計算機組織

Computer Architecture

TZU-CHUN HSU[1]

[1]vm3y3rmp40719@gmail.com

[1]Department of Computer Science, Zhejiang University

2020 年 11 月 21 日

Version 1.0.1

Disclaimer

本文「計算機組織」為台灣研究所考試入學的「計算機組織」考科使用，內容主要參考張凡先生的二本計算機組織參考書 [1][2]，以及 wjungle 網友在 PTT 論壇上提供的資料結構筆記 [3]。

1 Overview

1. 本文頁碼標記依照實體書 [1][2] 的頁碼。

2. TKB 筆記 [3] 章節頁碼：

Chapter	Page No.
1	1
2	27
3	81
4	101
5	119
6	165
7	221
8	238

3. 常考：（參考 TKB 筆記 [3] 中頁碼）

   (a) 19

   (b) 59

   (c) 84

   (d) 92

   (e) 141

   (f) 170

   (g) 200

4. 省略第一章重點十四，第二章重點五、六、十，第六章重點十四，第七章只看重點一到四及十，第八章重點五一致性協定範例。

2 Summary

1. **Theorem (10)** Endianness：
   - Big Endian：最左邊或 MSB 放在最低 address，e.g. MIPS。
   - Little Endian：最右邊或 LSB 放在最低 address，e.g. x86。
2. **Theorem (15)** 轉譯（Translation）：
   - 流程：compiler, assembler, linker, loader。
   - Linker：連結多個模組解決位址的參考問題。
   - Loader：將機械碼放在適當 address。
3. **Theorem (29)** `lb` 和 `lbu` 皆取 LSB，只差在 `lb` 要擴展； `sb` 存入 LSB。
4. **Theorem (39)** `swap $s0, $s1`：

```
xor $s0, $s0, $s1
xor $s1, $s0, $s1
xor $s0, $s0, $s1
```

5. **Theorem (42))** 常數邏輯類運算用 0 擴充，e.g. andi；常數算術類運算用 sign 擴充，e.g. addi。
6. **Theorem (47)**
   - `srl rd, rt, shamt # rs = 5'0`
   - `lw/sw rt, imm(rs)`
   - `beq/bne rs, rt, addr`
   - `addi rt, rs, imm`
7. **Theorem (49, 51)**
   - `branch`： `imm` 填入與 $PC+4$ 距離除以 4，range： $PC-2^{17} \sim PC+2^{17}-4$。
   - `jump`：`addr` 填入地址第 $[27-2]$，及去除最左 4 位和最右 2 位，實際地址：$PC[31-28]+addr+2'0$，只能在同一個 block，即看最左 4 位。
8. **Theorem (62)**

```
int fact (int n) {
    if (n < 1)
        return 1;
    else
        return (n * fact(n - 1));
}
```

```
fact:
    addi $sp, $sp, -8
    sw $ra, 4($sp)
    sw $a0, 0($sp)
    slti $t0, $a0, 1
    beq $t0, $zero, L1
    addi $v0, $zero, 1
    addi $sp, $sp, 8
    jr $ra
L1:
    addi $a0, $a0, -1
    jal fact
    lw $a0, 0($sp)
    lw $ra, 4($sp)
    addi $sp, $sp, 8
    mul $v0, $a0, $v0
    jr $ra
```

9. **Theorem (137)** 1's 補數，在 overflow 時，需額外加 1 修正。

10. **Theorem (141)** `sltu` 可以用來判斷負數，因為會大於正數。

11. **Theorem (144)** `add, addi, sub` overflow 時會 exception，但 `addu, addiu, subu` 則不會，因為較多用於計算 address。

12. **Theorem (165, 167)** 無號數乘法：

    - 傳統乘法：Multiplier 右移，multiplicand 初始放低位、左移且與 product 皆兩倍長。

- Hardware-friendly multiplication：Multiplicand 一倍長，product 兩倍長、初始放低位、加到高位且右移，捨去 mulitplier。

13. **Theorem (171, 173)** 有號數乘法：

- Booth's algorithm：
  - 多一位 mythical bit。
  - Multiplicand 一倍長，product 兩倍長、初始為 multiplier 放低位且操作在高位、右移，無 multiplier。
  - 01: $+$, 10: $-$, 00, 11: skip.
- 快速乘法硬體：
  - 沒有 clock，屬於 combinational unit，不消耗 1 個 clock cycle。
  - 容易 optimize 和 pipeline。

14. **Theorem (177, 180)** 除法：

- 傳統除法：quotient 一倍長且左移，divisor 兩倍長、初始放高位且右移，remainder 兩倍長且初始為 dividend 放低位。
- Hardware-friendly division：divisor 一倍長，remainder 兩倍長、初始為 divident 放低位、初始時左移 1 位、最後高位右移 1 位且左移，無 quotient。

15. **Theorem (183)**

- MIPS 乘法：

```
mult $s2, $s3
mfhi $s0
mflo $s1
```

- MIPS 除法：

```
# Lo = $s2 / $s3
# Hi = $s2 % $s3
div $s2, $s3
```

16. **Theorem (190)** 浮點數：

-
$$
\begin{aligned}
(-1)^{sign} \times (0.fraction) \times 2^{(exponent-bias)} \\
bias = 2^{n-1} - 1
\end{aligned}
\tag{1}
$$

- Denormalized number：

$$(-1)^{sign} \times (0.fraction) \times 2^{-126} \tag{2}$$

	Sign	Exponent	Fraction	Underflow	Overflow
Single precision	1	8	23	$> 0, < 2 \times 10^{-38}$	$> 2 \times 10^{38}$
Double precision		11	52	$> 0, < 2 \times 10^{-308}$	$> 2 \times 10^{308}$

Single precision		Double precision		Representation
Exponent	Fraction	Exponent	Fraction	
0	0	0	0	± 0
0	$\neq 0$	0	$\neq 0$	$\pm$ denormalized number
$1 \sim 254$	$\times$	$1 \sim 2046$	$\times$	$\pm$ floating-point number
255	0	2047	0	$\pm\infty$
255	$\neq 0$	2047	$\neq 0$	NaN

17. **Theorem (205)** 四捨五入：

LSB	Guard	Round	Sticky	Action
0	1	0	0	↓
0	1	0	1	↑
1	1	0	0	↑
1	1	0	1	↑

18. **Theorem (213)** 右移代替除法：若為負數需先加上 $2^n - 1$ 再位移，其中 n 為要右移的位數。

19. **Theorem (284)** 軟硬體影響性能：

	IC	CPI	Clock rate
Algorithm	√	√	
Programming language	√	√	
Compiler	√	√	
ISA	√	√	√
Computer organization		√	√
VLSI			√

20. **Theorem (284)**

	IC	CPI	Clock rate
RISC	大	小	小
CISC	小	大	大

21. **Theorem (292)** MIPS：

- $$MIPS = \frac{\text{IC}}{\text{execution time} \times 10^6} = \frac{\text{clock rate}}{\text{CPI} \times 10^6} \tag{3}$$
- MIPS 不考慮指令的能力。
- 在同一台電腦上的程式，MIPS 也可能不同。
- MIPS 可能與效能成反比。

22. **Theorem (306)**

- $$SPECratio = \frac{\text{execution time referenced}}{\text{execution time measured}} \tag{4}$$

  SPECratio 越大代表效能越好。
- 若 A 以 B 為基準取 norm，值為 $\frac{B}{A}$。
- 先取 geometric mean 再 norm 和先 norm 再 geometric mean 結果相同。
- Geometric mean 和 execution time、norm 的基準皆無關。
- Geometric mean 無法預測 execution time。

23. **Theorem (365)** `beq`：$PC := (PC + 4) + (sign - ext(imm) << 2)$

24. **Theorem (374)** $ALUOp$：

- 00：Add (`lw` , `sw`).
- 01：Subtract (`beq`).
- 10：Other R-type.

25. **Theorem (371)** 只有 `jump` 和 `MemtoReg` 上面 1 下面 0，其他皆相反。

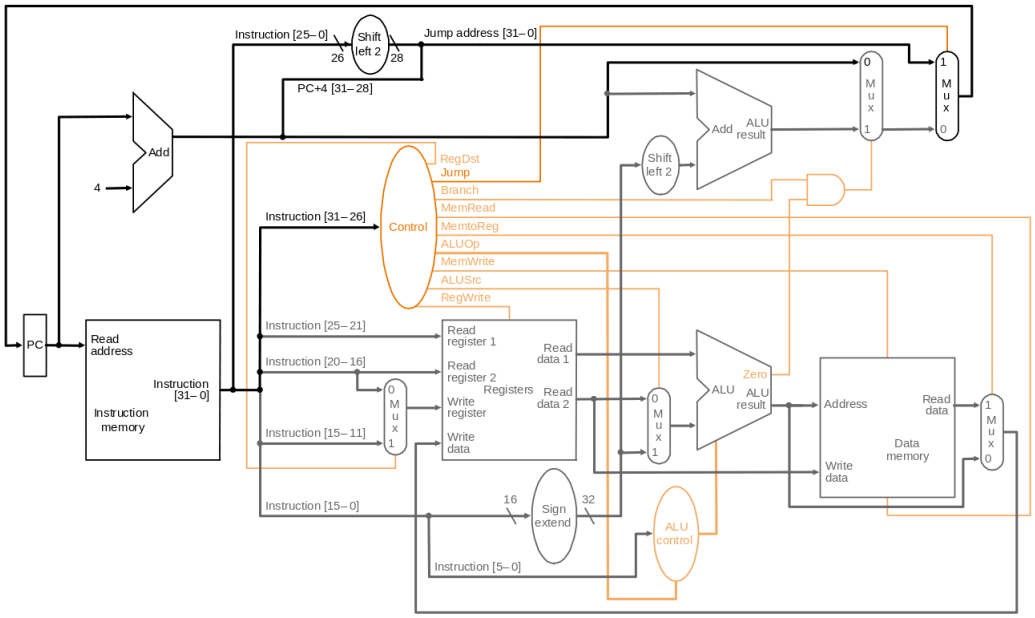

Figure 1: Single-cycle CPU with jump and branch.

26. **Theorem (429)** Pipeline 不會對單一工作的 latency 有幫助，但會提升整體工作的 throughput。

27. **Theorem (441)** 原始 pipeline 設計：

    - `beq` 在 MEM 決定是否要跳。
    - `RegDst` 在 EX。

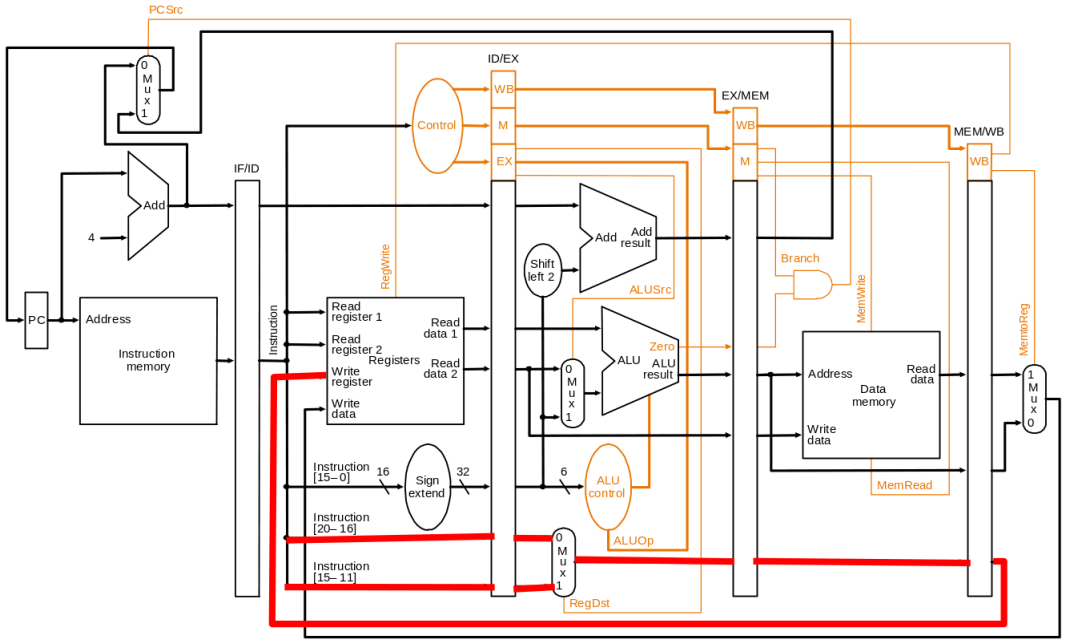

Figure 2: Original pipeline.

Stage	IF	ID	EX	MEM	WB
Signal	×	×	RegDst ALUSrc ALUOp	MemRead MemWrite branch	RegWrite MemtoReg

28. **Theorem ()** NOP 無法解決 structual hazard，因為 NOP 本質上也是指令。

29. **Theorem (450, 455, 457, 458)** Data hazards：

- Insert NOP： `slt $0, $0, 0`，浪費 2 CC。
- Forwarding：Combinational units，放在 EX 因為 ALU。

if (EX/MEM.RegWrite $\wedge$ (EX/MEM.Rd $\neq 0$) $\wedge$ (EX/MEM.Rd = ID/
ForwardA/B = 10

Listing 1: EX hazard.

if (MEM/WB.RegWrite $\wedge$ (MEM/WB.Rd $\neq 0$) $\wedge$ ($\neg$ EX_hazard) $\wedge$
ForwardA/B = 01

Listing 2: MEM hazard.

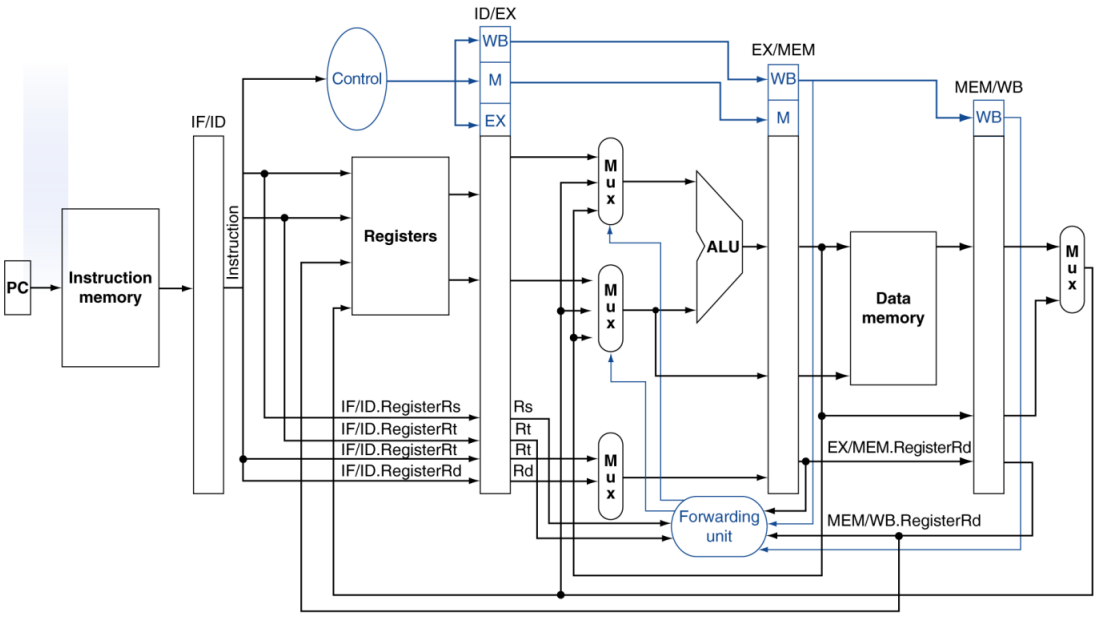

Figure 3: Pipeline with forwarding.

- Stall:

```
if (ID/EX.MemRead ∧ (ID/EX.Rt = IF/ID.Rs/Rt))
    IF/ID.Write := 0
    PC.Write := 0
```

Listing 3: Stall.

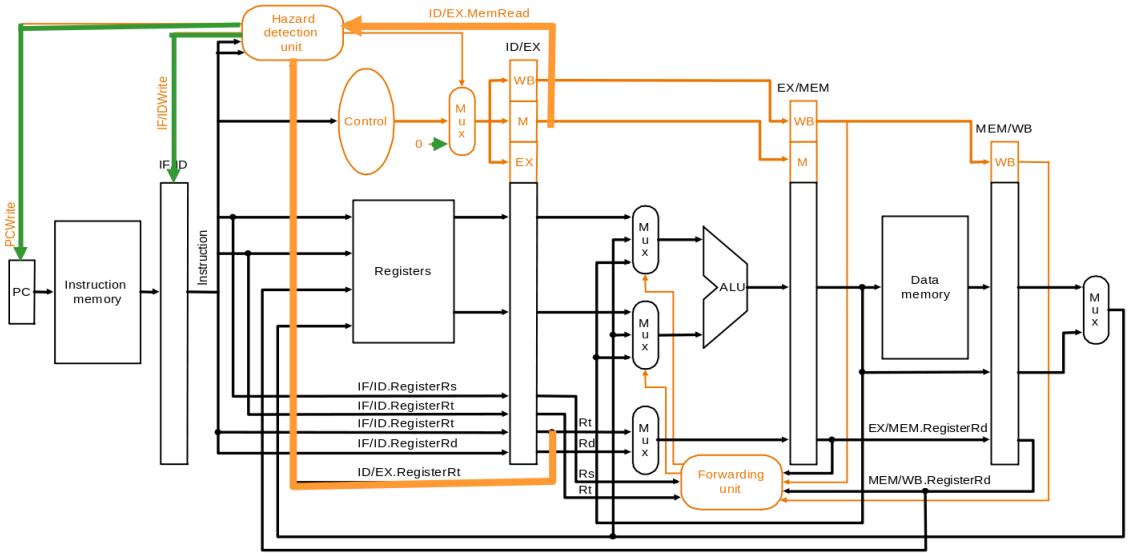

Figure 4: Pipeline with hazard detection and forwarding units.

30. **Theorem (473)** RAW: True data dependency.

31. **Theorem (478, 487, 494, 559)** Control hazards：

    - 若分支指令與**前一個 ALU 指令**或**前面第二個** `lw` 有 data dependency，必須 stall 1 CC。
    - 若分支指令與**前一個** `lw` 有 data dependency，必須 stall 2 CC。
    - 分支指令通過 XOR 再 NOR 比較是否相同。

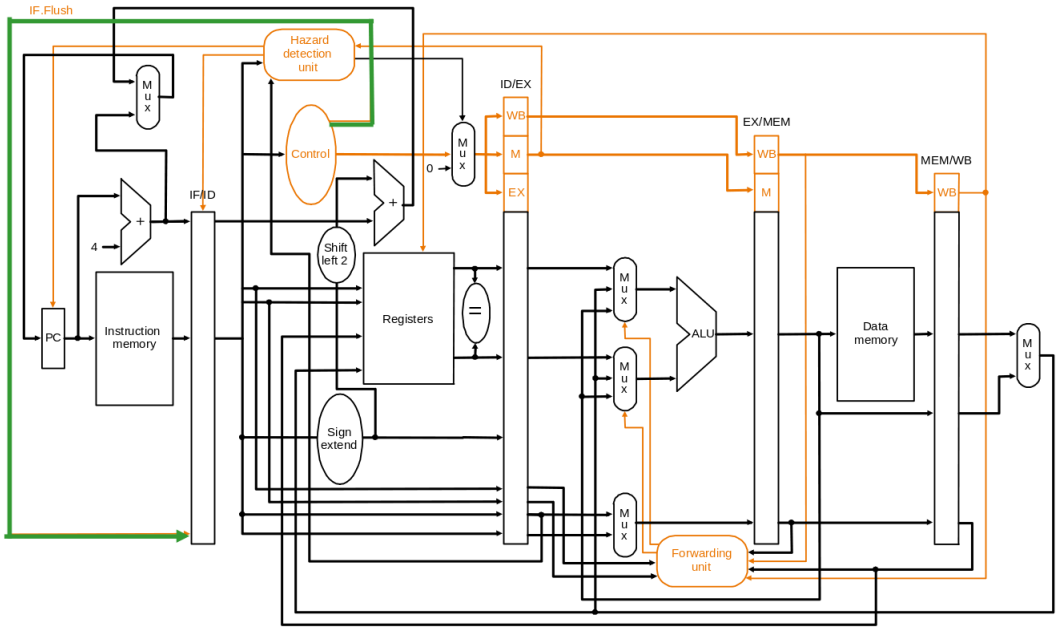

Figure 5: Pipeline with hazard detection, forwarding units and flush.

- Branch Target Buffer (BTB) check the branch in IF.
- Delayed branch：
  - NOT suitable for deep pipeline.
  - From before：最佳方法，不管跳或不跳皆提升。
  - From target：用於 branch 發生機率高。
  - From fall through：用於 branch 發生機率低。

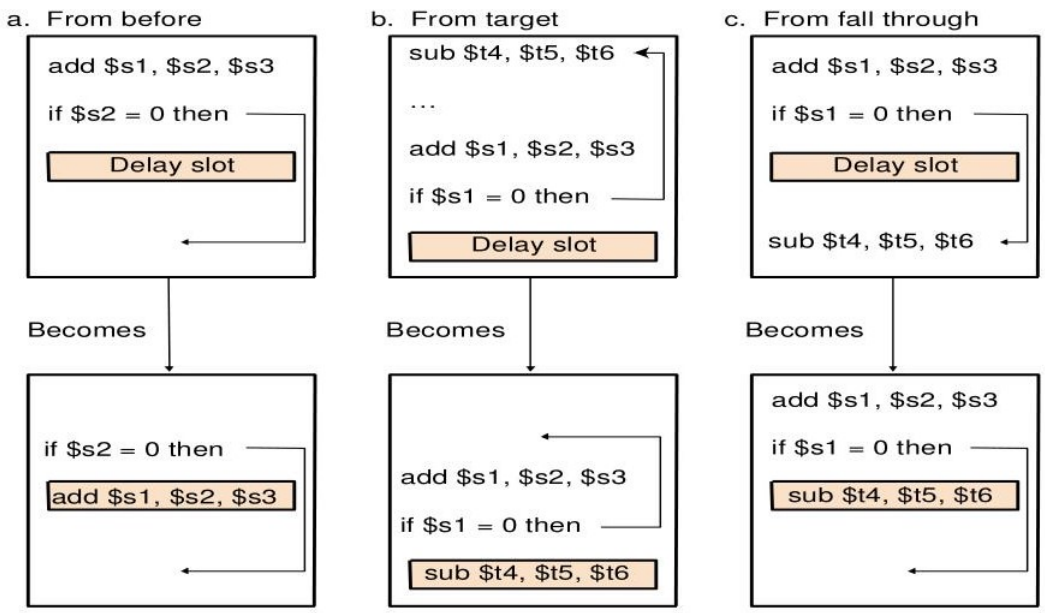

Figure 6: Example of delayed branch.

32. **Theorem (498)** Advanced pipeline：

- Instruction-Level Parallelism (ILP)：
  - Increase pipeline depth：superpipeline，但 hazard 也上升。
  - Multiple issue：大量複製 pipeline 單元，讓每個 stage 都可以執行多個指令。
    - 可分為 static 和 dynamic，前者在編譯時決策，MIPS64 採用，後者在執行時決策，又稱 super scaler。
    - Static：軟體實現，Code scheduling, loop unrolling, Very Long Instruction Word (VLIW).
    - Dynamic：硬體實現，pipeline 被分為 in-order issue units，out-of-order execute units 和 in-order commit units。
- Speculation：猜測指令的性質，以便執行其他相關指令。軟硬體共同實現。

33. **Theorem (515)** Exception：

- Exception：內部；Interrupt：外部。
- Exception Program Counter (EPC)：存放引發 exception 指令地址；Status/Cause register：存放 exception 原因。
- Imprecise interrupt：不知道發生的 stage，只能 kill program。

34. **Theorem (7)** DRAM 是 capacitor，比較便宜且體積小，但速度較慢；SRAM 是 latch。Cache 用 SRAM。

35. **Theorem (25, 27, 38, 48, 124)** Cache：

    - Write：
      - Write-through：寫入 cache 和 memory。
      - Write-buffer：寫入 cache 和 buffer 且 CPU 繼續執行，當 buffer 滿時，CPU 須等到 buffer 有空位。
      - Write-back：只寫入 cache。
      - Write allocate：從 memory 拉進 cache，在 cache 中修改。
      - Write around (No write allocate)：不拉進 cache，只在 memory 中修改。
      - 通常 write-though 配 write around，write-back 配 write allocate。
    - Split cache 通常有較差 hit ratio，提升 bandwidth，但不提升 speed。
    - 增加 associativity：降低 miss rate，增加 hit time。
    - L1 cache：注重減少 hit time；L2 cache：注重減少 miss ratio。
    - Cache miss is handled by cache controller.

36. **Theorem (53)**

$$L_nGMR = \prod_{i=1}^{n} L_iLMR \tag{5}$$

37. **Theorem (54)** Average Memory Access Time (AMAT)：

$$AMAT = hit\ time + Miss\ rate \times Miss\ penalty \tag{6}$$

    Using multilevel cache and non-blocking cache both reduce miss penalty.

38. **Theorem (61, 66, 71, 78, 86, 88, 124)** Virtual memory：

    - fully mapping 因為怕存取 HD 太慢，write-back，large page size。
    - 減少 page table size：
      - 使用 limit register 限制分頁大小。
      - 兩張 page tables 和兩個分開的 limit registers。
      - Inverted page table：儲存 virtual page no. 和 physical page no.，並使用 hash function，因此 virtual page table size 和 physical page table 一樣即可。

  - Multilevel page table：但 address translation 太複雜。
  - 允許 page table 也被分頁，但必須避免無止盡的 page fault。
- reference bit：決定 victim page。
- TLB：
  - 存 tag 和 physical page no.。
  - TLB 一般使用 fully associative 的 TLB，lower miss rate，且 size 較小，lower cost，且一般使用 random 置換。
- Page fault is signaled by hardware.
- TLB miss result in TLB exception.
-
  - Physically indexed and physically tagged：將所有 virtual address translate physical address 再 access。
  - Virtually indexed and virtually tagged：使用 virtual address access cache，only translate if going to memory，但若有一 page 被 share，在 cache 有多個 objects，造成 aliasing。
  - Virtually indexed and physically tagged：Always translate before going to cache. 解決 virtual addressed cache aliasing 問題。

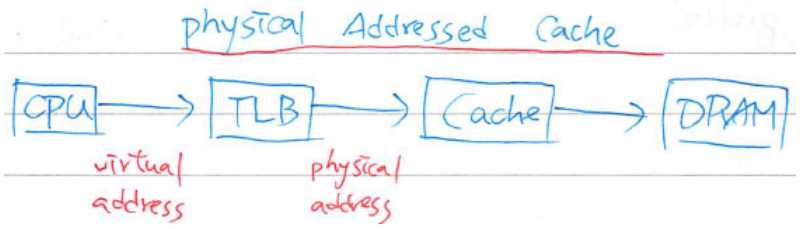

Figure 7: Physical adressed cache.

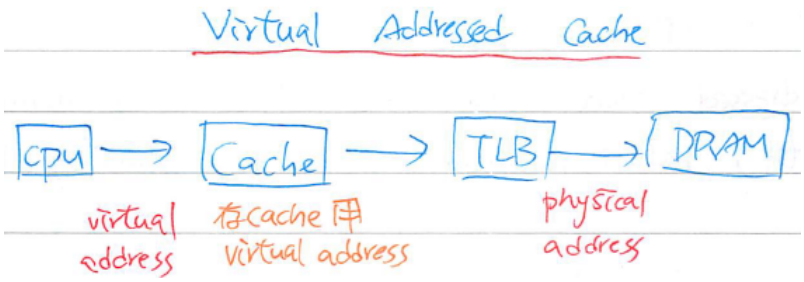

Figure 8: Virtual addressed cache.

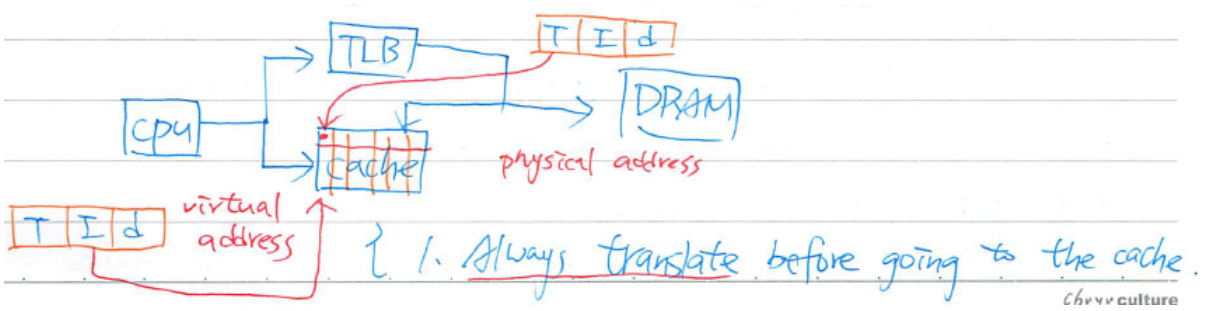

Figure 9: Virtually indexed and physically tagged cache.

- Non-blocking cache：Cache miss 時，CPU 繼續 access cache，通常用在 superscalar 的 out of order execution，用來隱藏 cache miss time。

39. **Theorem (203)**

$$\text{Rotational time} = \frac{0.5}{\text{RPS}} \tag{7}$$

40. **Theorem (207)** Flash：

- NOT lost info if it loses power.
- Write is much slower than read.
- NOR flash：Read access time computation is similar to DRAM.
- NAND flash：Read bandwidth computation is similar to disk. Much cheaper per GB.

41. **Theorem (210, 213, 278, 289)** RAID：

- Dependability：服務品質。
- Reliabilitiy：時間區間內可以正確執行其功能的機率。
- Availability：某時間點可以正確執行其功能的機率。

$$\text{Availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} \tag{8}$$

- Data stripping：Data 分散到不同 disks，一次 data 存取會使多個 disks 存取，提升 performance，但不能改善 reliability。
- Redundancy：利用多餘 disks 提升 availablity，但不能改善 reliability。
- RAID 0：Block-stripping，非容錯，沒有多餘 disks。
- RAID 1：Mirroring (shadowing)，最貴，data 總是有額外一個 copy，$2n$ disks。

- RAID 2：Hamming code，Write 需要讀取所有 disks，從新計算 Hamming code 並寫入 ECC disks，效率差，$2n-1$ disks。
- RAID 3：
  - Bit-interleaved parity.
  - Reliability 和 RAID 2 相同。
  - 不做備份，花費較多時間恢復 data，$n+1$ disks。
  - 當 1 個 disk 出錯可救回來，多個則否。
  - Availability cost 為 $\frac{1}{N}$，其中 N 為 protection group disks 數量。
  - Parity 集中存放一個 disk。
- RAID 4：
  - Block-interleaved parity。
  - 只對 protection group 其中一 disk 做 small reads。
  - $n+1$ disks，parity 集中存放一個 disk。
  - 當 1 個 disk 出錯可救回來，多個則否。
- RAID 5：
  - Distributed Block-interleaved parity
  - Write 就不會有單一 disk 瓶頸
  - $n+1$ disks，parity 被分散到所有 disks。
- RAID 0 與 RAID 1 組合：
  - RAID 0+1：先 stripping，再整體 mirror。一個 disk 壞，需整組 copy 替換。
  - RAID 1+0：先個別 mirror，再交錯 stripping。一個 disk 壞，只需該 disk 的 copy 替換，較好。
- Read latency：RAID 3 最短；Write latency：RAID 0 或 RAID 3 最短。
- RAID 3 has worst throughput for small writes.
- RAID 3, 4, 5 have same throughput for large writes.
- RAID 4, 5 perform same for parallel small reads and small writes, but RAID 3 can NOT for both access.

42. **Theorem (309, 319)** Shared-memory Multiprocessor (SMP)：
    - UMA：通過 bus 傳輸，因此同質 CPU；NUMA：Netwrok 傳輸，can scale to larger sizes and have lower latency to local memory。

- Snooping 解決 UMA 和 NUMA 中 cache coherence problem：
  - Write-invalidate：先將其他 caches invalidate 後，在 update 自己的 cache，降低 bus bandwidth 需求。
  - Write-update (broadcast)：通過 bus broadcast 最新的值。

43. **Theorem (325)** Multithreading：

- Fine-grained multithreading：Switches between threads on each instruction pack, but slows down the execution of individual threads. 隱藏 long/short stalls 造成的 throughput 損失。
- Coarse-grained multithreading：Switches only on costly stalls, e.g. L2 cache misses, but cost at startup because of flush and refill on thread switches.
- Simultaneous Multithreading (SMT)：Hardware multithreading. Multiple-issue, dynamically scheduled processor (superscalar) to exploit ILP and Thread-level parallelism (TLP).

44. **Theorem (332)** GPU：

- NOT rely on multilevel caches, but rely on enough threads to hide the latency to memory.
- GPU towards bandwidth rather than latency.
- NO support for double precision floating-point arithmetic.

References

[1] 張凡. 計算機組織與結構重點直擊（上）. 鼎茂圖書出版股份有限公司, 3 edition, 2019.

[2] 張凡. 計算機組織與結構重點直擊（下）. 鼎茂圖書出版股份有限公司, 3 edition, 2019.

[3] wjungle@ptt. 計算機組織 @tkb 筆記. https://drive.google.com/file/d/0B8-2o6L73Q2VUkpEMWVLb1pRZE0/view?usp=sharing, 2017.