Computer Architecture Homework 4

110062219

June 4, 2023

Contents

1 Assembly Coding

1.1 Evidence of Correctness

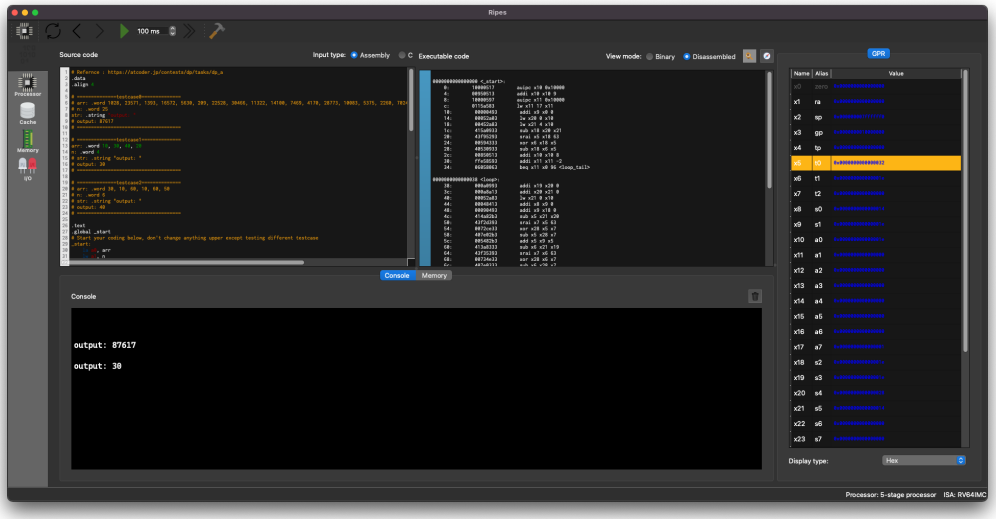


Figure 1: Testcase 1

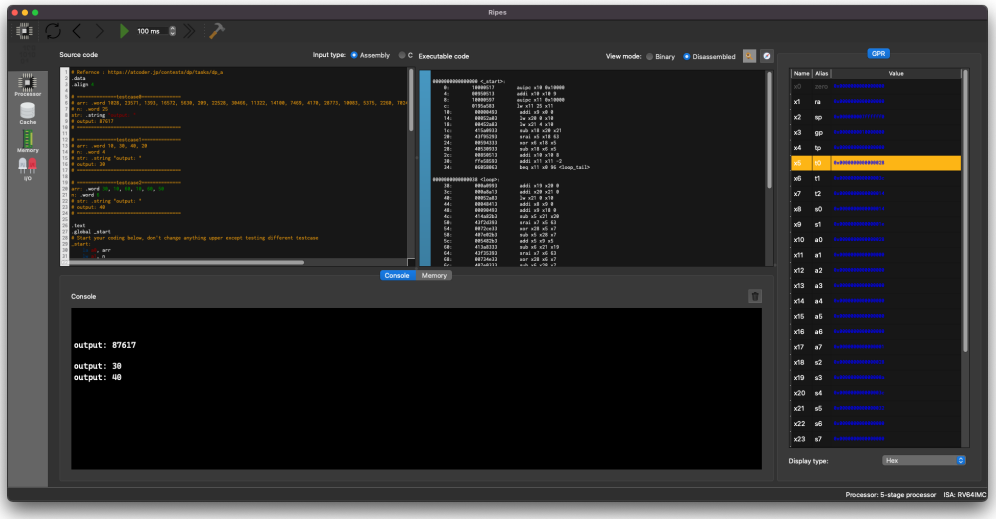


Figure 2: Testcase 2

In addition, I generated a testcase of length 25 and also pass it.

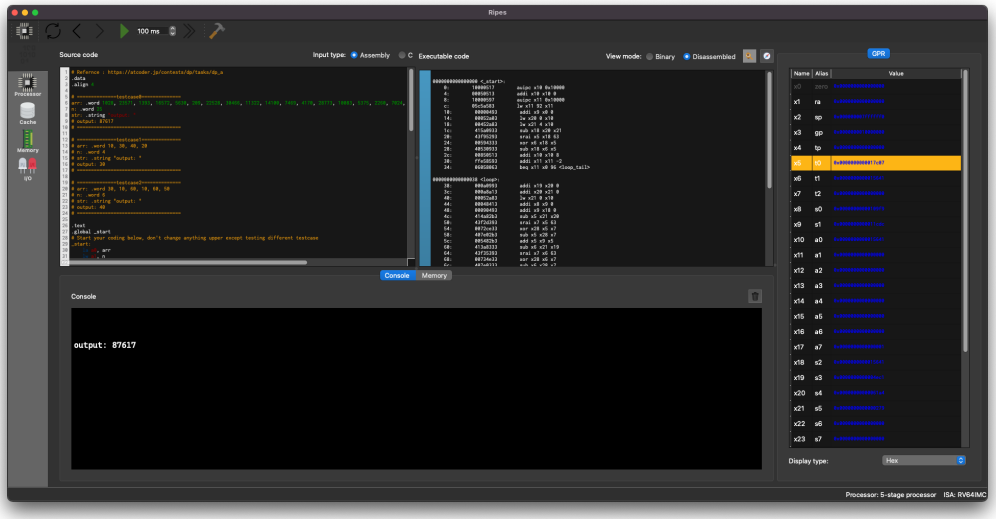


Figure 3: Testcase 0

In fact, I came up with the assembly codes from C codes degree by degree. So I submitted my vary low-level, assembly-like, i.e., to name variables after registers and use `goto` and labels instead of loops, C codes to AtCoder and got accepted to ensure the correctness.

1.2 Flowchart

First, I store the address of *arr* in `a0` and n in `a1`. By means of technique similar to **rolling array**, we needn't maintain the whole *dp*. Instead, we only need keep track of the 3 recent elements of *dp*, which are stored in `s0`, `s1` and `s2`, respectively. Likewise, I only record the 3 recent elements of *arr* in `s3`, `s4` and `s5`, respectively, and advance `a0`, recede `a1` each time in loops.

The most notable part of my codes is that, as a competitive programmer, I tend to calculate the minimum and abstract values by fancy and elegant **bitwise operations** rather than **branches**. Due to 2's complement, we have

$$\min\{l, r\} = r \oplus ((l \oplus r) \wedge -(l < r))$$

and

$$|x| = (x \oplus (x \gg (\text{sizeof}(x) \times \text{BYTE\_BITS} - 1))) - (x \gg (\text{sizeof}(x) \times \text{BYTE\_BITS} - 1))$$

.

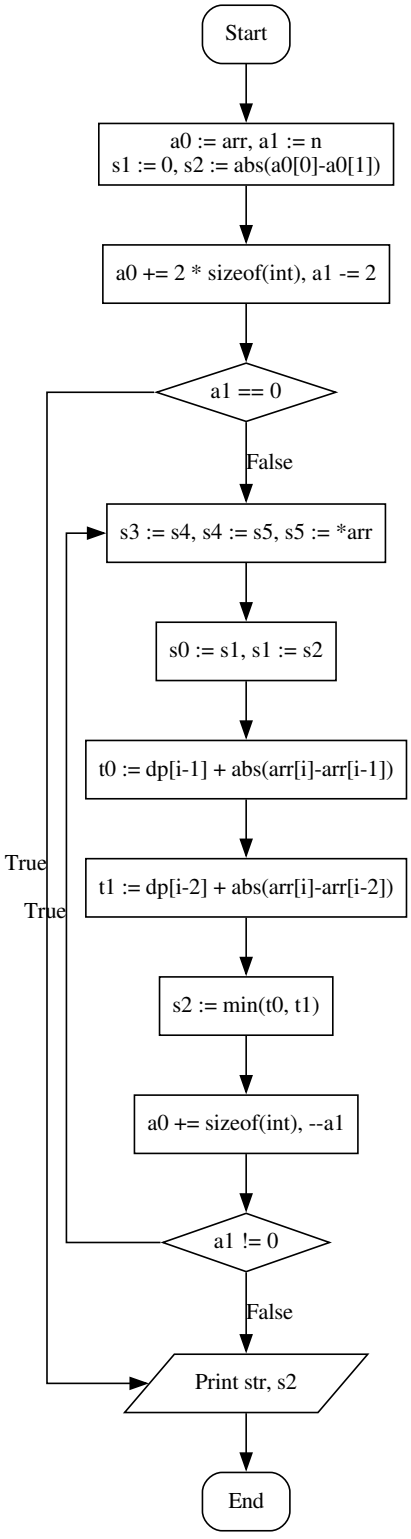


Figure 4: Flowchart

2 Hazards in Codes

2.1 R-type RAW (EX)

The `srai` instruction at line 38 reads the register `s2` as its first operand, which is written by the `sub` instruction at line 36. As a consequence, an EX hazard would occur.

In this case, the forwarding unit would set the signal `alu_reg1_forwarding_ctrl` to be `MemStage`.

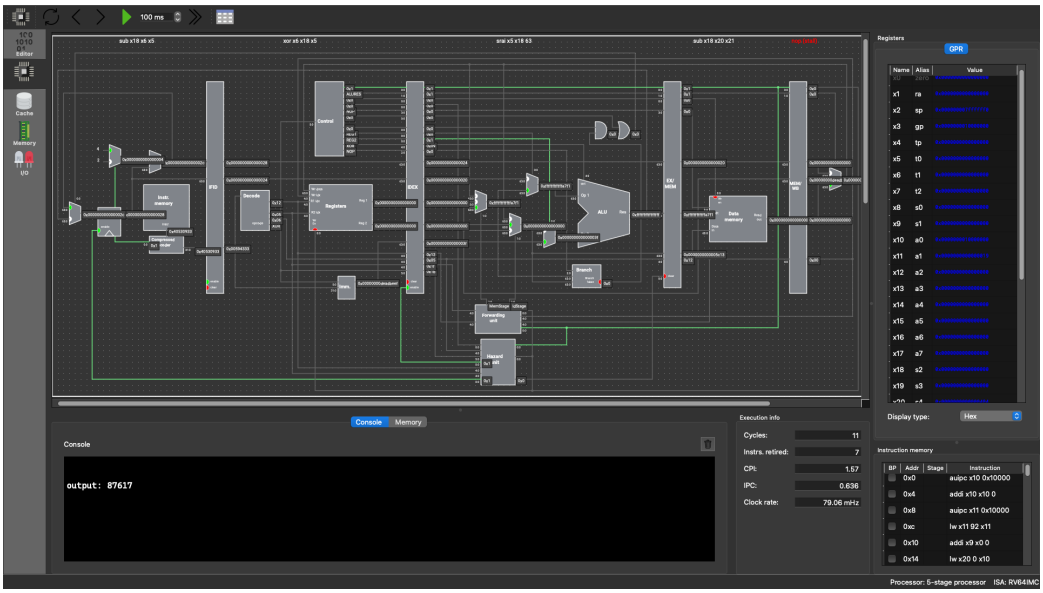


Figure 5: Type 1

2.2 R-type RAW (MEM)

The `xor` instruction at line 39 reads the register `s2` as its first operand, which is written by the `sub` instruction at line 36. Therefore, a MEM hazard would occur.

In this case, the forwarding unit would set the signal `alu_reg1_forwarding_ctrl` to be `WbStage`.

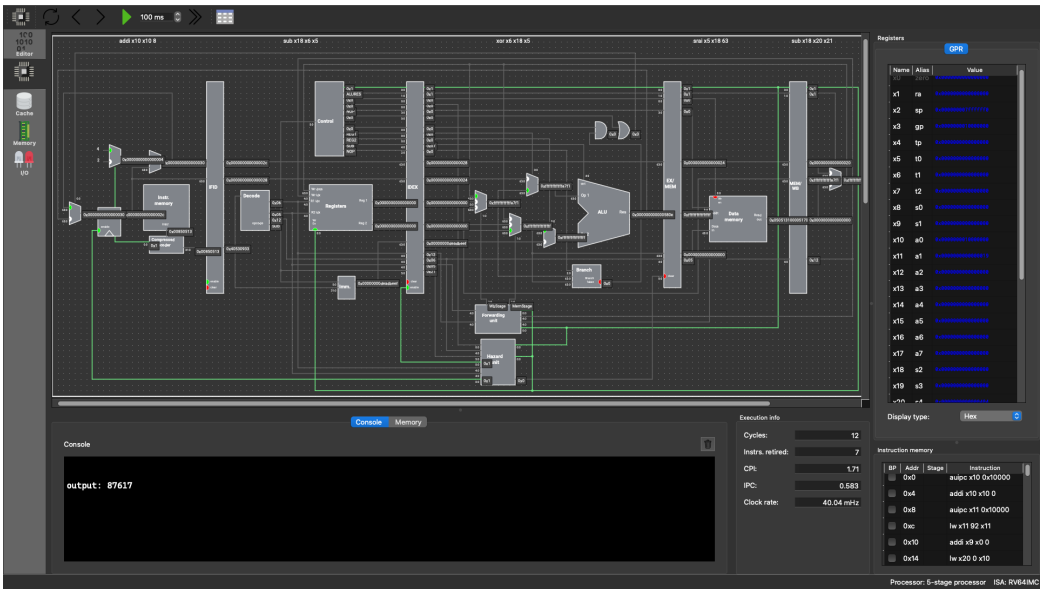
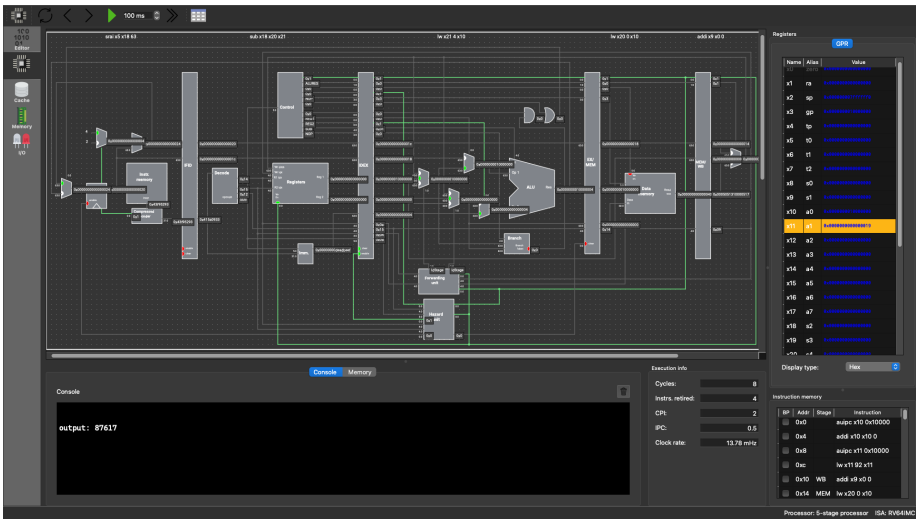


Figure 6: Type 2

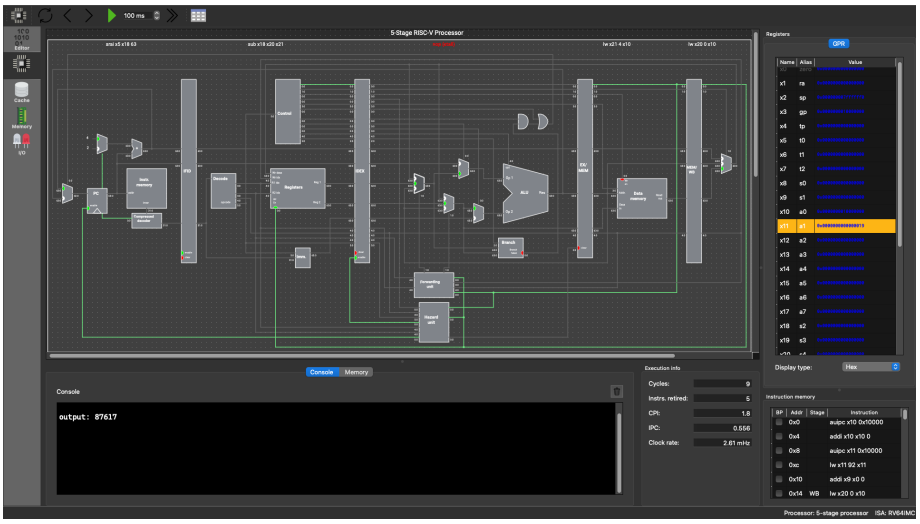
2.3 Load RAW Immediately

The `sub` instruction at line 36 reads the register `s5` as its second operand, which is written by the `lw` instruction at line 35. Accordingly, a load-use hazard would occur.

In this case, the hazard unit have to reset the signal `hazardFEEnable` to stall the pipeline, inserting a `nop` bubble between the two instructions. And then, in the next cycle, the forwarding unit would set the signal `alu_reg2_forwarding_ctrl` to be `WbStage`.



(a)



(b)

Figure 7: Type 3

2.4 Load RAW Between One Instruction

The `sub` instruction at line 57 reads the register `s5` as its first operand, which is written by the `lw` instruction at line 52. (There are 2 blank lines and a comment line in between.) Whence a hazard would occur.

In this case, the forwarding unit would set the signal `alu_reg1_forwarding_ctrl` to be `WbStage`.

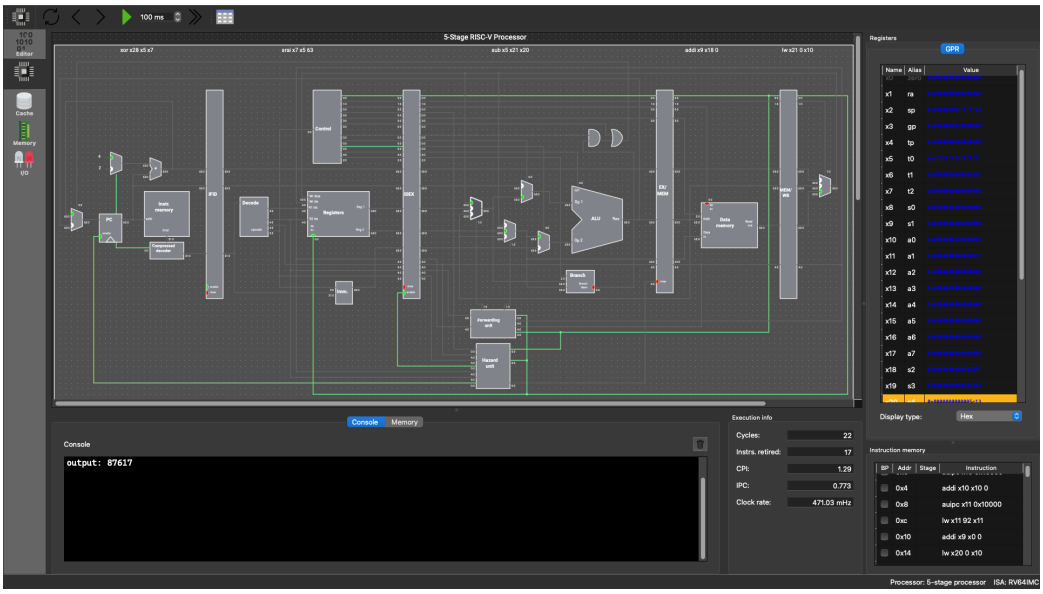
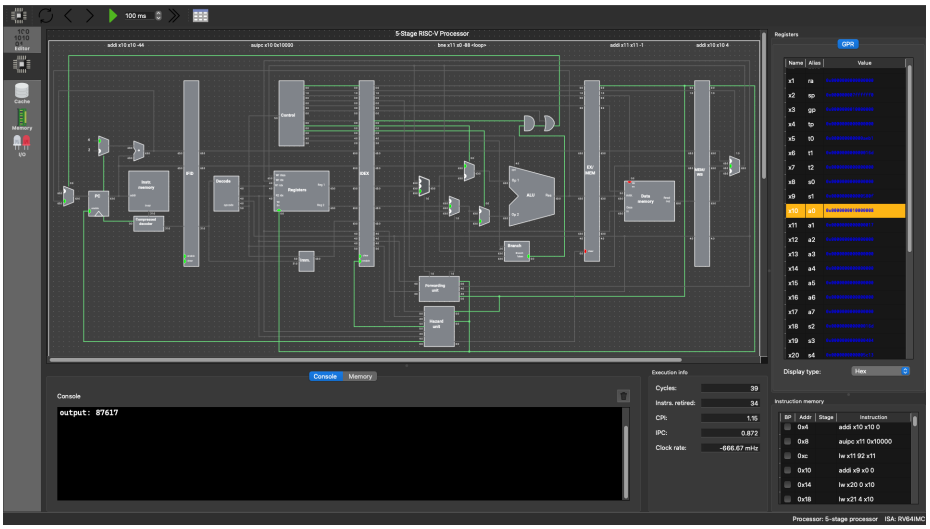


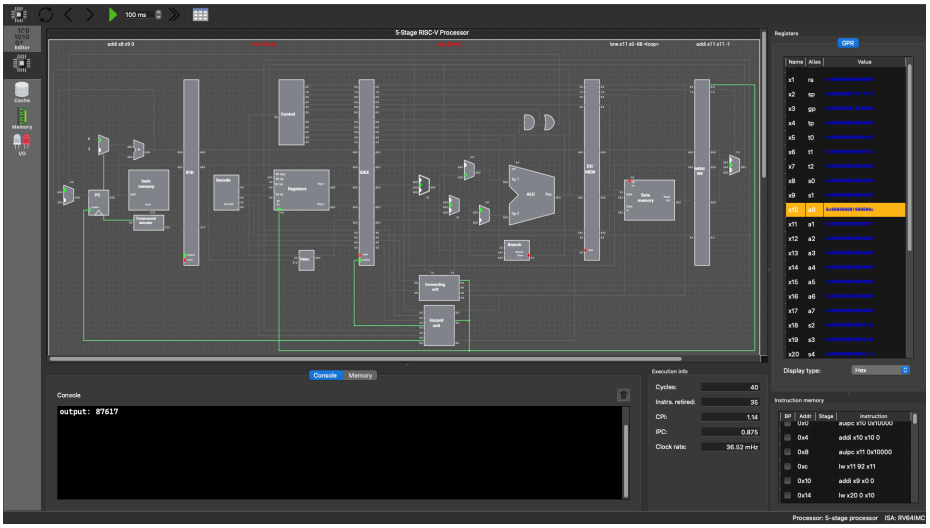
Figure 8: Type 4

2.5 Branches

Ripes happens to be of the **always-not-taken** policy. So when branches taken place actually, for instance at line 80, then the `clear` of `ID/EX` and `IF/ID` must be triggered and the pipeline is flushed thereby.



(a)



(b)

Figure 9: Type 5