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

IIC2343 - Arquitectura de Computadores (I/2023)

Tarea 4

Pauta de evaluación

Consideraciones

- Respuestas sin desarrollo o justificación no tendrán puntaje.
- Cada pregunta podría tener más de un desarrollo válido. La pauta evalúa eso y este documento solo muestra una alternativa de solución.
- Cualquier detección de infracción al código de honor será sancionada.

Código de Honor de la UC

"Como miembro de la comunidad de la Pontificia Universidad Católica de Chile me comprometo a respetar los principios y normativas que la rigen. Asimismo, prometo actuar con rectitud y honestidad en las relaciones con los demás integrantes de la comunidad y en la realización de todo trabajo, particularmente en aquellas actividades vinculadas a la docencia, el aprendizaje y la creación, difusión y transferencia del conocimiento. Además, velaré por la integridad de las personas y cuidaré los bienes de la Universidad."

Pregunta 1: Memoria caché (30 ptos.)

Para esta parte, deberá implementar en Python 3.x un **simulador de accesos a memoria caché**. Este programa recibirá un archivo `.txt` como *input* a través de la línea de comandos y tendrá el siguiente formato:

```
CACHE_SIZE
LINE_SIZE
MEM_SIZE
CORR_FUNCTION
ACCESS_SEQUENCE
```

Donde cada variable indica lo siguiente:

- `CACHE_SIZE`: Tamaño **en bytes** de la memoria caché.
- `LINE_SIZE`: Tamaño **en bytes** de una línea de la caché.
- `MEM_SIZE`: Tamaño **en bytes** de la memoria principal.
- `CORR_FUNCTION`: Función de correspondencia a aplicar. Puede tener uno de los siguientes valores: `DM` (*Directly Mapped*), `FA` (*Fully Associative*), `2W` (2-*Way associative*).
- `ACCESS_SEQUENCE`: Secuencia de direcciones de memoria accedidas. Corresponde a una secuencia de números separados por una coma.

Un ejemplo concreto de *input* es el siguiente:

```
8
2
16
2W
0, 1, 5, 7, 10, 13, 4, 6
```

Este ejemplo representa una memoria caché de 8 bytes con líneas de 2 bytes (por ende, 4 líneas); una memoria principal de 16 bytes, una función de correspondencia 2-*way associative* y una secuencia de accesos a las direcciones 0, 1, 5, 7, 10, 13, 4, 6.

En respuesta, su programa deberá imprimir:

- La secuencia de accesos, con una letra al lado de cada número: una `H` si hubo *hit* y una `M` si hubo *miss*.
- *Hit rate* total de la memoria caché.
- Estado final de la memoria caché, incluyendo para cada línea: Índice de conjunto en bits (si corresponde), Índice de línea en bits, *tag* del contenido en bits y el bit de validez.

La impresión debe seguir el formato señalado a continuación:

```
0M, 1H, 5M, 7M, 10M, 13M, 4H, 6H
HR=0.375
ÍNDICE CONJUNTO | ÍNDICE LÍNEA | TAG | BIT VALIDEZ
0 | 00 | 11 | 1
0 | 01 | 01 | 1
1 | 10 | 01 | 1
1 | 11 | 10 | 1
```

Este es el *output* esperado para el ejemplo anterior. Ahora, se muestra el *input* y *output* esperado para el mismo ejemplo, pero con funciones de correspondencia *directly mapped* y *fully associative*:

```
# Input Directly Mapped
8
2
16
DM
0, 1, 5, 7, 10, 13, 4, 6
# Output
0M, 1H, 5M, 7M, 10M, 13M, 4M, 6H
HR=0.25
ÍNDICE LÍNEA | TAG | BIT VALIDEZ
00 | 0 | 1
01 | 1 | 1
10 | 0 | 1
11 | 0 | 1
```

```
# Input Fully Associative
8
2
16
FA
0, 1, 5, 7, 10, 13, 4, 6
# Output
0M, 1H, 5M, 7M, 10M, 13M, 4H, 6H
HR=0.375
ÍNDICE LÍNEA | TAG | BIT VALIDEZ
00 | 110 | 1
01 | 010 | 1
10 | 011 | 1
11 | 101 | 1
```

Notar que en estos ejemplos no se incluye el índice de conjunto al no ser utilizado en estas funciones. Adicionalmente, debe realizar los siguientes supuestos:

- Todas las líneas de la caché parten siendo inválidas (*i.e.* su bit de validez parte en 0).
- Todos los tamaños en bytes serán potencias de dos.
- Si corresponde, las cachés utilizan una política de reemplazo `LRU`.
- Todas las direcciones de la secuencia son válidas para la configuración de memoria entregada.
- El formato del archivo de *input* siempre será válido.
- No es necesario redondear ni truncar el *hit-rate*.
- Si una línea de la caché termina inválida, puede imprimir el símbolo "-" en lugar del *tag*.
- Solo se debe simular los accesos de memoria y copia de bloques de datos a través del almacenamiento del *tag*, no se debe simular el contenido "real" de la caché a partir de una memoria "real".
- No se utilizarán rutas de archivos que no existan para probar su tarea.

Distribución de puntaje: Para evaluar su programa, se utilizarán 2 configuraciones de memoria y secuencias de acceso distintas para probar cada función de correspondencia y el funcionamiento general de su simulación, es decir, se aplicarán 6 *tests* en total. Por cada *test* se entregará puntaje de la siguiente forma:

- **5 puntos** si el *test* pasa correctamente.
- **3 puntos** si el *test* presenta un error en la secuencia de accesos, *hit rate* o en el estado final de la caché.
- **1 punto** si presenta dos errores.
- **0 puntos** si presenta más de dos errores o si no corre.

Solución: Se prueban las implementaciones con los siguientes *tests* y *outputs* esperados:

```
# Input Test 1
32
4
64
DM/FA/2W   # 1 input por función
0, 49, 19, 34, 2, 51, 16, 35
```

```
# Output Test 1 DM
0M, 49M, 19M, 34M, 2M, 51M, 16M, 35M
HR=0.0
ÍNDICE LÍNEA | TAG | BIT VALIDEZ
000 | 1 | 1
001 | - | 0
010 | - | 0
011 | - | 0
100 | 0 | 1
101 | - | 0
110 | - | 0
111 | - | 0
```

```
# Output Test 1 FA
0M, 49M, 19M, 34M, 2H, 51H, 16H, 35H
HR=0.5
ÍNDICE LÍNEA | TAG | BIT VALIDEZ
000 | 0000 | 1
001 | 1100 | 1
010 | 0100 | 1
011 | 1000 | 1
100 | ---- | 0
101 | ---- | 0
110 | ---- | 0
111 | ---- | 0
```

```
# Output Test 1 2W
0M, 49M, 19M, 34M, 2M, 51M, 16M, 35M
HR=0.0
ÍNDICE CONJUNTO | ÍNDICE LÍNEA | TAG | BIT VALIDEZ
00 | 000 | 01 | 1
00 | 001 | 10 | 1
01 | 010 | -- | 0
01 | 011 | -- | 0
10 | 100 | -- | 0
10 | 101 | -- | 0
11 | 110 | -- | 0
11 | 111 | -- | 0
```

```
# Input Test 2
128
16
1024
DM/FA/2W   # 1 input por función
263, 951, 848, 540, 22, 653, 220, 1013, 503, 119, 238, 665, 879, 661, 250, 199, 762, 311, 220, 628,
619, 163, 2, 941, 443, 36, 544, 864, 948, 640, 896, 255, 926, 691, 754, 294, 689, 83, 632, 160, 28,
928, 503, 712, 95, 462, 534, 83, 906, 178, 346, 696, 2, 190, 92, 862, 0, 547, 609, 613, 268, 120, 302,
20, 615, 530, 1011, 97, 293, 868, 23, 17, 962, 575, 948, 670, 1018, 195, 148, 888, 872, 466, 601, 831,
633, 820, 394, 370, 202, 331, 943, 617, 263, 419, 699, 320, 633, 169, 564, 911
```

```
# Output Test 2 DM
263M, 951M, 848M, 540M, 22M, 653M, 220M, 1013M, 503M, 119M, 238M, 665M, 879M, 661H, 250M, 199M, 762M,
311M, 220H, 628M, 619M, 163M, 2M, 941M, 443M, 36M, 544M, 864M, 948M, 640M, 896M, 255M, 926M, 691M, 754M,
294M, 689H, 83M, 632M, 160M, 28M, 928M, 503M, 712M, 95H, 462M, 534M, 83H, 906H, 178M, 346M, 696M, 2M,
190M, 92M, 862M, 0H, 547M, 609M, 613H, 268M, 120M, 302M, 20M, 615H, 530M, 1011M, 97M, 293H, 868M, 23M,
17H, 962M, 575M, 948M, 670M, 1018H, 195M, 148M, 888M, 872H, 466M, 601M, 831M, 633M, 820H, 394M, 370M,
202H, 331M, 943M, 617M, 263M, 419M, 699M, 320H, 633M, 169M, 564M, 911M
HR=0.16
ÍNDICE LÍNEA | TAG | BIT VALIDEZ
000 | 111 | 1
001 | 001 | 1
010 | 001 | 1
011 | 100 | 1
100 | 010 | 1
101 | 100 | 1
110 | 100 | 1
111 | 100 | 1
```

```
# Output Test 2 FA
263M, 951M, 848M, 540M, 22M, 653M, 220M, 1013M, 503M, 119M, 238M, 665M, 879M, 661H, 250M, 199M, 762M,
311M, 220M, 628M, 619M, 163M, 2M, 941M, 443M, 36M, 544M, 864M, 948M, 640M, 896M, 255M, 926M, 691M, 754M,
294M, 689H, 83M, 632M, 160M, 28M, 928M, 503M, 712M, 95H, 462M, 534M, 83H, 906M, 178M, 346M, 696M, 2M,
190H, 92H, 862M, 0H, 547M, 609M, 613H, 268M, 120M, 302M, 20M, 615H, 530M, 1011M, 97M, 293H, 868M, 23H,
17H, 962M, 575M, 948M, 670M, 1018M, 195M, 148M, 888M, 872M, 466M, 601M, 831M, 633M, 820H, 394M, 370M,
202M, 331M, 943M, 617M, 263M, 419M, 699M, 320H, 633M, 169M, 564M, 911M
HR=0.14
ÍNDICE LÍNEA | TAG | BIT VALIDEZ
000 | 111000 | 1
001 | 010100 | 1
010 | 011010 | 1
011 | 010000 | 1
100 | 001010 | 1
101 | 100011 | 1
110 | 101011 | 1
111 | 100111 | 1
```

```
# Output Test 2 2W
263M, 951M, 848M, 540M, 22M, 653M, 220M, 1013M, 503M, 119M, 238M, 665M, 879M, 661H, 250M, 199M, 762M,
311M, 220H, 628M, 619M, 163M, 2M, 941M, 443M, 36M, 544M, 864M, 948M, 640M, 896M, 255M, 926M, 691M, 754M,
294M, 689H, 83M, 632M, 160M, 28M, 928M, 503M, 712M, 95H, 462M, 534M, 83H, 906M, 178M, 346M, 696M, 2M,
190H, 92H, 862M, 0H, 547M, 609M, 613H, 268M, 120M, 302M, 20M, 615H, 530M, 1011M, 97M, 293M, 868M, 23H,
17H, 962M, 575M, 948M, 670M, 1018M, 195M, 148M, 888M, 872H, 466M, 601M, 831M, 633M, 820H, 394M, 370M,
202H, 331M, 943M, 617M, 263M, 419M, 699M, 320H, 633M, 169M, 564M, 911M
HR=0.16
ÍNDICE CONJUNTO | ÍNDICE LÍNEA | TAG | BIT VALIDEZ
00 | 000 | 1110 | 1
00 | 001 | 0101 | 1
01 | 010 | 0111 | 1
01 | 011 | 1001 | 1
10 | 100 | 0110 | 1
10 | 101 | 0010 | 1
11 | 110 | 1001 | 1
11 | 111 | 1000 | 1
```

Pregunta 2: Multiprogramación (15 ptos.)

Considere un sistema con páginas de 16KB, direcciones virtuales de 48 bits y direcciones físicas de 30 bits.

- Indique, en bytes, la cantidad máxima de memoria que puede direccionar cada proceso. **(1 punto)**

  Solución: La cantidad máxima de memoria direccionable por proceso está dada por la cantidad de memoria **virtual** que posee el sistema. Asumiendo que cada dirección almacena una palabra de memoria de un byte, se tienen entonces 2^{48} direcciones virtuales y 2^{48}B = 256TB de memoria direccionable. Se otorga el puntaje solo si la respuesta coincide de forma exacta o si se calcula asumiendo 1000B = 1KB.

- Indique, en bytes, la cantidad máxima de memoria física que puede direccionar el sistema. **(1 punto)**

  Solución: La cantidad máxima de memoria direccionable del sistema está dada por la cantidad de memoria **física** que posee. Asumiendo que cada dirección almacena una palabra de memoria de un byte, se tienen entonces 2^{30} direcciones físicas y 2^{30}B = 1GB de memoria direccionable. Se otorga el puntaje solo si la respuesta coincide de forma exacta o si se calcula asumiendo 1000B = 1KB.

- Calcule el tamaño, en bytes, de una tabla de páginas de 1 nivel para este sistema, considerando 4 bits para *metadata*. **(2 puntos)**

  Solución: El tamaño de la tabla de páginas es igual a la cantidad de entradas multiplicada por el ancho, en bytes, de cada una de ellas. La cantidad de entradas es igual a la cantidad de páginas que existan, lo que se determina a partir de los bits de dirección virtual y el tamaño de página. Como el tamaño de una página es de 16KB, se tienen $\log_2(2^{14}) = 14$ bits de *offset* y $48 - 14 = 34$ bits de número de página, lo que deriva en un total de 2^{34} páginas. El ancho de cada entrada es igual a la suma de los bits de número de marco físico y los bits de *metadata*. Como se tienen en total $30 - 14 = 16$ bits de marco físico (ya que el tamaño de los marcos es igual al de las páginas), se tienen por entrada $16 + 4 = 20$ bits, lo que es igual a 2,5B. Finalmente, el tamaño de la tabla de páginas es igual a $2^{34} * 2{,}5$B = 40GB. Se otorgan **0.5 ptos.** por el cálculo correcto de bits de *offset*, **0.5 ptos.** por el cálculo correcto de bits de número de página; **0.5 ptos.** por el cálculo correcto de bits de entrada de la tabla de páginas y **0.5 ptos.** por el cálculo correcto del tamaño de la tabla de páginas, ya sea de forma exacta o si se calcula asumiendo 1000B = 1KB.

- Proponga una modificación en el esquema de paginación que asegure que una tabla de páginas **quepa** en una página. **(6 puntos)**. Se otorgarán **4 puntos** si su propuesta presenta como máximo un error de implementación; **2 puntos** si posee al menos dos errores de implementación; y **0 puntos** si presenta más de dos errores o si no responde.

  Solución: Existen dos alternativas posibles para cumplir con lo debido. La primera alternativa consiste en aumentar el tamaño de página hasta que su tamaño sea mayor o igual al de la tabla de páginas resultante: Si el tamaño de página aumenta a 32MB, entonces se tienen $\log_2(2^{25}) = 25$ bits de *offset*; $48 - 25 = 23$ bits de número de página; y $30 - 25 = 5$ bits de número de marco, lo que deriva en un tamaño de tabla de páginas igual a $2^{23} * 9\text{b} = 9\text{MB}$, cumpliendo lo solicitado. La segunda alternativa consiste en hacer uso de un esquema de paginación multinivel: Si los bits de número de página se distribuyen de forma "equitativa" para la implementación de 3 niveles (11 bits para los primeros dos niveles y 12 bits para el tercer nivel), las tablas de primer y segundo nivel tendrán un tamaño igual a $2^{11} * 30\text{b} = 7.5\text{KB}$ (teniendo cada entrada 30 bits por contener la dirección física de la tabla del siguiente nivel); mientras que la tabla de tercer nivel tendrá un tamaño igual a $2^{12} * 2{,}5\text{B} = 10\text{KB}$, cumpliendo lo solicitado para todos los niveles. Se otorga el puntaje según lo descrito por enunciado, considerando también como válidos cálculos que asuman 1000B = 1KB.

- Asuma que ahora se añade al esquema una TLB (*Translation Lookaside Buffer*) de 256 entradas. Indique la cantidad mínima de bits que requiere cada entrada para poder ser utilizada para realizar traducciones de memoria virtual a física **(3 puntos)** y el tamaño mínimo de esta memoria caché **(2 puntos)**. Se otorgarán **2 puntos** en total si presenta valores errados pero **cercanos** al esperado.

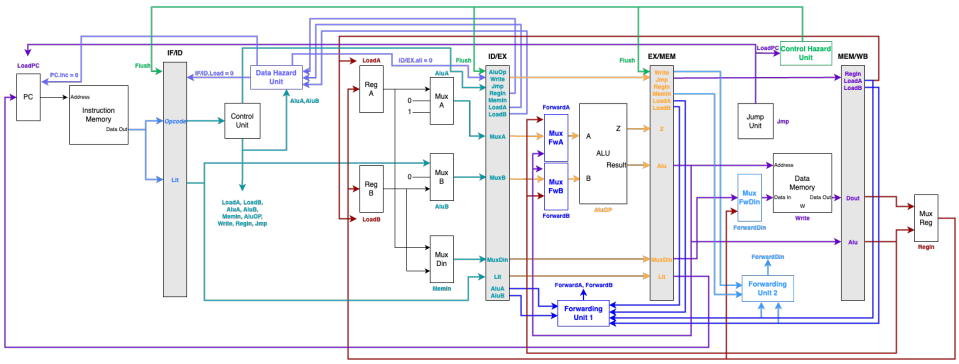
  Solución: En la TLB, cada entrada posee como *tag* los bits de número de página y como contenido los bits de número de marco para realizar la traducción. Por otra parte, como la TLB es una caché, debe tener **al menos** un bit de validez en cada entrada para indicar que su contenido es válido. Por lo tanto, en este esquema se tiene un total de $34 + 16 + 1 = 51$ bits por entrada de la TLB y un tamaño igual a $2^8 * 51\text{b} = 1632\text{B} = 1.59375\text{KB}$. Se considera como "cercana" cualquier respuesta que considere **al menos** los bits de número de página y marco físico, pero no el bit de validez de una entrada de caché. También se asumen correctas respuestas que incluyan en el cálculo los bits de *metadata* de las entradas de la tabla de páginas, pero estos no son necesarios.

Para todos los ítems, **debe incluir la fórmula o explicación de cómo obtuvo el valor numérico**. Si solo incluye el tamaño, no se le otorgará puntaje en la pregunta. Por otra parte, puede utilizar las siguientes transformaciones numéricas para simplificar sus respuestas:

- 8 bits = 8b = 1 byte = 1B
- $2^{10}\text{B} = 1\text{KB}$
- $2^{20}\text{B} = 2^{10}\text{KB} = 1\text{MB}$
- $2^{30}\text{B} = 2^{10}\text{MB} = 1\text{GB}$
- $2^{40}\text{B} = 2^{10}\text{GB} = 1\text{TB}$

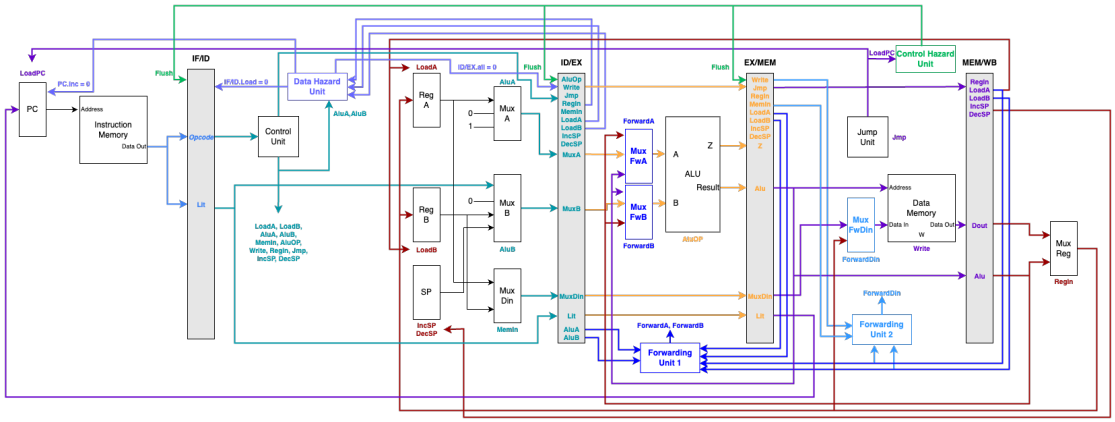
Pregunta 3: Paralelismo a Nivel de Instrucción (ILP) (15 ptos.)

Desarrolle las preguntas de esta sección a partir del diagrama del computador básico con *pipeline*:



- Modifique la arquitectura del computador básico con *pipeline* para usar la memoria de *stack* a través de las instrucciones `PUSH A`, `PUSH B`, `POP A` y `POP B`, ejecutándose en solo 5 ciclos cada una y actualizando el valor del registro `SP` en la etapa *Writeback* **(8 puntos)**. Puede ignorar los *hazards* que podrían darse en su implementación. Se descontará **1 punto** por cada conexión, señal faltante o error en su implementación. No es necesario que asigne un *opcode* a cada instrucción, pero sí debe incluir una tabla con la combinación de señales para cada una de ellas. Se descontarán **2 puntos** si no la incluye.

Solución: Para esta implementación, se posicionará el contador `SP` en la etapa *Instruction Decode*, ya que esto nos permite hacer uso del componente `Mux A` para computar la dirección correcta en las instrucciones `POP`. Luego, las señales `IncSP` y `DecSP` se propagan entre etapas hasta cargarlas de vuelta en `SP`:



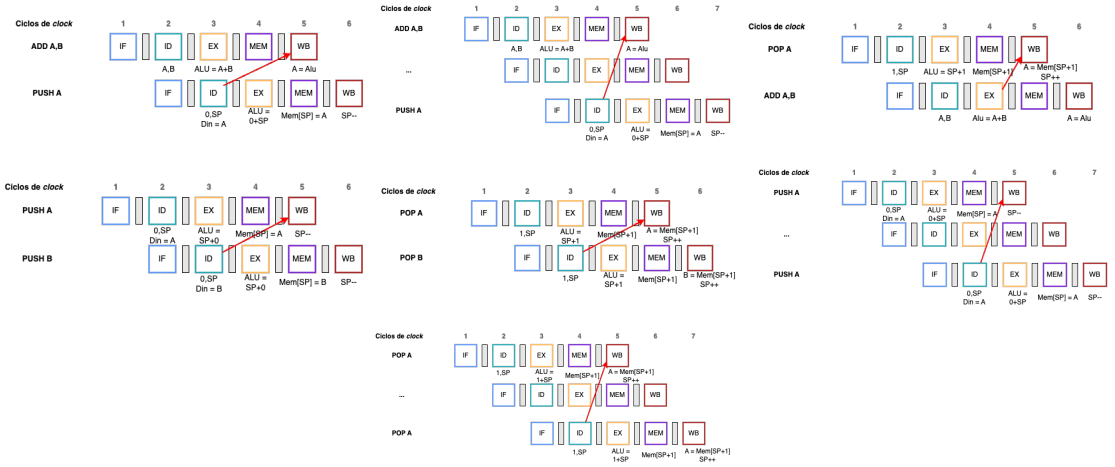
Con esta microarquitectura, la tabla de señales queda de la siguiente forma:

Instrucción	LoadA	LoadB	LoadPC	Write	IncSP	DecSP	Jmp	AluA	AluB	MemIn	AluOP	RegIn
PUSH A	0	0	0	1	0	1	0	0	SP	A	ADD	-
PUSH B	0	0	0	1	0	1	0	0	SP	B	ADD	-
POP A	1	0	0	0	1	0	0	1	SP	-	ADD	DOUT
POP B	0	1	0	0	1	0	0	1	SP	-	ADD	DOUT

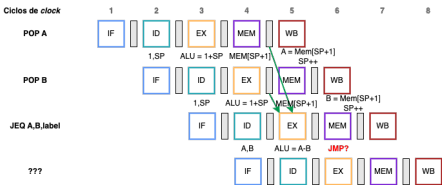
- Indique, con ejemplos, los tipos de *hazard* que podrían surgir de las instrucciones añadidas **(2 puntos)**. No necesita de la implementación anterior para responder este ítem, pero sí debe señalar las etapas en las que ocurren.

Solución: En este casos, se tienen *hazards* de datos y *hazards* de control.

***Hazards* de datos:** Estos ocurren por dos motivos: por lecturas o escrituras de memoria con registros `A`, `B` no actualizados; y también por la no actualización de `SP` entre operaciones `PUSH` y `POP` sucesivas. Se muestran a continuación con ejemplos todos los casos existentes:



***Hazards* de control:** Estos ocurren cuando existe una instrucción `POP` antes de un salto condicional. Se muestra a continuación con un solo ejemplo todos los casos existentes:



- A partir del siguiente programa del computador básico con *pipeline* y memoria de *stack*:

```
DATA:
  n 2
  res 0
CODE:
  MOV A,(n)
  MOV B,0
  PUSH B
  main:
    POP B
    JEQ A,B,end    // If A == B -> Salto a end.
    ADD B,1
    PUSH B
    JMP main
  end:
    ADD A,B
    MOV (res),A
```

Determine el número de ciclos que demora su ejecución detallando en un diagrama los estados del *pipeline* por instrucción. Asuma que posee la arquitectura para hacer *forwarding* para el manejo de *stack* de la misma manera que se resuelven los *hazards* de lectura y escritura en la memoria de datos; que el manejo de *stalling* es por *software* a través de la instrucción `NOP`; y que la unidad predictora de saltos asume que estos nunca se realizan, es decir, se incrementa el registro `PC` y se realiza *flushing* en la etapa *Memory* si es que la predicción fue incorrecta. Indique en el diagrama cuando ocurre *forwarding* (flechas de registro a etapa), *stalling* (con instrucción `NOP`) y *flushing* (con tachado en las instrucciones *flusheadas*) **(5 puntos)**.
Se descontará un punto por cada error o indicación faltante en su diagrama.

Solución: A continuación, se muestra el *pipeline* resultante de la ejecución del programa.

Del *pipeline* anterior se deduce que el programa termina de correr después de 33 ciclos. Cabe destacar que el *forwarding* de color rosado entre la tercera y la cuarta instrucción se realiza de esa forma porque en la siguiente solución se resolverán estos *hazards* propagando las señales `IncSP`, `DecSP`. Este caso **no se considera** dentro de los descuentos de puntaje.

- **BONUS:** Modifique la arquitectura que realizó en el primer ítem para resolver **todos los *hazards*** que haya detectado, independiente de su tipo **(5 puntos)**. Se otorgará este puntaje adicional **solo** si su implementación funciona y los *hazards* detectados son correctos. Puede añadir unidades nuevas o modificar las ya existentes para el funcionamiento correcto de su modificación. Si realiza modificaciones, cuide que no afecten el funcionamiento del resto de instrucciones y detecciones de *hazards*.

Solución: Por construcción, los siguientes casos están cubiertos:

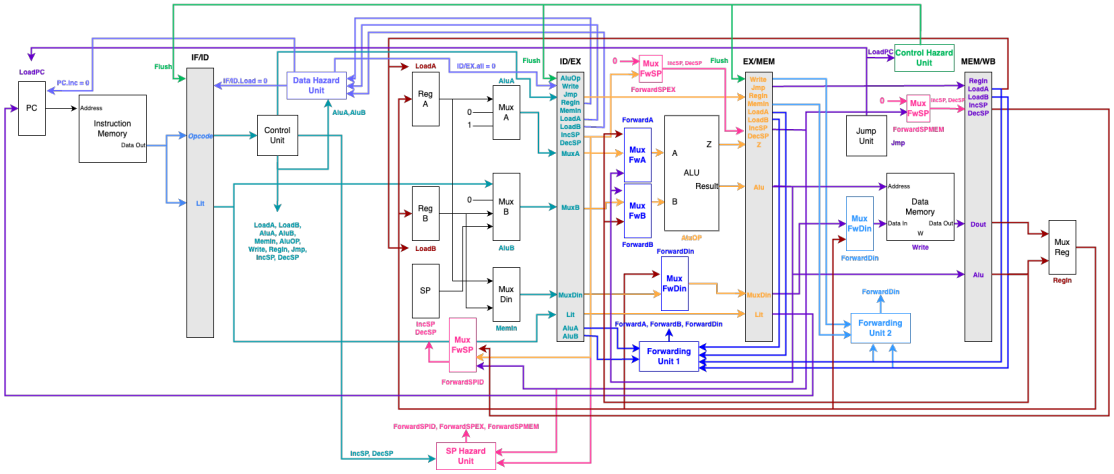
- *Hazards* de datos que involucran registros que luego escriben en memoria (`PUSH`) ya que son detectados y resueltos por la *forwarding unit* 2 con la siguiente combinación de señales: `MEM/WB.LoadX == 1 and EX/MEM.MuxDin == X and EX/MEM.Write == 1`.
- *Hazards* de datos que involucran inserción de burbujas por lectura de memoria (`POP`) ya que son detectados y resueltos por la *data hazard unit* con la siguiente combinación de señales: `ID/EX.LoadX == 1 and ID/EX.RegIn == Dout and ID.AluX == X`.
- *Hazards* de control, ya que la *control hazard unit* los detecta y resuelve por *flushing* de la misma forma que con otras instrucciones.

Los casos no resueltos son los siguientes:

- Escritura en memoria (`PUSH`) de un registro modificado en la instrucción previa en la anterior. Esto es algo que también ocurre para cualquier tipo de escritura en memoria, este caso **no está cubierto en el computador básico con *pipeline* del curso**. Para resolverlo, se agrega un nuevo `Mux FwDin` a la etapa *Execute* y una nueva señal de salida `ForwardDin` para la *forwarding unit* 1 de forma que pueda resolverlo vía *forwarding* a través de la siguiente combinación de señales: `MEM/WB.LoadX == 1 and ID/EX.MemDin == X and ID/EX.Write == 1 and EX/MEM.Write == 0`. Esta modificación **no será considerada para el bonus**.
- Uso consecutivo de instrucciones `PUSH` y/o `POP`. En estos casos, no se alcanza a actualizar el *stack pointer* para que la siguiente instrucción se calcule de forma correcta. Por lo tanto, se agregará una nueva componente llamada *stack pointer unit* que hará lo siguiente:
  - Detectará que debe realizar *forwarding* de las señales `IncSP` o `DecSP` del registro `ID/EX` según el siguiente criterio: `(ID.IncSP == 1 or ID.DecSP == 1) and (ID/EX.IncSP == 1 or ID/EX.DecSP == 1)`. En este caso, hará *forwarding* de las señales `ID/EX.IncSP` y `ID/EX.DecSP` a través de un nuevo componente `Mux FwSP` ubicado en la etapa `ID` para actualizar `SP` y, a partir de otro componente `MuxFwSP` ubicado en la etapa `EX`, hará *forwarding* de señales constantes 0 para evitar que la instrucción anterior modifique `SP`, puesto que ya será actualizado por la instrucción actual.

- Detectará que debe realizar *forwarding* de las señales `IncSP` o `DecSP` del registro `EX/MEM` según el siguiente criterio: `(ID.IncSP == 1 or ID.DecSP == 1) and (EX/MEM.IncSP == 1 or EX/MEM.DecSP == 1) and (ID/EX.IncSP == ID/EX.DecSP == 0)`. En este caso, hará *forwarding* de las señales `EX/MEM.IncSP` y `EX/MEM.DecSP` a través del `Mux FwSP` ubicado en la etapa `ID` para actualizar `SP` y, a partir de otro componente `MuxFwSP` ubicado en la etapa `MEM`, hará *forwarding* de señales constantes 0 para evitar que la instrucción previa a la anterior modifique `SP`, puesto que ya será actualizado por la instrucción actual.

Las modificaciones antes señaladas se transparentan en el siguiente diagrama:



No es relevante que la implementación sea exactamente igual a esta, pero sí se espera que se resuelvan *hazards* que no sean detectados por las unidades ya existentes como mínimo para optar al bonus, sobre todo en el caso del *stack pointer*.