UNIVERSIDAD DE ORIENTE

NÚCLEO DE ANZOÁTEGUI

ESCUELA DE INGENIERÍA Y CIENCIAS APLICADAS

DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS

SISTEMAS DE MICROPROCESADORES



**EMULACIÓN DEL PROTOCOLO I2C EN EL MICROPROCESADOR 8085 DE INTEL**

|  |  |
| --- | --- |
| **Profesor:** | **Bachiller** |
| Pedro Rene Cabrera | Claudia Rodriguez.  C.I.: 27.943.668 |
|  |  |

Barcelona, agosto de 2024

**BASES TEÓRICAS**

* **I2C**

El protocolo de comunicación serial I2C (Inter-Integrated Circuit) se utiliza para la transmisión de datos entre dispositivos a través de dos líneas principales: SDA (Serial Data Line) y SCL (Serial Clock Line).

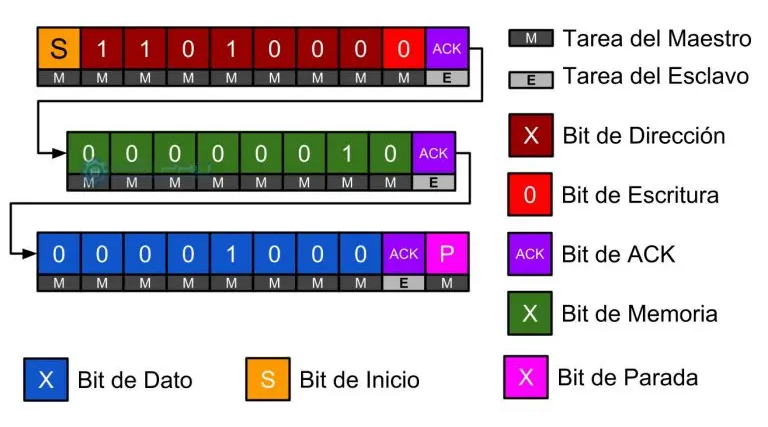
* **Línea SDA (Serial Data):** Se encarga de transmitir los bits de datos de manera serial entre el dispositivo maestro y el esclavo.
* **Línea SCL (Serial Clock):** Proporciona los pulsos de reloj necesarios para sincronizar la transmisión de datos. La línea SCL es controlada por el dispositivo maestro.

En una comunicación I2C, se identifican dos roles principales:

* **Dispositivo Maestro:** Controla el bus de comunicación, genera el pulso de reloj y coordina la transferencia de datos. En este caso, el microprocesador 8085 actúa como el dispositivo maestro.
* **Dispositivo Esclavo:** Recibe y responde a las solicitudes del dispositivo maestro.

El protocolo I2C define una serie de bits importantes para la correcta transmisión de datos:

* **Inicio (Start - S):** Señala el comienzo de la comunicación. El maestro genera una condición de inicio para iniciar la transmisión de datos.
* **Parada (Stop - P):** Marca el final de la comunicación. El maestro genera una condición de parada para concluir la transmisión.
* **Confirmación (ACK - Acknowledge):** Indica que el dato enviado ha sido recibido correctamente por el receptor.
* **No Confirmación (NACK - No Acknowledge):** Se utiliza para indicar que el dato no ha sido recibido correctamente o que no hay más datos para enviar.
* **Lectura/Escritura (L/W):** Determina si el maestro desea leer datos del esclavo o escribir datos en el esclavo.
* **Dirección del dispositivo esclavo:** Un campo de 7 bits para identificar la dirección del dispositivo esclavo. Algunos dispositivos pueden usar 8 o 16 bits para la dirección.
* **Dirección del registro en el dispositivo esclavo.**
* **Datos:** Un campo de 8 bits que representa la información transmitida entre el maestro y el esclavo.



**Figura 1.** Ejemplo de la trama de datos en el protocolo I2C para la escritura.



**Figura 2.** Ejemplo de la trama de datos en el protocolo I2C para la lectura.

**REQUERIMIENTOS PARA DISEÑAR EL PROTOCOLO**

Para el desarrollo del código de simulación del protocolo de comunicación I2C utilizando el microprocesador 8085, se deben cumplir los siguientes requerimientos:

1. *Conocimientos en el Uso del Simulador 8085:*

Familiaridad con el entorno y las herramientas del simulador del microprocesador 8085, incluyendo su configuración y la ejecución de simulaciones.

1. *Conocimientos del Protocolo de Comunicación Serial I2C:*

Comprensión detallada del protocolo I2C, incluyendo la estructura de las tramas de datos, las señales de inicio y parada, y los mecanismos de confirmación (ACK) y no confirmación (NACK).

1. *Entendimiento de los Pines en el Simulador:*

Conocimiento de los pines correspondientes a Serial Input Data (SID) y Serial Output Data (SOD) en el simulador. Estos pines son fundamentales para la observación y simulación del protocolo I2C.

1. *Comprensión de las Instrucciones del Microprocesador 8085:*

Familiaridad con el conjunto de instrucciones del microprocesador 8085 y su impacto en los registros y las banderas del procesador durante la ejecución de instrucciones relacionadas con la comunicación I2C.

1. *Diseño de la Lógica de Comunicación I2C:*

Capacidad para implementar la lógica necesaria para emular el protocolo I2C en el código de simulación. Esto incluye la correcta generación y gestión de señales de reloj, así como la transmisión y recepción de datos.

**MANIPULACIÓN DEL PROGRAMA**

Para llevar a cabo la simulación y emulación del protocolo I2C, se han establecido los siguientes elementos y procedimientos:

1. *Datos de Prueba Utilizados:*

* Dirección del Dispositivo Esclavo + Bit de Escritura/Lectura: Se especifica la dirección del dispositivo esclavo con el que se desea comunicar más la operación que se va a realizar.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| Dirección del dispositivo esclavo | | | | | | | R / W |

* Registro del Dispositivo Esclavo: Se determina el registro dentro del dispositivo esclavo al que se accederá para leer o escribir datos.
* Dato a Enviar: Se define el dato que se enviará al dispositivo esclavo para su procesamiento.

1. *Subrutina Principal:*

* Subrutina MAIN: En esta subrutina se implementará la lógica del protocolo I2C. Esta sección del programa se encargará de ejecutar las operaciones necesarias para emular el protocolo, incluyendo la transmisión y recepción de datos.

1. *Simulación de ACK/NACK:*

* Control del Pin SID: Se utilizará el pin SID (Serial Input Data) para simular la recepción de señales de ACK (Acknowledgement) y NACK (Not Acknowledge). El estado del pin SID se ajustará para representar las respuestas del dispositivo esclavo durante la comunicación.

1. *Envío de bits desde el dispositivo esclavo:*

* Control del Pin SOD: Se activará la casilla del pin SOD para enviar un 1 o se desactivará para enviar un 0. Se envía desde el bit menos significativo al más significativo.

**Algoritmo**

1. El programa inicia en la dirección de memoria 3000H.
2. Si se desea, los datos a utilizar se pueden modificar. Estos se encuentran al inicio del código ASM en el apartado de “Direcciones y datos de ejemplo” en donde:

* **SLAVE\_ADDR:** Es utilizado para indicar en los bits desde el D7 al D1 la dirección del dispositivo esclavo y el bit D0 para indicar la operación a realizar.
* **MEMORY\_ADDR:** Es una variable en donde se almacena la dirección de memoria dentro del dispositivo esclavo en donde se leerá o se almacenará la información.
* **DATA\_TO\_SEND:** En caso de ser una operación de escritura, en esta variable se almacena los datos a escribir en el **MEMORY\_ADDR.**
* **READ\_WRITE:** Esta variable indicará que tipo de operación se realizará.

1. Se inicia la ejecución del programa, es decir, se envía el bit de inicio. Se visualizarán llamados a las subrutinas PULSE\_SCL, SCL\_HIGH y SCL\_LOW. Esto es con objetivos de simular la línea del SCL. Esta simulación se ve en los puertos de salida 00H del simulador. Se recuerda que esta es controlada **solamente** por el dispositivo maestro (**8085**).
2. Se envía la dirección del dispositivo esclavo, esto se hace independientemente del tipo de operación. Este corresponde desde el bit D7 al D1. El D0 corresponde al bit de escritura o lectura.
3. Se espera el bit de ACK/NACK de parte del esclavo. Este se ingresa mediante la casilla / pin SID. Para indicar que el dispositivo esclavo recibió la información de manera correcta, enviará un ACK. Este se envía mandando un cero. En caso de NACK, se activa la casilla para indicar un 1.
   1. En caso del NACK, se tienen 4 intentos de reenvío. Si no se logró enviar, se llama a la subrutina I2C\_STOP para que de fin a la comunicación.
   2. Si el dispositivo maestro recibe el ACK de parte del esclavo, continúa la comunicación.
4. Se procede a enviar la dirección del registro desde donde se va a leer o escribir. Esta dirección de memoria es de 8 bits. De igual forma a cuando se envía la dirección del dispositivo esclavo, se tiene un numero de reintentos en caso de recibir un NACK y si se recibe el ACK, se continua.
5. Dependiendo del tipo de operación, se tienen dos alternativas:
   1. **Operación Escritura:** 
      1. Se envían los datos mediante el pin SOD. Desde el bit menos significativo hasta el bit más significativo.
      2. Luego de enviar los datos, se necesita recibir el ACK / NACK del dispositivo esclavo a el dispositivo maestro. Para ello, se desactiva la casilla del SID en caso de enviar el ACK.
      3. Se envía el bit de parada con la subrutina I2C\_STOP.
   2. **Operación Lectura:**
      1. Para simular esta operación, se envía otro bit de inicio con la subrutina de I2C\_START.
      2. Se vuelve a enviar la dirección del dispositivo esclavo y el ultimo bit (D0) va a ser igual a 1. Se recibe el ACK del dispositivo esclavo indicando que se recibió la información.
      3. El dispositivo esclavo envía la información al dispositivo maestro. Para ello, se activa o desactiva la casilla del pin SID, se enviará desde el menos significativo al más significativo.
      4. Para terminar la comunicación se envía un NACK por el SOD al dispositivo esclavo, para indicar que el dispositivo maestro no necesita más bit y se envía el bit de STOP, de parte del dispositivo maestro al esclavo.
6. **Finalización de la comunicación.** Por último, la ejecución se detiene con la instrucción **HLT.**

**Código**

; Código para emulación del protocolo I2C en el microprocesador 8085

; Utilizando los pines SID y SOD y las instrucciones RIM y SIM

; Definir las direcciones de memoria para las variables

.DEFINE

SDA\_DATA 01H ; Variable para el dato en SDA

SCL\_PIN 00H ; Variable para el pin de reloj SCL

; Direcciones y datos de ejemplo

SLAVE\_ADDR AEH ; Dirección del dispositivo esclavo (7 bits) + Bit ; R/W

MEMORY\_ADDR 5BH ; Dirección de memoria (8 bits)

DATA\_TO\_SEND 9FH ; Dato a enviar (8 bits)

READ\_WRITE 00H ; Variable para el bit de Lectura/Escritura

; Inicio del programa principal

.org 3000H

MAIN:

MVI A, 00H ; 0 para escribir, 1 para leer

MOV READ\_WRITE, A

CALL I2C\_START ; Generar señal de inicio

CALL SEND\_SLAVE\_ADDR ; Enviar dirección del esclavo + Bit escritura / lectura

MOV A,B ; B almacena el bit de escritura / lectura

CPI 00H ; 00 H = escribir

JZ ESCRIBIR ; salta a la subrutina de escribir

CPI 01H ; 01 H = leer

JZ LEER ; salta a la subrutina leer

; En caso de que el bit READ\_WRITE = 00H

ESCRIBIR:

CALL SEND\_MEMORY\_ADDR ; Enviar dirección de memoria

CALL SEND\_DATA ; Enviar datos

CALL I2C\_STOP ; Generar señal de parada

HLT ; Detener el programa

; En caso de que el bit READ\_WRITE = 01H

LEER:

CALL SEND\_MEMORY\_ADDR ; Enviar dirección de memoria

CALL I2C\_START ; Bit de inicio repetido

CALL SEND\_SLAVE\_ADDR ; Enviar dirección del esclavo + Bit ; escritura / lectura

; HLT

CALL READ\_DATA ; Leer datos

CALL I2C\_STOP ; Generar señal de parada

HLT ; Detener el programa

.org 1000H

; Subrutina para generar una señal de inicio (START)

; Configuración inicial / Inicialización

I2C\_START:

MVI A, 00H ; Inicializar A

OUT SCL\_PIN ; Inicializar el pin SCL

CALL SCL\_HIGH ; Asegurar SCL está en alto

MVI A, C0H ; Asegurar SDA está en alto

SIM

MVI A, 40H ; Bajar SDA (Inicio)

SIM

CALL SCL\_LOW ; Bajar SCL

RET

; Subrutina para generar una señal de parada (STOP)

I2C\_STOP:

CALL SCL\_HIGH ; Subir SCL

MVI A, 40H ; Asegurar SDA está en bajo

SIM

MVI A, C0H ; Asegurar SDA está en alto

SIM

CALL SCL\_LOW ; Asegurar SCL está en bajo

MVI A, 00H ; Bajar SDA

RET

; Subrutina para generar un pulso de reloj en SCL

PULSE\_SCL:

CALL SCL\_HIGH ; Subir SCL

CALL SCL\_LOW ; Bajar SCL

RET

SCL\_HIGH:

MVI A, 01H

OUT SCL\_PIN ; SCL en alto

NOP

NOP

NOP

RET

SCL\_LOW:

MVI A, 00H

OUT SCL\_PIN ; SCL en bajo

NOP

NOP

NOP

RET

; Subrutina para enviar un bit

SEND\_BIT:

MOV A, SDA\_DATA ; Cargar el dato a enviar desde SDA\_DATA

RRC

MOV SDA\_DATA, A

ANI 80H

ORI 40H

SIM ; Colocar el bit en SOD

CALL PULSE\_SCL ; Generar un pulso de reloj en SCL

MOV A, SDA\_DATA ; Actualizar SDA\_DATA con el último valor ; enviado

OUT SDA\_DATA ; Colocar el bit en el puerto de salida SDA

RET

; Subrutina para leer un bit desde

; el SID. Es decir, el esclavo le envía un dato

; de 8 bits al dispositivo maestro.

READ\_BIT:

CALL PULSE\_SCL ; Generar un pulso de reloj en SCL

MVI D, 08H ; Limpia la señal del acumulador

RIM ; bit 0

SUB D ; Limpia bit 3 del acumulador

RRC ; Desplaza bit a la derecha

MOV B,A ; Almacenar bit obtenido / resultado en el ; registro B

RIM ; bit 1

SUB D

ADD B

RRC

MOV B,A

RIM; ; bit 2

SUB D

ADD B

RRC

MOV B,A

RIM ; bit 3

SUB D

ADD B

RRC

MOV B,A

RIM ; bit 4

SUB D

ADD B

RRC

MOV B,A

RIM ; bit 5

SUB D

ADD B

RRC

MOV B,A

RIM ; bit 6

SUB D

ADD B

RRC

MOV B,A

RIM ; bit 7

SUB D

ADD B

MOV B,A

MOV A,B

RET ; se regresa

; Subrutina para enviar un byte

SEND\_BYTE:

MVI E, 08H ; Enviar 8 bits

MOV A, SDA\_DATA ; Cargar el dato a enviar desde SDA\_DATA

MOV READ\_WRITE, A ; Copiar el byte completo al registro D

RAR ; Rotar el bit hacia la derecha (el primer bit ; ahora es el LSB)

MOV A, READ\_WRITE ; Mover el byte modificado al acumulador ; A

ANI 01H ; Máscara para aislar el primer bit (LSB)

MOV READ\_WRITE, A ; Almacenar el bit de escritura/lectura en ; el registro D

; Subrutina para enviar bit por bit

; hasta que el REG E = 00H

SEND\_LOOP:

RAR ; Rotar el bit hacia la derecha

CALL SEND\_BIT ; Enviar un bit

DCR E

JNZ SEND\_LOOP ; Repetir hasta que se envíen 8 bits

CALL READ\_ACK ; Leer el bit de ACK/NACK

RET

; Subrutina para leer un byte

READ\_BYTE:

MVI A, 00H ; Inicializar A

; Subrutina para leer un bit

; hasta que el reg. E = 00H

READ\_LOOP:

CALL READ\_BIT ; Leer un bit

MVI E, 01H

DCR E

JNZ READ\_LOOP ; Repetir hasta que se lean 8 bits

CALL SEND\_ACK ; Enviar un bit de ACK/NACK

RET

; Subrutina para leer ACK/NACK

READ\_ACK:

CALL SCL\_HIGH

MVI A,00H

RIM

ANI 80H

RRC

RRC

RRC

RRC

RRC

RRC

RRC

CPI 00H ; Comparar con 0 (ACK)

JZ ACK\_RECEIVED ; Si es 0, es ACK

CALL REINTENTOS\_CONTEO

JMP NACK\_RECEIVED ; Si no es 0, es NACK

REINTENTOS\_CONTEO:

MVI D, 04H ; Nmro de reintentos

RET

ACK\_RECEIVED: ; Código para manejar ACK

NOP ; Realizar acciones necesarias para ACK

CALL SCL\_LOW

RET

NACK\_RECEIVED: ; Código para manejar NACK

CALL SCL\_LOW

DCR D

JNZ SEND\_BYTE

JZ I2C\_STOP

RET

; Subrutina para enviar ACK

SEND\_ACK:

MVI A, 00H ; Preparar ACK

MOV SDA\_DATA, A

CALL SEND\_BIT ; Enviar ACK

RET

; Subrutina para enviar NACK

SEND\_NACK:

MVI A, 01H ; Preparar NACK

MOV SDA\_DATA, A

CALL SEND\_BIT ; Enviar NACK

RET

; Subrutina para enviar la dirección del esclavo y el bit R/W

SEND\_SLAVE\_ADDR:

MVI A, SLAVE\_ADDR

MOV B, A

MVI A, READ\_WRITE

RLC

ORA B

MOV SDA\_DATA, A

CALL SEND\_BYTE

RET

; Subrutina para enviar la dirección de memoria + el bit de escritura / lectura

; al dispositivo esclavo

SEND\_MEMORY\_ADDR:

MVI A, MEMORY\_ADDR

MOV SDA\_DATA, A

CALL SEND\_BYTE

RET

; Subrutina para enviar datos al dispositivo esclavo

SEND\_DATA:

MVI A, DATA\_TO\_SEND

MOV SDA\_DATA, A

CALL SEND\_BYTE

RET

; Subrutina para leer datos del dispositivo esclavo

READ\_DATA:

CALL READ\_BYTE

MOV A, SDA\_DATA ; Guardar el dato leído

CALL SEND\_NACK

RET