1. 实验名称

指令集体系结构

2. 实验报告作者

信息安全 1410658 杨旭东

3. 实验内容

设计能够满足基本算术运算需求的 RISC 型指令集体系结构 NK-CPU。

4. 实验设计依据

对桌面应用程序的期望：

（1） 以载入-存储体系结构使用通用寄存器。

（2） 支持以下寻址方式：位移量（地址偏移大小为 12~16 位）、立即数（大小为 8~16 位）和寄存器间接寻址。

（3） 支持以下数据大小和类型：8 位、16 位、32 位和 64 位整数以及 64 位 IEEE754 浮点数。

（4） 支持以下简单指令（它们占所执行指令的绝大多数）：载入、存储、加、减、移动寄存器和移位。

（5） 相等、不等于、小于、分支（长度至少为 8 位的 PC 相对地址）、跳转、调用和返回。

（6） 如果关注性能则使用定长指令编码，如果关注代码规模则使用变长指令编码。

（7） 至少提供 16 个通用寄存器，确保所有寻址模式可应用于所有数据传送指令，希望获取最小规模指令集。

以 MIPS64 为蓝本，设计 NK-CPU。

5. 实验结果与分析

5.1. 寄存器数目（比特长度、用途）

32 个 64 位通用寄存器（GPR）和三个特殊寄存器（PC、HI、LO）

REGISTER	NAME	USAGE
0	$zero	the value 0 常量 0
1	$at	(**a**ssembler **t**emporary) reserved by the assembler 汇编保留寄存器（不可做其他用途）
2-3	$v0 - $v1	(**v**alues) from expression evaluation and function results （**V**alue 简写）存储表达式或者是函数的返回值
4-7	$a0 - $a3	(**a**rguments) First four parameters for subroutine. Not preserved across procedure calls （**A**rgument 简写）存储子程序的前 4 个参数，在子程序调用过程中释放

8-15	$t0 - $t7	(**t**emporaries) Caller saved if needed. Subroutines can use w/out saving. Not preserved across procedure calls （**T**emp 简写）临时变量，同上调用时不保存
16-23	$s0 - $s7	(**s**aved values) - Callee saved. A subroutine using one of these must save original and restore it before exiting. Preserved across procedure calls （**S**aved 简写）调用时保存的，或如果用，需要 SAVE/RESTORE 的
24-25	$t8 - $t9	(**t**emporaries) Caller saved if needed. Subroutines can use w/out saving. These are in addition to $t0 - $t7 above. Not preserved across procedure calls. （**T**emp 简写）算是前面$0~$7 的一个继续，属性同$t0~$t7
26-27	$k0 - $k1	reserved for use by the interrupt/trap handler （**K**ernel 简写）中断函数返回值，不可做其他用途
28	$gp	**g**lobal **p**ointer. Points to the middle of the 64K block of memory in the static data segment. （**G**lobal **P**ointer 简写）全局指针。指向 64k(2^16)大小的静态数据块的中间地址
29	$sp	**s**tack **p**ointer Points to last location on the stack. (**S**tack **P**ointer 简写）栈指针，指向的是栈顶
30	$s8/$fp	**s**aved value / **f**rame **p**ointer Preserved across procedure calls (**S**aved/**F**rame **P**ointer 简写)帧指针
31	$ra	**r**eturn **a**ddress 返回地址，目测也是不可做其他用途

PC（Program Counter 程序计数器）、HI（乘除结果高位寄存器）、LO（乘除结果低位寄存器）。进行乘法运算时，HI 和 LO 保存乘法运算的结果，其中 HI 存储高 32 位，LO 存储低 32 位；进行除法运算时，HI 和 LO 保存除法运算的结果，其中 HI 存储余数，LO 存储商。

5.2. 数据类型

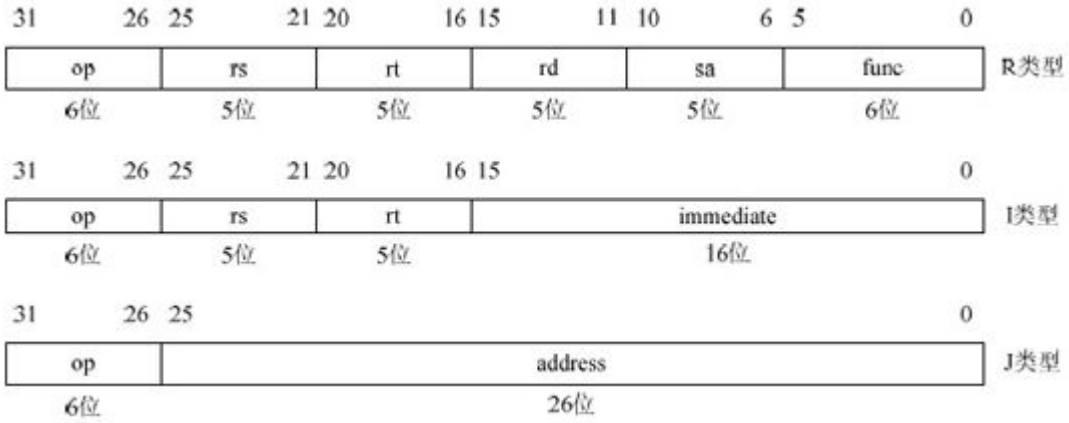
8 位字节、16 位半字、32 位字和 64 位双字整型数据。字节、半字和字被载入通用寄存器中，并通过重复 0 或符号位来填充 GPR 的 64 个位。

5.3. 寻址方式

立即数寻址和位移寻址，均采用 16 位字段。

5.4. 指令格式和编码形式

所有指令都是 32 位，也就是 32 个 0、1 编码连在一起表示一条指令，有三种指令格式。其中 op 是指令码、func 是功能码。



5.4.1. R 类型

具体操作由 op、func 结合指定，rs 和 rt 是源寄存器的编号，rd 是目的寄存器的编号，比如：假设目的寄存器是$3，那么对应的 rd 就是 00011（此处是二进制）。MIPS32 架构中有 32 个通用寄存器，使用 5 位编码就可以全部表示，所以 rs、rt、rd 的宽度都是 5 位。sa 只有在移位指令中使用，用来指定移位位数。

5.4.2. I 类型

具体操作由 op 指定，指令的低 16 位是立即数，运算时要将其扩展至 32 位，然后作为其中一个源操作数参与运算。

5.4.3. J 类型

具体操作由 op 指定，一般是跳转指令，低 26 位是字地址，用于产生跳转的目标地址。

5.5. 指令集及每条指令的功能说明

5.5.1. 移位操作指令

有 2 条指令：sll、srl。实现逻辑左移、右移。

SLL -- *Shift left logical*

Description:	Shifts a register value left by the shift amount listed in the instruction and places the result in a third register. Zeroes are shifted in.
Operation:	$d = $t << h; advance_pc (4);
Syntax:	sll $d, $t, h
Encoding:	0000 00ss ssst tttt dddd dhhh hh00 0000

SRL -- *Shift right logical*

Description:	Shifts a register value right by the shift amount (shamt) and places the value in the destination register. Zeroes are shifted in.
Operation:	$d = $t >> h; advance_pc (4);
Syntax:	srl $d, $t, h
Encoding:	0000 00-- ---t tttt dddd dhhh hh00 0010

5.5.2. 移动操作指令

有 2 条指令：mfhi、mflo，用于通用寄存器与 HI、LO 寄存器的数据移动。

MFHI -- ***Move from HI***

Description:	The contents of register HI are moved to the specified register.
Operation:	$d = $HI; advance_pc (4);
Syntax:	mfhi $d
Encoding:	0000 0000 0000 0000 dddd d000 0001 0000

MFLO -- ***Move from LO***

Description:	The contents of register LO are moved to the specified register.
Operation:	$d = $LO; advance_pc (4);
Syntax:	mflo $d
Encoding:	0000 0000 0000 0000 dddd d000 0001 0010

5.5.3. 算术运算指令

有 5 条指令：add、sub、slt、mult、div，实现了加法、减法、比较、乘法、除法运算。

AND -- ***Bitwise and***

Description:	Bitwise ands two registers and stores the result in a register
Operation:	$d = $s & $t; advance_pc (4);
Syntax:	and $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 0100

SUB -- ***Subtract***

Description:	Subtracts two registers and stores the result in a register
Operation:	$d = $s - $t; advance_pc (4);
Syntax:	sub $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 0010

SLT -- ***Set on less than (signed)***

Description:	If $s is less than $t, $d is set to one. It gets zero otherwise.
Operation:	if $s < $t $d = 1; advance_pc (4); else $d = 0; advance_pc (4);
Syntax:	slt $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 1010

MULT -- ***Multiply***

Description:	Multiplies $s by $t and stores the result in $LO.
Operation:	$LO = $s * $t; advance_pc (4);
Syntax:	mult $s, $t
Encoding:	0000 00ss ssst tttt 0000 0000 0001 1000

DIV -- ***Divide***

Description:	Divides $s by $t and stores the quotient in $LO and the remainder in $HI
Operation:	$LO = $s / $t; $HI = $s % $t; advance_pc (4);
Syntax:	div $s, $t
Encoding:	0000 00ss ssst tttt 0000 0000 0001 1010

5.5.4. 控制流指令

有 4 条指令：jr 、j 、beq、bgtz，其中既有无条件转移，也有条件转移，用于程序转移到另一个地方执行，跳转寄存器、跳转、等于、大于。

JR -- ***Jump register***

Description:	Jump to the address contained in register $s
Operation:	PC = nPC; nPC = $s;
Syntax:	jr $s
Encoding:	0000 00ss sss0 0000 0000 0000 0000 1000

J -- ***Jump***

Description:	Jumps to the calculated address
Operation:	PC = nPC; nPC = (PC & 0xf0000000) | (target << 2);
Syntax:	j target
Encoding:	0000 10ii iiii iiii iiii iiii iiii iiii

BEQ -- ***Branch on equal***

Description:	Branches if the two registers are equal
Operation:	if $s == $t advance_pc (offset << 2)); else advance_pc (4);
Syntax:	beq $s, $t, offset
Encoding:	0001 00ss ssst tttt iiii iiii iiii iiii

BGTZ -- ***Branch on greater than zero***

Description:	Branches if the register is greater than zero
Operation:	if $s > 0 advance_pc (offset << 2)); else advance_pc (4);
Syntax:	bgtz $s, offset
Encoding:	0001 11ss sss0 0000 iiii iiii iiii iiii

5.5.5. 加载存储指令

有 2 条指令：lw、sw，以“lw”是加载指令，以“sw”是存储指令，用于从存储器中读取数据，或者向存储器中保存数据。

LW -- ***Load word***

Description:	A word is loaded into a register from the specified address.
Operation:	$t = MEM[$s + offset]; advance_pc (4);
Syntax:	lw $t, offset($s)
Encoding:	1000 11ss ssst tttt iiii iiii iiii iiii

SW -- ***Store word***

Description:	The contents of $t is stored at the specified address.
Operation:	MEM[$s + offset] = $t; advance_pc (4);
Syntax:	sw $t, offset($s)
Encoding:	1010 11ss ssst tttt iiii iiii iiii iiii

5.6. 设置数据存储空间和数据初始值的宏指令

5.6.1. 数据段

以 **.data** 为开始标志。声明变量后，即在主存中分配空间。
数据声明格式：

```
name:        storage_type              value(s)

变量名：      数据类型                   变量值
```

5.6.2. 代码段

以 **.text** 为开始标志。内容为各项指令操作。
程序入口以 **main：**为标志。
结束是一个空行。

5.6.3. 示例

```
# Comment giving name of program and description of function

# Template.s

# Bare-bones outline of MIPS assembly language program

        .data       # variable declarations follow this line

                    # ...

        .text       # instructions follow this line

main:               # indicates start of code (first instruction to ex
ecute)

                    # ...

# End of program, leave a blank line afterwards to make SPIM happy
```

6. 实验心得

依照 MIPS64 设计 NK-CPU 相比于直接设计一套指令集体系结构要轻松一些，因为有很多现成的设计可以参考。实验要求设计一个 RISC 精简指令集，我仅仅从 MIPS 的指令

集中挑选了我觉得比较基础实用的寥寥数条指令，组成了 NK-CPU 的 RISC 可谓说是非常精简。挑选都是根据以往的汇编经验决定的，之后的实验中如果有需要，我觉得还可以继续添加新的指令或者减少冗余指令，并不想局限于此报告。通过设计指令集体系结构，又阅读了许多相关资料和网页，觉得对指令集体系结构有了更加深入的了解。而且对老师说的“计算机体系结构是计算机软硬件的粘合剂”有了一定感受。希望在接下来将之实现的实验中有更加清晰地认识。

7. 参考资料

[1] JohnL.Hennessy, DavidA.Patterson, 亨尼西,等. 计算机体系结构:量化研究方法[M]. 人民邮电出版社, 2013.

[2] MIPS 编程入门, http://www.itnose.net/detail/6126292.html.

[3] MIPS 通用寄存器, http://blog.csdn.net/flyingqr/article/details/7073088.

[4] MIPS32 指令集架构简介, http://imgtec.eetrend.com/blog/3151.

[5] 实验 1 附件 - Common MIPS Instructions.pdf

[6] 实验 1 附件 - MIPS Instruction Reference.pdf

[7] 实验 1 附件 - MIPS Instruction Set (core).pdf