Introduction to Computer Architecture Project

LEIC, IST

Beyond Mars: passing through the asteroid belt

2022/2023

1 - Objectives

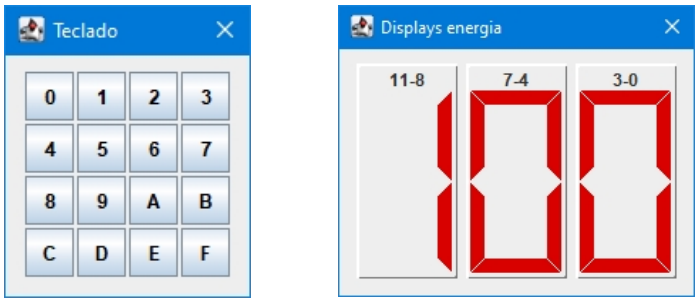
This project aims to practice the fundamentals of Computer Architecture, namely *assembly* language programming, peripherals and interrupts.

In the coming decades, humanity will continue its exploration of the solar system, taking astronauts where no one has gone before. The aim of this project is to create a simulation game of the interplanetary journey of a spaceship manned by intrepid astronauts, who venture beyond Mars and have to cross the asteroid belt, located between the orbits of Mars and Jupiter.

It's a dangerous area, given the high number of asteroids, and it's crucial to avoid a collision that would be fatal. It is possible to fire a probe with a missile to destroy an asteroid, but this uses up precious energy. However, some asteroids contain valuable metals and compounds that can be mined by a robot sent by the spacecraft to increase its energy, which is so necessary for ion thrusters.

The game interface consists of a screen, a keyboard for control/navigation and a set of displays to show the ship's energy. The image shows a possible game start screen.

The simulator's MediaCenter module has a variety of multimedia capabilities, allowing you to set background images, play sounds and videos, multiple image planes built pixel by pixel, a front backdrop for signs, etc. Lab guide 4 provides more details on this module.



The keyboard allows you to click on some of the keys to control the game (*start*, *pause* and *stop,* as well as controlling the launch of probes that either explode asteroids or mine them to send energy to the ship).

The displays show the spacecraft's current energy, which varies over time. It gradually decreases simply because the spacecraft is moving (ion propulsion uses a lot of energy) and with each probe shot. It increases when a probe encounters a mineable asteroid and its robot produces energy from the asteroid and sends it to the spacecraft.

Each game state (starting, playing, paused, finished, etc.) should have a different background scene or video, or a sign (front scene, an image with letters and a transparent background), in order to give a visual indication of the game state. The figure on the previous page illustrates a possible entry scenario, where the user has to press the C key to start the game. It could also be a video, with the letters superimposed over a front scene (with a transparent background).

Alongside this statement is a short video (**demo.mp4**) illustrating the main situations in the game.

2 - General description of the game

The game generally consists of the following ideas:

- The screen during the game represents the front window of the spacecraft (seen by the astronauts), with an instrument panel in the middle. There are only 3 possible commands (by clicking on 3 different keys on the keyboard): launch a probe to the left (at 45°), forward or to the right (at 45°);
- The probes move in a straight line until they collide with an asteroid or move out of range of the spacecraft (maximum 12 movements, one pixel at a time), at which point they disappear. The previous image shows two probes (purple pixels) launched diagonally, while the probe launched straight ahead has hit the asteroid that can be seen with the explosion effect;
- It is possible to launch probes in all three directions simultaneously, but in a given direction it is only possible to launch again after the previous one has disappeared (by collision or by going out of range);
- Asteroids appear at the top of the screen and can be of two types: mineable (green, in the example) or unmineable (red, in the example). The type must be random, in a ratio of 1 minable to 3 non-minable. In the event of a probe colliding with an asteroid, there is an explosion (visual and sound effect) if it is unmineable and a different effect if it is mineable (in the example, the asteroid reduces in size until it disappears as it is mined/consumed, and there is a "nice work!" sound);
- If an asteroid (of any kind) is not intercepted by a probe and collides with the ship (with the instrument panel), the ship is destroyed and the game is over;
- At any given time, there should be 4 asteroids on the screen. When an asteroid disappears (by explosion, mining or being lost in the background), another one should appear at the top of the screen, with the following characteristics chosen at random:

  -Type : mineable (25% of the time) or unmineable (75%);

  -Column : at the top left of the screen, in the middle or at the top right;

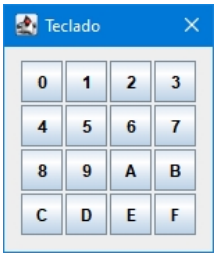
  -Direction : those coming out of the corners move diagonally (at 45°), towards the inside of the screen. Those coming out of the middle can descend vertically or diagonally, to the left or to the right (the latter cannot collide with the ship, but there is still an interest in launching a probe to pick up the minables to obtain energy);

  -The 5 possible column/direction combinations must be equiprobable;
- The ship uses energy simply by working (at a constant rate) and every time it launches a probe. It gains energy for each probe that reaches a mineable asteroid. There is an initial energy (100%), which can go up or down. If the energy reaches 0, the ship stops working and the game ends;
- The aim of the game is to keep it going as long as possible, avoiding the destruction of the ship (collision with an asteroid) and the total depletion of energy;
- There are also commands (keys on the keyboard) for controlling the game: *start*, *pause* and *stop*. All of them should produce sound effects and show a different image on the screen.

3 - Project details

3.1 - Keyboard and game control

The game is controlled by the user using keys on a keyboard (operated by mouse click), like the one in the following picture:



The assignment of keys to commands is free and up to you. One possible assignment is as follows:

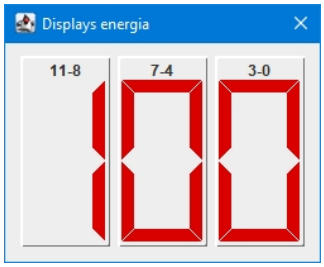
- Keys 0, 1 and 2: launch a probe to the left, forward and right respectively;
- C key: start the game (must reset the ship's power to 100%);
- D key: suspend/continue the game;
- E key: end the game (you must keep the ship's final energy visible).

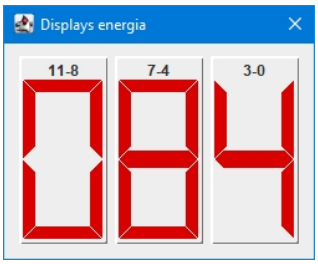
IMPORTANT - Each key should only execute one command. For a new command, even the same one, you must release the key and press it again.

Lab script 3 teaches you how to work the keyboard.

3.2 - Ship displays and power

There are three 7-segment displays, which should show the ship's energy as a percentage of the initial value, at each instant (in decimal, which implies conversion from the hexadecimal values that PEPE uses). The following figures illustrate the initial energy and a possible value after the game evolves:





Energy starts at 100 (%) and must be decremented by 3% periodically (every 3 seconds).

For each probe launched, energy decreases by 5%. For every mineable asteroid reached by a probe, the energy increases by 25%, as a result of absorbing the asteroid's energy.
Depending on the luck and skill of the ship's pilot, the energy can rise well above 100%. If the energy reaches 0%, the game ends. The image of the game ending due to lack of energy should be different from that resulting from the collision of an asteroid with the ship. Ditto for the sound effects.

If the game is ended by this collision, the energy you had at the time must remain. The energy should only be restored to 100% when a new game is started.

Lab script 3 teaches you how to work with the displays.

3.3 - Screen, scenery, sounds and puppets

The screen for interacting with the player has 32 x 64 pixels (rows x columns). Each pixel can have a different color, in 4 components (Alpha, Red, Green and Blue, or ARGB), all with 4 bits (unsigned values, from 0 to F). The first component defines opacity (0 is totally transparent, F totally opaque). The pixel can either be on (with the color defined by the 4 components) or off (with everything set to zero, in which case you can't see it because it's transparent).

The ship's instrument panel, asteroids and probes are dolls drawn pixel by pixel.

Behind this screen of pixels is the background image or video (the one you see in the previous images), which typically has a much higher resolution (although it should have a similar 2 x 1 rectangular form factor so that it doesn't appear distorted).

You can change the background image/video through the PEPE program. You are therefore expected to use a different background for each situation. An image doesn't always have to change. You can edit it by adding text that indicates the situation (pause, for example), thus generating variants of the same image. Or you can use a front scene, with an image with the letters and a transparent background, which appears in front of the background image or video. There are also commands that the program can give you to turn this backdrop on and off.

No images or videos are provided for backdrops, but there are plenty of suitable images that you can obtain for free on the web for this personal use. The multimedia design is up to you. The following figure illustrates a possible scenario after an asteroid collides with the spacecraft.

There should also be sound effects, which again can be obtained from the web. Small sound files are ideal. The MediaCenter module does, however, allow you to restrict the start and end times of a sound (or a video), should the need arise.

These sound effects must be played when a probe is launched, an asteroid is destroyed, the energy of an asteroid is mined by a probe, the game ends due to lack of energy or a collision, etc. This is a game!

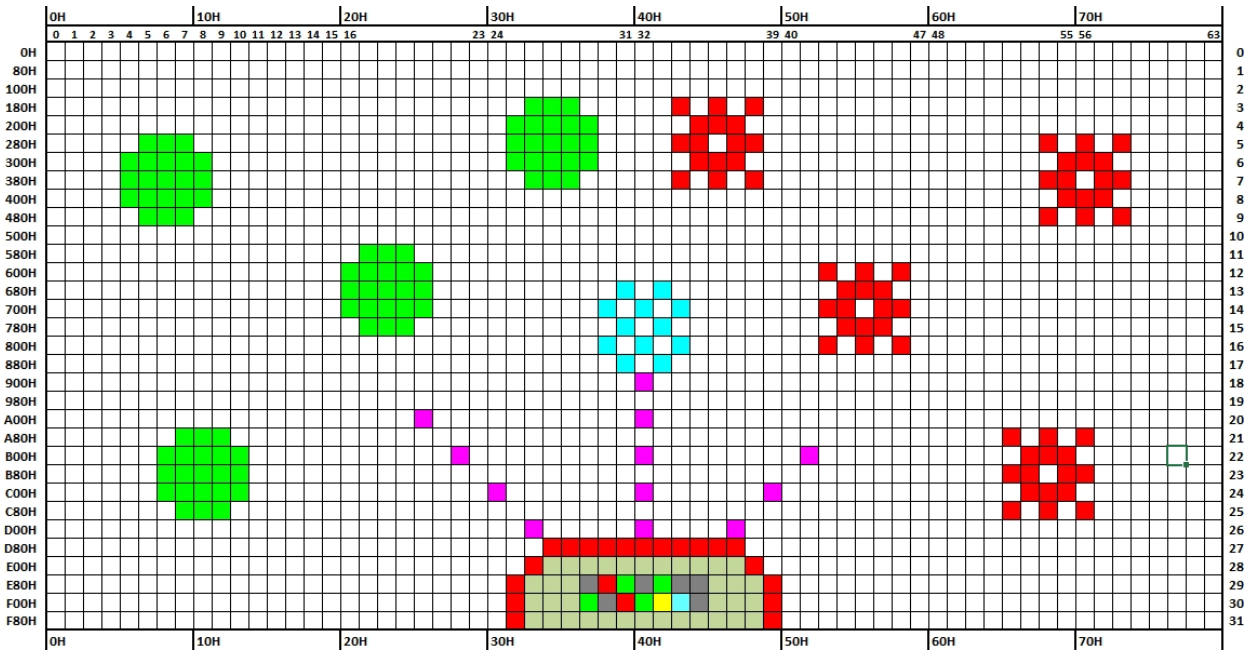
FUNDAMENTAL - Once you have defined the image, video and sound files you want to add to the MediaCenter module, you must save the circuit in a directory containing these files (either directly or in a subdirectory of that directory). This is the only way for the files to be included in the circuit description file in a portable way. Otherwise, the absolute *path of* the file is saved, and then it won't work on another computer (namely, that of the teacher who will be assessing the project!).

Drawing a doll on the screen (in front of the backdrop) means drawing its pixels, with color or turned off (transparent), in adjacent positions (row and column) on the screen, according to the shape defined for that doll.

One of the doll's pixels (top left corner, for example) is taken as an indicator of that doll's position on the screen and all the other pixels of that doll are drawn in positions relative to that reference pixel.

To move a doll is to erase it in its current position, change its position and draw it in the new position. The visual effect is that the doll appears to move.

The Excel file **screen-32x64.xlsx**, which is provided, simulates the squares of the screen and shows that the screen is actually a memory (note the addresses of each line on the left-hand side), where each pixel is a word (2 bytes) of that memory, with the color of the pixel. There are 64 pixels, or 128 bytes, for each line of the screen.



You can use this file to plan the size and shape of your dolls, as they don't have to have the shapes and sizes shown in the picture above, nor the same colors (and each doll can have different colored pixels). The picture is just an illustrative suggestion. You can invent here.

Each pixel can be drawn by writing directly to the screen memory or by using a command. Choose your method. Lab guide 4 contains details on how to use the MediaCenter module.

You want 4 asteroids simultaneously, but you can start with just one in an initial version of the program. They can appear completely visible at the top of the screen, but those that reach the bottom of the screen should disappear gracefully (line by line, until the last one disappears).

The probes (1 x 1 dollies) come out of the ship's instrument panel at the top (in the middle and in the right and left corners) and should have a limited range (12 moves maximum is suggested), at the end of which they are extinguished if they hit nothing. You can launch 3 probes almost at the same time, but in different directions. You can only launch a new probe in a given direction after the previous one in that direction has disappeared (due to range limit or collision).

The ship's instrument panel has a set of rapidly changing lights. This is purely cosmetic (it always looks more SciFi...) and can be implemented with a set of dolls with the same shape but different colors, where you (re)draw each one in the same position, rotating periodically. The example in the **demo.mp4** video uses 8 puppets.

The MediaCenter module makes it possible for each doll to be drawn on a different pixel screen (with all the pixel screens overlapping). This has the advantage that asteroids following the same path overlap gracefully (as if they were different windows in an operating system), instead of one destroying the drawing of the previous one.

Since the ship and the probes never overlap, they can use the same pixel screen. You will have to define the number of pixel screens you use in MediaCenter, in Design mode in your configuration panel.

Lab script 4 teaches you how to work with the MediaCenter module, in terms of pixels (and drawing puppets), pixel screens, scenery and sounds.

3.4 - Timings

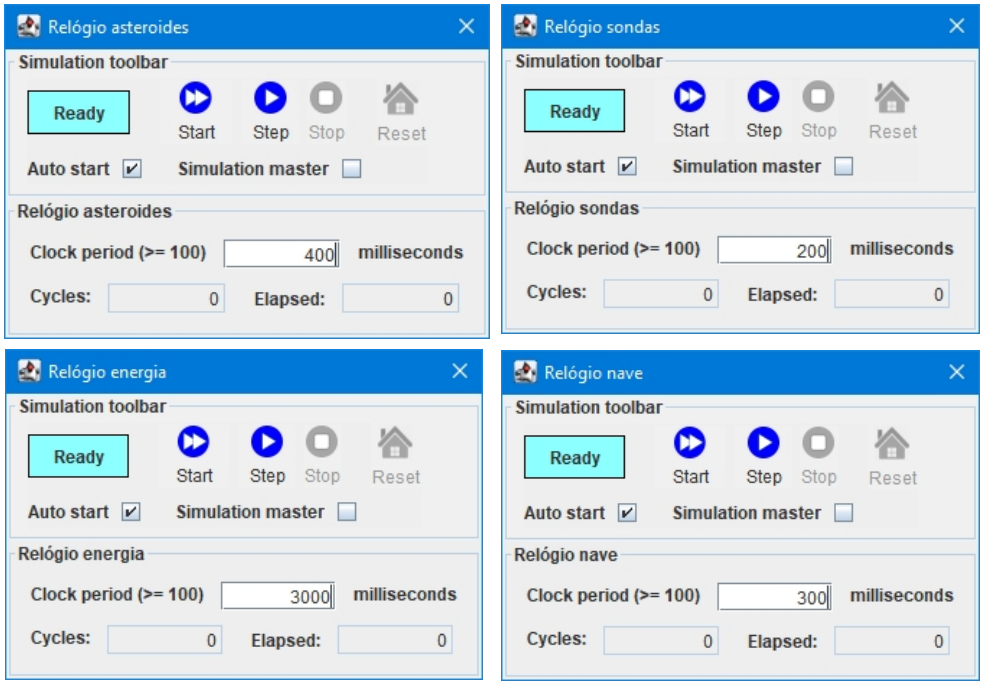
The evolution of the game requires 4 different timings (which can vary to your liking):

- Movement of asteroids (period of 400 milliseconds);
- Probe movement (200 millisecond period);
- Periodic decrease in the ship's energy (a period of 3000 milliseconds, or 3 seconds);
- Variation in the configuration of the lights on the ship's instrument panel (300 milliseconds).

The periods indicated, which mark the rate at which each of the events occurs, are generated by 4 real-time clocks, which generate a one-bit signal that varies periodically between 0 and 1, with a given period. Without the real time marked by these clocks, the game would evolve much more quickly.

quickly and in an uncontrollable way, depending only on the processing speed of the computer running the simulator.

These clocks are included in the circuit used in this game and are pre-programmed with these times, but you can change them to improve the gameplay or run tests.



The clocks start automatically, so you don't even need to open their simulation windows. Lab guide

6 shows you how to use clocks to set timings.

3.5 - Pseudo-random choices

When an asteroid is "born" (it appears at the top of the screen), you must decide whether it will be minable or not, taking its color and shape, which column it should appear in, and which direction it should take.

Around 25% of asteroids are expected to be mineable and 75% unmineable.

For the column, at the top left of the screen, in the middle or at the top right. Note that, given the size of the asteroids, the reference pixel column for each asteroid may not be right in the middle or in the corner of the screen.

For direction, those coming from the corners move diagonally (at 45°), towards the inside of the screen. Those coming out of the middle can move vertically or diagonally, to the left or to the right. The 5 possible column/direction combinations must be equiprobable (20% chance each).

These choices must be made reasonably randomly. As PEPE has no mechanism for generating random values, a simple trick is used. Reading an input peripheral (PIN, in the project circuit) generates random values in the bits that are "in the air", i.e. not connected to something that

force a value. This is the case with bits 7 to 4 of the PIN, where bits 3 to 0 link to the keyboard but nothing links to bits 7 to 4. A read is made from the peripheral (with MOVB, 8 bits are read), followed by a right shift (SHR instruction) of 4 bits, which places bits 7 to 4 (random) of the peripheral in bits 3 to 0 of the register, thus leaving you with a random value between 0 and 15. From here, you can obtain:

- the type of asteroid at random (isolating the 2 bits with the lowest weight, which gives 4 hypotheses, one of which is used to generate mineable asteroids);
- the column-direction pair (by doing the rest of the division by 5 with the MOD instruction, for example). Each of the 5 different combinations can then index a table in which each element has two values (initial column and direction: -1, 0 or 1, i.e. the value to be added to the asteroid's column each time it moves).

4 - Project phasing

The project will take place in two phases, an intermediate and a final version.

IMPORTANT - Don't forget to identify the code files (groupXX.asm, where XX is the group number) in comments, right at the start of the program, with:

- the group number;
- the number and names of the students who participated in the development of the program.

INTERMEDIATE VERSION:

- Worth 25% of the project grade (or 10% of the final IAC grade);
- It must be submitted to Fenix (intermediate version project) by **11:59 p.m.** on **May 26**, **2023,** via a file (**groupXX.zip,** where XX is the group number) with the following files:
  - A **groupXX.asm** file with the code, ready to be loaded into the simulator and executed (it should have the group number, student numbers and names). You should create a copy of the latest version of the code, cleaning up any "junk" and temporary things, in order to compile and run the requested functionality. Code organization and comments will be evaluated, as in the final version;
  - All image and sound files used in the "MediaCenter" module;
  - A **projectXX.cir** file with the project circuit, but saved after defining all the image and sound files used in the "MediaCenter" module. Don't forget that these files must be saved in the same directory as the circuit, or in a subdirectory of it;
- **IMPORTANT** - Use the circuit in the project, **project.cir** (and not the one in any lab script). Note that the peripheral of the displays is 16 bits (you must use MOV) and not 8 bits (MOVB);

- **It must meet the following objectives** (in the intermediate version):
  - The keyboard must be fully functional, detecting all the keys;
  - You must draw the ship's instrument panel;
  - There must be a backdrop;
  - You have to draw an asteroid (mineable or not) at the top left of the screen. This asteroid must move diagonally down one line on the screen each time you press a key (you choose which one), but only one line for each key click. You don't have to detect the possible collision with the ship;
  - It should have a sound effect every time you press the button for the asteroid to descend;
  - You must draw a probe, initially in the middle column of the line above the ship's instrument panel. This probe should rise vertically one line on the screen each time a key is pressed (you choose which one), but only one line for each key click;
  - Use two other keys of your choice to increase and decrease the value on the displays by one for each key click. For now it can be in hexadecimal, but in the final version you'll have to make a routine to convert any hexadecimal number to decimal digits.

FINAL VERSION:

- Worth 75% of the project grade (or 30% of the final grade);
- **It must meet all the specifications in the statement**;
- It must be submitted on Fenix (Final version project) by **11:59 p.m.** on **June 9, 2023,** via a file (**groupXX.zip,** where XX is the group number) with the following files:
  - A **groupXX.pdf** file, a very simple report in free format, but with the following information (a possible report template is provided along with this statement):
    -Identification of the group number, student numbers and names;
    -Relevant definitions, if they have done something different from what the statement asks for or indicates (different keys, extra functionality, etc.);
    -Concrete indication of the requested functionalities that the code sent does NOT satisfy;
    -Any other comments or conclusions.
  - A **groupXX.asm** file with the code, ready to be loaded into the simulator and run (it should also have the group number, student numbers and names);
  - All image, video and sound files used in the "MediaCenter" module;

- A **projectXX.cir** file with the project circuit, but saved after defining all the image, video and sound files used in the "MediaCenter" module. Don't forget that these files must be saved in the same directory as the circuit, or in a subdirectory of it.

5 - Implementation strategy

The lab scripts are aligned with partial objectives to be achieved in terms of project development. Try to meet them to ensure that the project is completed by the due dates, both in the intermediate and final versions.

Cooperative routines or cooperative processes (lab guide 7) should be used to support the various, apparently simultaneous, actions in the game. The following processes are recommended:

- Control (to handle the keys for starting, suspending/continuing and ending the game).
- Keyboard (scanning and reading the keys, as described in the script for lab 3);
- Ship (to draw the instrument panel and produce the effect of the lights);
- Ship energy (to implement periodic energy expenditure);
- Probe (to control the launch, implement the movement, range limit and collision detection of each probe);
- Asteroid (to control the actions and evolution of each asteroid, including collision checks with the ship).

As a general order of project implementation, the following strategy is recommended (you can of course adopt another one):

1. Keyboard and displays;
2. Screen routines for drawing/deleting:
   - a pixel in a given row and column (from 0 to 31 and 0 to 63, respectively);
   - a generic doll, described by a table that includes its width, height and the ARGB color of each of its pixels. Use one of these pixels, for example the top left corner of the doll, as a reference for the doll's position (row and column) and draw the remaining pixels in relation to the coordinates of this reference pixel;
3. Drawing of the ship's instrument panel;
4. A single asteroid, moved first by a button and later by an interruption;
5. A single probe, operated first by a key and later by an interrupt;
6. Periodic expenditure of the ship's energy through an interruption;
7. Processes, so that you can control the various aspects of the game independently;
8. Collision detection (between the probe and an asteroid and between an asteroid and the spacecraft);

9. The rest of the specifications, including an extension for the 4 asteroids and 3 probes. This extension has some complications, so it's better to have just one asteroid and one probe working than to try everything and run the risk of nothing working.

IMPORTANT:

- Interrupt routines stop the main program while they are running. Therefore, they should only signal the processes when they occur, by means of variables (LOCKs, in the case of using cooperative processes). The game processing should be done by the processes and not by the interrupt routines;
- If you use negative values (for example, -1 to add to a doll's column so that it moves to the left), the corresponding variables must be 16 bits (declared with WORD, not BYTE).

Also take into account the following recommendations:

- PUSH and POP all registers that you use in a routine and that are not output values (but not systematically all registers, from R0 to R11!). It's very easy not to notice that a given register is changed during a CALL, causing errors that can be difficult to debug. Pay attention to the pairing of PUSHs and POPs, as well as their relative order;
- Test all the routines you make and when you change them. It's much harder to discover a bug in a program that's already complex and untested;
- Structure the program well, with a data zone at the start, both for constants and variables, and auxiliary routines for implementing each process alongside it;
- Produce plenty of comments, not forgetting headings for the routines with description, input and output records (see examples in the lab guides);
- Don't put numeric constants (with a few exceptions, such as 0 or 1) in the middle of the code. Define symbolic constants at the beginning and use them later in the program;
- As a good practice, variables in memory should be 16 bits (WORD), so that they can support negative values without problems. PEPE can only do 2's complement arithmetic with 16 bits;
- 8-bit peripherals and BYTE tables must be accessed with the MOVB instruction. Variables defined with WORD (which are 16 bits) and 16-bit peripherals must be accessed with MOV;
- **ATTENTION!!!** Unlike the lab scripts, the POUT-1 peripheral is 16-bit (because of the 3 displays) and not 8-bit (it must be accessed with MOV, not MOVB);
- Don't duplicate code (with *copy-paste*). Use a routine with parameters to cover the various cases in which the corresponding behavior is used.

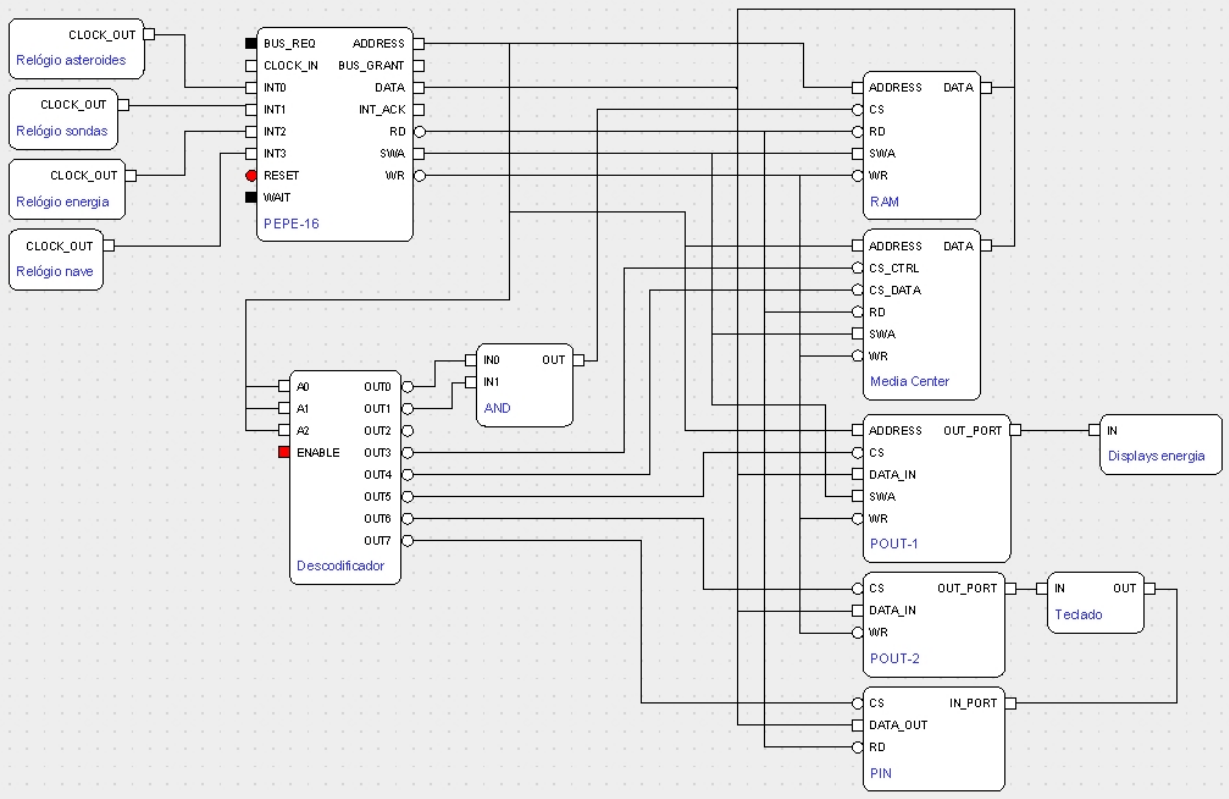
6 - Evaluation criteria

The assessment criteria and their relative weight in the project's final grade (expressed in values) are shown in the table below:

Criteria	Intermediate version	Final version
Basic functionality	1	3
Data and code structure	3	6
Comments	1	2
Various asteroids and probes: data and code		4
Total	**5**	**15**

7 - Project circuit

The following figure shows the circuit to be used (supplied, **project.cir** file). Use this circuit for the project (the ones in the scripts are not suitable).



The following modules have a running control panel ("Simulation" mode):

- Asteroid clock - Real-time clock, to be used as a basis for timing the movement of asteroids. It is connected to PEPE interrupt pin 0;
- Probe clock - Real-time clock, to be used as the basis for timing the movement of the probes. It is connected to PEPE interrupt pin 1;
- Energy clock - Real-time clock, to be used as the basis for timing the periodic decrease in the ship's energy. It is connected to PEPE interrupt pin 2;
- Ship clock - Real-time clock, to be used as the basis for timing the set of lights on the ship's instrument panel. It is connected to interrupt pin 3 of the PEPE;
- MediaCenter - multimedia module that includes a 32 x 64 pixel screen. This screen is accessed by commands or as if it were a 2048 pixel memory (or 4096 bytes: 128 bytes on each line, 32 lines). This peripheral has 2 *chip selects*, one for memory access and the other for command access. You can see the addresses of each byte (relative to the base address of the screen) in the excel file screen-32x64**.xlsx.** Lab guide 4 provides more details;
- Three 7-segment displays, connected to bits 11-8, 7-4 and 3-0 of the POUT-1 peripheral, to show the ship's power. **ATTENTION!!!**: unlike the laboratory scripts, this peripheral is 16 bits and must be accessed with MOV (not MOVB);
- Keyboard, 4 x 4 keys, with 4 bits connected to the POUT-2 peripheral and 4 bits connected to the PIN peripheral (bits 3-0). Detection of which key has been pressed is done by scanning. Note that these peripherals are 8-bit and must be accessed with MOVB (not MOV). Also note that only the 4 smallest bits (3 to 0) are significant. The rest (7 to 4) are in the air and read random values. You should therefore use a mask to eliminate them when trying to detect key presses;
- Memory (RAM), which has 16 data bits and 14 address bits, with byte addressing capability, such as PEPE and MediaCenter;
- PEPE-16 (16-bit processor).

The address map (where the devices can be accessed by PEPE) is as follows:

Device	Addresses
RAM	0000H to 3FFFH
MediaCenter (access to commands)	6000H to 6069H (see lab guide 4)
MediaCenter (access to your memory)	8000H to 8FFFH
POUT-1 (16-bit output peripheral)	0A000H to 0A001H
POUT-2 (8-bit output peripheral)	0C000H
PIN (8-bit input peripheral)	0E000H