

# Project 1 Executuion Platforms

Hamden Brini, Wilches Juan, Barau Elena, Marculescu Tudor

January 2025

## 1 Introduction to RISC-V Instruction Set Architecture

In this section we are focusing on a decomposition of RISC-V hex instruction into the ASM instruction. The instruction format is determined based on the opcode, funct3 and funct7 fields. Table 1 depicts a detailed translation of each instruction from it's hex code to it's format fields. Some of them not used in certain instruction types, and are marked with a dash (-) in the table. The ultimate factor that determines the instruction type is the "Type" column, which indicates instruction format label R, I, S or SB.

### 1.1 Program Instructions Decomposition

Address	Hex Code	Opcode (6:0)	rd (11:7)	funct3 (14:12)	rs1 (19:15)	rs2 (24:20)	funct7 (31:25)	imm[11:0] (31:20)	imm[11:5] (31:25)	imm[4:0] (11:7)	Type
0x0	0x00050893	0010011	10001	000	01010	-	-	000000000000	-	-	I
0x4	0x00068513	0010011	01010	000	01101	-	-	000000000000	-	-	I
0x8	0x04088063	1100011	-	000	10001	00000	-	-	0000010	00000	SB
0xc	0x04058263	1100011	-	000	01011	00000	-	-	0000010	00100	SB
0x10	0x04060063	1100011	-	000	01100	00000	-	-	0000010	00000	SB
0x14	0x04d05063	1100011	-	101	00000	01101	-	-	0000010	00000	SB
0x18	0x00088793	0010011	01111	000	10001	-	-	000000000000	-	-	I
0x1c	0x00269713	0010011	01110	001	01101	-	-	000000000010	-	-	I
0x20	0x00e888b3	0110011	10001	000	10001	01110	0000000	-	0000000	10001	R
0x24	0x0007a703	0000011	01110	010	01111	-	-	000000000000	-	-	I
0x28	0x0005a803	0000011	10000	010	01011	-	-	000000000000	-	-	I
0x2c	0x01070733	0110011	01110	000	01110	10000	0000000	-	0000000	01110	R
0x30	0x00e62023	0100011	-	010	01100	01110	-	-	000000000000	00000	S
0x34	0x00478793	0010011	01111	000	01111	-	-	00000000100	-	-	I
0x38	0x00458593	0010011	01011	000	01011	-	-	00000000100	-	-	I
0x3c	0x00460613	0010011	01100	000	01100	-	-	00000000100	-	-	I
0x40	0xffff1792e3	1100011	-	001	01111	10001	-	-	1111111	00101	SB
0x44	0x000008067	1100111	00000	000	00001	-	-	000000000000	-	-	I
0x48	0xffff00513	0010011	01010	000	00000	-	-	111111111111	-	-	I
0x4c	0x000008067	1100111	00000	000	00001	-	-	000000000000	-	-	I
0x50	0xffff00513	0010011	01010	000	00000	-	-	111111111111	-	-	I
0x54	0x000008067	1100111	00000	000	00001	-	-	000000000000	-	-	I

Table 1: Decomposition of Hex Codes to RISC-V Instruction Format

In order to better understand the provided program, Table 2 is introduced to map the hex codes with their corresponding assembly instructions. The register numbers are provided in the (x-registers) column and their symbolic names are provided in the (ABI) column for better clarity.

Address	Hex Code	ASM Instruction (ABI)	ASM Instruction (x-registers)
0x0	0x00050893	addi a7, a0, 0	addi x17, x10, 0
0x4	0x00068513	addi a0, a3, 0	addi x10, x13, 0
0x8	0x04088063	beq a7, zero, 64	beq x17, x0, 64
0xc	0x04058263	beq a1, zero, 68	beq x11, x0, 68
0x10	0x04060063	beq a2, zero, 64	beq x12, x0, 64
0x14	0x04d05063	bge zero, a3, 64	bge x0, x13, 64
0x18	0x00088793	addi a5, a7, 0	addi x15, x17, 0
0x1c	0x00269713	slli a4, a3, 2	slli x14, x13, 2
0x20	0x00e888b3	add a7, a7, a4	add x17, x17, x14
0x24	0x0007a703	lw a4, 0(a5)	lw x14, 0(x15)
0x28	0x0005a803	lw a6, 0(a1)	lw x16, 0(x11)
0x2c	0x01070733	add a4, a4, a6	add x14, x14, x16
0x30	0x00e62023	sw a4, 0(a2)	sw x14, 0(x12)
0x34	0x00478793	addi a5, a5, 4	addi x15, x15, 4
0x38	0x00458593	addi a1, a1, 4	addi x11, x11, 4
0x3c	0x00460613	addi a2, a2, 4	addi x12, x12, 4
0x40	0xff1792e3	bne a5, a7, -28	bne x15, x17, -28
0x44	0x00008067	jalr zero, ra, 0	jalr x0, x1, 0
0x48	0xffff00513	addi a0, zero, -1	addi x10, x0, -1
0x4c	0x00008067	jalr zero, ra, 0	jalr x0, x1, 0
0x50	0xffff00513	addi a0, zero, -1	addi x10, x0, -1
0x54	0x00008067	jalr zero, ra, 0	jalr x0, x1, 0

Table 2: RISC-V Instructions with register numbers, symbolic names and addresses

## 1.2 Branch Delay Slot Concept

The branch delay slot concept is interesting when discussing pipelined processors. Essentially, when a branch instruction is taken, the instructions that were fetched after the branch instructions might

become invalid, if the branch is taken. To avoid this, the branch delay slot declares that the instruction immediately following a branch instruction is always executed, regardless of whether the branch is taken or not. This helps to mitigate the performance penalty associated with branch instructions. As explained in [1]: "The idea of the branch shadow or delay slot is to recover one of those clocks. If you declare that the instruction after a branch is always executed then when a branch is taken the instruction in the decode slot also gets executed, the instruction in the fetch slot is discarded and you have one hole of time not two. So instead of execute, empty, empty, execute, execute you now have execute, execute, empty, execute, execute... in the execute stage of the pipeline. The branch is 50% less painful, your overall average execution speed improves, etc."

A disadvantage of using branch delay slots is that it may create complications in code debugging, since if the instruction in the delay slot has side effects, it may lead to an unexpected state of the registers and the memory. Also, it adds to the waiting time when trying to execute interruptions, since they will be deferred until the delay slot instruction is executed.

Advantages of using branch delay slots include improved performance in pipelined architectures, since it helps to reduce the number of pipeline stalls caused by branch instructions.

plus branch delay slots was initially a good idea then a bad idea

### 1.3 Branch Instructions Analysis

In order to understand better the provided program, it is useful to check the branch instructions and where they lead to.

Table 3: RISC-V Instructions with Addresses

Address	Conditional branch	Branch to
0x08	beq a7, zero, 64	0x48: addi a0, zero, -1
0xc	beq a1, zero, 68	0x50: addi a0, zero, -1
0x10	beq a2, zero, 64	0x50: addi a0, zero, -1
0x14	bge zero, a3, 148	0xA8: ? - the immediate field is sign extended in riskv
0x40	bne a5, a7, -16	0x30: sw a4, 0(a2)

### 1.4 Program Functionality

The function provided is summing up the elements of two arrays until the end of one of the arrays is reached. The function will return -1 if one of the arrays has 0 elements, otherwise it will return 0 upon successful completion.

## **2 RISC-V Tool Chain**

To be done.

## References

- [1] James. What is a branch delay slot and why is it used?, 2020. Accessed: June 10, 2024.