計算機組織

Computer Architecture

TZU-CHUN HSU[1]

[1]vm3y3rmp40719@gmail.com

[1]Department of Computer Science, Zhejiang University

2021 年 1 月 17 日

Version 3.2

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]。

本文作者為 TZU-CHUN HSU，本文及其 LaTeX 相關程式碼採用 **MIT 協議**，更多內容請訪問作者之 GitHub 分頁Oscarshu0719。

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. 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 (28, 47, 59)**

   - `srl/sll rd, rt, shamt # rs = 5'0`
   - `lw/sw rt, imm(rs)`
   - `beq/bne rs, rt, addr`
   - `addi rt, rs, imm`
   - `lb`：Load 最高 address（LSB）到最高 address（LSB），`sb`：Store 最高 address（LSB）到最低 address（MSB）。
   - `jr rs # rt = rd = shamt = 5'0`：R-type

3. **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
```

4. **Theorem (190)** 浮點數：

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

5. **Theorem (214, 215)** Overflow detection：

- 有號數：

```
addu $t0, $t1, $t2
xor $t3, $t1, $t2
slt $t3, $t3, $zero # $t3 = 1 if sign differs
bne $t3, $zero, NO_OVERFLOW
xor $t3, $t0, $t1 # Check if the sum sign differs
slt $t3, $t3, $zero
bne $t3, $zero, OVERFLOW
```

- 無號數：

```
addu $t0, $t1, $t2
nor $t3, $t1, $zero # 2^32 - $t1 - 1
sltu $t3, $t3, $t2 # 2^32 - $t1 - 1 < $t2 -> 2^32 - 1 < $t1 + $t2
bne $t3, $zero, OVERFLOW
```

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

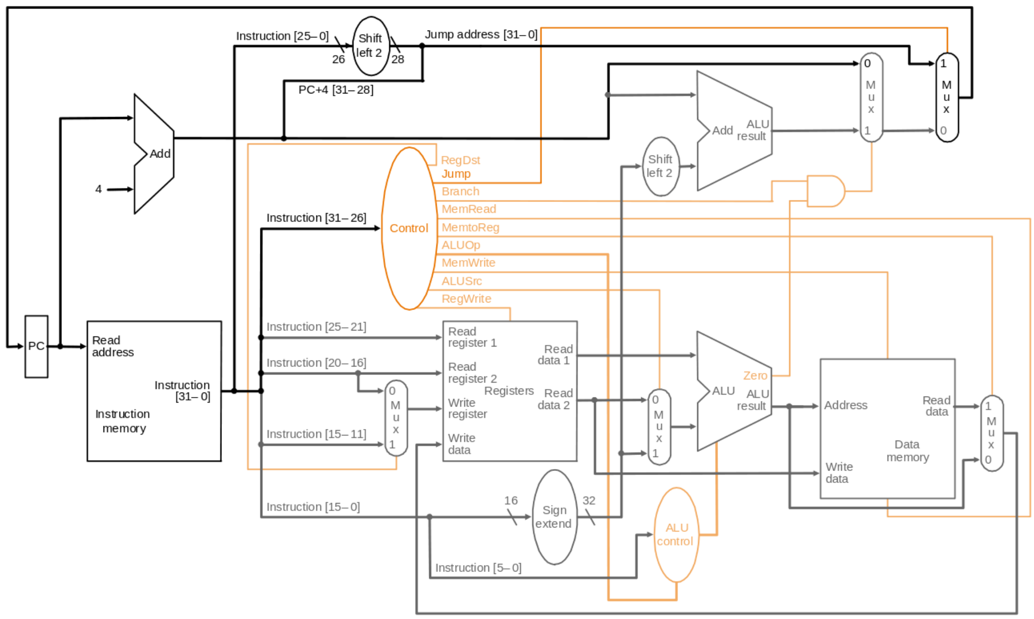

Figure 1: Single-cycle CPU with jump and branch.

7. **Theorem (384)**

Instruction	ALUOp1	ALUOp2
`lw/sw`	0	0
`beq`	×	1
R-type	1	×

8. **Theorem (441)** 原始 pipeline 設計：

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

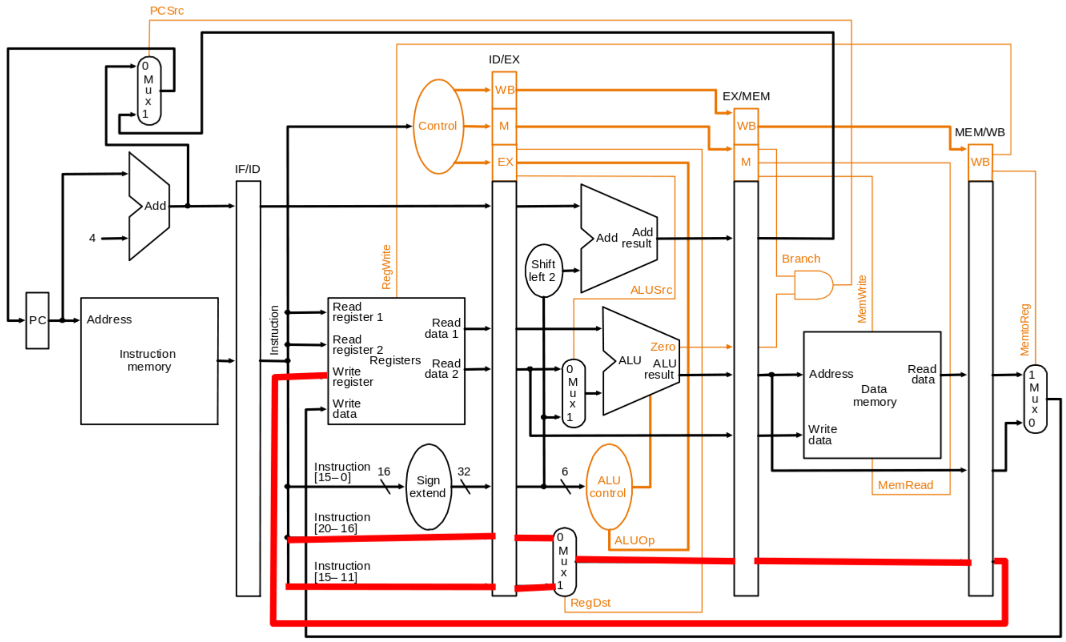

Figure 2: Original pipeline.

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

- Forwarding：Combinational units，放在 EX 因為 ALU。

```
if (EX/MEM.RegWrite ∧ (EX/MEM.Rd ≠ 0) ∧
    (EX/MEM.Rd = ID/EX.Rs/Rt))
    ForwardA/B = 10
```

Listing 1: EX hazard.

```
if (MEM/WB.RegWrite ∧ (MEM/WB.Rd ≠ 0) ∧
    (¬ EX_hazard) ∧ (MEM/WB.Rd = ID/EX.Rs/Rt))
    ForwardA/B = 01
```

Listing 2: MEM hazard.

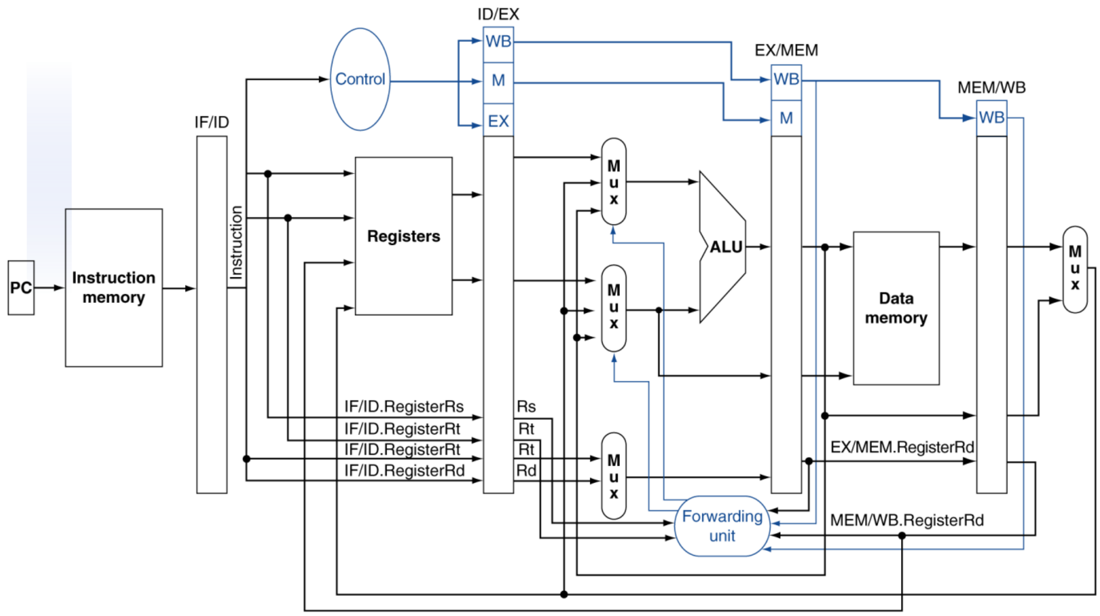

Figure 3: Pipeline with forwarding.

- Stall:

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

Listing 3: Stall.

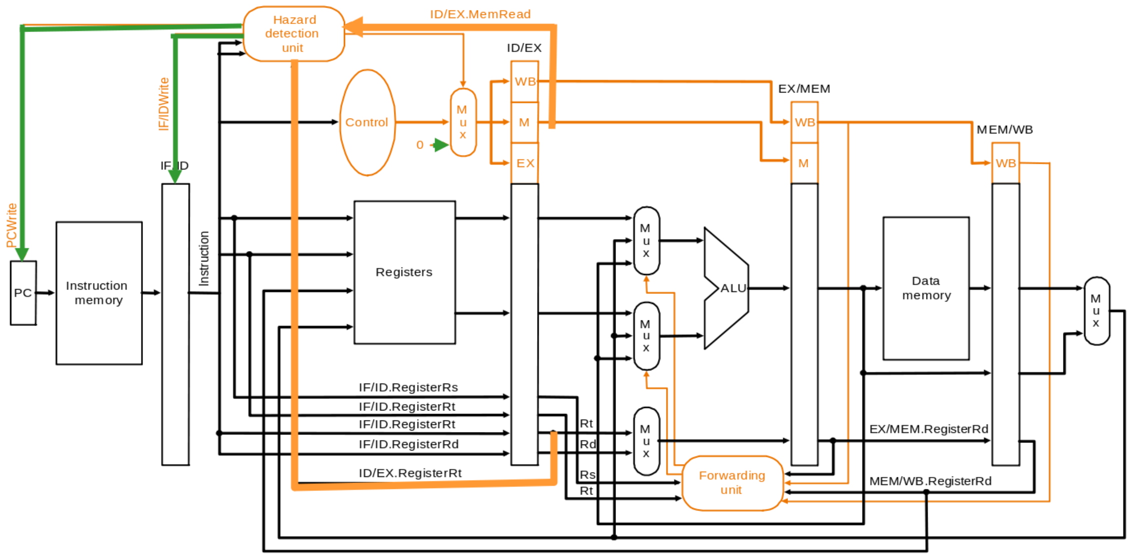

Figure 4: Pipeline with hazard detection and forwarding units.

10. **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。
- 分支指令通過 `xor` 再 `nor` 比較是否相同。

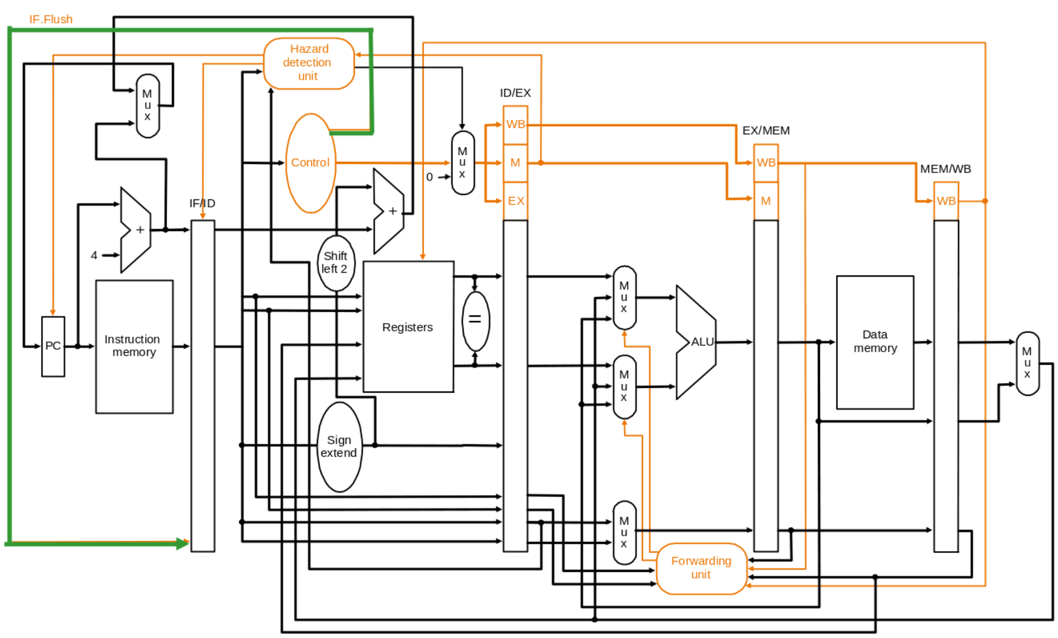

Figure 5: Pipeline with hazard detection, forwarding units and flush.

- 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，有跳才提升。
  - From fall through：用於 branch 發生機率低，不跳才提升。

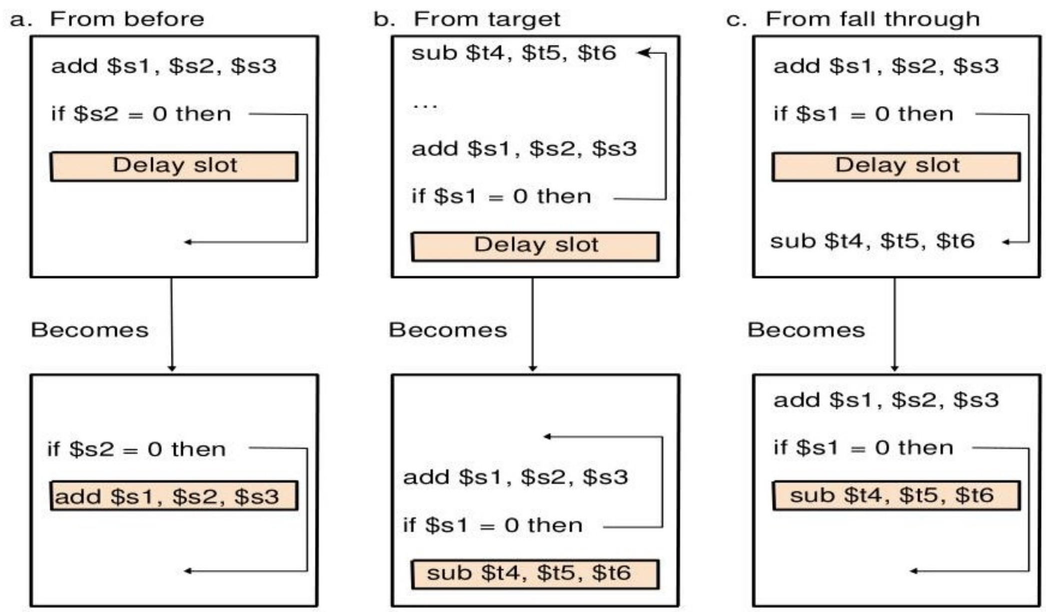

Figure 6: Example of delayed branch.

11. **Theorem (499, 505, 511)**

- Intel IA-64 (EPIC)：
  - 支援利用 compiler 開發的平行度。
  - 可以猜測，並利用 if-else 取代 branch。
  - Registers 比 MIPS 多很多。
  - Instruction group is a sequence of instructions which does **NOT** have data dependency and can be executed parallelly.
- Speculation 錯誤復原：
  - 軟體提供修補程式。
  - 硬體 CPU 將猜測結果暫時儲存，若正確，則將猜測結果寫回 register 或 memory，否則 flush buffer。

Advanced pipeline		
Technique	Hardware	Software
Branch prediction	√	√
Speculation	√	√
Intel IA-64 (EPIC)	√	√
Register renaming	√	√
Prediction		√

12. **Theorem (515)** MIPS exception handling：
    - 利用 cause register 儲存 exception 原因。
    - 將造成 exception 的 instruction memory address 存在 EPC ($PC + 4$)。
    - 使用 entry point switch to kernel。
    - Exception handling routine 須將 $PC - 4$。
13. **Theorem (27, 48)** Cache：
    - Split cache 通常有較差 hit ratio，提升 bandwidth，但不提升 speed。
    - L1 cache：注重減少 hit time；L2 cache：注重減少 miss ratio。
14. **Theorem (195)** Non-blocking cache：
    - Does **NOT** allow **miss under hit** to hide miss latency.
    - **Miss under miss** allows multiple outstanding cache misses.
    - Allow a **load** instruction to access the cache if the previous **load** is a cache miss.
15. **Theorem (213, 278, 289)** RAID：
    - 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：
        - 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：
  - 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：
  - Write 就不會有單一 disk 瓶頸。
  - $n + 1$ disks，parity 被分散到所有 disks。
  - 可允許 1 個 disk 故障。
- RAID 6：
  - 與 RAID 5 相比，增加第二個獨立的 parity block。
  - 通常通過硬體實現。
  - $n + 2$ disks。
  - 可允許 2 個 disk 故障。
- RAID 3 has **worst** throughput for **small writes**.
- RAID 3 has **best small writes latency**.
- RAID 3, 4, 5 have same throughput for **large writes**.
- RAID 1 can **NOT** have **small writes** in parallel.
- RAID 3 can **NOT** have **small writes or reads** in parallel.
- RAID 4, 5 perform same for parallel **small reads and writes**.
- RAID 4 does **NOT** have better **big reads** performance than RAID 3.
- RAID 1+0 has better **write throughput** than RAID 0+1.

16. **Theorem (320)**

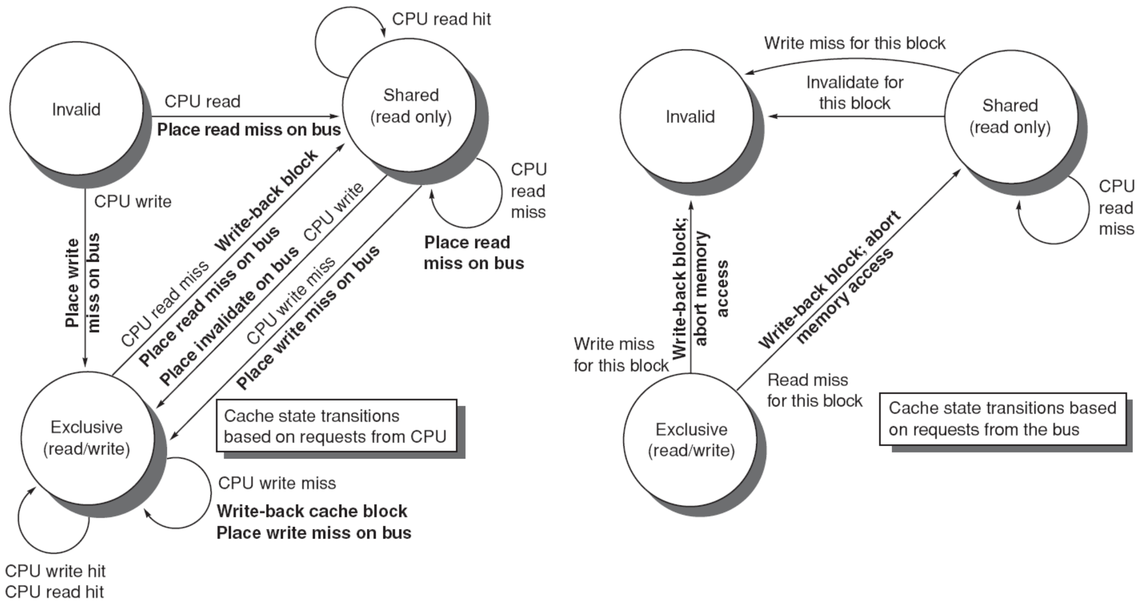

Figure 7: Snooping states.

17. **Theorem (325)** Multithreading：

- Coarse-grained multithreading: Switch threads only on costly stalls, such as L2 cache misses. Pipeline **start-up** costs.
- Fine-grained multithreading: Switch between threads on each instruction packs. It can hide the throughput losses.
- SMT: **ILP** and **TLP**; coarse-grained and fine-grained: **TLP only**.

18. **Theorem (334)** Network topology：

- Performance measure：
  - Network bandwidth.
  - Bisection bandwidth：平均切為二所減少的 bandwidth，越高容錯力越高。
  - Diameter：任兩點最短路的最大值，越低越好。
  - Nodal degree：CPU degree，越高容錯力越高。
- Omega network hardware: $2n\log_2 n$.
- Crossbar network hardware: n^2.

19. **Theorem ()** NAS vs SAN：

- NAS operates at **file** level while SAN operates at **block** level.

- CIFS/SMB and NFS are examples of NAS.
- SAN is often the preferred choice over NAS.
- Almost any machine running Microsoft Windows with LAN connectivity can be configured to access a NAS.

20. **Theorem ()** Log-structured file system：
    - 將要 write 的 data 合成一串，再一次 write。
    - Read 都在 cache，因為 cache 夠大。
    - Disk access 的 seek 和 rotation 是 bottleneck，sequential access 比 random access 好。
21. **Theorem ()**
    - Meltdown: read arbitrary kernel memorya, and it does **NOT** rely on software vulnerabilities.
    - Spectre: Making other applications to access arbitrary contents in memory.
    - Both belongs to **side channel attacks**.
    - Does **NOT** leave records in traditional log.
    - Hard for antivirus software to detect them.
    - Processors which are able to implementat out-of-order execution is risky.
    - IA-64 is immune to Spectre and Meltdown.
22. **Theorem ()** Power：
    - CMOS does **NOT** consume power when it's **static** ($power_{static} = 0$), so it can decrease **frequency** to save power.
    - **Static** power dissipation occurs because of leakage current that flows even when a transistor is **off**.
    - Computers at **lower utilization** does **NOT** use less power proportionally.
    - The main reason for the switch from high-performance uniprocessors to multiprocessors with simpler cores and lower clock rates in recent years is the **power limit** and **memory gap**.
23. **Theorem ()**

- Out-of-order execution in **cache** level do NOT fail.
- GPGPU usually runs **SPMT** (Single Program Multiple Thread), and GPU runs SIMT.
- L1 data cache is usually seperated from L1 instruction cache to **increase bandwidth**.
- Data cache is usually deployed at **MEM** stage.
- Increasing number of **used sticky bits** do NOT improve accuracy.
- **Memory hazard** do NOT cause stall, e.g. `sw` after `lw`.
- Branch target buffer is used by **CPU**, which is checked at **IF** stage.
- Program is a **passive** entity, process is an **avtive** entity.
- Branch prediction buffer is good to predict the **branch outcome**, but it does **NOT** help in predicting the **branch target**.
- Many routers are equipped with **firewall** and **VPN** functions.
- In hash-based page tables using **linked list** to solve collision, **each element** contains a frame number and a page number.
- Multiple-cycles CPU requires **minimum function units**.
- Control hazards can **NOT** be avoided.
- Conversion from single-precision to double-precision causes loss of precision.
- Compiler identifies **basic blocks** for code optimization.
- Vector processors need **less bandwidth** than conventional processors.
- Ripple Carry Adder: Critical path delay is $2N$ gate delay (carry out), and sum delay is $2N + 1$ gate delay (actual sum).
- To form the machine code, the value of label of branch instructions is computed by **linker** when the label is an **external** reference.
- **NOT** each computer support **direct addressing mode**.
- Converting an integer variable to a **single** precision FP number will lose precision, but **double** precision does **NOT**.
- When the block size is very large, the **spatial locality** within the block is lower.
- MIPS uses **two** seperated page tables and two limit registers, one for **stack** and the other for **heap**.
- **Conflict** misses do **NOT** occur in **fully associative** caches.

- In modern preocessors, **L1** data and instruction caches are split, but L2 does **NOT**. Both L1 and L2 caches are **write-back**.
- **MIPS** and **ARM** use **memory-mapped I/O**.
- Writes are much **slower** than reads for flash. NAND flash is **cheaper** than NOR flash.
- False sharing: Irrelative variables are stored in the same block, when one variable is written, the whole block is changed, and other variables are affected.
- GPUs do **NOT** rely on **multilevel caches**.

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.