DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

IIC2343 - Arquitectura de Computadores (II/2025)

Guía de ejercicios: Assembly y Saltos

Ayudantes: Daniela Ríos (danielaarp@uc.cl), Alberto Maturana (alberto.maturana@uc.cl), José Mendoza (jfmendoza@uc.cl)

Pregunta 1: Preguntas conceptuales

(a) ¿Cómo se podría implementar en el computador básico una señal para que esta indique luego de realizar una operación cuando el resultado es par o impar?

(b) ¿Qué pasaría si se quita el registro `STATUS` del computador básico y se conectaran directamente las señales `Z`, `N`, `C` y `O` a la unidad de control?

Solución:

(a) Esto se puede hacer de forma muy sencilla si nos damos cuenta del siguiente hecho: Todo número par en representación binaria termina en 0 y todo número impar termina en 1. Esto, ya que todo bit representa una potencia de 2, salvo el último que representa un 1. Por lo tanto, bastaría con añadir a la salida de la `ALU` una señal P equivalente al último bit del resultado (o su negación) y que fuera almacenada dentro del registro `STATUS` de forma directa. Solo faltaría una nueva instrucción asociada a un nuevo *opcode* (que podríamos llamar JEN -*Jump Even Number*- o JON -*Jump Odd Number*-, por ejemplo).

(b) Al no estar conectadas a la señal *clock*, estas señales perderán la sincronización con el flanco de subida. Esto implica que las señales que estén ingresadas en la unidad de control no necesariamente son correspondientes a la última instrucción ejecutada, causando posibles errores en las instrucciones de salto (y por ende, en todo el programa).

Pregunta 2: Explique el código

Detalle, basándose en los nombres de las variables y labels, lo que realizan los siguientes fragmentos de códigos desarrollados en el Assembly del computador básico visto en clases:

(a)

```
DATA:
    n 3
    x_value 6
    f_x 0
CODE:
    loop:
        MOV A,(n)
        CMP A,0
        JEQ end
        SUB A,1
        MOV (n),A
        MOV A,(x_value)
        SHL A,A
        MOV (x_value),A
        JMP loop
end:
    MOV A,(x_value)
    MOV (f_x),A
```

(b)

```
DATA:
    n 8
    x_value 12
    f_x 0
CODE:
    MOV A, (x_value)
    main:
        SHR A, A
        JCR case_2
    case_1:
        MOV B, A
        MOV A, (n)
        SUB A, 1
        CMP A, 0
        JEQ end
        MOV (n), A
        MOV A, B
        JMP main
    case_2:
        INC (f_x)
        JMP case_1
    end:
```

Solución:

(a) En este caso, se observa que se realiza la operación SHL una cantidad n de veces. Si tomamos la variable f_x como $f(x)$, entonces el fragmento de código es equivalente a $f(x) = x \times 2^n$.

(b) En este caso, se observa que se incrementa el valor de f_x cada vez que se realiza el salto por *carry*. Ahora, es importante notar que este salto ocurre cuando la *flag* `C` del registro `STATUS` es igual a 1. Esto no ocurre por un *carry* de la suma, sino por ser el bit descartado en la operación *shift* como se vio en clases. Por lo tanto, si tomamos la variable f_x como $f(x)$, entonces el fragmento de código computa $f(x)$ como la cantidad de bits iguales a 1 que tiene el valor x de 8 bits (establecido a partir de n).

Pregunta 3: Arregle el código

El siguiente programa, desarrollado en el Assembly del computador básico visto en clases, **debiese calcular el aŕea de un triangulo dado las coordenadas de sus tres puntos en el plano**. Este asume que los puntos están ordenados, *i.e.* $P1_x < P2_x$ y $P1_y = P2_y < P3_y$.

```
DATA:
    area     0
    base     0
    altura   0
    P1       6 ; coordenada x del punto P1
             1 ; coordenada y del punto P1
    P2       8 ; coordenada x del punto P2
             1 ; coordenada y del punto P2
    P3       1 ; coordenada x del punto P3
             10 ; coordenada y del punto P3
    contador 0
CODE:
    MOV A, (P2)
    MOV B, (P1)
    SUB (base)
    MOV A, P3
    MOV B, P1
    SUB (altura)
    loop:
        MOV A, (altura)
        MOV B, (contador)
        CMP A, B
        JEQ end
        MOV A, (area)
        MOV B, (base)
        ADD (area)
        INC (contador)
        JMP loop
    end:
        MOV A, (area)
        SHL A, A
        MOV (area), A
```

Sin embargo, no lo hace correctamente. Si compila y ejecuta el código, se dará cuenta que no entrega el resultado esperado. Busque el error y, una vez encontrado, arréglelo para que el fragmento anterior entregue el resultado esperado

Solución: Este código presenta dos problemas fundamentales:

1. La altura se está calculando como la resta de las direcciones de memoria de los puntos P2 y P3, no como la resta de sus alturas.
2. El resultado de la multiplicación final se multiplica por 2 a través de un *shift left* y no se divide por dos a través de un *shift right*.

Con eso en consideración, podemos resolver el problema de la siguiente forma:

```
DATA:
    area     0
    base     0
    altura   0
    P1       6 ; coordenada x del punto P1
             1 ; coordenada y del punto P1
    P2       8 ; coordenada x del punto P2
             1 ; coordenada y del punto P2
    P3       1 ; coordenada x del punto P3
             10 ; coordenada y del punto P3
    contador 0
CODE:
    MOV A,(P2)
    MOV B,(P1)
    SUB (base)
    MOV B,P3
    INC B
    MOV A,(B)
    MOV B, P1
    INC B
    MOV B,(B)
    SUB (altura)
    loop:
        MOV A, (altura)
        MOV B, (contador)
        CMP A, B
        JEQ end
        MOV A,(area)
        MOV B,(base)
        ADD (area)
        INC (contador)
        JMP loop
    end:
        MOV A,(area)
        SHR (area)
```

Pregunta 4: Modificación del computador básico

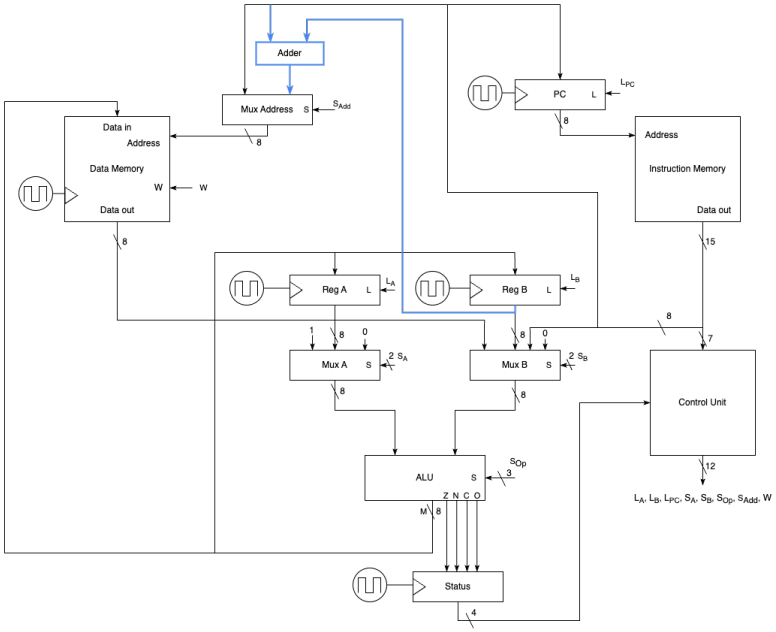
(a) Modifique la arquitectura del computador básico para que el registro `STATUS` se actualice solo después de la ejecución de una instrucción `CMP`.

> **Solución:** La modificación más simple que se puede hacer para lograr el objetivo consiste en crear una nueva señal (llamémosla $\texttt{L}_{stat}$). $\texttt{L}_{stat}$ será la señal que habilita la carga de los datos en el registro `STATUS`. Finalmente, basta con que la `Control Unit` se encargue de transmitir $\texttt{L}_{stat} = 0$ para todo *opcode* que no corresponda a la instrucción `CMP`.

(b) Modifique la arquitectura del computador básico para implementar las instrucciones:

- `MOV A,(B+offset)`
- `MOV (B+offset),A`
- `MOV B,(B+offset)`
- `MOV (B+offset),B`

Es decir, instrucciones de direccionamiento indirecto con registro `B` y *offset*, siendo este último un literal. Para cada instrucción, deberá incluir la combinación completa de señales que la ejecutan. Por cada señal de carga y escritura, deberá indicar si se activan (1) o no (0); en las señales de selección, deberá indicar el nombre de la entrada escogida ("-" si no afecta).

> **Solución:** Para la implementación de estas instrucciones, se puede modificar la conexión entre el registro `B` y el componente `Mux Address` para que ahora reciba el resultado de un sumador que sume el valor de `B` y el literal de la instrucción:
>
> 
>
>
> De esta forma, no es necesario agregar ni modificar señales, pero sí asegurar que el

compilador haga uso del literal 0 para las instrucciones `MOV A,(B)`, `MOV B,(B)`, `MOV (B),A` y `MOV (B),B`. La tabla de señales es como sigue para las instrucciones implementadas:

Instrucción	L_A	L_B	L_{PC}	W	S_A	S_B	S_{OP}	S_{Add}
`MOV A,(B+offset)`	1	0	0	0	ZERO	DOUT	ADD	B+offset
`MOV B,(B+offset)`	0	1	0	0	ZERO	DOUT	ADD	B+offset
`MOV (B+offset),A`	0	0	0	1	A	ZERO	ADD	B+offset
`MOV (B+offset),B`	0	0	0	1	ZERO	B	ADD	B+offset