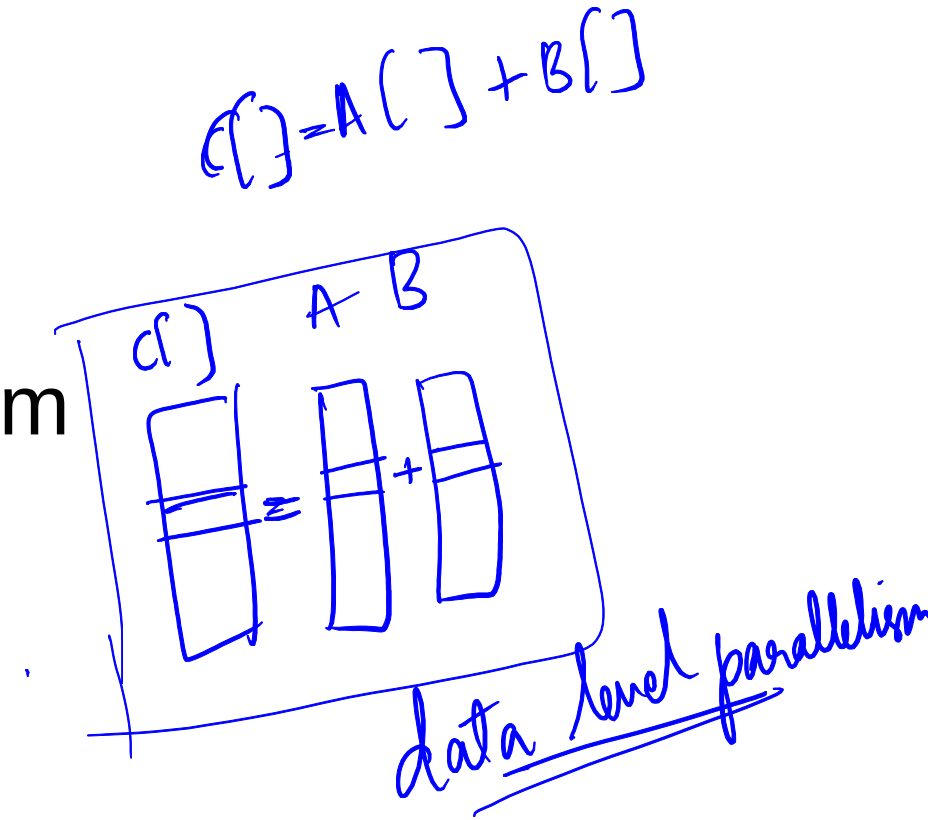
Parallelism via Instructions

Outline

- Pipeline, Pipelined datapath
- Dependences, Hazards
  - Structural, Data - Stalling, Forwarding
- Control Hazards
- Branch prediction
- ILP, Multiple Issue, Superscalar Processors

ILP

- Instruction Level Parallelism



CPI < 1.0

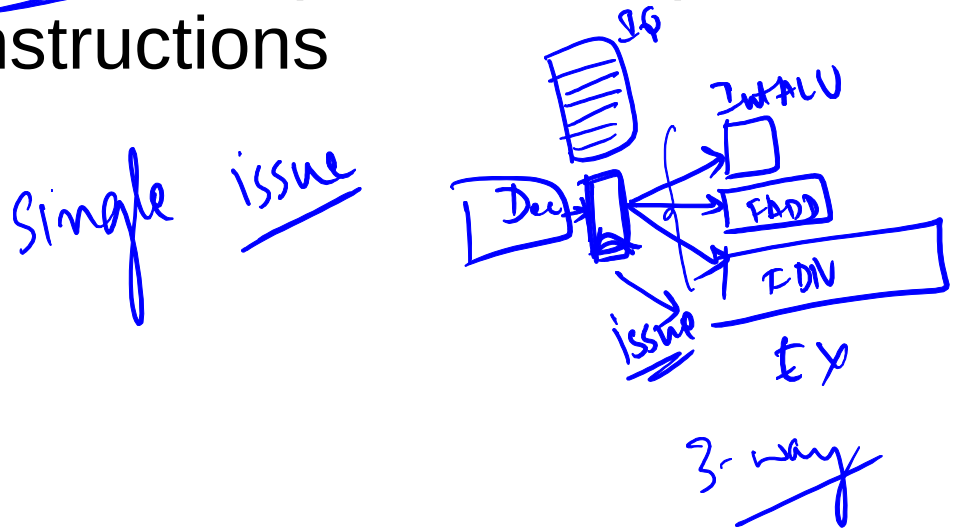
ILP

- Instruction Level Parallelism
  - Increasing the depth of the pipeline

K ↑

ILP

- Instruction Level Parallelism
  - Increasing the depth of the pipeline
  - Multiple Issue - Replicate components to launch multiple instructions



Multiple Issue Pipelines

Multiple Issue Pipelines

- 3 way – 6 way

Multiple Issue Pipelines

- 3 way – 6 way
- IPC > 1 (CPI < 1)

Multiple Issue Pipelines

- 3 way – 6 way
- IPC > 1 (CPI < 1)
- Static issue vs. Dynamic issue

Multiple Issue Pipelines

- 3 way – 6 way
- IPC > 1 (CPI < 1)
- Static issue vs. Dynamic issue
- Multiple Issue requires:
  - Packing instructions into **Issue Slots**
  - Dealing with data and control hazards

Speculation

Speculation

- Instructions after a predicted branch

Speculation

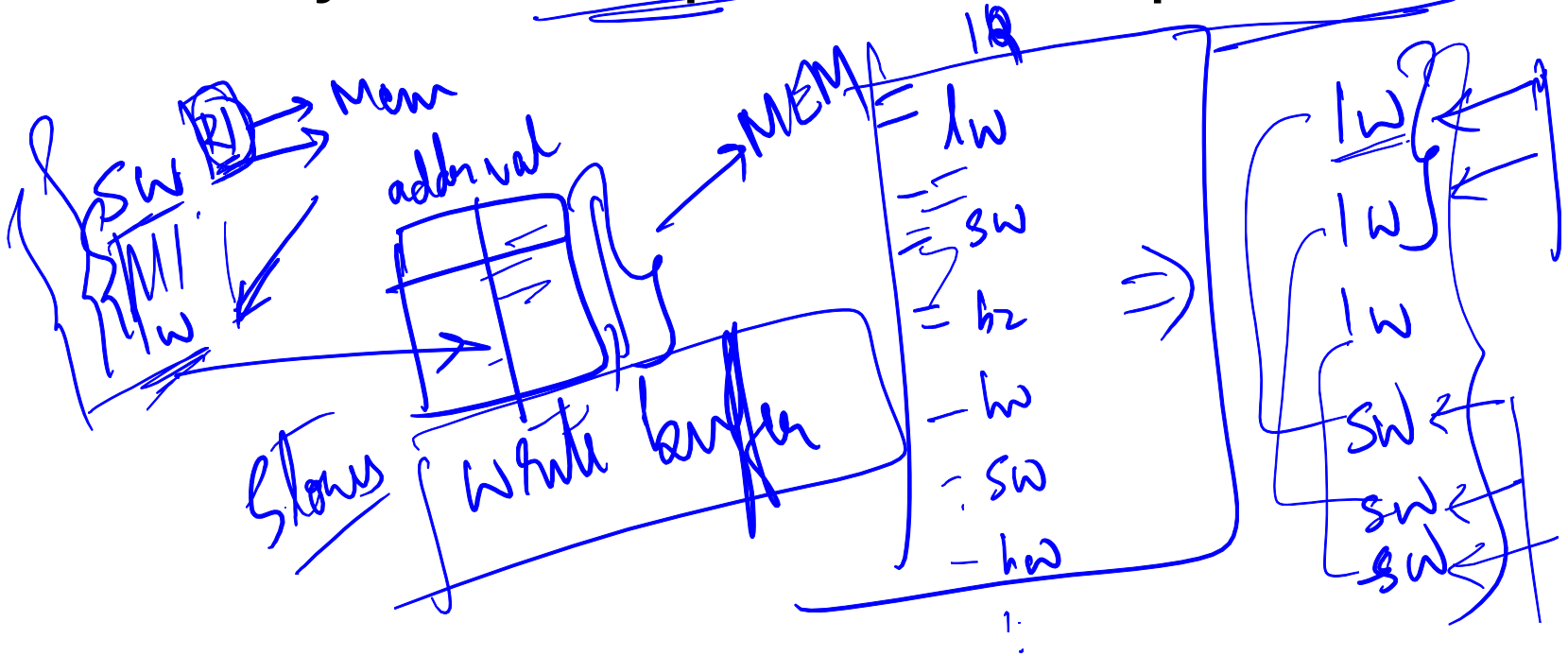
- Instructions after a predicted branch
- Executing a Load earlier than a Store

Speculation

- Instructions after a predicted branch
- Executing a Load earlier than a Store
- Unroll after incorrect speculation

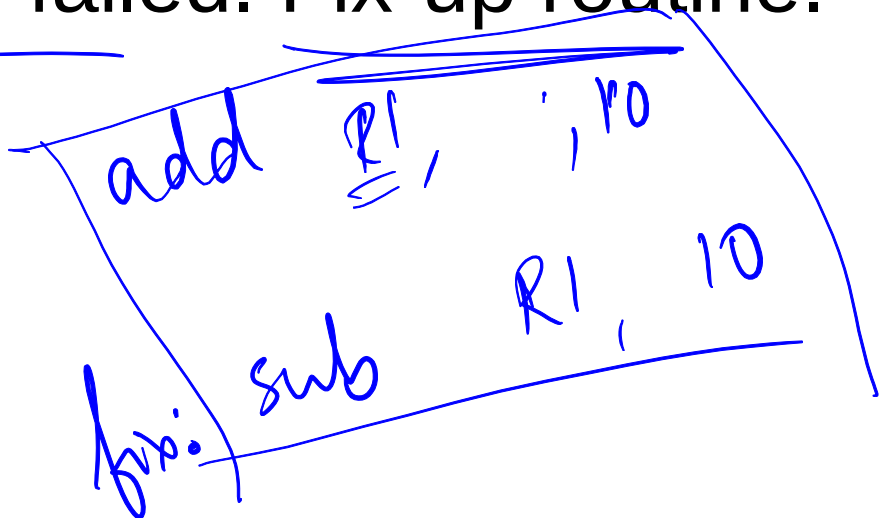
Speculation

- Instructions after a predicted branch
- Executing a Load earlier than a Store
- Unroll after incorrect speculation
- Is done both by the Compiler and the processor



Speculation

- Instructions after a predicted branch
- Executing a Load earlier than a Store
- Unroll after incorrect speculation
- Is done both by the Compiler and the processor
- Compiler inserts instructions to check if speculation failed. Fix-up routine.



Speculation

- Instructions after a predicted branch
- Executing a Load earlier than a Store
- Unroll after incorrect speculation
- Is done both by the Compiler and the processor
- Compiler inserts instructions to check if speculation failed. Fix-up routine.
- Processor keeps results of speculated instructions in a buffer

Speculation

- Instructions after a predicted branch
- Executing a Load earlier than a Store
- Unroll after incorrect speculation
- Is done both by the Compiler and the processor
- Compiler inserts instructions to check if speculation failed. Fix-up routine.
- Processor keeps results of speculated instructions in a buffer
- Speculation complicates Exception handling

Static Multiple Issue

Static Multiple Issue

Instruction type	Pipe stages					
ALU or branch instruction	IF	ID	EX	MEM	WB	
Load or store instruction	IF	ID	EX	MEM	WB	
ALU or branch instruction		IF	ID	EX	MEM	WB
Load or store instruction		IF	ID	EX	MEM	WB

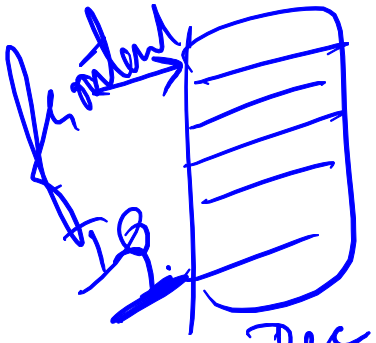
Static Multiple Issue

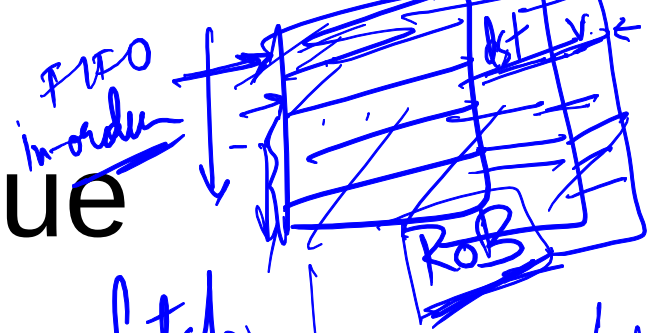
- Issue Packet

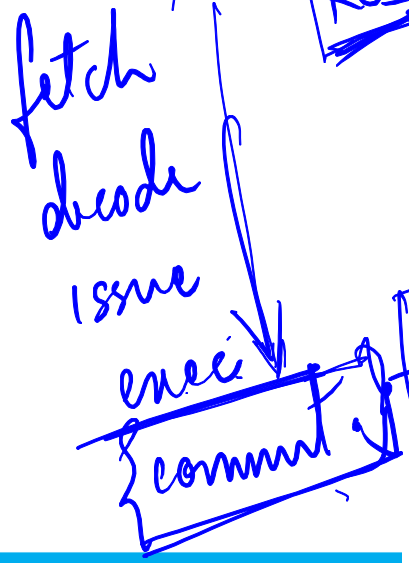
Instruction type	Pipe stages					
ALU or branch instruction	IF	ID	EX	MEM	WB	
Load or store instruction	IF	ID	EX	MEM	WB	
ALU or branch instruction		IF	ID	EX	MEM	WB
Load or store instruction		IF	ID	EX	MEM	WB

Static Multiple Issue

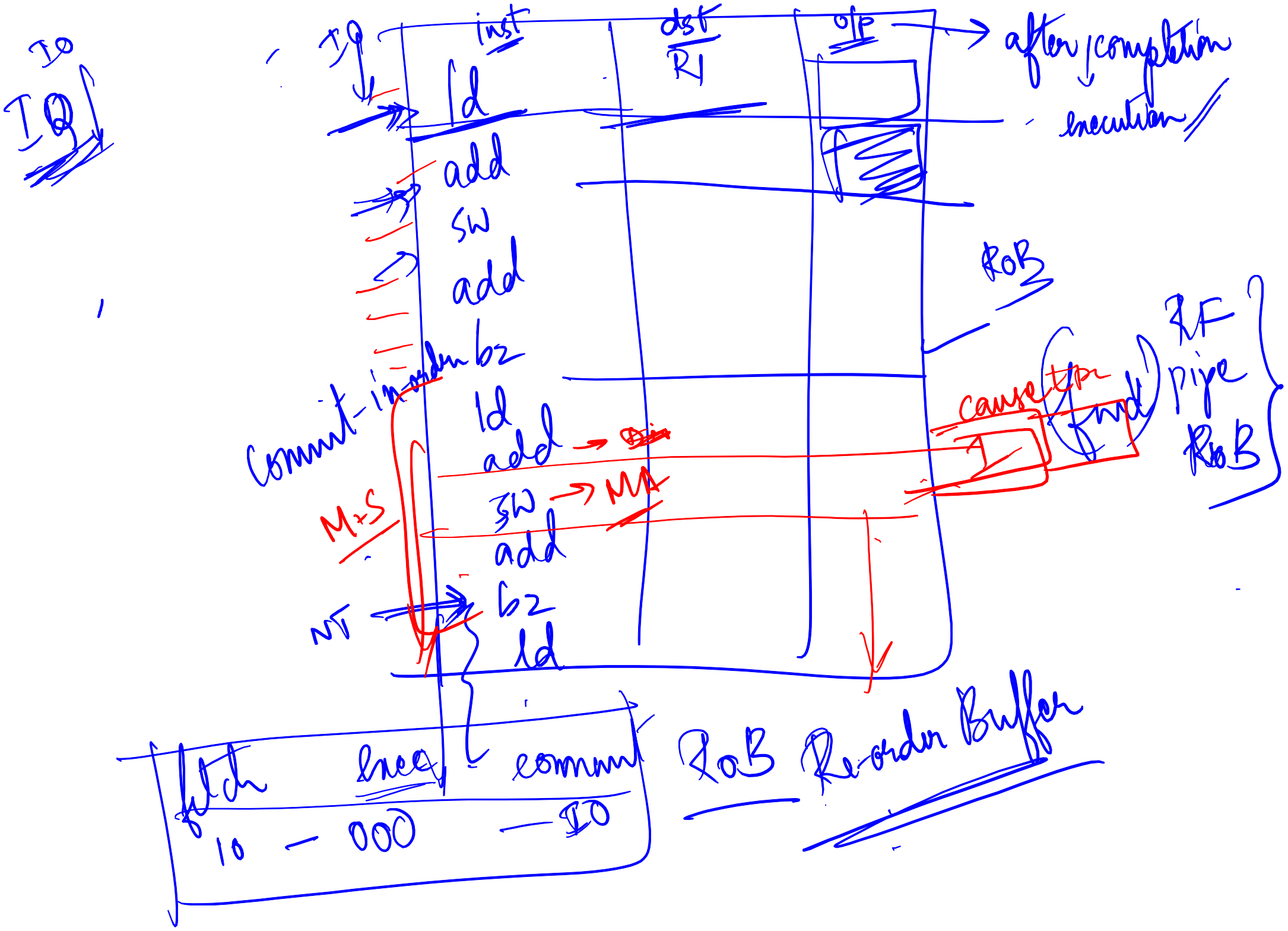
- Issue Packet
- Compiler technique
  - VLIW



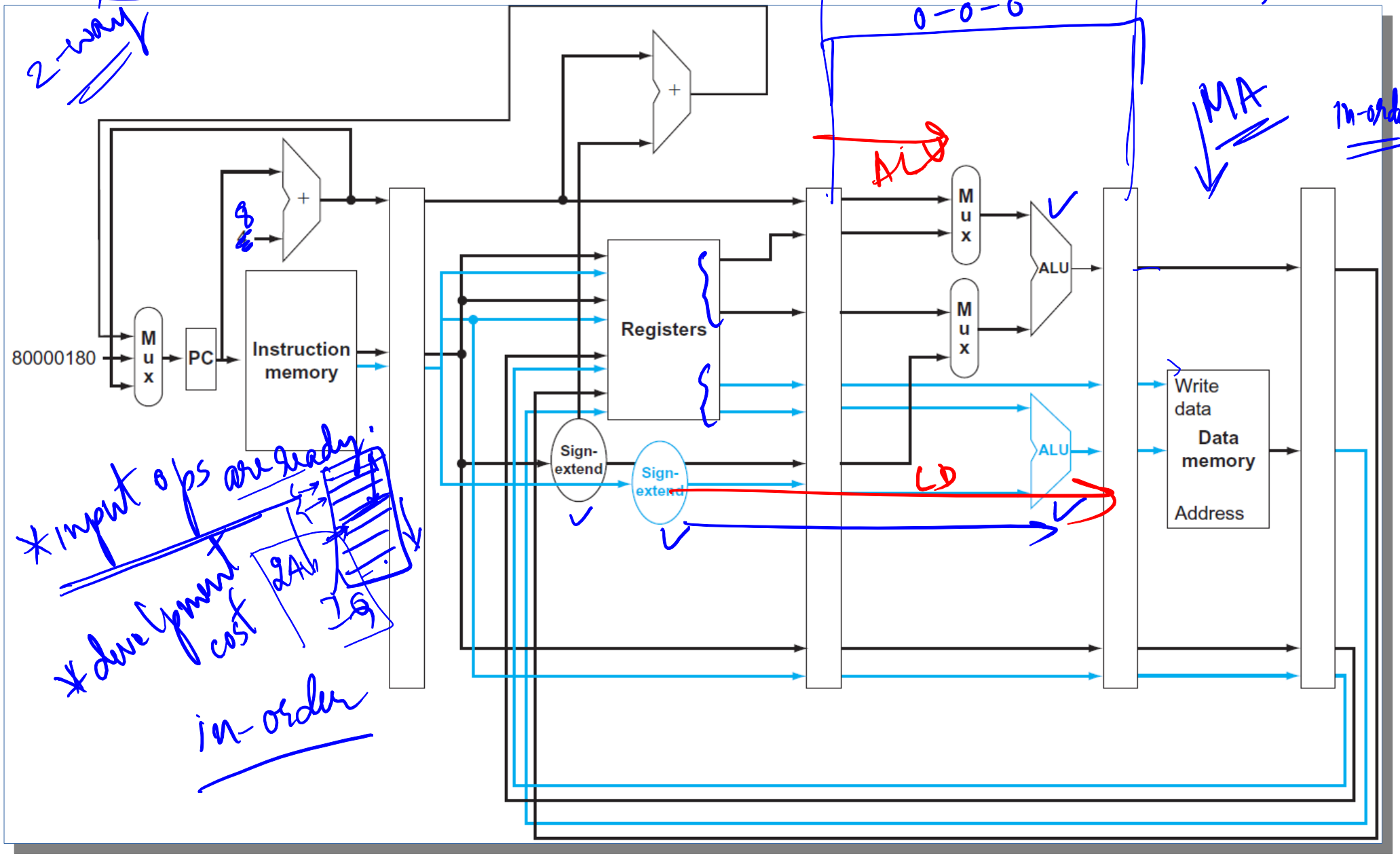




Instruction type	Pipe stages					
ALU or branch instruction	IF	ID	EX	MEM	WB	
Load or store instruction	IF	ID	EX	MEM	WB	
ALU or branch instruction		IF	ID	EX	MEM	WB
Load or store instruction		IF	ID	EX	MEM	WB


IQ
IQ
inst
dst
R1
op
after completion
execution
ld
add
sw
add
br2
ld
add
sw
add
br2
ld
ROB
commit in order
cause typ
end
RF
pipe
RoB
M+S
MA
NT
fetch
issue
commit
10 - 000 - IQ
RoB Re-order Buffer

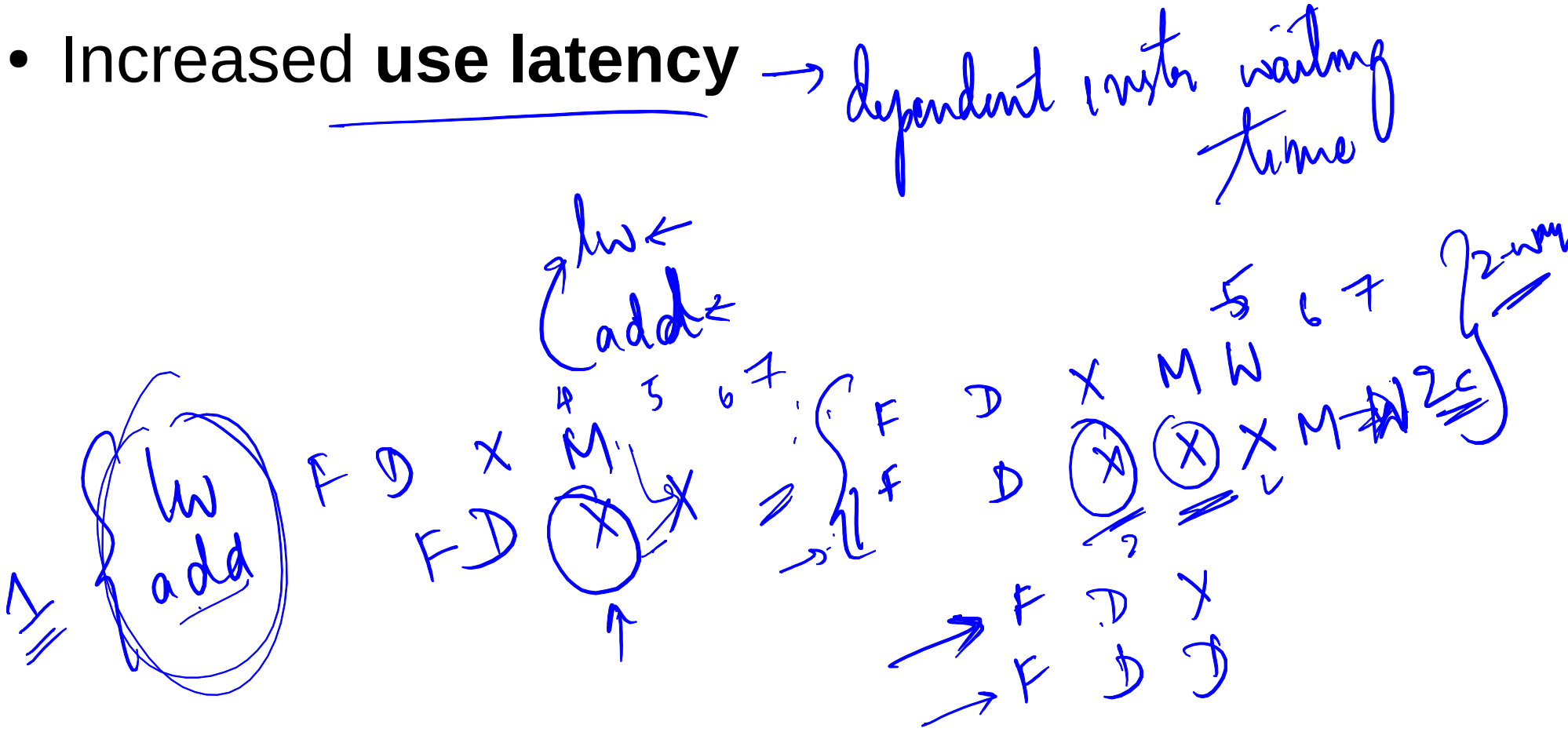
Multiple Issue Pipeline



Multiple Issue

Multiple Issue

- Increased **use latency**



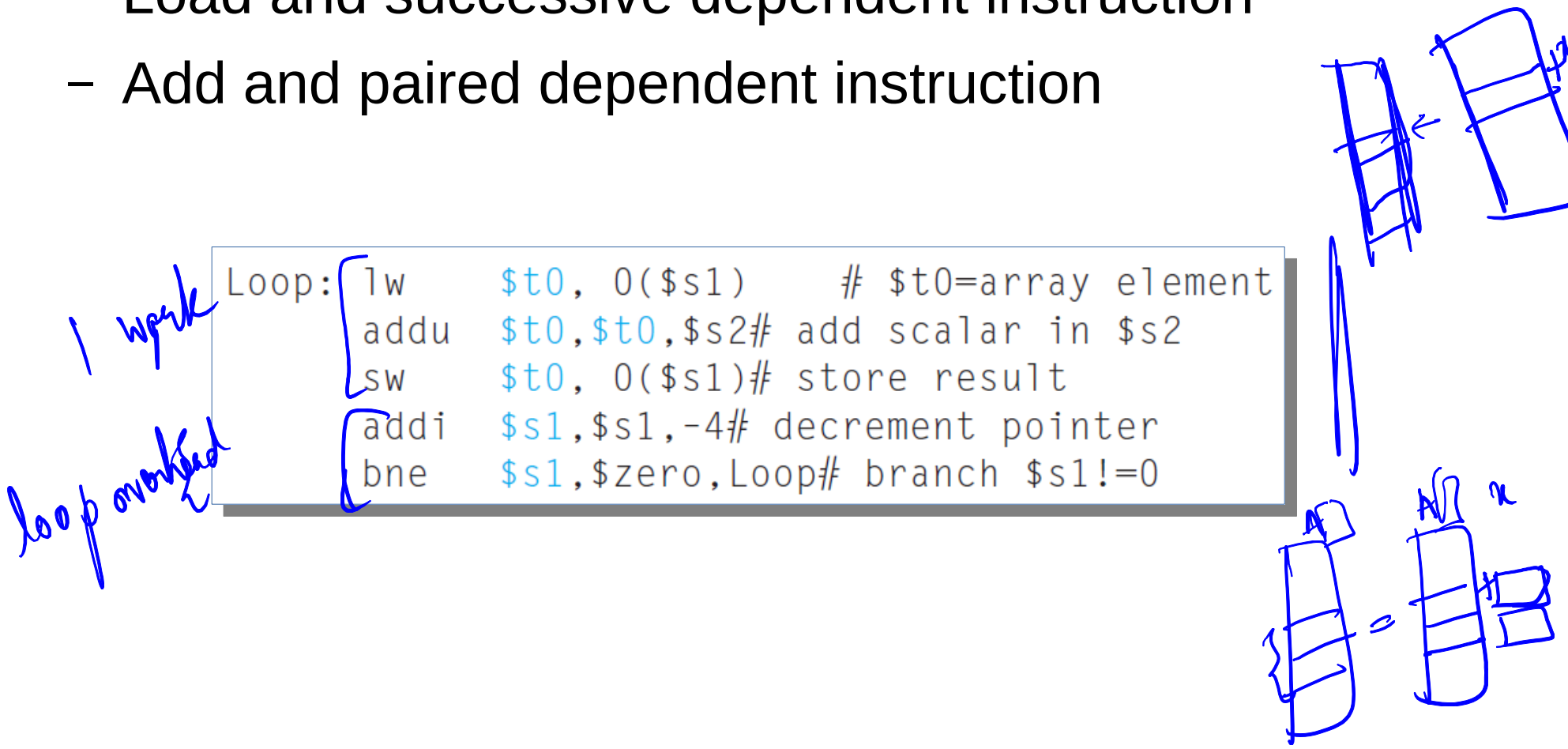
Multiple Issue

- Increased **use latency**
  - Load and successive dependent instruction
  - Add and paired dependent instruction

Multiple Issue

- Increased **use latency**
  - Load and successive dependent instruction
  - Add and paired dependent instruction

```
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
```



Loop Unrolling

- increase the size of BB
- decreases loop overhead

Loop Unrolling

Original Loop

```
Loop:  L.D      F0, 0(R1)
       ADD.D    F4, F0, F2
       S.D      F4, 0(R1)
       DADDUI   R1, R1, #-8
       BNE      R1, R2, Loop
```

UNROLLED LOOP

Loop Unrolling

Original Loop

```
Loop:  L.D      F0, 0(R1)
       ADD.D    F4, F0, F2
       S.D      F4, 0(R1)
       DADDUI   R1, R1, #-8
       BNE      R1, R2, Loop
```

Loop has been unrolled 4 times

```
Loop:  L.D      F0, 0(R1)
       ADD.D    F4, F0, F2
       S.D      F4, 0(R1)
       L.D      F6, -8(R1)
       ADD.D    F8, F2, F6
       S.D      F8, -8(R1)
       L.D      F10, -16(R1)
       ADD.D    F12, F2, F10
       S.D      F12, -16(R1)
       L.D      F14, -24(R1)
       ADD.D    F16, F2, F14
       S.D      F16, -24(R1)
       DADDUI   R1, R1, #-32
       BNE      R1, R2, Loop
```

UNROLLED LOOP

Loop Unrolling

Original Loop

```
Loop:  L.D      F0, 0(R1)
       ADD.D    F4, F0, F2
       S.D      F4, 0(R1)
       DADDUI   R1, R1, #-8
       BNE      R1, R2, Loop
```

Loop has been unrolled 4 times

$$\frac{Loop\ Work}{Loop\ Overhead} = \frac{12}{2}$$

UNROLLED LOOP

```
Loop:  L.D      F0, 0(R1)
       ADD.D    F4, F0, F2
       S.D      F4, 0(R1)
       L.D      F6, -8(R1)
       ADD.D    F8, F2, F6
       S.D      F8, -8(R1)
       L.D      F10, -16(R1)
       ADD.D    F12, F2, F10
       S.D      F12, -16(R1)
       L.D      F14, -24(R1)
       ADD.D    F16, F2, F14
       S.D      F16, -24(R1)
       DADDUI   R1, R1, #-32
       BNE      R1, R2, Loop
```

Dynamic Multiple Issue Processors

- Superscalar Processors

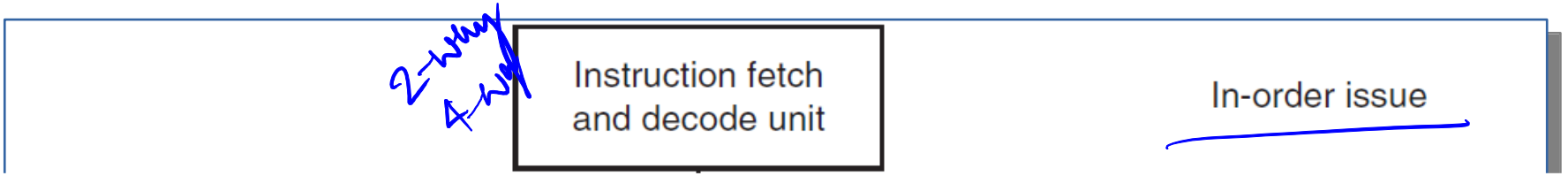
Dynamic Multiple Issue Processors

- Superscalar Processors
- Dynamic pipeline scheduling

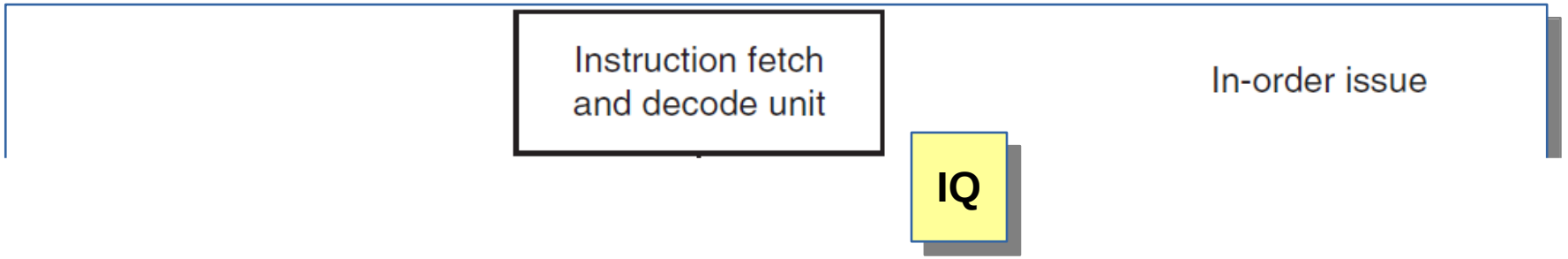
Dynamic Multiple Issue Processors

- Superscalar Processors
- Dynamic pipeline scheduling
  - Choose next instruction to be executed based on ready inputs and available functional units

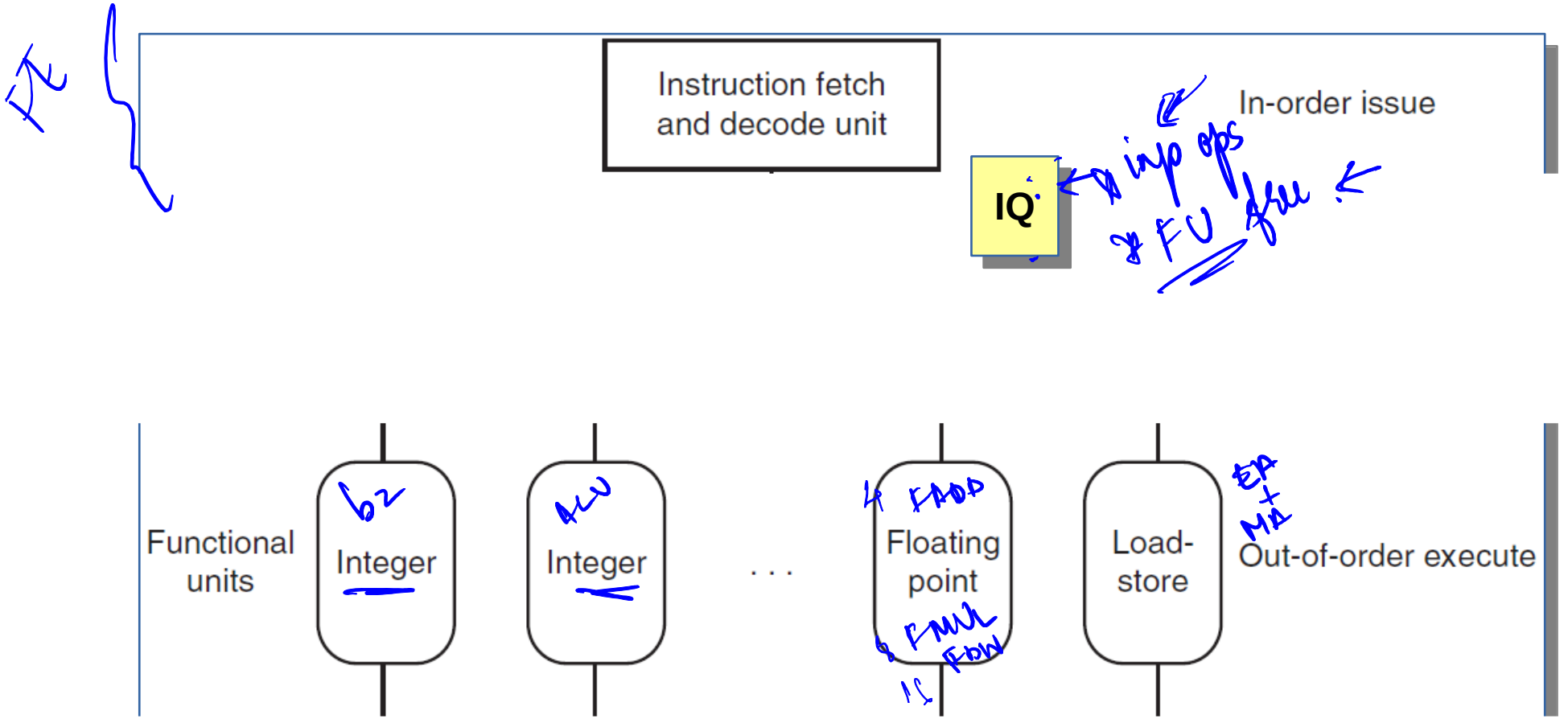
Dynamically Scheduled Pipeline



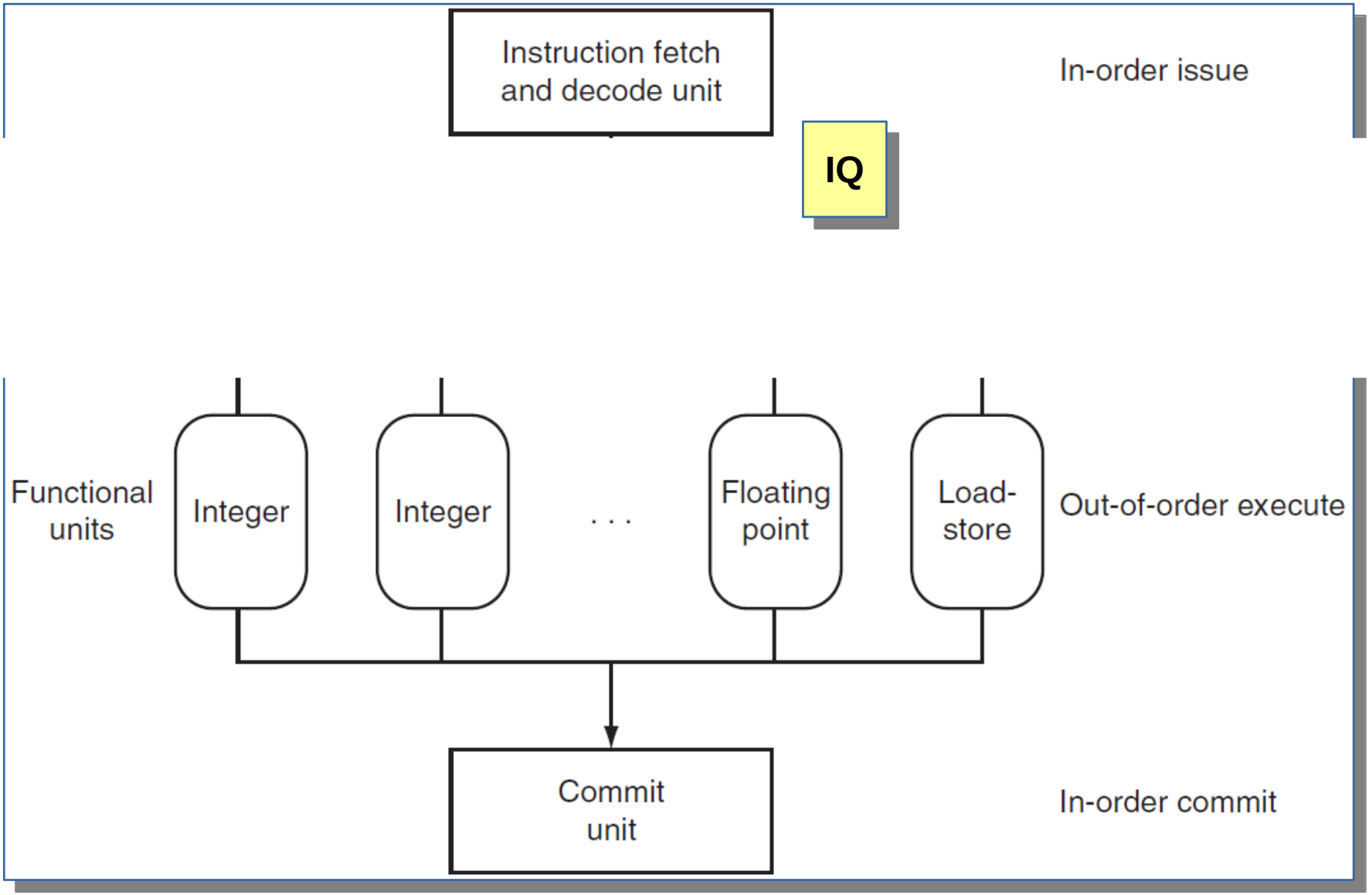
Dynamically Scheduled Pipeline



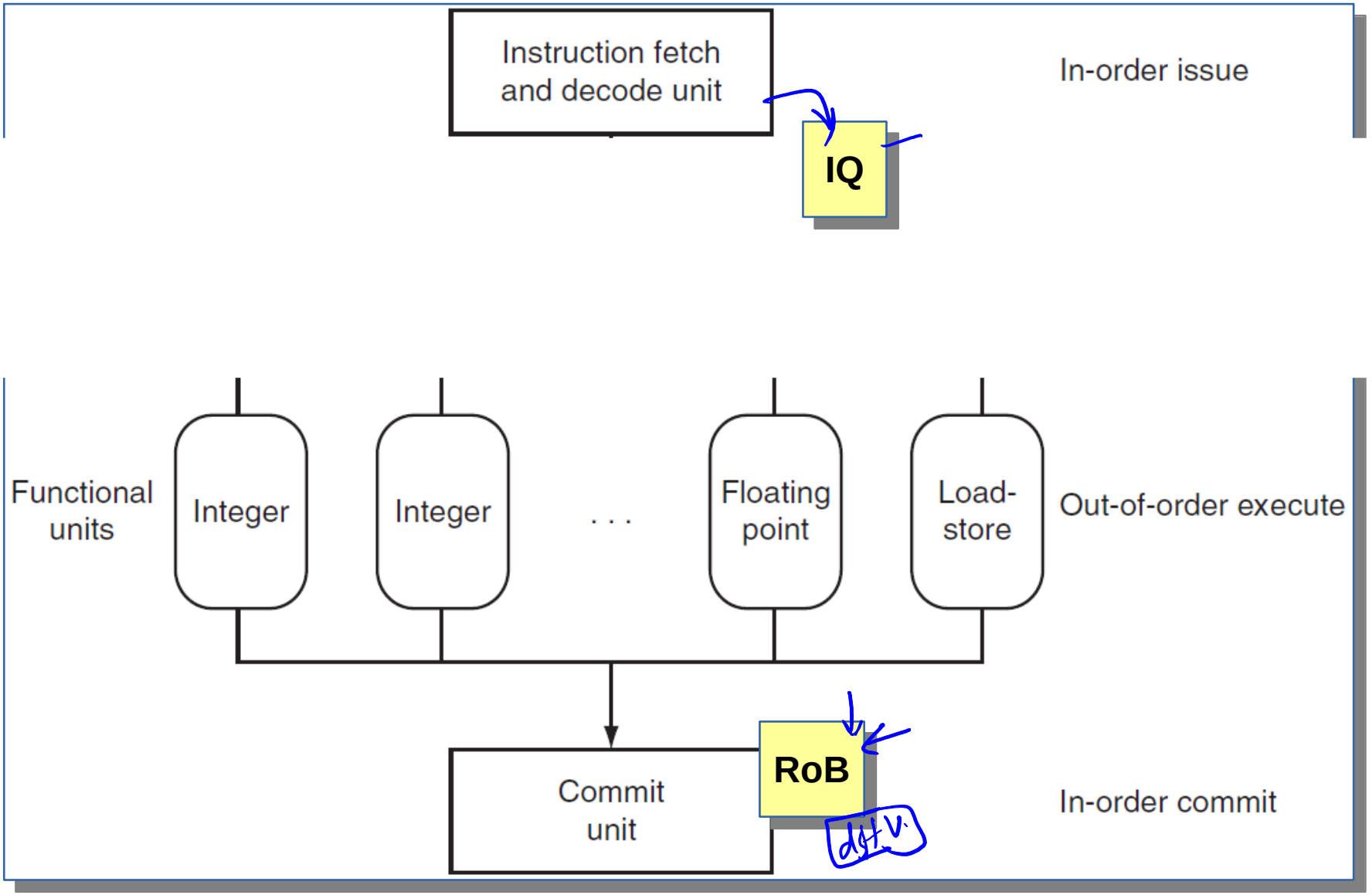
Dynamically Scheduled Pipeline



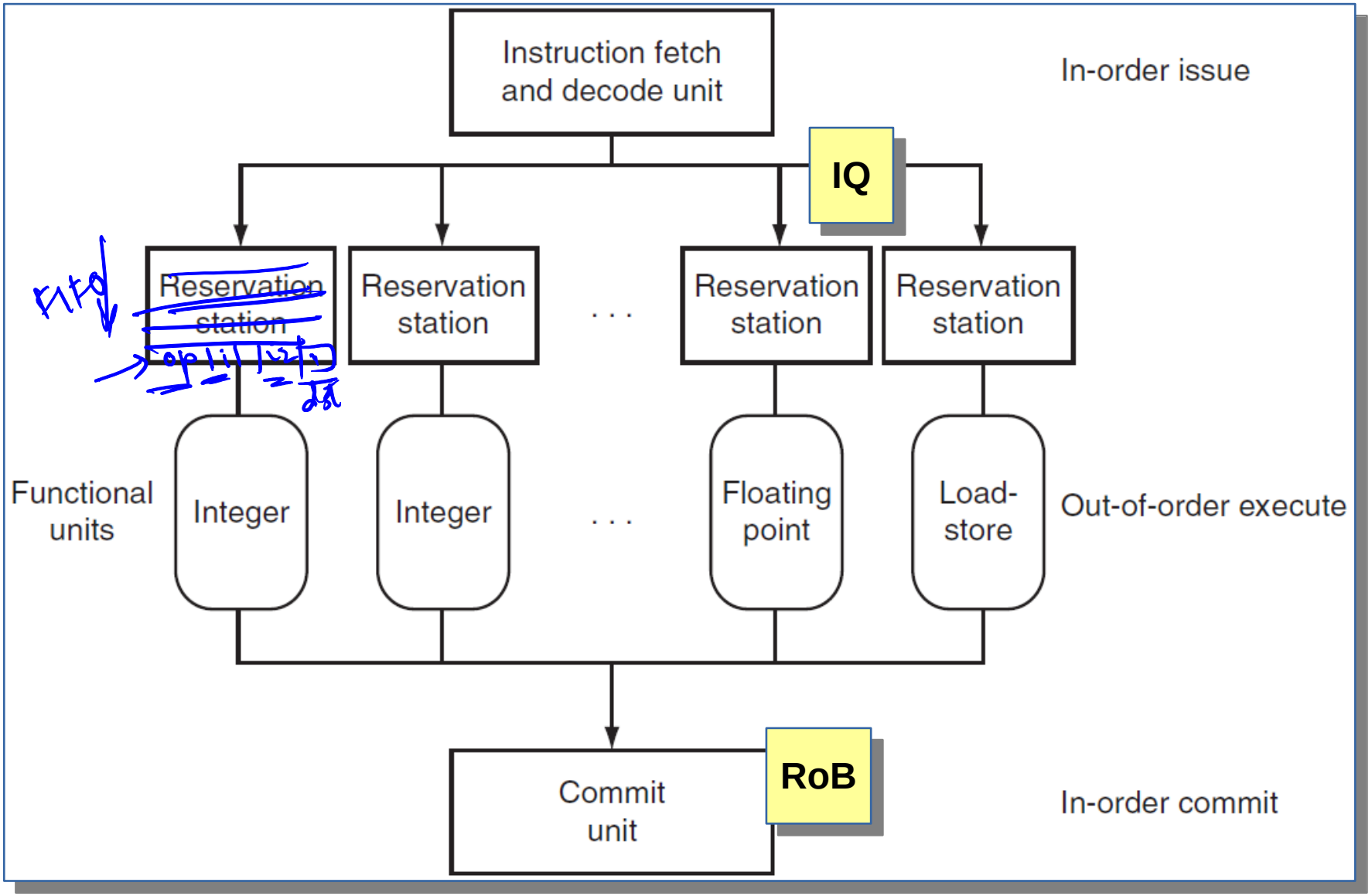
Dynamically Scheduled Pipeline



Dynamically Scheduled Pipeline

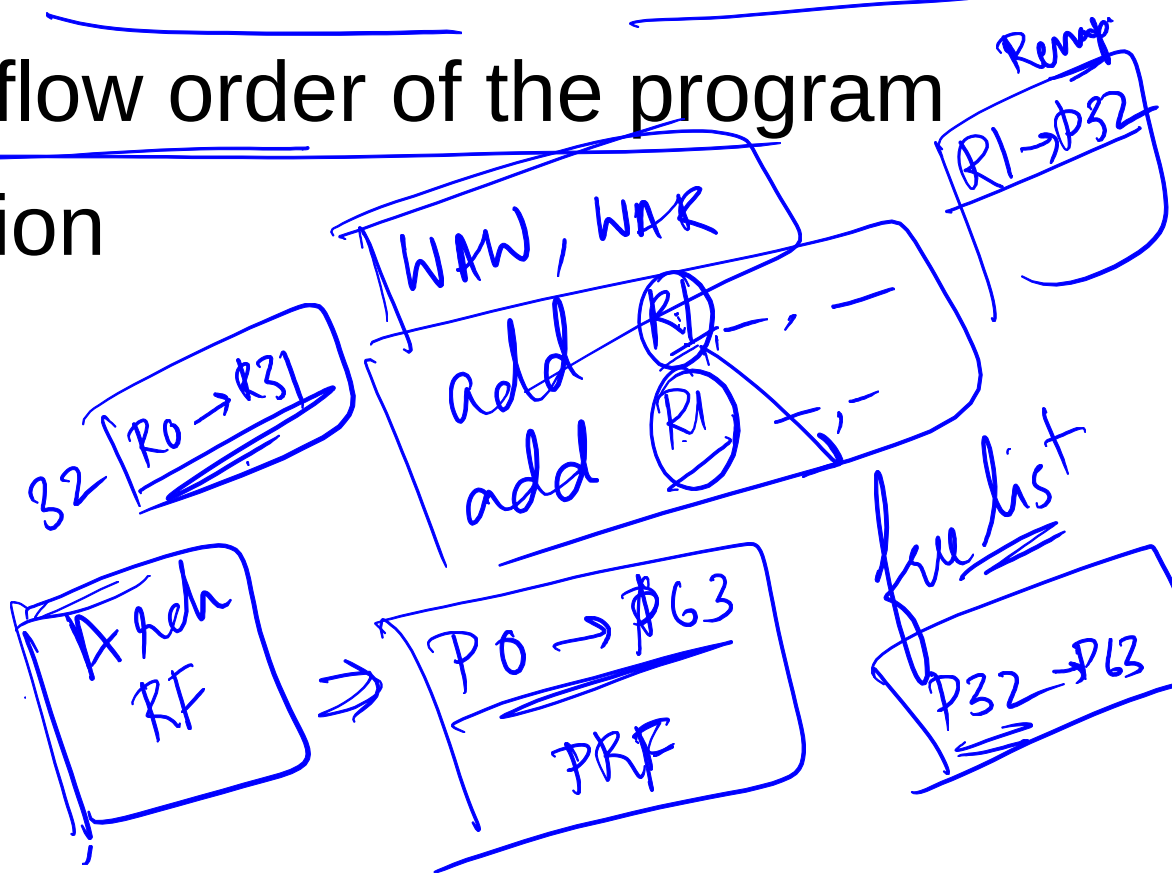


Dynamically Scheduled Pipeline



Dynamically Scheduled Pipeline

- Register Renaming
- Instruction Queue, Reorder Buffer, Store buffer
- Preserves the data flow order of the program
- Out-of-order execution
- In-order Commit



Intel Processors

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue Width	Out-of-Order/ Speculation	Cores/ Chip	Power	
Intel 486	1989	25 MHz	5	1	No	1	5	W
Intel Pentium	1993	66 MHz	5	2	No	1	10	W
Intel Pentium Pro	1997	200 MHz	10	3	Yes	1	29	W
Intel Pentium 4 Willamette	2001	2000 MHz	22	3	Yes	1	75	W
Intel Pentium 4 Prescott	2004	3600 MHz	31	3	Yes	1	103	W
Intel Core	2006	2930 MHz	14	4	Yes	2	75	W
Intel Core i5 Nehalem	2010	3300 MHz	14	4	Yes	1	87	W
Intel Core i5 Ivy Bridge	2012	3400 MHz	14	4	Yes	8	77	W

Intel Processors

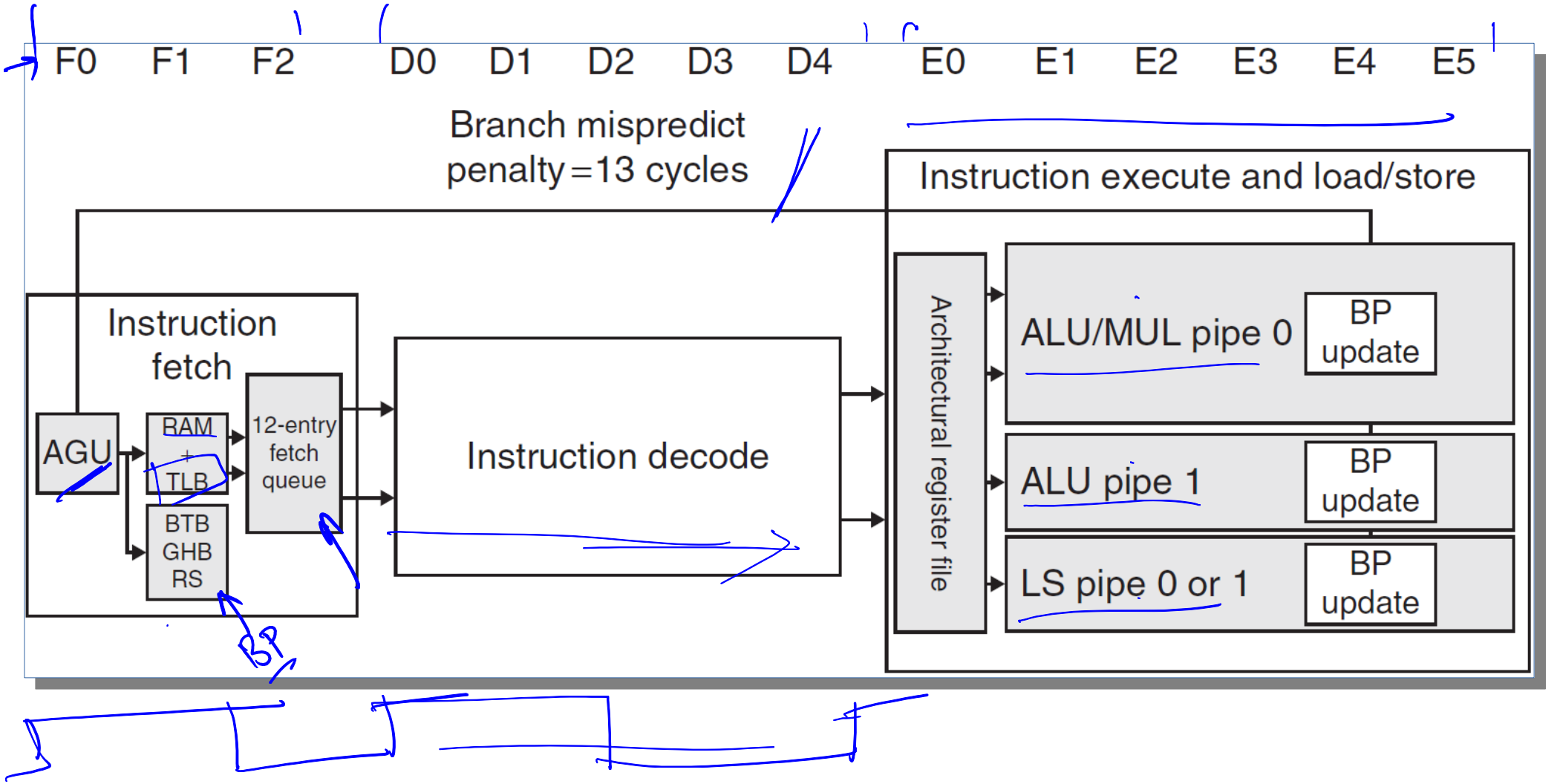
Microprocessor	Year	Clock Rate	Pipeline Stages	Issue Width	Out-of-Order/ Speculation	Cores/ Chip	Power	
Intel 486	1989	25 MHz	5	1	No	1	5	W
Intel Pentium	1993	66 MHz	5	2	No	1	10	W
Intel Pentium Pro	1997	200 MHz	10	3	Yes	1	29	W
Intel Pentium 4 Willamette	2001	2000 MHz	22	3	Yes	1	75	W
Intel Pentium 4 Prescott	2004	3600 MHz	31	3	Yes	1	103	W
Intel Core	2006	2930 MHz	14	4	Yes	2	75	W
Intel Core i5 Nehalem	2010	3300 MHz	14	4	Yes	1	87	W
Intel Core i5 Ivy Bridge	2012	3400 MHz	14	4	Yes	8	77	W

- Energy efficiency
  - Instructions per Joule

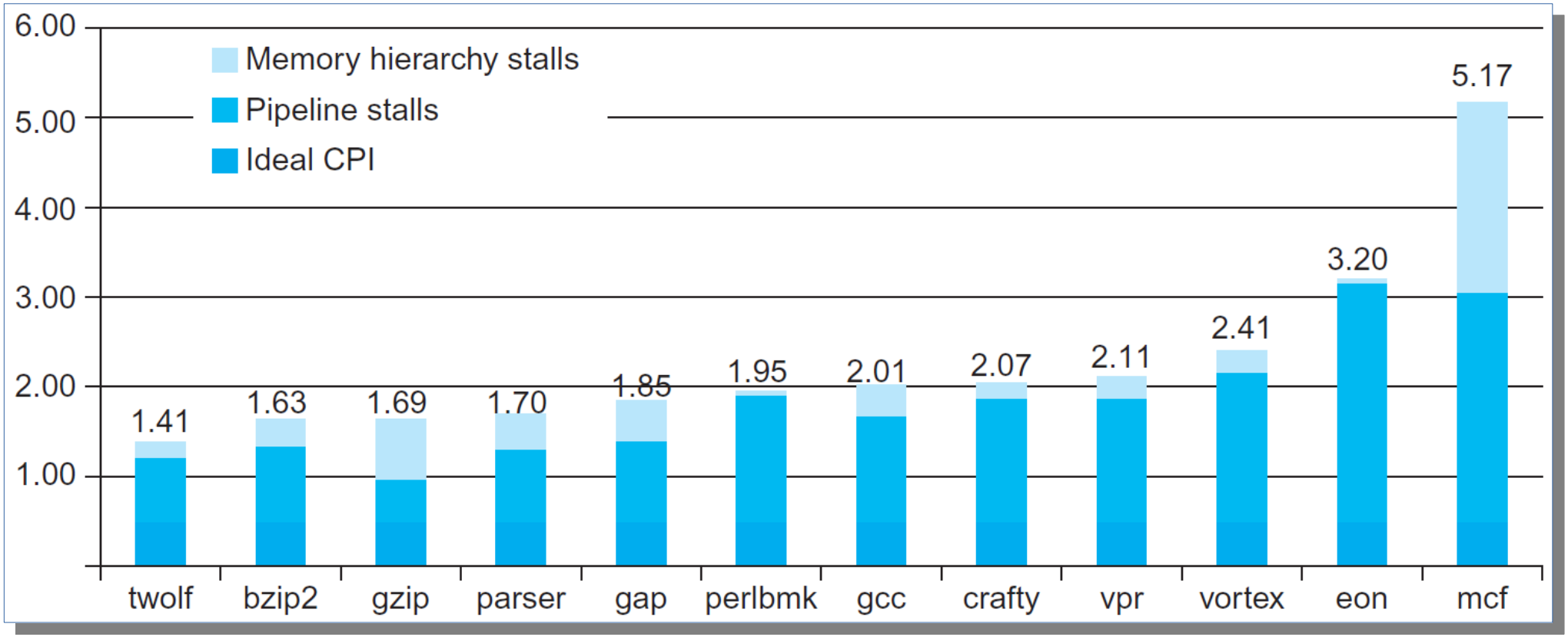
ARM A8 vs. Intel i7-920

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 cache / core	128 - 1024 KiB	256 KiB
3rd level cache (shared)	–	2 - 8 MiB

ARM Cortex-A8

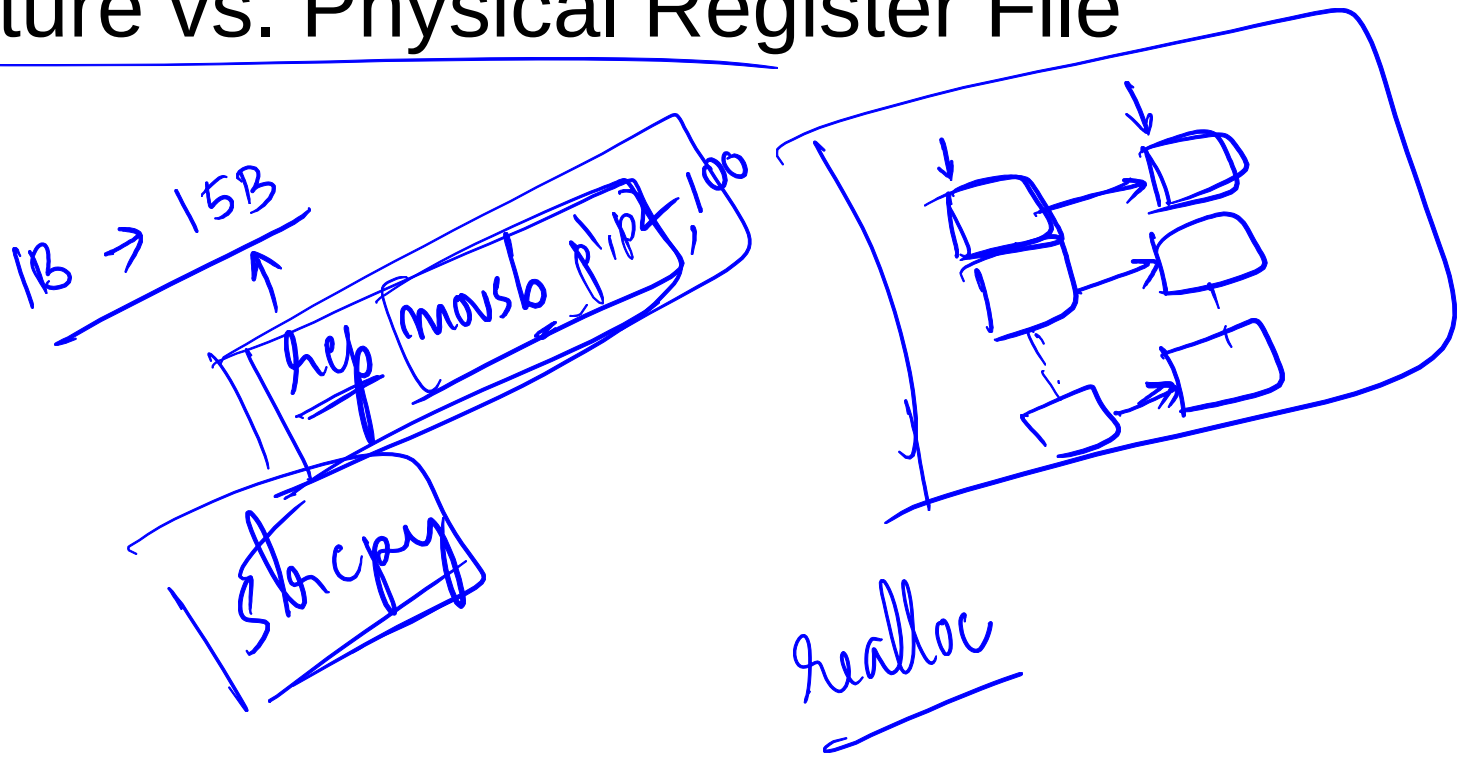


ARM Cortex A8 – Performance



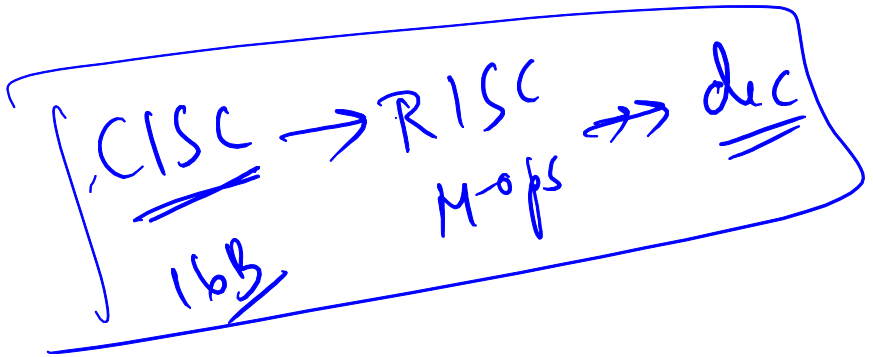
Intel i7

- CISC vs. RISC
- Micro-operations (μ-ops)
- Architecture vs. Physical Register File



Intel i7 – Instruction Execution

- Instruction fetch
  - Fetches 16 bytes from the instruction cache into predecode instruction buffer.
  - Misprediction penalty ~ 15 cycles

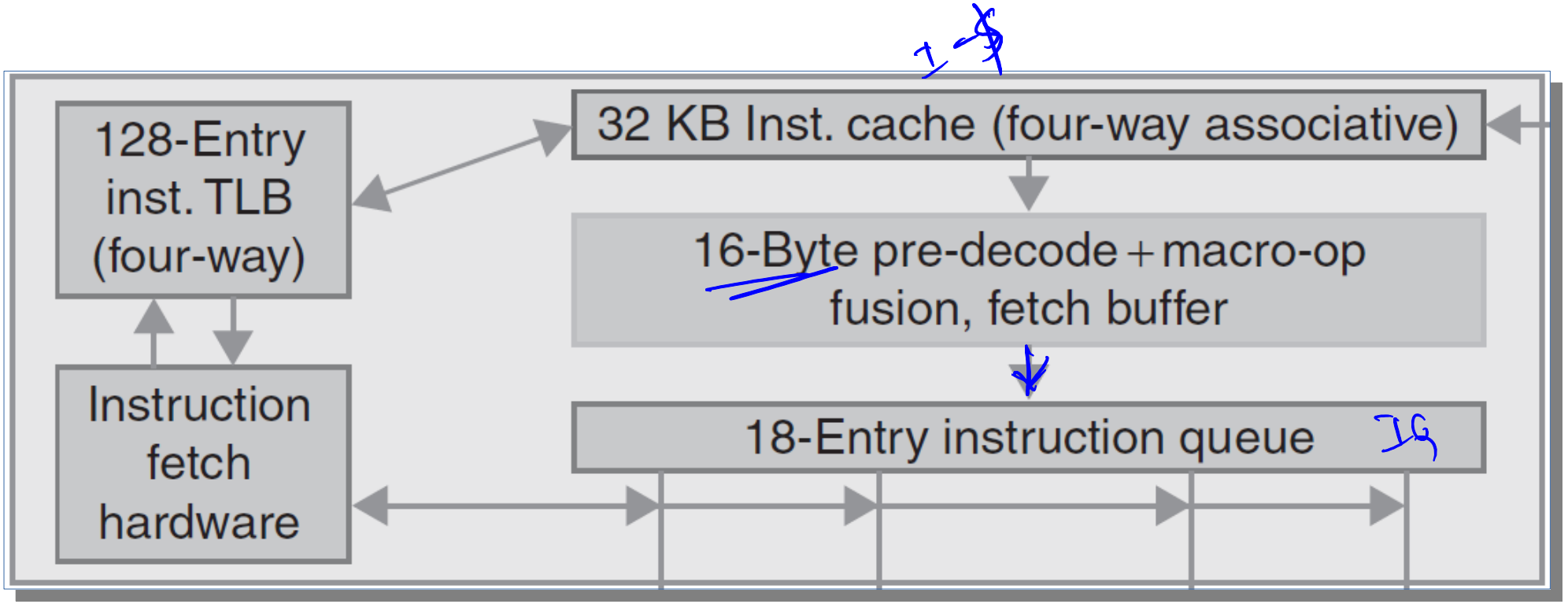


Intel i7 – Instruction Execution

- Instruction fetch
  - Fetches 16 bytes from the instruction cache into predecode instruction buffer.
  - Misprediction penalty ~ 15 cycles
- Predecode stage
  - 16 bytes into individual x86 instructions.
  - 18-entry instruction queue.

Intel i7 – Instruction Execution

- Instruction fetch
- Predecode stage



Intel i7 – Instruction Execution

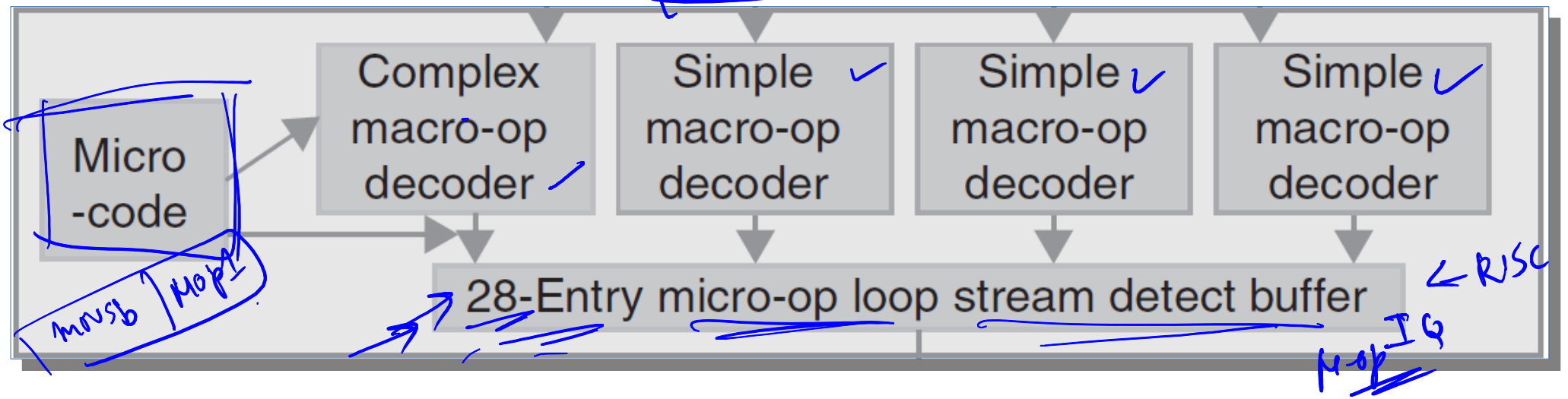
- μ-op decode
  - 3 decoders to handle simple instructions
  - 1 microcode engine for complex instructions
  - 28-entry micro-op buffer.

Intel i7 – Instruction Execution

- μ-op decode
  - 3 decoders to handle simple instructions
  - 1 microcode engine for complex instructions
  - 28-entry micro-op buffer.
- Loop stream detection
  - For small loops (less than 28 instructions or 256 bytes in length)
  - To directly issue the micro-ops from the buffer

Intel i7 – Instruction Execution

- Micro-op decode
- Loop stream detection



Intel i7 – Instruction Execution

- Basic instruction issue

Intel i7 – Instruction Execution

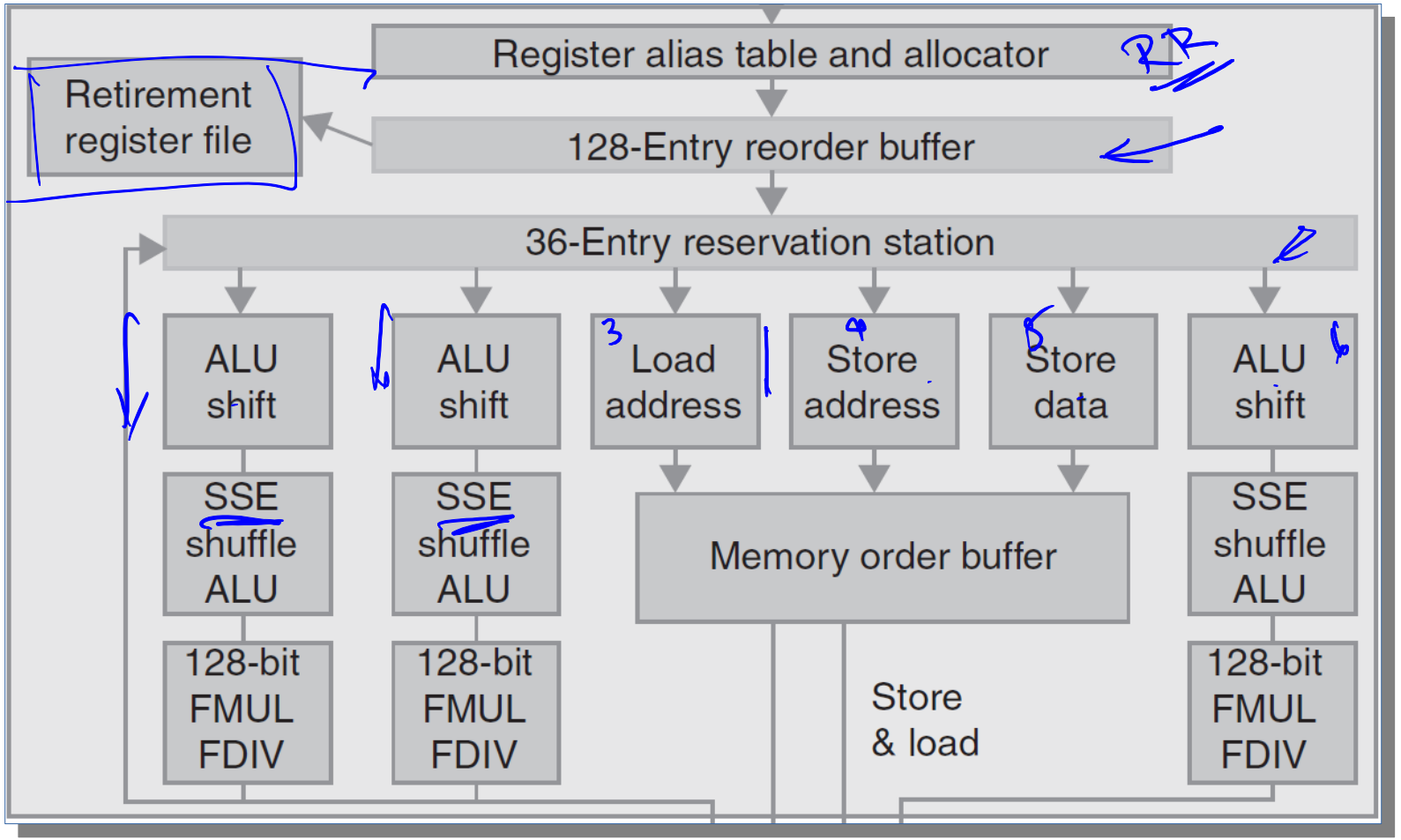
- Basic instruction issue
  - Look up register locations in the rename table
  - Register renaming
  - Allocate a reorder buffer entry
  - Fetch any results from the registers or reorder buffer
  - Send instruction to reservation station

Intel i7 – Instruction Execution

- Basic instruction issue
  - Look up register locations in the rename table
  - Register renaming
  - Allocate a reorder buffer entry
  - Fetch any results from the registers or reorder buffer
  - Send instruction to reservation station
- 6 FUs, 36-entry centralized reservation stations
  - Up to 6 micro-ops can be dispatched per clock cycle

Intel i7 – Instruction Execution

- Basic instruction issue
- 6 FUs, 36-entry centralized reservation stations



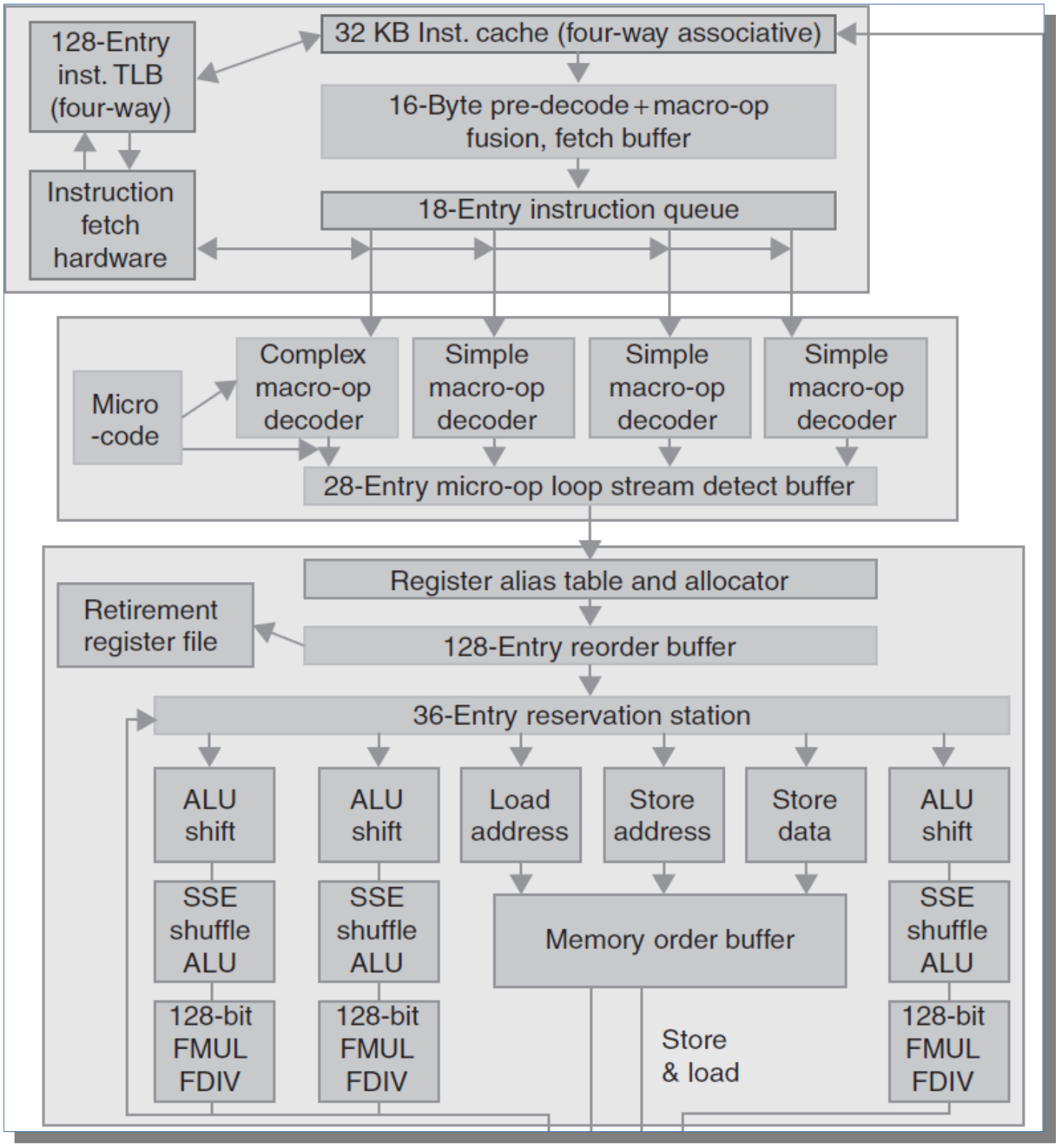
Intel i7 – Instruction Execution

- u-op execution; results sent to any waiting RS and Register Retirement Unit
  - Will stay until instruction is no longer speculative
  - RoB entry for the instruction is flagged as complete.

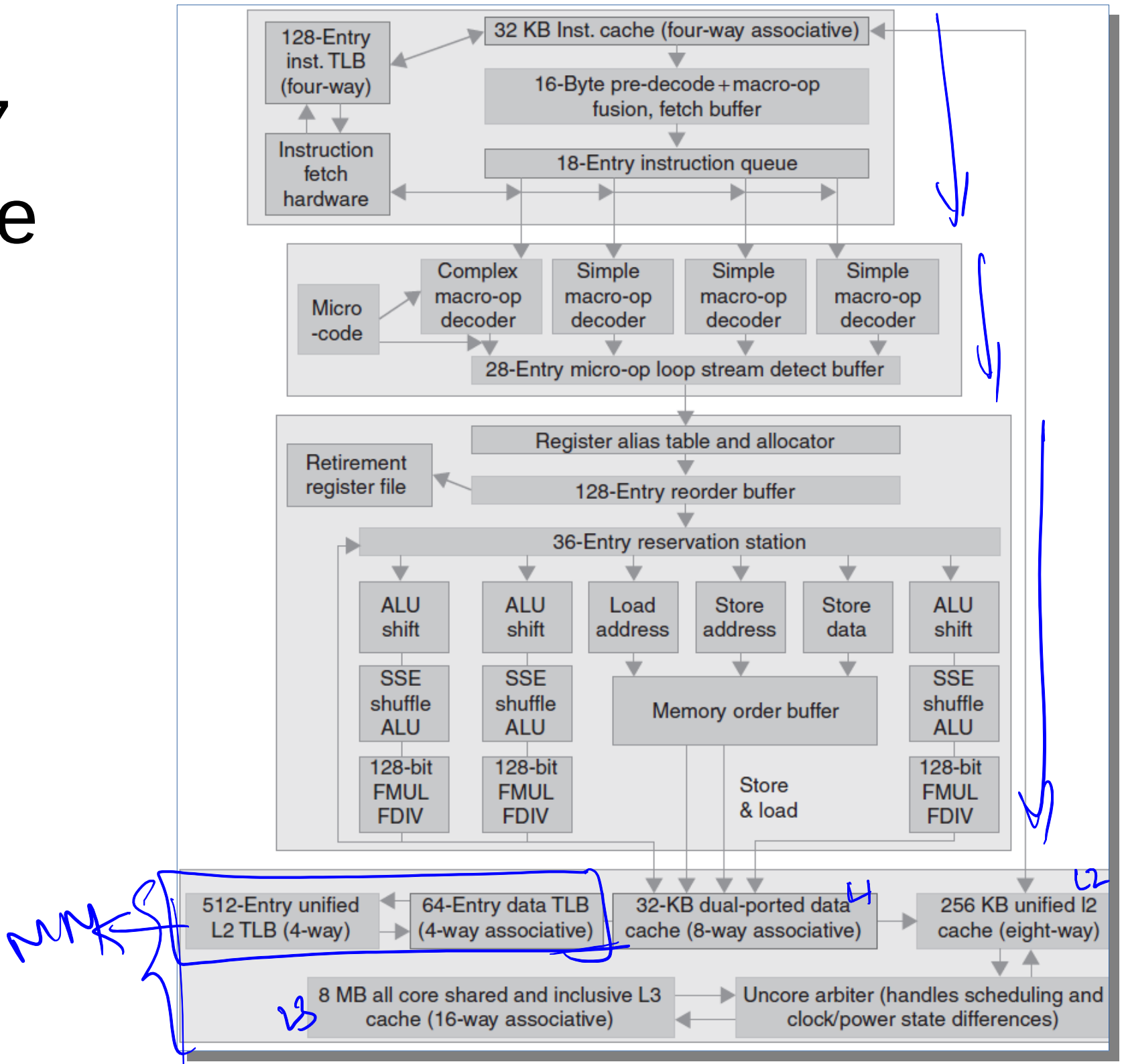
Intel i7 – Instruction Execution

- u-op execution; results sent to any waiting RS and Register Retirement Unit
  - Will stay until instruction is no longer speculative
  - RoB entry for the instruction is flagged as complete.
- Pending writes in the RRU are executed; Instructions are removed from RoB.

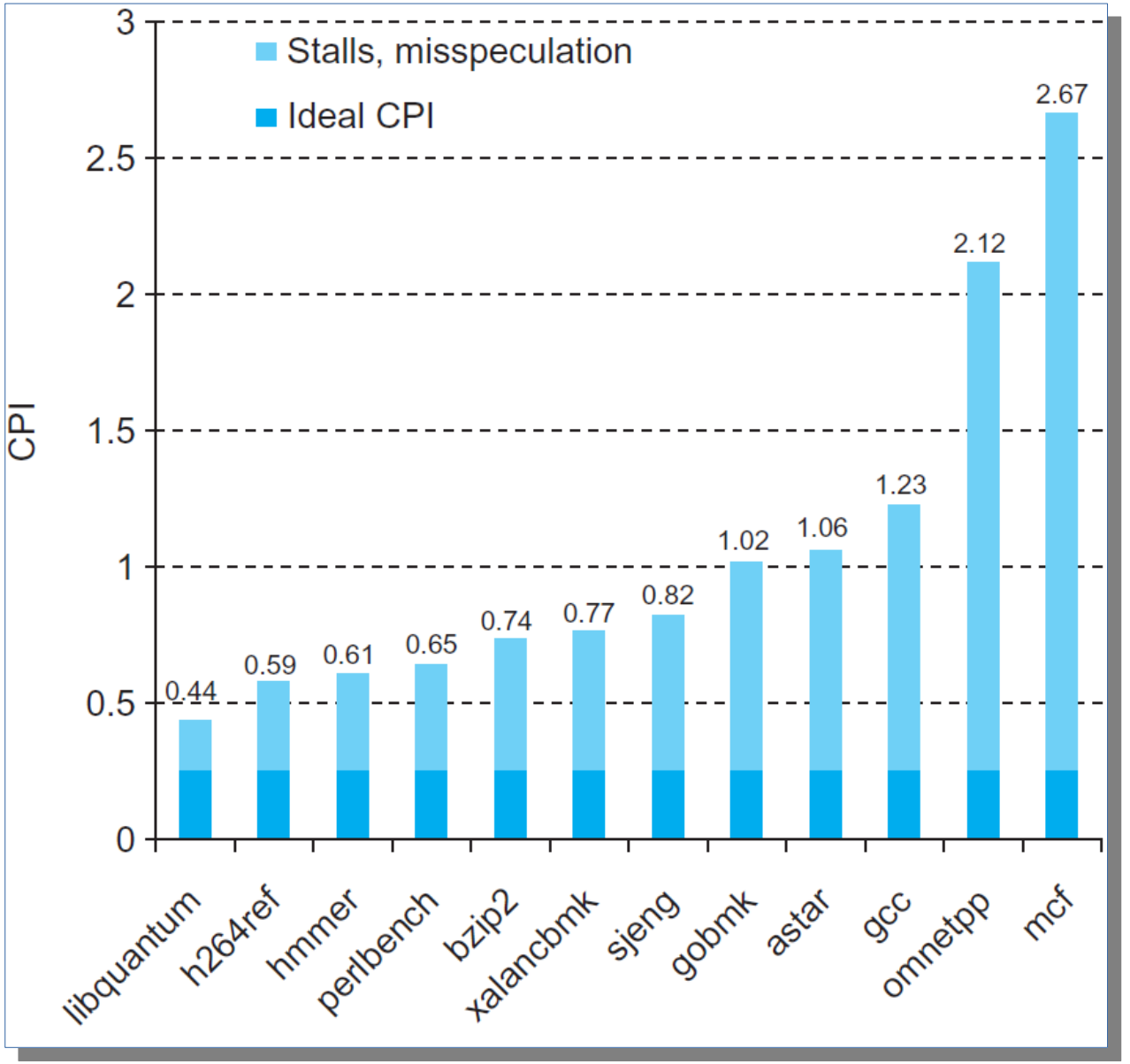
Core i7 Pipeline



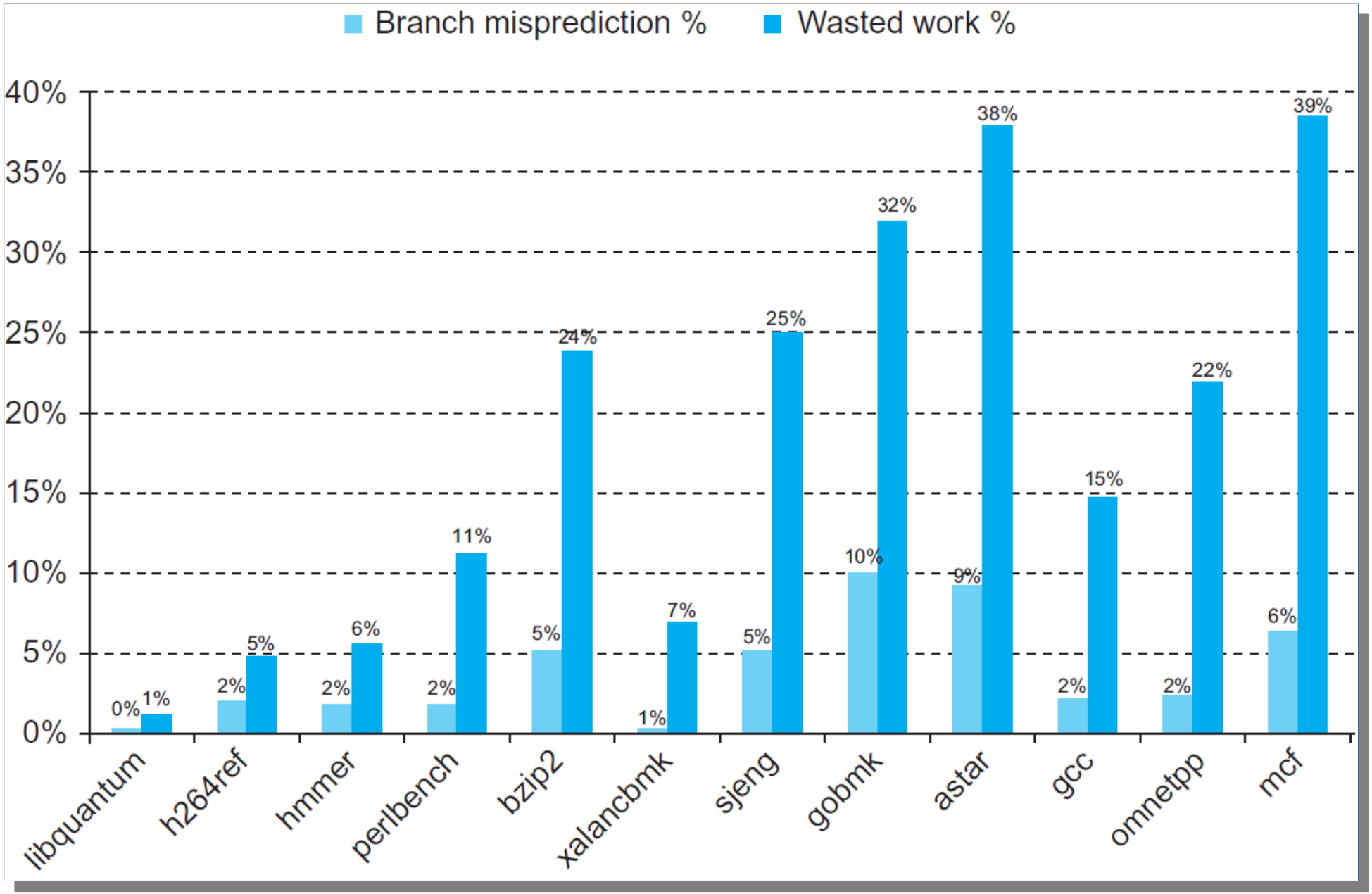
Core i7 Pipeline



i7 Performance



i7 – Loss in Performance



Outline

- Pipeline, Pipelined datapath
- Dependences, Hazards
  - Structural, Data - Stalling, Forwarding
- Control Hazards
- Branch prediction
- ILP, Multiple Issue, Superscalar Processors

Multiple Issue Example

```
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 or branch instruction	Data transfer instruction	Clock cycle

Multiple Issue Example

```
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 or branch instruction	Data transfer instruction	Clock cycle
Loop:		lw $t0, 0($s1)	1
	addi $s1,$s1,-4		2
	addu $t0,$t0,$s2		3
	bne $s1,$zero,Loop	sw $t0, 4($s1)	4

Loop Example

Original Loop

```
Loop:  L.D      F0, 0(R1)
       ADD.D    F4, F0, F2
       S.D      F4, 0(R1)
       DADDUI   R1, R1, #-8
       BNE      R1, R2, Loop
```

Loop overhead = 2 instructions

Execution Time of the loop?

Producer Instruction	Consumer Instruction	Latency to avoid a stall
FP ALU op	Another FP ALU op	**3**
FP ALU op	Store Double	**2**
Load Double	FP ALU op	**1**
Load Double	Store double	**0**
1 Branch delay slot		

Loop Example

Original Loop

```
Loop:  L.D      F0, 0(R1)
       ADD.D    F4, F0, F2
       S.D      F4, 0(R1)
       DADDUI   R1, R1, #-8
       BNE      R1, R2, Loop
```

Producer Instruction	Consumer Instruction	Latency to avoid a stall
FP ALU op	Another FP ALU op	**3**
FP ALU op	Store Double	**2**
Load Double	FP ALU op	**1**
Load Double	Store double	**0**
1 Branch delay slot		

Scheduled Loop

```
Loop:  L.D      F0, 0(R1)
       ■
       ADD.D    F4, F0, F2
       ■
       ■
       S.D      F4, 0(R1)
       DADDUI   R1, R1, #-8
       ■
       BNE      R1, R2, Loop
       ■
```

10 clock cycles

Can we do better?

Loop Example – Smart Schedule

Scheduled Loop

```
Loop:  L.D      F0, 0(R1)
       ■
       ADD.D    F4, F0, F2
       ■
       ■
       S.D      F4, 0(R1)
       DADDUI   R1, R1, #-8
       ■
       BNE      R1, R2, Loop
       ■
```

10 clock cycles

Smart Schedule

```
Loop:  L.D      F0, 0(R1)
       DADDUI   R1, R1, #-8
       ADD.D    F4, F0, F2
       ■
       ■
       S.D      F4, 8(R1)
       ■
       BNE      R1, R2, Loop
       ■
```

Loop Example – Smart Schedule

Scheduled Loop

```
Loop:  L.D      F0, 0(R1)
       ■
       ADD.D    F4, F0, F2
       ■
       ■
       S.D      F4, 0(R1)
       DADDUI   R1, R1, #-8
       ■
       BNE      R1, R2, Loop
       ■
```

10 clock cycles

Smart Schedule

```
Loop:  L.D      F0, 0(R1)
       DADDUI   R1, R1, #-8
       ADD.D    F4, F0, F2
       ■
       BNE      R1, R2, Loop
       S.D      F4, 8(R1)
```

6 clock cycles

Loop Example – Smart Schedule

Producer Instruction	Consumer Instruction	Latency to avoid a stall
FP ALU op	Another FP ALU op	**3**
FP ALU op	Store Double	**2**
Load Double	FP ALU op	**1**
Load Double	Store double	**0**
1 Branch delay slot		

Smart Schedule

```
Loop:  L.D      F0, 0(R1)
       DADDUI   R1, R1, #-8
       ADD.D    F4, F0, F2
       ■
       BNE      R1, R2, Loop
       S.D      F4, 8(R1)
```

6 clock cycles

Loop Example – Smart Schedule

Smart Schedule

```
Loop:  L.D      F0, 0(R1)
       DADDUI   R1, R1, #-8
       ADD.D    F4, F0, F2
       ■
       BNE      R1, R2, Loop
       S.D      F4, 8(R1)
```

6 clock cycles

Loop overhead: 2 instructions

Maximize the work done in the loop

$$\frac{Loop\ Work}{Loop\ Overhead} = \frac{3}{2}$$

Ideal: 3 clock cycles for 3 useful loop instructions (0 loop overhead)

Loop Unrolling

Original Loop

```
Loop:  L.D      F0, 0(R1)
       ADD.D    F4, F0, F2
       S.D      F4, 0(R1)
       DADDUI   R1, R1, #-8
       BNE      R1, R2, Loop
```

Loop has been unrolled 4 times

UNROLLED LOOP

```
Loop:  L.D      F0, 0(R1)
       ADD.D    F4, F0, F2
       S.D      F4, 0(R1)
       L.D      F6, -8(R1)
       ADD.D    F8, F2, F6
       S.D      F8, -8(R1)
       L.D      F10, -16(R1)
       ADD.D    F12, F2, F10
       S.D      F12, -16(R1)
       L.D      F14, -24(R1)
       ADD.D    F16, F2, F14
       S.D      F16, -24(R1)
       DADDUI   R1, R1, #-32
       BNE      R1, R2, Loop
```

Execution Time?

$$\frac{Loop\ Work}{Loop\ Overhead} = \frac{12}{2}$$

Loop Unrolling

Producer Instruction	Consumer Instruction	Latency to avoid a stall
FP ALU op	Another FP ALU op	**3**
FP ALU op	Store Double	**2**
Load Double	FP ALU op	**1**
Load Double	Store double	**0**
1 Branch delay slot		

Execution Time = 28 cycles

```
Loop: L.D      F0, 0(R1)
      ADD.D    F4, F0, F2
      S.D      F4, 0(R1)
      L.D      F6, -8(R1)
      ADD.D    F8, F2, F6
      S.D      F8, -8(R1)
      L.D      F10, -16(R1)
      ADD.D    F12, F2, F10
      S.D      F12, -16(R1)
      L.D      F14, -24(R1)
      ADD.D    F16, F2, F14
      S.D      F16, -24(R1)
      DADDUI   R1, R1, #-32
      BNE      R1, R2, Loop
```

Loop Unrolled and Scheduled

Instr producing result	*Instr using result*	*Latency to avoid a stall*
FP ALU op	Another FP ALU op	3
FP ALU op	Store Double	2
Load Double	FP ALU op	1
Load Double	Store double	0

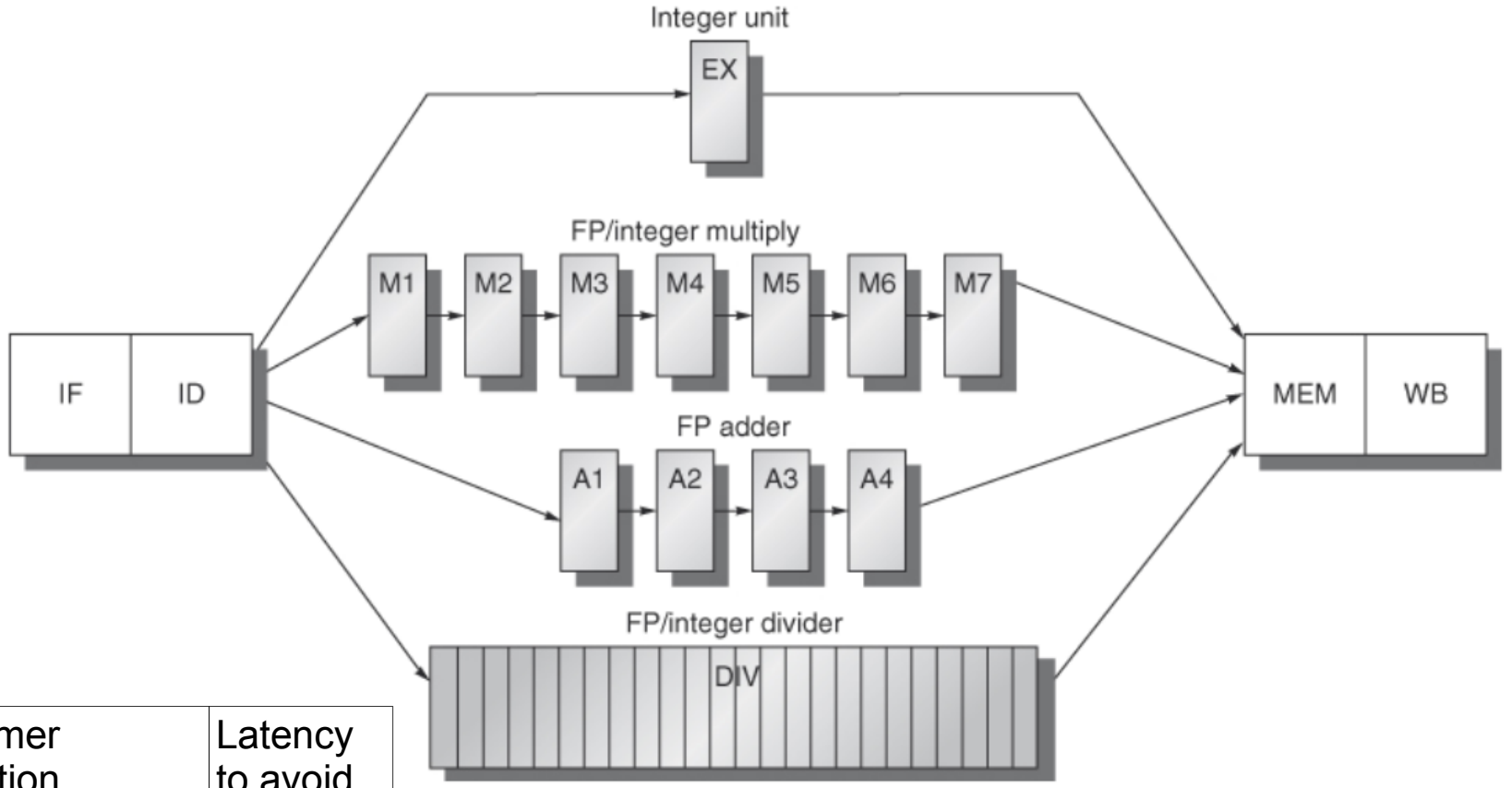
```
Loop:  L.D     F0, 0(R1)
       L.D     F6, -8(R1)
       L.D     F10, -16(R1)
       L.D     F14, -24(R1)
       ADD.D   F4, F0, F2
       ADD.D   F8, F2, F6
       ADD.D   F12, F2, F10
       ADD.D   F16, F2, F14
       S.D     F4, 0(R1)
       S.D     F8, -8(R1)
       DADDUI  R1, R1, #-32
       S.D     F12, 16(R1)
       BNE     R1, R2, Loop
       S.D     F16, 8(R1)
```

Execution Time = 14 cycles.
3.5 cycles per iteration.

- Code Size
- Register pressure

Can we do better?

Multicycle Instructions



Producer Instruction	Consumer Instruction	Latency to avoid a stall
FP ALU op	Another FP ALU op	**3**
FP ALU op	Store Double	**2**
Load Double	FP ALU op	**1**
Load Double	Store double	**0**