EE2401 微算機系統Fall 2019

HW#3 (8051 Timer, Serial port, Interrupt) (10/14/2019)

Due date: 11/7/2019. Severe penalty will be given to late homework.

107061112 王昊文

Note:

(a) The homework will be graded based on your **documentation** and **demonstration**.

(b) For all **(Software Design)** problems, you are required to us **MCU8051IDE**simulators to simulate and verify your programs.

(c) You are required to **type** your homework (first the problem then your solution) by using a **word processor** and submit in .doc(or .docx) format under a filename **EE2401f19-hw3-student_no-vn.doc(or .docx)**, where **student_no** is your student number, e.g., **107061xxx** and **vn** is your version number, e.g., **v3**. You should **upload your .doc file** in **iLMS** by the specified deadline whenever you have a newer version. Follow the iLMS upload homework process to upload your file.

(d) The homework will be graded based on your **latest version**. Old version(s) will be discarded.

(e) Each homework assignment will have full score of 100 points. 5 points will be deducted if you do not comply with the naming convention. Severe grade penalty will be given to late homework. 20 points will be taken off per day after deadline till zero point.

(f) Please treat the above requirements as a kind of training in writing a decent homework report. If you have any problem regarding this homework, please feel free to consult with TA or me. If you think the time is too short to accomplish this homework, please let me know in class.

1. (Software Design) (20%)

Square wave/pulse generation

(a) Design an 8051 program that can create a square wave on P1.0 with a frequency of 50kHz, 5kHz, 1kHz, and 100Hz, assuming a 12MHz 8051 is used. Analyze the accuracy of your design for these 4 square waves.

50kHZ:

Thought Process:

This is a relatively fast clock cycle, the counting needed is only 5us per half wave cycle. Hence, we only need the NOP instruction, acting as a delay time, to finish our counting.

```
        ; 50kHZ, 5us per half cycle
LOOP:   SETB    P1.0        ; 1us
        NOP                 ; 1us
        NOP                 ; 1us
        NOP                 ; 1us
        NOP                 ; 1us, half cycle completed
        CLRP1.0 ; 1us
        NOP                 ; 1us
        NOP                 ; 1us
        SJMP    LOOP        ; 2us, repeat
```

5kHz, 1kHz, and 100Hz,

Thought process:

These frequencies are designed in the same process. Since these frequencies require longer counting, we can't use the NOP instruction, we use the internal timer for our time counting.

When using the internal counter, we are using the timer mode 1, that is, we have to manually initialize the value for the timer every time it overflows. Once it is setup, the problem simplifies down to how much the timer should count.

For 5kHZ, it counts for 50us per half wave cycle. Let's look at my code for 5kHZ

```
        ; 5kHZ    , 50us per half clock cycle

        MOV     TMOD,   #01H    ; T0 16-bit mode
LOOP:   MOV     TH0,    #0FFH   ; 2us
        MOV     TL0,    #0DAH   ; 2us
        SETB    TR0             ; 1us
WAIT:   JNB     TF0,    $       ; 2us, count 38, 12us execution
        CLR     TR0             ; 1us
        CLR     TF0             ; 1us
        CPL     P1.0            ; 1us
        SJMP    LOOP            ; 2us
                                ; total 50us
        ; 38D = 26H,   10000H – 26H = FFDAH
```

Our goal is to make the delay time between the instructions "SETB TR0" and "CLR TR0" a total of 50us. Although we have to consider the execution time for initializing and JMP instructions. Those instructions add up to 12us. Hence we only have to count to 38us.

This design can extend to other frequencies required, simply assign different counting times to it.

```
        ; 1kHZ    , 500us per half clock cycle

        MOV     TMOD,   #01H    ; T0 16-bit mode
LOOP:   MOV     TH0,    #0FEH   ; 2us
        MOV     TL0,    #018H   ; 2us
        SETB    TR0             ; 1us
WAIT:   JNB     TF0,    $       ; 2us, count 448, 12us execution
        CLR     TR0             ; 1us
        CLR     TF0             ; 1us
        CPL     P1.0            ; 1us
        SJMP    LOOP            ; 2us
                                ; total 500us
        ; 488D = 01E8H,    10000H – 01E8H = 0FE18H
```

```
        ; 100HZ   , 5000us per half clock cycle
        MOV     TMOD,   #01H    ; T0 16-bit mode
```

```
LOOP:   MOV     TH0,    #0ECH   ; 2us
        MOV     TL0,    #084H   ; 2us
        SETB    TR0             ; 1us
WAIT:   JNB     TF0,    $       ; 2us, count 4988, 12us execution
        CLR     TR0             ; 1us
        CLR     TF0             ; 1us
        CPL     P1.0            ; 1us
        SJMP    LOOP            ; 2us
                                ; total 5000us
        ; 4988D = 0137CH,      10000H – 0137CH = 0EC84H
```

(b) Design an 8051 program to generate a 4kHz 60/40 duty cycle pulse wave on P1.4. (60/40 means 60% high and 40% low in one cycle) Analyze the accuracy of your design for this wave.

```
        ; 4kHZ   , the up cycle, for 150us
        MOV     TMOD,   #01H    ; T0 16-bit mode
LOOP1:  MOV     TH0,    #0FFH   ; 2us
        MOV     TL0,    #076H   ; 2us
        SETB    TR0             ; 1us
WAIT1:  JNB     TF0,    $       ; 2us, count 138, 12us execution
        CLR     TR0             ; 1us
        CLR     TF0             ; 1us
        CPL     P1.0            ; 1us
        SJMP    LOOP2           ; 2us
                                ; total 150us
        ; 4kHZ   , the down cycle for 100us
LOOP2:  MOV     TH0,    #0FFH   ; 2us
        MOV     TL0,    #0A8H   ; 2us
        SETB    TR0             ; 1us
WAIT2:  JNB     TF0,    $       ; 2us, count 88, 12us execution
        CLR     TR0             ; 1us
        CLR     TF0             ; 1us
        CPL     P1.0            ; 1us
        SJMP    LOOP1           ; 2us
                                ; total 100us
```

That adds up to 250us, which is required for 4kHZ frequency.

2. (Software Design) (20%)

Music playing

(a) Refer to the C example in the class for playing Do, Re, Mi, Fa, So, La, Ti, Do-H, redesign the program in 8051 assembly language.

Using T0 as our time counter, T1 as our note counter.
T0 will count 0.05 second per timeout.
R6 will the store the value of timer repeat counting times. It can control the length of time of every note playing. In this program, we set it to count five times, that is a total of 0.25 seconds per note.
R7 will be the pointer to the note table. Note that we are using DW when constructing the table, but our TH1, TL1 only have a capacity of 2 bytes each.

Using PC to point to the table, we have to multiply R7 by every time in order to count to the correct note.

```
;*****************************************************************
MONITOR     CODE    00BCH   ; MON51 (V12) entry point
COUNT   EQU     0EC78H      ; 0.05 seconds per timeout
REPEAT  EQU     5    ; 5 x 0.05 = 0.25 seconds/note
;*****************************************************************
; Note: X3 not installed on SBC-51, therefore
; interrupts directed to the following jump table
; beginning at 0000H
;*****************************************************************

            ORG     0000H       ; RAM entry points for...
            LJMP    MAIN        ; main program
            ORG     0003H
            LJMP    EXT0ISR     ; External 0 interrupt
            ORG     000BH
            LJMP    T0ISR       ; Timer 0 interrupt
            ORG     0013H
            LJMP    EXT1ISR     ;External 1 interrupt
            ORG     001BH
            LJMP    T1ISR       ;Timer 1 interrupt
            ORG     0023H
            LJMP    SPISR       ;Serial Port interrupt
            ORG     002BH
            LJMP    T2ISR       ;Timer 2 interrupt

;*****************************************************************
; MAIN PROGRAM BEGINS
;*****************************************************************
MAIN:   MOV     TMOD,   #11H            ;both timers 16-bit mode
        MOV     R7,     #0              ;use R7 as note counter
        MOV     R6,     #REPEAT         ;use R6 as timeout counter
        MOV     IE,     #8AH            ;Timer 0 & 1 interrupts on
        SETB    TF1                     ;force Timer 1 interrupt
        SETB    TF0                     ;force Timer 0 interrupt
        SJMP    $                       ;ZzZzZzZz time for a nap

;*****************************************************************
; TIMER 0 INTERRUPT SERVICE ROUTINE (EVERY 0.05 SEC.)
;*****************************************************************
T0ISR:  CLR     TR0                     ;stop timer
        MOV     TH0,    #HIGH (COUNT)   ;reload
        MOV     TL0,    #LOW (COUNT)
        DJNZ    R6,     EXIT            ;if not 5th time, exit
        MOV     R6,     #REPEAT         ;if 5th, reset
        INC     R7                      ;increment note
        CJNE    R7,     #LENGTH,EXIT    ;beyond last note?
        MOV     R7,     #0              ;yes: reset, A=440 Hz
```

```
EXIT:   SETB    TR0             ;no: start timer, go
        RETI                    ;back to ZzZzZzZ

;*****************************************************************
; TIMER 1 INTERRUPT SERVICE ROUTINE (PITCH OF NOTES)
;
; Note: The output frequencies are slightly off due
; to the length of this ISR. Timer reload values
; need adjusting.
;*****************************************************************
T1ISR:  CPL     P1.7            ;music maestro!
        CLR     TR1             ;stop timer
        MOV     A,  R7          ;get note counter
        RL      A               ;multiply (2 bytes/note)
        CALL    GETBYTE         ;get high-byte of count
        MOV     TH1,    A       ;put in timer high register
        MOV     A,      R7      ;get note counter again
        RL      A               ;align on word boundary
        INC     A               ;past high-byte (whew!)
        CALL    GETBYTE         ;get low-byte of count
        MOV     TL1,    A       ;put in timer low register
        SETB    TR1             ;start timer
        RETI                    ;time for a rest
;*****************************************************************
; GET A BYTE FROM LOOK-UP OF NOTES IN A MAJOR SCALE
;*****************************************************************
GETBYTE:    INC     A           ;table look-up subroutine
            MOVC    A,  @A+PC
            RET

TABLE:      DW      0F887H      ;C
            DW      0F95AH      ;D
            DW      0FA14H      ;E
            DW      0FA69H      ;F
            DW      0FB05H      ;G
            DW      0FB90H      ;A
            DW      0FC0CH      ;B
LENGTH      EQU     7           ;LENGTH = # of notes
;*****************************************************************
; UNUSED INTERRUPTS - BACK TO MONITOR PROG (ERROR)
;*****************************************************************
EXT0ISR:
EXT1ISR:
SPISR:
T2ISR:      CLR     EA          ; shut off interrupts and
            LJMP    MONITOR         ; return to MON51
            END
```

(b) Modify the program in (a) so that the time duration for each note can be varying. (You might need to use interrupt in your design if you like. Please study the class material and technical references posted in the website)

Using a similar approach, the most part of the program is left unchanged. Since we have to change each note's length, we will modify the ISR for T0, we have to construct another time table, storing time values inside the table and using R5 as our pointer. What R5 points to will then be stored inside R6, and the other part of the program remains the same.

```
;*************************************************************
MONITOR    CODE    00BCH   ; MON51 (V12) entry point
COUNT  EQU     0EC78H  ; 0.05 seconds per timeout
;*************************************************************
; Note: X3 not installed on SBC-51, therefore
; interrupts directed to the following jump table
; beginning at 0000H
;*************************************************************

            ORG     0000H   ; RAM entry points for...
            LJMP    MAIN    ; main program
            ORG     0003H
            LJMP    EXT0ISR     ; External 0 interrupt
            ORG     000BH
            LJMP    T0ISR   ; Timer 0 interrupt
            ORG     0013H
            LJMP    EXT1ISR     ;External 1 interrupt
            ORG     001BH
            LJMP    T1ISR   ;Timer 1 interrupt
            ORG     0023H
            LJMP    SPISR   ;Serial Port interrupt
            ORG     002BH
            LJMP    T2ISR   ;Timer 2 interrupt


;*************************************************************
; MAIN PROGRAM BEGINS
;*************************************************************
MAIN:       MOV     TMOD,   #11H    ;both timers 16-bit mode
            MOV     R7,  #0 ;use R7 as note counter
            MOV     R6,  #1 ;use R6 as timeout counter
            MOV     IE,  #8AH   ;Timer 0 & 1 interrupts on
            SETB    TF1         ;force Timer 1 interrupt
            SETB    TF0         ;force Timer 0 interrupt
            SJMP    $           ;ZzZzZzZz time for a nap

;*************************************************************
; TIMER 0 INTERRUPT SERVICE ROUTINE (EVERY 0.05 SEC.)
; USE R5 TO BE THE TABLE COUNTER
```

```
;*************************************************************
T0ISR:          CLR     TR0                             ;stop timer
                MOV     TH0,    #HIGH (COUNT) ;reload
                MOV     TL0,#LOW (COUNT)
                MOV     A,      R5              ; get time counter
                DJNZ    R6,     EXIT    ;if not finished playing, jump
                CALL    GETTIME         ; if zero, get next time length
                MOV     R6,  A          ; get time length
                INC     R5              ; increment time table pointer
                INC     R7              ;increment note
                CJNE    R7,#LENGTH,EXIT     ;beyond last note?
                MOV     R7,     #0              ;yes: reset, A=440 Hz
EXIT:           SETB    TR0             ;no: start timer, go
                RETI                    ;back to ZzZzZzZ
GETTIME:        INC A
                MOVC    A,   @A+PC
                RET
TIMETABLE:DB    1
                DB  2
                DB  3
                DB  4
                DB  5
                DB  6
                DB  7
;*****************************************************************
; TIMER 1 INTERRUPT SERVICE ROUTINE (PITCH OF NOTES)
;
; Note: The output frequencies are slightly off due
; to the length of this ISR. Timer reload values
; need adjusting.
;*****************************************************************
T1ISR:          CPL     P1.7    ;music maestro!
                CLR     TR1             ;stop timer
                MOV     A,   R7         ;get note counter
                RL  A                   ;multiply (2 bytes/note)
                CALL    GETBYTE         ;get high-byte of count
                MOV     TH1,    A   ;put in timer high register
                MOV     A,   R7         ;get note counter again
                RL  A                   ;align on word boundary
                INC     A               ;past high-byte (whew!)
                CALL    GETBYTE         ;get low-byte of count
                MOV     TL1,A   ;put in timer low register
                SETB    TR1             ;start timer
                RETI                    ;time for a rest
;*************************************************************
; GET A BYTE FROM LOOK-UP OF NOTES IN A MAJOR SCALE
;*************************************************************
GETBYTE:    INC     A       ;table look-up subroutine
            MOVC    A,   @A+PC
            RET
```

```
TABLE:      DW      0F887H  ;C
            DW      0F95AH          ;D
            DW      0FA14H          ;E
            DW      0FA69H          ;F
            DW      0FB05H          ;G
            DW      0FB90H          ;A
            DW      0FC0CH          ;B
LENGTH      EQU     0CH     ;LENGTH = # of notes

;*************************************************************
; UNUSED INTERRUPTS - BACK TO MONITOR PROG (ERROR)
;*************************************************************
EXT0ISR:
EXT1ISR:
SPISR:
T2ISR:      CLR     EA      ; shut off interrupts and
            LJMP    MONITOR         ; return to MON51
            END
```

3. (Hardware schematic design and software design) (20%)

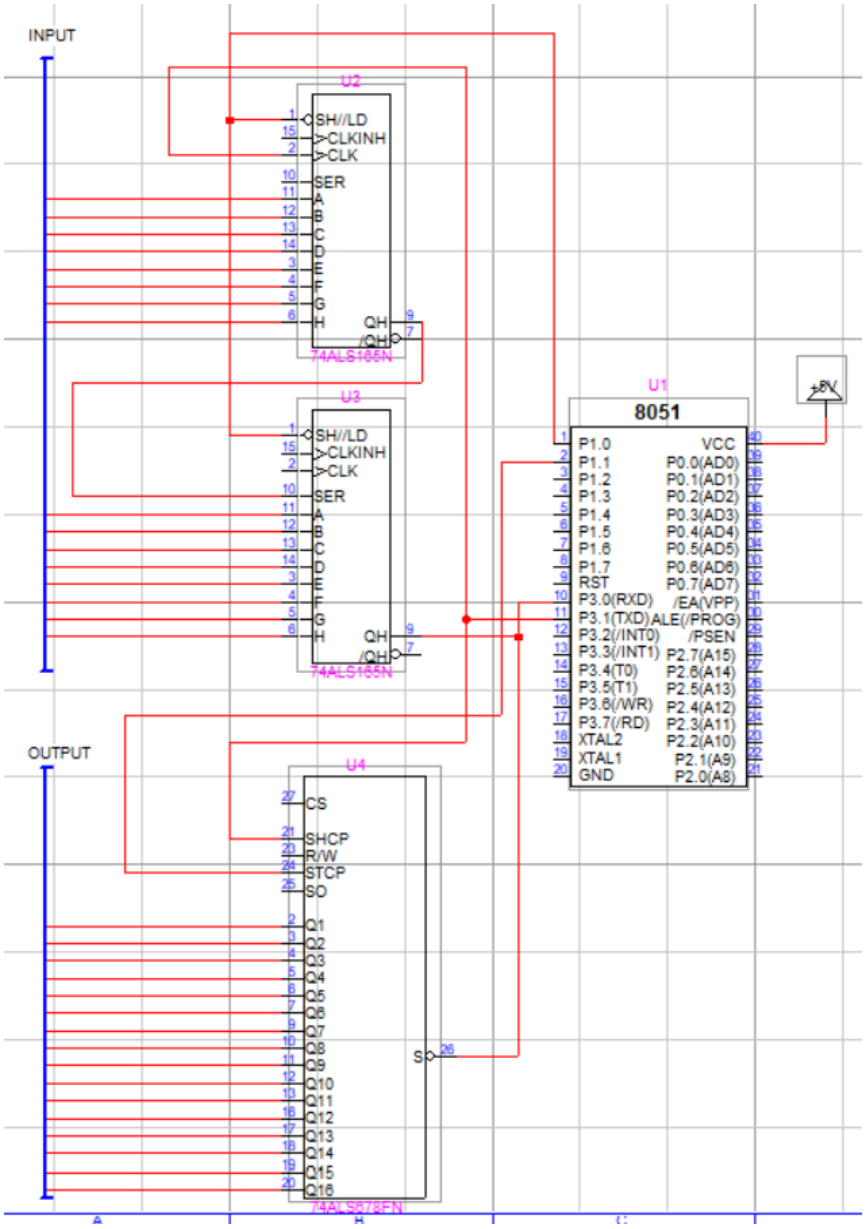
I/O expansion using **shift registers of serial port**

We want to use serial port in **mode 0** to expand 8051 I/O pins. Assume that we want to have 16 extra output pins with each bit value stored in bit-addressable memory 20H, 21H and 16 extra input pins read into bit-addressable memory 22H, 23H using serial-in-parallel-out and parallel-in-serial-out shift registers connected to 8051 serial port for I/O expansion, respectively. The I/O pins should be updated at least 100 times per second.

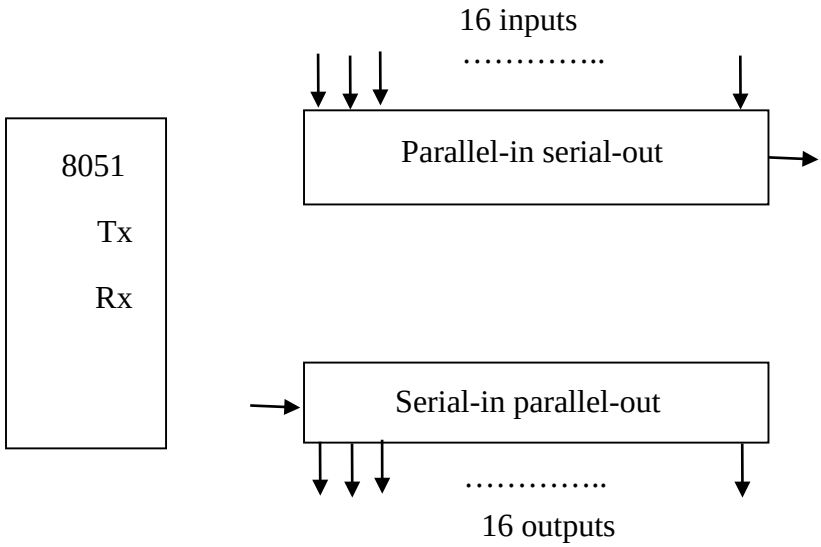
(a) Please pick the proper TTL shift registers and ICs whenever needed, study their datasheets, and draw a simplified schematic diagram showing the necessary connection of them with 8051.

Parallel-in-serial out shift register: 74165
Serial-in-parallel-out shift register: 74675

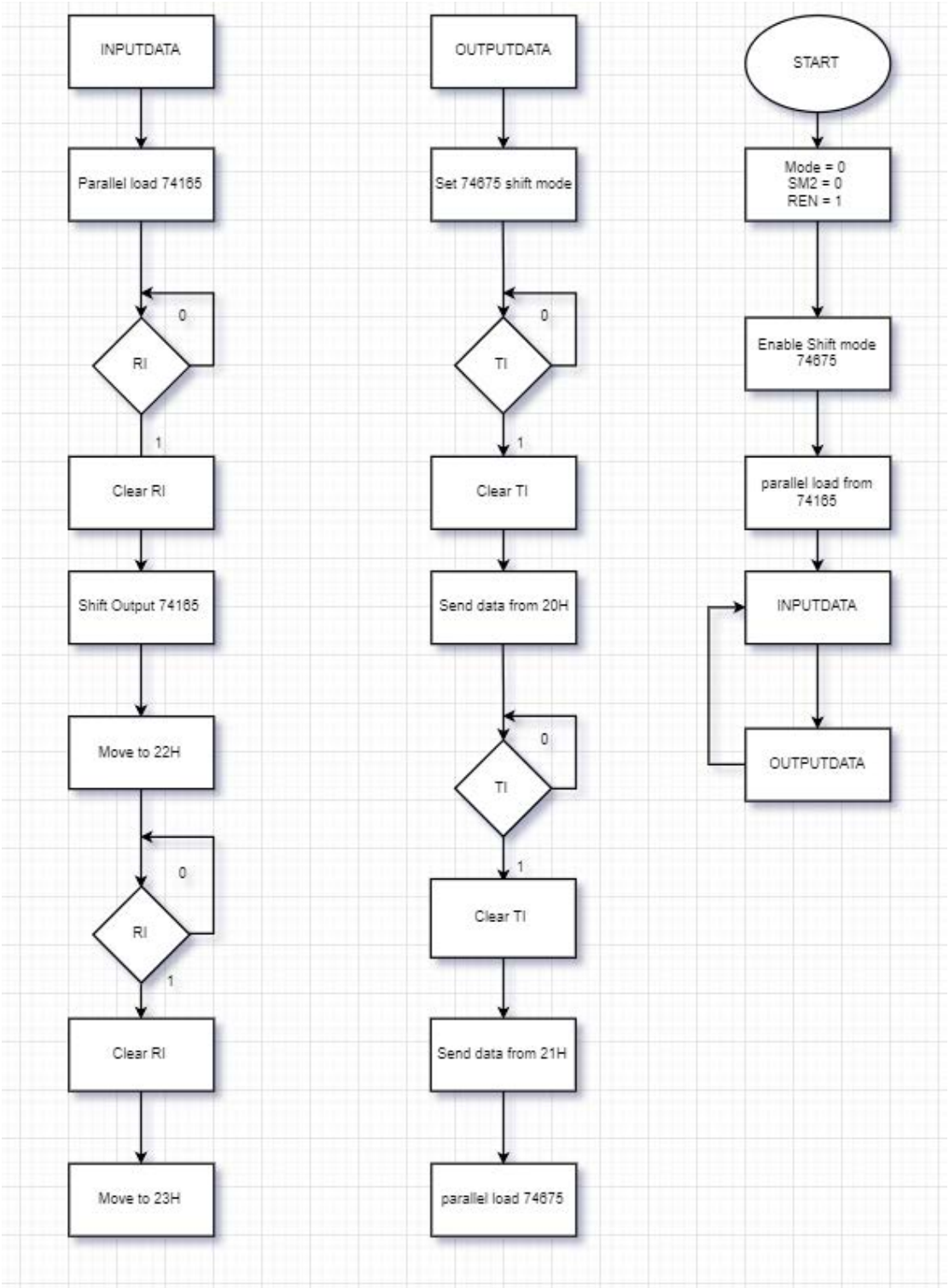
Since we are in mode 0, TXD act as the clock for the shift registers and RXD are for data input and output.



(b) Design an 8051 program for fulfilling this task by first designing your flow chart, examining its logical validity, then designing the codes.


16 inputs
……..
8051
Tx
Rx
Parallel-in serial-out
Serial-in parallel-out
……..
16 outputs

Flow Chart:




```
                MOV     SCON,   #12H        ; mode 0, SM2 = 0, REN = 1
; ***************************************************************************
; Using 16 extra input ports, we have to use 74165 to convert the parallel data to serial
; and then store it in 22H and 23H
;****************************************************************************
                SETB    P1.0                ; parallel load from 74165
                CLR     P1.1                ; disable parallel load from 74675
; start transmit to 20H
INPUTDATA:      JNB     RI,   $             ; RX ready?
```

```
                    CLR     RI
                    CLRP1.0                 ; shift output to 8051
                    MOV     22H,    SBUF    ; move the input data to 20H
; start transmit to 21H
                    JNB     RI,  $          ; TX ready?
                    CLR     RI
                    MOV     23H,    SBUF    ; move the input data to 21H
;*********************************************************************
; Using 16 extra output ports, we have to use 74675 to convert the serial data from the
; 8051 to 16 output parallel form.
;*********************************************************************
OUTPUTDATA:  CLR    P1.1    ; disable parallel load from 74675
; send out 20H
                    JNB     TI,  $
                    CLR     TI
                    MOV     SBUF,   20H ; send output data
; send out 21H
                    JNB     TI,  $  ; TX ready?
                    CLR     TI
                    MOV     SBUF,   21H ; send output data
                    SETB    P1.1    ; Enable parallel load to 74675 to send out
data
                    LJMP    INPUTDATA
```

4. (Software Design) (20%)

Serial communication

(a) Write a main program that will first initialize the 8051 serial port in **mode 1** with 9600 baud, **odd parity**, non-interrupt-driven, and then call a subroutine called **INLINE** to read from serial port a line of ASCII codes terminated with a carriage return character code (0DH) and store it in a line buffer with an extra null character (00H) for its termination. It will then call a subroutine **OUTSTR** to send out the received line in the line buffer via serial port. Assume the line buffer is residing in the **external data memory** beginning at address **60H**with the length of the line buffer limited to 40 bytes. When calling both subroutines, the **R0** is used as the pointer to the line buffer.

Thoughts:

Using serial mode 1 can use timer to control the baud rates.
Set up the serial modes, counter, setup the baud rate.
WHILE(1):
WHILE(not return code and not out of memory) {
R0 points to 60H
Call INLINE:
Is RI ready??
CLR RI
Get the character from SBUF, put in A (Read a character)
IF [out of memory range]
Move #00H to A
Move A @R0
Return

```
                    ELSE IF [character return code]
                        Move #00H to A
                        Move A @R0
                        Return
                    ELSE
                        Move A to @R0
                        Return
            }
            WHILE(not #00H null character) {
                R0 points to 60H
                Call OUTSTR:
                    Get char from R0 to A
                    R0 points to next data
                    IF [not null character]
                        Deal with the parity bit
                        Is TI ready?
                        CLR TI
                        Throw A into SBUF
                    ELSE send return code
                        Move return code into A
                        Send out to SBUF and output.
              }
CR    EQU     0DH; carriage return
NC    EQU     00H ; null character
              MOV     SCON,   #70H      ; mode1, SM2 = 1, REN = 1
              ANL     PCON,   #7FH      ; SMOD = 0 for 9600 baud
              MOV     TMOD,   #20H      ; timer 1 mode 2 for baud rate
MAIN:         MOV     TL1,    #0FDH     ; 9.6Kbps
              MOV     TH1,    #0FDH     ; auto reload
              SETB    TR1               ; start t1
              MOV     R0,     #60H      ; initialize pointer
              LCALL   INLINE            ; recieve
              MOV     R0,     #60  ; initial pointer after recieve
              LCALL   OUTSTR
              SJMP    MAIN

INLINE:       LCALL   INCHR
              CJNE    R0,     #87H,INRANGE ; check if out of range
              SJMP    NULL
INRANGE:      CJNE    A,      #CR, KEEPGOIN
NULL:         MOV     A,      #0   ; assign null character
              MOVX    @R0,    A    ; to buffer
              RET

INCHR:        JNB     RI,  $          ; is the system ready?
              CLR     RI              ; stop interrupt
              MOV     A,   SBUF       ; move the recieve data to acc
              MOV     C,   P
              CPL C
              CLRA.7
```

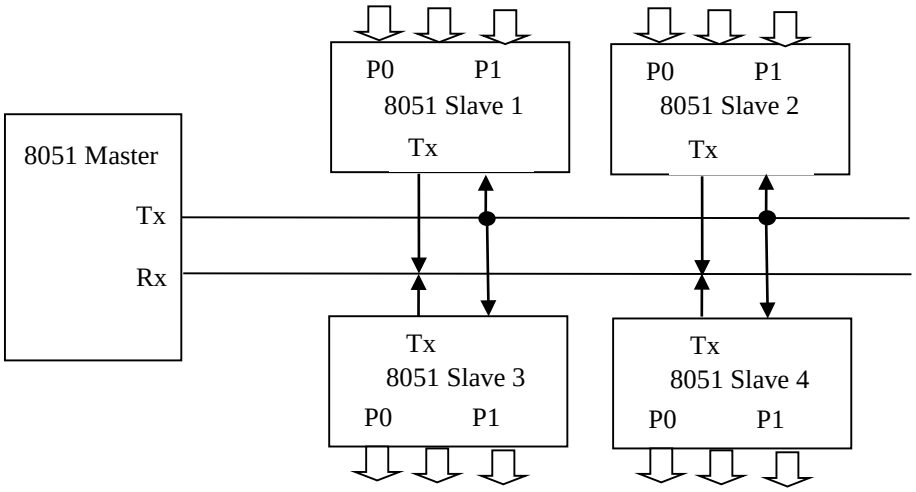
```
                RET
KEEPGOIN:       MOVX    @R0,    A       ; a valid character, store in reg
                INC R0          ; point to the next address
                AJMP    INLINE          ; input next word

OUTSTR:         MOVX    A,  @R0
                INC R0
                CJNE    A,  #00H, KEEPGOUT ; check if null
                MOV     A,  #CR             ; if null, send CR
                LCALL   OUTCHR              ; send
                RET
KEEPGOUT:       LCALL   OUTCHR              ; send char
                AJMP    OUTSTR              ; continue
OUTCHR:         MOV     C,  P               ; parity bit in C
                CPL C                       ; change to odd parity
                MOV     A.7, C              ; add to character
                JNB TI, $                   ; is system ready to transmit?
                CLRTI                       ; disable interrupt
                MOV     SBUF,   A           ; output the datas
                CLRA.7                      ; strip off parity bit
                RET
                End
```

(b) Repeat (a) with serial port enabled in interrupt-driven mode.

```
CR  EQU     0DH     ; carriage return
NC  EQU     00H; null character

                ORG     0000H
                LJMP    MAIN
                ORG     0023H
                LJMP    SPISR       ; serial port interrupt service

                ORG     0030H
MAIN:           MOV     SCON,   #70H    ; mode1, SM2 = 1, REN = 1
                ANL     PCON,   #7FH    ; SMOD = 0 for 9600 baud
                MOV     TMOD,       #20H    ; timer 1 mode 2 for baud
rate
LOOP:           MOV     TL1,    #0FDH   ; 9.6Kbps
                MOV     TH1,    #0FDH   ; auto reload
                SETB    TR1     ; start t1
                MOV     IE,     #90H    ; interrupt, enable = 1, ES = 1
                MOV     R0,     #60H    ; initialize pointer
                SJMP    $               ; jump until interrupt

SPISR:          JB      RI,   INLINE
                JB      TI, OUTSTR
```

```
                SJMP    SPISR


INLINE:         CLRRI                   ; stop interrupt
                MOV     A,  SBUF        ; move the receive data to acc
                MOV     C,  P
                CPL C
                CLRA.7
                CJNE    R0, #87H,INRANGE ; check if out of range
                SJMP    NULL
INRANGE:        CJNE    A, #CR, KEEPGOIN
NULL:           MOV     A,  #NC; assign null character
                MOVX    @R0,    A       ; to buffer
                RETI
KEEPGOIN:       MOVX    @R0,    A       ; a valid character, store in reg
                INC R0          ; point to the next address
                AJMP    INLINE          ; input next word

OUTSTR:         CLR     TI
                MOVX    A,  @R0
                INC R0
                CJNE    A, #NC, KEEPGOUT ; check if null
                MOV     A,  #CR ; if null, send CR
                LCALL   OUTCHR              ; send
                RETI
KEEPGOUT:       LCALL   OUTCHR              ; send char
                AJMP    OUTSTR          ; continue
OUTCHR:         MOV     C,  P           ; parity bit in C
                CPL C                   ; change to odd parity
                MOV     A.7,    C       ; add to character
                CLRTI                   ; disable interrupt
                MOV     SBUF,   A       ; output the datas
                CLRA.7                  ; strip off parity bit
                RET
                End
```

5. (Hardware and software design) (20%)
I/O expansion using **multiprocessor communication**



As shown above, we have an 8051 master and 4 8051slaves connected in multiprocessor communication configuration using serial ports. Each slave has its id as shown. Slave 1 and 2 provide 6 x 8 = 48 input pins; slave 3 and 4 provide 48 output pins. The states of those 96 I/O pins are to be stored at master's bit-addressable memory starting at bit address 00H ~ bit address 5FH (i.e., byte address 20H ~ 2BH). Please design codes for master and each slave (slave 1 and 2 may have similar code, so do slave 3 and 4) to fulfill the task. You should provide:

- a subroutine to initialize master and slave 8051's and configure the input, output ports,
- a subroutine to read in all input ports and placed them in master's internal bit-addressable bits,
- a subroutine to output from masters bit-addressable bits to slave 3 and 4 ports, and
- a main program that can continuously handle those I/O pins.

(a) Design flowcharts or pseudo codes for Init subroutine, Input subroutine, and Output subroutine for 8051 master and slaves.

Before drawing flowchart and writing pseudo code, just trying to express the problem in my words.

The objective is to use the slaves as our I/O expansion, so the result is that we can view the ports of the slaves as a part of the master. The input slaves help the master to receive the data inputs. So the input slaves have to send the data to the master. The output slaves help the master to send out the data stored in master's memory, hence the master has to send the data to the output slaves. These are the works we have to do.

1. Master send out slave id to each slave, but only want to select input slaves.
2. Each slave checks the broadcasting from the master.
3. Input slave try to send data to the master.
4. Master receive the data and store it in the corresponding memory.
5. When the pointer points to output memories, the master sends out the slave id to select the output slaves.

6. Each slave checks the broadcasting from the master, the corresponding slave should prepare to get data from the master.
7. The master sends the data to output slave.

The pseudo code for the master

```
INIT()
BEGIN
    [serial port mode 2]
    [SM2 = 0]                // the master does not have to receive address
    [REN = 1]
    [SMOD = 0]
    [R0 = 20H]               // initialize the pointer
END
/* The input function is for the input slaves, the master will send the id to the input
slaves and receive the input data from the slaves */
INPUT()
BEGIN
    WHILE [pointer >= 26H]
    BEGIN
        [TB8 = 1]       // start to send address bit to each slave
        [Wait for TI to be ready]
        [The master send out the slave id]
        [Wait for RI]
        [The master receive data from the specific slave]
        [Master store the data in the memory using the pointer]
        [Pointer jumps to next memory]
    END
    RETURN
END

/* This function is for the output slaves. The master will send out the slave id to the
output slaves. Then the master will send out the output data from the memory to the
designated output slave */

OUTPUT()
BEGIN
    WHILE [pointer <= 2CH]      // out of the designated memory
    BEGIN
        [TB8 = 1]       // start to send address bit to each slave
        [Wait for TI to be ready]
        [The master send out the slave id]
        [Wait for RI]
        [The master send data to the designated slave]
        [Master send the data in the memory using the pointer]
        [Pointer jumps to next memory]
    END
    RETURN
END
```

The pseudo code for the slaves

Slave 1 and slave 2(input slaves)

```
INIT()
BEGIN
    [serial mode 2]
    [SM2 = 1]                   // the slaves should be able to receive address bit
    [REN = 1]
    [SMOD = 0]
END
```

/* This function will let the slaves listen to the broadcast, and send out the data from its port. */

Slave 1:

```
INPUT()
BEGIN
WHILE [1]
    [Wait for RI]               // prepare to receive address bit
    [Start receiving]
    [Put the received address bit into accumulator for matching]
    IF [address byte = 20H]         // If slave id matched
    BEGIN
        [Wait for TI]
        [start transmit output port data to the master]
        BREAK
    END
    ELSE IF [address byte == 21H]     // next slave id
    BEGIN
        [Wait for TI]
        [Start transmit output port data to the master]
        BREAK
    END
    ELSE IF [address byte == 22H]
    BEGIN
        [Wait for TI]
        [Start transmit output port data to the master]
        BREAK
    END
END
```

Slave 2:

```
INPUT()
BEGIN
WHILE [1]
    [Wait for RI]               // prepare to receive address bit
    [Start receiving]
    [Put the received address bit into accumulator for matching]
```

```
    IF [address byte = 23H]         // If slave id matched
    BEGIN
        [Wait for TI]
        [start transmit output port data to the master]
        BREAK
    END
    ELSE IF [address byte == 24H]     // next slave id
    BEGIN
        [Wait for TI]
        [Start transmit output port data to the master]
        BREAK
    END
    ELSE IF [address byte == 25H]
    BEGIN
        [Wait for TI]
        [Start transmit output port data to the master]
        BREAK
    END
END
```

Slave 3 and Slave 4 (output slaves)

/* This function is for the output slaves. It will receive the slave id from the master and check if id matched. If id matched, it will receive the data from the master and act as a output port of the master */

Slave 3:

```
[serial mode 2]
[SM2 = 1]                   // the slaves should be able to receive address bit
[REN = 1]
[SMOD = 0]

WHILE[1]                    // always doing the same work
BEGIN
    [Check if RI ready]         // prepare to receive slave id
    [Receive ID from master]
    [Store the data into the accumulator of the slave for checking ID]
    [Clear SM2 to get the actual data from the master]       // disable the effect of the
ninth bit
    IF [slave id == #26H]     // check if id matched?
    BEGIN
        [Check if RI ready]
        [Start to receive data from the master]
        [Store the data into the slave's output port]
        [Enable SM2 to listen to the master again]
    END
    ELSE IF [slave id == #27H]
    BEGIN
        [Check if RI ready]
```

```
        [Start to receive data from the master]
        [Store the data into the slave's output port]
        [Enable SM2 to listen to the master again]
    END
    ELSE IF [slave id == #28H]
    BEGIN
        [Check if RI ready]
        [Start to receive data from the master]
        [Store the data into the slave's output port]
        [Enable SM2 to listen to the master again]
    END
END

Slave 4:
[serial mode 2]
[SM2 = 1]                   // the slaves should be able to receive address bit
[REN = 1]
[SMOD = 0]

WHILE[1]                    // always doing the same work
BEGIN
    [Check if RI ready]         // prepare to receive slave id
    [Receive ID from master]
    [Store the data into the accumulator of the slave for checking ID]
    [Clear SM2 to get the actual data from the master]       // disable the effect of the
ninth bit
    IF [slave id == #26H]     // check if id matched?
    BEGIN
        [Check if RI ready]
        [Start to receive data from the master]
        [Store the data into the slave's output port]
        [Enable SM2 to listen to the master again]
    END
    ELSE IF [slave id == #27H]
    BEGIN
        [Check if RI ready]
        [Start to receive data from the master]
        [Store the data into the slave's output port]
        [Enable SM2 to listen to the master again]
    END
    ELSE IF [slave id == #28H]
    BEGIN
        [Check if RI ready]
        [Start to receive data from the master]
        [Store the data into the slave's output port]
        [Enable SM2 to listen to the master again]
    END
END
```

(b) Finally, based on (a), design the main and serial port programs for the master and each slave 8051s.

ASM FOR MASTER:

```
                MOV     SCON,   #52H    ; mode 2, SM2 = 0, REN = 1
                ANL     PCON,   #72H    ; SMOD = 0
                MOV     R0,     #20H    ; initialize pointer

INPUT:          CJNE    R0,#26H, NEXT1  ; input done, continue to output
                SJMP    OUTPUT
NEXT1:          ; sending address bit to input slaves
                SETB    TB8             ; send address bit mode
                JNB     TI,     $       ; TX empty?
                CLR     TI      ; start transmission
                MOV     SBUF,   R0      ; move the current slave id into sbuf
                ; receiving data from the input slaves
                JNB     RI,     $       ; RX empty?
                CLR     RI              ; start receiving
                MOV     @R0,    SBUF    ; store it to the register where R0 is
pointing to
                INC     R0              ; point to next register
                LJMP    INPUT           ; repeat until all input port received

OUTPUT:         CJNE    R0,  #2CH, NEXT2        ; output done, continue to input
                SJMP    INPUT
                ; sending address bit to select output slaves.
NEXT2:          SETB    TB8     ; send address bit mode
                JNB     TI,     $       ; TX empty?
                CLR     TI      ; start transmission
                MOV     SBUF,   R0      ; send the current slave id into sbuf
                ; sending data to corresponding output slave
                CLR     TB8     ; send data mode
                JNB     TI,     $       ; TX ready?
                CLR     TI      ; start transmission
                MOV     SBUF,   @R0     ; send the output data to output slave
                INC     R0      ; point to next position
                LJMP    OUTPUT          ; repeat until all output data are outputted
                end
```

ASM for SLAVE 1:

```
; This program is for input slave. It will first receive the id from the master and send
the input data to the master
        ; initialize
                MOV     SCON,   #72H    ; mode 2, SM2 = 1 for RB8, REN = 1
                ANL     PCON,   #7FH    ; SMOD = 0
```

```
        ; start to receive slave id
INPUT:  JNB     RI,     $       ; RX empty?
        CLR     RI              ; start receiving
        MOV     A,      SBUF    ; receive slave ID
        CJNE    A,  #20H,   MEM21   ; if not correct id, check next port
        ; correct id, start to transmit to master
        JNB     TI,     $       ; TX ready?
        CLR     TI
        MOV     SBUF,   P0      ; start to transmit data from p0
        LJMP    INPUT
MEM21:  CJNE    A, #21H, MEM22  ; if not correct id, check next port
        JNB     TI,     $       ; correct id, TX ready?
        CLR     TI
        MOV     SBUF,   P1  ; send data from p1
        LJMP    INPUT
MEM22:  CJNE    A, #22H,   INPUT ; if all id don't match, receive a new id
        JNB     TI,     $   ; TX ready?
        CLR     TI      ; start to transmit
        MOV     SBUF,   P2  ; correct id, send p2 to master
        SJMP    INPUT       ; receive a new ID
```

ASM for SLAVE 2:

; This program is for input slave. It will first receive the id from the master and send the input data to the master

```
        ; initialize
        MOV     SCON,   #72H    ; mode 2, SM2 = 1 for RB8, REN = 1
        ANL     PCON,   #7FH    ; SMOD = 0
        ; start to receive slave id
INPUT:  JNB     RI,     $       ; RX empty?
        CLR     RI              ; start receiving
        MOV     A,      SBUF    ; receive slave ID
        CJNE    A, #23H, MEM24  ; if not correct id, check next port
        ; correct id, start to transmit to master
        JNB     TI,     $       ; TX ready?
        CLR     TI
        MOV     SBUF,   P0      ; start to transmit data from p0
        LJMP    INPUT
MEM24:  CJNE    A, #24H, MEM25  ; if not correct id, check next port
        JNB     TI,     $       ; correct id, TX ready?
        CLR     TI
        MOV     SBUF,   P1  ; send data from p1
        LJMP    INPUT
MEM25:  CJNE    A, #25H,   INPUT ; if all id don't match, receive a new id
        JNB     TI,     $   ; TX ready?
        CLR     TI      ; start to transmit
        MOV     SBUF,   P2  ; correct id, send p2 to master
        SJMP    INPUT       ; receive a new ID
```

ASM for slave 3

```
; This program is for output slaves. It will receive the slave id from the master and if
id matched. If id matched, it will receive the data from the master and act as a output
port of the master.
                MOV     SCON,   #70H    ; mode 2, SM2 = 1 for RB8, REN = 1
                ANL     PCON,   #7FH    ; SMOD = 0
                ; receive slave id from master
OUTPUT:         JNB     RI,     $       ; RX ready?
                CLR     RI              ; start receiving
                MOV     A,      SBUF    ; get id
                CJNE    A, #26H, MEM27  ; jump if not correct id
                ; if right slave, start to receive the output data from the master
                CLR     SM2             ; receive data mode
                JNB     RI,     $       ; RX ready?
                CLR     RI              ; start receiving
                MOV     P0,     SBUF    ; put received output into output port
                SETB    SM2     ; output received, prepare to listen to the next
address
                LJMP    OUTPUT

MEM27:          CJNE    A, #27H, MEM28  ; jump if not correct id
                ; if right slave, start to receive the output data from the master
                CLR     SM2             ; receive data mode
                JNB     RI,     $       ; RX ready?
                CLR     RI              ; start receiving
                MOV     P1,     SBUF    ; put received output into output port
                SETB    SM2     ; output received, prepare to listen to the next
address
                LJMP    OUTPUT

MEM28:          CJNE    A, #28H, OUTPUT; all id not match, listen to the master
again
                ; if right slave, start to receive the output data from the master
                CLR     SM2             ; receive data mode
                JNB     RI,     $       ; RX ready?
                CLR     RI              ; start receiving
                MOV     P1,     SBUF    ; put received output into output port
                SETB    SM2             ; output received, prepare to listen to the
next address
                LJMP    OUTPUT
```

ASM for SLAVE 4

```
; This program is for output slaves. It will receive the slave id from the master and if
id matched. If id matched, it will receive the data from the master and act as a output
port of the master.
                MOV     SCON,   #70H    ; mode 2, SM2 = 1 for RB8, REN = 1
                ANL     PCON,   #7FH    ; SMOD = 0
```

```
                ; receive slave id from master
OUTPUT:         JNB     RI,     $       ; RX ready?
                CLR     RI              ; start receiving
                MOV     A,      SBUF    ; get id
                CJNE    A, #29H, MEM3A  ; jump if not correct id
                ; if right slave, start to receive the output data from the master
                CLR     SM2             ; receive data mode
                JNB     RI,     $       ; RX ready?
                CLR     RI              ; start receiving
                MOV     P0,     SBUF    ; put received output into output port
                SETB    SM2     ; output received, prepare to listen to the next
address
                LJMP    OUTPUT

MEM3A:          CJNE    A, #3AH, MEM3B ; jump if not correct id
                ; if right slave, start to receive the output data from the master
                CLR     SM2             ; receive data mode
                JNB     RI,     $       ; RX ready?
                CLR     RI              ; start receiving
                MOV     P1,     SBUF    ; put received output into output port
                SETB    SM2     ; output received, prepare to listen to the next
address
                LJMP    OUTPUT

MEM3B:          CJNE    A, #3BH, OUTPUT       ; all id not match, listen to the
master again
                ; if right slave, start to receive the output data from the master
                CLR     SM2             ; receive data mode
                JNB     RI,     $       ; RX ready?
                CLR     RI              ; start receiving
                MOV     P1,     SBUF    ; put received output into output port
                SETB    SM2             ; output received, prepare to listen to the
next address
                LJMP    OUTPUT
```