반도체디스플레이기술학회지 제23권 제3호(2024년 9월)
Journal of the Semiconductor & Display Technology, Vol. 23, No. 3. September 2024.

고성능 임베디드 디바이스를 위한 RV32IMC명령어 확장을 지원하는 RISC-V 파이프라인 프로세서 설계 및 구현

박경우[*] · 심현진[*] · 김선희[*] · 김용우[**†]
[*†]상명대학교 시스템반도체공학과, [**]한국교원대학교 기술교육과

Design and Implementation of RISC-V Pipeline Processor Supporting RV32IMC Instruction Extensions for High-Performance Embedded Devices

Kyeongwoo Park[*], Hyeonjin Sim[*], Sunhee Kim[*] and Yongwoo Kim[†]**
[*]Department of System Semiconductor Engineering, Sangmyung University,
[**†]Department of Technology Education, Korea National University of Education

ABSTRACT

Recent research on embedded systems has become increasingly important due to their central role in high-performance embedded devices, including artificial intelligence, autonomous driving, and energy management technologies. Embedded systems are specialized computer systems designed to perform specific tasks while optimizing performance and minimizing memory usage. RISC-V, an open RISC-based instruction set architecture developed by the University of Berkeley in 2010, is well-suited for these systems. In addition to the base 32-bit integer instruction set, RISC-V supports extensions such as the M-extension for multiplication and division and the C-extension for instruction compression. In this paper, we propose the design of a 32-bit 5-stage pipeline RV32IMC processor aimed at high-performance embedded devices. By incorporating the RV32IMC instruction set, the proposed processor achieves enhanced computational efficiency and reduced code size, making it a strong candidate for energy-efficient, high-performance embedded applications. Furthermore, the design was validated on an Artix-7 field-programmable gate array, demonstrating the processor's feasibility and potential benefits for embedded systems.

Key Words : RISC-V, Processor, RV32IMC, 5-stage pipeline, FPGA

1. 서 론

최근 인공지능, 자율주행, 에너지 관리 기술 등이 적용된 고성능 임베디드 디바이스의 핵심인 임베디드 시스템 관련 연구가 여러 분야에서 진행되고 있다. 임베디드 시스템은 특정 작업을 위해서 하드웨어를 소프트웨어에 맞게 최적화할 수 있으며, 성능에 영향을 끼치지 않으면서, 최소한의 메모리를 사용하는 것이 중요하다[1]. 이에 적합한 프로세서로 2010년 버클리 대학에서 개발한 RISC(Reduced Instruction Set Computer) 계열의 무료 개방형 ISA(Instruction Set Architecture)인 RISC-V가 대표적이다. RISC-V는 32-bit 뿐만 아니라 64-bit와 128-bit 기반으로도 구분이 되며, 32-bit 정수형 명령어 집합 "I" 외에도 곱셈 및 나눗셈을 지원하는 "M", 16-bit의 압축 명령어를 지원하는 "C", 부동소수점 관련 명령어를 지원하는 "F" 등 사용자의 필요에 따라 명

[†]E-mail: yongwoo.kim@knue.ac.kr

령어 확장을 통해 프로세서 설계가 가능하다[2].

본 연구에서는 고성능 임베디드 디바이스를 위한 RV32IMC 명령어 확장을 지원하는 5단계 파이프라인 프로세서를 설계한다. 곱셈 및 나눗셈을 지원하는 M 확장 명령어, 압축 명령어를 지원하는 C 확장 명령어를 지원하기 위해서 각각의 필요한 기능과 모듈을 추가하여 설계한다. Artix-7 NEXYS A7-100T FPGA(Field Programmable Gate Array) 보드를 통해 Dhrystone[3] 및 Coremark[4] 벤치마크를 실행하고, 동작을 검증한다. 또한 이를 통해 얻어지는 DMIPS(Dhrystone Millions of Instructions Per Second), Coremark 점수와 설계한 프로세서의 ISA에 따른 GCC 컴파일러로 생성된 코드 크기에 대한 비교 및 평가를 수행한다.

본 연구의 구성은 다음과 같다. 2장에서는 RISC-V 프로세서의 설계에 관한 연구들에 대해 설명한다. 3장에서는 제안하는 RV32IMC 프로세서 설계 및 구조를 설명한다. 4장에서는 제안하는 프로세서의 동작 검증 및 기존의 프로세서들의 성능 비교를 수행하며 5장에서 결론 및 추후 연구에 대해 설명한다.

2. 관련 연구

2.1 RISC-V 프로세서 활용

RISC-V ISA를 기반으로 인공지능, 기계학습, 임베디드 시스템, 보안 등 여러 분야에서 연구가 진행되고 있다. BARVINN[5]은 임베디드 시스템에 적용될 수 있는 저정밀 DNN(Deep Neural Network) 하드웨어 가속기로, RISC-V를 기반으로 한 맞춤형 내장 코어를 포함한다. 이는 DNN 가속기 어레이를 관리하는 역할로, DNN 연산을 효율적으로 제어할 수 있음을 보인다.

Ibex[6] 프로세서는 32-bit RISC-V 코어로 정수형 I 또는 임베디드 E 그리고 정수 곱셈 및 나눗셈 M, 압축 C 및 비트 조작 B 확장 명령어들을 지원한다. 이를 통해 고성능의 코어 역할을 하며, 임베디드 디바이스에 적합하다. 또한 다양한 하드웨어 구성 옵션과 그에 대한 검증을 통해 사용자의 필요에 맞춰 프로세서 구현이 가능함을 보인다.

비 순차적 실행 프로세서인 BOOM(Berkeley Out-of-Out Machine)[7]은 Chisel(Constructing Hardware in a Scala Embedded Language) 하드웨어 구성 언어를 이용하여 고성능, 합성 및 매개변수화 가능한 코어를 만드는데 중점을 두고 설계된 RV64IMAFD RISC-V 코어이다. 그러나 해당 프로세서는 압축 명령어를 지원하는 C 확장이 지원되지 않기 때문에 코드 크기가 커지게 되면서 전력 소모가 크다는 단점을 가진다.

Chipyard[8][9]는 Berkeley 대학에서 Chisel 기반 SoC(System of Chip)의 개발을 위한 오픈 소스 프레임워크이다. Rocket [10], BOOM[7], CVA6(Ariane) [11] 같은 프로세서 코어와, Gemmini[12]와 같은 가속기, 메모리 시스템 및 주변 장치를 포함하여 개발한 통합 설계, 시뮬레이션 및 구현 프레임워크이다.

SCR1[13]은 Syntacore에서 면적과 전력에 최적화되도록 설계한 RISC-V 코어로 RV32I 또는 RV32E ISA를 기반으로 하여 사용자의 필요에 따라 RVM이나 RVC ISA를 지원한다. 해당 코어는 2에서 4단계의 파이프라인 단계 구조로 이루어져 있으며, 인터럽트 컨트롤러와 32-bit의 AXI4(Advanced eXtensible Interface 4)/AHB-Lite(Advanced High-performance Bus Lite) 외부 인터페이스를 지원한다.

PicoRV32[14]는 소형화를 중점으로 두고 단일 사이클(single-cycle)로 대부분의 명령어를 처리하는 구조이며, RISC-V RV32IMC ISA를 지원한다. 또한 하드웨어를 간단하게 구성할 수 있도록 설계되어 인터럽트 모듈을 선택할 수 있다. 그러나 명령어를 단일 사이클로 처리했을 때 명령어 처리 속도가 느리다는 단점을 가진다.

3. 제안하는 32-bit RISC-V RV32IMC

Fig. 1은 제안하는 RV32IMC 프로세서 구조를 나타낸다. Fig. 1에서 볼 수 있듯이 5단계 파이프라인 구조의 32-bit정수형 명령어를 지원하는 RV32I의 구조[15]를 기본 구조로 설계한다. 5단계 파이프라인 구조는 메모리에서 명령어를 가져오는IF (Instruction Fetch) 단계, 명령어를 해석하는 ID (Instruction Decode) 단계, 연산을 수행하는EXE (Execution) 단계, 데이터 메모리에 데이터를 쓰거나 읽는 MEM (Memory) 단계, 데이터를 레지스터 파일에 쓰는WB (Write Back) 단계로 구분이 된다. 제안한 RV32IMC 프로세서에서는 32-bit 곱셈 및 나눗셈 명령어 그리고 16-bit의 압축 명령어들을 지원하기 위한 모듈과 기능이 추가된다.

본 장에서는 5단계 파이프라인으로 구성된 RV32IMC 프로세서의 구조와 M확장 및 C확장 명령어를 위한 설계 방법을 설명한다.

3.1 제안하는 5 단계 파이프라인 RV32IMC 프로세서 구조

RV32IMC는 FENCE, ECALL, EBREAK 명령어들이 포함되지 않은 37개의 정수형 명령어, 곱셈 및 나눗셈을 지원하는 8개의 명령어와 단정밀 부동소수점 관련 명령어를 제외한 압축 명령어를 지원하는 26개의 명령어를 합쳐 총 71개의 명령어를 지원한다.

Fig. 2 (a)에서 볼 수 있듯이 곱셈 및 나눗셈 명령어를 지원하기 위해 EXE와 MEM 단계에 모듈과 기능이 추가되는 것을 나타낸다. 이러한 처리는 ID 단계의 Control Unit에

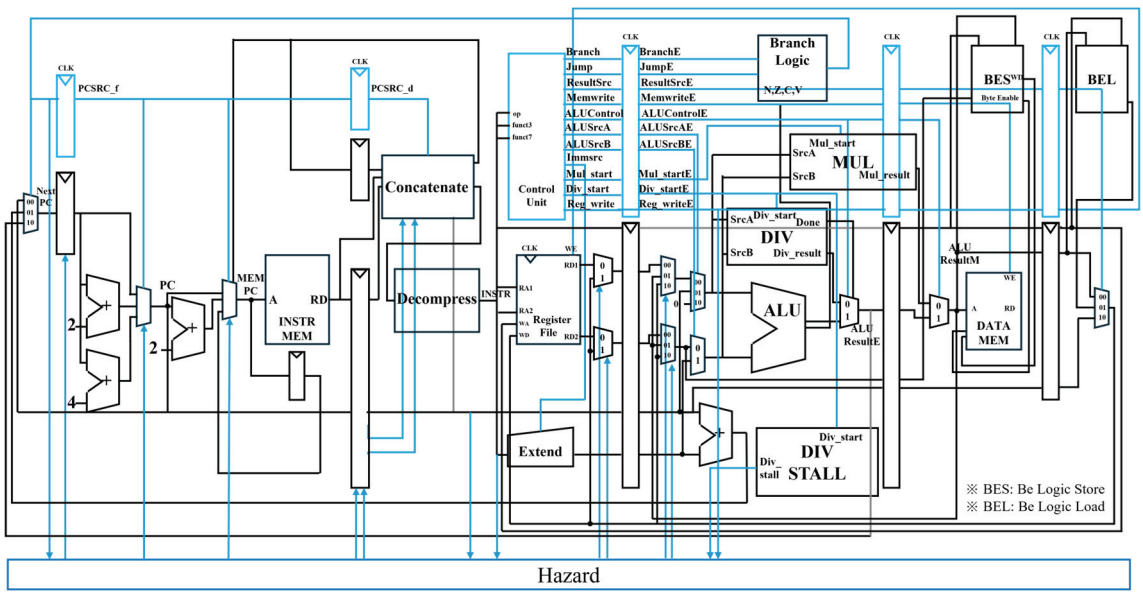

Fig. 1. The five-stage pipeline processor for supporting RV32IMC ISA.

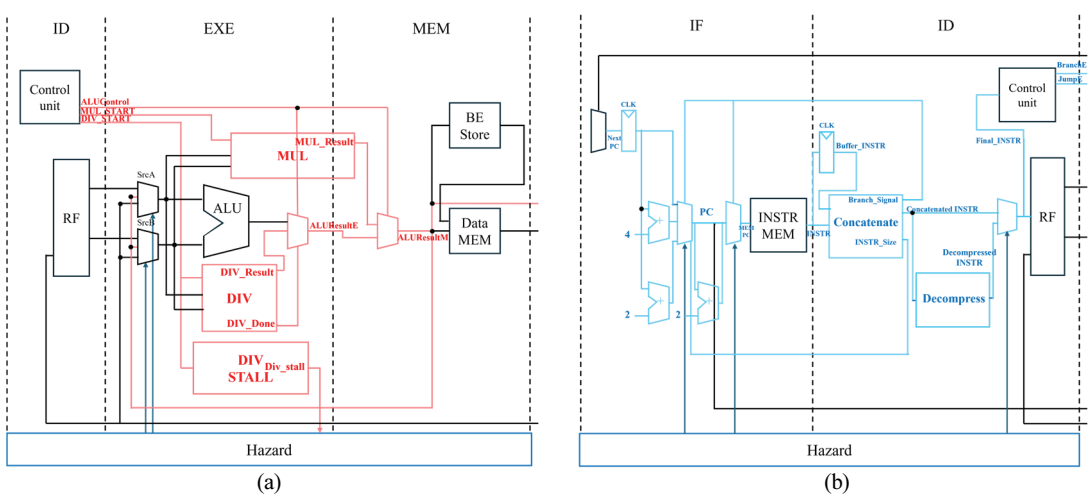

Fig. 2. Proposed RV32IMC processor Datapath highlighting a) M extension, b) C extension.

서 나오는 신호를 활용하여 곱셈 및 나눗셈 연산을 수행한다. 곱셈 명령어를 지원하기 위해서 기존 RV32IM 프로세서[16]를 참고하며, 나눗셈 명령어를 지원하기 위해서 비복원 알고리즘[17]을 사용한 나눗셈기를 사용한다.

Fig. 2 (b) 에서 볼 수 있듯이 압축 명령어를 지원하기 위해서는 IF 및 ID 단계에 모듈과 기능이 추가되는 것을 나타낸다. 이를 위해 IF 단계에서는 명령어의 길이와 분기 및 지연 여부를 판단하여 PC(Program Counter) 값을 결정한다. ID 단계에서 Concatente 모듈은 버퍼 데이터에 저장된 값과 메모리 데이터에 저장된 값을 통해 명령어를 반환한다. Decompress 모듈은 반환된 명령어가 16-bit 압축 명령어일 경우 정해진 형식[18]에 따라 32-bit 명령어로 확장하여 동일한 프로세서에서 작동하도록 한다.

3.2 제안하는 곱셈 및 나눗셈 명령어를 위한 설계 방법

본 연구에서 곱셈 명령어를 처리하기 위해 기존 RV32IM 프로세서에서 사용한 Xilinx의 1단계 파이프라인 DSP IP기

반의 곱셈기와, non-stalling 기법을 사용한다. Fig. 2 (a) 에서 볼 수 있듯이 non-stalling 기법과 1단계 파이프라인 단계를 가지는 곱셈기를 이용하여 곱셈의 연산 결과 값을 MEM 단계에서 결정한다.

나눗셈 명령어의 경우에는 Fig. 2 (a)에서 볼 수 있듯이 EXE 단계에서 나눗셈의 연산 결과 값을 결정한다. 나눗셈 연산은 32 사이클이 소요되기 때문에 DIV_STALL을 통해서 나눗셈 연산에 필요한 신호들은 유지시키고, IF 및 ID단계의 신호들을 지연시킨다. 이 과정에서 기존의 RV32IM 프로세서에서는 복원 알고리즘을 사용하여 나눗셈기를 사용한다. 그러나 복원 알고리즘은 비복원 알고리즘에 비해 실행 시간이 더 오래 걸리고 더 많은 면적을 필요로 하는 단점 때문에, 비복원 알고리즘을 이용하여 나눗셈기를 설계한다.

3.3 제안하는 압축 명령어를 위한 설계 방법

본 연구에서는 32-bit 명령어뿐만 아니라 16-bit 명령어도 처리한다. 이를 위해 Next_PC 값을 계산할 때 현재 PC에 2 또는 4를 더하는 것을 고려해야 하며, 분기 주소와 현재 PC를 유지하는 추가적인 고려사항이 존재한다. Next_PC 값을 결정하기 위해서 IF 단계에서 ID 단계의 명령어 길이를 확인하고, MEM 단계에서의 분기 및 지연 여부를 판단한다.

16-bit의 압축 명령어를 지원함에 따라, 메모리에는 Aligned PC 형태로만 들어오는 것이 아닌 Not Aligned PC 형태로도 들어온다. Fig. 3은 Not Aligned PC에서 32-bit의 C라는 명령어가 들어오는 상황을 예로 나타낸다. 이때 C라는 명령어를 가져오기 위해서는 두 번의 명령어 접근이 필요하므로, 첫 번째 메모리 접근에서 나온 값을 버퍼 데이터에 저장한다[19]. 이로 인한 버퍼 지연을 해결하기 위해 IF 단계에서 MEM-PC에 PC + 2값을 할당함으로써 문제를 해결한다. 그러나 분기 시Not Aligned PC의 32-bit의 명령어를 가져오는 상황이라면 버퍼 데이터에는 분기 명령어에 대한 데이터가 아닌 다른 데이터가 저장된다. 이 경우, 두번의 메모리 접근이 필요하다. 처음에는 PC 주소로 메모리 접근을 하여 버퍼 데이터에 저장을 하고 다음 메모리 접근은 PC+2를 사용하여 접근한다.

Fig. 2 (b)에서 보여지는 Concatenate 모듈은 저장된 버퍼 데이터와 메모리 데이터의 값을 통해 32-bit의 명령어로 변환하는 기능을 한다. Aligned PC의 명령어일 경우 하위 2-bit를 통해 명령어 길이를 판단하고, 메모리 데이터에서 명령어를 가져온다. 반면, Not Aligned PC의 명령어일 경우 명령어의 길이는 버퍼 데이터의 [17:16] bit에 따라서 판단한다. Fig. 4(a)에서와 같이 Not Aligned PC의 명령어가 32-bit일 경우, 메모리 데이터의 하위 16-bit와 버퍼 데이터의 상위 16-bit를 합쳐서 명령어를 가져온다. Fig. 4(b)에서와 같이, Not Aligned PC의 명령어가 16-bit인 경우, 버퍼 데이터의 상위 16-bit를 통해 명령어를 가져온다.

이렇게 Concatenate 통해 반환된 명령어는 Fig. 2 (b)에서 나타낸 Decompress를 통해서 처리된다. Decompress 모듈은 명령어의 길이를 판단하고 16-bit의 명령어 일 경우 32-bit의 명령어로 확장하여 반환하는 기능을 한다.

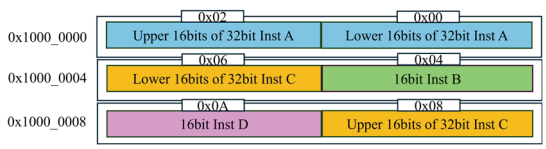

Fig. 3. The example of the 32-bit instruction at the Not Aligned PC.

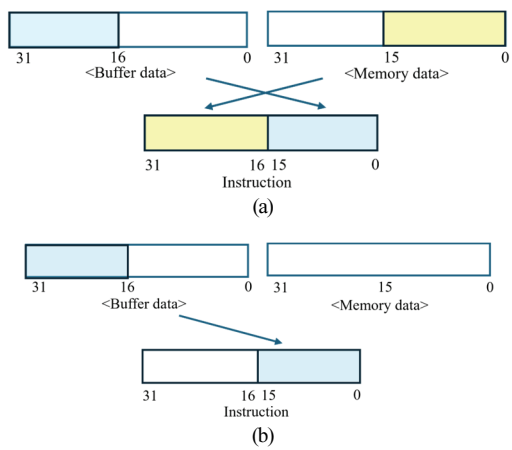

Fig. 4. The block diagram of proposed Concatenate in a) 32-bit instruction b) 16-bit instruction at the Not Aligned PC.

4. 동작 검증 및 성능 평가

본 장에서는 제안하는 32-bit RISC-V 5 단계 파이프라인 RV32IMC 프로세서의 동작을 검증하고 성능을 측정하여 다른 프로세서와 비교 및 평가한다. 동작 검증과 성능측정을 위해서 Xilinx의 Artix-7 NEXYS A7-100T 기반 FPGA 보드를 사용하며, Vivado 2019.02 버전을 통해서 합성 및 구현을 한다. Table 1은 제안하는 RV32I, RV32IM, RV32IC, RV32IMC 프로세서의 합성 및 구현 결과를 보여주며, 로직 사용량, 전력 소모, 최대 동작 주파수를 나타낸다.

본 연구에서는 설계된 프로세서의 성능 측정 및 평가를 위해 Dhrystone 및 Coremark 벤치마크를 실행한다. Fig.5는 제안하는 RV32IMC 프로세서가 적용된 Coremark 벤치마크 결과를 UART(Universal Asynchronous Receiver and Trans-

mitter)를 통해서 나타낸다. Dhrystone 및 Coremark 벤치마크 모두 GCC 8.2.0 버전을 이용하여 컴파일하고, 최적화 옵션은 O3를 적용한다. 반복횟수는 Dhrystone 벤치마크는 500번, Coremark 벤치마크는 1,000번으로 설정한다.

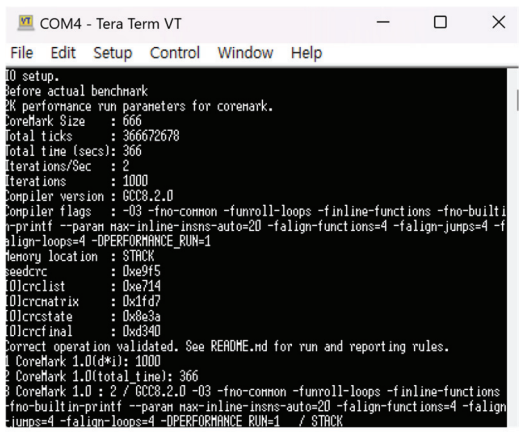

Fig. 5. The Coremark Benchmark result of proposed RV32IMC processor.

Table 2는 제안하는 프로세서들과 기존 프로세서들의 벤치마크 점수를 나타낸다. Table 3은 제안하는 프로세서들이 적용된 Coremark핵사 파일들의 코드 크기를 나타낸다. 먼저 제안하는 RV32IM 프로세서의 DMIPS/MHz 및 Coremark/MHz가 기존의 RV32IM 프로세서인full 옵션의 VexRiscv[20]에 비해 각각 2.48%, 20.43% 높다. 또한 제안하는 RV32IMC 프로세서의 DMIPS/MHz가 PicoRV32[14] 비해 137.25% 높으며, Coremark/MHz의 경우 Zero-riscy[21]에 비해 13.52% 높다.

Table 1. Hardware implementation results of proposed processors.

Resource	Utilization			
	RV32I	RV32IM	RV32IC	RV32IMC
LUT	1,961 (3.09%)	2,662 (4.20%)	2,041 (3.22%)	2,773 (4.37%)
FF	1,640 (1.29%)	1,945 (1.53%)	1,586 (1.25%)	1,839 (1.45%)
BRAM	32 (23.70%)	32 (23.70%)	32 (23.70%)	32 (23.70%)
DSP	0(0%)	4(1.67%)	0(0%)	4(1.67%)
Power	0.169W	0.208W	0.168W	0.204W
Max Frequency	52.77 MHz	50.93 MHz	52.49 MHz	50.42 MHz

PicoRV32[14]는 소형화를 중점으로 두고 단일 사이클로 설계된다. Zero-riscy[21]는 곱셈 및 나눗셈 지연에 각각 1사이클, 32 사이클을 소요된다. full 옵션의 VexRiscv[20]는 곱셈 및 나눗셈 지연에 각각 2사이클, 34사이클을 소요된다. 제안하는 프로세서들은 5단계 파이프라인 구조이다. 곱셈 지연은 없고, 나눗셈 지연은 32 사이클 소요된다. 따라서 기존의 프로세서들에 비해 더 나은 성능을 나타내는 것으로 판단된다. ARM Cortex-M33[22]에 경우 3단계 파이프라인 구조로 설계되었으며, 곱셈, 나눗셈, 압축 명령어뿐만 아니라 부동소수점 명령어를 지원한다. 이러한 ARM Cortex-M33[22] 비해 제안하는 RV32IMC 프로세서의 DMIPS/MHz 및 Coremark/MHz가 각각 19.33%, 31.09% 낮은 결과를 보인다.

Table 2. Benchmark results of the proposed processors and conventional processors

Processor	ISA	Pipeline Stage	DMIPS /MHz	Coremark /MHz
Pico-RV32[14]	RV32IMC	1	0.51	-
VexRiscv full[20]	RV32IM	5	1.21	2.30
Zero-riscy[21]	RV32IMC	2	-	2.44
Cortex-M33[22]	Armv8-M Mainline	3	1.50	4.02
Proposed RV32I	RV32I	5	1.17	0.98
Proposed RV32IM	RV32IM	5	1.24	2.77
Proposed RV32IC	RV32IC	5	1.14	0.97
Proposed RV32IMC	RV32IMC	5	1.21	2.73

Table 3. Coremark hex files code size of proposed RISC-V RV32I, RV32IM, RV32IC, RV32IMC.

Proposed Processor	Code Size
RV32I	75.0KB
RV32IM	73.0KB
RV32IC	63.8KB
RV32IMC	62.8KB

Table 2에서 제안하는 RV32IMC의 DMIPS/MHz 및 Coremark/MHz는 제안하는 RV32IM에 비해 각각 2.42%, 1.44% 낮은 결과를 보인다. 또한, Table 3에서는 제안하는 RV32IMC의 코드 크기는 RV32I, RV32IM, RV32IC에 비해 각각16.27%, 13.97%, 1.57% 줄어든다. 제안하는 RV32IMC의 벤치마크 결과가 C확장으로 인해서 추가된 기능 및 모듈 때문에 제안하는 RV32IM 보다 낮은 것으로 판단된다. 또한, 코드 크기 측면에서는 M과 C를 포함한 RV32IMC ISA를 지원하기에 가장 작은 것으로 판단된다.

본 연구에서 제안하는 RV32IMC 프로세서의 벤치마크 결과가 Cortex-M33[22]보다 낮은 이유는 부동소수점 명령어를 지원하지 않기 때문인 것으로 판단된다. 따라서 부동소수점 연산을 지원하는 확장 명령어 F를 추가로 구현한다면 성능이 향상될 것으로 예상된다.

5. 결 론

본 연구에서는 성능 향상과 코드 크기 최적화를 목표로, 곱셈 및 나눗셈 명령어를 지원하는 M 확장과, 압축 명령어를 지원하는 C 확장을 갖춘 프로세서를 설계한다. NEXYS A7-100T FPGA보드를 통해 합성 및 구현을 하며, Dhrystone과 Coremark 벤치마크와 코드 크기를 통해 제안하는 프로세서의 성능을 평가한다. 추후 연구로 부동소수점 연산을 지원하는 확장 명령어 F를 구현하여 성능 향상시킬 예정이다.

감사의 글

다음의 성과는 과학기술정보통신부와 연구개발특구진흥재단이 지원하는 과학벨트지원사업으로 수행된 연구결과입니다.

참고문헌

1. J. Ramanujam, J. Hong, M. Kandemir, A. Narayan, and A. Agarwal, “Estimating and reducing the memory requirements of signal processing codes for embedded systems”, IEEE, vol.54, pp. 286–294, 2006.
2. A. Waterman, K. Asanovi, and RISC-V International, “The RISC-V instruction set manual, Volume I: User-Level ISA, document version 20191213”, 2019.
3. R. P. Weicker, “Dhrystone: A synthetic systems programming benchmark”, Communications of the ACM, Vol.27, No.10, pp.1013-1030, 1984.
4. EEMBC, “CoreMark — CPU Benchmark – MCU Benchmark”, https://www.eembc.org/coremark/.
5. M. Askarihemmat, S. Wagner, O. Bilaniuk, Y. Hariri, Y. Savaria, J. David, “BARVINN: Arbitrary Precision DNN Accelerator Controlled by a RISC-V CPU”, Retrieved December, 31, 2022, from https://arxiv.org/abs/2301.00290.
6. lowRISC, ibex. Retrieved May, 3, 2024, from https://github.com/lowRISC/ibex.
7. riscv-boom, riscv-boom. Retrieved March, 18, 2024 from https://github.com/riscv-boom/riscv-boom.
8. A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, P. Rigge, C. Schmidt, J. Wright, J. Zhao, Y. Sophia Shao, K. Asanovic, B. Nikolic, “Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs”, IEEE, vol.40, pp. 10–21, 2020.
9. ucb-bar, chipyard. Retrieved May, 7, 2024, from https://github.com/ucb-bar/chipyard.
10. chipsalliance, rocket-chip. Retrieved May, 13, 2024, from https://github.com/chipsalliance/rocket-chip
11. openhwgroup, cva6. Retrieved May, 14, 2024, from https://github.com/openhwgroup/cva6
12. ucb-bar, gemmini. Retrieved May, 11, 2024, from https://github.com/ucb-bar/gemmini
13. syntacore, scr1. Retrieved May, 7, 2024, from https://github.com/syntacore/scr1.
14. YosysHQ, PicoRV32. Retrieved March, 27, 2024, from https://github.com/YosysHQ/picorv32.
15. S. Jo, J. Lee, Y. Kim, “A Design and Implementation of 32-bit Five-Stage RISC-V Processor Using FPGA”, Journal of the Semiconductor & Display Technology, vol. 21, no. 4, pp. 27–32, 2022.
16. S. Park, Y. Kim, “A Design and Implementation of 32-bit RISC-V RV32IM Pipelined Processor in Embedded Systems”, Journal of the Semiconductor & Display Technology, vol. 22, no. 4, pp. 81–86, 2023.
17. U. S. Patankar, A. Koel, "Review of basic classes of dividers based on division algorithm", IEEE, vol. 9, pp. 23035 - 23069, 2021.
18. A. Waterman, Y. Lee, D. Patterson, K. Asanovi, “The RISC-V Compressed Instruction Set Manual, document version 20151205”, 2015.
19. T. Kanamori, H. Miyazaki, K. Kise, “RVCoreP-32IC: A high-performance RISC-V soft processor with an efficient fetch unit supporting the compressed instructions”, Retrieved November, 23, 2020, from https://arxiv.org/abs/2011.11246.
20. SpainHDL, VexRiscv. Retrieved November, 17, 2023, from https://github.com/SpinalHDL/VexRiscv.
21. P. Davide, F. Conti, D. Rossi, M. Gautschi, A. Pullini, E. Flamand, L. Benini “Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications” 2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), pp.1-8, 2017.
22. Arm Limited Arm Cortex-M Processor Comparison Table. Retrieved October, 5, 2022, from https://developer.arm.com/documentation/102787/0100.

접수일: 2024년 5월 30일, 심사일: 2024년 9월 5일,
게재확정일: 2024년 9월 12일