4190.308: Computer Architecture Lab. 3

Jaehoon Shim
Seongyeop Jeong
Ilkueon Kang
Wookje Han
Jinsol Park
(snucsl.ta@gmail.com)

Systems Software & Architecture Lab.

Seoul National University

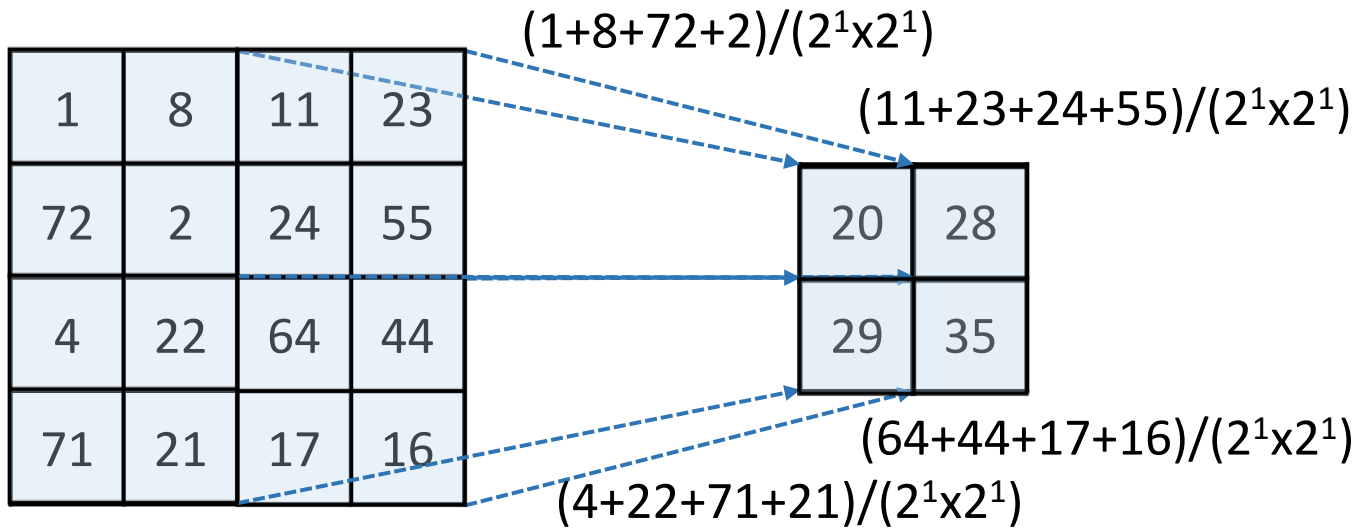
Fall 2022

Image Resizing

Image Resizing

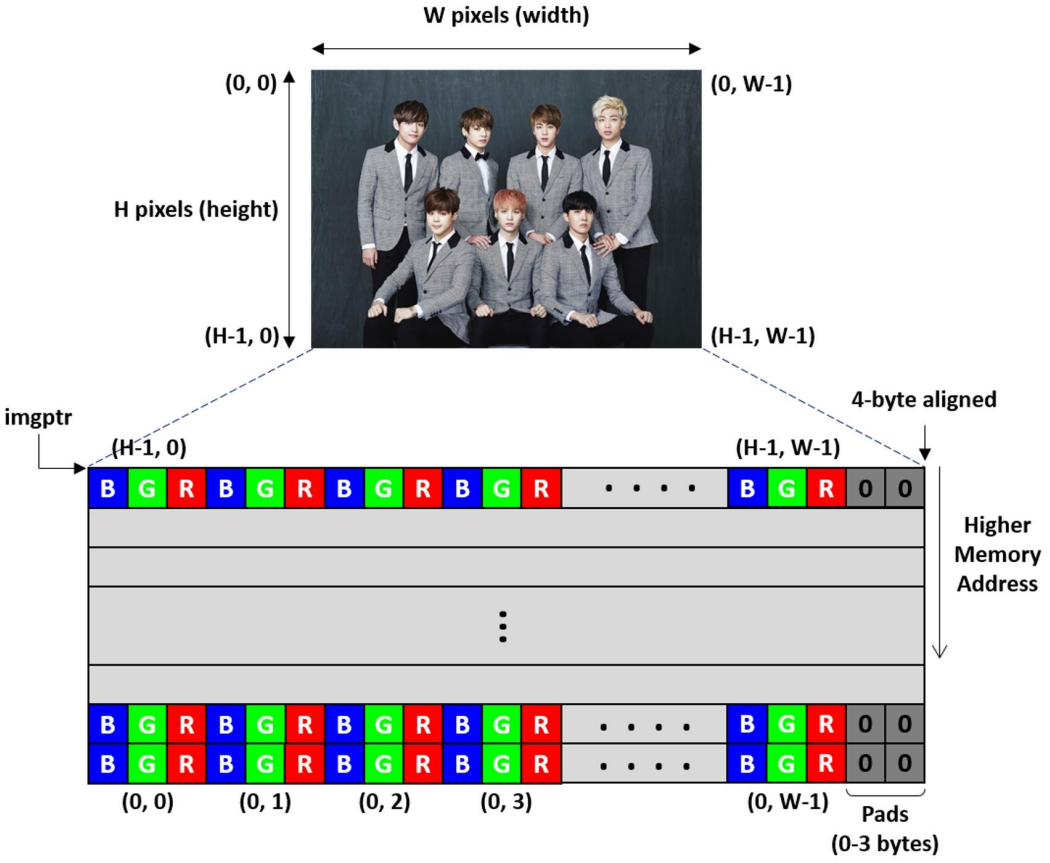
- We will only consider scaling down an image by a factor of 2^k, where each $2^k x 2^k$ pixels in the original image is replaced by a single pixel
  - The value of the new pixel is determined by taking an average of the original pixels
- Since each pixel has three color values (BGR), you need to compute the average value for each color

Input image: 4x4
Scaling factor: 2^1
Output image: 2x2



BMP Data Format

- Bitmap data describes the image pixel by pixel
- Each pixel consists of 8-bit blue(B), green(G), and red(R) byte
- Pixels are stored "upside-down"
- # of bytes occupied by each row should be a multiple of 4
  - If not, the remaining bytes are padded with zero



Specification

- *void bmpresize(unsigned char *imgptr, int h, int w, int k, unsigned char *outptr)*
  - *imgptr* points to the bitmap data that stores the actual image, pixel by pixel
  - *h* & *w* represent the height and width of the given image (in pixels)
  - *k* is the scaling factor (image is scaled down by 2^k)
  - *outptr* points to the address of the output image

  - *k* is an integer larger than zero
  - $h >= 2^k$, $w >= 2^k$
  - Both *h* and *w* are the multiple of 2^k

- parameters for *bmpresize*() are available in the a0 ~ a4 registers

Restrictions

- You should use only the following registers in bmpresize.s
  : zero(x0), sp, ra, a0~a4, t0~t4
  - Use stack as temporary storage if needed (maximum 128 bytes)

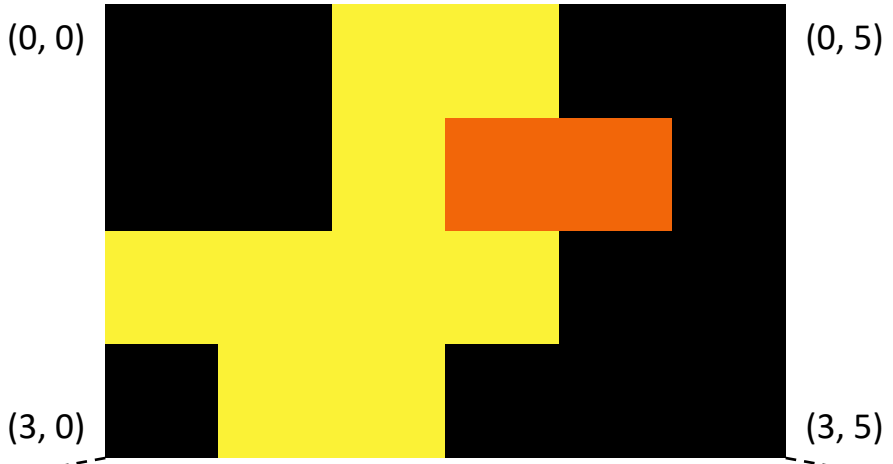
- You should use *lw* and *sw* RISC-V instructions to access data in memory
  - *lw/sw*: load/store a 4-byte word from/to memory
  - You should consider byte ordering (little endian)

- The padding area in the output image should be set to zero

Restrictions

- Contents of the output buffer after the output image should not be corrupted
- Your solution should not contain any register names other than the allowed ones
- Your solution will be rejected if it contains keywords like
  - .data, .octa, .quad, .long, .int, .word, .short, .hword, .byte, .double, .single, .float, etc
- Your solution should finish within a reasonable time

Example – Input Image



imgptr

0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x0a	0x6a	0xfa	0x0a	0x6a	0xfa	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

Example – Input Image

- Input stream (in 4-byte unit)

0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x0a	0x6a	0xfa	0x0a	0x6a	0xfa	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

```
0x35000000
0xf235fbf2
0x000000fb
0x00000000
0x00000000

0x35fbf235
0xf235fbf2
0xfbf235fb
0x00000000
0x00000000

0x00000000
0xf2350000
0xfa6a0afb
0x00fa6a0a
0x00000000

0x00000000
0xf2350000
0xfbf235fb
0x00000000
0x00000000
```

Example – Input Image

- Input stream (in 4-byte unit)

RISC-V uses little endian
(Least significant byte has the smallest address)

0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x0a	0x6a	0xfa	0x0a	0x6a	0xfa	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

```
0x35000000
0xf235fbf2
0x000000fb
0x00000000
0x00000000

0x35fbf235
0xf235fbf2
0xfbf235fb
0x00000000
0x00000000

0x00000000
0xf2350000
0xfa6a0afb
0x00fa6a0a
0x00000000

0x00000000
0xf2350000
0xfbf235fb
0x00000000
0x00000000
```

Example – Input Image

- Input stream (in 4-byte unit)

RISC-V uses little endian
(Least significant byte has the smallest address)

0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x0a	0x6a	0xfa	0x0a	0x6a	0xfa	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

```
0x35000000
0xf235fbf2
0x000000fb
0x00000000
0x00000000

0x35fbf235
0xf235fbf2
0xfbf235fb
0x00000000
0x00000000

0x00000000
0xf2350000
0xfa6a0afb
0x00fa6a0a
0x00000000

0x00000000
0xf2350000
0xfbf235fb
0x00000000
0x00000000
```

Example – Input Image

- Input stream (in 4-byte unit)

RISC-V uses little endian
(Least significant byte has the smallest address)

0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x0a	0x6a	0xfa	0x0a	0x6a	0xfa	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

```
0x35000000
0xf235fbf2
0x000000fb
0x00000000
0x00000000

0x35fbf235
0xf235fbf2
0xfbf235fb
0x00000000
0x00000000

0x00000000
0xf2350000
0xfa6a0afb
0x00fa6a0a
0x00000000

0x00000000
0xf2350000
0xfbf235fb
0x00000000
0x00000000
```

Example – Input Image

- Input stream (in 4-byte unit)

RISC-V uses little endian
(Least significant byte has the smallest address)

0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x0a	0x6a	0xfa	0x0a	0x6a	0xfa	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

```
0x35000000
0xf235fbf2
0x000000fb
0x00000000
0x00000000

0x35fbf235
0xf235fbf2
0xfbf235fb
0x00000000
0x00000000

0x00000000
0xf2350000
0xfa6a0afb
0x00fa6a0a
0x00000000

0x00000000
0xf2350000
0xfbf235fb
0x00000000
0x00000000
```

Example – Input Image

- Input stream (in 4-byte unit)

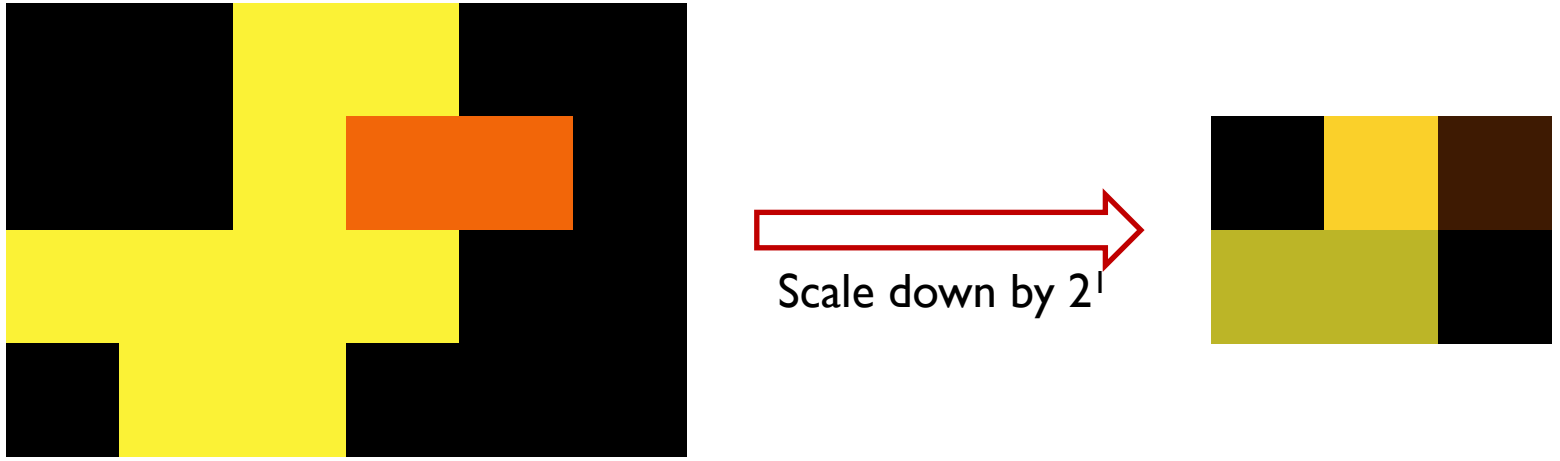
RISC-V uses little endian
(Least significant byte has the smallest address)

0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x0a	0x6a	0xfa	0x0a	0x6a	0xfa	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

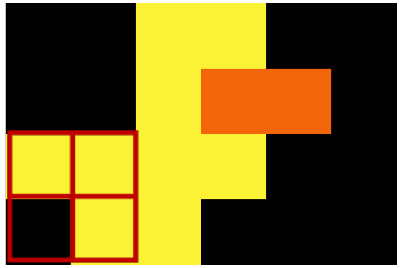
```
0x35000000
0xf235fbf2
0x000000fb
0x00000000
0x00000000

0x35fbf235
0xf235fbf2
0xfbf235fb
0x00000000
0x00000000

0x00000000
0xf2350000
0xfa6a0afb
0x00fa6a0a
0x00000000

0x00000000
0xf2350000
0xfbf235fb
0x00000000
0x00000000
```

Example – Image Resizing

- Input image size: 4x6
- k = 1 → scale down the original image by 2^1
- Output image size: $(4/2^1)x(6/2^1)$ = 2x3



Example – Image Resizing



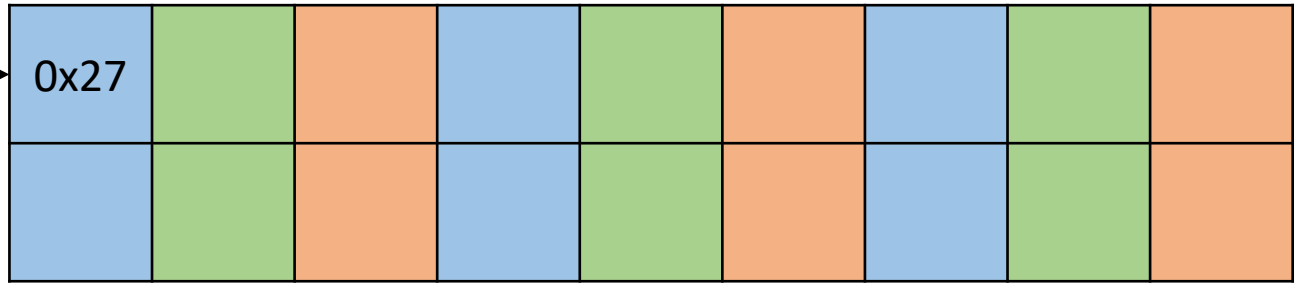
Start with 8-bit blue byte

0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x0a	0x6a	0xfa	0x0a	0x6a	0xfa	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00

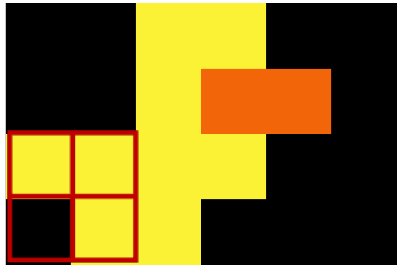
`avg(0x00+0x35+0x35+0x35) =`

`(0+53+53+53)/4 = 39(`**`0x27`**`)`

outptr → (*h*:2,*w*:3)



Example – Image Resizing

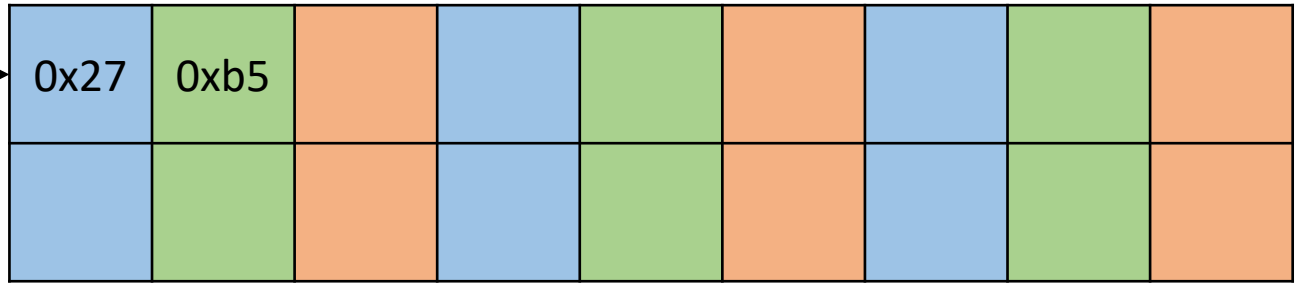


Continue with 8-bit green byte

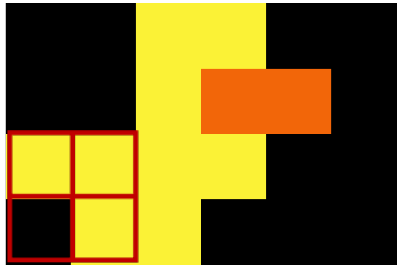
0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x0a	0x6a	0xfa	0x0a	0x6a	0xfa	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00

```
avg(0x00+0xf2+0xf2+0xf2) =

(0+242+242+242)/4 = 181(0xb5)
```

outptr → (*h*:2,*w*:3)



Example – Image Resizing



Continue with 8-bit red byte

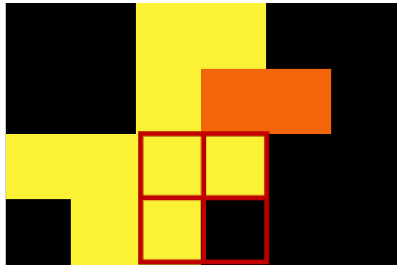
0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x0a	0x6a	0xfa	0x0a	0x6a	0xfa	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00

```
avg(0x00+0xfb+0xfb+0xfb) =

(0+251+251+251)/4 = 188(0xbc)
```

outptr → (*h*:2,*w*:3)

0x27	0xb5	0xbc						

Example – Image Resizing



Move to next target pixels and start again with 8-bit blue byte

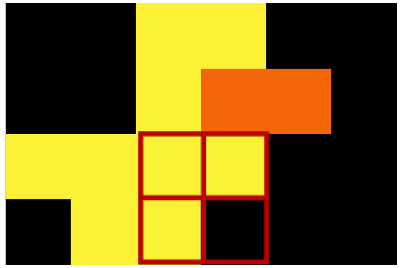
0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x0a	0x6a	0xfa	0x0a	0x6a	0xfa	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00

```
avg(0x35+0x00+0x35+0x35) =

(53+0+53+53)/4 = 39(0x27)
```

outptr →
(*h*:2,*w*:3)

0x27	0xb5	0xbc	0x27					

Example – Image Resizing



Continue with 8-bit green byte

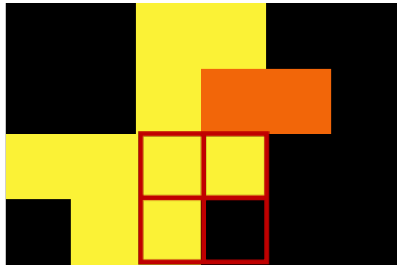
0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x0a	0x6a	0xfa	0x0a	0x6a	0xfa	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00

```
avg(0xf2+0x00+0xf2+0xf2) =

(242+0+242+242)/4 = 181(0xb5)
```

outptr → (*h*:2,*w*:3)

0x27	0xb5	0xbc	0x27	0xb5				

Example – Image Resizing



Continue with 8-bit red byte

0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x0a	0x6a	0xfa	0x0a	0x6a	0xfa	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

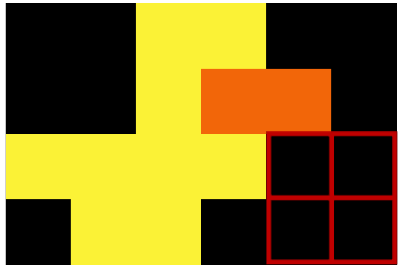
`avg(0xfb+0x00+0xfb+0xfb) =`

`(251+0+251+251)/4 = 188(`**`0xbc`**`)`

outptr →
(*h*:2,*w*:3)

0x27	0xb5	0xbc	0x27	0xb5	0xbc			

Example – Image Resizing



Move to next target pixels and start again with 8-bit blue, green, red byte

0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x0a	0x6a	0xfa	0x0a	0x6a	0xfa	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x35	0xf2	0xfb	0x35	0xf2	0xfb	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

`avg(0x00+0x00+0x00+0x00) =`

`(0+0+0+0)/4 = 0(`**`0x00`**`)`

outptr → (*h*:2,*w*:3)

0x27	0xb5	0xbc	0x27	0xb5	0xbc	0x00	0x00	0x00

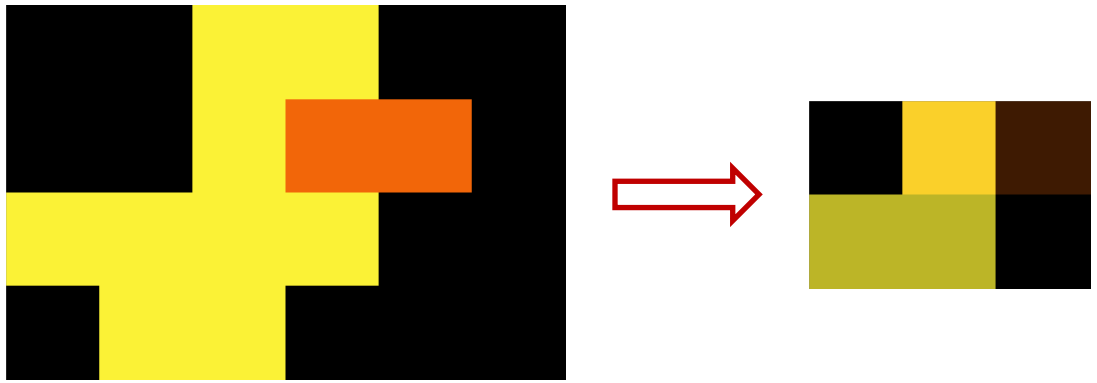
Example – Output Image

- In the bitmap data format, # of bytes occupied by each row should be a multiple of 4
- The remaining bytes should be padded with zeroes

outptr → (*w*:3, *h*:2)

0x27	0xb5	0xbc	0x27	0xb5	0xbc	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x2a	0xd0	0xfa	0x02	0x1a	0x3e	0x00	0x00	0x00

Padding



Submission

- Due: 11:59PM, November 20 (Sunday)
  - 25% of the credit will be deducted for every single day delay
- Submit the bmpresize.s file to the submission server
  - Program that contains unallowed keywords or register names will be rejected
- Submitted code will NOT be graded instantly
  - It will be graded every 6 hours (12am, 6am, 12pm, 6pm)
  - Only the last version submitted will be graded
- The top 10 implementations with smallest code size will receive a 10% extra bonus
- The next 10 implementations will receive a 5% extra bonus

How to use PyRISC

PyRISC

- It provides various RISC-V toolset written in Python
- It has snurisc, a RISC-V instruction set simulator that supports most of RV32I base instruction set (32-bit version!)
- You should work on either Linux or MacOS
  - We highly recommend you to use Ubuntu 18.04 LTS or later
- For Windows, we recommend installing WSL(Windows Subsystem for Linux) and Ubuntu

PyRISC Prerequisites

- PyRISC toolset requires Python version 3.6 or higher.
- You should install Python modules(numpy, pyelftools)

For Ubuntu 18.04 LTS,

```
$ sudo apt-get install python3-numpy python3-pyelftools
```

For MacOS,

```
$ pip install numpy pyelftools
```

RISC-V GNU Toolchain

- In order to work with the PyRISC toolset, you need to build a RISC-V GNU toolchain for the RV32I instruction set

- Please take the following steps to build it on your machine

Building RISC-V GNU toolchain

1. Install prerequisite packages

For Ubuntu 18.04 LTS,

```
$ sudo apt-get install autoconf automake autotools-dev curl libmpc-dev
$ sudo apt-get install libmpfr-dev libgmp-dev gawk build-essential bison flex
$ sudo apt-get install texinfo gperf libtool patchutils bc zlib1g-dev libexpat-dev
```

Building RISC-V GNU Toolchain

1. Install prerequisite packages

For MacOS, install Xcode command line tools and brew utility first

```
$ sudo xcode-select --install
$ /bin/bash -c "$(curl –fsSL
  https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

```
$ brew install gawk gnu-sed gmp mpfr libmpc isl zlib expat texinfo flock
```

Building RISC-V GNU Toolchain

2. Download the RISC-V GNU Toolchain from Github

```
$ git clone https://github.com/riscv/riscv-gnu-toolchain
```

3. Configure the RISC-V GNU toolchain

```
$ cd riscv-gnu-toolchain
$ mkdir build
$ cd build
$ ../configure --prefix=/opt/riscv --with-arch=rv32i --disable-gdb
```

Building RISC-V GNU Toolchain

4. Compile and install them

```
$ sudo make
```

For MacOS, if errors occur due to PATH or linking

```
$ sudo ln -s /opt/homebrew/bin /usr/local/bin
$ sudo ln -s /opt/homebrew/include /usr/local/include
$ sudo ln -s /opt/homebrew/lib /usr/local/lib
```

Building RISC-V GNU Toolchain

5. Add `/opt/riscv/bin` in your PATH

```
$ export PATH=/opt/riscv/bin:$PATH
```

Running RISC-V Executable File

- You should modify the `Makefile` in your ca-pa3 directory so that it can find the snurisc simulator

```
# in ca-pa3/Makefile
…

PYRISC = /dir1/dir2/pyrisc/sim/snurisc.py
PYRISCOPT = -l 1

…
```

Write the path where you downloaded pyrisc

Running RISC-V Executable File

- Now, you can run your RISC-V executable file for assignment 3 by performing '`make run`'!

```
../pyrisc/sim/snurisc.py       -l 1 bmpresize
Loading file bmpresize
Execution completed
Registers
=========
zero ($0): 0x00000000    ra ($1):   0x80000008    sp ($2):   0x80018000    gp ($3):   0x00000000
tp ($4):   0x00000000    t0 ($5):   0x00000002    t1 ($6):   0x00000002    t2 ($7):   0x80018014
s0 ($8):   0x00000000    s1 ($9):   0x00000000    a0 ($10):  0x800101a8    a1 ($11):  0x00000004
a2 ($12):  0x00000014    a3 ($13):  0x00000002    a4 ($14):  0x80018000    a5 ($15):  0x00000000
a6 ($16):  0x00000000    a7 ($17):  0x00000000    s2 ($18):  0x80010014    s3 ($19):  0x8001002c
s4 ($20):  0x80018018    s5 ($21):  0x80010244    s6 ($22):  0x80010244    s7 ($23):  0x00000000
s8 ($24):  0x0000003e    s9 ($25):  0x00000000    s10 ($26): 0x00000000    s11 ($27): 0x00000000
t3 ($28):  0x0000003e    t4 ($29):  0x00000000    t5 ($30):  0x00000000    t6 ($31):  0x00000000
56430 instructions executed in 56430 cycles. CPI = 1.000
Data transfer:    11675 instructions (20.69%)
ALU operation:    31633 instructions (56.06%)
Control transfer: 13122 instructions (23.25%)
```

If t6 is 0 after executing all instructions, your output image for every test is correct

If t6 is not 0, t6 contains the index of the testcase you failed

Running RISC-V Executable File

- When you failed to pass all test cases, check t5 to see the memory address of the first incorrect word

```
../pyrisc/sim/snurisc.py       -l 1 bmpresize
Loading file bmpresize
Execution completed
Registers
=========
zero ($0): 0x00000000    ra ($1):   0x80000044    sp ($2):   0x80017ffc    gp ($3):   0x00000000
tp ($4):   0x00000000    t0 ($5):   0x00000000    t1 ($6):   0x00000000    t2 ($7):   0x00000000
s0 ($8):   0x00000000    s1 ($9):   0x00000000    a0 ($10):  0x8001002c    a1 ($11):  0x00000004
a2 ($12):  0x00000004    a3 ($13):  0x00000001    a4 ($14):  0x80018000    a5 ($15):  0x00000000
a6 ($16):  0x00000000    a7 ($17):  0x00000000    s2 ($18):  0x80010000    s3 ($19):  0x80010010
s4 ($20):  0x80018000    s5 ($21):  0x800100c0    s6 ($22):  0x800100d0    s7 ($23):  0x00000000
s8 ($24):  0x02020302    s9 ($25):  0x00000000    s10 ($26): 0x00000000    s11 ($27): 0x00000000
t3 ($28):  0x00000000    t4 ($29):  0x00000000    t5 ($30):  0x80018000    t6 ($31):  0x00000001
25 instructions executed in 25 cycles. CPI = 1.000
Data transfer:    9 instructions (36.00%)
ALU operation:    11 instructions (44.00%)
Control transfer: 5 instructions (20.00%)
```

failed to pass test 1

Your program failed to match the value output at memory address 0x80018000

Debugging Tips (1)

- If you want to see the values of registers after the specific instruction, insert 'ebreak' to stop the simulator

```
    .globl bmpresize
bmpresize:
    addi a0, zero, 1  # a0 = 1
    ebreak
    addi a0, zero, 2  # a0 = 2
    ret
```

```
../pyrisc/sim/snurisc.py       -l 1 bmpresize
Loading file bmpresize
Execution completed
Registers
=========
zero ($0): 0x00000000    ra ($1):   0x80000044    sp ($2):   0x80017ffc    gp ($3):   0x00000000
tp ($4):   0x00000000    t0 ($5):   0x00000000    t1 ($6):   0x00000000    t2 ($7):   0x00000000
s0 ($8):   0x00000000    s1 ($9):   0x00000000    a0 ($10):  0x00000001    a1 ($11):  0x00000004
a2 ($12):  0x00000004    a3 ($13):  0x00000001    a4 ($14):  0x80018000    a5 ($15):  0x00000000
a6 ($16):  0x00000000    a7 ($17):  0x00000000    s2 ($18):  0x80010000    s3 ($19):  0x80010010
s4 ($20):  0x00000000    s5 ($21):  0x00000000    s6 ($22):  0x00000000    s7 ($23):  0x00000000
s8 ($24):  0x00000000    s9 ($25):  0x00000000    s10 ($26): 0x00000000    s11 ($27): 0x00000000
t3 ($28):  0x00000000    t4 ($29):  0x00000000    t5 ($30):  0x00000000    t6 ($31):  0x00000001
18 instructions executed in 18 cycles. CPI = 1.000
Data transfer:    5 instructions (27.78%)
ALU operation:    10 instructions (55.56%)
Control transfer: 3 instructions (16.67%)
```

a0 is 1, not 2
Instructions after ebreak are not executed

Debugging Tips (2)

- You can change the log level by changing the number of **`PYRISCOPT`** in `ca-pa3/Makefile`

```
# in ca-pa3/Makefile
…

PYRISC = /dir1/dir2/pyrisc/sim/snurisc.py
PYRISCOPT = -l 1

…
```

Change this number

0: shows no output message
1: dumps registers at the end of the execution (default)
2: dumps registers and data memory at the end of the execution
3: 2 + shows instruction executed in each cycle
4: 3 + shows full information for each instruction
5: 4 + dumps registers for each cycle
6: 5 + dumps data memory for each cycle

Debugging Tips (3)

- You can add another option(-c) to **PYRISCOPT**

```
# in pa3/Makefile
…

PYRISC = /dir1/dir2/pyrisc/sim/snurisc.py
PYRISCOPT = -l 3 -c m

…
```

Shows logs after cycle m (default: 0)
Note that it is only effective for log level 3 or 4

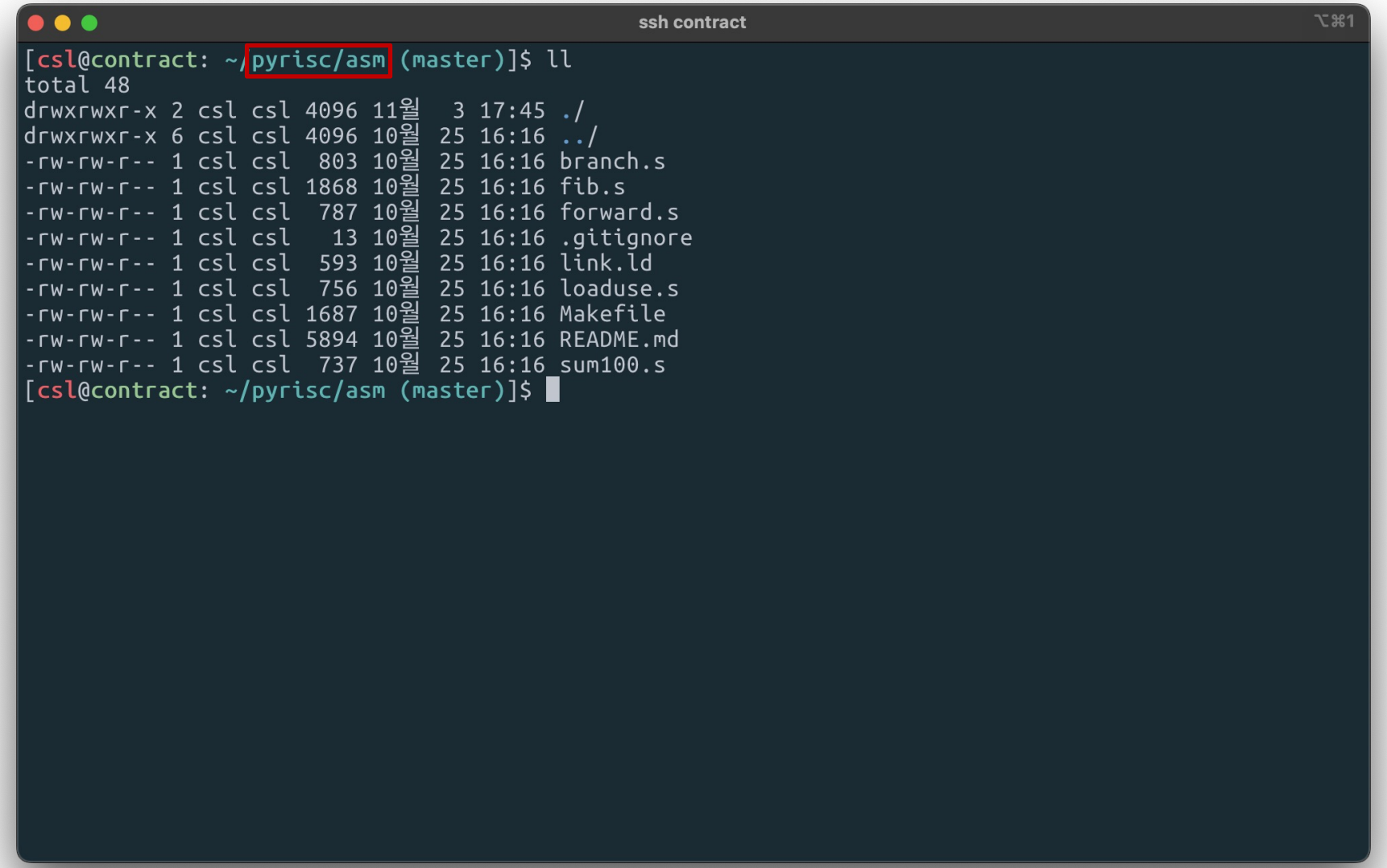
RISC-V Executable File – Code Size

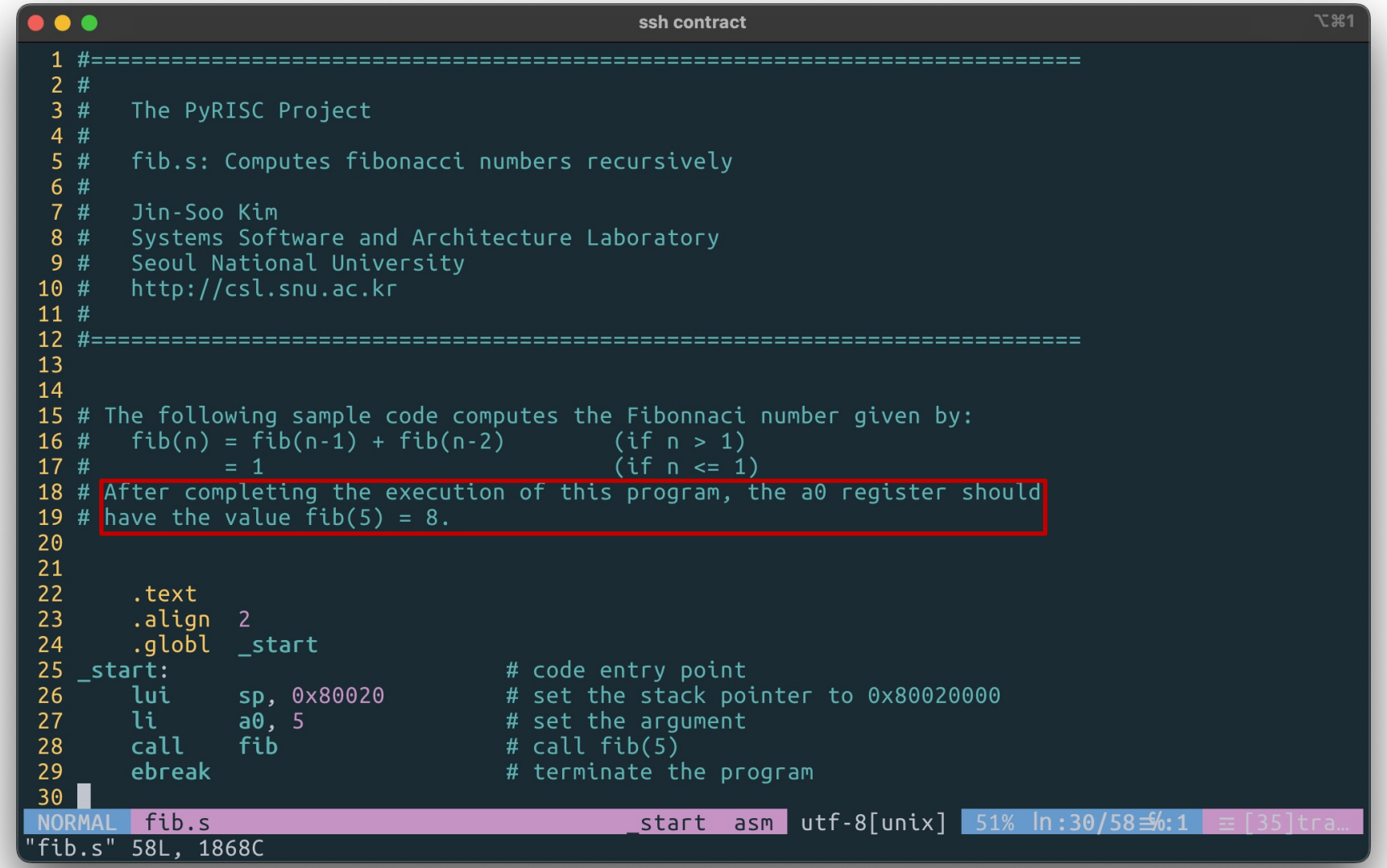
- The code size is measured by the total number of bytes for the text section of *bmpresize.s*

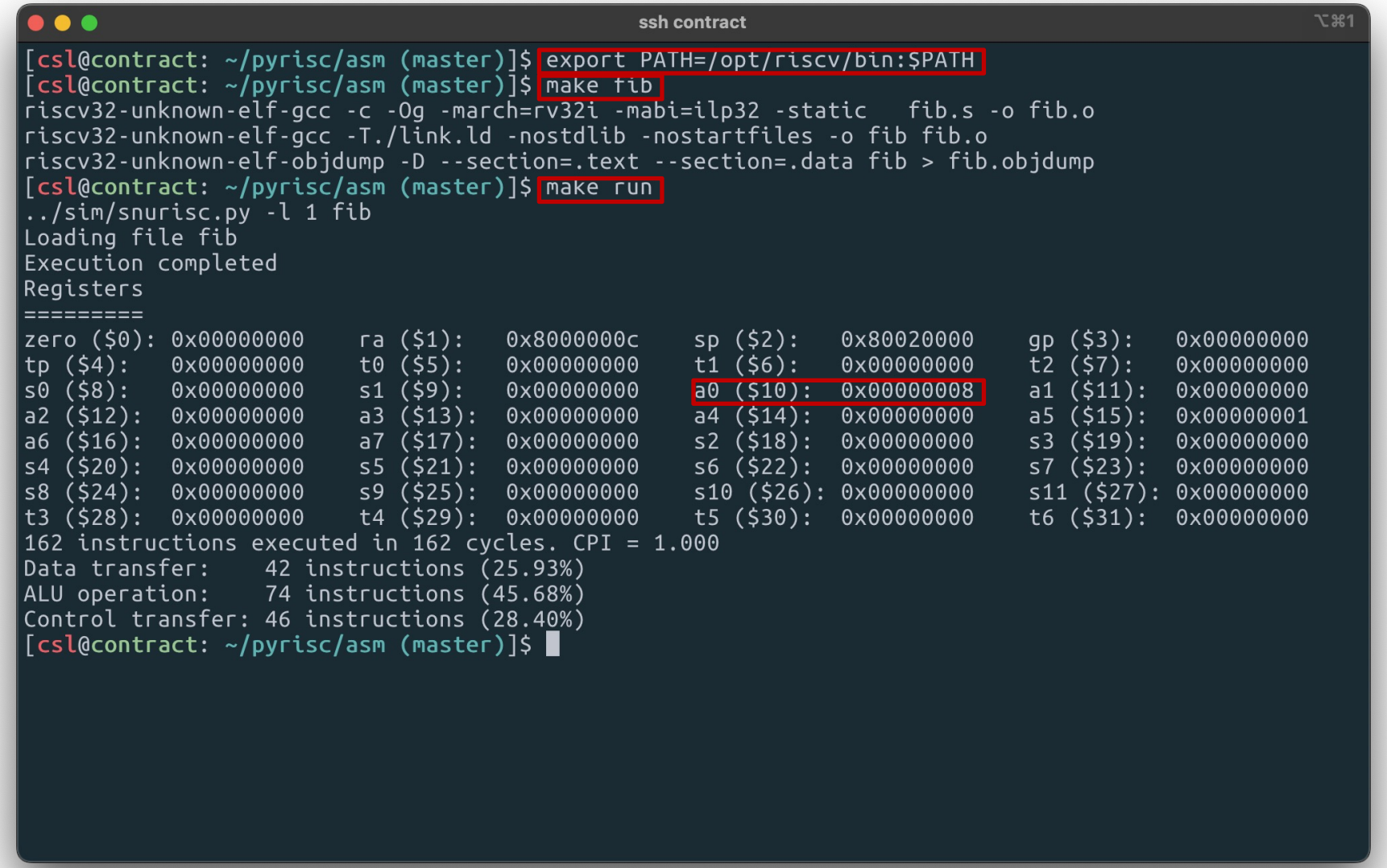
```
$ riscv32-unknown-elf-size bmpresize.o
```

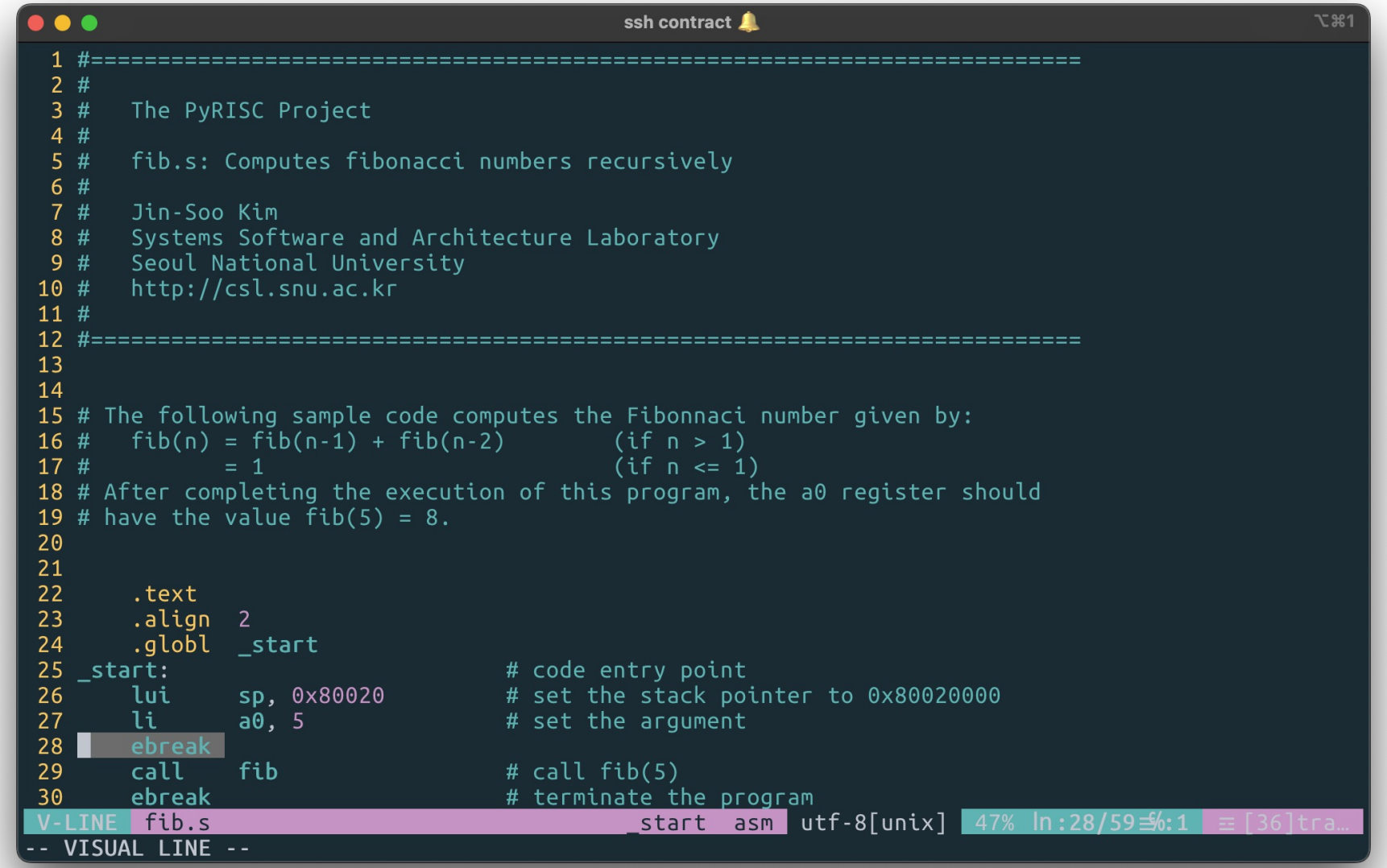
```
text    data     bss     dec     hex filename
 252       0       0     252      fc bmpresize.o
```

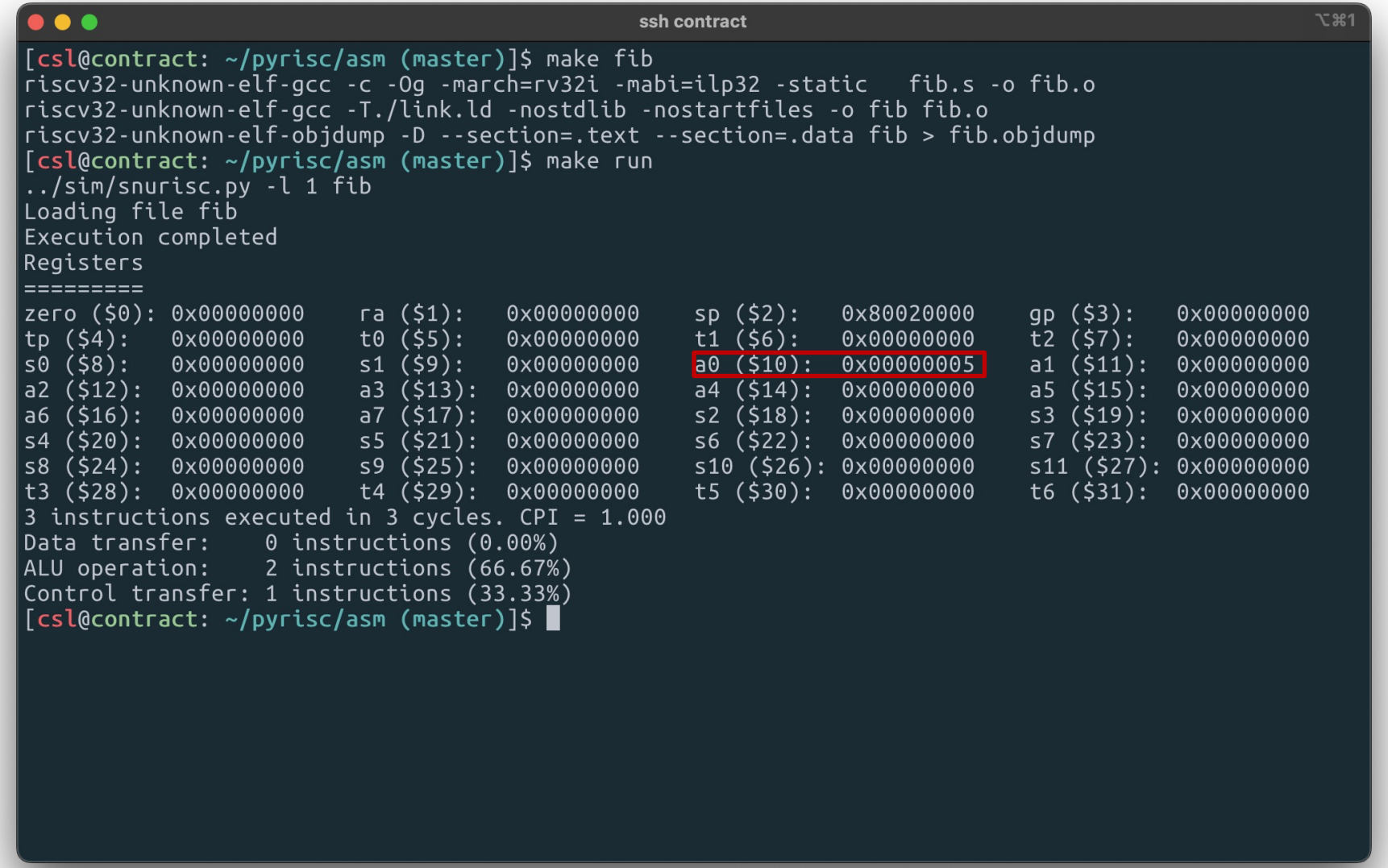
PyRISC Examples

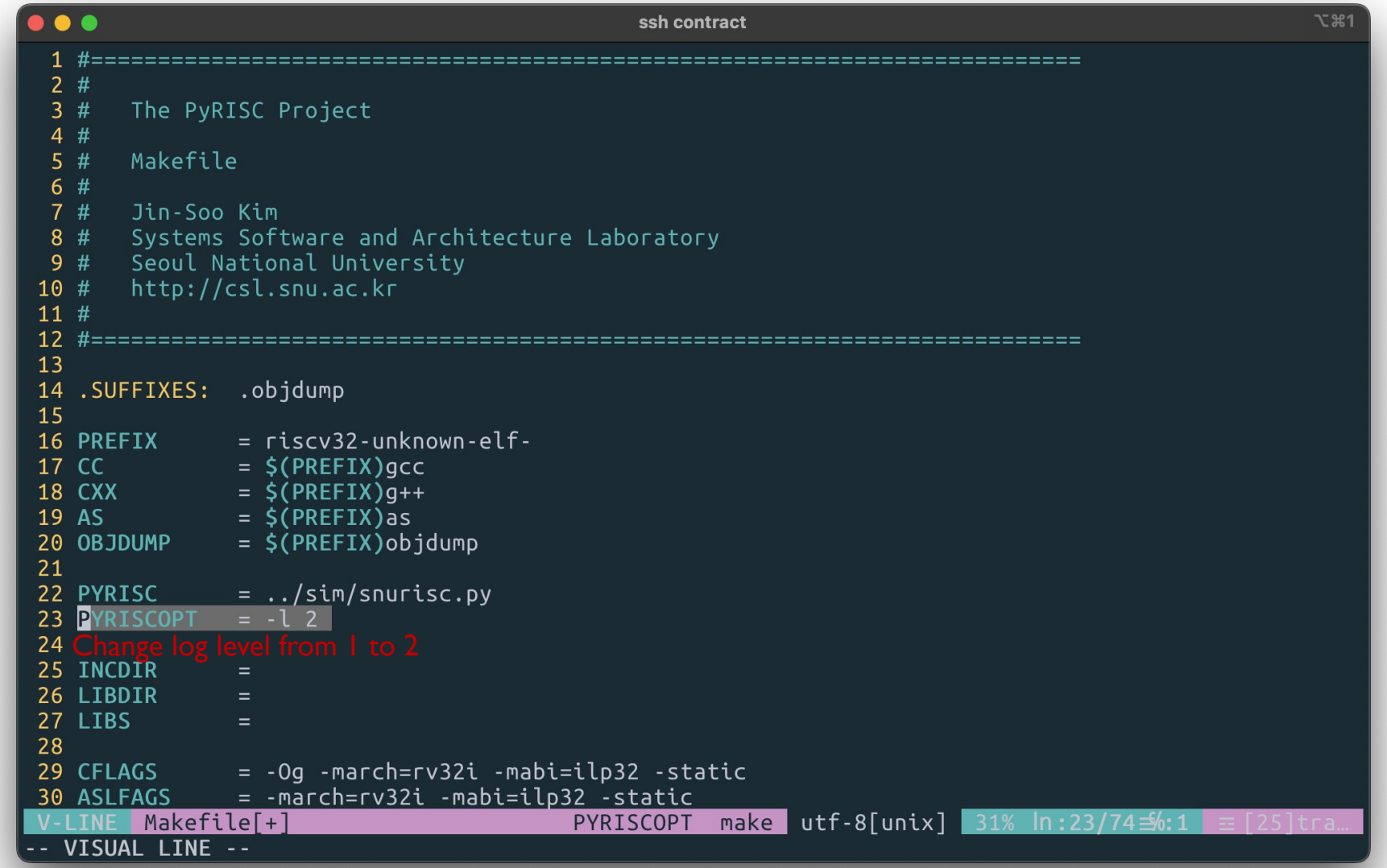

ssh contract
⌥⌘1
[csl@contract: ~/pyrisc/asm (master)]$ ll
total 48
drwxrwxr-x 2 csl csl 4096 11월 3 17:45 ./
drwxrwxr-x 6 csl csl 4096 10월 25 16:16 ../
-rw-rw-r-- 1 csl csl 803 10월 25 16:16 branch.s
-rw-rw-r-- 1 csl csl 1868 10월 25 16:16 fib.s
-rw-rw-r-- 1 csl csl 787 10월 25 16:16 forward.s
-rw-rw-r-- 1 csl csl   13 10월 25 16:16 .gitignore
-rw-rw-r-- 1 csl csl 593 10월 25 16:16 link.ld
-rw-rw-r-- 1 csl csl 756 10월 25 16:16 loaduse.s
-rw-rw-r-- 1 csl csl 1687 10월 25 16:16 Makefile
-rw-rw-r-- 1 csl csl 5894 10월 25 16:16 README.md
-rw-rw-r-- 1 csl csl 737 10월 25 16:16 sum100.s
[csl@contract: ~/pyrisc/asm (master)]$

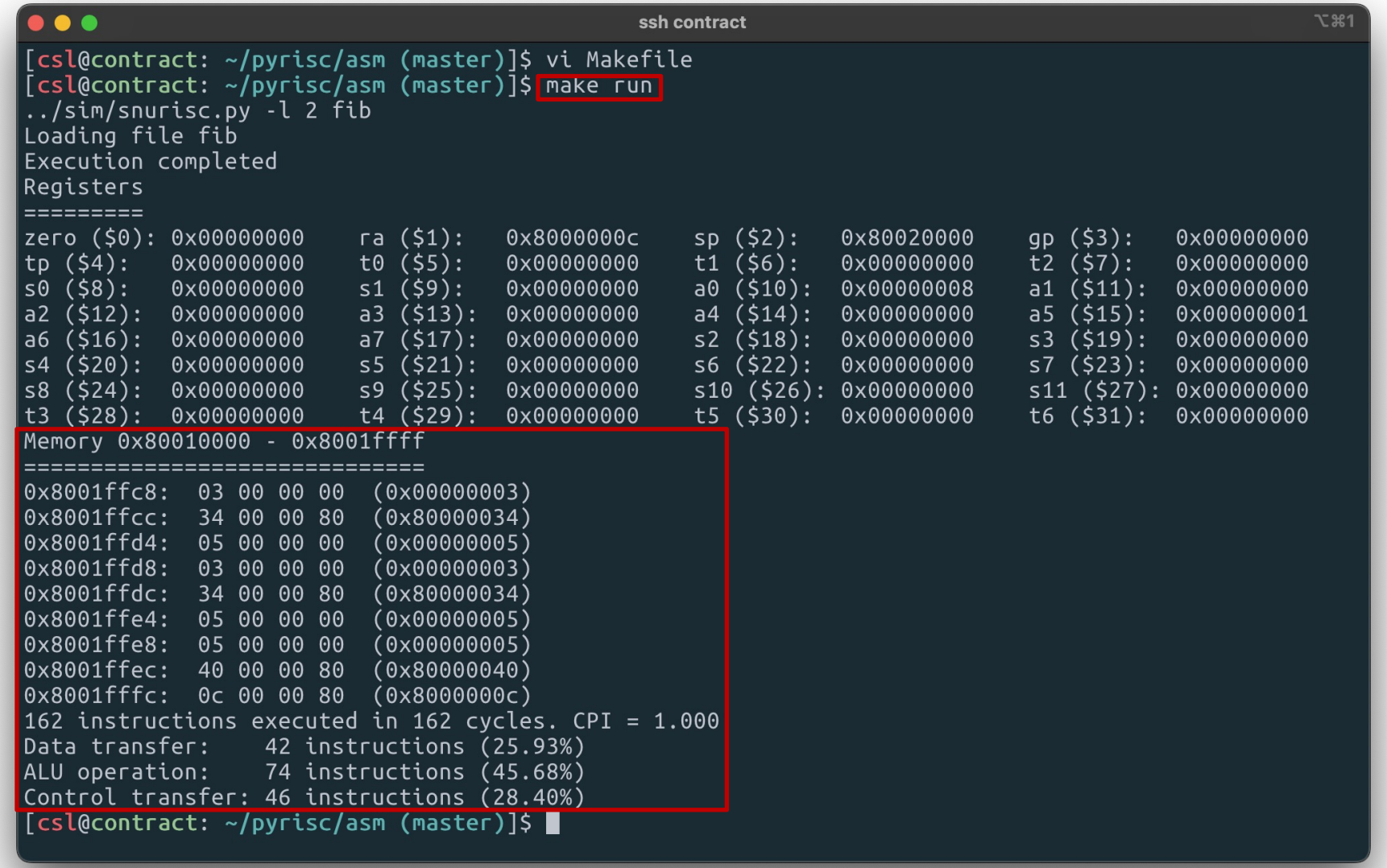

ssh contract
⌥⌘1
1 #===
2 #
3 #   The PyRISC Project
4 #
5 #   fib.s: Computes fibonacci numbers recursively
6 #
7 #   Jin-Soo Kim
8 #   Systems Software and Architecture Laboratory
9 #   Seoul National University
10 #   http://csl.snu.ac.kr
11 #
12 #===
13
14
15 # The following sample code computes the Fibonnaci number given by:
16 #   fib(n) = fib(n-1) + fib(n-2)   (if n > 1)
17 #   = 1   (if n <= 1)
18 # After completing the execution of this program, the a0 register should
19 # have the value fib(5) = 8.
20
21
22   .text
23   .align 2
24   .globl _start
25 _start: # code entry point
26   lui   sp, 0x80020   # set the stack pointer to 0x80020000
27   li   a0, 5   # set the argument
28   call   fib   # call fib(5)
29   ebreak   # terminate the program
30
NORMAL fib.s _start asm utf-8[unix] 51% ln:30/58 ≡%:1 ≡ [35]tra...
"fib.s" 58L, 1868C


ssh contract
⌥⌘1
[csl@contract: ~/pyrisc/asm (master)]$ export PATH=/opt/riscv/bin:$PATH
[csl@contract: ~/pyrisc/asm (master)]$ make fib
riscv32-unknown-elf-gcc -c -Og -march=rv32i -mabi=ilp32 -static   fib.s -o fib.o
riscv32-unknown-elf-gcc -T./link.ld -nostdlib -nostartfiles -o fib fib.o
riscv32-unknown-elf-objdump -D --section=.text --section=.data fib > fib.objdump
[csl@contract: ~/pyrisc/asm (master)]$ make run
../sim/snurisc.py -l 1 fib
Loading file fib
Execution completed
Registers
=========
zero ($0): 0x00000000 ra ($1): 0x8000000c sp ($2): 0x80020000 gp ($3): 0x00000000
tp ($4):   0x00000000 t0 ($5): 0x00000000 t1 ($6): 0x00000000 t2 ($7): 0x00000000
s0 ($8):   0x00000000 s1 ($9): 0x00000000 a0 ($10): 0x00000008 a1 ($11): 0x00000000
a2 ($12): 0x00000000 a3 ($13): 0x00000000 a4 ($14): 0x00000000 a5 ($15): 0x00000001
a6 ($16): 0x00000000 a7 ($17): 0x00000000 s2 ($18): 0x00000000 s3 ($19): 0x00000000
s4 ($20): 0x00000000 s5 ($21): 0x00000000 s6 ($22): 0x00000000 s7 ($23): 0x00000000
s8 ($24): 0x00000000 s9 ($25): 0x00000000 s10 ($26): 0x00000000 s11 ($27): 0x00000000
t3 ($28): 0x00000000 t4 ($29): 0x00000000 t5 ($30): 0x00000000 t6 ($31): 0x00000000
162 instructions executed in 162 cycles. CPI = 1.000
Data transfer: 42 instructions (25.93%)
ALU operation: 74 instructions (45.68%)
Control transfer: 46 instructions (28.40%)
[csl@contract: ~/pyrisc/asm (master)]$

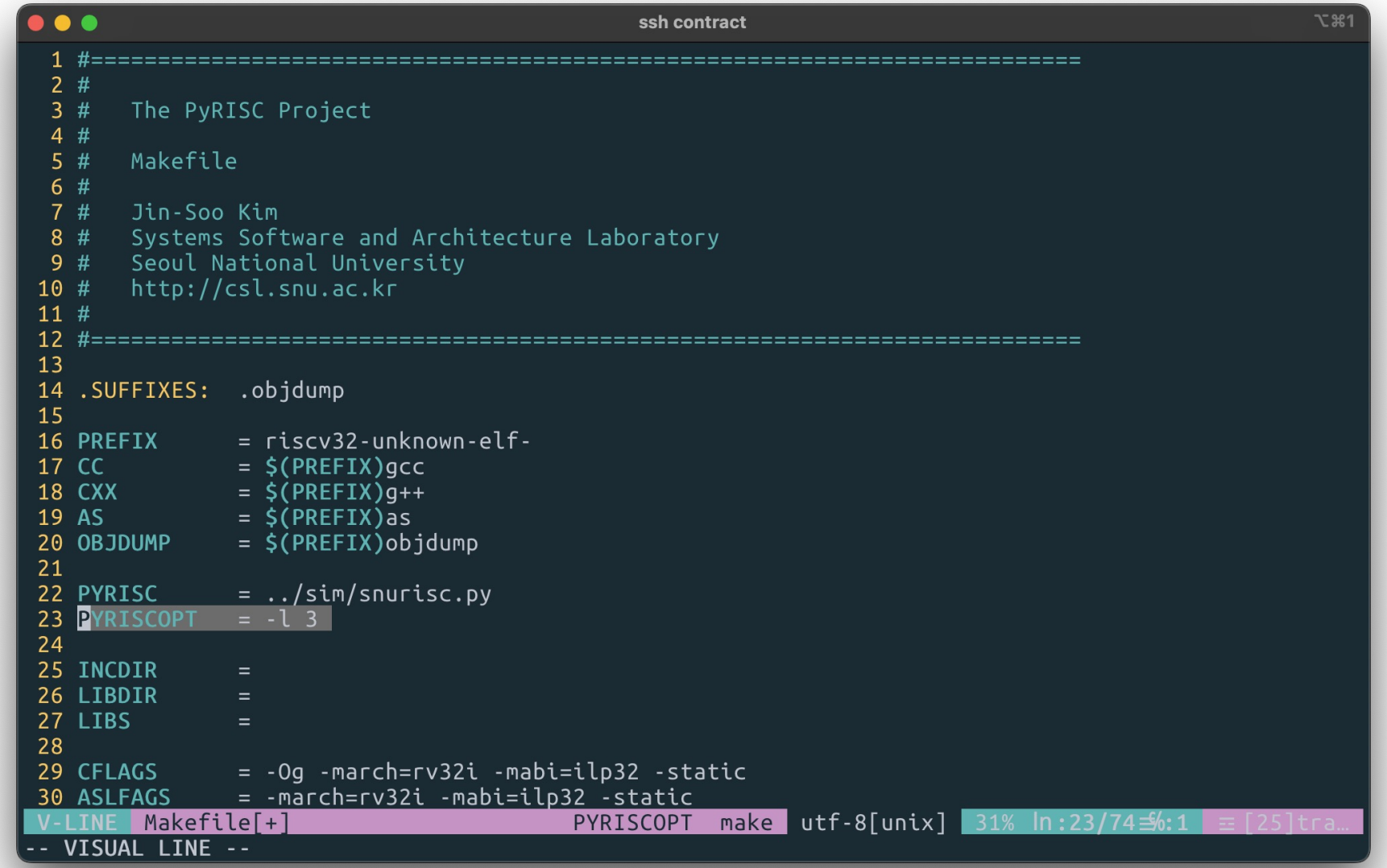

ssh contract
#
The PyRISC Project
#
fib.s: Computes fibonacci numbers recursively
#
Jin-Soo Kim
Systems Software and Architecture Laboratory
Seoul National University
http://csl.snu.ac.kr
#
The following sample code computes the Fibonnaci number given by:
fib(n) = fib(n-1) + fib(n-2)   (if n > 1)
= 1   (if n <= 1)
After completing the execution of this program, the a0 register should
have the value fib(5) = 8.
.text
.align 2
.globl _start
_start: # code entry point
lui sp, 0x80020   # set the stack pointer to 0x80020000
li a0, 5   # set the argument
ebreak
call fib   # call fib(5)
ebreak   # terminate the program
V-LINE fib.s _start asm utf-8[unix] 47% ln:28/59 %:1
-- VISUAL LINE --

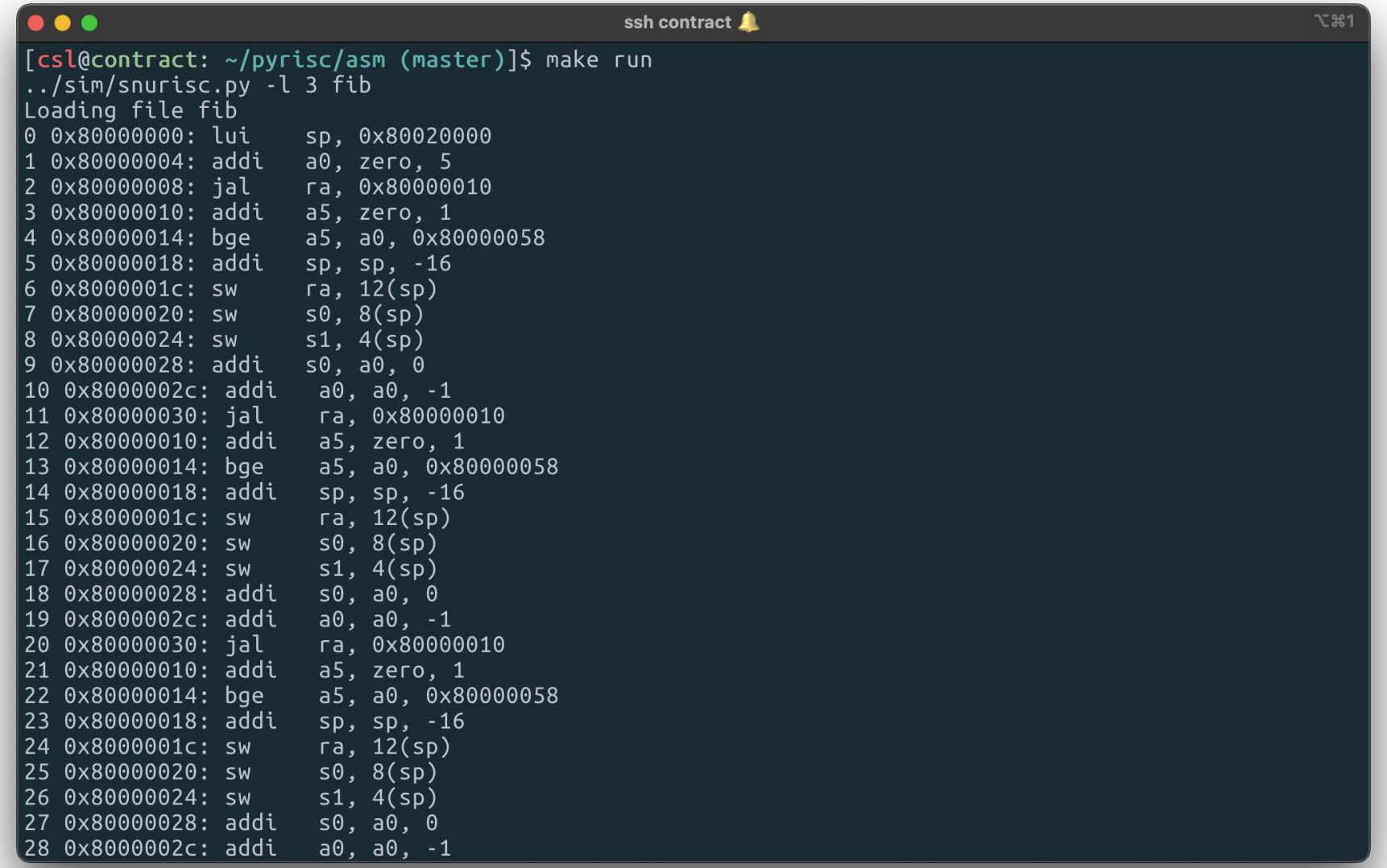

ssh contract
⌥⌘1
[csl@contract: ~/pyrisc/asm (master)]$ make fib
riscv32-unknown-elf-gcc -c -Og -march=rv32i -mabi=ilp32 -static   fib.s -o fib.o
riscv32-unknown-elf-gcc -T./link.ld -nostdlib -nostartfiles -o fib fib.o
riscv32-unknown-elf-objdump -D --section=.text --section=.data fib > fib.objdump
[csl@contract: ~/pyrisc/asm (master)]$ make run
../sim/snurisc.py -l 1 fib
Loading file fib
Execution completed
Registers
=========
zero ($0): 0x00000000 ra ($1): 0x00000000 sp ($2): 0x80020000 gp ($3): 0x00000000
tp ($4):   0x00000000 t0 ($5): 0x00000000 t1 ($6): 0x00000000 t2 ($7): 0x00000000
s0 ($8):   0x00000000 s1 ($9): 0x00000000 a0 ($10): 0x00000005 a1 ($11): 0x00000000
a2 ($12): 0x00000000 a3 ($13): 0x00000000 a4 ($14): 0x00000000 a5 ($15): 0x00000000
a6 ($16): 0x00000000 a7 ($17): 0x00000000 s2 ($18): 0x00000000 s3 ($19): 0x00000000
s4 ($20): 0x00000000 s5 ($21): 0x00000000 s6 ($22): 0x00000000 s7 ($23): 0x00000000
s8 ($24): 0x00000000 s9 ($25): 0x00000000 s10 ($26): 0x00000000 s11 ($27): 0x00000000
t3 ($28): 0x00000000 t4 ($29): 0x00000000 t5 ($30): 0x00000000 t6 ($31): 0x00000000
3 instructions executed in 3 cycles. CPI = 1.000
Data transfer: 0 instructions (0.00%)
ALU operation: 2 instructions (66.67%)
Control transfer: 1 instructions (33.33%)
[csl@contract: ~/pyrisc/asm (master)]$


ssh contract
⌥⌘1
#===
#
The PyRISC Project
#
Makefile
#
Jin-Soo Kim
Systems Software and Architecture Laboratory
Seoul National University
http://csl.snu.ac.kr
#
#===
.SUFFIXES: .objdump
PREFIX   = riscv32-unknown-elf-
CC   = $(PREFIX)gcc
CXX   = $(PREFIX)g++
AS   = $(PREFIX)as
OBJDUMP   = $(PREFIX)objdump
PYRISC   = ../sim/snurisc.py
PYRISCOPT   = -l 2
Change log level from 1 to 2
INCDIR   =
LIBDIR   =
LIBS   =
CFLAGS   = -Og -march=rv32i -mabi=ilp32 -static
ASLFAGS   = -march=rv32i -mabi=ilp32 -static
V-LINE Makefile[+] PYRISCOPT make utf-8[unix] 31% ln:23/74 ≡%:1 ≡ [25]tra…
-- VISUAL LINE --

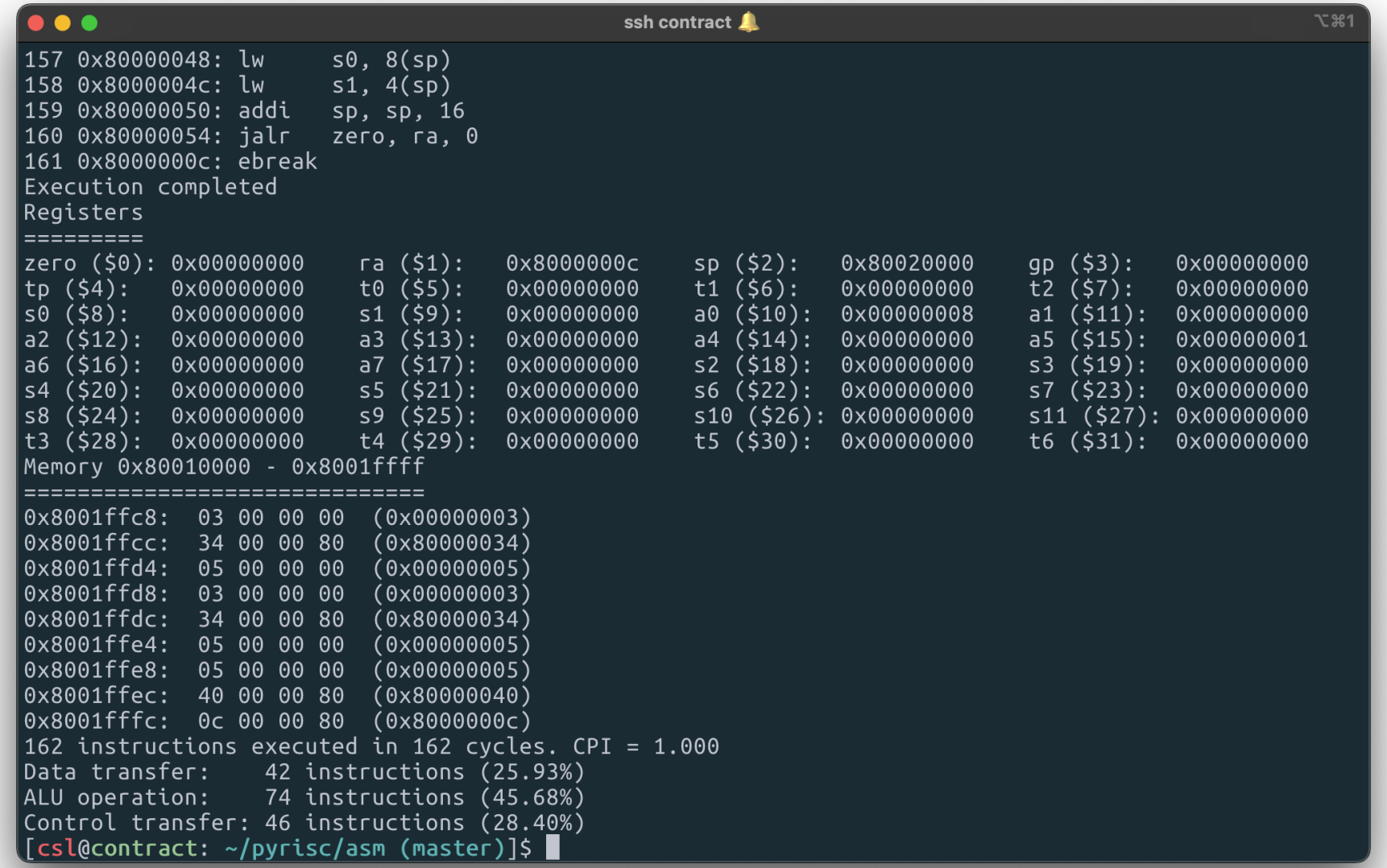


Log level 2: dumps registers and data memory at the end of the execution

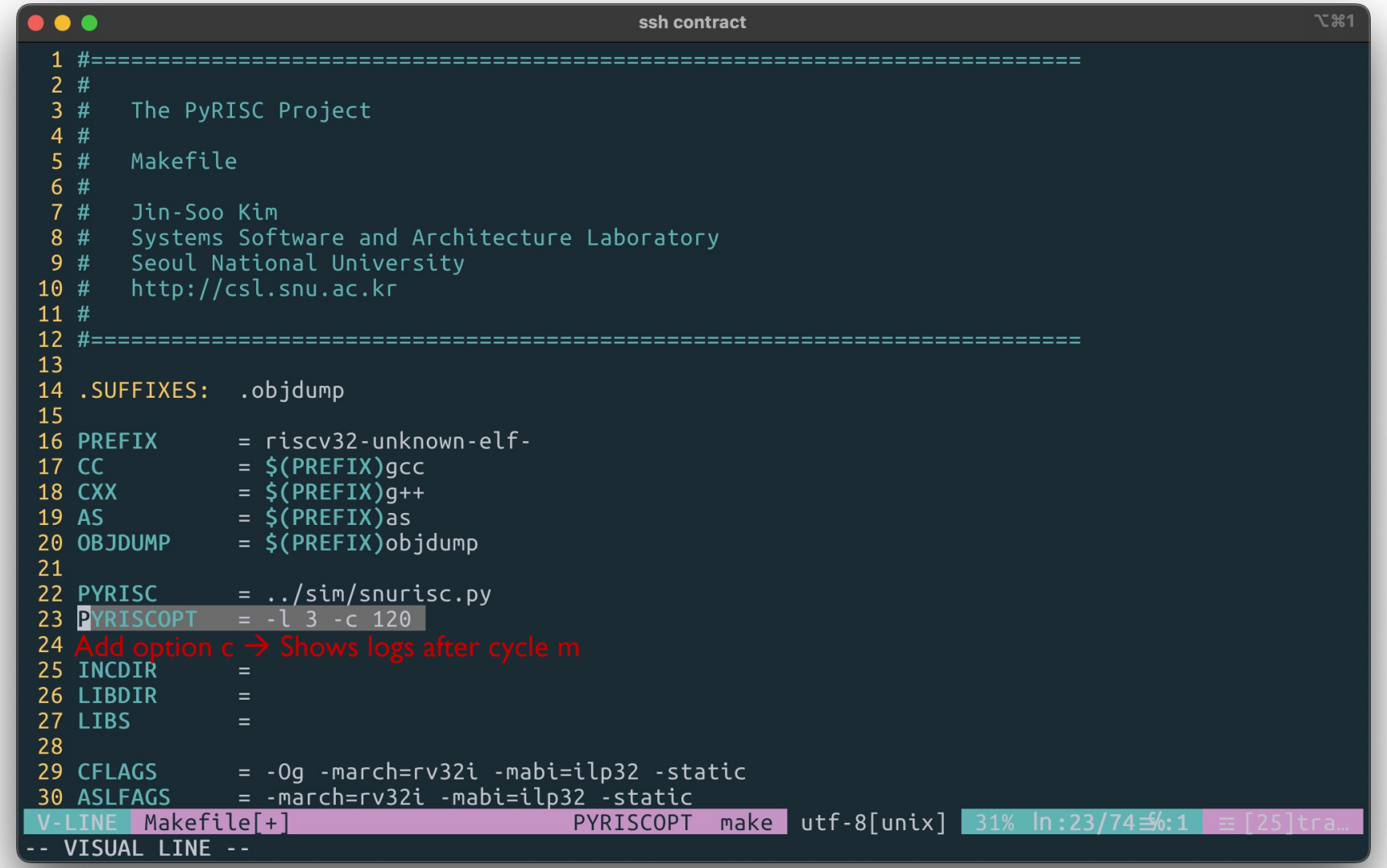

ssh contract
⌥⌘1
#===
#
The PyRISC Project
#
Makefile
#
Jin-Soo Kim
Systems Software and Architecture Laboratory
Seoul National University
http://csl.snu.ac.kr
#
#===
.SUFFIXES: .objdump
PREFIX   = riscv32-unknown-elf-
CC   = $(PREFIX)gcc
CXX   = $(PREFIX)g++
AS   = $(PREFIX)as
OBJDUMP   = $(PREFIX)objdump
PYRISC   = ../sim/snurisc.py
PYRISCOPT   = -l 3
INCDIR   =
LIBDIR   =
LIBS   =
CFLAGS   = -Og -march=rv32i -mabi=ilp32 -static
ASLFAGS   = -march=rv32i -mabi=ilp32 -static
V-LINE Makefile[+] PYRISCOPT make utf-8[unix] 31% ln:23/74 ≡%:1 ≡ [25]tra…
-- VISUAL LINE --

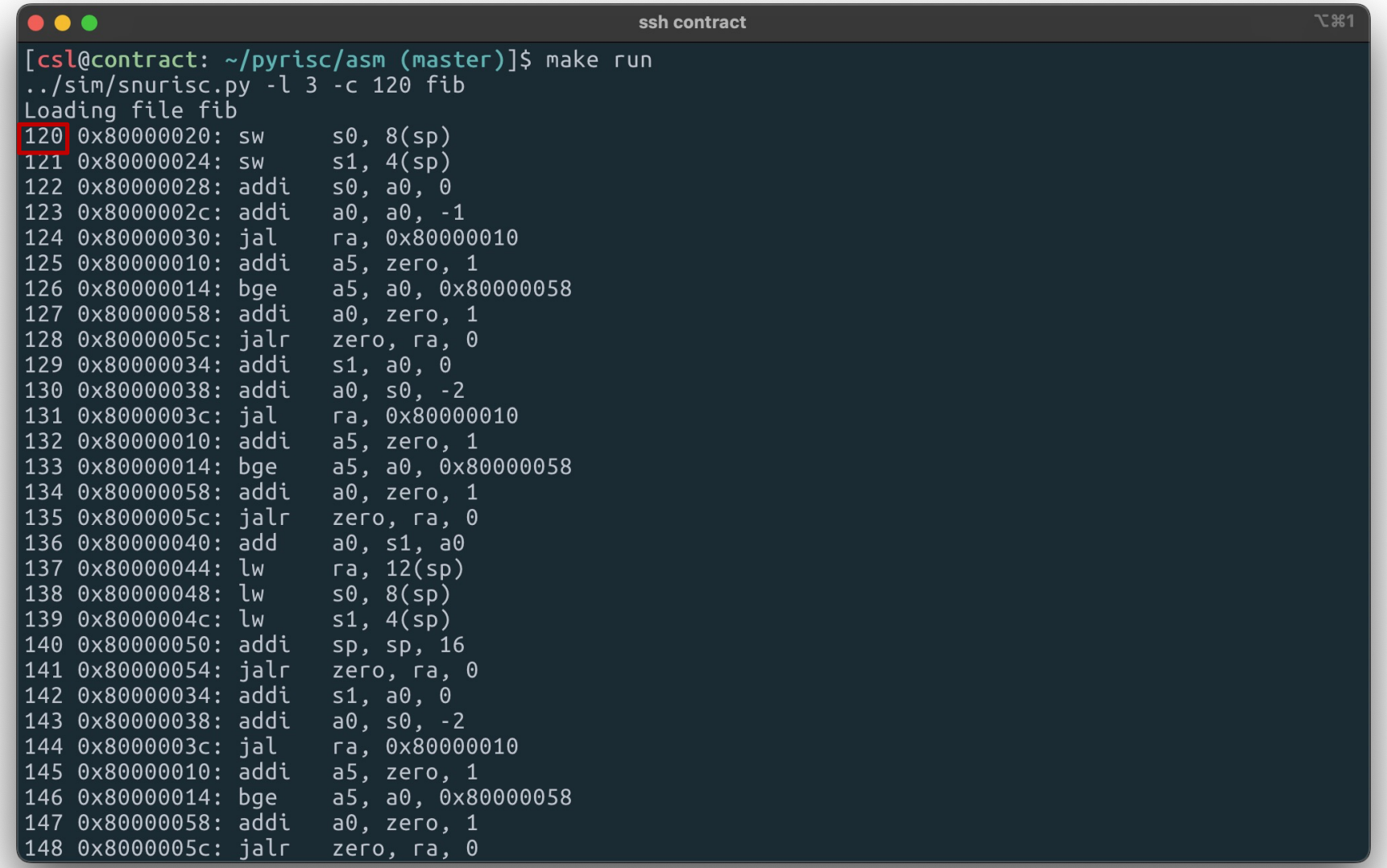


Log level 3: 2 + shows instruction executed in each cycle



Log level 3: 2 + shows instruction executed in each cycle


ssh contract
⌥⌘1
#===
#
The PyRISC Project
#
Makefile
#
Jin-Soo Kim
Systems Software and Architecture Laboratory
Seoul National University
http://csl.snu.ac.kr
#
#===
.SUFFIXES: .objdump
PREFIX   = riscv32-unknown-elf-
CC   = $(PREFIX)gcc
CXX   = $(PREFIX)g++
AS   = $(PREFIX)as
OBJDUMP   = $(PREFIX)objdump
PYRISC   = ../sim/snurisc.py
PYRISCOPT   = -l 3 -c 120
Add option c → Shows logs after cycle m
INCDIR   =
LIBDIR   =
LIBS   =
CFLAGS   = -Og -march=rv32i -mabi=ilp32 -static
ASLFAGS   = -march=rv32i -mabi=ilp32 -static
V-LINE Makefile[+] PYRISCOPT make utf-8[unix] 31% ln:23/74 ≡%:1 ≡ [25]tra…
-- VISUAL LINE --


ssh contract
⌥⌘1
[csl@contract: ~/pyrisc/asm (master)]$ make run
../sim/snurisc.py -l 3 -c 120 fib
Loading file fib
120 0x80000020: sw s0, 8(sp)
121 0x80000024: sw s1, 4(sp)
122 0x80000028: addi s0, a0, 0
123 0x8000002c: addi a0, a0, -1
124 0x80000030: jal ra, 0x80000010
125 0x80000010: addi a5, zero, 1
126 0x80000014: bge a5, a0, 0x80000058
127 0x80000058: addi a0, zero, 1
128 0x8000005c: jalr zero, ra, 0
129 0x80000034: addi s1, a0, 0
130 0x80000038: addi a0, s0, -2
131 0x8000003c: jal ra, 0x80000010
132 0x80000010: addi a5, zero, 1
133 0x80000014: bge a5, a0, 0x80000058
134 0x80000058: addi a0, zero, 1
135 0x8000005c: jalr zero, ra, 0
136 0x80000040: add a0, s1, a0
137 0x80000044: lw ra, 12(sp)
138 0x80000048: lw s0, 8(sp)
139 0x8000004c: lw s1, 4(sp)
140 0x80000050: addi sp, sp, 16
141 0x80000054: jalr zero, ra, 0
142 0x80000034: addi s1, a0, 0
143 0x80000038: addi a0, s0, -2
144 0x8000003c: jal ra, 0x80000010
145 0x80000010: addi a5, zero, 1
146 0x80000014: bge a5, a0, 0x80000058
147 0x80000058: addi a0, zero, 1
148 0x8000005c: jalr zero, ra, 0

Thank You!

- Don't forget to read the detailed description before you start your assignment

- If you have any questions about the assignment, feel free to ask via KakaoTalk

- This file will be uploaded after the lab session☺