Assignment 2- Report

Team- Subham Agarwala(IMT2022110)

Chandrima Nandi(IMT2022062)

Overview:

We were asked to implement a MIPS Processor (pipelined and non-pipelined) using any programming language. We used Python to implement the various features of a MIPS processor, like control signals, MUX to decide inputs and outputs conditionally, ALU, Hazard Detection and forwarding unit, and pipeline flush. The program for both pipelined and non-pipelined processors takes machine code, here a sorting algorithm, and executes the code in 5 phases: Instruction Fetch, Instruction Decode, Execute, Memory and Write Back.

Question 1: Non-Pipelined Processor

The machine_code dictionary has the instruction memory address as the key and the instruction as the value. When the program calls the fetch function it fetches the instructions from the dictionary one by one and returns it to main. From there the instruction is passed on to decode phase where the decode function identifies the type of instruction and decodes the binary instruction into fields like rs, rt, rd, funct, etc. The control unit is also called in the decode phase which generates control signals according to which the flow of the program is decided. Depending on the control signals, the execute phase performs the various operations and forwards them to mem and write back phase. The ALU control unit generates ALU control signal which decides which operation is performed in the ALU. In the mem phase data is written to memory or extracted from memory which is later written back to the register file.

The program also prints the number of clock cycles taken to execute all instructions.

Assumptions and Limitations:

The number of inputs and outputs is restricted to 10.

We are initializing the input to specific registers as done in Assignment-1

Result:

```
PS C:\Users\subha\OneDrive - iiit-b\College docs\semester 3\Computer Architecture> python -u "c:\Users\subha\OneDrive - iiit-b\College docs\semester 3\Computer Architecture\Assignments\Assignment 2\non-pipelined.py"
Enter the number of inputs: 3
Enter input: 3
Enter input: 2
Enter input: 1
The sorted array is:
1
2
3
Number of clock cycles:  505
PS C:\Users\subha\OneDrive - iiit-b\College docs\semester 3\Computer Architecture> python -u "c:\Users\subha\OneDrive - iiit-b\College docs\semester 3\Computer Architecture\Assignments\Assignment 2\non-pipelined.py"
Enter the number of inputs: 5
Enter input: -20
Enter input: 45
Enter input: 32
Enter input: 0
Enter input: 1
The sorted array is:
-20
0
1
32
45
Number of clock cycles:  1055
PS C:\Users\subha\OneDrive - iiit-b\College docs\semester 3\Computer Architecture> python -u "c:\Users\subha\OneDrive - iiit-b\College docs\semester 3\Computer Architecture\Assignments\Assignment 2\non-pipelined.py"
Enter the number of inputs: 10
Enter input: 43
Enter input: 23
Enter input: -67
Enter input: -100
Enter input: 65
Enter input: 0
Enter input: 2
Enter input: 85
Enter input: -2
Enter input: 41
The sorted array is:
-100
-67
-2
0
2
23
41
43
65
85
Number of clock cycles:  3470
```

Question 2: Pipelined Processor

This is a modification of the previous code with pipelining implemented. Pipeline registers (IF_ID, ID_EX, EX_MEM, MEM_WB) are added which store the data retrieved at each stage along with the data from the previous phases. The data gets forwarded from the previous pipeline register to the next one at each stage. We made separate lists with names as the 5 phases. The IF list stores all machine codes initially. The rest of the lists will store a copy of the pipelined registers. Whenever a jump or branch (taken) is detected, the pipeline is flushed, i.e. the IF list is emptied and instructions from the target to the end of the machine code are copied to IF. A function is implemented to detect hazards and forwarding is done in the same.

Output:

```
PS C:\Users\subha\OneDrive - iiit-b\College docs\semester 3\Computer Architecture> python -u "c:\Users\subha\OneDrive - iiit-b\College docs\semester 3\Compute
r Architecture\Assignments\Assignment 2\pipelined_final.py"
Enter the number of inputs: 5
Enter input: 4
Enter input: -5
Enter input: 0
Enter input: 7
Enter input: 2
```

```
{268501184: 4, 268501188: -5, 268501192: 0, 268501196: 2, 268501200: 7, 268501204: 0, 268501208: 0, 268501212: 0, 268501216: 0, 268501220: 0, 268501224: -5, 2
68501228: 0, 268501232: 2, 268501236: 4, 268501240: 7, 268501244: 0, 268501248: 0, 268501252: 0, 268501256: 0, 268501260: 0, 268501264: 0}
The sorted array is:
-5
0
2
4
7
PS C:\Users\subha\OneDrive - iiit-b\College docs\semester 3\Computer Architecture>
```

We have printed the memory along with the result to show the address in the memory where the input and output are stored.