CS23BT071 assignment4

PRIYANSHU RAO

March 2025

Program	Cycles Taken	OF Stall	Branch Taken (Instructions)
Descending Order	600	143	176
Even or Odd	19	9	0
Fibonacci	148	34	32
Palindrome	138	71	14
Prime	66	23	10

Table 1: Pipeline Performance Metrics for Different Programs (Test cases)

Program	Cycles Taken	OF Stalls	Branch Taken (Instructions)
Descending Order	898	410	136
Even or Odd	18	6	2
Fibonacci	146	37	20
Palindrome	32	10	4
Prime	104	27	22

Table 2: Pipeline Performance Metrics for My codes

1 Descending

1.1 More Cycles

The cycles are intutively high as descending.out generated from descending.asm contatins many iteration over loops for arranging the elements in descending. More cycles show as that why sorting takes longer time.

1.2 More OF stalls

As we are iterating over the loops and also we are having many data dependencies in each loop due to frequently accessing the same register again and again. Results in more OF stalls.

1.3 More Branches

As looping results in more branches, thus results and more branch taken results in more branch hazards.

2 Even or Odd

2.1 Less cycles

As this is just a simple code from evenorodd.out from evenorodd.asm having just modulo operator for checking if the number is even or odd. Results in small number of cycles.

2.2 Less OF stalls

As the code is too small and the same register accessing is also not frequent which might have lead to high OF stalls is also not there. Hence, it just have very less number of OF stalls.

2.3 Less branches

As we just have to check if a number is even or odd, thus we can have only one branch which is either taken or not. Hence, Branch taken instructions can be also either 0 or 2.

3 Fibonacci

3.1 More cycles

As the program is haaving frequent access to registers for computing fibonacci series from fibonacci.out from fibonacci.asm i.e. having loop (iteration) for going to the next term. This loop itself has to check for many conditions resulting in more time to complete the execution, resulting in more cycles.

3.2 Moderate stalls

As for going to the next term frequent access of the previous terms is required resulting in Data Hazards for which we have to have more stalls (but moderate not too high) for resolving those hazards.

3.3 Moderate branches

As this program has to loop over the same instructions for computing fibonacci terms, each loop has to check for a condition of continuing or not leads to taking branch for checking leads to more branch hazards resolving (but moderate not too high).

4 Palindrome

4.1 More cycles

Calculating palindrome takes too much time and on toop of that frequent accessing adds to the stalls, resulting in more cycles.

4.2 Moderate OF stalls

As checking palindrome has to access the number frequently and updating it, this leads to data dependency resulting in more stalls (but moderate not too high).

4.3 Less branches

As this palindrome check does not have many branches, the probability of taking the branch also reduces, and hence this has less branch hazards to be resolved.

5 Prime

5.1 Less cycles

Less complicated program results in less cycles as it is fairly simple program to be executed, and hence leads to less cycles. Since primality testing mostly involves a division loop with no deep dependencies, stalls are lower than sorting but higher than Even/Odd.

5.2 Less OF stalls

As frequent data access (data dependency) is not there. Hence, it results in fewer OF stalls. Testing mostly involves a division loop with no deep dependencies, stalls are lower.

5.3 Less branches

Very less branches are there and hence, reduces the chances of branch taken reduces branch hazards to be resolved.