## Q.2

### Pipelining

* Key implementation technique for speeding up CPUs
* Breaks each instruction into a series of steps and executes them in parallel
* Clock rate set by the time needed for longest step

### Benefits of Pipelining

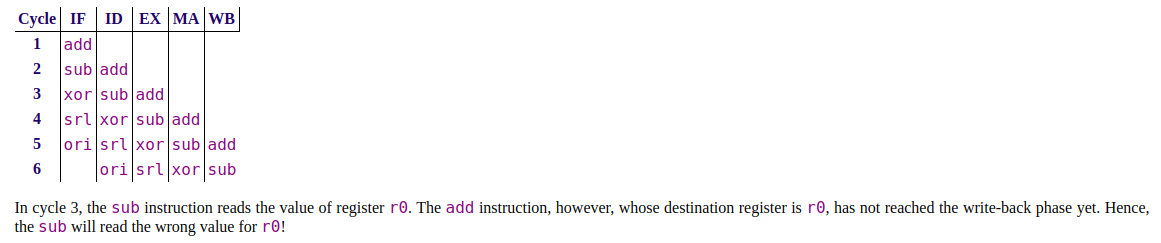
* Throughput increased by the depth of the pipeline
* Clock frequency n-times faster than a non-pipelined processor, where n is the number of stages
* Good performance if no stalls

### DLX Microprocessor

* IF - Instruction Fetch - Sends out the PC and fetches the instruction from memory into the instruction register (IR).
* ID - Instruction Decode & register fetch - Decodes the instruction and accesses the register file to read the registers.
* EX - Execution and effective address calculation - ALU operates on the operand prepared in the prior cycle.
* MA - Memory access - Accesses memory if needed.
* WB - Write back to register - Writes the result into the register file.

### Data Hazards

* Occur when instructions that exhibit data dependence modify data in different stages of the pipeline



### Two Techniques that Prevent Stalls that can overcome Data Hazards

1. Pipeline Forwarding
   * All registers are clocked synchronously
   * The ALU results from “previous” two instructions can be forwarded to ALU inputs from ALUOut0 and ALUOut1 pipeline registers before results are written back into register file
2. Two-Phase Clocking
   * DLX register file can be written then read in a single clock cycle
   * Written during first half of the cycle (WB phase)
   * Read during second half (ID phase)
   * No need for third forwarding reg

### Determine the resulting value of r1 and the number of clock cycles needed to execute the code segment if:

<https://www.scss.tcd.ie/Jeremy.Jones/vivio%205.1/dlx/printable.htm>

^This explains how to do this question and explains interlocks etc

<https://www.scss.tcd.ie/Jeremy.Jones/CSU34021/t4-answer.pdf>

^This is the solution (Q2)

### (i) ALU Forwarding Enabled

Clock cycles = 10

R1 = 0x15

Because the results are forwarded to ALUout0 and ALUout1 before the result is written back into memory, you get the right result as when forwarding is disabled with interlocks and the same amount of clock ticks as when forwarding is disabled without interlocks

### (ii) ALU Forwarding Disabled CPU Data Dependency Interlocks Enabled

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cycle | IF | ID | EX | MA | WB |
| 1 | ADD1 |  |  |  |  |
| 2 | ADD2 | ADD1 |  |  |  |
| 3 | ADD3 | ADD2 | ADD1 <- stall |  |  |
| 4 | ADD3 | ADD2 | NOP | ADD1 |  |
| 5 | ADD3 | ADD2 | NOP | NOP | ADD1 |
| 6 | ADD4 | ADD3 | ADD2 <- stall | NOP | NOP |
| 7 | ADD4 | ADD3 | NOP | ADD2 | NOP |
| 8 | ADD4 | ADD3 | NOP | NOP | ADD2 |
| 9 | ADD5 | ADD4 | ADD3 <- stall | NOP | NOP |
| 10 | ADD5 | ADD4 | NOP | ADD3 | NOP |
| 11 | ADD5 | ADD4 | NOP | NOP | ADD3 |
| 12 |  | ADD5 | ADD4 <- stall | NOP | NOP |
| 13 |  | ADD5 | NOP | ADD4 | NOP |
| 14 |  | ADD5 | NOP | NOP | ADD4 |
| 15 |  |  | ADD5 | NOP | NOP |
| 16 |  |  |  | ADD5 | NOP |
| 17 |  |  |  |  | ADD5 |
| 18 | HALT |  |  |  |  |

Clock cycles = 18

Result in R1 = 0x15

|  |  |  |
| --- | --- | --- |
|  | R1 = 0x01 | R2 = 0x02 |
| ADD1 R1 = R1 + R2 | R1 = 0x03 |  |
| ADD2 R2 = R1 + R2 |  | R2 = 0x05 |
| ADD3 R1 = R1 + R2 | R1 = 0x08 |  |
| ADD4 R2 = R1 + R2 |  | R2 = 0x0D |
| ADD5 R1 = R1 + R2 | R1 = 0x15 |  |

(iii) ALU Forwarding Disabled, CPU Data Dependency Interlocks Disabled

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cycle | IF | ID | EX | MA | WB |
| 1 | ADD1 |  |  |  |  |
| 2 | ADD2 | ADD1 |  |  |  |
| 3 | ADD3 | ADD2 | ADD1 |  |  |
| 4 | ADD4 | ADD3 | ADD2 | ADD1 |  |
| 5 | ADD5 | ADD4 | ADD3 | ADD2 | ADD1 |
| 6 |  | ADD5 | ADD4 | ADD3 | ADD2 |
| 7 |  |  | ADD5 | ADD4 | ADD3 |
| 8 |  |  |  | ADD5 | ADD4 |
| 9 |  |  |  |  | ADD5 |
| 10 | HALT |  |  |  |  |

Clock cycles = 10

R1 = 0x06