# Ejercicio 1:

**Calcular analíticamente el CPI del siguiente código, utilizando las características de la microarquitectura explicadas en el libro:**

|  |
| --- |
| add $s0, $0, $0 # i = 0 add $s1, $0, $0 # sum = 0 addi **$t0**, $0, 1000 # $t0 = 1000 loop:  slt **$t1**, **$s0**, **$t0** # if (i < 1000), $t1 = 1, else $t1 = 0  beq **$t1**, $0, done # if $t1 == 0 (i >= 1000), branch to done  add $s1, $s1, $s0 # sum = sum + i  addi **$s0**, $s0, 1 # increment i  j loop done: |

## 1º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | add $s0, $0, $0 |  |

## 2º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | add $s1, $0, $0 |  |
| Decode | add $s0, $0, $0 |  |

## 3º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | addi $t0, $0, 1000 |  |
| Decode | add $s1, $0, $0 |  |
| Execute | add $s0, $0, $0 |  |

## 4º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | slt $t1, $s0, $t0 | Como esta en Fetch no hay problema. |
| Decode | addi $t0, $0, 1000 |  |
| Execute | add $s1, $0, $0 |  |
| Memory | add $s0, $0, $0 |  |

## 5º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | beq $**t1**, $0, done |  |
| Decode | slt $**t1**, $s0, $**t0** | $t0 No actualizado ¿$s0 actualizado?rdD = t1, rsD = s0, rtD=t0 |
| Execute | addi $**t0**, $0, 1000 | rtE = t0, MemtoRegE = 0 |
| Memory | add $s1, $0, $0 |  |
| Writeback | add $s0, $0, $0 |  |

## 6º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | add $s1, $s1, $s0 |  |
| Decode | beq $**t1**, $0, done | BranchD = 1**, rsD = $t1** (en un tipo I) |
| Execute | slt $**t1**, $s0, $**t0** | $t0 no actualizado. $s0 sí actualizado.RegWriteE = 1, **WriteRegE = $t1** ( en un tipo R el registro destino es rs), MemtoRegE = 0rd = t1, rs = s0, rt =t0 |
| Memory | addi $**t0**, $0, 1000 | *WriteRegM = t0* |
| Writeback | add $s1, $0, $0 | *WriteRegW = s1* |

Se comprueba las siguientes condiciones:

|  |
| --- |
| ***branchstall***= *BranchD*AND *[RegWriteE*AND ((*WriteRegE*== *rsD)*OR (*WriteRegE*== *rtD*))  OR [*MemtoRegM*AND ((*WriteRegM*== *rsD)*OR (*WriteRegM*== *rtD*))]*StallF* = *StallD* = *FlushE* = *lwstall* OR *branchstall* |

Como BranchD = 1, RegWrite = 1 y WriteRegE == rsD, se verifica que branchstall = 1 y por consiguiente

StallF = StallD = FlushE = 1. Actualiza t1 en el siguiente ciclo.

Por lo que en las etapas Fetch y Deco se paran las instrucciones y en Execute se introduce una burbuja, en el siguiente ciclo.

Por otro lado rtE = WriteRegM = t0 y RegWriteM = 1 -> ForwardingBE ->Actualiza t0

## 7º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | add $s1, $s1, $s0 |  |
| Decode | beq $**t1**, $0, done | rsD = $t1 (tipo I), ForwardAD = 1 |
| Execute | bubble |  |
| Memory | slt $**t1**, $s0, $t0 | WriteRegM = $t1 (tipo R), RegWriteM = 1. |
| Writeback | addi $t0, $0, 1000 |  |

|  |
| --- |
| ***ForwardAD*** = (*rsD*!=0) AND (*rsD*== *WriteRegM*) AND *RegWriteM****ForwardBD*** = (*rtD*!=0) AND (*rtD*== WriteRegM) AND *RegWriteM* |

Se cumple que ForwardAD = 1, por lo tanto $t1 en la etapa de Memory pasa directamente al operando 1 del ciclo de Decode y por tanto en beq $t1 si está actualizado.

## 8º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | addi $s0, $s0, 1 |  |
| Decode | add $s1, $s1, $s0 |  |
| Execute | beq $t1, $0, done | $t1 sí actualizado por Forwading ciclo anterior. |
| Memory | bubble |  |
| Writeback | slt $t1, $s0, $t0 |  |

## 9º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | j loop |  |
| Decode | addi $s0, $s0, 1 |  |
| Execute | add $s1, $s1, $s0 |  |
| Memory | beq $t1, $0, done |  |
| Writeback | bubble |  |

## 10º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | ---------------- | PC+4 no tomado |
| Decode | j loop | BranchD = 1  |
| Execute | addi $s0, $s0, 1 |  |
| Memory | add $s1, $s1, $s0 |  |
| Writeback | beq $t1, $0, done |  |

Una instrucción J en Decode implica que en el siguiente ciclo se introduce una burbuja, ya que durante la etapa Decode se está buscando la siguiente dirección de la instrucción siguiente, sin embargo en Fetch hay ya una instrucción que no se debería ejecutar.

## 11º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | slt $t1, $**s0**, $t0 |  |
| Decode | bubble |  |
| Execute | j loop |  |
| Memory | addi $**s0**, $s0, 1 | Cuando este en WB slt estará en Decode no hay problema. |
| Writeback | add $s1, $s1, $s0 |  |

add $s0, $0, $0 # i = 0

add $s1, $0, $0 # sum = 0

addi **$t0**, $0, 1000 # $t0 = 1000

loop:

 slt **$t1**, **$s0**, **$t0** # if (i < 1000), $t1 = 1, else $t1 = 0

 beq **$t1**, $0, done # if $t1 == 0 (i >= 1000), branch to done

 add $s1, $s1, $s0 # sum = sum + i

 addi **$s0**, $s0, 1 # increment i

 j loop

done:

Las tres primeras instrucciones del código solo se ejecutan una sola vez, en el bucle loop se ejecutará 1000 veces y el número de ciclos que realizará será 6 (5 debido a las instrucciones + 1 debido al bubble). Por último realizará una comprobación cuando llegue al final y realizará tres instrucciones (hasta que salte al loop)

El número de ciclos: **3** (tres primeras instrucciones) + **7\*1000** (7 ciclos por vuelta, ya que hay **dos burbujas** más 5 ciclos de las cinco instrucciones que hay) + 2 (en la última comprobación del bucle se ejecutan las isntrucciones slt y beq) + 5 (la última instrucción en ejecutar necesita cinco ciclos/4 ciclos para terminar de completarse) **= 7010 ciclos de reloj**.

El número de instrucciones son: 3 (tres primeras instrucciones) + 5\*1000 (5 instrucciones dentro del bucle) + 2 (2 instrucciones que se ejecutan en la comprobación de la condición de la última vuelta del bucle) = **5005 instrucciones ejecutadas**.

El CPI calculado es:

$$CPI=\frac{Ciclos de reloj}{Nº instrucciones}=\frac{7010}{5005}=1.4$$

# Ejercicio 2:

**Calcular analíticamente el CPI del código anterior, utilizando las características de la microarquitectura microAptiv:**

Se debe tener en cuenta las siguientes diferencias:

* MicroAptiv no se bloquea debido a una dependencia RAW entre una instrucción Aritmética-Lógica y una subsecuente instrucción beq. ¿Entonces la burbuja del sexto ciclo no se introducce? No se produce una burbuja.
* MicroAptiv implementa ramas retardadas. ¿A que se refiere? Después de cada salto la siguiente instrucción siempre se ejcuta.
* La memoria de la instrucción no es ideal en microAptiv, así, I $ falla introducir un cierto retraso. La memoria cache no es ideal, comete fallos.

|  |
| --- |
| add $s0, $0, $0 # i = 0 add $s1, $0, $0 # sum = 0 addi **$t0**, $0, 1000 # $t0 = 1000 loop:  slt **$t1**, **$s0**, **$t0** # if (i < 1000), $t1 = 1, else $t1 = 0  beq **$t1**, $0, done # if $t1 == 0 (i >= 1000), branch to done  add $s1, $s1, $s0 # sum = sum + i  addi **$s0**, $s0, 1 # increment i  j loop done: |

El CPI calculado es:

$$CPI=\frac{Ciclos de reloj}{Nº instrucciones}=\frac{7011}{7006}=1.0007$$

Número de instrucciones ejecutadas: NI= 3+7\*1000 (5 instrucciones + 2 nop que te incluye)+3 (2 instrucciones + otra nop) = 7006

Número de ciclos de reloj de la CPU: 3+7\*1000 + 6 (la última instrucciones pasa todas las estapas que son 6) + 2 (instrucciones nop por cada salto)= 7011

**NOTA: Los nops no se tienen en cuenta como isntruccion…**

# Ejercicio 3:

**Calcular empíricamente el CPI del código anterior, haciendo uso de los contadores de rendimiento:**

Analizando el código anterior con el programa proporcionado para la presente práctica, se obtienen los siguientes resultados en los displays de las Nexys DDR:

* Switch 0 en estado OFF: 7025 (Indica el número de ciclos)
* Switch 0 en estado ON: 7012 (Indica el número de instrucciones completadas)

$$CPI=\frac{Ciclos de reloj}{Nº instrucciones}=\frac{7025}{7012}=1.008$$

# Ejercicio 4:

**Comparar los resultados obtenidos del CPI usando microaqruitectura del libro, microarquitectura de microAptiv y resultados empíricos del programa usado.**

# Ejercicio 5:

**Calcular empiricamente el CPI del código anterior modificandole, haciendo uso de los contadores de rendicmiento:**

Se obtienen los siguientes resultados en los displays de las Nexys DDR:

* Switch 0 en estado OFF: 6026 (Indica el número de ciclos)
* Switch 0 en estado ON: 6014 (Indica el número de instrucciones completadas)

El CPI calculado es:

$$CPI=\frac{Ciclos de reloj}{Nº instrucciones}=\frac{5026}{5014}=1.0019$$

add $s0, $0, $0 # i = 0

add $s1, $0, $0 # sum = 0

addi $t0, $0, 1000 # $t0 = 1000

loop:

 slt $t1, $s0, $t0 #

 beq $t1, $0, done #

 add $s1, $s1, $s0 # sum = sum + i

 addi $s0, $s0, 1 # increment i

 j loop

done:

 " add $s0, $0, $0;"

 " add $s1, $0, $0;"

 " addi $t0, $0, 1000;"

 " loop:"

 " slt $t1, $s0, $t0;"

 " beq $t1, $0, done;"

 " add $s1, $s1, $s0;"

 " j loop;"

 **" addi $s0, $s0, 1;"**

 " done:"

 **" sub $s1, $s1, $s0;"**

# Ejercicio 6:

**Ejercicio avanzado:**

|  |
| --- |
|  lui $**t6**, 0x8000; addiu $t6, $**t6**, test\_array; addi $t1,$0,100; LOOP: lw $t2,0($t6); lw **$t3**,400($t6); sub **$t2**,**$t3**,**$t2**; sw $t2,800($t6); addi $**t1**,$t1,-1; addi $t6,$t6,4; bne $**t1**,$0,LOOP; addi $t1,$t1,100; |

## 1º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | lui $t6,0x8000; |  |

## 2º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | addiu $t6, $t6, test\_array; |  |
| Decode | lui $t6,0x8000; |  |

## 3º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | addi $t1,$0,10; |  |
| Decode | addiu $**t6**, $t6, test\_array; | rsD = $t6, rtD = $t6 |
| Execute | lui $**t6**,0x8000; | rtE = $t6 |

**NO METE BURBUJA YA QUE LUI ES PARA CREAR CONSTANTES**

## 4º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | lw $t2,0($t6); |  |
| Decode | addi $t1,$0,10; |  |
| Execute | addiu $t6, $**t6**, test\_array; | rsE = $t6, rtE = $t6 |
| Memory | lui $**t6**,0x8000; | WriteRegM = $t6 |

**ForwardAE = 10** -> rsE dif de 0 y rtE dif de 0, rsE == WriteRegM, RegWriteM = 1, entonces ForwardAE = 10

FORWARD BE = 00 DEBIDO A QUE EL OPERANDO B SE SACA DEL INMEDIATO, ALUSRCE = 1

## 5º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | lw $t3,400($t6); |  |
| Decode | lw $t2,0($**t6**); | t6 No actualizado rsD = t6 |
| Execute | addi $t1,$0,10; |  |
| Memory | addiu $**t6**, $t6, test\_array; | t6 Está actualizado. WriteRegM = t6 |
| Writeback | lui $t6,0x8000; |  |

~~rsD == WriteRegM == t6 y RegWriteM = 1 ForwardAD = 1~~

**MAL ESE FORWARD SOLO SE UTILIZA EN SALTOS CONDICIONALES, QUE REALIZAN A COMPARACION EN ETAPA D. EL RESTO DE INSTRUCCIONES, COMO EL LW, HACEN LA OPERACIÓN EN ETAPA E.**

## 6º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | sub **$t2**,**$t3**,**$t2**; |  |
| Decode | lw **$t3**,400($t6); | t6 esta actualizaod en la segunda mitad del ciclo rsD = t6 |
| Execute | lw $**t2**,0($**t6**); | rsE = t6 actualizado |
| Memory | addi $t1,$0,10; |  |
| Writeback | addiu $**t6**, $t6, test\_array; | *WriteRegW=t6* |

## ForwardAE= 01 -> t6 actualizado

## 7º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | sw $t2,800($t6); |  |
| Decode | sub **$t2**,**$t3**,**$t2**; | rsD = $t2, rtD = $t3, t2 y t3 no actualizado |
| Execute | lw **$t3**,400($t6); | rtE = $t3, MemtoRegE = 1 |
| Memory | lw $t2,0($t6); | rtM = $t2 t6 Actualizado |
| Writeback | addi $t1,$0,10; |  |

Lwstall = 1, debudo a que rtD= rtE=$t3 and MetmTorEGE =1 -> Introduce burbuja

## 8º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | sw $t2,800($t6); |  |
| Decode | sub **$t2**,**$t3**,**$t2**; | rdD = t2, rsD = t3, rtD = t2 Actualizado t2 |
| Execute | bubble |  |
| Memory | lw **$t3**,400($t6); | RegWrtieM = 1 *WriteRegM = t3* |
| Writeback | lw $t2,0($t6); |  |

T2 se escribe en el banco de registros en la primera mitad del ciclo y se lee en Decode en la segunda mitad del ciclo.

## 9º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | addi $t1,$t1,-1; |  |
| Decode | sw $**t2**,800($t6); | rtD = $t2, MemtoRegE = -- |
| Execute | sub **$t2**,$t3,$t2; | rdE = t2, rsE = t3, rtE = t2 |
| Memory | bubble |  |
| Writeback | lw $t3,400($t6); | RegWriteW=1, WriteRegW = t3 |

## ForwardAE = 01 rsE == WriteRegW -> Actualizo T3

## 10º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | addi $t6,$t6,4; |  |
| Decode | addi $t1,$t1,-1; | rsD = $t1,  |
| Execute | sw $**t2**,800($t6); | rtE = $t2, MemtoRegE = -- |
| Memory | sub **$t2**,$t3,$t2; | WriteRegM = t2 |
| Writeback | bubble |  |

rtE == WriteRegM, RegWriteM = 1 -> **ForwardBE = 10 -> Actualizo t2**

**t2 actualizado**

## 11º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | bne $**t1**,$0,LOOP; |  |
| Decode | addi $t6,$t6,4; |  |
| Execute | addi $t1,$t1,-1; |  |
| Memory | sw $t2,800($t6); |  |
| Writeback | sub $t2,$t3,$t2; |  |

## 12º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | bubble |  |
| Decode | bne $**t1**,$0,LOOP; | rsD = t1, rtD = 0 (Salto tomado) |
| Execute | addi $t6,$t6,4; |  |
| Memory | addi $**t1**,$t1,-1; | ReWriteM = 1, WriteRegM = t1 MemtoRegM = 0 |
| Writeback | sw $t2,800($t6); |  |

Hago forwarding de M a D:

WriteRegM == rsD y RegWriteM =1 -> **ForwardAD = 1.** T1 actualizado…

BNE es salto tomado introduce burbuja….

## 13º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | lw $t2,0($t6); |  |
| Decode | BUBBLE |  |
| Execute | bne $t1,$0,LOOP; | Salto tomado Se introduce BUBBLE |
| Memory | addi $t6,$t6,4; | WriteRegM = t6 |
| Writeback | addi $t1,$t1,-1; |  |

## 14º Ciclo:

|  |  |  |
| --- | --- | --- |
| Ciclos | Instrucción | Comentarios |
| Fetch | lw **$t3**,400($t6); |  |
| Decode | lw $t2,0($t6); |  |
| Execute | BUBBLE |  |
| Memory | bne $t1,$0,LOOP; | Salto tomado Se introduce BUBBLE |
| Writeback | addi $t6,$t6,4; | WriteRegM = t6 |

T6 actualizado

El CPI calculado es:

$$CPI=\frac{Ciclos de reloj}{Nº instrucciones}=$$

NI = 3 + 7\*100 +1 = 704

Ciclos = 3 +(7+2)\*99 + 8 (Una vuelta del ciclo (7 instrucciones) + \*\*una burbuja la que hay entre LW y SUB) +1 (addi)+4(ciclos necesarios para compeltar que salgan todas las intrucciones) =907

CPI = 907/704 = 1.289

2 burbujas (la que hay entre LW y SUB y la que introduce el salto bne por ser un salto tomada, execepto en el último paso)

\*\* En la última vuelta, el salto es no tomado, por lo que acierta y se va a la instrucción addi por lo que no introduce burbuja en esa última vuelta el bne.

**Para el procesador de microaptiv:**

**907**

**704**

El CPI calculado es:

$$CPI=\frac{Ciclos de reloj}{Nº instrucciones}=\frac{907}{704}=1.289$$

Número de instrucciones ejecutadas: NI= 3+7\*100 (5 instrucciones ) +1 = 704

Número de ciclos de reloj de la CPU: 3+(7+2 (1 burbuja + 1 nop))\*100 + 5 (la ultima instrucciones pasa todas las estapas que son 6) = 908

1 burbuja

Nop lo mete siempre que hay salto

**Para el procesador de microaptiv experimentalmente:**

Se obtienen los siguientes resultados en los displays de las Nexys DDR:

* Switch 0 en estado OFF: 923 (Indica el número de ciclos)
* Switch 0 en estado ON: 812 (Indica el número de instrucciones completadas)

El CPI calculado es:

$$CPI=\frac{Ciclos de reloj}{Nº instrucciones}=\frac{923}{812}=$$

**Codigo reordenado experimentalmente:**

Se obtienen los siguientes resultados en los displays de las Nexys DDR:

* Switch 0 en estado OFF: 719 (Indica el número de ciclos)
* Switch 0 en estado ON: 709 (Indica el número de instrucciones completadas)

 ".set noreorder;"

 " lui $t6, 0x8000;"

 " addiu $t6, $t6, test\_array;"

 " addi $t1,$0,100;"

 "LOOP: LW $t2,0($t6);"

 " LW $t3,400($t6);"

 **" ADDI $t1,$t1,-1;"**

 " SUB $t2,$t3,$t2;"

 " SW $t2,800($t6);"

 **" BNE $t1,$0,LOOP;"**

 " ADDI $t6,$t6,4;"

 " addi $t1,$t1,100;"

 ".set reorder;"