PA1 开天辟地的篇章：最简单的计算机

PA1-1 实现正常的寄存器结构体

在本节任务中，我们需要实现正常的寄存器结构体。原框架给出的代码如下：

```
typedef struct {
  struct {
    uint32_t _32;
    uint16_t _16;
    uint8_t _8[2];
  } gpr[8];

  /* Do NOT change the order of the GPRs' definitions. */

  /* In NEMU, rtlreg_t is exactly uint32_t. This makes RTL instructions
   * in PA2 able to directly access these registers.
   */
  rtlreg_t eax, ecx, edx, ebx, esp, ebp, esi, edi;

  vaddr_t eip;

} CPU_state
```

在 CPU 中，包含 8 个基本寄存器和 1 个 eip，每个寄存器总大小为 4 字节，并且可以按照 2 字节，1 字节访问。显然，这里应该使用匿名联合。

```
typedef struct {
  union{
    union {
      uint32_t _32;
      uint16_t _16;
      uint8_t _8[2];
    } gpr[8];

    /* Do NOT change the order of the GPRs' definitions. */

    /* In NEMU, rtlreg_t is exactly uint32_t. This makes RTL instructions
     * in PA2 able to directly access these registers.
     */
    struct {
      rtlreg_t eax, ecx, edx, ebx, esp, ebp, esi, edi;
    };
  };
  vaddr_t eip;
} CPU_state;
```

在上述代码中，gpr 的类型为匿名联合，作用是使同一寄存器变量指向的内存地址相同。而较外层的联合，作用是给 gpr 数组中元素重命名为 eax，ecx…等寄存器名称。

PA1-2

在这一阶段，难点在于表达式求值。框架将表达式求值分成两部分，第一步是词法分析，第二步是递归求值。在词法分析的过程中，使用正则表达式即可完成词法匹配，难点在于递归求值和括号匹配。

给定一个算式（1+2）*（3*4），括号匹配的方式如图 1-1 所示。

(	1	+	2	)	*	(	3	*	4	)
1	1	1	1	0	0	1	1	1	1	0

图 1-1 括号匹配辅助数组

在上图中，第一行是给定的算式，第二行是构造的辅助数组。辅助数组与算式等长，位置一一对应。当算式中对应的符号为“(”时，数组相应内容为前一位加一；为“)”时，内容为前一位减一。假设第零位是 0。

我们可以从辅助数组中看出括号匹配情况。当且仅当辅助数组所有内容非负，并且最后一位为 0 时，括号匹配正常。