CS/ECE 5381/7381 Computer Architecture Spring 2023

Dr. Manikas

Computer Science

Lecture 10: Feb. 28, 2023

Project 2

- Due TODAY Tues, Feb. 28 (11:59 pm)
- MARS tool
- Assignment:
  - Translate high-level language code into MIPS code
  - Run code on MARS tool

Project 3 (7381 only)

- Due Thur., Mar 2 (11:59 pm)
- For CS/ECE 7381 students ONLY
- Additional MIPS programming assignment using MARS tool

Instruction-Level Parallelism (ILP) and Its Exploitation

(Chapter 3, Hennessy and Patterson)

Note: some course slides adopted from publisher-provided material

Outline

- 3.1 ILP Background
- 3.2 Basic Compiler Techniques for ILP
- 3.3 Branch Prediction
- 3.4 Data Hazards and Dynamic Scheduling
- 3.5 Dynamic Scheduling Algorithm
- 3.6 Hardware-Based Speculation

Scheduling

- Adding stalls is a simple way to address data dependencies and latencies
- However, adding stalls = adding cycles, which affects overall performance
- An alternate approach is to re-arrange the instruction order
  - Also called **instruction re-ordering** or **scheduling**

Instruction Dependencies

Opcode	Operands	Latency	Inputs	Outputs
L.D	F0,0(R1)	1	R1	F0
ADD.D	F4,F0,F2	2	F0, F2	F4
S.D	F4,0(R1)	0	F4, R1	(memory)
DADDUI	R1,R1,#-8	1	R1	R1
BNE	R1,R2,LOOP	0	R1,R2	(branch)

Register	Used as Input	Used as Output
R1	L.D, S.D, DADDUI, BNE	DADDUI
R2	BNE	
F0	ADD.D	L.D
F2	ADD.D	
F4	S.D	ADD.D

		Latency
L.D	F0, 0(R1)	1
ADD.D	F4, F0, F2	2
S.D	F4, 0(R1)	0
DADDUI	R1, R1, #-8	1
BNE	R1, R2, Loop	0

DEPENDENCIES: F0, F4, R1

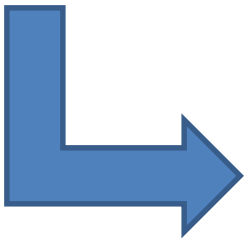
CAN WE "SCHEDULE" (RE-ORDER)

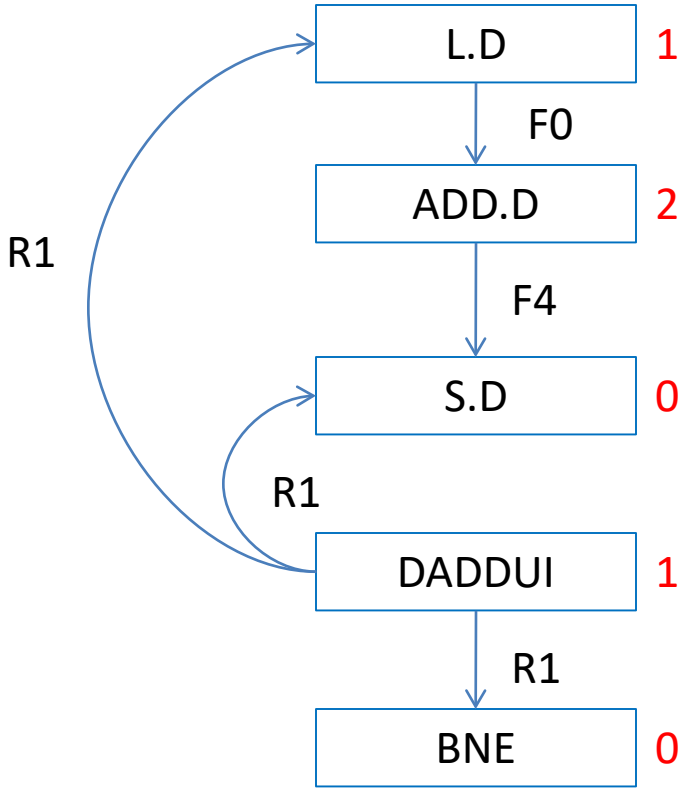
INSTRUCTIONS TO MINIMIZE STALLS?

IF WE DO THIS, PROGRAM MUST STILL WORK.

Graphical Representation of Code

```
LOOP:   L.D      F0,0(R1)
        ADD.D    F4,F0,F2
        S.D      F4, 0(R1)
        DADDUI   R1,R1,#-8
        BNE      R1,R2,LOOP
```

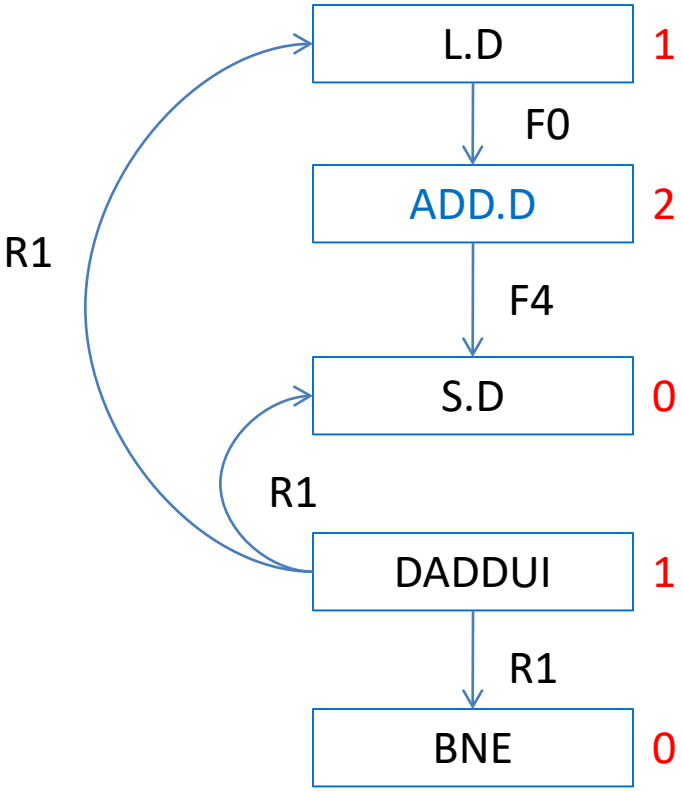




Instruction latencies in RED

x — Input-output dependency of register x

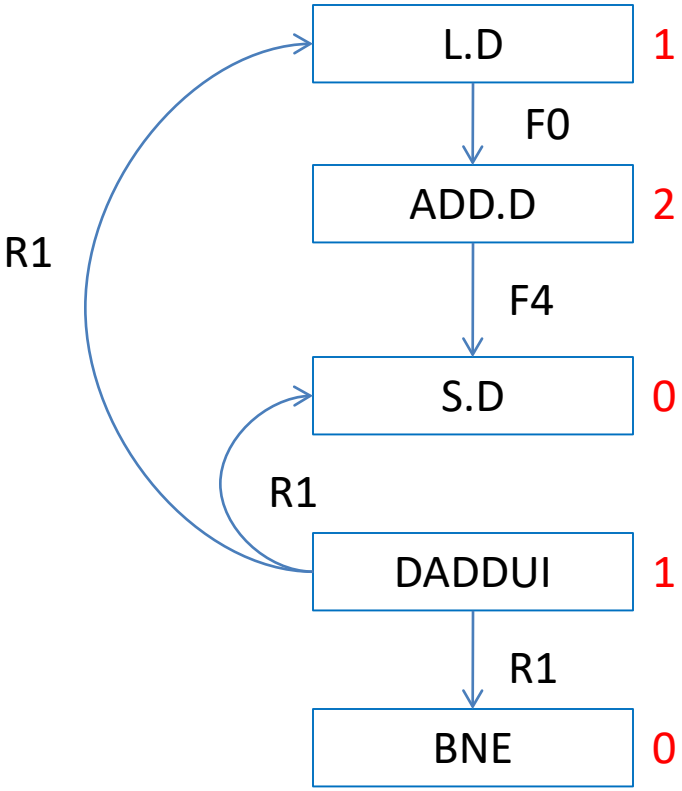
Code Modification



Need to add 2 stalls after ADD.D as before:

```
LOOP:   L.D      F0,0(R1)
        ADD.D    F4,F0,F2
        stall
        stall
        S.D      F4, 0(R1)
        DADDUI   R1,R1,#-8
        BNE      R1,R2,LOOP
```

Using Graph to Revise Code



Note: ADD.D is independent of DADDUI , so both instructions can execute in parallel – **revise code:**

```
LOOP:   L.D      F0,0(R1)
        DADDUI   R1,R1,#-8
        ADD.D    F4,F0,F2
        stall
        stall
        S.D      F4, 8(R1)
        BNE      R1,R2,LOOP
```

Instruction latencies in RED

Input-output dependency of register x

Loop Unrolling

- How to handle branch scheduling?
  - "unroll" loops
    - Multiple replications of loop body
  - Eliminates branching (and associated hazards)
  - But: increases code size (more instructions)

Example – Loop Unrolling

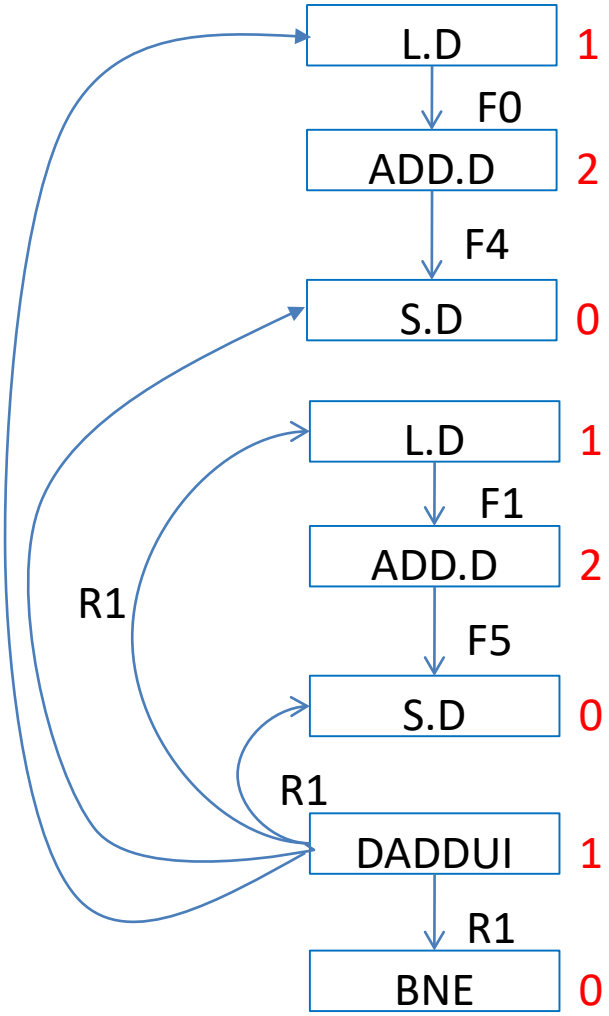
Original Code

```
LOOP:   L.D      F0,0(R1)
        ADD.D    F4,F0,F2
        S.D      F4,0(R1)
        DADDUI   R1,R1,#-8
        BNE      R1,R2,LOOP
```

Now, assume that we know the loop will be executed at least twice – unroll the loop for these two iterations

```
LOOP:   L.D      F0,0(R1)      ;first memory  location
        ADD.D    F4,F0,F2
        S.D      F4,0(R1)
        L.D      F1,-8(R1)     ;second memory location
        ADD.D    F5,F1,F2      ;use new registers (F1, F5)
        S.D      F5,-8(R1)     ;to help with scheduling
        DADDUI   R1,R1,#-16    ;decrement by 2 addresses
        BNE      R1,R2,LOOP
```

Scheduling the Unrolled Loop



Note: can do iterative parts in parallel – just make sure to consider latencies

Due to ADD.D latency

```
LOOP:   L.D      F0,0(R1)
        L.D      F1,-8(R1)
        ADD.D    F4,F0,F2
        ADD.D    F5,F1,F2
        DADDUI   R1,R1,#-16
        S.D      F4,0(R1)
        S.D      F5,-8(R1)
        BNE      R1,R2,LOOP
```

Outline

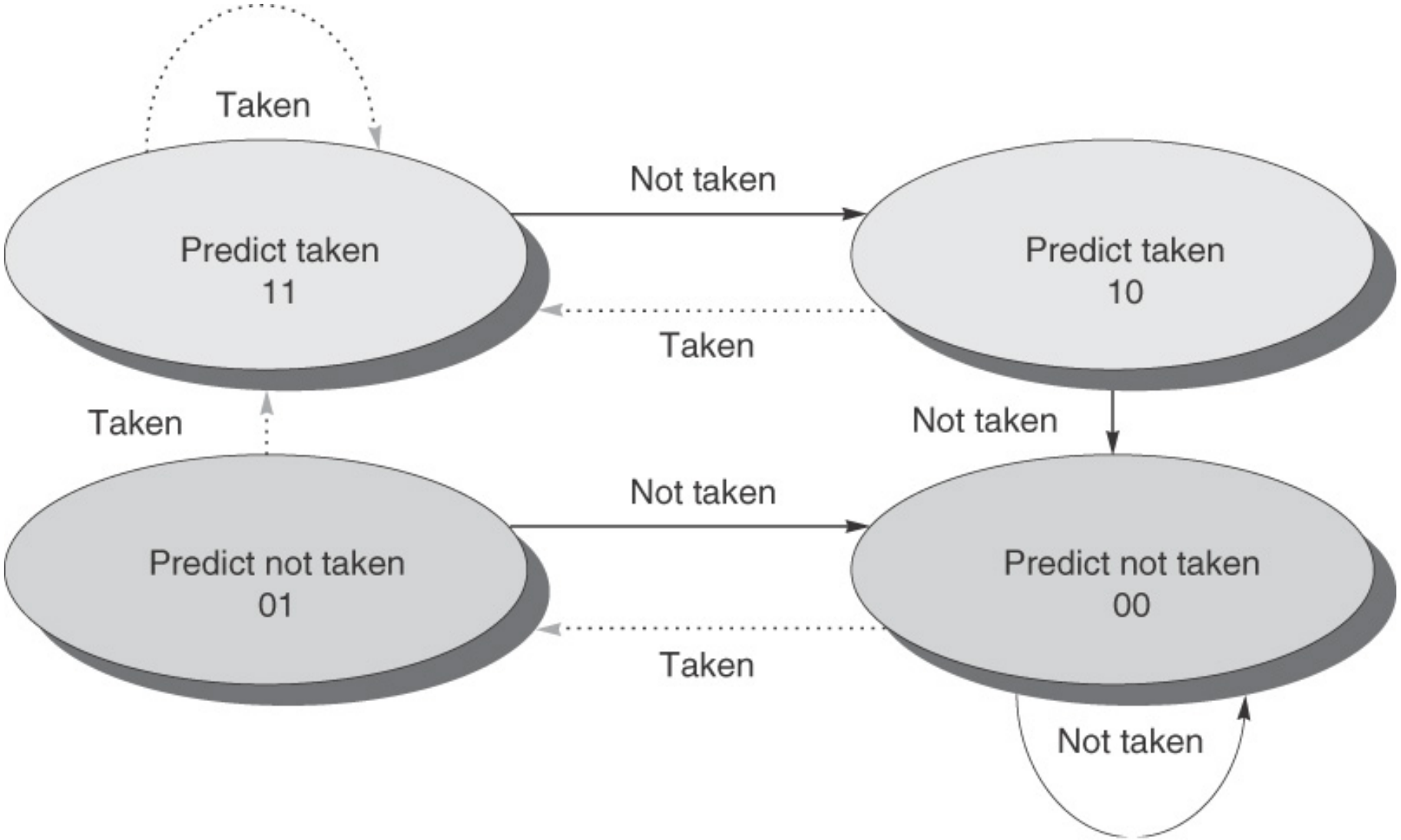
- 3.1 ILP Background
- 3.2 Basic Compiler Techniques for ILP
- 3.3 Branch Prediction
- 3.4 Data Hazards and Dynamic Scheduling
- 3.5 Dynamic Scheduling Algorithm
- 3.6 Hardware-Based Speculation

Branch Prediction

- Loop unrolling can handle simple branching instances (if we can estimate number of iterations)
  - Difficult for more complex branching (if-then-else, subroutine calls)
- Alternate approach – try to predict branch behavior

Static vs. Dynamic Prediction

- Static – done *before* execution – during compiling
  - Requires statistical estimation
  - Often inaccurate
- Dynamic – done *during* execution
  - More commonly used
  - Predictions updated based on program behavior



Correlating Branch Predictors

- 2-bit predictor: uses recent behavior of *single* branch
- What if we also examine recent behavior of *other* branches?

```
if (aa == 2)
        aa = 0;
if (bb == 2)
        bb = 0;
if (aa != bb)
{
        :
}
```

Branch B1

B2

B3

Note that branch B3 depends on outcome of B1, B2

If **both** B1 and B2 taken, then B3 not taken

There is a correlation between B3, B2, B1

Outline

- 3.1 ILP Background
- 3.2 Basic Compiler Techniques for ILP
- 3.3 Branch Prediction
- 3.4 Data Hazards and Dynamic Scheduling
- 3.5 Dynamic Scheduling Algorithm
- 3.6 Hardware-Based Speculation

Data Hazards and Dynamic Scheduling

- *Static scheduling*: compiler schedules instructions
  - Done *before* execution
- *Dynamic scheduling*: hardware schedules instructions
  - Rearranges instructions *during* execution
  - "out-of-order" execution
  - Can be used to overcome data hazards

The good news is the computer is going to do it not us we don't have to worry about it.

Dynamic Scheduling

- Rearrange order of instructions to reduce stalls while maintaining data flow
- Advantages:
  - Compiler doesn't need to have knowledge of microarchitecture
  - Handles cases where dependencies are unknown at compile time
- Disadvantage:
  - Substantial increase in hardware complexity
  - Complicates exceptions

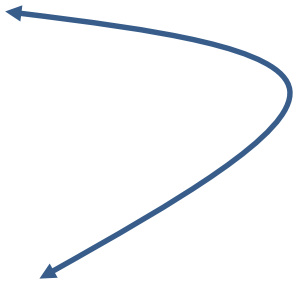
Dynamic Scheduling

- Dynamic scheduling implies:
  - Out-of-order execution
  - Out-of-order completion
- Creates the possibility for WAR and WAW hazards
- Tomasulo's Approach
  - Tracks when operands are available
  - Introduces *register renaming* in hardware
    - Minimizes WAW and WAR hazards

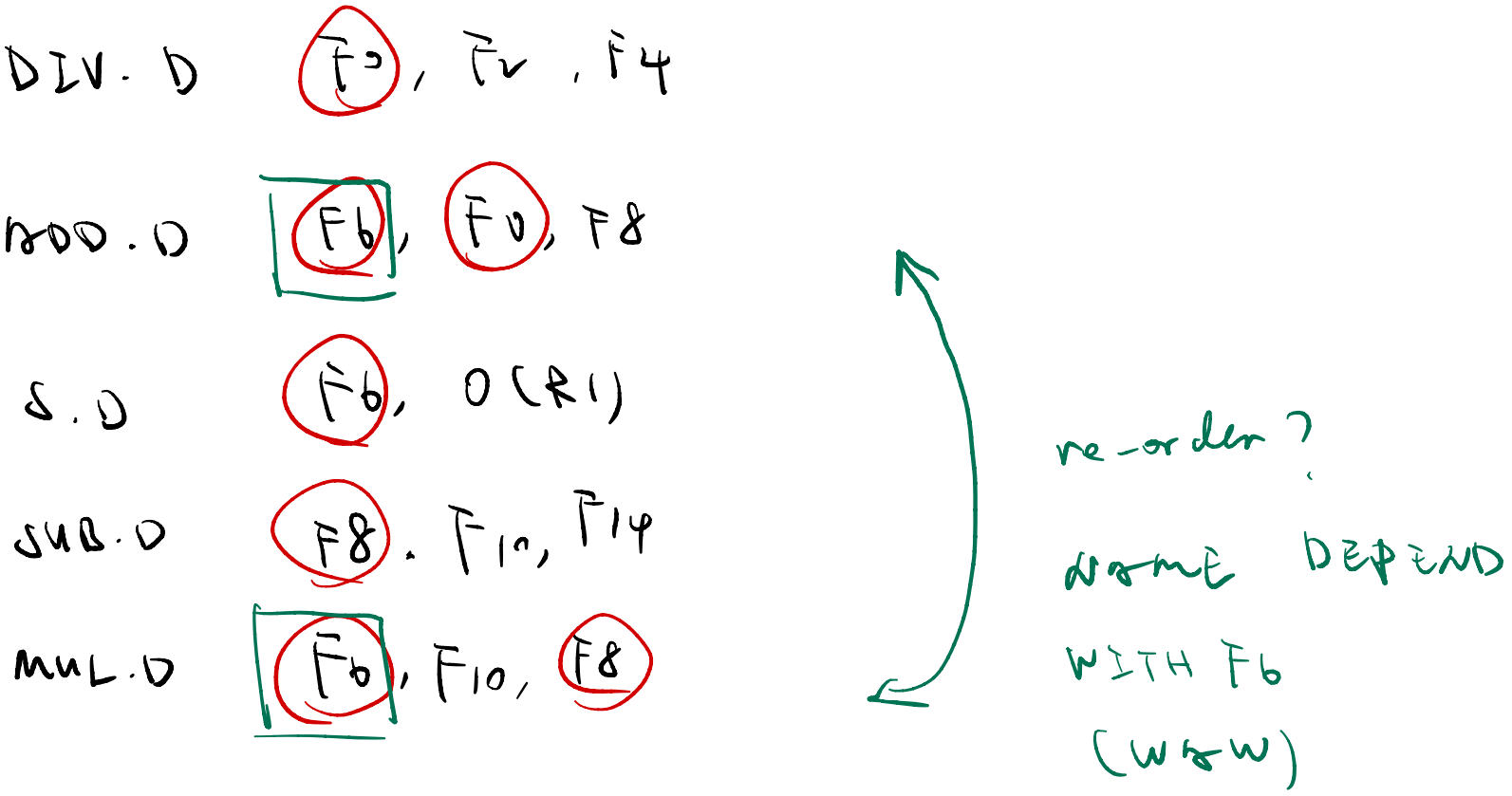
Register Renaming

- Example:

DIV.D F0,F2,F4
ADD.D F6,F0,F8
S.D F6,0(R1)
SUB.D F8,F10,F14
MUL.D F6,F10,F8



+ name dependence with F6



DEPENDENCIES: F0, F6, F8

Register Renaming

- Example:

  DIV.D F0,F2,F4

  ADD.D S,F0,F8

  S.D S,0(R1)

  SUB.D T,F10,F14

  MUL.D F6,F10,T

- Now only RAW hazards remain, which can be strictly ordered

Outline

- 3.1 ILP Background
- 3.2 Basic Compiler Techniques for ILP
- 3.3 Branch Prediction
- 3.4 Data Hazards and Dynamic Scheduling
- 3.5 Dynamic Scheduling Algorithm
- 3.6 Hardware-Based Speculation

DYNAMIC SCHEDULE:

TOMASULO'S ALGORITHM

- "RESCHEDULED INSTRUCTIONS"
  - CHANGE ORDER OF INSTRUCTION EXECUTION
  - TAKES ORIGINAL CODE IN ORDER (FROM PROGRAMMER)
  - DETERMINES EXECUTION ORDER
    - BASED ON DEPENDENCY, LATENCY

A Dynamic Algorithm: Tomasulo's

- For IBM 360/91 (before caches!)
  - $\Rightarrow$ 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, …

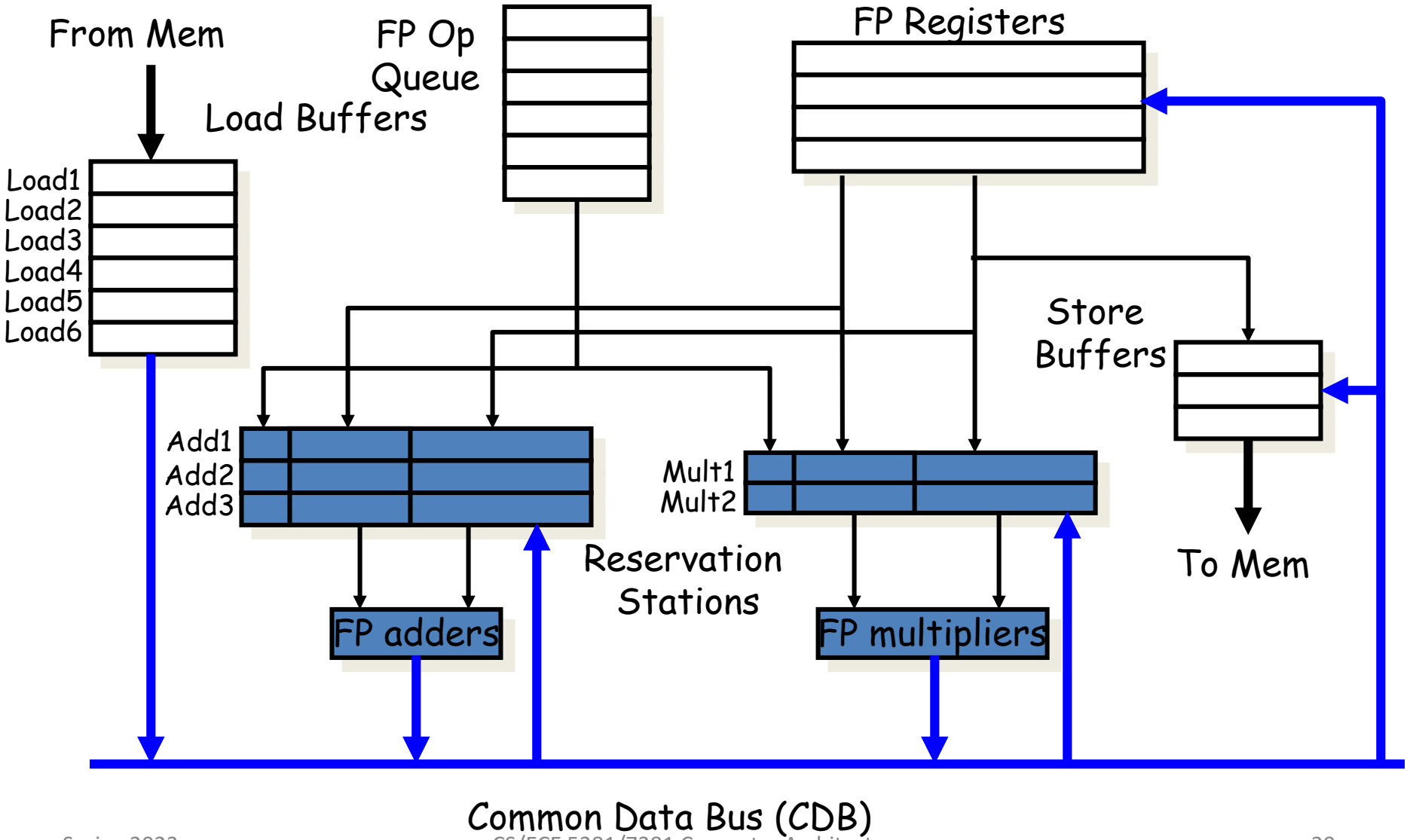
Tomasulo Algorithm

- 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

Tomasulo Algorithm

- 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

Tomasulo Organization



Reservation Station Components

Op: Operation to perform in the unit (e.g., + or –)

Vj, Vk: Value of Source operands

- Store buffers has V field, result to be stored

Qj, Qk: Reservation stations producing source registers (value to be written)

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

Busy: Indicates reservation station or FU is busy

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

Three Stages of Tomasulo Algorithm

1. Issue—get instruction from FP Op Queue

   If reservation station free (no structural hazard),
   control issues instr & sends operands (renames registers).

2. Execute—operate on operands (EX)

   When both operands ready then execute;
   if not ready, watch Common Data Bus for result

3. Write result—finish execution (WB)

   Write on Common Data Bus to all awaiting units;
   mark reservation station available

Assume we have the following MIPS code – how is this handled using Tomasulo's algorithm?

	Latency
L.D F6, 34(R2)	2
L.D F2, 45(R3)	2
MUL.D F0, F2, F4	10
SUB.D F8, F6, F2	2
DIV.D F10, F0, F6	40
ADD.D F6, F8, F2	2

		LATENCY
L.D	F6, 34(R2)	2
L.D	F2, 45(R3)	2
MULD	F0, F2, F4	10 *
SUB.D	F8, F6, F2	2
DIV.D	F10, F0, F6	40 *
ADD.D	F6, F8, F2	2

⇐ 10 STALLS CAN WE RE-SCHEDULE INSTRUCTIONS TO MINIMIZE?

DEPENDENCIES

→ But at least there's no dependency on F10. It's not affecting last instruction so we're ok there.

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2			
L.D	F2	45	R3			
MUL.D	F0	F2	F4			
SUB.D	F8	F6	F2			
DIV.D	F10	F0	F6			
ADD.D	F6	F8	F2			

Load buffers:

	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	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	F14
0	FU								

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2	**1**		
L.D	F2	45	R3			
MUL.D	F0	F2	F4			
SUB.D	F8	F6	F2			
DIV.D	F10	F0	F6			
ADD.D	F6	F8	F2			

Load buffers:

	Busy	Address
Load1	**Yes**	**34+R2**
Load2	No	
Load3	No	

memory address

CYCLE 1: issue first instruction (L.D with latency 2)

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	F14
1	FU				**Load1**				

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2	1		
L.D	F2	45	R3	**2**		
MUL.D	F0	F2	F4			
SUB.D	F8	F6	F2			
DIV.D	F10	F0	F6			
ADD.D	F6	F8	F2			

Load buffers:

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

CYCLE 2: issue second instruction (L.D with latency 2)

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	F14
2	FU		**Load2**		Load1				

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2	1	3	
L.D	F2	45	R3	2		
MUL.D	F0	F2	F4	3		
SUB.D	F8	F6	F2			
DIV.D	F10	F0	F6			
ADD.D	F6	F8	F2			

Load buffers:

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

Issue MUL.D with latency 10, can't start yet until we have value for Load2

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	MUL.D		F4	Load2	
	Mult2	No					

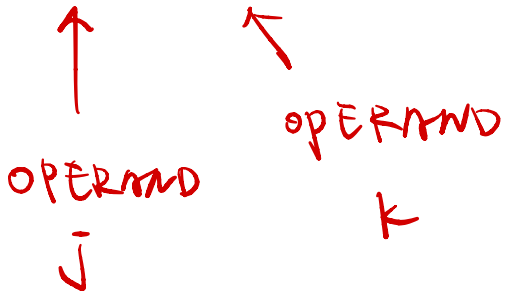
Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	F14
3	FU	Mult1	Load2		Load1				

L.D F6, 34(R2)

L.D F2, 45(R3)

MULD F0, F2, F4



V_j, V_k = NO DEPEND.

Q_j, Q_k = DATA DEPEND

MUL.D OPERAND j = F2 DEPENDS ON RESULT OF SECOND L.D

Q_j

OPERAND k = F4 NO DATA DEPENDENCY

V_k

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2	1	3	4
L.D	F2	45	R3	2	4	
MUL.D	F0	F2	F4	3		
SUB.D	F8	F6	F2	4		
DIV.D	F10	F0	F6			
ADD.D	F6	F8	F2			

Execution complete until 3

we done so we can clear it

Load buffers:

	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	SUB.D	F6			Load2
	Add2	No					
	Add3	No					
	Mult1	Yes	MUL.D		F4	Load2	
	Mult2	No					

waiting

Issue SUB.D with latency 2, also waiting for Load2

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	F14
4	FU	Mult1	Load2		M[34+R2]	Add1			

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2	1	3	4
L.D	F2	45	R3	2	4	**5**
MUL.D	F0	F2	F4	3		
SUB.D	F8	F6	F2	**4**		
DIV.D	F10	F0	F6	**5**		
ADD.D	F6	F8	F2			

Load buffers:

	Busy	Address
Load1	No	
Load2	**No**	
Load3	No	

Issue DIV.D with latency 40, waiting for Mult1

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
2	Add1	Yes	SUB.D	F6	**F2**		
	Add2	No					
	Add3	No					
10	Mult1	Yes	MUL.D	**F2**	F4		
	Mult2	**Yes**	**DIV.D**		**F6**	**Mult1**	

Timer starts down for Add1 (latency 2), Mult1 (latency 10)

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	F14
5	FU	Mult1	M[45+R3]		M[34+R2]	Add1	**Mult2**		

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2	1	3	4
L.D	F2	45	R3	2	4	5
MUL.D	F0	F2	F4	3		
SUB.D	F8	F6	F2	4		
DIV.D	F10	F0	F6	5		
ADD.D	F6	F8	F2	**6**		

Load buffers:

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Issue ADD.D with latency 2, waiting for Add1

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
1	Add1	Yes	SUB.D	F6	F2		
	Add2	**Yes**	**ADD.D**		**F2**	**Add1**	
	Add3	No					
9	Mult1	Yes	MUL.D	F2	F4		
	Mult2	Yes	DIV.D		F6	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	F14
6	FU	Mult1	M[45+R3]		**Add2**	Add1	Mult2		

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2	1	3	4
L.D	F2	45	R3	2	4	5
MUL.D	F0	F2	F4	3		
SUB.D	F8	F6	F2	4	7	
DIV.D	F10	F0	F6	5		
ADD.D	F6	F8	F2	6		

Load buffers:

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Add1 (SUB.D) completing

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
0	Add1	Yes	SUB.D	F6	F2		
	Add2	Yes	ADD.D		F2	Add1	
	Add3	No					
8	Mult1	Yes	MUL.D	F2	F4		
	Mult2	Yes	DIV.D		F6	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	F14
7	FU	Mult1	M[45+R3]		Add2	Add1	Mult2		

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2	1	3	4
L.D	F2	45	R3	2	4	5
MUL.D	F0	F2	F4	3		
SUB.D	F8	F6	F2	4	7	**8**
DIV.D	F10	F0	F6	5		
ADD.D	F6	F8	F2	6		

Load buffers:

	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	ADD.D	**F6-F2**	F2		
	Add3	No					
7	Mult1	Yes	MUL.D	F2	F4		
	Mult2	Yes	DIV.D		F6	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	F14
8	FU	Mult1	M[45+R3]		Add2	**F6-F2**	Mult2		

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2	1	3	4
L.D	F2	45	R3	2	4	5
MUL.D	F0	F2	F4	3		
SUB.D	F8	F6	F2	4	7	8
DIV.D	F10	F0	F6	5		
ADD.D	F6	F8	F2	6		

Load buffers:

	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	ADD.D	F6-F2	F2		
	Add3	No					
6	Mult1	Yes	MUL.D	F2	F4		
	Mult2	Yes	DIV.D		F6	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	F14
9	FU	Mult1	M[45+R3]		Add2	F6-F2	Mult2		

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2	1	3	4
L.D	F2	45	R3	2	4	5
MUL.D	F0	F2	F4	3		
SUB.D	F8	F6	F2	4	7	8
DIV.D	F10	F0	F6	5		
ADD.D	F6	F8	F2	6	**10**	

Load buffers:

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Add2 (ADD.D) completing

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
0	Add2	Yes	ADD.D	F6-F2	F2		
	Add3	No					
5	Mult1	Yes	MUL.D	F2	F4		
	Mult2	Yes	DIV.D		F6	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	F14
10	FU	Mult1	M[45+R3]		Add2	F6-F2	Mult2		

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2	1	3	4
L.D	F2	45	R3	2	4	5
MUL.D	F0	F2	F4	3		
SUB.D	F8	F6	F2	4	7	8
DIV.D	F10	F0	F6	5		
ADD.D	F6	F8	F2	6	10	**11**

Load buffers:

	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	MUL.D	F2	F4		
	Mult2	Yes	DIV.D		F6	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	F14
11	FU	Mult1	M[45+R3]		**F8+F2**	F6-F2	Mult2		

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2	1	3	4
L.D	F2	45	R3	2	4	5
MUL.D	F0	F2	F4	3		
SUB.D	F8	F6	F2	4	7	8
DIV.D	F10	F0	F6	5		
ADD.D	F6	F8	F2	6	10	11

Load buffers:

	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	MUL.D	F2	F4		
	Mult2	Yes	DIV.D		F6	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	F14
12	FU	Mult1	M[45+R3]		F8+F2	F6-F2	Mult2		

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2	1	3	4
L.D	F2	45	R3	2	4	5
MUL.D	F0	F2	F4	3		
SUB.D	F8	F6	F2	4	7	8
DIV.D	F10	F0	F6	5		
ADD.D	F6	F8	F2	6	10	11

Load buffers:

	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	MUL.D	F2	F4		
	Mult2	Yes	DIV.D		F6	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	F14
13	FU	Mult1	M[45+R3]		F8+F2	F6-F2	Mult2		

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2	1	3	4
L.D	F2	45	R3	2	4	5
MUL.D	F0	F2	F4	3		
SUB.D	F8	F6	F2	4	7	8
DIV.D	F10	F0	F6	5		
ADD.D	F6	F8	F2	6	10	11

Load buffers:

	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	MUL.D	F2	F4		
	Mult2	Yes	DIV.D		F6	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	F14
14	FU	Mult1	M[45+R3]		F8+F2	F6-F2	Mult2		

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2	1	3	4
L.D	F2	45	R3	2	4	5
MUL.D	F0	F2	F4	3	**15**	
SUB.D	F8	F6	F2	4	7	8
DIV.D	F10	F0	F6	5		
ADD.D	F6	F8	F2	6	10	11

Load buffers:

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Mult1 (MUL.D) completing

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	**No**					
	Add3	No					
0	Mult1	Yes	MUL.D	F2	F4		
	Mult2	Yes	DIV.D		F6	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	F14
15	FU	Mult1	M[45+R3]		F8+F2	F6-F2	Mult2		

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2	1	3	4
L.D	F2	45	R3	2	4	5
MUL.D	F0	F2	F4	3	15	16
SUB.D	F8	F6	F2	4	7	8
DIV.D	F10	F0	F6	5		
ADD.D	F6	F8	F2	6	10	11

Load buffers:

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Now Mult2 (DIV.D) can start (latency 40)

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	DIV.D	**F0**	F6		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	F14
16	FU	F2*F4	M[45+R3]		F8+F2	F6-F2	Mult2		

Faster than light computation (skip a couple of cycles)

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2	1	3	4
L.D	F2	45	R3	2	4	5
MUL.D	F0	F2	F4	3	15	**16**
SUB.D	F8	F6	F2	4	7	8
DIV.D	F10	F0	F6	5	**56**	
ADD.D	F6	F8	F2	6	10	11

Load buffers:

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Mult2 (DIV.D) completing

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	DIV.D	**F0**	F6		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	F14
56	FU	F2*F4	M[45+R3]		F8+F2	F6-F2	Mult2		

Instruction status table:

Instruction		j	k	Issue	Exec comp	Write result
L.D	F6	34	R2	1	3	4
L.D	F2	45	R3	2	4	5
MUL.D	F0	F2	F4	3	15	**16**
SUB.D	F8	F6	F2	4	7	8
DIV.D	F10	F0	F6	5	56	**57**
ADD.D	F6	F8	F2	6	10	11

Load buffers:

	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	No					

All instructions completed:

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

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	F14
57	FU	F2*F4	M[45+R3]		F8+F2	F6-F2	**F0/F6**		

Outline

- 3.1 ILP Background
- 3.2 Basic Compiler Techniques for ILP
- 3.3 Branch Prediction
- 3.4 Data Hazards and Dynamic Scheduling
- 3.5 Dynamic Scheduling Algorithm
- 3.6 Hardware-Based Speculation