# 内存管理 - 请求调页存储管理方式模拟

2152402 段婷婷

目录

[内存管理 - 请求调页存储管理方式模拟 1](#_Toc14348)

[1. 项目目的 2](#_Toc27945)

[2. 项目简述与功能 2](#_Toc20949)

[3. 项目设计 2](#_Toc22844)

[3.1 项目开发环境 2](#_Toc7798)

[3.2 项目运行方式 2](#_Toc2411)

[3.3 项目整体设计 2](#_Toc29134)

[4. 页面置换算法 2](#_Toc29007)

[4.1 FIFO（先进先出）算法 3](#_Toc27519)

[4.2 LRU（最近最久未使用）算法 3](#_Toc27328)

[5. 项目界面 3](#_Toc23484)

[5.1 起始界面 3](#_Toc21499)

[5.2 点击“Click to Start”开始模拟 4](#_Toc855)

[5.3 多次模拟 4](#_Toc32748)

[6. 模拟结果 4](#_Toc12644)

[7. 项目实现 4](#_Toc4549)

[5.1 全局变量 5](#_Toc4857)

[5.2 函数 5](#_Toc26184)

[5.2.1 init函数 5](#_Toc28104)

[5.4.2 generate\_instructions 函数 5](#_Toc114)

[① 随机选取一个起始执行指令 6](#_Toc1189)

[③ 直至生成320条指令。 6](#_Toc23655)

[5.4.3 execute\_simulation 函数 6](#_Toc7488)

[5.4.4 start 函数 7](#_Toc15778)

[5.4.5 is\_Available 函数 8](#_Toc1570)

[5.4.6 update\_table函数 8](#_Toc21762)

[8. 项目总结与心得 9](#_Toc15127)

1. **项目目的**

* 设计页面、页表，加深对地址转换的理解
* 实现FIFO与LRU算法来置换页面
* 加深对请求调页系统的原理和实现过程的理解

1. **项目简述与功能**

* 每个页面可存放10条指令，分配给一个作业的内存块为4。
* 模拟一个作业的执行过程，该作业有320条指令，即它的地址空间为32页，目前所有页还没有调入内存。
* 首先**生成作业中的指令访问次序**，按照以下原则生成：50%的指令是顺序执行的，25%是均匀分布在前地址部分，25％是均匀分布在后地址部分。
* **置换算法**采用FIFO与LRU算法。
* 在**模拟过程**中，如果所访问指令在内存中，则显示其物理地址，并转到下一条指令；如果没有在内存中，则发生缺页，此时需要记录缺页次数，并将其调入内存。如果4个内存块中已装入作业，则需进行页面置换。
* 所有320条指令执行完成后，计算并显示作业执行过程中发生的缺页率。

1. **项目设计**

**3.1 项目开发环境**

* + 系统：Windows 11 家庭中文版
  + IDE：Visual Studio Code 1.78.2
  + 语言：HTML、CSS、JavaScript

**3.2 项目运行方式**

* + 双击index.html进入网页即可运行。
  1. **项目整体设计**
  + index.html：通过标记和元素来描述页面的结构，包括标题、段落、列表等。
  + main.js：用于实现内存管理模拟的页面交互逻辑，包括生成指令访问次序、模拟请求调页管理方式、更新网页显示的内容等。
  + main.css：用于控制网页样式和外观的样式。通过选择器和属性来选择页面元素，并定义其样式和布局。

1. **页面置换算法**

**4.1 FIFO（先进先出）算法**

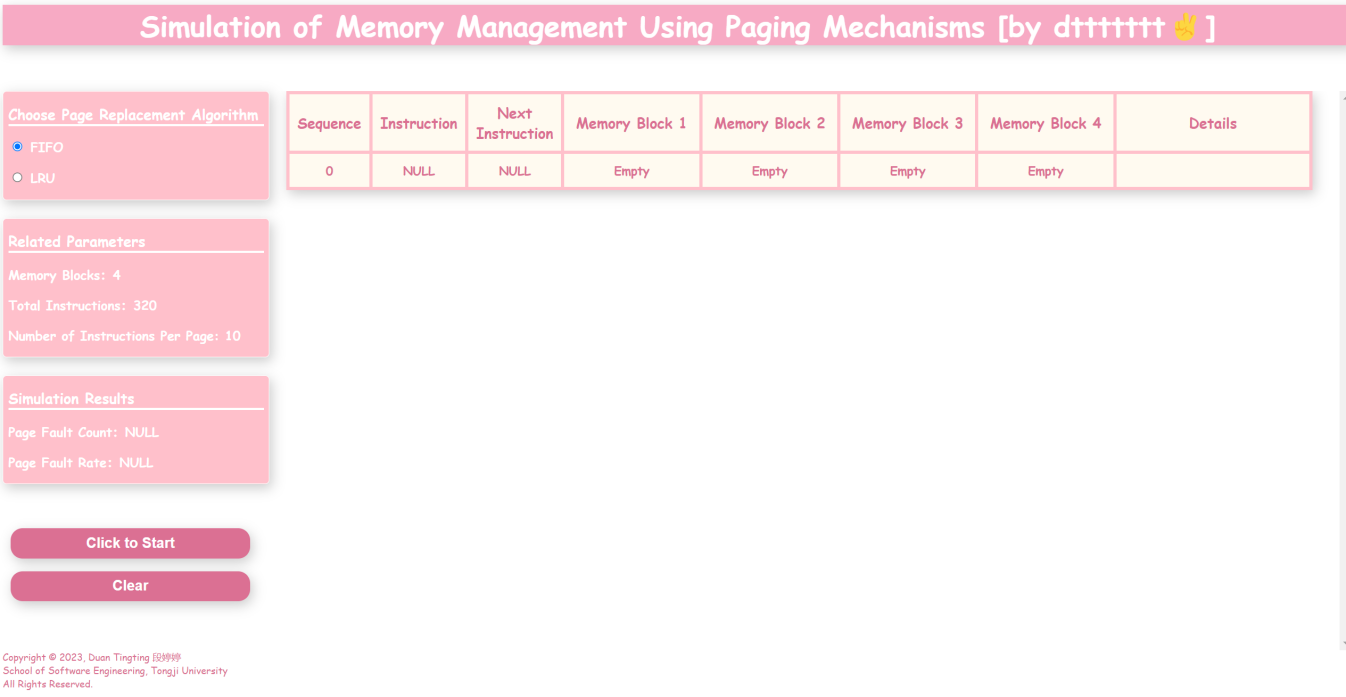
* + FIFO算法是一种简单的页面置换算法，它将最早进入内存的页面视为最先被置换出去的页面。类似于队列的先进先出原则。
  + FIFO算法的实现相对简单，适用于内存中页面访问没有明显的规律或者没有关联性的情况。然而，FIFO算法有一个明显的缺点，就是无法区分页面的重要性和访问频率，可能会导致性能下降，尤其在长时间运行的情况下。

**4.2 LRU（最近最久未使用）算法**

* + LRU算法基于页面的使用历史，将最近最久未使用的页面视为最需要被置换出去的页面。它假设在最近一段时间内没有被访问的页面，在未来也不太可能被访问到。
  + LRU算法能够比较好地适应访问模式的变化，对于经常被访问的页面会保持在内存中，提高了缓存命中率和性能。然而，实现LRU算法需要维护访问顺序的数据结构，增加了额外的开销。

1. **项目界面**

**5.1 起始界面**



总体来说分为三个部分：

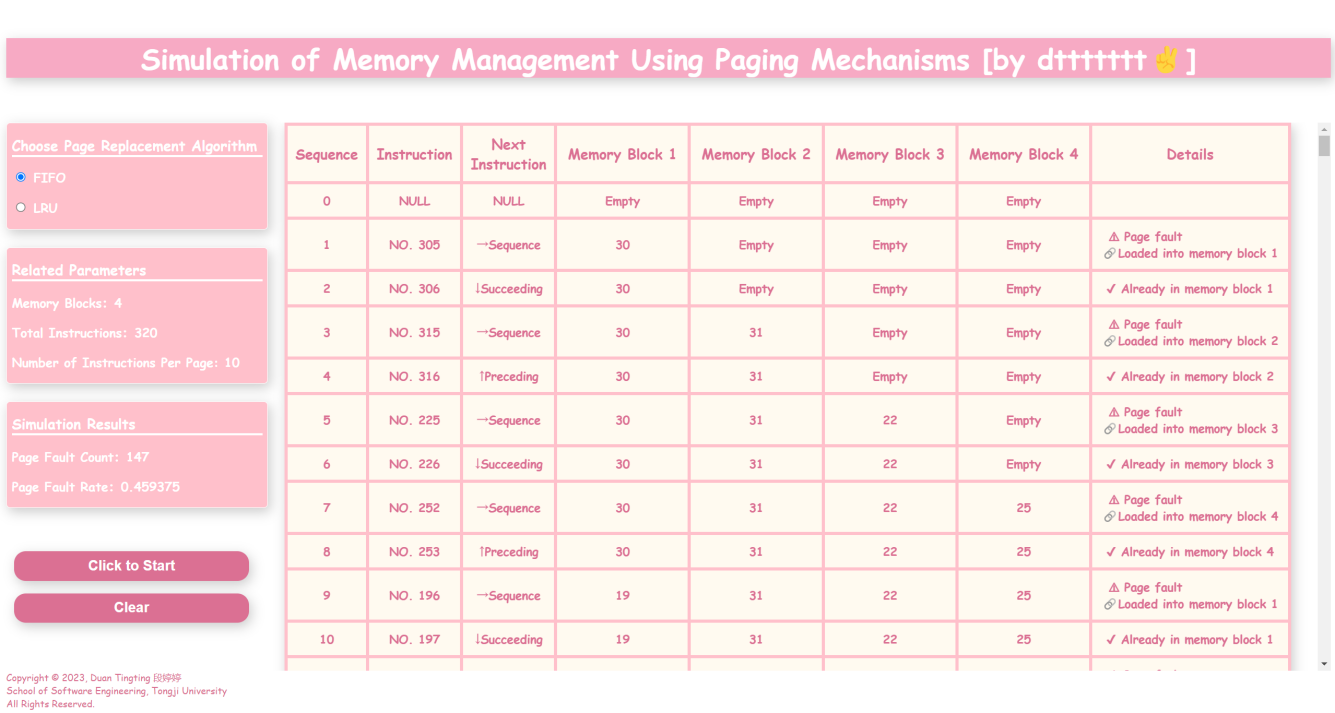
（1）顶部：页面标题——Simulation of Management Using Paging Mechanisms，即内存的请求调页管理方式的模拟。

（2）左侧：

* + 选择调页算法：可选FIFO或LRU
  + 显示模拟相关参数：内存块数目、作业指令总数目、单页指令数目
  + 显示模拟结果：缺页次数、缺页率
  + 开始与清空按钮：点击”Click to Start”开始模拟，点击”Clear”清空页面

1. 右侧：显示内存块中存储的页号与指令运行情况等。

**5.2 点击“Click to Start”开始模拟**



下面详细解释右侧表格的内容：

* + **Sequence**：当前执行过的指令数目，按照设定，一共会执行320条指令。
  + **Instruction**：当前执行的指令序号，这个序号是根据指令所在的地址标定的。
  + **Next Instruction**：下一条执行指令的类型，有Sequence（顺序执行）、Preceding（跳转至后地址）、Succeeding（跳转至前地址）三种类型。
  + **Memory Block i**：描述第i个内存块中存储的页号，一共有4个内存块。
  + **Details**：当前指令的调入情况——若已在内存中，则显示所在的内存块号；若不在内存中，则调入内存，显示调入块号。注意调入时可能需要置换页面。

左侧模拟结果一栏会显示模拟结果的缺页次数与缺页率。

**5.3 多次模拟**

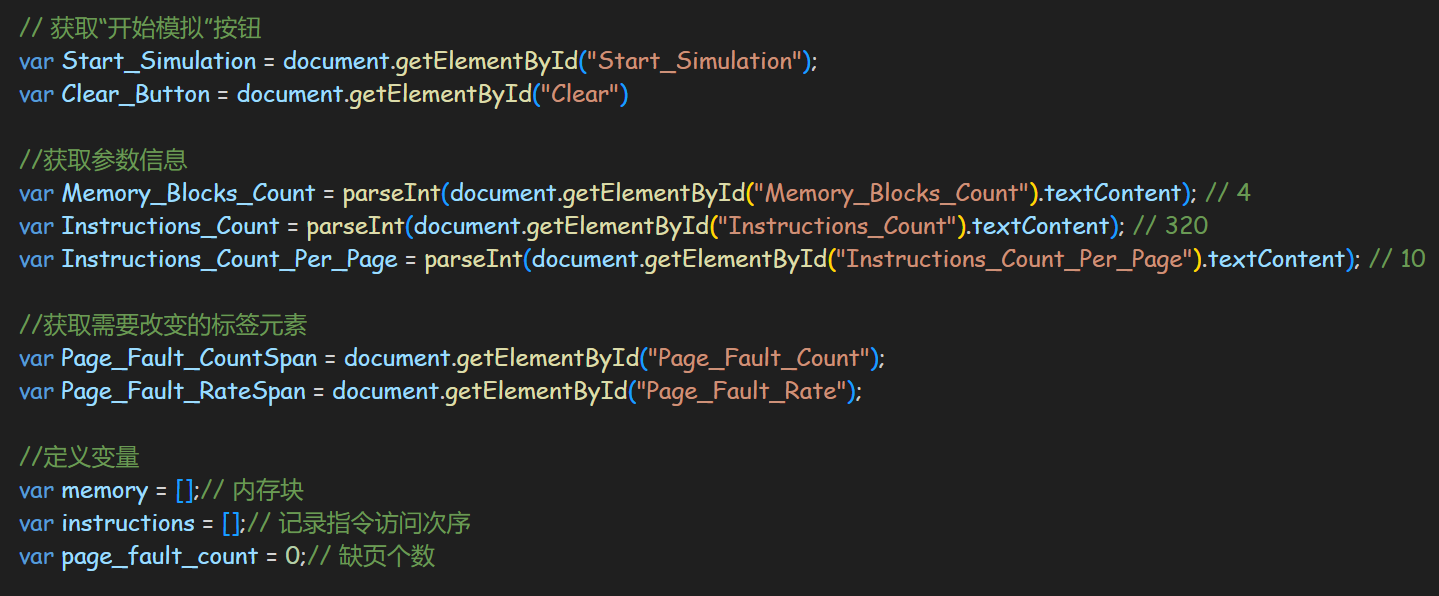
* 方式一：点击”Clear”清空页面，然后重新选择换页算法并点击”Click to Start”运行。
* 方式二：直接重新选择换页算法并点击”Click to Start”运行。

1. **模拟结果**



1. **项目实现**

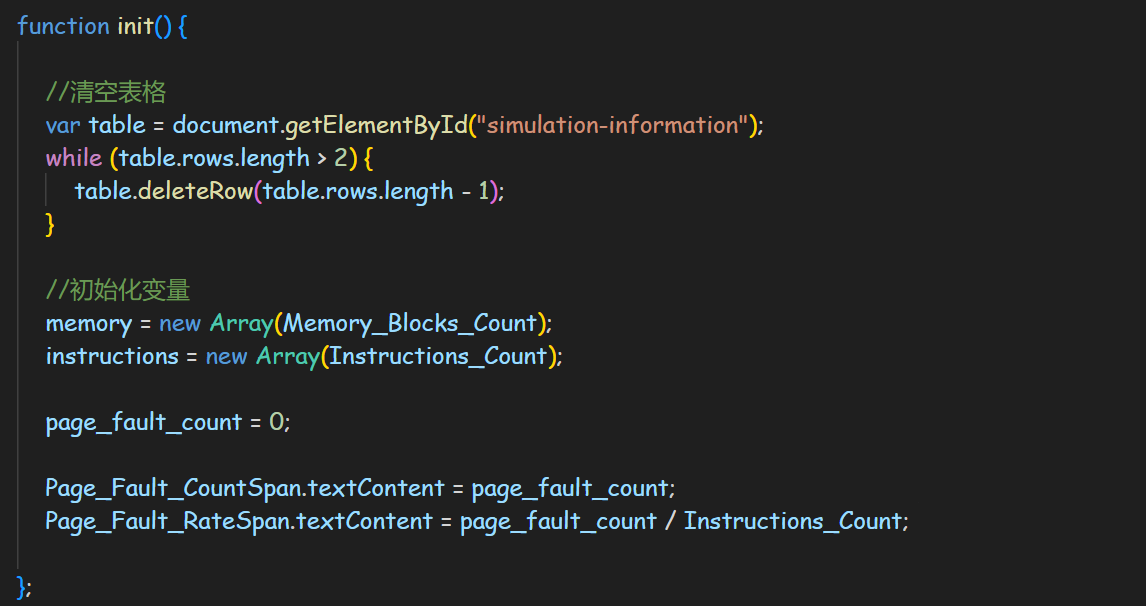
**5.1 全局变量**



**5.2 函数**

**5.2.1 init函数**

init函数用于模拟的初始化，包括清空页面中的模拟信息和初始化全局变量。



**5.4.2 generate\_instructions 函数**

generate\_instructions函数用于随机生成指令访问次序，

**生成原则**：

① 50%的指令是顺序执行的

② 25%是均匀分布在前地址部分

③ 25％是均匀分布在后地址部分。

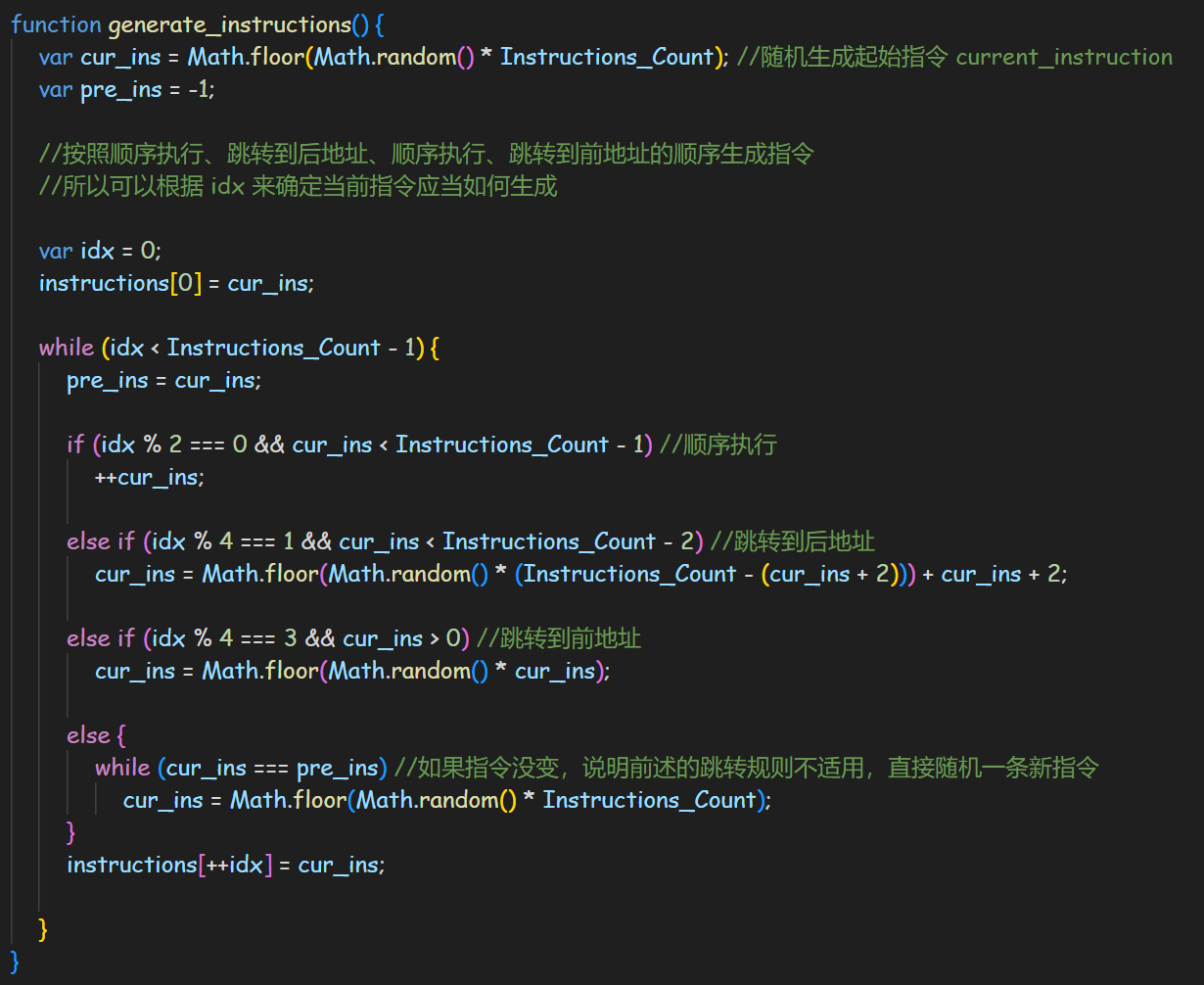
**具体实施方法**：

① 随机选取一个起始执行指令

② 依次按照顺序执行、跳转至前地址、顺序执行、跳转至前地址的顺序确定下一条指令。其中跳转的指令都是通过随机数生成跳转指令。

③ 直至生成320条指令。

生成的指令访问序列存入instructions[]中。



**5.4.3 execute\_simulation 函数**

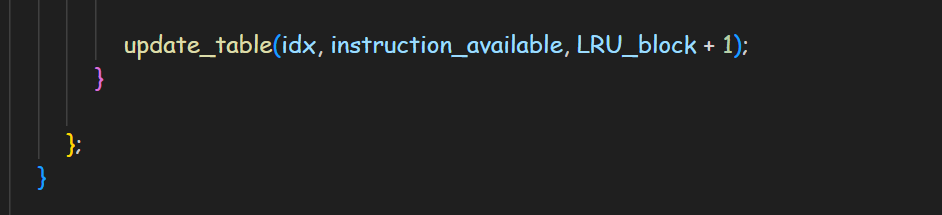
execute\_simulation函数用于执行请求调页内存管理方式的模拟。

首先获取对换页算法的选择，然后按照此前生成的指令访问序列依次访问指令，对于每一个，执行：检查指令是否在内存中，若存在则更新页面中的内容，显示所在内存快好；若不存在则需要调页。

FIFO算法的实现：一直按照内存块1~内存块4的顺序装入即可。

LRU算法的实现：记录一个vis\_seq数组，为访问顺序，靠近末尾的为最近访问的。每次调入vis\_seq[0]的内存块中，即为最近最久未使用的内存块。每次访问指令后，需要将指令所在的内存块设置为最新访问的块，即在vis\_seq数组中移至末尾。

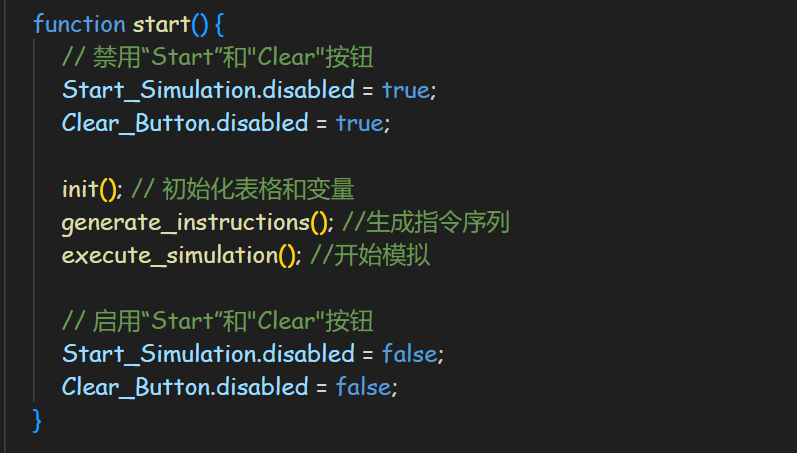




**5.4.4 start 函数**

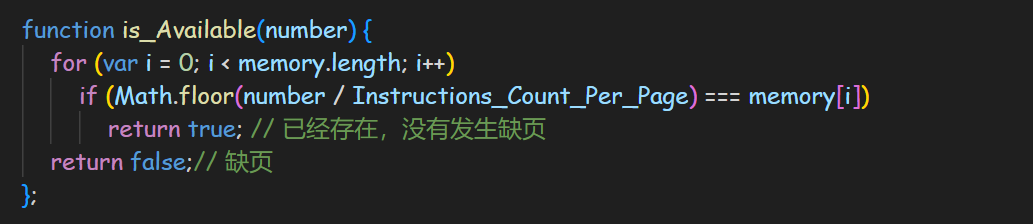
当“Click to Start”按钮被点击后，会执行start()函数：

1. 首先禁用Start与Clear按钮，避免按钮点击对模拟过程造成干扰。
2. 调用init()函数来初始化页面与变量。
3. 调用generate\_instructions()函数来按照指定原则随机生成指令访问序列。
4. 调用execute\_simulation()函数来模拟请求调页内存管理方式，并显示结果。
5. 启用Start与Clear按钮，以便进行下一次模拟。



**5.4.5 is\_Available 函数**

Is\_Available(number)函数用于判断指令number是否在内存中。



**5.4.6 update\_table函数**

该函数用于跟新页面中用于显示模拟信息的表格。



1. **项目总结与心得**

通过这次项目实现，我对请求调页的内存管理方式的认识更加深刻，对FIFO和LRU两种页面置换算法的理解也更加清晰。

此外，通过这个项目，我第一次接触、学习并尝试使用HTML、CSS和JavaScript来构建网页的前端和后端。通过编写HTML页面结构、应用CSS样式以及使用JavaScript实现交互和逻辑，我成功地将内存管理模拟器呈现为一个功能完善的网页应用。这个经历让我熟悉了前端和后端开发的基本概念，并培养了我的网页开发技能。

总的来说，这个项目带给我很多收获。我不仅加深了对内存管理方式的理解，还学到了如何运用前端和后端技术构建一个实用的网页应用。我期待将这些学到的知识和经验应用到未来的项目中，并不断提升自己在软件开发领域的能力。