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

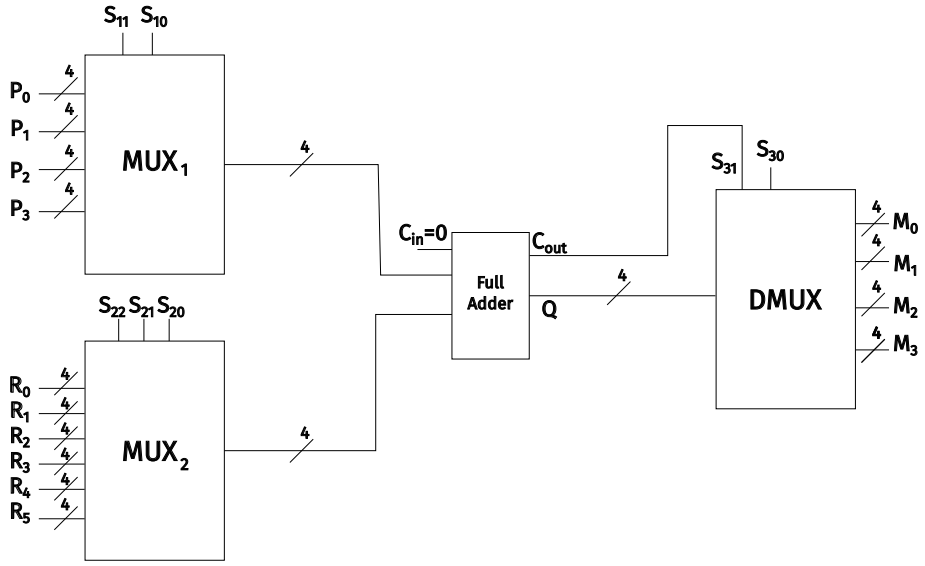
IIC1001 — Algoritmos y Sistemas Computacionales — 2024-1

Interrogación 2

Miércoles 29-Mayo-2024

Duración: 120 minutos
Responder cada pregunta en una hoja separada.

1. **[15p]** Considere el siguiente circuito lógico que incluye los multiplexores MUX_1 (4 entradas), MUX_2 (6 entradas), el demultiplexor DMUX, y un *Full Adder* con dos entradas y salida Q.



En cada entrada se ingresan los siguientes valores hexadecimales positivos: $P_0 = \texttt{0x1}$, $P_1 = \texttt{0x5}$, $P_2 = \texttt{0x8}$, $P_3 = \texttt{0xC}$, $R_0 = \texttt{0x3}$, $R_1 = \texttt{0x4}$, $R_2 = \texttt{0x7}$, $R_3 = \texttt{0x9}$, $R_4 = \texttt{0xA}$, $R_5 = \texttt{0xE}$.

Responda las siguientes preguntas. **Los resultados deben expresarse como hexadecimal**.

a) **[3p]** Si $S_{11}S_{10} = 00$, $S_{22}S_{21}S_{20} = 000$ y $S_{30} = 0$, ¿qué valor se obtiene en cada salida M_i?

R.

Si $S_{11}S_{10} = 00$, entonces la salida de MUX_1 es $P_0 = \texttt{0x1}$.

Si $S_{22}S_{21}S_{20} = 000$, entonces la salida de MUX_2 es $R_0 = \texttt{0x3}$.

El *Full Adder* recibe `0x1` y `0x3` cuya suma es `0x4`. Por lo tanto $Q = \texttt{0x4}$ y $C_{\text{out}} = 0$.

El selector de DMUX es $S_{31}S_{30} = 00$, por lo tanto la salida $M = \{M_0, M_1, M_2, M_3\} = \{\texttt{0x4}, \texttt{0x0}, \texttt{0x0}, \texttt{0x0}\}$

Puntaje

3pts: Salida correcta, indicando **todos** los valores que salen de M_i

2pts: Salida correcta, pero solo indican el `0x4` sin especificar los otras salidas, o bien indican M_0.

1pto: Algún error de cálculo, pero procedimiento mayormente correcto.

0pts: Más de un valor incorrecto, o error en procedimiento.

Si resultado final no está en hexadecimal, máximo 2pts.

b) **[3p]** ¿Qué valores deben tener $S_{11}S_{10}$, $S_{22}S_{21}S_{20}$ y S_{30} para que la suma de P_1 y R_2 salga por M_1? ¿Cuál es el valor que sale por M_1?

R.

Para que MUX_1 escoja P_1, debe tener $S_{11}S_{10} = 01$.

Para que MUX_2 escoja R_2, debe tener $S_{22}S_{21}S_{20} = 010$.

El Full Adder recibe `0x5` y `0x7`. La suma es `0xC`, por lo tanto $Q =$ `0xC` y $C_{\text{out}} = 0$.

Para que DMUX entregue el resultado en M_1, entonces $S_{31}S30 = 01$, por lo tanto $S_{30} = 1$.

El valor que sale por $M_1 =$ `0xC`.

Puntaje

3pts: Todos los valores S_i correctos. Valor que sale por M_1 correcto.

2pts: A lo más un valor de S_i incorrecto, o bien valor de M_1 incorrecto.

1pto: Más de un valor incorrecto, pero con un procedimiento mayormente correcto.

0pts: Más de un valor incorrecto, o error en procedimiento.

Si resultado final no está en hexadecimal, máximo 2pts.

c) **[3p]** ¿Qué valores deben tener $S_{11}S_{10}$ y $S_{22}S_{21}S_{20}$ para que el *Full Adder* calcule la suma de P_2 y R_5? Si $S_{30} = 0$, ¿qué valor se obtiene en cada salida de M_i?

R.

Para que MUX_1 escoja P_2, debe tener $S_{11}S_{10} = 10$.

Para que MUX_2 escoja R_5, debe tener $S_{22}S_{21}S_{20} = 101$.

El Full Adder recibe `0x8` y `0xE`. La suma es `0x16`, por lo tanto $Q =$ `0x6` y $C_{\text{out}} = 1$.

Si $S_{30} = 0$, entonces $S_{31}S_{30} = 10$, por lo tanto se escoge M_2. La salida $M = \{M_0, M_1, M_2, M_3\} = \{$`0x0`, `0x0`, `0x6`, `0x0`$\}$

Puntaje

3pts: Todos los valores de S_i y M_i correctos.

2pts: A lo más un valor de S_i o de M_i incorrecto. Es incorrecto decir que la salida $M_2 =$ `0x16` porque eso requiere 5 bit, y todas las salidas son de 4 bit. La salida del Full Adder Q ya debe ser de 4 bit; por eso es `0x6` y no `0x16`.

1pto: Más de un valor incorrecto, pero con un procedimiento mayormente correcto.

0pts: Más de un valor incorrecto, o error en procedimiento.

Si resultado final no está en hexadecimal, máximo 2pts.

d) **[3p]** Si $S_{11}S_{10} = 11$, y $S_{22}S_{21}S_{20} = 101$, y $S_{30} = 0$, ¿qué valor se obtiene en cada salida M_i?

R.

Si $S_{11}S_{10} = 11$, entonces la salida de MUX_1 es $P_3 =$ `0xC`.

Si $S_{22}S_{21}S_{20} = 101$, entonces la salida de MUX_2 es $R_5 =$ `0xE`.

El *Full Adder* recibe `0xC` y `0xE` cuya suma es `0x1A`. Por lo tanto $Q =$ `0xA` y $C_{\text{out}} = 1$.

El selector de DMUX es $S_{31}S_{30} = 10$, por lo tanto la salida $M = \{M_0, M_1, M_2, M_3\} = \{$`0x0`, `0x0`, `0xA`, `0x0`$\}$

Puntaje

3pts: Salida correcta, indicando **todos** los valores que salen de M_i

2pts: Salida correcta, pero solo indican el `0xA` sin especificar los otras salidas, o bien indican solo M_2, o bien indican el valor como `0x1A`.

1pto: Error de cálculo, pero procedimiento mayormente correcto.

0pts: Más de un valor incorrecto, o error en procedimiento.

Si resultado final no está en hexadecimal, máximo 2pts.

e) **[3p]** Suponga que los valores ingresados en P_i y en R_j están escritos en complemento de 2. Eso significa que el bit de la izquierda indica el signo, y los otros bit indican la magnitud.

Si $S_{11}S_{10} = 01$, $S_{22}S_{21}S_{20} = 100$, y $S_{30} = 1$, ¿qué valor se obtiene en cada salida M_i? Escriba también el valor en base 10 (con signo).

R.

Si $S_{11}S_{10} = 01$, entonces la salida de MUX_1 es $P_1 = \texttt{0x5}$.

Si $S_{22}S_{21}S_{20} = 100$, entonces la salida de MUX_2 es $R_4 = \texttt{0xA}$.

El *Full Adder* recibe `0x5` y `0xA` cuya suma es `0xF`. Por lo tanto $Q = \texttt{0xF}$ y $C_{\text{out}} = 0$.

El selector de DMUX es $S_{31}S_{30} = 01$, por lo tanto la salida $M = \{M_0, M_1, M_2, M_3\} = \{\texttt{0x0}, \texttt{0xF}, \texttt{0x0}, \texttt{0x0}\}$.

Es útil notar que el hecho que se esté usando complemento de 2 no afecta el funcionamiento de los componentes (eso es bueno) porque solo es una representación que usamos para poder trabajar con números con signo. Solo al momento de convertir el valor a un entero en base 10 con signo, nos interesa interpretar la notación en complemento de 2.

Como el valor `0xF` utiliza complemento de 2, el primer bit debe ser el signo. Al hacer la conversión de $\texttt{0xF} = \texttt{0b1111} \rightarrow \texttt{0b0001} = \texttt{0x1}$. Por lo tanto el valor `0xF` escrito en complemento de 2 corresponde al valor $(-1)_{10}$. La respuesta a la salida de M en base 10 es $M = \{0, -1, 0, 0\}$.

Alternativamente si se quiere operar con números en base 10, la suma sería de $\texttt{0x5} = (5)_{10}$, y $\texttt{0xA} = (-6)_{10}$. La suma es $(-1)_{10}$. Para obtener su representación binaria en complemento de 2, tenemos que convertir $\texttt{0b0001} \rightarrow \texttt{0b1111} = \texttt{0xF}$. Obtenemos el mismo resultado.

Puntaje

3pts: Salida correcta, indicando **todos** los valores que salen de M_i en base 10.

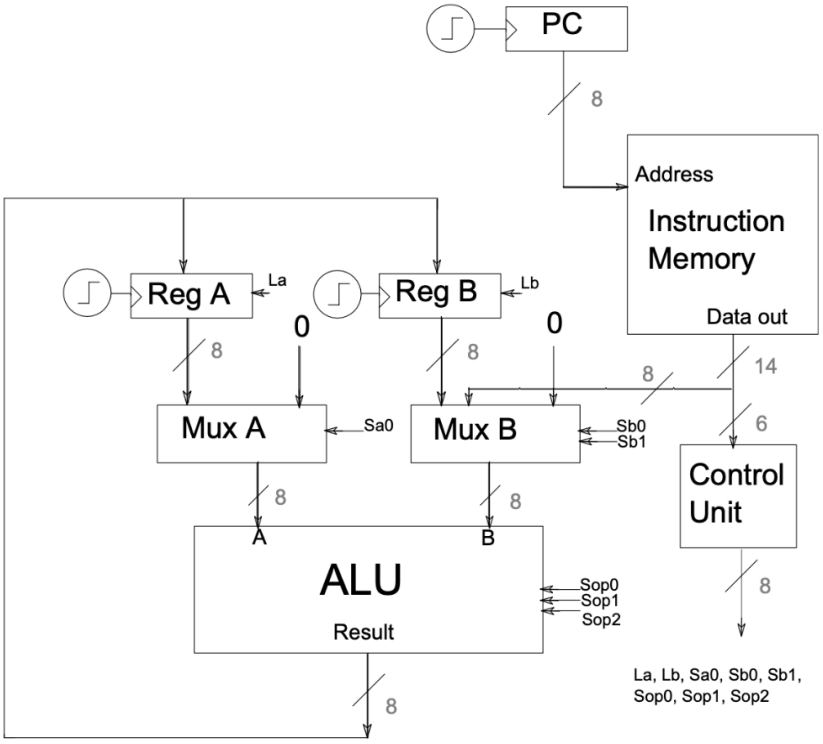
2pts: Salida correcta, pero solo indican el `0xF`, o sólo el -1 sin especificar los otras salidas.

1pto: Algún error de cálculo, pero procedimiento mayormente correcto.

0pts: Más de un valor incorrecto, o error en procedimiento.

Si resultado final no está en hexadecimal, máximo 2pts.

2. **[16p]** Considere la descripción revisada en clase del computador que contiene los registros de 8 bit `RegA` y `RegB`, el Program Counter `PC`, dos multiplexores `MUX A` y `MUX B`, una ALU, una memoria de instrucciones, y una unidad de control.



El contenido de la unidad de control es el siguiente:

Instrucción	Opcode	La	Lb	Sa0	Sb0	Sb1	Sop2	Sop1	Sop0	Operación
MOVAB	000000	1	0	1	0	0	0	0	0	A=B
MOVBA	000001	0	1	0	1	1	0	0	0	B=A
MOVAL	000010	1	0	1	0	1	0	0	0	A=Lit
MOVBL	000011	0	1	1	0	1	0	0	0	B=Lit
ADDA	000100	1	0	0	0	0	0	0	0	A=A+B
ADDB	000101	0	1	0	0	0	0	0	0	B=A+B
ADDL	000110	1	0	0	0	1	0	0	0	A=A+Lit
SUBA	000111	1	0	0	0	0	0	0	1	A=A-B
SUBB	001000	0	1	0	0	0	0	0	1	B=A-B
SUBL	001001	1	0	0	0	1	0	0	1	A=A-Lit
ANDA	001010	1	0	0	0	0	0	1	0	A=A and B
ANDB	001011	0	1	0	0	0	0	1	0	B=A and B
ANDL	001100	1	0	0	0	1	0	1	0	A=A and Lit
ORA	001101	1	0	0	0	0	0	1	1	A=A or B
ORB	001110	0	1	0	0	0	0	1	1	B=A or B
ORL	001111	1	0	0	0	1	0	1	1	A=A or Lit
NOTA	010000	1	0	0	0	0	1	0	0	A=notA
NOTB	010001	0	1	0	0	0	1	0	0	B=notA
XORA	010010	1	0	0	0	0	1	0	1	A=A xor B
XORB	010011	0	1	0	0	0	1	0	1	B=A xor B
XORL	010100	1	0	0	0	1	1	0	1	A=A xor Lit
SHLA	010101	1	0	0	0	0	1	1	0	A=shift left A
SHLB	010110	0	1	0	0	0	1	1	0	B=shift left A
SHRA	010111	1	0	0	0	0	1	1	1	A=shift right A
SHRB	011000	0	1	0	0	0	1	1	1	B=shift right A

Suponga que los registros se inician con los valores `RegA = 0x05` y `RegB = 0x27`. La memoria de instrucciones contiene lo siguiente:

dirección	contenido
0x0	ADDA
0x1	ANDB
0x2	MOVAB
0x3	MOVBL 0x06
0x4	SUBB

Complete la siguiente tabla indicando los valores de los registros antes de ejecutar cada instrucción, los valores de los registros después de ejecutar cada instrucción, y el valor que **sale** de cada multiplexor. **Los valores deben expresarse en hexadecimal**.

RegA inicial	RegB inicial	Instrucción	MUX A	MUX B	RegA final	RegB final
0x05	0x27	ADDA				

R,

Instrucción `0x0: ADDA`. Semántica es $A = A + B$, por lo tanto debería quedar `RegA` $=$ `0x2C`, y `RegB` queda igual. La salida de `MUX A` está dada por $S_a = 0$, por lo tanto `MUXA` $=$ `0x05`. La salida de `MUX B` está dada por $S_{b1}S_{b0} = 00$, por lo tanto `MUXB` $=$ `0x27`. El valor de `RegA` se modifica porque $L_a = 1$, y el valor de `RegB` permanece porque $L_b = 0$.

Puntaje: [2pts]. 0,5 por cada valor correcto. Errores de arrastre se descuentan solo una vez.

RegA inicial	RegB inicial	Instrucción	MUX A	MUX B	RegA final	RegB final
0x05	0x27	ADDA	0x05	0x27	0x2C	0x27

Instrucción `0x1: ANDB`. Semántica es $B = A$ AND B, por lo tanto debería quedar `RegA` sin modificar, y `RegB` $=$ `0x24`. La salida de `MUX A` está dada por $S_a = 0$, por lo tanto `MUXA` $=$ `0x2C`. La salida de `MUX B` está dada por $S_{b1}S_{b0} = 00$, por lo tanto `MUXB` $=$ `0x27`. El valor de `RegA` permanece porque $L_a = 0$, y el valor de `RegB` se modifica porque $L_b = 1$.

Puntaje: [3.5pts]. 0,5 por cada valor correcto. Errores de arrastre se descuentan solo una vez.

RegA inicial	RegB inicial	Instrucción	MUX A	MUX B	RegA final	RegB final
0x05	0x27	ADDA	0x05	0x27	0x2C	0x27
0x2C	0x27	ANDB	0x2C	0x27	0x2C	0x24

Instrucción `0x2: MOVAB`. Semántica es $A = B$, por lo tanto debería quedar `RegA` $=$ `0x24`, y `RegB` sin modificar. La salida de `MUX A` está dada por $S_a = 1$, por lo tanto `MUXA` $=$ `0x00`. La salida de `MUX B` está dada por $S_{b1}S_{b0} = 00$, por lo tanto `MUXB` $=$ `0x24`. El valor de `RegA` se actualiza porque $L_a = 1$, y el valor de `RegB` permanece porque $L_b = 0$.

Puntaje: [3.5pts]. 0,5 por cada valor correcto. Errores de arrastre se descuentan solo una vez.

RegA inicial	RegB inicial	Instrucción	MUX A	MUX B	RegA final	RegB final
0x05	0x27	ADDA	0x05	0x27	0x2C	0x27
0x2C	0x27	ANDB	0x2C	0x27	0x2C	0x24
0x2C	0x24	MOVAB	0x00	0x24	0x24	0x24

Instrucción `0x3: MOVBL 0x06`. Semántica es $B =$ `Lit` donde el literal es `0x06`, por lo tanto debería quedar `RegA` sin modificar, y `RegB` $=$ `0x06`. La salida de `MUX A` está dada por $S_a = 1$, por lo tanto `MUXA` $=$ `0x00`. La salida de `MUX B` está dada por $S_{b1}S_{b0} = 10$, por lo tanto `MUXB` $=$ `0x06`, ya que aquí llegan los 8 bit del literal desde la memoria de instrucciones. El valor de `RegA` permanece porque $L_a = 0$, y el valor de `RegB` se actualiza porque $L_b = 1$.

Puntaje: [3.5pts]. 0,5 por cada valor correcto. Errores de arrastre se descuentan solo una vez.

RegA inicial	RegB inicial	Instrucción	MUX A	MUX B	RegA final	RegB final
0x05	0x27	ADDA	0x05	0x27	0x2C	0x27
0x2C	0x27	ANDB	0x2C	0x27	0x2C	0x24
0x2C	0x24	MOVAB	0x00	0x24	0x24	0x24
0x24	0x24	MOVBL 0x06	0x00	0x06	0x24	0x06

Instrucción `0x4: SUBB`. Semántica es $B = A - B$, por lo tanto debería quedar `RegA` sin modificar, y `RegB` $=$ `0x1E`. La salida de `MUX A` está dada por $S_a = 0$, por lo tanto `MUXA` $=$ `0x24`. La salida de `MUX B` está dada por $S_{b1}S_{b0} = 00$, por lo tanto `MUXB` $=$ `0x06`. El valor de `RegA` permanece porque $L_a = 0$, y el valor de `RegB` se actualiza porque $L_b = 1$.

Puntaje: [3.5pts]. 0,5 por cada valor correcto. Errores de arrastre se descuentan solo una vez.

RegA inicial	RegB inicial	Instrucción	MUX A	MUX B	RegA final	RegB final
0x05	0x27	ADDA	0x05	0x27	0x2C	0x27
0x2C	0x27	ANDB	0x2C	0x27	0x2C	0x24
0x2C	0x24	MOVAB	0x00	0x24	0x24	0x24
0x24	0x24	MOVBL 0x06	0x00	0x06	0x24	0x06
0x24	0x06	SUBB	0x24	0x06	0x24	0x1E

3. **[15p]** Considere un sistema computacional con una CPU, 968 MB RAM, y un sistema operativo multiproceso. Los procesos poseen un número de identificación, un nombre de ejecutable, un tiempo de llegada, una duración y un requerimiento de memoria. Se presentan 5 procesos de acuerdo a la siguiente tabla:

identificador	ejecutable	t_{llegada}	duración	memoria requerida
`P0`	tarea1.exe	0	10	16 MB
`P1`	worst.exe	15	20	128 MB
`P2`	shrome.exe	25	40	512 MB
`P3`	worst.exe	30	5	64 MB

3.1) **[5p] Suponga que todos los procesos caben en memoria**. El sistema utiliza una política de atención "por orden de llegada". Esto significa que los procesos se atienden en el orden en que llegan. Si un proceso llega y hay otro proceso en ejecución, el proceso debe esperar que terminen todos los procesos que estaban antes que él. En $t = 0$ no hay ningún proceso en ejecución, por lo tanto el proceso `P0` espera 0 y empieza de inmediato. ¿Cuánto debe esperar cada uno de los otros procesos **desde que llega** hasta que empieza a ejecutar?

R.

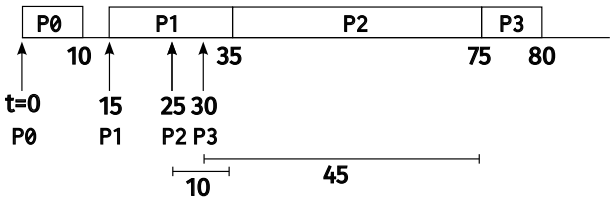
La indicación de que todos caben en memoria indica que no hay que preocuparse de la última columna.

`P0` llega en $t = 0$ y termina en $t = 10$. Espero 0 para ejecutar.

`P1` llega en $t = 15$. Cuando llega no hay ningún proceso en ejecución, por lo tanto esperado 0 para ejecutar. Empieza en $t = 15$ y termina en $t = 35$ ya que dura 20.

`P2` llega en $t = 25$. Cuando llega, está `P1` en ejecución. Tiene que esperar hasta $t = 35$ para ejecutar, por lo tanto espera 10. Empieza en $t = 35$ y termina en $t = 75$ ya que dura 40.

`P3` llega en $t = 30$. Cuando llega, está `P1` en ejecución. Tiene que esperar que termina `P1` y luego que ejecute completamente `P2`, en $t = 75$, por lo tanto espera 45. Empieza en $t = 75$, y termina en $t = 80$.



Puntaje:

3 pts: 1 pto por el tiempo de espera de cada proceso `P1`, `P2`, `P3` calculado correctamente.

1 pto: usa el identificador y no el nombre de ejecutable para referirse a cada proceso, pues hay dos procesos con el mismo ejecutable.

1 pto: cada proceso se atiende en orden de llegada

No es obligatorio hacer el diagrama. Basta calcular los valores correctamente.

3.2) **[5p] Suponga que todos los procesos caben en memoria y que en $t = 0$ todos han llegado y están listos para ejecutar en el orden `P0`, `P1`, `P2`, `P3`**. Ahora el sistema ejecuta cada proceso en turnos de 5 unidades de tiempo. Por ejemplo, `P0` ejecutan durante 5 unidades y luego se va al final de la cola; a continuación `P1` ejecuta durante 5 unidades y se va al final de la cola. Los primeros turnos se cumplen de la siguiente manera:

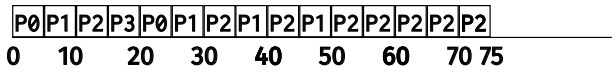
`P0`	`P1`	`P2`	`P3`	`P0`	...

Complete la secuencia de turnos de 5 unidades de tiempo hasta que no queden más procesos.

R.

La indicación de que todos caben en memoria y que se encuentran listos para ejecutarnen $t = 0$ indica que no hay que considerar las columnas de memoria y de t_{llegada}.

Si se asignan 5 unidades de tiempo a cada proceso entonces `P0` necesita 2 turnos para poder terminar su ejecución, `P1` necesita 4, `P2` necesita 8, y `P3` necesita 1.



Puntaje:

4 pts: 1 punto por indicar correctamente, y en orden, los turnos de cada proceso

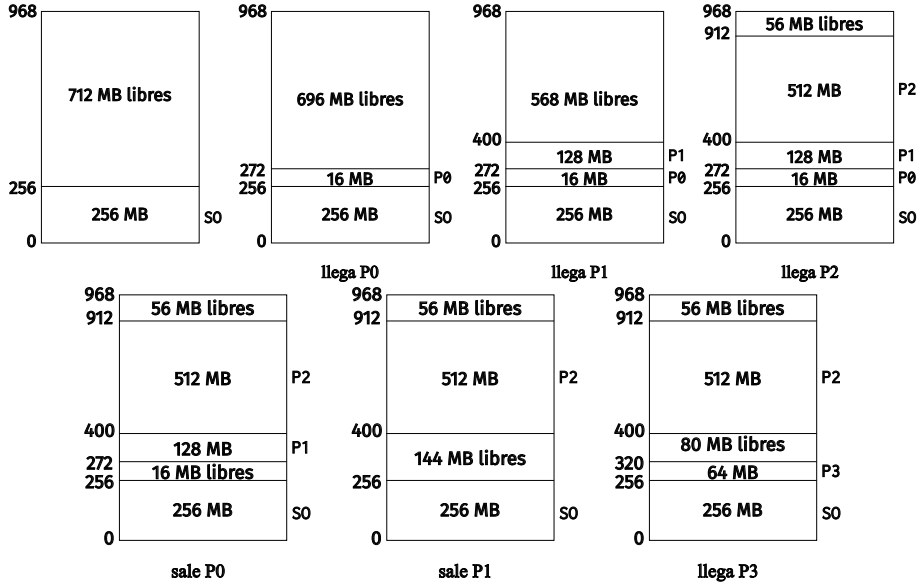
1 pto por considerar turnos seguidos para P2 (y no hacer un turno largo de 25)

3.3) **[5p] Suponga que todos los procesos ejecutan durante 100 unidades de tiempo** El sistema operativo utiliza 256 MB desde la dirección `0x0000`. La asignación de memoria se hace de acuerdo a la política *first-fit* (el primer espacio vacío empezando desde las direcciones de memoria más bajas hacia la más altas). Dibuje la asignación de memoria una vez que ha llegado el último proceso. Si un proceso no cabe en memoria, debe eliminar el que lleva más tiempo en ejecución hasta que haya suficiente espacio. Indique el tamaño de cada proceso y de cada espacio libre en memoria.

R.

La indicación de que todos los procesos ejecutan durante 100 unidades de tiempo es para indicar que no se tenga en cuenta la duración de cada uno.

A continuación se muestra la memoria desde el estado inicial y luego de la llegada de cada proceso.



A medida que los procesos van llegando, se asignan en las direcciones de memoria más bajas disponibles, de acuerdo a la política *first-fit*. Cuando llega el proceso `P3`, no hay espacio contiguo suficiente, por lo tanto se debe sacar a los que llevan más tiempo en memoria, lo cual ocurre en el orden `P0`, `P1`, `P2`. El Sistema Operativo **NO puede salir de memoria** pues siempre debe estar disponible.

Al sacar a `P0`, no queda espacio contiguo suficiente, pero al sacar a `P1` sí se logra que hayan 144MB contiguos disponibles.

Puntaje:

Solo se solicitaba el estado final. Sin embargo, si se muestran estados intermedios y hay errores durante el desarrollo, se puede optar a puntaje parcial.

3 pts: 1 punto por asignar correctamente a `P0`, `P1` y `P2`

1 pto: considerar correctamente la ubicación del sistema operativo, y que éste **NO** debe salir de la memoria

1 pto: asignar correctamente a `P3` incluyendo sacar a `P0` y `P1`.

Errores comunes que descuentan puntaje solo la primera vez (después se corrige de manera consistente): sacar al sistema operativo; mover la ubicación de los procesos que están en memoria para “hacer más espacio”; usar otro criterio para sacar procesos; asignar ubicaciones de acuerdo a otras políticas, o en la posiciones más altas de la memoria.

4. **[14p]** Para cada uno de los siguiente algoritmos, determine su complejidad **usando notación asintótica**. Justifique brevemente con palabras por qué se llega a cada resultado.

   No es obligatorio detallar todo el proceso de cálculo, pero si lo hace puede obtener puntaje parcial en caso de errores.

   Para todos los casos, `A` es una lista de enteros, y n es la cantidad de elementos en la lista (`len(A)`).

   4.1) **[2p]**

```python
s = 0
for i in range(len(A)):
    s = s * 40
```

   R.

   Complejidad: $O(n)$.

   Parcialmente correcto: $O(cn)$, $O(cn + 1)$, con c constante.

   Explicación: se realiza un ciclo con n iteraciones.

   4.2) **[2p]**

```python
s = 0
for i in range(5000000):
    s = s + 1
```

   R.

   Complejidad: $O(1)$

   Parcialmente correcto: $O(5000000)$, $O(c)$, con c constante $\neq 1$.

   Explicación: se ejecutan siempre la misma cantidad de iteraciones (una cantidad constante). No depende del tamaño de `A`.

   4.3) **[2p]**

```python
s = 0
for i in range(5000000):
    s = s + 1
for i in range(len(A)):
    s = -s
    s += 2*A[i]
    print("Suma parcial", s)
```

   R.

   Complejidad: $O(n)$

   Parcialmente correcto: $O(cn + d)$, con c y d constantes.

   Explicación: hay un ciclo que se ejecuta una cantidad constante, y un ciclo que se ejecutan n veces. La cantidad de instrucciones dentro del ciclo es constante.

   4.4) **[2p]**

```python
s = 0
for i in range(len(A)):
    for j in range(len(A)):
        s += A[i] * A[j]
```

   R.

   Complejidad: $O(n^2)$

   Parcialmente correcto: $O(cn^2 + dn + e)$, con c, d, e constantes.

   Explicación: hay un ciclo que se ejecuta n veces y cada iteración incluye otro ciclo que también se ejecuta n veces.

4.5) **[2p]**

```
s = 0
for i in range(len(A)/2, len(A)):
    if s % 2 == 0:
        s += A[i]
    else:
        s -= A[i]
```

R.

Complejidad: $O(n)$

Parcialmente correcto: $O(cn + d)$, con c, d constantes, $c \neq 1$, por ejemplo $O(n/2)$. Es incorrecto indicar que la complejidad depende lo que pasa en los `if/else`.

Explicación: hay un ciclo que se ejecuta $n/2$ veces y cada iteración incluye una cantidad constante de pasos. Al ser $n/2$ iteraciones, igualmente depende de manera lineal de n por lo tanto la complejidad es $O(n)$ (y no $O(n/2)$).

4.6) **[2p]**

```
s = 0
d = 1
while len(A)/2**d >= 1:
    for j in range(0, len(A)):
        s += A[j] * A[j]
    d += 1
```

Complejidad: $O(n \log n)$. También es correcto explicitar una base constante para el logaritmo. Por ejemplo $O(n \log_2 n)$.

Parcialmente correcto: $O(cn \log n + dn + e \log n + f)$, con c, d, e, f constantes. Es incorrecto decir $O(n)$ o con alguna constante, o algo de grado menor.

Explicación: en cada iteración del `while`, el valor de `len(A)/2**d` se reduce a la mitad por lo tanto la cantidad de veces que se itera depende de $\log_2 n$. En cada una de estas iteraciones, se ejecuta un `for` que itera n veces. Por lo tanto la cantidad total es $n \log_2 n$.

4.7) **[2p]**

```
s = 0
for i in range(len(A)):
    for j in range(3*len(A)):
        for k in range(len(A)):
            for m in range(2*len(A)):
                s += A[i]
```

Complejidad: $O(n^4)$.

Parcialmente correcto: $O(cn^4 + dn^3 + en^2 + fn + g)$, con c, d, e, f, g constantes.

Explicación: se trata de 4 ciclos anidados, donde cada uno itera sobre variables independientes entre sí, y cada uno itera una cantidad de de veces proporcional a n.

Puntaje:

Se acepta si usan la notación $\Theta()$ para indicar complejidad. No es correcto usar la notación $\Omega()$. Tampoco es correcto no usar ninguna notación (solo mostrar la función). Si no usan ninguna notación, máximo 1 punto en la pregunta.

Si la función final no es la correcta, pero es una parcialmente correcta, se considera como máximo 1 pto. en la pregunta.

1 punto: Notación asintótica correcta

1 punto: Explicación breve y satisfactoria con palabras. No se solicita un cálculo formal o línea por línea. Si no es muy claro pero tiene una idea correcta, se puede considerar 0,5.