332
Advanced Computer Architecture
Chapter 4

Instruction Level Parallelism and Dynamic Execution

January 2004

Paul H J Kelly

These lecture notes are partly based on the course text, Hennessy and Patterson's *Computer Architecture, a quantitative approach (3rd ed)*, and on the lecture slides of David Patterson and John Kubiatowicz's Berkeley course *(CS252, Jan 2001)*

Recall from Pipelining Review

- Pipeline CPI = Ideal pipeline CPI + Structural Stalls + Data Hazard Stalls + Control Stalls
  - Ideal pipeline CPI: measure of the maximum performance attainable by the implementation
  - Structural hazards: HW cannot support this combination of instructions
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline
  - Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps)

Ideas to Reduce Stalls

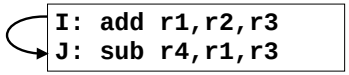
	Technique	Reduces
Textbook Chapter 3	Dynamic scheduling	Data hazard stalls
	Dynamic branch prediction	Control stalls
	Issuing multiple instructions per cycle	Ideal CPI
	Speculation	Data and control stalls
	Dynamic memory disambiguation	Data hazard stalls involving memory
Textbook Chapter 4	Loop unrolling	Control hazard stalls
	Basic compiler pipeline scheduling	Data hazard stalls
	Compiler dependence analysis	Ideal CPI and data hazard stalls
	Software pipelining and trace scheduling	Ideal CPI and data hazard stalls
	Compiler speculation	Ideal CPI, data and control stalls

Instruction-Level Parallelism (ILP)

- Basic Block (BB) ILP is quite small
  - BB: a straight-line code sequence with no branches in except to the entry and no branches out except at the exit
  - average dynamic branch frequency 15% to 25%
    => 4 to 7 instructions execute between a pair of branches
  - Plus instructions in BB likely to depend on each other
- To obtain substantial performance enhancements, we must exploit ILP across multiple basic blocks
- Simplest: loop-level parallelism to exploit parallelism among iterations of a loop
  - Vector is one way
  - If not vector, then either dynamic via branch prediction or static via loop unrolling by compiler

Data Dependence and Hazards

- $Instr_J$ is data dependent on $Instr_I$
  $Instr_J$ tries to read operand before $Instr_I$ writes it



- or $Instr_J$ is data dependent on $Instr_K$ which is dependent on $Instr_I$
- Caused by a "True Dependence" (compiler term)
- If true dependence caused a hazard in the pipeline, called a Read After Write (RAW) hazard

Data Dependence and Hazards

- Dependences are a property of programs
- Presence of dependence indicates potential for a hazard, but actual hazard and length of any stall is a property of the pipeline
- Importance of the data dependencies

1) indicates the possibility of a hazard

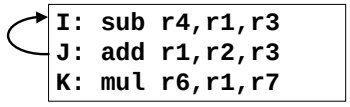
2) determines order in which results must be calculated

3) sets an upper bound on how much parallelism can possibly be exploited

- Today looking at HW schemes to avoid hazard

Name Dependence #1: Anti-dependence

- Name dependence: when 2 instructions use same register or memory location, called a name, but no flow of data between the instructions associated with that name
- There are two kinds:
- Name dependence #1: anti-dependence/WAR
  - $Instr_J$ writes operand ***before*** $Instr_I$ reads it

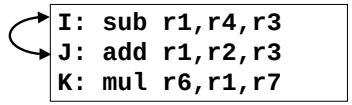


Called an "anti-dependence" by compiler writers.
This results from reuse of the name "r1"

  - If anti-dependence caused a hazard in the pipeline, called a Write After Read (WAR) hazard

Name Dependence #2: Output dependence

- $Instr_J$ writes operand ***before*** $Instr_I$ writes it.



- Called an "output dependence" by compiler writers
  This also results from the reuse of name "r1"
- If anti-dependence caused a hazard in the pipeline, called a Write After Write (WAW) hazard

ILP and Data Hazards

- HW/SW must preserve program order: order instructions would execute in if executed sequentially 1 at a time as determined by original source program
- HW/SW goal: exploit parallelism by preserving program order only where it affects the outcome of the program
- Instructions involved in a name dependence can execute simultaneously if name used in instructions is changed so instructions do not conflict
  - Register renaming resolves name dependence for regs
  - Either by compiler or by HW

Control Dependencies

- Every instruction is control dependent on some set of branches, and, in general, these control dependencies must be preserved to preserve program order

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

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

Control Dependence Ignored

- Control dependence need not be preserved
  - willing to execute instructions that should not have been executed, thereby violating the control dependences, if can do so without affecting correctness of the program
- Instead, two properties critical to program correctness are **exception behavior** and **data flow**

Exception Behavior

- Preserving exception behavior => any changes in instruction execution order must not change how exceptions are raised in program (=> no new exceptions)
- Example:

```
     DADDU      R2,R3,R4
     BEQZ       R2,L1
     LW         R1,0(R2)
L1:
```

- Problem with moving LW before BEQZ?

Data Flow

- Data flow: actual flow of data values among instructions that produce results and those that consume them
  - branches make flow dynamic, determine which instruction is supplier of data
- Example:

```
DADDU    R1,R2,R3
BEQZ     R4,L
DSUBU    R1,R5,R6
L:  …
OR       R7,R1,R8
```

- OR depends on DADDU or DSUBU? Must preserve data flow on execution

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

HW Schemes: Instruction Parallelism

- 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 and allows out-of-order completion
- Will distinguish when an instruction *begins execution* and when it *completes execution*; between these two times, the instruction is *in execution*
- In a dynamically scheduled pipeline, all instructions pass through issue stage in order (in-order issue)

Dynamic Scheduling Step 1

- Simple pipeline had 1 stage to check both structural and data hazards: Instruction Decode (ID), also called Instruction Issue
- Split the ID pipe stage of simple 5-stage pipeline into 2 stages:
- *Issue*—Decode instructions, check for structural hazards
- *Read operands*—Wait until no data hazards, then read operands

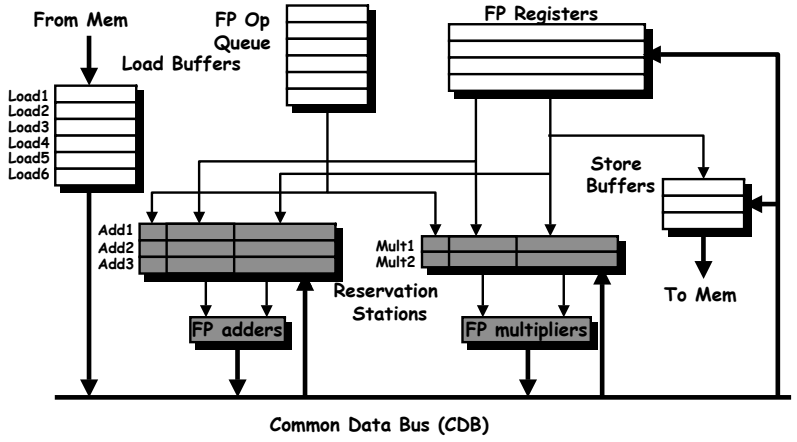
A Dynamic Algorithm: Tomasulo's Algorithm

- For IBM 360/91 (before caches!)
- Goal: High Performance without special compilers
- Small number of floating point registers (4 in 360) prevented interesting compiler scheduling of operations
  - This led Tomasulo to try to figure out how to get more effective registers — renaming in hardware!
- Why study a 1966 Computer?
- The descendents of this have flourished!
  - Alpha 21264, HP 8000, MIPS 10000/R12000, Pentium II/III/4, AMD K5,K6,Athlon, PowerPC 603/604/G3/G4/G5, …

Tomasulo Algorithm

- Control & buffers distributed with Function Units (FU)
  - FU buffers called "reservation stations"; have pending operands
- Registers in instructions replaced by values or pointers to reservation stations(RS); called register renaming ;
  - avoids WAR, WAW hazards
  - More reservation stations than registers, so can do optimizations compilers can't
- Results to FU from RS, not through registers, over Common Data Bus that broadcasts results to all FUs
- Load and Stores treated as FUs with RSs as well
- Integer instructions can go past branches, allowing FP ops beyond basic block in FP queue

Tomasulo Organization



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

Busy: Indicates reservation station or FU is busy

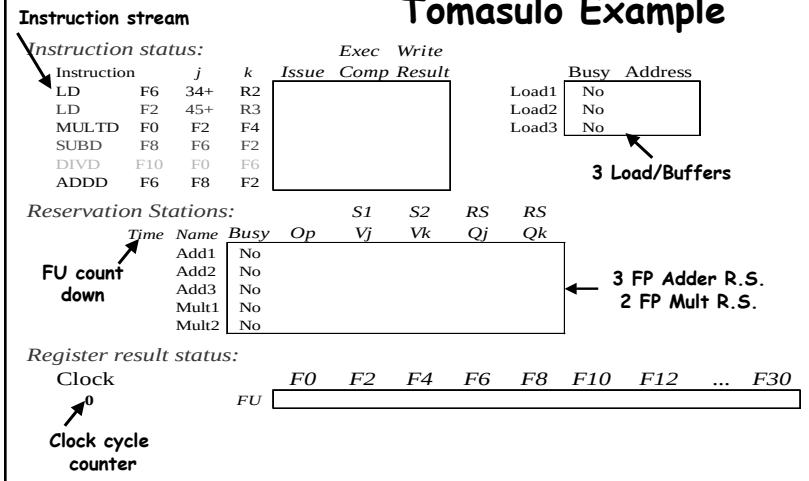
Register result status—Indicates which functional unit will write each register, if one exists. Blank when no pending instructions that will write that register.

Three Stages of Tomasulo Algorithm

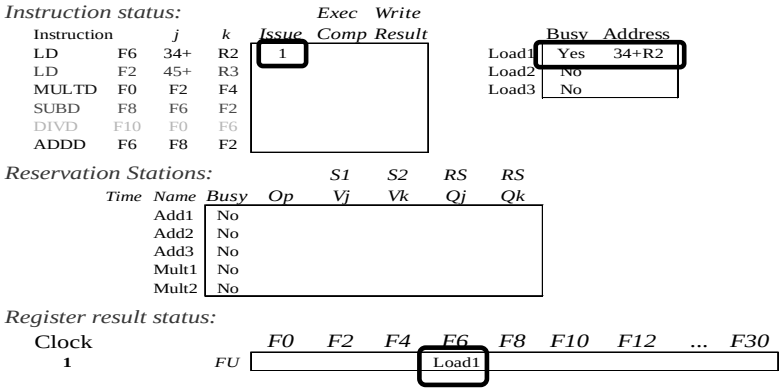
1. Issue—get instruction from FP Op Queue
   If reservation station free (no structural hazard),
   control issues instr & sends operands (renames registers).
2. Execute—operate on operands (EX)
   When both operands ready then execute;
   if not ready, watch Common Data Bus for result
3. Write result—finish execution (WB)
   Write on Common Data Bus to all awaiting units;
   mark reservation station available

- Normal data bus: data + destination ("go to" bus)
- Common data bus: data + source ("come from" bus)
  - 64 bits of data + 4 bits of Functional Unit source address
  - Write if matches expected Functional Unit (produces result)
  - Does the broadcast
- Example speed:
  3 clocks for Fl .pt. +,-; 10 for * ; 40 clks for /

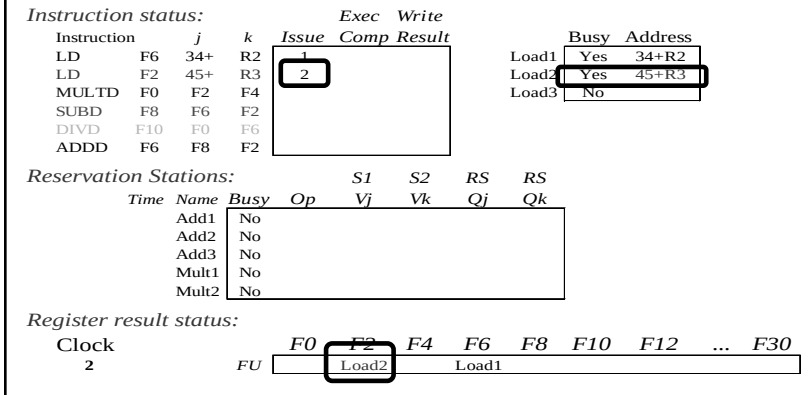
Tomasulo Example



Tomasulo Example Cycle 1



Tomasulo Example Cycle 2



Note: Can have multiple loads outstanding

Tomasulo Example Cycle 3

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	
LD	F2	45+	R3	2		
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		R(F4)	Load2	
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
3	FU	Mult1	Load2		Load1					

- Note: registers names are removed ("renamed") in Reservation Stations; MULT issued
- Load1 completing; what is waiting for Load1?

Tomasulo Example Cycle 4

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	Yes	SUBD	M(A1)			Load2
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		R(F4)	Load2	
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
4	FU	Mult1	Load2		M(A1)	Add1				

- Load2 completing; what is waiting for Load2?

Tomasulo Example Cycle 5

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
2	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	No					
	Add3	No					
10	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
5	FU	Mult1	M(A2)		M(A1)	Add1	Mult2			

- Timer starts down for Add1, Mult1

Tomasulo Example Cycle 6

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
1	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	Yes	ADDD		M(A2)	Add1	
	Add3	No					
9	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU	Mult1	M(A2)		Add2	Add1	Mult2			

- Issue ADDD here despite name dependency on F6?

Tomasulo Example Cycle 7

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
0	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	Yes	ADDD		M(A2)	Add1	
	Add3	No					
8	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
7	FU	Mult1	M(A2)		Add2	Add1	Mult2			

- **Add1 (SUBD) completing; what is waiting for it?**

Tomasulo Example Cycle 8

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
2	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
7	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
8	FU	Mult1	M(A2)		Add2	(M-M)	Mult2			

Tomasulo Example Cycle 9

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
1	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
6	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
9	FU	Mult1	M(A2)		Add2	(M-M)	Mult2			

Tomasulo Example Cycle 10

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
0	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
5	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
10	FU	Mult1	M(A2)		Add2	(M-M)	Mult2			

- **Add2 (ADDD) completing; what is waiting for it?**

Tomasulo Example Cycle 11

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
4	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
11	FU	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

- **Write result of ADDD here?**
- **All quick instructions complete in this cycle!**

Tomasulo Example Cycle 12

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
3	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
12	FU	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

Tomasulo Example Cycle 13

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
2	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
13	FU	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

Tomasulo Example Cycle 14

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
1	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
14	FU	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

Tomasulo Example Cycle 15

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
0	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
15	FU	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

- **Mult1 (MULTD) completing; what is waiting for it?**

Tomasulo Example Cycle 16

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
40	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
16	FU	M*F4	M(A2)		(M-M+M	(M-M)	Mult2			

- **Just waiting for Mult2 (DIVD) to complete**

Skip a few cycles: Cycle 55

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
1	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
55	FU	M*F4	M(A2)		(M-M+M	(M-M)	Mult2			

Tomasulo Example Cycle 56

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5	56	
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
0	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
56	FU	M*F4	M(A2)		(M-M+M	(M-M)	Mult2			

- **Mult2 (DIVD) is completing; what is waiting for it?**

Tomasulo Example Cycle 57

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5	56	57
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
56	FU	M*F4	M(A2)		(M-M+M	(M-M)	Result			

- Once again: In-order issue, out-of-order execution and out-of-order completion.

Tomasulo Drawbacks

- Complexity
  - delays of 360/91, MIPS 10000, Alpha 21264, IBM PPC 620
  - Many associative stores (CDB) at high speed
- Performance limited by Common Data Bus
  - Each CDB must go to multiple functional units ⇒high capacitance, high wiring density
  - Number of functional units that can complete per cycle limited to one!
    - Multiple CDBs ⇒ more FU logic for parallel assoc stores
- Non-precise interrupts!
  - We will address this later

Tomasulo Loop Example

```
Loop: LD      F0   0    R1
      MULTD   F4   F0   F2
      SD      F4   0    R1
      SUBI    R1   R1   #8
      BNEZ    R1   Loop
```

- This time assume Multiply takes 4 clocks
- Assume 1st load takes 8 clocks (L1 cache miss), 2nd load takes 1 clock (hit)
- To be clear, will show clocks for SUBI, BNEZ
  - Reality: integer instructions ahead of Fl. Pt. Instructions
- Show 2 iterations

Loop Example

Instruction status:

Iter-ation Count	ITER	Instruction	j	k	Exec Issue	Exec Comp	Write Result
	1	LD	F0	0	R1		
	1	MULTD	F4	F0	F2		
	1	SD	F4	0	R1		
	2	LD	F0	0	R1		
	2	MULTD	F4	F0	F2		
	2	SD	F4	0	R1		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

Added Store Buffers

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Code:

```
LD      F0   0    R1
MULTD   F4   F0   F2
SD      F4   0    R1
SUBI    R1   R1   #8
BNEZ    R1   Loop
```

Instruction Loop

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
0	80	Fu									

Value of Register used for address, iteration control

Loop Example Cycle 1

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Code:			
	Add1	No						LD	F0	0	R1
	Add2	No						MULTD	F4	F0	F2
	Add3	No						SD	F4	0	R1
	Mult1	No						SUBI	R1	R1	#8
	Mult2	No						BNEZ	R1	Loop	

Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
1	80	Load1								

Loop Example Cycle 2

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1		
1	MULTD F4	F0	F2	2		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Code:			
	Add1	No						LD	F0	0	R1
	Add2	No						MULTD	F4	F0	F2
	Add3	No						SD	F4	0	R1
	Mult1	Yes	Multd		R(F2)	Load1		SUBI	R1	R1	#8
	Mult2	No						BNEZ	R1	Loop	

Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
2	80	Load1		Mult1						

Loop Example Cycle 3

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1		
1	MULTD F4	F0	F2	2		
1	SD F4	0	R1	3		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Code:			
	Add1	No						LD	F0	0	R1
	Add2	No						MULTD	F4	F0	F2
	Add3	No						SD	F4	0	R1
	Mult1	Yes	Multd		R(F2)	Load1		SUBI	R1	R1	#8
	Mult2	No						BNEZ	R1	Loop	

Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
3	80	Load1		Mult1						

- Implicit renaming sets up data flow graph

Loop Example Cycle 4

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1		
1	MULTD F4	F0	F2	2		
1	SD F4	0	R1	3		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	Qk	Code:			
	Add1	No						LD	F0	0	R1
	Add2	No						MULTD	F4	F0	F2
	Add3	No						SD	F4	0	R1
	Mult1	Yes	Multd		R(F2)	Load1		SUBI	R1	R1	#8
	Mult2	No						BNEZ	R1	Loop	

Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
4	80	Load1		Mult1						

- Dispatching SUBI Instruction (not in FP queue)

Loop Example Cycle 5

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1		
1	MULTD F4	F0	F2	2		
1	SD F4	0	R1	3		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk	Code:			
	Add1	No						LD	F0	0	R1
	Add2	No						MULTD	F4	F0	F2
	Add3	No						SD	F4	0	R1
	Mult1	Yes	Multd		R(F2)	Load1		SUBI	R1	R1	#8
	Mult2	No						BNEZ	R1	Loop	

Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
5	72	Load1		Mult1						

- And, BNEZ instruction (not in FP queue)

Loop Example Cycle 6

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1		
1	MULTD F4	F0	F2	2		
1	SD F4	0	R1	3		
2	LD F0	0	R1	6		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk	Code:			
	Add1	No						LD	F0	0	R1
	Add2	No						MULTD	F4	F0	F2
	Add3	No						SD	F4	0	R1
	Mult1	Yes	Multd		R(F2)	Load1		SUBI	R1	R1	#8
	Mult2	No						BNEZ	R1	Loop	

Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
6	72	Load2		Mult1						

- Notice that F0 never sees Load from location 80

Loop Example Cycle 7

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1		
1	MULTD F4	F0	F2	2		
1	SD F4	0	R1	3		
2	LD F0	0	R1	6		
2	MULTD F4	F0	F2	7		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk	Code:			
	Add1	No						LD	F0	0	R1
	Add2	No						MULTD	F4	F0	F2
	Add3	No						SD	F4	0	R1
	Mult1	Yes	Multd		R(F2)	Load1		SUBI	R1	R1	#8
	Mult2	Yes	Multd		R(F2)	Load2		BNEZ	R1	Loop	

Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
7	72	Load2		Mult2						

- Register file completely detached from computation
- First and Second iteration completely overlapped

Loop Example Cycle 8

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1		
1	MULTD F4	F0	F2	2		
1	SD F4	0	R1	3		
2	LD F0	0	R1	6		
2	MULTD F4	F0	F2	7		
2	SD F4	0	R1	8		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk	Code:			
	Add1	No						LD	F0	0	R1
	Add2	No						MULTD	F4	F0	F2
	Add3	No						SD	F4	0	R1
	Mult1	Yes	Multd		R(F2)	Load1		SUBI	R1	R1	#8
	Mult2	Yes	Multd		R(F2)	Load2		BNEZ	R1	Loop	

Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
8	72	Load2		Mult2						

Loop Example Cycle 9

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1	9	
1	MULTD F4	F0	F2	2		
1	SD F4	0	R1	3		
2	LD F0	0	R1	6		
2	MULTD F4	F0	F2	7		
2	SD F4	0	R1	8		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	Yes	Multd		R(F2)	Load2	

Code:			
LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
9	72	Fu	Load2			Mult2					

- Load1 completing: who is waiting?
- Note: Dispatching SUBI

Loop Example Cycle 10

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1	9	10
1	MULTD F4	F0	F2	2		
1	SD F4	0	R1	3		
2	LD F0	0	R1	6	10	
2	MULTD F4	F0	F2	7		
2	SD F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
4	Mult1	Yes	Multd	M[80]	R(F2)		
	Mult2	Yes	Multd		R(F2)	Load2	

Code:			
LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
10	64	Fu	Load2			Mult2					

- Load2 completing: who is waiting?
- Note: Dispatching BNEZ

Loop Example Cycle 11

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1	9	10
1	MULTD F4	F0	F2	2		
1	SD F4	0	R1	3		
2	LD F0	0	R1	6	10	11
2	MULTD F4	F0	F2	7		
2	SD F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
3	Mult1	Yes	Multd	M[80]	R(F2)		
4	Mult2	Yes	Multd	M[72]	R(F2)		

Code:			
LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
11	64	Fu	Load3			Mult2					

- Next load in sequence

Loop Example Cycle 12

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1	9	10
1	MULTD F4	F0	F2	2		
1	SD F4	0	R1	3		
2	LD F0	0	R1	6	10	11
2	MULTD F4	F0	F2	7		
2	SD F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
2	Mult1	Yes	Multd	M[80]	R(F2)		
3	Mult2	Yes	Multd	M[72]	R(F2)		

Code:			
LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
12	64	Fu	Load3			Mult2					

- Why not issue third multiply?

Loop Example Cycle 13

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1	9	10
1	MULTD F4	F0	F2	2		
1	SD F4	0	R1	3		
2	LD F0	0	R1	6	10	11
2	MULTD F4	F0	F2	7		
2	SD F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk	Code:			
	Add1	No						LD	F0	0	R1
	Add2	No						MULTD	F4	F0	F2
	Add3	No						SD	F4	0	R1
1	Mult1	Yes	Multd	M[80]	R(F2)			SUBI	R1	R1	#8
2	Mult2	Yes	Multd	M[72]	R(F2)			BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
13	64	Fu	Load3		Mult2						

- Why not issue third store?

Loop Example Cycle 14

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1	9	10
1	MULTD F4	F0	F2	2	14	
1	SD F4	0	R1	3		
2	LD F0	0	R1	6	10	11
2	MULTD F4	F0	F2	7		
2	SD F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk	Code:			
	Add1	No						LD	F0	0	R1
	Add2	No						MULTD	F4	F0	F2
	Add3	No						SD	F4	0	R1
0	Mult1	Yes	Multd	M[80]	R(F2)			SUBI	R1	R1	#8
1	Mult2	Yes	Multd	M[72]	R(F2)			BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
14	64	Fu	Load3		Mult2						

- Mult1 completing. Who is waiting?

Loop Example Cycle 15

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1	9	10
1	MULTD F4	F0	F2	2	14	15
1	SD F4	0	R1	3		
2	LD F0	0	R1	6	10	11
2	MULTD F4	F0	F2	7	15	
2	SD F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	[80]*R2
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk	Code:			
	Add1	No						LD	F0	0	R1
	Add2	No						MULTD	F4	F0	F2
	Add3	No						SD	F4	0	R1
	Mult1	No						SUBI	R1	R1	#8
0	Mult2	Yes	Multd	M[72]	R(F2)			BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
15	64	Fu	Load3		Mult2						

- Mult2 completing. Who is waiting?

Loop Example Cycle 16

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1	9	10
1	MULTD F4	F0	F2	2	14	15
1	SD F4	0	R1	3		
2	LD F0	0	R1	6	10	11
2	MULTD F4	F0	F2	7	15	16
2	SD F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	[80]*R2
Store2	Yes	72	[72]*R2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk	Code:			
	Add1	No						LD	F0	0	R1
	Add2	No						MULTD	F4	F0	F2
	Add3	No						SD	F4	0	R1
4	Mult1	Yes	Multd		R(F2)	Load3		SUBI	R1	R1	#8
	Mult2	No						BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
16	64	Fu	Load3		Mult1						

Loop Example Cycle 17

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1	9	10
1	MULTD F4	F0	F2	2	14	15
1	SD F4	0	R1	3		
2	LD F0	0	R1	6	10	11
2	MULTD F4	F0	F2	7	15	16
2	SD F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	[80]*R2
Store2	Yes	72	[72]*R2
Store3	Yes	64	Mult1

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk	Code:			
	Add1	No						LD	F0	0	R1
	Add2	No						MULTD	F4	F0	F2
	Add3	No						SD	F4	0	R1
	Mult1	Yes	Multd		R(F2)	Load3		SUBI	R1	R1	#8
	Mult2	No						BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
17	64	Fu	Load3		Mult1						

Loop Example Cycle 18

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1	9	10
1	MULTD F4	F0	F2	2	14	15
1	SD F4	0	R1	3	18	
2	LD F0	0	R1	6	10	11
2	MULTD F4	F0	F2	7	15	16
2	SD F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	[80]*R2
Store2	Yes	72	[72]*R2
Store3	Yes	64	Mult1

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk	Code:			
	Add1	No						LD	F0	0	R1
	Add2	No						MULTD	F4	F0	F2
	Add3	No						SD	F4	0	R1
	Mult1	Yes	Multd		R(F2)	Load3		SUBI	R1	R1	#8
	Mult2	No						BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
18	64	Fu	Load3		Mult1						

Loop Example Cycle 19

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1	9	10
1	MULTD F4	F0	F2	2	14	15
1	SD F4	0	R1	3	18	19
2	LD F0	0	R1	6	10	11
2	MULTD F4	F0	F2	7	15	16
2	SD F4	0	R1	8	19	

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	No		
Store2	Yes	72	[72]*R2
Store3	Yes	64	Mult1

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk	Code:			
	Add1	No						LD	F0	0	R1
	Add2	No						MULTD	F4	F0	F2
	Add3	No						SD	F4	0	R1
	Mult1	Yes	Multd		R(F2)	Load3		SUBI	R1	R1	#8
	Mult2	No						BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
19	56	Fu	Load3		Mult1						

Loop Example Cycle 20

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1	9	10
1	MULTD F4	F0	F2	2	14	15
1	SD F4	0	R1	3	18	19
2	LD F0	0	R1	6	10	11
2	MULTD F4	F0	F2	7	15	16
2	SD F4	0	R1	8	19	20

	Busy	Addr	Fu
Load1	Yes	56	
Load2	No		
Load3	Yes	64	
Store1	No		
Store2	No		
Store3	Yes	64	Mult1

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk	Code:			
	Add1	No						LD	F0	0	R1
	Add2	No						MULTD	F4	F0	F2
	Add3	No						SD	F4	0	R1
	Mult1	Yes	Multd		R(F2)	Load3		SUBI	R1	R1	#8
	Mult2	No						BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
20	56	Fu	Load1		Mult1						

- **Once again: In-order issue, out-of-order execution and out-of-order completion.**

Why can Tomasulo overlap iterations of loops?

- **Register renaming**
  - Multiple iterations use different physical destinations for registers (dynamic loop unrolling).
- **Reservation stations**
  - Permit instruction issue to advance past integer control flow operations
  - Also buffer old values of registers - totally avoiding the WAR stall that we saw in the scoreboard.
- **Other perspective: Tomasulo building data flow dependency graph on the fly.**

Tomasulo's scheme offers two major advantages

(1) the distribution of the hazard detection logic

- distributed reservation stations and the CDB
- If multiple instructions waiting on single result, & each instruction has other operand, then instructions can be released simultaneously by broadcast on CDB
- If a centralized register file were used, the units would have to read their results from the registers when register buses are available.

(2) the elimination of stalls for WAW and WAR hazards

What about Precise Interrupts?

- **Tomasulo had:**

  In-order issue, out-of-order execution, and out-of-order completion

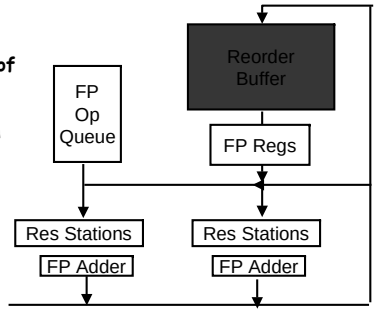
- **Need to "fix" the out-of-order completion aspect so that we can find precise breakpoint in instruction stream.**

Relationship between precise interrupts and speculation:

- **Speculation is a form of guessing.**
- **Important for branch prediction:**
  - Need to "take our best shot" at predicting branch direction.
- **If we speculate and are wrong, need to back up and restart execution at point at which we predicted incorrectly:**
  - This is exactly same as precise exceptions!
- **Technique for both precise interrupts/exceptions and speculation: *in-order completion or commit***

HW support for precise interrupts

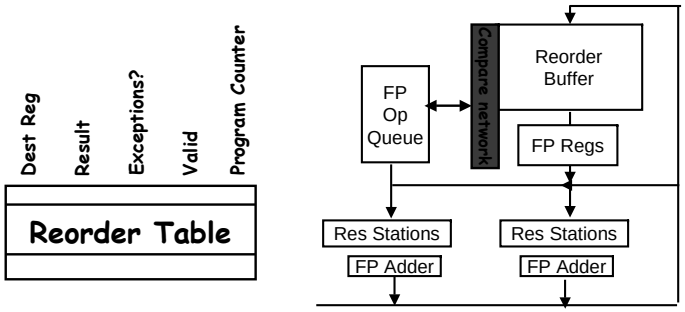
- Need HW buffer for results of uncommitted instructions: *reorder buffer*
  - 3 fields: instr, destination, value
  - Use reorder buffer number instead of reservation station when execution completes
  - Supplies operands between execution complete & commit
  - (Reorder buffer can be operand source => more registers like RS)
  - Instructions commit
  - Once instruction commits, result is put into register
  - As a result, easy to undo speculated instructions on mispredicted branches or exceptions



Four Steps of Speculative Tomasulo Algorithm

1. Issue—get instruction from FP Op Queue
   If reservation station and reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination (this stage sometimes called "dispatch")
2. Execution—operate on operands (EX)
   When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute; checks RAW (sometimes called "issue")
3. Write result—finish execution (WB)
   Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.
4. Commit—update register with reorder result
   When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer. Mispredicted branch flushes reorder buffer (sometimes called "graduation")

What are the hardware complexities with reorder buffer (ROB)?



- How do you find the latest version of a register?
  - Looks like we need associative comparison network
  - Could use future file or just use the register result status buffer to track which specific reorder buffer has received the value
- Need as many ports on ROB as register file

See S. Weiss and J. E. Smith, "Instruction Issue Logic for Pipelined Supercomputers". ISCA, 1984 (http://citeseer.nj.nec.com/weiss84instruction.html)

Some subleties…

- It's vital to reduce the branch misprediction penalty. Does the Tomasulo+ROB scheme described here roll-back as soon as the branch is found to be mispredicted?
- Stores are buffered in the ROB, and committed only when the instruction is committed. A load can be issued while several stores (perhaps to the same address) are uncommitted. We need to make sure the load gets the right data.
- What if a second conditional branch is encountered, before the outcome of the first is resolved?
- This discussion has assumed a single-issue machine. How can these ideas be extended to allow multiple instructions to be issued per cycle?
  - Issue
  - Monitoring CDBs for completion
  - Handling multiple commits per cycle

Tomasulo + ROB: Summary

- Reservations stations: *implicit register renaming* to larger set of registers + buffering source operands
  - Prevents registers as bottleneck
  - Avoids WAR, WAW hazards of Scoreboard (see textbook)
  - Allows loop unrolling in HW
- Not limited to basic blocks (integer units gets ahead, beyond branches)
- Today, helps cache misses as well
  - Don't stall for L1 Data cache miss (insufficient ILP for L2 miss?)
- Lasting Contributions
  - Dynamic scheduling
  - Register renaming
  - Load/store disambiguation
- 360/91 descendants are Pentium III; PowerPC 604; MIPS R10000; HP-PA 8000; Alpha 21264 and more

Tomasulo Algorithm and Branch Prediction

- 360/91 predicted branches, but did not speculate: pipeline stopped until the branch was resolved
  - No speculation; only instructions that can complete
- Speculation with Reorder Buffer allows execution past branch, and then discard if branch fails
  - just need to hold instructions in buffer until branch can commit

Case for Branch Prediction when Issue N instructions per clock cycle

1. Branches will arrive up to *n* times faster in an *n*-issue processor
2. Amdahl's Law => relative impact of the control stalls will be larger with the lower potential CPI in an *n*-issue processor

7 Branch Prediction Schemes

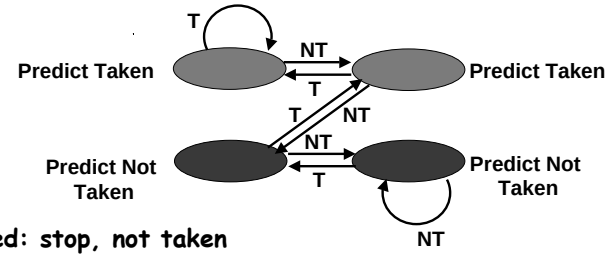
1. 1-bit Branch-Prediction Buffer
2. 2-bit Branch-Prediction Buffer
3. Correlating Branch Prediction Buffer
4. Tournament Branch Predictor
5. Branch Target Buffer
6. Integrated Instruction Fetch Units
7. Return Address Predictors

Dynamic Branch Prediction

- Performance = *f*(accuracy, cost of misprediction)
- Branch History Table: Lower bits of PC address index table of 1-bit values
  - Says whether or not branch taken last time
  - No address check (saves HW, but may not be right branch)
- Problem: in a loop, 1-bit BHT will cause 2 mispredictions (avg is 9 iterations before exit):
  - End of loop case, when it exits instead of looping as before
  - First time through loop on *next* time through code, when it predicts *exit* instead of looping
  - Only 80% accuracy even if loop 90% of the time

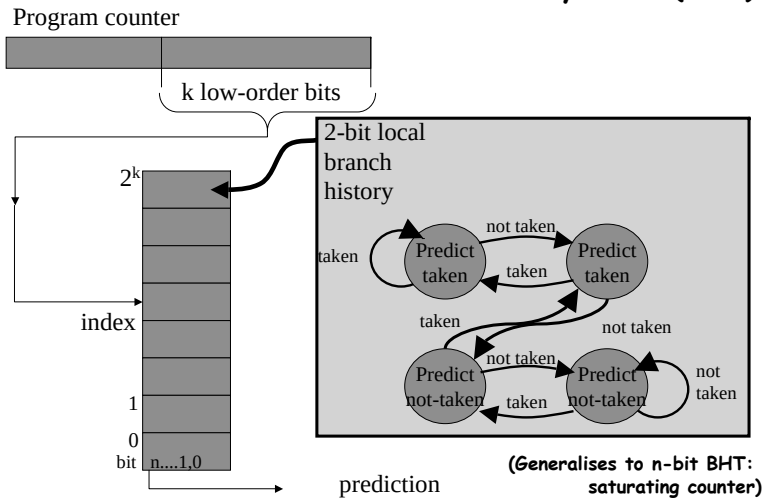
Dynamic Branch Prediction (Jim Smith, 1981)

- Solution: 2-bit scheme where change prediction only if get misprediction *twice:* (Figure 3.7, p. 198)

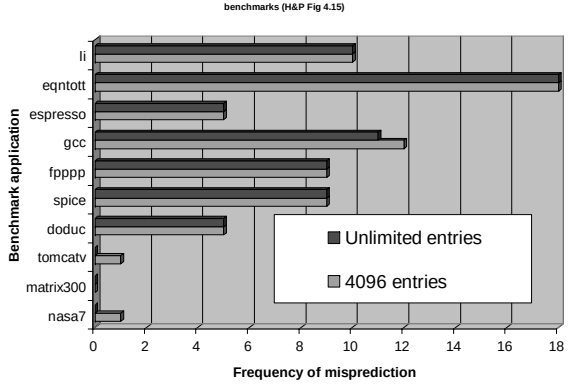


- Red: stop, not taken
- Green: go, taken
- Adds *hysteresis* to decision making process

The 2-bit branch history table (BHT)



(Generalises to n-bit BHT: saturating counter)



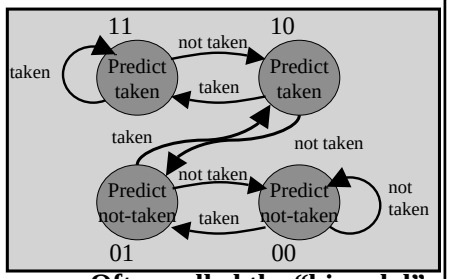
n-bit BHT - how well does it work?

- 2-bit predictor often very good, sometimes awful
- Little evidence that BHT capacity is an issue
- 1-bit is usually worse, 3-bit is not usefully better

N-bit BHT - why does it work so well?

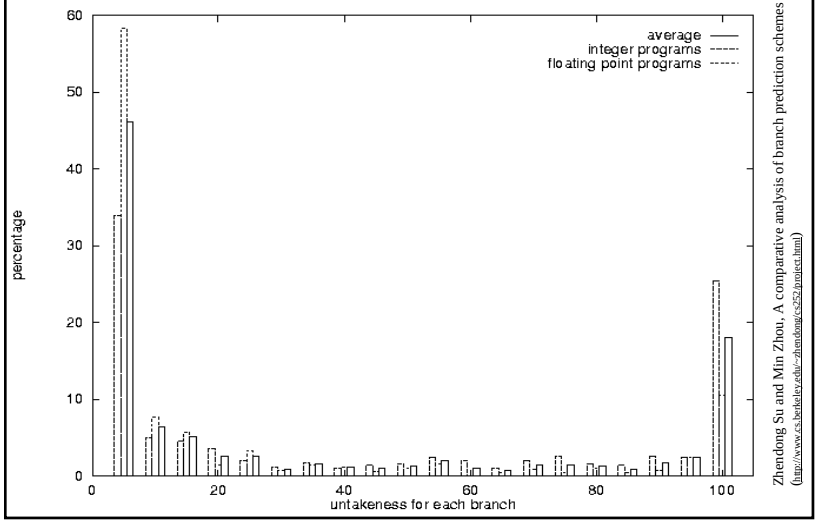
- n-bit BHT predictor essentially based on a saturating counter: taken increments, not-taken decrements
- predict taken if most significant bit is set

- Most branches are highly ***biased:*** either almost-always taken, or almost-always not-taken
- Works badly for branches which aren't



Often called the "bimodal" predictor

Bias



Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes (http://www.cs.berkeley.edu/~zhendong/cs252-project.html)

Is local history all there is to it?

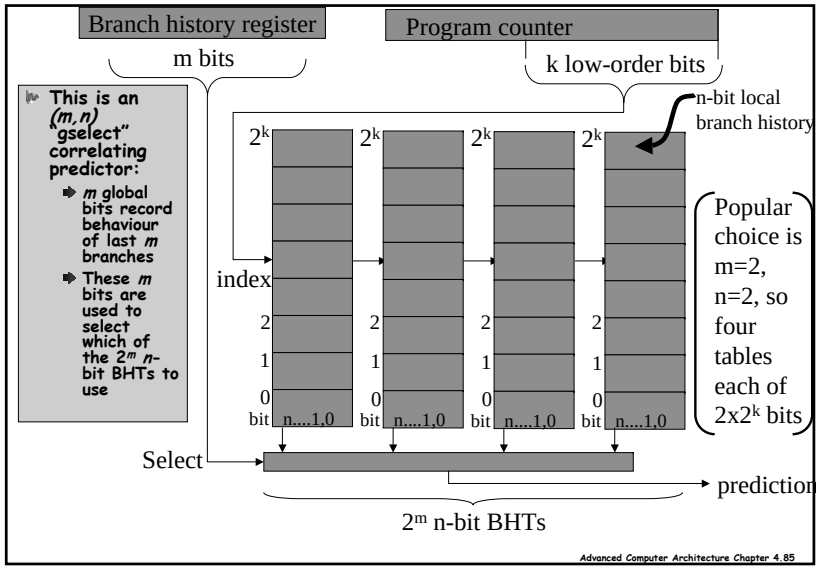
- The bimodal predictor uses the BHT to record "local history" - the prediction information used to predict a particular branch is determined only by its memory address
- Consider the following sequence:

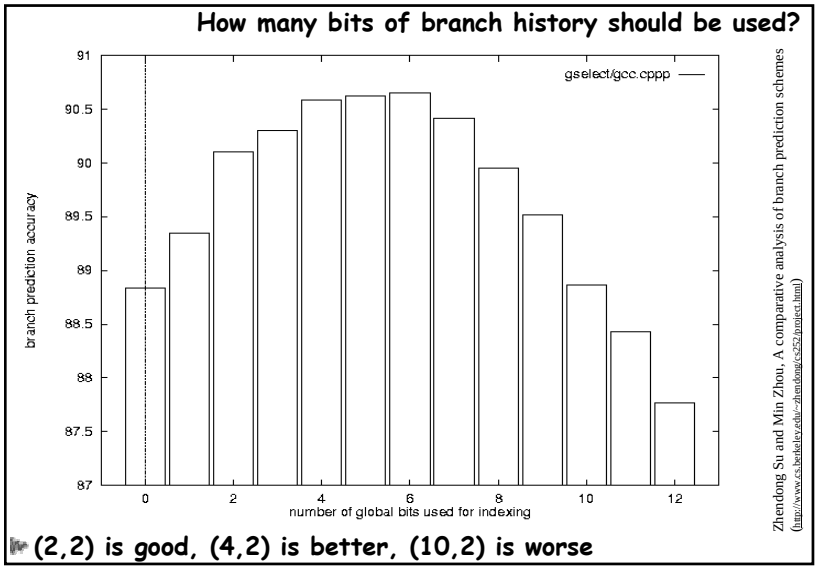
```
if (C1)  then
   S1;
endif
if (C2) then
   S2;
endif
if (C3) then
   S3;
endif
```

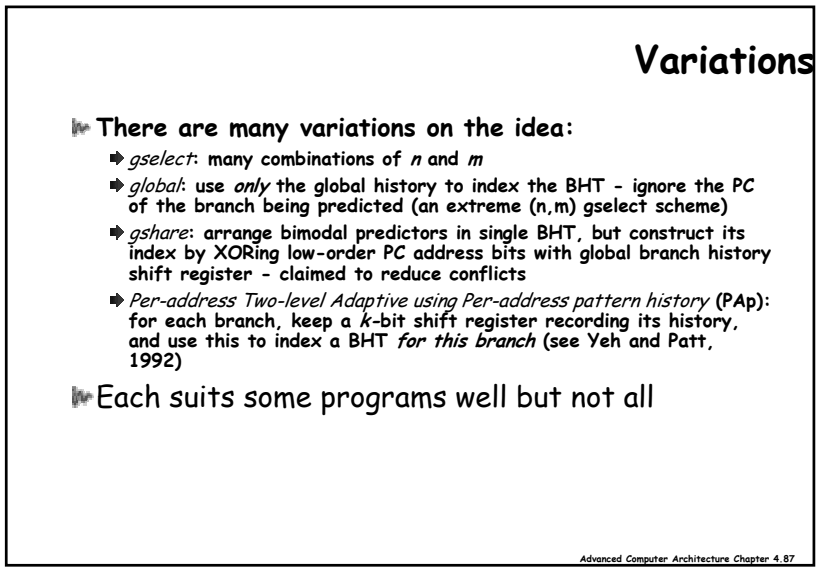
- It is very likely that condition C2 is correlated with C1 - and that C3 is correlated with C1 and C2
- How can we use this observation?

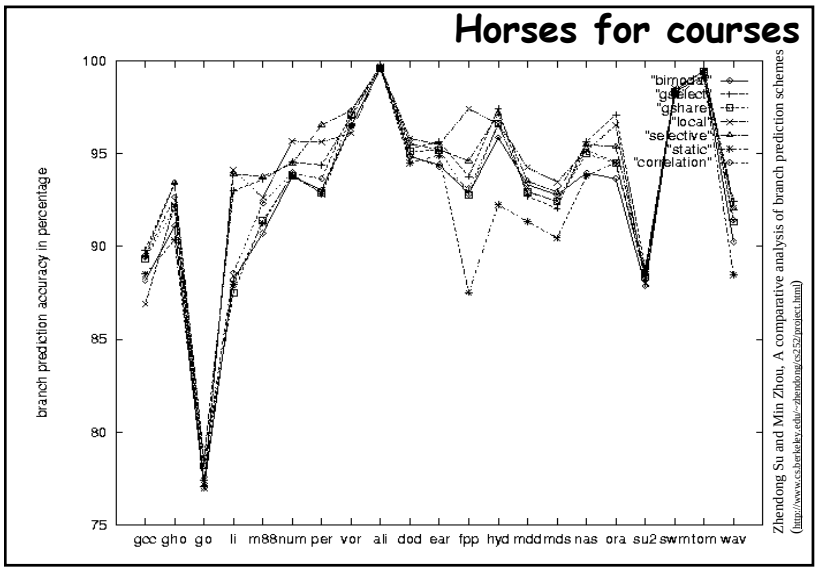
Global history

- Definition: ***Global history***. The taken - not-taken history for all previously-executed branches.
- Idea: use global history to improve branch prediction
- Compromise: use *m* most recently-executed branches
- Implementation: keep an *m*-bit Branch History Register (BHR) - a shift register recording taken - not-taken direction of the last m branches
- Question: How to combine local information with global information?

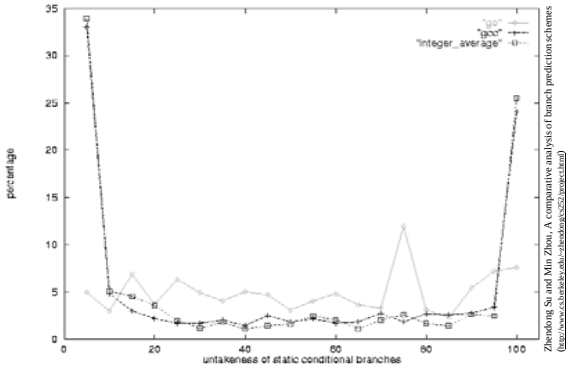

Branch history register
Program counter
m bits
k low-order bits
This is an (m,n) "gselect" correlating predictor:
m global bits record behaviour of last m branches
These m bits are used to select which of the 2^m n-bit BHTs to use
n-bit local branch history
2^k
index
2
1
0
bit
n....1,0
Popular choice is m=2, n=2, so four tables each of 2x2^k bits
Select
prediction
2^m n-bit BHTs
Advanced Computer Architecture Chapter 4.85


How many bits of branch history should be used?
gselect/gcc.cppp
branch prediction accuracy
number of global bits used for indexing
91
90.5
90
89.5
89
88.5
88
87.5
87
0
2
4
6
8
10
12
Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes (http://www.cs.berkeley.edu/~zhendong/cs252/project.html)
(2,2) is good, (4,2) is better, (10,2) is worse


Variations
There are many variations on the idea:
gselect: many combinations of n and m
global: use only the global history to index the BHT - ignore the PC of the branch being predicted (an extreme (n,m) gselect scheme)
gshare: arrange bimodal predictors in single BHT, but construct its index by XORing low-order PC address bits with global branch history shift register - claimed to reduce conflicts
Per-address Two-level Adaptive using Per-address pattern history (PAp): for each branch, keep a k-bit shift register recording its history, and use this to index a BHT for this branch (see Yeh and Patt, 1992)
Each suits some programs well but not all
Advanced Computer Architecture Chapter 4.87


Horses for courses
branch prediction accuracy in percentage
"bimodal"
"gselect"
"gshare"
"local"
"selective"
"static"
"correlation"
100
95
90
85
80
75
gcc gho go li m88num per vor ali dod ear fpp hyd mdd mds nas ora su2 swm tom wav
Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes (http://www.cs.berkeley.edu/~zhendong/cs252/project.html)

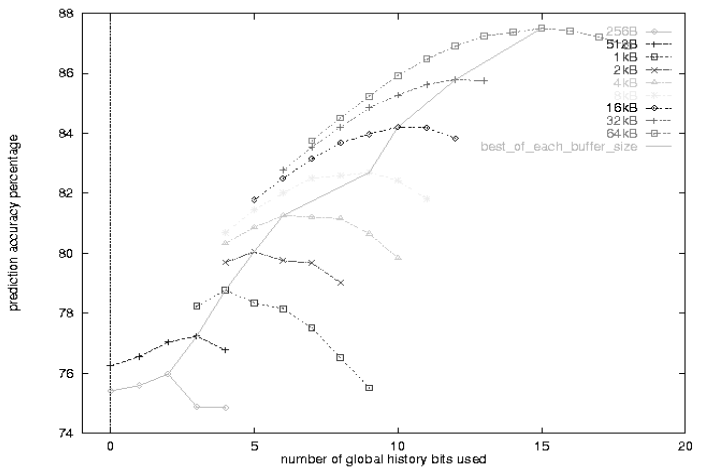
Extreme example - "go"



- "go" is a SPEC95 benchmark code with highly-dynamic, highly-correlated branch behaviour

- The bias of "go"s branches is more-or-less evenly spread between 0% taken and 100% taken
- All known predictors do badly

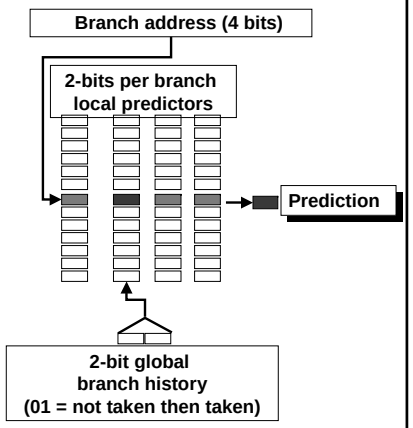
Some dynamic applications have highly-correlated branches



- For "go", optimum BHR size (m) is much larger

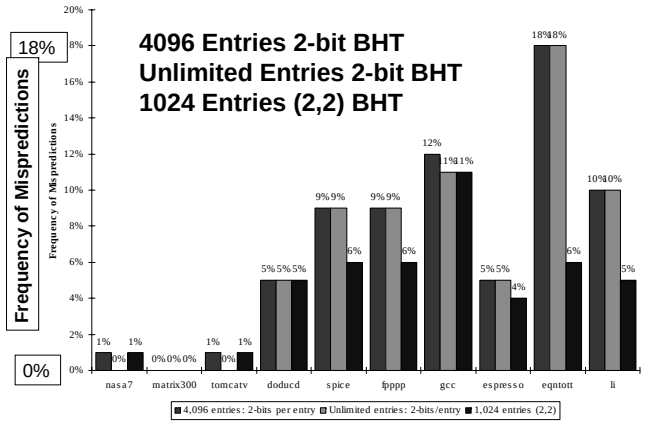
Review: Correlating Branches

- Idea: taken/not taken of recently executed branches is related to behavior of next branch (as well as the history of that branch behavior)
  - Then behavior of recent branches selects between, say, 4 predictions of next branch, updating just that prediction
- (2,2) predictor: 2-bit global, 2-bit local



Accuracy of Different Schemes

(H&P3ed Figure 3.15, p. 206)



Re-evaluating Correlation

- Several of the SPEC benchmarks have less than a dozen branches responsible for 90% of taken branches:

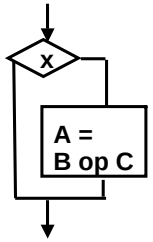
program	branch %	static	# = 90%
compress	14%	236	13
eqntott	25%	494	5
gcc	15%	9531	2020
mpeg	10%	5598	532
real gcc	13%	17361	3214

- Real programs + OS more like gcc
- Small benefits beyond benchmarks for correlation? problems with branch aliases?

Predicated Execution

- Avoid branch prediction by turning branches into conditionally executed instructions:

  if (x) then A = B op C else NOP
  - If false, then neither store result nor cause exception
  - Expanded ISA of Alpha, MIPS, PowerPC, SPARC have conditional move; PA-RISC can annul any following instr.
  - IA-64: 64 1-bit condition fields selected so conditional execution of any instruction
  - This transformation is called "if-conversion"



- Drawbacks to conditional instructions
  - Still takes a clock even if "annulled"
  - Stall if condition evaluated late
  - Complex conditions reduce effectiveness; condition becomes known late in pipeline

BHT Accuracy

- Mispredict because either:
  - Wrong guess for that branch
  - Got branch history of wrong branch when index the table
- 4096 entry table programs vary from 1% misprediction (nasa7, tomcatv) to 18% (eqntott), with spice at 9% and gcc at 12%
- For SPEC92, 4096 about as good as infinite table

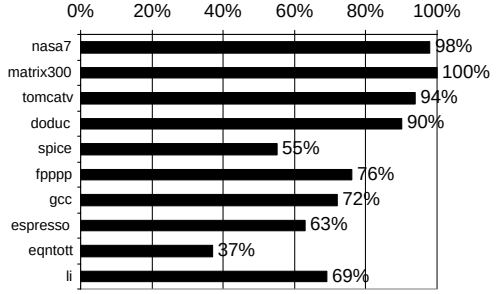
Tournament Predictors

- Motivation for correlating branch predictors is 2-bit predictor failed on important branches; by adding global information, performance improved
- Tournament predictors: use 2 predictors, 1 based on global information and 1 based on local information, and combine with a selector
- Hopes to select right predictor for right branch

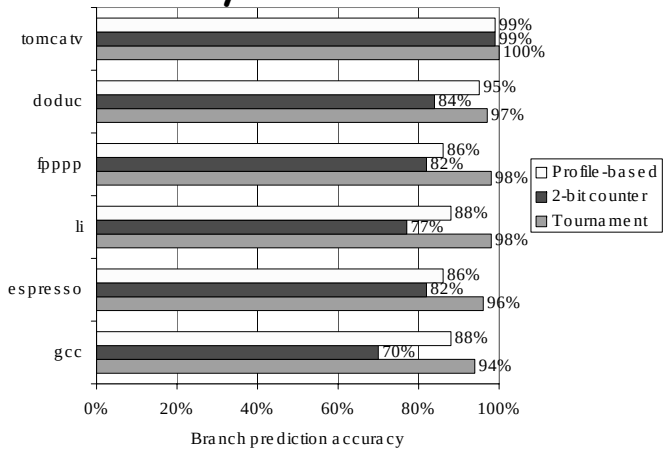
Tournament Predictor in Alpha 21264

- 4K 2-bit counters to choose from among a global predictor and a local predictor
- Global predictor also has 4K entries and is indexed by the history of the last 12 branches; each entry in the global predictor is a standard 2-bit predictor
  - 12-bit pattern: ith bit 0 => ith prior branch not taken; ith bit 1 => ith prior branch taken;
- Local predictor consists of a 2-level predictor:
  - Top level a local history table consisting of 1024 10-bit entries; each 10-bit entry corresponds to the most recent 10 branch outcomes for the entry. 10-bit history allows patterns 10 branches to be discovered and predicted.
  - Next level Selected entry from the local history table is used to index a table of 1K entries consisting a 3-bit saturating counters, which provide the local prediction
- Total size: 4K*2 + 4K*2 + 1K*10 + 1K*3 = 29K bits!
  (~180,000 transistors)

% of predictions from local predictor in Tournament Prediction Scheme

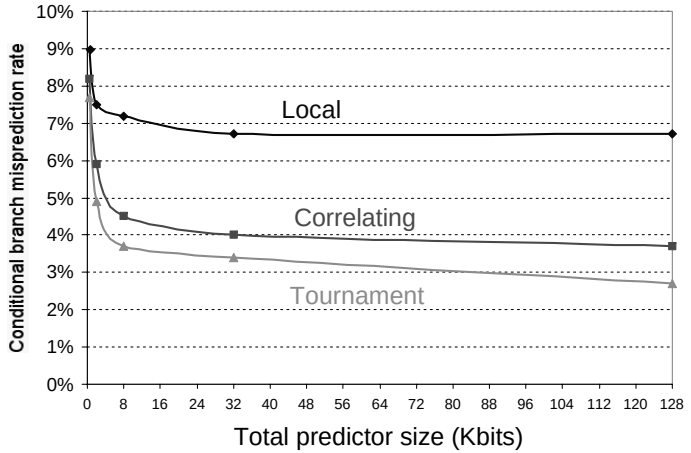


Accuracy of Branch Prediction



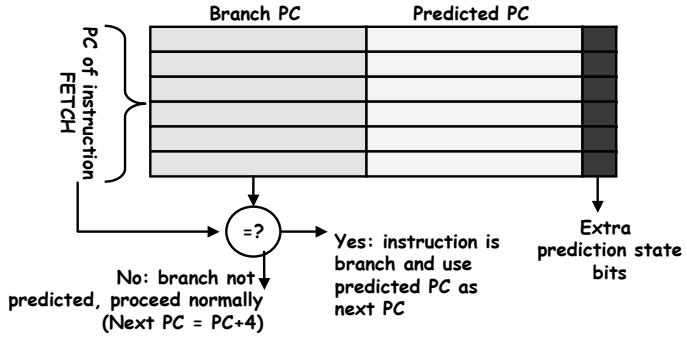
- Profile: branch profile from last execution (static in that in encoded in instruction, but profile)

Accuracy v. Size (SPEC89)



Need Address at Same Time as Prediction

- Branch Target Buffer (BTB): Address of branch index to get prediction AND branch address (if taken)
  - Note: must check for branch match now, since can't use wrong branch address (Figure 3.19, p. 262)



Special Case Return Addresses

- Register Indirect branch hard to predict address
- SPEC89 85% such branches for procedure return
- Since stack discipline for procedures, save return address in small buffer that acts like a stack: 8 to 16 entries has small miss rate

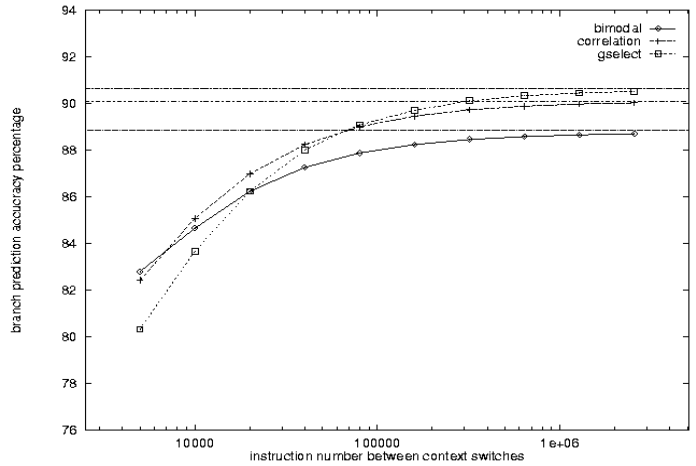
Pitfall: Sometimes bigger and dumber is better

- 21264 uses tournament predictor (29 Kbits)
- Earlier 21164 uses a simple 2-bit predictor with 2K entries (or a total of 4 Kbits)
- SPEC95 benchmarks, 21264 outperforms
  - 21264 avg. 11.5 mispredictions per 1000 instructions
  - 21164 avg. 16.5 mispredictions per 1000 instructions
- Reversed for transaction processing (TP) !
  - 21264 avg. 17 mispredictions per 1000 instructions
  - 21164 avg. 15 mispredictions per 1000 instructions
- TP code much larger & 21164 hold 2X branch predictions based on local behavior (2K vs. 1K local predictor in the 21264)

Warm-up effects and context-switching

- In real life, applications are interrupted and some other program runs for a while (if only the OS)
- This means the branch prediction is regularly trashed
- Simple predictors re-learn fast
  - in 2-bit bimodal predictor, all executions of given branch update same 2 bits
- Sophisticated predictors re-learn more slowly
  - for example, in (2,2) gselect predictor, prediction updates are spread across 4 BHTs
- *Selective* predictor may choose fast learner predictor until better predictor warms up

Warm-up...



Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes, (http://www.cs.berkeley.edu/~zhendong/cs252/project.html)

- Best predictor takes 20,000 instructions to overtake bimodal

Dynamic Branch Prediction Summary

- Prediction becoming important part of scalar execution
- Branch History Table: 2 bits for loop accuracy
  - Saturating counter (bimodal) scheme handles highly-biased branches well
  - Some applications have highly dynamic branches
- Correlation: Recently executed branches correlated with next branch.
  - Either different branches
  - Or different executions of same branches
- Tournament Predictor: more resources to competitive solutions and pick between them
- Branch Target Buffer: include branch address & prediction
- Predicated Execution can reduce number of branches, number of mispredicted branches
- Return address stack for prediction of indirect jump

Getting CPI < 1: Issuing Multiple Instructions/Cycle

- Vector Processing: Explicit coding of independent loops as operations on large vectors of numbers
  - Multimedia instructions being added to many processors
- Superscalar: varying no. instructions/cycle (1 to 8), scheduled by compiler or by HW (Tomasulo)
  - IBM PowerPC, Sun UltraSparc, DEC Alpha, Pentium III/4
- (Very) Long Instruction Words (V)LIW: fixed number of instructions (4-16) scheduled by the compiler; put ops into wide templates (TBD)
  - Intel Architecture-64 (IA-64) 64-bit address
    - Renamed: "Explicitly Parallel Instruction Computer (EPIC)"
  - Will discuss shortly
- Anticipated success of multiple instructions lead to Instructions Per Clock_cycle (IPC) vs. CPI

Getting CPI < 1: Issuing Multiple Instructions/Cycle

- Superscalar MIPS: 2 instructions, 1 FP & 1 anything
  - Fetch 64-bits/clock cycle; Int on left, FP on right
  - Can only issue 2nd instruction if 1st instruction issues
  - More ports for FP registers to do FP load & FP op in a pair

Type	*Pipe Stages*					
Int. instruction	IF	ID	EX	MEM	WB	
FP instruction	IF	ID	EX	MEM	WB	
Int. instruction		IF	ID	EX	MEM	WB
FP instruction		IF	ID	EX	MEM	WB
Int. instruction			IF	ID	EX	MEM	WB
FP instruction			IF	ID	EX	MEM	WB

- 1 cycle load delay expands to 3 instructions in SS
  - instruction in right half can't use it, nor instructions in next slot

Multiple Issue Issues

- issue packet: group of instructions from fetch unit that could potentially issue in 1 clock
  - If instruction causes structural hazard or a data hazard either due to earlier instruction in execution or to earlier instruction in issue packet, then instruction does not issue
  - 0 to N instruction issues per clock cycle, for N-issue
- Performing issue checks in 1 cycle could limit clock cycle time: $O(n^2-n)$ comparisons
  - => issue stage usually split and pipelined
  - 1st stage decides how many instructions from within this packet can issue, 2nd stage examines hazards among selected instructions and those already been issued
  - => higher branch penalties => prediction accuracy important

Multiple Issue Challenges

- While Integer/FP split is simple for the HW, get CPI of 0.5 only for programs with:
  - Exactly 50% FP operations AND No hazards
- If more instructions issue at same time, greater difficulty of decode and issue:
  - Even 2-scalar => examine 2 opcodes, 6 register specifiers, & decide if 1 or 2 instructions can issue; (N-issue ~$O(N^2-N)$ comparisons)
  - Register file: need 2x reads and 1x writes/cycle
  - Rename logic: must be able to rename same register multiple times in one cycle! For instance, consider 4-way issue:

```
add r1, r2, r3          add p11, p4, p7
sub r4, r1, r2    ⇒     sub p22, p11, p4
lw  r1, 4(r4)           lw  p23, 4(p22)
add r5, r1, r2          add p12, p23, p4
```

    Imagine doing this transformation in a single cycle!
  - Result buses: Need to complete multiple instructions/cycle
    - So, need multiple buses with associated matching logic at every reservation station.
    - Or, need multiple forwarding paths

Dynamic Scheduling in Superscalar The easy way

- How to issue two instructions and keep in-order instruction issue for Tomasulo?
  - Assume 1 integer + 1 floating point
  - 1 Tomasulo control for integer, 1 for floating point
- Issue 2X Clock Rate, so that issue remains in order
- Only loads/stores 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

Register renaming, virtual registers versus Reorder Buffers

- 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)

How much to speculate?

- Speculation Pro: uncover events that would otherwise stall the pipeline (cache misses)
- Speculation Con: speculate costly if exceptional event occurs when speculation was incorrect
- Typical solution: speculation allows only low-cost exceptional events (1st-level cache miss)
- When expensive exceptional event occurs, (2nd-level cache miss or TLB miss) processor waits until the instruction causing event is no longer speculative before handling the event
- Assuming single branch per cycle: future may speculate across multiple branches!

Limits to ILP

- Conflicting studies of amount
  - Benchmarks (vectorized Fortran FP vs. integer C programs)
  - Hardware sophistication
  - Compiler sophistication
- How much ILP is available using existing mechanisms with increasing HW budgets?
- Do we need to invent new HW/SW mechanisms to keep on processor performance curve?
  - Intel MMX, SSE (Streaming SIMD Extensions): 64 bit ints
  - Intel SSE2: 128 bit, including 2 64-bit Fl. Pt. per clock
  - Motorola AltiVec: 128 bit ints and FPs
  - Supersparc Multimedia ops, etc.

Limits to ILP

Initial HW Model here; MIPS compilers.

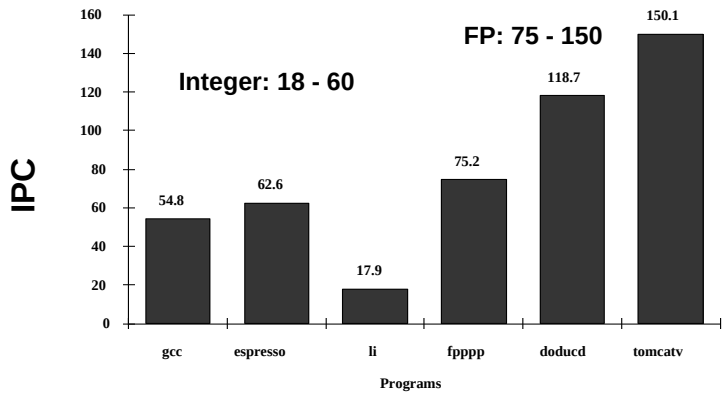
Assumptions for ideal/perfect machine to start:

1. *Register renaming* – infinite virtual registers => all register WAW & WAR hazards are avoided
2. *Branch prediction* – perfect; no mispredictions
3. *Jump prediction* – all jumps perfectly predicted
   2 & 3 => machine with perfect speculation & an unbounded buffer of instructions available
4. *Memory-address alias analysis* – addresses are known & a store can be moved before a load provided addresses not equal

Also:
unlimited number of instructions issued/clock cycle;
perfect caches;
1 cycle latency for all instructions (FP *,/);

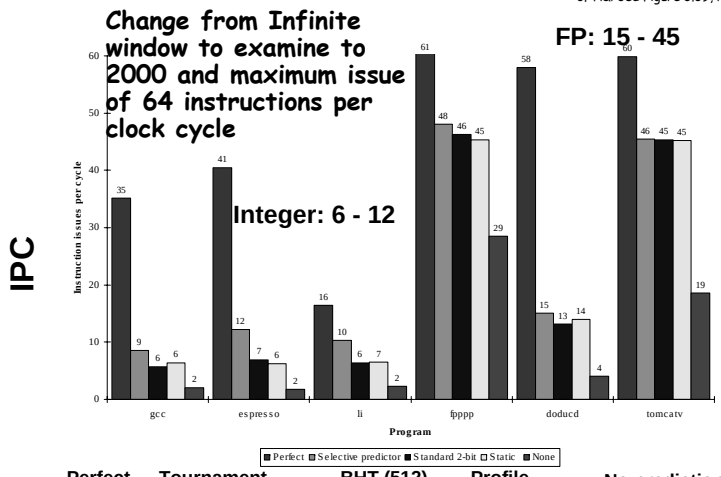
Upper Limit to ILP: Ideal Machine

(H&P3ed Figure 3.35, page 242)



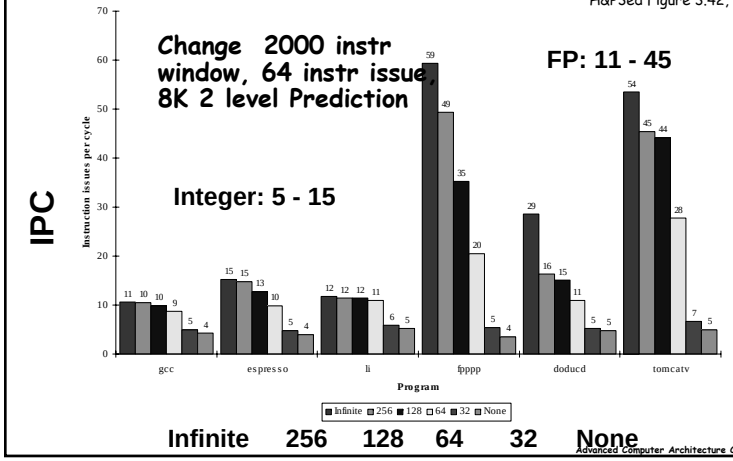
More Realistic HW: Branch Impact

cf H&P3ed Figure 3.39, Page 248



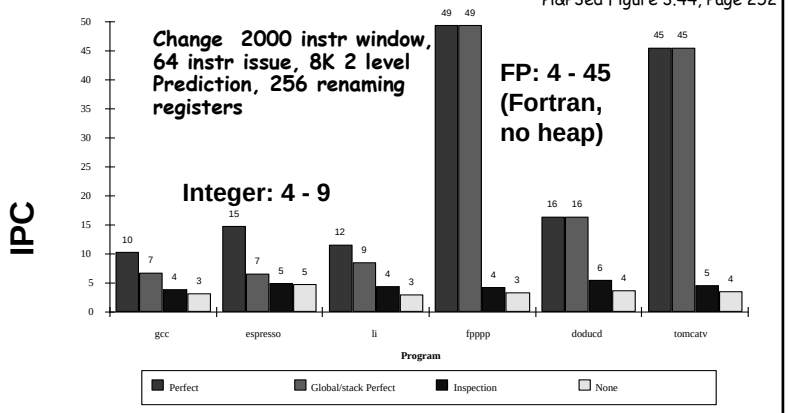
More Realistic HW: Renaming Register Impact

H&P3ed Figure 3.42, Page 251



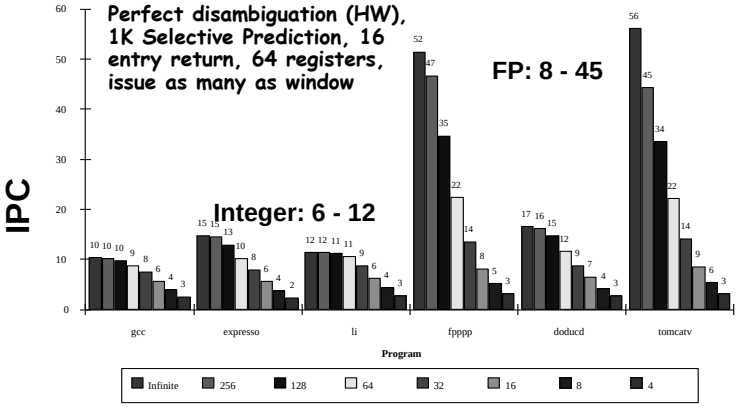
More Realistic HW: Memory Address Alias Impact

H&P3ed Figure 3.44, Page 252



Realistic HW for '00: Window Impact

(Figure 3.45, Page 309)



How to Exceed ILP Limits of this study?

- WAR and WAW hazards through memory: eliminated WAW and WAR hazards through register renaming, but not in memory usage
- Unnecessary dependences (compiler not unrolling loops so iteration variable dependence)
- Overcoming the data flow limit: value prediction, predicting values and speculating on prediction
  - Address value prediction and speculation predicts addresses and speculates by reordering loads and stores; could provide better aliasing analysis, only need predict if addresses =

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	32I / 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

SPEC 2000 Performance 3/2001 *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	None	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	568	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	371	427	252	189	340
SPECfp_base2000	590	304	400	356	329	549	319	274	427

1.5X 1.8X 1.2X 1.6X 1.7X

Conclusion

- 1985-2000: 1000X performance
  - Moore's Law transistors/chip => Moore's Law for Performance/MPU
- "industry been following a roadmap of ideas known in 1985 to exploit Instruction Level Parallelism and (real) Moore's Law to get 1.55X/year"
  - Caches, Pipelining, Superscalar, Branch Prediction, Out-of-order execution, …
- ILP limits: To make performance progress in future need to have explicit parallelism from programmer vs. implicit parallelism of ILP exploited by compiler, HW?
  - Otherwise drop to old rate of 1.3X per year?
  - Less than 1.3X because of processor-memory performance gap?
- Impact on you: if you care about performance, better think about explicitly parallel algorithms vs. rely on ILP?