Ch3--
ILP & its exploration

Ch3-1

- **Instruction Level Parallelism**
- **Basic Compiler Techniques for Exposing ILP**
- **Dynamic scheduling--Scoreboard**

3.1, 3.2, App C7

What is Instruction-Level Parallelism ?

- Instruction-level parallelism
  - The potential overlap among instructions

- Basic Block ILP is quite small
  - Basic Block: 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

Recall from Pipelining Review

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

How to exploit ILP?

- there are two main approaches:
  - Hardware-based dynamic approaches
    - Used in server and desktop processors
    - Not used as extensively in PMP processors
  - Compiler-based static approaches
    - Not as successful outside of scientific applications

Chapter 3

- ILP: Concepts and Challenge
- Basic compiler Techniques for exposing ILP
- Overcoming Data Hazards with Dynamic Scheduling
- Reducing Branch Costs with Dynamic Branch Prediction
- Hardware-based Speculation
- Exploiting ILP with Multiple Issue & Static Scheduling
- Exploiting ILP with Dynamic scheduling, Multiple issue, & speculation
- Advanced Techniques for instruction Delivery and speculation
- Multithreading: exploiting TLP improve uniprocessor throughput

Ideas to Reduce Stalls

	Technique	Reduces	Section
ChC	Forwarding and bypassing	Potential data hazard stalls	C.2,C.3
	Simple branch scheduling and prediction	Control hazard stalls	C.3
Ch3	Basic compiler pipeline scheduling	Data hazard stalls	C.2, 3.2
	Loop unrolling	Control hazard stalls	3.2
	Dynamic branch prediction	Control stalls	3.3
	Dynamic scheduling (scoreboard)	Data hazard stalls	C.7
	Dynamic scheduling (Tomasulo)	DH stalls from Anti and output dependences	3.4, 3.5
	Hardware-based speculation	**Control stalls**	3.6
	Issuing multi-instructions per cycle	Ideal CPI	3.7
	Dynamic scheduling + multiple issue +Speculation	Data and control stalls	3.8
	Multi-threading	Data parallelism	3.11
ChH	Compiler dependence analysis, software pipelining, trace schedule	Ideal CPI and data hazard stalls	H.2, H.3
	Hardware support for compiler speculation	Ideal CPI and data hazard stalls, branch hazard stalls	H.4, H.5

Instruction-Level Parallelism (ILP)

- To obtain substantial performance enhancements, we must exploit ILP across multiple basic blocks

- Simplest: loop-level parallelism to exploit parallelism among iterations of a loop
  - Vector & GPU is one way
  - If not vector, then either dynamic via branch prediction or static via loop unrolling by compiler

Data Dependence & hazard

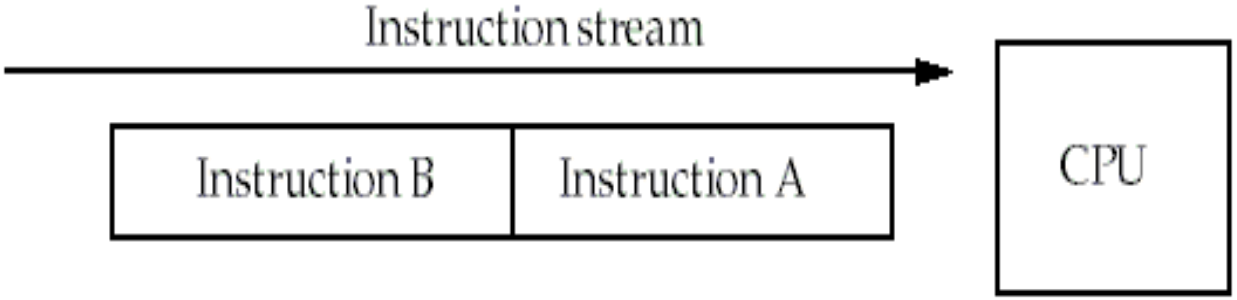
- Dependencies are a property of programs, presence of dependence indicates potential for a hazard,
- Pipeline organization determines if dependence is detected and if it causes a stall, actual hazard and length of any stall is a property of the pipeline

- Data dependence conveys:
  - Possibility of a hazard (register & memory location)
  - Order in which results must be calculated
  - Upper bound on exploitable instruction level parallelism

- Dependencies that flow through memory locations are difficult to detect

Recall: Types of data hazards

- Consider two instructions, A and B. A occurs before B.



- RAW(Read after write) true dependence
  - Instruction A writes Rx，instruction B reads Rx
- WAW(Write after write) output dependence
  - Instruction A writes Rx，instruction B writes Rx
- WAR(Write after read) anti-denpendence
  - Instruction A reads Rx，instruction B writes Rx
- Hazards are named according to the ordering that MUST be preserved by the pipeline

True Data Dependence and Hazards

- True Data Dependence:
  - $Instr_J$ is data dependent on $Instr_I$
    $Instr_J$ tries to read operand before $Instr_I$ writes it

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

  - or $Instr_J$ is data dependent on $Instr_K$ which is dependent on $Instr_I$
- Caused by a "True Dependence" (compiler term)

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

Name Dependence 1:Anti-dependence

- Name dependence: when 2 instructions use same register or memory location, called a name, but no flow of data between the instructions associated with that name;
- $Instr_J$ writes operand *before* $Instr_I$ reads it

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

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

Name Dependence 2: Output dependence

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

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

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

Name Dependence

- Two instructions use the same name but no flow of information
  - Not a true data dependence, but is a problem when reordering instructions
  - Antidependence: instruction j writes a register or memory location that instruction i reads
    - Initial ordering (i before j) must be preserved
  - Output dependence: instruction i and instruction j write the same register or memory location
    - Ordering must be preserved

ILP and Data Hazards

- HW/SW must preserve program order:
  order instructions would execute in if executed sequentially 1 at a time as determined by original source program

- HW/SW goal: exploit parallelism by preserving program order only where it affects the outcome of the program

- Instructions involved in a name dependence can execute simultaneously if name used in instructions is changed so instructions do not conflict
  - Register renaming resolves name dependence for registers
  - Either by compiler or by HW

Control Dependencies

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

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

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

Control Dependence Ignored

- Control dependence need not be preserved
  - willing to execute instructions that should not have been executed, thereby violating the control dependences, if can do so without affecting correctness of the program

- Instead, 2 properties critical to program correctness are exception behavior and data flow

Examples

- Example 1:
- add x1,x2,x3
- **beq x4,x0,L**
- sub x1,x1,x6
- **L: …**
- or x7,x1,x8

- or instruction dependent on add and sub

- **Example 2:**
- **add x1,x2,x3**
- beq x12,x0,skip
- **sub x4,x5,x6**
- add x5,x4,x9
- **skip:**
- **or** x7,x8,x9
-

- Assume x4 isn't used after skip
  - Possible to move sub before the branch

Exception Behavior

- Preserving exception behavior

  => any changes in instruction execution order must not change how exceptions are raised in program

  (=> no new exceptions)

  - Example:

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

- Problem with moving LW before BEQZ?

A short summary

- ILP
  - The potential overlap among instructions
- Reduce stalls from
  - Structural hazards
  - Data hazards
  - Control hazards
- To keep the program correctness, we should
  - Preserving Data flow
  - Preserving exception behavior

Lecture for ILP: Software approaches

- Basic Compiler Technique for Exposing ILP
  - Loop unrolling
- Static Branch Prediction
- Static multiple Issue: VLIW
- Advanced Compilor Support for Exposing and Exploiting ILP
  - Software pipelining
  - Global Code scheduling
- Hardware Support for Exposing More Parallelism at compile time
  - Conditional or Predicated instructions
  - Compiler speculation with hardware support

FP Loop: Where are the Hazards?

```
Loop:  LD     F0,0(R1)    ;F0=vector element
       ADDD   F4,F0,F2    ;add scalar from F2
       SD     0(R1),F4    ;store result
       SUBI   R1,R1,8     ;decrement pointer 8B (DW)
       BNEZ   R1,Loop     ;branch R1!=zero
       NOP                ;delayed branch slot
```

Instruction producing result	*Instruction using result*	*Execution in cycles*	*Latency in cycles*
FP ALU op	**Another FP ALU op**	**4**	**3**
FP ALU op	**Store double**	**3**	**2**
Load double	**FP ALU op**	**1**	**1**
Load double	**Store double**	**1**	**0**
Integer op	**Integer op**	**1**	**0**

Where are the stalls?

Specification for the latency

- ALU F1, -,- : IF ID FD FD FD FD WB
- ALU -, F1,-: IF ID s s s FD FD FD FD WB

- ALU: IF ID FD FD FD FD WB
- SW: IF ID s s EX DM

- LW F1, - : IF ID EX DM WB
- SW: F1, 8(R1): IF ID EX DM WB

MEM/WB.LDMR --→DM input port

Reducing stalls from scheduling in BB and delayed branch

```
Loop: LD   F0, 0(R1)
      ADDD F4, F0, F2
      SD   0(R1),  F4
      SUBI R1, R1, #8
      BNEZ R1, Loop
```

F D X M W

F D s A_1 A_2 A_3 A_4 W

F s D s s X M W

F s s D X M W

F s D X M

W

10 CC F F

```
Loop: LD   F0, 0(R1)
      SUBI R1, R1,#8
      ADDD F4, F0, F2
      BNEZ R1, Loop
      SD   +8(R1),  F4
```

F D X M W

F D X M W

F D A_1 A_2 A_3 A_4 W

F D X M W

F D s X M W

6 CC F s D X M W

Unroll Loop Four Times (straightforward way)

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

1 cycle stall

2 cycles stall

1 cycle stall

1 cycle stall(waiting for F16

1 cycle control stall

Rewrite loop to minimize stalls?

14 + 3 x (1+2) +1 +1 +1= 26 clock cycles, or 6.5 per iteration

Assumes R1 is multiple of 4

Unrolled Loop That Minimizes Stalls

```
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       SUBI   R1,R1,#32
12       SD     +16(R1),F12
13       BNEZ   R1,LOOP
14       SD     8(R1),F16        ; 8-32 = -24
```

- What assumptions made when moved code?
  - OK to move store past SUBI even though changes register
  - OK to move loads before stores: get right data?
  - When is it safe for compiler to do such changes?

14 clock cycles, or 3.5 per iteration

Ideas to Reduce Stalls

	Technique	Reduces	Section
ChC	Forwarding and bypassing	Potential data hazard stalls	C.2,C.3
	Simple branch scheduling and prediction	Control hazard stalls	C.3
Ch3	Basic compiler pipeline scheduling	Data hazard stalls	C.2, 3.2
	Loop unrolling	Control hazard stalls	3.2
	Dynamic branch prediction	Control stalls	3.3
	Dynamic scheduling (scoreboard)	Data hazard stalls	C.7
	Dynamic scheduling (Tomasulo)	DH stalls from Anti and output dependences	3.4, 3.5
	Hardware-based speculation	**Control stalls**	3.6
	Issuing multi-instructions per cycle	Ideal CPI	3.7
	Dynamic scheduling + multiple issue +Speculation	Data and control stalls	3.8
	Multi-threading	Data parallelism	3.11
ChH	Compiler dependence analysis, software pipelining, trace schedule	Ideal CPI and data hazard stalls	H.2, H.3
	Hardware support for compiler speculation	Ideal CPI and data hazard stalls, branch hazard stalls	H.4, H.5

Why Dynamic Scheduling ?

- Example1 :
  - DIVD F0,F2,F4
    ADDD F10,F0,F8
    SUBD F12,F8,F14

- Example2: Structure Hazard

```
DIVD    F2,F2,F4
ADDD    F10,F0,F8        ; FP ADDer unpipelined
ADDD    F12, F0,F4
MULD    F16, F14, F4
```

- **Problem: instruction (SUBD, MULD) stalled due to irrelevent forward instructions.**

HW Schemes: Dynamic scheduling

- Key idea: Allow instructions behind stall to proceed.
  Rearrange order of instructions to reduce stalls while maintaining data flow

- Enables out-of-order execution
  and allows out-of-order completion

- Will distinguish when an instruction *begins execution* and when it *completes execution*; between 2 times, the instruction is *in execution*

- In a dynamically scheduled pipeline, all instructions pass through issue stage in order (in-order issue)

Adv. Of Dynamic Scheduling

- Handles cases when dependences unknown at compile time
  - (e.g., because they may involve a memory reference)
- It simplifies the compiler, Compiler doesn't need to have knowledge of microarchitecture
- Allows code that compiled for one pipeline to run efficiently on a different pipeline
- Hardware speculation, a technique with significant performance advantages, that builds on dynamic scheduling

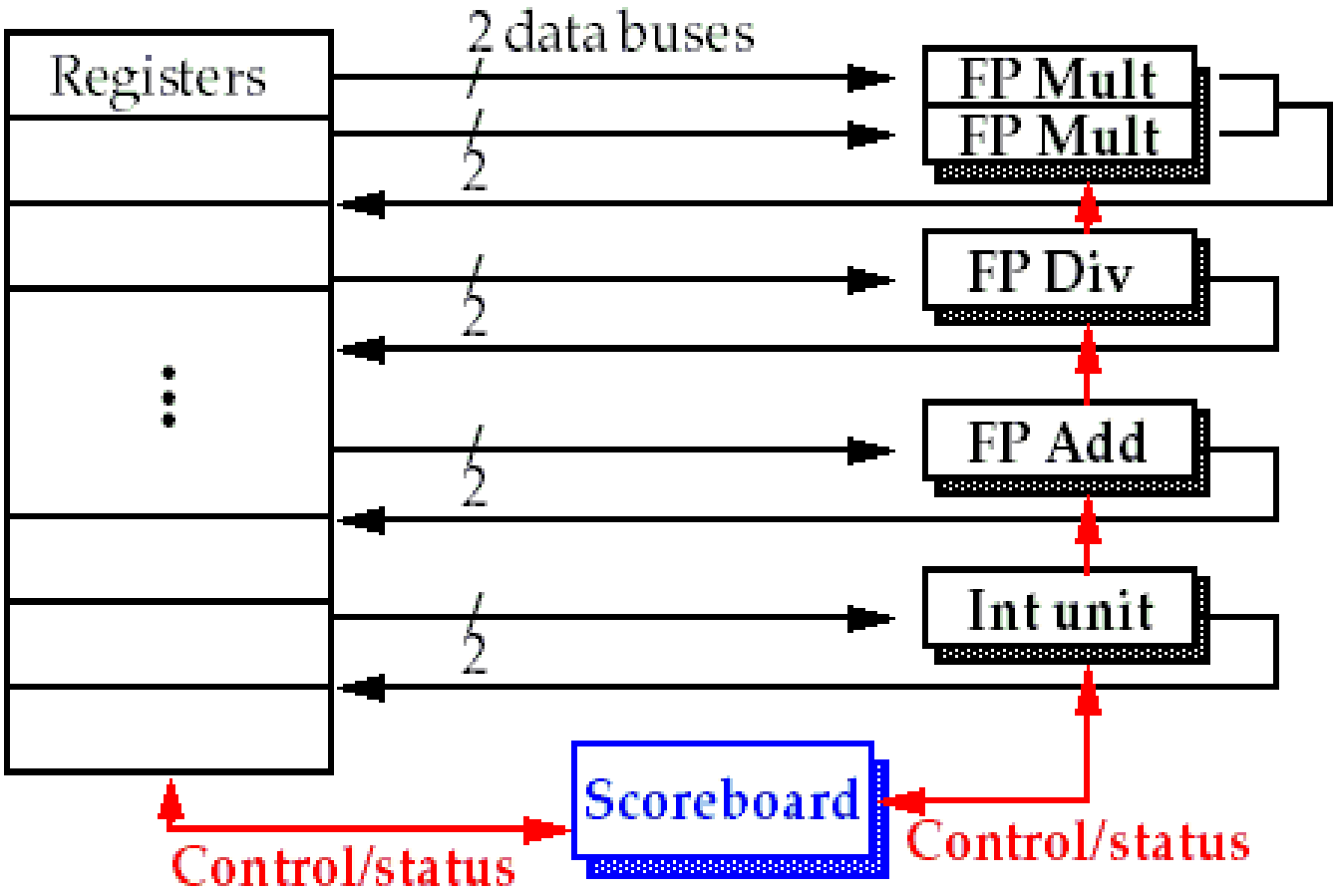
Dynamic Scheduling Step 1

- 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

Dynamic Scheduling with a Scoreboard

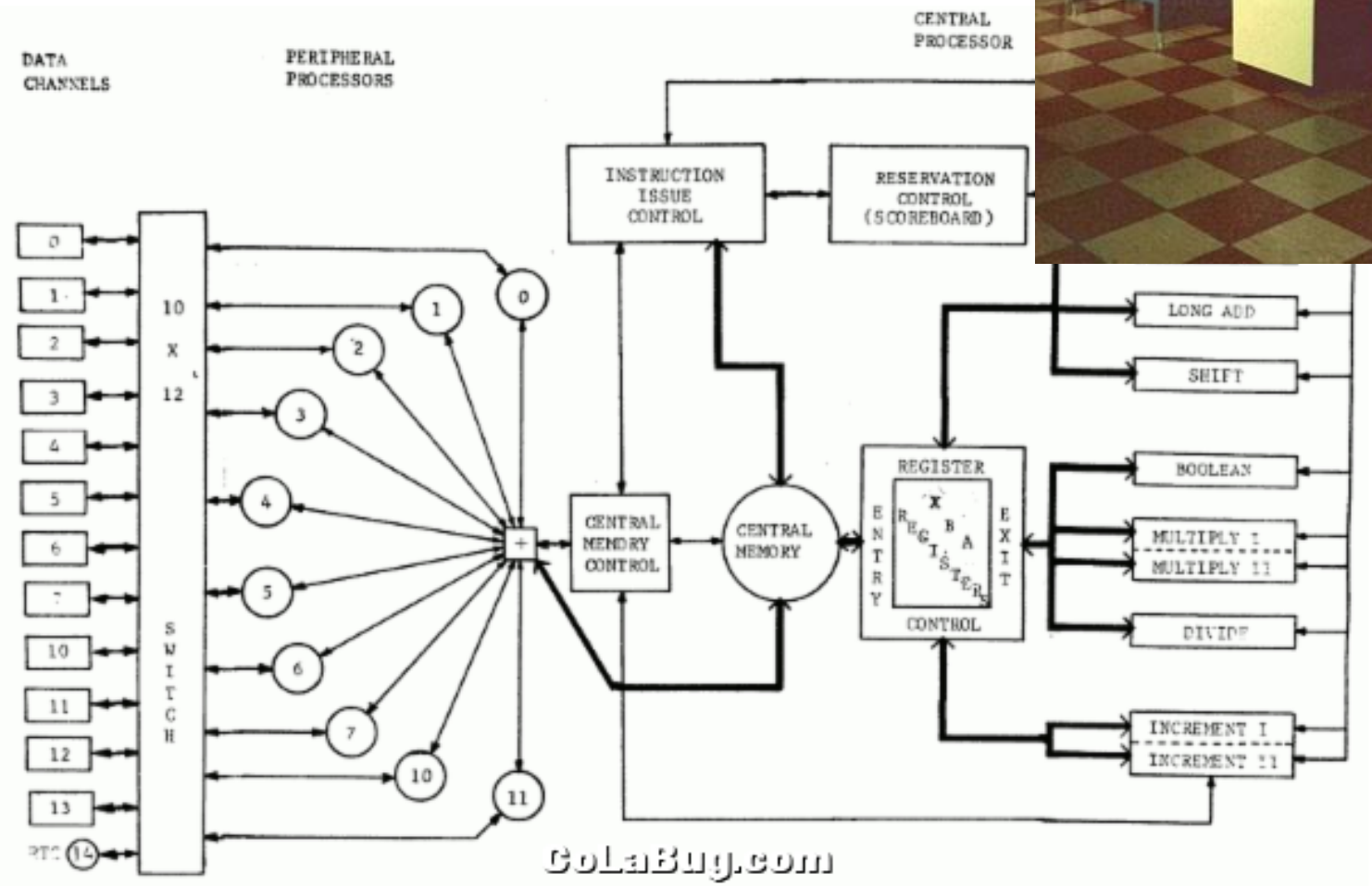
- Scoreboarding
  - Named after CDC6600 scoreboard
  - Allowing instructions to execute out of order when there are sufficient resources and no data dependences.
  - In-order issue
  - Out-of order completion
  - Executing an instruction as early as possible

Basic structure of a pipelined processor with a scoreboard



CDC6600 –First Supercomputer

top1 1964-1969



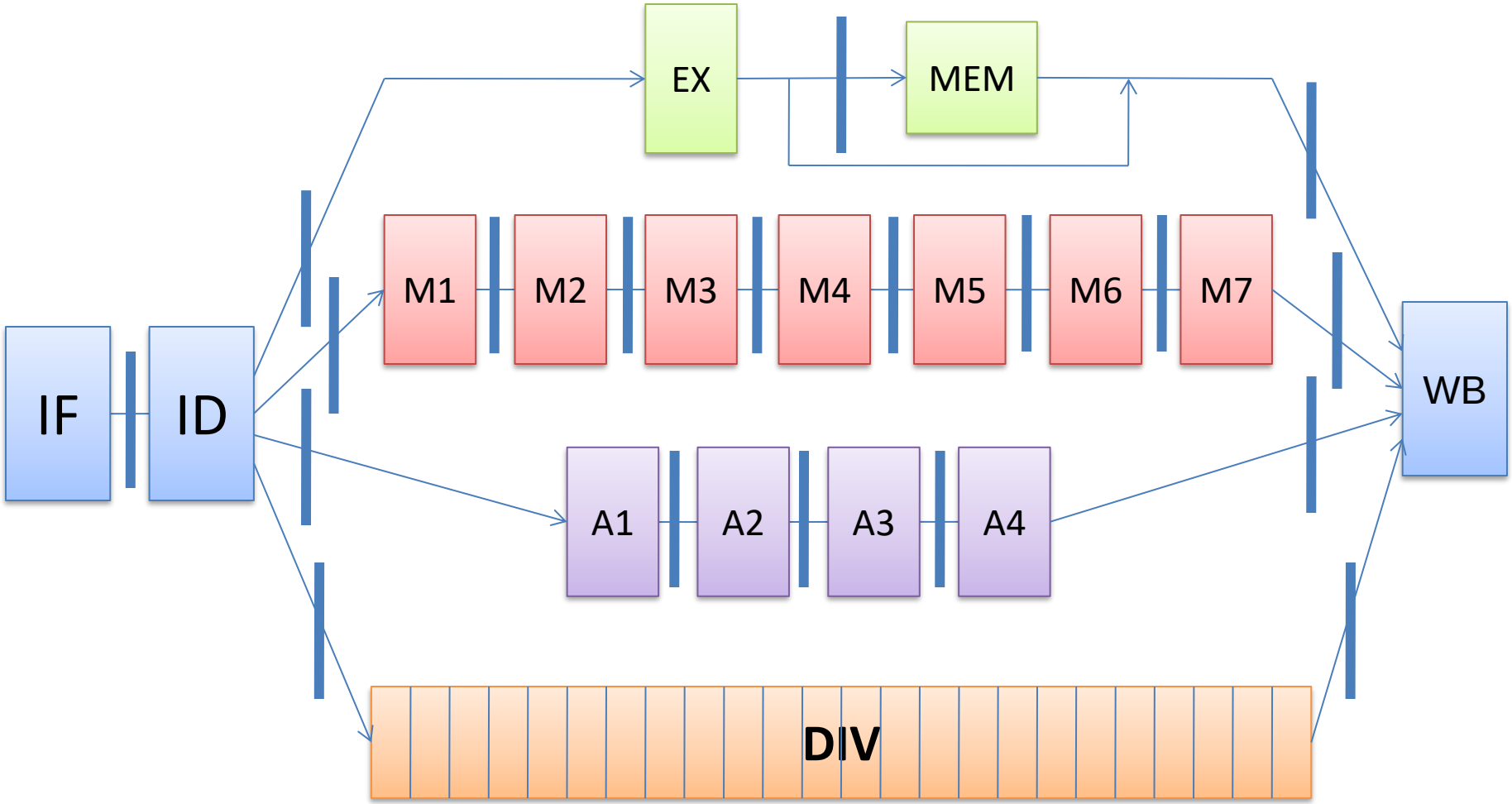
GoLaBug.com

The pipeline stages with scoreboard

- The Five stages: IF, ID, EX, MEM, WB
  - IF: the same for all instructions
  - ID: split into two stages: issue and read operands
  - EX: no change
  - MEM: omitted for only concentrating on the FP operations
  - WB: no change
- So, the stages are: IF, IS, RO, EX,WB.

• Another way to look at missing MEM ?

Pipeline supports multiple outstanding FP operations



Scoreboard Pipeline stage description

- **Issue:** a instruction is issued when
  - *The functional unit is available and*
  - *No other active instruction has the* **same** *destination register.*
  - Avoid **strutural** hazard and **WAW** hazard
- **Read Operands** (RO)
  - *The read operation is delayed until both the operands are available.*
  - *This means that no previously issued but ncompleted instruction has the operand as its destination.*
  - This resolves **RAW** hazards **dynamically**
- **Execution (EX)**
  - *Notify the scoreboard when completed so the functional unit can be reused.*
- **Write result (WB)**
  - *The scoreboard checks for* ***WAR*** *hazards and stalls the completing instruction if necessary.*

The scoreboard algorithm

- Scoreboard-takes full responsibility for instruction issue and execution
  - Create the dependence records
  - Decide when to fetch the operand
  - Decide when to enter execution
  - Decide when the result can be written into the register file
- Three data structure
  - Instruction status:
    - which of the four steps the instruction is in
  - Functional unit status: buzy,op,Fi, Fj,Fk,Qj,Qk ,Rj,Rk
  - Register result status:
    - which functional unit will write that register

Example: Instruction status

LD F6, 34(R2)
LD F2, 45(R3)
MULTD F0, F2, F4
SUBD F8, F6, F2
DIVD F10, F0, F6
ADDD F6, F8, F2

Instruction	Instruction status			
	IS	RO	EX	WB
LD	√	√	√	√
LD	√	√	√	
MULTD	√			
SUBD	√			
DIVD	√			
ADDD				

Scoreboard Example

Instruction status:

Instruction		*j*	*k*	*Issue*	Read Oprand	*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				

Function Unit Statous:

Clock cycle counter

Time	*Name*	*Busy*	*Op*	des *Fi*	*S1 Fj*	*S2 Fk*	*RS Qj*	*RS Qk*	*Rj*	*Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

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

Scoreboard Cycle 1

Instruction status:

Instruction		*j*	*k*	*Issue*	Read Oprand	*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				

Function Unit Statous:

Clock cycle counter

Time	*Name*	*Busy*	*Op*	des *Fi*	*S1 Fj*	*S2 Fk*	*RS Qj*	*RS Qk*	*Rj*	*Rk*
	Integer	Yes	load	F6	R2				Yes	
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

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

Scoreboard Cycle 2

Instruction status:

Instruction		*j*	*k*	*Issue*	Read Oprand	*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				

Function Unit Statous:

Clock cycle counter

Time	*Name*	*Busy*	*Op*	des *Fi*	*S1 Fj*	*S2 Fk*	*RS Qj*	*RS Qk*	*Rj*	*Rk*
	Integer	Yes	load	F6	R2				No	
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

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

Scoreboard Cycle 3

Instruction status:

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

Function Unit Statous:

Clock cycle counter

Time	*Name*	*Busy*	*Op*	des *Fi*	*S1 Fj*	*S2 Fk*	*RS Qj*	*RS Qk*	*Rj*	*Rk*
	Integer	Yes	load	F6	R2				No	
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

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

Scoreboard Cycle 4

Instruction status:

Instruction		*j*	*k*	*Issue*	Read Oprand	*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				

Access Data Cache

Function Unit Statous:

Clock cycle counter

Time	*Name*	*Busy*	*Op*	des *Fi*	*S1 Fj*	*S2 Fk*	*RS Qj*	*RS Qk*	*Rj*	*Rk*
	Integer	Yes	load	F6	R2				No	
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

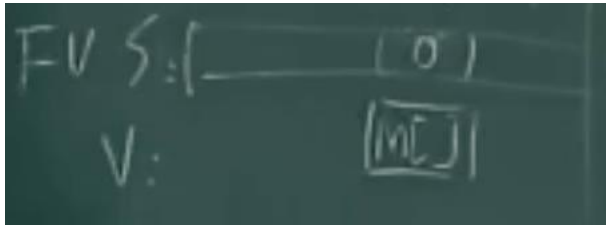
Register result status:

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

Scoreboard Cycle 5

Instruction status:

Instruction		*j*	*k*	*Issue*	Read Oprand	*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				



Function Unit Statous:

Clock cycle counter

Time	*Name*	*Busy*	*Op*	des *Vi*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*	*Rj*	*Rk*
	Integer	No	load	F6	V_{R2}				No	
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F3*
0	*FU*				M[R2+34]					

Scoreboard Cycle 6

Instruction status:

Instruction		*j*	*k*	*Issue*	Read Oprand	*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				

Function Unit Statous:

Clock cycle counter

Time	*Name*	*Busy*	*Op*	des *Vi*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*	*Rj*	*Rk*
	Integer	Yes	load	F2	R3				Yes	
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
0	*FU*		Int		M[R2+34]					

Scoreboard Cycle 7

Instruction status:

Instruction		*j*	*k*	*Issue*	Read Oprand	*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				

Function Unit Statous:

Time	*Name*	*Busy*	*Op*	des *Fi*	*S1 Fj*	*S2 Fk*	*RS Qj*	*RS Qk*	*Rj*	*Rk*
	Integer	Yes	load	F2	R3				No	
	Mult1	Yes	Mul	F0	F2	F4	Int		No	Yes
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F*[illegible]
0	*FU*	Mult1	Int		M[R2+34]					

Scoreboard Cycle 8

Instruction status:

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

Function Unit Statous:

Clock cycle counter

Time	*Name*	*Busy*	*Op*	des *Fi*	*S1 Fj*	*S2 Fk*	*RS Qj*	*RS Qk*	*Rj*	*Rk*
	Integer	Yes	load	F2	R3				No	
	Mult1	Yes	Mul	F0	F2	F4	Int		No	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		Int	Yes	No
	Divide	No								

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
0	*FU*	Mult1	Int.47		M[R2+34	Add				

Scoreboard Cycle 9

Instruction status:

Instruction		*j*	*k*	*Issue*	Read Oprand	*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				

Access Data Cache

Function Unit Statous:

Clock cycle counter

Time	*Name*	*Busy*	*Op*	des *Fi*	*S1 Fj*	*S2 Fk*	*RS Qj*	*RS Qk*	*Rj*	*Rk*
	Integer	Yes	load	F2	R3				No	
	Mult1	Yes	Mul	F0	F2	F4	Int		No	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		Int	Yes	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
0	*FU*	Mult1	Int		M[R2+34	Add	Div			

Example: Function unit status and Register status

Name	Functional unit status								
	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	Yes	Load	F2	F3				No	
Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
Mult2	No								
Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

	Register result status								
	F0	F2	F4	F6	F8	F10	F12	…	F30
FU	Mult1	Integer			Add	Divide		…	

Scoreboard Cycle 10

Instruction status:

Instruction		*j*	*k*	*Issue*	Read Oprand	*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				

Function Unit Statous:

Clock cycle counter

Time	*Name*	*Busy*	*Op*	des *Fi*	*S1 Fj*	*S2 Fk*	*RS Qj*	*RS Qk*	*Rj*	*Rk*
	Integer	No	load	F2	R3				No	
	Mult1	Yes	Mul	F0	F2	F4	~~Int~~		Yes	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		~~Int~~	Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
0	*FU*	Mult1	M[R3+45]		M[R2+34	Add	Div			

Scoreboard Cycle 11

Instruction status:

Instruction		*j*	*k*	*Issue*	Read Oprand	*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				

Function Unit Statous:

Clock cycle counter

Time	*Name*	*Busy*	*Op*	des *Fi*	*S1 Fj*	*S2 Fk*	*RS Qj*	*RS Qk*	*Rj*	*Rk*
	Integer	No	load	F2	R3				No	
	Mult1	Yes	Mul	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
0	*FU*	Mult1	M[R3+45]		M[R2+34	Add	Div			

Scoreboard Cycle 12

Instruction status:

Instruction		*j*	*k*	*Issue*	Read Oprand	*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				

Assume Mul takes 7 cycles

Assume Sub takes 3 cycles

Function Unit Statous:

Time	*Name*	*Busy*	*Op*	des *Fi*	*S1 Fj*	*S2 Fk*	*RS Qj*	*RS Qk*	*Rj*	*Rk*
	Integer	No	load	F2	R3				No	
	Mult1	Yes	Mul	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
0	*FU*	Mult1	M[R3+45]		M[R2+34	Add	Div			

Scoreboard Cycle 15

Instruction status:

Instruction		*j*	*k*	*Issue*	Read Oprand	*Exec Comp*	*Write Result*	
LD	F6	34+	R2					
LD	F2	45+	R3					
MULTD	F0	F2	F4			√		Assume Mul takes 7 cycles
SUBD	F8	F6	F2				√	Assume Sub takes 3 cycles
DIVD	F10	F0	F6	√				
ADDD	F6	F8	F2					

Function Unit Statous:

Time	*Name*	*Busy*	*Op*	des *Fi*	*S1 Fj*	*S2 Fk*	*RS Qj*	*RS Qk*	*Rj*	*Rk*
	Integer	No	load	F2	R3				No	
	Mult1	Yes	Mul	F0	F2	F4			No	No
	Mult2	No								
	Add	No	Sub	F8	F6	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
0	*FU*	Mult1	M[R3+45]		M[R2+34	V-	Div			

Scoreboard Cycle 16

Instruction status:

Instruction		*j*	*k*	*Issue*	Read Oprand	*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	√			

Assume Mul takes 7 cycles

Function Unit Statous:

Time	*Name*	*Busy*	*Op*	des *Fi*	*S1 Fj*	*S2 Fk*	*RS Qj*	*RS Qk*	*Rj*	*Rk*
	Integer	No	load	F2	R3				No	
	Mult1	Yes	Mul	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	ADD	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
0	*FU*	Mult1	M[R3+45]		Add	V-	Div			

Scoreboard Cycle 17

Instruction status:

Instruction		*j*	*k*	*Issue*	Read Oprand	*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		√		

Assume Mul takes 7 cycles

Function Unit Statous:

Time	*Name*	*Busy*	*Op*	des *Fi*	*S1 Fj*	*S2 Fk*	*RS Qj*	*RS Qk*	*Rj*	*Rk*
	Integer	No	load	F2	R3				No	
	Mult1	Yes	Mul	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	ADD	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
0	*FU*	Mult1	M[R3+45]		Add	V-	Div			

Scoreboard Cycle 18

Instruction status:

Instruction		*j*	*k*	*Issue*	Read Oprand	*Exec Comp*	*Write Result*	
LD	F6	34+	R2					
LD	F2	45+	R3					
MULTD	F0	F2	F4			√		Last cycle of Mul
SUBD	F8	F6	F2					
DIVD	F10	F0	F6	√				
ADDD	F6	F8	F2			√		

Function Unit Statous:

Time	*Name*	*Busy*	*Op*	des *Fi*	*S1 Fj*	*S2 Fk*	*RS Qj*	*RS Qk*	*Rj*	*Rk*
	Integer	No	load	F2	R3				No	
	Mult1	Yes	Mul	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	ADD	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
0	*FU*	Mult1	M[R3+45]		Add	V-	Div			

Scoreboard Cycle 19

Instruction status:

Instruction		*j*	*k*	*Issue*	Read Oprand	*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			√	

Function Unit Statous:

Time	*Name*	*Busy*	*Op*	des *Fi*	*S1 Fj*	*S2 Fk*	*RS Qj*	*RS Qk*	*Rj*	*Rk*
	Integer	No	load	F2	R3				No	
	Mult1	No	Mul	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	ADD	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	~~Mult1~~		Yes	Yes

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
0	*FU*	V*	M[R3+45]		Add	V-	Div			

Examples: Dynamic Scheduling

- Dynamic scheduling implies:
  - Out-of-order execution
  - Out-of-order completion
- Example 1:

```
fdiv.d f0,f2,f4
fadd.d f10,f0,f8
fsub.d f12,f8,f14
```

  - fsub.d is not dependent, issue before fadd.d
- Example 2:

```
fdiv.d f0,f2,f4
fmul.d f6,f0,f8
fadd.d f0,f10,f14
```

  - fadd.d is not dependent, but the antidependence makes it impossible to issue earlier without register renaming

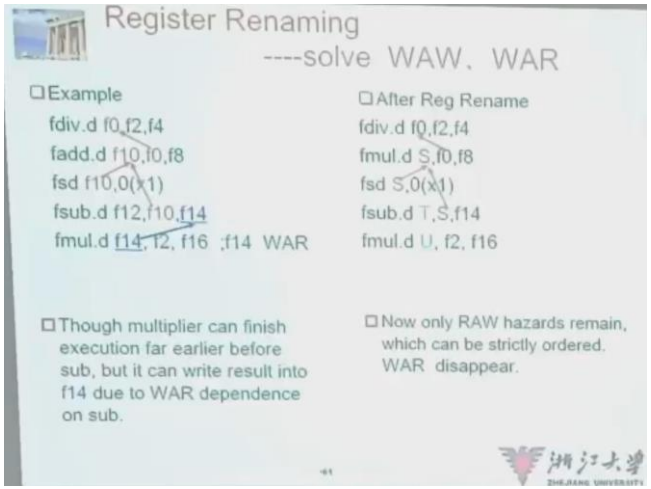
Register Renaming

- Example 3:

fdiv.d f0,f2,f4
fadd.d f6,f0,f8
fsd f6,0(x1)
fsub.d f8,f10,f14
fmul.d f6,f10,f8

•antidependence

•antidependence



Register Renaming

- Example 3:

```
fdiv.d f0,f2,f4
fadd.d S,f0,f8
fsd S,0(x1)
fsub.d T,f10,f14
fmul.d f6,f10,T
```

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

Limitations of Scoreboard-1

- ILP
  - If we can't find independent instructions to execute, scoreboard (or any dynamic scheduling scheme for that matter) helps very little.
- Size of the "issued" queue
  - This determines how far ahead the CPU can look for instructions to execute in parallel.
  - It's called the window.
  - For now, we assume that a window can **not** span a branch.
  - In other words, the window includes instructions only within basic blocks.

Limitations of Scoreboard-2

- Number, types, and speed of the functional units
  - This determines how often a structural hazard results in stall.
- The presence of anti-dependences and output dependences
  - **WAR** and **WAW** hazards limit the scoreboard more than **RAW** hazards, lead to WAR and WAW stalls.
  - **RAW** hazards are problems for any technique.
  - But **WAR** and **WAW** hazards can be solved in ways other than scoreboards.

Scoreboard vs. Tomasulo

- 特点
  - Multiple multiplier, etc. Funcs
  - Issue in order, Complete OOO
  - IF→ Issue, Ro
  - 4 stages pipeline
  - Scoreboare centralized control
- 缺点
  - Stall when WAW, WAR

- Fewer Func, unpipelined
- Issue in order, Complete OOO
- FP op. queue, Reservation station, LD/ST buffer, CDB
- Reg. Rename→No WAW, WAR
- Reduce structural hazard
- RAW detection decentralized—reservatio
- CDB→ forwarding path