Effects of the pipeline depth in a microprocessor

Eyanshu Gupta
2019H1030018P

Vibhor
2019H1030517P

***Abstract*—There is a fundamental decision to be made in the design of a microprocessor is to choose a wide range of pipeline structures and depth from among that exist. This paper discusses the question of the optimum pipeline depth for a microprocessor that gives an optimum performance. As we have seen in the class, The pipeline depth affects the performance of the microprocessor by various factors. we have treated the problem in the form of an analytical study as well by a simulation. The simulation closely follows a MIPS architecture as studied in our course. the simulation can be easily used to compare the programs involving different instructions and their effect on the single-cycle or multicycle processor.**

Project—https://github.com/eyanshu1997/Simple-MIPS-Simulator

I. Introduction

This paper discusses the comparison of the depth of the pipelines in the microprocessor by means of an analytical study that closely follows the same done by [1]. They worked on a proprietary simulator to prove their analytical study and developed their own pipeline structure to monitor various pipeline depths. The simulation methodology employed by them is not easily reproducible and displayable in a class setting and presented easily and proprietary simulators and not easily usable and accessible. During the course of this study we have developed a simulator that closely follows each of the MIPS microprocessor detail with all hazards as well as each pipeline stage. our simulators working is very close to the real processor.

II. Analytical Study

Derived an expression for the time taken for instruction processing taking into account various factors such as latch delay, number of stages in the pipeline, superscalar processor, branch prediction accuracy, etc.

$$T/N_I = \left(\frac{t_o}{\alpha} + \frac{\gamma N_H}{N_I} t_p\right) + \frac{t_p}{\alpha p} + \frac{\gamma N_H t_o}{N_I} p.$$

- T_P=is the total logic delay of the processor
- N_i=no of instruction
- N_h=no of hazards
- T_o=latch delay
- p=depth of pipeline
- γ= is the fraction of total pipeline delay averaged over all hazards and contains all of the information about the hazards in a particular piece of code as well as details of the microarchitecture of the processor.
- α= is a measure of the average degree of superscalar processing

The first term in this expression is independent of the pipeline depth; the second term varies inversely with p the last term varies linearly with p. This result depends on numerous parameters. The ratio N_h/N_i is mainly dependent on the workload being executed, but is also dependent on the microarchitecture through, for instance, the branch prediction accuracy. γ and α are mainly microarchitecture dependent, but also depend on the workload. T_o is technology-dependent, while t_p dependent on both technology and the microarchitecture.

III. Developing the Simulator

There are several design questions we need to take care of while developing the simulator so as to closely simulate the actual hardware performance, we have closely followed various research that has laid out the strategy for developing the simulators.

The simulator closely follows the MIPS architecture. MIPS (Microprocessor without Interlocked Pipelined Stages) is a reduced instruction set computer instruction set architecture developed by MIPS Computer Systems. It supports a variety of instructions. A complete list of instructions supported by our simulator are: add, sub, mul, beq, lw, sw, addi. The datapath of the multicycle is as displayed in fig 2.The pipeline structure in the multicycle is as follows:

	Clock number								
Instruction number	1	2	3	4	5	6	7	8	9
Instruction i	IF	ID	EX	MEM	WB				
Instruction i+1		IF	ID	EX	MEM	WB			
Instruction i+2			IF	ID	EX	MEM	WB		
Instruction i+3				IF	ID	EX	MEM	WB	
Instruction i+4					IF	ID	EX	MEM	WB

Fig 1: Pipeline Structure of MIPS

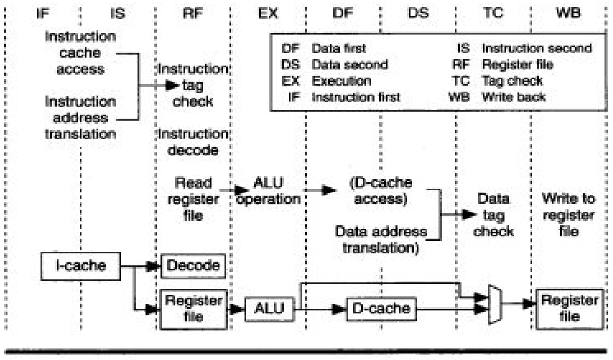

Fig 2: R4000 pipeline activities.

We have also implemented the R4000 MIPS simulator that contains 8 pipeline depth as defined as per standards. The pipeline structure is defined as follows:

• IF - Instruction Fetch, First Half- PC (Program Counter) selection actually happens here, together with initiation of instruction cache access.

• IS - Instruction Fetch, Second Half-complete instruction cache access. Note that in this project we assume instruction accesses always hit the instruction cache.

• RF - Register Fetch- Fetch the required register values.

• EX - Execution- includes effective address calculation, ALU operation, and branch-target computation and condition evaluation.

• DF - Data Fetch, First Half of data cache access.

• DS - Data Fetch, Second Half, completion of data cache access.

• TC - Tag Check, check of cache load miss may occur. other memory stage-related functions also performed.

• WB - Write Back- for loads and register-register operations.

The simulator can simulate a single cycle as well as a five-stage and eight-stage multi-cycle microprocessor.

IV. IMPLEMENTATION

The simulator consists of four files each function are defined as follows:

1. Helperfunc.cc- This class contains all the helper functions for other class:
    - trimInstructions- It is used to convert each line of the program to standard form to be understood by the simulator.
    - extractOpcode- It extracts the opcode from the instruction.
    - regvalue- It returns a register value according to standard converts a1,t1, etc to a numeric value between 1-32.
    - extractregister- It extracts a register value.
    - extractBase -To extract the base register for ex-lw $s0, 8($s1) here s1 is base register
    - extractImmediate- To extract the immediate value in the I type instructions

2. otherclasses.cc

    Instructions class

    - contains various parameters for handling of various instructions
    - contains three constructors
    - utilizes the methods from helperfunc class to manipulate the instruction to the desired format and extract various elements from an instruction like src register, des reg, opcode, etc.

    latch

    - depicts the latch between the various stages of processing

    prog

    - an instance of a program. it is a list of the instructions and a program counter.
3. simulatorclass.cc
    - contains a single cycle simulator class and pipelined simulator class.
    - class singlecycle_simulator
        - array type data structure for registers and data locations
        - execute a method for simulating all stages of execution in one go, contains conditional statements for simulation of R type or I type or Branch type instruction. The aftermath following simulation is reflected in the data structure described above
        - simulate method for simulating the single cycle only it loops over all instructions and call execute method described above as well as calculates the time required for completion
    - class pipelined_simulator
        - latch type data structure contains four instances of latch type DS depicting the latches between the various stages
        - array type data structure for registers and data locations
        - rawHazard method for determining if hazards exist or not
        - fetch method for depicting the fetch stage incrementing PC as well as turning on the l1 latch
        - decode method for depicting the decode stage as well as checks for hazards and branch instructions
        - execute method for depicting the execute stage, different execution sequences according to instruction type
        - mem method for checking into data location as well as checks for the validity of its location and memory alignment
        - wb method to finally write the data into the data location
        - lastly, simulate which combines all the above methods and run each stage individually for instruction and counts the cycles for each instruction completion

and prints the utilization of the each stage

- class r4000_pipelined_simulator
    - the class structure is the same as the previous class, it has similar methods for each stage defined in the previous section
    - This contains eight stages IF,IS,RF,EX,DF,DS,TC,WB
    - It also contains a simulator method that executes these methods.

4. simulator.cc
    - a menu-driven program that lets the user choose which simulator to run a single cycle or pipelined one and provide a path to the instruction i.e. ASY file.

V. Conclusion

We ran different programs which consisted of one type two type and multiple types of instruction, some contained only R type, some R and I type, and some which contained all kinds of instruction, we ran them on a single cycle and pipelined and found their total cycles, completion times and individual stage utilization. In relation to pipeline depth we can say that deeper pipelines have better performance. Basically, what it says is that the total logic delay of the processor is divided up into separate parts for each pipeline stage. The performance is given by the rate at which instructions finish the pipeline, not the time it takes for instructions in the pipeline, but Deeper pipelines suffer a larger penalty from hazards. Deeper pipelines also suffer from a larger penalty from hazards; also each additional hazard induced pipeline stall causes a performance degradation from that latch overhead. If no hazards were present, the optimum performance would occur for an infinitely deep pipeline. As the number of hazards per instruction increases, the optimum pipeline depth is pushed to lower values. As the fraction of the pipeline stalled by any hazard increases, the optimum pipeline depth decreases. As the latch overhead increases relative to the total logic delay, pipelining becomes more of a burden, and the optimum pipeline depth decreases.

The performance of the processor is dependent on three factors: instruction count, CPI, and the cycle time of the processor. the performance of the microprocessor is directly dependent upon these three parameters. As the pipeline depth increases, the instruction count remains the same, the CPI increases with an increase in pipeline depth but the clock cycle time of the processor decreases. there exists an optimum depth with the product of these is maximized.

References

[1] A. Hartstein and T. R. Puzak, "The optimum pipeline depth for a microprocessor," Proceedings 29th Annual International Symposium on Computer Architecture, Anchorage, AK, USA, 2002,
pp. 7-13.

[2] Doug Burger and Todd M. Austin, “The simple scanlar toolset, Version 2”.

[3] Ben Juurlink, Koen Bertels, Bei Li, “A Flexible Simulator of Pipelined Processors”

[4]http://www.ece.lsu.edu/lpeng/ee7700-2/project2/MIPS-R04K-uman3.pdf

[5] https://people.eecs.berkeley.edu/~kubitron/courses/cs252-S07/handouts/papers/R4000.pdf

[6] João Carlos de Oliveira Quintas,”MIPSter32 - A 32 bit MIPS Simulator “ June 2016

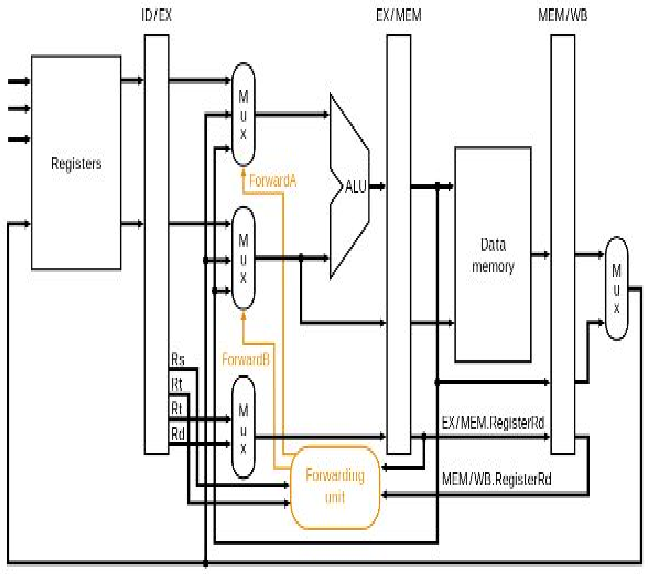

Fig 3: DataPath