Part 1: Introduction to MIPS Assembly

Before we can do security research on MIPS devices, we need to learn the basics of the MIPS Instruction Set Architecture. Note that this will not cover more advanced topics such as the cache, exception handling, and floating point or SIMD instructions. This series assumes the reader knows nothing about MIPS-32 assembly and is for those who need a quick crash course to start learning it.

If you are already familiar with the concept of assembly, data representation (bits, bytes, words, hex), endianness, memory, and pointers: skip directly to learning about the MIPS registers.

First, what is assembly?

Imagine you are in a room by a window and the only thing inside is a light bulb with a light switch.

You have a friend that's outside of the room and the only way you can communicate is by turning the light switch on or off. You start simple at first by intending to say "hi" with one flick of the switch. You realize you can represent more words if you continue to turn the light switch on and off in patterns.

Similarly, a computer is designed around patterns represented by pre-defined electrical signals of on or off. The signal being turned off is represented with the digit 0 and the signal being on is represented by the digit 1. Via a combination of 1's and 0's we can put them together to form machine code instructions.

A machine code instruction is simply a pattern of 1's and 0's that the computer is designed around in hardware to perform some work.

This can look like **0011 1100 0000 1001 0000 0000 0110 0100**

However, these numbers are not easily recognizable by people and thus each machine code instruction has a corresponding name.

The 1's and 0's previously mentioned above represent the MIPS instruction: **lui $t9, 100**

These assembly instructions are what we are going to learn and is what the computer executes as work.

When programmers write code, they typically write code in a higher level language that is even more human readable than assembly. But the computer doesn't directly understand higher level code and can't digest it.

So then there is a program that will convert or compile the higher level code into machine code which is now executable by the computer.

It's like how after you go grocery shopping, you can't immediately eat the ingredients raw. You have to prepare them by cutting up your food into bite size pieces, putting some oil on the pan, cooking the food, and then you can eat it. In the computer's case, the code has to go through pre-processing, compiling, assembling, and linking before its ready for the computer to execute.

What is MIPS?

Created by some brilliant people at Stanford, MIPS is a reduced instruction set computer (RISC) architecture commonly found in embedded devices such as routers. The acronym MIPS stands for Microprocessor without Interlocked Pipelined Stages. The word **pipeline** is important and we'll get into that in a moment.

But, why (male models) MIPS?

Your desktop computer is probably an x86 or complex instruction set computer (CISC) which features more flexible programmability but greater power usage. RISC is typically used in embedded devices due to its lower cost and power consumption. MIPS is used greatly for embedded devices and with the advent of the Internet of Things and MIPS going open source, I think the number of MIPS devices will only increase.

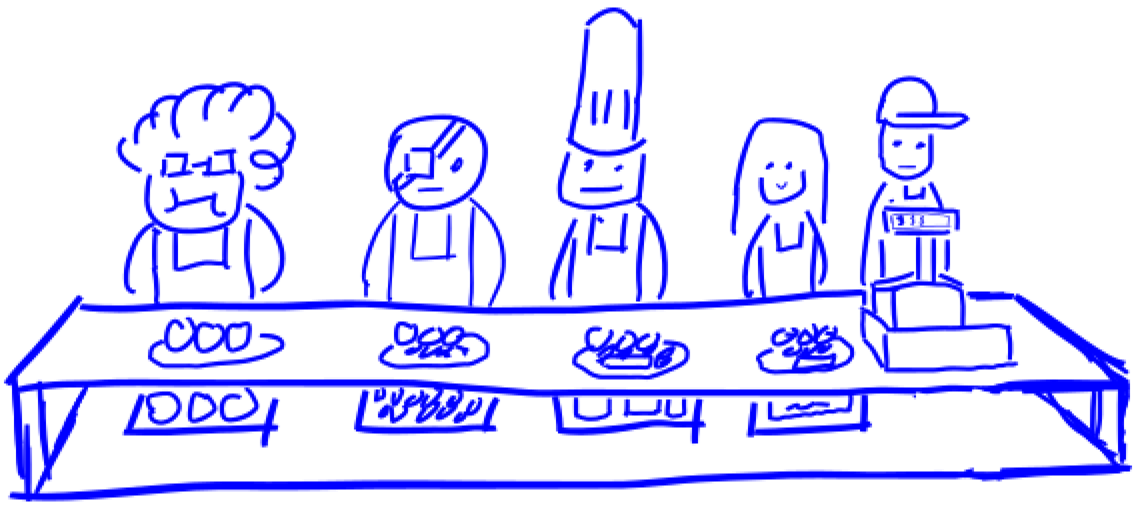
Since all MIPS instructions are the same size and the format is regular, it's easy to learn most of the MIPS instruction set!

What is pipelining?

MIPS is designed around the idea of pipelining.

It's a methodology for the CPU to process multiple instructions more efficiently at one time by separating the execution of an instruction into stages of a pipeline.

Paraphrasing the analogy from the book See MIPS Run, the pipeline is like a cafeteria lunch line. Instead of having one person serve you your mashed potatoes, corn, meat loaf, and gravy, the lunch line or pipeline is the idea where each of these servings is separated into a stage in the pipeline. So, one person serves potatoes, while the next person can serve corn at the same time for another customer's plate. In summary, serving multiple people at once at different stages in the lunch line.



When we translate the idea to MIPS, there are five stages in the pipeline:

1. Instruction Fetch
2. Instruction Decode
3. Execution
4. Memory Access
5. Write Back

Say we have three instructions to execute, they will propagate the pipeline and as soon as the first instruction completes the instruction fetch stage, in the next cycle, that instruction will move on to the instruction decode phase while the next instruction will begin its instruction fetch in the same CPU cycle and so on and so forth.

With this pipelining model, **three** instructions are completed in **seven** cycles.

Cycles	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Instruction 1	IF	ID	EX	ME	WB		
Instruction 2	IF	ID	EX	ME	WB		
Instruction 3	IF	ID	EX	ME	WB		

Without pipelining, **three** instructions are completed in **fifteen** cycles.

Cycles	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Instruction 1	IF	ID	EX	ME	WB										
Instruction 2	IF	ID	EX	ME	WB										
Instruction 3	IF	ID	EX	ME	WB										

But pipelining is not so simple. There can be hold-ups and slow-downs. For example, in our analogy, an order can get messed up or a customer may want to speak to the manager. For the CPU, there are several types of pipeline hazards (structural, data, control). Importantly, one of the main efforts MIPS makes to deal with control hazards that is unique to the architecture is the **branch delay slot**. This will be discussed further in Part 6: Jumps and Branches.

Further Reading

1. RISC vs CISC Table

2. Instruction pipelining [wikipedia]

Part 2: Computer Organization Crash Course