
CPE 631 Lecture 09: Instruction Level Parallelism and Its Dynamic Exploitation

Aleksandar Milenković, milenka@ece.uah.edu
Electrical and Computer Engineering
University of Alabama in Huntsville

CPE
631
©AM

Outline

- Instruction Level Parallelism (ILP)
- Recap: Data Dependencies
- Extended MIPS Pipeline and Hazards
- Dynamic scheduling with a scoreboard

ILP: Concepts and Challenges

- **ILP (Instruction Level Parallelism) – overlap execution of unrelated instructions**
- Techniques that increase amount of parallelism exploited among instructions
 - reduce impact of data and control hazards
 - increase processor ability to exploit parallelism
- **Pipeline CPI = Ideal pipeline CPI + Structural stalls + RAW stalls + WAR stalls + WAW stalls + Control stalls**
 - Reducing each of the terms of the right-hand side minimize CPI and thus increase instruction throughput

Two approaches to exploit parallelism

- **Dynamic techniques**
 - largely depend on hardware to locate the parallelism
- **Static techniques**
 - rely on software

Techniques to exploit parallelism

Technique (Section in the textbook)	Reduces
Forwarding and bypassing (Section A.2)	Data hazard (DH) stalls
Delayed branches (A.2)	Control hazard stalls
Basic dynamic scheduling (A.8)	DH stalls (RAW)
Dynamic scheduling with register renaming (3.2)	WAR and WAW stalls
Dynamic branch prediction (3.4)	CH stalls
Issuing multiple instruction per cycle (3.6)	Ideal CPI
Speculation (3.7)	Data and control stalls
Dynamic memory disambiguation (3.2, 3.7)	RAW stalls w. memory
Loop Unrolling (4.1)	CH stalls
Basic compiler pipeline scheduling (A.2, 4.1)	DH stalls
Compiler dependence analysis (4.4)	Ideal CPI, DH stalls
Software pipelining and trace scheduling (4.3)	Ideal CPI and DH stalls
Compiler speculation (4.4)	Ideal CPI, and D/CH stalls

14/02/2005

UAH-CPE631

5

Where to look for ILP?

- Amount of parallelism available within a basic block
 - BB: straight line code sequence of instructions with no branches in except to the entry, and no branches out except at the exit
 - Example: Gcc (Gnu C Compiler): 17% control transfer
 - 5 or 6 instructions + 1 branch
 - Dependencies => amount of parallelism in a basic block is likely to be much less than 5
=> look beyond single block to get more instruction level parallelism
- Simplest and most common way to increase amount of parallelism among instruction is to exploit parallelism among iterations of a loop => Loop Level Parallelism

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

- Vector Processing: see Appendix G

14/02/2005

UAH-CPE631

6

Definition: Data Dependencies

- Data dependence: instruction j is data dependent on instruction i if either of the following holds
 - Instruction i produces a result used by instruction j, or
 - Instruction j is data dependent on instruction k, and instruction k is data dependent on instruction i
- If dependent, cannot execute in parallel
- Try to schedule to avoid hazards
- Easy to determine for registers (fixed names)
- Hard for memory (“**memory disambiguation**”):
 - Does $100(R4) = 20(R6)$?
 - From different loop iterations, does $20(R6) = 20(R6)$?

Examples of Data Dependencies

Loop:	LD.D	F0, 0(R1)	; F0 = array element
	ADD.D	F4, F0, F2	; add scalar in F2
	SD.D	0(R1), F4	; store result and
	DADUI	R1, R1, #-8	; decrement pointer
	BNE	R1, R2, Loop	; branch if R1!=R2

Definition: Name Dependencies

- Two instructions use same name (register or memory location) but don't exchange data
 - Antidependence (WAR if a hazard for HW)
Instruction j writes a register or memory location that instruction i reads from and instruction i is executed first
 - Output dependence (WAW if a hazard for HW)
Instruction i and instruction j write the same register or memory location; ordering between instructions must be preserved. If dependent, can't execute in parallel
- Renaming to remove data dependencies
- Again Name Dependencies are Hard for Memory Accesses
 - Does $100(R4) = 20(R6)$?
 - From different loop iterations, does $20(R6) = 20(R6)$?

Where are the name dependencies?

```

1 Loop: L.D    F0, 0(R1)
2      ADD.D  F4, F0, F2
3      S.D    0(R1), F4      ;drop DSUBUI & BNEZ
4      L.D    F0, -8(R1)
5      ADD.D  F4, F0, F2
6      S.D    -8(R1), F4     ;drop DSUBUI & BNEZ
7      L.D    F0, -16(R1)
8      ADD.D  F4, F0, F2
9      S.D    -16(R1), F4    ;drop DSUBUI & BNEZ
10     L.D    F0, -24(R1)
11     ADD.D  F4, F0, F2
12     S.D    -24(R1), F4
13     SUBUI  R1, R1, #32    ;alter to 4*8
14     BNEZ  R1, LOOP
15     NOP

```

How can remove them?

Where are the name dependencies?

```

1 Loop: L.D    F0, 0(R1)
2      ADD.D  F4, F0, F2
3      S.D    0(R1), F4      ;drop DSUBUI & BNEZ
4      L.D    F6, -8(R1)
5      ADD.D  F8, F6, F2
6      S.D    -8(R1), F8     ;drop DSUBUI & BNEZ
7      L.D    F10, -16(R1)
8      ADD.D  F12, F10, F2
9      S.D    -16(R1), F12  ;drop DSUBUI & BNEZ
10     L.D    F14, -24(R1)
11     ADD.D  F16, F14, F2
12     S.D    -24(R1), F16
13     DSUBUI R1, R1, #32    ;alter to 4*8
14     BNEZ   R1, LOOP
15     NOP

```

The Original "register renaming"

Definition: Control Dependencies

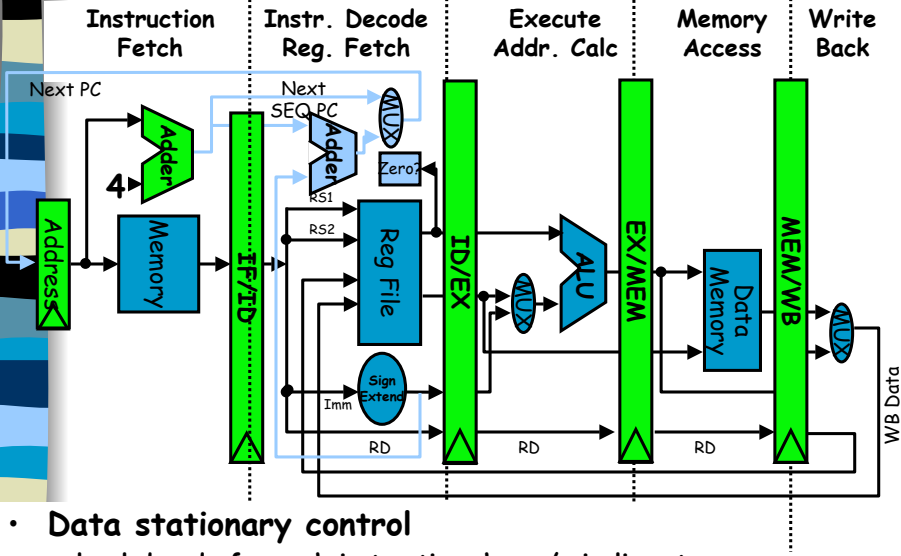
- Example
 - if p1 {S1;}; if p2 {S2;};
 - S1 is control dependent on p1 and
 - S2 is control dependent on p2 but not on p1
- Two constraints on control dependences:
 - An instruction that is control dep. on a branch cannot be moved before the branch, so that its execution is no longer controlled by the branch
 - An instruction that is not control dep. on a branch cannot be moved to after the branch so that its execution is controlled by the branch

```

DADDU R5, R6, R7
ADD R1, R2, R3
BEQZ R4, L
SUB R1, R5, R6
L:   OR R7, R1, R8

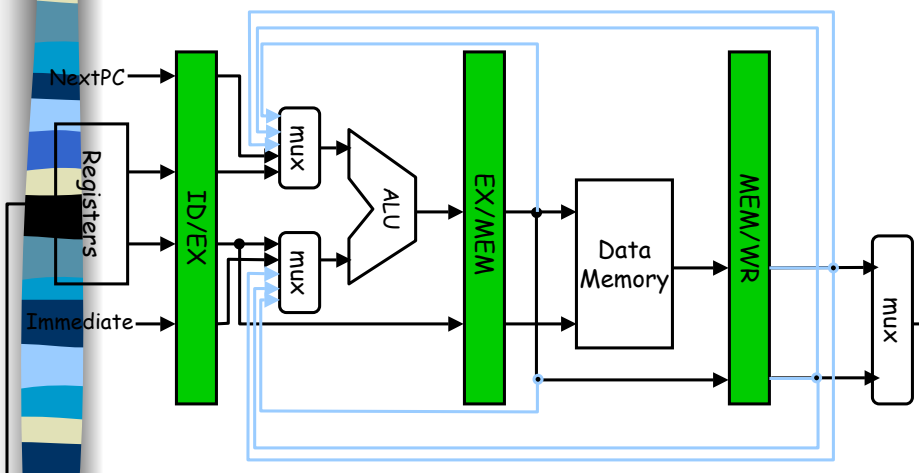
```

Pipelined MIPS Datapath



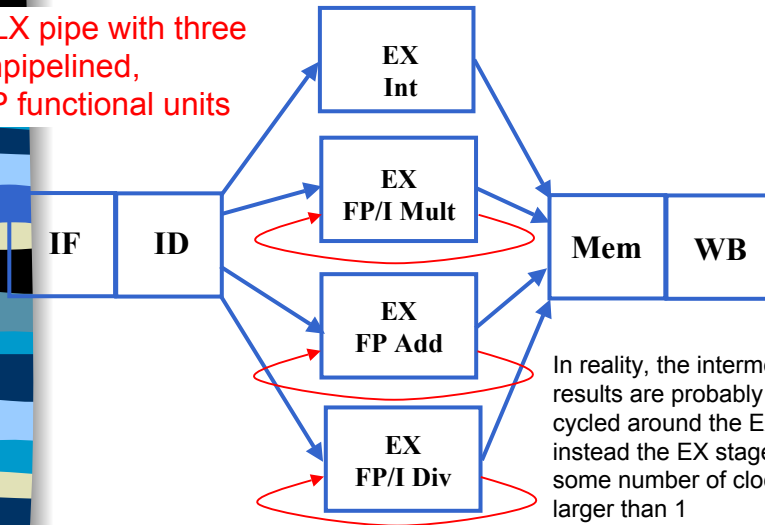
- **Data stationary control**
 - local decode for each instruction phase / pipeline stage

HW Change for Forwarding



Extended MIPS Pipeline

DLX pipe with three unpipelined, FP functional units

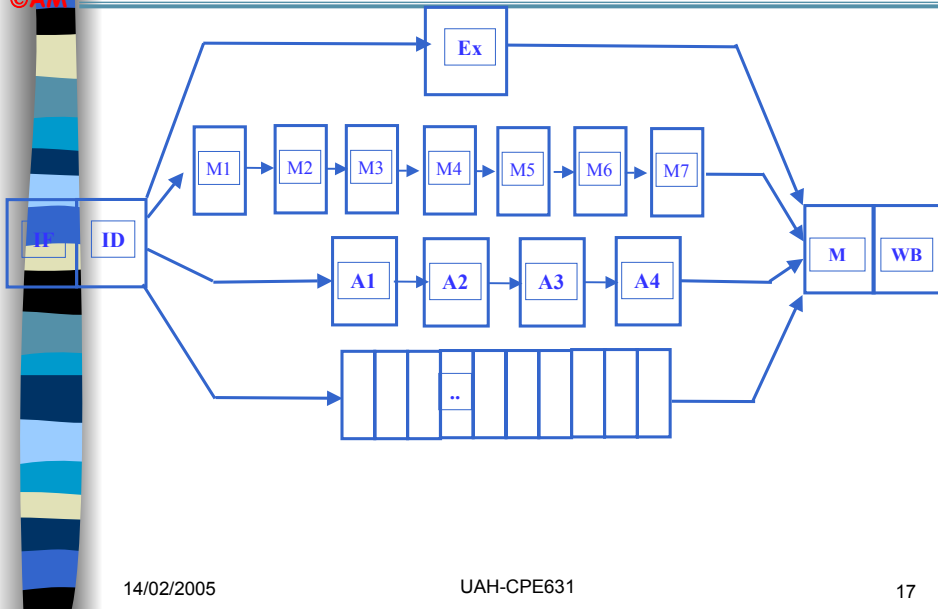


Extended MIPS Pipeline (cont'd)

- **Initiation or repeat interval:** number of clock cycles that must elapse between issuing two operations
- **Latency:** the number of intervening clock cycles between an instruction that produces a result and an instruction that uses the result

Functional unit	Latency	Initiation interval
Integer ALU	0	1
Data Memory	1	1
FP Add	3	1
FP/Integer Multiply	6	1
FP/Integer Divide	24	25

Extended MIPS Pipeline (cont'd)



Extended MIPS Pipeline (cont'd)

- Multiple outstanding FP operations
 - FP/I Adder and Multiplier are fully pipelined
 - FP/I Divider is not pipelined
- Pipeline timing for independent operations

MUL.D	IF	ID	M1	M2	M3	M4	M5	M6	M7	Mem	WB
ADD.D		IF	ID	A1	A2	A3	A4	Mem	WB		
L.D				IF	ID	Ex	Mem	WB			
S.D					IF	ID	Ex	Mem	WB		

Hazards and Forwarding in Longer Pipes

- Structural hazard:
 - divide unit is not fully pipelined
 - detect it and stall the instruction
- Structural hazard: number of register writes can be larger than one due to varying running times
- WAW hazards are possible
- Exceptions!
 - instructions can complete in different order than they were issued
- RAW hazards will be more frequent

Examples

- Stalls arising from RAW hazards

L.D F4, 0(R2)	IF	ID	EX	Mem	WB														
MUL.D F0, F4, F6		IF	ID	stall	M1	M2	M3	M4	M5	M6	M7	Mem	WB						
ADD.D F2, F0, F8			IF	stall	ID	stall	stall	stall	stall	stall	stall	A1	A2	A3	A4	Mem	WB		
S.D 0(R2), F2					IF	stall	stall	stall	stall	stall	stall	ID	EX	stall	stall	stall	Mem		

- Three instructions that want to perform a write back to the FP register file simultaneously

MUL.D F0, F4, F6		IF	ID	M1	M2	M3	M4	M5	M6	M7	Mem	WB							
...			IF	ID	EX	Mem	WB												
...				IF	ID	EX	Mem	WB											
ADD.D F2, F4, F6					IF	ID	A1	A2	A3	A4	Mem	WB							
...						IF	ID	EX	Mem	WB									
...							IF	ID	EX	Mem	WB								
L.D F2, 0(R2)								IF	ID	EX	Mem	WB							

Solving Register Write Conflicts

- First approach: track the use of the write port in the ID stage and stall an instruction before it issues
 - use a shift register that indicates when already issued instructions will use the register file
 - if there is a conflict with an already issued instruction, stall the instruction for one clock cycle
 - on each clock cycle the reservation register is shifted one bit
- Alternative approach: stall a conflicting instruction when it tries to enter MEM or WB stage
 - we can stall either instruction
 - e.g. give priority to the unit with the longest latency
 - Pros: does not require to detect the conflict until the entrance of MEM or WB stage
 - Cons: complicates pipeline control; stalls now can arise from two different places

WAW Hazards

	IF	ID	EX	Mem	WB					
ADD.D F2, F4, F6		IF	ID	A1	A2	A3	A4	Mem	WB	
			IF	ID	EX	Mem	WB			
L.D F2, 0(R2)				IF	ID	EX	Mem	WB		

- Result of ADD.D is overwritten without any instruction ever using it
 - WAWs occur when useless instruction is executed
 - still, we must detect them and provide correct execution
- Why?

```

BNEZ R1, foo
DIV.D F0, F2, F4 ; delay slot from fall-through
...
foo: L.D F0, qrs
    
```

Solving WAW Hazards

- First approach: delay the issue of load instruction until ADD.D enters MEM
- Second approach: stamp out the result of the ADD.D by detecting the hazard and changing the control so that ADD.D does not write; LD issues right away
- Detect hazard in ID when LD is issuing
 - stall LD, or
 - make ADD.D no-op
- Luckily this hazard is rare

Hazard Detection in ID Stage

- Possible hazards
 - hazards among FP instructions
 - hazards between an FP instruction and an integer instr.
 - FP and integer registers are distinct, except for FP load-stores, and FP-integer moves
- Assume that pipeline does all hazard detection in ID stage

Hazard Detection in ID Stage (cont'd)

- Check for structural hazards
 - wait until the required functional unit is not busy and make sure that the register write port is available
- Check for RAW data hazards
 - wait until source registers are not listed as pending destinations in a pipeline register that will not be available when this instruction needs the result
- Check for WAW data hazards
 - determine if any instruction in A1, .. A4, M1, .. M7, D has the same register destination as this instruction; if so, stall the issue of the instruction in ID

Forwarding Logic

- Check if the destination register in any of EX/MEM, A4/MEM, M7/MEM, D/MEM, or MEM/WB pipeline registers is one of the source registers of a FP instruction
- If so, the appropriate input multiplexer will have to be enabled so as to choose the forwarded data

Dynamically Scheduled Pipelines



CPE
631
©AM

Overcoming Data Hazards with Dynamic Scheduling

- Why in HW at run time?
 - Works when can't know real dependence at compile time
 - Simpler compiler
 - Code for one machine runs well on another

- Example

DIV.D	F0, F2, F4
ADD.D	F10, F0, F8
SUB.D	F12, F8, F12

SUB.D cannot execute because the dependence of ADD.D on DIV.D causes the pipeline to stall; yet SUBD is not data dependent on anything!

- Key idea: Allow instructions behind stall to proceed

Overcoming Data Hazards with Dynamic Scheduling (cont'd)

- Enables out-of-order execution => out-of-order completion
- Out-of-order execution divides ID stage:
 - 1. Issue—decode instructions, check for structural hazards
 - 2. Read operands—wait until no data hazards, then read operands
- Scoreboarding – technique for allowing instructions to execute out of order when there are sufficient resources and no data dependencies (CDC 6600, 1963)

Scoreboarding Implications

- Out-of-order completion => WAR, WAW hazards?

DIV.D	F0, F2, F4
ADD.D	F10, F0, F8
SUB.D	F8, F8, F12

DIV.D	F0, F2, F4
ADD.D	F10, F0, F8
SUB.D	F10, F8, F12

- Solutions for WAR
 - Queue both the operation and copies of its operands
 - Read registers only during Read Operands stage
- For WAW, must detect hazard: stall until other completes
- Need to have multiple instructions in execution phase => multiple execution units or pipelined execution units
- Scoreboard keeps track of dependencies, state or operations
- Scoreboard replaces ID, EX, WB with 4 stages

Four Stages of Scoreboard Control

- ID1: Issue — decode instructions & check for structural hazards
- ID2: Read operands — wait until no data hazards, then read operands
- EX: Execute — operate on operands; when the result is ready, it notifies the scoreboard that it has completed execution
- WB: Write results — finish execution; the scoreboard checks for WAR hazards. If none, it writes results. If WAR, then it stalls the instruction

DIV.D	F0, F2, F4
ADD.D	F10, F0, F8
SUB.D	F8, F8, F12

Scoreboarding stalls the the SUBD in its write result stage until ADDD reads its operands

Four Stages of Scoreboard Control

- 1. **Issue**—decode instructions & check for structural hazards (ID1)
 - If a functional unit for the instruction is free and no other active instruction has the same destination register (WAW), the scoreboard issues the instruction to the functional unit and updates its internal data structure. If a structural or WAW hazard exists, then the instruction issue stalls, and no further instructions will issue until these hazards are cleared.
- 2. **Read operands**—wait until no data hazards, then read operands (ID2)
 - A source operand is available if no earlier issued active instruction is going to write it, or if the register containing the operand is being written by a currently active functional unit. When the source operands are available, the scoreboard tells the functional unit to proceed to read the operands from the registers and begin execution. The scoreboard resolves RAW hazards dynamically in this step, and instructions may be sent into execution out of order.

Four Stages of Scoreboard Control

- 3. **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.
- 4. **Write result**—finish execution (WB)
 - Once the scoreboard is aware that the functional unit has completed execution, the scoreboard checks for WAR hazards. If none, it writes results. If WAR, then it stalls the instruction.
 - Example:

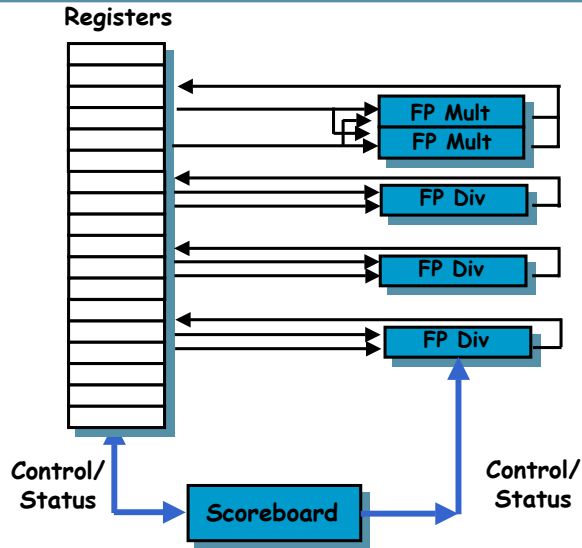
```
DIV.D      F0, F2, F4
ADD.D      F10, F0, F8
SUB.D      F8, F8, F14
```

- CDC 6600 scoreboard would stall SUBD until ADD.D reads operands

Three Parts of the Scoreboard

- 1. **Instruction status**—which of 4 steps the instruction is in (Capacity = window size)
- 2. **Functional unit status**—Indicates the state of the functional unit (FU). 9 fields for each functional unit
 - Busy—Indicates whether the unit is busy or not
 - Op—Operation to perform in the unit (e.g., + or –)
 - Fi—Destination register
 - Fj, Fk—Source-register numbers
 - Qj, Qk—Functional units producing source registers Fj, Fk
 - Rj, Rk—Flags indicating when Fj, Fk are ready
- 3. **Register result status**—Indicates which functional unit will write each register, if one exists. Blank when no pending instructions will write that register

MIPS with a Scoreboard



14/02/2005

UAH-CPE631

35

Detailed Scoreboard Pipeline Control

Instruction status	Wait until	Bookkeeping
Issue	Not busy (FU) and not result (D)	$Busy(FU) \leftarrow yes; Op(FU) \leftarrow op;$ $Fi(FU) \leftarrow 'D'; Fj(FU) \leftarrow 'S1';$ $Fk(FU) \leftarrow 'S2'; Qj \leftarrow Result('S1');$ $Qk \leftarrow Result('S2'); Rj \leftarrow not Qj;$ $Rk \leftarrow not Qk; Result('D') \leftarrow FU;$
Read operands	Rj and Rk	$Rj \leftarrow No; Rk \leftarrow No$
Execution complete	Functional unit done	
Write result	$\forall f((Fj(f) \neq Fi(FU) \text{ or } Rj(f) = No) \&$ $(Fk(f) \neq Fi(FU) \text{ or } Rk(f) = No))$	$\forall f(\text{if } Qj(f) = FU \text{ then } Rj(f) \leftarrow Yes);$ $\forall f(\text{if } Qk(f) = FU \text{ then } Rj(f) \leftarrow Yes);$ $Result(Fi(FU)) \leftarrow 0; Busy(FU) \leftarrow No$

14/02/2005

UAH-CPE631

36

Scoreboard Example: Cycle 2

Instruction status

Instruction	j	k
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

Functional unit status

Time	Name
	Integer
	Mult1
	Mult2
	Add
	Divide

Register result status

Clock	FU
2	

Read Executi Write Issue operan complei Result

Issue	operan	complei	Result
1	2		

Busy Op dest S1 S2 FU for FU for Fj? Fk?

Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Yes	Load	F6		R2				Yes
No								
No								
No								

F0	F2	F4	F6	F8	F10	F12	...	F30
Integer								

Issue 2nd L.D?

Structural hazard!
No further instructions will issue!

Scoreboard Example: Cycle 3

Instruction status

Instruction	j	k
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

Functional unit status

Time	Name
	Integer
	Mult1
	Mult2
	Add
	Divide

Register result status

Clock	FU
3	

Read Executi Write Issue operan complei Result

Issue	operan	complei	Result
1	2	3	

Busy Op dest S1 S2 FU for FU for Fj? Fk?

Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Yes	Load	F6		R2				Yes
No								
No								
No								

F0	F2	F4	F6	F8	F10	F12	...	F30
Integer								

Issue MUL.D?

Scoreboard Example: Cycle 4

Instruction status

Instruction	j	k
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

Functional unit status

Time Name
Integer
Mult1
Mult2
Add
Divide

Register result status

Clock	FU
4	Integer

Read Executi Write

Issue	operan	complei	Result
1	2	3	4

Busy Op dest S1 S2 FU for FU for Fj? Fk?

Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Yes	Load	F6		R2				Yes
No								
No								
No								

Check for WAR hazards!
If none, write result!

14/02/2005

UAH-CPE631

41

Scoreboard Example: Cycle 5

Instruction status

Instruction	j	k
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

Functional unit status

Time Name
Integer
Mult1
Mult2
Add
Divide

Register result status

Clock	FU
5	Integer

Read Executi Write

Issue	operan	complei	Result
1	2	3	4
5			

Busy Op dest S1 S2 FU for FU for Fj? Fk?

Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Yes	Load	F2		R3				Yes
No								
No								
No								

Issue 2nd L.D!

14/02/2005

UAH-CPE631

42

Scoreboard Example: Cycle 6

Instruction status

Instruction	j	k
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

Read Executi Write

Issue	operan	complei	Result
1	2	3	4
5	6		
6			

Functional unit status

Time Name
Integer
Mult1
Mult2
Add
Divide

Busy Op dest S1 S2 FU for FU for Fj? Fk?

Busy Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Yes	Load	F2	R3				Yes
Yes	Mult	F0	F2	F4	Integer	No	Yes
No							
No							

Register result status

Clock	FU
6	

F0	F2	F4	F6	F8	F10	F12	...	F30
Mult1	Integer							

Issue MUL.D!

Scoreboard Example: Cycle 7

Instruction status

Instruction	j	k
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

Read Executi Write

Issue	operan	complei	Result
1	2	3	4
5	6	7	
6			
7			

Functional unit status

Time Name
Integer
Mult1
Mult2
Add
Divide

Busy Op dest S1 S2 FU for FU for Fj? Fk?

Busy Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Yes	Load	F2	R3				Yes
Yes	Mult	F0	F2	F4	Integer	No	Yes
No							
Yes	Sub	F8	F6	F2		Integer	Yes
No							

Register result status

Clock	FU
7	

F0	F2	F4	F6	F8	F10	F12	...	F30
Mult1	Integer			Add				

Issue SUB.D!

Scoreboard Example: Cycle 8

Instruction status

Instruction	j	k
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

Issue	Read operands		Execute/Write Result	
	1	2	3	4
1				
5	6	7	8	
6				
7				
8				

Functional unit status

Time	Name
	Integer
	Mult1
	Mult2
	Add
	Divide

Busy	Op	dest	S1	S2	FU for .FU for .Fj?	Fk?
Fi	Fj	Fk	Qj	Qk	Rj	Rk
Yes	Load	F2		R3		Yes
Yes	Mult	F0	F2	F4	Integer	No Yes
No						
Yes	Sub	F8	F6	F2	Integer	Yes No
Yes	Div	F10	F0	F6	Mult1	No Yes

Register result status

Clock	FU
8	

F0	F2	F4	F6	F8	F10	F12	...	F30
Mult1	Integer			Add	Divide			

Issue DIV.D!

Scoreboard Example: Cycle 9

Instruction status

Instruction	j	k
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

Issue	Read operands		Execute/Write Result	
	1	2	3	4
1				
5	6	7	8	
6	9			
7	9			
8				

Functional unit status

Time	Name
	Integer
10	Mult1
	Mult2
2	Add
	Divide

Busy	Op	dest	S1	S2	FU for .FU for .Fj?	Fk?
Fi	Fj	Fk	Qj	Qk	Rj	Rk
No						
Yes	Mult	F0	F2	F4	Integer	Yes Yes
No						
Yes	Sub	F8	F6	F2	Integer	Yes Yes
Yes	Div	F10	F0	F6	Mult1	No Yes

Register result status

Clock	FU
9	

F0	F2	F4	F6	F8	F10	F12	...	F30
Mult1				Add	Divide			

Read operands for MUL.D and SUB.D!

Assume we can feed Mult1 and Add units in the same clock cycle.

Issue ADD.D? Structural Hazard (unit is busy)!

Scoreboard Example: Cycle 11

Instruction status

Instruction	j	k
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

Read Executi Write

Issue	operan	complei	Result
1	2	3	4
5	6	7	8
6	9		
7	9	11	
8			

Functional unit status

Time	Name
8	Mult1
0	Add
	Divide

Busy Op dest S1 S2 FU for FU for Fj? Fk?

Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
No								
Yes	Mult	F0	F2	F4	Integer		Yes	Yes
No								
Yes	Sub	F8	F6	F2		Integer	Yes	Yes
Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock	FU
11	

F0	F2	F4	F6	F8	F10	F12	...	F30
Mult1				Add	Divide			

Last cycle of SUB.D execution.

Scoreboard Example: Cycle 12

Instruction status

Instruction	j	k
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

Read Executi Write

Issue	operan	complei	Result
1	2	3	4
5	6	7	8
6	9		
7	9	11	12
8			

Functional unit status

Time	Name
7	Mult1
	Mult2
	Add
	Divide

Busy Op dest S1 S2 FU for FU for Fj? Fk?

Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
No								
Yes	Mult	F0	F2	F4	Integer		Yes	Yes
No								
Yes	Sub	F8	F6	F2		Integer	Yes	Yes
Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock	FU
12	

F0	F2	F4	F6	F8	F10	F12	...	F30
Mult1				Add	Divide			

Check WAR on F8. Write F8.

Scoreboard Example: Cycle 13

Instruction status

Instruction	j	k
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

Read Executi Write

Issue	operan	complei	Result
1	2	3	4
5	6	7	8
6	9		
7	9	11	12
8			
13			

Functional unit status

Time	Name
6	Mult1
	Mult2
	Add
	Divide

Busy Op dest S1 S2 FU for FU for Fj? Fk?

Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
No								
Yes	Mult	F0	F2	F4	Integer		Yes	Yes
No								
Yes	Add	F6	F8	F2			Yes	Yes
Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock	FU
13	

F0	F2	F4	F6	F8	F10	F12	...	F30
Mult1			Add		Divide			

Issue ADD.D!

Scoreboard Example: Cycle 14

Instruction status

Instruction	j	k
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

Read Executi Write

Issue	operan	complei	Result
1	2	3	4
5	6	7	8
6	9		
7	9	11	12
8			
13	14		

Functional unit status

Time	Name
5	Mult1
	Mult2
2	Add
	Divide

Busy Op dest S1 S2 FU for FU for Fj? Fk?

Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
No								
Yes	Mult	F0	F2	F4	Integer		Yes	Yes
No								
Yes	Add	F6	F8	F2			Yes	Yes
Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock	FU
14	

F0	F2	F4	F6	F8	F10	F12	...	F30
Mult1			Add		Divide			

Read operands for ADD.D!

Scoreboard Example: Cycle 15

Instruction status

Instruction	j	k
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

Read		Execute		Write	
Issue	operand	complete	Result		
1	2	3	4		
5	6	7	8		
6	9				
7	9	11	12		
8					
13	14				

Functional unit status

Time	Name
4	Mult1
	Mult2
1	Add
	Divide

Busy	Op	dest	S1	S2	FU for	FU for	Fj?	Fk?
Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk	
No								
Yes	Mult	F0	F2	F4	Integer		Yes	Yes
No								
Yes	Add	F6	F8	F2			Yes	Yes
Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock	FU
14	

F0	F2	F4	F6	F8	F10	F12	...	F30
Mult1			Add		Divide			

Scoreboard Example: Cycle 16

Instruction status

Instruction	j	k
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

Read		Execute		Write	
Issue	operand	complete	Result		
1	2	3	4		
5	6	7	8		
6	9				
7	9	11	12		
8					
13	14	16			

Functional unit status

Time	Name
3	Mult1
	Mult2
0	Add
	Divide

Busy	Op	dest	S1	S2	FU for	FU for	Fj?	Fk?
Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk	
No								
Yes	Mult	F0	F2	F4	Integer		Yes	Yes
No								
Yes	Add	F6	F8	F2			Yes	Yes
Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock	FU
16	

F0	F2	F4	F6	F8	F10	F12	...	F30
Mult1			Add		Divide			

Scoreboard Example: Cycle 17

Instruction status

Instruction	j	k
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

Read		Execute		Write	
Issue	operand	complete	Result	Issue	operand
1	2	3	4		
5	6	7	8		
6	9				
7	9	11	12		
8					
13	14	16			

Functional unit status

Time	Name
Integer	
2	Mult1
	Mult2
	Add
	Divide

Busy	Op	dest	S1	S2	FU for	FU for	Fj?	Fk?
Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk	
No								
Yes	Mult	F0	F2	F4	Integer	Yes	Yes	
No								
Yes	Add	F6	F8	F2		Yes	Yes	
Yes	Div	F10	F0	F6	Mult1	No	Yes	

Register result status

Register	Value
F0	Mult1
F2	
F4	
F6	Add
F8	
F10	Divide
F12	
...	
F30	

F0	F2	F4	F6	F8	F10	F12	...	F30
Mult1			Add		Divide			

Why cannot write F6?

Scoreboard Example: Cycle 19

Instruction status

Instruction	j	k
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

Read		Execute		Write	
Issue	operand	complete	Result	Issue	operand
1	2	3	4		
5	6	7	8		
6	9	19			
7	9	11	12		
8					
13	14	16			

Functional unit status

Time	Name
Integer	
0	Mult1
	Mult2
	Add
	Divide

Busy	Op	dest	S1	S2	FU for	FU for	Fj?	Fk?
Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk	
No								
Yes	Mult	F0	F2	F4	Integer	Yes	Yes	
No								
Yes	Add	F6	F8	F2		Yes	Yes	
Yes	Div	F10	F0	F6	Mult1	No	Yes	

Register result status

Register	Value
F0	Mult1
F2	
F4	
F6	Add
F8	
F10	Divide
F12	
...	
F30	

F0	F2	F4	F6	F8	F10	F12	...	F30
Mult1			Add		Divide			

Scoreboard Example: Cycle 20

Instruction status

Instruction	j	k
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

Read		Execute		Write	
Issue	operand	complete	Result	Issue	operand
1	2	3	4		
5	6	7	8		
6	9	19	20		
7	9	11	12		
8					
13	14	16			

Functional unit status

Time	Name
	Integer
	Mult1
	Mult2
	Add
	Divide

Busy	Op	dest	S1	S2	FU for	FU for	Fj?	Fk?
Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk	
No								
Yes	Mult	F0	F2	F4	Integer		Yes	Yes
No								
Yes	Add	F6	F8	F2			Yes	Yes
Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock	FU
20	

F0	F2	F4	F6	F8	F10	F12	...	F30
Mult1			Add		Divide			

Scoreboard Example: Cycle 21

Instruction status

Instruction	j	k
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

Read		Execute		Write	
Issue	operand	complete	Result	Issue	operand
1	2	3	4		
5	6	7	8		
6	9	19	20		
7	9	11	12		
8	21				
13	14	16			

Functional unit status

Time	Name
	Integer
	Mult1
	Mult2
	Add
	Divide

Busy	Op	dest	S1	S2	FU for	FU for	Fj?	Fk?
Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk	
No								
No								
No								
Yes	Add	F6	F8	F2			Yes	Yes
Yes	Div	F10	F0	F6	Mult1		Yes	Yes

Register result status

Clock	FU
21	

F0	F2	F4	F6	F8	F10	F12	...	F30
			Add		Divide			

Scoreboard Example: Cycle 22

Instruction status

Instruction	j	k
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

Read		Execute		Write	
Issue	operand	complete	Result	Issue	operand
1	2	3	4		
5	6	7	8		
6	9	19	20		
7	9	11	12		
8	21				
13	14	16	22		

Functional unit status

Time	Name
	Integer
	Mult1
	Mult2
	Add
40	Divide

Busy	Op	dest	S1	S2	FU for	FU for	Fj?	Fk?
Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk	
No								
No								
No								
Yes	Add	F6	F8	F2			Yes	Yes
Yes	Div	F10	F0	F6	Mult1		Yes	Yes

Register result status

Clock	FU
22	

F0	F2	F4	F6	F8	F10	F12	...	F30
			Add		Divide			

Write F6?

Scoreboard Example: Cycle 61

Instruction status

Instruction	j	k
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

Read		Execute		Write	
Issue	operand	complete	Result	Issue	operand
1	2	3	4		
5	6	7	8		
6	9	19	20		
7	9	11	12		
8	21	61			
13	14	16	22		

Functional unit status

Time	Name
	Integer
	Mult1
	Mult2
	Add
0	Divide

Busy	Op	dest	S1	S2	FU for	FU for	Fj?	Fk?
Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk	
No								
No								
No								
No								
Yes	Div	F10	F0	F6	Mult1		Yes	Yes

Register result status

Clock	FU
61	

F0	F2	F4	F6	F8	F10	F12	...	F30
					Divide			

Scoreboard Example: Cycle 62

Instruction status

Instruction	<i>j</i>	<i>k</i>
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

Read		Execute		Write	
Issue	operand	complete	Result	Issue	operand
1	2	3	4		
5	6	7	8		
6	9	19	20		
7	9	11	12		
8	21	61	62		
13	14	16	22		

Functional unit status

Time	Name
	Integer
	Mult1
	Mult2
	Add
	Divide

Busy	Op	dest		S1	S2	FU for .FU for .Fj?	Fk?
		Fi	Fj	Fk	Qj	Qk	Rj
No							
No							
No							
No							
Yes	Div	F10	F0	F6	Mult1		Yes Yes

Register result status

Clock	FU
62	

F0	F2	F4	F6	F8	F10	F12	...	F30
					Divide			

Scoreboard Results

- For the CDC 6600
 - 70% improvement for Fortran
 - 150% improvement for hand coded assembly language
 - cost was similar to one of the functional units
 - surprisingly low
 - bulk of cost was in the extra busses
- Still this was in ancient time
 - no caches & no main semiconductor memory
 - no software pipelining
 - compilers?
- So, why is it coming back
 - performance via ILP

Scoreboard Limitations

- Amount of parallelism among instructions
 - can we find independent instructions to execute
- Number of scoreboard entries
 - how far ahead the pipeline can look for independent instructions (we assume a window does not extend beyond a branch)
- Number and types of functional units
 - avoid structural hazards
- Presence of antidependences and output dependences
 - WAR and WAW stalls become more important

Things to Remember

- Pipeline CPI = Ideal pipeline CPI + Structural stalls + RAW stalls + WAR stalls + WAW stalls + Control stalls
- Data dependencies
- Dynamic scheduling to minimise stalls
- Dynamic scheduling with a scoreboard