Amirkabir University of Technology

Final Project

SAYEH Extension
(Cache Controller / Pipelined SAYEH)

Farzan Dehbashi
Parham Alvani

supervised by
Dr. Saeid Shiri Gheydari

May 30, 2017

Contents

Chapter 1

Overview

This project is an extension to your previous project (SAYEH Basic Computer) which makes it capable of caching data in a separated module between the *Memory* and the *Processor*.

Chapter 2

Cache Module

Figure1 shows a **2 way set associative cache**. In such memory each set has **64 rows** and each row only has **a single 16-bit word**.

In this structure we don't need any offset (why?). As you have learnt, *Index* is used to select desired row in cache. after selection, input tag would be compared with the existing tag of the selected row and in case of accommodation cache declares *hit* and *miss* elsewhere. if *hit* occurs desired data would form the output.

SAYEH uses cached data for both Instruction and Data purposes. *Data Array* module has already been implemented and can be found here.
Main cache components are as the following:

- **Data Array:** Responsible for retention of each way.
- **Miss-Hit Logic:** Checks occurrence of accommodation.
- **Most Recently Used (MRU) Array:** Selects the desired row, to be replaced.
- **Tag-Valid Array:** Stores *Tag* and *Valid* section of each cache's row.

2.1 Data Array

It is designed to maintain *Data* section of each row.

In this array *Address* signal, selects the row which is going to be written in or read from. read data would be written to output by *Data* signal. Obviously writing process is synchronous. as *WriteEnable* signal become "one" (active), input data (defined by *WriteData* signal) would be written in the specified address.

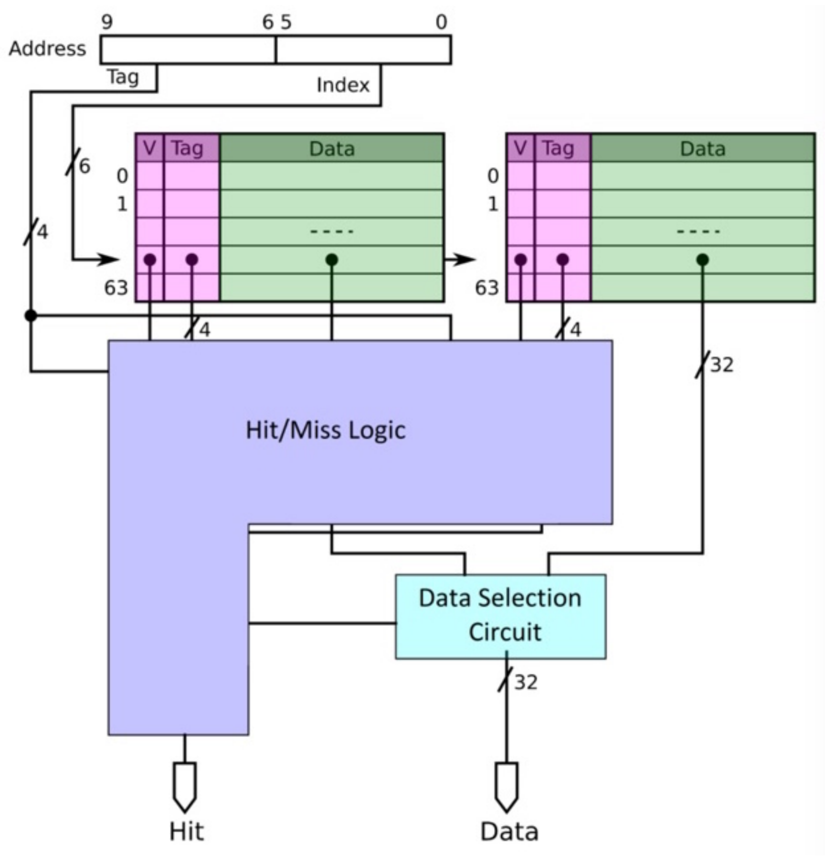

Figure 2.1: Cache structure

2.2 Miss-Hit Logic

As the title discloses, this module is designed for determination of accommodation. in case of any accommodation, this module determines the set accommodation has happened in.

Write0 contains of *Valid* and *Tag* signals of the first set and also *Write1* contains the second's (refer to figure1).*Tag* signal is the tag we are searching for.

To clarify this module's functionality this example looks useful.

Example. Assume that we are searching for A, it's $tag = 4(0100)$ and $index = 20$. So we check 20th row of cache, assume that cache contains of a piece of data with these features in way-0: $tag = 7, valid = 1$ and in way-1: $tag = 4, valid = 1$. then *Write0* signal would become 10111 means and *Write1* signal would be 10100. As in this case desired data has been found in way-1, outputs would be $hit = 1, write1\_valid = 1$ and $write0\_valid = 0$.

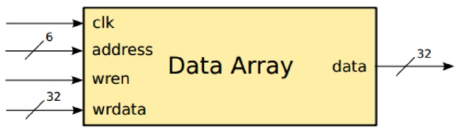

Figure 2.2: Data Array Module

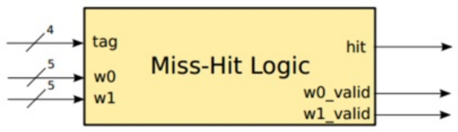

Figure 2.3: Miss-Hit Array

2.3 Tag-Valid Array

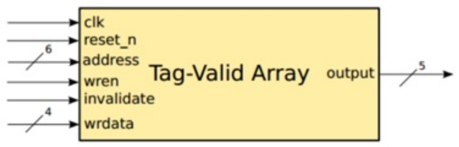

Figure 2.4: Tag-Valid Array

This module is about to save 4-bit Tag and 1-bit Valid.

Writing and also reading process in this module is like Data Array module, there are only two differences, *WriteData* and *Invalidate*. *WriteData* contains of 4-bit input Tag and *Invalidate* signal is to declare any alternation of Valid bit in this array, if *Invalidate* is one, Valid bit in the row specified by *Address* signal would become zero.

2.4 Cache Replacement Policy

As you know there is a policy declares the suitable transference **time** from Memory to Cache, it also declares **where** to place the data just came from Memory. For simplicity in SAYEH we use MRU(Most Recently Used) policy, in such a policy we select the row which has been used the most recently.

This Module is about to know about the row which is currently replacing and if one has been replaced this module would update the row which has been used the most in recent time.

This module's output is a single bit declares weather we should replace way1 or way2.

2.5 Cache Controller

Cache controller first checks that given address is in cache or not, if it cached the given address and it returns that address and if not it gets the address from the memory and caches it.

Chapter 3

Pipelined Controller

3.1 Von Neumann Architecture

This architecture narrates that there are 4 steps needed to perform a complete action to an instruction:

- **Fetch:** In this step processor attempts to fetch the instruction from *Main Memory* into the processor.
- **Decode:** In this step processor decodes the instruction to get to know the job desired to do.
- **Execute:** This step's functionality is to do the desired job, which now is known according to the former step. note that all instructions need to be executed.
- **Write-back:** Finally, processor would write back the data produced in the previous step into the *Main Memory*.

3.2 What is Pipeline?

Let's take an example: the following figure is a generic pipeline with four stages: fetch, decode, execute and write-back. The top gray box is the list of instructions waiting to be executed, the bottom gray box is the list of instructions that have had their execution completed, and the middle white box is the pipeline.

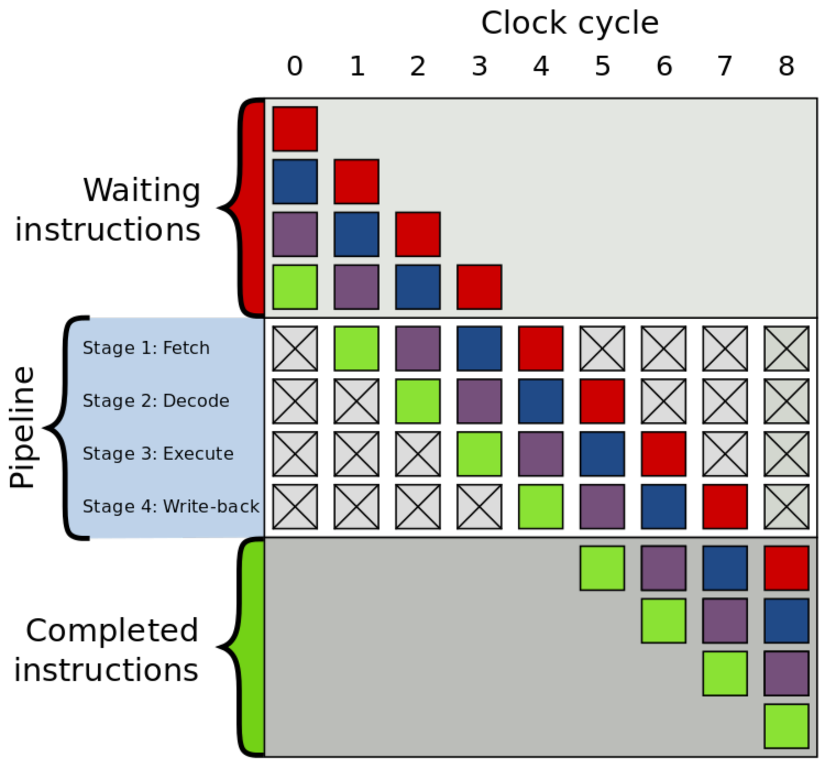

Figure 3.1: Generic 4-stage pipeline; the colored boxes represent instructions independent of each other

Execution will go on like the folloing:

- **Time= 0:** Four instructions are waiting to be executed.
- **Time= 1:** The green instruction is fetched from memory
- **Time= 2:**
  - The green instruction is decoded
  - The purple instruction is fetched from memory
- **Time= 3:**
  - The green instruction is executed (actual operation is performed)
  - The purple instruction is decoded
  - The blue instruction is fetched
- **Time= 4:**

  - The green instruction's results are written back to the register file or memory
  - The purple instruction is executed
  - The blue instruction is decoded
  - The red instruction is fetched

- **Time= 5:**
  - The execution of green instruction is completed
  - The purple instruction is written back
  - The blue instruction is executed
  - The red instruction is decoded

- **Time= 6:**
  - The execution of purple instruction is completed
  - The blue instruction is written back
  - The red instruction is executed

- **Time= 7:**
  - The execution of blue instruction is completed
  - The red instruction is written back

- **Time= 8:**
  - The execution of red instruction is completed

- **Time= 9:**
  - The execution of all four instructions is completed

3.3 What is Pipeline Stall (Bubble)?

In computing, a bubble or pipeline stall is a delay in execution of an instruction in an instruction pipeline in order to resolve a hazard.

During the decoding stage, the control unit will determine if the decoded instruction reads from a register that the instruction currently in the execution stage writes to. If this condition holds, the control unit will stall the instruction by one clock cycle. It also stalls the instruction in the fetch stage, to prevent the instruction in that stage from being overwritten by the next instruction in the program.

To prevent new instructions from being fetched when an instruction in the decoding stage has been stalled, the value in the PC register and the instruction in the fetch stage are preserved to prevent changes. The values are preserved until the bubble has passed through the execution stage.

The execution stage of the pipeline must always be performing an action. A bubble is represented in the execution stage as a NOP instruction, which has no effect other than to stall the instructions being executed in the pipeline.

3.4 Pipeline in MIPS Processor

In this processor we have 5 stage for each instruction:

- IFetch: Fetch Instruction, Increment PC
- Decode: Instruction, Read Registers
- Execute: Calculate Address & Perform Operation
- Memory: Save & Load
- Writeback

and as you learn in class, if we want to create pipeline for this processor we use following diagram:

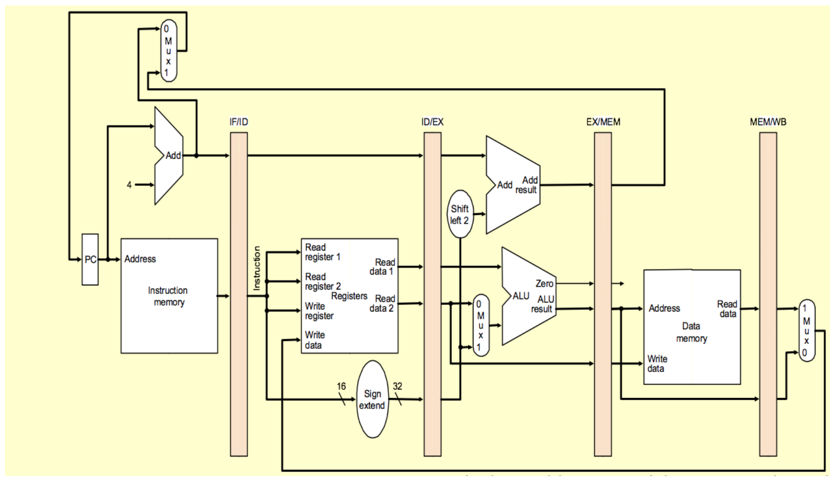

Figure 3.2: MIPS 5-stage pipeline

In this project we want you to draw a SAYEH computer with pipeline.

3.5 What is Hazards?

In the domain of central processing unit (CPU) design, hazards are problems with the instruction pipeline in CPU micro-architectures when the next instruction cannot execute in the following clock cycle, and can potentially lead to incorrect computation results.

Hazard types are described in the following sections.

3.5.1 Data Hazard

Data hazards occur when instructions that exhibit data dependence modify data in different stages of a pipeline.

- read after write (RAW), a true dependency
- write after read (WAR), an anti-dependency
- write after write (WAW), an output dependency

Read after write (RAW)

A read after write (RAW) data hazard refers to a situation where an instruction refers to a result that has not yet been calculated or retrieved.

```
i1. R2 <- R1 + R3
i2. R4 <- R2 + R3
```

Write after read (WAR)

A write after read (WAR) data hazard represents a problem with concurrent execution.

```
i1. R4 <- R1 + R5
i2. R5 <- R1 + R2
```

Write after write (WAW)

A write after write (WAW) data hazard may occur in a concurrent execution environment.

```
i1. R2 <- R4 + R7
i2. R2 <- R1 + R3
```

3.5.2 Structural hazards

A structural hazard occurs when a part of the processor's hardware is needed by two or more instructions at the same time.

3.5.3 Control hazards

Branching hazards (also termed control hazards) occur with branches.

3.6 Let's eliminate hazards

In this project we want you to create hazard control unit like one that you see in MIPS in class for SAYEH computer.

3.7 What you should do ?

Do these items in paper (no VHDL !)

1. Draw SAYEH computer pipeline.
2. How many registers do you add for creating pipeline ?
3. Which type of Hazards we have in SAYEH computer pipeline ?
4. Design a hazard controller unit for these hazards.

Chapter 4

Handin Instructions

1. All your project should be implemented by VHDL not Verilog or...

2. About 10% of your score would be devoted to the report you deliver within your projects code. In this report you would explain the whole job and anything special you have done. (note that it shouldn't be shorter than this document and should be typed!)

3. Cache Controller should be designed by an Finite State Machine (FSM) and this FSM should be mentioned in your project report.

4. In addition to bonus sections mentioned beforehand, implementation of this project on *Altera DE2* FPGA boards will result in extra mark as well.

5. You can simulate your code with softwares like: Modelsim, Altera Quartus, GHDL, Xilinx ISE, Active HDL, Xilinx Vivado Design Suite,... (note that Modelsim and Quartus has got free versions and GHDL is totally free so it's better to obey *Copyright*) but the TA's laptop is only equipped with Modelsim and in case of delivery you should be able of running your code in that environment.

6. All questions would be answered by the course's email: computerarchitecture95@gmail.com.

7. Place all your modules and report into a .zip file named as "FirstName LastName StudentID" before upload. if any other module is used in your implementation but hasn't been mentioned in this document place it in it's proper place next to modules within the same hierarchy.

8. Deadline of the project is up to the last day of regular classes before the exam preparation week for any change contact the mentioned email.

9. In case of delivery, your code will be downloaded by the responsible TA from Moodle, so the only way to convey your code is Moodle and in if you

need to reform your code please upload it when possible to be used in the due date.

10. **Cheating Alert!**
    (a) All your codes would be processed, both manually and automatically and in case of any similarity by any means, both of individuals involved, would be fined by getting -35 percent of the project's score (note that if pushing your code to Github or any other VCS, exposing your code to a friend or ... results in unexpected similarities of others, you ALL, are responsible!).
    (b) Any cooperation beyond members of the group is forbidden!
    (c) The only source you are allowed to copy form, is AUT/CEIT/Arch101 repo which has been devoted to this course, copying any other source from the Internet,... would be consider just like from another classmate.
    (d) By the way, in case of any similarity with any of the previous students of this course, you blame the same.

11. Remember that, any HDL(Hardware Design Language) is a piece of art :) and could be really enjoyable, if you try your best to understand what's going on instead of just doing it to make it end.

Good Luck