## Multiple Cycle Machine 概念

將指令分解成一連串的步驟，每個步驟花一個 clock cycle time 執行

執行時間長的步驟越多 ⇒ clock cycle 多

⇒ 以最長 Step 做為 Multiple Cycle Machine 的 Clock Cycle Time

∴ Step 和 Step 間執行時間越接近（平衡）越好（切割得越平衡越好）

⇒ 規定：一個 Step 只能用一個主要功能單元（如 Reg, IM, DM, ALU, ADDER）

## MIPS 各指令的 Step 和 CPI

Step 只要分為：1. IF (instruction fetch)：擷取指令，PC+4（用到 IM）

2. ID & RF (instruction decode & register fetch)：解碼指令，抓暫存器內容（用到 Reg）

3. ALU Operation：ALU 運算，算記憶體位址（用到 ALU）

4. MA：記憶體資料存取（用到 DM）

5. WB (Write Back)：將運算結果寫入 Register file（用到 Reg）

| Instruction | IF | ID RF | EX | MA | WB | CPI |
|---|---|---|---|---|---|---|
| R-type | v | v | v | | v | 4 |
| lw | v | v | v | v | v | 5 |
| sw | v | v | v | v | | 4 |
| beq | v | v | v | | | 3 |
| j | v | v | v | | | 3 |

## Multiple Cycle Machine 效能

1. 求出各個指令的 CPI；
2. 利用各類指令 freq，算出平均 CPI
3. 決定出 clock cycle time
4. 算出平均指令的 Exe Time

## Multi-Cycle Microarchitectures

- Goal: Let each instruction take (close to) only as much time it really needs
- Idea
  - Determine clock cycle time independently of instruction processing time
  - Each instruction takes as many clock cycles as it needs to take
    - Multiple state transitions per instruction
    - The states followed by each instruction is different

### Single-cycle machines

- Each instruction takes a single clock cycle
- All state updates made at the end of an instruction's execution
- Big disadvantage: The slowest instruction determines cycle time → long clock cycle time

### Multi-cycle machines

- Instruction processing broken into multiple cycles/stages
- State updates can be made during an instruction's execution
- Architectural state updates made only at the end of an instruction's execution
- Advantage over single-cycle: The slowest "stage" determines cycle time

## Multi-Cycle Microarchitecture

AS = Architectural (programmer visible) state at the beginning of an instruction

↓

Step 1: Process part of instruction in one clock cycle

↓

Step 2: Process part of instruction in the next clock cycle

↓

...

↓

AS' = Architectural (programmer visible) state at the end of a clock cycle

有 immediate state 存在

Single-cycle machine:

- Control signals are generated in the same clock cycle as the one during which data signals are operated on
- Everything related to an instruction happens in one clock cycle (serialized processing)

Multi-cycle machine:

- Control signals needed in the next cycle can be generated in the current cycle
- Latency of control processing can be overlapped with latency of datapath operation (more parallelism)

## Benefits of Multi-Cycle Design

- Critical path design
  - Can keep reducing the critical path independently of the worst-case processing time of any instruction
- Bread and butter (common case) design
  - Can optimize the number of states it takes to execute "important" instructions that make up much of the execution time
- Balanced design
  - No need to provide more capability or resources than really needed
    - An instruction that needs resource X multiple times does not require multiple X's to be implemented
    - Leads to more efficient hardware: Can reuse hardware components needed multiple times for an instruction

## Microprogrammed Multi-Cycle uArch

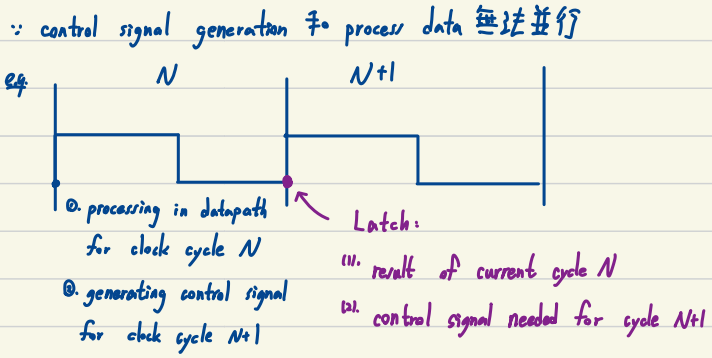
- Key Idea for Realization
  - One can implement the "process instruction" step as a finite state machine that sequences between states and eventually returns back to the "fetch instruction" state
  - A state is defined by the control signals asserted in it
  - Control signals for the next state determined in current state

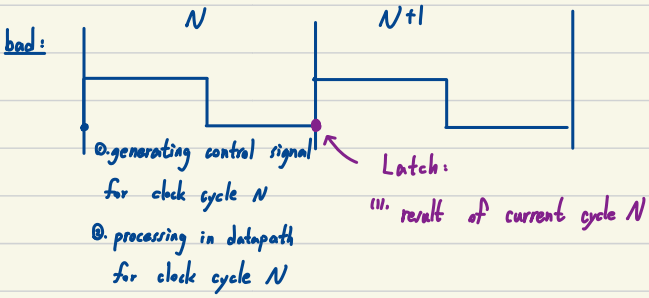
## A Basic Multi-Cycle Microarchitecture

- Instruction processing cycle divided into "states"
  - A stage in the instruction processing cycle can take multiple states
- A multi-cycle microarchitecture sequences from state to state to process an instruction
  - The behavior of the machine in a state is completely determined by control signals in that state
- The behavior of the entire processor is specified fully by a *finite state machine*
- In a state (clock cycle), control signals control two things:
  - How the datapath should process the data
  - How to generate the control signals for the next clock cycle

Question: why not generate control signals for the current cycle in the current cycle?

- This will lengthen the clock cycle
- Why would it lengthen the clock cycle?




∵ ② depends on ①, 若 ① 需花費時間, clock cycle time 上升

- What limitations do you see with the multi-cycle design?
- Limited concurrency
  - Some hardware resources are idle during different phases of instruction processing cycle
  - "Fetch" logic is idle when an instruction is being "decoded" or "executed"
  - Most of the datapath is idle when a memory access is happening

## Can We Use the Idle Hardware to Improve Concurrency?

- Goal: Concurrency → throughput (more "work" completed in one cycle)
- Idea: When an instruction is using some resources in its processing phase, process other instructions on idle resources not needed by that instruction
  - E.g., when an instruction is being decoded, fetch the next instruction
  - E.g., when an instruction is being executed, decode another instruction
  - E.g., when an instruction is accessing data memory (ld/st), execute the next instruction
  - E.g., when an instruction is writing its result into the register file, access data memory for the next instruction