# Verilog Lab3

# Single Cycle CPU with a Cache

# Deadline

12/9 23:59

# Testcase

- You may change the test pattern by executing Python in the "./code/Gen_pattern"
  - I1: factorial
  - I2: arithmetic operations
  - I3: insertion sort
  - I4: bubble sort
- All assembly code finishes with **ECALL** (00000073)
- All test cases should finish in 10000 cycles.

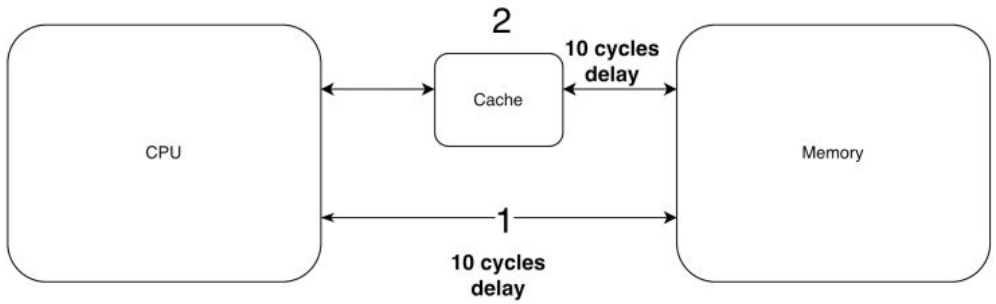
# Implementation Outline - Single Cycle CPU with a cache

You need to implement

- CPU
- Cache

You don't need to implement

- Memories
- Regfile



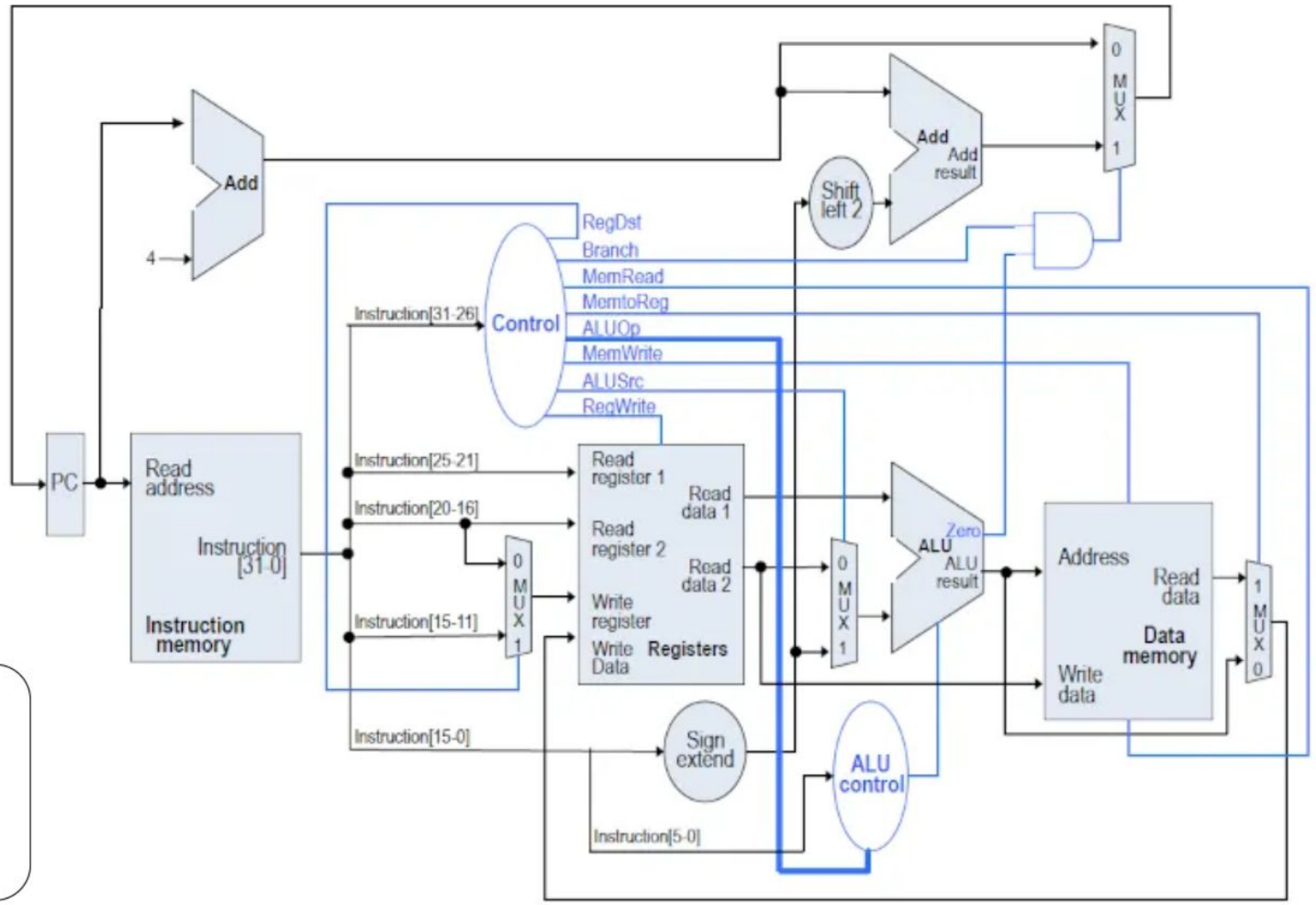
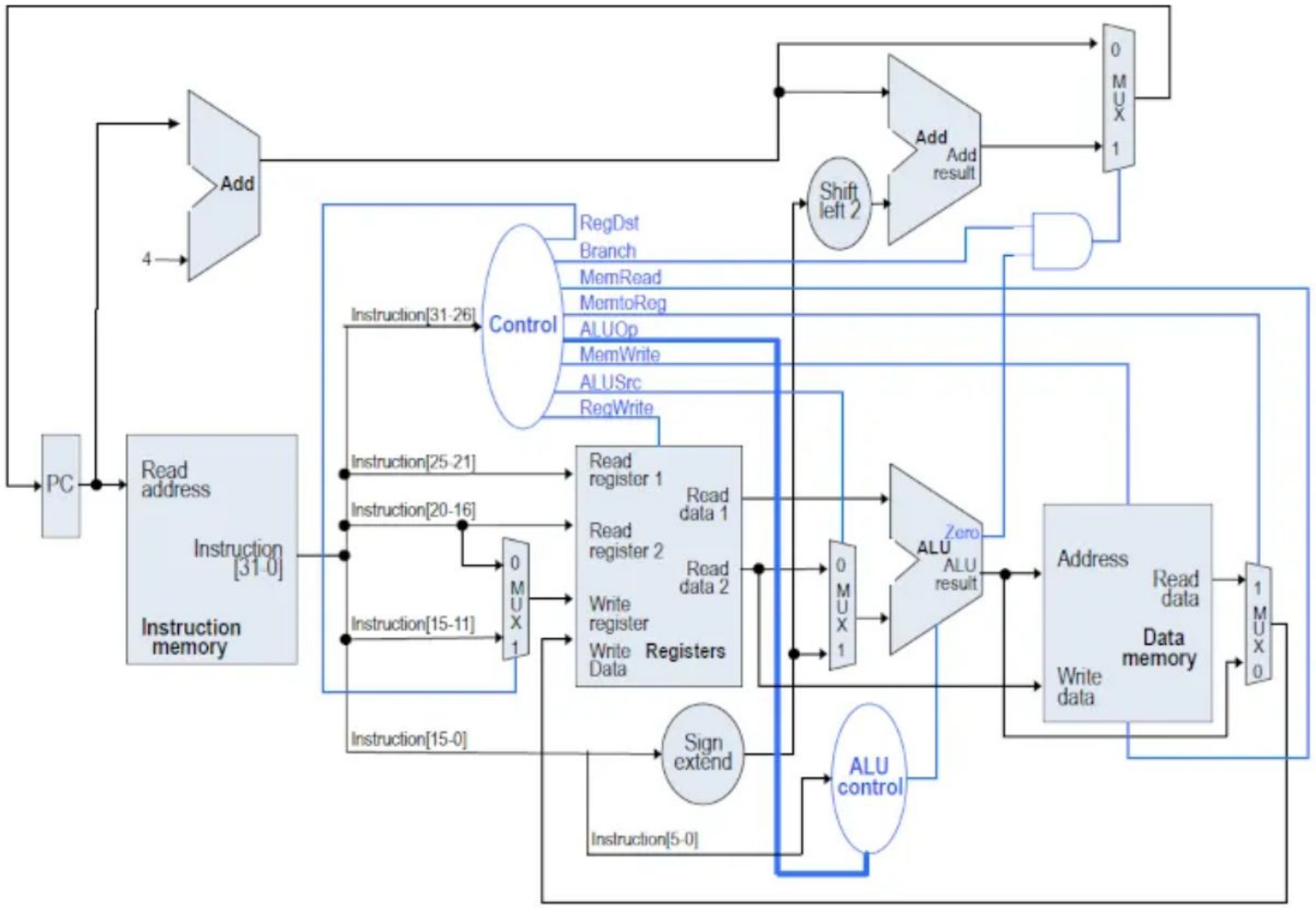




*Figure 2: Memory Hierarchy and Delay*

# Implementation Detail - CPU

- Single Cycle CPU with Cache
- Similar to HW1
- CPU features:
  - 32 registers
  - Input: 32-bit instructions



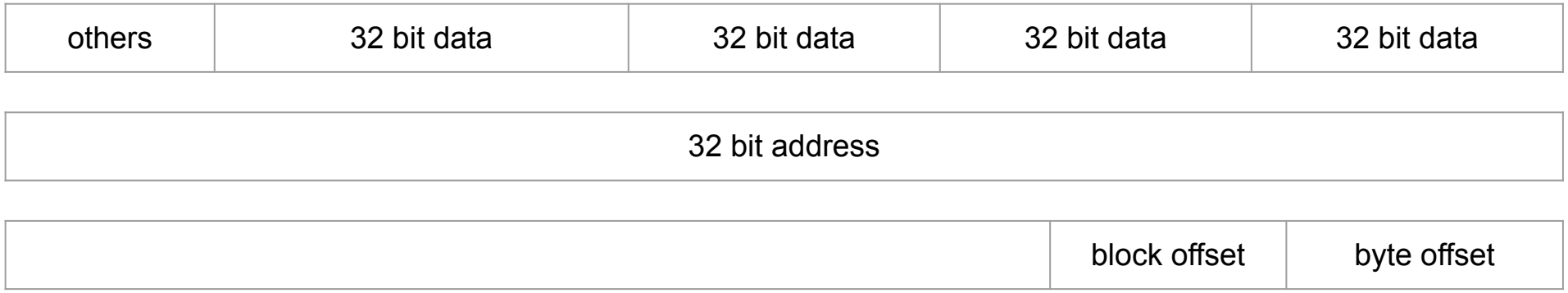
# Implementation Detail - CPU

Besides instruction implemented in HW1, some instructions also need to be supported:

- **R-type:** ADD, SUB, AND, XOR, MUL
- **I-type:** ADDI, SLLI, SLTI, SRAI, LW, JALR
- **S-type:** SW
- **B-type:** BEQ, BNE, BLT, BGE
- **U-type:** AUIPC
- **J-type:** JAL
- **System:** ECALL (Triggers `o_finish, 00000073`)

# Implementation Detail - Memory

- **TA has already implemented**
- Data Memory has a read/write penalty of 10 cycles
- Ensure the **Read/Write (R/W)** control signals for the data memory are held stable until the data is successfully returned.
- A block in data memory is 128 bits, containing four 32-bit words (or "distinct data") per request.
- The last 4 bits of the data memory address are structured as: **2 bits for the block offset and 2 bits for the byte offset. (Note: You may set all 4 bits to 0 in cache.v when addressing the main memory.)**




```
assign o_mem_addr   = {tag, index, 4'b0000} + i_offset;
```

# Implementation Detail - Cache

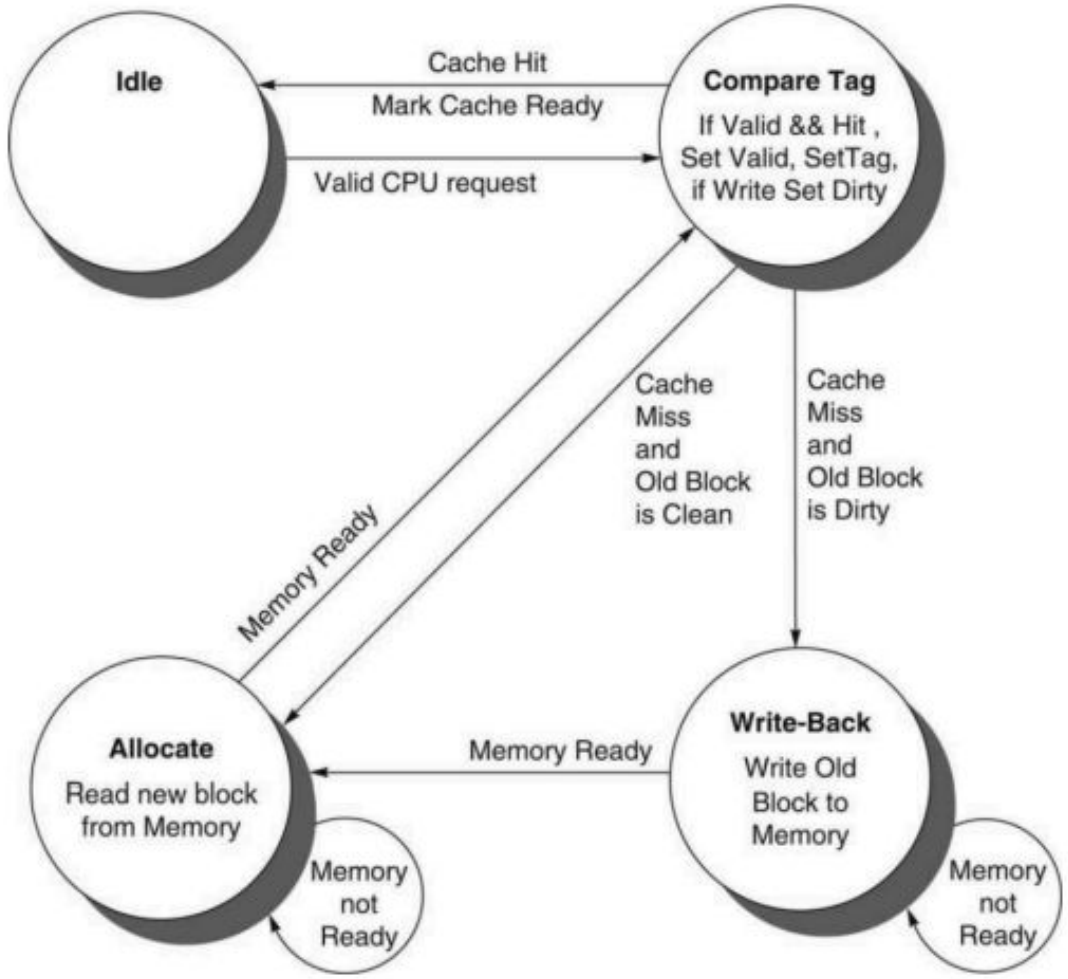
- Used to accelerate Data Memory Check the TA's implementation in the reference.
- Be careful when handling the processor address.
  - Before performing any address operations on i_proc_addr , subtract i_offset . Before outputting to the data memory ( o_DMEM_addr ), add the offset back.

```
assign o_cache_available = 1; // change this value to 1 if the cache is implemented

//------------------------------------------//
    //  default connection                  //
// assign o_mem_cen = i_proc_cen;             //
// assign o_mem_wen = i_proc_wen;             //
// assign o_mem_addr = i_proc_addr;           //
// assign o_mem_wdata = i_proc_wdata;         //
// assign o_proc_rdata = i_mem_rdata[0+:BIT_W];//
// assign o_proc_stall = i_mem_stall;         //
//------------------------------------------//
```

# Implementation Detail - Cache

- You can choose any cache implementation

  Write through, Write back Write around
- Direct-mapped, 2-way, 4-way….



# Implementation Detail - Area Limitation

Your design should not violate area limitation, or you can’t get the ranking score.

- CPU: 12000 mm^2
- Cache: 20000 mm^2

TA has already removes the area of Regfiles.

# Synthesizability

- All RTL code must be synthesizable (no unintended latches).
- The TA will check your design using **yosys** ( `synth.log` ). You do not need to modify any yosys scripts. The results can be found in Section 10.6 of `./log/synth.log` , including the estimated area of your module.
- Run parse.py to get area and latch results for CPU and cache, paste the results, and fill in the numbers in the report

# Report

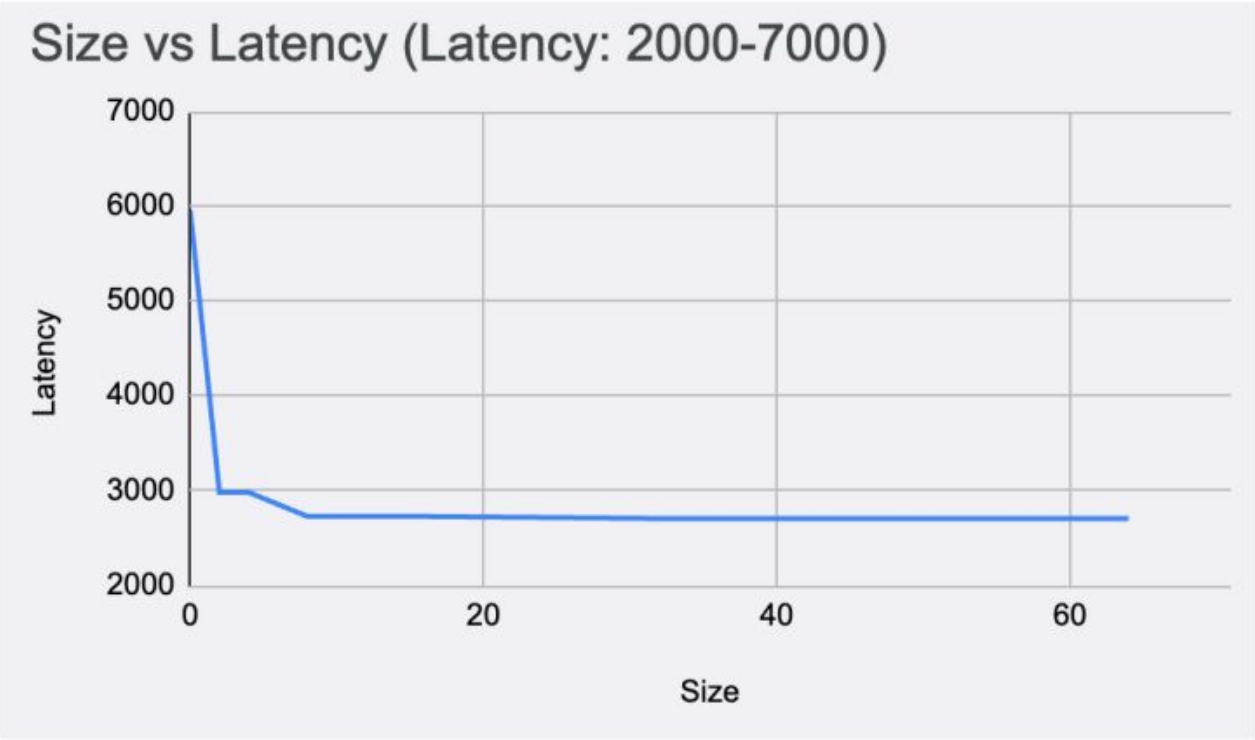
- **Explain your CPU and Cache.**
- The report also includes:
  - Synthesizability / latch check screenshot.
  - Development environment (OS, compiler, IDE).

# Report

Pareto figure for **cache size versus execution cycles** and Figure for **cache size versus cache area**

- Draw the figure with the I4 test case.
- Clearly specify the number of different cache sizes tested. You must evaluate at least four distinct cache sizes.
- Paste the “pass for simulation results” for each cache size.
- You can use your own testcase.
- If figure is clear enough, you don’t need to add text descriptions, but you can describe it as you wish.

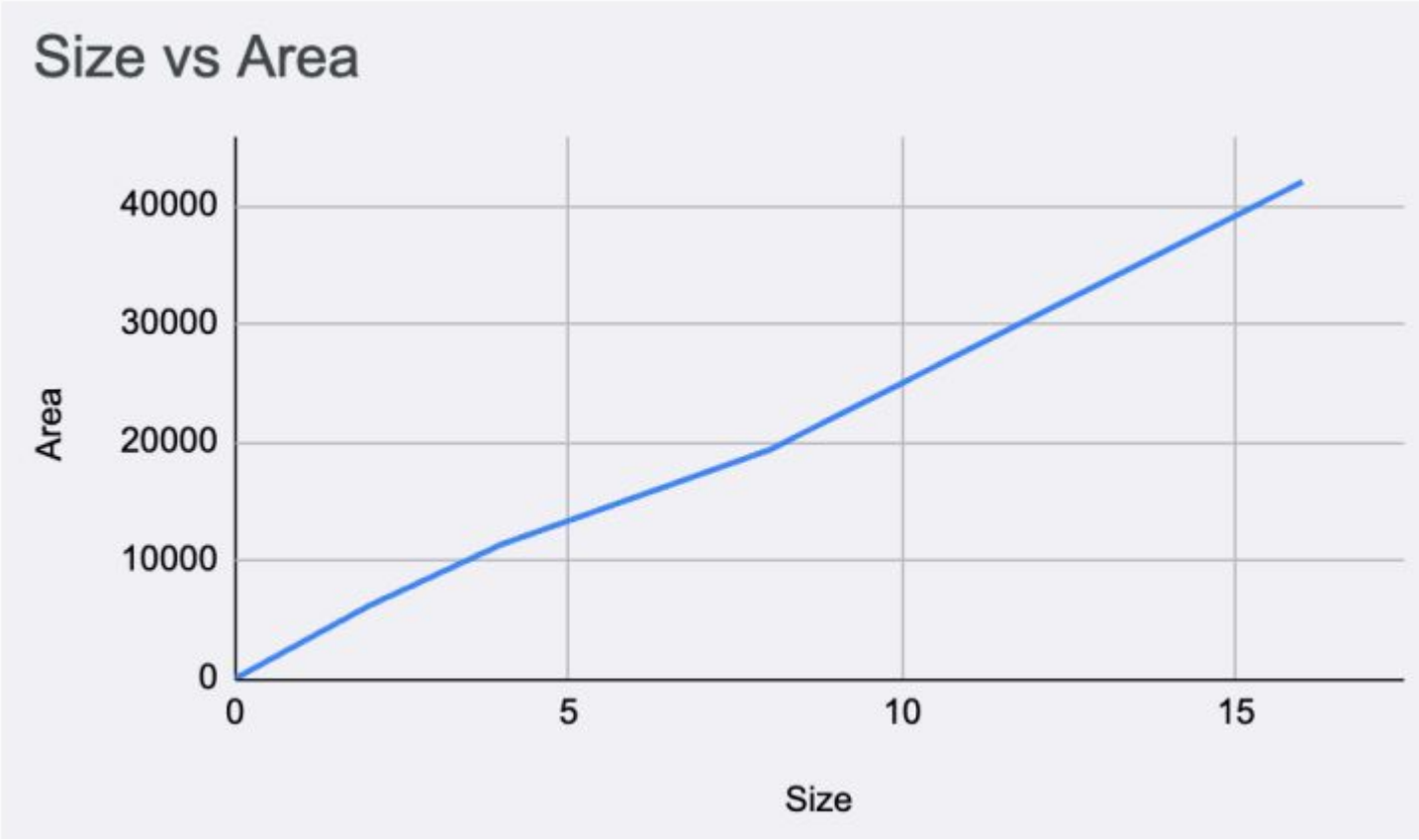
# Report




```
------------------------------------------------------------

START!!! I4 Simulation Start .....

------------------------------------------------------------

============================================================

Success!
The test result is .....PASS :)
Total execution cycle :         995
============================================================
```

# Report

Figure for cache size versus cache area



# Directory Structure

```
Lab3/
├── code
│   ├── Gen_Pattern
│   │   ├── I1_fact_gen.py
│   │   ├── I2_hw1_gen.py
│   │   ├── I3_insertion_sort_gen.py
│   │   └── I4_bubble_sort_gen.py
│   ├── Pattern
│   │   ├── I1
│   │   │   ├── golden.dat
│   │   │   ├── mem_D.dat
│   │   │   └── mem_I.dat
│   │   ├── I2
```

```
│   │   │   ├── golden.dat
│   │   │   ├── mem_D.dat
│   │   │   └── mem_I.dat
│   │   ├── I3
│   │   │   ├── golden.dat
│   │   │   ├── mem_D.dat
│   │   │   ├── mem_I.dat
│   │   │   └── mem_I_listing.txt
│   │   └── I4
│   │       ├── golden.dat
│   │       ├── mem_D.dat
│   │       ├── mem_I.dat
│   │       └── mem_I_listing.txt
│   ├── src
│   │   ├── cache.v
│   │   └── CHIP.v
│   ├── supplied
│   │   ├── memory.v
│   │   └── Regfile.v
│   └── tb
│       └── tb.v
├── docker-compose.yml
├── dockerfile
├── judge.yaml
├── log
│   ├── cache_syn.log
│   ├── cpu_syn.log
│   ├── output_1.txt
│   ├── output_2.txt
│   ├── output_3.txt
│   └── output_4.txt
├── Makefile
├── parse.py   # check for latch and area
└── run_all.sh # use for simulation
```

# Submission Format

```
studentID_lab3/
  ├── src/   (all Verilog codes you wrote)
  ├── studentID_lab2_report.pdf
```

Do not include: tb.v, memory.v, Regfile.v

12/9 23:59

# Grading Policy

- **Pass Testcase (40%)**
  - Public  testcases: 20% (4 × 5%)
  - Hidden testcases: 20% (4 × 5%)
- **Performance (30%)**
  - Baseline (20%) : Total execution cycles in public testcases (I1~I4) less than 5000 (implement cache)
  - Ranking (10%): Shortest total execution cycles
    - You get 0 in ranking once:
      - late submission
      - violate area limitations
      - have latch violations
      - the design is unsynthesizable
      - compile error
    - Score with linear ranking
- **Report (30%)**
  - Description (10%)
    - Describe your CPU and Cache design
    - If you use LLM, clearlys state which parts you use.
    - We take Plagiarism seriously.
  - **Pareto** and area diagram: 10% (5% for each)
  - Demo 10 %

# Grading Policy

- Bonus (20%)
  - TA strongly recommends a novel or customized design of the cache. If you do exceptionally well on this, no worry about the ranking score.
  - If you implement the design from paper or some complicated cache, feel free to write in the report!
  - Judged by the TA, however, implementing the cache mentioned in the class won't get any bonuses.

# Grading Policy

- **Other penalties**:
  - Compilation error (naming, or minor errors in 3 lines): –5%
  - Submitting unnecessary files (tb or supplied): -5%
  - Wrong directory format: –5%.
  - Plagiarism: get 0 in this homework, and TA will inform teacher
  - Late submission: -10% per day, at most a week
  - Major mistakes causing compilation error → programming part 0. (Judged by TA )

# Reference

| | W/o cache | W cache |
|---|---|---|
| I1 | 806 | 692 |
| I2 | 858 | 744 |
| I3 | 2008 | 1098 |
| I4 | 2051 | 995 |
| Total | 5723 | 3529 |
| | | |
| | | area:11356 |

# Easter Egg

```
`define COOL_OUTPUT 0  // Modify to 1 for cool output figures
```