Universidad
Carlos III de Madrid

COMPUTER STRUCTURE

BACHELOR IN COMPUTER SCIENCE AND ENGINEERING

Assignment 2
Introduction to microprogramming

Course 2023/2024

Salvador Ayala Iglesias | 100495832 | 100495832@alumnos.uc3m.es | g89
Inés Fuai Guillén Peña | 100495752 | 100495752@alumnos.uc3m.es | g89

Universidad
Carlos III de Madrid

Table of Contents

[Exercise 1](#)
[Exercise 2](#)
[Problems encountered](#)
[Time spent](#)
[Conclusions](#)

Universidad Carlos III de Madrid

Exercise 1

We use the structure of the WepSIM Simulator.

Fetch:

1. MAR ← PC | T2, C0
2. MBR ← MP[MAR] | Ta, R, BW = 11, C1, M1
   PC ← PC+4 | M2, C2
3. IR ← MBR | T1, C3
4. Decode

Name of the instruction	RT Language	Control signals	Design decisions
la RR1, U32	5. MAR ← PC 6. MBR ← MP[MAR], PC ← PC+4 7. RR1← MBR	5. T2, C0 6. Ta, R, BW=11, M1=1, C1 M2=1, C2 7. T1, SelC=10101, MR=0, LC, A0=1, B=1, C=0	We have to fetch a second time to obtain the value U32, given that this instruction occupies two words. We save it in register RR1 and then, we jump to fetch.
sc RR1, RR2, (RR3)	5. MAR ← RR3 6. MBR ← RR1 7. MP[MAR] ← MBR 8. MAR ← RR3+4 9. MBR ← RR2 10. MP[MAR] ← MBR	5. SelA=1011, T9, C0 6. SelA=10101, T9, M1=0, C1 7. Ta, Td, W 8. SelA=1011, MB=10, SelCop=1010, T6, C0 9. SelA=10000, T9, M1=0, C1 10. Ta, Td, W, A0=1, B=1, C=0	We overwrite the content of the memory position of RR3 with the content of register RR1. Now, we add 4 to the register RR3 and we overwrite the content of the memory position of the result obtained with the content of register RR2. Finally, we jump to fetch.
lc RR1, RR2, (RR3)	5. MAR ← RR3 6. MBR← MP[MAR] 7. RR1← MBR 8. MAR← RR3 + 4 9. MBR← MP[MAR] 10. RR2← MBR	5. SelA=1011, T9, C0 6. Ta, R, M1, C1, BW=11 7. T1, SelC=10101, LC 8. SelA=1011, MB=10, SelCop=1010, T6, C0 9. Ta, R, M1, C1 10. T1, SelC=10000, LC, A0=1, B=1, C=0	We save in the register RR1 the value obtained from the memory position of register RR3. Now, we add 4 to the register RR3 and save the value obtained from the memory position of the addition in the register RR2. Finally, we jump to fetch.
addc RR1, RR2, RR3, RR4	5. RR1← RRI+RR3 6. RR2 ← RR2+RR4	5. SelA=10101, SelB=1011, MR=0, SelCop=1010, MC=1, SelC=10101, T6, LC, SelP=11, M7=1, C7 6. SelA=10000, SelB=110, MR=0, SelCop=1010, MC=1, SelC=10000, LC, T6, SelP=11, M7=7, C7, A0=1, B=1, C=0	We add the values from registers RR1 and RR3 and we save the result in register RR1. We update the status register We add the values from registers RR2 and RR4. Later, we save the result in register RR2. Finally, we jump to fetch.
mulc RR1, RR2, RR3, RR4	5. RT1 ← RR1*RR3 6. RT2 ← RR2*RR4	5. SelA=10101, SelB=1011, MR=0, MC=1, SelCop=1100, T6, C4 6. SelA=10000, SelB=110, MR=0, MC=1, SelCop=1100, T6, C5	First, we calculate the multiplication of the values in the registers RR1 and RR3 and save the result in the temporary register RT1. We do the same operation for registers RR2 and RR4 and save the result in

	7. RT3 ← RT1-RT2	7. MA=1, MB=01, MC=1 SelCop=1011, C6, SelP=11, M7, C7	temporary register RT2. Finally, we calculate the subtraction of both values, already calculated, and save it in the temporary register RT3, in order to not overwrite the original value from RR1 for the following operations..
	8. RT1 ← RR1*RR4	8. SelA=10101, SelB=110, MR=0, MC=1, SelCop=1100, T6, C4	Now, we calculate the multiplication of registers RR1 and RR4 and save the result in the temporary register RT1. We do the same operation for registers RR2 and RR3 and save the result in temporary register RT2.
	9. RT2 ← RR2*RR3	9. SelA=10000, SelB=1011, MR=0, MC=1, SelCop=1100, T6, C5	
	10. RR1 ← RT3	10. T7, SelC=10101, LC	We move the value from RT3 to the register RR1.
	11. RR2 ← RT1+RT2	11. MA=1, MB=01, MC=1, SelCop=1010, T6, SelC=10000, LC, A0=1, B=1, C=0	We calculate the addition of the values of RT1 and RT2, already calculated, and save it in register RR2. Finally, we jump to fetch.
beqc RR1, RR2, RR3, RR4, S6	5. RT1← SR	5. T8,C4	First, we save the value of the status register in RT3 by multiplying it by 1. It will be later restored.
	6. RT3 ← RT1 (SR)	6. MA=1, MB=11, SelCop=1100, C6	
	7. [none] ← RR1 - RR3 SR ← flags	7. SelA=10101, SelB=1011, MR=0, MC=1, SelCop=1011, SelP=11, M7=1, C7	Then, we compare the real part. If they are different, we directly jump to fetch. We use the flag Z for this, subtracting both registers.
	8. IF SR.z==1, jump to fetch	8. A0=0, B=1, C=110, MADDR=14	
	9. [none] ← RR2 - RR4 SR ← flags	9. SelA=10000, SelB=110, MR=0, MC=1, SelCop=1011, SelP=11, M7=1, C7	Now we compare the imaginary part the same way we did with the real part. Two comparisons are done in order to avoid possible unnecessary cycles.
	10. IF SR.z==1, jump to fetch	10. A0=0, B=1, C=110, MADDR=14	
	11. RT1 ← PC	11. T2, C4	If both numbers are the same, we branch by adding the offset (found at the end of the instruction) to the next instruction (PC).
	12. RT2 ← IR(S6)	12. SE=1, Size=110,Offset=0, T3, C5	
	13. PC ← RT1+RT2	13. MA=1, MB=01, MC=1, SelCop=1010, T6, M2=0, C2	
	14. SR ← RT3	14. T5, M7=0, C7, A0=1, B=1, C=0	We restore the status register and jump to fetch.
call U20	5. R1(ra) ← PC	5. T2, SelC=1, MR=1, LC	We save the value of PC in the register R1(ra)
	6. PC ← IR(U20)	6. SE=0, SIZE=20, Offset=0, T3, M2=0, C2, A0=1, B=1, C=0	We save the value obtained from the IR of U20 on the PC. Jump to fetch.

ret	5. PC ← R1(ra)	5. SelA=1, MR=1, T9, M2=0, C2, A0=1, B=1, C=0	We save the value of the register R1(ra) on the PC. Finally, we jump to fetch.
hcf	5. PC ← R0(0x00) SR ← R0(0x00)	5. SelA=0, MR=1, T9, M2=0, C2, M7, C7, A0=1, B=1, C=0	We save the value from the register R1(0x00) on the PC and in the SR.

Exercise 2

	Clock cycles without extension	Clock cycles with extension	Improvements (%)
A == B	95	86	10.47%
A != B	93	88	5.69%

We can see a significant difference between our implementation and the one using base RISC-V instructions. Although the improvement might seem small, we have to consider that we are only using two instructions (beqc and either addc or mulc). In a bigger program, the difference in clock cycles would be much bigger, so an extension of the basic instructions of RISC-V is highly recommended if working with complex numbers.

Program code:

```
.text
    no_ext:
        # To implement with RISC-V instructions (without extension)
        lw t0 0(a0) # load r1 real
        lw t1 4(a0) # load r2 imaginary
        lw t2 0(a1) # load r3 real
        lw t3 4(a1) # load r4 imaginary

        beq t0 t2 16 # skip the sum
        add a0 t0 t2 # real
        add a1 t1 t3 # imaginary
        ret # return

        beq t1 t3 16 # skip the sum
        add a0 t0 t2 # real
        add a1 t1 t3 # imaginary
        ret # return

        mul t5 t0 t2 # first part of the formula given in ex1
        mul t6 t1 t3
        sub a0 t5 t6
```

```
        mul t5 t0 t3 # second part of the formula given in ex2
        mul t6 t1 t2
        add a1 t5 t6

        ret # return

    with_ext:
        # To implement with RISC-V instructions (with extension)
        # The order of load and the register used are changed in order to have the result
        directly in a0 and a1 instead of needing to move it later to return.

        lw t0 0(a1) # load r3 real
        lw t1 4(a1) # load r4 imaginary
        lw a1 4(a0) # load r2 imaginary
        lw a0 0(a0) # load r1 real

        beqc a0 a1 t0 t1 12 # skip the sum
        addc a0 a1 t0 t1 # sum of complexes
        ret # return

        mulc a0 a1 t0 t1 # multiplication of complexes
        ret # return

main:
        ##### WITH new extension #####
        rdcycle s0
        la a0, a
        la a1, b
        call with_ext
        rdcycle s1
        sub s1 s1 s0

        ##### WITHOUT extension #####
        rdcycle s0
        la a0, a
        la a1, b
        call no_ext
        rdcycle s2
        sub s2 s2 s0

        # the end
        hcf
```

Problems encountered

The main problems encountered during the development of this practice were the lack of familiarity with WepSIM. Taking that out of the equation, no major problems were found. With our prior knowledge in microprogramming, the development of the practice was not complicated. Taking care of toggling the correct signals and not sending more than one piece of data to the internal bus in each cycle are the most important things to take into account.

Time spent

The development of this practice has taken us around 20 hours, around 1 week. The usage of WepSIM to edit, compile and debug the code was key to achieve the objectives in such a short time.

Conclusions

The practice has significantly enhanced our comprehension of how microprogramming works, particularly in relation to the processor and the control unit. This practical experience has played a key role in reinforcing our understanding of the respective units covered in the course.

We prioritized optimizing code efficiency without altering the fundamental structure of provided formulas. Using WebSim for processor simulations provided us a friendly space, letting us see and control different parts of how processors work, along with the debugging process.

In conclusion, learning about processors and using WebSim improved our academic experience notoriously. It showed us how the things we studied in theory are connected with real-world practice and reinforced significantly the knowledge needed for our careers.