ASSIGNMENT 2
COMPUTER STRUCTURE

Introduction to microprogramming

Laura Belizón Merchán (100452273) & Jorge Lázaro Ruiz (100452172)
Applied Mathematics and Computing
Group 121

100452273@alumnos.uc3m.es
100452172@alumnos.uc3m.es

Laura Belizón Merchán (100452273)
Jorge Lázaro Ruiz (100452172)
Applied Mathematics and Computing
Group 121

Index

Laura Belizón Merchán (100452273)
Jorge Lázaro Ruiz (100452172)
Applied Mathematics and Computing
Group 121

Exercise 1

Name of instruction	Elementary operations	Control signals	Design decisions
mov RE1, U32	MAR ← PC	T2, C0	Join two processes in the same clock cycle.
	MBR ← MM[MAR], PC ← PC + 4	Ta, R, BW=11, M1=1, C1, M2=1, C2	
	IR[RE1] ← MBR	T1, SelC=10000, MR=0, LC=1	
	Jump to fetch	A0, B, C=0	
str RE1, (RE2)	MBR ← RF[RE1]	SelA=10101, MR=0, T9, M1=0, C1	Join two processes in the same clock cycle.
	MAR ← RF[RE2]	SelA=1011, MR=0, T9, C0	
	MM ← MAR, MBR, Jump to fetch	Ta, W, BW=11, Td, A0, B, C=0	
ldr RE1, (RE2)	MAR ← RF[RE2]	SelA=1011, MR=0, T9, C0	Join two processes in the same clock cycle.
	MBR ← MAR	Ta, R, BW=11, M1, C1	
	RF[RE1] ← MBR, Jump to fetch	T1, LC, SelC=10101, MR=0, A0, B, C=0	
adds RE1, RE2, RE3	RF[RE1] ← RF[RE2] + RF[RE3], SR	MR=0, SelA=10000, SelB=1011, MC=1, SelCop=1010, T6, SelC=1010, LC=1, SelP=11, M7, C7	Join two processes in the same clock cycle.
	Jump to fetch	A0, B, C=1	
adds RE1, RE2, S16	RT1 ← S16	SE=1, Offset=0, Size=10000, T3, C4	Load the value of S16 in a temporal register.
	RF[RE1] ← RT1 + RF[RE2], SR, Jump to fetch	MR=0, SelB=10000, MA=1, MB=00, MC=1, SelCop=1010, T6, SelC=10101, LC=1, SelP=11, M7, C7, A0, B, C=0	Since we loaded S16 in RT1, we need to take the value in RE2 through RB. Join three processes in the same clock cycle.
mvns RE1, RE2	RF[RE1] ← not(RF[RE2]), Jump to fetch	MC=1, MR=0, SelA=10101, MA=0, SelCop=1011, T6, SelC=1011, LC=1, SelP=11, M7, C7, A0, B, C=0	Join all processes in the same clock cycle.
cmp RE1, RE2	SR ← RF[RE1] – RF[RE2], Jump to fetch	MR=0, SelA=10101, SelB=1011, MC=1, SelCop=1011, SelP=11, M7, C7, A0, B, C=0	The result is not stored, but we update the status register (we will later use it to see the Z bit). Join both processes in the same clock cycle.

beq S16	RT2 ← SR	T8, C5	If Z=0, we go to "bck2ftch".
		A0=0, B=1, C=110, MADDR=bck2ftch	
	SR ← RT2	T5, M7=0, C7	
	PC ← S16	SE=1, Offset=0, Size=10000, T3, M2=0, C2	
	Jump to fetch	A0, B, C=0	
	bck2ftch	T5, M7=0, C7	We reset the SR but we do not change the value of the PC.
		A0, B, C=0	
bl U16	RF[LR] ← PC	T2, C, LC=1, MR=1, SelC=1110	Join two processes in the same clock cycle.
	PC ← PC + U16, Jump to fetch	SE=0, Size=10000, Offset=0, T3, M2=0, C2, A0, B, C=0	
bx RE	PC ← RF[RE], Jump to fetch	SelA=10000, MR=0, T9, M2=0, C2, A0, B, C=0	Join both processes in the same clock cycle.
halt	PC ← RF[R0]	SelA=0, MR=0, T9, M2=0, C2	Join two processes in the same clock cycle.
	RF[R13] ← RF[R0], Jump to fetch	SelA=0, MR=0, T9, LC=1, SelC=1101, A0, B, C=0	

Laura Belizón Merchán (100452273)
Jorge Lázaro Ruiz (100452172)
Applied Mathematics and Computing
Group 121

Exercise 2

ARM	MIPS32	Advantages and disadvantages
mov RE1, U32	li RE1 inm la RE1 address	ARM covers the functionality of two MIPS instructions. However, two words are needed for this instruction.
str RE1, (RE2)	sw RE, address	For ARM we first need to load the address on a register. In addition, an offset cannot be stated directly, unlike in MIPS32.
ldr RE1, (RE2)	lw RE, address	For ARM we first need to load the address on a register. In addition, an offset cannot be stated directly, unlike in MIPS32.
adds RE1, RE2, RE3	add RE1, RE2, RE3	In ARM we have the same name for adding a register and an immediate, while in MIPS32 we need two different names.
adds RE1, RE2, S16	addi RE1, RE2, inm	
mvns RE1, RE2	nor RE1, RE2, RE2	In MIPS32 this instruction does not exist directly so we have to use nor over a same register.
cmp RE1, RE2	beq RE1, RE2, label	The instruction cmp does not exist in MIPS32, and neither does beq S16. However, when combined in ARM, we obtain the equivalent instruction to beq in MIPS32. Therefore, the main disadvantage for ARM is that we need two instructions instead of one to perform this branch.
beq S16		
bl U16	jal label	We see no significant difference between this two instructions.
bx RE	b label jr RE	To use bx in ARM as an equivalent of b in MIPS32, we need to move the address of the label to a register.
halt	li $v0 10, syscall	While ARM has a dedicated instruction, in MIPS32 we need to perform a system call to exit the program.

To test the functionality of this assembly program, we have checked the results of some examples:

Vector	Result (stored in R5 in signed form)
vector: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) length: 10	55
vector: (1, 0, 1, 0, 1) length: 5	3
vector: (-1, -2, -3, -4, -5, -6, -7, -8, -9, -10) length: 10	-55
vector: (0, 0, 0, 0, 0, 0, 0) length: 7	0

Laura Belizón Merchán (100452273)
Jorge Lázaro Ruiz (100452172)
Applied Mathematics and Computing
Group 121

Conclusions and problems found

While we were designing the instructions in Exercise 1, at first we did not understand how storing and accessing data in memory worked, so we had to spend some extra time understanding this correctly.

Furthermore, to do the “translation” for Exercise 2 from MIPS32 to ARM, we had to associate each of our instructions with the ones we already knew in assembly. Since they do not match exactly, we had to look for instructions in the MIPS32 set that were as close as possible to the ones we microprogrammed, which was one of the hardest parts of this assignment. In addition, we had to learn a different parameter convention, as the name and assigned functionality of the registers in ARM and in MIPS32 differ.

To sum up, learning how to microprogram helped us to understand how assembly works on an electronic and circuitry level. It also allows us to understand how the processors in our own computers work, so that we can optimize our own programs.