Advanced Lab in Computer Architecture

1. **Harvard Architecture**

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages:** |
| Using two memories (data, instructions) allows using them in more effective way (reduces hazards, variable word size and high bandwidth) | Separating memories can create a wasteful situation where the instruction memory is full, and the data memory isn't but no more instructions can be loaded even when there is free memory. |
| Setting priority in memory access (fetch Vs load/store) is no longer needed | Control unit is more complex to implement |
| Using pipeline will increase CPI and reduce execution time | Additional registers and logic will require more area and will increase chip cost |
|  | A more complex chip, with more inputs and outputs will require a more sophisticated board, which will increase board cost |

1. **Hazards**
   1. **Structural Hazards**

A structural hazard occurs when 2 (or more) pipelined instructions try to access a resource which can provide only 1 of them. In Harvard architecture this is happening when an **LD** operation is following and **ST** operation (both try to access memory in the same cycle). Another option is when **DMA** is activated (again, memory access on same cycle). These structural hazards are resolved in the following ways:

1. LD-ST Hazard: stalling the pipeline until ST finished EXEC1 state.
2. DMA-CPU Hazard: prioritizing CPU over DMA to avoid unnecessary stalls.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cycle | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Inst 0 – ST | FETCH0 | FETCH1 | DECODE0 | DECODE1 | EXEC0 | EXEC1 |  |  |
| Inst 1 - LD |  | FETCH0 | FETCH1 | DECODE0 | DECODE1 | (STALL) | EXEC0 | EXEC1 |

* 1. **Data Hazards**

A data hazard occurs when there are dependencies between pipelined instruction. when instruction A needs to update register Ri and instruction B needs Ri for execution.

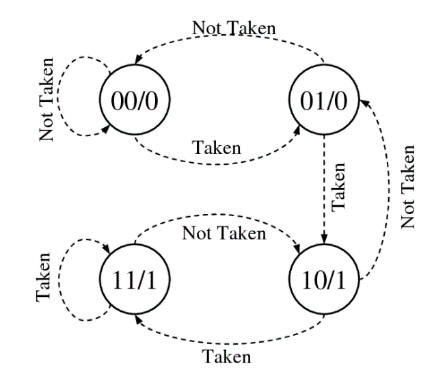
To detect data hazards, source registers of instruction (in decode stage) will be compared to destination registers of previous instructions which are still in the pipeline.  
these hazards are resolved stalling the pipeline unstill the sources of the instruction are ready, meaning are not destination registers of previous instructions.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cycle | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| ADD r2 r3 r4 | FETCH0 | FETCH1 | DECODE0 | DECODE1 | EXEC0 | EXEC1 |  |  |  |
| SUB r5 r2 r3 |  | FETCH0 | FETCH1 | DECODE0 | (STALL) | (STALL) | DECODE1 | EXEC0 | EXEC1 |

* 1. **Control Hazard**

A control hazard occurs when a branch was wrongly taken, meaning wrong instructions entered the pipeline.

To decrease the frequency of those hazards we use branch prediction. If , even with prediction, a wrong branch has been taken all wrong instructions have to be flushed out of the pipeline. After flushing, new (and correct) instructions will be fetched.

1. In our design the branch predictor is 2-bit branch predictor.  
   It can be described using the following state machine:

We will use a single predictor for the whole CPU which will be used in every branch and updated in every branch resolution.

1. ***Implemented in code***
2. **Files are provided**

|  |  |  |  |
| --- | --- | --- | --- |
| Application | Previous Execution Time | Current Execution Time | Speed Up |
| Example | 366 cycles |  |  |
| Multiplication | 1248 cycles |  |  |
| Fibonacci | 1644 cycles |  |  |

1. ***DMA is Implemented in code***
2. DMA annotated code:  
   ***Other Files are attached***