232 Project Journal – Julian (Orange):

10/03/2023	Translated Euclid Algorithm to RISC-V. (1.1) Retrieved required Instructions from RISC-V Code. (1.2) Thought of optimization strategies in terms of new instructions to minimize code. Could not find anything. (1.3)
10/04/2023	Meeting with Prof. Williamson. Changed Instruction Set Literals. Thought of ways to minimize Code length… still no progress. Seems minimal.

10/05/2023	Rewriting Euclid Code in New Instruction Format. Got an idea! Maybe let ra be x1 and sp x2. Then for /\ instructions to jump back to caller function, combine sp break down and Jump! No clue if it is better to implement as pseudo instruction or basic instruction. Since two arithmetic ops are needed, a lot of cycles are required to fetch values from registers, add and save back in registers. (2.1) New English Descriptions for Instruction set.
10/08/2023	Implementation of jump instruction: use immediate of /\ inst for stack break down. Take immediate and increase rd+1 in each case. Speeds up jump back to caller. If no register should be increased, provide immediate `0x0`.
10/09/2023	Continued with exemplary instructions.
10/10/2023	Jump RTL implementation + fixing instruction fetch RTL. Full code translation to machine code.
10/11/2023	Worked on Components Table.

Appendix:

1.1

```
relPrime:   addi sp, sp, -16
            sw ra, 0(sp)
            sw s0, 4(sp)
            sw s1, 8(sp)
            addi s0, x0, 2
            addi s1, x0, 1
```

```
            sw a0, 12(sp)
loop:       addi a1, s0, 0
            lw a0, 12(sp)
            jal ra, gcd
            beq a0, s1, done
            addi s0, s0, 1
            jal x0, loop
done:       addi a0, s0, 0
            lw ra, 0(sp)
            lw s0, 4(sp)
            lw s1, 8(sp)
            addi sp, sp, 16
            jalr x0, ra
```

```
gcd:        addi sp, sp, -4
            sw ra, 0(sp)
            beq a0, x0, returnb
loop:       beq a1, x0, returna
            blt a1, a0, agreater
            sub a1, a1, a0
            jal x0, loop
agreater:   sub a0, a0, a1
            jal x0, loop
returnb:    addi a0, a1, 0
returna:    lw ra, 0(sp)
            addi sp, sp, 4
            jalr x0, ra
```

1.2

Required Instructions:

```
addi rd, rs, imm
sub rd, rs1, rs2
sw rd, imm (rs)
lw rd, imm (rs)
beq rs1, rs2, imm
blt rs1, rs2, imm
jal rd, imm
jalr rd, imm
```

1.3

Optimization: Do not assign zero register but rather use value 1 in `x0`.
Enables imm values for stack frame building by using shift instructions.
Also enables beq without first assigning value to register.
Introducing, bez (branch equal zero) instruction.

2.1

```
/\: R[rd+1] += SE(imm) then PC = R[rd] + SE(imm)
```