SUSTech Southern University of Science and Technology

SME309 – Microprocessor Design

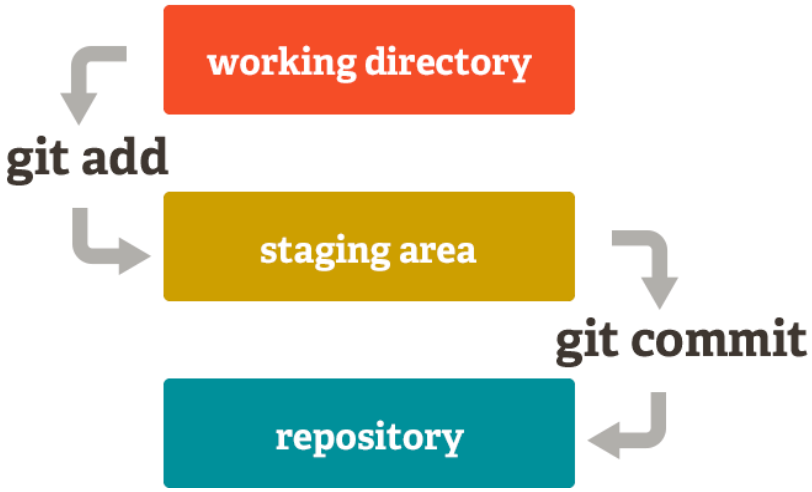
Project Intro

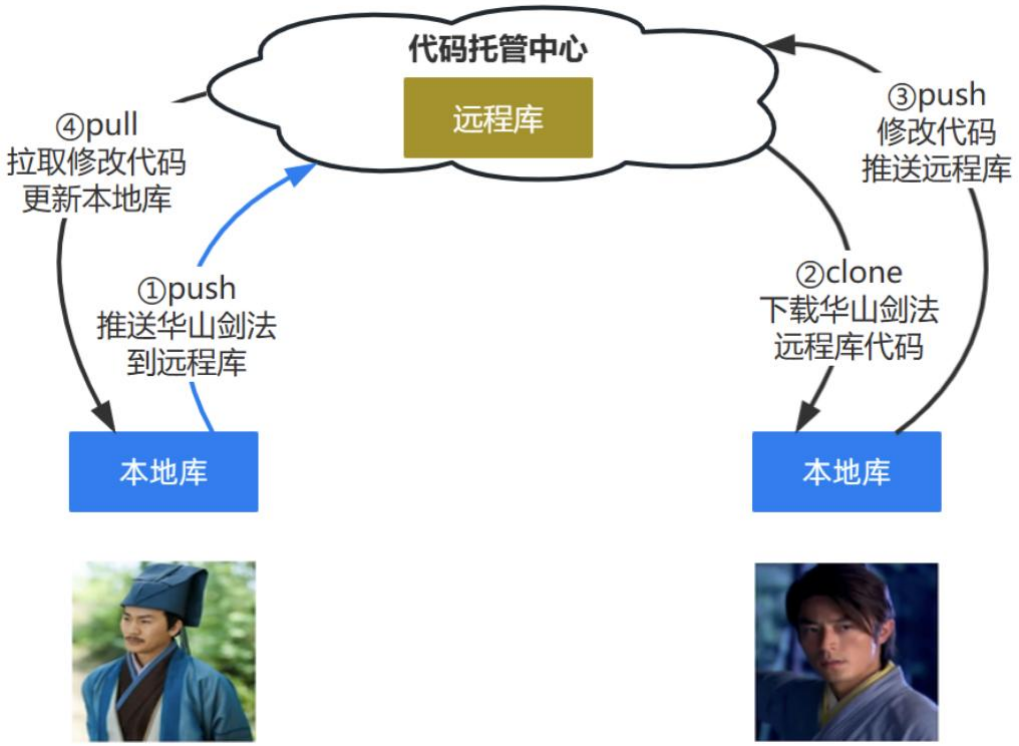
Project Intro / Outline

SUSTech Southern University of Science and Technology

- Git (Bonus)
- RISC-V
- Hints
- Lab Review & Summary

Git / 代码管理





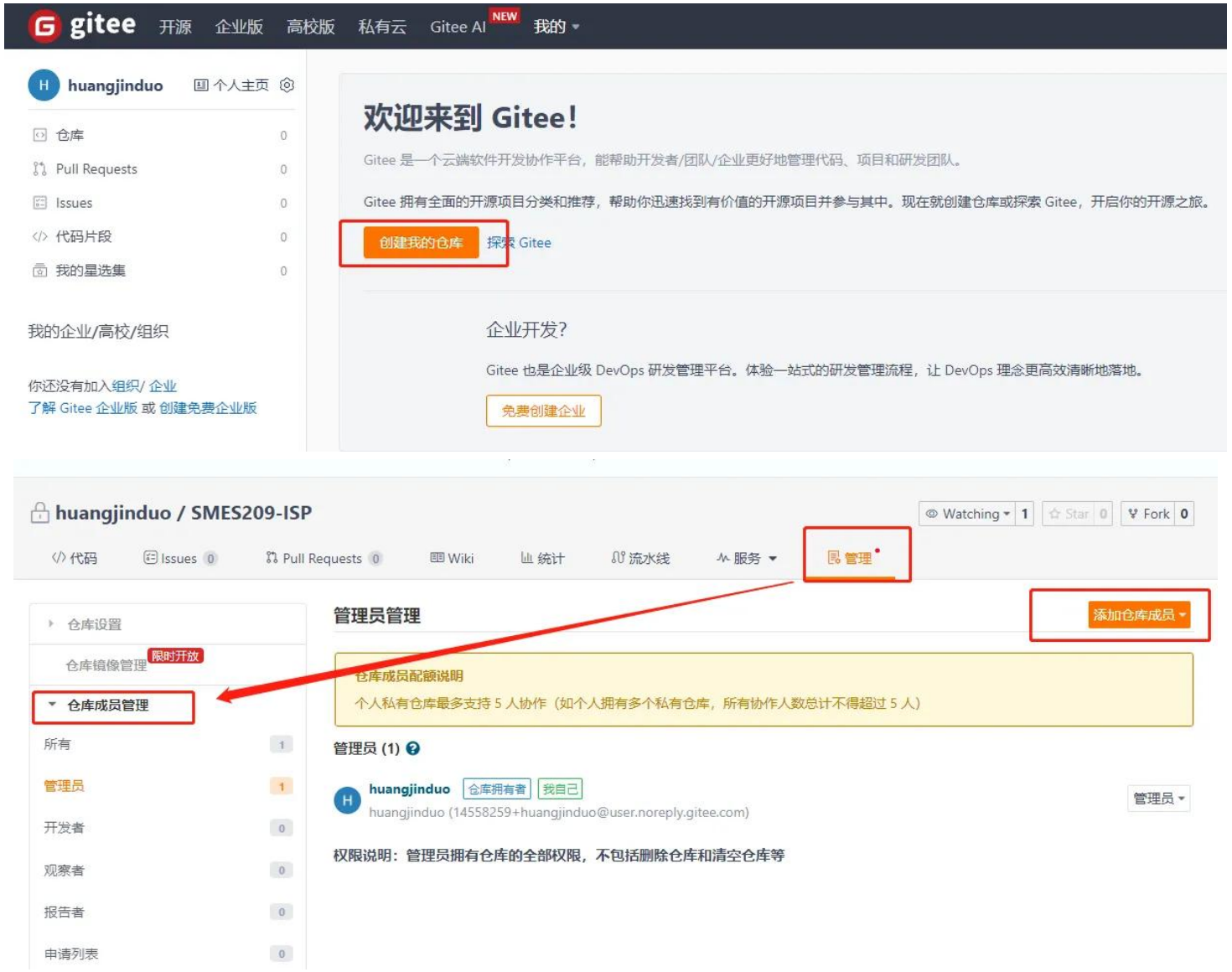
https://www.yuque.com/yuqueyonghuqmmgtj/ohkzvd/hg81irip3yr2f7xy?singleDoc#

《Git 安装与使用说明》 密码：xotu

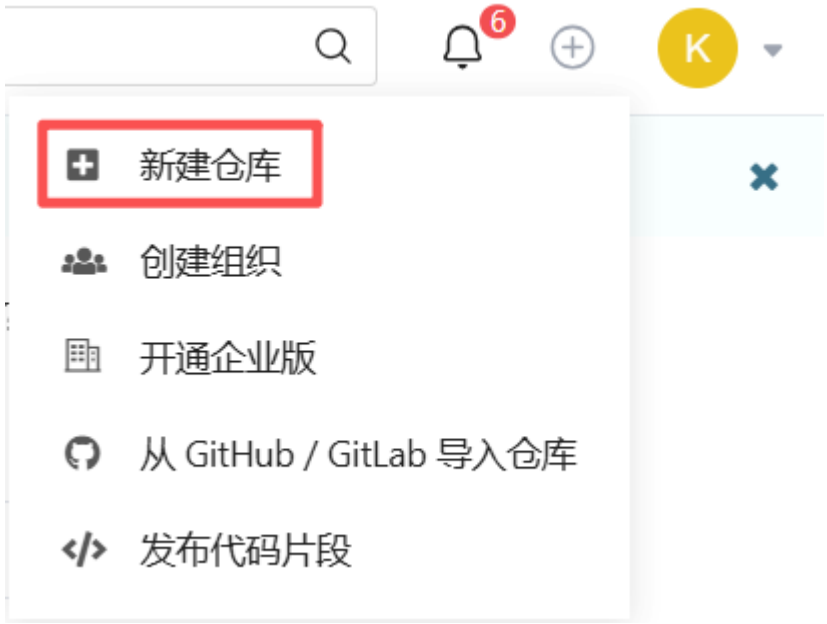
Git / 代码管理

团队合作：gitee / github

- 队长创建代码仓库（私有，不可开源）
- 邀请队友加入 git 仓库



Git / 创建代码库



Step1, 在 github/gitee 创建代码库

注意：设置为私有，不可开源!

Git / 创建代码库

Step1, 在 github/gitee 创建代码库

创建成功后，页面提示下一步提交代码

Git入门? 查看 帮助 , Visual Studio / TortoiseGit / Eclipse / Xcode 下如何连接本站, 如何导入仓库

简易的命令行入门教程:

Git 全局设置:

```
git config --global user.name "kelvinkong"
git config --global user.email "12260936+kelvinkong@user.noreply.gitee.com"
```

创建 git 仓库:

```
mkdir SME309-RISCV
cd SME309-RISCV
git init
touch README.md
git add README.md
git commit -m "first commit"
git remote add origin https://gitee.com/kelvinkong/SME309-RISCV.git
git push -u origin "master"
```

已有仓库?

```
cd existing_git_repo
git remote add origin https://gitee.com/kelvinkong/SME309-RISCV.git
git push -u origin "master"
```

Git / 创建代码库

SUSTech Southern University of Science and Technology

```
PS D:\work\SME309-RISCV-tmpl> git init
Initialized empty Git repository in D:/work/SME309-RISCV-tmpl/.git/
PS D:\work\SME309-RISCV-tmpl> git remote add origin https://gitee.com/kelvinkong/SME309-RISCV.git
```

Step2, 使用 powershell 进入代码目录，依次执行

git init

git remote add origin …. (即执行上一步的提示，见上一页)

Git / 创建代码库

SUSTech Southern University of Science and Technology

Step3, 创建首个提交 (commit), 继续在 powershell 中依次执行

git add .

git commit –m “xxxx”

```
PS D:\work\SME309-RISCV-tmpl> git add .
warning: in the working copy of 'pds/top.fdc', LF will be replaced by CRLF the
warning: in the working copy of 'pds/top.v', LF will be replaced by CRLF the ne
warning: in the working copy of 'rtl/benchmark.v', LF will be replaced by CRLF
warning: in the working copy of 'rtl/dram_driver.v', LF will be replaced by CRL
PS D:\work\SME309-RISCV-tmpl> git commit -m "setup"
[master (root-commit) fa7419d] setup
 69 files changed, 12269 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 README.md
 create mode 100644 pds/RISCV.2022.pds
 create mode 100644 pds/RISCV.2023.pds
 create mode 100644 pds/RISCV.lite.pds
 create mode 100644 pds/impl.tcl
 create mode 100644 pds/ipcore/DRAM/.last_generated
```

Git / 创建代码库

SUSTech Southern University of Science and Technology

```
PS D:\work\SME309-RISCV-tmpl> git log
commit fa7419d607d90b3870891980aa8aa588749de96c (HEAD -> master, origin/master)
Author: KelvinKong <jia.w86@163.com>
Date:   Mon Nov 17 14:13:30 2025 +0800

    setup
```

Step3, 创建首个提交 (commit), 继续在 powershell 中依次执行

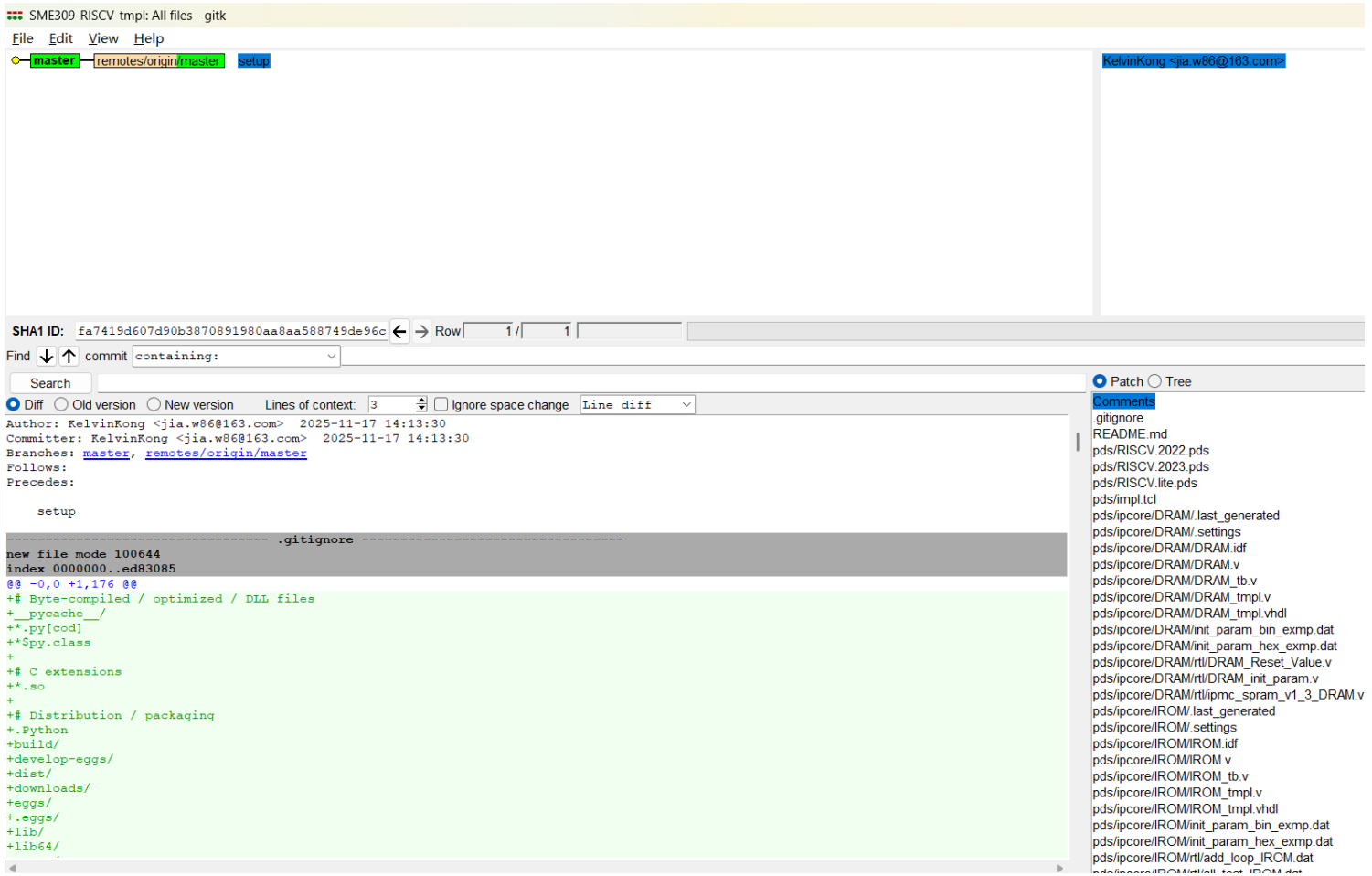
注意：创建成功后，可以通过 git log 查看提交信息

Git / 创建代码库

```
PS D:\work\SME309-RISCV-tmpl> gitk
```

Step3, 创建首个提交 (commit), 继续在 powershell 中依次执行

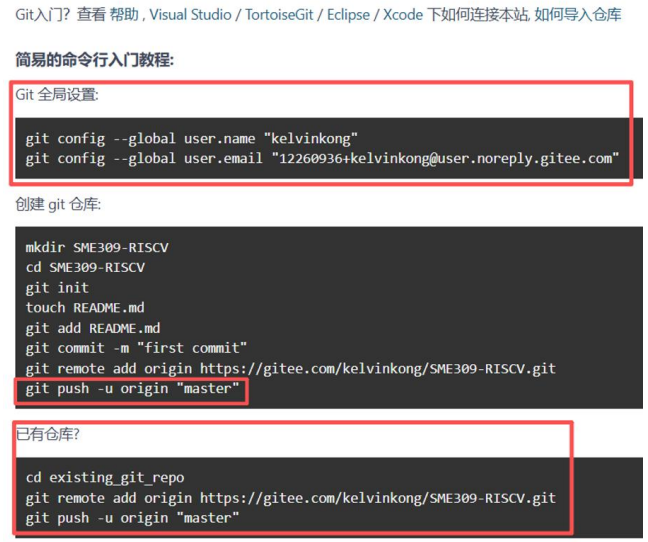
注意：创建成功后，也可以通过 gitk 查看提交信息



Git / 创建代码库

Step4, 将提交 (commit)同步到远程服务器, 继续在 powershell 中执行

git push –u origin “master”(即执行前面提示的最后一步，如右图)

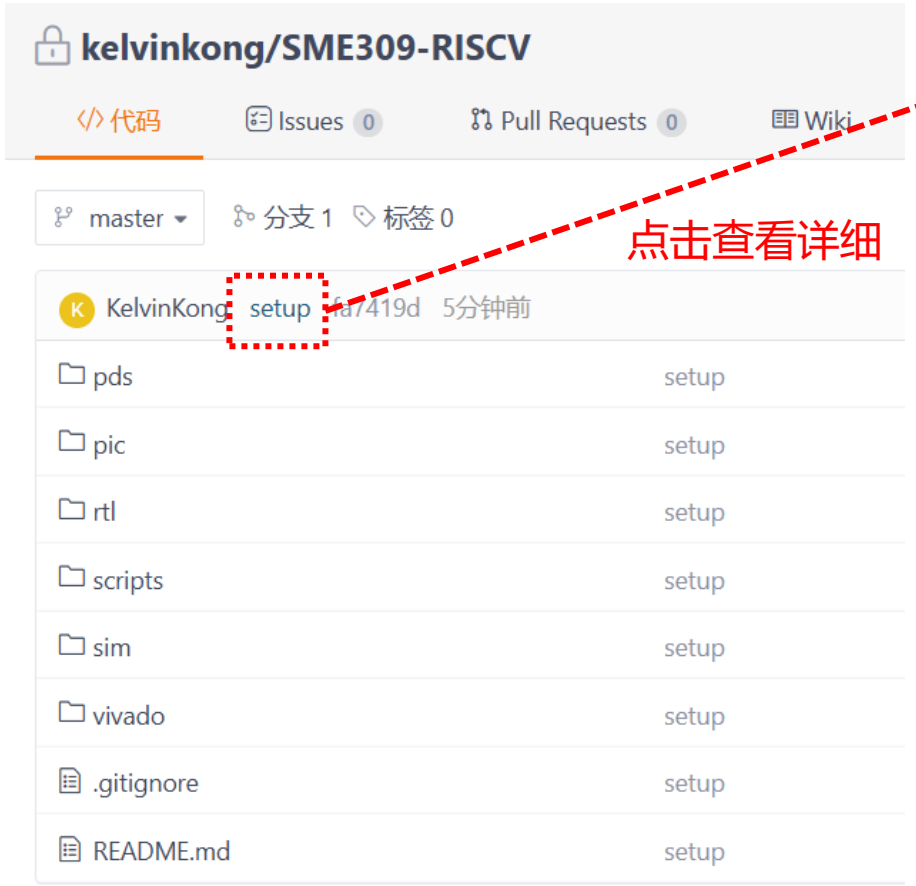



```
PS D:\work\SME309-RISCV-tmpl> git push -u origin "master"
Enumerating objects: 84, done.
Counting objects: 100% (84/84), done.
Delta compression using up to 16 threads
Compressing objects: 100% (78/78), done.
Writing objects: 100% (84/84), 94.38 KiB | 6.74 MiB/s, done.
Total 84 (delta 18), reused 0 (delta 0), pack-reused 0
remote: Powered by GITEE.COM [1.1.23]
remote: Set trace flag 894597b2
To https://gitee.com/kelvinkong/SME309-RISCV.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

Git / 创建代码库

Step4, 将提交 (commit)同步到远程服务器

注意：执行成功后，github/gitee平台将有代码更新！



点击查看详细

Git / 代码管理

常见命令

⇒ cd <工程目录>

⇒ git init (git项目初始化)

⇒ git add . (将当前变化缓存到 待提交commit 中)

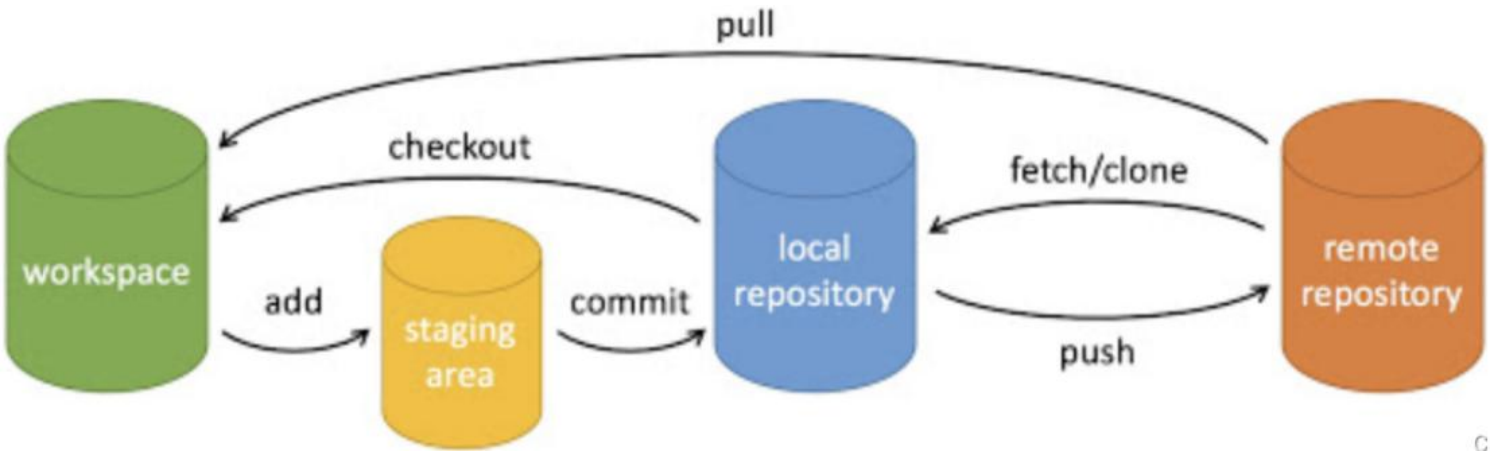
⇒ git commit -m “xxxxxx” （代码提交, 形成一个commit）

⇒ git status (查看哪些文件有变化)

⇒ git diff （查看具体文件有哪些变化）

⇒ git push origin master:master (提交本地代码到服务器)

⇒ git pull (拉取服务器最新代码到本地)

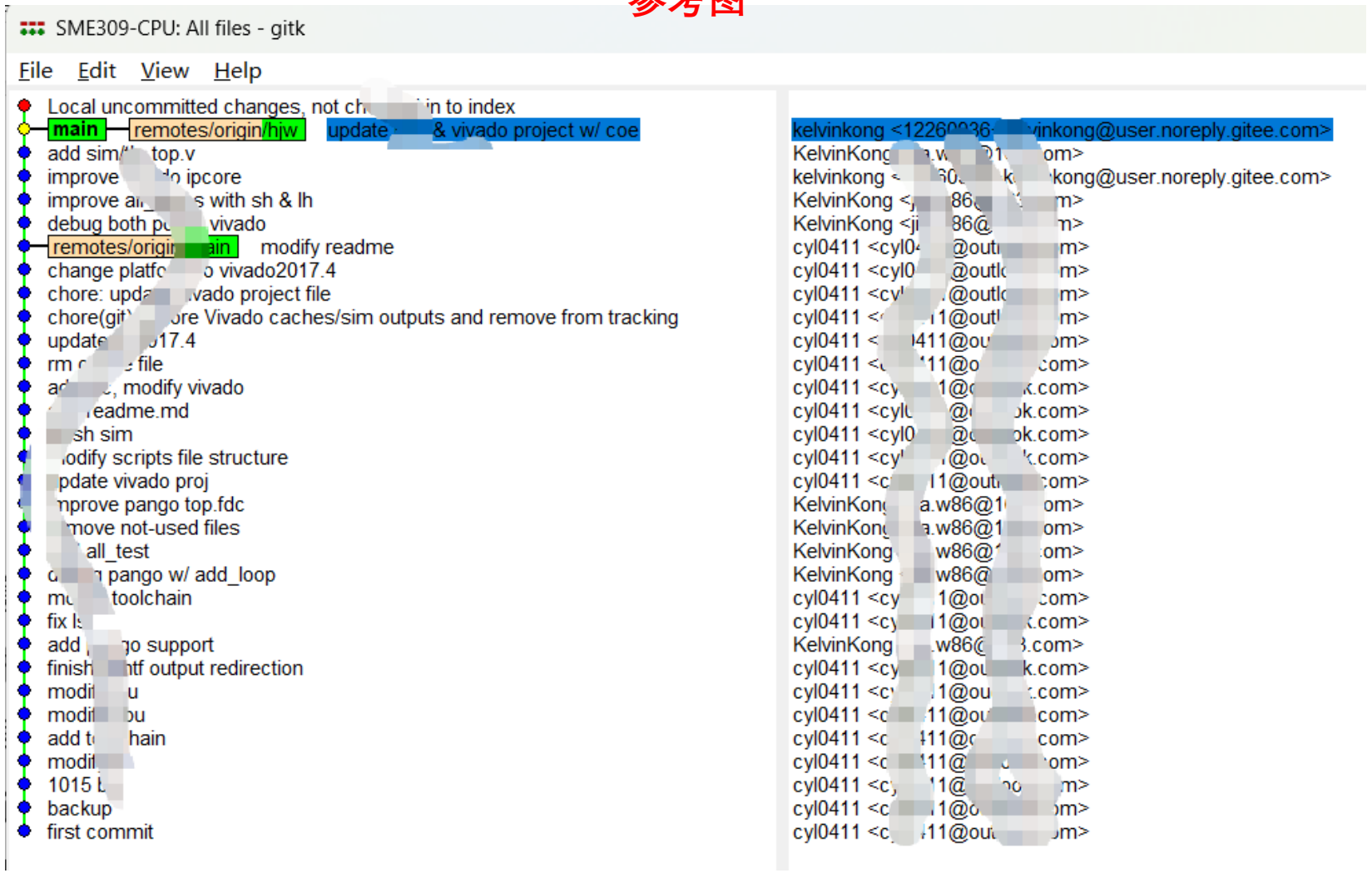


Git / Bonus 说明

注意：

- 提交代码压缩包需包含 .git 隐藏目录，即可通过 gitk 查看完整的提交记录 （如右图）
- 报告中有提交记录的截图，包含多人协作开发过程

参考图



Project Intro / Outline

SUSTech Southern University of Science and Technology

- Git
- **RISC-V**
- Hints
- Lab Review & Summary

RISC-V / 问题

SUSTech Southern University of Science and Technology

- **2. Design and Benchmarking of a RISC-V Processor [60 points]**

In this task, you will implement a CPU core to support simple RISC-V ISA. Your design should support basic instructions defined in the RV32I extension (Refer to RISC-V reference manual for more details), which is the minimal implementation for a RISC-V CPU.

Upon completion, we will provide a benchmark suite consisting of functional tests and a performance benchmarking program. You must first ensure your design passes the functional tests to verify correctness. Once verified, the benchmarking program will assess your CPU's performance; higher scores will directly contribute to a better grade.

To achieve superior performance, consider implementing advanced architectural features such as **Pipeline**, **Cache**, and **Dynamic Branch Prediction**. For further details, please refer to the **Appendix**.

RISC-V / 背景

SUSTech Southern University of Science and Technology

概览

- 加州大学伯克利分校 RISC ISA 设计的第五版，始于 2010 年
- 设计理念：简洁、高效
- 开源：BSD协议，允许设计者在不受任何专利限制的情况下自由创新

RISC-V基金会:
- 成立 2015 年
- 覆盖50 个国家和地区，
- 拥有约 1,000 多个会员

RISC-V / 背景

SUSTech Southern University of Science and Technology

RISC / 精简指令集：ARM vs RISC-V

- 架构篇幅：几千页 vs ~300页
- 模块化、可扩展性：不支持 vs 支持
- 指令数量：繁多(分支之间不兼容) vs 基础40~50(扩充其他常用不过百)
- 易实现性：复杂度高 vs 简单
- 两者差距主要体现：上下游产业生态和规模化应用，ARM统治移动端领域

RISC-V / RV32I 指令

- 模块化指令集

- Base 指令集:
  - RV32I / RV64I / RV128I – 32/64/128位整数
  - RV32E – 针对嵌入式

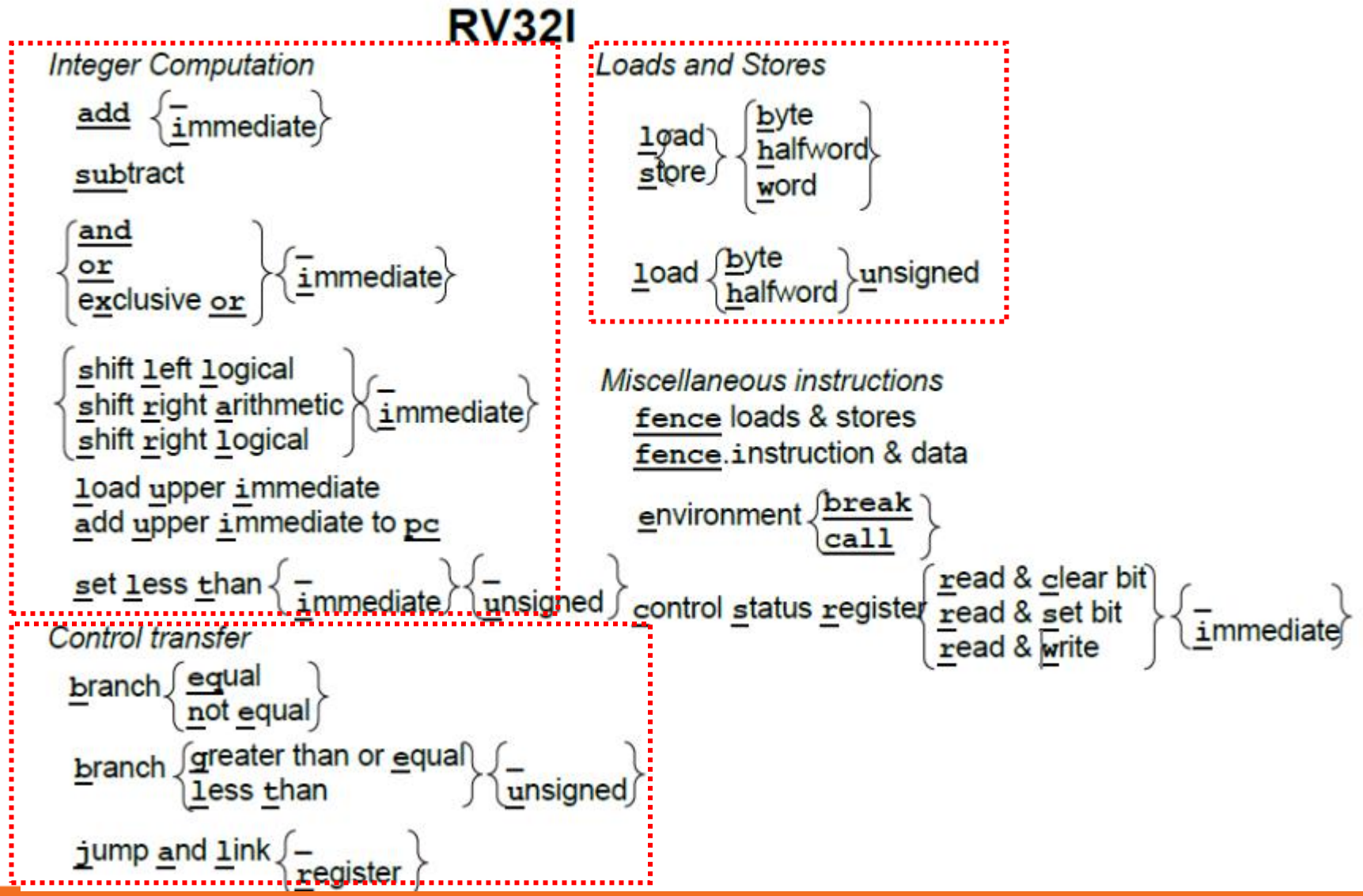
- Extension 指令集
  - M – 整数乘除法
  - F, D, Q – 单、双、四精度浮点运算
  - …

Base	Version	Status
RVWMO	2.0	Ratified
RV32I	**2.1**	**Ratified**
RV64I	**2.1**	**Ratified**
RV32E	*1.9*	*Draft*
RV128I	*1.7*	*Draft*
Extension	**Version**	**Status**
Zifencei	**2.0**	**Ratified**
Zicsr	**2.0**	**Ratified**
M	**2.0**	**Ratified**
A	*2.0*	Frozen
F	**2.2**	**Ratified**
D	**2.2**	**Ratified**
Q	**2.2**	**Ratified**
C	**2.0**	**Ratified**
Ztso	*0.1*	*Frozen*
Counters	*2.0*	*Draft*
L	*0.0*	*Draft*
B	*0.0*	*Draft*
J	*0.0*	*Draft*
T	*0.0*	*Draft*
P	*0.2*	*Draft*
V	*0.7*	*Draft*
N	*1.1*	*Draft*
Zam	*0.1*	*Draft*

SUSTech Southern University of Science and Technology

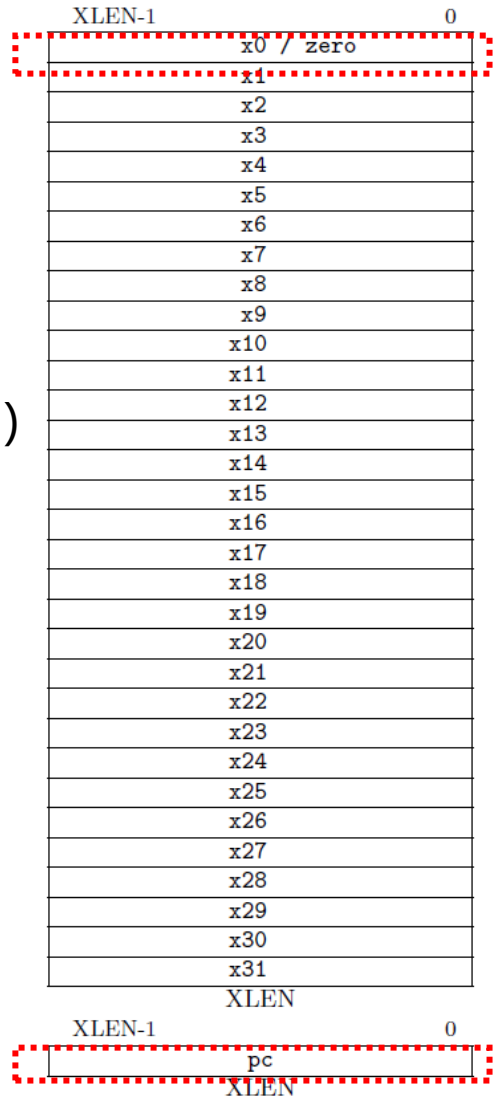
RISC-V / RV32I 指令

- 思想：最小实现
  - 足够支持上层编译器和现代操作系统构建
- 精简：47条独立的指令
  - 系统指令：9条 (Skip)



RISC-V / RV32I 指令

- 32个寄存器 + PC
  - 指令中 5 bits 地址
- 方便阅读/编程 -> ABI (程序二进制接口)
  - 应用于：汇编



Register	ABI Name	Description
x0	zero	Hard-wired zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5	t0	Temporary/alternate link register
x6–7	t1–2	Temporaries
x8	s0/fp	Saved register/frame pointer
x9	s1	Saved register
x10–11	a0–1	Function arguments/return values
x12–17	a2–7	Function arguments
x18–27	s2–11	Saved registers
x28–31	t3–6	Temporaries

Figure 2.1: RISC-V base unprivileged integer register state.

RISC-V / RV32I 指令

6 种指令格式

- 用于寄存器-寄存器的操作：R-type
- 用于短立即数-寄存器的操作：I-type, S-type, B-type
  - I-type，含有：load 操作
  - S-type，关键：store 操作
  - B-type，关键：跳转 （branch）
- 用于长立即数：U-type，J-type

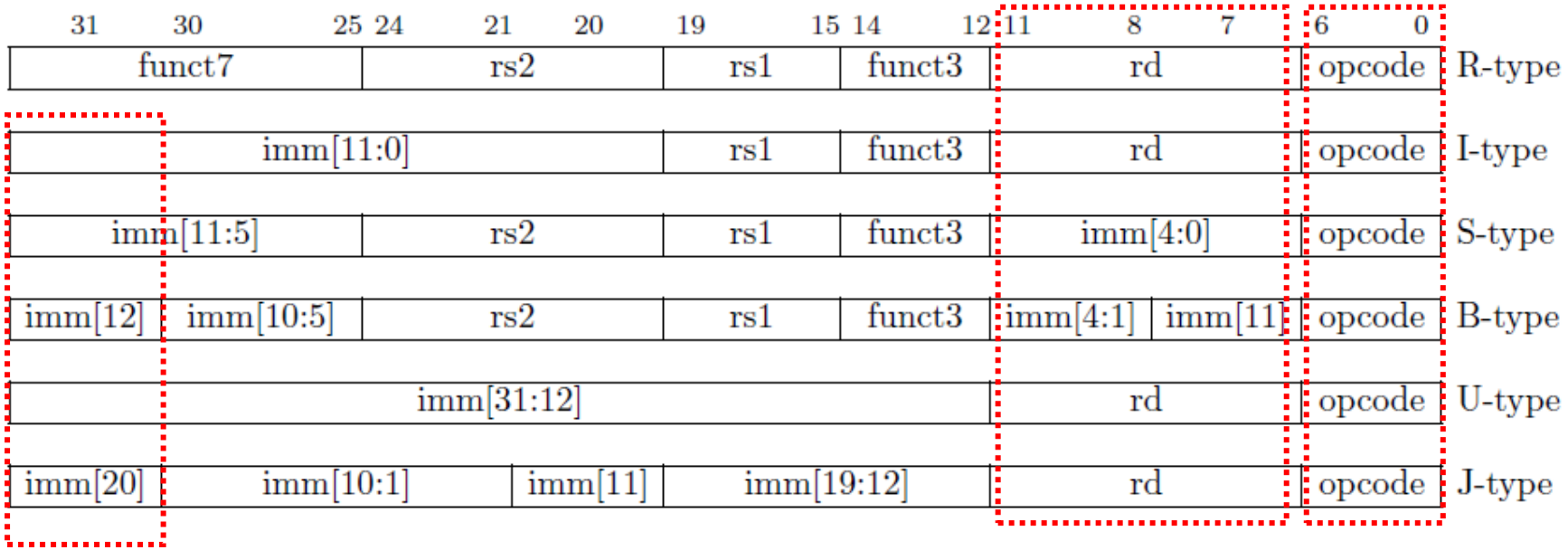


Figure 2.3: RISC-V base instruction formats showing immediate variants.

注意：

- opcode 与 指令格式的关系？
- rd / rs1 / rs2 位置的共性
- imm sign bit 共性
- 条件(cond)字段？

RISC-V / RV32I 指令

深入看例子

- Opcode = 011_0011
- Opcode = 001_0011
  - DP 指令
- Opcode = 000_0011
- Opcode = 010_0011
  - 内存读/写指令
- Opcode = 110_0011
  - 跳转指令

...

RV32I Base Integer Instructions

Inst	Name	FMT	Opcode	funct3	funct7	Description (C)	Note
add	ADD	R	0110011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0110011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0110011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0110011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0110011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0110011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0110011	0x5	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0110011	0x5	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2	0x00	rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3	0x00	rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0		rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x4		rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x6		rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x7		rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	imm[5:11]=0x00	rd = rs1 << imm[0:4]	
srli	Shift Right Logical Imm	I	0010011	0x5	imm[5:11]=0x00	rd = rs1 >> imm[0:4]	
srai	Shift Right Arith Imm	I	0010011	0x5	imm[5:11]=0x20	rd = rs1 >> imm[0:4]	msb-extends
slti	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltiu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≥	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1 + imm	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	imm=0x0	Transfer control to OS	
ebreak	Environment Break	I	1110011	0x0	imm=0x1	Transfer control to debugger	

RISC-V / 汇编 ->Test Code

Coding: 测试汇编程序

参考代码模板的 scripts/src/add_loop.s (如右图)

思考：**如何转化为机器码?**

add_loop.s

scripts > src > add_loop.s

```
# 程序：计算 1 到 1000 的累加和，并存入内存地址 0x90000000
# 使用的寄存器：
#   t0: 循环计数器 (i)
#   t1: 累加和 (sum)
#   t2: 常数 1000
#   t3: 地址 0x90000000

    .text
    .global _start

_start:
    # 初始化寄存器
    li t0, 1          # t0 = i = 1
    li t1, 0          # t1 = sum = 0
    li t2, 1000       # t2 = 1000 (循环上限)
    lui t3, 0x90000   # t3 = 0x90000000 (设置高20位)
    addi t3, t3, 0    # t3 = t3 + 0 (低12位为0，确保地址完整)
    lui t4, 0xffff0   # t3 = 0xffff0000 (设置高20位)

loop:
    bgt t0, t2, done  # 如果 i > 1000，跳转到 done
    add t1, t1, t0    # sum = sum + i
    addi t0, t0, 1    # i = i + 1
    j loop            # 跳转回 loop

done:
    sw t1, 0(t3)      # 将 sum 写入地址 0x90000000
    sw t3, 12(t3)      # 将 sum 写入地址 0x9000000C
    # （可选）程序结束，可通过断点或仿真器停止
    # 在实际嵌入式环境中，可能需要进入死循环或触发中断
    j done            # 停留在此处
```

RISC-V / 汇编 ->Test Code

SUSTech Southern University of Science and Technology

如何转化为机器码?

=> 工具链：**riscv-none-embed-gcc**

https://github.com/ilg-archived/riscv-none-gcc

GNU MCU Eclipse RISC-V Embedded GCC

Rationale

GNU MCU Eclipse RISC-V Embedded GCC is a GCC toolchain distribution for RISC-V devices that complements the official RISC-V distribution, maintained by SiFive.

For main benefits for the users are:

- convenience: binaries for all major platforms are provided (Windows 64/32-bits, GNU/Linux 64/32-bits, macOS);
- uniform and portable install: the toolchain is also available as a binary xPack, and can be easily installed with `xpm`;
- improved support for Continuous Integration usage: as for any xPack, the toolchain can be easily used in test environments.

下载地址:
https://pan.baidu.com/s/1x0D_5pliOLB59QpCbiZDcw?pwd=sia5
提取码: sia5

二选一即可

RISC-V / 汇编 ->Test Code

如何转化为机器码?

=> 工具链：**riscv-none-embed-gcc**

=> xxx-**as**.exe: .s 生成 .o

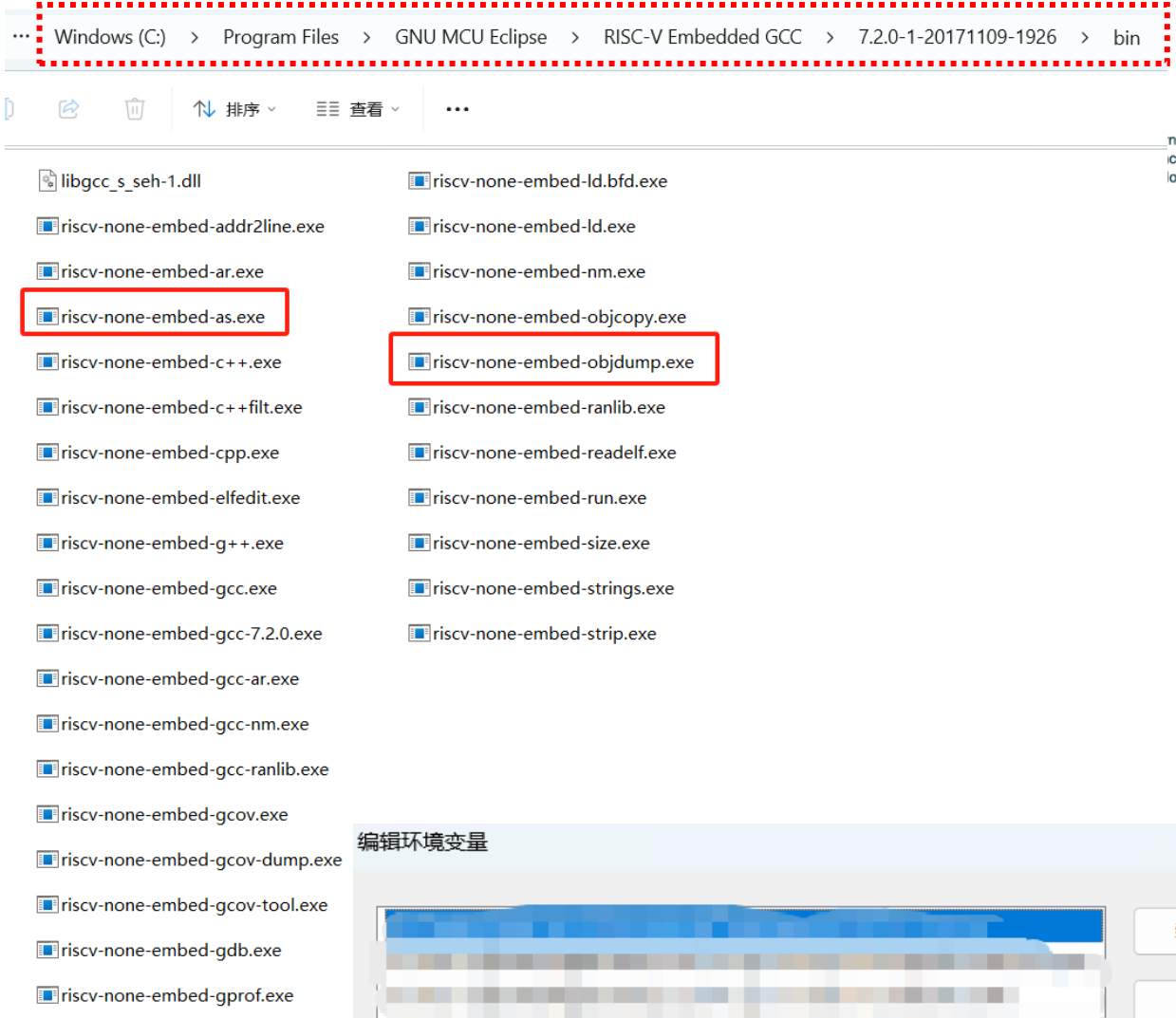
=> xxx-**objcopy**.exe: .o 转为为 二进制机器码

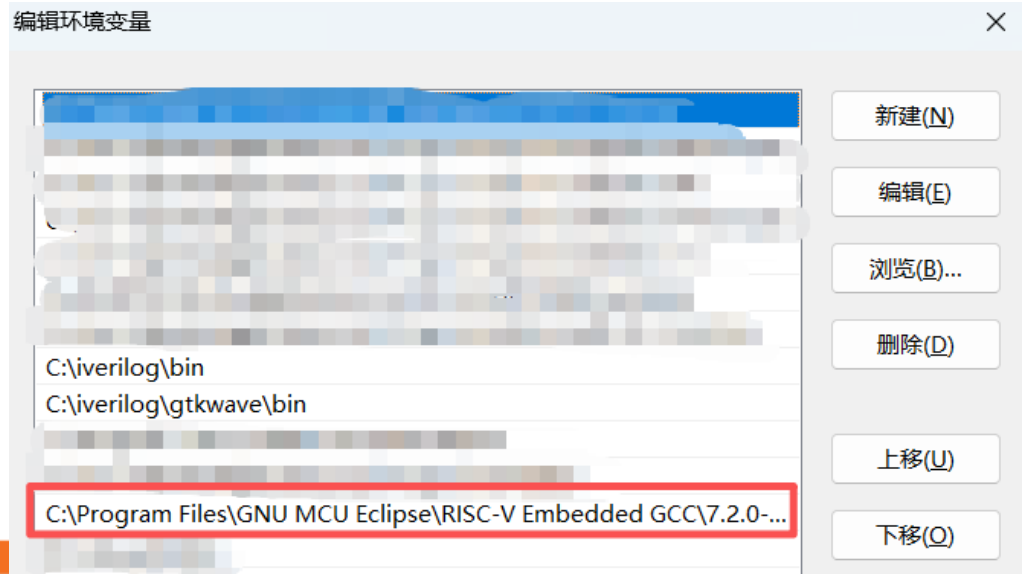
=> xxx-**objdump**.exe: .o 提取 更可读的机器码

或者工具链：**riscv64-unknown-elf-gcc**

注意：安装成功后，需要将该路径(xxx/bin)

添加到 系统 Path 变量中





RISC-V / 汇编 ->Test Code

SUSTech Southern University of Science and Technology

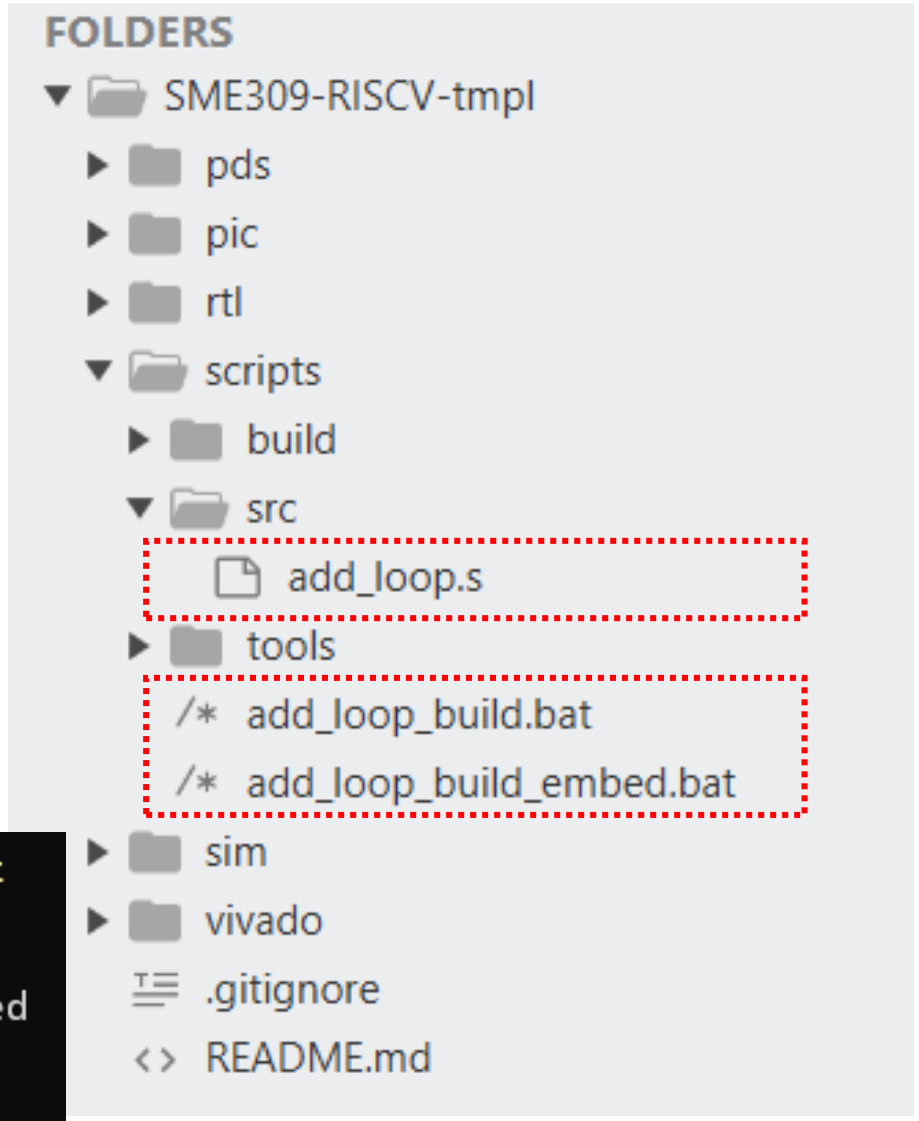
如何转化为机器码?

安装好 riscv gcc 工具链后，在代码模板 scripts 子目录点击 add_loop_build.bat 或 add_loop_build_embed.bat

针对工具链 riscv-none-embed-gcc ， 执行 add_loop_build_embed.bat

脚本会对 scripts/src/add_loop.s (参考汇编代码) 进行编译，

```
PS D:\work\SME309-RISCV-tmpl\scripts> .\add_loop_build_embed.bat
Building add_loop...
src\add_loop.s: Assembler messages:
src\add_loop.s: Warning: end of file in comment; newline inserted
Usage: python bin2coe.py <bin_file> <coe_file>
Usage: python bin2hex.py <bin_file> <dat_file>
Build complete! Output files are in the build\ directory.
```



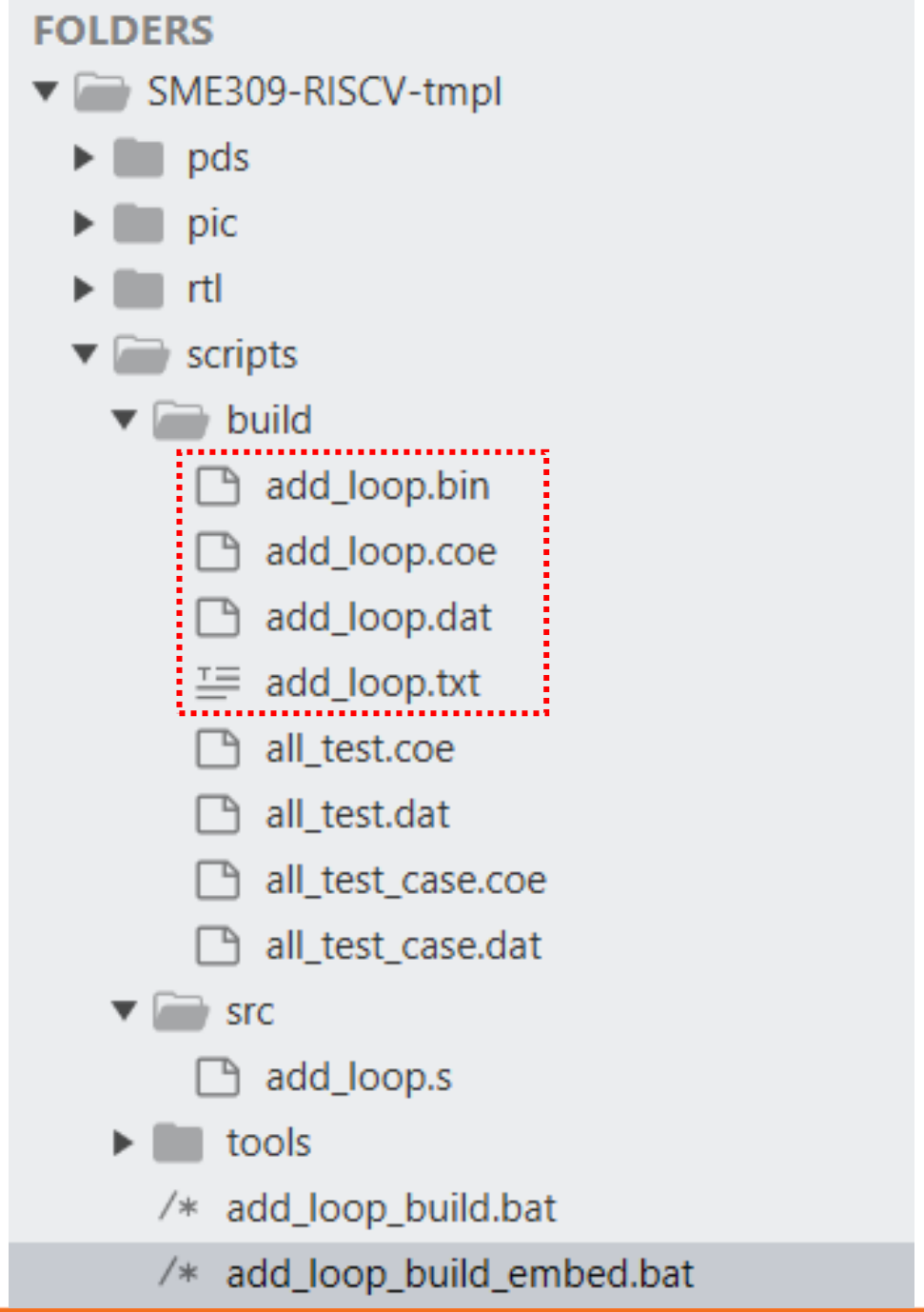
RISC-V / 汇编 ->Test Code

如何转化为机器码?

安装好 riscv gcc 工具链后，在代码模板 scripts 子目录点击 add_loop_build.bat 或 add_loop_build_embed.bat

针对工具链 riscv-none-embed-gcc ， 执行 add_loop_build_embed.bat

脚本会对 scripts/src/add_loop.s (参考汇编代码）进行编译，并**生成对应的二级制指令文件（在 scripts/build 子目录）**



RISC-V / 汇编 ->Test Code

add_loop.s

=> 可对该代码进行修改，用于后续测试

注意到：**li 是什么指令?**

add_loop.s

scripts > src > add_loop.s

```
1   # 程序：计算 1 到 1000 的累加和，并存入内存地址 0x90000000
2   # 使用的寄存器：
3   #   t0: 循环计数器 (i)
4   #   t1: 累加和 (sum)
5   #   t2: 常数 1000
6   #   t3: 地址 0x90000000
7
8       .text
9       .global _start
10
11  _start:
12      # 初始化寄存器
13      li t0, 1            # t0 = i = 1
14      li t1, 0            # t1 = sum = 0
15      li t2, 1000         # t2 = 1000 (循环上限)
16      lui t3, 0x90000     # t3 = 0x90000000 (设置高20位)
17      addi t3, t3, 0      # t3 = t3 + 0 (低12位为0，确保地址完整)
18      lui t4, 0xffff0     # t3 = 0xffff0000 (设置高20位)
19
20  loop:
21      bgt t0, t2, done    # 如果 i > 1000，跳转到 done
22      add t1, t1, t0      # sum = sum + i
23      addi t0, t0, 1      # i = i + 1
24      j loop              # 跳转回 loop
25
26  done:
27      sw t1, 0(t3)        # 将 sum 写入地址 0x90000000
28      sw t3, 12(t3)        # 将 sum 写入地址 0x9000000C
29      # （可选）程序结束，可通过断点或仿真器停止
30      # 在实际嵌入式环境中，可能需要进入死循环或触发中断
31      j done              # 停留在此处
```

RISC-V / 汇编 ->Test Code

SUSTech Southern University of Science and Technology

add_loop.txt

=> 方便对 RV32I 指令的理解

注意到：li 是什么指令?

如：li t0, 1

本质是：addi t0, x0, 1

```
00000000000000000 <_start>:
    0: 00100293           li  t0,1
    4: 00000313           li  t1,0
    8: 3e800393           li  t2,1000
    c: 90000e37           lui t3,0x90000
   10: 000e0e13           mv  t3,t3
   14: ffff0eb7           lui t4,0xffff0

0000000000000018 <loop>:
   18: 0053c863           blt t2,t0,28 <done>
   1c: 00530333           add t1,t1,t0
   20: 00128293           addi  t0,t0,1
   24: ff5ff06f           j 18 <loop>

0000000000000028 <done>:
   28: 006e2023           sw  t1,0(t3) # ffffffff90000000
   2c: 01ce2623           sw  t3,12(t3)
   30: ff9ff06f           j 28 <done>
```

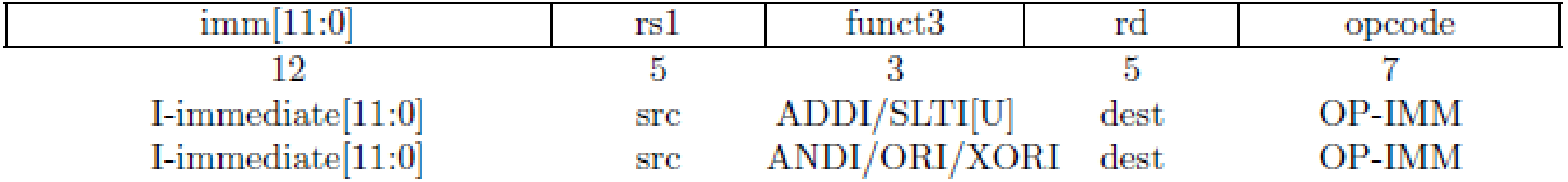
RISC-V / 指令分析 / addi

```
0:    00100293                li      t0,1
```

x5

Integer Register-Immediate Instructions

0000_0000_0001 _0000_0 000 _0010_1 001_0011



imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
I-immediate[11:0]	src	ADDI/SLTI[U]	dest	OP-IMM
I-immediate[11:0]	src	ANDI/ORI/XORI	dest	OP-IMM

RISC-V / 指令分析 / addi

```
0:    00100293                li      t0,1
```

x5

Integer Register-Immediate Instructions

0000_0000_0001 _0000_0 000 _0010_1 001_0011

31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
I-immediate[11:0]	src	ADDI/SLTI[U]	dest	OP-IMM
I-immediate[11:0]	src	ANDI/ORI/XORI	dest	OP-IMM

ADDI adds the sign-extended 12-bit immediate to register *rs1*. Arithmetic overflow is ignored and the result is simply the low XLEN bits of the result. ADDI *rd*, *rs1*, *0* is used to implement the MV *rd*, *rs1* assembler pseudoinstruction.

RISC-V / 指令分析 / add

SUSTech Southern University of Science and Technology

```
1c: 00530333                    add t1,t1,t0
```

x6 x5

Integer Register-Register Operations

RV32I defines several arithmetic R-type operations. All operations read the *rs1* and *rs2* registers as source operands and write the result into register *rd*. The *funct7* and *funct3* fields select the type of operation.

0000_000 0_0101 _0011_0 000 _0011_0 011_0011

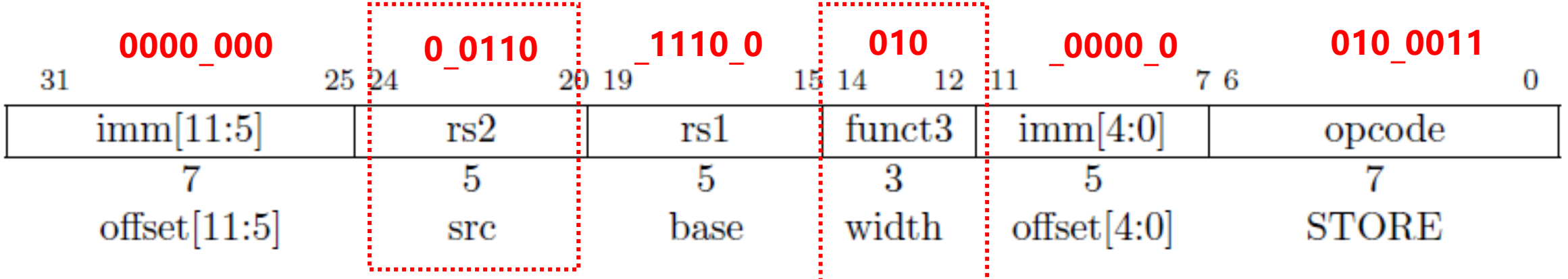
31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	src2	src1	ADD/SLT/SLTU	dest	OP
0000000	src2	src1	AND/OR/XOR	dest	OP
0000000	src2	src1	SLL/SRL	dest	OP
0100000	src2	src1	SUB/SRA	dest	OP

RISC-V / 指令分析 / store

```
28: 006e2023                    sw  t1,0(t3)
```

x6 x28

0000_000 0_0110 _1110_0 010 _0000_0 010_0011



imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW

RISC-V / 指令分析 / branch

x7 x5

```
18:  0053c863                  blt t2,t0,28 <done>
0000000000000028 <done>:
```

x5

0000_000 0_0101 _0011_1 100 _1000 _0 110_0011

31	30 25	24 20	19 15	14 12	11 8	7	6 0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
1	6	5	5	3	4	1	7
offset[12|10:5]		src2	src1	BEQ/BNE	offset[11|4:1]		BRANCH
offset[12|10:5]		src2	src1	BLT[U]	offset[11|4:1]		BRANCH
offset[12|10:5]		src2	src1	BGE[U]	offset[11|4:1]		BRANCH

imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU

RISC-V / 指令分析 / branch

x7 x5

```
18:  0053c863                  blt t2,t0,28 <done>
```

```
0000000000000028 <done>:
```

x5 x7

0000_000 0_0101 _0011_1 100 _1000 _0 110_0011

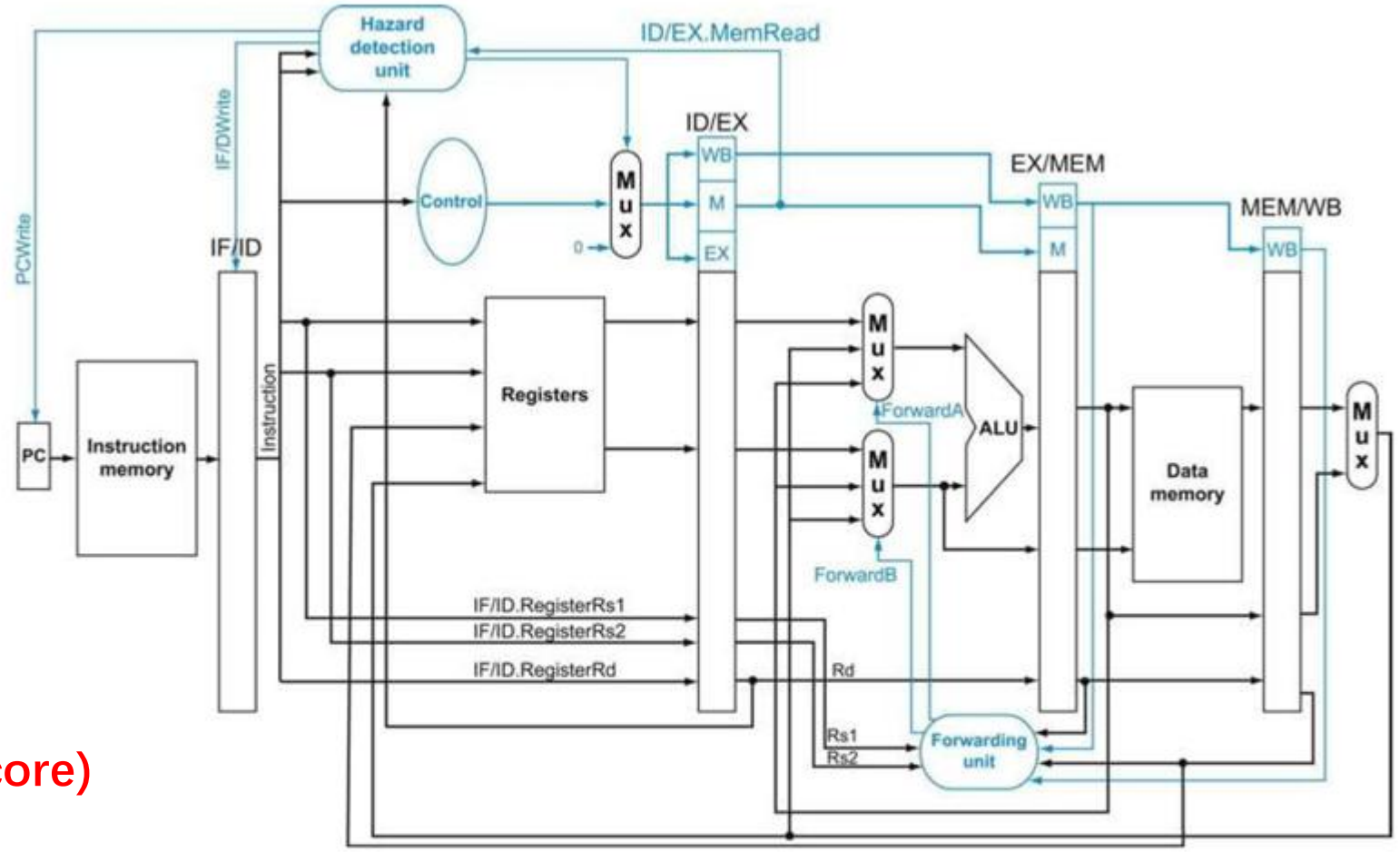
31	30 25	24 20	19 15	14 12	11 8	7	6 0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
1	6	5	5	3	4	1	7
offset[12|10:5]		src2	src1	BEQ/BNE	offset[11|4:1]		BRANCH
offset[12|10:5]		src2	src1	BLT[U]	offset[11|4:1]		BRANCH
offset[12|10:5]		src2	src1	BGE[U]	offset[11|4:1]		BRANCH

All branch instructions use the B-type instruction format. The 12-bit B-immediate encodes signed offsets in multiples of 2 bytes. The offset is sign-extended and added to the address of the branch instruction to give the target address. The conditional branch range is ±4 KiB.

PC += imm

RISC-V / 迭代思路

Pipeline RISC-V CPU

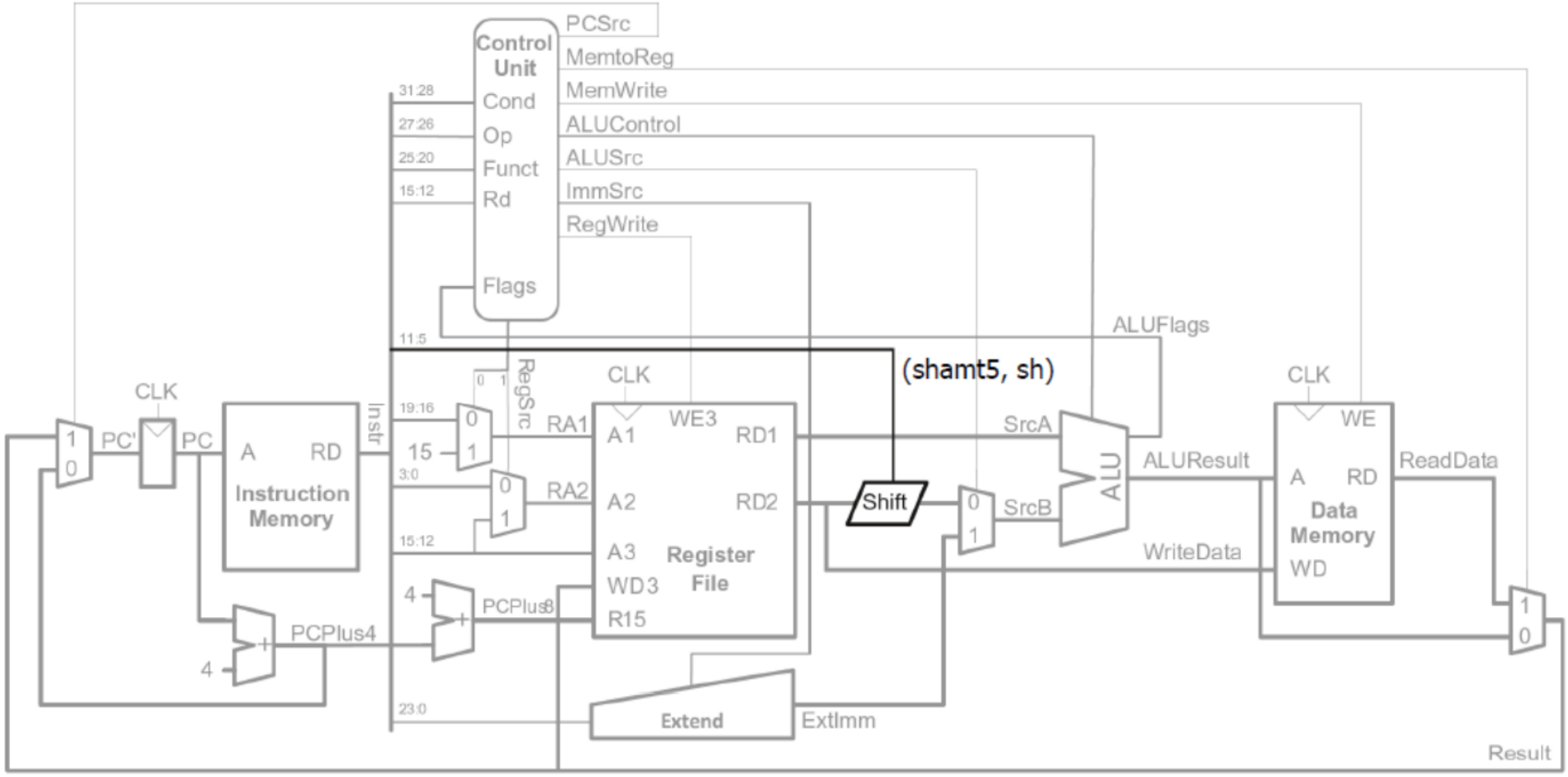


Single-cycle -> Pipelined (higher score)

The **pipelined RISC-V CPU** is derived from the single-cycle RISC-V core and implements the RV32I ISA while splitting instruction execution across five pipeline stages to enable higher instruction throughput. The five stages are: Instruction Fetch (**IF**), Instruction Decode / Register Fetch (**ID**), Execute (**EX**), Memory Access (**MEM**), and Write Back (**WB**).

RISC-V / Single-Cycle CPU，思路提示

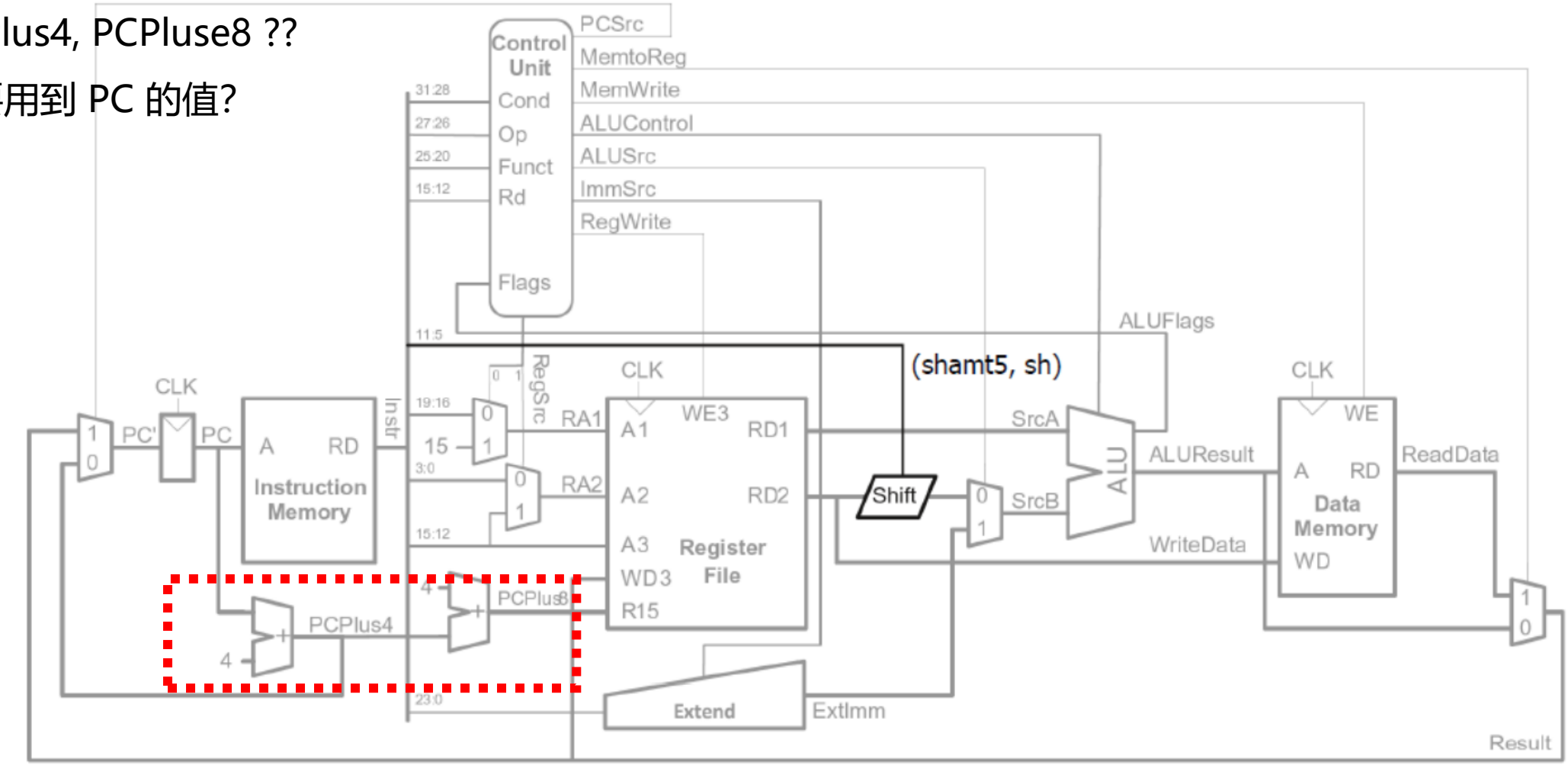
思考：需要改哪些地方?



RISC-V / Single-Cycle CPU，思路提示

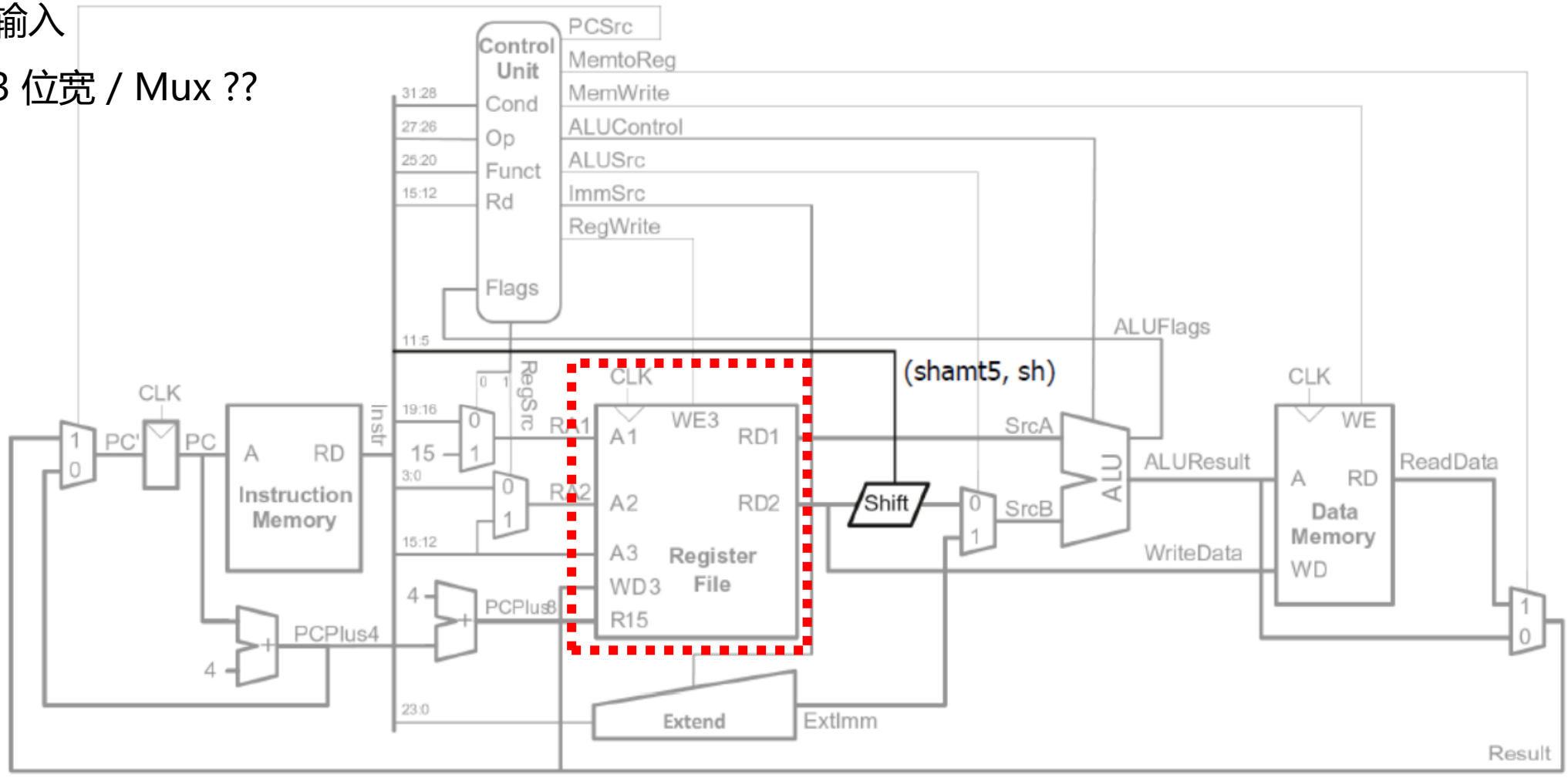
SUSTech Southern University of Science and Technology

- PC
  - 去掉 PCPlus4, PCPluse8 ??
  - ALU 需要用到 PC 的值?



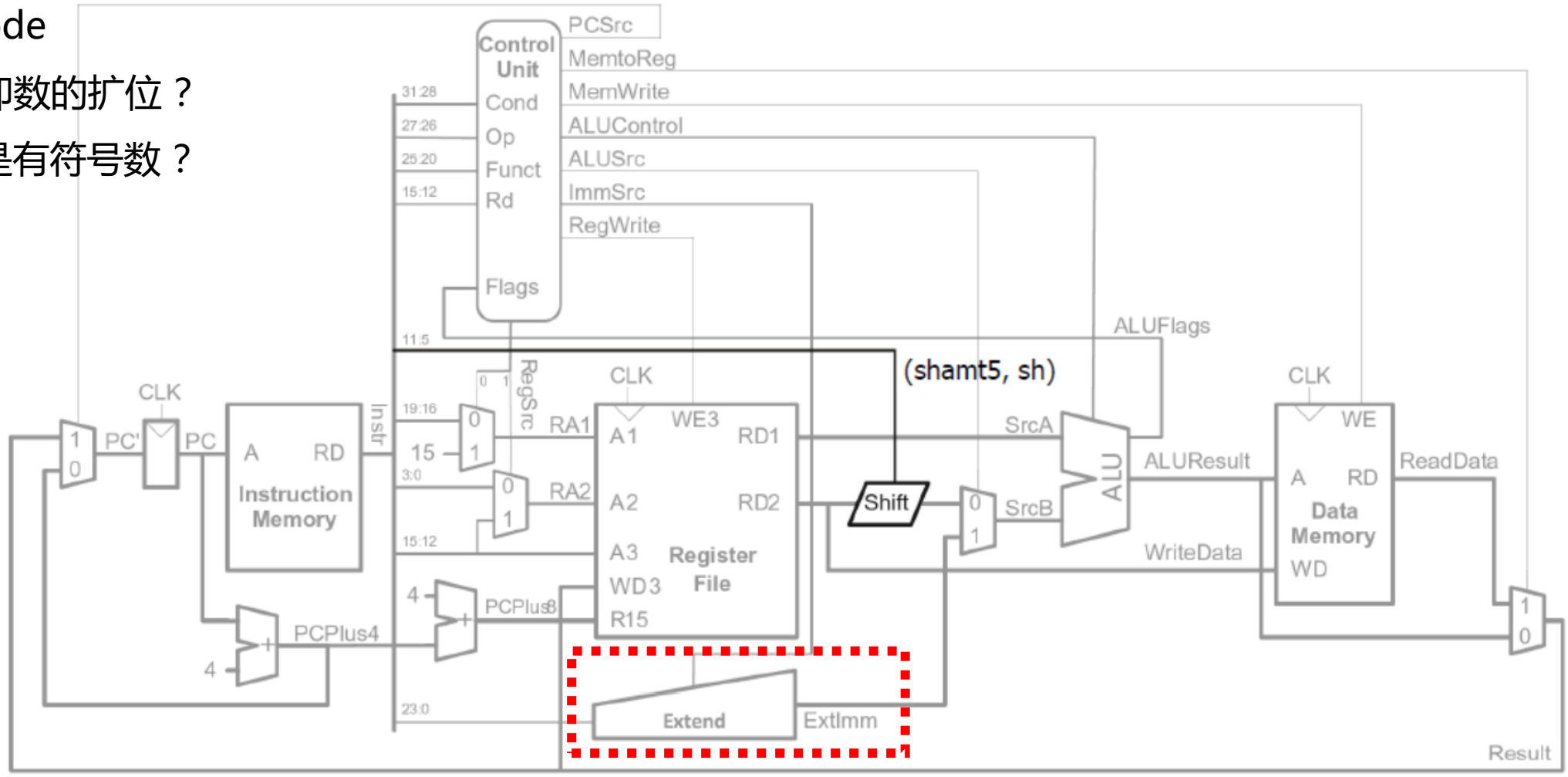
RISC-V / Single-Cycle CPU，思路提示

- 寄存器：16个 -> 32个
  - 去掉 R15 输入
  - A1, A2, A3 位宽 / Mux ??



RISC-V / Single-Cycle CPU，思路提示

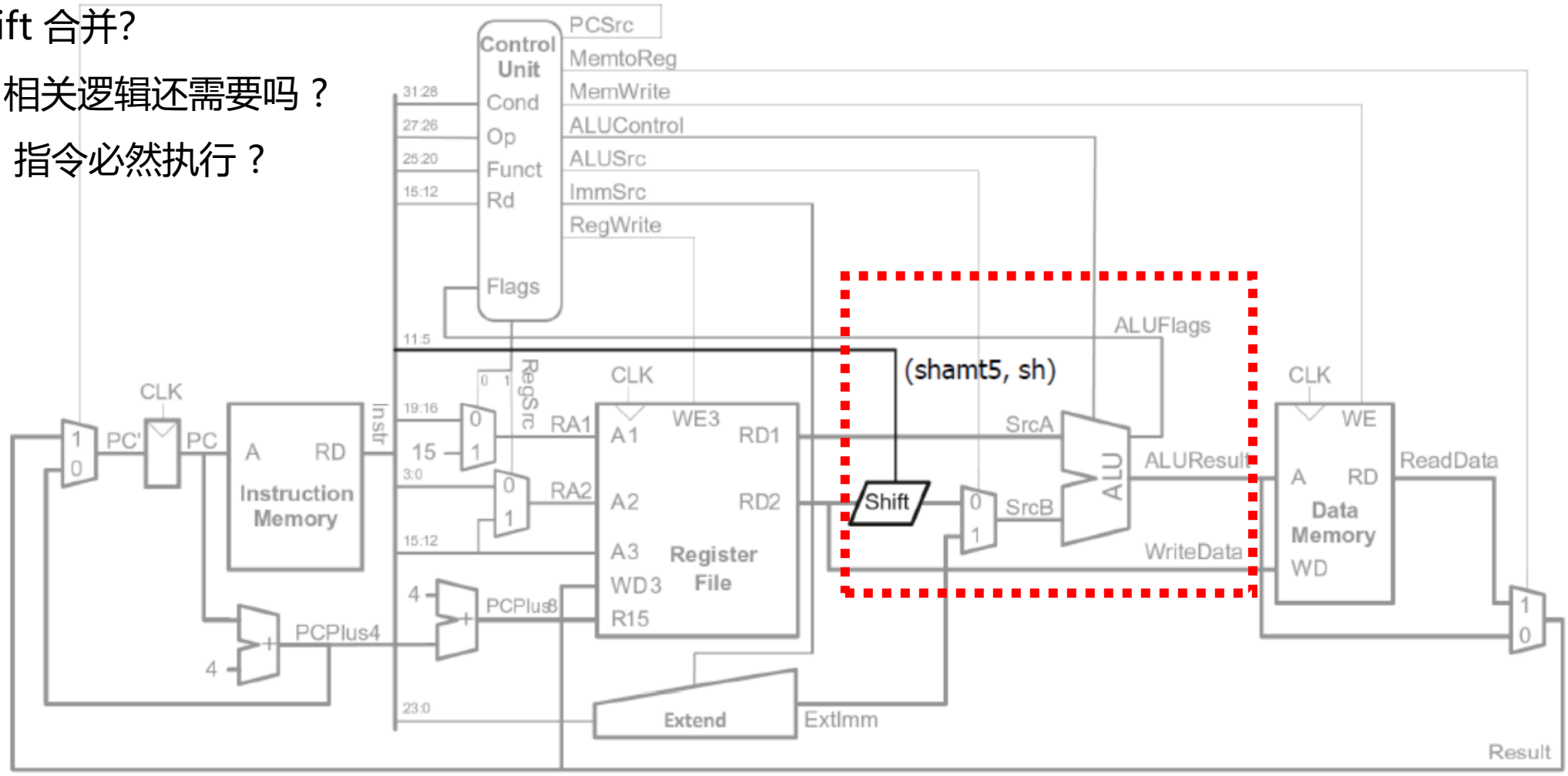
- Extend
  - 基于 opcode
  - 长 / 短立即数的扩位？
  - 哪些情况是有符号数？



RISC-V / Single-Cycle CPU，思路提示

SUSTech Southern University of Science and Technology

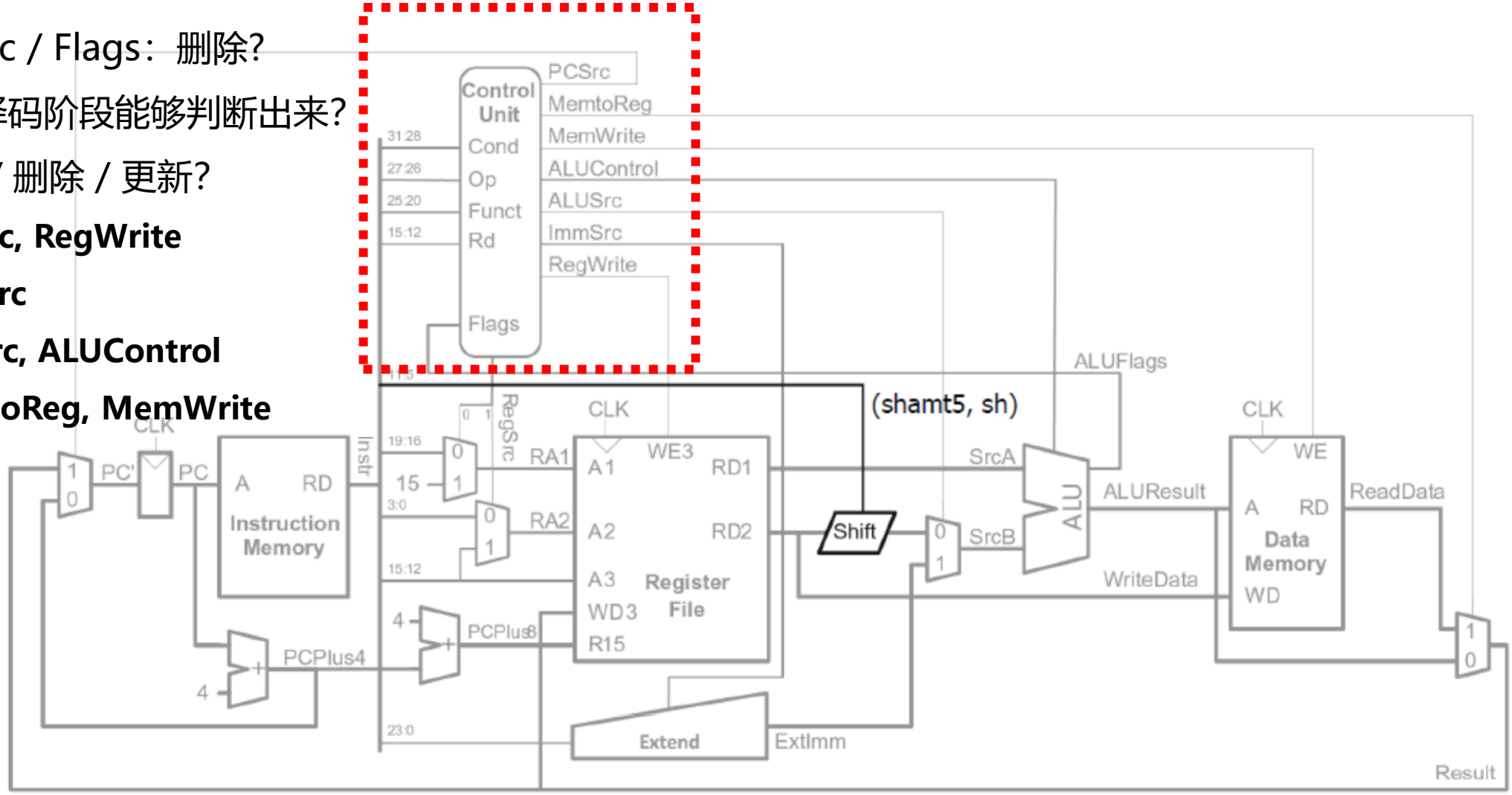
- 运算单元：
  - ALU + Shift 合并?
  - ALUFlags 相关逻辑还需要吗？
  - 条件判断：指令必然执行？



RISC-V / Single-Cycle CPU，思路提示

SUSTech Southern University of Science and Technology

- ControlUnit：
  - CondLogic / Flags：删除?
  - **PCSrc**：译码阶段能够判断出来?
  - 哪些保留 / 删除 / 更新?
    - **RegSrc, RegWrite**
    - **ImmSrc**
    - **ALUSrc, ALUControl**
    - **MemtoReg, MemWrite**



RISC-V / 感受：简约而不简单

SUSTech Southern University of Science and Technology

- ✓ **少而完备**：base + extension，模块化
- ✓ **整齐划一**：e.g. 寄存器索引

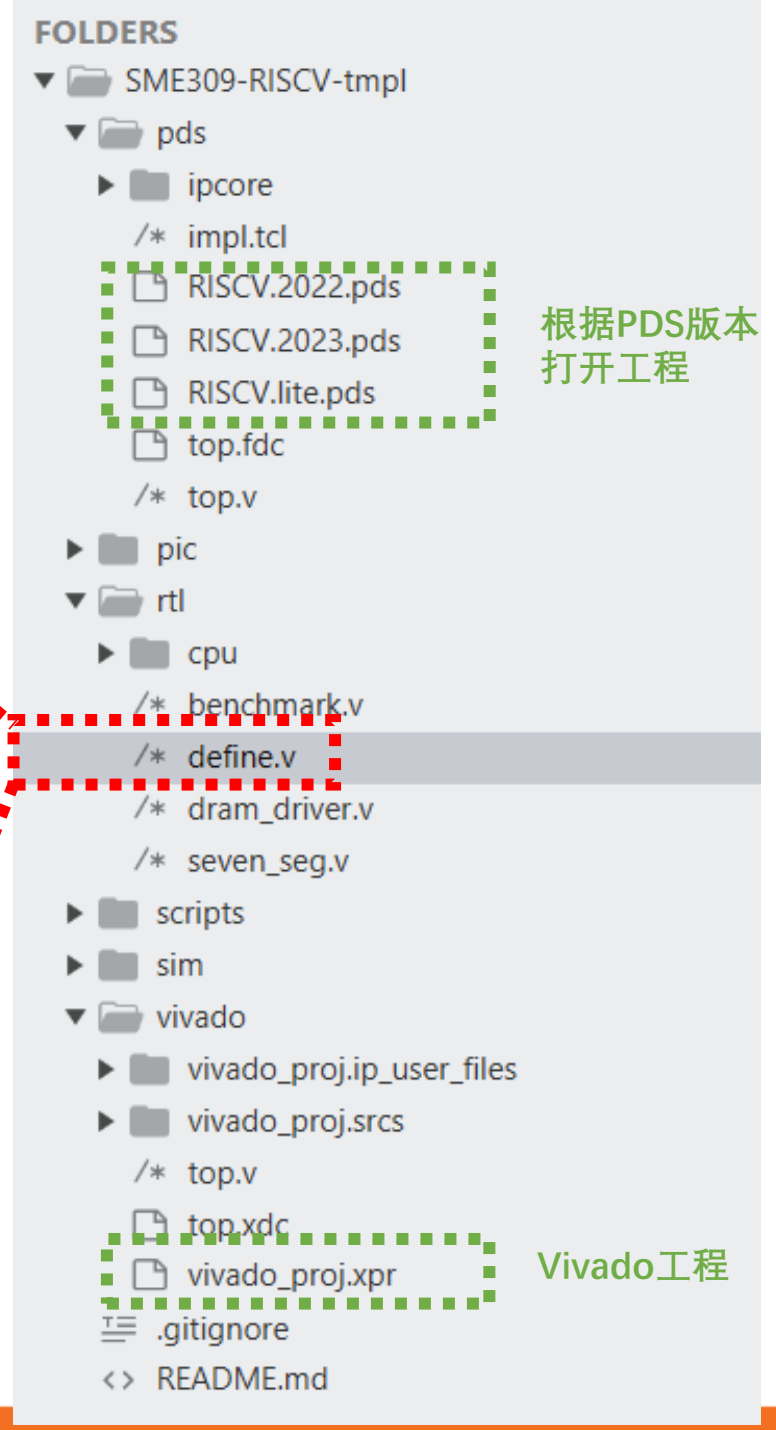
RISC-V /代码模板说明

Vivado & PDS 差异：在代码模板 rtl / define.v 文件做配置

- 针对 vivado，删除下面第2、3行
- 针对 PDS 正式版，删除第3行（LITE版不需要更改）

define.v

```
1   // remove PANGO definition, if you use vivado
2   `define PANGO
3   `define PDS_LITE
4
5   // CPU Configurations
6   // Here we define 0x8000_0000 ~ 0x8FFF_FFFF as the data memory space
7   `define MEM_ADDR_START 32'h8000_0000
8   `define MEM_ADDR_END   32'h8FFF_FFFF
9   `define XLEN 32
10
11  `define PERIP_RES0_ADDR 32'h90000000
12  `define PERIP_RES1_ADDR 32'h90000004
13  `define PERIP_RES2_ADDR 32'h90000008
14  `define PERIP_RES3_ADDR 32'h9000000c
15  `define PERIP_RES4_ADDR 32'h90000010
```



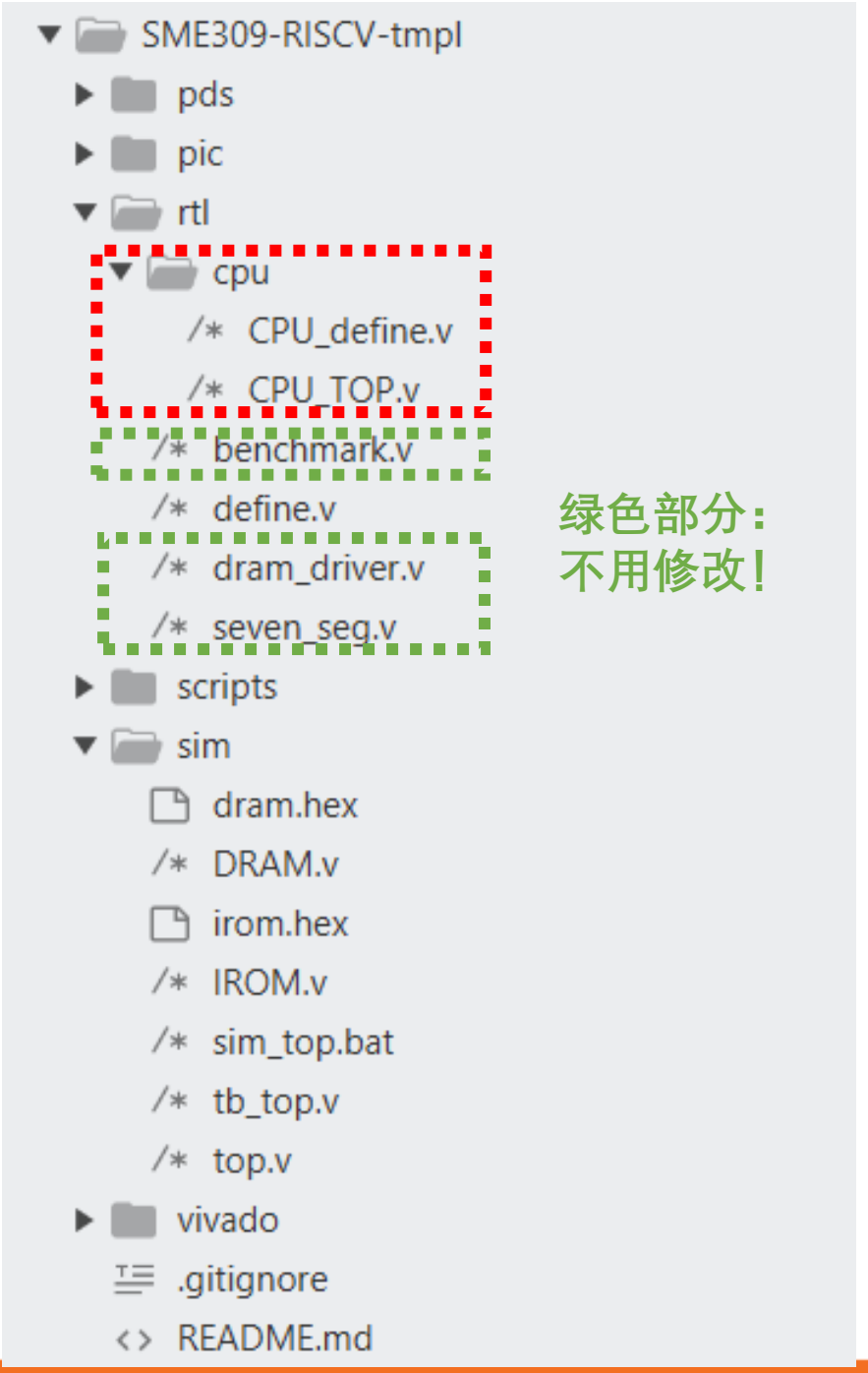
RISC-V / 代码模板说明

待开发 CPU 核：在 rtl/cpu 子目录下，完成 CPU_TOP.v 的实现

- CPU 相关子模块，需要加到 rtl/cpu 子目录中
- 注意：不能修改 CPU_TOP 模块的名称、输入输出端口，和右图绿色部分代码

CPU_TOP.v

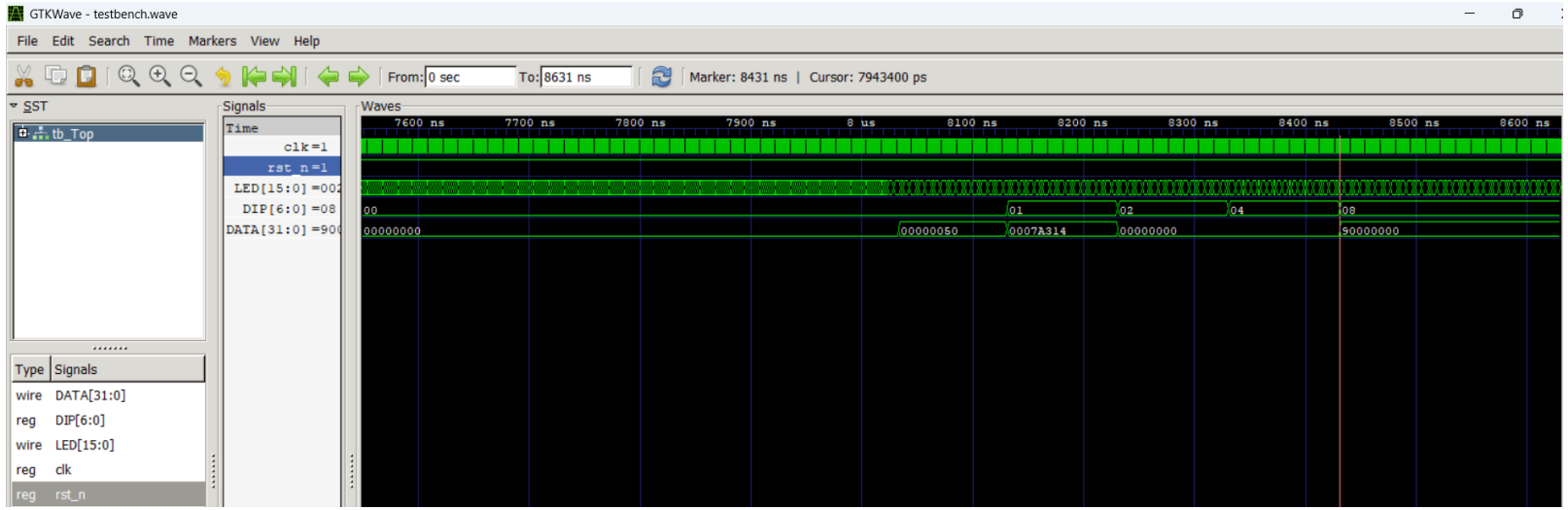
```
1  `include "CPU_define.v"
2  module CPU (
3      input          clk,
4      input          rst_n,
5      input          run,
6
7      // Interface to IROM
8      output [31:0]  pc,
9      input  [31:0]  instruction,
10
11     // Interface to DRAM & MMIO
12     output [31:0]  perip_addr,
13     output [31:0]  perip_wdata,
14     output         perip_wen,
15     output [1:0]   perip_mask,
16     input  [31:0]  perip_rdata
17 );
```



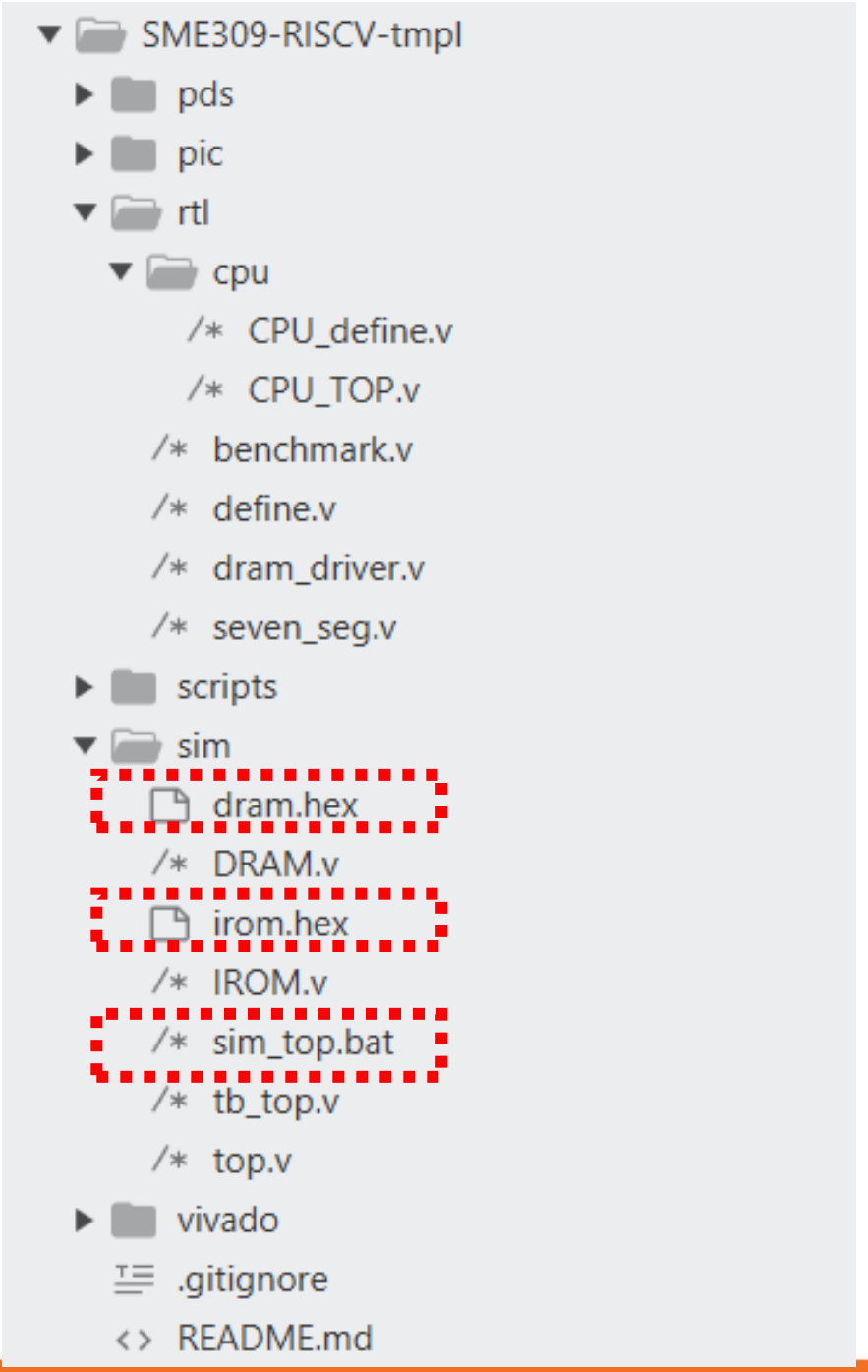
RISC-V / 代码模板说明

仿真：需先安装 iverilog 环境，在执行 sim 子目录的 sim_top.bat

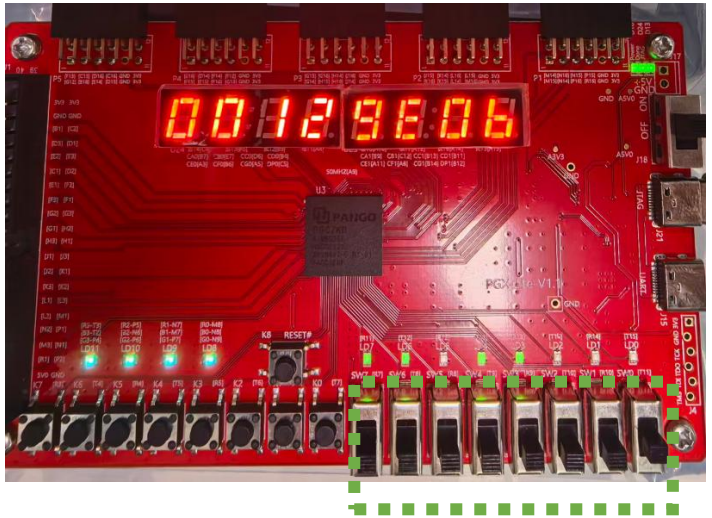
- irom.hex 为仿真时 Instr Memory（内容可替换为 add_loop.dat）
- dram.hex 为仿真时 Data Memory 初始化数据
- 注意：sim 子目录下的 Verilog 代码不用修改



Tip: Gtkwave 查看波形图会比 vivado 自带工具方便



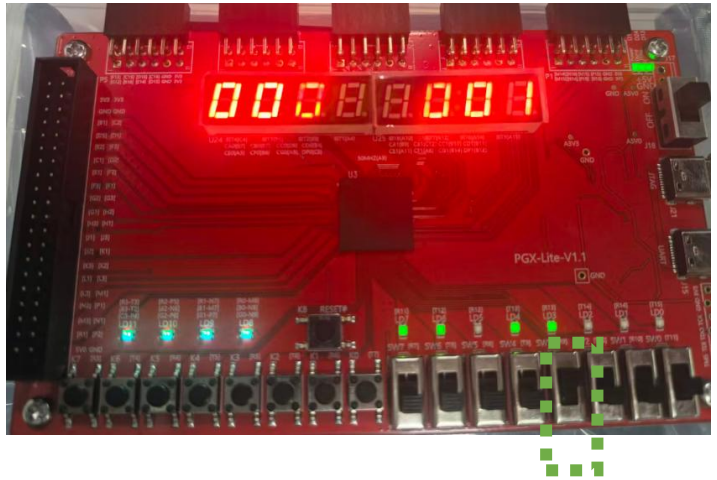
RISC-V / Demo (Xilinx 类似，以下结果需在报告中)



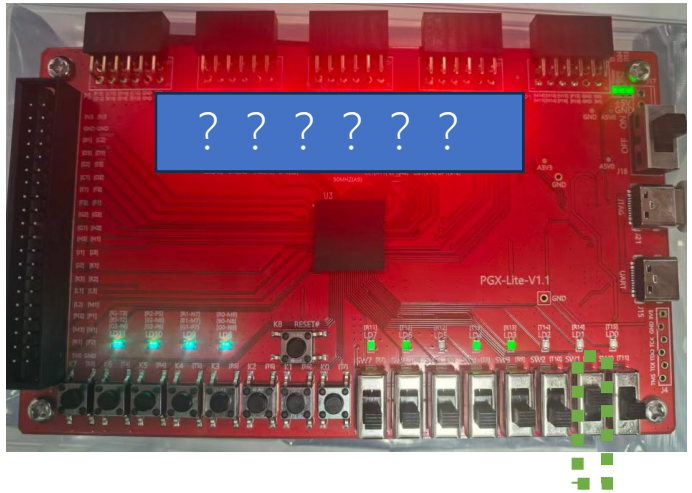
1）测评程序运行结束，SW 全拨下

LED 表示 PC，显示：1101_1000

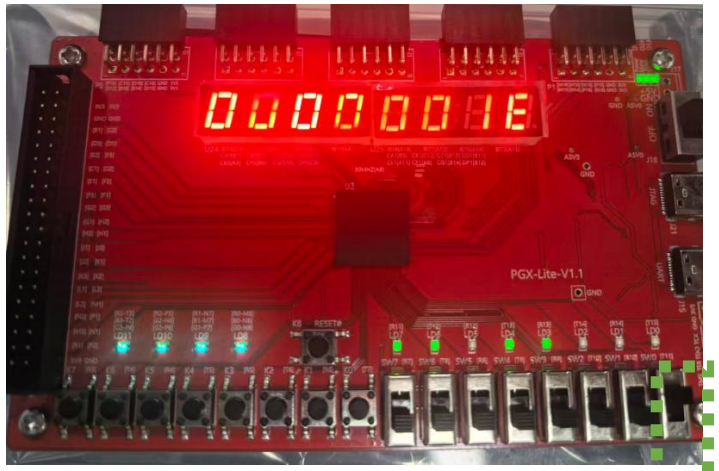
数码管显示：**CPU运算时间**



2）SW3 拨上：**测评结束返回结果**



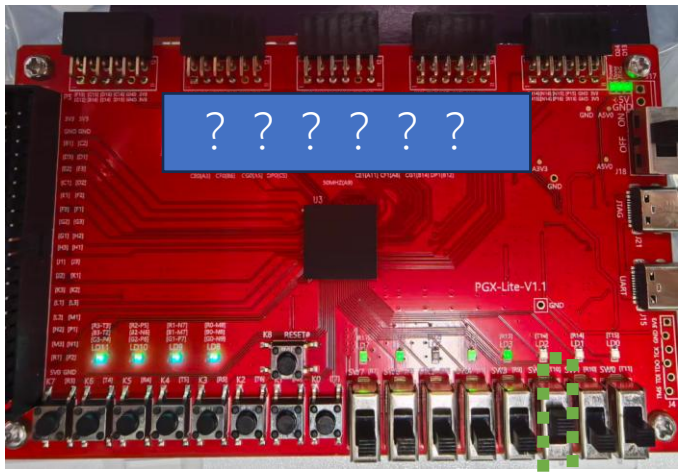
4）SW1 拨上：**中间计算结果1**（防作弊）



3）SW0 拨上：**指令测试通过多少case?**

注意：如 2）3) 跟截图结果不一致，

说明 CPU 基础功能有问题！

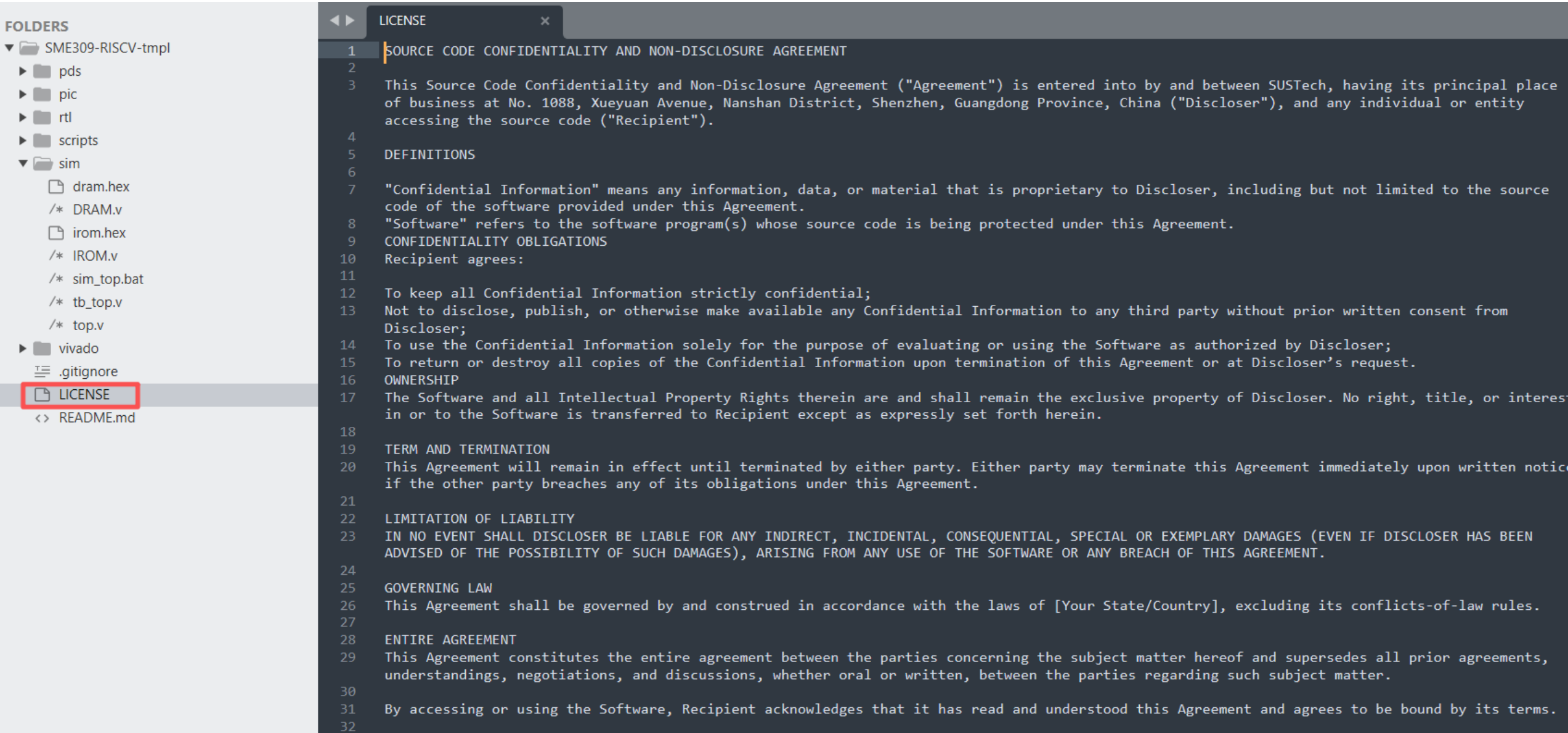


5）SW2 拨上：**中间计算结果2**（防作弊）

RISC-V / 代码模板版权

不得擅自复制、分发、公开源代码
仅限于 SME309 Project 小组内部分使用

SUSTech Southern University of Science and Technology



Project Intro / Outline

SUSTech Southern University of Science and Technology

- Git
- RISC-V
- **Hints**
- Lab Review & Summary

Hints / Pipelined CPU

SUSTech Southern University of Science and Technology

In this project, you will implement a five-stage pipeline processor based on the single-cycle processor you developed in Lab2, as demonstrated in the lecture. The architecture is shown below. Take care of data hazards (data forwarding, stall, flush) and control hazards (early BTA, flush).

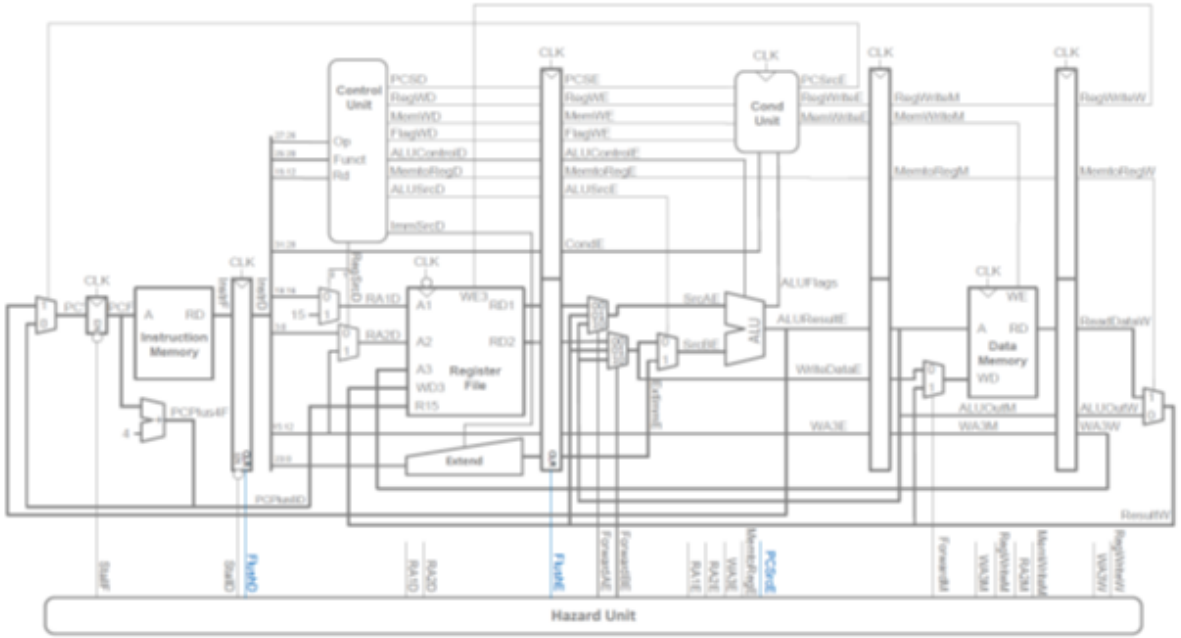
The following techniques should be implemented to handle pipeline hazards:

(1) Using Different Clock Edges

(2) Inserting NOPs

(3) Data Forwarding

(4) Early BTA

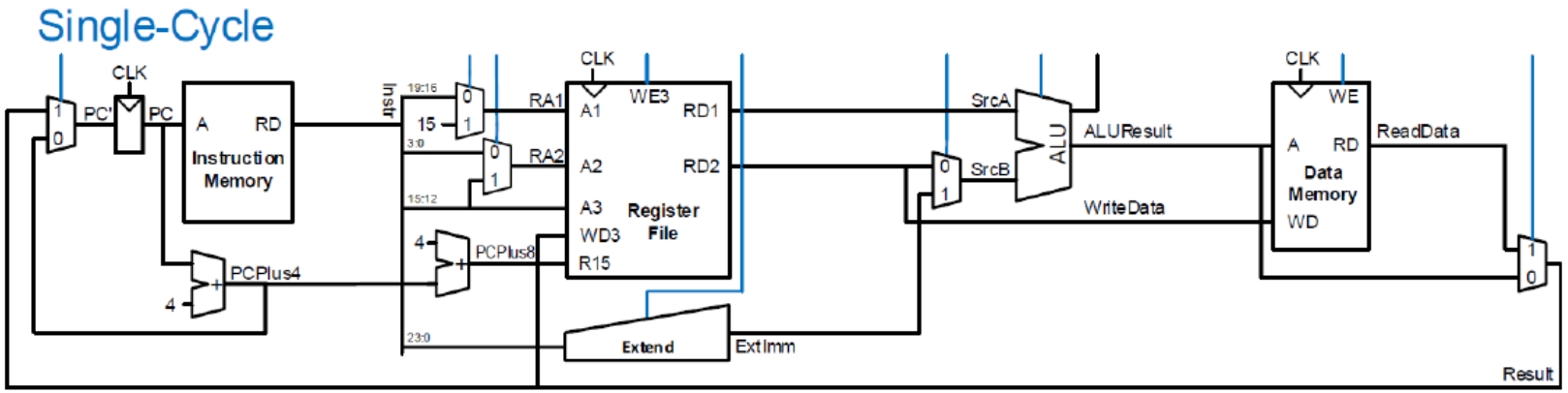


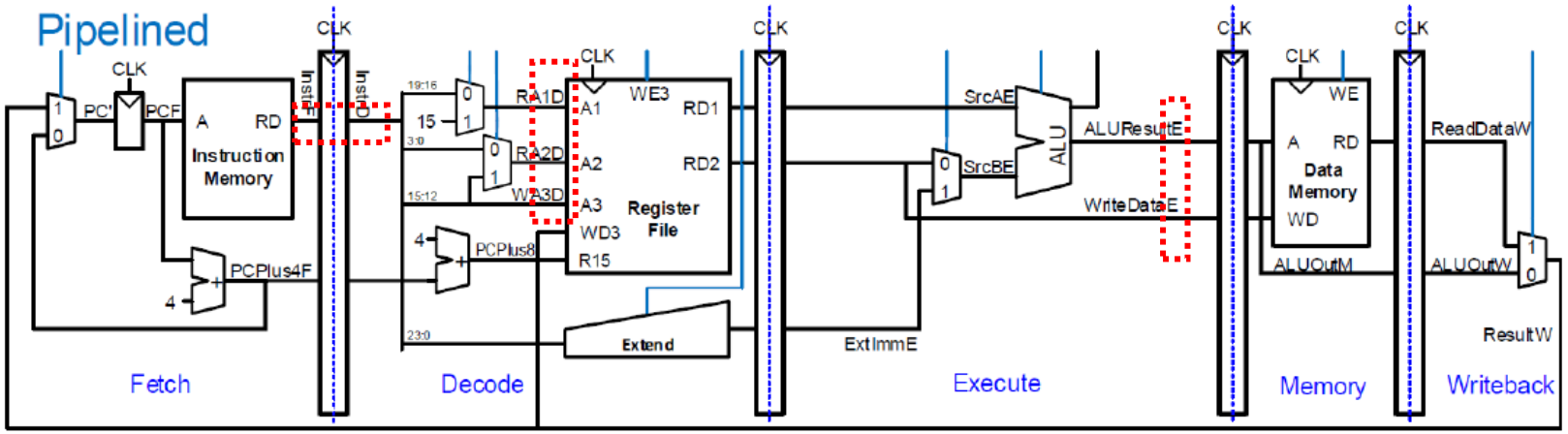
Pipelined CPU / Step1, break down 5 stages --【挖坑】

SUSTech Southern University of Science and Technology

取指 / F -> 译指 / D -> 执行 / E-> Mem读写 / M -> Reg回写 / W

- 注意变量后缀



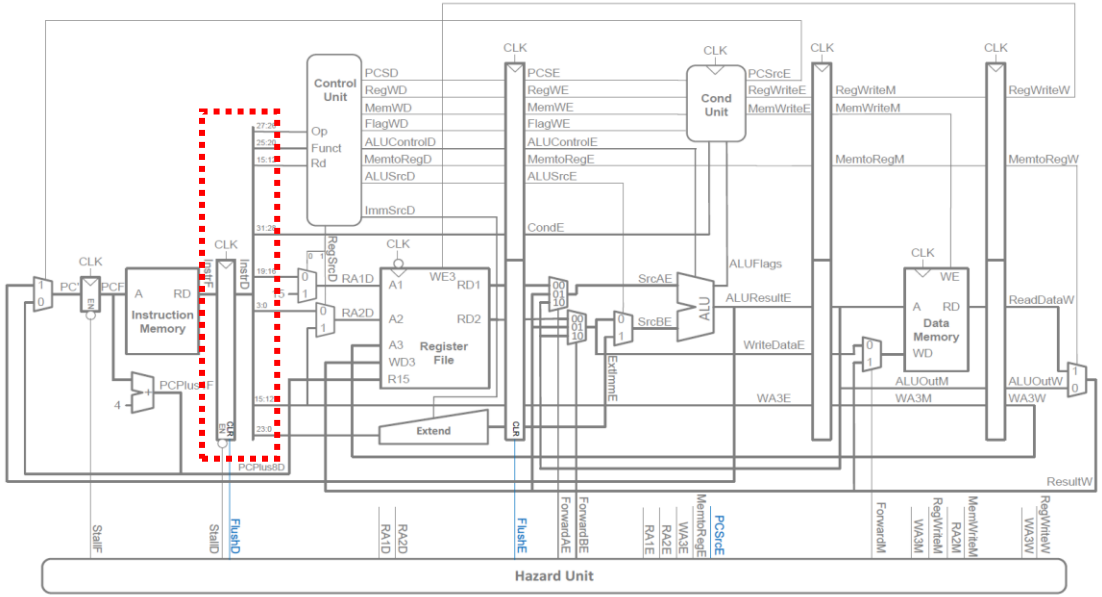


Pipelined CPU / Step1, break down 5 stages --【挖坑】

SUSTech Southern University of Science and Technology

取指 / F -> 译指 / D -> 执行 / E-> Mem读写 / M -> Reg回写 / W

- 注意变量后缀
- 变量在流水线的传递，不同阶段对应不同变量 (e.g. Inst)



git diff

```
+
+//////////////////////////////   5 stages     ///////
+
+reg [31:0] InstrD = 32'd0;
+
+always @(posedge CLK or posedge Reset) begin
+    if (Reset) begin
+        InstrD <= 32'd0;
+    end
+    else begin
+        InstrD <= Instr;
+    end
+end
```

```
 ////////////////////////////// Control Unit ///////////
 ControlUnit u_ControlUnit(
-    .Instr(Instr),
+    .Instr(InstrD),
     .ALUFlags(ALUFlags),
     .CLK(CLK),
     .MemtoReg(MemtoReg),
@@ -74,16 +88,16 @@ assign PC_Plus_8 = PC + 4'b1000;
```

```
 ////////////////////////////// Register ///////////////////////////
-assign RA1 = (RegSrc[0] == 1'b0) ? Instr[19:16] : 4'b1111;
+assign RA1 = (RegSrc[0] == 1'b0) ? InstrD[19:16] : 4'b1111;

-assign RA2 = (RegSrc[1] == 1'b0) ? Instr[3:0] : Instr[15:12];
+assign RA2 = (RegSrc[1] == 1'b0) ? InstrD[3:0] : InstrD[15:12];

 RegisterFile u_RegisterFile(
     .CLK(CLK),
     .WE3(RegWrite),
     .A1(RA1),
     .A2(RA2),
-    .A3(Instr[15:12]),
+    .A3(InstrD[15:12]),
     .WD3(Result),
     .R15(PC_Plus_8),
     .RD1(Src_A),
```

Pipelined CPU / Step1, break down 5 stages --【填坑】

SUSTech Southern University of Science and Technology

取指 / F -> 译指 / D -> 执行 / E-> Mem读写 / M -> Reg回写 / W

- 注意变量后缀
- 变量在流水线的传递，不同阶段对应不同变量 (e.g. Inst)
- 思考：shifter 放到哪个 stage ?
- 复杂细节先往后放，如：ControlUnit

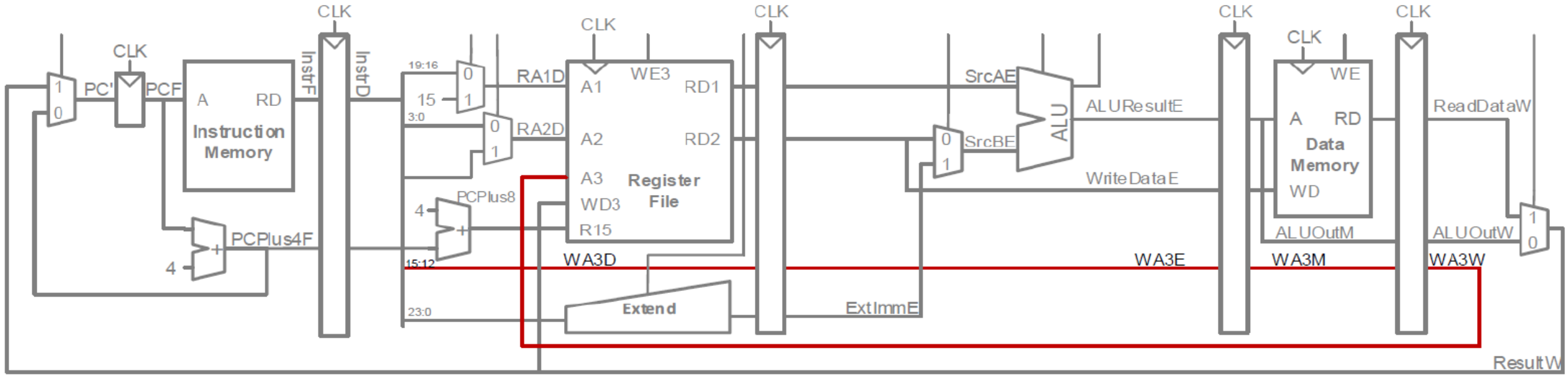
Pipelined CPU / Step1, break down 5 stages --【填坑】

SUSTech Southern University of Science and Technology

细节处理

- 针对往回走的信号线失效，需要放到 pipeline 中进行传递

Write Back: A3, WD3



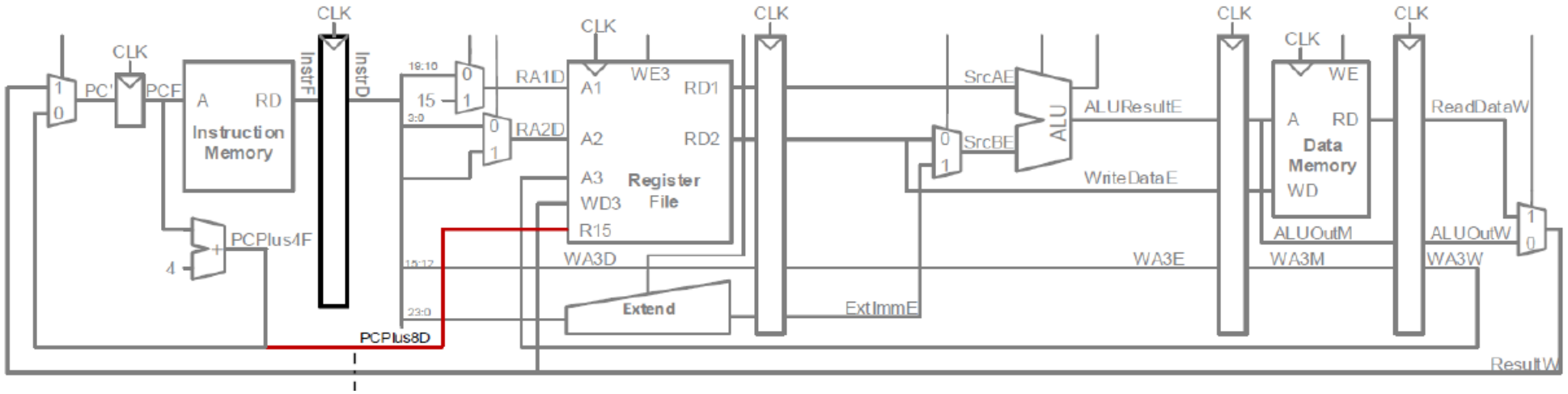
Pipelined CPU / Step1, break down 5 stages --【填坑】

SUSTech Southern University of Science and Technology

细节处理

- 针对往回走的信号线失效，需要放到 pipeline 中进行传递
- 针对往前跳的信号线失效，...

译码: PC + 8 使用 PC + 4 替代



Pipelined CPU / Step1, break down 5 stages --【填坑】

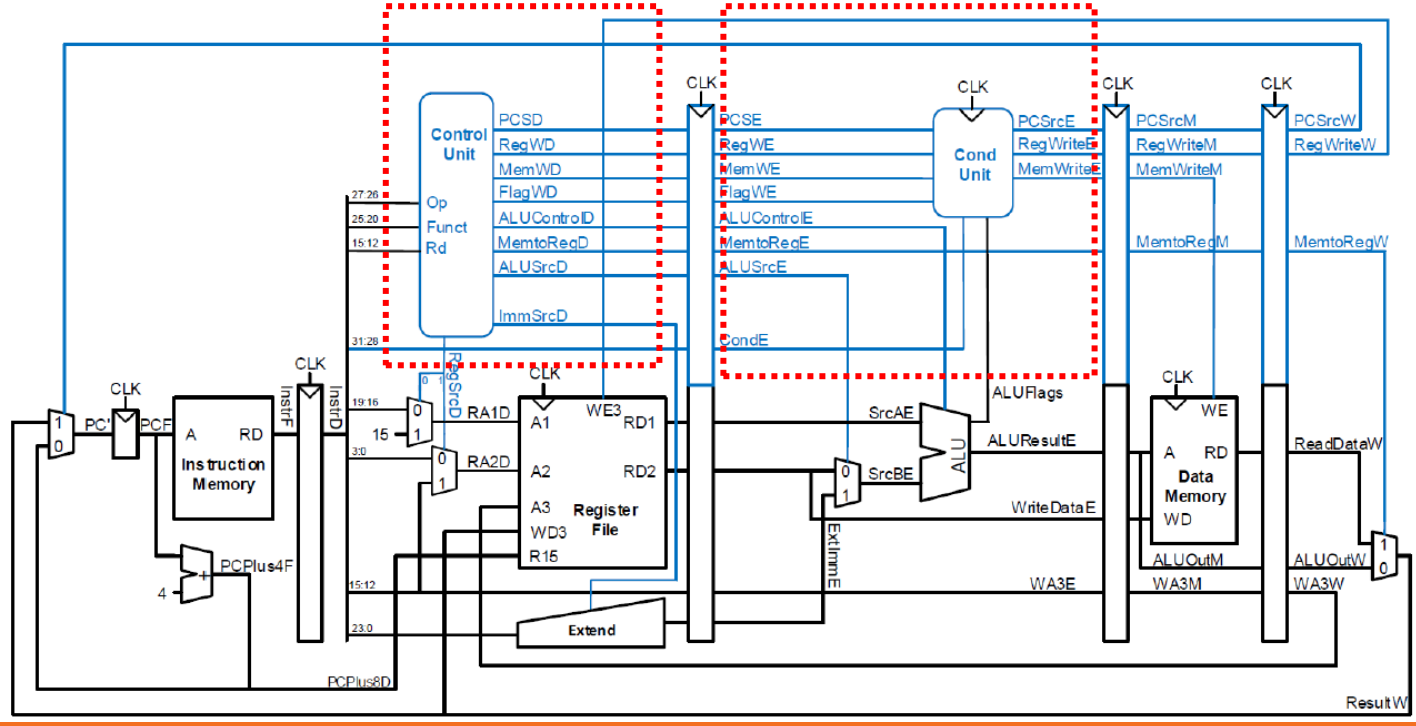
SUSTech Southern University of Science and Technology

细节处理

- 针对往回走的信号线失效，需要放到 pipeline 中进行传递
- 针对往前跳的信号线失效，...
- **ControlUnit 分拆出：Decoder(即下图ControlUnit) 和 CondUnit**

注意：

- NoWrite 信号
- 哪些信号往前走、往回走？

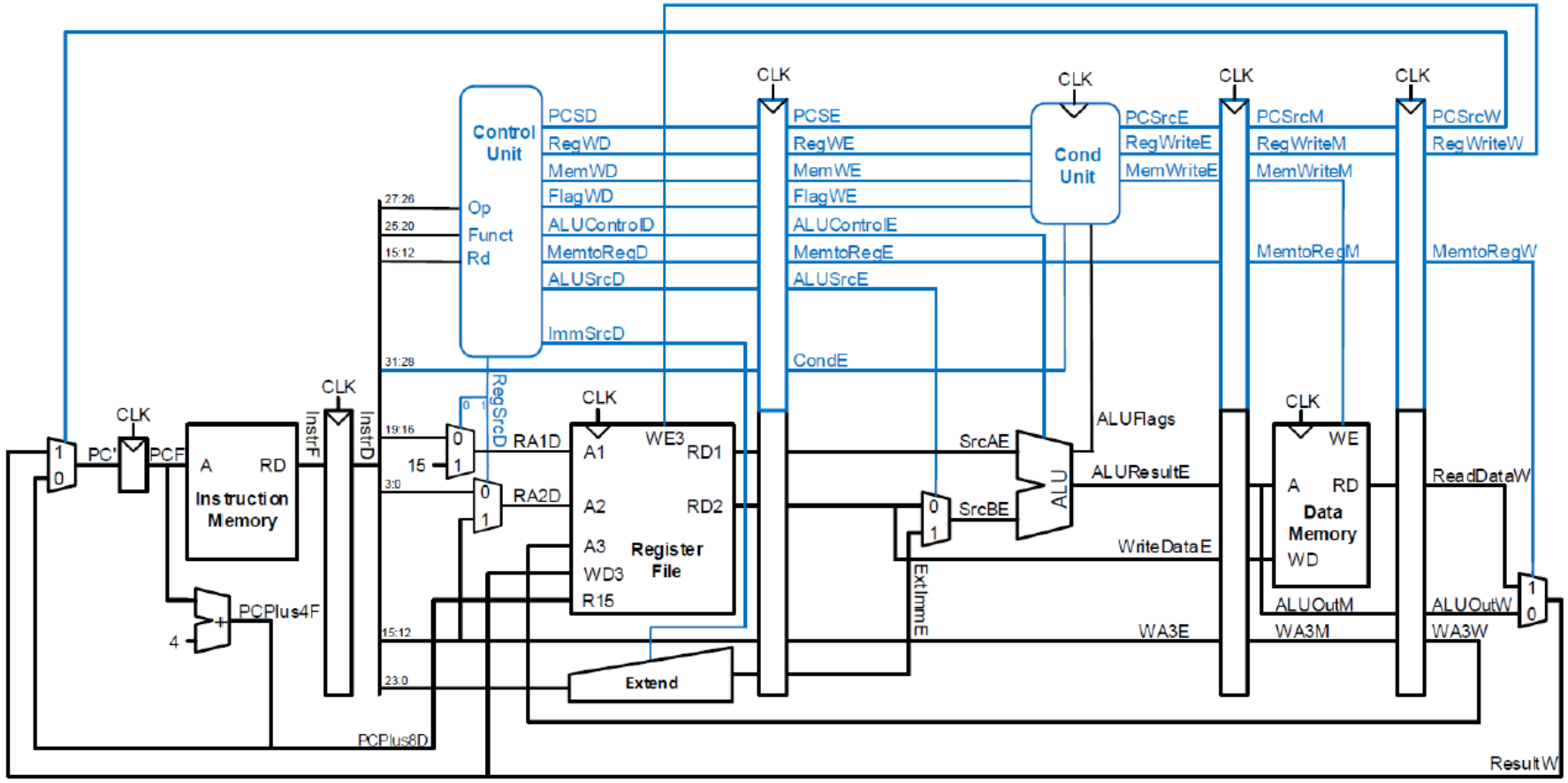


Pipelined CPU / Step1, break down 5 stages --【填坑】

SUSTech Southern University of Science and Technology

停下来，思考一下

- 如何测试？-- 坑填完了吗):
- Lab2 的测试代码，能跑起来吗？
- 如果不能，为什么？



Pipelined CPU / Step2, 引入 HazardUnit

SUSTech Southern University of Science and Technology

Hazard 分类

- Structure: 硬件资源的争夺
- Data: 数据依赖
- Control：控制流处理

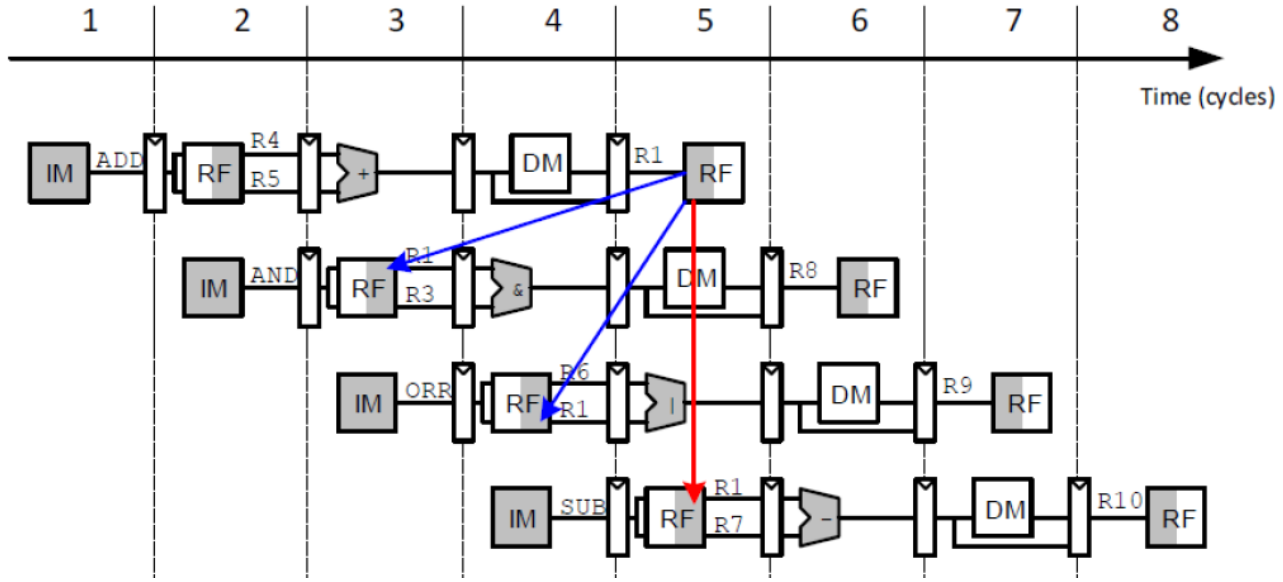
针对 Project 中 CPU, 需要解决哪些?

Pipelined CPU / Step2, 引入 HazardUnit

SUSTech Southern University of Science and Technology

Data Hazard

- 从 WB 看，细化情况
  - WB vs Dec
  - WB vs Exec
  - WB vs Mem



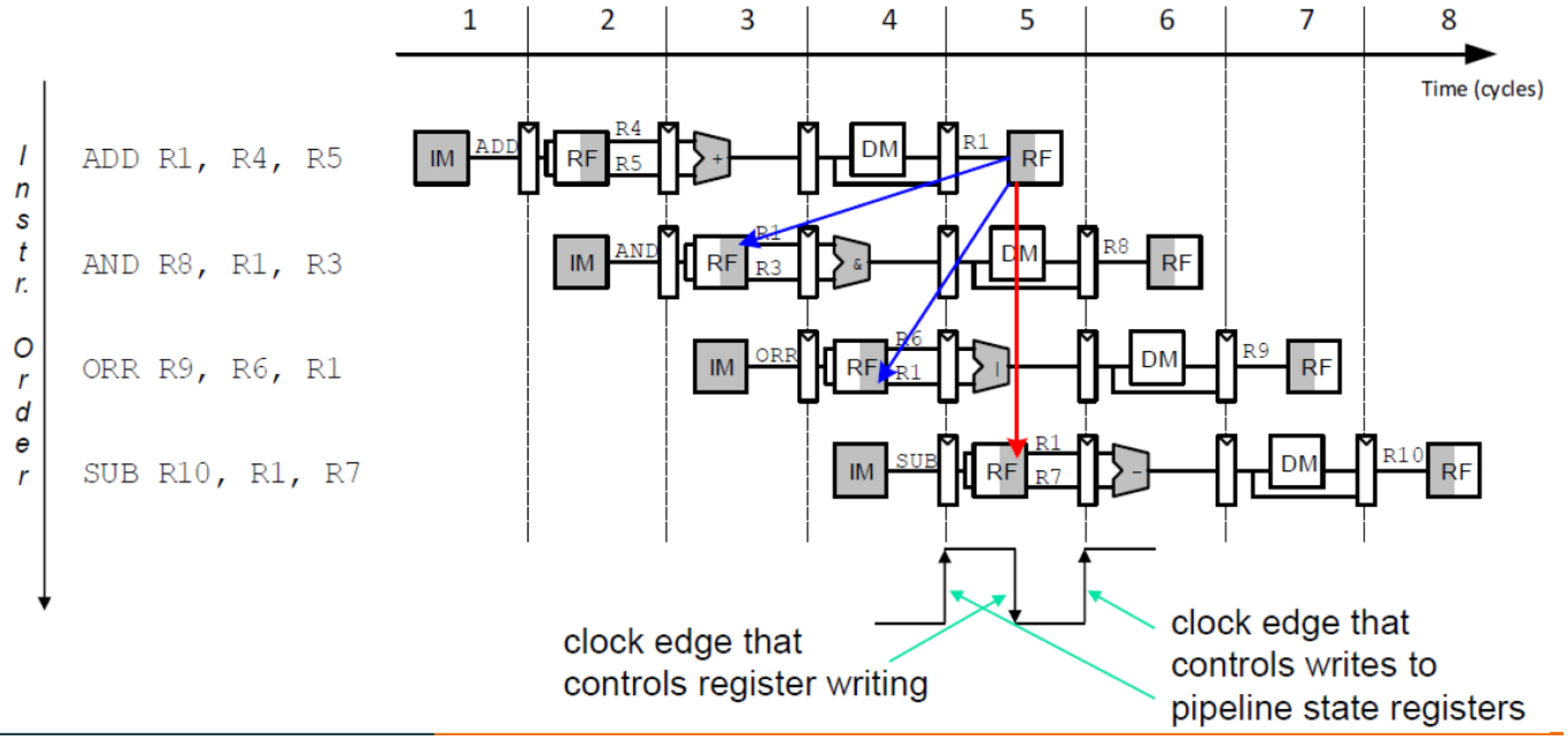
Pipelined CPU / Step2, 引入 HazardUnit

Data Hazard

- 从 WB 看，细化情况
- WB vs Dec：调整写入的时钟触发时机

问题：需要读取的数据，正要写入

Timing Matter!



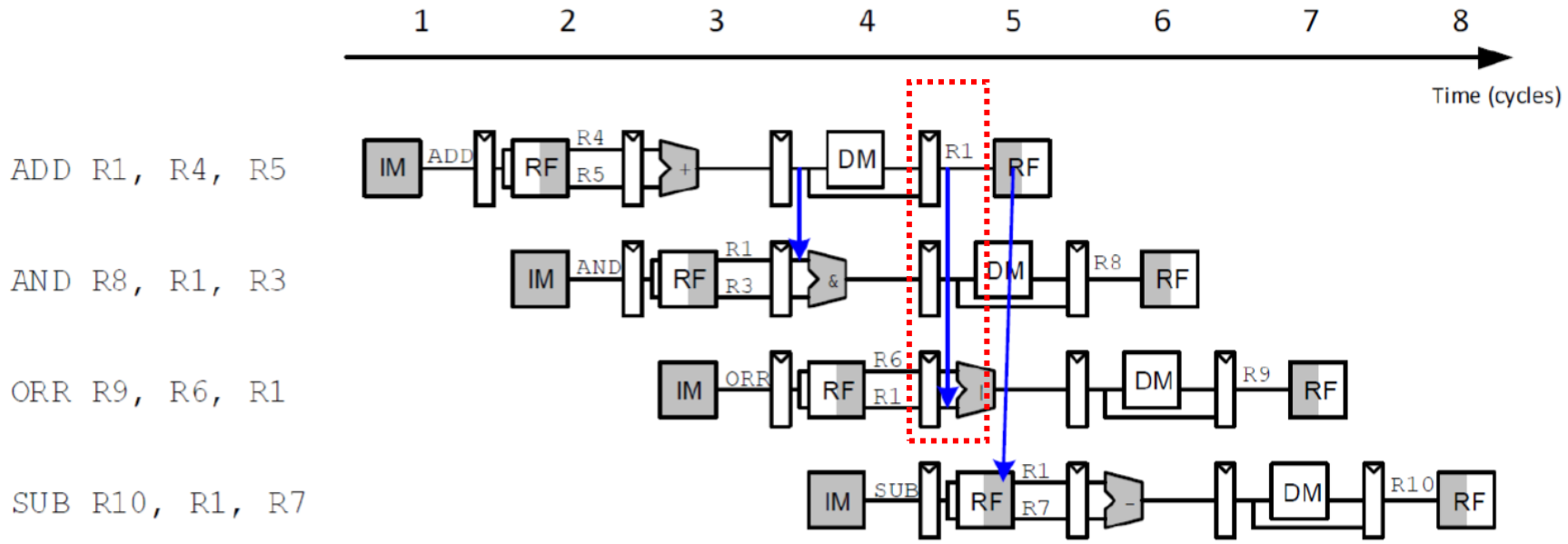
Pipelined CPU / Step2, 引入 HazardUnit

Data Hazard

- 从 WB 看，细化情况
- WB vs Dec：调整写入的时钟触发时机
- WB vs Exec： Data Forwarding

问题：需要计算的数据，正要写入

找到一个切入点，逐步迭代: HazardUnit



Pipelined CPU / Step2, 引入 HazardUnit

SUSTech Southern University of Science and Technology

Data Hazard

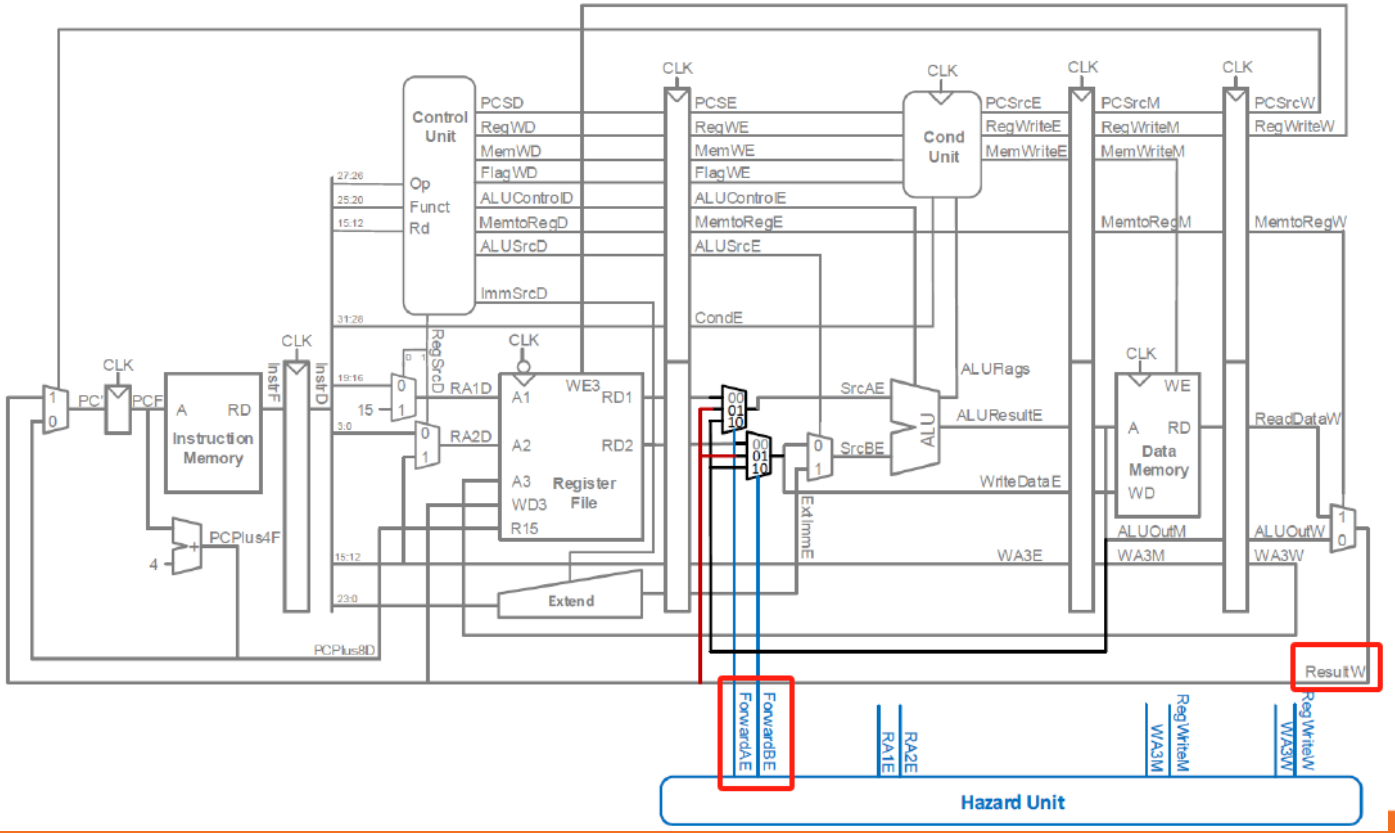
- 从 WB 看，细化情况
- WB vs Dec：调整写入的时钟触发时机
- WB vs Exec： Data Forwarding

第1个版本：HazardUnit

针对 ALU 的两个操作数

- ResultW to RD1E / RD2E,
  - when WA3W == RA1E / RA2E, and RegWriteW
- ALUOutM to RD1E / RD2E
  - when WA3M == RA1E / RA2E, and RegWriteM

思考：以上条件有问题吗?

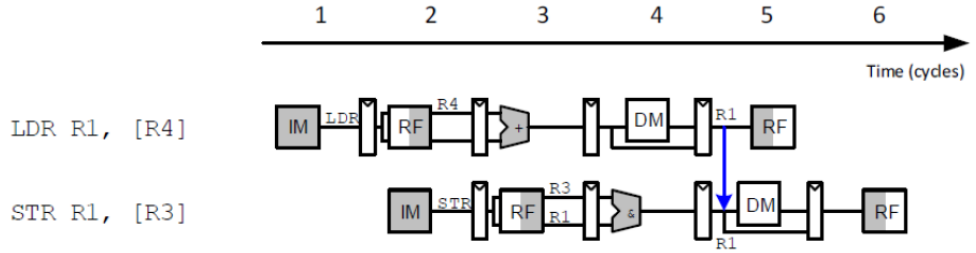


Pipelined CPU / Step2, 引入 HazardUnit

SUSTech Southern University of Science and Technology

Data Hazard

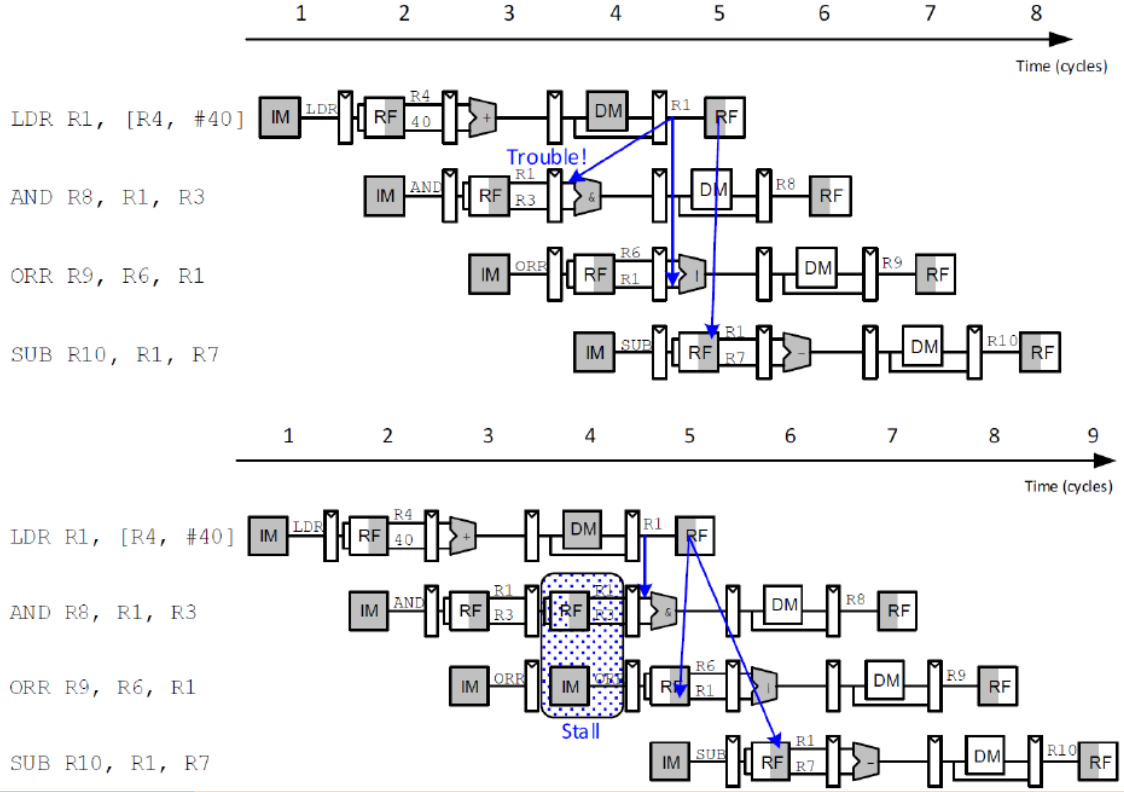
- 从 WB 看，细化情况
- WB vs Dec：调整写入的时钟触发时机
- WB vs Exec： Data Forwarding
- **WB vs Mem： Data Fowarding / Stalling**



- Check if the register used in Memory stage (i.e., *Rd* of `STR`) by the `STR` instruction matches register written by `LDR` in Writeback stage (i.e., *Rd* of `LDR`)

进一步细化问题

- 需要写入 Memory 的数据，正要回写 Reg ---> Mem-Mem copy
- 需要读/写 Memory 的地址，正要回写 Reg
  - 回写 Reg 的数据，经过 Exec 已经 Ready ---> Data forwarding
  - 回写 Reg 的数据，需要 Mem 进行读取 ---> Stall !!



Pipelined CPU / Step2, 引入 HazardUnit

SUSTech Southern University of Science and Technology

Data Hazard

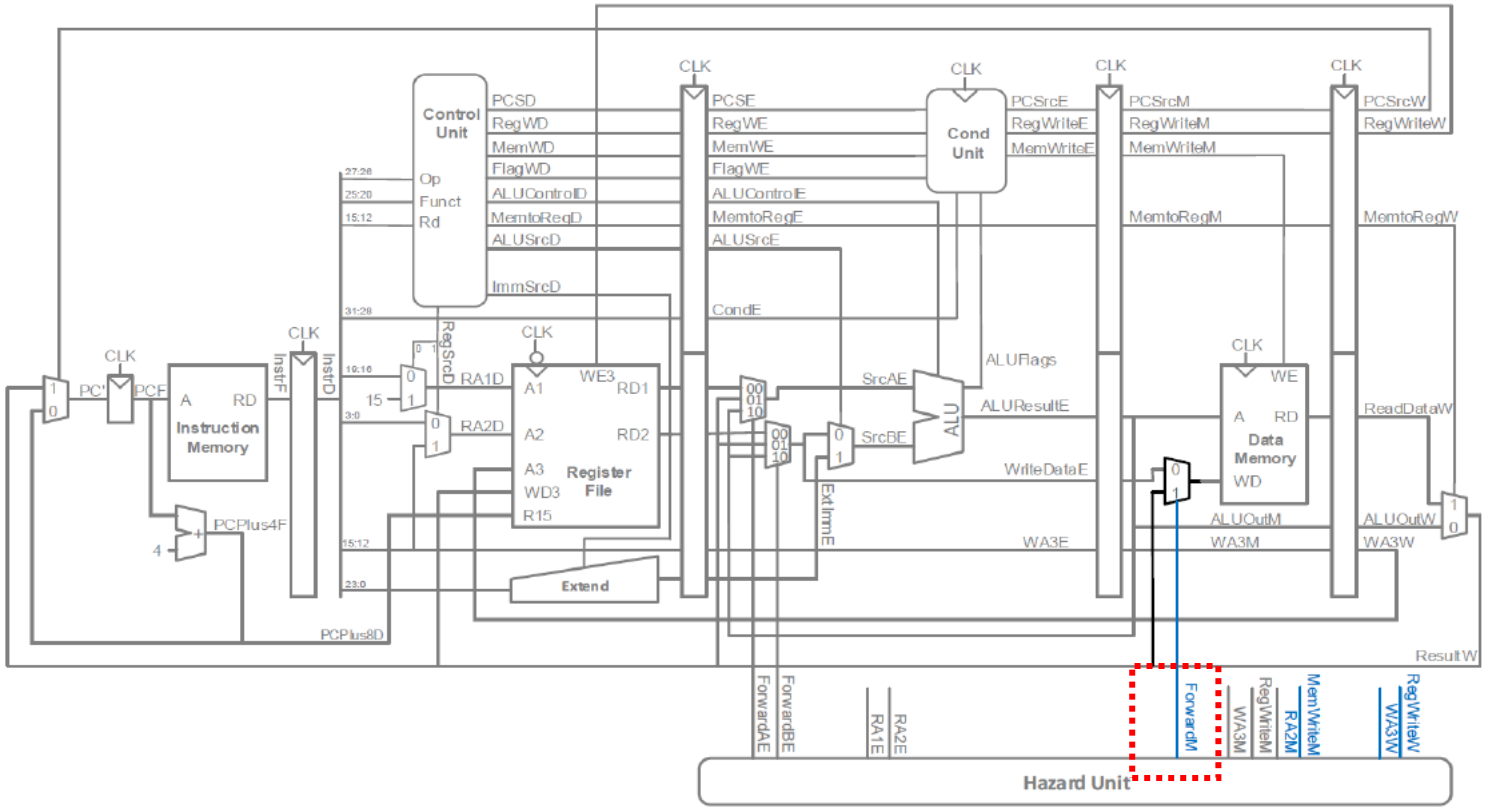
- 从 WB 看，细化情况
- WB vs Dec：调整写入的时钟触发时机
- WB vs Exec： Data Forwarding
- **WB vs Mem： Data Fowarding / Stalling**

第2个版本：HazardUnit

=> 支持 Mem-Mem Copy

=> 优化 Data Forwarding, 增加信号： ForwardM

=> 思考： 需要满足什么条件?



Pipelined CPU / Step2, 引入 HazardUnit

SUSTech Southern University of Science and Technology

Data Hazard

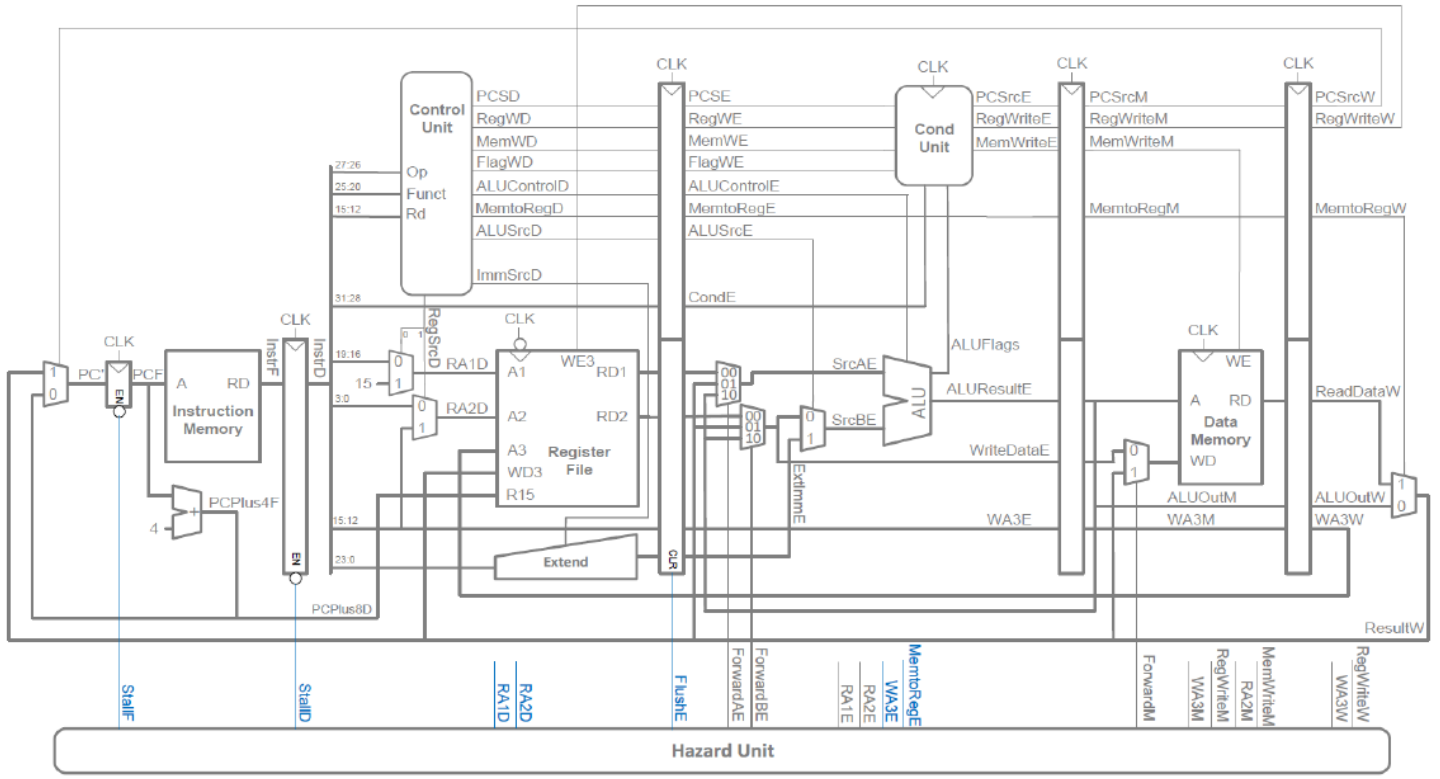
- 从 WB 看，细化情况
- WB vs Dec：调整写入的时钟触发时机
- WB vs Exec：Data Forwarding
- **WB vs Mem：Data Fowarding / Stalling**

第3个版本：HazardUnit

=> 引入 Fetch, Decode Stall, Exec Flush

=> 关键：Fetch，Decode 延迟1个周期，而且Decode 的指令不往后传递

=> 思考：决策时机?



Pipelined CPU / Step2, 引入 HazardUnit

SUSTech Southern University of Science and Technology

Data Hazard

- 从 WB 看，细化情况
- WB vs Dec：调整写入的时钟触发时机
- WB vs Exec：Data Forwarding
- **WB vs Mem：Data Fowarding / Stalling**

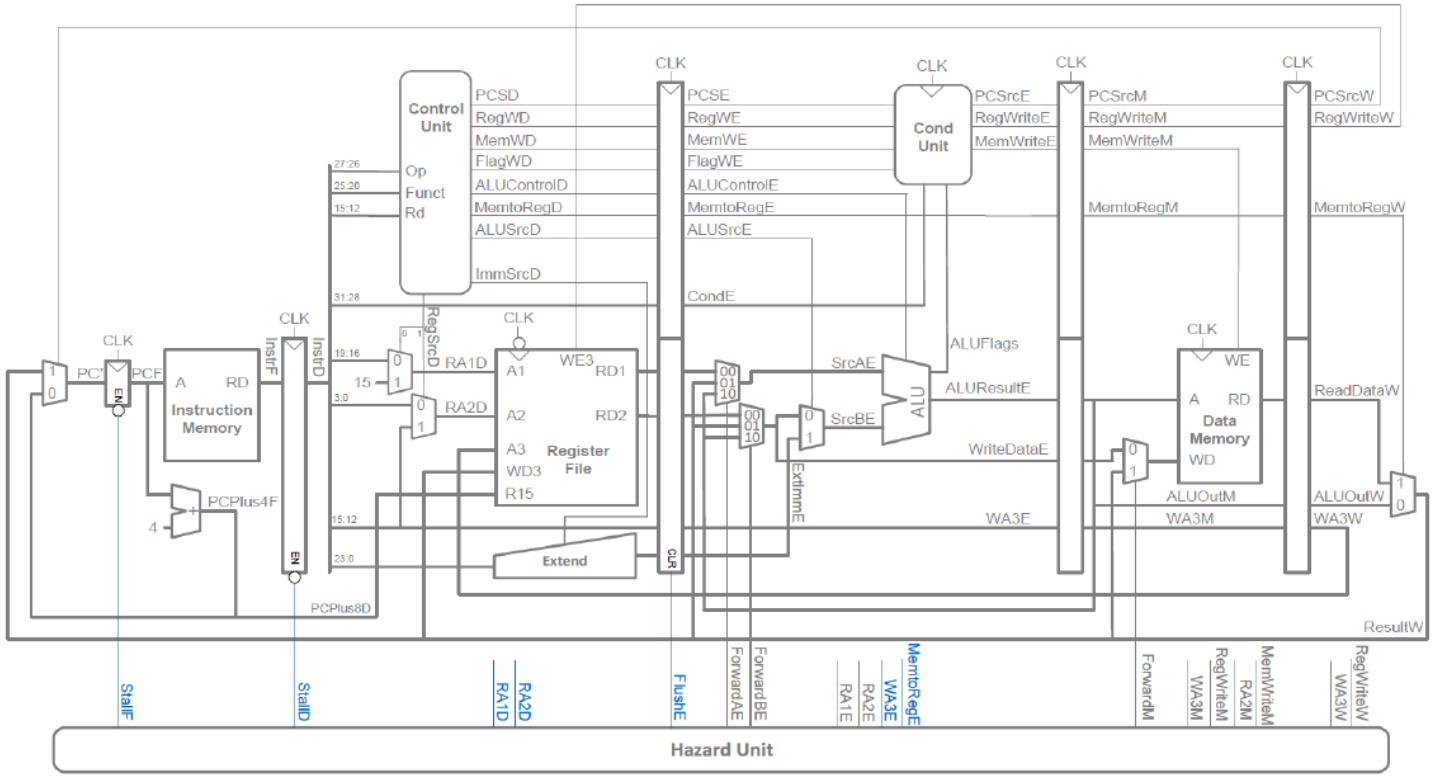
第3个版本：HazardUnit

=> 引入 Fetch, Decode Stall, Exec Flush

=> 关键：Fetch，Decode 延迟1个周期，而且Decode 的指令不往后传递

=> 思考：决策时机？--- Exec 执行的时候，决定后面的 Fetch、Decode 是否延迟1个周期

=> 针对目前情况(后面可补充)判断条件 ??



Pipelined CPU / Step2, 引入 HazardUnit

SUSTech Southern University of Science and Technology

Data Hazard

- 从 WB 看，细化情况
- WB vs Dec：调整写入的时钟触发时机
- WB vs Exec： Data Forwarding
- **WB vs Mem： Data Fowarding / Stalling**

第3个版本：HazardUnit

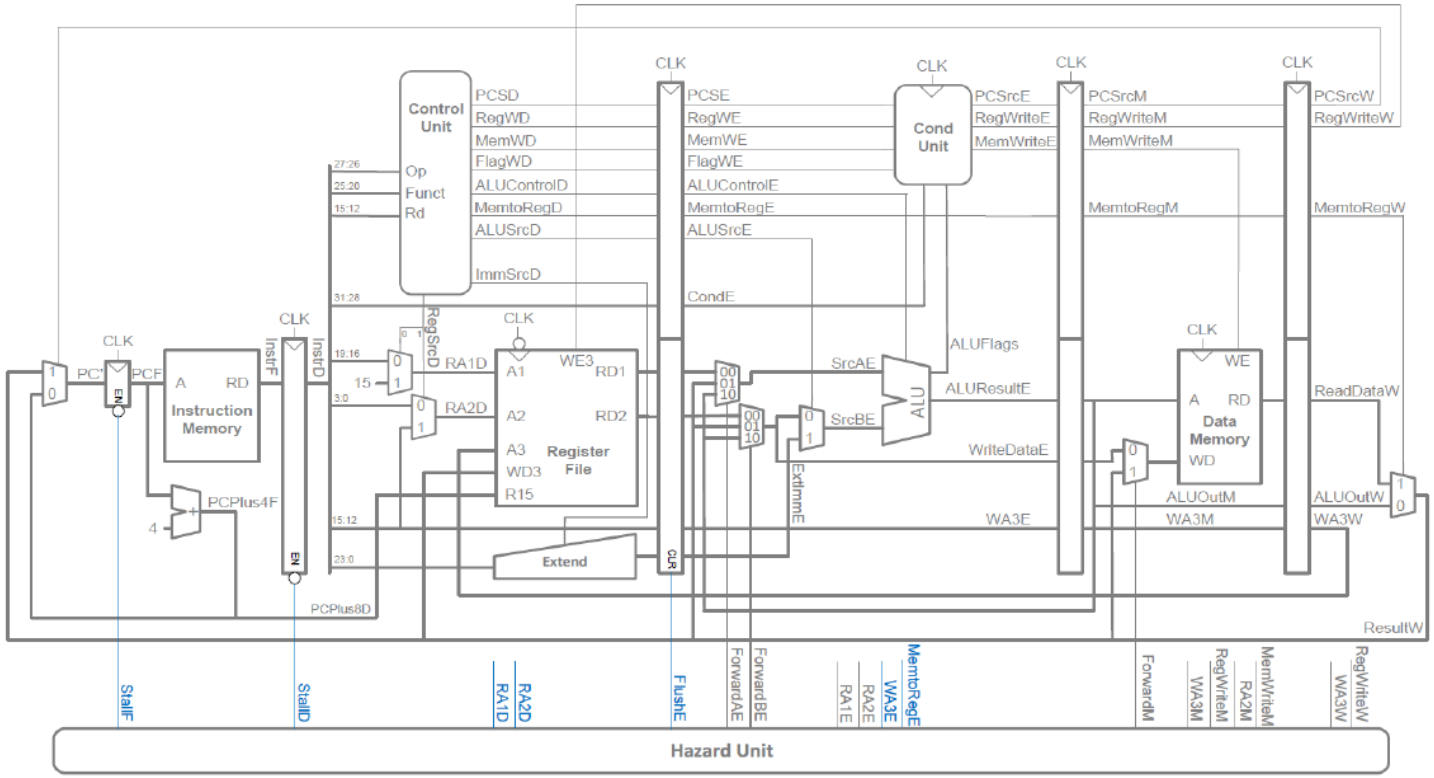
=> 引入 Fetch, Decode Stall, Exec Flush

=> 关键： Fetch，Decode 延迟1个周期，而且Decode 的指令不往后传递

=> 思考：决策时机？--- Exec 执行的时候，决定后面的 Fetch、Decode 是否延迟1个周期

=> 针对目前情况(后面可补充)判断条件： WA3E == RA1D / RA2D, 而且 MemToRegE && RegWriteE

=> 输出： StallF, StallD, FlushE



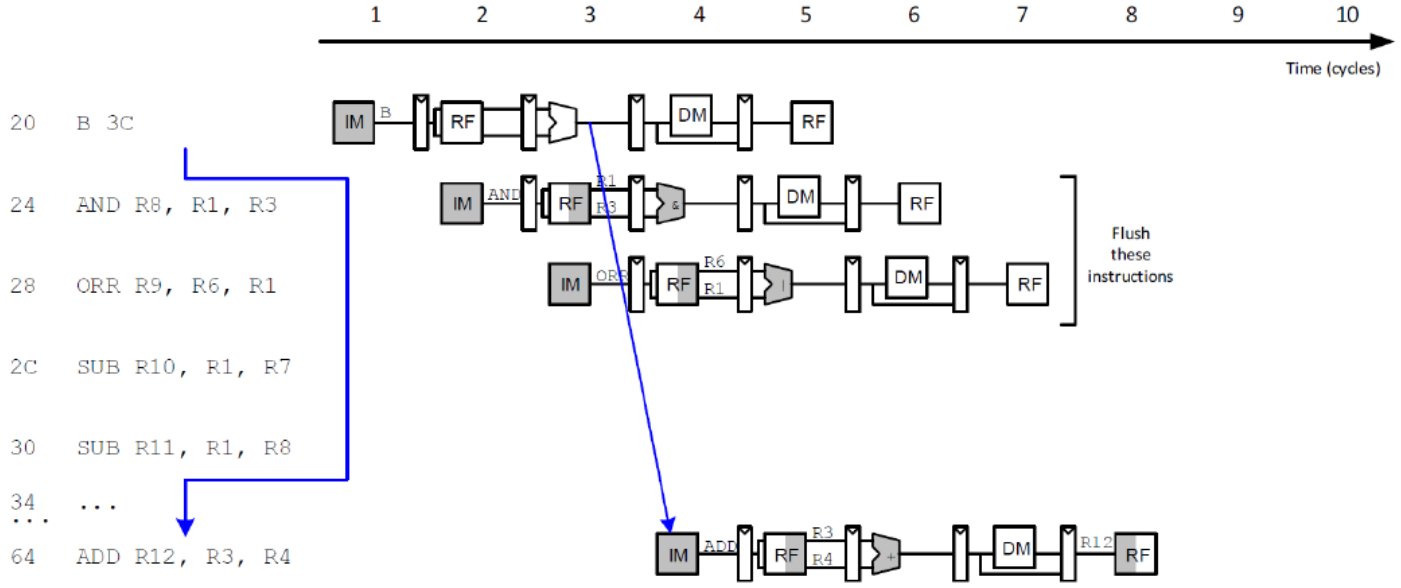
Pipelined CPU / Step2, 引入 HazardUnit

SUSTech Southern University of Science and Technology

Control Hazard

- 遇到 Branch指令 怎么办?

- Early BTA
  - Determine BTA in Execute stage
  - For now, ignoring the case of `LDR PC, ...` (`LDR` with `PC` as destination)
  - Branch misprediction penalty = 2 cycles
  - Could increase the critical path delay ☹



Pipelined CPU / Step2, 引入 HazardUnit

SUSTech Southern University of Science and Technology

Control Hazard

- 遇到 Branch指令 怎么办?

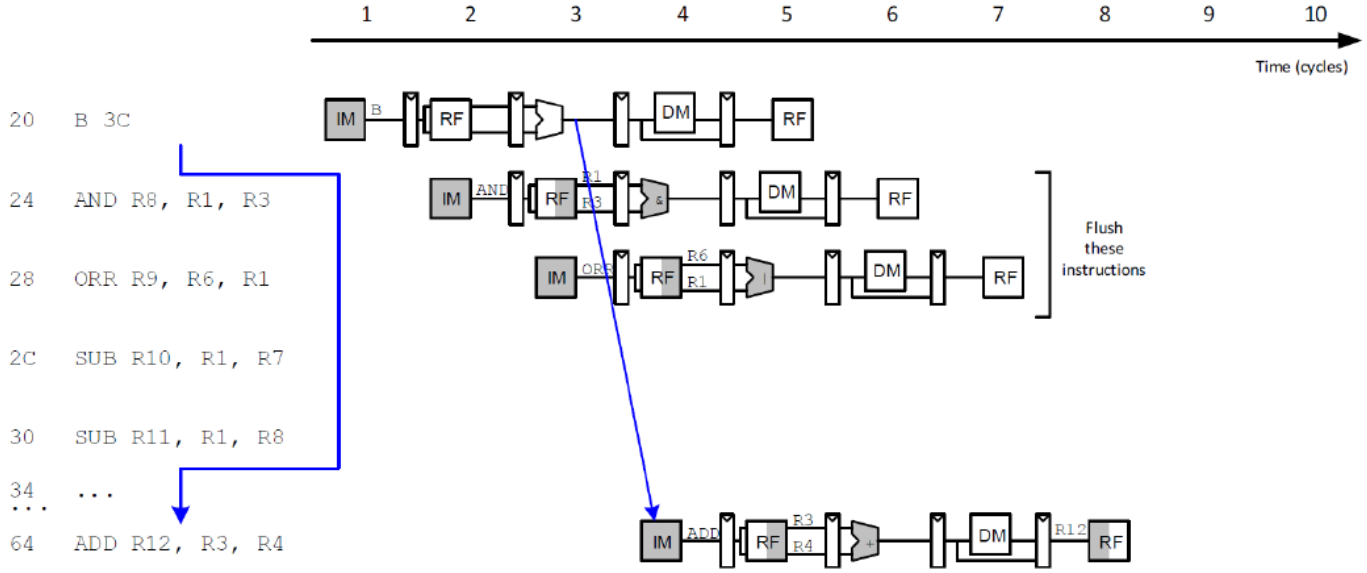
第4个版本：HazardUnit

=> 引入 Early BTA

=> ALU 计算出来的 BTA，bypassing 到 next PC

=> flush 流水线：Decode, Fetch 的指令清空

- 复位关键寄存器，区分阶段去思考
  - Exec: ...
  - Decode: ...



Pipelined CPU / 工作量预估

SUSTech Southern University of Science and Technology

```
jiaw8@LAPTOP-SAR5DOAF MINGW64 /d/work/sme309_fproj (master)
$ git commit -m "dev 5 stages pipelined"
[master 767614d] dev 5 stages pipelined
 6 files changed, 478 insertions(+), 33 deletions(-)
 create mode 100644 vivado_proj/sme309_armv3/sme309_armv3.srcs/sources_1/imports/Design_Source/HazardUnit.v
```

核心修改在：ARM.v

Hints / Non-stalling for Mcycle

SUSTech Southern University of Science and Technology

In this task, you will implement a non-stalling CPU for multi-cycle instructions (e.g., MUL instruction). When a multi-cycle instruction is being executed, the CPU should continue executing subsequent instructions, provided there is no data dependency between the previous instruction and the current one. Specifically:

When a multi-cycle instruction (e.g., MUL instruction) is executed, the CPU should execute the next instruction (instead of stalling the pipeline) if there is no data dependency between the previous instruction. For example, instruction 1 is

MUL R5, R6, R7

And the next instruction (instruction 2) is

ADD R1, R2, R3

There is no data dependency between instruction 2 and instruction 1. When CPU is executing instruction 1, it can execute instruction 2 at the same time. Because Mcycle is an independent module, when it is busy, other parts of CPU can handle other instructions at the same time.

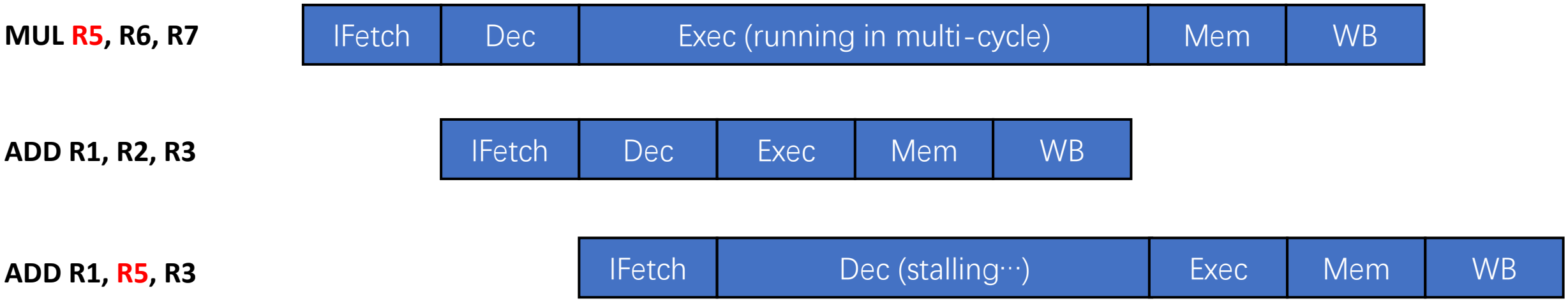
However, if instruction 2 is

ADD R1, R5, R3

The instruction 2 has a data dependency on instruction 1. In this case, the pipeline should stall until the result of the instruction 1 is available.

Non-stalling for Mcycle / 基本思路

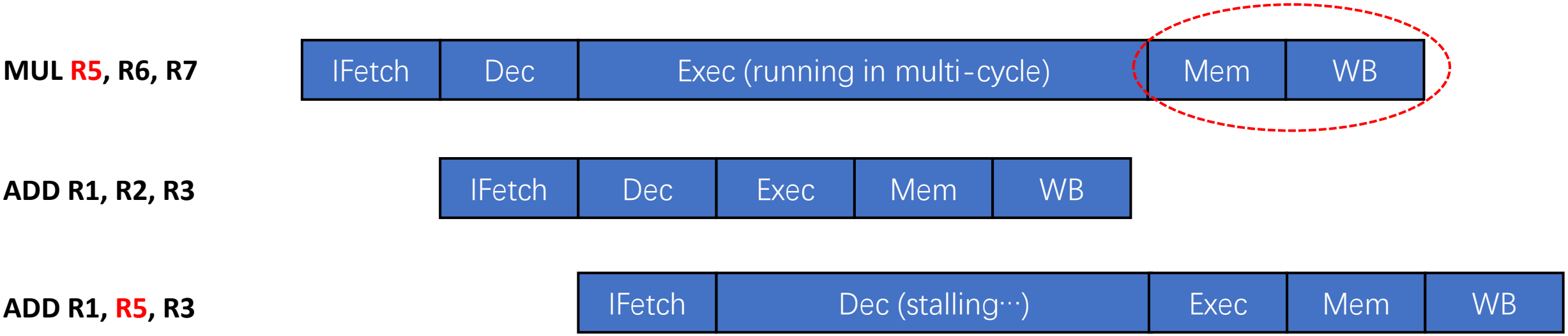
SUSTech Southern University of Science and Technology



Non-stalling for Mcycle / 思路提示

SUSTech Southern University of Science and Technology

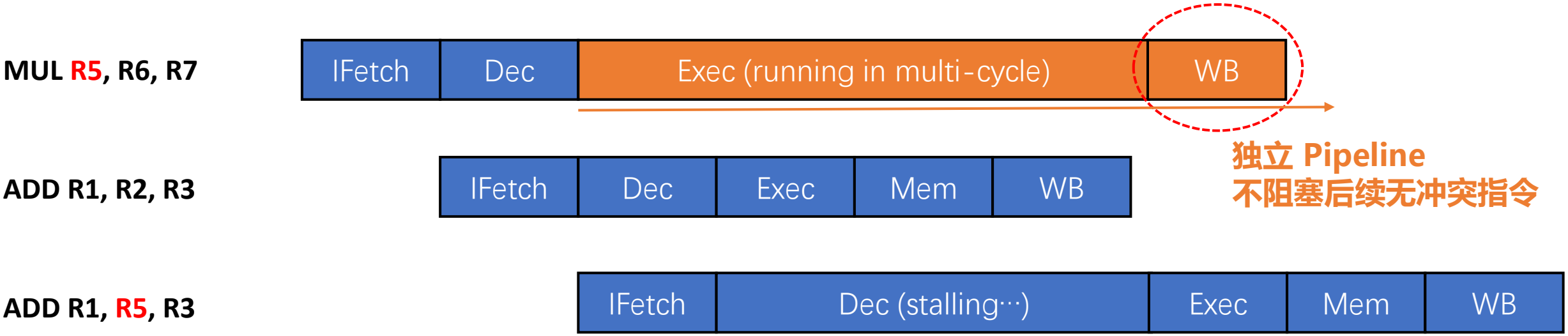
思考：能否只做 WB？



Non-stalling for Mcycle / 思路提示

SUSTech Southern University of Science and Technology

跳出来：独立通道写入 Registers ?

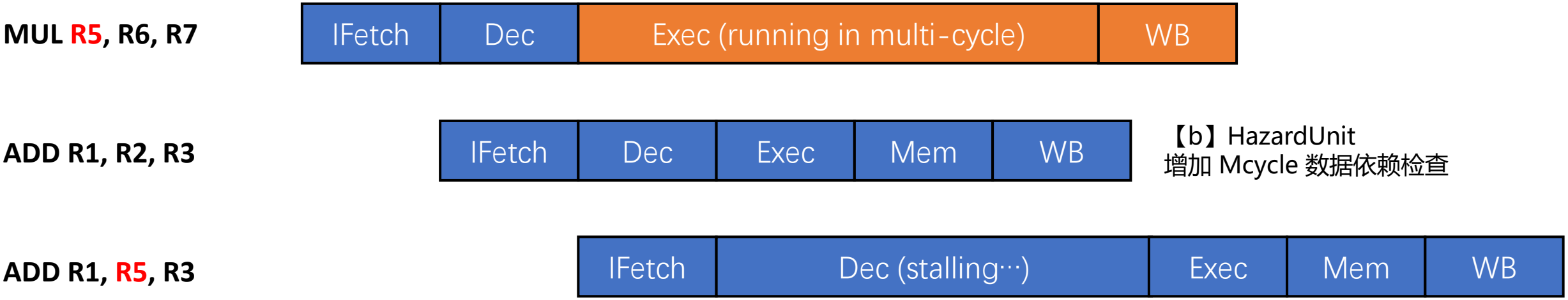


独立 Pipeline
不阻塞后续无冲突指令

Non-stalling for Mcycle / 基础思路 -> 完善细节

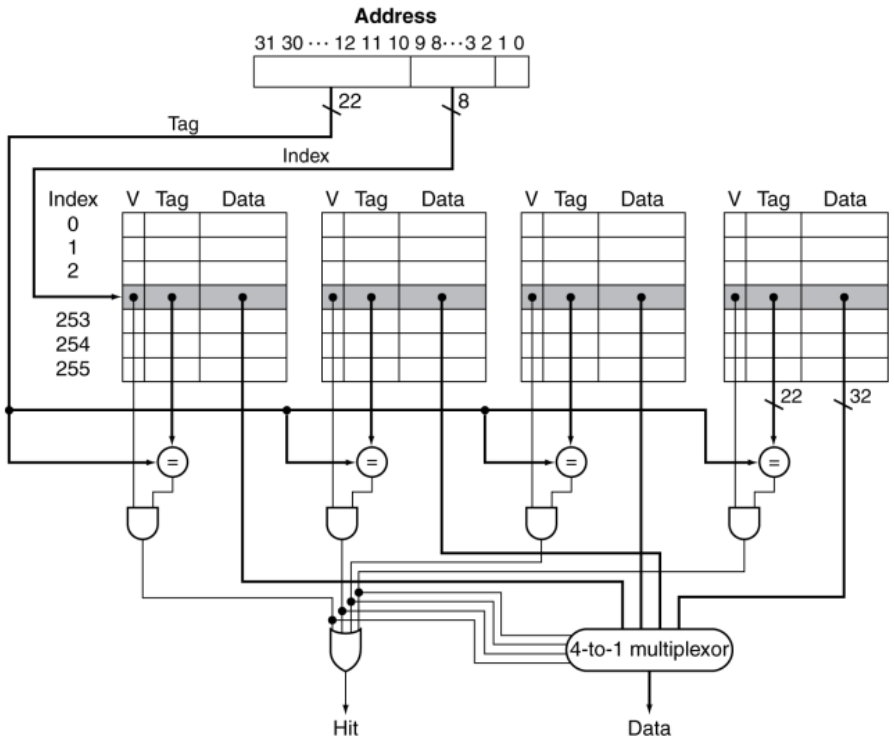
SUSTech Southern University of Science and Technology

【a】RegisterFiles
接收Mcycle输出，同样也在时钟下降沿写入?



【b】HazardUnit
增加 Mcycle 数据依赖检查

Hints / Cache



In this task, you will implement a **4-way set associative cache** between memory and the ARM CPU. The schematic of the 4-way set associative cache is shown above. The cache size is **4KB** (256 rows x 4 ways x 4 bytes). The cache uses **write-allocate** and **write-back** scheme. Integrating this cache will further add complexity to the Store and Load instructions. There are 4 situations when accessing the cache:

(1) When **read hit**, directly load data from cache to register.

(2) When **read miss**, load data from memory to cache, then load data from cache to register.

(3) When **write hit**, write to cache only, but set the block "dirty", write back to memory when dirty data is replaced (write-back strategy).

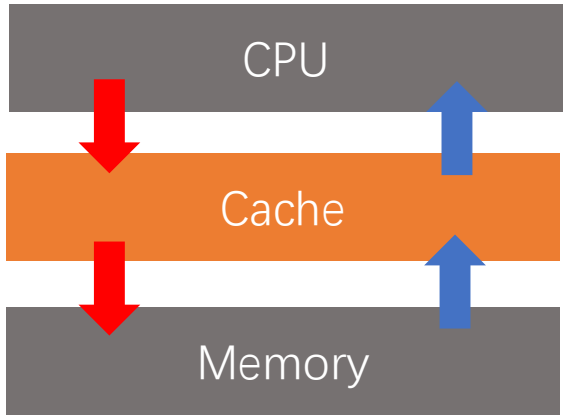
(4) When **write miss**, if the block is dirty, write the dirty block to memory, and load the data to replace the cache (write-allocate); if not dirty, directly write to memory, and then load the data into the cache.

Cache / 模块设计：从接口入手

SUSTech Southern University of Science and Technology

思考点

- 输入，输出?
- 输入到输出，是否有延时 (多少个时钟)?
- 模块间如何交互/连接?



参考设计

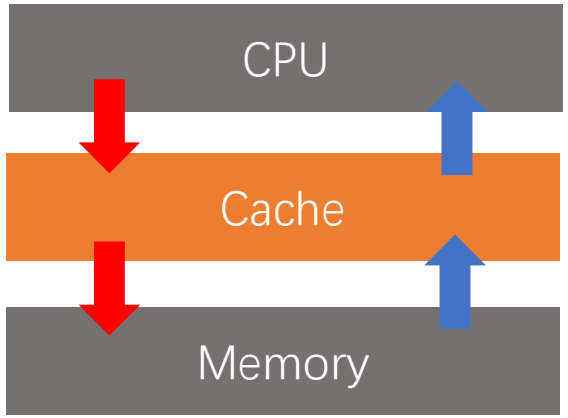
```
1   module AssociativeCache4Way1W #(
2       parameter ROW = 256,
3       parameter ROW_WIDTH = 8,
4       parameter TAG_WIDTH = 22,
5       parameter FLAG_RST = 256'b0
6   ) (
7       input CLK,
8       input Reset,
9
10      input WriteEnable,              // from CPU instr / STR
11      input  [31:0] RWAddr,           // from CPU instr / LRD
12      input  [31:0] WriteData,        // from CPU instr / STR
13      output reg [31:0] ReadData,     // to CPU Registers
16
21      output reg [31:0] MemReadAddr,  // to Data Memory
22      input  [31:0] MemReadData,      // from data memory
23      output reg [31:0] MemWriteAddr, // to Data Memory
24      output reg [31:0] MemWriteData  // to Data Memory
25  );
```

Cache / 模块设计：从接口入手

SUSTech Southern University of Science and Technology

思考点

- 输入，输出?
- 输入到输出，是否有延时 (多少个时钟)?
- 模块间如何交互/连接?



参考设计

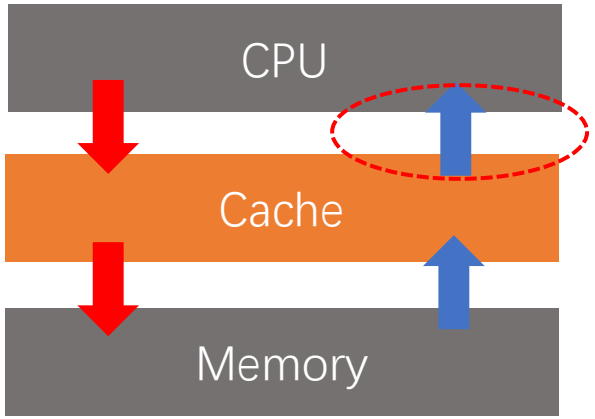
```
module AssociativeCache4Way1W #(
    parameter ROW = 256,
    parameter ROW_WIDTH = 8,
    parameter TAG_WIDTH = 22,
    parameter FLAG_RST = 256'b0
) (
    input CLK,
    input Reset,

    input WriteEnable,              // from CPU instr / STR
    input  [31:0] RWAddr,           // from CPU instr / LRD
    input  [31:0] WriteData,        // from CPU instr / STR
    output reg [31:0] ReadData,     // to CPU Registers
    output reg WriteReady,          // to CPU control unit
    output reg ReadReady,           // to CPU control unit

    input MemReadFinish,            // from data memory
    input MemWriteFinish,           // from data memory
    output reg MemReadStart,        // to Data Memory
    output reg MemWriteStart,       // to Data Memory
    output reg [31:0] MemReadAddr,  // to Data Memory
    input  [31:0] MemReadData,      // from data memory
    output reg [31:0] MemWriteAddr, // to Data Memory
    output reg [31:0] MemWriteData  // to Data Memory
);
```

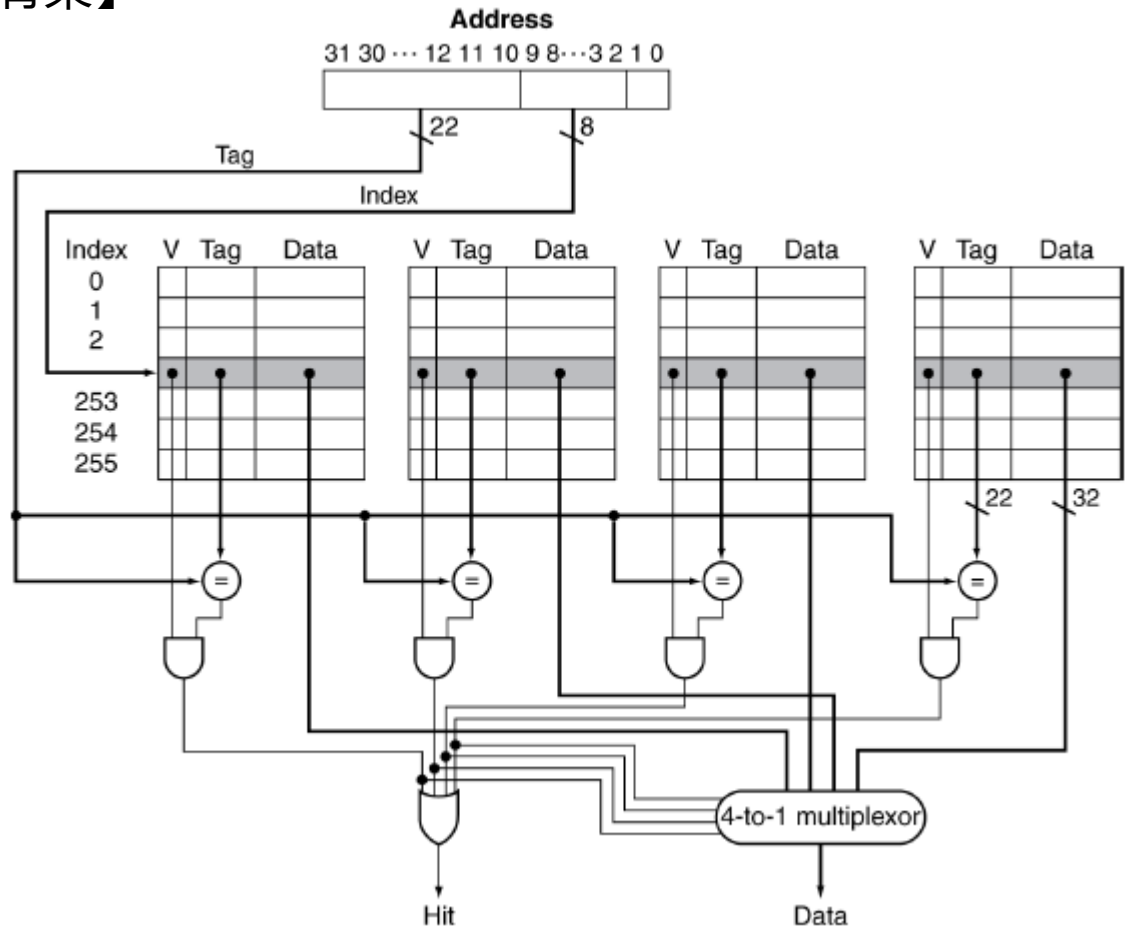
Cache / 模块设计：内部实现

SUSTech Southern University of Science and Technology

逐步迭代

- 全局切入点：CPU read hit Cache 【简单、清晰，形成骨架】



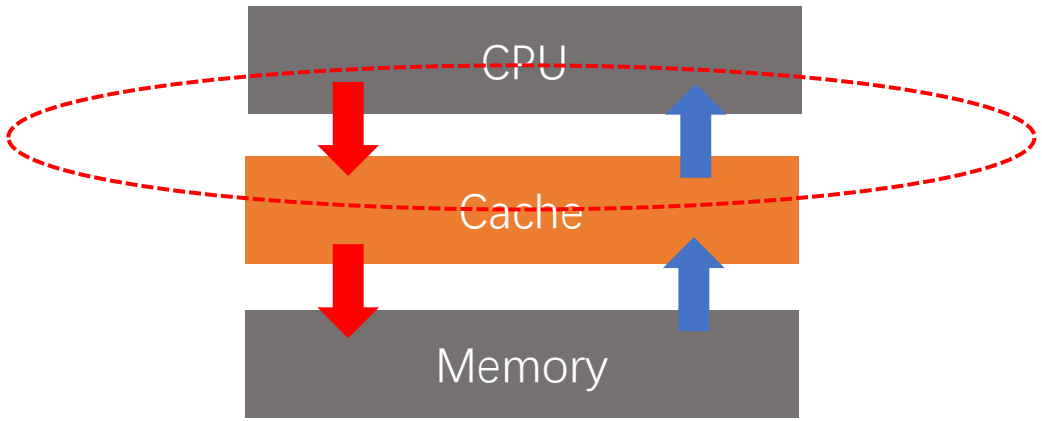


Cache / 模块设计：内部实现

SUSTech Southern University of Science and Technology

逐步迭代

- 全局切入点：CPU read hit Cache 【简单、清晰，形成骨架】
- “写入”切入：CPU write hit / empty Cache【跑通 CPU <-> Cache 链路，确保部分测试通过】



思考：当前版本，能跑通 Lab2 测试吗?

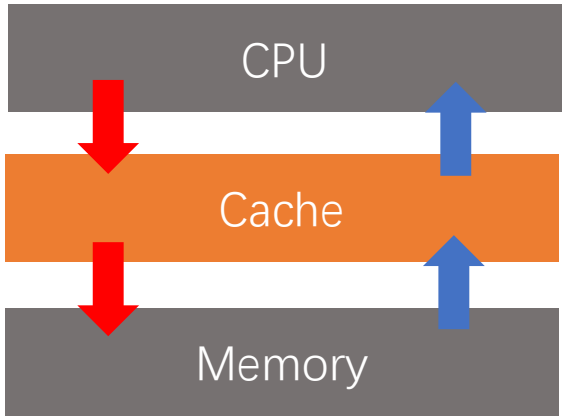
- 如果：Cache 只对 Data RW Memory 作用
- 如果：Cache 同时对 Data RO & RW Memory 作用

Cache / 模块设计：内部实现

SUSTech Southern University of Science and Technology

逐步迭代

- 全局切入点：CPU read hit Cache 【简单、清晰，形成骨架】
- “写入”切入：CPU write hit / empty Cache【跑通 CPU <-> Cache 链路，确保部分测试通过】
- “读取”迭代：CPU read empty Cache，which need read from Memory
- “写入”迭代：CPU write dirty Cache，which need write-back Memory
- “读取”迭代：CPU read dirty Cache，which need write-back Memory, and read from Memory



思考：当前版本，能跑通 Lab2 测试吗?

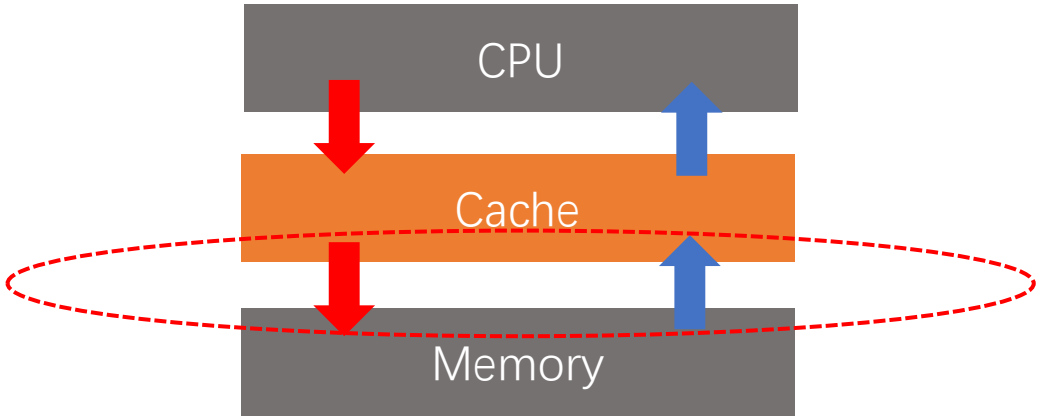
=> 数码管输出怎样处理?

Cache / 模块设计：内部实现

SUSTech Southern University of Science and Technology

逐步迭代

- 全局切入点：CPU read hit Cache 【简单、清晰，形成骨架】
- “写入”切入：CPU write hit / empty Cache【跑通 CPU <-> Cache 链路，确保部分测试通过】
- “读取”迭代：CPU read empty Cache，which need read from Memory
- “写入”迭代：CPU write dirty Cache，which need write-back Memory
- “读取”迭代：CPU read dirty Cache，which need write-back Memory, and read from Memory



- **考虑到真实 memory 读写存在时延**
- **涉及 CPU Pipeline Stalling**

Cache / 工作量预估

SUSTech Southern University of Science and Technology

```
[master 6d5ab39] add AssociativeCache4Way1W
 2 files changed, 643 insertions(+)
 create mode 100644 vivado_proj/sme309_armv3/sme309_armv3.srcs/sim_1/new/tb_Cache.v
 create mode 100644 vivado_proj/sme309_armv3/sme309_armv3.srcs/sources_1/new/AssociativeCache4Way1W.v
```

注意: 部分综合器 对 signed

支持不足!

建议采用 60-62行的方式，对两个有符号数进行比较!

```
29  module led(
30      input in,
31      output [7:0] out
32  );
33
34      reg [31:0] r1;
35      reg [31:0] r2;
36
37      always @(*) begin
38          if (in) begin
39              r1 <= 32'hffff_ffff; // -1
40              r2 <= 32'd1;
41          end
42          else begin
43              r1 <= 32'd2;
44              r2 <= 32'd3;
45          end
46      end
47
48      wire signed [31:0] sr1;
49      wire signed [31:0] sr2;
50      assign sr1 = r1;
51      assign sr2 = r2;
52
53      wire o7;
54      assign o7 = r1 < r2 ? 1'b1 : 1'b0;
55
56      wire o6;
57      assign o6 = sr1 < sr2 ? 1'b1 : 1'b0;
58
59      wire o5;
60      assign o5 = (~r1[31] & r2[31]) ? 1'b0 :  // r1 >= 0, r2 < 0
61                  (r1[31] & ~r2[31]) ? 1'b1 :  // r1 < 0, r2 >=0
62                  (r1 < r2);
63
64      assign out = {o7, o6, o5, {5{in}}};
65
66  endmodule
```

Project Intro / Outline

SUSTech Southern University of Science and Technology

- Git
- RISC-V
- Hints
- **Lab Review & Summary**

Lab Review

SUSTech Southern University of Science and Technology

Lab1
Keil & Vivado

Lab2
单周期 CPU

Lab3
+ 多周期乘除法

Project
+ Pipelined / Cache / FPU / …

Lab Review

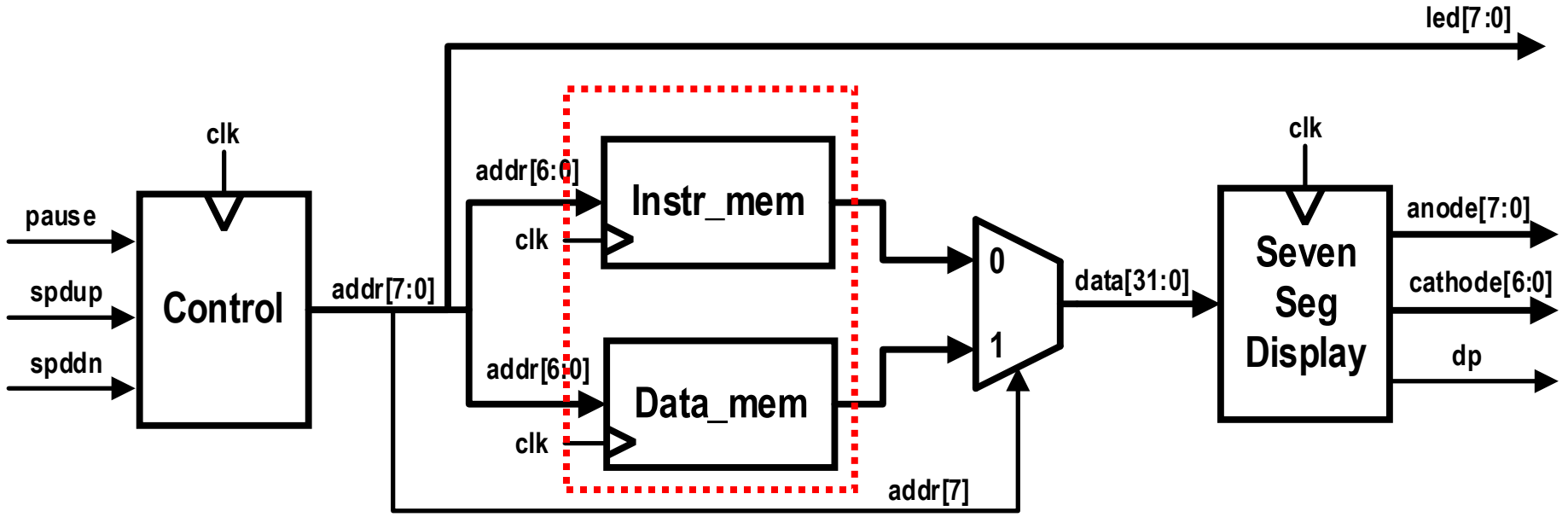
SUSTech Southern University of Science and Technology

Lab1
Keil & Vivado

Lab2
单周期 CPU

Lab3
+ 多周期乘除法

Project
+ Pipelined / Cache / FPU / …



Lab Review

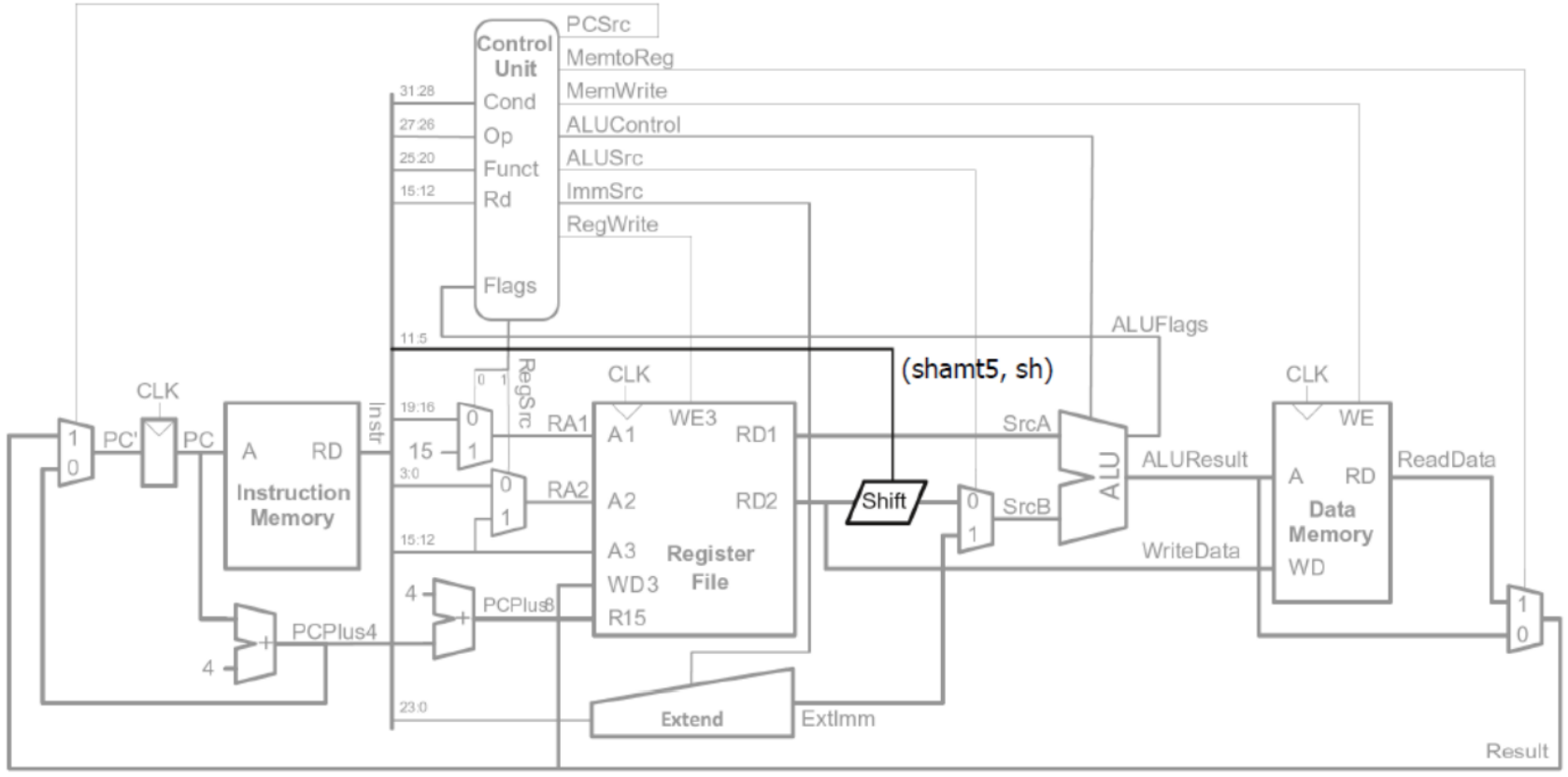
SUSTech Southern University of Science and Technology

Lab1
Keil & Vivado

Lab2
单周期 CPU

Lab3
+ 多周期乘除法

Project
+ Pipelined / Cache / FPU / …



Lab Review

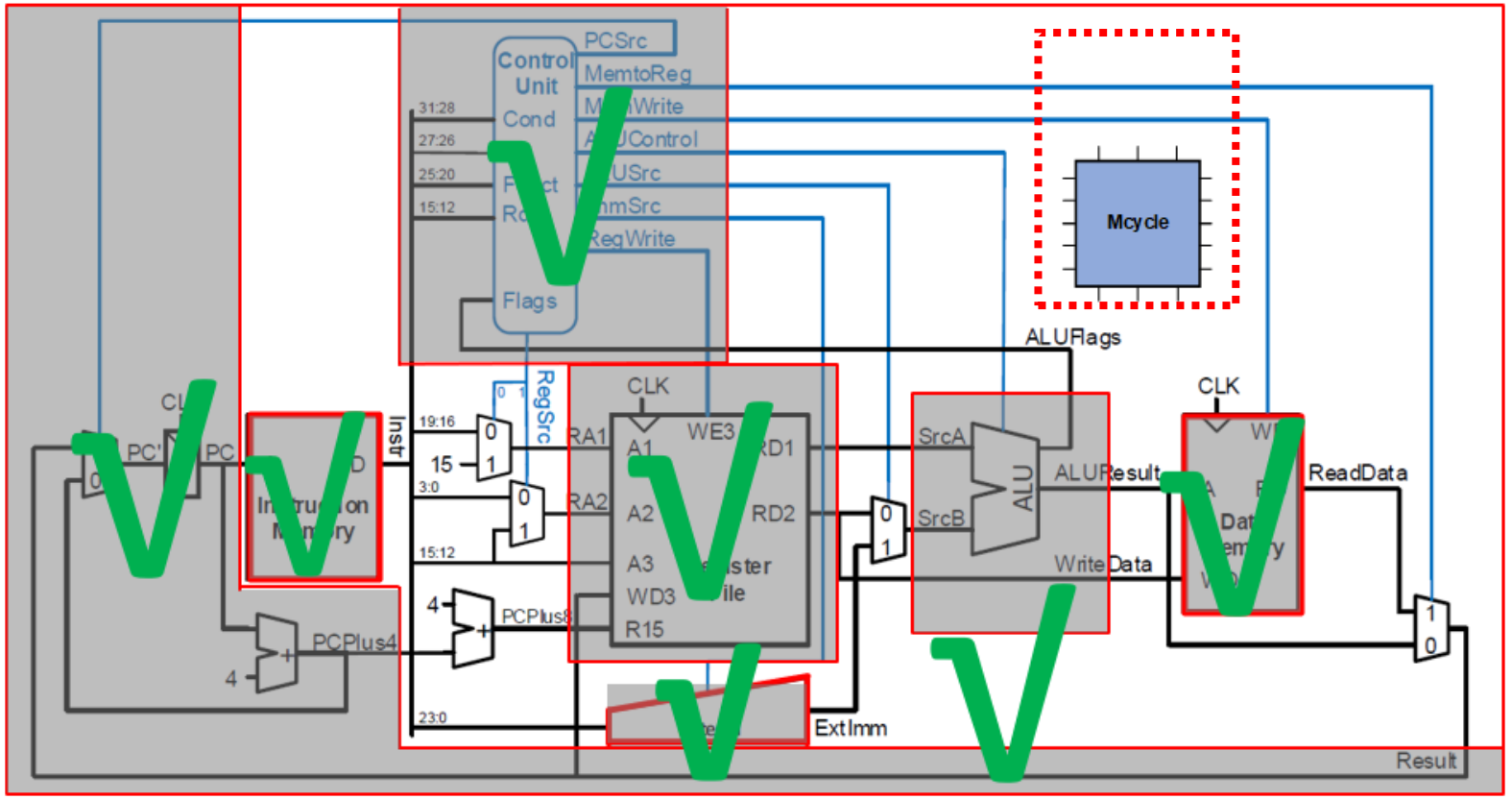
SUSTech Southern University of Science and Technology

Lab1
Keil & Vivado

Lab2
单周期 CPU

Lab3
+ 多周期乘除法

Project
+ Pipelined / Cache / FPU / …



Lab Review

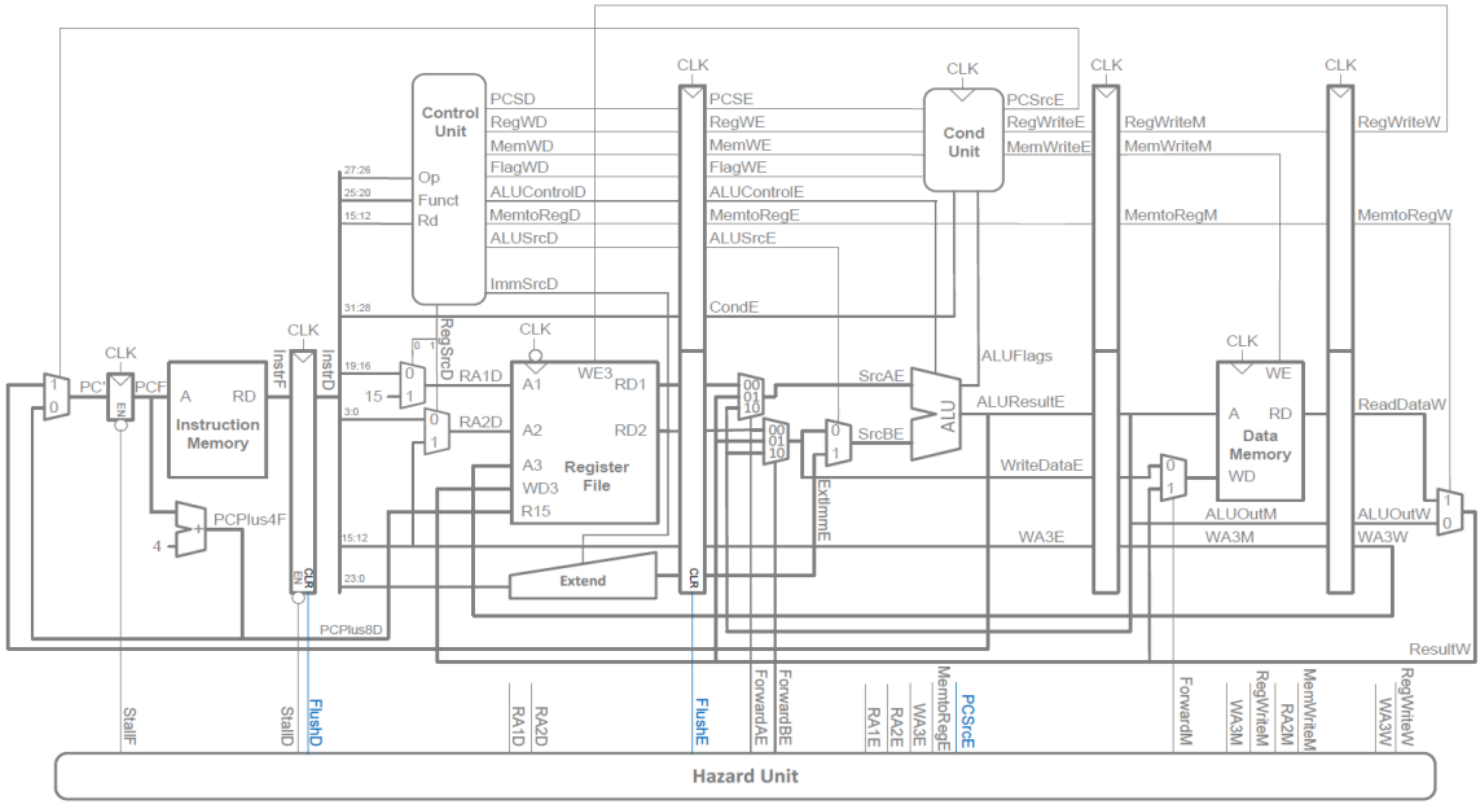
SUSTech Southern University of Science and Technology

Lab1
Keil & Vivado

Lab2
单周期 CPU

Lab3
+ 多周期乘除法

Project
+ Pipelined /
Cache / FPU / …



SUSTech Southern University of Science and Technology

THANK YOU