模拟电路与数字系统 III

计算机组成原理与接口技术
——基于 MIPS 架构

Chapter 4
MEMORY MANAGEMENT

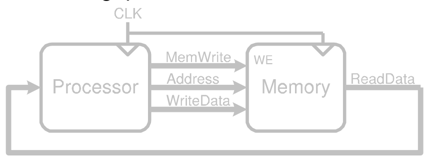
张江山
zhangjs@hust.edu.cn
信息工程系

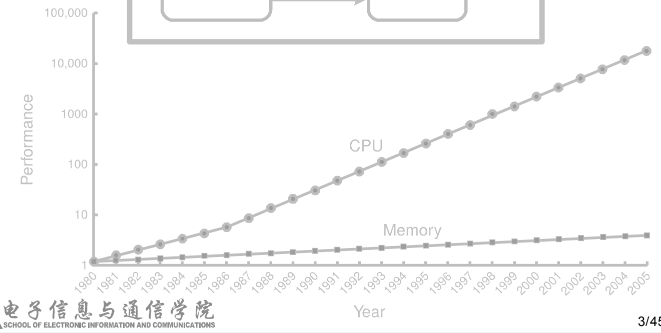
Content & Objectives

- **Content**
  - Memory Hierarchy
  - Cache
  - Virtual Memory
- **Objectives**
  - Understanding memory addressing
  - Master cache mapping
  - Understanding the management of virtual memory

1 Memory Hierarchy

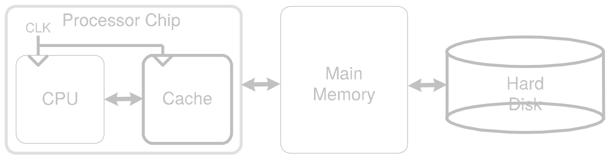
- Computer performance depends on the memory system
  - The performance gap between the CPU and memory

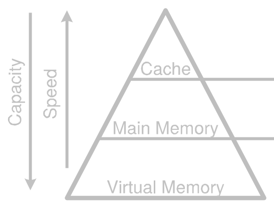




1 Memory Hierarchy

- **Typical Memory Hierarchy**
  - Cache: Temporarily place loaded data in faster memory for reuse

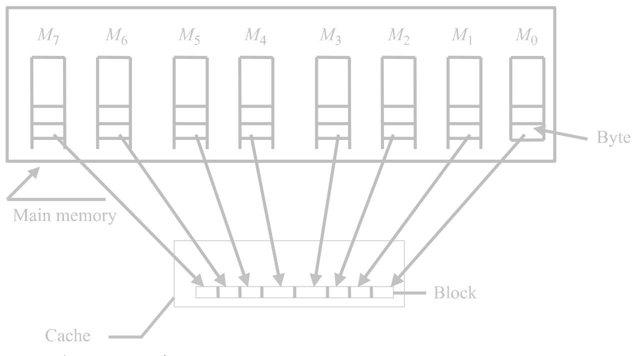




Technology	Cost/GB	Access Time
SRAM	~ $10,000	~ 1 ns
DRAM	~ $100	~ 100 ns
Hard Disk	~ $1	~ 10,000,000 ns

2 Cache

- **Cache Hit**
  - Occurs when the requested data can be found in a cache
  - The larger cache capacity, the higher the hit rate



2 Cache

- **Cache Read Policy**
  - Cache Hit
    - Read Data from Cache
  - Cache Miss
    - Cache Line Fill:
      - Fill an entire line of data into cache
    - Cache Replacement Policy
      - LRU (Least Recently Used replacement policy)
      - FIFO (First-in first-out)
      - Random replacement

2 Cache

- **Cache Write Policy**
  - Cache Hit
    - Write-through
      - Write to the cache, and write to the main memory at the same time
    - Write-back
      - Write to the main memory is postponed, until a replacement is needed
      - A Modify-flag-bit be required in the Cache line
  - Cache Miss
    - Write-allocate
      - The main memory block is updated, and brought to the cache
    - Write-no-allocate
      - The main memory block is updated, not brought to the cache

2 Cache

- **Cache Organization**
  - Memory address is mapped to a unique block in the cache
  - $Cache(B, S, W)$ for N-bit address space
    - B-byte per data block
    - S-set of data blocks organized to cache
    - W-way per set (1-line per way, 1-block per line)
    - $Set\_id = Mem\_address / B \% S$
    - $Cache\_size = S \times W \times (1 + N - \log_2 B - \log_2 S + 8 \times B)$ (bits)

Memory Address ($N-1$ … 0)

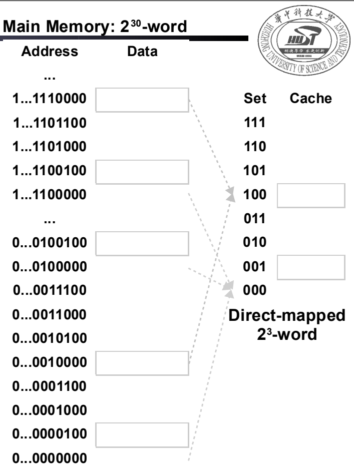
Tag	SetID	BlockByteOffset
$N - \log_2 B - \log_2 S$ bits	$\log_2 S$ bits	$\log_2 B$ bits

Cache Line

Valid	Tag	1 Block
1 bit	$N - \log_2 B - \log_2 S$ bits	$8 \times B$ bits

2 Cache

- **Direct-mapped (1-way)**
  - $Cache(4B, 8S, 1W)$ for 32-bit addr. space
- **Set-associative**
  - $Cache(4B, 4S, 2W)$ for 32-bit addr. space
- **Fully-associative (1-set)**
  - $Cache(4B, 1S, 8W)$ for 32-bit addr. space



Set-associative
2^3-word

Set	Way 1	Way 0
11		
10		
01		
00		

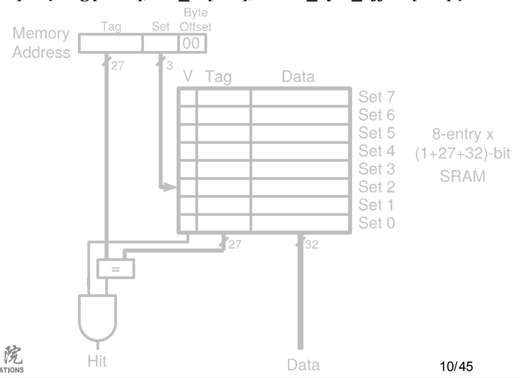
Fully-associative
2^3-word

Way 7	Way 6	Way 5	Way 4	Way 3	Way 2	Way 1	Way 0

2.1 Direct-mapped

- **Example 1: Direct-mapped**
  - $Cache(4B, 8S, 1W)$ for 32-bit address space
  - 4-byte per block, 8-set, 1-way per set

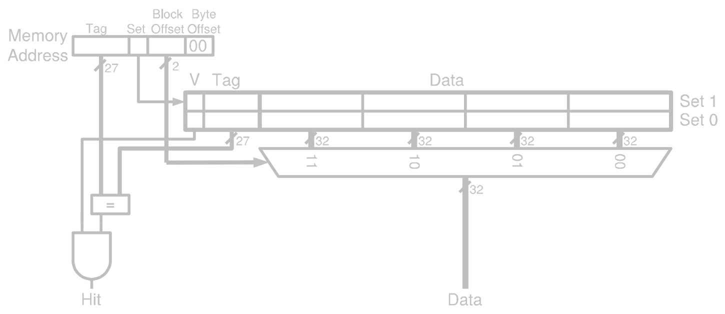
  $Mem\_address[31:0] = \{ Tag[26:0], Set\_id[2:0], Block\_byte\_offset[1:0] \}$



2.1 Direct-mapped

- **Example 2: Direct-mapped**
  - $Cache(16B, 2S, 1W)$ for 32-bit address space
  - 16 bytes per block, 2-set, 1-way per set

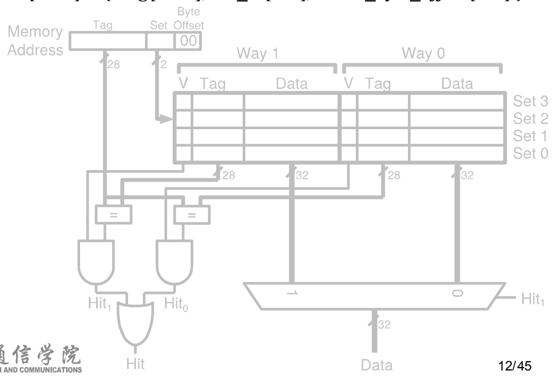
  $Mem\_address[31:0] = \{ Tag[26:0], Set\_id[0], Block\_byte\_offset[3:0] \}$



2.2 Set-associative

- **Example 3: Set-associative**
  - $Cache(4B, 4S, 2W)$ for 32-bit address space
  - 4-byte per block, 4-set, 2-way per set

  $Mem\_address[31:0] = \{ Tag[27:0], Set\_id[1:0], Block\_byte\_offset[1:0] \}$



2.3 Fully-associative

- **Example 4: Fully-associative**
  - *Cache*(4*B*, 1*S*, 8*W*) for 32-bit address space
  - 4-byte per block, 1-set, 8-way per set

  Mem_address[31:0] = { *Tag*[29:0], *Block_byte_offset*[1:0] }

Way 7			Way 6			Way 5			Way 4			Way 3			Way 2			Way 1			Way 0		
V	Tag	Data	V	Tag	Data	V	Tag	Data	V	Tag	Data	V	Tag	Data	V	Tag	Data	V	Tag	Data	V	Tag	Data

2.5 Performance Analysis

- **Example 5:**
  - *Cache*(1*B*, 8*S*, 1*W*) for 8-bit address space $R_{\text{CacheHit}} = 2/7 = 29\%$
  - 1-byte per block, 8-set, 1-way per set
    - ➢ *Mem_address*[7:0] = { *Tag*[4:0], *Set_id*[2:0] }
  - To load data from memory addresses:
    - ➢ 26_H, 22_H, 26_H, 18_H, 16_H, 18_H, 02_H

Set	V	Tag 5-bit	Data 1-byte
111			
110		00100	xx
101			
100			
011			
010		00100	xx
001			
000		00011	xx

```
Miss *(0010_0110) → SetID = 0x26 % 8 = 6
Miss *(0010_0010) → SetID = 0x22 % 8 = 2
HIT  *(0010_0110) → SetID = 0x26 % 8 = 6
Miss *(0001_1000) → SetID = 0x18 % 8 = 0
Miss *(0001_0110) → SetID = 0x16 % 8 = 6
HIT  *(0001_1000) → SetID = 0x18 % 8 = 0
Miss *(0000_0010) → SetID = 0x02 % 8 = 2
```

2.5 Performance Analysis

- **Example 6:**
  - *Cache*(1*B*, 4*S*, 1*W*) for 5-bit address space $R_{\text{CacheHit}} = 0/5 = 0\%$
  - 1-byte per block, 4-set, 1-way per set
    - ➢ *Mem_address*[4:0] = { *Tag*[2:0], *Set_id*[1:0] }
  - To load data from memory addresses: `0, 8, 0, 6, 8`

Set	V	Tag 3-bit	Data 1-byte
11			
10		001	xx
01			
00		010	xx

```
Miss *(0_0000) → SetID = 0x0 % 4 = 0
Miss *(0_1000) → SetID = 0x8 % 4 = 0
Miss *(0_0000) → SetID = 0x0 % 4 = 0
Miss *(0_0110) → SetID = 0x6 % 4 = 2
Miss *(0_1000) → SetID = 0x8 % 4 = 0
```

2.5 Performance Analysis

- **Example 7:**
  - *Cache*(1*B*, 2*S*, 2*W*) for 5-bit address space $R_{\text{CacheHit}} = 1/5 = 20\%$
  - 1-byte per block, 2-set, 2-way per set
    - ➢ *Mem_address*[4:0] = { *Tag*[3:0], *Set_id*[0] }
  - To load data from memory addresses: `0, 8, 0, 6, 8`

	Way 1			Way 0		
Set	V	Tag 4-bit	Data 1-byte	V	Tag 4-bit	Data 1-byte
1						
0		0100	xx		0000	xx

```
Miss *(0_0000) → SetID = 0x0 % 2 = 0
Miss *(0_1000) → SetID = 0x8 % 2 = 0
HIT  *(0_0000) → SetID = 0x0 % 2 = 0
Miss *(0_0110) → SetID = 0x6 % 2 = 0
Miss *(0_1000) → SetID = 0x8 % 2 = 0
```

2.5 Performance Analysis

- **Example 8:**
  - *Cache*(1*B*, 1*S*, 4*W*) for 5-bit address space $R_{\text{CacheHit}} = 2/5 = 40\%$
  - 1-byte per block, 1-set, 4-way per set
    - ➢ *Mem_address*[4:0] = *Tag*[4:0]
  - load data from memory addresses: `0, 8, 0, 6, 8`

Way 3			Way 2			Way 1			Way 0		
V	Tag 5-bit	Data 1-byte	V	Tag 5-bit	Data 1-byte	V	Tag 5-bit	Data 1-byte	V	Tag 5-bit	Data 1-byte
				00110	xx		01000	xx		00000	xx

```
Miss *(00000)
Miss *(01000)
HIT  *(00000)
Miss *(00110)
HIT  *(01000)
```

2.5 Performance Analysis

- **Example 9: Program Performance Analysis**
  1. Direct-mapped *Cache*(4*B*, 8*S*, 1*W*) for 8-bit address space $R_{\text{CacheHit}} = ?$

     Mem_address[7:0] = { *Tag*[2:0], *Set_id*[2:0], *Block_byte_offset*[1:0] }

Address	2-byte	2-byte
00111100	a[15][1]	a[15][0]
...	...	...
00100000	a[8][1]	a[8][0]
00011100	a[7][1]	a[7][0]
00011000	a[6][1]	a[6][0]
00010100	a[5][1]	a[5][0]
00010000	a[4][1]	a[4][0]
00001100	a[3][1]	a[3][0]
00001000	a[2][1]	a[2][0]
00000100	a[1][1]	a[1][0]
00000000	a[0][1]	a[0][0]

Set	V	Tag 3-bit	Data 4-byte
111			
110			
101			
100			
011			
010			
001			
000			

```
short a[16][2];
int assign_array_rows() {
  int i, j, sum = 0;
  for(i=0; i<16; i++)
    for(j=0; j<2; j++)
      sum += a[i][j];
  return sum;
}
int assign_array_cols() {
  int i, j, sum = 0;
  for(j=0; j<2; j++)
    for(i=0; i<16; i++)
      sum += a[i][j];
  return sum;
}
```

2.5 Performance Analysis

- **Example 9: Program Performance Analysis**
  1. Direct-mapped $Cache(4B, 8S, 1W)$ for 8-bit address space $R_{\text{CacheHit}} = 50\%$

$Mem\_address[7:0] = \{ Tag[2:0], Set\_id[2:0], Block\_byte\_offset[1:0] \}$

`assign_array_rows`

Address	2-byte	2-byte
00111100	a[15][1]	a[15][0]
...	...	...
00100000	a[8][1]	a[8][0]
00011100	a[7][1]	a[7][0]
00011000	a[6][1]	a[6][0]
00010100	a[5][1]	a[5][0]
00010000	a[4][1]	a[4][0]
00001100	a[3][1]	a[3][0]
00001000	a[2][1]	a[2][0]
00000100	a[1][1]	a[1][0]
00000000	a[0][1]	a[0][0]

Set	V	Tag 3-bit	Data 4-byte
111		000	a[7][1] a[7][0]
110		000	a[6][1] a[6][0]
101		000	a[5][1] a[5][0]
100		000	a[4][1] a[4][0]
011		000	a[3][1] a[3][0]
010		000	a[2][1] a[2][0]
001		000	a[9][1] a[9][0]
000		000	a[8][1] a[8][0]

```
a[i][0]           a[i][1]
 [0][0] MIS FILL   [0][1] HIT
 [1][0] MIS FILL   [1][1] HIT
 [2][0] MIS FILL   [2][1] HIT
 [3][0] MIS FILL   [3][1] HIT
 [4][0] MIS FILL   [4][1] HIT
 [5][0] MIS FILL   [5][1] HIT
 [6][0] MIS FILL   [6][1] HIT
 [7][0] MIS FILL   [7][1] HIT
 [8][0] MIS FILL   [8][1] HIT
 [9][0] MIS FILL   [9][1] HIT
[10][0] MIS FILL  [10][1] HIT
[11][0] MIS FILL  [11][1] HIT
[12][0] MIS FILL  [12][1] HIT
[13][0] MIS FILL  [13][1] HIT
[14][0] MIS FILL  [14][1] HIT
[15][0] MIS FILL  [15][1] HIT
```

2.5 Performance Analysis

- **Example 9: Program Performance Analysis**
  1. Direct-mapped $Cache(4B, 8S, 1W)$ for 8-bit address space $R_{\text{CacheHit}} = 0\%$

$Mem\_address[7:0] = \{ Tag[2:0], Set\_id[2:0], Block\_byte\_offset[1:0] \}$

`assign_array_cols`

Address	2-byte	2-byte
00111100	a[15][1]	a[15][0]
...	...	...
00100000	a[8][1]	a[8][0]
00011100	a[7][1]	a[7][0]
00011000	a[6][1]	a[6][0]
00010100	a[5][1]	a[5][0]
00010000	a[4][1]	a[4][0]
00001100	a[3][1]	a[3][0]
00001000	a[2][1]	a[2][0]
00000100	a[1][1]	a[1][0]
00000000	a[0][1]	a[0][0]

Set	V	Tag 3-bit	Data 4-byte
111		000	a[7][1] a[7][0]
110		000	a[6][1] a[6][0]
101		000	a[5][1] a[5][0]
100		000	a[4][1] a[4][0]
011		000	a[3][1] a[3][0]
010		000	a[2][1] a[2][0]
001		000	a[9][1] a[9][0]
000		000	a[8][1] a[8][0]

```
a[i][0]           a[i][1]
 [0][0] MIS FILL   [0][1] MIS FILL
 [1][0] MIS FILL   [1][1] MIS FILL
 [2][0] MIS FILL   [2][1] MIS FILL
 [3][0] MIS FILL   [3][1] MIS FILL
 [4][0] MIS FILL   [4][1] MIS FILL
 [5][0] MIS FILL   [5][1] MIS FILL
 [6][0] MIS FILL   [6][1] MIS FILL
 [7][0] MIS FILL   [7][1] MIS FILL
 [8][0] MIS FILL   [8][1] MIS Fill
 [9][0] MIS FILL   [9][1] MIS FILL
[10][0] MIS FILL  [10][1] MIS FILL
[11][0] MIS FILL  [11][1] MIS FILL
[12][0] MIS FILL  [12][1] MIS FILL
[13][0] MIS FILL  [13][1] MIS FILL
[14][0] MIS FILL  [14][1] MIS FILL
[15][0] MIS FILL  [15][1] MIS FILL
```

2.5 Performance Analysis

- **Example 9: Program Performance Analysis**
  2. Direct-mapped $Cache(8B, 4S, 1W)$ for 8-bit address space $R_{\text{CacheHit}} = ?$

$Mem\_address[7:0] = \{ Tag[2:0], Set\_id[1:0], Block\_byte\_offset[2:0] \}$

Address	2-byte	2-byte	2-byte	2-byte
00111000	a[15][1]	a[15][0]	a[14][1]	a[14][0]
...	...	...	...	...
00100000	a[9][1]	a[9][0]	a[8][1]	a[8][0]
00011000	a[7][1]	a[7][0]	a[6][1]	a[6][0]
00010000	a[5][1]	a[5][0]	a[4][1]	a[4][0]
00001000	a[3][1]	a[3][0]	a[2][1]	a[2][0]
00000000	a[1][1]	a[1][0]	a[0][1]	a[0][0]

Set	V	Tag 3-bit	Data 8-byte
11			
10			
01			
00			

```
short a[16][2];
int assign_array_rows() {
  int i, j, sum = 0;
  for(i=0; i<16; i++)
    for(j=0; j<2; j++)
      sum += a[i][j];
  return sum;
}
int assign_array_cols() {
  int i, j, sum = 0;
  for(j=0; j<2; j++)
    for(i=0; i<16; i++)
      sum += a[i][j];
  return sum;
}
```

2.5 Performance Analysis

- **Example 9: Program Performance Analysis**
  2. Direct-mapped $Cache(8B, 4S, 1W)$ for 8-bit address space $R_{\text{CacheHit}} = 75\%$

$Mem\_address[7:0] = \{ Tag[2:0], Set\_id[1:0], Block\_byte\_offset[2:0] \}$

`assign_array_rows`

Address	2-byte	2-byte	2-byte	2-byte
00111000	a[15][1]	a[15][0]	a[14][1]	a[14][0]
...	...	...	...	...
00100000	a[9][1]	a[9][0]	a[8][1]	a[8][0]
00011000	a[7][1]	a[7][0]	a[6][1]	a[6][0]
00010000	a[5][1]	a[5][0]	a[4][1]	a[4][0]
00001000	a[3][1]	a[3][0]	a[2][1]	a[2][0]
00000000	a[1][1]	a[1][0]	a[0][1]	a[0][0]

Set	V	Tag 3-bit	Data 8-byte
11		000	a[7][1] a[7][0] a[6][1] a[6][0]
10		000	a[5][1] a[5][0] a[4][1] a[4][0]
01		000	a[3][1] a[3][0] a[2][1] a[2][0]
00		000	a[9][1] a[9][0] a[8][1] a[8][0]

```
a[i][0]           a[i][1]
 [0][0] MIS FILL   [0][1] HIT
 [1][0] HIT        [1][1] HIT
 [2][0] MIS FILL   [2][1] HIT
 [3][0] HIT        [3][1] HIT
 [4][0] MIS FILL   [4][1] HIT
 [5][0] HIT        [5][1] HIT
 [6][0] MIS FILL   [6][1] HIT
 [7][0] HIT        [7][1] HIT
 [8][0] MIS FILL   [8][1] HIT
 [9][0] HIT        [9][1] HIT
[10][0] MIS FILL  [10][1] HIT
[11][0] HIT       [11][1] HIT
[12][0] MIS FILL  [12][1] HIT
[13][0] HIT       [13][1] HIT
[14][0] MIS FILL  [14][1] HIT
[15][0] HIT       [15][1] HIT
```

2.5 Performance Analysis

- **Example 9: Program Performance Analysis**
  2. Direct-mapped $Cache(8B, 4S, 1W)$ for 8-bit address space $R_{\text{CacheHit}} = 50\%$

$Mem\_address[7:0] = \{ Tag[2:0], Set\_id[1:0], Block\_byte\_offset[2:0] \}$

`assign_array_cols`

Address	2-byte	2-byte	2-byte	2-byte
00111000	a[15][1]	a[15][0]	a[14][1]	a[14][0]
...	...	...	...	...
00100000	a[9][1]	a[9][0]	a[8][1]	a[8][0]
00011000	a[7][1]	a[7][0]	a[6][1]	a[6][0]
00010000	a[5][1]	a[5][0]	a[4][1]	a[4][0]
00001000	a[3][1]	a[3][0]	a[2][1]	a[2][0]
00000000	a[1][1]	a[1][0]	a[0][1]	a[0][0]

Set	V	Tag 3-bit	Data 8-byte
11		000	a[7][1] a[7][0] a[6][1] a[6][0]
10		000	a[5][1] a[5][0] a[4][1] a[4][0]
01		000	a[3][1] a[3][0] a[2][1] a[2][0]
00		000	a[9][1] a[9][0] a[8][1] a[8][0]

```
a[i][0]           a[i][1]
 [0][0] MIS FILL   [0][1] MIS FILL
 [1][0] HIT        [1][1] HIT
 [2][0] MIS FILL   [2][1] MIS FILL
 [3][0] HIT        [3][1] HIT
 [4][0] MIS FILL   [4][1] MIS FILL
 [5][0] HIT        [5][1] HIT
 [6][0] MIS FILL   [6][1] MIS FILL
 [7][0] HIT        [7][1] HIT
 [8][0] MIS FILL   [8][1] MIS FILL
 [9][0] HIT        [9][1] HIT
[10][0] MIS FILL  [10][1] MIS FILL
[11][0] HIT       [11][1] HIT
[12][0] MIS FILL  [12][1] MIS FILL
[13][0] HIT       [13][1] HIT
[14][0] MIS FILL  [14][1] MIS FILL
[15][0] HIT       [15][1] HIT
```

2.5 Performance Analysis

- **Hit rate of cache**

$$R_{\text{CacheHit}} = \frac{N_{\text{CacheHit}}}{N_{\text{Access}}}$$

- **Miss rate of cache**

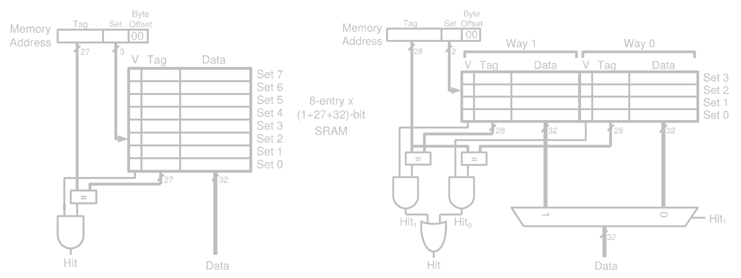
$$R_{\text{CacheMiss}} = \frac{N_{\text{CacheMiss}}}{N_{\text{Access}}} = 1 - R_{\text{CacheHit}}$$

- **Average access time of memory**
  - T_{Cache} : Access times of the cache
  - $T_{\text{MainMemory}}$: Access times of the main memory

$$T_{\text{Access}} = T_{\text{Cache}}\left(1 - R_{\text{CacheMiss}}\right) + R_{\text{CacheMiss}}\left(T_{\text{MainMemory}} + T_{\text{Cache}}\right)$$
$$= T_{\text{Cache}} + R_{\text{CacheMiss}}\, T_{\text{MainMemory}}$$

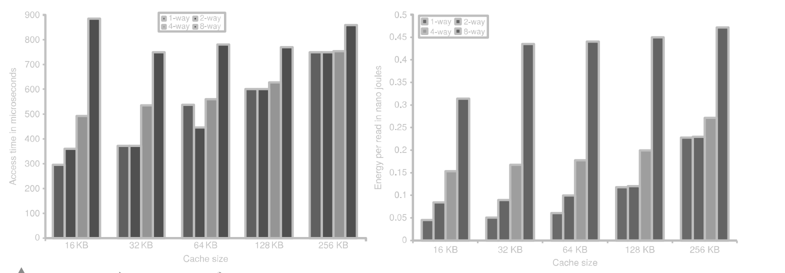
2.5 Performance Analysis

- **Multi-level Cache (for Performance Optimization)**
  - Larger cache size → Higher hit-rate
  - More ways → Higher hit-rate
  - But more cache access time and energy are consumed



2.5 Performance Analysis

- **Multi-level Cache (for Performance Optimization)**
  - Larger cache size → Higher hit-rate
  - More ways → Higher hit-rate
  - But more cache access time and energy are consumed



2.5 Performance Analysis

- **Multi-level Cache (for Performance Optimization)**
  - Multi-level Caches, small and simple 1st-level caches
    - Reducing the cache access time and energy

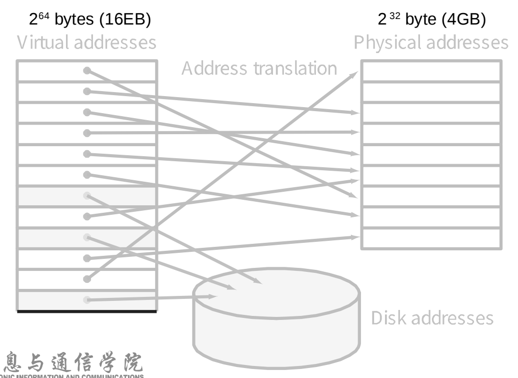
$$T_{\text{Access}} = T_{\text{Cache}} + R_{\text{CacheMiss}} T_{\text{MainMemory}}$$

$$T_{\text{Cache}} = T_{\text{L1}} + R_{\text{L1Miss}} \left(T_{\text{L2}} + R_{\text{L2Miss}} T_{\text{L3}} \right)$$

Year	CPU	MHz	L1 Cache	L2 Cache
1985	R2000	16.7	none	none
1990	R3000	33	32 KB direct mapped	none
1991	R4000	100	8 KB direct mapped	1 MB direct mapped
1995	R10000	250	32 KB two-way	4 MB two-way
2001	R14000	600	32 KB two-way	16 MB two-way
2004	R16000A	800	64 KB two-way	16 MB two-way

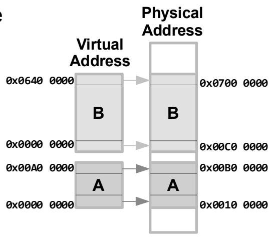
3 Virtual Memory

- Enables main memory to be used as a *Cache* for disks
- Enables memory to be shared by programs
- To alleviate the problem of insufficient main memory



3.1 Segmented Memory Management

- More memory addressing space
  - Introduced on the Intel 8086 (16-bit) in 1978
    - As a way to address more than 64 KB of memory
- Each program has its own addressing space
  - Introduced a 2 version on 80286 in 1982
    - Support virtual memory & memory protection
    - Original version was called ***Real*** mode
    - New version was called ***Protected*** mode
  - Introduced 64-bit mode in 2003
    - Segmentation is generally disabled
    - Flat ***linear address*** is generally enabled



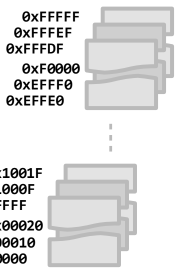
3.1 Segmented Memory Management

- **Real-Mode (Compatible with 8086)**
  - ***Seg_base_address*** assigned by OS to program is stored in ***Seg_register***
  - ***Seg_register*[15:0]**
    - Segment ***Offset*** is used for addressing instruction in a program
    - 2^{16} bytes (64 KB) per segment
  - ***Phy_address*[19:0] = (*Seg_register*[15:0] << 4)+ *Offset*[15:0]**
    - Addressing space is 2^{20} bytes (1 MB)
    - Segment spaces may overlap in physical space

```
  0000 0110 1110 1111 0000 Seg_register << 4
+      0001 0010 0011 0100 Offset
  ----------------------------------------
  0000 1000 0001 0010 0100 Phy_address(20-bit)
```

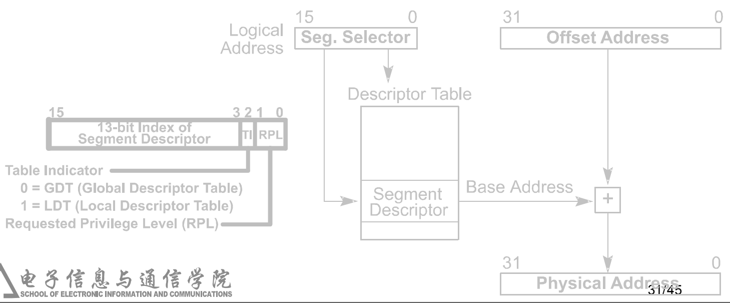
Physical address

$0000{:}0020_H \rightarrow$
$0001{:}0010_H \rightarrow 00020_H$
$0002{:}0000_H \rightarrow$



3.1 Segmented Memory Management

- **Protected-Model (80386)**
  - *Seg_base_address* is in segment descriptor stored in memory
  - *Seg_register*[15:0] = *Seg_selector*
    - ➢Index of Segment Descriptor
  - *Logic_address* = { *Seg_selector*[15:0], *Offset*[31:0] }
  - *Phy_address*[31:0] = *Seg_base_address*[31:0] + *Offset*[31:0]



3.1 Segmented Memory Management

- **Segment Descriptor (64-bit)**
  - Maximum Size of Segment: *Seg_limit* << (12 × *G*)

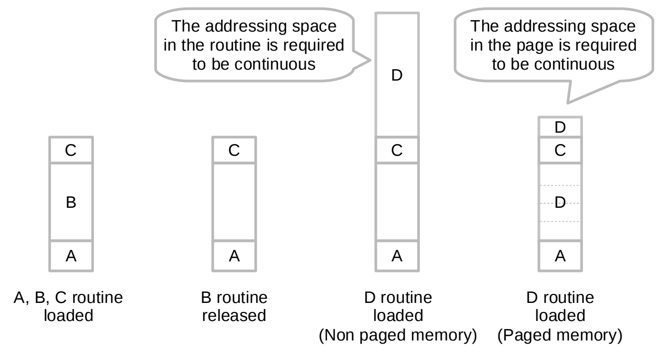
Byte	80x86				
7	Seg_base_address[31:24]				
6	G	D	L	V	Seg_limit[19:16]
5	Access_rights[7:0]				
4	Seg_base_address[23:0]				
3					
2					
1	Seg_limit[15:0]				
0					

G: Granularity
D: 1/0, 32/16 bit segment
L: 64-bit code segment (IA-32e only)
V: Available for use by system software

```
Example: The 80386 segment descriptor 0x34_D3_00_23_12_89_01_03
segment start address: 0x34_23_12_89;
segment size limit:    0x3_01_03 << (12 × G) = 0x30_10_30_00;
segment end address:   0x34_23_12_89 + 0x30_10_30_00 - 1
                     = 0x64_33_42_88
```

3.2 Paged Virtual Memory

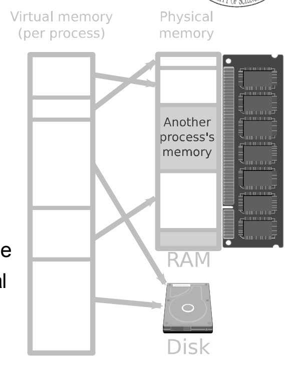
- **Purpose of Memory Paging**
  - Due to memory fragmentation, memory is allocated to the program in smaller granularity units



The addressing space in the routine is required to be continuous
The addressing space in the page is required to be continuous
C
B
A
A, B, C routine loaded
C
A
B routine released
D
C
A
D routine loaded (Non paged memory)
D
C
D
A
D routine loaded (Paged memory)

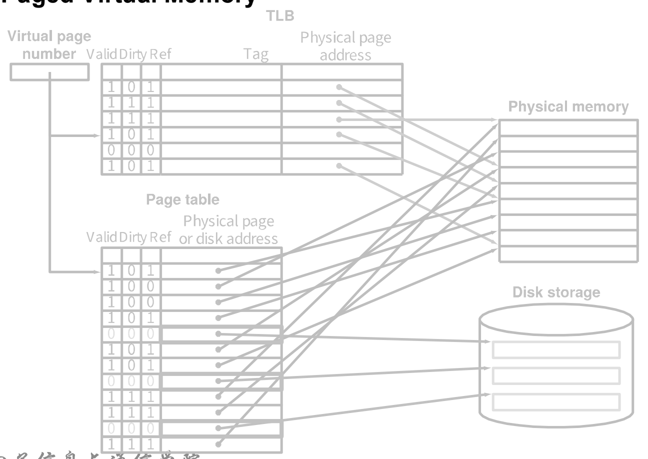
3.2 Paged Virtual Memory

- **Paged Virtual Memory**
  - Virtual Addresses
    - ➢Storage resources are mapped to a unified address space
  - Page Table
    - ➢Translate the virtual addresses into physical addresses
  - TLB (Translation-Lookaside Buffer)
    - ➢Address-translation Cache / Fast Table
    - ➢Stored the recent translations of virtual memory to physical memory
    - ➢to reduce the time taken to access a user memory location



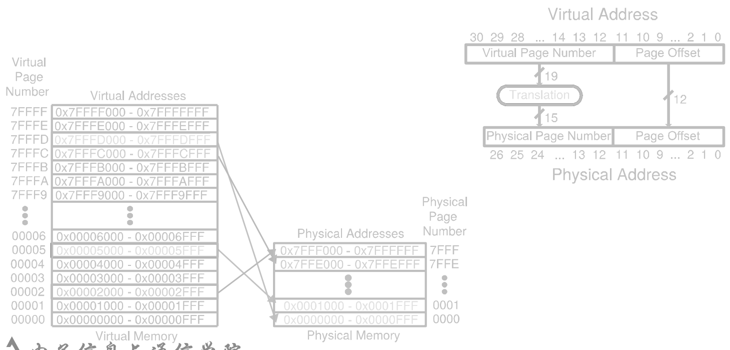
3.2 Paged Virtual Memory

- **Paged Virtual Memory**



3.2.1 Address Translation

- **Translation:** *Virtual_address* → *Phy_address*
  - *Virtual_address* = { *Page_id*, *Offset* }
  - *Phy_address* = { *Phy_page_id*, *Offset* }



Virtual Address
30 29 28 ... 14 13 12 11 10 9 ... 2 1 0
Virtual Page Number
Page Offset
19
Translation
15
12
Physical Page Number
Page Offset
26 25 24 ... 13 12 11 10 9 ... 2 1 0
Physical Address
Virtual Page Number
Virtual Addresses
7FFFF 0x7FFFF000 - 0x7FFFFFFF
7FFFE 0x7FFFE000 - 0x7FFFEFFF
7FFFD 0x7FFFD000 - 0x7FFFDFFF
7FFFC 0x7FFFC000 - 0x7FFFCFFF
7FFFB 0x7FFFB000 - 0x7FFFBFFF
7FFFA 0x7FFFA000 - 0x7FFFAFFF
7FFF9 0x7FFF9000 - 0x7FFF9FFF
00006 0x00006000 - 0x00006FFF
00005 0x00005000 - 0x00005FFF
00004 0x00004000 - 0x00004FFF
00003 0x00003000 - 0x00003FFF
00002 0x00002000 - 0x00002FFF
00001 0x00001000 - 0x00001FFF
00000 0x00000000 - 0x00000FFF
Virtual Memory
Physical Page Number
Physical Addresses
0x7FFF000 - 0x7FFFFFF 7FFF
0x7FFE000 - 0x7FFEFFF 7FFE
0x0001000 - 0x0001FFF 0001
0x0000000 - 0x0000FFF 0000
Physical Memory

3.2.1 Address Translation

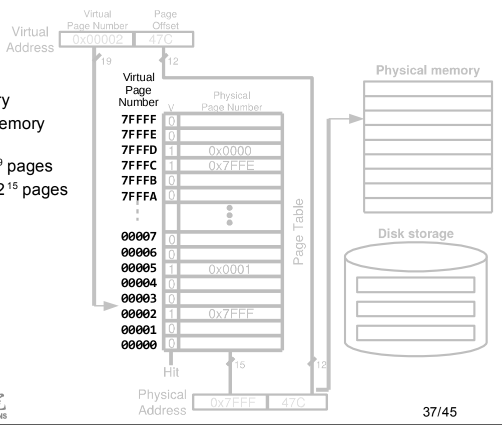
- **Page Table (for Address Translation)**

Virtual_address[30:0] = { *Vir_page_id*[18:0], *Offset*[11:0] }
Phy_address[26:0] = { *Phy_page_id*[14:0], *Offset*[11:0] }

Example:
Page_size: 2^{12} bytes (4KB)

2^{31} bytes (2GB) virtual memory
2^{27} bytes (128MB) physical memory

Virtual memory is split into 2^{19} pages
Physical memory is split into 2^{15} pages



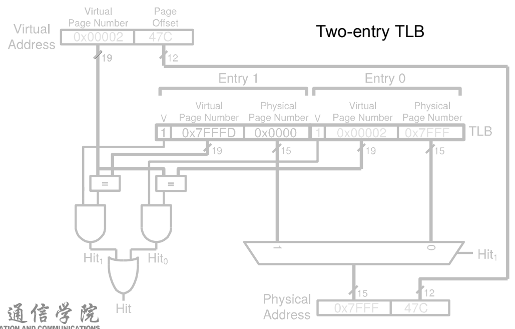
3.2.1 Address Translation

- **Page Fault**
  - Exception if virtual address is not mapped to physical memory
  - OS loads the page from the disk into physical memory
  - Physical memory is used as the cache for the disk

Cache	Virtual Memory
Block	Page
Block size	Page size
Block offset	Page offset
Miss	Page fault
Tag	Virtual page number

3.2.2 Translation-Lookaside Buffer

- **TLB (Fast Table, for caching Page Table)**
  - TLB on chip can hold several recently used page table lines
  - TLB is organized as a fully associative cache
  - Using ***Page_id*** to access the TLB
  - TLB Line: { ***V, Page_id, Phy_page_id*** }

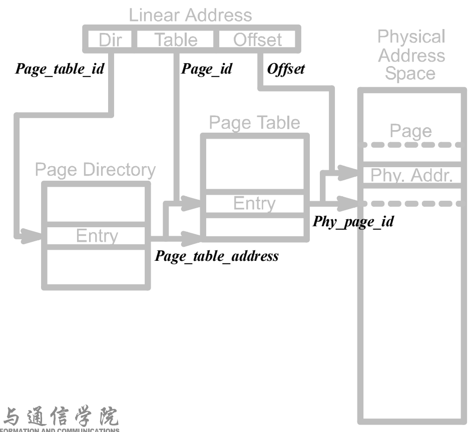


3.3 Paged Memory Management in intel 80386

- **Paging**
  - Introduced on the Intel 80386 (Option) to support virtual memory
  - ***Linear_address*** (Virtual address) → ***Phy_address***
  - ***Phy_address* = { *Phy_page_id*, *Offset* }**
  - ***Linear_address* = { *Dir*, *Table*, *Offset* }**
    - ➢***Dir*** field: Virtual ***Page_table_id***
    - ➢***Table*** field: Virtual ***Page_id***
    - ➢***Offset*** field: Address ***Offset***
- **Page Directory**
  - The set of ***Page_table_address*** (Indexed by ***Page_table_id***)
- **Page table**
  - The set of ***Phy_page_id*** (Indexed by virtual ***Page_id***)

3.3 Paged Memory Management in intel 80386

- **Paging in Intel 80386**
  - ***Linear_address*[31:0] = { *Dir*[9:0], *Table*[9:0], *Offset*[11:0] }**



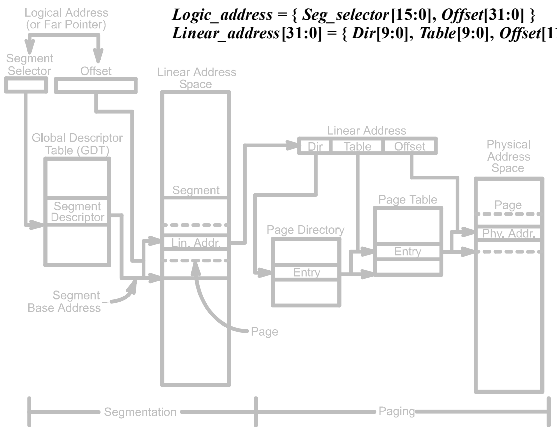
3.4 Segmentation and Paging

- **Segmentation**
  - ***Logic_address* → *Linear_address***
- **Paging**
  - ***Linear_address* → *Phy_address***
- **e.g. Intel x86**
  - ***Logic_address* = { *Seg_selector*[15:0], *Offset*[31:0] }**
  - ***Linear_address*[31:0] = { *Dir*[9:0], *Table*[9:0], *Offset*[11:0] }**

3.4 Segmentation and Paging

- **Segmentation and Paging in Intel x86**

Logic_address = { *Seg_selector*[15:0], *Offset*[31:0] }
Linear_address[31:0] = { *Dir*[9:0], *Table*[9:0], *Offset*[11:0] }



Homework

- P164
  - 6
  - 7
  - 8
  - 9
  - 10
  - 11