Examen Arquitectura de Computadores

Nombre: Respuestas Propuestas

Semestre 2019/2

1. **Sofanor Rodríguez ltda, requiere de su ayuda para diseñar un Multiplexor (4x1) con una serie de compuertas lógicas de bajo consumo y baja latencia. Sofanor Rodríguez ltda es líder en el mercado en el diseño de multiplexores (2x1). Se le pide entonces: (50 pts)**
   a. **Diseñar un circuito combinacional para el multiplexor (2x1). Sea explícito en mostrar tabla de verdad, mapas de Karnaugh, implicantes y diagrama de compuertas lógicas que correspondan al circuito (no hay restricciones en el número y tipo de compuertas) (30 Pts)**

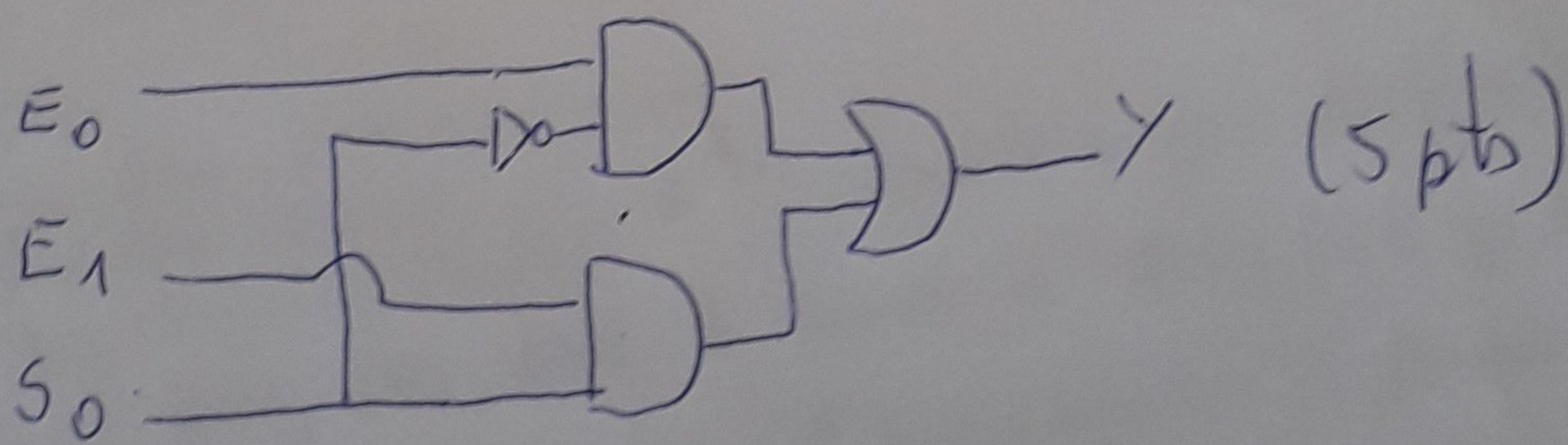
Tabla de Verdad (5pts)

E0	E1	S0	Y
0	X	0	0
1	X	0	1
X	0	1	0
X	1	1	1

Mapas de Karnaugh (10pts)

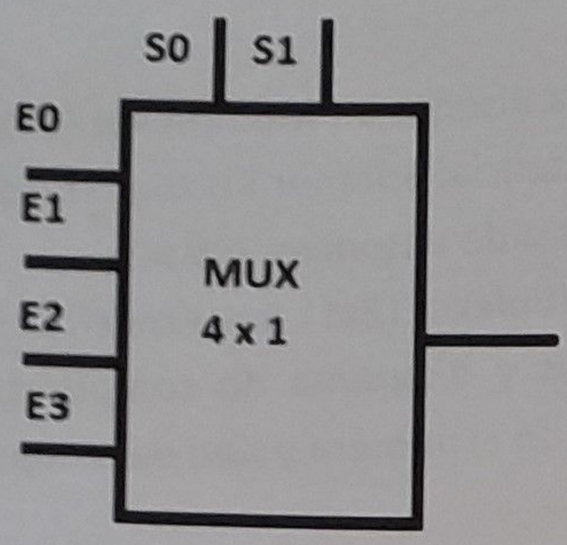
E0 E1 \ S0	0	1
0 0	0	0
0 1	0	1
1 1	1	1
1 0	1	0

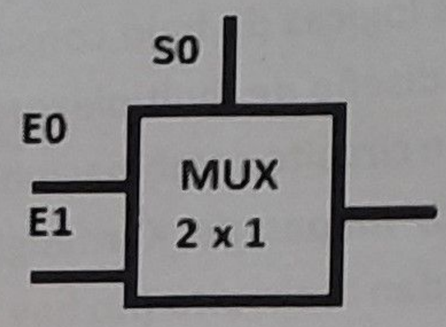
$$Y = E_0 \bar{S}_0 + E_1 S_0 \quad (10\text{pts})$$

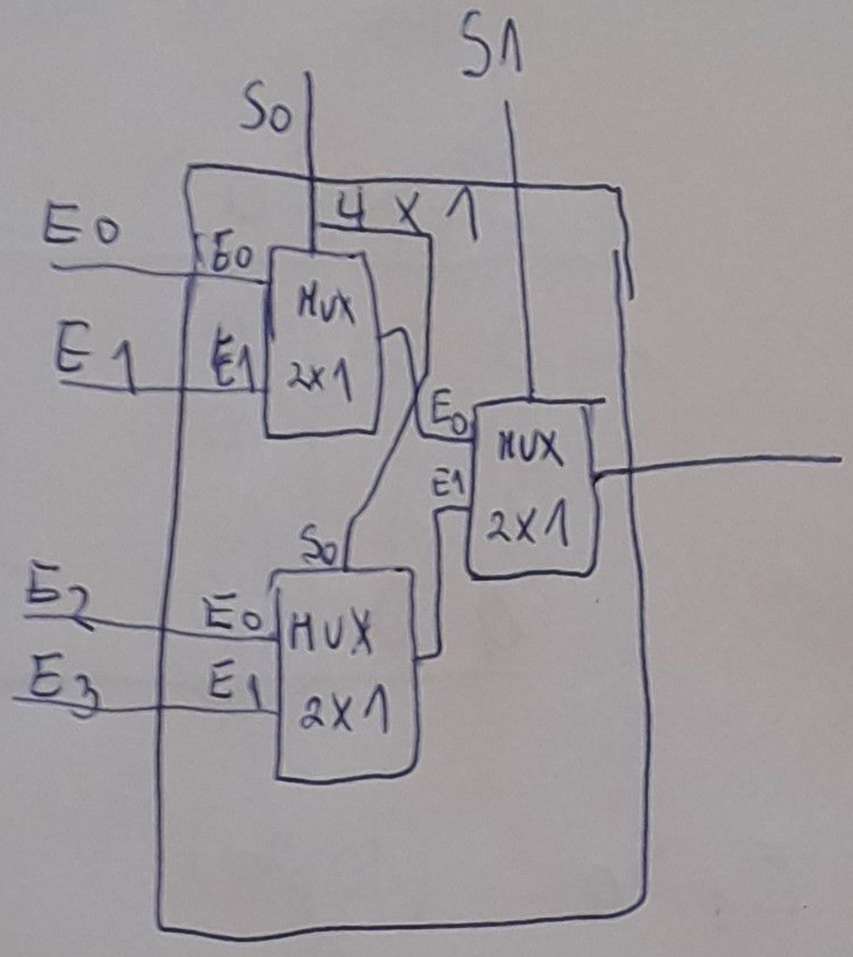


(5pts)

b. Empleando el menor número posible de multiplexores (2x1), diseñe un multiplexor 4x1. (20 pts)







(20 pts)

2. **Asuma el siguiente arreglo con 10 valores enteros al azar (25 Pts):**

```
                .data
Arreglo:        .word   21, 7, 0, 25, 3, 87, 38, 11, 41, 65
len:            .word   10
```

a. **Construya una función en assembler MIPS que calcule el máximo de los valores del arreglo.**

```
                        ← opcional (bonus) →
                        pushpila: sub $SP, $SP, 12      poppila:
                                  sw $a0, 8($SP)          lw $a0, 8($SP)
.text                             sw $a1, 4($SP)          lw $a1, 4($SP)
.globl main                       sw $ra, 0($SP)          lw $ra, 0($SP)
                                  j inicio                j return

main:   la $t1, Arreglo
        la $t2, len
        lw $S1, 0($t2)

        add $a0, $t1, $0   # Colocando direccion del arreglo como argumento
        add $a1, $S1, $0   # Colocando largo arreglo como segundo argumento
        jal maximo
                           # fin programa, ver máximo en $v0
end:    j end

maximo: j pushpila   ~> # respaldo en pila por convención
inicio: add $t0, $t0, $0
        lw $V0, 0($a0)    # 1er elemento arreglo es definido como máximo
repetir: lw $S0, 0($a0)
        beq $t0, $a1, fin_max
        slt $t1, $S0, $V0   # Consulto si $t0 < $a0 => $t1 = 1
        beq $t1, $0, cambiar  # si es verdadero => Nuevo máximo
proximo: add $a0, $a0, 4    # proximo elemento (dirección)
        add $t0, $t0, 1     # actualizar indice
        j repetir
                            #(Nuevo Maximo
                            cambiar: add $V0, $S0, $0
                                     j proximo
fin_max: j poppila  ~> recupera pila
return: jr $ra
```

3. **Considere el siguiente código (25 Pts):**

   a. **Explique brevemente el propósito del siguiente código assembler MIPS. (10 Pts)**

```
        .data
tam:    .word  8
datos:  .word  2, 4, 6, 8, -2 -4, -6 -7
~~res:    .word 0~~

        .text
main:   lw $8, tam($0)
        la $9, datos    //la, carga una dirección de memoria en registro
        addi $11, $0, $0                              suma ≡ $11
loop:   lw $10, 0($9)         → datos[i]
        add $11, $11, $10     suma = suma + datos[i]
        addi $9, $9, 4        → dirección próximo elemento
        addi $8, $8, -1       → un elemento menos que chequear
        beq $8, $0, salir     → si ya no hay más salir
        j loop                → volver
salir:  j salir
```

Suma los elementos del arreglo, el resultado lo deja en $11, el cual es 1

b. Muestre un código C equivalente. (15 Pts)

```
int main() {
    int i = 0;
    int [] datos = {2,4,6,8,-2,-4,-6,-7};
    int term = 8;
    int suma = 0;
    do {
        suma = suma + datos[i];
        i = i++;
        term = term - 1;
    } while (term > 0);
    printf("Suma: ", suma); ← es especial
```

3.