서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

Systems Software Laboratory
Dept. of Computer Science and Engineering

4190.308: Computer Architecture (Spring 2018)

Project #4: Drawing grid lines on a sequential Y86-64 processor

Due: May 27th (Sunday), 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) and writing a problem that works on it. This project is organized into two parts. In Part A, you will extend the SEQ simulator with new instructions. In Part B, you will write the same program as Project #3, but this time using Y86-64 instructions only.

2. Part A: Enhancing the SEQ simulator with new instructions

2.1 New instructions

Your task in Part A is to extend the sequential Y86-64 processor to support new instructions: `iaddq`, `mulq`, `rmmovb`, and `mrmovb`.

2.1.1 `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.

2.1.2 `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.

2.1.3 `rmmovb` : M_1`[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.

2.1.4 mrmovb : rA ← Sign_Extend(M_1[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 set to the sign bit of the original value read from the memory. No condition codes are affected.

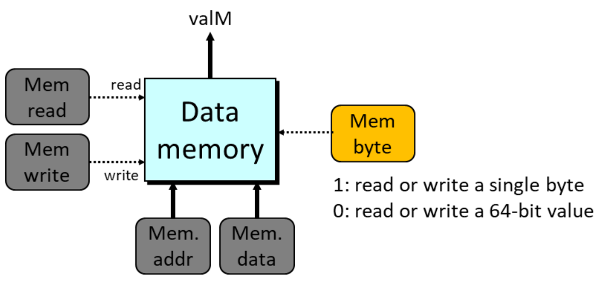
2.2 Enhancing the SEQ simulator

You will be mainly working in ./misc and ./seq directories for this part. 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.



3. Part B: Writing the bmp_grid() function in Y86-64 assembly code

Your task in Part B is to rewrite the `bmp_grid()` function in Project #3 using the Y86-64 ISA enhanced with `iaddq`, `mulq`, `rmmovb`, and `mrmovb` instructions. For this part, you will be working in the `./bmpgrid` directory.

```
void bmp_grid (unsigned char *imgptr, long long width, long long height,
               long long gap);
```

As in Project #3, four arguments are passed in `%rdi`, `%rsi`, `%rdx`, and `%rcx` registers, respectively. In this project, there is no limitation in the register use. You can freely use all the registers available in the Y86-64 architecture (e.g., `%rax`, `%rbx`, `%rcx`, `%rdx`, `%rsi`, `%rdi`, `%rbp`, `%rsp`, `%r8` ~ `%r14`). Remember that there is no `%r15` in Y86-64. The basic skeleton of the Y86-64 assembly code is shown below.

```
# bmp_grid() starts here
    .pos    0x400
    .align 8
bmp_grid:
    # imgptr is in %rdi
    # width  is in %rsi
    # height is in %rdx
    # gap    is in %rcx

    # ---- FILL HERE ----



    ret
```

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.
`bmpgrid/`	This directory contains a template file for implementing `bmp_grid()`. In this directory, you can perform "`make test`" to see if your code produces the correct result.
`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`, `mrmovb.ys`, and `mrmovb.ys`.
`ptest/`	This directory contains automatic testing scripts for individual instructions. For more details, please refer to the `simguide.pdf` file.

5. Requirements

- You can freely use all the registers available in the Y86-64 architecture.
- Your `bmp_grid()` implementation should work for BMP images of any size.
- Your `bmp_grid()` implementation should work for any positive value of "`gap`".
- Your `bmp_grid()` implementation should leave the bytes in the padding area untouched.

6. Hand in instructions

- You need to submit `./misc/isacore.c`, `./seq/seq-full.hcl`, `./seq/ssimcore.c` and `./bmpgrid/bmpgrid.ys` 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.skku.edu`).

7. 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.

8. Step-by-step examples

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

```
$ cd /pa4
$ ls
bmpgrid  Makefile  misc  ptest  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'
(cd bmpgrid; make)
make[1]: Entering directory '/pa4/bmpgrid'
/bin/cat bmpmain.ys bmpgrid.ys > bmptest.ys
../misc/yas bmptest.ys bmptest.yo
make[1]: Leaving directory '/pa4/bmpgrid'
$ ./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
$ cd bmpgrid
$ make test                                                    // Test your implementation of bmpgrid.ys
make[1]: Entering directory '/pa4/bmpgrid'
/bin/cat bmpmain.ys bmpgrid.ys > bmptest.ys
../misc/yas bmptest.ys bmptest.yo
../seq/ssim -s bmptest.yo
8 instructions executed
```

```
make[1]: Leaving directory '/pa4/bmpgrid'
1a2,1657
> 0x1000:      0x33b02815d1c865fe     0x33b02815d1ff0000
> 0x1018:      0x73ca4175ec41ec9d     0x00004175ec41ec9d
> 0x1020:      0x1eeef5bbbb4492a6     0x1eeef5bbbb4492ff
....
> 0x8bd8:      0x06ece637ac94020a     0x0000ff0000ff0000
> 0x8be0:      0xade9c8d8c15fa43e     0x00ff0000ff0000ff
> 0x8be8:      0x000000000000f12a     0x000000000000ff00
Makefile:47: recipe for target 'test' failed            // Your output is incorrect. Fix it.
make: *** [test] Error 1
$ 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 64
-rw------- 1 jinsoo jinsoo  538  5월 17 14:02 bmpgrid.ys
-rw-r--r-- 1 jinsoo jinsoo 9736  5월 17 14:02 isacore.c
-rw-rw-r-- 1 jinsoo jinsoo 6566  5월 17 14:02 pa4.tar.gz         // Submit this file
-rw-r--r-- 1 jinsoo jinsoo 7123  5월 17 14:02 seq-full.hcl
-rw-r--r-- 1 jinsoo jinsoo 8755  5월 17 14:02 ssimcore.c
```

Have fun!

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