Lab# 13 A PIPELINED IMPLEMENTATION

Instructor: I Putu Danu Raharja.

Objectives:
Learn how to observe pipelining mechanism.

Method:

Observing the effect of pipelining using PCSPIM simulator.

Preparation:

Read the slides.

File To Use:

13.1 OVERVIEW:

As explained in the class, the 32-bit MIPS processor has a five-stage pipeline implementation. The functions performed in these five stages are as follows:

1. **Instruction Fetch Stage (IF)**: Fetch the instruction from cache memory and load it into the instruction register. Increment the program counter by four.
2. **Instruction Decode/register file read (ID)**: Fetch values Rs and Rt from the register file. If this is a branch instruction and the branch condition is met, then load the PC with the branch target address.
3. **Execution/address calculation (EX)**: Perform an arithmetic or logic operation in the ALU and load the result register. This is the stage where an addition is performed to calculate the effective address for a load or store instruction.
4. **Memory access (MEM)**: If the instruction is a load, a read from the data cache occurs. If the instruction is a store, write to the data to cache occurs. Otherwise, pass the data in the result register on to the write back register.
5. **Write back (WB)**: Store the value in the write back register to the register file.

13.2 DATA HAZARD

The term data hazard refers to the following situation. Suppose we have 3 sequential instructions x, y, and z that come into the pipeline and suppose also that x is the first instruction into the pipeline followed by y and z. If the results computed by instruction x are needed by y or z, then we have a data hazard. The hardware solution to this problem

is to include forwarding paths in the machine's datapath so that even though the results have not yet been written back to the register file, the needed information is forwarded from the result register or the write back register to the input of the ALU.

One type of data hazard cannot be solved with forwarding hardware. This is a situation where a load instruction is immediately followed by an instruction that would use the value fetched from memory. The only solution to this problem is to rearrange the assembly language code so that the instruction following the load is not an instruction that uses the value being fetched from memory. In recognition of this situation we refer to load instructions on pipelined processors as being **delayed loads**. If existing instructions in the algorithm cannot be rearranged to solve the data hazard then a **no-operation** (nop) instruction is placed in memory immediately following the load instruction.

13.3 CONTROL HAZARD

Associated with every branch or jump instruction we have a **control hazard**. The improvement of this kind of hazard is to assume that the branch will not be taken and thus continue execution down the sequential instruction stream. If the branch is taken, the instructions that are being fetched and decoded must be discarded.

13.4 EXERCISE

1. Download the file *Lab13a.s* from the Website. Run this program using single-step mode in a pipelined implementation. Fix the problems in such a way the program can run correctly.
2. Download the file *Lab13b.s* from the Website. Fix the problems in such a way the program can run correctly.