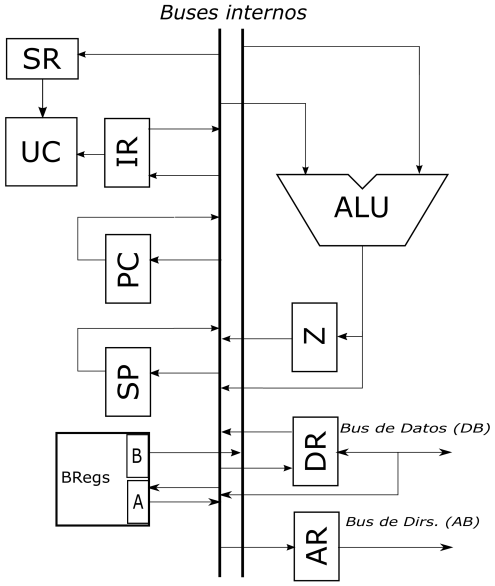
1 [3 puntos] Sea la CPU cuyo esquema simplificado (o *datapath*) aparece en la figura. La ALU, todos los registros, rutas de datos y de direcciones son de 32 bits.

PC: Reg. contador de programa
AR: Reg. de direcciones
IR: Reg. de instrucción
Z: registro *transparente* o *temporal*
BRegs: banco de registros de *propósito general*, R0..R7
SP: Reg. puntero de pila
DR: Reg. de datos
SR: Reg. de estado



Suponiendo que:

1. el banco de registros dispone de dos puertas, A y B, que permiten a la UC seleccionar cualquier pareja de registros en cada ciclo.
2. cada instrucción ocupa una palabra.
3. el campo #desp de la instrucción se indica como IR.desp del reg. de instrucción.
4. la memoria es direccionable a palabra y necesita para operar *dos* ciclos de reloj.
5. el tiempo de ciclo de reloj es 50 ns.
6. la pila se llena 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**, común a todas las instrucciones.

b) las siguientes cuatro instrucciones, indicando con "fetch" la secuencia anterior (que se supone al principio de cada instrucción).

```
1) ST .R4, #14[.R5]

2) ADD .R2, .R4, .R6

3) POP .R3

4) CALL [.R6++]
```

SOLUCIÓN

a) fetch

Funcionamiento:

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

"M(PC) → IR"

"PC+1 → PC"

Secuencia de microoperaciones:

```
i:    PC → AR
      PC + 1 → Z
i+1:  M(AR) → ; 1er ciclo de M
      Z → PC
i+2:  M(AR) → IR; 2o ciclo de M
```

b) instrucciones:

```
1) ST .R4, #14[.R5]
```

Funcionamiento:

Almacena el contenido del registro R4 en la dirección de memoria que corresponde al contenido del registro R5 más el desplazamiento.

"R4 → M(R5+IR.desp)"

Secuencia de microoperaciones:

```
      fetch
i+3:  IR.desp + R5 → Z
i+4:  Z → AR
i+5:  R4 → DR
i+6:  DR → M(AR); 1er ciclo de M
i+7:  DR → M(AR); 2o ciclo de M
```

```
2) ADD .R2, .R4, .R6
```

Funcionamiento:

Suma al contenido de R4 al contenido de R6 y deja el resultado en R2, a la vez que actualiza el registro de estado, SR

"R4 + R6 → R2" y actualiza SR

Secuencia de microoperaciones:

```
      fetch
i+3:  R4 + R6 → Z; ACTUALIZAR_SR
i+4:  Z → R2
```

```
3) POP .R3
```

Funcionamiento:

Desapilar el R3:

"SP+1 → SP"

"M(SP) → R3"

Secuencia de microoperaciones:

```
        fetch
i+3:  SP + 1 → Z; nuevo valor de SP
i+4:  Z → SP; actualiza SP
      Z → AR
i+5:  M(AR) → ; 1er ciclo de M
i+6:  M(AR) → PC; 2o ciclo de M
```

`4) CALL [.R6++]`

Funcionamiento:

Apila el PC (que es la dirección de retorno) y pone en el PC el contenido de R6, que es la dirección de la rutina, que es luego postincrementado:

"Apilar PC"

"R6 → PC"

"R6 + 1 → R6"

A su vez, apilar el PC conlleva las siguientes acciones:

"PC → M(SP)"

"SP - 1 → SP"

Secuencia de microoperaciones:

```
        fetch
i+3:  SP → AR
      SP - 1 → Z; nuevo valor de SP
i+4:  PC → DR
i+5:  DR → M(AR); 1er ciclo de M
      Z → SP; actualiza SP
i+6:  DR → M(AR); 2o ciclo de M
      R6 → PC; actualiza PC con el contenido de R6
      R6 + 1 → Z; nuevo valor de R6
i+7:  Z → R6; actualiza R6 con el incremento
```

2 [3 puntos] Sea un computador con un ancho de palabra de 64 bits y cuyo sistema de memoria compuesto por sólo dos niveles, memoria caché y memoria principal con las siguientes características:

- Memoria principal:
  - tiempo de acceso: 30 ns
- Memoria caché:
  - tiempo de acceso: 1 ns
  - tamaño de los bloques: 32 bytes
  - completamente asociativa
  - política de escritura: WTWNA, *write-through with not allocation.*

a) Se pide calcular razonadamente los siguiente tiempos de acceso:
a.1) mínimo en lectura **a.2)** mínimo en escritura **a.3)** máximo en lectura **a.4)** máximo en escritura

b) Calcule el tiempo medio de acceso suponiendo una tasa de aciertos del 92 %. Razone si sería necesario conocer el porcentaje de lecturas, %R, y escrituras, %W. Si su respuesta es afirmativa, indíquelo en el cálculo anterior.

c) Indique razonadamente cómo afectaría al cálculo del tiempo medio de acceso realizado en el apartado anterior el que el tamaño de los bloques de caché fuera 64 bytes en lugar de los 32 originales.

d) Indique también razonadamente cómo afectaría al cálculo del tiempo medio de acceso realizado en el apartado anterior el que la política de escritura fuese CBWA, *copy back with allocation*.

SOLUCIÓN

a)

a.1) mínimo en lectura El tiempo mínimo se corresponde con un acierto en Mca, y por tanto es 1 ns:

$$t_{LecturaMin} = t_{Mca} = 1\ ns$$

a.2) mínimo en escritura El tiempo mínimo en este caso se corresponde en este caso al de una escritura en memoria, dada la política de escritura WT:

$$t_{EscrituraMin} = t_{Mp} = 30\ ns$$

a.3) máximo en lectura

Se produciría en los accesos en lectura con fallo, habría que sumar el tiempo correspondiente a copia –120 ns correspondientes a las cuatro palabras del bloque– el bloque desde Mp, "subirlo". Nótese que "WNA" sólo se aplica a los fallos en escritura.

$$t_{LecturaMax} = 1\ ns + 120\ ns + 1\ ns = 122\ ns$$

a.4) máximo en escritura

En este caso es el tiempo de acceso es tiempo de Mp, por "WNA", el bloque que ha producido el fallo no se copia a la caché:

$$t_{EscrituraMax} = t_{Mp} = 30\ ns$$

b)

En este caso, las lecturas y las escrituras tienen un tratamiento diferente. En el caso de las lecturas, pasan por la caché y si hay fallo tiene el sobrecoste de copiar el bloque desde Mp y otro tiempo de caché –tal y como se ha supuesto en clase–. En el caso de las escrituras, el tiempo de acceso es siempre el tiempo de Mp, ya que en los fallos no se copia el bloque a la caché y se escribe directamente en el nivel inferior, Mp.

Una expresión matemática podría ser la siguiente, donde %R representa la proporción de lectura (en tanto por uno), por lo que 1- %R será la proporción de escrituras.

$$\overline{t}_{acceso} = (1 - \%R) \times 30\ ns + 0,08 \times \%R \times (1\ ns + 120\ ns + 1\ ns)$$

c) Si los bloques fuesen de 64 bytes corresponderían a 8 palabras en lugar de a las 4 originales. Este nuevo tamaño haría que el tiempo necesario para copiar un bloque pasase a ser 8 pal x 30 ns/pal, 240 ns

d) Si la política de escritura fuese CBWA no habría que distinguir entre lecturas y escrituras. El tiempo mínimo en ambos casos correspondería a aciertos en cachés, 1 ns.

El tiempo máximo correspondería al caso de que se produciese fallo –tanto en lectura como en escrituras– en la caché y el bloque que se reemplazase estuviese modificado. En este caso habrá que copiar el bloque que se reemplaza en Mp ("bajarlo"), y luego copiar el bloque que produjo el fallo de la Mp a la caché (o "subirlo"). El tiempo de copiar un bloque a o desde memoria es de 120 ns, correspondientes a las cuatro palabras del bloque por los 30 ns de su tiempo de acceso.

3 [4 puntos] Se dispone de un texto ASCII, almacenado en memoria, que finaliza con el carácter NUL (código ASCII 0x00). Cada carácter del texto se representa mediante un entero sin signo de 8 bits. Se desea escribir un programa en ensamblador del 88110 que cifre dicho texto. El cifrado consiste en sustituir cada carácter por otro que es el que ocupa la posición correspondiente a su código ASCII en la cadena de cifrado. Así, por ejemplo, el carácter ASCII 'a' (cuya codificación es 0x61) se transformaría en el carácter que ocupe la posición 0x61 de la cadena de cifrado (en el ejemplo 0x30).

Texto:

a	b	e	c	NUL

Texto (Codificación ASCII): 0x61, 0x62, 0x65, 0x63, 0x00

Cadena de cifrado	Posición:	...	0x61	0x62	0x63	0x64	0x65	....
	Contenido:	...	0(0x30)	a(0x61)	b(0x62)	1(0x31)	c(0x63)	....

Texto cifrado:

0	a	c	b	NUL

Texto cifrado (Codificación ASCII): 0x30, 0x61, 0x63, 0x62, 0x00

a) Programe en ensamblador del MC88110 la subrutina `cifra(texto, cifrado)` que realiza la transformación indicada en el enunciado: `texto` es la cadena de caracteres que se desea cifrar y es un parámetro de entrada/salida, puesto que el resultado de la propia transformación se almacenará en esta cadena. `cifrado` es la cadena de cifrado y ambos parámetros se pasan por dirección.

Para la realización de este ejercicio 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 las subrutinas llamantes dejan disponibles todos los registros excepto r1, r30 (SP) y r31 (FP); que la pila crece hacia direcciones de memoria decrecientes y el puntero de pila apunta a la última posición ocupada.

SOLUCIÓN

La solución que se muestra a continuación consiste en recorrer el texto a cifrar hasta que se encuentra el carácter NUL. Cada carácter del texto se utiliza como índice a la cadena de cifrado, recogiendo dicho carácter y sustituyéndolo en la cadena `texto`.

Puesto que esta subrutina no realiza ninguna llamada ni utiliza variables locales en la pila, no se almacena la dirección de retorno en la pila ni se crea marco de pila.

```
cifra:  ld r21,r30,4        ; Se carga la cadena de cifrado
        ld r20,r30,0        ; Se carga texto
buc:    ld.bu r2,r20,r0     ; Se carga el carácter a cifrar
        cmp r7,r2,r0        ; Si es NUL se acaba
        bb1 eq,r7,fin
        ld.bu r3,r21,r2     ; Se obtiene el carácter "transformado"
        st.b r3,r20,r0      ; Se almacena el carácter cifrado en su posición
        addu r20,r20,1      ; Se incrementa el puntero al texto
        br buc
fin:    jmp(r1)
```