Implementation Report

Assignment 7: CSE-BUBBLE (CS220)

Soham Bharambe (210264)
Pranjal Singh (210744)

List of files/modules:

- sorting.vcd - VCD file storing signal values.
- cu_tb.v - test bench for CSE-BUBBLE
- cu2.v - Control Unit - Top level module. Instantiates ALU, main memory, general purpose registers and program counter.
- alu2.v - ALU. Instantiates submodules for each operation.
- submodules.v - Contains submodules for arithmetic and logical operations.
- pc.v - Program Counter
- mux32.v - Multiplexer to select one of two 32-bit inputs.
- bubble_sort.asm - Assembly language code for bubble sort

PDS1 - Usage Protocol of Registers

- As instructed in class, the registers in use are instruction memory and data memory in VEDA, and a separate program counter (PC). There is a register storing the state of the control unit which controls the other modules.
- As we are making a FSM, a separate register will be used to store the state and control other units. This will be an 8-bit register named “state”. Its size is equal to the number of control signals.
- The PC shall store the address of the next instruction at the beginning (posedge) of a clock cycle.
- 2 VEDA modules have been instantiated for data memory and instructions. A separate external memory storing 256 words or 1024 bytes will be used.

Control Path elements:

Control signals:

- In control unit - (For details, please see the comments in the file.) mux_imm, conditional_branch, jump, we_dm, we_ext, where_to_jump, jal, load_from_ext
- General Purpose Registers: Write enable signal (we_dm)
- Main Memory: Write enable signal (we_ext)
- ALU: opcode is used to selected which operation to perform

Data Path elements:

- Control Unit: There are many multiplexers: imm_or_reg, jReg_or_jAddress, input port of “dm” instance of veda1 has a multiplexer (behavioural coding used)

- Program Counter: combinational adders adder4 and adderb for increment and branch, multiplexers c_branch and jump_mux for conditional branch and jump statements.
- ALU: all submodules performing arithmetic and logical operations - add, sub, addu, subu, or1, and1, slt, eq, sll, srl. Case statement acts as a multiplexer and selects the output of the correct operation based on opcode
- General Purpose Registers: multiplexers in the read ports
- Main Memory: multiplexer controlling read port

PDS2 - Size of memory and instruction size

- There are memory elements (VEDA) of height 32 and width 32.
- Each instruction is a 32-bit word.
- Each element in memory is also a word of size 32 bits.

PDS3 - Instruction layout for I-type, J-type and R-type instructions

- Roughly speaking, we follow the assembly format of the MIPS assembly language.
- We have divided the instructions into 5 types as below and 8 states for the FSM.

I-type instructions

- For I-type instructions, the first 6 bits are opcode, next 5 are rs (first register source operand), next 5 are rt (second register source operand) and the remaining 16 are constant or address.

6 bits	5 bits	5 bits	16 bits
opcode	rs	rt	Constant or address

- For easy identification, the first bit (MSB) of opcode for most I-type instructions is 1.
- Conditional branch instructions are immediate in the sense that relative address is provided in the instruction, but the mux controlling input to the ALU is not involved in these operations.
- Similarly, load word and store word are assigned opcodes with MSB 1. But the actual control wires are driven by the state of the machine.

Instruction	Opcode
addi	100000
addui	100001
ori	100100
andi	100101
sll	000110

srl	000111
slti	101000

Load Word and Store Word operations

- For load word and store word, the format is:

6 bits	5 bits	5 bits	16 bits
opcode	Base address (base address in the stack is stored in this register)	Source/Destination address (data will be written to/from this register)	Offset (sign extension will be done to convert value to 32 bits)

- The opcodes are:

lw	110010
sw	110011

R-type instructions

- The first 16 bits are the same as above. In the least significant 16 bits, the first 5 are rd (destination register source operand), next 5 are shamt (shift amount, whose maximum value is $2^5 = 32$) and last 6 are funct (function code).

6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
opcode	rs - first operand source register	rt - second operand source register	rd - Destination register	Shamt - number of positions to shift left or right	Funct (not used)

- We have listed shift left and shift right under R-type instructions because there is no immediate operand, and the shamt field is needed.The opcodes are:

Instruction	Opcode
add	000000
sub	000010
addu	000001
or	000100

subu	000011
and	000101
slt	001000
beq	111000
bne	111001
bgt	111010
bgte	111011
ble	111100
blte	111101

Unconditional Jump and Jump-And-Link

- The first 6 bits are the opcode. The next 26 bits are the jump address.
- The FSM and control path ensure that the value of the PC is written to the 32nd register.

6 bits	26 bits
opcode	Jump address

Jump Register

- In case of jump register (jr), the first 6 bits are opcode and the next 5 bits are the register whose address the program counter must jump to. This resembles the rs field for I-type and R-type instructions.

6 bits	5 bits	21 bits
opcode	Register containing address to jump to	Not used

PDS4 - Instruction Fetch Phase

- The program counter is updated at the end of a clock cycle, i.e. at the posedge of the clock. The read operations are combinational as they use only registers and multiplexers, so they are done along with the computation itself, when the clock is low.

PDS5 - Decoding Instructions

- In our design, the clock cycle begins and ends at the positive edge.
- At the positive clock edge, the program counter takes its next value and it is fed to the control unit through wires during the positive half of the clock cycle. The instruction wires

(ireg) take the new value during the positive half. Instruction fetch happens in the positive half of the clock cycle.
- At the negative edge, when the signal is stable, the state register takes a new value based on the instruction that has been fetched. Instruction decode happens at the negative clock edge. The negative half is used for the combinational elements such as ALU and multiplexers to compute their output. So, the correct signals are attained and are stable by the negative clock edge. As mentioned before, the program counter is updated at this point. All write instructions are executed at the positive edge, at the end of the cycle.

PDS6 - Designing ALU

- We have designed the ALU to be entirely combinational, so that all operations are done between clock edges and we need not worry about clocking it.
- We have an ALU module named “alu0” instantiated in the control unit. It instantiates submodules for elementary operations such as add, add immediate, logical operations, shifts, evaluating branch conditions, etc.
- An always block evaluates the opcode and selects the output of the required submodule. Its output is fed to the output port of the ALU.
- For unsigned operations, the first 16 bits of the second operand fed are automatically padded with zeroes. This is correct only for unsigned operands. For signed operations, we pad the first 16 bits with the first bit of the lower half of the input (in2[15]) using concatenation and replication operators. With this, 16-bit negative numbers are correctly converted to 32-bit negative values.
- This is important because we will be using signed operations in sw and lw operations, to add the (possibly and commonly) negative offset to the base address.

PDS7 - Branching Operation

- Branching operations use the ALU to evaluate the logical condition. The LSB of the ALU output is the boolean output of the specified logical operation.
- The “conditional_branch” port of the program counter uses these values. It is AND-ed with the “jump” wire of the control unit, as the LSB of ALU output may be 1 for operations other than branching operations also.
- There is a multiplex in the PC which executed branching operations when its selector is asserted.

PDS8 - Designing FSM

- The task of the FSM is to control wires like write enable for general purpose registers and main memory, branch, jump, immediate instructions, etc. So, we define each state as a 8-bit parameter, where each of the 8 bits correspond to one of the control ports.
- The states are:

RTYPE	Register-type instructions other than conditional branches
ITYPE	Immediate instructions other than lw and sw
C_BRANCH	Conditional Branch instructions
JUMP	Unconditional jump instruction
JR	Jump register instruction
JAL	Jump-and-link instruction
LW	Load word instruction
SW	Store word instruction

- At rising clock edges, the PC's value is updated and write operations are executed. At falling edges, the state is entered, as the instruction signal is stable by then. The time between falling and rising edge is used to perform computation.

PDS9 - Writing MIPS code for Bubble Sort

- MIPS code was written for bubble sort and tested on QTSPIM.
- For simplicity, the number of iterations of the nested loop was kept constant at n-1, where n is the number of input elements.
- The machine level code was written using parameters for each opcode to increase readability. Concatenation was extensively used.

PDS10 - Executing the program

- Machine level code can be copied directly into the instruction memory.
- Our program requires register 29 of the data memory to point to a location in the main memory that stores the input size. The registers with addresses above this register contain the input. The output is stored in the same location in the main memory.