Compiler assisted Data Forwarding in VLIW/EPIC architectures

Natarajan Kannan, Palanidaran Chidambaram, Suriya Narayanan M Subramanian, and Ranjani Parthasarathi

School of Computer Science and Engineering
Anna University
Chennai - 25
{natarajan,palanidaran,mssnlayam,rp}@cs.annauniv.edu

Abstract. This paper proposes a mechanism for reducing the complexity of forwarding hardware in VLIW/EPIC processors. The necessary information for data forwarding is known at compile time. This paper proposes a way to incorporate the forwarding information along with the instruction itself, thereby reducing the hardware complexity of forwarding logic with implications for power saving and reducing chip area.

1 Introduction

EPIC (Explicitly Parallel Instruction Computing) is a design philosophy that tries to reduce hardware complexity by providing explicit information about parallelism. EPIC style of architecture is an evolution of VLIW. Compilers conforming to EPIC philosophy craft a static schedule which is honoured by hardware. Information conveyed by an EPIC compiler include branch hints, cache specifiers, speculative memory operations, prioritized memory references, etc. [1]. This paper proposes an addition to the list of compiler controlled architectural features namely forwarding. Decision regarding forwarding of data between different pipeline stages is taken at run time, by comparing the destination register specifier of instructions that have completed execution but have not yet written their result to the register file, with the source register specifiers of instructions that are yet to be executed. A significant portion of all operand accesses is made via bypasses or forwarding. Therefore forwarding is indispensable in any architecture. This paper proposes a scheme to simplify the forwarding hardware, by generating signals required for forwarding, at compile time. The proposed scheme can be used in VLIW and EPIC based processors.

This paper is organized as follows: In section 2, work related to the present approach and the motivation for this paper are presented. The existing forwarding architecture is presented in section 3. In section 4, the compiler assisted data forwarding model is proposed. The benefits of the proposed model are analyzed in section 5. Section 6 concludes the paper.

2 Related Work and Motivation

Development of efficient forwarding logic for VLIW processors is a well researched area. IBM's VLIW [2] has an elaborate bypassing network. Philips' LIFE processor

[2] uses funnel files to pass the result of one functional unit to another. These processors have expensive hardware units, to offset which, clusters are used. In such an architecture, the output of one processing element is linked to the inputs of other processing elements in a cluster. Accesses crossing clusters are performed via a register file and may need an additional cycle. The above schemes for bypassing and forwarding are rather complex.

Abnous et al [3] have suggested removing of the standard memory pipeline stage of a RISC processor in order to reduce the address comparison circuitry of the forwarding hardware. This is achieved by replacing the displacement addressing mode with the register indirect addressing mode, and can have a negative effect on performance.

In recent years, there has been a paradigm shift in exploiting ILP, with the compiler assuming an important role of exposing ILP explicitly. For instance, Sami et al [5] propose a way to use compile time information to reduce register file accesses. The compiler encodes information about short-lived registers to avoid register file accesses, reducing the power consumption of the register file. Inter-stage registers are used to store these variables.

This paper proposes the use of compile time information to reduce complexity of forwarding logic. In this scheme, the compiler adds extra bits to each instruction to convey the forwarding information.

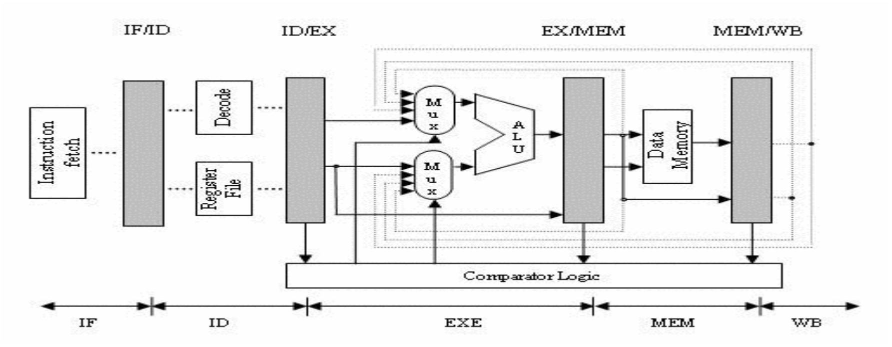

Fig. 1. Data Forwarding Hardware

3 Existing Forwarding Mechanism

The existing forwarding mechanism is easily explained with a generic VLIW architecture with five pipeline stages [2], [6]; Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory (MEM) and Write Back (WB) as shown in figure 1. In such an architecture, data forwarding can be done from the EX/MEM pipeline register and the MEM/WB pipeline register to the execution unit. Instructions like `add`, `sub`, `mul`, etc. can forward their results after their EX stage, i.e. either from the EX/MEM pipeline register or from the MEM/WB pipeline register. Instructions such as `load` can forward the their result only from the MEM/WB pipeline register after completing their MEM stage.

In order to forward, comparisons are made between the EX/MEM.`destination_register`, and ID/EX.`source_register_1` and ID/EX.`source_register_2`. Similarly the MEM/WB.`destination_register` is compared with ID/EX.`source_register_1` and ID/EX.`source_register_2`. These comparisons select the outputs of the multiplexers, which act as source operands to the execution unit.

When only one functional unit is present, the above comparisons are sufficient. The number of comparisons increases quadratically with increase in number of functional units [3].

4 Proposed Model

The model proposed here is aimed at removing the comparators that are required to make forwarding decisions. Since the information about dependences between instructions is known at compile time, decision regarding forwarding of data between dependent instructions can be determined at compile time itself. By conveying this information along with the instruction, the comparators required to make forwarding decisions can be eliminated, and therefore the forwarding logic is simplified. The details of such a design are given below.

The case of forwarding within a single functional unit with the pipeline stages as shown in figure 1 is considered here. The following bits are added as part of each instruction:

Do Forward Bit (DF): A bit to indicate if this instruction is to forward its result.

When to forward (W): These bits indicate the stage after which forwarding is to be done. In the above case, 1 bit is required to indicate whether forwarding is to be done to the execution unit from the MEM/WB pipeline register or the EX/MEM pipeline register.

Source Operand (S): This bit is used to indicate to which of the two operands of the execution unit the result is to be forwarded to.

The use of these bits is best illustrated using a sample code sequence as shown below:

`add r1,r2,r3` – in n^{th} VLIW instruction
`sub r4,r5,r6` – in $(n+1)^{th}$ VLIW instruction
`mul r3,r6,r7` – in $(n+2)^{th}$ VLIW instruction

The instructions are of the format: `operation src1, src2, dest`.

The `mul` instruction requires the results from the `add` and the `sub` instruction. The results of the `add` instruction and the `sub` instruction are available after their respective EX stages. When the `mul` instruction is to enter the EX stage, the result of the `add` instruction is available in the MEM/WB pipeline register,

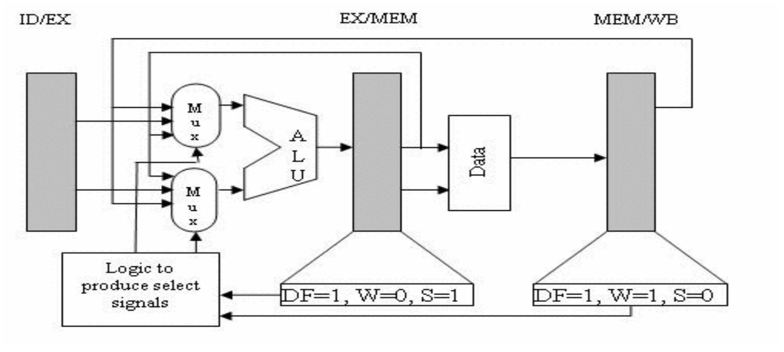

Fig. 2. Data Forwarding Hardware with Compiler assistance

and the result of the `sub` instruction is available in the EX/MEM pipeline register. Therefore these results are to be forwarded to the `mul` instruction. This forwarding information as added by the compiler is shown in figure 2.

`add`.DF	= 1	`add` must forward its result
`sub`.DF	= 1	`sub` must forward its result
`add`.W	= 1	`add`'s result must be forwarded from MEM/WB
`sub`.W	= 0	`sub`'s result must be forwarded from EX/MEM
`add`.S	= 0	`add`'s result must be forwarded to the 1^{st} operand of the `mul` instruction
`sub`.S	= 1	`sub`'s result must be forwarded to the 2^{nd} operand of the `mul` instruction

The control signals needed to select the appropriate input of the multiplexers are generated from the DF, W and S bits. This logic is much simpler than the comparator logic present in the conventional forwarding hardware.

When there is more than one functional unit each functional unit is given an ID. This ID is used to specify the destination instruction's functional unit.

For an architecture with one functional unit and the above pipeline scheme, three bits per instruction are required to specify the forwarding information. When a number of functional units are present, the size of the instruction increases by the number of bits required for specifying the functional unit ID. The addition of these bits incurs an expense of increased instruction size. The ISA must be modified to accommodate this. Typical VLIW architectures have unutilized bits which can be used for this purpose.

5 Benefits

In a VLIW processor with n functional units and d pipeline stages between ID and WB stages, $2dn^2$ comparators are required [3]. These comparators present a costly area penalty which can be removed by the proposed scheme.

By removing the comparators of the forwarding logic using the proposed scheme, the complexity of the forwarding logic is greatly reduced. The chip area required for the proposed forwarding scheme is also reduced. Due to the reduced chip area and reduced complexity of this scheme, the power consumption will be much lower than that of the conventional forwarding logic. The effect of longer instructions on power consumption is to be examined.

6 Conclusion

In this paper, a compiler assisted technique for data forwarding in VLIW/EPIC processors is discussed. With the VLIW/EPIC philosophy emphasizing the need for reducing hardware complexity, this method attempts to do the same by conveying the forwarding information explicitly at compile time. The proposed scheme reduces the chip area and lowers the power consumption of the forwarding hardware.

References

1. Micheal S. Schlansker, B. Ramakrishna Rau: "EPIC: An Architecture for Instruction-Level Parallel Processors," HP Laboratories Palo Alto, HPL-1999-111, February 2000.
2. "Exploitation of Fine-Grain Parallelism," Lecture Notes in Computer Science, Volume 942, Springer Verlag.
3. A. Abnous and N. Bagherzadeh: "Architectural Design and analysis of a VLIW Processor," M.S. Thesis, University of California, Irvine, 1991.
4. A. Abnous and N. Bagherzadeh: "Pipelining and Bypassing in a VLIW Processor," IEEE Trans. on Parallel and Distributed Systems, Vol. 5, No. 6, June 1994, pp. 658-663.
5. M. Sami, D. Sciuto, C. Silvano, et al.: "Exploiting data forwarding to reduce the power budget of VLIW embedded processors," Processor Design, Automation and Test in Europe, pp. 252-257 2001.
6. J. Hennessy and D. A. Patterson: "Computer Architecture: A Quantitative Approach," Morgan Kaufmann Publishers, San Mateo, CA, Second Edition, 1996.