COMPUTER ARCHITECTURE LAB

STUDENT'S GUIDE

ISA and Pipelining

Goal

Review of relevant characteristics of an ISA of type CISC and RISC

Addressing modes

The addressing modes are described in:

- Notes on the teacher's ISA in BB-Diapos, Slides & Docs
- Hennessy-Patterson, Computer Architecture: A Quantitative Approach, 1st edition: chap. 3rd, 6th edition: appendix A
- Anasagasti, Fundamentals of Computers, 9th edition chap. 6
- Virtually any text on Computer Architecture will describe addressing modes for instruction Instruction Sets

RTL notation

Throughout the course, the notation of transfer between registers (RTL) is used to describe both the elementary steps of executing an instruction and to define the operation performed by a process type instruction (destination ← operating1 operation operand2), transfer type (destination ← source) or control type (PC ← next instruction address) where the Program Counter is PC, and destination and source registers or memory locations. M (X) means the memory content in the X address.
Suppose we must add two numbers a and b and leave the result in c. The three variables are memory locations that contain some integer value. To do this in a processor, depending on the support, the execution model R-R, R-M or M-M would be:

R-R	**R-M**	**M-M**
R1 ← M(a)	R1 ← M(a)	R1 ← M(a) + M(b)
R2 ← M(b)	R1 ← R1 + M(b)	M(c) ← R1
R1 ← R2 + R1	M(c) ← R1	o simplemente:
M(c) ← R1		M(c) ← M(a) + M(b)

To access a memory location pointed to by a register (indirect register mode), say R1, we would write M (R1). For the relative mode with immediate offset, we would write M (R1 + despl).

Using a CISC ISA (Ie 8086)

The following example adds the constant 7 to each element of the nums list in an ISA 8086. In this repertoire, the RB register is used as a pointer and therefore receives the name of the base register. The SI register is also used as an index

to scroll through a list of values from a given address. Observe the RTL notation next to each instruction.

```
.DATA
nums  DB  1,2,3,4,5,6
.CODE
Ini:
MOV AX, @DATA       ; AX ← Dirección .DATA
MOV DS, AX          ; DS ← AX  (segmt Datos)
; ----------------------  Using mode relative with index register SI
MOV  SI, 0          ; SI  ← 0
MOV  CX, 6          ; CX ← 6   (número de elementos)
Bucle:
    ADD nums[SI], 7     ; M(SI+nums)← M(SI+nums) + 7
    INC SI              ; SI ← SI +1
LOOP Bucle              ; CX ← Cx-1 ; Si CX>0 go to bucle


; - - - - - - - - - - - - - - - - - use of mode Register indirect (Reg. base BX)
MOV  CX, 6          ; CX ← 6   (número de elementos)
LEA BX, nums        ; BX ← nums     (DS:BX address of nums)
Bucle:
  MOV DL, [BX]      ; DL← M(BX)
  ADD DL, 7         ; DL ← DL + 7
  MOV [BX], DL      ; M(BX) ← DL
  INC BX            ; BX ← BX + 1
LOOP Bucle          ; CX ← Cx-1 ; If CX>0 go to bucle
```

In other Instruction Sets it would be possible to increase the BX or SI register in the instruction itself in which it is used to access the memory: MOV [BX ++], DL or ADD nums [SI ++], 7, saving the consequent instructions. These addressing modes are called self-incremented and are typical of CISC.

Exercise 1: rewrite the previous loop using auto-incremented modes.

Exercise 2: following the RTL descriptions below, write the corresponding assembler program using the instruction set of a 2-address CISC machine with 8 registers (R1-R8), that supports the R-R and R-M execution modes, and addressing

modes: immediate, register direct, and relative to register using immediate offset (without autoincrement). Assume **ADD** and **MOVE** mnemonics only. For conditional jump instructions use "**BNEZ loop**" (and not LOOP). For each line there must be a single assembler instruction and its corresponding RTL description. You may copy the data declaration section from the 8086 example.

R1 ← 6

R2 ← 0

loop:

R3 ← M(R2+nums)

R3 ← R3 + 7

M(R2+nums) ← R3

R2 ← R2 + 1

R1 ← R1 - 1

Si R1>0 go to loop

Exercise 3: Do the program corresponding to this simple loop in c-like code using the CISC Instruction set specified in exercise 2 and RTL notation:

```
int i=20, s, j;
int a[20] = {1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,20}

j=0;
s=0;
while i>0
  i--;
  s = s + a(j++);
end-while
```