GUÍA PARA APROBAR EC - SEGMENTACIÓN

En estos problemas se partirá siempre de un procesador MIPs segmentado que puede tener algunas modificaciones que las dará el propio problema. Habrá también escrito un código en ARM con el que se trabajará a lo largo del ejercicio (aun así no hace falta saber 100% que hace cada instrucción, con una idea general normalmente basta).

En general, es muy común que caigan los siguientes apartados:

1. Dibujar diagrama tiempo-instrucción de las primeras X iteraciones
2. Hallar el CPI del código
3. Reordenar el código para eliminar ciclos de espera

Reservados todos los derechos.
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

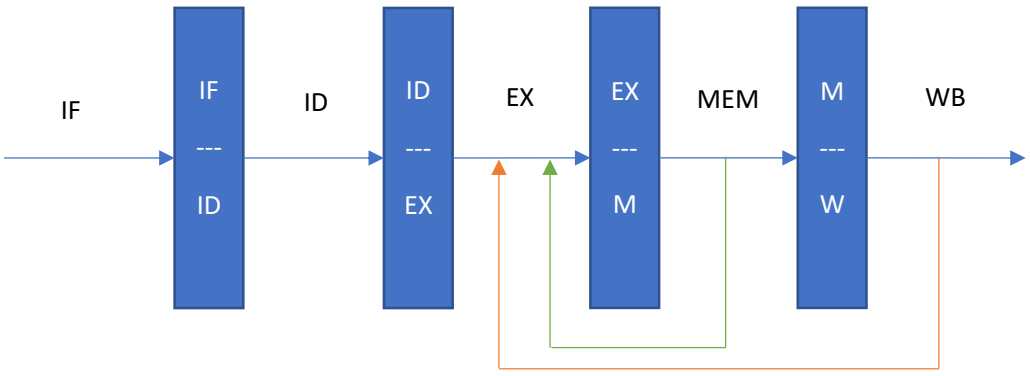
Dibujar diagrama tiempo-instrucción de las primeras X iteraciones

En estos problemas hay que dibujar una cuadrícula donde, para cada instrucción del código ARM dado, se debe dividir en cada una de sus 5 (o más, si el problema lo especifica) etapas. Esto normalmente se debe repetir 1 ó 2 veces si existe un bucle.

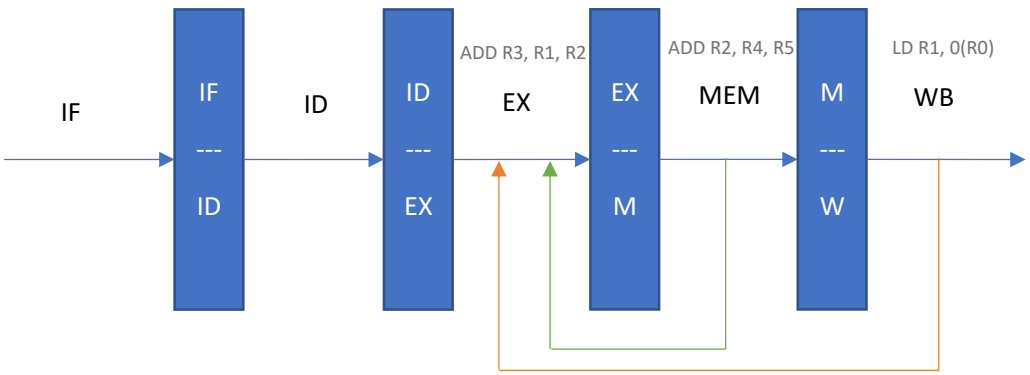
Cada instrucción pasa por las etapas:

- **Instruction Fecth (IF).** La instrucción está siendo leída de la memoria de instrucciones.
- **Instruction Decode (ID).** La instrucción está siendo decodificada (lectura de los bancos de registros).
- **Execution (EX).** La instrucción está siendo ejecutada (sumando, restando en la ALU…).
- **Memory (MEM).** La instrucción está escribiendo/leyendo de la memoria de datos.
- **Write Back (WB).** La instrucción está escribiendo los resultados en el banco de registros.

El dibujo que tenéis que tener en mente para sacar este apartado es el siguiente:



Por él irán pasando las instrucciones una a una como se muestra:



Reservados todos los derechos.
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

¿Qué significa cada cosa?

- Las cajas azules son **registros** donde se guardan los resultados calculados en las **etapas** (las flechas azules). Estos registros sirven como "puertas" entre dos etapas: cuando hay un flanco de reloj la "puerta" se abre y se pasa a la siguiente etapa. Mientras la "puerta" (recordemos, el registro) no se abra (no haya flanco de reloj) la etapa seguirá haciendo operaciones que pueden afectar a su resultado. Por ejemplo, hasta que no haya un flanco de reloj no se escribirá en el registro "EX/MEM" el resultado de calcular "ADD R3, R1, R2".

- A las flechas rojas y verdes se las llama **cortocircuitos**; dicho de otra manera, las flechas son atajos de datos. ¿Para qué sirve tener un atajo de datos? Pues se usará para que las instrucciones puedan anticipar operandos de instrucciones que no han terminado aún de ejecutarse. En el ejemplo del dibujo, si no hubiese cortocircuitos, la instrucción "ADD R3, R1, R2" debería esperar a que las instrucciones "ADD R2, R4, R5" y "LD R1, 0(R0)" terminasen de ejecutarse (llegaran a su etapa WB), ya que esta instrucción depende de las otras dos para conseguir sus operandos (R1 y R2). No obstante, con los cortocircuitos, podemos adelantar los resultados de las dos instrucciones anteriores a la etapa EX de "ADD R3, R1, R2" y con ello conseguir los operandos necesarios para ejecutar la instrucción (es decir, realizar la suma). Hay que tener en cuenta que el problema puede cambiar la forma en la que se realizan los cortocircuitos, así que hay que leer detenidamente.

Cosas a tener en cuenta:

1. **Las instrucciones no tienen por qué hacer algo en todas las etapas**. Por ejemplo, una instrucción ADD no escribirá nada en la memoria de datos en la etapa MEM (si eso escribe en el banco de registros en la etapa WB).
2. Los cortocircuitos se realizan siempre (a no ser que lo especifique el problema) **después de los registros** entre cada etapa (como veremos más abajo).
3. Las instrucciones pueden detenerse en una de sus etapas para **evitar RIESGOS** en la ejecución del procesador. Estos riesgos los veremos ahora en nada.
4. El cortocircuito de la etapa MEM (flecha verde) solo sirve **para adelantar operandos de instrucciones aritmético-lógicas** (ADD, SUB, MULD, ADDD…), NO para adelantar operandos de operaciones de lectura ("LD R1, 0(R0)", por ejemplo). Por otro lado, el cortocircuito de la etapa WB (la flecha naranja) sirve para adelantar operandos tanto de instrucciones aritmético-lógicas como operandos de instrucciones de lectura (LD). Por lo tanto, si se tiene una instrucción "LD R2, 0(R0)" en la etapa MEM, ese operando no se puede cortocircuitar (adelantar) a la etapa EX y habría que insertar un ciclo de espera (como veremos más adelante), mientras que si está en la etapa WB, SÍ que se puede adelantar a la etapa EX a través de la flecha roja.

Reservados todos los derechos.
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Usaremos las abreviaturas de antes para los diagramas instrucción-tiempo:

CICLOS	1	2	3	4	5	6	7	8	9
LD R1, 0(R0)	IF	ID	EX	MEM	WB				
LD R2, 4(R0)		IF	ID	EX	MEM	WB			
ADD R3, R1, R2			IF	IDp	ID	EX	MEM	WB	
ADD R4, R5, R6				IFp	IF	ID	EX	MEM	WB

¿Qué es lo COMPLICADO del ejercicio?

Hay algunas instrucciones que "dependen" entre ellas, y nuestro objetivo es detener uno o más ciclos aquellas instrucciones que se vayan a ejecutar de forma errónea. Dicho de otra manera, se detienen instrucciones que puedan causar RIESGOS.

Existen tres tipos de riesgos:

1. **Riesgo de datos**. Estos son los más comunes:

    a. *Lectura después de escritura (LDE)*. Es el más común de todos. Ocurre cuando se intenta leer un dato que no se ha terminado de escribir. En la tabla de arriba la instrucción "ADD R3, R1, R3" se detiene un ciclo en la etapa ID (IDp, la 'p' simboliza que esta "parado") para dar tiempo a la instrucción "LD R2, 4(R0)" a llegar a la etapa WB para aprovechar su cortocircuito hacia la etapa "EX" y así usar su operando R2. Con la instrucción "LD R2, O(R0)" no hay riesgo de datos ya que a esa instrucción le ha dado tiempo de sobra a terminar de ejecutarse.

    b. *Escritura después de lectura (EDL)*. Este es raro y solo puede ocurrir con Unidades Funcionales. Sucede cuando se va a sobreescribir un dato que no se ha terminado de leer en una instrucción anterior.

    ADD R1, R4, R3
    ADD R4, R5, R2

    Si el segundo ADD escribe antes en R4 antes que el primer ADD, se produce este riesgo.

    c. *Escritura después de escritura (EDE)*. Otro que solo puede ocurrir con Unidades Funcionales. Sucede cuando se tienen dos instrucciones de escritura y se escribe en orden inverso.

    ADD R3, R1, R2
    ADD R3, R3, R3

    Si el primer ADD termina de escribir en R3 después del segundo, se produce este riesgo.

2. **Riesgo estructural**. Sucede cuando dos instrucciones intentan, en el mismo ciclo de reloj, acceder a una misma etapa. En la tabla de arriba la instrucción "ADD R4, R5, R6" se detiene un ciclo en su etapa IF (IFp) para evitar que en el siguiente ciclo coincida con la etapa ID de la anterior instrucción.

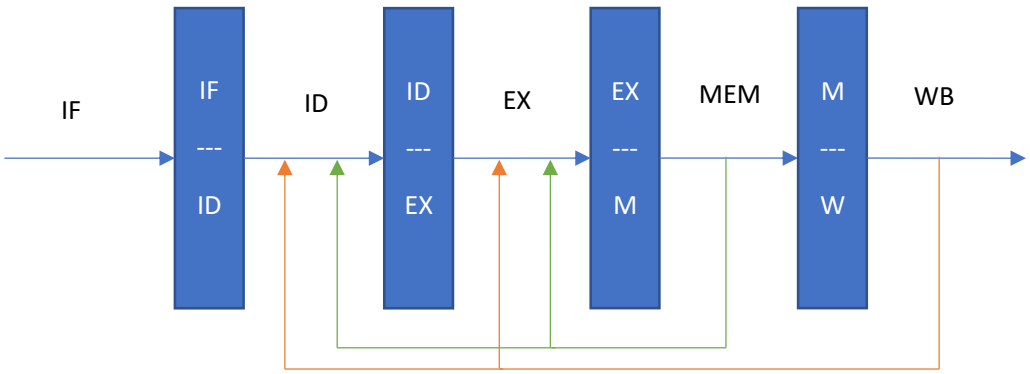
Reservados todos los derechos.
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

3. **Riesgo de control**. Solo ocurre con los saltos condicionales. Estos se tratan de una manera un poco más especial y se suele especificar en el mismo problema como se deben resolver estos riesgos. Hay dos formas posibles:

   a. **Saltos retardados**. Es una forma sutil de decir *"meter suficiente relleno entre dos instrucciones hasta que haya tanta mierda que desaparezca por si solo el riesgo"*. El problema normalmente pone instrucciones por el código ARM que no dependen de otras metas donde las metas, ya sea porque escribe en registros que nunca se usan o porque su orden de ejecución puede cambiar de alguna forma, por lo que tu trabajo es cogerlas y ver donde caben. También se pueden meter instrucciones que no alteren el código. No obstante esto solo debe realizarse si no encuentras ninguna otra salida.

   b. **Cancelación de la siguiente instrucción**. Si el salto se toma la siguiente instrucción que se ha metido es basura, así que lo que se hace es rellenar con 0's sus registros. La decisión del salto se toma en la etapa ID, es decir, en la etapa EX ya se debe haber rellenado con 0's o no. Para poder tomar esta decisión se pueden usar cortocircuitos a la etapa ID



Hay que recordar que la flecha verde solo adelanta operandos de instrucciones aritmético-lógicas (ADD, SUB…) mientras que la naranja adelanta operandos de instrucciones de lectura (LD).

Los guiones en la tabla de abajo representa "rellenar con 0's".

CICLOS	1	2	3	4	5	6	7
BEQ R1, R1, LOOP	IF	ID	EX	MEM	WB		
LD R2, 4(R0)		IF	-	-	-	-	
LOOP: ADD R3, R1, R2			IF	ID	EX	MEM	WB

Reservados todos los derechos.
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

¿Cuántos ciclos hay que detener una etapa?

Pues la respuesta es sencilla: los suficientes hasta que se elimine cualquier riesgo.

¿Qué etapas pueden ser detenidas si se va a producir un riesgo de datos?

Lo dice el problema, pero normalmente se detiene la etapa ID para evitar que entren más instrucciones. No obstante, hay que recordar que cuando se detiene una etapa en el 99% de los casos hay que detener también la de la siguiente instrucción para evitar un riesgo estructural.

Ahora solo resta una cosa más por ver para saber todo lo que necesitas para resolver este apartado: las **unidades funcionales**.

Una unidad funcional es un tipo de módulo que usa el procesador para realizar operaciones, normalmente en varios ciclos. En verdad, no necesitas saber cómo funciona una unidad funcional en sí, no al menos para los problemas que se han presentado en los exámenes. En el problema siempre te darán una tabla con el siguiente aspecto:

UF	Cantidad	Latencia	Segmentación
FP ADDD	1	3	SÍ
FP MULD	1	5	SÍ
…	…	…	…
INT ALU	1	1	NO

Significado de cada columna:

- **UF**: unidad funcional
- **Cantidad**: el número de unidades funcionales de ese tipo que tiene el procesador.
- **Latencia**: el número de ciclos que tarda esta unidad funcional en completar la etapa EX.
- **Segmentación**: una unidad funcional segmentada puede tolerar distintas instrucciones operando sobre la misma unidad funcional mientras estén en distintas etapas de ejecución.

Posibles unidades funcionales: ADDD (sumar), MULD (multiplicar), DIVD (dividir), SUBD (restar)…

Esto se ve mejor con ejemplos:

Reservados todos los derechos.
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Ejemplo 1

(situación normal)

UF	Cantidad	Latencia	Segmentación
FP ADDD	1	3	SÍ
FP MULD	1	5	SÍ
INT ALU	1	1	NO

ADDD R1, R2, R3	IF	ID	A1	A2	A3	MEM	WB				
MULD R4, R5, R6		IF	ID	M1	M2	M3	M4	M5	MEM	WB	
MULD R7, R8, R9			IF	ID	M1	M2	M3	M4	M5	MEM	WB

Ejemplo 2

(una unidad funcional no segmentada se pelea por usarla con otra instrucción con música de linkin park de fondo)

UF	Cantidad	Latencia	Segmentación
FP ADDD	1	3	NO
FP MULD	1	5	NO
INT ALU	1	1	NO

ADDD R1, R2, R3	IF	ID	A1	A2	A3	MEM	WB			
MULD R4, R5, R6		IF	ID	M1	M2	M3	M4	M5	MEM	WB
MULD R7, R8, R9			IF	IDp	IDp	IDp	IDp	ID	M1	M2...

El segundo MULD tiene que esperar a que el primero acabe su ejecución para poder usar la unidad funcional (los puntos suspensivos del último M2 es porque no cabe el resto, pero seguiría M3, M4, M5, MEM Y WB).

Ejemplo 3

(dos unidades funcionales no segmentadas funcionan a la vez gracias a que hay más de una)

UF	Cantidad	Latencia	Segmentación
FP ADDD	1	3	NO
FP MULD	2	5	NO
INT ALU	1	1	NO

ADDD R1, R2, R3	IF	ID	A1	A2	A3	MEM	WB				
MULD R4, R5, R6		IF	ID	M1	M2	M3	M4	M5	MEM	WB	
MULD R7, R8, R9			IF	ID	M1	M2	M3	M4	M5	MEM	WB

Aquí, como hay dos MULD, dos instrucciones pueden hacer uso de ella, pese a no estar segmentada.

Reservados todos los derechos.
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Hallar el CPI de todo el código

Bueno, una vez superado el primer apartado, este ya es más fácil. Solo hay que coger el número total de ciclos que ha llevado el código ejecutarse y dividirlo entre el número total de instrucciones. No se por qué me enrollo, toma tu fórmula:

$$CPI = \frac{nº\ de\ ciclos}{nº\ de\ instrucciones}$$

Reordenar el código

Este apartado ya se ha medio explicado en los *riesgos de control* en el primer punto. Lo que te piden es que reordenes el código de tal forma que desaparezcan la mayor cantidad de riesgos (de cualquier tipo), de tal forma que consigas que no haya ninguna etapa parada (con la 'p' al final). Esto se consigue cambiando el orden de instrucciones **PERO SIN ALTERAR EL SIGNIFICADO DEL CÓDIGO**. Lo último es importante, ya que sino sería lo mismo que hacer una sopa de letras con todas las instrucciones.

No es lo mismo:

ADD R1, R2, R3

ADD R4, R1, R2

Que:

ADD R4, R1, R2

ADD R1, R2, R3

Dicho esto, normalmente el cómo se consigue hacer esto es localizando las instrucciones que no dependen de ninguna otra e insertándolas entre medias de otras instrucciones que sí dependan entre ellas.

Ejemplo

CICLOS	1	2	3	4	5	6	7	8	9	10
ADD R6, R6, R6	IF	ID	EX	MEM	WB					
ADDD R1, R2, R3		IF	ID	A1	A2	A3	A4	MEM	WB	
MULD R4, R1, R5			IF	IDp	IDp	IDp	ID	M1	M2	M3…
ADD R7, R7, R7				IFp	IFp	IFp	IF	ID	EX	MEM…

Aquí es bastante obvio qué instrucciones se deben mover para eliminar esos ciclos de espera:

Reservados todos los derechos.
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

CICLOS	1	2	3	4	5	6	7	8	9	10
ADDD R1, R2, R3	IF	ID	A1	A2	A3	A4	MEM	WB		
ADD R6, R6, R6		IF	ID	EX	MEM	WB				
ADD R7, R7, R7			IF	ID	EX	MEM	WB			
MULD R4, R1, R5				IF	IDp	ID	M1	M2	M3	M4…

Con esta reordenación, se ha llegado a la etapa M3 en 9 ciclos en vez de 10, ahorrándonos uno de espera.

Reservados todos los derechos.
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH