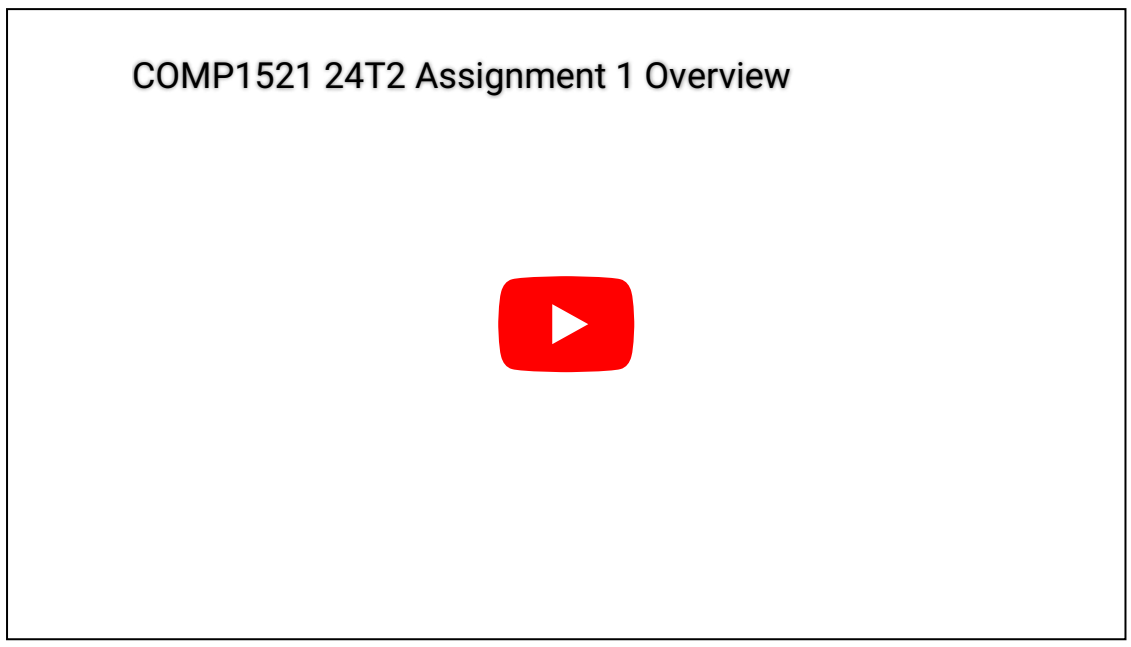
Assignment 1: Breakout in MIPS

version: 1.0 last updated: 2024-06-11 12:00:00

You may find the Assignment 1 overview video to be a good starting point:



Aims

- to give you experience writing MIPS assembly code
- to give you experience translating C to MIPS
- to give you experience with data and control structures in MIPS

Getting Started

Create a new directory for this assignment called `breakout`, change to this directory, and fetch the provided code by running these commands:

```
mkdir -m 700 breakout
cd breakout
1521 fetch breakout
```

If you're not working at CSE, you can download the provided files as a zip file or a tar file.

This will add the following files into the directory:

- `breakout.s`: a stub MIPS assembly file to complete.
- `breakout.c`: a reference implementation of Breakout in C.
- `breakout.simple.c`: a copy of the reference implementation of Breakout, for you to simplify.
- `input.txt`: example input file.
- `breakout.mk`: a *make* fragment for compiling `breakout.c`.

Breakout: The Game

`breakout.c` is an implementation of a version of *Breakout*, a popular and influential video game.

An example game of *Breakout* can be seen to the right.

A game of *Breakout* takes place on a 2D grid, where the player must move a paddle to bounce a ball (`*`) into a group of bricks (digits).

You can move the paddle left (`a` and `A`) and right (`d` and `D`).

Hitting bricks with the ball will destory the bricks, and reward the player with score points. Every 10 bricks destroyed will spawn a new ball, with up to 3 balls on the screen at any given time.

If a ball leaves the bottom of the screen then it is destroyed, and if there are no more balls left the game ends.

To get a feel for this game, try it out in a terminal:

```
dcc breakout.c -o breakout
./breakout
```

You should read through `breakout.c`. There are comments throughout it that should help you understand what the program is doing [*citation needed*] — which you'll need for the next part of the assignment.

breakout.s: The Assignment

Your task in this assignment is to implement `breakout.s` in MIPS assembly.

You have been provided with some assembly and some helpful information in `breakout.s`. Read through the provided code carefully, then add MIPS assembly so it executes exactly the same as `breakout.c`.

The functions `run_command`, `print_deubg_info` and `print_screen_updates` have already been translated to MIPS assembly for you.

You have to implement the following functions in MIPS assembly:

- `print_welcome`
- `main`
- `read_grid_width`
- `game_loop`
- `initialise_game`
- `move_paddle`
- `count_total_active_balls`
- `print_cell`
- `register_screen_update`
- `count_balls_at_coordinate`
- `print_game`
- `spawn_new_ball`
- `move_balls`
- `move_ball_in_axis`
- `hit_brick`
- `check_ball_paddle_collision`
- `move_ball_one_cell`

```
1521 mipsy breakout.s
Welcome to 1521 breakout! In this game you
control a paddle (---) with
the a and d (or A and D for fast movement)
keys, and your goal is
to bounce the ball (*) off of the bricks
(digits). Every ten bricks
destroyed spawns an extra ball. The . key
will advance time one step.

Enter the width of the playing field: 12

 SCORE: 0
==============
|            |
|            |
|000111222333|
|000111222333|
|000111222333|
|000111222333|
|            |
|            |
|      *     |
|   ------   |
 >> ;
 SCORE: 5
==============
|            |
|            |
|000111222333|
|000111222333|
|000111222333|
|000111222333|
|000111222333|
|000   222333|
|     *      |
|            |
|            |
|   ------   |
 >> q
```

You must translate each function separately to MIPS assembler, following the standard calling conventions used in lectures. When translating a function, you must not make any assumptions about the behaviour or side effects of any other function which is called.

Subsets

This assignment is split into four subsets. Later subsets will involve more complex translation.

Subset	Functions	Performance Weight
Subset 0	• `print_welcome` • `main`	10%
Subset 1	• `read_grid_width` • `game_loop` • `initialise_game` • `move_paddle` • `count_total_active_balls` • `print_cell`	45%
Subset 2	• `register_screen_update` • `count_balls_at_coordinate` • `print_game` • `spawn_new_ball` • `move_balls`	25%
Subset 3	• `move_ball_in_axis` • `hit_brick` • `check_ball_paddle_collision` • `move_ball_one_cell`	20%

Commands

The `run_command` function calls various other functions. When translating you should follow the exact behaviour of the C code, however when testing you may find it useful to consult the following table of commands.

Command	Description	Function called
`a`	Move the paddle one cell left	`move_paddle`
`d`	Move the paddle one cell right	`move_paddle`
`A`	Move the paddle three cells left	`move_paddle`
`D`	Move the paddle three cells right	`move_paddle`
`.`	Simulate the movement of the ball(s)	`move_balls`
`;`	Simulate the movement of the ball(s) for 3 steps	`move_balls`
`,`	Simulate the movement of the ball(s) for ⅓ of a step	`move_balls`
`?`	Output the internal state of the game	`print_debug_info`
`h`	Output the welcome message	`print_welcome`
`s`	Output changes to the screen (used by `play-breakout`)	`print_screen_updates`
`p`	Print the game, and turn off auto-printing	`print_game`
`q`	Quit the game	—

Running & Testing

To run your MIPS code, simply enter the following in your terminal:

```
1521 mipsy breakout.s
```

Once you have finished your translation, to test your implementation, you can compile the provided C implementation, run it to collect the expected output, run your assembly implementation to collect observed output, and then compare them.

The game takes a lot of input, so it's a good idea to write a file with the input you want to test, and then pipe that into your program.

You have been given a file called `input.txt` as an example.

```
dcc breakout.c -o breakout
cat input.txt | ./breakout | tee c.out
cat input.txt | 1521 mipsy breakout.s | tee mips.out
diff -s c.out mips.out
Files c.out and mips.out are identical
```

Try this for different sequences of inputs. When testing some functions you may find using the `?` command (which calls `print_debug_info`) to be useful.

Hints

You should implement all the functions from one subset before moving on to the next.

You may find the provided `run_command`, `print_debug_info` and `print_screen_updates` function implementations to be useful guidance for your implementation including comments, label names, indentation and register usage.

Simplified C code

You are encouraged to simplify your C code to remove any loop constructs and if-else statements, and test that your simplified code works correctly before translating it to MIPS, in a separate file `breakout.simple.c`.

This file will not be marked - you do not need to submit it.

In order to allow you to check that your simplified code works correctly, we have provided a simple set of automated tests.

You can run these tests by running the following command:

```
1521 autotest breakout.simple
```

An example game of Breakout

Select to toggle an example game of Breakout

Assumptions, Clarifications, and Restrictions

Like all good programmers, you should make as few assumptions as possible.

Your submitted code must be hand-written MIPS assembly, which you yourself have written.
You may not submit code in other languages.
You may not submit compiled output.

You may not copy a solution from an online source. e.g. Github.

Your functions will be tested individually. They must exactly match the behaviour of the corresponding C function and they must follow MIPS calling conventions.

The C code defines constants using `#define`. Your MIPS translation should use the corresponding provided named constants, in the places where a `#define` is used in the C code. You should not use a `#define` constant in your MIPS translation if it is not used in the corresponding part of the C code.

There will be a correctness penalty for assignments that do not follow standard MIPS calling conventions including:

- Function arguments are passed in registers `$a0`..`$a3`.
- Function return values are passed in register `$v0`
- Values in registers `$s0`..`$s7` are preserved across function calls.
  If a function changes these registers, it must restore the original value before returning.
- The only registers' values that can be relied upon across a function call are `$s0`..`$s7`, `$gp`, `$sp`, and `$fp`.
  All other registers must be assumed to be have, an undefined value after a function call, except `$v0` which has the function return value.

If you need clarification on what you can and cannot use or do for this assignment, ask in the class forum.

You are required to submit intermediate versions of your assignment. See below for details.

Breakout wrapper

If you complete this assignment, you may notice that the finished game is not particularly fun to play. To make the game more interesting to play, you can run a wrapper script that adds extra functionality to your MIPS translation. In a directory which contains your **completed** `breakout.s`, run:

```
1521 play-breakout 60
```

You can change the parameter 60 to other numbers for different grid widths. You can also optionally supply another parameter `slow`, `medium` (which is the default), `fast` or `increasing` to alter the game speed. For example, this will start a game with `fast` speed and a grid width of 42:

```
1521 play-breakout 42 fast
```

This adds colour and automatic time progression amongst other things. Inputs from the set `aAdD` are recognised. You may need to experiment with which terminal you use as well as terminal size to get the best playing experience. **Note that this wrapper is just for fun, there are no marks associated with whether or not your translation works with the wrapper script.**

Change Log

Version 1.0
(2024-06-11 12:00:00)

- Initial release

Assessment

Testing

We have provided some automated tests to help you check the correctness of your translation. To run all the provided tests, execute the following command:

```
1521 autotest breakout
```

Some of these tests check only a specific function, and some test your whole program. To run all the tests for a specific function, pass the name of the function to `autotest`. For example, to run all the tests for the `print_welcome` function, run the command:

```
1521 autotest breakout print_welcome
```

You can also run all the autotests for a particular subset. For example, to run all the autotests for subset 1, run the command:

```
1521 autotest breakout S1
```

To run the autotests which test your program as a whole, run the command:

```
1521 autotest breakout whole_prog
```

Some tests are more complex than others. If you are failing more than one test, you are encouraged to focus on solving the first of those failing tests. To do so, you can run a specific test by giving its name to the `autotest` command:

```
1521 autotest breakout print_welcome_S0_0
```

Whilst we can detect that errors have occurred, it is often substantially harder to explain what that error was.
The errors from `1521 autotest` will be less clear and useful than in labs.
You will need to do your own debugging and analysis.

`1521 autotest` will not test everything. You are strongly encouraged to do your own testing. The provided autotests are less comprehensive for later subsets.

Whilst the function autotests for subset 0, 1 and 2 check for your conformance to the MIPS calling convention ('strict' autotests), the subset 3 autotests will not check whether your code follows the MIPS calling convention. However, the marking tests **will** check for conformance to the MIPS calling convention. This means that it is important that you check yourself that your code follows the MIPS calling convention, particularly for your subset 3 code.

Submission

When you are finished working on the assignment, you must submit your work by running `give`:

```
give cs1521 ass1_breakout breakout.s
```

You must run `give` before **Week 5 Friday 18:00:00** to obtain the marks for this assignment. Note that this is an individual exercise, the work you submit with `give` must be entirely your own.

You can run `give` multiple times.
Only your last submission will be marked.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

You *cannot* obtain marks by emailing your code to tutors or lecturers.

You can check your latest submission on CSE servers with:

```
1521 classrun check ass1_breakout
```

You can check the files you have submitted here.

Manual marking will be done by your tutor, who will mark for style and readability, as described in the **Assessment** section below. After your tutor has assessed your work, you can view your results here; The resulting mark will also be available via give's web interface.

Due Date

This assignment is due **Week 5 Friday 18:00:00** (2024-06-28 18:00:00).

The UNSW standard late penalty for assessment is 5% per day for 5 days - this is implemented hourly for this assignment.

Your assignment mark will be reduced by 0.2% for each hour (or part thereof) late past the submission deadline.

For example, if an assignment worth 60% was submitted half an hour late, it would be awarded 59.8%, whereas if it was submitted past 10 hours late, it would be awarded 57.8%.

Beware - submissions 5 or more days late will receive zero marks. This again is the UNSW standard assessment policy.

Assessment Scheme

This assignment will contribute **15** marks to your final COMP1521 mark.

80% of the marks for assignment 1 will come from the performance of your code on a large series of tests.

20% of the marks for assignment 1 will come from hand marking. These marks will be awarded on the basis of clarity, commenting, elegance and style. In other words, you will be assessed on how easy it is for a human to read and understand your program.

An indicative assessment scheme for performance follows.
The lecturer may vary the assessment scheme after inspecting the assignment submissions, but it is likely to be broadly similar to the following:

100% for performance	implements all behaviours perfectly, following the spec exactly.
85% for performance	implements all simple and most difficult functions correctly.
65% for performance	implements all simple and some moderate difficulty functions correctly.
≤ 50% for performance	good progress, simple functions work correctly.

An indicative assessment scheme for style follows.
The lecturer may vary the assessment scheme after inspecting the assignment submissions, but it is likely to be broadly similar to the following:

100% for style	perfect style
90% for style	great style, almost all style characteristics perfect.

80% for style	good style, one or two style characteristics not well done.
70% for style	good style, a few style characteristics not well done.
60% for style	ok style, an attempt at most style characteristics.
≤ 50% for style	an attempt at style.

An indicative style rubric follows.
The lecturer may vary the assessment scheme after inspecting the assignment submissions, but it is likely to be broadly similar to the following:

- Formatting **(8/20)**:
  - Whitespace
  - Indentation (consistent, tabs or spaces are okay)
  - Line length (below 120 characters unless very exceptional)
  - Line breaks (using vertical whitespace to improve readability)
- Documentation **(12/20)**:
  - Header comment (with name, zID, description of program)
  - Function comments (above each function with a description)
  - Sensible commenting throughout the code
  - Descriptive label names, indicating structure

Note that the following penalties apply to your total mark for plagiarism:

0 for assignment 1	knowingly providing your work to anyone and it is subsequently submitted (by anyone).
0 FL for COMP1521	submitting any other person's work; this includes joint work.
academic misconduct	submitting another person's work without their consent; paying another person to do work for you.

Intermediate Versions of Work

You are required to submit intermediate versions of your assignment.

Every time you work on the assignment and make some progress you should copy your work to your CSE account and submit it using the `give` command above. It is fine if intermediate versions do not compile or otherwise fail submission tests. Only the final submitted version of your assignment will be marked.

Assignment Conditions

- **Joint work** is **not permitted** on this assignment.

  This is an individual assignment. The work you submit must be entirely your own work: submission of work even partly written by any other person is not permitted.

  Do not request help from anyone other than the teaching staff of COMP1521 — for example, in the course forum, or in help sessions.

Do not post your assignment code to the course forum. The teaching staff can view code you have recently submitted with give, or recently autotested.

Assignment submissions are routinely examined both automatically and manually for work written by others.

Rationale: this assignment is designed to develop the individual skills needed to produce an entire working program. Using code written by, or taken from, other people will stop you learning these skills. Other CSE courses focus on skills needed for working in a team.

- The use of **code-synthesis tools**, such as **GitHub Copilot**, **ChatGPT**, **Google Bard**, etc. are **not permitted** on this assignment.

  Rationale: this assignment is designed to develop your understanding of basic concepts. Using synthesis tools will stop you learning these fundamental concepts, which will significantly impact your ability to complete future courses.

- **Sharing, publishing, or distributing** your assignment work is **not permitted**.

  Do not provide or show your assignment work to any other person, other than the teaching staff of COMP1521. For example, do not message your work to friends.

  Do not publish your assignment code via the Internet. For example, do not place your assignment in a public GitHub repository.

  Rationale: by publishing or sharing your work, you are facilitating other students using your work. If other students find your assignment work and submit part or all of it as their own work, you may become involved in an academic integrity investigation.

- **Sharing, publishing, or distributing** your assignment work after the completion of COMP1521 is **not permitted**.

  For example, do not place your assignment in a public GitHub repository after this offering of COMP1521 is over.

  Rationale: COMP1521 may reuse assignment themes covering similar concepts and content. If students in future terms find your assignment work and submit part or all of it as their own work, you may become involved in an academic integrity investigation.

Violation of any of the above conditions may result in an academic integrity investigation, with possible penalties up to and including a mark of 0 in COMP1521, and exclusion from future studies at UNSW. For more information, read the UNSW Student Code, or contact the course account.

COMP1521 24T2: Computer Systems Fundamentals is brought to you by
the School of Computer Science and Engineering
at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs1521@cse.unsw.edu.au
CRICOS Provider 00098G