Universidad Carlos III
Computer Structure
Microprogramming introduction
Course 2019-20

Members:

Álvaro Marco Pérez

Ramón Hernández León

GROUP: **88**

Index

Exercise 1: Design of the instructions

Instruction name	C	Elementary operations	Control signals	Design decisions
ld	C1	BR[R_target] ← BR[R_source]	SELC=10101, SELA=10000, MR=0, LC=1, T9=1	As we are not using the internal bus, we can operate the instruction completely in only one cycle.
ldi	C1	BR[R_target] ← U16	SELC=10101, MR=0, SE=0, SIZE=10000, OFFSET=00000, T3, C, LC	As we only use the internal bus for taking U16 from IR, we can operate the instruction in only one cycle.
ld	C1	MAR← BR[R_source]	SELA=10000, MR=0, T9=1, C0=1	In the codification of this instruction we have used three cycles as we had to use two times the internal bus and we had to wait MAR's signal to be written before reading it
	C2	MBR ← M[MAR]	TA=1, R=1, BW=11, M1=1, C1=1	
	C3	BR[R_target] ← MBR	SELC=10101, MR=0, T1=1, LC=1	
add_a	C1	A ← A + BR[R_source]	SelA=10101, MR=0, T9, C4	Two cycles were needed in this instruction because we had to take the source register with MR=0 (from IR) and A register with MR=1, which obliges us to divide the operation
	C2	A ← A + R_source	SELB=00100, MR=1, MA=1, MB=00, SELCOP=01010, MC=1, SELP=11, M7=1, C7=1, T6, SELC=00100, MR=1, LC=1	
addi_a	C1	RT2 ← S16	SE=1, SIZE=10000, OFFSET=00000, T3, C5	We have to wait to S16 to be stored in RT2, this makes a cycle. Then we store the value of the addition in A.
	C2	A ← RT2 + A	SELA=00100, MR=1, MA=0, MB=01, SELCOP=01010, MC=1, SELP=11, M7, C7, T6, SelC=00100, MR=1, LC=1	
inc	C1	R_source ← BR[R_source] +1	SelA=10101, MR=0, MA=0, MB=11, SELCOP=01010, MC=1, T6, SelP=11, M7=1,	We have been able to take the source register from IR, add

			C7, SelC=10101, MR=0, LC	one to its content, and store this value in it.
dec	C1	R_source ← BR[R_source] -1	SelA=10101, MR=0, MA=0, MB=11, SELCOP=01011, MC=1, T6, SelP=11, M7=1, C7, SelC=10101, MR=0, LC	We have been able to take the *source* register from IR, subtract one to its content, and store this value in it.
jp	C1	RT2 ← S16	SE=1, Size=10000, Offset=00000, T3, C5	For the first and second cycle we stored *S16* and *PC* respectively, as both need to use the internal bus. Then we perform the addition.
	C2	RT1← PC	T2, C4	
	C3	PC ← RT1+ RT2	MA=1, MB=01, SELCOP=01010, MC=1, T6, M2=0, C2	
jpz	C1	RT2 ← PC	T2, C5	We have make the instruction as fast as possible. The first two cycle were needed to add *PC* and *S16*. The separation between cycles 3 and 4 was also needed because we had to use twice the ALU. On the 5th cycle we needed to check if *S16* was equal to 0 and then, if not, writing this value on *PC*.
	C2	RT1 ← S16 if (previous SR ==0) → go to fetch	SE=1, Offset=00000, Size=10000, T3, C4, MADDR=fetch, B=0, A0=0, C=110	
	C3	RT3 ← RT1 + RT2	MA=1, MB=01, SelCop=1010, MC=1, C6	
	C4	RT1 ← SR SR ← S16*1	MA=1, MB=11, SelCop=1100, MC=1, SelP=11, M7=1, C7, T8, C4	
	C5	If (S16==0) → go to fetch	MADDR=fetch, B=0, A0=0, C=110	
	C6	PC ← RT3 SR ← RT1 * 1	T7, M2=0, C2, MA=1, MB=11, SelCop=1100, MC=1, SelP=11, M7=1, C7	
call	C1	MAR ← SP - 4 SP ← SP - 4	SelA=11101 MR=1, MA=0, MB=10, SelCop=1011, MC=1, T6, C0, SelC=11101, LC	In this instruction we have used the less possible number of cycles. In each one of them we use the internal bus for different purposes.
	C2	MBR ← PC	T2, M1=0, C1	
	C3	Mem[MAR] ← MBR PC ← U16	Ta, BW=11, Td, W, SE=0, Size=10000, Offset=0, T3, M2=0,C2, SE=0, Size=10000, Offset=00000, T3	

Ret	C1	MAR ← SP RT3 ← SP + 4	SelA=11101, MR=1, T9, C0, MA=0, MB=10, SelCop=1010, MC=1, C6	In the second cycle we have take advantage that we had to access memory to pass the value in *RT3* to the *SP* register.
	C2	SP ← RT3 MBR ← M[MAR]	Ta, R, BW=11, M1=1, C1, T7, SelC=11101, MR=1, LC	
	C3	PC ← MBR	T1, M2=0, C2	
Halt	C1	PC ← RA + RB	MA=0, MB=00, SelCop=1010, MC=1, T6, M2=0, C2	As *RA* and *RB* are set by default = 0 which each cycle (unless they are used), we use them to add and write the result in PC.
push	C1	SP ← SP - 4 MAR ← SP - 4	SelA=11101, MR=1, MA=0, MB=10, SelCop=1011, MC=1, T6, SelC=11101, MR=1, LC, C0	The reasons to have three cycles on push instruction were that on cycles one and two we use the internal bus for different reasons, and, in the last cycle, we needed to wait to MBR to be written.
	C2	MBR ← R_source	SelA=10101, MB=0, T9, M1=0, C1	
	C3	M[MAR] ← MBR	Ta, BW=11, Td, W	
pop	C1	MAR ← SP RT3 ← SP +4	SelA=11101, MR=1, T9, C0, MA=0, MB=10, SelCop=1010, MC, C6	We use three cycles in this instruction as in each one of them the internal bus is used for different purposes.
	C2	MBR ← M[MAR] SP ← RT3	Ta, R, BW=11, M1=1, C1, MR=1,SelC=11101, MR=1, T7, LC	
	C3	SP ← MBR	SelC=10101, MR=0, T1, LC	

At the end of each instruction we jump to fetch, with the following microcode instructions:

C=0, B=1, A0=1

Exercise 2: instruction testing

```
.data
    vector: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
.text
    addv:
      ##push $a0 and $a1
        #------addi $sp $sp -8
         #------sw $a0 0($sp)
          push IX
        #------sw $a1 4($sp)
          push IY
      ##$v0 will contain the addition of the vector elements
         #------li $v0 0
          ldi HL 0
       b1:
        #------beq $a0 $0 f1
          ld A IX
          addi_a 0
     jp0: jpz 4
           jp 44
         #------lw $t0 ($a1)
          ld BC (IY)
         #------add $v0 $v0 $t0
          ld A HL
          add_a BC
          ld HL A
         #------addi $a1 $a1 4
          ld A IY
          addi_a 4
          ld IY A
         #------addi $a0 $a0 -1
          ld A IX
          addi_a -1
          ld IX A
         #------b b1
       jp1:  jp -56
       ##pop $a1 and $a0
```

```
f1:
 #------lw $a0 0($sp)
   pop IY
  #------lw $a1 4($sp)
   pop IX
  #------addi $sp $sp 8
##return
  #------jr $ra
   Ret
main:
  ##call addv subroutine
    #------li $a0 10
     ldi IX 10
    #------la $a1 vector
     ldi IY 0x01000
    #------jal addv
     call 0x08000
  # Finish the program
    #------li $v0 10
    #------syscall
     Halt
```

Advantages and disadvantages of Z80

On one hand, Z80 has few instructions and registers available. This makes instructions that were direct in MIPS32, much complex on Z80.

On the other hand, for this reason Z80 is easier to debug and update, which makes the program more dynamic.

Conclusions

Microprogrammed control units are much flexible and powerful than wired ones since they can be adapted to any other architecture. In addition, if the instructions are microprogrammed optimally, the performance of the control unit is also very high.

Apart from this design conclusion, we have have learned to code a microprogrammed a control unit.

Taking all together we have taken a good insight into the physical execution of the instructions inside the processor. In the end just performs simple arithmetic operations with binary values while the control unit takes all important decisions.

Problems encountered

When we were encoding and testing, separately, each instruction all of them were executed correctly. Nevertheless, when more than one came together some strange behaviour was observed (and afterwards corrected). Specially during the execution of the test program we noticed that many of MIPS32 instructions do not have an equivalent instruction in Z80. For that reason the same test program written with Z80 instructions is longer than the MIPS32 version.

From the beginning of this assignment, the interactive tools available in WepSim have made the testing of the instructions much easier. The values of all the components can be seen and modified. Learning how to use this tools has been crucial for reaching the goal of the practice.