Project 2

Anna Hauk
December 5, 2023

I pledge my honor that I have abided by the Stevens Honor System

CPU: AnnaBinAnna

This design implements a single clock sequential data path Harvard Architecture CPU.

There are four General Purpose Registers labeled R0, R1, R2 and R3. Each register is 8 bits wide. The General Purpose Register subsystem allows for output of two registers to the RegData1 and RegData2 outputs. RegData1 output has a path for storing to the RAM. When writing a value to a register, two bits WriteReg are used via a decoder to energize one register for which the RegWrite signal, when high will clock the value on RegDataW into the selected register. ReadReg1 and ReadReg2, both 2 bits are inputs to a mux to select one register each for output to the 8 bit RegData1 and RegData2 ports.

The RAM is 256 8 bit bytes of memory. The address bus of the RAM is 8 bits wide and the data path is also 8 bits wide.

The CPU has an ALU that takes two 8 bit inputs and is capable of performing unsigned addition or subtraction. The borrow in and carry in inputs are tied to zero. The Operation bit is 0 for addition and 1 for subtraction.

The CPU has a 256 address by 16 bit wide ROM. The address bus is 8 bits wide to address the 256 locations. The instruction set supports a load register immediate, load register from memory, store register to memory, an add instruction and a subtraction instruction.

The system has a program counter register system that contains an adder to increment the program counter address by 1 every time the circuit is clocked. The address is connected to the address bits, 8 of them, on the ROM. The 16 bits of output of the ROM is connected to a 16 bit Instruction Register that is used to hold the current instruction for instruction decoding.

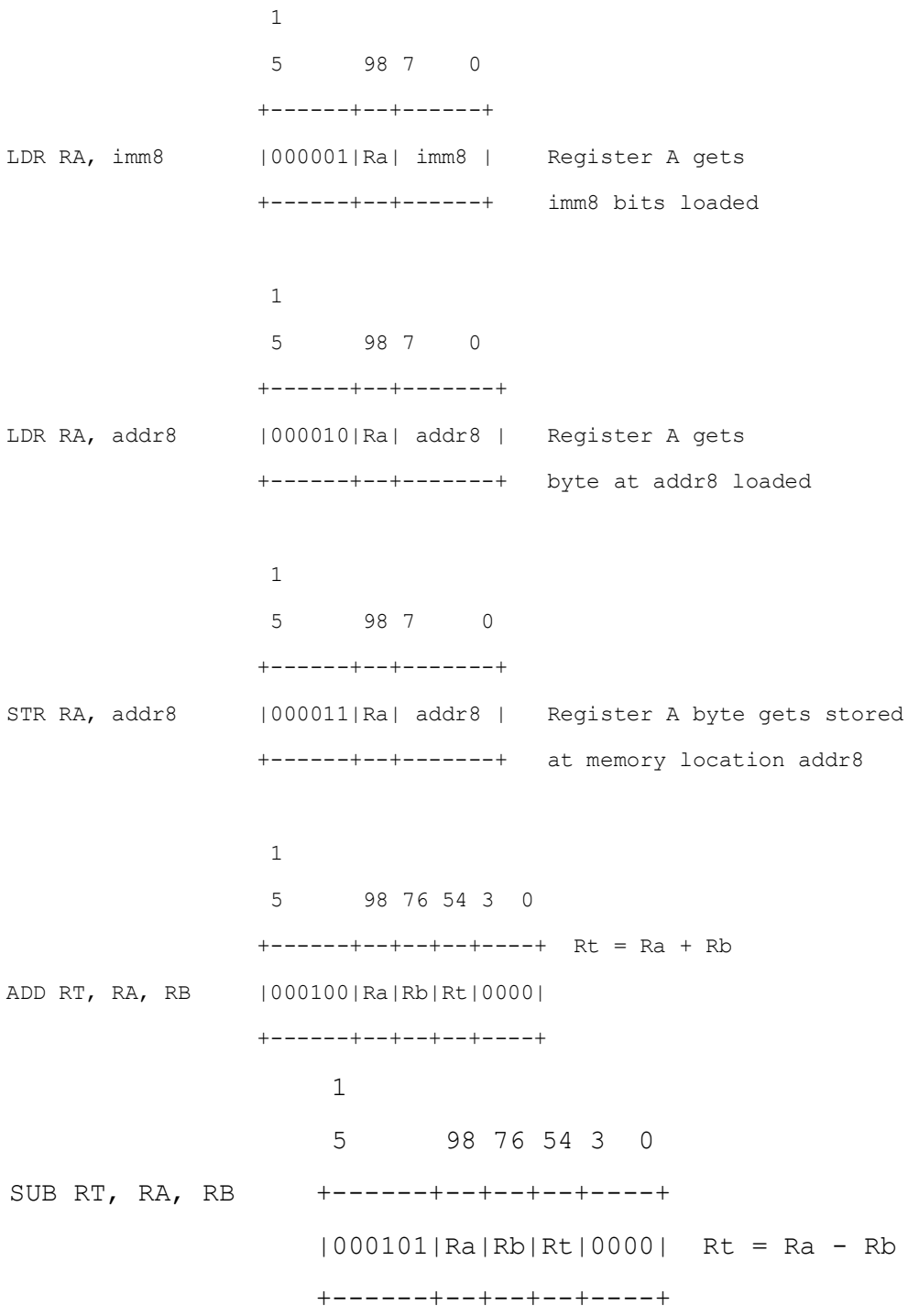
The system has a Control Unit, a combinatorial circuit that uses four bits to encode what kind of instruction we are executing. LoadImmediate indicates the load register immediate instruction. The Load pin indicates when a Load Register from RAM is being executed. The Store pin indicates when a Store Register to RAM instruction is being executed. Lastly, the RegToReg pin indicates that an Add or Subtract instruction is being executed and is a mnemonic for register

to register transition. The Op output is a 1 for subtraction and a 0 for addition. The Data output is the immed8 8 bit output. The Addr output is for when a load register from memory or load register to memory instruction is being executed and decodes which address in the RAM is being addressed. There are three 2 bit signals Ra, Rb, and Rt which are used to select 1 of the 4 general purpose registers.

There are 3 tri-state buffers used to control the 8 bit value that gets placed on the RegDataW bus as an input to the registers. The controls are whether it is a RegToReg transfer, a Load from RAM, or a LoadImmediate value as decoded from the low 8 bits of the instruction register when it's a LoadImmediate instruction. Another tri-state buffer is controlled by the Store pin when the instruction being executed is a Store instruction to only enable the input path to the RAM when a Store instruction is being executed. Lastly two more tri-state buffers are used to drive the WriteReg input, connecting Ra register when the instruction is a LoadImmediate or a Load from Memory as the address of which register needs to be selected for writing to. The ReadToReg signal indicates that an ALU output needs to be written to one of the registers in which case the Rt 2 bit signal is selecting the Register that needs to be written to.

The next page details the instruction set. I chose 16 bits to allow for enough selectability in opcodes, so more instructions could be added later and also to be able to target 3 registers in the arithmetic instructions which take up a total of 6 bits worth of instruction bits when using the arithmetic instructions.

Instruction Set



I have built a Python program called annaBinAnnasm.py which acts as an assembler from assembly instructions to machine language. It provides for a .text section as well as a .data section. You invoke it on a source file like so:

python annaBinAnnasm.py sourceFile.s

It will output a **ram** image file that can be loaded into the RAM of the CPU and a **rom** file with the instructions that can be loaded into the ROM of the CPU design.

When using the Load Immediate instruction, the immediate value is introduced with a hash (#) character and what follows can be a decimal value or if a hex value is desired, you need a 0x prefix.

For a load register from memory instruction you omit the (#) from the parameter and then the number can be either a decimal, or a hex value which must be prefixed with 0x to treat it as hex. The same is true for the store register instruction. The argument is treated the same way.

The ADD instruction or the SUB instruction merely have to identify the target register first followed by register A and finally register B separated by commas.

The assembler permits comments by two / characters, e.g. // and the rest of the line is ignored from the assembly process as are blank lines.

I have copied the possible data directives from the example program. Notice that you can embed a quote character in a string by escaping it with a backslash (\) character.

```
// .byte     allocate one-byte  block of memory, and specify the initial contents
// .hword    allocate two-byte  block of memory, and specify the initial contents
// .word     allocate four-byte block of memory, and specify the initial contents
// .skip     n allocate n bytes of uninitialized storage, just a label placeholder
// .string   "the string" allocates ascii bytes with a null terminating byte

single_byte: .byte 0xaa
two_bytes:   .hword 0xbeef
four_bytes:  .word 0xfeedf00d
scratchpad:  .skip 1
a_string:    .string "38\"2392"
another_two: .hword 0xaabb
more_scratch: .skip 30
ping:        .byte 0xff
pong:        .byte 0xee
```