TASK 1: TO RUN BUBBLE SORT ALGORITHM ON SINGLE CYCLE PROCESSOR

https://edaplayground.com/x/phDW

APPROACH AND WORKING

In order to run bubble sort we made changes in the single cycle processor implemented in lab 11. The modules in this processor are:

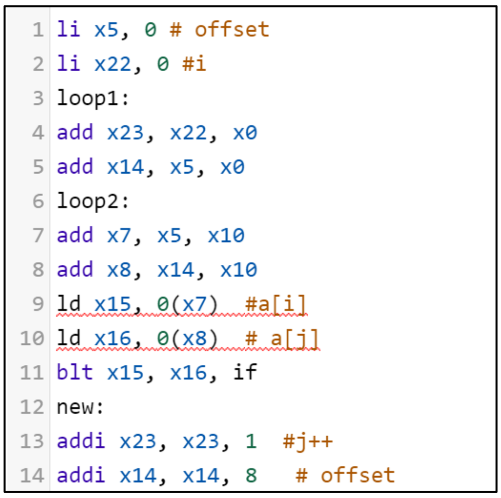
- Immediate Generator
- MUX_64
- Instruction Parser
- Register File
- ALU_64
- Instruction Memory
- Data Memory
- PC Counter
- Adder
- Control Unit
- ALU Control
- Comparator

FOLLOWING ARE SOME OF THE MODIFICATIONS WE DID:

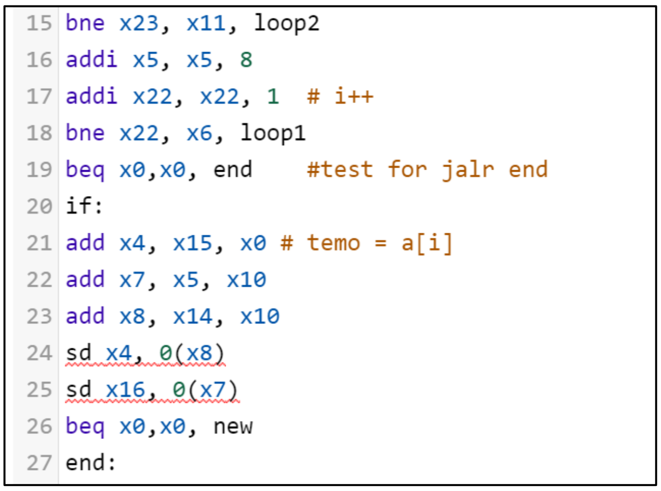
1) The Instruction memory was populated with the instructions that are used for bubble sorted. These are the one that we did in Lab 4. We wrote code for bubble sort algorithm on Venus and copied their instructions and placed them in the instruction's memory. We also made the following changes in the bubble sort algorithm code:
   a. JAL and JALR were and implemented without it without a function.
   b. mv instructions were replaced by add instructions.
   c. SW and LW were replaced by SD and LD
2) In Data Memory, we initialized an array of random numbers that were to be sorted and printed the arrays in log.

3) The most important task was to make sure that BEQ, BNE and BLT instructions work fine, for which we created a comparator. The function of the **comparator** was to compare two values and check if the PC has to be incremented by 4 or some immediate values. It assigns 1 to a wire if two values are equal, not equal or one is less then other. This wire is AND with BRANCH signal and if the output is 1, then we have PC + immediate.

BELOW IS CODE FOR BUBBLE SORT ALGORITHM:

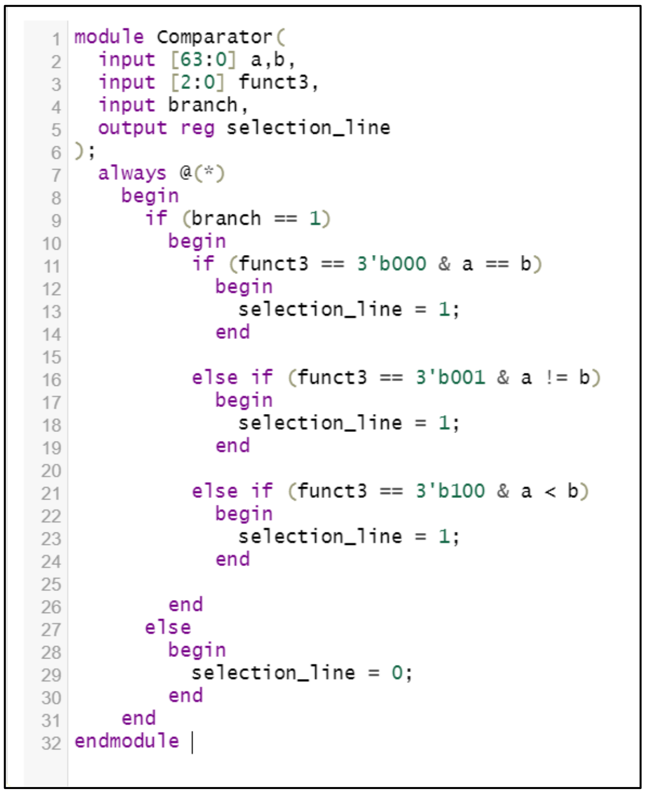


```
li x5, 0 # offset
li x22, 0 #i
loop1:
add x23, x22, x0
add x14, x5, x0
loop2:
add x7, x5, x10
add x8, x14, x10
ld x15, 0(x7)  #a[i]
ld x16, 0(x8)  # a[j]
blt x15, x16, if
new:
addi x23, x23, 1  #j++
addi x14, x14, 8   # offset
```



```
bne x23, x11, loop2
addi x5, x5, 8
addi x22, x22, 1  # i++
bne x22, x6, loop1
beq x0,x0, end    #test for jalr end
if:
add x4, x15, x0 # temo = a[i]
add x7, x5, x10
add x8, x14, x10
sd x4, 0(x8)
sd x16, 0(x7)
beq x0,x0, new
end:
```

THE CODE FOR COMPARATOR IS GIVEN BELOW:



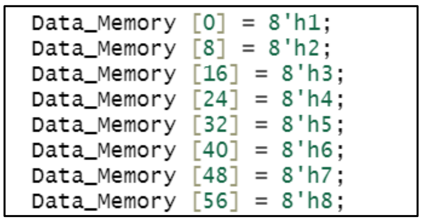
```
module Comparator(
  input [63:0] a,b,
  input [2:0] funct3,
  input branch,
  output reg selection_line
);
  always @(*)
    begin
      if (branch == 1)
        begin
          if (funct3 == 3'b000 & a == b)
            begin
              selection_line = 1;
            end

          else if (funct3 == 3'b001 & a != b)
            begin
              selection_line = 1;
            end

          else if (funct3 == 3'b100 & a < b)
            begin
              selection_line = 1;
            end

        end
      else
        begin
          selection_line = 0;
        end
    end
endmodule
```

Figure b: Code for comparator

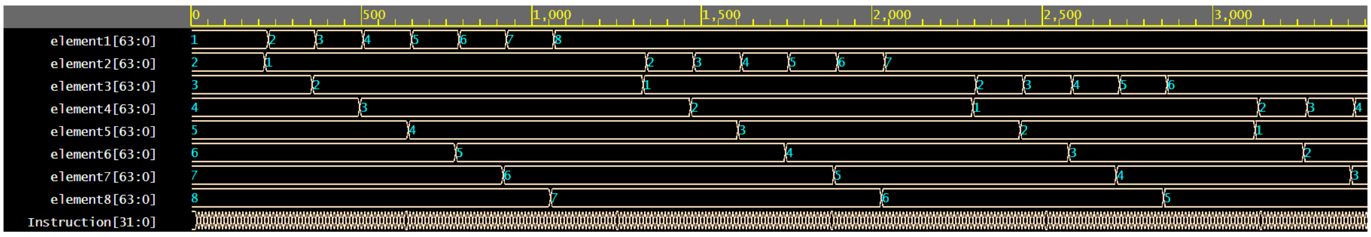


```
Data_Memory [0]  = 8'h1;
Data_Memory [8]  = 8'h2;
Data_Memory [16] = 8'h3;
Data_Memory [24] = 8'h4;
Data_Memory [32] = 8'h5;
Data_Memory [40] = 8'h6;
Data_Memory [48] = 8'h7;
Data_Memory [56] = 8'h8;
```

Figure a: Initialize array

RESULTS AND OUTPUTS

The sorting works fine for different inputs but it takes around 5000ns. On EP Wave it shows till 3455 ns:



We can see that elements are changing their positions, it can be better visualized by printing the arrays, like shown below;

```
Initial array:    # '{8, 7, 6, 5, 4, 3, 2, 1}

Sorted array:    # '{1, 2, 3, 4, 5, 6, 7, 8}
```