B0911007Y-01/02　2020-2021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

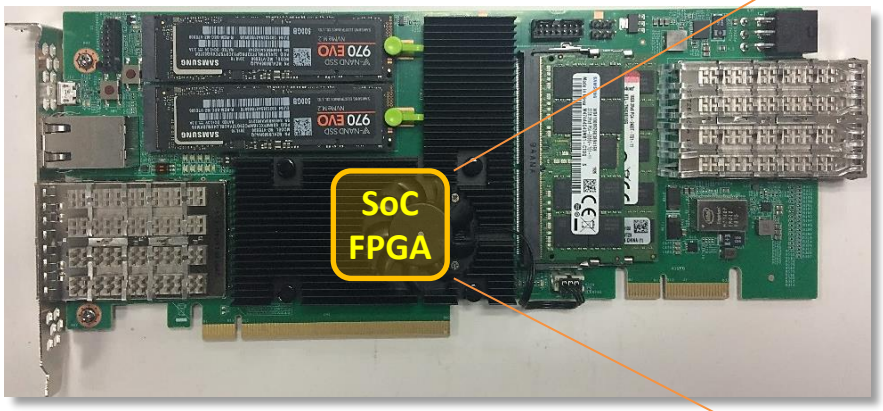
2021年4月23日

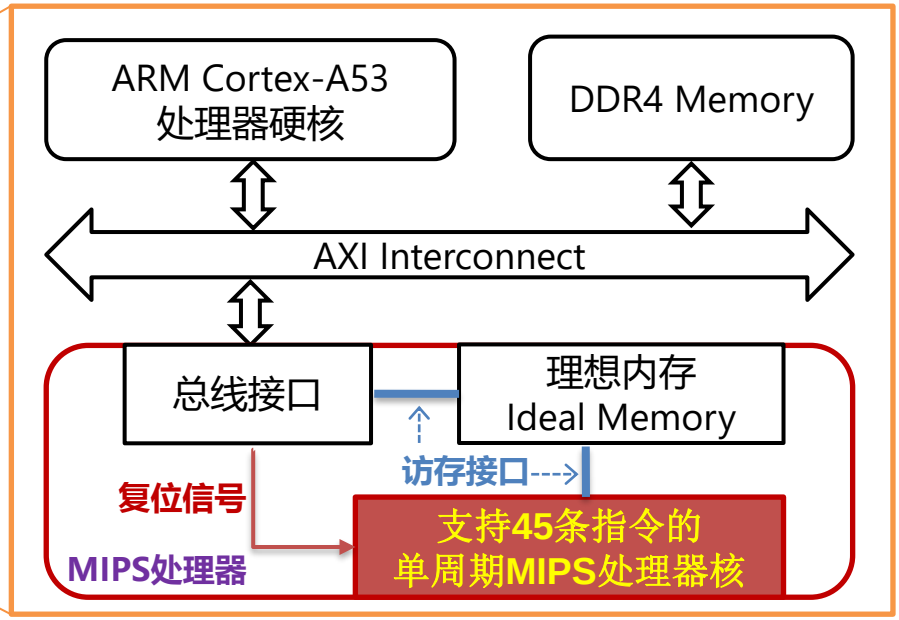
0.1 实验目的

❑ 设计可运行MIPS指令子集的简易单周期功能型处理器

- 实现基于理想内存（Ideal Memory）的单周期MIPS处理器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测试方法

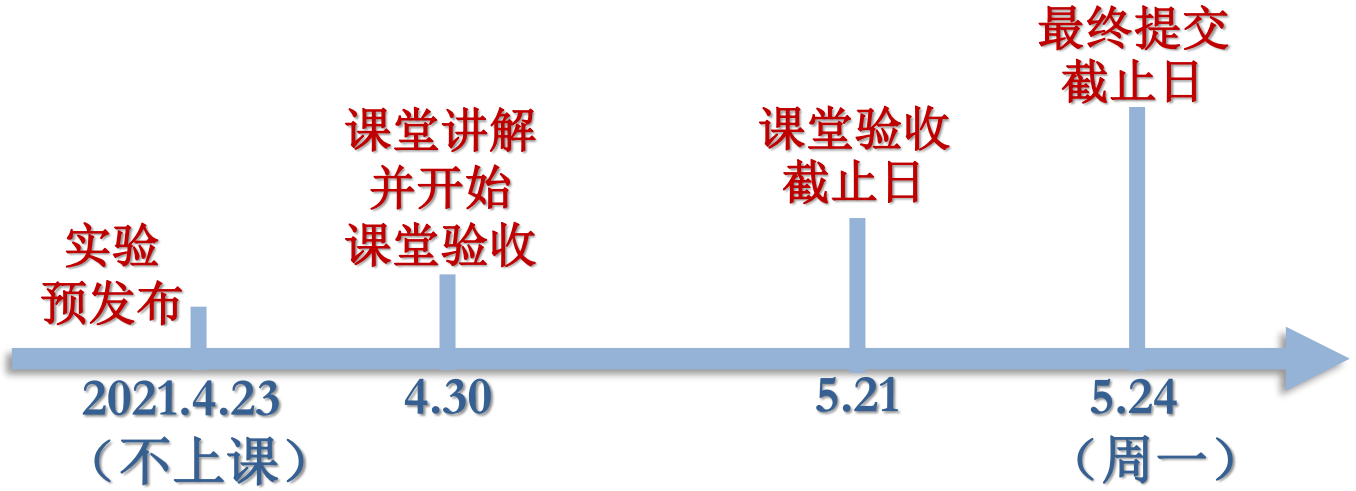
0.2 实验环境及工程框架





- 实现可支持45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（reg_file.v）和算术逻辑单元（alu.v）等基本组件，并按需添加新组件
- 使用30个benchmark，测试单周期MIPS处理器核功能（benchmark分为basic、medium和advanced三组）

0.3 实验项目进度安排



- 本次实验不设阶段提交
- 自4月30日上课时间起开始进行课堂验收
- 课堂验收截止：5月21日下课时（18:59:59）
- 最终提交截止时间前（5月24日周一23:59:59）需提交完整RTL代码及实验报告

0.4 课堂验收要求

❑ 课堂验收要求

- 检查单周期MIPS处理器最终实现，在FPGA上执行benchmark的情况

- 检查单周期MIPS处理器的RTL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）

实验项目接收

❑ 实验项目个人远程仓库URL地址

https://gitlab.agileserve.org.cn:8001/<GitLab用户名>/prj2

- 例如，学生张三（ zhangsan19@mails.ucas.ac.cn ）的个人远程仓库地址：https://gitlab.agileserve.org.cn:8001/zhangsan19/prj2
- 如个人远程仓库在浏览器中无法打开，请联系助教老师

❑ 创建实验项目的个人本地仓库

- 在虚拟机shell终端中执行git clone命令

cd && git clone ssh://git@gitlab.agileserve.org.cn:8082/<GitLab用户名>/prj2.git

cd ~/prj2 && git submodule update --init flow

<GitLab用户名>替换为UCAS邮箱前缀

（例如：ssh://git@gitlab.agileserve.org.cn:8082/zhangsan19/prj0.git）

- **注意：虚拟机此时需要连通互联网**

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

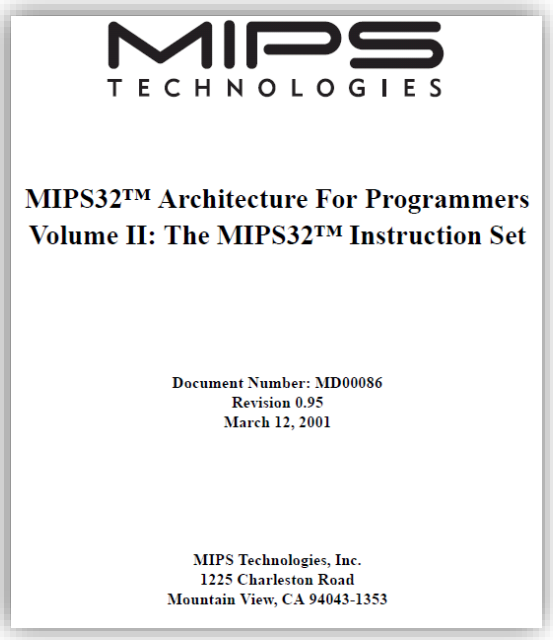
1.1 需支持的45条MIPS指令

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（10条）	bne, beq, bgez, bgtz, 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

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 Config1_CA = 0 then
        PC ← temp
     else
        PC ← temp_GPRLEN-1..1 || 0
        ISAMode ← temp_0
     endif
```

无需考虑

LH指令

Operation:

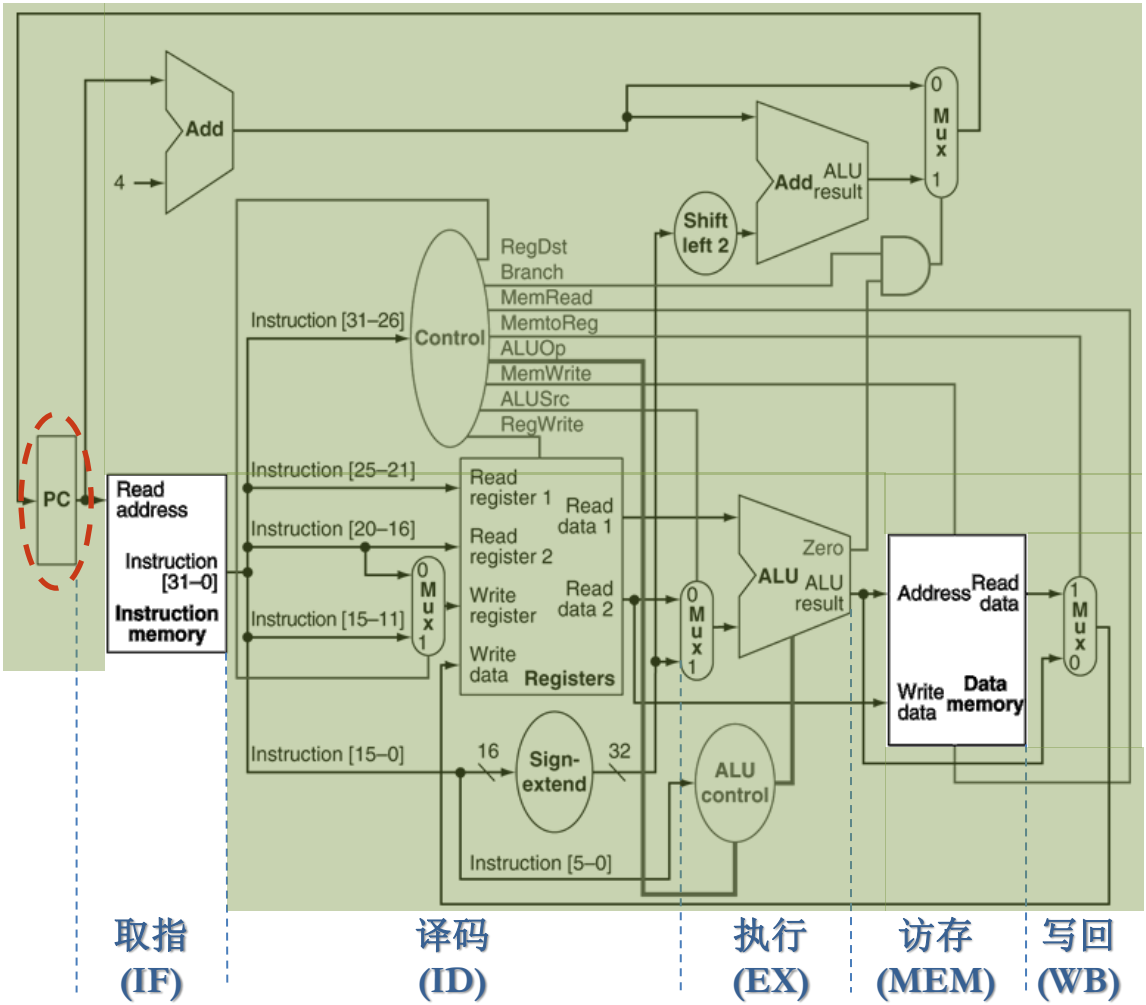
```
vAddr ← sign_extend(offset) + GPR[base]
if vAddr_0 ≠ 0 then
   SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation (vAddr, DATA, LOAD)
pAddr ← pAddr_PSIZE-1..2 || (pAddr_1..0 xor (ReverseEndian || 0))
memword ← LoadMemory (CCA, HALFWORD, pAddr, vAddr, DATA)
byte   ← vAddr_1..0 xor (BigEndianCPU || 0)
GPR[rt] ← sign_extend(memword_15+8*byte..8*byte)
```

无需考虑

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32/16/8-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（详见1.3.4小节）
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

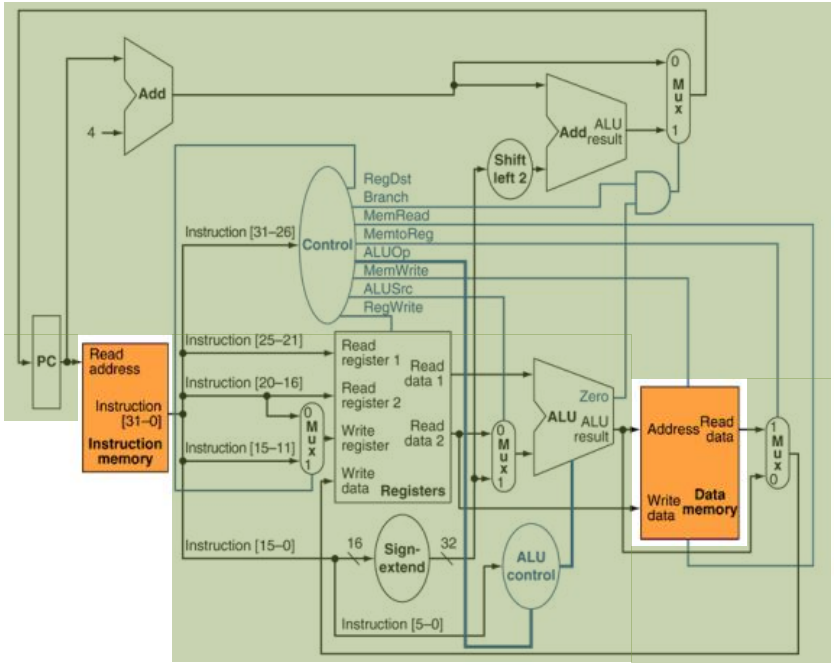


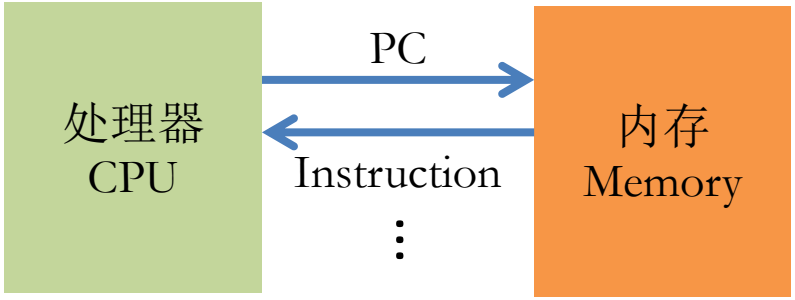
- 本次实验项目需实现图中绿色部分（即指令内存Instruction memory和数据内存Data memory之外的所有逻辑电路）
- ALU及通用寄存器堆（图中Registers）已在实验项目1中实现，可直接修改复用
  - 某些指令需要在ALU中添加新的操作，需修改ALU代码
- PC是一个寄存器（时序电路）
  - 复位有效时，PC寄存器的值为32'd0
  - 跳转目标地址使用一个单独加法器计算（右上角）
- 结构图与表格输入输出端口信号名可能有少量不符，原则上不影响理解
  - 若仍然有问题请联系助教老师

1.4.1 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





1.4.2 理想内存（Ideal Memory）

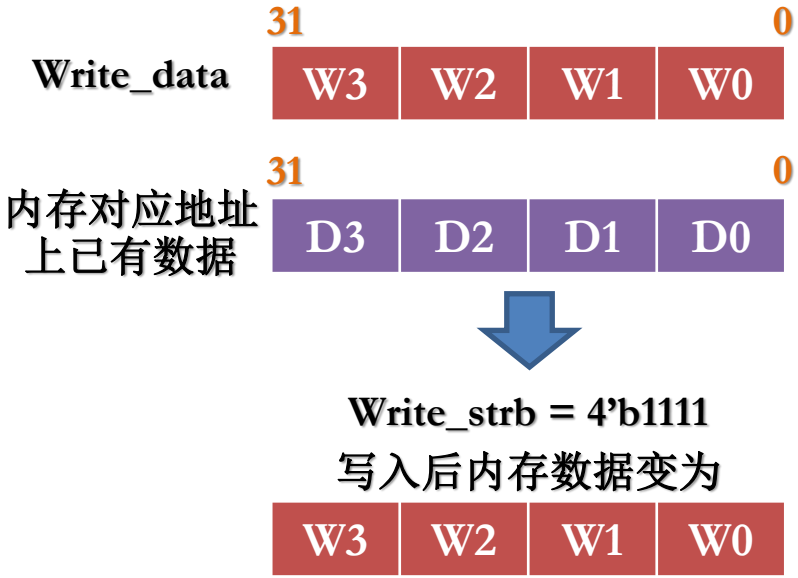
- **假设内存访问与寄存器访问具有相同的延时**
  - 确保指令单周期处理
    - 读/写操作全部在一个时钟周期内完成
    - 同步写，异步读
- **通过2读1写端口支持指令和数据访问**
  - 读端口1用于指令读，读端口2用于数据读
  - 写端口用于数据写
- **理想内存的RTL代码助教已写好**
  - 代码位置：flow/hardware/sources/hdl/ideal_mem.v
  - 已在flow/hardware/sources/hdl/mips_cpu_top.v中对理想内存进行实例化，**无需同学们进行任何操作**

1.4.3 内存读写地址对齐

- 本次实验项目及后续实验项目中，
  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
  - Load类指令：对Read_Data进行选择
  - Store类指令：设置正确的Write_strb信号

1.4.4 内存Write_strb信号使用



1.5 指令译码部分的逻辑设计

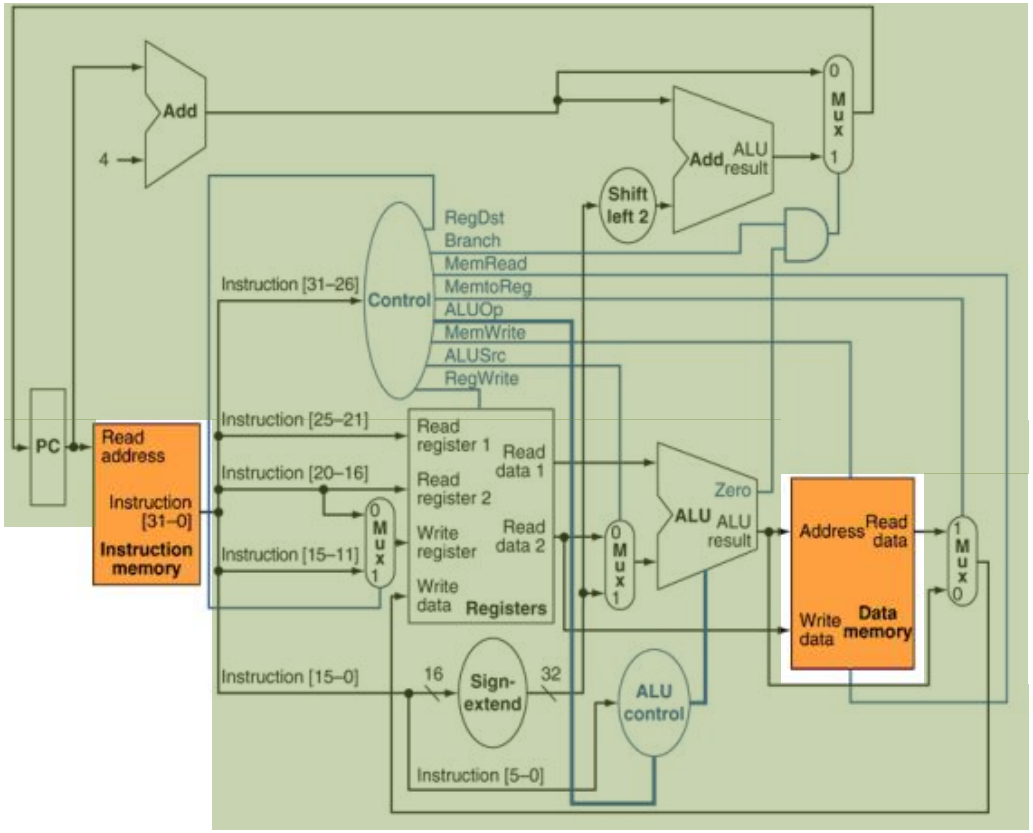
- 译码逻辑要充分利用指令格式和指令分类

不要像下图这样，每条指令单独译码，产生控制信号

```
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)));
```

1.5.1 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控制+数据信号
- 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及地址
- 什么是数据信号？
  - 指令中的立即数



1.5.1 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

1.5.1 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，找到不同指令间的共性特征
    - ➢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45条指令~~
    - ➢没有充分使用不同指令的共性特征

1.5.2 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	ALUop	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
- 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'b1：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)
bltz	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
- 执行阶段计算跳转目标地址及跳转条件
  - 跳转条件计算：产生ALU的操作数（rdata1、rdata2）及ALUop控制信号（SLT）
  - 目标地址计算：立即数offset送入地址加法器运算
- PC值更新
  - 根据ALU产生的标志位（Zero）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

<table>
<tr><th>指令</th><th>opcode
（6-bit）</th><th>rs
（5-bit）</th><th>rt
（5-bit）</th><th>imm (16-bit)</th></tr>
<tr><td>beq</td><td>0001 00</td><td rowspan="4">rs</td><td rowspan="2">rt</td><td rowspan="4">offset</td></tr>
<tr><td>bne</td><td>0001 01</td></tr>
<tr><td>blez</td><td>0001 10</td><td rowspan="2">00000</td></tr>
<tr><td>bgtz</td><td>0001 11</td></tr>
</table>

- opcode[5:2] == 4'b0001
- rs、rt作为寄存器读地址
- 执行阶段计算跳转目标地址及跳转条件
  - 跳转条件计算：产生ALU的操作数（rdata1、rdata2）及ALUop控制信号（SUB：beq/bne；SLT：blez/bgtz）
  - 目标地址计算：立即数offset送入地址加法器运算
- PC值更新
  - 根据ALU产生的标志位（Zero）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（1.5.3小节）

I-Type内存读指令

指令	opcode（6-bit）	base（5-bit）	rt（5-bit）	imm (16-bit)
lb	100 000	base	rt	offset
lh	100 001			
lw	100 011			
lbu	100 100			
lhu	100 101			
lwl	100 010			
lwr	100 110			

- 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	101 000	base	rt	offset
sh	101 001			
sw	101 011			
swl	101 010			
swr	101 110			

- 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

1.5.3 ALUop编码

<table>
<tr><th>R-Type
指令</th><th>func
（6-bit）</th><th>ALUop
（3-bit）</th><th>ALUop编码</th><th>opcode
（6-bit）</th><th>I-Type
指令</th><th>ALUop编码</th></tr>
<tr><td>add</td><td>10 00 00</td><td rowspan="2">010</td><td rowspan="3">ADD/SUB：func[3:2] == 2’b00

ALUop = {func[1], 2’b10}</td><td>001 0 00</td><td>addi</td><td rowspan="2">ADD：opcode[2:1] == 2’b00
ALUop = {opcode[1], 2’b10}</td></tr>
<tr><td>addu</td><td>10 00 01</td><td>001 0 01</td><td>addiu</td></tr>
<tr><td>sub</td><td>10 00 10</td><td rowspan="2">110</td><td></td><td></td><td></td></tr>
<tr><td>subu</td><td>10 00 11</td><td></td><td></td><td></td></tr>
<tr><td>and</td><td>10 01 00</td><td>000</td><td rowspan="3">逻辑运算：func[3:2] == 2’b01

ALUop = {func[1], 1’b0, func[0]}</td><td>001 1 00</td><td>andi</td><td rowspan="3">逻辑运算：opcode[2] == 1’b1

ALUop = {opcode[1], 1’b0, opcode[0]}</td></tr>
<tr><td>or</td><td>10 01 01</td><td>001</td><td>001 1 01</td><td>ori</td></tr>
<tr><td>xor</td><td>10 01 10</td><td>100</td><td>001 1 10</td><td>xori</td></tr>
<tr><td>nor</td><td>10 01 11</td><td>101</td><td>001 1 11</td><td>lui</td><td>非运算类指令，需单独处理</td></tr>
<tr><td>slt</td><td>10 10 10</td><td>111</td><td rowspan="2">比较运算：func[3:2] == 2’b10
ALUop = {~func[0], 2’b11}</td><td>001 0 10</td><td>slti</td><td rowspan="2">比较运算：opcode[2:1] == 2’b01
ALUop = {~opcode[0], 2’b11}</td></tr>
<tr><td>sltu</td><td>10 10 11</td><td>011</td><td>001 0 11</td><td>sltiu</td></tr>
</table>

- 新增3个ALUop，分别对应XOR、NOR和SLTU操作（表格中橙色字）

1.5.4 移位运算部件与op编码

- 可设计一个单独的移位运算部件模块，处理所有移位指令

R-Type指令	func（6-bit）	Shiftop（2-bit）	Shiftop编码
sll	000 0 00	左移 00	Shiftop = func[1:0] func[2]用于选择移位长度来自立即数还是寄存器
sllv	000 1 00		
sra	000 0 11	算数右移 11	
srav	000 1 11		
srl	000 0 10	逻辑右移 10	
srlv	000 1 10		

1.6 转移指令的处理

- 实现单周期跳转指令（如beq）时，可以把指令手册中当前周期（左图I：）和下一周期（左图I+1：）的操作都合并到当前周期（I：）
- 单周期处理器实验中，测试程序在编译时通过编译选项，在跳转类指令后固定放置一条nop空指令（不会被执行的指令）
  - **跳转条件成立：**下一周期会直接执行跳转目标地址上的指令（如左图中的<main+0xcc>）
  - **跳转条件不成立：**下一周期直接执行nop后的一条指令

Format: `BEQ rs, rt, offset`

Purpose:

To compare GPRs then do a PC-relative conditional branch

Description: `if rs = rt then branch`

Operation:

```
I:     target_offset ← sign_extend(offset || 0^2)
       condition ← (GPR[rs] = GPR[rt])
I+1:   if condition then
          PC ← PC + target_offset
       endif
```

一个周期内完成

```
148: 24820278  addiu v0,a0,632
14c: 00621021  addu  v0,v1,v0
150: 8c420000  lw  v0,0(v0)
154: 10a20006  beq a1,v0,170 <main+0xcc>
158: 00000000  nop
```

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

1.7 benchmark简介

Basic组	
序列号	名称
01	memcpy

阶段I

Medium组	
序列号	名称
02	sum
03	mov-c
04	fib
05	add
06	if-else
07	pascal
08	quick-sort
09	select-sort
10	max
11	min
12	switch
13	bubble-sort

阶段II

Advanced组	
序列号	名称
14	shuixianhua
15	sub-longlong
16	bit
17	recursion
18	fact
19	add-longlong
20	shift
21	wanshu
22	goldbach
23	leap-year
24	prime
25	mul-longlong
26	load-store
27	to-lower-case
28	movsx
29	matrix-mul
30	unalign

阶段III

- 位于本地仓库flow/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

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，仿真时自动加载到理想内存中		

反汇编指令序列文件（*.S）格式举例

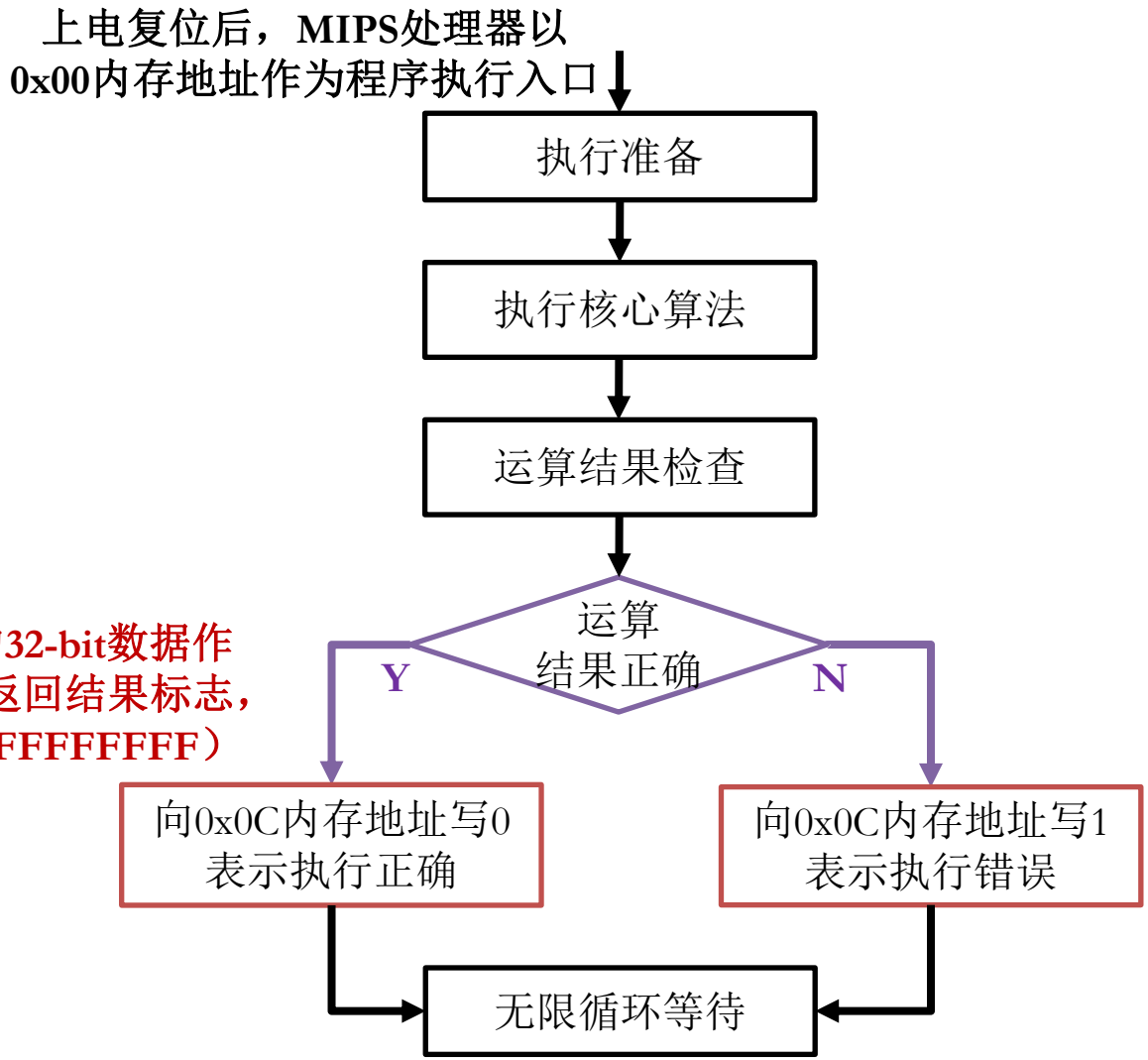
注意：指令助记符可能和指令手册有出入，此时请以指令码为准确定指令功能

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

benchmark可执行程序基本执行流程

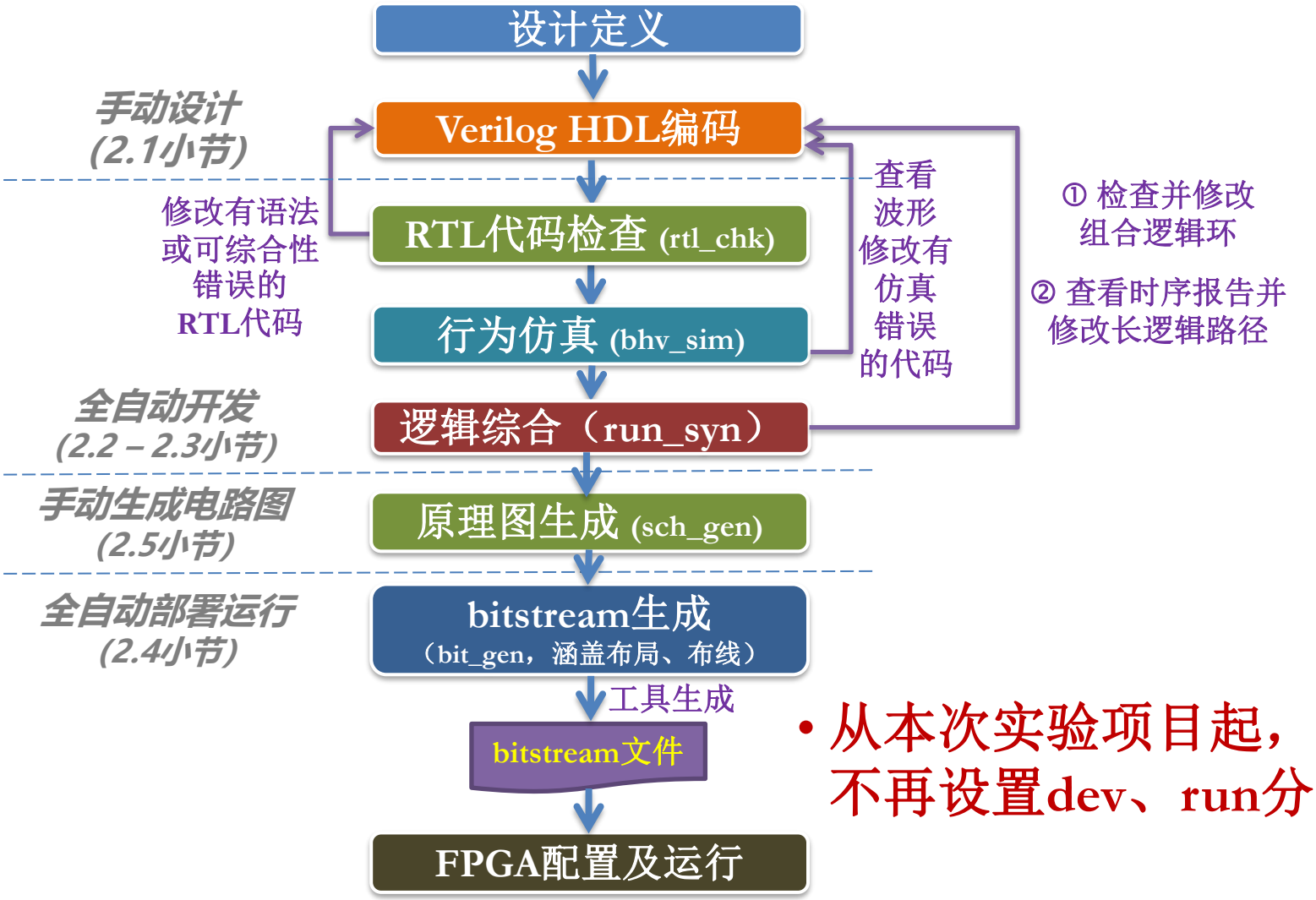


0x0C地址开始的32-bit数据作为benchmark的返回结果标志，初始值为-1（0xFFFFFFFF）

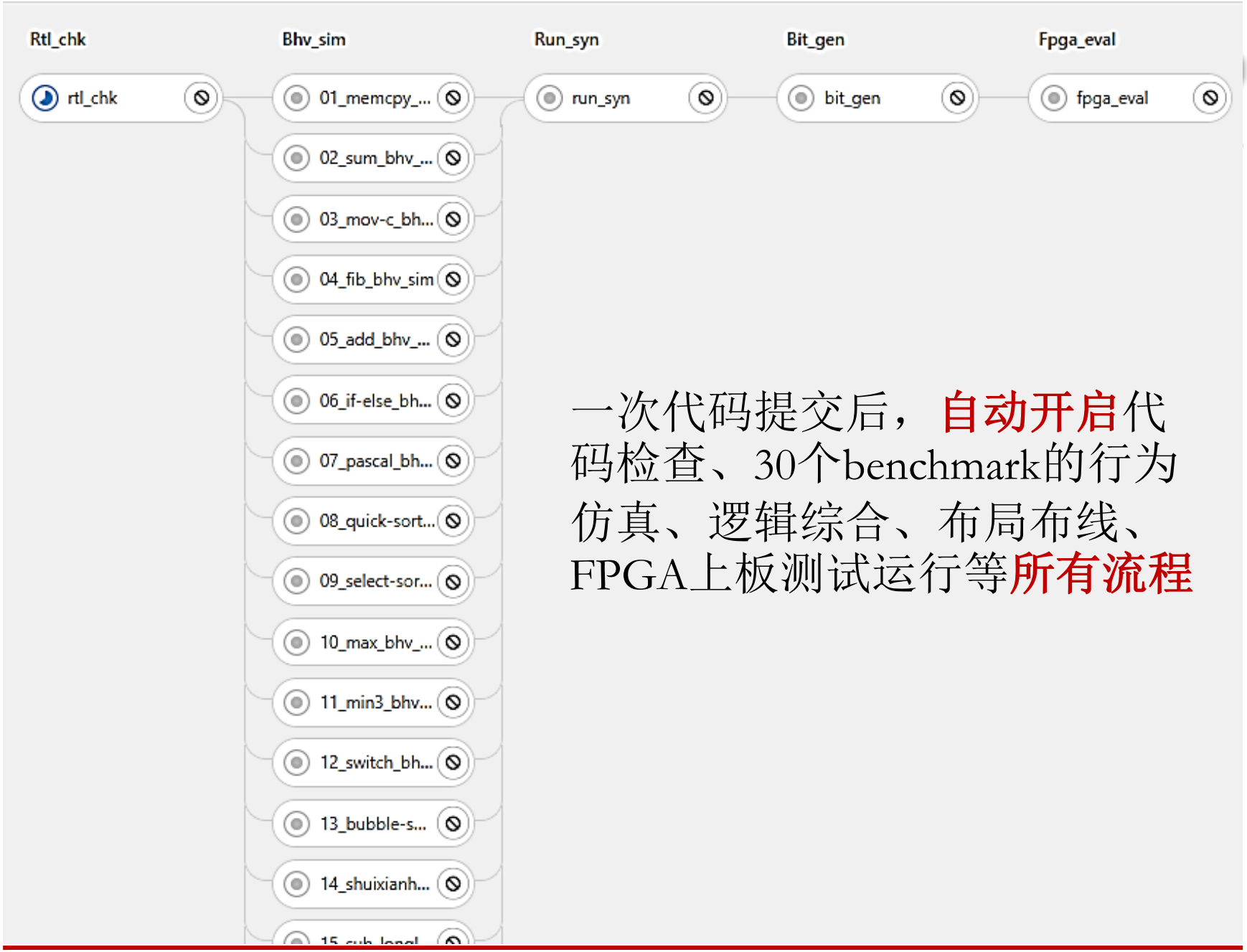
内容大纲

- 实验要点讲解
- 实验操作流程
- 其他事项

实验流程



- 从本次实验项目起，不再设置dev、run分支



一次代码提交后，**自动开启**代码检查、30个benchmark的行为仿真、逻辑综合、布局布线、FPGA上板测试运行等**所有流程**

2.1.1 设计输入 — 要编写的RTL代码

- **MIPS处理器相关逻辑代码放在~/prj2目录**

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
shifter.v	新增加的移位运算单元模块

- **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**
  - alu.v可参考PPT第1.5.3小节进行修改
- **需实现移位运算部件模块shifter.v，并在mips_cpu.v中进行模块实例化**
  - 实现方法可参考PPT第1.5.4小节

```
`define DATA_WIDTH 32

module shifter (
        input [`DATA_WIDTH - 1:0] A,
        input [`DATA_WIDTH - 1:0] B,
        input [1:0] Shiftop,
        output [`DATA_WIDTH - 1:0] Result
);

        // TODO: Please add your logic code here

endmodule
```

信号名	I/O	说明
A[31:0]	Input	需要移位的数据
B[31:0]	Input	移位长度
Shiftop[1:0]	Input	移位器操作类型
Result[31:0]	Output	移位器输出结果

mips_cpu.v的输入输出端口

```
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

endmodule
```

PC是一个时序电路

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

2.1.3 设计输入 — 代码提交

- 每次修改代码后，需要手动把修改提交（commit）到本地仓库

```
cd  ~/prj2
git  add  <拷贝或修改的.v文件名>
git  commit  -m  “commit message”
```

  - **commit message需要按具体情况进行替换**（每个commit message**务必**充分描述本次代码修改的目的、内容、预期达到的效果等信息）

2.2 触发全自动开发运行流程

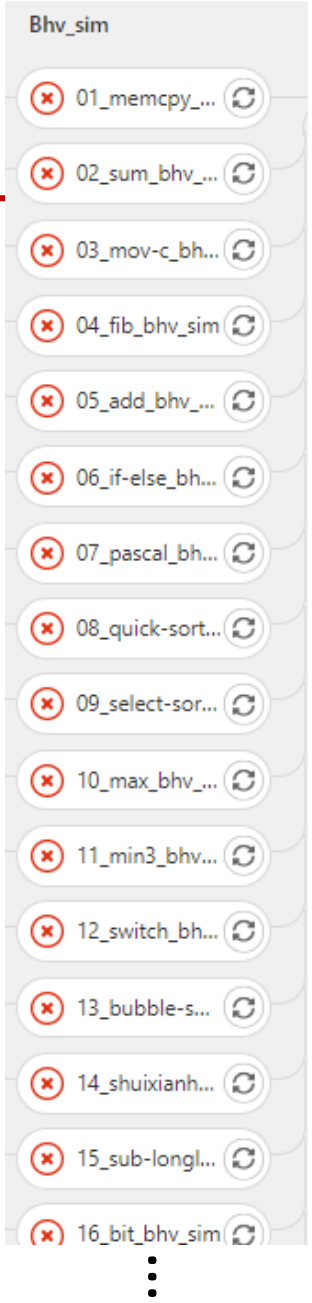
- 在完成一个阶段的代码修改后（包含一次或多次代码提交Commits），可将代码推送到云平台，触发自动开发运行流程

  cd ~/prj2 && git push origin master

- 在虚拟机Web浏览器中打开GitLab上个人远程仓库查看进展
  - https://gitlab.agileserve.org.cn:8001/<GitLab用户名>/prj2
  - 必做：代码语法错误的查看方法与之前实验项目一致，不再赘述
  - 必做：行为仿真（bhv_sim）出错时的操作**详见2.3.1小节**
  - 必做：检查逻辑综合（run_syn）报告，确认设计没有组合逻辑环（combinational loop）**（2.3.2小节）**
  - 选做：检查逻辑综合（run_syn）产生的时序报告，分析并修改代码中的组合逻辑长路径**（5月7日详细介绍）**
  - 必做：查看FPGA部署运行结果**（2.4小节）**

2.3.1 行为仿真

- 云平台自动开启30个benchmark的行为仿真任务（如右图）



- 可按照从上往下的顺序，找到第一个错误的仿真任务开始调试
  - 每个行为仿真任务的命名方式为<序号_benchmark名>_bhv_sim
  - 所有benchmark的序号和名称可查看PPT的1.7小节

2.3.1.1 查看某个仿真任务的错误信息

- **仿真testbench会在每个时钟上升沿判定指令执行的正确性**
  - 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

- **在信息框中找到两行“================ ================”之间的内容，就是处理器在运行当前的benchmark时出错的指令结果及比对情况（Reference是金标准处理器输出的正确结果）**

2.3.1.2 查看某个仿真任务的错误波形

cd ~/prj2/flow && make HW_ACT=wav_chk HW_VAL=<序号_benchmark名> CI_RUN=y vivado_prj && cd ~/prj2

- HW_VAL举例：01_memcpy，09_select-sort，17_recursion …
- 需要保证虚拟机可以上网
- 可按需在波形中添加要观察的信号，添加信号后不要忘记保存

- 如在波形中添加了新的信号（未添加新信号时可忽略），需要在关闭Vivado后，执行如下操作重新开始仿真

cd ~/prj2

git add waveform/behav.wcfg

git commit -m "behav.wcfg: Add xxx signal and remove xxx signal in waveform"

（其中xxx需替换为自己添加/删除的信号名）

git push origin master

2.3.1.3 仿真调试说明（1）

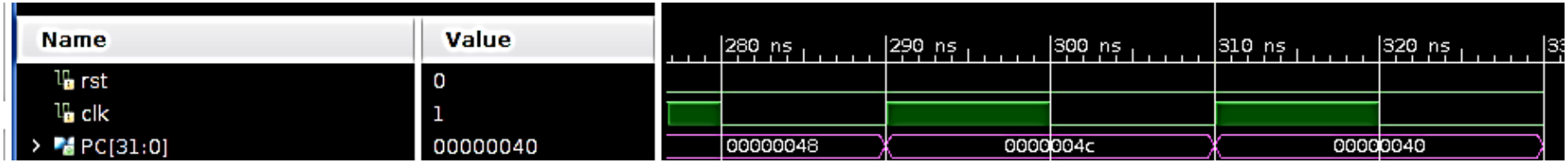
❑ PC寄存器值出错举例

```
================================================
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？

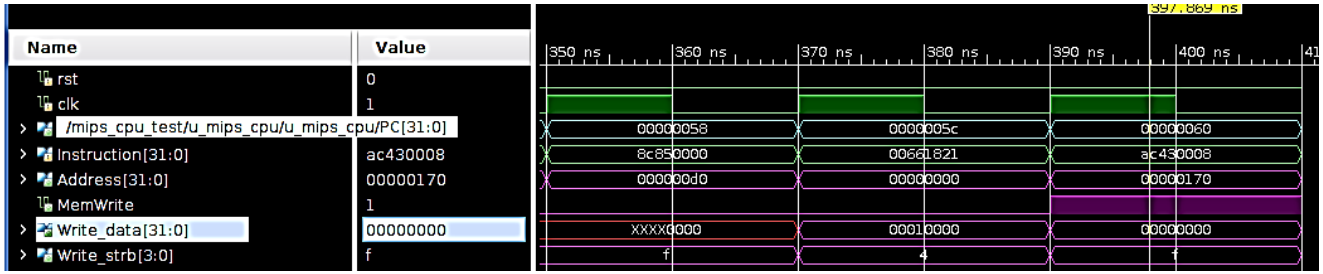
2.3.1.3 仿真调试说明（2）

❑ 内存写操作出错举例

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

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

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

2.3.1.3 仿真调试说明（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或RF_wen拉高
  - 读内存（ld类）指令执行时，MemWrite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**

2.3.2 检查设计中是否包含组合逻辑环

- 当设计中存在组合逻辑环时，run_syn不会报错，也没有警告
  - 需要手工检查相应的报告内容
- 在GitLab网站上打开run_syn的报告内容，搜索“9. checking loops”

```
9. checking loops
-----------------
 There are 2 combinational loops in the design. (HIGH)
Result[7]_INST_0_i_1/CO[7]
Result[15]_INST_0_i_1/CI
Result[15]_INST_0_i_1/CO[7]
Result[23]_INST_0_i_1/CI
Result[23]_INST_0_i_1/CO[7]
Overflow_INST_0_i_1/CI
Overflow_INST_0_i_1/O[7]
Result[7]_INST_0_i_10/I4
Result[7]_INST_0_i_10/O
Result[7]_INST_0_i_1/S[0]
Result[7]_INST_0_i_1/CO[7]
CarryOut_INST_0/O
CarryOut_INST_0/I1
CarryOut_INST_0/O
```

第一条组合逻辑环的完整路径

第二条组合逻辑环的完整路径

☒ 存在组合逻辑环（举例）

路径中的基本单元名都是Vivado产生的
并不能和代码中定义的变量名完全一致
但可看出与Verilog源代码中变量名的关联

```
9. checking loops
-----------------
 There are 0 combinational loops in the design.
```

☑ 无组合逻辑环

2.4 查看FPGA全自动运行结果

- bit_gen运行时间一般较长，请耐心等待
- 查看FPGA运行结果请点击上图右侧的“fpga_eval”按钮
- 这两部分一般不会出错，如出错请及时与助教联系

```
Completed FPGA configuration
Evaluating basic benchmark suite...
Launching memcpy benchmark...
Hit good trap
pass 1 / 1
======================================
Evaluating medium benchmark suite...
Launching sum benchmark...
Hit good trap
Launching mov-c benchmark...
Hit good trap
```

⋮

```
Launching bubble-sort benchmark...
Hit good trap
pass 12 / 12
======================================
Evaluating advanced benchmark suite...
Launching shuixianhua benchmark...
Hit good trap
Launching sub-longlong benchmark...
Hit good trap
Launching bit benchmark...
Hit good trap
```

⋮

```
Launching unalign benchmark...
Hit good trap
pass 17 / 17
All benchmark evaluations passed
```

2.5 在虚拟机中生成原理图

```
cd ~/prj2/flow && make HW_ACT=sch_gen CI_RUN=y vivado_prj && cd ~/prj2
```

- 启动Vivado生成原理图（Schematics）
- 原理图PDF文件位于flow/hardware/vivado_out/rtl_chk目录

内容大纲

- 实验要点讲解
- 实验操作流程
- 其他事项

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GitLab个人远程仓库的代码提交记录、开发运行日志等
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，助教按沟通情况打分）**
4. **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**
5. **如果明显抄袭 0分记录（对代码完全说不清楚的，疑似抄袭）**

实验项目2最终完整推送

- 个人远程仓库的master分支中需包括
  - 所有.v文件的最新修改
  - 实验报告（本地提交方法下一页介绍）
- 推送方法

  cd ~/prj2 && git push origin master

- **截止时间—2021/5/24（周一） 23:59:59（NO EXTENSION！）**
  - **git push时间晚于截止时间的，一律视为“未提交”**
  - 助教以提交时远程仓库master分支的最终状态作为评分依据

实验报告

- 将实验报告的PDF版本放入个人虚拟机本地仓库的主目录，命名规则为“学号-prj2.pdf”（例如2019K8009929000-prj2.pdf），具体操作：

拷贝PDF文件到~/prj2

```
cd ~/prj2 && git add xxxx-prj2.pdf && git commit -m “Update project report”
```

- PDF文件大小尽量控制在5MB以内
- 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
  - 或者你的其它想法, 例如实验心得, 对提供帮助的同学的感谢等
  - 对实验内容扩展的想法及建议
- **更新的实验报告模板**已在SEP网站发布，请**重新下载**后编辑修改

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