A NOVEL CODE COMPRESSION APPROACH FOR EMBEDDED RISC PROCESSOR

Ramani.G[1] and Dr.K.Geetha[2]

[1]Department of Electrical and Electronics Engineering Nandha Engineering college, Erode.
ramani.govindasamy@nandhaengg.org
[2]Department of Electronics and Communication Engineering,Karpagam college of Engineering, Coimbatore
geetha.arulmani@gmail.com

Abstract: *Now a days most of the processors are high performance RISC processors .This paper introduces a novel code compression approach for embedded RISC processor, which reduces the code size and improves the compression ratio. Code compression is the technique to reduce the program size using various code compression algorithms to original instruction sets. There are two methods of compression is used in this paper. One is Dictionary based and the other is statistical compression. Our implementation is assessed through various benchmarking performed on embedded programs. In this approach, an efficient code compression is achieved using lookup table and Canonical Huffman decoder.*

Key words*:RISC processors, Statistical compression, Dictionary based compression, Code compression.*

1. Introduction

Code compression: In general, the efficiency of any compression technique is measured using Compression ratio**.** Compression ration is defined as the ratio between the compressed code size and the original code size. Basically there are two typres of compression techniques. First one is Dictionary based compression and the second is Statistical compression. These two methods were Applied for RISC processors to improve the code compression efficieny.

Dictionary based compression: In this method, the entire sequences of common instructions are selected and replaced by a single new code word and then, which is used as an index to the dictionary that contains the original sequence of instructions. In both the methods look up tables are used to store the original instructions and the compressed instructions serve as indices to the tables.

Statistical compression: In statistical compression technique, the frequency of the instruction is used to select the size of the code words and replaces the original. Hence, the shorter code are used for the most frequent sequences of instruction and the longer code words are replaced by less frequent sequences. The look up tables are used.

Look up table: The table, which is generated from statistical and dictionary compression methods by sorting the table entries to reduce the number of bit toggles between every two sequential instructions and then it is optimized to improve code compression ratio.

2. Related Work

Dictionary based compression Method

In existing system, the Dictionary was created using dynamic frequency algorithm.Here,first the profile creation, which is used to identifying the basic blocks. In second step, program codes are compressed and stored in dictionary. The look up table indicates the starting address, End address and new address of the dictionary as shown in (Figure1)

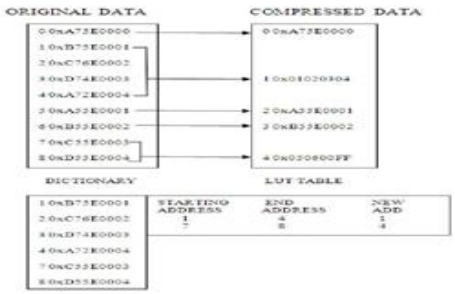


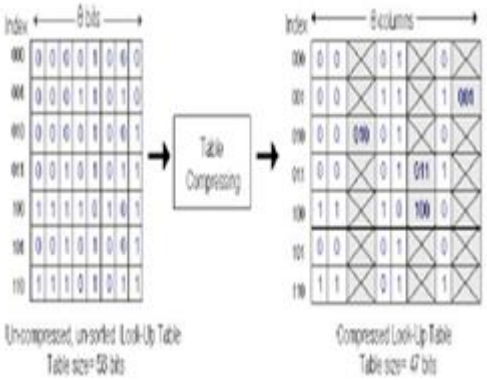
Figure.1

In this system, the look up table minimization scheme is used to generate look up table ,which is used in dictionary based compression. The table is used to minimize the number of bit transitions per column and used to save the indices. In figure 1 shows the look up table with number of entries 7 and the instruction word length of WL=8.The size of the original table is 56bits.By using compression algorithm(Yoshidha) the size of the lookup table is reduced to 47 bits. With the help of sorting the table with more number of columns were compressed. The higher table compression is achieved by compressing more table columns and that is basically depends on the way of sorting entries.The sorting of entries is carried in two phases. In phase I, the gray code is generated for the word length WL,

then we locate each table entry in its corresponding position in the generated gray code. In phase II, the Lin-kernihnan algorithm is used to sort the table entries. The sorting of table entries are based on the distance between two entries. Here ,the distance between two entries is the number of positions for which the corresponding bits are different. This method of sorting have no impact on the compressed instructions, because all of them have same code length. This will decrease the of the lookup table and there by increases the compression ratio.

Procedure:

- Un compressed binary code ,the entire instruction word is arranged
- All the unique instruction words are stored in lookup table
- In the original code, the every unique instruction word with a binary index to the lookup table in ascending order starting from 0
- The index has a fixed length and it is equal to log2 of the number of unique instructions



Unused bits in the compressed column

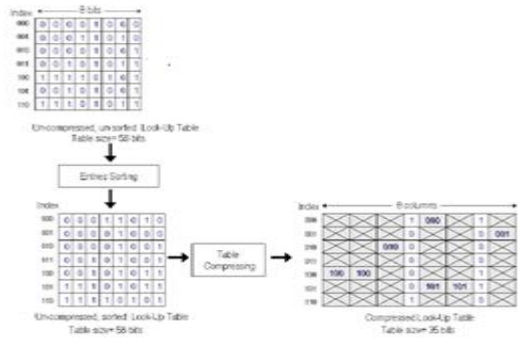


Figure-2

The code compression efficency is calculated as follows,

Where,

$$\text{Compression ratio} = (I_o * \log_2(I_u) + \sum_{i=1}^{WL} Ci) / (WL * I_o)$$

WL: Instruction word length (Fixed)

I_o : Number of original instructions

I_u : Number of table entries

Ci: Size of table column i in bits

For WL=8, Iu=7, Ci where i=1 to 8 and original code table size=56 bits, The compressed table using entry sorting followed by the table compression gives better compression ratio of 62.5%. But, the direct table compression method gives 83.9% of compression ratio. The following table1 gives the difference between two methods.

Table.1

S.NO	METHOD	ORIGINAL TABLE CODE	COMPRESSED TABLE	COMPRESSION RATIO	DE COMPRESSION
1	COMPRESSED TABLE USING TABLE COMPRESSION	56 bits	47 bits	83.9	Data loss is high
2	COMPRESSED TABLE USING ENTRY SORTING AND TABLE COMPRESSION	56 bits	35 bits	62.5	Very minimum data loss

Statistical Code Compression method:

In this method, the lookup table is generated using Canonical Huffman coding and the generated lookup table is equal to the number of different instruction code lengths.Here,the table compression method is used to reduce the table column. Most frequently used instructions are encoded with shortest codes and others vice versa. In Huffman coding method ,the variable length codes can't be decoded, it is difficult when implementing the hardware. But in canonical Huffman ,the code words with the same length are binary representations of consecutive integers(Figure-3)

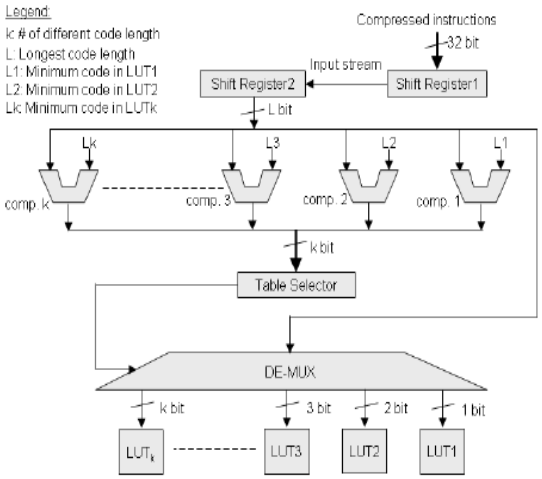


Figure .3

The decoder has two shift registers i.e. 32 bit and L bit registers. The compressed instructions to keep the L-bit register filled each time its content is reduced by shifting the compressed instruction word serially into it. The L-bit shift register transfers the L bit code words to the comparator. The incoming L bit is and the length of the encoded instruction is decoded with the help of comparator. All the comparators compares the incoming L bits with minimum index of the table, the Comparator outputs as either "1" or "0".The table selector find outs the smallest comparator which outputs "1".This comparator refers the code word length and compressed lookup table. The compressed lookup tables are decoded using lookup table decoder. Compression ratio in this scheme as:

$$\text{Compression Ratio} = \frac{\sum_{i=1}^{L} Ni * CLi + \sum_{i=1}^{L} \sum_{j=1}^{WL} Cji}{WL * N}$$

Where,

Ni : Number of instructions which have the code length i

CLi : Code length i

Cji : The size of the column j in table i

If Ni=1 (only one lookup table),then we will obtain the same compression ratio formula. If any instructions are transferred from one look up table to another look up table ,the compression ratio is calculated using the formula given below.

Efficiency = Compressed table gain-
Compressed code loss

Here, the compressed table gain is the difference between the size of the compressed table before and after transferring instructions between them

3.Conclusion & Future Scope:

Normally for the RISC processor ,the Huffman decoders are used for the code compression. In this paper, The lookup table compression method along with canonical Huffman coding is used for improving the code compression ratio. This method provide an average compression ratio of 62% for various application programs.

Power consumption is also a major factor during code compression and decompression process. Power saving is to be concentrated in future. The second factor is speed , code compression and decompression speed is also be considered in future.

References

[1] Ramani G & Geetha Arulmani, *An efficient code compression for MIPS32 processor using dictionary and bit-mask based static and dynamic frequency algorithm & COMPEL - The international journal for computation and mathematics in electrical and electronic engineering* , vol. 35, no. 5 2016.

[2] Ramani G & Geetha Arulmani, .*Bitmask-based Code Compression Technique for MIPS32 Bit Processor*, International Journal of Trend in Reasearch and Development vol. 3, no. 4. (IF:3.025) 2016.

[3] Ramani G, Dinesh Babu N, , *Combined Dictionary And Bit-Masking Based Code Compression For Embedded Systems*, International Journal of Applied Engineering Research (IJAER), vol. 10, no. 20, pp.17896-17899. (IF:0.14) 2015.

[4] Y. Xie, W. Wolf and H. Lekatsas, *A code decompression architecture for VLIW Processors*, In Proceedings 34th ACM/IEEE International Symposium on Microarchitecture, pp. 66-75, 2001.

[5] Y. Xie, W. Wolf and H. Lekatsas,*Code compression for VLIW processors using variable-to-fixed coding. In IEEE Transactions on Very Large Scale Integration (VLSI) System,* Vol. 14, No. 5, pp. 525-536, 2006.

[6] S. Seong and P. Mishra, *A Bitmask-based Code Compression Technique for Embedded Systems*,24th IEEE/ACM International Conference on Computer-Aided Design (ICCAD06), pp. 251254, 2006.

[7] S. Seong and P. Mishra,*An efficient code compression technique using applicationawarebitmask and dictionary selection methods*,IEEE/ACM Proc. of Design Automation and Test in Europe Conference (DATE07), pp. 582-587, 2007.

[8] X. Kavousianos, E. Kalligeros and D. Nikolos, *Multilevel Huffman Coding: An Efficient Test-Data Compression Method for IP Cores*, IEEE Transaction on Computer- Aided Design of Integrated Circuits and

Systems, Vol. 26, No. 6, pp. 1070-1083, June 2007.
[9] H. Lekatsas, J. Henkel, and W. Wolf, *Arithmetic Coding for Low Power Embedded System Design*,Princeton University, NEC USA, 2000.
[10] H. Lekatsas, J. Henkel, and W. Wolf, *Code Compression as a Variable in Hardware/- Software Co-Design. International Workshop on Hardware/Software Co-Design*, 2000.
[11] H. Lekatsas, J. Henkel, andW.Wolf, *H/S Embedded Systems: Design and simulation of pipelined decompression architecture for embedded systems*, Proceedings of the international symposium on systems synthesis, 2001.
[12] T. Bonny and J. Henkel., *LICT: Left-uncompressed Instructions Compression Technique to Improve the Decoding Performance of VLIW Processors*, In 46th ACM/EDA/IEEE Design Automation Conference (DAC'09), pp. 903-906, San Francisco CA, USA, July 2009.
[13] Heikkinen, J, Takala, J & Corporaal, H, *Dictionary-based program compression on customizable processor architectures', Microprocessors and Microsystems*, vol. 33, no. 2, pp. 139-153, 2009
[14] Thuresson, M & Stenstrom, P 2005, *Evaluation of extended dictionary-based static code compression schemes', in Proceedings of the 2nd conference on Computing frontiers*, pp. 77-86.
[15] Yang, L, Dick, RP, Lekatsas, H & Chakradhar, S 2005, *'CRAMES: compressed RAM for embedded systems', in Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 93-98.