

332
Advanced Computer Architecture
Chapter 3

Instruction Level Parallelism and Dynamic Execution

February 2006
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 Kubiawicz's Berkeley course (*CS252*)

Advanced Computer Architecture Chapter 3.1

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)

Advanced Computer Architecture Chapter 3.2

Ideas to Reduce Stalls

Textbook
Chapter 3

Textbook
Chapter 4

Technique	Reduces
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
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

Advanced Computer Architecture Chapter 3.3

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

Advanced Computer Architecture Chapter 3.4

Data Dependence and Hazards

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

```

    I: add r1,r2,r3
    J: sub r4,r1,r3
  
```

- ↳ 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**

Advanced Computer Architecture Chapter 3.5

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

Advanced Computer Architecture Chapter 3.6

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

```

    I: sub r4,r1,r3
    J: add r1,r2,r3
    K: mul r6,r1,r7
  
```

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**

Advanced Computer Architecture Chapter 3.7

Name Dependence #2: Output dependence

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

```

    I: sub r1,r4,r3
    J: add r1,r2,r3
    K: mul r6,r1,r7
  
```

- ↳ 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**

Advanced Computer Architecture Chapter 3.8

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

Advanced Computer Architecture Chapter 3.9

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.

Advanced Computer Architecture Chapter 3.10

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**

Advanced Computer Architecture Chapter 3.11

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?

Advanced Computer Architecture Chapter 3.12

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

Advanced Computer Architecture Chapter 3.13

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

Advanced Computer Architecture Chapter 3.14

HW Schemes: Instruction Parallelism

• **Key idea:** Allow instructions behind stall to proceed

```
DIVD    F0,F2,F4
ADD     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**)

Advanced Computer Architecture Chapter 3.15

Dynamic Scheduling Step 1

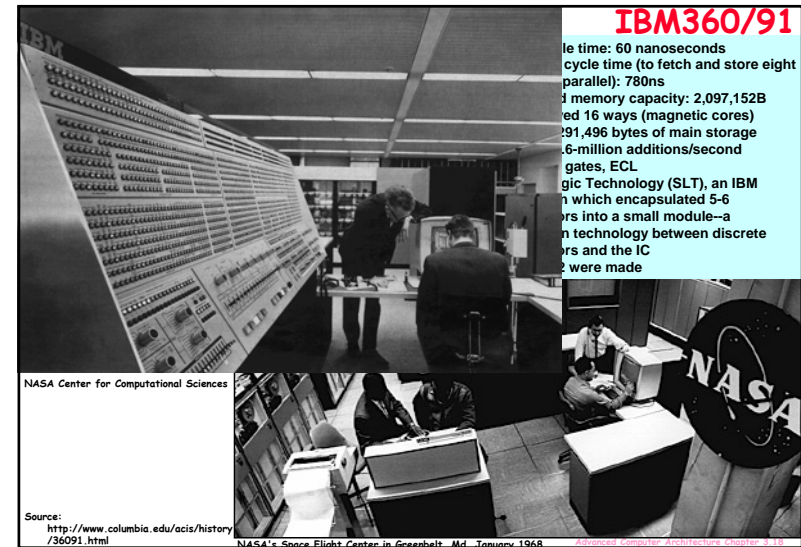
- Simple pipeline had 1 stage to check both structural and data hazards: Instruction Decode (ID), also called Instruction Issue
- Split the ID pipe stage of simple 5-stage pipeline into 2 stages:
 - **Issue**—Decode instructions, check for structural hazards
 - **Read operands**—Wait until no data hazards,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

Advanced Computer Architecture Chapter 3.16

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 descendants of this have flourished!
 - Alpha 21264, HP 8000, MIPS 10000/R12000, Pentium II/III/4, AMD K5,K6,Athlon, PowerPC 603/604/63/64/65, ...

Advanced Computer Architecture Chapter 3.17

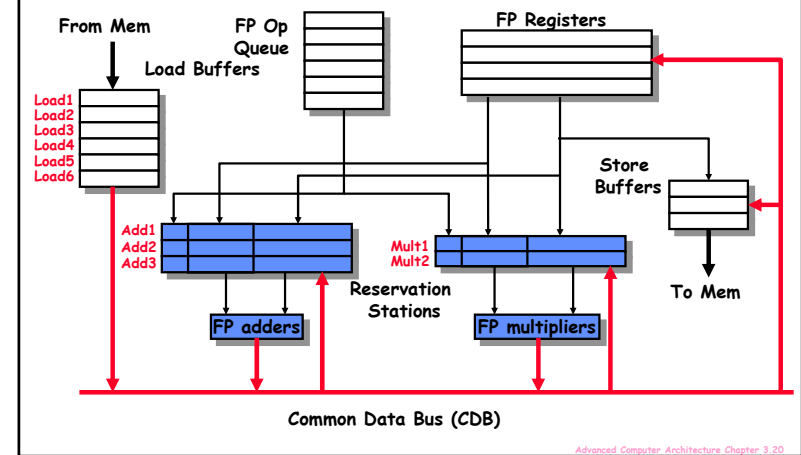


Tomasulo Algorithm

- 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

Advanced Computer Architecture Chapter 3.19

Tomasulo Organization



Advanced Computer Architecture Chapter 3.20

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.

Advanced Computer Architecture Chapter 3.21

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 FI .pt. +, -; 10 for * ; 40 clks for /

Advanced Computer Architecture Chapter 3.22

► The IBM 360/91's pipeline:

360/91 pipeline

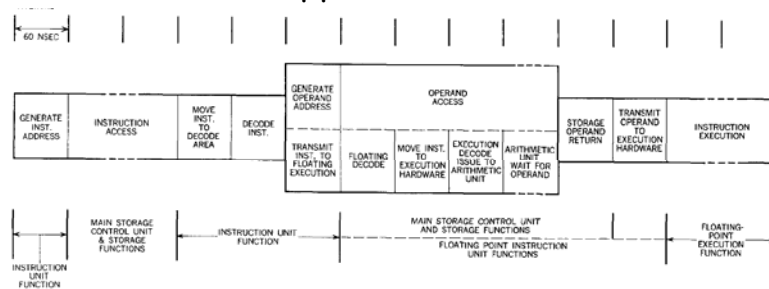


Figure 3 CPU "assembly-line stations required to accommodate a typical floating-point storage-to-register instruction.

► 11-12 circuit levels per pipeline stage, of 5-6ns each

► CPU consists of three physical frames, each having dimensions 66" L X 15" D X 78" H

See: The IBM System/360 Model 91: Machine Philosophy and Instruction-Handling, by D. W. Anderson, F. J. Sparacio, R. M. Tomasulo. IBM J. R&D (1967).
<http://www.research.ibm.com/journal/rd/111/ibmrd1101C.pdf>

Advanced Computer Architecture Chapter 3.23

Tomasulo Example

Instruction stream

Instruction status:

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

Load	Busy	Address
Load1	No	
Load2	No	
Load3	No	

3 Load/Buffers

Reservation Stations:

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

Register result status:

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
0									

Clock cycle counter

Advanced Computer Architecture Chapter 3.24

Tomasulo Example Cycle 1

Instruction status:

Instruction	j	k	Issue	Exec	Write	Comp	Result	Busy	Address
LD	F6	34+	R2	1				Yes	34+R2
LD	F2	45+	R3					No	
MULTD	F0	F2	F4					No	
SUBD	F8	F6	F2					No	
DIVD	F10	F0	F6					No	
ADDD	F6	F8	F2					No	

Reservation Stations:

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

Register result status:

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

Advanced Computer Architecture Chapter 3.25

Tomasulo Example Cycle 2

Instruction status:

Instruction	j	k	Issue	Exec	Write	Comp	Result	Busy	Address
LD	F6	34+	R2	1				Yes	34+R2
LD	F2	45+	R3	2				Yes	45+R3
MULTD	F0	F2	F4					No	
SUBD	F8	F6	F2					No	
DIVD	F10	F0	F6					No	
ADDD	F6	F8	F2					No	

Reservation Stations:

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

Register result status:

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

Note: Can have multiple loads outstanding

Advanced Computer Architecture Chapter 3.26

Tomasulo Example Cycle 3

Instruction status:

Instruction	j	k	Issue	Exec	Write	Comp	Result	Busy	Address
LD	F6	34+	R2	1		3		Yes	34+R2
LD	F2	45+	R3	2				Yes	45+R3
MULTD	F0	F2	F4	3				No	
SUBD	F8	F6	F2					No	
DIVD	F10	F0	F6					No	
ADDD	F6	F8	F2					No	

Reservation Stations:

Time	Name	Busy	Op	S1	S2	RS	RS
				Vj	Vk	Qj	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?

Advanced Computer Architecture Chapter 3.27

Tomasulo Example Cycle 4

Instruction status:

Instruction	j	k	Issue	Exec	Write	Comp	Result	Busy	Address
LD	F6	34+	R2	1		3		No	
LD	F2	45+	R3	2		4		Yes	45+R3
MULTD	F0	F2	F4	3				No	
SUBD	F8	F6	F2	4				No	
DIVD	F10	F0	F6					No	
ADDD	F6	F8	F2					No	

Reservation Stations:

Time	Name	Busy	Op	S1	S2	RS	RS
				Vj	Vk	Qj	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?

Advanced Computer Architecture Chapter 3.28

Tomasulo Example Cycle 5

Instruction status:

Instruction	j	k	Issue	Comp	Result	Busy	Address
LD	F6	34+	R2	1	3	4	Load1
LD	F2	45+	R3	2	4	5	Load2
MULTD	F0	F2	F4	3			Load3
SUBD	F8	F6	F2	4			
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	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

Advanced Computer Architecture Chapter 3.25

Tomasulo Example Cycle 6

Instruction status:

Instruction	j	k	Issue	Comp	Result	Busy	Address
LD	F6	34+	R2	1	3	4	Load1
LD	F2	45+	R3	2	4	5	Load2
MULTD	F0	F2	F4	3			Load3
SUBD	F8	F6	F2	4			
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	6			

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	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?

Advanced Computer Architecture Chapter 3.26

Tomasulo Example Cycle 7

Instruction status:

Instruction	j	k	Issue	Comp	Result	Busy	Address
LD	F6	34+	R2	1	3	4	Load1
LD	F2	45+	R3	2	4	5	Load2
MULTD	F0	F2	F4	3			Load3
SUBD	F8	F6	F2	4		7	
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	6			

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	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?

Advanced Computer Architecture Chapter 3.31

Tomasulo Example Cycle 8

Instruction status:

Instruction	j	k	Issue	Comp	Result	Busy	Address
LD	F6	34+	R2	1	3	4	Load1
LD	F2	45+	R3	2	4	5	Load2
MULTD	F0	F2	F4	3			Load3
SUBD	F8	F6	F2	4	7	8	
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	6			

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	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		

Advanced Computer Architecture Chapter 3.32

Tomasulo Example Cycle 9

Instruction status:

Instruction	j	k	Issue	Exec	Write	Busy	Address
LD	F6	34+	R2	1	3	4	Load1
LD	F2	45+	R3	2	4	5	Load2
MULTD	F0	F2	F4	3			Load3
SUBD	F8	F6	F2	4	7	8	
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	6			

Reservation Stations:

Time	Name	Busy	Op	S1	S2	RS	RS
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		

Advanced Computer Architecture Chapter 3.33

Tomasulo Example Cycle 10

Instruction status:

Instruction	j	k	Issue	Exec	Write	Busy	Address
LD	F6	34+	R2	1	3	4	Load1
LD	F2	45+	R3	2	4	5	Load2
MULTD	F0	F2	F4	3			Load3
SUBD	F8	F6	F2	4	7	8	
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	6	10		

Reservation Stations:

Time	Name	Busy	Op	S1	S2	RS	RS
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?

Advanced Computer Architecture Chapter 3.34

Tomasulo Example Cycle 11

Instruction status:

Instruction	j	k	Issue	Exec	Write	Busy	Address
LD	F6	34+	R2	1	3	4	Load1
LD	F2	45+	R3	2	4	5	Load2
MULTD	F0	F2	F4	3			Load3
SUBD	F8	F6	F2	4	7	8	
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	6	10	11	

Reservation Stations:

Time	Name	Busy	Op	S1	S2	RS	RS
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!

Advanced Computer Architecture Chapter 3.35

Tomasulo Example Cycle 12

Instruction status:

Instruction	j	k	Issue	Exec	Write	Busy	Address
LD	F6	34+	R2	1	3	4	Load1
LD	F2	45+	R3	2	4	5	Load2
MULTD	F0	F2	F4	3			Load3
SUBD	F8	F6	F2	4	7	8	
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	6	10	11	

Reservation Stations:

Time	Name	Busy	Op	S1	S2	RS	RS
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			

Advanced Computer Architecture Chapter 3.36

Tomasulo Example Cycle 13

Instruction status:

Instruction	j	k	Issue	Exec	Write	Busy	Address
LD	F6	34+	R2	1	3	4	Load1
LD	F2	45+	R3	2	4	5	Load2
MULTD	F0	F2	F4	3			Load3
SUBD	F8	F6	F2	4	7	8	
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	6	10	11	

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	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+N)	(M-M)	Mult2			

Advanced Computer Architecture Chapter 3.37

Tomasulo Example Cycle 14

Instruction status:

Instruction	j	k	Issue	Exec	Write	Busy	Address
LD	F6	34+	R2	1	3	4	Load1
LD	F2	45+	R3	2	4	5	Load2
MULTD	F0	F2	F4	3			Load3
SUBD	F8	F6	F2	4	7	8	
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	6	10	11	

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	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+N)	(M-M)	Mult2			

Advanced Computer Architecture Chapter 3.38

Tomasulo Example Cycle 15

Instruction status:

Instruction	j	k	Issue	Exec	Write	Busy	Address
LD	F6	34+	R2	1	3	4	Load1
LD	F2	45+	R3	2	4	5	Load2
MULTD	F0	F2	F4	3	15		Load3
SUBD	F8	F6	F2	4	7	8	
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	6	10	11	

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	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+N)	(M-M)	Mult2			

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

Advanced Computer Architecture Chapter 3.39

Tomasulo Example Cycle 16

Instruction status:

Instruction	j	k	Issue	Exec	Write	Busy	Address
LD	F6	34+	R2	1	3	4	Load1
LD	F2	45+	R3	2	4	5	Load2
MULTD	F0	F2	F4	3	15	16	Load3
SUBD	F8	F6	F2	4	7	8	
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	6	10	11	

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	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+N)	(M-M)	Mult2			

- Just waiting for Mult2 (DIVD) to complete

Advanced Computer Architecture Chapter 3.40

Skip a few cycles: Cycle 55

Instruction status:

Instruction	j	k	Issue	Exec	Write	Busy	Address
LD	F6	34+	R2	1	3	4	Load1
LD	F2	45+	R3	2	4	5	Load2
MULTD	F0	F2	F4	3	15	16	Load3
SUBD	F8	F6	F2	4	7	8	
DIVD	F10	F0	F6	5	56	57	
ADDD	F6	F8	F2	6	10	11	

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	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	M*F4	M(A2)		(M-M+N)	(M-M)	Mult2			

Advanced Computer Architecture Chapter 3.41

Tomasulo Example Cycle 56

Instruction status:

Instruction	j	k	Issue	Exec	Write	Busy	Address
LD	F6	34+	R2	1	3	4	Load1
LD	F2	45+	R3	2	4	5	Load2
MULTD	F0	F2	F4	3	15	16	Load3
SUBD	F8	F6	F2	4	7	8	
DIVD	F10	F0	F6	5	56	57	
ADDD	F6	F8	F2	6	10	11	

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	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	M*F4	M(A2)		(M-M+N)	(M-M)	Mult2			

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

Advanced Computer Architecture Chapter 3.42

Tomasulo Example Cycle 57

Instruction status:

Instruction	j	k	Issue	Exec	Write	Busy	Address
LD	F6	34+	R2	1	3	4	Load1
LD	F2	45+	R3	2	4	5	Load2
MULTD	F0	F2	F4	3	15	16	Load3
SUBD	F8	F6	F2	4	7	8	
DIVD	F10	F0	F6	5	56	57	
ADDD	F6	F8	F2	6	10	11	

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	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	M*F4	M(A2)		(M-M+N)	(M-M)	Result			

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

Advanced Computer Architecture Chapter 3.43

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

Advanced Computer Architecture Chapter 3.44

Tomasulo Loop Example

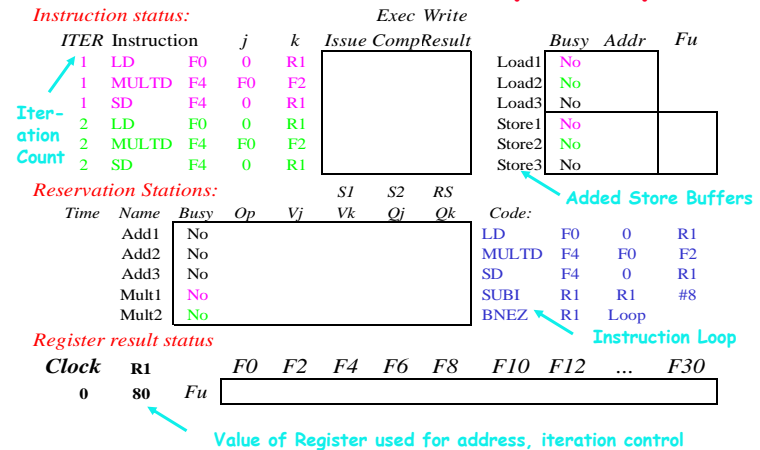
```

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

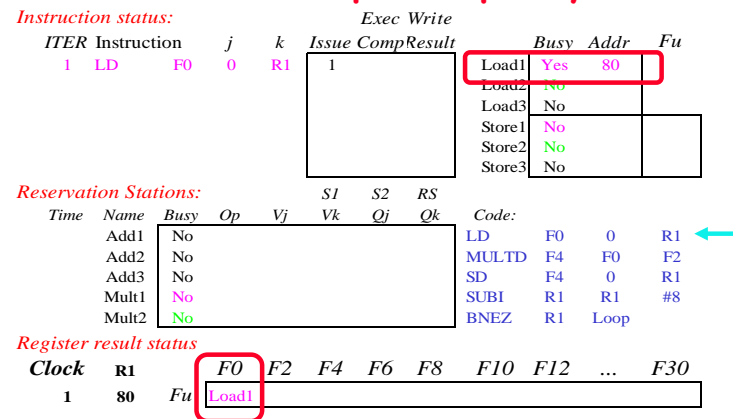
Advanced Computer Architecture Chapter 3.45

Loop Example



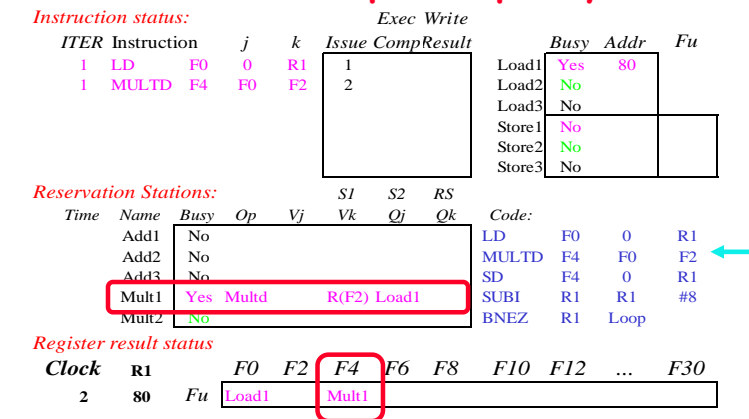
Advanced Computer Architecture Chapter 3.46

Loop Example Cycle 1



Advanced Computer Architecture Chapter 3.47

Loop Example Cycle 2



Advanced Computer Architecture Chapter 3.48

Loop Example Cycle 3

Instruction status:

ITER	Instruction	j	k	Issue	CompResult	Busy	Addr	Fu
1	LD	F0	0	R1	1	Yes	80	
1	MULTD	F4	F0	F2	2	No		
1	SD	F4	0	R1	3	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						SUBI R1 R1 #8
Mult2	No							BNEZ R1 Loop

Register result status

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

Implicit renaming sets up data flow graph

Loop Example Cycle 4

Instruction status:

ITER	Instruction	j	k	Issue	CompResult	Busy	Addr	Fu
1	LD	F0	0	R1	1	Yes	80	
1	MULTD	F4	F0	F2	2	No		
1	SD	F4	0	R1	3	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						SUBI R1 R1 #8
Mult2	No							BNEZ R1 Loop

Register result status

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

Dispatching SUBI Instruction (not in FP queue)

Loop Example Cycle 5

Instruction status:

ITER	Instruction	j	k	Issue	CompResult	Busy	Addr	Fu
1	LD	F0	0	R1	1	Yes	80	
1	MULTD	F4	F0	F2	2	No		
1	SD	F4	0	R1	3	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						SUBI R1 R1 #8
Mult2	No							BNEZ R1 Loop

Register result status

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

And, BNEZ instruction (not in FP queue)

Loop Example Cycle 6

Instruction status:

ITER	Instruction	j	k	Issue	CompResult	Busy	Addr	Fu
1	LD	F0	0	R1	1	Yes	80	
1	MULTD	F4	F0	F2	2	Yes	72	
1	SD	F4	0	R1	3	No		
2	LD	F0	0	R1	6	Yes	80	Mult1
					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						SUBI R1 R1 #8
Mult2	No							BNEZ R1 Loop

Register result status

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

Notice that F0 never sees Load from location 80

Loop Example Cycle 7

Instruction status:

ITER	Instruction	j	k	Issue	CompResult	Busy	Addr	Fu
1	LD F0 0 R1	0	R1	1		Yes	80	
1	MULTD F4 F0 F2	F0	F2	2		Yes	72	
1	SD F4 0 R1	0	R1	3		No		
2	LD F0 0 R1	0	R1	6		Yes	80	Mult1
2	MULTD F4 F0 F2	F0	F2	7		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	Yes	Multd		R(F2)	Load2			BNEZ R1 Loop

Register result status

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

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

Advanced Computer Architecture Chapter 3.53

Loop Example Cycle 8

Instruction status:

ITER	Instruction	j	k	Issue	CompResult	Busy	Addr	Fu
1	LD F0 0 R1	0	R1	1		Yes	80	
1	MULTD F4 F0 F2	F0	F2	2		Yes	72	
1	SD F4 0 R1	0	R1	3		No		
2	LD F0 0 R1	0	R1	6		Yes	80	Mult1
2	MULTD F4 F0 F2	F0	F2	7		Yes	72	Mult2
2	SD F4 0 R1	0	R1	8		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	Yes	Multd		R(F2)	Load2			BNEZ R1 Loop

Register result status

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

Advanced Computer Architecture Chapter 3.54

Loop Example Cycle 9

Instruction status:

ITER	Instruction	j	k	Issue	CompResult	Busy	Addr	Fu
1	LD F0 0 R1	0	R1	1	9	Yes	80	
1	MULTD F4 F0 F2	F0	F2	2		Yes	72	
1	SD F4 0 R1	0	R1	3		No		
2	LD F0 0 R1	0	R1	6		Yes	80	Mult1
2	MULTD F4 F0 F2	F0	F2	7		Yes	72	Mult2
2	SD F4 0 R1	0	R1	8		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	Yes	Multd		R(F2)	Load2			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

Advanced Computer Architecture Chapter 3.55

Loop Example Cycle 10

Instruction status:

ITER	Instruction	j	k	Issue	CompResult	Busy	Addr	Fu
1	LD F0 0 R1	0	R1	1	9	No		
1	MULTD F4 F0 F2	F0	F2	2		Yes	72	
1	SD F4 0 R1	0	R1	3		No		
2	LD F0 0 R1	0	R1	6	10	Yes	80	Mult1
2	MULTD F4 F0 F2	F0	F2	7		Yes	72	Mult2
2	SD F4 0 R1	0	R1	8		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
4	Mult1	Yes	Multd	M[80]	R(F2)			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
10	64	Fu	Load2	Mult2						

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

Advanced Computer Architecture Chapter 3.56

Loop Example Cycle 11

Instruction status:

ITER	Instruction	j	k	Issue	CompResult	Busy	Addr	Fu
1	LD F0 0 R1	0	R1	1	9 10	No		
1	MULTD F4 F0 F2	F0	F2	2		No		
1	SD F4 0 R1	0	R1	3		Yes	64	
2	LD F0 0 R1	0	R1	6	10	Yes	80	Mult1
2	MULTD F4 F0 F2	F0	F2	7		Yes	72	Mult2
2	SD F4 0 R1	0	R1	8		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
3	Mult1	Yes	Multd	M[80]	R(F2)			SUBI R1 R1 #8
4	Mult2	Yes	Multd	M[72]	R(F2)			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	CompResult	Busy	Addr	Fu
1	LD F0 0 R1	0	R1	1	9 10	No		
1	MULTD F4 F0 F2	F0	F2	2		No		
1	SD F4 0 R1	0	R1	3		Yes	64	
2	LD F0 0 R1	0	R1	6	10 11	Yes	80	Mult1
2	MULTD F4 F0 F2	F0	F2	7		Yes	72	Mult2
2	SD F4 0 R1	0	R1	8		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
2	Mult1	Yes	Multd	M[80]	R(F2)			SUBI R1 R1 #8
3	Mult2	Yes	Multd	M[72]	R(F2)			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	CompResult	Busy	Addr	Fu
1	LD F0 0 R1	0	R1	1	9 10	No		
1	MULTD F4 F0 F2	F0	F2	2		No		
1	SD F4 0 R1	0	R1	3		Yes	64	
2	LD F0 0 R1	0	R1	6	10 11	Yes	80	Mult1
2	MULTD F4 F0 F2	F0	F2	7		Yes	72	Mult2
2	SD F4 0 R1	0	R1	8		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
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	CompResult	Busy	Addr	Fu
1	LD F0 0 R1	0	R1	1	9 10	No		
1	MULTD F4 F0 F2	F0	F2	2	14	No		
1	SD F4 0 R1	0	R1	3		Yes	64	
2	LD F0 0 R1	0	R1	6	10 11	Yes	80	Mult1
2	MULTD F4 F0 F2	F0	F2	7		Yes	72	Mult2
2	SD F4 0 R1	0	R1	8		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
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	Comp	Result	Busy	Addr	Fu
1	LD	F0	0	R1	1	9	No		
1	MULTD	F4	F0	F2	2	14	No		
1	SD	F4	0	R1	3		Yes	64	
2	LD	F0	0	R1	6	10	Yes	80	[80]*R2
2	MULTD	F4	F0	F2	7	15	Yes	72	Mult2
2	SD	F4	0	R1	8		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
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?

Advanced Computer Architecture Chapter 3.61

Loop Example Cycle 16

Instruction status:

ITER	Instruction	j	k	Issue	Comp	Result	Busy	Addr	Fu
1	LD	F0	0	R1	1	9	No		
1	MULTD	F4	F0	F2	2	14	No		
1	SD	F4	0	R1	3		Yes	64	
2	LD	F0	0	R1	6	10	Yes	80	[80]*R2
2	MULTD	F4	F0	F2	7	15	Yes	72	[72]*R2
2	SD	F4	0	R1	8		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
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						

Advanced Computer Architecture Chapter 3.62

Loop Example Cycle 17

Instruction status:

ITER	Instruction	j	k	Issue	Comp	Result	Busy	Addr	Fu
1	LD	F0	0	R1	1	9	No		
1	MULTD	F4	F0	F2	2	14	No		
1	SD	F4	0	R1	3		Yes	64	
2	LD	F0	0	R1	6	10	Yes	80	[80]*R2
2	MULTD	F4	F0	F2	7	15	Yes	72	[72]*R2
2	SD	F4	0	R1	8		Yes	64	Mult1

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)	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						

Advanced Computer Architecture Chapter 3.63

Loop Example Cycle 18

Instruction status:

ITER	Instruction	j	k	Issue	Comp	Result	Busy	Addr	Fu
1	LD	F0	0	R1	1	9	No		
1	MULTD	F4	F0	F2	2	14	No		
1	SD	F4	0	R1	3		Yes	64	
2	LD	F0	0	R1	6	10	Yes	80	[80]*R2
2	MULTD	F4	F0	F2	7	15	Yes	72	[72]*R2
2	SD	F4	0	R1	8		Yes	64	Mult1

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)	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						

Advanced Computer Architecture Chapter 3.64

Loop Example Cycle 19

Instruction status:

ITER	Instruction	j	k	Issue	Comp	Result	Busy	Addr	Fu
1	LD F0 0 R1	1	9	10	Load1	No			
1	MULTD F4 F0 F2	2	14	15	Load2	No			
1	SD F4 0 R1	3	18	19	Load3	Yes	64		
2	LD F0 0 R1	6	10	11	Store1	No			
2	MULTD F4 F0 F2	7	15	16	Store2	Yes	72	[72]*R2	
2	SD F4 0 R1	8	19		Store3	Yes	64	Multi	

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
Multi1	Yes	Multi		R(F2)	Load3			SUBI R1 R1 #8
Multi2	No							BNEZ R1 Loop

Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
19	56	Fu	Load3	Multi1						

Advanced Computer Architecture Chapter 3.65

Loop Example Cycle 20

Instruction status:

ITER	Instruction	j	k	Issue	Comp	Result	Busy	Addr	Fu
1	LD F0 0 R1	1	9	10	Load1	Yes	56		
1	MULTD F4 F0 F2	2	14	15	Load2	No			
1	SD F4 0 R1	3	18	19	Load3	Yes	64		
2	LD F0 0 R1	6	10	11	Store1	No			
2	MULTD F4 F0 F2	7	15	16	Store2	No			
2	SD F4 0 R1	8	19	20	Store3	Yes	64	Multi	

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
Multi1	Yes	Multi		R(F2)	Load3			SUBI R1 R1 #8
Multi2	No							BNEZ R1 Loop

Register result status

Clock	R1	F0	F2	F4	F6	F8	F10	F12	...	F30
20	56	Fu	Load1	Multi1						

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

Advanced Computer Architecture Chapter 3.66

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.

Advanced Computer Architecture Chapter 3.67

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

Advanced Computer Architecture Chapter 3.68

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.

Advanced Computer Architecture Chapter 3.65

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!

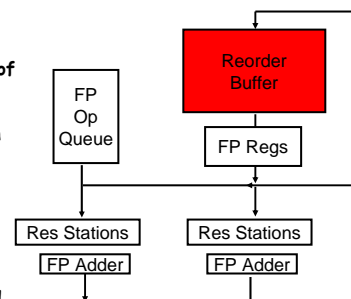
• Technique for both precise interrupts/exceptions and speculation: *in-order completion or commit*

Advanced Computer Architecture Chapter 3.70

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



Advanced Computer Architecture Chapter 3.71

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”)

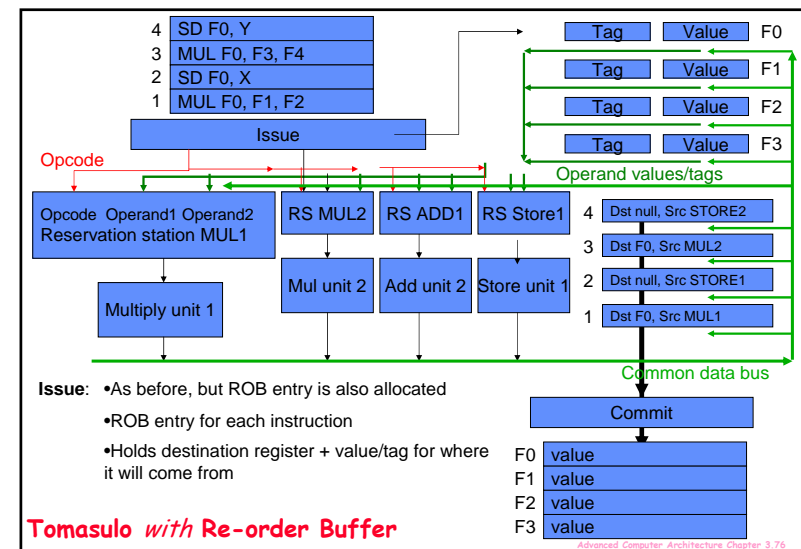
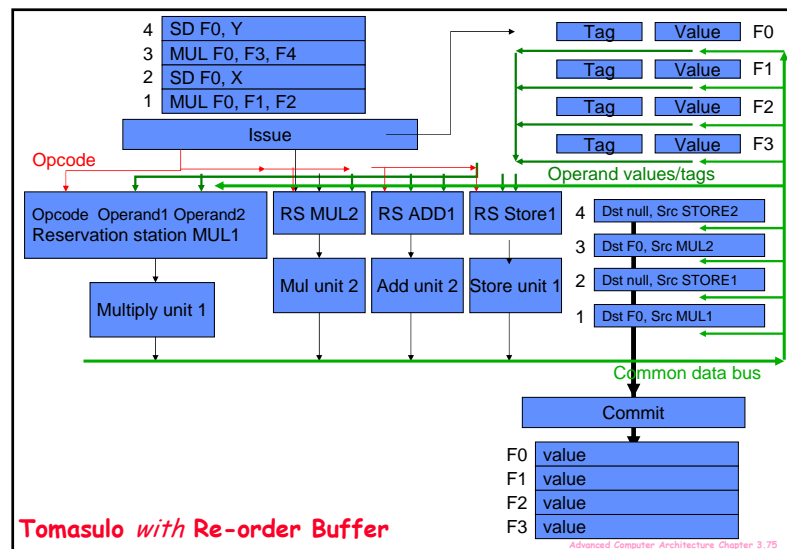
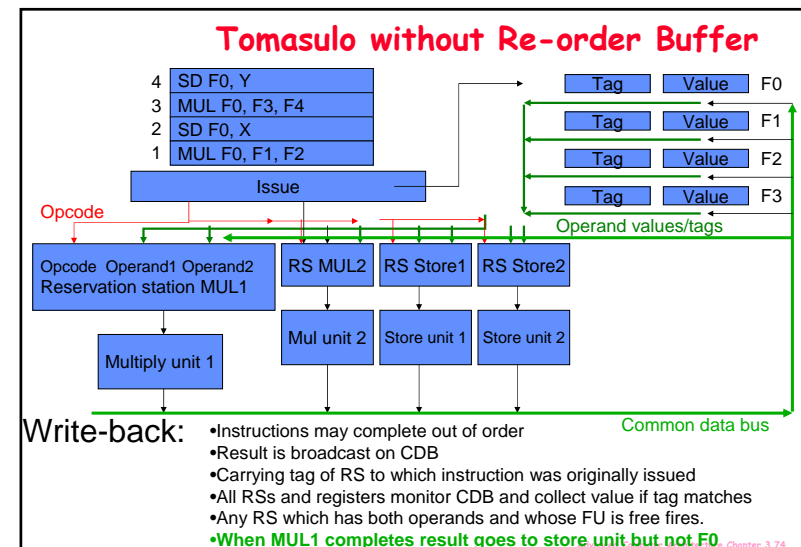
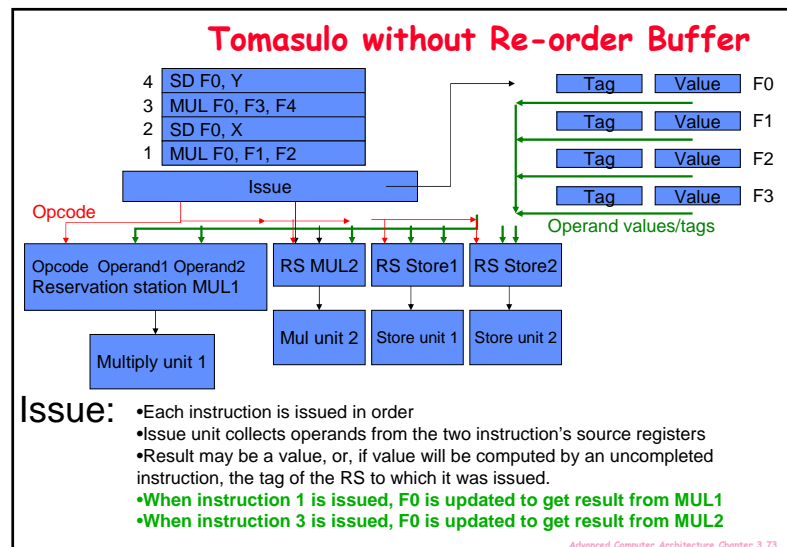
3. **Write result**—finish execution (WB)

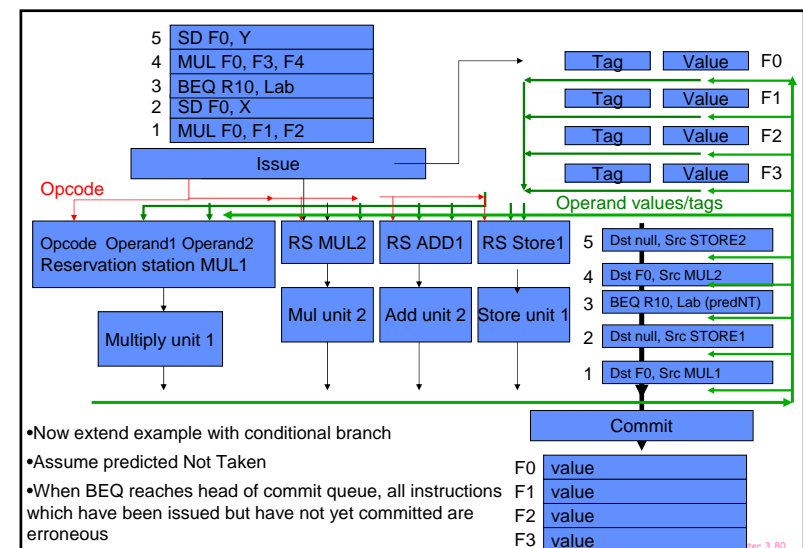
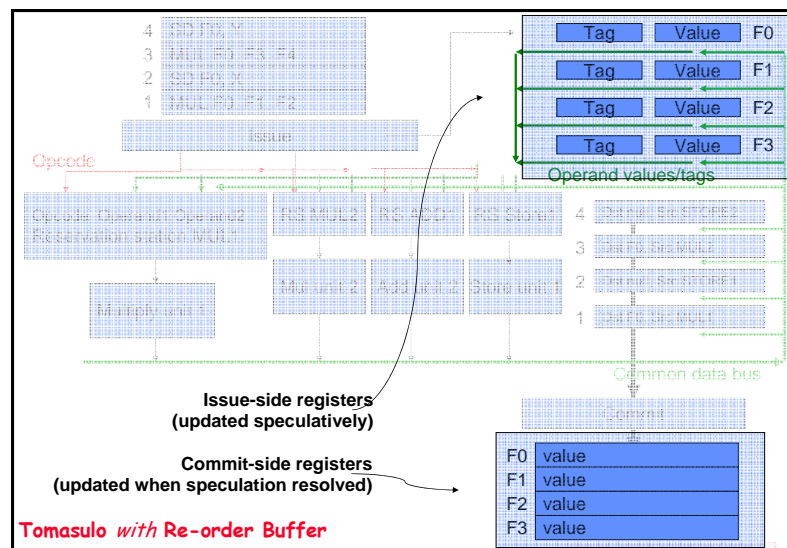
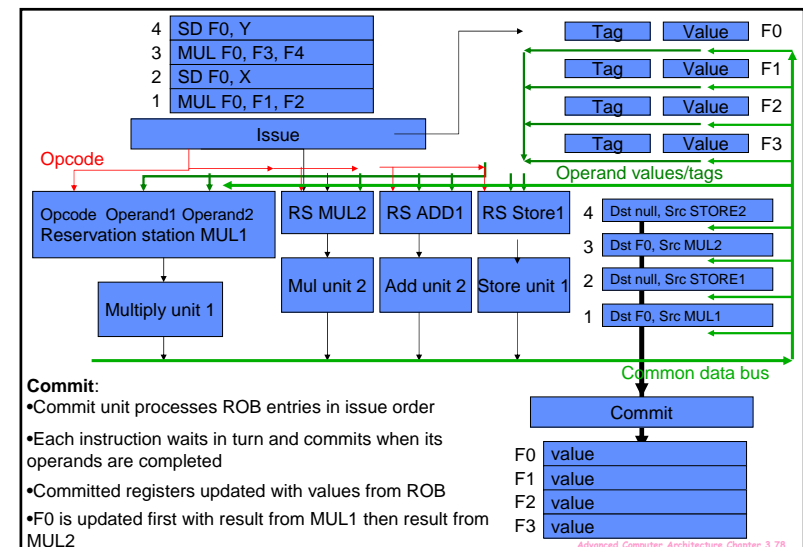
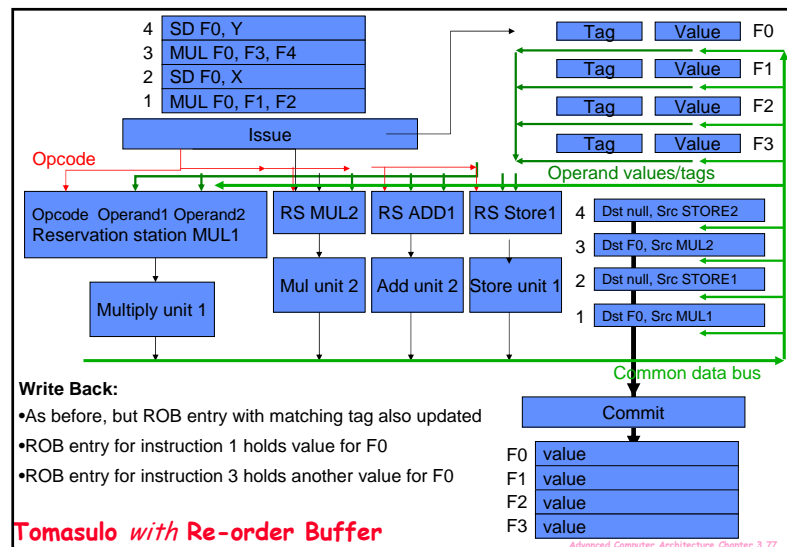
Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.

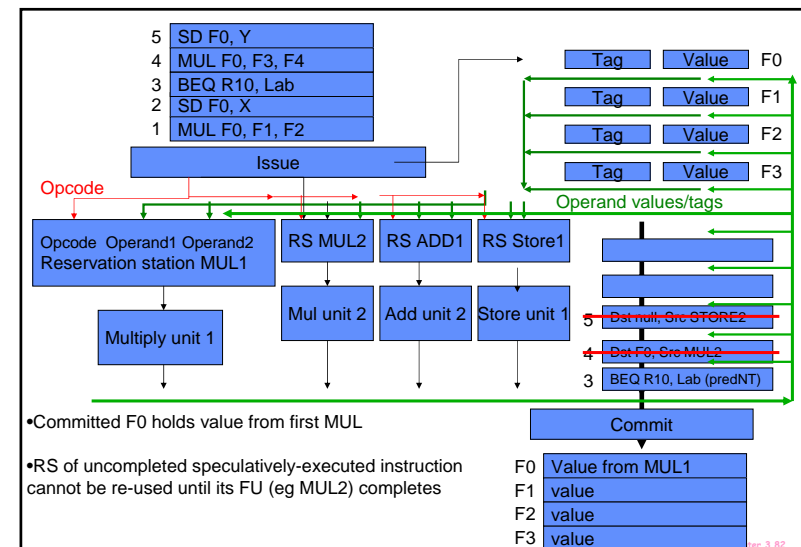
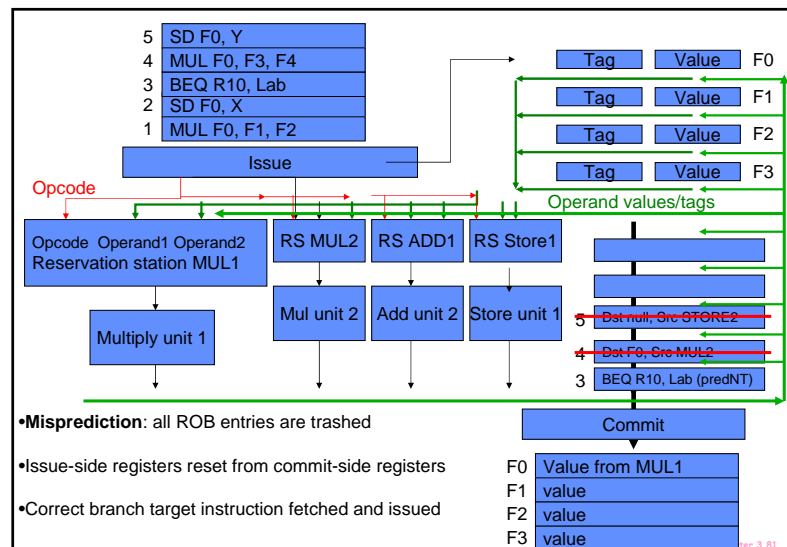
4. **Commit**—update register with reorder result

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”)

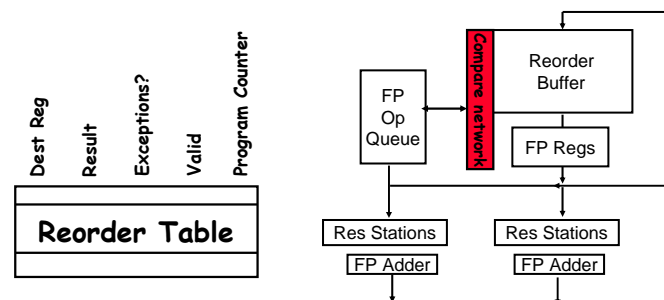
Advanced Computer Architecture Chapter 3.72







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>)

Advanced Computer Architecture Chapter 3.83

Some subtleties...

- 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

Advanced Computer Architecture Chapter 3.84

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

Advanced Computer Architecture Chapter 3.85

Resources

Papers:

- Instruction issue logic for high-performance, interruptable pipelined processors. G. S. Sohi, S. Vajapeyam. International Conference on Computer Architecture, 1987 (<http://doi.acm.org/10.1145/30350.30354>)
- Towards Kilo-instruction processors. Cristal, Santana, Valero, Martinez ACM Trans. Architecture and Code Optimization (<http://doi.acm.org/10.1145/1044823.1044825>)

Animations:

- SATSim SimpleScalar
 - <http://www.ece.gatech.edu/research/pica/SATSim/satsim.html>
- WebHase Tomasulo model:
 - www.dcs.ed.ac.uk/home/hase/webhase/demo/tomasulo.html
- Other WebHase animations - simple pipeline, Scoreboarding etc:
 - <http://www.icsa.informatics.ed.ac.uk/research/groups/hase/javahase/app-list.html>
- Israel Koren at U Massachusetts Amhurst:
 - <http://www.ecs.umass.edu/ece/koren/architecture/Tomasulo/AppletTomasulo.html>
 - <http://www.ecs.umass.edu/ece/koren/architecture/>

Processor performance

- SPEC benchmarks - see <http://www.spec.org/>
 - CPU benchmarks: <http://www.spec.org/cpu2000/results/cpu2000.html>
 - HPC benchmarks: <http://www.spec.org/hpc2002/results/hpc2002.html>
- Ace's hardware SPEC summary:
 - <http://www.aceshardware.com/SPECmine/top.jsp>

Other simulators:

- Liberty: <http://liberty.cs.princeton.edu/>
- MicroLib: <http://microlib.org/>

Advanced Computer Architecture Chapter 3.86

360/91 design choices...

- Speculation:
 - "Rather than wait for a valid CC, fetches are initiated for two instruction double-words as a hedge against a successful branch. Following this, it is assumed that the branch will fail, and a "conditional mode" is established. In conditional mode, shown in Fig. 8, instructions are decoded and conditionally forwarded to the execution units, and concomitant operand fetches are initiated. The execution units are inhibited from completing conditional instructions. When a valid condition code appears, the appropriate branching action is detected and activates or cancels the conditional instructions."
- Prediction:
 - [after mispredict] "the role of conditional mode is reversed, i.e., when the conditional branch is next encountered, it will be assumed that the branch will be taken. The conditionally issued instructions are from the target path rather than from the nobranch path as is the case when not in loop mode. A cancel requires recovery from the branch guess."
- Right:
 - Organizationally, primary emphasis is placed on (1) alleviating the disparity between storage time and circuit speed, and (2) the development of high speed floating-point arithmetic algorithms.
- Wrong:
 - "The complications of conditional mode, coupled with the fact that it is primarily aimed at circumventing storage access delays, indicate that a careful re-examination of its usefulness will be called for as the access time decreases."

Advanced Computer Architecture Chapter 3.87

Tomasulo Algorithm and Branch Prediction

➤ 360/91 predicted branches, but lacked full speculation:

- Instructions along predicted branch path can complete
- But results cannot be forwarded until branch outcome resolved

➤ Speculation with Reorder Buffer allows execution past branch, and then discard if branch fails

- The key difference is that speculative instructions can pass values to each other
- just need to hold instructions in buffer until branch can commit

Advanced Computer Architecture Chapter 3.88

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 \Rightarrow relative impact of the control stalls will be larger with the lower potential CPI in an n -issue processor

Advanced Computer Architecture Chapter 3.85

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

Advanced Computer Architecture Chapter 3.86

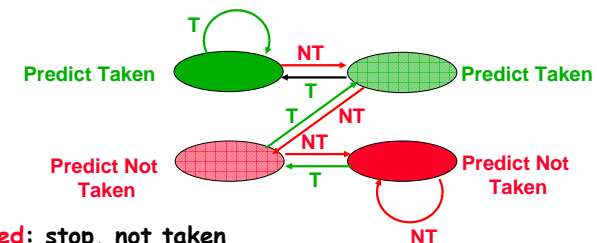
Dynamic Branch Prediction

- Performance = $f(\text{accuracy}, \text{cost of misprediction})$
- 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

Advanced Computer Architecture Chapter 3.91

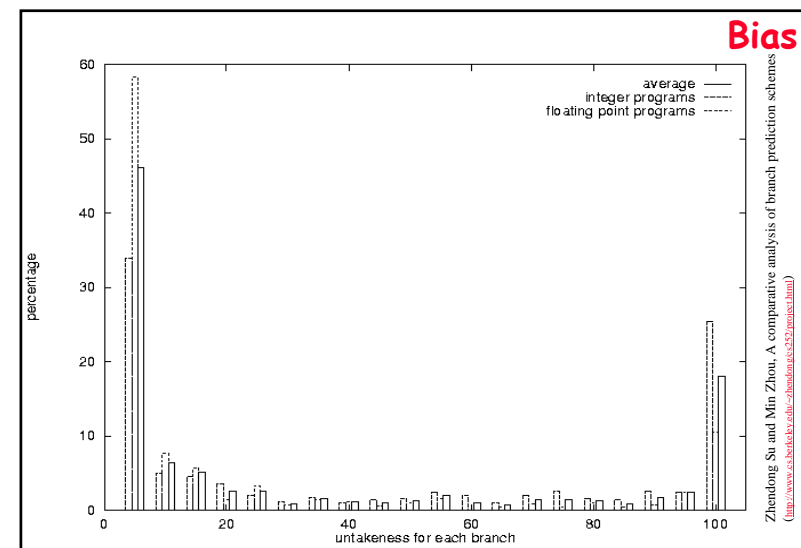
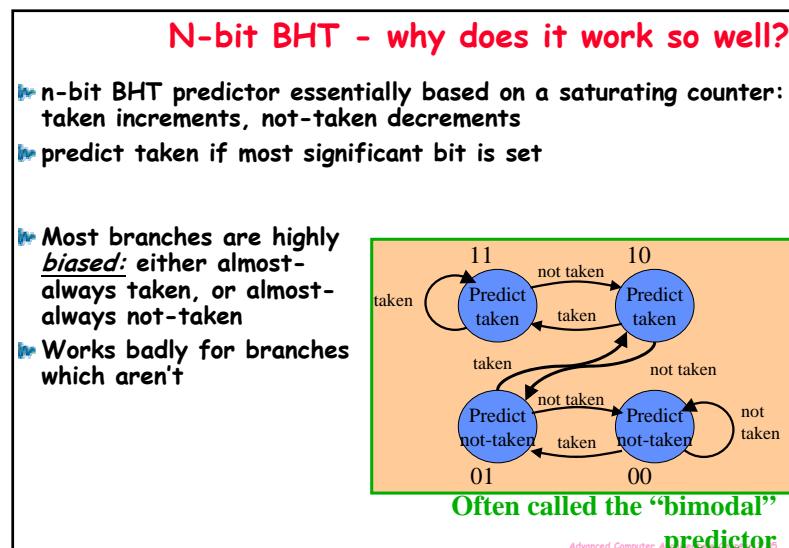
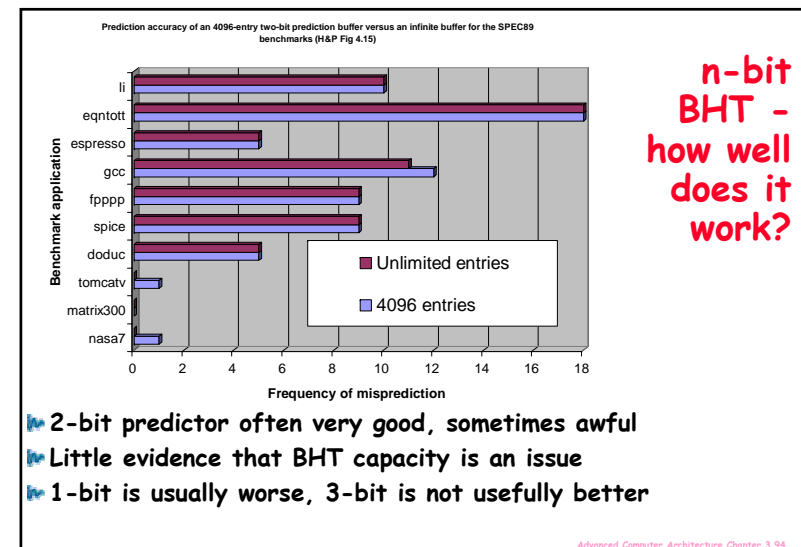
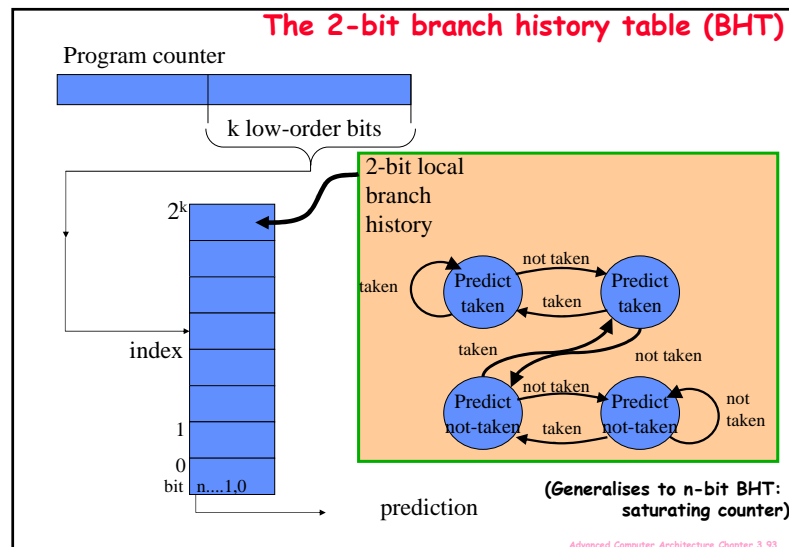
Dynamic Branch Prediction (Jim Smith, 1981)

- 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

Advanced Computer Architecture Chapter 3.92



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:

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?

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

Advanced Computer Architecture Chapter 3.97

Global history

Definition: **Global history**. The taken - not-taken history for all previously-executed branches.

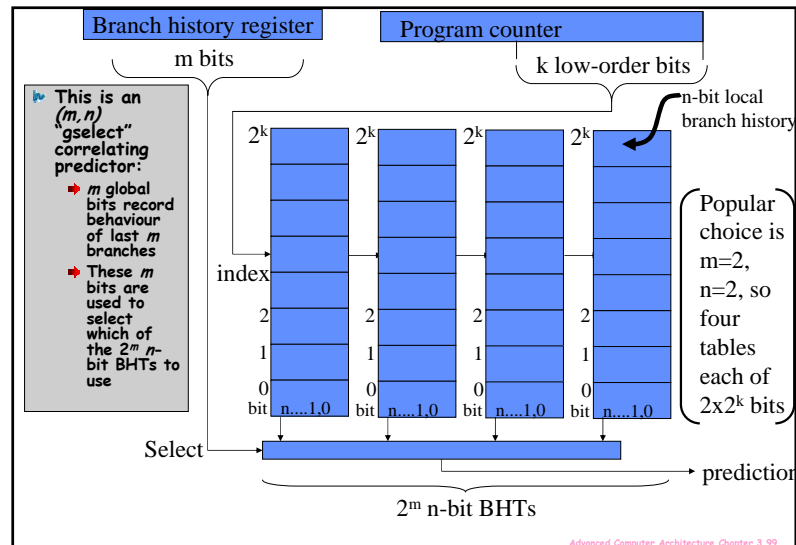
Idea: use global history to improve branch prediction

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

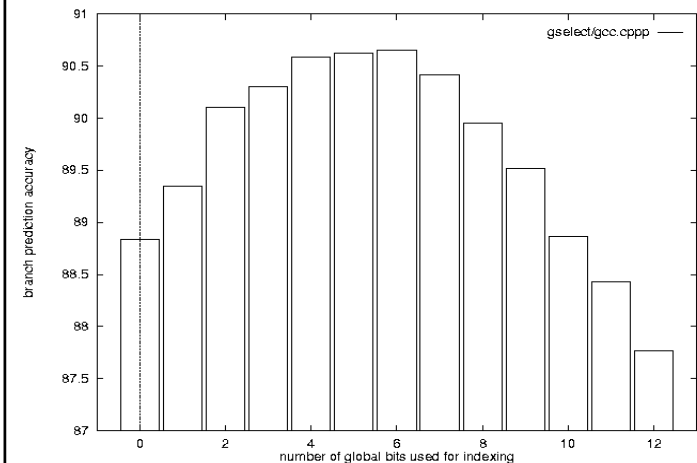
Question: How to combine local information with global information?

Advanced Computer Architecture Chapter 3.98



Advanced Computer Architecture Chapter 3.99

How many bits of branch history should be used?



(2,2) is good, (4,2) is better, (10,2) is worse

Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes (<http://www.cs.bkkhku.edu/~zhendong/su/2002/branch.html>)

Variations

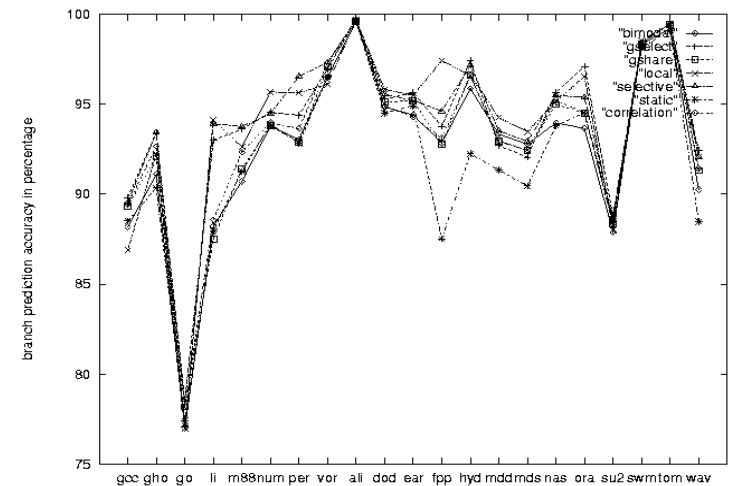
There are many variations on the idea:

- ♦ **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

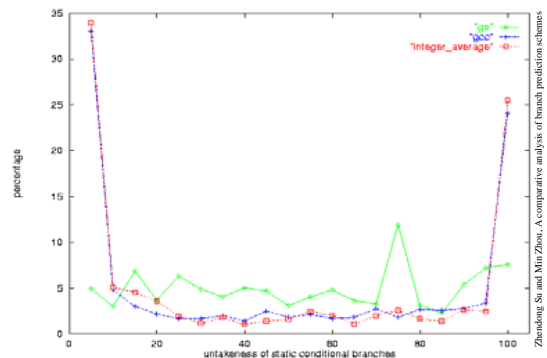
Advanced Computer Architecture Chapter 3.103

Horses for courses



Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes
(<http://www.cs.bklyn.cuny.edu/~zhendong/su20project.html>)

Extreme example - "go"



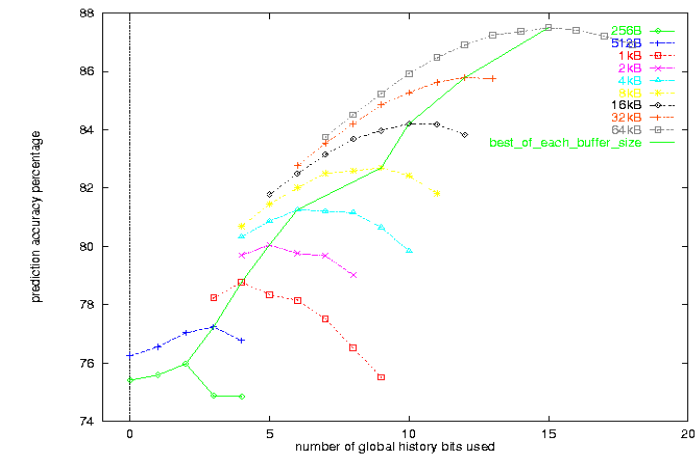
"go" is a SPEC95 benchmark code with highly-dynamic, highly-correlated branch behaviour

Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes
(<http://www.cs.bklyn.cuny.edu/~zhendong/su20project.html>)

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

Advanced Computer Architecture Chapter 3.103

Some dynamic applications have highly-correlated branches



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

Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes
(<http://www.cs.bklyn.cuny.edu/~zhendong/su20project.html>)

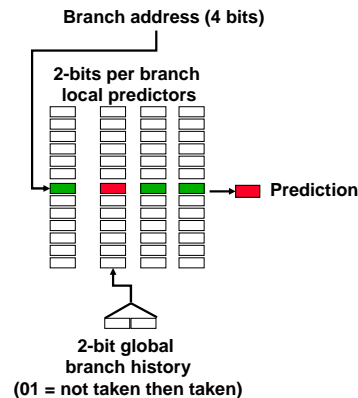
Advanced Computer Architecture Chapter 3.104

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

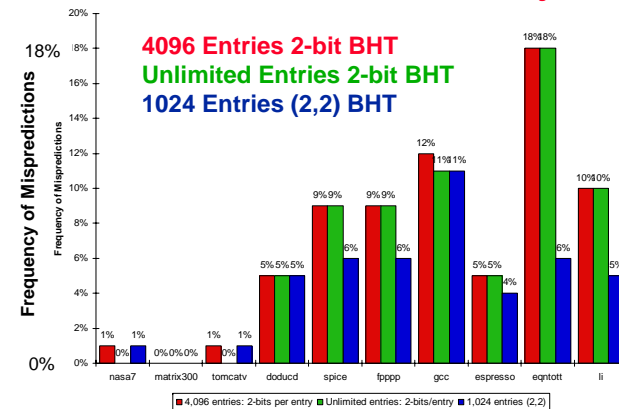
(2,2) predictor: 2-bit global, 2-bit local



Advanced Computer Architecture Chapter 3.105

Accuracy of Different Schemes

(H&P3ed Figure 3.15, p. 206)



Advanced Computer Architecture Chapter 3.105

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
egntott	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?

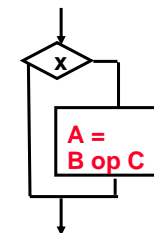
Advanced Computer Architecture Chapter 3.107

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

Advanced Computer Architecture Chapter 3.108

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

Advanced Computer Architecture Chapter 3.109

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

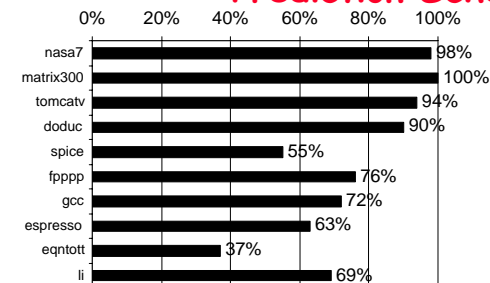
Advanced Computer Architecture Chapter 3.110

Tournament Predictor in Alpha 21264

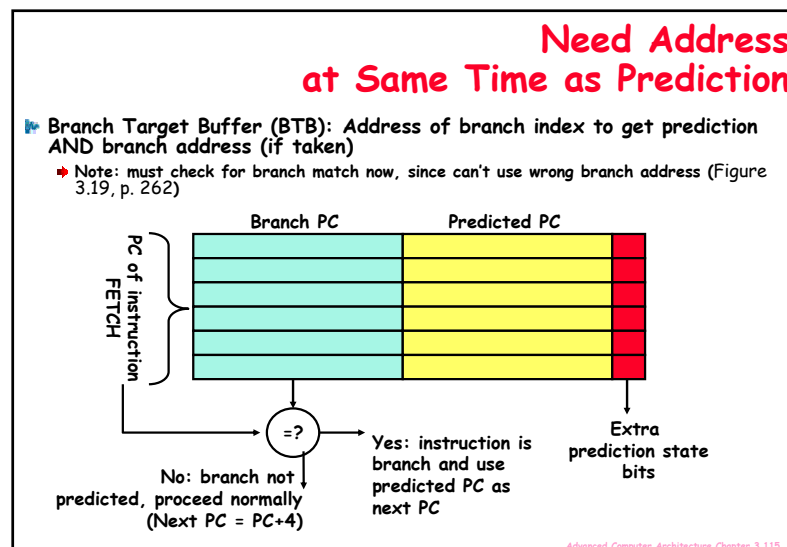
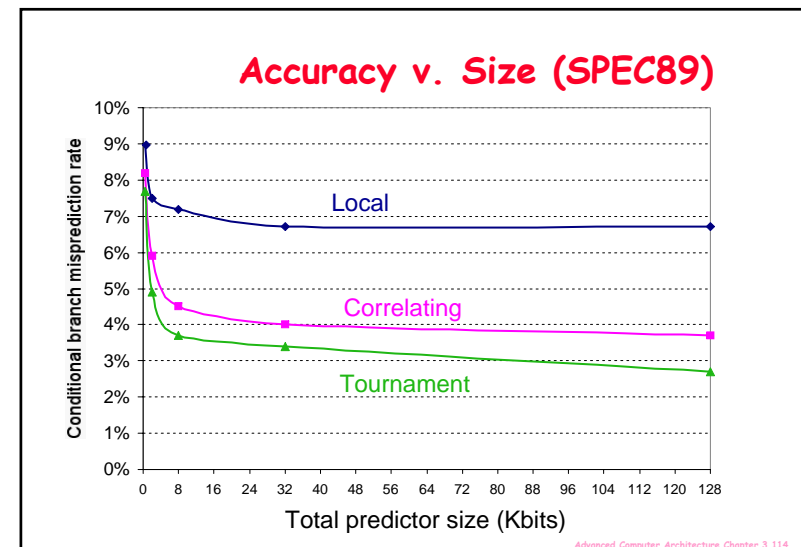
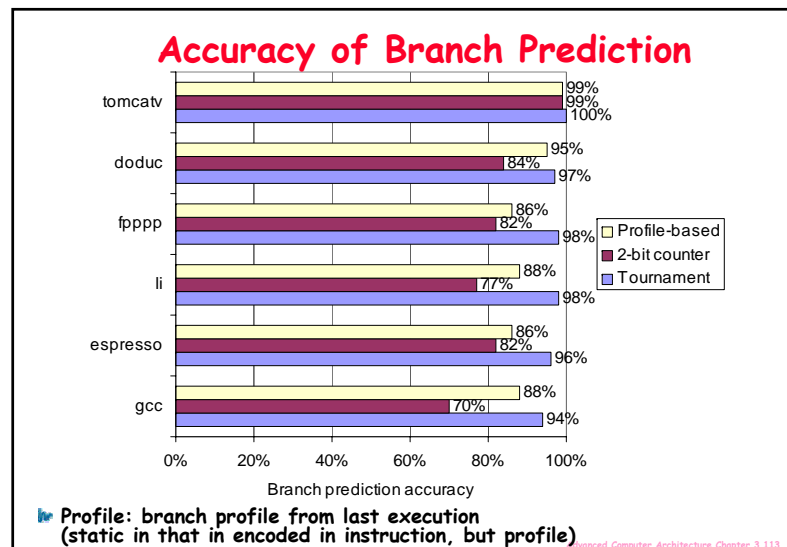
- ✚ 4K 2-bit counters to choose from among a global predictor and a local predictor
- ✚ **Global predictor** also has 4K entries and is indexed by the history of the last 12 branches; each entry in the global predictor is a standard 2-bit predictor
 - ✚ 12-bit pattern: ith bit 0 => ith prior branch not 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: $4K \cdot 2 + 4K \cdot 2 + 1K \cdot 10 + 1K \cdot 3 = 29K$ bits!
(~180,000 transistors)

Advanced Computer Architecture Chapter 3.111

% of predictions from local predictor in Tournament Prediction Scheme



Advanced Computer Architecture Chapter 3.112



- ### 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
- Advanced Computer Architecture Chapter 3.116

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)

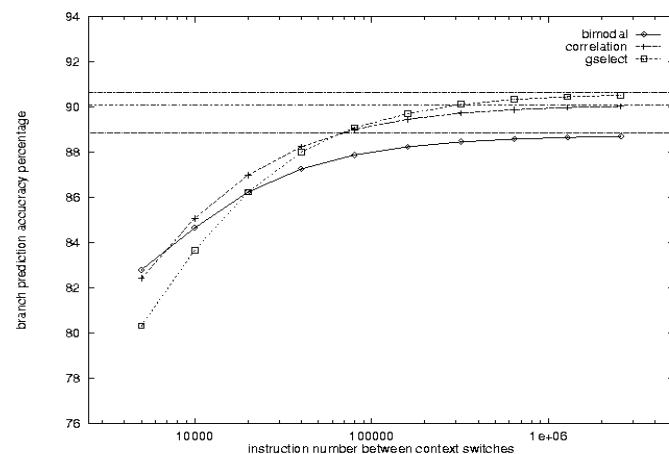
Advanced Computer Architecture Chapter 3.117

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

Advanced Computer Architecture Chapter 3.118

Warm-up...



Best predictor takes 20,000 instructions to overtake bimodal

Zhendong Su and Min Zhou, A comparative analysis of branch prediction schemes
(<http://www.cs.bkky.edu/~zhendong/su/2002/branch.html>)

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

Advanced Computer Architecture Chapter 3.120

Branch prediction resources

- Design tradeoffs for the Alpha EV8 Conditional Branch Predictor (André Seznec, Stephen Felix, Venkata Krishnan, Yiannakis Sazeides)

- ♦ SMT: 4 threads, wide-issue superscalar processor, 8-way issue, 512 registers (cancelled June 2001 when Alpha dropped)
- ♦ Paper: <http://citeseer.ist.psu.edu/seznec02design.html>
- ♦ Talk: <http://ce.et.tudelft.nl/cecoll/slides/PresDelft0803.ppt>

- Branch prediction in the Pentium family (Agner Fog)

- ♦ Reverse engineering Pentium branch predictors using direct access to BTB
- ♦ <http://www.x86.org/articles/branch/branchprediction.htm>

- Championship Branch Prediction Competition (CBP-1), organised by the Journal of Instruction-level Parallelism

- ♦ <http://www.jilp.org/cbp/>

Advanced Computer Architecture Chapter 3.121

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

Advanced Computer Architecture Chapter 3.122

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

Advanced Computer Architecture Chapter 3.123

Multiple Issue Issues

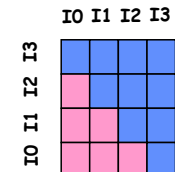
- issue packet:** group of instructions from fetch unit that could potentially issue in 1 clock

- ♦ 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



Advanced Computer Architecture Chapter 3.124

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 $\sim O(N^2-N)$ 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

Advanced Computer Architecture Chapter 3.125

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

Advanced Computer Architecture Chapter 3.126

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)

Advanced Computer Architecture Chapter 3.127

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: aggressive designs may speculate across multiple branches!

Advanced Computer Architecture Chapter 3.128

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.

Advanced Computer Architecture Chapter 3.139

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

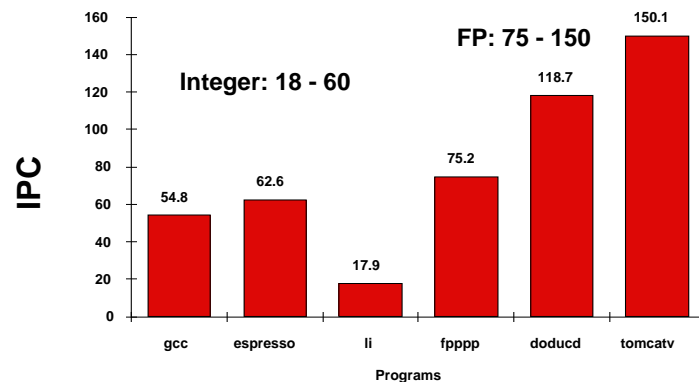
Also:

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

Advanced Computer Architecture Chapter 3.139

Upper Limit to ILP: Ideal Machine

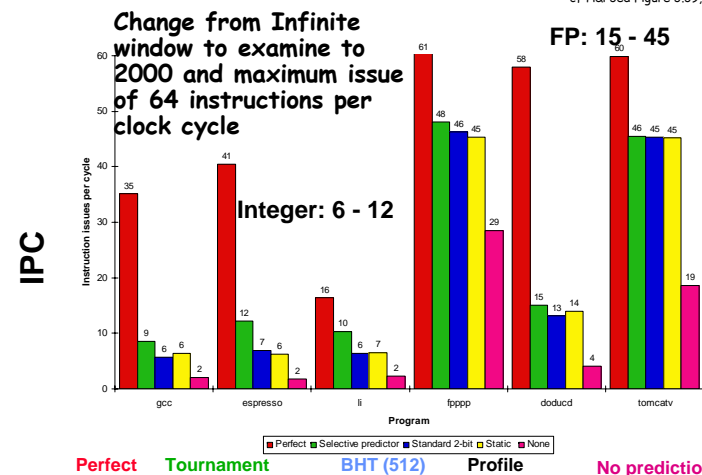
(H&P3ed Figure 3.35, page 242)



Advanced Computer Architecture Chapter 3.131

More Realistic HW: Branch Impact

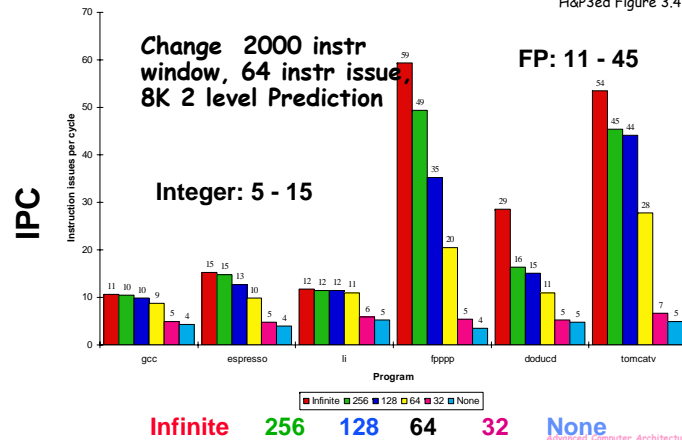
cf H&P3ed Figure 3.39, Page 248



Advanced Computer Architecture Chapter 3.132

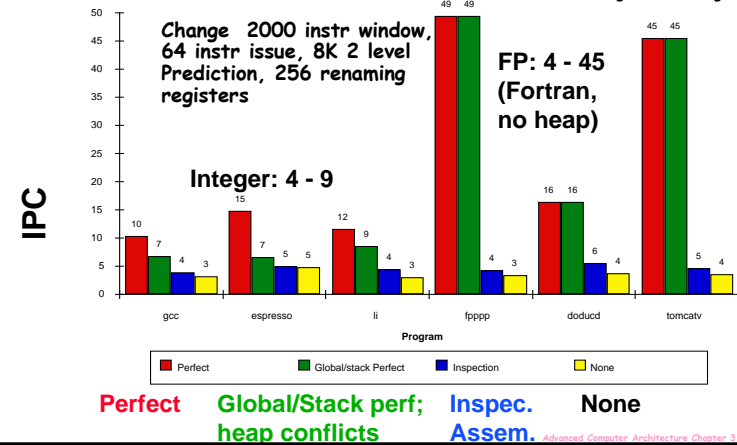
More Realistic HW: Renaming Register Impact

H&P3ed Figure 3.42, Page 251



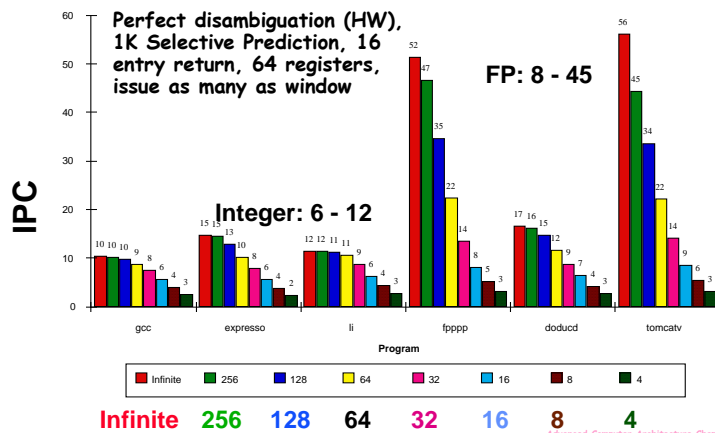
More Realistic HW: Memory Address Alias Impact

H&P3ed Figure 3.44, Page 252



Realistic HW for '00: Window Impact

(Figure 3.45, Page 309)



Limits to ILP - resources

Limits of Control Flow on Parallelism .

Monica S. Lam, Robert P. Wilson.
19th ISCA, May 1992, pages 19-21.

Limits of Instruction-Level Parallelism .

David W. Wall.
DEC-WRL Research Report 93/6, Nov. 1993

The Distribution of Instruction-Level and Machine Parallelism and Its Effect on Performance .

Norman P. Jouppi.
IEEE Transactions on Computers, Dec. 1989.

Advanced Computer Architecture Chapter 3.136

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 =

Value Locality and Load Value Prediction. Mikko H. Lipasti, Christopher B.
Wilkerson, John Paul Shen. Slides by Kundan Nepal:
http://www.lems.brown.edu/~iris/en291s9-04/lectures/kundanvalue_pred.pdf

Advanced Computer Architecture Chapter 3.137

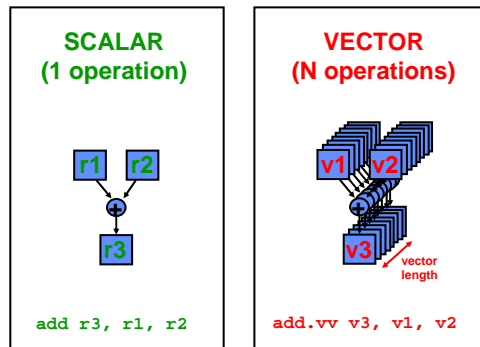
How to Exceed ILP Limits of this study?

- Vector instructions
 - Next section of this Chapter
- Simultaneous Multi-threading
 - Later section of this Chapter
- Multiprocessors
 - Later Chapter

Advanced Computer Architecture Chapter 3.138

Alternative Model: Vector Processing

- Vector processors have high-level operations that work on
linear arrays of numbers: "vectors"



25

Advanced Computer Architecture Chapter 3.139

Properties of Vector Processors

- Each result independent of previous result
 - => long pipeline, compiler ensures no dependencies
 - => high clock rate
- Vector instructions access memory with known pattern
 - => highly interleaved memory
 - => amortize memory latency of over - 64 elements
 - => no (data) caches required! (Do use instruction cache)
- Reduces branches and branch problems in pipelines
- Single vector instruction implies lots of work (- loop)
 - => fewer instruction fetches

Advanced Computer Architecture Chapter 3.140

Operation & Instruction Count: RISC v. Vector Processor

(from F. Quintana, U. Barcelona.)

Spec92fp Program	Operations (Millions)			Instructions (M)		
	RISC	Vector	R / V	RISC	Vector	R / V
swim256	115	95	1.1x	115	0.8	142x
hydro2d	58	40	1.4x	58	0.8	71x
nasa7	69	41	1.7x	69	2.2	31x
su2cor	51	35	1.4x	51	1.8	29x
tomcatv	15	10	1.4x	15	1.3	11x
wave5	27	25	1.1x	27	7.2	4x
mdljdp2	32	52	0.6x	32	15.8	2x

Vector reduces ops by 1.2X, instructions by 20X

Advanced Computer Architecture Chapter 3.143

Styles of Vector Architectures

➤ **memory-memory vector processors:** all vector operations are memory to memory

➤ **vector-register processors:** all vector operations between vector registers (except load and store)

- Vector equivalent of load-store architectures
- Includes all vector machines since late 1980s: Cray, Convex, Fujitsu, Hitachi, NEC
- We assume vector-register for rest of lectures

Advanced Computer Architecture Chapter 3.142

Components of Vector Processor

➤ **Vector Register:** fixed length bank holding a single vector

- has at least 2 read and 1 write ports
- typically 8-32 vector registers, each holding 64-128 64-bit elements

➤ **Vector Functional Units (FUs):** fully pipelined, start new operation every clock

- typically 4 to 8 FUs: FP add, FP mult, FP reciprocal (1/X), integer add, logical, shift; may have multiple of same unit

➤ **Vector Load-Store Units (LSUs):** fully pipelined unit to load or store a vector; may have multiple LSUs

➤ **Scalar registers:** single element for FP scalar or address

➤ Cross-bar to connect FUs, LSUs, registers

Advanced Computer Architecture Chapter 3.143

"DLXV" Vector Instructions

Instr.	Operands	Operation	Comment
➤ ADDV	V1, V2, V3	V1 = V2 + V3	vector + vector
➤ ADD_SV	V1, <u>F0</u> , V2	V1 = <u>F0</u> + V2	scalar + vector
➤ MULTV	V1, V2, V3	V1 = V2 × V3	vector × vector
➤ MULSV	V1, <u>F0</u> , V2	V1 = <u>F0</u> × V2	scalar × vector
➤ LV	V1, R1	V1 = M[R1..R1+63]	load, stride=1
➤ LV_{WS}	V1, R1, R2	V1 = M[R1..R1+ <u>63</u> *R2]	load, stride=R2
➤ LV_I	V1, R1, V2	V1 = M[R1+ <u>V2</u> i], i=0..63]	indir. ("gather")
➤ CeqV	VM, V1, V2	VMASKi = (V1i = V2i)?	comp. setmask
➤ MOV	<u>VLR</u> , R1	Vec. Len. Reg. = R1	set vector length
➤ MOV	<u>VM</u> , R1	Vec. Mask = R1	set vector mask

Advanced Computer Architecture Chapter 3.144

Memory operations

Load/store operations move groups of data between registers and memory

Three types of addressing

- Unit stride
 - Fastest
- Non-unit (constant) stride
- Indexed (gather-scatter)
 - Vector equivalent of register indirect
 - Good for sparse arrays of data
 - Increases number of programs that vectorize

32

Advanced Computer Architecture Chapter 3.145

DAXPY ($Y = a * X + Y$)

Assuming vectors X, Y are length 64

Scalar vs. **Vector**

```

LD    F0,a           ;load scalar a
LV    V1,Rx          ;load vector X
      → MULTS V2,F0,V1 ;vector-scalar mult.
LV    V3,Ry          ;load vector Y
      ADDV V4,V2,V3   ;add
      SV   Ry,V4      ;store the result

LD    F0,a
ADDI  R4,Rx,#512      ;last address to load
loop: LD    F2,0(Rx)   ;load X(i)
      MULTD F2,F0,F2   ;a*X(i)
      LD    F4,0(Ry)   ;load Y(i)
      ADDD  F4,F2,F4   ;a*X(i) + Y(i)
      SD    F4,0(Ry)   ;store into Y(i)
      ADDI  Rx,Rx,#8    ;increment index to X
      ADDI  Ry,Ry,#8    ;increment index to Y
      SUB   R20,R4,Rx  ;compute bound
      BNZ   R20,loop   ;check if done
    
```

578 (2+9*64) vs. 321 (1+5*64) ops (1.8X)
 578 (2+9*64) vs. 6 instructions (96X)
 64 operation vectors + no loop overhead
 also 64X fewer pipeline hazards

Advanced Computer Architecture Chapter 3.145

Example Vector Machines

Machine	Year	Clock	Regs	Elements	FUs	LSUs
Cray 1	1976	80 MHz	8	64	6	1
Cray XMP	1983	120 MHz	8	64	8	2 L, 1 S
Cray YMP	1988	166 MHz	8	64	8	2 L, 1 S
Cray C-90	1991	240 MHz	8	128	8	4
Cray T-90	1996	455 MHz	8	128	8	4
Conv. C-1	1984	10 MHz	8	128	4	1
Conv. C-4	1994	133 MHz	16	128	3	1
Fuj. VP200	1982	133 MHz	8-256	32-1024	3	2
Fuj. VP300	1996	100 MHz	8-256	32-1024	3	2
NEC SX/2	1984	160 MHz	8+8K	256+var	16	8
NEC SX/3	1995	400 MHz	8+8K	256+var	16	8

Advanced Computer Architecture Chapter 3.147

Vector Linpack Performance (MFLOPS)

Machine	Year	Clock	100x1001kx1k	Peak(Procs)
Cray 1	1976	80 MHz	12 110	160(1)
Cray XMP	1983	120 MHz	121 218	940(4)
Cray YMP	1988	166 MHz	150 307	2,667(8)
Cray C-90	1991	240 MHz	387 902	15,238(16)
Cray T-90	1996	455 MHz	705 1603	57,600(32)
Conv. C-1	1984	10 MHz	3 --	20(1)
Conv. C-4	1994	135 MHz	160 2531	3240(4)
Fuj. VP200	1982	133 MHz	18 422	533(1)
NEC SX/2	1984	166 MHz	43 885	1300(1)
NEC SX/3	1995	400 MHz	368 2757	25,600(4)

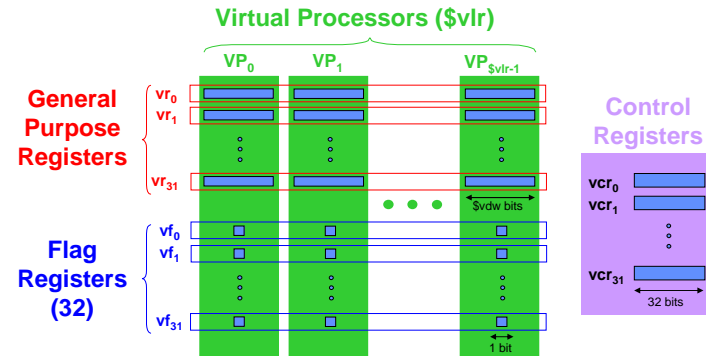
Advanced Computer Architecture Chapter 3.148

Virtual Processor Vector Model

- Vector operations are SIMD (single instruction multiple data) operations
- Each element is computed by a virtual processor (VP)
- Number of VPs given by vector length
 - vector control register

Advanced Computer Architecture Chapter 3.149

Vector Architectural State



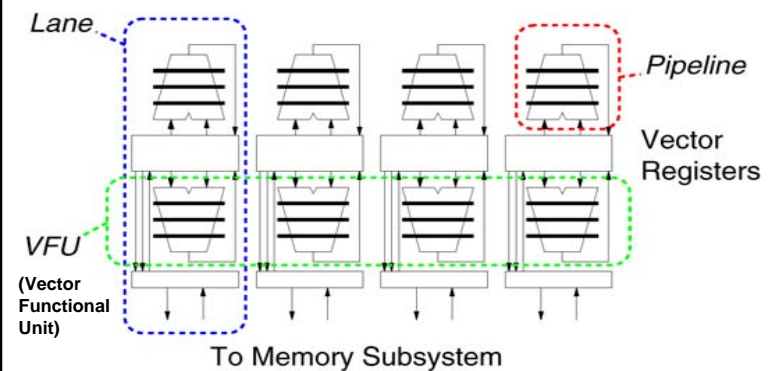
Advanced Computer Architecture Chapter 3.150

Vector Implementation

- Vector register file
 - Each register is an array of elements
 - Size of each register determines maximum vector length
 - Vector length register determines vector length for a particular operation
- Multiple parallel execution units = "lanes" (sometimes called "pipelines" or "pipes")

Advanced Computer Architecture Chapter 3.151

Vector Terminology: 4 lanes, 2 vector functional units



Advanced Computer Architecture Chapter 3.152

Vector Execution Time

Time = f(vector length, data dependencies, struct. hazards)

Initiation rate: rate that FU consumes vector elements
(= number of lanes; usually 1 or 2 on Cray T-90)

Convoy: set of vector instructions that can begin execution in same clock (no struct. or data hazards)

Chime: approx. time for a vector operation

m convoys take m chimes: if each vector length is n , then they take approx. $m \times n$ clock cycles (ignores overhead; good approximation for long vectors)

```

1: LV   V1,Rx    ;load vector X
2: MULV V2,F0,V1 ;vector-scalar mult.
   LV   V3,Ry    ;load vector Y
3: ADDV V4,V2,V3 ;add
4: SV   Ry,V4    ;store the result
    
```

4 convoys, 1 lane, VL=64
=> 4 x 64 = 256 clocks
(or 4 clocks per result)

Advanced Computer Architecture Chapter 3.153

Start-up time: pipeline latency time (depth of FU pipeline); another sources of overhead

Operation Start-up penalty (from CRAY-1)

Vector load/store 12

Vector multiply 7

Vector add 6

Assume convoys don't overlap; vector length = n :

Convoy	Start	1st result	last result	
1. LV	0	12	11+n (12+n-1)	
2. MULV, LV	12+n	12+n+12	23+2n	Load start-up
3. ADDV	24+2n	24+2n+6	29+3n	Wait convoy 2
4. SV	30+3n	30+3n+12	41+4n	Wait convoy 3

Advanced Computer Architecture Chapter 3.154

Vector Load/Store Units & Memories

Start-up overheads usually longer for LSUs

Memory system must sustain (# lanes x word) / clock cycle

Many Vector Procs. use banks (vs. simple interleaving):

1) support multiple loads/stores per cycle
=> multiple banks & address banks independently

2) support non-sequential accesses (see soon)

Note: No. memory banks > memory latency to avoid stalls

♦ m banks => m words per memory latency / clocks

♦ if $m < l$, then gap in memory pipeline:

clock: 0 ... / /+1 /+2 ... /+m-1 l+m ... 2 /
word: -- ... 0 1 2 ... m-1 -- ... m

♦ may have 1024 banks in SRAM

Advanced Computer Architecture Chapter 3.156

Vector Length

What to do when vector length is not exactly 64?

vector-length register (VLR) controls the length of any vector operation, including a vector load or store. (cannot be > the length of vector registers)

do 10 i = 1, n

10 Y(i) = a * X(i) + Y(i)

Don't know n until runtime!

n > Max. Vector Length (MVL)?

Advanced Computer Architecture Chapter 3.157

Strip Mining

- Suppose Vector Length > Max. Vector Length (MVL)?
- Strip mining**: generation of code such that each vector operation is done for a size S to the MVL
- 1st loop do short piece ($n \bmod \text{MVL}$), rest $\text{VL} = \text{MVL}$

```

low = 1
VL = (n mod MVL) /*find the odd size piece*/
do 1 j = 0, (n / MVL) /*outer loop*/
  do 10 i = low, low+VL-1 /*runs for length VL*/
    Y(i) = a*X(i) + Y(i) /*main operation*/
  10 continue
  low = low+VL /*start of next vector*/
  VL = MVL /*reset the length to max*/
1 continue

```

Advanced Computer Architecture Chapter 3.158

Common Vector Metrics

- R_∞ : MFLOPS rate on an infinite-length vector
 - vector "speed of light"
 - Real problems do not have unlimited vector lengths, and the start-up penalties encountered in real problems will be larger
 - (R_n is the MFLOPS rate for a vector of length n)
- $N_{1/2}$: The vector length needed to reach one-half of R_∞
 - a good measure of the impact of start-up
- N_V : The vector length needed to make vector mode faster than scalar mode
 - measures both start-up and speed of scalars relative to vectors, quality of connection of scalar unit to vector unit



Advanced Computer Architecture Chapter 3.159

Vector Stride

- Suppose adjacent elements not sequential in memory


```

do 10 i = 1, 100
  do 10 j = 1, 100
    A(i, j) = 0.0
    do 10 k = 1, 100
      10 A(i, j) = A(i, j) + B(i, k) * C(k, j)

```
- Either B or C accesses not adjacent (800 bytes between)
- stride**: distance separating elements that are to be merged into a single vector (caches do **unit stride**)
 - => **LVWS** (load vector with stride) instruction
- Strides => can cause bank conflicts (e.g., stride = 32 and 16 banks)
- Think of address per vector element

Advanced Computer Architecture Chapter 3.160

Compiler Vectorization on Cray XMP

Benchmark	%FP	%FP in vector
ADM	23%	68%
DYFESM	26%	95%
FLO52	41%	100%
MDG	28%	27%
MG3D	31%	86%
OCEAN	28%	58%
QCD	14%	1%
SPICE	16%	7%
TRACK	9%	23%
TRFD	22%	10%

(1% overall)

Advanced Computer Architecture Chapter 3.161

Vector Opt #1: Chaining

Suppose:

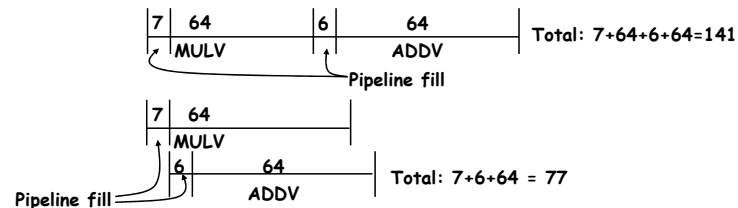
MULV V1, V2, V3

ADDV V4, V1, V5 ; separate convoy?

chaining: vector register (V1) is not as a single entity but as a group of individual registers, then pipeline forwarding can work on individual elements of a vector

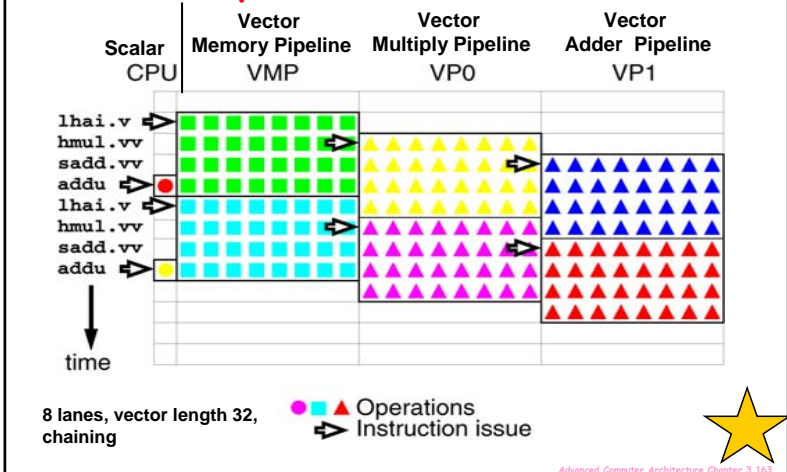
Flexible chaining: allow vector to chain to any other active vector operation => more read/write port

As long as enough HW, increases convoy size



Advanced Computer Architecture Chapter 3.162

Example Execution of Vector Code



Advanced Computer Architecture Chapter 3.163

Vector Opt #2: Conditional Execution

Suppose:

do 100 i = 1, 64

if (A(i) .ne. 0) then

A(i) = A(i) - B(i)

endif

100 continue

vector-mask control takes a Boolean vector: when **vector-mask register** is loaded from vector test, vector instructions operate only on vector elements whose corresponding entries in the vector-mask register are 1.

Still requires clock even if result not stored; if still performs operation, what about divide by 0?

Advanced Computer Architecture Chapter 3.164

Vector Opt #3: Sparse Matrices

Suppose:

do 100 i = 1, n

100 A(K(i)) = A(K(i)) + C(M(i))

gather (LVI) operation takes an **index vector** and fetches the vector whose elements are at the addresses given by adding a base address to the offsets given in the index vector => a nonsparse vector in a vector register

After these elements are operated on in dense form, the sparse vector can be stored in expanded form by a **scatter** store (SVI), using the same index vector

Can't be done by compiler since can't know K_i elements distinct, no dependencies; by compiler directive

Use CVI to create index 0, 1xm, 2xm, ..., 63xm

Advanced Computer Architecture Chapter 3.165

Sparse Matrix Example

Cache (1993) vs. Vector (1988)

	IBM RS6000	Cray YMP
Clock	72 MHz	167 MHz
Cache	256 KB	0.25 KB
Linpack	140 MFLOPS	160 (1.1)
Sparse Matrix (Cholesky Blocked)	17 MFLOPS	125 (7.3)

Cache: 1 address per cache block (32B to 64B)

Vector: 1 address per element (4B)



Advanced Computer Architecture Chapter 3.166

Applications

Limited to scientific computing?

- Multimedia Processing (compress., graphics, audio synth, image proc.)
- Standard benchmark kernels (Matrix Multiply, FFT, Convolution, Sort)
- Lossy Compression (JPEG, MPEG video and audio)
- Lossless Compression (Zero removal, RLE, Differencing, LZW)
- Cryptography (RSA, DES/IDEA, SHA/MD5)
- Speech and handwriting recognition
- Operating systems/Networking (memcpy, memset, parity, checksum)
- Databases (hash/join, data mining, image/video serving)
- Language run-time support (stdlib, garbage collection)
- even SPECint95

Advanced Computer Architecture Chapter 3.167

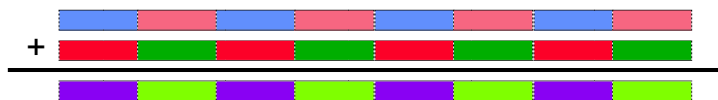
Vector for Multimedia?

Intel MMX/SSE instruction set extensions

- Similar extensions on other processor families, eg PowerPC AltiVec

Idea: pack multiple short-word operands into one long register

- Eg 128-bit register
 - 2 64-bit doubles
 - 4 32-bit floats or ints
 - 8 16-bit ints or fixed-point
 - 16 8-bit ints
 - Often with media-specific instructions eg saturated arithmetic



Claim: overall speedup 1.5 to 2X for 2D/3D graphics, audio, video, speech, comm., ...

- Initially hand-coded, accessible using special intrinsic functions
- Delivered via libraries such as the Intel Performance Primitives (IPP)
- Some support from compilers such as Intel's, but awkward constraints (eg alignment of operands)



Advanced Computer Architecture Chapter 3.168

Mediaprocesing: Vectorizable? Vector Lengths?

Kernel

- Matrix transpose/multiply
- DCT (video, communication)
- FFT (audio)
- Motion estimation (video)
- Gamma correction (video)
- Haar transform (media mining)
- Median filter (image processing)
- Separable convolution (img. proc.)

Vector length

- # vertices at once
- image width
- 256-1024
- image width, iw/16
- image width
- image width
- image width
- image width

(from Pradeep Dubey - IBM,
<http://www.research.ibm.com/people/p/pradeep/tutor.html>)

Advanced Computer Architecture Chapter 3.169

Vector Pitfalls

- ❖ Pitfall: Concentrating on peak performance and ignoring start-up overhead: N_v (length faster than scalar) > 100!
- ❖ Pitfall: Increasing vector performance, without comparable increases in scalar performance (Amdahl's Law)
 - failure of Cray competitor from his former company
- ❖ Pitfall: Good processor vector performance without providing good memory bandwidth
 - MMX?

Advanced Computer Architecture Chapter 3.172

Vector Advantages

- ❖ Easy to get high performance: N operations:
 - are independent
 - use same functional unit
 - access disjoint registers
 - access registers in same order as previous instructions
 - access contiguous memory words or known pattern
 - can exploit large memory bandwidth
 - hide memory latency (and any other latency)
- ❖ Scalable (get higher performance as more HW resources available)
- ❖ Compact: Describe N operations with 1 short instruction (v. VLIW)
- ❖ Predictable (real-time) performance vs. statistical performance (cache)
- ❖ Multimedia ready: choose $N * 64b$, $2N * 32b$, $4N * 16b$, $8N * 8b$
- ❖ Mature, developed compiler technology
- ❖ Vector Disadvantage: Out of Fashion



Advanced Computer Architecture Chapter 3.171

Vector Summary

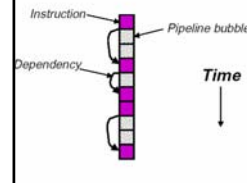
- ❖ Alternate model accommodates long memory latency, doesn't rely on caches as does Out-Of-Order, superscalar/VLIW designs
- ❖ Much easier for hardware: more powerful instructions, more predictable memory accesses, fewer hazards, fewer branches, fewer mispredicted branches, ...
- ❖ What % of computation is vectorizable?
- ❖ Is vector a good match to new apps such as multimedia, DSP?

Advanced Computer Architecture Chapter 3.173

Beyond ILP: Multithreading, Simultaneous Multithreading (SMT)

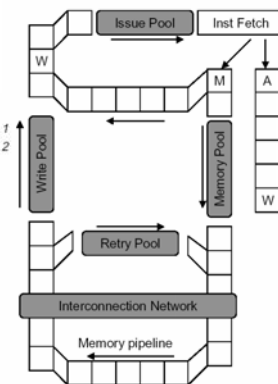
❖ Cray/Tera MTA

- <http://www.cray.com/products/systems/mta/>
- <http://www.utc.edu/~jdumas/cs460/papers/fa01/craymta/>

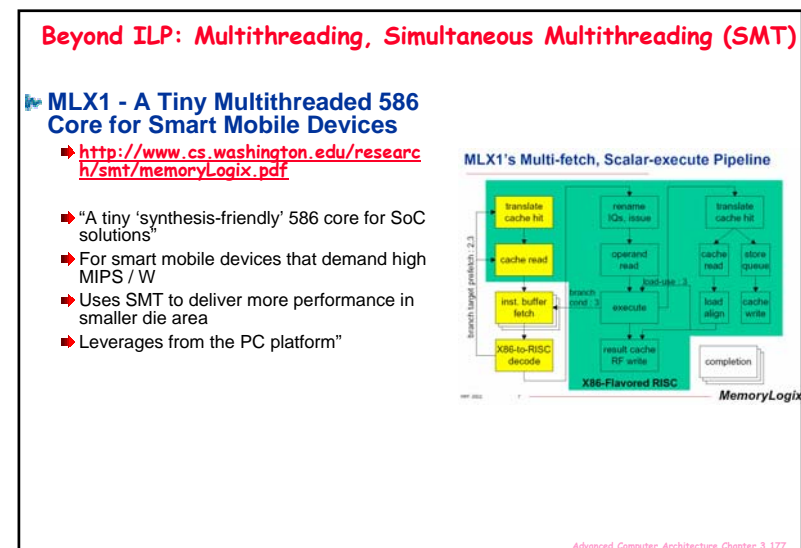
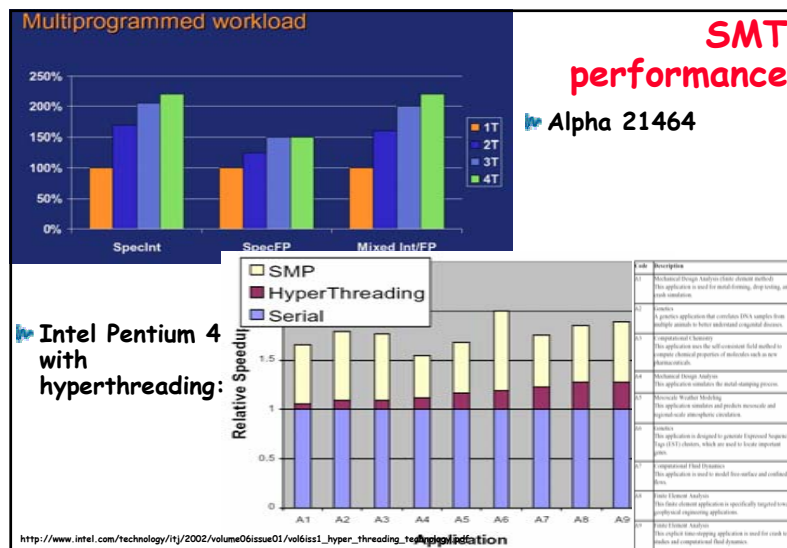
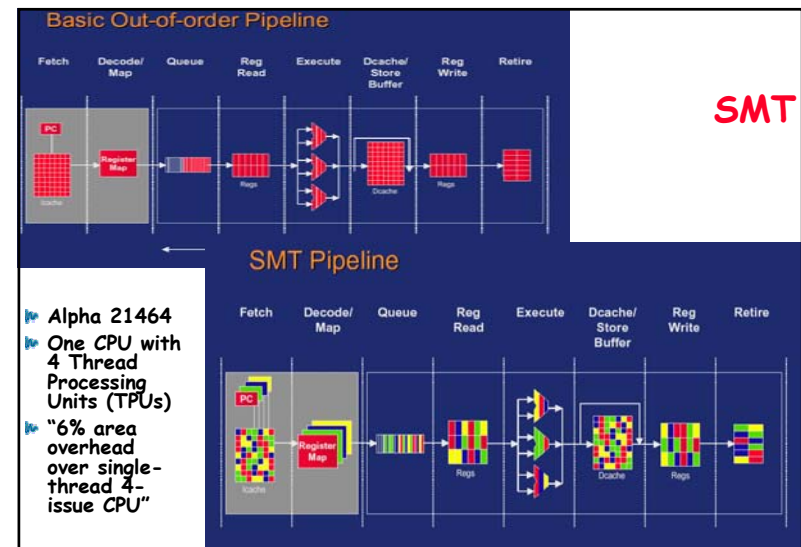
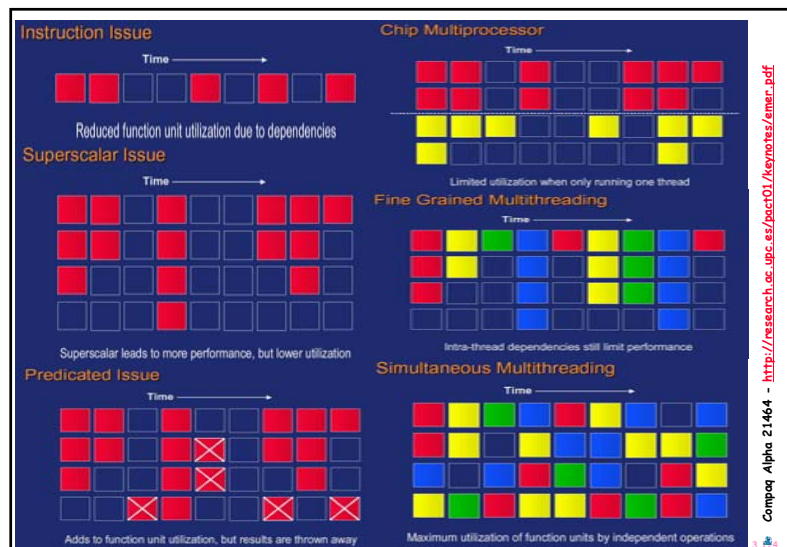


- Problem: Dependencies limit sustainable throughput of single instruction stream
- Solution: Interleave execution of two or more instruction streams on same hardware to increase utilization

(Source: Asanovic <http://www.cog.lcs.mit.edu/6.893-f2000/lectures/106-tera.pdf>)

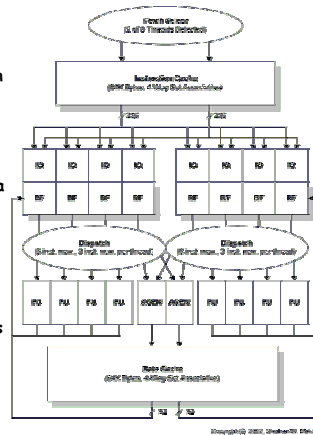


Advanced Computer Architecture Chapter 3.173



Clearwater Networks CNP810SP

- <http://www.zytek.com/~melvin/clearwater.html> SMT in a network processor
- 8 threads execute simultaneously, utilizing variable number of resources on a cycle by cycle basis.
- In each cycle 0-3 instructions can be executed from each of the threads depending on instruction dependencies and availability of resources
- Maximum IPC of the entire core is 10
- In each cycle two threads are selected for fetch and their respective program counters (PCs) are supplied to the dual-ported instruction cache
- Each port supplies eight instructions, so there is a maximum fetch bandwidth of 16 instructions
- The two threads chosen for fetch in each cycle are the two that have the fewest number of instructions in their respective IQs
- The 8 threads are divided into two clusters of 4 for ease of implementation.
- Thus the dispatch logic is split into two groups where each group dispatches up to six instructions from four different threads
- Eight function units are grouped into two sets of four, each set dedicated to a single cluster
- There are also two ports to the data cache that are shared by both clusters.
- A maximum of 10 instructions can be dispatched in each cycle. The function units are fully bypassed so that dependent instructions can be dispatched in successive cycles.



Advanced Computer Architecture Chapter 3.178

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?

Advanced Computer Architecture Chapter 3.179