Digital System Design Final Project Pipelined MIPS Design

Yu-Sheng Ting, Ting Wang

***Abstract*—The multi-state pipelined MIPS introduced in the course improves the efficiency of the original single-state MIPS, and in our baseline design, we applies the Branch Forwarding Stall Unit, Read-only ICache, and the Buffer Technique to enhance the performance of the pipelined MIPS. As to the part of extension, we utilized the prediction history table in our Branch Prediction design, the 256-words Level-2 Cache in our L2 Cache design, and the iterative approach in our Multiplication and Division design.**

I. Introduction

THE design of the single-cycle MIPS processor is shown in Fig.1. With this structure, every single operation takes the same amount of time–a single cycle. That is to say, all instructions take the same time as the slowest one, which will lower the overall efficiency. To solve the problem, the technique is utilized. Pipelining enables a processor to overlap the execution of several instructions, and the processor will be led to a better performance.

The five stages in the pipelined datapath are IF, ID, EX, MEM, and WB. Instruction fetching, instruction decoding, executing, memory accessing, and writing back are done in those stages respectively as shown in Fig.2.

However, the pipelined structure will cause data hazards, which occurred under the circumstance in which a planned instruction cannot execute in the proper clock cycle because data needed to execute the instruction is not yet available. The forwarding unit is added to solve the problem.

The instructions in the stages, which is former than the stage at which the branch decision is made, need to be flushed if the branch prediction is wrong. In order to reduce the delay caused by instruction flushing, branch address calculation can be done at ID stage. The branch forwarding unit needs to be added to avoid data hazards as shown in Fig.3.

II. Methods

A. Baseline

The pipelined MIPS processor is attached with an instruction cache (ICACHE) and a data cache (DCACHE). The caches are implemented in direct-mapped architecture and write-back mechanism. The whole module hierarchy is shown in Fig.4.

Three tips are used to lower the cycle time and the area of the whole design.

Y. Ting, T. Wang were with the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan. E-mail: b05901026@ntu.edu.tw, b05901016@ntu.edu.tw

Manuscript received June 25, 2019; revised June 25, 2019.

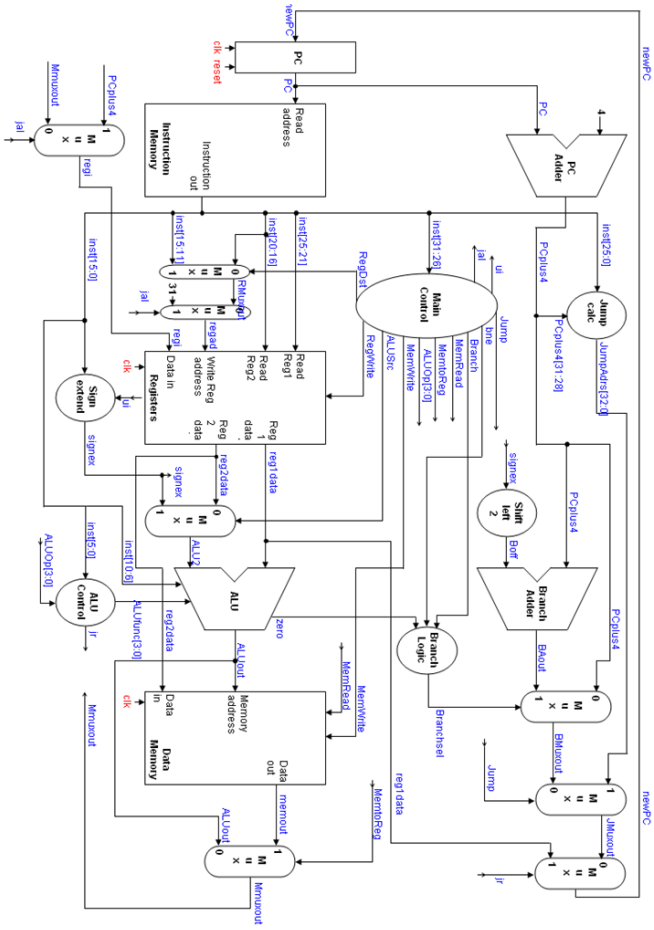

Fig. 1. Control and datapath for the processor

1) Branch Forwarding Stall: As shown in Fig.5, the critical path through ALU and the branch execution unit is too long, which limits the compression of the cycle time. To solve the problem, the forwarding path is avoided by inserting a bubble (NOP operation) at EX stage. By doing so, the data which is needed to be forwarded will be pushed to the MEM stage and pass forward through the path which is much shorter. Although one more cycle is needed because of the inserted bubble, the length of the critical path can be lowered significantly.

2) Read-only Cache: Since the data in the instruction memory will never need to be written, ICACHE can be modified to read-only version. In a read-only cache, mem_wdata and mem_write can be set to 0 directly, and the registers used to record whether the data has been written can be removed. The number of the states can also be reduced. Since the

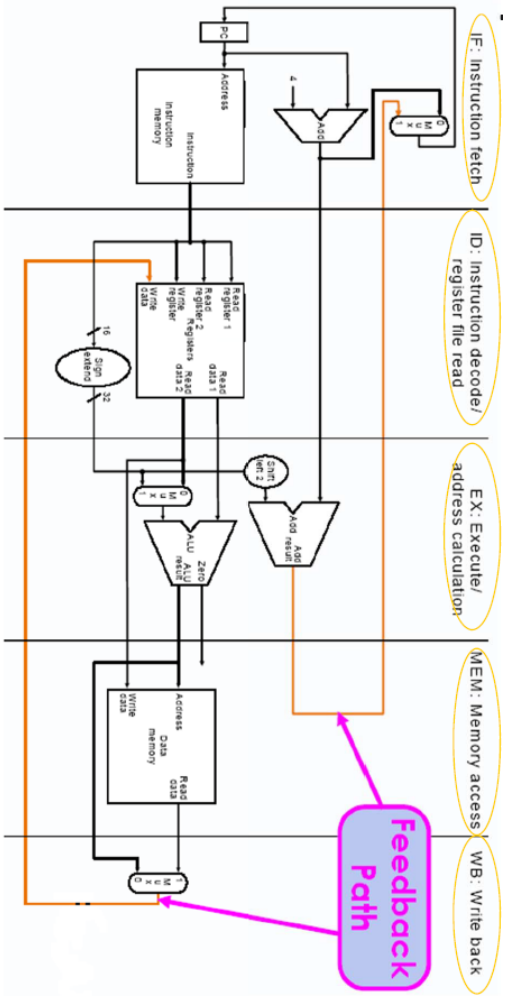

Fig. 2. Pipelined datapath

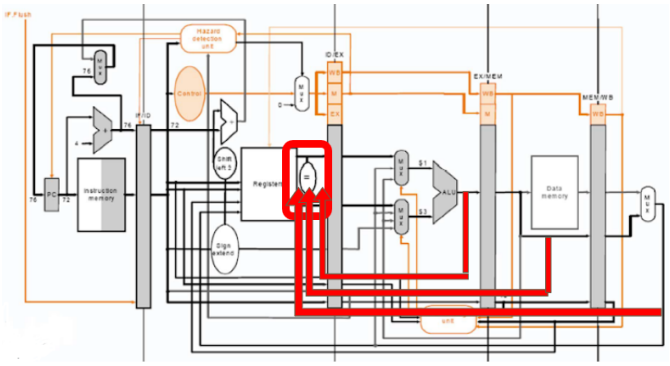

Fig. 3. Branch forwarding

multiplexers are removed, the area of the design and the length of the critical path can both be lowered.

3) Buffer technique: Data from memory change at negative edges of the clock signal, while the cache design is positive edge-triggered. If mem_ready is used as the signal to change the next_data, the computation has to be completed in the half of the cycle, as shown in the red interval shown in Fig.6. Therefore, a ”buffer” is used as a signal instead. The buffer changes its value at positive edges of the clock signal, which

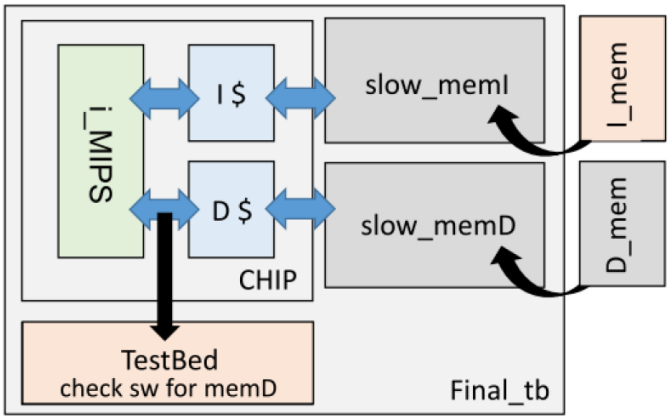

Fig. 4. Module hierarchy

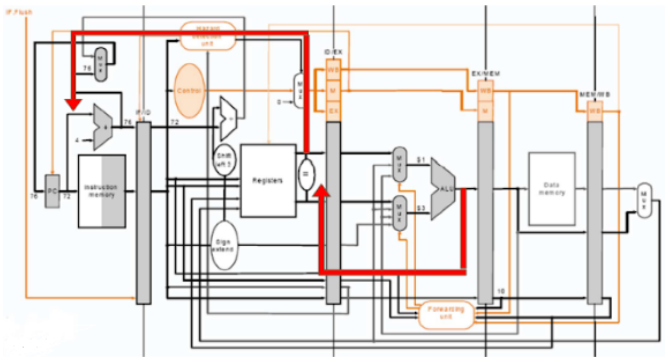

Fig. 5. Critical path before using the stall unit

allows the computation to be completed in a cycle. That is to say, the cycle time could be compressed because of the looser constraint.

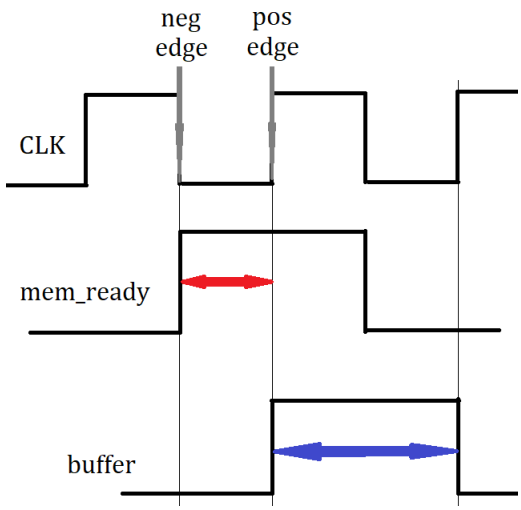

Fig. 6. Using buffer as a flag

B. Branch Prediction

In the baseline pipelined MIPS, the branch-related instructions, in other words, the beq and bne instructions, will take one more cycle to flush the original following instruction

and reassign the other instruction if the Branch conditions are met. The wasted cycle time may do harm to the efficiency of the MIPS, especially when the consecutive branch instructions(such as the for-loop condition) appeared. Thus, the branch prediction mechanism in the early IF stage is utilized, in order to effectively decrease the operation time when the branch-related instructions occurred. In our design, we apply the 4-state Prediction Unit and the Prediction History Table separately, to accomplish the task of Branch Prediction.

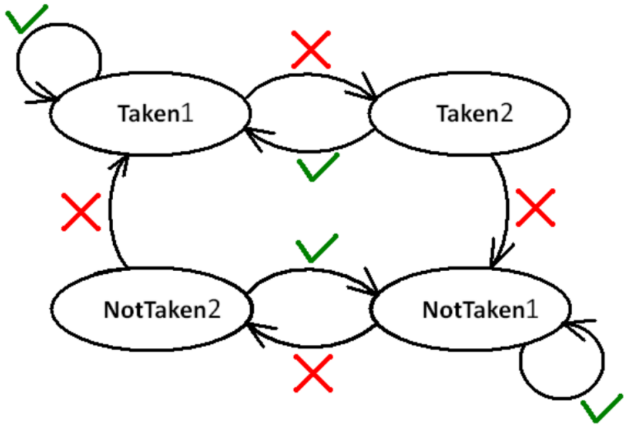

Fig. 7. Finite state graph of 4-state Prediction Unit

1) 4-state Prediction Unit: In the Branch Prediction design, we can apply the 4-state Prediction Unit. The state graph of the Prediction Unit is shown in Fig.7. In our design, the initial state will be NotTaken1, which means the prediction unit's prediction is NotBranch. Next, if the prediction is wrong, the state will go to NotTaken2, while the state will stay in NotTaken1 if the prediction is right. The next-state patterns of the other three states follow the same forms, according the the state graph. In state NotTaken1 and NotTaken2, the prediction is NotBranch, while in state Taken1 and Taken2, the prediction will be Branch. The mechanism will solve the for-loop problem mentioned above, since if consecutive Branch instructions occurred, the state graph will adjust the state of IF-stage prediction unit. The improved design can make the Branch instruction's operation time shorter, compared with the previous design, whose state in the IF-stage is actually AlwaysNotBranch.

2) Prediction History Table: In the previous 4-state Prediction Unit, the simple consecutive Branch conditions can be predicted accurately by the 4-state graph. However, if the branch conditions becomes more complicate, the simple prediction unit will not be able to predict effectively. For instance, if the Branch conditions are interleaved, such as the pattern "NotBranch/Branch/NotBranch/Branch...", the state in the previous 4-state graph will jump between NotTaken and Taken frequently, which means the prediction is not accurate. As a result, we apply the Prediction History Table in our design, which is shown in Fig.8. The Prediction History Table uses n bits to record the last n branch results, and every single composition of the results will lead to its own prediction. Our design utilizes a 2-bit Prediction History Table, which can record the last 2 branch results, and in the table, 0 means NotBranch, and 1 means Branch. That is to say, 00, 01, 10, and 11 in the history table will lead to their own predictions, and since the table consider a more complicated situation of previous branch history, the design with the Prediction History Table is predicted to perform better than the one with a simple 4-state prediction unit.

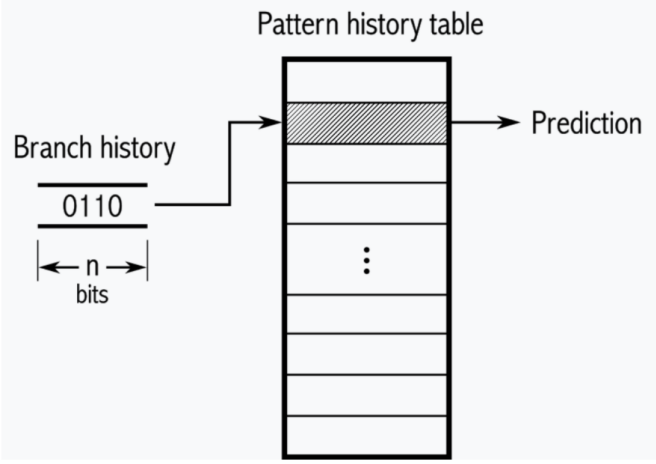

Fig. 8. Method of prediction history table

C. Two-level Cache

Before introducing the two-level cache, let's take a look at the design of the MIPS used in the baseline first. The MIPS mainly consists of a processor(CPU), a cache, and a memory. The processor asks for wanted data or writes data to memory via the cache. As the wanted data is not stored in the cache, the cache will stall the processor and access the memory for the wanted data. Basically, the operation time from a processor to a cache will be much shorter than the one from a cache to a memory. In other words, if the hit rate from the processor to the cache can be higher, the operation time will decrease. In order to raise the hit rate, we can add the second cache, names as the level-two cache or the L2 cache, between the original cache and the memory. The original cache will be named as level-one cache or the L1 cache. When the required data is not found in L1 cache, the L1 cache can ask for the data from L2 cache, which is designed to have bigger size than L1 cache. Only when the L2 cache doesn't have the required data will the L2 cache access the memory to take the wanted data and pass it back to the L1 cache and the processor. Since the access time from the L1 cache to the L2 cache is still much shorter than the one from the L2 cache to the memory, the mechanism, which is called two-level cache, will decrease the frequency of accessing the memory and improve the total operation time. On the other hand, the size of the L2 cache does affect the efficiency of the total design. To find the most suitable size of the L2 cache, we test the efficiency of 16-block, 32-block, 64-block, and 128-block L2 cache and find the relation between the operation time and the L2 cache size.

D. Multiplication and Division

Iterative approach is adopted to implement multiplication and division. In this approach, multiple cycles are used for ALU computation of *mult* and *div*. During the iterations, the successive instructions should be stalled to avoid hazards. The algorithms and the circuit diagrams are shown in Fig.9 and Fig.10. The two blocks are combined in the design as shown in Fig.11.

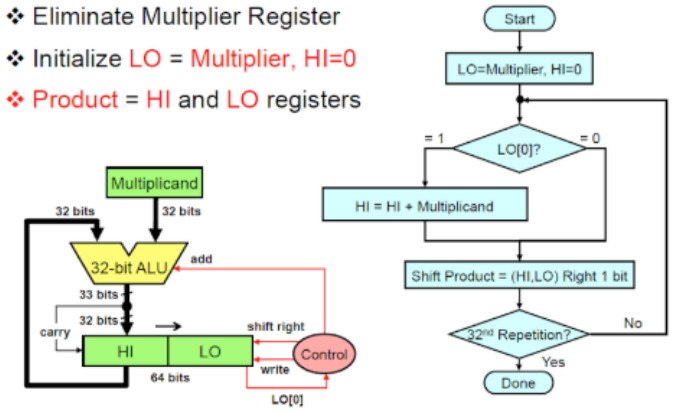

Fig. 9. Multiplication

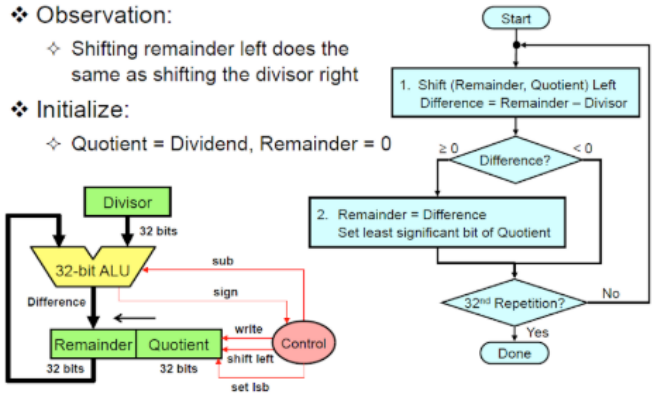

Fig. 10. Division

III. Results

A. Baseline

The performance is evaluated by the a-t value, which is the product of the area of the design and the total simulation time. Lower a-t value means better performance.

1) Branch Forwarding Stall: The improvement on the compression of the cycle time after adding the branch forwarding stall unit is significant. With the synthesis whose cycle time is 2.3 ns, the lowest cycle time passing the simulation is reduced from 6.5 ns to 4.31 ns, which lowers the a-t value a lot.

2) Read-only Cache: With the modification of ICACHE, both the cycle time and the area have no marked improvement. The areas of the design before and after the modification are 276,837 um^2 and 279,585 um^2, respectively, which are almost the same. Both designs can pass the simulation with the cycle time of 2.3 ns. Therefore, the a-t vlue doesn't improve.

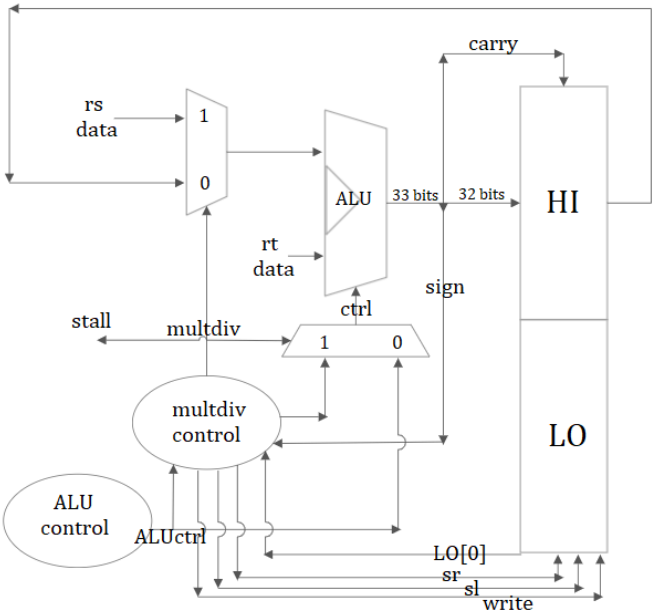

Fig. 11. The design combining multiplication and division

3) Buffer technique: The cycle time and the area of the design are lowered after modifying the ICACHE. With the synthesis whose cycle time is 2.3 ns, the lowest cycle time passing the simulation is reduced from 3.04 ns to 2.3 ns, and the area is reduced from 283,533 um^2 to 279,585 um^2. Thus, the a-t value is much lower than the original design.

B. Branch Prediction

To test if the prediction unit improves the operation time under consecutive branch instructions, we use the testbed BrPred to test. We can adjust the number of NotBranch, InterBranch, and Branch as we want and observe the difference of the operation time between different testbed. The result is shown in Fig.12 and Fig.13. The CHIP_ini and the CHIP_NEW represents the chip without the prediction unit and the one with the prediction unit separately. In the form, only the number of NotBranch and Branch are adjusted, and the number of InterBranch is fixed to 1, since practically, the InterBranch test data is just two consecutive Branch test data. Next, in Fig.14, the CHIP_LATEST, which represents the MIPS with prediction history table, is taken into account. In every test data of the form, the performances of CHIP_NEW and CHIP_LATEST are the same.

C. Two-level Cache

In the two-level cache design, the operation time between different size of cache is compared in Fig.15, Fig.16, and Fig.17..

D. Multiplication and Division

To study the relation between the number of operations of multiplication and division and the total computation time, different testbed are generated to test. The testbed will use

(notBr, Br)	CHIP_ini	CHIP_NEW
(90,10)	5615ns	5545ns
(80,20)	5615ns	5345ns
(70,30)	5615ns	5145ns
(60,40)	5615ns	4945ns
(50,50)	5615ns	4745ns
(40,60)	5615ns	4545ns
(30,70)	5615ns	4345ns
(20,80)	5615ns	4145ns
(10,90)	5615ns	3945ns

Fig. 12. Comparison of the operation time between the original MIPS and the MIPS with the 4-state prediction unit

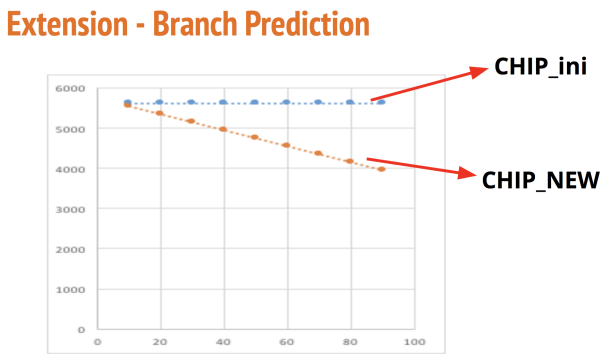

Fig. 13. Trend line of the operation time and the MIPS with and without branch prediction unit

the design to compute the factorial of a positive number, and the term to be computed (nb) can be adjusted. To analyze the change in the total simulation time as nb changes from 1 to 12, the simulation time for each testbed are recorded and presented as a line graph as shown in Fig.16. It can be noticed that the change in simulation time is linear.

IV. Discussion

A. Baseline

The tips of using the branch forwarding stall unit and using the signal buffer can effectively improve the performance of the design. However, the area of the design using read-only cache isn't compressed successfully but even larger. In fact, the structure of ICACHE doesn't change a lot after being modified to read-only version. Simply removing some multiplexers may not have a significant impact on the area because the registers, which are used to store the data, account mostly for the area of the cache. In addition, the synthesizer may use different

(notBr, Br)	CHIP_ini	CHIP_NEW	CHIP_HISTORY
(90,10)	5615ns	5545ns	5545ns
(80,20)	5615ns	5345ns	5345ns
(70,30)	5615ns	5145ns	5145ns
(60,40)	5615ns	4945ns	4945ns
(50,50)	5615ns	4745ns	4745ns
(40,60)	5615ns	4545ns	4545ns
(30,70)	5615ns	4345ns	4345ns
(20,80)	5615ns	4145ns	4145ns
(10,90)	5615ns	3945ns	3945ns

Fig. 14. Comparison of the operation time between the original MIPS, the MIPS with the 4-state prediction unit, and the MIPS with the prediction history table

(nb, incr)	no L2	16 Blocks 64 words	32 Blocks 128 words	64 Blocks 256 words	128 Blocks 512 words
HasHazard	160965ns	150365ns	150365ns	150365ns	150365ns
(3,3)	13185ns	13345ns	13345ns	13345ns	13345ns
(5,3)	30425ns	30625ns	30625ns	30625ns	30625ns
(10,3)	113295ns	109535ns	109535ns	109535ns	109535ns
(15,3)	267385ns	241385ns	241385ns	241385ns	241385ns
(20,3)	481175ns	437935ns	426335ns	426335ns	426335ns
(25,3)	753965ns	706035ns	662285ns	662285ns	662285ns
(30,3)	1085515ns	1046635ns	948735ns	948735ns	948735ns
(35,3)	1476305ns	1458845ns	1294495ns	1286685ns	1286685ns
(40,3)	1926095ns	1931945ns	1711195ns	1675635ns	1675635ns

Fig. 15. Operation time between different size of L2 cache in different nb

policies to synthesize different devices. Therefore, the gate-level circuit may not be synthesized as expected.

B. Branch Prediction

In Fig.12, the operation time of the MIPS with the 4-state prediction unit is shorter than the one of the original MIPS under all testdata, which proves that the the prediction unit improves the performance of the MIPS. However, it's strange to see that CHIP_ini's operation time is the same in all kinds of testdata. The result is caused by the difference of the operation time in a NotBranch testdata and a Branch testdata. One NotBranch testdata takes more cycles to complete than one Branch testdata does. As a result, although the CHIP_ini, whose prediction is AlwaysNotBranch, is supposed to perform better when there are more NotBranch testdata, the CHIP_ini's performance doesn't change with the change of testdata. In addition, the CHIP_NEW performs better as the number of NotBranch decreases. It proves that the NotBranch testdata really takes more cycles to complete than the Branch testdata. Moreover, from Fig.14, it can be clearly seen that CHIP_LATEST, which is attached with prediction history table, performs the same as CHIP_NEW, while CHIP_LATEST is supposed to perform better. The result can be explained

(nb, incr)	no L2	16 Blocks 64 words	32 Blocks 128 words	64 Blocks 256 words	128 Blocks 512 words
(3,3)	13185ns	13345ns	13345ns	13345ns	13345ns
(3,5)	25155ns	25355ns	25355ns	25355ns	25355ns
(3,10)	74895ns	74395ns	74395ns	74395ns	74395ns
(3,15)	162995ns	151405ns	151405ns	151405ns	151405ns
(3,20)	286365ns	259655ns	258325ns	258325ns	258325ns
(3,25)	441155ns	403335ns	393495ns	393495ns	393495ns
(3,30)	627795ns	58665ns	556375ns	556375ns	556375ns

Fig. 16. Operation time between different size of L2 cache in different incr

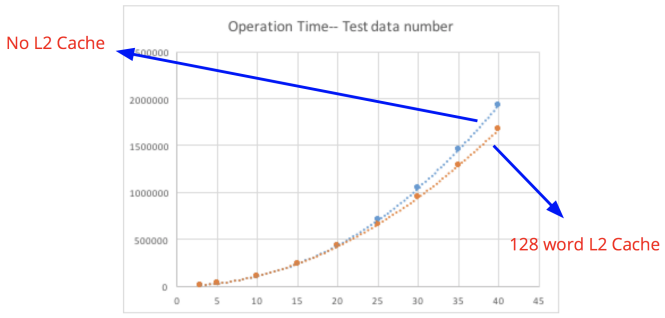

Fig. 17. Trend line of the operation time and the Two-Level cache MIPS

by the fact that the test data we utilized couldn't test the difference between the two different MIPS, since the testdata is too simple. Thus, in addition to the BrPred testdata, we tested the three different MIPS in Fig.14 with the hasHazard testbed, and the result is shown in Fig 19. The hasHazard testbed's nb represents the order of the wanted number in a fibonacci sequence, and as the nb gets bigger, the operation time will be longer, and the testdata sequence will be more complicated. Under the hasHazard testbed, the CHIP_LATEST has the smallest operation time due to the more complicated branch conditions in the testbed.

C. Two-Level Cache

From Fig.12 and Fig.13, the MIPS with Two-Level cache generally performs better than the original MIPS. Moreover, the MIPS with a bigger-sized L2 cache can have the shortest operation time under all testing cases, which is not the same as our prediction. The situation is due to the gap between simulation and the real hardware execution. In reality, as the L2 cache gets bigger, it takes more time for the L1 cache to find the wanted data. However, the size of L2 cache doesn't affect the operation time from L1 cache to L2 cache, and since the bigger-sized L2 cache can store more data from the memory, the hit rate from L1 cache to L2 cache will be raised. From the result, it seems that the efficiency of the MIPS can be raised simply by enlarging the size of the L2 cache, which is in fact not available. The failure results in the actual cost of cache. The cost of cache is about ten times higher than the memory, and it's not economical to enlarge the size of cache without limitation. On the other hand, as mentioned above, the time for accessing the L2 cache will be raised for a bigger-sized L2 cache, and the two main factors make it necessary to limit the size of the L2 cache.

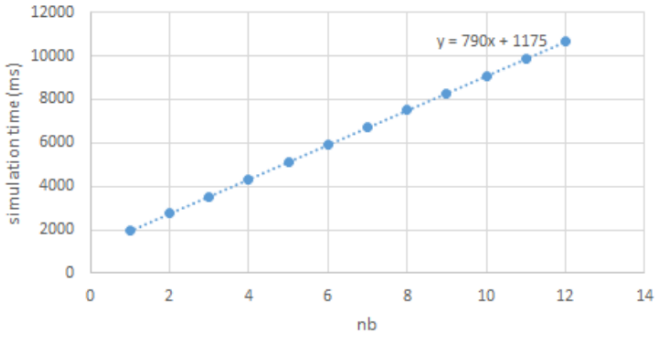

Fig. 18. The simulation time for each testbed

	CHIP_ini	CHIP_NEW	CHIP_LATEST
hasHazard	21295ns	22225ns	21285ns

Fig. 19. Operation time between different branch-prediction-mechanism MIPS under hasHazard testbed

D. Multiplication and Division

The algorithm used in iterative approach will make each operation of multiplication or division take 32 cycles to complete the computation no matter how large the number is. It is necessary for the multiplier or the dividend to be shifted 32 times in order to check on all the bits. However, it may be a waste in the case of operations for small numbers, several leftmost bits of which in the binary representation are all zeros. In order to avoid the waste in time, a comparator can be added to decide how many times the multiplier or the dividend should be shifted. For example, if the multiplier in a multiplication is smaller than 2^16, it only needs to be shifted 16 times because the 16 leftmost bits are all zeros and thus there's no need to check on those bits. In this case, the result can be obtained by directly shifting the HI-LO register by 16 bits after the former 16-cycle computation. By doing so, the operation take 16 cycles less than the original design, which can lead to an improvement on performance. By the way, the comparator can be added at ID stage so that the critical path will not be too long.

V. Conclusion

In our baseline design, we utilize three methods to improve the efficiency. The branch forwarding stall unit and the buffer technique makes great effort on improving the overall a-t value after a great number of tests. On the other hand, in the extension part, we accomplished the design of branch prediction, L2 cache, multiplication, and division, and meanwhile makes analysis under different conditions of testdata. In branch prediction, the MIPS with prediction history table performs the best, and in L2 cache prediction, without considering the cost of cache, the MIPS with a 256-word L2 cache outstands. In the design of multiplication and division, the iterative approach we utilized still has room for us to improve.

Acknowledgment

- All authors contributed equally.

- This work is the final project of Digital System Design. It is supported by the Department of Electrical Engineering, National Taiwan University.
- We appreciate Prof. An-Yeu Wu for providing many great advice.
- We appreciate TAs for the assistance whenever we have problems.

Yu-Sheng Ting is an undergraduate student in the Department of Electrical Engineering at the National Taiwan University (NTU) and will be graduating in 2021 with a BS in Electrical Engineering. His research projects mainly focus on machine learning on network optimization problems.

Ting Wang is an undergraduate student in the Department of Electrical Engineering at the National Taiwan University (NTU) and will be graduating in 2021 with a BS in Electrical Engineering. His research projects mainly focus on cryptography.