Assignment 1: mips_sim, MIPS simulator

version: 1.4 last updated: 2020-10-26 10:10:00

Aims

- Understanding encoding of MIPS instructions
- Understanding semantics of MIPS instructions
- Generally building a concrete understanding of an example CPU
- Practising C, including bit operations

The Assignment

Your task in this assignment is to write `mips_sim` a simulator for a small simple subset of the MIPS .

The input to `mips_sim` will be the 32-bit instruction codes for MIPS instructions as hexadecimal numbers.

The command `1521 spim2hex` will give you the hex codes for MIPS instructions.

```
$ cat examples/42.s
    li   $a0, 42          # printf("%d", 42);
    li   $v0, 1
    syscall
    li   $a0, '\n'      # printf("%c", '\n');
    li   $v0, 11
    syscall
$ 1521 spim2hex examples/42.s
3404002a
34020001
c
3404000a
3402000b
c
```

`mips_sim.c` should simulate executing these instruction like this:

```
$ cat examples/42.hex
3404002a
34020001
c
3404000a
3402000b
c
$ dcc mips_sim.c -o mips_sim
$ ./mips_sim examples/42.hex
0: 0x3404002A ori  $4, $0, 42
>>> $4 = 42
1: 0x34020001 ori  $2, $0, 1
>>> $2 = 1
2: 0x0000000C syscall
>>> syscall 1
<<< 42
3: 0x3404000A ori  $4, $0, 10
>>> $4 = 10
4: 0x3402000B ori  $2, $0, 11
>>> $2 = 11
5: 0x0000000C syscall
>>> syscall 11
<<<
```

If the command-line argument **-r** is given then only the output from syscalls should be shown, like this:

```
$ ./mips_sim -r examples/42.hex
42
```

Reference implementation

A reference implementation is available as `1521 mips_sim` which can use to find the correctoutput for any input, like this:

```
$ cat examples/square.hex
34100004
34110003
72108002
72318802
2302020
34020001
c
3404000a
3402000b
c
$ 1521 mips_sim examples/square.hex
0: 0x34100004 ori  $16, $0, 4
>>> $16 = 4
1: 0x34110003 ori  $17, $0, 3
>>> $17 = 3
2: 0x72108002 mul  $16, $16, $16
>>> $16 = 16
3: 0x72318802 mul  $17, $17, $17
>>> $17 = 9
4: 0x02302020 add  $4, $17, $16
>>> $4 = 25
5: 0x34020001 ori  $2, $0, 1
>>> $2 = 1
6: 0x0000000C syscall
>>> syscall 1
<<< 25
7: 0x3404000A ori  $4, $0, 10
>>> $4 = 10
8: 0x3402000B ori  $2, $0, 11
>>> $2 = 11
9: 0x0000000C syscall
>>> syscall 11
<<<

$ 1521 mips_sim -r examples/square.hex
25
```

Provision of a reference implementation is a common, efficient and effective method to provide or define an operational specification, and it's something you will likely need to work with after you leave UNSW.

Where any aspect of this assignment is undefined in this specification you should match the reference implementation's behaviour.

Discovering and matching the reference implementation's behaviour is deliberately part of the assignment.

If you discover what you believe to be a bug in the reference implementation, report it in the class forum. If it is a bug, we may fix the bug, or indicate that you do not need to match the reference implementation's behaviour in this case.

MIPS Instruction Subset

You need to implement only these 10 MIPS instructions:

Assembler	C	Bit Pattern
add $d, $s, $t	d = s + t	000000ssssstttttddddd00000100000
sub $d, $s, $t	d = s - t	000000ssssstttttddddd00000100010
slt $d, $s, $t	d = s < t	000000ssssstttttddddd00000101010
mul $d, $s, $t	d = s * t	011100ssssstttttddddd00000000010
beq $s, $t, I	if (s == t) PC += I	000100ssssstttttIIIIIIIIIIIIIIII
bne $s, $t, I	if (s != t) PC += I	000101ssssstttttIIIIIIIIIIIIIIII
addi $t, $s, I	t = s + I	001000ssssstttttIIIIIIIIIIIIIIII
ori $t, $s, I	t = s | I	001101ssssstttttIIIIIIIIIIIIIIII
lui $t, I	t = I << 16	00111100000tttttIIIIIIIIIIIIIIII
syscall	syscall	00000000000000000000000000001100

The instruction ***bit pattern*** uniquely identifies each instruction:

- **0**: Literal bit zero
- **1**: Literal bit one
- **I**: Immediate (16-bit signed number)
- **d**, **s**, **t**: five-bit register number

System Calls

You only need to implement these 3 system calls.

Description	$v0	Pseudo-C
print integer	1	`printf("%d", $a0)`
exit	10	`exit(0)`
print character	11	`printf("%c", $a0)`

Syscall 11 should print the low byte (lowest 8 bits) of `$a0`.

If an invalid syscall number is supplied an error message should be printed:

Match the reference implementation's message:

```
$ 1521 mips_sim examples/bad_syscall.hex
0: 0x34021092 ori  $2, $0, 4242
>>> $2 = 4242
1: 0x0000000C syscall
>>> syscall 4242
Unknown system call: 4242
```

Registers

All 32 registers are set to be zero when execution begins.

The value of register $0 ($zero) is always 0. Instructions which attempt to change it have no effect.

The values of registers $2 ($v0) and $4 ($a0) are used by the syscall instruction.

The other 31 registers have no special meaning and can be used for any purpose.

Halting

Execution halts if an exit syscall is executed.

Execution halts if it reaches the location after the last instruction.

Execution halts if there is a branch to the location immediately after the last instruction.

Execution halts with an error message if there is a branch to any other location beyond the range of specified instructions:

```
Illegal branch to address before instructions: PC = ??
Illegal branch to address after instructions: PC = ??
```

Examples

Some example MIPS programs are available as a zip file

You will also need to do your own testing and construct your own examples using `1521 spim2hex`.

Note the assembler for the example programs contains pseudo-instructions such as **li**.

To make it easy to use existing code as examples for this assignment `1521 spim2hex` translates some pseudo-instructions and a few instructions outside the subset of this assignment to instructions in the subset for this assignment. This is just for convenience. You do not have to implement pseudo-instructions or instructions outside the specified subset. Also for convenience `1521 spim2hex` deletes the last instruction if it is **jr** because **jr** is not in the subet for this assignment.

Getting Started

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

```
$ mkdir -m 700 mips_sim
$ cd mips_sim
$ 1521 fetch mips_sim
$ unzip examples.zip
....
```

Or, if you're not working on CSE, you can download the `examples.zip` and starting code

You have been given starting code for this assignment in `mips_sim.c` which already implements handling command line arguments and reading the hexademical instruction code into an array.

```
$ dcc mips_sim.c -o mips_sim
$ ./mips_sim examples/42.hex
0: 3404002a
1: 34020001
2: 0000000c
3: 3404000a
4: 3402000b
5: 0000000c
```

```
5: 00000000
```

The code calls the function **execute_instructions** to simulate execution but the supplied code in **execute_instructions** only prints the instruction codes.

You need to change the code in **execute_instructions** to simulate the execution of the instructions.

You will need to add extra functions and #defines.

You may create extra.c or .h files.

Assumptions and Clarifications

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

If in doubt, match the output of the reference implementation.

You can assume `mips_sim.c` is given a single file as a command line argument

You do not have to implement MIPS instructions, system calls, or features which are not explicitly mentioned in the tables above.

Your program should print an error message if given a hexadecimal number which does not correspond to an instruction in the above MIPS subset.

You can print this error message before executing the program or when execution reaches the invalid instruction code.

Match the reference implementation's message:

```
$ echo 12858AA > invalid.hex
$ cat invalid.hex
12858AA
$ 1521 mips_sim invalid.hex
0: 0x012858AA invalid instruction code
```

The reference implementation uses `%08X` to print invalid instruction codes.

You will not be penalized if you implement extra MIPS instructions beyond the subset above and do not print an error message for them.

Execution halts with an error message if there is a system call which is not in this subset. You can assume overflow does not occur during arithmetic or other operations.

You do not need to handle instructions which access memory such as **lw** or **sw**.

You do not need to handle branch labels. **1521 spim2hex** translates these into the relative offset which is part of the branch instruction code.

Some of the example assembler (`.s`) files contain pseudo-instructions, for example **li** and instructions outside the assignment subset (e.g **jr**). You do not need to handle these instructions/pseudo-instruction .

The corresponding `.hex` files contains only instructions in the assignment subset.

For convenience **spim2hex** translates some psuedo-instructions and a few instructions outside the subset into instruction codes in the assignment subset.

When running MIPS programs the reference implementation print all messages to **stdout** including messgaes indicating errors in the program. Some of these messages might normally be printed to **stderr**. This would complicateing autotest/automarking. You should also print all messages to **stdout**.

Your submitted code must be C only. You may call functions from the standard C library (e.g., functions from `stdio.h`, `stdlib.h`, `string.h`, etc.) and the mathematics library (`math.h`). You may use `assert.h`.

You may not submit code in other languages. You may not use `system` or other C functions to run external programs. You may not use functions from other libraries; in other words, you cannot use dcc's `-l` flag.

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.

Your program must not require extra compile options. It must compile with `dcc *.c -o mips_sim`, and it will be run with `dcc` when marking. Run-time errors from illegal C will cause your code to fail automarking.

If your program writes out debugging output, it will fail automarking tests: make sure you disable debugging output before submission.

Assessment

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 1521 autotest mips_sim mips_sim.c [any other .c or .h files]
```

Submission

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

```
$ give cs1521 ass1_mips_sim mips_sim.c [other .c or .h files]
```

You must run `give` before **Sunday November 01 21:00 2020** 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 e-mailing your code to tutors or lecturers.

You can check your latest submission on CSE servers with:

```
$ COMP1521 classrun -check ass1_mips_sim
```

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 tentatively due **Sunday November 01 21:00 2020**.

If your assignment is submitted after this date, each hour it is late reduces the maximum mark it can achieve by 2%. For example, if an assignment worth 74% was submitted 10 hours late, the late submission would have no effect. If the same assignment was submitted 15 hours late, it would be awarded 70%, the maximum mark it can achieve at that time.

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

HD (90+)	beautiful documented code, works perfectly for all programs
CR/DN (70+)	very readable code, works for most programs
PS/CR (60+)	readable code, works for some simple programs
PS (50+)	close to working for some simple very short programs
0%	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 below. 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.

All these intermediate versions of your work will be placed in a Git repository and made available to you via a web interface at `https://gitlab.cse.unsw.edu.au/`*`z5555555`*`/20T3-comp1521-ass1_mips_sim` (replacing *z5555555* with your own zID). This will allow you to retrieve earlier versions of your code if needed.

Attribution of Work

This is an individual assignment.

The work you submit must be entirely your own work, apart from any exceptions explicitly included in the assignment specification above. Submission of work partially or completely derived from any other person or jointly written with any other person is not permitted.

You are only permitted to request help with the assignment in the course forum, help sessions, or from the teaching staff (the lecturer(s) and tutors) of COMP1521.

Do not provide or show your assignment work to any other person (including by posting it on the forum), apart from the teaching staff of COMP1521. If you knowingly provide or show your assignment work to another person for any reason, and work derived from it is submitted, you may be penalized, even if that work was submitted without your knowledge or consent; this may apply even if your work is submitted by a third party unknown to you. You will not be penalized if your work is taken without your consent or knowledge.

Do not place your assignment work in online repositories such as github or any where else that is publically accessible. You may use a private repository.

Submissions that violate these conditions will be penalised. Penalties may include negative marks, automatic failure of the course, and possibly other academic discipline. We are also required to report acts of plagiarism or other student misconduct: if students involved hold scholarships, this may result in a loss of the scholarship. This may also result in the loss of a student visa.

Assignment submissions will be examined, both automatically and manually, for such submissions.

Change Log

Version 1.0
(2020-10-14 12:00:00)
- Initial release.

Version 1.1
(2020-10-16 09:00:00)
- printf format in supplied code changed to use %08X

Version 1.3
(2020-10-17 20:00:00)
- Fixed a bug in 1521 mips_sim where ORI and ADDI would load incorrect values

Version 1.4
(2020-10-26 10:10:00)
- Example output from starter code corrected