Lecture 07 – Dependences and Pipelining

기초

ㅋㅋ 장점만 해보아...

Jaejin Lee

Dept. of Computer Science and Engineering, College of Engineering

Dept. of Data Science, Graduate School of Data Science

Seoul National University

http://aces.snu.ac.kr

Processor, CPU, and Core

- Poorly defined terms in these days
  - A processor or CPU typically refers to the physical chip (package)
- Core
  - A hardware unit that fetches instructions and executes them
  - Contains hardware components involved in executing instructions
    - ALU, FPU, (private) L1/L2 caches, etc.
- Processor or CPU
  - The combination of one or more cores with shared resources between cores and some supporting hardware
  - A processor sometimes refers to a processor that contains a single core or the core itself depending on the context

Uncore

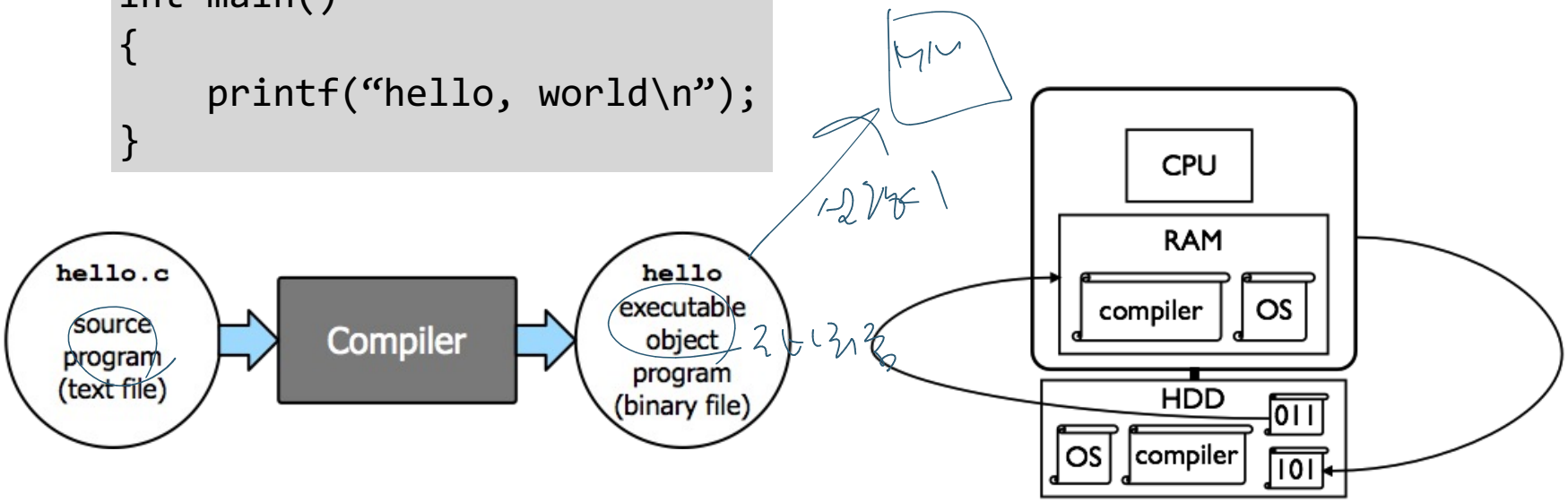
- A term used by Intel
- Hardware components that are not in the core
  - QPI controllers, last level caches (e.g., L3 cache), on-chip memory controllers, etc.

Compilers and Compilation Process

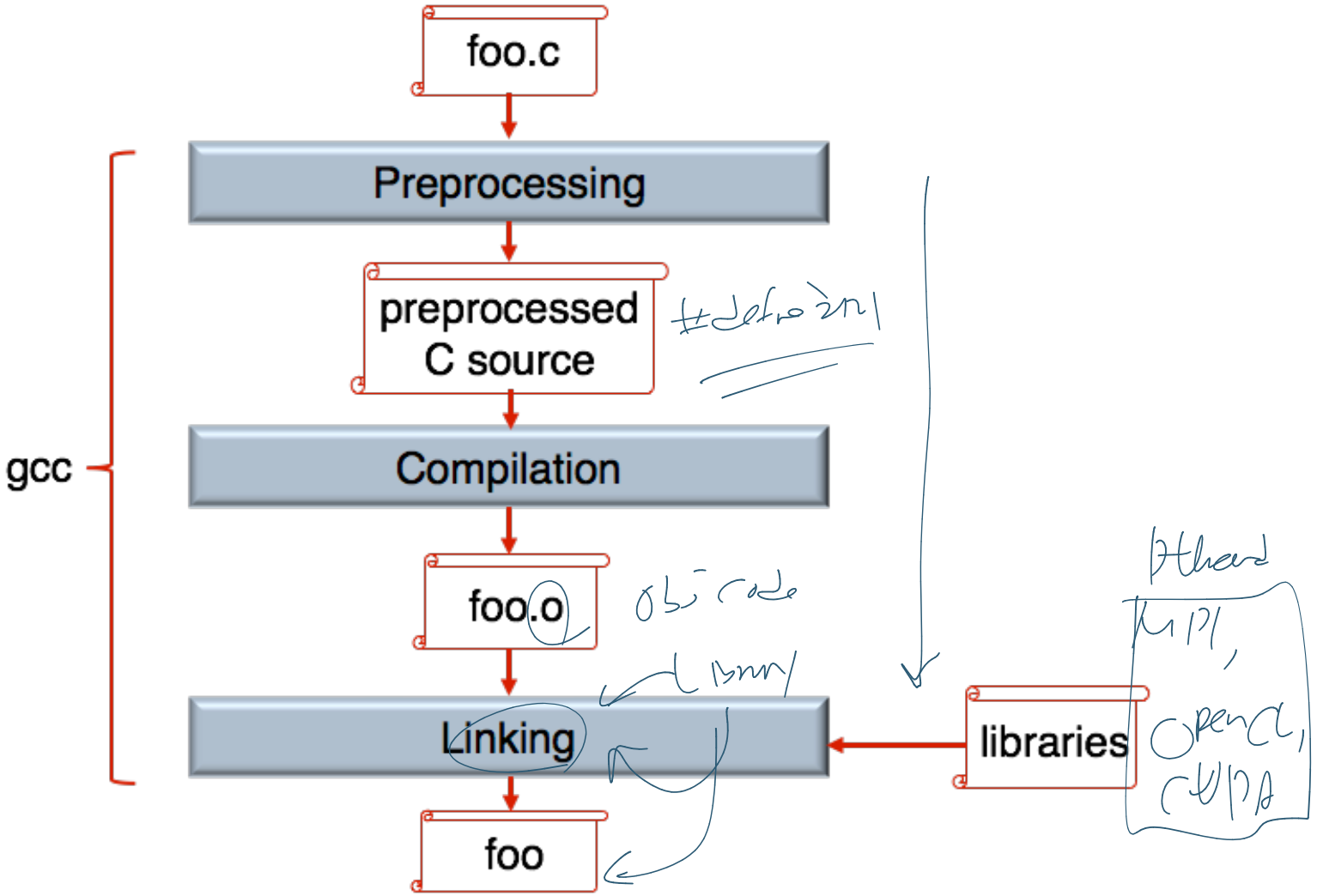
- A compiler is a program that automatically translates another program from some programming language to machine code

```
#include <stdio.h>

int main()
{
    printf("hello, world\n");
}
```



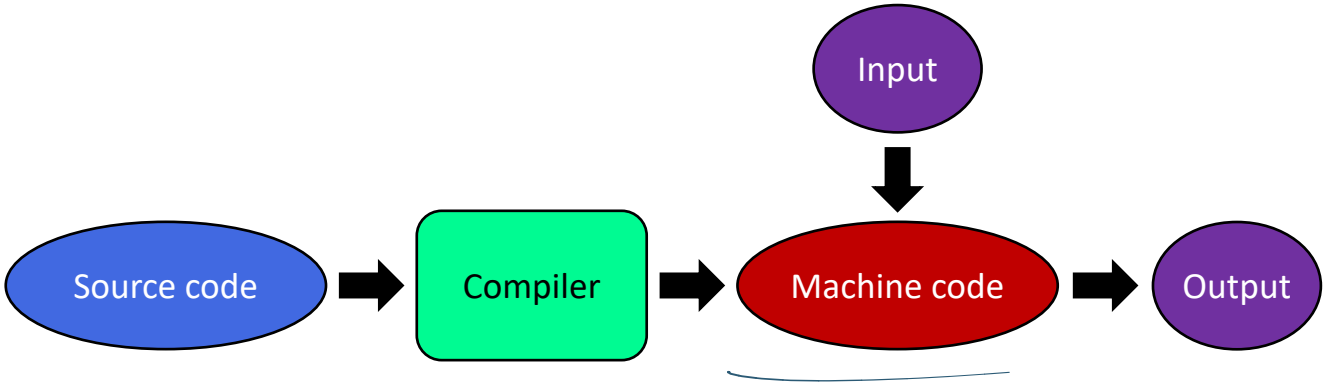
Typical Compilation Phases

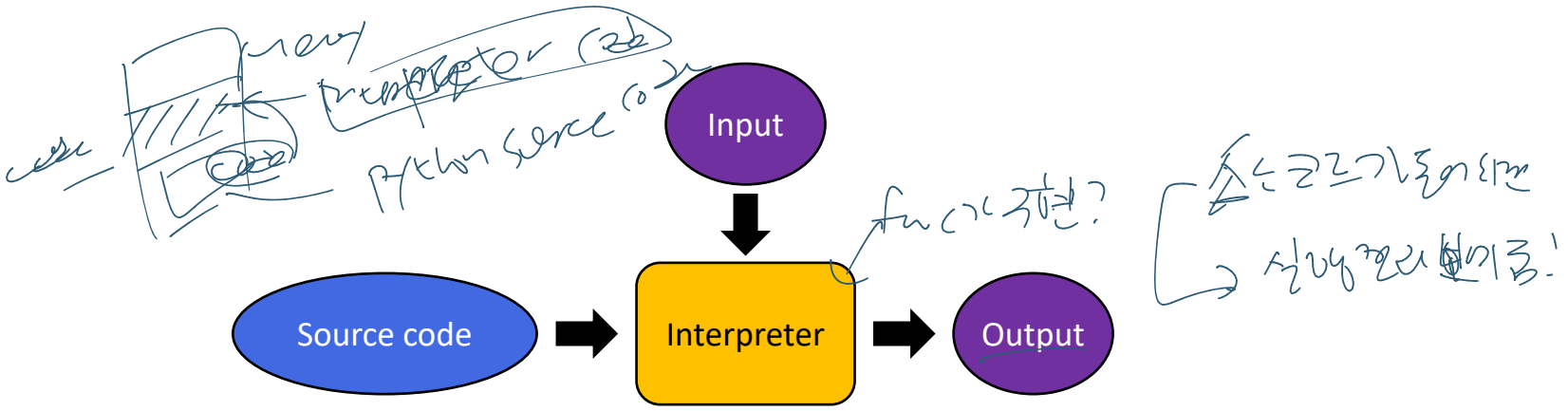


Interpreters and Interpretation Process

- An interpreter is a program that takes as input a specification of a program in some language and the output of the input program is its output
  - It translates the source code into some intermediate representation
    - A parser in the interpreter takes a string representation of a program and produces a structural parse of the input program
  - An evaluator in the interpreter executes the intermediate representation
  - The interpreter may execute stored precompiled code made by a compiler
- Python, Java bytecode interpreter

Compilation vs. Interpretation





Techniques to Improve a Single Core Performance

- As VLSI technology improves, more room becomes available on a chip
- Two major techniques
  - Instruction pipelines
  - On-chip caches

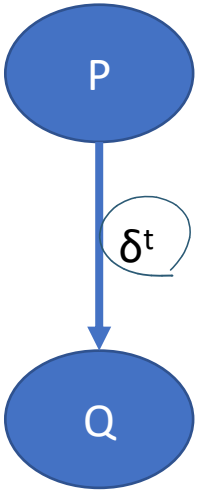
Dependences

- An ordering relationship between two computations
  - Any ordering of execution that obeys all dependences will produce the same result as that of the original program
- Data dependences
  - Flow (true) dependence
  - Anti dependence
  - Output dependence
  - Input dependence (not really a dependence)
- Control dependences

Flow Dependence

- True dependence
- Instruction P writes a memory location that instruction Q later reads (read after write)

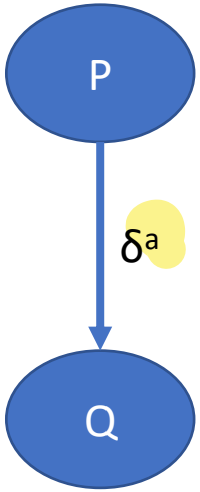
```
P: a = …;
…
Q: … = a;
```



Anti Dependence

- False dependence (can be removed)
- Instruction P reads a memory location that instruction Q later writes (write after read)

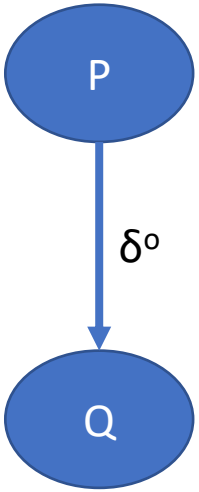
```
P: … = a;
...
Q: a = …;
```



Output Dependence

- False dependence (can be removed)
- Instruction P writes a memory location that instruction Q later writes (write after write)

```
P: a = …;
...
Q: a = …;
```



Input Dependence

- Not really a dependence
  - For caches
- Instruction P reads a memory location that instruction Q later reads (read after read)

```
P: … = a;
...
Q: … = a;

//cache hit for the access to a
```

Basic Blocks

- A sequence of statements that is always entered at the beginning and exited at the end without halt or possibility of branching except at the end
- Two consecutive instructions are in the same basic block if only if the execution of the first instruction guarantees the execution of the next instruction

```
    read m
    t0 = 1
    t1 = 1
    if m <= 1 goto L2
    i = 2
L0: if i <= m goto L1
    return t0
L1: t2 = t0 + t1
    t0 = t1
    t1 = t2
    i = i + 1
    goto L0
L2: return m
```

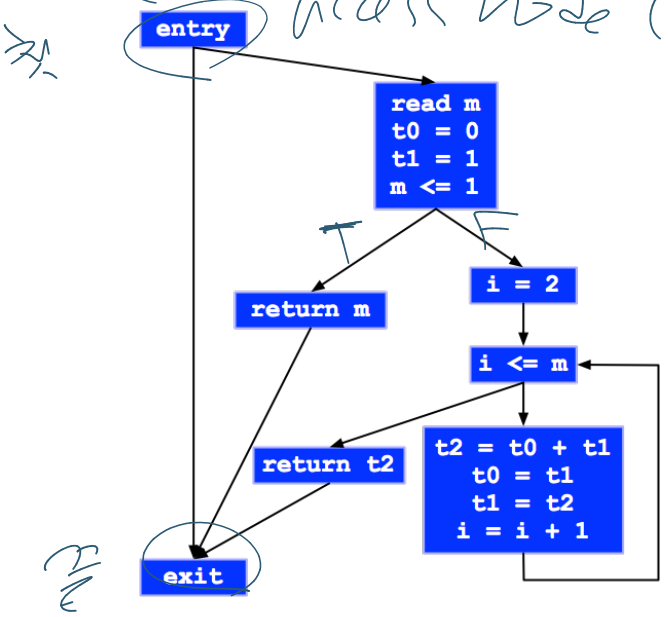
Identifying Basic Blocks

- The first instruction is a leader
- Any instruction that is the target of a conditional or unconditional jump is a leader
- Any instruction that immediately follows a conditional or unconditional jump is a leader
- For each leader, its basic block consists of itself and all instructions up to but not including the next leader or the end of the intermediate program

Control-Flow Graphs

- A control-flow graph (CFG) is a directed-graph representation of all paths that might be traversed through a program during its execution
- Flow-of-control information
- A directed graph
  - Each node in the graph represents a basic block
  - Directed edges are used to represent the potential flow of control

```
     read m
     t0 = 1
     t1 = 1
     if m <= 1 goto L2
     i = 2
L0:  if i <= m goto L1
     return t0
L1:  t2 = t0 + t1
     t0 = t1
     t1 = t2
     i = i + 1
     goto L0
L2:  return m
```

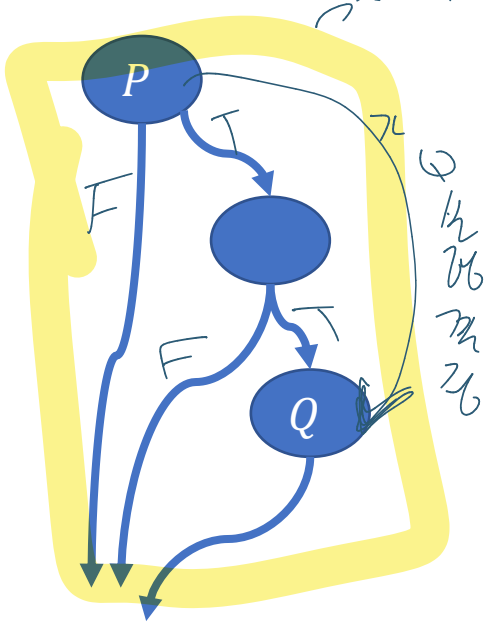


Control Dependences

- An instruction Q has a control dependence on a preceding instruction P if the outcome of P determines whether Q should be executed or not
- An instruction Q is said to be control dependent on another statement P if and only if
  - There exists a path from P to Q such that every instruction $I \neq P$ within the path will be followed by Q in each possible path to the end of the program, and
  - P will not necessarily be followed by Q, i.e., there is an execution path from P to the end of the program that does not go through Q

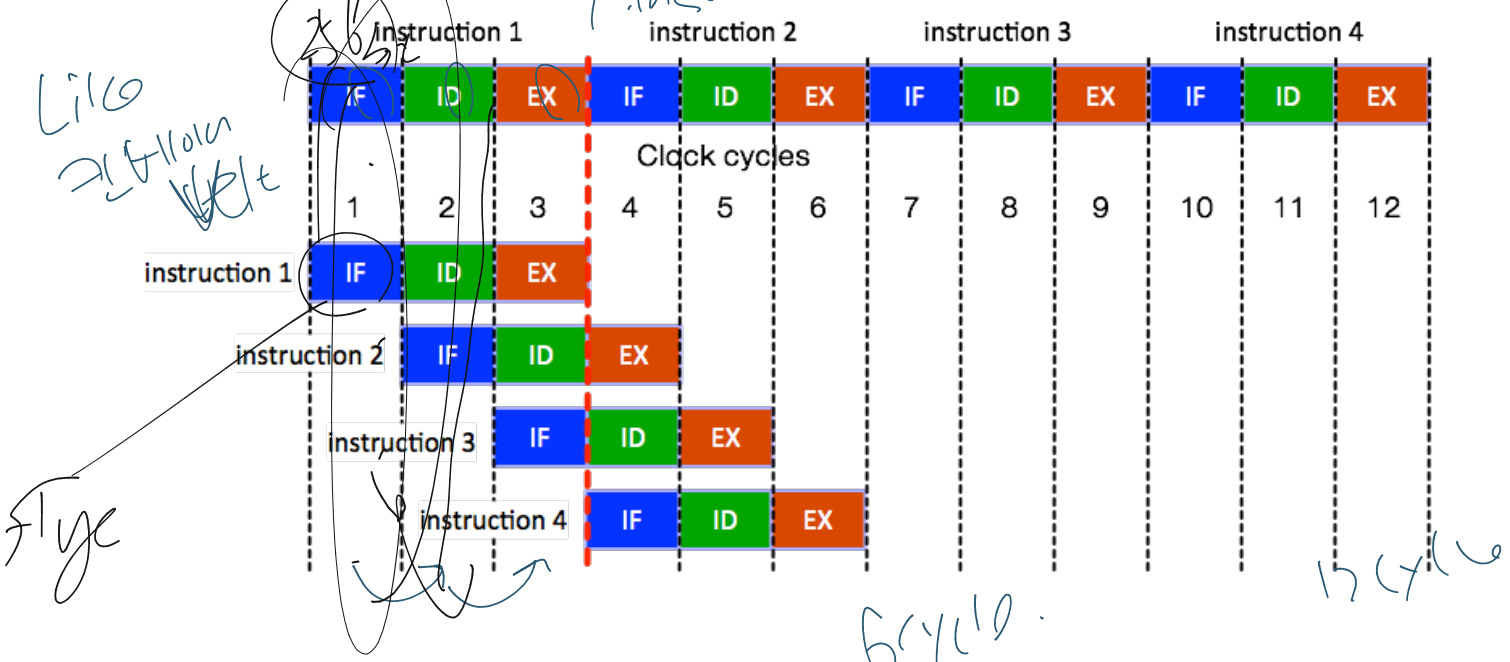
```
P: if (x > 3)
Q:     a = …;
R: a = …;
```

```
P: if (x > 3)
       if (y > 5)
Q:         a = …;
R: a = …;
```



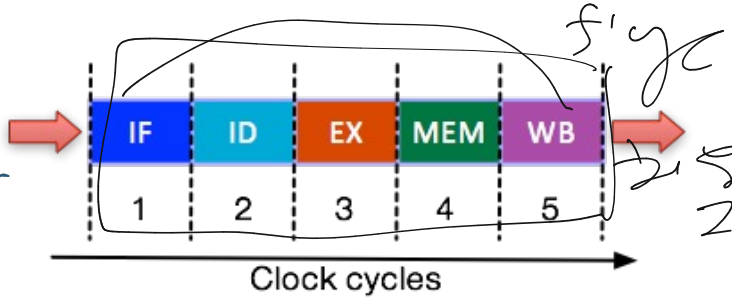
Instruction Pipeline

- Pipelining is a hardware technique that increases instruction throughput (e.g., the number of instructions executed in a CPU clock cycle)
- Assume
  - Each instruction takes 3 cycles to complete
  - Each pipeline stage takes a single cycle



Pipeline Hazards

- Typical five-stage pipeline
  - IF: instruction fetch
  - ID: instruction decode and register fetch
  - EX: execute
  - MEM: memory access
  - WB: register write back



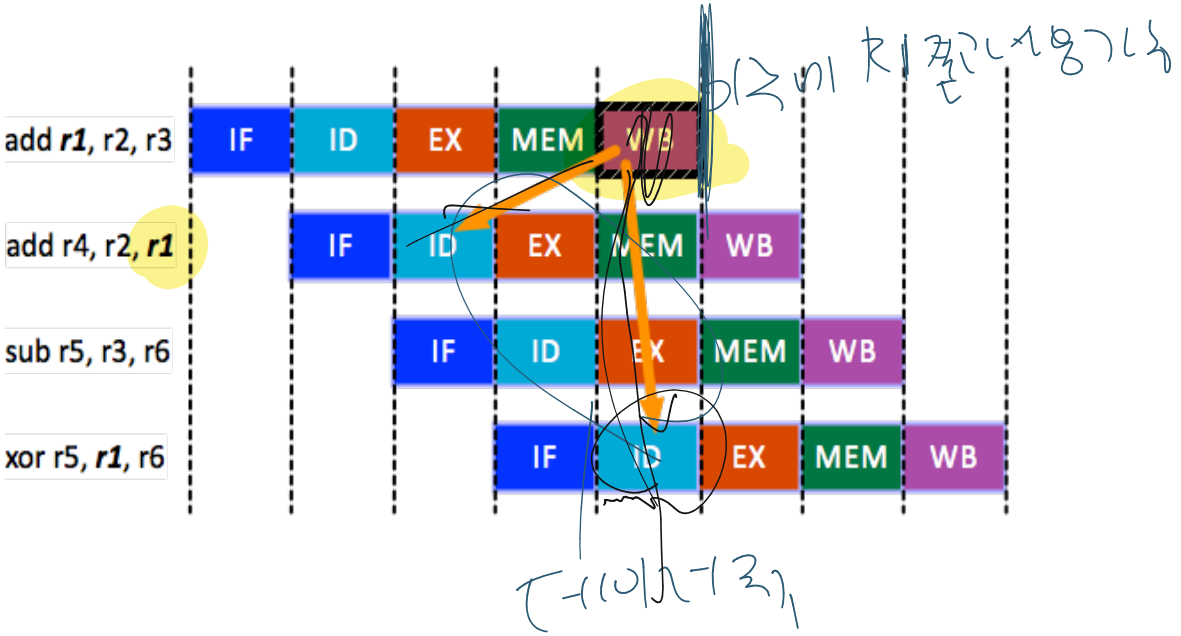
Pipeline Hazards (cont'd)

- Occur when the next instruction does not execute in the following clock cycle

- Data hazards
- Structural hazards
- Control hazards

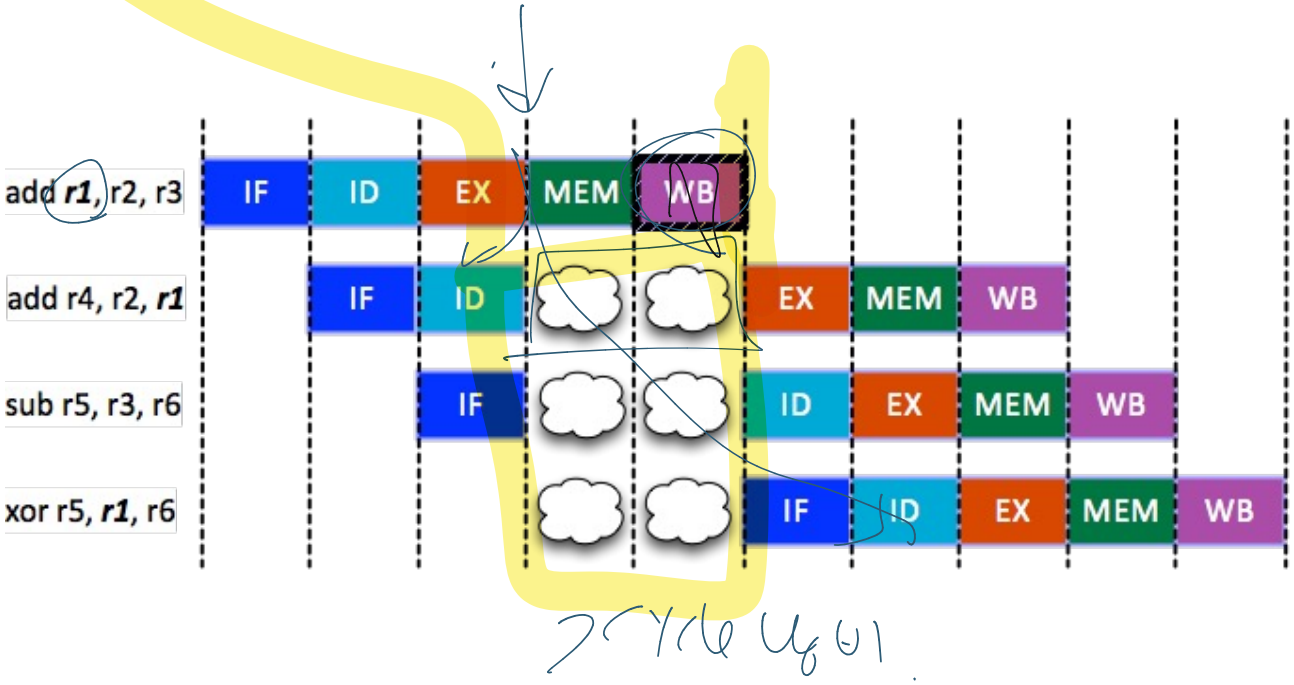
Data Hazards

- Occur when a result is needed in the pipeline before it is available



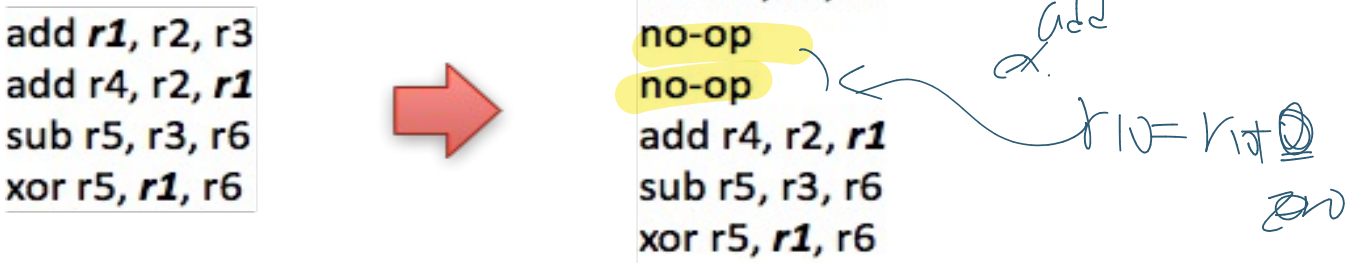
Resolving Data Hazards

- Stalling the pipeline
  - A bubble represents a delay in execution of an instruction in the pipeline to resolve a hazard



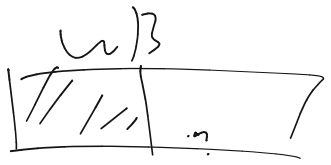
Resolving Data Hazards (cont'd)

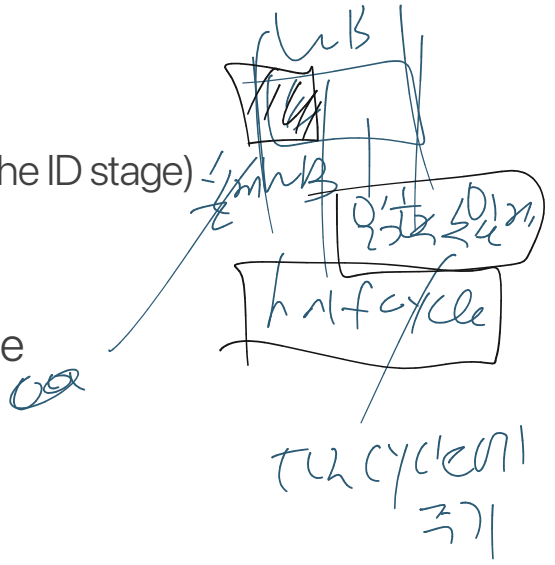
- Software
  - Insert independent instructions (or no-ops)



Resolving Data Hazards (cont'd)

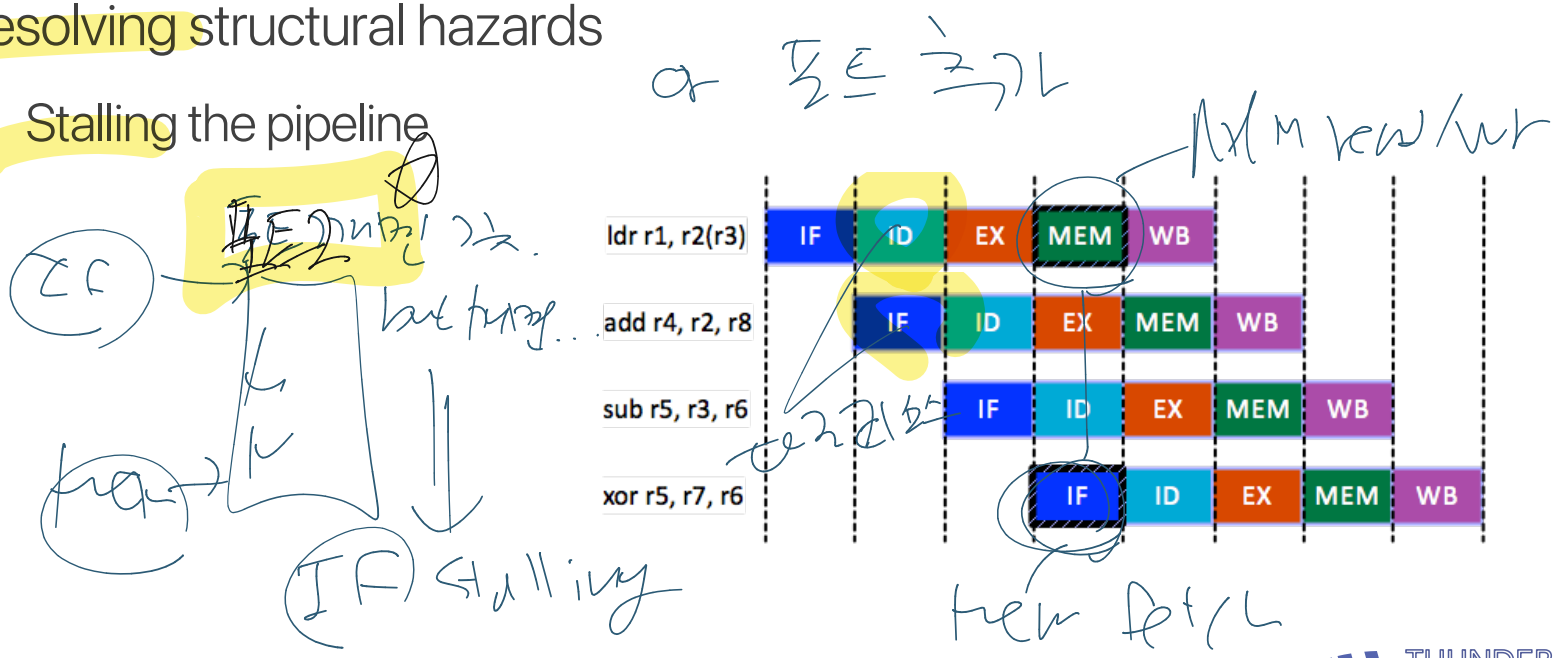
- Hardware
  - Stalling the pipeline (i.e., insert bubbles)
  - Transparent register file
    - Resolves data hazard at the **WB** stage
    - If the register file is asked to read and write the same register in the same cycle, the register file allows the written data forwarded to the stage in which the data is needed
      - First half cycle: the register is written with the data
      - Second half cycle: the register is read into the pipeline (in the ID stage)
  - Data forwarding
    - Feeding output data into a previous stage of the pipeline
    - Resolves data hazard at the **EX** and **MEM** stages





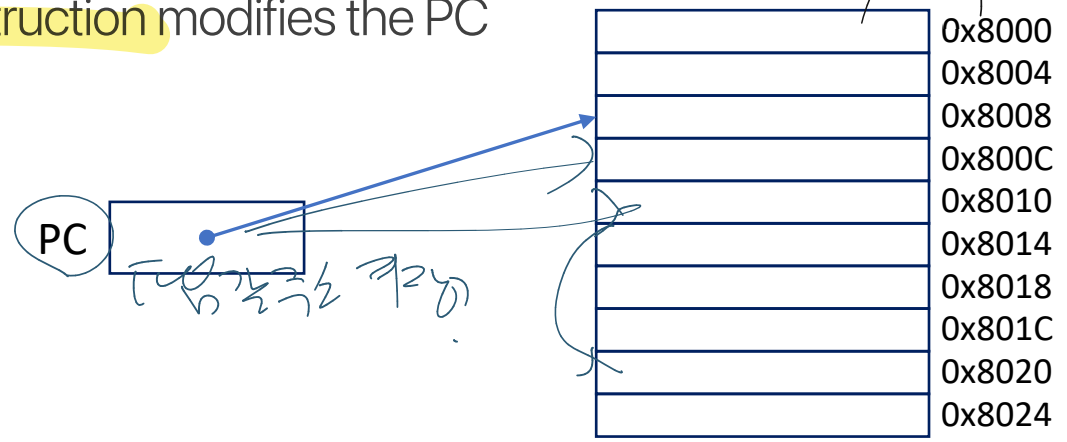
Structural Hazards

- Occur when a part of the processor's hardware is needed by two or more instructions at the same time
- Due to resource conflicts
  - A single memory unit that is accessed both in the IF stage and the MEM stage
- Resolving structural hazards
  - Stalling the pipeline



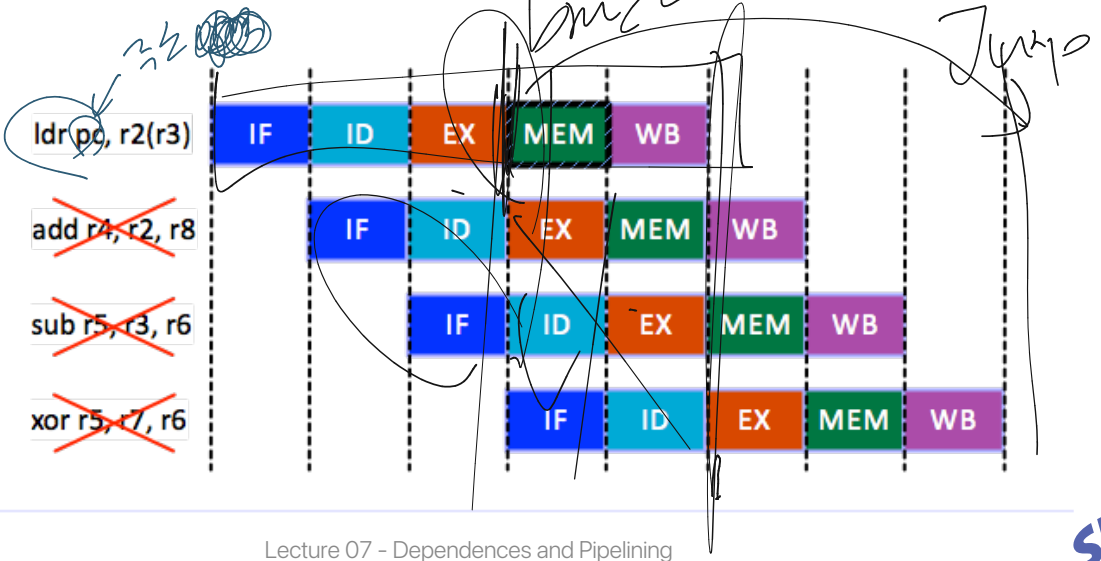
The Program Counter

- The program counter (PC) (a.k.a. the instruction pointer, the instruction address register, and the instruction counter) is a processor register that indicates where a computer is in its program sequence
- The PC is incremented after fetching an instruction, and holds the memory address of ("points to") the next instruction that would be executed
  - A branch instruction modifies the PC



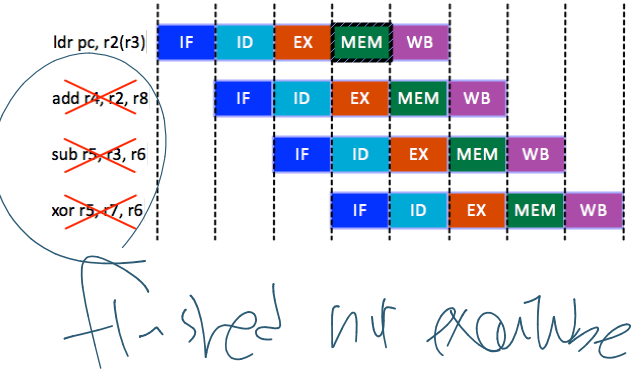
Control Hazards

- Occur because of branches
- The pipeline does not know the branch target until the instruction reaches the MEM stage
  - Assume PC is set in the MEM stage
- The pipeline continues fetching instructions sequentially
  - These fetched instructions cannot be allowed to execute because the programmer has diverted control



Resolving Control Hazards

- Software
  - Delayed branch
    - Perform instruction scheduling into branch delay slots with instructions before the branch instructions
      - From the target address (when it is known to be taken)
      - From fall through (when it is known to be not-taken)



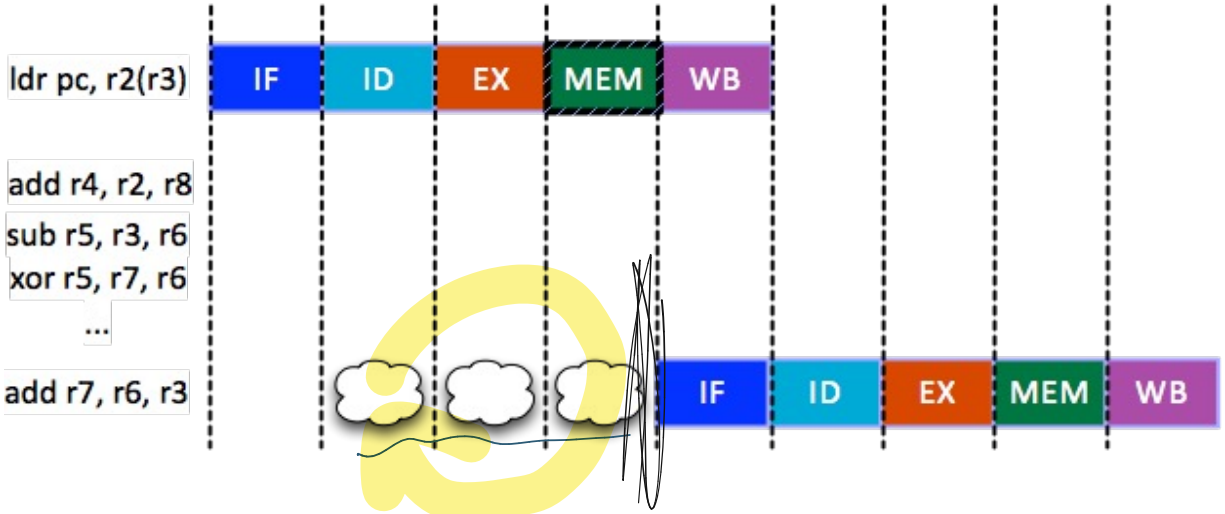
...
sub r7, r7, r2
add r9, r8, r7
add r10, r11, r9
ldr pc, r2(r3)
add r4, r2, r8
sub r5, r3, r6
xor r5, r7, r6
...

...
sub r7, r7, r2
add r9, r8, r7
add r10, r11, r9
ldr pc, r2(r3)
branch delay slot
branch delay slot
branch delay slot
add r4, r2, r8
sub r5, r3, r6
xor r5, r7, r6
...

...
ldr pc, r2(r3)
sub r7, r7, r2
add r9, r8, r7
add r10, r11, r9
add r4, r2, r8
sub r5, r3, r6
xor r5, r7, r6
...

Resolving Control Hazards (cont'd)

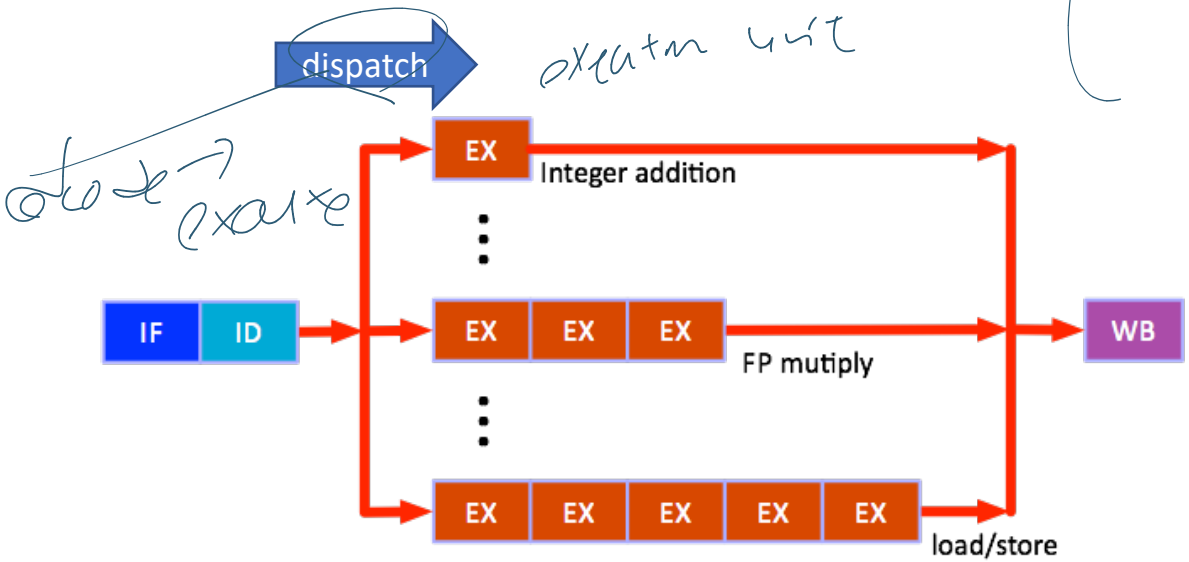
- Hardware
  - Stalling the pipeline until the branch target is known
  - Branch predictors (complicated) to prevent stalling the pipeline



- Programs written for a pipelined processor deliberately avoid branching to avoid performance degradation

In-order Execution

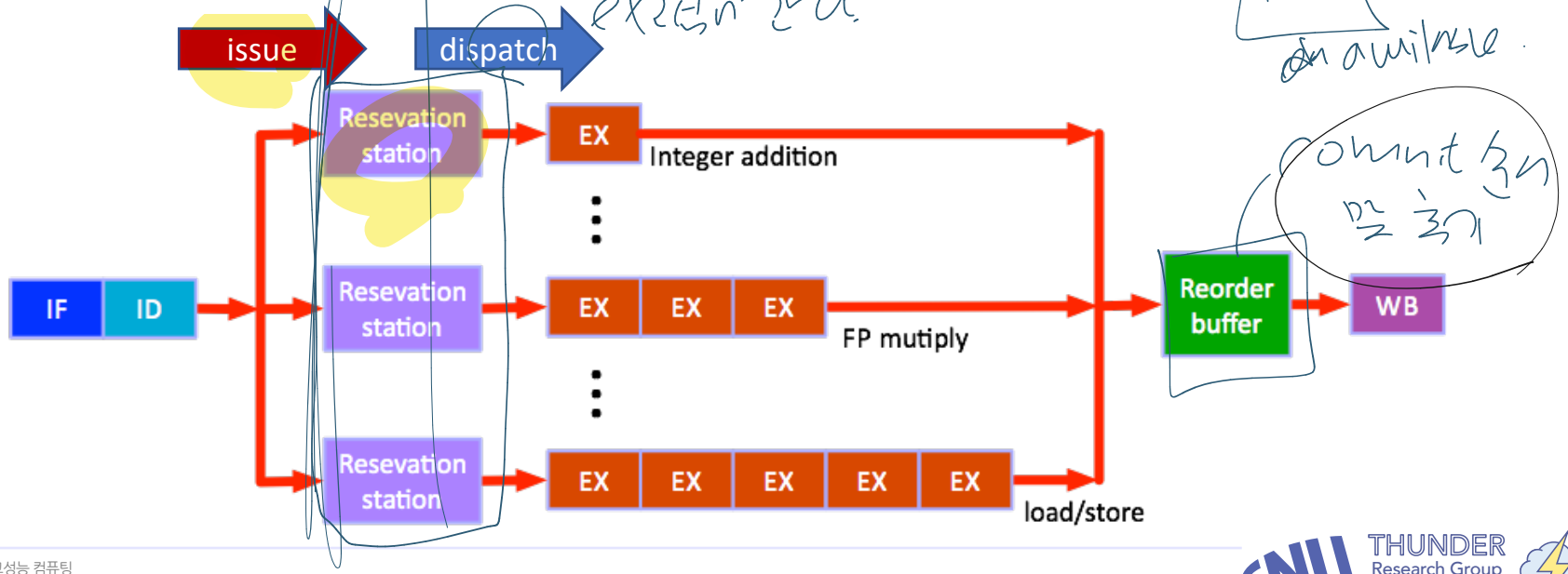
- Steps
  - Fetch and decode the next instruction
  - If its operands are available, the instruction is dispatched to the appropriate functional unit
  - Otherwise, the processor stalls until they are available
  - The result is written back to register file
    - A register file is an array of processor registers in a CPU



Assume that the EX stage for load/store instructions performs memory operations (i.e., MEM)

Out of Order Execution (OoO)

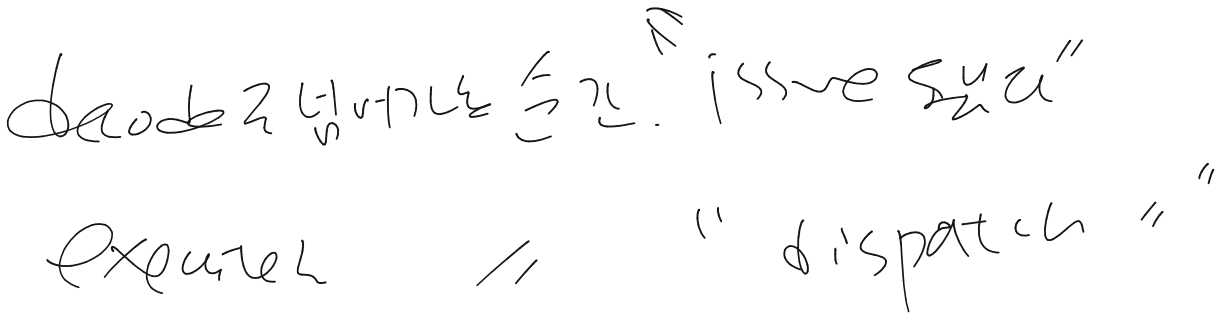
- Dynamic instruction scheduling by hardware
- Steps
  - Fetch and decode the next instruction
  - Issue it to the appropriate reservations station
  - It waits in the reservation station until its operands are available
  - The instruction is dispatched to the appropriate functional unit and executes
  - The execution completes, and the result is queued in the reorder buffer in the commit unit
  - The commit unit write results to registers and memory in program fetch order



Issuing and Dispatching an Instruction

- Once the instruction has passed the ID stage, we say that the instruction is issued

- When all operands are available, we say that the instruction is dispatched from a reservation station to the execution (functional) unit



Tomasulo's Algorithm

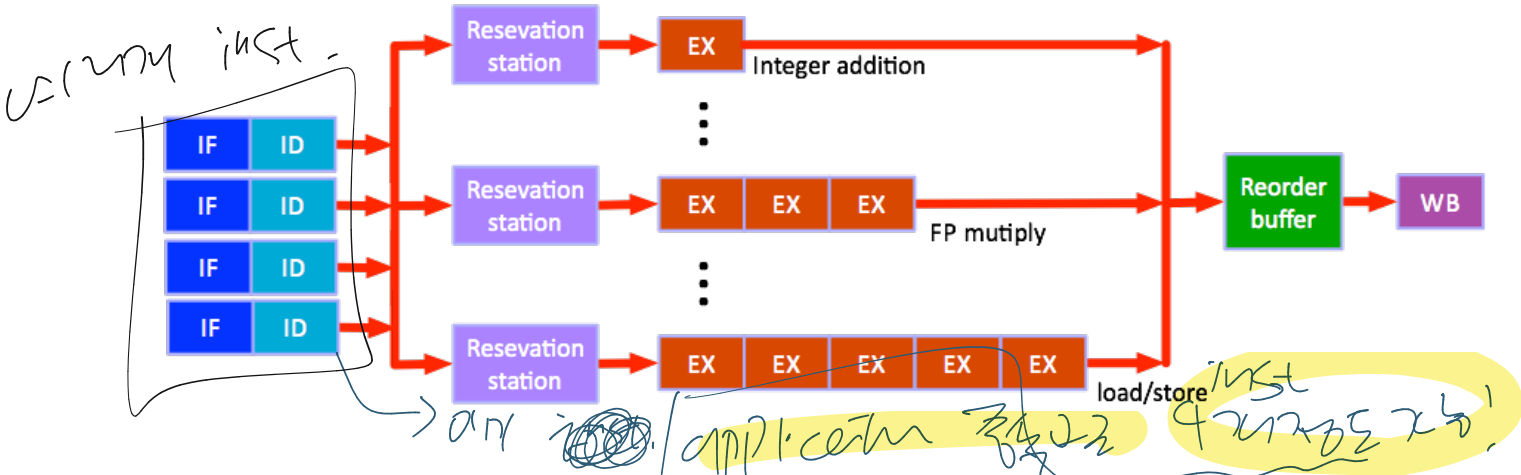
- IBM 360/91
- Single issue ?
- In-order issue
- Out-of-order dispatch
- Out-of-order execution
- In-order commit

Retirement (Graduation)

- An instruction retires when the reorder buffer slot of an instruction is freed either
  - because the instruction commits (the result is made permanent) or
  - because the instruction is removed (without making permanent changes)

Superscalar Processors

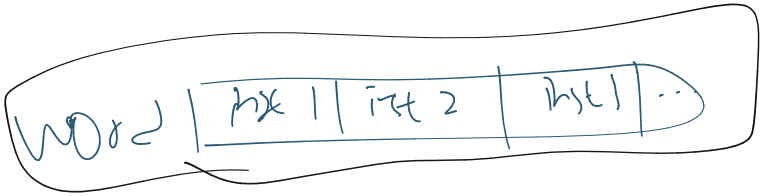
- Dynamically issue multiple instructions in each clock cycle
  - A typical superscalar processor fetches and decodes several instructions at a time
  - In-order or out-of-order issue



- Exploit the potential of ILP (Instruction Level Parallelism)
  - How many instructions can be executed simultaneously?
  - Limited amount of ILP in an application

VLIW Processors

- Use a long instruction word that contains a fixed number of instructions that are fetched, decoded, issued, and executed synchronously



- Static instruction scheduling by a compiler