EECS 252 Graduate Computer Architecture

Lec 8 – ILP in loops

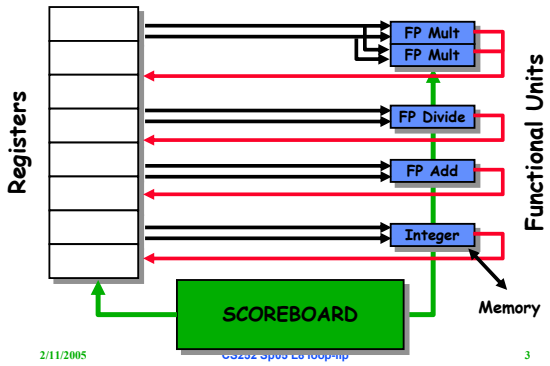
David Culler
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~culler
http://www-inst.eecs.berkeley.edu/~cs252

Review: Dynamic hardware techniques for out-of-order execution

- HW exploitation of ILP
  - Works even when can't know dependence at compile time.
  - Code for one machine runs well on another
- Scoreboard (ala CDC 6600 in 1963)
  - Centralized control structure
  - No register renaming, no forwarding
  - Pipeline stalls for WAR and WAW hazards.
  - Are these fundamental limitations??? (No)
- Reservation stations (ala IBM 360/91 in 1966)
  - Distributed control structures
  - Implicit renaming of registers (dispatched pointers)
  - WAR and WAW hazards eliminated by register renaming
  - Results broadcast to all reservation stations for RAW

Review: Scoreboard Architecture (CDC 6600)



Review: Four Stages of Scoreboard Control

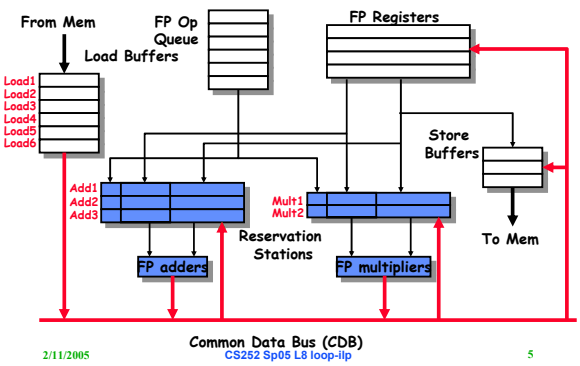
- Issue—decode instructions & check for structural hazards (ID1)
  - Instructions issued in program order (for hazard checking)
  - Don't issue if structural hazard
  - Don't issue if instruction is output dependent on any previously issued but uncompleted instruction (no WAW hazards)
- Read operands—wait until no data hazards, then read ops (ID2)
  - All real dependencies (RAW hazards) resolved in this stage, since we wait for instructions to write back data.
  - No forwarding of data in this model!
- Execution—operate on operands (EX)
  - The functional unit begins execution upon receiving operands. When the result is ready, it notifies the scoreboard that it has completed execution.
- Write result—finish execution (WB)
  - Stall until no WAR hazards with previous instructions:

Example:

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

CDC 6600 scoreboard would stall SUBD until ADDD reads operands

Review: Tomasulo Organization



Review: 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. Execution—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

Review: Comparison Cycle 62

Instruction status:

Instruction	j	k	Issue	Read Oper	Exec Comp	Write Result	Issue	Exec Comp	Write Result
LD	F6	34+ R2	1	2	3	4	1	3	4
LD	F2	45+ R3	5	6	7	8	2	4	5
MULTD	F0	F2 F4	6	9	19	20	3	15	16
SUBD	F8	F6 F2	7	9	11	12	4	7	8
DIVD	F10	F0 F6	8	21	61	62	5	56	57
ADDD	F6	F8 F2	13	14	16	22	6	10	11

- Why take longer on scoreboard/6600?
  - Structural Hazards
  - Lack of forwarding
- Deeper issue: WAW stalls

Outline

- Tomasulo on loops
- Register renaming
- R1000 example
- VLIW / EPIC
- Case Study
- Limits on Instruction Level Parallelism

Tomasulo Loop Example

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

- Assume Multiply takes 4 clocks
- Assume first load takes 8 clocks (cache miss), second load takes 1 clock (hit)
- To be clear, will show clocks for SUBI, BNEZ
- Reality: integer instructions ahead

Loop Example

Instruction status:

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

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

Reservation Stations:

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

Code:

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

Register result status

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

Loop Example Cycle 1

Instruction status:

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

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

Reservation Stations:

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

Code:

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

Register result status

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

Loop Example Cycle 2

Instruction status:

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

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

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F4)	Load1	
	Mult2	No					

Code:

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

Register result status

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

Loop Example Cycle 3

Instruction status: *Exec Write*

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

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

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F4)	Load1	
	Mult2	No					

Code:

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

Register result status

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

- Implicit renaming sets up "DataFlow" graph

Loop Example Cycle 4

Instruction status: *Exec Write*

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

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

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F4)	Load1	
	Mult2	No					

Code:

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

Register result status

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

- Dispatching SUBI Instruction

Loop Example Cycle 5

Instruction status: *Exec Write*

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

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

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F4)	Load1	
	Mult2	No					

Code:

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

Register result status

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

- And, BNEZ instruction

Loop Example Cycle 6

Instruction status: *Exec Write*

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

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

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F4)	Load1	
	Mult2	No					

Code:

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

Register result status

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

- Notice that F0 never sees Load from location 80

Loop Example Cycle 7

Instruction status: *Exec Write*

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

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

Reservation Stations:

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

Code:

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

Register result status

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

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

Loop Example Cycle 8

Instruction status: *Exec Write*

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

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

Reservation Stations:

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

Code:

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

Register result status

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

Loop Example Cycle 9

Instruction status: Exec Write

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

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

Reservation Stations:

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

Code:

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

Register result status

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

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

Loop Example Cycle 10

Instruction status: Exec Write

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

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

Reservation Stations:

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

Code:

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

Register result status

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

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

Loop Example Cycle 11

Instruction status: Exec Write

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

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

Reservation Stations:

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

Code:

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

Register result status

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

- Next load in sequence

Loop Example Cycle 12

Instruction status: Exec Write

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

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

Reservation Stations:

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

Code:

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

Register result status

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

- Why not issue third multiply?

Loop Example Cycle 13

Instruction status: Exec Write

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

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

Reservation Stations:

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

Code:

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

Register result status

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

Loop Example Cycle 14

Instruction status: Exec Write

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

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

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
0	Mult1	Yes	Multd	M[80]	R(F2)		
1	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

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

Register result status

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

- Mult1 completing. Who is waiting?

Loop Example Cycle 15

Instruction status:

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

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

Reservation Stations:

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

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

Register result status

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

- **Mult2 completing. Who is waiting?**

Loop Example Cycle 16

Instruction status:

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

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

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

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

Register result status

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

Loop Example Cycle 17

Instruction status:

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

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

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

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

Register result status

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

Loop Example Cycle 18

Instruction status:

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

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

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

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

Register result status

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

Loop Example Cycle 19

Instruction status:

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

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

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

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

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
19	64	Fu	Load3		Mult1						

Loop Example Cycle 20

Instruction status:

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

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

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

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

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
20	64	Fu	Load3		Mult1						

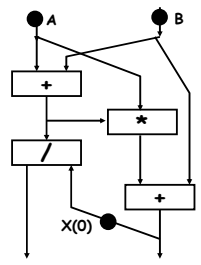
Why can Tomasulo overlap iterations of loops?

- 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
- Other idea: Tomasulo building dynamic "DataFlow" graph from instructions.

Data-Flow Architectures

- Basic Idea: Hardware respresents direct encoding of compiler dataflow graphs:

```
Input: a,b
   y:= (a+b)/x
   x:= (a*(a+b))+b
output: y,x
```

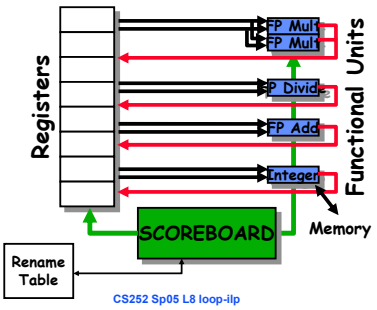


- Data flows along arcs in "Tokens".
- When two tokens arrive at compute box, box "fires" and produces new token.
- Split operations produce copies of tokens

Explicit Register Renaming

- Make use of a *physical* register file that is larger than number of registers specified by ISA
- Keep a translation table:
  - ISA register => physical register mapping
  - When register is written, replace table entry with new register from freelist.
  - Physical register becomes free when not being used by any instructions in progress.
- Pipeline can be exactly like "standard" DLX pipeline
  - IF, ID, EX, etc....
- Advantages:
  - Removes all WAR and WAW hazards
  - Like Tomasulo, good for allowing full out-of-order completion
  - Allows data to be fetched from a single register file
  - Makes speculative execution/precise interrupts easier:
    - » All that needs to be "undone" for precise break point is to undo the table mappings

Question: Can we use explicit register renaming with scoreboard?



Scoreboard Example

Instruction status:

Instruction		j	k	Issue	Oper	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				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
	Mult1	No								
	Add	No								
	Divide	No								

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
	FU	P0	P2	P4	P6	P8	P10	P12		P30

- Initialized Rename Table

Renamed Scoreboard 1

Instruction status:

Instruction		j	k	Issue	Oper	Comp	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				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	Yes	Load	P32		R2				Yes
	Int2	No								
	Mult1	No								
	Add	No								
	Divide	No								

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
1	FU	P0	P2	P4	P32	P8	P10	P12		P30

- Each instruction allocates free register
- Similar to single-assignment compiler transformation

Renamed Scoreboard 2

Instruction status:

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

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	Yes	Load	P32		R2				Yes
	Int2	Yes	Load	P34		R3				Yes
	Mult1	No								
	Add	No								
	Divide	No								

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
2	FU	P0	P34	P4	P32	P8	P10	P12		P30

Renamed Scoreboard 3

Instruction status:

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

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	Yes	Load	P32		R2				Yes
	Int2	Yes	Load	P34		R3				Yes
	Mult1	Yes	Multd	P36	P34	P4	Int2		No	Yes
	Add	No								
	Divide	No								

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
3	FU	P36	P34	P4	P32	P8	P10	P12		P30

Renamed Scoreboard 4

Instruction status:

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

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	Yes	Load	P34		R3				Yes
	Mult1	Yes	Multd	P36	P34	P4	Int2		No	Yes
	Add	Yes	Sub	P38	P32	P34		Int2	Yes	No
	Divide	No								

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
4	FU	P36	P34	P4	P32	P38	P10	P12		P30

Renamed Scoreboard 5

Instruction status:

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

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
5	FU	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 6

Instruction status:

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

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
10	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
2	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 7

Instruction status:

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

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
9	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
1	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
7	FU	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 8

Instruction status: Read Exec Write

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

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
8	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
0	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
8	FU	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 9

Instruction status: Read Exec Write

Instruction		j	k	Issue	Oper	Comp	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
7	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
9	FU	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 10

Instruction status: Read Exec Write

Instruction		j	k	Issue	Oper	Comp	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10			

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
6	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	Yes	Addd	P42	P38	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

WAR Hazard gone!

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
10	FU	P36	P34	P4	P42	P38	P40	P12		P30

- Notice that P32 not listed in Rename Table
  - Still live. Must not be reallocated by accident

Renamed Scoreboard 11

Instruction status: Read Exec Write

Instruction		j	k	Issue	Oper	Comp	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11		

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
5	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
2	Add	Yes	Addd	P42	P38	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
11	FU	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 12

Instruction status: Read Exec Write

Instruction		j	k	Issue	Oper	Comp	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11		

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
4	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
1	Add	Yes	Addd	P42	P38	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
12	FU	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 13

Instruction status: Read Exec Write

Instruction		j	k	Issue	Oper	Comp	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
3	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
0	Add	Yes	Addd	P42	P38	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
13	FU	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 14

Instruction status:

Instruction		j	k	Issue	Oper	Comp	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
2	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
14	FU	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 15

Instruction status:

Instruction		j	k	Issue	Oper	Comp	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
1	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
15	FU	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 16

Instruction status:

Instruction		j	k	Issue	Oper	Comp	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6	16	
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
0	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
16	FU	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 17

Instruction status:

Instruction		j	k	Issue	Oper	Comp	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6	16	17
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
	Mult1	No								
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		Yes	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
17	FU	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 18

Instruction status:

Instruction		j	k	Issue	Oper	Comp	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6	16	17
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5	18		
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

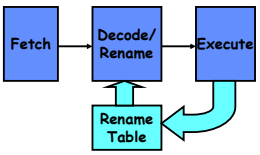
Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
	Mult1	No								
	Add	No								
40	Divide	Yes	Divd	P40	P36	P32	Mult1		Yes	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
18	FU	P36	P34	P4	P42	P38	P40	P12		P30

Explicit Register Renaming

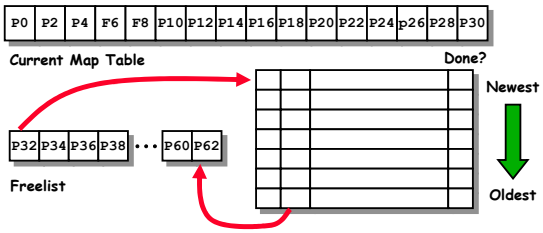
- **Make use of a *physical* register file that is larger than number of registers specified by ISA**
- **Keep a translation table:**
  - **ISA register => physical register mapping**
  - **When register is written, replace table entry with new register from freelist.**
  - **Physical register becomes free when not being used by any instructions in progress.**



Explicit Renaming Support Includes:

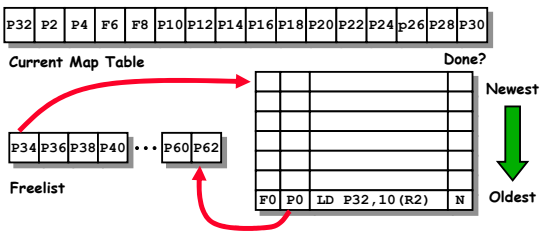
- Rapid access to a table of translations
- A physical register file that has more registers than specified by the ISA
- Ability to figure out which physical registers are free.
  - No free registers ⇒ stall on issue
- Thus, register renaming doesn't require reservation stations. However:
  - Many modern architectures use explicit register renaming + Tomasulo-like reservation stations to control execution.

Explicit register renaming:
R10000 Freelist Management



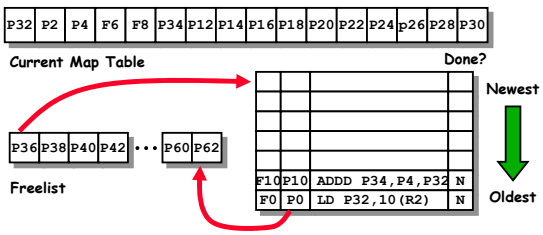
- Physical register file larger than ISA register file
- On issue, each instruction that modifies a register is allocated new physical register from freelist
- Used on: R10000, Alpha 21264, HP PA8000

Explicit register renaming:
R10000 Freelist Management

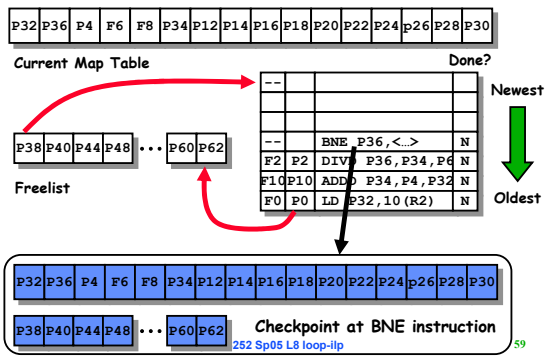


- Note that physical register P0 is "dead" (or not "live") past the point of this load.
  - When we go to commit the load, we free up

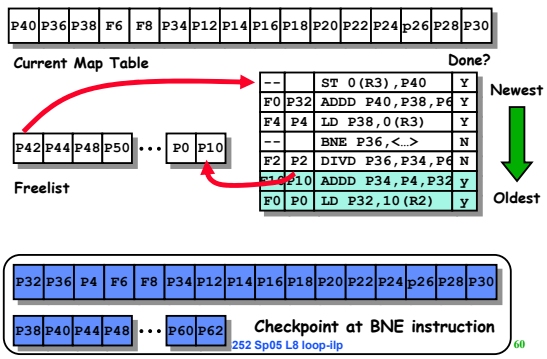
Explicit register renaming:
R10000 Freelist Management



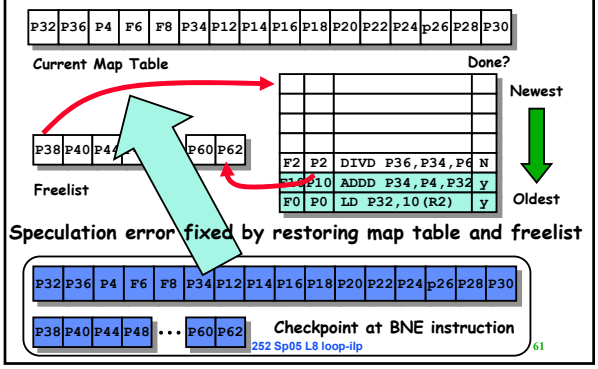
Explicit register renaming:
R10000 Freelist Management



Explicit register renaming:
R10000 Freelist Management



Explicit register renaming:
R10000 Freelist Management



What about Precise Interrupts?

- Both Scoreboard and Tomasulo have:

  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.

- Next lecture

Advantages of Explicit Renaming

- Decouples *renaming* from *scheduling:*
  - Pipeline can be exactly like "standard" DLX pipeline (perhaps with multiple operations issued per cycle)
  - Or, pipeline could be tomasulo-like or a scoreboard, etc.
  - Standard forwarding or bypassing could be used
- Allows data to be fetched from single register file
  - No need to bypass values from many places
  - This can be important for balancing pipeline
- Many processors use a variant of this technique:
  - R10000, Alpha 21264, HP PA8000
- Precise interrupt points:
  - All that needs to be "undone" for precise break point is to undo the table mappings
  - As long as old physical registers not yet reclaimed

Getting CPI < 1: Issuing Multiple Instructions/Cycle

- Two variations
- Superscalar: varying no. instructions/cycle (1 to 8), scheduled by compiler or by HW (Tomasulo)
  - IBM PowerPC, Sun UltraSparc, DEC Alpha, HP 8000
- (Very) Long Instruction Words (V)LIW: fixed number of instructions (4-16) scheduled by the compiler; put ops into wide templates
  - Joint HP/Intel agreement in 1999/2000?
  - Intel Architecture-64 (IA-64) 64-bit address
  - Style: "Explicitly Parallel Instruction Computer (EPIC)"
- Anticipated success lead to use of Instructions Per Clock cycle (IPC) vs. CPI

Getting CPI < 1: Issuing Multiple Instructions/Cycle

- Superscalar DLX: 2 instructions, 1 FP & 1 anything else
  - 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

Review: Unrolled Loop that Minimizes Stalls for Scalar

```
1  Loop: LD    F0,0(R1)
2        LD    F6,-8(R1)
3        LD    F10,-16(R1)
4        LD    F14,-24(R1)
5        ADDD  F4,F0,F2
6        ADDD  F8,F6,F2
7        ADDD  F12,F10,F2
8        ADDD  F16,F14,F2
9        SD    0(R1),F4
10       SD    -8(R1),F8
11       SD    -16(R1),F12
12       SUBI  R1,R1,#32
13       BNEZ  R1,LOOP
14       SD    8(R1),F16    ; 8-32 = -24
```

LD to ADDD: 1 Cycle
ADDD to SD: 2 Cycles

14 clock cycles, or 3.5 per iteration

Loop Unrolling in Superscalar

	Integer instruction	FP instruction	Clock cycle
Loop:	LD F0,0(R1)		1
	LD F6,-8(R1)		2
	LD F10,-16(R1)	ADDD F4,F0,F2	3
	LD F14,-24(R1)	ADDD F8,F6,F2	4
	LD F18,-32(R1)	ADDD F12,F10,F2	5
	SD 0(R1),F4	ADDD F16,F14,F2	6
	SD -8(R1),F8	ADDD F20,F18,F2	7
	SD -16(R1),F12		8
	SD -24(R1),F16		9
	SUBI R1,R1,#40		10
	BNEZ R1,LOOP		11
	SD -32(R1),F20		12

- Unrolled 5 times to avoid delays (+1 due to SS)
- 12 clocks, or 2.4 clocks per iteration (1.5X)

Dynamic Scheduling in Superscalar

- 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 FP loads might cause dependency between integer and FP issue:
  - Replace load reservation station with a load queue; operands must be read in the order they are fetched
  - Load checks addresses in Store Queue to avoid RAW violation
  - Store checks addresses in Load Queue to avoid WAR,WAW
  - Called "decoupled architecture"

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
  - 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
  - Multiported rename logic: must be able to rename same register multiple times in one cycle!
  - Rename logic one of key complexities in the way of multiple issue!
- VLIW: tradeoff instruction space for simple decoding
  - The long instruction word has room for many operations
  - By definition, all the operations the compiler puts in the long instruction word are independent => execute in parallel
  - E.g., 2 integer operations, 2 FP ops, 2 Memory refs, 1 branch
    - » 16 to 24 bits per field => 7*16 or 112 bits to 7*24 or 168 bits wide
  - Need compiling technique that schedules across several branches

Loop Unrolling in VLIW

Memory reference 1	Memory reference 2	FP operation 1	FP op. 2	Int. op/ branch	Clock
LD F0,0(R1)	LD F6,-8(R1)				1
LD F10,-16(R1)	LD F14,-24(R1)				2
LD F18,-32(R1)	LD F22,-40(R1)	ADDD F4,F0,F2	ADDD F8,F6,F2		3
LD F26,-48(R1)		ADDD F12,F10,F2	ADDD F16,F14,F2		4
		ADDD F20,F18,F2	ADDD F24,F22,F2		5
SD 0(R1),F4	SD -8(R1),F8	ADDD F28,F26,F2			6
SD -16(R1),F12	SD -24(R1),F16				7
SD -32(R1),F20	SD -40(R1),F24			SUBI R1,R1,#48	8
SD -0(R1),F28				BNEZ R1,LOOP	9

Unrolled 7 times to avoid delays
7 results in 9 clocks, or 1.3 clocks per iteration (1.8X)
Average: 2.5 ops per clock, 50% efficiency
Note: Need more registers in VLIW (15 vs. 6 in SS)

Recall: Software Pipelining with Loop Unrolling in VLIW

Memory reference 1	Memory reference 2	FP operation 1	FP op. 2	Int. op/ branch	Clock
LD F0,-48(R1)	ST 0(R1),F4	ADDD F4,F0,F2			1
LD F6,-56(R1)	ST -8(R1),F8	ADDD F8,F6,F2		SUBI R1,R1,#24	2
LD F10,-40(R1)	ST 8(R1),F12	ADDD F12,F10,F2		BNEZ R1,LOOP	3

- Software pipelined across 9 iterations of original loop
  - In each iteration of above loop, we:
    - » Store to m,m-8,m-16 (iterations I-3,I-2,I-1)
    - » Compute for m-24,m-32,m-40 (iterations I,I+1,I+2)
    - » Load from m-48,m-56,m-64 (iterations I+3,I+4,I+5)
- 9 results in 9 cycles, or 1 clock per iteration
- Average: 3.3 ops per clock, 66% efficiency

Note: Need less registers for software pipelining (only using 7 registers here, was using 15)

Advantages of HW (Tomasulo) vs. SW (VLIW) Speculation

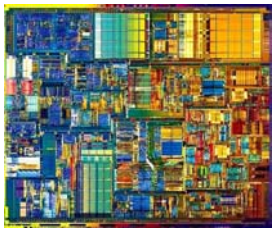
- HW determines address conflicts
- HW better branch prediction
- HW maintains precise exception model
- HW does not execute bookkeeping instructions
- Works across multiple implementations
- SW speculation is much easier for HW design

Superscalar v. VLIW

- Smaller code size
- Binary compatability across generations of hardware

- Simplified Hardware for decoding, issuing instructions
- No Interlock Hardware (compiler checks?)
- More registers, but simplified Hardware for Register Ports (multiple independent register files?)

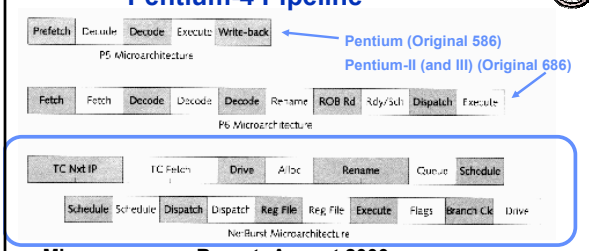
First Pentium-4: Willamette



Die Photo

Heat Sink

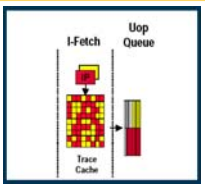
Pentium-4 Pipeline



- Microprocessor Report: August 2000
  - 20 Pipeline Stages!
  - Drive⇒ Wire Delay!
  - Trace-Cache: caching *paths* through the code for quick decoding.
  - Renaming: similar to Tomasulo architecture
  - Branch and *DATA prediction!*

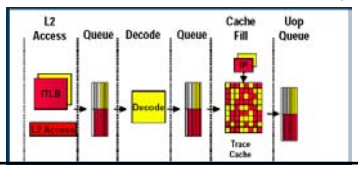
Where is the P4 "Decode"?

Hit:
(no Decode!)



- On Hit:
  - Trace Cache holds µops
  - Renamed/Scheduled
- Hit on complex ops:
  - Trace Cache only holds pointer to decode ROM
- On Miss:
  - Must decode x86 instructions into µops
  - Potentially slow!

Miss:
(Decode)



VLIW: Very Large Instruction Word

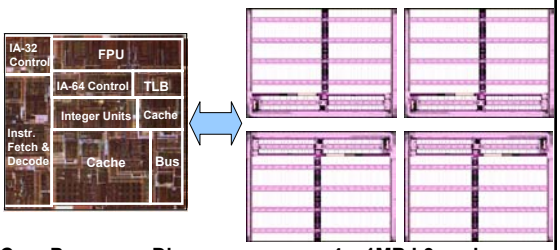
- Each "instruction" has explicit coding for multiple operations
  - In EPIC, grouping called a "packet"
  - In Transmeta, grouping called a "molecule" (with "atoms" as ops)
- Tradeoff instruction space for simple decoding
  - The long instruction word has room for many operations
  - By definition, all the operations the compiler puts in the long instruction word are independent => execute in parallel
  - E.g., 2 integer operations, 2 FP ops, 2 Memory refs, 1 branch
    - » 16 to 24 bits per field => 7*16 or 112 bits to 7*24 or 168 bits wide
  - Need compiling technique that schedules across several branches

Intel/HP "Explicitly Parallel Instruction Computer (EPIC)"

- 3 Instructions in 128 bit "groups"; field determines if instructions dependent or independent
  - Smaller code size than old VLIW, larger than x86/RISC
  - Groups can be linked to show independence > 3 instr
- 128 integer registers + 128 floating point registers
  - Not separate register files per functional unit as in old VLIW
- Hardware checks dependencies (interlocks => binary compatibility over time)
- Predicated execution (select 1 out of 64 1-bit flags) => 40% fewer mispredictions?
- IA-64: instruction set architecture; EPIC is type
  - VLIW = EPIC?
- Itanium™ is name of first implementation (2000/2001?)
  - Highly parallel and deeply pipelined hardware at 800Mhz
  - 6-wide, 10-stage pipeline at 800Mhz on 0.18 µ process

Itanium™ Processor Silicon

(Copyright: Intel at Hotchips '00)



Core Processor Die

4 x 1MB L3 cache

Itanium™ Machine Characteristics

(Copyright: Intel at Hotchips '00)

Frequency	800 MHz
Transistor Count	25.4M CPU; 295M L3
Process	0.18u CMOS, 6 metal layer
Package	Organic Land Grid Array
Machine Width	6 insts/clock (4 ALU/MM, 2 Ld/St, 2 FP, 3 Br)
Registers	14 ported 128 GR & 128 FR; 64 Predicates
Speculation	32 entry ALAT, Exception Deferral
Branch Prediction	Multilevel 4-stage Prediction Hierarchy
FP Compute Bandwidth	3.2 GFlops (DP/EP); 6.4 GFlops (SP)
Memory -> FP Bandwidth	4 DP (8 SP) operands/clock
Virtual Memory Support	64 entry ITLB, 32/96 2-level DTLB, VHPT
L2/L1 Cache	Dual ported 96K Unified & 16KD; 16KI
L2/L1 Latency	6 / 2 clocks
L3 Cache	4MB, 4-way s.a., BW of 12.8 GB/sec;
System Bus	2.1 GB/sec; 4-way Glueless MP Scalable to large (512+ proc) systems

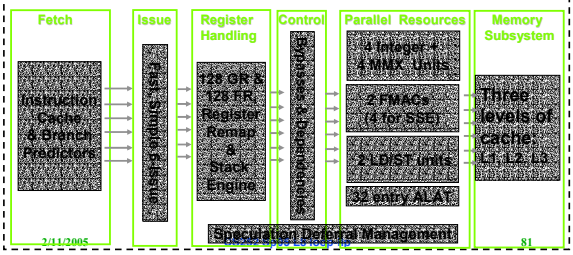
Itanium™ EPIC Design Maximizes SW-HW Synergy

(Copyright: Intel at Hotchips '00)

Architecture Features programmed by compiler:

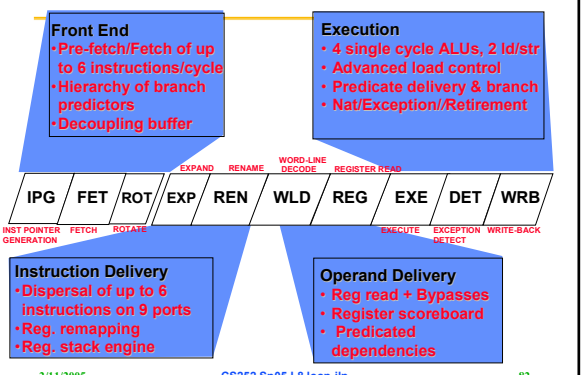
Branch Hints | *Explicit Parallelism* | *Register Stack & Rotation* | *Predication* | *Data & Control Speculation* | *Memory Hints*

Micro-architecture Features in hardware:



10 Stage In-Order Core Pipeline

(Copyright: Intel at Hotchips '00)



Limits to Multi-Issue Machines

- **Inherent limitations of ILP**
  - 1 branch in 5: How to keep a 5-way VLIW busy?
  - Latencies of units: many operations must be scheduled
  - Need about Pipeline Depth x No. Functional Units of independent operations to keep all pipelines busy.
  - Difficulties in building HW
  - Easy: More instruction bandwidth
  - Easy: Duplicate FUs to get parallel execution
  - Hard: Increase ports to Register File (bandwidth)
    - » VLIW example needs 7 read and 3 write for Int. Reg. & 5 read and 3 write for FP reg
  - Harder: Increase ports to memory (bandwidth)
  - Decoding Superscalar and impact on clock rate, pipeline depth?

Limits to Multi-Issue Machines

- **Limitations specific to either Superscalar or VLIW implementation**
  - Decode issue in Superscalar: how wide practical?
  - VLIW code size: unroll loops + wasted fields in VLIW
    - » IA-64 compresses dependent instructions, but still larger
  - VLIW lock step => 1 hazard & all instructions stall
    - » IA-64 not lock step? Dynamic pipeline?
  - VLIW & binary compatibilityIA-64 promises binary compatibility

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 mechanims with increasing HW budgets?
- Do we need to invent new HW/SW mechanisms to keep on processor performance curve?
  - Intel MMX
  - Motorola AltaVec
  - Supersparc Multimedia ops, etc.

Limits to ILP

Initial HW Model here; MIPS compilers.

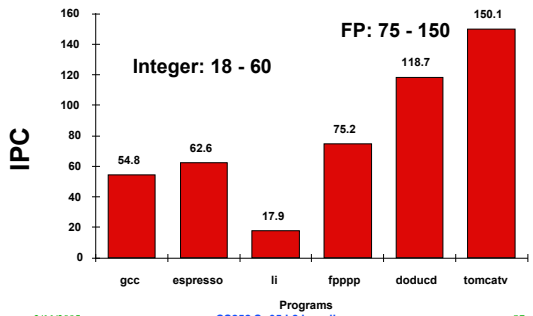
Assumptions for ideal/perfect machine to start:

1. *Register renaming*–infinite virtual registers and all WAW & WAR hazards are avoided
2. *Branch prediction*–perfect; no mispredictions
3. *Jump prediction*–all jumps perfectly predicted => 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

1 cycle latency for all instructions; unlimited number of instructions issued per clock cycle

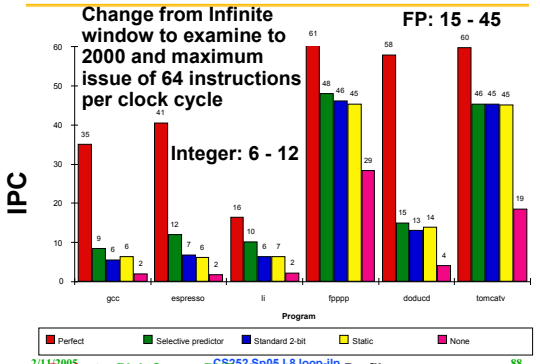
Upper Limit to ILP: Ideal Machine

(Figure 4.38, page 319)



More Realistic HW: Branch Impact

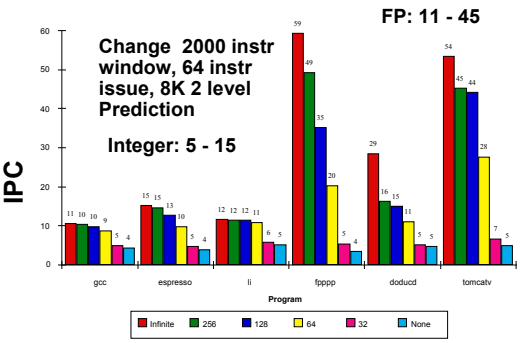
Figure 4.40, Page 323



Perfect Pick Cor. or BHT BHT (512) Profile No prediction

More Realistic HW: Register Impact

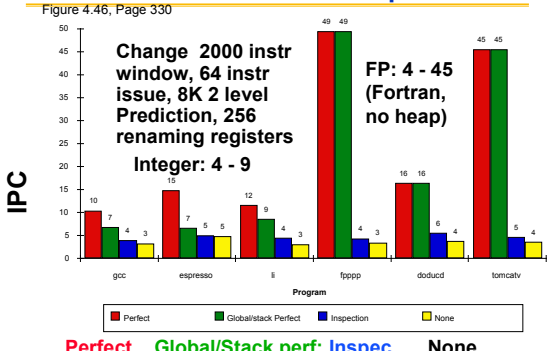
Figure 4.44, Page 328



Infinite 256 128 64 32 None

More Realistic HW: Alias Impact

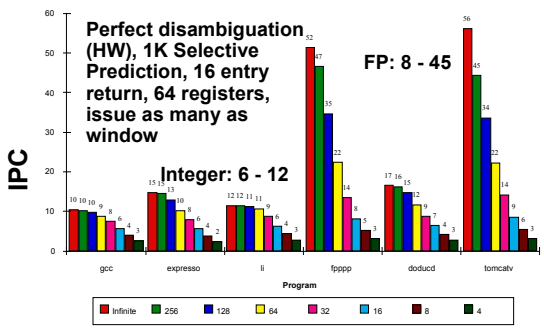
Figure 4.46, Page 330



Perfect Global/Stack perf; heap conflicts Inspec. Assem. None

Realistic HW for '9X: Window Impact

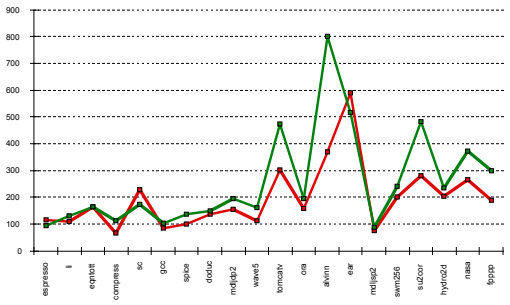
(Figure 4.48, Page 332)



Infinite 256 128 64 32 16 8 4

Braniac vs. Speed Demon(1993)

- 8-scalar IBM Power-2 @ 71.5 MHz (5 stage pipe) vs. 2-scalar Alpha @ 200 MHz (7 stage pipe)



Problems with scalar approach to ILP extraction

- Limits to conventional exploitation of ILP:
  - *pipelined clock rate*: at some point, each increase in clock rate has corresponding CPI increase (branches, other hazards)
  - *branch prediction*: branches get in the way of wide issue. They are too unpredictable.
  - *instruction fetch and decode*: at some point, its hard to fetch and decode more instructions per clock cycle
  - *register renaming*: Rename logic gets really complicate for many instructions
  - *cache hit rate*: some long-running (scientific) programs have very large data sets accessed with poor locality; others have continuous data streams (multimedia) and hence poor locality

Cost-performance of simple vs. OOO

MIPS MPUs	R5000	R10000	10k/5k
• Clock Rate	200 MHz	195 MHz	1.0x
• On-Chip Caches	32K/32K	32K/32K	1.0x
• Instructions/Cycle	1(+ FP)	4	4.0x
• Pipe stages	5	5-7	1.2x
• Model	In-order	Out-of-order	---
• Die Size (mm^2)	84	298	3.5x
– without cache, TLB	32	205	6.3x
• Development (man yr.)	60	300	5.0x
• SPECint_base95	5.7	8.8	1.6x

Summary

- DataFlow view:
  - Data triggers execution rather than instructions triggering data
- Dynamic hardware schemes can unroll loops dynamically in hardware
  - Form of limited dataflow
  - Register renaming is essential
- Explicit Renaming: more physical registers than needed by ISA.
  - Rename table: tracks current association between architectural registers and physical registers
  - Uses a translation table to perform compiler-like transformation on the fly
- Precise Interrupts:
  - Must commit things back in order
  - Reorder buffer: temporarily holds results until commit possible
  - Toss out things to achieve precise interrupt point

Summary

- Explicit Renaming: more physical registers than needed by ISA.
  - Separates *renaming* from *scheduling*
    - » Opens up lots of options for resolving RAW hazards
  - Rename table: tracks current association between architectural registers and physical registers
  - Potentially complicated rename table management
- Superscalar and VLIW: CPI < 1 (IPC > 1)
  - Dynamic issue vs. Static issue
  - More instructions issue at same time => larger hazard penalty
  - Limitation is often number of instructions that you can successfully fetch and decode per cycle ⇒ "Flynn barrier"
- Other models of parallelism: Vector processing