Advanced Computer Architecures notes

Elia Ravella

March 4, 2021

Contents

I Introduction **2**

1 Taxonomy of Computer Architectures **2**
1.1 Flynn Taxonomy 2

II Performance and Cost **2**

2 Performance **2**
2.1 Evaluating performance 2
2.2 Quantifying the Design Process 3
2.3 Indexes on performance 3
2.4 Benchmarking 3
2.5 Energy, Power 4

III Pipelining **4**

3 MIPS architecture **4**
3.1 Assembly basics 4
3.2 MIPS instruction execution 5
3.3 MIPS Chip Architecture 5

4 Data and Control **6**

Part I
Introduction

1 Taxonomy of Computer Architectures

1.1 Flynn Taxonomy

Flynn defined four categories to group computing architectures. This so called taxonomy helps system architects to distinguish among possible models of processors or system themselves.

1. SISD: Single Instruction Single Data, uniprocessor systems;
2. MISD: Multiple Instructions Single Data, no commercial application;
3. SIMD: Single Instruction Multiple Data, low overhead, used in custom integrated circuits;
4. MIMD: Multiple Instruction Multiple Data, off the shelf micros

SISD is the simplest and most diffuse model of computation, one data stream in input and a *single* instruction executed per istant of time. SIMD introduces parallelism in a vector fashion: multiple processes execute a single instruction (the *same* instruction) on different data elements. Another type of parallelism: MIMD. Just SIMD with multiple different instructions per instant.

Part II
Performance and Cost

2 Performance

2.1 Evaluating performance

We need to distinguish between how to measure performance of a computer architecture and what impacts it without affecting performance. The amount of memory used is not a performance evaluator, but it affects performance in some way. The power consumed, the latency and the execution time are. To evaluate the performance of a CA I must use all the performance indicator and find the right tradeoff. Also, context! A CA is inserted in a context that bounds its scope, so the environment itself shapes the way performance is evaluated
Every performance indicator says something about the CA. So, the whole system must be evaluated with a *whole set* of indexes that each tell a different thing about the system itself. This provides a full view of the system strenghts and weaknesses and highlights the things to optimize.

2.2 Quantifying the Design Process

"The frequent case" case: when designing a CA, the "frequent case" represents the set of conditions met *the most times* during a standard computation run. Optimizing the standard case means optimizing *singnificately* all the system behaviour.

Amdahl's Law Amdahl's law describes the concept of the speedup as a function of execution time. The set of formulas that describes the law is

$$EXTIME_{new} = EXTIME_{old} \times [(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}] \tag{1}$$

$$SPEEDUP_{overall} = \frac{EXTIME_{old}}{EXTIME_{new}} = (1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}})^{-1} \tag{2}$$

and calculates the speedup of a system taking into account all his part, the speedupped one and the invariate one.

2.3 Indexes on performance

Overview of performance indicators and measurements.

- Instruction Count: the effective number of instructions executed. This is influenced also by the program we're trying to execute.
- Tyme per Cycle: the clock rate. OBVIOUSLY influenced by the hardware architecture.
- MIPS and MFLOPS indicates already a very *program bound*-like perfomance indicator. They represents the number of integer/floating point operations that the architecture can carry out in a second. They're very synthetic indexes.
- CPI - Average CPI (clock per instruction) represents obviously yhe amount of clock cicles that are necessary to carry out a specific operation. The element that most impacts the CPI is the ISA: the alphabet we use shapes the words we can create.

Notice how execution time is not present. That's because *reducing execution time is the goal*, so execution time does not modify performance, it just *represents* it.

2.4 Benchmarking

Benchmarking is "running a fake program with the sole purpose to evaluate a system and find bottlenecks". It's a technical test. Workloading a CA means feeding it with a real work load to test what is a response to a non-syntethic stimulus.
Benchmarking must be

1. representative: we need to benchmark the right aspects of the architecture we're testing.

2. updated: old benchmark does not test the architecture well, because of the very architecture is changed and the benchmark is *not anymore representative* because it's *old*.

2.5 Energy, Power

Energy and power consumption are a "cost - like" performance indicator. The less the power, the less the cost of keep the architecture running, and tha's easy. Also, reducing the power consumed means *reducing the power needed from a battery* and this impacts a lot in the mobile scenario. Energy per task is a good indicator for energy consumption. The power used by an architecture is influenced by a ton of factors, not least the TDP (thermal design power). Also architecture-bound aspects (clock rate, voltages, busses) influence power consumption *heavily*.

Part III
Pipelining

3 MIPS architecture

Reference architecture for this course is MIPS.
MIPS is a RISC architecture (Reduced Instruction Set Computer) this means that the only possible operation (that are very optimized) are basic operations. This kind of architecture is the opposite of CISC (Complex ISC) where more complex operation are supported at chip level.
MIPS is a LOAD/STORE architecture. This means that data cannot come into the ALU from anywhere besides the CPU general purpose registers. MIPS uses 32-bit instructions that can be

- Register Register operations
- Register Immediate operations
- Branch operations
- Jump/Call operations

Operand lenght is 6-bit. We have 64 possible operations.

3.1 Assembly basics

We'll see three classes of MIPS instructions:

- ALU: add "reg target" "reg source" "reg source", or sub "reg target" "reg source" "immediate"
- LOAD/STORE: lw "reg target" "offset register" (load word)
- Branches/Jumps: classic bne or bge or jmp operations

3.2 MIPS instruction execution

A fetch - decode - execute cycle in this architecture works this way:

1. Fetching the instruction means sending the PC content to the memory in order to access the memory location where the instruction is. Then, the PC is updated (PC + 4, each instruction take 32 bits).

2. Decoding the instruction: in this phase a fixed-field operation is put in place to deconde the instruction itself, and the registers needed for the computation are read.

3. Execute operation: the ALU calculates the result of the instruction supplied.

4. Memory Access and Write Back phases: in these two phases the CPU takes care of the memory access (LOAD or STORE instructions) and the registers update.

3.3 MIPS Chip Architecture

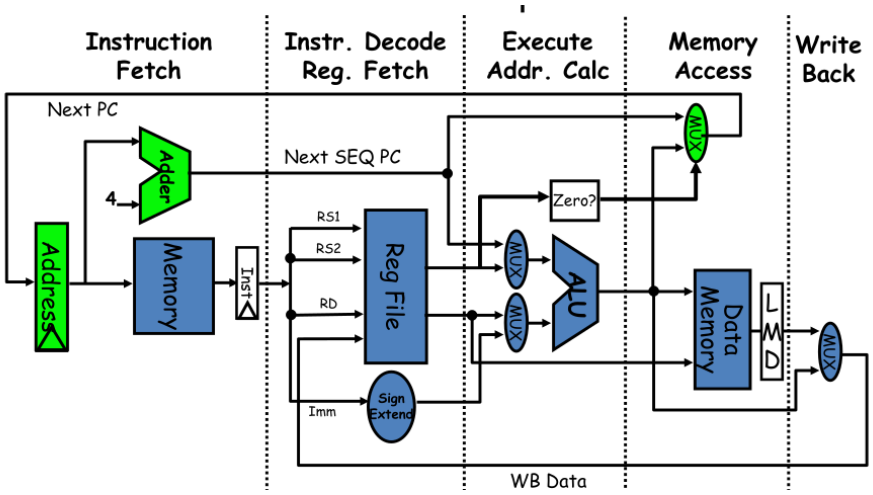


Figure 1: The MIPS data path with pipeline stages highlighted

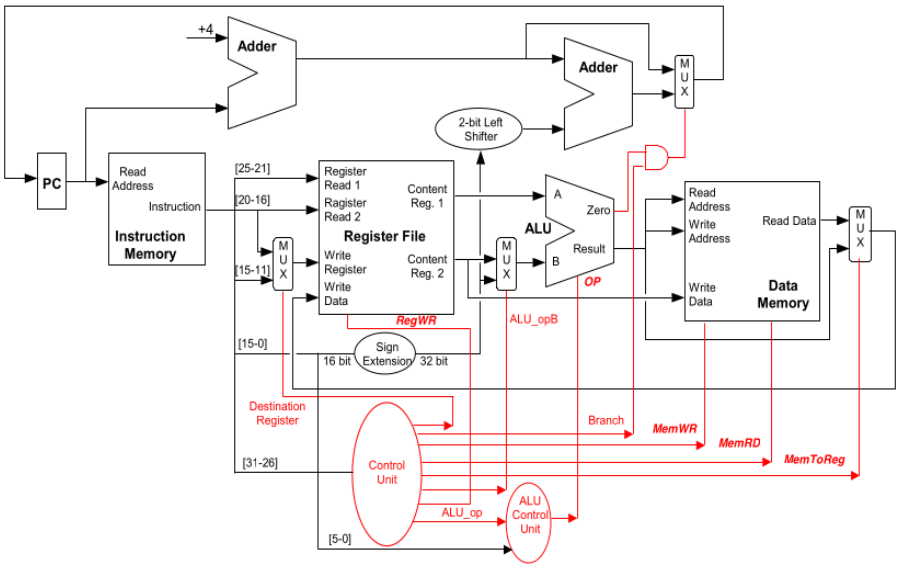


Figure 2: The MIPS full CPU, data path integrated with control unit

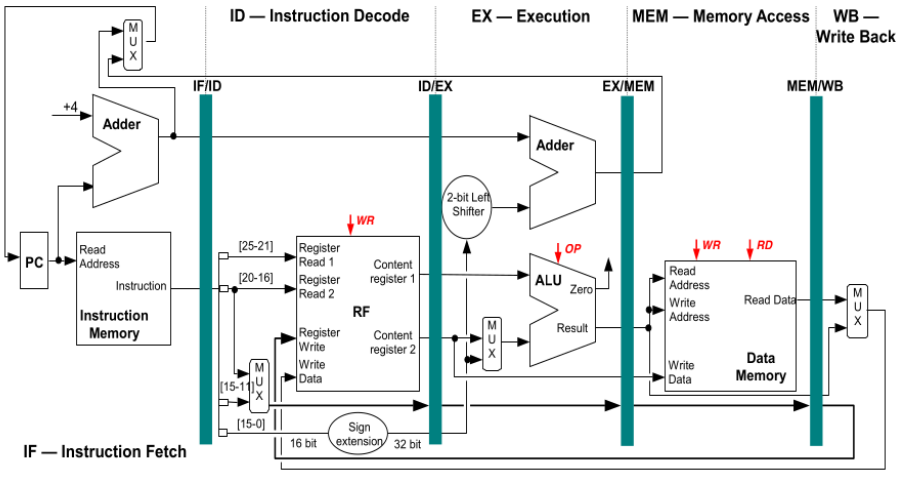


Figure 3: Pipelined MIPS

4 Data and Control

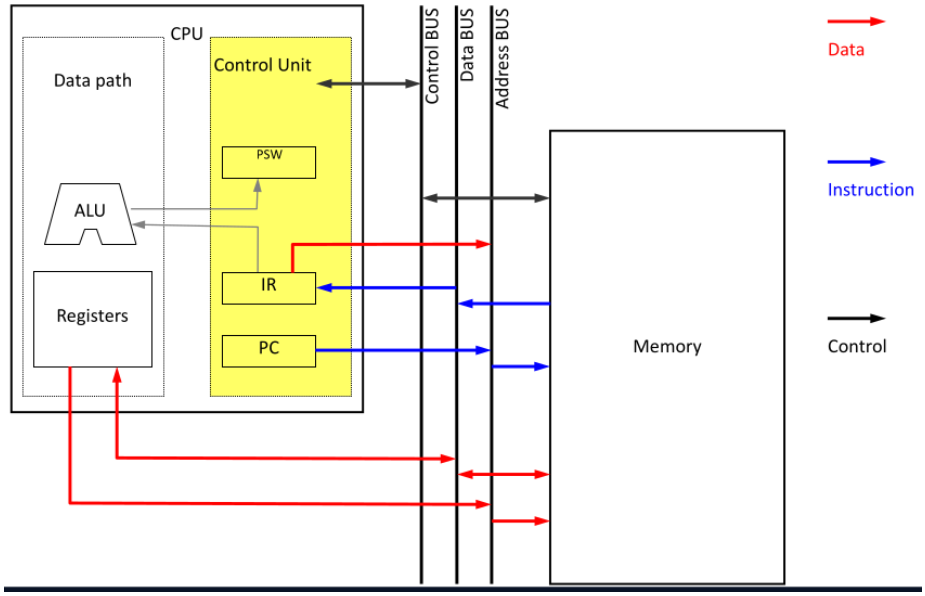
Every computer architecture is composed of 2 main parts: the one that computes data (datapath) and the one that controls the execution (control unit).

Datapath The datapath runs the house. All data pass through this module, and the registers here are used to pipeline operation.

Control Unit The CU guides the computations. It takes "run time decisions" in order to pilot execution of intructions. The registers here are architecture-

specific.

Memory And memory? "Slow primary memory" is connected with the CPU (so DP + CU) by means of buses. The structure is similar to:



Things to be noted:

1. the datapath does not connect to the control path
2. the control unit does not write on the data bus