

GENİŞLETİLMİŞ TOMASULO ALGORİTMASI VE KURALDIŞI DURUMLARIN İŞLENMESİ

Müge Sayıt* ve Ahmet Bilgili*

(*)Ege Üniversitesi, Uluslararası Bilgisayar Enstitüsü, 35100, İZMİR
muge.fesci@ege.edu.tr, ahmetbilgili@gmail.com

ÖZET

1967 yılında geliştirilen Tomasulo algoritması, yazmaç yeniden isimlendirme yöntemi ile WAR ve WAW risklerine karşı beklemeye neden olmadan çözüm üretir. Genişletilmiş Tomasulo algoritması, yazmaçlar için ayrı bir ünite tasarlanmıştır. Bu çalışmada, Genişletilmiş Tomasulo algoritması bir MIPS simülöründe uygulanmış ve Yazmaç Güncelleme Ünitesi (YGÜ) boyutlarına göre performans değerlendirmesi yapılmıştır.

Anahtar Kelimeler: Tomasulo Algoritması, Genişletilmiş Tomasulo, Kuraldışı Durum.

EXTENDED TOMASULO ALGORITHM AND EXCEPTION HANDLING

ABSTRACT

Tomasulo algorithm, which was developed in 1967, avoids WAR and WAW hazards using register renaming. In Extended Tomasulo algorithm, a different unit is designed for registers. In our work, Extended Tomasulo algorithm is implemented in a MIPS simulator and performance is measured according to different size of Register Update Unit (RUU).

Keywords: Tomasulo Algorithm Extended Tomasulo, Exception.

1. GİRİŞ

Programlarda yer alan veri riskleri (data hazards) işhatlı (pipeline) bir işlemcinin bu risk ortadan kalkana kadar beklemesine (stall) neden olmaktadır. Bekleme süresini azaltmak ya da yoketmek için işhatlarında statik ve/veya dinamik zamanlama kullanılır. Statik zamanlama, derleyici tarafından derlenme zamanında ele alınırken; dinamik zamanlama donanımsal olarak çalışma zamanında çözülür.

Dinamik zamanlama yönteminde kullanılan belli başlı algoritmalar arasında Scoreboard ve Tomasulo sayılabilir. Scoreboarding algoritması, tüm veri risklerini tespit ederek ortadan kaldırır ve eğer çalışmada herhangi bir risk yoksa komutları sırasız olarak çalıştırabilir.

Tomasulo algoritması ise yazmaçları yeniden isimlendirerek (renaming) oluşabilecek veri risklerini (WAR ve WAW) ortadan kaldırmaktadır. Scoreboard algoritmasında hedef yazmaçlar yeniden isimlendirilemediği için tekrar kullanılınca kadar sakladığı bilginin korunması gerekir. Bu durum işlemcinin beklemesine neden olur.

Genişletilmiş Tomasulo algoritması, Yazmaç Güncelleme Ünitesi (YGÜ) ile tüm yazmaçları bir havuzda toplar ve kuraldışı durumların işlenmesinde kolaylık sağlar.

Makalenin izleyen bölümleri şu şekildedir: 2. bölümde Tomasulo algoritması, 3. bölümde Genişletilmiş Tomasulo algoritmasından söz edilmiştir, 4. bölümde gerçekleştirilen uygulama ve 5. bölümde sonuçlar yer almaktadır.

2. TOMASULO ALGORİTMASI

Tomasulo algoritması işletilirken, komutları sırayla yayınladıktan (issue) sonra parametlerinin hazır olup olmadığına bakılır. Eğer parametler hazır değilse komut, bu parametler tarafından kullanılacak olan işlevsel ünitelerin rezervasyon istasyonlarında beklemeye alınır. Komutlarda yer alan yazmaçlar, rezervasyon istasyonlarındaki değerler ile eşlenir. Bu işleme yazmaç yeniden isimlendirilmesi denir. Tomasulo algoritması Scoreboard algoritmasının aksine yazmaç yeniden isimlendirilmesi ile aynı yazmaçların tekrar kullanılmasına izin verir, WAR ve WAW veri risklerini önlenmesini

sağlar ve bu sebeple işlemcide oluşabilecek beklemlerin önüne geçer [3].

İşlevsel ünitelerden çıkan sonuçlar ortak veri hattı ile tüm bekleyenlere iletilir. Bir komut içerisindeki tüm parametreler bu yolla hazır olduğunda, komutun gerektirdiği işlevsel üniteyi kullanarak işlem yapar [3]. İşlem tamamlandığında, işlevsel ünite serbest kalır, hedef yazmaca işlem sonucu yazılır.

Her bir kaynak yazmaç için, o yazmacın meşgul olup olmadığını belirten bir bit bulunmaktadır. Bir yazmaç, çalışmakta olan bir komutun hedef yazmacı konumunda ise meşguldür. Bir rezervasyon istasyonunun alanları şekil 1’de gösterilmiştir.

Kaynak parametre 1		Kaynak parametre 2		Hedef		
Hazır	Etiket	İçerik	Hazır	Etiket	İçerik	Yazmaç

Şekil 1 Rezervasyon İstasyonu Alanları

Eğer bir kaynak yazmaç, komut yayınlanma aşamasına geçtiğinde meşgulse, bu yazmaç için kullandığı işlevsel üniteyi belirten bir etiket (tag) verilir ve komut rezervasyon istasyonuna alınır. Eğer hedef yazmaç meşgulse, o yazmaç için belirlenen etiket rezervasyon istasyonuna yazılır. Bu etiketlerin belirttiği işlevsel üniteler hazır olduğunda, komut tamamlanmış olur ve rezervasyon istasyonundan çıkarılır.

Tomasulo algoritmasının 3 adımı kısaca şu şekilde özetlenir [3]:

1.Yayınlama: Eğer rezervasyon istasyonu boşsa, algoritma parametreleri gönderir ve yazmaçları yeniden isimlendirir.

2.Çalıştırma: Her iki parametre de hazırsa, komut çalıştırılır, aksi halde ortak veri hattından parametrelerin hazır olmasıyla yayınlanan sonuçlar beklenir.

3.Sonuçları yazma: Tüm hazır parametreler ortak veri hattına konur, rezervasyon üniteleri kullanılabilir olduğunu belirtecek şekilde işaretlenir.

3. GENİŞLETİLMİŞ TOMASULO ALGORİTMASI

3.1. Etiket Ünitesi

Standart Tomasulo algoritmasında, hedef yazmaçlar için etiket belirleme işlemi yapılmaktadır. Her yazmaç, komutların işleniş sırasında hedef yazmaç olabileceği için her birine ayrı etiket verilmektedir. Yazmaç ile etiketi arasında eşleştirme yapabilmek için ise bu işlemi

gerçekleştirebilmeyi sağlayacak bir karşılaştırma yapısına ihtiyaç duyulmaktadır [2].

Genişletilmiş Tomasulo algoritmasında, yazmaç ve etiket eşleştirme işleminin getirdiği ek yükü azaltmak için ayrı bir etiket ünitesi tasarlanmıştır. Tasarlanan bu yeni etiket ünitesinde, sadece kullanımda yani aktif olan yazmaçlar için bu yazmaçlara etiket atanmaktadır. Sistemde o andaki tüm aktif etiketler birleştirilerek Etiket Ünitesi oluşturulmuştur [2]. Eğer kaynak yazmaç meşgulse, Etiket Ünitesi bu yazmaç için bir etiket atayarak bunu rezervasyon istasyonuna iletir. Komut yayın aşamasında eğer uygun etiket yoksa yani Etiket Ünitesi dolu ise komut beklemeye alınır.

Etiket ünitesinde bir yazmaç için birden fazla etiket verilmiş olabilir. Yapılan işlemlerin karışmaması için yazmaçlar için sadece en son verilen etiket değeri dikkate alınır [2]. Etiket ünitesinin eklenmesinden sonra oluşturulan bir rezervasyon istasyonunun yapısı şekil 2’de gösterilmiştir.

Kaynak parametre 1 Kaynak parametre 2 Hedef

Kaynak parametre 1		Kaynak parametre 2		Hedef		
Hazır	Etiket	İçerik	Hazır	Etiket	İçerik	TU slotu

Şekil 2 Rezervasyon İstasyonunun Yapısı

Etiket Ünitesi’nin alanları şekil 3’te, örnek bir etiket ünitesi ise Tablo 1’de gösterilmiştir.

Etiket No	Yazmaç No	Etiket Boş	Son kopya
-----------	-----------	------------	-----------

Şekil 3 Etiket Ünitesi

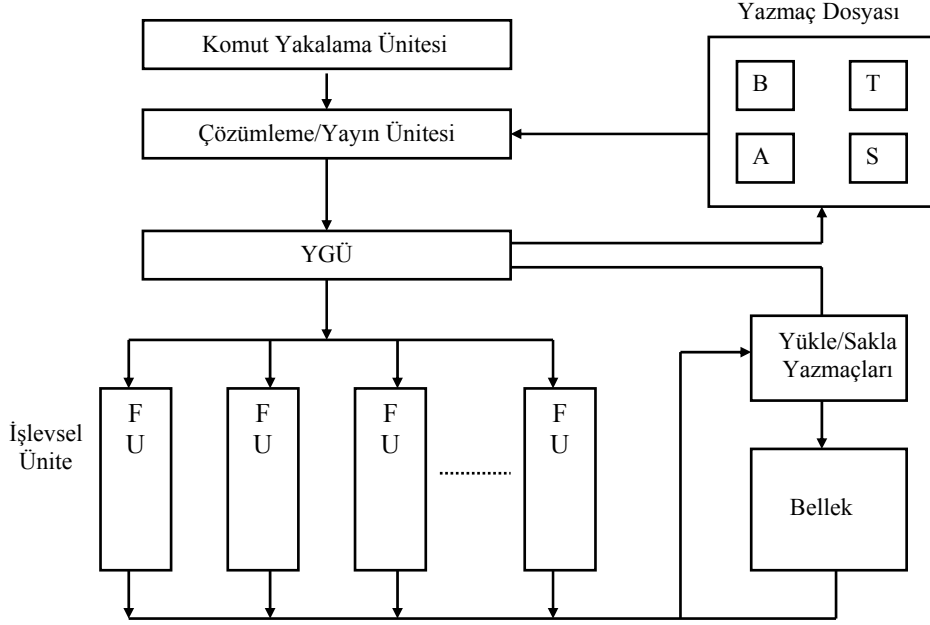
Ünite kaydında yer alan ‘yazmaç no’ o etiketin bağlı olduğu yazmacın numarasını (adını), ‘etiket boş’ o etiket numarasının boş olup olmadığını, ‘son kopya’ ise verilen etiketin o yazmaç için en güncel etiket olup olmadığını belirtir [2]. Tabloda E, H ve T sırasıyla evet, hayır ve tanımsız anlamında kullanılmıştır.

Tablo 1 Örnek bir Etiket ünitesi

Etiket No.	Yazmaç No.	Etiket Boş	Son kopya
1	A0	H	E
2	S0	H	E
3	NIL	E	T
4	S4	H	E
5	S0	H	H
6	S3	H	E

Etiket Ünite Bilgisi			Kaynak parametre 1				Kaynak parametre 2				Hedef
Etiket No	Yazmaç	Etiket Boş	Son kopya	Hazır	Etiket	İçerik	Hazır	Etiket	İçerik	Yazmaç	

Şekil 4 RİEÜ Alanları



Şekil 5 Tomasulo Algoritması Genel Yapısı

3.2. YGÜ

Herbir işlevsel ünite için ayrı rezervasyon istasyonları kullanmak yerine bu rezervasyon istasyonları tek bir birim altında birleştirilebilir. Eğer birleştirilen rezervasyon istasyonları havuzunda yer yoksa yeni bir komut yayınlanamaz. Rezervasyon istasyonlarında bulunan ve tamamlanan komutlar ilgili işlevsel ünitelere iletilir ve algoritma bundan sonra daha standart Tomasulo algoritmasında olduğu gibi çalışmaya devam eder [2].

Ayrı rezervasyon istasyonları olduğu durumda bir işlevsel ünitenin rezervasyon istasyonları dolu olduğu için o işlevsel üniteye bağlı bir komut işletilemediği durumda bir başka işlevsel ünite rezervasyon ünitesi boş olabilir. Rezervasyon istasyonlarının birleştirilmesinin avantajı, bu tip durumların ortadan kalkmasını sağlamaktır [2].

Genişletilmiş Tomasulo algoritmasında önerilen bir başka yenilik Etiket Ünitesi ile rezervasyon istasyonları havuzunun birleştirilmesidir. Etiket Ünitesi'nde rezervasyon istasyonu ya da işlevsel ünite için ayrı birer giriş bulunduğu için rezervasyon istasyonları ya da işlevsel ünite ile Etiket Ünitesi birimleri arasında birebir eşleştirme

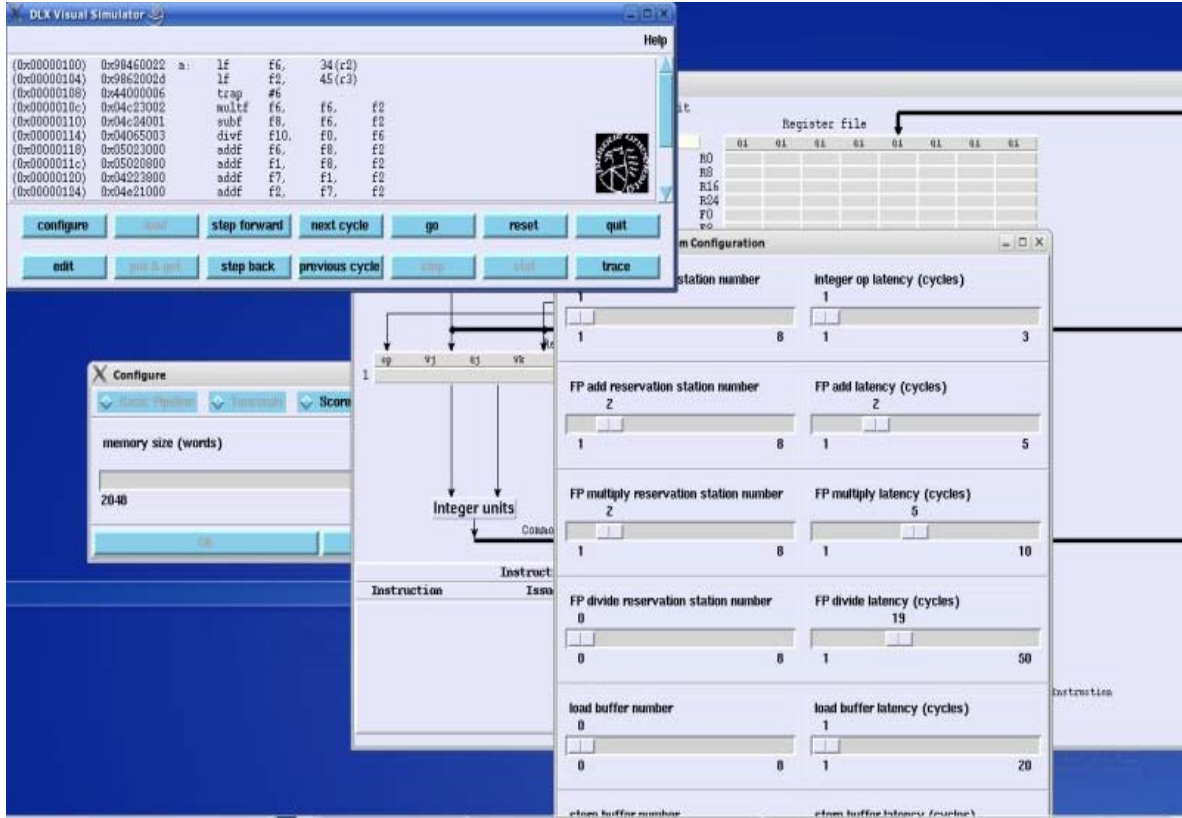
yapılabilir. Oluşturulan bu yeni ünite RİEÜ (Rezervasyon İstasyonu Etiket Ünitesi) olarak adlandırılmaktadır.

RİEÜ içinde etiket ve rezervasyon üniteleri aynı anda saklanabilir. Bir RİEÜ kaydının alanları şekil 4'te gösterilmiştir [2].

RİEÜ, yeni bir yaklaşımla komutları yayımlandığı sıraya bağlı kalarak saklayabilecek bir tampon yapısına dönüştürülebilir. Bunun için RİEÜ, dairesel bir kuyruk yapısında oluşturulur, bu yeni yapı ise YGÜ olarak adlandırılır. Her bir komut yayımlandığında, YGÜ'ya yazılır, komut tamamlandığında ise YGÜ'dan çıkarılır [2]. Eğer yeni bir komut yayınlanmak istendiğinde YGÜ'da yer yoksa, sistem bir komutun bitmesini beklemek durumundadır.

Şekil 5'te YGÜ ünitesi eklenen Tomasulo algoritmasının genel yapısı gösterilmiştir.

YGÜ içinde tamamlanmamış her komutun ve yazmaçların o anki değerlerinin bir kaydı bulunduğu için, kuraldışı durumda (exception) sistem konumunu kuraldışı durum ile karşılaşılan komuta geri alabilir. Yazmaçların son değerleri yeniden eski değerlerine getirilerek yazılır ve sistem kaldığı yerden çalışmaya devam eder.



Şekil 6 DLXView Simulatörü

Kuraldışı durumdan dönülmesi şu şekilde gerçekleşir [2].

- 1) İlgili kuraldışı durum komutu dahil olmak üzere, dairesel kuyruğun en sonuna kadar olan tüm komutların kullandığı hedef yazmaçlarına bakılır, eğer yazılmış yazmaç varsa ve daha önce bu yazmaçlar kullanılmışsa, kuraldışı durum veren komuttan önceki ilgili hedef yazmacın 'son kopya' değeri evet yapılır.
- 2) İlgili kuraldışı durum komutu dahil olmak üzere, dairesel kuyruğun en sonuna kadar olan tüm komutlar silinir.
- 3) İlgili kuraldışı durum veren komut sisteme tekrar yayınlanır.

Yukarıda anlatılan sistemin avantajı, Tomasulo algoritmasında işi biten komutların sistemden atılması durumunda yaşanabilecek kayıpları ortadan kaldırabilmesidir.

4. UYGULAMA

Genişletilmiş Tomasulo algoritması DLXView [1] adı verilen MIPS simulatörü kullanılarak uygulanmıştır. DLXView UNIX ortamında C ve Tcl/Tk kullanılarak geliştirilmiş, MIPS 32/64 mimarisini simüle

edebilen açık kaynak kodlu bir projedir. Bünyesinde “Basic MIPS Pipeline”, “Scoreboard” ve “Tomasulo” algoritmalarını barındırmaktadır. Temel olarak sanal bir işlemcinin kodları çalıştırdığı alt yapı üzerine, örnek MIPS kodlarını çalıştırabilecek grafiksel bir arayüze sahiptir. (Şekil 6) DLXView adı verilen MIPS simulatöründe kullanılan en önemli yapılar aşağıdaki gibidir.

- 1) Çalışma anında kullanılan her bir komut aşağıdaki C yapısı ile ifade edilmektedir.

typedef struct Op {

```
// Komutun 1. ve 2. kaynak operandlarının
// yazmaç numaraları
int rs1, rs2;
```

```
// Komutun 1. ve 2. kaynak operandlarının
// hazır olacağı dönüş (cycle) sayısı
int rs1Ready, rs2Ready;
```

```
// Kaynak 1 ve 2 değeri
int source1[2], source2[2];
```

```
// Hedef yazmaç numarası
int rd;
```

```
// Hedef registerin hazır olacağı dönüş
// numarası
int rdReady;
```

```

// Sonuç register numarası
int result;

// Komutun hangi işlevsel birime ait olduğu
int unit;

// Yükle/Sakla komutları için verinin
okunacağı veya saklanacağı adres
unsigned int address;

// Sonraki komut
struct Op *nextPtr;
} Op;

```

2) İşlemciye eklenen YGÜ, rezervasyon istasyonlarının her biri aşağıdaki gibidir.

```

typedef struct TagUnitInfo
{
    // Etiket numarası
    int tagNo;

    // Etiket boş mu ?
    int tagFree;

    // Yeniden adlandırılmış yazmaçın son kopyası mı ?
    int latestCopy;

    // Komut
    Op *op;
} TagInfo;

```

3) YGÜ birimi işlemci yapısına aşağıdaki gibi eklenmiştir.

```

typedef struct
{
    .... ( İşlemci ile ilgili yapılar)

    // MAX_RUU_COUNT kadar maximum rezervasyon
    istasyonu sayısı
    TagInfo RUU[MAX_RUU_COUNT];

    // Kullanılabilen YGÜ rezervasyon istasyonu sayısını
    ilerde dinamik olarak belirlemek için RUUSize (<=
    MAX_RUU_COUNT)
    int RUUSize;

    // YGÜ'nde çalışma anında dairesel kuyruk yapısının
    başlangıç noktası
    int RUUHead;

    // YGÜ'nde çalışma anında dairesel kuyruk yapısının
    bitiş noktası
    int RUUTail;
} DLX;

```

4) YGÜ'nde ilgili yazmaçla ilişkilendirilmiş etiket numarası arayan kod aşağıdaki gibidir.

```

static
int FindTagForRegister(DLX* machPtr, int registerNo)
{
    int i;
    if(!FirstTime)

```

```

{
    for (i=machPtr->RUUTail;
    i != mod(machPtr->RUUHead-1,machPtr->RUUSize);
    i=mod(i-1,machPtr->RUUSize))
    {
        if((registerNo == machPtr->RUU[i].op->rd) &&
        (machPtr->RUU[i].latestCopy > 0))
            return i;
    }
    return -1;
}

```

5) Aşağıdaki kodda bir yazmaç yazılacağı zaman ilgili yazmaçın beklendiği alanlara bekleme miktarları yazılmaktadır.

```

static
void WriteRegisterToTagUnit(DLX* machPtr, int
registerNo,int cycleCount)
{
    int i;
    for (i=machPtr->RUUTail; i != machPtr->RUUHead;
    i=mod(i-1,machPtr->RUUSize))
    {
        if(registerNo == machPtr->RUU[i].op->rs1)
            machPtr->RUU[i].op->rs1Ready = cycleCount;

        if(registerNo == machPtr->RUU[i].op->rs2)
            machPtr->RUU[i].op->rs2Ready = cycleCount;
    }
}

```

6) Aşağıdaki kodla verilen bir register için, YGÜ'nde bu register için bekleme miktarları okunmaktadır.

```

static int
ReadRegisterReadyFromTagUnit(DLX* machPtr,int
registerNo)
{
    int tagNo;
    tagNo = FindTagForRegister(machPtr,registerNo);
    if(tagNo == -1)
        return -1;
    return machPtr->RUU[tagNo].op->rdReady;
}

```

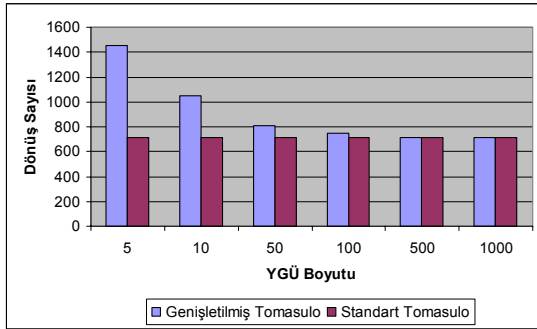
7) Her komut yayınlandığında ilk olarak komutun işlevsel ünite tipine göre gecikmeleri, kaç adet ünite olduğu belirlenir. YGÜ'ne ait dairesel kuyruk yapısının kuyruk kısmı bir adım ilerletilir. Komuta ait ilk değerler atanarak, komut YGÜ birimine, kuyruktaki etiket numarasına atanır. Yapısal hazardlar kontrol edilir, bir ünite boşalınca kadar beklenir. Hedef yazmacı o anki komutun kendisi ile aynı, kendi üstünden başlayarak bir YGÜ rezervasyon istasyonu

aranır, ve böyle bir rezervasyon istasyonu bulunduğunda ‘son kopya’ değeri HAYIR yapılır, kendi ‘son kopya’ değeri EVET yapılır. Komutun çalıştırılmaya başlaması için dairesel kuyruğun başındaki komutun bitirilmesi beklenir ve son olarak komut çalıştırılır.

2 tamsayı (integer), 2 kayan noktalı (floating-point) ekleme, 2 kayan noktalı çarpma, 1 kayan noktalı bölme işlevsel ünitesi, 1 yükleme tamponu ve 1 saklama tamponu bulunduğu durumda Program1 için hem değişen YGÜ boyutlarına göre (5, 10, 50, 100, 500, 1000) geliştirilmiş Tomasulo algoritmasında hem de standart Tomasulo algoritmasında 32 dönüş sürmüştür. Program1 kodu aşağıda verilmiştir.

```
foo:   lf      f2, 0(r1)
      multf  f4, f2, f0
      lf      f6, 0(r2)
      addf   f6, f4, f6
      sf     0(r2), f6
      addi   r1, r1, 8
      addi   r2, r2, 8
      sgti  r3, r1, done
      beqz  r3, foo
      trap  #0 //program bitişini belirtir
```

Şekil 7’de aynı sayıda işlevsel ünite, ve aynı sayıda yükleme tamponu ve saklama tamponu bulunduğu durumda Program2 için değişen YGÜ boyutuna göre dönüş sayılarını belirten grafik verilmiştir. Program2 kodu aşağıda verilmiştir. Aynı kod, standart Tomasulo algoritması için 718 dönüşte sona ermiştir.

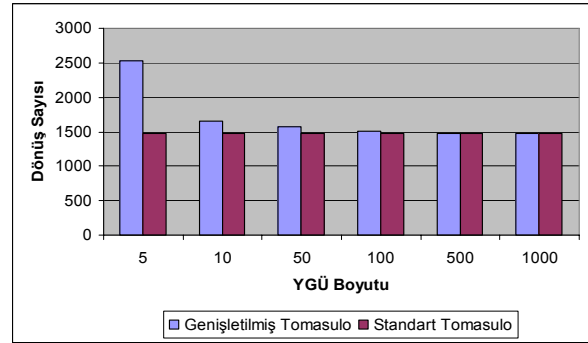


Şekil 7 1.durum için 2. programın karşılaştırması

```
lf      f6,      34(r2)
lf      f2,      45(r3)
multf   f0,      f2,      f4
subf    f8,      f6,      f2
divf    f10,     f0,      f6
addf    f6,      f8,      f2
trap    #0
```

1 tamsayı, 2 kayan noktalı ekleme, 2 kayan noktalı çarpma, 0 kayan noktalı bölme işlevsel ünitesi, 0 yükleme tamponu ve 0 saklama tamponu bulunduğu durumda Program1 için hem değişen YGÜ boyutlarına göre (5, 10, 50, 100, 500, 1000) geliştirilmiş Tomasulo algoritmasında hem de standart Tomasulo algoritmasında 32 dönüş sürmüştür.

Şekil 8’de aynı sayıda işlevsel ünite, ve aynı sayıda yükleme tamponu ve saklama tamponu bulunduğu durumda Program2 için değişen YGÜ boyutuna göre dönüş sayılarını belirten grafik verilmiştir. Aynı kod, standart Tomasulo algoritması için 1482 dönüşte sona ermiştir.



Şekil 8 2.durum için 2. programın karşılaştırması

Her iki durum için de YGÜ boyutu 500 olduğunda dönüş sayılarının eşit olduğu görülmektedir.

5. SONUÇ

Uygulanan YGÜ ünitesinin performansı YGÜ boyutlarına bağlı olarak değişmektedir. Yeterli bir boyuta sahip olduğu durumlarda kuraldışı durumlarda kurtarma sağlanmasına rağmen standart Tomasulo algoritması ile eşit dönüş sayısında programları tamamlamaktadır. YGÜ boyutu özellikle döngü içeren programlarda performansı etkilemede önemlidir. Döngülerin çalışması sırasında döngü içindeki aynı komutlar tekrar yayınlanır. YGÜ’nde ise ilk sırada bulunan komutlar tamamlanmadan dairesel kuyruğun başı ile sonu arasındaki komutlar tamamlanmış olsa da YGÜ’nden çıkarılamaz. Bu sebeplerden dolayı pencere boyutuna sığmayan komutlar nedeniyle dönüş sayısı artmaktadır, başka bir deyişle performans düşmektedir.

6. KAYNAKLAR

[1]. Hostetler, L.B. ve Mirtich, B., “DLXSim – A Simulator for DLX”, Technical Report, 1998.

- [2]. Sohi, G.S., "Instruction Issue Logic for High-performance, Interruptible, Multiple Functional Unit, Pipelined Computers", *IEEE Transaction of Computer*, 39(3):349-359, 1990.
- [3]. Tomasulo, R.M., "An Efficient Algorithm for Exploiting Multiple Arithmetic Units", *IBM Journal of Research and Development*, 25-33, 1967.
- [4]. Vijeyan, B., Rajendran, M., ve Veluswami, S., "Out-of-order Commit Logic with Precise Exception Handling for Pipelined Processors", International Conference on High Performance Computing, 2002, Hindistan.