27-8-21

Ejercicio (1)

Escribir un programa que lee una palabra de 32 bits entregada por un periferico mapeado en la dir C2100308h, esta comprende dos numeros enteros signados en sus 16 bits más altos y en sus 16 bits más bajos respectivamente. El main pasa esta palabra de 32 bits a una rutina vía stack para que se encargue de separar ambos numeros y devolverlos también a través del stack

El programa principal suma estos dos numeros y escribe el resultado en el mismo periferico desde donde leyó el valor original y wepo termina. El programa principal y la rutina van en el mismo módulo.

```
.begin
.org 2048
.macro  push reg
        add %r14, -4, %r14
        ST  reg , %r14
.endmacro
.macro  pop arg
        ld  %r14, arg
        add %r14 , 4 , %r14
.endmacro
add %r15 , 0 , %r16
```

```
main:   SETHI   C2100h, %r2
        SLL     %r2, 2, %r2
        add     %r2, 308h, %r2   ! almacena la dir
        ld %r2, %r1                del perifenco
        push %r1
        call    buscar_numeros
        pop     %r5     ! devuelve el 1° num
        pop     %r3     ! devuelve el 2do num
        add     %r0, %r0, %r4   ! %r4 tendrá el
                                   resultado
        add     %r5, %r3, %r4
        ST      %r4, %r2        ! cargo el resultado en
                                  el periferico.
        jmpl %r16+4, %r0

buscar_numeros:  pop %r10
                 ld %r10, %r12
                 SRL %r10, 16, %r10
                 push %r10
                 SLL %r12, 16, %r12
                 SRL %r12, 16, %r12
                 push %r12
                 jmpl %r15+4, %r0

.end
```

Ejercicio ②

Un procesador ARC esta ejecutando el siguiente código:

```
.begin
        .org 2048
        cte     .equ    19AA9h
        padron  .equ    10614
2048    add     %r0, cte, %r10
2052    sub     %r20, Padro, %r10
2056    st      %r10, [num]
2060    ld      %r10, [num]
2064    nor     %r10, %r20, %r0
2068    add     %r10, %r20, %r0
2072    jmpl    %r15, 4, %r0
        num     .dwb 1          4 by
.end
```

El pc en 814h =

1000000010100

11 10 9 8 7 6 5 4 3 2 1 0

Cuando el pc apunta a 814h se esta ejecutando:

add %r10, %r20, %r0

(a) La instruccion add su cod de op en 10, el op no esta en el arc, propongo 010111

La dir será entonces 1628 (x el decode)

decode 10 010111 00

microcódigo:

1628 = If R[IR[13]] then goto 1630

1629 : R[rd] ← addi (R[rs1], R[rs2]) GOTO 2047

1630: R[tempo] ← sextB (R[rs1])

1631 : R[rd] ← add (R[rs1], R[tempo])

GOTO 2047

(b) Al momento de la segunda micro

IR

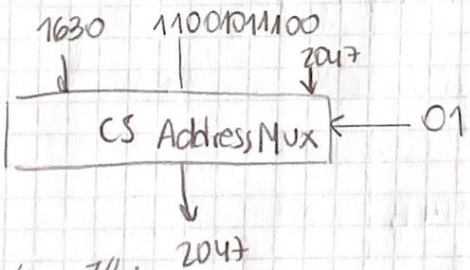
10 00000 010111 01010 0 00000000 10100

2^da micro = R[rd] ← add (R[rs1], R[rs2]) goto 2047

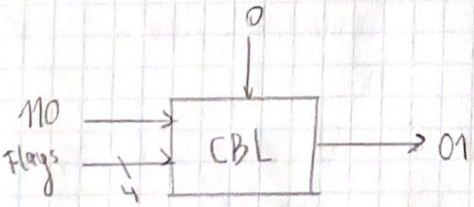
MIR

000000 1 000000 1 000001 00 1000 110 111111111111

Multiplexor de la memoria de control

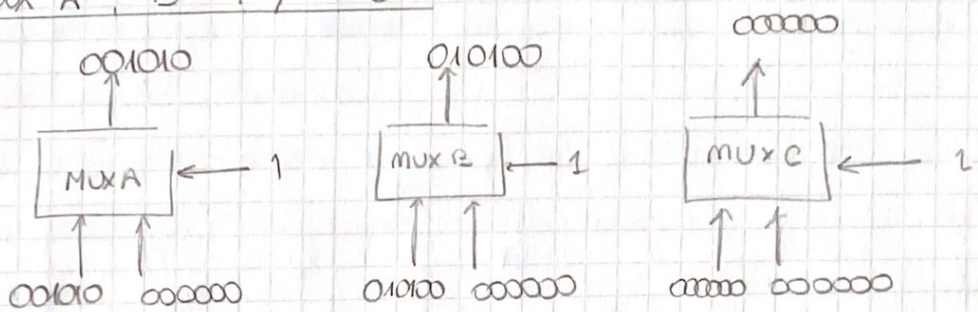


Logica de control de saltos

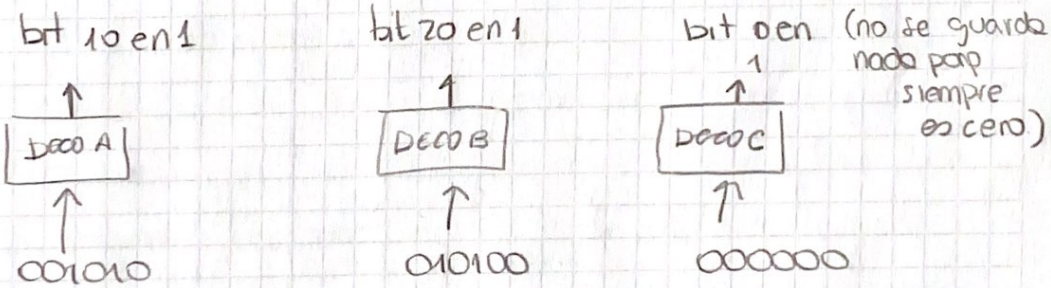


Registro $r10$ y $r20$ → no lo se

MUX A, B, y C



DECODIFICADORES A, B Y C



Ejercicio (3) Plantear una discusión respecto de la verasidad o falsedad de las sig afirmaciones

a) "La técnica de bancos entrelazados está orientada a compensar limitaciones inherentes a la tecnología de memoria estática"

b) "La implementación de variables en la pila permite minimizar los tiempos de acceso durante la ejecución de una subrutina"

c) "La tecnología de puentes ("bridges") no es comparable con la de "bus de sistema"

(a) Falso. Esta orientada a compensar la memoria dinámica, la cual es lenta y debe ser refrescada periodicamente ya que su funcionamiento está basado en capacitores. La técnica de bancos entrelazados tiene el objetivo de enmascarar el tiempo de refresco. Esto lo realiza distribuyendo direcciones de memorias contiguas en bancos distintos, de modo que al acceder a una dirección los demás bancos son refrescados paralelamente. Por lo tanto, al querer acceder a una dirección cercana, (lo que sucede en la mayoría de los casos por el principio de localidad), es muy probable que ésta esté en un banco que ya ha sido refrescado mientras yo realizaba el 1° accesso a memoria y de esta forma, se ve enmascarado el tiempo de refresco de los capacitores.

(b) Falso. La pila se encuentra en memoria principal, por lo tanto, si se le pasaran los parámetros a la subrutina mediante la pila, se perderá mucho tiempo para acceder a dicha información. Más rápido resultaría pasar los parámetros mediante registros, el inconveniente que se nos presenta en este caso, es que al no ser muchos los registros accesibles para el programador, en ocasiones donde son muchos parametros o donde muchos registros esten siendo utilizados por el resto del programa, y no sean suficientes para la cantidad de parametros de la subrutina, habrá que recurrir a utilizar la pila (stack) o por área reservada de memoria, que ambos generan que sea más lento el programa ya que acceden a la memoria principal.

(c) Falso. Los puentes llegan como solución al problema de que en el bus del sistema se comunican componentes de velocidades muy variadas, lo que puede generar un cuello de botella. Se agrupan entonces las componentes más veloces en un puente (el northbridge), como ser la RAM, CPU, y por otro lado, los componentes en el southbridge, como por ej USB, o placa de audio.

Sin embargo, la CPU, los puentes y los componentes siguen conectados por buses, por lo que se podría continuar hablando de un "BUS DE SISTEMA", solo que con esta tecnología la disposición de los elementos permite una mayor eficiencia impidiendo que componentes de velocidades MUY DISTINTAS usen el mismo BUS.