Facultad de Informática
Universidad Complutense de Madrid

EXAMEN FINAL – FUNDAMENTOS DE COMPUTADORES II

26 DE JUNIO 2024

El *Apollo Guidance Computer* (AGC) fue uno de los primeros computadores digitales implementados con circuitos integrados de silicio y fue diseñado por la NASA en 1966 para las misiones del programa Apollo. El AGC contaba con una memoria de 16 bits, que estaba protegida por un sistema de paridad parar evitar posibles errores en la información. El sistema funcionaba de la siguiente manera. Cada posición de memoria contaba con 15 bits de datos más un bit de paridad. El bit de paridad se añadía de tal forma que el número total de unos en la posición de memoria fuera un número impar (lo que se conoce como paridad impar). Al leer un dato de memoria, el computador contaba el número de unos que contenía. Si el número de unos era impar, se suponía que el dato era correcto. Pero si el número de unos era par, eso significaba que alguno de los bits había sido alterado por causas ajenas, como por ejemplo el efecto de la radiación espacial. En ese caso, se encendía un led llamado *parity_alarm* en el panel del control.

Se ha implementado una versión de este algoritmo en un RISC-V de repertorio completo, el cual usa la operación **`xor`** para determinar si el número de unos en una cierta posición de memoria *W* es par o impar. (Suponga en todos los ejercicios que la pseudointrucción **`la`** se traduce a una única instrucción **`addi`** en la fase de ensamblado, que la sección **`.text`** se ubica a partir de la dirección 0x0 de memoria y que la secciones **`.data`**, **`.bss`** se ubican consecutivamente a partir de la dirección 0x10000).

```
.global main

.equ N,16

.data
    W:      .half 0b0001110001110011
.bss
    parity_alarm: .space 1   # será 0 si W es correcto y 1 si hay error

.text
main:
    add     t2,zero,zero      # t2 es el acumulador
    addi    t1,zero,N         # t1 es el contador del bucle
    la      t0,W
    lh      s1,0(t0)          # carga el dato a chequear, W
loop:
    add     s2,zero,s1        # copia temporal del dato
    andi    s2,s2,0x00000001 # máscara para aislar el bit menos significativo
    xor     t2,t2,s2          # hace la xor del bit con el acumulador
    srli    s1,s1,1           # siguiente bit del dato
    addi    t1,t1,-1
    bne     t1,zero,loop
end_loop:
    la      t0,parity_alarm
    xori    t2,t2,0x00000001 # invierte el resultado ya que funciona con paridad impar
    sb      t2,0(t0)          # almacena el resultado en memoria

    j   .
.end
```

1. Monociclo [1 pt]. Si el código dado se ejecuta en el procesador monociclo, indique los valores que toman los siguientes registros y posiciones de memoria en los ciclos seleccionados. En el caso de la memoria nótese que en la tabla se proporcionan direcciones de bytes.

	Ciclo 6	Ciclo 10	Ciclo 16	Ejecución de `j .`
t0				
t1				
t2				
s1				
s2				
M[10000]				
M[10001]				
M[10002]	0x00			
M[10003]	0x00			

2. Monociclo [1 pt]. En un determinado instante de la ejecución del código en el procesador monociclo, la salida de la memoria de instrucciones tiene el valor 0x00900933. ¿Cuál es el contenido del PC en ese instante?

3. Multiciclo [1 pt]. El código dado se ejecuta hasta la instrucción j . (no incluida) en un procesador multiciclo con una frecuencia de 100 MHz y las siguientes características:

Instrucciones	Latencia
Aritmético-lógicas	4 ciclos
Load y store	5 ciclos
Saltos condicionales tomados	4 ciclos
Saltos condicionales no tomados	3 ciclos

a) Calcule el tiempo de ejecución.
b) Calcule los ciclos promedio por instrucción (CPI).
c) Determine el valor de la métrica MIPS.

4. Segmentado [1.5 pt]. En el procesador segmentado sin gestión de conflictos (pero con escritura del BR a mitad de ciclo):

a) Inserte en todo el programa las instrucciones **`nop`** necesarias para que se ejecute correctamente.
b) Realice el diagrama de ejecución de la primera iteración del bucle.
c) Calcule los ciclos promedio por instrucción (CPI) en la ejecución completa del bucle*.

A efectos de ejecución, considerar que **`lh`** se comporta como **`lw`**, **`sb`** como **`sw`**, **`xor`** como las instrucciones aritmético-lógicas de tipo R, y **`srli`** y **`xori`** como las instrucciones aritmético-lógicas de tipo I.

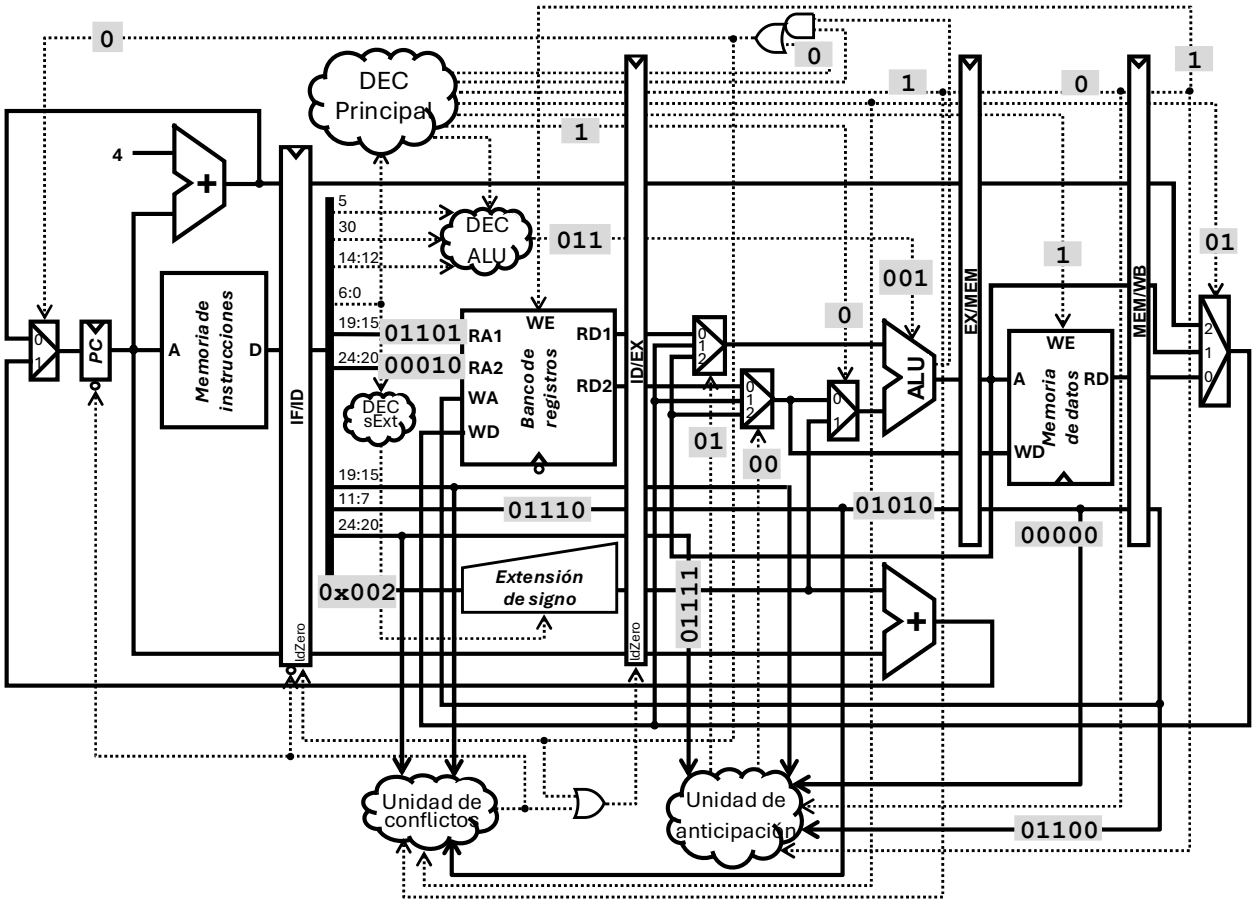
5. Segmentado [1.5 pt]. En el procesador segmentado con gestión de conflictos completa (escritura del BR a mitad de ciclo, unidad de anticipación y unidad de conflictos con predicción de salto no tomado):

a) Realice el diagrama de ejecución de la primera iteración del bucle
b) Calcule el número de ciclos que se tarda en ejecutar esa primera iteración*.
c) Calcule los ciclos promedio por instrucción (CPI) en la ejecución del programa completo (hasta el lanzamiento de la instrucción **`j .`** no incluida)*.

A efectos de ejecución, considerar que **`lh`** se comporta como **`lw`**, **`sb`** como **`sw`**, **`xor`** como las instrucciones aritmético-lógicas de tipo R, y **`srli`** y **`xori`** como las instrucciones aritmético-lógicas de tipo I.

* A la hora de contar los ciclos que tarda en ejecutarse un bloque de código, debe contarse desde el ciclo en que lanza la primera instrucción del bloque (incluido) hasta el ciclo que se lanza la siguiente instrucción al bloque (no incluido).

6. Segmentado [1.5 pt]. Suponer ahora que el procesador segmentado del problema anterior ejecuta un código distinto al del enunciado. En un determinado ciclo, los valores de algunas señales de control en este procesador son los que se muestran en la figura siguiente. Complete de manera razonada el fragmento de código con las cinco instrucciones que se están ejecutando en dicho momento. Es obligatorio justificar las respuestas.



add ☐ , x28, x29

☐ x30, 0(x31)

☐

☐

beq x10, x14, 32

7. Excepciones [1 pt]. Responda razonadamente a las siguientes preguntas suponiendo que ahora el código inicial del enunciado se ejecuta en el RISC-V segmentado de arquitectura reducida:

a) ¿La instrucción **`sb t2,0(t0)`** podría producir una excepción del tipo dirección de almacenaje desalineada?
b) La instrucción **`xor t2,t2,s2`** produce una cierta excepción al ejecutarse. ¿De qué tipo sería?
c) ¿En qué ciclo y etapa del procesador se produciría por primera vez la excepción del apartado anterior?
d) ¿En qué ciclo empezaría a ejecutarse la RTE después de producirse la excepción anterior?

8. Ensamblador [1 pt]. Convertir el programa inicial en una función, llamada *parity_check*, que reciba como parámetro de entrada el dato a chequear y que devuelva como parámetro de salida el resultado del chequeo. El código propuesto debe cumplir el convenio de llamada a funciones de RISC-V.

9. Ensamblador [0.5 pt]. Implementar un programa principal que llame a la función generada en el apartado anterior y que actualice el resultado devuelto por dicha función en memoria. El código propuesto debe cumplir el convenio de llamada a funciones de RISC-V.