ISSN: 0974-5823 Vol. 7 No. 1 January, 2022

International Journal of Mechanical Engineering

Single Cycle 32-bit RISC-V ISA Implementation and Verification

Hyogeun An[1], Sudong Kang[2], Dennis A. N. Gookyi[3], Guard Kanda[4], Kwangki Ryoo[*5]

Department of Information and Communication Engineering, Hanbat National University, Daejeon, 34158, South Korea

Abstract.
Background/Objectives: This paper provides an insight into the internal verification of a 32-bit single cycle processor that implements the Reduced Instruction Set Computer Five Instruction Set Architecture.
Methods/Statistical analysis: This paper accesses the internal operation and information of a RISC-V 32-bit single cycle processor using a Field Programmable Gate Array board. The internal components such as the register file, the Program Counter, the instruction memory, and the data memory are displayed on peripherals such seven-segment display, text LCD, dot matrix, and LEDs. The hardware structures were realized using Verilog Hardware Description Language and synthesized using Xilinx Integrated Synthesis Environment 14.3 that incorporates ISE Simulator for simulation purposes.
Findings: With the recent trend of increasing demand and scope in the market for Internet of Things ubiquitous platforms, low-cost System-on-Chip devices are currently been deployed as sensors. Processor cores that implement the RISC-V ISA are suitable for low-cost SoCs due to their use of minimal hardware resources. The RISC-V ISA is fairly new but is taking over the open-source market and it is therefore important for hardware designers in the computer architecture field to understand the architecture of the ISA. This paper outlines the steps in implementing a single cycle 32-bit RISC-V ISA using Verilog HDL. The uniqueness of this work is in the verification of each instruction in the ISA. The verification is achieved using an FPGA device with peripherals such as seven-segment display, text LCD, dot matrix, keypad, and LEDs. These peripherals are used to display contents from the RISC-V processor core such as the register file, PC, instruction memory, and data memory. This work is vital because it enables researchers who are new to the RISC-V ISA quickly understand the internal operation of a processor core during real world operation on an FPGA board.
Improvements/Applications: The synthesis report of the RISC-V single cycle processor core with the various peripheral modules utilized 6834 Look-up-Tables at a maximum frequency of 64 MHz. This indicates that the core is suitable for low-cost IoT SoC devices and can serve as tutorial material for the computer architecture course.

Keywords: RISC-V ISA, Single Cycle Processor, FPGA, IoT, SoC, Hardware Verification.

1. INTRODUCTION

The Central Processing Unit (CPU) is an electronic circuit designed specifically to execute instructions called programs. The CPU executes many forms of instructions such as arithmetic and logic instructions as well as memory access instructions. These instructions which are specific to each CPU vendor are collectively known as the ISA. The ISA describes a specific instruction set that can be compiled by dedicated compilers and translated into machine codes. There are many systems in today's society that require very large computations such as deep learning and image-based processing. Therefore, the demand for embedded systems requiring low power, low cost, and high performance is also increasing. The paper in [1] gives an example of a system that efficiently recognizes vehicle license plate using an embedded systems and AI. In addition, the usability of IoT is gaining popularity and increasing user convenience based on its applicability in real life. Using the Lora module, paper [2] designed a system that broadcasts kindergarten school bus location notifications to users on SMS or apps on mobile devices. Among them, processors are indispensable components. But the commercially available ISAs belong to companies such as Intel and AMD which include the x86 family of ISAs [3], [4]. These ISAs are patented and cannot be used without license that cost a lot of money and thereby preventing researchers in academia and hobbyist from using them. Moreover, the licenses prevent designers from implementing the ISAs but instead limits them to use the processor cores provided by the companies. This prevents competition, innovation, and sometimes trust issues as companies could imbed malicious circuitry in the processor for spying [5].

The licensing issue associated with commercial processor is solved by the design of a new ISA known as RISC-V [6], [7] by the computer architecture group in University of California, Berkeley. The RISC-V ISA is an open-source ISA available for implementation under the free Berkeley open-source license. The RISC-V project started in 2010 and has rapidly grown with their board of directors that come from companies such as NVIDIA and Google. There are a number of open-source processors that implement the RISC-V ISA which include Rocket [8], Berkeley Out of Order Machine (BOOM) [8], PICORV32 [9] and many more. These processors are complex with pipelined structures that achieve high throughput with low hardware footprint [10]. These processors are therefore difficult to understand by researchers seeking information about the RISC-V ISA architecture. This paper therefore implements a 32-bit single cycle RISC-V ISA core with peripherals to monitor the internal operations of the processor.

The objectives of this paper are as follows

- The paper illustrates design and implementation of the control and datapath for a 32-bit single cycle RISC-V ISA core using Verilog HDL
- Hardware controllers for peripherals such as seven-segment display, text LCD, dot matrix, LEDs, and keypad are added to monitor the internal activities of the processor core. This is done to provide an easy way to input instruction and observe the internal data of components such as the register file, PC, instruction memory, and data memory.
- A verification platform is built and loaded onto an FPGA board for inputting instruction and observing the data in the internal architecture of the processor

The rest of this paper is organized as follows: Section 2 discusses some related works, Section 3 gives an introduction to the RISC-V ISA, Section 4 illustrates the design and implementation of the 32-bit single cycle RISC-V ISA, Section 5 describe the verification module for testing the processor core, Section 6 present the hardware resource utilized by the processor core together with the peripheral modules while the conclusion of the paper is discussed in Section 7.

2. RELATED WORK

Since the introduction of the RISC-V ISA in the last decade, several processor cores have been proposed and implemented for different reasons. Most the processor cores are meant for academic use while a few others are commercial. The academic processor cores are designed with interfaces for easy integration into an SoC. This is good for advances hardware engineers to quickly assemble an SoC but very difficult for beginners to understand the working of the processor core. This section therefore explores RISC-V processor cores with input/output peripherals that enable a designer to gain understanding of the internal operation of the processor cores.

The authors of [11] implemented the RISC-V ISA with instruction which include environment call, break, status registers, control, branch, memory, and arithmetic logic. A total of 38 instructions were simulated using ModelSim and Quartus-II simulators. This work only resulted to functional simulation without real hardware implementation on an FPGA board. This makes it difficult to know if the processor will work in the real world.

The authors of [12] proposed an Integrated Machine Code Monitor (iMCM) and implemented together with a RISC-V processor on an FPGA. The iMCM monitors functions according to the verification method of the RISC-V processor. A total of 27 compressed RISC-V ISA instructions were monitored through simulations and FPGA evaluation. The verification of the processor instruction is through a terminal to displays only the memory related commands and the results of the program trace in hexadecimal values. This makes it difficult to understand the actual internal operations of the processor under evaluation.

The work that comes close to this paper is proposed by [13]. The author of [13] proposed and implemented a fully synthesizable RISC-V processor core on an FPGA device to text LCD display port. A user inputs Assembly code which is converted to machine code and executed by the processor with the results displayed on the text LCD. The text LCD shown only current PC value and the results of executed RISC-V instruction. The work of [13] is extended with peripherals such as seven-segment display, LEDs, dot matrix, and keypad to form the basis of this paper. The displays show the internal content of the processor which include the PC, register file, instruction memory, and the data memory. These displayed values are shown for each RISC-V instruction which makes it easy to understand the internal operation of a typical RISC-V processor core.

3. RISC-V ISA TECHNICAL SPECIFICATION

The RISC-V ISA is a load-store ISA based on the principles of RISC. The load-store ISA format typically divide instructions into two groups which include memory access (where only store and load instructions have access to memory) and ALU operations (where the operands are in registers). The simplicity of the RISC-V ISA design makes it possible execute instructions in just one clock cycle.

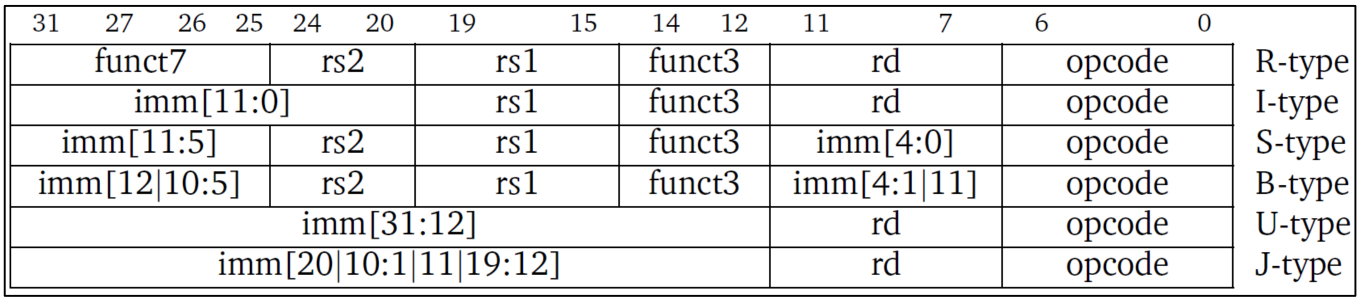
RISC-V has a base ISA which is referred to as RISC-V 32-bit Integer (RV32I) which is compulsory for any implementation. The RV32I instructions which include loads, stores, control flow, and integer computations. The RV32I instructions can be extended to include multiplication and division (RV32IM), atomic operations (RV32IA), single precision floating point (RV32IF), and double precision floating point (RV32ID). These extensions are optional to implement. It must be noted that the RISC-V ISA also allows for data widths of 64-bit and 128-bit as shown in Table 1 but this paper implements a data width of 32-bit.

Table 1: Summary of RISC-V ISAs

ISA	Description
RV32/64/128I	Base Integer Instruction Set
RV32/64/128M	Extension for Integer Multiplication and Division Instructions
RV32/64/128A	Extension for Atomic Instructions
RV32/64/128F	Extension for Single-Precision Floating-Point Instructions
RV32/64/128D	Extension for Double-Precision Floating-Point Instructions

There are a total of 32 registers used by the RISC-V ISA out of which 31 are general purposed registers (register x1 to x31) while one register (register x0) is hardware wired to a constant 0. In addition to the 32 registers, a special register is used to store the address of the current instruction known as the PC register. The width of the registers could be 32-bit, 64-bit, or 128-bit depending on the implementation.

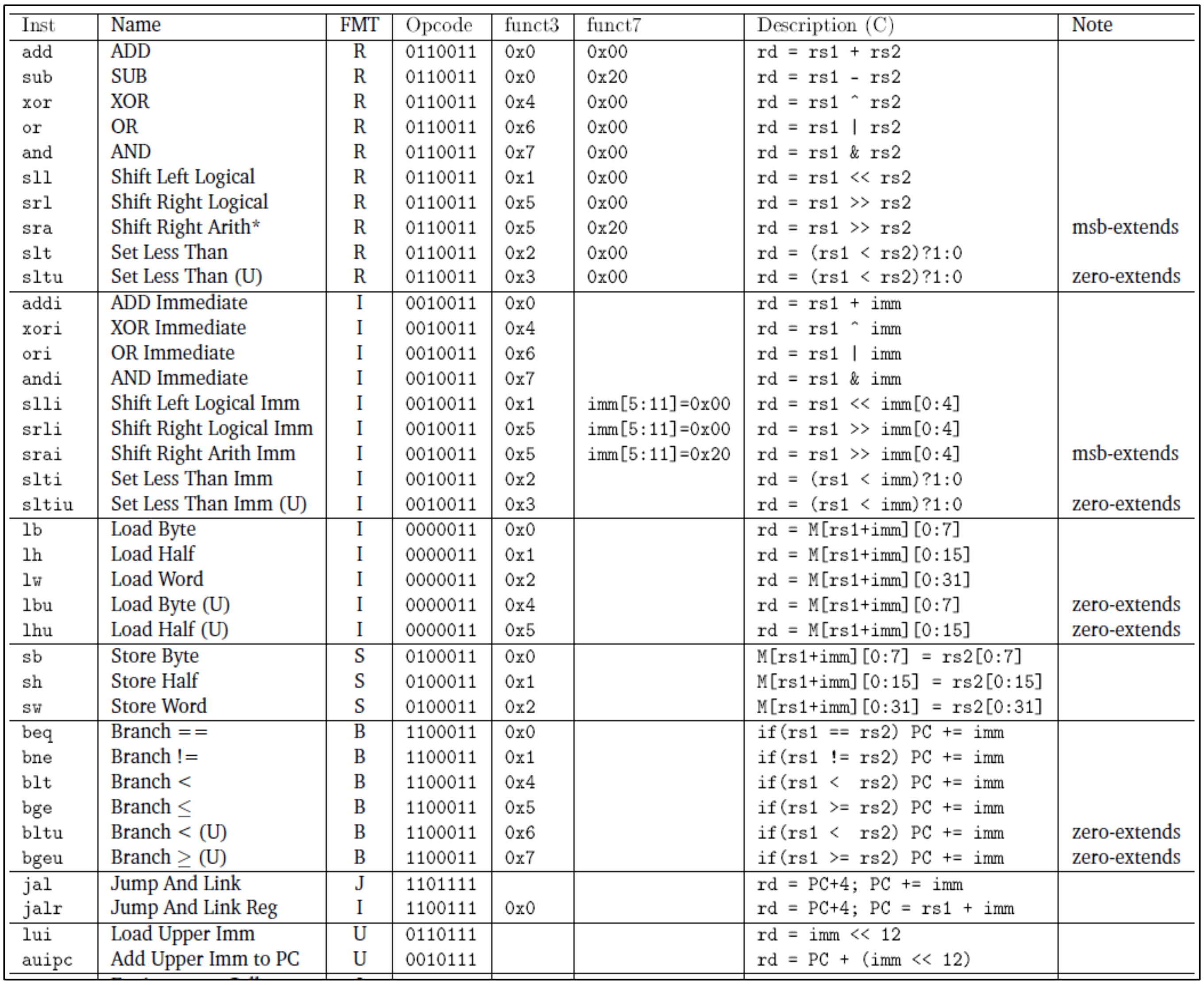
The RISC-V ISA consists of six main instruction formats which include, R-type (register-register instructions), I-type (loads and jump-and-link instructions), S-type (store instructions), B-type (branch instructions), U-type (load-upper-immediate instructions), and J-type (jump instructions). Figure 1 illustrates the various instruction format [14], [15] where funct stands for function, opcode stands for operation code, imm stands for immediate, rd stands for destination registers, rs1 stands for source register 1, and rs2 stands for source register 2.



31 27 26 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20|10:1|11|19:12]				rd	opcode	J-type

Figure 1. RISC-V 32-bit Instruction Format

The RISC-V RV32I which is implemented in this paper consist of a total of 39 instructions. This paper implements 37 of the 39 instructions which are illustrated in Figure 2.



Inst	Name	FMT	Opcode	funct3	funct7	Description (C)	Note
add	ADD	R	0110011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0110011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0110011	0x4	0x00	rd = rs1 ˆ rs2	
or	OR	R	0110011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0110011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0110011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0110011	0x5	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0110011	0x5	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2	0x00	rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3	0x00	rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0		rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x4		rd = rs1 ˆ imm	
ori	OR Immediate	I	0010011	0x6		rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x7		rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	imm[5:11]=0x00	rd = rs1 << imm[0:4]	
srli	Shift Right Logical Imm	I	0010011	0x5	imm[5:11]=0x00	rd = rs1 >> imm[0:4]	
srai	Shift Right Arith Imm	I	0010011	0x5	imm[5:11]=0x20	rd = rs1 >> imm[0:4]	msb-extends
slti	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltiu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≤	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1 + imm	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	

Figure 2. RISC-V RV32I Base Instruction Set

4. RV32I ISA SINGLE CYCLE PROCESSOR HARDWARE ARCHITECTURE

The hardware architecture for the RV32I ISA was designed by expanding on an architecture implemented in the RISC-V book [15]. The authors of [15] implemented a basic RISC-V hardware architecture capable of executing a total of seven instructions which include load doubleword (ld), store double word (sd), add, sub, and, or, and branch if equal (beq). This architecture was expanded to execute a total of 37 instructions and formed the basis of this work. Since the main objective of this work is to help researchers understand the inner workers of the RISC-V processor, the implementation of the RV32I ISA is simplified. The hardware architecture of the RV32I ISA implementation is shown in Figure 3. The architecture is a single cycle design which means it is capable of fetching, decoding, and executing the 37 RISC-V instructions in just a single cycle using components such as the PC, ALU, register file, instruction memory, data memory, and some basic logic gates. This section examines each component of Figure 3.

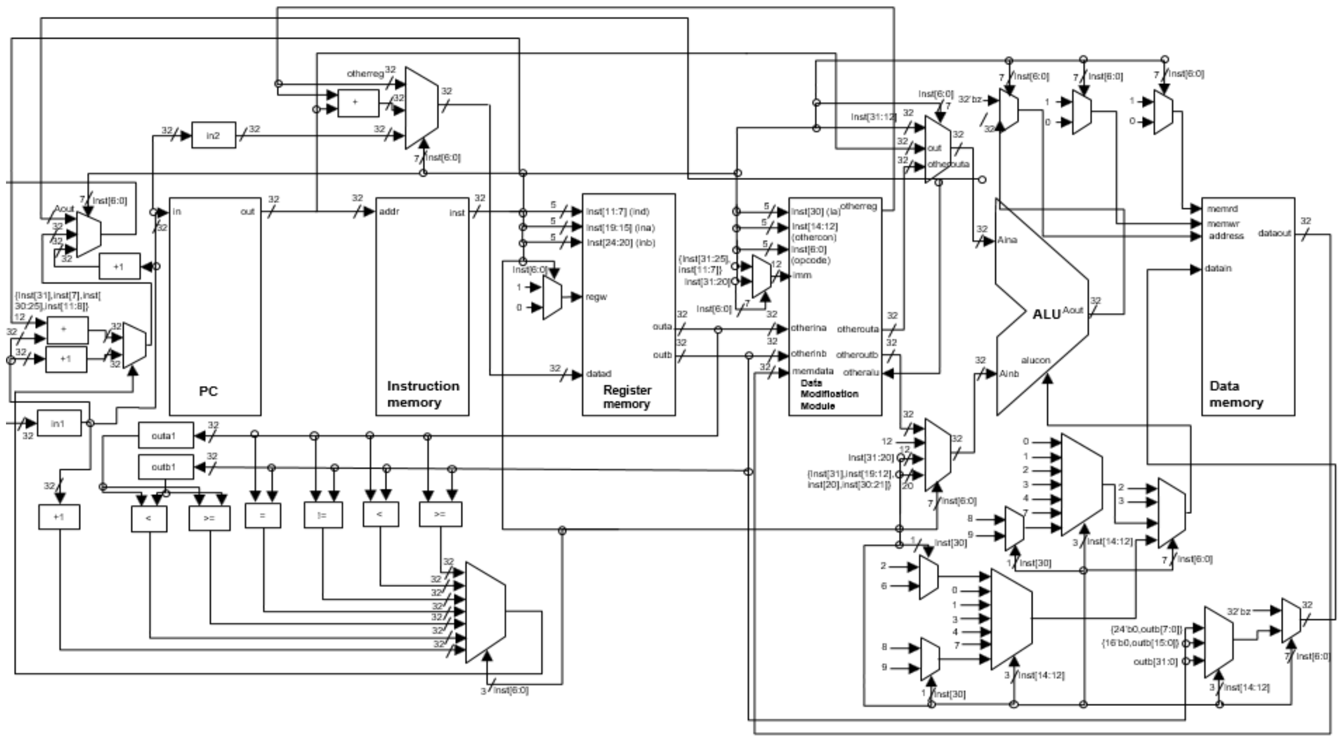


Figure 3. The Hardware Architecture of the RV32I ISA Design

4.1. PROGRAM COUNTER REGISTER

The PC is a special register inside the processor that stores the address of the next processor instruction. The PC register in the RV32I ISA is has a data width of 32-bit. The instruction is the machine code which is in the instruction memory. Generally, in the design of a CPU, the PC is usually increased by 4 during the normal instruction execution. The increased by a factor 4 is because most CPUs are byte addressable. For simplicity of the memory design, this work increases the PC by a factor of 1 to fetch all 32-bit instruction in the instruction memory. When executing B-type (branch), U-type (load-upper-immediate), and J-type (jump), the PC register stores the calculated value of the jump addresses. The output of the PC register is assigned to the instruction memory, register file, or the ALU depending on the type of instruction performed. Figure 4 shows a diagram of the PC register input/output signals.

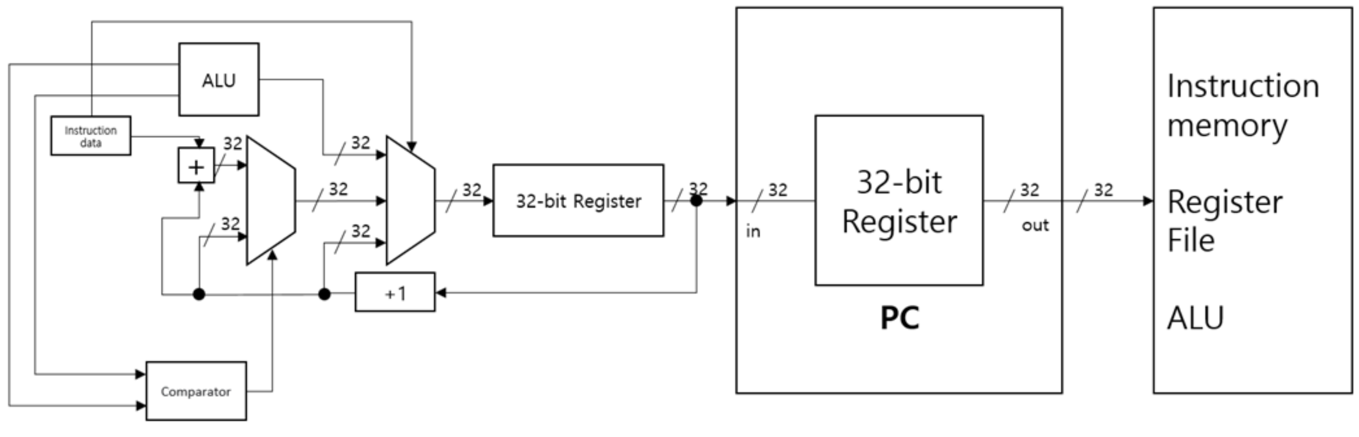
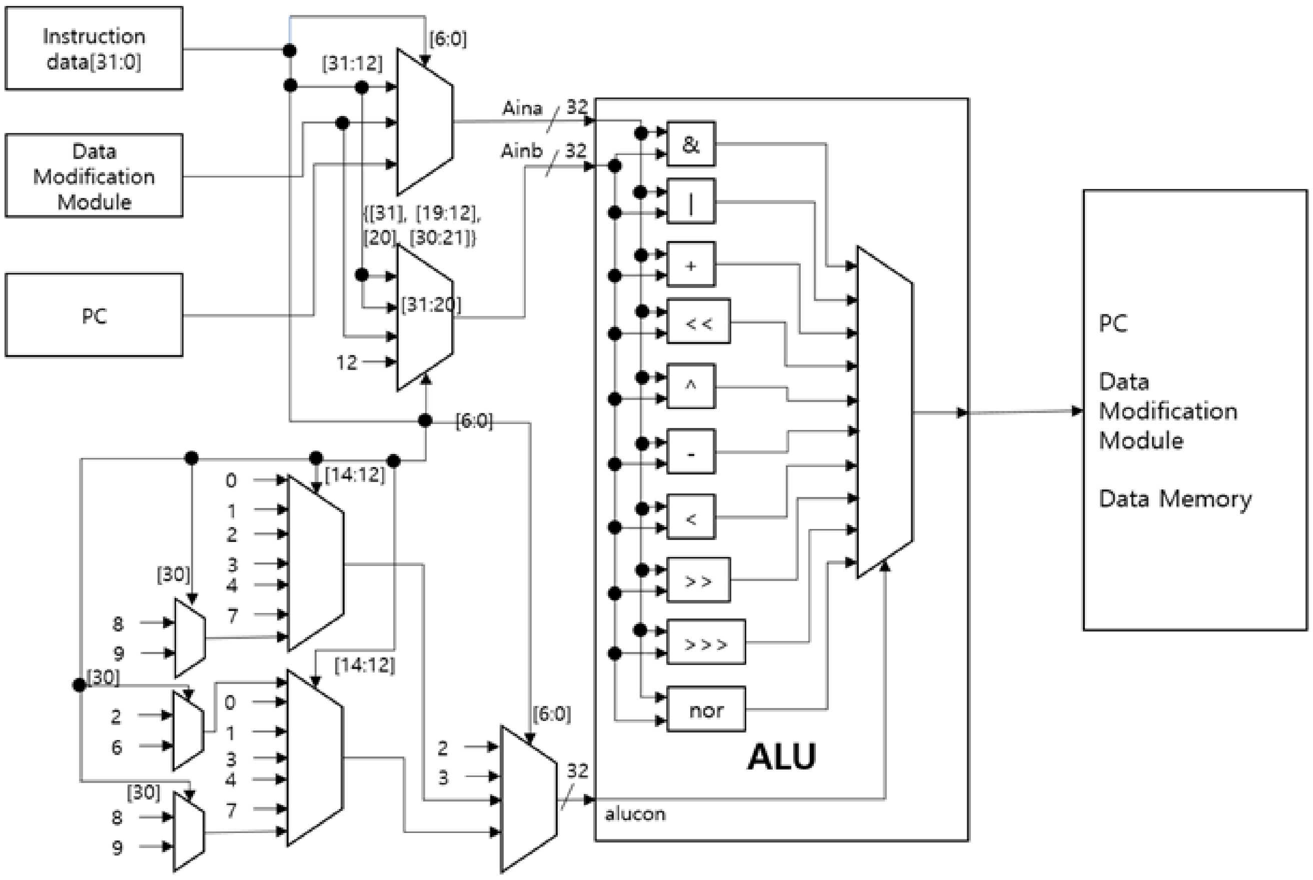


Figure 4. PC Register Input/Output Signals

4.2. ALU ARCHITECTURE

The ALU is responsible for performing arithmetic and logical operations. The ALU is a fundamental basic block of every processor. The ALU cannot store data on its own, it must go through a register. The built-in ALU designed in this work has three 32-bit inputs and one 32-bit output. The inputs consist of a data from the PC (JAL instruction execution) and data from the instruction memory (R-type instruction from the register). A multiplexer is used to route the signals to the input of the ALU. The internal circuitry of the ALU can perform 10 operations which include AND, OR, ADD, SLL, XOR, SUB, SLT, SRL, SRA, and NOR. The output signal is assigned to the PC register, the data memory, and the Data Modification Module depending on the instruction type. Figure 5 shows the architecture of the ALU



.

Figure 5. ALU Architecture

4.3. REGISTER FILE ARCHITECTURE

The RV32I ISA consist of a total of 32 32-bit registers for holding data. A register is a memory device that temporarily remembers the data that the processor needs to process the instruction. The registers are few but are directly connected to the ALU which makes the computation and storage of intermediate data very fast. The register file implemented in the work consist of five inputs and two outputs. The input labeled in a, in b, and in d each of 5-bit width select the register to write to or read from depending on the instruction type. Two multiplexors with select signals from the instruction are used to route data to selected registers. The internal architecture of the register file is made up 32x32 memory. The value of in a, in b determines the address of the 32-bit output ports outa, out b. The output signals are assigned to the Data Modification Module which in goes to the PC register depending on the type of instruction. Figure 6 shows the input/output logic of the register file implemented in this work.

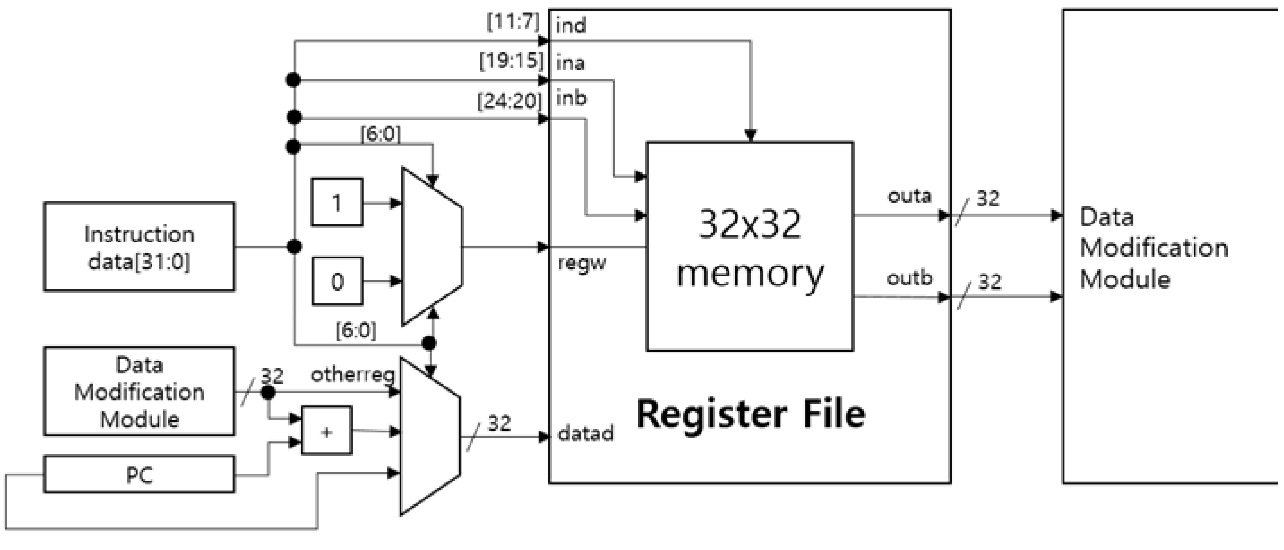


Figure 6. Register File Architecture

4.4. DATA MODIFICATION MODULE

The Data Modification Module is responsible for the division of data into bytes, handling of signed and unsigned numbers, and some computation of values. Data Modification Module consist of seven inputs and three outputs. The three input labeled othercon, opcode, and imm serve as control signals for activating the Data Modification Module. The signal labeled otherina and otherinb are data from the register file and the signal labeled memdata is from data memory. The internal circuitry of the Data Modification Module consists of logic that converts 32-bit data into the signed format and logic that divides 32-bit data into 8-bit, 16-bit, and 24-bits. The output signals labeled otherouta and otheroutb are assigned to the ALU while the output signal labeled otherreg is assigned to the register file. Figure 7 shows the architecture of the Data Modification Module.

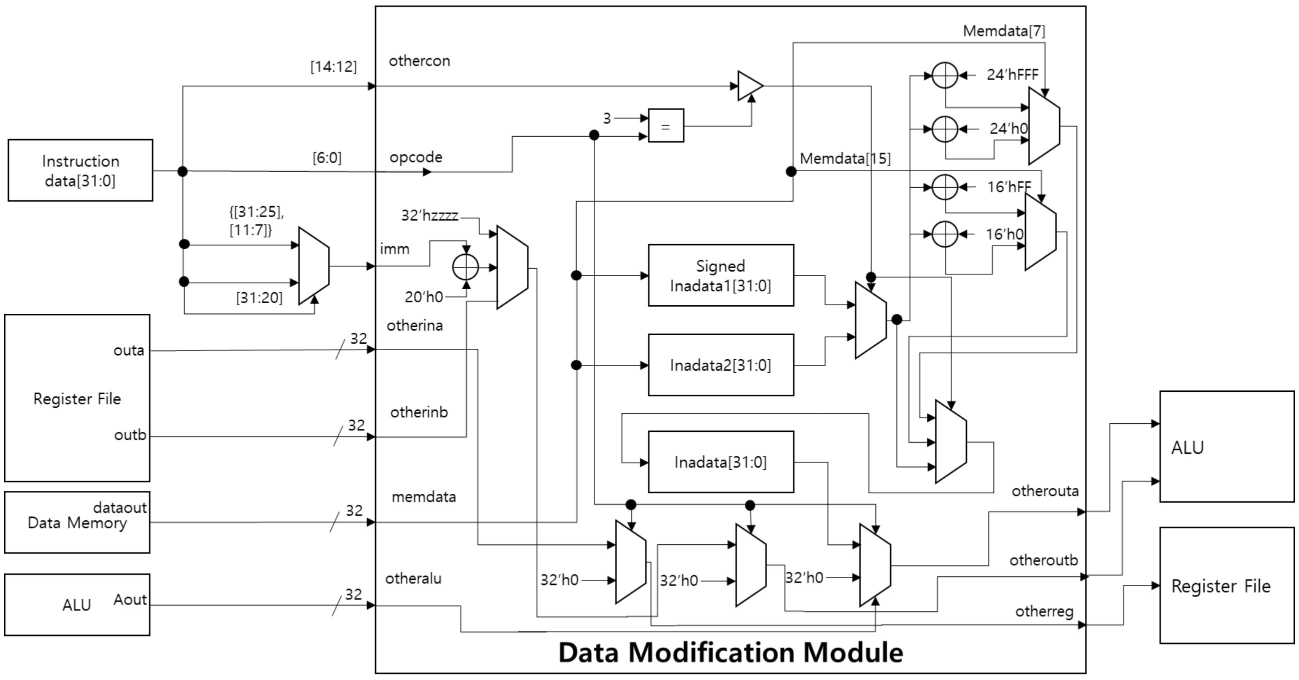


Figure 7. Data Modification Module Architecture

4.5. INSTRUCTION MEMORY

The instruction memory is responsible for storing 32-bit machine code (instructions). The instruction memory implemented in this work is made up of 32-bit data width with a depth of 512. This memory is read only and therefore only needs an address port to get access to a particular instruction. The PC output port provides the address of an instruction to be read from the instruction memory. The 32-bit instruction from the output of the instruction memory is assigned to modules such as the PC, register file, ALU, data memory, and Data Modification Module. The 32-bit output serve as control signal for activating the values modules depending on the type of instruction to execute. Figure 8 illustrates the input/output signal of the instruction memory.

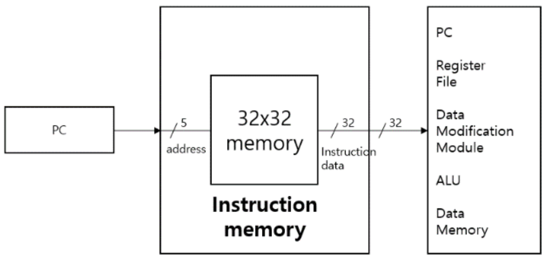


Figure 8. Instruction Memory Input/Output

4.6. DATA MEMORY

The data memory is responsible for storing data that is not instruction or machine code. This data is usually generated when the processor operational. This memory is a type of Random-Access Memory (RAM) in which data can be read and written. RISC-V is little endian which means that when storing data in memory, the Least Significant Byte (LSB) is stored first. The data memory consists of four inputs and one output. The 32-bit output of the instruction memory is assigned to the Data Modification. Figure 9 shows the input/output signals of the data memory.

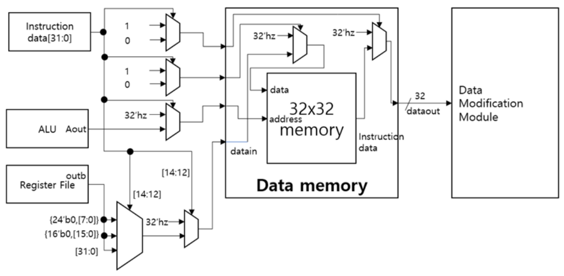


Figure 9. Data Memory Input/Output

4.7. GENERAL FLOW OF RISC-V ISA

The typical operation of the RISC-V processor consists of five steps which include instruction fetch, decode, execute, memory access, and writeback. Figure 10 shows the flow of a typical RISC-V processor operation. In the information fetch stage, the PC output the address to the instruction memory which make available the 32-bit instruction or machine code. The decode stage group the instruction code and sent to the various modules as control signals. In the execute stage, the various modules which the ALU, register file, and Data Modification Module are used to perform the instruction. In the memory access stage, data is read or stored in the data memory depending on the type of instruction. The writeback stage stores value in the register file depending on the type of instruction.

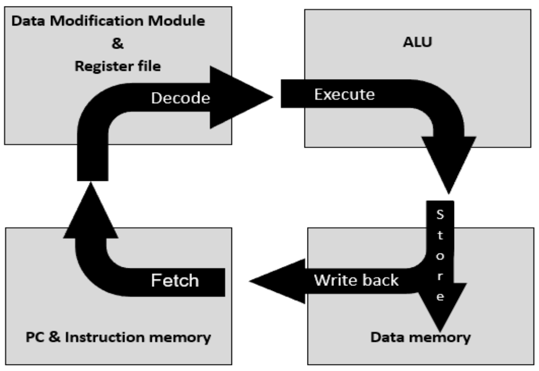


Figure 10. General Flow of RISC-V Instruction Execution

5. HARDWARE VERIFICATION MODULE OF RISC-V PROCESSOR

A verification module is proposed to observe the functionality of the RISC-V RV32I processor core. This module is important because it enables observers to easily understand the RISC-V ISA by observing real internal values when an instruction is operating. To observe the internal operation of the processor core, various peripherals are used to access real-time values from components such as registers, PC registers, data memory, and instruction memory during normal operation of the processor. Figure 11 shows the various peripheral used to observe the internal operation of the RISC-V processor. The peripherals and with their functionalities are shown in Table 1.

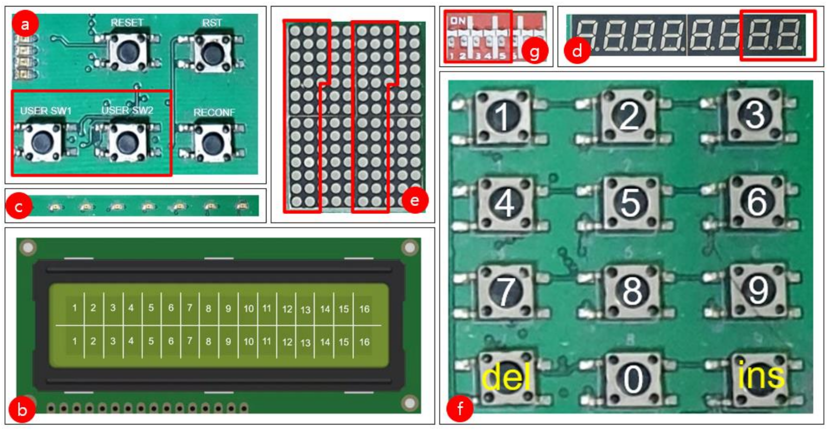


Figure 11. RISC-V Processor Verification Peripherals

Table 1: Functionalities of RISC-V Processor Verification Peripherals

Peripheral	Functionality
(a) Push Button	USER SW1 button is used to increment of the PC by 1 while USER SW2 button is used to increment the address of the instruction memory.
(b) Text LCD	First row 2-4 displays status. First row 8-15 display user value entered. The second row 2-5 display the current instruction. Second row 7-14 displays the current value in the instruction memory.
(c) LEDs	Eight LEDs are used to display the current PC value.
(d) 7-Segment Display	The last two segments display the address of the instruction memory
(e) Dot Matrix	The Dot matrix is used to display data stored in the instruction memory, data memory and register file.
(f) Keypad	The del button is used to delete an entered value. The ins button is used to insert an entered value. Button 1 to 9 are used for entering the machine code.
(g) Dip Switch	The first five are used to set the register and data memory address.

5.1. PROCESSOR VERIFICATION ON AN FPGA BOARD

The RV32I single cycle processor was verified on an FPGA board designed by HANBACK Electronics. The test board is equipped with Virtex-4 XC4VLX80 FPGA device. The peripherals for observing the operation of the processor include, push buttons, text LCD, LEDs, 7-segment display, dot matrix, keypad, and dip switch. Controller modules were designed for each of the peripherals. Figure 12 shows the connections of the peripheral to the FPGA device. From the figure, the Test_module consist of the various peripheral controllers which are also connected to the processor.

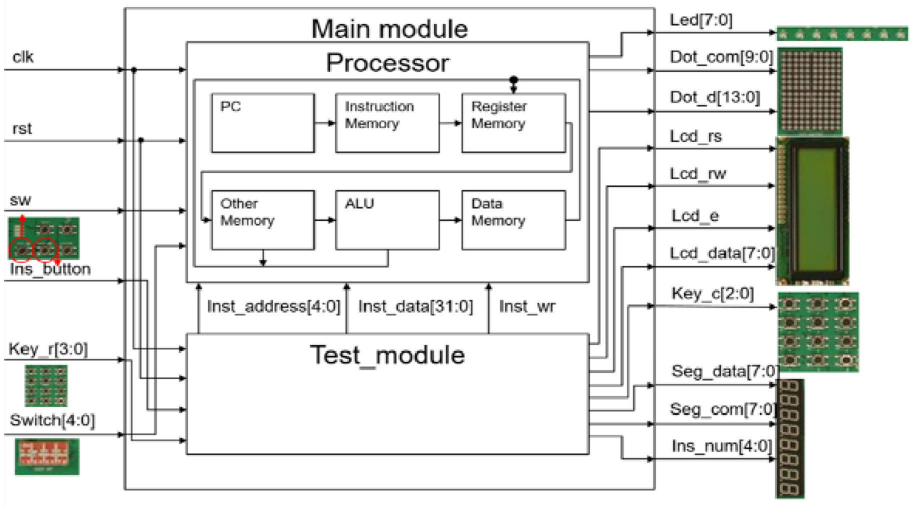


Figure 12. Connections of Peripheral to the FPGA Device

Figure 13 shows the flow chart for checking RISC-V processor instructions on the FPGA board. When the FPGA is powered up, all registers and memory locations are initialized to zero. The user uses the keypad to insert a instruction that starts with the Operation Code (OPCODE). The USER SW2 button is used to set the instruction address shown in seven segments before entering the instruction data. The desired number of buttons are used to insert the instruction code. By entering all the values of the instruction code, it is stored in FPGA instruction memory at the address previously specified. The USER SW1 button is used to manually increase the value of the PC which is shown on the LED. The dip switch is used to display values from the register file, instruction memory, and data memory on the dot matrix. Figure 14 illustrates the output of ADD UPPER IMMEDIATE TO PC (AUIPC) instruction on the FPGA board.

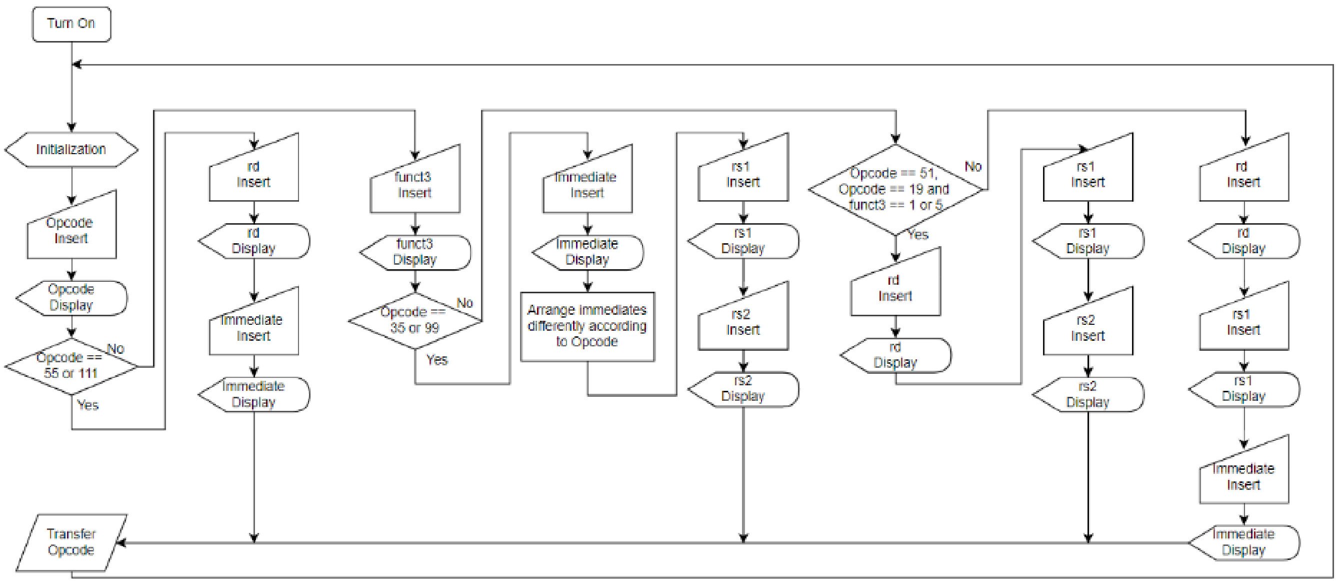


Figure 13. RISC-V Processor Verification Flow Chart

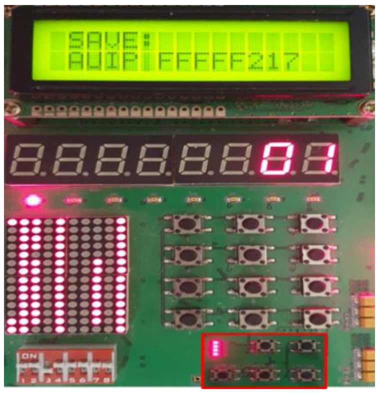


Figure 14. Display of AUIPC RISC-V Instruction of an FPGA Board

5.2. HARDWARE SYNTHESIS RESULTS OF RISC-V VERIFICATION MODULE

The processor core together with the peripheral controllers was synthesized using Xilinx Virtex4 FPGA device which resulted in 6834 LUTs at a maximum frequency of 64 MHz as shown in Table 2. Don et al. [13] designed a RISC-V processor with Text LCD peripheral for verification. When this work is compared to that of [13], this work consumed more LUTs because of the use of several peripheral controllers for the processor internal state verification than that of reference [13] but achieved twice the frequency of reference [13].

Table 2: Synthesis Results and Comparison

Design	**Processor Type and Peripherals**	**Area (LUTs)**	**Frequency (MHz)**
[13]	RV32I Processor, Text LCD	5578	32
This Work	RV32I Processor, Text LCD, LEDs, Dip Switch, Push Buttons, 7-Segment, Dot Matrix, Keypad	6834	64

6. HARDWARE VERIFICATION MODULE OF RISC-V PROCESSOR

This work provides an insight into the internal operation of a RISC-V RV32I processor using an FPGA device and input/output peripherals. This work can serve as teaching material for the computer architecture course that will enable students to understand the internal operation of a RISC-V processor core. In the future, a user-friendly desktop/web application will be designed to serve as an interface to the FPGA and display the internal values of the RISC-V processor core.

7. REFERENCE

1. Ryu JW, Lee JH, An Efficient Vehicle License Plate Recognition System Based on Embedded Systems, Journal of Next-generation Convergence Technology Association. 2021 Feb. 5(1) 22-27.
2. Ryu HG, Ryu KK, Kindergarten school bus notification service using IoT network, Journal of Next-generation Convergence Technology Association. 2019 Mar. 3(1) 21-28.
3. Daniel D. Open hardware: Initial experience with synthesizable open cores [dissertation]. [Uppsala]: Uppsala Universitet; 2019.
4. Kim WY. "The Linux 'RISC-V' singularity in the CPU world has begun to break through," [Online], Retrieved 25 January, 2021. https://zdnet.co.kr/view/?no=20201201123035.
5. Swarup B, Michael SH, Mainak B, Seetharam N. Hardware trojans attacks: Threat analysis and countermeasures, Proceedings of the IEEE. 2014 Aug; 102(8): 1229-1247.
6. Andrew W, Yunsup L, David P, Krste A. The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA. Technical Report UCB/EECS-2011-62, EECS Department, University of California, Berkeley, May 2011.
7. Nari L. "SiFive to showcase RISC-V based PC in October," [Online], Retrieved 26 February, 2021. http://www.thelec.net/news/articleView.html?idxno=1584.
8. Krste A, Rimas A, Jonathan B, Scott B, David B, Christopher C, Henry C, Daniel D, John H, Adam I, Sagar K, Ben K, Donggyu K, John K. The Rocket chip generator. Technical Report UCB/EECS-2016-17, EECS Department, University of California, Berkeley, Apr 2016.
9. Clifford W. "PICORV32 – A size-optimized RISC-V CPU," [Online], Retrieved 26 February, 2021. https://github.com/cliffordwolf/picorv32.
10. Dennis ANG, Kwangki R. Selecting a synthesizable RISC-V processor core for low-cost hardware devices, Journal of Information Processing Systems. 2019 Dec. 15(6), 1406-1421.
11. Lee JB, Simulation and synthesis of RISC-V processors, The Journal of the Institute of Internet, Broadcasting and Communication. 2019 Feb. 19(1), 239-245.
12. Hiroaki K, Akinori K. An integrated machine code monitor for a RISC-V processor on an FPGA, Artificial Life Robotics. 2020 Mar. 25, 427–433.
13. Don KD, Ayushi P, Virk SS, Sajal A, Tanuj S, Arit M, Kailash CR, Single cycle RISC-V micro architecture processor and its FPGA prototype, Proceedings of 7th International Symposium on Embedded Computing and System Design. 2017 Dec. India.
14. Bucknell University Computer Science Department. "RISC-V instruction reference (green sheet)," [Online], Retrieved 26 February, 2021. http://csci206sp2020.courses.bucknell.edu/files/2020/01/riscv-card.pdf
15. David AP, John LH. Computer organization and design: The hardware/software interface, RISC-V 5th edn, Elsevier, 2017.