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

IIC2343 - Arquitectura de Computadores (I/2026)

Interrogación 2 - Parte 1 - Sección 1

Pauta de evaluación

Pregunta 1: Computador básico con subrutinas (6 ptos.)

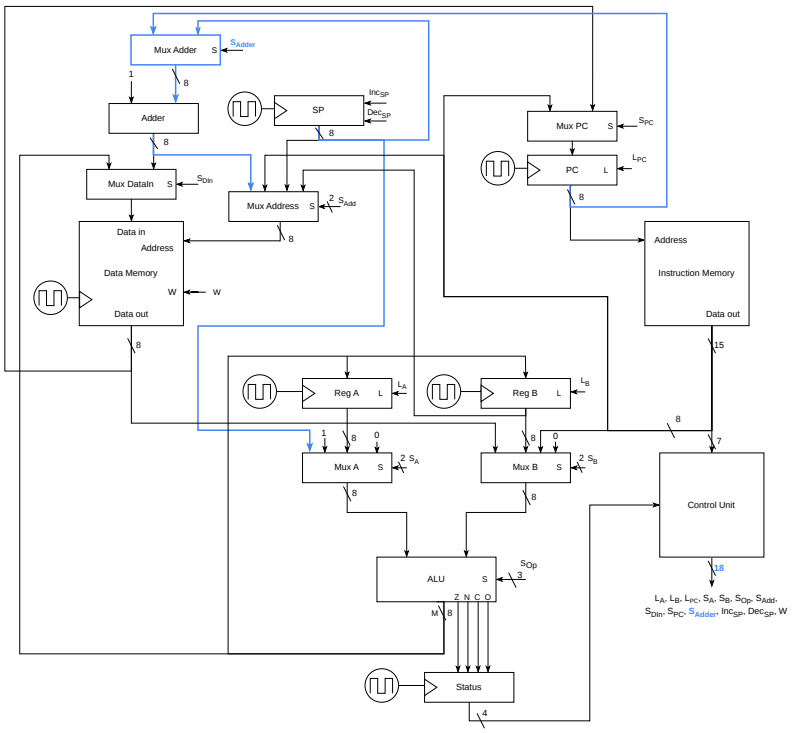
En los ítems **(a)**, **(b)** y **(c)**, se le describirán instrucciones que deberá implementar en el computador básico con subrutinas a partir del diagrama adjunto. Para cada una de ellas, debe incluir la combinación **completa** de señales que las ejecutan. En las señales de carga/escritura/incremento/decremento, debe indicar si se activan (1) o no (0); en las señales de selección, debe indicar el **nombre** de la entrada escogida ("-" si no afecta). Debe realizar todas las modificaciones de *hardware* necesarias en un solo diagrama. Además, puede implementar instrucciones sin modificar el *hardware* si es que la arquitectura se lo permite, indicando la forma de hacerlo a partir de las señales de control. **Cada instrucción debe ejecutarse en un ciclo**.

(a) (1 pto.) `POPRD A`. Almacena el valor del tope del *stack* en el registro `A` sin modificar el *stack pointer*.

(b) (1 pto.) `MOV A, SP`. Almacena en el registro `A` el valor de `SP`.

(c) (1 pto.) `POP1 A`. Extrae el valor del tope del *stack* y lo almacena en el registro `A` **en un solo ciclo**. En otras palabras, es la versión óptima de `POP` al realizar la misma operación en un ciclo y no en dos.

Solución: A continuación, se muestra el diagrama con las conexiones necesarias para ejecutar todas las instrucciones solicitadas, incluyendo una descripción de cada conexión añadida y lo que habilita.



- Se modifica la entrada del *Adder* que computa `PC+1` para añadir un multiplexor `Mux Adder` con señal de selección S_{Adder}. Este selecciona entre el valor de `PC` y el de `SP` para realizar la suma, lo que da la posibilidad de computar `PC+1` o `SP+1`.
- Se conecta la salida del *Adder* con la entrada restante del `Mux Address`, permitiendo seleccionar `PC+1` o `SP+1` (el que nos interesa) como dirección de la memoria de datos.
- Se conecta la salida de `SP` con una entrada de `Mux A`, de forma que se pueda tener este valor como salida de la ALU para su almacenamiento en los registros `A` o `B`.

A continuación, la tabla de señales resultante:

Instrucción	L_A	L_B	L_{PC}	Inc_{SP}	Dec_{SP}	W	S_A	S_B	S_{OP}	S_{Add}	S_{DIn}	S_{PC}	S_{Adder}
`POPRD A`	1	0	0	0	0	0	ZERO	DOUT	ADD	ADDER	-	-	SP
`MOV A, SP`	1	0	0	0	0	0	SP	ZERO	ADD	-	-	-	-
`POP1 A`	1	0	0	1	0	0	ZERO	DOUT	ADD	ADDER	-	-	SP

Por cada instrucción, se otorgan:

- **0.5 ptos.** por *hardware* correctamente modificado; **0.3 ptos.** si existe **un solo error** de implementación; **0.1 ptos.** si existen **solo dos errores**; y **0 ptos.** en otro caso.
- **0.5 ptos.** por entregar una combinación de señales correcta; **0.3 ptos.** si existe **un solo error**; **0.1 ptos.** si existen **solo dos errores**; y **0 ptos.** en otro caso. Si una instrucción se define en más de un ciclo, tampoco se otorga puntaje en este criterio. Si la instrucción se implementa correctamente sin modificación de *hardware*, se otorga el puntaje completo.

(d) (3 ptos.) A continuación, se adjuntan dos fragmentos de código **(1)** y **(2)** escritos en el Assembly del computador básico, con la diferencia de que **(2)** incorpora las instrucciones implementadas en los ítems anteriores. Para cada fragmento, indique los valores de los registros `A` y `B` al finalizar la ejecución del código. Si no incluye desarrollo o no justifica cómo llegó a los valores obtenidos, **no se otorgará puntaje**.

(1)

```
DATA:
CODE:
  MOV A,7     ; Dir. memoria 0x00
  MOV B,1     ; Dir. memoria 0x01
  PUSH B      ; Dir. memoria 0x02
  PUSH A      ; Dir. memoria 0x03
  RET         ; Dir. memoria 0x04-0x05
  MOV A,0     ; Dir. memoria 0x06
  MOV B,0     ; Dir. memoria 0x07
  JMP end     ; Dir. memoria 0x08
end:
```

(2)

```
DATA:
  n    5
  res  0
CODE:
  MOV A,(n)             ; Dir. memoria 0x00
  CALL fun              ; Dir. memoria 0x01
  JMP end               ; Dir. memoria 0x02
  fun:
    CMP A,0             ; Dir. memoria 0x03
    JLE fun_end         ; Dir. memoria 0x04
    PUSH A              ; Dir. memoria 0x05
    SUB A,1             ; Dir. memoria 0x06
    JMP fun             ; Dir. memoria 0x07
    fun_end:
      loop:
        MOV A,SP        ; Dir. memoria 0x08
        CMP A,254       ; Dir. memoria 0x09
        JEQ end_loop    ; Dir. memoria 0x0A
        read:
          POPRD A       ; Dir. memoria 0x0B
          MOV B,A       ; Dir. memoria 0x0C
          AND A,1       ; Dir. memoria 0x0D
          JGT continue  ; Dir. memoria 0x0E
          MOV A,(res)   ; Dir. memoria 0x0F
          ADD (res)     ; Dir. memoria 0x10
        continue:
          POP1 A        ; Dir. memoria 0x11
          JMP loop      ; Dir. memoria 0x12
      end_loop:
        RET             ; Dir. memoria 0x13-0x14
  end:
    MOV A,(res)         ; Dir. memoria 0x15
```

Solución: El fragmento de código **(1)** gatilla la carga del valor numérico del tope del *stack* en el registro `PC` al ejecutar `RET`. Al leer este valor, el programa saltará a la dirección 0x07, saltándose la ejecución de `MOV A,0` (instrucción en la dirección 0x06). De esta forma, el valor final del registro `A` es igual a 7 y el del registro `B` es igual a 0.

El fragmento de código **(2)**, por otro lado, hace lo siguiente: (i) llama a la subrutina `fun` y *pushea* los valores `n`, `n-1`, `n-2`, ..., 1; (ii) revisa el valor del tope del *stack* –con `POPRD A`– y, **si es par** –`AND A,1` < 1–, lo suma a la variable `res`; (iii) independiente de su paridad, se descarta con `POP1` y repite el procedimiento hasta descartar todos los números *pusheados*; (iv) una vez que los revisa todos –`SP` $= 254$, siendo el tope del *stack* la dirección de retorno de `fun`–, retorna y almacena el valor de `res` en `A`. Por ende, `A` es igual a 6 ($4 + 2$, los únicos pares en el rango $[1, 5]$), mientras que `B` es igual a 5, dado que la última vez que se actualiza su valor es con la instrucción `MOV B,A` bajo el *label* `read`, en la iteración donde se lee el último valor *pusheado* al *stack* –correspondiente a `n` $= 5$–.

A continuación, se muestra el resultado de la ejecución para ambos fragmentos:

(1)

```
DATA:
CODE:
  MOV A,7     ; A = 7
  MOV B,1     ; B = 1
  PUSH B      ; MEM[255] = 1, SP = 254
  PUSH A      ; MEM[254] = 7, SP = 253
  RET         ; SP = 254
              ; PC = MEM[254] = 7
  MOV A,0     ; No ejecuta
  MOV B,0     ; RET carga aquí -> B = 0
  JMP end     ; Termina programa
end:          ; A = 7, B = 0
```

(2)

```
DATA:
  n    5
  res  0
CODE:
  MOV A,(n)             ; A = 5
  CALL fun              ; MEM[255] = 0x02
  JMP end               ; Termina el programa
  fun:
    CMP A,0             ; if A <= 0
    JLE fun_end         ;   -> Salta a fun_end
    PUSH A              ; MEM[254] = 5, MEM[253] = 4, ..., MEM[250] = 1 y SP = 249
    SUB A,1             ; A = 4, 3, 2, 1, 0
    JMP fun             ; Repetir hasta que A == 0
    fun_end:
      loop:
        MOV A,SP        ; A = 249, 250, ..., 254
        CMP A,254       ; if A == 254
        JEQ end_loop    ;   -> Salta a end_loop
        read:
          POPRD A       ; A = MEM[250] = 1, MEM[251] = 2, ..., MEM[254] = 5
          MOV B,A       ; B = 1, 2, ..., 5 (valor final de B)
          AND A,1       ; 1 AND 1 = 1, 2 AND 1 = 0, 3 AND 1 = 1, 4 AND 1 = 0, 5 AND 1 = 1
          JGT continue  ; Si A es impar, continúa sin actualizar res
          MOV A,(res)   ; e.o.c. (A par), A = res
          ADD (res)     ; res = A + B -> res = res + B -> res += 2, 4 (suma solo pares)
        continue:
          POP1 A        ; A = MEM[250] y SP = 250, ..., A = MEM[254] y SP = 254
          JMP loop      ; Repetir hasta que SP == 254
      end_loop:
        RET             ; PC = MEM[255] = 0x02 (retorna a JMP end)
  end:
    MOV A,(res)         ; A = res = 2 + 4 = 6; B = 5 (última actualización de MOV B,A)
```

Para cada fragmento, se otorgan **0.75 ptos.** por registro correctamente computado siempre que su cómputo esté respaldado por desarrollo o justificación. Si no se llega a ningún resultado final correcto por **error de arrastre**, se otorgan **0.5 ptos.** en el fragmento.