

CS184b: Computer Architecture (Abstractions and Optimizations)

Day 7: April 11, 2005
Instruction Level Parallelism
ILP 1



Caltech CS184 Spring2005 -- DeHon

Today

- ILP – beyond 1 instruction per cycle
 - Parallelism
 - Dynamic exploitation
 - Scoreboarding
 - Register renaming
 - Control flow
 - Prediction
 - Reducing

2

Caltech CS184 Spring2005 -- DeHon

Real Issue

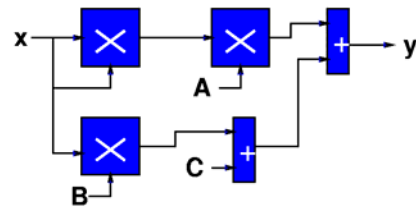
- Sequential ISA Model adds an artificial constraint to the computational problem.
- Original problem (real computation) is **not** sequentially dependent as a long critical path.
 - Path Length != # of instructions

3

Caltech CS184 Spring2005 -- DeHon

Dataflow Graph

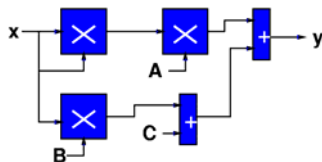
- Real problem is a graph



Caltech CI

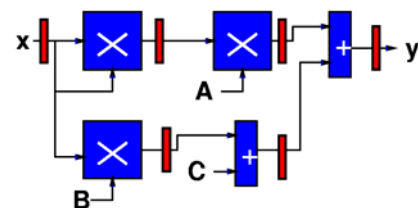
Task Has Parallelism

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



Caltech CS184 Spring2005 -- DeHon

More when pipelined



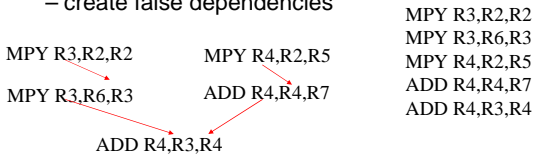
- Working on stream (loop)
- may be able to perform all ops at once
 - ...appropriately staggered in time.

6

Caltech CS184 Spring2005 -- DeHon

Problem

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



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

Caltech CS184 Spring2005 -- DeHon

7

ILP

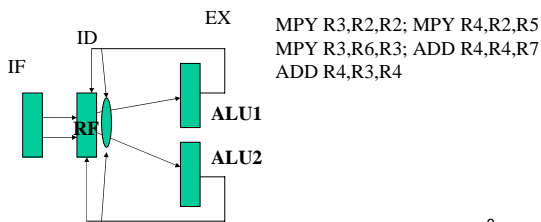
- The original problem had parallelism
- Can we exploit it?
- Can we rediscover it after?
 - linearizing
 - scheduling
 - assigning resources

Caltech CS184 Spring2005 -- DeHon

8

If we can find the parallelism...

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



Caltech CS184 Spring2005 -- DeHon

9

First Challenge: Multi-issue, maintain depend

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

Caltech CS184 Spring2005 -- DeHon

10

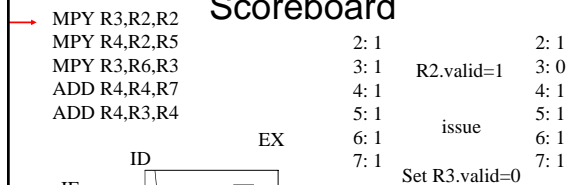
Scoreboarding

- 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

Caltech CS184 Spring2005 -- DeHon

11

Scoreboard



Caltech CS184 Spring2005 -- DeHon

12

Scoreboard

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

Set R4.valid=0

Caltech CS184 Spring2005 -- DeHon

13

Scoreboard

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

Caltech CS184 Spring2005 -- DeHon

14

Scoreboard

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

Caltech CS184 Spring2005 -- DeHon

15

Scoreboard

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

Caltech CS184 Spring2005 -- DeHon

16

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 CS184 Spring2005 -- DeHon

17

Ordering

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

Caltech CS184 Spring2005 -- DeHon

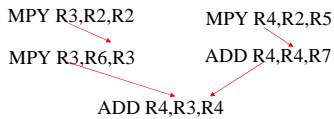
18

Example

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

Another linearization →

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



Caltech CS184 Spring2005 -- DeHon

19

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
```

Caltech CS184 Spring2005 -- DeHon

20

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
MPY R4,R2,R5 can be issued while R3 waiting

Caltech CS184 Spring2005 -- DeHon

21

False Sequentialization on Register Names

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

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

Caltech CS184 Spring2005 -- DeHon

22

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.

Caltech CS184 Spring2005 -- DeHon

23

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...

Caltech CS184 Spring2005 -- DeHon

24

→

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

Caltech CS184 Spring2005 -- DeHon

25

→

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	Free Table:	Free Table:
Issue: ADD P1,P7,P8	P1	P3
	P3	P4
	P4	P11
	P11	

Caltech CS184 Spring2005 -- DeHon

26

→

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 CS184 Spring2005 -- DeHon

27

→

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 R1	Free Table:	Free Table:
Issue: ADD P3,1,P2	P3	P4
	P4	P11
	P11	

Caltech CS184 Spring2005 -- DeHon

28

→

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	Free Table:	Free Table:
Issue: ADD P4,P9,P10	P4	P11
	P11	

Caltech CS184 Spring2005 -- DeHon

29

→

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:
	P11	P11

Caltech CS184 Spring2005 -- DeHon

30

Free Physical Register

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

Caltech CS184 Spring2005 -- DeHon

31

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 CS184 Spring2005 -- DeHon

32

Parallelism and Control

- 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 CS184 Spring2005 -- DeHon

33

Avoiding Control Flow Limits

Caltech CS184 Spring2005 -- DeHon

34

Control Flow

- Previously saw data hazards on control force stalls
 - for multiple cycles
- With superscalar, may be issuing multiple instructions per cycle
- Makes stalls even more expensive
 - wasting n slots per cycle
 - *e.g.*
 - with 7 instructions / branch
 - issue 7 instructions, hit branch, stall for instructions to complete...

Caltech CS184 Spring2005 -- DeHon

35

Control/Branches

- Cannot afford to stall on branches for resolution
- Limit parallelism to basic block
 - average run length between branches
 - ...which is typically 5--7

Caltech CS184 Spring2005 -- DeHon

36

Idea

- Predict branch direction
- Execute as if that's correct
- Commit/discard results after know branch direction
- Use ideas from precise exceptions to separate
 - working values
 - architecture state

Caltech CS184 Spring2005 -- DeHon

37

Goal

- Correctly predicted branches run as if they weren't there (noops)
- Maximize the expected run length between mis-predicted branches

Caltech CS184 Spring2005 -- DeHon

38

Expected Run Length

- $E(l) = L1 + L2 \cdot P1 + L3 \cdot P1 \cdot P2 + L4 \cdot P1 \cdot P2 \cdot P3$
- $L_i = L, P_i = p$
- $E(l) = L(1 + p + p^2 + p^3 + \dots)$
- $E(l) = L/(1-p)$
- $E(l) = L/(\text{probability of mispredict})$

Caltech CS184 Spring2005 -- DeHon

39

Expected Run Length

- $P=0.9$ $10L$
- $p=0.95$ $20L$
- $p=0.98$ $50L$
- $p=0.99$ $100L$
- Halving mispredict rate \rightarrow doubles run length

Caltech CS184 Spring2005 -- DeHon

40

IPC

- Run for $E(l)$ instructions
- Then mispredict
 - waste ~ pipeline delay cycles (and all work)
- Pipe delay: d
- Base IPC: n
- $E(l)/n$ cycles issue n
- d cycles issue nothing useful
- $IPC = E(l)/(E(l)/n + d) = n/(1 + dn/E(l))$

Caltech CS184 Spring2005 -- DeHon

41

Example

- $IPC = E(l)/(E(l)/n + d) = n/(1 + dn/E(l))$
- Say $E(l) = 100, n = 7, d = 10$
- $7/(1 + 70/100) = 7/1.7 \approx 4$

Caltech CS184 Spring2005 -- DeHon

42

Branch Prediction

- Previous runs
- (dynamic) History
- Correlated

Caltech CS184 Spring2005 -- DeHon

43

Previous Run

- **Hypothesis:** branch behavior is largely a characteristic of the **program**.
 - Can use data from previous runs (different input data set)
 - to predict branch behavior
- Fisher: Instructions/mispredict: 40-160
 - even with different data sets

Caltech CS184 Spring2005 -- DeHon

44

Data Prediction

- Example shows value (and validity) of feedback
 - run program
 - measure behavior
 - use to inform compiler, generate better code
- Static/procedural analysis
 - often cannot yield enough information
 - most behavioral properties are undecidable

Caltech CS184 Spring2005 -- DeHon

45

Branch History Table

- **Hypothesis:** we can predict the behavior of a branch based on its recent past behavior.
 - If a branch has been taken, we'll predict it's taken this time.
- To exploit dynamic strength, would like to be responsive to changing program behavior.

Caltech CS184 Spring2005 -- DeHon

46

Branch History Table

- Implementation
 - Saturating counter
 - count up branch taken; down on branch not taken
 - Predict direction based on majority (which side of mid-point) of past branches
- Saturation
 - keeps counter small (cheap)
 - typically 2b
 - limits amount of history kept
 - time to "learn" new behavior

Caltech CS184 Spring2005 -- DeHon

47

Correlated Branch Prediction

- **Hypothesis:** branch directions are highly correlated
 - a branch is likely to depend on branches which occurred before it.
- Implementation
 - look at last m branches
 - shift register on branch directions
 - use a separate counter to track each of the 2^m cases
 - contain cost: only keep a small number of entries and allow aliasing

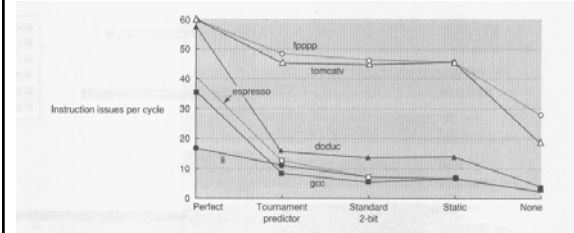
Caltech CS184 Spring2005 -- DeHon

48

Branch Prediction

- ...whole host of schemes and variations proposed in literature

Branch Prediction



[Hennessey & Patterson Fig 3.38/e3] 50

Prediction worked for Direction...

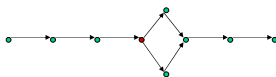
- Note:
 - have to decode instruction to figure out if it is a branch
 - then have to perform arithmetic to get branch target
- So, don't know where the branch is going until cycle (or several) after fetch
 - IF ID EX

Branch-Target Buffer

- Take it one step back and predict target address
- Cache
 - in parallel with Memory Fetch (IF)
 - stores predicted target PC
 - and branch prediction
 - tagged with PC to avoid aliasing

Reducing Number of Branches

- A mispredicted branch costs more than a few cycles in these wide-issue machines
 - potentially $n \cdot d$
- Especially in cases of reconvergent flow and even branch probabilities



Conditional Operations

- **Idea:** create guarded operations
 - only change register if some result holds
- *e.g.*
 - 8b saturating add
 - $c = a + b$
 - if $(t1 > 255) c = 255$
 - if $(t1 < 0) c = 0$

```
ADD R1,R2,R3
SUB R4,R1,#255
CMOVP R1,#255,R4
COMVN R1,#0,R1
```

Two Control Options

1. Local control

- unify choices
 - build all options into spatial compute structure and select operation

2. Instruction selection

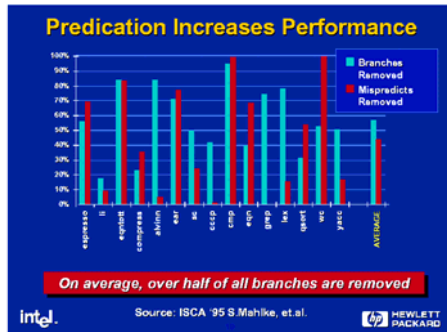
- provide a different instruction (instruction sequence) for each option
- selection occurs when choose which instruction(s) to issue

Conditional execution is local control.

Conditional Operation Prospect

- For unpredictable branch ($p \approx 0.5$)
 - $E(\text{wasted issue slots}) = p * n * d$ ($n * d / 2$)
- With conditional move
 - assume l cycles inside conditional clauses
 - one sided if:
 - $E(\text{wasted}) = p * l$ e.g. ($l/2$)
 - two sided (both length l)
 - $E(\text{wasted}) = l$
- Net benefit for short guarded blocks
 - on wide issue machines

Impact of Conditional Operations



Speculation

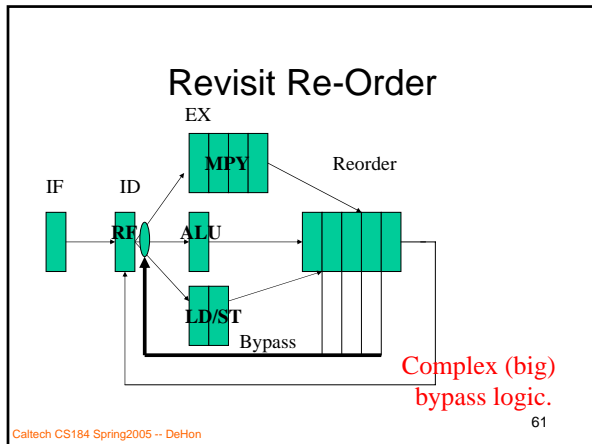
- Branch prediction allows us to continue executing
- still have to deal with branch being wrong
- In simple pipelined ISA
 - outstanding branch resolved before writeback

Speculation

- Wide-issue ISA?
 - Likely to have more instructions in flight than mean latency between branches ($nd > l$)
 - to exploit parallelism, need to continue computing assuming the chosen path is correct
 - means making result values visible to subsequent instructions which may be wrong if control flow goes another way

Old Problem

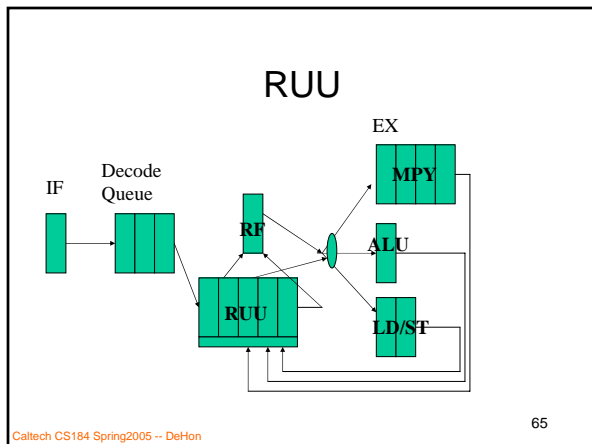
- Mostly the same problem as precise exceptions
 - want to continue computing forward with tentative values
 - want to preserve old state so can roll-back to known state



- ### Speculation and Re-Order Buffer
- Compute and bypass values from re-order buffer
 - At end of re-order buffer
 - commit to RF (architectural state) in proper program order after branches resolved
 - if branch wrong,
 - nullify its effect (results predicated upon)
 - flush re-order buffer (pipeline)
 - direct control flow back to correct branch direction
- Caltech CS184 Spring2005 -- DeHon 62

- ### Details
- As before,
 - exception delivery must be deferred until can commit instruction
 - memory operations require re-order/bypass/commit as well
 - History/Future File work
 - ...but transfer time may be more critical in this case
- Caltech CS184 Spring2005 -- DeHon 63

- ### Register Update Unit (RUU)
- Simple scalar uses this
 - FIFO unit for instruction management serves for both issue and in-order commit
 - Decode: fills empty slots
 - Issue: picks next set of runnable instructions
 - Execution results return here
 - Commit: completed instructions in order from head of FIFO
- Caltech CS184 Spring2005 -- DeHon 64



- ### RUU
- Needs to hold all outstanding instructions
 - from: considering for issue
 - to: completion and final RF writeback
 - Needs to be relatively large
 - Complex?
- Caltech CS184 Spring2005 -- DeHon 66

Admin

- Feedback for deLorimier, Caspi lecture
- P2: project
- W: Michael on SMVM
 - GM style application
 - Project network design relevant to
- F: ILP2
- M,W: no class
 - Will be new assignment out M

Caltech CS184 Spring2005 -- DeHon

67

Big Ideas

- Parallelism **does** exist in the problem
 - obscured by ISA linearization
- Dataflow Interpretation
 - preserve dependencies, not control flow sequence
 - rediscover non-linear “graph”

Caltech CS184 Spring2005 -- DeHon

68

Big Ideas

- Interruptions in Control Flow limit our ability to exploit parallelism
- There is structure in programs
 - predictability in control flow
- Make the common case fast
- Predict/guess common case control flow
 - to generate larger blocks
- Nullify effects of erroneous instructions when guess wrong

Caltech CS184 Spring2005 -- DeHon

69