4190.308: Computer Architecture (Fall 2018)

Project #4: Enhancing the Sequential Y86-64 Processor

Due: December 3rd (Monday), 11:59PM

1. Introduction

In this project, you will learn about the design and implementation of a sequential Y86-64 processor by enhancing its ISA (Instruction Set Architecture).

2. New instructions for Y86-64 processor

Your task is to extend the Y86-64 ISA to support new instructions: `iaddq`, `mulq`, `divq`, `rmmovb`, and `mrmovb`.

2.1 **iaddq** : `rB ← rB + V`

`iaddq V, rB`

C	0	F	rB	V

The `iaddq` instruction adds the 64-bit constant value `V` to register `rB`. The condition codes should be set accordingly.

2.2 **mulq** : `rB ← rB * rA`

`mulq rA, rB`

6	4	rA	rB

The `mulq` instruction multiplies `rA` and `rB` and stores the result to register `rB`. The condition codes should be set accordingly.

2.3 **divq** : `rB ← rB / rA`

`divq rA, rB`

6	5	rA	rB

The `divq` instruction divides `rB` by `rA` and stores the result to register `rB`. The result should be the same as that of the integer division in C (i.e., non-integral results are truncated towards 0). When the divisor (the value of `rA`) is zero, the result is set to **TMin** (= `0x8000000000000000`) and the OF flag is set. Other ZF and SF flags are set according to the result (even when the divisor is zero).

2.4 **rmmovb** : `M1[rB+D] ← LSB(rA)`

`rmmovb rA, D(rB)`

4	1	rA	rB	D

The `rmmovb` instruction stores the LSB (Least Significant Byte) of the register `rA` into the memory address `rB + D`, where `D` is a 64-bit constant. No condition codes are affected.

2.5 **mrmovb** : `rA ← Zero_Extend(M1[rB+D])`

`mrmovb D(rB), rA`

5	1	rA	rB	D

The `mrmovb` instruction reads a single byte from the memory address `rB + D` and stores it into the LSB (Least Significant Byte) of the register `rA`. Other remaining bits in register `rA` are cleared to 0. No condition codes are affected.

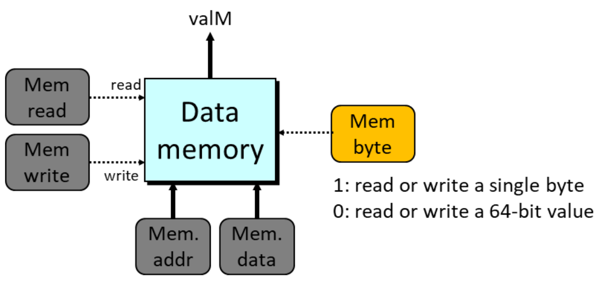
3. Enhancing the Sequential Y86-64 simulator (SEQ)

You will be mainly working in `./misc` and `./seq` directories for this project. The files in `./misc` directory are used by the Y86-64 assembler (`yas`) and Y86-64 instruction simulator (`yis`). We have already modified some of files to make the Y86-64 assembler (`yas`) understand the new instructions when it generates the binary codes (`*.yo`) from the assembly codes (`*.ys`).

Your task is to enhance the SEQ simulator so that it simulates the operations of new instructions correctly. Most of the SEQ simulator codes are stored in `./seq` directory, but some of files in `./misc` directory are used as well to build the SEQ simulator (e.g., `./misc/isa.c`, `./misc/isacore.c`, and `./misc/isa.h`). Among others, you need to pay attention to the following files:

`./seq/ssimcore.c`	This file implements the core logic of the SEQ simulator.
`./seq/seq-full.hcl`	This file describes the control logic of the SEQ simulator in HCL (Hardware Control Language).
`./misc/isacore.c`	This file contains the codes that simulate the operation of each instruction.

Note that, in order to support `rmmovb` and `mrmovb` instructions, we have slightly enhanced the memory model in the SEQ simulator by adding a control signal called "`mem_byte`". When the `mem_byte` signal is set to 1, only a single byte is read from or written to the given memory address. If the `mem_byte` signal is 0, the whole 64-bit data is accessed. Therefore, you have to generate the `mem_byte` control signal properly according to the current instruction executed.



4. Skeleton codes

The following skeleton codes are provided for this project. These skeleton codes are based on the Y86-64 toolset developed by textbook authors available at http://csapp.cs.cmu.edu/3e/sim.tar

`README`	The original README file in the Y86-64 toolset.
`README.SNU`	This file summarizes some of changes made for this project.
`Makefile`	This is a top-level Makefile used by the GNU `make` utility.
`simguide.pdf`	An official guide to the Y86-64 simulators. You must read this file first.
`misc/`	This directory contains the files for Y86-64 assembler (`yas`) and instruction simulator (`yis`).
`seq/`	This directory contains the files for implementing the SEQ Y86-64 simulator.
`y86-code/`	This directory has the sample codes written in Y86-64. We have added several test programs for new instructions such as `iaddq1.ys`, `iaddq2.ys`, `mulq1.ys`, `mulq2.ys`, `divq1.ys`, `divq2.ys`, `mrmovb.ys`, and `mrmovb.ys`.

5. Hand in instructions

- You need to submit `./misc/isacore.c`, `./seq/seq-full.hcl`, and `./seq/ssimcore.c` files only. You can do this by performing "`make handin`" in the top directory. It will generate the `pa4.tar.gz` file in the `./handin` directory. Upload this file to the submission site (`http://sys.snu.ac.kr`).

6. Logistics

- You will work on this assignment alone.
- If you have any questions, please feel free to post them in the QnA board.
- Only the assignments submitted before the deadline will receive the full credit. 25% of the credit will be deducted for every single day delay.
- You can use up to 5 ***slip days*** during this semester. Please let us know the number of slip days you want to use in the QnA board in the submission site within 1 week after the deadline.
- Any attempt to copy others' work will result in heavy penalty (for both the copier and the originator). Don't take a risk.

Have fun!

Jin-Soo Kim
Systems Software & Architecture Laboratory
Dept. of Computer Science and Engineering
Seoul National University

Appendix: Step-by-step examples

Assume that the skeleton code is extracted in the /pa4 directory.

```
$ cd /pa4
$ ls
Makefile  misc  README  README.SNU  seq  simguide.pdf  y86-code
$ make                                                    // make everything
(cd misc; make all)
make[1]: Entering directory '/pa4/misc'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/pa4/misc'
(cd seq; make all GUIMODE= TKLIBS="" TKINC="")
make[1]: Entering directory '/pa4/seq'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/pa4/seq'
(cd y86-code; make all)
make[1]: Entering directory '/pa4/y86-code'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/pa4/y86-code'
$ ./seq/ssim y86-code/iaddq1.yo                            // run iaddq1.yo on SEQ simulator
21 bytes of code read
IF: Fetched irmovq at 0x0.  ra=----, rb=%rax, valC = 0x105e
IF: Fetched iaddq at 0xa.  ra=----, rb=----, valC = 0x0
2 instructions executed
Status = INS                                               // Initially, you get invalid instr. error. Fix it.
Condition Codes: Z=1 S=0 O=0
Changed Register State:
%rax:   0x0000000000000000      0x000000000000105e
Changed Memory State:
$ ./seq/ssim y86-code/rmmovb.yo                            // run rmmovb.yo on SEQ simulator
31 bytes of code read
IF: Fetched irmovq at 0x0.  ra=----, rb=%rdx, valC = 0x100
IF: Fetched irmovq at 0xa.  ra=----, rb=%rax, valC = 0xcafebabe12345678
IF: Fetched rmmovb at 0x14.  ra=%rax, rb=%rdx, valC = 0x0
Wrote 0xcafebabe12345678 to address 0x100
IF: Fetched halt at 0x1e.  ra=----, rb=----, valC = 0x0
4 instructions executed
Status = HLT
Condition Codes: Z=1 S=0 O=0
Changed Register State:
%rax:   0x0000000000000000      0xcafebabe12345678          // Currently, you have wrong result. Fix it.
%rdx:   0x0000000000000000      0x0000000000000100
Changed Memory State:
0x0100: 0x0000000000000000      0xcafebabe12345678
$ make handin                                               // Make a tar file for submission
Submit handin/pa4.tar.gz file to the sys.snu.ac.kr server
$ ls -l handin
total 56
-rw-r--r-- 1 jinsoo jinsoo 9736 11월 26 18:04 isacore.c
-rw-rw-r-- 1 jinsoo jinsoo 6271 11월 26 18:04 pa4.tar.gz    // Submit this file
-rw-r--r-- 1 jinsoo jinsoo 7123 11월 26 18:04 seq-full.hcl
-rw-r--r-- 1 jinsoo jinsoo 8755 11월 26 18:04 ssimcore.c
```