Propuesta de resolución a la colección de problemas de la Sección 2: 1.12, 2.1, 2.2, 2.5, 2.6

Marta Granero I Martí

27 Febrero 2021

TEMA 1:

1.12 FIABILIDAD, DISPONIBILIDAD

Tomamos las horas como unidad de tiempo del MTTF.

a. Calcula el tiempo medio hasta fallos del sistema

Debemos calcular el tiempo medio que tarda el sistema desde que empieza a funcionar hasta que hay algún fallo y deja de funcionar.

Sabemos que el tiempo medio hasta fallo, o $\text{MTTF}_{\text{sist}}$, lo podemos expresar como:

$$\text{MTTF}_{\text{sist}} = \frac{1}{\sum_{\forall comp} \frac{1}{\text{MTTF}_{\text{comp}}}} = \frac{1}{\frac{1}{125\times10^3} + \frac{1}{10^6} + \frac{1}{2\times10^5} + \frac{4}{10^6} + \frac{1}{5\times10^5} + \frac{8}{10^5}} = 10000 \text{ horas}$$

b. El tiempo medio para reemplazar un componente que fa fallado(MTTR) es de 20 horas. Calcula el tiempo medio entre fallos(MTTB)

Por definición sabemos que el tiempo medio entre fallos(MTTB) se obtiene del tiempo necesario para restablecer el sistema(MTTR) más el tiempo que funciona este hasta que falla(MTTF), por tanto:

$$\text{MTTB} = \text{MTTR} + \text{MTTF} \overset{\text{a.}}{=} 10000 + 20 = 10020 \text{ horas}$$

c. ¿Cuál es la disponibilidad del sistema?

El termino disponibilidad del sistema es el cociente entre las horas de cumplimento del servicio entre la suma del estado de realización y incumplimiento del servicio, lo cuantificamos como:

$$\text{disponibilidad} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} \overset{\text{b.}}{=} \frac{10000}{10000 + 20} = 0.998$$

TEMA 2:

2.1 OPERADORES LÓGICOS

Expresión	0b	0x	Expresión	0b	0x
x & y	00000010	02	x && y	00000001	01
x | y	11110111	F7	x || y	00000001	01
∼x | ∼y	11111101	FD	!x || !y	00000000	00
x & !y	00000000	00	x && ∼y	00000001	01

2.2 DESPLAZAMIENTOS

```
print("x << 4: ")
```

```
1                    def shl(dest, count):
2                        return hex(dest << count)
```

```
print("Shift Right Logico(x >> 3): ")
```

```
1                    def sha(dest, count):
2                        return hex(dest >> count)
```

```
print("Shift Right Aritmetico(x >> 3): Replicamos el bit de signo para rellenar los bits restantes, ejemplo: ")
```

```
1                    resultatLogic = "01"
2                    scale = 16
3                    res = bin(int(resultatLogic,scale)).
                         zfill(8)
4                    print("Resultat: ",str(res))
```

Hexadecimal	x << 4	x >> 3(lógico)	x >> 3 (aritmético)
0xF0	0xF00	0x1E	0xFE
0x0F	0xF0	0x01	0x01
0xCC	0xCC0	0x19	0xF9
0x55	0x550	0x0A	0x0A
0x80	0x800	0x10	0xF0
0x02	0x20	0x00	0x00

Binario	x << 4	x >> 3(lógico)	x >> 3 (aritmético)
0xF0 $\Longrightarrow$ 11110000	00000000	00011110	11111110
0x0F $\Longrightarrow$ 00001111	11110000	00000001	00000001
0xCC $\Longrightarrow$ 11001100	11000000	00011001	11111001
0x55 $\Longrightarrow$ 01010101	01010000	00001010	00001010
0x80 $\Longrightarrow$ 10000000	00000000	00010000	11110000
0x02 $\Longrightarrow$ 00000010	00100000	00000000	00000000

2.5 TRADUCCIÓN

Code Listing 1: Traducción a ensamblador

```
        .bss
        .comm A, 256, 1 ;Tenemos dos variables globales: A[256] y tabla[256]
        .comm tabla, 256, 1
        .text
        .globl _main
        _main:
                movl $A, %eax
                movl $tabla, %ebx
                movl $0, %ecx ;i = 0
        _AssigFor:
                cmpl $255, %ecx ;eval condicion
                jle _Endfor
                movsbl (%eax,%ecx), %edx ;D=0, copiamos el valor de la posicion eax+
                    ecx -> edx
                movb (%ebx,%edx),%dl ;ebx+edx -> dl
                movb %dl, (%eax,%ecx) ;dl -> eax+ecx
                incl %ecx
                jmp _AssigFor
        _EndFor:
```

2.6 TRADUCCIÓN

Code Listing 2: Traducción a ensamblador

```
        _main:

        _sorpresa:
                pushl %ebp ;apilamos ebp, ebp := base de la pila de la subrutina
                movl %esp, %ebp ; esp := puntero a la cima de la pila
                movl 8(%ebp), %eax
                movl 12(%ebp), %ebx
        if:
                cmpl $-10, %eax ;comparamos el segundo operando con el primero
                jle else

                cmpl $10, %eax
                jge else

                movl %eax, %ebx
                jmp endif:
        else:
                lea 8(%ebp), %eax ;movemos direccion memoria(8+ebp) -> eax
                movl %eax, 12(%ebp)
        endif:
        _EndS:
                movl 12(%ebp), %ecx
                popl %ebp ;desapilamos el elemento de la cima y lo guardamos en ebp
                popl %eip ;salimos de la subrutina apuntando a la siguiente
                    instruccion
```