

Genişletilmiş Tomasulo Algoritması ve

Kuraldışı Durumların İşlenmesi

Müge Sayıt, Ahmet Bilgili

Ege Üniversitesi, Uluslararası Bilgisayar Enstitüsü, 35100, İzmir
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 simülatöründe 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. Giriş

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 risk-

lerini (WAR ve WAW) ortadan kaldırmaktadır. Scoreboard algoritmasında hedef yazmaçlar yeniden isimlendirilemediği için tekrar kullanılmaya 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 Algoritması

Tomasulo algoritması işletilirken, komutları sırayla yayınladıktan (issue) sonra parametre-

rinin hazır olup olmadığına bakılır. Eđer parametreler hazır deęilse komut, bu parametreler tarafından kullanılacak olan iřlevsel ünitelerin rezervasyon istasyonlarında beklemeye alınır. Komutlarda yer alan yazmaçlar, rezervasyon istasyonlarındaki deęerler ile eřlenir. Bu iřleme 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 saęlar ve bu sebeple iřlemcide oluřabilecek beklemlerin önüne geęer [3].

İřlevsel ünitelerden çıkan sonuçlar ortak veri hattı ile tüm bekleyenlere iletilir. Bir komut içerisindeki tüm parametreler bu yolla hazır olduęunda, komutun gerektirdięi iřlevsel üniteyi kullanarak iřlem yapar [3]. İřlem tamamlandıęında, iřlevsel ünite serbest kalır, hedef yazmaçta iřlem sonucu yazılır.

Her bir kaynak yazmaç için, o yazmacın meřgul olup olmadığını belirten bir bit bulunmaktadır. Bir yazmaç, çalışmakta olan bir komutun hedef yazmacı konumunda ise meřguldür. Bir rezervasyon istasyonunun alanları Őekil 1'de gösterilmiřtir.

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

Őekil 1 Rezervasyon İstasyonu Alanları

Eđer bir kaynak yazmaç, komut yayınlanma ařamasına geçtięinde meřgulse, bu yazmaç için kullandığı iřlevsel üniteyi belirten bir etiket (tag) verilir ve komut rezervasyon istasyonuna alınır. Eđer hedef yazmaç meřgulse, o yazmaç için belirlenen etiket rezervasyon istasyonuna yazılır. Bu etiketlerin belirttięi iřlevsel üniteler hazır olduęunda, komut tamamlanmış olur ve rezervasyon istasyonundan çıkarılır. Tomasulo algoritmasının 3 adımı kısaca Őu Őekilde özetlenir [3]:

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

2.Çalıştırma: Her iki parametre de hazırrsa, 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 olduęunu belirtecek Őekilde iřaretlenir.

3. Geniřletilmiř Tomasulo Algoritması

3.1. Etiket Ünitesi

Standart Tomasulo algoritmasında, hedef yazmaçlar için etiket belirleme iřlemi yapılmaktadır. Her yazmaç, komutların iřleniři sırasında hedef yazmaç olabileceęi için her birine ayrı etiket verilmektedir. Yazmaç ile etiketi arasında eřleřtirme yapabilmek için ise bu iřlemi geręekleřtirebilmeyi saęlayacak bir karřılařtırma yapısına ihtiyaç duyulmaktadır [2].

Geniřletilmiř Tomasulo algoritmasında, yazmaç ve etiket eřleřtirme iřleminin getirdięi 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 birleřtirilerek Etiket Ünitesi oluřturulmuřtur [2]. Eđer kaynak yazmaç meřgulse, Etiket Ünitesi bu yazmaç için bir etiket atayarak bunu rezervasyon istasyonuna iletir. Komut yayın ařamasında eđer uygun etiket yoksa yani Etiket Ünitesi dolu ise komut beklemeye alınır.

Etiket ünitesinde bir yazmaç için birden fazla etiket verilmiş olabilir. Yapılan iřlemlerin karışmaması için yazmaçlar için sadece en son verilen etiket deęeri dikkate alınır [2]. Etiket ünitesinin eklenmesinden sonra oluřturulan bir rezervasyon istasyonunun yapısı Őekil 2'de gösterilmiřtir.

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

Őekil 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

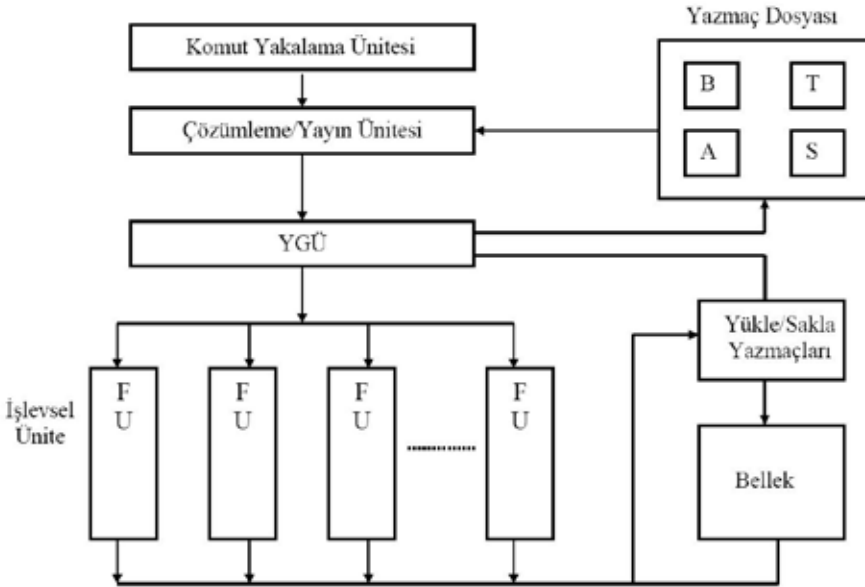
[2]. Tabloda E, H ve T sırasıyla evet, hayır ve tanımsız anlamında kullanılmıştır.

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

Tablo 1 Örnek bir Etiket ünitesi

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].

Geniřletilmiř Tomasulo algoritmasında önerilen bir bařka yenilik Etiket Üniteleri ile rezervasyon istasyonları havuzunun birleřtirilmesidir. Etiket Üniteleri'nde rezervasyon istasyonu ya da iřlevsel ünite için ayrı birer giriř bulunduđu için rezervasyon istasyonları ya da iřlevsel ünite ile Etiket Üniteleri birimleri arasında birebir eřleřtirme yapılabilir. Oluřturulan bu yeni ünite RİEÜ (Rezervasyon İstasyonu Etiket Üniteleri) olarak adlandırılmaktadır.

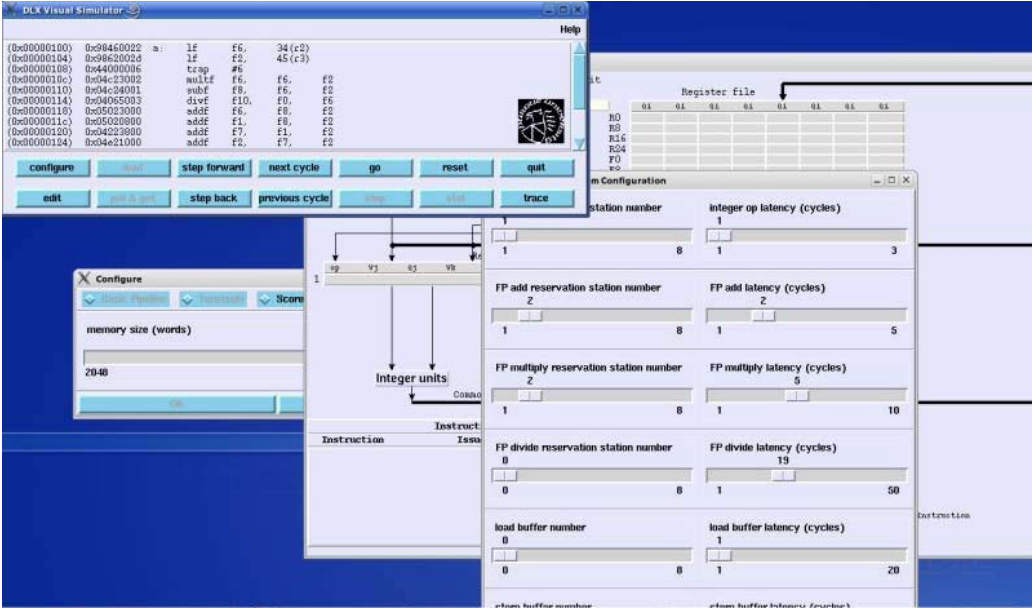
RİEÜ içinde etiket ve rezervasyon üniteleri aynı anda saklanabilir. Bir RİEÜ kaydının alanları Őekil 4'te gösterilmiřtir [2].

RİEÜ, yeni bir yaklařımla komutları yayınladığı sıraya bađlı olarak saklayabilecek bir tampon yapısına dönüřtürülebilir. Bunun için

RİEÜ, dairesel bir kuyruk yapısında oluřturulur, bu yeni yapı ise YGÜ olarak adlandırılır. Her bir komut yayımlandığında, YGÜ'ya yazılır, komut tamamlandığında ise YGÜ'dan çıkarılır [2]. Eđer yeni bir komut yayınlanmak istendiğinde YGÜ'da yer yoksa, sistem bir komutun bitmesini beklemek durumundadır.

Őekil 5'te YGÜ ünitesi eklenen Tomasulo algoritmasının genel yapısı gösterilmiřtir.

YGÜ içinde tamamlanmamıř her komutun ve yazmaçların o anki deđerlerinin bir kaydı bulunduđu için, kuraldıřı durumda (exception) sistem konumunu kuraldıřı durum ile karřılařılan komuta geri alabilir. Yazmaçların son deđerleri yeniden eski deđerlerine getirilerek yazılır ve sistem kaldığı yerden çalıřmaya devam eder.



Őekil 6 DLXView Simülatorü

Kuraldıřı durumdan dönülmesi řu Őekilde gerçekleřir [2].

1) İlgili kuraldıřı durum komutu dahil olmak üzere, dairesel kuyruđun en sonuna kadar olan tüm komutların kullandığı hedef yazmaçlarına bakılır, eđer yazılmıř yazmaç varsa ve daha

önce bu yazmaçlar kullanılmıřsa, kuraldıřı durum veren komuttan önceki ilgili hedef yazmaçın 'son kopya' deđerleri evet yapılır.

2) İlgili kuraldıřı durum komutu dahil olmak üzere, dairesel kuyruđun 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 simülatörü kullanılarak uygulanmıştır. DLXView UNIX ortamında C ve Tcl/Tk kullanılarak geliştirilmiş, MIPS 32/64 mimarisini simule 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 çalıştıracak grafiksel bir arayüze sahiptir. (Şekil 6)

DLXView adı verilen MIPS simülatöründe 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 ileride 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;
}
return -1;
}
```

5) Ařağıdaki kodda bir yazmaç yazılacağı zaman ilgili yazmaçın beklendiğı 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) Ařağıdaki kodla verilen bir register için, YGÜ'nde bu register için bekleme miktarları okunmaktadır.

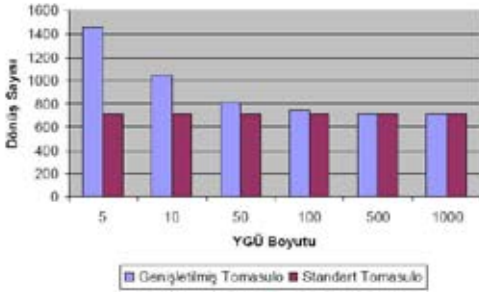
```
static int
ReadRegisterReadyFromTagUnit (DLX*
machPtr,int registerNo)
{
int tagNo;
tagNo = FindTagForRegister(machPtr,r
egisterNo);
if(tagNo == -1)
return -1;
return machPtr->RUU[tagNo].op->rdReady;
}
```

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 bitiřini 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.

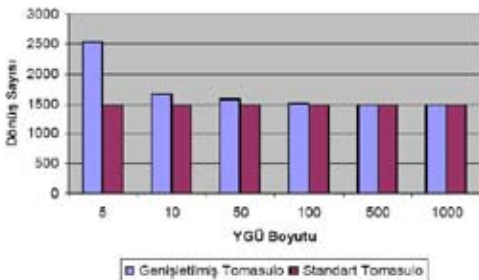


Şekil 7 1.durum için 2. programın karşılaştırması

If

```
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]. Vijayan, 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.