A Single-Stream Pipelined Instruction Decompression System for Embedded Microprocessors

[1]Yuan-Long Jeang , [2]Tzuu-Shaang Wey, [3]Hung-Yu Wang , and [3]Chih-Chung Tai

[1]Department of Information Engineering,
[2]Department of Electronic Engineering,
Kun Shan University
[3]Department of Electronic Engineering,
National Kaohsiung University of Applied Sciences
yljeang@ms22.hinet.net

Abstract

For instruction decompression, techniques such as single buffering, double buffering and pipelining have been proposed. However, due to jumping penalty, these techniques incur more delays in pipeline or system has to be stopped to refill the cache buffers. A Pipeline with Back-up for Flushing (PBF) technique has been developed that incurs no delay and without stopping due to jumping. However, the first instruction of each basic block should not be compressed and be put in another ROM, and thus the compression ratio should be sacrificed. This paper improves the PBF technique such that a single program ROM needed only. The simulation results for several benchmarks show that the average compression ratio is decreased about 11%, and the hardware cost deceased about 8%.

1. Introduction

Cost, speed and power consumptions are three important issues for system on a chip. The area of the program memory usually occupies 40%-80% of the whole chip. Instruction compression can make the program memory needed be greatly reduced. Therefore, reducing program memory capacity to decrease cost and power consumption is a popular research. Researches about code (or instruction) compression can be divided into two categories mainly: modifying instruction set by processor core manufactures (such as the Thumb for ARM series or MIP16e for MIPS series) or using algorithms of data compression (such as the Huffman or the Arithmetic Coding).

In general, the programs of RISC processors are not so short and efficient when comparing with those of CISC processors. However, there is a very high degree of repetitions of instructions in a program of RISC processors. So, according to these characteristics, we propose a new method to compress instruction codes; hence it could reduce program memory size, die size and cost. However, code compression is performed before the chip design while the code decompression is performed when the chip is running. Therefore, the efficiency of the decompression affects not only the compression ratio but also the execution time and cost.

2. Related Work

Wolfe and Chanin [1] is the first to present the code compression and decompression schemes for embedded processor in 1992. The system is called CCRP. The system builds each embedded processor with an instruction cache (single buffering), and all instructions are fetched through the instruction cache. Their decompression circuit is inserted between cache and main memory. They use a byte-based Huffman coding method to compress program code and use Line Address Table (LAT) to map the compressed block address for solving the branch issues. Their average compression ratio is 73% for MIPS code. The compression ratio is defined as the ratio of compressed program size over original program size.

Lefurgy et al. [2] uses the dictionary-based code compression technique. To solve branch issues, they don't compress the branch instruction, and modify the branch offset to map the compressed program. Average compression ratios are 61%, 66% and 74% for the Power PC, ARM and i386 processors, respectively.

Haris ed. al. [3] proposed pipelined decompression architecture. They classify all instructions into 4 categories. Each category has its own decompression pipe. The longest pipe has 6 stages including the 3 stages inside the CPU. As a result, the performance of system can increase by up to 46%.

A double buffering technique has been proposed by our previous work in [8]. However, when jumping occurs, it still takes much time to refill the buffers full.

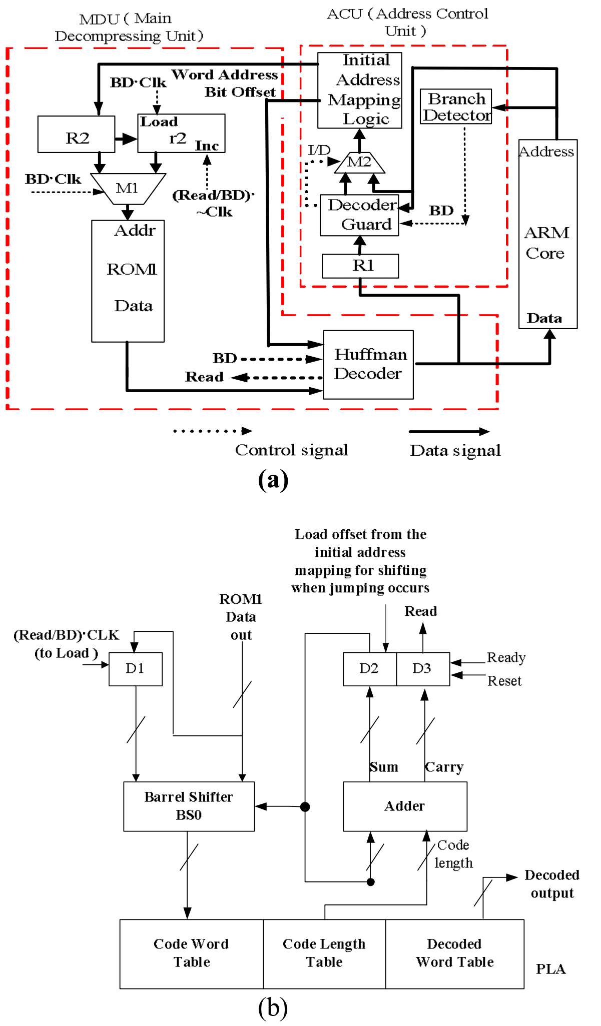

Figure 1. (a) The new version of decompression engine without back-up for flushing unit, (b) the corresponding Huffman code decoder.

A dual-stream pipelined decompression engine was developed in our previous work in [11, 12]. It incurs no delay and without stopping to refill the cache when jumping. However, when jumping occurs, to fetch the instruction of the destination address of the jumping, there is one clock cycle late. Therefore, the first instruction of each basic block should not be compressed and should be put in another ROM. The execution thus goes through two streams, i.e., one stream comes from the ROM accommodating all uncompressed first instructions of all basic blocks, the other stream comes from the other ROM accommodating all the other compressed instructions of all basic blocks. Thus the compression ratio will be sacrificed. (A **basic block** is a block of instructions that has only one entry point. An entry point is an instruction which is a destination of a jump or return destination of a subroutine call or interrupt call).

In this paper, we present a new pipelined decompression engine, as shown in Figure 1, with only one program ROM. All instructions can be compressed and the execution comes from a single stream. Therefore, the hardware cost and the compression ratio can be greatly decreased. In the following sections we will describe the details of the architecture.

3. Architecture of Decompression Engine

In the following sections, the word length of ROM1 is dependent on the longest word that the Huffman coding coded. Here, we suppose the word length of ROM1 is **w** (=2^k) bit*s*.

3.1. The Decompression Process Overview

3.1.1. Processing of Branch Instructions

When an instruction issued by Huffman decoder is a branch instruction, then, in the next clock, the instruction is latched into R1 and is fetched into ARM core at the same time.

Then, when the instruction is being decoded in ARM core, at the same time, the instruction is being processed in Decoder Guard and Initial Address Mapping Logic (IAML). The function of the decoder guard is to detect whether the instruction is a branch instruction and then calculate the destination address (uncompressed) to feed into IAML. Then, IAML issues the CIA of the corresponding compressed instruction.

Then, in the next clock, the CIA is latched into R2, and at the same time, the branch instruction is executed in the ARM core.

Once the branch instruction has been successfully executed and causes a branching, the BD signal cause the value of R2 (address of the first word of the basic block) being selected to be input to ROM1 when clock is "1". Then, the corresponding value is read out and is latched to D1 of the Huffman Decoder. At the same time, the value of R2 is latched into r2. When clock is "0", the value of r2 is incremented (the address of the second word) and the value of r2 is selected as an input to ROM1 and thus the corresponding value is read out on the "ROM1 Data Out" port of the Huffman Decoder. That is, two words of the basic block have been read for regular Huffman decoding process (See the Section of Huffman Decoder). Since two compressed words contain more than one uncompressed instructions, the first instruction of the jumping destination is therefore ready for decompression without any delay.

3.1.2. Processing of Non-branch Instructions

After the first instruction of a basic block has been decompressed, since the next word has also been fetched and if the first instruction is not a branch instruction, the second instruction will be decoded in the next clock cycle.

After the second instruction has been decoded, then, in the next clock cycle, if the sum of the lengths of the first and the second instructions is longer than w bits, the Huffman decoder will issue a Read signal. When clock is “1”, the second word will be loaded into D1 of the Huffman Decoder. Then, when clock is “0”, the value of r2 will be incremented, and be selected as the input of ROM1 so that the next word will be read out on the input port of the Huffman Decoder.

Then, in each clock cycle, if the previous instruction is not a branch instruction, if necessary, Huffman Decoder will issue a Read signal to increment the value of r2. Then r2 provides the next address in ROM1 to be read out and decompressed.

3.2. Branch Detector

This unit is used to detect whether the current address is greater than the previous address by 4 (bytes). If so, no branching is occurring. Otherwise, a branching has been successfully executed. Then, a signal, named BD in Figure 1(a), is issued.

3.3. Decoder Guard

The function of the decoder guard is to detect whether the instruction is a branch instruction and then calculate the destination address to feed into IAML. Then IAML issues the compressed address of the corresponding compressed instruction.

If the branch instruction is in direct addressing mode, then, the destination address can be calculated by adding the offset (which is part of a branching instruction), current program counter value and 8. The destination address then will be issued to the IAML. Therefore, there is a Shadow Program Counter (SPC) associated with this unit as a shadow of the program counter inside the ARM core. The SPC is incremented according to the instruction length when there is no branching occurs. When branching occurs (BD signal activated), the SPC is loaded the destination address as a new program counter value.

If the branch instruction is in indirect addressing mode, then, the destination address is directly sent from the address port of ARM core. The destination address will be sent to Decoder Guard to update the SPC value using the BD signal. Once the Decoder Guard have detected the instruction is an indirect branch instruction, it generates the signal I/D to switch the M2 multiplexer, as shown in Figure 1(a), so that the destination address sent into IAML is coming from the address port of ARM core instead of from Decoder Guard as the direct addressing node do.

3.4. Initial Address Mapping Logic (IAML)

Given the destination address of a branch instruction by the Decoder Guard, IAML generates the CIA of the first instruction of the basic block. There are two parts of the address, i.e., a word address part specifies where is the instruction located in ROM1 and a k-bit offset part (for the ROM1 with 2k-bit = w per word) specifies the compressed instruction is located from which bit in a word.

3.5. Huffman Decoder

For the Huffman decoder, our previous versions in [11, 12] use the one in [9, 10]. In this new version, we design a revised one as shown in Figure 1(b).

The Huffman Decoder is started after D1 has been filled and the next word has been put on the input “ROM1 Data Out”. The reading of these two words can be completed in a single clock cycle. When the clock is “1”, and one of Read or BD signals is activated, the contents of the address in R2 (for jumping) or r2 (for sequentially executing) will be latched to D1 of the Huffman Decoder. When the clock is “0”, the contents of the address in r2 (latched from the value of R2 plus 1 for a jumping or the value of r2 plus 1 for sequentially executing) will be sent to the input “ROM1 Data Out”.

Then, these two words are sent to the barrel shifter BS0 simultaneously. The number of bits to be shifted comes from D2 which is the sum result of an adder. The adder adds the lengths of the current codeword and the previous codeword. If the sum is greater than w bits, a carry is generated to signal that the next word should be read in from the ROM1. The contents of D2 may also come from the k-bit offset field of the output of IAML if jumping occurs.

5. Experimental Results

The correctness for circuit and timing is verified by simulation using the ModelSimXEIII software by modeling each benchmark in Verilog HDL. The Verilog programs then have been implemented using the synthesizer Design Compiler of Synopsis Co. and TSMC .18 cell library. The die size comparison results are shown in Table 1. The compression ratio for each bench mark is compared with our previous version of decompression engine and the results are shown in Table 2. Note that the compress ratio in Table 2 is not based on field-partition compression method as described in [11, 12] but based on the compression on the whole instruction.

6. Conclusions

This paper has presented a single-stream pipelined architecture for the execution of compressed instructions without delay penalty. A highly localized program style helps the reducing of the cost and helps the execution efficiency using our architecture.

10. References

[1]A. Wolfe and A. Chanin. “Executing Compressed Programs on an Embedded RISC Architecture.” Proceedings of the 25th Annu. Inter. Sympo. On Microarchitect., pp.81-91, Dec. 1992.

[2]C. Lefurgy, P. Bird, I. Chen and T. Mudge. “Improving Code Density Using Compression Techniques.” Proceedings of the Thirtieth Annual IEEE/ACM International Symposium on Micro-architecture, pp. 194-203, Dec. 1997.

[3]Haris Lekatsas, Jorg Henkel, W. Wolf, “Design and Simulation of a Pipelined Decompression Architecture for Embedded Systems.” ISSS’01, Oct. 1-3, 2001.

[4]H. Lekatsas and W. Wolf. “SAMC: A Code Compression Algorithm for Embedded Processors.” IEEE Transactions on Computer Aided Design, Vol.18:1689-1701, December 1999.

[5]Y. Xie, W. Wolf, and H.Lehtsas, “Compression Ratio and Decompression Overhead Tradeoffs in Code Compression for VLIW Architectures.” Proceedings of the 4th International Conference on ASIC, pages 337-341, Oct. 2001.

[6]Y. Xie, W. Wolf and H. Lekatsas. “Code Compression for VLIW using Variable-to-fixed coding.” Proc. of ISSS, 2002.

[7]C. H. Lin, Y. Xie and W. Wolf. “LZW-Based Code Compression for VLIW Embedded Systems.” Proc. of DATE, 2004.

[8]Yuan-Long Jeang, Jen-Wei Hsieh, Yong-Zong Lin, “An Efficient code Compression/Decompression System Based on Field Partitioning.” International Technical Conference on Circuits/System, Computers and Communications (ITC-CSCC 2005).

[9]Ming-Ting Sun and Shaw-Min Lei. “An Entropy Coding System for Digital HDTV Application.” IEEE Tran. On Circuits and System for Video Technology, Vol. 1, p. 147-155, Mar. 1991.

[10]Ming-Ting Sun and Shaw-Min Lei, “High-Speed entropy for HDTV.” IEEE 1992 Custom Integrated Circuits Conf.

[11]Yuan-Long Jeang, Jen-Wei Hsieh, Yong-Zong Lin, “An Efficient Instruction Compression/Decompression System Based on Field Partitioning,” 2005 IEEE International Midwest Symposium on Circuits and Systems, Aug. 7-10, 2005.

[12]Yuan-Long Jeang, Chih-Chung Tai, Yong-Zong Lin, “A New and Efficient Field-Partition Based Code Compression and Its Pipelined Decompression System” 2006 International Conference on Innovative Computing, Information and Control (ICICIC-06), Aug. 30-Sept. 1, 2006, Beijing, China.

Table 1 The die size comparisons for benchmarks using old and new version of decompression engine

Benchmarks	ADC (μm^2)	DAC (μm^2)	LCD (μm^2)	RTC (μm^2)	Keyboard (μm^2)
Previous version	41629.89	42085.61	45451.52	39337.62	45282.29
New version	38336.76	37837.80	39627.40	35702.25	42947.15
Decreased (%)	7.91	10.09	6.65	9.24	5.16

Table 2 The compression ratio comparisons for benchmarks using old and new version of decomp. engine

Benchmarks	ADC (%)	DAC (%)	LCD (%)	RTC (%)	Keyboard (%)
Previous version	**34.73**	**35.25**	**36.96**	**30.41**	**27.36**
New version	**21.72**	**21.92**	**20.13**	**18.92**	**23.57**
Decreased (%)	**13.01**	**13.33**	**16.83**	**11.49**	**3.79**