Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 6.017

IJCSMC, Vol. 6, Issue. 6, June 2017, pg.302 – 306

Enhancing Performance in Multiple Execution Unit Architecture using Tomasulo Algorithm

Vasantha.N.S.[1], Meghana Kulkarni[2]

[1]Department of VLSI and Embedded Systems, Centre for PG Studies, VTU Belagavi, India

[2]Associate Professor Department of VLSI and Embedded Systems, Centre for PG Studies, VTU Belagavi, India

[1] vasanthareddyns@gmail.com ; [2] meghanak@vtu.ac.in

Abstract— Tomasulo's algorithm is a computer architecture hardware algorithm for dynamic scheduling of instructions that allows out-of-order execution, designed to efficiently utilize multiple execution units. It was developed by Robert Tomasulo at IBM. The major innovations of Tomasulo's algorithm include register renaming in hardware. It also uses the concept of reservation stations for all execution units. A common data bus (CDB) on which computed values broadcast to all reservation stations that may need them is also present. The algorithm allows for improved parallel execution of instructions that would otherwise stall under the use of other earlier algorithms such as scoreboarding.

Keywords— Reservation Station, Register renaming, common data bus, multiple execution unit, register file

I. INTRODUCTION

The instructions in any program may be executed in any of the 2 ways namely sequential order and the other is the data flow order. The sequential order is the one in which the instructions are executed one after the other but in reality this flow is very rare in programs. Out of order execution of instructions happens in most of the programs due to existence of control structures, loops, jumps etc. This out of order execution leads to errors such as read after write (RAW), write after read (WAR) and write after write (WAW). It also leads to incorrect jumps and interrupts. These problems also occur in case there is multiple execution units as the result of one may be over written on to another causing errors.

Tomasulo algorithm makes sure that out of order execution of instructions takes place without any of these errors by using a concept called as reservation station. The errors can occur also due to an instruction waiting for an operand but it will be sent to the execution unit with random values. For example say that there is an instruction which adds the data in registers 1 and 2 and saves the data in register 3. Say there is another instruction which adds the contents of register 3 and 4 and saves it in register 5, and then the 2nd instruction has to wait for the 1st instruction to complete as the register 3 content is calculated by the 1st instruction. But when we go for sequential flow of instructions it does not wait for the 1st instruction to complete and computes the 2nd instruction with random value of register 3 which leads to errors.

As a result, to avoid errors Tomasulo algorithm was proposed. The reservation station is the place where the instructions are buffered from the instruction fetch unit and are dispatched only when both the operands are ready. The reservation station holds the opcode and the operands which are to be dispatched to the execution unit. The instruction which is sent from the instruction fetch unit is decoded and the addresses are sent to the register to get the operand values from the register. The destination address is made to wait for the result of the instruction by saving the reservation station index in it. Once the execution unit broadcasts the result along with the reservation station id the destination register is updated. Simultaneously the reservation station operands which were waiting for this result also update themselves as the bus is common to both the register file and the reservation station.

It was developed by Robert Tomasulo at IBM. It uses the reservation station to do register renaming that is the registers are renamed by putting the reservation station tag id in them rather than the register index which would lead to errors. This algorithm helped in parallel execution of instructions in an error free manner. The number of reservation stations depends on the number of execution units. Ideally we would like to have one reservation station per execution unit. The speed of execution also increases as there would be parallel execution of instructions in different execution units such as adders, multipliers, dividers etc.

Reservation stations control the flow of instructions to the execution units. They are dispatched only when both the operands are ready. The registers are renamed with tag ids of reservation station to ensure that no instruction is dispatched with wrong values of operands. This renaming is done in hardware. Name dependency is removed.

The speed is improved by using a common data bus between the reservation station and register file. The instructions which are waiting for the result of an instruction need not have to wait till the result is updated in the register file and then access it, the result is readily available to the reservation station via the common data bus and thus 2 clock cycles can be saved in this process. The instructions in the reservation station which were waiting for the result can be dispatched directly thus improving the processing speeds. This involves in increasing the hardware in the way of adding reservation station, but at the same time it ensures proper execution of instructions, parallel execution and also faster execution of instructions. Thus a tradeoff between speed and hardware is done here. The general block diagram of Tomasulo algorithm is as shown below.

II. LITERATURE SURVEY

In a paper by Acosta, Jacob and Torng which deals with improving the performance in multiple functional unit processors the concept of Tomasulo algorithm has been used. The reservation station proposed by Tomasulo algorithm has been used. It deals with processors such as CRAY FPS, etc which are used for scientific computations. It uses a concept of dispatch stack which is an instruction issuing scheme similar to the reservation station which ensures orderly execution of instructions.

This instruction issuing scheme helps in dynamic scheduling of instructions at run time rather than the sequential order if issuing the instructions. The paper also deals with issuing and executing multiple instructions simultaneously so that the throughput may be increased. According to this paper there are several ways to increase the throughput that is increasing the number of instructions executed per unit time such as reducing the clock cycles, by reducing the access time of the memory and by keeping the execution units as busy as possible without allowing it to be free. The dispatch stack concept is slightly different from that of the reservation station and it was conceived by Torng. [1]

In case of the dispatch stack concept it concentrates on increasing the throughput by issuing multiple instructions per clock cycle, but the errors such as raw, war and waw are still a concern here. Tomasulo's reservation station even though slower than the dispatch stack of Torng makes sure of the error free execution of instructions. There are a few short comings of Tomasulo reservation station which holds up data independent instructions and slows down their execution speeds but it is very effective technique to dynamically schedule data dependent instructions. [2]

In another paper by McMillan we see that the Tomasulo algorithm is verified with the help of compositional model checking technique. This paper checks the implementation of the finite state machines for out of order execution of instruction techniques. The technique of register renaming is formally verified here. The verification carried out is simple in nature. It is done at the bit level. A verifier such as PVS can be used for this purpose. Symbolic checking techniques can also be used to verify the Tomasulo algorithm.

The proof is done using auxiliary variables and maps which are of refinement in nature. These maps represent the semantics of the desired state. The Tomasulo algorithm increases the speed of execution by avoiding stalls in the pipeline and immediately releases an instruction as soon as it is ready given that the execution unit is free. If the execution unit is not free then only the release is stalled. If all the reservation stations are full also then using a gating signal the instructions are stalled from the instruction fetch unit. [3]

The verification was carried out with a few assumptions which are important for us in this project. The result of arithmetic instructions by themselves holds no importance as the order of execution is what is important here. The verification was successfully carried out using compositional techniques which gives us proof that the Tomasulo is a valid technique for executing instructions with the help of dynamic scheduling. The help of state machine was used in the process of verification.

Based on these 2 papers we took up the project of implementing the Tomasulo algorithm by using Verilog Behavioral model with the hope of removing the hazards as well as increasing the speed of execution of instructions. The common data bus helped to save essential clock cycles. Multiple functional units which could be used in parallel also helped speed up the process. The reservation stations made sure that the execution took place without any hazards. The verification of the execution of instructions was done by simulating using isim simulator which conformed to the behavior expected of the algorithm. [4]

III. METHODOLOGY

Coding of the project is divided into different parts which will be later integrated using a main program.

A. BLOCK DIAGRAM

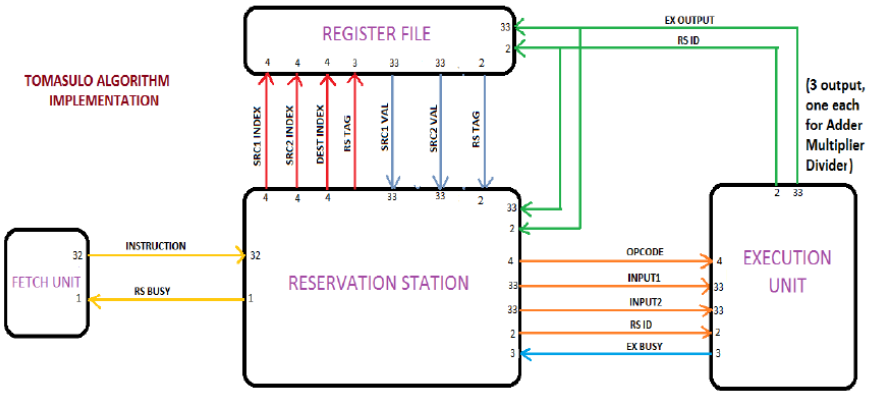

Figure 1 Block diagram of the Tomasulo algorithm

B. DESIGN OF THE INSTRUCTION FETCH UNIT

The instruction fetch unit sends out instructions at each clock cycle. There is a signal called Reservation station busy bit which is anded with the global clock to generate the instruction clock. This will ensure that whenever all the reservation stations are full the instructions are not sent from the instruction fetch unit. When there is no gating signal the instruction fetch unit sends out instructions. Different instructions are sent from the instruction fetch unit for the purpose of simulation. There is a 4 bit counter which generates a signal to acts as clock for the Instruction fetch unit.

C. DESIGN OF RESERVATION STATION

Reservation station is an intermediate buffer between the instruction fetch unit and the execution units. It makes sure that an instruction is released only when both the operands of the instruction are ready and thus the hazards such as war, waw and raw are avoided. A 32 bit instruction stream is received from the instruction fetch unit. In this the first 16 bits is relevant information and the last 16 bits are ignored. The first 4 bits are the opcode which has to be stored as a part of the 75 bit reservation station. These are the bits from 71 down to 68. The 73rd and 72nd bit forms the reservation station index. There are 4 such reservation stations each of 75 bits. The other 12 bits of the instruction stream are the 4 bit source and destination indices to be sent to the register file from the reservation station during the positive half of the clock cycle.

The register file sends 2 source values each of 33 bits to the reservation station. It saves the reservation station index which is sent from the reservation station at the destination address. The 33rd bit at the destination is made one to indicate that the data stored is a tag id and not the actual value. The 33 bit data coming from the register file are saved from bits 32 down to 0 and 66 down to 35. The 34th bit indicates whether the operand 1 is ready. The 67th bit indicates whether the operand 2 is ready. The 75th bit indicates whether the instruction in the reservation station is ready and can be dispatched to the execution units.

The reservation station sends 2 source operands each of 33 bits to the execution unit over the next positive half of the clock cycle. It also sends 2 bit tag id along with the operands to the execution units so that when the execution unit completes computation the result can be updated at the register file and reservation station accordingly.

The reservation station has one more important task. As soon as the results are computed by the execution unit it is put on the common data bus during the positive half of the clock cycle. This result has to be immediately updated in the operands which are waiting for this result holding the tag id of that particular reservation station. The reservation station whose tag id matches that of the tag id on the data bus has finished its duty and this it is emptied.

D. DESIGN OF EXECUTION UNITS

In this project we have considered only blocks which compute mathematical functions such as addition, subtraction, multiplication and division. One single execution unit satisfies both the functions of addition and subtraction as we have used an adder/ subtractor unit which acts as adder if a particular bit is zero an as a subtractor if that bit is one. The execution units have inputs from the reservation station and they give out output to the common data bus. The opcode which is used to indicate is a particular operation is decoded by using a 4 to 16 decoder out of which only 3 are used.

Based on these opcode the particular execution unit will be invoked and the computation is carried out. For addition or subtraction it takes 1 clock cycle, for multiplication it takes 2 clock cycles and for division it takes 3 clock cycles. Thus for one instruction to be computed from instruction fetch unit to the execution unit it takes 4 clock cycles. Thus for addition it takes 5 clock cycles. For subtraction also it takes 5 clock cycles. For multiplication it takes 6 clock cycles. For division it takes 7 clock cycles.

E. DESIGN OF REGISTER

The register file is a combination of 16 registers each of 32 bits along with a T/V bar bit (MSB) which indicates that the data in the register is either tag id if T/V bar bit is 1 or value if T/V bar bit is zero. The data is initialized to five at the beginning to all the registers. Afterwards according to the computations each register gets updated. For simulation purpose we have taken 8 instructions and all the registers are updated after the completion of the simulation. The register receives the indices of the source and destination which act as pointers to the registers. This ensures that the wrong over writing on the registers are avoided.

IV. CONCLUSIONS

The project based on Tomasulo algorithm to carry out dynamic scheduling of instructions has been proposed using Xilinx ISE tool. The hazard free out of order execution of instructions along with increasing in computation speed is taking place in the sample being tested. The concept of reservation station as proposed by Tomasulo was used as an intermediary between the instruction fetch unit and the register file as well as between the instructions fetches unit and the execution units. This helped in removing the errors such as RAW, WAW and WAR.

The concept of common data bus helped in saving crucial clock cycles as it simultaneously updated the results to both the reservation station and the register file. With the help of multiple execution units, parallelism and pipeling we were able to bring down the execution time from 365ns to 205ns which is an improvement of 43%. These results pertain to the sample that has been chosen. The results for the further samples have been tabulated in table 1.

There are areas for further research based in this project. The reservation station size, register file size, number of execution units can all be increased. There is scope to involve parallelism during transfer of instructions between reservation station and execution unit which is being done serially now. The performance enhancement in case of multiple execution unit architecture with the help of Tomasulo algorithm has been tested.

TABLE I Run Time for Various Instruction Sets with and without Tomasulo Algorithm

Instruction Set	Run time(no of cycles)	Run time without Tomasulo algorithm	Performance Improvement (%)
1	21	37	43.24
2	26	39	33.33
3	27	32	15.62
4	25	42	40.47
5	20	39	48.72

ACKNOWLEDGEMENT

I have taken efforts in this Project report. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to Dr. Meghana Kulkarni for her guidance and constant supervision as well as for providing necessary information regarding the Project & also for his support in completing the Project.

I am thankful to the external guide and the staff at KuVi Innovations Pvt. Ltd. for their constant support and encouragement.

I also thank our H.O.D. of VLSI and Embedded Systems Department, Dr.T.C.Thanuja, for providing her valuable suggestions and constant encouragement in completing the Report.

I would like to express my gratitude towards my parents & members of VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAVI for their kind co-operation and encouragement which helped me in completion of this Internship.

REFERENCES

[1] RAMON D. ACOSTA, JACOB KJELSTRUP AND H. C. TORNG, *An Instruction Issuing Approach to Enhancing Performance in Multiple Functional Unit Processors*, IEEE TRANSACTIONS ON COMPUTERS, VOL. C-35, NO. 9, SEPTEMBER 1999.

[2] K. L. McMillan, *Verification of an Implementation of Tomasulo's Algorithm by Compositional Model Checking*, Cadence Berkeley Labs, Berkeley, CA.

[3] Matt K., and Andrew R, *Implementation of Reservation station*, The 25th Annual Symposium on Hardware Architecture, IEEE Computer Society Press, May 2005.

[4] Hick, Matteo D, *Aspects of Tomasulo Algorithm and Instruction Buffer Performance,* M.S. Th., University of Austin, Texas, 2016.

[5] Walker, Davis W, *Available Instruction-Level Parallelism*, Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, IEEE Computer Society Press, April 2015.

[6] Norman P, *Architectural and Organizational Tradeoffs in the Design of the Multi Titan CPU*, The 20th Annual Symposium on Computer Architecture, IEEE Society Press, May, 2012, pp. 281-289.

