实验指导文档

一、实验目标

1.使用提供的开发板,在 FPGA 上编程实现一个基于标准 32 位 MIPS 指令集的子集的流水 CPU,支持异常、中断、TLB 等。

2. 在该 CPU 上运行 ucore 操作系统,进入用户态及 shell 环境,正常执行 shell 命令。

3. 修改 ucore,实现简单的远程文件执行功能,即通过串口从 PC 上获取 ELF 文件,并在本地执行。

4. 修改编译原理课程中的 decaf 编译器,并结合 GNU Binutils,编译出可在 ucore 上执行的 ELF 文件。

5. 可选实现对 VGA、ps/2 keyboard 等其它外设的支持。

二、实验环境

1.硬件环境

提供的一块 THINPAD 教学计算机开发板,主要核心部件为 Xilinx Spartan6 xc6slx100 FPGA

FPGA	Xilinx Spartan6 xc6slx100
RAM	32-bit 字长, 4 块, 共 8MB
Flash	16-bit 字长, 共 8MB
CPLD	与 FPGA 相连, 用于 I/O
串口	2 个
USB 串口	1 个
ps/2 接口	1 个
LED 灯	16 个
数码管	2 个
拨码开关	32 个
以太网接口	1 个, 100MB
VGA 接口	1 个
USB OTG	1 个
自复位开关	4 个

2.软件环境

Xilinx ISE, 用于软件开发

三、指令系统

在本系统中我们采用的是 MIPS32 的标准子集作为指令集, 共计 48 条指令, 每一条指令是一个 32 位字。

1.指令集

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	1	0	0	1	rs					rt					immediate															
指令格式	ADDIU rt rs immediate																															
指令功能	R[t] ← R[s] + Sign-extend(immediate)																															
功能说明	对立即数 immediate 进行符号扩展后与寄存器 rs 的值求和, 结果保存到寄存器 rt 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	rs					rt					rd					0	0	0	0	0	1	0	0	0	0	1

指令格式	ADDU rd rs rt
指令功能	R[d] ← R[s] + R[t]
功能说明	将寄存器 rs 与寄存器 rt 的值求和，结果保存到寄存器 rd 中

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	rs					rt					rd					0	0	0	0	0	1	0	1	0	1	0
指令格式	SLT rd rs rt																															
指令功能	if(R[s] < R[t]) then R[d] = 1,else R[d] = 0																															
功能说明	比较寄存器 rs 与寄存器 rt 的值并根据结果对寄存器 rd 赋值																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	1	0	1	0	rs					rt					immediate															
指令格式	SLTI rt rs immediate																															
指令功能	if(R[s] < Sign-extend(immediate)) R[t] = 1, else R[t] = 0																															
功能说明	比较寄存器 rs 的值与进行符号扩展后的立即数的值，并根据结果对寄存器 rt 赋值																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	1	0	1	1	rs					rt					immediate															
指令格式	SLTIU rt rs immediate																															
指令功能	if(R[s] < Zero-extend(immediate)) R[t] = 1, else R[t] = 0																															
功能说明	比较寄存器 rs 的值与进行零扩展后的立即数的值，并根据结果对寄存器 rt 赋值																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	rs					rt					rd					0	0	0	0	0	1	0	1	0	1	1
指令格式	SLTU rd rs rt																															
指令功能	if(R[s] < R[t]) R[d] = 1, else R[d] = 0																															
功能说明	比较寄存器 rs 与寄存器 rt 的值，并根据结果对寄存器 rt 赋值																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	rs					rt					rd					0	0	0	0	0	1	0	0	0	1	1
指令格式	SUBU rd rs rt																															
指令功能	R[d] ← R[s] - R[t]																															
功能说明	用寄存器 rs 的值减寄存器 ry 的值，结果保存到寄存器 rz 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	rs					rt					0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
指令格式	MULT rs rt																															
指令功能	HI|LO ← R[s] * R[t]																															
功能说明	将寄存器 rs 与寄存器 rt 的值相乘，结果的高 32 位保存到 HI 寄存器中，低 32 位保存到 LO 寄存器中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

指令编码	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	rd					0	0	0	0	0	0	1	0	0	1	0
指令格式	MFLO rd																															
指令功能	R[d] ← LO																															
功能说明	将 LO 寄存器的值保存到寄存器 rd 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	rd					0	0	0	0	0	0	1	0	0	0	0
指令格式	MFHI rd																															
指令功能	R[d] ← HI																															
功能说明	将 HI 寄存器的值保存到寄存器 rd 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	rs					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1
指令格式	MTLO rs																															
指令功能	LO ← R[s]																															
功能说明	将寄存器 rs 的值保存到 LO 寄存器中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	rs					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
指令格式	MTHI rs																															
指令功能	HI ← R[s]																															
功能说明	将寄存器 rs 的值保存到 HI 寄存器中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	1	0	0	rs					rt					immediate															
指令格式	BEQ rs rt immediate																															
指令功能	if(R[S] = R[t]) PC ← PC + Sign-extend(immediate)																															
功能说明	如果寄存器 rs 与寄存器 rt 的值相等，则跳转到目的地址执行，否则顺序执行下一条指令																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	1	rs					0	0	0	0	1	immediate															
指令格式	BGEZ rs immediate																															
指令功能	if(R[s] >= 0) PC ← PC + Sign-extend(immediate)																															
功能说明	如果寄存器 rs 的值大于等于 0，则跳转到目的地址执行，否则顺序执行下一条指令																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	1	1	1	rs					0	0	0	0	0	immediate															
指令格式	BGTZ rs immediate																															
指令功能	if(R[s] > 0) PC ← PC + Sign-extend(immediate)																															
功能说明	如果寄存器 rs 的值大于 0，则跳转到目的地址执行，否则顺序执行下一条指令																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	1	1	0	rs					0	0	0	0	0	immediate															
指令格式	BLEZ rs																															
指令功能	if(R[s] <= 0) PC ← PC + Sign-extend(immediate)																															
功能说明	如果寄存器 rs 的值小于等于 0，则跳转到目的地址执行，否则顺序执行下一条指令																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	1	rs					0	0	0	0	0	immediate															
指令格式	BLTZ rs																															
指令功能	if (R[s] < 0) PC ← PC + Sign-extend(immediate)																															
功能说明	如果寄存器 rs 的值小于 0，则跳转到目的地址执行，否则顺序执行下一条指令																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	1	0	1	rs					rt					immediate															
指令格式	BNE rs rt																															
指令功能	if(R[s] != R[t]) PC ← PC + Sign-extend(immediate)																															
功能说明	如果寄存器 rs 与寄存器 rt 的值不相等，则跳转到目的地址执行，否则顺序执行下一条指令																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	1	0	immediate																									
指令格式	J immediate																															
指令功能	PC ← Sign-extend(immediate)																															
功能说明	无条件跳转至目的地址执行																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	1	1	immediate																									
指令格式	JAL																															
指令功能	PC ← Sign-extend(immediate), RA ← RPC																															
功能说明	无条件跳转至目的地址执行，将延时槽后一条指令的地址保存到 RA 寄存器中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	rs					0	0	0	0	0	rd					0	0	0	0	0	0	0	1	0	0	1
指令格式	JALR rs rd																															
指令功能	PC ← R[s], R[d] ← RPC																															
功能说明	无条件跳转至寄存器 rs 中所存地址执行，将延时槽后一条指令的地址保存到 RA 寄存器中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	rs					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
指令格式	JR rs																															
指令功能	PC ← R[s]																															

功能说明	无条件跳转至寄存器 rs 中所存地址执行

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	1	0	0	0	1	1	rs					rt					immediate															
指令格式	LW rt rs immediate																															
指令功能	R[t] ← MEM[R[s] + Sign-extend(immediate)]																															
功能说明	将寄存器 rs 的值与立即数 immediate 符号扩展后相加所得地址中的数据取出来保存至 rt 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	1	0	1	0	1	1	rs					rt					immediate															
指令格式	SW rt rs immediate																															
指令功能	MEM[R[s] + Sign-extend(immediate)] ← R[t]																															
功能说明	将寄存器 rt 的值存入寄存器 rs 的值与立即数 immediate 符号扩展后相加所得地址中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	1	0	0	0	0	0	rs					rt					immediate															
指令格式	LB rt rs immediate																															
指令功能	R[t] ← Sign-extend(MEM_Byte[R[s] + Sign-extend(immediate)])																															
功能说明	将寄存器 rs 的值与立即数 immediate 符号扩展后相加所得地址中的第一个字节取出来符号扩展后保存在寄存器 rt 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	1	0	0	1	0	0	rs					rt					immediate															
指令格式	LBU rt rs immediate																															
指令功能	R[t] ← Zero-extend(MEM_Byte[R[s] + Sign-extend(immediate)])																															
功能说明	将寄存器 rs 的值与立即数 immediate 符号扩展后相加所得地址中的第一个字节取出来零扩展后保存在寄存器 rt 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	1	0	1	0	0	0	rs					rt					immediate															
指令格式	SB rt rs immediate																															
指令功能	MEM_Byte[R[s] + Sign-extend(immediate)] ← LOW_BYTE[R[t]]																															
功能说明	将寄存器 rt 的最低字节取出来保存在寄存器 rs 的值与立即数 immediate 符号扩展后相加所得地址中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	rs					rt					rd					0	0	0	0	0	1	0	0	1	0	0
指令格式	AND rd rs rt																															
指令功能	R[d] ← R[s] & R[t]																															
功能说明	将寄存器 rs 与寄存器 rt 的值向与后的结果保存至寄存器 rd 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

指令编码	0	0	1	1	0	0	rs	rt	immediate
指令格式	ANDI rt rs immediate								
指令功能	R[t] ← R[s] & Zero-extend(immediate)								
功能说明	将寄存器 rs 的值与立即数零扩展后相与的结果保存至寄存器 rt 中								

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	1	1	1	1	0	0	0	0	0	rt					immediate															
指令格式	LUI rt immediate																															
指令功能	R[t] ← immediate * 65536																															
功能说明	将 16 位立即数放至寄存器 rt 的高 16 位中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	rs					rt					rd					0	0	0	0	0	1	0	0	1	1	1
指令格式	NOR rd rs rt																															
指令功能	R[d] ← ~(R[s] | R[t])																															
功能说明	将寄存器 rs 与寄存器 rt 的值或非后的结果保存至寄存器 rd 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	rs					rt					rd					0	0	0	0	0	1	0	0	1	0	1
指令格式	OR rd rs rt																															
指令功能	R[d] ← R[s] | R[t]																															
功能说明	将寄存器 rs 与寄存器 rt 的值相或后的结果保存至寄存器 rd 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	1	1	0	1	rs					rt					immediate															
指令格式	ORI rt rs immediate																															
指令功能	R[t] ← R[s] | Zero-extend(immediate)																															
功能说明	将寄存器 rs 与立即数 immediate 零扩展后相或的结果保存至寄存器 rd 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	rs					rt					rd					0	0	0	0	0	1	0	0	1	1	0
指令格式	XOR rd rs rt																															
指令功能	R[d] ← R[s] ^ R[t]																															
功能说明	将寄存器 rs 与寄存器 rt 的值异或后的结果保存至寄存器 rd 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	1	1	1	0	rs					rt					immediate															
指令格式	XORI rt rs immediate																															
指令功能	R[t] ← R[s] ^ Zero-extend(immediate)																															
功能说明	将寄存器 rs 与立即数 immediate 零扩展后相异或的结果保存至寄存器 rd 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	0	0	0	0	0	rt					rd					immediate					0	0	0	0	0	0
指令格式	SLL rd rt immediate																															
指令功能	R[d] ← R[t] << immediate																															
功能说明	将寄存器 rt 中的值逻辑左移立即数 immediate 位后的结果保存至寄存器 rd 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	rs					rt					rd					0	0	0	0	0	0	0	0	1	0	0
指令格式	SLLV rd rt rs																															
指令功能	R[d] ← R[t] << R[s]																															
功能说明	将寄存器 rt 中的值逻辑左移寄存器 rs 中的值位后的结果保存至寄存器 rd 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	0	0	0	0	0	rt					rd					immediate					0	0	0	0	1	1
指令格式	SRA rd rt immediate																															
指令功能	R[d] ← R[t] >> immediate(arithmetic)																															
功能说明	将寄存器 rt 中的值算数右移立即数 immediate 位后的结果保存至寄存器 rd 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	rs					rt					rd					0	0	0	0	0	0	0	0	1	1	1
指令格式	SRAV rd rt rs																															
指令功能	R[d] ← R[t] >> R[s](arithmetic)																															
功能说明	将寄存器 rt 中的值算数右移寄存器 rs 中的值位后的结果保存至寄存器 rd 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	0	0	0	0	0	rt					rd					immediate					0	0	0	0	1	0
指令格式	SRL rd rt immediate																															
指令功能	R[d] ← R[t] >> immediate(logical)																															
功能说明	将寄存器 rt 中的值逻辑右移立即数 immediate 位后的结果保存至寄存器 rd 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	rs					rt					rd					0	0	0	0	0	0	0	0	1	1	0
指令格式	SRLV rd rt rs																															
指令功能	R[d] ← R[t] >> R[s](logical)																															
功能说明	将寄存器 rt 中的值逻辑右移寄存器 rs 中的值位后的结果保存至寄存器 rd 中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
指令格式	SYSCALL																															
指令功能	中断号 ← SYSCALL																															

功能说明	执行后除法中断

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	1	0	1	1	1	1	rs					rt					immediate															
指令格式	CACHE																															
指令功能	无																															
功能说明	不做 cache，视为 NOP																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
指令格式	ERET																															
指令功能	PC ← EPC																															
功能说明	返回至 EPC 寄存器的地址执行，并设置 Status 寄存器的 EXL 位为 0																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	1	0	0	0	0	0	0	0	0	0	rt					rd					0	0	0	0	0	0	0	0	0	0	0
指令格式	MFC0 rt rd																															
指令功能	R[t] ← CP0[R[d]]																															
功能说明	将协处理器 0 中的 rd 寄存器的值保存到 rt 寄存器中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	1	0	0	0	0	0	0	1	0	0	rt					rd					0	0	0	0	0	0	0	0	0	0	0
指令格式	MTC0 rd rt																															
指令功能	CP0[R[d]] ← R[t]																															
功能说明	将寄存器 rt 的值保存到协处理器 0 中的 rd 寄存器中																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
指令格式	TLBWI																															
指令功能																																
功能说明	写索引 TLB 项																															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
指令编码	1	0	0	1	0	1	rs					rt					immediate															
指令格式	LHU rt rs immediate																															
指令功能	R[t] ← Zero-extend(MEM_HALFWORD[R[s] + Sign-extend(immediate]))																															
功能说明	将寄存器 rs 的值与立即数 immediate 符号扩展后相加所得地址中的低两个字节取出来零扩展后保存在寄存器 rt 中																															

2.特殊指令处理

（1）对于读写内存以及跳转的指令，地址应当按照 4 字节对齐，这意味着在进行立即数扩展时应当将立

即数乘以4后再进行处理。

（2）读写内存都是以一个字为单位，则存储字节SB指令执行时，应当先取出当前地址的数据操作后再进行存储。

（3）乘法指令时间较长并且输出到特殊的寄存器位置，可单独进行处理。

四、实现内容

1.CPU

1.1.ALU

ALU的实现可参照16位机的方法。由于数据操作变成了32位，追求更高性能的同学可能需要考虑对当前ALU实现的修改，包括使用超前进位的加法器及减法器等。

1.2.乘法器

考虑到乘法指令的特殊性，乘法器的实现通常是独立于ALU的。同一般算逻指令相比乘法器最大的不同在于其耗时较长，同访存操作一样我们必须要设置一个busy标志以表示乘法器是否在进行乘法操作，避免冲突。最后的乘法结果要写入HI和LO寄存器。

同学们可以自行设计乘法器，也可以使用已有的mult模块。下面介绍一种常用的乘法器实现理论—Booth算法。

Booth算法就是对乘数从低位开始判断，根据两个数据位的情况决定进行加法、减法还是仅仅移位操作。判断的两个数据位位当前位及其右边的位（初始化需要 增加一个辅助位0），移位操作是向右移动。当二进制数第一次遇到1时，可以用减法取代一串的1，而当遇到最后一个1后面的0时，再加上被乘数。

假设X、Y都是用补码形式表示的机器数，[X]补和[Y]补=Ys.Y1Y2…Yn，都是任意符号表示的数。比较法求新的部分积，取决于两个比较位的数位，即Yi+1Yi的状态。

布斯乘法规则归纳如下：

首先设置附加位Yn+1=0，部分积初值[Z0]补=0。

当n≠0时，判YnYn+1，

若YnYn+1=00或11，即相邻位相同时，上次部分积右移一位，直接得部分积。

若YnYn+1=01，上次部分积加[X]补，然后右移一位得新部分积。

若YnYn+1=10，上次部分积加[-X]补，然后右移一位得新部分积。

当n=0时，判YnYn+1(对应于Y0Y1)，运算规则同上只是不移位。即在运算的最后一步，乘积不再右移。

注意：

（A）比较法中不管乘数为正为负，符号位都参加运算，克服了校正法的缺点

（B）运算过程中采用变形补码运算（双符号位）

（C）算法运算时的关键是YnYn+1的状态：后者（Yn+1）减前者（Yn），判断是加减（+/-X）

实现步骤：

（1）使用Booth算法自行设计乘法器或者使用已有的乘法器模块。

（2）对MULT指令将寄存器访问所得值传输到乘法器中。

（3）等待乘法器完成乘法功能，如果发生冲突，阻塞系统。

（4）将结果写回到Hi，Lo寄存器中。

1.3.CP0

系统控制协处理器CP0是系统必须的，它主要提供管理CPU资源所需的机制，包括MMU、TLB、异常处理与Cache控制。本次实验中我们主要要通过CP0实现对TLB、MMU及异常处理的管理。

下表列出了CP0的可能用到的一些寄存器及其功能，这些寄存器并非全部必须实现。

编号	寄存器名称	寄存器功能	建议实现
0	Index	用于TLBWI指令访问TLB入口的索引序号	是

1	Random	每个时钟周期减 1 并循环，其范围为[Wired 寄存器值，TLB 入口数-1]	
2	EntryLo0	作为 TLBWI 及其他 TLB 指令接口，管理偶数页入口	是
3	EntryLo1	作为 TLBWI 及其他 TLB 指令接口，管理奇数页入口	是
4	Context	包含一个指向页表的指针	
5	UserLocal	包含软件信息，不由硬件操作	
6	Pagemask	读取和写入 TLB	
7	Wired	设定 Random 下限值	
8	HWREna	包含一个屏蔽位，该屏蔽位可决定哪个硬件寄存器是可访问的	
9	BadVAddr	捕捉最近一次地址错误或 TLB 异常（重填、失效、修改）时的虚拟地址	是
10	Count	每隔一个时钟增加 1，用作计时器，并可使能控制	是
11	EntryHi	TLB 异常时，系统将虚拟地址部分写入 EntryHi 寄存器中用于 TLB 匹配信息	是
12	Compare	Compare 保持一定值，当 Count 值与 Compare 相等时，SI_TimerInt 引脚变高电平直到有数值写入 Compare，用于定时中断	是
13	Status	表示处理器的操作模式、中断使能及诊断状态	是
14	IntCtl	控制扩展的中断使能，包括了向量中断并支持外部向量控制	
15	Cause	记录最近依次异常的原因，控制软件中断请求以及中断处理派分的向量	是
16	EPC	存储异常处理之后程序恢复执行的地址	是
17	PRId	包含处理器信息的只读处理器	
18	EBase	识别多处理器系统中不同的处理器异常向量的基地址	是
19	Config	处理器各种配置和功能信息	
20	Config1	作为 Config 的附属，包含一些缓存信息	
21	Config2	作为 Config 的附属，包含 2 级或 3 级缓存信息	
22	Config7	包含实现指定配置信息，可用于禁用核内某些使性能加强的功能	
23	WatchLo	WatchLo 和 WatchHi 寄存器共同提供一个调试点功能的接口，如果指令或数据访问的地址与这两个寄存器中存放的地址相同，便产生一个观测异常。	
24	WatchHi		
25	Debug	控制调试异常，提供异常产生的原因信息	

下面详解其中几个常用寄存器

（1）Index 寄存器

Index 寄存器是一个 32 位的读/写寄存器，可用于 TLBP、TLBR 和 TLBWI 指令访问 TLB 入口的索引序号。Index 区域的大小根据具体实现方 式随 TLB 的入口个数而定。对于基于 TLB 的内存管理单元 MMU 来说，该区域的最小值为 Ceiling(Log2(TLBEntries))。

如果一个写入 Index 寄存器的值大于等于 TLB 入口数，则该处理器的操作是未定义的。

该寄存器仅对 TLB 有效。

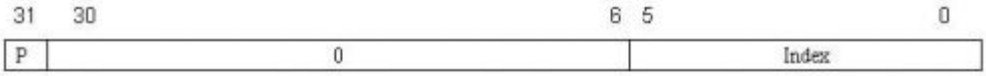

图 7-1 Index 寄存器格式

表 7-3 Index 寄存器区域描述

区域		描述	读/写	重置状态
名称	比特			
P	31	检测故障。当先前的 TLBProbe（TLBP）指令没有在 TLB 中寻得匹配时，硬件会将其置为 1。	R/W	未定义
0	30：6	必须写为 0；读取时返回 0。	0	0
Index	5：0	TLB 入口的索引值，受到 TLBRead 和 TLBWrite 指令影响。对于 16 或 32 个入口的 TLB，如果索引指向一个不存在的入口，则该行为是未定义的。	R/W	未定义

（2）EntryLo1/EntryLo0 寄存器

这对 EntryLo 寄存器的作用等同于 TLB、TLBR、TLBWI 和 TLBWR 指令间的接口。对基于 TLB 的 MMU 而言，EntryLo0 管理偶数页的 入口，EntryLo1 管理奇数页的入口。如果出现了地址错误，TLB 失效，TLB 修改或是 TLB 重填异常的行为，那么 EntryLo0 和 EntryLo1 寄存器的内容将会成为未定义的。只有当基于 TLB 的存储管理单元存在时，这些寄存器才有效。

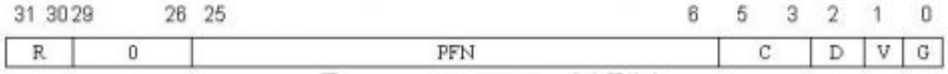

图 7-3 EntryLo0，EntryLo1 寄存器格式

：7-5 EntryLo0，EntryLo1 寄存器区域描述

区域		描述	读/写	重置状态
名称	比特			
R	31：30	保留区域。写入时应被忽略；读取时返回 0。	R	0
0	29：26	在一般情况下，这 4 个位为 PFN 的一部分。但由于 24k 仅支持 32 位的物理地址，PFN 仅有 20 位的位宽，所以，该寄存器的 29：26 位强制写入 0。	R	0
PFN	25：6	页帧号：有助于物理地址高位的定义。PFN 区域对应了物理地址的 31~12 位。	R/W	未定义
C	5：3	页面一致性属性。	R/W	未定义
D	2	已使用或写使能位：表示了该页面已经被写入，并且/或者是可写入的。如果D=1，则允许写入该页；如果D=0，则不允许写入该页，否则会引起TLB修改异常。	R/W	未定义
V	1	有效位：指明当前TLB入口是否有效，即虚拟页面映射是否有效。如果该位为1，那么允许进入该页；如果V=1，则允许访问该页；如果V=0，访问该页会引起TLB 无效异常。	R/W	未定义
G	0	全局位：在对TLB 入口进行写操作时，EntryLo0和EntryLo1 寄存器中的G位与运算的结果作为TLB入口的G位。如果TLB入口的G位是1，ASID比较将在TLB匹配中被忽略掉。在对TLB入口进行读操作时，EntryLo0和EntryLo1的G位都反映了TLB的G位的状态。	R/W	未定义

表 7-6 缓存一致性属性

C[5:3]值	缓存一致性属性
0	可缓存、非一致性、写通、无写分配
1	保留
2	不可缓存
3	缓存可、非一致性、写回、写分配
4，5，6	保留
7	无缓存加速

（3）EntryHi 寄存器

EntryHi 寄存器包含了用于 TLB 读、写和访问操作的虚拟地址匹配信息。当 TLB 异常（TLB Refill，TLB

Invalid 或 TLB Modified）发生时，系统将虚拟地址的[31:13]位写入 EntryHi 寄存器的 VPN2 区域。TLBR 指令将选中的 TLB 入口相应的区域写入 EntryHi 寄存器。软件（通常是操作系统）将当前地址空间标识符写入 ASID 区 域，该区域在 TLB 比较过程中用于确定 TLB 是否可以匹配。

由于 ASID 区域被 TLBR 指令重填覆盖了，软件必须保存和重新存储有关 TLBR 使用的 ASID 的值。这在发生 TLB 失效和 TLB 修改异常时，以及在其它存储管理软件中尤为重要。

在发生了地址错误的异常后，EntryHi 寄存器的 VPN2 区域将成为未定义的，并且该区域可能在发生地址错误异常的过程中被硬件修改。EntryHi 寄 存器的软件写操作（通过 MTC0）不会导致 BadVAddr 和 Context 寄存器中的地址相关区域发生隐式的写入（implicit write）。

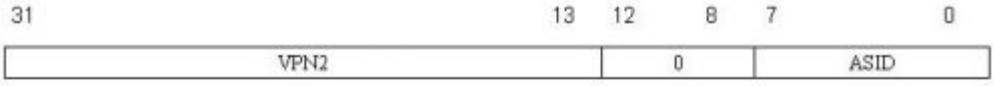

图 7-12 EntryHi 寄存器格式

表 7-16 EntryHi 寄存器区域描述

区域		描述	读/写	重置状态
名称	比特			
VPN2	31..13	虚地址（虚拟页面数目的一半）的 $VA_{31..13}$。在 TLB 异常或 TLB 进行读取时，该区域被硬件写入；在 TLB 写之前，该区域被软件写入。	R/W	未定义
0	12..8	必须写为 0；在读取时返回 0。	0	0
ASID	7..0	地址空间标识符。在TLB读取时，该区域由硬件写入；通过软件写该区域，可以为TLB写操作建立当前的ASID值，这样可以避免对TLB 的访问匹配所有TLB 入口的ASID 值。	R/W	未定义

该寄存器仅对 TLB 有效。

（4）Status 寄存器

Status 寄存器是一个读/写寄存器，可以表示处理器的操作模式、中断使能以及诊断状态。该寄存器的区域联合作用，可以创建处理器的工作模式。

中断使能：当以下所有条件都成立时启用中断：

Status[0]:IE = 1

Status[0]:EXL = 0

Status[0]:ERL = 0

额外的：Debug[0]:DM = 0

当这些条件都符合时，设置 IM（Status[16:9]）位和 IE 位可以使能中断。

EXL 与 ERL 任一位置 1 都可使系统进入 Kernel 模式，否则为 User 模式

（5）Cause 寄存器

Cause 寄存器主要记录最近一次异常的原因，也控制软件中断请求以及中断处理派分的向量。除了 IP1..0、DC、IV 和 WP 区域，Cause 寄存器中其它的所有 区域都是只读的。第二版架构在外部中断控制器（EIC）的中断模式下，增加了可选项。在这个模式下，IP7. .2 表示请求中断优先级（RIPL）。

Cause[6:2] 表示 ExcCode，即异常号

（6）Ebase 寄存器

EBase 寄存器是一个读/写寄存器，包含了在 StatusBEV 为 0 时所使用的异常向量的基地址及一个只读 CPU 号，该 CPU 号可以被软件用来区分多处理器系统中不同的处理器。

EBase 寄存器使软件能在多处理器系统中识别特定的处理器，并允许每个处理器的异常向量可以不同，特别是对由多种处理器组成的系统。当 StatusBEV 为 0 时，EBase 寄存器的位 31:12 由 0 构成，形成异常向量的基地址。当 StatusBEV 为 1 时，或在任何 EJTAG 调试异常的 情况下，异常向量的基地址由固定的缺省值确定。

EBase 寄存器的位 31:30 固定为 2#10，从而强制异常基地址在 kseg0 或 kseg1 的无映射的虚拟地址段中。在缓存错误异常中，异常基地址的位 29 将被强制置为 1，从而使异常处理器从无缓存的 kseg1 段开始执行。如果需要改变异常基地址寄存器的值，则操作必须在 StatusBEV 为 1 的情况下 进行。如果在 StatusBEV 为 0 时，在异常基地址区域写入了一个不同的值，处理器的操作将成为未定义的。

将位 31:12 与异常基地址区域相结合，可允许将异常向量的基地址置于任何 4KB 字节大小的页面的边界上。

图 7-21 EBase 寄存器格式

表 7-27 EBase 寄存器区域描述

区域		描述	读/写	重置状态
名称	比特			
1	31	在写入时，该位被忽略。在读取时，返回 0。	R	1
Exception	29:12	与位 31_30 结合，该区域在 $Status_{BEV}$ 为 0 时，指定了异常的基地址。	R/W	0
CPUNum	9:0	该区域包含了一个标识符，它对多处理器系统的每个 CPU 都是唯一的。软件可以用此来判断它运行的位置。这个区域的值由连接到内核的 *SI_CPUNum[9:0]*静态输入管脚设置。	R	外部设置
0	30, 11:10	必须写为 0；在读取时返回 0。	0	0

通过调用 MFC0, MTC0 指令, CP0 提供了统一的对外接口以完成对寄存器组的访问。

实现步骤：

(1) 选择要实现的 CP0 寄存器, 可考虑推荐实现的寄存器或自行决定额外寄存器实现。

(2) 按照 CP0 寄存器的功能分别在 cpu 的不同模块完成各寄存器的赋值。

a. 在译码、运算、及访存阶段发生地址错误时将错误地址赋值给 BadVAddr

b. 异常处理开始时, 若为 tlb 异常, 则将错误地址高 20 位赋值给 EntryHi 高 20 位

c. 异常处理开始时, 将 Status[1]赋值为 1；在执行 ERET 指令时将 Status[1]赋值为 0

d. 异常处理开始时, 将 Cause[6:2]赋值为异常号

e. 异常处理开始时, 将 EPC 赋值为 victim 指令地址

f. 每个周期, Count 加 1

g. 其他写操作由软件完成

(3) 实现 MFC0, MTC0 指令访问 CP0 寄存器的功能。

(4) 通过各寄存器值控制相应功能

1.4.中断

下表列出了一些可能用到的中断及异常的情况

异常号	异常名	描述
0	Interrupt	外部中断, 异步发生, 由硬件引起
1	TLB Modified	内存修改异常, 发生在 Memory 阶段
2	TLBL	读未在 TLB 中映射的内存地址触发的异常
3	TLBS	写未在 TLB 中映射的内存地址触发的异常
4	ADEL	读访问一个非对齐地址触发的异常
5	ADES	写访问一个非对齐地址触发的异常
8	SYSCALL	系统调用
10	RI	执行未定义指令异常
11	Co-Processor Unavailable	试图访问不存在的协处理器异常
23	Watch	Watch 寄存器监控异常

下表列出可能用到的中断号

中断号	设备

0	系统计时器
1	键盘
3	通讯端口 COM2
4	通讯端口 COM1

中断/异常处理的一般流程如下：

（1）保存中断信息，主要是 EPC，BadVAddr，Status，Cause 等寄存器的信息。

EPC：存储异常处理之后程序恢复执行的地址。对于一般异常，当前发生错误的指令地址即为 EPC 应当保存的地址；而对于硬件中断，由于是异步产生则可以任意设定一条并未执行完成的指令地址保存，但在进入下一步处理之前，该指令前的指令都应当被执行完。

BadVAddr：捕捉最近一次地址错误或 TLB 异常（重填、失效、修改）时的虚拟地址。

Status：将 EXL 位置为 1，进入 kernel 模式进行中断处理

Cause：记录下异常号。

EnrtyHi：tlb 异常时，记录下 BadVAddr 的部分高位。

（2）根据 Cause 中的异常号跳转到相应的异常处理函数入口

（3）中断处理

（4）通过调用 ERET 指令恢复现场，返回 EPC 所存地址执行并且将 Status 中的 EXL 重置为 0 表示进入 user 模式。

实现步骤：

（1）在可能发生异常的位置实现对异常的记录。

a.访存时可能发生 ADEL，ADES，TLBM，TLBL，TLBS，Watch 异常

b.译码后可能发生 RI，SYSCALL，Co-ProcessorU 异常

（2）实现对中断的记录。

a.硬件产生中断时将信息写入 CP0 寄存器

（3）根据异常记录信息判断是否产生异常。

（4）进入异常处理流程。

1.5.MMU

内存管理单元 MMU 为系统提供了从虚拟地址到物理地址的转换，同时能为处理器发出的地址进行合法性检验，在硬件上提供了内存访问授权控制。MMU 建立起了 CPU 与内存之间访问的接口。

工作流程：

处理器发出的虚拟地址经过合法性检测 -> 虚拟地址静态地址映射 -> 静态映射算法计算出物理地址 -> 直接交由总线接口访存；

处理器发出的虚拟地址经过合法性检测 -> 虚拟地址动态地址映射 -> TLB 查找地址映射表 -> TLB 命中得到物理地址 -> 交由总线接口访存；

处理器发出的虚拟地址经过合法性检测 -> 虚拟地址动态地址映射 -> TLB 查找地址映射表 -> TLB 缺失异常 -> 异常处理重填 TLB -> 返回现场重新执行指令；

TLB 的设计：

TLB 实际上就是一个寄存器组，用作页表的缓存。根据表大小以及相连度不同而不同，TLB 可以有多种设计方案，这里提供一种设计方案。

对于一个拥有 2 的 N 次方个表项的 TLB 来说，每个表项有 64 位长，每次更新时它的信息被存放在如下寄存器中：目录为 INDEX[N-1:0]，对应的每个表项即每个入口为{EntryHi[31:13]，EntryLo1[25:6]，EntryLo1[2:1]，EntryLo0[25:6]，EntryLo0[2:1]}。

即高 22 位存储了虚拟地址高 22 位，后面分别跟着两组物理地址（奇偶）及其标志位（Valid，Global），匹配虚拟地址与标志后将实地址合并得到物理地址。

实现步骤：

（1）设计实现 TLB，包括查询与重写，注意异常信息的处理

（2）实现虚拟地址到物理地址的转换。

2.BIOS

BIOS即为启动 ucore 所用的 bootloader 程序，通常我们直接将其放置在一块只读的存储单元中，可以是在 FLASH 中。

另一种实现的方法是在 FPGA 里面建立一块 ROM，因为 bootloader 很小，我们可以直接将其指令放置在该 ROM 中，并且设置我们的 CPU 访问起始地址从该 ROM 中开始。这就有效避免了因为不安全的 FLASH 的读写操作对 BIOS 造成的破坏，并且将 ucore 与其独立开来。

3.Ucore

为了实现从 PC 上抓取文件并在本地执行,需要实现一个抓取远程文件的系统调用。为此,首先修改了控制台驱动,当输出一个字符时,会先往串口写入一个 MAGIC0 字符,再写入实际输出的字符;另外增加了抓取文件的系统调用即 fetchrun,它被调用时会先向串口写入 MAGIC1 字符,接下来进入抓取文件相关的通信协议。同时需要修改 PC 上的终端程序,让它遇到 MAGIC0 时就显示接下来的字符,遇到 MAGIC1 时就进入文件传输模式。另外由于 ucore 的 VFS 不支持新建文件,因此只能先在 ramdisk 中预留好一个足够大的空文件,把抓取到的文件内容写入这个其中。

4.decaf 编译器

4.1.汇编指令生成

由于简化的 CPU 中并未实现 add、sub 指令,需要把 decaf 的 MIPS 后端里生成 add、sub 指令的部分改成 addu、subu,区别仅在于溢出时后者不会产生异常。

另外,CPU 中也未实现除法指令,不过由于所用测试程序中没有除法运算,因此也未进行相关修改;如果需要除法,可以用其它指令手动实现除法函数,并把除法翻译成函数调用。

当然,一个更好的方法应该是修改 ucore 系统,在异常处理中捕捉非法指令异常,并软件模拟未实现的指令。这样,对于编译器而言,目标机器就是一个标准的 MIPS32 CPU 了。

4.2.库函数调用及 calling convention

标准 MIPS32 使用 O32 ABI,函数调用的前四个参数通过$a0-$a3 四个寄存器传输;但 decaf 编译出的程序的参数全都在栈上传递。当然,无论什么 calling convention,只要能自恰,程序本身就应该能正常运行,所以需要解决的问题只有用户程序与 C 实现的库函数及操作系统交互的部分。

一种常规解决方案是修改 decaf 编译器,使得其遵循 O32 ABI,直接调用相应的函数。但这需要对后端进行较大的改动,也会造成与现有 decaf 编译出的二进制代码的不兼容。

在这里,如果把我们的 MIPS 系统看作一个要移植到的目标平台,并追求对 decaf 尽量少的改动,可以采用一种逆向的思路:在 decaf 和库函数之间增加一个适配器层,将 decaf 的调用约定翻译成 O32 ABI 再调用库函数。

4.3.程序入口及退出

我们直接使用了 ucore 里的 linker script(user.ld)以及用户静态函数库 libuser.a,在该环境下系统会设置好一些全局变量,然后跳转到 main 执行。我们修改了 decaf 编译器,将其输出的 main 重命名为 decaf_main,然后汇编实现了一个新的 main 函数。由于 decaf 的 main 是 void 类型,我们便默认其都执行成功返回 0,于是在 decaf_main 返回后直接调用 exit(0)。

5.VGA 与 ps/2 键盘

参照 16 位机的实现方案。

6.物理设备的管理

一个常用的物理设备管理方法为内存映射，将某个物理设备映射到未被使用的物理地址，这样根据地址的不同就可以通过访存的方法进行物理设备的管理。