Introduction to Pipelining

- *Latency vs. throughput*
- *Latency*
    - *Each instruction takes a certain time to complete.*
    - *This is the latency for that operation.*
    - *It's the amount of time between when the instruction is* **issued** *and when it completes.*
- *Throughput*
    - *The number of instructions that complete in a span of time.*
    - *This is not necessarily the same as dividing the time span by the latency if pipelining is used.*

Pipelining

- **Definition**
    - *Pipelining is the ability to overlap execution of different instructions at the same time.*
        - *It exploits parallelism among instructions and is* **NOT** *visible to the programmer.*
    - *This is similar to building a car on an assembly line.*
        - *While it may take two hours to build a single car, there are hundreds of car in progress at any time.*

  - *The throughput of the assembly line is the # of cars completed per hour.*
  - *The throughput of a CPU pipeline is the # of instructions completed per second.*

- **Pipeline stages**
  - *Each step in a pipeline is called a pipe stage .*
  - *In our assembly line example, a stage corresponds to a work station on the assembly line.*

Pipelining

- **Cycle time**
  - *Everything in a CPU moves in lockstep, synchronized by the clock ("heartbeat" of the CPU.)*
  - *A machine cycle : time required to complete a single pipeline stage.*
    - *A machine cycle is usually one, sometimes two, clock cycles long, but rarely more.*
  - *In machines with* **no** *pipelining:*

- *The machine cycle must be long enough to complete a single instruction*
- *Or each instruction must be divided into smaller chunks (multiple clock cycles per instruction).*

- **Pipeline cycle time**
  - *All pipeline stages must, by design, take the same time.*
    - *Thus, the machine cycle time is that of the* **longest** *pipeline stage.*
  - *Ideally, all stages should be exactly the same length.*

Pipelining

- **Pipeline speedup**
  - *The ideal speedup from a pipeline is equal to the number of stages in the pipeline.*

$$\frac{\text{Time per instruction on unpipelined machine}}{\text{Number of pipe stages}}$$

  - *However, this only happens if the pipeline stages are all of equal length.*
    - *Splitting a 40 ns operation into 5 stages, each 8 ns long, will result in a 5x speedup.*
    - *Splitting the same operation into 5 stages, 4 of which are 7.5 ns long and one of which is 10 ns long will result in only a 4x speedup.*
  - *If your starting point is a multiple clock cycle per instruction machine then pipelining decreases CPI.*
  - *If your starting point is a single clock cycle per*

  instruction machine then pipelining decreases cycle time.
  - *We will focus on the first starting point in our analysis.*

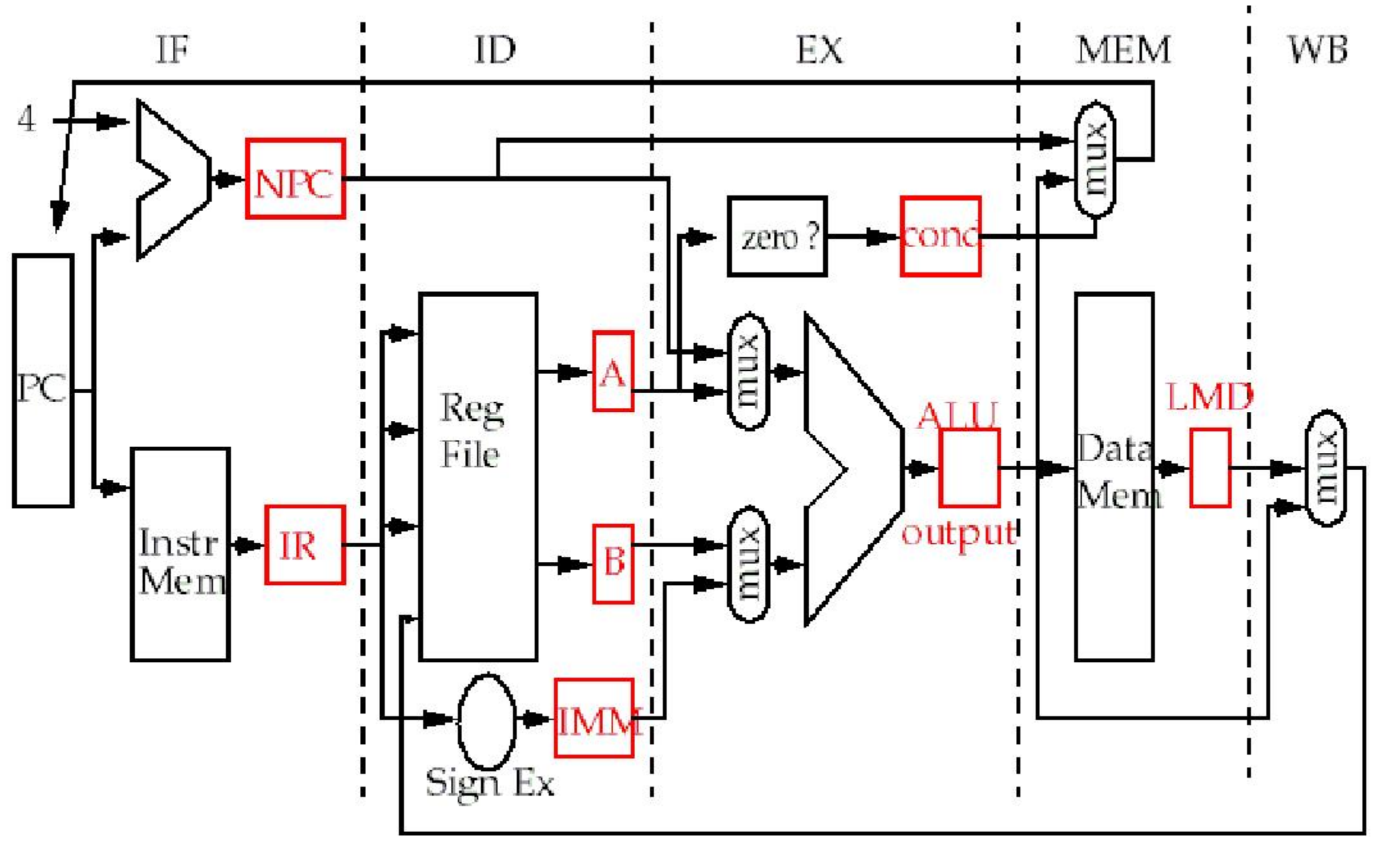
Simple DLX operation (without pipelining)

- *Each DLX instruction has five phases.*
  - *Thus, each instruction requires five cycles to execute (CPI = 5)*

- *Instruction fetch (IF)*
  - *Get the next instruction.*
- *Instruction decode & register fetch (ID)*
  - *Decode the instruction and get the registers from the register file.*
- *Execution/effective address calculation (EX)*
  - *Perform the operation.*
    - *For load and stores, calculate the memory address (base + immed).*
    - *For branches, compare and calculate the branch destination.*
- *Memory access/branch completion (MEM)*
  - *For load and stores, perform the memory access.*
  - *For* **taken** *branches, update the program counter.*
- *Writeback (WB)*
  - *Write the result to the register file.*
  - *For stores and branches, do nothing.*

Simple DLX operation (without pipelining)

- *Datapath for the unpipelined version:*



- *Red boxes are temporary storage locations.*

Simple DLX operation (without pipelining)

- *The temporary storage locations were added to the datapath of the unpipelined machine to make it easy to pipeline.*

- *Note that branch and store instructions take 4 clock cycles.*
    - *Assuming branch frequency of 12% and a store frequency of 5%, CPI is 4.83.*

- *This implementation is* **not** *optimal. Improvements include:*

- *Completing ALU instructions during the MEM cycle (drops CPI to 4.35 assuming 47% ALU operation frequency).*

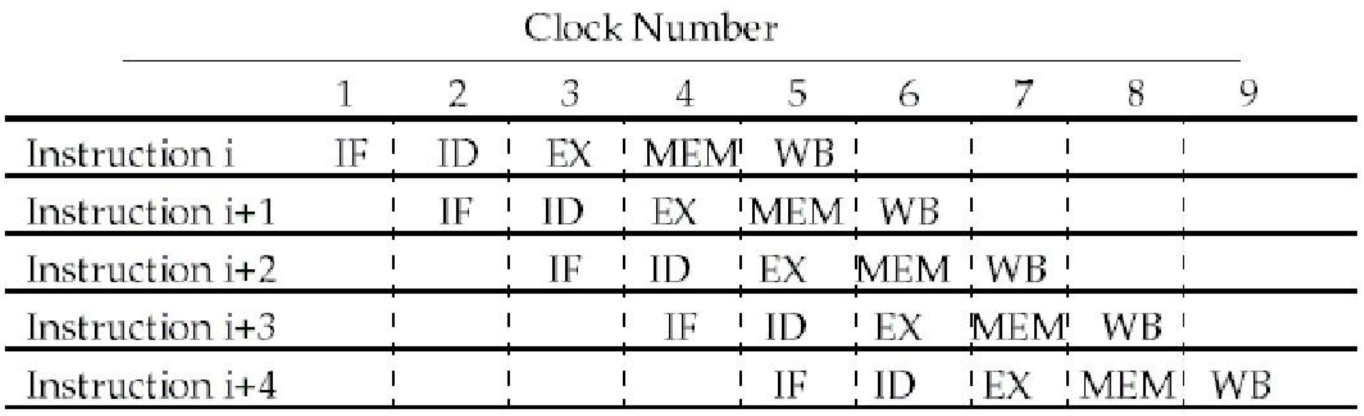
- *Other improvements to CPI are possible but are likely to increase the clock cycle time.*

- *Also, several hardware redundancies exist:*
  - *ALU can be shared.*
  - *Data and instruction memory can be combined since access occurs on different clock cycles.*

Pipelining DLX

- *Since there are five separate stages, we can have a pipeline in which one instruction is in each stage.*

- *This will decrease CPI to 1, since one instruction will be issued (or finish) each cycle.*



	Clock Number								
	1	2	3	4	5	6	7	8	9
Instruction i	IF	ID	EX	MEM	WB				
Instruction i+1		IF	ID	EX	MEM	WB			
Instruction i+2			IF	ID	EX	MEM	WB		
Instruction i+3				IF	ID	EX	MEM	WB	
Instruction i+4					IF	ID	EX	MEM	WB

- *During any cycle, one instruction is present in each stage.*

- *Ideally, performance is increased five fold !*
- *However, this is rarely achievable as we will see.*