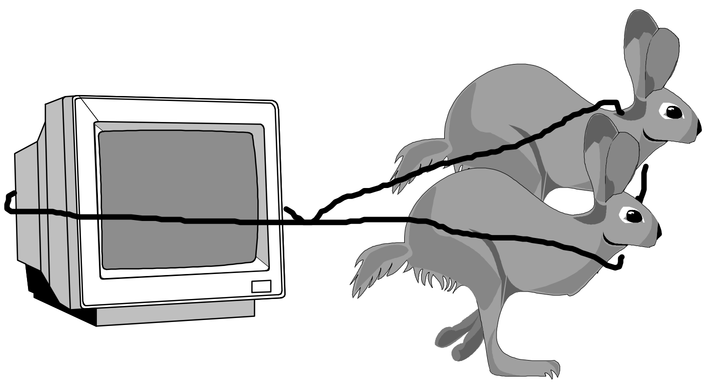
6004 Spring 1998: L51

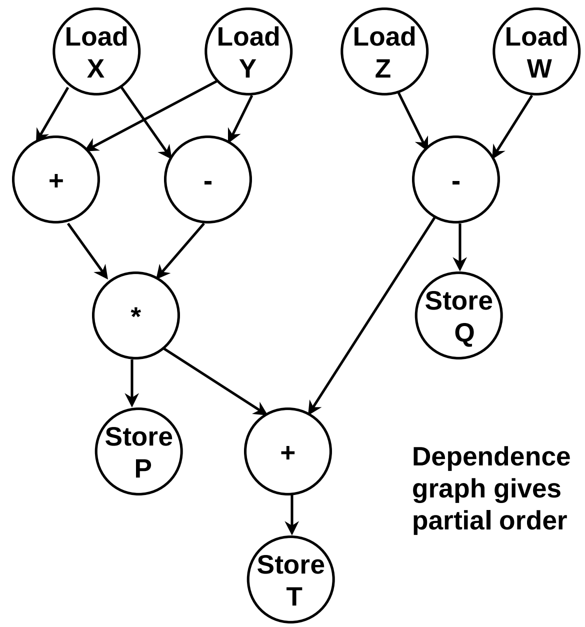
Instruction-Level Parallelism



1

Data Dependence Graph

$P = (X + Y) * (X - Y)$
$Q = Z - W$
$T = P + Q$



Dependence graph gives partial order

2

6004 Spring 1998: L51

Compiled Assembly Code

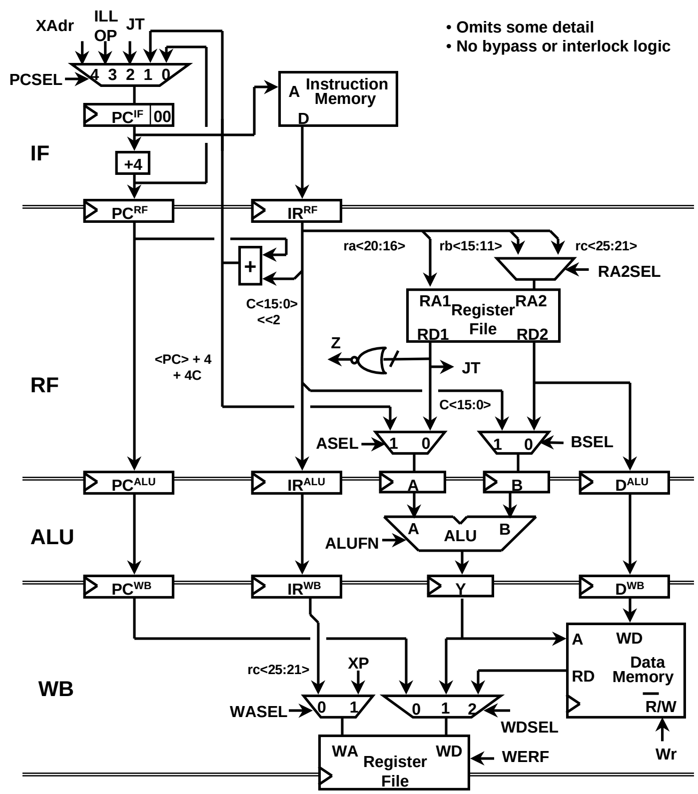
LD(r31, X, r1)
LD(r31, Y, r2)
ADD(r1, r2, r3)
SUB(r1, r2, r4)
MUL(r3, r4, r5)
ST(r5, P, r31)
LD(r31, Z, r6)
LD(r31, W, r7)
SUB(r6, r7, r8)
ST(r8, Q, r31)
ADD(r5, r8, r9)
ST(r9, T, r31)

Compiler selects a <u>total</u> order

Does it matter what total order is selected?

3

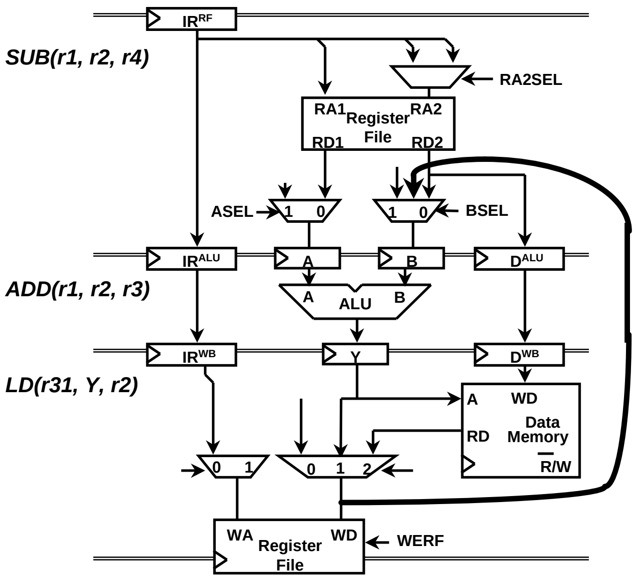
4-Stage β Pipeline



• Omits some detail
• No bypass or interlock logic

4

Load Bypass Paths



Even with full bypassing we need _____ cycle(s) of stalling between *LD(r31, Y, r2)* and *ADD(r1, r2, r3)*

Avoiding Stalls

Stalls

LD(r31, X, r1)		*LD(r31, X, r1)*
LD(r31, Y, r2)		*LD(r31, Y, r2)*
ADD(r1, r2, r3)		*LD(r31, Z, r6)*
SUB(r1, r2, r4)		*LD(r31, W, r7)*
MUL(r3, r4, r5)		*ADD(r1, r2, r3)*
ST(r5, P, r31)		*SUB(r1, r2, r4)*
LD(r31, Z, r6)	➡	*MUL(r3, r4, r5)*
LD(r31, W, r7)		*SUB(r6, r7, r8)*
SUB(r6, r7, r8)		*ST(r5, P, r31)*
ST(r8, Q, r31)		*ADD(r5, r8, r9)*
ADD(r5, r8, r9)		*ST(r8, Q, r31)*
ST(r9, T, r31)		*ST(r9, T, r31)*

Average CPI = ____ **Average CPI = ____**

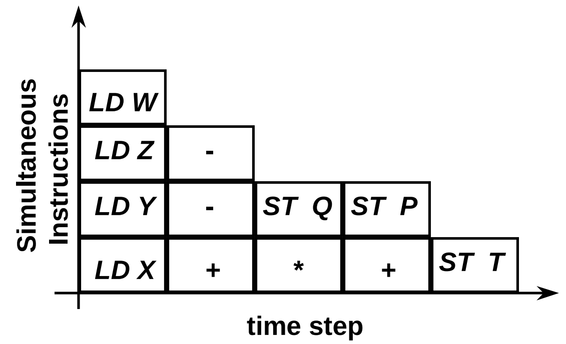
Finding the best ordering of instructions to minimize (or avoid) stalls is a <u>scheduling</u> problem

Questions

- When can we hide load latency and thus avoid stalls?
- Could we build a processor such that CPI < 1.0 ?

Instruction-Level Parallelism

- Suppose we had a machine which could execute lots of instructions per cycle, limited only by data dependences

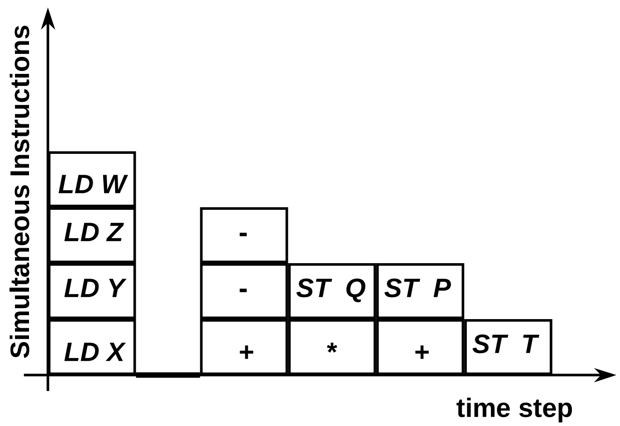


Average CPI = 5/12

- No load or store stall cycles gives us an idealized <u>parallelism profile</u>

Accounting for Load Latency

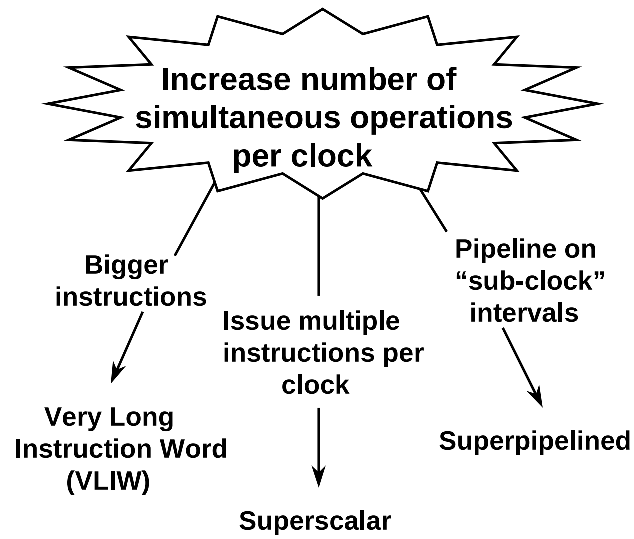
- Assume 1 stall cycle for loads



Average CPI = 6/12

- We cannot complete in fewer than 6 steps

Realizing CPI < 1.0

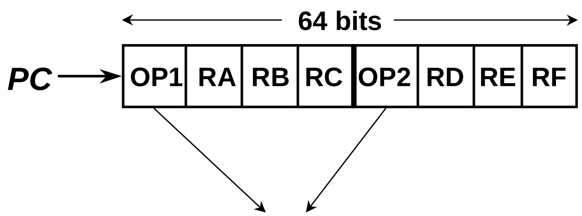


All require instruction-level parallelism

6004 Spring 1998: L51

VLIW Architecture

- Examples are Multiflow, i860



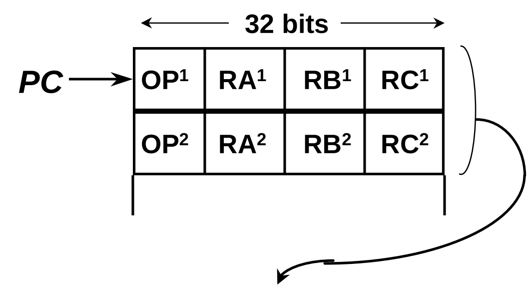
Usually many restrictions
e.g., OP1 = { *LD*, *ST*, +, - }
OP2 = { +, -, *, logical }

- Compiler <u>statically</u> combines instructions such that data dependences are obeyed

11

Superscalar Architecture

- Examples are SuperSparc, IBM Power



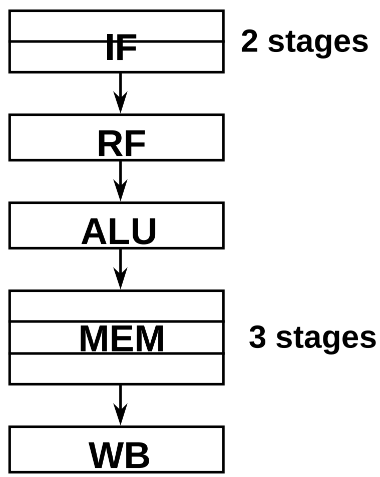
Issue both if:

- Hardware dynamically combines instructions such that data dependences are obeyed

12

Superpipelined Architecture

- Example is MIPS R4000
- Just a fancy name for > 4 deep pipelines
  - – Run clock faster



- Number of load stall cycles = 4

13

Architecture Characteristics

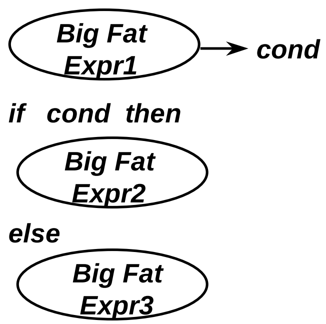
Backward compatibility means old (serial) machine COMPILED code can run on new (parallel) machine

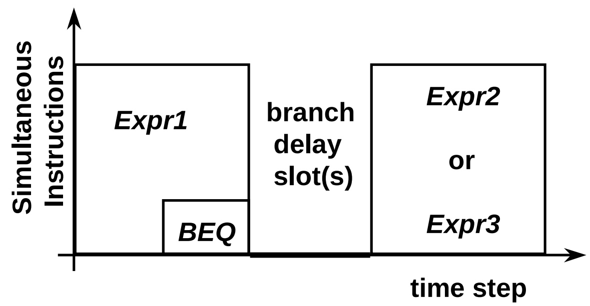
	Hardware Complexity	Backward Compatible
VLIW	Medium	____
Superscalar	High	____
Superpipelined	Low	____

14

6004 Spring 1998: L51

Branches

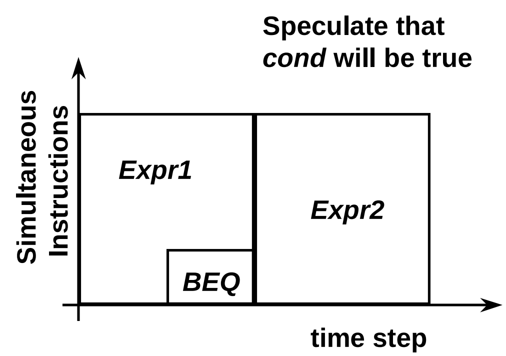




Branch penalty in terms of instructions increases as machine parallelism increases

Speculative Execution

- Statistically avoid branch delay slots by guessing which way it will go, and then execute *Expr2* or *Expr3* speculatively



- Must be able to undo *Expr2* if we find that *cond* was false!

Implementation

Guessing Algorithm
- static → **Compiler hints**
- dynamic → **What happened last time?**

+

Target Instruction Insertion Algorithm
- static → **Compiler hints**
- dynamic → **Target Instruction Cache in RF stage**

+

Undoing Algorithm
- hardware → **History buffers in RF stage**
- software → **Explicit fixup code**

17