Section 4.10

Introduction to Advanced Pipelining

(From 5-Stage MIPS to Powerful MIPS for PC)

(Optional)

Section 4.10

Instruction-Level Parallelism (ILP)

- Inside your PC

(Problem solving to develop powerful machines)

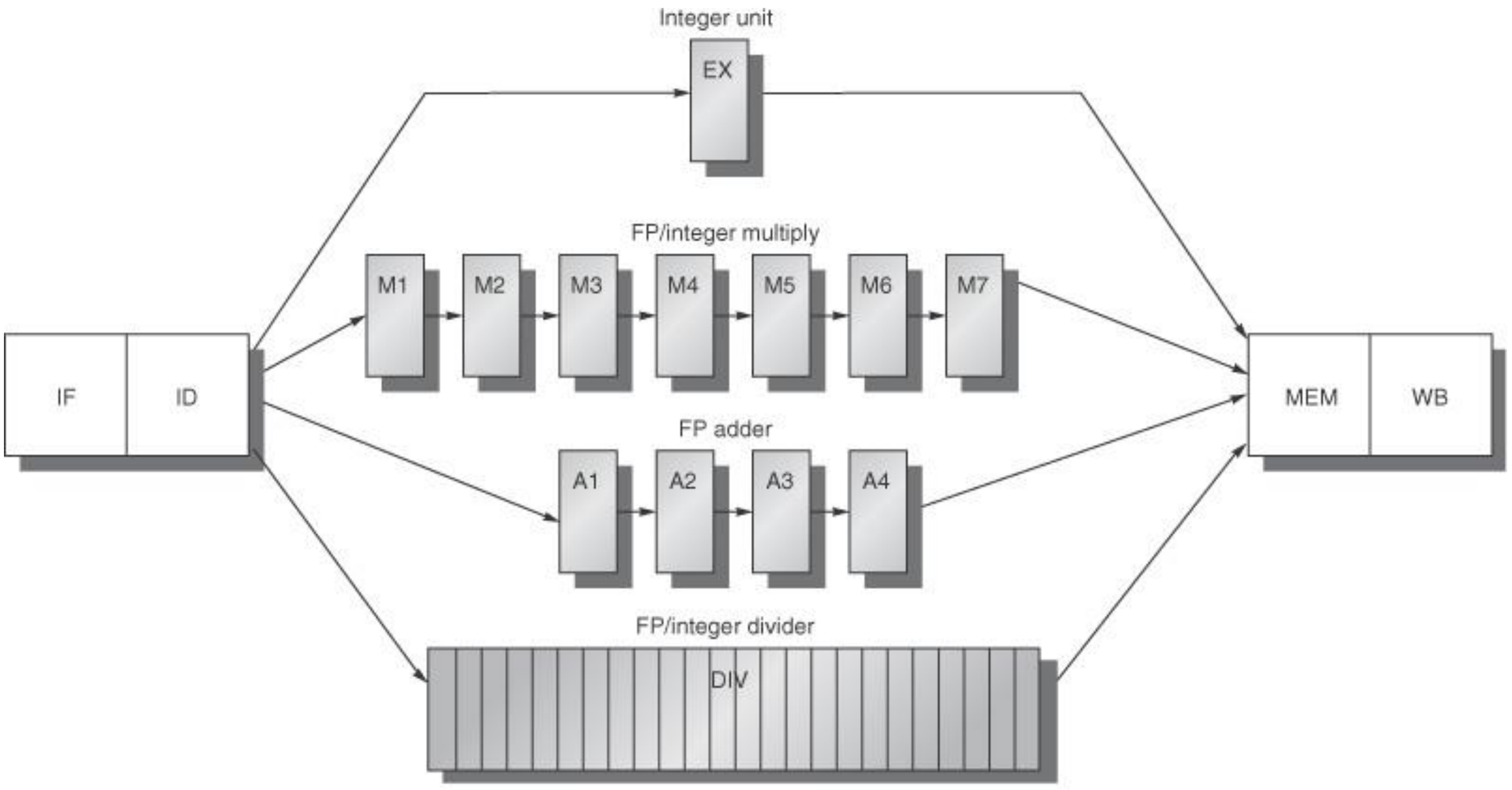
Some of authors' slides are modified

Key Concepts (다음의 용어를 이해)

- ❑ Floating-point pipelines
- ❑ Deep pipelines
- ❑ Multiple issue
  - Static (c.f., VLIW)
  - Dynamic (c.f., superscaler)
- ❑ Loop unrolling
- ❑ Speculation

❖ Computer Architecture: A Quantitative Approach, Hennessy and Patterson

Floating-Point Pipelines

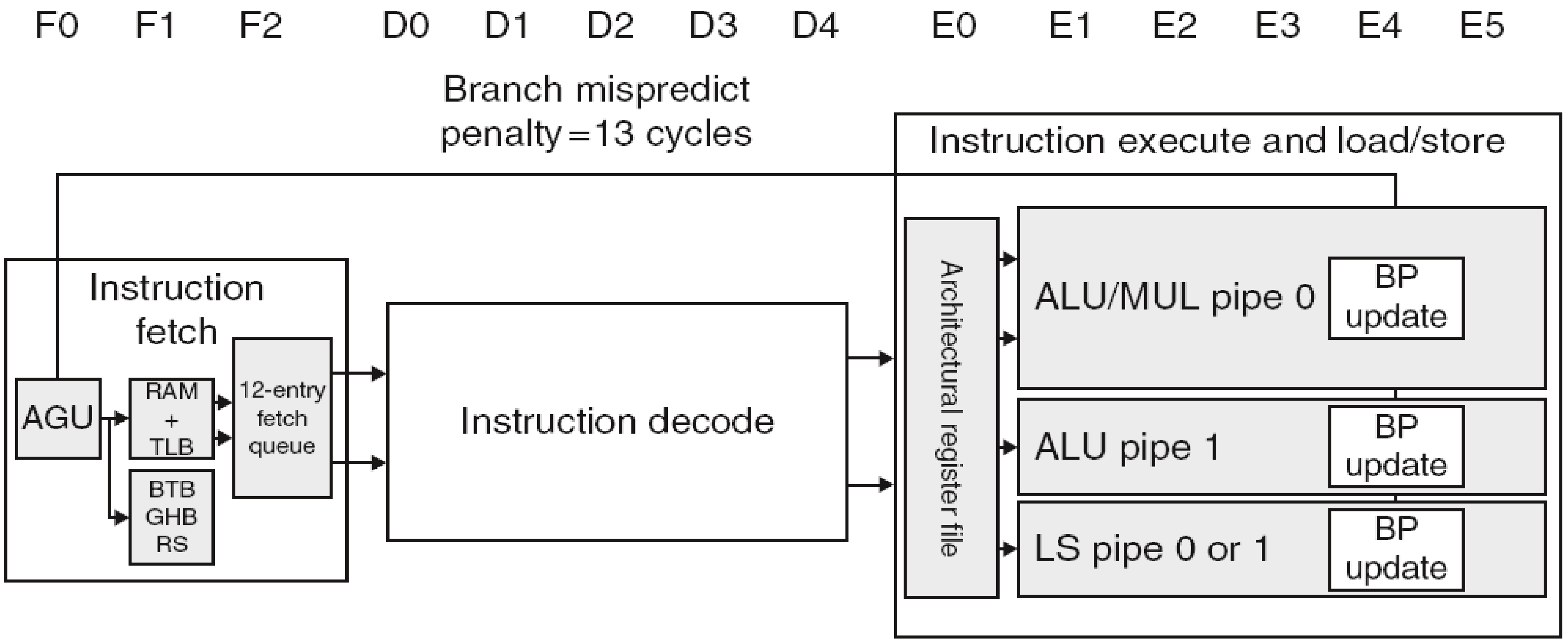


- Sometimes, better not to pipeline
  - Return on investment (e.g., divider)

Instruction-Level Parallelism (ILP)

- Pipelining: execute multiple instructions in parallel
- To increase ILP
  - Deeper pipeline (CPI = 1)
    - Less work per stage ⇒ shorter clock cycle
  - Multiple issue
    - Replicate pipeline stages ⇒ multiple pipelines
    - Start multiple instructions per clock cycle
      - † CPI = 0.25 with 4GHz 4-way multiple-issue
    - But dependencies reduce this in practice

ARM Cortex-A8 Pipeline (참고)



Key Concepts (다음의 용어를 이해)

- ❑ Floating-point pipelines
- ❑ Deep pipelines
- ❑ Multiple issue
  - Static (c.f., VLIW)
  - Dynamic (c.f., superscaler)
- ❑ Loop unrolling
- ❑ Speculation

Static Multiple Issue

- Compiler groups instructions that can be issued on a single cycle (“issue packets”)
- Compiler must remove some/all hazards
  - Reorder instructions; pad with “nop” if necessary

Instruction	Instruction	nop	nop
instruction	Instruction	Instruction	nop

⋮

- Think of an issue packet as a very long instruction
  - Specifies multiple concurrent operations

    ⇒ Very Long Instruction Word (VLIW)

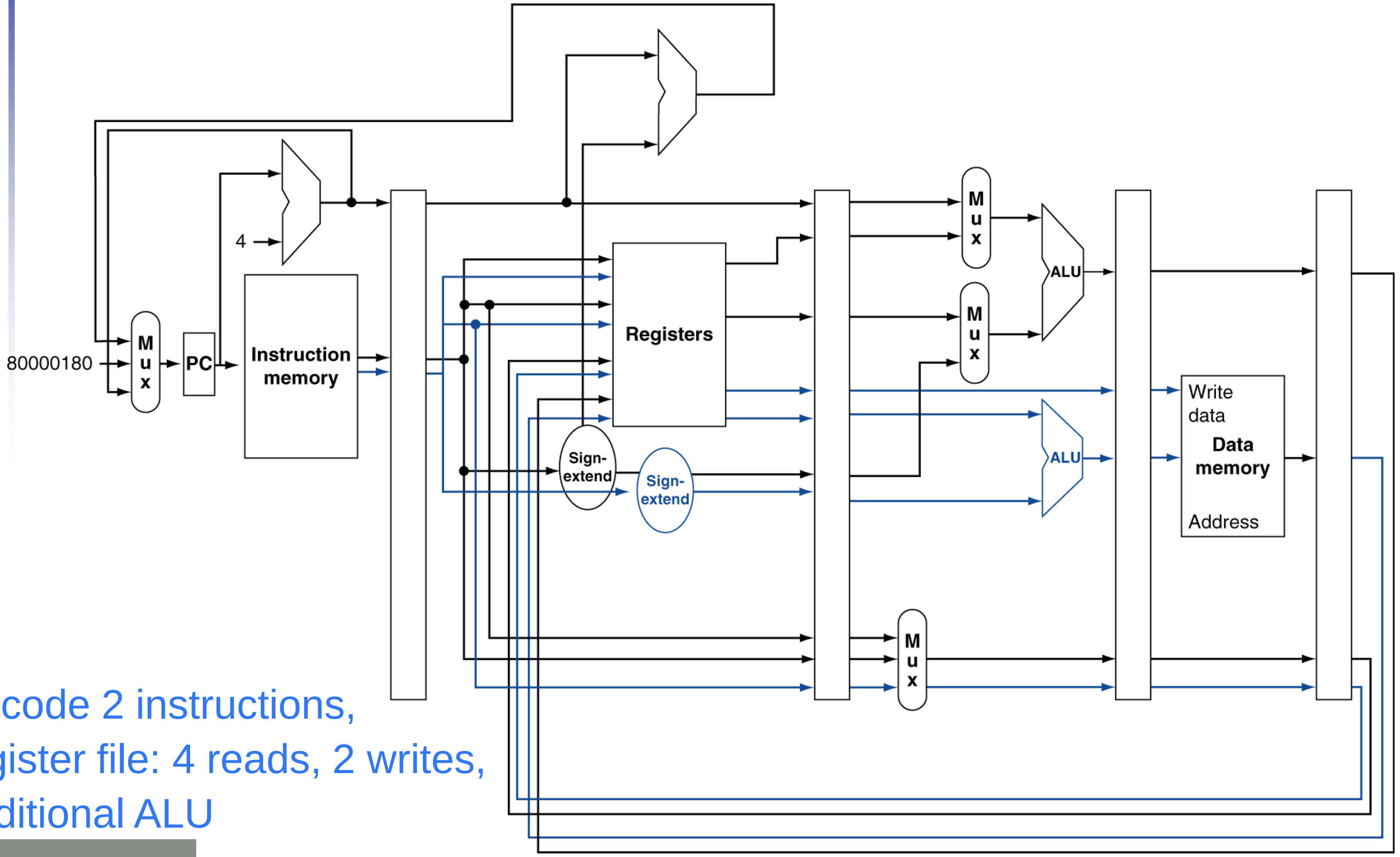
MIPS with Static Dual Issue

- Two-issue packets
  - One ALU/branch instruction + one load/store instruction
  - 64-bit aligned (pad unused instruction with “nop”)

Address	Instruction type	Pipeline Stages						
n	ALU/branch	IF	ID	EX	MEM	WB		
n + 4	Load/store	IF	ID	EX	MEM	WB		
n + 8	ALU/branch		IF	ID	EX	MEM	WB	
n + 12	Load/store		IF	ID	EX	MEM	WB	
n + 16	ALU/branch			IF	ID	EX	MEM	WB
n + 20	Load/store			IF	ID	EX	MEM	WB

- Ideal CPI = 0.5 (IPC = 2)

MIPS with Static Dual Issue



Decode 2 instructions, register file: 4 reads, 2 writes, additional ALU

Hazards in the Dual-Issue MIPS

(참고)

- More instructions executing in parallel
- Load-use hazard
  - One cycle use latency, but now two instructions
- EX data hazard
  - Can't use ALU result in load/store in same packet

```
addu $t0, $t0, $s2
sw   $t0, 4($s1)
```

    - Split into two packets, effectively a stall
- More aggressive scheduling required

Scheduling Example

Loop: a[i] = a[i] + k

```
Loop: lw   $t0, 0($s1)      # $t0=array element
      addu $t0, $t0, $s2    # add scalar in $s2
      sw   $t0, 0($s1)      # store result
      addi $s1, $s1,–4      # decrement pointer
      bne  $s1, $zero, Loop # branch $s1!=0
```

	ALU/branch	Load/store	cycle
Loop:	nop	lw $t0, 0($s1)	1
	addi $s1, $s1,–4	nop	2
	addu $t0, $t0, $s2	nop	3
	bne $s1, $zero, Loop	sw $t0, 4($s1)	4

- IPC = 5/4 = 1.25 (c.f. peak IPC = 2)

Loop Unrolling

- Replicate loop body to expose more parallelism
  - Reduces loop-control overhead
    - By compilers or programmers
- Use different registers per replication
  - Called “register renaming” (dependence 발생 방지)
    - Iteration 1: use $t0
    - Iteration 2: use $t1
    - Iteration 3: use $t2
    - Iteration 4: use $t3

Loop Unrolling

- Strip-mining or loop sectioning
  - Fragmenting a large loop into sections or strips

```
for (i = 0; i < 10000;  i++)
          a[i] += k;
```

⇨

```
 for (i = 0; i < 10000;  i += 4)
{       a[i] += k;
        a[i+1] += k;
        a[i+2] += k;
        a[i+3] += k;
}
 // remaining iterations if any
```

Loop Unrolling Example

	ALU/branch	Load/store	cycle
`Loop:`	`addi $s1, $s1,–16`	`lw $t0, 0($s1)`	1
	nop	`lw $t1, 12($s1)`	2
	`addu $t0, $t0, $s2`	`lw $t2, 8($s1)`	3
	`addu $t1, $t1, $s2`	`lw $t3, 4($s1)`	4
	`addu $t2, $t2, $s2`	`sw $t0, 16($s1)`	5
	`addu $t3, $t4, $s2`	`sw $t1, 12($s1)`	6
	nop	`sw $t2, 8($s1)`	7
	`bne $s1, $zero, Loop`	`sw $t3, 4($s1)`	8

- IPC = 14/8 = 1.75
  - Closer to 2, but at cost of registers and code size

Key Concepts (다음의 용어를 이해)

- ❑ Floating-point pipelines
- ❑ Deep pipelines
- ❑ Multiple issue
  - Static (c.f., VLIW)
  - Dynamic (c.f., superscaler)
- ❑ Loop unrolling
- ❑ Speculation

Dynamic Multiple Issue

- "Superscalar" processors (why the name?)
- CPU decides whether to issue 0, 1, 2, … instructions each cycle (at runtime)
  - Avoiding structural and data hazards
- Avoids the need for compiler scheduling
  - Though it may still help
  - Code semantics ensured by the CPU

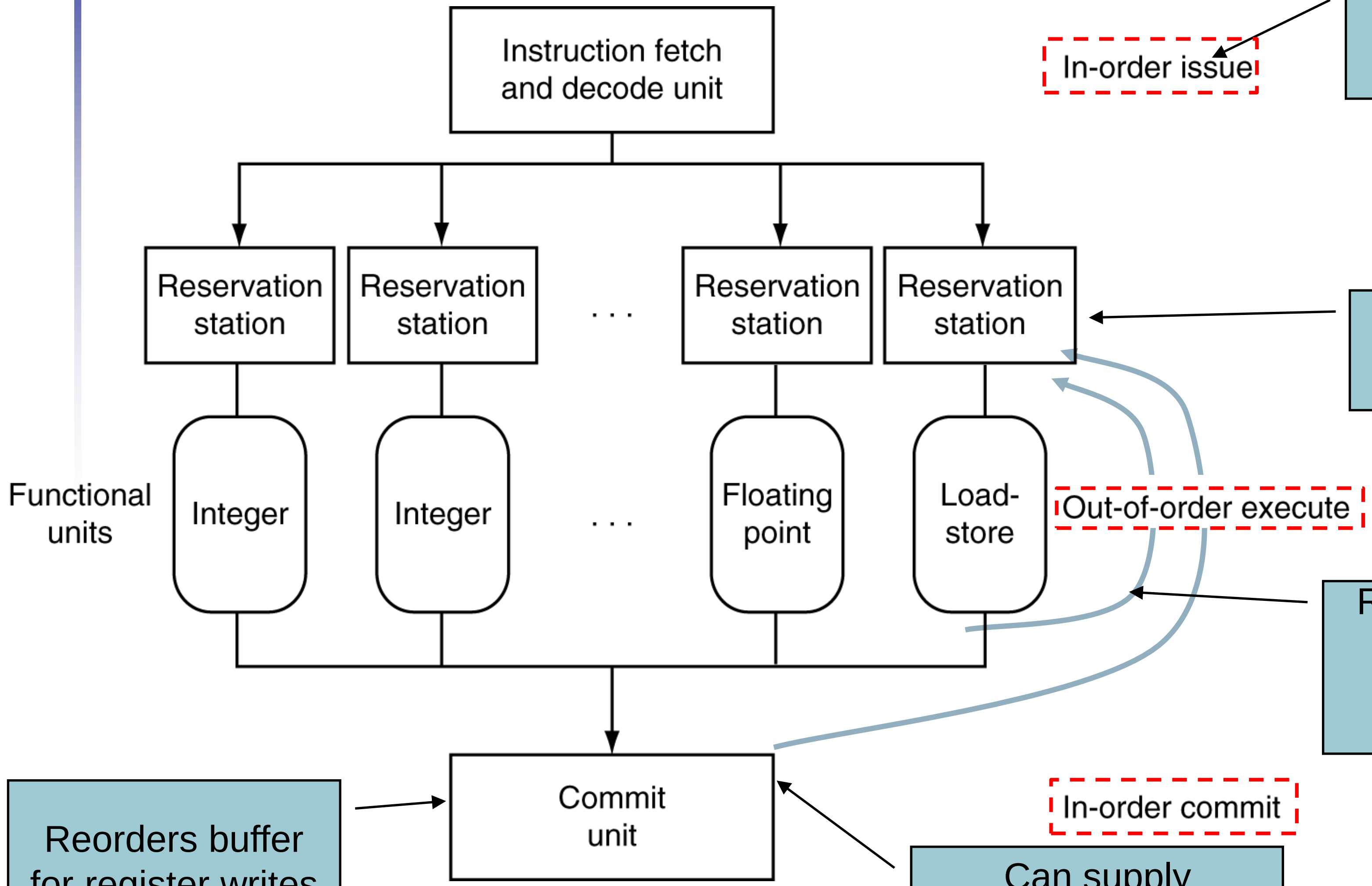
Dynamic Pipeline Scheduling

- CPU executes instructions out of order to avoid stalls
  - But commit result to registers in order
- Example

```
lw    $t0, 20($s2)
addu  $t1, $t0, $t2
sub   $s4, $s4, $t3
slti  $t5, $s4, 20
```

  - Can start **sub** while **addu** is waiting for **lw**
    - But **sub** doesn't commit until **lw** and **addu** commit

Dynamic Pipeline Scheduling



Preserves dependencies

Hold pending operands

Results also sent to any waiting reservation stations

Reorders buffer for register writes

Can supply operands for issued instructions

Register Renaming (참고)

- Reservation stations and reorder buffer effectively provide register renaming
- On instruction issue to reservation station
  - If operand available in register file or reorder buffer
    - Copied to reservation station
  - If operand is not yet available
    - Will be provided to reservation station by a FU
    - Name of the functional unit is tracked

Why Do Dynamic Scheduling?

- Why not just let the compiler schedule code?
  - Not all stalls are predicable
    - e.g., cache misses
  - Can’t always schedule around branches
    - Branch outcome is dynamically determined
  - Different implementations of an ISA have different latencies and hazards
    - No compiler dependence

Does Multiple Issue Work?

- Yes, but not as much as we'd like
  - Programs have real dependencies that limit ILP
  - Some dependencies are hard to eliminate
    - e.g., pointer aliasing
  - Some parallelism is hard to expose
    - Limited window size during instruction issue
  - Memory delays and limited bandwidth
    - Hard to keep pipelines full
  - Speculation can help if done well

Power Efficiency

- Complexity of dynamic scheduling and speculations requires power
- Multiple simpler cores may be better

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue width	Out-of-order/ Speculation	Core s	Power
i486	1989	25MHz	5	1	No	1	5W
Pentium	1993	66MHz	5	2	No	1	10W
Pentium Pro	1997	200MHz	10	3	Yes	1	29W
P4 Willamette	2001	2000MHz	22	3	Yes	1	75W
P4 Prescott	2004	3600MHz	31	3	Yes	1	103W
Core	2006	2930MHz	14	4	Yes	2	75W
UltraSparc III	2003	1950MHz	14	4	No	1	90W
UltraSparc T1	2005	1200MHz	6	1	No	8	70W

Key Concepts (다음의 용어를 이해)

- ❑ Floating-point pipelines
- ❑ Deep pipelines
- ❑ Multiple issue
  - Static (c.f., VLIW)
  - Dynamic (c.f., superscaler)
- ❑ Loop unrolling
- ❑ Speculation

Speculation

- "Guess" what to do with an instruction
  - Start operation as soon as possible
  - Check whether guess was right
    - If so, complete the operation
    - If not, roll-back and do the right thing
- Common to static and dynamic multiple issue
- Speculate on branch outcome

  Roll back if path taken is different
- Speculate on load
  - Roll back if location is updated

Class Topics (클래스 홈페이지 참조)

- ❑ Part 1: Fundamental concepts and principles
- ❑ Part 2: 빠른 컴퓨터를 위한 ISA design
  - Topic 1 Computer performance and ISA design (Ch. 1)
  - Topic 2 RISC (MIPS) instruction set (Chapter 2)
  - Topic 3 Computer arithmetic and ALU (Chapter 3)
- ❑ Part 3: ISA 의 효율적인 구현 (pipelining, cache memory)
  - Topic 4 Processor design (Chapter 4)
  - Topic 5 Memory system design (Chapter 5)
    - Cache memory
    - Cache memory and virtual memory

Section 4.9

Exceptions (skip)

Exceptions and Interrupts

- "Unexpected" events requiring change in flow of control
  - Different ISAs use the terms differently
- Exception
  - Arises within the CPU
    - e.g., undefined opcode, overflow, syscall, …
- Interrupt
  - From an external I/O controller
- Dealing with them without sacrificing performance is hard

Section 4.11

Real Stuff: The ARM Cortex-A8 and Intel Core i7 Pipelines

(skip)

ARM Cortex A8 and Intel i7

Processor	ARM A8	Intel Core i7 920
Market	Personal Mobile Device	Server, cloud
Thermal design power	2 Watts	130 Watts
Clock rate	1 GHz	2.66 GHz
Cores/Chip	1	4
Floating point?	No	Yes
Multiple issue?	Dynamic	Dynamic
Peak instructions/clock cycle	2	4
Pipeline stages	14	14
Pipeline schedule	Static in-order	Dynamic out-of-order with speculation
Branch prediction	2-level	2-level
1st level caches/core	32 KiB I, 32 KiB D	32 KiB I, 32 KiB D
2nd level caches/core	128-1024 KiB	256 KiB
3rd level caches (shared)	-	2- 8 MB

Section 4.12

Instruction-Level Parallelism and Matrix Multiply

(skip)