Proceedings of the 5th WSEAS Int. Conf. on Instrumentation, Measurement, Circuits and Systems, Hangzhou, China, April 16-18, 2006 (pp235-239)

Study on LZW algorithm for Embedded Instruction Memory.

ADBULLAH A. HUSSAIN
MICROELECTRONICS
Harbin Institute of Technology
Flat No. 202, Building No.6, Harbin 150001,
CHINA

MAO ZHIGANG
MICROELECTRONICS
Harbin Institute of Technology
Science Building, 10th floor, Harbin 15000,
CHINA

Abstract: - Diminishing instruction size is playing an increasingly important role for decreasing the dissipation power in portable computing and wireless communication markets and embedded system. We study compression instructions with LZW algorithm for three programs under two RISC processors. Implementation of algorithm on the offline-instructions is examined to measure the compression ratio. This study is useful for compressed code systems where instructions are stored in a compressed format and decompressed on demand. The result is a significant reduction on the size of the instruction memory and power consumption as well. Our scheme supposes untouched architecture of processor and retains all the processor functions by doing compression process on the 32-bit instruction format. This paper is a step of designing decompressor unit according to LZW in terms of small size instruction memory and low-power embedded system.

Key-Words: - Code Compression, LZW, Algorithm, Embedded System, Low-Power, Instruction compression.

1 Introduction

Previously, the research project was focusing on instruction compression to reduce memory size. Wolfe and Chanin were the first to propose an embedded processor design which incorporates code compression [1] where Huffman coding algorithm is used to compress cache blocks. Charles Lefurgy et al [2] proposed techniques that compress instructions so that they are easily de-compressible. A number of industrial efforts have also emerged, including MIPS16 [3] and ARM's Thumb processor [4]. Haris Lekatsas and Wayne Wolf [5] compressed the instruction segment by using algorithm of arithmetic coding in combination with a Markov model. They provided experimental results on two architectures, Analog Devices Sharc and ARM's. ARM and Thumb instruction sets, show that programs can often be reduced more than 50%.

Some other research project has investigated code compression's effect on power consumption; Yoshida et al. [6] suggested a logarithmic-based compression scheme where 32-bit instructions map to fixed but smaller width compressed instructions. They investigated the power consumption of the system using memory area only. Benini et al. [7] proposed a scheme where some frequently appearing instructions are compressed to 8 bits. Since only a small (but frequently appearing) subset of instructions is compressed, 8 bits are sufficient for encoding them. This ensures compressed instructions are always fixed-length (8 or 32 bits) hence decoding is very simple and fast.

In the embedded system area, the investigation should involve the design schemes that would lead to reduce power dissipation on system as well as reducing memory size. It reduced by compressing instruction size which will lead up to small embedded memory on the system, diminish buses transaction, and power dissipation on the system see [8] [9] [10].

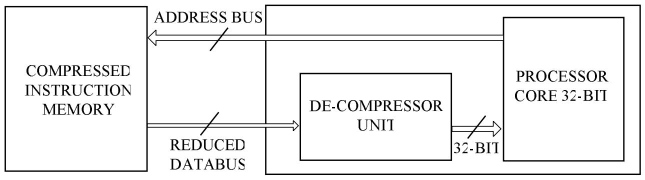

Fig.1: architecture for compressed memory.

Our offering architecture is showed in figure (1), compressed instruction memory, general purpose core 32-bit, and added de-compressor unit. We are assuming no touching the processor architecture. However, the added de-compressor unit to the processor core in order to return the demand instruction to original format. Our scheme considered the added size and power to the system should maintain less than previous modification, but it leads up to small size of instruction memory.

2 Compressing method

In this section, we will introduce the data compression algorithm that used in compression of the instructions of the programs under two RISC processors SPARC and ARM. The programs are WLAN that we implement 802.11 protocol, and SPEC-Benchmarks; go and li.

2.1 Algorithm description

Reducing the size of program would be done by compressing the size of the instruction utilizing a compression algorithm. That is selected according to; compressing ratio (The ratio of the size of compressed instruction segment over the size of original instruction segment), decompressing speed, and algorithm is not much complicate to reform the instruction to original format. The selected algorithm by our case is one of Ziv-Lempel family algorithms that is called Lempel-Ziv-Welch algorithm (LZW) [11]. The features of the algorithm stimulated us to employ it which presents good compression ratio, high decompressed speed and amazingly simple. The arbitrary length of the algorithm outputs considered a serious drawback when implementing it on instruction set.

The key insight of the method that is possible to automatically build a dictionary of previously seen strings in the text being compressed. The dictionary does not have to be transmitted with the compressed text, since the decompressor can build it the same way the compressor does, and if coded correctly, will have exactly the same strings that the compressor dictionary had at the same point in the text.

2.2 Building dictionary and coding

The dictionary starts off with 256 entries, one for each possible character (8-bit string). Every time a string not already in the dictionary is seen, a longer string consisting of that string appended with the single character following it in the text is stored in the dictionary. The output consists of numeric codes (integer indices) into the dictionary. These initially are 9 bits each, and as the dictionary grows, can increase to up to 16 bits. A special symbol is reserved for "flush the dictionary" which takes the dictionary back to the original 256 entries, and 9 bit indices. This is useful if compressing a text which has variable characteristics, since a dictionary of early material is not of much use later in the text. This use of variably increasing index sizes is one of Welch's contributions. Another was to specify an efficient data structure to store the dictionary. The LZW compression algorithm in its simplest form is shown in figure 2.

Routine LZW_COMPRESS

```
STRING = get input character
WHILE there are still input characters DO
        CHARACTER = get input character
  IF STRING+CHARACTER is in the string table then
      STRING = STRING + character
  ELSE
        output the code for STRING
        add STRING+CHARACTER to the string table
        STRING = CHARACTER
  END of IF
END of WHILE
output the code for STRING
```

Fig. 2: The Compression Algorithm

Taking an example would give better explanation about the algorithm. To begin with, all characters that may occur in the binary file are assigned a code. Suppose the MOV instruction that has chosen from the 32-bit ARM processor Instruction set as binary file to be compressed, only substituting the characters ‘a’ instead of code ‘0’ and ‘b’ instead of code ‘1’ for simplicity.

ababaabbbabbaabbababaaabbbbaaaba

The mapping between character strings and their codes is stored in a dictionary. Each dictionary has two fields; key and code. The character string represented by code is stored by the field key. The dictionary for above example is showed in table (1).

The LZW decompression algorithm in its simplest form is shown in figure 3. The decompression algorithm is a companion algorithm for compression one. It needs to be able to take the stream of codes output from the compression algorithm, and use them to exactly recreate the input stream. it is reverse direction process with using same dictionary.

code	key	code	key
0	a	9	bba
1	b	10	aab
2	ab	11	bbab
3	ba	12	baba
4	aba	13	aaa
5	aa	14	abbb
6	abb	15	bbaa
7	bb	16	aaba
8	bab		

Table 1: LZW compression dictionary to the example.

```
Routine LZW_DECOMPRESS
    Read OLD_CODE
    output OLD_CODE
    WHILE there are still input characters DO
        Read NEW_CODE
        STRING = get translation of NEW_CODE
        output STRING
        CHARACTER = first character in STRING
                add OLD_CODE + CHARACTER to the
translation table
        OLD_CODE = NEW_CODE
        END of WHILE
```

Fig. 3: The Decompression Algorithm

3 Experimental Results

Compressing the execution program idea is not new and the compressing algorithm as well. However, the compressing-decompression scheme could be promising that we are presenting. However, storing the instruction in memory depends on the instruction size and the instruction location. If the compressing process diminishes the size of instruction and keep the location of new form of instruction as it was. That would be fine solution to the random access to the instruction after compression. The idea is showed in figure 4.

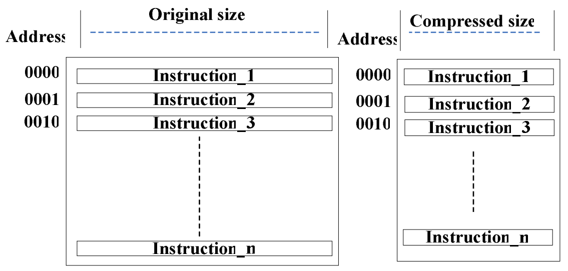

Fig. 4: Retaining memory location of compressed Inst.

The architecture design of some processors may support two modes; one for normal use and the other for the low power use. That what has design in ARM processor which the ARM Thumb shrinks the instruction size. The ARM Thumb has diminished some functions of original design.

However, our scheme supposes to compress the instruction 32-bit to be in a compressed format without touch the architecture of the processor. This technique has given fine compression ration and more than 50% for the three programs and under the ARM processor and SPARC processor. Moreover, processor functions will not be lost. The figures (5) and (6) represent the statistical study on the three programs that implement on the two RISC processors. Figures are showing the usage instructions by compiler to interpret programs. These varieties of instructions are not too much that would effect positively on the dictionary which is not expanded largely during the compression processing. However, table (2) is shown compression ratio of the programs in binary version with considering all above arguments in study.

processor	Compression ration %		
	WLAN	go	li
ARM	55	58	59
SPARC	50	55	53

Table 2: Compression ratio under our scheme.

We expected the used power by the compressed program would be lower, and we are going to improved after complete the design of the decompressed unit. Only one thing remaining that performance of the system under these considerations should not be much lower the case the system with out decompressor unit. However, improving that will be next step of our research on this system under our scheme.

4 Conclusion

This paper has examined LZW algorithm to compress the offline-instructions for three programs under ARM and SPARC processors. The compression ratio is significantly reducing size the instructions without losing any of functions support by the processor. Future work will continue to prove and design decompressor unit for embedded system in terms of low-power.

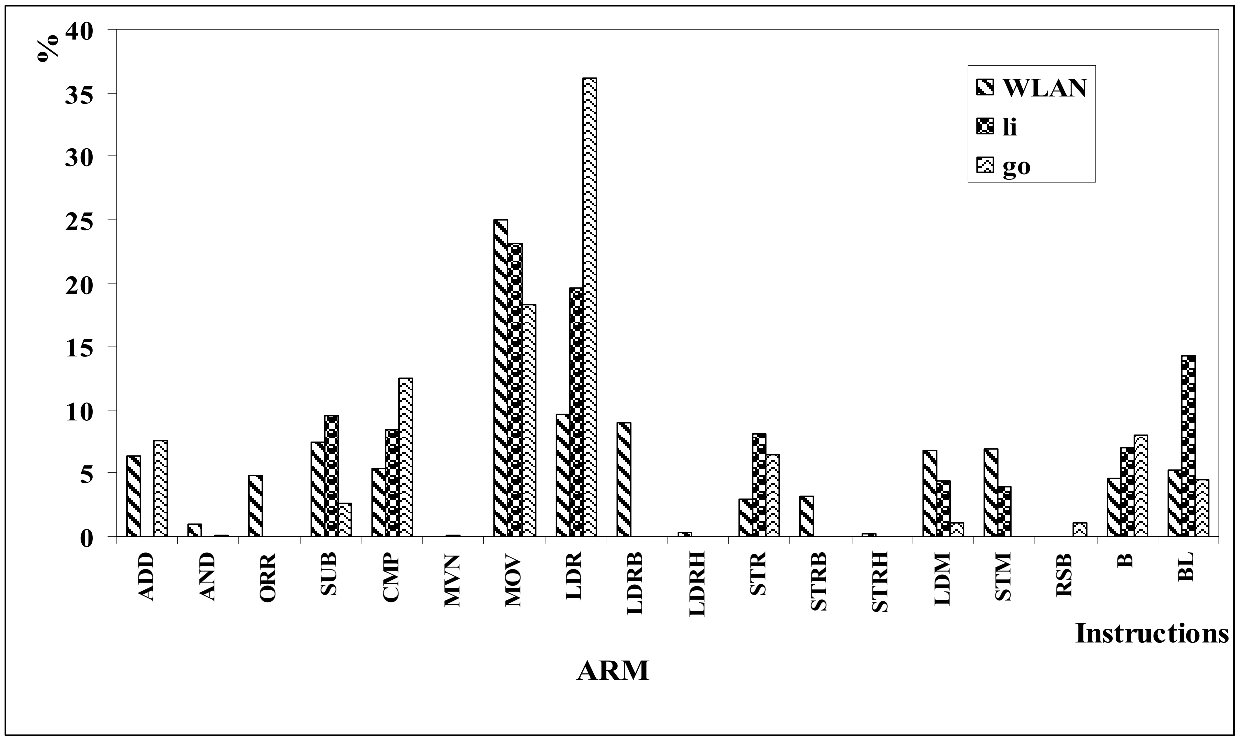

Fig. 5: Instructions usage under ARM processor.

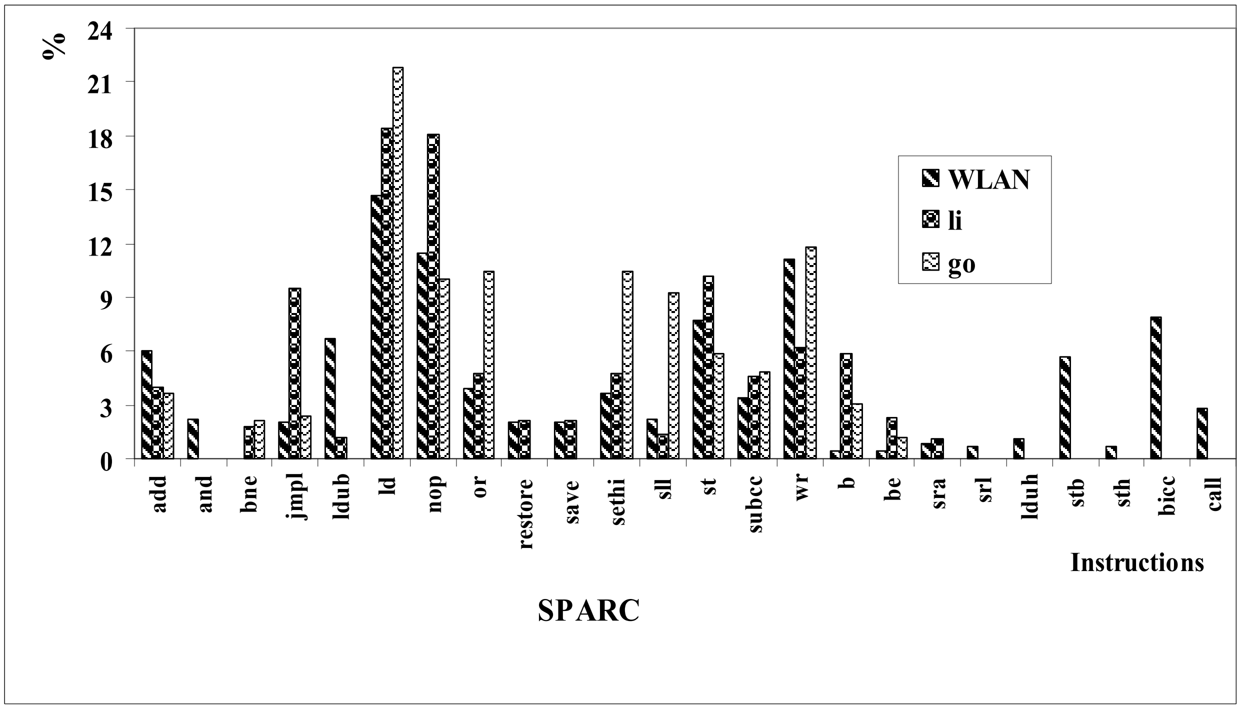

Fig. 6: Instructions usage under SPARC processor.

References:

[1] A.Wolfe and A. Chanin, Executing Compressed Programs on an Embedded RISC Architecture, *Proc. 25th Ann. Int'l Symp. On Microarchitecture*, pp. 81-91, December, 1992.

[2] C. Lefurgy, P. Bird, I.-C. Cheng, and T. Mudge. Improving code density using compression techniques, *In Proc. 30th International Symposium on Microarchitecture*, December, 1997 pp. 194-203.

[3] K.D. Kissell, MIPS16: High Density MIPS for the Embedded Market, *Silicon Graphics Group*, 1997.

[4] Advanced RISC Machines Ltd., An Introduction to Thumb, March, 1995.

[5] H. Lekatsas and W. Wolf, SAMC: a code compression algorithm for embedded processors, I*EEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 18, No. 12, 1999, pp. 1689-701.

[6] Y. Yoshida, B.-Y. Song, H. Okuhata, T. Onoye, I. Shirakawa, An Object Code Compression Approach to Embedded Processors, *Proc. of the Int'l Symp. On Low Power Electronics and Design (ISLPED-97)*, 1997, pp. 265-268.

[7] L. Benini, E. Macii and M. Poncino. Selective Instruction Compression for Memory Energy Reduction in Embedded Systems, *IEEE/ACM Proc. of Int'l Symp. On Low Power Electronics and Design*, 1999, pp. 206-211.

[8] E. Billo, R. Azevedo, G. Araujo, P. Centoducatte, Design of a decompressor engine on a SPARC processor, *Proceedings of the 18th annual symposium on Integrated circuits and system design, Brazil*, 2005,pp 110 - 114.

[9] L. Benini, A. Macii, and A. Nannarelli, Code compression for cache energy minimization in embedded systems, *IEE Proceedings on Computers and Digital Techniques*, Vol. 149, No. 4, July 2002, pp.157-163.

[10] Haris Lekatsas, Jörg Henkel, Wayne Wolf, Code compression for low power embedded system design, *Proceedings of the 37th conference on Design automation, June 05-09, Los Angeles, California, United States*, 2000, pp.294-299.

[11] T. Welch, A technique for High Performance Data Compression, *IEEE Computer*, Vol. 16, No. 6, 1984, pp. 8 -19.