Instruction Level Parallelism (ILP)

Colin Stevens

Instruction-Level Parallelism (ILP)

- **Pipelining** exploits the potential **parallelism** among instructions. This parallelism is called **instruction-level parallelism (ILP)**.
- Pipelining: executing multiple instructions in parallel
- To increase ILP
  - Deeper pipeline
    - Less work per stage ⇒ shorter clock cycle
  - Multiple issue
    - Replicate pipeline stages ⇒ multiple pipelines
    - Start multiple instructions per clock cycle
    - CPI < 1, so use Instructions Per Cycle (IPC)
    - E.g., 4GHz 4-way multiple-issue
      - 16 BIPS, peak CPI = 0.25, peak IPC = 4
    - But dependencies reduce this in practice

What is a parallel instruction?

- ILP is a measure of the number of instructions that can be performed during a single clock cycle.
- Parallel instructions are a set of instructions that do not depend on each other to be executed.

IPC

- Launching multiple instructions per stage allows the instruction execution rate to exceed the clock rate or, stated alternatively, the CPI to be less than 1.
- it is sometimes useful to fl ip the metric and use *IPC*, or *instructions per clock cycle*.
- A 4 GHz four-way multiple-issue microprocessor can execute a peak rate of 16 billion instructions per second and have a best-case CPI of 0.25,
- or an IPC of 4.

Implementations of ILP

- Pipelining
- Superscalar Architecture
  - Dependency checking on chip.
  - Multiple Processing Elements eg. ALU, Shift
- VLIW (Very Long Instruction Word Architecture)
  - Simple hardware, Complex Compiler
- Multi processor computers

- There are two major ways to implement a multiple-issue processor, with the major difference being the division of work between the compiler and the hardware
- The division of work dictates whether decisions are being made statically (that is, at compile time) or dynamically (that is, during execution), the approaches are sometimes called **static multiple issue** and **dynamic multiple issue**.

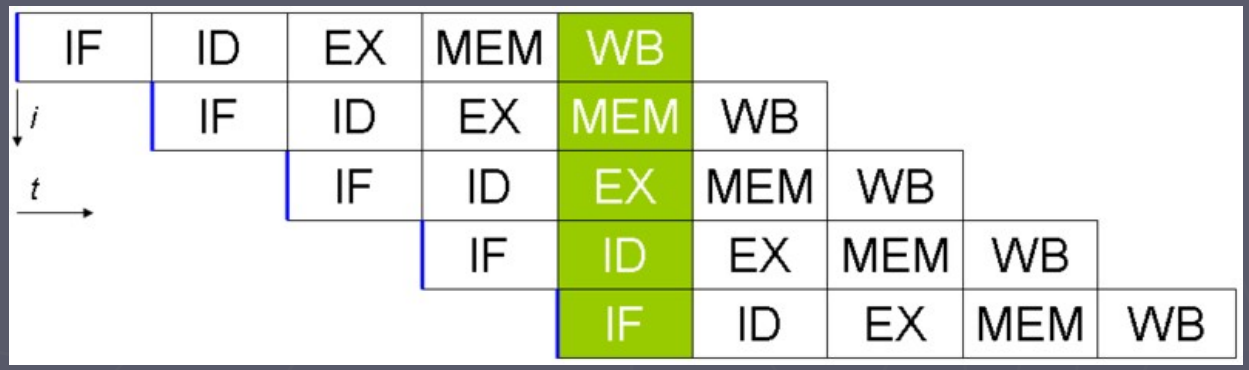
Multiple Issue

- Static multiple issue
  - Compiler groups instructions to be issued together
  - Packages them into "issue slots"
  - Compiler detects and avoids hazards
- Dynamic multiple issue
  - CPU examines instruction stream and chooses instructions to issue each cycle
  - Compiler can help by reordering instructions
  - CPU resolves hazards using advanced techniques at runtime

Dynamic Multiple-Issue Processors

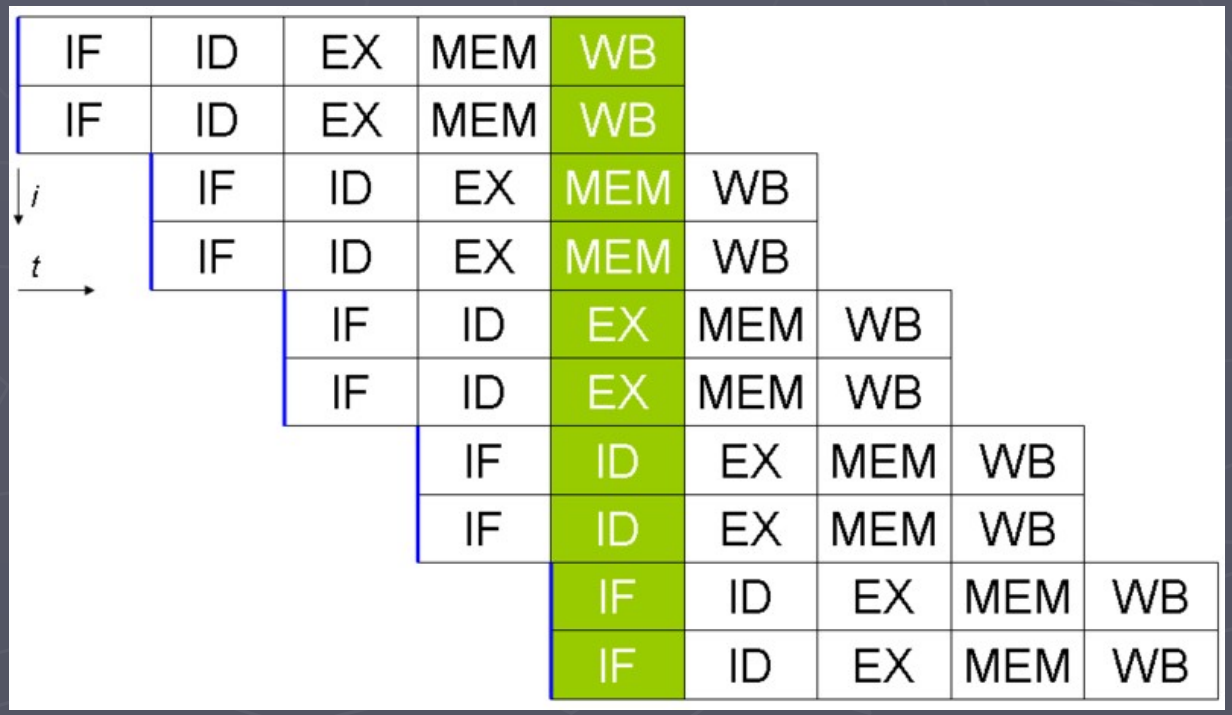
- Dynamic multiple-issue processors are also known as **superscalar** processors, or simply superscalars.
- In the simplest superscalar processors, instructions issue in order, and the processor decides whether zero, one, or more instructions can issue in a given clock cycle.
- Obviously, achieving good performance on such a processor still requires the compiler to try to schedule instructions to move dependences apart and thereby improve the instruction issue rate.

- Pipelining



http://en.wikipedia.org/wiki/Image:Fivestagespipeline.png

- Superscalar



http://en.wikipedia.org/wiki/Image:Superscalarpipeline.png

Concluding Remarks

- ISA influences design of datapath and control
- Datapath and control influence design of ISA
- Pipelining improves instruction throughput using parallelism
  - More instructions completed per second
  - Latency for each instruction not reduced
- Hazards: structural, data, control
- Multiple issue and dynamic scheduling (ILP)
  - Dependencies limit achievable parallelism
  - Complexity leads to the power wall