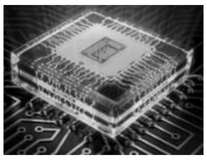
National and Kapodistrian University of Athens
Department of Informatics and Telecommunications
COURSE: **COMPUTER ARCHITECTURE II**



Handout 1 – Pipelining

FALL SEMESTER 2021 - 2022

Objective

The goal is to design a MIPS microprocessor with a pipeline that executes a specific program in the shortest possible time per unit of cost (cost effectiveness). The execution of the program and the evaluation of your designs will be done in the QtMips simulator that we use in the lab.

The program must process an array of 198 positive integers [k1, ..., k198] that has random and unordered content and each number is less than 100. For each triplet of consecutive numbers, the program must compute the greatest common divisor (GCD) and their least common multiple (LCM). The results must be written in two separate integer arrays of size 66 positions each (one element for each triplet). The initial array will already have contents in the .data section of the program and will not be given numbers by the user or printed results to the console. The correct values of the GCD and lcm matrices should simply be in memory.

You can consider different MIPS microprocessor pipeline configurations based on cost and choose the one that gives your program the best performance per unit cost. In this work we assume that the memory is ideal and thus in all configurations caches should be disabled and main memory should have an access time of 1 clock cycle. Your microprocessor design options are summarized in the following table. The overhead and clock overhead are additive to the basic cost K and the basic clock cycle C.

Parameter	Additional Cost	Clock Penalty
No Data Hazard Unit	0	0
Stall when Data Hazard is detected	+ 2 %	0
Stall or Forward when Data Hazard is detected	+ 5 %	+ 3 %
Delay Slot	0	0
Stall	+ 2 %	0
Branch Predictor 1-bit with 6-bit BHT	+ 7 %	+ 1 %
Branch Predictor 2-bit with 6-bit BHT	+ 8 %	+ 2%
Resolution at ID instead of EX	+ 1 %	0

In addition to your programs, which you must submit separately, complete the following tables with your choices and results. If you take measurements with more than one table contents give all the data.

Parameters	Yes/No
No Data Hazard Unit	No
Stall when Data Hazard is detected	No
Stall or Forward when Data Hazard is detected	Yes
Delay Slot	Yes
Stall	No
Branch Predictor 1-bit with 6-bit BHT	No
Branch Predictor 2-bit with 6-bit BHT	No
Resolution at ID instead of EX	Yes

Cost	Clock Cycle	Cycles	Instructions	CPI	Execution Time (Cycles * Clock Cycle)	Cost Performance (1/[Time*Cost])
1.06 * K	1.03 * C	5594	5122	1.09	5761.82 * C	1 / [6107.53 * C K]

Implementation

To calculate the GCD I used Euclid's algorithm initially for the first two integers and then the third, since:

$$GCD(a, b, c) = GCD(c, GCD(a, b))$$

Then I used the GCD to calculate the LCM so that fewer operations are performed:

$$LCM(a, b) = a * b / GCD(a, b)$$

$$LCM(a, b, c) = LCM(c, LCM(a, b)) = c * LCM(a, b) / GCD(c, LCM(a, b))$$

In this way, only two multiplications are needed for each triplet of numbers. Thus, the cost of operations is greatly reduced. I furthermore added an extra check, if the number in the divisor becomes 1, then to stop there since this will be the GCD. I tested this version of the program in QtMips without pipeline to make sure that it was working correctly. Then, I tried to use as few register-to-register transfers as possible by reusing registers when they are not needed. Also to eliminate data dependencies I reordered instructions where it was possible.

Finally, I added nops where it was necessary so that the program works correctly with a delay slot or a branch predictor. Then, after the branch commands I replaced the nops with add and div commands, which can enter the pipeline replacing the stalls that would have existed. Consequently, only the nops after the jump commands were required.

Parameter Selection

The way I've implemented this algorithm made it obvious, that it would run optimally on a processor with a delay slot or BHT as a control hazard unit. Also, a data hazard unit was necessary, otherwise I would have to write the assembly code in a different manner and add more nops, but that would in turn increase the cycles.

Furthermore, resolving the branches in the ID stage is necessary to correctly execute the code when using a delay slot. If the branch resolution was done in the EX stage, I would have to double the nops, which would increase the commands. Also the small additional cost (+1%) of using the ID resolution has and the zero clock overhead makes it a much better choice.

Finally, using stall when detecting a data hazard, although it has a greater cost than the delay slot, does not offer better performance since a lot of stalls are added. Nevertheless, I did a few tests with this parameter as well to verify it.

Measurements

Initially, I made measurements using a delay slot as a control hazard unit, resolving the branches in the ID stage and testing with both available options for the data hazard unit.

	Cycles	Instructions	CPI
Delay Slot | Resolution ID | Stall when Data Hazard is detected	6207	5122	1.21183
Delay Slot | Resolution ID | Stall or Forward when Data Hazard is detected	5594	5122	1.09215

Predictably, by using forwarding when data hazard is detected, the CPI improves by a lot.

Afterwards, to rule out the stall option, I tested it against the best parameters for the delay slot, resolution at the ID stage and data hazard detection and forwarding. That way since the cost of the stall is greater than that of the delay slot, if it is not faster, then it isn't efficient enough and I can discard it immediately from the available options.

	Cycles	Instructions	CPI
Stall | Resolution ID | Stall or Forward when Data Hazard is detected	6527	4462	1.4628
Delay Slot | Resolution ID | Stall or Forward when Data Hazard is detected	5594	5122	1.09215

As is evident, the stall is not only more expensive, but also slower than the delay slot. (On the other hand, the delay slot works correctly, because I've taken the necessary measures in the code, while stall could work without these changes as well.)

Finally, I examined the case of using a branch predictor of one or two bits, solving the branches at the ID stage and with the two available settings of data hazard units with stall and forwarding.

	Cycles	Instructions	CPI	Accuracy
Branch Predictor 1-bit with 6-bit BHT | ID | FORWARD	5584	4787	1.16649	79.5982%
Branch Predictor 1-bit with 6-bit BHT | ID | STALL	6589	4787	1.37644	79.5982%
Branch Predictor 2-bit with 6-bit BHT | ID | FORWARD	5580	4785	1.16614	79.7238%
Branch Predictor 2-bit with 6-bit BHT | ID | STALL	6585	4785	1.37618	79.7238%

From the above table, it is evident that the 1-bit branch prediction and the 2-bit branch prediction have almost the same results and similar prediction accuracy, while executing the program. So since the 2-bit branch prediction has more cost and also a greater penalty on the clock it is rejected. Also, using the data hazard unit that stalls, instead of forwarding increases the cycles by 1000 (it raises the CPI to 1.37644 from 1.16649) so this could also be dropped, but I'll keep it under consideration due to the additional cost of forwarding and clock time overhead.

Evaluation

Based on these test results, I examined the two best options for the microprocessor design which are: 1-bit BHT's branch predictor or using a delay slot with the aforementioned parameters. So from these two I have the following measurements:

Cost	Clock Cycle	Cycles	Instructions	CPI	Execution Time (Cycles * Clock Cycle)	Cost Performance (1/[Time*Cost])
1.06 * K	1.03 * C	5594	5122	1.09	5761.82 * C	1 / [6107.53 * C * K]
1.03 * K	1 * C	6207	5122	1.21	6207 * C	1 / [6393.21 * C * K]
1.13 * K	1.04 * C	5584	4787	1.17	5807.36 * C	1 / [6562.32 * C * K]
1.10 * K	1.01 * C	6589	4787	1.38	6654 * C	1 / [7320.38 * C * K]

Evaluating the cost effectiveness of these settings, it follows that the maximum performance to cost will be by using a delay slot as control hazard unit, with the resolution of branches at the ID stage and a data hazard unit with forwarding.

So for my program the pipelined MIPS microprocessor design will be:

Control hazard unit: **Delay Slot**

Resolution: **ID**

Data hazard unit: **Stall or forward when data hazard is detected**