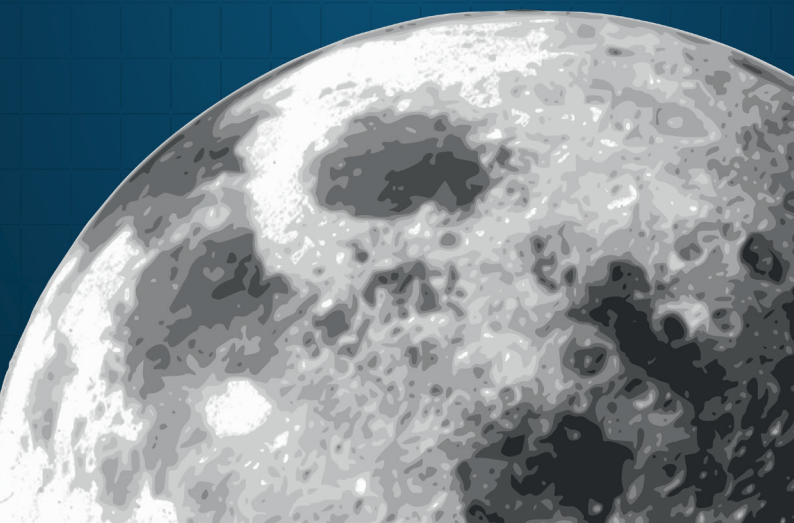
111001

ENGLISH



In 1969, millions of people around the Earth gathered around their television sets to witness an event that was happening 384,000 km away. The Eagle lunar landing was on its way to land on the **moon**, and it relied on the non-stop calculations of its on-board computer, which accomplished this with less computing power than a modern digital watch. However, three minutes before landing, the computer triggered several alarms: a radar that should have been switched off during the landing procedure unexpectedly switched on. The on-board computer, which needed to devote its scarce computing resources to landing the lunar module, could easily become overwhelmed by this extra work. Fortunately, the software that managed the on-board computer, designed by a team of engineers led by Margaret Hamilton, was smart enough to detect the problem. The computer alerted the astronauts, effectively 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 landing." Without this novel and intelligent design, we may have never taken that "small step for man and giant leap for mankind".

It took more than 30 years before NASA recognized **Margaret Hamilton's** accomplishments. Hamilton was the director of MIT's Software Engineering Division, charged with developing the on-board software for the entire Apollo program: the only piece of software that has allowed humanity to set foot on other worlds.

MOON is an educational game where players will simulate a **simple computer**.

Help the astronauts of the Eagle lunar module to fulfill their mission while you learn how to count in **binary**, perform **logical** operations, and find out how a computer works... while also having fun!

MOON is recommended for **10-year-olds** and older, for **1 to 4 players** and an estimated duration of **15-45 minutes** (depending on the selected difficulty).

SET UP A GAME

1. Place the **4 CPU registers** (A, B, C, and D) and their corresponding switched-off bits in the middle of the table. This will be the central board.

2. Place the **operation** cards on the left of the central board, sorted by their energy usage: first those requiring 2 energy units (INC, DEC), then those requiring 1 energy unit (NOT, ROL, ROR, ROL, MOV), and finally those requiring 1/2 energy units (OR, AND, XOR).

3. Shuffle the **goal cards** and place the deck face-down on the right side of the central board. These cards represent the calculations astronauts need to make to land on the moon.

4. Take 3 **energy units** and place them next to you.

MOON simulates a **real computer**. Operations modify data in the same way as in real microprocessors. So, let's review how to count in binary 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 position has an associated number (1, 2, 4 and 8 in 4-bit registers).

If all the bits of a register are **switched off**, the value **zero** is stored.

If there are any switched-on bits, you have to **add** the numbers placed on the top of the CPU. This will be the value stored in that register.

For example, this combination represents the number **3** because the bits in positions **1** and **2** are switched on, so 1 + 2 = 3.

This represents the number **9** because the bits in positions **1** and **8** are switched on, so 1 + 8 = 9.

COOPERATIVE MODE

Prepare the game as explained on page 3 of this manual. For the first game, we recommend to use 3 energy units per round and not to use OR, AND, and XOR operations. Later, you can adjust the difficulty of the game to your level.

To win the game, you must help the astronauts fulfill their mission by **solving** all the goal cards. Each goal cards shows a **combination of bits**. To solve the goal card, you must store that combination of bits in register A of the CPU.

At the beginning of the game, flip the first three goal cards and copy them in registers B, C and D. Once they are properly placed, discard these 3 goal cards.

Then, flip the top card in the stack of goal cards, and place it face-up next to the stack.

In each round, you can perform as many operations as you wish depending on the **energy** units available (remember that there are operations such as OR that require 1/2 energy units, while others like INC require 2 energy units).

You are not required to spend all energy units in one round, but you **cannot save** energy for the next round.

Use your energy units to perform **operations** on registers A, B, C and D of the CPU and achieve the goal. Remember that a goal card will not be solved until its value is stored in **register A** of the CPU.

Whether you have resolved the goal card or not, at the end of the round, you have to **move** the unsolved goal cards up one position,

take the top card in the stack of goal cards, and place it face-down next to the stack:

This is when you will recover all the energy you had at the beginning of the round.

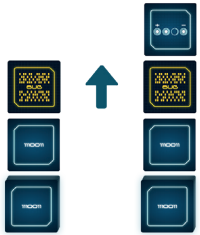
If a goal card advances to **5th position** at the end of the round, you are a slow CPU, the game is over and the lunar mission failed.

This can happen even if there are no goal cards left in the stack but it takes you more than **5 rounds** to solve the last goal cards.

In other words, at the end of the round the goal cards move up regardless the number of cards left in the stack.

On the other hand, if you manage to solve all the goals of the stack promptly, astronauts will be able to land into the moon safely and you **win**!

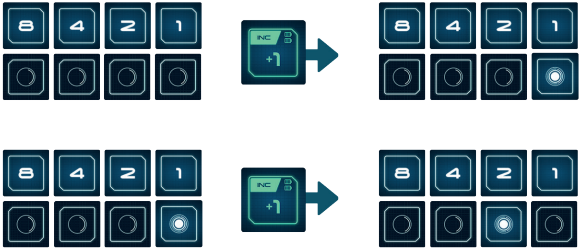
Moreover, there are goal cards that don't have a combination of bits but a **bug**.



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.

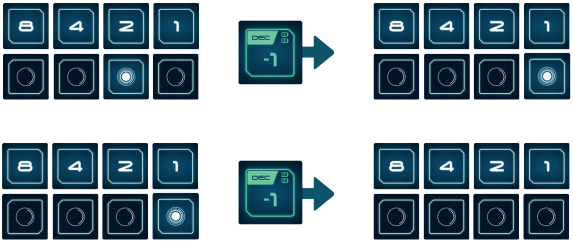
During the game, you will need to modify the bits of the CPU registers to reach a goal specified in a goal card. You will do this by using certain CPU **operations**: INC, DEC, ROL, ROR, MOV, NOT, OR, AND and XOR.

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



If the register stores the maximum value (all bits switched on), an **overflow** happens and the register is reset to zero:

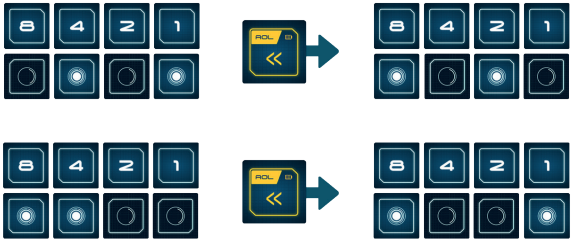
DEC This operation is used on **1 register** and costs **2 energy** units. It **subtracts** 1 to the total value stored in the register:



If all the bits of the register are switched off, subtracting 1 will cause an **underflow** that sets all the bits of the register to one (switched on):

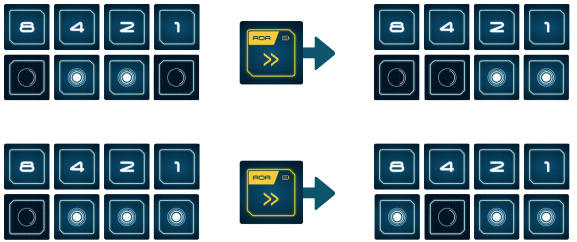
ROL This operation is used on **1 register** and costs **1 energy** unit. It involves **shifting** every bit on the register to the left

and placing the remaining bit on the **left** in the rightmost position:



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

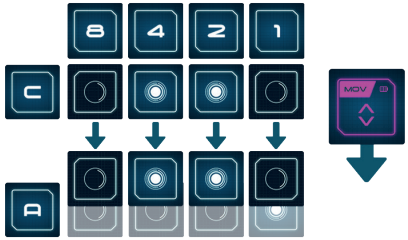
ROR This operation is used on **1 register** and costs **1 energy** unit. It involves **shifting** every bit on the register to the **right** and placing the remaining bit on the right in the leftmost position:



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

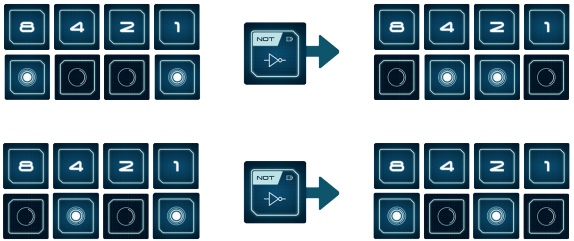
MOV This operation is used on **2 registers** or 1 register and a **RAM** module and costs **1 energy** unit (1/2 in competitive mode).

It **copies** all bits from one register to another, **overwriting** the value stored in the destination (it can be useful to copy a value into your RAM module and then recover it later to prevent other players from modifying it).



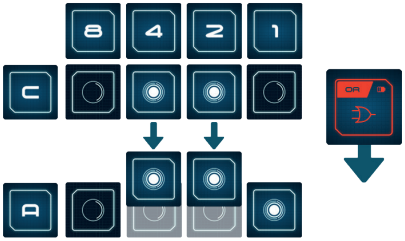
NOT This operation is used on **1 register** and costs **1 energy** unit.

It **negates** every bit on the register: switched-on bits are switched off, and switched-off bits are switched on. This involves **flipping** all the bits of the register.



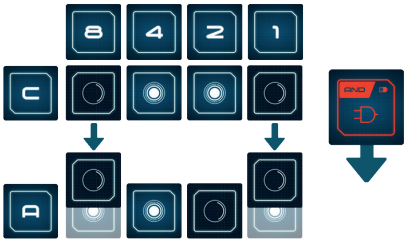
OR This operation is used on **2 registers** and costs **1/2 energy** unit.

It **copies** only the **switched-on** bits from one register to another.



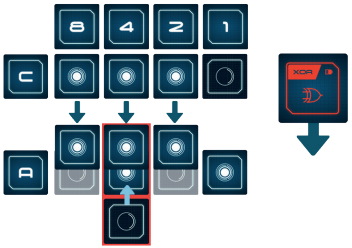
AND This operation is used on **2 registers** and costs **1/2 energy** unit.

It **copies** only the **switched-off** bits from one register to another.



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

It **copies** only the **switched-on** bits from one register to another, but if the bit was already on, it's **turned off**.



DIFFICULTY

You can adapt the difficulty of the game in several ways:

1. Changing the number of **energy** units available per round. We suggest you use 3 units of energy (easy) for the first games and then reduce it progressively (normal: 2.5; hard: 2; master: 1.5).

2. Changing the number of "**bug**" cards the goal cards deck will have (easy: none; normal: 1; hard: 2; master: 2).

3. Changing the **initial state** of the registers. For an easy difficulty, take the first 3 goal cards of the deck at the start of the game, and copy their values into registers B, C and D (these three cards can then be considered solved). For normal difficulty, do the same with the first 2 goal cards and registers B and C. For hard difficulty, the first goal card is copied to register B. For master difficulty, register A is set to value 1 and the rest of registers are set to 0.

4. Adding **event** cards to the goal cards deck.

EVENTS

Add event cards to the deck of goal cards to make the games more exciting:

RESET cards **reset** a registers (all its bits are switched off):

ERROR register cards **disable** the register:

ERROR operation cards **disable** the operation.

OK cards **repair** an existing ERROR in a register or operation (you cannot keep them around to repair future errors).

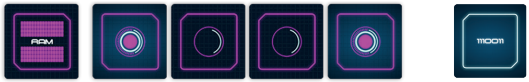
COMPETITIVE MODE

Before playing in competitive mode, it's a good idea to have played in collaborative mode first. Review the previous sections to learn the basics of the game.

Prepare the game as explained on page 3 of this manual. Each player **chooses a color**, takes the **RAM** card of that color and sets all the positions of their RAM module to zero. Distribute the **energy** units to each player according to the game's desired difficulty: 4 for easy, 3 for normal, 2.5 for difficult, and 2 for master.

Shuffle the **goal cards** and place them face-down to the right of the CPU. Each player takes a goal card from the deck, looks at it

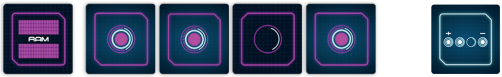
(without showing it to the other players) and places it **face-down** next to their RAM module.



In each turn, each player may play as many operation cards as allowed by their energy units. A player is not required to use all their energy units. Energy units **cannot be transferred** from one player to another.

Any player can modify the values of any of the bits in the 4 registers of the CPU in their turn, but will **not** be able to **copy or modify** the values stored in the **RAM** of other players.

If a player manages to store their goal in **register A** of the CPU during their turn, the player will **show** their goal card to the rest of the players, keep it next to their RAM module and take another goal card from the deck.



Once the deck of goal cards deck is exhausted, the player who has solved the most goal cards will be the **winner**.

In competitive mode, there are two additional **changes** with respect to the cooperative mode:

1. The MOV operation requires **1/2 energy** unit (instead of 1 energy unit).

2. Bug cards are used **to view another player's goal card** at any time during the game. When a player takes this card from the deck, they show it to the other players and saves it for later use. The bug

card can only be used during the player's turn and, when used, the player who was forced to show their goal card **will take possession** of the bug card, which they can then use during one of their turns. These cards **do not count** as a solved goal to decide the winner at the end of the game.

HACKERS

You can also make each player have **special** features in the competitive games. Look at the back of the RAM cards to see which hacker you want to be:

Green: you can use **INC** or **DEC** by consuming only 1 energy unit.

Yellow: you can use **ROL** or **ROR** consuming only 1/2 energy units.

Purple: you can make 2 **MOV** operations without consuming energy on each turn.

Red: you can do 2 logic operations **OR**, **AND**, **XOR** without consuming energy on each turn.

EXTRA BITS

MOON's modular design allows you to **expand** the number of bits in its CPU, substantially increasing the complexity of the game.

The base game comes with **8 extra bits** to extend the registers A, B, C and D to 5 bits or 6 bits.

Since goal cards only contain 4-bit combinations, we will need **two goal cards** to indicate the 5-bit or 6-bit combinations to be solved.

In **cooperative** mode, we will draw two goal cards when solving a goal:

- You solve a goal
- You show the following goal.
- At the end of the round, you draw one goal card.

The goal will be made up of all the **rightmost** bits of the two goal cards. For example, playing with registers of 6 bits, the goal will comprise the 4 bits of the card on the right and the 2 rightmost bits of the card on the left:

In **competitive** mode, you will put each part of the goal to **each side** of your RAM module to avoid mixing them.

This game was created within the framework of the COMPUS project co-funded by the Erasmus+ Programme of the European Union and developed by the University of Deusto (Spain), AGR Priority (Spain), Fundación Educativa ACI - Esclavas SC-Fatima (Spain), Şcoala Gimnazială Ferdinand I (Romania) and OEIIZK: Osrodek Edukacji Informatycznej i Zastosowan Komputerow w Warszawie (Poland). To learn more about the project, visit http://compus.deusto.es.

English translation: Borja Sotomayor.