**关于MIPS指令**

关于MIPS，是单字长定点指令平均执行速度 Million Instructions Per Second的缩写，即每秒处理的百万级的机器语言指令数。这是衡量CPU速度的一个指标。

汇编指令集分为复杂指令集（CISC）和精简指令集（RISC）两种。现在只有一种处理器是真正的RISC架构─MIPS，是以美国斯坦福大学开发的RISC架构为基础。至于x86处理器架构一开始是纯CISC设计，但过去几年来越来越朝类RISC结构发展。*ARM处理器架构近似RISC，但采用更多CISC功能，包括增加的Thumb 1与Thumb 2指令集架构。ARM体系结构除了支持执行效率很高的32位ARM指令集以外，同时支持16位的Thumb指令集。Thumb指令集是ARM指令集的一个子集，允许指令编码为16位的长度。*

**一、MIPS32指令的三种格式**

**R类型：**

31 26 25 21 20 16 15 11 10 6 5 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **op** | **rs** | **rt** | **rd** | **sa** | **func** |

6位 5位 5位 5位 5位 6位

**I类型**：

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **op** | **rs** | **rt** | **immediate**（立即数） |

6位 5位 5位 16位

**J类型：**

31 26 25 0

|  |  |
| --- | --- |
| **op** | **address** |

6位 26位

其中，

**op：**为操作码；

**rs：**为第1个源操作数寄存器，寄存器地址（编号）是：00000~11111；

**rt：**为第2个源操作数寄存器，或目的操作数寄存器，寄存器地址（同上）；

**rd：**为目的操作数寄存器，寄存器地址（同上）；

**sa：**为位移量（shift amt），移位指令用于指定移多少位；

**func：**为功能码，在寄存器类型指令中（R类型）用来指定指令的功能；

**immediate：**为16位立即数，用作无符号的逻辑操作数、有符号的算术操作数、数据加载（Load）/数据保存（Store）指令的数据地址字节偏移量和分支指令中相对程序计数器（PC）的有符号偏移量；

**address：**为地址。

MIPS32 CPU中所有寄存器助记符，如下表：



说明：在应用中助记符前要加上“$”，如$zero,$at等。

实验设计时，寄存器可以表示为：(**R0=0或$0=0**)

助记符：

**R0、R1、R2~R30、R31，或$0、$1、$2、$3~$30、$31。**

**编号：**

**00000、00001、00010-01001、01010-11111**

**比如指令中可表示为：**

**ADD R1，R2，R3;**

**或**

**ADD $1，$2，$3**

**实验中必须实现的指令如下所示：**

**==> 算术运算指令**

(1) **ADD rd , rs, rt** （**说明：以助记符表示，是汇编指令；以代码表示，是机器指令**）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 000000 | rs(5位) | rt(5位) | rd(5位) | reserved |

功能：rd←rs+rt。**reserved为预留部分，即未用，一般填“0”。**

(2) SUB rd , rs , rt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 000001 | rs(5位) | rt(5位) | rd(5位) | reserved |

完成功能：rd←rs-rt

**==> 逻辑运算指令**

(3) ORI rt , rs ,**immediate**

|  |  |  |  |
| --- | --- | --- | --- |
| 010010 | rs(5位) | rt(5位) | **immediate**(16位) |

功能：rt←rs **|** (zero-extend)**immediate；immediate**做“0”扩展再参加“或”运算

(4) AND rd , rs , rt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 010001 | rs(5位) | rt(5位) | rd(5位) | reserved |

功能：rd←rs & rt；逻辑与运算

(5) OR rd , rs , rt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 010000 | rs(5位) | rt(5位) | rd(5位) | reserved |

功能：rd←rs **|** rt；逻辑或运算

**==> 传送指令**

1. MOVE rd , rs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 100000 | rs(5位) | rt(5位) | rd(5位) | reserved |

功能：rd←rs+$r0 ；$r0=$zero=0，助记符zero=0

**==> 存储器读/写指令**

(7) SW rt ,**immediate(**rs) 写存储器

|  |  |  |  |
| --- | --- | --- | --- |
| 100110 | rs(5位) | rt(5位) | **immediate**(16位) |

功能：memory[rs+ (sign-extend)**immediate**]←rt；**immediate**符号扩展再相加

(8) LW rt , **immediate**(rs) 读存储器

|  |  |  |  |
| --- | --- | --- | --- |
| 100111 | rs(5位) | rt(5位) | **immediate**(16位) |

功能：rt ← memory[rs + (sign-extend)**immediate**]；**immediate**符号扩展再相加

**==> 分支指令**

(9)BEQ rs,rt,**immediate**

|  |  |  |  |
| --- | --- | --- | --- |
| 110000 | rs(5位) | rt(5位) | **immediate**(位移量，16位) |

功能：if(rs=rt) PC ←PC + 4 + (sign-extend)**immediate**<<2；

特别说明：**immediate是从PC+4地址开始和转移到的指令之间指令条数。immediate**符号扩展之后左移2位再相加。为什么要左移2位？由于跳转到的地方地址肯定是4的倍数（每条指令占4个字节），最低两位是“00”，因此将**immediate**放进指令码中的时候，是右移了2位的，也就是以上说的“指令之间指令条数”。

**==> 停机指令**

(10) HALT

|  |  |
| --- | --- |
| 111111 | 00000000000000000000000000(26位) |

功能：停机；不改变PC的值，PC保持不变。

**二、指令执行的几个阶段**

CPU在处理指令时，一般需要经过以下几个步骤：

(1) 取指令(**IF**)：根据程序计数器PC中的指令地址，从存储器中取出一条指令，同时，PC根据指令字长度自动递增产生下一条指令所需要的指令地址，但遇到“地址转移”指令时，则控制器把“转移地址”送入PC，当然得到的“地址”需要做些变换才送入PC。

(2) 指令译码(**ID**)：对取指令操作中得到的指令进行分析并译码，确定这条指令需要完成的操作，从而产生相应的操作控制信号，用于驱动执行状态中的各种操作。

(3) 指令执行(**EXE**)：根据指令译码得到的操作控制信号，具体地执行指令动作，然后转移到结果写回状态。

(4) 存储器访问(**MEM**)：所有需要访问存储器的操作都将在这个步骤中执行，该步骤给出存储器的数据地址，把数据写入到存储器中数据地址所指定的存储单元或者从存储器中得到数据地址单元中的数据。

(5) 结果写回(**WB**)：指令执行的结果或者访问存储器中得到的数据写回相应的目的寄存器中。

单周期CPU，是在一个时钟周期内完成这五个阶段的处理。

**图1 单周期CPU指令处理过程**

**取指令**

**IF**

**指令译码**

**ID**

**指令执行**

**EXE**

**存储器访问**

**MEM**

**结果写回**

**WB**