**Question #1**

add r5, r2, r1

nop

nop

lw r3, 4(r5)

lw r2, 0(r2)

nop

or r3, r5, r3

nop

nop

sw r3, 0(r5)

Some instructions can be moved up/down and swapped with other instructions so that hazards can be avoided when an instruction has a dependency or dependencies. This helps eliminate the use of some nop(s).

1: add r5, r2, r1

3: lw r2, 0(r2)

nop

2: lw r3, 4(r5)

nop

nop

4: or r3, r5, r3

nop

nop

5: sw r3, 0(r5)

A hazard detection unit is required b/c it stalls when the instruction lw is used. This is required with forwarding b/c stalling keeps the processor from using the value until lw is fully done. If we didn’t have hazard detection, the instruction that follows the lw and uses the value won’t get the right data. This is extremely bad for computing. Thus, hazard detection is imperative!

…

CYCLE

↓

add r5, r2, r1 🡪 IF ID EX MEM WB [ 1 ] [ PCWrite = 1, ALUin1 = x, ALUin2 = x ]

lw r3, 4(r5) 🡪 IF ID EX MEM [ 2 ] [ PCWrite = 1, ALUin1 = x, ALUin2 = x ]

lw r2 0,(r2) 🡪 IF ID EX [ 3 ] [ PCWrite = 1, ALUin1 = 0, ALUin2 = 0 ]

or r3, r5, r3 🡪 IF ID [ 4] [ PCWrite = 1, ALUin1 = 1, ALUin2 = 0 ]

sw r3, 0(r5) 🡪 IF [ 5] [ PCWrite = 1, ALUin1 = 0, ALUin2 = 0 ]

**Question #2**

1. See Table

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruc.** | **Reg**  **Dest** | **ALU**  **Src.** | **Mem**  **To**  **Reg** | **Reg**  **Write** | **Mem**  **Read** | **Mem**  **Write** | **Branch** | **ALU**  **Op** | **ALU**  **CTRL** | **Func.**  **Code** |
| add | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 10 | 0010 | 100000 |
| sub | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 10 | 0110 | 100010 |
| AND | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 10 | 0000 | 100100 |
| OR | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 10 | 0001 | 100101 |
| beq | X | 0 | X | 0 | X | X | 1 | 01 | 0110 | X |
| j | X | X | X | 0 | X | X | X | X | X | X |
| lw | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 00 | 0010 | X |
| sw | X | 1 | X | 0 | 0 | 1 | 0 | 00 | 0010 | X |

1. See Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction** | **ALU OP** | **ALU Control** | **Function Code** |
| add | 10 | 0010 | 100000 |
| sub | 10 | 0110 | 100010 |
| AND | 10 | 0000 | 100100 |
| OR | 10 | 0001 | 100101 |
| beq | 01 | 0110 | X |
| j | X | X | X |
| lw | 00 | 0010 | X |
| sw | 00 | 0010 | X |

add: IF, ID, EX, WB

sub: IF, ID, EX, WB

AND: IF, ID, EX, WB

OR: IF, ID, EX, WB

beq: IF, ID, EX, WB, MEM

j: IF, ID, EX

lw: IF, ID, EX, WB, MEM

sw: IF, ID, EX, MEM

**Question #3**

The clock cycle time [CCT] in a non-pipelined processor [NPP] is the sum of the latencies for the individual stages of the data-path. So, that is:

250 + 350 + 150 + 300 + 200 = 1250ps

IF + ID + EX + MEM + WB = CCT For NPP

The clock cycle time [CCT] in a pipelined processor [PP] is the slowest stage. In this case, the slowest stage is ID, which has a latency of 350ps. Therefore:

CCT For PP = 350ps

Total latency for of an LW instruction in a pipelined processor is: 1250ps

(Add up all the latencies)

Total latency for an LW instruction in a non-pipelined processor is:

Total Latency = (Number Of Stages) (Clock Cycle Time)

= (5) (350)

= 1750ps

For Single Cycle:

Since each instruction takes one cycle to execute

Clock Cycle Time [CCT] = 1250ps

Execution Time [ET] = 1250ps / 350 ps = 3.57142857 ~=~ 3.57

For Multi-Cycle:

ALU completes in 4 cycles

IF + ID + EX + WB = 250 + 350 + 150 + 200 = 950ps

lw completes in 5 cycles

IF + ID + EX + MEM + WB = 250 + 350 + 150 + 300 + 200 = 1250ps

sw completes in 4 cycles

IF + ID + EX + MEM = 250 + 350 + 150 + 300 = 1050 ps

beq completes in 3 cycles

IF + ID + EX = 250 + 350 + 150 = 750ps

Clock Cycle Time [CCT] =

950 \* 0.45

+

1250 \* 0.20

+

1050 \* 0.15

+

750 \* 0.20

= 985ps

Execution Time [ET] =

4 \* 0.45

+

3 \* 0.20

+

5 \* 0.20

+

4 \* 0.15

= 4.0

We know the CCT for pipeline is 350ps. Since it’s the lowest, therefore it is the fastest.