Assignment-7

CS220

—

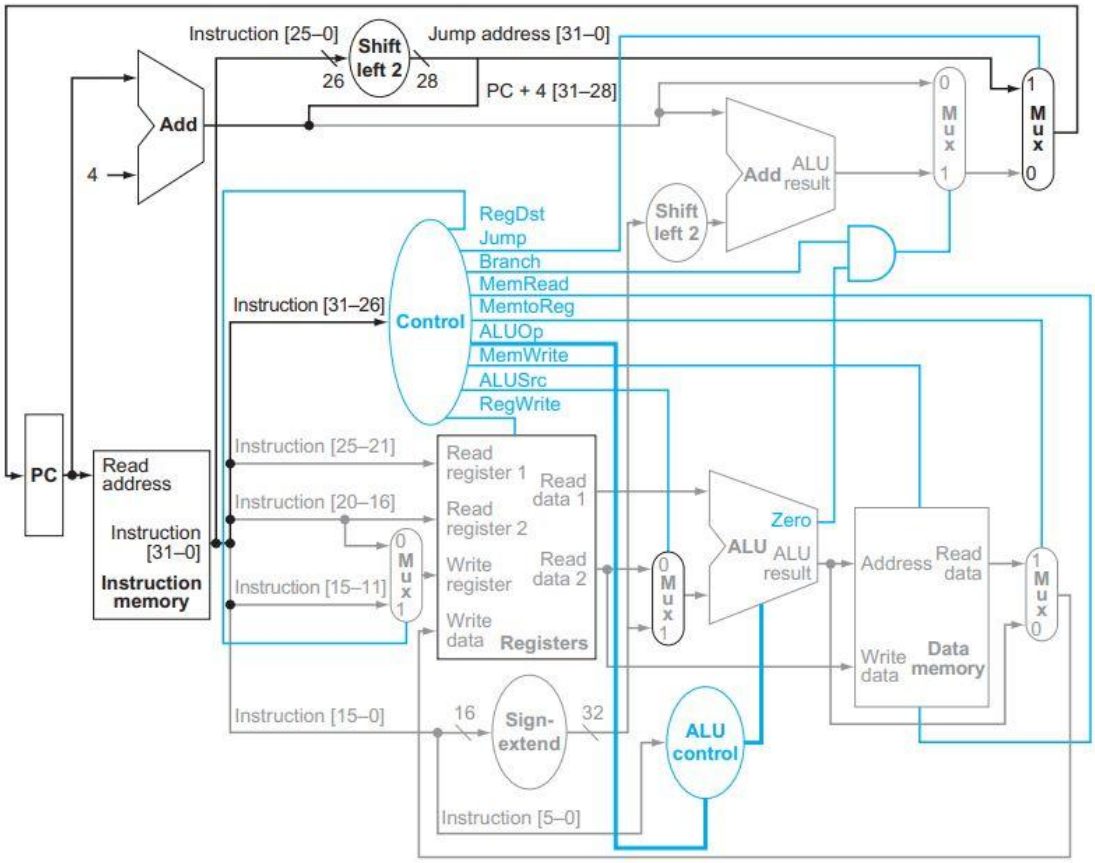
Kalika
210482

Sarthak Kalankar
210935

Overview

We have created a new processor CSE-BUBBLE that has the instruction set architecture as given. The processor word size and instruction size are 32 bits and we use the modified VEDA memory. The CSE-BUBBLE has 32 registers of which certain registers follow the same roles as in MIPS-32 ISA. For example, Program Counter (PC) register holds the address of the next instruction. We have created the op-code formats for the given ISA, decided upon the data path elements (such as addition, subtraction, shift, and jump), and then developed different verilog modules(top, pc, instruction_memory, Veda, Logic_Unit, Control, Data_memory) for the same along with the finite state machine for the control path.

We have referred to the following design for implementing the processor which is also given in the book-



Goals

1. [PDS1] Decide the registers and their usage protocol.
2. [PDS2] Decide upon the size of instruction and data memory in VEDA.
3. [PDS3] Design the instruction layout for R-, I- and J-type instructions and their respective encoding methodologies.
4. [PDS4] Now design and implement an instruction fetch phase where the instruction next to be executed will be stored in the instruction register.
5. [PDS5] Next design and implement a module for instruction decode to identify which data path element to execute given the opcode of the instruction.
6. [PDS6] Design and implement the Arithmetic Logic Unit (ALU) in top-down approach to develop different modules for different types of instructions. There might be

some hardware parts in the ALU that can be shared by different modules if required. This will reduce the footprint of the hardware.

7. [PDS7] Design and implement the branching operation along the ALU implementation.
8. [PDS8] Design the finite state machine for the control signals to execute the processor. Please ensure that every instruction should be executed in single clock cycle. Finally, write the test benches to simulate the CSE-BUBBLE.
9. [PDS9] Develop the MIPS code for Bubble Sort. Then convert it into machine code following the ISA given above.
10. [PDS10] Store the machine code in the instruction memory and execute CSE-BUBBLE. Store the output of the bubble sort in the data memory.

Implementation

PDS-1

Registers follow the usual MIPS standard as follows:-

$0 : always holds the value 0

$1 : $at reserved by the assembler

$2, $3 : $v0, $v1 expression evaluation and function results

$4, $7 : $a0, $a3 first 4 function parameters

$8 - $15 : $t0 - $t7 temporaries

$16 - $23 : $s0 - $s7 saved values

$24, $25 : $t8, $t9 temporaries

$26, $27 : $k0, $k1 reserved for use by operating system

$28 : $gp global pointer

$29 : $sp stack pointer

$30 : $s8 saved value

$31: $ra return address

Usage protocol:

The registers $zero, $at, $k0, and $k1 are read-only and cannot be written to.

The registers $v0 and $v1 are used to store the return values of functions.

The registers $a0-$a3 are used to pass arguments to functions.

The registers $t0-$t9 can be used by the programmer as temporary registers.

The registers $s0-$s7 are saved registers that must be preserved across function calls.

The register $gp is used to store the global pointer. The register $sp is used to store the stack pointer.

The register $fp is used to store the frame pointer. The register $ra is used to store the return address.

PDS-2

We have used 3 kinds of memories to implement the whole memory part of the CSE-BUBBLE, namely Data_memory, Instruction_memory, Veda (register module).

Data_memory is used to store the data which is to be executed. It stores 32 bit words and the height of the memory is also 32.

Instruction_memory is used to store the instructions(ISA instructions), which are to be executed by the processor. It stores 32 bit instructions, and the height of the memory is 17 as to implement the bubble sort we need 17 instructions.

Veda (Register module) is used to store registers involved during the process as defined in **PDS-1**. It is a 32*32-bit memory.

PDS-3

In this step, we will design the instruction set architecture for the CSE-BUBBLE processor. We will be using three different instruction types: R-type, I-type, and J-type.

• R-Type Instructions:

31-26	25-21	20-16	15-11	10-6	5-0
OpCode	rs	rt	rd	shamt	funct

- **Opcode** (6 bits): Specifies the type of R-type instruction.
- **rs** (5 bits): Specifies the first register operand.
- **rt** (5 bits): Specifies the second register operand.
- **rd** (5 bits): Specifies the register where the result is stored.
- **shamt** (5 bits): Specifies the amount of shift to be performed (for shift instructions).
- **funct** (6 bits): Specifies the specific function to be performed by the instruction.

• I-Type Instructions:

31-26	25-21	20-16	15-0
OpCode	rs	rd	immediate

- **Opcode** (6 bits): Specifies the type of I-type instruction.
- **rs** (5 bits): Specifies the register operand.
- **rt** (5 bits): Specifies the register where the result is stored.
- **Immediate** (16 bits): Specifies the immediate value used in the operation.

• J-Type Instructions:

OpCode	Offset
31-26	25-0

Opcode (6 bits): Specifies the type of J-type instruction.

Jump Address (26 bits): Specifies the memory address where the program should jump.

PDS-4

To fetch the instructions we use the pc register. When we finish executing an instruction, we shift the program counter to the next instruction and fetch that from the instruction memory. This fetched instruction is stored in the instruction register.

PDS-5

The instruction is then fed into the Logic_Unit, and Control modules which decode the instruction.

PDS-6

Now, the Logic_Unit and Control modules serve the purpose to decode the instructions and execute them accordingly and store the resultant back in the data memory.

PDS-7

Branching operation when decoded makes branch variable = 1, it tells the pc module that it is a branch operation and executes it accordingly.

PDS-8

Now, the CSE-BUBBLE can execute every instruction in a single clock cycle.

PDS-9

The following is the MIPS code and machine code- It is also given in the modules.

MIPS :-

```
MIPS code
0 addi $s2, $s2, 0x7
1 addi $s2, $s2, -1
2 addi $t2, $0, 0
3 addi $t1, $0, 0[[loop2]]
4 sub $s5, $s2, $t2
5 addi $s1, $s0, 0
6 lw $t5, 0($s1) [[loop1]]
7 lw $t4, 1($s1)
8 slt $t8,$t4, $t5
9 beq $t8, $zero, dont_swap
10 sw $t4, 0($s1)
11 sw $t5, 1($s1)
12 addi $s1, $s1, 1 [[dont_swap]]
13 addi $t1, $t1, 1
14 bne $t1, $s5, loop1
15 addi $t2, $t2, 1
16 bne $t2,$s2,loop2
```

Machine code :-

```
32'b00100010010100100000000000000111 ;
32'b00100010010100101111111111111111 ;
32'b00100000000010100000000000000000 ;
32'b00100000000010010000000000000000;
32'b00000010010010101010100000100010;
32'b00100010000100010000000000000000;
32'b10001110001011010000000000000000  ;
32'b10001110001011000000000000000001    ;
32'b00000001100011011100000000101010 ;
32'b00010011000000000000000000000010 ;
= 32'b10101110001011000000000000000000 ;
= 32'b10101110001011010000000000000001;
= 32'b00100010001100010000000000000001 ;
= 32'b00100001001010010000000000000001 ;
= 32'b00010101001101011111111111110111 ;
= 32'b00100001010010100000000000000001;
= 32'b00010101010100101111111111110010;
```

PDS-10

The CSE-BUBBLE accurately executes the bubble sort.

How to execute

- Store all the files in a single directory
- run the a7new_toptb.v file.
- After running, the file gives the following result

```
PS C:\Users\Hp\OneDrive - IIT Kanpur\Sem 4\CS220\lab\New folder> iverilog -o a7new_toptb a7new_toptb.v ; if ($?) { vvp a7new_toptb }

sorted_array
          1          3          4          8          9         10         15
a7new_toptb.v:43: $finish called at 567890000 (1ps)
```