## 汇编语言课程项目

**————视频播放器实验报告**

2251334 倪朗恩

## 项目简介

使用python对普遍格式视频预处理成文本文件后，通过8086程序播放4fps的视音频，无法中间退出以及停止，播放完后按键退出。图像使用VGA模式进行缓冲区以及调色板刷新4fps，音频使用定时器中断3fps（极差）。图像以及音频文件播放帧数以及文件路径可在8086文件数据段调整，实际文件数在python中定义。

## 项目目标

1. **将视频进行格式转换，实现8086可读入**
2. **编写8086视频播放器（无终止，按键退出）**

## 视频处理

### 设计

已知8086的一般段大小在64KB，超过容易导致内存溢出问题，同时，考虑到视频通常较大，使用更改文件指针的方法可能导致寄存器溢出，故将视频图像按所需帧率处理，每一帧一个文件；音频大小取决于采样大小，选择牺牲音频（硬件不适应），故将音频文件单独提出。

### 具体实现（python）

#### 图像

import cv2

import numpy as np

from sklearn.cluster import KMeans

from collections import Counter

# 为了给每个像素找到最接近的颜色，计算每个像素到聚类中心的距离

def get\_nearest\_color(pixel, centers):

    distances = np.linalg.norm(centers - pixel, axis=1)

    nearest\_color\_idx = np.argmin(distances)

    return centers[nearest\_color\_idx], nearest\_color\_idx

if \_\_name\_\_ == '\_\_main\_\_':

    # 视频文件路径

    video\_path = 'Never Gonna Give You Up.mp4'

    # 打开视频文件

    cap = cv2.VideoCapture(video\_path)

    # 获取视频的帧率（fps）

    fps = cap.get(cv2.CAP\_PROP\_FPS)

    # 获取视频的总帧数

    frame\_count = int(cap.get(cv2.CAP\_PROP\_FRAME\_COUNT))

    # 设置截取间隔为每秒一帧

    frame\_interval = int(fps) // 4

    # 用于存储每秒截取的画面颜色数据

    colors = []

    all\_pixels = []

    frame\_number = 0

    # 打开视频文件

    cap = cv2.VideoCapture(video\_path)

    # 循环遍历视频的每一帧

    frame\_number = 0

    while frame\_number <= 848 \* frame\_interval:

        ret, frame = cap.read()

        if not ret:

            break

        # 每隔一秒截取一个画面

        if frame\_number % frame\_interval == 0:

            # 将画面缩放到 320x200

            resized\_frame = cv2.resize(frame, (320, 200))

            all\_pixels = resized\_frame.reshape(-1, 3)

            # 使用KMeans聚类算法提取224种颜色

            kmeans = KMeans(n\_clusters = 224, random\_state=42)

            kmeans.fit(all\_pixels)

            # 获取聚类中心（即最常见的颜色）

            dominant\_colors = kmeans.cluster\_centers\_

            # 获取每个像素所属的簇标签

            labels = kmeans.labels\_

            # 将每个像素替换为最接近的颜色，并获取索引

            recolored\_pixels = []

            indices = []

            for pixel in all\_pixels:

                color, idx = get\_nearest\_color(pixel, dominant\_colors)

                recolored\_pixels.append(color)

                indices.append(idx)

            # 将重新着色的像素数据重塑回原图像形状

            recolored\_image = np.array(recolored\_pixels).reshape(resized\_frame.shape)

            filename = "pic" + str(frame\_number // frame\_interval).zfill(3) + ".txt"

            # 写入文件

            with open(filename, 'wb') as f:

                # 写入 dominant\_colors

                for dominant\_color in dominant\_colors:

                    dominant\_color[0] = dominant\_color[0] \* 63.0 / 255.0

                    dominant\_color[1] = dominant\_color[1] \* 63.0 / 255.0

                    dominant\_color[2] = dominant\_color[2] \* 63.0 / 255.0

                    temp = np.array(dominant\_color, dtype=np.uint8)

                    f.write(temp[2].tobytes())

                    f.write(temp[1].tobytes())

                    f.write(temp[0].tobytes())

                # 写入 indices

                for idx in indices:

                    f.write(np.uint8(idx+32).tobytes())

            print(frame\_number // frame\_interval)

        frame\_number += 1

    # 释放视频文件

    cap.release()

使用聚类方法，4帧每秒（保证视频质量），对视频切片处理。

由于8086的VGA模式下，rgb是3\*6位数据表示，故写入时先预处理然后按字节写入；之后根据像素顺序，按字节写入根据聚类得到的图像索引列表。

使用224聚类是因为8086的VGA模式下，有256色的调色板，图像颜色根据调色板索引获取，其中32-255为自定义色区。

#### 音频

##### PIP\_OP

import moviepy as mp

import numpy as np

import librosa

import librosa.display

import numpy as np

import matplotlib.pyplot as plt

if \_\_name\_\_ == '\_\_main\_\_':

    # # 提取视频中的音频

    # video\_path = 'Never Gonna Give You Up.mp4'

    # video = mp.VideoFileClip(video\_path)

    # # 提取音频并保存为文件

    # audio = video.audio

    # audio\_path = 'extracted\_audio.wav'

    # audio.write\_audiofile(audio\_path)

    # 加载音频文件

    audio\_path = 'extracted\_audio.wav'

    y, sr = librosa.load(audio\_path, sr=None)

    # 每秒采样6次，获取每1/6秒的频率

    sampling\_points = 1200

    # 计算短时傅里叶变换（STFT）

    D = np.abs(librosa.stft(y))

    # 获取频率和时间轴

    frequencies = librosa.fft\_frequencies(sr=sr)

    times = librosa.frames\_to\_time(np.arange(D.shape[-1]), sr=sr)

    # 每1/6秒提取频率信息

    frequencies\_6\_per\_second = []

    for i in range(0, len(times), len(times) // sampling\_points):

        # 获取当前时间点的频谱数据

        frame\_freqs = D[:, i]  # 当前时间点的频率成分（幅度谱）

        # 获取幅度最大的频率

        max\_freq\_idx = np.argmax(frame\_freqs)  # 找到幅度最大的频率索引

        dominant\_freq = frequencies[max\_freq\_idx]  # 对应的频率就是主频

        temp = 1193180 // dominant\_freq

        frequencies\_6\_per\_second.append(np.int16(temp).tobytes())

        print(f"Time: {times[i]:.2f}s, Dominant Frequency: {dominant\_freq:.2f}Hz")

    # 输出每1/6秒的频率信息

    print("Frequencies per 1/6 second:", frequencies\_6\_per\_second)

    print(len(frequencies\_6\_per\_second))

    filename = "sound.txt"

    # 写入文件

    with open(filename, 'wb') as f:

        # 写入 dominant\_colors

        for i in range(len(frequencies\_6\_per\_second)):

            if i < 1200:

                f.write(frequencies\_6\_per\_second[i])

            else:

                break

先将音频从图像中分离出来，然后根据采样密度（3fps）获取频率值，将每个2byte频率值进行转换1193180//dominant\_freq，因为8086中使用定时器2进行音频输出，其中输出到端口的2字节值作为除数，与定时器2频率相除获取频率（选择方波，因为不考虑质量）。

#### video.asm

##### Main

主函数，实现播放器功能。

具体实现：

1. 读取音频文件后，设置定时器，改变定时中断程序指向，使其调用Timer\_ISR函数
2. 改变图像模式
3. 读取帧文件并更新、延迟
4. 恢复

MAIN PROC FAR

    ; 初始化数据段

    mov ax, DATASEG

    mov ds, ax

    mov es, ax

    ; lea dx, color

    ; lea dx, color\_buffer

    ; lea dx, frame

    ; lea dx, frame\_buffer

    mov cx, 2400

    lea dx, sound\_buffer

    mov bx, dx

    lea dx, sound

    call read

    ; 安装中断处理程序

    ; 将中断处理程序的地址设置为 0x08h (定时器中断)

    cli                      ; 禁用中断

    lea dx, Timer\_ISR          ; 获取 Timer\_ISR 偏移地址

    mov ax, 08h                ; 定时器中断向量索引为 0x08

    mov cl, 2

    shl ax, cl                 ; 每个向量占用 4 字节，乘以 4

    mov bx, ax                 ; BX 存储中断向量表的位置

    mov word ptr [bx], ds        ; 设置段地址

    add bx, 2

    mov word ptr [bx], dx        ; 设置偏移地址

    sti                      ; 启用中断

    ; 设置定时器中断处理程序

    ; 初始化定时器

    mov al, 36h                ; 设置计数器 0 的工作模式 (0011 0110)

    out 43h, al                ; 向 43h 端口发送命令（选择计数器 0）

    mov ax, 0ffffh            ; 1193180 / 1000h = 18.2

    out 40h, al

    mov al, ah

    out 40h, al

    ; 切换到 VGA 13h 模式（640×480 256 色模式）

    mov ax, 13h

    int 10h

    ; 处理文件读取和显示

    mov cx, [count]  ; count 帧

    mov ax, PICTURE

    mov es, ax

frame\_loop:

    call update\_filename

    ; 打开 data 文件夹下的 pic???.txt 文件

    push cx

    mov cx, 64672

    lea dx, frame\_buffer

    mov bx, dx

    lea dx, frame

    call read

    pop cx

    call display

    ; 延迟控制帧率

    call delay

    loop frame\_loop       ; 处理下一帧

    mov al, 30h       ; 0x30: 选择计时器 2，设置为中断触发模式（停止状态）

    out 43h, al       ; 发送控制字节到 PIT 控制端口 0x43

    ; 等待用户按键退出

    mov ah, 0

    int 16h

    ; 恢复文本模式

    mov ax, 03h

    int 10h

    ; 退出程序

    mov ah, 4Ch

    int 21h

MAIN ENDP

##### Read

文件读取函数，自动读取并关闭，可以指定读取内容上限。

参数：

1. Ds为读取文件名称所在段地址
2. Es为存放读取数据的缓冲区所在段地址
3. CX说明读取文件大小

; 参数在stack里，顺序为：文件名地址，缓冲区地址

; cx 保存文件大小, dx保存文件位置, bx保存缓冲区位置

read proc

    push ax

    push ds

    ; 打开 data 文件夹下的文件

    mov ah, 3Dh           ; INT 21h 打开文件功能

    mov al, 0             ; 只读模式

    int 21h               ; 调用 DOS 中断

    mov dx, bx

    mov bx, ax            ; 保存文件句柄

    jc  read\_error

    ; 读取文件内容

    mov ax, es

    mov ds, ax

    mov ah, 3Fh           ; INT 21h 读取文件功能

    int 21h               ; 调用 DOS 中断

    jc  read\_error

    ; 关闭文件

    pop ds

    mov ah, 3Eh           ; INT 21h 关闭文件功能

    int 21h               ; 调用 DOS 中断

read\_error:

    pop ax

    ret

read endp

##### main

更新帧读取文件名，与帧保存位置以及命名格式绑定。

参数：

1. CX在外循环，内部自动处理

; 更新文件名

update\_filename proc

    push ax

    push bx

    push dx

    ; 设置文件名为 frame\_nnn.txt

    ; nnn 为文件编号，存储在 ah，al，bh 中

    mov ax, [count]

    sub ax, cx

    mov bx, 10

    xor dx, dx

    div bx

    push dx

    xor dx, dx

    div bx

    mov ah, al

    mov al, dl

    pop dx

    mov bh, dl

    add ah, '0'           ; 转换为字符

    add al, '0'

    add bh, '0'

    mov frame[8], ah  ; 将字符放入文件名位置

    mov frame[9], al

    mov frame[10], bh

    pop dx

    pop bx

    pop ax

    ret

update\_filename endp

##### Display

播放播放，刷新屏幕，与帧文件格式绑定。

参数：

1. Es保存帧内容缓冲区所在段

; 显示图像

display proc

    push ax

    push bx

    push cx

    push dx

    push si

    push ds

    xor bx, bx

    mov al, 32            ; 设置第一个颜色

    mov dx, 03C8h

    out dx, al        ; 将调色板索引输出到 0x03C8

    mov cx, 224            ; 我们设置后 224 个颜色

    mov dx, 03C9h

set\_color\_loop:

    mov al, es:[bx]         ; 读取红色值

    out dx, al        ; 设置红色

    inc bx

    mov al, es:[bx]         ; 读取绿色值

    out dx, al        ; 设置绿色

    inc bx

    mov al, es:[bx]         ; 读取蓝色值

    out dx, al        ; 设置蓝色

    inc bx

    loop set\_color\_loop

   ; 显示图像

   xor si, si

   mov ax, 0A000h        ; VGA 显存地址

   mov ds, ax

   mov cx, 64000         ; 每次显示 64000 个像素

display\_loop:

   mov al, es:[bx]          ; 从缓冲区读取一个像素

   mov ds:[si], al       ; 将像素写入 VGA 显存

   inc bx                ; 缓冲区指针自增

   inc si

   loop display\_loop     ; 循环直到显示完所有像素

    pop ds

    pop si

    pop dx

    pop cx

    pop bx

    pop ax

    ret

display endp

##### Timer\_ISR

定时器中断函数，实现定时调用播放音频。（跟main关联，播放频率为3fps，使用定时器0）

参数：

; 定时器中断服务程序,控制声音输出

Timer\_ISR proc

    ; 保存寄存器状态

    push ax

    push bx

    push cx

    push dx

    push ds

    push si

    push di

    mov ax, [pos]

    xor dx, dx

    mov bx, 3

    div bx

    cmp dx, 0

    jnz no\_action

    cmp ax, 1200

    jge no\_action

    mov bx, 2

    mul bx

    ; 播放简单的音效（使用 PC Speaker）

    call play\_sound

no\_action:

    ; 重新加载计数器，以便下次定时器中断仍然触发

    mov ax, 0ffffh          ; 重新设置计数器值

    out 40h, al             ; 低字节 (0xFF)

    mov al, ah

    out 40h, al             ; 高字节 (0xFF)

    mov ax, [pos]

    inc ax

    mov [pos], ax

    ; 发送 EOI 信号

    mov al, 20h             ; 设置 EOI 命令

    out 20h, al             ; 发送 EOI 信号到 PIC

    ; 恢复寄存器状态

    pop di

    pop si

    pop ds

    pop dx

    pop cx

    pop bx

    pop ax

    ; 中断返回

    iret

Timer\_ISR endp

##### Play\_sound

音频播放函数，实际设置频率。（嵌套在使用定时器2）

参数：

1. AX指向sound\_buffer中播放的位置，由Timer\_ISR处理

; 波音，ax为指针

play\_sound proc

    push ax

    push bx

    push ds

    ; 播放简单的音效（使用 PC Speaker）

    mov bx, ax

    mov ax, DATASEG

    mov ds, ax

    ; 设置频率

    mov al, 36h           ; 控制字节 0x36 -> 设置计时器 2（用于音频），16 位计数器，模式 3（方波）

    out 43h, al           ; 发送到音频控制寄存器

    ; 音频频率 = 时钟频率 / (频率计数器值 + 1)，通常时钟频率为 1193180 Hz

    mov al, 00ffh

    mov al, sound\_buffer[bx]   ; 低字节

    out 42h, al            ; 发送低字节到音频数据寄存器 0x42

    INC bx

    mov al, sound\_buffer[bx] ; 高字节

    out 42h, al            ; 发送高字节到音频数据寄存器 0x42

    pop ds

    pop bx

    pop ax

    ret

play\_sound endp

##### main

延迟函数，控制帧率，影响视频整体质量。

参数：

; 延迟

delay proc

    push ax

    push cx

    mov ax, 11

delay\_loop\_big:

    ; 简单的延迟，控制播放速度

    mov cx, [delay\_count]

delay\_loop:

    loop delay\_loop

    dec ax

    jnz delay\_loop\_big

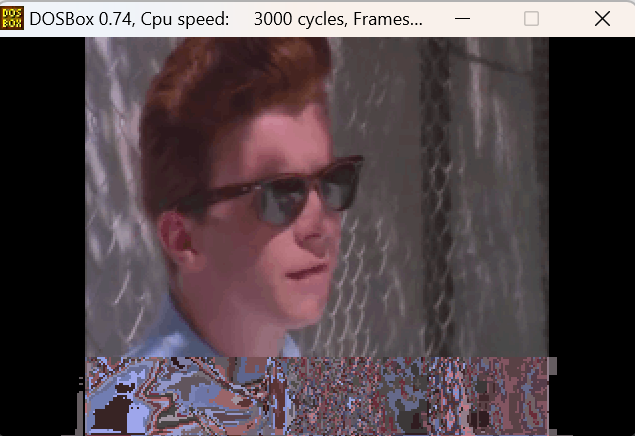
    pop cx

    pop ax

    ret

delay endp

## 实验结果



## 问题与心得

1. 端口

端口分为8位传输和16位传输，根据目前经验所得，端口为1字节为8位，端口为2字节为16位。16位端口需要使用dx存储端口，然后out dx，al，16位端口要先传低位字节，后传高位字节。

1. 延时

通过双循环实现延时（单循环太快），此时要注意参数大小，如果过大可能导致实际为负，进入死循环导致运行失败；太快容易导致画面颜色滞留严重，太慢降低流畅性

未能解决Bug：如果延迟设置太大或者切换ax，cx的参数来源，出现不可解决运行失败。

1. 音频

无法发出声音，经过网上搜证以及电脑设备检查，初步怀疑是硬件不支持导致（没有PCSpeaker）。

1. 段地址

对段地址、偏移地址、lea等8086寻址有了更深入的了解：

A．lea只返回数据相对于自己所在段的偏移地址，需要自己控制段寄存器；

B．数据寻址默认段地址根据assume声明实现。

1. 文件读取

充分了解了学习乐8086的文件读取方式，实现了按需读取以及debug方法。

1. VGA模式

充分学习了VGA模式的显示方法以及调色盘结构和改变方式，知道了256色调色盘以及索引显色方法、6位表示单通道颜色，并制作了简单验色文件seecolor.exe。

Bug：由于硬件速度不匹配，每次刷新时先刷新调色板后刷新画面，导致未刷新的画面变色形成色斑。

可行解决方法：每次刷新修改调色板可变区域的一半的，同时修改元数据格式使相邻两帧之间的颜色索引不相交，可以保证没有色斑（代价是画面可能色质降低，不过考虑到每一帧的颜色大概率不需要完全使用112/128个索引，故影响局限甚至可能不计）。

1. 定时器

学习了8086的定时器使用、定时中断使用，了解到定时器不止能用于定时中断，还用于控制音频的输出。

1. 数据结构

深刻地意识到合理的数据结构的重要性。本来打算将颜色与图片分离，可是视频质量严重下降（颜色不足），故将颜色数据与图像数据结合；通过源代码注释内容可见原先结构。

1. Debug

充分掌握以及深刻地意识到debug的重要性，通过debug解决了大部分的绝大部分问题。