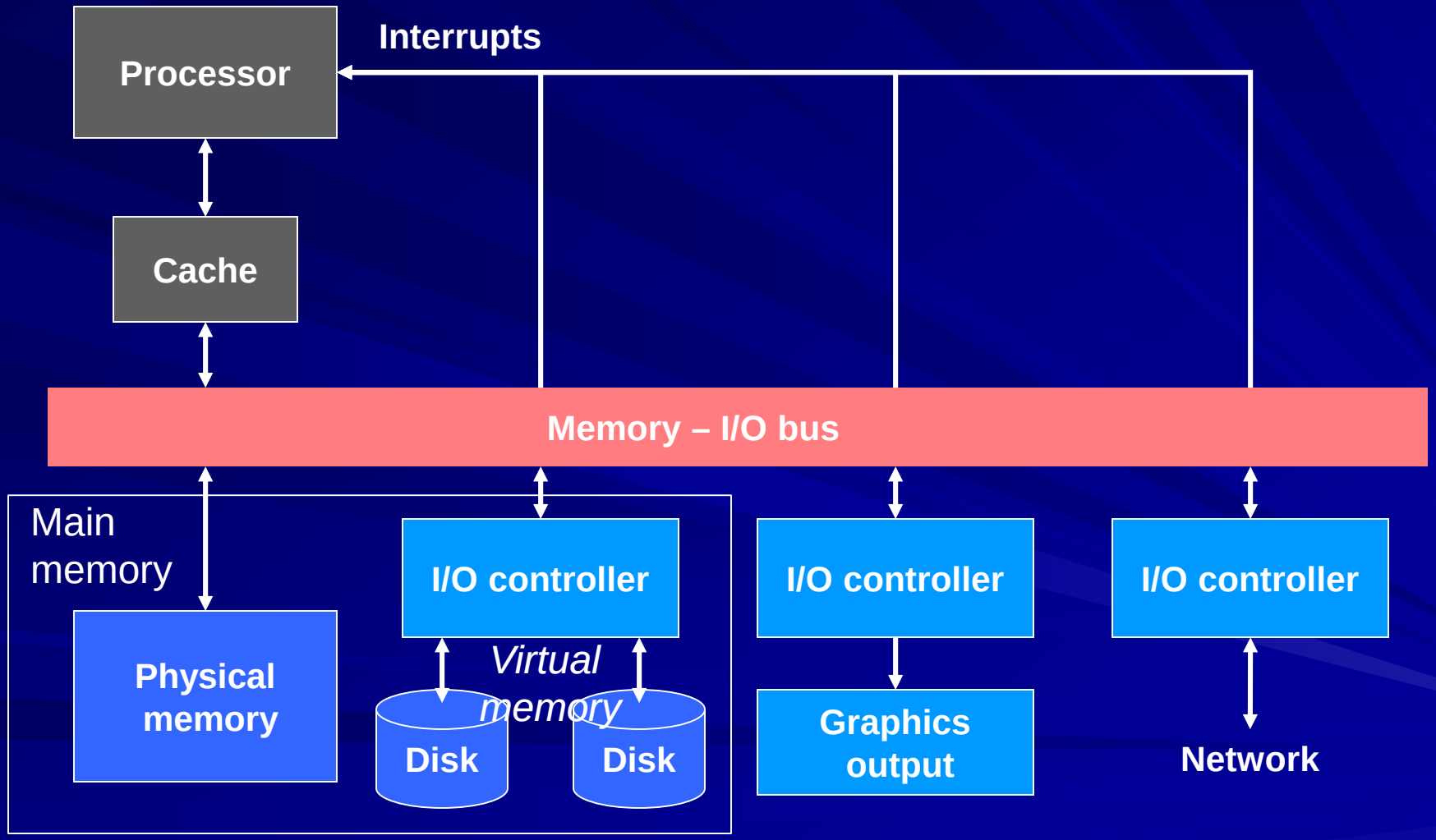
ELEC 5200-001/6200-001
Computer Architecture and Design
Spring 2019
Parallelism

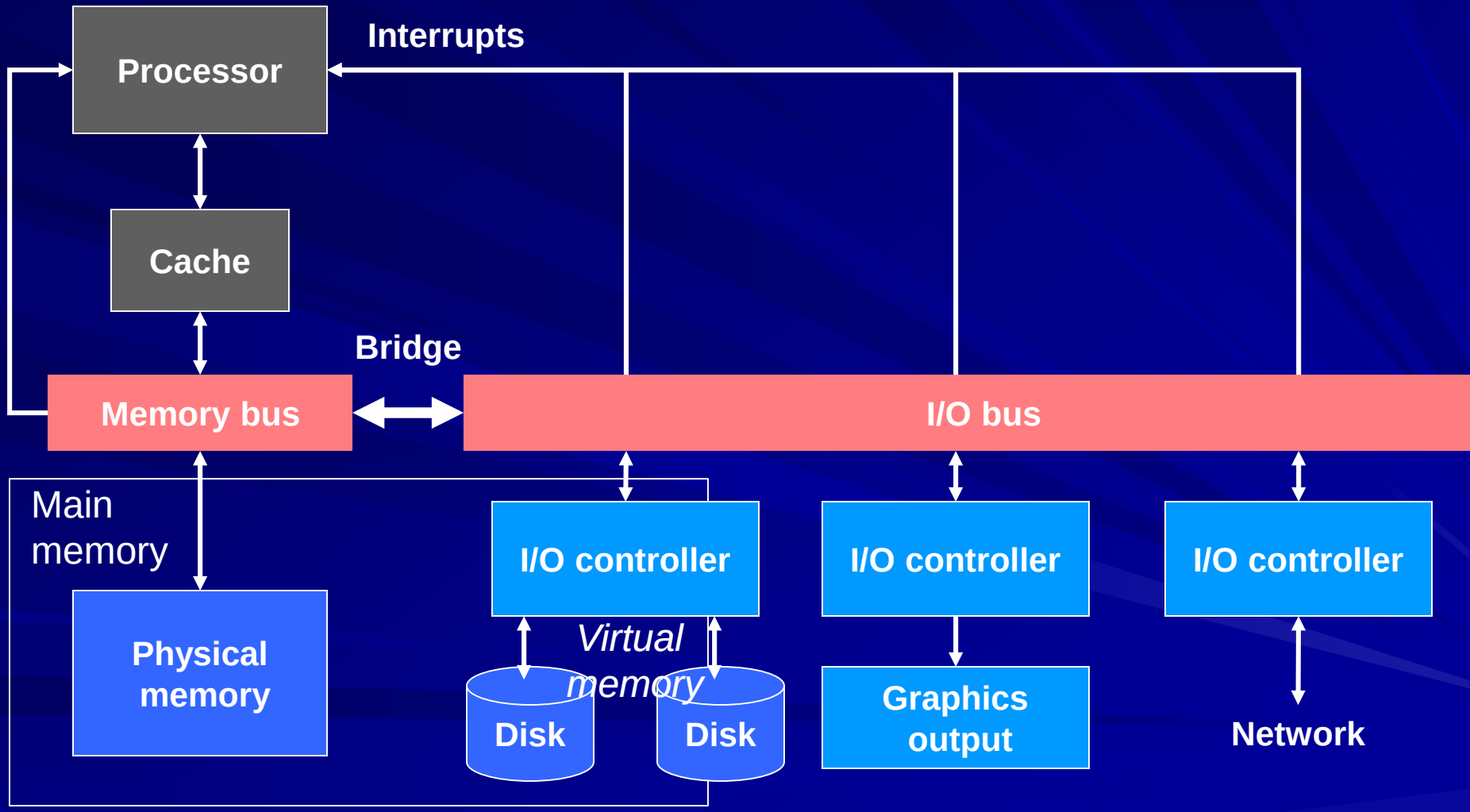
Christopher B. Harris
Assistant Professor
Department of Electrical and Computer Engineering
Auburn University, Auburn, AL 36849

(Adapted from slides by Vishwani D. Agrawal)

A Computer System



A Computer System



Amdahl's Law: Revisited

$$Speedup = \frac{Old\ time}{New\ time}$$

$$= \frac{1}{(1 - f_{enh}) + (f_{enh}/n)}$$

- We can speed up a computer by dividing the processing work by a factor of n. Thus, we enter the age of parallelism.

Three Types of Parallelism

- Instruction level parallelism (ILP)
- Data level parallelism
- Thread level parallelism

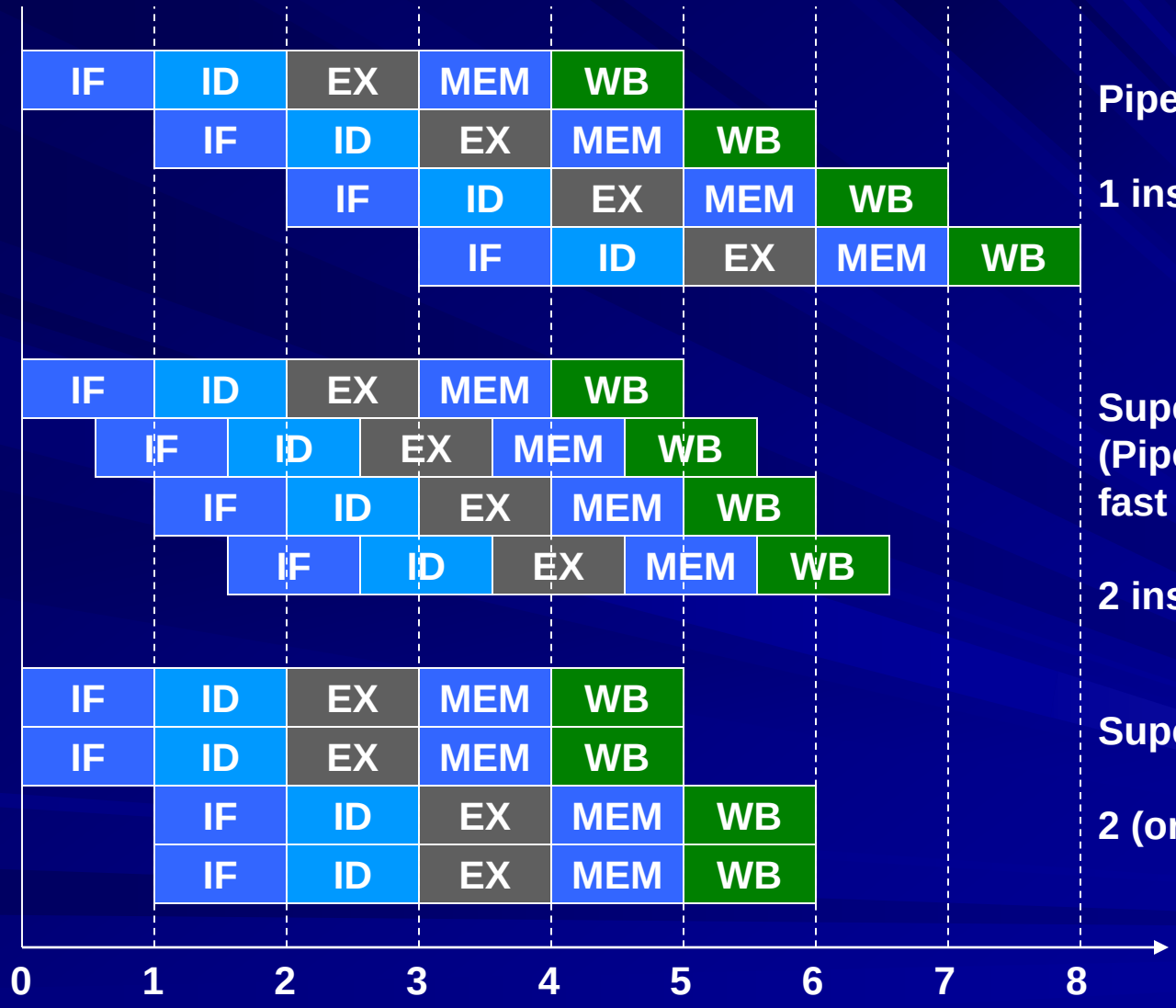
Nomenclature of Parallelism

- Single Instruction Single Data (SISD)
  - One instruction, one set of operands (data)
- Single Instruction Multiple Data (SIMD)
  - Single instruction, multiple sets of operands
- Multiple Instruction Multiple Data (MIMD)
  - Different instructions and different operands
- Multiple Instruction, Single Data (MISD)
  - Different instructions but same set of operands

Advanced Architectures – ILP

- Instruction level parallelism (ILP): multiple instructions fetched and executed simultaneously.
- ILP is used in addition to pipelining.
- Processors with ILP are called *multiple-issue processors* – multiple instructions launched in 1 clock cycle. Two ways:
  - MIMD: Multiple Instructions Multiple Data
    - Superpipeline
    - Superscalar – dynamic multiple issue
    - Very long instruction word (VLIW) – static multiple issue
  - SIMD: Single Instruction Multiple Data
    - Vector processor

Superpipeline and Superscalar



Pipeline

1 instruction/cycle

Superpipeline
(Pipeline clock is twice as fast as the system clock)

2 instructions per cycle

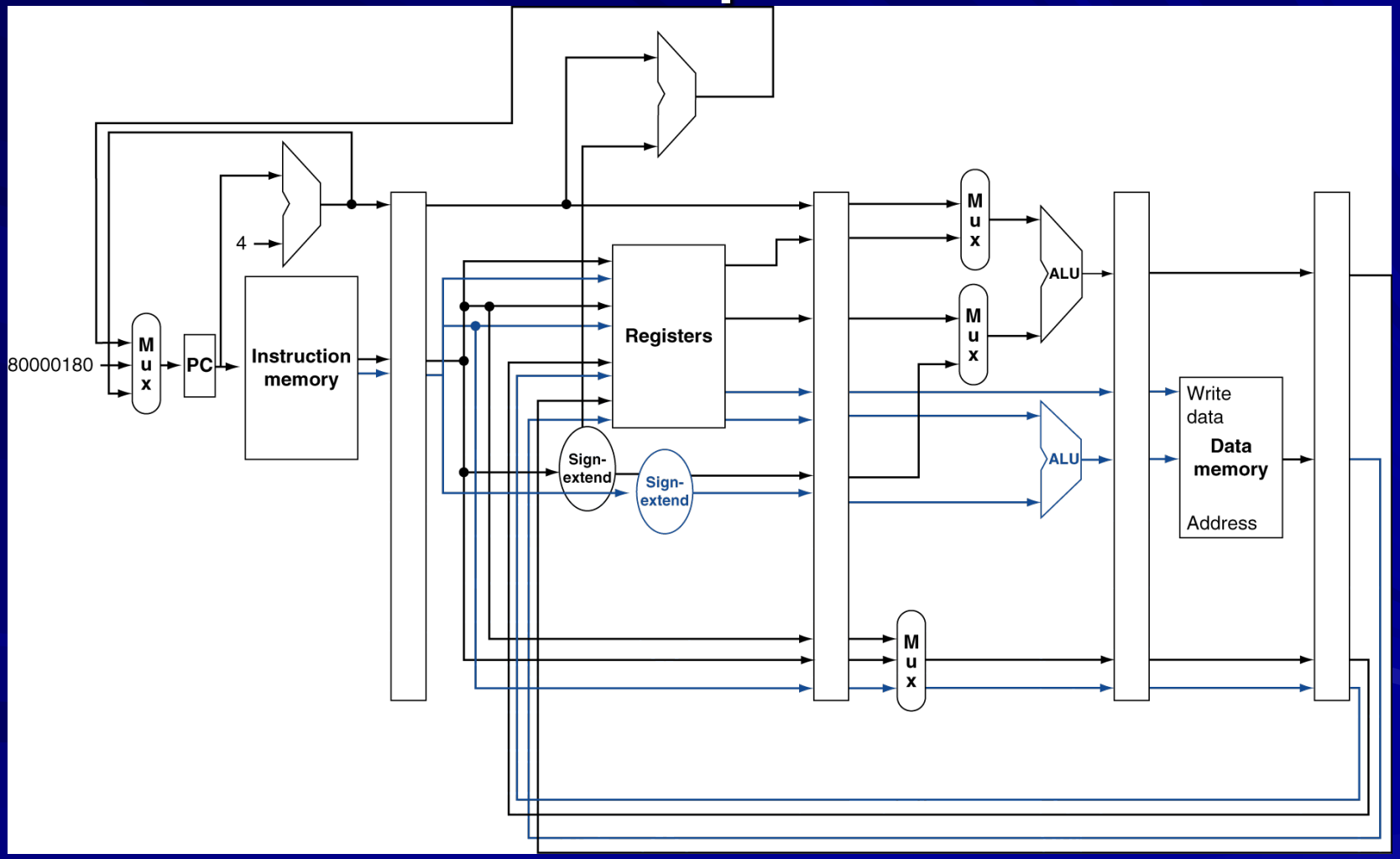
Superscalar

2 (or more) instructions/cycle

A Static Two-Issue MIPS Pipeline

- Read two instructions per cycle:
  - An ALU or branch instruction, and
  - A load or store instruction
  - Insert one nop if above pair is not available
- Added hardware (Figure 4.69, page 336):
  - A second instruction memory
  - Additional input/output ports in register file
  - Additional ALU in execute stage for address calculation

Static Dual-Issue MIPS Datapath



An Example (Page 337)

```
Loop:   lw      $t0, 0($s1)
        addu    $t0, $t0, $s2
        sw      $t0, 0(s1)
        addi    $s1, $s1, – 4
        bne     $s1, $0, Loop
```

Static Two-Issue Execution

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

Note code reordering and change in sw argument.

CPI = 4/5 = 0.8 > 0.5 (ideal)

Loop Unrolling (Index Multiple of 4)

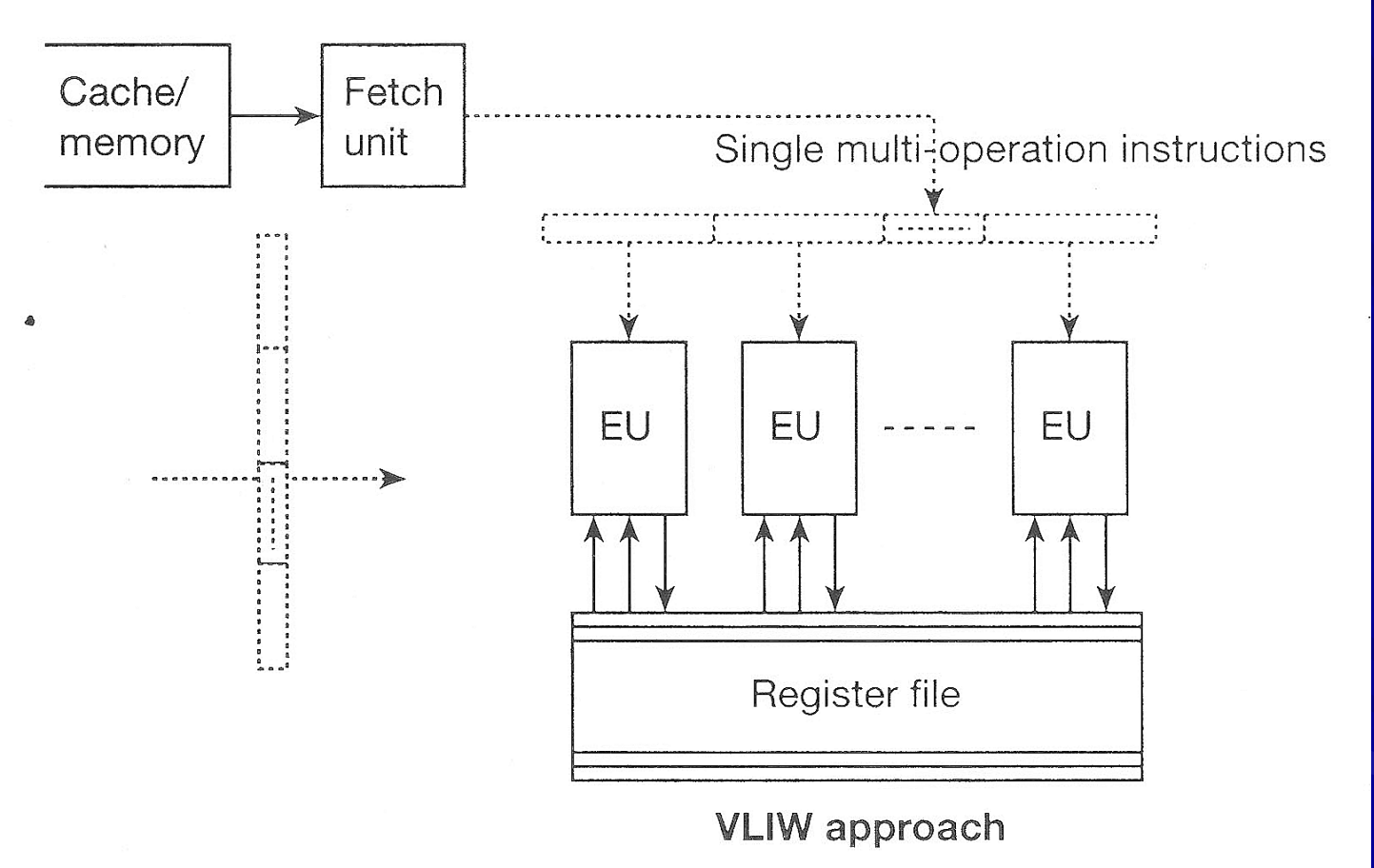
	ALU or branch instruction	Data transfer instruction	Clock cycle
Loop:	addi $s1, $s1, – 16	lw $t0, 0($s1)	1
	nop	lw $t1, 12($s1)	2
	addu $t0, $t0, $s2	lw $t2, 8($s1)	3
	addu $t1, $t1, $s2	lw $t3, 4($s1)	4
	addu $t2, $t2, $s2	sw $t0, 16($s1)	5
	addu $t3, $t3, $s2	sw $t1, 12($s1)	6
	nop	sw $t2, 8($s1)	7
	bne $s1, $0, Loop	sw $t3, 4($s1)	8

CPI = 8/14 = 0.57 > 0.5 (ideal)

VLIW: Very Long Instruction Word

- Static multiple issue architecture.
- Datapath contains multiple parallel execution units connected to a large register file.
- Instructions with multiple operations are packed into very long words of a wide instruction memory.

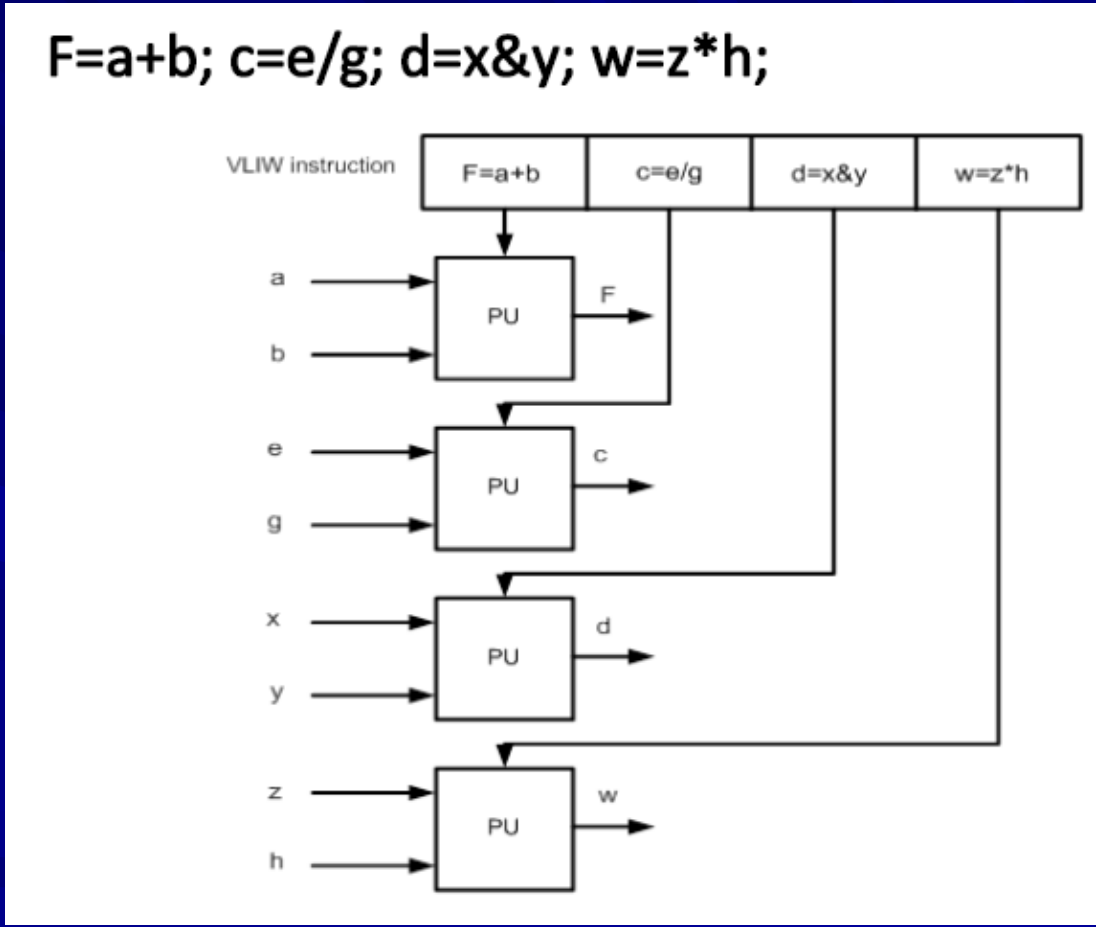
Ideal VLIW Processor



Typical instruction word length ranges from 52 bits to 1 Kbits.

VLIW Instruction Example

- Compiler groups instructions that have no data or resource conflicts for parallel execution.
- Speedup is highly program dependent.
- VLIW machines are examples of MIMD.



Register Files for VLIW

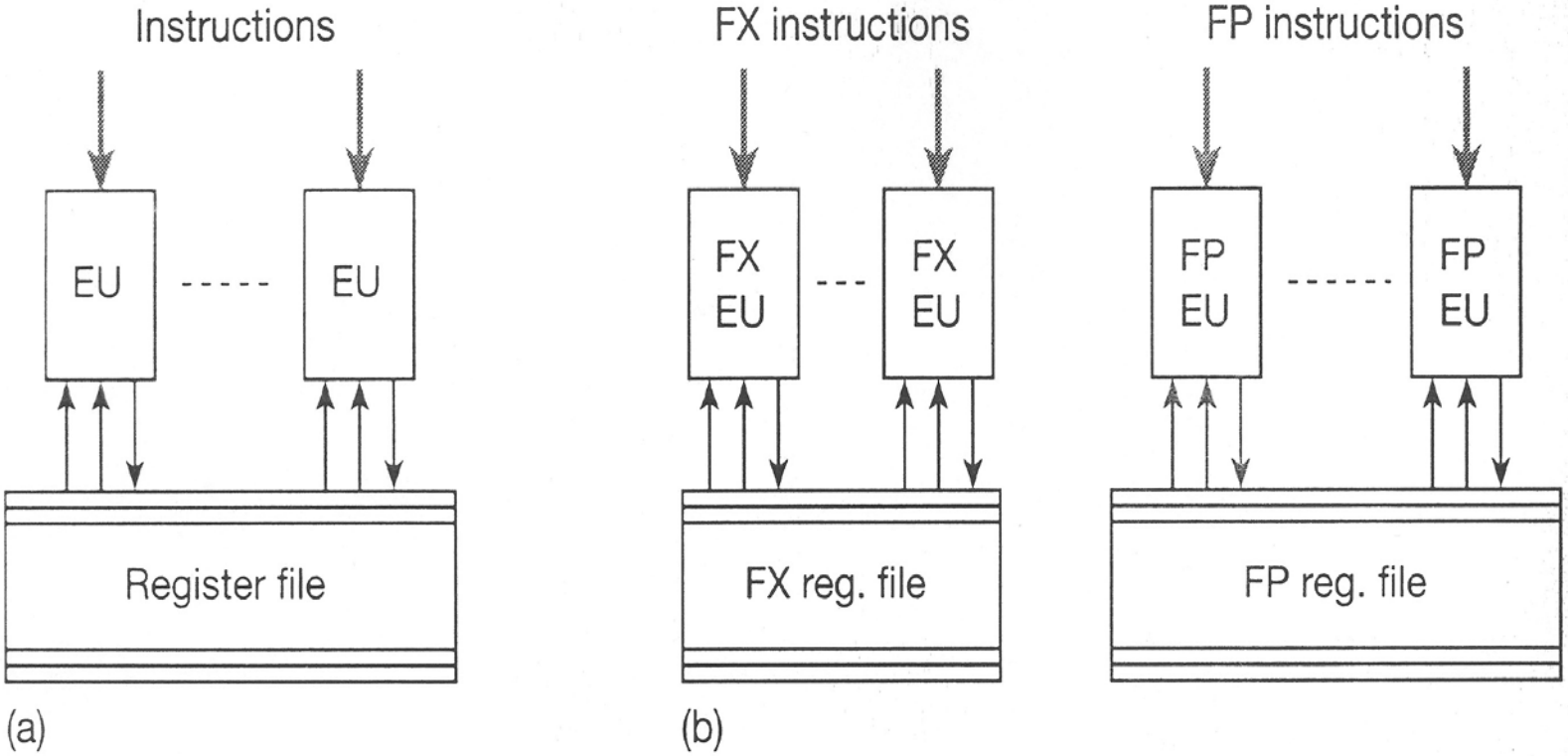
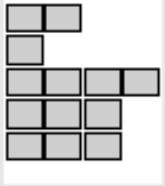
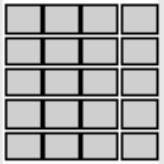
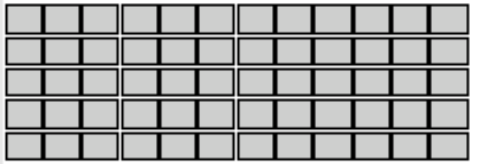


Figure 6.1 Common basic structure of superscalar and VLIW architectures (a) Unified register file; (b) split register file.

A major challenge in register file design for VLIW: R/W ports!

VLIW vs Other Architectures

ARCHITECTURE CHARACTERISTIC	CISC	RISC	VLIW
INSTRUCTION SIZE	Varies	One size, usually 32 bits	One size
INSTRUCTION FO RMAT	Field placement varies	Regular, consistent placement of fields	Regular, consistent placement of fields
INSTRUCTION SEMANTICS	Varies from simple to complex; possibly many dependent operations per instruction	Almost always one simple operation	Many simple, independent operations
REGISTERS	Few, sometimes special	Many, general-purpose	Many, general-purpose
MEMORY REFE RENCES	Bundled with operations in many different types of instructions	Not bundled with operations, i.e., load/store architecture	Not bundled with operations, i.e., load/store architecture
HARDWARE DESIGN FOCUS	Exploit microcoded implementations	Exploit implementations with one pipeline and & no microcode	Exploit implementations with multiple pipelines, no microcode & no complex dispatch logic
PICTURE OF FIVE TYPICAL INSTRU CTIONS  = I BYTE			

Code Expansion in VLIW

- It is found that code in VLIW is expanded roughly by a factor of three.
- For “long” VLIW, more opcode fields will be empty. This will result in wasting bandwidth and storage space.

VLIW: Further Reading

- J. A. Fisher, “Very Long Instruction Word Architecture and ELI-512,” *Proc. 10th Symp. on Computer Architecture*, Stockholm, June 1983, pp. 478-490.

- J. A. Fisher, P. Faraboschi and C. Young, Embedded Computing: *A VLIW Approach to Architecture, Compilers and Tools,* Morgan Kaufmann.

Limitations of Static Multi-Issue

- In VLIW (static multi-issue) we rely on the compiler to find operations we can do in parallel.
- Compilers can only find so much parallel work to do because of dependencies.

Dynamic Multi-Issue

- Let the CPU decide whether to issue 1, 2, 3, etc instructions each cycle.
  - Avoiding structural and data hazards.
- Can do better than compiler only multi-issue.
- Dynamic multi-issue processors are called “superscalar”.

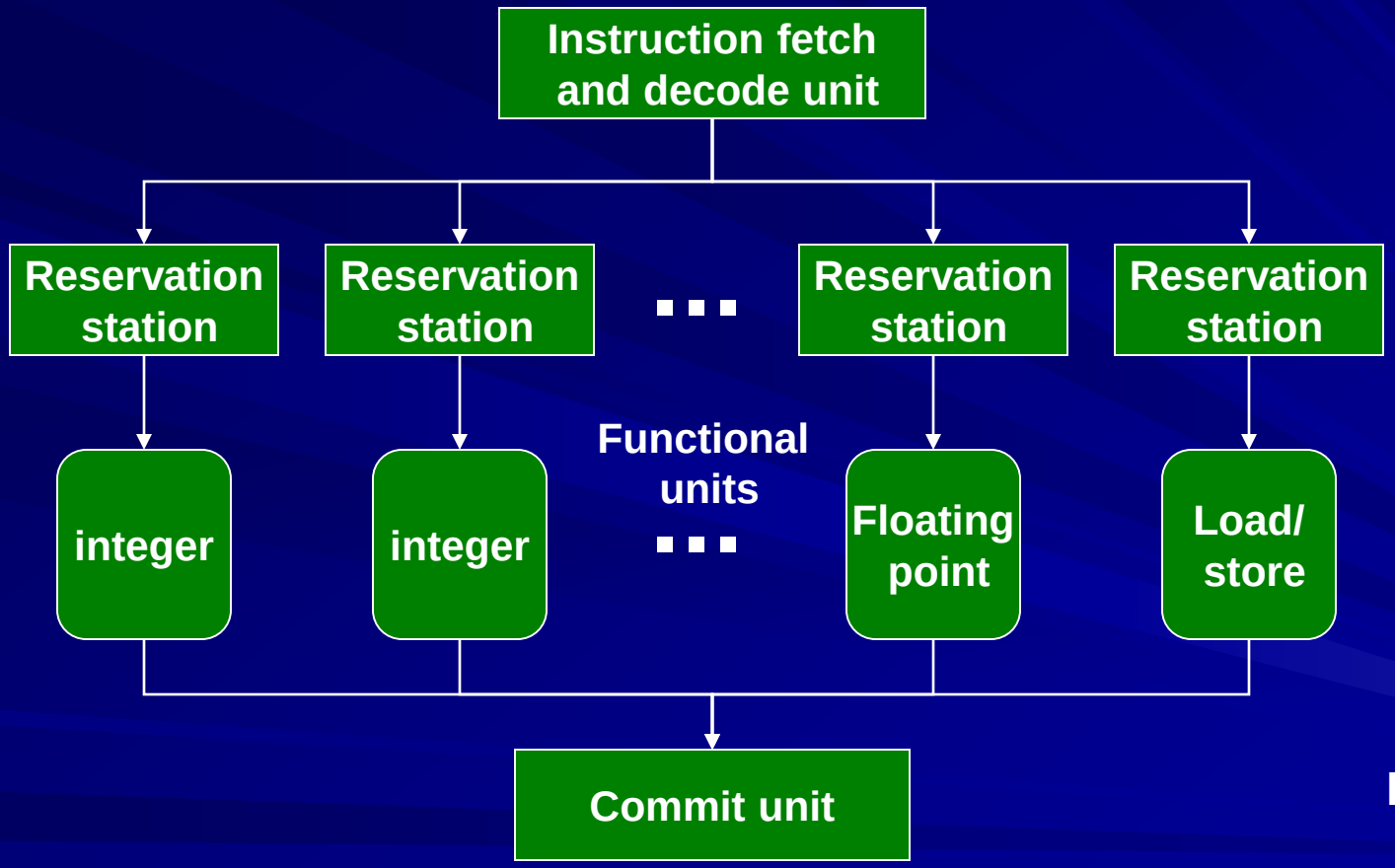
Dynamic Instruction Scheduling

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

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

  - We can start sub while add is waiting for lw

Superscalar: Dynamic Scheduling and Out-of-Order Execution



In-order issue

Out-of-order issue

Out-of-order execution

In-order commit (retire)

Out of Order Execution (OOE)

- A procedural programming language sequences instructions.
- Sequencing assumes an order of execution – no parallelism.
- OOE must preserve correctness of result.
- Principle: Two instructions can be executed in parallel if they do not have dependences.

Two Types of Dependencies

- Data dependence
- Name dependence

Data Dependence

- Instruction j is data dependent on instruction i if:
  - Instruction i produces a result that may be used by instruction j
  - Instruction j is data dependent on instruction k and instruction k is data dependent on instruction i
- Dependent instructions cannot be executed simultaneously

Data Dependence

- Dependencies are a property of programs
- Pipeline organization determines if dependence is detected and if it causes a stall
- Data dependence conveys:
  - Possibility of a hazard
  - Order in which results must be calculated
  - Upper bound on exploitable instruction level parallelism
- Dependencies that flow through memory locations are difficult to detect

Name Dependence

- Two instructions use the same name but no flow of information
  - Not a true data dependence, *but is a problem when reordering instructions*
  - *Antidependence*: instruction j writes a register or memory location that instruction i reads
    - Initial ordering (i before j) must be preserved
  - *Output dependence*: instruction i and instruction j write the same register or memory location
    - Ordering must be preserved
- To resolve, use register renaming techniques

Data Hazards Revisited

- Recall that data hazards are caused by dependencies.
- Data Hazards – 3 types
  - Read after write (RAW)
  - Write after write (WAW)
  - Write after read (WAR)
- Given two instructions X, and Y (in that order)…
  - RAW – Y attempts to read a register before X writes to it, so Y gets the old value.
  - WAW – Y attempts to write a register before it is written by X, thus writes happen in the wrong order.
  - WAR – Y attempts to write to a register before it is read by X, thus X gets an incorrect (new) value.

RAW Dependence

- Read after write (RAW): A dependent instruction reads from a register being written to by another instruction.
- Example:

```
add  $s1, $s2, $s3
sub  $s2, $s1, $s3
```

sub has RAW dependence on add (there is an actual need for communication)

WAR Dependence

- Write after read (WAR): An instruction writes to a register before it is read by a dependent instruction.
- Example:

```
add  $s1, $s2, $s3
sub  $s2, $s1, $s3
```

sub has an antidependence on add. This results from the reuse of the name $s2.

WAW Dependence

- Read after write (RAW): One instruction writes to a register to being written to by another instruction.
- Example:

```
add  $s2, $s2, $s3
sub  $s2, $s1, $s3
```

sub has an output dependence on add. This also results from the reuse of the name $s2.

Register Renaming

- Example:

DIV.D F0,F2,F4
ADD.D F6,F0,F8
S.D F6,0(R1)
SUB.D F8,F10,F14
MUL.D F6,F10,F8

+ name dependence with F6

Register Renaming

- Example:

  DIV.D F0,F2,F4
  ADD.D S,F0,F8
  S.D S,0(R1)
  SUB.D T,F10,F14
  MUL.D F6,F10,T

- Now only RAW hazards remain, which can be strictly ordered

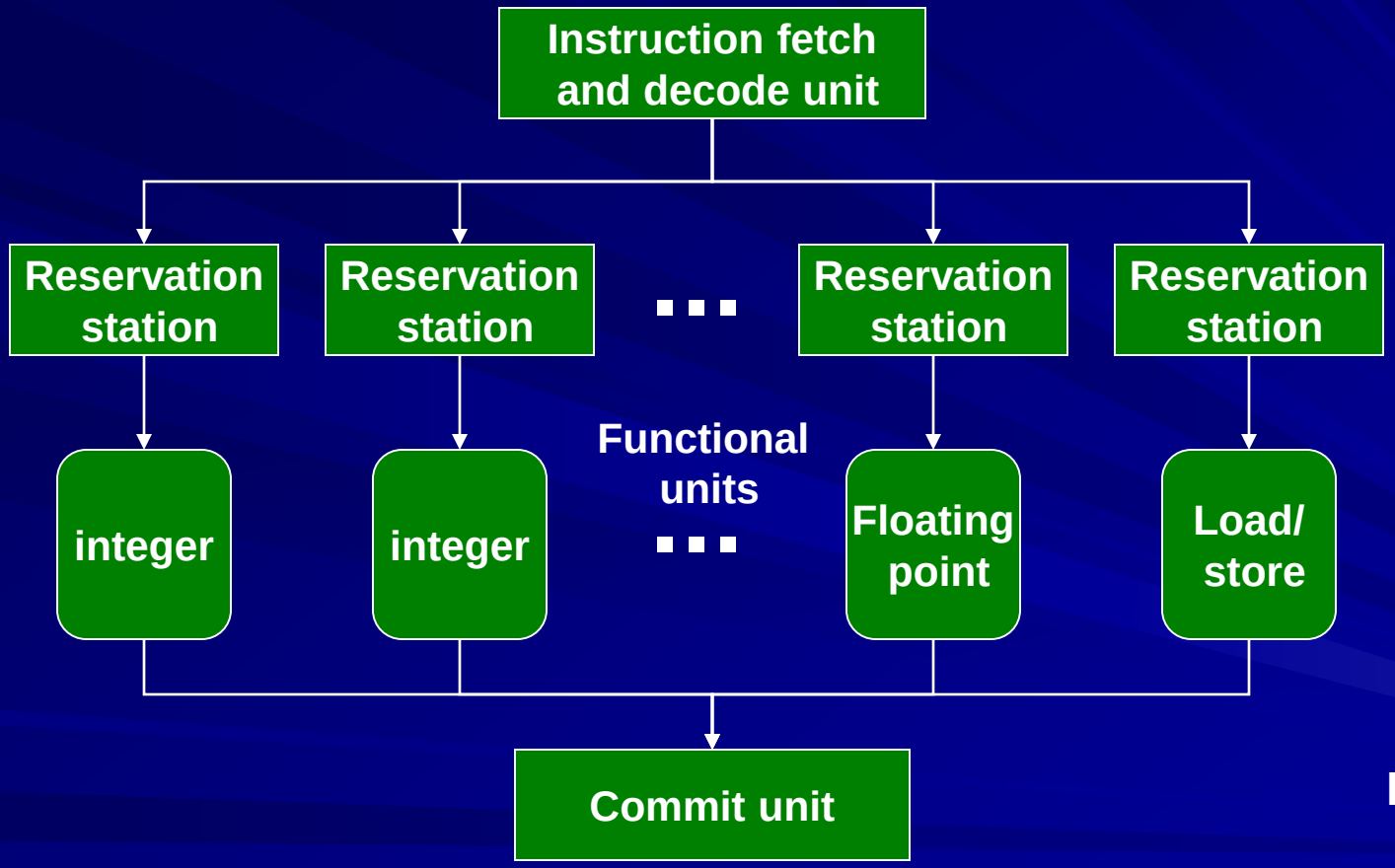
Register Renaming

- Register renaming is provided by reservation stations
  - Contains:
    - The instruction
    - Buffered operand values (when available)
    - Reservation station number of instruction providing the operand values
  - RS fetches and buffers an operand as soon as it becomes available (not necessarily involving register file)
  - Pending instructions designate the RS to which they will send their output
    - Result values broadcast on a result bus, called the common data bus (CDB)
  - Only the last output updates the register file
  - As instructions are issued, the register specifiers are renamed with the reservation station
  - There may be more reservation stations than registers

Superscalar Instruction Scheduling

- Can execute multiple instructions at once if there is not a dependency.
- One technique: Scoreboarding
- Characteristics:
  - RAW dependence – Resolved during register read phase.
  - WAR dependence – If the result register is being read, stall during write back.
  - WAW dependence – If the result register is being written, do not issue.

Superscalar: Dynamic Scheduling and Out-of-Order Execution

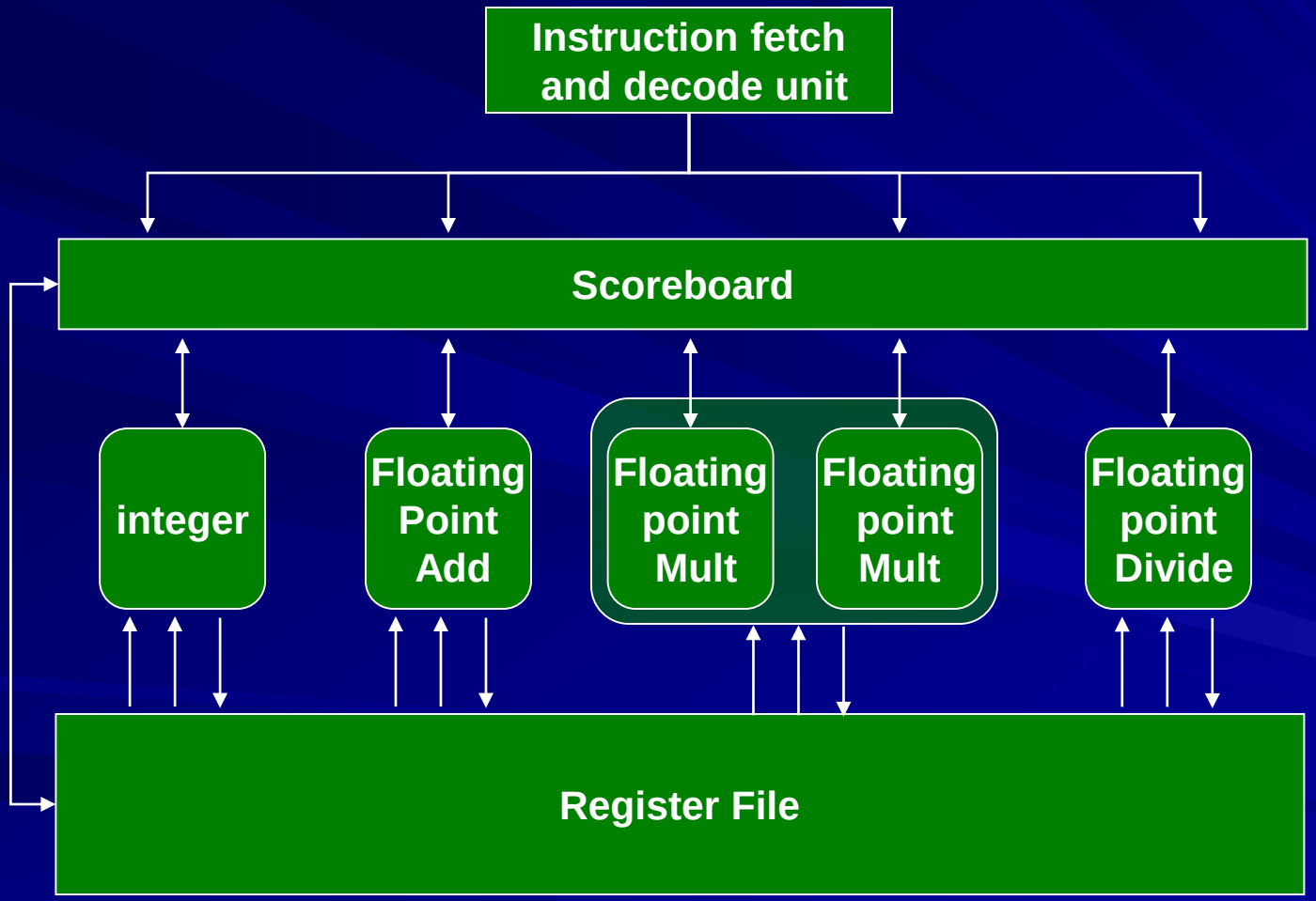


In-order issue

Out-of-order execution

In-order commit (retire)

Superscalar: Dynamic Scheduling with Scoreboarding



In-order issue

Out-of-order execution

commit (retire)

Scoreboarding

- A “scoreboard” is a table which keeps a cycle by cycle record of registers and execution units showing how many instructions are using them.
- The scoreboard has 3 parts…

Components of a Scoreboard

- Instruction status:
  - Indicates which of the 4 steps the instruction is in.
- Functional unit status:
  - Indicates state of the functional unit (9 fields per FU).
    - Busy: Is the FU currently being used
    - Op: Operation unit is performing
    - Fj, Jk: Source registers
    - Qj, Qk: FUs producing source registers
    - Rj, Rk: Flags indicating if Fj and Fk are available but not read
- Register result status:
  - Indicates which FU will write which register.

Instruction Status

Instruction j k

Issue	Read Operands	Execute Complete	Write Results

Functional Unit Status

Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj rdy? Rj	Fk rdy? Rk
Integer									
Mult 1									
Mult 2									
Add									
Divide									

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU									

4 Stages of Scoreboarding: (1) Instruction Issue

- Decode instructions and check for structural hazards:
  - If a FU is free and no other active instruction has the same destination register (WAW) then…
    - Issue instruction to FU update scoreboard
  - If a structural or WAW hazard exists then …
    - Stall instruction
    - No further instructions can issue until hazard(s) clear(s)
- Key steps:
  - In order issue, multiple issues per cycle, check for hazards.

4 Stages of Scoreboarding: (2) Read Operands

- Wait until no data hazards then read operands:
  - Operand is available if no previous instruction is going to write it (or if it is currently being written)
  - If operand is available then…
    - Read operands and send to proper FU to begin execution
    - Instructions can be sent OOO
    - RAW hazards are resolved dynamically in this step
- Key steps:
  - Wait for operands, Check reg results status (RAW), send operands to FU.

4 Stages of Scoreboarding: (3) Execution

- FU operates on operands:
  - Starts work upon receiving operands
  - Update scoreboard when complete
  - Each individual FU can be pipelined

4 Stages of Scoreboarding: (4) Write Results

- Finish instruction:
  - When FU is finished check for WAR hazards (stall if present)
  - If no hazards, write result to destination register
  - Update scoreboard since FU is now free

Instruction Status

Instruction j k

Issue	Read Operands	Execute Complete	Write Results

Functional Unit Status

Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj rdy? Rj	Fk rdy? Rk
Integer									
Mult 1									
Mult 2									
Add									
Divide									

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU									

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Read Operands	Execute Complete	Write Results
1			

Clock Cycle 1

Functional Unit Status

Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj rdy? Rj	Fk rdy? Rk
Integer	Yes	Load	F6		R2				Yes
Mult 1	No								
Mult 2	No								
Add	No								
Divide	No								

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU				Int					

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Read Operands	Execute Complete	Write Results
1	2		

Clock Cycle 2

Functional Unit Status

Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj rdy? Rj	Fk rdy? Rk
Integer	Yes	Load	F6		R2				Yes
Mult 1	No								
Mult 2	No								
Add	No								
Divide	No								

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU				Int					

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Read Operands	Execute Complete	Write Results
1	2	3	4

Clock Cycle 4

Functional Unit Status

Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj rdy? Rj	Fk rdy? Rk
Integer	Yes	Load	F6		R2				No
Mult 1	No								
Mult 2	No								
Add	No								
Divide	No								

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU				Int					

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Read Operands	Execute Complete	Write Results
1	2	3	4
5			

Clock Cycle 5

Functional Unit Status

Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj rdy? Rj	Fk rdy? Rk
Integer	Yes	Load	F2		R3				Yes
Mult 1	No								
Mult 2	No								
Add	No								
Divide	No								

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU		Int							

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Read Operands	Execute Complete	Write Results
1	2	3	4
5	6		
6			

Clock Cycle 6

Functional Unit Status

Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj rdy? Rj	Fk rdy? Rk
Integer	Yes	Load	F2		R3				Yes
Mult 1	Yes	Mult	F0	F2	F4	Int		No	Yes
Mult 2	No								
Add	No								
Divide	No								

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU	Mult1	Int							

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Read Operands	Execute Complete	Write Results
1	2	3	4
5	6	7	
6			
7			

Clock Cycle 7

Functional Unit Status

Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj rdy? Rj	Fk rdy? Rk
Integer	Yes	Load	F2		R3				No
Mult 1	Yes	Mult	F0	F2	F4	Int		No	Yes
Mult 2	No								
Add	Yes	Sub	F8	F6	F2		Int	Yes	No
Divide	No								

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU	Mult1	Int			Add				

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Read Operands	Execute Complete	Write Results
1	2	3	4
5	6	7	
6			
7			
8			

Clock Cycle 8 part a

Functional Unit Status

Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj rdy? Rj	Fk rdy? Rk
Integer	Yes	Load	F2		R3				No
Mult 1	Yes	Mult	F0	F2	F4	Int		No	Yes
Mult 2	No								
Add	Yes	Sub	F8	F6	F2		Int	Yes	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU	Mult1	Int			Add	Div			

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Read Operands	Execute Complete	Write Results
1	2	3	4
5	6	7	8
6			
7			
8			

Clock Cycle 8 part b

Functional Unit Status

Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj rdy? Rj	Fk rdy? Rk
Integer	No								
Mult 1	Yes	Mult	F0	F2	F4	Int		Yes	Yes
Mult 2	No								
Add	Yes	Sub	F8	F6	F2		Int	Yes	Yes
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU	Mult1				Add	Div			

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Read Operands	Execute Complete	Write Results
1	2	3	4
5	6	7	8
6	9		
7	9		
8			

Clock Cycle 9

Functional Unit Status

Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj rdy? Rj	Fk rdy? Rk
Integer	No								
Mult 1	Yes	Mult	F0	F2	F4	Int		Yes	Yes
Mult 2	No								
Add	Yes	Sub	F8	F6	F2		Int	Yes	Yes
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU	Mult1				Add	Div			

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Read Operands	Execute Complete	Write Results
1	2	3	4
5	6	7	8
6	9		
7	9	11	
8			

Clock Cycle 11

Functional Unit Status

Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj rdy? Rj	Fk rdy? Rk
Integer	No								
Mult 1	Yes	Mult	F0	F2	F4	Int		No	No
Mult 2	No								
Add	Yes	Sub	F8	F6	F2		Int	No	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU	Mult1				Add	Div			

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Read Operands	Execute Complete	Write Results
1	2	3	4
5	6	7	8
6	9		
7	9	11	12
8			

Clock Cycle 12

Functional Unit Status

Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj rdy? Rj	Fk rdy? Rk
Integer	No								
Mult 1	Yes	Mult	F0	F2	F4	Int		No	No
Mult 2	No								
Add	No								
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU	Mult1					Div			

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Read Operands	Execute Complete	Write Results
1	2	3	4
5	6	7	8
6	9		
7	9	11	12
8			
13			

Clock Cycle 13

Functional Unit Status

Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj rdy? Rj	Fk rdy? Rk
Integer	No								
Mult 1	Yes	Mult	F0	F2	F4	Int		No	No
Mult 2	No								
Add	Yes	Add	F6	F8	F2			Yes	Yes
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU	Mult1			Add		Div			

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Read Operands	Execute Complete	Write Results
1	2	3	4
5	6	7	8
6	9		
7	9	11	12
8			
13	14		

Clock Cycle 14

Functional Unit Status

Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj rdy? Rj	Fk rdy? Rk
Integer	No								
Mult 1	Yes	Mult	F0	F2	F4	Int		No	No
Mult 2	No								
Add	Yes	Add	F6	F8	F2			Yes	Yes
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU	Mult1			Add		Div			

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Read Operands	Execute Complete	Write Results
1	2	3	4
5	6	7	8
6	9	16	
7	9	11	12
8			
13	14	16	

Clock Cycle 16

Functional Unit Status

Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj rdy? Rj	Fk rdy? Rk
Integer	No								
Mult 1	Yes	Mult	F0	F2	F4	Int		No	No
Mult 2	No								
Add	Yes	Add	F6	F8	F2			No	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU	Mult1			Add		Div			

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Read Operands	Execute Complete	Write Results
1	2	3	4
5	6	7	8
6	9	16	17
7	9	11	12
8	17		
13	14	16	

Clock Cycle 17

Functional Unit Status

Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj rdy? Rj	Fk rdy? Rk
Integer	No								
Mult 1	No								
Mult 2	No								
Add	Yes	Add	F6	F8	F2			No	No
Divide	Yes	Div	F10	F0	F6	Mult1		Yes	Yes

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU				Add		Div			

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Read Operands	Execute Complete	Write Results
1	2	3	4
5	6	7	8
6	9	16	17
7	9	11	12
8	17		
13	14	16	18

Clock Cycle 18

Functional Unit Status

Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj rdy? Rj	Fk rdy? Rk
Integer	No								
Mult 1	No								
Mult 2	No								
Add	No								
Divide	Yes	Div	F10	F0	F6	Mult1		No	No

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU						Div			

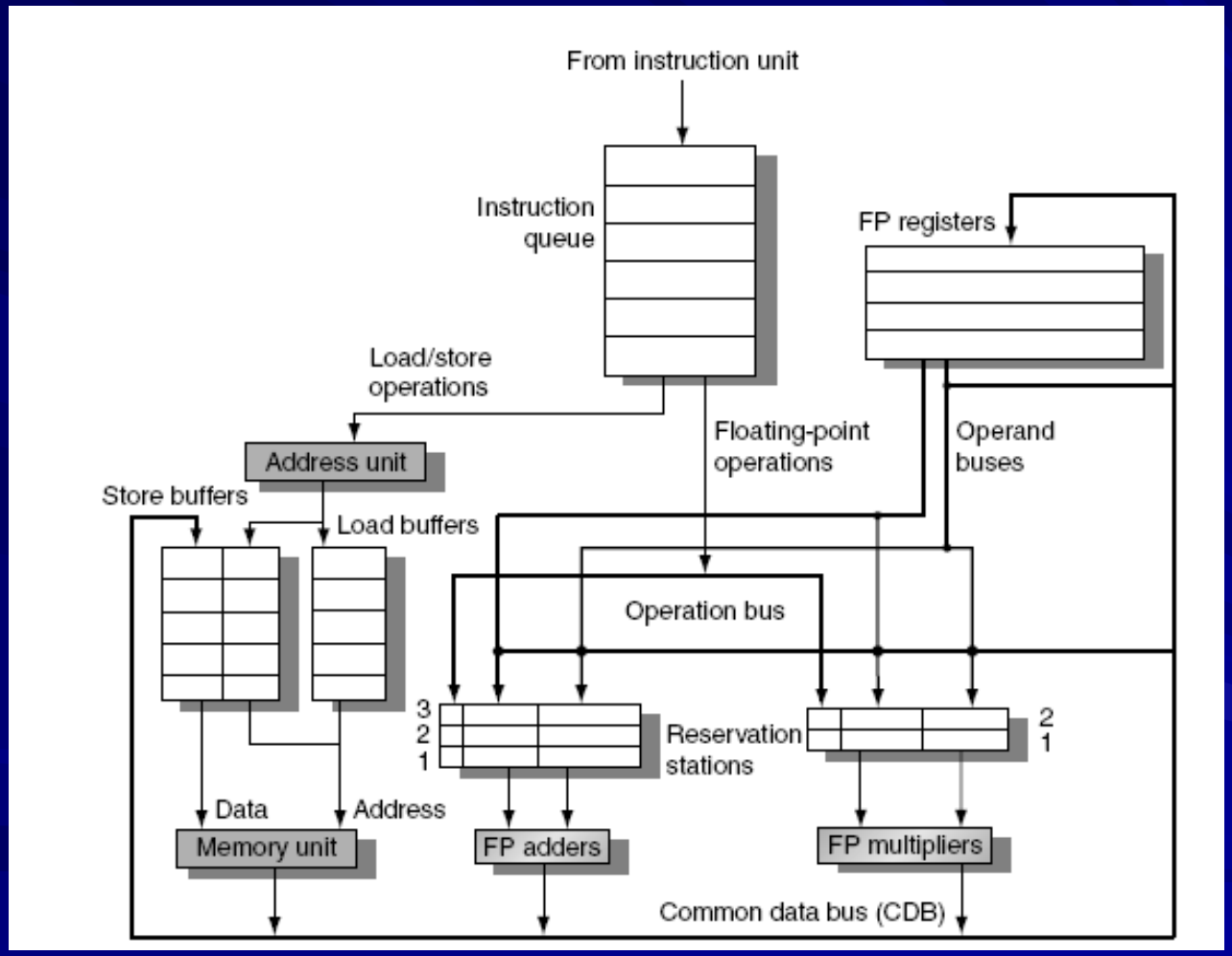
Summary of Scoreboarding Approach

- Limitations:
  - Small number of instructions considered at once (instruction window)
  - Relatively small number of FUs
  - Does not issue if WAW or structural hazard
  - Waits for WAR hazard
- Key idea: Allows instructions behind stall to proceed
  - Decode -> issue instructions, read operands
  - Enables OOO execution

Tomasulo's Algorithm: Another Method of Dynamic Scheduling

- Scoreboarding resolved RAW hazards during register read but still had to stall for WAW and WAR hazards.
- Tomasulo's Algorithm uses register renaming to reduce WAW and WAR hazards.

Tomasulo's Algorithm: Three Steps



1. In-order issue

2. Out-of-order execution

3. Write result

Tomasulo's Algorithm: Step 1

- Issue
  - Get next instruction from FIFO queue
  - If RS is available, issue instruction to RS with operands (if available)
  - If operands not available, stall instruction

Tomasulo's Algorithm: Step 2

- Execute
  - If operand is not ready, watch the CDB for it
  - When operand becomes available, store it in any reservation station waiting for it
  - When all operands are ready, send to FU
  - Loads / Stores are always maintained in program order
  - No instruction is allowed to start execution until all preceding branches are resolved

Tomasulo's Algorithm: Step 3

- Write Result
  - Broadcast result on common data bus to reservation stations and store buffers

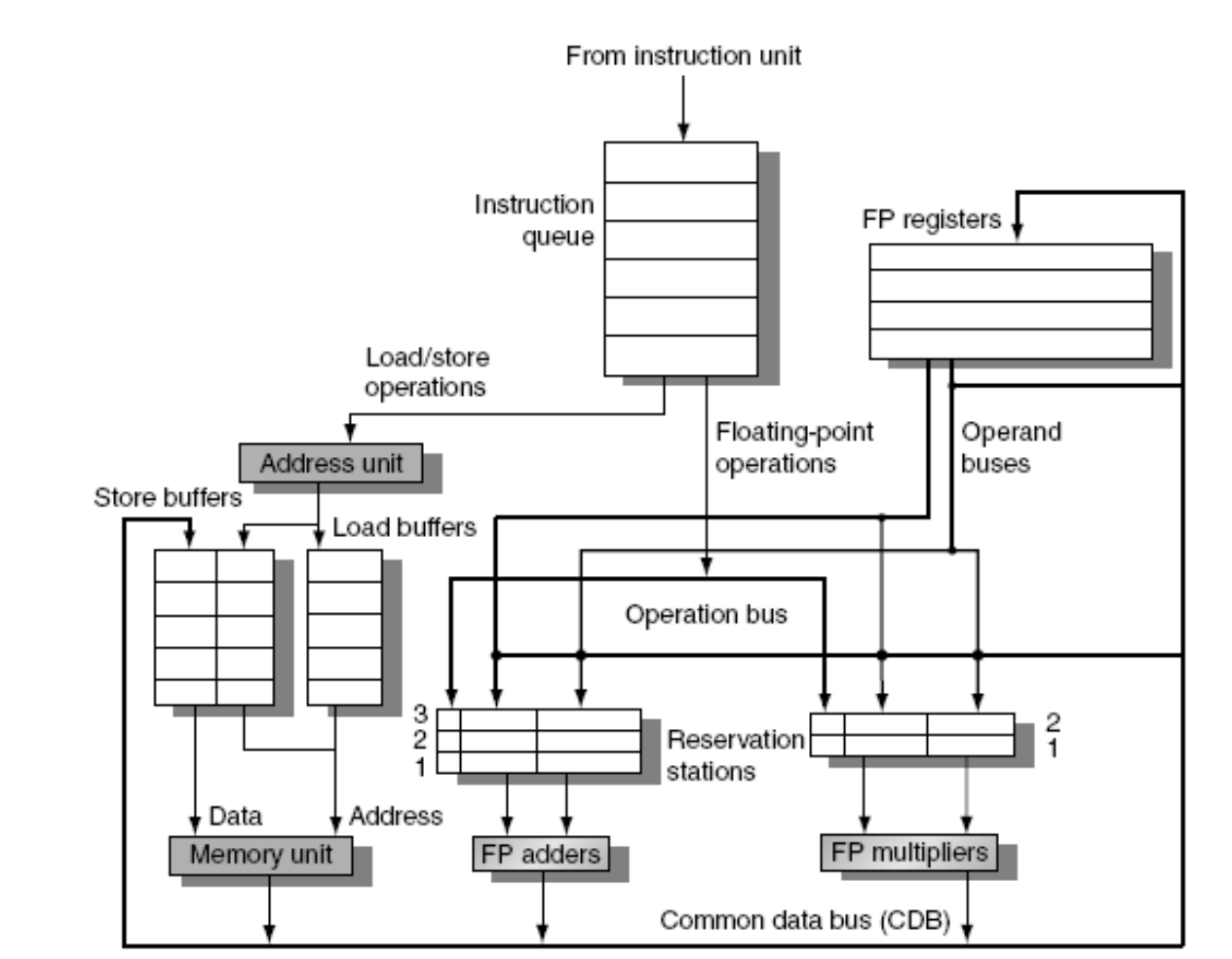
Note: Store instructions must wait until both address and data value are ready.

Reservation Station Components

- Reservation Station Fields:
  - Busy: Is the FU currently being used
  - Op: Operation unit is performing
  - Qj, Qk: FUs producing source registers
  - Vj, Vk: Value of source operands (not reg #)
  - Rj, Rk: Flags indicating if Vj and Vk are ready

- Register result status:
  - Indicates which FU will write which register.

Tomasulo's Algorithm: Example



1. In-order issue

2. Out-of-order execution

3. Write result

Instruction Status

Instruction j k

Issue	Execute Complete	Write Results

	Busy	Addr
Load 1		
Load 2		
Load 3		

Reservation Stations

Name	Busy	Op	S1 Vj	S2 Vk	FU for j Qj	FU for k Qk
Add1						
Add 2						
Add 3						
Mult 1						
Mult 2						

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU									

Instruction Status

Instruction		j	k
L.D	F6,	34	(R2)
L.D	F2,	45	(R3)
MUL.D	F0,	F2,	F4
SUB.D	F8,	F6,	F2
DIV.D	F10,	F0,	F6
ADD.D	F6,	F8,	F2

Issue	Execute Complete	Write Results
1		

	Busy	Addr
Load 1	Yes	34+R2
Load 2	No	
Load 3	No	

Clock Cycle 1

Reservation Stations

Name	Busy	Op	S1 Vj	S2 Vk	FU for j Qj	FU for k Qk
Add1						
Add 2						
Add 3						
Mult 1						
Mult 2						

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU				Load1					

Instruction Status

Instruction		j	k
L.D	F6,	34	(R2)
L.D	F2,	45	(R3)
MUL.D	F0,	F2,	F4
SUB.D	F8,	F6,	F2
DIV.D	F10,	F0,	F6
ADD.D	F6,	F8,	F2

Issue	Execute Complete	Write Results
1		
2		

	Busy	Addr
Load 1	Yes	34+R2
Load 2	Yes	45+R3
Load 3	No	

Clock Cycle 2

Reservation Stations

Name	Busy	Op	S1 Vj	S2 Vk	FU for j Qj	FU for k Qk
Add1						
Add 2						
Add 3						
Mult 1						
Mult 2						

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU		Load2		Load1					

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Execute Complete	Write Results
1	3	
2		
3		

	Busy	Addr
Load 1	Yes	34+R2
Load 2	Yes	45+R3
Load 3	No	

Clock Cycle 3

Reservation Stations

Name	Busy	Op	S1 Vj	S2 Vk	FU for j Qj	FU for k Qk
Add1						
Add 2						
Add 3						
Mult 1	Yes	Mult		R(F4)	Load2	
Mult 2						

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU	Mult1	Load2		Load1					

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Execute Complete	Write Results
1	3	4
2	4	
3		
4		

	Busy	Addr
Load 1	No	
Load 2	Yes	45+R3
Load 3	No	

Clock Cycle 4

Reservation Stations

Name	Busy	Op	S1 Vj	S2 Vk	FU for j Qj	FU for k Qk
Add1	Yes	Sub	M(34+R2)			Load2
Add 2						
Add 3						
Mult 1	Yes	Mult		R(F4)	Load2	
Mult 2						

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU	Mult1	Load2		M(34+R2)	Add1				

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Execute Complete	Write Results
1	3	4
2	4	5
3		
4		
5		

	Busy	Addr
Load 1	No	
Load 2	No	
Load 3	No	

Clock Cycle 5

Reservation Stations

Name	Busy	Op	S1 Vj	S2 Vk	FU for j Qj	FU for k Qk
Add1	Yes	Sub	M(34+R2)	M(45+R3)		
Add 2						
Add 3						
Mult 1	Yes	Mult	M(45+R3)	R(F4)		
Mult 2	Yes	Div		M(34+R2)	Mult1	

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU	Mult1	M(45+R3)			Add1	Mult2			

Instruction Status

Instruction	j	k
L.D F6,	34	(R2)
L.D F2,	45	(R3)
MUL.D F0,	F2,	F4
SUB.D F8,	F6,	F2
DIV.D F10,	F0,	F6
ADD.D F6,	F8,	F2

Issue	Execute Complete	Write Results
1	3	4
2	4	5
3		
4		
5		
6		

	Busy	Addr
Load 1	No	
Load 2	No	
Load 3	No	

Clock Cycle 6

Reservation Stations

Name	Busy	Op	S1 Vj	S2 Vk	FU for j Qj	FU for k Qk
Add1	Yes	Sub	M(34+R2)	M(45+R3)		
Add 2	Yes	Add		M(45+R3)	Add1	
Add 3						
Mult 1	Yes	Mult	M(45+R3)	R(F4)		
Mult 2	Yes	Div		M(34+R2)	Mult1	

Register Result Status

	F0	F2	F4	F6	F8	F10	F12	…	F30
FU	Mult1			Add2	Add1	Mult2			

Tomasulo's Summary

- Reservation Stations: renaming to a set of registers + buffering source operands
  - Prevents registers as bottleneck
  - Avoids WAR, WAW hazards of socreboarding
  - Allows loop unrolling in HW

Scoreboarding Vs Tomasulo's

- 4 stages vs 3 stages
- Central control vs distributed buffers
- Registers vs buffers / RS
- Stall on WAR/WAW vs renaming
- Forwarding vs Common data bus (CDB)


fin