Scoreboard Algorithm With Cache Implementation

Phani Teja Kesha
Dept of CSEE
University of Maryland
Baltimore County
Arbutus, United States
Phani1@umbc.edu

Bhanu Phanindra
Dept of CSEE
University of Maryland
Baltimore County
Arbutus, United States
bhanuph1@umbc.edu

Manoj Kashyap
Dept of CSEE
University of Maryland
Baltimore County
Arbutus, United States
manoj.kashyap@umbc.edu

***Abstract*—A Simulator for MIPS architecture with Dynamically Scheduled Pipelines using ScoreBoard algorithm. This project also allows us to modify and reuse the simulator for analyzing cache performance for different number of blocks, block sizes and cache size.**
***Keywords*—Dynamic Scheduling, MIPS Architecture, RISC, ScoreBoard Algorithm.**

I. Introduction

Dynamic scheduling, as its name implies, is a method in which the hardware determines which instructions to execute, as opposed to a statically scheduled machine, in which the compiler determines the order of execution. With dynamic scheduling, the processor can execute instructions out of order. This helps in improving the speed by executing instructions without considering the order in which they appear. Although the resources for the processor needs to be accounted for. Execution of instructions takes place considering the availability of source operands and functional units.

Parallelism can be utilized by dynamic scheduled machine which cannot be visible during the compile time. Since hardware takes care of most of the scheduling, thus the code does not have to be recompiled to run efficiently. As far as statistically scheduled hardware is concerned the code has to be recompiled to have the hardware advantages in it.

Instuction Hazards

To control the flow of data among registers and multiple arithmetic units during conflicts due to lack of sufficient hardware resources (structural hazards) and dependencies in instructions (data hazards), Scoreboards were designed. Data hazards are complicated, they can be view as output dependencies (write-after-write), anti-dependencies (write-after-read) and flow dependencies (read-after-write).

Read-After-Write (RAW) Hazards

A read-after-write hazard takes place when the result of a previously issued instruction is required by currently running instruction.

Example:
R6 ← R1 x R2
R7 ← R5 + **R6**

Write-After-Write (WAW) Hazards

Write-after-write hazard occurs if both current instruction and previously issued instruction try to write their result to the same register.

Example:
R6 ← R1 * R2
R6 ← R4 + R5

Write-After-Read (WAR) Hazards

Write-after-read hazard takes place when the current instruction tries to write its result to a register which is yet to provide value to the previously issued instruction. This is a very rare hazard that do not usually occur.

Example:
X5 ← **X4** * X3
X4 ← X0 + X6

Structural Hazards

Structural Hazard occurs when the same hardware resource is accessed by two or more instructions. This would occur if an instruction requiring arithmetic unit for three clock cycles is issued while its previous instruction requires different arithmetic unit for four clock cycles.

MIPS is a reduced instruction set computer (RISC) architecture developed by MIPS Technologies. The MIPS architectures were initially 32-bit later in the course of time 64-bit was introduced. During mid 1900s gaming consoles such as PlayStation Portable, Nintendo 64, PlayStation 2 etc., used processors that use MIPS architecture. During early 2000 and late 1990s even supercomputers used processors of MIPS architecture. This architecture was very popular in embedded applications at first, and later on MIPS became a major presence in the embedded processor market and by the 2000s, most MIPS processors were built for embedded systems. It was estimated then that one in three RISC microprocessors produced was from MIPS technologies.

Scoreboard algorithm is built for dynamically scheduling pipelines so that the execution can be in out of order. Instructions enter pipelining stages when there is no hazard thus, instructions are checked at each stage of pipeline for hazard and handled.

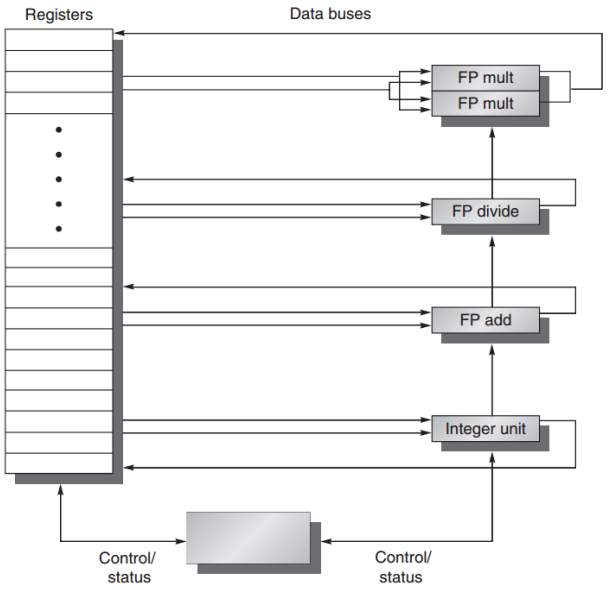

Fig. 1. Scoreboard Simulator Overview

ScoreBoard Algorithm
Instructions in Scoreboard algorithm go through the sthese stages

Issue Stage
In this stage registers effected by this instruction are known, to avoid dependencies are also known that cause write-after-write hazard and structural hazard. If a dependency is found then the instructions are stalled until instructions writing their result to same register are completed one after the other in their actual order. In this stage instructions can be stalled when they require functional units that are busy a that moment.

Read Stage
During this stage when an instruction is issued to the required hardware module, the instruction has to wait until all the operands required by it are available. This avoids read dependencies in the system a.k.a. read-after-write hazard since registers which were being modified by another instruction are considered unavailable until the write is complete.

Execution Stage
When all the required operands are fetched, the instruction starts its execution on functional unit. ScoreBoard is notified as soon as the result is ready.

Write Result Stage
Result of the instruction is written into the destination register in this stage, although this stage sounds simple it has to be delayed until the earlier instructions which are yet to read data from the register have completed their read stage from the register to which this executed instruction has to write result to. This process is followed to handle data dependency i.e., write-after-read hazard

II. History

Static scheduling was followed by the systems in the initial computers. Compiler techniques were used to improve the efficiency for static scheduling of instructions. Even though this method could avoid most hazards and stalls, the total number of clock cycles required for each instruction to get executed was huge. To decrease the clock cycles and improve the efficiency of the system dynamic scheduling proved to be an effective solution

Intel created its first processor 4004 which had about 2300 transistors in it. Currently available chips have more than 15 million transistors, the reason for having such large number of transistors is for the caches. Caches must have many paths connected to write and read ports. For some caches, store/load unit should be able to access every point of the cache. This gets worse when there are a lot of store/load units. Considering Pentium Pro by Intel as an example, its CPU chip developed had about 5 million transistors, a single package of it included an L2 cache and CPU chip. Comparing 4004 processor and todays processors it is clear that the processors are becoming more and more complex with time. This complexity is due to chip designers trying to improve the processing technology and to develop a faster and efficient processor and all of this with affordable cost. As processor technology developed more number of transistors were able to fit in same die area. It soon became cost effective with improved features in processor making them effective and faster. One of such developments was dynamic scheduling.

III. Methodology

In this project we implemented MIPS simulator using scoreboarding algorithm in eclipse environment using JAVA. We also made this simulator as flexible as possible. Simulator performance and results are independent of hardware used for simulation because of same java virtual machine layer that runs on top hardware.

A. *MIPS simulation*

The following section tells about techniques used to implement simulator. Reason for choosing java platform is its excellent IDE eclipse, vast array of libraries and huge amount of documentation available. Some libraries used include import java.util.* for hashmap and arraylist, import java.io.* for exception handling and file reading. When program is started instruction set is read from text file and parsed along with configuration and data text files which contain which contain initial state of registers. After instruction is parsed operation to be performed is matched and registers utilized is noted

down. Functional unit count is checked and if available and destination registers are marked free from flags and advanced to next stage in pipeline. Registers in use are marked using flag bits which are used for identifying hazards. These registers are kept busy based on operation clock cycles, after which registers are marked free and ready for next use. Challenges faced while implementing are bringing modularity to code and making simulator flexible along cache and memory latency simulation.

B. Configuring Simulator

The advantage of using simulator over physical counterpart is flexibility. It is difficult to scale or making small adjustments is both time consuming and brings additional cost. With software simulation tools it possible to configure processor specifications like functional units cache size, time taken by each operation, clock frequency. Thus simulator has been best choice for many debuggers from testing new Branch prediction strategies to designing new cache mapping with new replacement policy. Things that can be easily configured in our project include functional units count, cycles taken to perform that function, cache size in number of blocks and number of blocks. These are stored in text file and every time simulator program is executed, it reads simulator

C. Cache Implementation

MIPS machine is assumed to have an instruction cache (I-Cache) with an access (hit) time of one cycle per word. The organization of I-Cache is direct-mapped. In addition, the machine has a data cache (D-Cache) with hit time of one cycle per word. D-Cache is a 2-way set associative with a total of four 4-words blocks. When cache is full some replacement policy is necessary to replace block with new block. For this simulator LRU(least recently used policy) is applied.

The I-Cache and D-Cache are both connected to main memory using a shared bus. Due to shared bus mechanism when miss occur in both D-cache and I-cache, a priority scheme is chosen and one of them is served first. Here I-cache is served first. If main memory is busy serving the other cache we have to wait for it to be free and then start accessing it.Therefore there will be possibility of higher memory latency than given cycles depending on the time of a request and the state of Main MemoryRegister FileInstruction CacheData. The main memory is accessible through one-word-wide bus, and its access time is 3 cycles per word. If both caches experience a miss at the same cycle, the I-Cache will be given priority. An instruction cache miss will add the I-Cache miss penalty to the fetch time.

When unconditional jumps are encountered in issue stage, already fetched instruction is useless and must be flushed out due to which one clock cycle is wasted. But conditional branches are resolved in the Read stage. Meanwhile the CPU will go ahead and fetch the next instruction and flush instructions if necessary (depends on branch predictor).

IV. Experimental Results and Analysis

The MIPS simulator with scoreboarding can be configured for different block sizes and word sizes of the instruction cache. Let us consider the following instruction set for the cache size as 4,2 where 4 is the block size and 2 is the number of words per block.

LI R2,272
LI R3,300
L.D F6,34(R2)
L.D F2,45(R3)
MUL.D F0,F2,F4
SUB.D F8,F6,F2
DIV.D F10,F0,F6
ADD.D F6,F8,F2

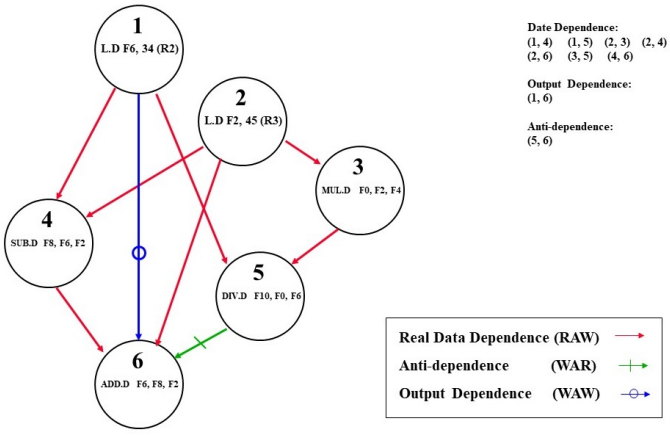

Fig. 2. Dependancy in the above instructions

The configuration of the MIPS simulator is

FP adder: 2, 2
FP Multiplier: 2, 10
FP divider: 1,40
Instruction Cache: 4,2

Label	Instruction	Fetch	Issue	Read	Exec	Write	RAW	WAW	Struct
	LI R2,272	7	8	9	10	11	N	N	N
	LI R3,300	8	12	13	14	15	N	N	Y
	LD F6,34(R2)	15	16	17	36	37	N	N	N
	LD F2,45(R3)	16	38	39	59	60	N	N	Y
	MUL.D F0,F2,F4	38	39	61	71	72	Y	N	N
	SUB.D F8,F6,F2	39	40	61	63	64	Y	N	N
	DIV.D F10,F0,F6	46	47	73	113	114	Y	N	N
	ADD.D F6,F8,F2	47	48	65	67	68	Y	N	N

Fig. 3. Output of the simulator

Total number of access requests for instruction cache: 8 Number of instruction cache hits: 4 Instruction Hit Ratio = 0.5 Total number of access requests for data cache: 4 Number of data cache hits: 2 Data Hit Ratio = 0.5

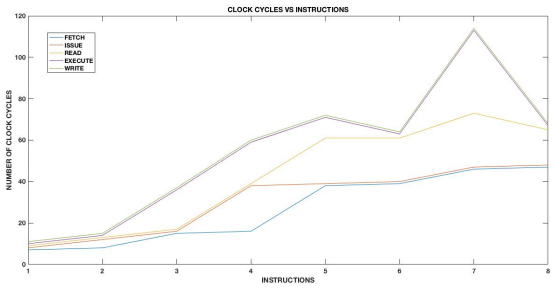

Fig. 4. Visualization of output

From the above analysis we can draw some conclusions. Wherever there is a hazard and the instructions are stalled then we can see large value of slopes for given particular stage. The total number of clock cycles required for the given assembly language code to get executed is 114 for the configurations mentioned in the configurations file. Now let us consider a instruction set which has more number of instructions for benchmarking purposes. The cache size is fixed but the block size is varying. The assembly code used here and other codes tested on this simulator is included in zip as a assemblyprograms.txt file.

FOR FIXED CACHE SIZE AND VARYING BLOCK SIZE

The number of functional units and clock cycles taken for that operations is same through out this test. Its values are

FP adder: 2, 2
FP Multiplier: 2, 10
FP divider: 1, 40

Instruction-Cache:1,20
Number of Instruction Cache Requests: 88
Number of instruction cache hits: 81
instruction cache hits=81/88

Instruction-Cache:20,1
Number of Instruction Cache Requests: 88
Number of instruction cache hits: 12
instruction cache hits: 12/88

Instruction-Cache: 4,5
Number of Instruction Cache Requests: 88
Number of instruction cache hits: 69
instruction cache hits: 69/88

Instruction-Cache: 5,4
Number of Instruction Cache Requests: 88
Number of instruction cache hits: 69
instruction cache hits: 69/88

Instruction-Cache: 10,2
Number of Instruction Cache Requests: 88
Number of instruction cache hits: 50
instruction cache hits: 50/88

Instruction-Cache: 2,10
Number of Instruction Cache Requests: 88
Number of instruction cache hits: 76
instruction cache hits: 76/88

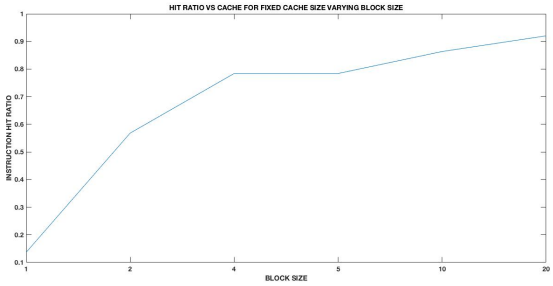

Fig. 5. Visualizing cache performance with variable block size

From the above graph we can generalize that if the block size of the cache is more then the hit ratio is generally high. This is due to the reason that if the block size is high then the cache loads all the instructions in the cache. The disadvantage for large block size is when there are to many branching conditions , if the branch condition fails then all the instructions which are fetched into the cache must be flushed. To overcome this challenge we need to choose optimum block size such that the performance is balanced.

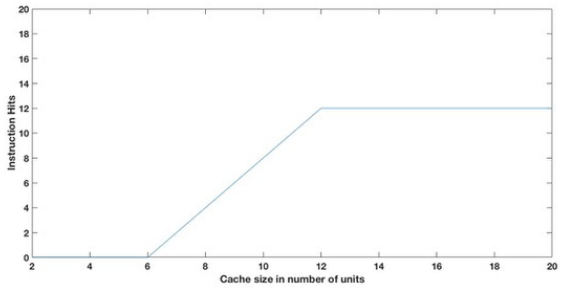

Fig. 6. Visualizing cache performance with variable cache size

In above graph is obtained by executing same program by varying cache size and keeping block size constant. From the graph we can conclude that initially there is significant increase in performance with increase in cache size, but after certain threshold there is no noticeable performance improvement(saturated). We can infer that programs can only take advantage of cache size only up to certain point.

V. Conclusions and Future Work

In above graphs and visualizations are rendered as expected and stalls can be noticed as well as large slopes in Figure.4. Also cache performance in Figure.5 shows dependency with block size. Up to certain block size advantages dominate its

cost. For large values of cache we can see that clock cycles taken for completion of same code is high, thus limiting its usefulness.

Future Works can include implementing multiple state of the art branch predictor in simulator and having multiple cache levels.

VI. Acknowledgements

We would like to thank, Dr. Ting Zhu and Mr. Yao Yao without whom this project would not be possible, We are grateful for providing us this opportunity to showcase our project and for the help provided to us for completing this project.

VII. Team Member Roles

Phani - Literature Survey - Parsing Instruction, configure files and Implementation of caches .

Bhanu - Literature Survey - Performing Operations and Hazard Handling

Manoj - Literature Survey - Report writing, creating and testing programs and visualizing plots.

References

[1] Harsh Arora and Yash Rajpurohit, "Enhanced Cycle Simulator for MIPS Architecture", Computer Modelling and Simulation (UKSim), 2013 UKSim 15th International Conference, June 2013.

[2] A.H.S. Lai and N.H.C. Yung, "A fast and accurate scoreboard algorithm for estimating stationary backgrounds in an image sequence", Proceedings of the 1998 IEEE International Symposium, August 2002.

[3] A. Bashteen, I. Lui and J. Mullan, "A superpipeline approach to the MIPS architecture", Compcon Spring '91 Digest of Papers, August 2002.

[4] Patrick Ian Bautista, Arnold Cruz and Amor Corazon Rio, "PROJECT GRAMS: A graphical reconfigurable architecture MIPS simulator", TENCON 2008-2008 IEEE Region 10 Conference, January 2009

[5] Lisheng Wang, Jianhong Ruan and Dongdong Zhang, "MIPS CPU test system for practice teaching", Computer Science and Education, 2017 12th International Conference, October 2017, pp-2473-9464.

[6] Safaa S. Omran and Laith F. Jumma, "Design of multithreading SHA-1 and SHA-2 MIPS processor using FPGA" Information Technology (ICIT), 2017 8th International Conference, October 2017.