ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

Hazard Detection and Data Forwarding Scheme for 5-stage Pipeline Structure of RISC Processor

Ankur Changela

Assistant Professor, Department of Electronics and Communication, Indus Institute of Technology and Engineering, Indus University, Ahmedabad, India.

ABSTRACT: This paper is about the implementation of data forwarding for 32 bit RISC processor in VHDL. The 32-bit RISC processor core is implemented using a five-stage pipeline consisting of fetch, decode, execute, memory and write back stages. Data Forwarding, stated in this paper is used to forward data into a previous stage of the pipeline when data dependency occurs due to sequential execution of instruction. This paper aims at reducing RAW Hazard. Load instruction takes only two clock cycles when the data from the load required in very next clock cycle. Simulation and functional verification of the VHDL code were carried out on Model-Sim.

KEYWORDS: RISC Processor, Instruction cycle, Pipeline, Hazards.

I. INTRODUCTION

Portable applications such as mobile phones, pagers, and PDAs are continuously growing in sophistication. This places increasing burden on the embedded microprocessor to provide high performance. Higher performance, in processors can be achieved by having multiple stages of pipeline. Data Forwarding has been introduced to the pipeline to reduce the number of interlock cases and hence, reduce the average number of Clocks per Instruction (CPI). The 32-bits RISC processor core is implemented using a five-stage pipeline consisting of fetch, decode, execute, memory and result write-back stages. The device has Harvard architecture, and the simple bus interface. The Data forwarding paths in the processor allow back-to-back data processing instructions to execute in the pipeline without stall cycles. Load data, available at the end of the memory cycle, is also forwarded into the pipeline. If the data from the load instruction is required in the very next cycle then, there is one cycle interlock. This happens because, the data is not returned until the end of the memory cycle and the load instruction occupies 2 execute cycles in the data-path.[1][9] However, if the data is not required until the next instruction then the data is forwarded without interlock and the instruction occupying one execute cycle in the data-path.

II. RELATED WORK

A hazard is a situation that stops the successful execution of successive instructions in instruction pipeline resulting in an incorrect computation. Due to this, hazards increase the CPI (clocks per instruction) and reduce the processor performance. [6] There are three classes of Hazard. [7][8]

1. **Structural Hazards:** They arise due to resource conflicts.

2. **Data Hazards:** They arise when an instruction depends on the result of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline.

3. **Control Hazards:** they arise from the pipelining of branches and other instructions that change the PC.[2]

There are several methods to reduce hazards including pipeline stalls, pipeline bubbling, data/register forwarding, and in the case of out-of-order execution, the score boarding method and the Tomasulo algorithm. [10] [11] Data forwarding, also known as register forwarding, is used to reduce data hazard. There are three types of data hazard. [5]

1. RAW(READ AFTER WRITE)
2. WAR(WRITE AFTER READ)
3. WAW(write after write).

Data forwarding described in this paper aims at reducing the RAW type of hazard. To understand how RAW hazard occurs, consider two instructions I1 and I2 where I1 occurs before I2 in program order. A read after write (RAW) data hazard refers to a situation where an instruction tries to read a result that has not yet been calculated or retrieved. This can occur because even though an instruction is executed after a previous instruction, the previous instruction has not been completely processed through the pipeline. [12] To understand this problem, let us take an example of sequence of instructions. Instructions are being executed in order.

I1: Add R1, R2, R3 (R1$\leftarrow$ R2+R3)
I2: Add R7, R4, R1 (R7$\leftarrow$ R4+R1)
I3: Add R8, R6, R7 (R8$\leftarrow$ R6+R7)
I4: STR R1, R10, offset (R1 $\leftarrow$ mem(R10+offset))
I5: SUB R11, R1, R9 (R11 $\leftarrow$ R1-R9)

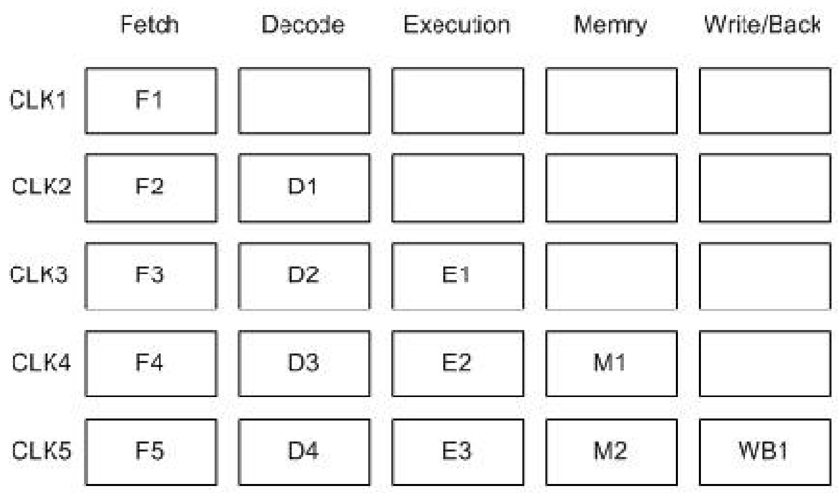

Fig.1 Instruction Execution in 5 stage pipeline

As shown in the Fig1, during the first clock cycle, I1: Add R1, R2, R3 is being fetched in the next clock cycle, fetch of I2: Add R7, R4, R1 and decode of the I1: Add R1, R2, R3 are being executed. In third clock cycle fetch of I3, Decode of I2: Add R7, R4, R1 and Execution of I1: Add R1, R2, R3 are being executed. In decode stage source registers value will be read. In case of I2 instruction value of R4 and R1 will be read in third clock cycle. But the value of R1 is updated by the I1 instruction and new value of R1 is written back in 'write back' stage of pipeline. Since the value of R1 is being updated in fifth clock cycle in 'write back' stage, updated value of R1 will not be read. There may be a chance that decoder reads the wrong value of register R1.To avoid this problem, pipeline has to be stalled for three clock cycles, increasing the CPI and degrading the performance of processor.[2][3]

III. Proposed Scheme for Hazard Detection and Data Forwarding

To solve this problem without stalling the pipeline, hazard has to be detected first. To understand how the hazard can be detected, let us take another example of sequence of instructions:

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

I1: MVI R3,427936(R3 427936)
I2: MVI R4,432544(R4 432544)
I3: ADD R5, R3, R4(R5 R3+R4)
I4: ADD R8, R7, R9(R8 R7+R9)
I5: SUB R11, R3, R10(R11 R3-R10)

In this example, Source registers for Instruction I3 are R3 and R4 and destination register for instruction I1 and I2 is R3 and R4 respectively. Due to this, instruction I3 completely depends on two previous instructions I1 and I2 and there will be a wrong execution of instruction if pipeline is not stalled. To detect this type of hazard, the source register of currently executing instruction is compared with the destination register of previous three instructions. In above example instruction I5:SUB R11, R3, R10 also depends on instruction I1: MVI R3,427936. But when the decode of instruction I5 is being executed, the value of register R3 is already updated in previous clock cycle so it is not necessary to compare the source registers of I5 with I1. Hence it justifies that only the destination of previous three instructions has to be compared with source of currently executing instruction. The problem with data hazards, introduced by this sequence of instructions can be solved with a simple hardware technique called 'forwarding'. Forwarding involves feeding output data into a previous stage of the pipeline. Forwarding is implemented by feeding back the output of an instruction into the previous stage(s) of the pipeline as soon as the output of that instruction is available.[4] To avoid this hazard, register value must be forwarded in the pipeline as shown in the Fig2 below.

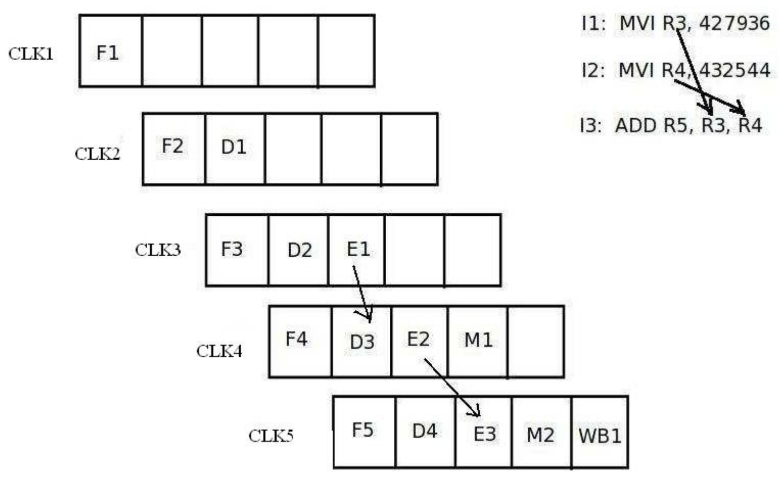

Fig.2: Data Forwarding Structure

Instruction I3 depends on instructions I2 and I1. Register R3 will be written back in fifth clock cycle and R4 will be written back in 6 clock cycle. Whereas, instruction I3 tries to read the registers value of R3 and R4 in fourth clock cycle in decoder stage. To avoid the decoder from reading a wrong value of register R4, correct value of register R4 must be forwarded from E2-M1 stage to D4-E3 stage as shown in Fig 2. Same way, the correct value of R3 register must be forwarded from E1 to F4-D3. Same concept is used in all data dependencies occurring at different stages of pipeline.

IV. Simulation Results

Simulation is carried out on Model-Sim. Different cases of data dependencies are simulated and results are shown in the figures below.

Case1:
MVI R3,427936
MVI R4,432544

ADD R5, R3, R4

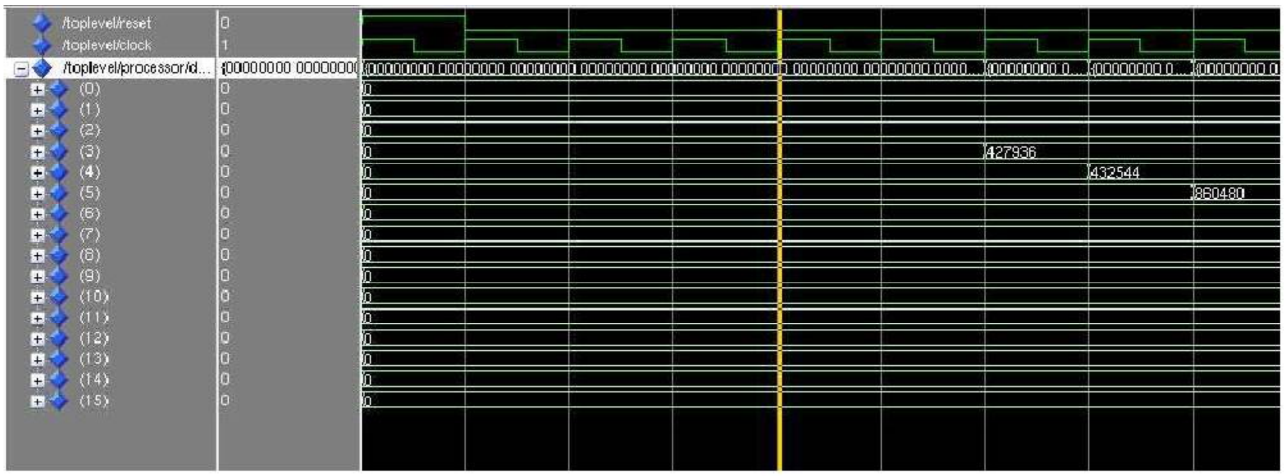

Fig.3: Simulation Result for Case1

As shown in the Fig3 above, instruction sequence will copy data 427936 to register R3 and 432544 to R4. Third instruction will add the R3 and R4 and result will be stored in R5. Third instruction requires correct values of R3 and R4 to execute. Register values of R3 and R2 should be forwarded.

Case2:
MVI R3,427936
MVI R4,432544
LDBI R5, R3, 13223
ADD R6, R4, R5

As shown in Fig4 below instruction sequence will load the data from memory location (R3 + 13223 = 441159) to register R5. Data pointer (dpt) will put 441159 on data address bus and data memory will return the data 336250197. This data will be stored in register R5 and in next instruction it will be added with register R3 and result will be stored in register R5. This is an example of base index addressing mode.

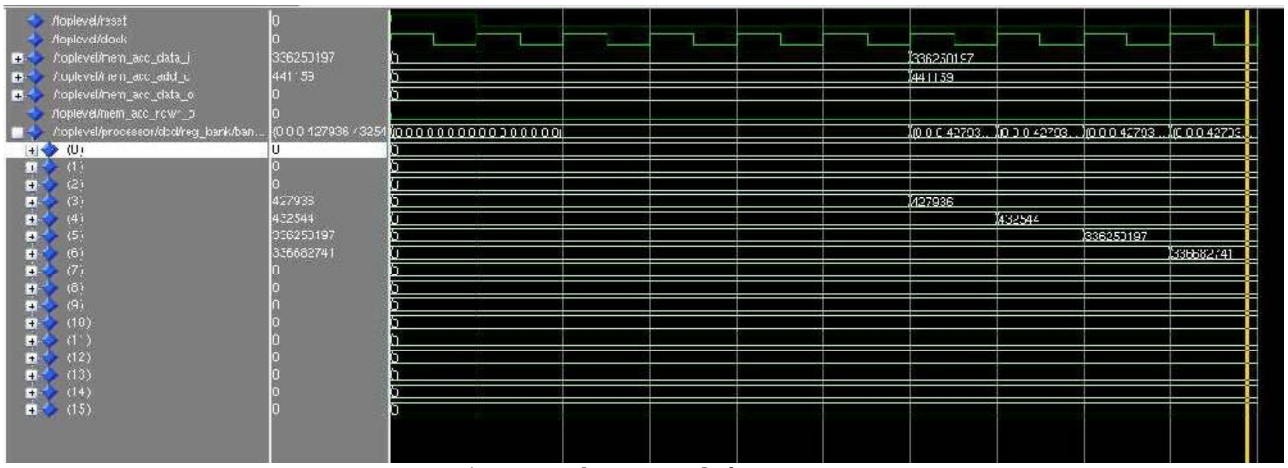

Fig.4: Simulation Result for Case2

Case3:

MVI R2,427040
MVI R3,427936
BNE R2, R3, 34647

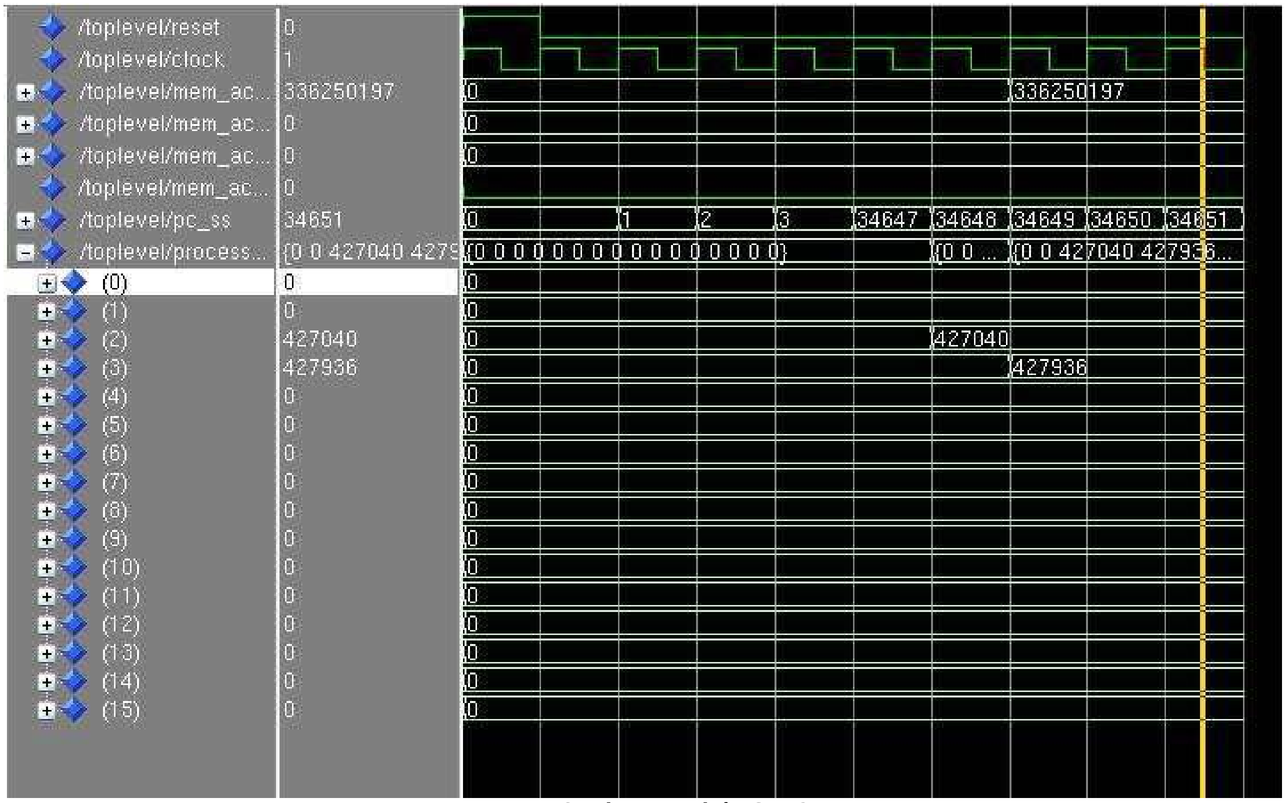

Fig.5: Simulation Result for Case-3

Above instruction sequence will compare the register value of R2 and R3. If R2 6= R3, then PC will be set to 34647. As shown in Fig5 R2 6= R3, so processor will put 34647 on address bus.

IV. Conclusion and Future Work

It is concluded that using data forwarding, described in this paper, RAW hazard can be reduced. All the instructions except branch and store, take 5 clock-cycles to execute, branch instructions take 3 clock-cycle to execute provided that branch is taken. Memory store instructions take 4 clock-cycles to store the data on memory without stalling the pipeline. Hence CPI of the processor can be reduced and the performance of the processor can be increased. Future work includes the implementation of ASIC of RISC processor with proposed pipeline.

References

[1] Liu Zhenyu; Qi Jiayue,"Implementation of precise exception in a 5-stage pipeline embedded processor," ASIC, 2003. Proceedings. 5th International Conference, vol no1, pp.447-451, Oct. 2003.
[2] J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, 2003.
[3] Islam, S.; Chattopadhyay, D.; Kumar Das, M.; Neelima, V.; Sarkar, S., "Design of High-Speed-Pipelined Execution Unit of 32-bit RISC Processor," India Conference, 2006 Annual IEEE , vol no 2 pp.1-5, Sept-2006.
[4] Haimin Chen; Bin Zheng, "A Design of 32-Bit Embedded Microprocessor," Communications and Intelligence Information Security (ICCIIS), 2010 International Conference , vol -4., pp.48-50, Oct. 2010.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

[5] P. Balaji; W. Mahmoud; E. Ososanya; K. Thangarajan "Survey of the counter flow pipeline processor architectures" , System Theory, 2002. Proceedings of the Thirty-Fourth Southeastern Symposium, pp.1-5, 2002.
[6] A. Shrivastava; E. Earlie; N. D. Dutt; A. Nicolau "Retargetable pipeline hazard detection for partially bypassed processors" IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume: 14, Issue: 8, Year: 2006.
[7] Aviral Shrivastava; Sanghyun Park; Eugene Earlie; Nikil D. Dutt; Alex Nicolau; Yunheung Paek " Automatic Design Space Exploration of Register Bypasses in Embedded Processors" IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume: 26, Issue: 12, Year: 2007.
[8] V. Zivojnovic; S. Pees; H. Meyr "LISA-machine description language and generic machine model for HW/SW co-design" International Conference on VLSI Signal Processing, IX, 1996, Year: 1996.
[9] Yu Qiao-yan; Liu Peng; Yao Qing-dong "A data hazard detection method for DSP with heavily compressed instruction set" Solid-State and Integrated Circuits Technology, 2004. Proceedings. 7th International Conference on, Volume: 3, Year: 2004.
[10] I. -J. Huang; A. M. Despain "Hardware/software resolution of pipeline hazards in pipeline synthesis of instruction set processors" International Conference on Computer-Aided Design, Year: 1993
[11] M. Zulkifli; Y. P. Yudhanto; N. A. Soetharyo; T. Adiono "Reduced stall MIPS architecture using pre-fetching accelerator" 2009 International Conference on Electrical Engineering and Informatics, Volume: 02, Year: 2009.
[12] M. Hatzimihail; G. Xenoulis; M. Psarakis; D. Gizopoulos; A. Paschalis "Software-based self-test for pipelined processors: a case study" 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05) Year: 2005

BIOGRAPHY

Ankur Changela is assistant professor in Electronics and Communication department of INDUS UNIVERSITY, Ahmedabad. He received his Master of Technology Degree (MTech) from National Institute of Technology Surathkal, Mangalore. Currently he is pursuing his PhD from IET, Ahmedabad University. His research interests are Low Power VLSI design and Processor Architecture.