11001

1–4

Controla la computadora del módulo lunar

+11

En 1969, millones de personas seguían los acontecimientos que sucedían a 384.000 km de la Tierra pegados a sus televisores. Tres minutos antes de que el Eagle se posara en la Luna, su computadora de a bordo lanzó varias alarmas. Por un fallo en los manuales, un radar no necesario para el aterrizaje estaba encendido cuando debería estar apagado. Enviaba señales erróneas a la computadora de a bordo, que tenía que lidiar con ellas además de encargarse de todas las operaciones propias del alunizaje. Afortunadamente, el software que gestionaba la computadora de a bordo fue lo suficientemente inteligente como para detectar el problema y la computadora avisó a los astronautas diciendo: "estoy sobrecargada con más tareas de las que debería estar haciendo en este momento, así que me voy a centrar solamente en las tareas importantes, es decir, las que tienen que ver con aterrizar". Recordemos que era una computadora con menos potencia de cálculo que el reloj digital de nuestra cocina. Sin este novedoso e inteligente diseño, el pequeño paso para el hombre y gran salto para la Humanidad habría acabado en siniestro total.

Hicieron falta más de 30 años antes de que la NASA reconociera el mérito de Margaret Hamilton, la directora del centro de ingeniería del software del MIT encargada de desarrollar el software de abordo de todo el programa Apollo, el único que ha sido capaz de permitirnos pisar otros mundos y volver para contarlo.

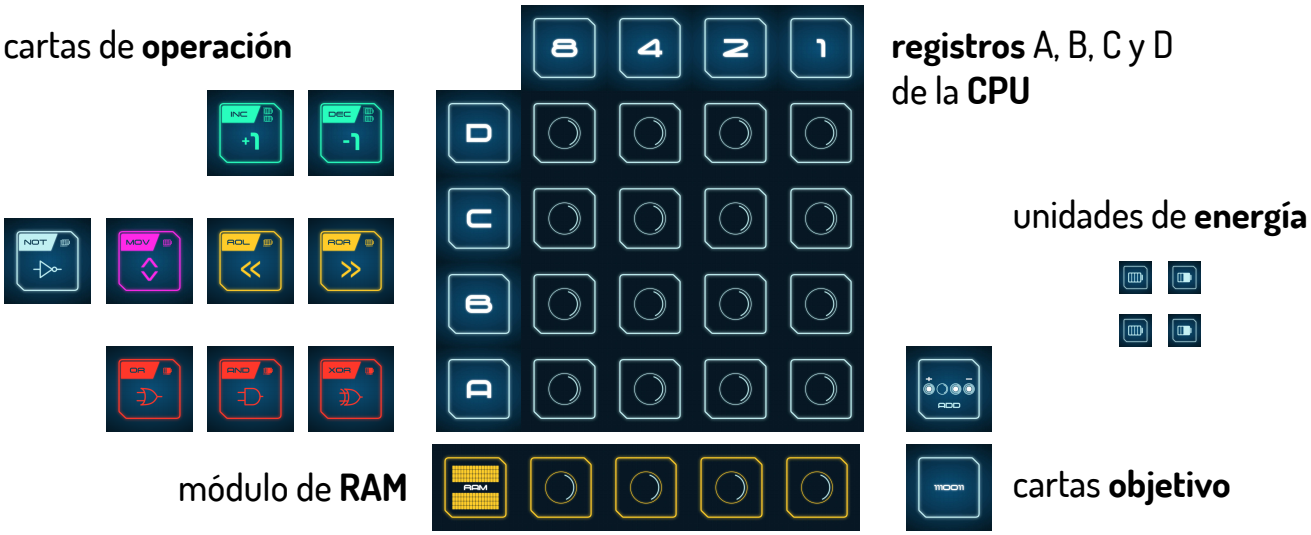
110011

MOON es un juego educativo en el que simularemos ser programas que se ejecutan en una computadora.

Jugando a MOON aprenderás a contar en binario, realizarás operaciones lógicas y descubrirás cómo funciona un ordenador por dentro al mismo tiempo que pasas un rato divertido.

MOON está recomendado para edades a partir de 11 años, de 1 a 4 jugadores, con una duración de entre 15 y 45 minutos (dependiendo de la dificultad elegida).

PREPARAR UNA PARTIDA



1. Sitúa los 4 registros de la **CPU** y sus correspondientes bits **apagados** en el medio de la mesa. Si es tu primera partida, es recomendable empezar con **4 bits** por registro.

2. Coloca las cartas de **operación** a la izquierda del tablero central, ordenadas en función de su coste de energía, primero las que cuestan 1/2 unidad de energía (OR, AND, XOR), luego las que cuestan 1 (NOT, ROL, ROL, MOV) y finalmente las que cuestan 2 (INC y DEC).

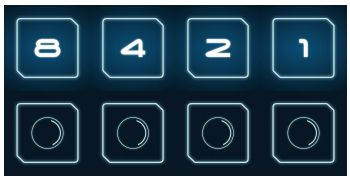
3. Baraja el mazo de cartas **objetivo** y colócalo a la derecha de los registros.

4. Da una carta de **RAM** a cada jugador y coloca tantos bits **apagados** como sea necesario junto a ellos.

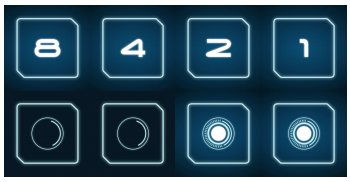
MOON **simula** el funcionamiento de una computadora **real** empleando operaciones y datos idénticos a los que se usan en microprocesadores reales. Por lo tanto, es conveniente repasar algunos **conceptos** antes de poder jugar:

CONTAR EN BINARIO

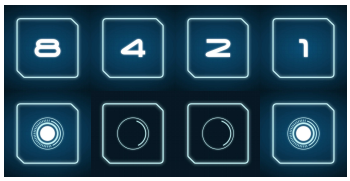
Tanto los módulos individuales de RAM como los registros de la CPU tienen diferentes bits que sirven como contadores binarios. Cada bit tiene un número asociado (del 1 al 8 en registros de 4 bits). Si todos los bits de todos los recuadros están a cero, se estará representando el número cero.



Si hay bits a uno, hay que sumar los números de los recuadros que tienen sus bit a uno para saber qué número se representa.



Por ejemplo, essta combinación representa el número 3 porque los bits de las posiciones 1 y 2 están activados, así que 1 + 2 = 3.



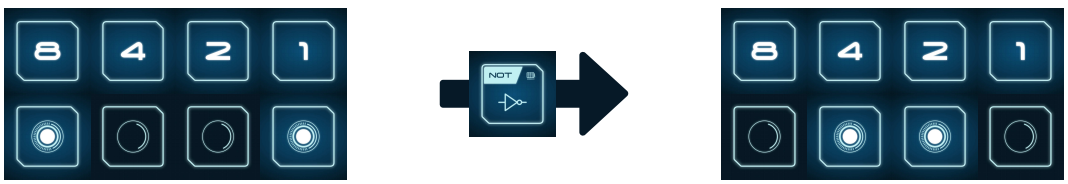
Esto representa el número 9 porque los bits de las posiciones 1 y 8 están activados, así que 1 + 8 = 9.

OPERACIONES

Para convertir los bits de los registros de la CPU en otros valores que necesitemos durante el juego, usamos las **operaciones**:

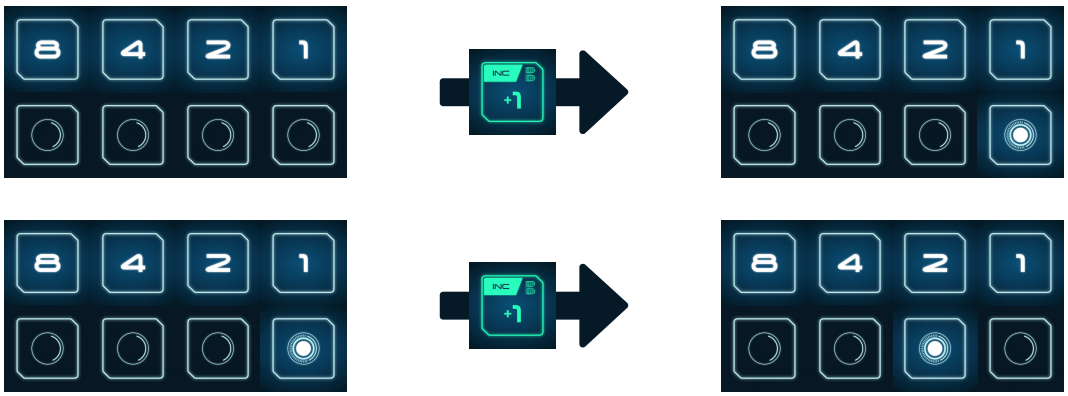
NOT

Esta operación se usa sobre **1 registro** y cuesta **1 unidad** de energía. **Niega**, es decir, pone lo **contrario** de lo que almacena cada uno de los bits del registro: si había un 1, pone un 0 y si había un 0, pone un 1 (damos la vuelta a todos los bits del registro):

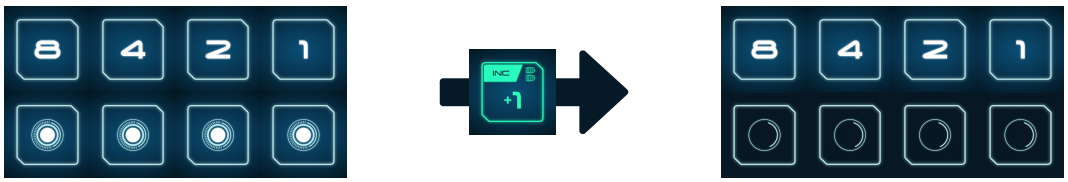


INC

Esta operación se usa sobre **1 registro** y cuesta **2 unidades** de energía **Suma** uno al valor almacenado en el registro:

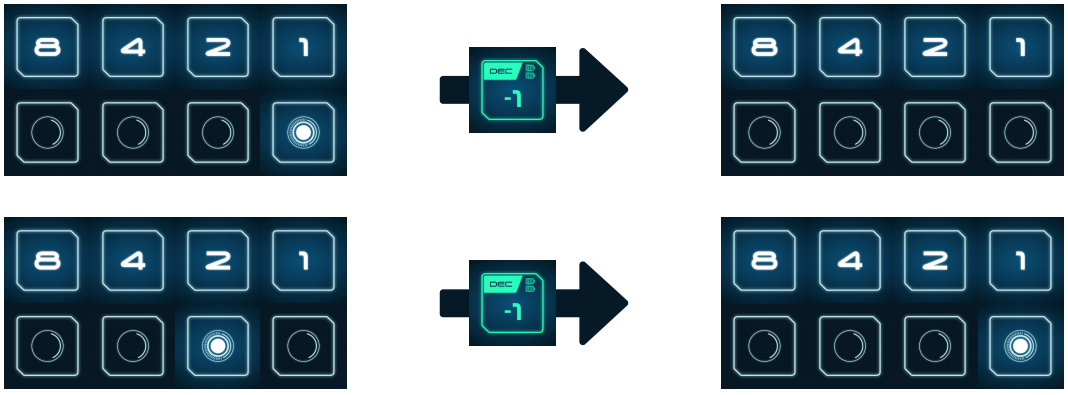


Si el registro almacena el valor **máximo** (todos los bits a 1), **se da la vuelta** al contador:

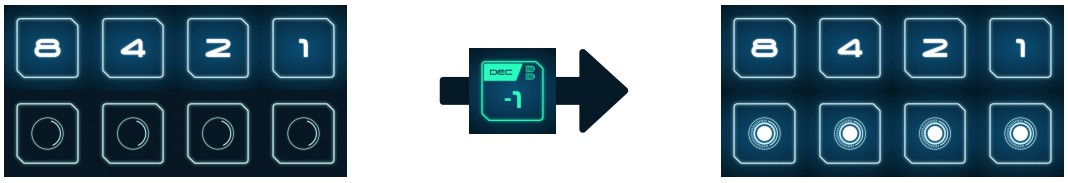


DEC

Esta operación se usa sobre **1 registro** y cuesta **2 unidades** de energía **Resta** uno al valor almacenado en el registro:

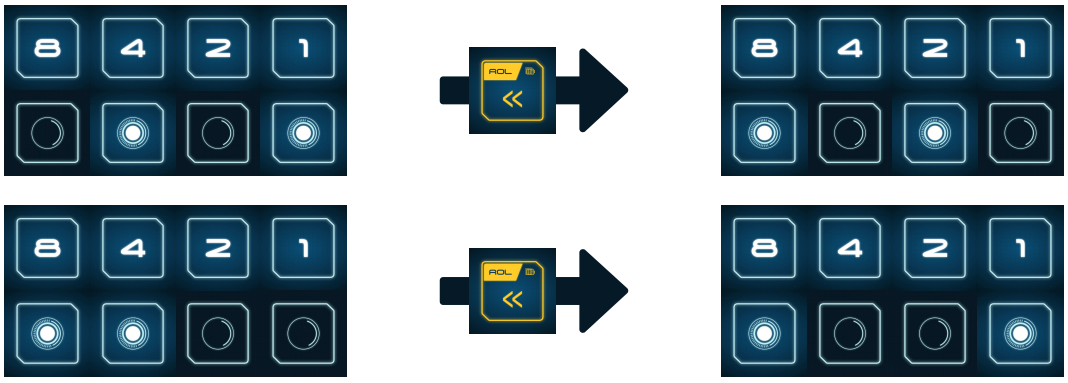


Si el registro almacena el valor **mínimo** (todos los bits a 0), **se da la vuelta** al contador:

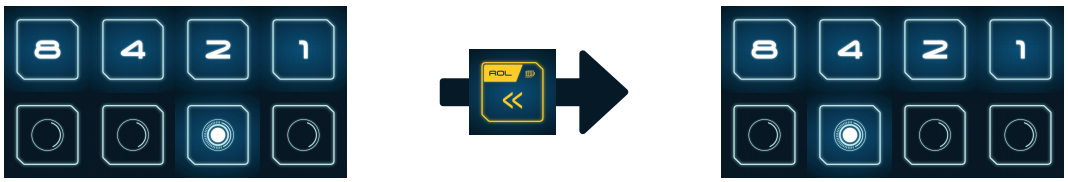


ROL

Esta operación se usa sobre **1 registro** y cuesta **1 unidad** de energía. **Desplaza** cada bit a la siguiente posición hacia la **izquierda**, salvo el último bit que ocupa la posición libre que ha dejado el primero:

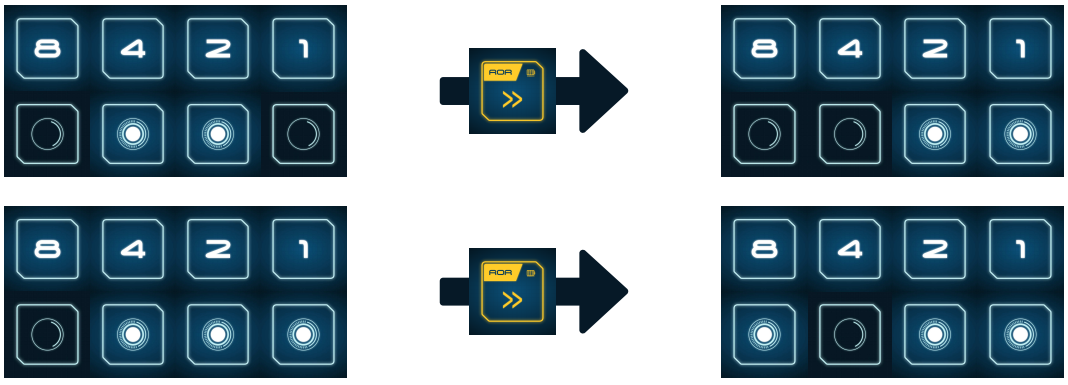


¡Truco! En muchas ocasiones, es equivalente a **multiplicar por 2** el valor del registro:

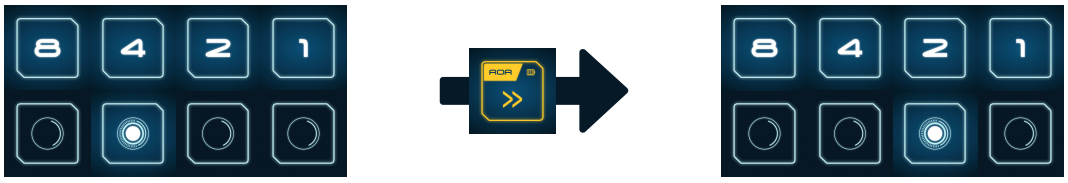


ROR

Esta operación se usa sobre **1 registro** y cuesta **1 unidad** de energía. **Desplaza** cada bit a la siguiente posición hacia la **derecha**, salvo el primer bit que ocupa la posición libre que ha dejado el último:



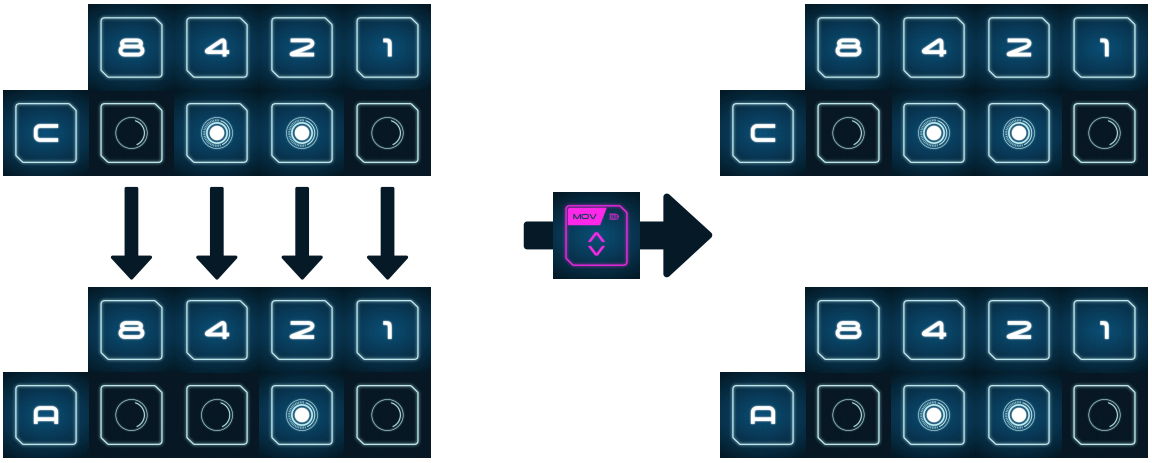
¡Truco! En muchas ocasiones, es equivalente a **dividir entre 2** el valor del registro:



MOV

Esta operación se usa sobre **2 registros** o sobre **un registro y la memoria RAM** y cuesta **1 unidad** de energía (1/2 en modo competitivo).

Copia todos los bits de un registro a otro, **sobreescribiendo** lo que contenía el destino (muy útil para copiar un valor en tu RAM y recuperarlo después sin que nadie te lo pueda modificar con sus operaciones).

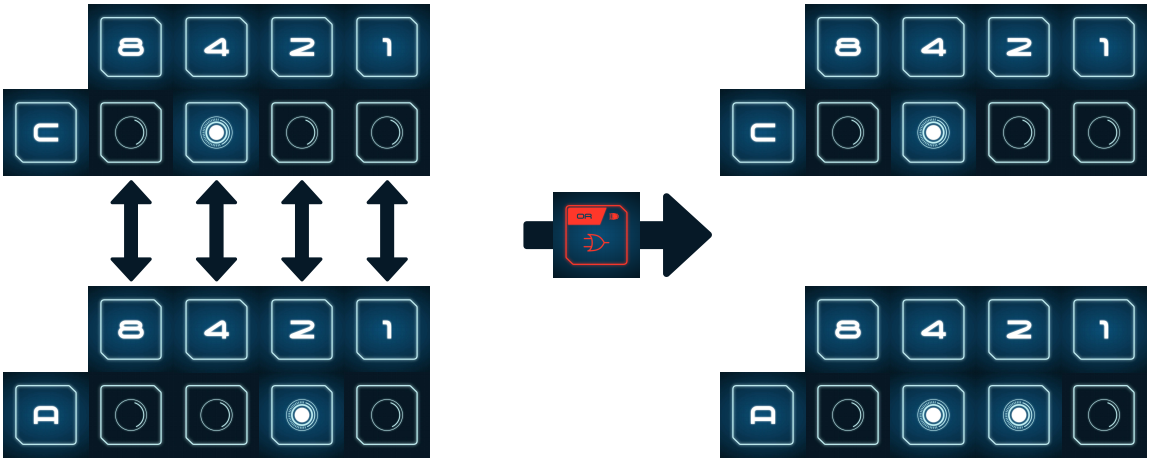


OR

Esta operación se usa sobre **2 registros** y cuesta **1/2 unidad** de energía.

Consiste en ir **comparando** cada bit de un registro con el bit correspondiente del otro registro (el 1º con el 1º, el 2º con el 2º, etc.) y si **alguno de los dos** es un 1, el resultado es un 1 (en caso contrario, el resultado es un 0).

El resultado final de todas esas comparaciones se almacena en el **primer registro** (el segundo se queda como estaba):

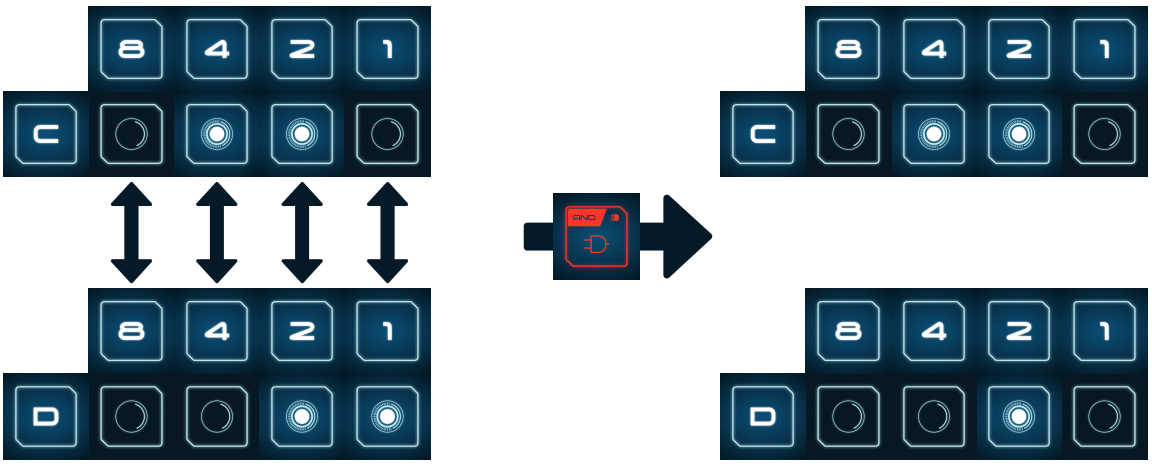


AND

Esta operación se usa sobre **2 registros** y cuesta **1/2 unidad** de energía.

Consiste en ir **comparando** cada bit de un registro con el bit correspondiente del otro registro (el 1º con el 1º, el 2º con el 2º, etc.) y si **ambos** contienen un 1, el resultado es un 1 (en caso contrario, el resultado es un 0). -

El resultado final de todas esas comparaciones se almacena en el **primer registro** (el segundo se queda como estaba):

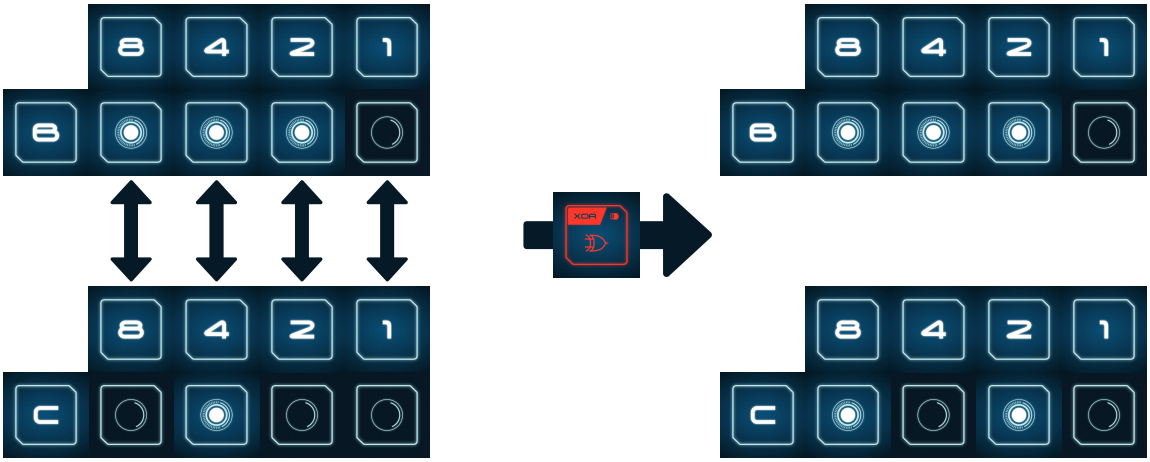


XOR

Esta operación se usa sobre **2 registros** y cuesta **1/2 unidad** de energía.

Consiste en ir **comparando** cada bit de un registro con el bit correspondiente del otro registro (el 1º con el 1º, el 2º con el 2º, etc.) y si contienen **un valor diferente** (un 0 en uno y un 1 en el otro o viceversa), el resultado es un 1 (en caso contrario, el resultado es un 0).

El resultado final de todas esas comparaciones se almacena en el **primer registro** (el segundo se queda como estaba):

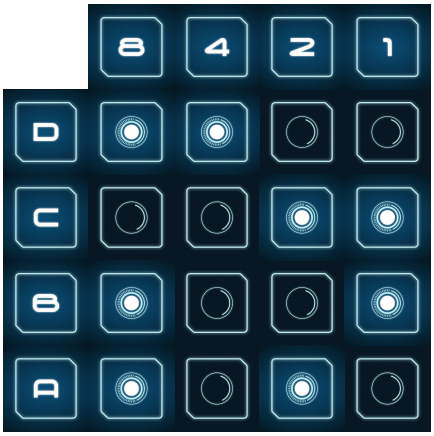


¡Truco! Si necesitas inicializar a 0 un registro, puedes usar una XOR de ese registro consigo mismo (ejemplo: XOR de A con A) y como no hay ninguna posición diferente, el resultado será poner todos los bits a 0.

ARQUITECTURA

Finalmente, vamos a repasar algunas ideas importantes sobre la **arquitectura interna** de una computadora:

Para poder lograr sus propósitos, los programas hacen operaciones en la CPU. Cuando es el turno de un programa, puede emplear cualquier registro (A, B, C y D) para hacer operaciones, pero **el registro A** es donde tiene que quedar almacenado el resultado final.



Esto en MOON supone que cualquier jugador podrá cambiar **cualquier registro** de la CPU mientras juega, pero tendrá que conseguir **su objetivo en el registro A** para ganar.

Los programas tienen asignado un tiempo para hacer operaciones y cuando lo acaben, tienen que **ceder la CPU** al siguiente programa.

En MOON cada jugador dispone de varias fichas de energía para cada turno, que usará para poder realizar operaciones y luego tendrá que **ceder la CPU** al siguiente jugador.

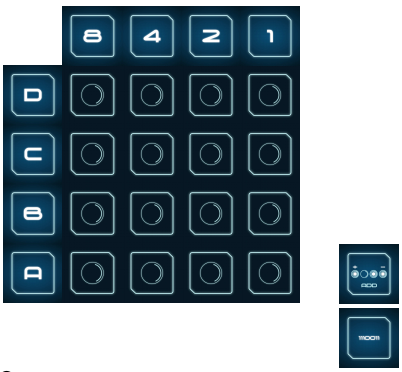
Como todos los programas **comparten** la CPU, a veces conviene **guardar** resultados parciales en la memoria RAM. Al contrario de lo que ocurre con la CPU, **la memoria RAM es propia** de cada programa y ningún programa puede modificar la RAM de otro programa.

En MOON, cada jugador tiene un módulo **individual** de RAM donde puede copiar alguno de los valores de los registros de la CPU (usando la operación **MOV**) y luego puede copiarlo de vuelta a cualquier registro de la CPU (usando MOV otra vez).

MODO COOPERATIVO

Tanto el los registros de la CPU como los módulos individuales de RAM pueden usarse con 4, 5 o 6 bits. Para las primeras partidas **sugerimos** usar **4 bits**.

Prepara la partida como se explica en la página 1 de este manual.



Al principio de la partida, toma una carta del mazo de cartas objetivo y colócala delante del mazo por la cara en la que se puede ver la combinación de bits objetivo:

Para **ganar**, deberéis resolver **todas** las cartas objetivo del mazo.

Las cartas objetivo contienen combinaciones de bits que tenéis que lograr en el registro A de la CPU. En cada turno, cada participante podrá realizar tantas **operaciones** como desee en función de las **unidades** de energía se disponga (recuerda que hay operaciones como OR que requieren 1/2 unidad de energía, mientras que otras como INC requieren 2). No es obligatorio gastar todas las unidades de energía en un turno y **no es posible ceder** las que no se desee usar a otros participantes.

Las cartas de operación realizan operaciones sobre los registros A, B, C y D de la CPU. Cualquier participante puede **modificar** los valores de todos los bits **en los 4 registros** de la CPU, pero no podrá **copiar** o **modificar** los valores almacenados en la **RAM** del resto de participantes.

Tanto si habéis logrado resolver la carta objetivo como si no, al terminar la ronda tendréis hacer **avanzar las cartas objetivo** hacia arriba una posición, tomar una nueva carta objetivo del mazo y colocarla delante del mazo:

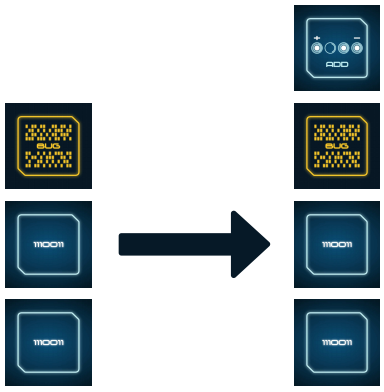
Si al finalizar una ronda una carta objetivo avanza hasta la **5ª posición**, habréis tardado demasiado, la partida terminará y habréis **fracasado**.

Esto puede ocurrir incluso si no quedan cartas objetivo en el mazo pero tardáis **más de 5 rondas** en resolver las últimas cartas. Es decir, las cartas objetivo van avanzando hacia adelante al final de cada ronda independientemente de que queden cartas en el mazo o no.

Si por el contrario, lográis resolver todos los objetivos del mazo sin que esto ocurra, habréis **ganado**.

Además, hay cartas objetivo que tienen no tienen una combinación de bits sino un **bug**.

Estas cartas especiales **no se pueden descartar** y os bloquearán una de las posiciones dentro de la lista de objetivos pendientes para el resto de la partida.



En función del **número de jugadores** y de la **dificultad** escogida, varía el número de unidades de energía por turno, el tamaño del mazo de cartas objetivo y el número de cartas objetivo que inicializarán los registros.

En el nivel **aprendiz**, se cogerán las primeras 3 cartas objetivo y se pondrán sus valores en los registros B, C y D. En nivel **normal**, se hace lo mismo con las primeras 2 cartas objetivo y los registros B y C. En el nivel **hacker**, se pondrá el valor 1 en el registro A, y el resto de registros a 0.

NIVEL **APRENDIZ**

4-BIT

1	4	8 +
2	4	10 +
3	3	12 +
4	3	16 +

6-BIT

1	5	8 +
2	5	19 +
3	4	12 +
4	4	16 +

NIVEL **NORMAL**

4-BIT

1	3	8 +
2	3	10 +
3	2	12 +
4	2	16 +

6-BIT

1	4	8 +
2	4	10 +
3	3	12 +
4	3	16 +

NIVEL **HACKER**

4-BIT

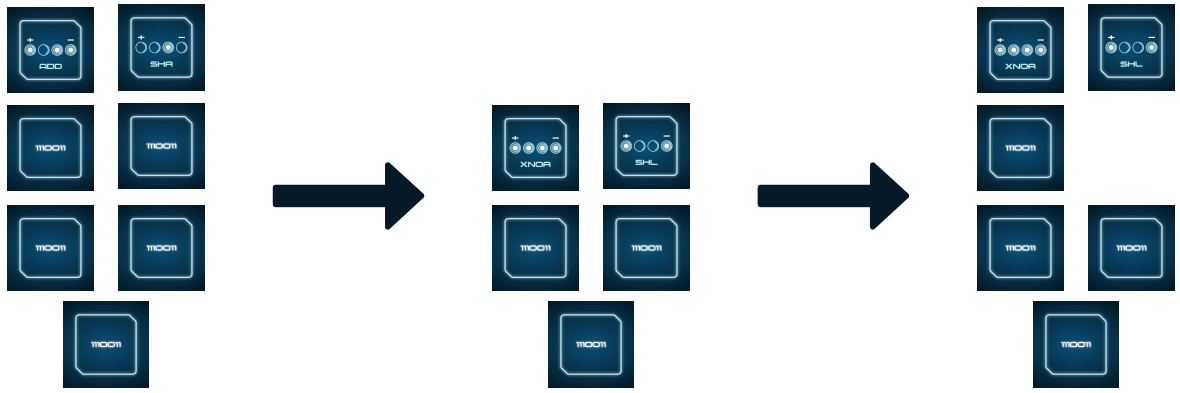
1	1.5	8 +
2	1.5	10 +
3	1	12 +
4	1	16 +

6-BIT

1	3	8 +
2	3	10 +
3	2	12 +
4	2	16 +

En el caso de la versión de **6 bits**, las cartas objetivo se deben resolver **de dos en dos** (la de la izquierda para los bits 32 y 16, y la de la derecha para los bits 8, 4, 2 y 1).

Sin embargo, al final de cada ronda **solamente se colocará una carta** objetivo nueva encima de la mesa (esto hace que el ritmo de objetivos sea la mitad de rápido que en la versión de 4 bits):

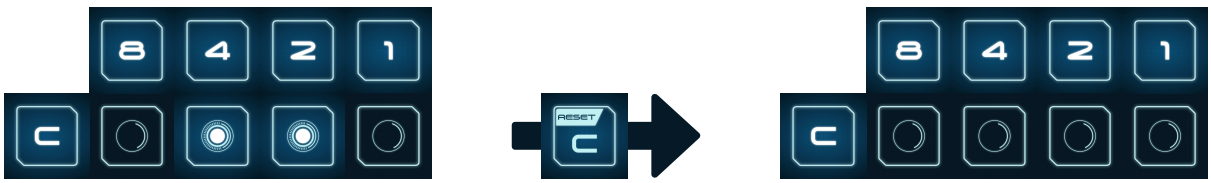


RESOLVEMOS
UN OBJETIVO

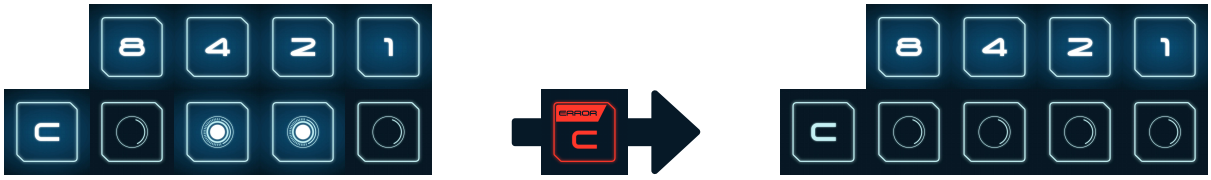
MOSTRAMOS EL
SIGUIENTE
OBJETIVO

AL TERMINAR
LA RONDA,
COLOCAMOS
SOLO UNA NUEVA
CARTA OBJETIVO

Si el nivel hacker se te queda corto, puedes aumentar la complejidad de las partidas mezclando las **cartas de eventos** dentro del mazo de objetivos. Las cartas **RESET registro, reinician** el registro indicado:



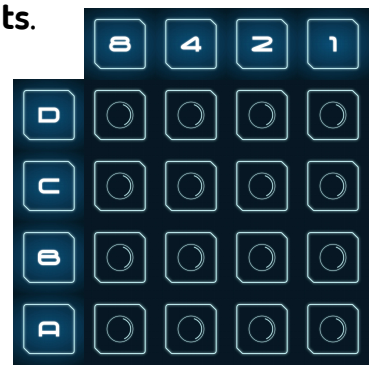
Las cartas **ERROR registro, inutilizan** el registro indicado, que solamente podrá usarse de nuevo en la partida si se consigue una carta **RECOVER**:



Las cartas **ERROR instrucción, inutilizan** la instrucción hasta el final de la partida, salvo que se use una carta **RECOVER** para repararlas.

MODO COMPETITIVO

Tanto el los registros de la CPU como los módulos individuales de RAM pueden usarse con 4, 5 o 6 bits. Para las primeras partidas **sugerimos** usar **4 bits**.



Prepara la partida como se explica en la página 1 de este manual.

Cada participante elige un color, toma la carta de RAM de ese color y coloca un bit a cero en todas las posiciones de su **módulo de RAM**.

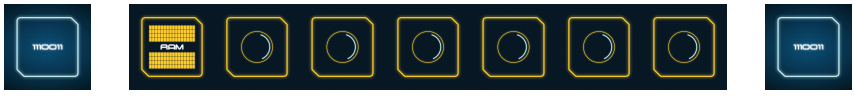
Reparte las unidades de energía a cada participante en función de la versión (4-6 bits) y el nivel (aprendiz, normal, hacker) de cada participante:

	APRENDIZ	NORMAL	HACKER
4-bit	4	3	2
6-bit	5	4	3

Baraja el mazo de **cartas objetivo** y colócalo a la derecha de la CPU.

En partidas de **4 bits,** cada participante toma una carta objetivo, la ve y la coloca junto a su módulo de RAM sin mostrar la combinación de bits correspondiente.

En partidas de **6 bits**, cada participante tomará **dos cartas** y las colocará una a cada lado de su módulo de RAM: la carta de la **derecha** corresponderá a los bits 8, 4, 2 y 1 del registro A y la carta de la **izquierda** a los bits 32 y 16.

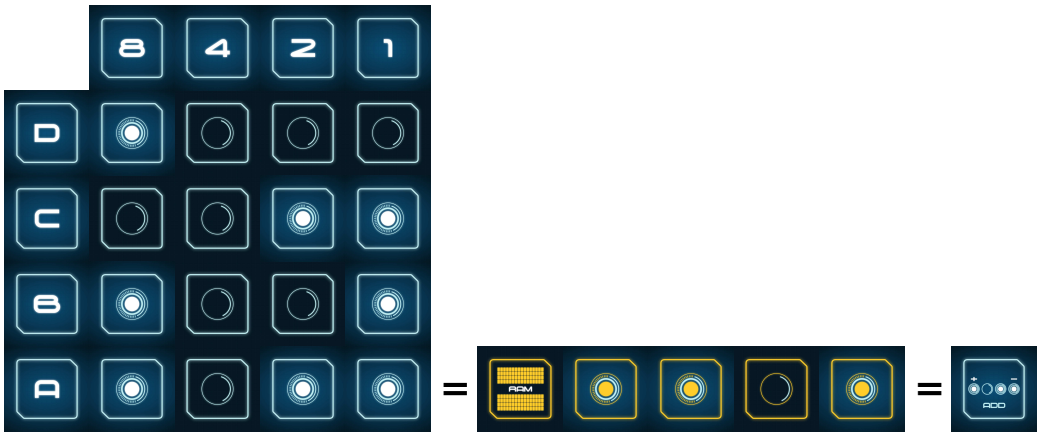


Las cartas objetivo contienen **combinaciones de bits** que tenéis que lograr en el **registro A** de la CPU. Al finalizar el mazo de cartas objetivo, **ganará** quien más objetivos haya logrado resolver.

En cada turno, cada participante deberá jugar tantas cartas de **operación** como **unidades de energía** se dispongan. No es obligatorio jugarlas todas, pero **no es posible ceder** energía que no se desee usar a otros participantes.

Las cartas de operación realizan operaciones sobre los registros A, B, C y D de la CPU. Cualquier participante puede modificar los valores de todos los bits **en los 4 registros** de la CPU, pero no podrá **copiar** o **modificar** los valores almacenados en la RAM del resto de participantes.

Si un participante logra **durante su turno** que el **registro A** de la CPU contenga su combinación de bits objetivo, **mostrará** su carta objetivo al resto de participantes, la **guardará** junto a su módulo de RAM y **tomará otra** carta objetivo del mazo.



En el caso de que sea una carta **bug**. se **mostrará** a los participantes y se guardará para su posterior uso (**no cuentan** como cartas objetivo en el recuento del final de la partida). En modo competitivo, una carta bug sirve para poder **mostrar la carta objetivo de otro participante** en cualquiera de sus turnos durante la partida. Al usarla, la carta bug **pasa a estar en posesión** de quien ha tenido que mostrar su carta objetivo y podrá usarla cuando lo desee contra cualquier participante.

Cuando un participante tome **la última carta del mazo de objetivos**, la partida **terminará** al terminar la ronda. **Ganará** quien **más objetivos** haya conseguido durante la partida.

EXTRAS

MOON es un juego **modular** que puedes **adaptar** a tus necesidades. Por ejemplo, jugadores muy noveles pueden eliminar las operaciones lógicas complejas (OR, AND, XOR) del juego en las primeras partidas. En el otro extremo, si quieres **jugar con 8 bits,** basta que juntes dos MOON de 4 bits.

Además de esta recomendación general, terminamos este manual explicando dos aspectos **extra** que pueden mejorar la experiencia de juego.

OPERACIONES EXTRA

En cada carta de operación hay unas siglas que corresponden a **operaciones extra** que puedes realizar **una vez** tras conseguir ese objetivo.

XCHG funciona como una MOV bidireccional, intercambia los valores de dos registros.

SHL y **SHR** funcionan de forma similar a ROL y ROR respectivamente, con la salvedad de que el bit que se queda fuera, siempre se convierte en un cero (ejemplo: SHL de 1011 = 0110).

ADD y **SUB** permiten sumar y restar aritméticamente dos registros (ejemplos: 0101 ADD 1010 = 1111 y 1101 ADD 1000 = (1)0101, el primer 1 no se pone si usamos registros de 4-bits, se pierde).

NOR, NAND, XNOR son la combinación de OR, AND o XOR y una NOT al final (por ejemplo, el resultado de NOR solo será 1 si sus dos entradas son 0).

HACKERS

Decide qué hacker quieres ser, con habilidades especiales diferentes:

- Hacker de las **sumas (azul)**: puede usar INC o DEC consumiendo solo una unidad de energía.

- Hacker de las **copias (morado)**: puede hacer una MOV sin consumir energía en cada turno.

- Hacker de las **rotaciones (amarillo)**: puede usar ROL o ROR consumiendo solo media unidad de energía.