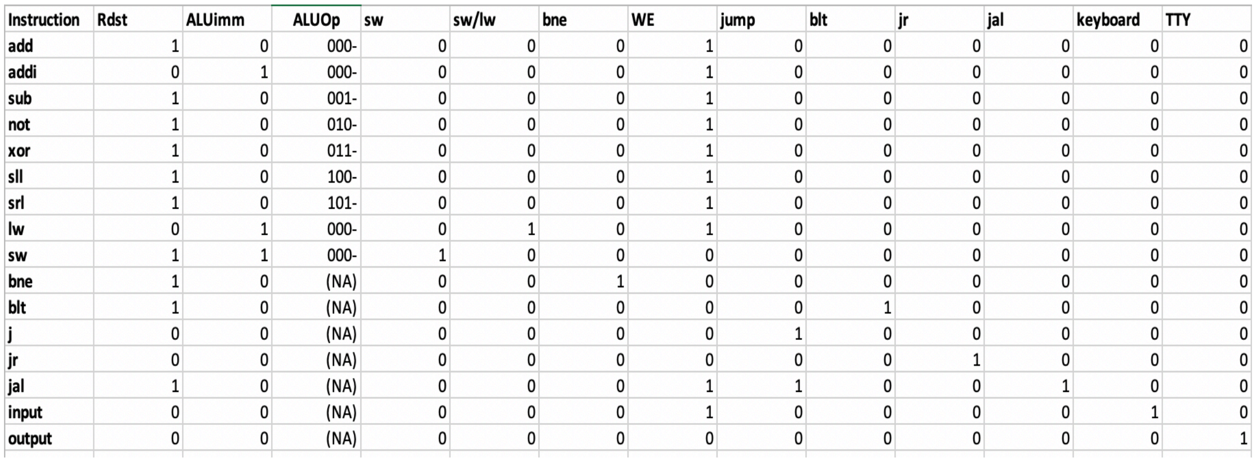
Jair Ulises Nino-Espino
Homework 4
11/16/2018



Instruction	Rdst	ALUimm	ALUOp	sw	sw/lw	bne	WE	jump	blt	jr	jal	keyboard	TTY
add	1	0	000-	0	0	0	1	0	0	0	0	0	0
addi	0	1	000-	0	0	0	1	0	0	0	0	0	0
sub	1	0	001-	0	0	0	1	0	0	0	0	0	0
not	1	0	010-	0	0	0	1	0	0	0	0	0	0
xor	1	0	011-	0	0	0	1	0	0	0	0	0	0
sll	1	0	100-	0	0	0	1	0	0	0	0	0	0
srl	1	0	101-	0	0	0	1	0	0	0	0	0	0
lw	0	1	000-	0	1	0	1	0	0	0	0	0	0
sw	1	1	000-	1	0	0	0	0	0	0	0	0	0
bne	1	0	(NA)	0	0	1	0	0	0	0	0	0	0
blt	1	0	(NA)	0	0	0	0	0	1	0	0	0	0
j	0	0	(NA)	0	0	0	0	1	0	0	0	0	0
jr	0	0	(NA)	0	0	0	0	0	0	1	0	0	0
jal	1	0	(NA)	0	0	0	1	1	0	0	1	0	0
input	0	0	(NA)	0	0	0	1	0	0	0	0	1	0
output	0	0	(NA)	0	0	0	0	0	0	0	0	0	1

The first few instructions use the ALU to do mathematical computations, whereas the rest of the instructions do not use the ALU so under the ALUop (NA) is written in since the signal will not be turned on.

The **add** instruction takes the value from two register and adds them up and places it in rd. Rdst is a signal to the mux to let the writereg either come from the mips code of Rt or Rd. Since we are dealing with an R instruction, Rdst is set to 1 to let the Rd opcode through. The ALU signal is turned on which will translate the 4-bit optype instruction into a 3-bit optype for the ALU. The ALU optype for add is 000 in my case. This will add the register designated by Rt and Rd and WE signal will be turned on to save that value in Rd.

Instructions **sub, not, xor, sll, srl** have a similar logic as the add instruction except the optypes will be different which will lead to different 3-bit ALU optype.

For instruction **addi**, the Rdst signal will be turned off to allow the Rt mips code to go to the writiereg and allow the final value to be written into whichever register Rt designates. The 3-bit ALU optype will still be 000 since we are still adding except, we are adding to a 6-bit value. This 6-bit value is sign extended to 16 and then the ALUimm signal will be turned on to allow the immediate value to travel through the readreg2 pipeline instead of Rt. Therefore, at the ALU Rs and the immediate are added.

The **lw** instruction turns on the signal ALUimm since the address in ROM is obtained by rs + D. D is the immediate that is sign extended and added to Rs. Since we are adding, the ALUOp signal is turned on to allow the 3-bit add optype to travel to the ALU. The address then travels into ROM and the value at that address is outputted to the other side. To allow the value to travel through the data pipeline, the lw/sw signal is turned on for the mux. This value then travels back into the register and placed in whichever register was designated for Rt. Since we are writing into Rt and not Rd, the Rdst signal is turned off and the WE signal is turned on.

The **sw** instruction also turns on the ALUimm and ALUOp signals to add the immediate and the Rs value and determine the address of storage in ROM. This address then travels into the ROM and in the same time, the value is Rt travels into the data of ROM. The sw signal is turned on to store the value of Rt to the designated address of rs + D. Since we are not writing into any of the registers, the lw/sw and WE signals are left off and the Rdst is turned on.

The ALU always checks if the two inputs from Rs and Rt are equal or not, no ALU optype is required. However, the alternative address generated by the **bne** instruction does not go into the PC until the bne signal is on. If the bne signal is on and Rs and Rt are not equal the mux allows PC+1+B to continue into the PC instead. B is sign extended to allow proper addition. Rdst is turned to 1 to allow the Rt value to travel through the pipeline and correctly compare the values in Rs and Rt.

Blt uses the similar logic as bne, an output was implemented that always checks if Rs is less than Rt. If so the blt signal is turned on and the alternative address is passed to the PC.

For the **jump** instruction all the signals are turned off since we are not using the register file to perform such instruction. If the jump signal is on, the mux will allow the jump address, with the attached four most significant bits from PC, to travel back to the PC.

The **jr** instruction also turns off all the signal expect for the jr signal for the mux. This signal allows the value stored in Rs to travel back to the PC as the new address.

Jal instruction uses the jump instruction since the address is manipulated the same way. However, the jal signal, when turned, activates the mux to allow the current PC+1 to travel in the data pipeline. The data is stored in the r7 register by turning on the Rdst signal. The jal signal allows the value of 7 to travel to the writereg instead of the Rd value to store the value in r7.

The **input** instruction turns on the WE and the keyboard signal. If the input is valid, it allows the 7-bit representation of the value to be extended to 16-bit with rt[7] being 0. The keyboard signal also turns on a mux to allow this value to travel through the data pipeline and store the value in the designated register of Rt, which is why the WE signal is also on. If the input is not valued the rt[7] value becomes 1 and the other bits are 0.

The **output** instruction turns off all the signals except just one which is the TTYWE signal. This instruction takes the lowest 7-bits of Rs and the TTYWE signal is turned on allowing the 7-bits to be translated into a character onto the TTY screen.