Chapter 2
Instruction Level Parallelism

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Instruction Level Parallelism

- **Instruction-Level Parallelism (ILP): overlap the execution of instructions to improve performance**
- **2 approaches to exploit ILP:**
  1) **Rely on hardware to help discover and exploit the parallelism dynamically (e.g., Pentium 4, AMD Opteron, IBM Power) , and**
  2) **Rely on software technology to find parallelism, statically at compile-time (e.g., Itanium 2)**

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

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. E.g.,**

```
for (i=1; i<=1000; i=i+1)
        x[i] = x[i] + y[i];
```

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Loop-Level Parallelism

- **Exploit loop-level parallelism to parallelism by "unrolling loop" either by**
  - **dynamic via branch prediction or**
  - **static via loop unrolling by compiler**
  - **(Another way is vectors, to be covered later)**
- **Determining instruction dependence is critical to Loop Level Parallelism**
- **If 2 instructions are**
  - **parallel, they can execute simultaneously in a pipeline of arbitrary depth without causing any stalls (assuming no structural hazards)**
  - **dependent, they are not parallel and must be executed in order, although they may often be partially overlapped**

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

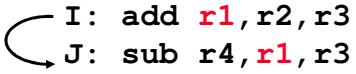
Types of Dependences

- Data Dependence (AKA "True dependence")
- Name Dependence
- Control Dependence

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Data Dependence and Hazards

- $Instr_J$ is data dependent on $Instr_I$ if:
  - $Instr_J$ reads an operand that $Instr_I$ writes



  - or $Instr_J$ is data dependent on $Instr_K$ which is dependent on $Instr_I$
- If two instructions are data dependent, they cannot execute simultaneously or be completely overlapped
- Data dependence in instruction sequence ⇒ data dependence in source code ⇒ effect of original data dependence must be preserved
- If data dependence caused a hazard in pipeline, this is a Read After Write (RAW) hazard

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

ILP and Data Dependences

- HW/SW must preserve program order:
  order instructions would execute in if executed sequentially as determined by original source program
- Presence of dependence indicates potential for a hazard, but actual hazard and length of any stall is 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
- HW/SW goal: exploit parallelism by preserving program order only where it affects the outcome of the program

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

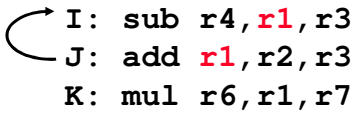
Name Dependence

- 2 instructions use same register or memory location (called a name) but there is no flow of data intended between the instructions associated with that name

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Name Dependence #1: Anti-dependence

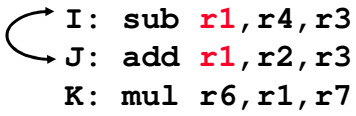
- $Instr_J$ writes operand ***before*** $Instr_I$ reads it



  Called an "anti-dependence" by compiler writers. This results from reuse of the name "r1"
- If anti-dependence caused a hazard in the pipeline, called a Write After Read (WAR) hazard
- WAR doesn't happen in most pipelines of the types we've seen so far because they have reads early (in ID) and writes late (in WB).

Name Dependence #2: Output dependence

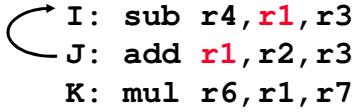
- $Instr_J$ writes operand ***before*** $Instr_I$ writes it.

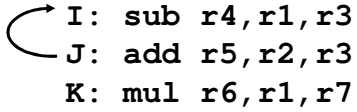


- Called an "output dependence" by compiler writers This also results from the reuse of name "r1"
- If anti-dependence caused a hazard in the pipeline, called a Write After Write (WAW) hazard

Resolving Name Dependences

- 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





Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

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`.

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Control Dependence Ignored

- **Control dependence need not be preserved**
  - **We are willing to execute instructions that should not have been executed, thereby violating the control dependences, if we can do so without affecting correctness of the program**
- **Instead, 2 properties critical to program correctness are**
  1) exception behavior and
  2) data flow

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

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:
```

  - **(Assume branches not delayed)**
- **If we move `LW` before `BEQZ`, we might have a (new) memory address exception**

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

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

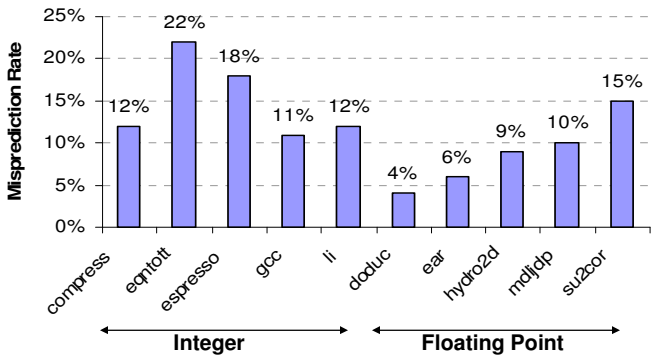
Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Static Branch Prediction

- To reorder code around branches, need to predict branch statically when compile
- Simplest scheme is to predict a branch as taken
  - Average misprediction = untaken branch frequency = 34% SPEC

• More accurate scheme predicts branches using profile information collected from earlier runs, and modify prediction based on last run:



Adapted from Patterson and Hennessey (Morgan Kauffman Pubs)

Dynamic Branch Prediction

- Why does prediction work?
  - Underlying algorithm has regularities
  - Data that is being operated on has regularities
  - Instruction sequence has redundancies that are artifacts of way that humans/compilers think about problems
- Is dynamic branch prediction better than static branch prediction?
  - Seems to be
  - There are a small number of important branches in programs which have dynamic behavior

Adapted from Patterson and Hennessey (Morgan Kauffman Pubs)

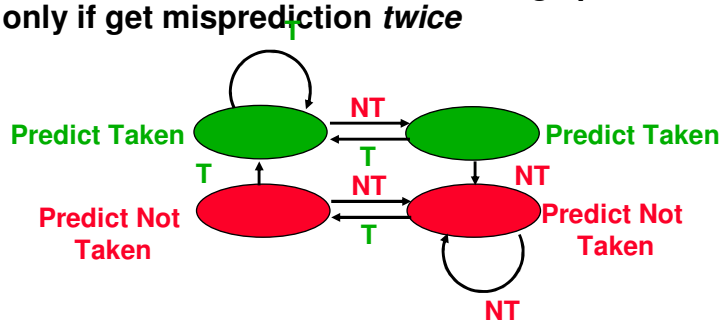
Dynamic Branch Prediction

- Performance = *ƒ*(accuracy, cost of misprediction)
- Branch History Table: Lower bits of PC address index table of 1-bit values
  - Says whether or not branch taken last time
  - No address check
- Problem: in a loop, 1-bit BHT will cause two 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

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Dynamic Branch Prediction

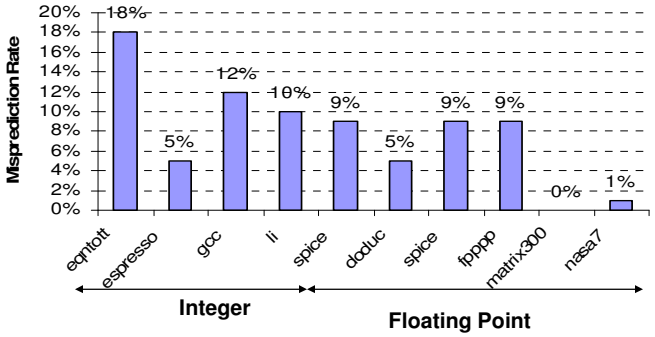
- Solution: 2-bit scheme where change prediction only if get misprediction *twice*



Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

BHT Accuracy

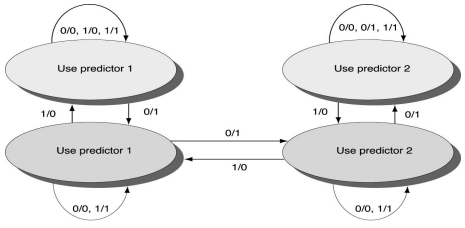
- **Mispredict because either:**
  - **Wrong guess for that branch**
  - **Got branch history of wrong branch when index the table**
- **4096 entry table:**



Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Tournament Predictors

- **Multilevel branch predictor**
- **Use *n*-bit saturating counter to choose between predictors**
- **Usual choice between global and local predictors**



Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Comparing Predictors (Fig. 2.8)

- Advantage of tournament predictor is ability to select the right predictor for a particular branch
  - Particularly crucial for integer benchmarks.
  - A typical tournament predictor will select the global predictor almost 40% of the time for the SPEC integer benchmarks and less than 15% of the time for the SPEC FP benchmarks

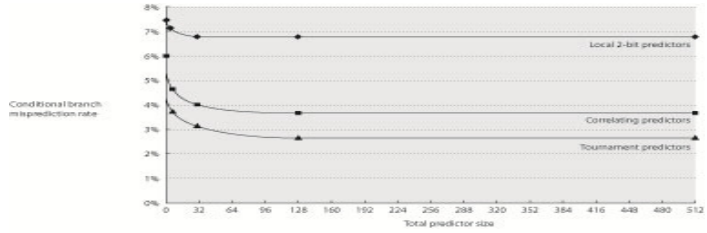


Figure 3.18— Hennessy/Patterson Computer Architecture Morgan Kaughmann Pub. Lineworks, Inc. — 12/07/01-tw

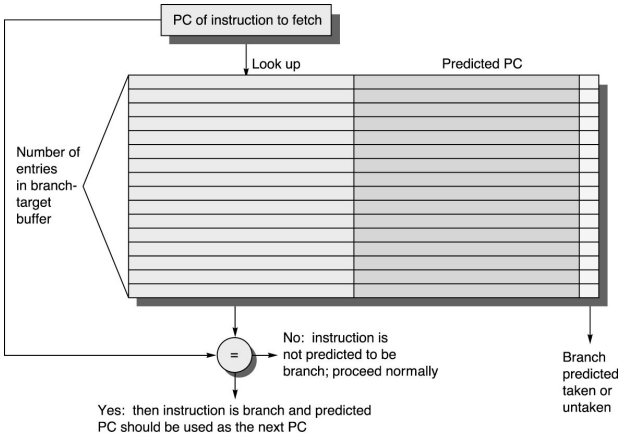
Adapted from Patterson and Hennessey (Morgan Kauffman Pubs)

Branch Target Buffers (BTB)

- Branch target calculation is costly and stalls the instruction fetch.
- BTB stores PCs the same way as caches
- The PC of a branch is sent to the BTB
- When a match is found the corresponding Predicted PC is returned
- If the branch was predicted taken, instruction fetch continues at the returned predicted PC

Adapted from Patterson and Hennessey (Morgan Kauffman Pubs)

Branch Target Buffers



Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Loop unrolling

```
Loop:   lw      $t0, 0($s1)
        addi    $s1, $s1, -4
        addu    $t0, $t0, $s2
        sw      $t0, 4($s1)
        bne     $s1, $zero, Loop
```

```
Loop:   lw      $t0, 0($s1)
        addi    $s1, $s1, -4
        addu    $t0, $t0, $s2
        sw      $t0, 4($s1)
        lw      $t0, 0($s1)
        addi    $s1, $s1, -4
        addu    $t0, $t0, $s2
        sw      $t0, 4($s1)
        bne     $s1, $zero, Loop
```

Dependency introduced by unrolling

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Loop unrolling (2)

```
Loop:   lw      $t0, 0($s1)
        addi    $s1, $s1, -4
        addu    $t0, $t0, $s2
        sw      $t0, 4($s1)
        lw      $t6, 0($s1)
        addi    $s1, $s1, -4
        addu    $t6, $t6, $s2
        sw      $t6, 4($s1)
        bne     $s1, $zero, Loop
```

Rename register to

Eliminate dependency

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Dynamic Scheduling (2.4)

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Advantages of Dynamic Scheduling

- **Dynamic scheduling -** hardware rearranges the instruction execution to reduce stalls while maintaining data flow and exception behavior
- It handles cases when dependences unknown at compile time
  - it allows the processor to tolerate unpredictable delays such as cache misses, by executing other code while waiting for the miss to resolve
- It allows code that compiled for one pipeline to run efficiently on a different pipeline
- It simplifies the compiler

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

HW Schemes: Instruction Parallelism

- Key idea: Allow instructions behind stall to proceed

```
DIVD  F0,F2,F4
ADDD  F10,F0,F8
SUBD  F12,F8,F14
```

- Enables out-of-order execution and allows out-of-order completion (e.g., `SUBD`)
  - In a dynamically scheduled pipeline, all instructions still pass through issue stage in order (in-order issue)
- Will distinguish when an instruction *begins execution* and when it *completes execution*; between 2 times, the instruction is *in execution*
- Note: Dynamic execution creates WAR and WAW hazards and makes exceptions harder

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

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

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

A Dynamic Algorithm: Tomasulo's

- For IBM 360/91 (before caches!)
  - ⇒ Long memory latency
- 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 1966 Computer?
- The descendants of this have flourished!
  - Alpha 21264, Pentium 4, AMD Opteron, Power 5, …

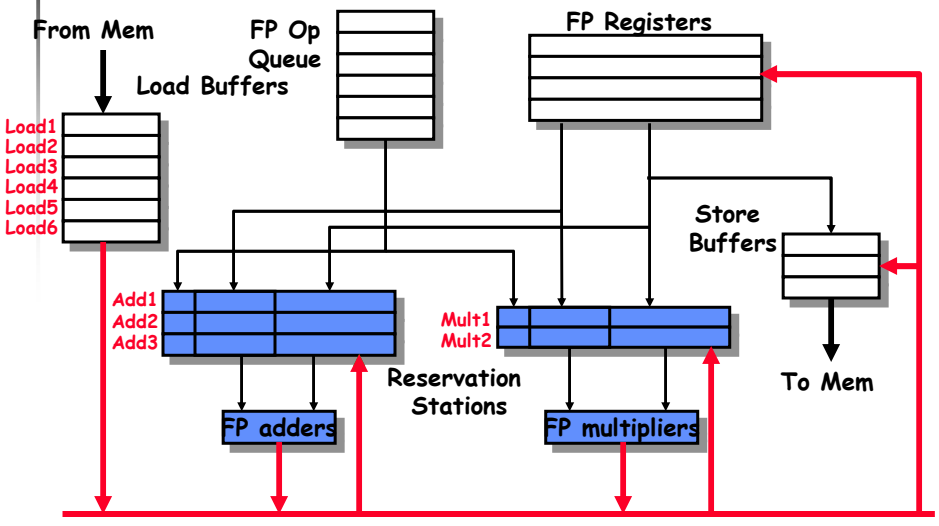
Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

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 ;
  - 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
  - Avoids RAW hazards by executing an instruction only when its operands are available
- Load and Stores treated as FUs with RSs as well
- Integer instructions can go past branches (predict taken), allowing FP ops beyond basic block in FP queue

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Tomasulo Organization



Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

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.

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Three Stages of Tomasulo Algorithm

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

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

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Tomasulo Example

Instruction stream

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

3 Load/Buffers

Reservation Stations:

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

FU count down

3 FP Adder R.S.
2 FP Mult R.S.

Register result status:

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

Clock cycle counter

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Tomasulo Example Cycle 1

Instruction status:

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

	Busy	Address
Load1	Yes	34+R2
Load2	No	
Load3	No	

Reservation Stations:

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

Register result status:

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

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Tomasulo Example Cycle 2

Instruction status:

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

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

Reservation Stations:

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

Tomasulo Example Cycle 3

Instruction status:

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

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

Reservation Stations:

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

Register result status:

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

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

Tomasulo Example Cycle 4

Instruction status:

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

	Busy	Address
Load1	No	
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

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

Register result status:

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

- **Load2 completing; what is waiting for Load2?**

Tomasulo Example Cycle 5

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

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

Register result status:

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

- **Timer starts down for Add1, Mult1**

Tomasulo Example Cycle 6

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

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

Register result status:

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

- **Issue ADDD here despite name dependency on F6?**

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Tomasulo Example Cycle 7

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

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

Register result status:

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

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

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Tomasulo Example Cycle 8

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

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

Register result status:

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

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Tomasulo Example Cycle 9

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

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

Register result status:

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

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Tomasulo Example Cycle 10

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

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

Register result status:

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

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

Tomasulo Example Cycle 11

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

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

Register result status:

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

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

Tomasulo Example Cycle 12

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

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

Register result status:

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

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Tomasulo Example Cycle 13

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

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

Register result status:

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

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Tomasulo Example Cycle 14

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

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

Register result status:

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

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Tomasulo Example Cycle 15

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

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

Register result status:

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

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

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Tomasulo Example Cycle 16

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

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

Register result status:

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

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

Faster than light computation (skip a couple of cycles)

Tomasulo Example Cycle 55

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

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

Register result status:

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

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Tomasulo Example Cycle 56

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

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

Register result status:

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

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

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Tomasulo Example Cycle 57

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

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

Register result status:

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

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

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**
- **Other perspective: Tomasulo building data flow dependency graph on the fly**

Tomasulo's scheme offers 2 major advantages

1. 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. Elimination of stalls for WAW and WAR hazards

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)

Tomasulo Drawbacks

- **Complexity**
  - **delays of 360/91, MIPS 10000, Alpha 21264, IBM PPC 620 in CA:AQA 2/e, but not in silicon!**
- **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**

Adapted from Patterson and Hennessey
(Morgan Kauffman Pubs)