B0911007Y 2019-2020学年春季学期

计算机组成原理实验

实验项目2 单周期处理器设计

常轶松 陈欲晓

2020年4月3日/10日

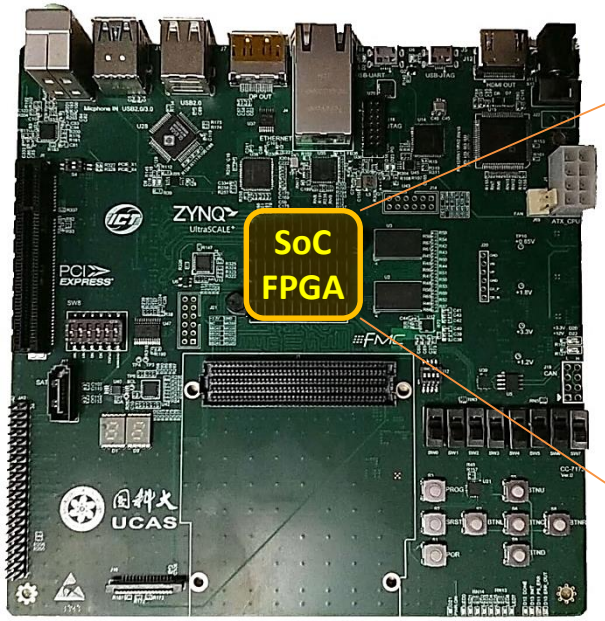
中国科学院大学 University of Chinese Academy of Sciences
中科院计算所 INSTITUTE OF COMPUTING TECHNOLOGY,CAS

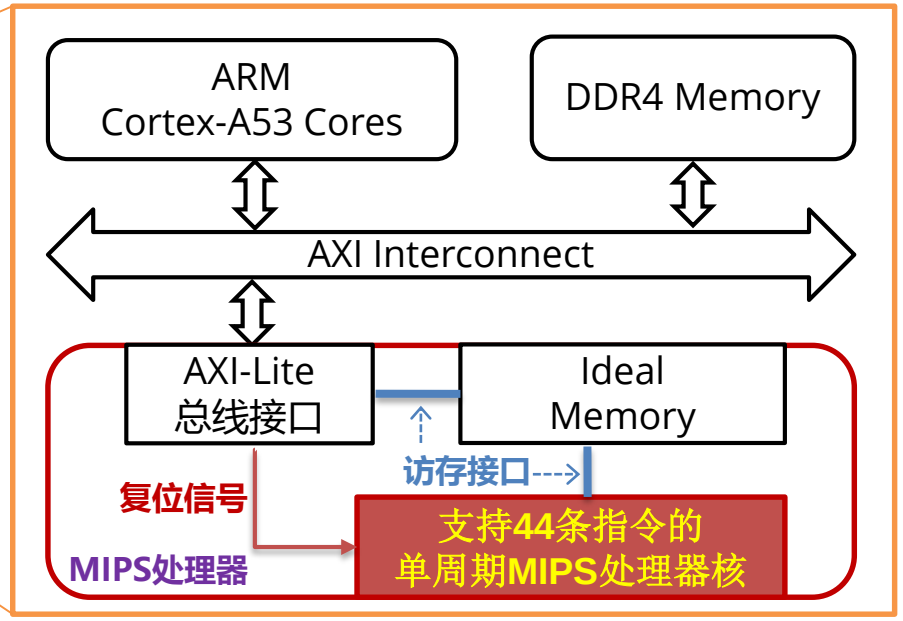
0.1 实验目的

❑设计可运行MIPS指令子集的简易单周期处理器

- 通过实现基于理想内存（ideal memory）的单周期MIPS处理器的数据通路和控制单元
  - 深入理解处理器取指、译码、执行、访存、写回五个处理阶段的逻辑功能
  - 理解MIPS指令格式，能够对照指令手册实现并调试自研简单处理器
  - 初步掌握基于基准测试程序（benchmark）评测处理器基本功能的设计思想

0.2 实验环境及工程框架

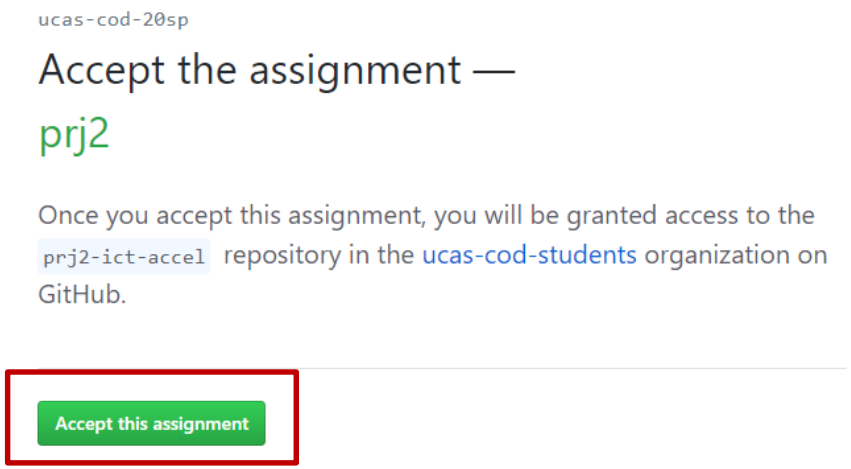




- 实现可支持44条指令执行的单周期MIPS处理器核
- 复用实验项目1中设计的通用寄存器堆（Register File）和算术逻辑单元（ALU）作为MIPS处理器核的基本可用组件
- 基于ARM辅助，使用30个benchmark测试单周期MIPS处理器核（benchmark分为basic、medium和advanced三组）

0.3 实验项目发布与接收流程（1）

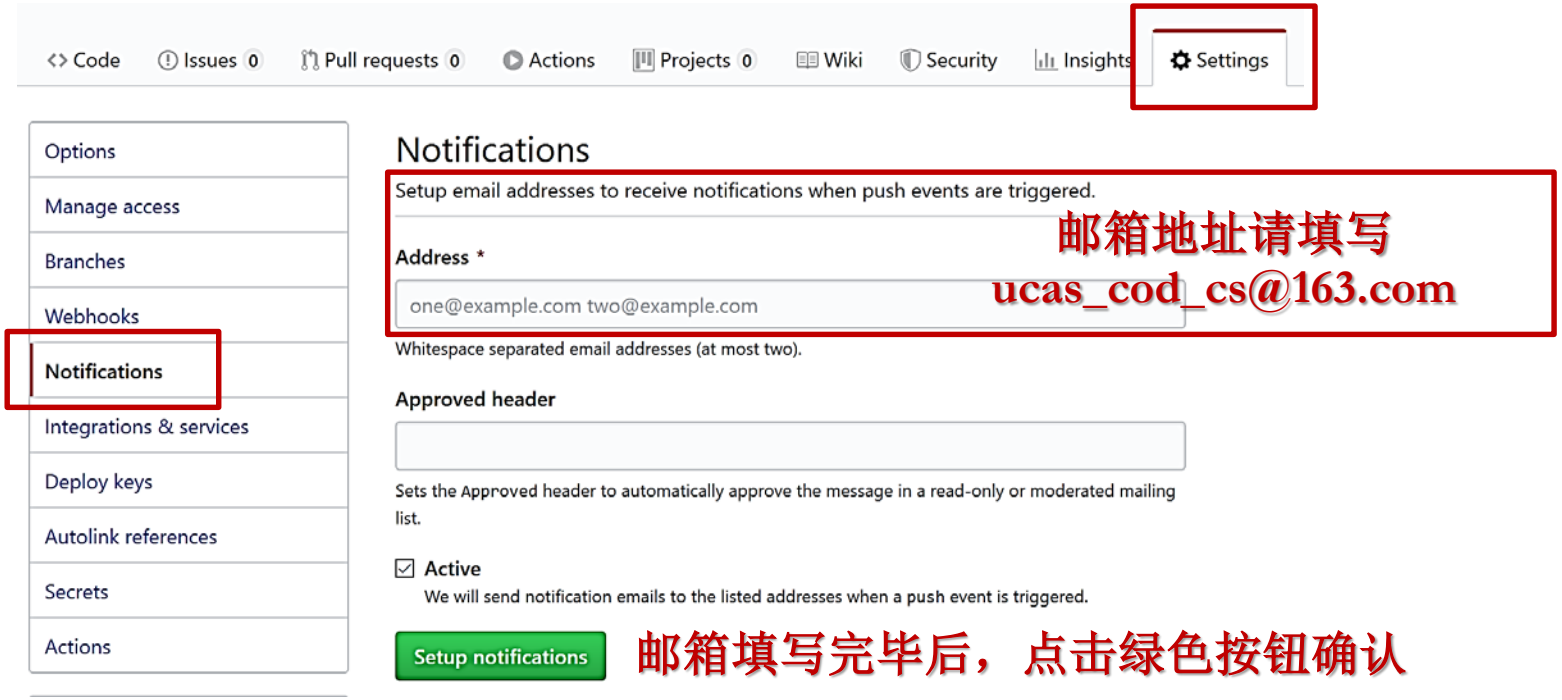
- **实验项目2参与邀请链接URL**
  - **https://classroom.github.com/a/QVxFSxfF**
  - 请同学们在浏览器中输入上述URL
    - 点击绿色<Accept this assignment>按钮接受本次作业安排（如下图标出）
    - 个人远程仓库链接：https://github.com/ucas-cod-students/prj2-xxx（xxx为个人GitHub用户名）
    - 使用git clone https://github.com/ucas-cod-students/prj2-xxx命令，在虚拟机中创建本地仓库



0.3 实验项目发布与接收流程（2）

❑非常重要：与实验成绩打分有关！！！

❑ 在浏览器中，点击个人远程仓库Settings -> Notification，进入如下界面

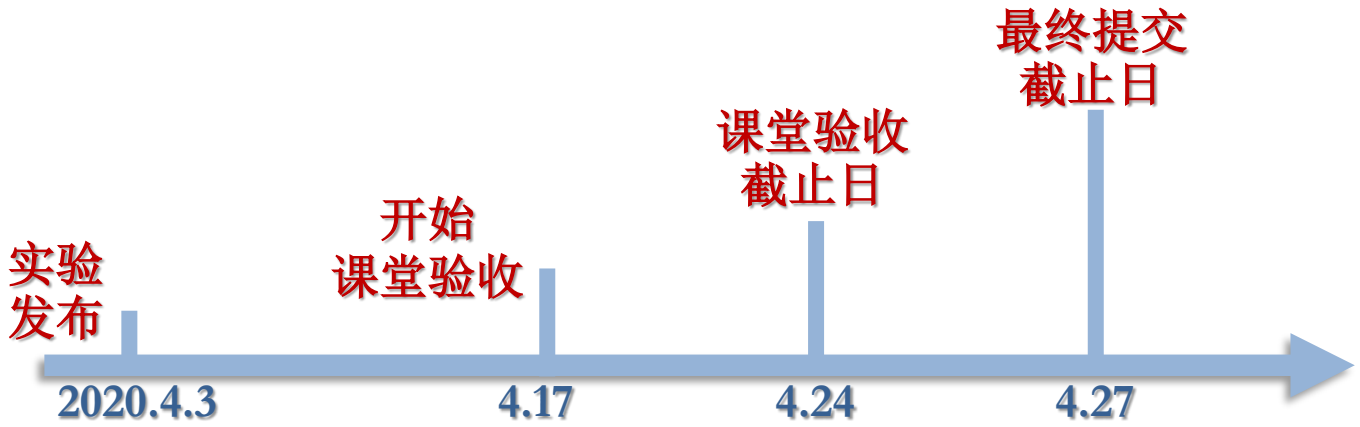


不能访问GitHub的同学接收作业

姓名	远程仓库地址
热伊莱	https://bitbucket.org/chang-steve/prj2-reyilai
热依汉古丽	https://bitbucket.org/chang-steve/prj2-reyihanguli
华为	https://bitbucket.org/chang-steve/prj2-alice-bor

- 请上述三名同学，首先查看邮箱，点击接收作业的邀请
- 之后在虚拟机里使用git clone命令，按对应远程仓库地址，创建本地仓库
- 如果访问有问题，请及时联系助教老师（changyisong@ict.ac.cn）

0.4 实验项目进度安排



- 本次实验不设阶段提交
- 自4月17日上课时间起开始进行课堂验收
- 课堂验收截止：4月24日下课时（17:49:59）
- 最终提交截止时间前（4月27日23:59:59），需提交完整RTL代码、FPGA云环境运行结果及实验报告
  - 执行命令：git push origin master

0.5 课堂验收要求

❑ 课堂验收要求

- 检查单周期MIPS处理器的最终实现在FPGA云环境下运行全部benchmark的执行情况
  - 先后运行basic、medium和advanced三组测试用例，检查30个测试程序是否全部运行通过（也可查看相应的运行日志文件）

- 检查RTL代码
  - 代码对应单周期处理器电路结构图
  - 按指令格式和指令类型设计处理器译码逻辑
  - 组合逻辑必须使用assign 语句描述
  - 时序逻辑PC的Verilog HDL描述是否符合代码规范（包括复位信号使用、赋值条件要互斥等要求）

0.6 基于理想内存的多周期处理器选做实验项目

- 在本次实验项目截止日前，鼓励同学们选做多周期处理器扩展实验，但不强制要求所有同学完成
  - 选做扩展实验并完成较好的同学，可取得实验项目2浮动加分
- 多周期处理器选做实验的参与邀请链接URL
  - https://classroom.github.com/a/n-3zCbJz
  - 个人远程仓库链接：https://github.com/ucas-cod-students/prj2-multi-cycle-xxx（xxx为个人GitHub用户名）
  - 重要，请务必执行：在浏览器中，点击个人远程仓库Settings -> Notification，设置邮箱地址ucas_cod_cs@163.com
  - 不能访问GitHub的同学，请在需要时联系助教老师创建选做实验仓库
- 课堂验收时请提醒助教验收自己的多周期处理器设计
- 最终提交：完整RTL代码及FPGA云环境运行结果
  提交命令： git push origin master
- 无需为选做实验单独撰写实验报告，可以把选做实验的关键点写在实验项目2的实验报告中，并从实验项目2的仓库中提交

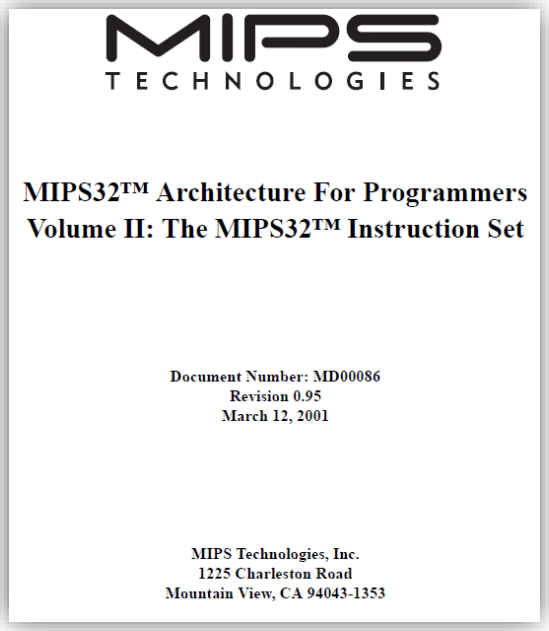
内容大纲

- 实验要点讲解
  - 单周期MIPS处理器
  - MIPS处理器基准测试程序集（benchmarks）
- 实验操作流程
- 其他事项

1.1 需支持的44条MIPS指令

指令分类	各阶段需完成的指令
运算类指令（14条）	addiu, addu, subu, and, andi, nor, or, ori, xor, xori, slt, slti, sltu, sltiu
移位指令（6条）	sll, sllv, sra, srav, srl, srlv
跳转类指令（9条）	bne, beq, bgez, blez, bltz, j, jal, jr, jalr
访存类指令（12条）	lb, lh, lw, lbu, lhu, lwl, lwr, sb, sh, sw, swl, swr
数据移动及立即数指令（3条）	movn, movz, lui

- 指令的具体格式和详细含义请参考指令手册
  “The MIPS32 Instruction Set”
  - 已在SEP网站发布


MIPS TECHNOLOGIES

MIPS32™ Architecture For Programmers
Volume II: The MIPS32™ Instruction Set

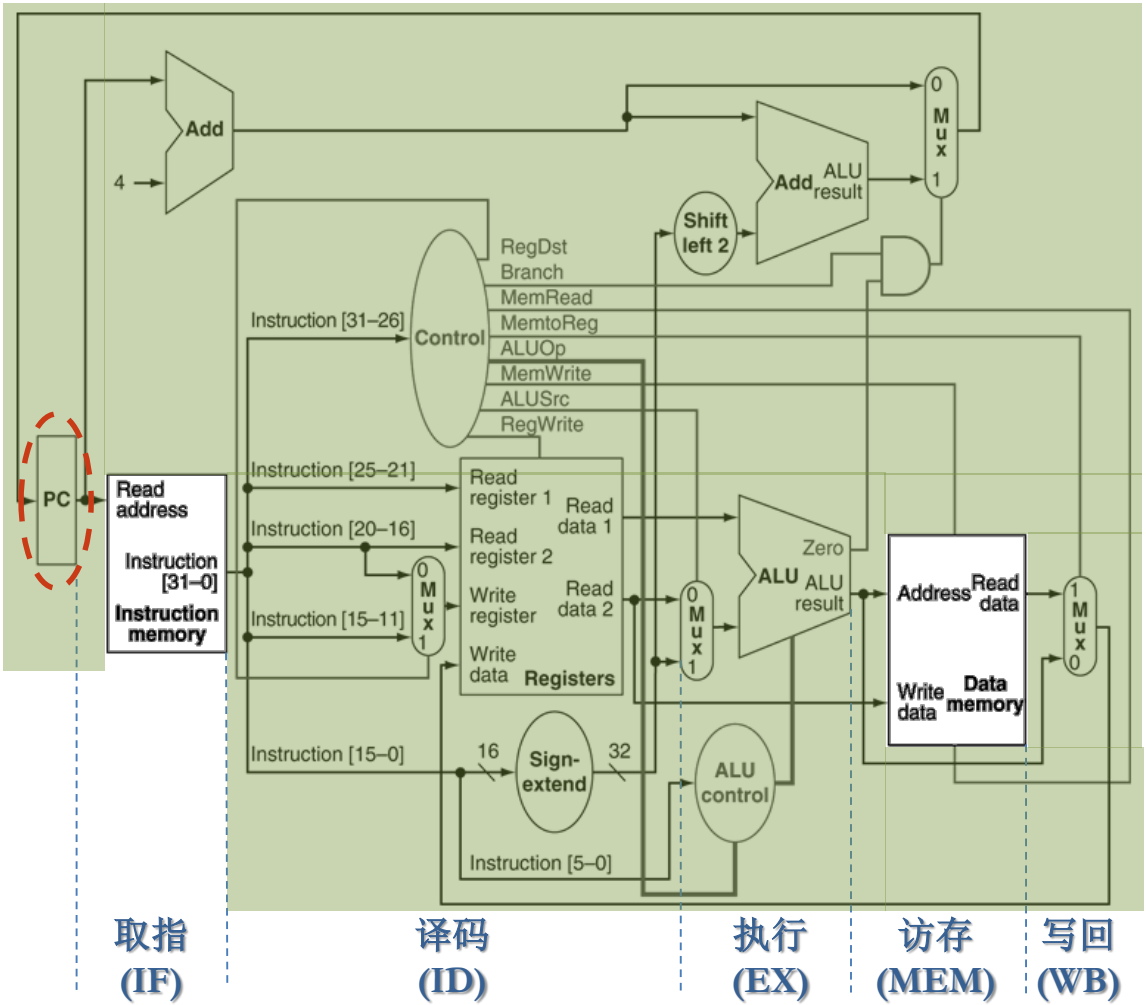
Document Number: MD00086
Revision 0.95
March 12, 2001

MIPS Technologies, Inc.
1225 Charleston Road
Mountain View, CA 94043-1353

1.2 单周期MIPS处理器接口定义

信号名	I/O	说明
rst	Input	同步高电平复位信号
clk	Input	时钟
PC[31:0]	Output	程序计数器，复位后初值为32’d0
Instruction[31:0]	Input	从内存（Memory）中读取至处理器的指令
Address[31:0]	Output	数据访存指令使用的内存地址
MemWrite	Output	内存访问的写使能信号（高电平有效）
Write_data[31:0]	Output	内存写操作数据
Write_strb[3:0]	Output	内存写操作字节有效信号（支持16-bit/8-bit内存写） Write_strb[i] == 1表示 Write_data[8 × (i + 1) - 1 : 8 × i]位会被写入 内存的对应地址
MemRead	Output	内存访问的读使能信号（高电平有效）
Read_data[31:0]	Input	从内存中读取的数据

1.3 单周期MIPS处理器结构图

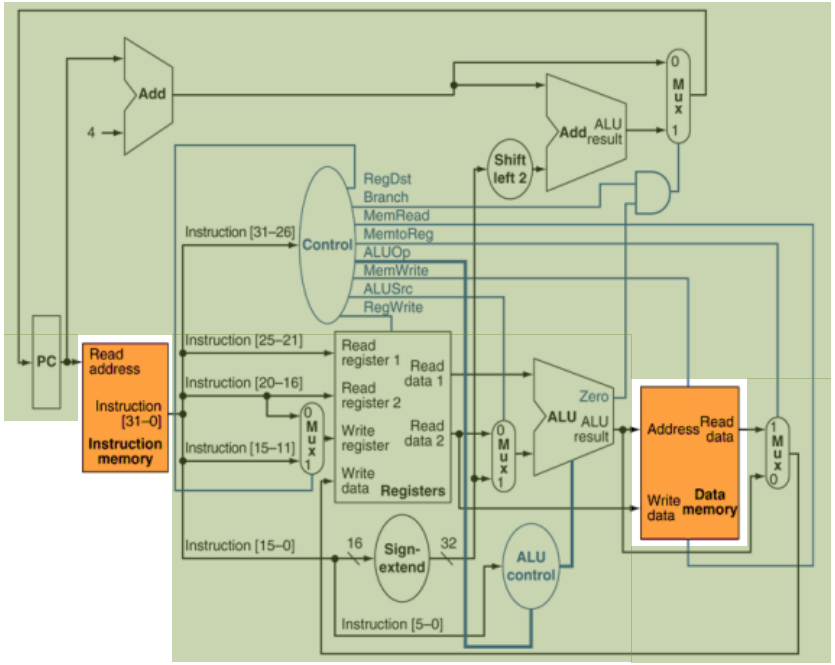


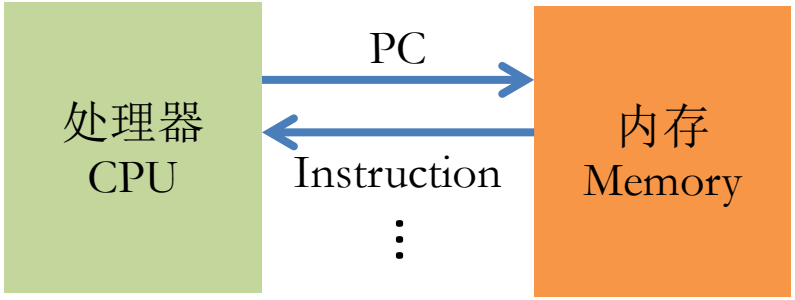
- 本次实验项目需实现图中绿色部分（即指令内存Instruction memory和数据内存Data memory之外的所有逻辑电路）
- ALU及通用寄存器堆（图中Registers）已在实验项目1中实现，可直接使用
  - 某些指令需要在ALU中添加新的操作，可自行修改ALU代码
- PC是一个寄存器（时序电路）
- 结构图与表格输入输出端口信号名可能有少量不符，原则上不影响理解
  - 若仍然有问题请联系助教老师

1.3.1 统一的指令和数据内存

- **指令内存和数据内存（橙色标注）**
  - 逻辑上（结构图）为两个
  - 物理上（RTL代码）可实现成一个
  - 目前设置的内存容量为4KB，按字节编址

- **内存（Memory）在处理器外部，需注意模块信号方向**
  - 例如：处理器向外发送PC
    - PC是处理器的输出端口
  - 又例如：处理器从外部获得指令
    - Instruction是处理器的输入端口
  - 同理可推导数据访存相关信号的输入输出方向

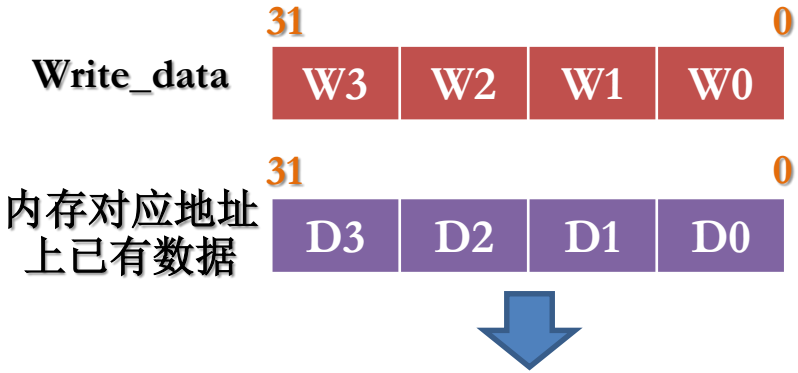




1.3.2 理想内存（Ideal Memory）

- **假设内存访问与寄存器访问具有相同的延时**
  - 确保指令单周期处理
    - 读/写操作全部在一个时钟周期内完成
    - 同步写，异步读
- **通过2读1写端口支持指令和数据访问**
  - 读端口1用于指令读，读端口2用于数据读
  - 写端口用于数据写
- **理想内存实现代码助教已写好**
  - 代码位置：hardware/sources/hdl/ideal_mem.v
  - 已在hardware/sources/hdl/mips_cpu_top.v中对理想内存进行实例化
  - **无需同学们进行任何其他操作**

1.4 内存Write_strb信号使用



每个小方块代表
一个字节（8-bit）

写全字
(32-bit)

Write_strb = 4’b1111
写入后内存数据变为

W3 W2 W1 W0

写半字
(16-bit)

Write_strb = 4’b0011

D3 D2 W1 W0

Write_strb = 4’b1100

W3 W2 D1 D0

写字节
(8-bit)

Write_strb = 4’b0001

D3 D2 D1 W0

Write_strb = 4’b0010

D3 D2 W1 D0

Write_strb = 4’b0100

D3 W2 D1 D0

Write_strb = 4’b1000

W3 D2 D1 D0

1.5 指令译码部分的逻辑设计

- 译码逻辑要充分利用指令格式和指令分类

- 不要每条指令单独译码，产生相应的控制信号（如下例）

```
assign RF_wen = (opcode == `SPECIAL)?( (Function == `ADDU || Function == `OR || Function == `SLT || Function == `SLL)?(1'b1):(1'b0) ):
                ( (opcode == `ADDIU || opcode == `LW || opcode == `SLL || opcode == `JAL || opcode == `LUI || opcode == `SLTI || opcod
assign wen = RF_wen;

assign RF_waddr = (opcode == `SPECIAL)?( (Function == `ADDU || Function == `OR || Function == `SLT)?(rd):
                ( (Function == `SLL)?(rd):(5'b0)) ):
                ( (opcode == `ADDIU || opcode ==  `LUI || opcode == `SLTI || opcode == `SLTIU || opcode == `LW)?(rt):
                ( (opcode == `JAL)?(5'd31):(5'b0)) );
assign waddr = RF_waddr;

assign RF_wdata = (opcode == `SPECIAL)?( (Function == `ADDU || Function == `OR || Function == `SLT)?(Result):(32'b0) ):
                ( (opcode ==  `ADDIU || opcode == `LUI || opcode == `SLTI || opcode == `SLL)?(Result):
                ( (opcode == `LW)?(Read_data):
                ( (opcode == `JAL)?(pcnext2):
                ( (opcode == `SLTIU)?({31'b0,CarryOut}):(32'b0) )) ) );
assign wdata = RF_wdata;

assign raddr1 = (opcode ==`SPECIAL)?( (Function == `ADDU || Function ==  `OR)?(rt):
                ( (Function == `SLT)?(rs):(5'b0)) ):
                ( (opcode == `SW || opcode == `LW)?(base):
                ( (opcode == `BNE || opcode == `BEQ)?(rt):(5'b0)));
```

内容大纲

- 实验要点讲解
  - 单周期MIPS处理器
  - MIPS处理器基准测试程序集（包含30个benchmarks）
- 实验操作流程
- 注意事项

1.5 benchmark简介 (30+1)

用户自定义
指令流序列测试激励，
仅支持仿真使用
（后续介绍激励添加方法）

basic组（共计1个）		medium组（共计12个）		advanced组（共计17个）	
序列号	benchmark名	序列号	benchmark名	序列号	benchmark名
00	test	01	sum	01	shuixianhua
01	memcpy	02	mov-c	02	sub-longlong
		03	fib	03	bit
		04	add	04	recursion
		05	if-else	05	fact
		06	pascal	06	add-longlong
		07	quick-sort	07	shift
		08	select-sort	08	wanshu
		09	max	09	goldbach
		10	min3	10	leap-year
		11	switch	11	prime
		12	bubble-sort	12	mul-longlong
				13	load-store
				14	to-lower-case
				15	movsx
				16	matrix-mul
				17	unalign

阶段I

阶段II

阶段III

- 位于本地仓库benchmark目录下，存在三个分别以basic，medium和advanced命名的子目录
- 各benchmark的简单功能描述请查阅本地仓库的README.md文件

1.6 实验项目提供的benchmark相关文件

组目录 / 文件 / 子目录	basic	medium	advanced
bin	可在MIPS处理器硬件上被执行（运行）的二进制文件，不可读且不可修改，上板运行时加载到理想内存中		
disassembly	反汇编软件程序源码（*.S），可读但不可修改，用于仿真调试时对照查看		
sim	指令流序列测试激励（*.mem），可读但不可修改[注1]，仿真时自动加载到理想内存中		

[注1]basic组中的test.mem可由同学们自行修改

反汇编指令序列文件（*.S）格式举例

```
00000000 <start>:
内存地址
   0:   241a0001        li      k0,1
   4:   17400002        bnez    k0,10 <real_start>
   8:   00000000        nop
        32-bit指令码     汇编指令助记符
0000000c <global_result>:
   c:   ffffffff        sdc3    $31,-1(ra)
   ⋮
```

注意：指令助记符可能和指令手册有出入，此时请以指令码为准确定指令功能

```
241a0001
17400002
00000000
ffffffff
24040000
24050064
ac8400c8
24840004
   ⋮
```

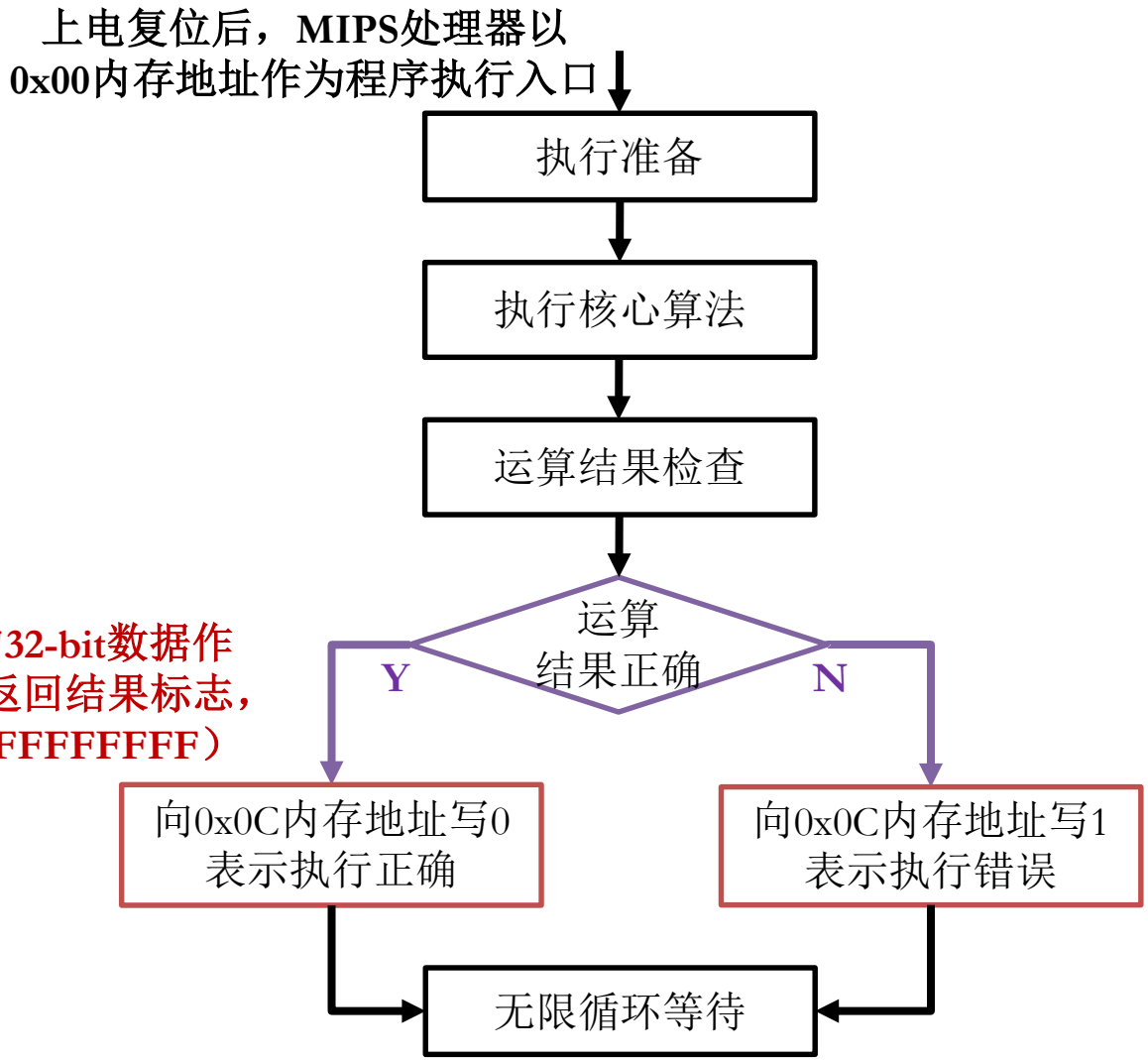
仿真测试激励文件（*.mem）格式

- 提取可执行文件的代码段（.text）、数据段（.data）、未初始化数据段（.bss）等，以十六进制文本格式，按地址偏移存放，并在仿真时被自动加载到理想内存中

自定义指令序列test.mem（basic组00号）

- 位于本地仓库benchmark/basic/sim目录内
  - 初始为空文件
  - 同学们可自行编辑
    - 每一行代表一条十六进制的32-bit指令码
  - 可用于调试全部44条指令
- 仅用于仿真，不能用于上板运行测试
  - 仿真时间设置为2μs
  - 请根据仿真时间及Ideal Memory内存容量大小，控制要添加的指令码数量
- 可根据个人调试情况，自行选择是否使用，不强制要求

benchmark可执行程序基本执行流程

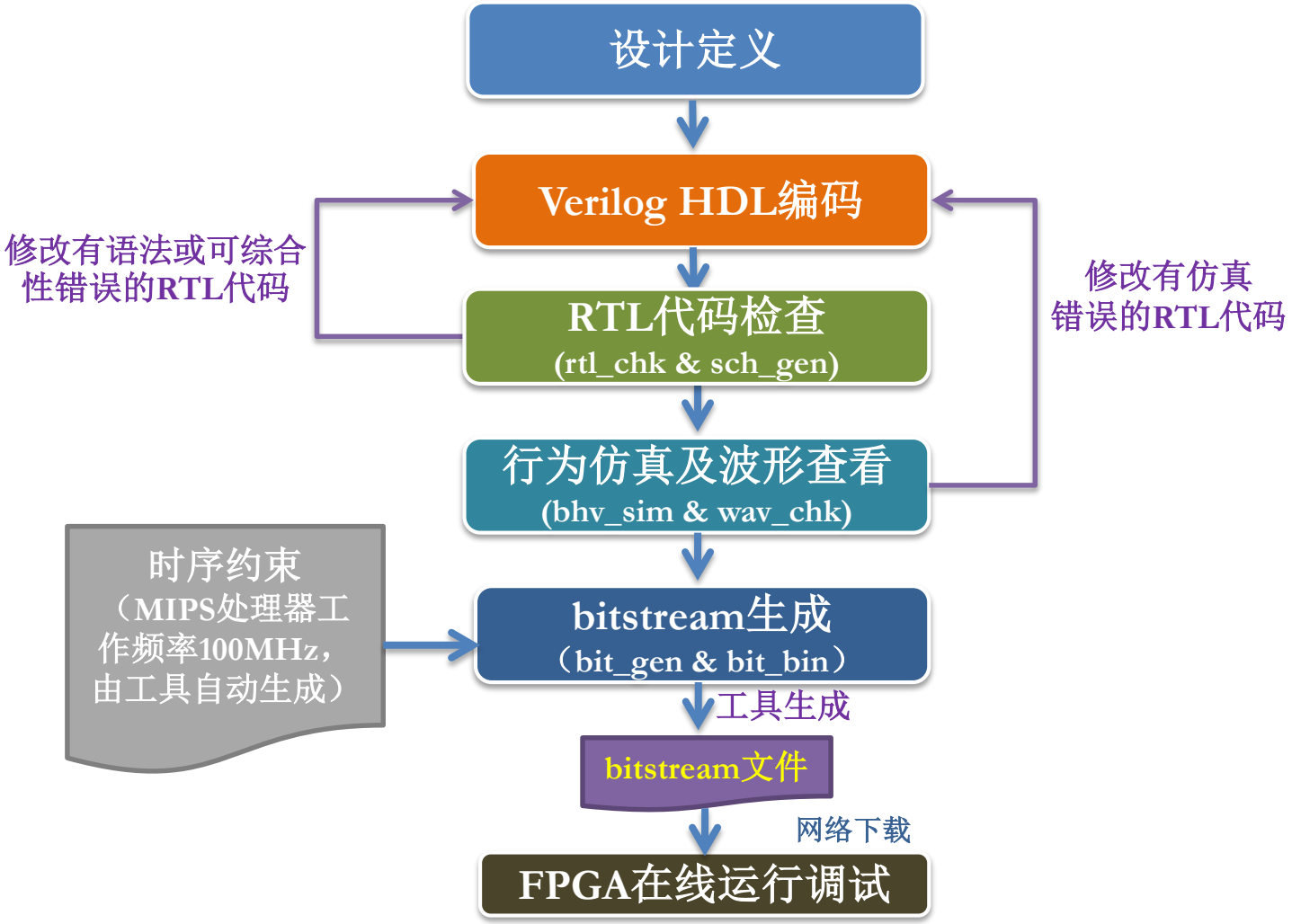


0x0C地址开始的32-bit数据作为benchmark的返回结果标志，初始值为-1 （0xFFFFFFFF）

内容大纲

- 实验要点讲解
- 实验操作流程
- 其他事项

实验流程


设计定义
Verilog HDL编码
修改有语法或可综合性错误的RTL代码
RTL代码检查
(rtl_chk & sch_gen)
修改有仿真错误的RTL代码
行为仿真及波形查看
(bhv_sim & wav_chk)
时序约束
（MIPS处理器工作频率100MHz，由工具自动生成）
bitstream生成
（bit_gen & bit_bin）
工具生成
bitstream文件
网络下载
FPGA在线运行调试

2.1 RTL代码编写

- MIPS处理器相关逻辑代码放在本地仓库hardware/sources/ip_catalog/mips_core/目录下

alu.v	复用实验项目1中设计的ALU模块
mips_cpu.v	MIPS处理器核顶层模块
reg_file.v	复用实验项目1中设计的reg_file模块

- 请将实验项目1中已设计好的alu.v和reg_file.v拷贝到上述目录
  - 某些指令需要在ALU中添加新的操作（如移位），可自行修改alu.v代码及输入接口的位宽定义

```
module mips_cpu(
    input  rst,
    input  clk,

    output reg [31:0] PC,
    input  [31:0] Instruction,

    output [31:0] Address,
    output MemWrite,
    output [31:0] Write_data,
    output [3:0] Write_strb,

    input  [31:0] Read_data,
    output MemRead
);

    // THESE THREE SIGNALS ARE USED IN OUR TESTBENCH
    // PLEASE DO NOT MODIFY SIGNAL NAMES
    // AND PLEASE USE THEM TO CONNECT PORTS
    // OF YOUR INSTANTIATION OF THE REGISTER FILE MODULE
    wire                RF_wen;
    wire [4:0]          RF_waddr;
    wire [31:0]         RF_wdata;

    // TODO: PLEASE ADD YOUT CODE BELOW
在此处插入同学们自己的逻辑代码
endmodule
```

不能修改这些信号的名称、位宽和变量类型

请仔细看上面的注释

2.2 RTL代码检查

- 在本地仓库顶层目录中执行
  make HW_ACT=rtl_chk vivado_prj

- 启动Vivado从顶层模块开始，递归地对设计中使用的全部模块做代码检查

- 请仔细核对出现的Warning和Error，并修改相应的RTL代码，直至没有Warning和Error出现

2.3 行为仿真

- 在本地仓库顶层目录中执行
  make HW_ACT=bhv_sim
  HW_VAL=<benchmark组名>:<benchmark组内序列号> vivado_prj
  - <benchmark组名>：basic/medium/advanced三选一
  - <benchmark组内序列号>有效值
    - basic组：00 - 01
    - medium组：01 - 12
    - advanced组：01 - 17
    - 注意：个位数的序列号前请务必加0（如序列号5务必输入为05）
  - 举例： make HW_ACT=bhv_sim HW_VAL=basic:01 vivado_prj
    - 使用basic组的memcpy测试程序进行仿真
    - 注意：basic和01之间的冒号（英文符号）不能去掉，且中间不能有空格
  - 无需编写testbench
  - 运行一次命令，仅能使用一个benchmark测试程序作为测试激励
    - 应逐一仿真调试实验项目提供的30个benchmark

2.3.1 行为仿真检查处理器执行结果

- **testbench会在每个时钟上升沿对指令的执行情况自动进行正确性判定（仿真basic:00学生自定义测试用例时除外）：**
  - PC寄存器值是否正确
  - 寄存器堆写使能信号（RF_wen）、写地址（RF_waddr）和写数据（RF_wdata）是否正确设置
  - 内存读写地址（Address）、写使能信号（MemWrite）、写数据（Write_data）和字节有效信号（Write_strb）是否正确设置

- **判定结果会在仿真运行时输出到屏幕上，并同时保存在仓库顶层目录下的hardware/vivado_out/run_log/bhv_sim.log中**

```
================================================
INFO: comparing trace finish, PASS!
================================================
```

表示针对某一benchmark的处理器行为仿真中该benchmark全部指令执行正确

- **如果仿真中出现指令执行错误的情况（判定结果错误输出）再查看仿真波形定位逻辑问题**
  - 波形查看命令：在本地仓库顶层目录中执行
    make HW_ACT=wav_chk HW_VAL=bhv vivado_prj
  - 如果在Vivado图形界面中对波形文件有所修改（添加/删除信号）并保存，需关闭Vivado图形界面重新执行仿真（HW_ACT=bhv_sim）后，再查看新波形

2.3.2 处理器执行结果仿真错误说明（1）

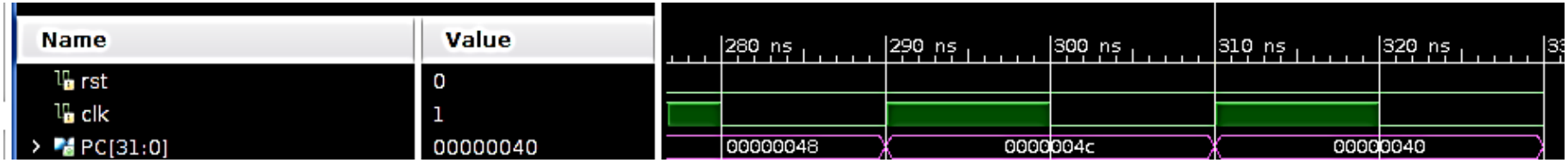
❑ PC寄存器值出错举例

```
================================================
Error: at                    330ns.
Yours:     PC = 0x00000040
Reference: PC = 0x0000003c
Please check assignment of PC at previous cycle.
================================================
```

第一行指示PC值比较错误发生的仿真时间

Yours：同学们代码中PC寄存器的输出值

Reference：当前周期应该得到的PC值



PC寄存器值在330ns的周期出现错误，请思考如下问题后查看波形，定位处理器逻辑错误：

- 错误值在哪一个周期被写入PC寄存器？
- 错误值在哪一个周期被计算出来？

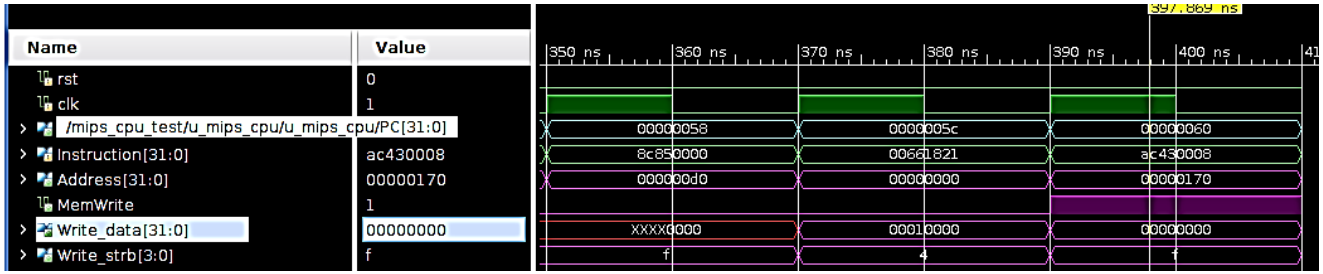
2.3.2 处理器执行结果仿真错误说明（2）

❑ 内存写操作出错举例

```
================================================
Error: at                    230ns.
Yours:     Address = 0x0000008c, Write_strb = 0xf, (Write_data & 0xffffffff) = 0x00000000
Reference: Address = 0x0000008c, Write_strb = 0xf, (Write_data & 0xffffffff) = 0x00000001
================================================
```

用于计算Write_data中实际要写入内存的有效位，此例中为全部32-bit

标明当前周期Write_data值有误



Write_data的值在410ns出错，这个值在哪一个周期被计算出来？

❑ 寄存器写操作出错举例

```
================================================
Error: at                    230ns.
Yours:     RF_waddr = 09, (RF_wdata & 0x0000ffff) = 0x00000000
Reference: RF_waddr = 09, (RF_wdata & 0x0000ffff) = 0x0000ffff
================================================
```

说明当前周期RF_wdata值有误

用于计算RF_wdata中实际要写入寄存器的有效位，此例中为低16-bit

2.3.2 处理器执行结果仿真错误说明（3）

- **内存读/写使能（MemRead/MemWrite）或寄存器写使能信号（RF_wen）错误**
  - 写内存（st类）指令执行时，MemRead拉高或RF_wen拉高
  - 读内存（ld类）指令执行时，MemWrite拉高或RF_wen拉低
  - 寄存器类指令执行时，MemRead或MemWrite拉高
  - 跳转指令（如bne指令）执行时，三个使能信号有一个或多个拉高（如下图提示信息）

```
================================================
Error: at                    530ns.
Yours:     MemWrite = 0, RF_wen = 1, MemRead = 0
Reference: MemWrite = 0, RF_wen = 0, MemRead = 0
================================================
```

- **如出现该类型错误，说明指令译码逻辑存在问题，应对照错误原因及时修正**

2.4 生成bitstream

- 在本地仓库中执行
  make HW_ACT=bit_gen vivado_prj
  - 自动完成综合、布局布线和bitstream生成（.bit后缀文件）
  - 生成的bitstream文件位于本地仓库顶层的hw_plat目录下

- 在本地仓库中执行make bit_bin
  - 在本地仓库顶层的hw_plat目录下生成二进制bitstream文件（.bit.bin后缀文件）
  - 准备用于上板测试运行

2.5 FPGA云环境测试运行

- 在虚拟机的本地仓库中执行

  make USER=<user_name> HW_VAL=<benchmark组名> cloud_run

  连接FPGA云远程板卡

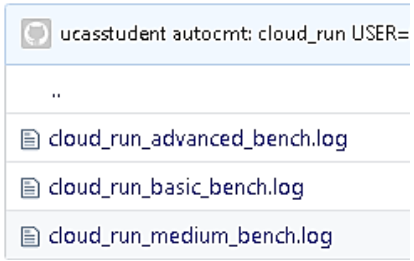
  - <benchmark组名>是basic/medium/advanced中的任意一个
    - 例如：make USER=zhangsan HW_VAL=medium cloud_run
  - 逐一运行指定组内的各个测试程序

运行结果说明

- **运行结果说明**
  - 自动统计当前运行的组（basic、medium或advanced）中通过测试的benchmark数量（如图倒数第二行黄线处）
  - “测试通过”表明程序执行完成后，理想内存各地址上的数据正确（打印Hit good trap）
    - 说明处理器对当前测试程序包含的指令全部执行正确
  - 当某一个benchmark执行错误时，会打印Hit bad trap
- **运行结果日志按组别存放在仓库的run/log子目录下**
  - 分别保留各组别benchmark的各次上板运行记录（最后一次运行结果保留在日志文件起始位置）

```
cod@cod-VirtualBox:~/ucas-cod/prj2-student$ make USER=changyisong HW_VAL="medium" cloud_run
Starting xl2tpd (via systemctl): xl2tpd.service.
Remote target: root@172.16.15.66
Warning: Permanently added '172.16.15.66' (ECDSA) to the list of known hosts.
Completed FPGA configuration
Evaluating medium benchmark suite...
Launching sum benchmark...
Hit good trap
Launching mov-c benchmark...
Hit good trap
Launching fib benchmark...
Hit bad trap
Launching add benchmark...
Hit bad trap
Launching if-else benchmark...
Hit good trap
Launching pascal benchmark...
Hit good trap
Launching quick-sort benchmark...
Hit good trap
Launching select-sort benchmark...
Hit good trap
Launching max benchmark...
Hit good trap
Launching min3 benchmark...
Hit good trap
Launching switch benchmark...
Hit good trap
Launching bubble-sort benchmark...
Hit good trap
pass 10 / 12
Stopping xl2tpd (via systemctl): xl2tpd.service.
```

prj2-zhang / run / log /



最终提交前，务必检查三个日志文件的最后一次运行结果中该组内的测试程序全部pass

仿真结果正确，FPGA云平台运行出错

- 如果出现这种问题，请仔细检查
  hardware/vivado_out/run_log/bit_gen.log

- 定位自己设计的模块产生的Warning/Critical Warning
  - 定位依据：自己设计模块的模块名、信号名、模块实例化名

2.6 多周期处理器选做实验说明

- 多周期处理器的行为仿真不提供指令结果比对功能
- 多周期处理器的上云测试与单周期处理器使用相同的benchmark，运行的命令也完全一致
- 多周期处理器的状态机状态采用one-hot编码
- 无需考虑处理器异常处理
- 实验课上使用的理想内存与理论课讲解的多周期处理器，使用的内存不同，要注意访存通路的信号

内容大纲

- 实验要点讲解
- 实验操作流程
- 其他事项

实验项目2的实验报告

- 请在实验项目2个人本地仓库中创建顶层目录doc（仅需执行一次）
  - mkdir –p doc
- 将实验报告的PDF版本放入doc目录，命名规则为“学号-prj2.pdf”，例如：
  - 2017K8009929000-prj2.pdf
  - PDF文件大小尽量控制在5MB以内
- 添加实验报告到本地仓库的方法
  - git add --all && git commit -m “doc: Update project report”
- 实验报告内容中不必详细描述实验过程, 但我们鼓励你在报告中描述如下内容:
  - 你遇到的问题和对这些问题的思考
  - 你的其它想法, 例如实验心得, 对提供帮助的同学的感谢等
- 多周期处理器扩展实验不要求单独撰写实验报告，可把相关内容一并写在“学号-prj2.pdf”文件中

充分利用远程仓库中的README

- 包含对实验项目的详细说明
  - 包括各benchmark的功能、实验用到的命令
  - 通过浏览器打卡个人远程仓库即可显示

README.md

Project #2 (prj2) in Experiments of Computer Organization and Design (COD) in UCAS

changyisong@ict.ac.cn

MIPS CPU Benchmarks

In this project, we provide three benchmark suites for simulation and FPGA on-board evaluation of your own single-cycle MIPS CPU core design. You can launch simulation or evaluation with a specified benchmark by explicitly setting parameters of benchmark suite name and serial number in MAKE command line.

Basic Benchmark Suite

Serial Number	Benchmark Name	Description
00	test	User's own instruction stream
01	memcpy	Simple memory copy function for 100 consecuitive 32-bit words

memcpy benchmark in this suite is used during the 1st phase of this project in which implementation of 5 basic MIPS CPU instructions is required. On the other hand, *test* benchmark is used to support custom instruction stream for each phase of this project, which provides a simple test environment for early debugging of your instruction implementations within a short simulation duration of 2us. Please note that ** *test* benchmark is not used for FPGA on-board evaluation**.

Medium Benchmark Suite

Serial Number	Benchmark Name	Description
01	sum	Calculate the summary of 1 to 100
02	mov-c	Move data to an array

Q & A ?


中国科学院大学
University of Chinese Academy of Sciences
ICT
中科院计算所
INSTITUTE OF COMPUTING TECHNOLOGY,CAS

PRJ2课堂验收要求

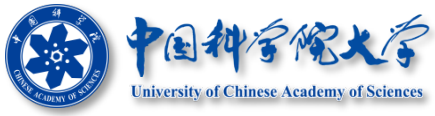
1. **实验态度端正，完成代码编写和上板测试，可得基准成绩40分**
   - 查看run/log目录下的三组benchmark运行结果
2. **其他检查要点 （满分50分）**
   - 组合逻辑必须用assign语句描述（+15分）
   - 时序逻辑PC的描述符合代码规范（包括复位信号使用、赋值条件互斥等，+15分）
   - 按指令格式和指令类型设计处理器译码逻辑（+10分）
   - 代码对应单周期处理器电路结构图（+10分）
3. **同学对RTL代码及实验内容的思考（满分10分，助教根据沟通情况打分）**
4. **多周期处理器选做实验不安排课堂验收，在最终提交后统一进行代码评审和得分统计**
5. **同学需根据助教建议修改代码，并在实验报告中进行说明。助教会根据课堂验收记录，检查最终提交代码修改情况**
6. **如果明显抄袭 0分记录（对代码完全说不清楚的，疑似抄袭）**

B0911007Y 2019-2020学年春季学期

计算机组成原理实验

实验项目2 单周期处理器设计
——主要问题讲解

2020年5月1日



实验项目进度调整

- 根据理论课进度情况，再次调整实验项目2单周期处理器和选做多周期处理器两个实验项目的截止时间
- 课堂验收截止：5月8日下课时（17:49:59）
- 最终提交截止时间（5月11日23:59:59）

  需提交完整RTL代码、FPGA云环境运行结果及实验报告
  - 执行命令：git push origin master
- 5月8日实验课上布置实验项目3

问题1：云平台连接不顺畅

- 尝试如下解决方法：
  - 在虚拟机的命令行中执行：

```
sudo bash -c "echo 'mtu 1410' >> /etc/ppp/peers/testvpn.l2tpd“
```

  - 输入命令时请注意引号必须是英文符号且不能遗漏
  - 如执行上述命令后访问云平台仍然不顺利，可把上面命令中
    的1410数值再改小（如1380）后重新执行
  - 注：本方法不能解决云平台中FPGA资源已被占满的情况，仅可提高虚拟机通过家中无线路由器连接云平台的稳定性

- **无论同学们之前是否可以正常访问云平台，都建议完成上述操作**

问题2：处理器的内存大小端

- 本次实验及后续实验全部采用小端内存顺序

- 实验过程中是否可以判断大小端？
  - 通过假设法来求证

  - 假设处理器使用大端内存顺序，在接收理想内存传来的Instruction后，处理器的Verilog HDL代码要如何处理，才能让自己的译码逻辑电路看到正确的指令码？

  - 如果我们在Verilog HDL代码中不做这样的处理，译码逻辑看到的指令码是什么？是否是正确的指令码？

问题3：内存读写地址对齐

- 本次实验及后续实验内存读写地址（PC和Address）全部采用4-byte对齐的地址

- 对于访问半字（lh/sh）或字节（lb/sb）指令，需要根据地址的最低2-bit，计算4-byte内部的偏移量，进而对Read_Data进行选择或设置正确的Write_strb信号

问题4：指令手册阅读

- 在MIPS处理器手册中，有些指令的操作（Operation）较为复杂，容易对指令理解造成困扰
  - **我们的实验只需实现指令的最简单操作语义**，有关虚拟地址-物理地址转换（vAddr-pAddr）、特权（Privilege）、异常（Exception）、系统配置（Config）、指令模式（ISAMode）等内容无需考虑

JR指令

Operation:

```
I:  temp ← GPR[rs]
I+1:if Config1CA = 0 then
        PC ← temp
    else
        PC ← tempGPRLEN-1..1 || 0
        ISAMode ← temp0
    endif
```

无需考虑

LH指令

Operation:

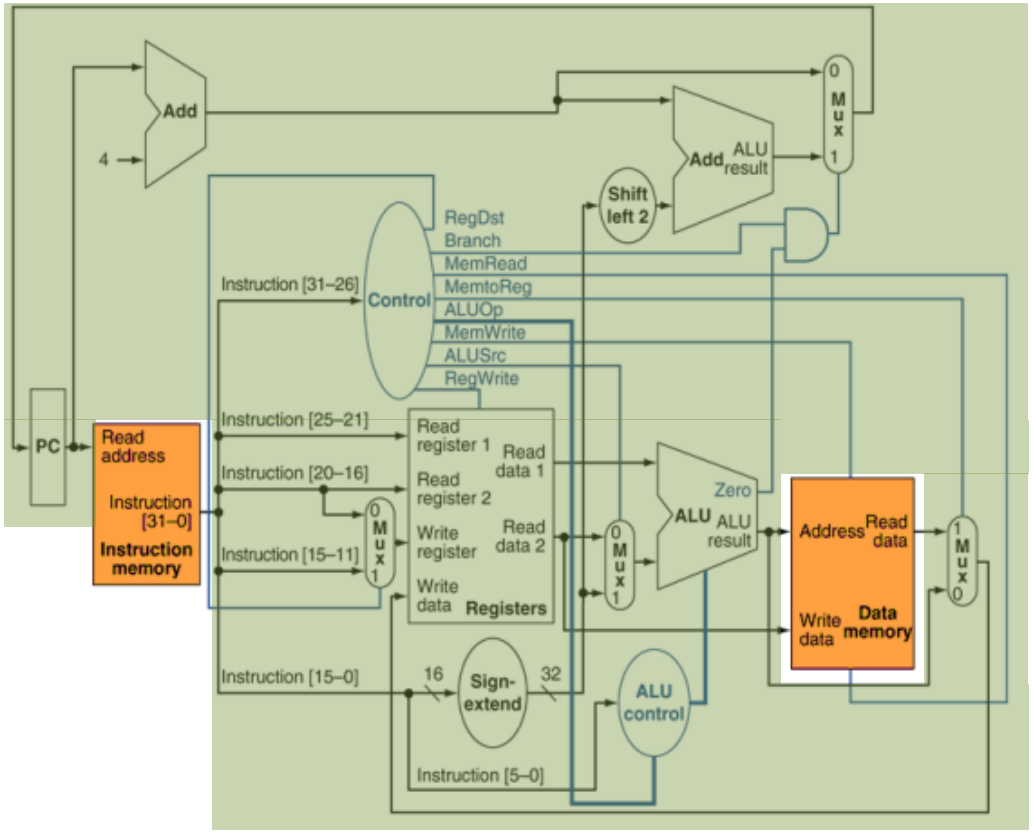
```
vAddr ← sign_extend(offset) + GPR[base]
if vAddr0 ≠ 0 then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation (vAddr, DATA, LOAD)
pAddr ← pAddrPSIZE-1..2 || (pAddr1..0 xor (ReverseEndian || 0))
memword ← LoadMemory (CCA, HALFWORD, pAddr, vAddr, DATA)
byte   ← vAddr1..0 xor (BigEndianCPU || 0)
GPR[rt] ← sign_extend(memword15+8*byte..8*byte)
```

无需考虑

把vaddr调整为4-byte对齐地址

问题5：如何设计指令译码逻辑（1）

- 什么是译码？
  - 32-bit二进制指令串 -> 完成指令功能所需的各种控制信号
- 什么是控制信号？
  - 处理器中主要部件的各个输入端口的数据选择信号



问题5：如何设计指令译码逻辑（2）

❑处理器各阶段需要的控制信号

- **译码（ID）：**寄存器堆读地址选择
- **执行（EX）：**ALU两个操作数选择、ALUOp、地址计算部件的选择信号
- **内存访问（MEM）：**内存读/写使能、读写地址选择、写数据（Write_Data + Write_strb）选择
- **写回（WB）：**寄存器堆写地址与写数据选择、写使能wen
- **PC寄存器更新：**目标PC值的选择

问题5：如何设计指令译码逻辑（3）

- 如何生成控制信号？
  - ☑通过指令格式和指令类型，找到不同指令间的通用性
    - ➢根据指令手册，生成自己的译码表
    - ➢根据译码表归纳出指令码与控制信号之间的关联
  - ~~☒逐个实现44条指令~~
    - ➢没有充分使用不同指令的共性特征

指令译码表

指令类型	控制信号														
	寄存器堆读		ALU			内存访问					跳转		寄存器堆写		
	raddr 1	raddr 2	A	B	ALU Op	Address	MemRead	MemWrite	Write_Data	Write_strb	跳转地址	地址更新条件	wen	waddr	wdata
R-Type															
REGIMM															
J-Type															
I-Type 分支指令															
I-Type 运算指令															
I-Type 访存指令															

- 根据每类指令的共性特征，在每一个表项中填写相应的Instruction字段和功能部件的输出信号组成的逻辑表达式
- 对表格中的地址和数据，除列出选择信号外，也可列出相应地址或数据的来源

R-Type指令

- **opcode[5:0] == 6'b000000**
  - func[5] == 1：运算指令（ALU控制信号）
  - func[5:3] == 3'b000：移位指令（移位器控制信号）
  - {func[5:3], func[1]} == 4'b0010：跳转指令（PC变化信号）
  - {func[5:3], func[1]} == 4'b0011：mov指令
- **rs、rt、rd可直接作为寄存器读写地址**
- **写回数据根据不同功能进行选择**
  - mov指令需控制wen
  - jr指令不产生wen

指令	opcode（6-bit）	rs（5-bit）	rt（5-bit）	rd（5-bit）	shamt（5-bit）	func（6-bit）
运算指令						
addu	000000	rs	rt	rd	00000	100001
subu						100011
and						100100
or						100101
xor						100110
nor						100111
slt						101010
sltu						101011
移位指令						
sll	000000	00000	rt	rd	sa	000000
sra						000011
srl						000010
sllv		rs			00000	000100
srav						000111
srlv						000110
跳转指令						
jr	000000	rs	00000	00000	00000	001000
jalr			00000	rd	00000	001001
mov指令						
movz	000000	rs	rt	rd	00000	001010
movn						001011

REGIMM类型指令

指令	opcode（6-bit）	rs（5-bit）	rt（5-bit）	imm (16-bit)
blez	000001	rs	00000	offset
bgez			00001	

- opcode[5:0] == 6'b000001
- rs作为寄存器读地址、另一个寄存器读地址可设置为0
- 执行阶段计算PC变化值及跳转条件
  - 产生ALU的操作数（rdata1、rdata2）及ALUOp控制信号（sub），计算跳转条件
- PC值更新
  - 根据ALU产生的标志位（Overflow）及rt选择PC值

J-Type指令

指令	opcode（6-bit）	Instr_index（26-bit）
j	000010	instr_index
jal	000011	

- opcode[5:1] == 5'b00001
- 执行阶段计算PC变化值
- jal指令需要产生wen信号，写r31

I-Type分支指令

指令	opcode（6-bit）	rs（5-bit）	rt（5-bit）	imm (16-bit)
beq	000100	rs	rt	imm
bne	000101			
blez	000110		00000	

- opcode[5:2] == 4'b0001
- rs、rt作为寄存器读地址
- 执行阶段计算PC变化值及跳转条件
  - 产生ALU的操作数（rdata1、rdata2）及ALUOp控制信号（sub），计算跳转条件
- PC值更新
  - 根据ALU产生的标志位（Zero、Overflow）及opcode[1:0]选择PC值

I-Type计算指令

指令	opcode（6-bit）	rs（5-bit）	rt（5-bit）	imm (16-bit)
addiu	001001	rs	rt	imm
lui	001111			
andi	001100			
ori	001101			
xori	001110			
slti	001010			
sltiu	001011			

- opcode[5:3] == 3'b001
- rs作为寄存器读地址，rt作为寄存器写地址（wen无条件有效）
- 执行阶段产生ALU操作数（rdata1、立即数），并根据opcode[3:0]产生ALUOp控制信号
  - 需要和R-Type计算指令一起来考虑ALUOp的编码方法

I-Type内存读指令

指令	opcode（6-bit）	base（5-bit）	rt（5-bit）	imm (16-bit)
lb	100000	base	rt	offset
lh	100001			
lw	100011			
lbu	100100			
lhu	100101			
lwl	100010			
lwr	100110			

- MemRead = opcode[5] & (~opcode[3])
- rt作为寄存器读写地址
- 执行阶段产生ALU操作数及ALUOp（add）控制信号来计算内存地址
- 根据opcode[2:0]和ALU输出的地址值计算字节掩码mask，用mask选择要写入rt寄存器的字节数据

I-Type内存写指令

指令	opcode（6-bit）	base（5-bit）	rt（5-bit）	imm (16-bit)
sb	101000	base	rt	offset
sh	101001			
sw	101011			
swl	101010			
swr	101110			

- MemWrite = opcode[5] & opcode[3]
- rt作为寄存器读地址
- 执行阶段产生ALU操作数及ALUOp（add）控制信号来计算内存地址
- 根据opcode[2:0]和ALU输出的地址值计算Write_strb

其他问题说明

- 寄存器堆的r1 – r31不要在rst信号有效时清零
  - 云平台基准测试假设上述寄存器不能复位清零