# 实验环境介绍

每次写代码前，用 git graph 插件查看代码仓库是否更新了。

```
git fetch --all
git merge origin/master
git submodule update --init
```

写完代码后，记得 commit。

# CPU 接口

流水线部分有指令内存接口与数据内存接口。

指令内存接口：

```
typedef struct packed {
    logic  valid;  // in request? (Used in Lab 2)
    addr_t addr;   // target address
} ibus_req_t;

typedef struct packed {
    logic  addr_ok;  // is the address accepted by cache? (Used in Lab 2)
    logic  data_ok;  // is the field "data" valid?        (Used in Lab 2)
    u32 data;        // the data read from cache
} ibus_resp_t;
```

数据内存接口：

```
typedef struct packed {
    logic    valid;   // in request?
    addr_t   addr;    // target address
    msize_t  size;    // number of bytes                (Used in Lab 2)
    strobe_t strobe;  // which bytes are enabled? set to zeros for read
request (in Lab 1, it is either '0 or '1)
    word_t   data;    // the data to write
} dbus_req_t;

typedef struct packed {
    logic  addr_ok;  // is the address accepted by cache?   (Used in Lab 2)
    logic  data_ok;  // is the field "data" valid?          (Used in Lab 2)
    word_t data;     // the data read from cache
} dbus_resp_t;
```

# 接入Verilator仿真

将 CPU 接入 Verilator Difftest 的仿真接口。

需要例化三个模块（所给框架中已例化好，需要接线）。

首先是当前周期提交的指令：

```

```

```
DifftestInstrCommit DifftestInstrCommit(
    .clock              (clk),
    .coreid             (0), // 无需改动
    .index              (0), // 多发射时，例化多个该模块。前四个 Lab 无需改动它
    .valid              (0), // 无提交（或提交的指令是flush导致的bubble时，为0）
    .pc                 (0), // 这条指令的 pc
    .instr              (0), // 这条指令的内容，可不改动
    .skip               (0), // 提交的是一条内存读写指令，且这部分内存属于设备（addr[31] == 0）时，skip为1
    .isRVC              (0), // 前四个 Lab 无需改动
    .scFailed           (0), // 前四个 Lab 无需改动
    .wen                (0), // 这条指令是否写入通用寄存器，1 bit
    .wdest              (0), // 写入哪个通用寄存器
    .wdata              (0)  // 写入的值
);
```

这个周期的指令提交后，通用寄存器的内容（已连接好）：

```
DifftestArchIntRegState DifftestArchIntRegState (
    .clock              (clk),
    .coreid             (0),
    .gpr_0              (regfile.regs_nxt[0]),
    // ...
);
```

这个周期的指令提交后，系统寄存器的内容（Lab4 的内容，前面的 Lab 可以不管）：

```
DifftestCSRState DifftestCSRState(

);
```

仿真通过的标志：控制台输出 `hit good trap`（通过后会报寄存器值错误，这个 `hit good trap` 在报错内容的上面）

# 生成波形图

不生成波形图时运行测试，使用`make test-lab1`；需要生成波形图，使用 `make test-lab1 VOPT="--dump-wave"`，运行结束后可在 `build` 目录下看到波形图，使用 gtkwave 打开。

默认截取前 $10^6$ 个时钟周期。如果需要调整，使用 `make test-lab1 VOPT="--dump-wave -b <begin> -e <end>"`。