Computer Architectures Chapter 4

Tien-Fu Chen

National Chung Cheng Univ.

© by Tien-Fu Chen@CCU

Advance Pipelining

□ Static Scheduling

Have compiler to minimize the effect of structural, data, and control dependence

- advantages: simple hardware
- Examples:

Loop unrolling Software Pipelining Trace Scheduling

Dynamic Scheduling

Have hardware to rearrange instruction execution to reduce the stalls

- advantages:
 - > handle dependence unknown at compile time
 - simplify compiler design
 - code compatible
- Examples:

Scoreboarding

Tomasulo's Algorithm

© by Tien-Fu C Dynearcic Branch Prediction

Instruction Level Parallelism

□ Taking advantages of ILP

- Superscalar multiple issue per cycle
- Superpipelining deeper pipelines
- VLIW very long instruction word

□ Increase ILP by compiler

• Loop unrolling -

Increase instructions between loop branch by replicating loop body multiple times

- Software Pipelining Reorganize loop code such that each iteration contains code chosen from different iterations
- Trace Scheduling
 Increase parallelism by selecting more code candidates

□ Usually hardware techniques require compiler support

- Superscalar needs instruction scheduling
- VLIW needs trace scheduling

© by Tien-Fu Chen@CCU

chap4-2

Static Scheduling - Loop Unrolling

Basic Idea					
loop:	LD ADD SUB BNE	F0, 0(R1) F4, F0, F2 R1, R1, #8 R1, loop	loop:	LD ADD SD	F0, 0(R1) F4, F0, F2 0(R1), F4
Features	SD 0(R ²				F6, 8(R1)
 reduce loop branch 	overhe	ead		ADD SD	8(R1), F8
 counter increase co increase reg 	de size gister re	equirement		LD ADD SD	F10, 16(R1) F12, F10, F2 16(R1), F12
Unrol	led by 4			LD ADD SD	F14, 24(R1) F16, F14, F2 24(R1), F16
				SUB BNE	R1, R1, #32 R1, loop

Identify Dependencies by Compiler

Goal:

compiler concerned about dependencies in program, whether or not a HW hazard depends on a given pipeline

- Data dependency (True)
- Name dependency (anti-/output dependency)
- Control dependency

□ (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.
- Easy to determine for registers (fixed names)
- Hard for memory:
 - > Does 100(R4) = 20(R6)?
 - > From different loop iterations, does 20(R6) = 20(R6)?

© by Tien-Fu Chen@CCU

Name dependency

□ Name dependence:

two instructions use same name 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.

• no value being transmitted between instructions

Register Renaming

- Statically by compiler
- Dynamic by hardware

Control dependency

□ 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.

□ Two 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 exception behavior
- preserve data flow

© by Tien-Fu Chen@CCU

chap4-6

Loop-carried dependence

□ Where are data dependencies?

for (i=1; i<=100; i=i+1)

}

- 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].

□ loop-carried dependence

dependency exists between different iterations of the loop

• Loop with loop-carried dependence cannot be executed in parallel

```
□ GCD test :

write A[a * i+b]

read A[c * i+d]

if loop-carried depend exists then

(d - b) mod GCD(a,c) = 0
```

Now Safe to Unroll Loop? (p. 240)

A[1] = A[1] + B[1];

NEW:

B[101] = C[100] + D[100];

© by Tien-Fu Chen@CCU

Software Pipelining

□ Software pipelining

Reorganize loops such that each iteration contains instruction sequences from different iterations in original code

• need start-up & finish blocks



Compared with loop unrolling

vs. once per each unrolled iteration in loop unrolling



Trace Scheduling

□ Trace

a most likely sequence of instructions

□ Two phases

• trace selection

- identify frequent codes
- may use loop unrolling to generate long trace
- should consider data dependence constraints and branch points
- trace compaction
 - squeeze trace into a small of wide instructions
 - move instruction as early as possible
 - compact instructions as few as possible
 - add compensation code

□ good or bad?

- scheduling across basic blocks
- code explosion

© by Tien-Fu Chen@CCU



Summary of Static Scheduling

□ Loop unrolling

- + multiple loop body for scheduling
- + reduce branch frequency
- expand code size
- must handle "residual" iterations
- increase register pressure

□ Software Pipelining

- + no dependences in loop body
- does not reduce branch hazards
- need start-up and finish blocks
- increase register pressure

Trace Scheduling

- + works for loops
- + increase most likely operations for VLIW
- more complex than loop unrolling
- code expansion

© by Tien-Fu Chen@CCU

chap4-12

Advance Pipelining again

□ Static Scheduling

Have compiler to minimize the effect of structural, data, and control dependence

Dynamic Scheduling

HW rearrange the instruction execution to reduce stalls

Key idea

Allow instructions behind stall to proceed DIVD F0,F2,F4 ADDD F10,F0,F8 SUBD F8,F8,F14

- Enables out-of-order execution => out-of-order completion
- ID stage checked both for structural & data dependencies
- Splitting ID into two stages
 - > Issue decode instructions, check for structural hazards
 - > Read operands wait until no data hazards, then read operands

Advantages of Dynamic Scheduling

- Handles cases when dependences unknown at compile time (e.g., because they may involve a memory reference)
- □ It simplifies the compiler
- □ Allows code that compiled for one pipeline to run efficiently on a different pipeline
- Hardware speculation, a technique with significant performance advantages, that builds on dynamic scheduling

© by Tien-Fu Chen@CCU

Out-of-order execution

- □ Split pipelining stages:
 - Original DLX pipelining
 - New DLX pipelining

Issue - decode, check structural hazards read operands - wait until no data hazards execute write back-

□ Scoreboarding

- multiple function units
- each instruction goes through the scoreboard
- centralized control scheme
 - control all instruction issue
 - detect all hazards
 - > maintain hazard resolutions
- implemented in CD 6600

© by Tien-Fu Chen@CCU

IF -

ID - decode, check for hazards, fetch operands

EX - execute instruction WB -

Scoreboarding

Dealing with hazards

- Issue: check structural hazards check for WAW hazards stall issue until hazard cleared
- Read operand: check for RAW hazards wait until data ready
- Execution: execute operations notify scoreboard when completion
- Write back: check for WAR
 stall write until clear

□ Maintain three data structures

- Instruction status indicate the four steps of instruction in
- Function unit status indicate states of each function units

• Register result status -

indicate which function unit will write a register © by Tien-Fu Chen@CCU

chap4-16

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

© by Tien-Fu Chen@CCU

Detailed control and data maintaining

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

□ Try to identify how scoreboard deals with

- RAW
- WAR
 - > Queue both the operation and copies of its operands
 - > Read registers only during Read Operands stage
- WAW

© by Tien-Fu Chen@CCU

chap4-18

9

Instruct	ion	Issue	Read operands	Execution complete	Write result
LD	F6,34(R2)	4	V	1	1
LD	F2,45(R3)	V	V	1	
MULTD	F0,F2,F4	V			
SUBD	F8,F6,F2	1			
DIVD	F10,F0,F6	4			
ADDD	F6,F8,F2				

	Functional unit status									
Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk	
Integer	Yes	Load	F2	R3				No		
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									
	F0	F2	F4	F6	F8	F10	F12		F30		
FU	Mult1	Integer			Sub	Divide					

© b

Scoreboarding (cont)

Benefits

- Improvement of 1.7 for Fortran programs
- 2.5 for hand-coded assembly program
- How about modern compiler??

- only as much as one function unit
- main cost in buses (4 times)

Limitations

- Results are through register file, never data forwarded.
- Data dependency must wait for access to register file
- Stalls when WAW hazards occur
- Limited to instructions in basic block (small window)

© by Tien-Fu Chen@CCU

Tomasulo's Algorithm

□ Key features:

- Reservation stations at each functional unit
- A Common Data Bus (CDB) to broadcast all results
- Employ register renaming
- Use "tag" to handle hazard control
- load/store buffers

□ Only three steps (not include IF, MEM)

- Issue -get instruction
 - check available function unit
 - or check available load buffer
 - stalls on structural hazards
- Execute execute when operands available if not, check CDB for operand
- Write back if CDB available, write result if not, stalls on CDB

Data structures are attached to reservation stations, reg file, load buffer

Data structure of Tomasulo's Algorithm



Reservation Station Components

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: No ready flags as in Scoreboard; Qj,Qk=0 => ready
- Store buffers only have Qi for RS producing result

Busy—Indicates reservation station or FU is busy

Register result status—Indicates which functional unit will write each register, if one exists. Blank when no pending instructions that will write that register.

Bookkeeping rules

Instruction status	Wait until	Action or bookkeeping
Issue	Station or buffer empty	<pre>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;</pre>
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	<pre>∀x(if (Register[x].Qi=r) {Fx← result; Register[x].Qi← 0}); ∀x(if (RS[x].Qj=r) {RS[x].Vj← result; RS[x].Qj ← 0}); ∀x(if (RS[x].Qk=r) {RS[x].Vk← result; RS[x].Qk ← 0}); ∀x(if (Store[x].Qi=r) {Store[x].V← result; Store[x].Qi ← 0}); RS[r].Busy← No</pre>

□ How to eliminating WAW and WAR?

© by Tien-Fu Chen@CCU

chap4-24

			Instruction status	
Instruct	ion	Issue	Execute	Write result
LD	F6,34(R2)	4	V	V
LD	F2,45(R3)	1	V	
MULTD	F0,F2,F4	1		
SUBD	F8,F6,F2	1		
DIVD	F10,F0,F6	4		
ADDD	F6,F8,F2	1		

	Reservation stations									
Name	Busy	Ор	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

C١

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

		Reservation stations									
Name	Busy	Ор	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]
© by	Qi	Mult1					Mult2] ·26

With load/store buffers

		Instruction status							
Instruction		From iteration	Issue	1	Execute	Write result			
LD	F0,0(R1)	1	4		4				
MULTD	F4,F0,F2	1	4						
SD	0(R1),F4	1	4						
LD	F0,0(R1)	2	4		4				
MULTD	F4,F0,F2	2	4						
SD	0(R1),F4	2	4						
			Reservati	ion stations					
Name	Busy	Fm	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	Lcad2		Mult2						

	Load buffers							
Field	Load 1	Load 2	Load 3					
Address	Regs[R1]	Regs[R1]-8						
Busy	Yes	Yes	No					

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

© by Tien-Fu Chen@CC

Tomasulo's Algorithm (cont)

□ Differences from scoreboarding

- Control & buffers distributed with Function Units,
- HW renaming of registers to avoid WAW and WAR hazards
- CDB to broadcast results rather than waiting on registers
- load/store buffers are treated as function unit

Reservation Stations

- Handle distributed hazard detection and instruction control
- Use 4-bit tag field to specify which RV station or load buffer will produce result

Register renaming

- Tag assigned on instruction issue
- Tag discarded after write back to register
- WAW and WAR are eliminated
- Even better if a branch-taken strategy on a loop
 multiple execution simultaneously loop is unrolled dynamically by hardware

© by Tien-Fu Chen@CCU

Comparisons

Scoreboaring

- Global data structure centralized control
- Use register designator + simple, low cost
- Structural stalls on FU
- Solve : RAW by register WAR in write WAW stalls on issue
- Limited:
 register serialized
 input output
- Stalls issue on block S1 <- S2 * S3 S4 <- S5 * S6

Tomasulo's Alg

- Distributed control
- Use tagged register
 - + register renaming
 - tag allocation and deallocation
 - associative compare
- Structural stalls on RS
- Solve: RAW by CDB WAR copy opnd to RS WAW by renaming
- Limited: CDB
 + broadcast result
 - one result per cycle

Dynamic Branch Prediction



where change prediction only if get misprediction twice.



© by Tien-Fu Chen@CCU

Performance of Prediction Buffer

□ Mispredict because either:

- Wrong guess for that branch
- Got history of wrong branch when index table

□ Prediction Accuracy

- 4096 entry table programs vary from 1% misprediction (nasa7, tomcatv) to 18% (eqntott), with spice at 9% and gcc at 12%
- 4096 about as good as infinite table, but 4096 is a lot of HW

□ Performance of prediction buffer:

Given that 90% hit in buffer and 90% of correct guess

 $Accuracy = (\% hit in buffer * \% correct) + [(1 - \% hit in buffer) * luck guess] = (90\% \times 90\%) + (50\% * 10\%) = 86\%$

© by Tien-Fu Chen@CCU

Correlating Branch Predictor

□ Idea: Branch address (4 bits) taken/not taken of recently executed branches is related to behavior of next branch 2-bits per branch (as well as the history of that local predictors branch behavior) Prediction □ a (m,n) predictor use the behavior of the last m branches choose from 2^m branch predictors each is n-bit predictor for a single branch cost: 2^m x n x # of prediction entries 2-bit global selected by addr branch history (01 = not taken then taken)

© by Tien-Fu Chen@CCU

chap4-32

Branch Target Buffer **Branch PC** Predicted PC **Guess both branch** PC of instruction cond and branch target FETCH IF: check BTB ID: check cond EX: refill BTB Yes: instruction Exťra Penalty is branch and Prediction hit in predactual Penalty use predicted state buffer iction branch PC as next PC Yes Taken Taken 0 bits N Taken Yes Taken 2 No: branch not No Taken 2 predicted, proceed normally (Next PC = PC+4)□ Total penalty (60% taken branch) *Penalty* = % *hit in buffer* *% *incorrect* *2 + (1 - % hit in buffer)*% taken*2 = (90%*10%*2) + (10%*60%*2)= 0.30 cycles chap4-33

© by Tien-Fu Chen@CCU

Variations of BTB

□ Separating target buffer from a prediction buffer - PowerPC 620

- target address prediction
- branch direction

□ Storing target instructions

- allow longer time to access BTB, better for a larger BTB
- allowed to perform branch folding

Predicting indirect jumps

destination address varies at run-time

- e.g., indirect procedure calls, procedure returns,
- return address stack

□ Reducing misprediction penalty

- fetching from both predicted and unpredicted direction ->requiring dualported instruction memory
- caching instructions from multiple paths

© by Tien-Fu Chen@CCU

chap4-34

Multiple-Issue Processors

□ Superscalar

- Issue varying # of instructions per clock
- be either statically scheduled by compiler or dynamically scheduled using hardware
- e.g. IBM PowerPC, Sun SuperSparc, DEC Alpha, HP 7100

□ Very Long Instruction Words (VLIW)

- fixed number of instructions scheduled by the compiler
- · inherently statically scheduled by compiler
- Joint HP/Intel agreement in 1998??

□ Hardware support for more ILP

- Conditional/Predicated Instructions
- Speculative instructions with renaming
- Speculative execution

<u>Superscalar</u>

□ Superscalar machines

- Issue multiple independent instructions per clock cycle
- if dependence exist, only first instruction issued
- hardware not dynamically concern about issue
- need *n* function units for degree of *n*
- would help if compiler properly schedule codes

Limitation

- contention on FP registers why: FP load/store vs FP operations solution: dual-ported register file?
- Delayed load
 - why: original one delay slot, now 3 slots
- Delayed branch

□ Advantages:

• code compatibility

chap4-36

Dynamic Scheduling in Superscalar

Dependencies stop instruction issue

- Code compiler for scalar version will run poorly on superscalar
- May want code to vary depending on how superscalar

□ Simple approach:

- separate Tomasulo Control for separate reservation stations for Integer FU/Reg and for FP FU/Reg
- Issue 2X Clock Rate, so that issue remains in order
- Only FP loads might cause dependency between integer and FP issue:
 - Replace load reservation station with a load queue; operands must be read in the order they are fetched
 - > Load checks addresses in Store Queue to avoid RAW violation
 - Store checks addresses in Load Queue to avoid WAR, WAW

VLIW (Very Long Instruction Word)

□ tradeoff instruction space for simple decoding

- long instruction word has room for many operations
- all the operations the compiler puts in the long instruction word can execute in parallel

E.g., 2 int ops, 2 FP ops, 2 mems, 1 branch

16 to 24 bits per field => 7*16 = 112 bits to 7*24 = 168 bits wide

□ Features :

- Central controller issues a single long instruction
- Each instruction initiates many independent operations
- Each operation executed in fixed cycles
- Operations can be pipelined

□ Compiler Aid

- loop unrolling
- scheduling code across basic blocks
 - trace scheduling

© by Tien-Fu Chen@CCU

chap4-38

Limitations on multiple-issue

Limited ILP

- 1 branch in 5 instructions => how to keep a 5-way VLIW busy?
- Latencies of units => many operations must be scheduled
- Need Pipeline Depth x No. Functional Units of independent operations to keep machines busy

□ Limits on VLIW

- Limited amount of parallelism available in instruction sequences
- require a large-number of memory and register bandwidth for different functional units at the same time
- code size explosion loop unrolling + no op code
- VLIW lock step => 1 hazard & all instructions stall
- · binary compatibility is practical weakness

<u>Comparisons</u> Superscalar

adv.:

- better code density
- code compatible

difference :

• issue decision at: Run Time

disadv :

- require more instruction fetching & decoding
- more delayed slot to be filled
- different functional units
- muti-ported reg. file
- limited parallelism

© by Tien-Fu Chen@CCU

adv.:

- fixed instruction format
- more parallelism provided
- by trace scheduling at : Compile Time

disadv :

- static scheduling compiler
- no dynamic decision
- code explosion
- multi-ported register file
- limited parallelism
- difficult to use data cache because of sharing

chap4-40

Hardware Support for more ILP

□ Conditional (predicate) instructions

- Eliminate branches by turning branches into conditionally executed instructions
 - if (x) then A=B op C else NOP
- If false, then neither store result or cause exception
- Expanded ISA of Alpha, MIPS, PowerPC, SPARC have conditional move; PA-RISC can nullify any following instructions

Limitation of Conditional Branches

- Still takes a clock even if annulled
- Stall if condition evaluated late
- Limited use when control flow involves more than a simple sequence; Complex conditions reduce effectiveness, condition becomes known late in pipeline
- May increase CPI for conditional instructions or cause a slower overall clock rate

VLIW

Speculation

□ Speculation:

Allow an instruction to issue that is dependent on branch predicted to be taken without any consequences (including exceptions) if branch is not actually taken (without undo)

Methods support ambitious speculation

□ HW/SW cooperation

- HW and OS handle exceptions and return an undefined value for any exception
- may not be acceptable

Poison bits

- a poison bit added to each register and instruction
- a fault occurs on using a value from poison reg

□ Speculation instruction with renaming

 boosting instruction by flagging as speculation and providing renaming and buffering © by Tien-Fu Chen@CCU

Hardware-based Speculation

Basic idea

- Allow instruction executed out of order but force them to <u>commit</u> in order; prevent irrevocable action until commits
- separate speculative bypassing of results from real bypassing of results

□ Introducing Reorder Buffer

- buffer results of uncommitted instructions
- pass results among speculated instructions
- 3 fields: instruction. type, destination, and value
- key differences from Tomasulo's Algorithm: Tomasulo: read result from register once result is written reorder buffer: supply operands between completion and commit time
- replace load/store buffer
- Use reorder buffer number instead of reservation station buffer

Speculative Tomasulo's Algorithm

□ Four steps of Tomasulo's algorithm

Issue

If reservation station or reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination.

• Execution

When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute

• Write result

Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.

• Commit

When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer.



© by Tien-Fu Chen@CCU

Speculative DLX using Tomasulo's Algorithm



<u>Register renaming, virtual registers versus</u> <u>Reorder Buffers</u>

- □ Alternative to Reorder Buffer is a larger virtual set of registers and register renaming
- Virtual registers hold both architecturally visible registers + temporary values
 - replace functions of reorder buffer and reservation station
- Renaming process maps names of architectural registers to registers in virtual register set
 - Changing subset of virtual registers contains architecturally visible registers
- Simplifies instruction commit: mark register as no longer speculative, free register with old value

□ Adds 40-80 extra registers: Alpha, Pentium,...

• Size limits no. instructions in execution (used until commit)

© by Tien-Fu Chen@CCU

chap4-46

Limitation of ILP

□ Hardware model of perfect processors

- Register renaming
 infinite virtual registers and all WAW & WAR hazards are avoided
- Branch prediction perfect; no mispredictions
- Jump prediction: jumps perfectly predicted => machine with perfect speculation & an unbounded buffer of instructions available
- Memory-address alias analysis addresses are known & a store can be moved before a load provided addresses not equal
- 1 cycle latency for all instructions
- □ measurement : IPC
 - instruction issues per cycle

Observation

• FP programs are more sensitive to limit window size

Workstation Microprocessors 3/2001

Processor	Alpha 21264B	AMD Athlon	HP PA-8600	IBM Power3-II	Intel Pentium III	Intel Pentium 4	MIPS R12000	Sun Ultra-II	Sun Ultra-III
Clock Rate	833MHz	1.2GHz	552MHz	450MHz	1.0GHz	1.5GHz	400MHz	480MHz	900MHz
Cache (I/D/L2)	64K/64K	64K/64K/256K	512K/1M	32K/64K	16K/16K/256K	12K/8K/256K	32K/32K	16K/16K	32K/64K
Issue Rate	4 issue	3 x86 instr	4 issue	4 issue	3 x86 instr	3 x ROPs	4 issue	4 issue	4 issue
Pipeline Stages	7/9 stages	9/11 stages	7/9 stages	7/8 stages	12/14 stages	22/24 stages	6 stages	6/9 stages	14/15 stages
Out of Order	80 instr	72ROPs	56 instr	32 instr	40 ROPs	126 ROPs	48 instr	None	None
Rename regs	48/41	36/36	56 total	16 int/24 fp	40 total	128 total	32/32	None	None
BHT Entries	4K ×9-bit	4K × 2-bit	2K ×2-bit	2K×2-bit	>= 512	4K ×2-bit	2K×2-bit	512 × 2-bit	16K×2-bit
TLB Entries	128/128	280/288	120 unified	128/128	321 / 64D	128I/65D	64 unified	64I/64D	128I/512D
Memory B/W	2.66GB/s	2.1GB/s	1.54GB/s	1.6GB/s	1.06GB/s	3.2GB/s	539 MB/s	1.9GB/s	4.8GB/s
Package	CPGA-588	PGA-462	LGA-544	SCC-1088	PGA-370	PGA-423	CPGA-527	CLGA-787	1368 FC-LGA
IC Process	0.18µ 6M	0.18µ 6M	0.25µ 2M	0.22µ 6m	0.18µ 6M	0.18µ 6M	0.25µ 4M	0.29µ 6M	0.18µ 7M
Die Size	115mm ²	117mm ²	477mm ²	163mm ²	106mm ²	217mm ²	204mm ²	126 mm ²	210mm ²
Transistors	15.4 million	37 million	130 million	23 million	24 million	42 million	7.2 million	3.8 million	29 million
Est mfg cost*	\$160	\$62	\$330	\$110	\$39	\$110	\$125	\$70	\$145
Power(Max)	75W*	76W	60W*	36W*	30W	55W(TDP)	25W*	20W*	65W
Availability	1Q01	4Q00	3Q00	4Q00	2Q00	4Q00	2Q00	3Q0	4Q00

Max issue: 4 instructions (many CPUs) Max rename registers: 128 (Pentium 4) Max BHT: 4K x 9 (Alpha 21264B), 16Kx2 (Ultra III) Max Window Size (OOO): 126 intructions (Pent. 4) Max Pipeline: 22/24 stages (Pentium 4)

Source: Microprocessor Report, www.MPRonline.com

Processor	Alpha 21264B	AMD Athlon	HP PA-8600	IBM Power 3-II	Intel PIII	Intel P4	MIPS R12000	Sun Ultra-II	Sun Ultra-III
System or Motherboard	Alpha ES40 Model 6	AMD GA-7ZM	HP9000 j6000	RS/6000 44P-170	Dell Prec. 420	Intel D850GB	SGI 2200	Sun Enterprs 450	Sun Blade 1000
Clock Rate	833MHz	1.2GHz	552MHz	450MHz	1GHz	1.5GHz	400MHz	480MHz	900MHz
External Cache	8MB	None	None	8MB	None	V None	V 8MB	8MB	8MB
164.gzip	392	n/a	376	230	545	553	226	165	349
175.vpr	452	n/a	421	285	354	298	384	212	383
176.gcc	617	n/a	577	350	401	588	313	232	500
181.mcf	441	n/a	384	498	276	473	563	356	474
186.crafty	694	n/a	472	304	523	497	334	175	439
197.parser	360	n/a	361	171	362	472	283	211	412
252.eon	645	n/a	395	280	615	650	360	209	465
253.perlbmk	526	n/a	406	215	614	703	246	247	457
254.gap	365	n/a	229	256	443	708	204	171	300
255.vortex	673	n/a	764	312	717	735	294	304	581
256.bzip2	560	n/a	349	258	396	420	334	237	500
300.twolf	658	n/a	479	414	394	403	451	243	473
SPECint_base2000	518	n/a	417	286	454	524	320	225	438
168.wupside	529	360	340	360	416	759	280	284	497
171.swim	1,156	506	761	279	493	1,244	300	285	752
172.mgrid	580	272	462	319	274	558	231	226	377
173.applu	424	298	563	327	280	641	237	150	221
177.mesa	713	302	300	330	541	553	289	273	469
178.galgel	558	468	569	429	335	537	989	735	1,266
179.art	1,540	213	419	969	410	514	995	920	990
183.equake	231	236	347	560	249	739	222	149	211
187.facerec	822	411	258	257	307	451	411	459	718
188.ammp	488	221	376	326	294	366	373	313	421
189.lucas	731	237	370	284	349	764	259	205	204
191.fma3d	528	365	302	340	297	427	192	207	302
200.sixtrack	340	256	286	234	170	257	199	159	273
301.aspi	553	278	523	349	3711.7	X 427	252	189	340
SPECfp_base2000	590	304	400	356	325	549	319	274	427

Dynamic Scheduling in Intel Pentium II, III

- Q: How pipeline 1 to 17 byte 80x86 instructions?
- □ Pentium does not pipeline 80x86 instructions
- Its decode unit translates the Intel instructions into 72-bit micro-operations (~ MIPS)
- Sends micro-operations to reorder buffer & reservation stations
- □ Many instructions translate to 1 to 4 micro-operations
- Complex 80x86 instructions are executed by a conventional microprogram (8K x 72 bits) that issues long sequences of micro-operations
- □ 14 clocks in total pipeline (~ 3 state machines)

Dynamic Scheduling in Intel Pentium

Parameter	80x86	micro-ops
Max. instructions issued/clock	3	6
Max. instr. complete exec./cloc	k	5
Max. instr. commited/clock		3
Window (Instrs in reorder buffe	er)	40
Number of reservations station	ns 20	
Number of rename registers	40	
No. integer functional units (FU No. floating point FUs No. SIMD FI. Pt. FUs	Js) 2 1 1	
No. memory Fus	1 load + 1 s	tore

Pipelining in Pentium



© by Tien-Fu Chen@CCU

chap4-52

Gradu-

ation

3 uops

/clk



Pentium III Die Photo



- □ EBL/BBL Bus logic, Front, Back
- □ MOB Memory Order Buffer
- □ Packed FPU MMX FI. Pt. (SSE)
- □ IEU Integer Execution Unit
- **Given Set Set :** FAU FI. Pt. Arithmetic Unit
- □ MIU Memory Interface Unit
- DCU Data Cache Unit
- PMH Page Miss Handler
- DTLB Data TLB
- □ BAC Branch Address Calculator
- RAT Register Alias Table
- SIMD Packed Fl. Pt.
- **RS Reservation Station**
- □ BTB Branch Target Buffer
- □ IFU Instruction Fetch Unit (+I\$)
- □ ID Instruction Decode
- **ROB Reorder Buffer**
- □ MS Micro-instruction Sequencer

chap4-54

© by Tien-Fu Chen@CCU