Design of a 32 Bit Processor

Verilog Assignment - Final

Group 14
Gummadi Vikhyath - 21CS10028
Pasupulety Chethan Krishna Venkat – 21CS30036

6. Bubble Sort demonstration on Vivado

In this section, we are going to use bubble sort to sort 10 integers.
The program implements the bubble sort algorithm to sort a set of 10 integers stored in the array {arr} .
The array, starting at address 100, contains the elements the following elements:
[15, 35, 5, 45, 75, 50, 66, 88, 109, 90].
In the testbench, the entire array is loaded into Data Memory for sorting, while initializing the Datamemory, we will initialize it with the above values in the following method :

```
module data_mem (
    input wire [9:0] inaddress,
    input wire clk,reset,
    input wire write,read,
    input wire [31:0] addr,
    input wire [31:0] din,
    output wire [31:0] dout,
    output wire [15:0] outdata
);

reg signed [31:0] dmem[0:1023];

initial
begin
 dmem[100] = 15;
 dmem[101] = 35;
 dmem[102] = 5;
 dmem[103] = 45;
 dmem[104] = 75;
 dmem[105] = 50;
 dmem[106] = 66;
 dmem[107] = 88;
 dmem[108] = 109;
 dmem[109] = 90;
end
```

Now, we will give the instructions in the instructions set, For that the code logic of bubble sort will be as follows:

This assembly code implements the bubble sort algorithm for an array of size 10. It uses registers R1 to R12 for various purposes, including array manipulation and loop control. The outer loop (X) iterates over the array, and the inner loop (Z) compares adjacent elements and swaps them if necessary (Y). The sorting process continues until all elements are in order. The program exits after completing the sorting process.

```
/* Instructions According to Processor */

ADDI  R1, R0, #100;     // arr[0]
MOVE  R2, R0;           // i=0
MOVE  R3, R0;           // j=0
ADDI  R4, R0, #10;      // n=10
ADDI  R5, R0, #10;      // n-i for inner loop
MOVE  R6, R1;           // for iterating addr by i
MOVE  R7, R1;           // for iterating addr by j
SUBI  R4, R4, #1;
MOVE  R3, R0;           // outer_loop // j=0                (X)
SUBI  R5, R5, #1;       // decreasing size for inner_loop
ADD   R7, R0, R1;       // resetting addr itr j
LD    R8, 0(R7);        // inner_loop // arr[j]             (Z)
ADDI  R7, R7, #1;       // addr itr j += 1
LD    R9, 0(R7);        // arr[j+1]
ADDI  R3, R3, #1;       // j++
SUB   R10, R8, R9;      // R10 = R8 - R9;
BMI   R10, #3;          // if R8 < R9 then Branch           (Y)
ST    R8, 0(R7);        // swap
ST    R9, -1(R7);
LD    R9, 0(R7);
SUB   R11, R3, R5;                                         (Y)
BZ    R11, #1;          // Exiting from inner_loop          (W)
BR    #-12;             // Address to inner_loop            (Z)
ADDI  R2, R2, #1;       // After  Exiting From inner_loop   (W)
SUB   R12,R2, R4;       // i!=n
BZ    R12, #1;          // Program Completed                (EXIT)
BR    #-19;             // outer_loop                       (X)
HLT;                                                       (EXIT)
```

6. GCD Demonstration on Vivado and FPGA

In this section, we are going to calculate the GCD of 2 numbers and also parallely show the output in the test FPGA unit.
To find the greatest common divisor (GCD) of two numbers, we'll use registers R1 and R2 to hold the first and second numbers, respectively.
Additionally, we'll store these values in data memory[0] and data memory[1] respectively.

Finally, the calculated GCD will be stored in data memory[2]. and can be displayed below :

```
dmem[0] = 39;  // Number 1 - Initially stored in R1
dmem[1] = 52;  // Number 2 - Initially stored in R2

// GCD is calculated

dmem[2] = 13;  // Output GCD stored in this location and displayed
```

Logic for calculating GCD and the corresponding code which is to be converted into the instruction memory for 32 bit instructions is given below.

The following assembly code calculates the greatest common divisor (GCD) of two numbers, 20 and 30. It uses a loop to iteratively subtract the smaller number from the larger one until they are equal (Z). The GCD is then stored in memory at location 2. The program terminates after finding the GCD.

```
/* Instructions According to Processor */

ADDI R1, R0, #39;
ADDI R2, R0, #52;
ST   R1, 0(R0);
ST   R2, 1(R0);
SUB  R3, R1, R2;              (W)
BMI  R3, #3;      // if R1<R2 (X)
BPL  R3, #6;      // if R1>R2 (Y)
BZ   R3, #8;      // if R1=R2 (Z)
HLT;
SUB  R3, R2, R1; // if R1<R2 (X)
MOVE R2, R3;
BR   #-8;                     (W)
HLT;
MOVE R1, R3;      // if R1>R2 (Y)
BR   #-11;                    (W)
HLT;
ST   R1, 2(R0);  // if R1=R2 (Z)
HLT;
```

The following is the test bench, First Number 1 will be displayed followed by number 2, And then finally the GCD (Result) will be displayed.

```
inaddress=0;
       #20
       $display("mem[0]=",outdata);

       inaddress=1;
       #20
       $display("mem[1]=",outdata);

       inaddress=2;
       #20
       $display("mem[2]=",outdata);
```