Assignment 2

Introduction to Assembly Language

Computer Structure

Alfonso Pineda (100472157)

Course 2023/2024

Degree in Computer Engineering

TABLE OF CONTENTS

Exercise 1 2
Exercise 2 6
Conclusion 6

Exercise 1

Name	RT Language	Control signals	Design decisions
La reg addr	MAR <- PC	(T2, C0),	It was simple to do this, because there was no room to improve the instruction
	MBR <- MEM[MAR]	(TA, R, BW=11, M1=1, C1),	
	REG <- MBR	(M2, C2, T1, LC, MR=0, SELC=10101, A0, B, C=0)	

lc reg1 reg2 reg3	RT1 <- REG3
	MAR <- REG3
		(MR=0, SELA=01011, T9, C0, C4)	First I used the content of register 3 to pull the content of the memory address to MBR, at the same time I calculated the next address in order to not waste cycles. Then I just repeated the same with reg3 + 4

	MBR <- MEM[MAR]
	MAR <- RT1 + 4
		(TA, R, BW=11, M1=1, C1, MA=1, MB=10, SELCOP=01010, MC, T6, C0, SELP=11, M7=1, C7)	

	MBR <- MEM[MAR]
	REG1 <- MBR
		(TA, R, BW=11, M1=1, C1, T1, LC, MR=0, SELC=10101, SE)	

	MBR <- MEM[MAR]
	REG2 <- MBR
		(T1, LC, MR=0, SELC=10000, SE, A0, B, C=0)	

addc reg1 reg2 reg3 reg4	reg1 <- reg1 + reg3	(MR=0, SELA=10101, MA=0, SELB=01011, MB=00, SELCOP=1010, MC, T6, SELC=10101, LC, SELP=11, M7, C7)	This was very straightforward

	reg2 <- reg2 + reg4	(MR=0, SELA=10000, MA=0, SELB=00110, MB=00, SELCOP=1010, MC, T6, SELC=10000, LC, SELP=11, M7, C7, A0, B, C=0)	
mulc reg1 reg2 reg3 reg4	RT1 <- reg1 * reg3	(MR=0, SELA=10101, MA=0, SELB=01011, MB=00, SELCOP=1100, MC, T6, C4, SELP=11, M7, C7)	First I multiply, reg1 and reg3 to store the result in RT1, then I did the same with reg2, reg4 and RT2, after that I subtracted them and store the result in RT3 in order to “repeat” the process with the imaginary part of the algorithm of multiplication between two complex numbers, after adding RT1 and RT2 I only needed to store the result in reg2 and RT3 in re1
	RT2 <- reg2 * reg4	(MR=0, SELA=10000, MA=0, SELB=00110, MB=00, SELCOP=1100, MC, T6, C5, SELP=11, M7, C7)	
	RT3 <- RT1 - RT2	(MA=1, MB=01, SELCOP=1011, MC, SELP=11, M7, C7, C6)	
	RT1 <- reg1 * reg4	(MR=0, SELA=10101, MA=0, SELB=00110, MB=00, SELCOP=1100, MC, T6, C4, SELP=11, M7, C7)	
	RT2 <- reg2 * reg3	(MR=0, SELA=10000, MA=0, SELB=01011, MB=00, SELCOP=1100, MC, T6, C5, SELP=11, M7, C7)	
	reg2 <- RT1 + RT2	(MA=1, MB=01, SELCOP=1010, MC, SELP=11, M7, C7, T6, SELC=10000, LC)	
	reg1 <- RT3	(T7, LC, SELC=10101, A0, B, C=0)	

hcf	pc <- 0	(EXCODE=0, T11, M2=0, C2, M7, C7, A0, B, C=0)	I only needed to store 0 in pc (I found out that this is an inside joke of IBM)
beqc reg1 reg2 reg3 reg4 offset	RT2 <- SR	(T8, C5)	I based this instruction in the one presented in riscv, following the same principles at trying to maintain the state register, then comparing registers and after that depending on the result make a jump to the fetch microinstruction or not and the final step is to return rt2 to sr (it's original value)
	SR <- REG1 == REG3	(SELA=10101, SELB=01011, MC=1, SELCOP=1011, SELP=11, M7, C7)	
	IF SR == 0 JUMP TO BCK2FTCH	(A0=0, B=1, C=110, MADDR=bck2ftch)	
	SR <- REG1 == REG3	(SELA=10000, SELB=00110, MC=1, SELCOP=1011, SELP=11, M7, C7)	
	IF SR == 0 JUMP TO BCK2FTCH	(A0=0, B=1, C=110, MADDR=bck2ftch)	
	RT1 <- PC	(T2, C4)	
	RT2 <- OFFSET	(SE=1, OFFSET=0, SIZE=10000, T3, C5)	
	PC <- RT1 + RT2	(MA=1, MB=1, MC=1, SELCOP=1010, T6, C2, A0=1, B=1, C=0)	
	SR <- RT2	bck2ftch: (T5, M7=0, C7, (A0, B, C=0)	
call addr	RA <- PC	(T2, MR=1, SELC=00001, LC)	This was also quite straightforward, I only needed to pass pc to ra and set pc to 0
	PC <- ADDR	(SE, SIZE=10100, OFFSET=0, T3, M2=1, C2, A0, B, C=0)	

ret	PC <- RA	(MR=1, SELA=00001, T9, M2=0, C2, A0, B, C=0)	only need to set pc to a value from a register (ra)

sc reg1, reg2, (reg3)	MAR <- REG3
	
	RT1 <- REG3
		(MR=0, SELA=01011, T9, C0, C4)	The idea was to set reg3 to mar and rt1 in one cycle, then set reg1 to mbr and then to use the memory unit and ALU “at the same time” sum a byte to rt1 and pick the contents of mar in memory. After that it was quite straightforward

	MBR <- REG1	(MR=0, SELA=10101, T9, M1=0, C1)	

	MEM[MAR] <- MBR
	
	MAR <- RT1 + 4
		(BW=11, TA, TD, W, MA=1, MB=10, SELCOP=1010, MC=1, T6, C0)	

	MBR <- REG2	(MR=0, SELA=10000, T9, M1=0, C1)	
	MEM[MAR] <- MBR	(BW=11, TA, TD, W, A0, B, C=0)	

Exercise 2

	Clock cycles without extension	Clock cycles with the extension	Improvements (%)
A == B	34	4	88.23%
A != B	34	4	88.23%

I am not entirely sure that I made reliable tests because it returns the same results over and over again (This doesn't make a lot of sense to me because it should be a difference in clock cycles between adding and multiplying) getting the same results looks not reliable, but it validates the theory because it was obvious that the version with the extension would be a lot faster, being the result almost 90% faster corresponds to the theory.

Starting the comparison between ISAs it's obvious that the one with the extension will be a lot faster always, but the drawback is that it requires more memory to run and a special processor for it, not being compatible with RISC-V processor meaning that the cost for this custom processors will be more expensive than the RISC-V ones. Long story short and answering the question of which would I pick to work with is that: If I can pay the one with the extension and I have no problem with the memory to store the extra instructions and microinstructions I would pick the one with the extension

Conclusion

This assignment and wepsim are perfect to familiarize myself with microinstructions and how they work. It's great to have a tool that is as good as wepsim to see possible errors and practice all the theory given in the course. Some problems that I found was to select registers, jump from one microinstruction to another that's it, this entire assignment was done in more or less 22 hours, it could have been less if I had organized myself better to ask questions to the professors but I did all the work as fast as I could and as good as it came.