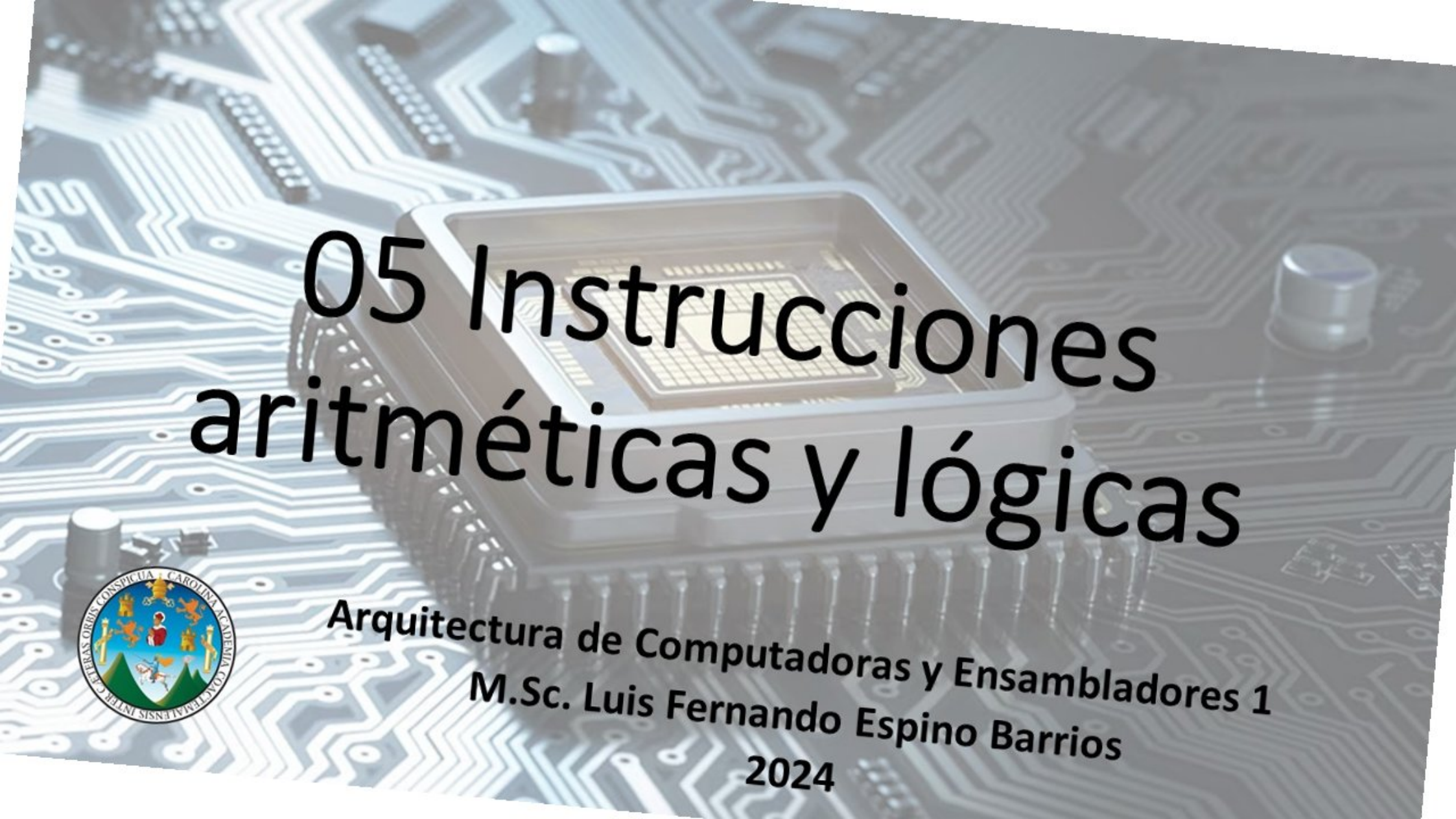
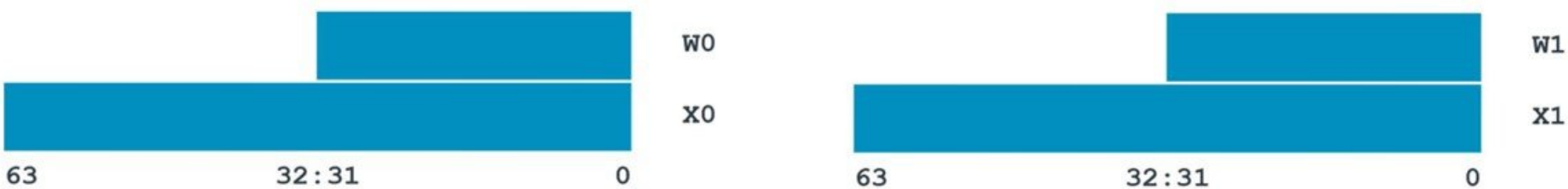


05 Instrucciones aritméticas y lógicas

Arquitectura de Computadoras y Ensambladores 1
M.Sc. Luis Fernando Espino Barrios
2024

Figure 6-1: Register diagram



Instrucciones aritméticas

Instrucción	Descripción
add Rd, Rn, Rm	Suma el operando Rn y Rm dejando el resultado en Rd.
sub Rd, Rn, Rm	Resta el operando Rn con Rm dejando el resultado en Rd.
neg Rd, Rn	Niega el operando Rn y deja el resultado en Rd.

Instrucciones con acarreo (carry)

- Las tres operaciones aritméticas básicas se le puede agregar la letra c a la instrucción y pueden procesar un registro más siendo el carry ya sea de manera positiva o negativa: addc, subc, negc

Instrucciones con set flag

- Las dos operaciones aritméticas básicas se le puede agregar la letra s a la instrucción y con esto activan el uso de las banderas NZCV cuando ocurra una acción: adds, subs

Operaciones de extensión

- Como Arm64 maneja diferentes tamaños de registros es necesario tener instrucciones de extensión entre tamaños para hacer compatibles operaciones entre diferentes registros.

Table 4.3: Extension operations in Operand2.

<extend_op>	Meaning
uxtb	Unsigned extend byte
uxth	Unsigned extend half-word
uxtw	Unsigned extend word
uxtx	Unsigned extend double-word
sxtb	Sign-extend byte
sxth	Sign-extend half-word
sxtw	Sign-extend word
sxtx	Sign-extend double-word
lsl	Logical Shift Left

Ejercicio

- Escribir un programa en Arm64 que solicite una cadena de 3 dígitos y devuelva como valor de retorno la suma de los 3 dígitos ingresados.

```
.global _start

.data
msg:
    .ascii "Enter 3 consecutive digits: "

.bss
input:
    .space 4

.text
_start:
    mov x0, 1       // stdout
    ldr x1, =msg    // load msg
    mov x2, 28      // size msg
    mov x8, 64      // write syscall_num
    svc 0           // syscall
```

```
    mov x0, 0         // stdin
    ldr x1, =input    // load input
    mov x2, 4         // size input with flush
    mov x8, 63        // read syscall_num
    svc 0             // syscall

    ldr x0, =input    // load input
    ldrb w1, [x0]     // load 1st digit
    sub w1, w1, 48    // ascii to num
    add x0, x0, 1     // input offset
    ldrb w2, [x0]     // load 2nd digit
    sub w2, w2, 48    // ascii to num
    add x0, x0, 1     // input offset
    ldrb w3, [x0]     // load 3rd digit
    sub w3, w3, 48    // ascii to num
    add w4, w1, w2    // sum two digit
    add w4, w4, w3    // sum three digit

    uxtb x0, w4       // extend return value
    mov x8, 93        // exit syscal_num
    svc 0             // generic syscall
```

Instrucciones lógicas

Instrucción	Descripción
and Rd, Rn, Rm	AND entre Rn y Rm, resultado en Rd.
bic Rd, Rn, Rm	AND-NOT entre Rn y Rm, resultado en Rd.
eor Rd, Rn, Rm	XOR entre Rn y Rm, resultado en Rd.
eon Rd, Rn, Rm	XOR-NOT entre Rn y Rm, resultado en Rd.
orr Rd, Rn, Rm	OR entre Rn y Rm, resultado en Rd.
orn Rd, Rn, Rm	OR-NOT entre Rn y Rm, resultado en Rd.
mvn Rd, Rm	NOT de Rm, resultado en Rd.

Instrucciones con set flag

- Las dos operaciones lógicas AND y BIC se les puede agregar la letra s a la instrucción y con esto activan el uso de las banderas NZCV cuando ocurra una acción: ands, bics

Ejercicio

- Solicite un dígito decimal, utilice alguna instrucción lógica para determinar si x es un número impar o impar, imprimiendo el resultado.

```
.global _start

.data
msg:
    .ascii "Enter 1 digit: "

.bss
input: .space 2

.text
_start:
    mov x0, 1       // stdout
    ldr x1, =msg    // load msg
    mov x2, 15      // size msg
    mov x8, 64      // write syscall_num
    svc 0           // syscall
```

```
    mov x0, 0       // stdin
    ldr x1, =input  // load input
    mov x2, 2       // size input with flush
    mov x8, 63      // read syscall_num
    svc 0           // syscall

    ldr x0, =input  // load input
    ldrb w1, [x0]   // load 1st digit
    sub w1, w1, 48  // ascii to num
    and w2, w1, 1   // check parity
    add w2, w2, 48  // num to ascii
    strb w2, [x0]   // store 1-odd 0-even

    mov x0, 1       // stdout
    ldr x1, =input  // load msg
    mov x2, 2       // size msg
    mov x8, 64      // write syscall_num
    svc 0           // syscall

    mov x0, 0       // return value
    mov x8, 93      // exit syscal_num
    svc 0           // syscall
```

Bibliografía

Elahi, A. (2022). Fundamentals of Computer Architecture and ARM Assembly Language. Springer.

Harris, S. & Harris, D. (2016). Digital Design and Computer Architecture: ARM Edition. Elsiever Inc.

Mazidi, M. & Otros. (2013). ARM Assembly Language: Programming & Architecture.

Smith, S. (2019). Raspberry Pi Assembly Language Programming: ARM Processor Coding. Apress.