KOÇ
ÜNİVERSİTESİ

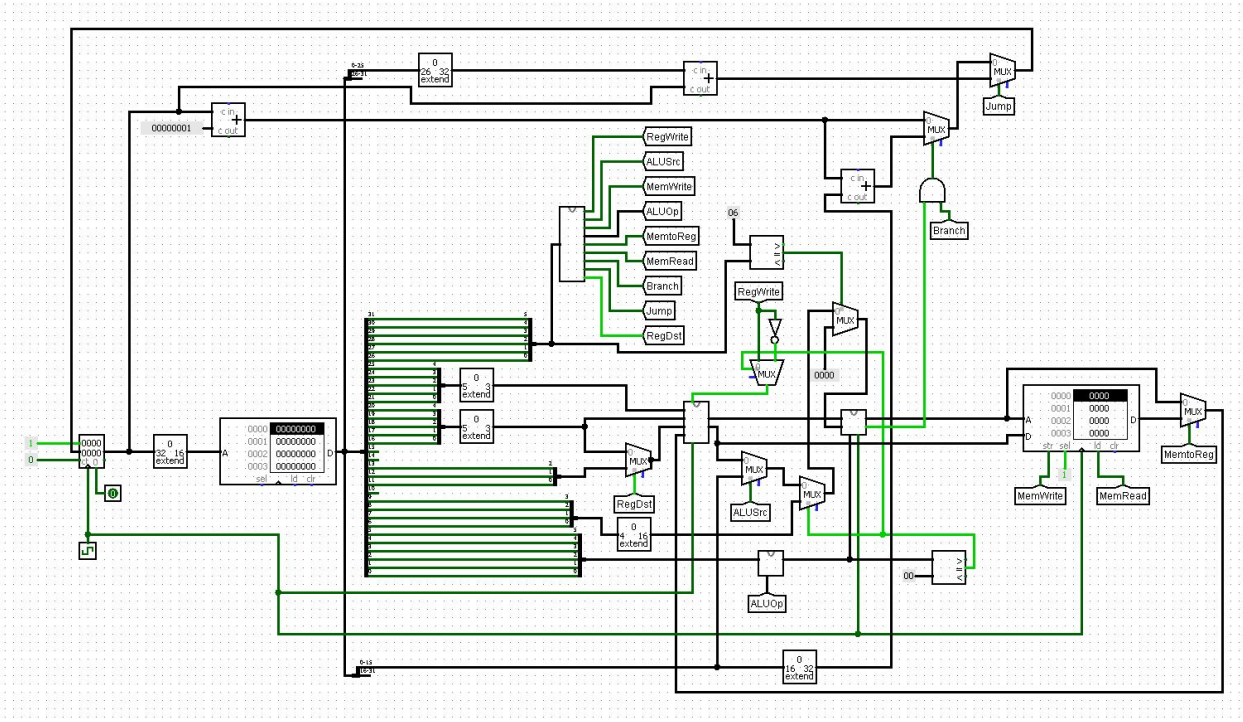
# COMP 303 – Term Project Report

# Single Cycle Processor Design

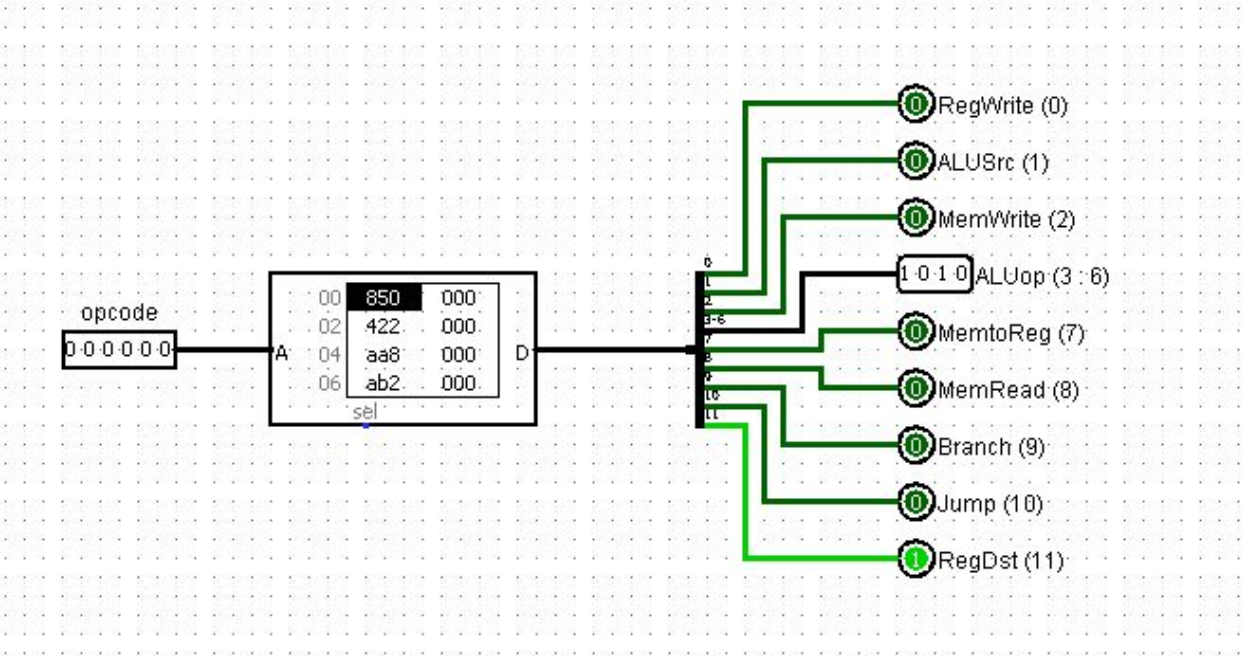
Mediha Meltem Gunay – 0054120

Eren Limon – 0054129

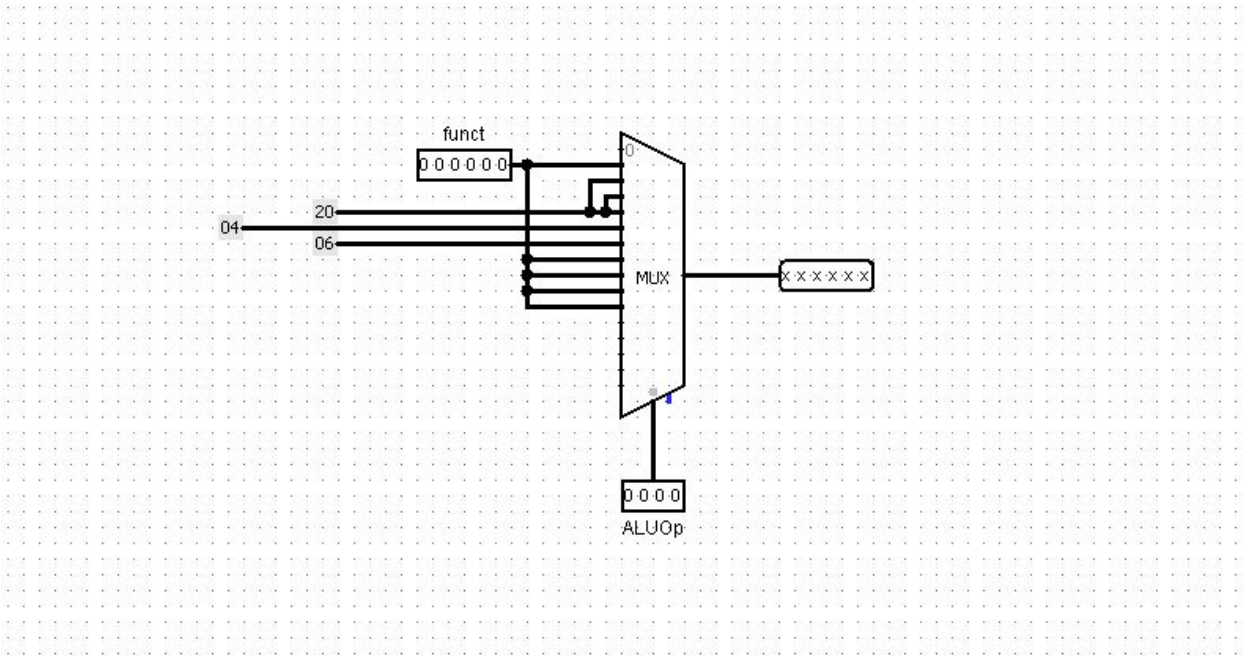
## Circuit Schematics



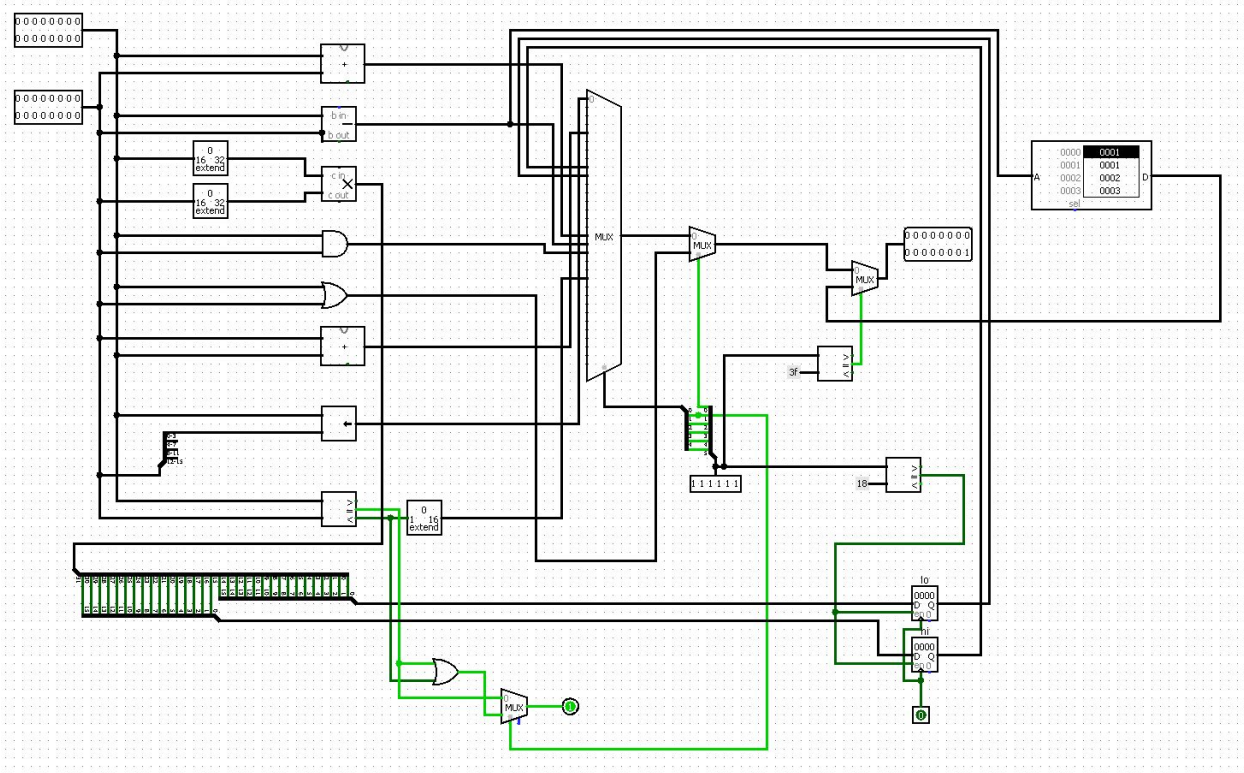

*Main CPU Circuit*




*Control Unit*




*ALU Control Unit*




*ALU*

**Description of Our Design and Our Custom Instruction**

In our design, we wanted to follow the book and tried to make it as much as similar to the way it seems on the book. To do that, we made sure that we have all control units (both Control Unit and ALU Control Unit) and all multpilexers located in the same places. For some part, we had to add extra multiplexers to the main circuit as not described in the book, but we our design works fine as well. For the control unit, after we made the control table, we used a ROM memory to map opcode input to the control signal outputs. This mapping could have been done via logic gates, but using a memory (like a HashMap data structure) sounded much more practice for us. For the ALU Control Unit, we used a multiplexer for the mapping process this time. Because the most of the time we are directly sending the funct code as a control input to our ALU, using a ROM was redundant. After the completion of our control unit, we designed the PC part and placed our instruction memory as a RAM. In this part, it is important to point out that our RAMs addresses increments 1 by 1. So, there is no such thing like “word aligned”. That’s why we are incrementing our PC by 1 in the standard condition.

Our custom instruction is defined as follows:

subFib $1 $2 $3

*Subtract the numbers in the registers $2 and $3. Let the difference be n. Then place the nth fibonacci number to the $1. (0 <= n <= 23)*

To do this, we again used the ROM for the mapping process. (We are mapping the order of the fibonacci number and the fibonacci number) Our ROM supports only first 24 fibonacci number, so if the difference is bigger than 23, it shows zero as default. Using ROM might be tricky, but for someone that does not see the implementation, it sounds cool.

| Type | Instruction | Opcode | RegDst | Jump | Branch | MemRead | MemToReg | AluOp | MemWrite | AluSrc | RegWrite | HexadecimalCode | Funct |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | add | 111110 | 1 | 0 | 0 | 0 | 0 | 0001 | 0 | 0 | 1 | 0809 | 100000 |
| R | sub | 111110 | 1 | 0 | 0 | 0 | 0 | 0001 | 0 | 0 | 1 | 0809 | 100010 |
| R | and | 111110 | 1 | 0 | 0 | 0 | 0 | 0001 | 0 | 0 | 1 | 0809 | 100100 |
| R | or | 111110 | 1 | 0 | 0 | 0 | 0 | 0001 | 0 | 0 | 1 | 0809 | 100101 |
| R | slt | 111110 | 1 | 0 | 0 | 0 | 0 | 0001 | 0 | 0 | 1 | 0809 | 000000 |
| R | mult | 011000 | x | 0 | 0 | 0 | x | 0111 | 0 | 0 | 0 | 0868 | 011000 |
| R | mfhi | 010000 | x | 0 | 0 | 0 | 0 | 1000 | 0 | x | 1 | 0843 | 010000 |
| R | mflo | 010010 | x | 0 | 0 | 0 | 0 | 1001 | 0 | x | 1 | 0846 | 010010 |
| R | sll | 000000 | 1 | 0 | 0 | 0 | 0 | 1010 | 0 | 0 | 0 | 0850 | 000000 |
| I | addi | 001000 | 0 | 0 | 0 | 0 | 0 | 0100 | 0 | 1 | 1 | 0023 | - |
| I | lw | 100011 | 0 | 0 | 0 | 1 | 1 | 0010 | 0 | 1 | 1 | 0193 | - |
| I | sw | 101011 | x | 0 | 0 | 0 | x | 0011 | 1 | 1 | 0 | 089e | - |
| I | beq | 000100 | x | 0 | 1 | 0 | x | 0101 | 0 | 0 | 0 | 0aa8 | - |
| I | blez | 000110 | x | 0 | 1 | 0 | x | 0110 | 0 | 1 | 0 | 0ab2 | - |
| J | jump | 000010 | 0 | 1 | 0 | 0 | 0 | 0100 | 0 | 1 | 0 | 0422 | - |
| R | My Ins | 111111 | x | 0 | 0 | 0 | 0 | 0001 | 0 | 0 | 1 | 0809 | 111111 |