QtRVSim Web Evaluation

Jakub Pelc

10. 10. 2024

Faculty of Electrical Engineering, Czech Technical University in Prague

Goals of this project

Bonus task evaluation

The main aim for this project is to allow external participants (as well as students) to improve their skills in computer architectures, by solving tasks in RISC-V assembly.

The original way involved students of `B35APO` solving a set of `bonus tasks`, which were available to submit through GitLab and subsequently evaluated using `QtRvSim`.

This is unfortunately not available to the general public, and that is the reason why this project was created.

Current state

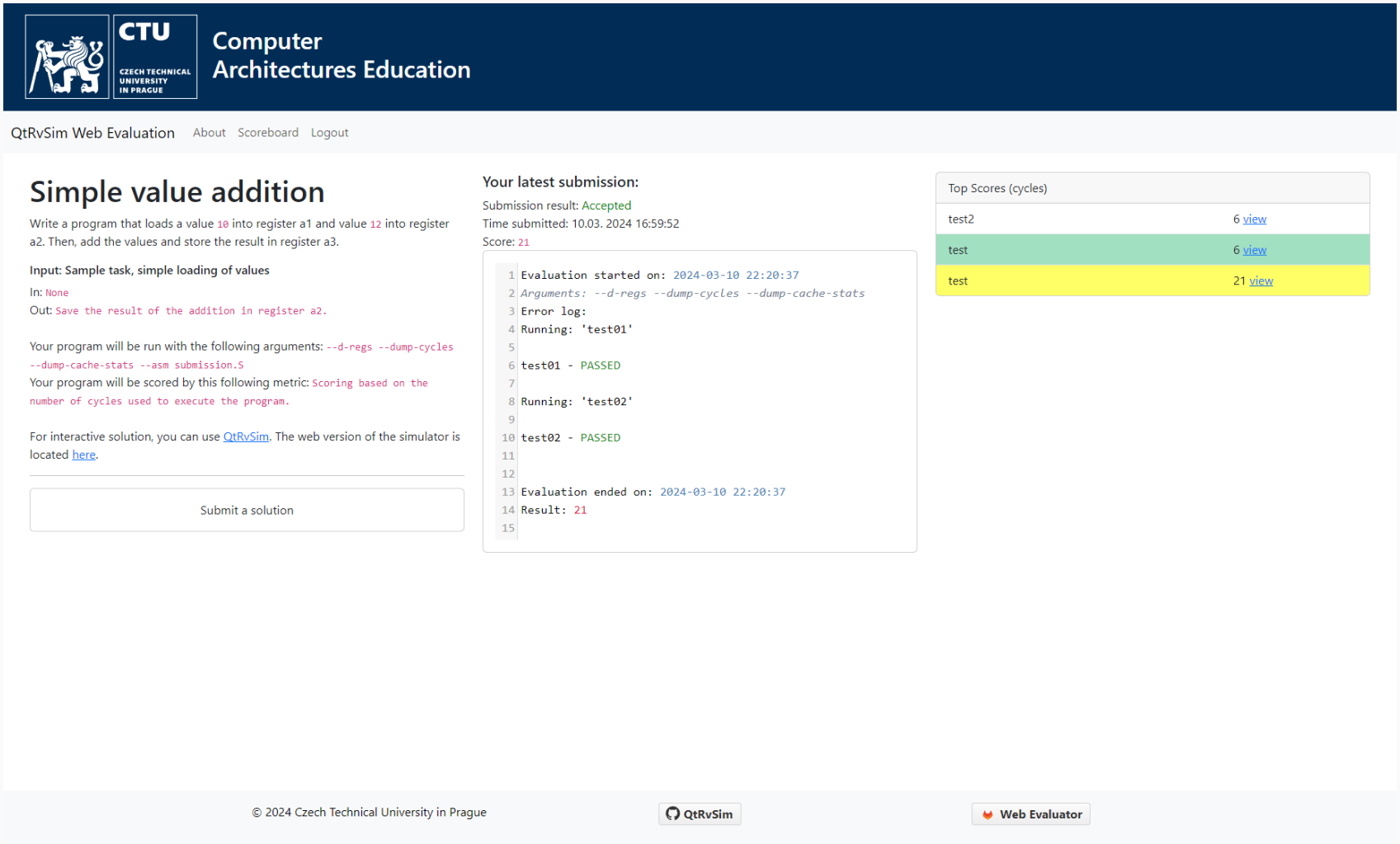
Registered users can register and submit solutions to the problems displayed on the frontpage and get immediate feedback on their solution. Local scoreboard is displayed for each task.

This is done by running a local evaluation procedure (with the use of `QtRvSim CLI`), which evaluates the correctness of the code submitted and yields the performance as a score.

The project needed to be rather simple, for it to allow easy modularity and optional modification in the future. It can be expanded by more features, language support, or task types.

This is why Flask was chosen as the web framework.

User interface



`eval.comparch.edu.cvut.cz`

Database

Communication with the database

In the web application, a PostgreSQL database is used.

Only a few tables are needed to store the information about the users, tasks, submissions and results.

PostgreSQL triggers are used to automatically update the best score and source code.

Users Table

Field	Type	Length	Default
id	int	32	AUTO_INCREMENT
username	varchar	128	None
password	varchar	128	None
email	varchar	128	None
salt	varchar	128	None
verification_code	varchar	128	None
user_verified	tinyint	1	0

GDPR

Email adresses of the users are not being saved (due to GDPR), but during the registration process, the users are required to provide an email address for verification purposes. So how is that achieved?

The email address is saved as a salted `SHA-256` hash. This way, the email address can we verified, but cannot be reverse engineered to obtain the original email address.

This also allows for password reset functionality, without the need to store the email address in a readable format. Users always need to provide the email address, which is then checked against the hash in the database.

Submissions Table

Field	Type	Length	Default
id	int	64	AUTO_INCREMENT
userid	int	64	None
taskid	int	64	None
file	text	64	None
evaluated	tinyint	1	0
time	datetime	None	current_timestamp()

Results Table

Field	Type	Default
userid	bigint	PRIMARY
taskid	bigint	PRIMARY
result_file	text	NULL
last_source	text	NULL
best_source	text	NULL
score_last	integer	-1
score_best	integer	-1
time	timestamp with time zone	CURRENT_TIMESTAMP
result	smallint	-1

```python
import psycopg2
import os

db_config = {
        'user': os.getenv('DB_USER'),
        'password': os.getenv('DB_PASSWORD'),
        'host': os.getenv('DB_HOST'),
        'database': os.getenv('DB_DATABASE'),
        'port': os.getenv('DB_PORT'),
        'sslmode': 'require',
        'connect_timeout': 10
}
```

```
def connect():
        db = psycopg2.connect(**db_config)
        cursor = db.cursor()
        return (db, cursor)

def get_user(username):
  (db, cursor) = connect()
  cursor.execute('SELECT password FROM \
        users WHERE username = %s', (username,))
  user = cursor.fetchone()
  cursor.close()
  db.close()
  return user
```

Evaluation using QtRvSim

Submission evaluation

Each of the submissions is being evaluated by a `qtrvsim_cli` python wrapper `qtrvsim.py`.

For each task, a `.toml` file defines its structure, this file is then parsed using an `evaluator.py` script. A new QtRvSim instance is initialized with needed parameters, the instance evaluates all the testcases declared in the task file and measures the performance of the user's submission.

The result, score, and the log are then displayed to the user.

```
from qtrvsim import QtRVSim

sim = QtRVSim(args="--d-regs --dump-cycles --cycle-limit 1000", submission_file="file.S")

ending_regs = {
        "a1": 2,
        "a2": 4,
        "a3": 6,
}

starting_mem = {
        "array_start": [2, 4],
}

ending_mem = {
        "array_start": [2, 4, 6],
}

sim.set_reference_ending_regs(ending_regs)

sim.set_starting_memory(starting_mem)

sim.set_reference_ending_memory(ending_mem)

#sim.set_private() #optional, if set to true, does not show errors

sim.run("Testcase 1")

print(sim.get_log())
print(sim.get_scores()["cycles"] if sim.get_result() == 0 else "-1")

sim.reset()
```

```toml
[task]
name = "Task"
template = "S_templates/template.S"

description = '''
# Description
The task description
'''

[arguments]
run = "--d-regs --dump-cycles --cycle-limit 1000"

[[testcases]]
name = "Testcase 1"
private = true

[[testcases.reference_regs]]
a1 = 2
a2 = 4
a3 = 6

[[testcases.starting_mem]]
array_start = [2, 4]

[[testcases.reference_mem]]
array_start = [2, 4, 6]

[score]
testcase = "Testcase 1"
```

Advanced tasks

The evaluator is also able to set a cache for the task, whose parameters are configurable as a part of the task. This is done by setting the maximum cache size for the task, users are then required to configure the cache parameters.

Serial input and output can also be used.

It is also possible, to create a task in `C`, but this also requires a custom Makefile to be provided in the taskfile. If custom files need to be present at compile time, they can also be provided.

```
[task]
name = "Cache example"
template = "S_templates/cache.S"
cache_max_size = 16

[arguments]
run = "--dump-cycles --read-time 10 --cycle-limit 5000 \
       --write-time 10 --burst-time 2"
```

```
[task]
name = "C example"
template = "S_templates/example.c"
c_solution = true

[[testcases]]
name = "test1"

[[testcases.input_uart]]
uart = "111\n222\n"

[[testcases.reference_uart]]
uart = "333\n"

[score]
testcase = "test1"

[make]
Makefile="""
#provide a rule that will compile the solution into a binary `submission`
#please provide a clean rule, this is run after evaluation
clean:
        rm -f *.o *.a $(OBJECTS) $(TARGET_EXE) depend
"""

[[files]]
name = "crt0local.S"
code = """
/* minimal replacement of crt0.o which is else provided by C library */
"""
```

References

References

Links and references:

Flask

eval.comparch.edu.cvut.cz

comparch.edu.cvut.cz

QtRvSim repository

Web Eval repository

Slides with examples

Jakub Pelc