Supongamos una arquitectura con planificación dinámica y especulación y con las siguientes características

- Los datos NO se pueden escribir en el CDB y utilizar por una instrucción dependiente en el mismo ciclo
- Las instrucciones de enteros utilizan el CDB
- Las instrucciones de LD y SD utilizan la unidad funcional de cálculo de dirección efectiva (EA)
- La instrucción BGEZ necesita 3 ciclos para conocer el resultado del salto, y EA, y utiliza su propia unidad para calcular la EA del salto, que estará disponible para realizar el ISSUE 5 ciclos después del ISSUE de BGEZ. Por ejemplo, si se realiza el ISSUE de BGEZ en el ciclo 1, y hay fallo en la predicción, hasta el ciclo 6 (como pronto) no podré hacer el ISSUE de la instrucción correcta.
- La instrucción BGEZ no ocupa ER pero si ocupa el CDB al enviar la EA en caso de fallo
  En caso de una predicción T y fallo:

Instrucción	Issue	Exec	Wb	Commit
BGEZ R5, LOOP	1	2-3	4	5
ADDD F4, F0,F2	6			

- Las unidades funcionales tienen las siguientes características

UF	CANTIDAD	SEGMENTADA?	LATENCIA	ER/buffers
FP ADD	2	SI	3	4
FP MUL	2	SI	5	4
INT ALU	4	NO	1	4
LOAD/SD	1	SI	2	4LD + 4SD

Contesta a las siguientes preguntas

1. Suponiendo que existe una predicción inicial de salto no tomado para BGEZ R5, LOOP, rellena el diagrama temporal (en que ciclo se realiza cada fase) para la primera iteración del bucle + 3 instrucciones correctamente ejecutadas. Añade o tacha instrucciones en la tabla si es necesario

#ins	LABEL	Instrucción	Issue	Exec	Wb	Commit	COMENTARIO
i1		ADDI R5,R0,DIM	1	2	3	4	R5=DIM; DIM=4
i2	LOOP	ADDI R5,R5,-1	2	4(1)	5	6	R5=R5-1
i3		LD F4,0(R1)	3	4-5	6	7	F4=[M[R1]] ; A(i)
i4		LD F6,0(R2)	4	5-6	7	8	F6=[M[R2]]; B(i)
i5		LD F8,0(R3)	5	6-7	8	9	F8=[M[R3]]; C(i)
i6		LD F10,0(R4)	6	7-8	9	10	F10=[M[R4]]; D(i)
i7		MULTD F12, F4,F6	7	8-12	13	14	F12=[M[R1]]* [M[R2]]
i8		MULTD F14, F8,F10	8	10-14(2)	15	16	F12=[M[R3]]* [M[R4]]
i9		ADDD F14, F10,F14	9	16-18(3)	19	20	F14= A(i)* B(i)+ C(i)* D(i)
i10		ADDI R1, R1, 8	10	11	12	21	i++
i11		ADDI R2, R2, 8	11	12	14(4)	22	i++
i12		ADDI R3, R3, 8	12	13	16(4)	23	i++
i13		SD 0(R4), F14	13	14-15(5)	19*	24	D(i) = A(i)* B(i)+ C(i)* D(i)
i14		ADDI R4, R4, 8	14	15	17	25	i++
i15		BGEZ R5, LOOP	15	16-17	18(6)	26	SALTA A LOOP SI [R5]>=0
i16		ADDD F4, F0,F2	16	17-19	20	26	PREDICCIÓN NT errónea
i17		ADDD F6, F0,F2	17	18-20	21	26	Mal especulada
i18		ADDD F8, F0,F2	18(7)	19-21	22	26	Mal especulada
i19		~~ADDD F12, F0,F2~~					No se ejecuta
i20	LOOP	ADDI R5,R5,-1	27(8)	28	29	30	
i21		LD F4,0(R1)	28	29-30	31	32	
i22		LD F6,0(R2)	29	30-31	32	33	

(1) Si suponemos que las instrucciones de enteros ocupan el CDB, suponemos también que habrá un sistema similar de ER y Tomasulo, por lo que se produce una parada por dependencia de LDE entre i1 e i2
(2) Dependencia de LDE
(3) Dependencia de LDE
(4) CDB ocupado
(5) Calculo la EA una vez esté disponible el valor de R4, si no tendría que retrasar la siguiente instrucción, además 19* no ocupa el CDB, la instrucción que hace el WB efectivo es i9
(6) En el ciclo 18 detectamos que la predicción es errónea y paramos el lanzamiento de más instrucciones y cuando llega el COMMIT de la instrucción de salto (26) uso ese ciclo para borrar todo lo que haya en las ER y ROB, por eso pongo el COMMIT de todas al 26. En otro ejemplo hecho en clase, nos especificaba que necesitábamos más de un ciclo, pero el enunciado de éste es distinto. Espero a que llegue el turno para completar lo que estaba terminado y evitar un borrado selectivo en el 19, que sería mucho más complejo, aunque factible.
(7) Podríamos evitar el lanzamiento de esta instrucción también, pero requiere mayor complejidad
(8) Necesitaríamos al menos dos ciclos para hacer el IF y el ID de la instrucción, pero han pasado 9 ciclos desde el 18, por lo que supongo que puedo hacer el issue sin problemas. Además el ejemplo del enunciado no lo tiene en cuenta, o están incluidos en los ciclos de penalización.

2.Se quiere añadir a la arquitectura una unidad de memoria que disponga de un Buffer para instrucciones de Load y Store, que chequee las posibles coincidencias de direcciones efectivas para evitar LOAD innecesarios, haciendo que la instrucción de SD le haga un forwarding del valor. Por ejemplo si tenemos el siguiente conjunto de instrucciones

ADDD F6, F2, F4
SD F6, 8(R3)
…
LD F8, 8 (R3)
Sabiendo que el valor de R3 no ha cambiado entre *SD F6, 8(R3)* y *LD F8, 8 (R3)* y que la suma termina en el mismo ciclo en que se podría completar el acceso a memoria del *LD*, explica como podríamos evitar que el LOAD tenga que esperar a que se almacene en memoria el valor y luego volver a cargarla. Explica con un esquema la posible implementación

Hacemos el wb en el ciclo en que finaliza la operación ADDD F6, F2, F4 para evitar tener un store pendiente a la misma dirección y evitar ese acceso a memoria. Suponemos que 8(R3) es igual para ambas instrucciones. En ese caso:

- en el ciclo X escribo en el ROB la información de ADDD F6, F2, F4, asignándole la etiqueta ROBX
- la instrucción de SD, en el ciclo Y>X, asignándole la etiqueta ROBX
- En el ciclo Y+1, si no hay dependencias pendientes, calculo el valor de 8(R3) y lo almaceno en el ROB y en la unidad de memoria
- en el ciclo Z >Y>X escribo en el ROB la información de LD Y calculo 8(R3) si me encuentro que son iguales, entonces la unidad de memoria tomará directamente el valor que salga de la unidad de ejecución. La instrucción de suma (etiqueta ROBX) es la que proporciona el valor tanto al SD como al LD, por lo tanto en la unidad de memoria se cambiará la dirección 8(R3) por la etiqueta ROBX.
- Es necesario incorporar algún elemento de anotación, para las instrucciones de SD que estén pendientes de comitear, esto lo podemos hacer en los buffer de load y store, modificándolos, o en una estructura adicional, similar a un TLB.

3. El siguiente fragmento de código se ejecuta en un DLX con segmentación de 5 etapas. Indica los ciclos en que se ejecutan cada fase de las instrucciones y señala cuando se realiza forwarding.

```
LOOP:  LD      F2, 0(R1)
       MULTD   F4, F2, F0
       LD      F6, 0(R2)
       ADDD    F6, F4, F6
       SD      0(R2), F6
       ADDI    R1, R1, 8
       ADDI    R2, R2, 8
       SGTI    R3, R1, DONE
       BEQZ    R3, LOOP
```

Suponiendo que:

- un dato se puede escribir en un registro y leer su valor en el mismo ciclo
- se dispone de lógica de cortocircuito
- los saltos se resuelven en la etapa de decodificación
- se emplea un *branch delay slot* de una instrucción
- las unidades funcionales tienen las siguientes características:

UF	Cantidad	Latencia	Segmentación
FP ADD	1	4	No
FP MUL	1	5	No
Int ALU	1	1	No

CICLO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
LD F2, 0(R1)	F	D	E	M	W																
MULTD F4, F2, F0		P	F	D	E1	E2	E3	E4	E5	M	W										
LD F6, 0(R2)				F	D	E	M	W													
ADDD F6, F4, F6					P	P	P	F	D	E1	E2	E3	E4	M	W						
SD 0(R2), F6									P	P	P	F	D	E	M	W					
ADDI R1, R1, 8													F	D	E	M	W				
ADDI R2, R2, 8														F	D	E	M	W			
SGTI R3, R1, DONE															F	D	E	M	W		
BEQZ R3, LOOP																P	F	D	E	M	W