Ramulator 2.0 Summary

DRAM Operations & States

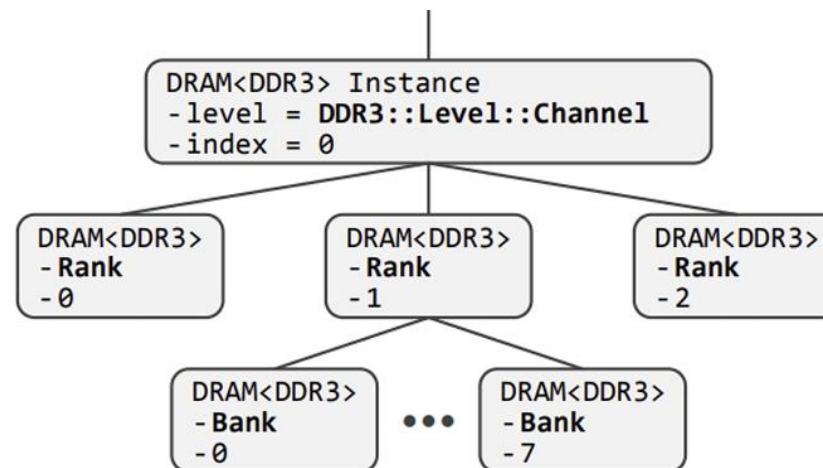❑ **DRAM Operations & States**

- **Main DRAM states**
  - **Activate**
  - **Read/Write**
  - **Precharge**

```
1   // DRAM.h
2   template <typename T>
3   class DRAM {
4     DRAM<T>* parent;
5     vector<DRAM<T>*>
6       children;
7     T::Level level;
8     int index;
9     // more code...
10  };
```

```
1   // DDR3.h/cpp
2   class DDR3 {
3     enum class Level {
4       Channel, Rank, Bank,
5       Row, Column, MAX
6     };
7
8
9     // more code...
10  };
```

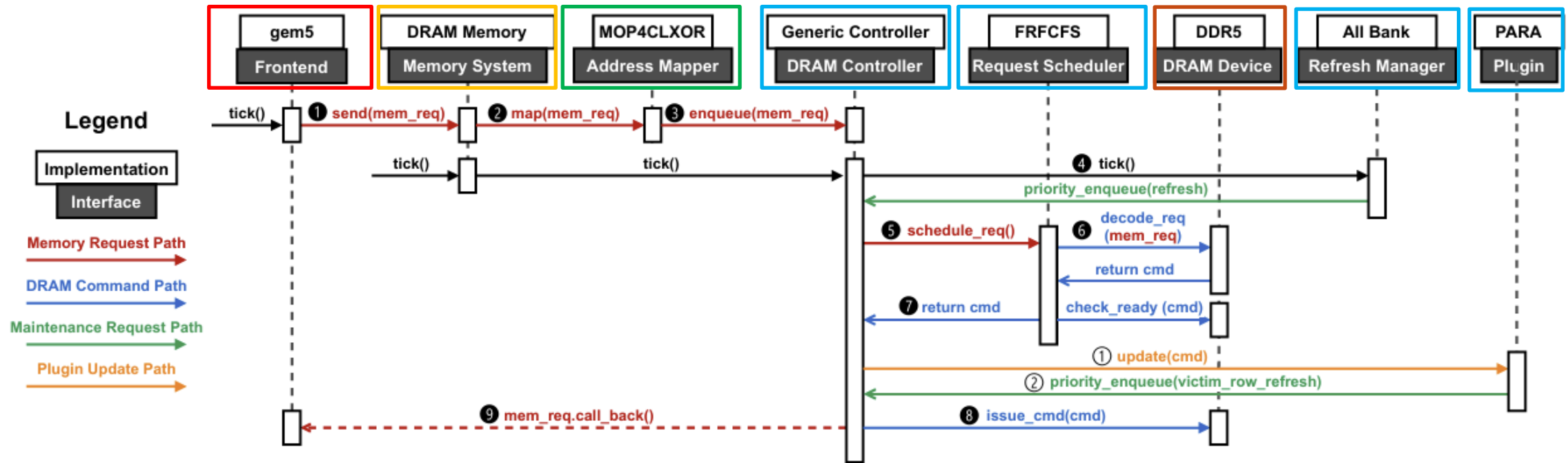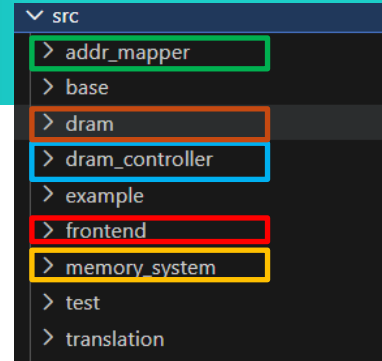# src files <=> DRAM Operation

❑ **Simulation Configuration**

Fig. 1: High-level software architecture of Ramulator 2.0 using an example DDR5 system configuration

src files <=> DRAM Operation
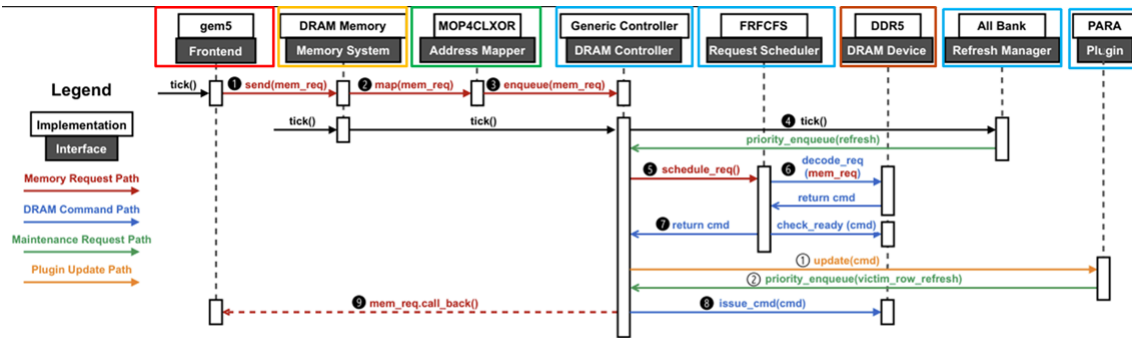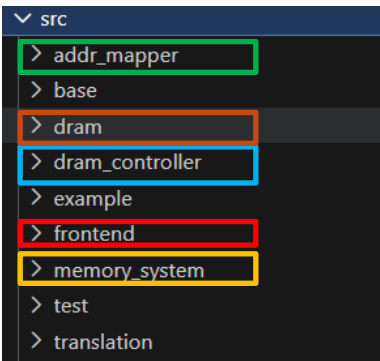
❑Simulation Configuration

Fig. 1: High-level software architecture of Ramulator 2.0 using an example DDR5 system configuration

1. **Requests are sent**: Front-end(trace file)에서 Memory Request를 보냄

2. **Memory Addresses are Mapped**: Address Mapper가 Request Address를 DRAM 구조에 맞게 변환

3. **Enqueue**: DRAM Ctrlr의 Buffer에 Request를 넣음

4. **DRAM Ctrlr - Ticking Refresh Manager**: Ctrlr가 Refresh Manager를 호출해 high-priority maintenance request(ex. Refresh)을 추가

5. **DRAM Ctrlr - Request Scheduling**: Request Scheduler에게 최적의 Request을 선택하라고 요청

6. **DRAM Device가 Request 확인**: Scheduler가 DRAM Device Model을 참조해 적합한 Command를 Decode

7. **Issue Command**: DRAM Ctrlr가 DRAM Command를 보냄

8. **Updates the behavior and timing information:** DRAM Command Issue시 State & Timing이 Update

9. **Notify the frontend**: Memory Request가 끝나면 callback으로 frontend에 알림

main function

❑ **main.cpp**

```cpp
13 ∨ int main(int argc, char* argv[]) {
14     // Parse command line arguments
15     argparse::ArgumentParser program("Ramulator", "2.0");
16     program.add_argument("-c", "--config").metavar("\"dumped YAML configuration\"")
17         .help("String dump of the yaml configuration.");
18     program.add_argument("-f", "--config_file").metavar("path-to-configuration-file")
19         .help("Path to a YAML configuration file.");
20     program.add_argument("-p", "--param").metavar("KEY=VALUE")
21         .append()
22         .help("Specify parameter to override in the configuration file. Repeat this option to change multiple parameters.");
```

```cpp
88     // Connect the frontend and the memory system together,
89     // this recursively calls the "setup" function in all instaniated components
90     // so that they can get each other's parameters (if needed) after their initialization
91     frontend->connect_memory_system(memory_system);
92     memory_system->connect_frontend(frontend);
93
94     // Get the relative clock ratio between the frontend and memory system
95     int frontend_tick = frontend->get_clock_ratio();
96     int mem_tick = memory_system->get_clock_ratio();
97
98     int tick_mult = frontend_tick * mem_tick;
99
00     for (uint64_t i = 0;; i++) {
01         if (((i % tick_mult) % mem_tick) == 0) {
02             frontend->tick();
03         }
04
05         if (frontend->is_finished()) {
06             break;
07         }
08
09         if ((i % tick_mult) % frontend_tick == 0) {
10             memory_system->tick();
11         }
12     }
13
14     // Finalize the simulation. Recursively print all statistics from all components
15     frontend->finalize();
16     memory_system->finalize();
17
18     return 0;
19 }
```

main.cpp

1. **Argument 받는 부분**
   - **Options**
     1. -c: command line dump
     2. -f: YAML document
     3. -p: overriding parameters in a YAML document

2. **Long for loop를 통한 tick() 기반 simul**
   1. frontend(core)가 발행한 예상 instructions들을 모두 처리시 is_finished()가 true가 됨

yaml file

❑example_config.yaml

```
 1 ∨ Frontend:
 2     impl: SimpleO3
 3     clock_ratio: 8
 4     num_expected_insts: 500000
 5 ∨   traces:
 6       - example_inst.trace
 7
 8 ∨ Translation:
 9     impl: RandomTranslation
10     max_addr: 2147483648
11
```

```
13   MemorySystem:
14     impl: GenericDRAM
15     clock_ratio: 3
16
17     DRAM:
18       impl: DDR4
19       org:
20         preset: DDR4_8Gb_x8
21         channel: 1
22         rank: 2
23       timing:
24         preset: DDR4_2400R
25
26     Controller:
27       impl: Generic
28       Scheduler:
29         impl: FRFCFS
30       RefreshManager:
31         impl: AllBank
32       RowPolicy:
33         impl: ClosedRowPolicy
34         cap: 4
35       plugins:
36
37     AddrMapper:
38       impl: RoBaRaCoCh
```

1. **Frontend Interface(IFrontEnd) 부분**
   - **trace file에서 Instruction읽고, Memory Request 생성**
   - *impl: SimpleO3*
     - ⇒ Simple Out-of-Order (O3) CPU
   - *clock_ratio: 8*
     - ⇒ global CLK 대비 Frontend CLK 속도
   - *num_expected_insts: 500000*
     - ⇒ Simulation이 해당 instruction 수에 도달 시 종료
   - *traces*
     - ⇒ Instruction trace file(include memory access Inst)
   - *impl: RandomTranslation*
     - ⇒ Physical Memory ↔ Virtual Memory 변환
     - ⇒ System의 Page Table 등을 간단히 Modeling
   - *max_addr: 2147483648*
     - ⇒ Translation 시 address overflow 방지

yaml file

❑ example_config.yaml

```
 1 ∨ Frontend:
 2     impl: SimpleO3
 3     clock_ratio: 8
 4     num_expected_insts: 500000
 5 ∨   traces:
 6       - example_inst.trace
 7
 8 ∨ Translation:
 9     impl: RandomTranslation
10     max_addr: 2147483648
11
```

```
13     MemorySystem:
14       impl: GenericDRAM
15       clock_ratio: 3
16
17       DRAM:
18         impl: DDR4
19         org:
20           preset: DDR4_8Gb_x8
21           channel: 1
22           rank: 2
23         timing:
24           preset: DDR4_2400R
25
26       Controller:
27         impl: Generic
28         Scheduler:
29           impl: FRFCFS
30         RefreshManager:
31           impl: AllBank
32         RowPolicy:
33           impl: ClosedRowPolicy
34           cap: 4
35         plugins:
36
37       AddrMapper:
38         impl: RoBaRaCoCh
```

2. MemorySystem Interface 부분

- **Frontend의 Request를 받아 DRAM Ctrlr을 통해 처리**
- **Latency, en/dequeue, Timing Constraints 처리**

- *impl: GenericDRAM*
    - ⇒ 기본 DRAM 기반 System, Ctrlr와 DRAM을 통합

- *clock_ratio: 3*
    - ⇒ global CLK 대비 MemorySystem CLK 속도
    - ⇒ 현재: DRAM이 CPU보다 느린 System (= 3:8)

yaml file

☐ example_config.yaml

```
1 ∨ Frontend:                          13    MemorySystem:
2      impl: SimpleO3                   14      impl: GenericDRAM
3      clock_ratio: 8                   15      clock_ratio: 3
4      num_expected_insts: 500000       16
5 ∨    traces:                          17      DRAM:
6        - example_inst.trace           18        impl: DDR4
7                                       19        org:
8 ∨  Translation:                       20          preset: DDR4_8Gb_x8
9      impl: RandomTranslation          21          channel: 1
10     max_addr: 2147483648             22          rank: 2
11                                      23        timing:
                                        24          preset: DDR4_2400R
                                        25
                                        26      Controller:
                                        27        impl: Generic
                                        28        Scheduler:
                                        29          impl: FRFCFS
                                        30        RefreshManager:
                                        31          impl: AllBank
                                        32        RowPolicy:
                                        33          impl: ClosedRowPolicy
                                        34          cap: 4
                                        35        plugins:
                                        36
                                        37      AddrMapper:
                                        38        impl: RoBaRaCoCh
```

2. MemorySystem Interface 부분

- **DRAM Section**
- *impl: DDR4*
  - ⇒ tick() 시 Timing Check / Command Issue 실행.
- *org: DDR4_8Gb_x8*
  - ⇒ 현재 DRAM preset - 8Gb 용량, x8bit data bus
  - ⇒ Channel/Rank 설정 시 기본 Preset설정을 Override 함
- *timing: DDR4_2400R*
  - ⇒ Timing preset - nRCD등의 Timing Contraint 정의
  - ⇒ 이를 이용해 tick() 시 Latency 계산

```
26    inline static const std::map<std::string, std::vector<int>> timing_presets = {
27      //  name         rate  nBL nCL nRCD nRP nRAS nRC  nWR nRTP nCWL nCCDS nCCDL nRRDS nRRDL nWTRS nW
28      {"DDR4_1600J",  {1600,  4,  10, 10,  10,  28,  38,  12,  6,   9,   4,    5,    -1,   -1,   2,
29      {"DDR4_1600K",  {1600,  4,  11, 11,  11,  28,  39,  12,  6,   9,   4,    5,    -1,   -1,   2,
30      {"DDR4_1600L",  {1600,  4,  12, 12,  12,  28,  40,  12,  6,   9,   4,    5,    -1,   -1,   2,
31      {"DDR4_1866L",  {1866,  4,  12, 12,  12,  32,  44,  14,  7,   10,  4,    5,    -1,   -1,   3,
32      {"DDR4_1866M",  {1866,  4,  13, 13,  13,  32,  45,  14,  7,   10,  4,    5,    -1,   -1,   3,
33      {"DDR4_1866N",  {1866,  4,  14, 14,  14,  32,  46,  14,  7,   10,  4,    5,    -1,   -1,   3,
34      {"DDR4_2133N",  {2133,  4,  14, 14,  14,  36,  50,  16,  8,   11,  4,    6,    -1,   -1,   3,
35      {"DDR4_2133P",  {2133,  4,  15, 15,  15,  36,  51,  16,  8,   11,  4,    6,    -1,   -1,   3,
36      {"DDR4_2133R",  {2133,  4,  16, 16,  16,  36,  52,  16,  8,   11,  4,    6,    -1,   -1,   3,
37      {"DDR4_2400P",  {2400,  4,  15, 15,  15,  39,  54,  18,  9,   12,  4,    6,    -1,   -1,   3,
38      {"DDR4_2400R",  {2400,  4,  16, 16,  16,  39,  55,  18,  9,   12,  4,    6,    -1,   -1,   3,
39      {"DDR4_2400U",  {2400,  4,  17, 17,  17,  39,  56,  18,  9,   12,  4,    6,    -1,   -1,   3,
40      {"DDR4_2400T",  {2400,  4,  18, 18,  18,  39,  57,  18,  9,   12,  4,    6,    -1,   -1,   3,
```

```cpp
class DDR4 : public IDRAM, public Implementation {
  RAMULATOR_REGISTER_IMPLEMENTATION(IDRAM, DDR4, "DDR4", "DDR4 Device Model")

public:
  inline static const std::map<std::string, Organization> org_presets = {
    //  name         density  DQ  Ch Ra Bg Ba  Ro      Co
    {"DDR4_2Gb_x4",   {2<<10,  4,  {1, 1, 4, 4, 1<<15, 1<<10}}},
    {"DDR4_2Gb_x8",   {2<<10,  8,  {1, 1, 4, 4, 1<<14, 1<<10}}},
    {"DDR4_2Gb_x16",  {2<<10, 16,  {1, 1, 2, 4, 1<<14, 1<<10}}},
    {"DDR4_4Gb_x4",   {4<<10,  4,  {1, 1, 4, 4, 1<<16, 1<<10}}},
    {"DDR4_4Gb_x8",   {4<<10,  8,  {1, 1, 4, 4, 1<<15, 1<<10}}},
    {"DDR4_4Gb_x16",  {4<<10, 16,  {1, 1, 2, 4, 1<<15, 1<<10}}},
    {"DDR4_8Gb_x4",   {8<<10,  4,  {1, 1, 4, 4, 1<<17, 1<<10}}},
    {"DDR4_8Gb_x8",   {8<<10,  8,  {1, 1, 4, 4, 1<<16, 1<<10}}},
    {"DDR4_8Gb_x16",  {8<<10, 16,  {1, 1, 2, 4, 1<<16, 1<<10}}},
    {"DDR4_16Gb_x4",  {16<<10, 4,  {1, 1, 4, 4, 1<<18, 1<<10}}},
    {"DDR4_16Gb_x8",  {16<<10, 8,  {1, 1, 4, 4, 1<<17, 1<<10}}},
    {"DDR4_16Gb_x16", {16<<10, 16, {1, 1, 2, 4, 1<<17, 1<<10}}},
  };
```

```cpp
class DDR4 : public IDRAM, public Implementation {
  void tick() override {

  void init() override {
    RAMULATOR_DECLARE_SPECS();
    set_organization();
    set_timing_vals();

    set_actions();
    set_preqs();
    set_rowhits();
    set_rowopens();
    set_powers();

    create_nodes();
  };
```

❑ example_config.yaml

```
1  ∨ Frontend:
2      impl: SimpleO3
3      clock_ratio: 8
4      num_expected_insts: 500000
5  ∨   traces:
6        - example_inst.trace
7
8  ∨ Translation:
9      impl: RandomTranslation
10     max_addr: 2147483648
11
13  MemorySystem:
14      impl: GenericDRAM
15      clock_ratio: 3
16
17      DRAM:
18        impl: DDR4
19        org:
20          preset: DDR4_8Gb_x8
21          channel: 1
22          rank: 2
23        timing:
24          preset: DDR4_2400R
25
26      Controller:
27        impl: Generic
28        Scheduler:
29          impl: FRFCFS
30        RefreshManager:
31          impl: AllBank
32        RowPolicy:
33          impl: ClosedRowPolicy
34          cap: 4
35        plugins:
36
37      AddrMapper:
38        impl: RoBaRaCoCh
```

2. MemorySystem Interface 부분

- **Controller Section**
- *impl: Generic*
    - ⇒ Generic - 기본 Ctrlr
    - ⇒ Request Queue/Scheduling 등 관리
- *Scheduler - impl: FRFCFS*
    - ⇒ FRFCFS - First-Ready First-Come-First-Serve
        - ⇒ 준비된 Request를 Queue에서 꺼내 우선 처리
- *RefreshManager - impl: AllBank*
    - ⇒ AllBank - 모든 Bank simultaneous Refresh
- *RowPolicy - impl: ClosedRowPolicy*
    - ⇒ ClosedRowPolicy - 사용 후 Row 즉시 닫음(Precharge)
    - ⇒ cap:4 - 열려있는 Row 최대 수 제한
- *plugins*
    - ⇒ 현재 Ramulator에서는 Row Hammering 완화 기법을 plugin으로 제공해줌

❑ example_config.yaml

```
 1 ∨ Frontend:
 2      impl: SimpleO3
 3      clock_ratio: 8
 4      num_expected_insts: 500000
 5 ∨    traces:
 6        - example_inst.trace
 7
 8 ∨  Translation:
 9      impl: RandomTranslation
10      max_addr: 2147483648
11
```

```
13   MemorySystem:
14     impl: GenericDRAM
15     clock_ratio: 3
16
17     DRAM:
18       impl: DDR4
19       org:
20         preset: DDR4_8Gb_x8
21         channel: 1
22         rank: 2
23       timing:
24         preset: DDR4_2400R
25
26     Controller:
27       impl: Generic
28       Scheduler:
29         impl: FRFCFS
30       RefreshManager:
31         impl: AllBank
32       RowPolicy:
33         impl: ClosedRowPolicy
34         cap: 4
35       plugins:
36
37     AddrMapper:
38       impl: RoBaRaCoCh
```

2. MemorySystem Interface 부분

- **AddrMapper Section**

- *impl: RoBaRaCoCh*
    - ⇒ Row-Bank-Rank-Column-Channel Mapping Scheme
    - ⇒ Requested Addres 변환(Physical → DRAM Vector)

    [Physical → DRAM Vector Example]
    - ⇒ Physical Address: 0x12345678
    - ⇒ DRAM Vector:
        [Channel:0, Rank:1, Bank:2, Row:128, Column:512]

trace file

☐ example_inst.trace & trace.cpp & core.cpp

- **simpleO3 CPU model기준**

    - **Ramulator는 "Memory" Simulator**

    - **Memory 명령어만 취급하기에, 3가지로만 Instruction을 분리한다.**

        1. **Not Memory Operation**

        2. **Load**

        3. **Store**

- **각 line은 load(& store)동작을 나타낸다.**

- **1st line : 3cycle동안 stall → load**

- **5th line : 8cycle동안 stall → load → store**

- **따라서, simpleO3 기반 trace파일:**
    - **1st column은 Not Memory Operation Cycle 수 (or ticks 수)**
    - **2nd column은 load operation address**
    - **3rd column은 store operation address**

trace file

☐ example_inst.trace & trace.cpp & core.cpp

- **Trace file을 arg로 받아, Frontend (ex: simpleO3.cpp)에서 처리되어 메모리 접근 request를 생성**

- **simpleO3 CPU model기준: trace의 1st token: bubble / 2nd token: load address / 3rd token: store address**

src/frontend folder

❑ **frontend**

- **abcd**

src/frontend folder

❑ **frontend**

- **abcd**

src/frontend folder

❑ **frontend**

- **abcd**

src/memory_system folder

❏ **memory_system**

- **abcd**

src/memory_system folder

❑ **memory_system**

- **abcd**

❑ **addr_mapper**

- **abcd**

❑ **addr_mapper**

- **abcd**

src/translation folder

❑ **translation**

- **abcd**

src/translation folder

❏ **translation**

- **abcd**

src/translation folder

❑ **translation**

  • **abcd**

src/dram folder

❑ **dram**

- **abcd**

src/dram folder

❑ **dram**

- **abcd**

src/dram folder

❑ **dram**

- **abcd**

src/dram_controller folder

❏ **dram_controller**

• **abcd**

src/dram_controller folder

❑ **dram_controller**

- **abcd**

src/dram_controller folder

❑ **dram_controller**

- **abcd**

src/dram_controller folder

❑ **dram_controller**

- **abcd**

src/base folder

❑ **base**

- **abcd**