MEMORY PRACTICE 2

Universidad Carlos III

Computer Structure 2021/2022

Marcos Caballero Cortés / 100451047 / 100451047@alumnos.uc3m.es / Group 89

Contenido

1. Exercise 1

Name of the statement	Elementary transfer operations	Control signals	Design decisions
mov RRE1, U32	C0: MAR←PC C1: PC←PC+4 MBR←MM[MAR] C2:BR[R1]←MBR C3: Jump to fetch	C0: T2, C0. C1: Ta, R, BW=11, M1, C1. C2: M2, C2, T1, SELC=10000, MR=0, LC=1. C3: A0=1, B=1, C=0.	As there are two words, we must go to the next word in the next clock cycle after putting the address in MAR to optimize the nº of clock cycle.
str RRE1, (RRE2)	C0: MAR←(RE2) C1: MBR←RE1 C2: MM[MAR]←MBR C3: Jump to fetch	C0: SELA=1011, MR=0, T9, C0. C1: MR=0, SELA=10101, T9, C1. C2: Ta, Td, BW=11, W. C3: A0=1, B=1, C=0.	The value in register 1 is stored in the memory address in the register 2. This implementation minimizes the number of clock cycles.
ldr RRE1, (RRE2)	C0: MAR←R2 C1: MBR←MM[MAR] C2: R1←MBR C3: Jump to fetch	C0: SelA=1011, MR=0, T9, C0. C1: Ta, R, BW=11, M1, C1. C2: T1, LC, SelC=10101. C3: A0,B,C=0	The value stored in the memory address given by R2 is stored in R1. This implementation minimizes the number of clock cycles.
adds RRE1, RRE2, RRE3	C0: R1←R2+R3, SR C1: Jump to fetch	C0: MR=0, SELA=10000, SELB=1011, MC, SelCop=1010, T6, SelC=10101, LC=1, SelP=11, M7, C7. C1: A0, B, C=0.	We store the sum of R2 and R3 in R1. This implementation minimizes the number of clock cycles. We also must update the status register.
adds RRE1, RRE2, S16	C0: RT2←IR[S16] C1: R1←R2+RT2, SR C2: Jump to fetch	C0: OFFSET=0, Size=10000, SE, T3, C5. C1: MR=0, SELA=10000, MB=01, MC, SelCop=1010, T6, SelC=10101, LC=1, SelP=11, M7, C7. C2: A0, B, C=0.	We store the sum of R2 and R3 in R1. This implementation minimizes the number of clock cycles. We also must update the status register.
mvns RRE1, RRE2	C0: BR[RE1]←NOTbitwise BR[RE2] C1: Jump to fetch	C0: SelA=1011, MR=0, MA=0, SelCop=0011, MC, T6, LC, SelC=10101, SelP=11, M7, C7. C1: A0, B, C=0.	We have to store in R1 the complement of the value stored in R2, for this we use now operation in the ALU. This implementation

			minimizes the number of clock cycles. We also must update the status register
cmp RRE1, RRE2	C0: BR[RRE1]🡐 BR[RRE2] C1: Jump to fetch	C0: SelA=10101, MR=0, SelB=01011, SelCop=1011, MC, SelP=11, M7, C7. C1: A0, B, C=0.	We compare the values of two registers to see in the status register if they were equal or one bigger than the other one. This implementation minimizes the number of clock cycles. We also must update the status register.
beq S16	C0: IF (bit Z of APRSR==0,go to fetch) Else: C1: RT1🡐IR[S16] C2: RT2🡐PC C3: PC🡐RT1+RT2 C4: Jump to fetch	C0: C=110, B=1, A0=0, MADDR=gotofetch. C1: OFFSET=0, Size=10000, SE, C4, T3. C2: T2, C5. C3: SelCop=1010, MC, MA=1, MB=01, T6, C2. C4: A0, B, C=0, gotofetch: (A0=1,B=1,C=0).	We have to search in the control memory if Z was equal to 0, if so, we had to go to the microinstruction selected. This implementation minimizes the number of clock cycles.
bl U16	C0: #BR[LR] 🡐 PC C1: PC 🡐 U16 C2: Jump to fetch	C0: T2, LC, SelC=1110, MR=1. C1: OFFSET=0, Size=10000, T3, C2. C2: A0, B, C=0.	This instruction was like 'jal' instruction in MIPS. This implementation minimizes the number of clock cycles.
bx RRE	C0: PC 🡐 BR[RE] C1: Jump to fetch	C0: SelA=10101, MR=0, T9, C2. C1: A0, B, C=0.	This instruction was like 'jr $ra' instruction in MIPS. This implementation minimizes the number of clock cycles.
halt	C0: PC🡐0x00 C1: SR🡐0x00 C2: Jump to fetch	C0: MR=1, SELA=00000, T9, M2=0, C2. C1: MR=1, SELB=00000, T10, M7=0, C7. C2: A0, B, C=0.	In this instruction we select the empty parts of the instruction to put them in PC and SR. This implementation minimizes the number of clock cycles.

U32 refers to a 32-bit unsigned integer. U16 to a 16-bit unsigned integer and S16 to a 16- bit signed integer. The value "S16" indicate that sign extension must be made while in "U16" no sign extension is made (filled with leading zeros).

BR will be used to refer to the Register File, and BR[reg1] to indicate the contents of the reg1 register. The integers stored in register are two complements 32 bits integers. MM[reg] refers to the contents of the memory position whose address is stored in the reg register.

2. Exercise 2.

```
sumav: # push $a0 and $a1
         addi $sp $sp -8
         sw $a0 4($sp)
         sw $a1 0($sp)
       # $v0 = sum of the vector elements
         li $v0 0
     b1: beq $a0 $0 f1
         lw $t0 ($a1)
         add $v0 $v0 $t0
         addi $a1 $a1  4
         addi $a0 $a0 -1
         b b1
       # pop $a1 and $a0
    f1:  lw $a1 0($sp)
         lw $a0 4($sp)
         addi $sp $sp 8
       # return
         jr $ra

main: # call sumav function
        li $a0 10
        la $a1 vector
        jal sumav
      # halt execution
        li $v0 10
        syscall
```

```
sumav:#push $R1 and $R2
      adds $SP $SP -4
      str $R1 ($SP)
      adds $SP $SP -4
      str $R2 ($SP)
      #push $R7
      adds $SP $SP -4
      str $R7 ($SP)
      #$R5 = sum of the vect
      mov $R5 0

  b1: cmp $R0 $R1
      beq f1
      ldr $R7 ($R2)
      adds $R5 $R5 $R7
      adds $R2 $R2 4
      adds $R1 $R1 -1
      cmp $R9 $R9 #as we don
      beq b1

  f1: #pop SR7
      ldr $R2 ($SP)
      adds $SP $SP 4
      #pop $R2 and $R1
      ldr $R2 ($SP)
      adds $SP $SP 4
      ldr $R1 ($SP)
      adds $SP $SP 4
      #return
      bx $LR

main: # call sumav function
      mov $R1 10
      mov $R2 vector
      bl sumav
      # halt execution
      halt
```

The main differences found between the two programs are:

- We couldn’t use the stack pointer when we wanted, in the instructions where we wanted to put more than two things in the stack, we had to subtract 4, get the value in the stack and repeat with the rest of things we wanted to add in the stack. In MIPS we could do ‘addi $sp $sp 8 /n sw $t1 4($sp) /n sw $t2 ($sp)’.
- Ending the program didn’t mean to store anything in v0, because we just had to call the instruction halt.
- We had to use cmp instruction to control the flow of the loops, while in MIPS we could do it with a branch instruction.
- We had to use ‘bx $LR’ instruction instead of b instruction to return to main instead of using ‘jr $ra’.

The advantages founded are:

- We had less instructions to use, so we had to understand less instructions, making it easier to find the correct way to do it.
- Instead of syscall we had to use halt to finish the program, which I think is easier, because you don’t have to memorize how you have to print or all the other 10 options you are available to do with syscall.

The disadvantages founded are:

- As there are less instructions to do the same thing as in MIPS, the way to do it in MIPS was simpler.
- When we return from the function, nothing is stored in $v0, so we had to store it in $R5.
- You can’t print something in the screen.
- You must use more lines to do the same things.

3. Conclusions and problems found.

I had a lot of problems understanding some of the requested instructions, I was alone all the time except the times I asked the teachers, which helped me a lot understanding those requested instructions.

I have found something in this subject in which I was totally focused because I really enjoyed doing this practice.

I had also problems in exercise 2 because some of the MIPS instructions such as ‘beq $R $R, end’, or ‘b loop’, where I had to spend some time thinking how I was going to do it.

I struggled a bit while doing it because I had to do it all on my own and I had to find a lot of time to finish this project.

I think I liked to do the project on my own because it helped me to understand more the processor making it my favorite part of the subject, in addition a little bit of help from a partner would have helped me to do it in less time.

I spent something around 24 hours to do this practice. At least I spent 18 hours trying to do ex1, where I had some troubles with some microinstructions such as ‘beq S16’, and at least 3 hours doing exercise 2, which was a little bit less demanding. The rest of the time was spent doing the memory.