Project 1 Exectuion 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 and its binary fields. Based on the cheat sheet provided in the assignment description [1], a Table 1 is provided in order to give the instruction format, register numbers and their symbolic name.

Table 1: Code listing with hex instructions and binary fields

Mem Address	Hex Instruction	Format	Binary Instruction	ASM Instruction
0x0	0x00050893	I	00000000000000101000010001001 0011	addi a7, a0, 0
0x4	0x00068513	I	00000000000000110100001010001 0011	addi a0, a3, 0
0x8	0x04088063	SB	00000100000010001000000001100011	beq a7, zero, 64
0xc	0x04058263	SB	00000100000001011000001001100011	beq a1, zero, 68
0x10	0x04060063	SB	00000100000001011000001001100011	beq a2, zero, 64
0x14	0x04d05063	SB	00000100110100000101000001100011	bge zero, a3, 148
0x18	0x00088793	I	00000000000010001000011110010011	addi a5, a7, 0
0x1c	0x00269713	I	00000000001001101001011100010011	addi a4, a3, 2
0x20	0x00e888b3	R	00000000111010001000100010110011	add a7, a7, a4
0x24	0x0007a703	I	00000000111010001000100010110011	lw a4, 0(a5)
0x28	0x0005a803	I	00000000000001011010100000000011	lw a6, 0(a1)
0x2c	0x01070733	R	00000001000001110000011100110011	add a4, a4, a6
0x30	0x00e62023	S	00000000111001100010000000100011	sw a4, 0(a2)
0x34	0x00478793	I	00000000010001111000011110010011	addi a5, a5, 4
0x38	0x00458593	I	00000000010001011000010110010011	addi a1, a1, 4
0x3c	0x00460613	I	00000000010001100000011000010011	addi a2, a2, 4
0x40	0xff1792e3	SB	11111111000101111001001011100011	bne a5, a7, -16
0x44	0x00008067	I	00000000000000001000000001100111	jalr zero, ra, 0
0x48	0xfff00513	I	11111111111100000000010100010011	addi a0, zero, -1
0x4c	0x00008067	I	00000000000000001000000001100111	jalr zero, ra, 0
0x50	0xfff00513	I	11111111111100000000010100010011	addi a0, zero, -1
0x54	0x00008067	I	00000000000000001000000001100111	jalr zero, ra, 0

The branch delay slot concept is interesting when discussing pipelined processors. Esentially, when a branch instruction is taken, the instructions that were fetched after the branch instructions might become invalid, if the branch is taken. Therefore, 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 in pipelined architecture. As explained in [2]: "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." [2]

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 deffered 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.

References

[1] *PR1-SE201-21*, 2021. Unpublished internal document.

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