A Method to Detect Hazards in Pipeline Processor

Yihui He¹, Han Wan¹*, Bo Jiang¹ and Xiaopeng Gao¹

¹School of Computer Science and Engineering, Beihang University, Beijing, China
{zy1506229, wanhan, jiangbo, gxp}@buaa.edu.cn

Abstract. In order to improve the throughput of the processors, pipeline technique is widely used to implement the instruction-level parallelism. However, this technique also leads to data hazards which has a great influence on the performance. This paper proposed a method called supply-matching to detect and solve data hazards efficiently. The logic of bypassing and stalling can be easily realized through this method. Furthermore, an RTL description of instructions was also introduced in this paper to reduce resource utilization. The case study was conducted through a five-stage microprocessor based on the PowerPC architecture with different approaches. Experiment results show our method requires less resources and achieves better performance.

1 Introduction

Pipeline technique is widely used in microprocessor design to improve performance. A pipeline processor can execute multiple instructions within a clock cycle. However, hazards arise if pipeline architecture is used, including data hazards, control hazards and structural hazards. Moreover, the problem becomes more complex when pipeline depth increases. Forwarding and stalling are effective solutions to resolve hazards for processors in embedded system. However, the complexity of detecting hazards increases rapidly as the number of instructions increases. In this case, many combination of instructions may lead to hazards in unanticipated way. Performing a highly effective method to resolve huge plenty of hazards oriented from deep pipeline and large instruction set is necessary.

Many studies have proposed some methods in pipeline processor design motivated by these situations mentioned above. Amit Pandey and Yu Qinayan used class-based method to detect data hazard [1, 2]. This method divides the whole instruction set into several parts so big problem are broken up into smaller ones. P. Bernardi and D. Boyang proposed a SBST algorithm [3]. Jiaying Lu designed a dynamic scheduling algorithm to improve the pipeline efficiency, which only increases one single-instruction buffer and some combination logic [4]. Also, Schönherr J, Schreiber I, and Fordran E proposed a method using symbolic model checking to detect hazards in pipelined processor [5].

In this work, we aim to find solutions to resolve huge plenty of hazards oriented from deep pipeline and large instruction set. Firstly, we briefly introduce the architecture, controllers and discuss our method of generating datapath according to the RTL description in Section 2. Next, a method called supply-matching is introduced to detect and resolve data hazards completely and efficiently in Section 3. Then we introduce control hazards and structural hazards in Section 4. Finally, we do an experiment to verify our method in Section 5.

2 Preliminary

2.1 Architecture

In this paper, processor adopts the architecture with five pipeline stages, as shown in Fig.1 (The italic words mean the name of the core units. Other words mean the name of interface and data).

Fig. 1. The architecture of a 5-stage-pipeline microprocessor

IF stage means fetching instructions from program memory. PC, Next PC (NPC) and Instruction Memory (IM) are assigned in IF stage. ID stage is assumed to decode the instruction and read or write register file. Also, the operation of expanding immediate is done in ID stage. EXE stage is used to execute arithmetic operations and logic operations. MEM stage is supposed to read or write memory. And WB stage means the execution result of the
instruction will be wrote back to register file. Four pipeline registers are distributed in the pipeline architecture to store data tentatively.

### 2.2 Notations and datapath

An RTL description of instructions was also introduced in this paper to reduce resource utilization. The rule is defined as follow: (1) A.B means port B of core unit A. (2) X.Y means pipeline register. For example, IF_ID means pipeline register between IF stage and ID stage. (3) B@X.Y means data B locked in pipeline register X.Y. (4) Z[z] means field z of unit Z. For example, Ins@IF_ID[Imm16] means 16-bits immediate operand of instruction stored in IF_ID. (5) A.B → C.D means data transfer from port B of unit A to port D of unit C.

It is more easily to structure datapath by using the RTL description. A datapath is a collection of functional units (such as ALU or multipliers), registers and buses. Follow the RTL rule, the data flow of each instruction is clear and all units have been linked. Then merging the data flow in the vertical direction to remove the repeated data flow of the whole instruction set. Adding MUX unit if the core unit has multiple inputs. The MUX control signal is defined as follow: (1) A.B means port B of core unit A. (2) X.Y means pipeline register between IF stage and ID stage. (3) B@X.Y means data B locked in pipeline register X.Y. (4) Z[z] means field z of unit Z. For example, Ins@IF_ID[Imm16] means 16-bits immediate operand of instruction stored in IF_ID. (5) A.B → C.D means data transfer from port B of unit A to port D of unit C.

### 3 Data hazard

#### 3.1 Problem definition

Data hazards are the hazards which are most frequently occurring in pipeline processor. Forwarding and stalling are effective solutions to resolve this problem. However, the complexity of detecting hazards increases rapidly as the number of instructions increases. In this situation, many combination of instructions may lead to hazards in unexpected way. It is imperative to take completeness detection of data hazards to ensure that all hazards combinations are considered.

#### 3.2 Solution

This paper proposed a method called supply-matching to detect and solve data hazards completely and efficiently. The method can be divided into two steps.

- Build $T_{use} - T_{new}$ matrix of all instructions for specified instruction set. Then all value of $T_{use}$ of registers and all value of $T_{new}$ for the processor can be got by synthesizing all records of $T_{use} - T_{new}$ matrix.
- Build register strategy matrix according to $T_{use} - T_{new}$ matrix. A $T_{use} - T_{new}$ record can be uniquely determined for specific register. According to the supply-matching model which is described in section 3.2.1, any data hazards can be detected and resolved by using formula (1).

#### 3.2.1 Supply-matching model

Data hazards occur when instructions that exhibit data dependence modify data in different stages of a pipeline. Therefore, data hazards detection can be transformed into the detection of relationship between data demand and data supply. In this case, provider is the pipeline register which saved the execution result of last instruction. For example, EXE_MEM pipeline register and MEM_WR pipeline register are the provider for all operation instructions. Demander is the components which need the most up-to-date value saved in provider at present. For example, ALU is the demander for all operation instructions.

Two basic principles are defined in this method.

Fig. 2. RTL description of five instructions
The stage of the instruction decoding must not be earlier than ID stage whether the centralized decoding mode or distributed decoding mode is adopted.

Forwarding technology has higher priority when both forwarding and stalling technology can be used to resolve data hazards. Besides, two parameters called $T_{use}$ and $T_{new}$ are defined.

- $T_{use}$ means the number of clock cycles that a certain functional unit will use the saved value in register after the instruction enters ID stage. $T_{use}$ is a static value and an instruction can have multiple $T_{use}$ according to the number of operands of the instruction. Meanwhile, $T_{use} \geq 0$.

- $T_{new}$ means the minimum number of clock cycles that the instructions which at stages after ID stage will produce the result that will be wrote back to registers. $T_{new}$ is a dynamic value. The value reduces by 1 as instruction flows through the pipeline stage and the value will no longer change once the value is 0. So an instruction has different $T_{new}$ at different stage. Meanwhile, $T_{new} \geq 0$. The management of $T_{new}$ and $T_{use}$ in pipeline processor is shown in Fig.3.

![Fig. 3. The management of $T_{new}$ and $T_{use}$ in pipeline processor](image)

Based on the above, solution of data hazards becomes digital. According to the definition of $T_{use}$ and $T_{new}$, each instruction has a set of digits which represent $T_{use}$ and $T_{new}$. For a specific instruction set, the comparison between $T_{use}$ and $T_{new}$ of all instructions of specific instruction set are taken into account. If the number of instructions increases, some additional work should be done to detect and resolve the data hazards for this part. Formula (1) is used to structure strategy matrix. The result is shown in Table 2.

$$\text{Operation} = \begin{cases} \text{Stall} & T_{use} < T_{new} \\ \text{Forward} & T_{use} \geq T_{new} \end{cases}$$

### 3.2.2 $T_{use}$-$T_{new}$ matrix

The $T_{use}$-$T_{new}$ matrix is used to find all the value of $T_{use}$ and $T_{new}$ of all instructions. The $T_{use}$-$T_{new}$ record of an instruction is certain depending on the definition of $T_{use}$ and $T_{new}$ if the pipeline architecture stays the same. Therefore, a $T_{use}$-$T_{new}$ matrix is certain too if all instructions of specific instruction set are taken into account. In this situation, all the work is focusing on the execution semantics of each instruction and completing the matrix line by line, rather than concentrating on the specific classification of the instructions. This method greatly reduces the complexity of hazards detection even if the number of instructions increases. For example, supposing the instruction set includes instructions of ADD, SUB, ANDI, ORI, LW, SW and BEQ. Concerning the five-stage pipeline, the $T_{use}$-$T_{new}$ matrix is shown in Table 1.

| Target | $T_{use}$ 1 | $T_{use}$ 2 | $T_{new}$ |
|--------|-------------|-------------|-----------|
| Ins    | RS | RT | EXE | MEM | WB         |
| ADD    | 1  | 1  | 1   | 0   | 0          |
| SUB    | 1  | 1  | 1   | 0   | 0          |
| ANDI   | 1  | Null | 1   | 0   | 0          |
| ORI    | 1  | Null | 1   | 0   | 0          |
| LW     | 1  | Null | 2   | 1   | 0          |
| SW     | 1  | 2   | Null | Null | Null       |
| BEQ    | 0  | 0   | Null | Null | Null       |

Using the supply-matching model can easily build $T_{use}$-$T_{new}$ matrix. The procedure also applies to processor which has deeper pipeline stage or larger instruction set.

#### 3.2.3 Register strategy matrix

Register strategy matrix provides the resolution strategy of data hazards for specific register. Based on the $T_{use}$-$T_{new}$ matrix, a complete strategy matrix can be built for specific register as the $T_{use}$-$T_{new}$ matrix considers all instructions and all pipeline stages except the stage before ID stage (Basic principle 2 makes the rule. If the number of stages which before ID stage more than one, some additional work should be done to detect and resolve the data hazards for this part). Formula (1) is used to structure strategy matrix. Taking the RT register as example, the result is shown in Table 2.

| $T_{new}$ | EXE | MEM | WB |   |
|-----------|-----|-----|----|---|
| 0         | Stall | Stall | Bypass | Stall | Bypass |
| 1         | Bypass | Stall | Bypass | Bypass | Bypass |
| 2         | Bypass | Bypass | Bypass | Bypass | Bypass |

This strategy matrix is correct and can be proofed. We can prove it from follow aspects:

1. According to the definition of $T_{use}$ and $T_{new}$, $T_{use} \geq 0$, $T_{new} \geq 0$. So there just three relationship between $T_{use}$ and $T_{new}$, which are $T_{use} > T_{new}$, $T_{use} < T_{new}$ and $T_{use} = T_{new}$.
2. If $T_{use} > T_{new}$, it indicates that the instruction before has already finished computing the write-back data which the current instruction needs. So bypassing can be established correctly.
3. If $T_{use} < T_{new}$, it means that current instruction cannot get the dependency data from the subsequent stages. Thus stalling is used to ensure processor performs correctly.
4. If $T_{use} = T_{new}$, it shows that current instruction can get related data immediately as the instruction before finish computing at the same time. Forwarding technology can be used here.

A complete stall or control signal can be created by using logic operation OR to integrate all the stalling conditions. In other cases, data hazards can be resolved by forwarding. However, there may have many forwarding sources represented data from multiple pipeline stages when using forwarding technology. In this situation, the priority of forwarding source should be set. This paper adopts a forwarding strategy based on the pipeline priority. The strategy sets the stage which $T_{use}$ stages behind ID stage has the highest priority among the whole pipeline stages and the priority of other stages behind it are decreasing in turn. Based above, just selecting forwarding source which has the highest priority when there are multiple forwarding sources.

4 Control hazard and structural hazard

Control hazards (branch hazards) cause by branch instructions. There are two main techniques to resolve branch hazards, including branch prediction and branch delay slot. Other studies have already done this part efficiently. Please refer to [6, 7, 8] for details.

Structural hazards occur when a part of the processor’s hardware is needed by two or more instructions at the same time. The strategy for resolving this hazards is simple. Just stalling the pipeline or copying the basic unit.

5 Experiment

5.1 Framework

The methods mentioned above were adopted to implement a five-stage PowerPC microprocessor which supports 72 instructions. It is significant to realize the whole microprocessor rather than the logic of hazards detection alone because microprocessor cannot work normally if the system only supports the logic of hazards detection.

There are two decoding modes which are centralized decoding and distributed decoding in processor design. In centralized decoding mode, controllers are assigned at ID stage. Then control signals created by controllers transfer through the pipeline. But in distributed decoding mode, controllers are distributed in multiple stages and controllers only create the control signal which related to the core units in the same stage [9, 10]. In this paper, the supply-matching method uses a hybrid decoding mode which means $T_{use}$ uses centralized decoding mode and $T_{new}$ uses distributed decoding mode. However, in order to set up a control experiment, we also implemented the supply-matching method only using centralized decoding mode.

The framework for the experiment as shown in Fig. 4. Firstly, instruction set, core units and pipeline architecture were determined according to architecture specification. After that, core unit modules, controllers, datapath, and multiplexers were implemented with Verilog HDL. Meanwhile, hazards in pipeline are resolved based on our method. Finally, binary codes were got from compiling the programs written in C or assembly by GCC. The result of registers after executing every instruction is obtained from QEMU in single-step mode. The correctness of the processor can be determined by comparing the value of registers with the result from QEMU during the simulation. Spartan6-6SLX150FGG484 made by Xilinx is the FPGA used to synthesize and implement with ISE in our experiment.

5.2 Result and analysis

By comparing the value of registers with the result from QEMU, the correctness of the processor has been proved. This also indicates the correctness of the supply-matching method. Meanwhile, we have compared the synthesize reports of two microprocessors which implemented the supply-matching method using different decoding modes in three aspects: clock frequency (MHz), the number of Flip-Flops (FF), the number of BELs (which includes all basic logic primitives like LUT, MUXCY, etc). The result is shown in Table 3.

Table 3. Comparisons of method implementation using different decoding mode

| Parameter Mode             | Clock | FF  | BEL  |
|----------------------------|-------|-----|------|
| Using centralized decoding mode | 8.46  | 3525| 12844|
| Using hybrid decoding mode  | 59.46 | 3481| 10380|

It is obvious to conclude that our method which using hybrid decoding mode can gain faster clock frequency with less resources.

6 Conclusions

In this paper, an efficient method called supply-matching to completely detect and resolve data hazards has been proposed. The logic of bypassing and stalling can be easily integrated and implemented due to this method. Finally, we conducted extensive experiments based on a state-of-art microprocessor, PowerPC architecture with five stage pipeline. Experiment results proof that our method can achieve faster clock frequency with less resource than the well-known method called stage-decoding.
Acknowledgment

This work has been supported by project of the Professional Group Construction of Beijing Educational Committee.

References

1. Pandey A. Study of data hazard and control hazard resolution techniques in a simulated five stage pipelined RISC processor. *International Conference on Inventive Computation Technologies*. IEEE, 1-4 (2017).

2. Qiao-Yan Y, Peng L, Qing-Dong Y. A data hazard detection method for DSP with heavily compressed instruction set. (2004).

3. Bernardi P, Boyang D, Ciganda L, et al. A functional test algorithm for the register forwarding and pipeline interlocking unit in pipelined microprocessors. *Design and Test Symposium. IEEE*, 1-6 (2014).

4. Lu J, Zhou X, Wang J. A novel dynamic scheduling algorithm of data hazard for embedded processor. *International Conference on Asic. IEEE*, 28-31 (2007).

5. Schönherr J, Schreiber I, Fordran E, et al. Hazard Checking in Pipelined Processor Designs Using Symbolic Model Checking. *Euromicro Conference, 1999. Proceedings. IEEE*, 1, 75-78 (1999).

6. E. Nurvitadhi, J. C. Hoe, T. Kam, and S. Lu. Automatic pipelining from transactional datapath specifications. In *Design, Automation and Test in Europe, DATE 2010, Dresden, Germany, March 8-12, 2010*, 1001-1004 (2010).

7. E. Nurvitadhi. Automatic pipeline synthesis and formal verification from transactional datapath specifications. *Terapevicheski Arkhiv*, 80(4), 73-6 (2008).

8. E. Nurvitadhi, J. C. Hoe, T. Kam, and S. Lu. Automatic pipelining from transactional datapath specifications. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 30(3), 441-454 (2011).

9. P. Yiannacouras, J. G. Steffan, and J. Rose. Exploration and customization of fpga-based soft processors. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 266–277 (2007).

10. Yiannacouras, Peter, Jonathan Rose, and J. Gregory Steffan. The microarchitecture of FPGA-based soft processors. *Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, 202-212 (2005).