Bilgisayar Mimarisi Final Notları

~ $Yunus\ Emre\ AK$ ©

Döküman Renklendirme Yapısı

PDF Başlığı

Ana Başlıklar

Alt Başlıklar

İç Başlıklar

En İç Başlıklar

Tablo	Başlığı

Bağlantılar

`Değişmez ifadeler`

$Formüller$

Önemli notlar

Terimsel ifadeler

> Yorum satırları

İçerikler

Pipeline

Pipeline durumunda her *cycle*'da birden fazla işlem yapılır.

Pipeline Hazards

- Structural hazards
  - Aynı işlem birden fazla yapılamaz
  - Örn: Alt alta 2 tane IF olamaz
- Data hazards
  - Register verileri hazırlanmadan kullanılamaz
  - Örn: R1'e 5 atanıyorsa, Memory işlemi yapılmadan R1'e erişilemez
- Control hazards
  - Koşula bağlı bir durum varsa koşul gerçekleştirilmeden işleme sokulamaz

Pipeline Örneği

Instruction	Clock cycle number															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
LW R1, 0 (R4)	IF	ID	Ex	M	W											
LW R2,400(R4)		IF	ID	Ex	M	W										
ADDI R3,R1,R2			IF	ID	*	*	Ex	M	W							
SW R3, 0 (R4)				IF	*	*	ID	Ex	*	M	W					
SUB R4, R4, #4							IF	ID	*	Ex	M	W				
BNEZ R4, L1								IF	*	ID	*	*	Ex	M	W	

Hız Hesaplama

$$CycleTime = (IF + n_m \times n_c) \times CPI$$

- $CycleTime$, Saat çevrim sayısı
- IF, ilk komuttaki IF katı
  - Genelde 1 olur
- n_m, Döngüdeki komut sayısı
- n_c, Döngüdeki çevrim sayısı
- CPI, Her talimat için çevrim sayısı (Cycle per instruction)
  - Genelde 1 olur

Bellek Yönetim Sistemi

Hız ve Maaliyet Kıyaslaması

$Register > Cache > Memory > Disk > Tape$

> Kapasite için tam tersi

Cache-Memory Terimleri

Terim	Açıklama
Cache	Önbellek
Memory	Bellek
Hit	Cache üzerinde verinin buluması
Miss	Hit olmaz ise verinin memory'den alınması

Memory with Cache (Önbellekli Bellek) Erişme

`lw $t0 0($t1)` instruction (talimat) için:

- `$t1` 1022_{ten} verisini içerir
- $Memory[1022] = 99$ olsun
- Cache varsa:
  - İşlemci 1022_{ten} adersini cache belirtir
  - Cache 1022_{ten} verisinin kopyası var mı kontrol eder
  - Eğer kopyası varsa (hit):
    - Cache 99'u okur ve işlemciye gönderir
  - Eğer kopyası yoksa (miss):
    - Cache 1022_{ten}'u memory'e gönderir
    - Memory 99 verisini 1022_{ten} adresinden okur
    - Memory 99 verisini cache'e gönderir
    - Cache verisini yeni 99 verisiyle değiştirir
    - Cache 99 verisini işlemciye gönderir

Cache-Memory Hesaplamaları

Cache Uzunluğu

$$S_{catch} = 2^{S_{index}} \times S_{tag} \times S_{line}$$

- S Uzunluk (size)

Hit ve Miss Oranları

$$R_{miss} = 1 - R_{hit}$$

- R: Rate, oran değeri (<1)

Hit ve Miss Süreleri

$$T_{cache} = T_{replace}$$

$$T_{miss} = T_{replace} + T_{deliver}$$

- T_{miss}: Miss durumunda harcanan süre (penalty)
- $T_{replace}$: Cache'deki bloğu yeniden yerleştirme süresi
- $T_{deliver}$: Bloğun işlemciye aktarılma süresi

Etkin Bellek Varış Zamanı

Etkin bellek varış zamanı aşağıdaki gibi tanımlanır:

$$T_e = T_{cache} \times R_{hit} + (1 - R_{hit}) \times T_{memory}$$

- T: Time, tamamlanması için geçen süre (penalty)
- R: Rate, oran

Ortalama Erişim Süresi

$$T_{avg} = T_{hit} \times R_{hit} + T_{miss} \times R_{mis}$$

- T: Time, tamamlanması için geçen süre (penalty)
- R: Rate, oran

Mapping Function

- Associative mapping (birleşik adresleme)
  - Cache'deki her block memory bloğunu taşıyabilir
  - En hızlı ve en pahalı yöntemdir
- Direct mapping
  - Memory blokları için cache'ye 1 yer vardır
- Set-associative mapping
  - Memory blokları için cache'de N farklı yer vardır, daha fazla R_{hit} mümkün
- Fully Associative Mapping
  - Memory blokları cache'de **herhangi** bir yere gidebilir

> Set ya da Fully ASsociative Cache'lerde cache bloklarının nereye atılacağı sorun oluşturur

Associative Mapping (Birleşik Adresleme)

- En hızlı ve en pahalı yöntemdir
- Cache'deki her block memory bloğunu taşıyabilir
- Hem adres hem de içerik bilgisi *mapping table*'da saklanır ve **CAM** tarafından yapılır

Direct Mapping

- Her memory bloğu için cache'de ayrılan **sadece bir yer** vardır
- Adressleme tablosu (*mapping table*) **RAM** tarafından yapılır
- Adres bitleri 3 parçaya ayrılır
  - Tag
  - Index, kelimenin indeks verisi
  - Offset, index'den kaç adım sonrası olacağını söyler
- Index alanından cache data tablosuna bakalır
  - Valid = 0 ise:
    - Veri cache'e aktarılır
    - Tag verisi aktarılır
    - Valid = 1 yapılır
  - Valid = 1 ise:
    - Tag kontrol edilir, uyuşuyorsa (hit):
      - Offset değeri kadar kaydırılara gerekli kelime alınır
    - Uyuşmuyorsa (miss):
      - Blok, yeni veri ve *tag* ile güncellenir
      - Offset değeri kadar kaydırılara gerekli kelime alınır

Set Associative Mapping

- Birden fazla alan olduğu için birden fazla yere bakılır
- Döngüsel olarak *Direct Mapping* işlemi yapılıyor denebilir (?)
  - Veriler Hit / Miss döngüsünden sonra gelecektir
  - *Direct Mapping*'de hit edip devam edebilme şansı var
- Mux olmasından dolayı işlemler daha yavaş tamamlanacaktır

Yerleştirme Algorimalatı

Yeni blok alındığında hangi bloğun yerine koyulacak.

Algoritma	Seçme Koşulu	Hız
Optimal	Uzun süredir kullanılmayan	İleride öğrenilecek 🤔
Least Recently Used (LRU)	Az çağrılan	Çok zor 😢
First-in First-out (FIFO)	İlk giren	Kolay
Random (RAND)	Rastgele	Aşırı kolay 😅

Cache Üzerinde Yazma İşlemleri (Cache Write)

Metod	Cache Yapısı
Write-Throug	V Tag Data
Write-Back	V D Tag Data

Write Through (Direkt Yazma)

- `V Tag Data` yapısına sahiptir
- Ld (load) komutu için:
  - Veri cache'de varsa (Hit):
    - Register'a yazılır
  - Yoksa (Miss):
    - Memory'deki bloklar cache alınır (*mem-read*)
    - Cache'deki veriler hesaplanır
      - $V = 1$ yapılır
      - $X_{tag} = X_{Adress} \div S_{data}$
      - $X_{data} = X_{memory}$
- St (store) komutu için:
  - Veri cache'de yoksa (Miss):
    - Memory'deki bloklar cache alınır (*mem-read*)
    - Cache'deki veriler hesaplanır
      - $V = 1$ yapılır
      - $X_{tag} = X_{Adress} \div S_{data}$
      - $X_{data} = X_{memory}$
  - Veri önce cache'e sonra memory'e yazılır (*mem-write*)

> X: Değer, S_{data}: Blok boyutu (kaç byte), X_{memory}: Memory'deki değer

Write-Back (Sonradan Yazma)

- `V D Tag Data` yapısına sahiptir
- Ld komutu için ek olarak $d = 0$ işlemi yapılır
- St komutunu işleyiş şekli:
  - Veri cache'de yoksa (Miss):
    - Cache üzerinde $d = 0$ olan blok yoksa:
      - $d = 1$ olan bloklardan biri memory'e yazılır (*mem-write*)
    - Memory'deki bloklar cache alınır (*mem-read*)
    - Cache'deki veriler hesaplanır
      - $V = 1$ ve $d = 0$ yapılır
      - $X_{tag} = X_{Adress} \div S_{data}$
      - $X_{data} = X_{memory}$
  - Veri cache'e yazılır

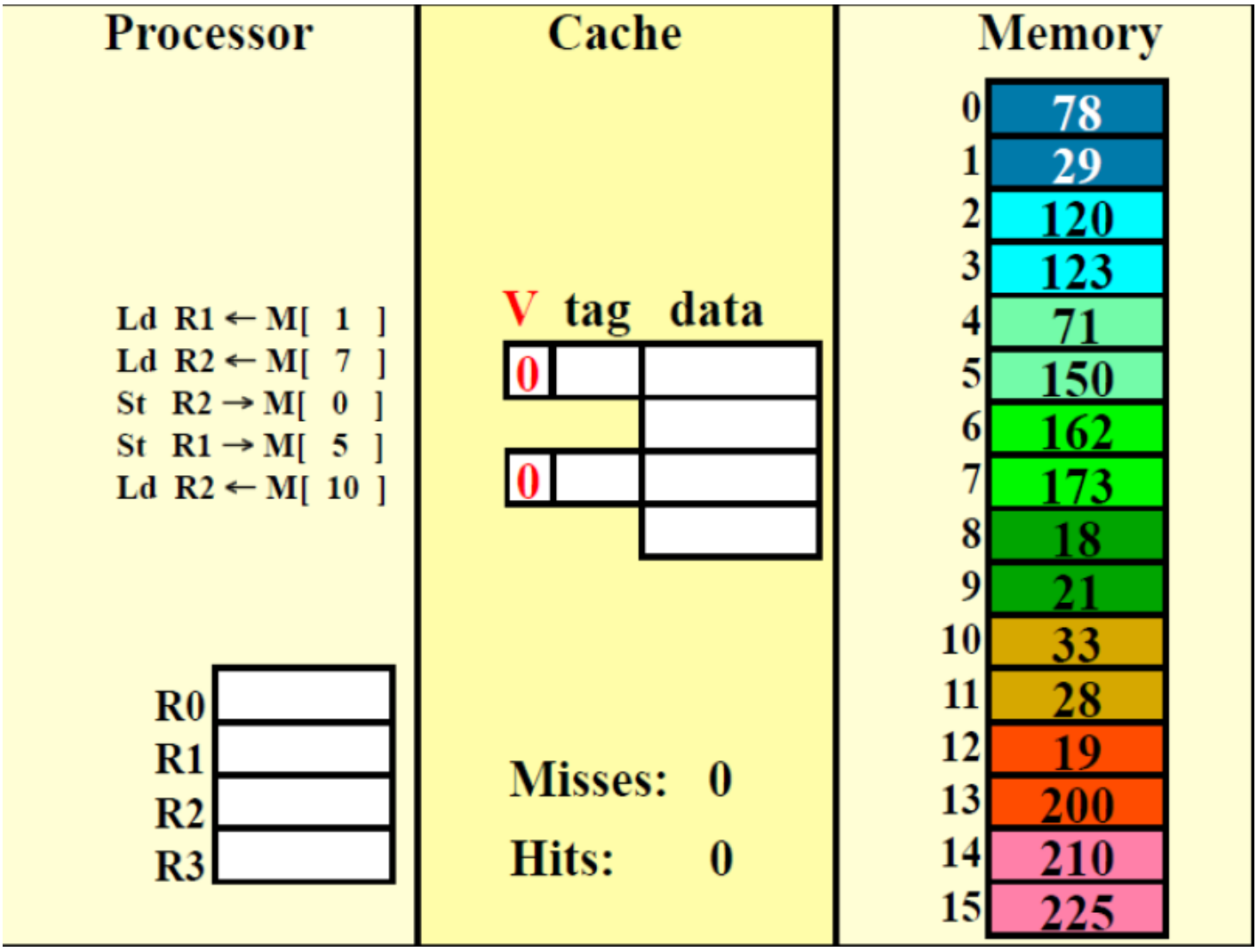
> d: Dirty Cache, cache'deki veri memory'den farklıysa yani cache'e yazma işlemi yapıldıysa 1 değerini alır

Cache Üzerinde Yazma İşleml Hesaplamaları

- Her *miss* işlemi için 1 **blok** okunur (*mem-read*)
  - S_{data} kadar byte demek
- Her *store* işlemi için 1 **byte** yazılır (*mem-write*)
- Her *dirty cache* işlemi için 1 **blok** yazılır (*mem-write*)

Alttaki sistem içim:

Write-Through	Write-Back
4 Miss 1 Hit 2 store	4 Miss 1 Hit 2 dirty cache
Toplam read: 8 bytes	Toplam read: 8 bytes
Toplam write: 2 bytes	Toplam write: 4 bytes

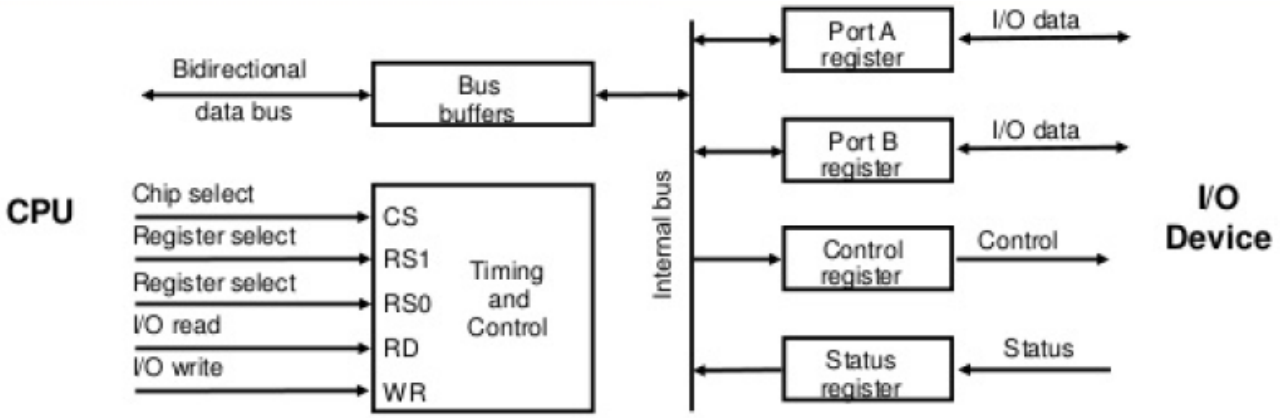


I / O Arabirimi

I/O Temel Görevleri

- Eleman adreslerinin kod çözümü
- Komutların kod çözümü
- Çevresel kontrolör için gerekli işaratleri seçer
- Data akşını senkronize eder ve yönlendirir
- Çevresel elemanlar ve CPU veya Bellek arasında transfer hızını ayarlar

I/O Blok Diyagramı



Asenkron Data Transfer Yöntemleri

Strobe Darbesi

- Transfer meydana gelmek zorunda olduğunda diğer bir birimi göstermek için üretilen işaret
- Her transferi zamanlamada tek bir kontrol işareti kullanılır
- Kaynak veya hedef birim tarafında gönderilebilir

El Sıkışma (Hand-Shaking)

- Verinin varlığını göstermek için iletilen her veriye eklenen kontrol işaraetidir
- Alıcı birim veriyi kabul ettiğinde bir başka işaret daha gönderir

I/0 Etkileşimlier

Etkileşim	Açıklama
Kaynak Etkili	Verinin alınıp alınmadığı ile ilgilenmez
Hedef Etkili	Verinin bus'a aktarıldığı hakkında bilgi yoktur, el sıkışma yöntemi ile çözülür