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

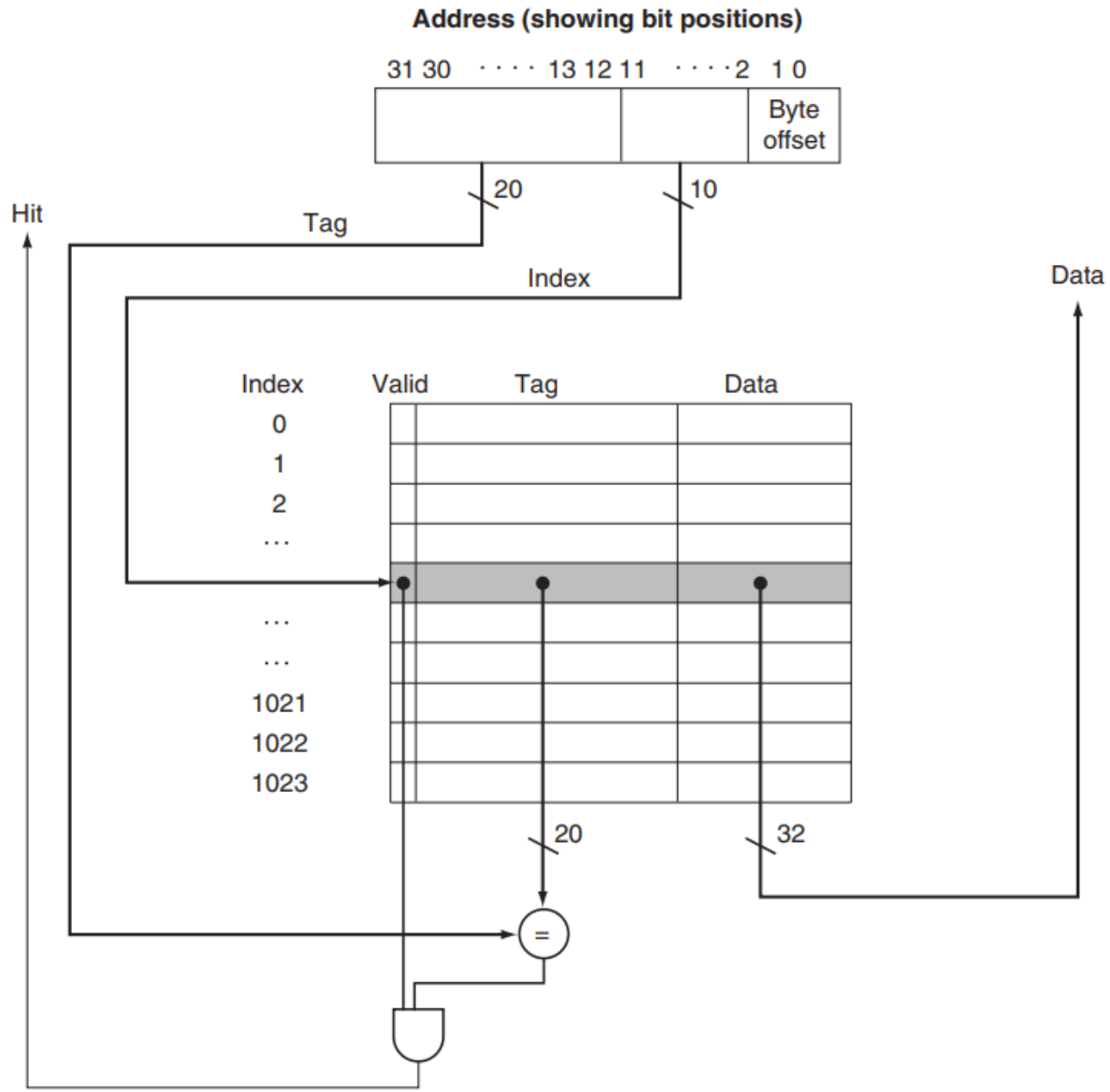
14. Hafta: Önbellek Örnekleri**,** Gömülü sistemler, mikrodenetleyiciler, SoCs

DIRECT-MAPPED CACHE IMPLEMENTATION

1024 blok → 4 kB Cache | 32-bit addressing

Peki önbellekte toplam kaç bit var?

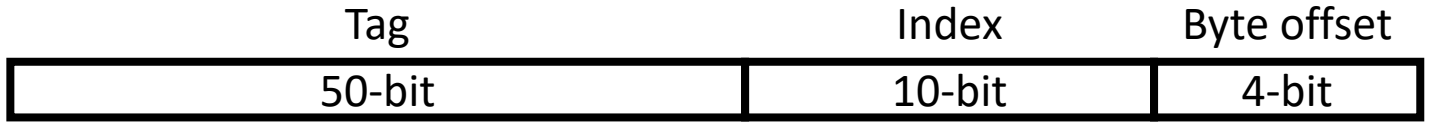
Data : 1024*32 = 32768
Tag : 1024*20 = 20480
Valid : 1024*01 = 1024
Toplam : = 54272
= 53 kb → 6.625 kB

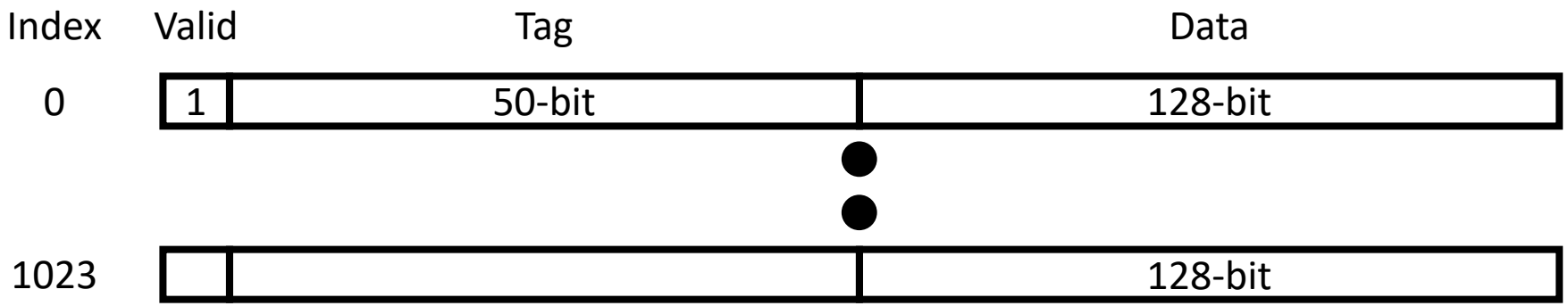


DIRECT-MAPPED CACHE | MULTIPLE WORDS

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

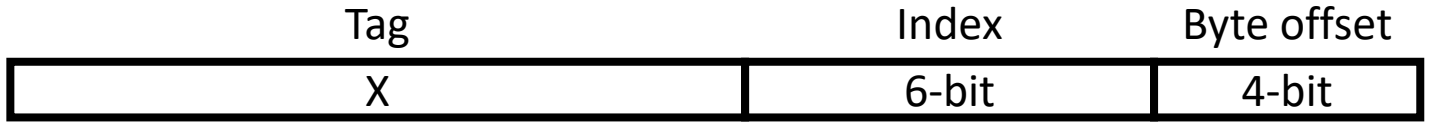


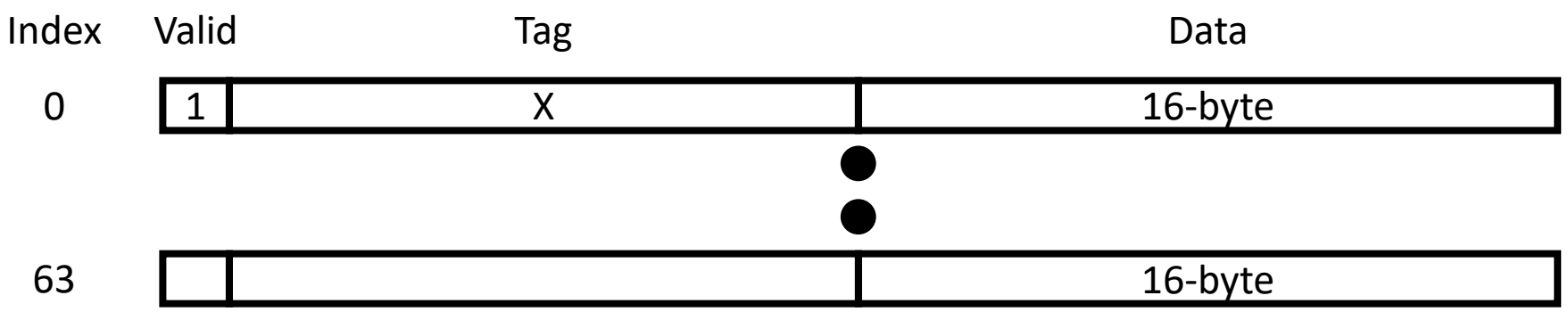


Data : 1024*128 = 131072
Tag : 1024*050 = 51200
Valid : 1024*001 = 1024
Toplam : = 183296
= 179 kb → 22.375 kB

DIRECT-MAPPED CACHE | MULTIPLE WORDS

Direct-mapped ve 64 adet bloktan oluşan bir önbellekte blok büyüklüğü 16 bayt ise 1200 adresindeki veri kaçıncı blokta yer alır?





Adreste ilk 4-bit byte offset için kullanılacak. Sonraki 6-bit ise index, yani mapping için kullanılacak.

→ 1200/16 = 75
→ 75 mod64 = 11
Öyleyse 1200 adresindeki veri, 11 adresindeki blokta bulunur.

0100 1011 0000
→ 0000 : byte offset
→ 001011 : index
→ 001011 = 11 (dec)

CACHE MISS PENALTY

CPU’nun erişmek istediği adres önbellekte yoksa (cache miss) nasıl bir yol izlenecek?

Cache miss gerçekleştiğinde, boru hattında bekleme durumu oluşur (pipeline stall)

Bu sırada ana belleğe okuma isteği gönderilir, veri önbelleğe getirilene kadar (birkaç saat vuruşu) beklenir

Cache miss durumunda ciddi gecikmeler meydana gelebilir, bu gecikmeleri önlemek için:

→ Cache miss durumunu en aza indirecek bir önbellek topoloji belirlenir
→ Ana bellek gecikmesi yerine daha az gecikmeye sahip seviyeli önbellek (L1, L2, L3) yapısı kullanılır

WRITE POLICY

İşlemci belleğe yazma (store) instruction işlettiğinde, yazılacak olan veri nasıl yazılacak?

Eğer veri sadece önbelleğe yazılıp ana belleğe yazılmazsa, önbellek ve ana bellek arasında uyumsuzluk meydana gelecektir (inconsistency).

Bunun önüne geçmek için bir yöntem, önbelleğe veri yazılacağı zaman ana belleğe de verinin yazılmasıdır (write-through).

Write-through yönteminin dezavantajı, her store instructionda ana belleğe de yazma işleminin gerçekleştirilmesidir ve ana belleğe yazmak en az 100 saat vuruşu kadar gecikmeye neden olmaktadır. Bu sırada işlemci beklemek durumunda kalır (pipeline stall).

Bu gecikmeyi ortadan kaldırmak için önerilen bir yöntem yazma buffer (write buffer) modülüdür. Store instruction işletildiğinde yazılacak olan veri write buffer'a yazılır ve işlemci beklemeden çalışmaya devam eder. Ana belleğe yazma işlemi tamamlandıktan sonra write buffer'daki veri silinir. Eğer bilgisayar programı sürekli store instruction içeriyorsa bir süre sonra write buffer dolar ve yeni gelen store instructionları nedeniyle bufferda yer açılana kadar işlemci yine beklemek durumunda kalır.

WRITE POLICY

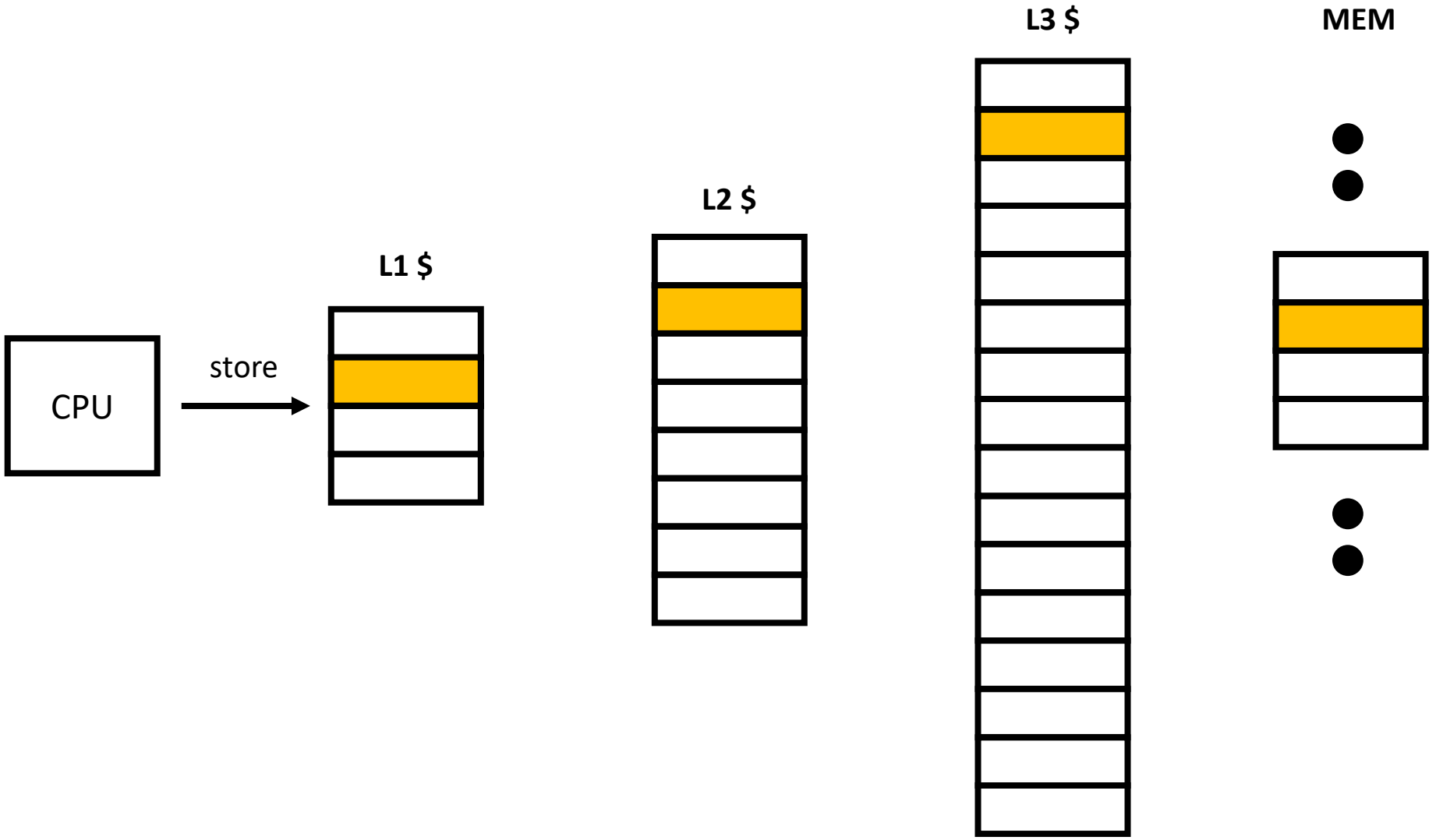
Write-through yöntemine bir diğer alternatif de write-back yöntemidir.

Write-back yönteminde, store instruction'da yazılacak olan veri sadece önbellekte ilgili bloğa yazılır. Blok önbellekte durduğu müddetçe ana belleğe yazma işlemi gerçekleştirilmez.

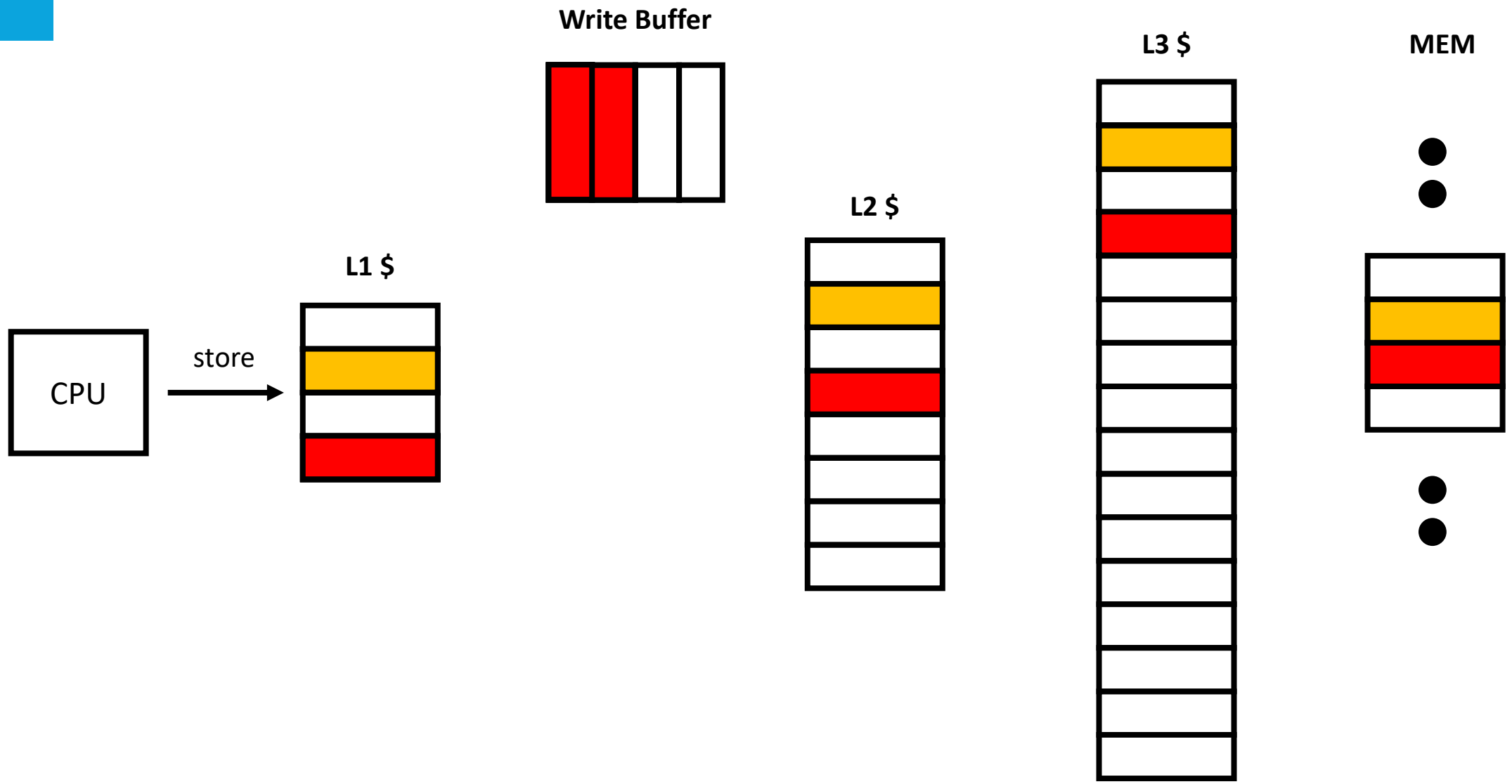
Ne zaman ki ilgili blok başka bir blokla değiştirilecektir, o zaman ana belleğe yazma işlemi gerçekleştirilir.

Write-back yöntemi, write-through yöntemine göre performans iyileşmesi sağlar fakat gerçeklenmesi donanımsal olarak daha karmaşıktır.

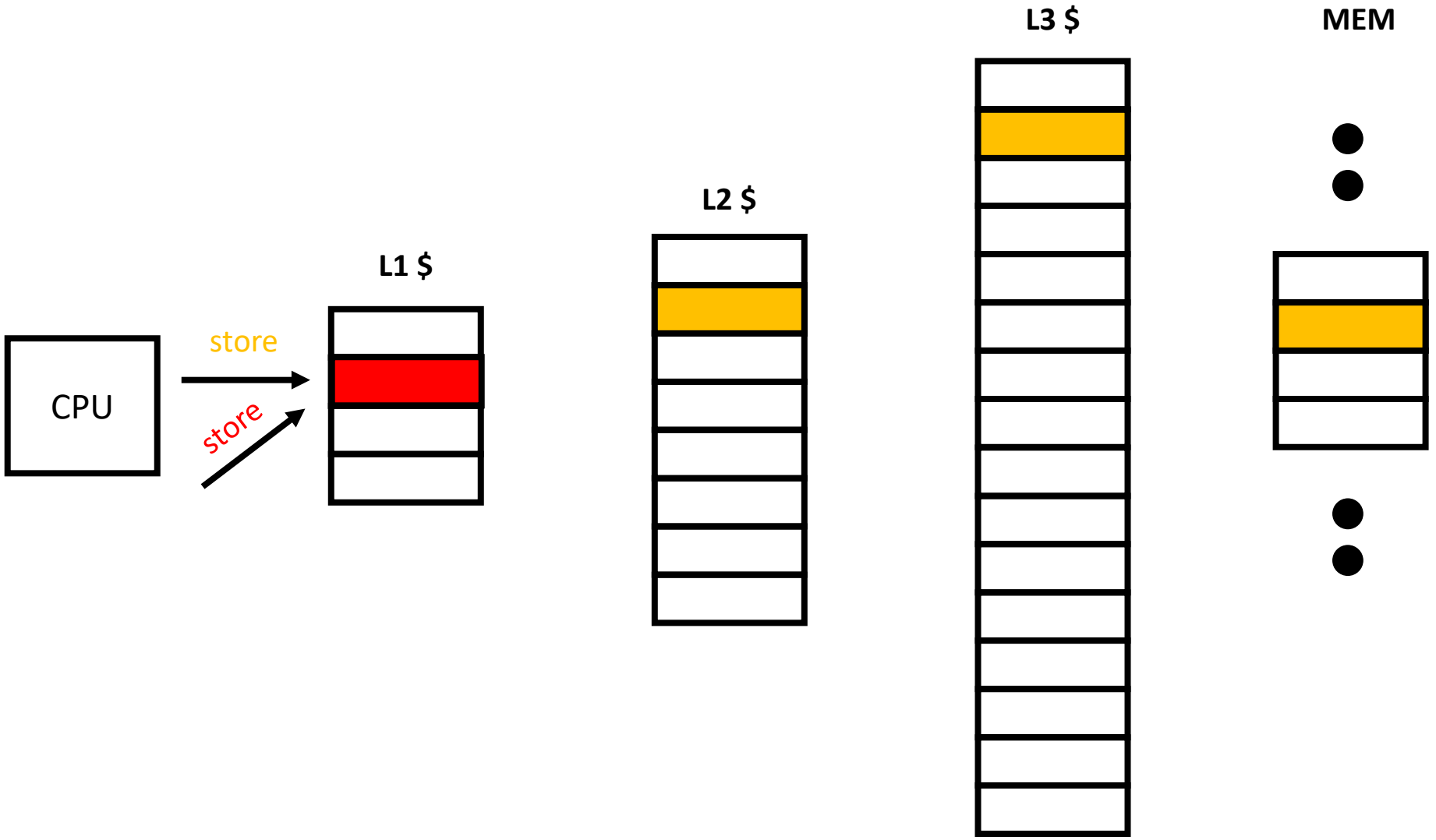
WRITE POLICY – WRITE THROUGH



WRITE POLICY – WRITE BUFFER



WRITE POLICY – WRITE BACK



CACHE ASSOCIATIVITY

Daha önce direct-mapped önbellek tasarım görmüştük. Bu önbellek çeşidinde ana bellekteki bir adresteki veri önbellekte yalnızca kendisine ait ve ayrılmış olan bloğa yazılabiliyordu.

Bunun tam tersi de mümkündür, yani ana bellekteki bir adresteki veri, önbellekte herhangi bir bloğa yazılabilir. Bu önbellek çeşidine fully-associative denir, yani ana bellekteki bir veri önbellekteki herhangi bir blokla ilişkilendirilebilir.

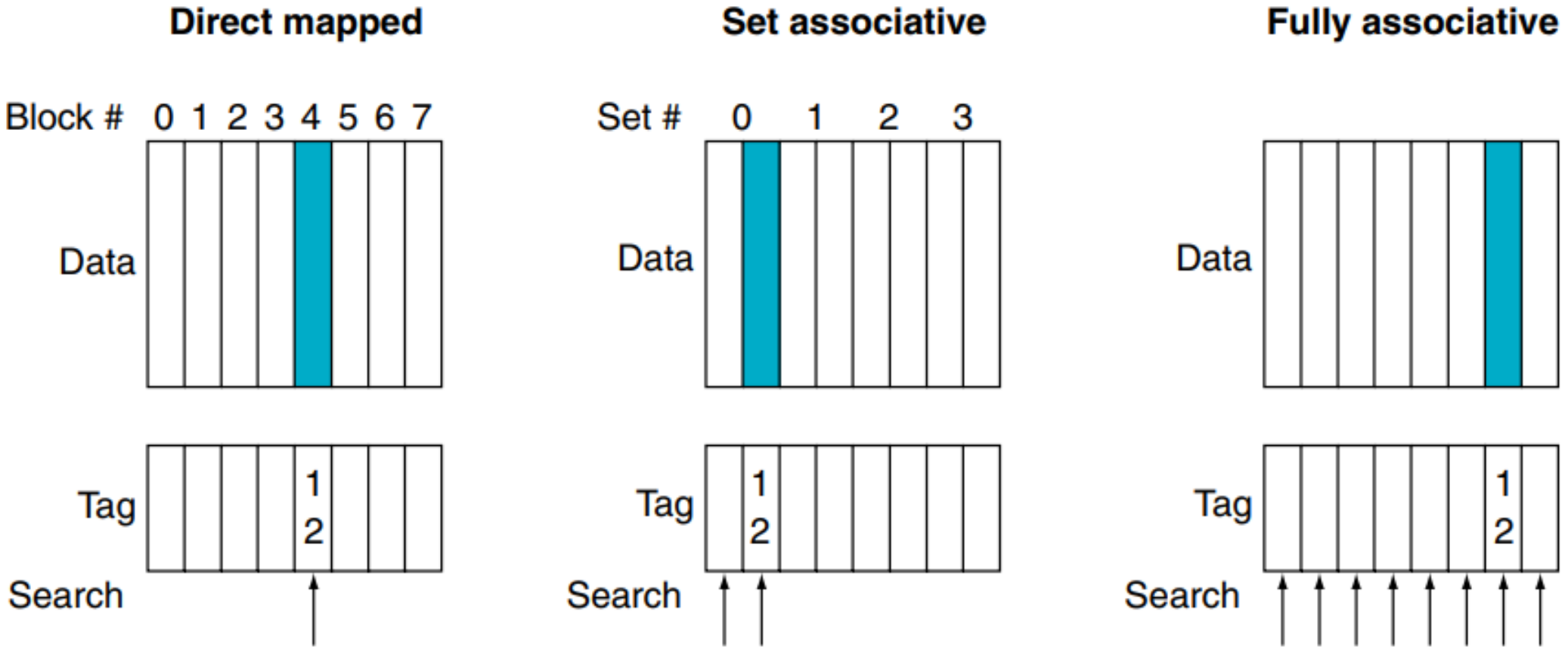
Fully-associative önbellek çeşidi direct-mapped önbelleğe göre müthiş bir esneklik sağlamaktadır. Fakat bir adresteki verinin önbellekte olup olmadığını kontrol etme algoritması açısından da karmaşıklık oluşturmaktadır.

Direct-mapped önbellekte bir adresteki verinin önbellekte olup olmadığı sadece tek bir bloktaki tag bitleri kontrol edilerek anlaşılabiliyorken, fully-associative bir önbellekte bütün blokların kontrol edilmesi gerekecektir.

Fully-associative önbellekte verinin hangi blokta olduğunu anlamak için yapılacak olan etiket karşılaştırması performans açısından bütün bloklarda aynı anda paralel bir şekilde yapılmalıdır. Bu da donanımsal karmaşıklığı arttırır, bu yüzden fully-associative bir önbelleğin çok büyük olması beklenemez.

Direct-mapped ve fully-associative arasındaki yöntem set-associative önbellektir. Bu önbellekte ise bir veri set sayısı kadar blokta yer alabilir. Örnek olarak 2-way set-associative bir önbellekte bir veri 2 bloktan bir tanesinde olabilir.

CACHE ASSOCIATIVITY



One-way set associative (direct mapped)

Set	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

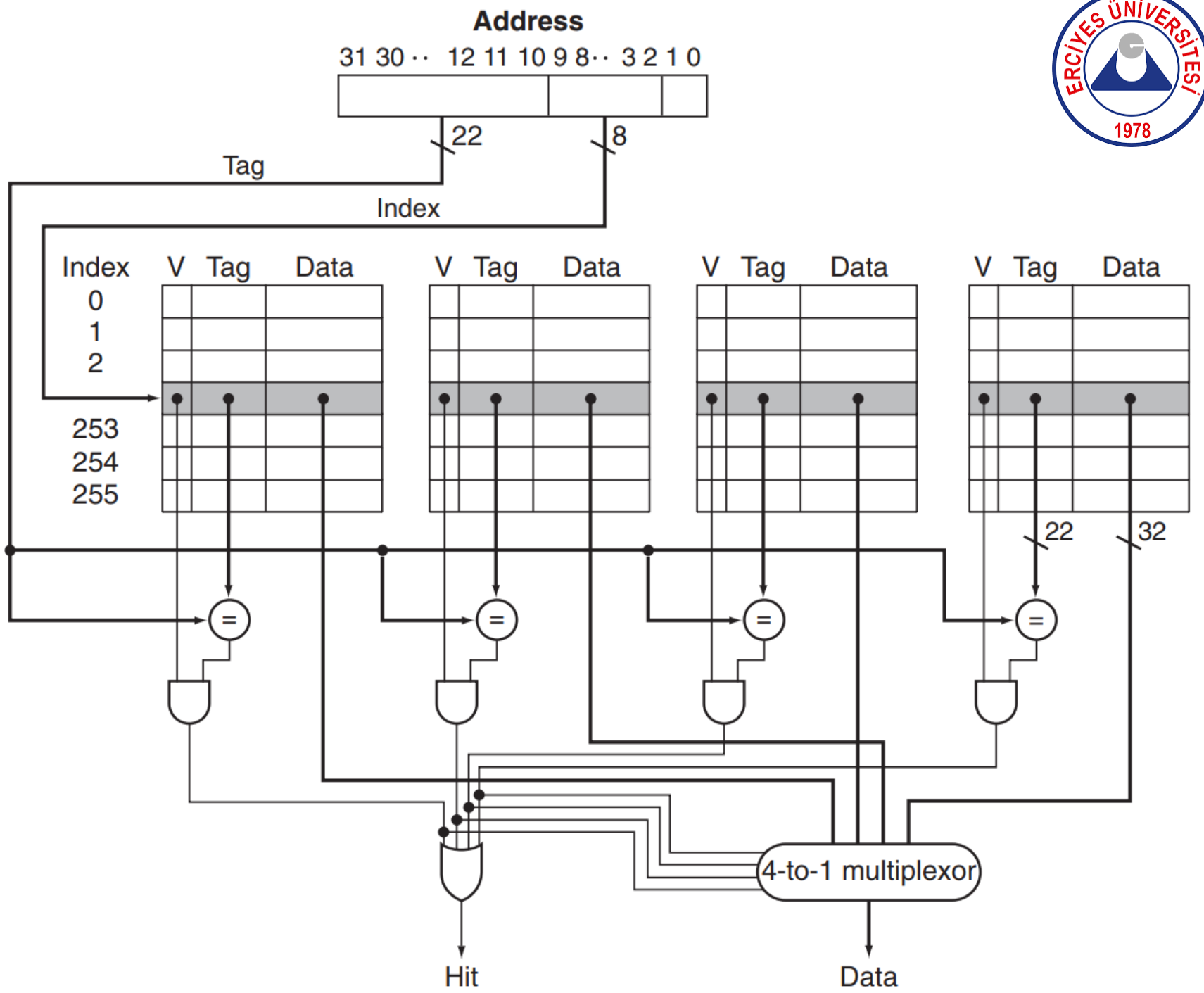
Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

4-way set-associative
4-byte block size
256 blocks

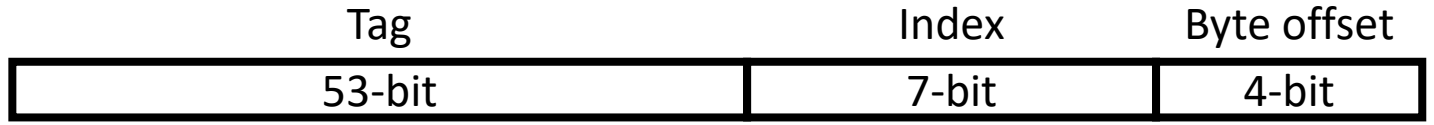
4 kB cache

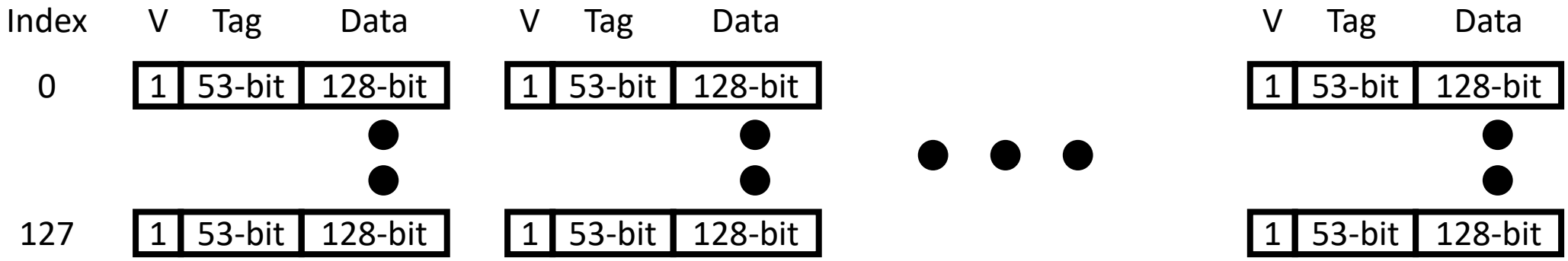


CACHE EXAMPLES

8-way set associative, 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 / 8





Direct-Mapped

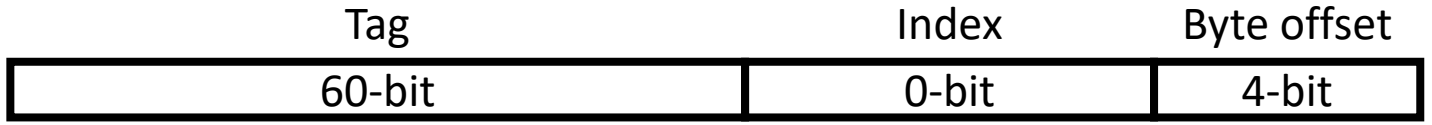
Data : 1024*128 = 131072
Tag : 1024*050 = 51200
Valid : 1024*001 = 1024
Toplam : = 183296
= 179 kb → 22.375 kB

Data : 128*128*8 = 131072
Tag : 053*128*8 = 54272
Valid : 001*128*8 = 1024
Toplam : = 186368
= 182 kb → 22.75 kB

CACHE EXAMPLES

Fully-associative, 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



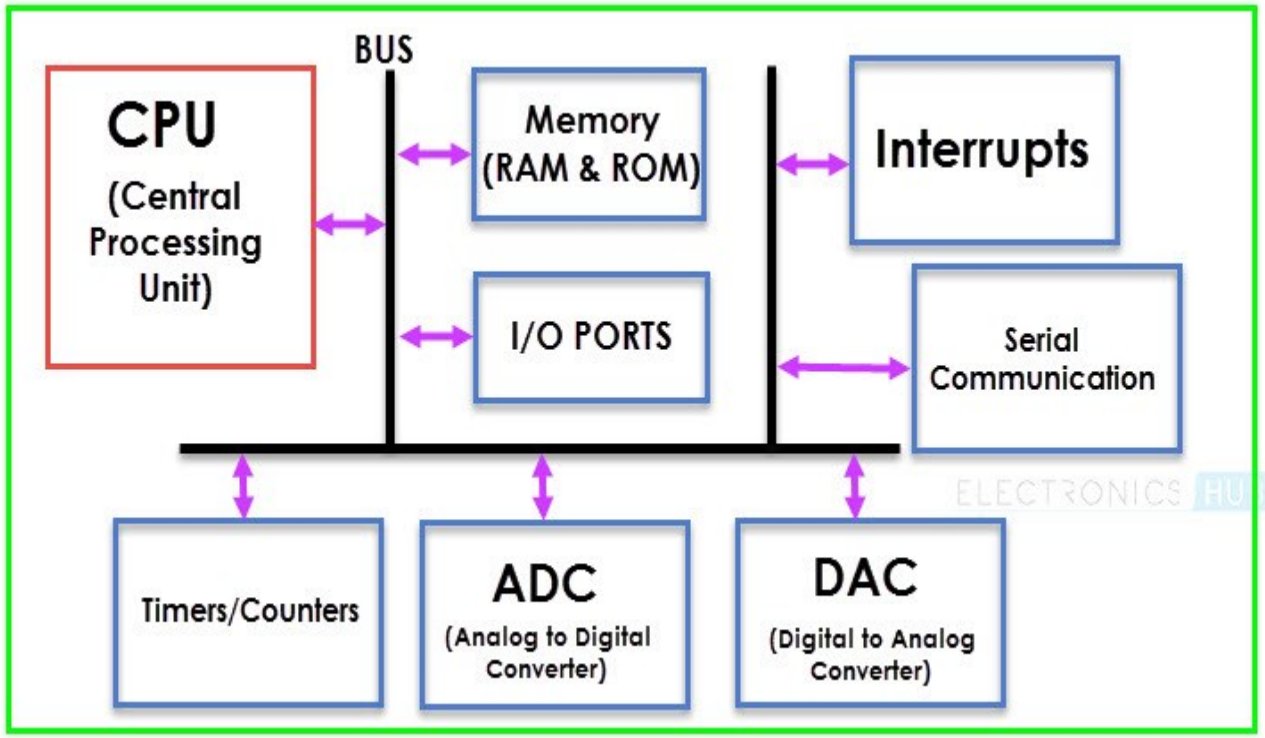
Direct-Mapped

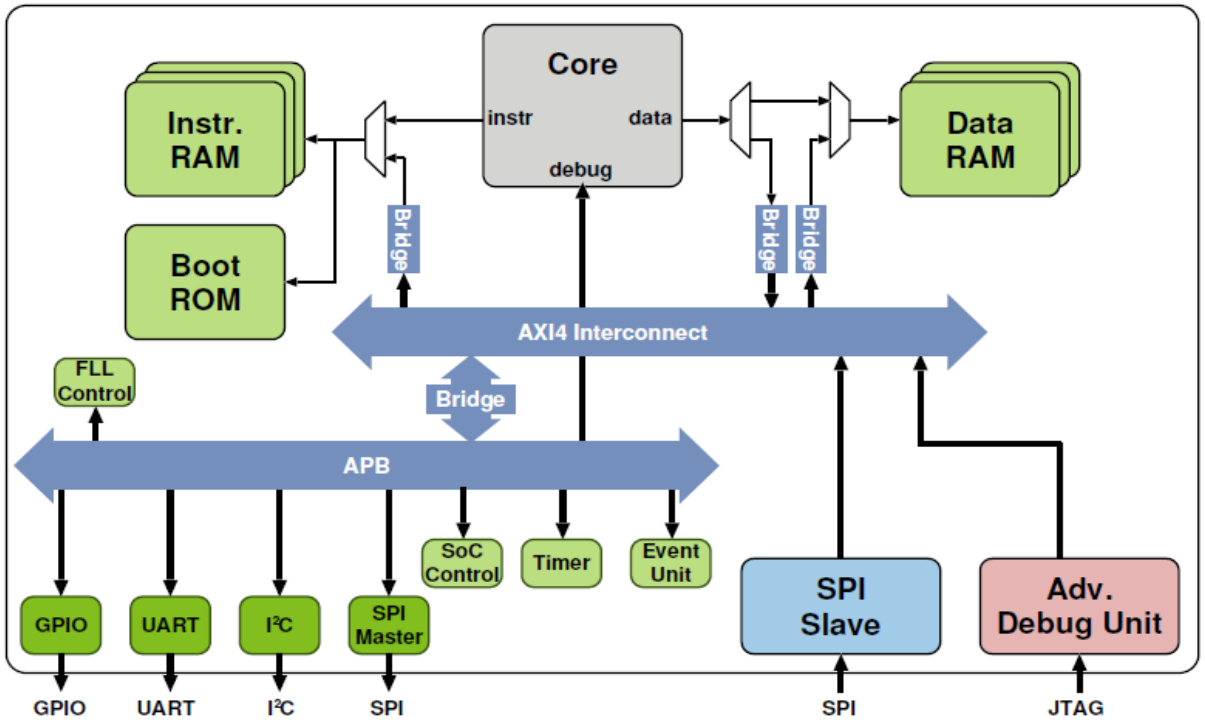
Data : 1024*128 = 131072
Tag : 1024*050 = 51200
Valid : 1024*001 = 1024
Toplam : = 183296
= 179 kb → 22.375 kB

Data : 128*1024 = 131072
Tag : 060*1024 = 61440
Valid : 001*1024 = 1024
Toplam : = 193536
= 189 kb → 23.625 kB

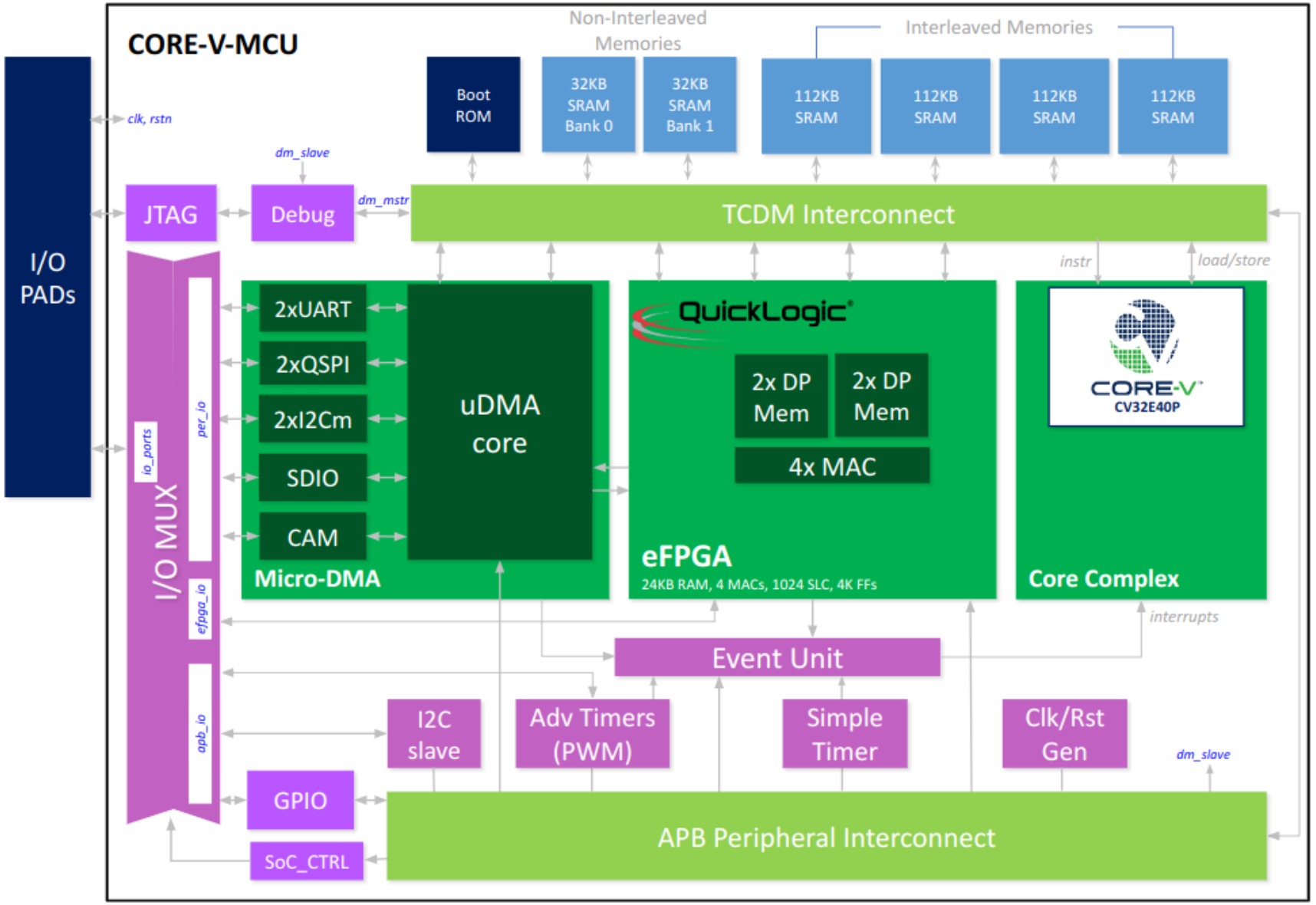
MICROCONTROLLER - MCU

A very basic MCU is composed of a CPU core, a memory to run applications, non-volatile internal or external memory to store applications, I/O peripherals, and a bus architecture system to connect all those together





MICROCONTROLLER - MCU



https://github.com/openhwgroup/core-v-mcu

CPU Core

A core simply has an ALU to perform arithmetic and logical operations, a program counter (PC) to trace instructions, a small memory called as register file, a fetch unit to get next instruction from the memory, an instruction decoding unit which provides control signals for multiplexers and ALU operation choices, a load/store unit to access memory to load data to register file or store data to memory

Most utilized cores for MCUs: ARM Cortex-M Series

- Requires up-front licensing fee and royalty per unit production sales

Royalty-free cores: RISC-V ISA based cores

- Numerous open-source verified CPU core implementations

8-, 16- and 32-bit CPU cores for embedded applications are available for MCUs

- 8 and 16-bit cores are usually for legacy systems and applications while 32-bit is mainstream

64-bit CPU cores for high-end SoCs, MPUs, desktop PCs

- Can run an OS such as Linux, Windows. For embedded, ARM Cortex A53, A72 etc.

CPU Core

64-bit MPU, SoC, Application Processor

- Run Linux, Android
- L1, L2 $
- Memory management unit (MMU)
- External high-speed memory interface (DDR)
- High speed I/Os: UHD HDMI, 10/25G ETH, PCIe etc.
- Graphics accelerators – ARM Mali etc.
- Many core architectures
- Parallel applications, multi-threading

32-bit MCU

- Run Bare-metal (standalone), FreeRTOS, Zephyr
- No or small L1 $
- No Memory management unit (MMU)
- Internal small RAM and ROM
- Low speed I/Os: UART, SPI, I2C, 10/100 ETH etc.
- No Graphics accelerators
- Single core architecture
- Sequential flow

ARM - AMBA

- AMBA (Advanced Microcontroller Bus Architecture)
  - AMBA Rev.2 [1999]
    - APB (Advanced Peripheral Bus)
    - AHB (Advanced High-Performance Bus)
  - AMBA Rev.3 [2003]
    - AXI3 (Advanced eXtensible Interface – 3)
  - AMBA Rev.4 [2010]
    - AXI4 (Advanced eXtensible Interface – 4)
      - AXI4-Full
      - AXI4-Lite
      - AXI4-Stream

TRENDS IN COMPUTER ARCHITECTURE

- IoT (Internet of Things)

- AI Acceleration

- Edge Computing

- 5G/6G

- Cloud Computing

- Computer Vision

- …