Fall 2015, CIS 314 Computer Organization

Major Class Project

Assigned
November 4, 2015

Milestone I - 10% of Course Grade
Milestone II - 10% of Course Grade
Final Report - 18% of Course Grade

Final Report Due, November 25th, 2015

https://www.cs.uoregon.edu/Classes/15F/cis314/index.html

Summary:

In lecture, we mentioned that modern digital computer has three major functional hardware units: CPU, Main Memory and Input/Output (I/O) Units. We saw that the *Central Processing Unit* (CPU) consists of the Control Unit, the ALU and Registers. The *Control Unit* monitors and directs sequences of instructions. The *Arithmetic-Logic Unit* (ALU) performs arithmetic and logical operations. *Registers* are memory cells built right into the CPU for temporary data needed by the ALU and the *Program Counter* (PC). We have the Clock to synchronize the CPU operations.

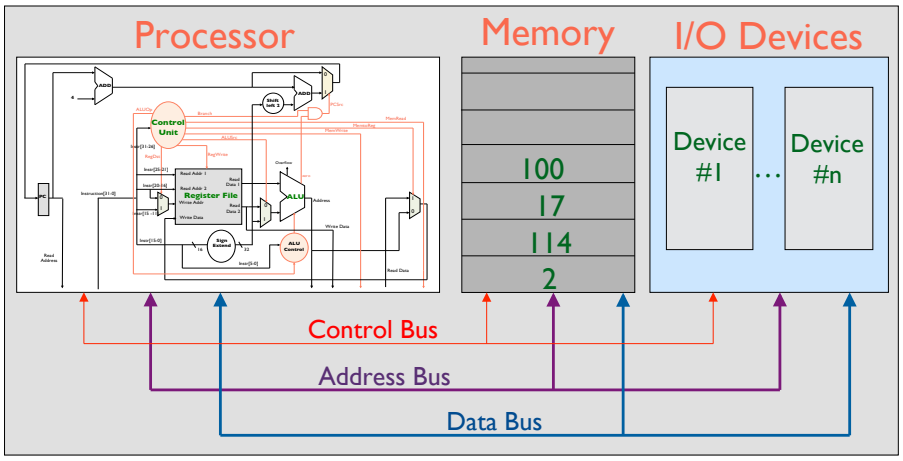

Figure 1: Canonical view of a computer system.

We will soon introduce the concept of pipelining for parallelism. We will observe the main difficult in pipelining is pipeline hazards, that is, data dependencies between instructions in the pipeline. We will discuss two possibilities for dealing with pipeline hazards in class: stalling and data forwarding. Stalling means that we stop issuing instructions when we detect that the next instruction that we will issue depends on the result of in-flight instructions. An alternative to stalling is data forwarding. Rather than waiting for the instruction to complete and commit data to the register file, we forward the data from a earlier instructions in the pipeline directly to later instructions.

The memory bottleneck is one of the paramount design concerns in modern processors. As we will see in lecture, while processor speeds have increased exponentially over the last 30 years, main memory speeds have only increased linearly since memories have been optimized for capacity rather than latency. Since memories are slow relative to the processor and memory accesses are frequent, a major area of

computer research is improving the perceived latency of the memory. In this project, we will explore the concept of memory caching, a common and highly effective mechanism for improving memory latency. A cache is a fast, but necessarily small memory that contains recently accessed areas of memory. The cache has a much lower access and update latency than the main memory; thus, if the memory locations that the process requires are present in the cache, the processor can obtain the data in a small number of cycles (as opposed to the many hundreds of cycles it would take to access the main memory), thereby improving processor performance.

Project has three major components:

1- Implement a single cycle processor simulation in C running MIPS assembly language with a main memory unit
2- Convert the single cycle processor into a pipelined 5-stage processor
3- Build a direct-mapped cache structure between the main memory and the 5-stage processor

The objectives for this project are (1) to understand the main functional units of a computer system by building them in software and (2) to enhance student's knowledge and understanding of the C programming language and programming in general.

This project's delievables:

1- Single cycle processor simulation in C: implement (a) a register file, (b) an ALU (c) control logic, and (d) main memory. [***C code and report - Due date: November 11th, 2015***]

2- Pipelined deliverable 1. [***C code and report - Due date: November 18th, 2015***]

3- A C implement of direct-mapped cache structure between the main memory and the pipelined CPU in deliverable 2. [***C code and final report - Due date: November 25th, 2015***]

For deliverables 1 and 2, summarize your design and testing results. Present full design and your conclusions in a 3 pages final report.

Implementation Hints (You do not have to use them)
1- All data will be integer numbers (no floating-point)
2- Memory is just a large array. You may want to partition the memory in two, instruction memory and data memory. You may even want to have two types of memory (instruction of type string and data of type integer).
3- Instructions in memory will be of the form `add $t1, $zero, $s0` (so assembly form and not binary as seen in actual processor).
4- So a fetch operation will return an instruction like this `add $t1, $zero, $s0` that you will need to decode/parse to get the field and operation needed.
5- The register file is also an array of the form `int RegiterFile [32] = {0, 0, ...,0}`

```
// operantA = RegiterFile [0]
```

```
// operantA = RegiterFile [16]
```

6- // You also need to think about overflow and zero

```
int ALU (int operantA, operandB, int Operation){
      // if (Operation == add) result = operantA + operandB;
      return result;
}
```

7- Memory operations

```
// int memory_load (int address)
// int memory_store (int address, int data)
```

8- You should have a way to print out the answer from the execution. Be aware of jump, branch instructions and start program counter.