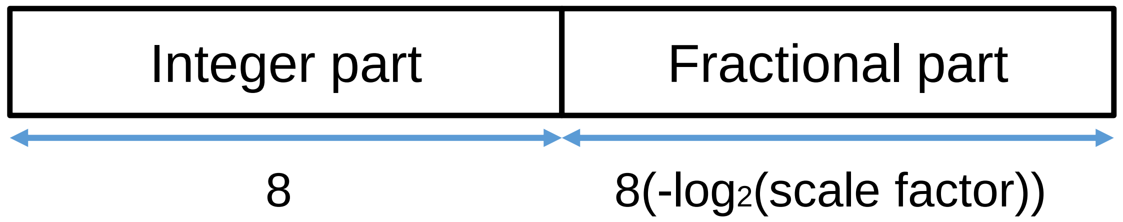
Term Project #2: Fixed-Point MV Multiply (Optional)

05/31/2018

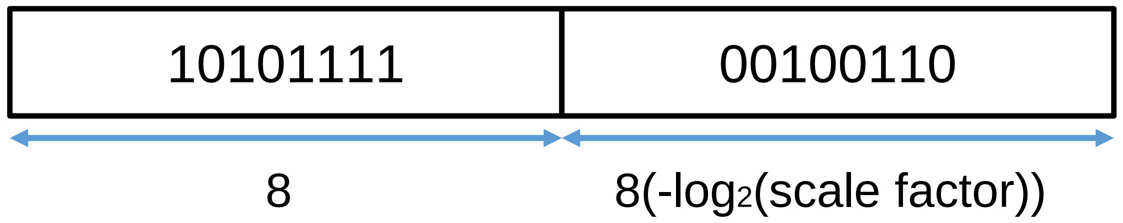
4190.309A: Hardware System Design
(Spring 2018)

Specifications of Term Project #2

- Fixed-point representation of a number
  - Example : 16-bit fixed point number (Scale factor : 1/256)



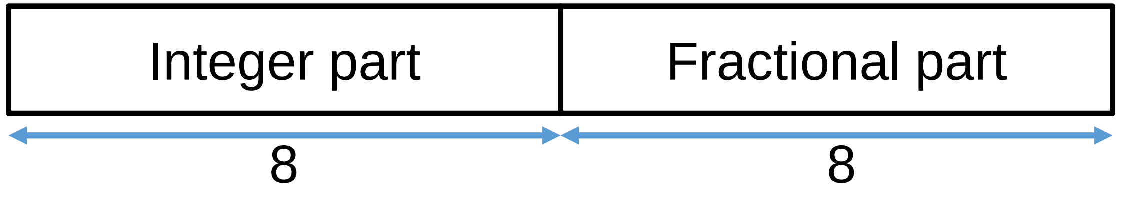
  - Representing 10101111.00100110 (Binary)



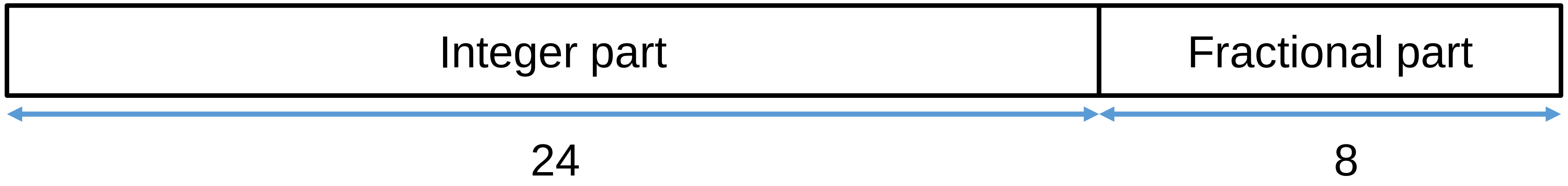
  - Stored as 0xAF26 (=1010111100100110) in memory.

Specifications of Term Project #2

- Input/output data format
  - Input : 16-bit Fixed point (Scale factor : 1/256)



  - Output : 32-bit Fixed point (Scale factor : 1/256)



  - There is no zero-padding for this project.
  - You don’t need to care for endianness in this project

Specifications of Term Project #2

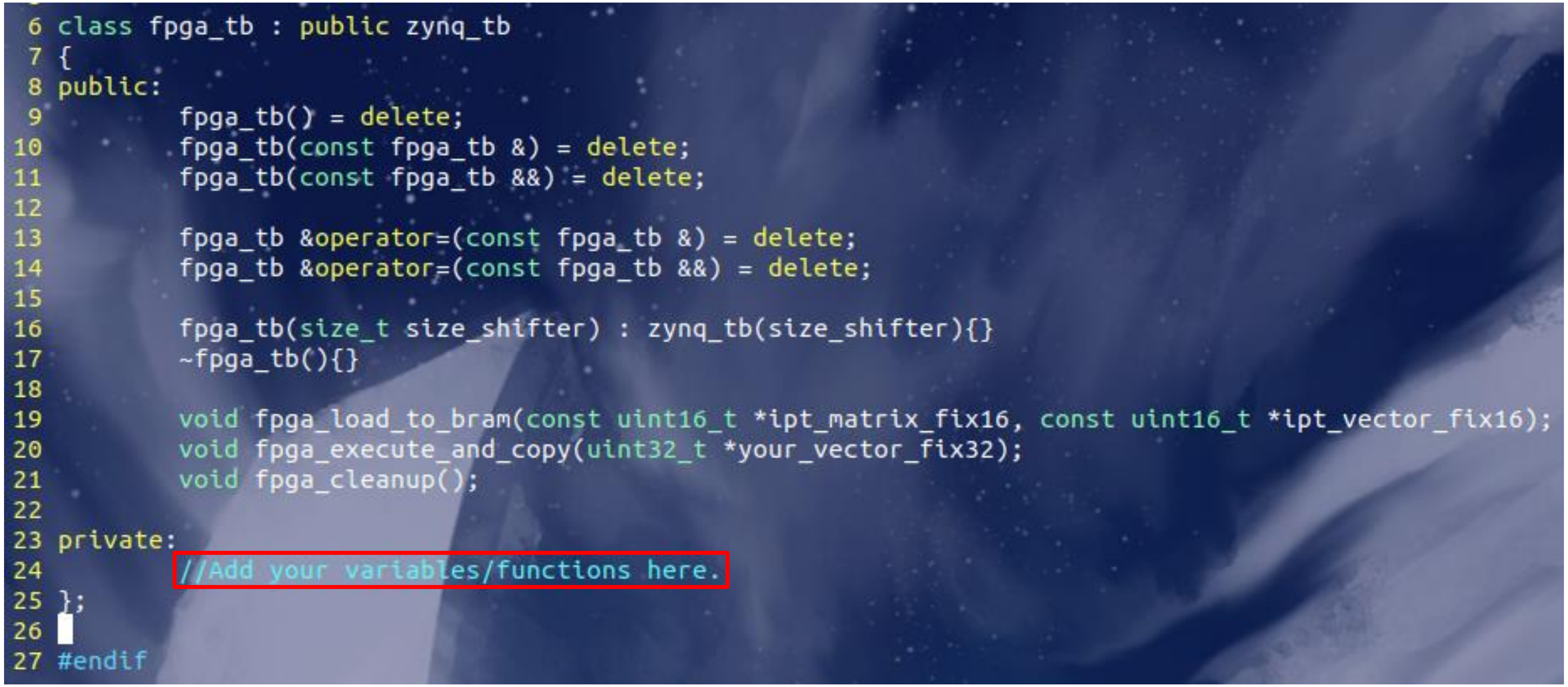
- Hardware files
  - Template: Freely use Verilog files from Lab8-2 / Term #1 / etc.
  - You may use Xilinx integer MAC IP.
  - You may have to modify the block design to use DMA.
    - Refer to Lab6 to learn about block design.

Specifications of Term Project #2

- Software files
  - Templates : `main.cpp zynq.cpp zynq.h Makefile`
  - Files you have to submit : `fpga.cpp fpga.h`
  - Input generator : `generator.c`
  - Input file : `input.txt`
- Add your variables to `fpga_tb` (at `fpga.h`) class
  - As private variables
  - Or you may add it as global variables in `fpga.cpp`
- Use software-implemented code as reference
  - `zynq.cpp:170 zynq_tb::arm_calculate()`

Specifications of Term Project #2

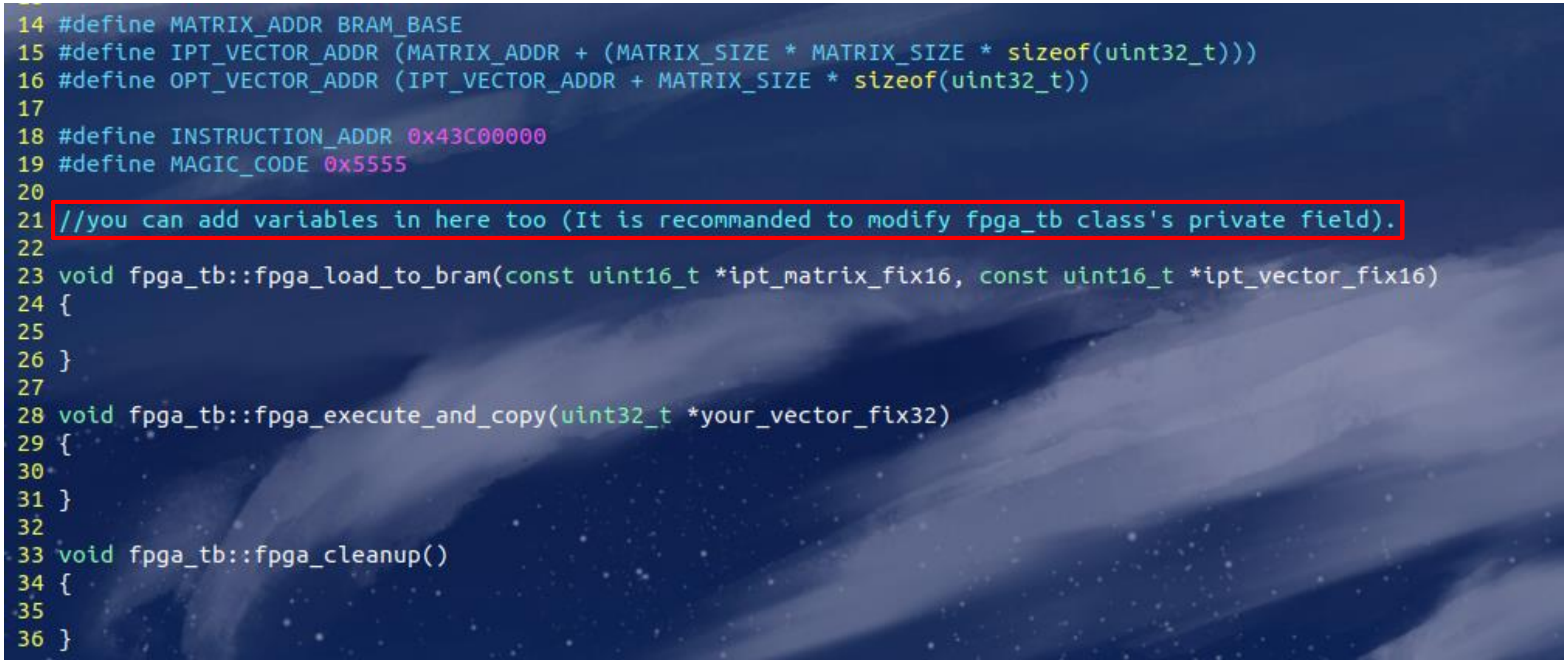
- Add your variables to the private section of `fpga_tb` class.



```
class fpga_tb : public zynq_tb
{
public:
        fpga_tb() = delete;
        fpga_tb(const fpga_tb &) = delete;
        fpga_tb(const fpga_tb &&) = delete;

        fpga_tb &operator=(const fpga_tb &) = delete;
        fpga_tb &operator=(const fpga_tb &&) = delete;

        fpga_tb(size_t size_shifter) : zynq_tb(size_shifter){}
        ~fpga_tb(){}

        void fpga_load_to_bram(const uint16_t *ipt_matrix_fix16, const uint16_t *ipt_vector_fix16);
        void fpga_execute_and_copy(uint32_t *your_vector_fix32);
        void fpga_cleanup();

private:
        //Add your variables/functions here.
};

#endif
```

Specifications of Term Project #2

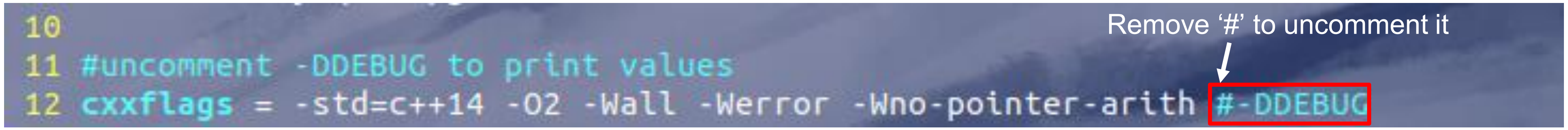
- Add your variables to the private section of `fpga_tb` class.
  - Or you may add it as global (static) variables in `fpga.cpp`



```
#define MATRIX_ADDR BRAM_BASE
#define IPT_VECTOR_ADDR (MATRIX_ADDR + (MATRIX_SIZE * MATRIX_SIZE * sizeof(uint32_t)))
#define OPT_VECTOR_ADDR (IPT_VECTOR_ADDR + MATRIX_SIZE * sizeof(uint32_t))

#define INSTRUCTION_ADDR 0x43C00000
#define MAGIC_CODE 0x5555

//you can add variables in here too (It is recommanded to modify fpga_tb class's private field).

void fpga_tb::fpga_load_to_bram(const uint16_t *ipt_matrix_fix16, const uint16_t *ipt_vector_fix16)
{

}

void fpga_tb::fpga_execute_and_copy(uint32_t *your_vector_fix32)
{

}

void fpga_tb::fpga_cleanup()
{

}
```

Specifications of Term Project #2

- Compile and run your software code
  - Compile : `$> sudo make`
  - Run : `$> sudo ./project2`
- Using input generator
  - `$> ./input_generator`
  - Generator will write data to `input.txt`
- Test will be done for 65536 random matrix-vector.
  - In `Makefile`, uncomment `–DDEBUG` to test with `input.txt`.
  - Recompile(`$> make clean ; make`) after modifying flag.



```

#uncomment -DDEBUG to print values
cxxflags = -std=c++14 -O2 -Wall -Werror -Wno-pointer-arith #-DDEBUG
```

Specifications of Term Project #2

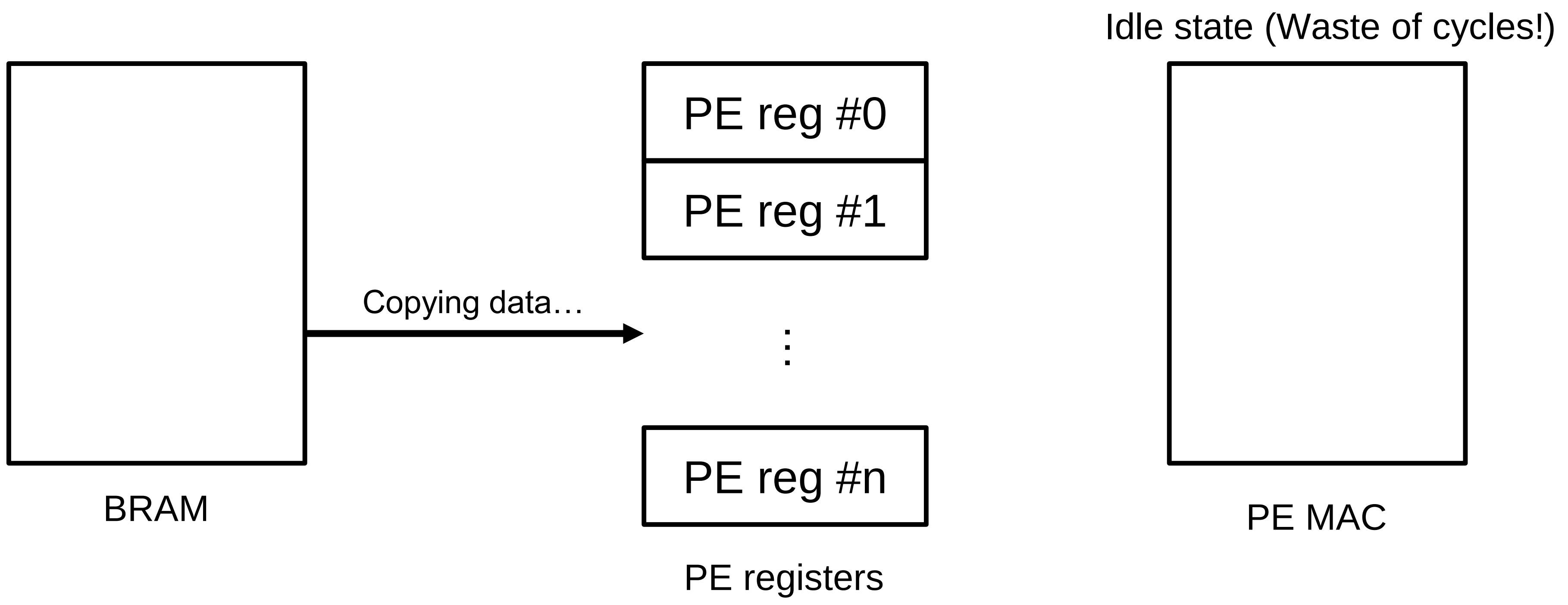
- **Due: 6/16 (Thursday class), 6/17 (Friday class)**

- Scoring policy
  - This project is totally optional.
  - You can get up to 25% more points for the term project (125 instead of 100 for full credits).
  - Note that the term project accounts for 20% of your total grade.

- Grade boost!
  - Top 2 teams will get a grade boost (B+ → A-, A0 → A+).
  - Only those teams who get full credits will be considered (no correctness issue).
  - Performance will be the tie breaker.

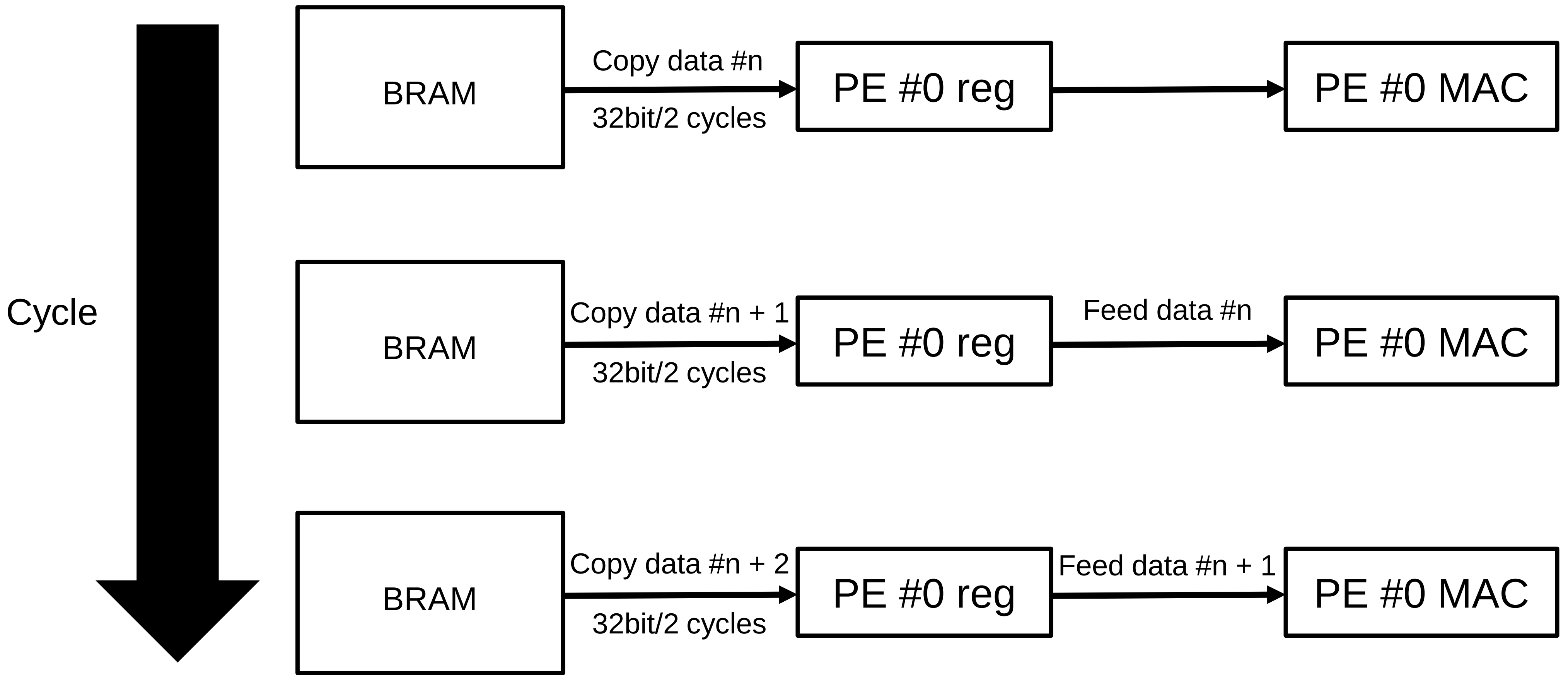
Tips for optimization

- Problem of naïve implementation
  - MAC suffers from data starvation while data is copied into registers.



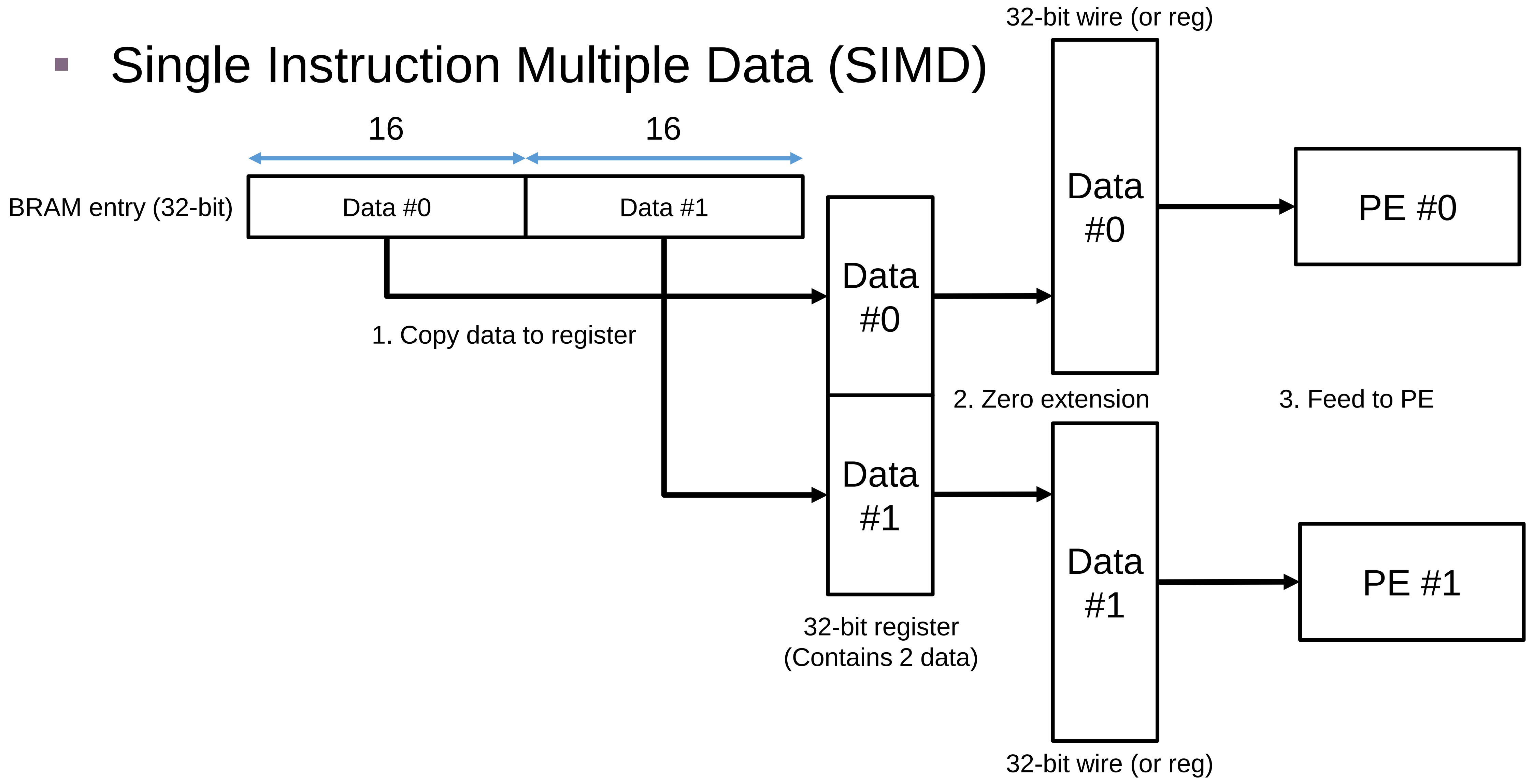
Tips for optimization

- Pipelining



Tips for optimization

- Single Instruction Multiple Data (SIMD)



Tips for optimization

- Bit width of AXI bus (32 bit) is bottleneck.
  - To feed 64 PEs in a cycle, BRAM should fetch 16 * 64 = 1024 bit / cycle.
  - Maximum theoretical utilization : 32 / 1024 = 3.125%
  - Waste of FPGA resources.
- Solution
  - ~~Expend bus's bit width (Hardware modification is needed)~~
  - Use resource efficiently (2 PEs are enough for 100% utilization).
    - Using less area leads to reduced cost in ASIC fabrication(production).