Assignment No. 7

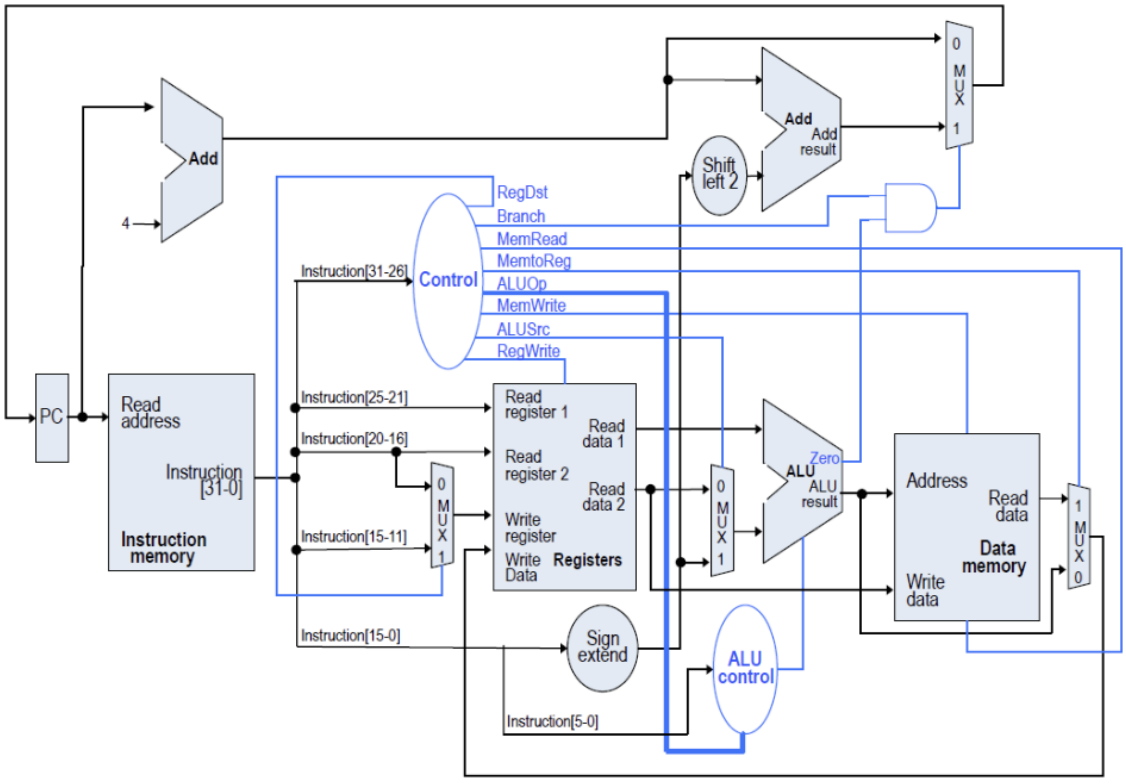
Computer Organisation - CS220

Done by: **Rohan Ravi (210870)** **Abhinav Garg (210029)**

OVERVIEW

This assignment requires us to build a single-cycle MIPS processor. The processor's word size is 32 bits.

We have followed the following diagram extensively to design the processor:



The design of the processor

HOW TO RUN THE CODE

Use the following commands inside the project file location:

```
cd 'folder path'
```

```
iverilog -o out.vvp tb.v
vvp out.vvp
gtkwave out.vcd`
```

GOALS

A single-cycle MIPS processor (CSE_BUBBLE) with the following features:-

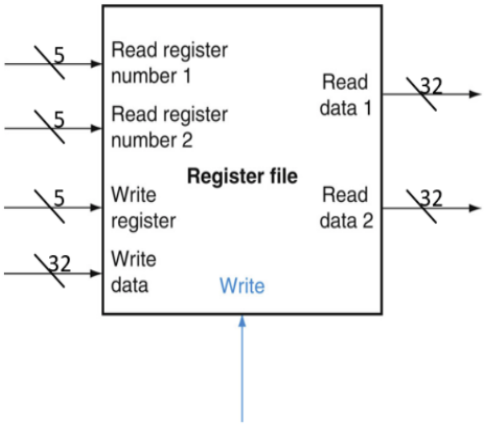
1. 32 bits word size.
2. VEDA memory architecture for : Instruction memory, Data Memory, both should be non-overlapping.
3. A register file: (RISC_BUBBLE): with 32 registers, where certain registers have the same role as in MIPS-32-ISA.
4. Design op-code formats for the processor, to decide the data path.

STAGES OF DEVELOPMENT

[PDS1] Decide the registers and their usage protocol

The register file has 32 registers, where each register has a specific role.

Register number	**Roles**
0	$zero register: stores constant ‘0’
1-9	$s0 to $s8 : Saved registers
10-18	$t0 to $t8 : Temporary registers
19-20	Kernel registers
21-23	$v0 to $v2: Result registers
24-31	Other registers with some roles.



The design of the register file.

RISC_BUBBLE.v

```v
//THIS IS THE REGISTER FILE WITH 32 registers
module RISC_BUBBLE (
    input         clk  ,
    //
    input  [ 4:0] addr1,
    input  [ 4:0] addr2,
    output [31:0] data1,
    output [31:0] data2,
    //
    input         write_enable   ,
    input  [ 4:0] addr3,
    input  [31:0] wdata
);

    reg [31:0] regmem[31:0];
    reg [31:0] data1       ;
    reg [31:0] data2       ;

    always @(addr1 or regmem[addr1]) begin
        if (0 == addr1) begin
            data1 = 0;
        end else begin
            data1 = regmem[addr1];
        end
    end

    always @(addr2 or regmem[addr2]) begin
        if (0 == addr2) begin
            data2 = 0;
        end else begin
            data2 = regmem[addr2];
        end
    end

    always@ (posedge clk) begin
        if(1'b1 == write_enable) begin
            regmem[addr3] = wdata;
        end
    end

endmodule

```

[PDS2] Decide upon the size for instruction and data memory in VEDA.

Size of instruction memory: 32x64: Instruction memory has a size to store 64 instructions at a time, where each instruction is of size 32 bits. This is a read only memory for the current implementation.

Size of data memory: 32x1024: Data memory has a size to store 1024 words at a time, where each word is of size 32 bits. This is a read + write memory for the current implementation.

[PDS3] Design the instruction layout for R-, I- and J-type instructions and their respective encoding methodologies

Class	Instruction	Meaning	Opcode (6 bit)
Arithmetic	add r0, r1, r2	r0=r1+r2	000000
	sub r0, r1, r2	r0=r1-r2	001000
	addu r0, r1, r2	r0=r1+r2 (unsigned addition, not 2's complement)	001001
	subu r0,r1,r2	r0=r1-r2 (unsigned addition, not 2's complement)	001010
	addi r0,r1,1000	r0=r1-r2 (unsigned addition, not 2's complement)	000001
	addiu r0,r1, 1000	r0=r1+1000 (unsigned additon, not 2's complement)	001011
Logical	and r0,r1,r2	r0= r1 & r2	001100
	or r0,r1,r2	r0=r1 | r2	001101
	andi r0,r1, 1000	r0= r1 & 1000	001110
	ori r0,r1, 1000	r0= r1 | 1000	001111
	sll r0, r1, 10	r0=r1<<10 (shift left logical)	000010
Data Transfer	lw r0,10(r1)	r0=Memory[r1+10]	000101

		(load word)	
	sw r0,10(r1)	Memory[r1+10]=r0 (store word)	000100
Conditional Branch	beq r0,r1,10	if(r0==r1) go to PC+4+10 (branch on equal)	000110
	bne r0,r1,10	if(r0!=r1) go to PC+4+10 (branch on not equal)	010000
Unconditional Branch	j 10	jump to address 10	000111
Comparison	slt r0,r1,r2	if(r1<r2) r0=1 else r0=0	000011

GUIDE THROUGH THE CODE / EXPLAINING THE CODE

The machine code instructions are stored in the instrMemory, which are fetched at each clock cycle and stored in the instr_data.

The main processor CSE_BUBBLE has 3 module instantiations: instrMemory, dataMemory and manipulate.

Manipulate takes the connections to instrMemory and datamemory and manipulates (decides) the actions to be done.

Inside the manipulate module the following modules are instantiated: Controller, Datapath.

Controller: takes the instruction >> decodes the command based on the opcode >> triggers on specific control signals.

Datapath: takes the control signals sent by the controller >> updates the program counter OR decides destination register OR writes/loads from data memory, all based on the control signal.

The ALU module takes the control signal `alucontrol` to decode what to do with the given inputs. The ALU has multiple mux which decides the output coming from the alu. Alu output also has branch condition that ANDs with the branch control signal to decide whether the program counter actually branches or not.

MIPS code for bubble sort is written and converted into machine code.

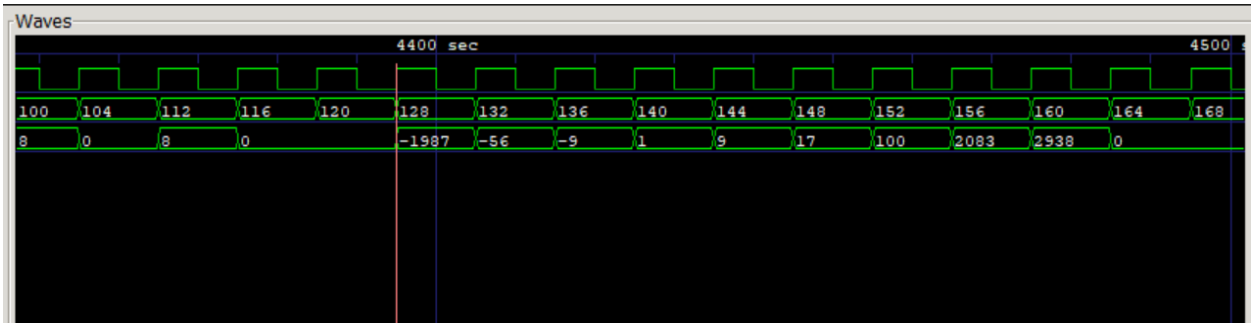
HOW TO CHECK IF IT WORKS

The data memory is first initiated with the input array for BUBBLE SORT in the following order: size of array=9 , size of array-1 = 8 , 9 values stored in the array at zeroth index and first index of the data memory.

The data memory is filled with the following input array:

9, -56, -9, 17, 100, 2938, -1987, 2083, 1

Instruction number 32: is used to load a register with the values in the data memory, just to show that the data memory is now written with the sorted array.



As we can see, the sorted array is:

-1987, -56, -9, 1, 9, 17, 100, 2083, 2938

THANK YOU :)