Mobile Processor Programming assignment #3: Simple pipelined MIPS

due date 30th May

Introduction

Now, we will build an enhanced MIPS emulator with 5 pipeline stages. Pipeline has been a big change in designing microarchitecture of modern microprocessors. Pipelining makes users best utilize the given hardware by concurrently executing multiple instructions at the same time. In five stage pipeline, instruction is processed in five distinct stages (or pipeline stages): IF, ID, EX, MEM, WB.

In a pipelined processor, an instruction is processed with multiple cpu clock cycles. For example, our five-stage pipeline MIPS, one instruction execution is completed in five cycles. At the first cpu clock, an instruction is fetched into CPU from memory. In the next cycle, the fetched instruction is decoded, and register values are read. Then, the decoded instruction is executed by ALU in the execution stage, which is the third cycle. Next, the execution accesses memory if needed in the fourth cycle. Finally, the register update and write back is performed in the fifth cycle.

In addition to multi-cycle execution, in a pipeline processor, the execution of different instructions are overlapped. Namely, five different instructions can be executed at the same time, each of which instructions utilizes different hardware unit. Thus, it increases performance in terms of throughput. Now, the clock cycle time does not have to rely upon the slowest instruction execution, and we can maximize hardware utilization by concurrent execution.

One concern about pipeline execution is hazard/or pipeline stall. Hazard represents the state at which pipeline is not full because an instruction in the pipeline is dependent upon another instruction, hindering the concurrent execution. Modern microprocessors address pipeline hazard with various techniques, including register forwarding, loop unrolling, instruction scheduling, branch prediction, speculative execution, etc.

In this programming assignment, you will emulate simple pipeline processor, fully in software. We consider the MIPS processor with five pipeline stages. You should resolve pipeline hazards, or dependency to fill in the pipeline as much as possible.

This program assignment is critical in your evaluation, so please work hard to complete in your schedule. If you need help, please ask for the help. (I am here for that specific purpose) I, of course, welcome for any questions on the subject. We will have demo time for some of your work (good/bad). In demo, you are asked to explain your software (structure/implementation). If you can, please think about the visualization of the emulator. Note for the one strict rule that do not copy code from any others. Deep discussion on the subject is okay (and encouraged), but same code (or semantics) will result in sad ending.

Extra implementation is highly encouraged, meaning that various branch prediction, data forwarding technique implementation would differentiate your work. (Unique /creative approaches are more appreciated, even trial has significant credit.)

Advice one: You may need some time to think about your software structure, and operation, and pipeline. Therefore, think with note and pen, before rushing the code work.

Advice two: Make basic structure robust, and reliable. If you make code on suspicious base, you are easy to lose the way. Make print logs, before you have gone too much. Make basic structure, at the first hand. Make some checkpoints before you got failure.

Advice three: You can make things work from small pieces to a larger one. You may consider writing code in the following sequences.

-. Make sure you have sound single cycle implementation
-. Establish pipeline structure
-. Make debug function

-. Make basic control flow work (sequential execution path)
-. Make reg. memory access instructions work
-. Make pipeline with forwarding
-. Make jump/branch work

Last advice, but not least: Begin as early as possible. Ask for help as early as possible. Try as early as possible. They are for your health-care.

The followings are specific requirements for your program.

1. The Objective: compare performance of pipelined vs. single-cycle u-processor.
    A. Your program should produce correct output. (and execution should be the same with code semantics)
    B. Compare the number of CPU clock cycles for single-cycle execution and pipelined execution. In theory, we can reduce CPU clock cycle time by 1/5 in 5-stage pipelined execution.
2. Machine Initialization:
    A. Before the execution, the binary file is loaded into the memory. Note that memory can be a data structure defined with large array. Read all the file content into your memory (data structure). Assume that all register values are all zero, except for RA, of which value is 0xFFFF:FFFF. Thus, when your PC becomes 0xFFFF:FFFF, your machine completes execution, and halts. Your application is loaded to 0x0, and stack pointer is 0x1000000.
3. Implementation requirements:
    A. You should implement five-stage pipelined MIPS processor emulator. The emulator should implement IF, ID, EX, MEM, WB stages.
    B. For each stage, instruction execution has to be latched. That is, you have to store the execution state for each stages. Latches are the temporal storage that remember the execution states for each stages execution.
    C. You can have separated instruction memory and data memory.
    D. The emulated processor runs with the emulated clock cycle. You need to generate clock, which is a variable. The clock cycle increases only after all the work done in the all the stages. Note that max. 5 different instructions can run at the same cpu clock cycle.
    E. You need to resolve data dependency among instructions. That is, you have to implement either stall, forwarding or register renaming. Forwarding is recommended as a basic implementation, but you can optionally choose to implement stall. You should make sure that it works, as in the program order (semantics).
    F. You need to resolve control dependency among instructions. According to the MIPS ISA, one delayed branch slot is defined. Invalidation is basic implementation, but you can optionally implement branch prediction mechanisms.
4. Output: At the end of each cycle, the simulator prints out the changed micro-architectural state from the previous state. Micro-architectural state consists of set of general register, PC, memory as well as invisible latch values and internal data structures.
    A. You can print out only changed state.
    B. Think about good presentation of pipelined execution.
    C. You can give log option in build time (e.g. –DDEBUG_IF)
5. Program completion/terminal condition:
    A. At the end of the execution, you need to print out the calculated result value. We know the end of execution by moving PC to 0xFFFF:FFFF. Along with the calculated result, you should print out the statistics of the execution.
    B. The final return value is stored in v0 (or r2) register.
    C. The print out statistics may include: the total # of instructions, # of memory operation instructions, # of register operation instructions, # of branch instructions, # of not-taken branches, and # of jump instructions.
6. Evaluation:
    A. Different credits are given for implementations, and demos.

   B. More credits are allotted for different (hardware-based) hazard resolving techniques such as branch prediction, forwarding, stalling, etc.

7. Input program binary: You can use your own MIPS code (compiled) or download binary.
   A. To make MIPS binary, use MIPS cross compiler toolchain. First, write C code, and compile mips-linux-gnu-gcc with "–c –mips1 " option (compile only). Second, translate the object binary, stripping ELF headers. mips-linux-gnu-objcopy –O binary –j .text input.o input.bin. Third, check the integrity between binary files: objectdump, mips-linux-gnu-objdump –d input.o , vi –b input.bin → check with :%!xxd option. (google internet) or hexedit tool
   B. To use with multiple functions, you need to hand-carve binary for function calls (jal 0, originally). For example, jal to 0x40 can be encoded as (0x0C000010).
   C. Sample input file examples can be downloaded from the e-learning site. Two representative example programs are 1) summation from 1 to 10, 2) calculating 4 factorial.
   D. Some good examples can be N-th Fibonacci number calculation, or capturing N-way of counting coins. Students who took system programming course can try them to implement in C/C++ language.

Enjoy MIPS emulator programming!

Good Luck for your health!