223|3407 王飞阳

编译第十次作业.

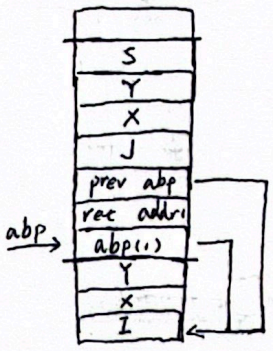
6-1:

1. 抽象类. 线性表

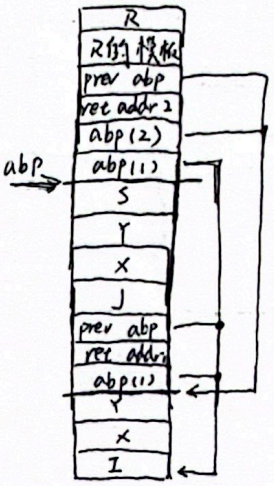
6-2:

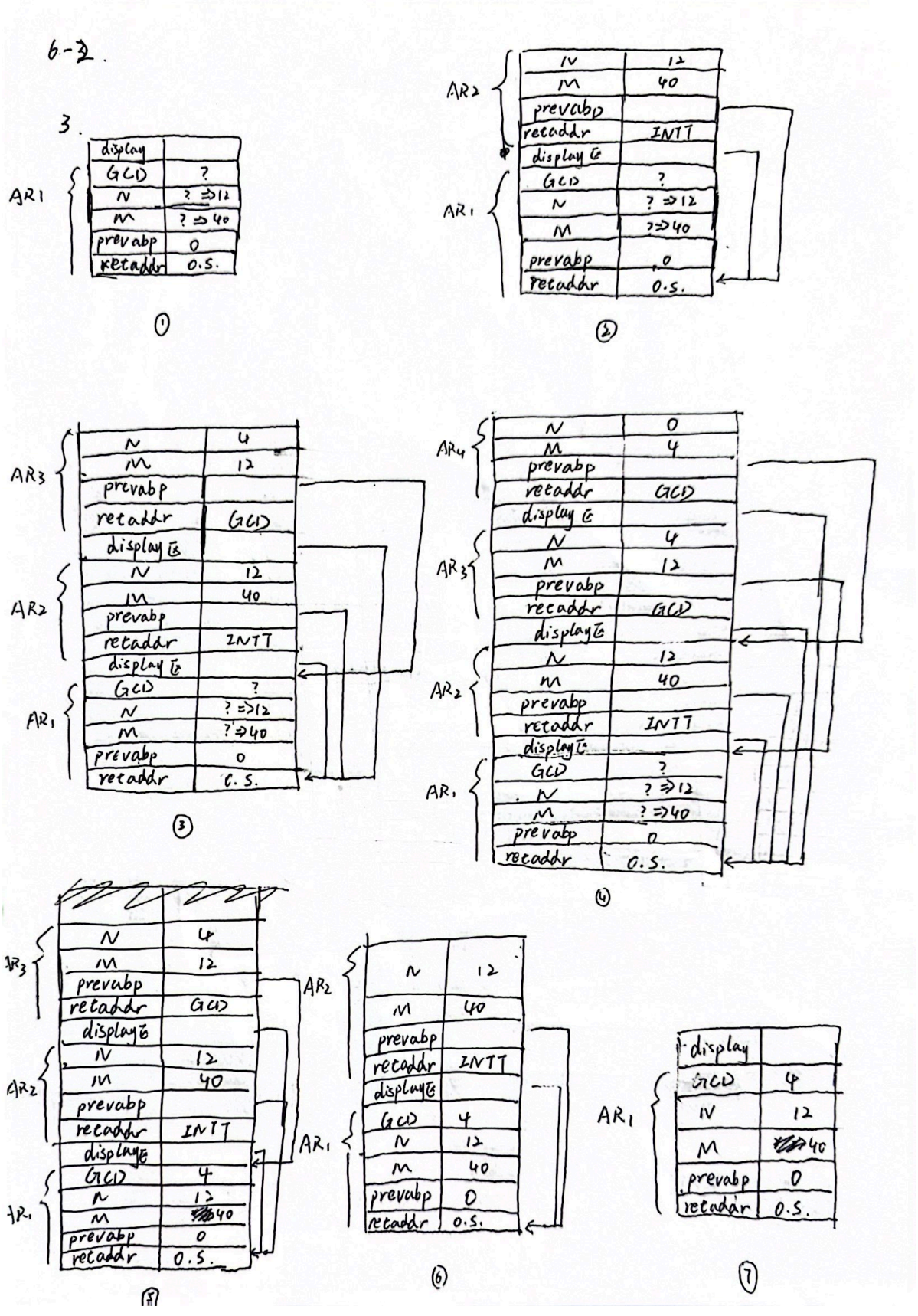
2.

执行到点①



执行到点②:





补充题：

带 display 区的优点：

1. **加速非局部变量访问**：在嵌套函数或嵌套作用域中，非局部变量的访问可能需要遍历多层的链式结构。带有 *display* 区的设计可以通过提供一组直接指针来快速访问外层作用域中的变量，避免了链式遍历的开销。

2. **更高的执行效率**：相较于动态链（dynamic link）或者静态链（static link）的实现，display 区可以减少作用域查找的时间复杂度。尤其是在有大量嵌套的函数或块作用域时，这种优化更加明显。
3. **栈帧管理**：display 区帮助编译器和运行时系统在处理递归调用时管理栈帧，确保在嵌套函数中可以快速、正确地找到外层函数的上下文。

带 display 区的缺点：

1. **额外的内存开销**：每次过程调用时，必须维护 *display* 区的数据结构。由于 *display* 区通常是一个指向外层栈帧的指针数组，这就意味着每次调用都要为该数组分配和管理内存，这在嵌套层次较深时，可能会引发较大的内存消耗。
2. **线程不安全**：在多线程环境中，如果多个线程同时访问同一个 *display* 区，可能会导致数据竞争，产生意外行为。因为不同线程可能会通过同一个 display 访问不同作用域中的变量，这样就需要额外的同步机制来确保正确性。
3. **只适用于静态作用域解析**：带 *display* 区的设计适用于静态作用域（lexical scope）的语言，尤其是在嵌套函数调用时。然而，动态作用域（dynamic scope）的语言无法使用这种技术，因为动态作用域解析依赖于运行时的调用链，而非编译时的作用域链。
4. **递归调用时的性能影响**：在深度递归调用中，*display* 区需要频繁更新栈帧指针的集合，这会带来额外的时间和空间开销，特别是在高度递归的算法中（例如深度优先搜索、数学递归计算等）

主流编程语言和编译系统中的 display 设计现状：

随着编程语言和编译技术的进步，带 display 区的设计逐渐被淘汰或变得不再必要，特别是在现代的优化编译器中。大多数现代编程语言采用了更高效的内存管理、闭包优化或内联函数处理机制，取代了早期的显示区概念。然而，某些编译器和语言在处理特定类型的嵌套作用域时，仍可能引入类似的优化机制。

以下是一些曾经或仍可能使用带 display 区设计的编程语言和编译系统：

1. **Pascal 和 Ada**：
   - *Pascal* 和 *Ada* 是使用 display 区的著名编程语言，尤其是在处理嵌套过程调用和作用域访问时。这些语言在处理静态嵌套时，需要频繁访问外层作用域的变量，因此 display 区的优化能提高性能。
   - 一些旧的 Pascal 编译器，特别是早期的 Turbo Pascal 和 Borland Pascal 版本，采用了 display 区设计来优化栈帧管理。
2. **Algol 系语言**：
   - 作为现代编程语言的祖先，*Algol* 系语言（如 Algol 60）广泛使用 display 区来优化嵌套过程的调用和静态作用域的变量查找。
   - *Algol* 的影响使得后续一些语言在早期设计中借鉴了类似的概念。
3. **Lisp 和 Scheme（早期实现）**：
   - 早期的一些 Lisp 语言实现，如 *Scheme*，在处理嵌套闭包时，也曾考虑使用类似 display 的设计来优化对闭包外层环境的访问。然而，随着垃圾回收和闭包优化技术的发展，这种设计逐渐被淘汰。
4. **Fortran（早期实现）**：
   - 在某些老版本的 *Fortran* 编译器中，也曾引入类似 display 区的机制，以优化在递归调用或复杂计算中的变量访问。