Computer Systems II

Li Lu
Room 605, CaoGuangbiao Building
li.lu@zju.edu.cn
https://person.zju.edu.cn/lynnluli

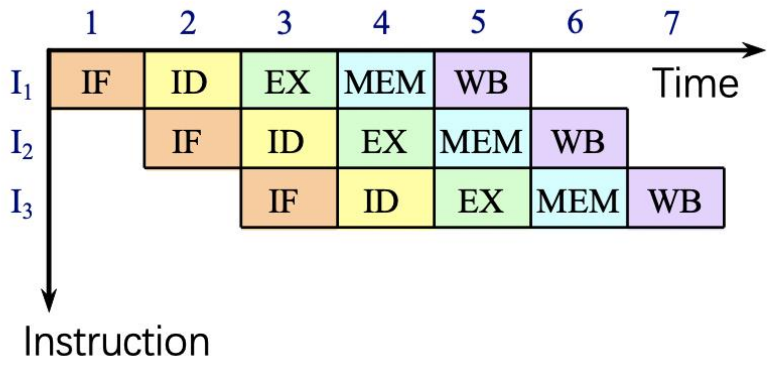
Multiple Issue

Instruction-Level Parallelism (ILP)

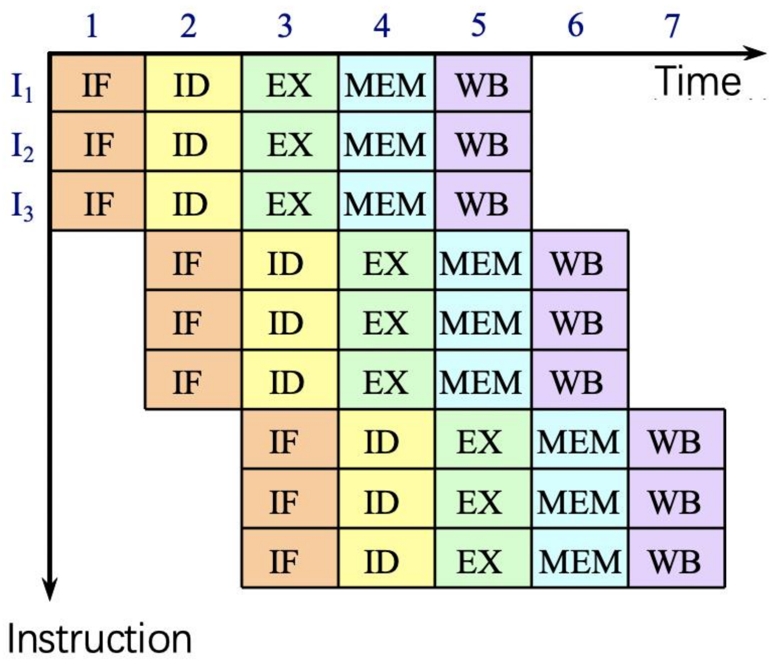
- Pipelining: executing multiple instructions in parallel
- To increase ILP
  - Deeper pipeline
    - Less work per stage → shorter clock cycle
  - Multiple issue
    - Replicate pipeline stages → multiple pipelines
    - Start multiple instructions per clock cycle
    - CPI < 1, so use Instructions Per Cycle (IPC)
    - E.g., 4GHz 4-way multiple-issue
      - 16 BIPS, peak CPI = 0.25, peak IPC = 4
  - But dependencies reduce this in practice

Comparison of the spatiotemporal diagrams of instructions executed by single-issue and multiple-issue processors

Single-issue spatiotemporal diagram



Multiple-issue spatiotemporal diagram



Multiple Issue

- Static multiple issue
  - **Compiler** groups instructions to be issued together
  - Packages them into “issue slots”
  - Compiler detects and avoids hazards
- Dynamic multiple issue
  - **CPU** examines instruction stream and chooses instructions to issue each cycle
  - Compiler can help by reordering instructions
  - CPU resolves hazards using advanced techniques at runtime

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

Compiler/Hardware Speculation

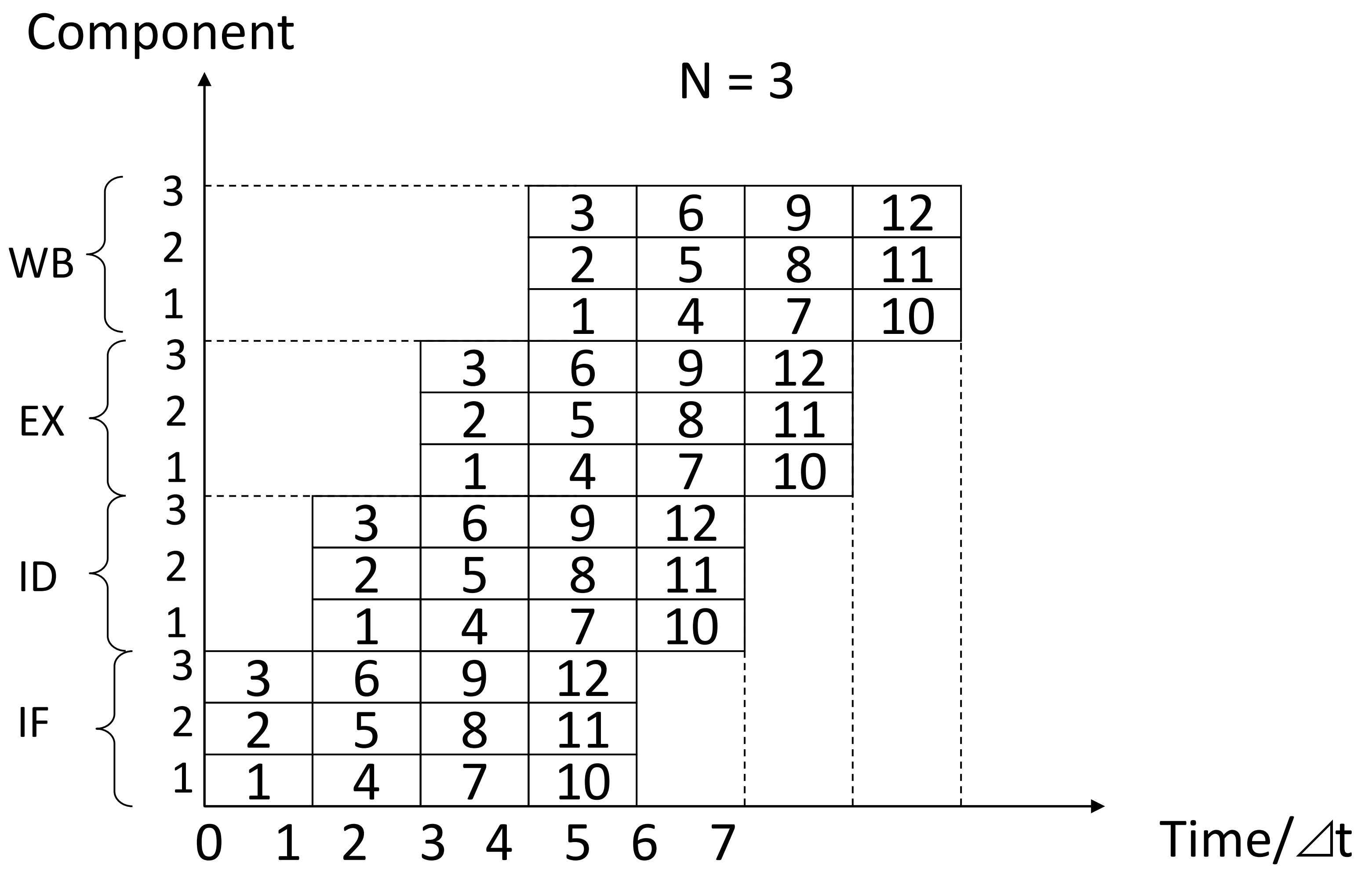
- Compiler can reorder instructions
  - e.g., move load before branch
  - Can include “fix-up” instructions to recover from incorrect guess
- Hardware can look ahead for instructions to execute
  - Buffer results until it determines they are actually needed
  - Flush buffers on incorrect speculation

Two types of multiple-issue processor

Superscalar

- The number of instructions which are issued in each clock cycle is not fixed. It depends on the specific circumstances of the code. (1-8, with upper limit)
- Suppose this upper limit is n, then the processor is called n-issue.
- It can be statically scheduled through the compiler, or dynamically scheduled based on Tomasulo algorithm.
- This method is the most successful method for general computing at present.

Two types of multiple-issue processor



Two types of multiple-issue processor

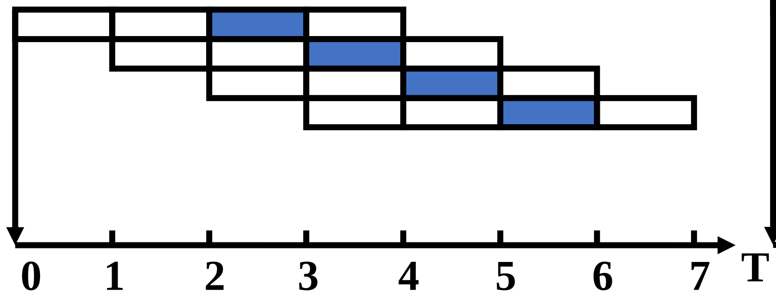
VLIW (Very Long Instruction Word)

- The number of instructions which are issued in each clock cycle is fixed (4-16), and these instructions constitute a long instruction or an instruction packet.
- In the instruction packet, the parallelism between instructions is explicitly expressed through instructions.
- Instruction scheduling is done statically by the compiler.
- It has been successfully applied to digital signal processing and multimedia applications.

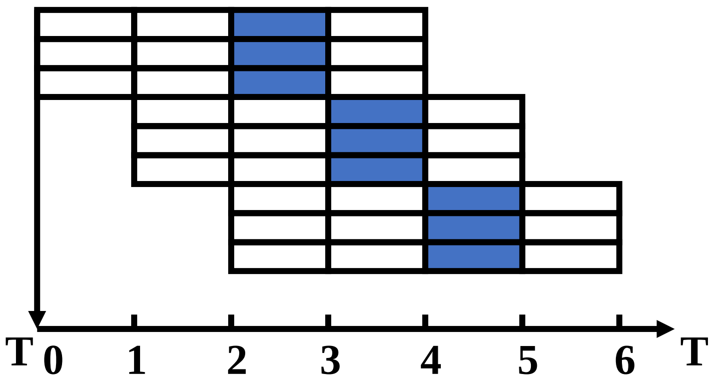
Superscalar & VLIW

- The superscalar structure is transparent to the programmer, because the processor can detect whether the next instruction can flow out, so there is no need to rearrange instructions to satisfy the issue of instructions.
- Even the code that has not been optimized by the compiler for scheduling and optimization of the superscalar structure or the code generated by the old compiler can run, of course, the running effect will not be very good.
- To achieve good results, one of the methods:
  - Use dynamic superscalar scheduling technology.

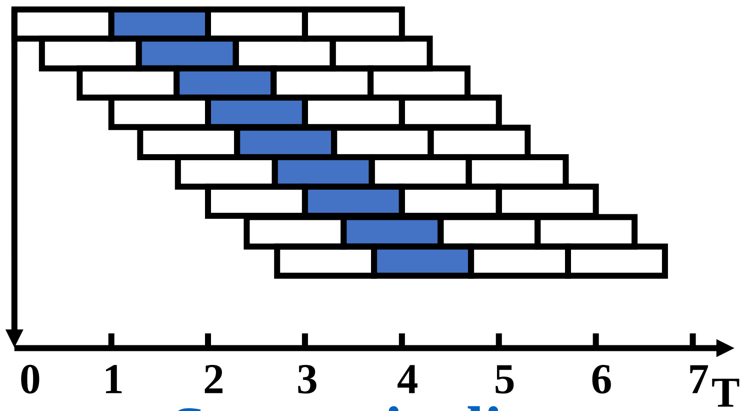
Superscalar & VLIW



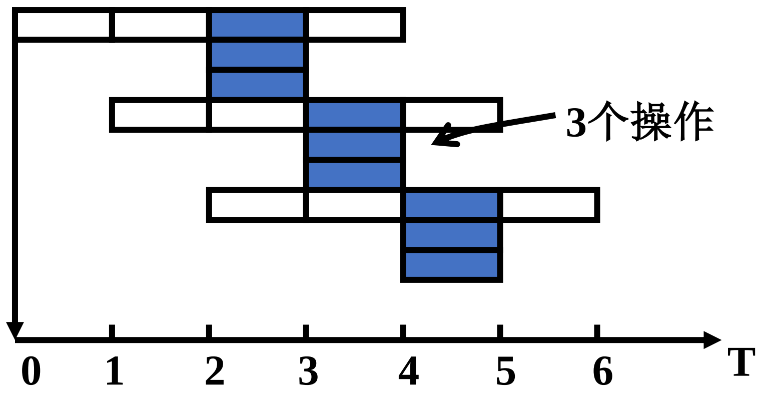
Normal Pipeline



Superscalar



Super pipeline



VLIW

§5.2 Multiple Issue

Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	Dynamic	Hardware	Static	In-order execution	Mostly in the embedded space: MIPS and ARM, including the ARM Cortex-A8
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	Some out-of-order execution, but no speculation	None at the present
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Out-of-order execution with speculation	Intel Core i3, i5, i7; AMD Phenom; IBM Power 7
VLIW/LIW	Static	Primarily software	Static	All hazards determined and indicated by compiler (often implicitly)	Most examples are in signal processing, such as the TI C6x
EPIC	Primarily static	Primarily software	Mostly static	All hazards determined and indicated explicitly by the compiler	Itanium

Multi-issue technology based on static scheduling

- In a typical superscalar processor, 1 to 8 instructions can be issued per clock cycle.
- Instructions flow out in order, and conflict detection is performed when they flow out.
  - In the current sequence of instructions, there is no data conflict or Close conflict.

Example: A statically scheduled superscalar processor with 4 issues

- In the instruction fetch stage, the pipeline will receive 1 to 4 instructions (called issue packets) from the instruction fetch component.
- In one clock cycle, all of these instructions may be able to flow out, or only a part of them may flow out.

Multi-issue technology based on static scheduling

The outgoing component detects structural hazards or data hazards.

- Generally implemented in two stages:
  - The first stage: Carry out the hazard detection in the outgoing package, and select the instructions that can be outflowed initially.
  - The second stage: Check whether the selected instruction conflicts with the instruction being executed.

How does the RISC-V processor achieve superscalar?

- Assumption: Two instructions flow out every clock cycle:
  - 1 integer instruction + 1 floating-point operation instruction
- Among them, load instructions, store instructions, and branch instructions are classified as integer instructions.

Multi-issue technology based on static scheduling

Claim:

- Fetch two instructions (64 bits) at the same time and decode two instructions (64 bits)

- The processing of instructions includes the following steps:
    - Fetch two instructions from memory
    - Determine which instructions can flow out (0~2 instructions)
    - Send them to the corresponding functional components
- The execution process of instructions in a multiple-issue superscalar pipeline
    - Assumption: all floating-point instructions are addition instructions, and their execution time is two clock cycles
    - For simplicity, integer instructions are always placed before floating-point instructions in the figure below

Type	Pipeline work bench							
Integer Instruction	IF	ID	EX	MEM	WB			
Floating-Point Instruction	IF	ID	EX	EX	MEM	WB		
Integer Instruction		IF	ID	EX	MEM	WB		
Floating-Point Instruction		IF	ID	EX	EX	MEM	WB	
Integer Instruction			IF	ID	EX	MEM	WB	
Floating-Point Instruction			IF	ID	EX	EX	MEM	WB
Integer Instruction				IF	ID	EX	MEM	WB
Floating-Point Instruction				IF	ID	EX	EX	MEM

Multi-issue technology based on static scheduling

- With the parallel outflow method of "1 integer instruction + 1 floating point instruction", the amount of hardware that needs to be increased is small
- Floating-point load or floating-point store instructions will use integer parts, which increases access conflicts to floating point registers
  - Add a read/write port for floating-point registers
- Since the number of instructions in the pipeline has doubled, the directional path has to be increased

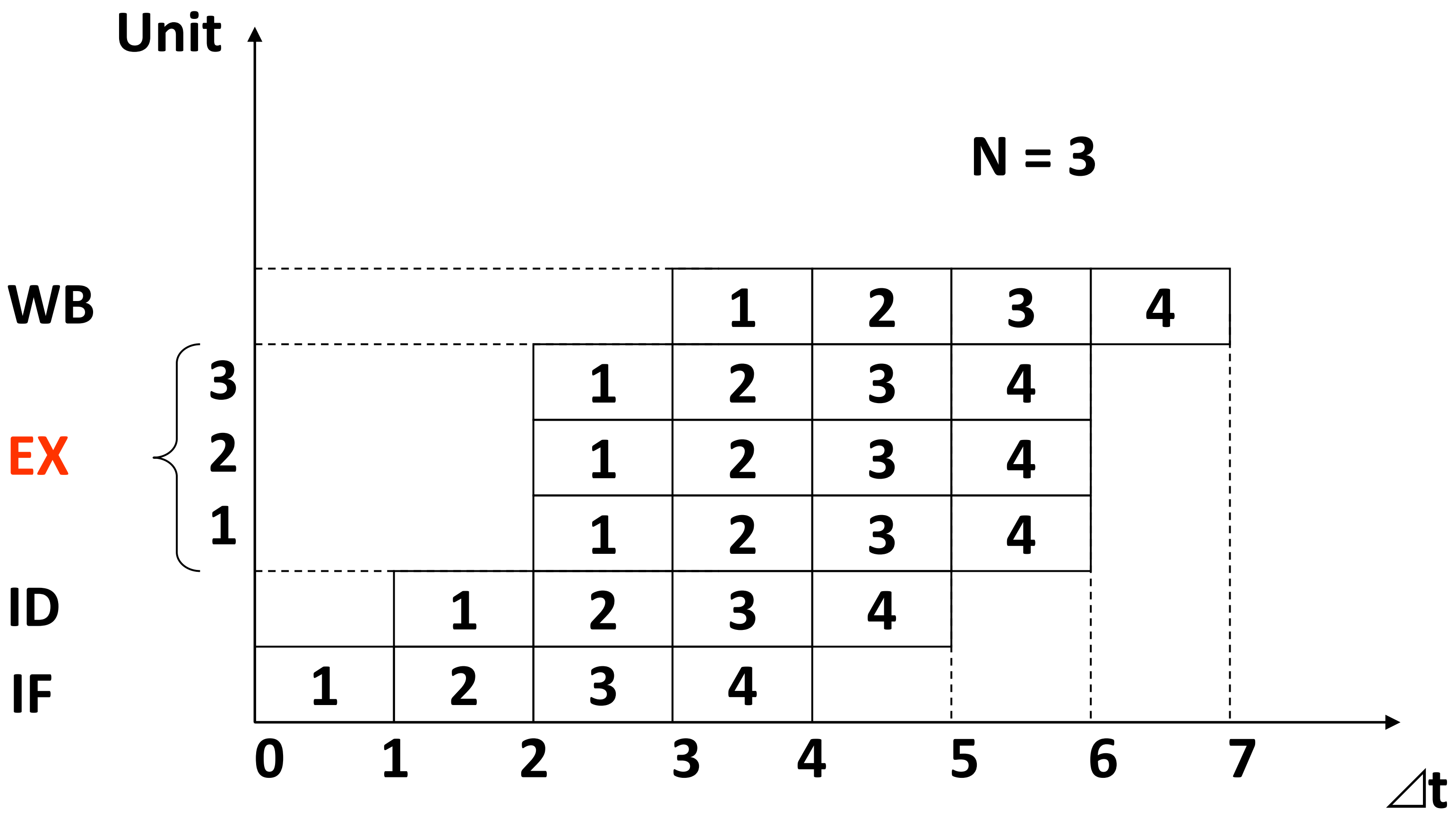
Static Multiple Issue

- Compiler groups instructions into “issue packets”
  - Group of instructions that can be issued on a single cycle
  - Determined by pipeline resources required
- Think of an issue packet as a very long instruction
  - Specifies multiple concurrent operations
  - → Very Long Instruction Word (VLIW)

Very long instruction word technology(VLIW)

- Assemble multiple instructions that can be executed in parallel into a very long instruction (more than 100 bits to hundreds of bits).
- Set up multiple features.
- The instruction word is divided into several fields, and each field is called an operation slot, which directly and independently controls a functional unit.
- In the VLIW processor, all processing and instruction arrangement are completed by the compiler.
- At compile time, multiple unrelated instructions or unrelated operations that can be executed in parallel are combined to form a very long instruction word with multiple operation segments.

VLIW



Very long instruction word technology(VLIW)

Example

- Assume that the VLIW processor can simultaneously stream 5 instructions per clock cycle: two memory access instructions, two floating-point operation instructions, and one integer instruction or branch instruction. Give its code sequence in the VLIW as follows.

	Instruction		Instruction		Instruction
1	ld f0, 0(x1)	7	add f8, f6, f2	13	add f16, f14, f2
2	add f4, f0, f2	8	sd f8, -8(x1)	14	sd f16, -24(x1)
3	sd f4, 0(x1)	9	ld f10, -16(x1)	15	ld f18, -32(x1)
4	addiu x1, x1, -40	10	add f12, f10, f2	16	add f20, f18, f2
5	bne x1, x2, Loop	11	sd f12, -12(x1)	17	sd f20, -32(x1)
6	ld f6, -8(x1)	12	ld f14, -24(x1)		

Very long instruction word technology(VLIW)

Solution: The code sequence is shown below.

- The running time is 8 clock cycles.
- An average of 1.6 clock cycles per stage.
- 17 instructions issued in 8 clock cycles, 2.1 instructions per clock cycle.
- There are 8 * 5 = 40 operating slots in 8 clock cycles, and the ratio of effective slots is 42.5%.

Clock Cycle	L/S Instruction1	L/S Instruction2	FP Instruction1	FP Instruction2	Integer/branch Instruction
1	ld f0,0(x1)	ld f6,-8(x1)	nop	nop	nop
2	ld f10,-16(x1)	ld f14,-24(x1)	nop	nop	nop
3	ld f18,-32(x1)	nop	add f4,f0,f2	add f8,f6,f2	nop
4	nop	nop	add f12,f10,f2	add f16,f14,f2	nop
5	nop	nop	add f20,f18,f2	nop	nop
6	sd f4,0(x1)	sd f8,-8(x1)	nop	nop	nop
7	sd f12,-16(x1)	sd f16,–24(x1)	nop	nop	addiu x1,x1,-40
8	sd f20,-32(x1)	nop	nop	nop	bne x1,x2,Loop

Very long instruction word technology(VLIW)

Some problems with VLIW

- Program code **length increased**
    - A large number of loop unrolling to improve parallelism.
    - The operation slot in the instruction word cannot always be filled.
    - Solution: Use the method of command sharing the immediate digital field, or use the method of command compression storage, transfer to Cache or expansion during decoding.
- **Lockstep** mechanism
    - When any operating part is paused, the entire processor must be paused.
- Machine code **incompatibility**

Very long instruction word technology(VLIW)

What are the limitations of the instruction multi-flow processor?

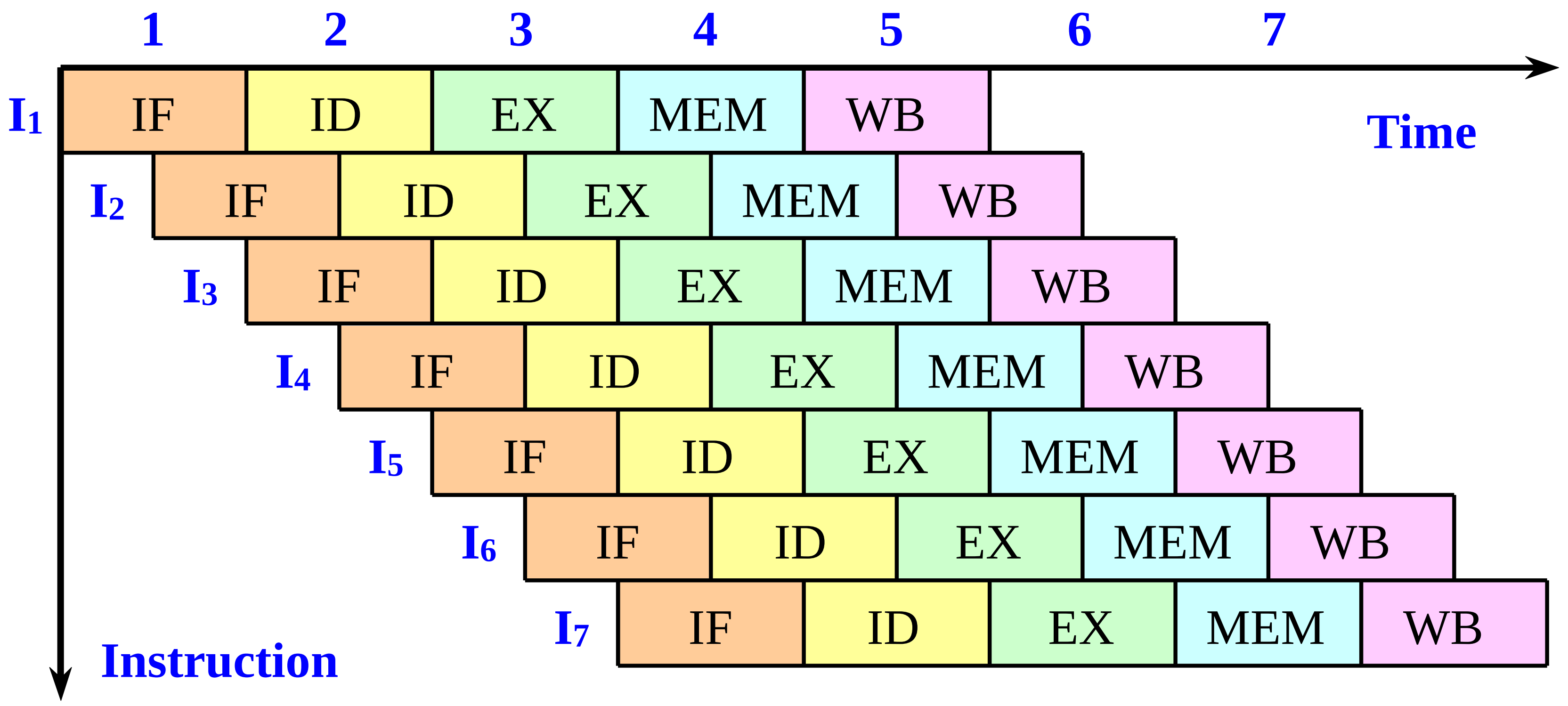
- Mainly affected by the following three aspects:
  - Instruction-level parallelism inherent in the program.
  - Difficulties in hardware implementation.
  - Technical limitations inherent in superscalar and super long instruction word processors.

Super-Pipeline

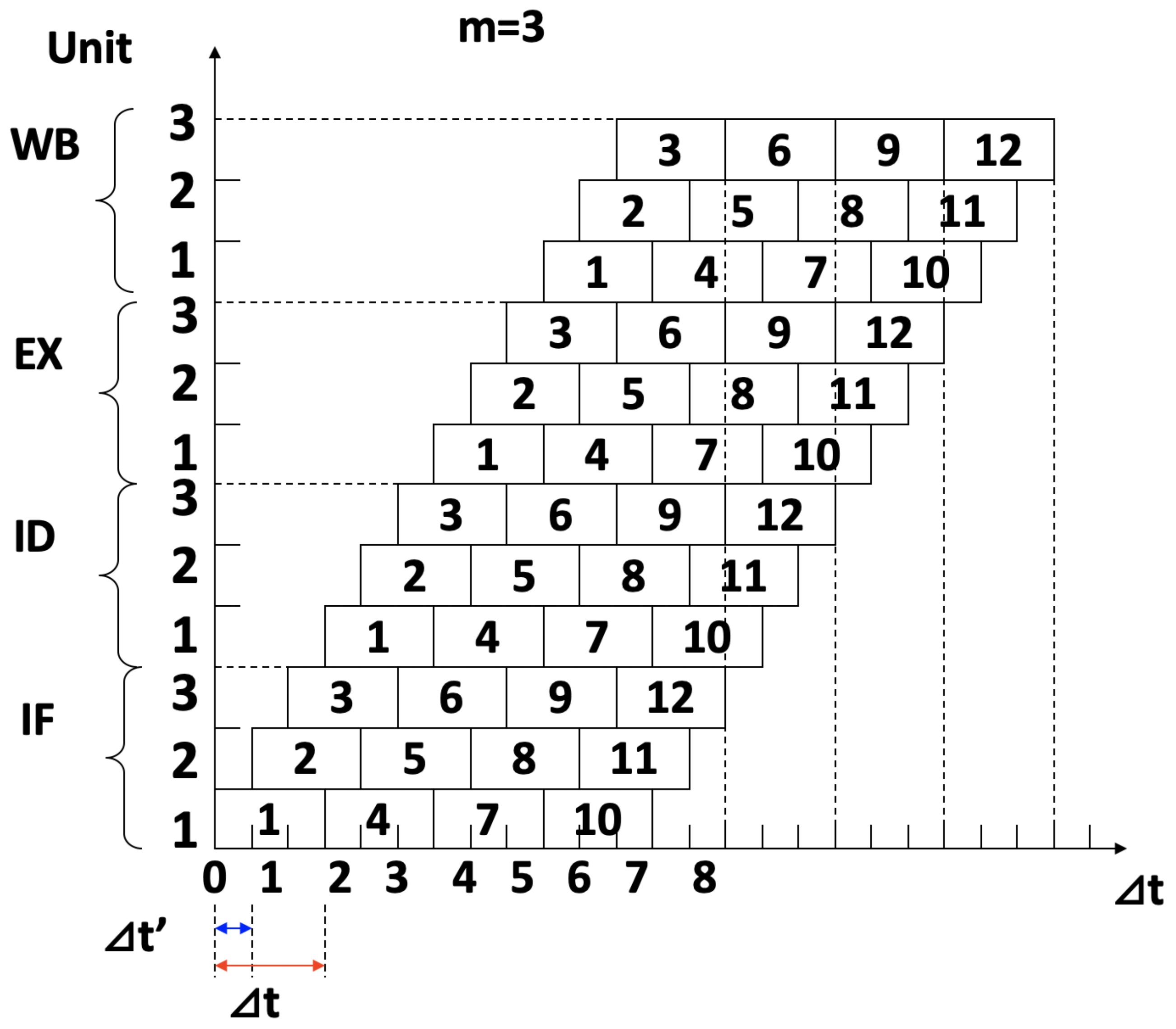
- Each pipeline stage is further subdivided, so that multiple instructions can be time-shared in one clock cycle. This kind of processor is called a super-pipelined processor.
- For a super-pipelined computer that can flow out *n* instructions per clock cycle, these *n* instructions are not flowed out at the same time, but one instruction is flowed out every 1/*n* clock cycle.
  - In fact, the pipeline cycle of the super-pipeline computer is 1/*n* clock cycles.

Super-Pipeline

The time-space diagram of a super-pipelined computer that issues two instructions in time-sharing every clock cycle.



§5.2 Multiple Issue



Scheduling Static Multiple Issue

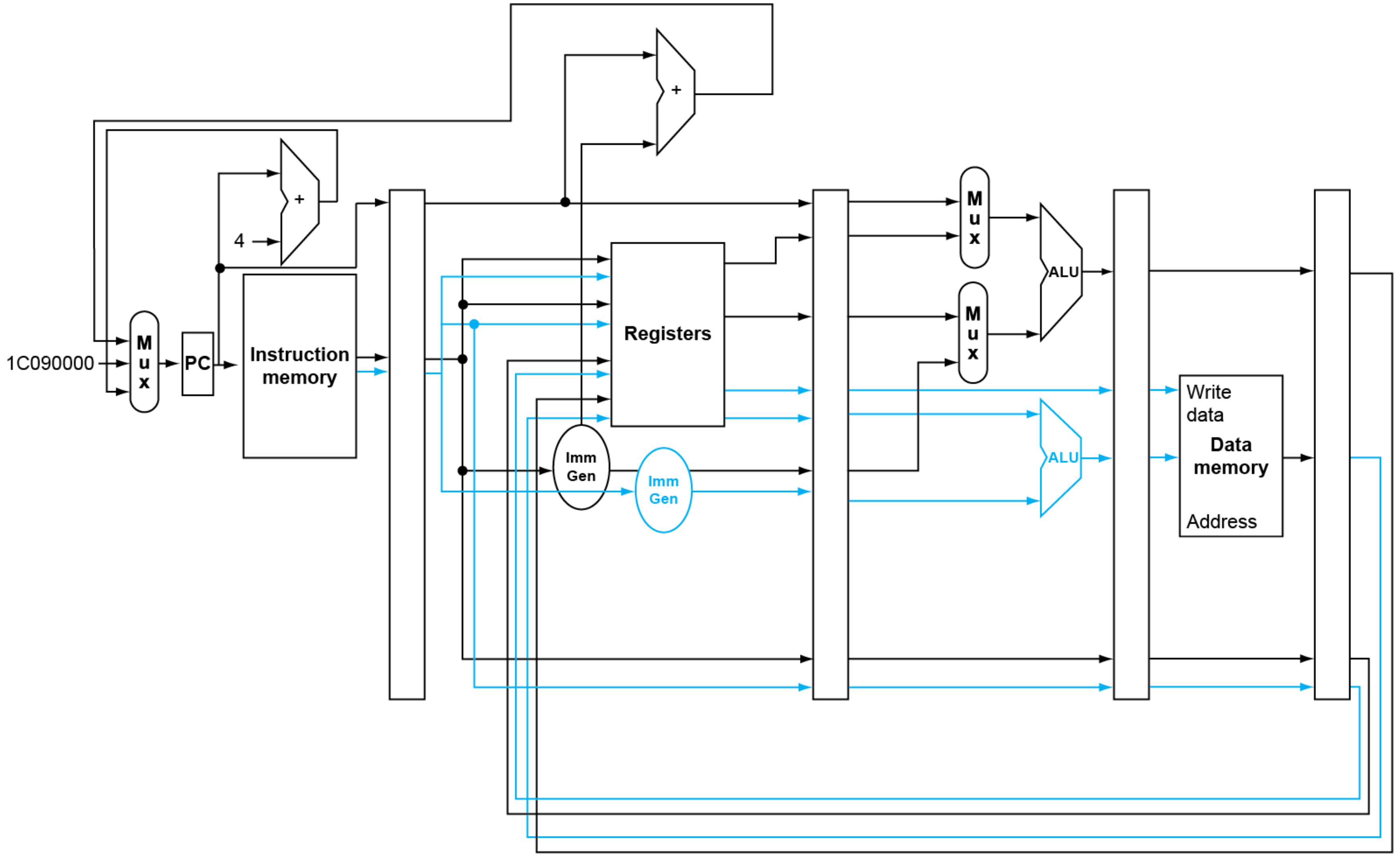
- Compiler must remove some/all hazards
  - Reorder instructions into issue packets
  - No dependencies with a packet
  - Possibly some dependencies between packets
    - Varies between ISAs; compiler must know!
  - Pad with nop if necessary

RISC-V with Static Dual Issue

- Two-issue packets
  - One ALU/branch instruction
  - One load/store instruction
  - 64-bit aligned
    - ALU/branch, then load/store
    - Pad an 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

RISC-V with Static Dual Issue



Hazards in Dual-Issue RISC-V

- More instructions executing in parallel
- EX data hazard
  - Forwarding avoided stalls with single-issue
  - Now can’t use ALU result in load/store in same packet

```
add  x10, x0, x1
ld   x2, 0(x10)
```

  - Split into two packets, effectively a stall
- Load-use hazard
  - Still one cycle use latency, but now two instructions
- More aggressive scheduling required

Scheduling Example

- Schedule this for dual-issue RISC-V

```
Loop: ld   x31,0(x20)     // x31=array element
      add  x31,x31,x21    // add scalar in x21
      sd   x31,0(x20)     // store result
      addi x20,x20,-8     // decrement pointer
      blt  x22,x20,Loop   // branch if x22 < x20
```

	ALU/branch	Load/store	cycle
Loop:	nop	ld x31,0(x20)	1
	addi x20,x20,-8	nop	2
	add x31,x31,x21	nop	3
	blt x22,x20,Loop	sd x31,8(x20)	4

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

Loop Unrolling

- Replicate loop body to expose more parallelism
  - Reduces loop-control overhead
- Use different registers per replication
  - Called “register renaming”
  - Avoid loop-carried “anti-dependencies”
    - Store followed by a load of the same register
    - Aka “name dependence”
      - Reuse of a register name

Loop Unrolling Example

	ALU/branch	Load/store	cycle
`Loop:`	`addi x20,x20,-32`	`ld x28, 0(x20)`	1
	nop	`ld x29, 24(x20)`	2
	`add x28,x28,x21`	`ld x30, 16(x20)`	3
	`add x29,x29,x21`	`ld x31, 8(x20)`	4
	`add x30,x30,x21`	`sd x28, 32(x20)`	5
	`add x31,x31,x21`	`sd x29, 24(x20)`	6
	nop	`sd x30, 16(x20)`	7
	`blt x22,x20,Loop`	`sd x31, 8(x20)`	8

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

Dynamic Multiple Issue

- “Superscalar” processors
- CPU decides whether to issue 0, 1, 2, … each cycle
  - Avoiding structural and data hazards
- Avoids the need for compiler scheduling
  - Though it may still help
  - Code semantics ensured by the CPU

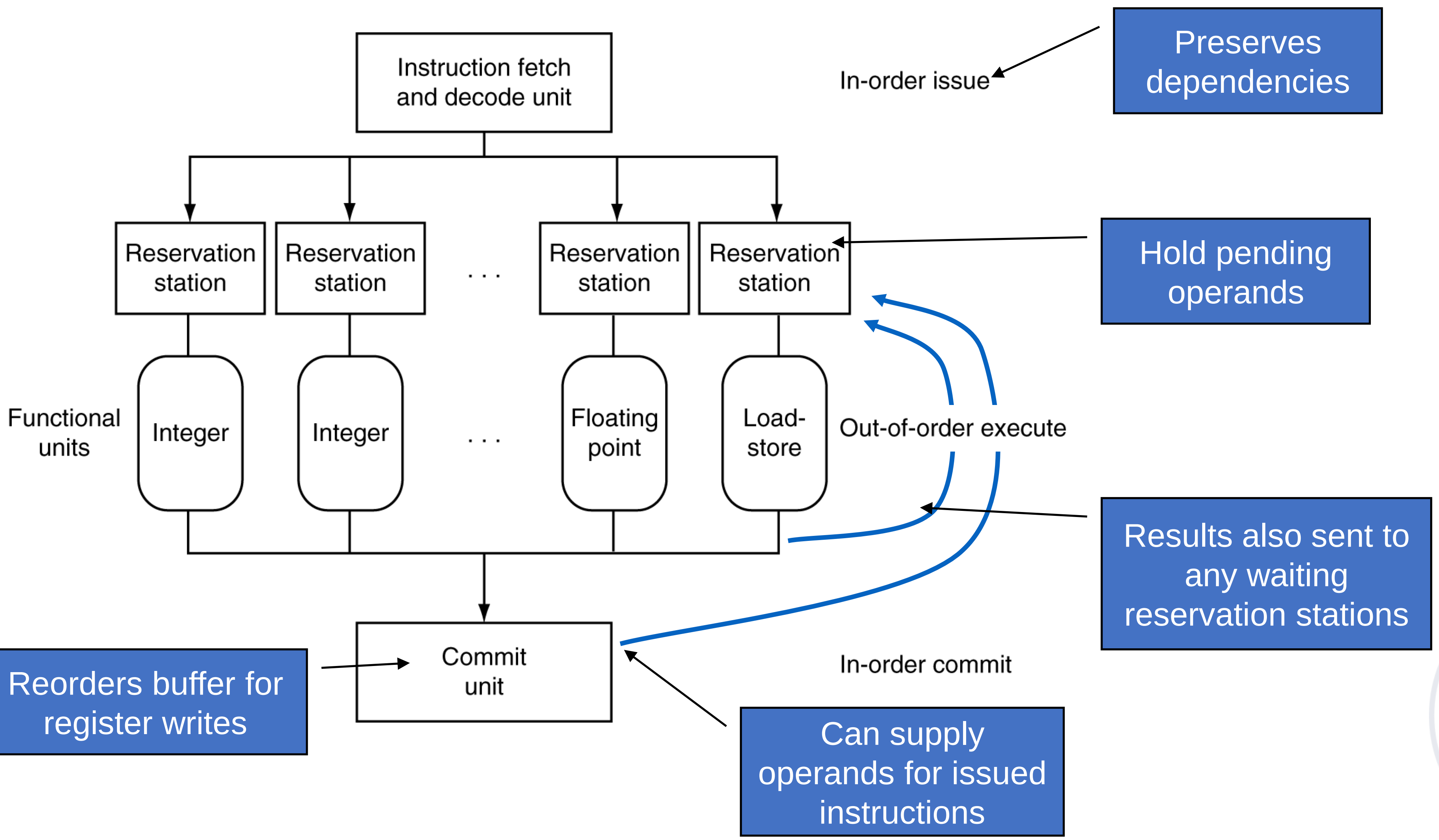
Dynamic Pipeline Scheduling

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

```
ld   x31,20(x21)
add  x1,x31,x2
sub  x23,x23,x3
andi x5,x23,20
```

  - Can start sub while add is waiting for ld

Dynamically Scheduled CPU



Register Renaming

- Reservation stations and reorder buffer effectively provide register renaming
- On instruction issue to reservation station
  - If operand is available in register file or reorder buffer
    - Copied to reservation station
    - No longer required in the register; can be overwritten
  - If operand is not yet available
    - It will be provided to the reservation station by a function unit
    - Register update may not be required

Load Speculation

- Predict branch and continue issuing
  - Don’t commit until branch outcome determined
- Load speculation
  - Avoid load and cache miss delay
    - Predict the effective address
    - Predict loaded value
    - Load before completing outstanding stores
    - Bypass stored values to load unit
  - Don’t commit load until speculation cleared

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

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	Cores	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