABSTRACT	v
TEŞEKKÜR	vi
İÇİNDEKİLER	vii
ÇİZELGELERİN LİSTESİ	xi
ŞEKİLLERİN LİSTESİ	xii
KISALTMALAR	xiii
SEMBOL LİSTESİ	xiv
BÖLÜM 1	1
1. GİRİŞ ve ÇALIŞMANIN AMACI	1
BÖLÜM 2	3
2. YÜKSEK BAŞARIMLI İŞLEMCİLERİN BİLEŞENLERİ	3
2.1. Komutların Getirilmesi	6
2.2. Çözümleme, Yeniden Adlandırma ve Aktarma	6
2.3. Yayınlama ve Paralel İşletim	7

2.4. Bellek İşlemlerinin Ele Alınışı 9

2.5. Sonlandırma 9

BÖLÜM 3 11

3. GEÇİCİ HATA 11

3.1. Geçici Hata Nedir? 11

3.2. Geçici Hatanın Tarihçesi ve IBM'in Çalışmaları 12

3.3. Geçici Hatanın Geleceği 17

3.4. Bellek Birimlerinde Geçici Hataların Ele Alınışı 18

3.4.1. Eşlik Biti Denetimi 18

3.4.2. Hamming Kodları 19

3.4.3. Fazlalık Yöntemi 20

3.5. İşlemci Bitlerinin Sınıflandırılması 20

3.5.1. DÇG Olmayan Bit Kaynakları 21

3.5.1.1. İşsiz ya da Geçersiz Durumlar: 21

3.5.1.2. Yanlış Tahmin Edilen Durumlar: 21

3.5.1.3. Tahmin Durumları: 21

3.5.1.4. DÇG Sonrası Durumu: 22

3.5.1.5. NOP Komutları: 22

3.5.1.6. Başarımı Artıran Komutlar: 22
3.5.1.7. Yanlış Doğrulanmış Komutlar: 22
3.5.1.8. Devingen Ölü Komutlar: 23
3.5.1.9. Mantıksal Maskeleme: 23
3.6. İşlemci Güvenilirliğinin Ölçülmesi 24
3.7. İşlemci Güvenilirliğini Artırmak İçin Yapılan Çalışmalar 26
BÖLÜM 4 29
4. ETİKETLERDE GEÇİCİ HATA TESPİT YÖNTEMLERİ 29
4.1. İşlenenlerin Kıyaslanması ile Geçici Hata Tespiti 29
4.1.1. Bir İşlenenli Komutlarda Geçici Hata Tespiti 30
4.1.2. İki İşleneni Birbirinin Aynı olan Komutlarda Geçici Hata Tespiti 34
4.1.3. Bir İşleneni Hazır Olan Komutlarda Geçici Hata Tespiti 34
4.2. MHÇ Hesaplanması 36
BÖLÜM 5 37
5. DENEYSEL SONUÇLAR ve KIYASLAMALAR 37
5.1. Benzetim Ortamları 37
5.1.1. M-Sim 37
5.1.2. PTLsim 38
5.2. Gerekli Parametreler 38

5.3. Benzetim Ortamında Benzetilen İşlemcinin Özellikleri 43

5.4. Sonuçlar 44

5.4.1. M-Sim sonuçları 45

5.5. Bellekten Kayıp Miktarı 53

BÖLÜM 6 54

6. SONUÇLAR 54

KAYNAKLAR 56

ÖZGEÇMİŞ 58

ÇİZELGELERİN LİSTESİ

Çizelge	**Sayfa**
Çizelge 4.1. HATA Oluşma Mantık Çizelgesi	32
Çizelge 5.1.Benzetimlik Yapılandırmaları	43
Çizelge 5.2. Benzetimlerin yapıldığı denektaşları	45
Çizelge 5.3. M-Sim Benzetim Sonuçları	46

ŞEKİLLERİN LİSTESİ

Şekil	Sayfa
Şekil 2.1. Yayın Kuyruğunda Karşılaştırıcı Devrelerin Durumu	8
Şekil 4.1 "aynı_etiket" Biti Kullanılarak Geçici Hatanın Tespit Devresi	31
Şekil 4.2. Bir Bit Farklı Etiketlerin Örneği	33
Şekil 5.1 Komutların Program Boyunca Yüzdelik Dağılımı	47
Şekil 5.2. Önerilen Tekniğin Kapsamı	48
Şekil 5.3.Yayın Kuyruğundaki Ortalama Anlık Komutlar	49
Şekil 5.4 Önerilen Teknikten Önce Etiketlerin MHÇ Değeri	50
Şekil 5.5. Önerilen Teknikten Sonra Etiketlerin MHÇ Değeri	50
Şekil 5.6.Yayın Kuyruğunun MHÇ değeri (Önerilen Teknikten Önce)	52
Şekil 5.7.Yayın Kuyruğu MHÇ Değeri (Önerilen Teknikten Sonra)	52

KISALTMALAR

Kısaltmalar	Açıklama
OKS	Yayın kuyruğunda yer alan ortalama komut sayısı
MHÇ	Mimari Hassasiyet Çarpanı
BDH	Bulunan, Düzeltilemeyen Hata
GVB	Gizli Veri Bozulması
DÇG	Doğru Çalışma için Gerekli
ACE	Architecturally Correct Execution
CAM	Content Addressable Memory
SRAM	Static Random Access Memory
YSO	Yazmadan Sonra Okuma
YSY	Yazmadan Sonra Yazma
OSY	Okumadan Sonra Yazma
ADÖ	Adres Dönüştürme Önbelleği
SEC/DEC	Single Error Correction/ Double Error Detection
BDÖ	Birinci Seviye Devingen Ölü
GDÖ	Geçişli Devingen Ölü
BSH	Birim Süredeki Hata
HAOS	Hatalar Arasındaki Ortalama Süre
HKGS	Hataya Kadar Geçen Süre
HTGS	Hatanın Telafisi için Geçen Süre
ÇBK	Çevrim Başına Komut
SSHT	Sonlandırma Sonrası Hata Takip
HTK	Hata Takip Kodu
YSK	Yeniden Sıralama Kuyruğu

SEMBOL LİSTESİ

Bu çalışmada kullanılmış olan simgeler açıklamaları ile birlikte aşağıda sunulmuştur.

Simgeler	**Açıklama**
K	Komut kümesi
k_i	Bir komut, $k_i \in K$
$kgiriş_i$	k_i komutunun yayın kuyruğuna girdiği saat vuruşu
$kçıkış_i$	k_i komutunun yayın kuyruğundan ayrıldığı saat vuruşu
$kkalış_i$	k_i komutunun yayın kuyruğunda kaldığı süre
T	Toplam saat vuruş sayısı
T	Saat vuruşu
K_t	t anında yayın kuyruğunda yer alan komut sayısı ($K_t \subset K$)

İndisler	**Açıklama**

Üsler	**Açıklama**

BÖLÜM 1

1. GİRİŞ ve ÇALIŞMANIN AMACI

Günümüz mikroişlemcilerinde, Moore yasasında da öngörüldüğü gibi, boyut ve gerilim değerlerinin artması geçici hatalarda bir artışa yol açmaktadır. Bu hatalar yonga üzerinde kalıcı bir hasara neden olmadıkları için geçici hata olarak adlandırılmakla birlikte, tespit edilemedikleri takdirde sistemin çökmesine neden olmaktadır [1]. Geçici hatanın tespit edilmesi durumunda ise boru hattındaki komutların temizlenmesi ve hatanın oluştuğu komuttan itibaren komutların yeniden getirilmesi ile işlemci çökmekten kurtarılıp güvenli çalışmasına devam edebilmektedir. İşte bu sebeple, tespit edilmedikleri takdirde işlemci boru hattında fark edilmeden gizli veri bozulmasına sebep olan geçici hataların tespiti önem kazanmaktadır.

Günümüzde kullanılan ve sırasız işletime izin veren mikroişlemcilerin en önemli bileşenlerinden bir tanesi de komutların zamanlanmasıdır. Komutlar getirilme, çözümlenme ve yeniden adlandırılma aşamalarından sonra yayın kuyruğuna atılmakta ve belirli bir işlem birimine tahsis edilmeden önce işlenenlerinin hazır konuma geçmesi için beklemektedirler. Yayın kuyruğunda yer alan tüm komutların, sonuç değerlerini tutan yazmaçların erişilebilir olup olmadığından haberdar edilmeleri gerekmektedir. Bu sebeple işletimi tamamlanan her komut, sonucunu ürettiği yazmacın etiketini yayın kuyruğunda yer alan tüm komutlara yayımlamaktadır. Komutlar, her saat vuruşunda, işlenenlerinin hazır olup olmadığını kontrol etmek için, kendi kaynak yazmaçlarının etiketleri ile yayımlanan yazmacın etiketini karşılaştırır. Saklanan işlenenin etiketi ile yayımlanan etiketi karşılaştırmak için CAM (İçerik Adresleyebilen Bellek) devreleri kullanılır. CAM hücreleri, düzenli SRAM bit hücrelerinin ve karşılaştırıcı devrelerin bir araya gelmesinden oluşur ve dinamik mantık kullanarak çalışırlar. Ancak "Etiket-Karşılaştırıcıları" çoğu zaman "aynı-değil" sinyali ürettikleri için etkin şekilde çalışmamaktadırlar [2].

İşlemcideki yayın kuyruğu aranan verilerin ön bellekte bulunamaması gibi olaylara bağlı olarak tamamen dolu olabilir ve komutlar herhangi bir işlem birimine atanmadan önce uzun süre yayın kuyruğunda kalabilirler. Yayın kuyruğunda geciken komutların parçacık çarpması nedeniyle zarar görmesi mümkündür. Önbellekte

aranan verilerin bulunmaması durumunda komut kuyruğunun temizlenmesi gibi çözümler önerilmiş olsa da böyle bir çözüm boru hattının yeniden doldurulmasını gerektirdiğinden başarımın azalmasına neden olmaktadır [3].

Bu çalışmada, işlemcilerin içyapısında zaten var olan etiket karşılaştırıcılar kullanılarak saklanan verilerde ya da ilgili devrelerde oluşabilecek geçici hataların tespit edilmesini sağlayan çeşitli tasarımlar önerilmiştir. İlk olarak işlemcinin kullandığı komutların pek çoğunun sadece bir işlenen kullandığı gözlemlenmiş ve bu işlenen değeri ikinci etiket alanına kopyalanarak, boşta kalan ikinci karşılaştırıcıların işlenen etiketlerinin doğrulanması için kullanılması sağlanmıştır. Daha sonra iki işleneni birbirinin aynı olan komutlarda, karşılaştırıcı devrelerin ürettiği aynı ya da aynı değil sinyalleri ile işlenenlerde oluşabilecek geçici hatalar tespit edilmiştir. Son olarak iki işlenenli komutların pek çoğunun yayın kuyruğuna girdiği sırada bir işleneninin zaten hazır olduğu gözlemine [4] dayanarak beklenen etiket değeri her iki işlenen etiket alanına kopyalanmış ve beklenen etiket değeri geçici hatalara karşı korunmuştur.

BÖLÜM 2

2. YÜKSEK BAŞARIMLI İŞLEMCİLERİN BİLEŞENLERİ

Programların ve komut kümelerinin doğasında bulunan sıralı işleme mantığı, ilk bilgisayarların yıllar önceki tasarım mantığı ile tıpatıp aynıdır. Bir programın komutlarının sıra ile işletilmesi ve ana bellekten bir komutun işlemciye getirilmesi için program sayacı kullanılır. Program sayacının gösterdiği adresteki komut işlemcinin her saat vuruşunda işlemciye getirilip işletilir. İşletim sırasında ana bellekten veri çekilmesi, ana belleğe veri yazılması ya da yazmaçlar üzerinde işlem yapılması gerekebilir. Programın işletim mantığına göre, komutun işletimi biter bitmez, program sayacının artırılmasıyla, ana bellekten sıradaki komut çekilmelidir.

Günümüzde kullanılan yüksek başarımlı işlemcilerin başlıca amacı verilen bir programın işletim süresini mümkün olduğunca azaltmaktır. Her bir komutun işletimi belirli bir gecikmeye neden olmaktadır. Programın işletim süresini azaltmak için ya her bir komutun gecikmesi azaltılmalı ya da aynı anda paralel olarak birden fazla komutun işletilmesi gerekmektedir.

Çağdaş mikroişlemcilerde, aynı saat vuruşunda birden fazla komutun başlatılması mantığı daha hızlı işlemcilerin geliştirilmesine yol açmıştır. Bir komutun işletimi adımlara bölünerek her adımda işlem gören komutların yer alabilmesi sağlanmış ve böylece sadece bir komutun işletilebileceği zamanda birden fazla komutun işlemini tamamlayabilmesi sağlanmıştır.

Bir komutun işletiminde geçtiği değişik aşamaların oluşturduğu yapı boru hattı olarak adlandırılır. Boru hattı, komutun getirilmesi, çözümlenmesi, yeniden adlandırılması, yayın kuyruğuna atılması, uyandırılıp seçilmesi, yazmacın okunması, çalıştırılması ve sonlandırılması olmak üzere 8 aşamadan oluşmaktadır. Her bir aşama birden fazla saat vuruşunda gerçekleşebilir. Örneğin load (yükle) komutu için çalıştırma aşaması, yazmanın yapılacağı bellek adresinin hesaplanması ve hesaplanan adrese erişilmesi olmak üzere iki ya da daha fazla saat vuruşu içerir [5].

Sıradan bir çok yollu işlemci aynı anda birden çok komutu bellekten getirip çözümleyebilir. Komutların ana bellekten getirilmesi aşamasında, komutların kesintisiz olarak işletilmesini sağlayabilmek için, şartlı dallanma komutlarının sonuçları tahmin edilir. Daha sonra art arda gelen komutlar, veri bağımlılıkları açısından incelenir ve komutun türüne göre işletim birimlerine dağıtılır. Paralel işletimin sağlanabilmesi için, program sırası yerine komutların işlenenlerinin hazır olup olmamasına bağlı olarak, komutlar işletime alınırlar. Pek çok değişik çok yollu işlemcide var olan bu önemli özellik "komutların devingen zamanlanması" olarak adlandırılır. İşletimin tamamlanmış olması şartıyla, komutların sonuçları yeniden sıraya dizilir. Böylece işletim sırası yeniden programın asıl, doğru sırasında olacak şekilde sonuçlar güncellenir. Komutların sadece işletim aşamaları paralel olarak gerçekleştiği için, işlemcilerin bu özelliği komut düzeyinde koşutluk olarak adlandırılır [6].

Bir uygulama programı üst düzey bir programlama dilinde yazılır, daha sonra derlenerek makine düzeyindeki durağan programa ya da başka bir deyişle ikilik düzendeki programa derlenir. Durağan program, komutların tam olarak işletileceği sırada yer aldığı bir program modelidir. Durağan bir program belirli bir veri kümesi üzerinde işlem yapıyorken, işletilmiş olan komutlar devingen komut sıraları biçiminde şekillenirler. Çalıştırılması gereken komutlar birbirini takip eden bir sırada olsalar da, program sayacının bir sonraki komutu gösterecek şekilde artırılmasıyla durağan komutların devingen bir şekilde işletilmesi sağlanır. Ancak şartlı bir dallanma komutu ya da jump (atla) komutu bulunduğunda program sayacının program sırasının dışında bir konuma atlayacak biçimde güncellenmesi gerekir. Bu noktadan sonra işletilen devingen komutlara, önceki dallanma ve atlama komutlarına "denetim açısından bağımlı" denir. Çünkü işletilen komut akışı önceki dallanma komutlarının denetimindedir. Program sayacının değerinin değişmesine neden olan artırma ve güncelleme durumlarında iki farklı bağımlılık oluşur.

Çalıştırılma aşamasına gelen komutlar veri bağımlılığı sınırlamasına takılabilirler. Çünkü art arda işletilen komutların ortak bir yazmaç ya da bellek alanına, okuma ya da yazma işlemleri için erişmeleri gerekebilir. Komutlar aynı saklama alanına erişmeye çalıştıklarında, alan üzerinde işlem yapacak komutların sıralı işletimini sağlayacak önlemler alınmadığı zaman, bir sorunla karşılaşılabilir. Aslında sadece aralarında gerçek veri bağımlığı bulunan komutların işletim sırası için bir sınırlama

söz konusudur ve önlem alınmalıdır. Gerçek veri bağımlılığı Yazmadan Sonra Okuma (YSO) gerektiği durumda oluşmaktadır. Okuma yapan tüketici komutun, ancak yazma yapan üretici komut gerekli değeri yazdıktan sonra okuma yapması gerekir.

Mimari düzeyinde başka bağımlılık türleri de tanımlanmaktadır ve programın çalışması sırasında komutların paralel işletimi için bu bağımlılıklar hakkında da önlemler alınmalıdır. Mimari düzeyindeki bu bağımlılıklar Okumadan Sonra Yazma (OSY) ve Yazmadan Sonra Yazma (YSY) bağımlılıklarıdır. OSY, yazma yapacak olan komutun, önceki değeri okuyacak olan tüm komutların işletiminin tamamlanmasını beklemesinin gerektiği durumlarda soruna neden olmaktadır. YSY sorunu ise birden fazla komutun aynı saklama alanını güncellerken alanı doğru sırada güncellemeleri gerektiği durumlarda oluşur. Mimari bağımlılıklar, elverişsiz kodlar, yazmaç alanlarının sınırlılığı, ana bellek alanının ekonomik kullanımı ve döngü olması gibi sebeplerden oluşabilir. Mimari ve denetim bağımlılıklarının çözümlenmesinden sonra komutlar paralel olarak yayınlanıp işletilebilirler.

Aslında donanım paralel işletim zamanlamasını şekillendirir. Bu zamanlamada gerçek veri bağımlılıkları ve işlem birimleri ya da veri yollarının sınırlı olması gibi durumları göz önünde bulundurur.

Komutların paralel çalışması, komutların programdaki asıl sıradan farklı olarak işletimlerini tamamlaması anlamına gelir. Tahmine dayalı çalışma ise komutların olağan işletim sırasında hiç çalışmaması gerekse bile işletime alınabilmesi durumudur. Bu durumun en belirgin örneği, dallanmanın yanlış tahmin edildiği bir durumda işletilen komutlardır.

Sonuç olarak dışarıdan görünen mimari saklama alanları komutun işletimi tamamlanır tamamlanmaz güncellenmezler. Bunun yerine, mimari olarak doğru durum oluşana kadar, komutların sonuçları geçici bellek alanlarında tutulur. Bunun da ötesinde, yüksek başarımın sağlanabilmesi için, sonuçlar bağlı komutlar tarafından kullanılabilmelidir. En son olarak, bir komutun sıralı işletime göre

işletilmiş olması gerektiğinde, komutun geçici sonucuna göre işlemcinin mimari durumu güncellenir ve geçici sonuçlar kalıcı hale getirilir [6].

2.1. Komutların Getirilmesi

Hemen hemen bütün mikroişlemcilerde, en son kullanılan komutların tutulduğu ve komut önbelleği olarak adlandırılan küçük bir bellek alanı yer alır. Bu bellek alanı gecikmeyi azaltır ve birim zamanda işlemciye getirilebilecek komut sayısını artırır. Komut önbelleği, birbirini takip eden komutları içeren bloklar ya da satırlar halinde düzenlenmiştir. Program sayacının içerik adreslenerek kullanılmasıyla, bir komutun önbellekte yer alıp almadığı anlaşılır. Eğer komut önbellekte ise isabetli değilse isabetsiz durumu oluşur. Ön bellekte bulunmadığı durumda komut ana bellekten önbelleğe getirilir [6].

2.2. Çözümleme, Yeniden Adlandırma ve Aktarma

Bu aşamada komutlar komut getirme ara belleğinden silinir, denetim ve veri bağımlılıkları denetlenerek boru hattının geri kalanı için bu bağımlılıklar kurulur. Ayrıca bu aşama verilerin YSO gibi gerçek bağımlılıklarının tespit edilmesini ve YSY ya da OSY gibi yanlış bağımlılıkların çözümlenmesini içerir [7]. Bu aşamada komutlar daha sonra yayınlanıp çalıştırılacakları ve donanımın işlem birimleriyle ilgili olan kuyruğa aktarılırlar. Komutlar kuyruğa aktarılmadan önce kullandıkları mimari yazmaç adları yeni fiziksel yazmaç adları ile değiştirilirler. Komutlar arasında yazmaçların veri bağımlılığını çözmeye yarayan bu işlem yeniden adlandırma işlemi olarak adlandırılır [5].

Genel olarak iki yeniden adlandırma metodu vardır. İlk yöntemde mimari yazmaç sayısından daha fazla yazmaç içeren fiziksel yazmaç dosyası bulunur. Fiziksel yazmacın o anda hangi mantıksal yazmaç yerine kullanıldığının ilişkilendirilmesi için bir adresleme tablosu tutulur. Komutlar işletim sırasında çözümlenir ve yeniden adlandırılır. Bir komut çözümlendiğinde sonucu boştaki fiziksel yazmaçlar listesinden bir yazmaca atanır. Atamanın yapıldığı fiziksel yazmaca o ana kadar mantıksal olarak başka bir değer atanmamıştır ve bu noktadan sonra adresleme tablosu bu mantıksal yazmacın değeri istendiğinde fiziksel yazmacın değerini

verecektir. Ardından atamanın yapıldığı fiziksel yazmaç boştaki fiziksel yazmaçlar listesinden silinir. Yeniden adlandırma aşamasının bir parçası olarak komutun kaynak yazmaçlarının o andaki fiziksel yazmaç değerleri de adresleme tablosundan bakılır. Bakılan değerler kaynak işlenenlerin hangi konumdan okunacağı bilgisini sağlar. Boştaki fiziksel yazmaç listesi boşaldığı anda komutların sevk edilmesi durdurulur.

Yeniden adlandırmanın ikinci yönteminde fiziksel yazmaç dosyasının boyu mantıksal yazmaç dosyası ile aynıdır ve aralarında birebir bir eşleme bulunur. Ayrıca sevk edilmiş ancak sonlanmamış her bir komut için bir satır içeren bir kuyruk yer alır. Bu kuyruk alanı aynı zamanda aykırı durumların oluşması durumunda komutların işletim sırasını düzelttiği için yeniden sıralama kuyruğu olarak adlandırılır [6]. Bu yöntemde, yazmaçların yeniden adlandırılması tek bir saklama hücresinin kullanılmasıyla doğrudan yapılır. Saklama alanları arasında mimari yazmaç dosyası ve yardımcı veri alanı ayrımı yoktur. Bunun yerine bütün değerler için yer ayrılan ve tüm değerlerin yazıldığı tek bir fiziksel dosya kümesi vardır. Mimari yazmaçlar kullanılabilen fiziksel yazmaçların bir alt kümesine adreslenir. Bu adresleme komutlar yayınlandıkça değişir. Mimari yazmaç isimleri ile fiziksel yazmaçların ilişkileri adresleme tablosunda tutulur. Adresleme tablosu her bir yazmaç ismi için bir satır içerir ve bu satır fiziksel yazmacın o anda bir mimari yazmaçla ilişkilendirilip ilişkilendirilmediğini ya da bir mimari yazmaç tarafından adreslenip adreslenmediği bilgisini tutar. Ayrıca gerekli olduğunda yeni fiziksel yazmaçların sağlanması için boş yazmaç havuzu yer alır. Komut sonucunda oluşan çıktı yazmaçları boş yazmaç havuzundan seçilen bir fiziksel yazmaca adreslenir. Bu yeni ilişkiyi göstermek için adresleme tablosu güncellenir. Çıktı yazmacının ismine ait satırdaki eski fiziksel yazmaç, yeni fiziksel yazmaç ile değiştirilir. Böylece yazmacın adı yeni yazmaca adreslenmiş ve eski fiziksel yazmaç da serbest bırakılmış olur [7].

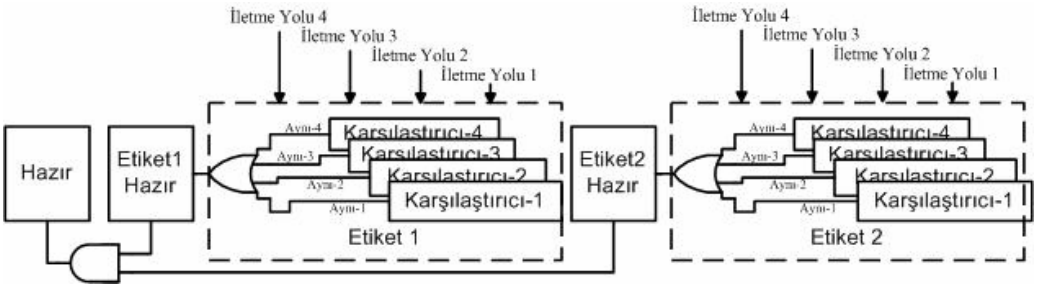
2.3. Yayınlama ve Paralel İşletim

Günümüz çağdaş mikroişlemcileri komutların sırasız işletimine olanak sağlayarak, işlem birimlerini verimli kullanmayı ve programların daha erken sonlanmasını sağlamaktadır. Veri bağımlılıklarının denetlenebilmesi için yayın kuyruğunda pek çok karşılaştırıcı devre mevcuttur. Şekil 2.1'de 4 yollu bir makine için karşılaştırıcı devrelerin durumu gösterilmiştir. 4 yollu makine yayın kuyruğundan bir seferde en

fazla 4 tane komut çıkmasına olanak veren ve üretilen 4 sonucun sonuçlarının karşılaştırılabilmesini sağlayan bir makinedir.

Her bir komut, her saat vuruşunda üretilen sonuçları kendisine ait her iki işlenen değeri ile karşılaştırır. Bu sebeple, her saat vuruşunda N adet sonuç üreten bir işlemci için, her etiket için N tane karşılaştırıcı devre gerekmektedir. Herhangi bir karşılaştırıcı "aynı" sinyali üretirse, ilgili etiket ulaşılabilir olarak kurulur. Her iki etiket değeri de ulaşılabilir olarak kurulduğunda, komut yayın kuyruğundan çıkmaya hazır olarak işaretlenir.

Her bir işlenen etiketinin sahip olması gereken bit sayısı işlemci içerisindeki fiziksel yazmaç sayısına bağlıdır. Örneğin M adet fiziksel yazmacı olan bir işlemcinin komutlarındaki işlenen etiketlerinin boyu $\log_2^M$'dir. Bu sayı aynı zamanda Şekil.2.1'deki karşılaştırıcı devrelerin girdi sayısını da gösterir.



Şekil 2.1. Yayın Kuyruğunda Karşılaştırıcı Devrelerin Durumu

Bir komut işlenenleri hazır olur olmaz işletilmeye hazırdır. Ancak komutun yayınlanmasında başka sınırlamalar da bulunabilir. Bu kısıtlamalara, işlem biriminin, aradaki bağlantıların ya da yazmaç dosyası kapılarının erişilemez olması verilebilir. Tomasulo tarafından tanımlandığı gibi, her bir işlem birimi için adanmış RSE kümesi mevcuttur [8].Yayınlanacak komutun seçilmesi, hazır olan komutlar havuzundan, her bir işlem birimiyle ilişkili olarak, bir sonraki saat vuruşunda işletilecek olan komutun seçilmesidir [5].

2.4. Bellek İşlemlerinin Ele Alınışı

Aritmetik Mantık Birimi işlemlerinde komutların çözülme aşamasında hangi yazmaç alanından işlenen değerinin elde edileceği bilinebiliyorken, bellek işlemlerinde bellekten getirme ya da belleğe yazma işleminin hangi adres kullanılarak yapılacağının bilinmesi yayınlama aşamasının son aşamasına kadar belli değildir. Bellekte erişilecek olan yer, bir adres hesaplama işlemi ile bulunur ve bu işlem genellikle bir tamsayı toplama işlemidir. Bunun için belleğe yazma ve bellekten okuma komutları, adres hesaplama işlemi gerçekleşirken yayınlanırlar. Adres hesaplama işleminden sonra adreslerin fiziksel adrese dönüştürülmesi de gerekebilir. Adres Dönüştürme Önbelleği (ADÖ) daha önceden erişilen sayfaların adres dönüşümlerini tutan bir bellek alanıdır ve amacı adres dönüşüm işlemini böylece hızlandırmaktır. Geçerli bir bellek adresi elde edildikten sonra okuma ya da yazma işlemi belleğe gönderilebilir. Adres dönüşümü ve belleğe erişim işlemlerinin art arda yapıldığı düşünülse de, birçok çok yollu işlemcide bu adımlar üst üste binmektedir. Ön belleğe ilk erişim, adres dönüşümü ile paralel olarak gerçekleştirilmektedir ve dönüştürülen adres ön bellek etiketi ile karşılaştırılıp aranan bilginin önbellekte olup olmadığı anlaşılır [6].

2.5. Sonlandırma

Bir komutun yaşam döngüsündeki son aşama sonlandırma ve emeklilik aşamasıdır. Bu aşamada komut işlemcinin mantıksal durumunu değiştirir. Sonlandırma işlemi için iki teknik önerilmiştir:

İlk teknikte belirli aşamalarda makinenin o andaki durumu “geçmiş kuyruğuna” ya da denetim noktalarına kaydedilir. Başka bir deyişle işlemci içinde programa ait denetim noktaları konur. Komutlar çalıştırılır çalıştırılmaz makinenin durumunu değiştirirler ve eğer geçmişteki bir duruma dönülmesine ihtiyaç duyulursa bu durum geçmiş kuyruğundan yeniden yüklenir. Bu yöntemin kilit noktası artık ihtiyaç duyulmayacak geçmiş durumların sonlandırma aşamasında yok edilmesidir.

İkinci teknikte, makinenin durumu gerçekleştirilen fiziksel durum ve mantıksal (mimari) durum olmak üzere ikiye bölünür. Fiziksel durum bir işlem tamamlanır tamamlanmaz değiştirilir. Makinenin mimari durumu ise program işletim sırasına göre değiştirilir; tahmine dayalı işlemlerde değiştirilmez. Bu teknikte yeniden sıralama kuyruğunda saklanan bir komutun sonlandırılması için komutun sonucu yeniden sıralama kuyruğundan silinip mimari yazmaç dosyasına yazılır. Eğer komut belleğe yazma komutuysa değer belleğe ancak sonlandırma sırasında yazılır ve yeniden sıralama kuyruğundaki satır da silinir. Ayrıca belleğe yazma komutlarının sonlandırma aşamasında belleğin yazma kuyruğuna bir sinyal gönderilerek verinin belleğe yazılması sağlanır [6].

BÖLÜM 3

3. GEÇİCİ HATA

Radyasyon olayları sonucunda saklanan bitlerin değerinin değişmesine geçici hata denir. Bu tür olaylarda işlemci üzerinde kalıcı bir hasar oluşmamakla beraber mikroişlemcinin çalışması istenmeyen şekilde sonlanır. İşlemcilerde kalıcı hata oranı 1-500 BSH (Birim Süredeki Hata) arasında iken geçici hata oranı gelişmiş teknolojilerde 50.000 BSH olabilmektedir.

Son yirmi yılda üç radyasyon mekanizması geçici hataların kaynağı olarak belirlenmiştir. Yongaların paketlenmesi sırasında oluşan alfa parçacıkları geçici hataların en önemli kaynağıdır. Benzer şekilde kozmik radyasyonlar sonucunda oluşan nötron parçacıkları da geçici hataların bir diğer kaynağıdır. Son olarak düşük enerjili kozmik nötron parçacıklarının cihaz üzerindeki materyallerle etkileşmesi de geçici hatanın bir kaynağıdır [9].

3.1. Geçici Hata Nedir?

Geçici hatalar özellikle nötron parçacıklarından kaynaklanan radyasyon tarafından oluşturulur. Ne yazık ki tümleşik devrelerin paketlenmesi sırasında fiziksel koruma yöntemleriyle nötronların durdurulması çok zordur. Yongaların paketlenmesi sırasındaki kirlilikten kaynaklı alfa parçacıkları gibi diğer radyoaktif kaynaklar da geçici hatalara neden olabilir, ancak bunların azaltılması çok daha kolaydır ve sistemin güvenilirliğini daha az tehdit etmektedirler.

Sistem üzerinde oluşan geçici hatalar, kalıcı hasara neden olmayıp düzeltilebildikleri için bu ismi alırlar [1].

Uzmanlar geçici hata sorununun, devrelerin şeklinin küçülmesi ve daha yoğun bir hal alması ile iyiye değil kötüye giden bir sorun olduğu konusunda hemfikirlerdir [10].

Yonga üreticileri için, güvenilirliğin öncelikli olduğu uygulamalarda devredeki arıza oluşma ihtimalinin ciddiyeti yüzünden, güvenilirlik başlıca ekonomik sorun haline gelebilir. Satılmış bir ürünün ya da aracın çökmesi pahalı bir saha tamiratı ve güvenilirlik üzerine kötü bir ün getirebilir [10].

Geçici hatalar daha çok, bellek birimleri, yazmaçlar ve mandallar gibi saklama birimlerini etkileme eğilimindedirler. Geçici hata oranı nötron yoğunluğu ile doğru orantılı olarak artmaktadır. Nötron yoğunluğu ise yükseklik ve enlemin her ikisine de bağlıdır. Bu sebepten ötürü, geçici hata sorunu yüksekteki uçuş araçlarında 100 - 1000 kat daha kötü durumdadır.

Genel olarak taşınabilir ve kablosuz ürünlerde kullanılan gömülü işlemciler kişisel bilgisayarlara göre düşük saat vuruş hızlarında işlem yapacak şekilde tasarlanmıştır. İki saat vuruşu arasındaki süre daha uzun olduğu için gömülü sistemlerde geçici bir hata oluştuğunda sistemin bu hata sebebiyle çökmesi daha geç olur. Bunun yanında gömülü sistemler daha az miktarda bellek alanı kullandığı için de geçici hatalardan daha az etkilenmektedir.

3.2. Geçici Hatanın Tarihçesi ve IBM'in Çalışmaları

IBM'in araştırmalar sayfasında, geçici hataların keşfi ve IBM içerisinde geçici bozulmalar hakkında yapılan çalışmalar, deneyler yayınlanmıştır. Yayınlanan bu araştırma 1978–1994 yılları arasında yapılan yaklaşık 15 yıllık çalışmaları içermektedir [11].Bu keşifler, çalışmalar ve deneyler ana hatlarıyla şu şekilde sıralanabilir:

İlk olarak 1954–1957 yılları arasında, yer üstü nükleer bomba testleri sırasında, görüntüleme araçlarında pek çok elektronik garipliklerin gerçekleştiği kaydedilmiştir. "Kalıcı Hata", aracın yüksek miktarda reaksiyona maruz kalıp çalışamaz hale gelmesi olarak tanımlanmıştır. Bir başka bozuk durum da, bombaların elektromanyetik şok dalgalarından doğan elektronik gürültüsünden, elektronik sinyallerin etkilenmesidir ve bu durumda hataların parçacık miktarı yaklaşık 10^{11} parçacık/cm^2 olduğunda başladığı düşünülmektedir.

1963 yılında, International Quiet Sun Year'da, ilk olarak kozmik ışınlar ve akıntılar hakkında ayrıntılı bilgi sağlanmıştır. İlk uydu deneylerinden alınan sonuçlar, 1GeV'luk enerjiye sahip parçacıkların 1600 parçacık/m^2-s ile uyduyu bombaladığını göstermiştir. Solar döngü, kozmik ışın akıntısının bir başka kaynağı olmakla birlikte bu parçacıklar deniz seviyesine ulaştıklarında çok düşük enerjiye sahip oldukları için herhangi bir akıma sebep olmamaktadırlar.

1962–1970 yılları arasında, daha önceki uyduların elektroniği güvensiz bulunduğu için devrelere önemli derecede fazlalıkların eklenmesi gerekmiştir. Uyduların başlıca sorunu güneş ışınlarında farklılık gösteren uydu yüklerinin gürültüye sebep olması ve uydu birimleri arasında yay çizmesidir. Bu durumun bir çözümü yük ve ısı farkını en aza indirmek için uydunun altınla kaplanmış MYLAR battaniye ile örtülmesidir. Ayrıca yer ile veri iletişimi gürültülü olmaktadır ve elektronik geçici bozulmalar iletişim hatalarından ayırt edilememektedir. Bu durumdan kurtulmak için iletişimin çoğunluğu küçük veri parçalarına bölünmüştür ve iletişim bilgilerine ek olarak verilere eşlik biti ve "el sıkışma" protokolleri eklenmiştir.

1975 yılında, Hughes Aircraft Co.'dan Binder ve arkadaşları, uydularda yer alan ve yük sorunlarından kaynaklanmadığını düşündükleri dört garipliği inceleyip yayınlamışlardır. Bu gariplikler işlemlerin 17 uydu-yılı incelenmesi sonucunda elde edilmiştir. Yazarlar bu olayların güneş ışınlarındaki ağır iyonlara bağlı olarak elektronların çarpması ile transistör yarı iletkenlerinde elektron deliklerinin oluşmasına bağlı olduğunu varsaymışlardır. Güneş ışınında yer alan 100MeV demir atomlarının bu garipliklerden sorumlu olduğu önerilmiştir. Araştırmacıların sunduğu verilerde, hatalı veri oranı dört yılda bir hata oluşması gibi çok cüzi miktarlarda olduğu için, makalelerinde bu durumun ciddi bir problem olmadığını açıklamışlardır. Bahsi geçen 100MeV demir parçacıkları hala uydulardaki hataların kaynağı olmakla birlikte bu parçacıklar atmosfere işlememektedir ve karadaki elektroniğin sonucu değildir.

1978 yılında, enerji parçacıklarının çarpmasıyla deniz seviyesinde gerçekleşen geçici hatanın delili Intelden May ve Woods tarafından bir makalede sunulmuştur [12]. Bu makale Intel'in 16 Kb DRAM'e sahip 2107 serisinde oluşan hataların bir sonucu

olarak yayınlanmıştır. Sorunun bellek birimlerindeki materyallerin radyoaktiflik izleri olduğu keşfedilmiştir.

1970lerde büyük ölçekli tümleşik devrelerin seramik paketlemesinde hızlı bir artış olmasıyla, uranyum madenlerinden geçen Green River'ın üzerine inşa edilen yeni fabrikanın kullandığı su, seramik tümleşik devre paketlerini etkileyen, yüksek düzeyde radyoaktif madde içermektedir. Yaşanan güvenilirlik sorunlarıyla geçici çöküşlerin farkına varılmasıyla ve May ve Woods'un makalesinin etkisiyle IBM geçici çöküşler üzerine çalışan kendi çalışma kolunu kurmuştur. Bu çalışma kolu alfa parçacıklarının aslında IBM'in güvenilirlik sorununun bir kaynağı olduğunu belirlemiştir.

1978 yılında IBM'den Ziegler alfa parçacıkları geçici çökmelere sebep oluyorsa, deniz seviyesinde kozmik ışın akımlarında ağır iyonlar ya da alfa parçacıkları olmasa da, kozmik ışınların da benzer sonuçlar doğurabileceğini düşünmüştür. Ziegler, Yale Üniversitesinde nükleer fizikçi olan W.Lanford ile kozmik ışınların her bir bileşeninin tümleşik devrelerle nasıl tepkimeye girdiği konusunda bir yıl çalışmışlardır. Uydu seviyesinde bulunan ve dünya atmosferinden geçemeyen kozmik parçacıklar ve güneş enerjisindeki parçacıklar hakkında yayınlar bulmuşlardır. Sadece ortalama enerjisi 2GeV ve 0,2/cm^2-s'den fazla olan gökadalar arası parçacıklar dünya atmosferine nüfuz edebilmektedir. Ancak bu parçacıklar incelendiğinde hiçbirisinin deniz seviyesine erişemediği, atmosfer atomlarıyla tepkimeye girerek yeni farklı parçacıklar ürettiği görülmektedir. Deniz seviyesine gelenler altıncı seviye parçacıklardır ve orijinal parçacık içermemektedirler. Bu kademeli parçacıklar sadece proton, nötron ve elektronların kararlı parçacıklarının yanında pions ve mouns gibi geçiş parçacıklarını da içeren her türlü yabancı parçacıkları da içermektedir.

1979'da yayınlanan makale bozulmaya neden olan deniz seviyesindeki kozmik ışınlar hakkında yapılan ilk ayrıntılı mekanizmadır. Deniz seviyesindeki parçacıkların çoğunu ve silikonla etkileşimini her parçacığın akımını birleştirerek incelemişlerdir. Büyük ölçekli tümleşik devrelerin bozulmasına neden olan çeşitli parçacıkların birbirleriyle karmaşık etkileşimlerinde önemli olan etkenlerin yalıtılıp yalıtılmayacağını anlamaya çalışmışlardır. Ziegler ve Landford yaptıkları

çalışmaların sonucunda, 64Mb belleğe sahip olacak olan yeni bilgisayarlarda, 64Mb'lık bellek alanında günde 1 geçici hata oluşacağını düşünmüşlerdir.

1979ların sonlarında Aeronspace Co.'dan Kolasinski ve arkadaşları, Binder ve arkadaşlarının varsayımları üzerine, ağır iyonlar tarafından bombardıman edilen uydulardaki geçici hatalar üzerine deneysel çalışmalar yürütmüşlerdir Çeşitli yongaları inceleyip geçici ve kalıcı hataların her ikisinde de çapraz kesişmeler bulmuşlardır (cross section). Binder'in makalesinde olduğu gibi sadece silikonlardaki iyonlar tarafından doğrudan enerji kaybının kayda değer mekanizma olduğunu farz ettikleri için H ve He gibi ışık iyonlarını önemsememişlerdir.

Yine 1979 yılında Guenzer ve arkadaşları proton ışınları kullandıklarında gözlemledikleri bozulmanın nükleer reaksiyonlardan kaynaklandığını keşfetmişlerdir. Bu makale "geçici bozulma" kavramının yerine terminolojiyi bir bitlik hata ("single-event upset") kavramı ile tanıştırdığı için de önemlidir [13].

1980 yılında Hitachi bazı çift kutuplu RAM'lerinin alfa parçacıkların bombardımanı altında bozulduğunu duyurmuştur. Bu durum çift kutuplu devrelerin de radyoaktif etkilerde hata verebileceğini anlamak açısından önemlidir.

T. O'Gorman, 288 Kb'lık 248 yonga içeren toplam 71 Mb'lık taşınabilir deneme cihazı tasarlamıştır. Ardından 3 yıl boyunca geçici hata ölçümlerini yer seviyesinden aşağıda, deniz seviyesinde ve 2 millik yükseklikte olmak üzere farklı yükseklik seviyelerinde gerçekleştirmiştir. Yaptığı deneyler sonucunda, kozmik ışınların şiddetinin yükseklere çıkıldıkça artmasına bağlı olarak, geçici hata oranının yerden 2 mil yukarıda 10 katına çıktığı görülmüştür.

1983 yılında J.F.Dicello ve Clarkson Üniversitesi tarafından meta kararlı parçacıklar üzerinde yapılan deneyler pions'un (20pslik ömür) büyük ölçekli tümleşik devre bileşenleri üzerinde geçici hatalara sebep olabileceğini göstermiştir. Bu deneyden sonra radyasyon oluşturan tüm parçacıkların belirli bir olasılıkla hataya sebep olduğu anlaşılmıştır.

1984'te IBM tarafından yükseklik etkisi hakkında yapılan ilk geniş çaplı araştırma, 2600 feet yükseklikteki ön bellek modülünde, deniz seviyesindeki aynı modüle göre iki katı daha fazla hata oluştuğunu göstermiştir. Bu çalışma IBM'e tamir edilmesi için geri gönderilen bellek birimlerinin en çok Denver bölgesinden gelmesi üzerine yapılmıştır.

Denver'da daha sonra gerçekleştirilen yoğun çalışmalarda, bazı eş zamanlı gerçekleşen birden fazla hata tespit edilmiştir. Bu hatalar daha önce deniz seviyesinde yapılan testlerde hiç gözlemlenmemiştir. Birbirine yakın bitlerde hata gerçekleşmesi durumuna karşı önlem almak için yeni bir mekanizma gerektiği görülmüştür. Denver'da kozmik ışınların şiddeti, deniz seviyesinden çok daha fazla olduğu için daha geniş gürültüye neden olmaktadır.

1984 yılında iki kutuplu bellek yongalarının kozmik ışın parçacıklarına karşı duyarlılığı denenmiştir. Deneyler, her biri 72 çift kutuplu bellek yongası içeren 2 önbellek modülü ile yapılmıştır. İlk bellek modülü radyasyon mağarasında 1 hafta boyunca çalışmaya bırakılmış ve bu sürede hiç hata ile karşılaşılmamıştır. Çalışma süresi boyunca hiç radyasyon bulunmamaktadır. Ardından mağaraya, yonganın içinden geçecek şekilde hedeflendirilmiş seyreltilmiş nötron bombardımanı uygulanmıştır. 20 sn. içerisinde ön bellek eşlik biti kaydedilmiş ve birkaç dakika içerisinde normal çalışmaya devam edebilmiştir. 40. sn. de yeni bir hata alınmış ve hata kurtarılmaya çalışılırken 42. sn. de yeniden hata alınmıştır.

1986 yılında IBM, "Kozmik Işınlar için IBM Devrelerinin İvmelendirilmiş Testi"ni bitirdiğini bildirmiştir.

1986 yılında çift kutuplu SRAM yongaları için geçici hata alan deneyi yapılmıştır. Alan deneycileri yüzlerce yongayı deniz seviyesinde denedikten sonra çeşitli yüksekliklerde denemektedirler. Bu deney, yongaların kesin çapraz kesişimlerini kurmakla birlikte farklı yeryüzü yüksekliklerinde beklenen değişim değerlerini de kurmaktadır.

1965–1968 yılları arasında Kanada Atom Enerji Kurumu yükseklik ve alçaklığın kozmik ışınlar üzerindeki etkisini incelemek için taşınabilir bir kozmik ışın laboratuarı inşa etmiştir. Bu laboratuar Kuzey Amerika'da pek çok yerde işlemler yapmış ve Havai'deki Haleakela dağında son bulan karşılaştırmalı değerler üretmişti. Araştırmalar sırasında bütün yazarların öldüğü ya da emekliye ayrıldığı öğrenilmiş ve yaklaşık 17-tonluk ve milyon dolarlık aracın yerinin bilinmediği görülmüştür. IBM Havai'de yapılan soruşturmalar sonunda bir saha mühendisi olan D.Fujimoto birkaç ay iz sürmenin sonunda aracı, Maui'nin terkedilmiş bir bölümünde bulmuştur. Araç üzerindeki Wailuku volkanından kaynaklandığı sanılan kurşun izi büyüklüğündeki delikler dışında iyi durumdadır ve IBM'in kozmik ışın deneylerinin yapıldığı merkeze taşınmıştır.

Geçici hatalar konusunda tarihte yaşanan en ilginç olay Hera sorunu olarak bilinir. 1987 yılının ilk yarısında büyük ölçekli tümleşik devre bellek birimlerindeki sorun oranı beklenenin 20 katı olmasına karşın Avrupa'da üretilen büyük ölçekli tümleşik devre bellekleri bu derece aşırı hata göstermemekteydi. 1987 Nisanında tasarımda yapılan değişiklikte yeni bellek birimine Hera adı verildiği için bu problem "Hera Sorunu" olarak bilinmektedir. Yapılan araştırmalarda ilk adım hatalı yongaların incelenmesiyle atılmıştır ve hatalı yongaların fark edilir derecede radyoaktivite içerdiği görülmüştür. Bu radyoaktifliğin kaynağı uzun süre araştırılmış ve bir miktar şansın da yardımıyla açılmamış bir nitrik asit şişesinin radyoaktif olduğu fark edilmiştir. Nitrik asidin tedarikçi firması incelendiğinde radyoaktifliğin şişe temizleme makinesinden bulaştırıldığı anlaşılmıştır. Temizleyici makine radyoaktif Po210 materyalini şişelerin içindeki elektrostatik tozları temizlemek için kullanılan hava memelerinin iyonlanmasında kullanmaktadır. Bazı hava memeleri radyoaktivite sızdırdığı için şişelerin bir kısmı rasgele olarak radyoaktiflenmekteydi. Bu olay 100 şişede sadece birkaç tanesinde gerçekleşmektedir. Olayın kaynağı anlaşıldıktan sonra eski Hera yongaları toplanıp, temiz olanları ile değiştirilmiştir.

3.3. Geçici Hatanın Geleceği

Aşağıda belirtilen sebeplerden ötürü geçici hata oranının artış eğiliminde olduğu görülmektedir [10,11].

- Silikon geometrisi küçüldükçe sistemler daha karmaşık hale gelmektedir. Basitçe, eğer bir tümleşik devrede daha çok bellek ve mantık devresi varsa, parçacığın çarpması için daha çok alan ve geçici hatanın oluşabilmesi için daha çok ihtimal vardır.
- Daha düşük gerilim değerlerinin ve daha küçük sığaların kullanılması, bir parçacık çarptığında 1 değerini 0 ya da tam tersi şekilde hücrenin durumunun değişme ihtimalini artırır.
- Elektronların rasgele şekilde boşalması da birbirine bağlı mantıksal blokların zaman ihlaline neden olabilir. Yani daha hızlı ve karmaşık tasarımlar zaman ihlaline karşı daha duyarlıdır.
- Küçülen geometrilerin birden çok hatanın aynı zamanda oluşmasını hızlandırdığını gösteren deliller de mevcuttur.

3.4. Bellek Birimlerinde Geçici Hataların Ele Alınışı

Geçici hatalar bellek alanları ve mantıksal devrelerin her ikisi için de sorun oluşturmaktadır. Geçici hatanın tespit edilmesi ve onarılması için farklı yöntemler mevcuttur. İlerleyen bölümde bu yöntemlerden birkaç tanesi anlatılacaktır [10].

3.4.1. Eşlik Biti Denetimi

Eşlik biti denetiminde her 32 bitlik sözcüğün sonuna 1 bit eşlik biti eklenir. Eşlik biti tek sayıda bitte hata oluştuğunda tespit eder ancak aynı anda çift sayıda bitte hata oluşursa hatayı tespit edemez. Ayrıca eşlik biti tek başına hatanın düzeltilmesi için kullanılamaz. Günümüzde geçici hata, tek bit hata olarak anlaşıldığı için kabul edilebilir ve etkin bir yöntem olarak görülmektedir.

Gerçekleştirim açısından bakıldığında, her bir bayt sonuna 1 bitlik eşlik biti eklendiği durumda gerçek SRAM alanından %12,5 oranında kayıp gerçekleşir.

Önbellek komutlarında bulunan bir hatanın düzeltilmesi için boru hattının alt ucunun silinerek gerekli ön bellek alanlarının temizlenmesi yeterlidir.

Birbirine bağlı bellek alanlarında oluşabilecek geçici hataların düzeltilmesi ise biraz daha zahmetli olabilir. Bunun için bir yaklaşım, geçici hata aykırı durumunun tetiklenmesi ile gerekli hata düzeltici kodun çalıştırılarak birbirine bağlı bellek alanlarının tazelenmesidir. Ancak bu yöntemin bir takım kötü yanları vardır. Hata düzeltici kodun kendisi de hatanın kaynağı olabilir. Aynı zamanda, hızlı kesilmelerin sınırlı gecikmesi hatayı düzelten kodlardaki gecikmeler yüzünden artabilir.

Eşlik biti hatalı olan önbellek satırlarında "kirli bit" kurulmuşsa bu bitin kaybedilmeden satırın ana bellekten temizlenmesi mümkün değildir. Bu durum doğrudan yazmalı (write-through) ve maliyeti yüksek önbellek kullanılarak çözümlenebilir. Bazı sistemlerde ise beklenen geçici hata oranı çok düşük olduğu için, tespit edilen hatanın düzeltilmesinde sistemin baştan başlatılması bu sistemler için kabul edilebilir bir yöntemdir.

3.4.2. Hamming Kodları

Tek bir parçacığın çarpması birbirine çok yakın konumda olan iki bitin değerini değiştirmesine neden olabilmektedir. Bu durum çift bitlik hata olarak adlandırılır [1].

Hamming kodları tek-bitlik hataların düzeltilmesine çift-bitlik hataların ise tespit edilmesine olanak verdikleri için SEC/DED (Single Error Correction/Double Error Detection) olarak bilinirler. Her bir bayt için 4 bitlik ekleme yapıldığı için bellek alanını %50 artırır. Hamming kodu eklenmiş bir komut kodunda bir bitlik bir hata oluştuğunda, komutun hangi adımda olduğunun bir önemi olmaksızın oluşan hata düzeltilebilmektedir. Eğer düzeltme çözümleme ya da yazma aşamasında gerçekleşirse, getirilen kod ya da veri, boru hattının standart işlemleriyle paralel olarak doğrulanabilir ya da düzeltilebilir. Eğer bir hata bulunduysa, boru hattı o noktada durdurulur ve düzeltilen veri boru hattının girişinden itibaren yeniden ilerletilir. Bu durum sadece 1 saat vuruşu gecikmeye mal olmaktadır.

Çift hatalar için düzeltme fazladan mantık devresi gerektirdiği için biraz daha karmaşıktır.

3.4.3. Fazlalık Yöntemi

Çift fazlalık olarak bilinen, her bir bellek alanının fazladan bir kopyasının tutulması yöntemiyle, oluşan her türlü hata tespit edilebilmektedir. Üçlü fazlalık yönteminde ise bellek alanının 2 tane fazla kopyası tutulur ve böylece oluşan geçici hata düzeltilebilir.

Bu yöntemde, bir değer saklama alanına yazılırken, bütün kopyalara da yazılmalıdır. Bir değer okunduğunda ise, her bir kopyadan gelen çıktı hatanın tespit edilmesi için karşılaştırılır. Bir hata ile karşılaşıldığında üçüncü bellek alanı hangi kopyanın doğru olduğunun tespit edilmesi için kullanılır.

Bu yöntem basit olduğu, başarımı yüksek olduğu ve çok sayıda hatayı ele alabildiği için yaralıdır. Kötü tarafı ise bellek alanını ikiye ya da üçe katlaması ve uygulanması için denetim mantık devreleri gerektirmesidir.

3.5. İşlemci Bitlerinin Sınıflandırılması

Weaver ve arkadaşları tarafından, mikroişlemci üzerinde yer alan bir bite parçacık çarpması sonucunda bitte geçici hata oluştuğu varsayıldığında, bu bitin mantık olarak nasıl sınıflandırılacağı anlatılmıştır [3].

Herhangi bir bitte geçici hata oluştuğunda eğer oluşan hata tespit edilen ve düzeltilen türden ise işlemci herhangi bir sorunla karşılaşmadan çalışmasına devam edecektir.

Hatanın sadece tespit edilebildiği durumda “Bulunan Düzeltilemeyen Hata” (BDH) oluşur. Oluşan bu hata programın çıktısını etkiliyorsa doğru-BDH aksi takdirde yanlış-BDH oluşur.

Okunan bit üzerinde herhangi bir koruma bulunmadığı durumda ise eğer bit program çıktısını etkilemiyorsa yine işlemci bir sorunla karşılaşmadan çalışmasına devam

edecektir. Aksi takdirde hata tespit edilemeden “Gizli Veri Bozulması ” (GVB) yaşanır ve bu durumlar içerisinde en tehlikeli durum budur.

Eğer bir bit, programın son çıktısını etkileyen bir değer içeriyorsa DÇG (Doğru Çalışma için Gerekli – ACE) olarak adlandırılır

3.5.1. DÇG Olmayan Bit Kaynakları

Programın son çıktısını etkilemeyen DÇG olmayan bitlerin görüldüğü yerler Mukherjee ve arkadaşları tarafından belirtilmiştir [14]. İlerleyen bölümlerde bu kaynaklar anlatılacaktır.

3.5.1.1. İşsiz ya da Geçersiz Durumlar:

Mikromimaride bir veri ya da durumun işsiz olduğu ya da geçersiz bilgi içerdiği pek çok an vardır. Bu veri ve durum bitleri DÇG değildir. Kontrol bitleri ise daima DÇG kabul edilir. Çünkü denetim bitlerinde gerçekleşen bir bozulma geçersiz bir veriyi geçerli hale getirebilir.

3.5.1.2. Yanlış Tahmin Edilen Durumlar:

Çağdaş mikroişlemcilerde başarımı artırmak için, daha sonra yanlış ya da doğruluğu anlaşılacak olan işlemler gerçekleştirilir. Örneğin dallanmanın yanlış tahmin edilmesi gibi, yanlış tahmin üzerine yürütülen komutlardaki bitler DÇG değildir.

3.5.1.3. Tahmin Durumları:

Mikroişlemcilerde daha sonrasında ihtiyaç duyulabilecek işlemlerin tahmin edilmesi için kullanılan yapılarda oluşan bir hata başarımı düşürse de programın çalışmasında hataya neden olmaz. Bunun için bu bitler DÇG değildir.

3.5.1.4. DÇG Sonrası Durumu:

Bir DÇG bit en son kullanımından sonra DÇG olmaktan çıkar, yani bit ölür. Örneğin dinamik olarak bir komutun komut kuyruğundan en son yayınlandığı andan itibaren işlemci komutun tekrar yayımlanmayacağını anlayana kadar komut kuyruğunda kalabilir. Bu anda komutun bitleri DÇG değildir

3.5.1.5. NOP Komutları:

Fash ve arkadaşlarının [7] hesapladığına göre işlemci içerisindeki komutların %10'u NOPtur ve işlemcinin mimari durumları üzerine hiçbir etkileri yoktur. Bu komutlar içerisinde komutun NOP olduğunu gösteren yer dışında kalanlar DÇG olmayan bitlerdir.

3.5.1.6. Başarımı Artıran Komutlar:

Birçok çağdaş mikroişlemci, başarımı artırmak için fazladan komut kullanmaktadır. Örneğin daha sonra ihtiyaç duyulacak değerlerin ön belleğe getirilmesi için araya eklenmiş olan komutlarda, işlem kodu kısmında olmamak şartıyla, oluşan bir hata program çıktısını etkilemediği için bu bitler DÇG değildir. Oluşan hata ihtiyaç duyulmayan bir komutun getirilmesine neden olabilir ancak bu durum sadece başarımı düşürecektir. Diğer taraftan eklenen bu komutların işlem kodunda gerçekleşen bir hata, geçici hataya sebep olabileceği için işlem kodunun bitleri DÇG'dir.

3.5.1.7. Yanlış Doğrulanmış Komutlar:

IA64 gibi mimariler, komutların işletimini yazmaçların doğrulanmasına göre sınırlandırır. Doğrulanan yazmaç geçerli ise, komut sonlandırılır. Eğer doğrulanan yazmaç yanlışsa komutun sonucu iptal edilir. Sonuç olarak geçersiz olduğu anlaşılan

bu tür komutlarda yazmaçları doğrulamaya yarayan bitlerin dışındaki bitler DÇG değildir.

3.5.1.8. Devingen Ölü Komutlar:

Sonucu kullanılmayan komutlar devingen ölü komutlardır. En basit şekliyle sonucu diğer komutlar tarafından okunmayan komutlar Birinci seviyede Devingen Ölü (BDÖ) komut olarak adlandırılır. Geçişli Devingen Ölü (GDÖ) komutlar ise ürettiği sonuç sadece BDÖ komutlar ya da diğer GDÖ komutlar tarafından okunan komutlardır. BDÖ ve GDÖ komutlarının işlem kodu ve hedef yazmaç tanımlayan alanlarındaki bitler DÇG'dir ve komuttaki diğer bitler DÇG değildir.

3.5.1.9. Mantıksal Maskeleme:

İşlenenler içerisinde hesaplama zincirinde, hesabın sonucunu etkilemeyen pek çok değer vardır. Bu bitlere mantıksal olarak maskelenmiş denir. Örneğin aşağıdaki kod akışı incelenebilir.

(1) R2 ← R3 OR 0x00FF

(2) R4 ← R2 OR 0xFF00

(3) R3 ← 0

(4) R2 ← 0

(5) output R4

Yukarıdaki işlem döngüsünde R4'ün alt 16 biti R2 ve R3 değerleri ne olursa olsun 0xFFFF olacaktır. Bir işlenendeki herhangi bir bit değeri işlemin sonucunu değiştirmiyorsa, bu bit maskelenmiş olur. Örneğimizde R3'ün 0. bitinden 7.bitine kadar olan alt bitleri ve R2'nin 8. bitinden 15.bitine kadar olan bitleri mantıksal olarak maskelenmiştir. Sonuç olarak bu bitler DÇG değildir.

3.6. İşlemci Güvenilirliğinin Ölçülmesi

GCB ve BDH olarak belirlenen ham hata oranları BSH (Birim Süredeki Hata) ile ifade edilir. Bir BSH, bir milyar saatte gerçekleşen hata miktarıdır. Yonganın toplam BSH oranı, yonga üzerindeki her bir yapının BSH oranlarının toplamıdır.

Benzer şeklide iki çakılma arasındaki ortalama zaman HAOS (Hatalar Arasındaki Ortalama Süre) olarak ifade edilir. HAOS, BSH ile ters orantılıdır ve bir yıldaki HAOS değeri 114,155 BSH (109/(24*365))'dir [1,3,14].

Çakılmaya kadar geçen süre HKGS (Hataya Kadar Geçen Süre) ve çakılmanın telafi süresi HTGS (Hatanın Telafisi için Geçen Süre) olarak ifade edilir. HAOS değeri bu iki değerin toplamı olarak ifade edilebilir.

$$\text{HAOS} = \text{HKGS} + \text{HTGS} \tag{3.1}$$

HAOS, geçici hata oranı hakkında bir ölçüm birimi sağlasa da, hata oranı ve işlemci başarımı arasındaki çelişkiyi çözmemizi sağlamaz. Bunun için yeni bir ölçü birimi olan HAKS (Hatalar Arasında işletilen Komut Sayısı) kullanılmalıdır [3]. HAKS iki çöküş arasında ortalama olarak kaç tane komut işletildiği anlamına gelir. HAKS'nin yüksek olması hatalar arasında daha çok iş yapıldığını gösterir. HAKS ve HKGS arasındaki ilişki denklem (3.2) ile gösterilmiştir.

$$\begin{aligned} HAKS &= \frac{\text{sonlanan komut sayısı}}{\text{gerçekleşen hata sayısı}} \\ &= \frac{\text{sonlanan komut sayısı}}{\dfrac{\text{toplam çalışma zamanı}}{\text{saat sıklığı x HKGS}}} \\ &= \text{ÇBK x saat sıklığı x HKGS} \end{aligned} \tag{3.2}$$

Denklemde geçen ÇBK (Çevrim Başına Komut) değeri, bir saat vuruşunda işletilen ortalama komut sayısı olarak ifade edilmektedir.

Bit başına düşen etkin BSH oranı zaman hassasiyeti, devre seviyesindeki hassasiyet ve mimari hassasiyet gibi farklı hassasiyet oranlarından etkilenmektedir.

Örneğin bir gerilim düzeyine duyarlı mandal, veriyi saklamak yerine veri talep ediyorsa, sakladığı bite zarar gelse de hata ile sonuçlanmayacaktır çünkü saklanan bitin üzerine yeni ve doğru bir değer yazılacaktır. Eğer mandal değeri zamanın %50'sinde yeni veri değeri bekliyorsa, mandalın zaman hassasiyeti %50'dir. Daha basit düşünmek için zaman hassasiyet değerinin zaten cihazın ham yanlışlık oranı ile ilgili olduğu kabul edilmiştir.

Mimari Hassasiyet Çarpanı (MHÇ) [14] saklama hücresinde bir bitin değeri değiştiğinde programın işleyişinin gözlemlenebilir bir hata ile sonlanması ihtimalidir. Bu değer işlemcinin gerçek hata oranı üzerinde önemli bir etkiye sahiptir.

Bir saklama biriminin MHÇ değeri, birimde bir zarar oluştuğunda, zararın program çıktısında hata oluşturacağı zamanların oranıdır. Bu durumda, bir bitlik saklama alanının MHÇ değeri basit şekilde alanın DÇG bit içerdiği zamanın oranıdır.

Bir başka ifadeyle MHÇ, yapıya bir parçacık çarptığında programın çalışmasının hata ile sonlanması ihtimalidir. Örneğin dallanma tahmin bitlerinin MHÇ değeri %0'dır. Çünkü bu bitlerde gerçekleşen bir hata sadece dallanmanın yanlış yönde tahmin edilmesine neden olur. Diğer taraftan program sayacında gerçekleşecek bir bitlik hata yanlış komutun çalıştırılmasına neden olacağı için program sayacının MHÇ değeri %100'dür. MHÇ değeri denklem (3.3) ile hesaplanır.

$$\text{MHÇ} = \frac{\text{Donanın Birimindeki Ortalama DÇG Bit Sayısı}}{\text{Donanım Birmindeki Toplam Bit Sayısı}}$$

$$\text{MHÇ} = \frac{\sum \text{DÇG Bitlerinin Yapıda Bulunduğu Saat Vuruşu Sayısı}}{\text{Donanımdaki Bit Sayısı x Toplam Saat Vuruşu}} \qquad (3.3)$$

Yapılan analizler sonucunda komut kuyruğunun MHÇ'si %28 ve çalışma birimlerinin MHÇ'si ise %9 olduğu görülmüştür [12].

Mikroişlemcinin GVB oranını hesaplamak için ise denklem (3.4) kullanılır [3].

$$\text{GVB Oranı} = \sum_{d=\text{tüm cihazlar}} \text{hata oranı}_d \text{ x } \text{GVB MHÇ}_d \tag{3.4}$$

Bir cihazın GVB MHÇ'si, bu cihaza bir parçacık çarptığında, programın son çıktısında hata oluşma ihtimalini ifade eder. Bir saklama hücresinin GVB MHÇ'si o hücrenin DÇG bit içerdiği saat vuruşlarının oranıdır.

Mikroişlemcinin BDH oranı denklem (3.5) ile hesaplanır [3].

$$\text{BDH Oranı} = \sum_{d=\text{tüm cihazlar}} \text{hata oranı}_d \text{ x BDH MHÇ}_d \tag{3.5}$$

BDH MHÇ, bir parçacık çarptığında BDH oluşturma ihtimalidir. Hataya karşı korunan bir cihazdaki toplam BDH MHÇ, en az cihazın korumasız versiyonunun GCB MHÇ'si kadardır ve genellikle de daha fazladır. Doğru-BDH MHÇ oranı, hatadan korunmak için herhangi bir önlem alınmadığındaki GVB MHÇ ile aynı olacaktır.

3.7. İşlemci Güvenilirliğini Artırmak İçin Yapılan Çalışmalar

Geçici hata sorununun üstesinden gelmek için pek çok yöntem mevcuttur. Ancak bu yöntemlerin hepsi başarım, güç tüketimi, yonga genişliği ya da tasarım zamanından kaybettirmektedir. Sonuç olarak tasarımcı bu yöntemlerin kazandırdıklarını ve kaybettirdiklerini iyi tartmalıdır. Çünkü hataya karşı korumasız bir mikroişlemci güvenilirlik açısından çok zayıf olabiliyorken başarım ve maliyet açısından çok güçlü olabilmektedir [3,10].

Christopher Weaver ve arkadaşları, geçici hata oranını azaltmak için iki teknik önermişlerdir [3]. Bu tekniklerden ilkinde, uzun bekleme süreleri sırasında, komut kuyruğunu geçerli komutlar yerine geçersiz kayıtlarla doldurarak, alfa ya da nötron parçacıklarının geçerli bir alana çarpma ihtimali azaltılır. Amaç komutların komut kuyruğunda uzun süre ihtiyaç duyulmadan beklemesini engellemek olduğu için, seçilen tetikleyicinin kuyruğa sokulmuş olan komutun uzun süredir orada bulunduğu anlamına gelmesi gerekmektedir. Bu sebeple bellek komutlarının aradıkları verileri önbellekte bulamamaları durumu tetikleyici olarak seçilmiştir. İşlemci aranan veriyi önbellekte bulamadığında, komut kuyruğunda yer alan komut silinip yeniden ana bellekten getirilebilir ve bu işleme komutun ezilmesi denir. Bu işlem sırasında yalnızca önbellekteki verinin bulunamamasına neden olan komuttan daha genç olanları ezilirler. Diğer taraftan getirme adımı daraltılarak yeni komutların komut kuyruğuna eklenmesi de önlenir.

Önerilen ikinci yöntem ise yanlış tespit edilen hataların azaltılmasını amaçlar. Hatadan etkilenen komut ya da veri için doğrudan hata sinyali vermek yerine sadece "olası hata" olarak işaretlenir. Mikroişlemci donanımının farklı parçaları arasındaki hata bilgisini yaymak için, "olası hata" anlamına gelen π-bit ortaya konmuştur. π-bit mantıksal olarak her bir komuta, çözümleme aşamasından emeklilik aşamasına kadar bağlıdır. İlk aşamada komutun sağlam olduğunu gösterecek şekilde π bit temizlenir. Komut kuyruğu komutu aldığında π biti de komutla beraber alır. Eşlik biti ya da benzer bir mekanizma ile herhangi bir hata tespit edildiği durumda, komut kuyruğu aykırı durum oluşmasına neden olacak şekilde hata sinyali vermek yerine sadece ilgili komutun π-bitini kurar. Daha sonra komut yayınlanır ve boru hattını takip etmeye devam eder. Sonlandırma aşamasına gelindiğinde, komutun doğru yolda olup olmadığı anlaşılır. Eğer yanlış yolda ise ve π-bitin değeri bir ise yanlış-BDH olduğu anlaşılır ve π-bit ihmal edilir. Komut doğru yolda ise, makinenin sonlandırma aşamasında hatayı denetleme seçimi vardır. π-bite çarpan bir parçacığın da yanlış-BDH' ya neden olacağı gözden kaçmamalıdır.

Anti-π bit olarak adlandırılan bir diğer bit ise, çözümleme aşamasında her bir komuta bağlı bir bittir. Sadece nötr türdeki komutlar için kurulur, diğer komutlar için ise kurulmaz. Donanımsal olarak emeklilik aşamasında komutun tekrar çözümlenmesi ile böyle bir bite gerek kalmayabilir.

π-bit kullanılarak hatanın tespit edilmesi için gerekli yöntemler şu şekildedir.

Bütün komutların logları emeklilik aşamasından sonra SSHT (Sonlandırma Sonrası Hata Takip) belleğinde tutulur. Özellikle bir SSHT satırı, bir komut ve π -bitini içerir. Eğer SSHT belleği dolarsa en yaşlı komut bellekten çıkarılır. Bu aşamada tahliye edilen komutun π-biti denetlenir. Eğer SSHT belleği başka bir komutun veri alanına yazması ya da araya girip düzeltmek için okuması gibi durumlarla komutun BDÖ olduğunu tespit edemezse bu komutta bir hata olduğunu bildirir.

Bir komutta yer alan π bit hedef yazmacına da kopyalanır. Böylece bu yazmaçtan okuma yapıldığında hata sinyali verilebilir. BDÖ komutlarının %100'ü bu yöntemle kurtarılmış olur ancak hatanın hangi komuttan kaynaklandığı bulunamaz.

GDÖ komutların izlenebilmesinin en kolay yolu işlemcinin sadece bellek alanlarıyla ya da giriş çıkış birimleriyle iletişime girdiğinde hata verilmesidir. Bu durumda işlemci içerisindeki bellek ve ön bellek alanlarında π bit yer almaz ancak diğer tüm birimlerde π bit bulunur.

Geçici hataların programı istenmeyen şekilde sonlandırmasına karşı önbellekte alınan bir önlem, SECDEC HTK (Hata Tespit Kodu) ile korunan bellek satırlarının ovalanması yöntemidir [1]. Önbellekte bulunan bitler okunur ve tek bitlik bir hata varsa hata bulunup düzeltilir. Yeni HTK değeri hesaplanarak bitler yeniden belleğe yazılır. Yapılan çalışmalar sonucunda, 10 megabayttan küçük önbellek alanların çok düşük çift bitlik hata üretmesi sebebiyle ovalanmasına gerek olmadığı görülmüştür. Diğer taraftan 100 MB'dan büyük bellek alanlarında ovalama önerilmektedir.

BÖLÜM 4

4. ETİKETLERDE GEÇİCİ HATA TESPİT YÖNTEMLERİ

4.1. İşlenenlerin Kıyaslanması ile Geçici Hata Tespiti

Mikroişlemcilerde yer alan komutlar en fazla iki işlenen içerebilmektedir ve bazı durumlarda bu iki işlenen birbirinin aynı olabilmektedir. Öte yandan, komutlar yayın kuyruğuna atılmadan önce kullandığı işlenenlerinin birisi ya da her ikisi birden hazır olabilmektedir. Bahsedilen bu ölçütler göz önünde bulundurulduğunda, komutlar şu şekilde sınıflandırılabilir.

- Hiç işlenen içermeyen komutlar: *0Kullanan*
- Bir işlenen içerip yayın kuyruğunda bu işlenenin hazır olmasını bekleyen komutlar:*1Kullanan0Hazır*
- Bir işlenen içeren ve yayın kuyruğuna girmeden önce bu işleneni zaten hazır olan komutlar: *1Kullanan1Hazır*
- İki işlenen içeren ve iki işleneni birbirinin aynı olan komutlar: *2Kullanan2Aynı*
- Birbirinden farklı iki işlenen içeren ve hiçbir işleneni önceden hazır olmayan komutlar: *2Kullanan0Hazır*
- İki işlenen içeren ve bu işlenenlerin birisinin hazır olduğu komutlar: *2Kullanan1Hazır*
- Birbirinden farklı iki işlenen içeren ve bu işlenenlerin ikisinin de komut yayın kuyruğuna girmeden önce hazır olduğu komutlar: *2Kullanan2Hazır*

Yukarıdaki sınıflandırmada iki işlenen içeren komutlar için öncelikle komutun işlenenlerinin aynı olup olmadığına bakılmaktadır. İki işlenenli komutlar için, işlenenlerin hazır olmasına göre yapılan sınıflandırmanın, sadece farklı işlenenler için yapılması yeterlidir. Çünkü yayın kuyruğunda yer alan etiketlerin zarar görmelerine karşı güvenilirliği artırmak için önerilen yöntemlerde, eğer iki işlenen

birbirinin aynı ise basit bir yöntemle etiketlerde oluşan geçici hatalar tespit edilebilmektedir. Farklı işlenenler için kullanılacak yapılar biraz daha karmaşıktır.

Yayın kuyruğunda yer alan komutların çalışmaya hazır hale gelebilmeleri için, her saat vuruşunda işletimi tamamlanan komutlar, değerlerini ürettikleri yazmacın etiketini yayın kuyruğundaki tüm komutlara yayımlarlar. Kuyrukta yer alan tüm komutlar da kendi işlenenlerinin etiketleri ile gelen etiketi karşılaştırırlar. Bu karşılaştırmanın yapılması için mikroişlemci üzerinde pek çok karşılaştırıcı devre mevcuttur. Bu karşılaştırma işlemlerinin sonucunda büyük çoğunlukla "farklıdır" sonucu elde edildiği için karşılaştırıcı devreler verimsiz kullanılmaktadır.

Bu çalışmada karşılaştırıcı devrelerin verimsiz kullanılmasını önlemek için bu devrelerin geçici hataların tespitinde kullanılabileceği önerilmektedir. İlerleyen kısımlarda sınıflandırılması yapılan komutlarda oluşabilecek geçici hataların tespit edilme yöntemleri ayrıntılı olarak anlatılmaktadır.

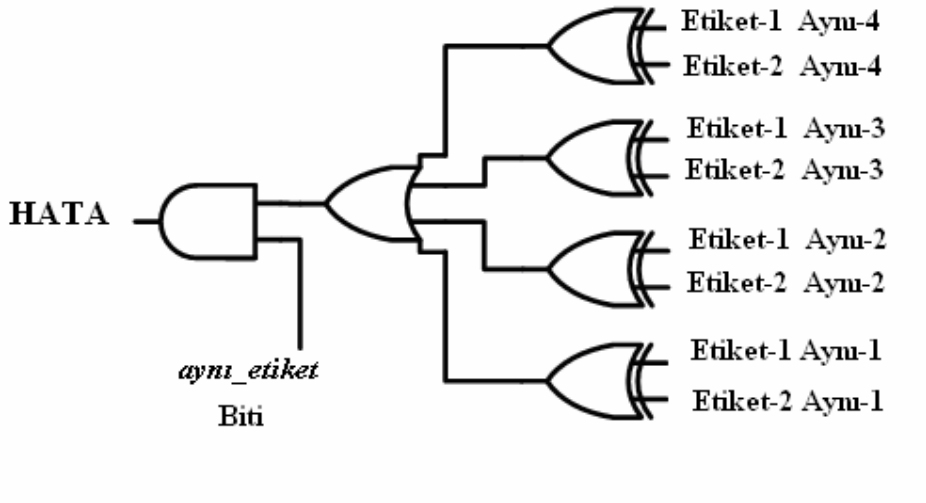
4.1.1. Bir İşlenenli Komutlarda Geçici Hata Tespiti

Mikroişlemci içerisinde yer alan komutların büyük bir çoğunluğu sadece bir işlenen içermektedir. Bu durumda yayın kuyruğundaki her bir komut için ayrılan satırlara bakıldığında ikinci işlenen için ayrılan alan ve bu alana ait karşılaştırıcı devreler gereksiz işlem yapmakta, komut yayınlanana kadar sadece birinci işlenenin etiket ve karşılaştırıcıları kullanılmaktadır.

Bir işlenenli komutlarda geçici hataların tespiti için, ikinci işlenen için ayrılmış ve kullanılmayan alana birinci işlenenin kopyalanması önerilmiştir. Bu kopyalama yapıldığında, karşılaştırmalar sonucunda her iki işlenen de aynı sonucu üretmelidir. İki karşılaştırıcının sonucu da aynı anda isabetli ya da aynı anda isabetsiz olmalıdır. İki işlenen etiketinin birbirinden farklı sonuçlar üretmesi durumunda, etiketlerde bir geçici hata oluştuğu anlaşılır. Bunun üzerine boru hattında bulunan komutların hatalı komuta kadar olanı boşaltılır ve komutlar yeniden boru hattına getirilmeye başlanır. Bu işlem sonucunda boru hattının yeniden dolması için gerekli zaman kaybedilmiş

olur ancak geçici hata tespit edildiği için, mikroişlemci çakılmadan güvenli çalışmasına devam edebilmektedir.

Bahsedilen mekanizmanın gerçekleştirilebilmesi için donanıma bir takım eklentiler yapılmalıdır. İlk olarak yayın kuyruğunda yer alan her komut için, komutun işlenen etiketlerinin aynı olduğunu gösteren "aynı_etiket" biti yerleştirilir. Komut kullanacağı işlenen sayısını işlem kodundan (*opcode*) bilindiğinden, bu bitin eklenmesi komutların işletiminde herhangi bir değişiklik yapmamaktadır. Her iki işlenen etiketinin de birbirinin aynı olduğu durumlarda etiketlerden birinde bir hata olursa bu hatanın yakalanması gerekmektedir.



Şekil 4.1 "aynı_etiket" Biti Kullanılarak Geçici Hatanın Tespit Devresi

Her bir saat vuruşunda karşılaştırıcı devreler *aynı* ya da *aynı değil* sinyali ürettiklerinde, komutta yer alan işlenen etiketinde ya da komutu uyandırma devrelerinde yer alan bir hatayı tespit eden devre şeması Şekil 4.1 de gösterilmiştir. Devre şeması, bir seferde en fazla dört tane komutun işletilip geri yazılmasına olanak veren 4 yollu bir işlemci için verilmiştir.

4 yollu bir makinede, yayın kuyruğundaki her işlenen etiketinde 4'er tane karşılaştırıcı devre yer alır. Her saat vuruşunda sonucu üretilen komutların sonuç yazmaçlarının etiketleri karşılaştırıcı devrelere gelir. Gelen etiket değerleri

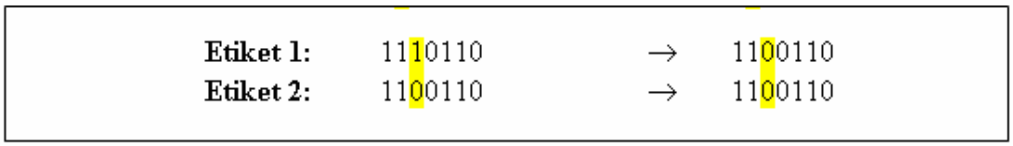
işlenenlerle karşılaştırılıp, sonuçları ÖZEL VEYA kapılarına gönderilir. Örneğin, Şekil 4.1'de gösterilen "Etiket-1 Aynı-2" değeri, komutun birinci işleneni ile ikinci kapıdan yayımlanan ikinci etiket değerinin karşılaştırılmasının sonucudur. ÖZEL VEYA işlemleri sonuçlarında, her iki işlenende yer alan karşılaştırıcı devreler aynı sonucu ürettiğinde ÖZEL VEYA kapısının sonucu sıfır olur. Diğer taraftan karşılaştırıcı devreler iki işlenende farklı sonuçlar ürettiğinde ÖZEL VEYA kapısının sonucu bir olur. Üretilen sıfır ve bir değerlerinin bir VEYA kapısından geçmesiyle, herhangi bir karşılaştırmada, iki işlenen karşılaşması birbirinden farklı sonuç ürettiyse bir değeri elde edilir. Bütün karşılaştırmalarda işlenenler birbirinin aynı sonucu verdiyse sıfır değeri üretilir. Son olarak elde edilen bu değer *aynı_etiket* biti ile VE kapısından geçirilir. Komutta her iki etiket birbirinin aynı olmasına rağmen, herhangi bir karşılaştırıcı için iki etiket birbirinden farklı bir sonuç üretiyorsa etiketlerde bir HATA oluştuğu anlaşılır. Aksi takdirde HATA üretilmez ve işlemci normal çalışmasına devam eder. "ayni_etiket" biti ile komut işlenenlerinin ürettiği karşılaştırma değerlerine göre geçici hata oluşması mantık çizelgesi Çizelge 4.1.'de gösterilmiştir.

Çizelge 4.1. HATA Oluşma Mantık Çizelgesi

aynı_etiket	**Karşılaştırma–1**	**Karşılaştırma–2**	**Geçici Hata**
0	**0**	**0**	**0**
0	**0**	**1**	**0**
0	**1**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**0**	**0**
1	**0**	**1**	**1**
1	**1**	**0**	**1**
1	**1**	**1**	**0**

Yayın kuyruğunda herhangi bir etiket değerinde hata oluştuğu anlaşıldığında, yayın kuyruğundaki ilgili komut satırına kadar olan tüm komutlar boru hattından boşaltılırlar ve ana bellekten tekrar çekilirler.

Her iki işlenenin birbirinden farklı olduğu durumda, bir işlenenli komutlar için geçerli olmamakla birlikte, "aynı_etiket" bitinin işleyişi açısından istisnai bir durum mevcuttur. İki etiketin birbirinden farklı olduğu durumda "aynı_etiket" biti kurulmaz ve iki etiketin de karşılaştırmalar sonucunda aynı anda isabetli değeri üretilmesi beklenmez "aynı_etiket" biti sıfır değerini almasına rağmen karşılaştırmalarda her iki işlenen etiketi de isabetli değerini üretiyorsa mantık olarak geçici bir hata oluştuğu açıktır. Oysa devre şemasında bu durumun sonucunda geçici hata sıfır değerini alır. Çünkü bahsedilen durum, iki etiket değerinin birbirinden sadece bir bitlik bir farklılık olmasını ve çarpan parçacığın da o bite isabet etmesini gerektirdiği için, oldukça istisnai bir durumdur ve gerçekleşmesi beklenmez. Gerçekleşme ihtimali çok düşük olan bu duruma Şekil 4.2'de bir örnek gösterilmiştir.



Şekil 4.2. Bir Bit Farklı Etiketlerin Örneği

"aynı_etiket" biti sıfır olmasına rağmen iki etiket değeri de aynı saat vuruşunda bir değerini üretiyorsa, parçacığın etiketlere çarpıp ikisini de aynı değere eşitleme ihtimali, parçacığın "aynı_etiket" bitine çarpıp değerini değiştirmiş olma ihtimalinden çok daha düşüktür. İkinci durumda da zaten geçici hata sinyali üretilmemelidir.

Çizelge 4.1.'deki mantık çizelgesi incelendiğinde "aynı_etiket" bitinin geçici hatalara karşı hasar görme hassasiyetinin olmadığı görülmektedir. Parçacık "aynı_etiket" bitine çarpıp değerini değiştirdiğinde, komutun iki etiket değeri birbirinden farklıyken bit kurulu hale gelirse, işlemci komutta olmayan bir hatayı yakalayacaktır

ve boru hattını boşaltıp hatalı sandığı komuttan itibaren tekrar komutları boru hattına çekecektir. Bu durum başarımın düşmesine neden olsa da işlemcinin güvenilirliğine zarar vermeyecektir. Diğer durumda ise yani iki etiket değerinin birbirinin aynı olmasına rağmen, "aynı_etiket" bitine parçacık çarpmasıyla bit değerinin sıfırlandığı durumda, zaten işlemci hata yakalamaya çalışmayacaktır.

4.1.2. İki İşleneni Birbirinin Aynı olan Komutlarda Geçici Hata Tespiti

Mikroişlemcinin kullandığı komut kümelerinde yer alan komutların bir kısmı iki işlenen üzerinde işlem yapmakta ve bu işlenen etiketleri birbirinin aynı olabilmektedir. Yayın kuyruğunda yer alan bir komutun iki işleneni birbirinin aynı ise bu işlenenlerde oluşabilecek bir hata, bir işlenen için oluşan hata tespiti ile aynı yöntemle tespit edilebilmektedir. Yayın kuyruğuna gelen komutun iki işleneninin birbirinin aynı olduğu anlaşıldığında, işlenenlerin kopyalanmasına gerek kalmadan, "aynı_etiket" biti kurulur ve Şekil 4.1'de mantıksal çizimi verilen devre şeması kullanılır. İki işlenen de aynı olduğu için, yayımlanan etiket değerleriyle yapılan karşılaştırmaların sonucu aynı olmalıdır. Yapılan karşılaştırmalarda farklı bir sonuç üretildiğinde geçici bir hata oluştuğu anlaşılır ve hata oluşan komuttan itibaren boru hattındaki tüm komutlar temizlenip, yeniden boru hattına çekilirler.

Komutun kullandığı işlenenler birbirinin aynı olduğunu gösteren ve iki farklı durum için kurulan *"aynı_etiket"* biti, komut yayın kuyruğundan ayrıldığı anda sıfırlanır ve hata denetimi başarıyla tamamlanmış olur.

4.1.3. Bir İşleneni Hazır Olan Komutlarda Geçici Hata Tespiti

Yayın kuyruğuna getirilmiş olan komutların büyük bir çoğunluğunda komut yayın kuyruğuna girmeden önce işlenenlerden en az bir tanesinin değeri daha önceden hesaplanmış durumdadır [4,15]. Bu çalışmada önerilen geçici hata tespit yönteminin etkinliğinin artırılması için, yayın kuyruğuna bir işleneni hazır olarak gelen komutların, hazır olmayan işlenenin etiketi zaten hazır olan işlenenin etiketi için ayrılan alana kopyalanarak ikinci alandaki karşılaştırıcılar, komut yayın kuyruğunda bulunduğu sürece, hazır olmayan etiket değerini korumak için kullanılabilir. Ancak

hazır olmayan işlenenin hazır olan işlenenin alanına kopyalanması güvenli bir işlem değildir, çünkü bu işlemden sonra kopyalamanın yapıldığı alan geçerli bir bilgi içermez hale gelmektedir. Bu çalışmada önerilen hata tespit yönteminin bir işleneni hazır olan komutlar tarafından kullanılabilmesi için hazır olan işlenenin etiketi, Sharkey'in önerdiği "komut paketleme" yönteminde [4] olduğu gibi, ayrı bir RAM türü bellek alanına kaydedilir. Bu yeni bellek alanı ikinci işlenenin saklanmasında kullandığı için işlenen etiketi ile aynı sayıda bit içerse de, işletimi tamamlanan komutların yayımlanan sonuçlarına bağlı değildir ve hiçbir karşılaştırıcı devre içermez.

İlgili hata tespit yönteminin gerçekleştirilmesi için sadece "aynı_etiket" bitinin kullanılmasının yeterli olmayacağı açıktır çünkü daha önce anlatılan durumlardan farklı olarak zaten hazır olan etiket eklenen bellek alanına kopyalandığı için komut ihtiyaç duyduğunda ikinci işlenen etiketini yeni bellek alanından okumalıdır. Bu durum, komutun ikinci işlenen etiket değerini RAM alanından okuması gerektiğini gösteren, "işlenen_RAMde" isimli ayrı bir işaret alanı eklenerek sağlanabilir. "aynı_etiket" bitinin DÇG olmayan bir bit olduğu daha önce gösterilmiştir. Diğer taraftan parçacık "işlenen_RAMde" bit alanına çarpıp değerini bozarsa, komut ikinci işlenenin değerini yanlış bir kaynaktan okuyup işlemcinin çakılmasına neden olabilir. Ayrıca fazladan kopyalama yapıldığı için enerji israfı söz konusudur. İkinci etiketin kopyalandığı RAM alanının hata oluşmasına karşı öncekine göre daha hassas olması söz konusu olmadığı için, bu işlem yayın kuyruğunun hata hassasiyetini artırmaz.

Komutların işlenen sayılarına göre yapılan sınıflandırılmaya tekrar bakıldığında önerilen teknik tarafından kapsanmayan türler olduğu görülecektir. Bu türlerden ilki hiç işleneni olmayan komutlardır ve bu komutların geçerli etiket alanları olmadığı için korunmalarına da gerek yoktur. İkinci olarak, iki işlenen içeren komutların her iki işlenen değeri de birbirinden farklı olduğu ve hiçbir işlenen değeri hazır olmadığı durum ise diğer durumlar ile kıyaslandığında oldukça düşük orandadır ve bu teknik içerisinde kapsanmamaktadır.

4.2. MHÇ Hesaplanması

İşlemci bileşenlerinden birisine bir parçacık çarptığında, bu durumun bir hata ile sonuçlanacağı kesin değildir. Parçacık çarptığında bitin değerini değiştirse bile değeri değişen bit geçerli bir bilgi içermiyor ya da içerdiği bilgi programın son çıktısını etkilemiyor olabilir. Programın son çıktısını etkileyen bitler "Doğru Çalışma için Gerekli" ("Architecturally Correct Execution") oldukları için DÇG bit olarak adlandırılırlar [14]. İşlemci içerisindeki DÇG bitlerin oranı, işlemcinin mimari olarak hataya hassasiyetini ölçen başlıca parametredir. İşlemcinin "Mimari Hassasiyet Çarpanı" (4.1) ile hesaplanır.

$$\text{MHÇ} = \frac{\text{Yapıdaki Bir Saat Vuruşundaki Ortalama DÇG Bit Sayisi}}{\text{Donanim Yapısının Boyutu}} \qquad (4.1)$$

Komutların bazı bitleri geçici hatalara karşı hassasiyet göstermezler. Örneğin, eğer komut anlık veri kullanmıyorsa, yayın kuyruğunda anlık veriler için ayrılmış olan alan komut tarafından kullanılmayacağı için, bu alanda hata oluşsa da programın çalışmasında herhangi bir hata olmayacaktır. Başka bir deyişle bu bitler hata oluşumuna karşı hassas olmadıkları için DÇG olmayan bitlerdir. Benzer şekilde bir işlenen içeren komutların, ikinci işlenen için ayrılan alanlarda geçici bir hata oluşsa da programın çalışması bozulmayacağı için bu alanlardaki bitler DÇG olmayan bitlerdir.

BÖLÜM 5

5. DENEYSEL SONUÇLAR ve KIYASLAMALAR

Yayın kuyruğunda yer alan komutların işlenen etiket alanlarının geçici hatalara karşı duyarlılığını artırmak için kullandığımız yöntemlerin etkinliklerinin test edilmesi için, iki farklı mikroişlemci benzetim ortamından veriler elde edilmiştir. İlerleyen kısımda öncelikle benzetim ortamları hakkında bilgi verilecek, ardından önerilen tekniğin etkinliğinin ölçülebilmesi için gerekli veriler açıklanacaktır. Benzetim ortamlarından alınan değerler sunulduktan sonra, son olarak tekniğin etkinliği hesaplanacaktır.

5.1. Benzetim Ortamları

Mikroişlemci benzetim ortamları, donanım altyapısında işletilen komutları yazılım seviyesinde yer alan benzetimlerde işleterek, mikroişlemci veriminin ölçülmesi için kullanılan programlardır. Bu çalışmada m-Sim ve PTLsim olmak üzere iki farklı benzetim ortamında işlemci başarımı test edilmiştir. Bu iki benzetim ortamı farklı komut kümeleri için geliştirilmiş olmakla beraber, m-Sim benzetim ortamı çoklu kullanımlı (multi-threaded) uygulamaları da desteklemektedir. PTLsim'in çoklu kullanımlı uygulamaları destekleyebilmesi için PTLsim/X sürümü kullanılmalıdır.

Önerilen tekniklerin başarımının test edilmesi için denek taşı deneme programları (benchmarks), kullanılan benzetimliklerin üzerinden geçirilmiştir. M-Sim için bu denek taşlarının Alpha derleyici ile derlenmiş kodları kullanılıyorken, PTLsim için x86 ile derlenmiş kodlar kullanılır.

5.1.1. M-Sim

Joseph Sharkey tarafından, SimpleScalar 3.0d benzetimliği üzerinde bir takım değişiklikler yapılarak geliştirilen M-Sim (Multi-Threaded Simulator); çok kanallı uygulamalara izin veren bir mikroişlemci benzetim ortamıdır. M-Sim, sadece Alpha

AXP derlenmiş kodlarını desteklemekle birlikte, SimpleScaler'in üzerinde çalışabildiği tüm bilgisayar ortamlarında da çalışabilmektedir [16, 17].

M-Sim benzetimlik ortamı, SimpleScalar'dan farklı olarak, boru hattının adımlarını birbirinden ayırdığı için, yayın kuyruğunda geçici hataların tespiti için önerilen tekniğin başarımının denenmesi için daha uygundur.

5.1.2. PTLsim

PTLsim x86 için bir benzetimlik ortamı olmakla beraber x86–64 komut kümesi için de sanal bir makinedir. Pentium 4+, Athlon 64 ve benzer makinelere ait tüm x86–64 komut kümesini desteklemenin yanı sıra x86 mikroişlemcilerini modelleyen ve herkesin kullanımına açık olan tek araçtır [18].

Geçici hataların tespiti için yapılan çalışmaların x86 makineler için de veriminin ölçülmesi için PTLsim benzetim ortamı ve sanal makinesi kullanılmıştır.

5.2. Gerekli Parametreler

Geçici hataların tespiti için önerilen tekniğin başarımının ölçülmesi için gereken ilk parametre, programın işletiminde komutların işlenenlerine göre dağılımlarıdır. Yapılan bu ölçümle program boyunca işletilen komutların ne kadarının bu yöntemle kapsanabildiği tespit edilebilmektedir.

Çalıştırılan bir denek taşı programı içerisinde, emekliye ayrılan komutlar arasında, her bir komut türünün kaç kere kullanıldığı tespit edilmelidir. Örneğin dallanmanın yanlış tahmin edildiği durumlarda olduğu gibi, emekli olmayacak bir komutta meydana gelen bir hata programın çakılmasına neden olamayacağı için, yapılan testlerde emekliye ayrılmayan komutlar sayıma dâhil edilmezler.

Yöntemin etkinliğini ölçmek için değerlendirilmesi gereken ikinci değer Mimari Hassaslık Çarpanıdır (MHÇ). MHÇ'nin hesaplanma formülü daha önce (4.1) de verilmişti. Bu formül yardımı ile yayın kuyruğunun MHÇ değerinin hesaplanma formülü (5.1)'deki gibi ifade edilebilir.

$$\text{MHÇ} = \frac{\text{Yayın Kuyrugundaki Ortalama DÇG bit sayısı}}{\text{Yayın Kuyruğunun Boyu}} \tag{5.1}$$

Yayın kuyruğunda, parçacık çarpması sonucu değeri değişse de programın son çıktısını etkilemeyen yani DÇG olmayan bitler şunlardır.

- Hiç işlenen kullanmayan komutlar için işlenen etiketleri için ayrılan bitler
- Sadece bir işlenen kullanan komutlar için diğer işlenenin etiketi için ayrılan alan
- Anlık veri kullanmayan komutlar için anlık veriler için ayrılan alan
- Geçerli bir komut içermeyen satırlar
- Ürettiği sonuç kullanılmayan komutlar, örneğin dallanmaların yanlış tahmin edildiği veya hiç emekliye ayrılmayan komutlar

Yayın kuyruğunda yer alan DÇG bitlerin sayısının bulunabilmesi için, her saat vuruşunda kuyrukta ortalama kaç tane komutun yer aldığı ve bu komutların kaç tane işlenen içerdiği sayılmalıdır. Bu sayım için, her saat vuruşunda sayılan komutlar ayrı ayrı toplanır ve son olarak toplam saat vuruşuna bölünür. Böylece yayın kuyruğunun ortalama olarak hangi komut türünden kaç komut içerdiği hesaplanmış olur.

Yayın kuyruğunda her saat vuruşunda yer alan komutların sayılabilmesi için, yayın kuyruğunun her seferinde baştan sona taranması gerekir. 32 satırlı bir yayın kuyruğunda 100 milyon saat vuruşu çalışan bir program için bile, bu işlem 3 milyardan fazla karşılaştırma içermektedir. Ayrıca burada sayılan komutların hepsinin de emekliye ayrılacağı kesin değildir.

Bir programın çalışması boyunca her saat vuruşunda yayın kuyruğunda yer alan ortalama komut sayısının bulunması için daha etkin bir yol kullanılması mümkündür. Her bir komutun kuyrukta yer aldığı sürelerin toplamının, toplam saat vuruşuna bölümü istenen değeri verecektir. Başka bir deyişle emekli olan her bir komutun yayın kuyruğunda kalış süresi toplama eklenir ve program sonlandığında bu süreler toplamı, toplam saat vuruşuna bölünür.

İki farklı yöntemle hesaplanan bu değerlerin aynı olduklarının ispatlanması için öncelikle bir takım değerler tanımlanacaktır.

K: Program işletiminde yayın kuyruğunda yer almış ve emekliye ayrılmış komut kümesi

k_i : Programın işletimi boyunca yayın kuyruğunda bulunmuş ve emekliye ayrılmış bir komut $k_i \in K$

kgiriş$_i$: k_i komutunun yayın kuyruğuna girdiği saat vuruşu

kçıkış$_i$: k_i komutunun yayın kuyruğundan ayrıldığı saat vuruşu

kkalış$_i$: k_i komutunun yayın kuyruğunda kaldığı süre

T: Toplam saat vuruş sayısı

t: saat vuruşu

K_t: t anında kuyrukta yer alan ve program sonlanmadan önce emekliye ayrılacak olan komutlar kümesi ($K_t \subset K$)

OKS: Program işletimi boyunca yayın kuyruğunda yer alan ortalama komut sayısı

$$\mathbf{f(i,t)} = \begin{cases} 1 & \text{kgiriş}_i \le t \le k\text{cıkış}_i \\ 0 & \text{aksi takdirde} \end{cases}$$

, f(i,t) fonksiyonu t anında k_i komutu yayın kuyruğunda ise 1, değilse 0 değerini alır.

Bir k_i komutunun yayın kuyruğunda kalacağı süre, denklem (5.2)'de görüldüğü gibi, komutun kuyruktan çıktığı saat vuruşu ile komutun kuyruğa girdiği saat vuruşunun farkı olacaktır.

$$\text{kkalış}_i = \text{kçıkış}_i - \text{kgiriş}_i \tag{5.2}$$

kkalis_i süresinin f fonksiyonu yardımı ile ifadesi denklem (5.3)'deki gibidir. Bu denklemde programın çalışmasından itibaren her saat vuruşunda k_i denklemi kuyrukta ise toplama 1 eklenmiştir.

$$\text{kkalış}_i = \sum_{t=1}^{T} f(i,t) \tag{5.3}$$

Yayın kuyruğunda yer alan ortalama komut sayısı denklem (5.4) ile hesaplanır.

$$OKS = \frac{\sum_{t=1}^{T} (K_t)_s}{T} \tag{5.4}$$

Burada $(K_t)_s$, t anında kuyrukta yer alan komutların sayısı olarak ifade edilmektedir ve denklem (5.5) ile hesaplanabilir.

$$(K_t)_s = \sum_{i=1}^{K} f(i,t) \tag{5.5}$$

$(K_t)_s$ değeri denklem (5.4)'te yerine yazıldığında elde edilen sonuç denklem (5.6)'daki gibidir.

$$OKS = \frac{\sum_{t=1}^{T} \sum_{i=1}^{K} f(i,t)}{T} \tag{5.6}$$

Toplam sembolünün yer değiştirme özelliğinden yaralanılırsa denklem (5.7)'deki halini alır.

$$OKS = \frac{\sum_{i=1}^{K} \sum_{t=1}^{T} f(i,t)}{T} \tag{5.7}$$

Denklem (5.7)'de bir komutun kuyrukta yer aldığı saat vuruşu sayısına ulaşılmıştır. Bu değer denklemde yerine konduğunda denklem (5.8) elde edilir.

$$OKS = \frac{\sum_{i=1}^{K} \text{kkalış}_i}{T} = \frac{\sum_{i=1}^{K} (\text{kçıkış}_i - \text{kgiriş}_i)}{\text{T}} \tag{5.8}$$

Sonuç olarak bir saat vuruşunda yayın kuyruğunda yer alan ortalama komut sayısı, emekliye ayrılan tüm komutlar için komutların yayın kuyruğundan çıktığı saat vuruşu ile komutun kuyruğa girdiği saat vuruşunun farklarının toplamının toplam saat vuruşuna bölümüne eşittir.

OKS'nin denklem (5.4) ile hesaplanma karmaşıklığı her saat vuruşunda yayın kuyruğundaki tüm komutlar tek tek kontrol edildiği için O(T.K)'dır. Ayrıca bu işlemde komutların emekliye ayrılıp ayrılmadığının da ayrıca kontrol edilebilmesi için ek veri yapılarının kullanılması gerekir. Denklem (5.8) ile OKS hesaplanabilmesinde ise, benzetim ortamında, yayın kuyruğuna giren her komuta giriş yaptığı saat vuruşunu gösteren bir bilgi eklenir. Programın işletimi sırasında bir komut emekliye ayrıldığı anda komutun yayın kuyruğunda kaldığı zaman toplama eklenir. Denklemden de görüleceği gibi OKS'nin bu yolla hesaplanma karmaşıklığı sadece O(K)'dır.

OKS değeri hesaplanırken dikkat edilmesi gereken bir nokta daha bulunmaktadır. Geçici hataların tespiti için önerilmekte olan tekniğin başarımını denemek için tüm komut türleri için toplam bir OKS hesaplamak yerine her bir komut türü için ayrı ayrı OKS hesaplanmasını gerekmektedir.

Sonuç olarak benzetim ortamında hesaplanması gereken parametreler şu şekildedir.

- Bir programın çalışma süresi boyunca emekliye ayrılan komutları arasında her bir komut türünden kaç tane emekliye ayrıldığı ve komut türlerinin oranı.
- Her saat vuruşunda yayın kuyruğunda, her komut türünden ortalama olarak kaç tane komutun yer aldığı.

5.3. Benzetim Ortamında Benzetilen İşlemcinin Özellikleri

Her bir denek taşında 1 milyar komut atlandıktan sonra 200 milyon komut çalıştırılmıştır. Benzetimliğin parametreleri Çizelge 5.1'de gösterilmiştir.

Çizelge 5.1.Benzetimlik Yapılandırmaları

Parametre	Yapılandırma
Makine Genişliği	4 komut getir, 4 komut yayınla, 4 komut sonlandır
Pencere Boyu	32 satır yayın kuyruğu 48 Satır yükleme saklama kuyruğu 128 satır YSB (Yeniden Sıralama Belleği) 128 adet yazmaç

İşlem Birimleri ve Gecikmeler	tamsayı AMB (1/1), yükleme/saklama birimi (2/1), tamsayı çarpma (3/1), tamsayı bölme (12/12), kayan noktalı sayı toplama (2/1), kayan noktalı sayı çarpma (4/1), kayan noktalı sayı bölme (12/12).
L1 Komut Önbelleği	32 KB, 2-yollu kümeli ilişkili, 32 bayt satır, 1 saat vuruşu isabet zamanı
L1 Veri Önbelleği	32 KB, 4-yollu kümeli ilişkili, 32 bayt satır, 1 saat vuruşu isabet zamanı
L2 Tümleşik Önbellek	512 KB, 8-yollu kümeli ilişkili, 64 bayt satır, 6 saat vuruşu isabet zamanı
BTB	512 satır, 4-yollu kümeli ilişkili
Dallanma Tahmin	2K satır çift durumlu
Bellek	8 bayt genişlik, ilk parça 100 vuruş, parçalar arası 1 vuruş
ADÖ	16 giriş (K) 4-yollu kümeli ilişkili, 32 satır (V) 4-yollu kümeli ilişkili, Bulamama gecikmesi 30 vuruş

5.4. Sonuçlar

Çizelge 5.2.'de PTLsim ve sim benzetim ortamlarında çalıştırılan denek taşları ayrı ayrı listelenmiştir. Kullanılan tüm denek taşlar spec2000 kümesine aittir. Benzetim ayarlarında da belirtildiği gibi, tüm deneylerde 1milyar komut atlanıp 200 milyon

komut çalıştırılmıştır. Herhangi bir program çalışmaya başladığında, ilk komutlarda boru hattının dolması, verilerin ön belleğe çekilmesi gibi işlemlerden ötürü, bu periyotta alınan ölçümler ortalama değerlerden farklıdır. Bu sebeple çalıştırılan her bir denek taşında ilk 1milyar komut, ölçüm alınmadan atlanmıştır.

Çizelge 5.2. Benzetimlerin yapıldığı denektaşları

gzip	gcc	crafty	bzip	vortex	Apsi	perlbmk
vpr	mcf	eon	twolf	wupwise	sixtrack	parser
swim	mgrid	Applu	mesa	ammp	lucas	equake

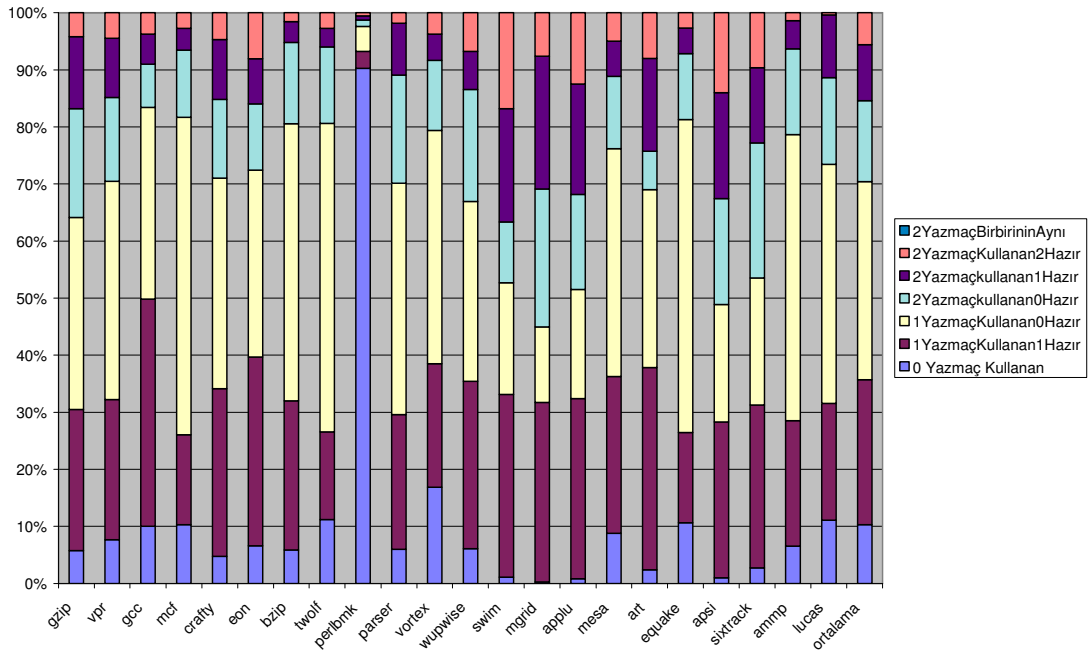
5.4.1. M-Sim sonuçları

M-Sim'de yapılan benzetimler sonucunda elde edilen veriler Çizelge 5.3'de gösterilmiştir. Çalıştırılan tüm programlar için emekliye ayrılan komutların kümesinin, komut türlerine göre sayıları gösterilmiştir.

Çizelge 5.3. M-Sim Benzetim Sonuçları

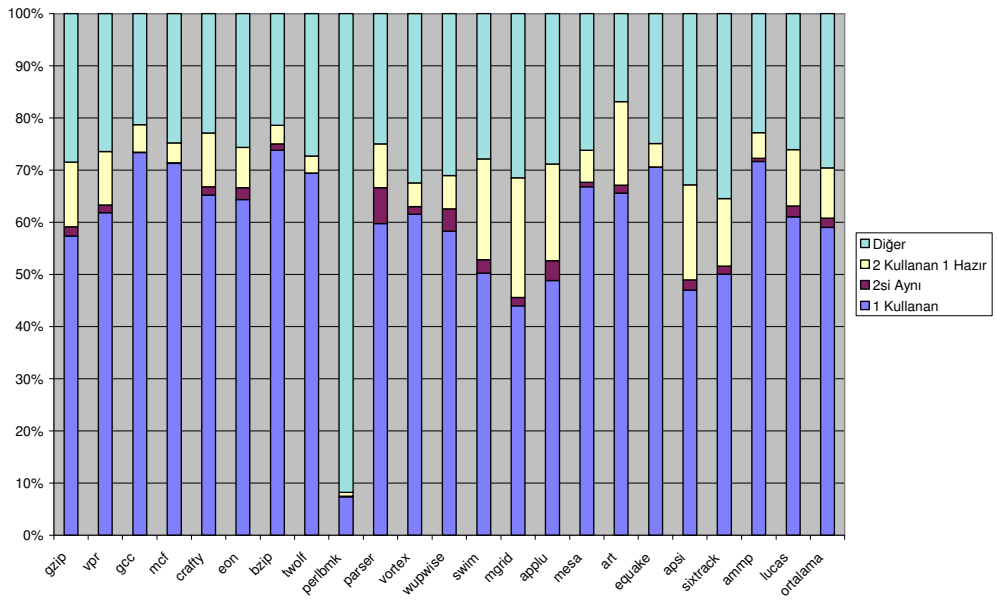
	0Kullanan	1Kullanan0Hazır	1Kullanan1Hazır	2Kullanan0Hazır	2Kullanan1Hazır	2Kullanan2Hazır	2 Aynı
Gzip	11532026	67354445	49384026	38117329	25215133	8397042	3679244
Vpr	15383040	76430693	49089751	29404893	20717251	8974374	3052099
Gcc	20060014	67212234	79633324	15068448	10509297	7516684	168254
Mcf	20600388	111248118	31468815	23649043	7574512	5459125	271201
Crafty	898769	6882016	5665230	2746123	1519996	688921	187411
Eon	13168660	65534443	66148961	23182584	15815570	16149783	4688723
Bzip	11662739	96999992	52391865	28557483	7254139	3133782	2494837
Twolf	22448875	108069701	30724146	26706889	6520892	5529497	0
Perlbmk	189586876	4706528	3169410	1174211	789989	572986	107493
Parser	12010183	81140818	47122406	37909266	18130464	3686865	14674875
Vortex	33781538	81732563	43204814	24613123	9198134	7469830	2924781
Wupwise	12107525	63105347	58704117	39226368	13358068	13498578	9037306
Swim	2233192	39136696	64023903	21346082	39620932	33639196	5344477
Mgrid	655129	28108775	59994618	46838464	47234183	17168833	2710155
Applu	626011	14990756	24718320	13039480	15112127	9788343	3107803
Mesa	17635046	79860283	54884774	25272630	12424181	9923086	1800679
Equake	19719426	105845751	34307132	24683176	9298966	6145551	691505
Apsi	1939050	41153294	54664859	37098856	37132393	28011550	4071703
Sixtrack	5399313	44576556	57123951	47263307	26319637	19317239	3074291
ammp	13102574	100191928	43947818	30078704	9894865	2784113	1216200
lucas	22182076	83754457	40932520	30353541	22045854	731555	4231759

M-Sim'de elde edilen sonuçların yüzdelik dağılımlarının grafiği Şekil 5.1'de gösterilmiştir.



Şekil 5.1 Komutların Program Boyunca Yüzdelik Dağılımı

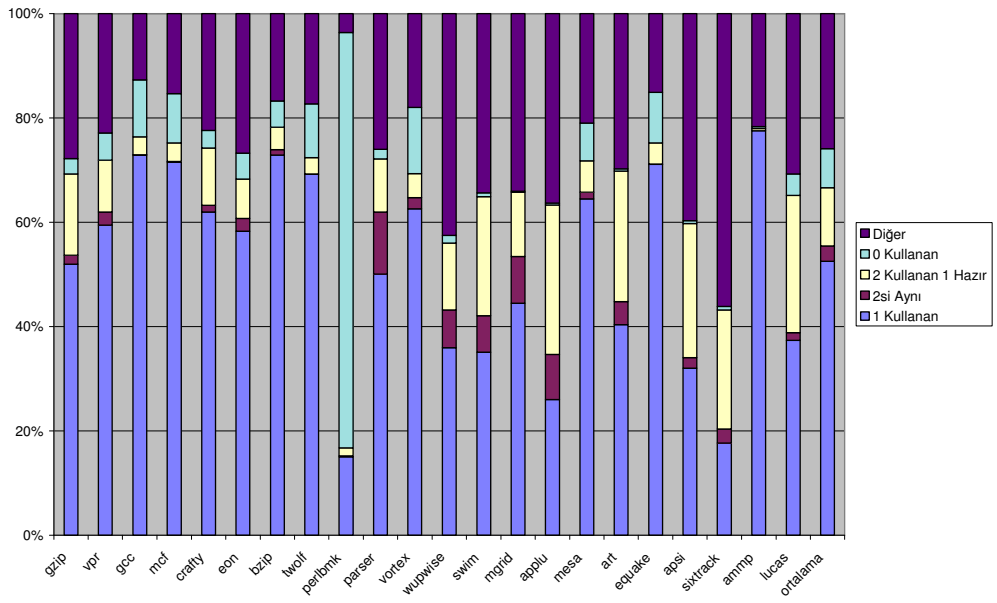
Etiket karşılaştırıcılarının kullanılmasıyla geçici hataların tespiti yöntemi, 1 işlenen kullanan komutları, 2 işlenenin birbirinin aynı olduğu komutları ve 2 işlenen kullanan komutlardan 1 işlenenin önceden hazır olduğu komutları içermektedir. M-Sim'de elde edilen sonuçların sadece önerilen tekniğin kapsadığı komutlar açısından dağılımı Şekil 5.2'de görülmektedir.



Şekil 5.2. Önerilen Tekniğin Kapsamı

Şekil 5.2'de görüleceği gibi geçici hataların tespiti için işlenen etiketlerinin karşılaştırılması yöntemi, Alpha komut kümesini kullanan bir işlemci için, program komutlarının ortalama %70'ini kapsamaktadır.

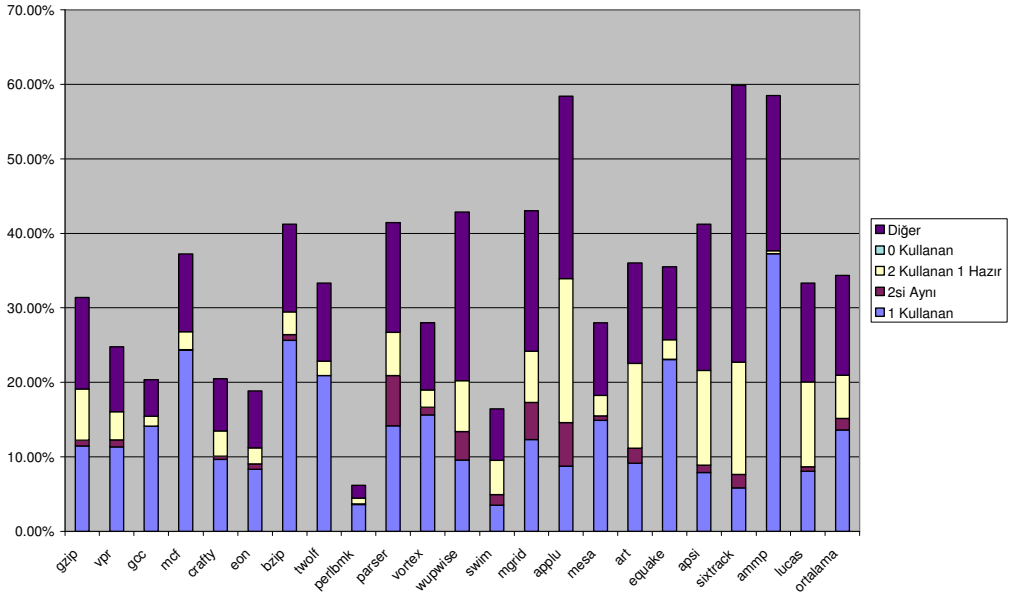
Önerilmekte olan tekniğin etkinliği MHÇ değeri ile ölçülecektir. Bunun için herhangi bir anda yayın kuyruğunda yer alan ortalama komut sayılarına ihtiyaç vardır. M-sim'de hesaplanan değerlerin grafik dağılımı Şekil 5.3'te gösterilmiştir.



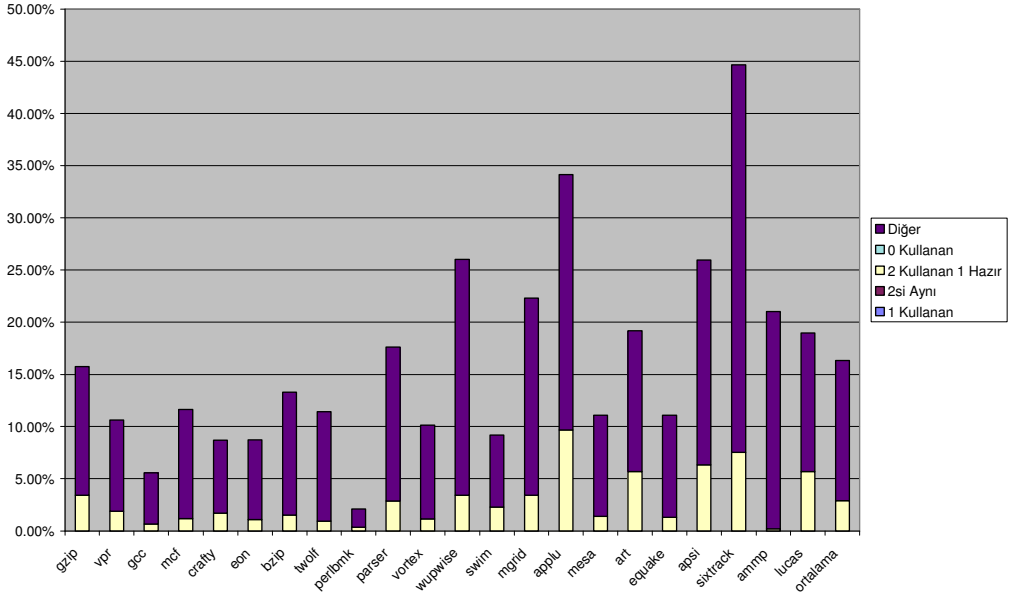
Şekil 5.3.Yayın Kuyruğundaki Ortalama Anlık Komutlar

Herhangi bir anda yayın kuyruğunda en çok bir işlenenli komutlar yer alsa da, her komut türündeki DÇG bit sayısı aynı değildir. Örneğin 1 işlenenli komutlarda 7 bit DÇG iken 2 işlenenli komutlarda DÇG bit sayısı 14'dür. Önerilen teknik uygulanmadan önce yayın kuyruğundaki etiket alanlarının MHÇ değeri Şekil 5.4'te gösterilmiştir.

Bir bitte hata oluşsa da program hasarsız çalışmasına devam ediyorsa o bit DÇG olmayan bir bittir. Bunun için önerilen teknik kapsamında korumaya alınan bitler DÇG olmayacağı için MHÇ değeri azalacaktır. Uygulanan teknik sonucunda yayın kuyruğundaki etiket alanlarının MHÇ değeri ise Şekil 5.5'te gösterilmiştir.



Şekil 5.4 Önerilen Teknikten Önce Etiketlerin MHÇ Değeri



Şekil 5.5. Önerilen Teknikten Sonra Etiketlerin MHÇ Değeri

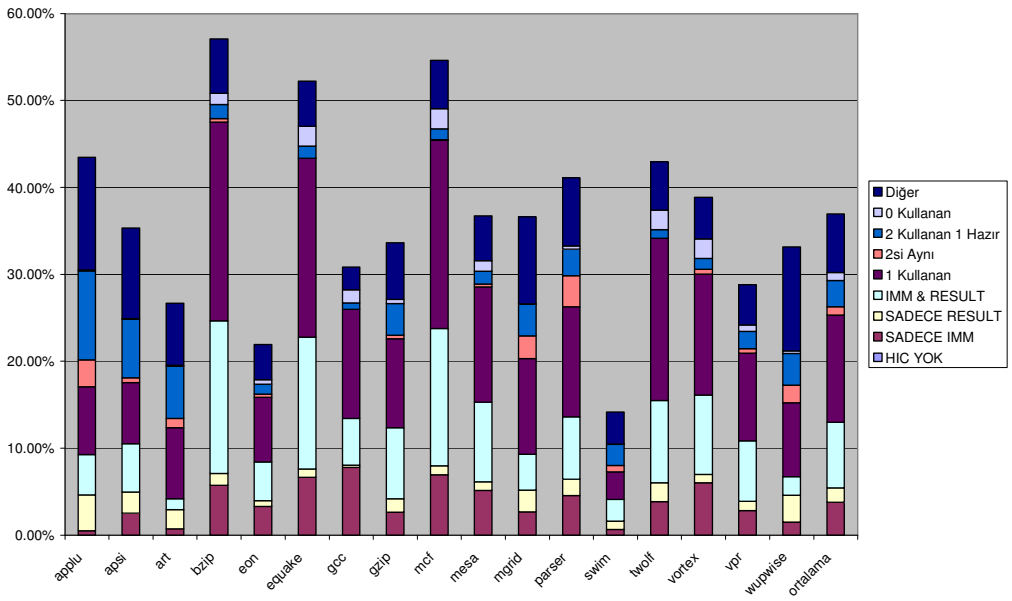
Şekil.5.4 ve Şekil.5.5'ten de görüleceği yayın kuyruğundaki etiketlerin MHÇ değeri %34.38 iken, önerilen teknikten sonra bu değer %16.33'e çekilmiştir. Önerilen tekniğin etiket alanları üzerindeki verimliliği % 52'dir.

$$\text{Etiket Verimi} = \frac{34,38 - 16,33}{34,38} * 100 = \%52 \tag{5.9}$$

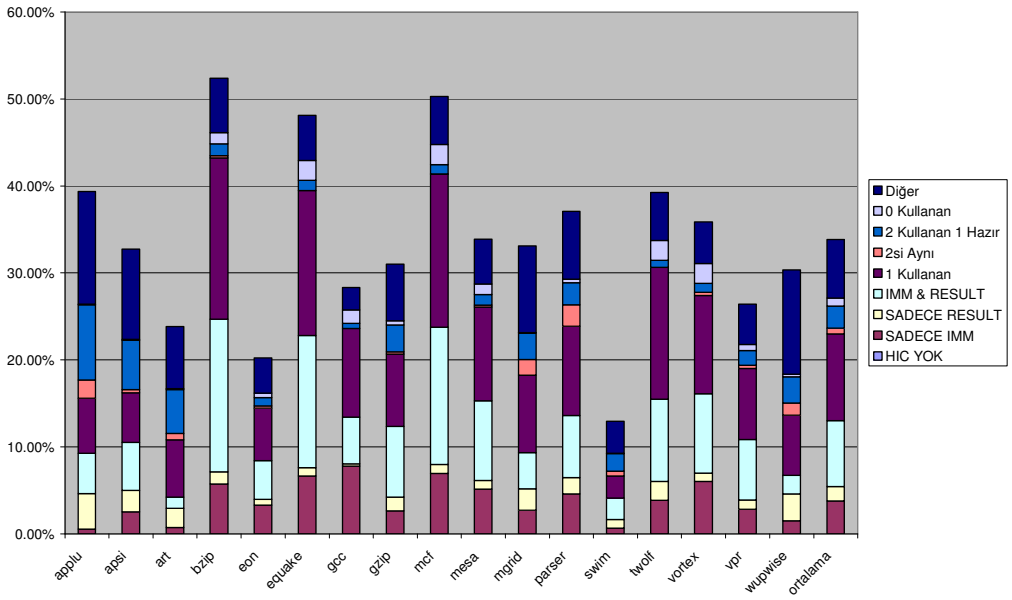
Önerilen tekniğin yayın kuyruğunun güvenilirliğine katkısının bulunabilmesi için anlık veri kullanan ve sonuç üreten komutların oranına ihtiyaç duyulmaktadır. Yayın kuyruğunda yer alan bir satırın içeriği şu şekildedir.

Geçerli Biti	→ 1 bit
Yeniden Sıralama Kuyruğu	→ 7 bit (128 satır YSK)
Yükleme Saklama Kuyruğu	→ 6 bit (48 satır)
Etiket1	→ 7 bit (128 yazmaç) + 1 bit
Etiket2	→ 7 bit (256 yazmaç) + 1 bit
Sonuç	→ 7 bit
Dallanma Etiketleri	→ 4 bit (16 dal)
İşlem Kodu	→ 10 bit
Anlık Veri	→ 32 bit
TOPLAM	→ 83 bit

Şekil 5.6 yayın kuyruğunun önerilen teknikten önceki MHÇ değerini, Şekil 5.7 ise önerilen teknikten sonraki MHÇ değerini göstermektedir.



Şekil 5.6.Yayın Kuyruğunun MHÇ değeri (Önerilen Teknikten Önce)



Şekil 5.7.Yayın Kuyruğu MHÇ Değeri (Önerilen Teknikten Sonra)

Yayın Kuyruğunun MHÇ değeri ortalama olarak %36,95'den %33,83'e indirilmiştir. Önerilen tekniğin yayın kuyruğundaki verimliliği %8,44'dür. Yayın kuyruğu

mikroişlemcinin %10'u olduğu için tekniğin mikroişlemcide sağladığı verim %0,844'dür.

5.5. Bellekten Kayıp Miktarı

Önerilmekte olan teknikte, her bir satır için iki bitlik bir kayıp söz konusudur. Bu alanın kaybettirdiği oran denklem (5.10) ile hesaplanmıştır.

$$\text{Yayin Kuyruğunda Kayip} = \frac{\text{Eklenen Bit Sayısı}}{\text{Bir Satırın Boyu}} * 100 = \frac{2}{83} * 100 = \%2,4 \qquad (5.10)$$

Bu kayıp miktarının bütün mikroişlemcideki oranı %0,24 kadardır.

Diğer taraftan, güvenilirliği artırırken kayba sebep olabilecek bir başka unsur da eklenen devrenin güç tüketimidir. Ancak bu tüketim, bütün mikroişlemci göz önünde bulundurulduğunda önemsenmeyecek kadar küçüktür. Önerilen teknikte zaten mikroişlemci üzerinde hazır bulunan elemanlar kullanıldığı için güçten kayıp çok azdır.

BÖLÜM 6

6. SONUÇLAR

Nötron ya da alfa parçacıklarının çarpmasıyla oluşan ve düzeltilmesi mümkün olan geçici hatalar, her geçen gün, mikroişlemci tasarımındaki önemli sorunlardan birisi haline gelmektedir. Günümüzde üretilen çok yollu işlemcilerde komutların sırasız işletilmesini olanaklı hale getirmek için yayın kuyruğunda, çoğu zaman atıl durumda bulunan, pek çok karşılaştırıcı devre mevcuttur. Karşılaştırıcı devrelerin verimsiz kullanılması yerine, işlenen etiketlerinde ya da karşılaştırma devrelerinde oluşabilen geçici hataların tespiti için kullanılması mümkündür.

Karşılaştırıcı devrelerin kullanılarak geçici hataların tespit edilmesi yöntemi, Alpha işlemcilerde, bir program boyunca işletilen komutların ortamla %70'i için kullanılan bir yöntemdir.

Yayın kuyruğunda herhangi bir anda yer alan komut sayısının hesaplanması için program boyunca her saat vuruşunda kuyrukta yer alan komut sayısının ortalaması alınabilir. Ancak karmaşıklığı yüksek olan bu işlem yerine, her bir komutun yayın kuyruğunda kaldığı süreler toplanarak, programın toplam işletildiği saat vuruşu sayısına bölmek daha etkin bir yöntemdir.

Yapılan testler sonucunda, geçici hataya karşı önlem alınmadan herhangi bir anda yayın kuyruğunun Mimari Hassaslık Çarpanı (MHÇ) %36,95 olarak hesaplanmıştır. Sadece işlenen etiketlerinin MHÇ değeri ise %34,38 olarak hesaplanmıştır.

Etiket karşılaştırıcılar kullanılarak geçici hatalara karşı önlem alındıktan sonra yayın kuyruğundaki etiketlerin MHÇ değeri %16,33'e indirilirken yayın kuyruğunun MHÇ değeri ise %33,83'e indirilmiştir.

Etiket karşılaştırıcılarının kullanılmasıyla geçici hataların tespit yöntemi etiket alanlarında %50 verimli iken bu oranın yayın kuyruğuna katkısı %8,44'tür. Yayın kuyruğu bütün mikroişlemcinin ortalama %10luk kısmını tuttuğu için önerilen tekniğin tüm işlemcideki verimi %0,844'tür.

Etiket karşılaştırıcılarının kullanılmasıyla geçici hata tespit yöntemi her bir yayın kuyruğu satırına iki bit eklenmesini ve her satıra 3 ÖZEL VEYA, 1 VE ve 1 VEYA kapısı içeren küçük bir devre eklenmesini gerektirmektedir. Eklenen fazladan bit alanlarının yayın kuyruğundaki kaybı %2,44 oranındadır ve bu oranın tüm işlemciye etkisi %0,244 oranındadır. Eklenen devre elemanlarının güç tüketimi ise tüm mikroişlemcinin yanında çok küçüktür.

Bu çalışmada önerilen yöntem, zaten işlemci içerisinde hazır olan ve etkin kullanılamayan karşılaştırıcıların geçici hata tespiti için de görevlendirilmesiyle sadece %0,244 oranında kayıpla, %0,844 oranında kazanç sağlamaktadır.

KAYNAKLAR

[1] Mukherjee, S. S., Emer, J., Fossum, T., Reinhardt, S. K , Cache Scrubbing in Microprocessors: Myth or Necessity?,10th International Symposium on Pacific Rim Dependable Computing (PRDC), Mart 2004

[2] Ponomarev, D. V., Ergin, O., Energy Efficient Comparators for Superscalar Datapaths, IEEE Transactions on Computers, 53, 7, Temmuz 2004

[3] Weaver, C., Emer, J., Mukherjee, S.S., Reinhardt, S.K. ,Techniques to Reduce the Soft Errors Rate in a High-Performance Microprocessor, 31st Annual International Symposium on Computer Architecture (ISCA),Haziran 2004

[4] Ernst, D., Austin, T., Efficient Dynamic Scheduling Through Tag Elimination, 29th Annual International Symposium on Computer Architecture (ISCA), Mayıs 2002

[5] Stark, J., Brown, M. D, Patt, Y. N, On Pipelining Dynamic Instruction Scheduling Logic, 2000. MICRO-33. Proceedings. 33rd Annual IEEE/ACM International Symposium, 57, 66, 2000

[6] Smith, J. E., Sohi, G. S.,The Microarchitecture of Superscalar Processors, Proceedings of the IEEE, 83(12) ,1609-1624, 1995

[7] Moudgill, M., Pingali, K., Vassiliadis, S., Register Renaming and Dynamic Speculation: an Alternative Approach**,** 26th annual international symposium on Microarchitecture ,Eylül 1993

[8] Tomasula, R., M., An Efficient Algorithm For Exploiting Multiple Arithmetic Units,IBM Journal, Ocak 1967

[9] Baumann, R., The Impact of Technology Scaling on Soft Error Rate Performance and Limits to the Efficacy of Error Correction, Electron Devices Meeting IEDM Digest. International, 329–332, 2002

[10] Phelan, R., Addressing Soft Errors in ARM Core-based Designs, White Paper, ARM, Aralık 2003

[11] "IBM Experiments in Soft-Fails in Computer Electronics (1978-1994)", erişim adresi: http://www.research.ibm.com/journal/rd/401/curtis.html, erişim tarihi: 17 Mayıs 2007

[12] May, C. T., Woods, M. H, Alpha Particle Induced Soft Errors in Dynamic Memories , IEEE Transactions on Electron Devices, ED-26, Ocak 1979

[13] C. S. Guenzer, E. A. Wolicki, and R. G. Allas, Single event upsets of dynamic RAM's by neutrons and protons, IEEE Trans. Nucl. Sci., NS-26, 5048, 1979.

[14] Mukherjee, S. S.,Weaver, C., Emer, J., Reinhardt,S. K., Austin,T., A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor, To appear in the Proceedings of the 36th Annual International Symposium on Microarchitecture (MICRO), Aralık 2003

[15] Sharkey,J. J., Ponomarev, D. V., Ghose, K., Ergin,O., Instruction Packing: Toward Fast and Energy-Efficient Instruction Scheduling, ACM Transactions on Architecture and Code Optimization, 3-2,156–181, Haziran 2006

[16] "M-Sim: The Multi Threaded Simulator" , erişim adresi: http://www.cs.binghamton.edu/~jsharke/m-sim/ , erişim tarihi: 11 Aralık 2006

[17] Sharkey, J., M-Sim: A Flexible, Multithreaded Architectural Simulation Environment, Technical Report CS-TR-05-DP01, Ekim 2005.

[18] "Ptlsim x86-64 Cycle Accurate Processor Simulation Design Infrastructure", erisim adresi: http://www.ptlsim.org/, erisim tarihi: 10 Aralık 2006

ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, adı : YALÇIN, Gülay

Uyruğu : T.C.

Doğum tarihi ve yeri : 07.11.1980 Ankara

Medeni hali : Bekar

Telefon : 0 (312) 292 40 76

Faks : 0 (312) 292 40 91

e-mail : gulay@etu.edu.tr

Eğitim

Derece	Eğitim Birimi	Mezuniyet tarihi
Lisans	HacettepeÜniversitesi/Bilgisayar Mühendisliği	2004

İş Deneyimi

Yıl	Yer	Görev
2004-2006	TOBB Ekonomi ve Teknoloji Üniversitesi	Araştırma Görevlisi

Yabancı Dil

İngilizce