# 平时作业2

> **姓名：孟启轩**
> **学号：2212452**
> **专业：计算机科学与技术**

## 问题1： P1>P2为何生成的Jbe

### If-else语句举例

```
int  get_cont( int *p1, int *p2 )
{
        if  ( p1 > p2 )
              return *p2;
        else
              return *p1;
}
```

**p1和p2对应实参的存储地址分别为R[ebp]+8、R[ebp]+12，EBP指向当前栈帧底部，结果存放在EAX。**

**作业：为何这里是”jbe”指令?**

```
  movl  8(%ebp), %eax    //R[eax]←M[R[ebp]+8], 即 R[eax]=p1
  movl  12(%ebp), %edx   //R[edx]←M[R[ebp]+12], 即 R[edx]=p2
  cmpl  %edx, %eax       //比较 p1 和 p2, 即根据p1-p2结果置标志
  jbe   .L1              //若 p1<=p2, 则转 L1 处执行
  movl  (%edx), %eax
  jmp   .L2              //无条件跳转到 L2 执行
.L1:
  movl  (%eax), %eax
.L2
```

**1. 指令功能与逻辑匹配**

`cmpl %edx, %eax` 指令比较寄存器 `%eax`（存储指针p1）和 `%edx`（存储指针p2）的值，并设置CPU标志位。随后的 `jbe .L1` 指令根据标志位判断是否满足“无符号数小于等于”条件，若成立则跳转到 `.L1` 执行else分支，返回 `*p1`，否则继续执行 `if` 分支，返回 `*p2`，直接映射了C代码的条件逻辑。

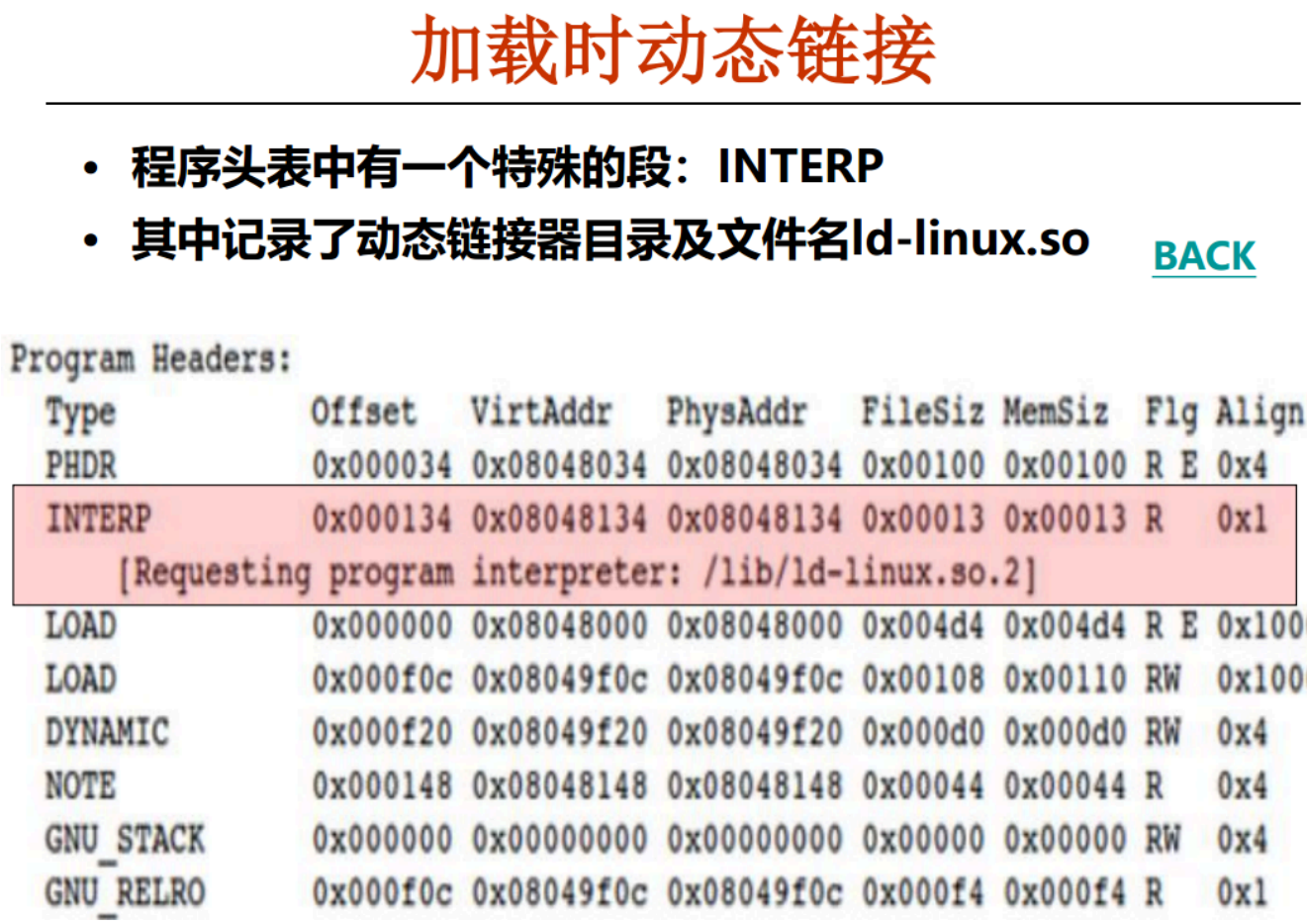
**2. 为何选择 `jbe` 而非其他跳转指令**

指针作为内存地址本质是无符号数，需要通过 `jbe` 进行无符号比较。jbe依赖 CF和ZF标志位，判断 `p1 ≤ p2` 的条件，与指针的无符号特性完全匹配，避免了符号数比较的歧义，比如高位符号位导致 `0xFFFFFFFF` 被误判为负数。同时，该逻辑与C代码 `if (p1 > p2)` 的分支条件一致，确保跳转的正确性。

**3. 效率优势**

jbe直接利用 `cmpl` 设置的标志位快速跳转，无需额外计算，显著提升执行效率。此外，无符号比较避免了符号数误判风险，例如在比较高位全1的指针地址时，jbe能正确识别其为最大地址值，而有符号比较jle会导致逻辑错误。

## 问题2：/lib/ld-linux.so.2中.2的含义




**/lib/ld-linux.so.2中的.2表示该动态链接器的版本号。**

- `.2` 明确标识了动态链接器的版本为2，表明该文件是特定版本的动态链接器，而非通用的 `ld-linux.so`。
- 版本号确保程序在运行时加载的动态链接器版本与编译时使用的版本一致。
- Linux系统允许多版本动态链接器共存（如ld-linux.so.1、ld-linux.so.2），通过版本

号区分，程序可通过INTERP段指定精确版本，确保动态库的正确加载。

- INTERP段中硬编码版本号，使程序在运行时直接绑定到特定版本的动态链接器，避免因系统更新或其他程序修改默认链接器导致的意外错误。

## 问题3：编译器将定义的符号存放在一个符号表(symbol table)中,符号表在编译器的哪里（编译出的文件中链接的符号表位置）？


编译器生成的文件中的符号表通常以特定段的形式嵌入在文件内部，例如在 ELF 格式中，静态符号表存储于 `.symtab` 段，动态链接所需的符号表存储于 `.dynsym` 段。链接器在处理文件时，会读取这些段中的符号信息，比如如符号名称、地址、类型等，以完成符号解析和重定位操作，确保程序中所有符号引用与定义正确关联。