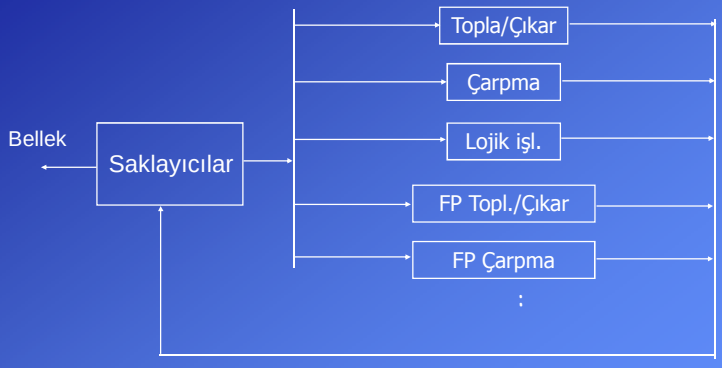
Paralel Veri İşleme

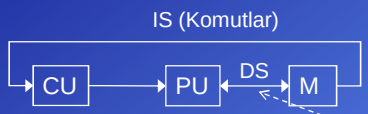
Amaç; aynı anda birden fazla veriyi işleyebilmek.
Farklı şekillerde paralellik sağlanabilir.
Örneğin işlev birimleri bağımsız çalışabilecek şekilde tasarlanabilir.



M.J. Flynn Gruplaması (*Flynn's Taxonomy*)

Michael J. Flynn (1934-)

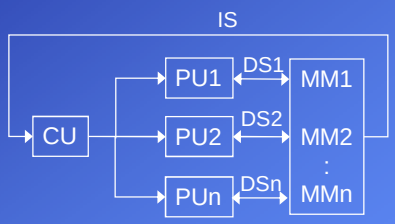
• SISD: Single Instruction Single Data



CU: Control Unit
PU: Processing Unit
M: Memory
IS: Instruction Stream
DS: Data Stream

Veriler

• SIMD:Single Instruction Multiple Data

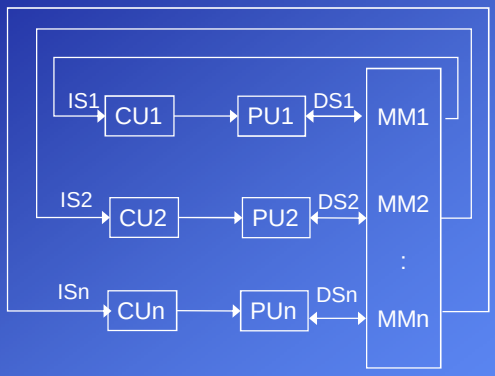


Aynı komut bir çok farklı veri üzerinde paralel olarak yürütülür.

MM: Memory Module

• MISD: Multiple Instruction Single Data (Gerçekçi değil)
Farklı sistemlerin (komutların) aynı veriye aynı sonucu verdiğini test etmek için kullanılır.

• MIMD : Multiple Instruction Multiple Data



CU: Control Unit
PU: Processing Unit
MM: Memory Module
IS: Instruction Stream
DS: Data Stream

İş Hattı (*Pipeline*)

Çok tekrarlanan ve alt parçalara bölünebilen işlerde kullanılır.

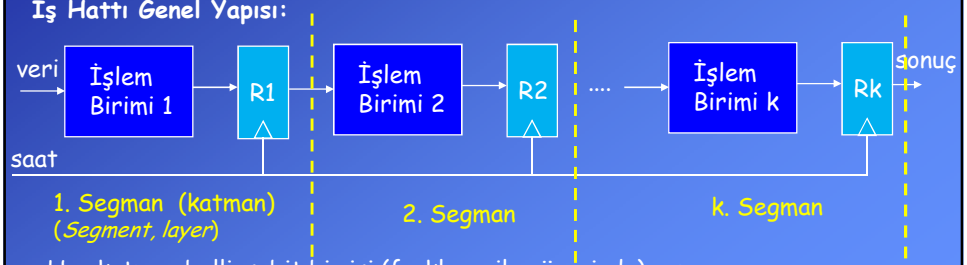
Büyük bir iş (*task*), küçük alt işlere bölünür ve bu alt işler kendi arlarında paralel olarak yürütülürler.

Örnek olarak bir fabrikadaki üretim/montaj bandı düşünülebilir.

Otomobil montaj bandında bir otomobil iş hattında ilerlerken her işçi önüne gelen otomobile belli bir parçayı monte eder.

Örneğin i. işçi bir otomobilin camını takarken aynı anda (i+1). sıradaki işçi bir önceki otomobilin tekerleklerini takmaktadır.

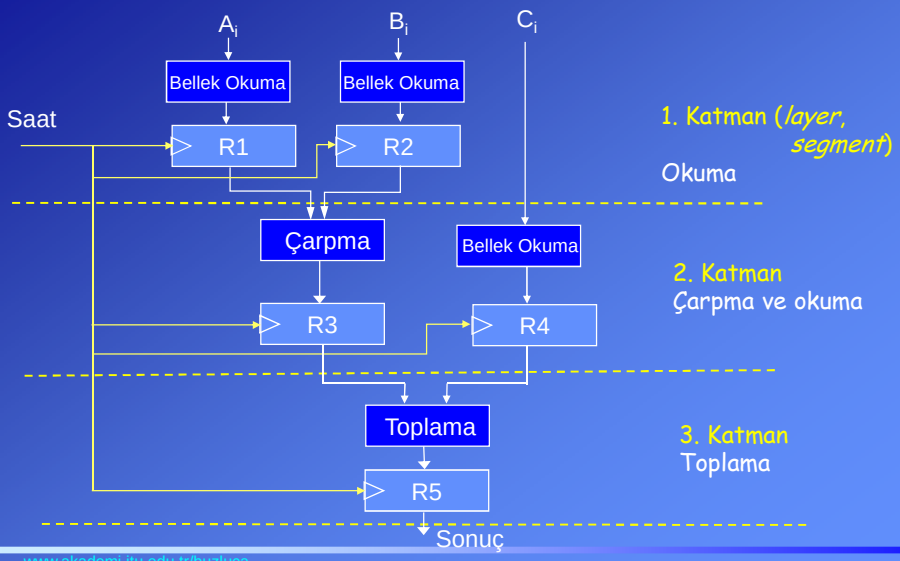
İş Hattı Genel Yapısı:



Her katman belli, sabit bir işi (farklı veriler üzerinde) yapar.

R1, R2 gibi saklayıcılar ara sonuçları tutarlar.

Örnek: A, B ve C dizisinin elemanları önce bellekten okunacak ardından aşağıdaki işlem yapılacaktır. $A_i*B_i + C_i$ i=1,2,3,....



Bu örnekte görev üç alt işleme bölünmüştür: Okuma, çarpma, toplama

Not: Verinin önceden hazır olduğu veya bellek okuma süresinin diğer işlemlere göre çok kısa olduğu sistemlerde bellekten okuma ayrı bir alt işlem olarak ele alınmaz. Bu durumda sadece aritmetik işlemi yapan iş hattı 3 yerine 2 katmanlı olarak tasarlanabilirdi.

Üç segmanlı olarak tasarlanan iş hattının çalışması:

Saat Darbesi	1. Segman		2. Segman		3.Segman
	R1	R2	R3	R4	R5
1	A_1	B_1	-	-	-
2	A_2	B_2	A_1*B_1	C_1	-
3	A_3	B_3	A_2*B_2	C_2	$A_1*B_1 + C_1$
4	A_4	B_4	A_3*B_3	C_3	$A_2*B_2 + C_2$
5	A_5	B_5	A_4*B_4	C_4	$A_3*B_3 + C_3$

4 Segmanlı Bir İş Hattının Uzay-Zaman Diyagramı (*Space-Time Diagram*)

Bir iş hattında belli bir anda hangi işin hangi segmanda işlem gördüğünü göstermek için uzay-zaman diyagramları kullanılır.

Aşağıdaki tabloda, saat darbeleri (adımlar) sütunlara, segmanlar satırlara, işleme giren veriler de tablonun içine yazılmıştır.

Bu tablo farklı şekilde de oluşturulabilir. Veriler (işler) satırlara, o anda etkin olan segman da tablonun içine yazılabilir. (Bkz. 4.11)

Saat Darbesi

Segman	1	2	3	4	5	6	7
1	T1	T2	T3	T4	T5	T6	
2		T1	T2	T3	T4	T5	T6
3			T1	T2	T3	T4	T5
4				T1	T2	T3	T4

1nci iş (T1) 4 saat darbesi (k=segman sayısı) sonunda tamamlandı.

k.'dan sonraki her saat darbesinde yeni bir iş tamamlanır.

İş Hattının sağladığı hızlanma (*Speedup*):

k: segman sayısı
tp: saat periyodu (En yavaş birime göre ayarlanır)
n: işin tekrar sayısı
Buna göre 1nci işin (T1) tamamlanma süresi: k·tp
Kalan n-1 işin tamamlanma süresi: (n-1)tp
Tüm işlerin toplam süresi: (k+n-1)tp
tn :İş hattı kullanılmasaydı bir işin süresi

Hızlanma (*Speedup*): $S = \frac{n \cdot t_n}{(k+n-1) \cdot t_p}$

İş sayısı çok artarsa: $\lim_{n \to \infty} S = \frac{t_n}{t_p}$

Eğer tn= k·tp varsayımı yapılırsa:

S = k (Teorik maksimum hızlanma)

Yorumlar:

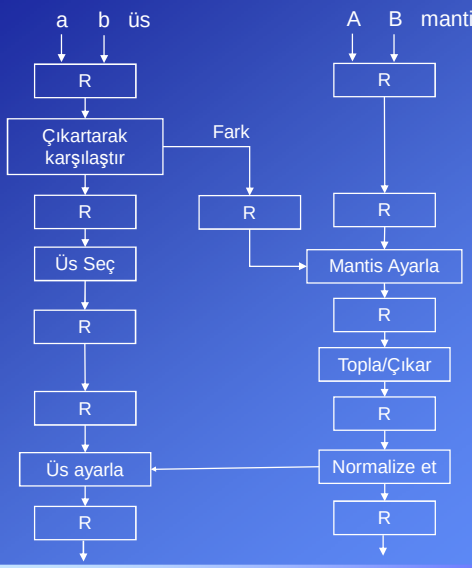
- İş hattının verimini arttırmak için bir işi mümkün olduğu kadar eşit sürelerdeki alt işlere bölmek gerekir.
(En yavaş birim süreyi belirler)
- Segman sayısı arttıkça ilk baştaki bekleme süresi artar.
Ancak eğer iş daha kısa süreli alt işlere bölünebiliyorsa saat işareti hızlanacağından segman sayısını arttırmak verimi arttırabilir.

Aritmetik İş Hattı

Örnek: Kayan Noktalı Sayıların Toplanması ve Çıkartılması

$X = A*2^a$

$Y = B*2^b$



Komut İş Hattı (*Instruction Pipeline*)

Merkezi işlem birimleri komutları işlerken belli alt işlemleri tekrar ederler.

En basit iş hattı yapısı iki segmanlı olarak kurulabilir:
1) Komut alma 2) Komut yürütme

Komut yürütme birimi belleğe erişmediği zamanlarda komut alma birimi sıradaki komutu bellekten alarak bir kuyruğa (komut kuyruğu) yazar.

Verimi arttırmak için komut işleme daha küçük alt işlemlere bölünerek 6 segmanlı bir iş hattı oluşturulabilir:

1. Komut al,
2. Komut çöz,
3. Efektif adres hesapla,
4. Operand al,
5. Komut yürüt,
6. Sonucu yaz.

Bu kadar ayrıntılı bölmeleme çeşitli problemler nedeniyle verimli olmaz:

- Her komut bütün alt işlemlere gerek duymaz,
- Değişik segmanlar aynı anda bellek erişimine gerek duyar,
- Segmanların süreleri çok farklı olabilir.

Bu nedenle komut iş hatları daha az segmanla (örneğin 4) oluşturulur.

Örnek: Dört Segmanlı Komut İş Hattı

1. FI (*Fetch Instruction*): Komut alma
2. DA (*Decode, Address*): Komutu çöz , efektif adresi hesapla
3. FO (*Fetch Operand*): Operand al
4. EX (*Execution*): Yürütme (İşlem yapılır, saklayıcılar güncellenir)

Komut alma ve operand alma işlemlerinin aynı anda yapılabilmesi için komut ve veri belleklerinin ayrı oldukları varsayılmıştır.

Koşulsuz dallanma olduğunda:

Komutlar \ Adımlar	1	2	3	4	5	6	7	8
1	FI	DA	FO	EX				
2		FI	DA	FO	EX			
Koşulsuz Dallan. 3			FI	DA	FO	EX		
4				FI	-	-	FI	DA
5								FI

Komut çözüldüğünde Dallanma olduğu anlaşılır

Dallanılacak adres alınıyor (Mutlak ya da bağıl)

PC güncelleniyor. PC=dallanılacak adres

Bu komut boşuna alındı İş hattından silinecek

Dallanma cezası (*Branch penalty*)

Dallanmadan sonra gidilen komut

Koşullu dallanma (koşul doğru değilse):

Komutlar \ Adımlar	1	2	3	4	5	6
1	FI	DA	FO	EX		
Koşullu Dallan. 2		FI	DA	FO	EX	
3			FI	DA	FO	EX

Önceki komut bayrakları (koşulları) belirliyor.

Koşula bakılmaksızın bir sonraki komuttan devam ediliyor.

Dallanma gerçekleşmedi. PC değişmedi. (Ceza yok)

Koşullu dallanma (koşul doğru ise):

Komutlar \ Adımlar	1	2	3	4	5	6	7
1	FI	DA	FO	EX			
Koşullu Dallan. 2		FI	DA	FO	EX		
3			FI	DA	FO	FI	DA
4				FI	DA		FI
5					FI		

Dallanılacak adres belirleniyor PC'ye yazılıyor. İş hattı boşaltılmalı.

Dallanma ile gidilen komut

İş hattı boşaltılıyor (*Branch penalty*)

Bu tablolarda, işler (komutlar) satırlara, segmanlar ise tabloların içine yazılmıştır.

İş Hattında Oluşabilen Problemler (*Pipeline Conflict*)

1. Kaynak Çatışması (*Resource Conflict*):
   a) İki farklı segmanda aynı bellek modülüne erişilmek istenirse
   Çözüm:
   • Harvard mimarisi: Komutlar ve veriler için ayrı bellek
   • Komut kuyruğu: Belleğe erişilmediği anlarda komutlar alınır.
   b) İki farklı segmanda aynı işlem birimine (ALU) gerek duyulursa.
   Çözüm: İşlem birimlerinin sayısı arttırılır. Örneğin adres hesabı ve veri işleme için iki ayrı ALU kullanılır.

2. Veri Bağımlılığı (*Data Dependency*): Bir komutun işlemi önceki komutun üreteceği sonuca bağımlı olabilir.

a) Operand bağımlılığı

```
ADD.W   D1, DATA
MOVE.W DATA, (A0)
```

b) Adres Bağımlılığı:

```
ADDA.W   #2, A0
MOVE.B   (A0)+, D0
```

3. Dallanma Problemi:
Koşullu, koşulsuz dallanma, kesmeler

Veri bağımlılığı ile ilgili çözümler:

• *Hardware interlock*: Bir donanım tüm komutları izler. Veri bağımlılığı olan komutun iş hattına girmesi geciktirilir.

• *Operand forwarding*: ALU'nun çıkışından girişine doğrudan bir bağlantı oluşturulur. ALU'da oluşturulan sonuç hedef saklayıcıya yazılırken aynı zamanda bir sonraki işlemde kullanabilmesi için ALU'nun girişine verilir.

• Gecikmeli Yükleme (*Delayed Load*): Problemi derleyici çözer
Mümkünse komutların yerini değiştirir.
Değilse bağımlı komutlar arasına NOP (*No Operation*) komutu koyar.

Dallanma problemine ilişkin çözümler:

• Önceden komut alma (*Target Instruction prefetch*): Koşullu dallanmalarda hem dallanmadan sonraki komutlar (koşul yanlış ise kullanılacak) hem de dallanma ile gidilmesi olası olan yerdeki komutlar (koşul doğru ise kullanılacak) iş hattına alınır.

• Dallanma hedef tablosu tutulur (*Branch target buffer*) Dallanma komutlarının adresleri ve son çalıştıklarında nereye gidildiği (gidilen yerdeki bir kaç komut) bir çağrışımlı bellekte (*associative memory*) tutulur. Böylece gidilecek adres hesaplanmadan önce dallanma komutundan sonraki komutlara erişilebilir.

• Dallanma öngörüsü (*Branch prediction*): Dallanmalar hakkında istatistik tutulur. Koşulun doğru ya da yanlış olma olasılığına göre uygun komutlar iş hattına alınır.

• Gecikmeli dallanma (*Delayed branch*) Mümkünse derleyici tarafından komutların yeri değiştirilir.

RISC İş Hattı (*RISC Pipeline*)

RISC işlemcilerde,
komutlar az sayıda, basit ve sabit uzunlukta olduğundan,
az sayıda ve basit adresleme kipi bulunduğundan,
veri işleme komutları sadece saklayıcılar üzerinde yapıldığından
üç segmanlı bir komut iş hattı yeterli olmaktadır.
Farklı sayıda segmana (örneğin dört) sahip RISC işlemciler de bulunmaktadır.

Örnek: Üç segmanlı bir RISC komut iş hattı

- I (*Instruction Fetch*): Komut alma
- A (*Decode, ALU Operation*): Komut çözme ve ALU işlemi
  ALU üç farklı iş için kullanılır.
  1. Saklayıcılar üzerindeki aritmetik/lojik komutlarda
  2. Bellek erişimi komutlarında adres hesabı için. LDL **(R5)#10**,R15
  3. Bağıl adreslemede PC ← PC+Y işlemi için.

  A (ALU) segmanında hem işlem yapılır hem de sonuç varış saklayıcısına (R veya PC) aynı saat darbesinde yazılmış olur.
- D (*Data*): Bu segman sadece bellek erişimi için kullanılır. Bellekten gelen veri saklayıcıya ya da saklayıcıdaki sonuç belleğe yazılıyor.

I ve D segmanlarında aynı anda bellek erişimi yapılabilmesi için komut ve veriler için paralel erişilebilen ayrı belleklerin (ve ayrı yolların) olması gerekir.

Örnek: RISC iş hattında veri bağımlılığı

Burada veri bağımlılığı 4.15'te verilen 3 segmanlı örnek iş hattı yapısına göre ele alınmıştır.

```
LOAD  :  R1 ← M[adres 1]
LOAD  :  R2 ← M[adres 2]
ADD   :  R3 ← R1 + R2
STORE :  M[adres 3] ← R3
```

İş hattında veri bağımlılığı problemi

Saat darbeleri / Komutlar	1	2	3	4	5	6
LOAD R1	I	A	D			
LOAD R2		I	A	D		
ADD R1,R2,R3			I	A	D	
STORE R3				I	A	D

Veri bağımlılığı:
Aynı saat darbesinde hem R2'ye değer yüklemek hem de toplamayı yapmak mümkün değil.

R3 ile ilgili veri bağımlılığı yoktur.

ADD komutu bu segmanı kullanmaz.

Veri bağımlılığını çözmek için derleyici, ya program satırlarının yerini değiştirir ya da gerekli yerlere NOOP (*No Operation*) komutları ekler.

Gecikmeli Yükleme (*Delayed Load*) ile Çözüm

Derleyici, program satırlarının yerini değiştirerek R2'ye değer yükleyen komut ile bu değerin kullanıldığı komut arasına başka komutlar yerleştirmeye çalışır.

Uygun bir komut yoksa araya NOOP (*No Operation*) komutu yerleştirilir.

Saat darbeleri / Komutlar	1	2	3	4	5	6	7
LOAD R1	I	A	D				
LOAD R2		I	A	D			
NOOP			I	A	D		
ADD R1,R2,R3				I	A	D	
STORE R3					I	A	D

R2'ye yazılıyor.

Farklı saat darbelerinde yapılıyorlar.

R2 işleme giriyor.

Geciktirilmiş Dallanma (*Delayed Branch*)

Dallanma komutu yürütülünceye kadar programın hangi adresten devam edeceği (hangi komutun alınacağı) belli değildir.

Gidilecek olan adres hesaplanıncaya kadar geçecek olan sürede iş hattı NOOP komutları ile doldurulur.

Örnek:

Load R1
Increment R2
Add R3 to R4
Subtract R5 from R6
Branch to address X
.....
X Instr.X

NOOP ile Çözüm

Saat darbeleri / Komutlar	1	2	3	4	5	6	7	8	9
LOAD R1	I	A	D						
INC R2		I	A	D					
ADD R3,R4			I	A	D				
SUB R5, R6				I	A	D			
BRANCH X					I	A	D		
NOOP						I	A	D	
X INSTR. X							I	A	D

PC güncelleniyor.
PC ← X (gidilecek adres)

Farklı saat darbelerinde yapılıyor.

Komutların yerini değiştirerek çözüm:

Derleyici eğer mümkünse komutların yerini değiştirir.

Saat darbeleri / Komutlar	1	2	3	4	5	6	7	8
LOAD R1	I	A	D					
INC R2		I	A	D				
ADD R3,R4			I	A	D			
BRANCH X				I	A	D		
SUB R5, R6					I	A	D	
X INSTR. X						I	A	D

PC güncelleniyor.
PC= X (gidilecek adres)

PC= X (gidilecek adres) olduğu için INSTR X alınır.

Burada komutların yer değiştirmesi programın işlevini etkilememektedir.
Dallanma komutu, PC değerini değiştirmeden önce, sonraki komut (örnekte SUB) iş hattına alınmıştı.

Not: Ders kapsamında tanıtılan 4 segmanlı komut iş hattı (4.11) ve bir RISC işlemeciye ait 3 segmanlı iş hattı (4.15) birer örnektir.
Değişik işlemcilerin iç yapıları ve buna bağlı olarak sahip oldukları iş hatları farklıdır.