Pipelining

Implementation technique (but is part of the architecture)

- overlaps execution of different instructions
  - execute all execution cycle phases simultaneously on different instructions
- increases instruction throughput (alot) by executing several instructions “in parallel”

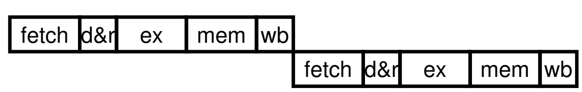
$$\textbf{optimal speedup} = \frac{T_{\text{without pipe}}}{T_{\text{with pipe}}} = \frac{i \times n}{i + n - 1} \approx \text{\# of pipe stages}$$

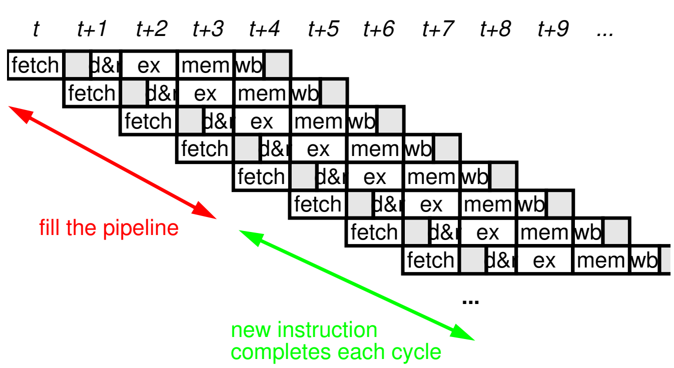
Not that simple!

- pipeline hazards (structural, data, control)
  - “limit” the number of stages
- increase instruction latency (a little)
  - pipeline registers for data
  - time for control to reach the later stages
  - clock skew
  - unequal work in pipe stages

IBM Stretch (1961): the first general-purpose pipelined computer

Pipelining





Hazards

Structural hazards

Data hazards

Control hazards

What happens on a hazard

- instruction that caused the hazard & previous instructions complete
- all subsequent instructions stall until the hazard is removed

Structural Hazards

Cause: instructions in different stages want to use the same resource in the same cycle

Solutions:

- more hardware (eliminate the hazard)
- stall (execute correct programs)
    - less hardware, lower cost
    - not really an option any more

Data Hazards

Cause:

- the result of an instruction further in the pipeline is needed by an instruction later in the pipeline
- delay between access & use of operand
  e.g., delayed load

Types

RAW (flow), WAR (anti) , WAW (output)

HW solutions

- forwarding hardware (eliminate the hazard)
- pipelined interlock (take the hazard out of the architecture)

Compiler solution

- code scheduling (for delayed loads)

Dynamic Scheduling

Eliminating WAW & WAR hazards

- flow dependence (potential RAW hazard)

```
ADDF F0, F1, F2
SUBF _, F0, _
```

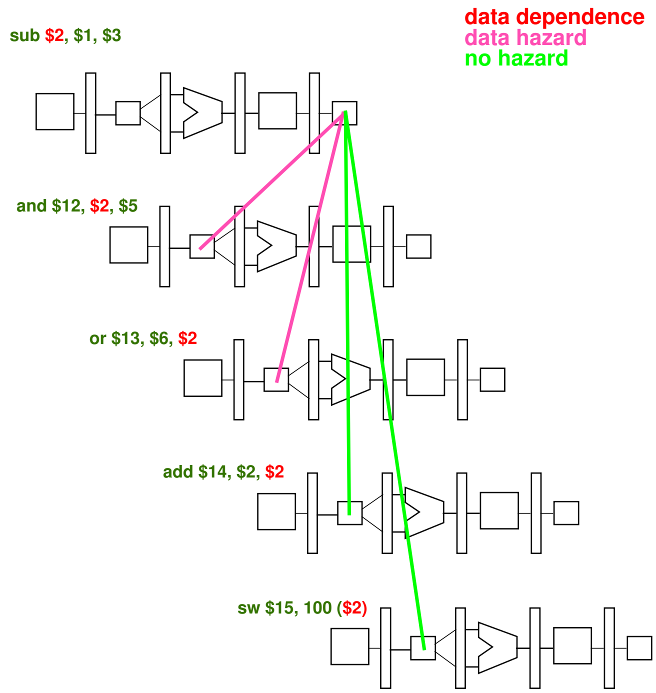
- anti-dependence (potential WAR hazard)

```
ADDF _, F1, _
MULTF F1, _, _
```

- output dependence (potential WAW hazard)

```
ADDF F0, F1, F2
MULTF F0, F2, F3
```

Dependences vs. Hazards



Control Hazards

Cause: condition & target determined after next fetch

(delayed branch: M)

Early HW solutions

- stall (who on earth does this now?)
- assume an outcome
- redesign pipeline (A M S X)

Compiler solutions

- code scheduling (delayed branches)
- static branch prediction (X) + cancelling branches (M S)