Philippine Engineering Journal (2000) XXI (1): 59-64

32-BIT PIPELINED RISC MICROPROCESSOR

Leonard Lee* Ma. Celeste Sangil, and Louis Alarcon
Intel Microprocessors Laboratory
Electrical and Electronics Engineering Department
University of the Philippines, Diliman

ABSTRACT

The design of microprocessors is a crucial step in determining the over-all performance of a processing system. A design technique may lead to several improvements in one aspect of processor performance but it may also worsen other aspects.

In terms of the design of the instruction set, there are mainly two approaches of processor design. The first approach, called Complex Instruction Set Computers (CISC), uses a few commands to implement complex instructions. Ideally, this leads to shorter programs. The other approach, called Reduced Instruction Set Computer (RISC), uses simple instructions to perform complex tasks.

A key advantage of RISC over CISC is its suitability for pipelining. Since the RISC approach implements simple instructions only, processing of instructions can be done in a parallel manner. This implies that more operations can be executed at a single instruction cycle. Hence, the technique of pipelining leads to higher throughput.

The project aims to convert an existing implementation of a 16-bit non-pipelined RISC microprocessor into a 32-bit 5-stage pipelined RISC microprocessor. The improvement of the previous project in terms of execution speed would be given focus so the layout size would have to be sacrificed. The speed of the pipelined processor in executing a simple test bubble-sorting algorithm will be compared to that of the former project and the Motorola 68000. Simulations and the design itself would be implemented using CADENCE and Visual HDL 5.2.

I. Introduction

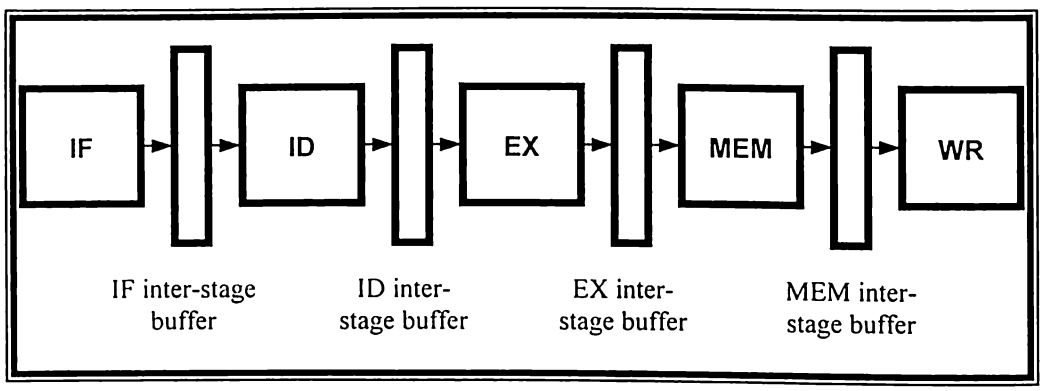
The improvement of an existing implementation has always been a part of the development of technology. Specifically, for a microprocessor, this improvement is achieved through the design of its architecture. For a RISC microprocessor, a suitable improvement can be the integration of pipelining in its design. [1]

Pipelining is the overlapping of the execution of instructions [2]. Its realization had its advent in the late 1950's [2]. The goal is to improve the throughput by making the instructions execute in a parallel manner. The ideal "speed-up" of the processor would be the number of pipeline stages.

A 16-bit non-pipelined RISC microprocessor has been previously designed. The authors of this project are Gemini Abad and Leonard Jarillas, both of whom were affiliated with Intel Microprocessors Laboratory. The system, could be interfaced with a memory unit having 2^{16} = 65,536 memory locations each 16 bits wide and, in addition, has 16 general-purpose registers. Small layout size and functionality were given priority, thus speed had suffered yielding an overall system clock of 10 MHz. However, since the project was

intended for future microprocessor design purposes, the authors had allotted room for pipelining [3].

It was through this room for improvement that this project came to realization. The project aims to incorporate a 32-bit five-stage pipeline in the existing implementation of a 16-bit non-pipelined RISC microprocessor. The improvement hopes to serve a lot of purposes. In addition to the benefits as a microprocessor, it can also be used as a reference for other [2].



II. Design Method

The design procedure is divided into three major phases, namely: 1.) Control Circuit Design, 2.) Datapath Design, and 3.) Testing, Simulation, and Comparison with Motorola 68000.

The first phase is carried out in five chronological steps. These steps are designing the: a.) Fetch Control; b.) Decode Control; c.) Execute Control; d.) Memory Access Control; and e.) Write to Register File Control.

The second phase involves the design of the datapath components of each of these stages.

The third stage is a thorough simulation and testing activity, after which the project Microprocessor's performance is compared to Motorola's 68000 using a bubble sort algorithm.

For every step that the designers accomplish, they simulate their design with the Leapfrog Simulator tool of Cadence.

III. Control Circuit Design

Fig. 1 The Pipeline Layout

There are five stages the control block will go through in executing the instructions: instruction fetch (IF), instruction decode (ID), execution (EX), memory access (MEM), and write to register (WR). In addition to the stages, two "speed-up" schemes were implemented in this processor: the two-bit branch prediction scheme and operand forwarding [2]. The inter-stage buffers hold the data processed by a former stage and its corresponding instruction [1]. Each of the five stages completes their execution after three clock cycles.

A. *Instruction Fetch (IF)*

This stage starts with a read request from the memory to "fetch" the instruction and stores it in the instruction register. After "fetching", the program counter (PC) is refreshed according to the type of instruction and according to the branch prediction. Two registers implemented in a queue are used to backup the PC in case of a branch prediction failure. This stage ends with the transfer of the contents of the instruction register to the IF inter-stage buffer which holds the current instruction to be used for the next stage (ID).

For a JUMP instruction, the new PC value can be immediately loaded in to the PC register (no stalls) by sign extending the jump address contained in the instruction register. If the fetched instruction is a JAL (jump and link) instruction, the system will stall while waiting for the PC + 1 value to be written to register file. The write will only occur in the WR stage so the stall will last up to that stage. In case of a JR (jump register) instruction, an RTS (return from subroutine) instruction, or a RTI (return from interrupt) instruction since the new PC value will only be available in the ID stage after fetching it from the register file, a stall will occur for one stage. The only difference between the JR and the RTS or the RTI instructions is the address space in the register file from where the PC value is obtained. The address space is specified in the instruction for a JR instruction while the address space is fixed for a RTS instruction (11100000_{hex}) and for a RTI instruction (10000000_{hex}). During the stalls, the IF control will not fetch a new instruction but it will pass a NOP (no operation) instruction to the IF inter-stage buffer.

B. *Instruction Decode (ID)*

While a new instruction is being fetched in the IF stage, the old instruction register value stored in the IF inter-stage buffer is passed on to the register file to fetch the operands, in preparation for the instruction's execution. The fetched operands are temporarily stored in two registers before they are written to the ID inter-stage buffer. Right before the ID inter-stage buffer are two data selectors, one for the destination data and the other for the source data, which choose the data to be passed on to the ID inter-stage buffer depending on the data dependencies. The selectors effectively implement the operand forwarding.

C. *Execute (EX)*

There are five types of instructions as described in the instruction set design part. These are the following: ALU instructions, Load/Store instructions, Branch Instructions, Jump Instructions, and Condition Code instructions.

1) ALU control: The ALU control is responsible for giving off the control signals for ALU operations. A selector, which chooses which ALU operation to perform, and a latch signal to update the condition code register characterize the control stage. For a NOP instruction however, the ALU does nothing and the instruction stored in the ID inter-stage buffer is passed on to the EX inter-stage buffer.

2) Load/Store control: A Load/Store instruction is not executed in the EX stage but in the MEM stage. However, the EX stage checks for a Load/Store instruction to prepare the system for a stall due to a structural hazard. A structural hazard occurs when one or more operations need to access the same resource. In this case, the two units are the fetch operation and the Load/Store operation. The operands and the instruction code stored in the ID inter-stage buffer are then passed on to the EX inter-stage buffer.

3) Branch control: The branch target address was already loaded during the IF stage according to the branch prediction. However, if, after checking the CCR, the branch prediction is incorrect, the system is prepared for a stall to load the correct PC value, which was previously stored in the queue, into the PC. During the stall the IF control will not fetch a new instruction but it will only pass a NOP instruction to the IF inter-stage buffer while the correct PC value is being loaded into the PC.

4) Jump control: Execution for most of jump instructions have already begun in the IF stage. If the instruction contained in the ID inter-stage buffer is a NOP instruction (following the RTI - return form interrupt instruction because of the stall), the CCR (condition code register) value fetched from the register file (address is 11000000_{hex}) is loaded into the CCR, thus restoring the state from where the interrupt occurred.

5) Condition Code control: The SM (set mask) instruction sets or resets the interrupt mask of the CCR (7 bits) according to instruction contained in the ID inter-stage buffer. The TCB (test condition-code bit) instruction asserts the test bit of the CCR according to the condition code bit that is to be tested.

D. Memory Access (MEM)

Two of the Load/Store instructions are processed here namely the LW (Load Word) instruction and the SW (store word) instruction. The address and the data are obtained from the EX inter-stage buffer. While processing any of the two instructions, the IF stage stalls, meaning no new instruction is fetched.

E. Write to Register (WR)

Only the remaining two Load/Store instructions (MOVE and LDI – load immediate) and ALU instructions are processed in this stage. For ALU instructions, the destination data contained in the MEM inter-stage buffer is written into the register

F. Interrupt Control

The interrupt control handles interrupts whenever they occur. An interrupt can only be recognized if the system is not in a stall state and if the interrupt mask bit of the condition code register is asserted low. The thirty-second bit of the interrupt vector is checked in the IF stage after a new instruction has been fetched. The control block lets the last instruction finish before jumping to the interrupt address. While waiting for the last instruction to finish its execution, the program counter is backed up into a register (backup1_pc register), before it is loaded with the interrupt address from the interrupt vector (sign-extended 31^{st} bit down to the least significant bit). After the last instruction has finished its execution, the "backed up" program counter and the condition code registers are written into the register file to save the status of the processor.

It is left for the user to copy the contents of the register file containing the PC value and the CCR before the interrupt occurred. Copying the PC value and the CCR value makes it possible for another interrupt to occur during the interrupt service routine. This technique can also be used for the JAL (jump and link) instruction.

IV. Datapath Components

The datapath components generally consist of the control block, the register file, the Arithmetic and Logic Unit (ALU), the memory, the registers, and the selectors. The control block is four-state state machine where the initial state is the reset state. The register file is an eight-bit addressable storage (256 words) for the registers, which has a latch for write requests. The ALU is the "computing unit" for the ALU instructions. The memory implemented is just for the purposes of simulation. Two latches, a read and a write latch were used. The registers are latched in either the negative-edge trigger or the positive-edge trigger of the clock. The selectors used are just bundles of tri-state buffers and sign-extenders.

V. Test Results

A sensible program, the bubble sort algorithm, was used for testing the microprocessor. It was used in this testing for two reasons: 1.) to test if the microprocessor is capable of handling a working program – that is the bubble sort program; and 2.) to compare the project Microprocessor's performance and code density with the already-noted performance of Motorola's 68000 in bubble sorting.

From the project scope, it is apparent that the designers had not expected the project Microprocessor to yield a higher code density but a faster execution time than Motorola's 68000. The goal of this testing is to determine how much better the 68000 performs than the project Microprocessor and to find out how the latter can be improved.

Table 1 states the difference between the two microprocessors in terms of two factors: 1.) the code size; and 2.) the number of execution cycles based on the bubble sort program that was run through these two different systems:

Table I
Summary of the comparison of the three microprocessors

Processor	Code Size	# of Execution Cycles
Motorola's 68000	48 bytes	960
16-Bit Nonpipelined RISC	50 bytes	1101
32-Bit Pipelined RISC	100 bytes	880.50

VI. Conclusion

Growth in microprocessor design has accelerated in the past few decades. This rapid development is experienced mainly because of the expertise gained from the academe.

With the continuous influx of information, simultaneously developing with other fields, lofty mentors attempt to improve the quality of education offered in their institutions, to cope with the ever-changing demand for development of the younger generation.

Part of this is the advancement in the field of technology and engineering. Part of this is understanding what goes with the field known as microelectronics and micro-processing.

This project, the design of the 32-Bit Pipelined RISC Microprocessor is an attempt to understand what these fields of microelectronics and micro-processing mean.

Part of this is the advancement in the field of technology and engineering. Part of this is understanding what goes with the field known as microelectronics and micro-processing.

This project, the design of the 32-Bit Pipelined RISC Microprocessor is an attempt to understand what these fields of microelectronics and micro-processing mean.

The completion of this project, with its tedious design procedures and an accompanying text and documentation, is a key towards the realization of the Department's goal of achieving a better value of education. The tools needed to improve on this are already available hence, more researchers are anticipated to make good use of it, for further improvement of the field, for the benefit of all.

References

[1] Abad, Gemini C. et al., 16-Bit Nonpipelined RISC Microprocessor, 1998

[2] Hamacher, V. Carl et al., Computer Organization, 4E. The McGraw-Hill Companies, Inc. 1996

[3] Mano, Morris M. Digital Design, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1984.

[4] Patterson David A. et al., Computer Architecture A Quantitative Approach, Morgan Kaufmann Publishers Inc., San Mateo, California, 1990