CE 6302.001 – Microprocessor and Embedded Systems – F22

Dhavalashri Prasad Net-ID : dxp210085	Sanmati Marabad Net-ID : sxm210368

Final Lab Report

Project #1 dealt with designing and implementing a Microcontroller Unit including several necessary functional units such as Program Counter, Instruction Memory, Instruction Decoder, Register File, Arithmetic Logic Unit (ALU), Data Memory and IO Module. The 8-bit input output MCU is designed using Verilog. All the units are connected using a 4 – stage pipelining. Also, an 8-bit program counter controls the operation of instructions.

Instruction Memory

Instruction memory is a ROM of size 17 x 256 which can hold 256 instructions of 17-bit length. It'll take PC as input and output is the corresponding instruction in the location of PC value.

Data Memory

Data Memory is a RAM which has 8 x 256 size where we can store data using 8-bit address and 8-bit values as input. Also, this has a control signal called write enable which controls read and write of data memory.

Register

Register is a 8 x 8 bit module which has eight 8-bit registers with input and output functionality. It also has a write enable control signal which controls read and write. A and B data outputs are controlled by A and B addresses. The D address will have the address of the register where 8-bit data input will be written.

Arithmetic and Logic Unit (ALU)

ALU is designed to perform basic arithmetic and logical operations with the inputs function select, shift, A and B. The outputs of the ALU are 8-bit data and four flags (Zero, Negative, Carry and Overflow)

Instruction Decoder

Instruction decoder is the main component in the MCU which takes 17-bit instruction as input and determines the control signals for MCU operations. The control signals from the decoder are:

Register read/write: controls the read/write of the register file
Data address: is used to select the register where the value needs to be stored.
MuxDsel: control signal used to control the data selection that will be written to the register.
Branch select: control signal used to control MuxC to determine the program counter when there is a branch instruction.
Memory write: this control bit determines if data will be written to data memory.
Function select: determines the function performed by ALU.
Mux A select: selects the input of MuxA for which the data will be passed to BusA.
Mux B select: selects the input of MuxB for which the data will be passed to BusB.
Register A address: selects the register from the register file which acts as RA.
Register B address: selects the register from the register file which acts as RB.
Constant select: determines the sign extension of immediate value.

I/O Module

The I/O module acts as an interface for MCU to connect two devices such as keyboard, mouse, LEDs etc.

Pipelining in MCU

A four stage pipeline architecture is implemented in the MCU designed.

1. Fetch
2. Decode
3. Execute
4. Writeback

Fetch

The pipeline's fetch stage is when the subsequent instruction to be executed is obtained. The next instruction is retrieved from program memory using the program counter. Once the instruction is delivered to the clock, the program counter is increased by 1, and the decoding phase. If a branch or jump occurs, the program counter will be changed to a unique value determined by control logic.

Decode stage

The fetch stage's instruction is passed on to the decode stage, where the instruction decoder is used to decode it. In order to prepare the data required in the following step, the instruction decoder will transmit the appropriate control signals to the register file and any other control logic. The execute and writeback stages may make use of the control signals. In order regulate branching, these signals may also be sent back to the fetch step. The inputs to the ALU (values stored in registers and maybe an immediate value from the instruction itself) will also be carried on to the next step of the pipeline, the execute stage, along with the control signals necessary in the other stages of the pipeline.

Execution stage

Computation and memory reads and writes happen during the execute phase of the pipeline. The address to branch to is also determined at this step. Even if a branch is not taken, the branch address must be calculated by a separate adder because the ALU is needed to determine branching circumstances such a value being zero. The relevant control signals and data are transferred to the writeback stage once a calculation is finished or a value is loaded or saved from memory.

Writeback stage

The pipeline's writeback stage is the last stop. The writeback stage's sole purpose is to save data in the register file after calculation. It should be noted that because each stage occurs concurrently, it is possible that the information that needs to be saved in a register is required at a previous level in the pipeline. Since the necessary data is not yet saved in the register file, this is referred to as a data hazard. Data risks will be covered in more detail later.

Data Hazards

A pipeline architecture has the drawback that writing data back into the register file requires a number of clock cycles. It follows that another instruction may be running that needs this result before one instruction saves it in the register file. A data hazard is what is experienced when this happens. We will put up a stall to stop this. A data hazard stall prevents incorrect data from being stored back into the register file or data memory and prevents the program counter and instruction register from updating. The program proceeds after the accurate data has been written back to the register file. In actuality, this indicates that an additional clock cycle is needed to execute the action using the proper data.

Branch Hazards

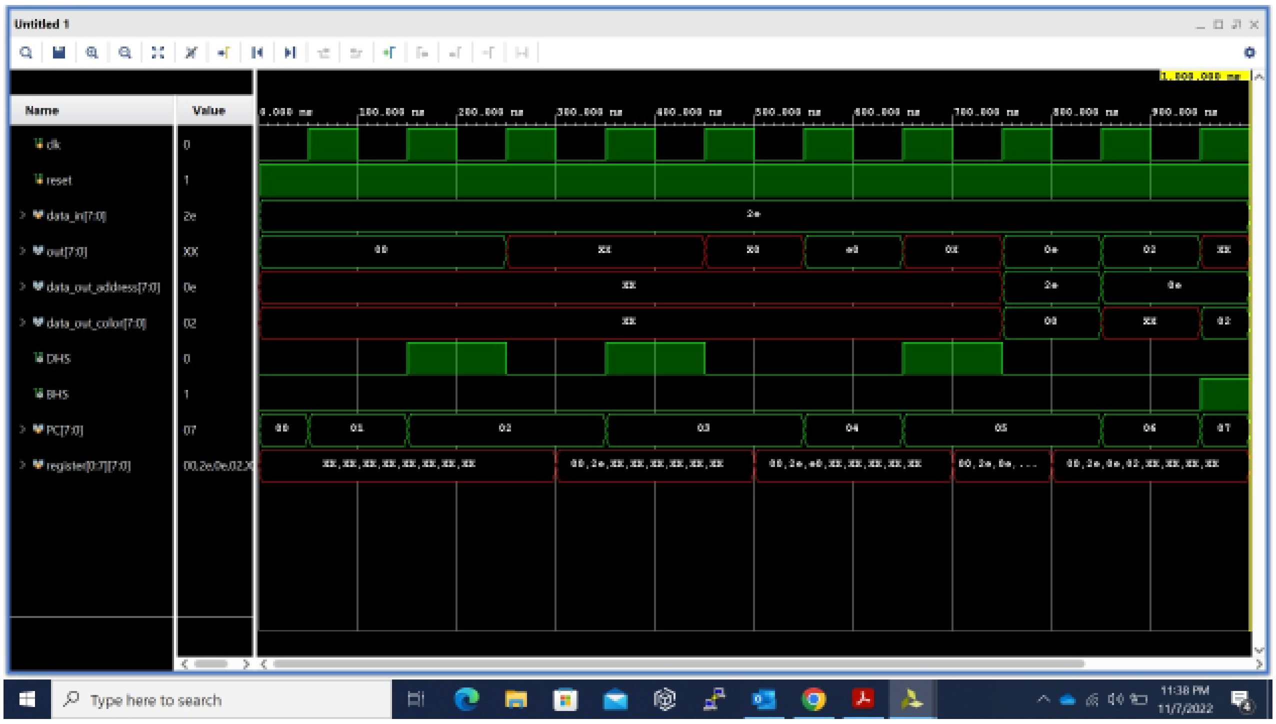
Instructions being carried out after a branch is taken with your MCU is another issue that can arise. Similar logic is required to prevent the instruction that follows the branch from happening since whether or not a branch is taken is decided after the next instruction is in the pipeline. To do this, turn off writing to the register file and program memory, set the next instruction to NOP.

Testing of the working MCU

The program given in the manual is tested.

```
0    INPUT  R1          | // Get input and store in R1
1      LSL  R2, R1, 4   | // Shift R1 left 4 bits and store into R2
2      LSR  R2, R2, 4   | // Shift R2 right 4 bits
3      LSR  R3, R1, 4   | // Shift R1 right 4 bits and store in R3
4   OUTPUT  R2, R3      | // Output R3 to address R2
5       JR  R0          | // Jump to start
```

Output



Explanation

- In the first clock cycle, the first instruction is moved to the IR register.
- In the next cycle, the input instruction is decoded and a decoder sends control signals IE and DA which decide that it is an input instruction which needs to be written into register1. Data is given through Data_in to the mux for which IE acts as a select value. Also the second instruction is fetched into the IR.
- In the third cycle, the IE value is passed to the writeback stage and the second instruction is decoded and the control signals detect that there is a data hazard between the first and the second instruction and the counter will be stalled.
- In the fourth cycle, writeback of the input instruction is completed and the register is ready to be read by the next instruction. The stalled counter is now resumed and the second instruction will start executing. Also, the third instruction will be fetched to the IR.
- In the fifth cycle, detecting the data hazard between second and third instruction, the PC will be stalled again and the execution of second instruction happens.

- In the sixth cycle, the writeback of second instruction output is completed and is ready to be read by the third instruction. Since there is no dependency between the second and the third instruction, the program counter increments and fetches the next output instruction.
- In the seventh cycle, the execution of right shift instruction happens and the output instruction is decoded and data hazard is detected. And hence, PC is stalled.
- In the eighth cycle, execution of fourth instruction happens and output instruction now starts executing. Also, the next Jump instruction is fetched.
- In the ninth cycle, execution of output instruction completes and data_out_color is written with 02 value and data_out_address is written with 0e value. Also, since there us jump instruction, branch hazard is detected and PC is assigned the value of register value mentioned in the instruction, ie R0 which is 0.