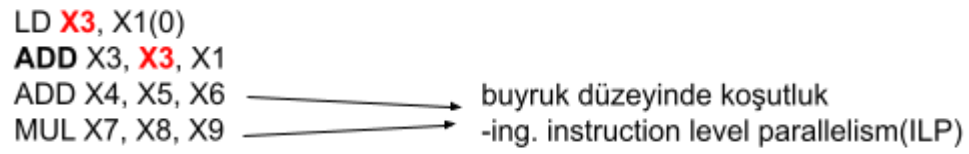
Sıralı Boru Hattı

Sıralı yürütüm yapan bir boru hattında yürütme aşamaları farklı çevrim sürdüğünden program sırasının kısmen dağılabildiğini, tam dağılmadığını ama yazmaç güncellemesinin program sırası dışında yapılmasının gerekli olabileceğinden bahsettik. Böyle olunca da tekrar program sırası içerisine dönmek için PST(program sırası tablosu) diye bir araç kullanmamız gerektiğinden bahsettik. Bu şekilde olunca yürütme aşaması program sırasına göre içeri giriyor ama program sırasına göre çıkmıyor. Yürütme aşamasının farklı çevrimlerde tamamlanmasıdan dolayı.

Dağıtım(-ing. dispatch): Buyrukların yürütme birimlerine (-ing. execution unit) dağıtılması. İşlemcilerin içerisinde tek bir aritmetik mantık birimi olabildiği gibi bağımsız işlem birimleri de olabilir. Güncel işlemcilerde aritmetik mantık birimi var ama sayma sayıları için ayrı aritmetik mantık birimi var, virgüllü sayılarla işlem yapmak için ayrı mantık birimi var. Ayrıca AGU(address generation unit) diye birim var; Adreslerin hesaplanması için ayrıca bir işlem birimi. Her işlemcinin işlem birimleri kendi ihtiyacına göre olur. Güncel işlemcilerin birden fazla sayma sayılarla işlem yapan aritmetik mantık birimi, birden fazla da virgüllü sayılarla işlem yapan aritmetik mantık birimi vardır. Buyruk geldiğinde bunlar arasında hangisine atama yapılacak? Bu sorunun cevabına dağıtım deniyor.
Yeniden adlandırma gerçekte olmayan bağımlılıkları ortadan kaldırır.

Gerçek bağımlılıklar **genç**(kısa zamandır boru hattında) buyrukların dağıtımını engelleyebilir. Çünkü öndeki bağlı olduğu buyruk henüz sonucunu üretmediyse, arkadaki buyruk gidip de bir işlem birimine atanamaz, işlem birimine atanamayınca beklenir. Program sırası içerisinde yaptığımız zaman işlemleri, yürütme işlemi başladığı zaman eğer bir buyruk öndekini bekliyorsa arkadakilerin tamamı da onu bekler. Sırasız yürütümün düşünülmesinin nedeni de budur.



ADD buyruğu tüm boru hattını duraklatıyor.

- ADD'in dağıtımı gerçekleştirilemiyor. (X3 hazır değil)
- sonraki buyruklar bağımsız, ancak onların da dağıtımı gerçekleştirilemiyor.

3. ve 4. buyrukların veri bağımlılığı yok. Ama ADD buyruğunun beklemesinden dolayı onlar da beklemek zorunda kalıyor. Veri bağımlılıkları olmadığı için aslında bağımsız olarak çalışabilir durumdadırlar. Buna ***buyruk düzeyinde koşutluk*** denir. Bunun düzeyi sırasız yürütümden ne kadar faydalanabildiğini etkiler. Programda ne kadar ileriye bakabilirsek daha fazla bağımsız buyruk bulma ihtimalimiz artar.(örn; daha fazla buyruğu boru hattı içerisine alarak, daha fazla buyruk keşfedilir)

Sıralı Boru Hattının Sorunu
Sorun: dağıtımın sırayla(program sırası) yapılması
Çözüm: dağıtımı sırasız yapmak

Diğer çözümler:

- derleyicinin buyrukların yeniden sıralaması
- değer tahmini (bağımlılıklar tahmin ile çözülebilir)

Program sırasıyla atmak yerine arkadan gelen buyruklar eğer öndekilere bağlı değillerse yürütmeye başlasınlar, sonuçlarını yazsınlar, öndeki tıkanıklık çözülünce sonuçlar hazır şekilde beklesin. Burada bazı sorunlar ortaya çıkabilir. Derleyici buyrukların sırasını değiştirmeye kalkabilir. ya da tahmin yapılarak bekleme ortadan kaldırılabilir. Güncel işlemcilerde bunlara rastlanmaz. Kullanıcı girişine bağlı buyruklar olduğu için derleyici bunları tahmin edemez. Örneğin; csGO oynarken tuşlarla hareket sağlamak gibi.

Sırasız Yürütme(-ing. Out-of-Order Execution)

Ana fikir: Veri bekleyen verisi hazır olan buyrukların “önünden çekmek”. Bağımlı olan bir şey bekleyen bir buyruk var, arkasındaki bağımsız buyruklara siz yürüyün siz devam edin işimizi tamamladığımızda size yetişiriz diyor.

Sırasız yürütmenin 2 temel aşaması vardır:

1.) Kaynak yazmaçları hazır olana kadar buyrukları bekletmek. Her buyruk yürütmeye başlamadan önce kaynak yazmaçları hazır olana kadar bekliyor. Artık programların yürütülmesi program sırasına bağlı olmuyor, yürütülmesi bağımlılıklara bağlı oluyor.
2.) Kaynak yazmaçları hazır olan buyrukların dağıtımını gerçekleştirmek

```
LD X3, X1(0)
ADD X3, X3, X1
MUL X4, X5, X6
ADD X7, X4, X9
```

- Sıralı yürütme (tam doğru kural dışı durumlar)

12 çevrim

- Sırasız yürütme (tam doğru kural dışı durumlar)

10 çevrim

Sırasız Yürütme için ne gerekli?

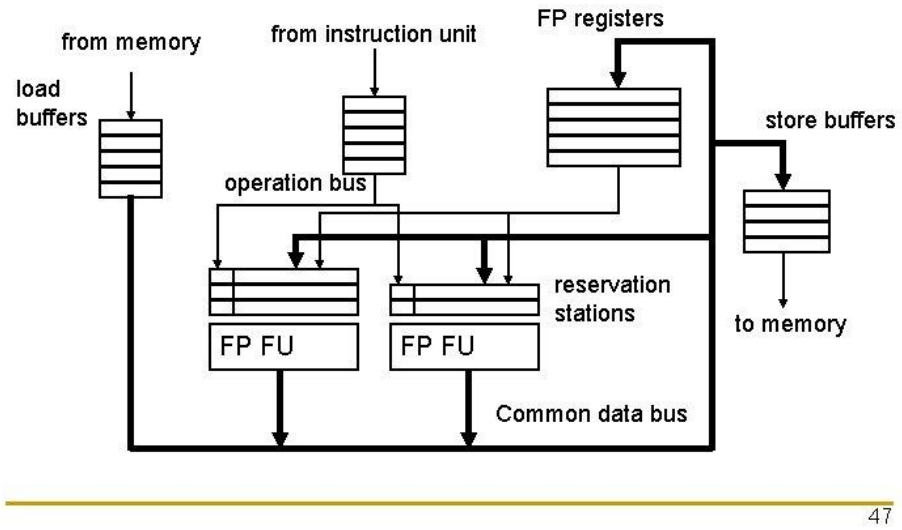
1. kaynak değerleri ile hedef değerleri arasında bağlantı sağlanmalı
    - yazmaç yeniden adlandırma: veri ile bir etiket eşleştirilir
2. buyrukların hazır olana kadar bekledikleri bir ara bellek
    - Hazır: kaynak değerleri hazır
    - ***ara bellek:*** (-ing. reservation station(tomasulo), issue window, instruction window, dispatch buffer, issue queue) daha doğru çevrim için bekleme durağı diyebiliriz. Buyrukların kaynak yazmaçların değerleri hazır olana kadar bekledikleri bir ara bellek, alan. Buyruklar kaynak değerlerinin hazır olduğunu nasıl anlayacak?
    - önceden gelen buyruklar sonuçlarını bitirdiklerinde şu anki buyruğun bundan haberdar olması lazım. Benim beklediğim değer artık hazır oldu, veri yönlendirmesi ile telden alacağım veya yazmaçtan alacağım nerden alınacaksa haberi olması lazım
3. Buyruklar kaynak değerlerini takip etmeli
    - buyruklar ara bellekte ***kaynak etiketleri*** ile beraber tutulur
    - bir yazmaç güncellendiğinde ara bellekteki tüm buyruklara etiketi iletilir
    - buyruklar kaynak etiketleri ile güncellenen yazmacın etiketini karşılaştırır, aynı olan kaynak etiketleri ***hazır*** olarak işaretlenir
4. Hazır buyrukların yürütme birimlerine dağıtılması
    - tüm kaynakları hazır olan buyruk uyanır(-ing. wake up)
    - birden çok buyruk uyanıksa(hazırsa) yürütme birimi sayısına göre aralarından seçilir

Bir Sırasız Yürütme Yöntemi: Tomasulo'nun Algoritması

Yazmaç yeniden adlandırma kullanan bir sırasız yürütme yöntemi. Günümüzdeki yöntemlerden temel farkı: Tam doğru kural dışı durumları desteklemiyor. Yani bir kural dışı durum olduğunda program sırasını tutacak bir mekanizma o zamanlarda düşünülmemiş. Ama o zamanlarda dallanma öngörüsünü düşünmüşler. Kural dışı durumlar olduğunda tam doğru işleyemiyor ve her şeyi atması gerekiyor.

Moder Yüksek-başarımlı işlemcilerin hepsi sırasız yürütme yöntemi kullanıyor:

- ilk kullanıcılar: Pentium PRO, AMD K5
- sonradan: Alpha 21264, MIPS R10K, IBM POWER5, IBM z196...



Bütün işlem birimleri virgüllü sayılardan oluşuyor. Tomasulo demiş ki: program sırasını bırakıyorum ilerlesin istiyorum. Bütün işlem birimleri önüne bekleme alanları koyarım (reservation stations). Buraya program sırası ile girerler(from instruction unit) ama içinde program sırası ile tutulmuyorlar çünkü program sırası dışında çıkabilirler bundan dolayı da herhangi bir yerine yazılabilirler. Hangi işlem biriminin bekleme alanında ise buyruk o işlem birimine girer. Çarpma işlemleri çarpma birimine, toplama işlemleri toplama birimine girer. Böylece dağıtılmış bir ara bellek mimarisi oluşturulmuş oluyor. Güncel mimarilerde bu bekleme alanı ortak. Bazen virgüllü sayılarla sayma sayıları ayıranlar var.

Common data bus: bellekten gelen bir değer varsa veya işlem biriminde bitirilen bir değer varsa bu değeri okumak isteyen buyruklara aktarmak için, veri yönlendirmesi için aslında ve de götürüp yazmaçlara yazmak için. Hem yazmaçlara yazmak hem de bekleyenlere bildirmek için diyebiliriz.

Tomasulo'nun Algoritmasına göre Yeniden Adlandırma

Yazmaç adlandırma tablosu(-ing. Register Aliasing Table - RAT)
Geçerli: yazmacın değeri geçerli (en güncel değer yazmaç öbeğinde)
Etiket: bu değeri üretecek reservation station elemanının numarası

Yazmaç	**Geçerli**	**Etiket**	Değer
R1	1	X	geçerli
R2	0	geçerli	geçersiz
R3			
R4			
R5			

Bütün yazmaçlar için 1 satırı olan bir tablodur. Yazmaç eğer geçerli ise etikete ihtiyaç olmuyor. Çünkü o değer yazmaçtan okuyor. Eğer yazmaç geçerli değilse etiket geçerli olabilir. O zaman bir değere ulaşmak etiketi kullanıyor. Değerin geçerli ya da geçersiz olduğunu bilmiyoruz. O değer daha hazırlanmamış olabiliyor. Henüz o değerin hazır olup olmadığını bilmiyoruz.

Bekleme alanlarında yer varsa:

- buyruğu ve yeniden adlandırılmış etiketleri bekleme alanlarından birine yerleştir.

yoksa: boru hattını duraklat.

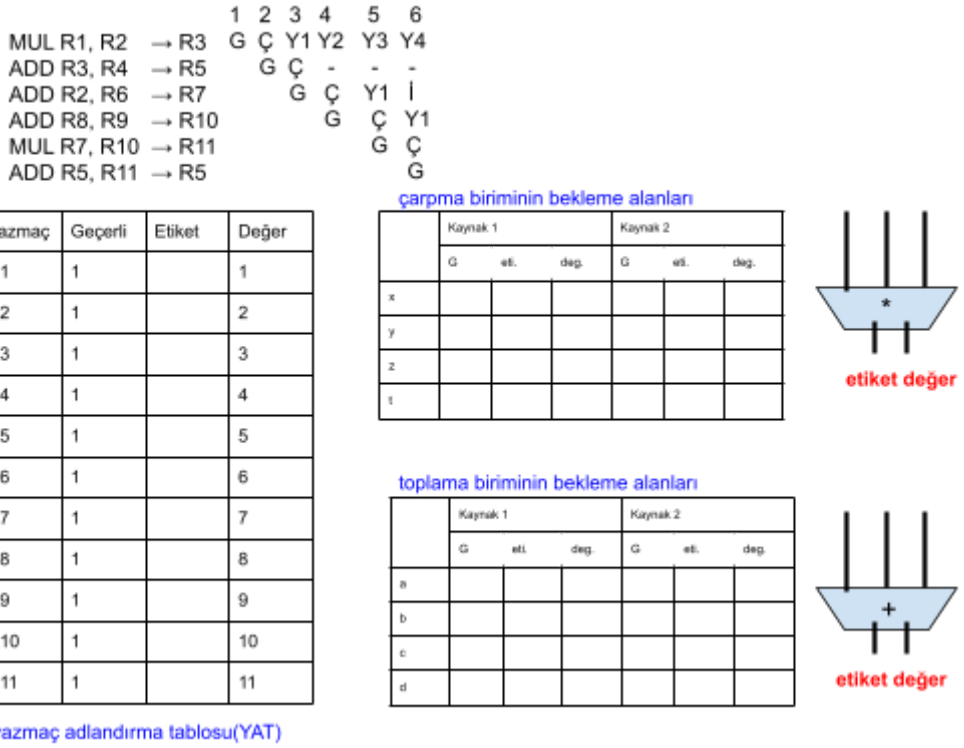
Bekleme alanlarındaki buyrukların her biri :

- Veri ağına(-ing. common data bus) bakar ve yeni hesaplanan etiketleri kaynak etiketleri ile karşılaştırır. Ben şimdi buyruk olarak bekliyorum kaynak yazmaçlarım var veya etiketleri var numaraları var, işini bitirenlerin numarası benimkiyle aynı mı? Ben bunu mu bekliyorum? diye o tel üzerindeki değerle karşılaştırıyor. Eğer tutarsa o değeri alır.
- eşleşirse veriyi bekleme alanında karşılık gelen kaynak değere yazar

Kaynak değerleri hazır olan buyrukların yürütme birimlerine dağıtımı gerçekleştirilir.
Yürütme biriminden çıkan buyruklar(sonucu hazır buyruklar):

- veri ağını kullanmak için sıra bekler
- verinin etiketini veri ağına iletir
- yazmaç öbeği veri ağını dinler:
    - etiketi eşleşen yazmaç değerleri güncellenir. ***Etiket:*** yazmacı güncelleyecek buyruğu belirtiyor.

- veri etiketi(bekleme alanındaki) uygun hale döner



Yazmaç	Geçerli	Etiket	Değer
R1	1		1
R2	1		2
R3	1		3
R4	1		4
R5	1		5
R6	1		6
R7	1		7
R8	1		8
R9	1		9
R10	1		10
R11	1		11

yazmaç adlandırma tablosu(YAT)

Başlangıç durumu:

- bekleme alanı elemanları **geçersiz**
- tüm yazmaç değerleri güncel: **geçerli**

Yürütme gecikmeleri:

- MUL 6 çevrim
- ADD 1 çevrim

ADD ve MUL birimlerinin ayrı veri ağları var ve bunlar farklı kablolar ile yazmaç öbeğine ve bekleme alanlarına bağlanmışlar

İlk çevrim:

sadece ilk buyruk getirildiği için daha çözmeye geçilmedi yapılan bir şey yok.

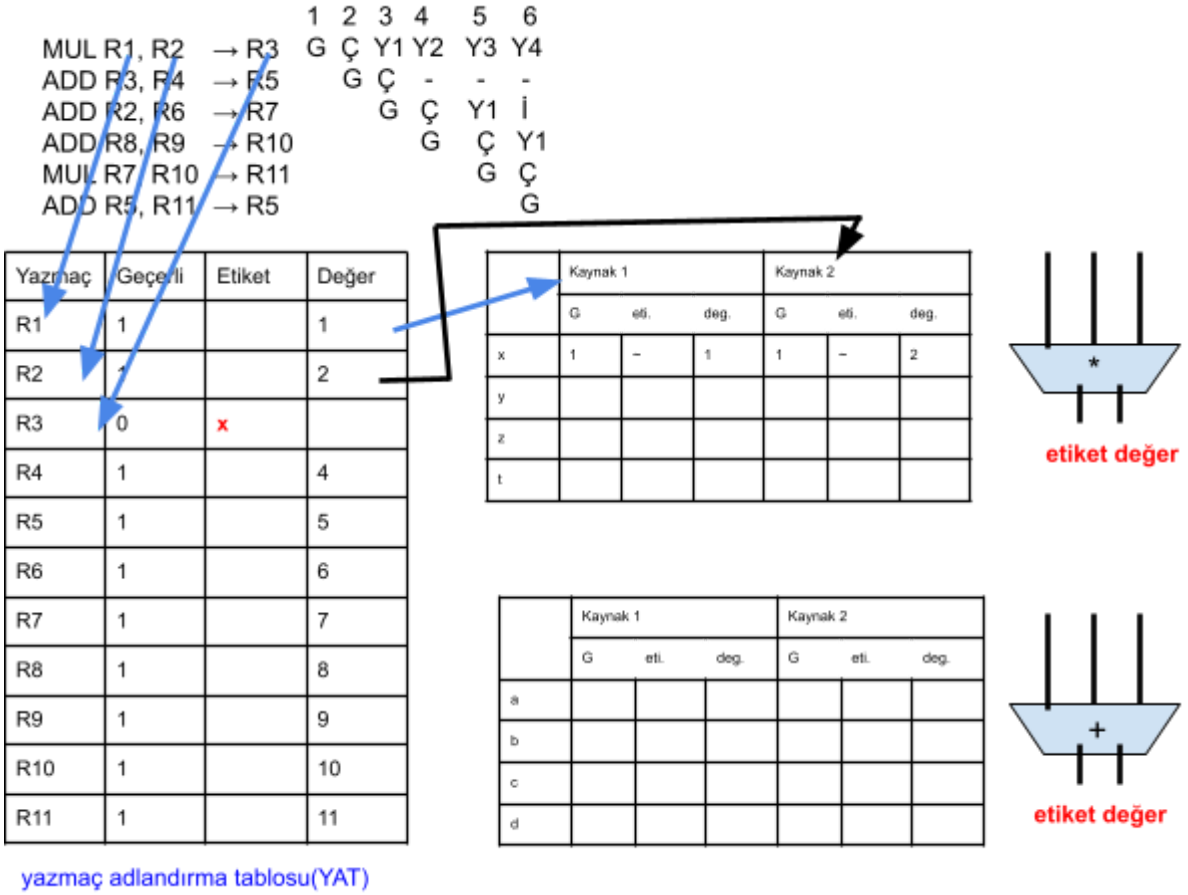
ikinci çevrim:

MUL'u çöz ve bekleme alanı ayır

1. adım: bekleme alanı bul. boş yer var
2. adım: YAT'a eriş
3. adım: kaynak yazmaç değerlerini x'e yerleştir
4. adım: R3'ü yeniden adlandır. R3 → x

R3'ün yeni ismi artık artık x: x etiketine yerleştirilen buyruk R3'ün en güncel halini hesaplayacak

x'teki MUL bir sonraki çevrim yürütülebilir. Tüm yazmaç değerleri hazır.



yazmaç adlandırma tablosu(YAT)

Üçüncü çevrim:

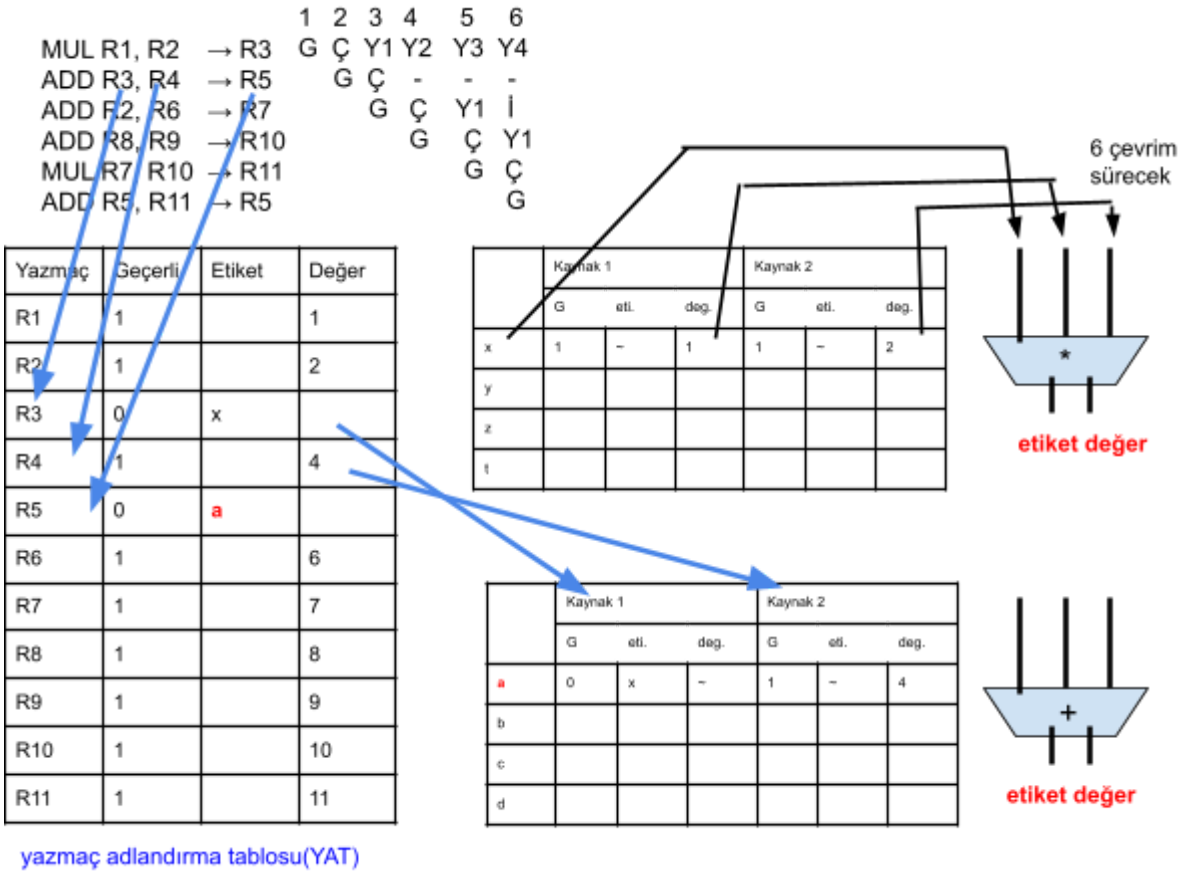
x'teki MUL'u yürüt

ADD'i çöz ve bekleme alanı ayır

x için kaynak değeri **hazır** mı?

Hazır. x'teki buyruğun dağıtımını gerçekleştir.

2.ADD için 1-4.adımları tekrar çalıştır.



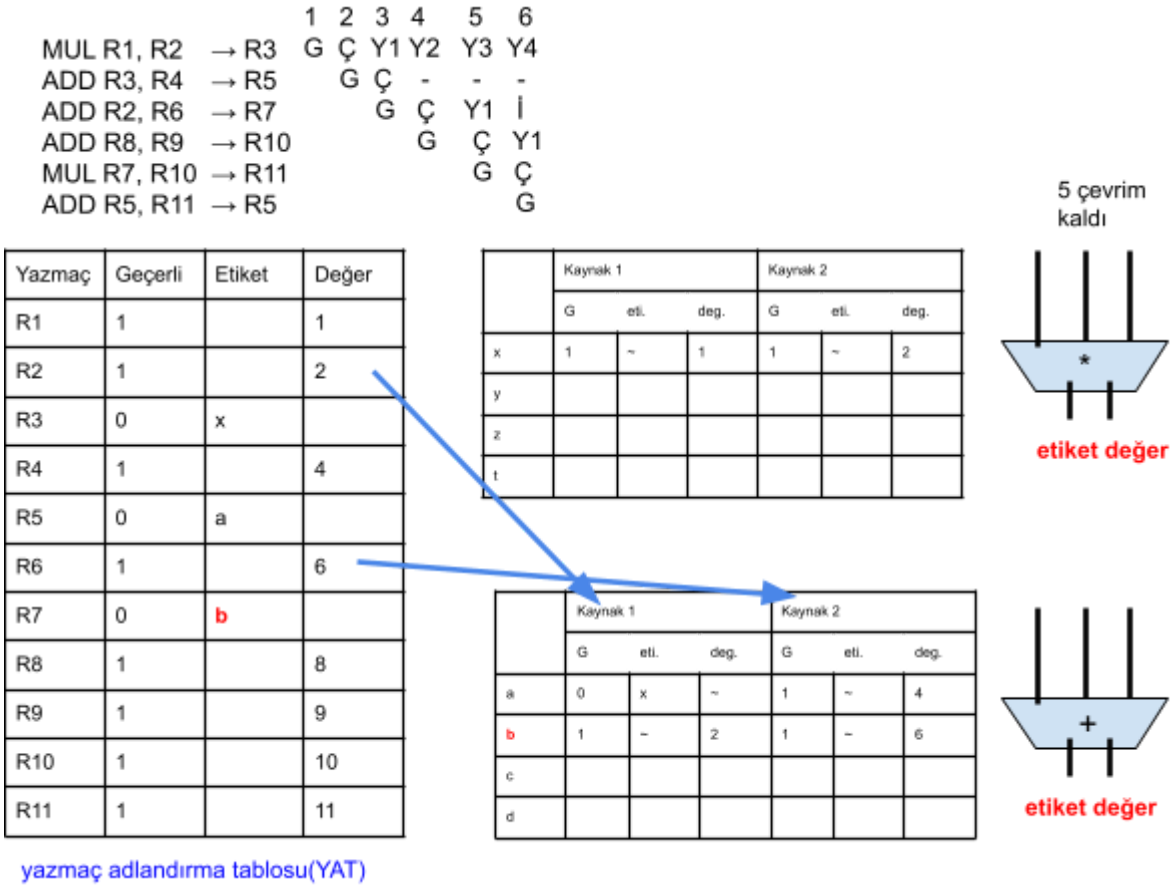
yazmaç adlandırma tablosu(YAT)

a'daki ADD bir sonraki çevrim yürütülemez. Kaynak 1 hazır değil.

Dördüncü çevrim:

a'daki ADD hala bekliyor

yeni ADD'i çöz ve yerini ayır



b'deki ADD bir sonraki çevrim yürütülecek: Tüm kaynak yazmaçları hazır.

Beşinci çevrim:

a'daki ADD hala bekliyor
b'deki ADD yürütülüyor
yeni ADD'i çöz ve yerini ayır

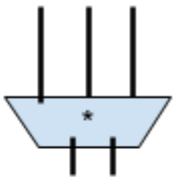
	1	2	3	4	5	6
MUL R1, R2 → R3	G	Ç	Y1	Y2	Y3	Y4
ADD R3, R4 → R5		G	Ç	-	-	-
ADD R2, R6 → R7			G	Ç	Y1	İ
ADD R8, R9 → R10				G	Ç	Y1
MUL R7, R10 → R11					G	Ç
ADD R5, R11 → R5						G

Yazmaç	Geçerli	Etiket	Değer
R1	1		1
R2	1		2
R3	0	x	
R4	1		4
R5	0	a	
R6	1		6
R7	0	b	
R8	1		8
R9	1		9
R10	0	c	
R11	1		11

yazmaç adlandırma tablosu(YAT)

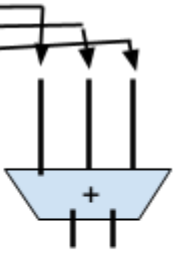
4 çevrim kaldı

	Kaynak 1			Kaynak 2		
	G	eti.	deg.	G	eti.	deg.
x	1	~	1	1	~	2
y						
z						
t						



etiket değer

	Kaynak 1			Kaynak 2		
	G	eti.	deg.	G	eti.	deg.
a	0	x	~	1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d						



etiket değer

c'deki ADD bir sonraki çevrim yürütülecek: Tüm kaynak yazmaçları hazır.

Altıncı Çevrim:

Çarpma birimindeki işlem bitince a'daki buyruk yürütülmeye başlanacak.

a'daki ADD hala bekliyor

b'deki ADD yürütüldü, sonucu etiketi ile beraber YAT'a ve bekleme alanlarına iletilecek.

c'deki ADD yürütülüyor.

Yeni MUL'u çöz ve yerini ayır.

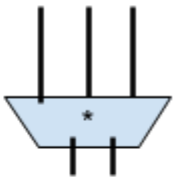
	1	2	3	4	5	6
MUL R1, R2 → R3	G	Ç	Y1	Y2	Y3	Y4
ADD R3, R4 → R5		G	Ç	-	-	-
ADD R2, R6 → R7			G	Ç	Y1	İ
ADD R8, R9 → R10				G	Ç	Y1
MUL R7, R10 → R11					G	Ç
ADD R5, R11 → R5						G

3 çevrim kaldı

Yazmaç	Geçerli	Etiket	Değer
R1	1		1
R2	1		2
R3	0	x	
R4	1		4
R5	0	a	
R6	1		6
R7	1		8
R8	1		8
R9	1		9
R10	0	c	
R11	0	y	

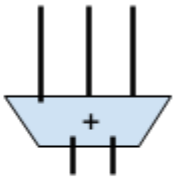
yazmaç adlandırma tablosu(YAT)

	Kaynak 1			Kaynak 2		
	G	eti.	deg.	G	eti.	deg.
x	1	~	1	1	~	2
y	1	~	8	0	c	~
z						
t						



etiket değer

	Kaynak 1			Kaynak 2		
	G	eti.	deg.	G	eti.	deg.
a	0	x	~	1	~	4
b						
c	1	~	8	1	~	9
d						



etiket değer

R7 ve y'nin Kaynak 1 etiketleri iletilen etiket ile aynı
YAT'da R7'nin değeri güncel, b artık boşta

Burdan sonrası adım adım aynı şekilde ilerliyor. Sonuçta her bitiren değerini bir sonraki yani arkadan gelen buyruğa göndererek bekleme alanındakilerin uyanmasını sağlıyor ve onların da işlem birimine hazır hale gelmesini sağlıyor.

Tam Doğru Kural Dışı Durumlar ve Sırasız Yürütme

Problem: Tomasulo'nun algoritması tam zamanlı kural dışı durumları desteklemiyor.
Fikir: Program Sırası Tablosu(PST) kullanmak. PST'yi biz değişken vuruşta işlem yapan boru hattında buyruklar için kullanmıştık, biz bunu Tomasulo'nun algoritmasıyla sırasız yürütüm için de kullanalım. Böyle yaparsak kural dışı durumları da çözebiliriz.

- buyruklar yürütüldüğünde YAT'ı günceller.
- buyruklar tamamlandığında (-ing. commit) mimari yazmaç öbeğini günceller
  - en yaşlı buyruk, mimari yazmaç öbeğini en önce günceller
  - bu sayede mimari yazmaç öbeği program sırasına göre güncellenir

Sırayı baştan dağıtmıyoruz, bellekten getirilmesi program sırası içinde, program sırası içinde yeniden adlandırıyoruz, daha sonra dağıtım gerçekleştiriliyor, bekleme alanına program sırası içinde giriyor ama program sırası dışında çıkıyor, yürütme program sırası dışında, sonuçların yazılması program sırası dışında, tekrar en son mimaride görülen yazmaçları yazmak program sırası içinde olması için PST kullanılıyor.

Kural dışı durum olduğunda:

(1) Boru hattını boşalt: kural dışı olan buyruk en yaşlı hale gelsin, PST'de en başa gelsin ondan sonra ne kadar buyruk varsa atılabilir. Böylece o buyruktan devam edilebilir.
(2) mimari yazmaç öbeğini YAT'a kopyala