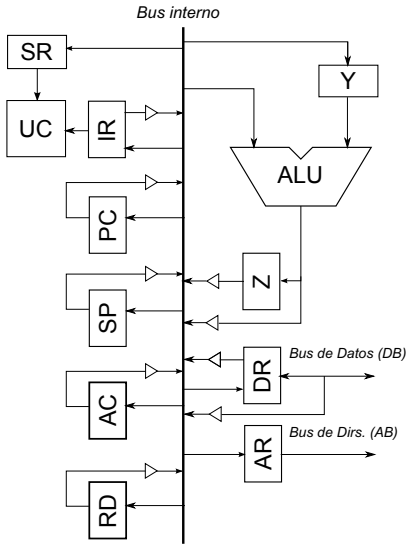
1 [3 puntos] Sea la CPU cuyo esquema simplificado (o *datapath*) aparece en la figura. La ALU, todos los registros, rutas de datos y de direcciones son de 32 bits.

PC: Reg. contador de programa
AR: Reg. de direcciones
IR: Reg. de instrucciones
Y, Z: registros *transparentes* o *temporales*
RD: es un registro *visible al usuario* que se usa sólo para direcciones
SP: Reg. puntero de pila
DR: Reg. de datos
SR: Reg. de estado
AC: es el único registro de *propósito general*



Suponiendo que:

1. las instrucciones ocupan todas una sola palabra y el direccionamiento es a nivel de byte.
2. las microoperaciones duran un ciclo de reloj excepto las que acceden a memoria, que ocupan *dos ciclos*.
3. el tiempo de ciclo es 50 ns.
4. la ALU realiza las operaciones habituales (suma, resta, incremento, decremento, etc.)
5. la pila crece hacia direcciones decrecientes y SP apunta a la primera dirección libre.

Realice la descomposición en operaciones elementales o microoperaciones para los casos que aparecen a continuación. *Siga las siguientes indicaciones:*

- en cada caso comience describiendo brevemente con palabras el funcionamiento o significado de cada una de las instrucciones propuestas, y de la fase de fetch que aparece primero.
- indique siempre claramente la o las microoperaciones *en cada uno de los ciclos de reloj* de la secuencia correspondiente.
- trate siempre de conseguir la secuencia con la menor duración posible.
- indique claramente con el texto **ACTUALIZAR_SR** los ciclos en los que, en su caso, se deba actualizar el registro de estado, SR.

a) el **fetch**.

b) las siguientes tres instrucciones (señale con "fetch" la secuencia anterior, supuesta al principio de cada instrucción) :

```
1) ST .AC, [.RD++]

2) POP .AC

3) BR [.RD]
```

SOLUCIÓN

a) fetch :

Lleva a IR la instrucción a la que apunta el PC y luego lo incrementa en una palabra:

"M(PC) → IR"

"PC+4 → PC"

```
i:    PC → AR
      PC → Y
i+1:  M(AR) →
      Y+4 → PC
i+2:  M(AR) → IR
```

b) instrucciones :

`1) ST .AC, [.RD++]`

Almacena el contenido de AC en la dirección de memoria RD e incrementa RD en una palabra:

"AC → M(RD)"

"RD+4 → RD"

```
      fetch
i+3:  RD → AR
      RD → Y
i+4:  AC → DR
i+5:  DR → M(AR); 1er ciclo de M
      Y+4 → RD
i+6:  DR → M(AR); 2o ciclo de M
```

`2) POP .AC`

Desapila AC:

"M(SP+4) → AC"

"SP+4 → SP"

```
      fetch
i+3:  SP → Y
i+4:  Y + 4 → SP
      Y + 4 → AR
i+5:  M(AR) → ; 1er ciclo de M
i+6:  M(AR) → AC; 2o ciclo de M
```

`3) BR [.RD]`

Lleva RD al PC:

"RD → PC"

```
       fetch
i+3:  RD → PC
```

2 [3 puntos] Dado un computador cuyo sistema de memoria está compuesto por una caché con política CBWA y tiempo de acceso de 3 ns y una memoria principal con tiempo de acceso de 60 ns:

a) Indique razonadamente cuántos accesos, y de qué tipo (lectura o escritura), llegan a la memoria principal cada vez que el procesador realiza una petición de acceso a memoria en los casos que se indican a continuación y cuál es el tiempo de acceso en cada caso. Considere para ello que el tamaño de los bloques es de 8 palabras.

1. Lectura con acierto en caché
2. Escritura con acierto en caché
3. Lectura con fallo en caché y bloque a reemplazar No modificado
4. Lectura con fallo en caché y bloque a reemplazar modificado
5. Escritura con fallo en caché y bloque a reempla-

zar No modificado

6. Escritura con fallo en caché y bloque a reemplazar modificado

b) Indique cuál o cuáles de estos casos no sería posible si la política de escritura de la caché fuese WTWNA. Para los casos que sí serían posibles indique si cambiaría y cómo, tanto el número de accesos a Mp como el tiempo de acceso.

SOLUCIÓN

a) Los accesos y tiempos pedidos son los siguientes:

Casos 1 y 2:

Ningún acceso a Mp

$Tacc = TMca = 3ns$

Casos 3 y 5:

8 accesos de lectura a Mp para leer las 8 palabras que componen el bloque

$Tacc = TMca + 8 \times TMp + TMca = 3 + 8 \times 60 + 3 = 486ns$

Casos 4 y 6:

8 accesos de escritura a Mp para actualizar en Mp el bloque reemplazado modificado, más

8 accesos de lectura de Mp para leer las 8 palabras que componen el bloque que produjo el fallo.

$Tacc = TMca + 2 \times 8 \times TMp + TMca = 3 + 16 \times 60 + 3 = 966ns$

b) Si la política de escritura fuese WTWNA no se podrían dar los casos 4 y 6 ya que con esta política la escritura se realiza a la vez en caché y en Mp por lo que un bloque que esté en caché y en Mp siempre estará actualizado en esta última. Los accesos a Mp y tiempos para el resto de los casos serían los siguientes:

Casos 1 y 3: Igual que con CBWA

Casos 2 y 5:

1 acceso de escritura a Mp

$Tacc = TMp = 60ns$

3 [4 puntos] Programe en ensamblador del MC88110 la siguiente subrutina:

a) Subrutina `MATRIZ_SEEK_COL(m, valor, matriz, resultado)`, que cuenta el número de veces que la matriz cuadrada de dimensión `m` contiene en cada columna el número `valor`. La matriz está almacenada por filas a partir de la dirección `matriz`. El resultado se almacena en un vector de elementos enteros a partir de la dirección `resultado`, donde escribirá en cada uno de sus elementos la cuenta del número de elementos iguales a `valor` en cada columna. Los parámetros se pasan en la pila: `m` y `valor` por valor y los otros por dirección.

Un ejemplo de una invocación a esta subrutina con los parámetros `MATRIZ_SEEK_COL(4, 1, matriz, resultado)` se muestra en la figura.

Para la realización del ejercicio se dispone de la función de librería `CUENTA(x, nro, vector)` que calcula el número de elementos que son iguales al valor `nro` en un vector de `x` elementos enteros de una palabra almacenados en posiciones consecutivas de memoria. Los parámetros se pasan en la pila: `x` y `nro` por valor y `vector` por dirección. El resultado se devuelve en el registro r29.

La subrutina `MATRIZ_SEEK_COL(m, valor, matriz, resultado)` deberá hacer uso de la función de librería `CUENTA`, por lo que antes de llamarla deberá extraer los elementos de cada columna de la matriz almacenándolos en pila en un vector con elementos consecutivos como variable local.

Para su realización se llevará a cabo el tratamiento del marco de pila descrito en clase y se supondrá que están definidas todas las macros que se han explicado en la parte teórica de la asignatura, que la subrutina llamante deja disponibles todos los registros (excepto r1, r30 (SP) y r31 (FP)), que la pila crece hacia direcciones de memoria decrecientes y que el puntero de pila apunta a la última posición ocupada.

matriz

1	-1	1	1
0	2	3	-4
1	0	3	0
1	0	3	1

resultado

3	0	1	2

SOLUCIÓN

a) La subrutina MATRIZ_SEEK_COL recibe en la pila la dirección del resultado, la dirección de la matriz, el orden de la matriz cuadrada almacenada por filas y el valor del número a contar en las columnas.

Tras crear el marco de pila se leerán de los parámetros, reservándose tantas palabras en la pila como elementos tiene cada columna de la matriz. El algoritmo a aplicar consiste en recorrer las columnas de la matriz, y por cada una, se llama a CUENTA, por lo que hará falta un contador de columnas de control del bucle, un puntero que apunte a columna (r20) y otro puntero que apunte el vector resultado (r22). Para cada columna, se extraen los elementos de la columna y se almacenan consecutivamente como un vector en pila en las posiciones reservadas para variables locales. Después se llama a la función CUENTA con la dirección del vector en pila, el valor numérico a buscar, y su número de elementos. Antes, hay que salvar en zona de variables locales los punteros a columna y al vector con resultados, por si los modifica CUENTA.

Tras retornar de CUENTA, sólo hay que escribir el resultado devuelto en r29 en la posición del puntero al vector resultado.

```
MATRIZ_SEEK_COL:  PUSH(r1)               ; salvaguardar dirección de retorno
                  PUSH (r31)             ; salvaguardar FP del llamante
                  or r31, r30, r0        ; crear marco de pila
                  ld r3, r31, 8          ; leer m
                  addu r4, r3, 8
                  subu r30, r30, r4      ; reservar espacio para el vector y dos punteros

                  ld r20, r31, 16        ; leer parámetro con dirección de matriz
                  ld r22, r31, 12        ; leer  parámetro con dir del resultado
                  or r21, r30, r0        ; inicializar puntero al vector en var local
                  or r6, r3, r0          ; inicializar contador de columnas

                  ; copiar la columna en direcciones consecutivas de pila
buc_copiar:       ld r6, r20, r7         ; cargar elemento de matriz
                  st r6, r21, r0         ; guardar elemento en vector
                  addu r20, r20, 4       ; avanzar puntero a columna
                  addu r21, r21, 4       ; avanzar puntero al vector
                  subu r6, r6, 1         ; decrementar contador  de columna
                  cmp r3, r6, r0
                  bb1 ne, r3, buc_copiar

                  ; llamar a CUENTA
contar_fila:      st r20, r31, -4        ; guardar puntero a fila en variable local
                  st r22, r31, -8        ; guardar puntero a resultado en variable local
                  or r21, r30, r0        ; recuperar puntero a vector
                  PUSH(r21)              ; pasar parámetro dirección de vector
                  ld r5, r31, 12         ; recuperar valor a buscar
                  PUSH (r5)              ; pasar parámetro valor a buscar
                  ld r5, r31, 8          ; recuperar tamaño del vector
                  PUSH (r5)              ; pasar parámetro tamaño del vector
                  bsr CUENTA             ; llamada a subrutina
                  addu r30, r30, 12      ; limpiar parámetros de pila
                  ld r20, r31, -4        ; recuperar de var locales puntero a fila
                  ld r22, r31, -8        ; recuperar de var locales puntero a resultado
                  st r29, r22, r0        ; almacenar cuenta en elemento del vector resultado
                  subu r6, r6, 1         ; decrementar contador  de columna
```

```
cmp r3, r6, r0
bb1 ne, r3, contar_fila
or r30, r31, r0         ; destruir variables locales
POP (r31)               ; recuperar FP del llamante
POP (r1)                ; recuperar dirección de retorno
jmp (r1)                ; retornar
```