HY425 Lecture 03: Exploiting Instruction-Level Parallelism in HW with Dynamic Scheduling

Dimitrios S. Nikolopoulos

University of Crete and FORTH-ICS

October 13, 2011

Pipelining

Performance equation

$$CPI_{pipelined} = CPI_{ideal} + Stalls_{structural} + Stalls_{RAW} + Stalls_{WAR} + Stalls_{WAW} + Stalls_{branch}$$

Instruction-level parallelism

- Pipelining overlaps instruction execution $\rightarrow$ ILP
- Hazards limit available ILP
- Will look into ways to increase available ILP in SW and HW

Reminders

- Second homework out this week
- Midterm scheduled on Friday October 29, 09:15–11:00 in class (Rho Alpha 203).

ILP

Looking for instruction parallelism

- Sequences of (control- or data-) independent instructions
- Within boundaries of a basic block
    - Block of code with single entry, single exit point
    - Objectives
        - Resolve hazards via hardware or software
        - Reduce the impact of hazards
        - Overlap latency of slow instructions (e.g. FP)

Looking for ILP

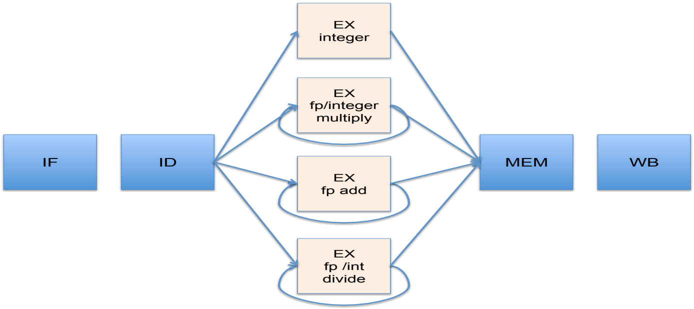
Loops are important sources of parallelism

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

- Every iteration can execute independently of any other
- No dependencies across iterations (loop-carried)
- Dependencies OK to exist in the loop body for one iteration
  - Producer and consumer instructions are in the same iteration

Pipelines with variable instruction latencies

Loops in the EX stage



Instructions with variable latency

Instruction latency

- Number of intervening cycles between instruction producing a result and instruction consuming a result
- One cycle less than the depth of the EX stage
- 0 for integer ALU operations to all operations consuming result in the EX stage (forwarding)
- Stores consume result earlier (ID), forwarding resolves in 0 cycles
- 1 for loads, result consumed a cycle later (MEM stage)
- $N - 1$ for a floating point operation with N EX cycles

Initiation interval

- Cycles between issuing two operations of the same type

Pipelines with variable instruction latencies

Hazards

- Execution units not fully pipelined (structural hazards)
- Instructions may need to write registers in the same cycle
  - increase write ports in register file, or stall instructions at ID
- WAW hazards now possible
  - fast operation may bypass slow operation
- Instructions may complete out of order
- Stalls and RAW hazards more frequent

Hazards with variable instruction latencies

10-cycle EX MULD, 2-cycle EX ADDD, 40-cycle DIVD

Instruction		Clock Cycle Number																
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
LD	F6,34(R2)	IF	ID	EX	MEM	WB												
LD	F2,45(R3)		IF	ID	EX	MEM	WB											
MULTD	F0,F2,F4			IF	ID	stall	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	MEM	WB
SUBD	F8,F6,F2				IF	ID	A1	A2	MEM	WB								
DIVD	F10,F0,F6					IF	ID	stall	stall	stall	stall	stall	stall	stall	stall	stall	D1	D2
ADDD	F6,F8,F2						IF	ID	A1	A2	MEM	WB						

- RAW hazard through `F2,F0`
- WAR hazard through `F6`
- What happens if first load misses in L1 cache and takes 6+ cycles?

Hazards with variable instruction latencies (cont.)

10-cycle EX MULD, 2-cycle EX ADDD, 40-cycle DIVD

Instruction	Clock Cycle Number																
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
LD F6,34(R2)	IF	ID	EX	MEM	WB												
LD **F2**,45(R3)		IF	ID	EX	MEM	WB											
MULTD **F0**,**F2**,F4			IF	ID	stall	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	MEM	WB
SUBD F8,F6,**F2**				IF	ID	A1	A2	MEM	WB								
DIVD F10,**F0**,**F6**					IF	ID	stall	stall	stall	stall	stall	stall	stall	stall	stall	D1	D2
ADDD **F6**,F8,F2						IF	ID	A1	A2	MEM	WB						

- Hazard detection logic at ID prevents instruction from executing op
- ID reveals registers and conflicts with prior instructions
- Forwarding harder with variable latency

Loops are important sources of parallelism

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

Assume latencies

Producer	Consumer	Latency
FP ALU op	Another FP ALU op	3
FP ALU op	store double	2
Load double	FP ALU op	1
Load double	store double	0

Instruction scheduling in SW

Naïve implementation in MIPS-like assembly

```
                         Clock cycle issued
Loop:  LD   F0, 0(R1)         1
       stall                  2
       ADDD F4, F0, F2        3
       stall                  4
       stall                  5
       SD   F4, 0(R1)         6
       SUBI R1, R1, 8         7
       BNEZ  R1, Loop         8
       stall                  9
```

- 9 cycles per loop iteration due to stalls

Instruction scheduling and reordering

Delayed branch, reordering of SUBI, SD

```
                            Clock cycle issued
Loop:  LD  F0, 0(R1)             1
       stall                     2
       ADDD F4, F0, F2           3
       SUBI R1, R1, 8            4
       BNEZ  R1, Loop            5
       SD  F4, 8(R1)             6
```

- Fill delay slot with SD
  - R1 has changed, so adjust offset
- Created enough distance between ADDD and SD to avoid 2-cycle stall
- 6 cycles per loop iteration

Loop unrolling

Unroll 3 iterations in example

```
                               Clock cycle issued
Loop:  LD   F0, 0(R1)                1
       ADDD F4, F0, F2               3
       SD   F4, 0(R1)                6
       LD   F6, -8(R1)               7
       ADDD F8, F6, F2               9
       SD   F8, -8(R1)               12
       LD   F10, -16(R1)             13
       ADDD F12, F10, F2             15
       SD   F12, -16(R1)             18
       LD   F14, -24(R1)             19
       ADDD F16, F14, F2             21
       SD   F16, -24(R1)             24
       SUBI R1, R1, 32               25
       BNEZ  R1, Loop                26
       stall                         27
```

Unrolling pros/cons

- Pros
  - Less branches, reduced impact of branches
  - 27 cycles for 4 iterations, 6.8 per iteration
- Cons
  - Increases code size
  - Need more registers (2 per unrolled iteration)

Instruction scheduling and reordering

Delayed branch, reordering of SUBI, SD

```
                          Clock cycle issued
Loop:  LD   F0, 0(R1)          1
       stall                   2
       ADDD F4, F0, F2         3
       SUBI R1, R1, 8          4
       BNEZ  R1, Loop          5
       SD   F4, 8(R1)          6
```

- Fill delay slot with SD
  - R1 has changed, so adjust offset
- Created enough distance between ADDD and SD to avoid 2-cycle stall
- 6 cycles per loop iteration

Loop unrolling

Unroll 3 with instruction scheduling

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

Improved instruction scheduling

- LDs moved up to eliminate LD-ADDD 1-cycle stall
- ADDDs moved up to eliminate ADDD-SD 2-cycle stalls
- SD move down to fill branch delay slot
- Need adjustment of SD indices
- 14 cycles for 4 iterations, or 3.5 cycles per iteration

Dependences

Data dependences

Instruction *j* is data-dependent on instruction *i* if:

- Instruction *i* precedes instruction *j* in program order
- Instruction *i* produces result used by instruction *j*
- Instruction *j* is data-dependent on instruction *k*, and *k* is data-dependent on instruction *i*
- Data dependences are true dependences due to data flow in the program

Dependences (cont.)

Name dependences

Instruction *j* is name-dependent on instruction *i* if:

- Instruction *i* precedes instruction *j* in program order or
- Instruction *j* writes to register or memory location read by instruction *i* first
  - Anti-dependence, WAR hazard
- Instruction *j* writes to register or memory location written by instruction *i*
  - Output dependence, WAW hazard
- Anti- and output-dependences resolved if we assign a different register (easier) or memory (harder) destination of dependent instruction (register/memory renaming)

Example with dependences

Unrolled code wit renaming

```
                              Clock cycle issued
Loop:  LD   F0, 0(R1)              1
       ADDD F4, F0, F2             3
       SD   F4, 0(R1)              6
       LD   F0, -8(R1)             7
       ADDD F4, F0, F2             9
       SD   F4, -8(R1)             12
       LD   F0, -16(R1)            13
       ADDD F4, F0, F2             15
       SD   F4, -16(R1)            18
       LD   F0, -24(R1)            19
       ADDD F4, F0, F2             21
       SD   F4, -24(R1)            24
       SUBI R1, R1, 32             25
       BNEZ  R1, Loop              26
       stall                       27
```

Dependences

- Pairs of LDs incur output dependences on F0
- Pairs of ADDDs incur output dependences on F4
- LD-ADDD pair in each iteration has true data dependence on F0

Example with dependences (cont.)

Unrolled code with renaming

```
                              Clock cycle issued
Loop:  LD   F0, 0(R1)              1
       ADDD F4, F0, F2             3
       SD   F4, 0(R1)              6
       LD   F6, -8(R1)             7
       ADDD F8, F6, F2             9
       SD   F8, -8(R1)             12
       LD   F10, -16(R1)           13
       ADDD F12, F10, F2           15
       SD   F12, -16(R1)           18
       LD   F14, -24(R1)           19
       ADDD F16, F14, F2           21
       SD   F16, -24(R1)           24
       SUBI R1, R1, 32             25
       BNEZ  R1, Loop              26
       stall                       27
```

Dependences

- LD-ADD pair in each iteration incurs true data dependences on F0, F6, F10, F14
- ADDD-SD pair in each iteration incurs true data dependence on F4, F8, F12, F16

Control dependences

Execution of instructions may depend on branches

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

- S1 control-dependent on p1
- S2 control-dependent on p2
- S2 not control-dependent on p1
- S1 not control-dependent on p2

Control dependence example

Unroll 3 iterations in example

```
1    Loop:  LD   F0,  0(R1)
2           ADDD F4,  F0,  F2
3           SD   F4,  0(R1)
4           SUBI   R1,R1,8
5           BEQZ R1,exit
7           LD   F6,  0(R1)
8           ADDD F8,  F6,  F2
9           SD   F8,  0(R1)
10          SUBI   R1,R1,8
11          BEQZ R1,exit
12          LD   F10,  0(R1)
13          ADDD F12,  F10,  F2
14          SD   F12,   0(R1)
15          SUBI   R1,R1,8
16          BEQZ R1,exit
17          LD   F14,  0(R1)
18          ADDD F16,  F14,  F2
19          SD   F16,  0(R1)
20          SUBI R1,  R1,  9
21          BNEZ  R1,  Loop
22   exit:
```

Control dependences

- 7–11 are control-dependent on BEQZ at 5
- 12–16 are control-dependent on BEQZ at 11
- 17–21 are control-dependent on BEQZ at 16
- Control dependences limit ILP and harm performance

Control dependences and exceptions

```
      BEQZ R2,  L
      LW   R1,  0(R2)
L:
```

- Reordering control-dependent instructions has side-effects
- Attempt to move LW should not trigger exception

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

- Reordering control-dependent instructions may alter data flow
- Attempt to move SUB before branch changes input to OR
- If violating control dependence does not alter control- or data-flow we can speculate on the branch outcome and move instructions to increase performance

Dynamic scheduling in hardware

Key ideas

- Pipeline imposes in-order issue (ID) and execution (EX) of instructions
- Stalled older instructions delay newer instructions, even if independent
- Check for structural and data hazards done at ID stage
- If no data hazard instruction can execute out-of-order
- Split instruction stage into:
    - Issue – decode, check for structural hazards
    - Read operands – wait for no data hazards, then read operands

Motivation for dynamic scheduling

Preventing data hazards

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

- ADDD waits on slow DIVD
- If SUBD is executed out-of-order it violates anti-dependence with ADD
- Triggers non-existing data dependence with ADDD

Scoreboard

Preventing data hazards

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

- Scoreboard keeps record of dependences per instruction
- Stalls if data dependence exists
- Releases instruction upon data dependence resolution pause
- Multiple execution units enable out-of-order execution
- Stalls register writes to avoid dependence violations

Scoreboard with instruction execution steps

- Issue – decode and check for structural hazards
  - Instructions issued strictly in program order
  - Instruction not issued if structural hazard exists
  - Instruction not issued if output-dependent on earlier instruction
- Read operands – wait until no hazards (data, structural) read operands
  - True dependences resolved in this stage
  - No value forwarding, result communicated through registers

Scoreboard instruction execution steps

- Execute – functional unit begins execution, notifies scoreboard upon completion
- Write result – stall if WAR hazards are violated (reordering), otherwise write result to register file

Scoreboard

Components

- Instruction status
  - Indicates which of the steps the instruction is in
- Functional unit status
  - Indicates state of functional unit
  - Busy – indicates unit busy or not
  - Op – indicates operation to perform by unit
  - F_i – indicates destination register
  - F_j, F_k – indicate input registers
  - Q_j, Q_k – indicate functional units producing input registers
  - R_j, R_k – flags indicating if functional units are ready
- Register result status
  - Indicates which functional unit will write each register
  - Blank if no instructions will write to that register

Scoreboard example

Instruction status:

Instruction		j	k	Issue	Read Oper	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				

Functional unit status:

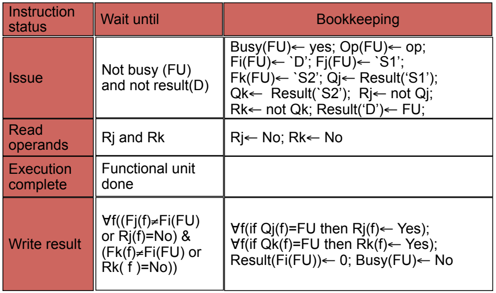
Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

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

Scoreboard control

Latencies: ADDD 2 cycles, MULD 10 cycles, DIVD 40 cycles



Instruction status	Wait until	Bookkeeping
Issue	Not busy (FU) and not result(D)	Busy(FU)← yes; Op(FU)← op; Fi(FU)← `D’; Fj(FU)← `S1’; Fk(FU)← `S2’; Qj← Result(‘S1’); Qk← Result(`S2’); Rj← not Qj; Rk← not Qk; Result(‘D’)← FU;
Read operands	Rj and Rk	Rj← No; Rk← No
Execution complete	Functional unit done	
Write result	∀f((Fj(f)≠Fi(FU) or Rj(f)=No) & (Fk(f)≠Fi(FU) or Rk(f)=No))	∀f(if Qj(f)=FU then Rj(f)← Yes); ∀f(if Qk(f)=FU then Rk(f)← Yes); Result(Fi(FU))← 0; Busy(FU)← No

Scoreboard example – cycle 1

Instruction status:

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

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? 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
1	FU									

Scoreboard example – cycle 2

- Second load does not issue due to structural hazard

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2		
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
	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
2	FU				Integer					

Scoreboard example – cycle 3

- MULTD does not issue because all instructions issue in-order

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	
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
	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
3	FU				Integer					

Scoreboard example – cycle 4

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
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
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
4	FU				Integer					

Scoreboard example – cycle 5

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5			
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
	Integer	Yes	LD	F2		R3				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
5	FU		Integer							

Scoreboard example – cycle 6

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6		
MULTD	F0	F2	F4	6			
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
	Integer	Yes	LD	F2		R3				Yes
	Mult1	Yes	MULTD	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	No								
	Divide	No								

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU	Mult1	Integer							

Scoreboard example – cycle 7

- MULTD operand in F2 not ready
- SUBD issues

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
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
	Integer	Yes	LD	F2		R3				No
	Mult1	Yes	MULTD	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	SUBD	F8	F6	F2		Integer	Yes	No
	Divide	No								

Register result status:

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

Scoreboard example – cycle 8

- DIVD issues
- MULTD, SUBD start execution

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6	8			
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
	Integer	No								
	Mult1	Yes	MULTD	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	SUBD	F8	F6	F2			Yes	Yes
	Divide	Yes	DIVD	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard example – cycle 9

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9		
DIVD	F10	F0	F6	8			
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
	Integer	No								
10	Mult1	Yes	MULTD	F0	F2	F4			Yes	Yes
	Mult2	No								
2	Add	Yes	SUBD	F8	F6	F2			Yes	Yes
	Divide	Yes	DIVD	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard example – cycle 10

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9		
DIVD	F10	F0	F6	8			
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
	Integer	No								
9	Mult1	Yes	MULTD	F0	F2	F4			No	No
	Mult2	No								
1	Add	Yes	SUBD	F8	F6	F2			No	No
	Divide	Yes	DIVD	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard example – cycle 11

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	
DIVD	F10	F0	F6	8			
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
	Integer	No								
8	Mult1	Yes	MULTD	F0	F2	F4			No	No
	Mult2	No								
0	Add	Yes	SUBD	F8	F6	F2			No	No
	Divide	Yes	DIVD	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard example – cycle 12

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
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
	Integer	No								
7	Mult1	Yes	MULTD	F0	F2	F4			No	No
	Mult2	No								
	Add	No								
	Divide	Yes	DIVD	F10	F0	F6	Mult1		No	Yes

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
12	FU	Mult1					Divide			

Scoreboard example – cycle 13

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13			

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
6	Mult1	Yes	MULTD	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	ADDD	F6	F8	F2			Yes	Yes
	Divide	Yes	DIVD	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard example – cycle 14

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14		

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
5	Mult1	Yes	MULTD	F0	F2	F4			No	No
	Mult2	No								
2	Add	Yes	ADDD	F6	F8	F2			Yes	Yes
	Divide	Yes	DIVD	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard example – cycle 15

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14		

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
4	Mult1	Yes	MULTD	F0	F2	F4			No	No
	Mult2	No								
1	Add	Yes	ADDD	F6	F8	F2			No	No
	Divide	Yes	DIVD	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard example – cycle 16

Instruction status:

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

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
3	Mult1	Yes	MULTD	F0	F2	F4			No	No
	Mult2	No								
0	Add	Yes	ADDD	F6	F8	F2			No	No
	Divide	Yes	DIVD	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard example – cycle 17

- ADDD stall on write – WAR hazard

Instruction status:

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

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
2	Mult1	Yes	MULTD	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	ADDD	F6	F8	F2			No	No
	Divide	Yes	DIVD	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard example – cycle 18

- ADDD stall on write – WAR hazard

Instruction status:

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

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
1	Mult1	Yes	MULTD	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	ADDD	F6	F8	F2			No	No
	Divide	Yes	DIVD	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard example – cycle 19

Instruction status:

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

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
0	Mult1	Yes	MULTD	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	ADDD	F6	F8	F2			No	No
	Divide	Yes	DIVD	F10	F0	F6	Mult1		No	Yes

Register result status:

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

Scoreboard example – cycle 20

Instruction status:

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

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	ADDD	F6	F8	F2			No	No
	Divide	Yes	DIVD	F10	F0	F6			Yes	Yes

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
20	FU				Add		Divide			

Scoreboard example – cycle 21

- DIVD can execute – WAR hazard gone
- ADDD can write result

Instruction status:

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

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	ADDD	F6	F8	F2			No	No
	Divide	Yes	DIVD	F10	F0	F6			Yes	Yes

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
21	FU				Add		Divide			

Scoreboard example – cycle 22

Instruction status:

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

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
39	Divide	Yes	DIVD	F10	F0	F6			Yes	Yes

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
22	FU				Add		Divide			

Scoreboard example – out-of-order execution

- In-order issue
- Out-of-order execution
- Out-of-order completion. What about precise exceptions?

Instruction status:

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

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
0	Divide	Yes	DIVD	F10	F0	F6			Yes	Yes

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
61	FU						Divide			

Scoreboard

Limitations

- Parallelism exploited only between instructions in a single basic block
- Large instruction window – number of scoreboard entries available to look for ILP
- Number and type of functional units – structural hazards
- Anti- and output-dependences – both stall the scoreboard on write-back
- Relies heavily on compiler to schedule instructions

Dynamic scheduling strength and limitations

Strengths

- High performance, if instruction window is sufficiently large
- Depends less on sophisticated compiler support
- Good for architectures where it is hard to optimally schedule code, or have few registers

Dynamic scheduling strength and limitations

Limitations

- Requires advanced techniques to increase instruction window, most notably branch prediction (e.g. to find instructions across the iterations of a parallel loop)
- High complexity, therefore high power consumption. Complex control logic for reservation stations, plus need high-speed associative searches
- Single common data bus can be a bottleneck

Improving ILP in HW/SW

- Branch prediction (next topic): Uncovering more instructions to overlap by predicting if branches or taken or not taken and reducing branch stall impact
- Multiple issue: Allow processor to issue and hopefully complete more than one instructions per cycle
- Hardware speculation: Allow processor to execute instructions without knowing the outcome of branches (speculatively)
- Combined techniques: Multiple issue with dynamic scheduling, multiple issue with static scheduling, multiple issue with branch prediction, multiple issue with speculation