Pipelined Microprocessor

VLSI DESIGN LAB EE705

Sadaf Parwaiz (153079016)
Meet Haria (153079029)
Kaushal Bhuva (163074003)
Kiran Kumar Grandhi (163074007)

- **Problem Statement:**

  To design a six stage pipelined microprocessor.

- **Introduction:**

The instruction pipelining is a technique that is used to execute multiple instructions parallel with a delay of one clock cycle.

The advantage of this technique is that it allows a faster throughput. In pipelining, the instruction execution is usually divided into stages. The number of stages vary depending onimplementation.

In our case, we have six stage pipelined structure. The instruction is split into six different steps which can be executed in parallel and the instructions can be processed concurrently, i.e. starting one instruction before finishing the previous one.

Pipelining increases the instruction throughput, but does not reduce latency, i.e. the time needed to complete a single instruction.

Rather, pipelining may increase the latency due to additional overhead by breaking the computation into separate steps, and depending on how often the pipeline stalls, or needs to be flushed.

There are 3 types of hazards that occur while implementing pipelining:

- Structural hazard

When a machine is pipelined, the overlapped execution of instructions require the pipelining of functional units and duplication of resources to allow all possible set of instructions in the pipeline. If some combination of instructions cannot be accommodated because of a resource conflict, the machine is said to have a structural hazard.

- Data hazard

Data hazard occurs when a current instruction needs the resource and previous instruction is still under execution. Data stalling and forwarding technique is used to prevent data hazard.

- Control hazard

If jump or branch instruction occurs, the already fetched instructions after jump or branch needs to be flushed. This decreases the throughput which is a control hazard.

The six stage pipeline that is implemented, eliminates the two hazards, structural and data hazard. The structural hazard is eliminated by using two separate memories: instruction memory and data memory.

The problem of Data hazard is solved by introducing stalling and forwarding logic.

- **Implementation of Pipelined Microprocessor Architecture**

16 bit processor RISC architecture is implemented. Six stages of pipelined architecture are as follows:

- Fetch stage
- Decode stage
- Register readstage
- Execute stage
- Data Memory Accessstage
- Write back stage

Eight registers, R0 to R7, each of 16 bits are used. The logic operation in each stage is purely combinational. The data flow is sequential in nature.

- **Instruction format**

There are three types of instruction formats that we have decided to implement:

1. R-type instruction format (Register type)
2. I-type instruction format (Immediate type)
3. J-type instruction format (Jump type)

- **Instruction set**

Instruction	Opcode	Register or Immediate values				
ADD	0000	RA	RB	RC	0	00
ADC	0000	RA	RB	RC	0	10
ADZ	0000	RA	RB	RC	0	01
ADI	0001	RA	RB	6-Bit Immediate		
NDU	0010	RA	RB	RC	0	00
NDC	0010	RA	RB	RC	0	10
NDZ	0010	RA	RB	RC	0	01
SHL	1010	RA	RB	RC	0	00
SHR	1010	RA	RB	RC	0	01
SAR	1010	RA	RB	RC	0	10
RTR	1010	RA	RB	RC	0	11
LHI	0011	RA	9 bit Immediate			
LW	0110	RA	RB	6 Immediate		
SW	0101	RA	RB	6 Immediate		
BEQ	1100	RA	RB	6 Immediate		
JAL	1000	RA	9 Immediate			
JLR	1001	RA	RB	6 Immediate		
LM	0110	RA	9 Immediate			
SM	0111	RA	9 Immediate			

- **Data path for pipelined microprocessor**

Each of the processor components have been implemented as a separate block. These processor blocks were integrated into different stages.

A. **Fetch stage**
B. **Decode Stage**
C. **Register Read stage**
D. **Execution stage**
E. **Data memory access page**
F. **Register write back stage**

Integration of Processor Components

Each of the processor components have been implemented as a separate block.These processor blocks were integrated into different stages:

1. **Instruction Fetch Stage**

- It consist of PC register, incrementer and instruction memory.
- With every clock cycle, PC gets incremented by 1. The instruction is fetched from instruction memory corresponding to the address of the PC.
- PC, PC+1, and instruction is passed to decode stage via registers.

2. **Decode Stage**

- The decoder component have been instantiated over here.
- The control signal are generated by decoder block are passed to register read stage along with PC and PC+1 via register.
- The control signals are generated for register read stage, execute stage, memory stage and write back stage.

3. **Register read stage**

- Register file is instantiated in register read stage.
- Based on opcode and remaining instruction bits, the controller generates the A1 and A2 (address lines for register). The data D1 and D2 are passed to execute stage via registers.

4. **Execute stage**

- In execute stage, actual ALU task is performed.

-ADD

-NAND

-Address computation for Load/Store instruction

-Shift/Rotate

-Carry zero flags are updated after ALU operation is performed.

The control signals decides which ALU task needs to be performed. The data/address/PC/PC+1/CZ flag/control signals are all passed to memory access stage via register.

5. **Data memory access stage:**

- The data memory is instantiated over here.
- For Load instruction, the address computed from execute stage is given to address of data memory. The data_out from data memory corresponding to address is passed to write back stage. During this process, memory read signal is high, memory write signal is low.
- For Store instruction, the address computed from execute stage is given to address of data memory. The data from previous stage D1/D2 is given to data_inof data memory. During this process memory write signal is high and memory read signal is low.
- For other instruction, data memory access will not come into play although the instruction will consume clock cycle for data memory access stage.

6. **Write back stage**

- The data from data_memory/ ALU output is written into register file corresponding to write address A3.
- The control signal will determine which of these will be selected as to be written in register file.

- **Approach of implementation**
    1. **Components of stages**

In ALU component, we implemented PC incrementer, sign extender, 16 bit adder, NAND, n bit register (different pipelined stages).

2. **Stages interconnection**

The common file Datapath contains integration of all stages. The output of one stage in form of signals are interconnected and passed on to next stage.

3. **Data forwarding logic implementation**

When the destination address matches with the source address, the Data forwarding logic generates a signal such that data is forwarded from memory access stage/ writeback stage to execute stage for the next dependent instruction.

4. **Instruction memory and Data memory**

Address range of instruction and data memory is from 00 to ff.

5. **Testbench and package component**

The test bench generates clock, 50ns reset pulse to reset all initial values of all registers to zero.

- **Modelsim simulation and testing of instructions.**

The instruction.txt file contains the set of instructions that needs to be executed. The instructions is encoded into hexadecimal format imstruction.mem file using a python script. Its serves as instruction memory. Similarly data memory is loaded with constant values.

The simulation results shows the execution of instruction in every clock cycle thus achieving pipelining of instructions.

- **Modelsim simulation and testing of instructions.**

The analysis and synthesis of the design was performed on Quartus. Once the analysis, synthesis, fitter(place and route), Assembler (Generate Programming files), TimeQuest Timing Analyser and EDA Netlist Writer steps are completed, then the clock and reset pins are assigned to the design using pin planner. The SignalTap II Embedded Logic Analyzer is a system-level debugging tool that captures and displays signals in circuits designed for implementation in Altera's FPGAs.So by using Signal Tap II Logic Analyser from the tools menu, we have programmed our design on the FPGA chip. Once the design is programmed, we can view the outputs as waveforms in different registers and memory elements.

- ## Convolution Block Implementation

A separate convolution block is implemented for DSP application. The pipelined processor has 8 register each of 16 bits. So convolution block for 4 input points is implemented.

(R0 , R1, R2, R4) * (R5, R6, R7, R8)

The output of convolution is of length 7. So atleast 7 write cycles is required to write back in registers thereby requiring 7 cycles stall.
We were unable to integrate the convolution block in overall datapath because of large number of stalling is required.

The logic for convolution is implemented as follows

S0 <= R4*R0
S1 <= R5*R0 + R4*R1
S2 <= R6*R0 + R5*R1 + R4*R2
S3 <= R7*R0 + R6*R1 + R5*R2 + R4*R3
S4 <= R7*R1 + R6*R2 + R5*R3
S5 <= R7*R2 + R6*R3
S6 <= R7*R3

- ## Conclusion:

Hence the pipelined microprocessor and a separate convolution block is implemented is implemented and tested.