Ethan Shanahan and Kieran Corson

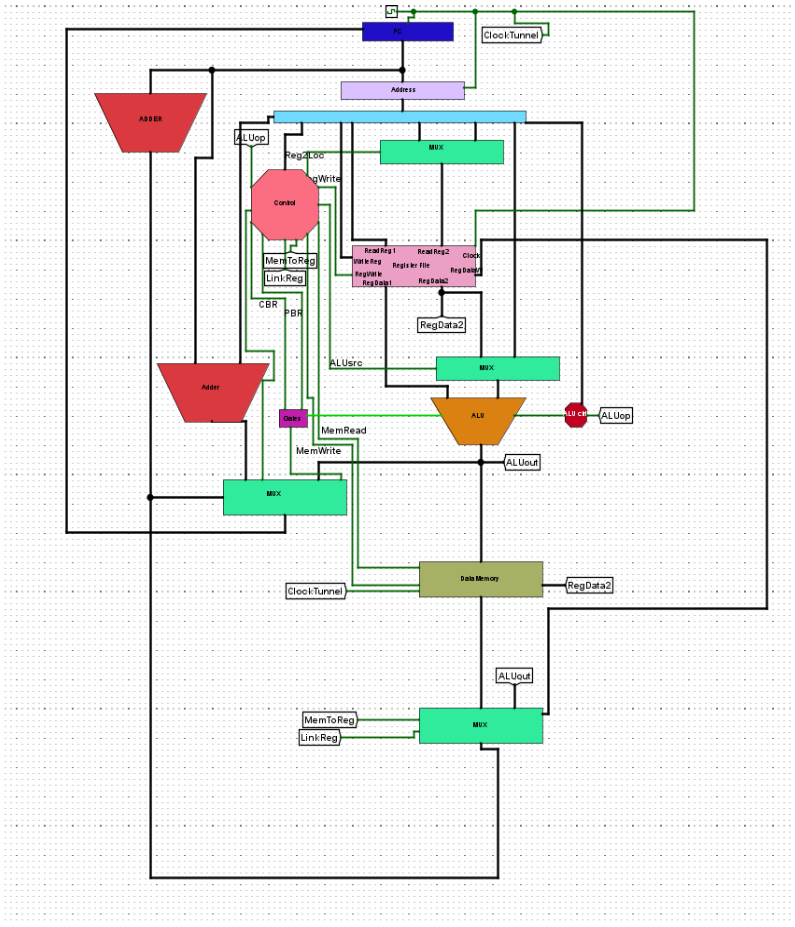
CS 382 – Computer Architecture

CPU Manual

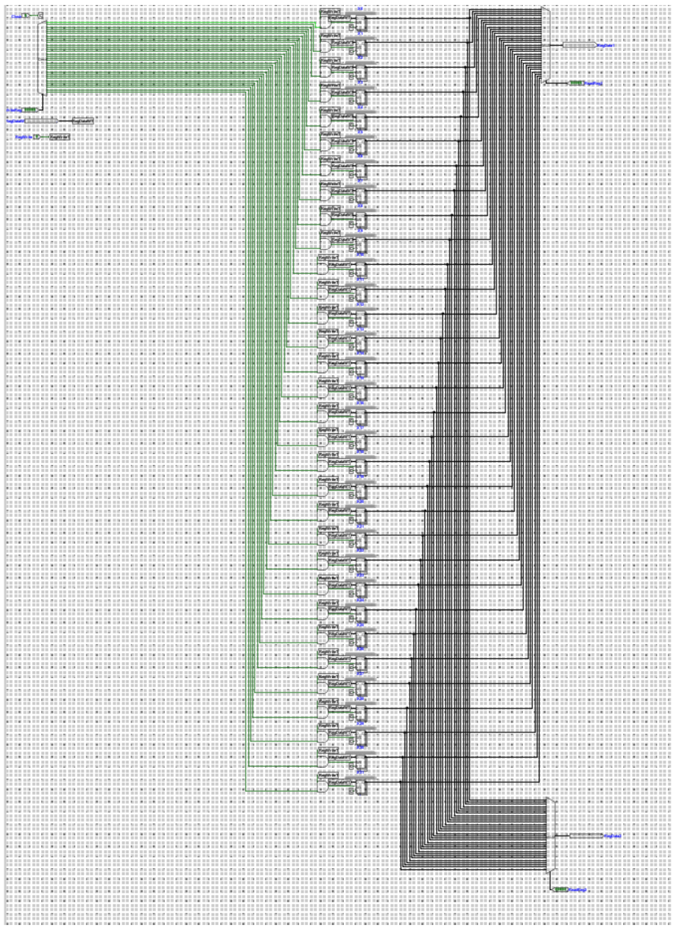
9 December 2023

Fried Liver CPU

CPU Visual



Register File



Assember Usage and Logism Simulation

The assembler is a c++ program that converts a text file into the proper format for logism-evolution to understand. There are no external libraries necessary, although the c++ file includes fstream, vector, iostream, and sstream. These libraries are used in order to read from the txt file that contains the assembly instructions, format the instructions for logism, and then write to the binary file for execution in logism.

The assembler script (called assembler.cpp) utilizes gcc for compilation. After compilation, the executable (assembler), runs on a Unix machine (although any machine with c++ compatibility should be able to execute the file). The assembler command goes as follows: ./assembler <file name>, where the <file name> is the name of the txt file that contains the assembly instructions. The resulting file, called bin, contains the encoded assembly instructions (in hex) based on the syntax of the Fried-Liver CPU with a header that specifies the format for logism to properly execute the instructions.

In order to simulate the resulting bin file containing the encoded instructions, you must open the circ file (the CPU) in logism-evolution. Then, open the instruction memory that is connected to both the PC and the instruction decoder. Right-click on the RAM available in the instruction memory and click load image. Find the bin file created from the assembler and load it into the instruction memory. The instruction memory should now contain the hexadecimal representation of the instructions that have been encoded into the bin file.

Architecture Design

As seen in the above image of the register file, this CPU contains 32 general purpose registers that contain sixteen bits of information each. The PC connects to the RAM (16M X 32) that stores the instructions in an array, loaded from the bin file created from the assembler. Each of the registers can be referenced using the prefix ‘x’ (i.e. x30 refers to register 30, and x1 refers to register 1). Because there are 32 registers, the registers span from x0 to x31.

This CPU contains four general instructions: add, mult, ldr, and str (addition, multiplication, load, and store respectively). The ALU separates the add and mult instructions based on the ALUop given by the ALUControl unit (after the opcode was separated by the Control Unit).

ASM Documentation

Instruction	Opcode	ALUop	ALU Operation
add rd, rn, rm	00000000101	0	ALUout = rn + rm
mult rd, rn, rm	00000000111	1	ALUout = rn * rm
add rd, rn, 11simm	00000001101	0	ALUout = rn + 11simm
mult rd, rn, 11simm	00000001111	1	ALUout = rn * 11simm
ldr rd, rn, 11simm	00000111100	0	ALUout = rn + 11simm
str rd, rn, 11simm	00001001000	0	ALUout = rn + 11simm
	Note: the opcode requires 11 bits for the 11 flags derived from the opcode in the control unit (i.e. Reg2Loc, ALUop, RegWrite, ALUsrc, MemRead, MemToReg, MemWrite, LinkReg, UBr, CBr, and PBr from lsb to msb)	Note: The ALU contains the capacity for two functions (add and mult) and thus only needs one bit.	

For Rm = x3 : Rn = x2 : Rd = x1, the binary encoding is as follows.

	Opcode	Rm	111000	Rn	Rd
add	00000000101	00011	111000	00010	00001
mult	00000000111	00011	111000	00010	00001

For Rm = 0b11111 : Rn = x2 : Rd = x1, the binary encoding is as follows.

	Opcode	11simm	Rn	Rd
add rd, rn, 11simm	00000000101	00000011111	00010	00001
mult rd, rn, 11simm	00000000111	00000011111	00010	00001
ldr rd, rn, 11simm	00000111100	00000011111	00010	00001
str rd, rn, 11simm	00001001000	00000011111	00010	00001

Each register requires 5 bits to represent since there are 32 registers in total (2^5 = 32).

Each instruction requires that the name of the operation (i.e. add, mult, ldr, str) is written in lowercase. The registers must also be written in lowercase, like x2 instead of X2. There is no comma between the operation and the destination register (rd), but there must be commas between the registers (see documentation above for examples). Each instruction maintains its own line. The assembler does not recognize comments or headers, therefore the txt file containing the instructions must be the instructions and only the instructions.

Contributions

Kieran Corson contributed to the creation of the circuit. The creation of the main logic of the circuit and much of the creation of creating the registers and such was done by Kieran Corson. This includes the instruction memory, data memory, alu, and, of course, the register file.

Ethan Shanahan contributed mainly to the logic behind the opcode, alu, and the assembler. This includes the compilation of the assembler.cpp file, the creation of the instructions file, and the creation of the bin file via the assembler.