|  |
| --- |
| MIPS 8 Stages PIPELINE SIMULATOR |
|  |

|  |  |
| --- | --- |
| Date | A SIMPLE SIMULATOR FOR BEGINNERS |

|  |
| --- |
| Type the abstract of the document here. The abstract is typically a short summary of the contents of the document. |

MIPS 8 Stages  
PIPELINE SIMULATOR

A SIMPLE SIMULATOR FOR BEGINNERS

# INTRODUCTION

In this project we built a simulator that simulates an advanced version of the pipelined MIPS CPU discussed in class, namely 8 stages MIPS CPU.

In MIPS 8 Stages, IF stage is now split into 2 stages IF1 & IF2. Data memory stage is now 3 stages instead of 1, MEM1, MEM2, MEM3, with data available only after 2nd stage. We can assume all cache accesses are cache hits. This means our pipeline will now have 8 stages.

# WORKING

In our main function, Pipeline stages are called in reverse order and in a loop for every processor cycle. The loop is executed until all the instructions are processed and written back. Timing statistics are printed to output file in corresponding stages. Final contents of memory and registers are written to output file out of while unnecessary newline character in buffer is flushed, input and output files are closed and user is prompted an option whether to run another simulation or exit.

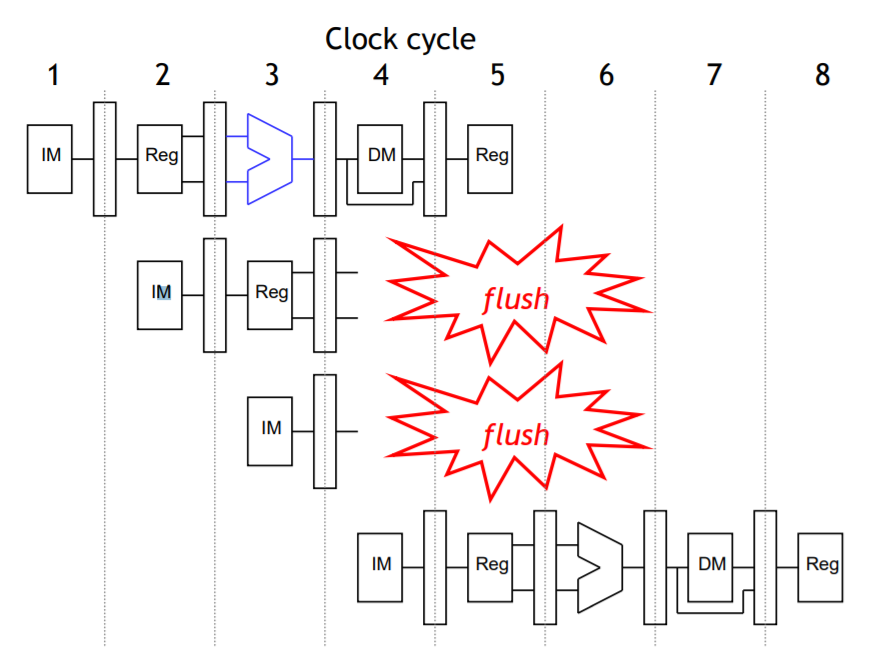
The printmemory () function prints memory content of all register’s memory [] is an array of 256 integers. All the memory content and address if not zero is printed to the output file.

The execute () function processes instructions based on opcode in the structure instruction. It checks for state of pipeline and checks for data dependencies. If found any data dependency, stalls the pipeline. Also, if predict not taken or flushing technique are used, branch instructions target address is compared with the label of all instructions and correct target instruction is fetched in the next cycle.

# TECHNIQUES

## FLUSHING

* We must flush one instruction (in its IF1 stage) if the previous instruction is BEQ and its two source registers are equal.
* We can flush an instruction from the IF2 stage by replacing it in the IF/ID pipeline register with a harmless no instruction.
* MIPS uses sll $0, $0, 0.
* This happens to have a binary encoding of all 0s: 0000 .... 0000.
* Flushing introduces a bubble into the pipeline, which represents the one cycle delay in taking the branch.
* The IF1.Flush control signal shown on the next page implements this idea, but no details are shown in the diagram.



This is 5 stage

flushing

technique to

illustrate in

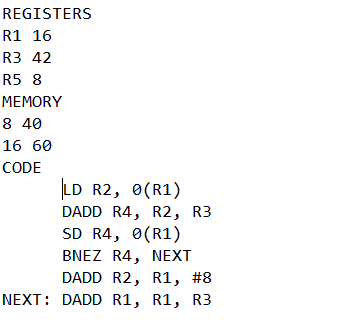
stages.

## BRANCH TAKEN / NOT TAKEN

Most of the work for a branch computation is done in the EX stage. The branch target address is computed. The source registers are compared by the ALU, and the Zero flag is set or cleared accordingly. We made a simple solution to stall the cycle of particular instruction when the branch is taken or not.

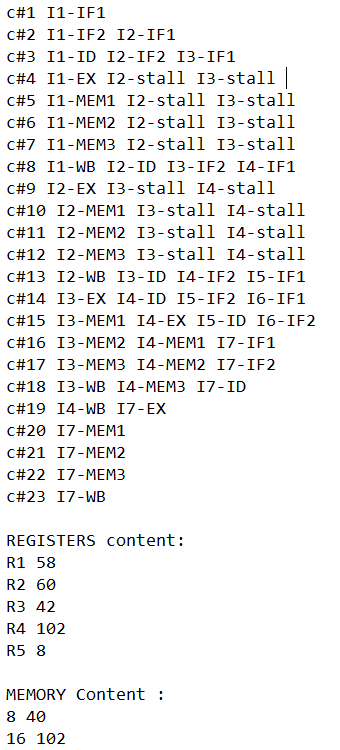
# SOME SCREENSHOTS:

INPUT:

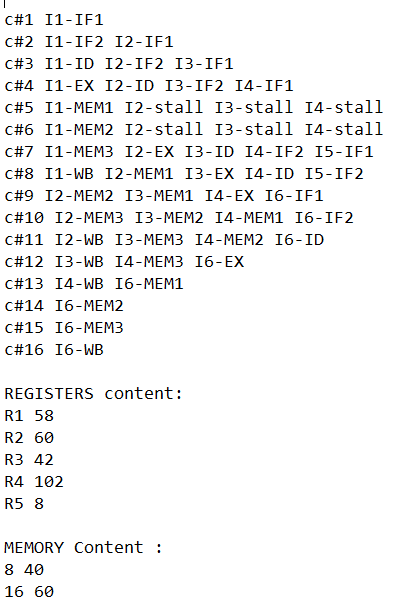


OUTPUT:

1. FLUSHING



1. BRANCH TAKEN



1. BRANCH NOT TAKEN

