A Cycle-Stealing Technique for Pipelined Instruction Decompression System for Embedded Microprocessors

Yuan-Long Jeang, Tzuu-shaang Wey, Hung-Yu Wang, Chih-Chung Tai, and Ching-Hua Chu
1Department of Information Engineering, 2Department of Electronic Engineering, 3Department of Electronic Engineering
National Kaohsiung University of Applied Sciences, Kun Shan University, Tainan County, Taiwan
Kaohsiung City, Taiwan
yljeang@ms22.hinet.net

Abstract

For instruction decompression, many techniques have been developed. However, when branching and cache missing occur, they may either incur delays to refill buffers or sacrifice the compression ratio or slow down the clock rate. This paper presents a new technique based on cycle-stealing technique to eliminate all these defects. The simulation results for several benchmarks show that the average compression ratio, the hardware cost and the speed are all better than other techniques.

1. Introduction

Cost, speed and power consumptions are three important issues for system on a chip. The area of the program memory in contemporary chips usually occupies 40%-80% area of the whole chip. In general, there is a very high degree of repetitions of instructions in a program. Therefore, based on Huffman Coding method for several benchmarks, compressing programs could reduce program memory size up to 70%. The die size for a program ROM with 1-M bits may almost equivalent to the die size of a RISC CPU. Therefore, reducing program memory capacity to decrease cost and power consumption is a popular research. However, code compression is performed before the chip design while the code decompression is performed when the chip is running. Thus, the efficiency of the decompression engine affects not only the execution time but also the compression ratio and cost.

2. Related Work

Wolfe and Chanin [1] is the first to present the code compression and decompression schemes for embedded processor. 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. A byte-based Huffman coding method is used to compress program code and use Line Address Table (LAT) to map the compressed block address for solving the branch issues. The average compression ratio is 73% for MIPS code. The compression ratio is defined as the ratio of compressed program size over original program size. Harised. 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.

A dual-stream pipelined decompression engine was developed in our previous work in [11, 12]. It doesn’t need to stop to refill the cache when jumping. However, 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, comes from the ROM accommodating all uncompressed first instructions of all basic blocks, and comes from the other ROM accommodating all the other compressed instructions of all basic blocks. The compression ratio will be sacrificed.

A single stream pipelined decompression engine [2] has been presented to eliminate the need of two program ROMs and all instructions can be compressed and the execution comes from a single stream. However, the clock length is longer than dual-stream pipelined technique.

This paper presents a cycle-stealing technique to eliminate the delay of refilling buffers when jumping and cache missing, and with a better compression ratio.
3. The Principle of Decompression Engine

3.1 A Three-Stage Decompressing Engine

Hereafter, an address in an uncompressed program is abbreviated as *uncompressed address* while an address in a compressed program is abbreviated as *compressed address*.

As shown in Fig. 1, since the output of processor is an uncompressed address while the input to program memory is a compressed address, a decompression engine should include an “**Address Mapping**” (AM) to map each uncompressed address to a corresponding compressed address. Therefore, for a system with compressed program, the obtaining of each instruction should go through three stages: AM, PM and decoder.

For sequential program, if there is a counter to provide the next address for the PM, despite of the uncompressed address issued from the processor, the processor also can obtain the next instruction at the right time. In Fig. 1, when an instruction has been put across a word boundary, the Decoder will issue a Read signal to read the next word to be decoded. The Read signal will increment the counter and get the next word continuously. Thus, for sequentially executed program, AM is not necessary.

![Figure 1 A 3-stage decompression engine](image)

However, if there is a branching, the compressed address of the target instruction should be put in AM and the target instruction will be obtained by going through these 3 stages and therefore involved a 3-clock delay. Our design can eliminate the delay.

3.2 Pre-detection of Branch Instructions

Since after an instruction has been fetched into processor, usually there are several clocks of leisure time (such as the decoding and execution phases) before it is executed. If we can know beforehand that a branch instruction has been decoded, then during these several clocks of leisure time, the branch target address can be calculated and the compressed address can be obtained. As shown in Fig. 2, a “**Decoder Guard**” (DG) is used for the detection of a branch instruction and for the calculation of the uncompressed target address by using an associated “**shadow program counter**” (SPC), which traces the current value of real program counter. Then this uncompressed target address will be sent to AM to obtain the corresponding compressed address when the decoded branch instruction is being in the decoding phase inside the processor. (The uncompressed address will be latched as the value of the SPC in the Decoder Guard).

![Figure 2 Decoder Guard and the competition of the PM by Decoder and AM](image)

However, if the Decoder continuously reads new words from PM, the compressed address of the target instruction coming out from AMU can’t be read out from PM before the execution of the branch instruction. The cycle stealing technique is used to steal a cycle to read the compressed target instruction out from PM.

If there is a clock cycle of leisure time that Decoder need not to get the compressed instruction from PM before the branch instruction has been executed, then the compressed target instruction can be read out in a buffer, say Target Buffer as shown in Fig. 2. Then, once the “Branch Detector” (BD) detects a branching, the target instruction will be decoded.

3.3 Principle of Cycle-Stealing

How to steal a cycle? We just expand the length of program memory for every reading. For example, if the length of the longest instruction coded is 36 bits for 32-bit instruction machine, then we let the word length for each reading is 72 bits, i.e., double the length of the longest coded instruction. Suppose the number of stages for an instruction pipeline is greater than or equal to 2, then there is at least a clock cycle can be stolen before the branch instruction executed. If the compressed branch instruction is located across the word boundary, then in the next clock cycle, the decoder will send a “Read” signal to get the next word. Since a word can accommodate at least two instructions, therefore, in the next second clock cycle (within two clock cycles), decoder will not read again and the target compressed address can be read out in
the target buffer. If the compressed instruction is not located across the word boundary, then in the next clock cycle, the decoder will not send a “Read” signal to get the next word. Thus, in the next clock cycle, the target compressed address can be read out in the target buffer. Thus, cycle stealing can be made.

### 3.4 The Decompression Process Overview

#### 3.4.1. Processing of Indirect Branch Instructions.

There are two types of branch instructions for most of processors, namely, direct branch (DB) and indirect branch (IB) instructions. For DB instructions, it is easy to detect the target address from the instruction code. However, for the IB instructions, the target address is sure only after the instruction has been executed. Thus, for the IB instructions, as shown in Fig. 3, we use an Indirect Jumping Memory (IJM) to hold the first two uncompressed instructions of the target address. Then, after two clocks, the third and the following instructions can be read out through AM, PM and Decoder. Here, the uncompressed address of the target address is used in AM to search the compressed address of the third instruction. The DG also issues an ID0 signal when an IB instruction has been decoded. Then, after the IB instruction (two clocks later) has been executed and an uncompressed address is issued and the target instruction is read out from IJM, the ID2, which is shift from ID0, will make the M2 select the right target instruction. Similarly, ID3 is used to select the second instruction of the target address of the IB instruction.

#### 3.4.2. Processing of Direct Branch Instructions.

When DG detects a branch instruction, then, in the next clock, the uncompressed target address is latched into R1 and the branch instruction is fetched into ARM core to be decoded at the same time. The DG also issue a Br0 signal and this signal will be shifted to left each clock. At the same time, AM issues the corresponding compressed address.

Then, in the next clock, the compressed address is latched into R2, and at the same time, the branch instruction is started to be executed in the ARM core and the Br1 activated. The compressed target instructions will be loaded into D1, D2 and D3, which are called Target Buffers.

Once the branch instruction has been successfully executed and causes a branching, the Branch Detector unit will issue a BD0 signal to cause the values in Target Buffers be selected into D4, D5, and D6 of Huffman Decoder. Otherwise, the decoding will be continued in the following instruction of the branching instructions.

#### 4. 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 and speed comparison results with our previous result in [13] are shown in Table 1. The main contribution of this paper is that the speeds of the decompression engines for every benchmark in this paper are usually much faster than our previous version in [13], though the die sizes are usually larger than previous version.
5. Conclusions

The paper has presented a pipelined architecture for the execution of compressed instructions without delay penalty. This version is much faster than our previous version since that the program memory and the decoder have been separated to two stages, which is a single stage in our previous versions. This version also deals with the indirect branch instructions. A highly localized program style helps the reducing of the cost and helps the execution efficiency using our architecture.

6. References


Table 1 the die size benefits for benchmarks using the decompression engine

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Uncompressed Program Size (Byte)</th>
<th>A. Uncomp. Memory Area (mm²)</th>
<th>Compressed Program Size (Byte)</th>
<th>B. Compressed Memory area (mm²)</th>
<th>C. Decomp. System area (mm²)</th>
<th>Benefits Area (mm²) (A-B-C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADPCM</td>
<td>34752</td>
<td>1.61</td>
<td>12024</td>
<td>0.67</td>
<td>0.294</td>
<td>0.646(40%)</td>
</tr>
<tr>
<td>BMR</td>
<td>34768</td>
<td>1.61</td>
<td>12042</td>
<td>0.68</td>
<td>0.297</td>
<td>0.633(39.3%)</td>
</tr>
<tr>
<td>DCT</td>
<td>60500</td>
<td>2.7</td>
<td>22050</td>
<td>1.17</td>
<td>0.471</td>
<td>1.059(39.2%)</td>
</tr>
<tr>
<td>MTMF</td>
<td>44544</td>
<td>2.04</td>
<td>15831</td>
<td>0.92</td>
<td>0.361</td>
<td>0.759(37.2%)</td>
</tr>
<tr>
<td>FFT</td>
<td>47404</td>
<td>2.165</td>
<td>16974</td>
<td>0.96</td>
<td>0.383</td>
<td>0.822(37.9%)</td>
</tr>
<tr>
<td>Filters</td>
<td>40040</td>
<td>1.864</td>
<td>14058</td>
<td>0.831</td>
<td>0.324</td>
<td>0.709(38%)</td>
</tr>
<tr>
<td>G.711</td>
<td>36224</td>
<td>1.67</td>
<td>12573</td>
<td>0.7</td>
<td>0.3</td>
<td>0.67(40.1%)</td>
</tr>
<tr>
<td>Huffman</td>
<td>48328</td>
<td>2.2</td>
<td>16695</td>
<td>0.967</td>
<td>0.362</td>
<td>0.871(39.6%)</td>
</tr>
<tr>
<td>IS51CE</td>
<td>33392</td>
<td>1.55</td>
<td>11529</td>
<td>0.65</td>
<td>0.286</td>
<td>0.614(39.6%)</td>
</tr>
<tr>
<td>Math</td>
<td>55972</td>
<td>2.51</td>
<td>19944</td>
<td>1.08</td>
<td>0.417</td>
<td>1.013(40.35%)</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-39.125%</td>
</tr>
</tbody>
</table>