Design of Five Stage Pipelined Microprocessor with a 16K Cache Memory

Aglow A George¹, Sanjana Sadasivan², Augusta Sophy³

School of Electronics Engineering, VIT University, Chennai, India

Abstract

Pipelining is a technique in which several instructions are overlapped. With this technique we can achieve a better system throughput. In this work, a five stage microprocessor is designed without interlocked pipelined stages with a 16K cache memory. MIPS (Microprocessor with Interlocked Pipeline Stages) has a 32 bit architecture with instructions of 32 bit and thirty two, 32 bit general purpose registers. Also, a cache memory is designed with a size of 16K having direct mapped configuration. The cache has four blocks, each of 4K memory size. The fundamental purpose of cache memory is to store instructions that are regularly used by the processor during the execution of program instructions. Pipelining includes subdividing the system into different stages. Each stage has buffers in between them. Here it is divided into five stages, instruction fetch stage (IF), instruction decode stage (ID), instruction execute stage (EX), memory stage (MEM) and write-back stage (WB). Each stage implements a parallelism that is executing instructions in parallel. There are cases in pipelining where subsequent instruction cannot be executed in the succeeding clock cycle. These cases or conditions are called hazards. These hazards are basically categorized into three: structural hazard, data hazard and control hazard. In this work all the above mentioned pipeline hazards are rectified.

Keywords: Microprocessor; MIPS; Pipelining; MIPS Instruction Set; Cache Memory; Direct Mapped Cache Configuration;

INTRODUCTION

Processor is a circuitry that controls a computer and works as its central processing unit (CPU). Processor is an integral part of many electronic devices such as printers, calculators, automobiles, etc. A general processor consists of control unit, arithmetic and logic unit (ALU), memory unit and interfacing units for I/O devices. Generally processor is a circuitry that establishes memory for any electronic device.

Microprocessors have two types of instruction set namely, Reduced Instruction Set Computer (RISC) and Complex Instruction Set Computer (CISC). RISC based processors have reduced number of instructions, large set of general purpose registers, instructions of fixed length and simplified addressing modes. RISC processors works on the principle of load-store architecture. Whereas CISC based processors have more number of instructions with different varieties of addressing modes and variable length instructions. RISC processors when compared to CISC processors will consume less area to implement execute faster and have better performance making RISC processors suitable for system on chip circuitry [1].

Pipelining is a technique in which several instructions are overlapped. If pipeline is not used, first instruction will be fetched from memory and only after the execution of that instruction it goes for next instruction that is, during the instruction fetch all other units like execution units will be in idle state. Whereas if pipeline is used, when execution unit is performing its operation, it allows next instruction to be fetched. In this case no unit has to wait for the instruction, in each clock cycle each unit will get instructions continuously.

The performance of a processor is measured by the access time required for the completion of an operation. Generally processors can perform much faster than a large size main memory. In order to increase the performance of main memory we can provide main memories with semiconductor memory having greater speed, but it is not economically possible. We can overcome this problem by introducing a small block called cache which is a high speed memory between processor and main memory. Instructions will be stored in both main memory and cache memory. When an instruction is to be fetched, processor will first check in the cache memory. If the instruction is not available in the cache memory it will check in the main memory [2].

The paper is arranged as follows, section II explains about MIPS processor, section III deals with the hazards in pipeline and how it is resolved, section IV defines the direct mapped cache, section V explains how the cache is implemented and section VI contains results and its discussion.

MIPS PROCESSOR

MIPS stand for Microprocessor without Interlocked Pipelining Stages. It was developed in 1980s by MIPS computer systems Inc. MIPS architecture is based on RISC microprocessor and is used by computer manufacturing companies like Sony, Cisco, Silicon Graphics, NEC etc.

MIPS architecture contains thirty two 32 bit general purpose registers and instructions of length 32 bit. It has a built-in register ‘r0’, which always holds zero value. In the processor, for word access it uses byte addressing. MIPS have three instruction formats such as R-format, I-format and J-format.

Each stage of the processor takes one clock cycle to complete the operation. It has five stages such as instruction fetch (IF), instruction decode (ID), execution (EX), memory access (MEM) and write back (WB). Program counter (PC) gives the address to instruction memory which specifies the address of the next instruction to be executed. This instruction is partitioned into separate fields as shown in table I. The op-
code field of 6 bits [31:26] is progressed to control unit. It determines which category of instruction is to be executed and identifies the control signals that have to be used. By decoding the instruction, it determines which instruction, the ALU has to perform. Rs and Rt are the two source registers for R-format instructions. ‘Rs’ field of 5 bits [25:21] and ‘Rt’ field of 5 bits [20:16] gives the address of operands to the register file. Rd field of 5 bits [15:11] stores the result of the operation. Rt field also behaves as a destination register for I-format instructions. Register file has two ports for read operation and one port for write operation. Register file generates the data required for the execution of instruction based on the address given by the Rs and Rt field. These data are then forwarded to ALU to perform the specified instruction. The operation to be performed on the instruction can be any of the following three types: to compute arithmetic output (for register instructions), to compute comparison (for branch instructions) or to compute memory address (for load or store instruction). For an arithmetic operation the ALU output should be stored in a register. For memory operations, ALU output gives the address to access the data memory. With the help of a separate adder branch target address is calculated during execution stage. Using a sign extention unit, 16 bit offset field of an instruction can be extended to 32 bit signed value. During write back stage, ALU output or memory value will be written back to register file. Figure 1 illustrates each stage of MIPS processor [3].

HAZARDS IN PIPELINING

In pipelining there are cases where the succeeding instructions cannot execute in the desired clock cycle, such cases or events are called hazards. These hazards are of three types.

A. Structural Hazards

When hardware cannot support or execute a given sequence of instructions that has to be executed in the desired clock cycle structural hazard occurs. Consider we have a single memory instead of two memories. Figure 2 shows that, the first instruction is accessing data from the memory whereas, fourth instruction is fetching the instruction from the same memory. This will result in structural hazard. So there is a need to provide two memories in the pipeline instead of one memory [5].

B. Data Hazards

Data hazards occurs when a desired instruction cannot be executed in the given clock cycle as the data required is not available, i.e. the given instruction is having dependency on previous instruction which is still in pipeline. Data hazard can be solved either by forwarding or stalling depending on...
instruction. Forwarding applies for the case of dependency between the register operations. Figure 3 shows the forwarding technique [5].

**Figure 3.** Forwarding in a Five Instruction Sequence

Consider a register instruction having dependency after a memory instruction. For this case stalling technique is applied as shown in figure 4.

**Figure 4.** Stall Inserted Into the Pipelined Sequence of Instructions

C. Control Hazards

Control hazard or branch hazard occurs when there is a need to decide based on the result of one instruction while other instructions are executed. This hazard arises when the instruction that has to be executed in the next clock cycle cannot be executed as the fetched instruction is not the one that is required. Consider a branch instruction if the branch condition is satisfying, then we must fetch the instruction to which sequence has to be branched in the very next clock cycle. But the pipeline cannot know what will be the next instruction as the instruction to which the pipeline should branch will be known in the MEM stage [5].

The solution is to stall the pipeline immediately after we fetch a branch instruction. Stalling refers to wait till the pipeline determines whether it should branch or not and knows which instruction address to be fetched [5]. Figure 5 shows the insertion of bubble and thereby stalling the pipeline.

**Figure 5.** Pipeline Showing Stalling on Every Conditional Branch

DIRECT MAPPED CACHE

Cache organization in which instructions in memory have exactly one location in cache is called direct mapped cache. Direct mapped cache is also known as single set associative cache. The Formula given below can be used to decide the specific location in cache to which main memory can be mapped [5].

\[
\text{Cache line number} = (\text{Main memory block no.}) \mod (\text{number of cache lines})
\]  

(1)

Memory address is divided into block address and block offset. Block address is used to identify the address in cache where the data or instruction from memory can be placed whereas block offset is used to access desired bytes within a block. Block address is further divided into block and tag. Block is used as an address to refer the cache. Block is also known as index. Tag is the most significant bits of block address that is used to differentiate between memory locations that map to the same cache block.

There is a tag memory which stores the tags of all memory addresses. In tag memory each tag can be parallelly searched. When a new block is added to the cache, its tag will be stored in respective memory location.

In direct mapped cache at the time of word retrieval, index segment of memory address is used for the selection of line from cache. Offset bit is used to identify the desired word out of the four words in the cache line. Valid bit is required to indicate whether a cache block is valid or not. Tag portion of the memory address is compared with the tag portion of cache line. If tag comparison matches and a cache block is valid, then a cache hit occurs. Otherwise, it is a cache miss.

The cache memory is empty when instructions are initially stacked into main memory. During the execution of instructions, valid bit is used to indicate whether the block contains the correct line that has to be executed. Valid bits are set to ‘0’ initially therefore first instruction will cause a miss during execution so first instruction will be taken from main memory. Then that block will be loaded into the cache memory so next three successive instructions can be taken from cache memory.
Figure 6 shows the implementation of direct mapped cache. In the figure we can see that tag portion of main memory is compared with the tag portion of cache memory. If both are same, then it will be checked with valid bit to produce a hit or else miss will occur.

Characteristics of the cache are, it is a direct mapped cache of size 16KB. Four blocks with a cache size of 4KB. It has a block offset of 4 bits, cache index of 10 bits and tag of 18 bits. It has a 1 bit valid bit to determine whether the block is valid or not.

IMPLEMENTATION OF CACHE

A. Cache Block and Block Selection

Cache size is 16KB, so there are four 4KB cache blocks. Each block is obtained from cache memory by providing their corresponding address, data and write enables. 8 bits of virtual address (i.e. ICache_VA [11:4]) indicates the cache block address. ‘0’ indicates that a block is enabled for writing whereas, ‘1’ indicates that a block is enabled for reading. Using a MUX we can select between four blocks. The select signal is ICache_PA[13:12]. The above mentioned MUX is illustrated in figure 7 [3].

B. Tag RAM and Tag Selection

Each tag is obtained from tag memory by providing their corresponding address, data and write enables. 8 bits of virtual address (i.e. ICache_VA [11:4]) indicates the tag address. ‘0’ indicates that tag RAM is enabled for writing whereas, ‘1’ indicates that tag RAM is enabled for reading. Input for the tag memory is Tag_Data which is given as Tag_Data = {1'b0, valid_bit, valid_bit_tag}, where valid_bit_tag is the tag segment of address (i.e. ICache_PA[31:14]). Using a MUX, selection between tags is done. The select signal is ICache_PA [13:12]. The above mentioned MUX is illustrated in figure 8 [3].

C. Main Memory and Cache Memory Selection

There are two memories main memory and cache memory. Based on Hit_or_Miss, memory selection is done. For a hit, instructions will be taken from cache memory whereas for a miss, instructions will be taken from main memory. Hit_or_Miss is controlled based on state machine. The above mentioned MUX is illustrated in figure 9 [3].

D. Controlling State Machine

In the state machine there are four states, such as, ‘Idle’, ‘Cache Miss’, ‘Cache Hit’ and ‘Cache Read’. Based on the states, the signals such as Hit_or_Miss, valid bit and the write enable for all the four blocks are set.

During Idle state, set the signals Hit_or_Miss and Valid bit to zero and also write enables for all the four blocks are set to read mode. As the Hit_or_Miss signal is set to ‘0’, at that time instruction will be read from main memory. After Idle state, the transition is directly to Cache Miss state.

In Cache Miss state, set the signals such as Hit_or_Miss, Valid bit to ‘0’ and ‘1’ respectively. The write enable signals for the cache blocks are enabled to write mode based on the physical address. If the physical address is ‘0’, then the write enable is set to write mode only for the first block. After Cache Miss state, the transition is directly to Cache Hit state.
During Cache Hit state, comparison of tag occurs. Based on the result of comparison, transition of state occurs. Tag comparison is done by comparing tag segment of the address with the tag portion of the selected line. If the comparison satisfies, it will remain in the same state. At this state, the signals, Hit_or_Miss, Valid bit are set to ‘1’ and ‘0’ respectively. The write enable signals for the cache blocks are enabled to read mode based on the physical address. If the physical address is ‘0’, then the write enable is set to read mode only for the first block. If the comparison fails, transition can be to either Cache Read or Cache Miss state. In order to determine to where the transition occurs, consider the block change. If there is a block change then the transition will be to Cache Miss state or else to Cache Read state. If the transition is to Cache Miss state, then the signals Hit_or_Miss & Valid bit are set to ‘1’ & ‘0’ respectively and the write enable signals are set to read mode for all the four blocks. Whereas if the transition is to Cache Read state, then the signals Hit_or_Miss and Valid bit are set to ‘0’ & ‘1’ respectively and the write enable signals for the cache blocks are enabled to write mode based on the physical address. If the physical address is ‘0’, then the write enable is set to write mode only for the first block.

From the Cache Read state, transition occurs to Cache Hit state. There the signals Hit_or_Miss and Valid bit are set to ‘1’ & ‘0’ respectively. Write enable signals for each block are set based on physical address. Write enable signals are set to read mode [4]. The state machine is illustrated in figure 10.

### RESULTS AND DISCUSSIONS

After the design of direct mapped cache and each stage of pipelined MIPS processor, the cache memory is combined with the MIPS processor and all the MIPS I instructions are executed. This section explains the method implemented to resolve the pipeline hazards.

#### A. Register Dependency and Load-Store With Register Dependency

In order to resolve the dependency between two register instructions (i.e. Rs-Rt dependency) or a register and load or store instruction (i.e. load or store followed by register) we have to check following dependencies: i) Between ID and EX ii) Between ID and MEM iii) Between ID and WB iv) Between ID-EX and ID-MEM v) Between ID-EX and ID-WB vi) Between ID-MEM and ID-WB vii) Between (ID-EX and ID-MEM), (ID-EX and ID-WB) and (ID-MEM and ID-WB)

i) Between ID and EX

In order to rectify this dependency, consider a MUX (M1). The instruction in EX stage can be either R-format or I-format. If the instruction is of R-format, then a signal is generated by comparing Rd of EX stage with Rs &Rt of ID stage and op-code of instruction in EX stage should be that of R-format instructions. Whereas, if the instruction is of I-format, then a signal is generated by comparing Rt of EX stage with Rs &Rt of ID stage and op-code of instruction in EX stage should be that of I-format instructions. The select signal for the forwarding MUX M1 is the OR output of both the signals. When the select signal is ‘1’ (i.e. when there is dependency), result of EX stage will be forwarded as the output of MUX or else register value itself will be forwarded.

ii) Between ID and MEM

In order to rectify this dependency, consider a MUX (M2). The instruction in MEM stage can be either R-format or I-format. If the instruction is of R-format, then a signal is generated by comparing Rd of MEM stage with Rs &Rt of ID stage and op-code of instruction in MEM stage should be that of R-format instructions. Whereas, if the instruction is of I-format, then a signal is generated by comparing Rt of MEM stage with Rs &Rt of ID stage and op-code of instruction in MEM stage should be that of I-format instructions. The select signal for the forwarding MUX M2 is the OR output of both the signals. When the select signal is ‘1’ (i.e. when there is dependency), result of MEM stage will be forwarded as the output of MUX or else register value itself will be forwarded.

iii) Between ID and WB

In order to rectify this dependency, consider a MUX (M3). The instruction in WB stage can be either R-format or I-format. If the instruction is of R-format, then a signal is generated by comparing Rd of WB stage with Rs &Rt of ID stage and op-code of instruction in WB stage should be that of R-format instructions. Whereas, if the instruction is of I-format, then a signal is generated by comparing Rt of WB stage with Rs &Rt of ID stage and op-code of instruction in WB stage should be that of I-format instructions. The select signal for the forwarding MUX M3 is the OR output of both the signals. When the select signal is ‘1’ (i.e. when there is dependency), result of WB stage will be forwarded as the output of MUX or else register value itself will be forwarded.
iv) Between (ID-EX) and (ID-MEM)

In order to rectify this dependency we provide a MUX (M4). The select line for M4 is generated by comparing the select lines of ID-EX MUX & ID-MEM MUX (i.e. M1 & M2). When the select signal of M4 is ‘1’, result of ID-MEM MUX (i.e. M2 output) will be forwarded as the output of M4 or else result of ID-EX MUX (i.e. M1 output) will be forwarded.

v) Between (ID-EX) and (ID-WB)

In order to rectify this dependency, consider a MUX (M5). The select line for M5 is generated by comparing the select lines of ID-EX MUX & ID-WB MUX (i.e. M1 & M3). When the select signal of M5 is ‘1’, result of ID-WB MUX (i.e. M3 output) will be forwarded as the output of M5 or else result of ID-EX MUX (i.e. M1 output) will be forwarded.

vi) Between (ID-MEM) and (ID-WB)

In order to rectify this dependency, consider a MUX (M6). The select line for M6 is generated by comparing the select lines of ID-MEM MUX & ID-WB MUX (i.e. M2 & M3). When the select signal of M6 is ‘1’, result of ID-WB MUX (i.e. M3 output) will be forwarded as the output of M6 or else result of ID-MEM MUX (i.e. M2 output) will be forwarded.

vii) Between (ID-EX and ID-MEM), (ID-EX and ID-WB) and (ID-MEM and ID-WB)

In order to select the output from above three combinations, consider a MUX (M7). The select line for M7 is generated by comparing the select lines of ID-EX MUX, ID-MEM MUX & ID-WB MUX (i.e. M1, M2 & M3). The output is forwarded as shown in Table II. Figure 11 shows the MUXes used to resolve the above mentioned dependencies.

<table>
<thead>
<tr>
<th>Table II. Output Forwarded By The MUXes</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1_sel</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Figure 11 shows the MUXes used to resolve the above mentioned dependencies.

B. Load and Register Dependency

In case of dependency between load instruction and a register instruction, i.e. dependency between register instruction in ID and load instruction in EX stage, stall the pipeline by inserting a NOP between them. If we insert a NOP, the dependency will be between ID and MEM stage, i.e. the load instruction in the MEM stage and the register instruction in the ID stage. That dependency can be resolved with the help of forwarding unit. Figure 13 shows the simulation result that verifies the instructions having register dependency followed by load.

Figure 12 shows the simulation result that verifies the instructions having register dependency and load-store with register dependency.
C. Load and Store Dependency

In order to rectify this dependency, consider a MUX (M8). A signal is generated by comparing Rt of EX stage with Rs & Rt of ID stage. The value of that signal is registered, as the output of load word will be available only in MEM stage. So the select line for M8 (i.e. M8_sel) is the registered signal.

When comparing Rt of EX and Rs of ID, whether there is dependency or not output of store instruction will be the register value. Dependency showing in this case is for the address because value in Rs of store instruction is used for the calculation of memory address. That dependency can be resolved by the forwarding MUX used in load-store with register dependency.

If there is dependency between Rt of EX and Rt of ID, output of load instruction should be forwarded or else register value should be forwarded. Table III shows how select signal for M8 is generated.

<table>
<thead>
<tr>
<th>Rt_dep</th>
<th>Rs_dep</th>
<th>M8_sel</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Reg value</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Reg value</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Out of load inst</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Out of load inst</td>
</tr>
</tbody>
</table>

Figure 14 shows the simulation result that verifies the instructions having load and store dependency.

![Figure 14. Simulation of instructions having load and store dependency](image)

CONCLUSION

In this paper, a five stage 32 bit MIPS processor with a 16k direct mapped cache memory is designed and implemented successfully. This processor has five stages of pipelining where one instruction is executed in each clock cycle. Design of cache memory is integrated with the pipeline stages and various R-format, I-format and J-format instructions were verified by this processor. Designs of hazard detection unit and forwarding unit have been done successfully which helps to resolve all pipeline based hazards. The processor is designed using Verilog-HDL and implemented using QuestaSIM 10.3.

REFERENCES


