Review of CMPUT 229

This is a review document for CMPUT 229 at the University of Alberta for both Nelson Amaral and Ali Karim. This review doc was written when school was remote so your experience may vary. 😔

Index

Binary

Starting at the beginning with binary representation and 2's complement notation and moving on to hexadecimal notation. 2's complement can be understood by watching the following video. Hexadecimal can be thought of grouping the 4 digits of a binary number and combined/mapped to 1 character.

Floating point numbers

Floating point numbers are a way to represent decimal numbers. A major limitation of the accuracy which will also depend on the magnitude of the number itself. A floating point number is divided into 3 sections and is inspired by the scientific notation but adapted for binary. The first section is 1-bit and represents the sign of the nubmer (positive or negative). The second section (length varies on implementation) is the exponent. The third section (length varies on implementation) is the significand.

When talking about floating point numbers, we usually use the IEEE 754 standard. You use this tool to play around with floating point numbers and understand how they work.

Architecture

Since this is an architecture course, system architecture will be explored. Each instruction can be mapped directly to a 4 byte value also known as a word. Both MIPS and RISC-V execute word by word via the PC (program counter). Jumps/branches change the path/flow of the program. Think of this as a `goto` statement in C/C++. Using jumps/branches, conditional statements (such as **if**) and loops (such as **while loops**) may be constructed

Computations

There will be some computations with architecture such as calculating the execution time given CPU clock cycles and frequencies. Most of the calculations can be solved with the following formulas:

$$\text{CPU Time} = \frac{\#instructions}{program} \times \frac{\#clock\ cycles}{instruction} \times \frac{\#seconds}{clock\ cycle}$$

$$\text{CPU Time} = \frac{IC \times avg.\ CPI}{Frequency}$$

Frequency = $\frac{\#clocks}{time}$

CPI= $\frac{\#clocks}{\#instructions}$

Execution Time = I (instructions) x CPI (cycles per instruction is usually fixed) / f (frequency)

Also note the units as

time usually ns (nanosecond)

CPI => cycles/instruction

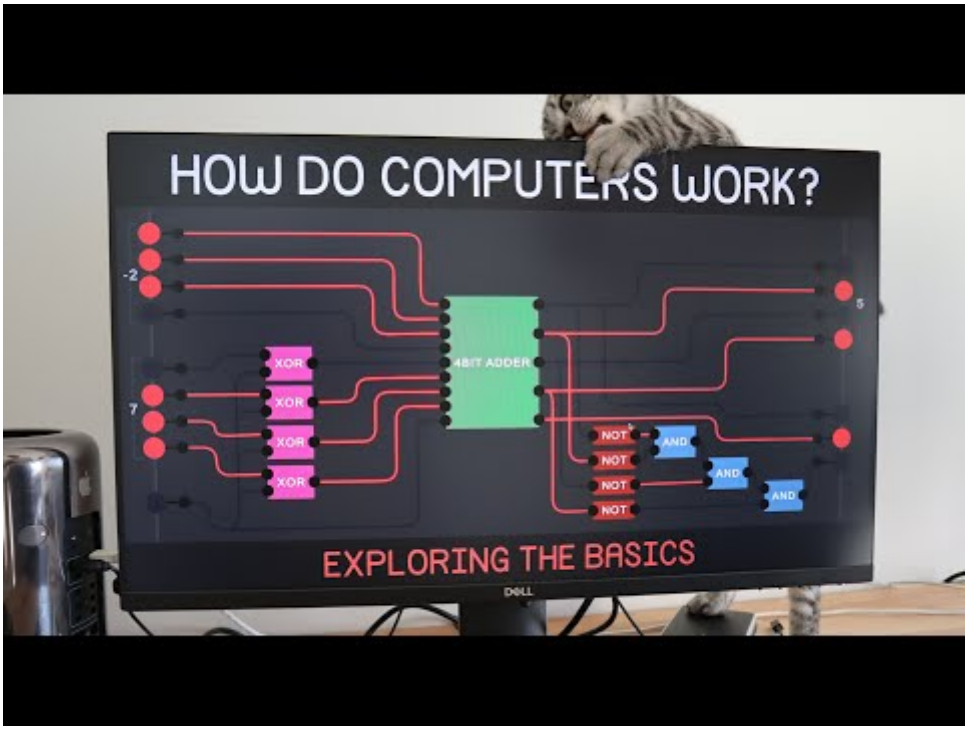
Frequency = Hz (1/time) [also could be cycles]

Stacks

Both MIPS and RISC-V have stacks that grow "downwards" as it increases in size.

Circuits

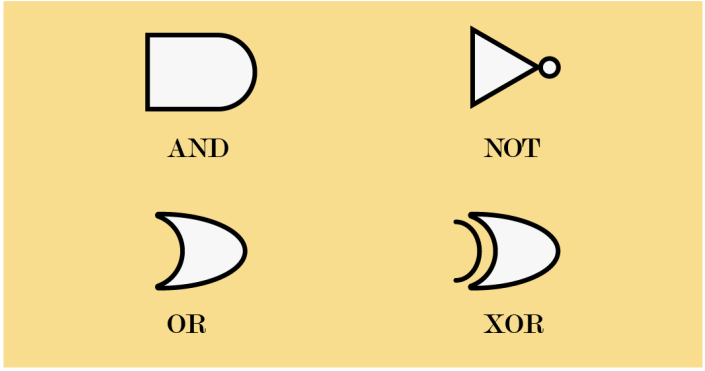
Circuits discussed in this course are not complex and do not approach the difficulty of the compE courses. The basic overview of this part of the course can be done by watching the following video by Sebastian Lague



Where the source code can be found here and the download here

I have made a completed save from what has been done on the video and packaged it up here on the release. Just follow the instructions and note that you still need to download the game from the itch.io page

Cheat sheet I made because the online ones are watermarked for some reason:



MIPS

This part will be a review about MIPS assembly taught by Ali Karim. This course will mostly focus on MIPS programing and is usually taught in the winter term.

Overview

MIPS is a assembly language that is used less and less, but is still used in some places. Not many systems support MIPS assembly out of the box so a emulator such as QtSpim is used to emulate a MIPS environment. Although MIPS is not RISC-V, they follow similar concepts such as branching, jumps, variables, etc. Most of the differences lie in the instruction set and instruction calls.

Basics

The next topic that does not fit nicely into the other topics is shift operators. Shift left operators are identical to multiplying by 2 to the power of the number shifted, eg:

```
sll $t0, $t0, k # where k is an integer
# is the same as $t0 <- $t0 * 2^k
```

This is exactly the same as

```
t0 = t0 << k // in C/C++
```

In actually, the number `$t0` is stored as a binary and k number of zeros are appended/added to the end of the number.

Likewise right right shifting is the same as floor dividing by 2 to the k-th power.

* Note that bits on the left or right side of the binary may be lost when shifting.

Assembly

Coding in assembly is required for this course. While some people despise assembly programming, after your first few takes of head bashing, it will come. The earlier you start programming in assembly the faster you will get it. I recommend coding something simple (ie Fibonacci calculator) at the start outside of the labs just so you can figure out how things work.

It is also important to go over some MIPS coding patterns so to better increase your coding efficiency and knowledge:

github.com/cmput229/MIPSPatterns

Also there are reference sheets that doesn't help too much when programming but are more for test taking:

studocu.com/en-us/document/california-state-university-long-beach/computer-architecture/lecture-notes/mips-green-sheet-mips-instruction-set-architecture

Sometimes you will need to go from C to MIPS so make sure you are somewhat familiar with C

Synchronization

When two CPU threads are communicating, synchronization is used.

Marking

Marks for this class include 6 quizzes (drop lowest one), 3 midterms, and a final project. The raw marks map directly to the grade you get from the regular marks table. Labs and assignments are also marked but do not count towards the final grade. These are highly recommended since they act as homework assignments, however even if you do not finish it by the hand in date, you can do them later.

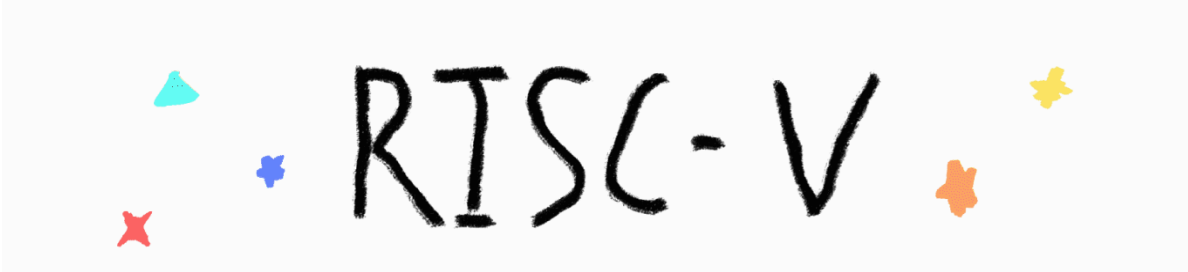
* The final for this course is notoriously difficult. Most people take in excess of 40 hours to complete it, so start this as soon as possible

Resources

Bank of questions contain lots of question that should be similar to the test although that's usually not the case :(

- MIPS coding patterns
- MIPS cheat sheet

RISC-V



This part will be a review about RISC-V assembly taught by Nelson Amaral. This course uses the RISC-V architecture with some basic extensions and is usually taught in the fall term.

Overview

RISC-V is an open standard ISA first developed in UC Berkley. Many companies such as Western Digital and SiFive are using RISC-V in their products. RARS is used to emulate a RISC-V environment and is used for labs.

Resources

- RISC-V cheat sheet

Authors

- Andrew Li - 229 (Karim) student
- Giancarlo Pernudi Segura - 229 (Nelson) student + 229 TA for Karim

License