LCD Interfacing

LCD: liquid crystal displays

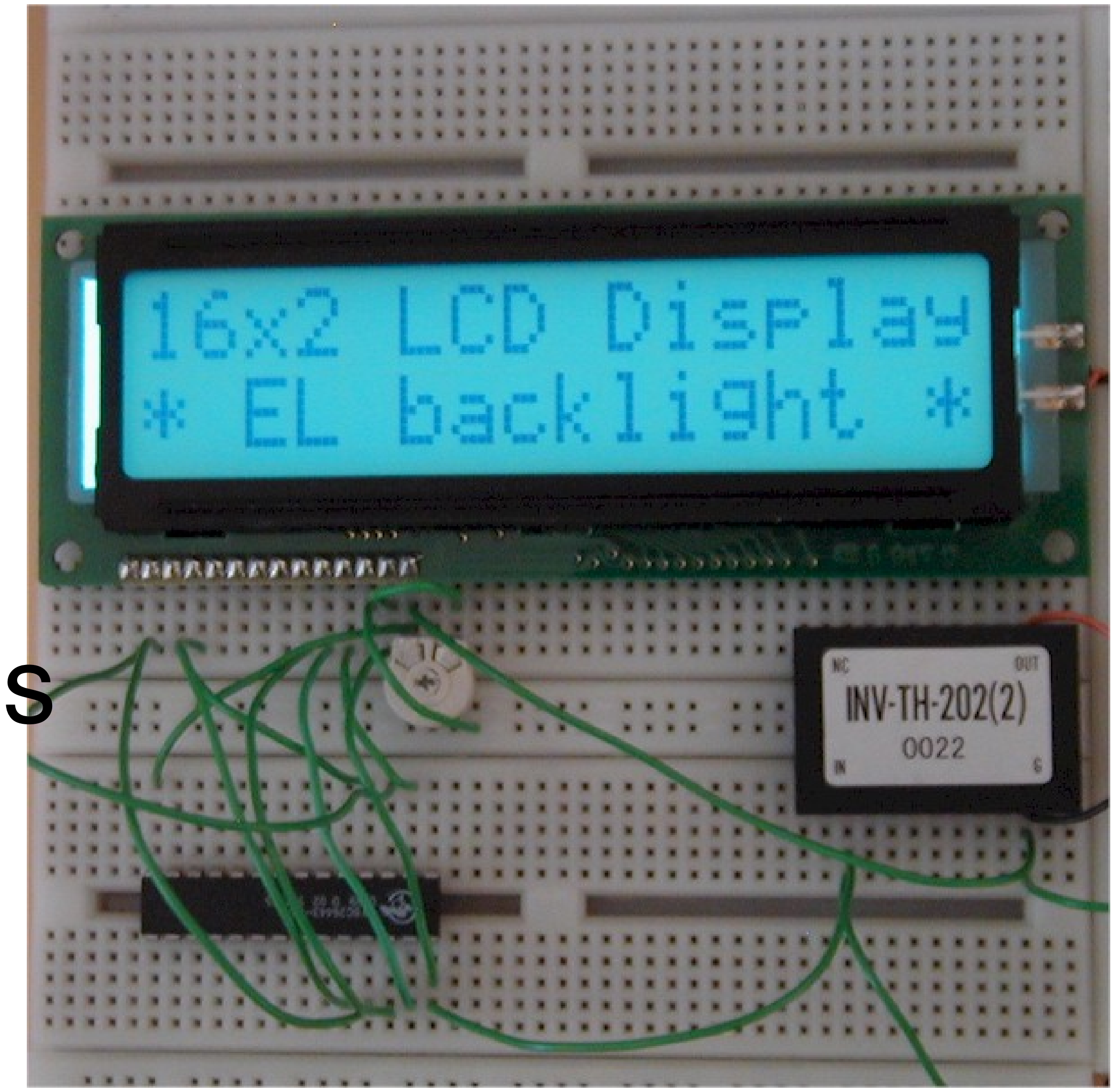
- Many kinds
  - Passive: multiplexed, simpler, slow refresh
  - Active: each pixel has its own transistor
- Related technologies
  - LED - light emitting diode, higher powe
  - OLED - organic LED, bright, low power

Segmented vs. Bitmapped LCD

- Segmented
  - 7 segments per digit, more for letters
  - Straightforward, but ugly
- Bitmapped
  - can create any shape
  - but many more dots to control!

Case study: I/O ports to control a 2-line LCD

- bitmapped font
- character display
  - has a cursor
  - diff. scrolling effects
  - blinking possible

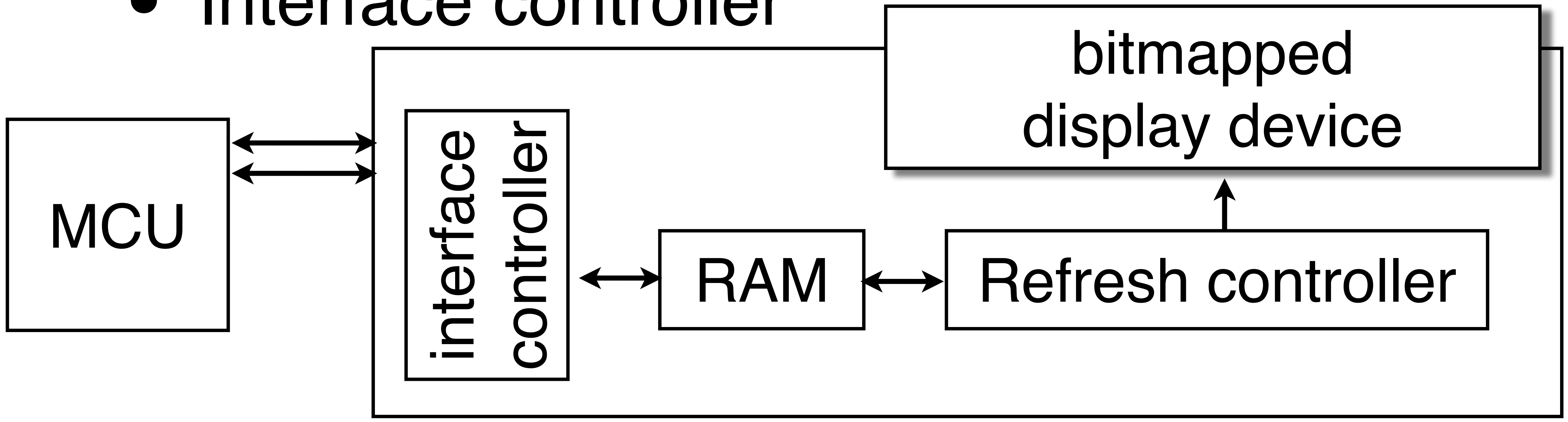


More details about the LCD

- Data sheet:
  - https://www.sparkfun.com/datasheets/LCD/HD44780.pdf
- Handling precautions (page 7)
  - connect unused pins to Vcc or GND avoid applying input signal w/out power, electrostatic discharge, direct sunlight, pressure, ...

What is in this LCD *module*

- Display device
- Refresh controller
- Display memory
- Interface controller



LCD Controller: Hitachi HD44780

- Nearly everybody uses it!
  - Same controller, at least same interface => 14 pin + 2 optional for backlight
  - Used inside different brands of LCDs
- Several sizes; some software configurable
  - 8x1, 16x1, 16x2, 20x1, 20x2, 20x4, 40x1, 40x2, ...

Capabilities of the LCD module

- 192 ROM chars, 8 user-defined chars
  - Compatible with ASCII subset
- Instruction functions
  - Display Clear, Cursor Home,
    Display on/off, Cursor on/off
    Char display Blink,
    Cursor Shift, Display Shift

Registers in the LCD controller

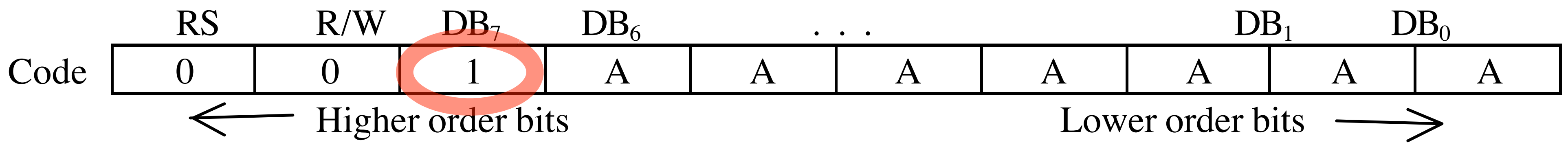
- IR (instruction register)
  - write-only, for command code
  - Also for display data or character-generation address
- DR (data register)
  - read/write, for Data read/written to RAM (either Display Data RAM or Character Generator RAM)

Display Data RAM (DD RAM)

- stores the 8-bit character code
  - up to 40 chars per line
- Addresses of the characters
  - example: 2 lines, 16 columns

0	1	2	3	4	5	6	7	8	9	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

  - As a transaction to set address,



	RS	R/W	DB_7	DB_6	. . .				DB_1	DB_0
Code	0	0	1	A	A	A	A	A	A	A

Busy Flag

- '1': busy
  - LCD module is performing operation
  - not accepting another instruction
- '0': not busy
  - ready to accept another operation
- LCD is a "slow" device

LCD Timing

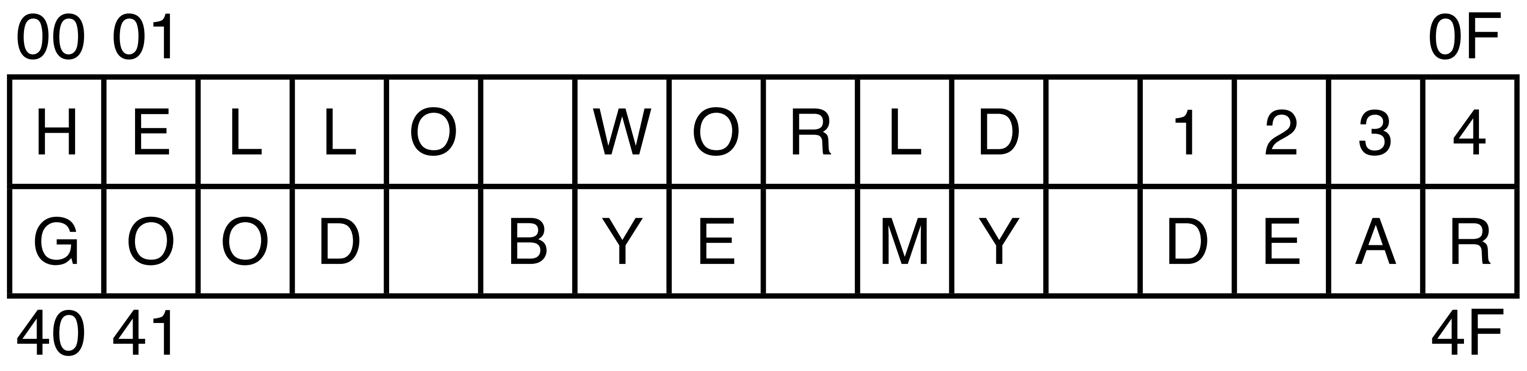
- LCD is a slow device!
  - Clearing screen, Return home => 1.64ms
  - @12MHz, that is 1640 instr. cycles!
  - Avg 40μs, still long => 40 instr. cycles
- Two ways to ensure not too fast
  - Delay sufficiently long
  - Check busy flag

Address Counter

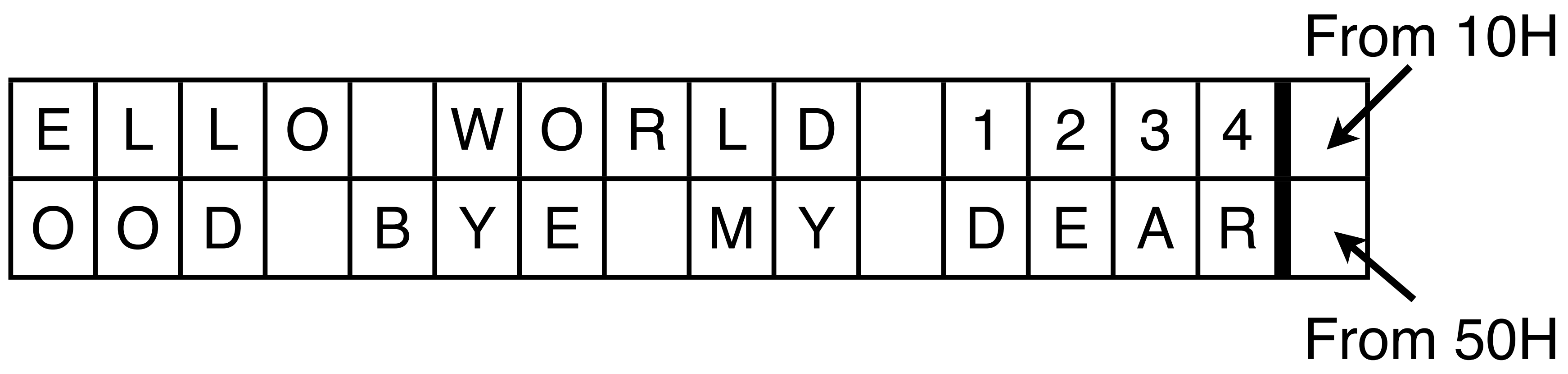
- Set using Set DD RAM Address instruction
- Automatically incremented or decremented
  - Address generation for sequential access
  - No need for MCU to generate each addr => more efficient
- Same counter for both DD and CG RAM

Option for DD shift (2 lines, 40 bytes/line)

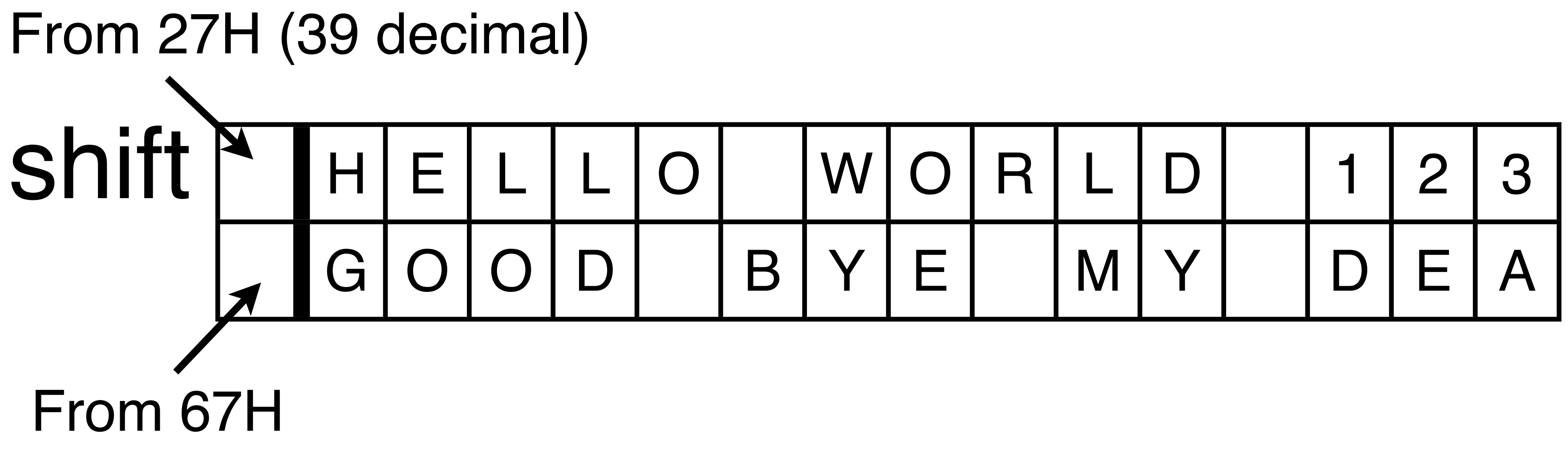
- Original


00
01
0F
H E L L O   W O R L D   1 2 3 4
G O O D   B Y E   M Y   D E A R
40
41
4F

- Left shift



- Right shift



LCD pins

- V_{CC}: supply power (+5V), V_{SS}: ground
- V_{EE}: LCD contrast control (analog)
- RS: Register select (in)
- $R/\overline{W}$: Read, ~write (1: read, 0: write) (in)
- E: Enable (in)
- DB0-DB7: data (in/out). (or D0-D7)
  8-bit or 4-bit interface

Connecting the 8051 to the LCD module

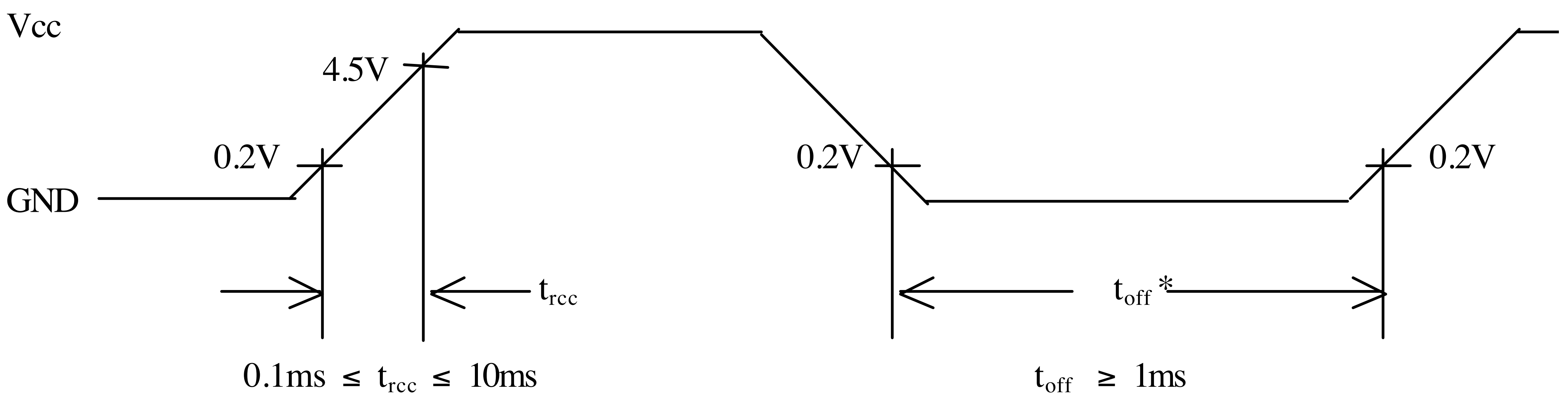
- Use GPIO pins
  - if enough pins, one 8-bit port for DB7..0, single bit ports for RS, R/W, E (11 total)
  - use 4 bits for DB7..4 (in 4-bit mode) single bits for RS, R/W, E
- Use memory-mapped I/O
  - map DB to data port

Initialization: two ways

- by Internal circuit reset
  - HD44780 has internal reset circuit
  - timing may be a bit tricky
- by software instructions
  - optional but highly recommended
  - Performed as a series of transactions

Initialization using internal reset circuit

- 8-bit, 1-line, display-off, cursor-off, no blink, +1 increment, no shift



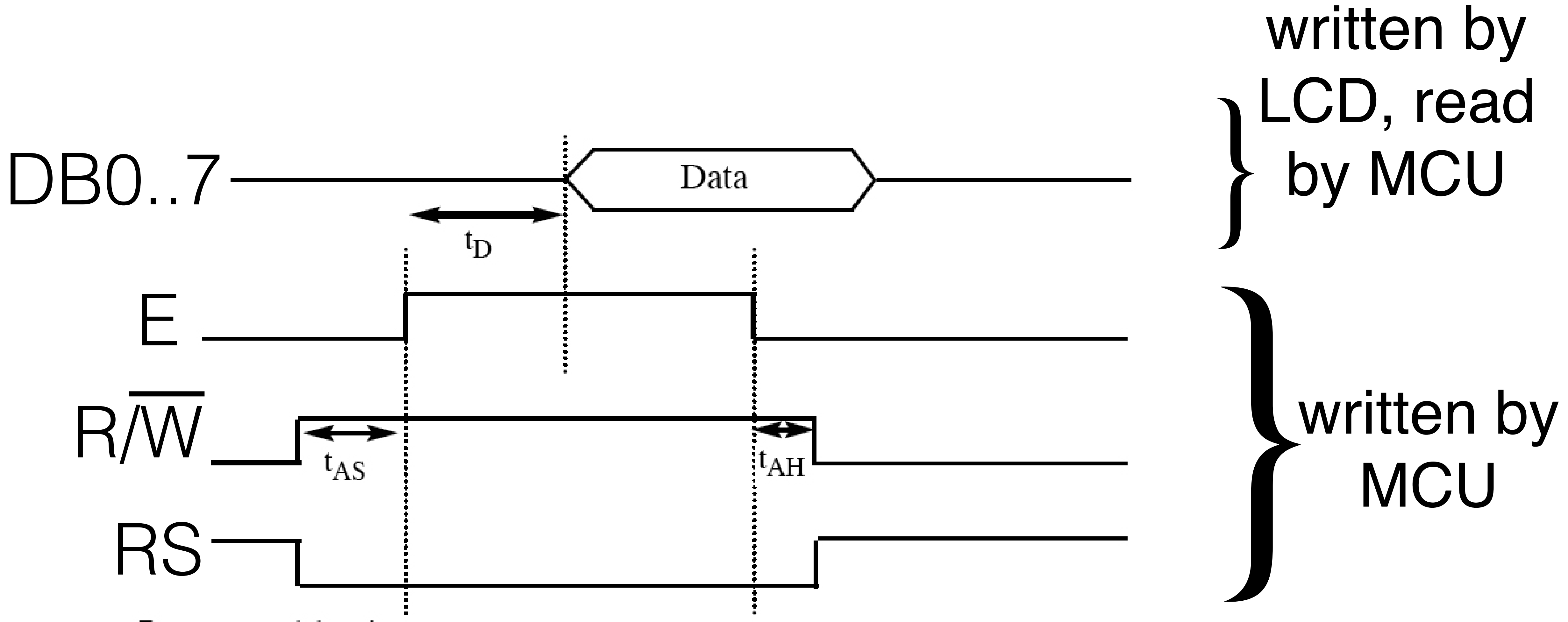
Concept: Bus Transaction

- An "atomic" sequence of signal changes
  - assert Control signals,
  - then Data transfer
- Purpose
  - Writing *Command* into the LCD module
  - Writing or reading *data* or *address*
- Representation: Timing Diagram

Transaction

- Sequence
  - RS = 0 (for Instruction Reg), 1 (for DR)
  - $R/\overline{W}$ = 1 for reading, = 0 for writing
  - Pulse the E signal (like a clock pulse)
  - Read or write DB on falling edge of E
- For EdSim51's LCD, see
  - http://www.edsim51.com/8051simulator/HD44780.pdf

IR-Read Transaction (8-bit, not EdSim)



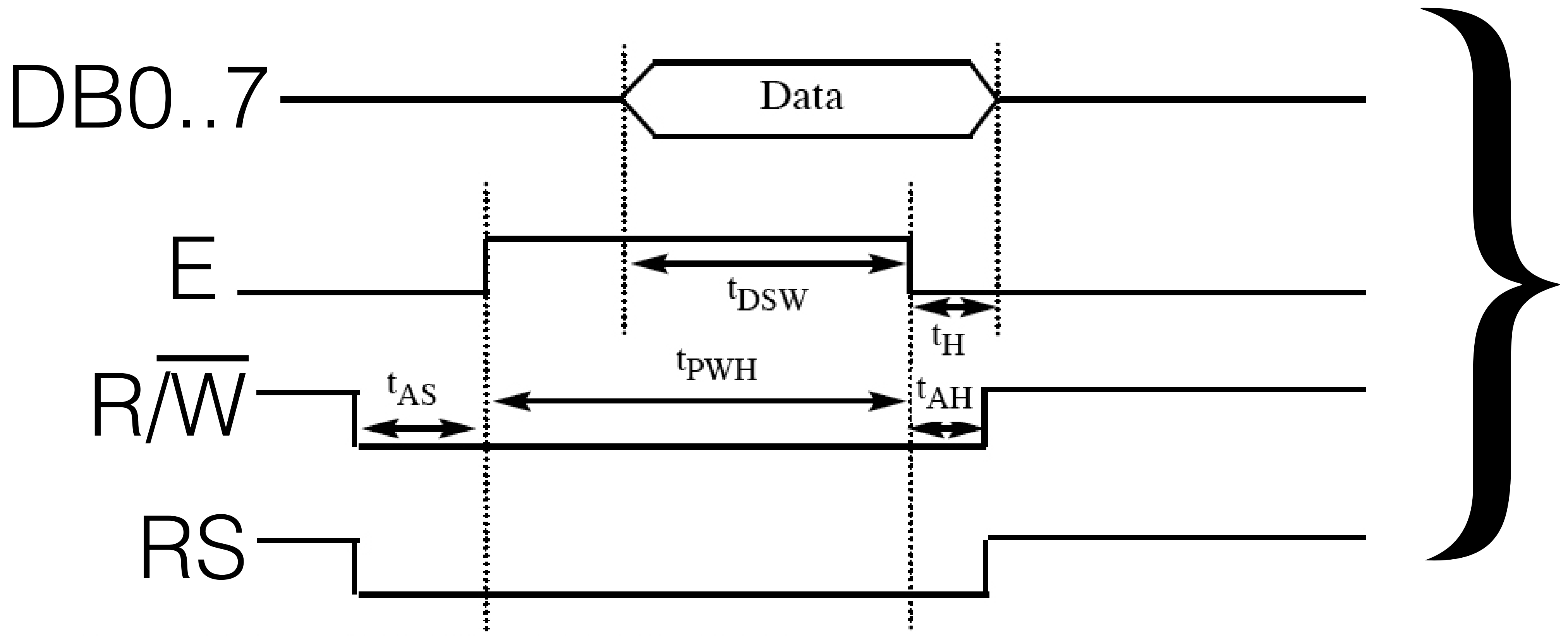
t_D = Data output delay time

t_{AS} = Setup time prior to E (going high) for both RS and R/W = 140 ns (minimum)

t_{AH} = Hold time after E has come down for both RS and R/W = 10 ns (minimum)

Note: Read requires an L-to-H pulse for the E pin.

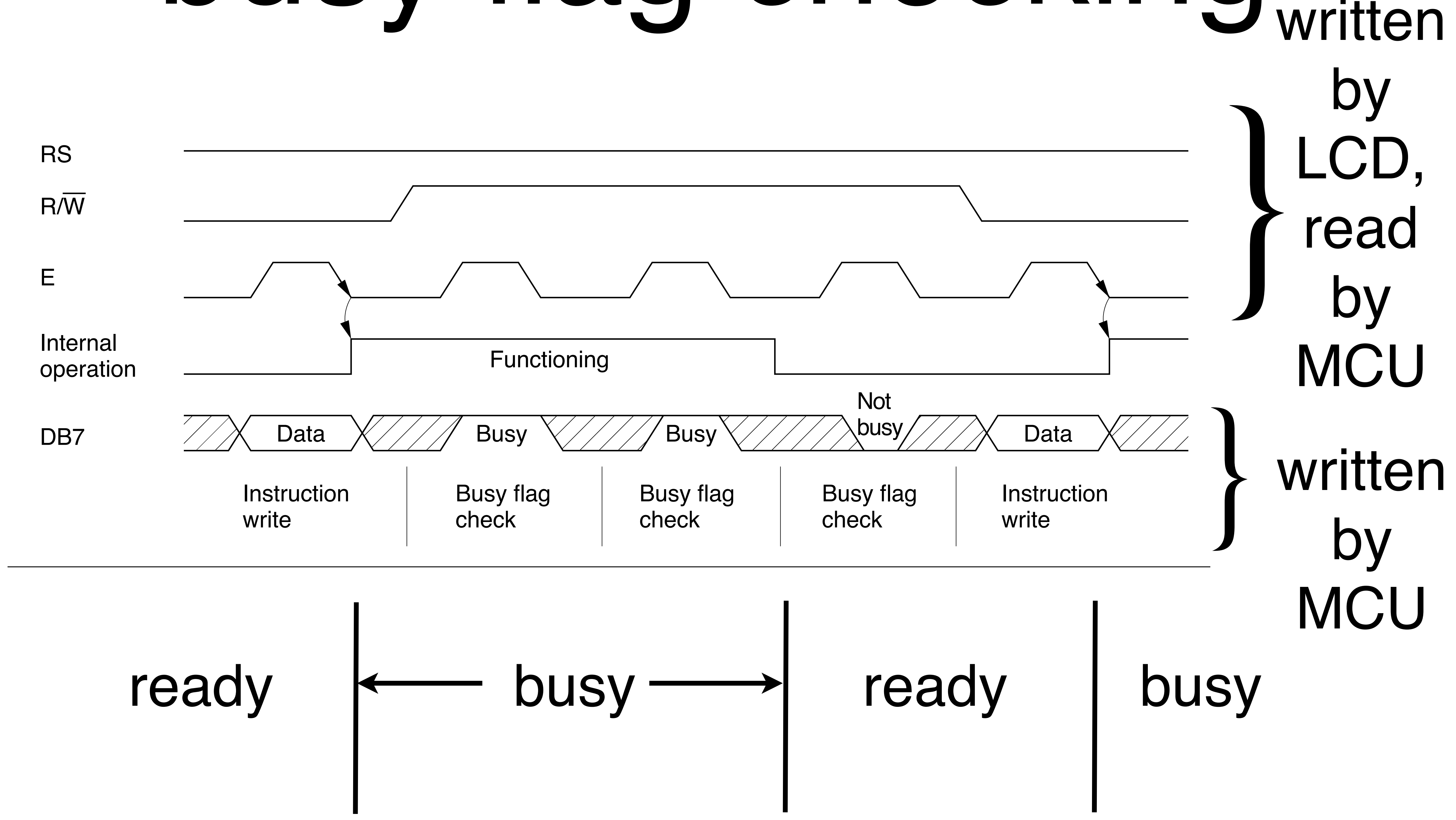
IR-Write Transaction (8-bit, not EdSim)



all written by MCU, sensed by LCD

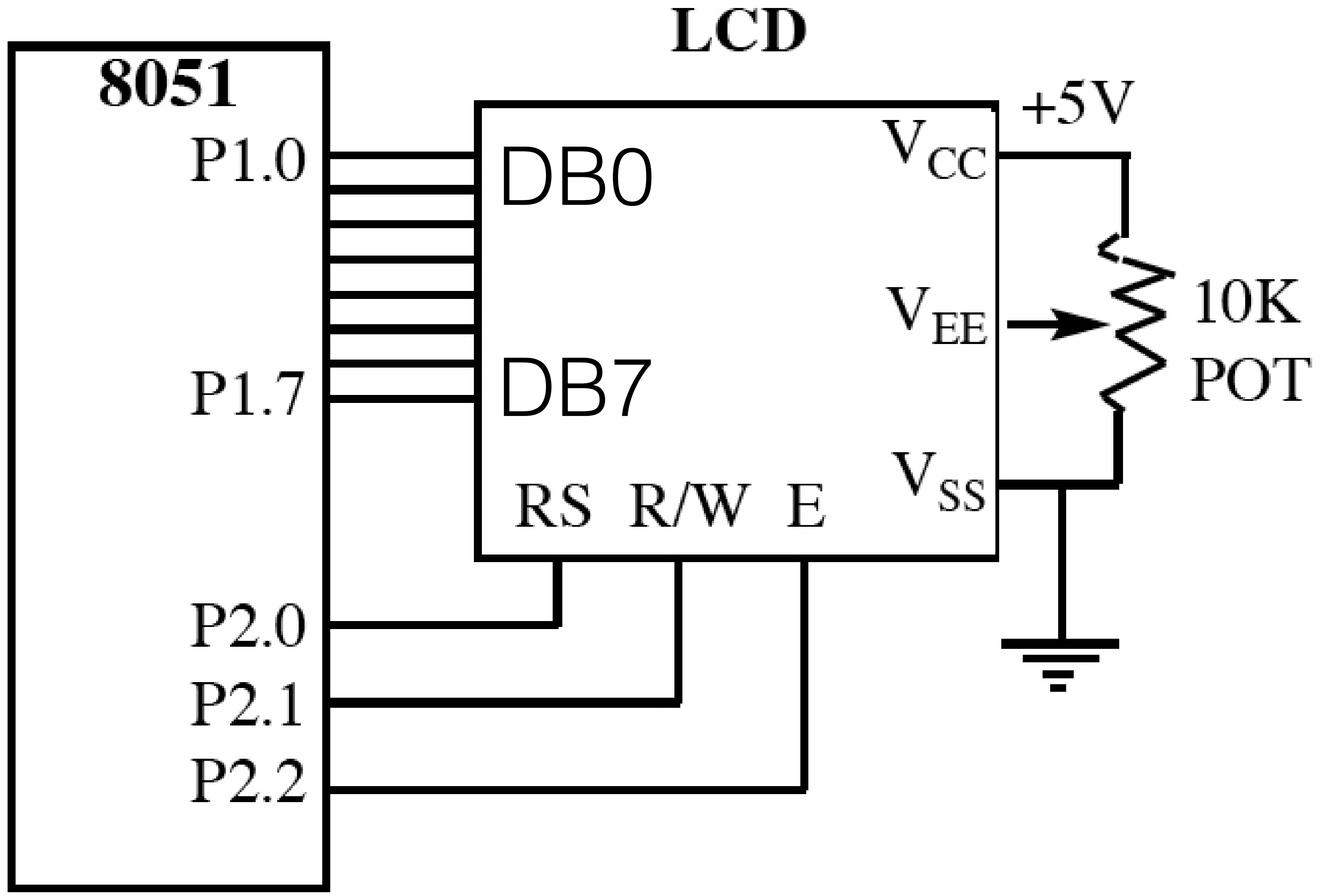
t_{PWH} = Enable pulse width = 450 ns (minimum)
t_{DSW} = Data setup time = 195 ns (minimum)
t_H = Data hold time = 10 ns (minimum)
t_{AS} = Setup time prior to E (going high) for both RS and R/W = 140 ns (minimum)
t_{AH} = Hold time after E has come down for both RS and R/W = 10 ns (minimum)

8-bit IR-Writes w/ busy flag checking



Hardware connection (8-bit, not EdSim)

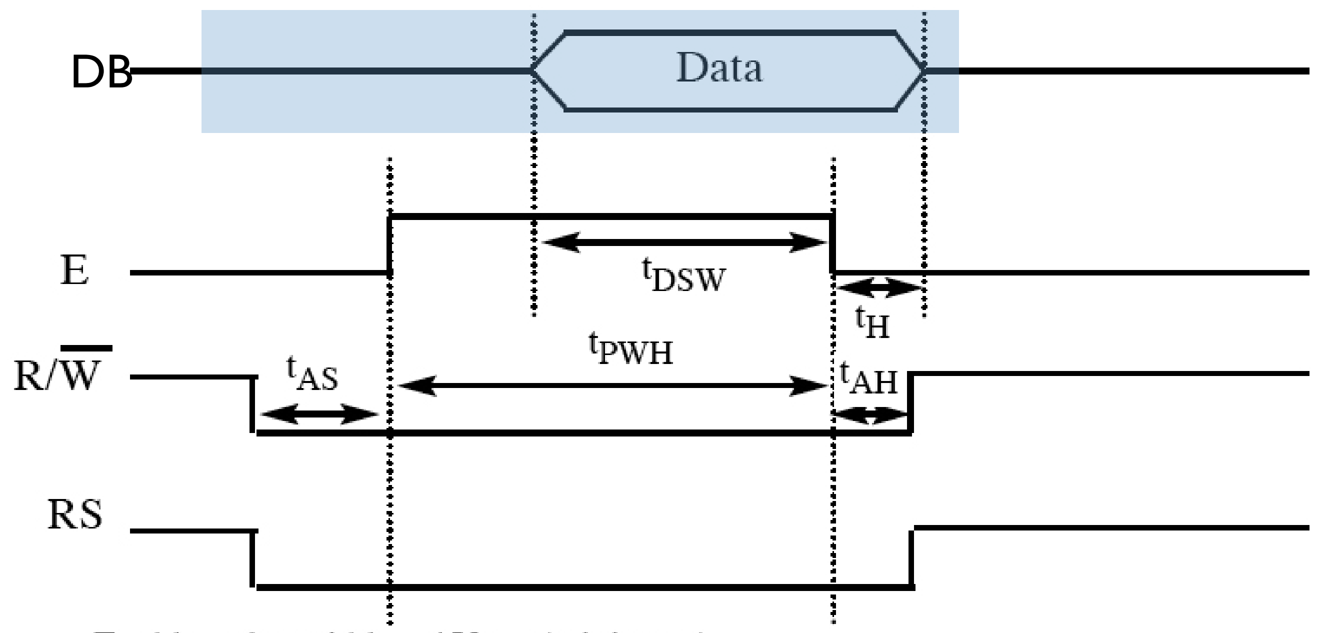
- DB7-DB0: an 8-bit I/O port
- RS, $R/\overline{W}$, E: single-bit ports



Subroutine for 8-bit IR-write transaction

- Pass parameter (instruction code) in the Accumulator

```
IR_WRT:
        MOV     P1, A    ;; DB
        CLR     P2.0     ;; RS
        CLR     P2.1     ;; RW
        SETB    P2.2     ;; E
        CALL    DELAY
        CLR     P2.2     ;; E
        RET
```

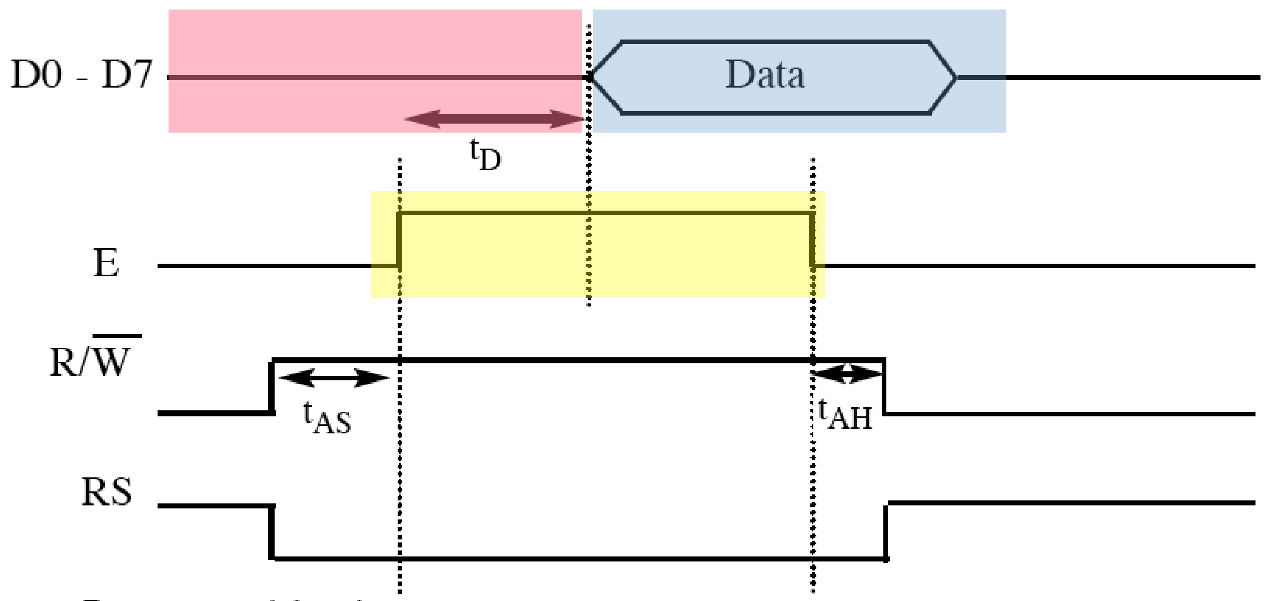


t_{PWH} = Enable pulse width = 450 ns (minimum)
t_{DSW} = Data setup time = 195 ns (minimum)
t_H = Data hold time = 10 ns (minimum)
t_{AS} = Setup time prior to E (going high) for both RS and R/W = 140 ns (minimum)
t_{AH} = Hold time after E has come down for both RS and R/W = 10 ns (minimum)

Subroutine for 8-bit IR-read transaction

- The only use: check busy flag (DB7)
- Done as a polling loop

```
CHKBUSY:
        SETB P1.7        ;; inp
        CLR  P2.0        ;; RS
        SETB P2.1        ;; RW
BACK:   SETB P2.2        ;; E
        CALL DELAY
        CLR  P2.2        ;; E
        JB   P1.7, BACK  ;; DB
        RET
```



t_D = Data output delay time

t_{AS} = Setup time prior to E (going high) for both RS and R/W = 140 ns (minimum)

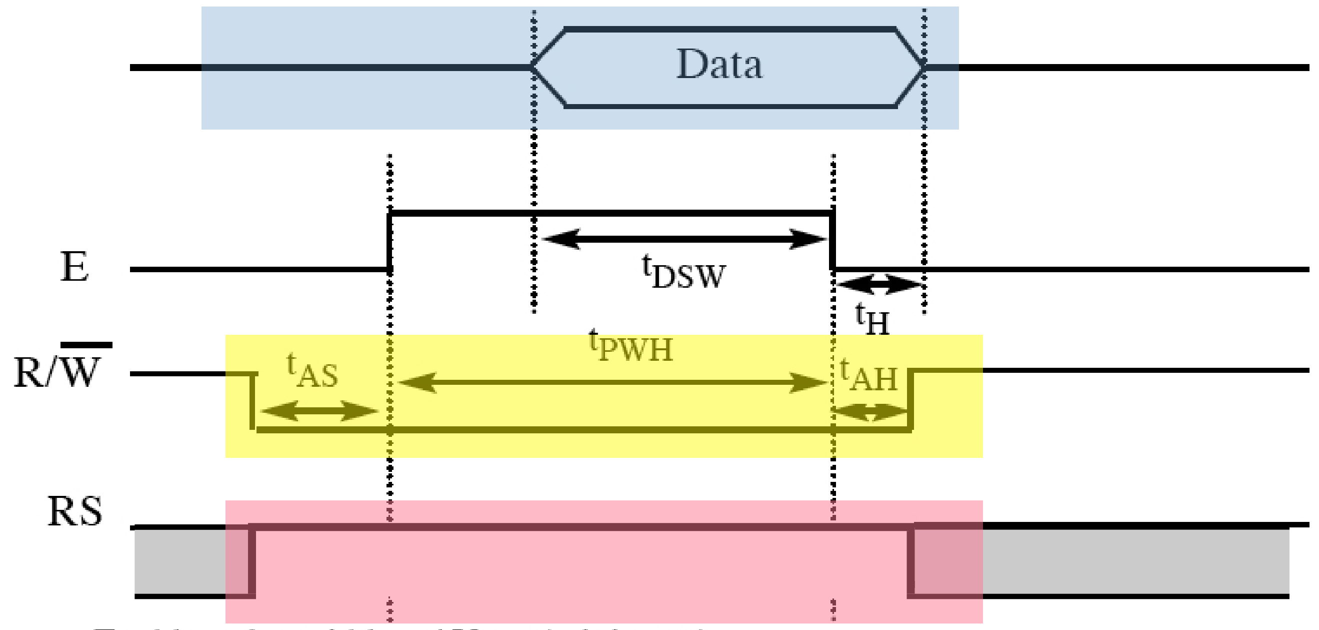
t_{AH} = Hold time after E has come down for both RS and R/W = 10 ns (minimum)

Note: Read requires an L-to-H pulse for the E pin.

Subroutine for 8-bit DR-write transaction

- Parameter
  - A: data to send to LCD

```
DR_WRT:
        MOV     P1, A   ;; DB
        SETB    P2.0    ;; RS
        CLR     P2.1    ;; RW
        SETB    P2.2    ;; E
        CALL    DELAY
        CLR     P2.2    ;; E
        RET
```

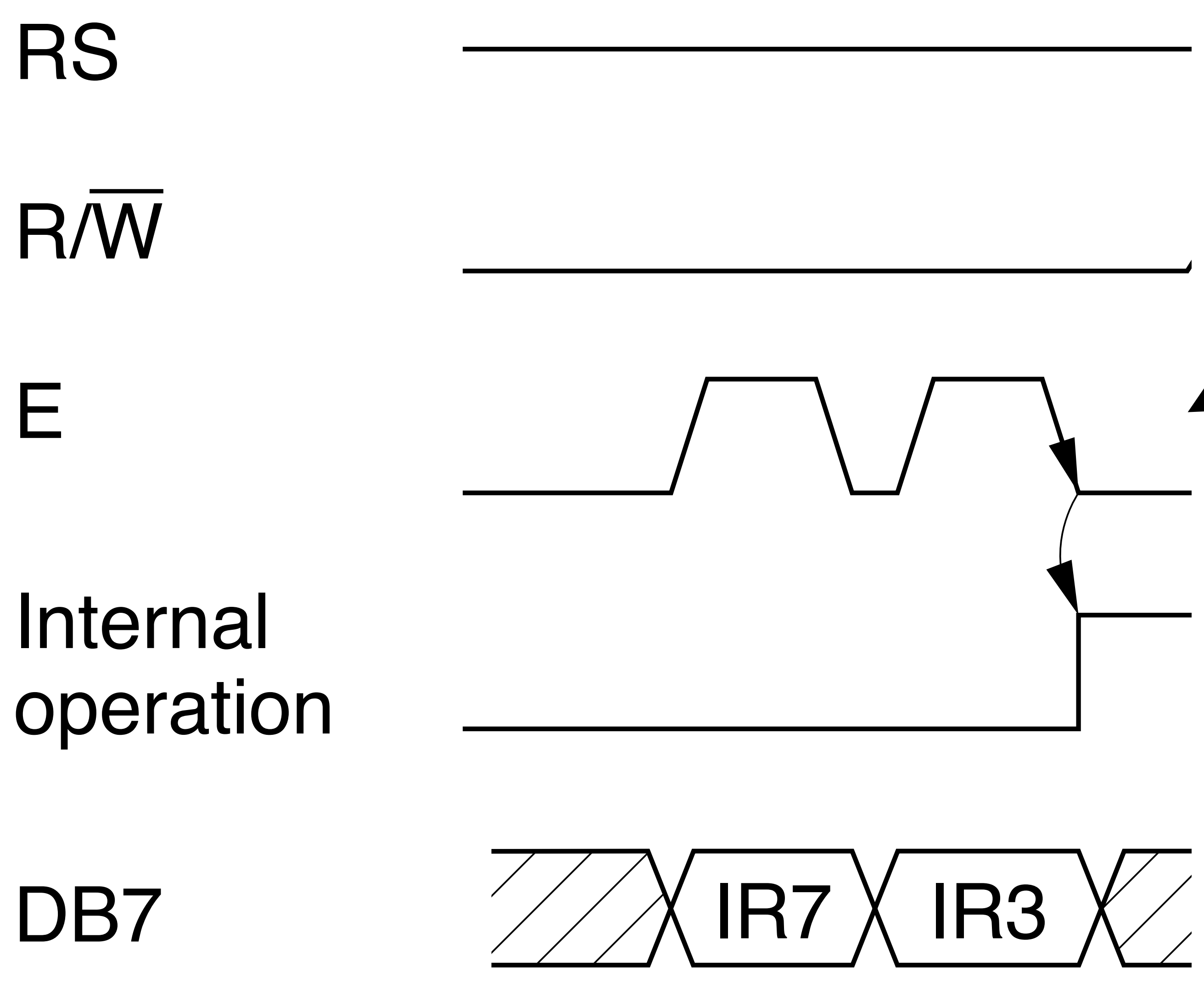


t_{PWH} = Enable pulse width = 450 ns (minimum)
t_{DSW} = Data setup time = 195 ns (minimum)
t_H = Data hold time = 10 ns (minimum)
t_{AS} = Setup time prior to E (going high) for both RS and R/W = 140 ns (minimum)
t_{AH} = Hold time after E has come down for both RS and R/W = 10 ns (minimum)

4-bit vs. 8-bit operation

- By default, 8-bit DB7..DB0 are used
  - issue: uses 8+3=11 GPIO pins on MCU!
- Can use 4-bit mode (e.g., Edsim51)
  - Use only DB7..DB4;
    <u>tie DB3..DB0 to low</u> (no need for GPIO)
  - Serialize: high-order, low-order nibbles
  - Uses 4+3=7 GPIO pins

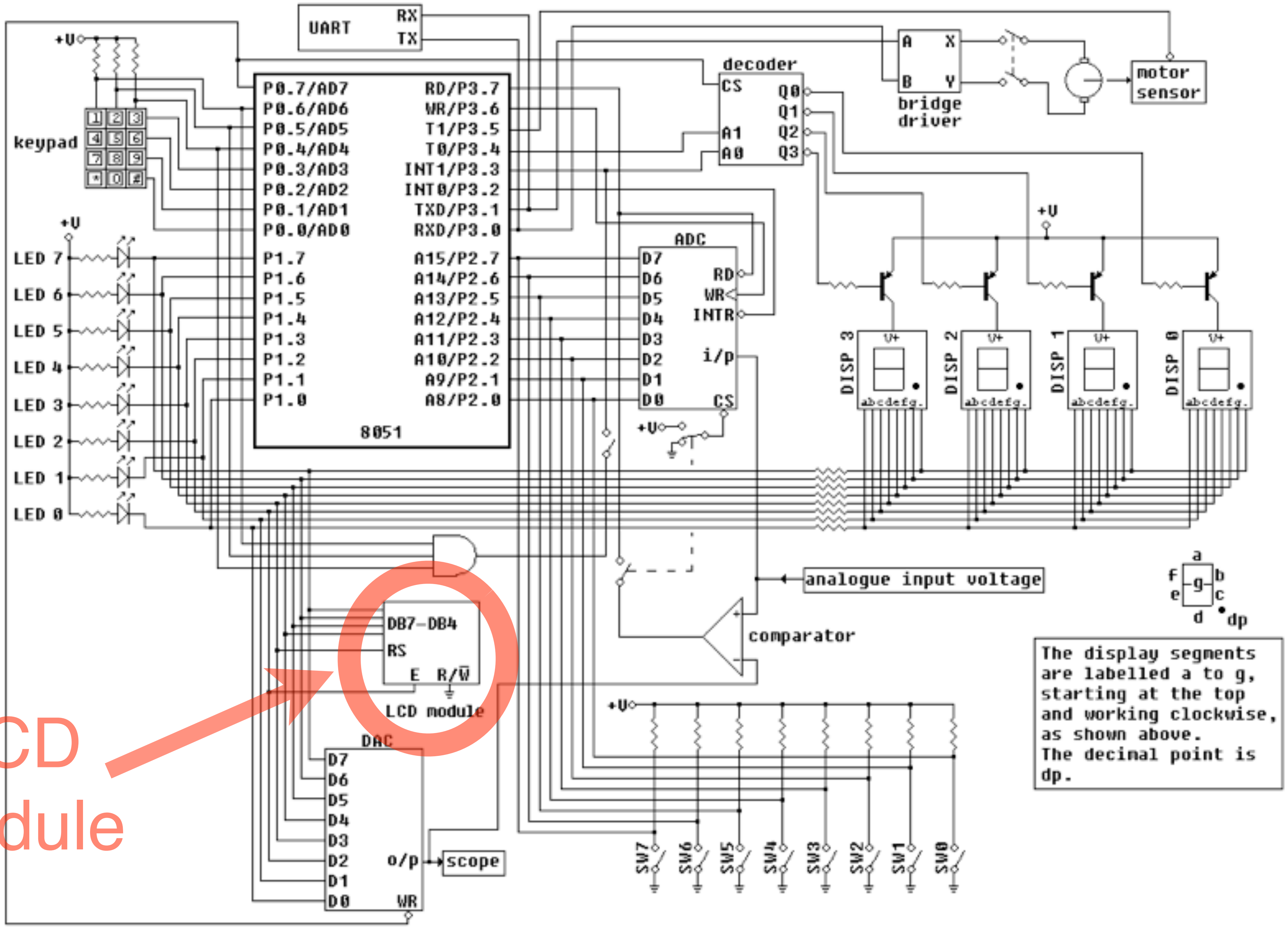
4-bit IR-Write



pulse E twice in one transaction

higher nibble, then lower nibble

EdSim51's schematic

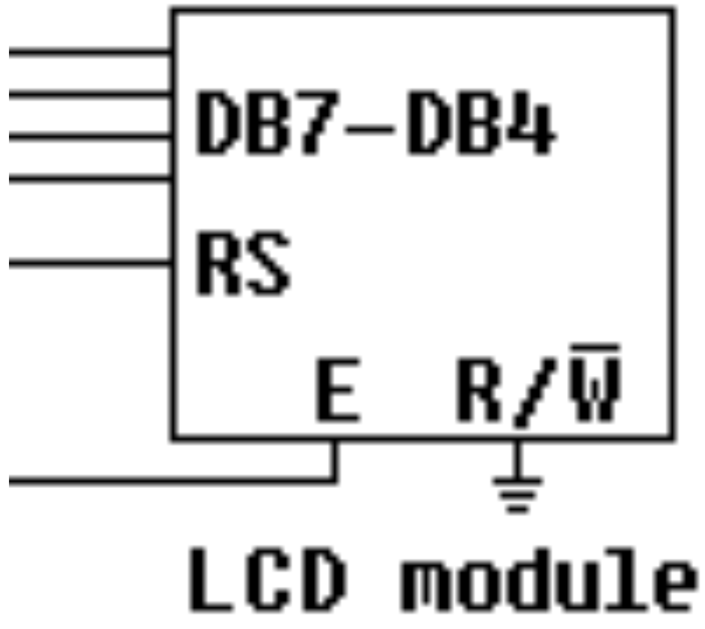


The display segments are labelled a to g, starting at the top and working clockwise, as shown above. The decimal point is dp.

LCD module

4-bit-mode connection to the LCD module

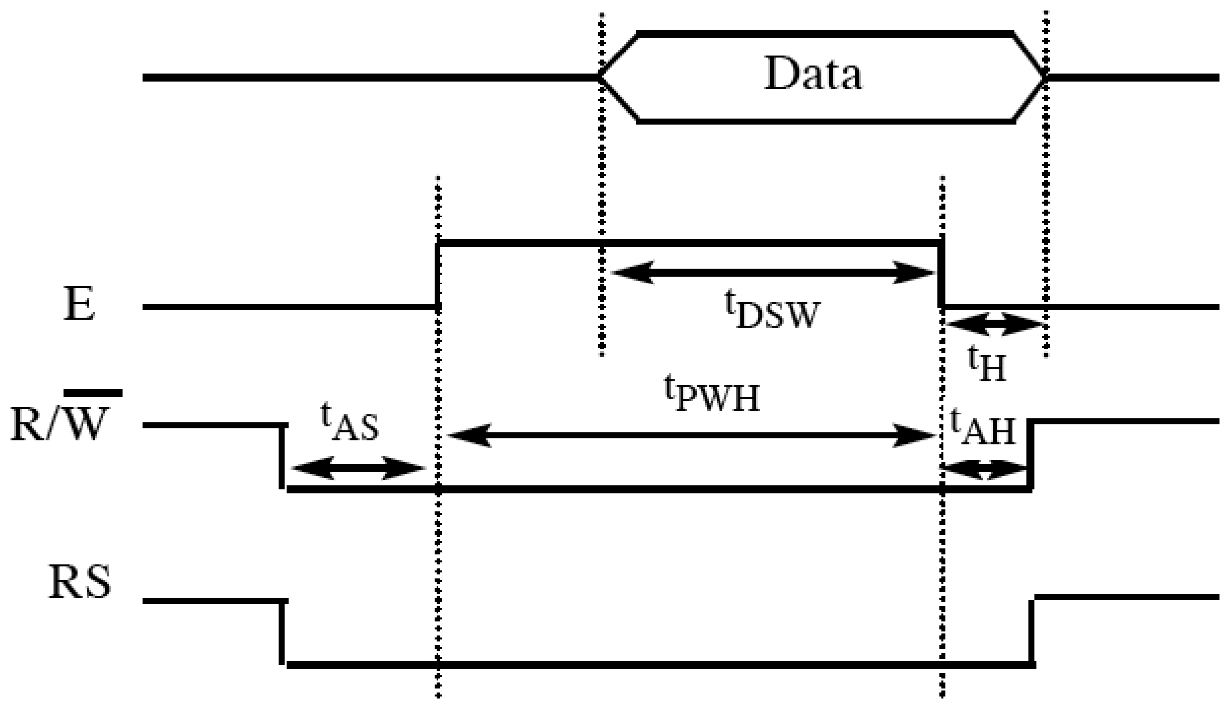
LCD	8-bit 8051	EdSim51
DB7 to DB4	P1.7 to P1.4	P1.7 to P1.4
DB3 to DB0	P1.3 to P1.0	no connect.
RS	P2.0	P1.3
E	P2.2	P1.2
$R/\overline{W}$	P2.1	GND



Implications of 4-bit connection of EdSim51

- 4-bit mode
  - Serialize as two pulses of E:
    one for high nibble, one for low nibble
- Write-only! ($R/\overline{W}$=0)
  - Can't read from LCD, but it's ok
  - Can't read Ready bit => just wait long enough

Instruction List (subset)

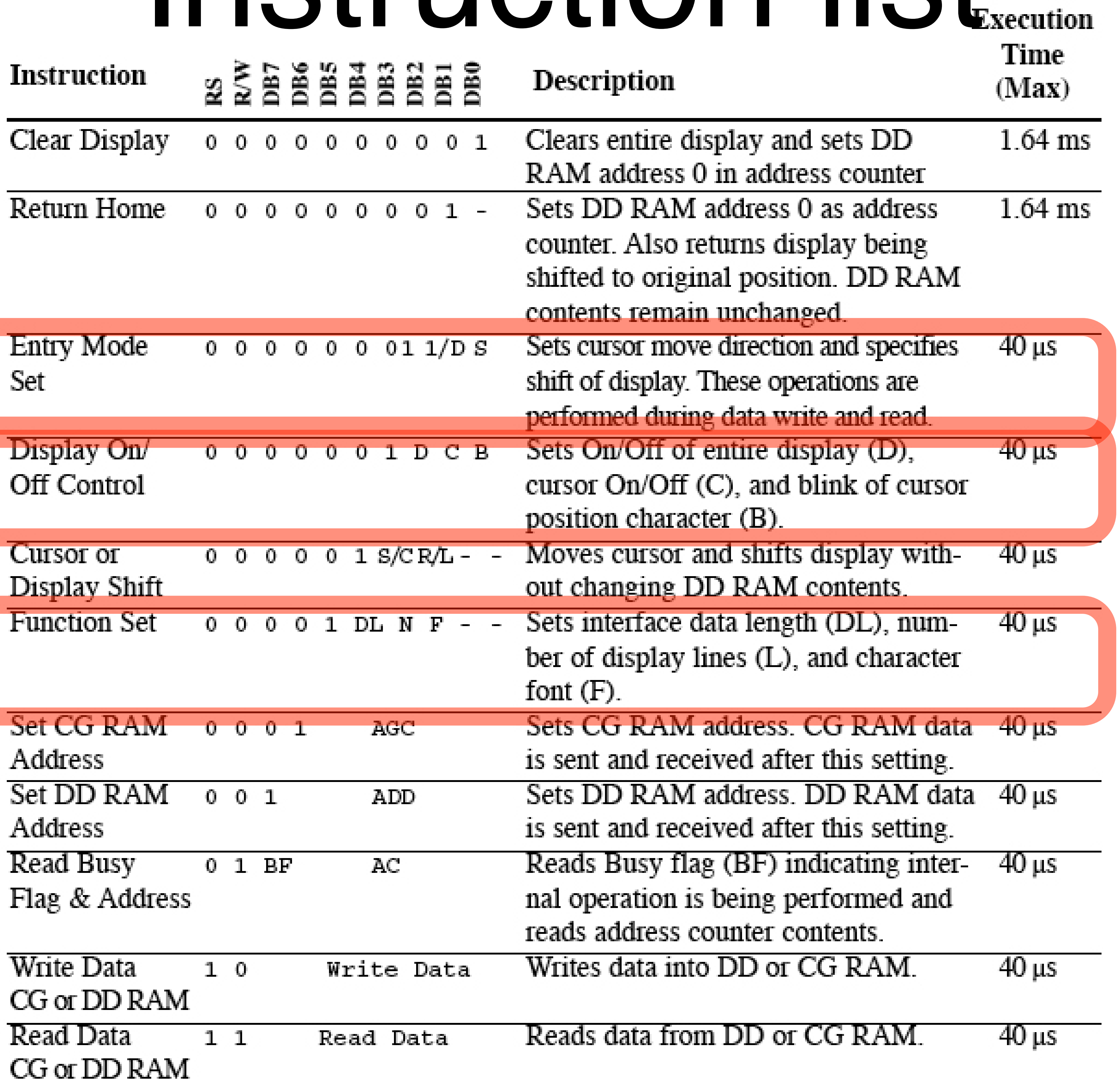


Executed as IR-Write transaction

(Hex)	Register
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to beginning of 1st line
C0	Force cursor to beginning of 2nd line
38	2 lines and 5x7 matrix

Instruction list

for init



Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (Max)
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DD RAM address 0 in address counter	1.64 ms
Return Home	0	0	0	0	0	0	0	0	1	-	Sets DD RAM address 0 as address counter. Also returns display being shifted to original position. DD RAM contents remain unchanged.	1.64 ms
Entry Mode Set	0	0	0	0	0	0	0	1	1/D	S	Sets cursor move direction and specifies shift of display. These operations are performed during data write and read.	40 µs
Display On/ Off Control	0	0	0	0	0	0	1	D	C	B	Sets On/Off of entire display (D), cursor On/Off (C), and blink of cursor position character (B).	40 µs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Moves cursor and shifts display without changing DD RAM contents.	40 µs
Function Set	0	0	0	0	1	DL	N	F	-	-	Sets interface data length (DL), number of display lines (L), and character font (F).	40 µs
Set CG RAM Address	0	0	0	1			AGC				Sets CG RAM address. CG RAM data is sent and received after this setting.	40 µs
Set DD RAM Address	0	0	1				ADD				Sets DD RAM address. DD RAM data is sent and received after this setting.	40 µs
Read Busy Flag & Address	0	1	BF				AC				Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.	40 µs
Write Data CG or DD RAM	1	0				Write Data					Writes data into DD or CG RAM.	40 µs
Read Data CG or DD RAM	1	1				Read Data					Reads data from DD or CG RAM.	40 µs

Technical details

- EdSim website
  - http://www.edsim51.com/simInstructions.html#lcdModule
  - http://www.edsim51.com/examples.html
- Data sheet
  - http://www.edsim51.com/8051simulator/HD44780.pdf

Sample code in C

- http://www.edsim51.com/examples/lcd.c
- Written for Keil C, convert to sdcc

#include <reg51.h>	#include <8051.h>
sbit DB7 = P1^7;	#define DB7 P1_7
void main(void) {	void Main(void) {

- Compile and link separately

```
sdcc -c lcd.c
sdcc --stack-loc 0x80 --data-loc 0x30 lcd.rel
mv lcd.ihx lcd.hex
```

- Sys.Clock=12MHz, Update Freq. = 500

What does lcd.c do?

- Initialize the display, write some initial text
- Main loop
  - P2.5: return cursor to home position
  - P2.6: shift display to the left
  - P2.7: shift display to the right
  - Assume P2.6 and P2.7 not simultaneous

API in LCD sample code

macros that call IRWrite

function	purpose
functionSet()	set interface length (4 or 8 bit), 1-2 lines, font size
entryModeSet(id, s)	auto increment/decrement, shift/don't shift display
displayOnOffControl (disp, curs, blink)	display on/off, cursor show/hide, blink/no blink
cursorOrDisplayShift (sc, lr)	sc: 0=cursor, 1=display. lr: 0=left, 1=right
setDdRamAddress (addr)	write register (RS=0), 7-bit address
sendChar(c)	write data register (RS=1)

Initialization, configuration

```
void Main(void) {
        functionSet();
        entryModeSet(1, 0);
            // increment and no shift
        displayOnOffControl(1, 1, 1);
             // display on, cursor on and blinking on
        sendString("EdSim51 LCD Module Simulation");
        setDdRamAddress(0x40);
             // set address to start of second line
        sendString("Based on Hitachi HD44780");
        while (1) {
```

void functionSet(void) with hardwired N, F

- Writes DB= 0010xxxx
  RS= 0
  Pulses it twice
  => configures it for 4-bit interface
- Writes 10xx for lower nibble of the code
  N=1: 2-line mode
  F=0: 5x7 font size

Hardwired in EdSim51

```
void functionSet(void) {
        DB7 = 0;
        DB6 = 0;
        DB5 = 1;
        DB4 = 0;
        RS = 0;
        E = 1;
        E = 0;
        delay();
        E = 1;
        E = 0;
        DB7 = 1; // N=1,F=0
        E = 1;
        E = 0;
        delay();
}
```

Configures for 4-bit

NFxx

Function Set

3.1.6 Function Set

Sets the interface data length, the number of lines, and character font.

	RS	R/W	DB_7	DB_6	. . .				DB_1	DB_0
Code	0	0	0	0	1	DL	N	F	x	x

Note: x = Don't Care

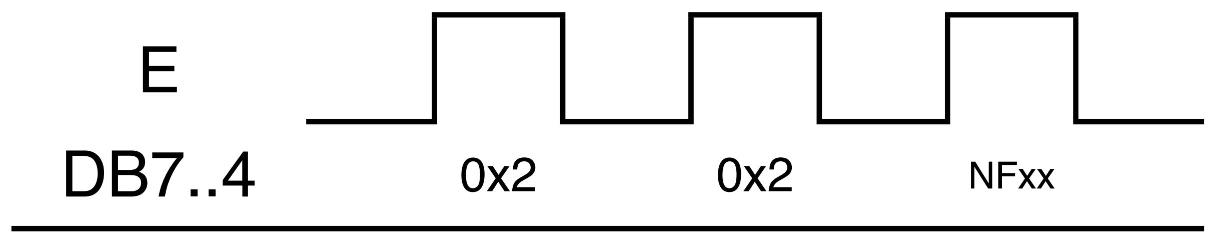
DL: Sets interface data length. Data is sent or received in 8-bit length (DB_7 ~ DB_0) when DL = "1", and in 4-bit length (DB_7 ~ DB_4) when DL = 0. When the 4-bit length is selected, data must be sent or received twice.

N: Sets the number of lines

N = "0"	:	1 line display (1/8 duty)
N = "1"	:	2 line display (1/16 duty)

F: Sets character font.

F = "1"	:	5 x 10 dots
F = "0"	:	5 x 7 dots



Note: Perform the function at the head of the program before executing all instructions (except Busy flag/address read). From this point, the function set instruction cannot be executed other than to change interface length.

void entryModeSet (__bit id, __bit s)

- Writes DB=0000 for higher nibble
  RS= 0 (instruction reg)
- Lower nibble
  DB3 = 0
  DB2 = 1
  DB1 = id (auto inc/dec)
  DB0 = s (shift display or not)

Think variable-length command code (6-bit command + 2-bit argument)

```
void entryModeSet(__bit id, __bit s) {
        RS = 0;
        DB7 = 0;
        DB6 = 0;
        DB5 = 0;
        DB4 = 0;

        E = 1;
        E = 0;

        DB6 = 1;
        DB5 = id;
        DB4 = s;

        E = 1;
        E = 0;
        delay();
}
```

Entry-mode Set

3.1.3 Entry mode set

	RS	S/W	DB_7	DB_6	. . .				DB_1	DB_0
Code	0	0	0	0	0	0	0	1	I/D	S

Sets the Increment/Decrement and Shift modes to the desired settings.

I/D: Increments (I/D = 1) or decrements (ID = 0) the DD RAM address by 1 when a character code is written into or read from the DD RAM.

The cursor or blink moves to the right when incremented by +1.

The same applies to writing and reading the CG RAM.

S: Shifts the entire display either to the right or to the left when S = 1: shift to the left when I/D = 1 and to the right when I/D = 0. Thus it looks as if the cursor stands still and only the display seems to move.

The display does not shift when reading from DD RAM nor when S = 0.

displayOnOffControl (display, cursor, blink)

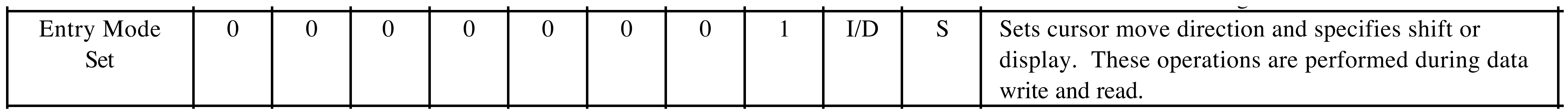
- Writes DB=0000 for higher nibble RS= 0 (instruction reg)
- Lower nibble
  DB3 = 1
  DB2 = display
  DB1 = cursor
  DB0 = blinking

Think variable-length command code (5-bit command + 3-bit argument)

```
void displayOnOffControl(__bit
display, __bit cursor, __bit blinking) {
        // implicit RS = 0;
        DB7 = 0;
        DB6 = 0;
        DB5 = 0;
        DB4 = 0;

        E = 1;
        E = 0;

        DB7 = 1;
        DB6 = display;
        DB5 = cursor;
        DB4 = blinking;

        E = 1;
        E = 0;
        delay();
}
```

Address counter

- Address counter (AC) is auto-incremented on data-write or data-read
  - I/D=1: increment; =0: decrement
  - S=1: display shift



Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies shift or display. These operations are performed during data write and read.

- The same AC for both DD and CG RAM!
  - Depending on R/S value

setDdRamAddress (char address)

- Writes DB= 0x80+address
  RS= 0 (instruction reg)
- Think variable-length command code (1-bit command + 7-bit address)
- getBit() extracts a bit from a byte

```
void setDdRamAddress(char
address) {
        RS = 0;
        DB7 = 1;
        DB6 = getBit(address, 6);
        DB5 = getBit(address, 5);
        DB4 = getBit(address, 4);
        E = 1;
        E = 0;
        DB7 = getBit(address, 3);
        DB6 = getBit(address, 2);
        DB5 = getBit(address, 1);
        DB4 = getBit(address, 0);
        E = 1;
        E = 0;
        delay();
}
```

sendChar(char c)

- Writes DB= c
  RS= 1 (data register)
- Data write: all 8 bits of c
- RS=1 selects the data

```
void sendChar(void c) {
        DB7 = getBit(c, 7);
        DB6 = getBit(c, 6);
        DB5 = getBit(c, 5);
        DB4 = getBit(c, 4);
        RS = 1;
        E = 1;
        E = 0;
        DB7 = getBit(c, 3);
        DB6 = getBit(c, 2);
        DB5 = getBit(c, 1);
        DB4 = getBit(c, 0);
        E = 1;
        E = 0;
        delay();
}
```

sendString(char* str)

- null-terminated strings
- Issues
  - index may be slower than pointer
  - int is 2 bytes on 8051
- What does the assembly code look like?

```
void sendString(char* str) {
        int index = 0;
        while (str[index] != 0) {
                sendChar(str[index]);
                index++;
        }
}
```

Compare to

```
void sendString(char* str) {
        char *p;
        for (p=str; *p; p++) {
                sendChar(*p);
        }
}
```

main loop in Main() continued

```
while (1) {
    if (ret == 0) {  // button #5
        returnHome();
    } else {
        if (left == 0 && right == 1) { // button #7
            cursorOrDisplayShift(1, 0);
        } else if (left == 1 && right == 0) { // #6
            cursorOrDisplayShift(1, 1);
        }
    }
}
```

void returnHome()

conceptually, the code is doing

- Higher nibble
  DB<7:4>= $\boxed{0000}$
  RS= 0 (instruction reg)
- Lower nibble (DB<3:0> in 8-bit)
  DB<7:4> = $\boxed{001x}$
  where x defaults to 0

Think variable-length command code (7-bit command with 1-bit don't care + no argument)

```
void returnHome(void) {
        RS = 0;
        DB7 = 0;
        DB6 = 0;
        DB5 = 0;
        DB4 = 0;
        E = 1;
        E = 0;
        DB5 = 1;
        E = 1;
        E = 0;
        delay();
}
```

Example Instructions

3.1.1 Clear Display

	RS	R/W	DB_7	DB_6	. . .				DB_1	DB_0
Code	0	0	0	0	0	0	0	0	0	1

Writes the space code "20" (hexadecimal) into all addresses of DD RAM. Returns display to its original position if it was shifted. In other words the display clears and the cursor or blink moves to the upper left edge of the display. The execution of clear display instruction sets entry mode to increment mode.

3.1.2 Return Home

	RS	R/W	DB_7	DB_6	. . .				DB_1	DB_0
Code	0	0	0	0	0	0	0	0	1	x

Note: x = Don't Care

Sets the DD RAM address "0" in address counter. Return display to its original position if it was shifted. DD RAM contents do not change.

The cursor or the blink moves to the upper left edge of the display. Text on the display remains unchanged.

cursorOrDisplayShift (bit sc, bit rl)

Conceptually,

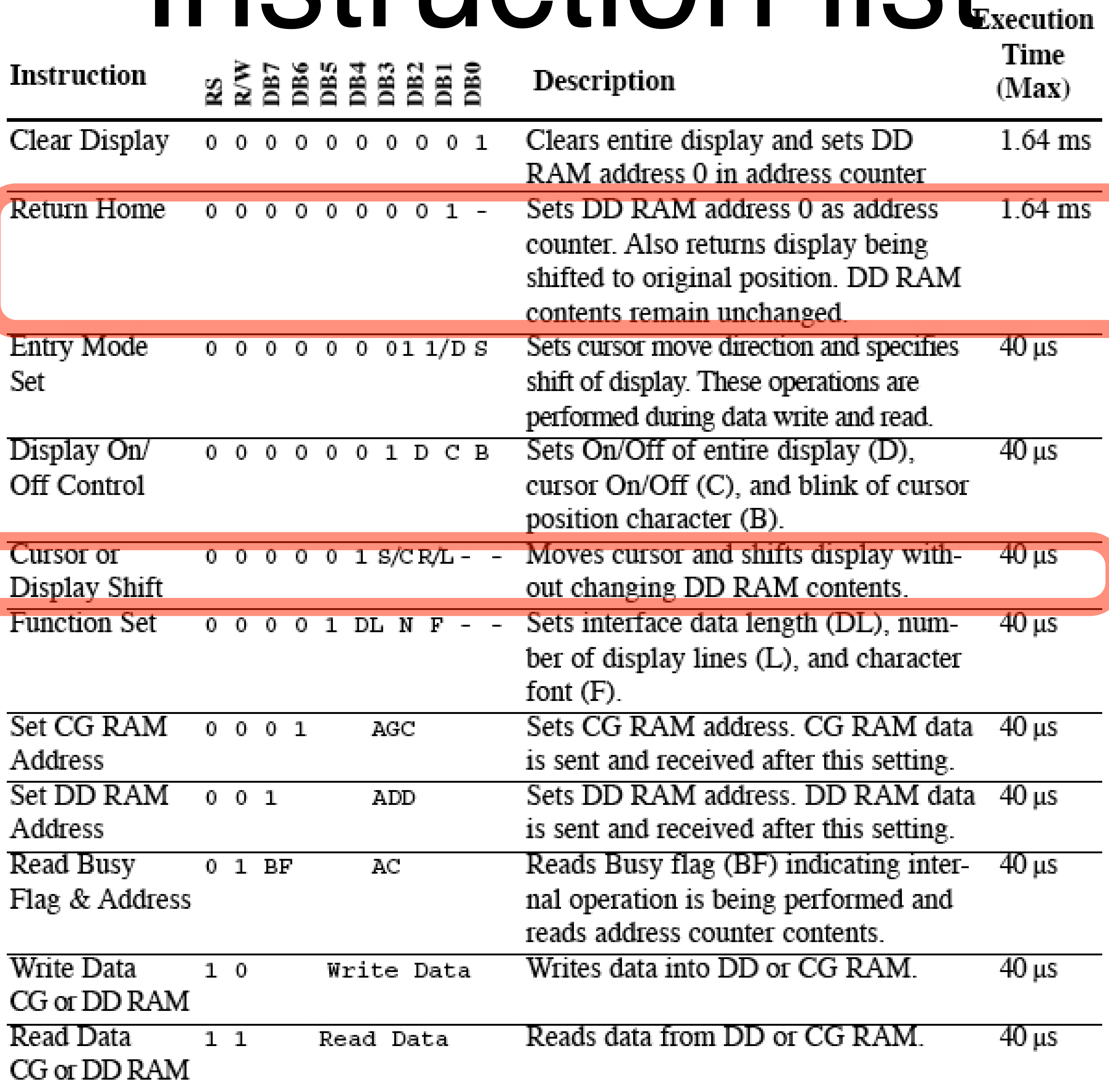
- Higher nibble
  DB<7:4>= $\boxed{0001}$
  RS= 0 (instruction reg)

- Lower nibble (DB<3:0> in 8-bit)
  DB<7> = sc
  DB<6> = rl
  DB<5:4> = don't care

Think variable-length command code (4-bit command with two 1-bit arg + two don't cares)

```
void cursorOrDisplayShift(__bit
sc, __bit rl) {
        RS = 0;
        DB7 = 0;
        DB6 = 0;
        DB5 = 0;
        DB4 = 1;

        E = 1;
        E = 0;

        DB7 = sc;
        DB6 = rl;

        E = 1;
        E = 0;
        delay();
}
```

Instruction list



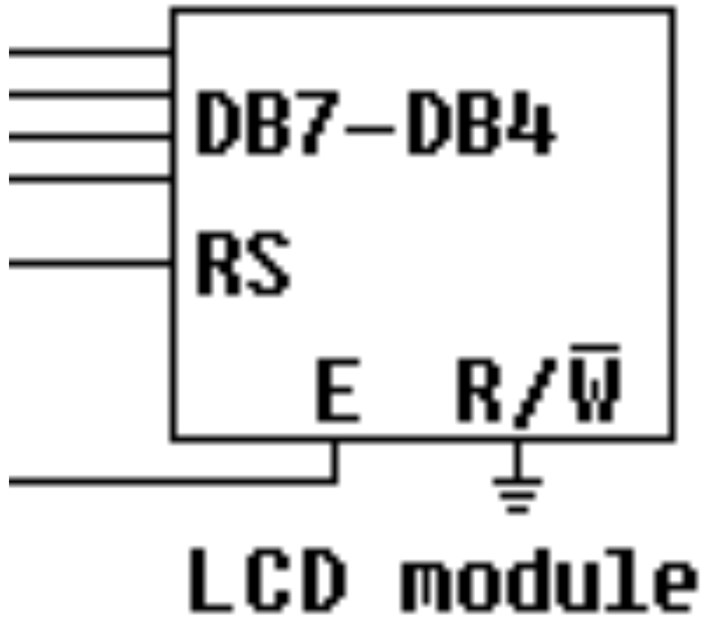
Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (Max)
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DD RAM address 0 in address counter	1.64 ms
Return Home	0	0	0	0	0	0	0	0	1	-	Sets DD RAM address 0 as address counter. Also returns display being shifted to original position. DD RAM contents remain unchanged.	1.64 ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies shift of display. These operations are performed during data write and read.	40 µs
Display On/ Off Control	0	0	0	0	0	0	1	D	C	B	Sets On/Off of entire display (D), cursor On/Off (C), and blink of cursor position character (B).	40 µs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Moves cursor and shifts display without changing DD RAM contents.	40 µs
Function Set	0	0	0	0	1	DL	N	F	-	-	Sets interface data length (DL), number of display lines (L), and character font (F).	40 µs
Set CG RAM Address	0	0	0	1	AGC						Sets CG RAM address. CG RAM data is sent and received after this setting.	40 µs
Set DD RAM Address	0	0	1	ADD							Sets DD RAM address. DD RAM data is sent and received after this setting.	40 µs
Read Busy Flag & Address	0	1	BF	AC							Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.	40 µs
Write Data CG or DD RAM	1	0	Write Data								Writes data into DD or CG RAM.	40 µs
Read Data CG or DD RAM	1	1	Read Data								Reads data from DD or CG RAM.	40 µs

Inefficient code in lcd.c

- Bit access
  - getBit(), setting DB7..DB4 individually
  - Should be able to do byte-writing
- No reuse of similar code
  - Most are IRWrite( ) transactions
  - Even IRWrite and DRWrite are similar

EdSim's 4-bit DB interface revisited

LCD	EdSim51
DB7 to DB4	P1.7 to P1.4
DB3 to DB0	no connect.
RS	P1.3
E	P1.2
$R/\overline{W}$	GND



Example: returnHome

Original
#define DB P1

```
void returnHome(void) {
        RS = 0;
        DB7 = 0;
        DB6 = 0;
        DB5 = 0;
        DB4 = 0;
        E = 1;
        E = 0;
        DB5 = 1;
        E = 1;
        E = 0;
        delay();
}
```

new, assume
P1.3=RS=0, P1.2=E=0,
P1.1, P1.0 unchanged

```
void returnHome(void) {
        P1 &= 0x03;  // 00000011
        // P1 = P1 & 0x03;
        E = 1;
        E = 0;
        P1 = 0x20;
        E = 1;
        E = 0;
        delay();
}
```

Example: how to include bit parameters

Original

```
void entryModeSet(__bit id, __bit s) {
        RS = 0;
        DB7 = 0;
        DB6 = 0;
        DB5 = 0;
        DB4 = 0;
        E = 1;
        E = 0;
        DB6 = 1;
        DB5 = id;
        DB4 = s;
        E = 1;
        E = 0;
        delay();
}
```

new

```
void entryModeSet(__bit id, __bit s) {
        P1 &= 0x03;
        // clear out all except P1.0, P1.1
        E = 1;
        E = 0;
        P1 = 0x20 | (id << 5) | (s << 4)
              | (P1 & 0x3);
        E = 1;
        E = 0;
        delay();
}
```

Most general: IRWrite() function (4-bit)

- Common routine shared by all these commands!
- can be used by routines
  returnHome()
  clearScreen()
  entryModeSet(id, s)
  displayOnOffControl(...)
  cursorOrDisplayShift(...)
  setDdRamAddress(...)
  and more...

```
void IRWrite(char c) {
        P1 = (c & 0xF0);
        // write all of higher nibble of c,
        // keep RS, E, and others low

        E = 1;
        E = 0;

        P1 = (c << 4);
        // write lower nibble of c,
        // keep RS, E, and others low

        E = 1;
        E = 0;
        delay();
}
```

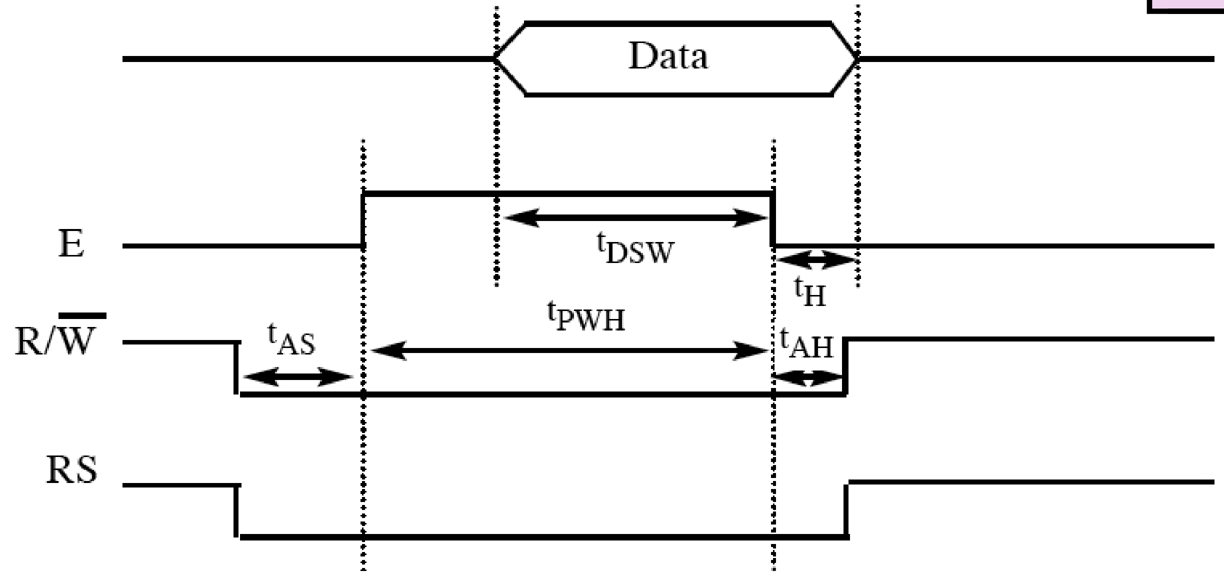
Instruction List (subset)

```
#define clearScreen() \
    IRWrite(1)
```

```
#define returnHome() \
    IRWrite(2)
```

(Hex)	Register
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on

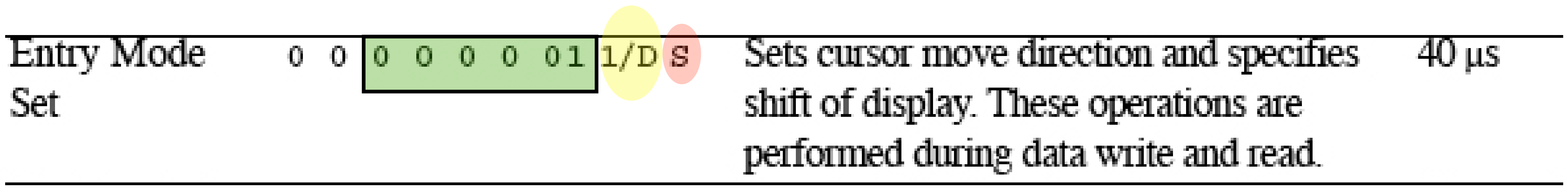
these are shown with hardwired parameters



All of these commands are really IR-Write transaction

Commands w/ args

- Example: entryModeSet



Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies shift of display. These operations are performed during data write and read.	40 µs

```
#define entryModeSet(id, s) \
  IRWrite(0x4 | (id << 1) | s)
```

Macros to IRWrite()

- Body of IRWrite( ) may be 4-bit or 8-bit
  we can use a macro to decide which version to select
- The rest of the macros can be unchanged!
- #define entryModeSet(id, s) can still be defined as IRWrite(0x4 | (id << 1) | s) regardless of 4 or 8 bit

```
#define LCD_4_bit
...
void IRWrite(char c) {
#ifdef LCD_4_bit
    ... code for 4 bit
#else
    ... code for 8 bit
#endif
```

Example Complete Sequence

No.	Instruction	Display	Operation
1	Power supply ON (Initialized by Internal reset circuit) RS | R/W | DB_7 ~ DB_0		Module is initialized.
2	Function set RS | R/W | DB_7 ~ DB_0 0 | 0 | 0 0 1 1 1 0 * *		Sets the interface data length to 8 bits and selects 2-line display and 5 x 7-dot character font.
3	Display ON/OFF Control RS | R/W | DB_7 ~ DB_0 0 | 0 | 0 0 0 0 1 1 1 0	_	Turns on display and cursor.
4	Entry mode set RS | R/W | DB_7 ~ DB_0 0 | 0 | 0 0 0 0 0 1 1 0	_	Sets mode to increment address by one and to shift the cursor to the right at the time of write to internal RAM

Example Complete Sequence

		RS	R/W	DB_7 ~ DB_0		
5	Write data to CG/DD RAM	1	0	0 1 0 0 1 1 0 0	L_	Writes “L”. Cursor is incremented by one and shifts to the right.
6	Write data to CG/DD RAM	1	0	0 1 0 0 0 0 1 1	LC_	Writes “C”
7	⋮				⋮	
8	Write data to CG/DD RAM	1	0	0 0 1 1 0 1 1 0	LCD MODULE DMC16	Writes “6”

Example Complete Sequence

No.	Instruction	Display	Operation
9	Set DD RAM address. RS | R/W | DB_7 ~ DB_0 0 | 0 | 1 1 0 0 0 0 0 0	LCD MODULE DMC16 _	Sets RAM address so that the cursor is positioned at the head of the 2nd line.
10	Write data to CG/DD RAM RS | R/W | DB_7 ~ DB_0 1 | 0 | 0 0 1 1 0 0 1 1	LCD MODULE DMC16 1_	Write “1”
11	Write data to CG/DD RAM RS | R/W | DB_7 ~ DB_0 1 | 0 | 0 0 1 1 0 0 1 0	LCD MODULE DMC16 16_	Writes “6”
12	⋮	⋮	

Example Complete Sequence

		RS	R/W	DB$_7$ ~ DB$_0$		
13	Write data to CG/DD RAM address	1	0	0 0 1 0 1 0 1 0	LCD MODULE DMC16 16 DIGITS, 2 LINES_	Writes "S"
14	Set DD/RAM address	0	0	1 0 0 0 0 0 0 0	L̲CD MODULE DMC16 16 DIGITS, 2 LINES	Moves cursor to original position
15	Clear display	0	0	0 0 0 0 0 0 0 1	_	Return both display and cursor to the original position
16						

3.1.4 Display ON/OFF Control

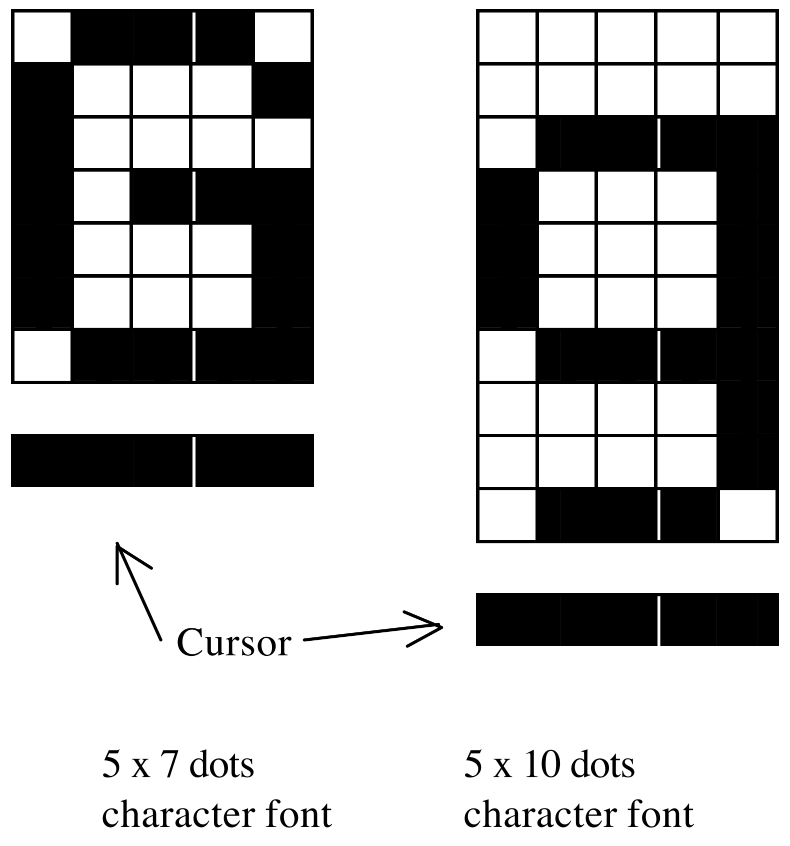
	RS	R/W	DB_7	DB_6	. . .				DB_1	DB_0
Code	0	0	0	0	0	0	1	D	C	B

Controls the display ON/OFF status, Cursor ON/OFF and Cursor Blink function.

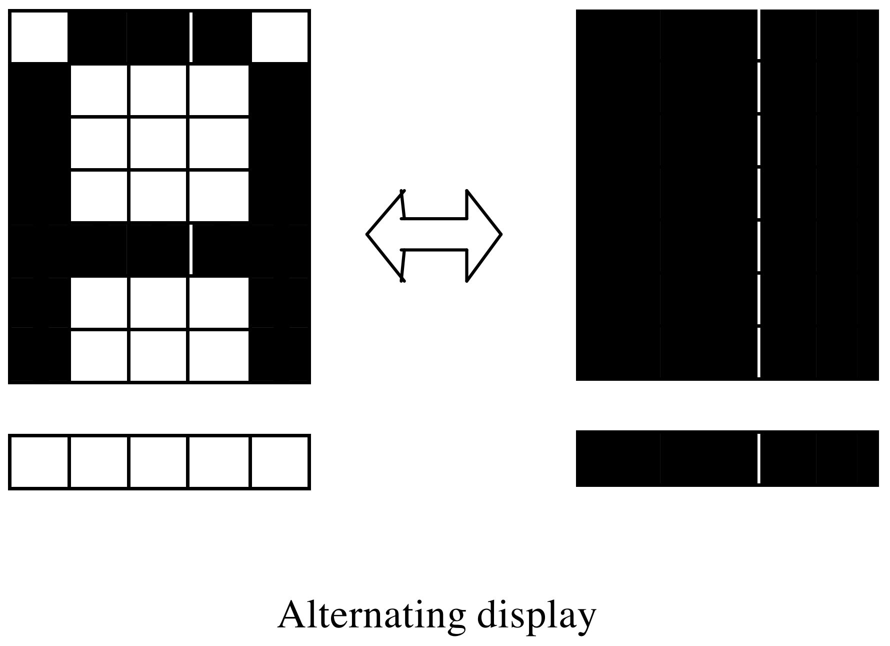
D: The display is ON when D = 1 and OFF when D = 0. When OFF due to D = 0, display data remains in the DD RAM. It can be displayed immediately by setting D = 1.

C: The cursor displays when C = 1 and does not display when C = 0. The cursor is displayed on the 8th line when 5 x 7 dot character font has been selected.

B: The character indicated by the cursor blinks when B = 1. The blink is displayed by switching between all blank dots and display characters at 0.4 sec intervals.
The cursor and the blink can be set to display simultaneously.



(a) Cursor display example
C = 1 ; B = 0



(b) Blink display example
C = 1 ; B = 1

Busy flag revisited

3.1.9 Read Busy Flag and Address

	RS	R/W	DB_7	DB_6		. . .			DB_1	DB_0
Code	0	1	BF	A	A	A	A	A	A	A

$\leftarrow$ Higher order bits Lower order bits $\rightarrow$

Reads the busy flag (BF) and value of the address counter (AC). BF = 1 indicates that on internal operation is in progress and the next instruction will not be accepted until BF is set to “0”. If the display is written while BF = 1, abnormal operation will occur.

The BF status should be checked before each write operation.

At the same time the value of the address counter expressed in binary AAAAAAA is read out. The address counter is used by both CG and DD RAM and its value is determined by the previous instruction. Address contents are the same as in sections 3.1.7 and 3.1.8.

Checking LCD's Busy Flag

- Use a transaction with RS=0, $R/\overline{W}$=1
  - Only one type of IR-read!
- Result: DB7 (data pin 7) is the busy flag
  - 1: busy
  - 0: ready to accept new command
- if not checking Busy Flag,
  must delay long enough (done in EdSim51)

Programming the LCD with EdSim51

- Works with 2-line, 5x8 font (hardwired in functionSet())
- Does not work with Reading
    - can't check busy flag
    - can't read from CG RAM or DD RAM

Issue: fine timing (ns)

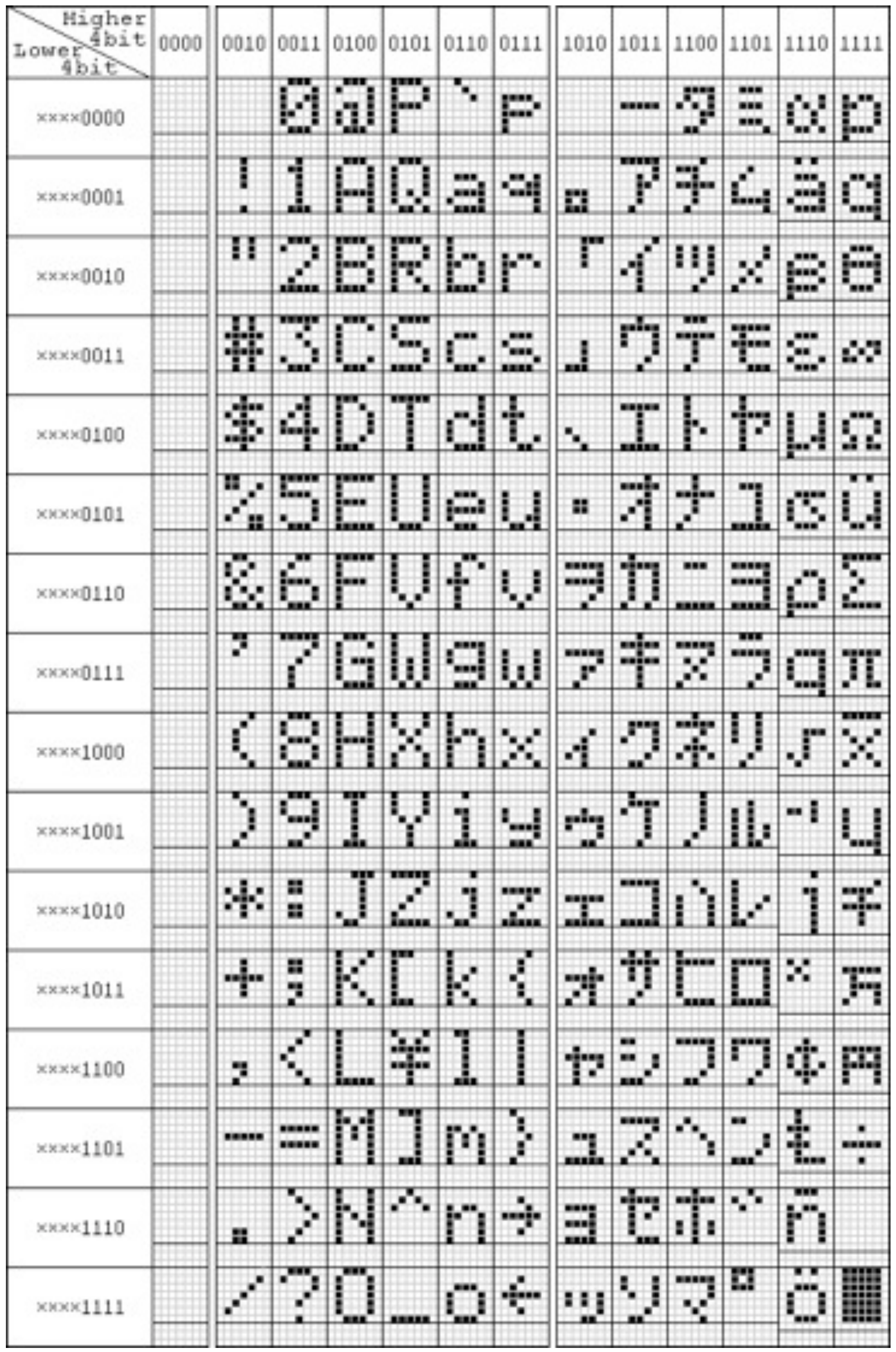
Item	Symbol	Min	Typ	Max
Enable cycle time	t_{cycE}	1000	-	-
Enable pulse width (high level)	PW_{EH}	450	-	-
Enable rise/fall time	t_{Er}, t_{Ef}	-	-	25
Address set-up time	t_{AS}	60	-	-
Address hold time	t_{AH}	20	-	-
Data set-up time	t_{DSW}	195	-	-
Data delay time	t_{DDR}	-	-	360
Data hold time (W)	t_H	10	-	-
Data hold time (R)	t_{DHR}	5	-	-

Instruction timing

- Traditional 8051
  - 12 oscillator cycles = 1 instruction cycle
  - i.e., osc at 12MHz => instr. cyc = 1μs
- Enable pulse width: 450ns min
  - SETB E, CLR E => 1000ns, OK
  - but instr. cyc. time might be < 1μs!!
    => may need delay to meet min PW_{EH}

Char Gen. RAM, ROM

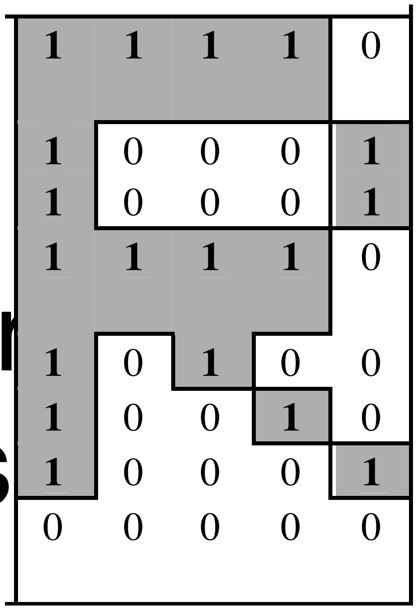
- Stores bitmap of font
- ROM: built-in
  - not user-addressed
- RAM: user defined
  - 64 bytes total
    (6-bit address)



Higher 4bit / Lower 4bit	0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
xxxx0000			0	@	P	`	p		ー	タ	ミ	α	p
xxxx0001		!	1	A	Q	a	q	。	ア	チ	ム	ä	q
xxxx0010		"	2	B	R	b	r	「	イ	ツ	メ	β	θ
xxxx0011		#	3	C	S	c	s	」	ウ	テ	モ	ε	∞
xxxx0100		$	4	D	T	d	t	、	エ	ト	ヤ	μ	Ω
xxxx0101		%	5	E	U	e	u	・	オ	ナ	ユ	σ	ü
xxxx0110		&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111		'	7	G	W	g	w	ァ	キ	ヌ	ラ	g	π
xxxx1000		(	8	H	X	h	x	ィ	ク	ネ	リ	√	x̄
xxxx1001		)	9	I	Y	i	y	ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010		*	:	J	Z	j	z	ェ	コ	ハ	レ	j	千
xxxx1011		+	;	K	[	k	{	ォ	サ	ヒ	ロ	ˣ	万
xxxx1100		,	<	L	¥	l	|	ャ	シ	フ	ワ	¢	円
xxxx1101		-	=	M	]	m	}	ュ	ス	ヘ	ン	£	÷
xxxx1110		.	>	N	^	n	→	ョ	セ	ホ	゛	ñ	
xxxx1111		/	?	O	_	o	←	ッ	ソ	マ	゜	ö	█

Character Generation RAM (CG RAM)

- 64 bytes RAM
    - Either eight 5x7 or four 5x10 bitmaps
- User defined chars have code 00H..07H
    - actually, 08H..0FH also select the same



1	1	1	1	0
1	0	0	0	1
1	0	0	0	1
1	1	1	1	0
1	0	1	0	0
1	0	0	1	0
1	0	0	0	1
0	0	0	0	0

char code	CG RAM address range	
0	0	7
1	8	0F
2	10	17
3	18	1F
4	20	27
5	28	2F
6	30	37
7	38	3F

Example: setting CG RAM for Euro symbol

```
const char EuroSymbol [ ] = {
            0b00000110,   //   **
            0b00001001,   //  *  *
            0b00011110,   //  ****
            0b00001000,   //  *
            0b00011110,   // ****
            0b00001001,   //  *  *
            0b00000110,   //   **
            0                // I end of data (in this example, 0
    };                     can be used to indicate end of data)
```

€

To write the symbol to display

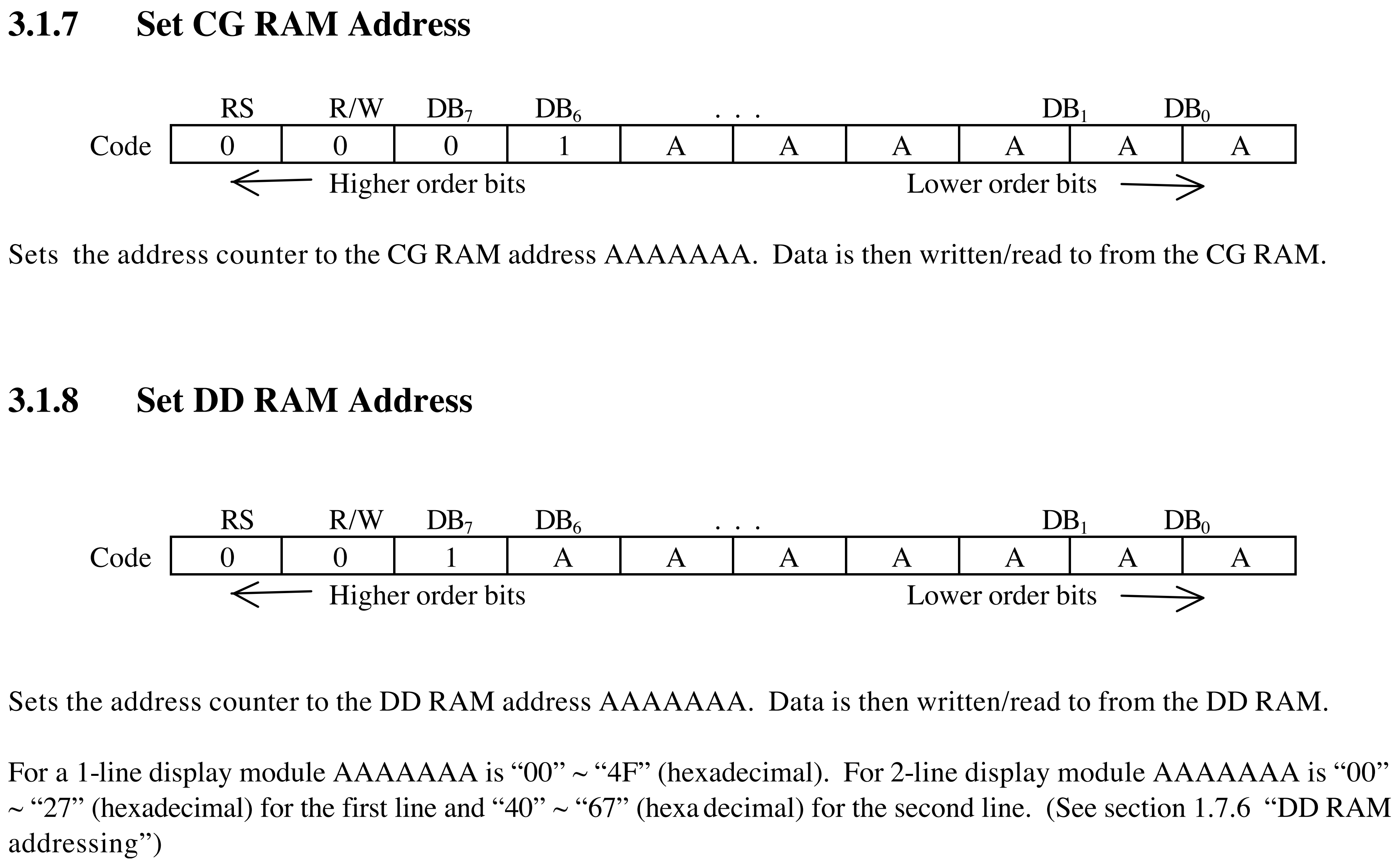
- To write the bitmap to CG memory
  - setCgRamAddress()// 0, 0x8, 0x10, ...
  - sendString(EuroSymbol) // write bitmap
- To write the character code to DD memory
  - setDdRamAddress()// 0,1..0x40, 0x41, ..
  - 8 possible custom-defined characters

Ex. writing to CG RAM and displaying custom char

- setCgRamAddress(0x8);
   // starting address for character 0x1.
  sendString(EuroSymbol);
  // send the bitmap data
  setDdRamAddress(0x0);
  // set display position to 1st row left col
  sendString("\x01"); // char code 0x1.

Setting address counter

- To set DD address 00..0F => DB=80..8F
  To set DD address 40..4F => DB=C0..CF
  To set CG address 00..3F => DB= 40..7F



3.1.7 Set CG RAM Address

	RS	R/W	DB_7	DB_6	. . .				DB_1	DB_0
Code	0	0	0	1	A	A	A	A	A	A

← Higher order bits Lower order bits →

Sets the address counter to the CG RAM address AAAAAAA. Data is then written/read to from the CG RAM.

3.1.8 Set DD RAM Address

	RS	R/W	DB_7	DB_6	. . .				DB_1	DB_0
Code	0	0	1	A	A	A	A	A	A	A

← Higher order bits Lower order bits →

Sets the address counter to the DD RAM address AAAAAAA. Data is then written/read to from the DD RAM.

For a 1-line display module AAAAAAA is “00” ~ “4F” (hexadecimal). For 2-line display module AAAAAAA is “00” ~ “27” (hexadecimal) for the first line and “40” ~ “67” (hexa decimal) for the second line. (See section 1.7.6 “DD RAM addressing”)

3.1.10 Write Data to CG or DD RAM

	RS	R/W	DB_7	DB_6	. . .				DB_1	DB_0
Code	1	0	D	D	D	D	D	D	D	D

$\leftarrow$ Higher order bits Lower order bits $\rightarrow$

Writes binary 8-bit data DDDDDDDD to the CG or DD RAM.

The previous designation determines whether the CG or DD RAM is to be written (CG RAM address set or DD RAM address set). After a write the entry mode will automatically increase or decrease the address by 1. Display shift will also follow the entry mode.

3.1.11 Read Data from CG or DD RAM

	RS	R/W	DB_7	DB_6	. . .				DB_1	DB_0
Code	1	1	D	D	D	D	D	D	D	D

$\leftarrow$ Higher order bits Lower order bits $\rightarrow$

Reads binary 8-bit data DDDDDDDD from the CG RAM or DD RAM.

The previous designation determines whether the CG or DD RAM is to be read.

Before entering the read instruction, you must execute either the CG RAM or DD RAM address set instruction.

If you don’t, the first read data will be invalidated. When serially executing the “read” instruction the next address data is normally read from the second read.