

ווסווו



Control the computer
of the lunar module



In 1969, millions of people in the Earth were following by TV the events that happened 384,000 km away. Three minutes before landing on the Moon, the on-board computer of the Eagle lunar module triggered several alarms. Due to a flaw in the manuals, a radar not needed for landing was switched on when it should be switched off. This radar overloaded the on-board computer, which was taking care of all the operations involved in landing. Luckily, the software that managed the on-board computer was designed by a team of engineers led by Margaret Hamilton, and it was intelligent enough to detect the problem. The computer alerted the astronauts, saying, "I am overloaded with more tasks than I should be doing right now, so I'm going to focus only on the important tasks, those that have to do with with landing." Remember it was a computer with less computing power than a digital clock. Without this novel and intelligent design, the "small step for man and great leap for Humanity" would have ended in total crash.

It took more than 30 years before NASA recognized Margaret Hamilton's merit. Hamilton was the director of MIT's software engineering center in charge of develop the onboard software for the entire Apollo program, the only one that has been able to to let us step on other worlds and come back safely.

110011

MOON is an educational game for 1 to 4 people (estimated minimum age to play: 11 years and older).

Playing MOON you will learn how to count in binary, perform logical operations and find out how a computer works while you're having fun.

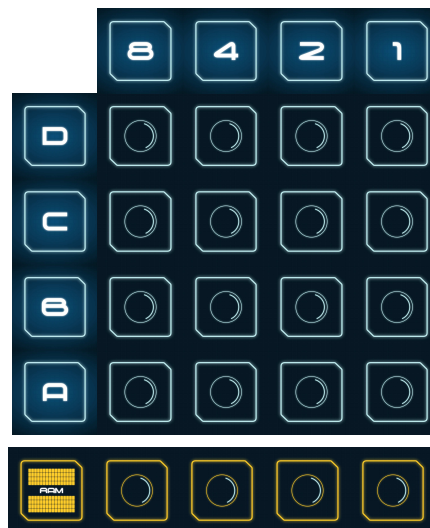
In MOON, each player will simulate being a computer program and try to get the desired result (a combination of bits) before the others.

SET UP

operation cards



RAM module



registers A, B, C y D of the CPU

power units



goal cards

1. Place the **4 CPU registers** and their corresponding bits **switched off** in the middle of the table. If this is your first game, we recommended you to start with 4 bits per register.
2. Place the **operation cards** on the left of the central board, sorted according to their cost of power: first those that cost 2 power units (INC, DEC), then those the ones that cost 1 power unit (NOT, ROL, ROL, ROL, MOV), finally those that cost 1/2 power units (OR, AND, XOR).
3. Shuffle the **goal cards** and place the deck on the right side of the registers.
4. Give each player a **RAM card** and place as many bits as necessary near to them.

MOON simulates a real computer. Operations modify data in the same way it happens in **real microprocessors**. Therefore, we think it is a good idea to remember some computer related **concepts** before you start playing.

COUNTING IN BINARY

Both the individual RAM modules and the CPU registers have several bits that work as **binary counters**. Each bit has an associated number (from 1 to 8 in 4-bit registers). If all the bits of a register are **switched off**, the value **zero** is stored.

If there are bits switched on, you have to add the numbers placed on the top of the CPU to know what number is stored.



For example, this combination represents the number 3 because the bits of positions 1 and 2 are active, so $1 + 2 = 3$.

This represents the number 9 because the bits of the positions 1 and 8 are activated, so $1 + 8 = 9$.



OPERATIONS

To change the bits of the CPU registers we use **operations**:

NOT

This operation is used on a **single register** and costs **1 power unit**. It **inverts** every bit on the register, that is, it converts the **zeros** (bits switched off) into **ones** (bits switched on) and vice versa. This involves **flipping all the bit cards** of a register.

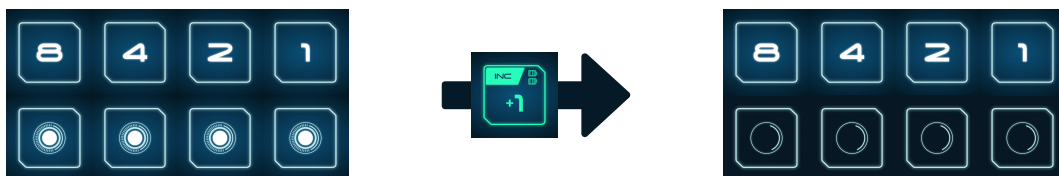


INC

This operation is used on a **single register** and costs **2 power units**. It **adds 1** to the total value stored in the register:



. If the register stores the **maximum** value (all bits switched on), the register is **reset** to 0:

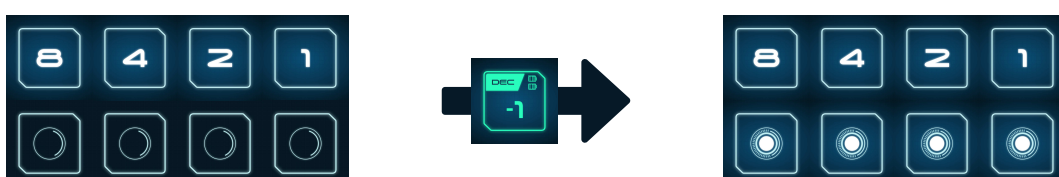


DEC

This operation is used on a **single register** and costs **2 power units**. It **subtracts 1** to the total value stored in the register.

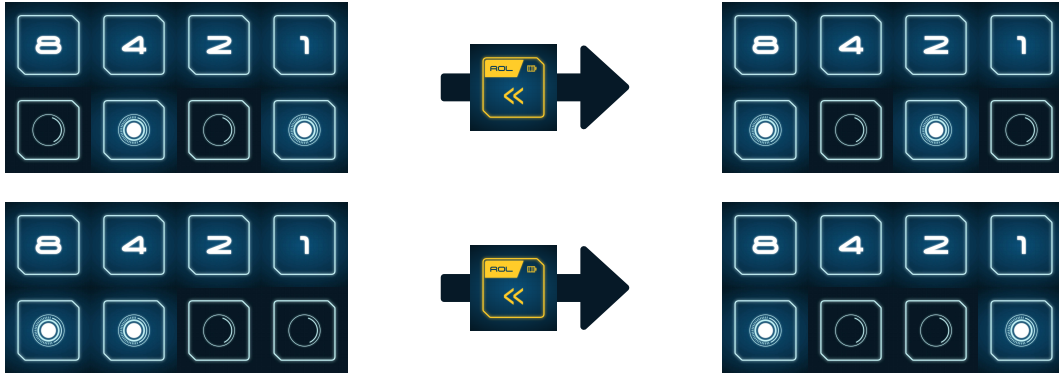


If register stores the **zero** value, subtracting 1 will set all the bits of the register to **one**.

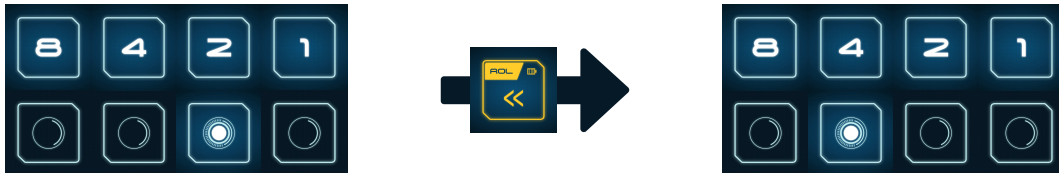


ROL

This operation is used on a **single register** and costs **1 power unit**. It involves **rotating** every bit on the register to the **left** and placing the remaining bit on the left in the rightmost position:

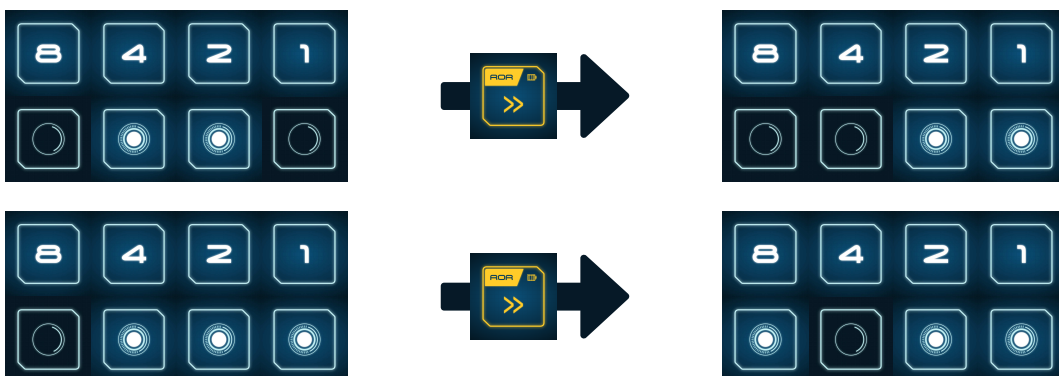


Trick! In many cases, it is equivalent to **multiplying** the value of the register by 2:

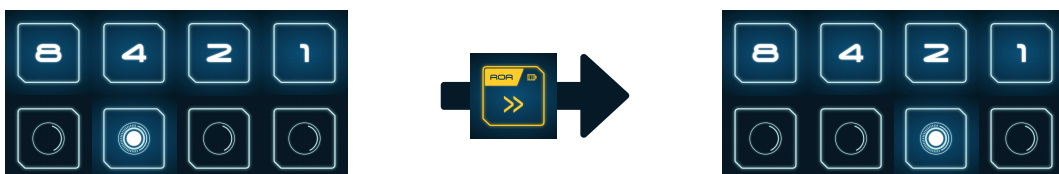


ROR

This operation is used on a **single register** and costs **1 power unit**. It involves **rotating** every bit on the register to the **right** and placing the remaining bit on the right in the leftmost position:



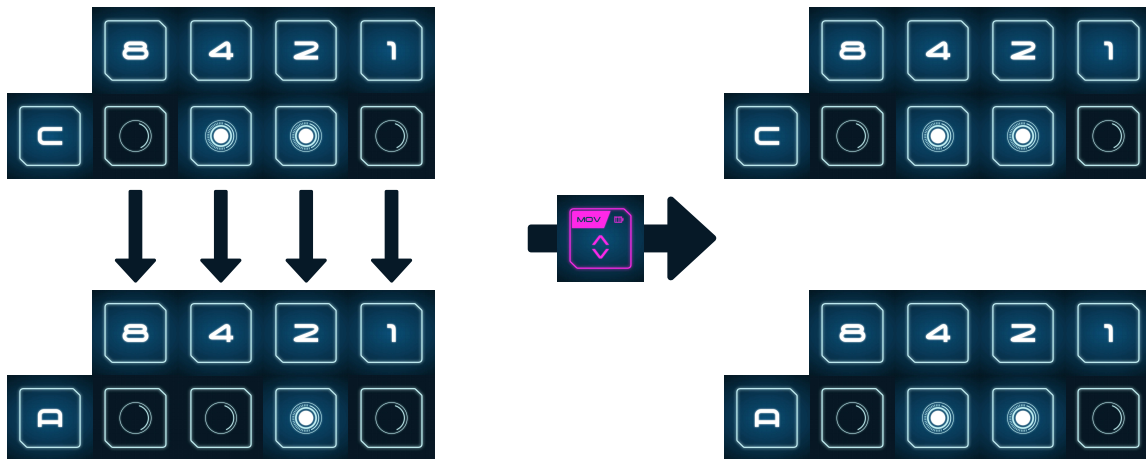
Trick! In many cases, it is equivalent to **dividing** the value of the register by 2:



MOV

This operation is used on **2 registers** or a **register and a RAM module** and costs **1 power unit** (1/2 in competitive mode).

This operation **copies** all bits from one register to another, **overwriting** the value stored in the destination (it is very useful to copy a value in your RAM and recover it later to prevent other players modifying it).

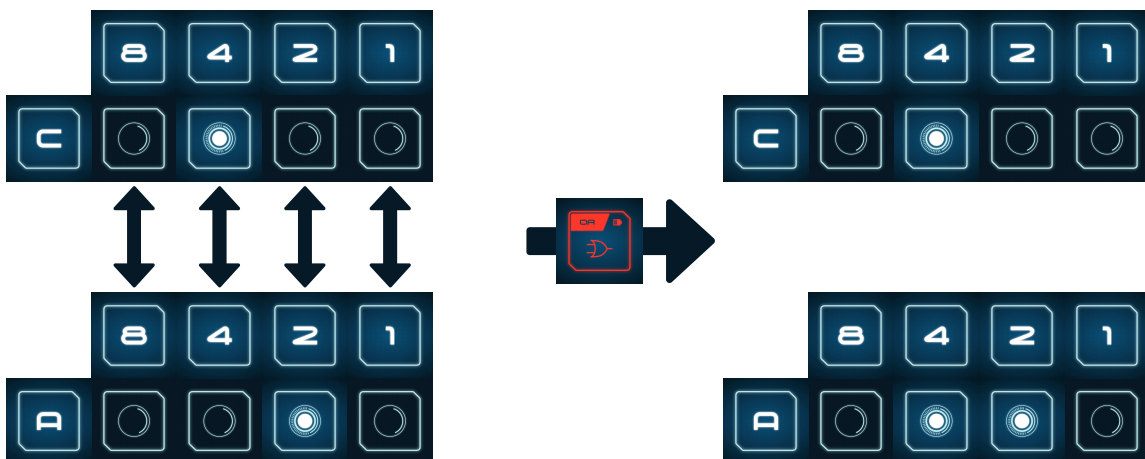


OR

This operation is used on **2 registers** and costs **1/2 power unit**.

It involves **comparing** each bit of one register with the corresponding bit of the other (the 1st bit with the 1st bit, the 2nd with the 2nd, etc.). If **any of the bits** is switched on, the resulting bit is switched on (otherwise, the resulting bit is switched off).

The final result of all these comparisons is stored in the **first register** (the second is not modified):

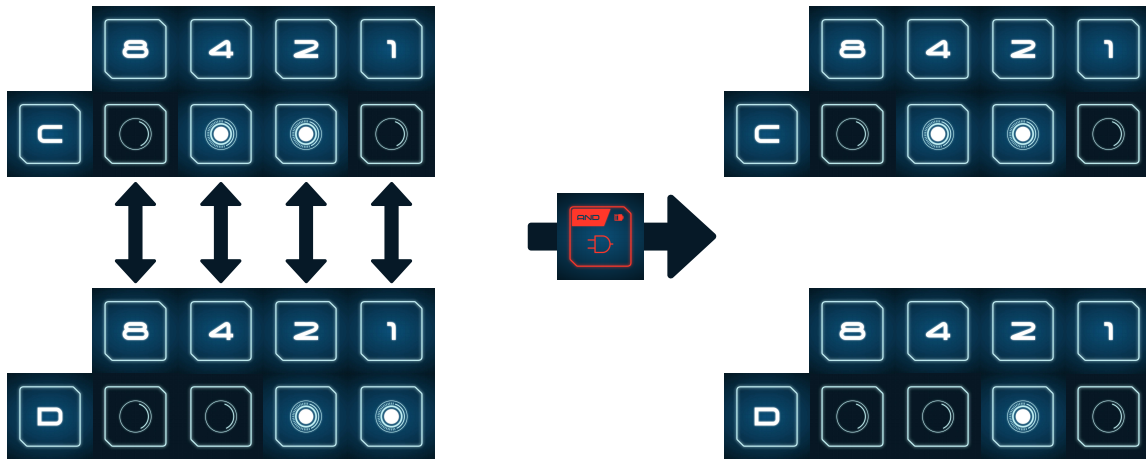


AND

This operation is used on **2 registers** and costs **1/2 power unit**.

This operation involves **comparing** each bit of one register with the corresponding bit of the other (the 1st bit with the 1st bit, the 2nd with the 2nd, etc.). If **both** bits are switched **on**, the resulting bit is switched on (otherwise, the resulting bit is switched off).

The final result of all these comparisons is stored in the **first register** (the second is not modified):

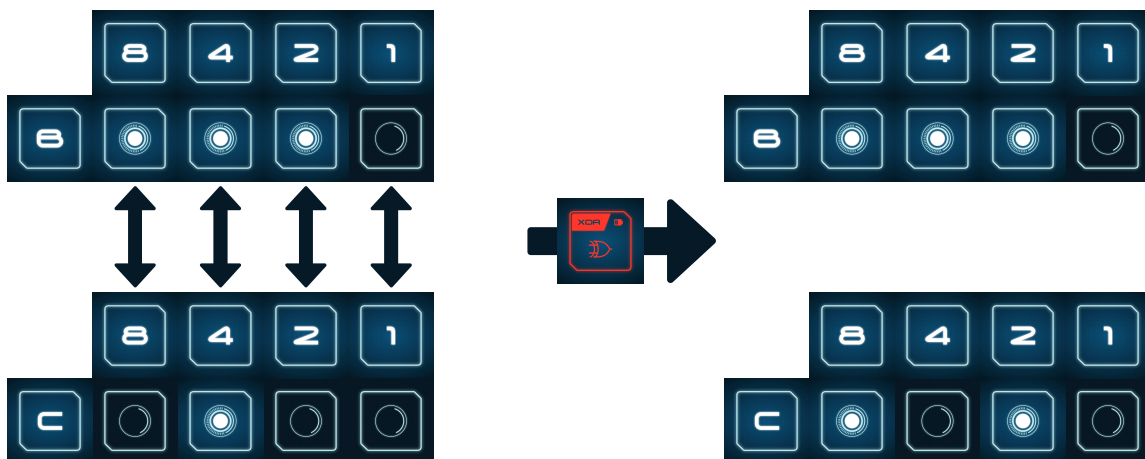


XOR

This operation is used on **2 registers** and costs **1/2 power unit**.

This operation involves **comparing** each bit of one register with the corresponding bit of the other. If both bits are **different** (one is switched on and the other is switched off), the resulting bit is switched on (otherwise, the resulting bit is switched off).

The final result of all these comparisons is stored in the **first register** (the second is not modified):



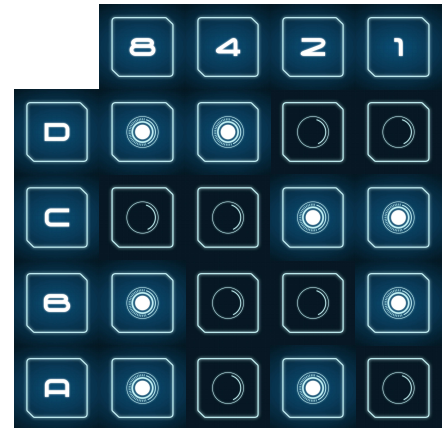
Trick! If you need to reset a register (all the bits switched off), you can use a XOR where the source and destination register is that register (example: XOR of A with A) because all bits are the same.

COMPUTERS ARCHITECTURE

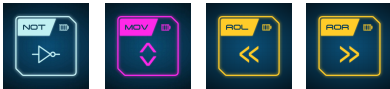
Finally, let's review some important ideas about the **internal architecture** of a computer:

In order to achieve their purposes, programs do **operations** on the CPU. When a program runs, it can use any register (A, B, C and D) to perform operations, but the result has to be stored in the **A register**.

In MOON, this means that any player can modify **any CPU register** in her turn, but you will have to achieve your goal in the A register to win.



The operating system assigns time to programs to run. When they run out of time, they have to **leave the CPU** for other programs.

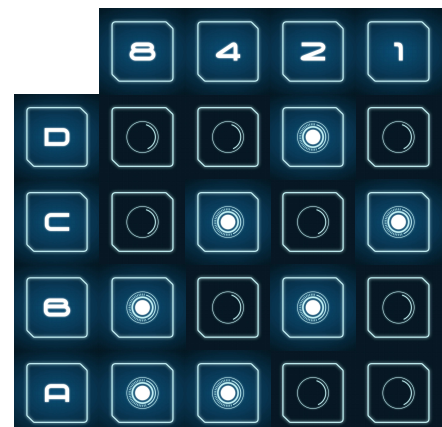


In MOON, each player has several power units for every turn. You will use them to perform operations and then **leave the CPU** registers to the next player.



Since all programs share the CPU, a partial results should be stored in the **RAM memory**. Contrary to what happens with the CPU registers, the RAM memory of each program is **protected** from other programs.

In MOON, each player has an **individual RAM** module where you can copy any of the values from the CPU registers (using the **MOV** operation) before your turn ends. Then you can copy it back to any register of the CPU (using MOV again) during your next turn.

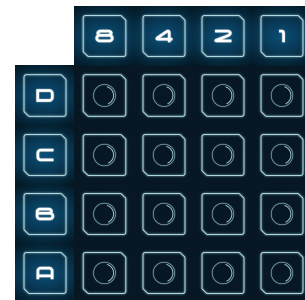


COOPERATIVE MODE

Both the CPU registers and the individual RAM modules can be used with 4, 5 or 6 bits. For the first games, we suggest using **4 bits**.

Set up the game as explained on page 1 of this manual.

At the beginning of the game, **draw** a card from the deck of goal cards and place it in front of the deck by the side where you can see the combination of bits:



To **win**, you must solve **all** the goal cards in the deck.

Goal cards have **combinations of bits** that you have to store in **register A** of the CPU.

In each turn, each player can perform as many **operations** as they wish depending on the power units available (remember that there are operations such as OR that require 1/2 power units, while others like INC require 2 power units). It's not mandatory to **spend** all power units in one turn and it is not possible to **share** the power units which other players.

Operation cards perform operations on the A, B, C and D registers of the CPU. Any player can modify the values stored in **all the 4 CPU registers**, but they cannot **copy** or **modify** the values stored in the the RAM modules of other players.



At **the end of the round**, you have to advance the goal cards up one position, draw a new goal card from the deck and place it in front of the deck:



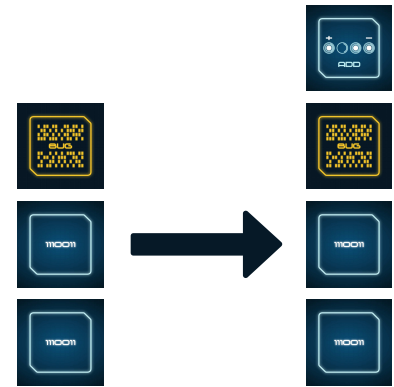
If a goal card advances to **5th position**, at the end of the round, the game is **over** and your mission **failed**.

This can happen even if there are no target cards left in the deck but it takes you more than **five rounds** to solve the last goal cards.

On the other hand, if you manage to solve all the goals of the deck promptly, you **win**.

Moreover, there are goal cards they have that don't have a combination of bits but a **bug**.

These special cards **cannot be discarded** and they will block one of the positions of the list of pending goals for the rest of the game.











You can adapt the **difficulty** of the game depending on the **number of players** and your **skills** varying the number of power units per turn, the size of the deck of goal cards and the number of goal cards that will initialize the registers.









In the **novice** level, draw the first 3 goal cards and place their values in registers B, C, and D respectively. In the **medium** level, the same is done with the first 2 goal cards. In the **hacker** level, the A register is initialized to the value 1 (off-off-off-on), and the rest of the registers are reset to 0.

NOVICE LEVEL

4-BIT









Player	Power Units	Goal Cards
1	4	8 +  
2	4	10 +  
3	3	12 +  
4	3	16 +  

6-BIT









Player	Power Units	Goal Cards
1	5	8 +  
2	5	19 +  
3	4	12 +  
4	4	16 +  

MEDIUM LEVEL

4-BIT









Player	Power Units	Goal Cards
1	3	8 +  
2	3	10 +  
3	2	12 +  
4	2	16 +  

6-BIT









Player	Power Units	Goal Cards
1	4	8 +  
2	4	10 +  
3	3	12 +  
4	3	16 +  

HACKER LEVEL

4-BIT

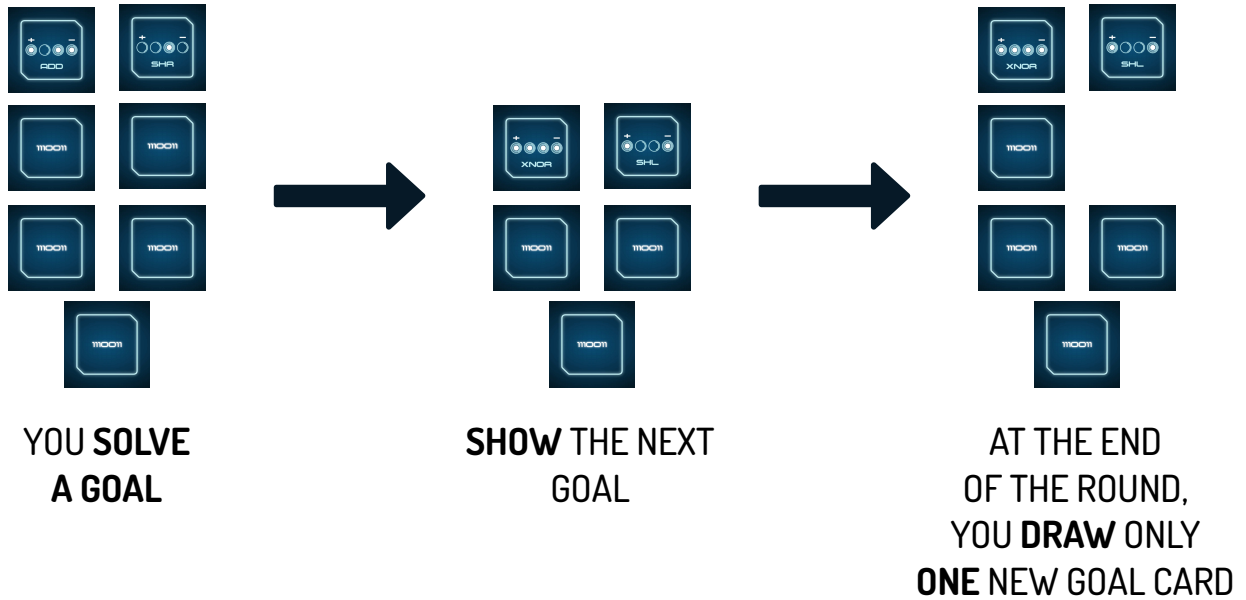
Player	Power Units	Goal Cards
1	1.5	8 +  
2	1.5	10 +  
3	1	12 +  
4	1	16 +  

6-BIT

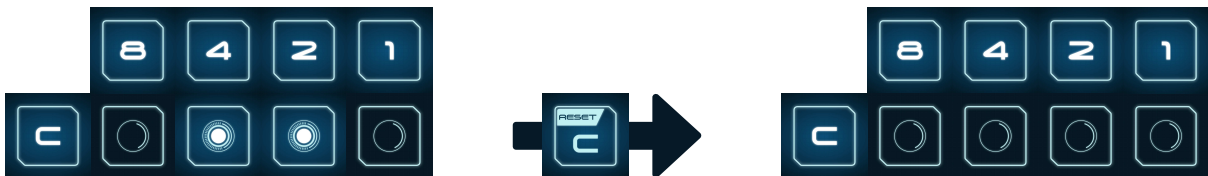
Player	Power Units	Goal Cards
1	3	8 +  
2	3	10 +  
3	2	12 +  
4	2	16 +  

In the case of the **6-bit** version, the goal cards must be solved in pairs (the **left card** for bits 32 and 16, and the **card on the right** for bits 8, 4, 2 and 1).

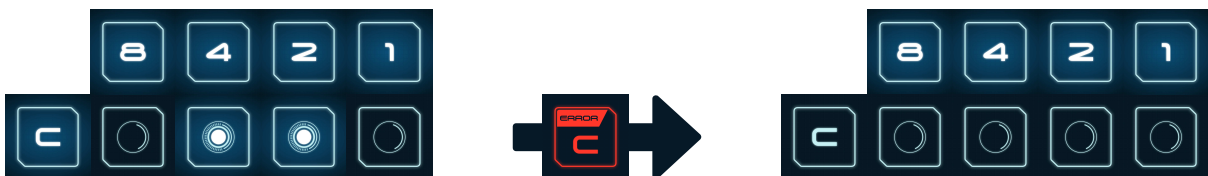
However, **only one** new goal card will be drawn at final for each round (this reduces the pace of the pending goals queue increase compared with the 4-bit version):



If the hacker level is a piece of cake for you, you can increase the difficulty adding the **event cards** into the goal card deck. **RESET** cards reset (all bits switched off) registers:



ERROR register cards disable the register for the rest of the game, unless you fix it using an **OK card**:



ERROR instruction cards disable the instruction for the rest of the game, unless you fix it using an **OK card**.

COMPETITIVE MODE

Both the CPU registers and the individual RAM modules can be used with 4, 5 or 6 bits. For the first games, we suggest using **4 bits**.

Set up the game as explained on page 1 of this manual.

Each player chooses a RAM card and sets all bits of the **RAM module** to zero (switched off).



Distribute the **power units** to each player according to the version (4-6 bits) and the level (novice, medium, hacker) of each player:

	NOVICE	MEDIUM	HACKER
4-bit	4	3	2
6-bit	5	4	3

Shuffle the deck of goal cards and place it to the right of the CPU.

In **4-bit** games, each player draws a goal card, sees it and places it next to her RAM module without showing the corresponding combination of bits.



In **6-bit** games, each player draws **two goal cards** and place them one on each side of her RAM module: the card on the **left** corresponds to bits 32 and 16 of register A, and the card on the **right**, to bits 8, 4, 2 and 1.

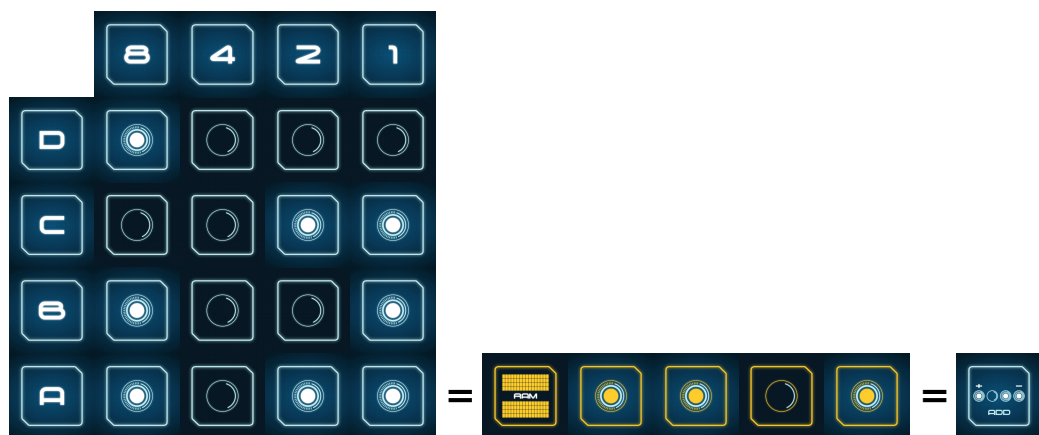


Goal cards have **combinations of bits** that you have to store in **register A** of the CPU.

In each turn, each player can perform as many **operations** as they wish depending on the power units available (remember that there are operations such as OR that require 1/2 power units, while others like INC require 2 power units). It's not mandatory to **spend** all power units in one turn and it is not possible to **share** the power units which other players.

Operation cards perform operations on the A, B, C and D registers of the CPU. Any player can modify the values stored in **all the 4 CPU registers**, but they cannot **copy** or **modify** the values stored in the the **RAM modules** of other players.

Every time you manage to have the combination of bits of your goal card in the **register A** of the CPU, you have to **show** your goal to the rest of players. Then, save it for the end of the game and **draw** another goal card from the deck.



In the case of drawing a **bug** card, you have to **show** it to the rest of the players and save it for the moment when you want to **hack** other player (bug cards do not count as goal cards at end of the game). In competitive mode, a bug card can be used to **force other player to show the goal card**. After using it, the hacked player **gets the bug card** and may use it at any time against any player.



When a player takes the **last goal card** from the deck, the game is over at the end of the round. Then, the player that solved the highest number of goal cards **wins** the game.

EXTRAS

MOON is a **modular game** that you can **adapt** to your needs. For example, very novice players can avoid using logical operations (OR, AND, XOR) in their first games. On the other hand, expert players can play with 8 bits, just putting two MOON games together.

In addition to this general recommendation, we end this handbook by explaining two more **extra** features that can enhance your gaming experience.

EXTRA OPERATIONS

In each goal card there is an acronym that corresponds to extra operations that you can perform **once** after achieving that goal.

XCHG works as a bidirectional MOV, it exchanges the values of two registers.

SHL and **SHR** works similarly to ROL and ROR respectively, but the bit that is left out always becomes a zero (example: SHL of 1011 = 0110).

ADD and **SUB** allow you to arithmetically add and subtract two records (examples: 0101 ADD 1010 = 1111 and 1101 ADD 1000 = (1)0101, the first 1 is discarded if we use 4-bit registers).

NOR, **NAND**, **XNOR** are the combination of OR, AND or XOR and then a NOT (for example, the result of a NOR operation will be 1 only if the two entries are 0).

HACKERS

Decide which hacker you want to be, with **different special skills**:

- **Addition hacker** (blue): you can use INC or DEC by consuming only one power unit.
- **Copy hacker** (purple): you can make a MOV operation without consuming power in each case turn.
- **Rotations hacker** (yellow): you can use ROL or ROR consuming only 1/2 power unit.
- **Logic hacker** (red): you can do two logic operations OR, AND, XOR without consuming power on each turn.

ווסוור