5.4.

```
DSEG SEGMENT
  STRING1  DB  'a'
  STRING2  DB  'ab'
  MAT      DB  'MATCH',0DH,0AH,'$'
  NM       DB  'NO MATCH',0DH,0AH'$'
DSEG ENDS
;
CSEG SEGMENT
  main proc far
       ASSUME CS:CSEG,DS:DSEG,ES:DSEG
start: PUSH DS
       SUB  AX,AX
       PUSH AX
       MOV  AX,DSEG      ;获取数据段地址
       MOV  DS,AX        ;给DS赋值
       MOV  ES,AX
;
BEGIN: LEA  SI,STRING1
       LEA  DI,STRING2
       CLD
       MOV  CX,STRING2-STRING1
       REPE CMPSB        ;字符串依次比较
       JNE  NO           ;不相等时跳转
       LEA  DX,MAT       ;匹配时获取MATCH
       JMP  DISP         ;输出MATCH
```

```
NO:  LEA  DX,NM   ;取取NO MATCH
DISP: MOV AH,9
      INT 21H
      RET         ;输出对应的字符串

main  ENDP
CSEG  ENDS
      END START
```

5.9

```
CODE SEGMENT
     ASSUME CS:CODE
START: MOV  BX,0
       MOV  CH,4
       MOV  CL,4
INPUT: SHL  BX,CL
       MOV  AH,1
       INT  21H
       CMP  AL,30H
       JB   INPUT    ;<0重新输入
       CMP  AL,39H
       JA   AF
       AND  AL,0FH   ;将0-9变为二进制
       JMP  BIN
AF:    AND  AL,1101 1111B ;字母变为大写
       CMP  AL,41H
       JB   INPUT    ;对于A重新输入
```

```
        CMP     AL,46H
        JA      INPUT           ;不正，也要重新输入
        AND     AL,0FH          ;将A-F变为二进制数
        ADD     AL,9
BLN:    OR      BL,AL           ;将输入的数组合在一起
        DEL     CH
        JNZ     INPUT
DISPN:  MOV     CX,16           ;将每一位数转换成ASCII码
DISP:   MOV     DL,0
        ROL     BX,1
        RCL     DL,1
        OR      DL,30H
        MOV     AH,2
        INT     21H             ;输出结果
        LOOP    DISP
STOP:   RET
CODE    ENDS
END     START
```

5.17

```
DSEG  SEGMENT
  MEM  DB  4 DUP(?)
  N    DW  2A49H
DSEG  ENDS
;
CSEG  SEGMENT
 MAIN  PROC FAR
       ASSUME CS:CSEG, DS:DSEG
START: PUSHS
       SUB   AX,AX
       PUSH  AX
       MOV   AX, DSEG
       MOV   DS, AX      ;设置DS的值
BEGIN: MOV   CH, 4
       MOV   CL, 4
       MOV   AX, N
       LEA   BX, MEM
ROTATE: MOV  DL, AL      ;从低位开始转换
       AND   DL, 30H
       CMP   DL, 3AH
       JL    NEXT       ; 0-9则跳转
       ADD   DL, 07H    ; A-F
NEXT:  MOV   [BX], DL    ;将ASCII存入内存
       INC   BX
       ROR   AX, CL      ;移动到下一位
       DEC   CH
```

```
        JNZ   ROTATE
        RET
MAIN    ENDP
CSEG    ENDS
;
END  START
```

5.21

```
DESG  SEGMENT
      ARRAY  DW 3 DUP(0?)
;
CESG  SEGMENT
 MAIN  PROC FAR
       ASSUME CS:CSEG, DS:DSEG
START: PUSH DS
       SUB  AX,AX    ;AX置0
       PUSH AX
       MOV  Ax,DSEG
       MOV  DS,AX    ;给DS赋值
BEGIN: LEA  SI,ARRAY
       MOV  DX,0
       MOV  AX,[SI]
       MOV  BX,[SI+2]
       CMP  AX,BX    ;比较1,2
       JNE  NEXT1
```

```
    INC   DX          ;两个相等,加1
NEXT1: CMP  [SI+4], AX    ;比较1,3
    JNE   NEXT2       ;不相等,跳转
    INC   DX          ;相等再加1
NEXT2: CMP [SI+4]BX,  ;比较2,3
    JNE   NUM
    INC   DX          ;相等时再加1
NUM: CMP  DX,3
    JL   DISP
    DEC   DX          ;都相等时设置为2
DISP: ADD  DL,30H     ;数换为ASCII
    MOV  AH,2         ;输出字符
    INT  21H
    RET
MAIN  ENDP
CSEG  ENDS
;   END START
```

题目 1

向量处理器是设计用于高效处理向量运算的处理器，通常支持单指令多数据. 通过谓词执行，可以在不中断向量运算的情况下处理向量中的部分元素。例如，在一个向量加法操作中，只有当谓词为真时，对应的元素才会被加入到结果向量中，否则会被跳过。而谓词执行则可以直接通过谓词寄存器来实现这一功能，简化了代码并且提高了效率。

Intel 的 Itanium 架构支持谓词执行，IA-64 引入了先进的指令集特性，包括显式的谓词寄存器，以支持高效的谓词执行。在 Itanium 中，每个指令都可以关联一个谓词，这样就可以有条件地执行该指令，而不必改变控制流。这有助于保持长流水线满载运行，并减少分支预测错误带来的性能损失。

题目 2

VLIW 架构依赖于一个智能的编译器来识别并行操作，并将它们打包成单条指令。这要求编译器能够准确地进行静态调度，以避免数据相关性和控制流问题。编译器的设计变得相当复杂，需要大量的资源来开发和维护。

VLIW 架构对于指令的打包有特定的要求，这可能导致它与其他非 VLIW 架构的二进制代码不兼容。

题目 3

java 不一定比 c 语言慢。人们认为 Java 等虚拟机语言较慢，主要是因为早期版本确实存在显著的性能差距，而且这种印象在开发者社区中流传已久。随着技术的发展，这一差距已经大大缩小，但在某些高性能计算或对实时性要求极高的领域，C/C++仍然可能是更好的选择。不过，对于大多数商业软件来说，Java 等语言提供的便利性和生产力优势通常足以弥补潜在的性能损失。

题目 4

全局寄存器着色算法能够产生高质量的寄存器分配结果，因为它考虑了整个函数或程序范围内的信息，从而可以做出更优的决策。然而，这种算法的缺点在于其复杂性和较高的计算开销，尤其是在处理大规模程序时。

线性扫描算法运行速度快，可以在单次线性扫描过程中完成寄存器分配。相比图着色算法，线性扫描算法的实现较为简单，不需要复杂的图论知识和数据结构支持，这降低了编译器的开发和维护成本。