[Downloaded from www.aece.ro on Monday, June 30, 2025 at 22:05:22 (UTC) by 172.70.127.79. Redistribution subject to AECE license or copyright.]

Addressing Mode Extension to the ARM/Thumb Architecture

Dae-Hwan KIM
Department of Computer and Information, Suwon Science College,
Gyeonggi-do 445-742, Rep. of Korea
kimdh@ssc.ac.kr

***Abstract*—In this paper, two new addressing modes are introduced to the 16-bit Thumb instruction set architecture to improve performance of the ARM/Thumb processors. Contrary to previous approaches, the proposed approach focuses on the addressing mode of the instruction set architecture. It adopts scaled register offset addressing mode and post-indexed addressing mode from the 32-bit ARM architecture, which is the superset of the 16-bit Thumb architecture. To provide the encoding space for the new addressing modes, the register fields in the LDM and STM instructions are reduced, which are not frequently executed. Experiments show the proposed extension achieves an average of 7.0% performance improvement for the seven benchmark programs when compared to the 16-bit Thumb instruction set architecture.**

***Index Terms*—computer architecture, computer performance, high performance computing, instruction set design, microprocessors.**

I. Introduction

Embedded systems often demand compact code size due to their tight memory constraints. Compressed instruction set architecture (ISA) thus becomes one of solutions to such code size problem, and dual instruction set processors such as ARM/Thumb [1] and MIPS/MIPS16 [2] can execute both a normal 32-bit instruction set and a compressed 16-bit instruction set, which in fact is a subset of frequently used 32-bit instructions. The compressed 16-bit Thumb instructions are dynamically decompressed into 32-bit ARM instructions during the instruction decoding stage [3].

To balance performance and code density, the Thumb-2 architecture [4] is introduced which combines the traditional 16-bit Thumb instructions and additional 32-bit ARM instructions in a single instruction set. However, the 16-bit Thumb instruction set in the dual instruction set is still widely used for low-end embedded systems because it reduces memory footprint by 30% and accordingly power consumption compared to the 32-bit ARM mode [3]. The ARM/Thumb dual instruction set is adopted in various processors such as ARM7TDMI, ARM9TDMI, ARM926EJ-S, XScale, and ARM1176JZ(F)-S. Among ARM processors, the ARM7TDMI processor has been one of the most successful processors with hundreds of millions sold in almost every kind of microcontroller equipped products, and it still remains one of the most widely shipped ARM cores in 2013 [5].

To fit into the 16-bit encoding space, the 16-bit Thumb ISA reduces the number of both accessible registers and addressing modes. This reduction limits the number of 32-bit ARM instructions that can be converted into the 16-bit Thumb format, thereby reducing both compression efficiency and performance. To improve the 16-bit Thumb ISA, numerous techniques have been proposed [6-12]. Some techniques aim to enhance performance by increasing the number of available registers [6-9] as for the 32-bit ARM ISA [13-14], while others (e.g., [10]) attempt to further reduce the code size. However, comparatively little attention has been paid to possible improvements through extending addressing modes in the Thumb ISA. Fiskiran *et al.* [15] propose efficient addressing modes for AES algorithms, and simulate the algorithms on EPIC (Explicitly Parallel Instruction Computer) processors, but the approach requires a considerable amount of hardware circuitry.

To improve both performance and compression efficiency of the 16-bit Thumb ISA, this paper suggests a new instruction set architecture, AMEX16 (Addressing Mode EXtension to the 16-bit Thumb ISA) which adopts two efficient addressing modes, scaled register offset addressing mode and post-indexed addressing mode, from the 32-bit ARM architecture. To provide the encoding space for the proposed approach, the register fields in less frequently used instructions are reduced, and new addressing modes are introduced for more frequent instructions by capitalizing on the saved bits.

II. Related Work

Krishnaswamy and Gupta [6] propose a way to utilize the invisible registers in Thumb code. Most Thumb instructions can access only 8 registers out of 16 general-purpose registers. To eliminate this limitation, the proposed approach introduces a new mask instruction to represent the visible subset of 16 registers, which enables all instructions to use all registers. However, code size is considerably increased because too many new instructions are generated to change the visible registers.

In [7], Krishnaswamy and Gupta enhance the Thumb instruction set by incorporating Augmenting eXtensions (AX) to exploit the property that there exists a Thumb instruction pair that is equivalent to a single ARM instruction. In this approach, each AX instruction is combined with the immediately following Thumb instruction, and these instructions are converted to a single ARM instruction at the decode stage. This approach requires the coalescing hardware not to increase the cycle time.

Similarly, Lee *et al.* [8-9] reconstruct the original register file into the banked one, and provide a new bank change

Digital Object Identifier 10.4316/AECE.2014.02014

1582-7445 © 2014 AECE

[Downloaded from www.aece.ro on Monday, June 30, 2025 at 22:05:22 (UTC) by 172.70.127.79. Redistribution subject to AECE license or copyright.]

instruction, which makes all the registers available for register allocation. The region-based banked register allocation is additionally proposed to efficiently use the proposed structure.

To further reduce code size, Kwon *et al.* [10] propose a new instruction set architecture and a register allocation optimized for this architecture. The proposed approach reduces the destination register field and allocates the saved bit to the immediate filed. However, the practical compression efficiency is not clear because the proposed experiments are performed on a set of small benchmark programs.

Dong *et al.* [11] add the thread switch instruction to the 16-bit Thumb instruction set architecture to provide a flexible thread switching mechanism. The switch instructions are handled by the support of the extended decoder, resulting in no extra cycles.

Instead of enhancing the instruction set architecture, [12] proposes the program memory compression hardware architecture. The proposed approach achieves a code size reduction of around 15% on the 16-bit Thumb code by employing a lossless data compression algorithm for the code.

III. Instruction Set Design

The ARM architecture supports offset addressing mode which calculates the effective address by adding the offset value to the base address where the offset value can be specified in one of three ways: immediate, register, and scaled register. Among them, scaled register offset addressing mode is useful when accessing an array or vector because the value of a register can be scaled by the size of each element. Another efficient addressing mode, post-indexed addressing mode, is also provided where the base address in a register is used as the effective address and it is then automatically updated to the next memory location, which is useful to optimize program loops.

Fig. 1 shows the example of post-indexed addressing mode where Fig. 1 (a) and Fig. 1 (b) show the C program, and its corresponding assembly code, respectively. In Fig. 1 (a), array X is initialized with zero. The offset of array X increases by one at each loop iteration because the element size of the array is one. Assume that R3 and R1 hold zero and the start address of array X, respectively. Then, the store operation and the address computation for next element can be combined by the one store instruction with post-indexed addressing mode 'STRB R3, [R1], #1' as shown in Fig. 1(b). This instruction stores the content of R3 to the memory address of R1, and then automatically increases the value of R1 by one.

The Thumb ISA designers made trade-offs among various instruction fields such as opcode, register, and addressing mode, and sacrificed both scaled register offset addressing mode and post-indexed addressing mode. Therefore, in Thumb, an extra ADD or shift instruction is required to increase or scale the address, respectively.

This decision may not be optimal because these two missing addressing modes are more frequent than some instructions that use specific registers in many embedded programs. Remind that these two addressing modes are useful for accessing an array or data list. On the other hand, LDMIA (Load multiple) and STMIA (Store multiple) block transfer instructions are mainly used for memory block copy, whereas PUSH (Push registers) and POP (Pop registers) instructions are for procedure entry and return, and it is rare that the LDMIA and STMIA instructions transfer eight registers at a time. This is because most 16-bit Thumb instructions can only address eight registers, and thus a considerable number of them are likely to be used for other computations. These observations lead AMEX16 to adopt the two missing addressing modes at the cost of reduced accessible registers in some instructions such as the LDMIA and STMIA instructions. Due to the limited encoding space, AMEX16 can only cover the most common subtype for each addressing mode. For scaled register offset addressing mode, a left shift is selected among logical left and right shifts, arithmetic right shift, and right rotate operation. Similarly, immediate is chosen among immediate, register, and scaled register post-indexed addressing modes.

```
unsigned char X[100];
int i;

for (i = 0; i < 100; i++)
     X[i] = 0;
```

(a)

```
        LDR   R1, #start address of array X
        ADD   R2, R1, #100
        MOV   R3, #0
|loop|
        STRB  R3, [R1], #1
        CMP   R1, R2
        BLT   |loop|
```

(b)

Figure 1. Post-indexed addressing mode example: (a) C program, (b) Assembly program

There are very few unused opcodes available in Thumb. Thus, to generate encoding space for the new addressing modes, the proposed approach reduces register fields in the LDMIA and STMIA instructions, each of which is not frequently used and has a sufficient number of operand bits to embed new addressing modes therein. The LDMIA instruction loads a subset of the general-purpose registers from memory. The STMIA instruction stores a subset of the registers into memory. In each instruction, register R7 is excluded from the possible transfer list, resulting in one free bit. This bit is used to distinguish between a reduced Thumb mode (R7 bit=0) and a new AMEX16 mode (R7 bit=1).

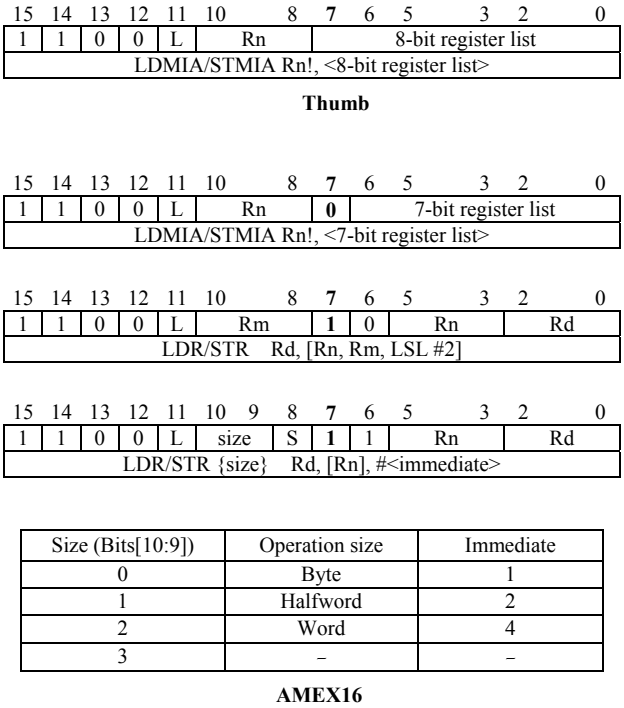
This modification is not expected to impose a burden on programmers because the reduction is to be limited to a couple of the least frequent instructions. Also, note that the Thumb ISA is originally designed as a compiler target and already not orthogonal [1]. The performance degradation is minor by the reductions because the reduced instructions are rarely used. On average, the LDM and STM instructions that use R7 only account for 0.0230%, and 0.0000029% of the total dynamic cycles, respectively. Details are discussed in Section IV.

The AMEX16 extension to 16-bit Thumb consists of two new addressing modes for the load and store instructions. Fig. 2 shows the modifications to the 16-bit Thumb LDMIA (L bit=1) and STMIA (L bit=0) instructions for AMEX16.

[Downloaded from www.aece.ro on Monday, June 30, 2025 at 22:05:22 (UTC) by 172.70.127.79. Redistribution subject to AECE license or copyright.]

In these two instructions, the transfer list is reduced from R0-R7 to R0-R6. The bit 7 discriminates the new addressing mode (bit 7=1) from the reduced Thumb instruction (bit 7=0). The bit 6 distinguishes between two addressing modes. Consider first the shifted register offset addressing mode (bit 6=0). AMEX16 restricts the shift operations to a 2-bit left shift, and adopts the LDR (Load word) and STR (Store word) instructions associated with this addressing mode. The L bit distinguishes the LDR and STR instructions. Note that the traditional ARM architecture such as version four supports this addressing for byte and word data types and the 2-bit left shift is the most frequent among them, which is useful to access an element of a word (32-bit) array. The destination, base, and offset registers are encoded in bits 2 to 0, bits 5 to 3, and bits 10 to 8, respectively.

The second is immediate post-indexed addressing mode (bit 6=1). Combined with this mode, the load and store instructions are introduced for byte, halfword, and word data types. This addressing mode increases the base register Rn by an immediate value, and the most frequent values are one, two, and four, which are useful to traverse byte, halfword, and word arrays, respectively. The S bit specifies whether a load is sign-extended (S=1) or zero-extended (S=0). The 2-bit size field specifies both the size of operation and the immediate value. The destination and base registers are encoded in bits 2 to 0 and bits 5 to 3, respectively.



Size (Bits[10:9])	Operation size	Immediate
0	Byte	1
1	Halfword	2
2	Word	4
3	–	–

Figure 2. AMEX16 instruction set design: Thumb LDMIA/STMIA instruction format and its corresponding AMEX16 instruction formats

IV. EVALUATION

Fig. 3 shows the performance efficiency of the proposed approach, AMEX16, relative to the 16-bit Thumb ISA. Experiments are performed on the FaCSim ARM simulator [16] targeting ARM9TDMI [17]. The simulator is extended to support the new AMEX16 instructions. The benchmarks are jpeg, mpeg, gzip, susan, gsm, adpcm, and blowfish programs. Jpeg and mpeg are two standards compression algorithms for still images and motion pictures, respectively, and gzip is a popular data compression program for the GNU project. Susan is structure preserving noise reduction image filter. Adpcm is one of the simplest forms of audio coding, and gsm is a standard speech codec. Blowfish is a symmetric block cipher with a variable length key. Jpeg, mpeg, and adpcm are taken from MediaBench [18], and susan, gsm, and blowfish are from Mibench [19], and gzip is from SPEC2000 [20]. Jpeg and adpcm also belong to Mibench. For each program, the ARM Realview compiler generates the 16-bit Thumb assembly code, which is post-processed and condensed into the AMEX16 code. The proposed approach shows performance improvement of an average of 7.0% compared to the 16-bit Thumb ISA. In the case of adpcm, the most significant improvement (10.8%) is achieved. This is because the program uses many LSL (Logical Shift Left) and ADD instructions for address computation that can be eliminated in AMEX16.

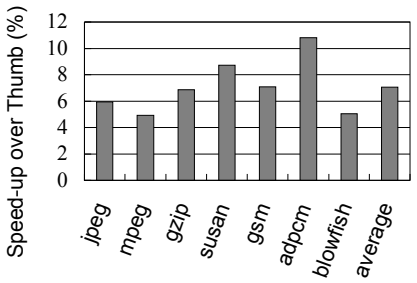


Figure 3. Speed-up of the proposed approach compared to 16-bit Thumb

Fig. 4 shows the frequency of new addressing modes which are introduced in AMEX16. Let lslMem and memInc denote the LDR and STR instructions with 2-bit left-shifted register offset addressing mode, and the proposed immediate post-indexed addressing mode, respectively. On average, lslMem, and memInc account for 3.0%, and 4.0% of the total execution cycles, respectively.

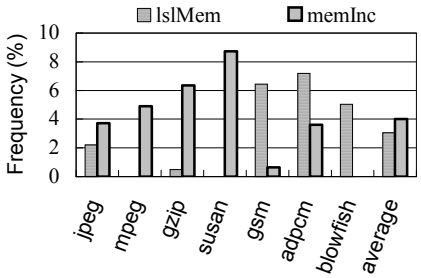


Figure 4. Frequency of the two proposed addressing modes

Fig. 5 shows the compression efficiency of AMEX16 compared to the Thumb. The efficiency of code size reduction improves by an average of 2.2%. This is because AMEX16 eliminates the LSL and ADD instructions in Thumb due to the missing addressing modes.

Table I shows the least frequently used instructions in Thumb for the benchmark programs. Total 17 instructions are shown whose dynamic frequency is less than 0.05% where the frequency is measured by the ARM Realview

[Downloaded from www.aece.ro on Monday, June 30, 2025 at 22:05:22 (UTC) by 172.70.127.79. Redistribution subject to AECE license or copyright.]

development suite. In the table, registers starting with H denote high registers R8-R15 whereas the others are in the range R0-R7. A few of the least frequent instructions need to be reduced to provide encoding space for the new addressing modes. In the instruction selection, the number of operand bits is first considered because each new addressing mode requires 10 bits for the operands and the AMEX16 mode as shown in Fig. 2. In this criterion, reduction candidates are three instructions: STMIA, LDMIA, and ADD immediate to PC (Program Counter). The STMIA instruction is first chosen in the frequency order, and then, the LDMIA instruction is selected instead of the ADD immediate instruction to make the reduced instructions symmetric. The performance degradation by this decision is a minor because the frequency difference is only 0.0387% between the LDMIA and ADD immediate instructions.

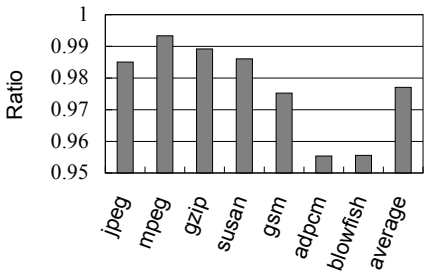


Figure 5. Compression efficiency of proposed approach compared to 16-bit Thumb

TABLE I. THE LEAST FREQUENT INSTRUCTIONS IN THUMB

Instruction		Frequency (%)	The number of operand bits
ADC/SBC/ROR	Rd, Rm	0.0000	6
TST/CMN	Rn, Rm	0.0000	6
ADD	Hd, Hm	0.0000	8
CMP	Hn, Hm	0.0000	8
STMIA	Rn!, <8-bit_reg. list>	0.0004	11
ADD	Rd, PC, #<8-bit imm.>	0.0005	11
CMP	Hn, Rm	0.0015	7
MOV	Hd, Hm	0.0019	8
BIC	Rd, Rm	0.0047	6
CMP	Rn, Hm	0.0076	7
SUB	Rd, Rn, #<3-bit imm.>	0.0122	9
LDMIA	Rn!, <8-bit_reg. list>	0.0402	11
LSR	Rd, Rs	0.0416	6
ASR	Rd, Rs	0.0419	6

Register R7, the last low register, is selected to be excluded in these two instructions because it is expected to be easy for the programmer to remember the number and accordingly use the reduced instructions. The other end register R0 may be the alternative selection, but this may complicate the program. Note that contrary to the singular usage of R7 for a variable, R0 is used for several purposes such as an argument, a return value, and a temporary in the register conventions. Table II shows the dynamic execution frequency of the two reduced instructions in Thumb. On average, the LDMIA and STMIA instructions that use register R7 only account for 0.0233% and 0.0000029% of the total dynamic cycles, respectively. Each instruction does not occur in six out of seven benchmark programs.

TABLE II. FREQUENCY OF LDMIA AND STMIA IN THUMB

Bench mark	Total cycles	LDMIA with R7 in the transfer list		STMIA with R7 in the transfer list	
		Cycles	Ratio (%)	Cycles	Ratio (%)
jpeg	9,790,443	0	0.000000	2	0.000020
mpeg	61,937,680	99,837	0.160000	0	0.000000
gzip	10,316,607	0	0.000000	0	0.000000
susan	37,402,371	0	0.000000	0	0.000000
gsm	395,051,939	0	0.000000	0	0.000000
adpcm	16,381,799	0	0.000000	0	0.000000
blowfish	20,897,219	0	0.000000	0	0.000000
average	-	-	0.023000	-	0.000029

V. CONCLUSION

In this paper, we realize the importance of the addressing mode, and extend the addressing mode of the 16-bit Thumb architecture, which enables performance improvement of an average of 7.0% compared to the 16-bit Thumb instruction set architecture. The proposed approach improves performance in a trade-off between the addressing mode extension and the reduced accessible registers in the less frequently used instructions in Thumb.

REFERENCES

[1] S. Segars, K. Clarke, and L. Goudge, "Embedded control problems, Thumb, and the ARM7TDMI ," IEEE Micro, vol. 15, no. 5, pp. 22-30, 1995.
[2] K. Kissell, MIPS16: High-Density MIPS for the Embedded Market, Silicon Graphics MIPS Group, Technical report, 1997.
[3] S. Furber. ARM system-on-chip architecture, Addison-Wesley, pp. 188-206, 2000.
[4] R. Phelan, Improving ARM Code Density and Performance, ARM Ltd., Technical report, June 2003.
[5] ARM Ltd., ARM Annual Report & Accounts 2013, 2014.
[6] A. Krishnaswamy and R. Gupta, "Efficient Use of Invisible Registers in Thumb Code," Proc. MICRO, 2005, pp. 30-42.
[7] A. Krishnaswamy, R. Gupta, "Dynamic coalescing for 16-bit instructions," ACM Transaction on Embedded Computing System, vol. 4, no. 1, pp. 3-37, 2005.
[8] J. H. Lee, S. M. Moon, and H. K. Choi, "Comparison of Bank Change Mechanisms for Banked Reduced Encoding Architectures," Proc. CSE Vol. 2, 2009, pp. 334-341.
[9] J. H. Lee, and J. Park, and S. M. Moon, "Securing More Registers with Reduced Instruction Encoding Architectures," Proc. RTCSA, 2007, pp. 417-425.
[10] Y. -J. Kwon, X. Ma, and H. J. Lee, "PARE: instruction set architecture for efficient code size reduction," IEE Electronics Letters, vol. 35, no. 24, pp. 2098-2099, 1999.
[11] L. Dong, Z. Ji, G. Gui, and M. Hu, "Multithreading extension for Thumb ISA and decoder support," Proc. EHAC, 2006, pp. 78-81.
[12] X. Xu, C. Clarke, and S. Jones, "High performance code compression architecture for the embedded ARM/THUMB processor," Proc. CF, 2004, pp. 451–456.
[13] H.-H. Chiang, H.-J. Cheng, and Y.-S. Hwang, "Doubling the Number of Registers on ARM Processors," Proc. INTERACT, 2012, pp. 1-8.
[14] H.-J. Cheng, Y.-S. Hwang, R.-G. Chang, and C.-W. Chen, "Trading Conditional Execution for More Registers on ARM Processors," Proc. EUC, 2010, pp. 53-59.
[15] A. M. Fiskiran and R. B. Lee, "Performance Impact of Addressing Modes on Encryption Algorithms," Proc. ICCD, 2001, pp. 542-545.
[16] J. Lee, J. Kim, C. Jang, S. Kim, B. Egger, K. Kim, and S. Han, "FaCSim: A Fast and Cycle-Accurate Architecture Simulator for Embedded Systems," Proc. LCTES, 2007, pp. 89-100.
[17] ARM Ltd., ARM9TDMITM Technical Reference Manual, 2000.
[18] C. Lee, M. Potkonjak, and H. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems," Proc. MICRO, 1997, pp. 330-335.
[19] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "Mibench: A free, commercially representative embedded benchmark suite," Proc. IISWC, 2001, pp. 3-14.
[20] J. L. Henning, "SPEC CPU 2000: Measuring CPU performance in the new millennium," IEEE Computer, vol. 33, no. 7, pp. 28-35, 2000.