计算组成原理

单周期 CPU 模拟器的实现

项目到期：11 月 30 日下午 23:59

1. 简介

在这个项目中，您将使用 C 语言实现一个称为 MiniCPU 的单周期 CPU 模拟器。 您的 MiniCPU 将演示 MIPS 处理器的一些功能以及数据路径和控制信号的原理。 MiniCPU 应该读入包含 MIPS 机器代码（以下指定格式）的文件，并逐周期模拟 MIPS 处理器的工作。 将为您提供一个名为 component.c 的 C 文件，该文件实现了单周期数据路径的每个组件，您需要修改并填写此文件中的函数主体。

2. 模拟器规格

2.1 模拟指令

附录中的图 1 列出了 14 条指令。 请注意，您无需处理导致异常的情况。

2.2 登记要处理

MiniCPU 应该处理 32 个通用寄存器。 在程序开始时，将寄存器初始化为 minicpu.c 中指定的值

2.3 内存使用情况

- MiniCPU 的内存大小为 64kB（地址 0x0000 至 0xFFFF）。
- 系统假定所有程序都从内存位置 0x4000 开始。
- 所有指令在存储器中都按字对齐，即所有指令的地址均为 4 的倍数。
- 模拟器（和 MIPS 处理器本身）将内存视为一个段。 （将内存

分为文本，数据和堆栈段只能由编译器/汇编器完成。）

- 在程序启动时，除“ -asc”文件中指定的内存外，所有内存均初始化为零，如提供的代码所示。
- 内存采用以下格式:

  例如 将 32 位数字 0xaabbccdd 存储在内存地址 0x0 – 0x3 中。

	Mem[0]			
Address	0x0	0x1	0x2	0x3
Content	aa	bb	cc	dd

2.4 MiniCPU 应该停止的条件

如果遇到以下情况之一，则全局标志 Halt 设置为 1，因此模拟停止。

- 遇到非法指令。 图 1 中超出指令列表的指令是非法的。
- 跳转到未字对齐的地址（4 的倍数）
- lw 或 sw 的地址未按字对齐
- 访问数据或跳转到内存以外的地址。

2.5 输入机器代码文件的格式

MiniCPU 将十六进制格式的机器代码（文件名为 xxx.asc）作为输入。.asc 文件的示例如下所示。 任何行上“＃”之后的代码均视为注释。

```
20010000    #addi $1, $0, 0
200200c8    #addi $2, $0, 200
10220003    #beq $1, $2, 3
00000020    #delay slot
20210001    #addi $1, $1, 1
00000020    #no operation
```

当遇到非法指令（例如 0x00000000）时，模拟结束。

2.6 分支寻址注意事项

MIPS 以及 MiniCPU 中的分支偏移与下一条指令 (PC + 4) 有关。例如:

Assembly code

```
        beq $1, $2, label
        beq $3, $4, label
label:  beq $5, $6, label
```

Machine codes

4	1	2	0x0001
4	3	4	0x0000
4	5	6	0xffff

3. 资源

3.1 提供的文件

请从 ftp://ftp.must.edu.mo/下载 project.zip，以下是解压缩后的文件:

minicpu.c minicpu.h component.c minicpuasm.pl incommand
test01.asm test01.asc test02.asm test02.asc

这些文件包含模拟器的主程序和其他支持功能。 该代码应该是不言自明的。

您需要填写并修改 component.c 中的功能。 **您不允许修改 minicpu.c 和 minicpu.h。 您的所有作品都应仅放在 component.c 中。 您无权添加新文件。 否则，您的程序将不会得分。**

3.2 MIPS 汇编器

提供一个简单的汇编程序 minicpuasm.pl 是为了方便您测试 MinCPU。

该命令是:

```
$minicpuasm.pl filename.asm > filename.asc
```

其中 filename.asm 是您的汇编代码文件，而 filename.asc 是十六进制格式的输出机器代码文件。

4. component.c 中的功能

首先，您需要在 component.c 中完成一个模拟 ALU 操作的函数（ALU（…））

```
void ALU(unsigned A, unsigned B, char ALUControl, unsigned *ALUresult, char *Zero)
{
if(ALUControl==0x0)*ALUresult=A+B;            //add
}
```

- ALU(…)

  1.根据 ALUControl 对输入参数 A 和 B 进行操作。

  2.将结果输出到 ALUresult。

  3.如果结果为零，则将 0 分配给 1；否则，为 0。 否则，分配 0。

  4.下表显示了 ALU 的操作。

ALUControl	Meaning
000	Z = A + B
001	Z = A – B
010	if A < B, Z = 1; otherwise, Z = 0
011	if A < B, Z = 1; otherwise, Z = 0 (A and B are unsigned integers)
100	Z = A AND B
101	Z = A OR B
110	Shift B left by 16 bits
111	Z = NOR(A,B)

其次，您需要在 component.c 中填写 9 个函数。 每个函数都模拟数据路径的一部分的操作。 附录中的图 2 显示了数据路径以及您需要模拟的数据路径部分。

在 minicpu.c 中，函数 Step（）是 MiniCPU 的核心功能。 该函数调用您需要实现的 9 个函数，以模拟在数据路径的组件之间传递的信号和数据。 彻底阅读 Step（）以便理解信号和数据传递，并实现 9 个功能。

- instruction_fetch(…)

  1.从内存中提取 PC 寻址的指令并将其写入指令。

  2.如果遇到无效指令，则返回 1；否则，返回 1。 否则，返回 0。

```
int instruction_fetch(unsigned PC, unsigned *Mem, unsigned *instruction)
{
 *instruction=Mem[PC>>2];
 return 0;
}
```

- instruction_partition(…)

  1.将指令分为几部分（op，r1，r2，r3，funct，offset 和 jsec）。

  2.阅读 minicpu.c 的 41 至 47 行以获取更多信息。

```
void instruction_partition(unsigned instruction, unsigned *op, unsigned *r1,
unsigned *r2, unsigned *r3, unsigned *funct, unsigned *offset, unsigned *jsec)
{
    *op = instruction >> 26;
}
```

- instruction_decode(…)

  1.根据操作码（op）解码指令。

  2.为结构控件中的变量（控制信号）分配适当的值。

  控制信号值的含义：

  对于 MemRead，MemWrite 或 RegWrite，值 1 表示启用，0 表示禁用，2 表示“无关”。

  对于 RegDst，Jump，Branch，MemtoReg 或 ALUSrc，值 0 或 1 表示多路复用器的选定路径； 2 表示“无关”。

  下表显示了 ALUOp 值的含义。

value (binary)	Meaning
000	ALU will do addition or “don’t care”
001	ALU will do subtraction
010	ALU will do “set less than” operation
011	ALU will do “set less than unsigned” operation
100	ALU will do “and” operation
101	ALU will do “or” operation
110	ALU will shift left *extended_value* by 16 bits
111	The instruction is an R-type instruction

3.如果发生停止情况，则返回 1；否则，返回 1。 否则，返回 0。

```
int instruction_decode(unsigned op, struct_controls *controls)
{
    if(op==0x0){ //R-format
    controls->RegWrite = 1;
    controls->RegDst = 1;
    controls->ALUOp = 7;
    }

    else return 1;      //invalid instruction
    return 0;
 }
```

- read_register(…)

1.从 Reg 读取 r1 和 r2 寻址的寄存器，并将读取的值分别写入 data1 和 data2。

```
void read_register(unsigned r1, unsigned r2, unsigned *Reg, unsigned *data1,
unsigned *data2)
{
*data1 = Reg[r1];
*data2 = Reg[r2];
}
```

- sign_extend(…)

1.将 offset 的符号扩展值分配给 extended_value。

```
void sign_extend(unsigned offset, unsigned *extended_value)
{

}
```

- ALU_operations(…)

1.基于 ALUOp 和 funct，对 data1，data2 或 extended_value 执行 ALU 操作。
2.调用函数 ALU（…）执行实际的 ALU 操作。
3.将结果输出到 ALUresult。
4.如果发生停止情况，则返回 1；否则返回 1。 否则，返回 0。

```
int ALU_operations(unsigned data1, unsigned data2, unsigned extended_value,
unsigned funct, char ALUOp, char ALUSrc, unsigned *ALUresult, char *Zero)
{
switch(ALUOp){
    // R-type
    case 7:
        // funct = 0x20 = 32, add
        if   (funct==0x20) ALU(data1, data2, 0x0, ALUresult, Zero);
        else return 1;                    //invalid funct
        break;
    default:
        return 1;          //invalid ALUop
}
return 0;
}
```

- rw_memory(…)

  1.根据 MemWrite 或 MemRead 的值确定内存写操作或内存读操作。

  2.读取 ALUresult 寻址到 memdata 的内存位置的内容。

  3.将 data2 的值写入 ALUresult 寻址的存储位置。

  4.如果发生停止情况，则返回 1；否则返回 1。 否则，返回 0。

```
int rw_memory(unsigned ALUresult, unsigned data2, char MemWrite, char
MemRead, unsigned *memdata, unsigned *Mem)
{
    if (MemRead==1){
    *memdata = Mem[ALUresult>>2];
    }
    return 0;
}
```

- write_register(…)

  1. 将数据（ALUresult 或 memdata）写入由 r2 或 r3 寻址的寄存器（Reg）。

```
void write_register(unsigned r2, unsigned r3, unsigned memdata, unsigned
ALUresult, char RegWrite, char RegDst, char MemtoReg, unsigned *Reg)
{
    Reg[r2] = memdata;
}
```

- PC_update(…)

1. 更新程序计数器（PC）。

```
void PC_update(unsigned jsec, unsigned extended_value, char Branch, char Jump, char Zero, unsigned *PC)
{
    *PC+=4;
}
```

文件 minicpu.h 是头文件，其中包含存储控制信号的结构的定义以及上述功能的原型。 这些功能可能包含一些参数。 阅读 minicpu.h 以获取更多信息。

5. 注意事项

1.该项目将使用 Dev C ++进行编译和标记。您可以从网站（http://www.bloodshed.net/dev/devcpp.html）下载它，并将其安装在计算机上。请记住，您应该下载并安装用于 C / C ++的 Dev C ++。

2.一些指令可能会尝试写入寄存器$ zero，我们假设它们是有效的。但是，您的模拟器应始终保持$ zero 0 的值。

3.您不应该在 component.c 中执行任何“打印”或“ printf（）”操作；否则，该操作会打乱标记过程，并扣除您的标记。

4.运行已编译的可执行文件：

A．在 Windows 中，打开命令提示符。

B．转到您的工作目录。

C．类型：your_executable input_asc_file <incommand>

其中 incommand 是下载的文件。输出显示所有寄存器和存储器位置的值，使您可以检查模拟器是否可以产生正确的结果。

```
cmd:
 cont

 cmd:
 $zero 00000000      $at   00000000      $v0   00000000      $v1   00000000
 $a0   00000000      $a1   00000000      $a2   00000000      $a3   00000000
 $t0   00000002      $t1   00000003      $t2   00000005      $t3   00000001
 $t4   00000002      $t5   00000003      $t6   00000002      $t7   00000000
 $s0   00010000      $s1   00000001      $s2   00000001      $s3   00000001
 $s4   00000001      $s5   00000000      $s6   00000000      $s7   00000000
 $t8   00000000      $t9   00000000      $k0   00000000      $k1   00000000
 $gp   0000c000      $sp   0000fffc      $fp   00000000      $ra   00000000
 $pc   00004034      $stat 00000000      $lo   00000000      $hi   00000000

 cmd:
 00000         00000002
 00004-03ffc   00000000
 04000         20080002
 04004         20090003
 04008         01285020
 0400c         01285822
 04010         01286024
 04014         01286825
 04018         ac080000
 0401c         8c0e0000
 04020         3c100001
 04024         0109882a
 04028         0109902b
 0402c         29130003
 04030         2d140003
 04034-0fffc   00000000

 cmd:
 quit
```

5.要调试程序，请检查所有寄存器和存储器的值是否与汇编程序匹配。以下是示例输出：

6. 要提交您的工作，请在组件的开头 component.c，键入您的英文名称，例如

```
/*
* Designer:     name, student id, email address
*/
```

Email your *component.c* to yyliang.fit.must@gmail.com. The subject of your email must be: **CO101 Project student_name**.