**Ex1**

**a)** Show the values of control signals for AND Rd, Rs, Rt:

* ALUOp: 10 (AND operation)
* RegWrite: 1 (Enable writing to register file)
* MemRead: 0 (No memory read operation)
* MemWrite: 0 (No memory write operation)
* RegDst: 1 (Destination register is specified by Rd)
* ALUSrc: 0 (Second ALU operand is from a register)

**b)** Which blocks/components produce useful output for this instruction?

* Register File: Supplies values from Rs and Rt.
* ALU: Performs the AND operation.
* Instruction Memory: Supplies the instruction to be executed.

**Ex2**

For each cut (1), (2), (3):

**(1):**

* Instruction that will fail: R-type add, addi,… (Requires ALU).
* Instruction that works: lw, sw (Only needs memory).

**(2):**

* Instruction that will fail: sw (Cannot access data memory).
* Instruction that works: The others instruction.

**(3):**

* Instruction that will fail: I-type ).
* Instruction that works: Instruction without immediate.

**Ex3**

**a)** Clock cycle time with/without MUL:

* Without MUL: Cycle time = 1130 ps.
* With MUL: Cycle time = 1430 ps.

**b)** CPU speedup/slowdown:

* Without MUL: CPI = 1.
* With MUL: Total instruction count decreases by 5%, but clock cycle increases.
* Speedup = (100% / 95%) \* (1130 ps / 1140 ps) = approximate 0.83 < 1.
* Conclusion: CPU slows down.

**c)** Is adding MUL a good design choice?

* Good choice for multiplication calculate but bad for other instructions.

**Ex4**

**a)** Fraction of cycles using data memory:

* lw: 25%.
* sw: 10%.
* Total: 25% + 10% = **35%.**

**b)** Fraction of cycles where imm-gen output is useful:

* addi: 20%.
* beq: 25%.
* lw: 25%.
* sw: 10%.
* Total: 20% + 25% + 10% + 25% = **80%.**

**Ex5**

**a)** Clock cycle time:

* Non-pipelined: Sum of latencies = **1250 ps (5 stages design).**
* Pipelined: Max stage latency = **350 ps (slowest stage).**

**b)** Total latency of lw instruction:

* Non-pipelined: **1250 ps.**
* Pipelined: 350 ps × 5 stages = **1750 ps.**

**c)** Utilization of instruction memory:

* Non-pipelined: 100% (Used for entire instruction execution).
* Pipelined: 1/5 = **20% (per cycle).**

**d)** Utilization of data memory and write-register port:

* Data memory: Accessed during lw and sw = 35%.
* Write-register port: Accessed for add + addi + lw = 65%.

**Ex6**

**a)** Hazards for the code:

* lw t0, 0(t0) followed by add t1, t0, t0:
  + Hazard: Data Hazard (Read After Write).
  + Solution: Forwarding or stall cycles.

**b)** add t1, t0, t0 followed by addi t2, t0, 5:

* Hazard: Data Hazard (Read After Write).
* Solution: Solution: Forwarding or stall cycles.
* addi t2, t0, 5 followed by addi t4, t1, 4:
  + Hazard: Data Hazard (Read After Write).
  + Solution: Forwarding or stall cycles.

**Ex7**

**a)** Potential hazard:

* Data Hazard (Read After Write).

**b)** Performance comparison:

* CPU1: No forwarding -> (3 +4 + 2 + 2) \* 250 ps = 2750.
* CPU2: Full forwarding -> (3 + 4 + 0) \* 300 ps = 2100.
* CPU3: 2 Partial forwarding -> (3 + 4 + 1) \* 290 ps = 2320.
* Fastest: **CPU2** (Full forwarding eliminates stalls).

**Ex8**

**a)** Clock cycle:

* Max latency of pipeline stages = **200 ps.**

**b)** Pipeline diagram:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| lw x16, 8(x6) | IF | ID | EX | MEM | WB |  |  |  |  |
| sw x16, 12(x6) |  | IF | ID | EX | MEM | WB |  |  |  |
| beq x5, x4, Label |  |  | IF | ID | EX | MEM | WB |  |  |
| add x5, x1, x4 |  |  |  | IF | ID | EX | MEM | WB |  |
| slt x5, x15, x4 |  |  |  |  | IF | ID | EX | MEM | WB |

**Ex9**

**a)**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| lw x1, 0(x1) | IF | ID | EX | MEM | WB |  |  |  |  |
| and x1, x1, x2 |  | IF | ID | EX | MEM | WB |  |  |  |
| lw x1, 0(x1) |  |  | IF | ID | EX | MEM | WB |  |  |
| lw x1, 0(x1) |  |  |  | IF | ID | EX | MEM | WB |  |
| beq x1, x0, loop |  |  |  |  | IF | ID | EX | MEM | WB |

**b)**

**Observe the pipeline when the code runs indefinitely:**

* At steady state, a new instruction enters the pipeline every cycle.
* All 5 stages are occupied most of the time.

**Utilization of Pipeline Stages**:

* Each stage is busy approximately **100% of the time** when the pipeline reaches a steady state.
* Stalls (due to hazards) slightly reduce utilization but are minimized by full forwarding.

**Average CPI (Cycles Per Instruction)**

An ideal pipeline:

* CPI ≈ 1 (each instruction starts in a new cycle).

But due to data hazards, there are stalls.

* Between lw and and, a stall of 1 cycle occurs (even with forwarding).
* Perfect branch prediction eliminates stalls due to branches.

Total cycles = 5 (instrusctions) + 1 (stall) = 6

* The Average CPI = Total cycles / Number of instructions = 6 / 5 = 1.2.