Tahmini Ders İçeriği
(Tentative Couse Schedule – Syllabus)

1. Hafta: Sayı sistemleri, onluk/ikilik taban sayı gösterimleri, mantıksal kapılar, computer system overview, başarım (performance)

2. Hafta: 2'lik tabanda işaretli sayılar, mikroişlemci tarihi, benchmarking, başarım,

3. Hafta: Başarım, Amdahl yasası, RISC-V development Environment, Verilog HDL ile Birleşik (Combinational) devreler

4. Hafta:, Verilog HDL ile sıralı (sequential) mantıksal devre ve sonlu durum makinası tasarımı, timing analysis

5. Hafta: Aritmetik devre tasarımları: Toplama, çıkarma, çarpma, bölme, trigonometri, square-root, hyperbolic, exponential, logarithm

6. Hafta: Fixed ve Floating-Point sayı gösterimleri

7. Hafta: RISC-V buyruk kümesi mimarisi (ISA) ve buyrukların tanıtımı

8. Hafta: RISC-V buyruk kümesi mimarisi (ISA) ve buyrukların tanıtımı

9. Hafta: Vize Çözümleri – CPU Tasarımına Giriş

10. Hafta: Tek-çevrim işlemci tasarımı (single-cycle CPU)

11. Hafta: Çok-çevrim işlemci tasarımı (multi-cycle CPU)

12. Hafta: Boruhatlı işlemci tasarımı (pipelined CPU)

13. Hafta: Bellek sistemi ve hiyerarşisi

14. Hafta: Gömülü sistemler, mikrodenetleyiciler, SoCs

BELLEK HİYERARŞİSİ

Multi-cycle ve pipelined işlemci tasarımında instruction ve data memory erişimlerini 1 saat vuruşu olarak kabul ettik

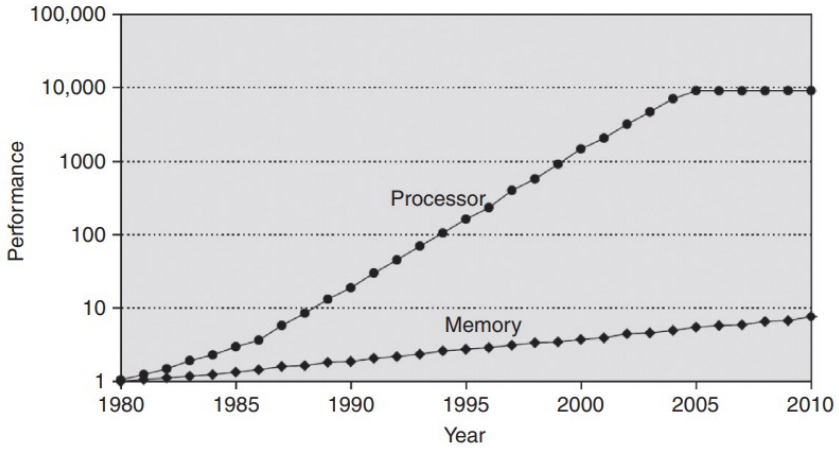
Peki gerçekte bu mümkün müdür?

Eğer veriyi ana bellekten (main memory) getireceksek mümkün değildir. Ana bellek günümüzde DRAM (Dynamic RAM) teknolojisi ile üretilir ve yüksek kapasiteye sahiptir (8 – 128 GB).

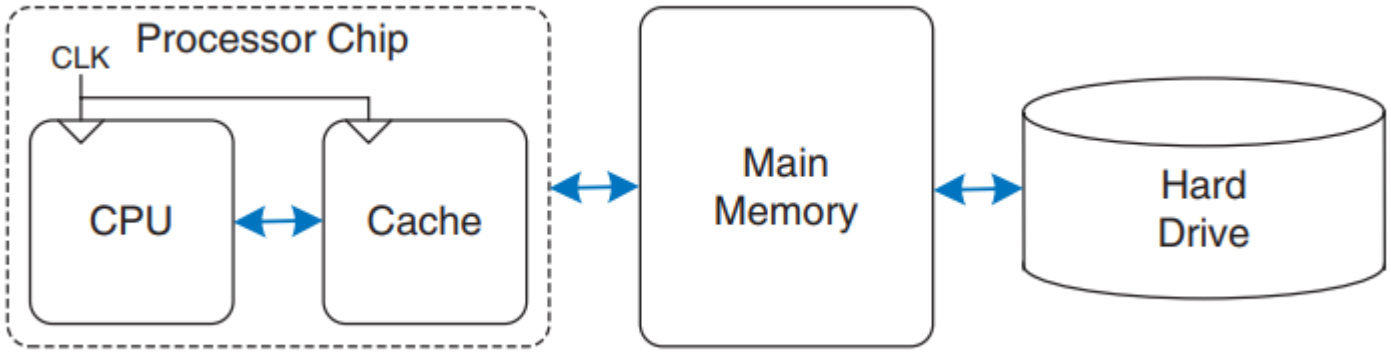
Fakat bu kadar büyük kapasiteli bir bellekten ihtiyacımız olan veriyi getirmek, işlemci saat hızına referansla 10-100 kat daha yavaş kalabilmektedir.

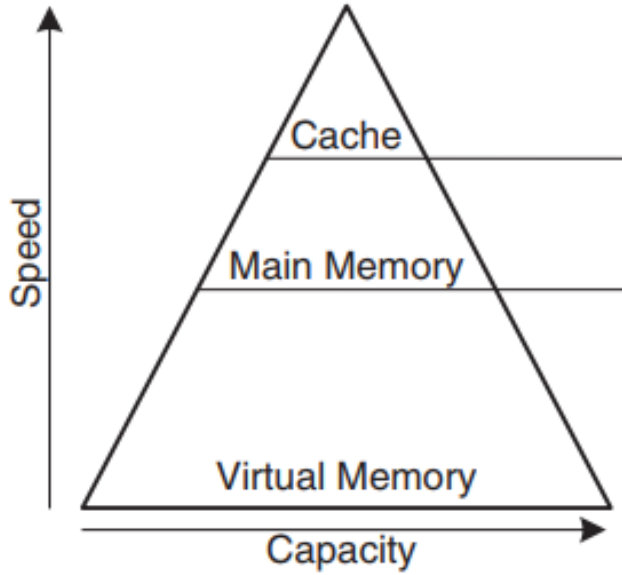
Hatta bazı programlar için ana bellek yetersiz kalabilir ve ana bellekten de çok daha yavaş olan hard disk üzerinden veri getirmek zorunda kalınabilir.

Çözüm → Bellek hiyerarşisi



BELLEK HİYERARŞİSİ





Technology	Price / GB	Access Time (ns)	Bandwidth (GB/s)
SRAM	$100	0.2–3	100+
DRAM	$3	10–50	30
SSD	$0.10	20,000	0.05–3
HDD	$0.03	5,000,000	0.001–0.1

BELLEK TEKNOLOJİSİ

SRAM (Static Random Access Memory): En pahalı ve en hızlı bellek teknolojisidir. DRAM'e göre bit başına daha fazla alan kaplar. İşlemci içerisindeki register file ve çip içerisindeki önbellek (cache) SRAM'den yapılır.

DRAM (Dynamic Random Access Memory): SRAM'e göre daha ucuz ama daha yavaş bir teknolojidir. SRAM'e göre bit başına daha az alan kaplar. Ana bellek DRAM teknolojisi kullanılarak yapılır. Dinamik denilmesinin nedeni belirli bir periyotta sürekli güç verilerek refresh edilmesi gerektiği içindir. SRAM'de böyle bir refresh mekanizmasına gerek duyulmaz.

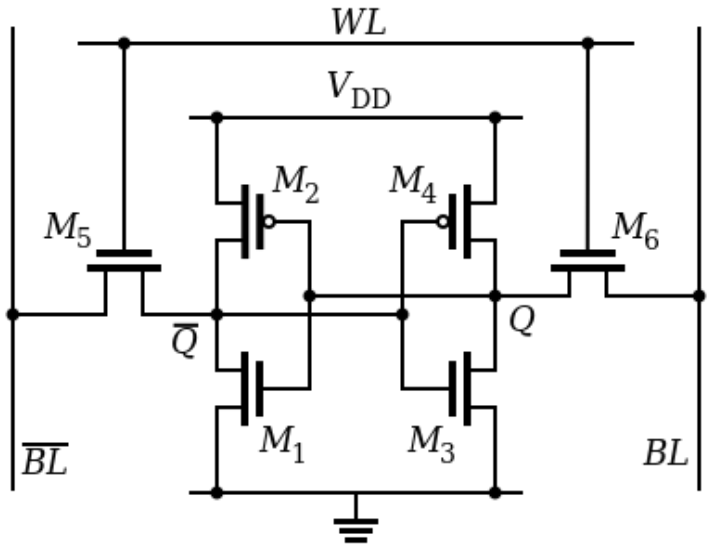
Flash Memory: Aslında USB bellek dediğimiz veya SSD bellek dediğimiz tip flash teknolojisi ile üretilmektedir. Eski manyetik hard disklerin yerini almıştır.

Magnetic Disk: Flash belleğin ucuzlamasıyla birlikte günümüzde fazla kullanım alanı kalmamıştır. RAM değildir, yani bütün bitlerine eşit zamanda ulaşılamaz, metal bir çubuğun manyetik alan okuması ile veriye erişilir.

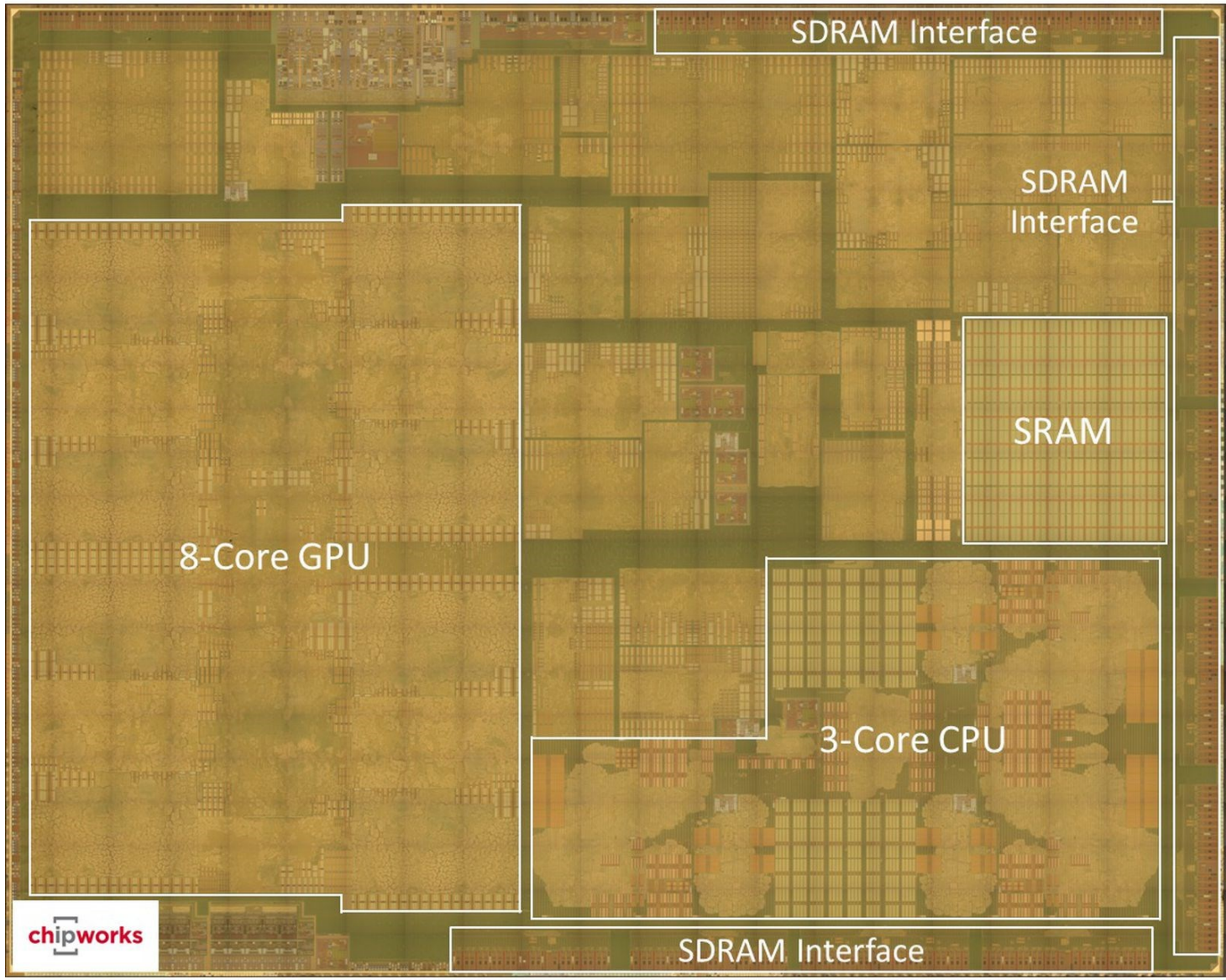
BELLEK TEKNOLOJİSİ

SRAM (Static Random Access Memory)

WL: Word Line, BL: Bit line

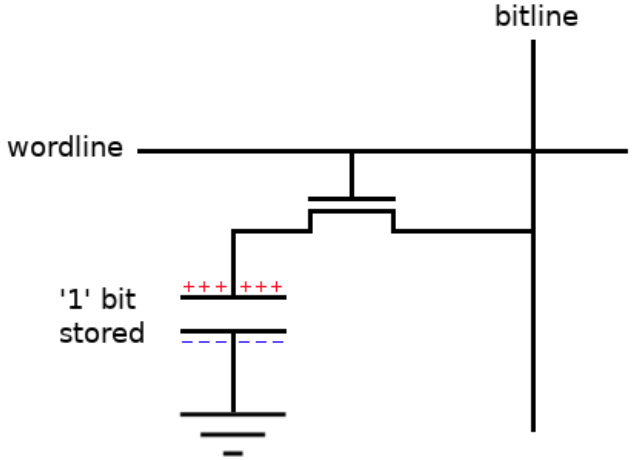


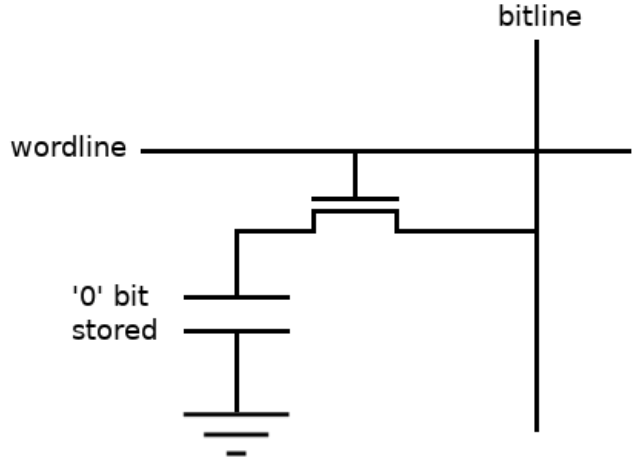
IPAD air2 A8X Chip Layout

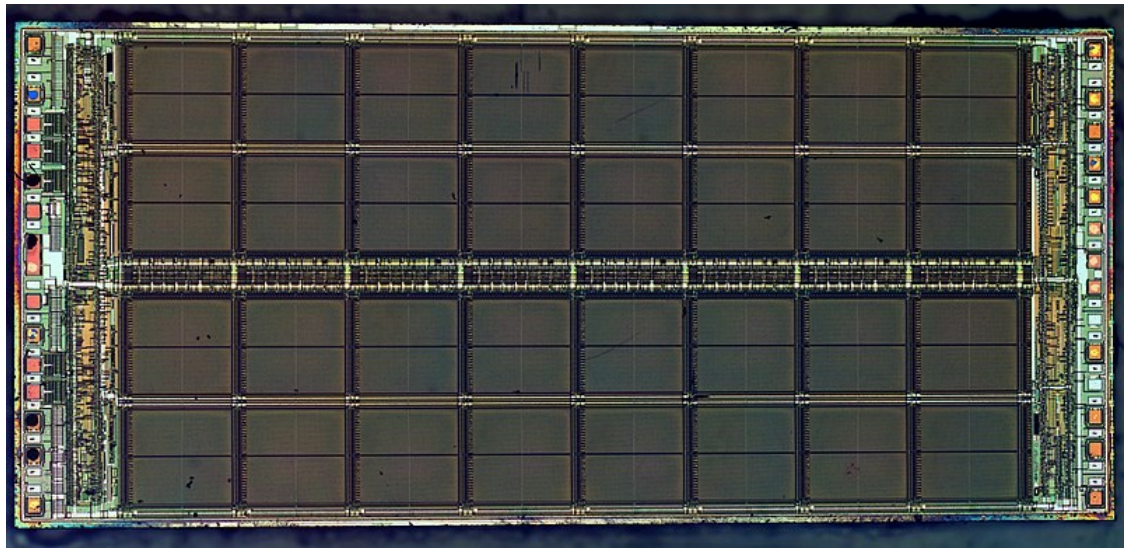


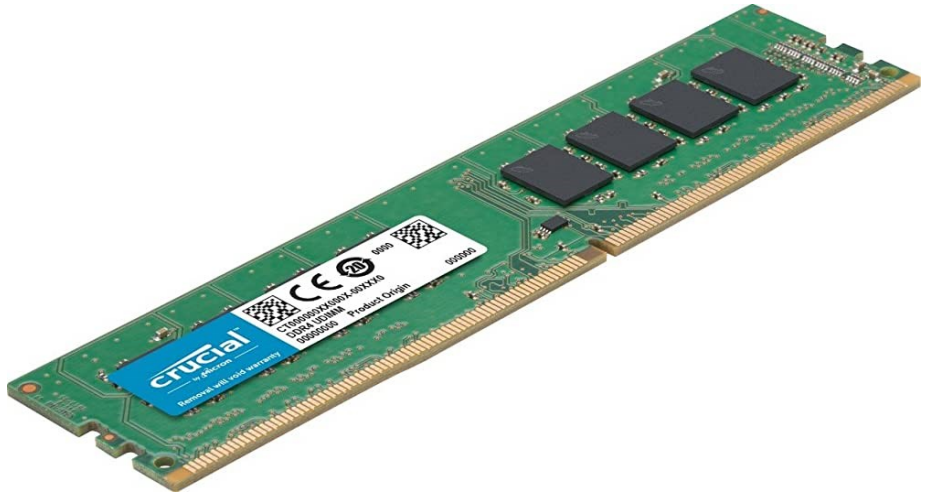
BELLEK TEKNOLOJİSİ

DRAM (Dynamic Random Access Memory)









BELLEK TEKNOLOJİSİ

Flash Memory

SSD form factors

2.5" 15mm U.2 SAS

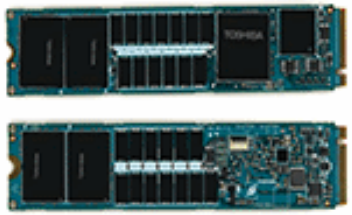
2.5" 15mm U.2 PCIe

Half-height,
half-length (HHHL)
AIC PCIe

2.5" 7mm SATA

Ball grid array (BGA)
surface-mount (M.2 1620)
and removable
module (M.2 2230)

M.2 2280



M.2 22110

BELLEK TEKNOLOJİSİ

Magnetic Disk

LOCALITY (YERELLİK)

Programs exhibit both temporal locality, the tendency to reuse recently accessed data items, and spatial locality, the tendency to reference data items that are close to other recently accessed items.

Memory hierarchies take advantage of temporal locality by keeping more recently accessed data items closer to the processor.

Memory hierarchies take advantage of spatial locality by moving blocks consisting of multiple contiguous words in memory to upper levels of the hierarchy.

Örnek: Matrix Multiplication

```
for i in 0..n
  for j in 0..m
    for k in 0..p
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

YAVAŞ: A[i][k] önbellekte ama B[k][j] değil. C[i][j] önemli değil son döngüde tek değişken durumunda.

```
for i in 0..n
  for k in 0..p
    for j in 0..m
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

HIZLI: B[k][j] ve C[i][j] önbellekte, A[i][k] önemli değil son döngüde tek değişken durumunda.

ÖNBELLEK (CACHE)

Önbellekler, çip içerisinde işlemciye en yakın bellek tipidir.

İşlemci, load veya store instruction yürüteceği zaman ilk olarak önbellekte bu adresi, dolayısıyla veriyi arar.

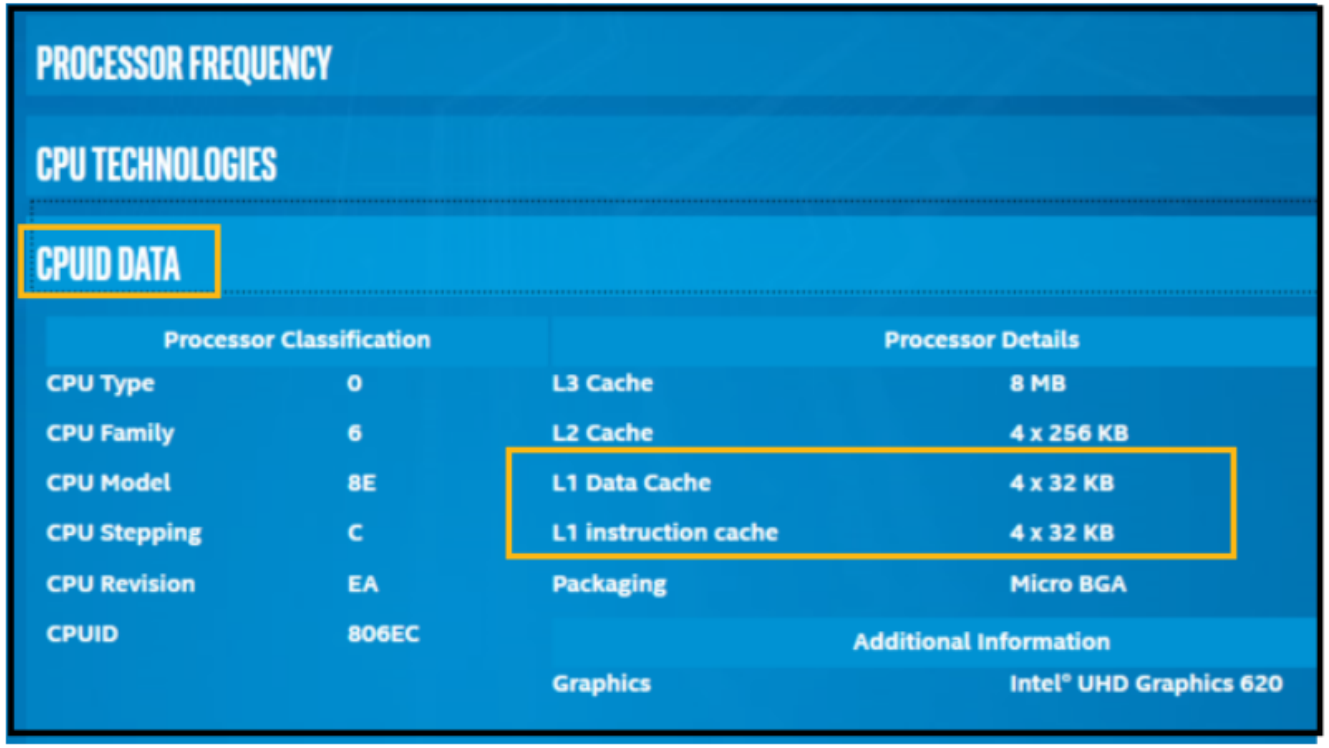
Önbelleğin, hızlı erişim için yeterince küçük, daha fazla veriyi aynı anda bulundurabilmek adına da yeterince büyük olması gerekmektedir.

Mikromimari tasarımcıları, benchmark analizleri sonrasında önbellekleri birden fazla seviyeye bölmeyi uygun bulmuşlardır:

İşlemciye en yakın, en hızlı ama en küçük önbellek
Seviye-1 önbellek (L1 $)

İşlemciye daha uzak, daha yavaş ama daha büyük önbellek
Seviye-2 önbellek (L2 $)

İşlemciye en uzak, en yavaş ama en büyük önbellek
Seviye-3 önbellek (L3 $)



CACHE TYPES - POLICIES

Önbellek görüldüğü üzere kısıtlı bir büyüklüğe sahiptir. Akla gelen bazı sorular:

İşlemci load instructionda aradığı adresi önbellekte bulamazsa ne olacak?

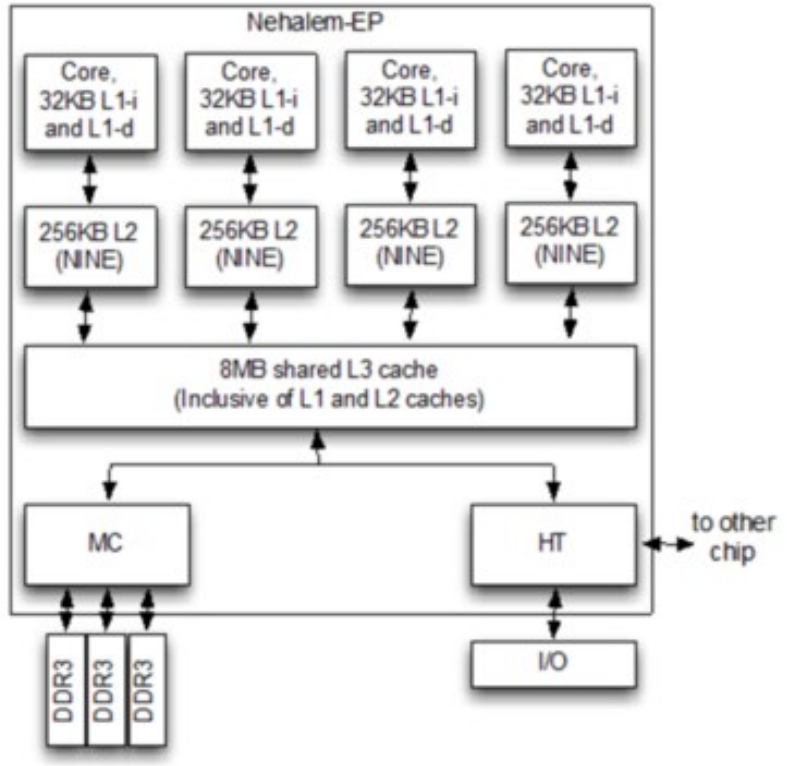
İşlemci store instructionda aradığı adresi önbellekte bulamazsa ne olacak?

İşlemci store instructionda aradığı adresi önbellekte bulursa ne olacak?

Önbelleğe getirilecek olan veri hangi veri yerine yerleştirilecek (replacement policy?)

İstenilen adresin önbellekte olup olmadığı nasıl anlaşılacak?

İstenilen adres önbellekte varsa tam olarak nerede?



DIRECT-MAPPED CACHE

Herhangi bir veri, önbellekte sadece tek bir yerde bulunabiliyorsa, buna direct-mapped cache denir.

En basit gerçekleştirilen önbellek çeşididir, çünkü bellek erişim instruction işletileceği zaman adres hesaplandıktan sonra, verinin önbellekte olup olmadığı sadece bir yer kontrol edilerek gerçekleştirilebilir.

Performansı ise daha karmaşık gerçeklemelere göre daha düşüktür. Örnek olarak önbellekte boş ya da uzun süredir erişilmeyen bir verinin yerini kullanmak istese de bu gerçeklemede verinin bulunabileceği yer sabit olduğundan hep aynı yere yazılmak zorundadır.

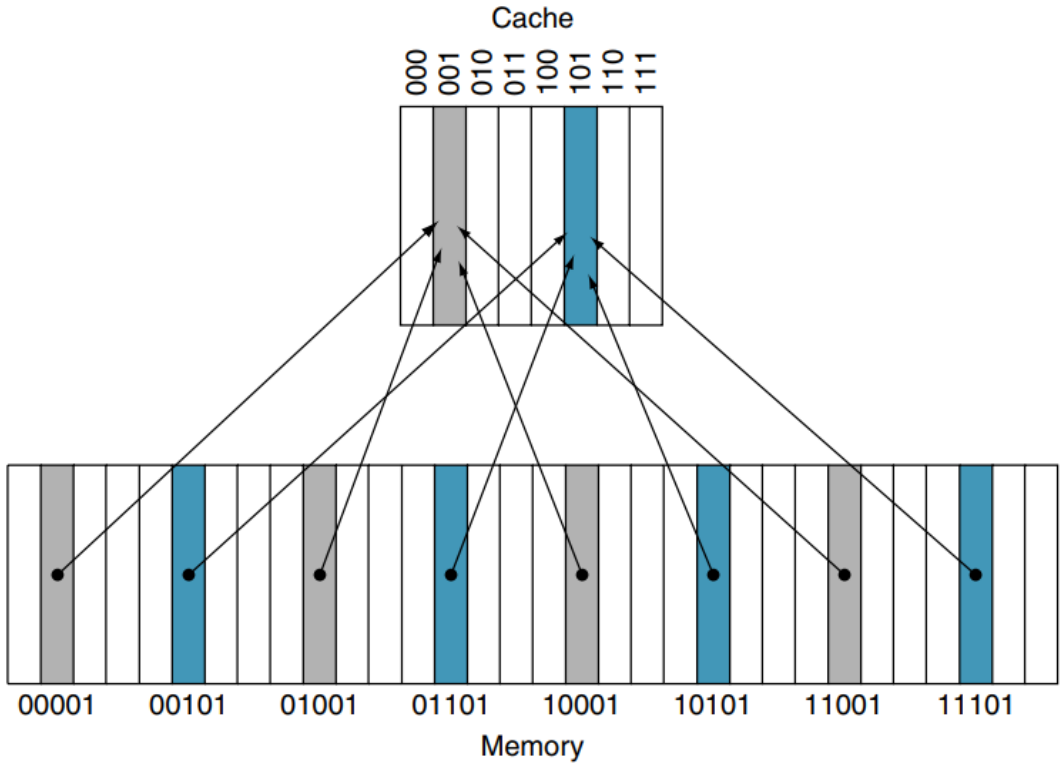
Örnek: 8 bloğa sahip bir önbellek:

Ana bellekte 00001,01001,10001,11001 adresindeki veriler önbellekte her zaman ve sadece 001 adresinde bulunabilirler.

Ana bellekte 00101,01101,10101,11101 adresindeki veriler önbellekte her zaman ve sadece 101 adresinde bulunabilirler.

Ana Bellek Adres (ModN) = Önbellek Adres

N: Önbellekte bulunan blok sayısı



DIRECT-MAPPED CACHE

Bir verinin önbellekte olup olmadığını anlamak için gerekli olan adres bitlerine tag (etiket) adı verilir.

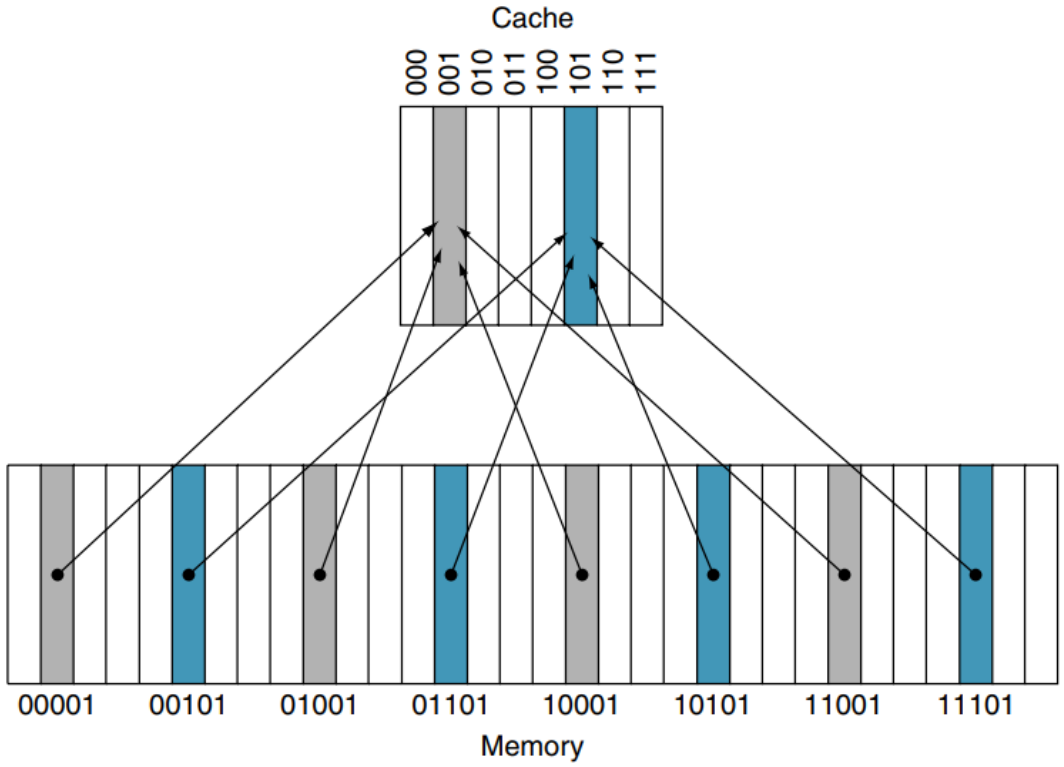
8 bloklu bir önbellek ve 32 bloklu bir bellek senaryosunda, verinin adresinin en düşük (LSB) 3 biti, önbellekte hangi blokta bulunabileceğini belirtir.

Örnek olarak ana bellekte 00101,01101,10101,11101 adresindeki veriler önbellekte her zaman ve sadece 101 adresinde bulunabilirler. Peki önbellekte 101 blok adresinde bulunan veri bu 4 adresteki verilerden hangisidir?

İşte verinin adresinin kalan üst bitleri önbellekte tag adı verilen alanda tutulur.

Veri adresinin üst bitleri tag ile karşılaştırılarak verinin önbellekte olup olmadığı anlaşılır.

1 bitten oluşan Valid (geçerli) alanı ise, önbellekte ilgili bloktaki verinin geçerli ve kullanılabilir bir veri olup olmadığını belirtir.



DIRECT-MAPPED CACHE - ÖRNEK

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (5.9b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (5.9c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (5.9d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (5.9e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (5.9f)	$(10010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$

8 bloklu bir önbellek ve 32 bloklu bir bellek senaryosunda yukarıdaki bellek erişim instructionlar gerçekleştiriliyor.

DIRECT-MAPPED CACHE - ÖRNEK

10110 miss
11010 miss
10110 hit
11010 hit
10000 miss
00011
10000
10010
10000

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Güç açıldığında önbellek

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

10110 miss

Index	V	Tag	Data
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

11010 miss

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

10000 miss

DIRECT-MAPPED CACHE - ÖRNEK

10110 miss
11010 miss
10110 hit
11010 hit
10000 miss
00011 miss
10000 hit
10010 hit
10000 miss

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

00011 miss

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

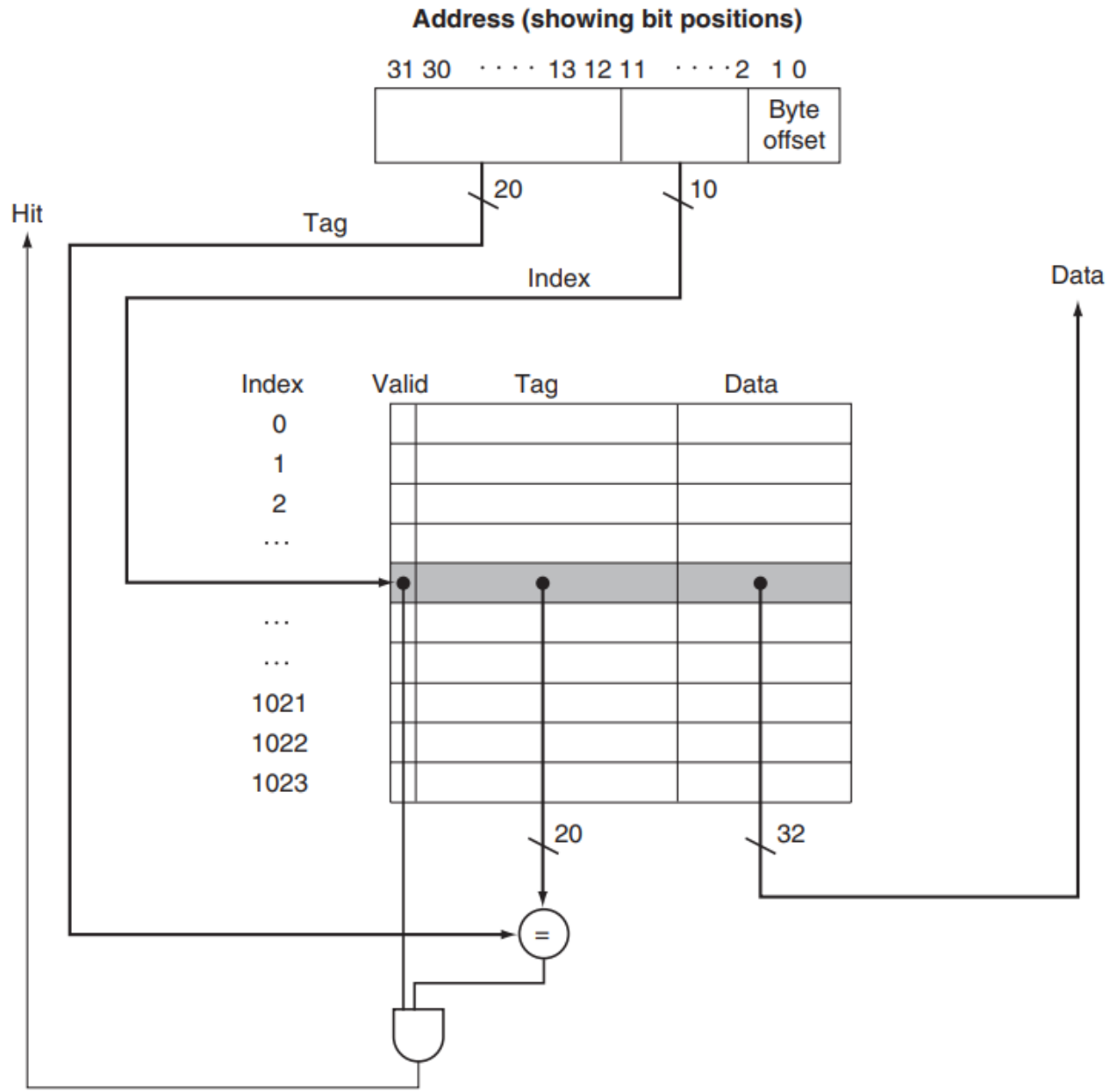
10010 miss

DIRECT-MAPPED CACHE IMPLEMENTATION

1024 blok → 4 kB Cache | 32-bit addressing

Peki önbellekte toplam kaç bit var?

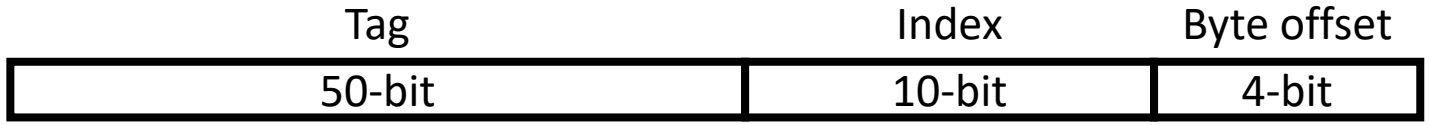
Data : 1024*32 = 32768
Tag : 1024*20 = 20480
Valid : 1024*01 = 1024
Toplam : = 54272
= 53 kb → 6.625 kB

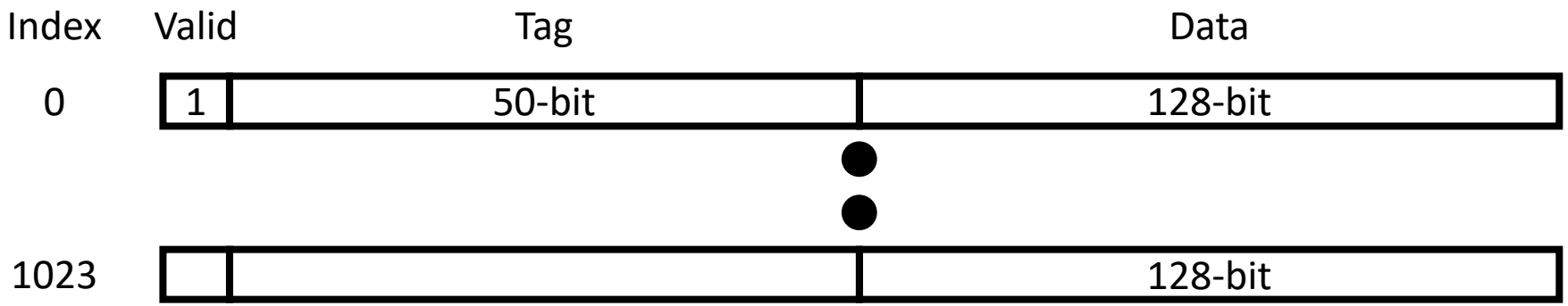


DIRECT-MAPPED CACHE | MULTIPLE WORDS

Direct-mapped 16 kB veri büyüklüğüne sahip ve blok büyüklüğü 4 word (yani her satırda 32-bit yerine 4*32-bit var) olan bir önbellekte 64-bit addressing varsa toplam ne kadar bit kullanılmıştır?

Satır sayısı = 16*1024 / 16





Data : 1024*128 = 131072
Tag : 1024*050 = 51200
Valid : 1024*001 = 1024
Toplam : = 183296
= 179 kb → 22.375 kB