

实验名称 分支与循环程序设计实验 成绩

指导教师 曹丹华

专业班级 光电 1802 姓 名 马笑天 学 号 U201813654

联系电话 18201000620

一、任务要求

1. 设有 8bits 符号数 X 存于外部 RAM 单元,按以下方式计算后的结果 Y 也存于外部 RAM 单元,请按要求编写程序。

$$Y = \begin{cases} X^2 & \text{当 } X \geq 40 \\ X/2 & \text{当 } 20 < X < 40 \\ \overline{X} & \text{当 } X \leq 20 \end{cases}$$

2. 利用 51 系列单片机设计一个 24 小时制电子时钟,电子时钟的时、分、秒数值分别通过 P0、P1、P2 端口输出(以压缩 BCD 码的形式)。P3.0 为低电平时开始计时,为高电平时停止计时。设计 1s 延时子程序(延时误差小于 10us,晶振频率 12MHz)。

提高部分(选做):

- 实现 4 位十进制加、减 1 计数,千位、百位由 P1 口输出;十位、个位由 P2 口输出。利用 P3.7 状态选择加、减计数方式。
- 利用 P3 口低四位状态控制开始和停止计数,控制方式自定。

二、设计思路

1. 设有 8bits 符号数 X 存于外部 RAM 单元,按以下方式计算后的结果 Y 也存于外部 RAM 单元,请按要求编写程序。

$$Y = \begin{cases} X^2 & \text{当 } X \geq 40 \\ X/2 & \text{当 } 20 < X < 40 \\ \overline{X} & \text{当 } X \leq 20 \end{cases}$$

题目要求 8bit 符号数 X, Y 均存于外部 RAM, 所以在使用转移指令时用 MOVX。由于 Y 是关于 X 的分段函数, 考虑采用分支结构进行三段的跳转。当 $X \geq 40$ 时, $Y \geq 1600$; 当 $20 < X < 40$ 且为奇数时, Y 为小数。结合这两点来看, 采用三字节来存放 Y 较为合适, 存放在 1FFFH, 2000H, 2001H, 其中 2001H 为小数部分, 当整数部分超过一个字节时, 就用两个字节存放。要强调的是, 应当对 20 和 40 两个边界条件加以特别的注意。

2. 利用 51 系列单片机设计一个 24 小时制电子时钟, 电子时钟的时、分、秒数值分别通过 P0、P1、P2 端口输出 (以压缩 BCD 码的形式)。P3.0 为低电平时开始计时, 为高电平时停止计时。设计 1s 延时子程序 (延时误差小于 10us, 晶振频率 12MHz)。

先说这个 1s 延时的子程序。采用了 5 层循环, 运用 DJNZ, CJNE 等函数, 循环量参数调了很长时间 (确切的说一晚上), 最后做到了 $1 \mu s$ 的误差, 远小于要求的 $10 \mu s$ 误差, 效果尚可。

再说 24 小时制电子时钟。利用 1s 延时子程序, 可以进行计时。并且需要引入 P3.0 的判断, 当为高电平时一直执行死循环, 一旦为低电平时即跳出循环, 开始计时。进位方面, 秒、分均是满 60 进一、小时则是满 24 连带分秒全部清零, 这些利用条件判断语句加上二进制-十进制转换指令实现。

提高部分 (选做):

a. 实现 4 位十进制加、减 1 计数, 千位、百位由 P1 口输出; 十位、个位由 P2 口输出。利用 P3.7 状态选择加、减计数方式。

b. 利用 P3 口低四位状态控制开始和停止计数, 控制方式自定。

根据题意, P1 为千百、P2 为十个, 我定义了 P3.0 的高电平和低电平分别对应状态的开始和结束, P3.7 的高电平和低电平分别对应了加、减计数, 这控制着程序分别执行加计数与减计数两块不同的程序段。加计数中利用了 ADD A, #1 和 DA A 配合, 减计数中利用了 ADD A, #99H 和 DA A 配合, 并且每次循环一开始就检测是否暂停, 以及是否切换加减计数。

三、资源分配

1. X 存在外部 RAM 1000H, Y 存在外部 RAM 1FFFH, 2000H, 2001H 处, 1FFFH, 2000H 存整数部分, 2001H 存小数部分。A, B, C 等为临时变量。R1, R2 负责自变量 X 的分段比较大小

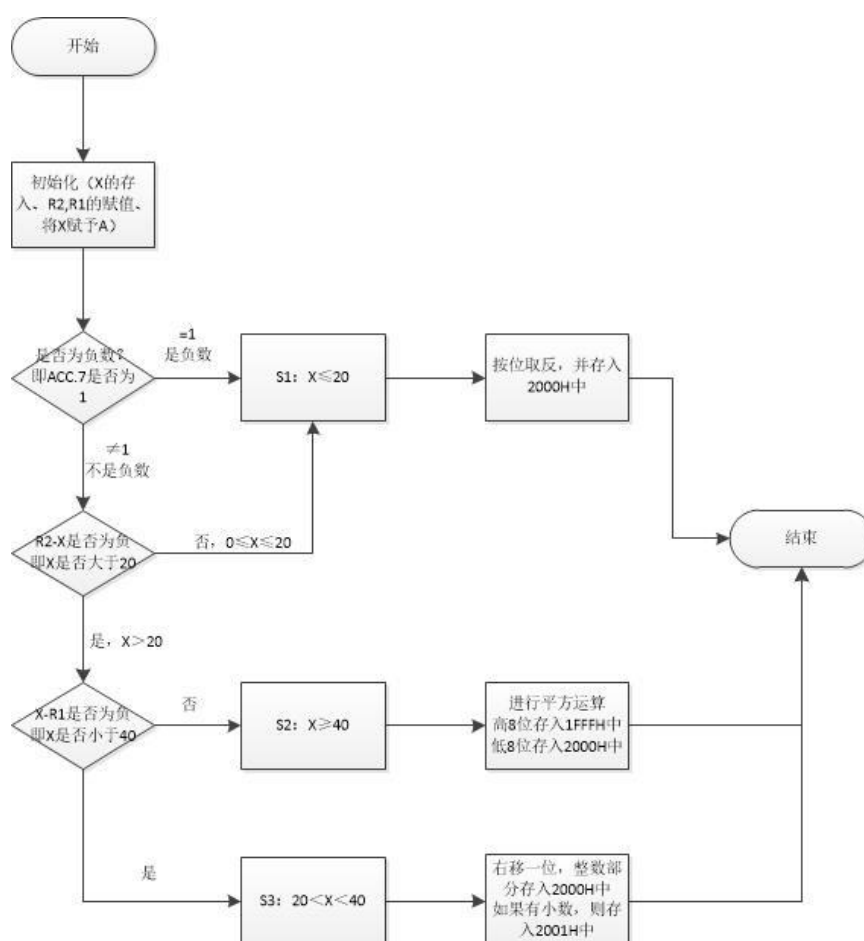
2. P3.0 操作计时开关, P0, P1, P2 分别为时分秒, R0, R1 为中间变量, R3-R7 为 1s 定时程序的控制参量。

提高部分 (选做):

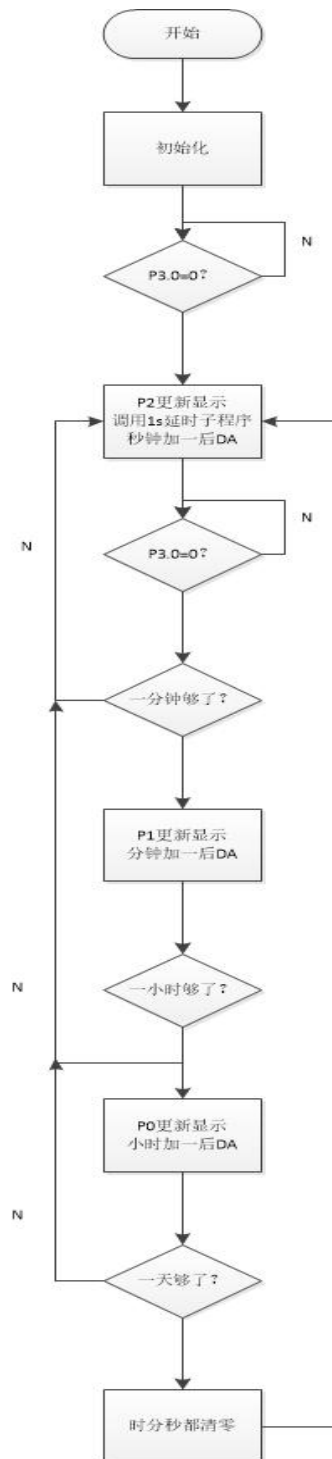
3. a. b. P3.7 定义为开始暂停键, P3.0 定义为加减切换键, P1 为千百, P2 为十个, A, C, R0, R1 为中间变量, R3-R7 为定时程序的控制参量。

四、流程图

1.

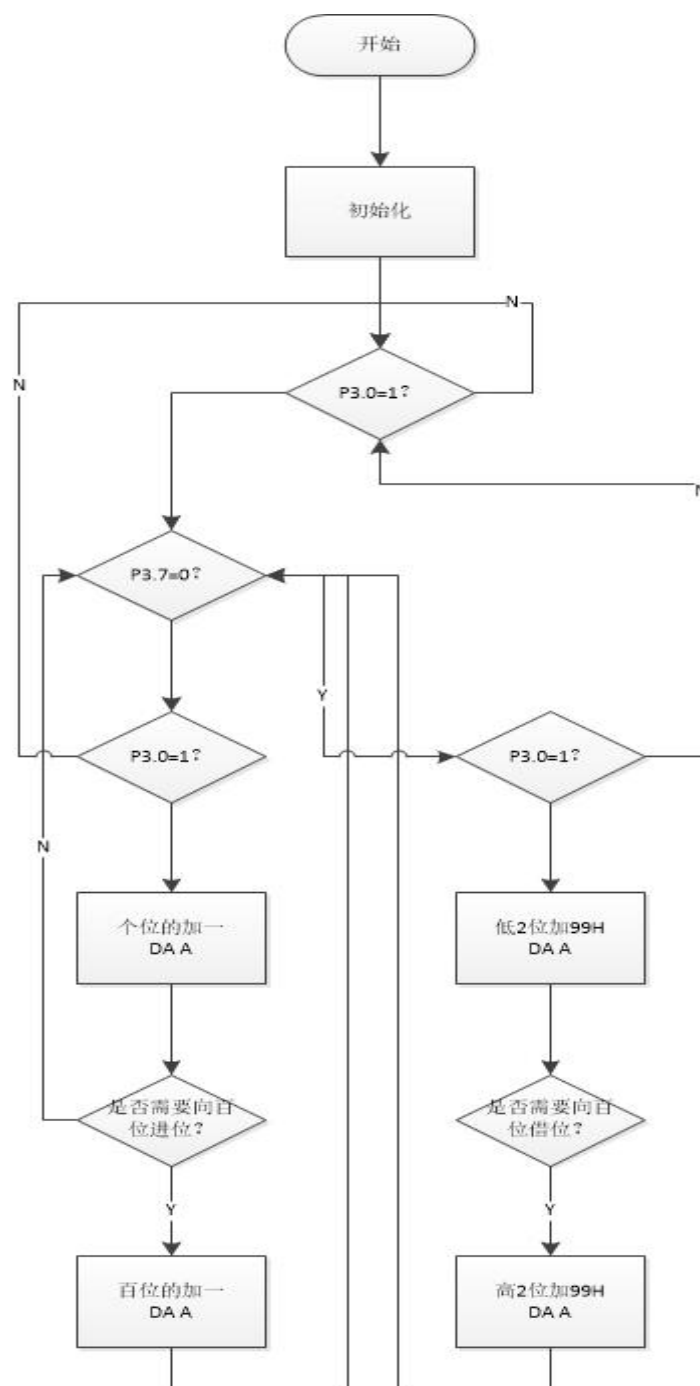


2.



提高部分（选做）：

a. b.



五、源代码（含文件头说明、语句行注释）

1.

;File name: yx.asm

;Description: 分段函数

;Designed by: MA Xiaotian

;Date:2020-4-16

ORG 0000H

LJMP MAIN

X EQU 121

ORG 2000H

;假设 Y 存放在 1FFFH, 2000H, 2001H, 其中 2001H 为小数部分

MAIN: MOV DPTR, #1000H

MOV A, #X

MOVX @DPTR, A ;将 X 存储在外部 RAM1000H 的位置

MOV R2, #20

MOV R1, #40

JB ACC.7, S1 ;如果是负数, 则归到 S1 中

CJNE R2, #X, S1

S1: JC S2 ;如果是大于 20 的数, 就跳出, 另行判断

CPL A

MOV DPTR, #2000H

MOVX @DPTR, A

SJMP \$

S2: CLR C

SUBB A, R1

JC S3

MOV A, #X

中

```
MOV R0, #20H
MOV 20H, #0
XCHD A, @R0      ;设 X=ABH, 则  $X^2 = (AOH)^2 + (OBH)^2 + 2 * (AOH) * (OBH)$ 
SWAP A           ;取 X 的高 4 位
MOV R1, A        ;X 的高 4 位存入 R1 中, 低 4 位存入 R0 所指的片上 RAM20H

MOV B, A
MUL AB           ;X 的高 4 位最大为 7H, 所以乘积最大值
MOV DPTR, #1FFFH ;为 0011 0001B, B 寄存器中为空, A 中结果存入 1FFFH 中
MOVX @DPTR, A
MOV A, @R0
MOV B, A
MUL AB           ;X 的低 4 位最大为 0FH, 所以乘积最大值为
MOV DPTR, #2000H ;为 1111 0001B, B 寄存器中为空, A 中结果存入 2000H 中
MOVX @DPTR, A
MOV A, @R0
MOV B, A
MOV A, R1        ;X 的高 4 位最大为 7H, 低 4 位最大为 0FH, 所以乘积
MUL AB           ;最大值为 0110 1001B, B 寄存器中为空, A 中结果需要
RL A
MOV @R0, #0      ;分别加入到 1FFFH 和 2000H 中, 具体来说, 应将 A 的高 4
SWAP A           ;位作为一个字节的低 4 位加入到 1FFFH 中, 而将 A 的低
XCHD A, @R0      ;4 位作为一个字节的高 4 位, 这个字节的低 4 位均为 0,
MOV R1, A        ;将这个字节加入到 2000H 中
MOVX A, @DPTR
ADD A, R1
MOVX @DPTR, A
MOV DPTR, #1FFFH
MOVX A, @DPTR
ADDC A, @R0
```

```
MOVX @DPTR, A

SJMP $

S3:  MOV A, #X           ;这里处理的是 21 到 39 的数
      RRC A             ;右移除以二
      MOV DPTR, #2000H
      MOVX @DPTR, A
      JNC HERE          ;如果是有小数 0.5, 则存在外部 RAM2001H 里
      INC DPTR
      MOV A, #80H
      MOVX @DPTR, A      ;2001H 内容为 1000 0000B
HERE: SJMP HERE

END
```

2.

```
;File name: TIMER.asm
;Description: 24 小时计时器的应用
;Designed by: MA Xiaotian
;Date:2020-4-16
```

```
ORG 0000H

LJMP START

ORG 0100H

START: MOV P0, #0
       MOV P1, #0
       MOV P2, #0
```



```
MOV R0, #1

MOV R1, #1

CLR A

WAIT:  JB P3.0, WAIT      ;等待 P3.0 为低电平

SEC:   MOV P2, A          ; 显示秒计数值
      ACALL DELAY
      ADD A, #1           ; BCD 码加 1 计数
      DA A
      JB  P3.0, WAIT
      CJNE A, #60H, SEC

MIN:   MOV A, R0
      DA A
      MOV P1, A           ;显示分钟计数值
      INC A
      DA A
      MOV R0, A
      CLR A
      CJNE R0, #61H, SEC  ;判断是否需要进位到小时
      MOV P1, #0         ;进位后的归零和 R0 复位
      MOV R0, #1

      HOUR:
      MOV A, R1
      DA A
      MOV P0, A          ;显示小时计数值
      INC A
      DA A
```

```
MOV R1, A

CLR A

CJNE R1, #25H, SEC      ;判断是否需要清零

MOV P0, #0              ;进位后的归零和 R0/R1 复位

MOV R1, #1

MOV R0, #1

LJMP SEC

;=====DELAY SECTION=====      ;太笨了参数调了一晚上还没调好

DELAY:  MOV R7, #2

DLOOP1:

HERE0:  NOP

        DJNZ R7, DLOOP2

        RET

DLOOP2: MOV R6, #9

HERE1:  MOV R3, #16

HEREX:  DEC R3

        CJNE R3, #0, HEREX

        DJNZ R6, DLOOP3

        LJMP HERE0

DLOOP3: MOV R5, #195

HERE2:  DJNZ R5, DLOOP4

        LJMP HERE1

DLOOP4: MOV R4, #213

DLOOP5: NOP

        DJNZ R4, DLOOP5

        LJMP HERE2

        END
```

提高部分（选做）：

a. b.

;File name: one more thing.asm

;Description: P2,P1 输出 4 位十进制数，加一减一、开始停止通过 P3 的两个按钮控制

;Designed by: MA Xiaotian

;Date:2020-4-17

```
ORG 0000H

LJMP MAIN

SWITCH BIT P3.7      ;P3.7 状态选择加、减计数方式,1 加 0 减

CONTROL BIT P3.0      ;P3.0 状态控制开始和停止计数

ORG 2000H

MAIN:  MOV P1, #0      ;千百

        MOV P2, #0      ;十个

        CLR C

        MOV R0, #0

        MOV R1, #0

        MOV A, #0

        MOV SP, #40H


HERE:   JNB CONTROL, HERE      ;低电平停止计数，高电平开始计数

PLUS:   JNB SWITCH, MINUS

        JNB CONTROL, HERE

        CLR C

        MOV A, R0

        ACALL DELAY

        ADD A, #1

        DA A

        MOV P2, A
```

```
MOV R0, A
JNC PLUS           ;是否需要向千百进位?
MOV A, R1
ADD A, #1
DA A
MOV P1, A
MOV R1, A
JC PLUS
```

```
MINUS: JB SWITCH, PLUS
JNB CONTROL, HERE
CLR C
MOV A, R0
ACALL DELAY
ADD A, #099H
DA A
MOV P2, A
MOV R0, A
CJNE A, #099H, MINUS;是否需要向千百借位?
MOV A, R1
ADD A, #099H
DA A
MOV P1, A
MOV R1, A
CJNE R1, #0, MINUS
CJNE R0, #0, MINUS
CLR CONTROL
SETB C
JC MINUS
```

;=====DELAY SECTION=====

DELAY: MOV R7, #2

DLOOP1:

HERE0: NOP

DJNZ R7, DLOOP2

RET

DLOOP2: MOV R6, #9

HERE1: MOV R3, #16

HEREX: DEC R3

CJNE R3, #0, HEREX

DJNZ R6, DLOOP3

LJMP HERE0

DLOOP3: MOV R5, #195

HERE2: DJNZ R5, DLOOP4

LJMP HERE1

DLOOP4: MOV R4, #13

DLOOP5: NOP

DJNZ R4, DLOOP5

LJMP HERE2

END

六、程序测试方法与结果

1. 采用测试数据的方法

测试数据 -2 结果 (2000H)=01H



测试数据 0 结果 (2000H)=FFH



《微机实验》报告

测试数据 10 结果 (2000H)=F5H

EXERCISE 2.0416 - uVision4

```

01 ORG 0000H
02 LJMP MAIN
03 X EQU 10
04 ORG 2000H
05 ;假设Y存放在1FFFH,2000H,2001H,其中2001H为小数部分
06 MAIN: MOV DPTR, #1000H
07 MOV A, #X
08 MOVX @DPTR, A ;将X存储在外RAM1000H的位置
09 MOV R2, #20
10 MOV R1, #40
11 JB ACC.7, S1 ;如果是负数,则归到S1中
12 CJNE R2, #X, S1 ;如果是大于20的数,就跳出,另行
13 S1: JC S2
14 CPL A
15 MOV DPTR, #2000H
16 MOVX @DPTR, A
17 SJMP $
18
19 S2: CLR C
20 SUBB A, R1
21 CJNE A, #0, S3 ;这里处理的是X>=40的数
22 JNC S3
23 MOV A, #X
24 MOV R0, 20H
25 MOV 20H, #0
26 XCHD A, @R0
27 SWAP A
28 MOV R1, A
29 MOV B, A
30 MUL AB
31 MOV DPTR, #1FFFH
32 MOVX @DPTR, A
33 MOV A, @R0
34 MOV B, A
35 MUL AB
36 MOV DPTR, #2000H
37 MOVX @DPTR, A
38

```

Memory 1

Address: 0x002000

X: 0x002000: F5

测试数据 20 结果 (2000H)=EBH

EXERCISE 2.0416 - uVision4

```

01 ORG 0000H
02 LJMP MAIN
03 X EQU 20
04 ORG 2000H
05 ;假设Y存放在1FFFH,2000H,2001H,其中2001H为小数部分
06 MAIN: MOV DPTR, #1000H
07 MOV A, #X
08 MOVX @DPTR, A ;将X存储在外RAM1000H的位置
09 MOV R2, #20
10 MOV R1, #40
11 JB ACC.7, S1 ;如果是负数,则归到S1中
12 CJNE R2, #X, S1 ;如果是大于20的数,就跳出,另行
13 S1: JC S2
14 CPL A
15 MOV DPTR, #2000H
16 MOVX @DPTR, A
17 SJMP $
18
19 S2: CLR C
20 SUBB A, R1
21 CJNE A, #0, S3 ;这里处理的是X>=40的数
22 JNC S3
23 MOV A, #X
24 MOV R0, 20H
25 MOV 20H, #0
26 XCHD A, @R0
27 SWAP A
28 MOV R1, A
29 MOV B, A
30 MUL AB
31 MOV DPTR, #1FFFH
32 MOVX @DPTR, A
33 MOV A, @R0
34 MOV B, A
35 MUL AB
36 MOV DPTR, #2000H
37 MOVX @DPTR, A
38

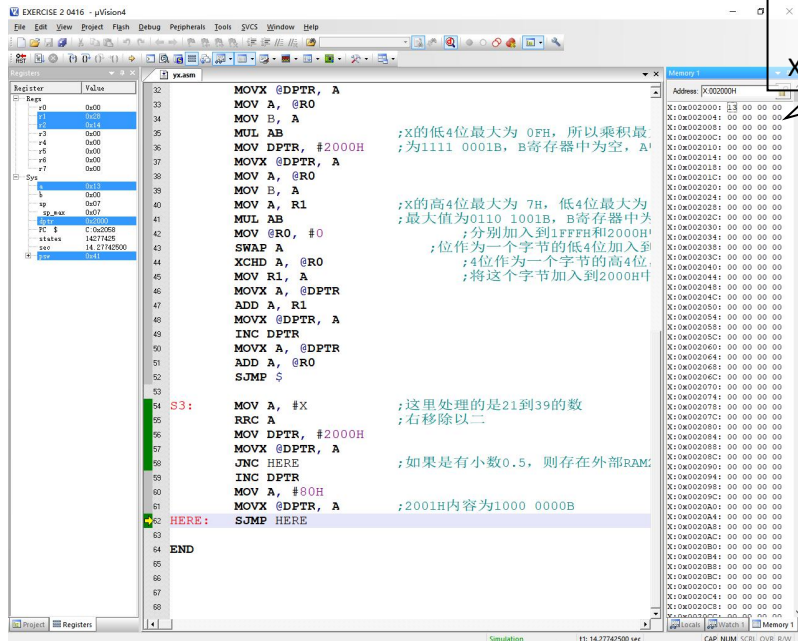
```

Memory 1

Address: 0x002000

X: 0x002000: EB

测试数据 38 结果 (2000H)=13H



```

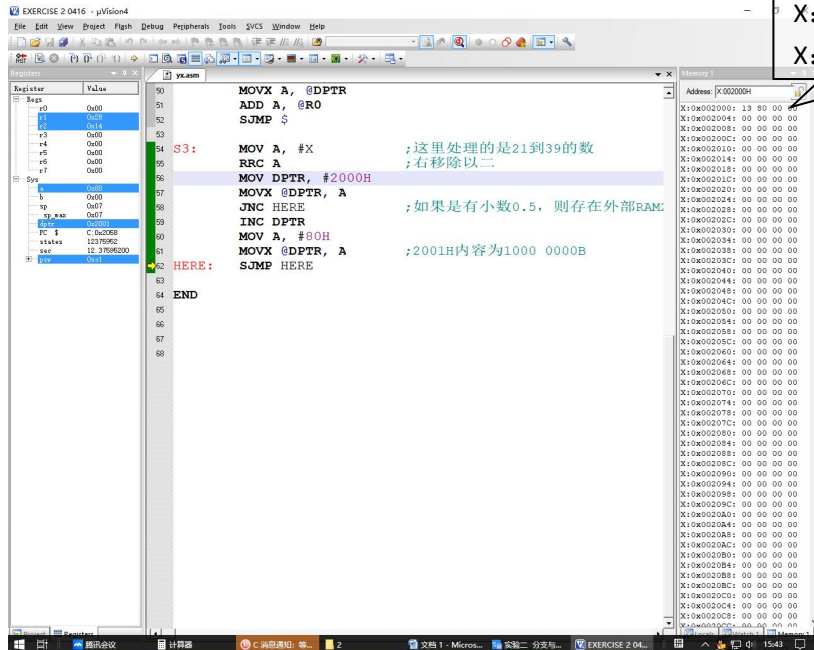
32      MOVX @DPTR, A
33      MOV A, @R0
34      MOV B, A
35      MUL AB
36      MOV DPTR, #2000H
37      MOVX @DPTR, A
38      MOV A, @R0
39      MOV B, A
40      MOV A, R1
41      MUL AB
42      SWAP A
43      XCHD A, @R0
44      MOV R1, A
45      MOVX A, @DPTR
46      ADD A, R1
47      MOVX @DPTR, A
48      INC DPTR
49      MOVX A, @DPTR
50      ADD A, @R0
51      SJMP $
52
53      MOV A, #X
54      RRC A
55      MOV DPTR, #2000H
56      MOVX @DPTR, A
57      JNC HERE
58      INC DPTR
59      MOV A, #80H
60      MOVX @DPTR, A
61      HERE: SJMP HERE
62
63      END
  
```

;X的低4位最大为 0FH, 所以乘积最大为 1111 0001B, B寄存器中为空, A!
 ;X的高4位最大为 7H, 低4位最大为 最大值为 0110 1001B, B寄存器中为
 ;分别加入到 1FFFH 和 2000H!
 ;位作为一个字节的低4位加入至
 ;4位作为一个字节的高4位
 ;将这个字节加入到 2000H!
 ;这里处理的是 21 到 39 的数
 ;右移除以二
 ;如果是有小数 0.5, 则存在外部 RAM
 ;2001H 内容为 1000 0000B

X EQU 38

X: 0x002000: 13

测试数据 39 结果 (2000H)=13H (2001H)=80H



```

50      MOVX A, @DPTR
51      ADD A, @R0
52      SJMP $
53
54      MOV A, #X
55      RRC A
56      MOV DPTR, #2000H
57      MOVX @DPTR, A
58      JNC HERE
59      INC DPTR
60      MOV A, #80H
61      MOVX @DPTR, A
62      HERE: SJMP HERE
63
64      END
  
```

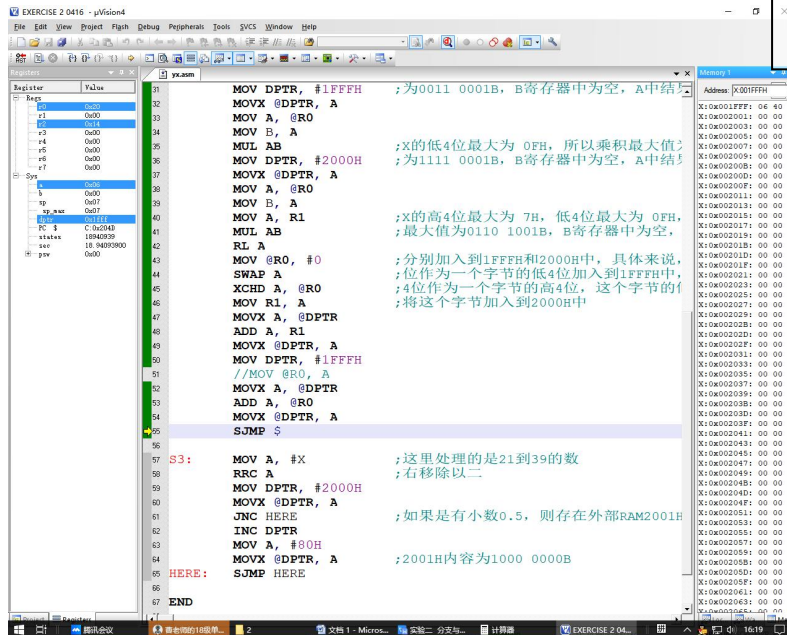
;这里处理的是 21 到 39 的数
 ;右移除以二
 ;如果是有小数 0.5, 则存在外部 RAM
 ;2001H 内容为 1000 0000B

X EQU 39

X: 0x002000: 13

X: 0x002001: 80

测试数据 40 结果 (1FFFF)=06H (2000H)=40H



```

MOV DPTR, #1FFFFH ;为0011 0001B, B寄存器中为空, A中结
MOVX @DPTR, A
MOV A, @R0
MOV B, A
MOV DPTR, #2000H ;X的低4位最大为 0FH, 所以乘积最大值
MOVX @DPTR, A ;为1111 0001B, B寄存器中为空, A中结
RL A
MOV A, @R0
MOV B, A
MOV A, R1
MOV B, R1 ;X的高4位最大为 7H, 低4位最大为 0FH,
MUL AB ;最大值为0110 1001B, B寄存器中为空,
RL A
MOV @R0, #0 ;分别加入到1FFFFH和2000H中, 具体来说,
SWAP A ;位作为一个字节的低4位加入到1FFFFH中,
XCHD A, @R0 ;4位作为一个字节的高4位, 这个字节的
MOV R1, A ;将这个字节加入到2000H中
MOVX A, @DPTR
ADD A, R1
MOVX @DPTR, A
MOV DPTR, #1FFFFH
//MOV @R0, A
MOVX A, @DPTR
ADD A, @R0
MOVX @DPTR, A
SUMP $

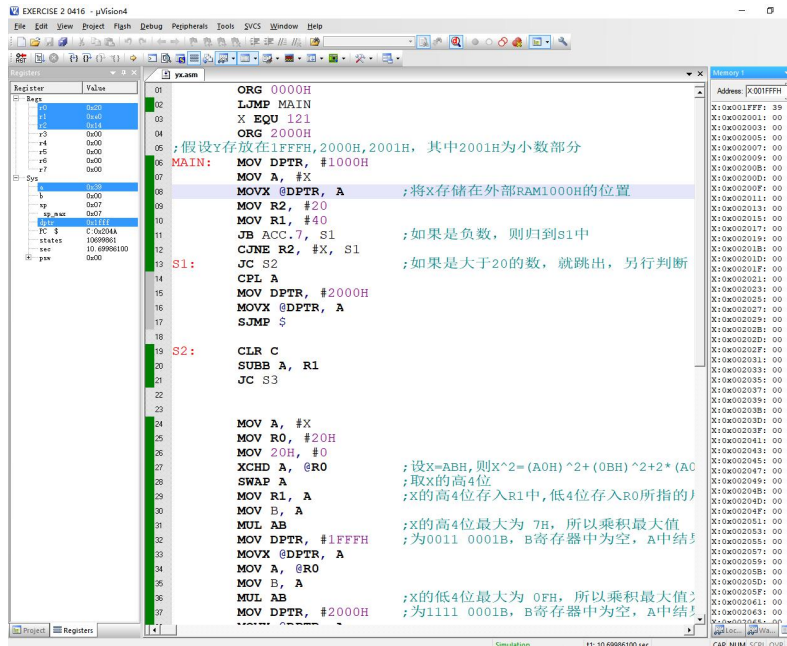
S3: MOV A, #X ;这里处理的是21到39的数
RRC A ;右移除以二
MOV DPTR, #2000H
MOVX @DPTR, A
JNC HERE ;如果有小数0.5, 则存在外部RAM2001H
INC DPTR
MOV A, #80H ;2001H内容为1000 0000B
MOVX @DPTR, A
HERE: SUMP HERE
END
  
```

X EQU 40

X: 0x001FFF: 06

X: 0x002000: 40

测试数据 121 结果 (1FFFF)=39H (2000H)=31H



```

ORG 0000H
LJMP MAIN
X EQU 121
ORG 2000H
;假设Y存放在1FFFFH, 2000H, 2001H, 其中2001H为小数部分
MAIN: MOV DPTR, #1000H
MOV A, #X
MOVX @DPTR, A ;将X存储在外部RAM1000H的位置
MOV R2, #20
MOV R1, #40
JB ACC.7, S1 ;如果是负数, 则归到s1中
CJNE R2, #X, S1 ;如果是大于20的数, 就跳出, 另行判断
JC S2
CLR A
MOV DPTR, #2000H
MOVX @DPTR, A
SJMP $

S2: CLR C
SUBB A, R1
JC S3

MOV A, #X
MOV R0, #20H
MOV @R0, #0
XCHD A, @R0 ;设X=ABH, 则X^2=(A0H)^2+(0BH)^2+2*(A
SWAP A ;取X的高4位
MOV R1, A ;X的高4位存入R1中, 低4位存入R0所指的
MOV B, A ;X的高4位最大为 7H, 所以乘积最大值
MUL AB ;为0011 0001B, B寄存器中为空, A中结
MOV DPTR, #1FFFFH
MOVX @DPTR, A
MOV A, @R0
MOV B, A ;X的低4位最大为 0FH, 所以乘积最大值
MUL AB ;为1111 0001B, B寄存器中为空, A中结
MOV DPTR, #2000H
MOVX @DPTR, A
SUMP $
  
```

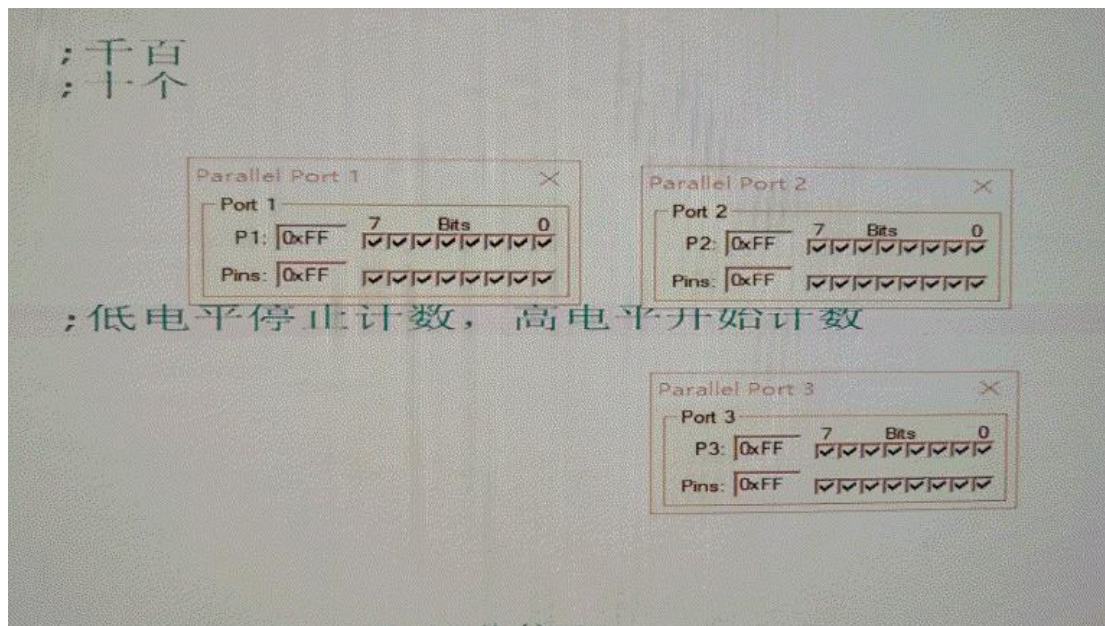
X EQU 121

X: 0x001FFF: 39

X: 0x002000: 31

提高部分（选做）：

a. b. 观察法，动图请点击下方 GIF（若超链接跳转失败，如需观察，可手动播放压缩包中的 GIF 文件）



思考题

1. 实现多分支结构程序的主要方法有哪些？举例说明。

答：实现多项分支的主要方法是采用分支表法。常用的分支表的组成有三种形式。

下面通过一个例子的三种不同解法，说明采用分支表法设计多向分支程序的要点。

例 4.2.8 根据 R3 的值，控制转向 8 个分支程序。

R3=0, 转向 SUBR0

R3=1, 转向 SUBR1

⋮

R3=7, 转向 SUBR7

(1) 分支地址表：它是由各个分支程序的首地址组成的一个线性表，每个首地址占连续的两个字节。

解法 1 采用分支地址表。

设分支地址表的标号为 BRATAB，程序设计的思路如下：

- $(BRATAB + R3 \times 2) \rightarrow DPTR$;
- $0 \rightarrow A$;
- 应用 $JMP @A + DPTR$ 指令。

根据上述思路设计的程序如下：

```

MOV    DPTR, #BRATAB    ;取表首地址
MOV    A, R3
ADD    A, R3             ;A ← R3 × 2
JNC    NADD
INC    DPH               ;R3 × 2 进位加到 DPH
NADD:  MOV    R4, A       ;暂存 A
       MOVC   A, @A + DPTR ;取分支地址高 8 位
       XCH    A, R4

       INC    A
       MOVC   A, @A + DPTR ;取分支地址低 8 位
       MOV    DPL, A       ;分支地址低 8 位送 DPL
       MOV    DPH, R4      ;分支地址高 8 位送 DPH
       CLR    A
       JMP    @A + DPTR    ;转相应分支程序
BRATAB: DW    SUBR0        ;分支地址表
       DW    SUBR1
       ⋮
       DW    SUBR7

```

(2) 转移指令表：它是由转移指令表组成的一个分支表，其各转移指令的目标地址即

为各分支程序的首地址。若采用短转移指令，则每条指令占两个字节；若采用长转移指令，则每条指令占三个字节。

解法 2 采用转移指令表。

设转移指令表的标号为 JMPTAB, 程序设计的思路如下:

- JMPTAB→DPTR;
 - $R3 \times K \rightarrow A$, 视转移指令表中的转移指令是二字节或三字节, 前者取 $K=2$, 后者取 $K=3$;
 - 应用 JMP@A+DPTR 指令。
- 根据上述思路设计的程序如下:

```

MOV    DPTR,# JMPTAB    ;取表首地址
MOV    A,R3
ADD    A,R3              ;A←R3×2
JNC    NADD
INC     DPH              ;有进位加到 DPH
NADD:  JMP    @A+DPTR    ;转相应分支程序
JMPTAB: AJMP   SUBR0      ;转移指令表
        AJMP   SUBR1
        :
        AJMP   SUBR7

```

转移表中使用 AJMP 指令, 限制分支的入口 SUBR0、SUBR1、…、SUBR7 必须和转移指令表位于同一个 2 K 字节范围内, 如果使用 LJMP 指令组成的转移指令表, 则可转入 64 K 字节空间范围的任何目标地址处。

(3) 地址偏移量表: 各分支程序首地址与地址偏移量表的标号之差 (为一个字节) 称为地址偏移量, 由这些地址偏移量组成地址偏移量表。

解法 3 采用地址偏移量表。

设地址偏移量表的标号为 DISTAB, 程序设计的思路如下:

- DISTAB→DPTR;
- (DISTAB+R3)→A;
- 应用 JMP@A+DPTR 指令。

根据上述思路设计的程序如下:

```

MOV    DPTR,# DISTAB    ;取表首地址
MOV    A,R3
MOVC   A,@A+DPTR        ;查表
JMP     @A+DPTR          ;转相应分支程序

```



```
DISTAB: DB      SUBR0-DISTAB      ;地址偏移量表
          DB      SUBR1-DISTAB
          ⋮
          DB      SUBR79-DISTAB
          ⋮
SUBR0:    ⋮
SUBR1:    ⋮
```

这种方法适用于分支路数较少,所有分支程序都处在同一页(256 个字节)的情况。

分支表的组织形式不同,实现分支转移的方法也各有差异,但都是利用转移指令 `JMP@A+DPTR` 来实现多向分支要求的。采用分支表法比多次使用比较转移指令来实现多路转移程序要精练一些,其执行速度快,特别是分支路数越多时,这种优点就更为明显。

2. 在编程上,十进制加 1 计数器与十六进制加 1 计数器的区别是什么? 怎样用十进制加法指令实现减 1 计数?

答: 由于计算机的二进制特性,其实默认的就是十六进制加 1 的计数器,但人类习惯于十进制的计数方式,所以在编程上为了便于观察可以设计为十进制加 1 的计数器。编程上,两者的区别在于: 十进制加 1 需要采用 `ADD A, #1` 以及 `DA A` 二-十进制转换指令。用十进制加法指令实现减 1 计数,则需要加 1 的十进制的补数,若仅涉及到一个字节内的显示,则使用 `ADD A, #99H` 以及 `DA A` 指令即可。

总结与反思:

这次试验由三个问题组成,层层递进吧,难度不小。我在第一个问题上耗时过长,导致后面的第二问没有在课上按时完成,提高部分更是在之后才写完。第一个问题,现在来看,有些小题大做的感觉,因为平方的那部分完全变成了我自己的数学推导,即平方和各项展开后相加,这是将问题复杂化所导致的。第二个问题是第三个问题的铺垫,我在调测 1s 定时程序的时候真的是很崩溃,调了一晚上是毫不夸张的,不过好在最后坚持了下来,并将误差控制在了 $1\mu s$, 远低于实验要求,结果很好,一分耕耘一分收获嘛,咬咬牙也就调出来了。这次试验真的是让我感到了一入单片机深似海,自己还是应该保持谦卑,认真学习,并且努力工作,所有的努力都不会白费,最后总会有结果!

本人承诺：

本报告内容真实，无伪造数据，无抄袭他人成果。本人完全了解学校相关规定，如若违反，愿意承担其后果。

签字：_____马笑天_____

_____2020_____年_____4_____月_____28_____日