THE SINGLE CYCLE PROCESSOR

OREGON STATE UNIVERSITY

CS 271 Winter 2020

Abhijith Balijepalli & Maxwell Franz

Table of Contents

Introduction

A single cycle processor can carry our one instruction in a single clock cycle. This document describes and analyses the components of pipelining and the purpose of a single cycle processor. This document does not cover system architecture design, digital logic. The document is arranged as follows: First, background information on the pipeline and processors will be presented, followed by an analysis of different components of pipelining: Instruction formats, Design logic, data-paths, CPU's parts, Pipelined vs single cycle execution., MIPS pipeline, Hazards, and solutions. Followed by a conclusion that discusses the timeline of innovation of the single-cycle processor. Finally, solutions to the exercises.

Background

Pipelining was introduced to increase productivity and efficiency. Originally, CPUs were not pipelined, they performed a single cycle at a time, and were slow. MIPS Pipelining counteracted this by increasing the efficiency of every cycle. MIPS stands for Microprocessor without Interlocked Pipeline Stages; this was the original product offered by MIPS Computer Systems. This is also how the original CPUs ran.

When pipelining was introduced by MIPS Computer Systems, they rebranded to MIPS Technology and reintroduced the MIPS Pipeline. The single-cycle processor was the predecessor of the MIPS Pipeline.

While not immediately related, the MIPS Pipeline quickly overtook the single-cycle processor in terms of popularity and speed. The single-cycle processor is a processor that performs one instruction in a single clock cycle. This is both similar to and different from MIPS Pipelining as the number of clock cycles an instruction may take can vary within the MIPS Pipeline.

Analysis

1. Instruction Formats

There are 3 subsets of instruction formats for different purposes. Memory reference instructions denoted as lw and sw. A memory reference is an instruction that has one or more of its operands that refers to a location in memory [1]. The instruction lw is used to load a word from memory into a register and the sw instruction stores data into a specified address [2,3].

The next instruction set is the Arithmetic-logical reference instructions command the arithmetic logic unit to do mathematical operations [4]. The mathematical operations between two operands are addition (add), subtraction (sub), comparing (and, or), and set if less than between two operands (slt) [5].

The last instruction format is a control transfer instruction; beq and j [6]. Control transfer instruction controls the flow of the program and directs that flow [6].

2. Design logic, data-paths

Information at the foundation of the computer is encoded in binary. A zero signifies low voltage and 1 for high voltage, and there is one wire per bit [7]. The combinational elements operate on output as a function of input; for example, AND-gates, adder, multiplexer and the ALU [7].

Another type of design logic is sequential elements that store elements inside a register. A clock signal is used to determine the status of the stored value inside the register [8]. Data paths are sets of units that carry out the data processing functions, they are a crucial part of the control unit and the Central processing unit (Figure 1) [9].

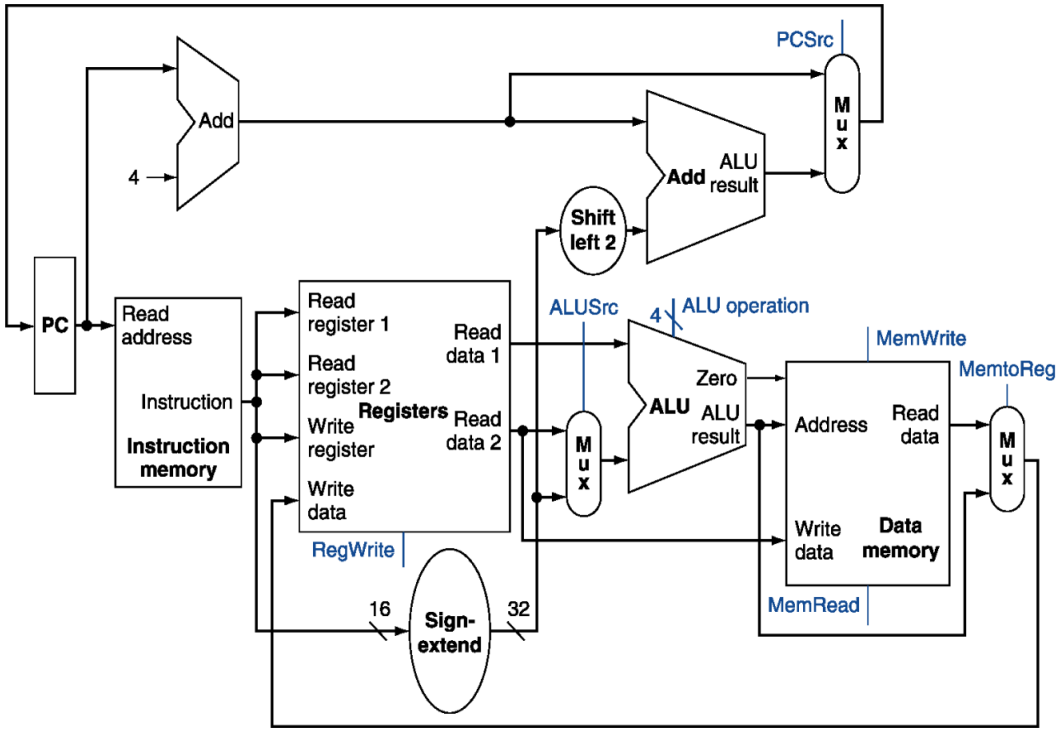

Figure 1: Datapath diagram with 4 instructions [8]

3. CPU's Parts

The control processing unit is the brain of the computer, it's where most of the calculations take place [8]. There are two main units inside CPU are the arithmetic logic unit (ALU) and the control unit (CU) [10]. The ALU performs arithmetic operations on incoming inputs and decides where it needs to be sent, it also communicates with the memory units [10]. The control unit analyses instructions from memory and executes the instruction, the ALU can work for the control unit when necessary [10].

There are three main memory units: ROM, RAM, and Cache [10]. RAM and ROM fall under primary memory, but random-access memory (RAM) is volatile and it's the most common type of memory found in computer systems. A read-only memory (ROM) which means it's non-volatile, this memory has been preinstalled on the computer [11]. Lastly Cache memory is a high-speed storage system that is a section of main memory, it reduces time to access data from the main memory [11].

4. Pipelined vs Single Cycle Execution

In terms of speed and performance, pipelined execution is more efficient. All instructions in a single cycle structure don't always take an equal amount of time for instruction but a new instruction cannot start until the next clock cycle [11].

Which can be inefficient compared to the pipelined structure processor [12]. The pipelined structure a lot of time can be saved by overlapping instructions even if the length of the instruction will increase (Figure 2). A similarity of each instruction is their latency, which can be the same but not the execution time or cycles consumed [12].

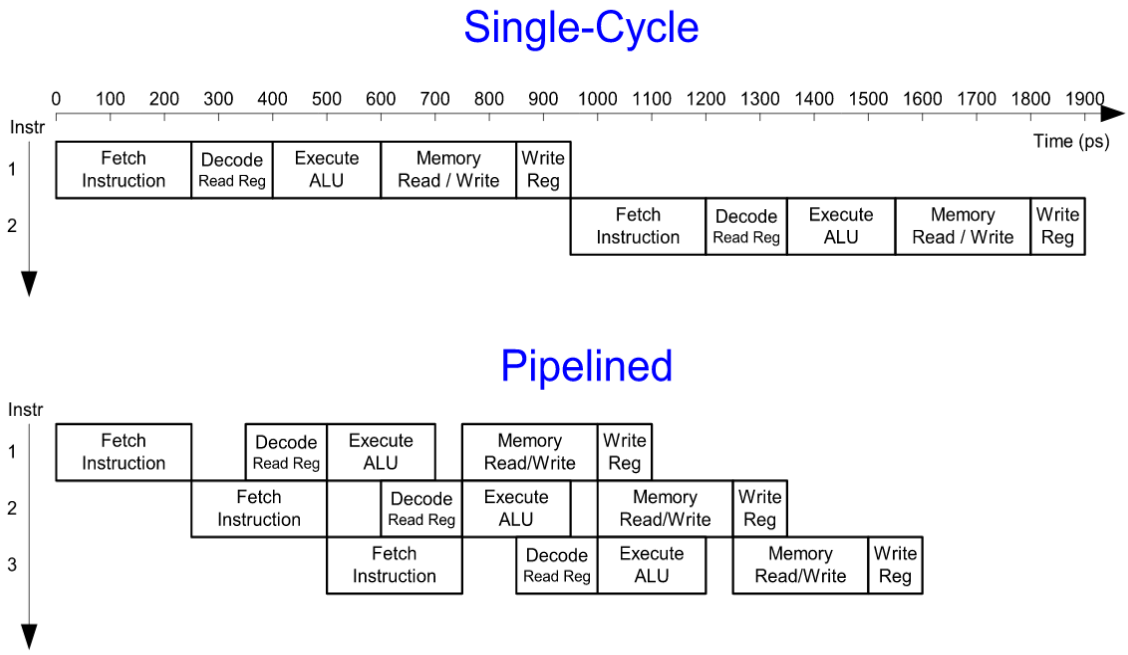

Figure 2: Comparison between Single-Cycle and Pipelined instruction

5. MIPS Pipeline

Pipelining is a process that allows the CPU to perform more than one instruction at a time. The MIPS Pipeline utilizes this method of efficient computing in order to accomplish very fast computing times. But first, we must introduce the stages of the MIPS Pipeline. It's broken down into five stages: IF, ID, EX, MEM, and WB [14]. Each stage and its execution time are integral in the efficiency and speed of a MIPS CPU.

The first stage, IF, or Instruction Fetch, does just that, it fetches the next instruction to be executed by the CPU [13, 15]. In doing so, it increases the Program Counter by 4, this is done because all instructions are 4 bytes long in size [16]. The Program Counter holds the address of the next instruction, so it's very important that this is tracked correctly.
The second stage in the process is ID, or Instruction Decode, and this stage decodes the instruction and figures out what to do with said instruction [13, 15]. The third stage is EX, or

Execution, and this is where the computation occurs [13, 15]. A CPU uses the built in Arithmetic Logic Unit, or ALU, to perform arithmetic and logic operations and is the most essential component in the CPU.

Using the ALU, the instruction is executed, and the result will just sit there until the next stage in the branch. The fourth stage is MEM, or Memory Access, and it will either store or read the result of the Execution stage and will access whatever memory addresses are necessary for doing so [13, 15].

The fifth stage is WB, or Write Back, and this stage will take the address of wherever the result was stored from the MEM stage, access it, and store the result in the appropriate register for later use [13, 15]. These are just the stages of MIPS Architecture, now on to how pipelining is implemented.

The process is easy to understand and makes for very efficient computation when in effect. The pipeline works as follows, when the first Instruction Fetch is called, nothing fancy occurs unfortunately. When the next stage of that branch occurs, another Instruction Fetch is called. Then when the Execution stage occurs in the first branch, the Instruction Decode is occurring in the second branch, and the Instruction Fetch occurs for the third branch. This will continue to happen until the program has completed execution and all results are written back for each branch. This makes for much more time efficient computation versus a non-pipelined system (Figure 3).

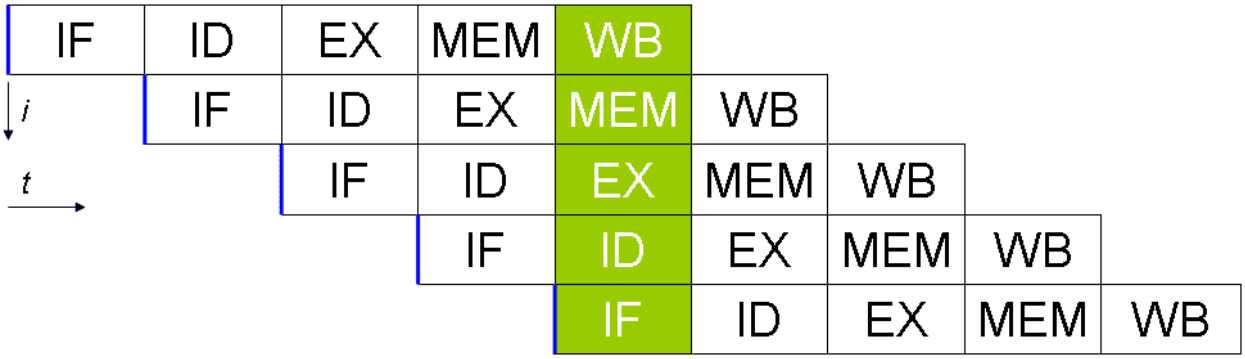

Figure 3: A pipeline with 5 branches working at the same time, offset by one stage each.

6. Hazards and Solutions

These extremely efficient computations don't come for free though, as hazards within the CPU may cause errors and results that are incorrect. There are three different types of hazards that one may run into when pipelining.

These hazards are as follows: Structure Hazards, Data Hazards, and Control Hazards. Structural hazards occur when two separate instructions are attempting to use the same resources for two different instructions [16, 17].

When implementing a pipeline, the overlapping of branches during execution requires that multiples of or duplicates of resources are available to allow all possible combinations of instructions in said pipeline. The solution for this is rather simple, in that you must ensure there are adequate resources for every combination of instructions to execute.

Data hazards occur because of the overlapping of multiple different branches on top of each other [16, 17]. The cause for data hazards is when an instruction down the line requires the result of an instruction that hasn't been completed yet [16, 17]. The first solution is bypassing, also known as operand forwarding, which utilizes a pair of bypass multiplexers to ensure that an instruction can't move forward without all integral pieces of information [16, 17]. Each multiplexer is set to select between three different options when it is in use.

The three choices of each multiplexer are the output of the decode stage, the current register pipeline of the ALU, or the current register pipeline of the access stage [16, 17]. Each selection can be utilized in separate circumstances and are there to ensure success of the pipelining. The second solution is pipeline interlocking, also known as stalling, which will stop one branch from continuing forward until it detects that the appropriate data is available for use [16, 17].

Control hazards occur when the Program Counter is off [16, 17]. At the start of each branch, the Program Counter will increase by 4 because every instruction is 4 bytes in size. If the Program Counter doesn't increase by 4, you will get a control hazard. This will cause the execution of instructions to be off. Solutions for these types of hazards are not always guaranteed to work, however they are as follows. The first solution is stalling, as mentioned above, can also be utilized to prevent control hazards from occurring [16].

By slowing down the execution time of certain instructions, control hazards can be avoided. The second solution is freezing or flushing the pipeline [16]. In this process, any instructions after the branch currently being executed will be held or deleted and will only be reinstated when the correct destination address for the results is known [16].

The third solution doesn't really have a name; however, the process is to allow the program to continue as if the current branch hasn't been executed. This means that the program won't have an accurate Program Counter, however that won't necessarily matter because each branch can still run without it [16].

This solution can be quite complex to implement though, because without the Program Counter there is no real tracking of what's happening. If the state of the processor were to change while running during this time, other hazards can arise [16].

7. Multi-Cycle vs Single-Cycle

Calculation speed is one of, if not the top factor when comparing CPUs and efficiency. In a single cycle system, all instructions wait for the full clock cycle and the cycle time is based on the slowest instruction [18]. This means that the total efficiency of a single cycle CPU is based on the slowest instruction.

Speed is one of the most important factors when it comes to computing, so relying on the slowest instruction simply won't do. Therefore, the multi cycle system exists, a multi cycle CPU will break the Datapath into sub-operations with the cycle time set by the longest sub-operation. Now instructions only take the number of clock cycles they need to perform their sub-operations [18]. This will result in a faster clock time because each instruction will take only the necessary number of cycles to complete instead of a determined number of cycles based off the slowest instruction.

Conclusion

In conclusion, the single-cycle processor was designed for pure efficiency and speed. It was designed to give engineers and software developers a quicker way to test their projects and code. It began to be used in the 1970s era supercomputers. During this time, pipelining was the most efficient form of instruction execution within CPUs, so it was the only option for supercomputers.

One of the first supercomputers ever built by Control Data Operation utilized pipelining as a testing ground to gauge its efficiency. Due to its low-cost method of instruction execution, it slowly began to take over. By the mid-1980s, pipelining within CPUs was being used by companies and software engineers all over the world.

Exercises

Problem 1:

1.1) 400ps + 100ps + 30ps + 120ps + 200ps + 350ps + 100ps = 1300ps without MUL
400ps + 100ps + 30ps + 120ps + 200ps + 350ps + 100ps + 300ps = 1600ps with MUL

1.2) $\frac{(New\ Clock\ Cycle \times 100)}{(Old\ Clock\ Cycle \times (100-5)} = \frac{(1600 * 100)}{(1300 * 95)} = 1.2955$

Problem 2:

2.1) lw and sw are using the data memory which would be 25% + 10% = 35%
2.2) add doesn't need a sing-extended circuit, so it will be 20+0+25+25+10 = 85%

Problem 3:

3.1) Original: 1010 1100 0110 0010 0000 0000 0001 0100
Sign extend: 0000 0000 0000 0000 0000 0000 0001 0100
Shift left 2: 0001 1000 1000 0000 0000 0101 0000

3.2) ALU = 00

3.3) From PC to Add (PC + 4) and then add 4 to the PC value and then goes to MUX, and then it jumps to PC.

3.4) Write registers are not being used, the value is stored in memory.

3.5) MUX will be written as a 2 or 0. Branch MUX and Jump MUX = pc +4. ALU = 20, since the binary value for 0001 0100 = 20.

3.6) Read Register 1 = 3 (0110), Read Register 2 = 2 (0010) , Write Data = X, Write register = (0 or 2), RegWrite = 0

Problem 4:

4.1) The OR in the first line write to r1, and then you can see that the next OR are dependent on r1 (RAW). You can assume that there is a write function there. The next line of OR writes to r2 and since r1 is already being written you can see r4 has a read after write (RAW).

4.2) By adding a NOP the hazards can be eliminated between the first two sets of instructions. The reason for this is because there is a RAW dependency from the first to the second, or to the third.

4.3) There are ALU instructions in this case which avoids our conclusion on having hazards.

4.4) since there are 3 sequences for the first instruction that will be in 5 different cycles. The last two will require only 2 because of pipeline architecture.
Without forwarding = 250 * 7 = 1750 With forwarding = 290 * (7 + 1 (NOP)) = 2320

4.5) Or r1, r2, r3
Or r2, r1, r4
NOP
NOP
Or r1, r1, r2

4.6) With ALU-ALU forwarding the instructions will not need a NOP so it will be 7 cycles.
290 * 7 = 2030, Speedup value = 2320/2030 = 1.14

Problem 5:

5.1) 11 clock cycles to come out of the EX stage to finish that instruction, while you would only need (11 - 1) for ID. So, the speed Up value = (11/10) = 1.1

5.2) Latency of ID = 120 + ((0.5) * 120) = 180ps
Latency in EX decreased by 10ps so = 150 - 10 = 140ps.
Speed up value will be = (11*200) / (10*200) = 1.1
So, there is no change in the speed up.

5.3) For EX: 150 - 20 = 130ps. If beq is used for memory address calculations and it's moved to the MEM stage, then it will change the execution time because it will add one more cycle. But the cycle cannot change the 20ps reduction, so the number of cycles with a branch in EX will still be 11. And the execution time will be 11 * 200 = 2200ps. After the new cycle is added due to beq then the cycles calculated in MEM will be 12. So, the execution time is MEM: 12 * 200 = 2400ps.

Problem 6:
6.1)

lw r1,0(r1)	WB
lw r1,0(r1)	EX, MEM, WB
beq r1, r0, loop	ID, Stall, EX, MEM, WB
lw r1,0(r1)	IF, Stall, ID, EX, MEM, WB
and r1, r1, r2	IF, ID, Stall, EX, MEM, WB
lw r1,0(r1)	IF, Stall, ID, EX, MEM, WB
lw r1,0(r1)	IF, ID, Stall
beq r1, r0, loop	IF, Stall

6.2) It would be 0 because there is never a point in all the 5 stages doing useful work when there 8 clock cycles in each iteration of the loop.

Work Cited

1. “memory reference instruction,” A Dictionary of Computing, 14-Feb-2020. [Online]. Available:https://www.encyclopedia.com/computing/dictionaries-thesauruses-pictures-and-press-releases/memory-reference-instruction. [Accessed: 02-Mar-2020].
2. N. Rehmat, “LW Mips: Load Word Opcode Example,” AssemblyLT, 12-Mar-2017. [Online]. Available: https://www.assemblylanguagetuts.com/lw-mips-load-word-opcode/. [Accessed: 02-Mar-2020].
3. “SW,” MIPS 101. [Online]. Available: https://www3.ntu.edu.sg/home/smitha/FYP_Gerald/swInstruction.html. [Accessed: 02-Mar-2020].
4. R. Fosner, P. Barry, and P. Crowley, “Embedded Processor Architecture,” Arithmetic Instruction - an overview | ScienceDirect Topics, 2012. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/arithmetic-instruction. [Accessed: 02-Mar-2020].
5. N. Rehmat, “SLT MIPS: Set If Less Than Instruction,” AssemblyLT, 18-Mar-2017. [Online]. Available: https://www.assemblylanguagetuts.com/slt-mips-set-if-less-than-instruction/. [Accessed: 02-Mar-2020].
6. “Oracle Homage,” *Control Transfer Instructions (x86 Assembly Language Reference Manual)*, 2010. [Online]. Available: https://docs.oracle.com/cd/E19120-01/open.solaris/817-5477/eoizl/index.html. [Accessed: 03-Mar-2020].
7. R. M, “Assembly Language Programming,” *Lesson 6: Binary Logic*. [Online]. Available: https://www.randomterrain.com/atari-2600-memories-tutorial-robert-m-06.html. [Accessed: 03-Mar-2020].
8. L. Chen, “Computer Architecture,” in *Computer Architecture*, 01-Mar-2020.
9. “What is a Data Path?,” *Computer Hope*, 26-Apr-2017. [Online]. Available: https://www.computerhope.com/jargon/d/datapath.htm. [Accessed: 03-Mar-2020].
10. V. Beal, “What is a CPU - Central Processing Unit? Webopedia Definition,” *What is a CPU - Central Processing Unit? Webopedia Definition*. [Online]. Available: https://www.webopedia.com/TERM/C/CPU.html. [Accessed: 03-Mar-2020].
11. V. Beal, “Computer Architecture Study Guide,” *Computer Architecture Reference - Webopedia Study Guide*. [Online]. Available: https://www.webopedia.com/quick_ref/computer-architecture-study-guide.html. [Accessed: 03-Mar-2020].
12. A. R. Lebeck, “IEEE Duke University,” *Additional Notes*, 2017. [Online]. Available: https://www2.cs.duke.edu/courses/fall98/cps104/lectures/week14-l2/sld034.htm. [Accessed: 07-Mar-2020].
13. [Online]. Available: https://www.brunel.ac.uk/~eestppk/EE5531/06MIPSPipeline.pdf. [Accessed: 03-Mar-2020].

14. [Online]. Available: https://eecs.oregonstate.edu/research/vlsi/teaching/ECE472_FA12/chapter4_pipelining_END_FA11.pdf. [Accessed: 03-Mar-2020].
15. [Online]. Available: https://courses.cs.washington.edu/courses/cse410/05sp/lectures/cse410-10-pipelining-a.pdf. [Accessed: 03-Mar-2020].
16. J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Cambridge, MA: Elsevier, 2019.
17. *Pipeline Hazards*. [Online]. Available: http://web.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/hazards.html. [Accessed: 03-Mar-2020].
18. [Online]. Available: https://ee.usc.edu/~redekopp/ee357/slides/EE357Unit16_Multi_Cycle_CPU_Notes.pdf. [Accessed: 09-Mar-2020].