

SCALE FOR PROJECT COREWAR (/PROJECTS/COREWAR)

Introduction

Please respect the following rules:

- Remain polite, courteous, respectful and constructive throughout the correction process. The well-being of the community depends on it.
- Identify with the person (or the group) graded the eventual dysfunctions of the work. Take the time to discuss and debate the problems you have identified.
- You must consider that there might be some difference in how your peers might have understood the project's instructions and the scope of its functionalities. Always keep an open mind and grade him/her as honestly as possible. The pedagogy is valid only if the peer-evaluation is conducted seriously.


Guidelines


- Only grade the work that is in the student or group's GiT repository.
- Double-check that the GiT repository belongs to the student or the group. Ensure that the work is for the relevant project and also check that "git clone" is used in an empty folder.
- Check carefully that no malicious aliases was used to fool you and make you evaluate something other than the content of the official repository.
- To avoid any surprises, carefully check that both the correcting and the corrected students have reviewed the possible scripts used to facilitate the grading.

- If the correcting student has not completed that particular project yet, it is mandatory for this student to read the entire subject prior to starting the defence.



- Use the flags available on this scale to signal an empty repository, non-functioning program, a norm error, cheating etc. In these cases, the grading is over and the final grade is 0 (or -42 in case of cheating). However, with the exception of cheating, you are encouraged to continue to discuss your work (even if you have not finished it) in order to identify any issues that may have caused this failure and avoid repeating the same mistake in the future.


Attachments

 Subject (<https://cdn.intra.42.fr/pdf/pdf/1849/corewar.en.pdf>)

 Resources (https://cdn.intra.42.fr/pdf/pdf/1861/resources_corewar.en.pdf)

 Resources (https://cdn.intra.42.fr/pdf/pdf/4801/resources_corewar.pdf)

 op.c (/uploads/document/document/26/op.c)  op.h (/uploads/document/document/27/op.h)

 VM and champs (/uploads/document/document/599/vm_champs.tar)

Mandatory part

Reminder : Remember that for the duration of the defence, no segfault, nor other unexpected, premature, uncontrolled or unexpected termination of the program, else the final grade is 0. Use the appropriate flag. This rule is active throughout the whole defence.

Basic stuff

Check the following:

- Something was submitted
- The author file is at the root of the repository and formatted as explained in the subject.
- Norm is OK (using the norminette)
- The whole groupe is present

If at least one isn't ok defence the is over and final grade is 0.

 Yes

 No

READ ME

TL;DR

Take your time, don't be a dick, do the whole defence, you should leave both leave this grown.
#####

In life a lot of people live following more or less complicated principles. During this defence as well as during life it's actually good to follow some principles such as: "Be rigorous" "Don't be a dick" By that we mean *Rigor, as always, must be applied; * but don't be STINGY, and focus on doing a fair and honest evaluation of the work submitted by the group and don't nitpick. The aim here ISN'T to hunt for a 0. There is a lot of things to check, this defense will be longer than usual. Don't think it as time lost, and remember that peer2peer correction is part of the program to help you learn. If you have to spend hours, do it, As explained in the subject some points are subject to interpretation, so is this defence, you'll discover it.

Use common sense to decide fairness of a point, and keep the "Don't be a dick" general principle. For example a VM that uses different option names, that need an option to use a champion, or refuses to work with just 1 champion... WELL, IT DOESN'T MATTER. To put a 0 for something so TRIVIAL would be a perfect example of "being a dick". The same can be applied for displays that wouldn't be the ones shown as example in the subject or any other small thing like this.

Always use your common sense to determine what is faulty and what can be considered embellishment. Any student that does this defence in a rush without worrying about its impact on people will be punished accordingly. (c) zaz.

 Yes

 No

The VM

The basics

The VM must be able to load at least one champion with a valid header, and recognize names and comments.
Then it'll run, and finish by displaying a message with the name of the winner.

 Yes

 No

Instructions execution

- The VM must execute instructions properly. To check this you can use a dummy champion that has only one instruction (or more to "prepare the ground"), and execute it with the -dump option to check that the memory is good. It's easy to check registries using st or carry using zjmp.
- The VM must execute processes in the right order, from the last born to the first born. Typically, when running a game with two champions, the initial process of the second one must be executed first.

 Yes

 No

Option management and errors

The VM must manage the following option listed in the subject:

- dump
- n

The following errors must be managed:

- Invalid header
- No code
- Too big champion

 Yes

 No

Specific execution parameters

- IDX_MOD: Action addresses have a IDX_MOD modulo applied. It concerns the following actions:
 - The reading of the whole memory to define the indirect (For indirect "516", we read only at PC + 4 and not at PC + 516)
 - The "ld" first argument
 - The "st" second argument
 - The "zjmp" argument
 - The sum of "ldi" first two arguments

- The sum of "sti" first two arguments
- The "fork" argument

- The champion placement must be properly distributed. Their starting point must be at equal distance (modulo rounding if there is three players). For example, for two champions on a 4096 bytes memory, the starting points must be spread with 2048 bytes. (For example the champion number 1 can start at 0 and champion number 2 at 2048).

- CYCLE_TO_DIE: The VM must manage properly different CYCLE_TO_DIE:
- A process that doesn't do at least one live during the CYCLE_TO_DIE cycles must die.
- If during a CYCLE_TO_DIE period of time there is at least NBR_LIVE live executed when the check-up is operated CYCLE_TO_DIE must be decreased by CYCLE_DELTA.
- If during a check-up there has been no CYCLE_TO_DIE since MAX_CHECKS check-ups, it's decreased.
- If during a check-up all processes are dead, the game end. Remember that it's not stipulated in the subject if the check-up has to be done at the beginning or at the end of a cycle. Both interpretation are acceptable.

✓ Yes

✗ No

Bonus

A lot of stuff

A lot of possible bonuses

- an awesome lexer / parser
- a .cor file disassembler
- beautiful message when compiling, with line numbers, and a cursor to show where is the error (like CLANG)
- an interface visualizer (integrated in the VM, via pipe or shm, or file, ...)
- a debug mode on the VM (step-by-step, memory inspection process state... infinite possibilities)

Rate it from 0 (failed) through 5 (excellent)



And everything else

("especially everything else" (c))

Some more error management

The following errors must be managed either by the VM or the ASM:

- Nameless champion
- Champion with no comment
- Champion with a too long name
- Champion with no code

A lot of ways to manage those errors are acceptable like for example:

- Refusing the compilation
- Refusing to run the champion
- To correct automatically the champion
- Run the champion even if there is no code and execute nothing

What matters here is that the ways it's handled makes sense and is properly explained.

☒ Yes

☐ No

A Champion

The group must have a champion. The champion's goal is just to make sure the group understood how to write corewar's ASM not to fight anything. Check that the champion compile without errors and if you feel motivated you can check that it can actually beat zork.s

☒ Yes

☐ No

A winner

The VM is able to find a winner, meaning the champion that did a "live" last when the game ended.

☒ Yes

☐ No

Group organisation

Evaluate here how the group was organised to work through the corewar project. Here again a lot of scenarios are acceptable stay open. Don't validate this if you feel like the group is messy and didn't really show any bit of organisation, or time management. This question is purely objective and is recognized as such. Know that subjective judgment of a hierarchy is part of your professional future and even if you don't understand it yet, you can act on it.

☒ Yes

☐ No

The ASM

Let's now go through the assembly language.

The .cor file and its header

The assembler must generate at least a .cor file with a valid header. Check out using the following command:
hexdump -vC not diff because some deviation in the way the name or comment is implemented are acceptable.
For example handling a default comment or name for a champion that don't have any will make the output different but isn't considered an ERROR.

☒ Yes

☐ No

Gestion des erreurs

The following errors must be handled in the ASM:

- Unknown instruction
- Wrong argument number for an instruction
- Wrong type of argument for an instruction
- Wrong character in a label
- Reference to an unexisting label from a direct or an indirect
- In a general manner every lexical error like a wrong character to finish a label, a direct, ...

☒ Yes

☐ No

The generated bytecode

The assembler must generate a valid bytecode for the champion's code. No deviation acceptable here, if there is one the champion generated isn't good and will not be able to work in the VM. It's not possible to be nice here, since an incorrect aspect will lead to the .cor file being executable only in the VM of the group (which in this case ends up with a bad management of that aspect).

- Instruction opcodes are the right ones. It's necessary.
- Argument's coding byte, when required, are there and right. They're not when not required.
- Registries are encoded on 1 byte and their values match up their number
- Indirects are encoded on 2 bytes and their values are correct (in big endian)
- Direct are encoded on 4 bytes for instructions that don't require an index and their values are correct (in big endian)
- Directs are encoded on 2 bytes for instruction that require an index, and their value are correct (in big endian)
- References to labels are converted to corresponding values (a label's address equals the address of the first byte following it and a reference type %:labelX must be converted to (address of the beginning of the current line) - (address of the beginning of the line concerned by the labelX))

Regarding the last 6 points (first one is MANDATORY) a 1 wrong point tolerance will be granted to validate this section.

✓ Yes

✗ No

Ratings

Don't forget to check the flag corresponding to the defense

✓ Ok

★ Outstanding project

📄 Empty work

📄 Incomplete work

💬 No author file

🧑‍🔬 Invalid compilation

📄 Norme

💻 Cheat

💥 Crash

👤 Incomplete group

🚫 Forbidden function

Conclusion

Leave a comment on this evaluation

Preview!!!

General term of use of the site
(<https://signin.intra.42.fr/legal/terms/6>)

Privacy policy
(<https://signin.intra.42.fr/legal/terms/5>)

Legal notices
(<https://signin.intra.42.fr/legal/terms/3>)

Declaration on the use of cookies
(<https://signin.intra.42.fr/legal/terms/2>)

Terms of use for video surveillance
(<https://signin.intra.42.fr/legal/terms/1>)

Feedback
(<https://signin.intra.42.fr/legal/terms/7>)