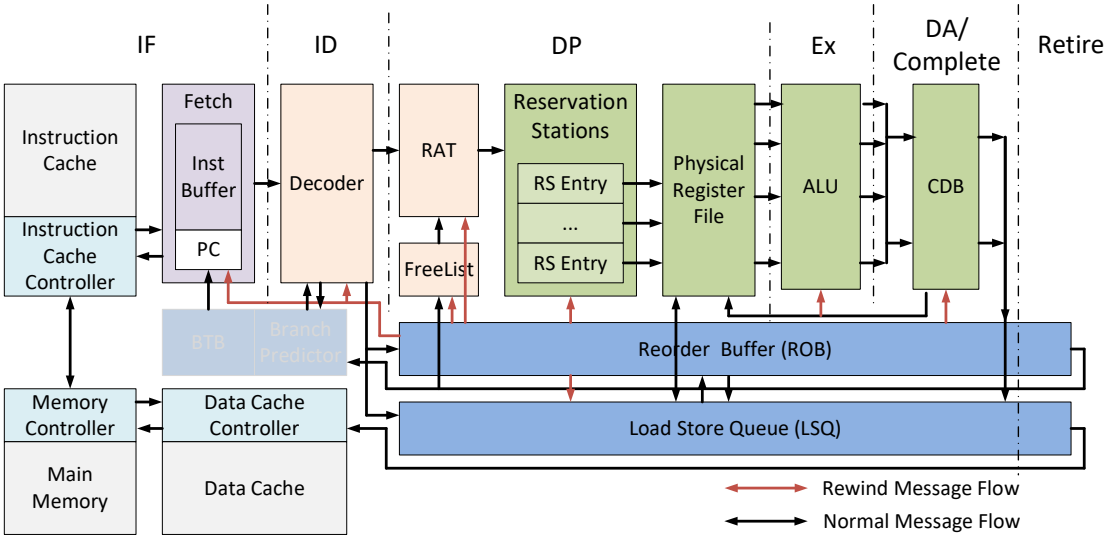
# 北京大学 - 智能硬件体系结构

## 2024年秋季实验1：MIPS RISC-Style Superscalar Out-of-Order Processor

指令集架构是当代处理器芯片的核心技术路径，也是未来智能硬件体系结构的关键发展方向。本项目基于开源 MIPS RISC-Style 指令集架构，旨在帮助同学们学习和理解超标量乱序执行指令集架构的运行原理，深入理解流水线计算单元（ALU）、多路数据缓存（DCACHE）、寄存器重命名（Register Renaming）和超标量指令乱序发射（Superscalar OoO Instruction Execution）等机制。

### 1、整体架构说明




MIPS RISC-Style 指令集架构示意图

本项目的 MIPS RISC-Style 指令集架构如上图所示，整体架构的代码描述在 verilog/oo_pipeline.v 中，各模块的功能描述、主要的 IO 和详细解释内容请参考课程网站的模块说明部分。

### 2、实验任务简介

本项目提供项目代码以及测试系统，同学们需要完成的模块包括以下 4 个：1) ALU、2）DCACHEMEM、3）RAT/RATTABLE 和 4）SUPER_RS。请同学们将这 4 个模块中缺失的代码部分填充完整，缺失代码的起始部分标注如下：

```
// ============ Start ====================
// ==========Descriptions of Functions=========
    /*
    待填充的代码模块
    */
// ==============End====================
```

**具体需要完成的工作如下：**

1）在 verilog/alu.v 源文件中，填补简单 ALU 模块的功能设计，完成基本算术、逻辑、位移和分支计算功能，模块输入输出解释请参考课程网站 Project 部分的模块功能介绍，ALUOp 需要完成的功能在 sys_defs.vh 中，可自行构建 testbench_alu.v 模块进行测试验证，alu.v 中需要填补 3 个代码块，每一个代码块大约 10~20 行左右；

ALU 需要完成的功能示例代码具体如下（假设输入为 opa 和 opb）：

```
`ALU_ADDQ:      opa + opb
`ALU_SUBQ:      opa - opb
`ALU_AND:       opa & opb
`ALU_BIC:       opa & ~opb;
`ALU_BIS:       opa | opb;
`ALU_ORNOT:     opa | ~opb;
`ALU_XOR:       opa ^ opb;
`ALU_EQV:       opa ^ ~opb;
`ALU_SRL:       opa >> opb[5:0];
`ALU_SLL:       opa << opb[5:0];
`ALU_SRA:       (opa >> opb[5:0]) | ({64{opa[63]}} << (64 - opb[5:0]));
`ALU_CMPULT:    { 63'd0, (opa < opb) };
```

```
`ALU_CMPEQ:     { 63'd0, (opa == opb) };
`ALU_CMPULE:    { 63'd0, (opa <= opb) };
`ALU_CMPLT:     { 63'd0, signed_lt(opa, opb) };
`ALU_CMPLE:     { 63'd0, (signed_lt(opa, opb) || (opa == opb)) };
```

2）在 verilog/dcachemem.v 源文件中，填补 2-Way Cache 的功能设计，完成双路读出、写入功能，可自行构建 testbench_dcachemem.v 模块进行测试验证，dcachemem.v 中需要填补 2 个代码块，每一个代码块大约 20 行左右；

3）在 verilog/ratable.v 源文件中，填补完成简单 RATTABLE 模块的功能设计，支持 Architectural Register 与 Physical Register 的映射与回收功能，需要填补的 1 个代码块，大约 10 行左右；

在 verilog/rat.v 源文件中，填补 freelist 相关逻辑实现对空闲 Physical Register 的状态检测与更新；可参考 testbench/testbench_rat.v 构建测试模块进行验证，需要填补的 1 个代码块，大约 30 行左右；

4）在 verilog/superrs.v 完成 SUPER_RS 模块的功能实际，将模块 RS（rs.v 中）连入 SUPER_RS 模块以支持超标量=2 的指令发射功能，可参考 testbench_RS.v 构建测试模块进行验证，需要填补的 1 个代码块，大约 30 行左右；

**3、项目文件简介**

本项目将提供代码以及整个测试系统。项目包结构如下：

./...

---Makefile　　包含所有 make command

---Make.*　　包含子模块的 make command，可新建子模块验证路径单独验证各

个子模块

示例：make all　　可自动运行 program.mem 内的程序

make clean　　删除所有编译产生文件

---debug_out　　仿真产生的.out 文件，提供简单的可视化观察模块和流水线的状态

---vs-asm　　MIPS RISC 指令的 Assembler 编译器，可以将 test_progs 路径下的指令汇编代码翻译成 HEX 格式，存入 program.mem 文件供 testbench 文件夹下的 tb 读取

示例：vs-asm test_progs/fib.s > program.mem

---pipeline_gold　　最简单的顺序 5 级流水线设计，产生用作验证的参考 write_back.out 和 memory.out，与本项目乱序执行超标量结果进行对比

---program.mem　　汇编指令代码的 HEX 格式文件，有 tb 读取作为 icache 的输入

---run_tests.sh　　运行该脚本可一次性验证 test_progs 内的所有汇编指令代码运行是否正确，并比较乱序超标量与顺序 5 级流水线的 CPI

---sys_defs.vh　　定义架构设计的顶层配置参数，无需改动

---test_progs　　包含所有用于测试验证的 MIPS RISC-Style 汇编代码

---testbench　　包含项目所需的所有验证 tb 文件和可视化支持文件

---verilog　　包含项目所需的所有架构设计 Verilog 源文件，具体解释详见课程网站 Project 部分

**4、CLAB 使用教程**

请参考课程网站 Project 部分的 CLAB 使用教程，完成 CLAB 的登录与配置工作。

**5、Linux 常见指令**

参考课程网站 Linux 使用内容。