Resumen Arqui

Primer Examen

Definición de la arquitectura de una computadora
Son los atributos de la computadora, dependiendo de lo que se necesita. Para crear una computadora:

1) Preguntarse para qué se ocupa, cuánto presupuesto tengo, tendencias actuales.
2) Arquitectura: **RISC** (conjunto de instrucciones reducido) y **CISC** (conjunto de instrucciones complejo). El conjunto de instrucciones se refiere a lo que la PC (el hardware) comprende. En RISC las instruccione son menos, más simples y eficientes, caso contrario en CISC donde las instrucciones son más complejas.
   a) **ISA** (Arquitectura del conjunto de instrucciones):
      - Operaciones aritméticas
      - Tipo de operandos
      - Almacenamiento de operandos (acumulador, pila...)
      - Tamaño de operandos
      - Modos de direccionamiento a memoria (por byte, alineada...)
      - Operaciones de control

   b) Lógica para ejecutar las instrucciones
   c) Unidades funcionales y el camino que sigue una instrucción al ejecutarse utilizando esas unidades funcionales. Aquí se trabaja con pipeline (en paralelo).
   d) Implementación (microarquitectura): MIPS utiliza arquitectura RISC.

Introducción a MIPS
MIPS es una arquitectura diseñada para optimizar la segmentación en unidades de control y para facilitar la generación automática de código máquina por parte de los compiladores.

Arquitecturas MIPS ISA (Instruction Set Architecture)

- Codificación de las instrucciones: Definir si la instrucción es fija o variable.
- 32 registros para enteros (R0, R1, R2, ..., R31) y para flotantes (F0, F1, F2, ..., F31).
- Los formatos de codificación de las instrucciones son de 3 tipos:

- **Tipo I**: Instrucciones Aritméticas (operaciones con inmediatos)

0 - 5	6 - 10	11 - 15	16 - 31
código de operación	registro fuente	registro destino/fuente	inmediato

- **Tipo R**: Instrucciones con Referencia a memoria (aritmético-lógicas)

0 - 5	6 - 10	11 - 15	16 - 20	21 - 25	26 - 31
código de operación	registro fuente 1	registro fuente 2	registro destino	despl	función (operación aritmética a

					realizar)

- **Tipo J**: Instrucciones de Salto (incondicional)

0 - 5	6 - 31
código de operación	dirección destino del salto (desplazamiento)

Pipeline en MIPS

Qué es pipeline? → es la ejecución en paralelo de instrucciones. En MIPS un ciclo de ejecución está compuesto por 5 etapas. Ciclo de ejecución de una instrucción en MIPS:

- **IF** (Ifetch): Lee una instrucción desde la memoria de instrucciones (búsqueda). Actualiza la PC a la siguiente PC secuencial agregando 4 (ya que cada instrucción tiene 4 bytes) a la PC.
  IR ← MemoryInstruction[PC];
  NewPC ← PC + 4;

- **ID** (Reg/dec): Decodifica la instrucción y lee registros paralelamente.
  A ← Regs[rs];
  B ← Regs[rt];
  Imm ← sign-extended immediate field of IR;

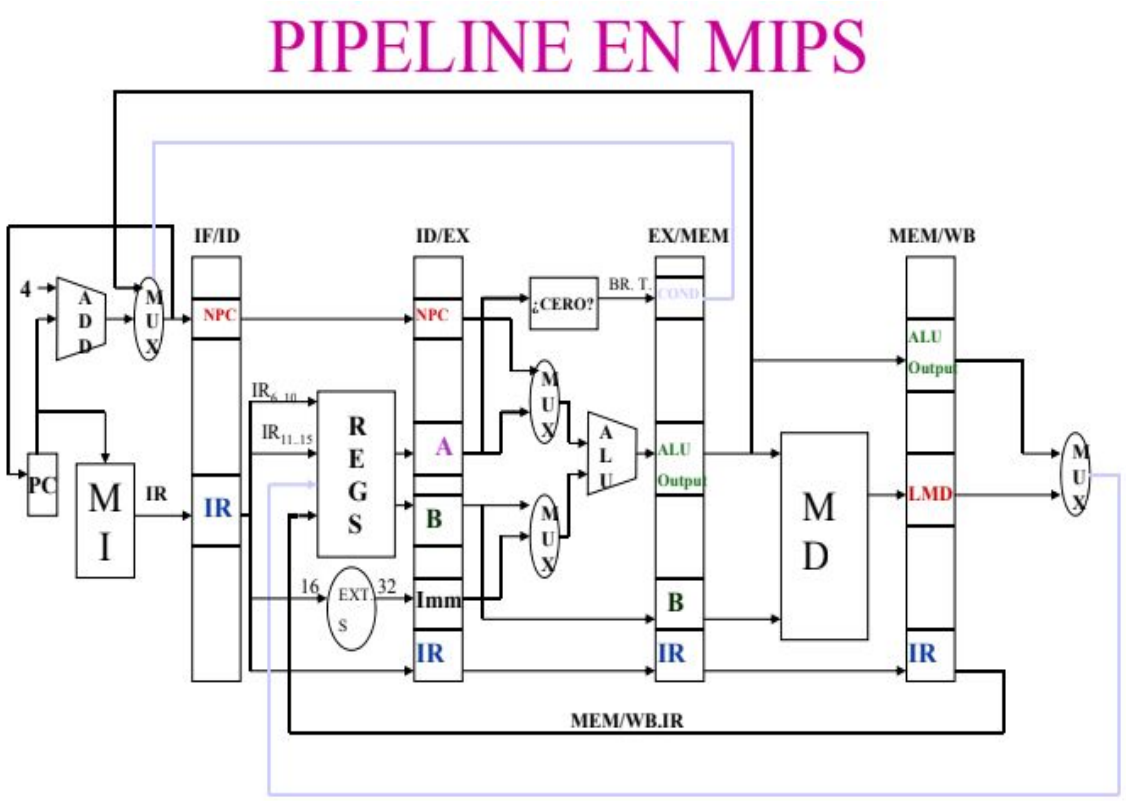
- **EX** (Ejecuta): La ALU opera en los operandos preparados en el ciclo anterior, realizando una de 3 funciones dependiendo del tipo de instrucción:
  Referencia de memoria ALU: la ALU agrega el registro base y el desplazamiento para formar la dirección efectiva.
  ALUOutput ← A + Imm;
  Registro-registro ALU: La ALU realiza la operación especificada por el código de operación ALU en los valores leídos del archivo de registro.
  ALUOutput ← A func B; donde func puede ser: +, -, *, /
  Registro-inmediato ALU: La ALU realiza la operación especificada por el código de operación ALU en el primer valor leído del archivo de registro y el inmediato.
  ALUOutput ← A op Imm; donde op puede ser: +, -, *, /
  Branch:
  ALUOutput ← NewPC + (Imm << 2);
  Cond ← (A == 0)

  En una arquitectura **Load-Store** la dirección efectiva y los ciclos de ejecución se pueden combinar en un solo ciclo de reloj, ya que ninguna instrucción necesita calcular simultáneamente una dirección de datos y realizar una operación en los datos.

- **MEM** (Memoria): Lee el dato de la memoria de datos. Si la instrucción es un LOAD la memoria hace una lectura usando la dirección efectiva calculada en el ciclo anterior.

Si se trata de un STORE la memoria escribe los datos del segundo registro leídos del archivo de registro usando la dirección efectiva.
LMD ← MemoryData[ALUOutput] or MemoryData[ALUOutput] ← B;

- **WB** (Write Back): Escribe el dato en el registro, ya sea que provenga del sistema de memoria (para un LOAD) o de la ALU (para una instrucción ALU).
  Register-register ALU: Regs[rd] ← ALUOutput;
  Register-immediate ALU: Regs[rt] ← ALUOutput;
  Load: Regs[rt] ← LMD



Fundamentos de un conjunto de instrucciones RISC: El conjunto de instrucciones MIPS proporciona 32 registros, aunque el registro 0 siempre tiene el valor 0. La mayoría de las arquitecturas RISC, como MIPS, tienen tres clases de instrucciones:

1) **ALU**: Estas instrucciones toman 2 registros o un registro y un inmediato, operan en ellos y almacenan el resultado en un tercer registro. Las operaciones típicas: add (DADD), resta (DSUB) y operaciones lógicas (como AND u OR).
2) **Load and store**: Estas instrucciones toman una fuente de registro, llamada registro base, y un campo inmediato (desplazamiento) como operandos. La suma (dirección efectiva) de los contenidos del registro base y el desplazamiento se utiliza como una dirección de memoria. En el caso de un LOAD un segundo operando de registro actúa como el destino para los datos cargados desde la memoria. En el caso de una STORE el segundo operando de registro es la fuente de los datos que se almacenan en la memoria.

3) **Branches and jumps**: Las ramas son transferencias condicionales de control. Para especificar la condición de branch MIPS usa la técnica de comparar entre y registro y un 0.

Conceptos prácticos:
IC = número de instrucciones que hacen Write Back.
CPI = ciclos por instrucción (retraso que tiene una instrucción con respecto a Write Back).
TE = tiempo de ejecución: IC * CPI * LCR

Jerarquías de memoria
Registros de CPU: interaccionan continuamente con la CPU (porque forman parte de ella). Tienen un tiempo de acceso pequeño (rápidas) pero es más cara.
Memorias caché: Son memorias de pequeña capacidad y pequeño tiempo de acceso. Este nivel de memoria se coloca entre la CPU y la memoria central. La memoria caché se divide en: datos e instrucciones. Los niveles de caché son: L1, L2, L3. La CPU siempre se comunica primero con la L1.
Memoria principal: En este nivel residen los programas y los datos. La CPU lee y escribe datos en él aunque con menos frecuencia que en los niveles anteriores. Tiene un tiempo de acceso más lento y gran capacidad.

Cuatro preguntas para cachés en una **jerarquía de memoria:**

1. Asignación de bloque?
- **Mapeo directo:** en la correspondencia directa el bloque Bj de la memoria principal se puede ubicar sólo en el marco de bloque MBi que cumple la siguiente relación i = j mod m, donde m es el número total de líneas que tiene la caché.
- **Completamente asociativa:** en la correspondencia asociativa un bloque Bj de la memoria principal se puede ubicar en cualquier marco de bloque de la caché.
- **Asociativa por conjuntos**: las líneas de la memoria principal se agrupan en $v=2^d$ conjuntos con k líneas. Se cumple que el número total de marcos de bloque que tiene la caché m = v*k. Un bloque Bj de Mp se puede ubicar sólo en el conjunto Ci de Mc que cumple la siguiente relación i = j mod v.

2. Búsqueda en caché de un bloque?
Depende de la estrategia usada en la asignación de bloques

3. Estrategias de reemplazo de bloques?
- **LRU**: se sustituye el bloque que hace más tiempo que no ha sido referenciado.
- **FIFO**: se sustituye aquel bloque que ha estado más tiempo en la caché.
- **Random**: se escoge un bloque al azar, solo si es completamente asociativa o asociativa por conjuntos.

Reducción de la tasa de fallos:
- **Aumento del tamaño del bloque**: al aumentar el tamaño de bloque disminuye la tasa de fallos iniciales. Sin embargo con el aumento del tamaño de bloque aumenta

la penalización de fallos, ya que el tiempo de lectura y transmisión serán mayores si los bloques son mayores.

- **Aumento de la asociatividad**: una caché asociativa por conjuntos de 8 vías es tan eficiente como una caché completamente asociativa. Una caché de correspondencia directa de tamaño N tiene aproximadamente la misma tasa de fallos que una asociativa por conjuntos de 2 vías de tamaño N/2. Al aumentar la asociatividad se incrementa el ciclo de reloj y por tanto la tasa de acierto.
- **Utilización de una caché de víctimas**: Se añade una pequeña caché completamente asociativa entre caché y memoria principal para contener sólo los bloques sustituidos por un fallo (víctimas). Ante un fallo se comprueba si el bloque está en la caché de víctima antes de acudir a la memoria principal.

4. Estrategia de escritura?

Determina la forma de **actualizar la memoria principal cuando se realizan operaciones de escritura.** Existen dos casos para la escritura: cuando la posición de memoria sobre la que se va a escribir está en memoria (hit) y cuando no (miss).

- **Acierto (Hit)**: Existen dos alternativas: escritura directa o inmediata (write through) o postescritura (write back).
    - **Write through**: Todas las operaciones de escritura se realizan en memoria principal y caché.
    - **Write back**: Las actualizaciones se hacen solo en la caché. Se utiliza un bit de actualización asociado a cada bloque de la caché para indicar la escritura del bloque en memoria principal cuando es sustituido por la política de reemplazamiento.
- **Fallo (Miss)**: Existen dos alternativas: asignación en escritura (write allocate) y no asignación (no write allocate).
    - **Write allocate**: El bloque se ubica en la caché cuando ocurre el fallo de escritura y a continuación se opera como en un acierto de escritura.
    - **No write allocate**: El bloque se modifica en memoria principal sin cargarse en la caché.

Coherencia de caché

Hace referencia a la integridad de los datos almacenados en las cachés locales de los recursos compartidos.

Protocolos para mantener la coherencia:

- **Protocolo Snoopy**: Cada controlador de caché está observando los eventos que suceden en el bus y toma decisiones en consecuencia. Se reserva un campo en el bloque de la caché para mostrar si el bloque fue compartido, modificado o invalidado.
- **Protocolo basado en directorios**: Se utiliza un directorio (centralizado o distribuido) con una entrada por cada bloque en la que se indica en qué caches existe copia y en qué estado.

Examen Final

Paralelismo en el nivel de instrucciones (ILP)

Dependencia y peligro de los datos: Determinar cómo una instrucción depende de otra es fundamental para determinar cuánto paralelismo existe en un programa y cómo se puede aprovechar ese paralelismo. Si 2 instrucciones son dependientes, no son paralelas y deben ejecutarse en orden, aunque a menudo pueden estar parcialmente superpuestas. Hay 3 tipos de dependencia: de **datos** (Instrucción i produce un resultado que puede ser usado por la instrucción j), de **nombre** (Se produce una dependencia de nombre cuando dos instrucciones usan el mismo registro o ubicación de memoria, llamado nombre, pero no hay flujo de datos entre las instrucciones asociadas con ese nombre) y de **control** (determina el orden de una instrucción i con respecto a una instrucción **branch** para que la instrucción i se ejecute en el orden correcto de programa).

Resolución para conflictos de datos

Para esto se utiliza la técnica conocida como **forwarding**.

Forwarding en el pipeline MIPS para flotantes, siempre a la primera etapa de la operación aritmética. Si se trata de un Store, espera en EX, no en M si el dato a enviar no está listo.

Reducción de los retrasos por culpa de conflictos de control (o sea por branches)

- **Esquema 2, predicción de branch no tomado** pág C-22. Lo más importante acá es que se dejan ingresar al pipeline las instrucciones que siguen al branch, si el branch es no tomado, se dejan continuar, y si es tomado, se evita que continúen en el pipeline y se trae al pipeline la instrucción del "salto" - esa predicción se puede hacer estáticamente -es decir, de fijo, siempre hacer la predicción de no tomado- o se puede hacer dinámicamente -basado en "estadísticas" se predice mientras se ejecuta el programa-.
- **Esquema 3, predicción de branch tomado** pág C-23. Esto implica traer al pipeline las instrucciones del salto (de la etiqueta en adelante) Sin embargo esto tiene sentido cuando se hace de manera dinámica, pues ya se ha ejecutado ese branch, se han hecho estadísticas y se ha calculado la dirección en memoria de la instrucción de la etiqueta.
- **Esquema 4, Branch retrasado (delayed branch)** de pág C-23 a C-26. Este esquema consiste en incluir por parte del compilador un número de instrucciones buenas (sin conflicto de datos ni de control, que no dependen del branch, ni afectan su ejecución) que se toman de las instrucciones del programa anteriores al branch, y que se deben ejecutar independientemente del resultado del branch. Estas instrucciones entran al pipeline inmediatamente luego del branch y el procesador permite su ejecución completa. El número de instrucciones que se incluyen por parte del compilador, y que el procesador espera, es como máximo, igual al número de ciclos de retraso que provoca el branch. Si se decide que el compilador incluya menos instrucciones que este número, entonces las instrucciones siguientes a estas, serán tomadas como predicción de branch no tomado. Para nuestro pipeline MIPS de 5 etapas para enteros, en donde el branch se resuelve en EX y Mem y hasta en

WB está listo el PC, el retraso por culpa del branch es de 3 ciclos (cpi de sig. instrucción, si no hay predicción es de 4) es por ello que el número de instrucciones máximo que se pueden utilizar por branch retrasado son 3. Así entonces, **si tanto procesador y compilador trabajan con menos de 3 instrucciones por branch retrasado, las o la siguiente se toman para hacer predicción no tomado para el branch.**

- **Predicción estática y predicción dinámica de branches** de pág C26 a la C-30. Se habla de predicción estática y predicción dinámica de branches, pero en nuestro caso, interesa básicamente lo que les acabo de explicar al respecto arriba.

El "pipeline" del MIPS para operaciones multiciclo.

No es práctico exigir que todas las operaciones MIPS de **coma flotante** se completen en 1 ciclo de reloj. Hacerlo significaría aceptar un reloj lento o usar enormes cantidades de lógica en las unidades de coma flotante, o ambas. En cambio, el pipeline de coma flotante permitirá una latencia más larga para las operaciones. Esto incluye dos cambios:

- Primero, el ciclo EX puede repetirse tantas veces como sea necesario para completar la operación; el número de repeticiones puede variar para diferentes operaciones.
- Segundo, puede haber múltiples unidades funcionales de coma flotante.

En cuanto a las unidades de coma flotante:

1. La unidad entera principal que maneja **load y store**, operaciones enteras de ALU y branches.
2. Coma flotante y **multiplicador** de enteros.
3. Coma flotante que maneja **suma, resta** y conversión de coma flotante.
4. Coma flotante y **divisor** de enteros.

MUL.D	IF	ID	*M1*	M2	M3	M4	M5	M6	**M7**	MEM	WB
ADD.D		IF	ID	*A1*	A2	A3	**A4**	**MEM**	WB		
L.D			IF	ID	*EX*	**MEM**	WB				
S.D				IF	ID	*EX*	*MEM*	WB			

Las etapas en *cursiva* muestran dónde se necesitan los datos, mientras que las etapas en **negrita** muestran dónde está disponible un resultado.

Núcleos Multihilo: Uso del multihilo para aumentar el rendimiento uniprocesador mediante el uso de varios hilos

Multihilo permite que varios subprocesos compartan las unidades funcionales de un único procesador de forma superpuesta. Por el contrario, un método más general para explotar el paralelismo de nivel de subprocesos (TLP) es con un multiprocesador que tiene múltiples hilos independientes operando a la vez y en paralelo. Multihilo, sin embargo, no duplica todo el procesador como lo hace un multiprocesador, si no que comparte la mayor parte del núcleo del procesador entre un conjunto de hilos, duplicando solo el estado privado, como los registros y el contador del programa. Además, el hardware debe admitir la capacidad de cambiar a un hilo diferente de manera rápida.

Hay tres enfoques principales de hardware para multihilo.

- **Multihilo de grano fino:** cambia entre subprocesos en cada reloj, lo que provoca la intercalación de la ejecución de instrucciones de varios subprocesos.
  Ventaja: puede ocultar las pérdidas de rendimiento que surgen ya que las instrucciones de otros hilos se pueden ejecutar cuando se detiene un hilo.
  Desventaja: ralentiza la ejecución de un hilo individual, ya que un hilo que está listo para ejecutarse sin paradas se retrasará por instrucciones de otros hilos.
- **Multihilo de grano grueso:** se inventó como una alternativa al grano fino. Los switches multiproceso de grano grueso solo se enrutan en puestos costosos, como fallas de caché de nivel dos o tres.
  Ventaja: aquí las instrucciones de otros hilos sólo se emitirán cuando un hilo se encuentre con una pérdida costosa.
  Desventaja: tiene una capacidad limitada para superar las pérdidas de rendimiento.
- **Multihilo simultáneo:** es una variación en el multihilo de grano fino que surge de forma natural cuando se implementa un multihilo de grano fino sobre un procesador programado dinámicamente. La idea clave en SMT es que el cambio de nombre de registro y la programación dinámica permiten que se ejecuten múltiples instrucciones de subprocesos independientes sin tener en cuenta las dependencias entre ellos; la resolución de las dependencias puede ser manejada por la capacidad de programación dinámica.

Paralelismo en el nivel de hilos (TLP)

Multiprocesadores -> se define como computadoras que consisten en procesadores estrechamente acoplados cuya coordinación y uso suelen estar controlados por un único sistema operativo y que comparten memoria a través de un espacio de direcciones compartido.

Los multiprocesadores de memoria compartida existentes se dividen en dos clases, según la cantidad de procesadores involucrados, lo que a su vez determina una organización de memoria y una estrategia de interconexión.

- **Multiprocesadores de memoria compartida centralizada**: los procesadores comparten una única memoria centralizada a la que todos tienen acceso. También se denominan **UMA** (Acceso de Memoria Uniforme), osea que todos los procesadores tienen una latencia uniforme de la memoria.
  - Este tipo de memoria compartida utiliza el **protocolo Snooping** para coherencia de caché, por lo que invalida las copias del bloque en las otras cachés (si lo tienen) para escritura (store), esto para garantizar que no hay otras copias legibles cuando se produce la escritura.
- **Multiprocesadores de memoria compartida distribuida**: los procesadores tienen una memoria distribuida físicamente. Para admitir recuentos de procesador más grandes, la memoria debe distribuirse entre los procesadores en lugar de centralizarse; de lo contrario, el sistema de memoria no podría soportar las demandas de ancho de banda de un mayor número de procesadores sin incurrir en una latencia de acceso excesivamente larga. También se denomina **DUMA** (Acceso a Memoria No Uniforme) ya que el tiempo de acceso depende de la ubicación de una palabra de datos en la memoria.
  - Este tipo de memoria compartida utiliza el **protocolo basado en directorios.**

Una DESVENTAJA es que la comunicación de datos entre los procesadores se vuelve algo más compleja, y un DSM requiere más esfuerzo en el software para aprovechar el ancho de banda de memoria aumentado que proporcionan las memorias distribuidas.

Load Link y Store Conditional

La implementación de una única operación de memoria atómica presenta algunos desafíos, ya que requiere una **lectura de memoria** y una **escritura** en una sola instrucción ininterrumpida.

Este requisito complica la implementación de la coherencia, ya que el hardware no puede permitir ninguna otra operación entre la lectura y la escritura, y aún así no debe interbloquear. Una alternativa es tener un **par de instrucciones** donde la segunda instrucción devuelve un valor del cual se puede deducir si el par de instrucciones se ejecutó como si las instrucciones fueran atómicas. El par de instrucciones es efectivamente atómico si parece que todas las demás operaciones ejecutadas por cualquier procesador ocurrieron antes o después del par. Por lo tanto, cuando un par de instrucciones es efectivamente atómico, ningún otro procesador puede cambiar el valor entre el par de instrucciones.

Par de instrucciones:

LL (load linked):

lee el “lock”

SC (store conditional):

IF hubo una interrupción luego de la lectura anterior

devuelve un 0 como valor de retorno.

(FALLAMOS...LA OPERACIÓN NO FUE ATÓMICA)

ELSE

escribe un 1 en la posición de mem del lock y devuelve un 1.

(LO LOGRAMOS...LA OPERACIÓN FUE ATÓMICA)

Et1	DADDI R1, RO, #1	R1 = 1
Et2	LL R2, 84(Ro)	R2 = M[84] Candado RL = 84
	SC R1, 84(Ro)	1ero Averigua si puede hacer la op atómica comparando el valor del RL de su procesador con la dir de memoria en donde debe escribir: ¿RL = 84? **Sí:** -hace el Store de R1 (que vale 1) en M[84] (al invalidar bloques en otras cachés debe revisar si el RL del hilillo que corre en ese procesador vale 84, en cuyo caso debe cambiarlo a -1) **No:** -no escribe (no hace el store de R1) -En R1 guarda un 0 Al final, haya obtenido un SÍ, o un NO a la preg. anterior, en el registro RL en donde se está ejecutando ese SC se pone un -1
	BEQZ R1, Et1	Si SC falló, o sea R1 = 0 vaya a intentar hacer la operación atómica de nuevo
	BNEZ R2, Et2	Si R2 no es 0, o sea el candado estaba ocupado, intenta de nuevo (se mantiene en espera activa). Pero si ya es un 0, entonces continúa con la siguiente instrucción, es decir, entra a sección crítica.

Al implementar en un simulador las instrucciones LL y SC, se debe además:

- Modificar su SW, para que al igual que el LL, cuando va a invalidar un bloque en una caché revise si RL = dir de memoria en donde hará su escritura, en cuyo caso, debe cambiar el RL de ese procesador y poner un -1.
- Cada vez que se hace cambio de contexto, en el RL del procesador debe ponerse el valor -1.