Addressing Mode and Bit Extensions to the Thumb-2 Instruction Set Architecture

Dae-Hwan Kim

***Abstract* — Thumb-2 is the most recent instruction set architecture for ARM processors which are one of the most widely used embedded processors. In this paper, two extensions are proposed to improve the performance of the Thumb-2 instruction set architecture, which are addressing mode extensions and sign/zero extensions combined with data processing instructions. To speed up access to an element of an aggregated data, the proposed approach first introduces three new addressing modes for load and store instructions. They are register-plus-immediate offset addressing mode, negative register offset addressing mode, and post-increment register offset addressing mode. Register-plus-immediate offset addressing mode permits two offsets and negative register offset allows offset to be a negative value of a register content. Post-increment register offset mode automatically modifies the offset address after the memory operation. The second is the sign/zero extension combined with a data processing instruction which allows the result of a data processing operation to be sign/zero extended to accelerate a type conversion. Several least frequently used instructions are reduced to provide the encoding space for the new extensions. Experiments show that the proposed approach improves performance by an average of 8.6% when compared to the Thumb-2 instruction set architecture.**

***Index Terms* — Instruction Set Architecture; Thumb-2; ARM; Addressing Mode; Sign/zero Extension; Embedded Processor.**

I. INTRODUCTION

ARM processor is one of the most widely used embedded processors, which is adopted in smartphones, tablets, laptops, consumer electronics, and various electronic devices. It is estimated that about 180 billion ARM-based chips have been shipped until 2020, including the 6.7 billion chips in the fourth quarter of 2020 [1].

In the embedded systems, small code size is often required because the memory size directly impacts on the cost of the system. To solve this problem, dual instruction set processors such as ARM/Thumb [2] and MIPS/MIPS16 [3] are presented which provides two separate instruction sets which are a normal instruction set and a compressed instruction set. The compressed instructions are a subset of the frequently used normal instructions, and they are decompressed into normal instructions during the instruction decoding stage [2]. Contrary to the separate instruction sets, Thumb-2 architecture [4] is proposed for ARM Cortex processors to meet both high performance and small code size in which a single instruction set architecture provides both the 16-bit Thumb instructions and additional 32-bit instructions which can be freely intermixed with each other.

Researches have been made to improve the ARM/Thumb instruction set architecture [5]-[27]. However, little attentions have been paid to the extension of specific modes such as addressing mode and sign/zero extension for the ARM instruction architectures. To improve performance and compression efficiency of the Thumb-2 instruction set architecture (ISA), this paper suggests a new instruction set architecture, named ABEX (Addressing mode extension and sign/zero Bit EXtension combined with data processing), which introduces efficient addressing modes for load and store instructions, and the sign/zero extension combined with data processing to accelerate a type conversion.

The rest of this paper is organized as follows. Section II provides the review of the related works. Section III presents the proposed instruction set design, and Section IV provides experimental results. Conclusions are presented in Section V.

II. RELATED WORK

Numerous approaches have been proposed to enhance the 32-bit ARM ISA. The ARM DSP extension [5] introduced in ARMv5 adds new DSP instructions to the ARM instruction set to accelerate signal processing applications. It supports the 16-bit multiplication instruction and the saturated add and subtract instructions, which provides up to 70% performance improvement in audio applications. This extension is incorporated in various processors such as ARM926EJ-S, ARM946E-S, and ARM966E-S.

The SIMD (Single Instruction Multiple Data) instructions in ARMv6 simultaneous operate on two 16-bit or four 8-bit values packed in a 32-bit register. More than 60 instructions are added which mainly targets multimedia applications, and the SIMD extension achieves 75% performance improvement for audio and video processing applications. This architecture is implemented in the ARM1136J(F), ARM1156T2(F), ARM1176JZ(F), and ARM11 MPCore processors.

The ARMv7 architecture employs the advanced SIMD extension called NEON [6] for the ARM Cortex-A processors, which provides flexible and powerful accelerations for multimedia and signal processing applications such as video codec, speech and audio codec, graphics, and image processing. The NEON instructions can operate on vectors stored in the 64-bit double word and 128-

Submitted on February 28, 2021.
Published on March 22, 2021.
Dae-Hwan Kim, Department of Computer and Electronics Engineering, Suwon Science College, Rep. of Korea.
(e-mail: kimdh@ssc.ac.kr).

bit quad word vector registers.

The ARMv8 architecture [7]-[8] introduces a 64-bit architecture, named AArch64, and a new A64 instruction set to the existing instruction set to support the 64-bit operation and the virtual addressing. ARMv8 supports two execution states, AArch64 for 64-bit operation and AAarch32 for 32-bit operation. A64 has 31 64-bit general-purpose registers. This architecture includes instruction-level cryptographic instructions while dropping LDM/STM and most conditional execution instructions.

Programs often manipulate data at subword level in the embedded system. Li and Gupta [10] propose a bit section extension to manipulate subword data which occur in network and multimedia applications. The new instructions can reduce explicit instructions for packing and unpacking narrow width data into memory words.

Increasing the number of physical registers would improve performance, but this requires additional bits to encode the register number. To maintain the width of 32-bit ARM instructions, references [11]-[12] observe that the conditional field is underutilized, and thus, trade the conditional field for the register field in the instruction, and use the 4-bit conditional field to encode the extra registers, which allows the number of physical registers to be doubled from 16 to 32 without increasing code size.

Bos et al. [13] propose the parallel computation model of the Montgomery multiplication to improve performance in the public-key cryptography applications. They evaluate the proposed approach on the ARM platform with NEON technology and show that it is necessary to incorporate new 256-bit SIMD instructions with two-way integer multipliers in the instruction set.

Erich et al. [14] evaluate ARM Cortex-M0+, ATmega, and MSP430 microprocessors in runtime, chip area, power, and energy characteristics for elliptic curve cryptography. The results show that the Cortex-M0+ is the fastest and most energy efficient processor. They present the resource-saving scalar multiplication algorithm, and integrate instruction set extensions for these processors.

Murray et al. [15] improve the efficiency of the instruction set extension generation. The proposed approach integrates the exploration of source-level code transformations and the identification of instruction set extensions. The proposed framework is based on the Intel XScale processor which is the Intel's implementation of the ARMv5 architecture. The approach improves performance by 49% on two benchmark suites.

Other techniques focus on enhancing the 16-bit Thumb instruction set architecture. In 16-bit Thumb, most instructions can reference only 8 registers out of 16 physical registers. Krishnaswamy and Gupta [17] address this inefficiency and introduce a new register mask set instruction to specify the visible set of registers. With the use of the mask instruction, every instruction can access all the registers. However, many mask instructions are emitted to change the visible subset of registers, resulting in the increase of code size.

Krishnaswamy and Gupta introduce augmenting extensions [18] to coalesce consecutive instructions and convert those instructions to a single 32-bit ARM instruction during the decode stage. Because some pairs of Thumb instructions are equivalent to a single ARM instruction, the compiler replaces patterns of Thumb instructions by the equivalent sequences of augmented Thumb instructions. Each augmented instruction is coalesced with the following non-augmented Thumb instruction in the decode stage which is redesigned to detect augmenting instructions and perform coalescing to generate the ARM instructions.

Lee et al. [19]-[20] construct the original register file into the banked one and provide a new bank change instruction for register allocation, and additional register allocation technique for this banked structure.

Kim [21]-[22] proposes the addressing mode extension to the 16-bit Thumb architecture [21], and two addressing modes to the Thumb-2 architecture [22]. On the EPIC processors, Fiskiran et al. [23] present the addressing modes for the AES algorithms. However, it demands a considerable amount of hardware circuitry.

To reduce code size without performance degradation, Canedo et al. [24] propose the queue-based reduced instruction set and the code generation algorithm optimized for the proposed instruction set. The proposed approach can generate 26% more compact code when compared to ARM/Thumb without reducing the parallelism in the program.

III. Instruction Set Design

In the offset addressing mode in the Thumb-2 architecture, an effective address is calculated as the sum of an offset value and the address in a base register. Offset can be specified in one of three types which are immediate value, the content of a register, a scaled value in a register, but the combination of offsets is not allowed. This degrades performance when accessing an elements of array structures. Another restriction is that negative register offset addressing mode is not supported which subtracts an offset value in a register from the address in a base register when forming the effective address.

In addition to offset addressing mode, Thumb-2 also provides post-indexed addressing mode where a base address is used as an effective address, which can be updated to the next memory location. However, this addressing mode is not efficient for consecutive memory addressing in which an offset value is required to be modified after each memory operation.

These limitations often lead to an extra ADD or SUB instruction to compute or update the effective address when accessing elements in an aggregated data such as an array. To reduce these limitations, ABEX introduces three new addressing modes, which are named in this paper register-plus-immediate offset addressing mode, post-increment register offset addressing mode, and negative register offset addressing mode. Register-plus-immediate offset addressing mode permits both a register offset and an immediate offset where the effective address is the sum of a base address and two offsets. Post-increment register offset addressing mode combines register offset addressing mode and post-indexed addressing mode in Thumb-2, which calculates the effective address as the sum of the base address and the offset address, and automatically modifies the offset address after the memory operation. Negative register offset addressing mode subtracts an offset value in a register from the base address when forming an effective address.

In addition to these new addressing modes, ABEX provides the ability to sign/zero extension for the result of the data processing operation. The Thumb-2 architecture already provides distinct sign/zero-extend instructions such as SXTB (Signed eXTend Byte) and extend-and-add instructions such as UXTAB (Unsigned eXTend and Add Byte) and SXTAH (Signed eXTend and Add Halfword). ABEX introduces sign/zero extension to the results of ADD, SUB, AND, ORR, ORN, EOR, BIC, ADC, SBC, and RSB operations.

Almost all the opcodes are already used in Thumb-2. Therefore, it is required to reduce several instructions to give space for the new extensions. Three unfrequently used instructions are selected which has a sufficient number of operand bits. They are the ADD immediate to PC (Program Counter), LDMIA (Load multiple) and STMIA (Store multiple) instructions. The ADD immediate to PC (Program Counter) instruction, normally denoted by ADR (ADdress to Register), adds an immediate value to the PC value, and writes the result to the destination register. The LDMIA and STMIA instructions transfer data between memory and a subset of the general-purpose registers. In each instruction, the destination register field, or the register list field is reduced by one bit, resulting in one free bit. This bit is used to distinguish between a reduced Thumb-2 instruction and a new ABEX instruction.

The performance degradation by this modification is minor by the reductions because the reduced instructions are not frequently used. On average, the restricted ADR, LDMIA, and STMIA instructions that use disabled registers only account for 0.000115%, 0.000118%, and 0.000118% of the total dynamic cycles, respectively. Details are discussed in Section 4.

Fig. 1 shows the modifications to the 16-bit ADR instruction format in Thumb-2 to support two addressing modes and sign/zero extended data processing instructions. In the ADR instruction format, the size of a destination register field is reduced from three to two, whereby saving bit 10. This bit is zero for the original ADR instruction and one for the ABEX instructions. Thumb-2 enforces 16-bit alignment on all instructions. Thus, the 32-bit Thumb-2 instruction is treated as two halfwords, hw1 (first halfword) and hw2 (second halfword) where hw1 is at the lower address whereas the 16-bit instructions have only first halfword. The bit 9 of hw1 specifies the new instruction type, which is zero for the new addressing modes and one for the extended data processing operations. The bit 11 of hw2 distinguishes between two addressing modes negative register offset addressing mode and post-increment offset addressing mode. In post-increment offset addressing mode, bits 10 to 8, and bits 7 to 4 of hw2 specify the left shift amount, and the post-increment immediate value, respectively. For the extended data processing instructions, bits 7 to 4 of hw1 and bits 15 to 14 of hw2 encode the data operation named DOP, and the extension operation named EOP, respectively. Bits 13 to 12 of hw2 identify the second operand type which can be register, scaled register, or immediate. For the scaled register type, bits 5 to 4 of hw2 encode the shift type which is one of LSL (Logical Shift Left), LSR (Logical Shift Right), ASR (Arithmetic Shift Right), and ROR (Rotate Right). Two bits in bits 7 to 6 encode the shift amount.

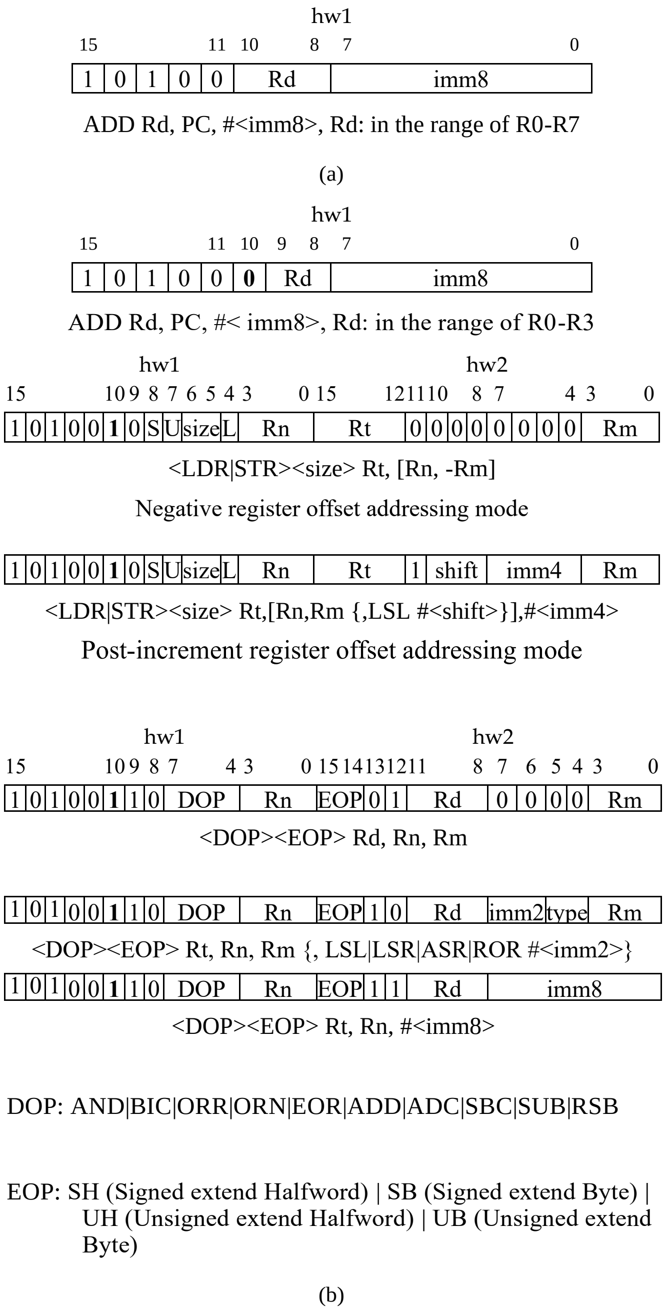
hw1
15 11 10 8 7 0
1 0 1 0 0 Rd imm8
ADD Rd, PC, #<imm8>, Rd: in the range of R0-R7
(a)
hw1
15 11 10 9 8 7 0
1 0 1 0 0 0 Rd imm8
ADD Rd, PC, #< imm8>, Rd: in the range of R0-R3
hw1 hw2
15 10 9 8 7 6 5 4 3 0 15 12 11 10 8 7 4 3 0
1 0 1 0 0 1 0 S U size L Rn Rt 0 0 0 0 0 0 0 0 0 Rm
<LDR|STR><size> Rt, [Rn, -Rm]
Negative register offset addressing mode
1 0 1 0 0 1 0 S U size L Rn Rt 1 shift imm4 Rm
<LDR|STR><size> Rt,[Rn,Rm {,LSL #<shift>}],#<imm4>
Post-increment register offset addressing mode
hw1 hw2
15 10 9 8 7 4 3 0 15 14 13 12 11 8 7 6 5 4 3 0
1 0 1 0 0 1 1 0 DOP Rn EOP 0 1 Rd 0 0 0 0 Rm
<DOP><EOP> Rd, Rn, Rm
1 0 1 0 0 1 1 0 DOP Rn EOP 1 0 Rd imm2 type Rm
<DOP><EOP> Rt, Rn, Rm {, LSL|LSR|ASR|ROR #<imm2>}
1 0 1 0 0 1 1 0 DOP Rn EOP 1 1 Rd imm8
<DOP><EOP> Rt, Rn, #<imm8>
DOP: AND|BIC|ORR|ORN|EOR|ADD|ADC|SBC|SUB|RSB
EOP: SH (Signed extend Halfword) | SB (Signed extend Byte) | UH (Unsigned extend Halfword) | UB (Unsigned extend Byte)
(b)
Fig. 1. Thumb-2 ADR instruction format and its corresponding ABEX instruction format: (a) Thumb-2 format, (b) ABEX format.

Fig. 2 shows the modifications to the 16-bit LDMIA and STMIA instructions where the transfer list is reduced from R0-R7 to R0-R6 by excluding register R7. The excluded bit is used for register-plus-immediate offset addressing mode for the load and store instructions. The bit 7 of hw1 is zero for the 16-bit LDMIA/STMIA instruction and one for the new ABEX instructions, which are 32-bit wide. ABEX allocates 8 bits to the immediate field similar to the Thumb-2 immediate instructions, which is encoded in bits 11 to 4 of hw2. The A bit distinguishes between ADD (A=1) immediate and SUB (A=0) immediate. Bits 11 to 10 of hw1 determine the scale factor for the index register, Rm. Among various shift types and amounts, 1-bit, 2-bit, and 3-bit left shifts are supported where 1-bit and 2-bit left shifts are useful for accessing 16-bit and 32-bit arrays, respectively. Three registers are encoded in the same places in the Thumb-2 ISA.

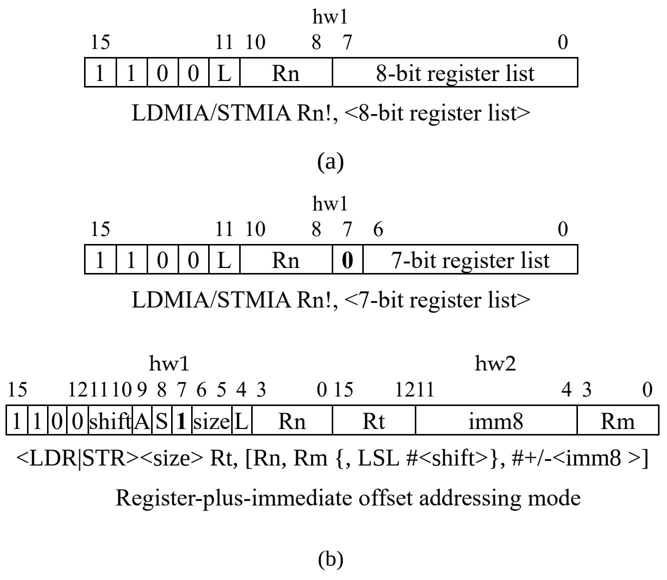
hw1
15 11 10 8 7 0
1 1 0 0 L Rn 8-bit register list
LDMIA/STMIA Rn!, <8-bit register list>
(a)
hw1
15 11 10 8 7 6 0
1 1 0 0 L Rn 0 7-bit register list
LDMIA/STMIA Rn!, <7-bit register list>
hw1
hw2
15 12 11 10 9 8 7 6 5 4 3 0 15 12 11 4 3 0
1 1 0 0 shift A S 1 size L Rn Rt imm8 Rm
<LDR|STR><size> Rt, [Rn, Rm {, LSL #<shift>}, #+/-<imm8 >]
Register-plus-immediate offset addressing mode
(b)

Fig. 2. Thumb-2 LDM/STM instruction format and its corresponding ABEX instruction format: (a) Thumb-2 format, (b) ABEX format.

PSR (Program Status Register) can be used to distinguish between ABEX mode and Thumb-2 mode if needed. Four bits, bits 23 to 20, are unused in the register, and one of them such as bit 23 can be used to indicate the ABEX extension, which is one for ABEX mode and zero for Thumb-2 mode.

Consider the hardware circuitry to support ABEX. One 32-bit adder is required for register-plus-immediate offset addressing mode and post-increment register offset addressing mode. The new post-extended data processing requires a sign/zero extension logic to be connected to result operand of a data processing operation, MUX, and control logics.

IV. Evaluation

Fig. 3 shows the performance of the proposed approach, ABEX, compared to the Thumb-2 ISA. Experiments are performed on the FacSim simulator [28] targeting the ARM Thumb-2 Cortex-M3 processor [29]. The benchmarks are gzip, susan, pegwit, adpcm, blowfish, and stringsearch programs. Gzip is a GNU zip program. Susan is the noise reduction image filter program. Pegwit performs the public-key encryption and authentication. Adpcm is a simple audio codec. Blowfish is a symmetric block cipher, and stringsearch, denoted by strsearch in this paper, searches for given words in phrases. Gzip is from SPEC2000 [30], Susan, adpcm, blowfish, and strsearch are from Mibench [31]. For each program, Thumb-2 assembly code generated by the compiler is post-processed and compacted into ABEX code. The speedup is calculated by the ratio of the number of total cycles of Thumb-2 code and that of ABEX code. In the six benchmark programs, performance is improved by an average of 8.6% compared to the Thumb-2 ISA. The most significant improvement is achieved in the Strsearch program which uses many pairs of LOAD and ADD instructions each of that can be combined into a single LOAD instruction with post-increment register offset addressing mode.

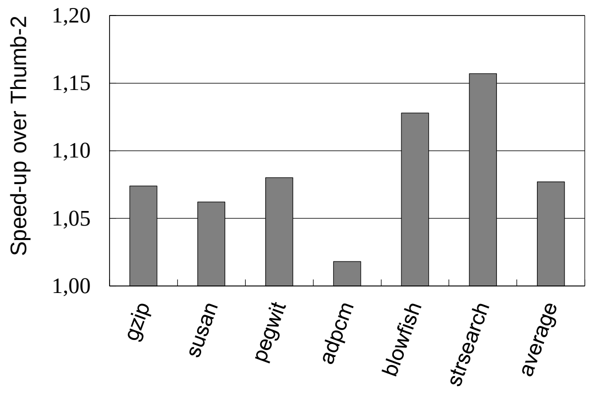

Fig. 3. Speed-up of ABEX compared to Thumb-2.

Fig. 4 shows the frequency of the ABEX instructions. Let reg+imm, -reg, and post+ denote register-plus-immediate addressing mode, negative register offset addressing mode, and post-increment register offset addressing mode, respectively, and let extData be new sign/zero extension combined with data processing instructions. On average, reg+imm, -reg, post+, and extData account for 2.5%, 0.8%, 2.5%, and 1.1% of the total execution cycles, respectively.

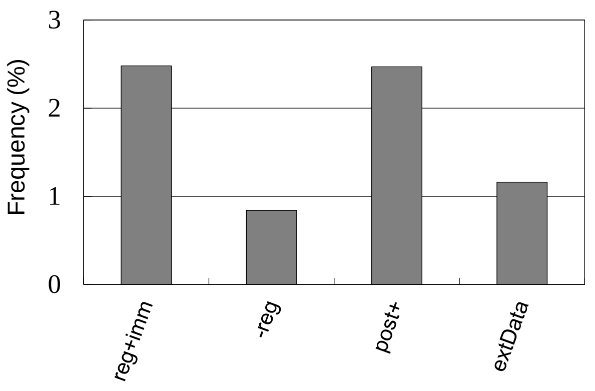

Fig. 4. Frequency of the proposed instructions.

Fig. 5 shows the compression efficiency of ABEX compared to Thumb-2. The code size is reduced by an average of 2.9%. This is because ABEX eliminates the ADD, SUB, LSL, and separate sign/zero extension instructions that are required in Thumb-2 code to update the address due to the missing addressing modes and post extension mode.

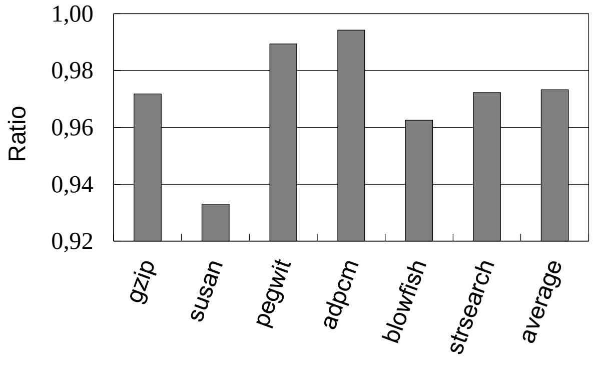

Fig. 5. Compression efficiency of ABEX.

Fig. 6 shows the execution frequency of the register field reduced that use the disabled registers in instructions in the proposed approach. Register R7 is disabled in the LDM and STM instructions, and four registers, R4, R5, R6, and R7, are excluded in the ADR instruction. On average, the ADR, LDMIA, and STMIA instructions that use disabled registers only account for 0.000115%, 0.000118%, and 0.000118% of the total dynamic cycles, respectively, and this shows the proposed restriction is minor in the performance for benchmark programs.

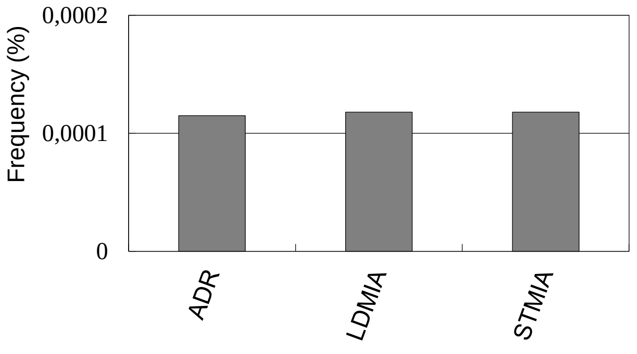

Fig. 6. Frequency of the ADR, LDMIA, and STMIA instructions that use disabled registers.

V. Conclusion

In this paper, the addressing mode and sign/zero bit extensions are proposed for Thumb-2 instruction set architecture, which improves both compression efficiency and performance of the architecture. This improvement requires additional small amount of hardware circuitry while the number of accessible registers is reduced in the several less frequently used instructions.

The NEON technology, the ARM advanced SIMD extensions, is beneficial for digital signal processing and multimedia algorithms. It remains as a future work to evaluate and improve the NEON instruction set architecture.

References

[1] https://www.businessweekly.co.uk/news/hi-tech/record-67bn-arm-chips-shipped-single-quarter-just-start, 2021.

[2] S. Segars, K. Clarke, and L. Goudge, "Embedded control problems, Thumb, and the ARM7TDMI," *IEEE Micro*, Vol. 15, No. 5, pp.22-30, 1995.

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

[4] R. Phelan, "*Improving ARM Code Density and Performance*," Technical Report, ARM Ltd., 2003.

[5] F. Hedley, *ARM DSP-Enhanced Extensions*, ARM Ltd., 2001.

[6] ARM Ltd., *Introducing NEON TM Development Article*, ARM Ltd., 2009.

[7] ARM Ltd., *ARMv8 Instruction Set Overview*, ARM Ltd., 2012.

[8] ARM Ltd., *ARM Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*, ARM Ltd., 2013.

[9] ARM Ltd., Introduction to Armv8.1-M architecture. 2019.

[10] B. Li, and R. Gupta, "Bit Section Instruction Set Extension of ARM for Embedded Applications," In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, Grenoble, France, pp.69-78, 2002.

[11] H. -J., Cheng, Y. -S. Hwang, R. -G. Chang, and C. -W. Chen, "Trading Conditional Execution for More Registers on ARM Processors," In *Proceedings of the 8th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC)*, Hong Kong, China, pp.53-59, 2010.

[12] H. -H. Chiang, H. -J. Cheng, and Y. -S. Hwang, "Doubling the Number of Registers on ARM Processors," In *Proceedings of the 16th Workshop on Interaction between Compilers and Computer Architectures (INTERACT-16)*, Louisiana, USA, pp.1-8, 2012.

[13] J. W. Bos, P. L. Montgomery, D. Shumow, G. M. Zaverucha, "Montgomery Multiplication Using Vector Instructions," *IACR Cryptology ePrint Archive*, pp. 519-535, 2013.

[14] W. Erich, U. Thomas, W. Mario, "8/16/32 Shades of Elliptic Curve Cryptography on Embedded Processors", In *Proceedings of the 14th International Conference on Cryptology in India*, Mumbai, India, pp.244-261, 2013.

[15] A. C. Murray, R. V. Bennett, B. Franke, N. Topham, "Code transformation and instruction set extension". *ACM Transactions on Embedded Computing Systems*, Vol. 8, No. 4, pp. 1-31, 2009.

[16] J. Goodacre, A. N. Sloss, "Parallelism and the ARM instruction set architecture," *IEEE Computer*, Vol. 38, No. 7, pp.42-50, 2005.

[17] A. Krishnaswamy, R. Gupta, "Efficient Use of Invisible Registers in Thumb Code", In *Proceedings of the 38th IEEE/ACM International Symposium on Microarchitecture*, Barcelona, Spain, pp.30-42, 2005.

[18] A. Krishnaswamy, R. Gupta, "Dynamic coalescing for 16-bit instructions," *ACM Transaction on Embedded Computing System*, Vol. 4, No. 1, pp. 3-37, 2005.

[19] J. H. Lee, J. Park, S. M. Moon, "Securing More Registers with Reduced Instruction Encoding Architectures", In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Vol. 2, Washington, pp.417-425, 2007

[20] J. H. Lee, S. M. Moon, H. K. Choi, "Comparison of Bank Change Mechanisms for Banked Reduced Encoding Architectures", In *Proceedings of the International Conference on Computational Science and Engineering* Vol. 2, Canada, pp.334-341, 2009.

[21] D.-H. Kim, "Addressing Mode Extension to the ARM/Thumb Architecture", *Advances in Electrical and Computer Engineering*, Vol.14, No. 2, pp.85-88, 2014.

[22] D.-H. Kim, S.-W. Kim, "Extending Offset Addressing Mode and Post-Indexed Addressing Mode of Thumb-2 Instruction Set Architecture", *The Journal of Korean Institute of Next Generation Computing*, Vol.9, No.6, pp.6-14, 2013.

[23] A. M. Fiskiran, R. B. Lee, "Performance Impact of Addressing Modes on Encryption Algorithms", In *Proceedings of the International Conference on Computer Design*, Austin, Texas, pp.542-54, 2001.

[24] A. Canedo, B. A. Abderazek, S. Masahiro, "Compiling for Reduced Bit-Width Queue Processors," *Journal of Signal Processing Systems*, Vol. 59, No. 1, pp. 45-55, 2010.

[25] S. Flur, K. Gray, C. Pulte, S. Sarkar, A. Sezgin, L. Maranget, W. Deacon, and P. Sewell, "Modelling the ARMv8 architecture, operationally: concurrency and ISA," *ACM SIGPLAN Notices*, pp. 608-621 2016.

[26] A. Akram, "A Study on the Impact of Instruction Set Architectures on Processor's Performance," M.S. Thesis, Western Michigan University, 2017.

[27] B. Simner, S. Flur, C. Pulte, A. Armstrong, J. Pichon-Pharabod, et al., "ARMv8-A system semantics: instruction fetch in relaxed architectures," In *Proceedings of 29th European Symposium on Programming (ESOP)*, Mar 2020.

[28] 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," In *Proceedings of the International Conference on Languages, Compilers, and Tools for Embedded Systems*, Tucson, Arizona, USA, pp. 89-100, 2007.

[29] ARM Ltd., *Cortex-M3 technical reference manual*, ARM Ltd., 2010.

[30] J. L. Henning, "SPEC CPU 2000: Measuring CPU performance in the new millennium," *IEEE Computer*, Vol. 33, No. 7, pp. 28-35, 2000.

[31] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "Mibench: A free, commercially representative embedded benchmark suite", In *Proceedings of the 4th IEEE International Workshop on the Workload Characterization*, Austin, TX, USA, pp.3-14, 2001.