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

메모리 효율성을 높이기 위한 압축 명령어를 지원하는 32-비트 파이프라인 RISC-V프로세서 설계 및 구현

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

A Design and Implementation of 32-bit Pipeline RISC-V Processor Supporting Compressed Instructions for Memory Efficiency

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

ABSTRACT

With the development of technologies such as the Internet of Things (IoT) and autonomous vehicles, research is being conducted on embedded processors that meet high performance, low power, and memory efficiency. The "C" expansion of the RISC-V processor is required to increase memory efficiency. In this paper, we propose an RV32IC processor and compare the benchmark performance score of the RV32I processor with the code size generated by the GCC compiler. In addition, we propose memory access and combination methods to support 16-bit compression commands, and command extension methods. The proposed RV32IC processor satisfies the maximum operating frequency of 50 MHz on the Artix-7 FPGA. The performance was checked using the benchmark programs of the Dhrystone and Coremark, and the code sizes of the RV32I and RV32IC generated by the GCC compiler were compared. The proposed processor RV32IC decreased DMIPS/MHz by 2.72% and Coremark/MHz by 0.61% compared to RV32I, but Coremark's code size decreased by 14.93%.

Key Words : RISC-V, Processor, Compressed Instruction, RV32I, RV32IC, 5-stage pipeline, FPGA

1. 서 론

최근 IoT(Internet of Things), 자율주행차와 같은 기술들의 발전으로 인해 고성능, 저전력, 메모리 효율성을 충족시키는 임베디드 프로세서의 연구가 진행되고 있다. 기존의 프로세서는 CISC (Complex Instruction Set Computer)방식과 RISC (Reduced Instruction Set Computer)방식을 사용해오고 있으며 이 중 임베디드 프로세서의 요구 조건을 만족시키는 것은 RISC 방식 프로세서이다[1].

[†]E-mail: yongwoo.kim@knue.ac.kr

RISC 기반의 ISA(Instruction Set Architecture)를 사용하는 프로세서는 대표적으로 ARM과 RISC-V가 존재하지만 ARM은 상업용 마이크로프로세서에서 사용되고 있는 반면 RISC-V는 무료 개방형 ISA로 인해 다양한 분야에서 활용되고 있다. RISC-V는 32, 64, 128bit 기반으로 구분되고 기본적으로 “I”로 명명된 정수형을 지원하며 추가로 표준 정수 곱셈 및 나눗셈 확장을 지원하는 “M”, 표준 단정밀 부동소수점 확장을 지원하는 “F”, 표준 압축 ISA확장을 지원하는 “C”등의 명령어 확장을 지원한다[2].

본 연구에서는 메모리 효율성 향상을 위해 16비트 압축 명령어를 지원하는 5단계 파이프라인 기반 RV32IC 프

로세서를 설계하였다. 프로세서의 성능을 검증하기 위해 Artix-7 Nexys A7-100T FPGA (Field Programmable Gate Array) 보드를 통해 Dhrystone[3] 벤치마크 점수와 Coremark[4] 벤치마크 점수를 확인하였고 메모리 효율성을 확인하기 위해 GCC 컴파일로 생성된 RV32I[5]와 RV32IC의 코드사이즈에 대한 비교를 진행하였다.

본 연구의 구성은 다음과 같다. 2장에서는 “C” 확장을 통해 지원하는 명령어 소개와 “C” 확장으로 인해 발생하는 문제를 해결한 관련 연구에 대해 설명한다. 3장에서는 RV32IC 프로세서의 구조와 2장에서 제시된 문제 해결 방법을 구현하기 위한 추가적인 설계 방법을 설명한다. 4장에서는 제안하는 RV32IC 프로세서의 동작을 검증하고 선행 연구된 프로세서와의 성능을 비교하며 5장에서는 결론 및 추후 연구에 대해 설명한다.

2. 관련 연구

RISC-V에서16비트 압축 명령어를 지원하는 “C” 확장을 통해 프로그램 내 RISC-V 명령어의 50~60%를 16비트 압축 명령어로 대체할 수 있으며 이를 통해 코드 크기를 25~30% 줄일 수 있다. 16비트 압축 명령어는 32비트 명령어와 혼합하여 사용 가능하며 32비트 명령어도 16비트 경계에서 시작 가능하다[6].

Table 1. Compressed 16-bit RVC instruction formats[6]

Format	Meaning	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CR	Register	funct4				rd/rs1					rs2					op	
CI	Immediate	funct3			imm	rd/rs1					imm					op	
CSS	Stack-relative Store	funct3			imm						rs2					op	
CIW	Wide Immediate	funct3			imm								rd			op	
CL	Load	funct3			imm			rs1			imm		rd			op	
CS	Store	funct3			imm			rs1			imm		rs2			op	
CB	Branch	funct3			offset			rs1			offset					op	
CJ	Jump	funct3			jump target											op	

Table 1은 압축 명령어의 8가지 형식에 대해 보여준다. CR, CI, CSS 형식은 레지스터에 32비트 명령어의 모든 레지스터를 사용할 수 있지만 CIW, CL, CS, CB 형식은 32비트 명령어의 8개 레지스터만 사용할 수 있다. 정수형 명령어에서 사용할 수 있는 8개 레지스터는 x8~x15번 레지스터이다. 또한, CI, CIW, CSS 형식에서는 기본 주소 레지스터로 스택 레지스터를 사용하는 명령어가 존재하며 이 명령어 이름에는 "SP"가 붙는다[6].

RV32IC는 32비트 정수형 명령어와 16비트 압축 명령어를 취급하는 프로세서로 모든 명령어는 메모리의 32비트 경계와 16비트 경계에서 시작할 수 있다. “C” 확장으로 달라진 메모리 구조로 인해 새로운 명령어 패치 방법이 필요하다. Fig. 1 (a)와 (b)는 동일한 명령어들을 32비트 메모리와 16비트 메모리에 할당한 구조를 보여주며 메모리에 따른 명령어 패치 방법을 설명한다.

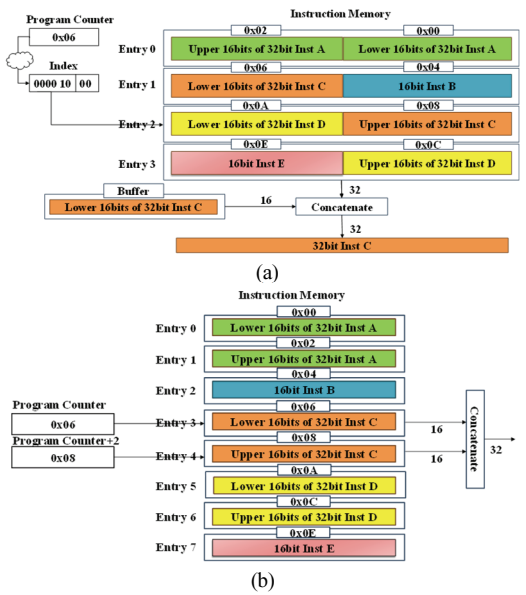

Fig. 1. (a) The buffering instruction fetch architecture, (b) The instruction fetch with two program counters into 16-bit wide instruction memory [7].

Fig. 1 (a)는 32비트 메모리에서 Not Aligned (PC=2,6,A,E) 주소의 32비트 명령어인 명령어 C를 가져오기 위해 메모리에 두 단계로 접근하여 생기는 IPC (Instruction Per Cycle) 감소에 대한 해결 방법으로 버퍼를 사용한 방법을 보여준다. 버퍼를 사용하는 방법은 일반적으로 메모리에 두 단계로 접근하는 경우에 많이 사용되며 PULP Platform[8] 프로세서에서는 FIFO 버퍼, VexRiscv[9] 프로세서에서는 최소 16비트 버퍼를 사용한다. 하지만 버퍼를 사용하는 방법은 분기 시 Not Aligned 주소의 32비트 명령어일 경우에는 명령어의 하위 16비트는 가져올 수 있지만 명령어의 상위 16비트는 동일한 접근으로 가져올 수 없기에 IPC가 낮아지는 단점이 있다[7].

Fig. 1 (a)의 단점을 해결하기 위해 두 개의 I/O포트를 가진 16비트 메모리를 사용하여 메모리에 두 번 접근하는 방법을 제시한다. Fig. 1 (b)는 16비트 메모리를 사용하여 메모리에 PC와 PC+2로 두 번 접근하여 명령어 C를 가져오

는 방법이다. 이 방식은 32비트 메모리를 사용하면 생기는 문제인 분기 시 Not Aligned 주소의 32비트 명령어도 한 사이클에 가져올 수 있다[7].

관련 연구[7]에서 제안하는 메모리의 구조 변경으로 분기 시 Not Aligned 주소의 32비트 명령어를 가져오는 문제를 해결할 수 있다. 하지만 본 연구에서는 이미 데이터 메모리와 명령어 메모리에 두 개의 I/O포트를 가진 32비트 메모리를 사용하고 있기에 메모리 구조 변경은 불가능하다. 이러한 이유로 본 논문에서는 버퍼를 사용한 방법으로 16비트 압축 명령어에 대해 처리하며, 16비트 압축 명령어를 32비트 정수형 명령어로 확장하는 Decompress 모듈 사용을 제안한다.

3. 제안 기법

본 장에서는 5단계 파이프라인 기반 RV32IC 프로세서의 구조와 버퍼를 사용한 명령어 패치 방법을 구현하기 위해 필요한 메모리 접근 방법과 명령어 결합 방법을 설명한다. 또한, 명령어 확장을 통한 명령어 처리방법에 대해 설명한다.

3.1 파이프라인

Fig. 2는 5단계 파이프라인 RV32IC 프로세서의 구조를 나타내며 FPGA 구현을 위해 명령어 메모리와 데이터 메모리는 동기 메모리를 사용한다.

Fig. 2의 5단계 파이프라인은 IF(Instruction Fetch), ID(Instruction Decode), EXE(Execute), MEM(Memory), WB(Write Back) 단계로 이루어진다. IF 단계에서는 명령어의 길이와 분기 주소에 따른 PC를 설정하고 PC에 따른 명령어를 메모리로부터 가져온다. ID단계에서는 메모리에서 나온 명령어와 버퍼에 저장되어 있는 명령어를 PC와 명령어의 길이에 따라 명령어를 합쳐주는 과정을 실행하고 16비트 압축 명령어는 32비트 정수형 명령어로 확장하여 명령어를 해석하는 과정을 거친다. EXE 단계에서는 ALU를 통한 연산을 수행하고 분기 여부를 판단한다. MEM단계에서는 데이터를 데이터 메모리에 쓰거나 읽고 마지막 단계인 WB단계에서는 명령어의 결과를 레지스터 파일에 저장 및 업데이트한다.

3.2 제안하는 메모리 접근 방법과 명령어 결합 방법

버퍼를 사용하여 Not Aligned 주소의 32비트 명령어를 처리할 수 있지만 단순히 PC에 해당하는 주소로 메모리에 접근하게 되면 다음 메모리에 저장되어 있는 명령어의 상위 16비트값을 가져오지 못하는 문제가 존재한다. 즉, Not Aligned 주소일 때에 버퍼에는 이미 PC에 해당하는 명령어가 존재해야 하며 메모리 접근은 다음 PC에 대해 접근해야 Not Aligned 주소의 32비트 명령어를 가져올 수 있다. 위와 같은 문제를 해결하기 위해 메모리에 PC+2로 접근하는 것을 제안한다.

하지만 제시한 메모리 접근 방법은 일반적인 상황에서는 문제없지만 분기 시 Not Aligned 주소의 32비트 명령어일 때에는 문제가 생긴다. 분기 시 Not Aligned 주소의 32비트 명령어일 때에는 버퍼에 명령어의 하위 16비트가 저장되어 있는 것이 아닌 분기 이전의 명령어가 저장되어 있다. 이러한 상황에서 PC+2로 메모리에 접근하면 명령

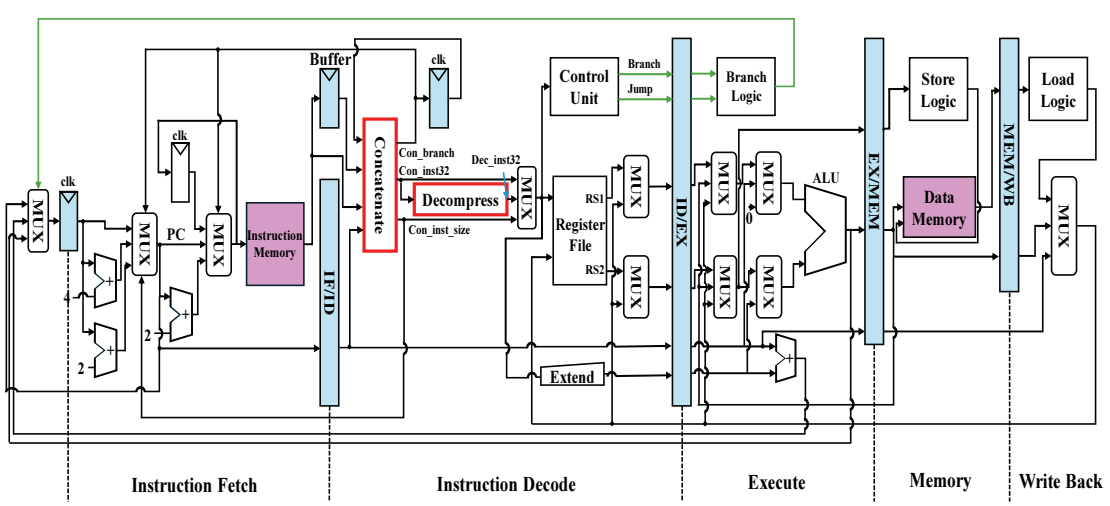

Fig. 2. A block diagram of five stage pipelined RV32IC Processor.

어의 상위 16비트만을 얻을 수 있다. 이때는 지연을 사용하여 첫번째 사이클에는 PC로 메모리에 접근하여 버퍼에 명령어 하위 16비트를 저장하고 두번째 사이클에는 PC+2로 메모리에 접근하여 해결할 수 있다.

버퍼를 사용하여 명령어를 가져오는 방법은 버퍼에 저장된 명령어와 메모리에서 나오는 명령어를 결합하는 과정이 필요하며 명령어를 결합하는 과정은 PC와 명령어의 길이에 따라 결정된다. 명령어의 길이는 명령어의 하위 2비트를 통해 알 수 있다. 하위 2비트가 2'b11이면 32비트 명령어이며 2'b11가 아니면 16비트 명령어이다.

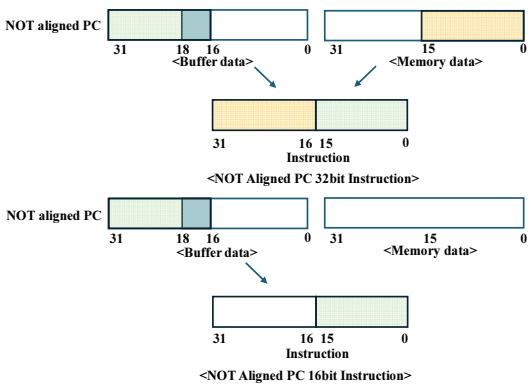

Fig. 3. Instruction fetching method for Not Aligned PC.

Aligned(PC=0,4,8,C) 주소에서 16비트 명령어와 32비트 명령어는 메모리 데이터의 하위 2비트를 확인하여 메모리 데이터에서 명령어를 가져올 수 있다. Fig. 3은 Not Aligned 주소에서 16비트 명령어와 32비트 명령어를 가져오는 방법이다. Not Aligned 주소에서는 버퍼 데이터의 [17:16]비트를 확인하여 길이를 판별할 수 있다. 16비트 명령어는 버퍼 데이터의 상위 16비트로 명령어를 가져올 수 있고 32비트 명령어는 버퍼 데이터의 상위 16비트와 메모리 데이터의 하위 16비트를 결합하여 명령어를 가져올 수 있다.

3.3 16 비트 명령어 확장 방법

RV32IC는 RV32I에 "C" 명령어를 확장한 프로세서이다. 16비트 압축 명령어를 지원하기 위해 모듈을 추가 설계하는 것은 프로세서 면적에 큰 단점이기 때문에 RV32I에 16비트 압축 명령어의 확장 모듈을 설계하여 해결할 수 있다. 즉, 32비트 명령어가 16비트로 압축이 가능하듯 반대로 16비트 명령어도 그에 대응하는 32비트 명령어로 확장이 가능하다.

Table 3은 제안하는 RV32IC에서 지원하는 압축 명령어 26개에 대해 16비트 압축 명령어 형식과 그에 대응하는 32비트 명령어를 나타낸다. 각각의 명령어들은 명령어의 하위 2비트와 상위 3비트를 통해 명령어를 확인할 수 있고 만약 하위 2비트와 상위 3비트가 같은 명령어들이 있다면 다른 추가적인 비트를 통해 명령어를 확인할 수 있다.

Table 2. 16-bit XOR instruction format and 32-bit XOR instruction format [2,6]

Compressed Instruction (C.XOR)				
15-10	9-7	6-5	4-2	1-0
100011	RD	01	RS2	01

Integer Instruction (XOR)							
31-27	26-25	24-20	19-15	14-12	11-7	6-2	1-0
00000	00	RS2	RS1	100	RD	01100	11

Table 2는 16비트 압축 명령어 C.XOR와 32비트 정수형 명령어 XOR의 형식을 나타낸다. Table 2에서 16비트 압축 명령어 C.XOR의 상위 3비트와 하위 2비트가 각각 3'b100, 2'b01인 것을 확인할 수 있고 Table 3을 통해 같은 상위 3비트와 하위 2비트를 가진 명령어들이 다수 존재하는 것을 알 수 있다. 이러한 경우 명령어의 11, 10, 6, 5번 비트의 값을 확인하여 C.XOR 명령어를 확인할 수 있다. C.XOR 명령어를 Table 2의 32비트 정수형 명령어 XOR명령어 형식을 통해 {7'b0, 2'b01, C.XOR[4:2], 2'b01, C.XOR[9:7], 3'b100, 2'b01, C.XOR[9:7], 7'b0110011}와 같은 형태로 확장할 수 있다.

4. 프로세서 성능 평가

RV32IC 프로세서 동작 검증을 위해 Xilinx의 Artix-7 NEXYS A7-100T기반 FPGA 보드를 사용하였으며, 프로세서 성능 측정은 Vivado 2020.01 버전을 이용하여 합성(Synthesis) 및 구현(Implementation)을 진행하였다. 합성 및 구현 옵션은 기본 옵션인 Vivado Synthesis Defaults와 Vivado Implementation Defaults로 설정하였다. 또한, Data Width는 32, Address Width는 15로 진행하였다.

Table 4는 RV32I[5]와 RV32IC 프로세서의 구현 결과인 LUT(Lookup Table), FF(FlipFlop), BRAM(Block RAM), 최대 동작 주파수 및 전력 사용량을 보여준다. RV32I[5]와 RV32IC 프로세서의 LUT는 각각 1,983과 2,016으로 RV32IC가 약 1.66% 증가하였고 FF은 1,673과 1,586로 약 5.2% 감소하였다. 파워는 각각 0.169와 0.167을 확인하였고 최대 동작 주파수도 52.95MHz와 51.88MHz임을 확인하였다.

프로세서 성능 측정을 위해 Dhrystone과 Coremark 벤치마크를 GCC 8.2.0 버전과 최적화 옵션 -O3를 사용해 각각 1000번 반복하여 실행하였다. Fig. 4는 제안하는 RV32IC 프로세서의 Coremark 벤치마크 결과를 FPGA 보드를 통해 UART로 나타낸 것이다.

Table 3. 26 compressed instruction formats supported by RV32IC and corresponding 32bit instructions[6]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	16-bit Instructions	32-bit Instructions
000			0									0		00		Illegal	
000			nzimm[5:4|9:6|2|3]									Rd		00		C.ADDI4SPN	ADDI
010			imm[5:3]			rs1			imm[2|6]			Rd		00		C.LW	LW
110			imm[5:3]			rs1			imm[2|6]			rs2		00		C.SW	SW
000			0	0					0					01		C.NOP	ADDI
000			nzimm[5]	rs1/rd!=0					nzimm[4:0]					01		C.ADDI	ADDI
001			offset[11|4|9:8|10|6|7|3:1|5]											01		C.JAL	JAL
010			imm[5]	rs1/rd!=0					imm[4:0]					01		C.LI	ADDI
011			nzimm[9]	2					nzimm[4|6|8:7|5]					01		C.ADDI16SP	ADDI
011			nzimm[17]	rs1/rd!={0,2}					nzimm[4:0]					01		C.LUI	LUI
100			nzimm[5]	00		rs1/rd			nzimm[4:0]					01		C.SRLI	SRLI
100			nzimm[5]	01		rs1/rd			nzimm[4:0]					01		C.SRAI	SRAI
100			imm[5]	10		rs1/rd			imm[4:0]					01		C.ANDI	ANDI
100			0	11		rs1/rd			00		rs2			01		C.SUB	SUB
100			0	11		rs1/rd			01		rs2			01		C.XOR	XOR
100			0	11		rs1/rd			10		rs2			01		C.OR	OR
100			0	11		rs1/rd			11		rs2			01		C.AND	AND
101			offset[11|4|9:8|10|6|7|3:1|5]											01		C.J	JAL
110			offset[8|4:3]			rs1			offset[7:6|2:1|5]					01		C.BEQZ	BEQ
111			offset[8|4:3]			rs1			offset[7:6|2:1|5]					01		C.BNEZ	BNE
000			nzimm[5]	rd!=0					nzimm[4:0]					10		C.SLLI	SLLI
010			imm[5]	rd!=0					imm[4:2|7:6]					10		C.LWSP	LW
100			0	rs1!=0					0					10		C.JR	JALR
100			0	rd!=0					rs2!=0					10		C.MV	ADD
100			1	rs1!=0					0					10		C.JALR	JALR
100			1	rd!=0					rs2!=0					10		C.ADD	ADD
110			imm[5:2|7:6]						rs2					10		C.SWSP	SW

Table 4. Hardware implementation results of proposed RV32I and RV32IC processor

Resource	Available	Utilization	
		RV32I[5]	RV32IC
LUT	63,400	1,983(3.13%)	2,016(3.18%)
FF	126,800	1,673(1.32%)	1,586(1.25%)
BRAM	135	32(23.7%)	32(23.7%)
Max Freq.	-	52.95MHz	51.88MHz
Power(W)	-	0.169	0.167

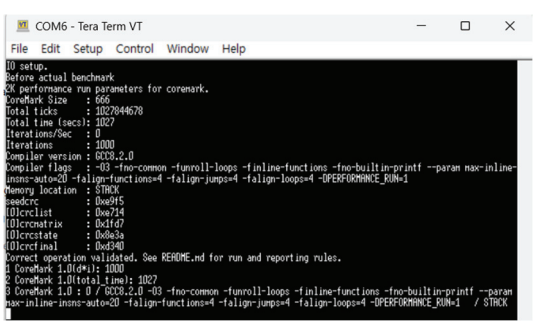

Fig. 4. The Coremark benchmark result of proposed RV32IC processor using UART.

제안하는 RV32IC의 성능을 비교하기 위해 선행 연구된 4개의 프로세서와 간접적으로 벤치마크 결과를 비교하였다. 선행 연구된 4개의 프로세서 중 RVP-c[6]는 32비트 정수형 명령어를 처리하는 RVCoreP-32I[10]를 기반으로 압축 명령어를 지원하기 위해 새로운 명령어 패치 방법 제시 및 디코더와 분기예측 방법을 변경한 프로세서이다.

RVP-c-buf[6]는 RVP-c에서 제안된 명령어 패치 방법의 효과를 입증하기 위해 버퍼를 추가한 프로세서로 명령어 패치 부분을 제외하곤 RVP-c[6]프로세서와 동일하다. RV32IC를 지원하는 VexRiscv[9]의 버전 중 분기 예측 방법을 갖춘 버전은 VR-bp-c[6]이고, 분기 예측 방법이 없는 버전은 VR-nobp-c[6]이다.

RVP-c[6]와 RVP-c-buf[6]프로세서는 이항 분기 예측기와 BTB 방식을 사용하는 VR-bp-c[6]에 비해 더 높은 예측 정확도를 가진 gshare 분기 예측기를 사용한다.

Table 5는 제안하는 RV32I[5]와 RV32IC 및 선행 연구된 프로세서의 DMIPS/MHz와 Coremark/MHz를 나타낸다. 제안하는 RV32IC 프로세서가 선행 연구된 프로세서 중 RVP-c[6]프로세서보다 DMIPS/MHz가 낮지만 다른 프로세

서들보다는 높은 것을 확인하였다. 하지만 Dhrystone은 주로 정수 연산 성능을 측정하기 때문에 전체적인 프로세서의 성능을 반영하기 어려운 단점이 있다.

Table 5. Benchmark results from the proposed and previously researched processors

Processor	DMIPS/MHz	Coremark/MHz
RVP-c[6]	1.149	1.122
RVP-c-buf[6]	1.113	1.105
VR-nobp-c[6]	0.803	0.789
VR-bp-c[6]	1.009	0.994
Proposed RV32I[5]	1.179	0.980
Proposed RV32IC	1.147	0.974

Dhrystone의 단점을 보완한 벤치마크인 Coremark는 임베디드 시스템에서 자주 사용하는 연결 리스트 처리와 행렬 연산 등을 포함하여 전체적인 프로세서 성능을 반영한다. Coremark/MHz를 기준으로 제안하는 RV32IC 프로세서는 VR-nobp-c[6]프로세서를 제외한 다른 선행 연구된 프로세서보다 낮은 점수를 갖는 것을 확인하였다. 이러한 결과는 분기 예측기를 지원하지 않았기에 나타나는 차이로 판단된다.

Table 5에 나타나는 제안하는 RV32IC 프로세서가 제안하는 RV32I[5] 프로세서보다 DMIPS/MHz는 2.72%, Coremark/MHz는 0.61%가 하락한 것을 확인하였다. 이러한 하락은 버퍼 사용으로 의한 지연으로 예상된다. 또한, 제안하는 RV32IC 프로세서의 Coremark/MHz가 DMIPS/MHz보다 낮은 것이 확인되며 이는 프로세서가 연결 리스트 처리와 행렬 연산 등 임베디드 시스템에 사용되는 연산의 성능이 떨어지는 것으로 판단된다.

Table 6은 GCC 컴파일로 생성된 RV32I프로세서와 RV32IC 프로세서의 코드사이즈를 보여주며 RV32IC프로세서가 RV32I프로세서보다 Dhrystone 코드에서 1.48%, Coremark 코드에서 14.93% 더 작은 코드 사이즈를 갖는 것을 확인하였다. 더 작은 코드 사이즈를 갖는 RV32IC프로세서가 메모리 효율성면에서 더 적합한 것으로 판단된다.

Table 6. Code sizes of hex files for RV32I and RV32IC generated by GCC compilation

Code Size	Dhrystone	Coremark
RV32I	135KB	75.0KB
RV32IC	133KB	63.8KB

5. 결 론

본 연구에서는 16비트 압축 명령어를 지원하기 위해 버퍼를 사용하였으며 메모리 접근 및 결합 방법, 명령어 확장 방법을 제안하여 5단계 파이프라인 RV32IC를 설계하였다. 또한, Xilinx의 Artix-7 NEXYS A7-100T기반 FPGA 보드를 사용하여 합성과 구현을 하였으며 Dhrystone, Coremark 벤치마크 점수와 코드 사이즈 확인을 통해 프로세서의 성능을 검증하였다.

성능을 비교한 결과 제안하는 RV32IC 프로세서가 RV32I프로세서보다 더 작은 코드 사이즈를 갖는 것을 확인하여 메모리 효율성 면에서 향상된 것을 확인하였다. 하지만 연결 리스트 처리 등 임베디드 시스템에 자주 사용되는 작업에 대한 프로세서의 성능 하락과 분기예측기를 지원하지 않아 생긴 성능 하락에 대해서는 추후 연구할 예정이다.

감사의 글

이 논문은 한국교원대학교 2024학년도 신임교수 학술연구비 지원을 받아 수행한 연구의 결과임.

참고문헌

1. GUL, S., AFTAB, N.. A Comparison between RISC and CISC Microprocessor Architectures. IJSEAT, North America, 4, jun. 2016. Available at: <http://www.ijseat.com/index.php/ijseat/article/view/663>. Date accessed: 09 May. 2024.
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. Weicker, Reinhold P., “Dhrystone: A Synthetic Systems Programming Benchmark,” Commun. ACM, vol. 27, no. 10, pp. 1013–1030, Oct.1984. [Online]. Available: http://doi.acm.org/10.1145/358274.358283
4. EEMBC, “CoreMark — CPU Benchmark – MCU Benchmark,” https://www.eembc.org/coremark/.
5. 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.
6. A. Waterman, Y. Lee, D. Patterson, K. Asanovi, “The RISC-V Compressed Instruction Set Manual, document version 20151205”, 2015
7. 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 May, 6, 2024, from https://arxiv.org/abs/2011.11246.
8. D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini, "Pulp: A parallel ultra low power platform for next generation iot applications," in 2015 IEEE Hot Chips 27 Symposium (HCS), Aug 2015, pp. 1–39.
9. SpinalHDL, "VexRiscv: A FPGA friendly 32 bit RISC-V CPU implementation," https://github.com/ SpinalHDL/VexRiscv.
10. H. Miyazaki, T. Kanamori , M. Ashraful Islam and K. Kise, "Rvcorep:An optimized risc-v soft processor of five-stage pipelining," arXiv preprint arXiv: 2002.03568, 2020.

접수일: 2024년 6월 4일, 심사일: 2024년 9월 5일,
게재확정일: 2024년 9월 12일