Proceedings of the 5th WSEAS Int. Conf. on Electronics, Hardware, Wireless and Optical Communications, Madrid, Spain, February 15-17, 2006 (pp78-81)

Multithreading Extension for Thumb ISA and Decoder Support

Lan Dong[1] Zhenzhou Ji[1] Guangzuo Cui[2] Mingzeng Hu[1]
Department of Computer Science and Technology
Harbin Institute of Technology [1]
Modern Education Technology Center, The Peking University [2]
No.92, West Da-Zhi Street,Harbin, 150001
CHINA

Abstract:- Dual width instruction set embedded processors such as ARM provide 16-bit instruction set in addition to the 32-bit instructions set for lower energy and memory cost. The combination of hardware multithreading technique with the 16-bit code design can provide a tradeoff between performance and code size. In this paper, extension thread switch instruction (Ts) is added to the Thumb instruction set. With the decoder supporting, no extra cycles are needed to handle the Ts instructions. From analysis, our approach is a more flexible thread switch mechanism and provides better performance with little extra hardware cost.

Key-Word: Embedded processor, multithreading processor, thread switch, Thumb, ARM

1. Introduction

Hardware multithreading is becoming a general applied technique in modern processor design for its high performance as well as not much additional hardware cost. Dual width ISA is simple to implement and provide a tradeoff between code size and performance in embedded systems. This paper, proposes a multithreading thread switch extension to the 16-bit ISA which is called Ts. Ts instructions are also 16-bit instructions, they have the energy saving and small code size properties of Thumb code. Unlike compiler generated VLIW instructions with switch information, St instruction do not merely improve the flexibility to explicitly indicate thread switch position, they do so without adding any cycles to the handling time with the decoder we design. While the Ts extensions described in this paper are for the ARM architecture, the idea of Ts instructions and the support decoder can be applied to other dual width processors. In previous work [2], we showed how to extend ARM architecture to multithreading architecture. In this paper, we describe the extension St instruction and the decoder support to achieve good performance.

The remainder of the paper is organized follows. Section 2 gives a narration of the Ts instructions including the incorporating method and the instruction format. Section 3 describes in detail of the microarchitecture to support the Ts instruction. Section 4 analyzes the performance generally. We give our conclusion and future work in section 5.

2. Extension of Thumb ISA with Thread Switch Instruction

More than 98% of all microprocessors are used in embedded products, the most popular 32-bit processor among them being the ARM family of embedded processors [4].To support explicit multithreading switch of embedded processors, we augment thread switch instruction to the thumb ISA. In real time multithreading systems, the fetching rate of different thread is a critical problem. Our approach is also a more flexible aid mechanism for that. The extension instructions are also 16-bit instructions so they have the benefit of 16-bit code. We augment a thread switch instruction (Ts) to Thumb instruction set. Ts instructions are also 16-bit instructions, they have the energy saving and small code size properties of Thumb code. While the Ts extensions described in this paper are for the ARM architecture, the idea of Ts instructions and the support decoder can be applied to other dual width processors. Ts instruction is incorporated to the instruction sequences at interval of 1 instruction ahead of the instruction tend to switch. So the Ts instruction is 2 instructions ahead of the

instruction inclined to switch. The Ts is decoded simultaneously with the Thumb instruction immediately ahead of it with our decoder support.

3. Microarchitecture

Our design is based on the extension of our previous work [2]. We apply the extension to a 5-stage pipeline. Additionally, we add banked register sets to the original design for multithreading contexts.

The five stages of the pipeline are:(i) instruction fetch; (ii)instruction decode and register read; branch target calculation and execution; (iii) Shift and ALU operation, including data transfer memory address calculation; (iv) data cache access; and (v) result write-back to register file.

The state of the current thread will be copied to a back register set.

3.1 Decoder Design

At first, we get a glimpse of the original design of the decode stage which allows the ARM processor to execute both ARM and Thumb instructions. The original decoder architecture is in Figure1. It has 32 bit instruction buffer and the two instructions in it are selected in consecutive cycles and fed into the thumb decompressor.

The key idea of our approach is to process an Ts instruction simultaneously with the processing of the immediately preceding Thumb instruction.

The overall operation of the hardware design is shown in Figure2.

We make constraints here that the Ts instruction must not be the first instruction in the instruction sequence and two Ts instructions are not allowed to be adjacent. This can be ensured on the general cases in real systems. We exploit the instruction buffer extension idea from [1]. The instruction buffer in the decode stage is modified to exploit the extra fetch bandwidth to keep at least two instructions in the buffer at all times. Thus it is expanded in size from 32 bits to 48 bits. At any time, the relative position in the instruction buffers is the first instruction in ib1, the second is in ib2 and the third in ib3.The Ts instruction is processed by the Ts decoder which generates the signal of thread switch to fetch unit. The Thumb instruction is processed by Thumb decompressor and then the ARM decoder. The design ensures that there is no additional lost in the processor cycle time. The Ts decoder's handling of the Ts instruction is entirely independent of handling of the thumb instruction by the decode stage. In the pipeline diagram Thumb-D and Ts-D denote handling of Thumb and Ts instructions by the decode stage respectively.

The Ts-decoder is very simple. It just recognizes the Ts instruction and sends the switch signal to fetch logic. So the hardware cost for the additional decoder is little. The switch signal indicates to make a switch in the next cycle. So in the next cycle two consecutive instructions are fetched from another thread. The third instruction after the St instruction may be abandoned when it is the instruction from the old thread sequence just following the instruction tend to switch and is fetched simultaneously with the instruction tend to switch. It is showed more clearly in Fig. 6 section 4.

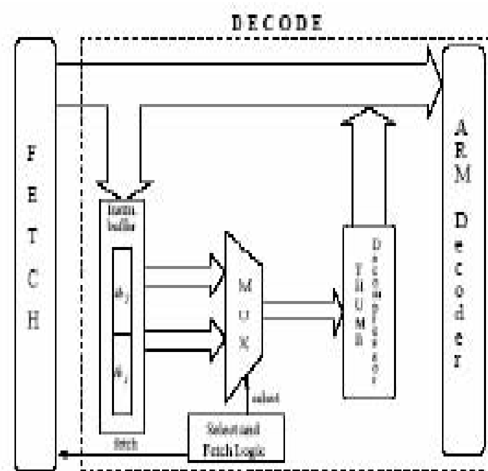

Fig. 1 Thumb Decode Implementation

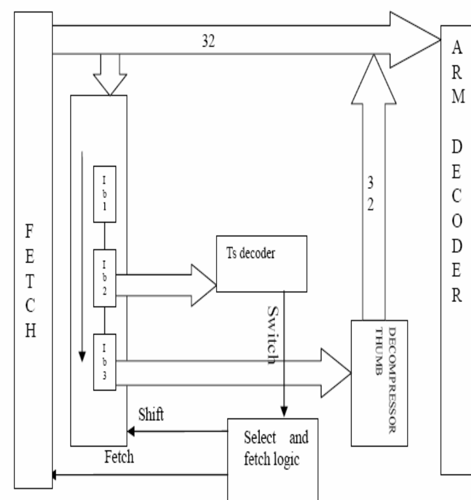

Fig.2 Ts-Thumb Decode Implementation

4. Performance Analysis

Proceedings of the 5th WSEAS Int. Conf. on Electronics, Hardware, Wireless and Optical Communications, Madrid, Spain, February 15-17, 2006 (pp78-81)

We discuss the performance generally here. The instruction sequence is i1, i2, i3, i4, i5, i6… We make assumptions: (1) i1 to be the first instruction of the two consecutive instructions fetched simultaneously. It will not affect the representative because we analyze it by relative position. (2) Other instructions except the instruction inclined to switch is normal instruction which doesn't give rise to extra latencies in the pipleline.

If i3 is an instruction inclined to switch, we incorporate the Ts instruction (ts) to the position 2 instructions ahead of i3. The instruction sequence is changed to i1,ts,i2,i3,b1,b2,b3….b1,b2,b3 are from another thread. The Timing diagram is in the Figure 3. The identifier * in the grid in following figures means the phase may be abnormal due to latencies which could be any kind of delay causing thread switch in common thread switch technique such as control dependence, caching missing and so on. If there are not any extra latencies, we will delete * in the corresponding grid. The latency of the preceding instruction can lead to latency of the following instructions. For more clear comparison of the two timing diagram we put * to the corresponding pipeline phases although these phases may not mean pipeline phases at that time. Of course i3 could be normal instruction without any extra latency to itself and its following instructions. For example if i3 is a conditional branch, and i4, i5, i6 are from the wrong destination, i4, i5, i6 must be abandoned and the pipeline must be refilled.

i1	F	D	E	M	W			
ts	F	Ts-D						
i2		F	D	E				
i3		F		D	E	M		
b1			F		D	E	M	
b2			F			D	E	W

Fig.3 Timing diagram for original architecture

i1	F	D	E	M	W				
i2	F		D	E	M	W			
i3			F	D	E*	M*			
i4			F*		D*	E*			
i5					F*	D*	E*	W*	
i6					F*		D*	E*	W*

Fig.4 the Timing diagram for Our Architecture

In Fig.4, because the Ts instruction is recognized in Ts-D stage, instructions following i3 are switched to another thread. We observe the execution time of 6 instructions in Fig.4 is at least no more than the execution time of 5 instructions in Fig.3. Equal condition meets only when the i3 doesn't cause any extra latency. But if the equal condition is not meet, the performance can be increased using the method of Fig.4.It shows while we add one more instruction (ts) to the instruction sequence, the throughput is increased in most cases and no decrease in any conditions. For example if i3 is a conditional branch instruction, we can use the St instruction to reduce the control dependence.

If the instruction tend to switch is i4, we can see the performance in the following Fig.5 and Fig.6.

i1	F	D	E	M	W				
i2	F		D	E	M	W			
i3		F		D	E	M	W		
i4		F			D	E*	M*	W*	
i5			F*			D*	E*	M*	W*
i6			F*				D*	E*	M*

Fig.5 the Timing Diagram for Original Architecture

i1	F	D	E	M	W				
i2	F		D						
ts		F	Ts-D						
i3		F		D	E				
i4			F		D	E	M	W	
i5			F						
b1				F		D	E	M	W
b2				F			D	E	W

Fig.6 the Timing Diagram for our Architecture

Similarly we can find the execution time of 7 instructions with Ts instruction in Fig.5 is at least no more than the execution time of 6 instructions in Fig.5 and the performance will be increased in most cases.

It should be noted that i5 will be abandoned. Figure.6.Because it is fetched simultaneous with an instruction tend to thread switch. i3 and i4 represent the typical relative positions of instructions in simultaneously fetching buffer. From our analysis, it can be conclude that our approach generally leads to better performance to switch at the switch point it arises abnormal latency. And our approach will not hurt the performance when switch happens at an arbitrary point. This feature is more meaningful for multithreading fetch policy in real-time multithreading embedded processors.

5. Conclusions and Future Work

In this paper, we proposed a novel thread switch approach by combining an augmenting thread switch

instruction with a support decoder. It provides a flexible thread switching mechanism as well as good performance for ARM/Thumb system. While the approach described here is implemented in the context of ARM architecture, they can be applied to other dual width embedded processors.

In the future, we will study further on the better combination of our mechanism with thread fetch policy and thread speculation techniques.

References:

[1] A. Krishnaswamy and R. Gupta,"Enhancing the Performance of 16-bit Code Using Augmenting Instructions," *ACM SIGPLAN Notices,* Vol.38 No.7 July 2003

[[2]Cui Guangzuo, "MT_ARM: multithreading Implementation in ARM7 Architecture," ASICON01, September, 2001, Shanghai, China

[[3]A. Krishnaswamy and R. Gupta, "Profile Guided Selection of ARM and Thumb Instructions," *ACM SIGPLAN Joint Conference on Languages Compilers and Tools for Embedded Systems & Software and Compilers for Embedded Systems(LCTES/SCOPES)*, Berlin, Germany, June 2002.

[4] Intel Corporation, "The Intel PXA250 Applications Processor, A White Paper," February 2002.

[[5] T.UNGERER, B.ROBIC, J.SILC. "A Survey of processors with explicit multithreading," *ACM Computing Surveys*, Vol. 35, No. 1, 2003, pp.29-63,

[6] SWANSON, S., MCDOWELL, L., SWIFT, M., EGGERS, S., AND LEVY, H. "An evaluation of speculative instruction execution on simultaneous multithreaded processors "*ACM Transactions on Computer Systems (TOCS)* archive Vol.21, No.3, 2003 pp.247-253