Pipelining				
Sandhya Dwarkadas				
A Pipelined Datapath				
Multiple instructions overlapped in execution				
Improve throughput, not individual instruction execution time				
Exploit parallelism among instructions in a sequential stream				
Balance length of each stage. Ideally - $Time between instrs_{pipelined} = \frac{Time between instrs_{non-pipelined}}{Number of pipe stages}$				

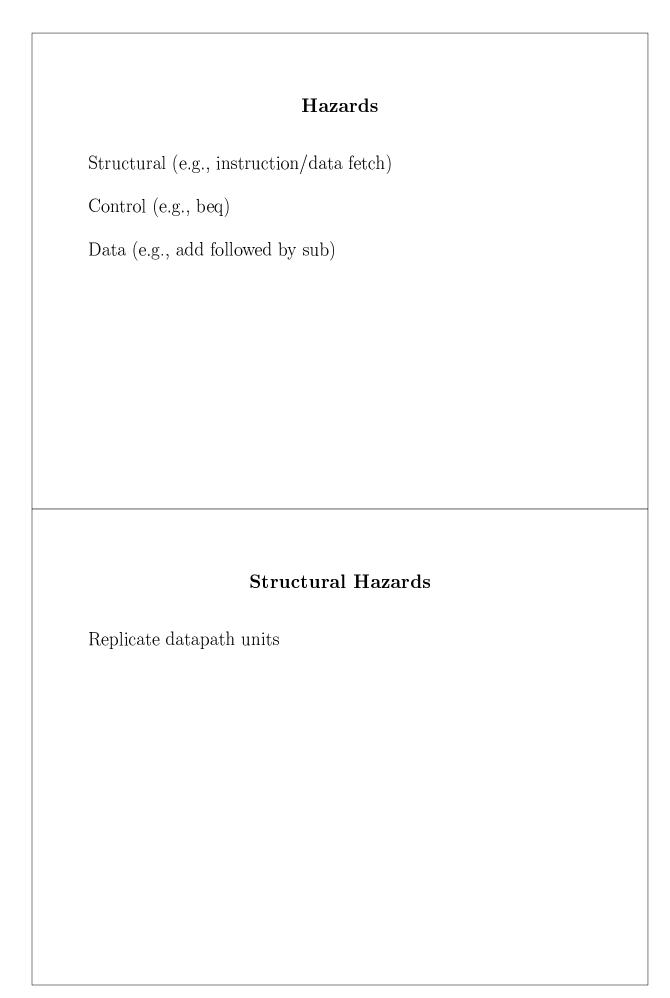
MIPS Advantages

- Instructions of the same length
- Few instruction formats
- \bullet Memory operands only in ld and $st \Rightarrow$ can use ALU for address computation
- Operands aligned in memory ⇒ only one access per instruction
- \bullet Single result computed per instruction

Additional Datapath Units?

Each component used within a single pipeline stage

To pass data from earlier to later stage, must place info in pipeline register



Control Hazards
Control Hazards
Stall
Predict
Delayed instruction
Data Hazard
Forward or bypass
Delayed instruction

Instruction-Level Parallelism

Super-pipelining

Dynamic pipelining

Very Large Instruction Word (VLIW)

Super-Scalar

Super-pipelining

Increase the number of pipeline stages

Fundamentally limited by latch speed and hazards

Dynamic Pipelining

Execute instructions out-of-order

- Hide memory latency
- Avoid stalls that the compiler could not avoid
- Speculatively execute instructions while waiting for hazards to be resolved

Types of Data Hazards

RAW - true dependence

WAR - anti dependence

WAW - Output dependence

Data Hazards: Forwarding

- The CDC 6600 Score-board architecture
- \bullet Tomasulo's generalized forwarding algorithm in the IBM360
 - used reservation stations and shared buses

Super-Scalar

Multiple instructions per pipeline stage

- in-order or out-of-order execution
- \bullet in-order or out-of-order completion

T 7	ГΤ	X.	7
v		W	/

Multiple instructions per pipeline stage

Dependences taken care of by the compiler

Availability of hardware resources assured

Need to recompile source even to run on the same ISA

Advanced Processor Design Techniques

Register Renaming - eliminate WAW, WAR hazards

Speculative execution

Branch prediction

Summary

Dynamic superscalar designs are dramatically more complex than scalar (and even in order superscalar)

The effort thus far seems to be worth it (Dynamic 21264~2x performance of in order 21164 (although some gains are due to a better memory system)

Focus seems to be shifting to memory performance imbalance