

CS184b: Computer Architecture [Single Threaded Architecture: abstractions, quantification, and optimizations]

Day7: January 25, 2000

Precise Exceptions

ILP intro

Today

- Handling Exceptions
- ILP
 - where?
 - scoreboard
 - tomasulo

Exceptions

- **Problem:** Maintain sequentially consistent view, while relaxing strict, sequential dependence ordering
- Sequential stream from ISA
- Data/control dependence less strict
- Relaxed dependence accelerates execution

In-Pipe

```
MPY R1,R2,R3 IF ID MPY1 MPY2 MPY3 WB
LW R4,16(R6) IF ID EX MEM ---- WB
```

Fault for later instruction should not be visible before earlier.

Out-of-Order Completion

MPY R1,R2,R3	IF	ID	EX	MPY1	MPY2	MPY3	MPY4	WB
LW R7,(R4)		IF	ID	ALU	MEM	WB		
ADD R4,R5,R6			IF	ID	ALU	---		WB

State changes from later operations should not be visible if earlier operations fail.

Solutions

- Stall side-effects as hazards
 - limit concurrency
- Imprecise exceptions
 - ? Recoverable / restartable
- Expose Pipeline
 - limit scalability, weaken abstraction
- Save list of PCs
 - cumbersome
- Precise Exception support

In-Order Completion

- Stall like data hazards
- Save up faults in pipeline until commit point
 - (faults, like WB occur in set place when know predecessors haven't faulted)

In-Order

```
MPY R1,R2,R3 IF ID MPY1 MPY2 MPY3 WB
LW R4,16(R6) IF ID EX MEM ---- WB
```

Commit fault with write back.

In-Order Completion

IO

MPY R1,R2,R3	IF	ID	EX	MPY1	MPY2	MPY3	MPY4	WB
LW R7,(R4)		IF	ID	ALU	MEM	WB		
ADD R4,R5,R6			IF	ID	ALU	---		WB

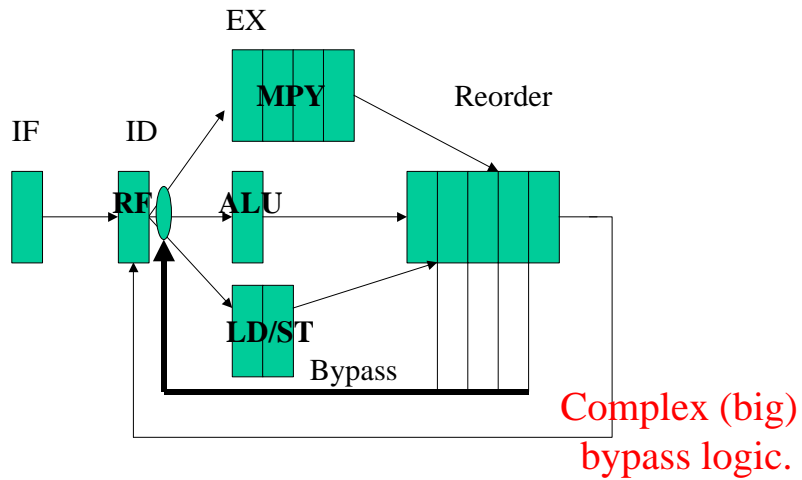
OO

MPY R1,R2,R3	IF	ID	EX	MPY1	MPY2	MPY3	MPY4	WB
LW R7,(R4)		IF	ID	ALU	MEM			WB
ADD R4,R5,R6			IF	ID	ALU			W

Re-Order Buffer

- Continue to execute
- Write-back to register file in-order
- Buffer results between completion and WB
- Bypass with newer results

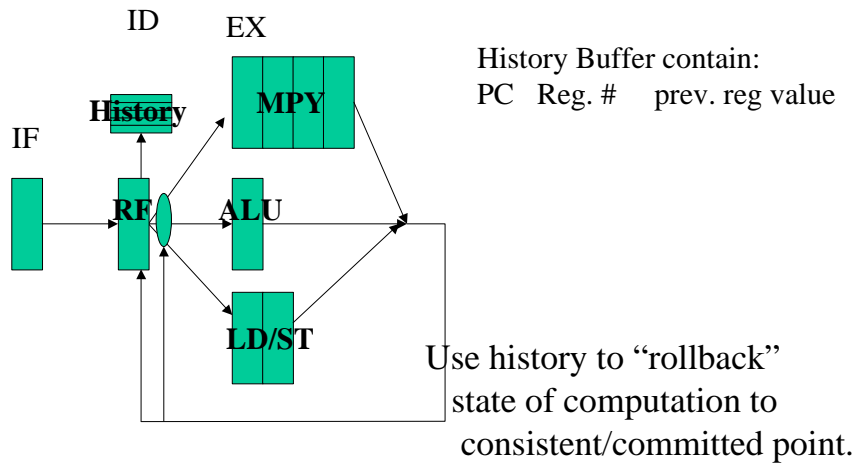
Re-Order



History Buffer

- Keep track of values overwritten in register file
- Can restore old state from there

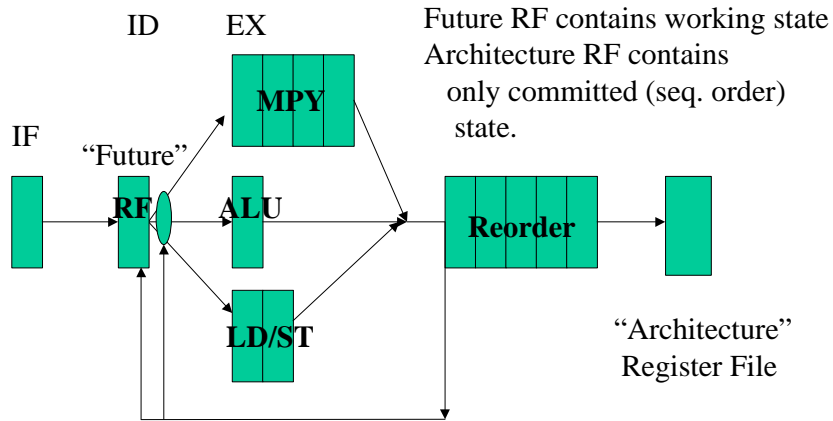
History



Future File

- Keep two copies of register file
 - committed / visible set
 - working set

Future



Memory

- Note: may need to do re-order/bypass to memory as well
 - same issue as RF
 - not want to make visible state change
 - may want to run ahead (avoid adding dep.)
- Bigger issue as we go to longer latencies, OO-issue, etc.

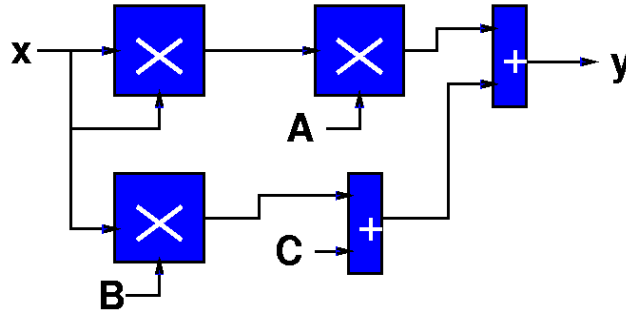
Instruction Level Parallelism

Real Issue

- Sequential ISA Model adds an artificial constraint to the computational problem.
- Original problem (real computation) is not sequentially dependent as a long critical path.
 - Path Length \neq # of instructions

Dataflow Graph

- Real problem is a graph

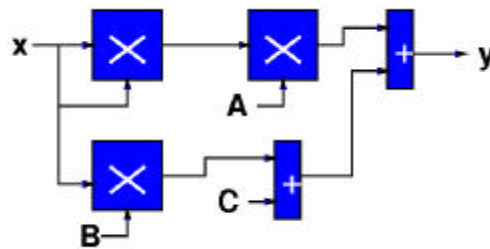


Caltech CS184b Winter2001 -- DeHon

19

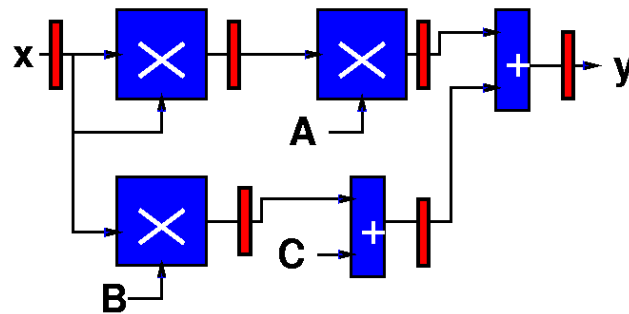
Task Has Parallelism

MPY R3,R2,R2 MPY R4,R2,R5
MPY R3,R6,R3 ADD R4,R4,R7
ADD R4,R3,R4



Caltech CS184b Winter2001 -- DeHon

More when pipelined



- Working on stream (loop)
- may be able to perform all ops at once
 - ...appropriately staggered in time.

Problem

- For sequential ISA:
 - must linearize graph
 - create false dependencies

MPY R3,R2,R2

MPY R4,R2,R5

MPY R3,R6,R3

ADD R4,R4,R7

ADD R4,R3,R4

MPY R3,R2,R2

MPY R3,R6,R3

MPY R4,R2,R5

ADD R4,R4,R7

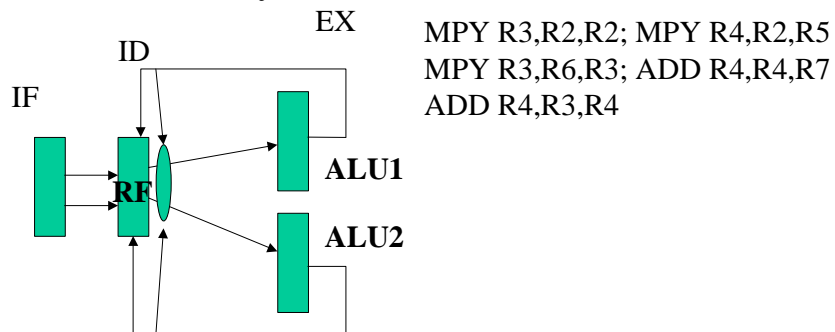
ADD R4,R3,R4

ILP

- The original problem had parallelism
- Can we exploit it?
- Can we rediscover it after?
 - linearizing
 - scheduling
 - assigning resources

If we can find the parallelism...

- ...and will spend the silicon area
- can execute multiple instructions simultaneously



First Challenge: Multi-issue, maintain depend

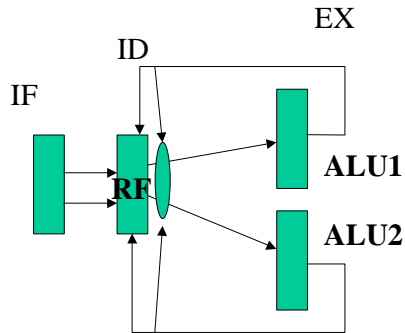
- Like Pipelining
- Let instructions go if no hazard
- Detect (potential hazards)
 - stall for data available

Scoreboarding

- Easy conceptual model:
 - Each Register has a valid bit
 - At issue, read registers
 - If all registers have valid data
 - mark result register invalid (stale)
 - forward into execute
 - else stall until all valid
 - When done
 - write to register
 - set result to valid

Scoreboard

→ MPY R3,R2,R2	2: 1		2: 1
MPY R4,R2,R5	3: 1	R2.valid=1	3: 0
MPY R3,R6,R3	4: 1		4: 1
ADD R4,R4,R7	5: 1	issue	5: 1
ADD R4,R3,R4	6: 1		6: 1
	7: 1	Set R3.valid=0	7: 1

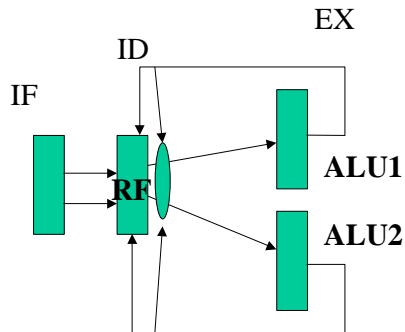


Caltech CS184b Winter2001 -- DeHon

27

Scoreboard

→ MPY R3,R2,R2	2: 1		2: 1
MPY R4,R2,R5	3: 0	R2.valid=1	3: 0
MPY R3,R6,R3	4: 1	R5.valid=1	4: 0
ADD R4,R4,R7	5: 1	issue	5: 1
ADD R4,R3,R4	6: 1		6: 1
	7: 1	Set R4.valid=0	7: 1



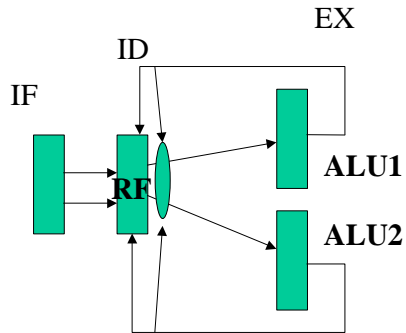
Caltech CS184b Winter2001 -- DeHon

28

Scoreboard

MPY R3,R2,R2
 MPY R4,R2,R5
 → MPY R3,R6,R3
 ADD R4,R4,R7
 ADD R4,R3,R4

2:	1	
3:	0	R3.valid=0
4:	0	R6.valid=1
5:	1	
6:	1	stall
7:	1	



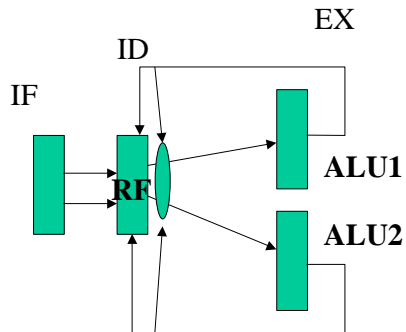
Caltech CS184b Winter2001 -- DeHon

29

Scoreboard

MPY R3,R2,R2
 MPY R4,R2,R5
 → MPY R3,R6,R3
 ADD R4,R4,R7
 ADD R4,R3,R4

2:	1		2: 1
3:	0		3: 1
4:	0	MPY R3	4: 0
5:	1	complete	5: 1
6:	1		6: 1
7:	1	Set R3.valid=1	7: 1



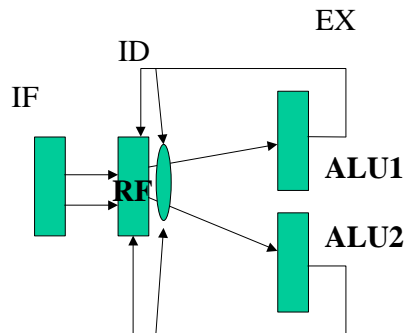
Caltech CS184b Winter2001 -- DeHon

30

Scoreboard

MPY R3,R2,R2
 MPY R4,R2,R5
 → MPY R3,R6,R3
 ADD R4,R4,R7
 ADD R4,R3,R4

2: 1		2: 1
3: 1		3: 0
4: 0	R3.valid=1	4: 0
5: 1	R6.valid=1	5: 1
6: 1	issue	6: 1
7: 1	Set R3.valid=0	7: 1



Caltech CS184b Winter2001 -- DeHon

31

Scoreboard

- Of course, bypass
 - bypass as we did in pipeline
 - incorporate into stall checks
 - so can continue as soon as result shows up
- Also, careful not to issue
 - when result register invalid (WAW)

Caltech CS184b Winter2001 -- DeHon

32

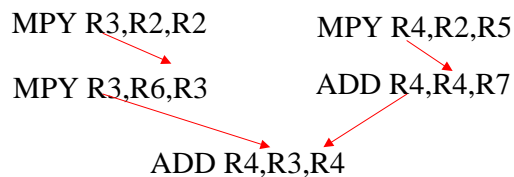
Ordering

- As shown
 - issue instructions in order
 - stall on first dependent instruction
 - get head-of-line-blocking
- Alternative
 - Out of order issue

Example

MPY R3,R2,R2
MPY R4,R2,R5
MPY R3,R6,R3
ADD R4,R4,R7
ADD R4,R3,R4

MPY R3,R2,R2
MPY R3,R6,R3
MPY R4,R2,R5
ADD R4,R4,R7
ADD R4,R3,R4



Example

- This sequence block on in-order issue
 - second instruction depend on first
- But 3rd instruction not depend on first 2.

```
MPY R3,R2,R2
MPY R3,R6,R3
MPY R4,R2,R5
ADD R4,R4,R7
ADD R4,R3,R4
```

Example

- Out of Order
 - look beyond head pointer for enabled instructions
 - issue and scoreboard next found

```
MPY R3,R2,R2
MPY R3,R6,R3
MPY R4,R2,R5
ADD R4,R4,R7
ADD R4,R3,R4
```

MPY R3,R6,R3 stalls for R3 to be computed

MPR4,R2,R5 can be issued while R3 waiting

False Sequentialization on Register Names

- **Problem:** reuse of small set of register names may introduce false sequentialization

```
ADD R2,R3,R4
SW  R2,(R1)
ADD R1,1,R1
ADD R2,R5,R6
SW  R2,(R1)
```

False Sequentialization

- Recognize:
 - register names are just a way of describing local dataflow

```
ADD R2,R3,R4
SW  R2,(R1)
ADD R1,1,R1
ADD R2,R5,R6
SW  R2,(R1)
```

This says:

the result of adding R5 and R6
gets stored into the address pointed
to by R1

R2 only describes the dataflow.

Renaming

- Trick:
 - separate ISA (“architectural”) register names from functional/physical registers
 - allocate a new register on definitions
 - (compare def-use chains in cs134b?)
 - keep track of all uses (until next definition)
 - assign all uses the new register name at issue
 - use new register name to track dependencies, bypass, scoreboarding...

Example



```
ADD R2,R3,R4
SW  R2,(R1)
ADD R1,1,R1
ADD R2,R5,R6
SW  R2,(R1)
```

Rename Table

```
R1: P2
R2: P6
R3: P7
R4: P8
R5: P9
R6: P10
```

Free Table:

```
P1
P3
P4
P11
```

Example

→ ADD R2,R3,R4	Rename Table	Rename Table
SW R2,(R1)	R1: P2	R1: P2
ADD R1,1,R1	R2: P6	R2: P1
ADD R2,R5,R6	R3: P7	R3: P7
SW R2,(R1)	R4: P8	R4: P8
	R5: P9	R5: P9
	R6: P10	R6: P10
Allocate P1 for R2		
Issue: ADD P1,P7,P8	Free Table:	Free Table:
	P1	P3
	P3	P4
	P4	P11
	P11	

Caltech CS184b Winter2001 -- DeHon

41

Example

ADD R2,R3,R4	Rename Table	Rename Table
→ SW R2,(R1)	R1: P2	R1: P2
ADD R1,1,R1	R2: P1	R2: P1
ADD R2,R5,R6	R3: P7	R3: P7
SW R2,(R1)	R4: P8	R4: P8
	R5: P9	R5: P9
	R6: P10	R6: P10
Issue: SW P1,(P2)	Free Table:	Free Table:
	P3	P3
	P4	P4
	P11	P11

Caltech CS184b Winter2001 -- DeHon

42

Example

ADD R2,R3,R4	Rename Table	Rename Table
SW R2,(R1)	R1: P2	R1: P3
→ ADD R1,1,R1	R2: P1	R2: P1
ADD R2,R5,R6	R3: P7	R3: P7
SW R2,(R1)	R4: P8	R4: P8
	R5: P9	R5: P9
	R6: P10	R6: P10
Allocate P3 for P1		
Issue: ADD P3,1,P2	Free Table:	Free Table:
	P3	P2
	P4	P4
	P11	P11

Example

ADD R2,R3,R4	Rename Table	Rename Table
SW R2,(R1)	R1: P3	R1: P3
ADD R1,1,R1	R2: P1	R2: P4
→ ADD R2,R5,R6	R3: P7	R3: P7
SW R2,(R1)	R4: P8	R4: P8
	R5: P9	R5: P9
	R6: P10	R6: P10
Allocate P4 for R2		
Issue: ADD P4,P9,P10	Free Table:	Free Table:
	P2	P2
	P4	P11
	P11	

Example

ADD R2,R3,R4	Rename Table	Rename Table
SW R2,(R1)	R1: P3	R1: P3
ADD R1,1,R1	R2: P4	R2: P4
ADD R2,R5,R6	R3: P7	R3: P7
→ SW R2,(R1)	R4: P8	R4: P8
	R5: P9	R5: P9
	R6: P10	R6: P10
Issue: SW P4,(P3)	Free Table:	Free Table:
	P2	P2
	P11	P11

Free Physical Register

- Free after **complete** last use
- Identify last use by next def?
- Or, allocate in order (LRU)
 - interlock if re-assignment conflict
 - (should correspond to having no free physical registers)

Tomasulo

- Register renaming
- Scoreboarding
- Bypassing

- IBM 1967

- ...what's keeping x86 ISA alive today
 - compensate for small number of arch. Registers
 - dusty deck code

Caltech CS184b Winter2001 -- DeHon

47

Today

- Seen can turn a basic block
 - (code between branches)
- Into executing dataflow graph
 - I.e. once issues, only dataflow dependencies limit parallelism

- ...all the more reason to want large basic blocks (minimize branch, branch effects)

Caltech CS184b Winter2001 -- DeHon

48

Reading Note

- Today: HP4.1-2, Tomasulo
- Next Week:
 - rest of HP4
 - Fisher/predict relevant
 - probably touch on Tuesday
 - Subbarao Quantifying...
 - probably Thursday
- Following Week: VLIW and EPIC
 - Fisher, IA-64...

Big Ideas

- Data Versioning
 - keep old copies, until commit
 - working versus finalized
- Parallelism **does** exist in the problem
 - obscured by ISA linearization
- Dataflow Interpretation
 - preserve dependencies, not control flow sequence
 - rediscover non-linear “graph”