

Chapter 2

Instruction-Level Parallelism and Its Exploitation

1

Overview

- Instruction level parallelism
- Dynamic Scheduling Techniques
 - Scoreboarding
 - Tomasulo's Algorithm
- Reducing Branch Cost with Dynamic Hardware Prediction
 - Basic Branch Prediction and Branch-Prediction Buffers
 - Branch Target Buffers
- Overview of Superscalar and VLIW processors

2

CPI Equation

Pipeline CPI = Ideal pipeline CPI + Structural stalls + RAW stalls + WAR stalls + WAW stalls + Control stalls

Technique	Reduces
Loop unrolling	Control stalls
Basic pipeline scheduling	RAW stalls
Dynamic scheduling with scoreboarding	RAW stalls
Dynamic scheduling with register renaming	WAR and WAW stalls
Dynamic branch prediction	Control stalls
Issuing multiple instructions per cycle	Ideal CPI
Compiler dependence analysis	Ideal CPI and data stalls
Software pipelining and trace scheduling	Ideal CPI and data stalls
Speculation	All data and control stalls
Dynamic memory disambiguation	RAW stalls involving memory

3

Instruction Level Parallelism

- Potential overlap among instructions
 - Blocks are small (6-7 instructions)
 - Instructions are dependent
- Exploit ILP across multiple basic blocks
 - Iterations of a loop
 - for ($i = 1000; i > 0; i=i-1$)
 - $x[i] = x[i] + s;$
 - Alternative to vector instructions

4

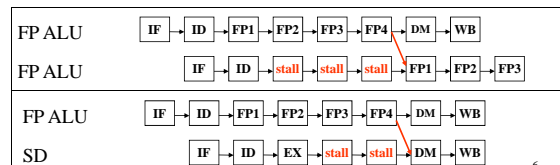
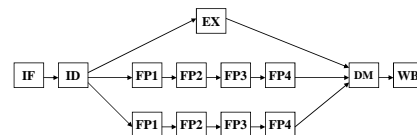
Basic Pipeline Scheduling

- Find sequences of unrelated instructions
- Compiler's ability to schedule
 - Amount of ILP available in the program
 - Latencies of the functional units
- Latency assumptions for the examples
 - Standard MIPS integer pipeline
 - No structural hazards (fully pipelined or duplicated units)
 - Latencies of FP operations:

Instruction producing result	Instruction using result	Latency
FP ALU op	FP ALU op	3
FP ALU op	SD	2
LD	FP ALU op	1
LD	SD	0

5

Sample Pipeline



6

Basic Scheduling

for (i = 1000; i > 0; i=i-1)

x[i] = x[i] + s;

Sequential MIPS Assembly Code

```
Loop: LD    F0, 0(R1)
      ADDD F4, F0, F2
      SD   0(R1), F4
      SUBI R1, R1, #8
      BNEZ R1, Loop
```

Pipelined execution:

```
Loop: LD    F0, 0(R1) 1
      stall          2
      ADDD  F4, F0, F2 3
      stall          4
      stall          5
      SD   0(R1), F4  6
      SUBI  R1, R1, #8 7
      stall          8
      BNEZ R1, Loop  9
      stall          10
```

Scheduled pipelined execution:

```
Loop: LD    F0, 0(R1) 1
      SUBI  R1, R1, #8 2
      ADDD  F4, F0, F2 3
      stall          4
      BNEZ  R1, Loop  5
      SD   8(R1), F4  6
```

7

Loop Unrolling

Unrolled loop (four copies):

```
Loop: LD    F0, 0(R1)
      ADDD  F4, F0, F2
      SD   0(R1), F4
      LD   F6, -8(R1)
      ADDD  F8, F6, F2
      SD   -8(R1), F8
      LD   F10, -16(R1)
      ADDD  F12, F10, F2
      SD   -16(R1), F12
      LD   F14, -24(R1)
      ADDD  F16, F14, F2
      SD   -24(R1), F16
      SUBI  R1, R1, #32
      BNEZ  R1, Loop
```

Scheduled Unrolled loop:

```
Loop: LD    F0, 0(R1)
      LD   F6, -8(R1)
      LD   F10, -16(R1)
      LD   F14, -24(R1)
      ADDD  F4, F0, F2
      ADDD  F8, F6, F2
      ADDD  F12, F10, F2
      ADDD  F16, F14, F2
      SD   0(R1), F4
      SD   -8(R1), F8
      SUBI  R1, R1, #32
      SD   16(R1), F12
      BNEZ  R1, Loop
      SD   8(R1), F16
```

8

Dynamic Scheduling

- Scheduling separates dependent instructions
 - Static – performed by the compiler
 - Dynamic – performed by the hardware
- Advantages of dynamic scheduling
 - Handles dependences unknown at compile time
 - Simplifies the compiler
 - Optimization is done at run time
- Disadvantages
 - Can not eliminate true data dependences

9

Out-of-order execution (1/2)

- Central idea of dynamic scheduling

– In-order execution:

DIVD F0, F2, F4	IF ID DIV
ADD F10, F0, F8	IF ID stall stall stall ...
SUBD F12, F8, F14	IF stall stall

– Out-of-order execution:

DIVD F0, F2, F4	IF ID DIV
SUBD F12, F8, F14	IF ID A1 A2 A3 A4 ...
ADD F10, F0, F8	IF ID stall

10

Out-of-Order Execution (2/2)

- Separate issue process in ID:
 - Issue
 - decode instruction
 - check structural hazards
 - in-order execution
 - Read operands
 - Wait until no data hazards
 - Read operands
- Out-of-order execution/completion
 - Exception handling problems
 - WAR hazards

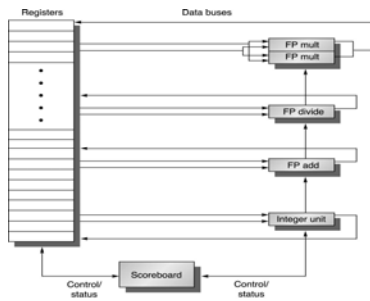
11

Dynamic Scheduling with a Scoreboard

- Details in Appendix A.7
- Allows out-of-order execution
 - Sufficient resources
 - No data dependencies
- Responsible for issue, execution and hazards
- Functional units with long delays
 - Duplicated
 - Fully pipelined
- CDC 6600 – 16 functional units

12

MIPS with Scoreboard



© 2003 Elsevier Science (USA). All rights reserved.

13

Scoreboard Operation

- Scoreboard centralizes hazard management
 - Every instruction goes through the scoreboard
 - Scoreboard determines when the instruction can read its operands and begin execution
 - Monitors changes in hardware and decides when an stalled instruction can execute
 - Controls when instructions can write results
- New pipeline

	ID	EX	WB
Issue	Read Regs	Execution	Write

14

Execution Process

- Issue
 - Functional unit is free (structural)
 - Active instructions do not have same Rd (WAW)
- Read Operands
 - Checks availability of source operands
 - Resolves RAW hazards dynamically (out-of-order execution)
- Execution
 - Functional unit begins execution when operands arrive
 - Notifies the scoreboard when it has completed execution
- Write result
 - Scoreboard checks WAR hazards
 - Stalls the completing instruction if necessary

15

Scoreboard Data Structure

- Instruction status – indicates pipeline stage
- Functional unit status
 - Busy – functional unit is busy or not
 - Op – operation to perform in the unit (+, -, etc.)
 - Fi – destination register
 - Fj, Fk – source register numbers
 - Qj, Qk – functional unit producing Fj, Fk
 - Rj, Rk – flags indicating when Fj, Fk are ready
- Register result status – FU that will write registers

16

Scoreboard Data Structure (1/3)

Instruction	Issue	Read operands	Execution completed	Write
LD F6, 34(R2)	Y	Y	Y	Y
LD F2, 45(R3)	Y	Y	Y	
MULTD F0, F2, F4	Y			
SUBD F8, F6, F2	Y			
DIVD F10, F0, F6	Y			
ADD F6, F8, F2				

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	Y	Load	F2	R3				N	Y
Mult1	Y	Mult	F0	F2	F4	Integer		N	Y
Mult2	N								
Add	Y	Sub	F8	F6	F2	Integer		Y	N
Divide	Y	Div	F10	F0	F6	Mult1		N	Y

Functional Unit	F0	F2	F4	F6	F8	F10	F12	...	F30
Mult1	Int								
Div									

17

Scoreboard Data Structure (2/3)

Instruction	Instruction status			
	Issue	Read operands	Execution complete	Write result
LD F6, 34(R2)	√	√	√	√
LD F2, 45(R3)	√	√	√	√
MULTD F0, F2, F4	√	√	√	●
SUBD F8, F6, F2	√	√	√	√
DIVD F10, F0, F6	√			
ADD F6, F8, F2	√	√	√	

Name	Functional unit status								
	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	Yes	Mult	F0	F2	F4			No	No
Mult2	No								
Add	Yes	Add	F6	F8	F2			No	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

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

Scoreboard Data Structure (3/3)

Instruction		Instruction status			
		Issue	Read operands	Execution complete	Write result
LD	F6, 34 (R2)	✓	✓	✓	✓
LD	F2, 45 (R3)	✓	✓	✓	✓
MULTD	F0, F2, F4	✓	✓	✓	✓
SUBD	F8, F6, F2	✓	✓	✓	✓
DIVD	F10, F0, F6	✓	✓	✓	●
ADD	F6, F8, F2	✓	✓	✓	✓

Functional unit status									
Name	Busy	Op	F1	FJ	Fk	Qj	Qk	Rj	Rk
Integer	No								
Multi1	No								
Multi2	No								
Add	No								
Divide	Yes	Div	F10	F0	F6			No	No

Register result status									
	F0	F2	F4	F6	F8	F10	F12	...	F30
FU									Divide

Scoreboard Algorithm

Instruction status	Wait until	Bookkeeping
Issue	Not busy (FU) and not result(D)	Busy(FU) ← yes; Op(FU) ← op; F1(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	$\forall f (Fj(f) \neq Fi(FU) \text{ or } Rj(f) = \text{No}) \text{ \& } (Fk(f) \neq Fi(FU) \text{ or } Rk(f) = \text{No})$	$\forall f (\text{if } Qj(f) = \text{FU then } Rj(f) \leftarrow \text{Yes}; \text{if } Qk(f) = \text{FU then } Rk(f) \leftarrow \text{Yes}; \text{Result}(F1(FU)) \leftarrow 0; \text{Busy}(FU) \leftarrow \text{No}$

20

Scoreboard Limitations

- Amount of available ILP
- Number of scoreboard entries
 - Limited to a basic block
 - Extended beyond a branch
- Number and types of functional units
 - Structural hazards can increase with DS
- Presence of anti- and output- dependences
 - Lead to WAR and WAW stalls

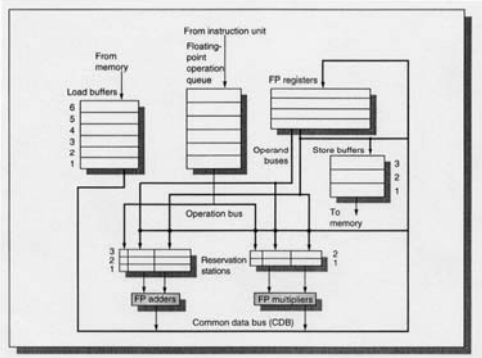
21

Tomasulo Approach

- Another approach to eliminate stalls
 - Combines scoreboard with
 - Register renaming (to avoid WAR and WAW)
- Designed for the IBM 360/91
 - High FP performance for the whole 360 family
 - Four double precision FP registers
 - Long memory access and long FP delays
- Can support overlapped execution of multiple iterations of a loop

22

Tomasulo Approach



Stages

- Issue
 - Empty reservation station or buffer
 - Send operands to the reservation station
 - Use name of reservation station for operands
- Execute
 - Execute operation if operands are available
 - Monitor CDB for availability of operands
- Write result
 - When result is available, write it to the CDB

24

Example (1/2)

Instruction status			
Instruction	Issue	Execute	Write result
LD F6, 34(R2)	√	√	√
LD F2, 45(R3)	√	√	√
MULTD F0, F2, F4	√	√	√
SUBD F8, F6, F2	√	√	√
DIVD F10, F0, F6	√	√	√
ADDD F6, F8, F2	√	√	√

Reservation stations						
Name	Busy	Op	Vj	Vk	Qj	Qk
Add1	Yes	SUB	Mem[34+Regs[R2]]			Load2
Add2	Yes	ADD			Add1	Load2
Add3	No					
Mult1	Yes	MULT		Regs[F4]		Load2
Mult2	Yes	DIV		Mem[34+Regs[R2]]		Mult1

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

25

Example (2/2)

Instruction status			
Instruction	Issue	Execute	Write result
LD F6, 34(R2)	√	√	√
LD F2, 45(R3)	√	√	√
MULTD F0, F2, F4	√	√	√
SUBD F8, F6, F2	√	√	√
DIVD F10, F0, F6	√	√	√
ADDD F6, F8, F2	√	√	√

Reservation stations						
Name	Busy	Op	Vj	Vk	Qj	Qk
Add1	No					
Add2	No					
Add3	No					
Mult1	Yes	MULT	Mem[45+Regs[R3]]	Regs[F4]		
Mult2	Yes	DIV		Mem[34+Regs[R2]]		Mult1

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

26

Tomasulo's Algorithm

Instruction status	Wait until	Action or bookkeeping
Issue	Station or buffer empty	if (Register['S1'].Qi ≠ 0) (RS[r].Qj ← Register['S1'].Qi) else (RS[r].Vj ← S1; RS[r].Qj ← 0); if (Register[S2].Qi ≠ 0) (RS[r].Qk ← Register[S2].Qi); else (RS[r].Vk ← S2; RS[r].Qk ← 0) RS[r].Busy ← yes; Register['D'].Qi ← r;
Execute	(RS[r].Qj=0) and (RS[r].Qk=0)	None—operands are in Vj and Vk
Write result	Execution completed at r and CDB available	Vx (if (Register[x].Qi=r) (Px ← result; Register[x].Qi ← 0)); Vx (if (RS[x].Qj=r) (RS[x].Vj ← result; RS[x].Qj ← 0)); Vx (if (RS[x].Qk=r) (RS[x].Vk ← result; RS[x].Qk ← 0)); Vx (if (Store[x].Qi=r) (Store[x].V ← result; Store[x].Qi ← 0)); RS[r].Busy ← No

An enhanced and detailed design in Fig. 2.12 of the textbook

27

Loop Iterations

Loop: LD F0, 0(R1)
MULTD F4, F0, F2
SD 0(R1), F4
SUBI R1, R1, #8
BNEZ R1, Loop

Instruction status				
Instruction	From iteration	Issue	Execute	Write result
LD F0, 0(R1)	1	√	√	√
MULTD F4, F0, F2	1	√	√	√
SD 0(R1), F4	1	√	√	√
LD F0, 0(R1)	2	√	√	√
MULTD F4, F0, F2	2	√	√	√
SD 0(R1), F4	2	√	√	√

Reservation stations						
Name	Busy	Fn	Vj	Vk	Qj	Qk
Add1	No					
Add2	No					
Add3	No					
Mult1	Yes	MULT		Regs[F2]		Load1
Mult2	Yes	MULT		Regs[F2]		Load2

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

Load buffers				Store buffers			
Field	Load 1	Load 2	Load 3	Field	Store 1	Store 2	Store 3
Address	Regs[R1]	Regs[R1]+8		Qi	Mult1	Mult2	
Busy	Yes	Yes	No	Busy	Yes	Yes	No
				Address	Regs[R1]	Regs[R1]+8	

28

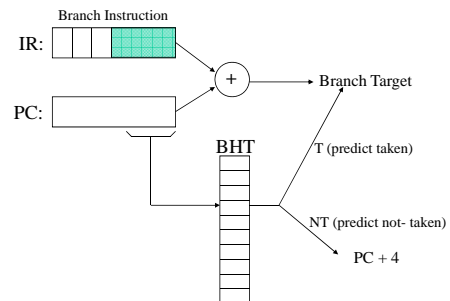
Dynamic Hardware Prediction

- Importance of control dependences
 - Branches and jumps are frequent
 - Limiting factor as ILP increases (Amdahl's law)
- Schemes to attack control dependences
 - Static
 - Basic (stall the pipeline)
 - Predict-not-taken and predict-taken
 - Delayed branch and canceling branch
 - Dynamic predictors
- Effectiveness of dynamic prediction schemes
 - Accuracy
 - Cost

29

Basic Branch Prediction Buffers

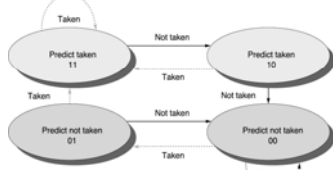
a.k.a. Branch History Table (BHT) - Small direct-mapped cache of T/NT bits



30

N-bit Branch Prediction Buffers

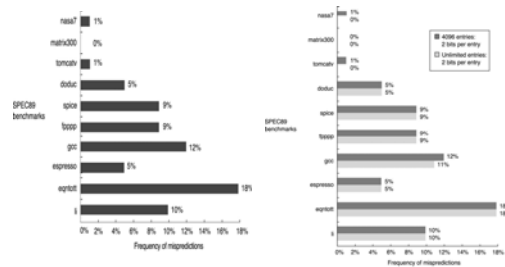
Use an n-bit saturating counter
Only the loop exit causes a misprediction
2-bit predictor almost as good as any general n-bit predictor



© 2003 Elsevier Science (USA). All rights reserved.

31

Prediction Accuracy of a 4K-entry 2-bit Prediction Buffer



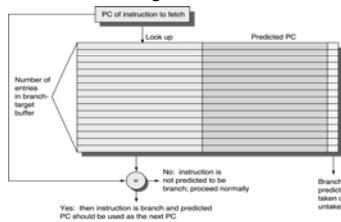
© 2003 Elsevier Science (USA). All rights reserved.

© 2003 Elsevier Science (USA). All rights reserved.

32

Branch-Target Buffers

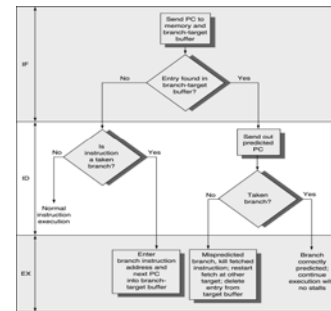
- Further reduce control stalls (hopefully to 0)
- Store the predicted address in the buffer
- Access the buffer during IF



© 2003 Elsevier Science (USA). All rights reserved.

33

Prediction with BTF



© 2003 Elsevier Science (USA). All rights reserved.

34

Performance Issues

- Limitations of branch prediction schemes
 - Prediction accuracy (80% - 95%)
 - Type of program
 - Size of buffer
 - Penalty of misprediction
- Fetch from both directions to reduce penalty
 - Memory system should:
 - Dual-ported
 - Have an interleaved cache
 - Fetch from one path and then from the other

35

Five Primary Approaches in use for Multiple-issue Processors

Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	dynamic	hardware	static	in-order execution	Sun UltraSPARC II
Superscalar (dynamic)	dynamic	hardware	dynamic	some out-of-order execution	IBM Power2
Superscalar (speculative)	dynamic	hardware	dynamic with speculation	out-of-order execution with speculation	Pentium III, MIPS R10K, Alpha 21264, HP PA 8500, IBM RS64III
VLW/LIW	static	software	static	no hazards between issue packets	Trimedia, i860
EPIC	mostly static	mostly software	mostly static	explicit dependencies marked by compiler	Itanium

Figure 3.23 The five primary approaches in use for multiple-issue processors and the primary characteristics that distinguish them. This chapter has focused on the hardware-intensive techniques, which are all some form of superscalar. The next chapter focuses on compiler-based approaches, which are either VLW or EPIC. Figure 3.61, at the end of this chapter, provides more details on a variety of recent superscalar processors.

36