

## HOJA DE SOLUCIONES

**ATENCION:** Sólo se debe contestar a 4 de las 5 cuestiones. En el caso de contestar a las 5 sólo se considerarán las 4 primeras. Marcar en el cuadro superior la cuestión no contestada.

**C1.-** Responda breve y concisamente a las siguientes preguntas. Extender innecesariamente las respuestas o contestar a lo que no se pregunta se puntuará negativamente.

**C1.1.** Defina MIPS y MFLOPS. ¿Bajo qué circunstancia(s) será correcta la comparación de la velocidad de dos micros a través sólo de estas medidas?

**MIPS:** Mega (o Millones de) Instrucciones por Segundo.

**MFLOPS:** Mega (o Millones de) Operaciones en Coma Flotante por Segundo.

La comparación será correcta si ambos micros tienen el mismo juego de instrucciones (si tienen distinto juego de instrucciones, el número de instrucciones para ejecutar el mismo programa puede ser distinto).

**C1.2.** Defina en pocas palabras qué es el Northbridge (puente norte) e indique a qué dispositivo(s) está conectado.

El Northbridge es un dispositivo que centraliza los buses de alta velocidad y los conecta al microprocesador. Junto con el Southbridge forma el "chipset". Está conectado al microprocesador, al Southbridge y a periféricos de alta velocidad, como la memoria o la tarjeta gráfica.

**C1.3.** De los riesgos RAW, WAR y WAW, indique cuáles pueden bloquear/parar la ejecución en los siguientes casos:

1. Micro segmentado con ejecución en orden: **RAW.**
2. Micro con ejecución fuera de orden: **RAW, WAR y WAW.**
3. Técnica del marcador (*scoreboard*): **RAW, WAR y WAW.**
4. Técnica de Tomasulo: **RAW.**

**C1.4.** ¿Qué es el buffer de escritura en las cachés? ¿Cómo queda la fórmula de tiempo de acceso medio a caché para el caso de post-escritura con buffer de escritura ideal?

El buffer de escritura es una FIFO que guarda los datos a escribir desde la caché a la memoria principal (o siguiente nivel de caché) hasta que realmente se escriben, permitiendo que no se bloquee la ejecución en las escrituras de caché a memoria principal. La fórmula queda:

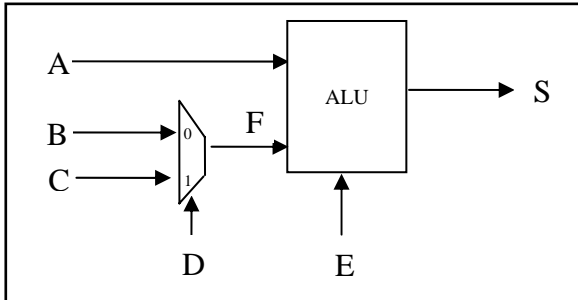
$$\overline{t}_{acc} = t_c + (1-H) \cdot t_B^{M \rightarrow C} \text{ eliminando el término } (1-H) \cdot w_M \cdot t_B^{C \rightarrow M}.$$

**C1.5.** Indique los principales inconvenientes de los procesadores VLIW (sólo enunciarlos).

Tamaño del código, mantener la compatibilidad del código objeto, ocupar todas las unidades de ejecución y conseguir una planificación eficiente por parte del compilador (en especial por accesos a memoria y saltos).

## E.P.S.- UAM.- Arquitectura e Ingeniería de Computadores, 26-Enero-2008

**C2.-** Implementar mediante código VHDL el esquema que se especifica a continuación en el dibujo:  
La entidad está completa, pero el alumno deberá completar buena parte de la arquitectura.



Se trata de un circuito que tiene cinco entradas y una salida. El circuito debe proporcionar a la salida el resultado de la operación la cual vendrá definida por el código de control de la señal E. La entrada A y la señal F son los operandos sobre los que realizar la operación según la siguiente tabla:

E	Operación	E	Operación
00	A + F	10	A << 2
01	A > F	11	A or F

La operación A > F proporcionará un uno (número) a la salida "S" si A es menor que F y un 0 en caso contrario.

**Importante:** Se han de crear necesariamente procesos o sentencias concurrentes por separado que modelen los 2 componentes especificados en el dibujo.

```
-- Examen ARQ Enero 2008
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity mi_componente is
    port(
        A : in STD_LOGIC_VECTOR(31 downto 0);
        B : in STD_LOGIC_VECTOR(31 downto 0);
        C : in STD_LOGIC_VECTOR(31 downto 0);
        D : in STD_LOGIC;
        E : in STD_LOGIC_VECTOR(1 downto 0);
        S : out STD_LOGIC_VECTOR(31 downto 0)
    );
end mi_componente;

architecture examen of mi_componente is
    -- DECLARACIONES A COMPLETAR POR EL ALUMNO
    Signal F: std_logic_vector(31 downto 0);

begin
    -- CUERPO DE LA ARQUITECTURA A COMPLETAR POR EL ALUMNO

    F <= C when D='1' else B;

    process (A,F,E)
    begin
        case E is
            when "00" =>
                S <= A + F;
            when "01" =>
                if A>F then
                    S <= x"00000001";
                else
                    S <= x"00000000";
                end if;
            when "10" =>
                S <= A(29 downto 0) & "00";
            when others =>
                S <= A or F;
            end case;
        end process;
    end examen;
```

## E.P.S.- UAM.- Arquitectura e Ingeniería de Computadores, 26-Enero-2008

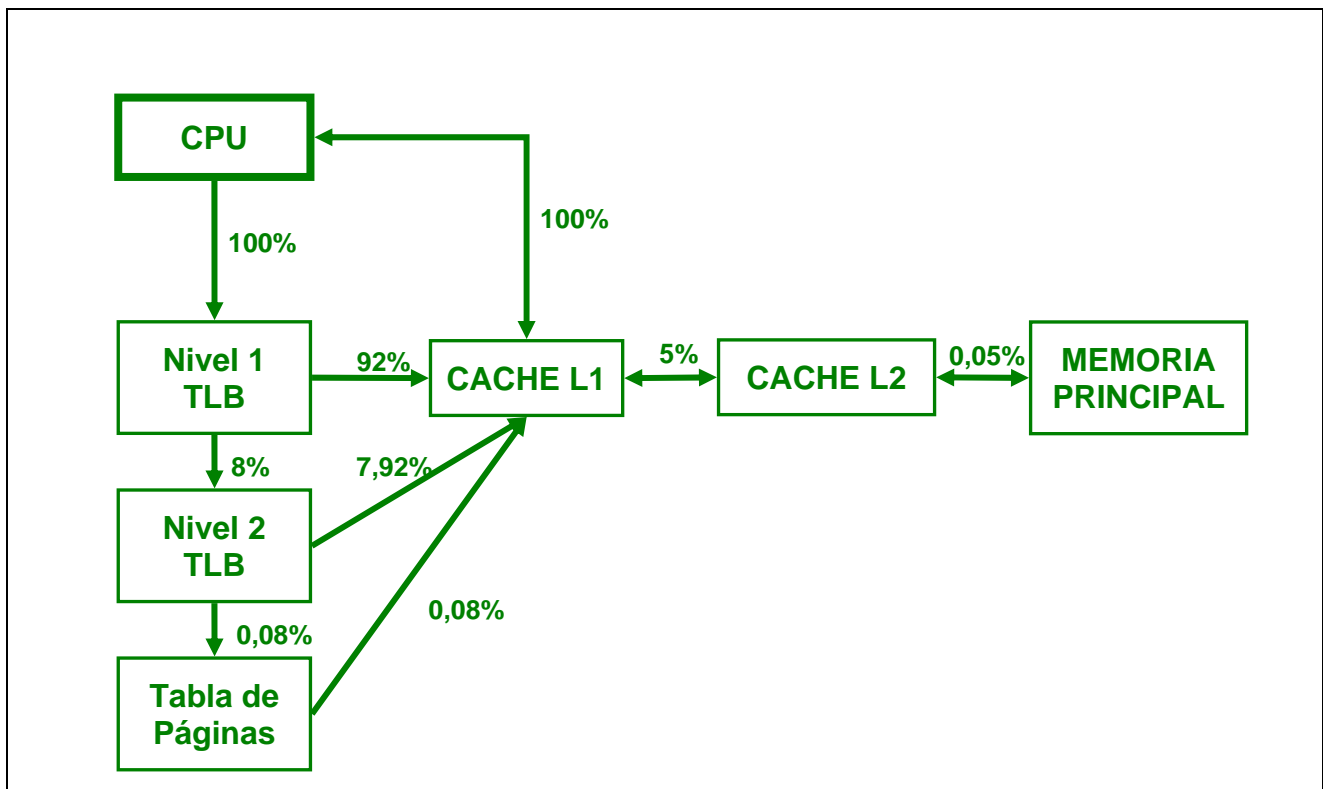
**C3.-** Un cierto sistema ordenador tiene la siguiente arquitectura asociada a la jerarquía de memoria: TLB de nivel 1, TLB de nivel 2, Tabla de Páginas, Caché (real) con dos niveles L1 y L2 y Memoria Principal. De la arquitectura se conocen los siguientes parámetros:

La tasa de aciertos para el nivel 1 del TLB es del 92% sabiendo que en caso de acierto no se considera penalización alguna. La tasa de aciertos para el nivel 2 del TLB es del 99% y el tiempo de acceso a este nivel penaliza con 1 ciclo, por último cada vez que se accede a la tabla de páginas se penaliza con 10 ciclos. En la Tabla de Páginas se encuentran todas las direcciones de página virtual y todas ellas se encuentran en memoria real.

La tasa de aciertos a la caché L1 es del 95% y el tiempo de acceso de 1 ciclo. En caso de fallo, si el dato está en el nivel de caché L2, se penaliza con 10 ciclos adicionales pero si hay que acceder a la memoria principal, la penalización es de 100 ciclos y esto ocurre con una probabilidad del 10%.

Se pide:

**a)** Dibujar un diagrama de flujo mostrando para un acceso a memoria desde la CPU, todas las transiciones posibles entre los diferentes elementos de la jerarquía, señalando para cada transición, la probabilidad absoluta de cada evento.



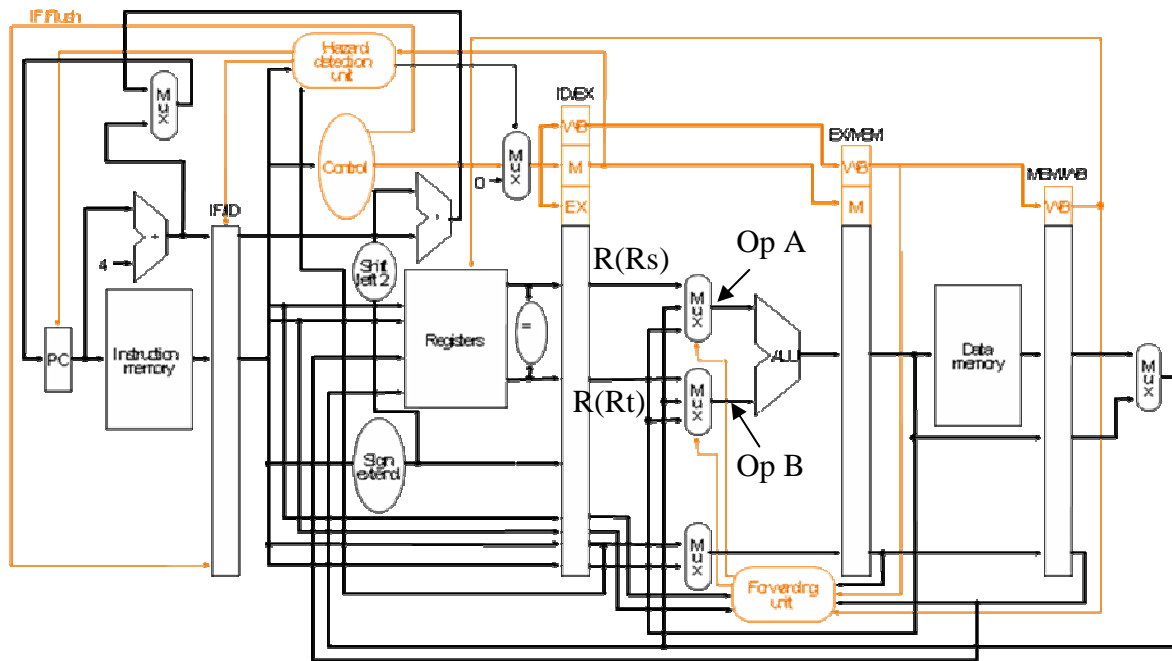
**b)** Dar un valor para el tiempo de acceso promedio en un acceso a memoria, suponiendo que el bloque sea de una palabra.

$$t_{ACC} = F_{TLB1} * \{t_{TLB2} + F_{TLB2} * t_{TPAG}\} + t_{CACHE} + F_{CACHE} * \{t_{MEM} + F_{MEM} * t_{DISCO}\}$$

$$t_{ACC} = 0,08 * \{1 + 0,01 * 10\} + 1 + 0,05 * \{10 + 0,1 * 100\}$$

$$t_{ACC} = 0,09 + 1 + 1 = 2,09 \text{ ciclos}$$

**C4.-** El microprocesador estudiado en el tema 4 y realizado en las prácticas de la asignatura posee una unidad de adelantamiento de datos (*data forwarding*) que permite adelantar los datos de las etapas MEM y WB a la etapa de ejecución (EX):



Los registros de las diferentes etapas se llaman PC, IF/ID, ID/EX, EX/MEM y MEM/WB respectivamente. La lógica implementada en la unidad de adelantamientos es:

<b>Adelantamientos desde la etapa MEM</b>	<b>Adelantamientos desde la etapa WB</b>
<b>if</b> (EX/MEM.EscrReg <b>and</b> EX/MEM.Reg.Rd $\neq$ 0 <b>and</b> EX/MEM.Reg.Rd = ID/EX.Reg.Rs) <b>then</b> AnticiparA = 10 <b>else</b> AnticiparA = 00  <b>if</b> (EX/MEM.EscrReg <b>and</b> EX/MEM.Reg.Rd $\neq$ 0 <b>and</b> EX/MEM.Reg.Rd = ID/EX.Reg.Rt) <b>then</b> AnticiparB = 10 <b>else</b> AnticiparB = 00	<b>if</b> (MEM/WB.EscrReg <b>and</b> MEM/WB.Reg.Rd $\neq$ 0 <b>and</b> EX/MEM.Reg.Rd $\neq$ ID/EX.Reg.Rs <b>and</b> MEM/WB.Reg.Rd = ID/EX.Reg.Rs) <b>then</b> AnticiparA = 01 <b>else</b> AnticiparA = 00  <b>if</b> (MEM/WB.EscrReg <b>and</b> MEM/WB.Reg.Rd $\neq$ 0 <b>and</b> EX/MEM.Reg.Rd $\neq$ ID/EX.Reg.Rt <b>and</b> MEM/WB.Reg.Rd = ID/EX.Reg.Rt) <b>then</b> AnticiparB = 01 <b>else</b> AnticiparB = 00

Se ejecutan en el procesador las siguientes 4 instrucciones

- I1: Add r1, r2, r3
- I2: Sub r5, r1, r6
- I3: And r6, r5, r1
- I4: Add r4, r1, r3

## E.P.S.- UAM.- Arquitectura e Ingeniería de Computadores, 26-Enero-2008

Determine el valor de las señales correspondientes a la ejecución del anterior código ensamblador. Complete en la tabla adjunta donde se puede ver la evolución de las instrucciones dentro del pipeline. Se debe completar solamente las señales indicadas de la etapa EX en los ciclos 4, 5 y 6.

Ciclo	IF	ID	EX	MEM	WB
1	Add <u>r1</u> , r2, r3	XXX r10, r1, r2			
2	Sub <u>r5</u> , <u>r1</u> , r6	Add <u>r1</u> , r2, r3	XXX r10, r1, r2		
3	And r6, <u>r5</u> , <u>r1</u>	Sub <u>r5</u> , <u>r1</u> , r6	Add <u>r1</u> , r2, r3	XXX r10, r1, r2	
4	Add r4, <u>r1</u> , r3	And r6, <u>r5</u> , <u>r1</u>	Sub <u>r5</u> , r1, r6 <b>(1)</b> anticiparA = <b>10</b> anticiparB = <b>00</b> ID/EX.Reg.Rs = <b>r1</b> ID/EX.Reg.Rt = <b>r6</b> EX/MEM.EscrReg = <b>1</b> EX/MEM.Reg.Rd = <b>r1</b> MEM/WB.EscrReg = <b>1</b> MEM/WB.Reg.Rd = <b>r10</b>	Add <u>r1</u> , r2, r3	XXX r10,r1, r2
5		Add r4, <u>r1</u> , r3	And r6, <u>r5</u> , <u>r1</u> <b>(2)</b> anticiparA = <b>10</b> anticiparB = <b>01</b> ID/EX.Reg.Rs = <b>r5</b> ID/EX.Reg.Rt = <b>r1</b> EX/MEM.EscrReg = <b>1</b> EX/MEM.Reg.Rd = <b>r5</b> MEM/WB.EscrReg = <b>1</b> MEM/WB.Reg.Rd = <b>r1</b>	Sub <u>r5</u> , <u>r1</u> , r6	Add <u>r1</u> , r2, r3
6			Add r4, <u>r1</u> , r3 <b>(3)</b> anticiparA = <b>00</b> anticiparB = <b>00</b> ID/EX.Reg.Rs = <b>r1</b> ID/EX.Reg.Rt = <b>r3</b> EX/MEM.EscrReg = <b>1</b> EX/MEM.Reg.Rd = <b>r6</b> MEM/WB.EscrReg = <b>1</b> MEM/WB.Reg.Rd = <b>r5</b>	And r6, <u>r5</u> , <u>r1</u>	Sub <u>r5</u> , <u>r1</u> , r6
7				Add r4, <u>r1</u> , r3	And r6, <u>r5</u> , <u>r1</u>
8					Add r4, <u>r1</u> , r3

Utilice este recuadro en caso que necesite realizar alguna aclaración

- (1) Se adelanta el operando A (r1) desde la etapa WB
- (2) Se adelanta el operando A (r5) desde la etapa MEM y el operando B (r1) desde la etapa WB
- (3) No es necesario adelantar el dato del registro r1, ya que este fue escrito en el ciclo 5 y leído en ese mismo en la etapa ID

## E.P.S.- UAM.- Arquitectura e Ingeniería de Computadores, 26-Enero-2008

**C5.-** Se tiene un microprocesador segmentado en las siguientes 5 etapas. **IF:** captura de instrucción. **ID:** decodificación de instrucción, detección de riesgos y captura de operandos. **EX:** ejecución en ALU, cálculo de la dirección de salto y de la dirección de acceso a memoria de datos. **M:** acceso a memoria de datos y resolución de la condición de salto. **W:** escritura en registros internos. El banco de registros es tal que permite en el mismo ciclo de reloj escribir un dato y luego leerlo. Se utiliza arquitectura Harvard y no hay detenciones en memoria. La ejecución es siempre en orden. El código a ejecutar es el siguiente, en el que se sabe que el salto condicional es efectivo:

Instr./Etiqu.	Ensamblador	Explicación
I1	ADD R1, R2, R3	; $R1 = R2 + R3$
I2	LOAD R4, 0(R1)	; $R4 = M[0 + R1]$
I3	MUL R5, R4, R4	; $R5 = R4 \cdot R4$
I4	BNE R1, R4, L1	; Si $R1 \neq R4$ , salta a L1 (sí se salta)
I5	SUB R5, R5, R2	; $R5 = R5 - R2$
I6	ADD R7, R4, R5	; $R7 = R4 + R5$
I7 / L1:	ADD R6, R1, R5	; $R6 = R1 + R5$
I8:	OR R1, R7, R8	; $R1 = R7 \text{ or } R8$
I9:	ST R6, 0(R1)	; $M[0 + R1] = R6$

a) Suponiendo que no hay adelantamiento de datos (*forwarding*) y que los saltos condicionales se predicen como no efectivos, rellenar el siguiente cronograma.

Instrucciones	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
ADD R1, R2, R3	IF	ID	EX	M	W																			
LOAD R4, 0(R1)		IF	ID	ID	ID	EX	M	W																
MUL R5, R4, R4			IF	IF	IF	ID	ID	ID	EX	M	W													
BNE R1, R4, L1						IF	IF	IF	ID	EX	M	W												
SUB R5, R5, R2									IF	ID	ID	-												
ADD R7, R4, R5										IF	IF	-												
L1: ADD R6, R1, R5												IF	ID	EX	M	W								
OR R1, R7, R8													IF	ID	EX	M	W							
ST R6, 0(R1)														IF	ID	ID	ID	EX	M	W				

b) Suponiendo que hay adelantamiento de datos implementado de la mejor forma posible y que los saltos condicionales se predicen como efectivos, rellenar el siguiente cronograma. Indique sobre el mismo los adelantamientos de datos realizados (flecha de la unidad que genera el dato a la que lo utiliza).

Instrucciones	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
ADD R1, R2, R3	IF	ID	EX	M	W																			
LOAD R4, 0(R1)		IF	ID	EX	M	W																		
MUL R5, R4, R4			IF	ID	ID	EX	M	W																
BNE R1, R4, L1				IF	IF	ID	EX	M	W															
SUB R5, R5, R2						IF	ID	EX	-															
ADD R7, R4, R5							IF	ID	-															
L1: ADD R6, R1, R5								IF	ID	EX	M	W												
OR R1, R7, R8									IF	ID	EX	M	W											
ST R6, 0(R1)										IF	ID	EX	M	W										

Aunque no entran más instrucciones de la serie Next, N+1 tras la fase EX del salto, las que están dentro siguen su ejecución hasta saber si se toma el salto o no (tras la fase M del salto se hace flushing de las que no debían tomarse).

## E.P.S.- UAM.- Arquitectura e Ingeniería de Computadores, 26-Enero-2008

**P1.-** La familia de microprocesadores DSP C64x de Texas Instruments (arquitectura VLIW) está segmentada en 3 etapas (fetch, decode, execute) por la que pasan todas las instrucciones. La etapa de captura (fetch) a su vez está segmentada en 4 etapas:

**PG:** Program address generate (generación de dirección)

**PS:** Program address send (envío de la dirección a la caché de instrucciones)

**PW:** Program access ready wait (estado de espera)

**PR:** Program fetch packet receive (recepción de la instrucción)

La etapa de decodificación a su vez tiene dos etapas de segmentación

**DP:** Instruction dispatch (donde se envía a la unidad funcional que corresponda)

**DC:** Instruction decode (decodificación de la instrucción)

Finalmente la ejecución está segmentada en 5 etapas (**E1, E2, E3, E4 y E5**) y se necesitan diferente cantidad de ciclos de ejecución dependiendo de la instrucción. Todas las instrucciones capturan sus operandos en la etapa E1. Las instrucciones no pueden capturar un registro en el mismo ciclo que otra lo escribe.

Posee dos rutas de datos con 4 unidades funcionales cada una. Las unidades funcionales se llaman:

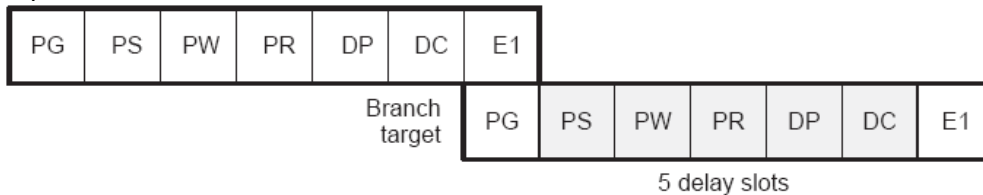
(L => Aritmética y lógica; S => Aritmética y saltos; M => Multiplicación y D => Load/store). En definitiva existen ocho unidades funcionales. Dos unidades de aritmética (suma/resta) y lógica (L1 y L2). Dos unidades de aritmética (suma/resta), desplazamiento y saltos (S1 y S2). Dos unidades de multiplicación (M1 y M2). Dos unidades de load/store (D1 y D2)

Además posee dos bancos de registros internos de 32 bits (total 64 registros). El procesador captura 8 instrucciones de 32 bits por ciclo de instrucción. Para aumentar las prestaciones, el procesador no tiene ninguna lógica de detección de riesgos (ni de datos ni de control), los que deben ser resueltos por el compilador. En la figura se muestra un esquema de las etapas de segmentación.

← Fetch →				← Decode →		← Execute →				
PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5

Respecto a la ejecución de algunas instrucciones:

- Los saltos se resuelven en E1 y modifican la dirección de programa en la (etapa PG) generando 5 ciclos perdidos.



- Las instrucciones de Load requieren las 5 etapas de ejecución escribiendo el resultado en la última etapa (E5)
- Las instrucciones de store escriben en la caché de datos en E3.
- Las instrucciones de suma calculan y escriben registros en E1
- Las de multiplicación escriben los registros en E2.

**a.** Determinar el CPI de un programa que tiene sólo sumas y restas, con otro que tenga sólo multiplicaciones y con un tercero que solo posea instrucciones de load y store. Sin considerar riesgos.

Si solo sumas, puede ejecutar 4 sumas por ciclo =>  $CPI = 1/4 = 0,25$

Si solo multiplicaciones, puede ejecutar 2 sumas por ciclo =>  $CPI = 1/2 = 0,5$

Si solo load/store, puede ejecutar 2 load/store por ciclo =>  $CPI = 1/2 = 0,5$

**b.** Cuanto tiempo tardaría el código C que se adjunta si **no hay límites para el tamaño del código y se pretende la máxima velocidad posible**. El procesador funciona a una velocidad de 200 MHz. Puede obviar los ciclos de llenado del pipeline para el cálculo. Justifique su respuesta usando el sitio adecuado y/o la tabla que representa las 8 unidades funcionales. Utilice los nemónicos LD, ST, ML, AD para las instrucciones de load, store, multiplicación y suma. Para las instrucciones de load y store (LD, ST) suponga que están definidas constantes A,B,C,D que apuntan a la primera posición de cada vector. Así LD r1, [A] es cargar A(0) y LD r2, [C+4] es cargar C(1). También existen las constantes X, Z que apuntan a las direcciones de X y Z respectivamente.

## E.P.S.- UAM.- Arquitectura e Ingeniería de Computadores, 26-Enero-2008

Z= 0; X=0

```
for (int i=1; i < 4; i++) {
    Z += A[i] * B[i] * C[i] * D[i];
    X += A[i] + B[i] + C[i] + D[i]; }
```

Justifique su respuesta, puede utilizar la tabla que representa las 8 unidades para ello:

D1	D2	M1	M2	L1	L2	S1	S2
Ld r1, [A+4]	Ld r2, [B]	-	-	-	-	-	-
Ld r3, [C+4]	Ld r4, [D]	-	-	-	-	-	-
Ld r5, [A+8]	Ld r6, [B+8]	-	-	-	-	-	-
Ld r7, [C+8]	Ld r8, [D+8]	-	-	-	-	-	-
Ld r9,[A+12]	Ld r10,[B+12]	-	-	-	-	-	-
Ld r11,[C+12]	Ld r12, [D+12]	ML r15, r1, r2	-	Ad r21, r1, r2	-	-	-
-	-	ML r16, r3, r4	-	Ad r22, r3, r4	-	-	-
-	-	ML r17, r5, r6	-	Ad r23, r5, r6	Ad r2,r21,r22	-	-
-	-	ML r18, r7, r8	ML r1,r15,r16	Ad r24, r7, r8	-	-	-
-	-	ML r19, r9, r10	-	Ad r25,r9,r10	Ad r4,r23,r24	-	-
-	-	ML r20, r11,r12	ML r3,r17,r18	Ad r26,r11,r12	-	-	Ad r30,r2,r4
-	-	-	-	-	Ad r6,r25,r26	-	-
-	-	-	ML r5,r19,r20	-	-	Ad r31,r1,r3	Ad r30,r30,r6
-	St r30, [X]	-	-	-	-	-	-
-	-	-	-	-	-	Ad r31,r31,r5	-
St r31, [Z]	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

Número de ciclos y tiempo de ejecución para el código:

Desenrollando completamente las 3 iteraciones se tiene:

5 etapas de pipeline (3 de fetch + 2 de decode) para comenzar a ejecutar (es válido tenerlo en cuenta o no). Además hay 5 ciclos del primer load y 3 bucles de 2 ciclos de retardo por iteración y 7 ciclos finales hasta terminar (1 etapa pipe mult + 2 último mult + 1 sum + 3 store)

Total= 5 ciclos del 1<sup>er</sup> load + 3\*2 ciclos mult + 1 pipe mult + 2 últ.mult + 1 sum + 3 store = 18 ciclos

Tiempo = 1/200 MHz \* 18 ciclos = 5 \* 18 = 90 ns

Si se tiene en cuenta el llenado del pipeline: 23 ciclos y 115 ns

c. Número de ciclos y tiempo necesarios para la ejecución del código anterior pero para cien iteraciones en vez de tres en el bucle for. (*for (int i=1; i < 101; i++)*)

Desenrollando completamente las 100 iteraciones y utilizando el razonamiento anterior, se tiene:

Total= 5 ciclos 1<sup>er</sup> load + 100\*2 ciclos mult + 1 pipe mult + 2 últ.mult + 1 sum + 3 store = 212 ciclos

Tiempo = 1/200 MHz \* 212 ciclos = 5 \* 212 = 1060 ns = 1,06 microseg



## E.P.S.- UAM.- Arquitectura e Ingeniería de Computadores, 26-Enero-2008

**P2.-** Sea un sistema de memoria virtual, que opera con datos de 32 bits. El sistema dispone de una unidad TLB, completamente asociativa de 8 entradas y de una unidad caché virtual unificada, asociativa de dos vías, con 16 entradas por vía, 16 bytes por bloque y un algoritmo LRU para las sustituciones. Se facilitan algunos contenidos relevantes en el estado inicial del TLB, de la caché y de memoria.

**Nota1:** sólo en algunas ocasiones, el contenido facilitado de la caché es relevante y el contenido de los descriptores nunca muestra los bits de control.

**Nota2:** Como en los ejemplos del curso, el byte más a la derecha en el bloque de la caché, corresponde a la dirección más baja de memoria y la notación que se utiliza es “*big endian*”.

A continuación, se ejecuta el código adjunto:

PC	Instrucción	Comentario	TLB
0A040	MOV R4, 0	; R4 <= 0	FC5 02
0A044	MOV R1, 2	; R1 <= 2	A30 B0
0A048	MOV R2, A305Ch	; R2 <= A305C <sub>16</sub>	---
0A04C	L1: LOAD R3, R2, R0	; R3 <= MEM[R2+0]	---
0A050	ADD R4, R4, R3	; R4 <= R4 + R3	---
0A054	ADD R2, R2, 4	; R2 <= R2 + 4	A31 B1
0A058	ADD R1, R1, -1	; R1 <= R1 - 1	---
0A05C	BNZ L1	; Salta a L1 si Z = 0	0BC F5
0A060	JMP FIN	; Salta a FIN	0A0 3C
----			
0A0E0	FIN: STORE R4, R2, A0h	; R4 => MEM[R2 + A0 <sub>16</sub> ]	
0A0E4	HALT		

UNIDAD CACHE				
VIA 1			VIA 2	
TAG	BLOQUE		TAG	BLOQUE
0 =>	032	B0030200 A02A305C A0300002 A0400000	BC0	A0A2D4D5 88AA3460 45803D59 95203580
1 =>				
4 =>	0A0	0020300B C503A02A 2000001A 0000004A	3F2	A0A3D420 40FFDC60 458032FF 45203580
5 =>	4B2	30405060 B0030200 3345AACF 2442B890	A30	10102020 30405060 708090A0 B0C0D0E0
6 =>	A03	A02A305C A0300002 5060D0E0 00FF004A	ABC	2442B890 40FFDC60 88AA3460 88AA3460
E =>	0A0	10102020 30405060 708090A0 5060D0E0	3F0	A0A3D420 40FFDC60 458032FF 45203580
F =>	3FF	A0A3D420 40FFDC60 458032FF 45203580	4C0	5060D0E0 88AA3460 5060D0E0 5060D0E0

MEMORIA															
DIR	VAL	DIR	VAL	DIR	VAL	-	DIR	VAL	-	DIR	VAL	DIR	VAL	DIR	VAL
3C50	50	3C58	51	3C60	B0		A030	F4		B058	10	B060	40	B068	20
3C51	04	3C59	01	3C61	0F		A031	DE		B059	20	B061	40	B069	40
3C52	04	3C5A	01	3C62	FF		A032	C0		B05A	30	B062	A0	B06A	3F
3C53	03	3C5B	FF	3C63	80		A033	0A		B05B	40	B063	A0	B06B	20
3C54	51	3C5C	F0	3C64	3F		A034	45		B05C	50	B064	C0	B06C	4B
3C55	02	3C5D	0F	3C65	40		A035	74		B05D	60	B065	F5	B06D	BC
3C56	02	3C5E	FF	3C66	55		A036	22		B05E	70	B066	D6	B06E	D0
3C57	04	3C5F	F6	3C67	D0		A037	80		B05F	80	B067	FF	B06F	00

Con la información facilitada, se pide:

a) Justificando brevemente la respuesta, el tamaño de página así como el de la memoria virtual y real.

En el código del programa, se identifica que las direcciones (virtuales) son de 20 bits. En el TLB, el campo asociado a la página virtual es de 12 bits y el asociado al marco de página real 8, por tanto el offset es de 8 bits. Las respuestas correctas son

El tamaño de página es de  $2^8 = 256$  bytes

La dirección virtual tiene 20 bits, luego la memoria virtual es de  $2^{20} = 1$  Mbytes

La dirección real tiene 16 bits, luego la memoria real es de  $2^{16} = 64$  kbytes

## E.P.S.- UAM.- Arquitectura e Ingeniería de Computadores, 26-Enero-2008

b) Justificando brevemente la respuesta, el porcentaje de aciertos al ejecutar el código completo

Las tres primeras instrucciones se ejecutan una sola vez y están presentes en la caché: 0 Fallos, 3 Aciertos.  
El bucle se ejecuta dos veces (5x2) y hay un fallo en la dirección 0A050 la primera vez: 1F, 9A  
La instrucción 0A060, se ejecuta una sola vez y falla en el acceso: 1F, 0A.  
Las dos últimas instrucciones se ejecutan una sola vez y están presentes en la caché: 0F, 2A  
En el bucle, se accede una vez a datos por cada iteración (2xLOAD), la primera vez se accede a la dirección (A305C) y se acierta, pero en la segunda dirección (A3060) se falla: 1F, 1A.  
Fuera del bucle hay un único acceso a datos (1xSTORE), a la dirección (A3104) que no se encuentra en caché: 1F, 0A.  
En total, hay 4 Fallos y 15 Aciertos, de los 19 accesos a memoria (16 instrucciones y 3 datos).  
La tasa de aciertos es  $15/19 = 78,9\%$

c) Suponiendo que la caché es virtual, indicar el valor en hexadecimal de los registros R2, R3 y R4, tras la ejecución del código.

	1ª iter. (R1 = 2)	2ª iter. (R1 = 1)	Resultado (R1=0)
Valor de R2:	000A305C <sub>16</sub>	000A3060 <sub>16</sub>	000A3064 <sub>16</sub>
Valor de R3:	20201010 <sub>16</sub>	4040A0A0 <sub>16</sub>	4040A0A0 <sub>16</sub>
Valor de R4:	20201010 <sub>16</sub>	6060B0B0 <sub>16</sub>	6060B0B0 <sub>16</sub>