INTRODUCTION TO ADVANCED PIPELINE

1 May 2020

Dr Noor Mahammad Sk

Review: Summary of Pipelining Basics

- Hazards limit performance
  - Structural: need more HW resources
  - Data: need forwarding, compiler scheduling
  - Control: early evaluation & PC, delayed branch, prediction
- Increasing length of pipe increases impact of hazards; pipelining helps instruction bandwidth, not latency
- Interrupts, instruction set, FP makes pipelining harder
- Compilers reduce cost of data and control hazards
  - Load delay slots
  - Branch delay slots
  - Branch prediction
- Today: Longer pipelines (R4000) → more instruction level parallelism → SW and HW loop unrolling

Case Study: MIPS R4000 (200MHz)

- 8 Stage Pipeline:
  - IF–first half of fetching of instruction; PC selection happens here as well as initiation of instruction cache access.
  - IS–second half of access to instruction cache.
  - RF–instruction decode and register fetch, hazard checking and also instruction cache hit detection.
  - EX–execution, which includes effective address calculation, ALU operation, and branch target computation and condition evaluation.
  - DF–data fetch, first half of access to data cache.
  - DS–second half of access to data cache.
  - TC–tag check, determine whether the data cache access hit.
  - WB–write back for loads and register-register operations.
- 8 Stages: What is impact on Load delay? Branch delay? Why?

Case Study: MIPS R4000

TWO Cycle
Load Latency

IF	IS	RF	EX	DF	$\boxed{\text{DS}}$	TC	WB
	IF	IS	RF	EX	DF	DS	TC
		IF	IS	RF	EX	DF	DS
			IF	IS	RF	$\boxed{\text{EX}}$	DF
				IF	IS	RF	EX
					IF	IS	RF
						IF	IS
							IF

THREE Cycle
Branch Latency

(conditions evaluated
during EX phase)

IF	IS	RF	$\boxed{\text{EX}}$	DF	DS	TC	WB
	IF	IS	RF	EX	DF	DS	TC
		IF	IS	RF	EX	DF	DS
			IF	IS	RF	EX	DF
				$\boxed{\text{IF}}$	IS	RF	EX
					IF	IS	RF
						IF	IS
							IF

Delay slot plus two stalls
Branch likely cancels delay slot if not taken

MIPS R4000 Floating Point

- FP Adder, FP Multiplier, FP Divider
- Last step of FP Multiplier/Divider uses FP Adder HW
- 8 kinds of stages in FP units:

Stage	*Functional unit*	*Description*
A	FP adder	Mantissa ADD stage
D	FP divider	Divide pipeline stage
E	FP multiplier	Exception test stage
M	FP multiplier	First stage of multiplier
N	FP multiplier	Second stage of multiplier
R	FP adder	Rounding stage
S	FP adder	Operand shift stage
U		Unpack FP numbers

MIPS FP Pipe Stages

FP Instr	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*…*
Add, Subtract	**U**	**S+A**	**A+R**	**R+S**					
Multiply	**U**	**E+M**	**M**	**M**	**M**	**N**	**N+A**	**R**	
Divide	**U**	**A**	**R**	**D^{28}**	**…**	**D+A**	**D+R, D+R, D+A, D+R, A, R**		
Square root	**U**	**E**	**$(A+R)^{108}$**		**…**	**A**	**R**		
Negate	**U**	**S**							
Absolute value	**U**	**S**							
FP compare	**U**	**A**	**R**						

Stages:

M	***First stage of multiplier***
N	***Second stage of multiplier***
R	***Rounding stage***
S	***Operand shift stage***
U	***Unpack FP numbers***
A	***Mantissa ADD stage***
D	***Divide pipeline stage***
E	***Exception test stage***

FP Loop: Where are the Hazards?

```
Loop:  LD      F0,0(R1)     ;F0=vector element
       ADDD    F4,F0,F2     ;add scalar from F2
       SD      0(R1),F4     ;store result
       SUBI    R1,R1,8      ;decrement pointer 8B (DW)
       BNEZ    R1,Loop      ;branch R1!=zero
       NOP                  ;delayed branch slot
```

Instruction producing result	Instruction using result	Latency in clock cycles
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
Integer op	Integer op	0

Where are the stalls?

FP Loop Hazards

```
Loop:   LD      F0, 0(R1)     ;F0=vector element
        ADDD    F4, F0, F2    ;add scalar from F2
        SD      0(R1), F4     ;store result
        SUBI    R1, R1, 8     ;decrement pointer 8 Bytes (DW)
        BNEZ    R1, Loop      ;branch R1!=zero
        NOP                   ;delayed branch slot
```

Instruction producing result	Instruction using result	Latency in clock cycles
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
Integer op	Integer op	0

FP Loop Showing Stalls

```
1    Loop:    LD      F0, 0(R1)       ;F0=vector element
2             Stall
3             ADDD    F4, F0, F2      F4,F0,F2
4             Stall
5             Stall
6             SD      0(R1), F4       ;store result
7             SUBI    R1, R1, 8       ; decrement pointer 8B (DW)
8             BNEZ    R1, Loop        ;branch R1 != zero
9             Stall                   ;delayed branch slot
```

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1

9 clocks: Rewrite code to minimize stalls?

Revised FP Loop Minimizing Stalls

1	Loop:	LD	F0, 0(R1)	
2		Stall		
3		ADDD	F4, F0, F2	
4		SUBI	R1, R1, 8	
5		BNEZ	R1, Loop	; delayed branch
6		SD	8(R1), F4	; altered when move past SUBI

Replace BNEZ stall with SD by changing address of SD

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1

9 clocks: Rewrite code to minimize stalls?

Unroll Loop Four Times (straightforward way)

```
1 Loop: LD      F0,0(R1)
2       ADDD    F4,F0,F2
3       SD      0(R1),F4        ;drop SUBI & BNEZ
4       LD      F6,-8(R1)
5       ADDD    F8,F6,F2
6       SD      -8(R1),F8       ;drop SUBI & BNEZ
7       LD      F10,-16(R1)
8       ADDD    F12,F10,F2
9       SD      -16(R1),F12     ;drop SUBI & BNEZ
10      LD      F14,-24(R1)
11      ADDD    F16,F14,F2
12      SD      -24(R1),F16
13      SUBI    R1,R1,#32       ;alter to 4*8
14      BNEZ    R1,LOOP
15      NOP
```

Rewrite loop to minimize stalls?

15 + 4 x (1+2) = 27 clock cycles

Unrolled Loop That Minimizes Stalls

```
1 Loop:  LD      F0,0(R1)
2        LD      F6,-8(R1)
3        LD      F10,-16(R1)
4        LD      F14,-24(R1)
5        ADDD    F4,F0,F2
6        ADDD    F8,F6,F2
7        ADDD    F12,F10,F2
8        ADDD    F16,F14,F2
9        SD      0(R1),F4
10       SD      -8(R1),F8
11       SD      -16(R1),F12
12       SUBI    R1,R1,#32
13       BNEZ    R1,LOOP
14       SD      8(R1),F16          ; 8-32 = -24
```

14 clock cycles,

When safe to move instructions?

- What assumptions made when moved code?
  - OK to move store past SUBI even though changes register
  - OK to move loads before stores: get right data?
  - When is it safe for compiler to do such changes?

Compiler Perspectives on Code Movement

- Definitions: compiler concerned about dependencies in program, whether or not a HW hazard depends on a given pipeline
- Try to schedule to avoid hazards
- (True) Data dependencies (RAW if a hazard for HW)
  - Instruction i produces a result used by instruction j, or
  - Instruction j is data dependent on instruction k, and instruction k is data dependent on instruction i.
- If dependent, can't execute in parallel
- Easy to determine for registers (fixed names)
- Hard for memory:
  - Does 100(R4) = 20(R6)?
  - From different loop iterations, does 20(R6) = 20(R6)?

Where are the data dependencies?

```
1 Loop:   LD     F0,0(R1)
2         ADDD   F4,F0,F2
3         SUBI   R1,R1,8
4         BNEZ   R1,Loop      ;delayed branch
5  SD     8(R1),F4            ;altered when move past SUBI
```

Compiler Perspectives on Code Movement

- Another kind of dependence called name dependence: two instructions use same name (register or memory location) but don’t exchange data
- Antidependence (WAR if a hazard for HW)
  - Instruction j writes a register or memory location that instruction i reads from and instruction i is executed first
- Output dependence (WAW if a hazard for HW)
  - Instruction i and instruction j write the same register or memory location; ordering between instructions must be preserved.

Where are the name dependencies?

```
1 Loop: LD      F0,0(R1)
2       ADDD    F4,F0,F2
3       SD      0(R1),F4  ;drop SUBI & BNEZ
4       LD      F0,-8(R1)
2       ADDD    F4,F0,F2
3       SD      -8(R1),F4 ;drop SUBI & BNEZ
7       LD      F0,-16(R1)
8       ADDD    F4,F0,F2
9       SD      -16(R1),F4 ;drop SUBI & BNEZ
10      LD      F0,-24(R1)
11      ADDD    F4,F0,F2
12      SD      -24(R1),F4
13      SUBI    R1,R1,#32  ;alter to 4*8
14      BNEZ    R1,LOOP
15      NOP
```

How can remove them?

Where are the name dependencies?

```
1 Loop: LD      F0,0(R1)
2       ADDD    F4,F0,F2
3       SD      0(R1),F4
4       LD      F0,-8(R1)
2       ADDD    F4,F0,F2
3       SD      -8(R1),F4
7       LD      F0,-16(R1)
8       ADDD    F4,F0,F2
9       SD      -16(R1),F4
10      LD      F0,-24(R1)
11      ADDD    F4,F0,F2
12      SD      -24(R1),F4
13      SUBI    R1,R1,#32
14      BNEZ    R1,LOOP
15      NOP
```

How can remove them?

```
1 Loop: LD      F0,0(R1)
2       ADDD    F4,F0,F2
3       SD      0(R1),F4
4       LD      F6,-8(R1)
5       ADDD    F8,F6,F2
6       SD      -8(R1),F8
7       LD      F10,-16(R1)
8       ADDD    F12,F10,F2
9       SD      -16(R1),F12
10      LD      F14,-24(R1)
11      ADDD    F16,F14,F2
12      SD      -24(R1),F16
13      SUBI    R1,R1,#32
14      BNEZ    R1,LOOP
15      NOP
```

Called "register renaming"

Compiler Perspectives on Code Movement

- Again Name Dependencies are Hard for Memory Accesses
  - Does 100(R4) = 20(R6)?
  - From different loop iterations, does 20(R6) = 20(R6)?
- Our example required compiler to know that if R1 doesn't change then:

$$0(R1) \neq -8(R1) \neq -16(R1) \neq -24(R1)$$

There were no dependencies between some loads and stores so they could be moved by each other

Compiler Perspectives on Code Movement

- Final kind of dependence called control dependence
- Example

```
if p1 {S1;};
if p2 {S2;};
```

S1 is control dependent on p1 and S2 is control dependent on p2 but not on p1.

Compiler Perspectives on Code Movement

- Two (obvious) constraints on control dependences:
  - An instruction that is control dependent on a branch cannot be moved before the branch so that its execution is no longer controlled by the branch.
  - An instruction that is not control dependent on a branch cannot be moved to after the branch so that its execution is controlled by the branch.
- Control dependencies relaxed to get parallelism; get same effect if preserve order of exceptions (address in register checked by branch before use) and data flow (value in register depends on branch)

Where are the control dependencies?

```
1 Loop: LD      F0,0(R1)
2       ADDD    F4,F0,F2
3       SD      0(R1),F4
4       SUBI    R1,R1,8
5       BEQZ    R1,exit
6       LD      F0,0(R1)
7       ADDD    F4,F0,F2
8       SD      0(R1),F4
9       SUBI    R1,R1,8
10      BEQZ    R1,exit
11      LD      F0,0(R1)
12      ADDD    F4,F0,F2
13      SD      0(R1),F4
14      SUBI    R1,R1,8
15      BEQZ    R1,exit
....
```

When Safe to Unroll Loop?

- Example: Where are data dependencies?
  (A,B,C distinct & nonoverlapping)

```
for (i=1; i<=100; i=i+1) {
        A[i+1] = A[i] + C[i];    /* S1 */
        B[i+1] = B[i] + A[i+1];} /* S2 */
```

  1. S2 uses the value, A[i+1], computed by S1 in the same iteration.

  2. S1 uses a value computed by S1 in an earlier iteration, since iteration i computes A[i+1] which is read in iteration i+1. The same is true of S2 for B[i] and B[i+1].
  This is a "loop-carried dependence": between iterations

- Implies that iterations are dependent, and can't be executed in parallel
- Not the case for our prior example; each iteration was distinct

HW Schemes: Instruction Parallelism

- Why in HW at run time?
  - Works when can't know real dependence at compile time
  - Compiler simpler
  - Code for one machine runs well on another
- Key idea: Allow instructions behind stall to proceed

```
DIVD    F0,F2,F4
ADDD    F10,F0,F8
SUBD    F12,F8,F14
```

  - Enables out-of-order execution → out-of-order completion
  - ID stage checked both for structural Scoreboard dates to CDC 6600 in 1963

HW Schemes: Instruction Parallelism

- Out-of-order execution divides ID stage:
  1. Issue—decode instructions, check for structural hazards
  2. Read operands—wait until no data hazards, then read operands
- Scoreboards allow instruction to execute whenever 1 & 2 hold, not waiting for prior instructions
- CDC 6600: In order issue, out of order execution, out of order commit (also called completion)

Scoreboard Implications

- Out-of-order completion ➔WAR, WAW hazards?
- Solutions for WAR
  - Queue both the operation and copies of its operands
  - Read registers only during Read Operands stage
- For WAW, must detect hazard: stall until other completes
- Need to have multiple instructions in execution phase ➔ multiple execution units or pipelined execution units
- Scoreboard keeps track of dependencies, state or operations
- Scoreboard replaces ID, EX, WB with 4 stages

Four Stages of Scoreboard Control

1. Issue—decode instructions & check for structural hazards (ID1)

   If a functional unit for the instruction is free and no other active instruction has the same destination register (WAW), the scoreboard issues the instruction to the functional unit and updates its internal data structure. If a structural or WAW hazard exists, then the instruction issue stalls, and no further instructions will issue until these hazards are cleared.

2. Read operands—wait until no data hazards, then read operands (ID2)

   A source operand is available if no earlier issued active instruction is going to write it, or if the register containing the operand is being written by a currently active functional unit. When the source operands are available, the scoreboard tells the functional unit to proceed to read the operands from the registers and begin execution. The scoreboard resolves RAW hazards dynamically in this step, and instructions may be sent into execution out of order.

Four Stages of Scoreboard Control

3. Execution—operate on operands (EX)

   The functional unit begins execution upon receiving operands. When the result is ready, it notifies the scoreboard that it has completed execution.

4. Write result—finish execution (WB)

   Once the scoreboard is aware that the functional unit has completed execution, the scoreboard checks for WAR hazards. If none, it writes results. If WAR, then it stalls the instruction.

   Example:

```
DIVD    F0,F2,F4
ADDD    F10,F0,F8
SUBD    F8,F8,F14
```

   CDC 6600 scoreboard would stall SUBD until ADDD reads operands

Three Parts of the Scoreboard

1. Instruction status—which of 4 steps the instruction is in
2. Functional unit status—Indicates the state of the functional unit (FU). 9 fields for each functional unit
   - Busy—Indicates whether the unit is busy or not
   - Op—Operation to perform in the unit (e.g., + or –)
   - Fi—Destination register
   - Fj, Fk—Source-register numbers
   - Qj, Qk—Functional units producing source registers Fj, Fk
   - Rj, Rk—Flags indicating when Fj, Fk are ready
3. Register result status—Indicates which functional unit will write each register, if one exists. Blank when no pending instructions will write that register

Detailed Scoreboard Pipeline Control

Instruction status	Wait until	Bookkeeping
Issue	Not busy (FU) and not result(D)	Busy(FU)← yes; Op(FU)← op; Fi(FU)← \`D'; Fj(FU)← \`S1'; Fk(FU)← \`S2'; Qj← Result('S1'); Qk← Result(\`S2'); Rj← not Qj; Rk← not Qk; Result('D')← FU;
Read operands	Rj and Rk	Rj← No; Rk← No
Execution complete	Functional unit done	
Write result	∀f((Fj(f)≠Fi(FU) or Rj(f)=No) & (Fk(f) ≠Fi(FU) or Rk(f)=No))	∀f(if Qj(f)=FU then Rj(f)← Yes); ∀f(if Qk(f)=FU then Rj(f)← Yes); Result(Fi(FU))← 0; Busy(FU)← No

Scoreboard Example

FP Add latency = 2 clocks, Multiply = 10, Divide = 40

Instruction status

Instruction		j	k	Issue	Read operanc	Executi complet	Write Result
LD	F6	34+	R2				
LD	F2	45+	R3				
MULTI	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
	FU									

Scoreboard Example Cycle 1

Instruction status

Instruction		j	k	Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1			
LD	F2	45+	R3				
MULTI	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
1	FU				Integer					

Scoreboard Example Cycle 2

Instruction status

Instruction	j	k		Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2		
LD	F2	45+	R3				
MULTI	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

				dest	S1	S2	FU for j	FU for k	Fj?	Fk?
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
2	FU				Integer					

- **Issue 2nd LD?**

Scoreboard Example Cycle 3

Instruction status

Instruction		j	k	Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	
LD	F2	45+	R3				
MULT	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
3	FU				Integer					

Scoreboard Example Cycle 4

Instruction status					Read	Execu	Write
Instructio	j	k		Issue	operai	compl	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3				
MUL	F0	F2	F4				
SUB	F8	F6	F2				
DIVD	F10	F0	F6				
ADD	F6	F8	F2				

Functional unit status				dest	S1	S2	FU foı	FU for	Fj?	Fk?
Tim	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status										
Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
4	FU				Integer					

Scoreboard Example Cycle 5

Instruction status

Instruction		j	k	Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5			
MULTI	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
5	FU		Integer							

Scoreboard Example Cycle 6

Instruction status

Instruction		j	k	Issue	Read operand	Executi complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6		
MULTI	F0	F2	F4	6			
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU	Mult1	Integer							

Scoreboard Example Cycle 7

Instruction status

Instruction		j	k	Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MULTI	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
	Divide	No								

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
7	FU	Mult1	Integer			Add				

Scoreboard Example Cycle 8a

Instruction status

Instruction	j	k		Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MULTI	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
8	FU	Mult1	Integer			Add	Divide			

Scoreboard Example Cycle 8b

Instruction status

Instruction	j	k		Issue	Read operand	Executi complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTI	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
8	FU	Mult1				Add	Divide			

Scoreboard Example Cycle 9

Instruction status

Instruction		j	k	Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9		
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
10	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
2	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
9	FU	Mult1				Add	Divide			

- **Read operands for MULT & SUBD? Issue ADDD?**

Scoreboard Example Cycle 11

Instruction status					Read	Executi	Write
Instruction		j	k	Issue	operand	complet	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTI	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status				dest	S1	S2	FU for j	FU for k	Fj?	Fk?
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
8	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
0	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
11	FU	Mult1				Add	Divide			

Scoreboard Example Cycle 12

Instruction status

Instruction		j	k	Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTI	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
7	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	No								
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
12	FU	Mult1					Divide			

- **Read operands for DIVD?**

Scoreboard Example Cycle 13

Instruction status

Instruction		j	k	Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTI	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13			

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
6	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
13	FU	Mult1			Add		Divide			

Scoreboard Example Cycle 14

Instruction status

Instruction	j	k		Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTI	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14		

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
5	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
2	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
14	FU	Mult1			Add		Divide			

Scoreboard Example Cycle 15

Instruction status

Instruction	j	k		Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTI	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14		

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
4	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
1	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
15	FU	Mult1			Add		Divide			

Scoreboard Example Cycle 16

Instruction status					Read	Executic	Write
Instruction		*j*	*k*	*Issue*	*operand*	*complete*	*Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTI	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status				*dest*	*S1*	*S2*	*FU for j*	*FU for k*	*Fj?*	*Fk?*
Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	No								
3	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
0	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status										
Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
16	*FU*	Mult1			Add		Divide			

Scoreboard Example Cycle 17

Instruction status

Instruction	j	k		Issue	Read operand	Executic complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTI	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
2	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
17	FU	Mult1			Add		Divide			

- **Write result of ADDD?**

Scoreboard Example Cycle 18

Instruction status

Instruction		j	k	Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTI	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
1	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
18	FU	Mult1			Add		Divide			

Scoreboard Example Cycle 19

Instruction status

Instruction	j	k		Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTI	F0	F2	F4	6	9	19	
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
0	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
19	FU	Mult1			Add		Divide			

Scoreboard Example Cycle 20

Instruction status

Instruction		j	k	Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTI	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
20	FU				Add		Divide			

Scoreboard Example Cycle 21

Instruction status

Instruction		j	k	Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21		
ADDD	F6	F8	F2	13	14	16	

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
21	FU				Add		Divide			

Scoreboard Example Cycle 22

Instruction status

Instruction		j	k	Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21		
ADDD	F6	F8	F2	13	14	16	22

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
40	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
22	FU						Divide			

Scoreboard Example Cycle 61

Instruction status

Instruction		j	k	Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTI	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	
ADDD	F6	F8	F2	13	14	16	22

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
0	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
61	FU						Divide			

Scoreboard Example Cycle 62

Instruction status

Instruction		j	k	Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	62
ADDD	F6	F8	F2	13	14	16	22

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
0	Divide	No								

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
62	FU									

Review: Scoreboard Example Cycle 62

Instruction status

Instruction		j	k	Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	62
ADDD	F6	F8	F2	13	14	16	22

In-order issue; out-of-order execute & commit

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
0	Divide	No								

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
62	FU									

CDC 6600 Scoreboard

- Limitations of 6600 scoreboard:
  - No forwarding hardware
  - Limited to instructions in basic block (small *window)*
  - Small number of functional units (structural hazards), especially integer/load store units
  - Do not issue on structural hazards
  - Wait for WAR hazards
  - Prevent WAW hazards

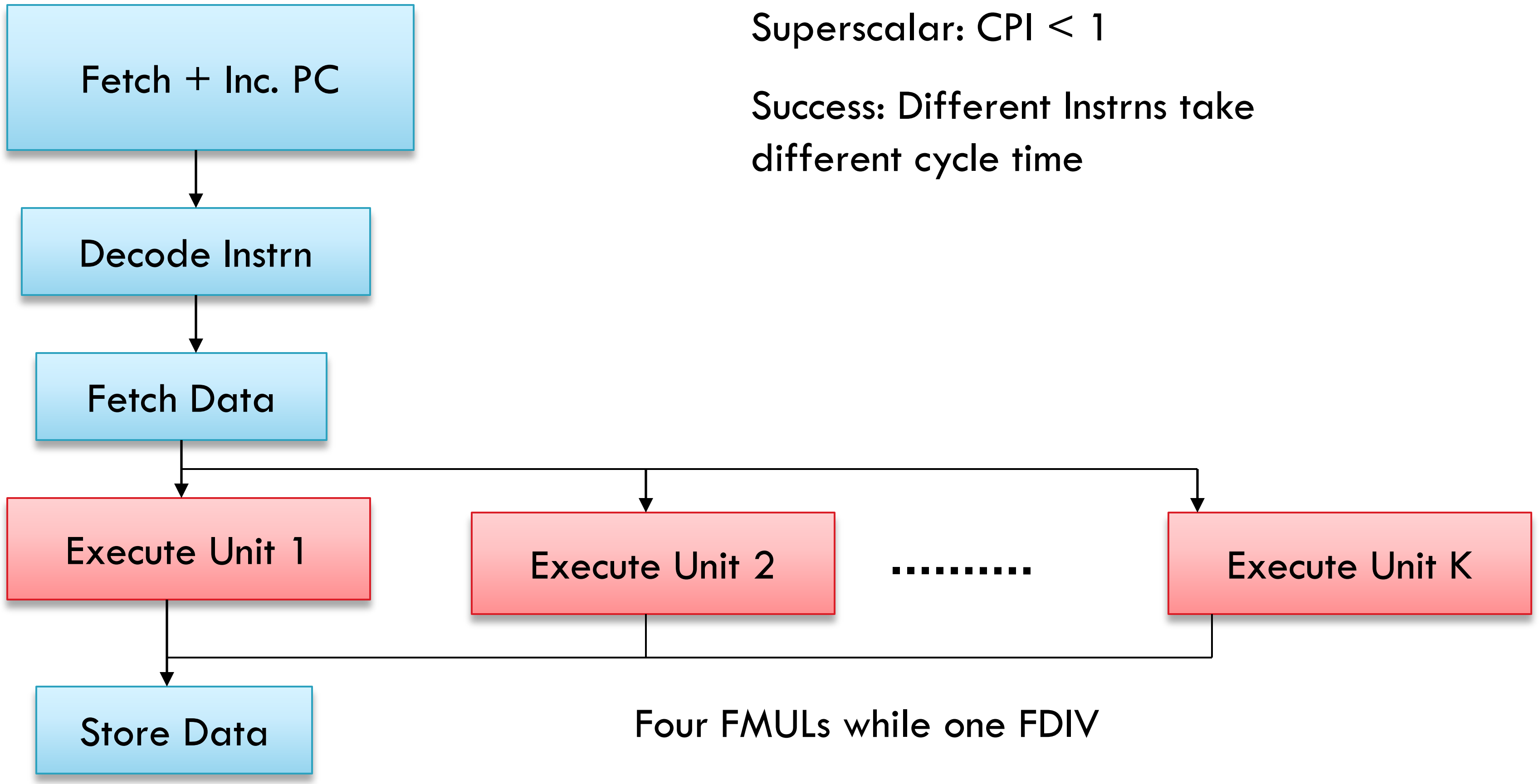
ANOTHER CASE STUDY EXAMPLE

1 May 2020

ILP Continues….

- Data Hazards
  - LOAD R1, [R2 + 10] // Loads into R1
  - ADD R3, R1, R2 //R3 = R1 + R2
- This is the "Read After Write (RAW)" Data Hazard for R1
  - LD R1, [R2+10]
  - ADD R3, R1, R12
  - LD R1, [R2 + 14]
  - ADD R12, R1, R2
- This shows the WAW for R1 and WAR for R12

ILP – Pipelining Advanced



Superscalar: CPI < 1

Success: Different Instrns take different cycle time

Four FMULs while one FDIV

Implies – Out-of-Order Execution

Difficulties in Superscalar Construction

- Ensuring no Data Hazards among several instructions executing in the different execution units at a same point of time.
- If this is done by compiler – then Static Instruction Scheduling – VLIW - Itanium
- Done by the hardware – then Dynamic Instruction Scehduling – Tomasulo – MIPS Embedded Processor

Static Instruction Scheduling

- Compiler make bundles of “K” instructions that can be put at the same time to the execution units such that there are no data dependencies between them.
  - **V**ery **L**ong **I**nstruction **W**ord (**VLIW**) to accommodate “K’ instructions at a time
- Lot of “NOPS” if the bundle cannot be filled with relevant instructions
  - Size of the executable
- Does not complicate the Hardware
- Source code portability – if I make the next gen processor with K+5 units (say) – then?
  - Solved by having a software/firmware emulator which has a negative say in the performance.

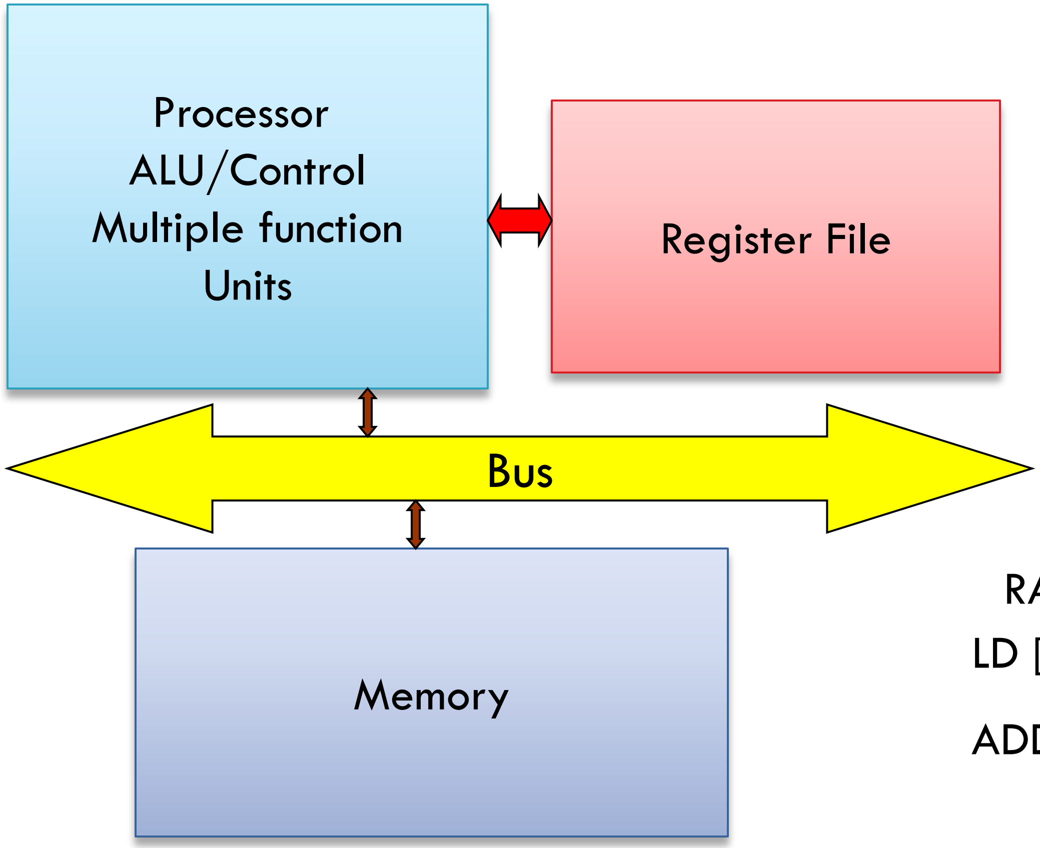
Dynamic Instruction Scheduling

- The data hazards are handled by the hardware
  - RAW using Operand Forwarding Technique
  - WAR and WAW using Register Renaming Technique

Processor Overview

Why should result of LD go to R2 in Reg file and then reload to ALU?

Forward the same on its way to reg file



RAW

LD [R1+20],R2

ADD R3,R2,R4

Register Renaming

1. ADD R1, R2, R3
2. ST R1, [R4+50]
3. ADD R1, R5, R6
4. SUB R7,R1,R8
5. ST R1, [R4 + 54]
6. ADD R1, R9, R10

Dependencies due to Reg R1

RAW: (1,2), (1,4), (1,5) (3,4) (3,5)

WAR: (2,3), (2,6), (4,6), (5,6)

WAW: (1,3), (1,6), (3,6)

Register Renaming: Static Scheduling

1. ADD R1, R2, R3
2. ST R1, [R4+50]
3. ADD R12, R5, R6
4. SUB R7,R12,R8
5. ST R12, [R4 + 54]
6. ADD R1, R9,R10

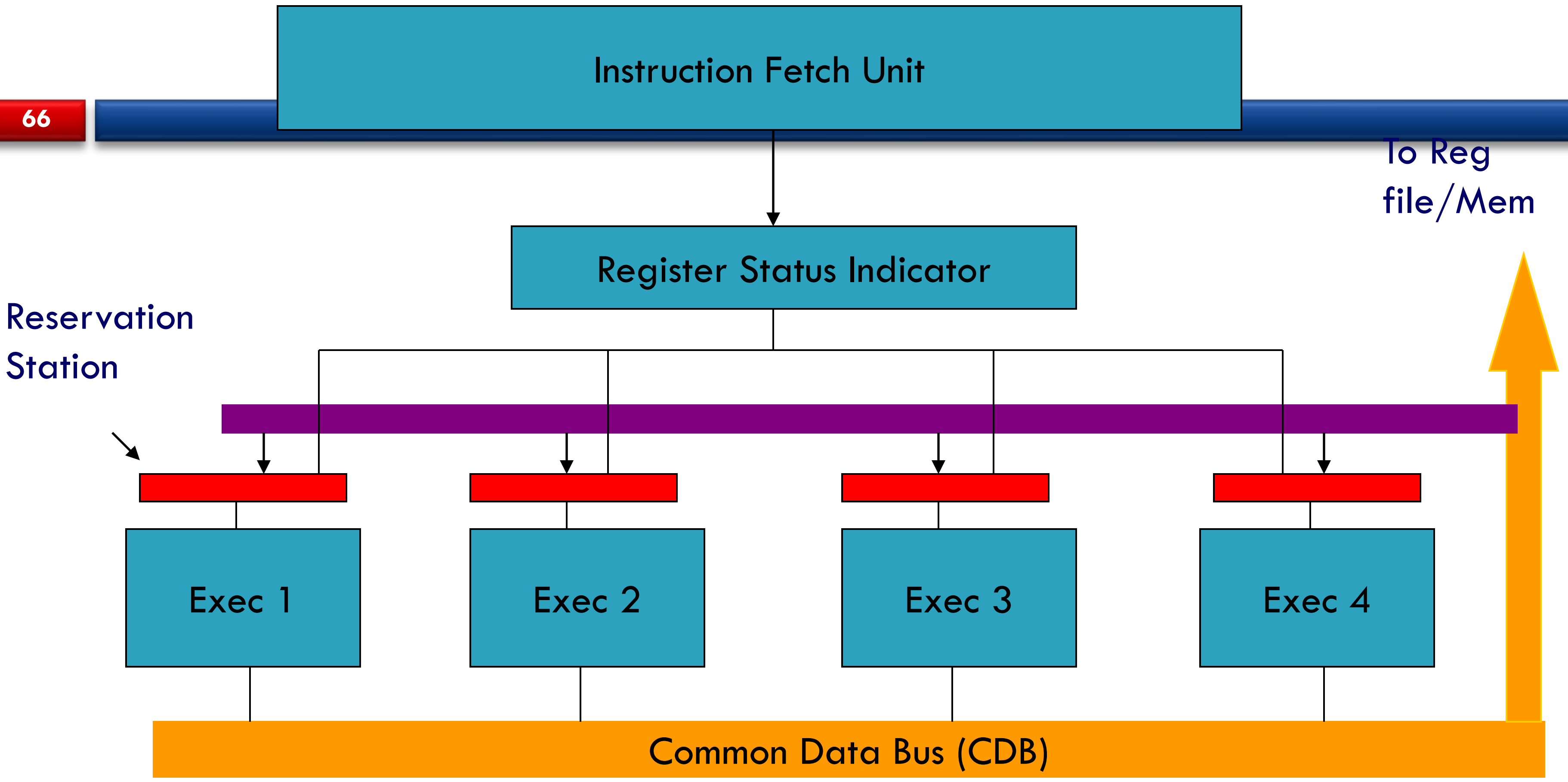
Rename R1 to R12 after Instruction 3 till Instruction 6

Dependency only within a window and not the whole program.

Only WAR and WAW are between (1,6) and (2,6) which are far away in the program order

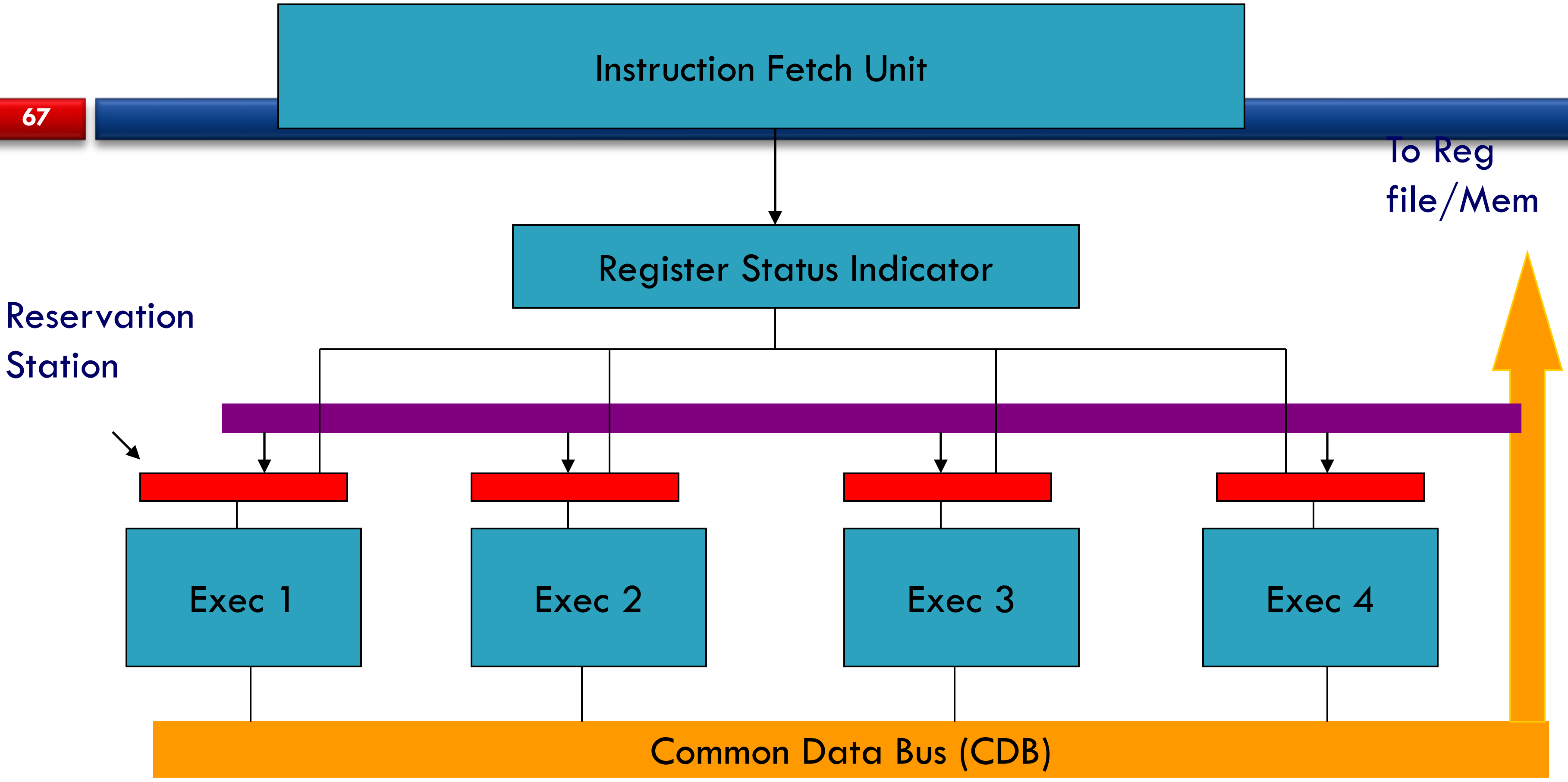
Increases Register pressure for the compiler

Dynamic Scheduling - Tomasulo



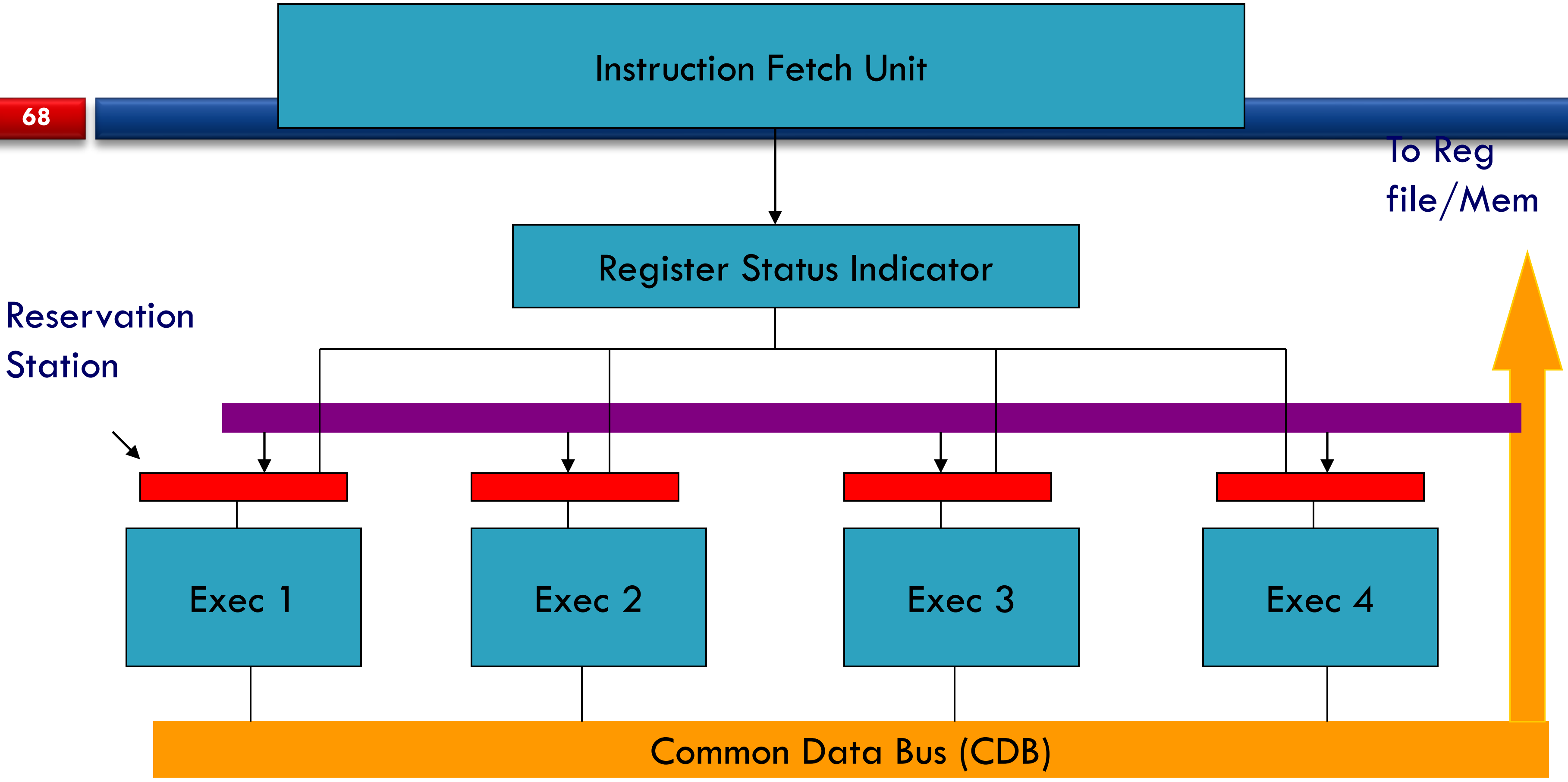
Instructions are fetched one by one and decoded to find the type of operation and the source of operands

Dynamic Scheduling - Tomasulo



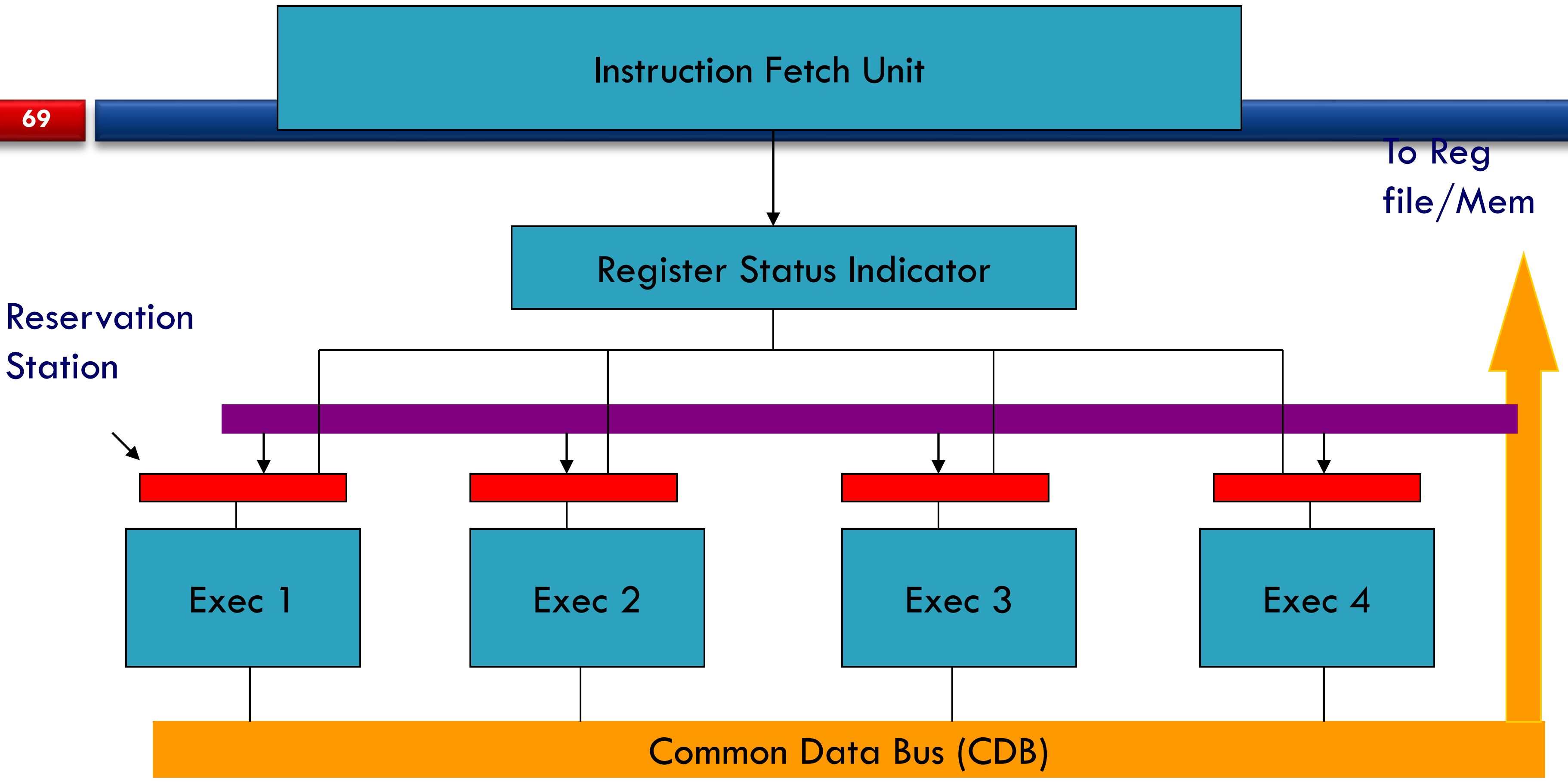
Register Status Indicator indicates whether the latest value of the register is in the reg file or currently being computed by some execution unit and if the latter it states the execution unit number

Dynamic Scheduling - Tomasulo



If all operands available then operation proceeds in the allotted execution unit, else, it waits in the reservation station of the allotted execution unit pinging the CDB

Dynamic Scheduling - Tomasulo



Every Execution unit writes the result along with the unit number on to the CDB which is forwarded to all reservation stations, Reg-file and Memory

An Example:

1. ADD R1, R2, R3
2. ST R1, [R4+50]
3. ADD R1, R5, R6
4. SUB R7,R1,R8
5. ST R1, [R4 + 54]
6. ADD R1, R9, R10

Instruction Fetch

Register Status Indicator

Reg Number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Status	0	0	0	0	0	0	0	0	0	0

Empty	Empty	Empty	Empty	Empty	Empty

An Example:

Instruction Fetch

ADD R1, R2, R3

1. --
2. ST R1, [R4+50]
3. ADD R1, R5, R6
4. SUB R7,R1,R8
5. ST R1, [R4 + 54]
6. ADD R1, R9, R10

Register Status Indicator

Reg Number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Status	1	0	0	0	0	0	0	0	0	0

Ins 1	Empty	Empty	Empty	Empty	Empty

An Example:

1. ---
2. ---
3. ADD R1, R5, R6
4. SUB R7,R1,R8
5. ST R1, [R4 + 54]
6. ADD R1, R9, R10

Instruction Fetch

ST R1, [R4+50]

Register Status Indicator

Reg Number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Status	1	0	0	0	0	0	0	0	0	0

I 1, E	I 2, W 1	Empty	Empty	Empty	Empty

An Example:

1. ---
2. ---
3. ---
4. SUB R7,R1,R8
5. ST R1, [R4 + 54]
6. ADD R1, R9, R10

Instruction Fetch

ADD R1, R5, R6

Register Status Indicator

Reg Number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Status	3	0	0	0	0	0	0	0	0	0

I 1, E	I 2, W 1	I 3, E	Empty	Empty	Empty

Note: Reservation Station stores the number of the execution unit that shall yield the latest value of a register.

An Example:

Instruction Fetch

SUB R7,R1,R8

1. ---
2. ---
3. ---
4. ---
5. ST R1, [R4 + 54]
6. ADD R1, R9, R10

Register Status Indicator

Reg Number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Status	3	0	0	0	0	0	4	0	0	0

I 1, E	I 2, W 1	I 3, E	I 4, W 3	Empty	Empty

An Example:

Instruction Fetch

ST R1, [R4 + 54]

1. ---
2. ---
3. ---
4. ---
5. ---
6. ADD R1, R9, R10

Register Status Indicator

Reg Number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Status	3	0	0	0	0	0	4	0	0	0

I 1, E	I 2, W 1	I 3, E	I 4, W 3	I 5, W 3	Empty

An Example:

1. ---
2. ---
3. ---
4. ---
5. ---
6. ---

Instruction Fetch

ADD R1, R9, R10

Register Status Indicator

Reg Number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Status	6	0	0	0	0	0	4	0	0	0

I 1, E	I 2, W 1	I 3, E	I 4, W 3	I 5, W 3	I 6, E

An Example:

1. ADD R1, R2, R3
2. ST U1, [R4+50]
3. ADD R1, R5, R6
4. SUB R7, U3, R8
5. ST U3, [R4 + 54]
6. ADD R1, R9, R10

Instruction Fetch

ADD R1, R9, R10

Register Status Indicator

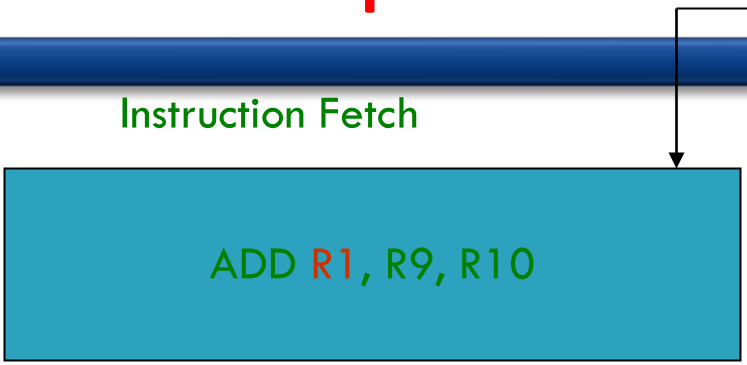
Reg Number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Status	6	0	0	0	0	0	4	0	0	0

I 1, E	I 2, W 1	I 3, E	I 4, W 3	I 5, W 3	I 6, E

Effectively three Instructions are executing and others waiting for the appropriate results. The whole program is converted as shown above.

An Example:

1. ADD R1, R2, R3
2. ST U1, [R4+50]
3. ADD R1, R5, R6
4. SUB R7, U3, R8
5. ST U3, [R4 + 54]
6. ADD R1, R9, R10



Register Status Indicator

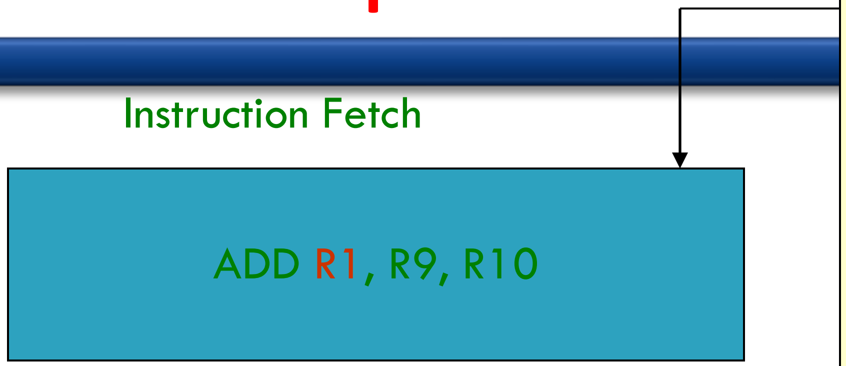
Reg Number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Status	6	0	0	0	0	0	4	0	0	0

I 1, E	I 2, W 1	I 3, E	I 4, W 3	I 5, W 3	I 6, E

See that Operand Forwarding and Register Renaming is done automatically

An Example:

1. ADD R1, R2, R3
2. ST U1, [R4+50]
3. ADD R1, R5, R6
4. SUB R7, U3, R8
5. ST U3, [R4 + 54]
6. ADD R1, R9, R10



Register Status Indicator

Reg Number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Status	6	0	0	0	0	0	4	0	0	0

I 1, E	I 2, W 1	I 3, E	I 4, W 3	I 5, W 3	I 6, E

Execution unit 6, on completion will make R1 entry in Register Status Indicator 0. Similarly unit 4 will make R7 entry 0.

Dynamic Scheduling

- Rearrange order of instructions to reduce stalls while maintaining data flow

- Advantages:
  - Compiler doesn’t need to have knowledge of microarchitecture
  - Handles cases where dependencies are unknown at compile time

- Disadvantage:
  - Substantial increase in hardware complexity
  - Complicates exceptions

Dynamic Scheduling

- Dynamic scheduling implies:
  - Out-of-order execution
  - Out-of-order completion

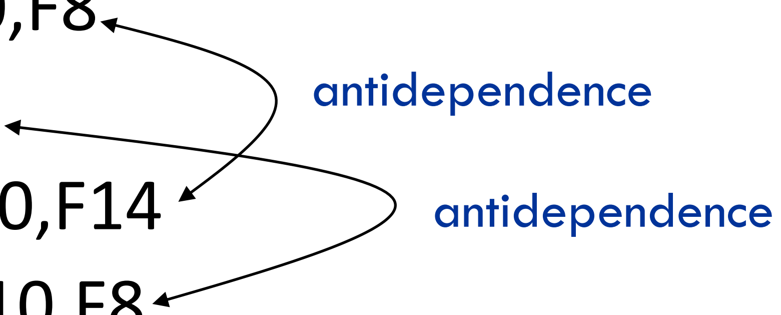
- Creates the possibility for WAR and WAW hazards

- Tomasulo's Approach
  - Tracks when operands are available
  - Introduces register renaming in hardware
    - Minimizes WAW and WAR hazards

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
```

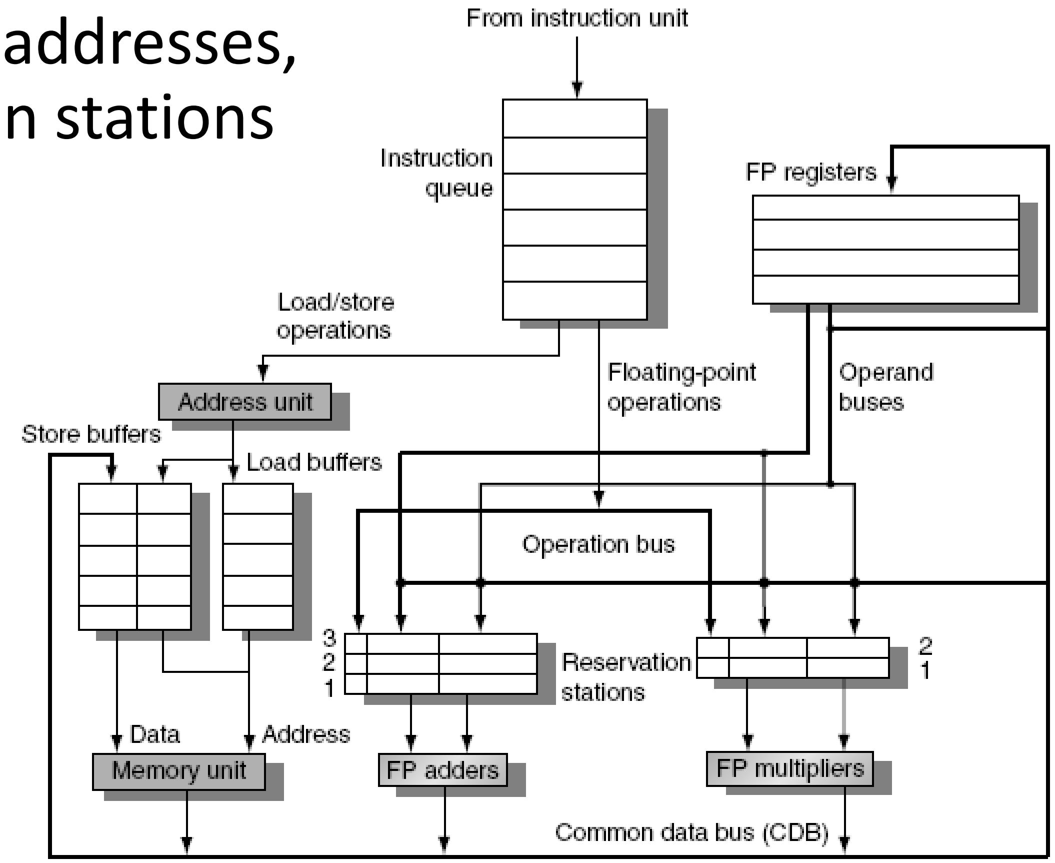
- Now only RAW hazards remain, which can be strictly ordered

Register Renaming

- Register renaming is provided by reservation stations (RS)
  - 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
  - May be more reservation stations than registers

Tomasulo's Algorithm

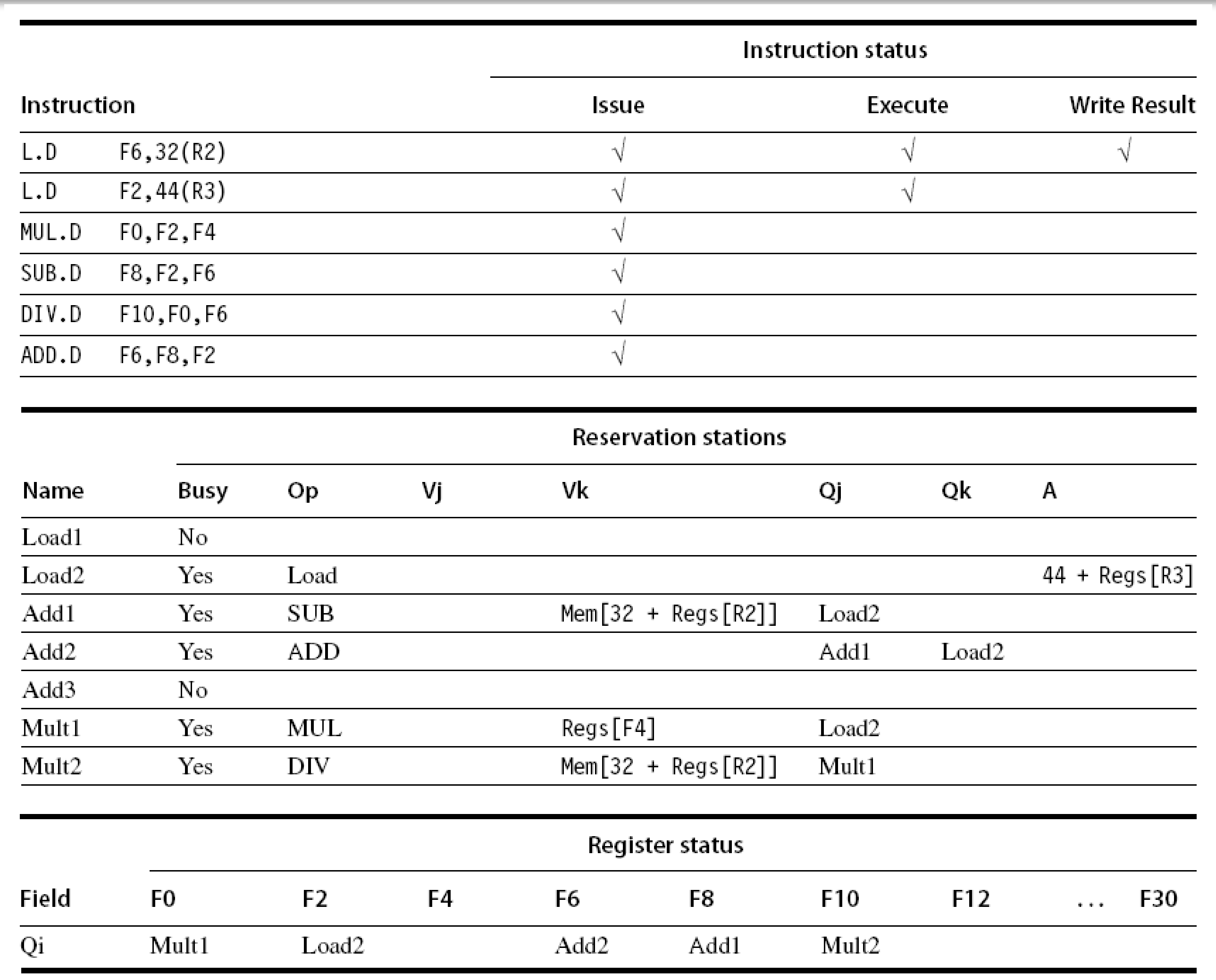
- Load and store buffers
  - Contain data and addresses, act like reservation stations
- Top-level design:



Tomasulo’s Algorithm

- Three Steps:
  - Issue
    - Get next instruction from FIFO queue
    - If available RS, issue the instruction to the RS with operand values if available
    - If operand values not available, stall the instruction
  - Execute
    - When operand becomes available, store it in any reservation stations waiting for it
    - When all operands are ready, issue the instruction
    - Loads and store maintained in program order through effective address
    - No instruction allowed to initiate execution until all branches that proceed it in program order have completed
  - Write result
    - Write result on CDB into reservation stations and store buffers
      - (Stores must wait until address and value are received)

Example



Instruction		Instruction status		
		Issue	Execute	Write Result
L.D	F6,32(R2)	√	√	√
L.D	F2,44(R3)	√	√	
MUL.D	F0,F2,F4	√		
SUB.D	F8,F2,F6	√		
DIV.D	F10,F0,F6	√		
ADD.D	F6,F8,F2	√		

	Reservation stations						
Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	Yes	Load					44 + Regs[R3]
Add1	Yes	SUB		Mem[32 + Regs[R2]]	Load2		
Add2	Yes	ADD			Add1	Load2	
Add3	No						
Mult1	Yes	MUL		Regs[F4]	Load2		
Mult2	Yes	DIV		Mem[32 + Regs[R2]]	Mult1		

	Register status								
Field	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Mult1	Load2		Add2	Add1	Mult2			

THANK YOU!!

1 May 2020