**Projekttitel**: Dragon-4 | Ein auf Logikebene selbst entwickelter Prozessor

**Projektbeschreibung**: *Beschreibe kurz, worum es in Deinem Projekt geht und welche Vorgehensweise Du planst.*

Schon als kleines Kind habe ich mich immer gefragt, wie es möglich ist, dass ein so kleiner Microchip so viel kann. Als ich älter wurde fing ich an, mich damit auseinander zu setzten. Man findet viele Informationsvideos oder Datenblätter zu Prozessoren und ihrer Funktionsweise. Doch sie alle haben eins gemeinsam: Sie veranschaulichen nur oberflächlich, wie solch ein Wunderwerk der Technik funktioniert. Eine Antwort wie: „Die ALU (Arithmetic Logic Unit) berechnet irgendwelche Zahlen“ oder „Der RAM Speichert Daten“ reicht mir nicht. Ich will wissen, wie genau, auf kleinster Ebene, solch ein Prozessor funktioniert. Wie werden Daten gespeichert? Wie rechnet die ALU? Diese Fragen versuche ich zu beantworten, indem ich selbst einen Prozessor entwickle, um seine Funktionsweise bis ins kleinste Detail zu verstehen. Dazu verwende ich keinen fertigen Microcontroller, sondern stelle mir die benötigte Logik selbst zusammen, um das so tief wie mögliche Verständnis zu erlangen.

Ideen

In welcher Pipeline stage muss welche Control line hin. -> näher erläutern, denn bis zur ersten execution hat es ziemlich lange gedauert…

Überlegt, macht es Sinn content von RAM direkt in MAIN BUS? Aber nein, weil dann REG Controls in der ersten Stage sein müssten, das sind zu viele Lines für die erste

Pipeline: muss 1 clock cycle behind sein, weil sonst speicher an die falsche stelle pointet. -> input der Pipeline latchen

Reg controls in first and second stage, zwar mehr control lines, aber dafür kein noFetch notwendig -> speicher bus ist free to use, wir müssen keine „op skippen“ | DOCH NICHT, da wir keine neue Fetchen können, da das nächste Value im Speicher keineOP ist, sonder just a constant. Also würden wir wenn wir keine 2 Stages brauchen würden und also noFetch weg lassen, das Value als OP fetchen…. Deswegen brauchen wir auch keine direkte Verbindung zwischen RAM und Mainbus, weil wir eh 2 cylcen brauchen bis wir was neues fetchen können.

Alles was ich anders machen wollte, hat sich herausgestellt macht keinen Sinn

Pipeline zu Comlex. Würde eine Copie von James sein.bräuchte mehr Zeit um selber drauf zu kommen, mein Ziel zu lernen würde es dennoch erfüllen. Nur Jugend Forscht nicht. Weil ich unbedingt was eigenes auf die beine stellen möchte für Jugend Forscht, mache ich single Stage.

Anfangskonzept von Ben Eater, mit ROM als Control Unit und Fetch Unit

So, I ran out of OP-Codes when I did the ALU

Why not include cMOV instructions?

Immer gedacht, pipeline notwendig für cpu die quch wirklich was machen können, aber liegt eig. Nur an Architektur.

RAM logic war aua, hahb a lot of truth tables gedrawed, um das hin zu kriegen (some truth tables in die arbeit adden)

Last minute Change, all regs inv clocks

Talk about problem with CREGs in simulation

And massive routing problems, talk about airwires, pin’s waren nicht connected