


Computer Architecture
A Quantitative Approach, Sixth Edition



Chapter 3

Instruction-Level Parallelism and Its Exploitation – Dynamic Scheduling

Copyright © 2019, Elsevier Inc. All rights Reserved 1

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

Copyright © 2019, Elsevier Inc. All rights Reserved 2

Dynamic Scheduling

- Dynamic scheduling implies:
 - Out-of-order execution
 - Out-of-order completion
- Example 1:


```
fdiv.d f0,f2,f4
fadd.d f10,f0,f8
fsub.d f12,f8,f14
```

 - fsub.d is not dependent, issue before fadd.d

Copyright © 2019, Elsevier Inc. All rights Reserved 3

Dynamic Scheduling

- Example 2:


```
fdiv.d f0,f2,f4
fmul.d f6,f0,f8
fadd.d f0,f10,f14
```

 - fadd.d is not dependent, but the antidependence makes it impossible to issue earlier without register renaming

MK Copyright © 2019, Elsevier Inc. All rights Reserved 4

Register Renaming

- Example 3:


```
fdiv.d f0,f2,f4
fadd.d f6,f0,f8
fsd f6,0(x1)
fsub.d f8,f10,f14
fmul.d f6,f10,f8
```

antidependence (between fadd.d and fsd)

Output dependence (between fsd and fmul.d)

 - name dependence with f6

MK Copyright © 2019, Elsevier Inc. All rights Reserved 5

Register Renaming

- Example 3:

fdiv.d f0,f2,f4	fdiv.d f0,f2,f4
fadd.d S,f0,f8	fadd.d f6,f0,f8
fsd S,0(x1)	fsd f6,0(x1)
fsub.d T,f10,f14	fsub.d f8,f10,f14
fmul.d f6,f10,T	fmul.d f6,f10,f8

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

MK Copyright © 2019, Elsevier Inc. All rights Reserved 6

Register Renaming

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

MK Copyright © 2019, Elsevier Inc. All rights Reserved 7

Register Renaming

- 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
- Load and store buffers
 - Contain data and addresses, act like reservation stations

MK Copyright © 2019, Elsevier Inc. All rights Reserved 8

Tomasulo's Algorithm

MK Copyright © 2019, Elsevier Inc. All rights Reserved 9

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 calculation
 - 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)

MK Copyright © 2019, Elsevier Inc. All rights Reserved 10

Tomasulo's Algorithm

Op: Operation to perform in the unit (e.g., + or -)

Vj, Vk: Value of Source operands

- Store buffers has V field, result to be stored

Qj, Qk: Reservation stations producing source registers (value to be written)

- Note: Qj, Qk=0 → ready
- Store buffers only have Qj for RS producing result

A: Used to hold info for the load store (initially immediate, then effective address)

Busy: Indicates reservation station or FU is busy

Register result status—

Qi indicates which functional unit will write each register, 0 means no write to this register

MK Copyright © 2019, Elsevier Inc. All rights Reserved 11

Example

Instruction	Instruction status		
	Issue	Execute	Write result
Fld r6, 32(x2)	√	√	√
Fld r2, 44(x3)	√	√	
Fmul.d r0, r2, r4	√		
Fsub.d r8, r2, r6	√		
Fdiv.d r0, r0, r6	√		
Fadd.d r6, r0, r2	√		

Reservation stations							
Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	Yes	Load					44 + Regs[x3]
Add1	Yes	SLB	Mem[32 + Regs[x2]]		Load2		
Add2	Yes	ADD			Add1	Load2	
Add3	No						
Mult1	Yes	MUL	Regs[r4]			Load2	
Mult2	Yes	DIV	Mem[32 + Regs[x2]]		Mult1		

Register status									
Field	r0	r2	r4	r6	r8	r10	r12	...	r30
Q	Mult1	Load2		Add2	Add1	Mult2			

MK Copyright © 2019, Elsevier Inc. All rights Reserved 12

Tomasulo's Algorithm

- Example loop:


```

            Loop: fld f0,0(x1)
                fmul.d f4,f0,f2
                fsd f4,0(x1)
                addi x1,x1,8
                bne x1,x2,Loop // branches if x16 != x2
            
```

Dynamic Scheduling

MK
Copyright © 2019, Elsevier Inc. All rights Reserved
13

Tomasulo's Algorithm

Instruction status				
Instruction	From iteration	Issue	Execute	Write result
Fld f0,0(x1)	1	✓	✓	
Fmul.d f4,f0,f2	1	✓		
Fsd f4,0(x1)	1	✓		
Fld f0,0(x1)	2	✓	✓	
Fmul.d f4,f0,f2	2	✓		
Fsd f4,0(x1)	2	✓		

Reservation stations						
Name	Busy	Op	Vj	Vk	Qj	Qk
Load1	Yes	Load				Regs[x1] + 0
Load2	Yes	Load				Regs[x1] - 0
ADD1	No					
ADD2	No					
ADD3	No					
Multi	Yes	MLL		Regs[F2]	Load1	
Multi2	Yes	MLL		Regs[F2]	Load2	
Store1	Yes	Store	Regs[x1]			Multi
Store2	Yes	Store	Regs[x1] - 0			Multi2

Register status									
Field	R0	F2	F4	F6	F8	F10	F12	...	F30
Qj		Load2	Multi2						

Dynamic Scheduling

MK
Copyright © 2019, Elsevier Inc. All rights Reserved
14

Hardware-Based Speculation

- Execute instructions along predicted execution paths but only commit the results if prediction was correct
- Instruction commit: allowing an instruction to update the register file when instruction is no longer speculative
- Need an additional piece of hardware to prevent any irrevocable action until an instruction commits
 - .i.e. updating state or taking an execution

Hardware-Based Speculation

MK
Copyright © 2019, Elsevier Inc. All rights Reserved
15

Reorder Buffer

- Reorder buffer – holds the result of instruction between completion and commit
- Four fields:
 - Instruction type: branch/store/register
 - Destination field: register number
 - Value field: output value
 - Ready field: completed execution?
- Modify reservation stations:
 - Operand source is now reorder buffer instead of functional unit

MK Copyright © 2019, Elsevier Inc. All rights Reserved 16

Reorder Buffer

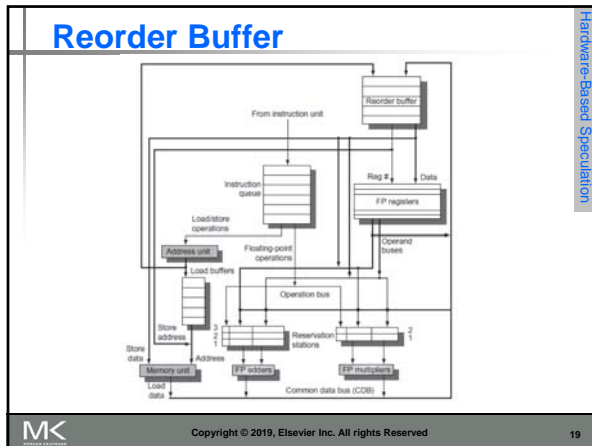
- Issue:
 - Allocate RS and ROB, read available operands
- Execute:
 - Begin execution when operand values are available
- Write result:
 - Write result and ROB tag on CDB
- Commit:
 - When ROB reaches head of ROB, update register
 - When a mispredicted branch reaches head of ROB, discard all entries

MK Copyright © 2019, Elsevier Inc. All rights Reserved 17

Reorder Buffer

- Register values and memory values are not written until an instruction commits
- On misprediction:
 - Speculated entries in ROB are cleared
- Exceptions:
 - Not recognized until it is ready to commit

MK Copyright © 2019, Elsevier Inc. All rights Reserved 18



Reorder Buffer

Reorder buffer						
Entry	Busy	Instruction	State	Destination	Value	
1	No	f1.d f6,32(x2)	Commit	f6	Mem[32 + Regs[x2]]	
2	No	f1.d f2,44(x3)	Commit	f2	Mem[44 + Regs[x3]]	
3	Yes	fma1.d f0,f2,f4	Write result	f0	#2 × Regs[f4]	
4	Yes	fsub.d f0,f2,f6	Write result	f0	#2 - #1	
5	Yes	fdiv.d f0,f0,f6	Execute	f0		
6	Yes	fadd.d f5,f0,f2	Write result	f5	#6 + #2	

Reservation stations								
Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Load1	No							
Load2	No							
Add1	No							
Add2	No							
Add3	No							
Mult1	No	fma1.d	Mem[44 + Regs[x3]]	Regs[f4]			#3	
Mult2	Yes	f01v.d		Mem[32 + Regs[x2]]	#3	#5		

FP register status										
Field	f0	f1	f2	f3	f4	f5	f6	f7	f8	f9
Reorder #	3						6	4	3	
Busy	Yes	No	No	No	No	No	Yes	...	Yes	Yes

MK
Copyright © 2019, Elsevier Inc. All rights Reserved
20

- ### Multiple Issue and Static Scheduling
- To achieve CPI < 1, need to complete multiple instructions per clock
 - Solutions:
 - Statically scheduled superscalar processors
 - VLIW (very long instruction word) processors
 - Dynamically scheduled superscalar processors
- Multiple Issue and Static Scheduling
MK
Copyright © 2019, Elsevier Inc. All rights Reserved
21

Multiple Issue

Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	Dynamic	Hardware	Static	In-order execution	Mostly in the embedded space: MIPS and ARM, including the Cortex-A53
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	Some out-of-order execution, but no speculation	None at the present
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Out-of-order execution with speculation	Intel Core i3, i5, i7; AMD Phenom; IBM Power 7
VLIW/LIW	Static	Primarily software	Static	All hazards determined and indicated by compiler (often implicitly)	Most examples are in signal processing, such as the TI C6x
EPIC	Primarily static	Primarily software	Mostly static	All hazards determined and indicated explicitly by the compiler	Itanium

MK Copyright © 2019, Elsevier Inc. All rights Reserved 22

VLIW Processors

- Package multiple operations into one instruction
- Example VLIW processor:
 - One integer instruction (or branch)
 - Two independent floating-point operations
 - Two independent memory references
- Must be enough parallelism in code to fill the available slots

MK Copyright © 2019, Elsevier Inc. All rights Reserved 23

VLIW Processors

Memory reference 1	Memory reference 2	FP operation 1	FP operation 2	Integer operation/branch
f1d f0,0(x1)	f1d f6,-8(x1)			
f1d f10,-16(x1)	f1d f14,-24(x1)			
f1d f18,-32(x1)	f1d f22,-40(x1)	fadd.d f4,f0,f2	fadd.d f8,f6,f2	
f1d f26,-48(x1)		fadd.d f12,f0,f2	fadd.d f16,f14,f2	
		fadd.d f20,f18,f2	fadd.d f24,f22,f2	
fsd f4,0(x1)	fsd f8,-8(x1)	fadd.d f28,f26,f24		
fsd f12,-16(x1)	fsd f16,-24(x1)			addi x1,x1,-56
fsd f20,24(x1)	fsd f24,16(x1)			
fsd f28,8(x1)				bne x1,x2,loop

- Disadvantages:
 - Statically finding parallelism
 - Code size
 - No hazard detection hardware
 - Binary code compatibility

MK Copyright © 2019, Elsevier Inc. All rights Reserved 24

Dynamic Scheduling, Multiple Issue, and Speculation

- Modern microarchitectures:
 - Dynamic scheduling + multiple issue + speculation
- Two approaches:
 - Assign reservation stations and update pipeline control table in half clock cycles
 - Only supports 2 instructions/clock
 - Design logic to handle any possible dependencies between the instructions
- Issue logic is the bottleneck in dynamically scheduled superscalars

MK Copyright © 2019, Elsevier Inc. All rights Reserved 25

Overview of Design

MK Copyright © 2019, Elsevier Inc. All rights Reserved 26

Multiple Issue

- Examine all the dependencies among the instructions in the bundle
- If dependencies exist in bundle, encode them in reservation stations
- Also need multiple completion/commit
- To simplify RS allocation:
 - Limit the number of instructions of a given class that can be issued in a "bundle", i.e. on FP, one integer, one load, one store


MK Copyright © 2019, Elsevier Inc. All rights Reserved 27

Example

```

Loop: ld x2,0(x1)      //x2=array element
      addi x2,x2,1     //increment x2
      sd x2,0(x1)     //store result
      addi x1,x1,8     //increment pointer
      bne x2,x3,Loop  //branch if not last
    
```


Dynamic Scheduling: Multiple Issue, and Speculation


Copyright © 2019, Elsevier Inc. All rights Reserved
28

Example (No Speculation)

Iteration number	Instructions	Issues at clock cycle number	Executes at clock cycle number	Memory access at clock cycle number	Write CDB at clock cycle number	Comment
1	ld x2,0(x1)	1	2	3	4	First issue
1	addi x2,x2,1	1	5		6	Wait for ld
1	sd x2,0(x1)	2	3	7		Wait for addi
1	addi x1,x1,8	2	3		4	Execute directly
1	bne x2,x3,Loop	3	7			Wait for addi
2	ld x2,0(x1)	4	8	9	10	Wait for bne
2	addi x2,x2,1	4	11		12	Wait for ld
2	sd x2,0(x1)	5	9	13		Wait for addi
2	addi x1,x1,8	5	8		9	Wait for bne
2	bne x2,x3,Loop	6	13			Wait for addi
3	ld x2,0(x1)	7	14	15	16	Wait for bne
3	addi x2,x2,1	7	17		18	Wait for ld
3	sd x2,0(x1)	8	15	19		Wait for addi
3	addi x1,x1,8	8	14		15	Wait for bne
3	bne x2,x3,Loop	9	19			Wait for addi


Dynamic Scheduling: Multiple Issue, and Speculation


Copyright © 2019, Elsevier Inc. All rights Reserved
29

Example (Mutiple Issue with Speculation)

Iteration number	Instructions	Issues at clock number	Executes at clock number	Read access at clock number	Write CDB at clock number	Commits at clock number	Comment
1	ld x2,0(x1)	1	2	3	4	5	First issue
1	addi x2,x2,1	1	5		6	7	Wait for ld
1	sd x2,0(x1)	2	3		7		Wait for addi
1	addi x1,x1,8	2	3		4	8	Commit in order
1	bne x2,x3,Loop	3	7			8	Wait for addi
2	ld x2,0(x1)	4	5	6	7	9	No execute delay
2	addi x2,x2,1	4	8		9	10	Wait for ld
2	sd x2,0(x1)	5	6		10		Wait for addi
2	addi x1,x1,8	5	6		7	11	Commit in order
2	bne x2,x3,Loop	6	10			11	Wait for addi
3	ld x2,0(x1)	7	8	9	10	12	Earliest possible
3	addi x2,x2,1	7	11		12	13	Wait for ld
3	sd x2,0(x1)	8	9		13		Wait for addi
3	addi x1,x1,8	8	9		10	14	Executes earlier
3	bne x2,x3,Loop	9	13			14	Wait for addi

Dynamic Scheduling: Multiple Issue, and Speculation


Copyright © 2019, Elsevier Inc. All rights Reserved
30

Branch-Target Buffer

- Need high instruction bandwidth
 - Branch-Target buffers
 - Next PC prediction buffer, indexed by current PC

MK Copyright © 2019, Elsevier Inc. All rights Reserved 31

Branch Folding

- Optimization:
 - Larger branch-target buffer
 - Add target instruction into buffer to deal with longer decoding time required by larger buffer
 - "Branch folding"

MK Copyright © 2019, Elsevier Inc. All rights Reserved 32

Fallacies and Pitfalls

- It is easy to predict the performance/energy efficiency of two different versions of the same ISA if we hold the technology constant

MK Copyright © 2019, Elsevier Inc. All rights Reserved 33

Fallacies and Pitfalls

- Processors with lower CPIs / faster clock rates will also be faster

Processor	Implementation technology	Clock rate	Power	SPECint2006 base	SPECfp2006 baseline
Intel Pentium 4 670	90 nm	3.8 GHz	115 W	11.5	12.2
Intel Itanium 2	90 nm	1.66 GHz	104 W approx. 70 W core core	14.5	17.3
Intel i7 920	45 nm	3.3 GHz	130 W total approx. 80 W core core	35.5	38.4

- Pentium 4 had higher clock, lower CPI
- Itanium had same CPI, lower clock

MK Copyright © 2019, Elsevier Inc. All rights Reserved 34

Fallacies and Pitfalls

- Sometimes bigger and dumber is better
 - Pentium 4 and Itanium were advanced designs, but could not achieve their peak instruction throughput because of relatively small caches as compared to i7
- And sometimes smarter is better than bigger and dumber
 - TAGE branch predictor outperforms gshare with less stored predictions

MK Copyright © 2019, Elsevier Inc. All rights Reserved 35

Fallacies and Pitfalls

- Believing that there are large amounts of ILP available, if only we had the right techniques

Benchmark	Window size 4	Window size 8	Window size 16	Window size 32
gcc	10	12	13	14
espresso	13	15	16	17
i	11	12	13	14
gccp	14	22	30	42
dotloc	13	17	18	19
tomcat	14	22	34	56

MK Copyright © 2019, Elsevier Inc. All rights Reserved 36
