Artificial Life and Robotics (2020) 25:59–63
https://doi.org/10.1007/s10015-019-00554-w

ORIGINAL ARTICLE

A design of EPIC type processor based on MIPS architecture

Takahito Hayashi[1] · **Akinori Kanasugi**[1]

Received: 23 April 2019 / Accepted: 23 August 2019 / Published online: 23 September 2019

Abstract
This paper proposes an EPIC (Explicitly Parallel Instruction Computing Architecture) type processor based on MIPS. VLIW processors can execute multiple instructions simultaneously, but due to dependency of instructions, it is often impossible to execute maximum parallel execution. As a result, program contains many NOP instructions. EPIC processor can reduce NOP instructions by changing number of instructions to be executed simultaneously. To implement EPIC type processor, five bit field is embedded in the machine instruction code. For comparison, a 5-stage pipeline processor (basic processor), and a Very Long Instruction Word (VLIW) processor are designed. The proposed processors are described in hardware description language (VHDL) and implemented using FPGA. Operations are confirmed by software Tera Term. Processors are evaluated for instruction parallelism and program size using bubble sort program. It is confirmed that the proposed processor is 1.9 times faster than the basic processor. In addition, the program size of the proposed processor is 64 bytes, the basic processor is 56 bytes, and the VLIW processor is 80 bytes.

Keywords Processor · EPIC · VLIW · MIPS · FPGA

1 Introduction

Many processors are required to control the robot. Figure 1 shows the robot arm as a basic component of the robot. There is a close interaction between the arms, and real-time control must be performed using multiple processors. To improve the performance of the processor, efficient operation of computing units has been required. One solution is parallel operation. Super-scalar and out-of-order are effective for this operation. However, the hardware had become complicated.

On the other hand, Very Long Instruction Word Architecture (VLIW), which left these operations to compilers has attracted attention. In this paper, we propose an Explicitly Parallel Instruction Computing Architecture (EPIC) type processor which improved VLIW. The proposed processor is based on the open-source MIPS architecture.

The processors are designed using hardware description language (VHDL) and implemented using FPGA. A bubble sort program is operated on these processors and confirms the operation and performance.

2 Basic processor

A basic processor in this paper has a five-stage pipeline. Compared with the pipeline processor and single cycle processor, the pipeline processor can reduce the clock period. Therefore, it can operate at faster frequency [1]. Since the merit of speeding up is large, though the time to complete one instruction becomes long, pipeline processing is being carried out by many processors in practical use [1]. Figure 2 shows a block diagram of the designed basic processor.

The basic processor has a simple five-stage pipeline architecture. The stages are (1) instruction fetch (IF), (2) decode (ID), (3) execution (EX), (4) memory (MEM), and (5) write back (WB).

Asynchronous read-only memory (ROM) is used for instruction memory; clock synchronous write and asynchronous read memory (RAM) are used for data memory. The

This work was presented in part at the 24th International Symposium on Artificial Life and Robotics (Beppu, Oita, January 23–25, 2019).

✉ Takahito Hayashi
19kmh15@ms.dendai.ac.jp

Akinori Kanasugi
kanasugi@mail.dendai.ac.jp

[1] Tokyo Denki University, Tokyo, Japan

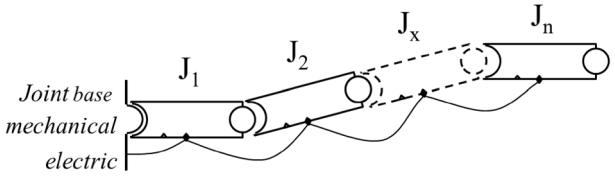

Fig. 1 An arm composed of joints

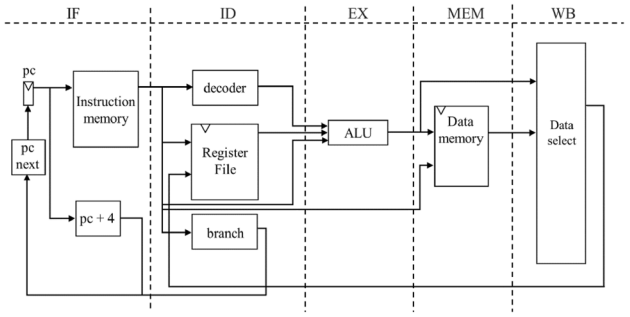

Fig. 2 Block diagram of basic processor

memory size is 4096 × 32 bit. Branch instructions are determined in the ID stage by dedicated comparators. This has avoided performance degradation by reducing the number of instructions flushed when a branch occurs. However, since processing at the ID stage is increased, there is a possibility that the operation speed is lowered.

The basic processor has 24 instructions of integer arithmetic instructions, branch instructions and load/store instructions from MIPS 32-bit instructions. The basic processor has two more functions to prevent performance degradation. If the instruction refers to the result of WB or MEM, the data to be referenced are transferred to ID stage or EX stage (forwarding). However, if the calculation result has not been output yet, or if data cannot be read from memory yet, processor might output incorrect results by referring to incorrect values. Therefore, a stall function to stop the pipeline is implemented.

3 VLIW processor

The VLIW processor can execute multiple instructions [2–7]. In this paper, the designed VLIW processor can process four instructions by combining four instructions from the integer operation instructions and two instructions from the load/store instructions. This is because it is said that a processor can process up to four processes simultaneously in a program [2]. The designed VLIW processor has 128-bit instructions that combined four MIPS 32-bit instructions.

As shown in Table 1, integer operation instructions and load/store instructions are frequently used (83% in total).

Table 1 Frequency of use of MIPS integer instructions of gap compiler [2]

Instruction	Frequency of use (%)
load	26.5
add	21.1
store	10.3
cond branch	9.3
or	7.9
load imm	4.8
and	4.3
shift	3.8
compare	2.8
xor	1.8
sub	1.7
call	1.6
return	1.6
mul	1.4
jump	0.8
cond move	0.4

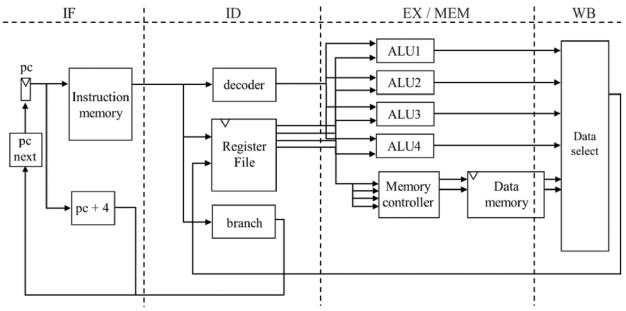

Fig. 3 Block diagram of VLIW processor

Therefore, the VLIW processor can improve performance by simultaneously processing integer arithmetic instructions and load/store instructions. Figure 3 shows the block diagram of the VLIW processor.

Since the memory controller is added to the designed VLIW processor, integer operation instructions and load/store instructions can be executed simultaneously. The address calculation using the dedicated adder of the memory controller is faster than using the ALU. Therefore, even if the EX stage and the MEM stage are integrated, the operating frequency will not be greatly reduced. It is also expected to reduce the number of clocks required for processing one instruction (improving latency). Then the number of pipeline stages has been reduced to four stages, comparing with the basic processor.

4 Proposed processor

VLIW [2, 8–10] has the advantage of simple hardware and low power consumption compared to super-scalar. However, if the instruction's width is changed, VLIW processor requires to update the compiler. Therefore, there is no compatibility in machine language program. Furthermore, if there is no valid instruction, compiler have to insert No operation (NOP), which leads to an increase in instruction size. In previous work, the VLIW architecture proposed in [11] and [12] uses stop bits to maintain instruction compression and compiler compatibility. Using this stop bit, cache hit rate is improved. Therefore, stop bit is an effective technique.

The proposed processor was designed based on EPIC architecture and MIPS instruction set proposed in [13]. The proposed processor defined bits which indicate the parallelism of machine language instructions, as in previous researches. In addition to this, the proposed processor defines operation bits for branch instructions and immediate extension instructions. These bits define not only instruction code compression but also branch prediction and multi-core support. To implement EPIC processor, we defined the new five bit function field in instruction code. To implement this defined function, we modified the instruction format and reduced five bit from the MIPS 32b instruction. Figure 4 shows the EPIC instruction format defined in this paper.

The five bit field is inserted for each instruction format as follows. (1) R-type shares the shamt field used for the shift command and rs field specifying a register. This is because the rs field is not used when the shamt field is used for shift instruction. (2) In case of I-type and J-type, immediate field (imm and addr field) is shortened by five bit and then new five bit function field is inserted.

Though the value represented in the immediate field has become smaller, it is not a severe problem in embedded systems with limited memory space.

To realize EPIC processor, each bit in function code indicates the following conditions:

1. "Parallel bit" indicates parallelism. Using this bit, the same operation as the NOP instruction can be executed.
2. "Immediate bit" indicates immediate value extension. It is possible to treat the following instruction sequence as an immediate value.
3. "Branch bit" decides whether to flush if branching occurs. It is referred when deciding whether to execute flush when a branch instruction occurred.
4. "Processor ID bit (2 bit)" is reserved for the corresponding multi-core processor.

These bits are considered for maximum compatibility.

Compared with usual VLIW, the proposed processor can considerably reduce NOP instructions by expressing NOP with only one bit. The prediction of branch instruction is entrusted to compiler. We also prepared expansion to multi-core.

However, to realize EPIC, since the immediate value has been deleted by five bit, if immediate value is large, 64-bit instruction is necessary. Therefore, the parallelism of instructions decreases. The usual VLIW processor has an advantage if branches exceeding 21 bit or immediate values beyond 11 bit are frequently used. A block diagram of the proposed processor is shown in Fig. 5.

The unit to confirm the function in machine language has added to the ID stage of the proposed processor. The proposed processor uses ROM as instruction memory and RAM as data memory. The ROM can specify two addresses concurrently.

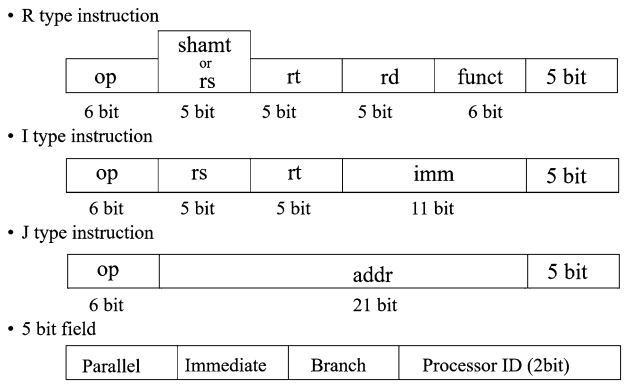

Fig. 4 EPIC instruction format

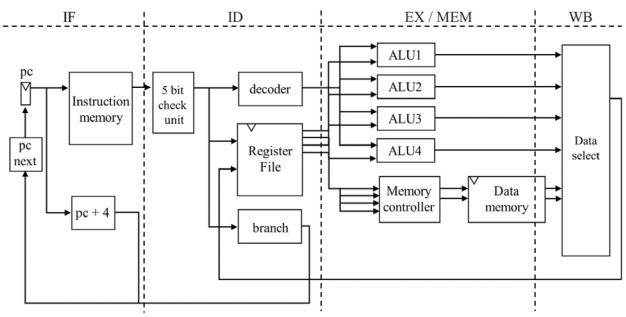

Fig. 5 Block diagram of the proposed processor

Fig. 6 Experimental environment with FPGA and Tera Term

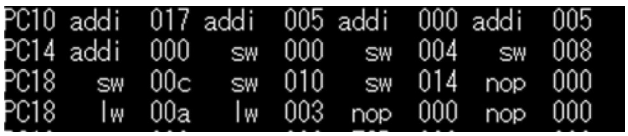

Fig. 7 Operation confirmation using Tera Term

Table 2 Major circuit elements of FPGA used by processor

Processor	LUT	FF
Basic processor	864	375
Proposed processor	7357	1851

5 Verification

The processors have been designed using VHDL. They were properly synthesized by the integrated circuit design tool Vivado (2018.1) of Xilinx Inc. The target FPGA and the evaluation board are Artix-7 of Xilinx Inc. and NEXYS 4 DDR of Digilent Inc., respectively. A terminal emulator program "Tera Term" is also used to confirm operation. Figures 6 and 7 show the experimental environment and the operation confirmation by Tera Term, respectively.

To confirm the operation of the processor, we have designed a clock count circuit and an instruction disassembly circuit. These data are displayed via serial communication of FPGA and Tera Term with UART.

To evaluate "parallel bit" of the proposed processor, we have prepared a simple bubble sort program, and compared the number of clocks from program start to finish with the program size. The number of clocks was confirmed using Tera Term.

Table 3 Number of clocks required to complete bubble sort

Processor	Number of clock number
Basic processor	79.8×10^6
Proposed processor	42.0×10^6

Table 4 Program size of bubble sort

Processor	Program size (byte)
Basic processor	56
VLIW processor	80
Proposed processor	64

6 Experimental result

The circuit size of the proposed processor is shown in Table 2. Since the FPGA is basically built with LUT and FF, the circuit scale can be compared by comparing these values. Furthermore, we have confirmed that the proposed processor can be synthesized correctly at the maximum operating frequency of 70 MHz.

We have used 4096 pseudorandom numbers as sample data for sorting. We have evaluated instruction parallelism by comparing the required clock number for execution. Table 3 shows the clock number of the pipeline processor and the proposed processor. The results show that the proposed processor is 1.9 times faster than the basic pipeline processor.

Table 4 compares the bubble sort program size for the basic processor, the VLIW processor, and the proposed processor. Though program size of the proposed processor increased by 8 bytes than the basic processor, it reduced NOP instructions by 16 bytes than the VLIW processor.

7 Conclusion

This paper proposed one of EPIC type processor based on MIPS architecture. In addition to the bits that compress the instruction codes, the proposed processor has branch prediction, immediate expansion, and multi-core capabilities. The instruction level parallelism can be further extracted by processing these bits effectively.

The proposed processor is 1.9 times faster than the basic pipeline processor. In addition, although program size of the proposed processor increased by 8 byte than the

basic processor, it reduced NOP instructions by 16 byte than the VLIW processor. In this paper, we have evaluated the effectiveness of "parallel bit" of the proposed processor. In the future, we would like to evaluate the effect of other bits of the proposed processor.

References

1. Harris DM, Harris SL (2009) Digital design and computer architecture. Shoeisha, Tokyo **(In Japan: Translated by Amano Hideharu, Suzuki Mitugu, Nakajo Hironori, et al.)**
2. Hennessy JL, Patterson DA (2014) Computer architecture: a quantitative approach 5/E. Morgan Kaufmann, USA
3. Chu WWS, Dimond RG, Perrott S et al. (2004) Customisable EPIC Processor: Architecture and Tools. Proceedings Design, Automation and Test in Europe Conference and Exhibition, Paris, France, Vol. 3, pp 236–241
4. Nakada T, Kataoka A, Nakashima Y (2009) An instruction scheduling method with VLIW instruction queue for superscalar processors. Inf Process Soc Jpn 2(2):48–62 **(in Japanese)**
5. Tyson, Romas AL, SI P, et al. (2009) A pipelined double-issue MIPS based processor architecture. International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), Kanazawa, Japan, Dec 7–9, 2009, pp 583–586
6. Santos R, Azevedo R, Araujo G (2006) 2D-VLIW: an architecture based on the geometry of computation. IEEE 17th International Conference on Application-specific Systems, Architectures and Processors (ASAP), steamboat Springs, CO, USA, 11–13 Sept, 2006, pp 87–94
7. Darsch A, Andre´ Seznec (2004) IATO: a flexible EPIC simulation environment. Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing, Foz do Iguacu, PR, Brazil, 27–29 Oct. 2004, pp 58–65
8. Kobayashi Y, Kobayashi S, Sakanushi K et al (2004) HDL generation method for configurable VLIW processor. J Inf Process (JIP) 45(5):1311–1320 **(in Japanese)**
9. Komatsu H, Koseki A, Suzuki E et al (1993) Parallel executing mechanisms in the extended VLIW processor GIFT. J Inf Process (JIP) 34(12):2599–2611 **(in Japanese)**
10. Colwell RP, Nix RP, O'Donnell JJ et al (1988) A VLIW architecture for a trace scheduling compiler. IEEE Trans Comput 37(8):967–979
11. Brandon A, Hoozemans J, Straten J et al. (2015) A sparse VLIW instruction encoding scheme compatible with generic binaries, 2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Mexico City, Mexico, 7–9 Dec 2015
12. Hübener B, Sievers G, Jungeblut T (2014) CoreVA: a configurable resource-efficient VLIW processor architecture, 2014 12th IEEE International Conference on Embedded and Ubiquitous Computing, Milano, Italy, 26–28 Aug 2014, pp 9–16
13. Sharangpani H, Arora K (2000) Itanium processor microarchitecture. IEEE Micro 20(5):24–43