Estructura de Computadores

Soluciones Parcial 2. 12-mayo2020

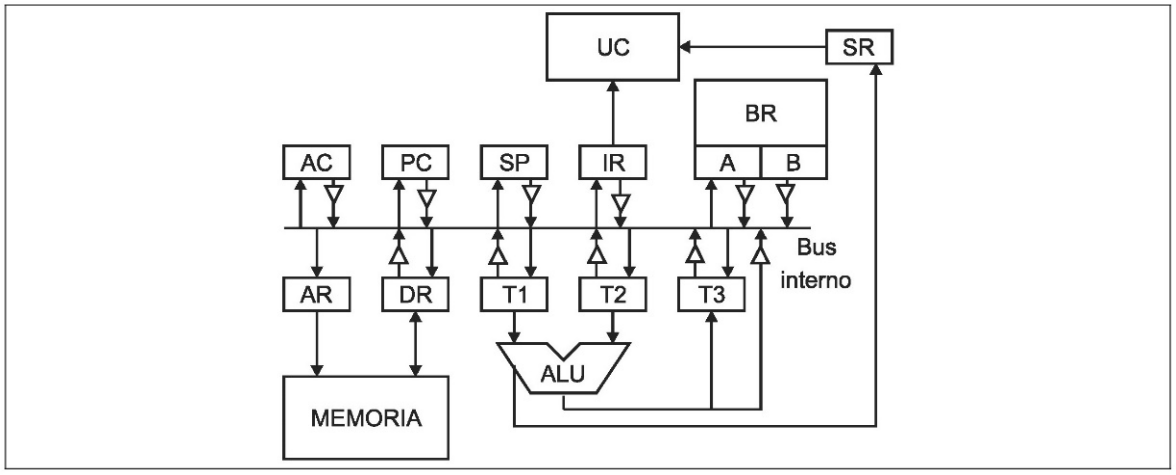
Ejercicios_procesador.-

1. En la secuencia de microoperaciones correspondiente a una instrucción de dos palabras:

- Las dos palabras de la instrucción se guardan en el registro de instrucción durante el fetch de la instrucción
- La segunda palabra hay que leerla y guardarla en algún registro cualquiera
- No se pueden ejecutar instrucciones de dos palabras

✓ Durante el fetch sólo se lee la primera palabra, y tras la decodificación es cuando hay que leer la segunda palabra

2. En la CPU de la siguiente figura:



✓ La secuencia de microoperaciones para realizar la instrucción de PUSH .R1 necesita de al menos dos ciclos de reloj correspondientes uno al necesario para hacer SP--> AR y otro al que se necesita para realizar R1 --> DR, separadamente.

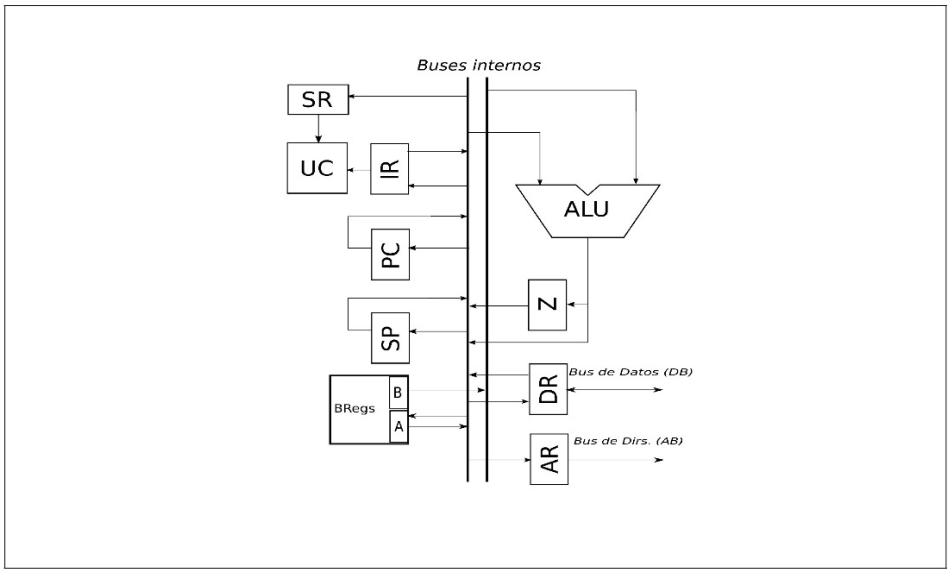
- En todos los ciclos de reloj se deberá actualizar el Registro de Estado con las condiciones aritméticas generadas por la ALU.
- La secuencia de microoperaciones para realizar la instrucción de PUSH .R1 permite realizar simultáneamente las operaciones elementales SP--> AR y R1 --> DR, durante el mismo ciclo de reloj, por no tener orden preestablecido entre ellas.
- No permite que se pueda realizar ninguna microoperación que use el bus interno a la vez que se accede a memoria.

3. En la siguiente CPU:

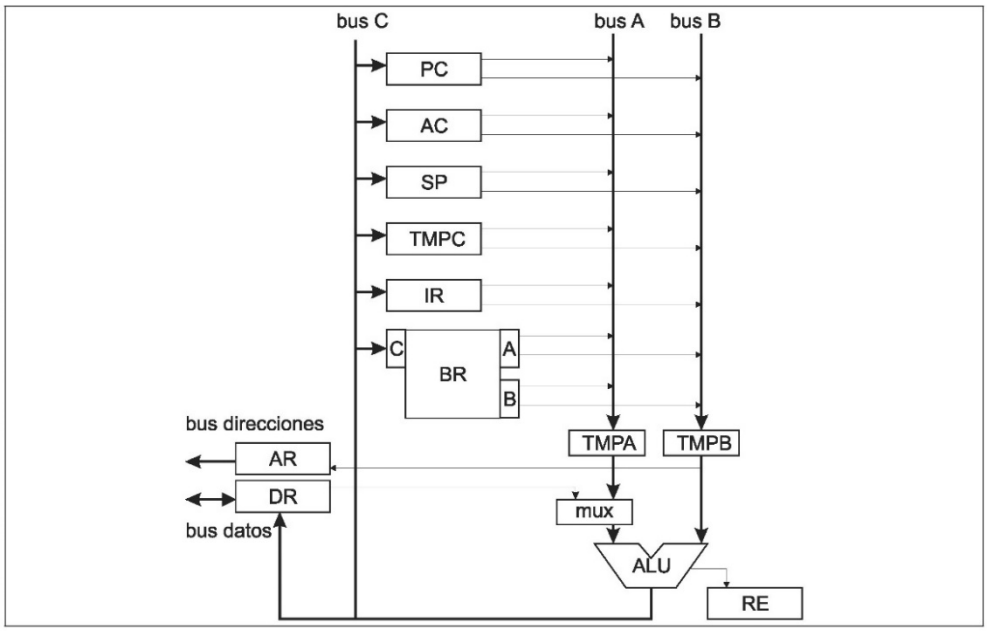
✓ Se puede realizar una microoperación arimética cuyos operandos sean dos registros diferentes del Banco de Registros.

- Se pueden realizar una microoperación aritmética cuyos operandos sean dos registros del Banco de Registros y cuyo destino sea otro registro del banco.

- No se puede realizar una microoperación aritmética cuyos operandos sean dos registros del banco de registros, porque habría conflicto en los buses internos.
- Se pueden realizar una microoperación aritmética cuyo resultado se deje en un registro del banco de registros y solapadamente en el mismo ciclo de reloj la actualización del registro de estado con la información de esa operación realizada en la ALU.



4. En la siguiente CPU:

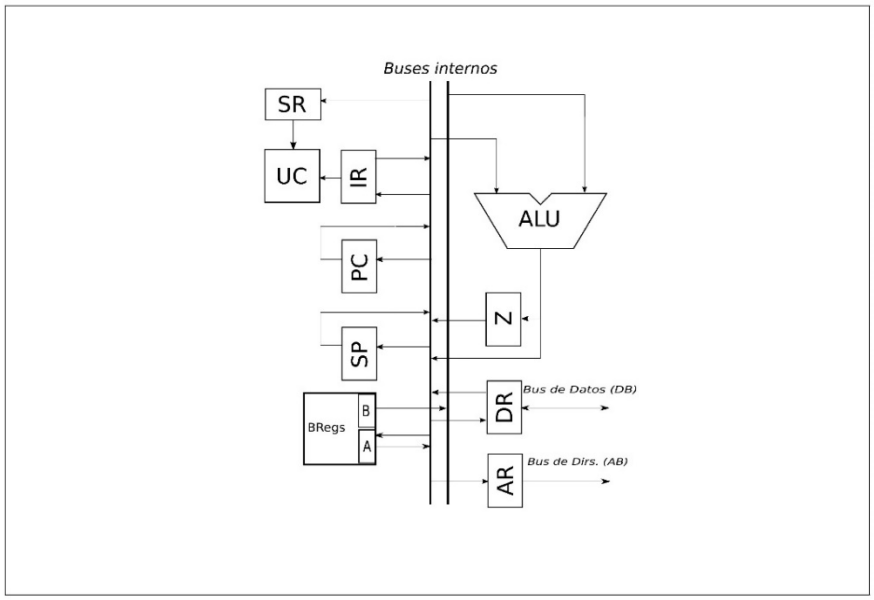


- Existe la microoperación SP --> AR

✓ Para llevar el contenido de DR a AR la secuencia más breve es la que lo pasa por los registros transparentes TMPC y TMPB.

- Para llevar el contenido de DR a AR la secuencia más breve es la que lo pasa por los registros transparentes TMPC y TMPA.
- No se puede llevar el contenido de DR a AR.

5. En la siguiente CPU:



✓ Se pueden ejecutar instrucciones con direccionamiento relativo a memoria, aunque el cálculo de la dirección del operando requiere varias operaciones elementales

- Se pueden ejecutar instrucciones con direccionamiento relativo a memoria, y el cálculo de la dirección del operando se puede realizar en un sólo ciclo de reloj
- No se pueden ejecutar instrucciones con direccionamiento relativo a memoria, ya que no está establecido el camino para ello
- Se pueden ejecutar instrucciones con direccionamiento relativo a memoria, si el desplazamiento se guarda en un registro general del banco de registros antes de calcular la dirección efectiva del operando

6. Se pueden solapar operaciones elementales:

✓ Podrían realizarse dos operaciones elementales simultáneas si no existiera conflicto en

- Siempre que no sea una de ellas un acceso a memoria, porque esta operación duraría varios ciclos
- el hardware o estructura de la CPU.
- No puede realizarse operaciones elementales simultáneas en ninguna CPU
- Podrían realizarse dos o más operaciones elementales simultáneas sólo cuando una de ellas es una operación de memoria.

7. En un único ciclo de reloj:

- Sólo puede realizarse una única operación elemental o microoperación.

✓ Podrían realizarse dos o más microoperaciones u operaciones elementales si no existiera conflicto en el hardware o estructura de la CPU.

- No puede realizarse una microoperación aritmética, ya que la ALU suelen necesitar varios ciclos de reloj para realizar sus operaciones.
- Se realizan habitualmente las microoperaciones de acceso a memoria.

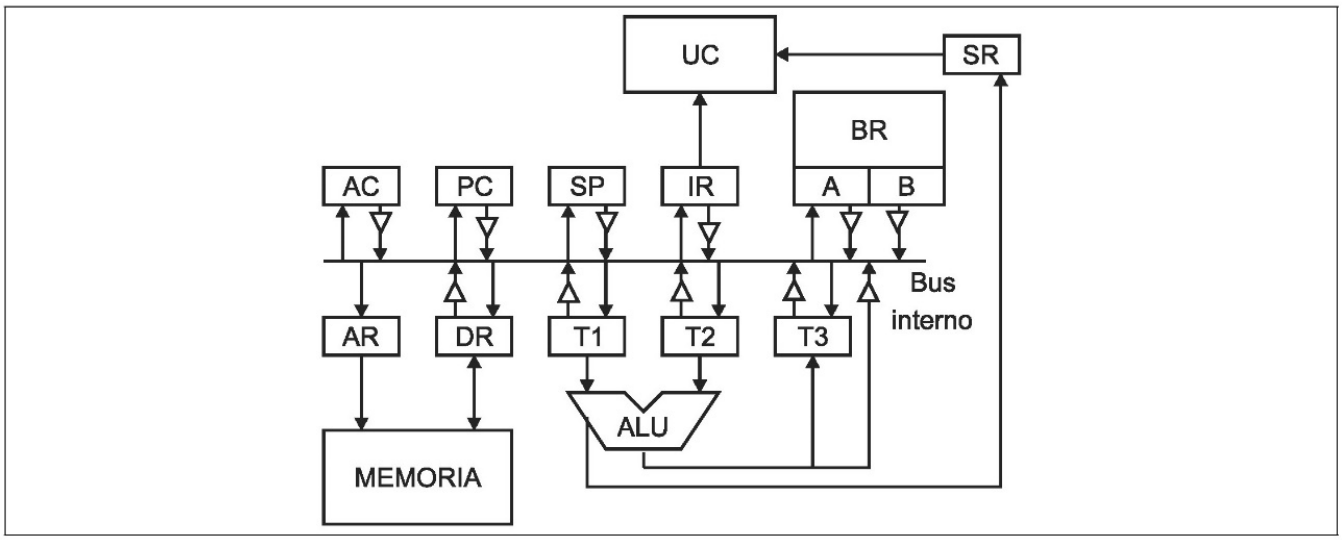
8. En la secuencia de microoperaciones correspondiente al fetch de la instrucción:

- Se actualiza siempre el Registro de Estado, ya que se incrementa el PC.

✓ Se suelen necesitar varios ciclos de reloj debido a que es necesario acceder a memoria.

- En todos los ciclos de reloj hay que actualizar el PC
- Habitualmente hay una microoperación u operación elemental que realiza todas las acciones necesarias en un único ciclo de rejoj.

9. En la CPU de la siguiente figura:



- La existencia de los registros transparentes T1, T2 y T3 añade siempre dos ciclos reloj a todas las microoperaciones aritméticas.
- La existencia de los registros transparentes T1, T2 y T3 añade al menos un ciclo de reloj a todas las microoperaciones aritméticas.
- No permite llevar el resultado generado por la ALU a ningún registro de propósito general.
- Si se suprimiesen T1 y T2 la ALU podría seguir utilizando dos operandos diferentes.

Ejercicios_memoria.-

1. *Sea un procesador con tamaño de palabra de 32 bits, direccionamiento a nivel de byte y una caché asociativa de 16 KB con bloques de 16 B y política de escritura aplazada con actualización (CBWA). En este computador se ejecuta el siguiente fragmento de código, que se muestra a la derecha en lenguaje de alto nivel, almacenado a partir de la dirección 0. Tanto las instrucciones como los elementos ocupan una palabra.*

```
     and .r1, #1024
     and .r2, #4096
     and .r7, #0
buc:  ld .r3, #0[.r1]        for (i=0; i<500; i=i+1)
     sub .r3, .r4                 c[i] = b[i]-a
     st .r3, #0[.r2]
     add .r1, #4
     add .r2, #4
     add .r7, #1
     cmp .r7, #500
     bnz $buc
```

Responda las siguientes cuestiones.

- *Calcule el número de bloques que ocupan tanto las instrucciones como los vectores {**253**}*

  Bloques de 4 pal; Instrucciones: 11/4 = 3 bloques; Datos: 500 elem_vector/4 = 125 bloques por vector. Total= 3+2·125 =253 bloques

- *Calcule el número total de accesos a memoria cache que se producen en la ejecución completa del código {**5003**}*

  3 inst + 500·(8 inst+2 datos) = 5003 accesos totales

- *Calcule el número total de fallos en memoria cache que se producen en la ejecución completa del código. Fallos de instrucciones: {**3**} Fallos de datos: {**250**}*

  Los datos caben en cache, por lo que sólo hay fallos forzosos, es decir, uno por bloque.

- *Calcule el número total de accesos aciertos en memoria cache que se producen en la ejecución completa del código {**4750**}*

  Aciertos = Totales – Fallos = 5003 – 253 = 4750 aciertos

- *Tasa de aciertos de la cache en %: {**94,94**}*

  Hr= 4750 aciertos/5003 totales = 0,9494 -> 94,94%

- *Calcule el tiempo medio de acceso a memoria (tiempo efectivo) sabiendo que el tiempo de acceso de la caché es 2ns y el de la Mp es 40ns (en ns). {**10,19**}*

  Hr= 4750 aciertos/5003 totales = 0,9494 -> 94,94%

  Tef = (4750 aciertos · 2ns + 253 fallos · (2ns + 4·40ns + 2ns)) / 5003 accesos =

$$Tef = \frac{4750 \text{ aciertos} \cdot 2\text{ns} + 253 \text{ fallos} \cdot (2\text{ns} + 4 \cdot 40\text{ns} + 2\text{ns})}{5003 \text{ accesos}} = 10{,}19 \text{ ns}$$

2. *Sea un procesador con tamaño de palabra de 32 bits, direccionamiento a nivel de byte y una caché asociativa de 16 KB con bloques de 64 B y política de escritura aplazada con actualización (CBWA). En este computador se ejecuta el siguiente fragmento de código, que se muestra a la derecha en lenguaje de alto nivel, almacenado a partir de la dirección 0, donde r1 y r2 contienen direcciones de comienzo de vectores a y b respectivamente, siendo sus valores iniciales r1=2048 y r2=1024. Tanto las instrucciones como los elementos ocupan una palabra.*

```
   and .r7, #0
buc:  ld .r3, #0[.r1]        for (i=0; i<160; i=i+1)
     mul .r3, .r4                 a[i] = b[i]*c + i
     add .r3, .r7
     st .r3, #0[.r2]
     add .r1, #4
     add .r2, #4
     add .r7, #1
     cmp .r7, #160
     bnz $buc
```

Responda las siguientes cuestiones.

- *Calcule el número de bloques que ocupan tanto las instrucciones como los vectores {**21**}*

  Bloques de 16 pal; Instrucciones: 10/16 = 1 bloque; Datos: 160 elem_vector/16 = 10 bloques por vector. Total= 1+2·10 = 21 bloques

- *Calcule el número total de accesos a memoria cache que se producen en la ejecución completa del código {**1761**}*

  1 inst + 160·(9 inst+2 datos) = 1761 accesos totales

- *Calcule el número total de fallos en memoria cache que se producen en la ejecución completa del código. Fallos de instrucciones: {**1**} Fallos de datos: {**20**}*

  Los datos caben en cache, por lo que sólo hay fallos forzosos, es decir, uno por bloque.

- *Calcule el número total de accesos aciertos en memoria cache que se producen en la ejecución completa del código {**1740**}*

  Aciertos = Totales – Fallos = 1761 – 21 = 1740 aciertos

- *Tasa de aciertos de la cache (en %): {**98,81**}*

  Hr= 1719 aciertos/1761 totales = 0,98807 -> 98,81%

- *Calcule el tiempo medio de acceso a memoria (tiempo efectivo) sabiendo que el tiempo de acceso de la caché es 2ns y el de la Mp es 40ns (en ns). {**10,67**}*
  Tef = (4750 aciertos · 2ns + 253 fallos · (2ns + 4·40ns + 2ns)) / 5003 accesos =

$$Tef = \frac{1740 \text{ aciertos} \cdot 3\text{ns} + 21 \text{ fallos} \cdot (3\text{ns} + 16 \cdot 40\text{ns} + 3\text{ns})}{1761 \text{ accesos}} = 10{,}67 \text{ ns}$$

Ejercicios programación en ensamblador.

1. *Sea el fragmento de código en el estándar IEEE 694 para un computador de palabras y direcciones de 32 bits. Prográmelo para el 88110 en la caja de texto o adjuntando un archivo*

```
        LD .R3,#VECTOR
BUCLE: LD .R2,[.R3++]
        CMP .R2,#4
        BNZ BUCLE
        ST .R3,[.R20++]

        LEA (r3,VECTOR)
bucle: ld r2,r3,r0
        addu r3,r3,4
        cmp r7,r2,4
        bb1 ne,r7,bucle
        st r3,r20,r0
        addu r20,r20,4
```

2. *Sea el fragmento de código en el estándar IEEE 694 para un computador de palabras y direcciones de 32 bits. Prográmelo para el 88110 en la caja de texto o adjuntando un archivo*

```
        LD .R3,#VECTOR
        ADD .R3,.R8
BUCLE: LD .R2,[--.R3]
        ST .R3,[.R20]
        CMP .R2,#4
        BNZ BUCLE

        LEA (r3,VECTOR)
        add r3,r3,r8
bucle: subu r3,r3,4
        ld r2,r3,r0
        st r3,r20,r0
        cmp r7,r2,4
        bb1 ne,r7,bucle
```

3. *Programe en ensamblador del 88110 los tres fragmentos de la subrutina funcion que se detalla a continuación. Suponga que los parámetros se pasan en la pila.*

```
// funcion recibe un entero (n) pasado por valor y un
// vector de enteros pasado por dirección

// Devuelve en r29 el resultado de la función que es un
valor entero.

int función (int n, int v [])
{
    int aux[n], i, j=0;

     /* Fragmento 0. Crea el marco de pila */
     /* Fragmento 1. Copia en la var local aux los
elementos diferentes a 0 del parámetro v, invirtiendo su
orden */

     for (i=0;i<n;i=i+1)
     {
          if (v[n-i-1]!=0)
          {
               aux[j] = v[n-i-1];
               j=j+1;
          }
     }

    /*Fragmento 2. */
     return j-3;
}
```

Solución:

```
funcion: PUSH(r1)
         PUSH(r31)
         or r31,r30,r30
         ld r2,r31,8   ; n
         ld r20,r31,12 ;v
         addu r3,r2,2
         mulu r3,r3,4
         subu r30,r30,r3
         st r0,r31,-4  ; j=0
         st r2,r31,-8  ; i
          ; Fragmento 1

         mulu r3,r2,4
         subu r3,r3,4
         addu r20,r20,r3 ; Apunta al último de v
         or r21,r30,r30  ; Apunta a aux
         or r4,r0,r0     ; j
bucle:   cmp r7,r2,r0
```

```
        bb1 eq,r7,fin_f2
        ld r3,r20,r0
        cmp r7,r3,r0
        bb1 eq,r7,cont
        st r3,r21,r0
        addu r21,r21,4
        addu r4,r4,1
cont:   subu r20,r20,4
        subu r2,r2,1
        br bucle
fin_f2: st r4,r31,-4

        ; Fragmento 2
        ld r29,r31,-4
        subu r29,r29,3
        or r30,r31,r31
        POP(r31)
        POP(r1)
        jmp(r1)
```

4. *Programe en ensamblador del 88110 los tres fragmentos de la subrutina funcion que se detalla a continuación. Suponga que los parámetros se pasan en la pila.*

```
     // funcion recibe un entero (n) pasado por valor y un
// vector de enteros pasado por dirección

     // Devuelve en r29 el resultado de la función que es
// un valor entero.

int función (int n, int v [])
{
    int aux[n], i, j=0, aux2;
     /* Fragmento 0. Crea el marco de pila */

     /* Fragmento 1. Copia en la var local aux algunos
elementos del parámetro v. */

     for (i=0;i<n;i=i+1)
     {
          aux2 = v[i] + v[n-i-1];
          if (aux2!=0)
          {
               aux[j] = v[i];
               j=j+1;
          }
     }

         /*Fragmento 2. */
     return j*2;
}
```

Solución:

```
funcion: PUSH(r1)
         PUSH(r31)
         or r31,r30,r30
         ld r2,r31,8   ; n
         ld r20,r31,12 ;v
         addu r3,r2,3
         mulu r3,r3,4
         subu r30,r30,r3
         st r0,r31,-4  ; j=0
         st r2,r31,-8  ; i

          ; Fragmento 1
         or r22,r20,r20
         or r23,r30,r30  ; Puntero a aux
         mulu r3,r2,4
         subu r3,r3,4
         addu r20,r20,r3 ; Apunta al último de v
         or r21,r30,r30  ; Apunta a aux
         or r4,r0,r0     ; j
bucle:   cmp r7,r2,r0
         bb1 eq,r7,fin_f2
         ld r7,r20,r0  ; elemento de final de v
         ld r3,r22,r0  ; elemento de v
         add r7,r3,r7
         cmp r7,r7,r0
         bb1 eq,r7, fin_f2
         st r3,r23,r0  ; Guardo en aux
         addu r23,r23,4 ; inc puntero a aux
         addu r4,r4,1   ; inc j
fin_f2:  addu r22,r22,4
         subu r20,r20,4
         subu r2,r2,1
         br bucle
         st r4,r31,-4  ; Actualiza j

                ; Fragmento 2
         ld r29,r31,-4
         add r29,r29,r29
         or r30,r31,r31
         POP(r31)
         POP(r1)
         jmp(r1)
```