

Recap

Local variables: scope and allocation
How these concepts apply to C
Binding, allocation, access, deallocation

Overview

I/O synchronization
LCD interface
Implementing local variables with a stack frame
Parameter passing

Blind Cycle Counting Synchronization

Blind cycle counting is appropriate when the I/O delay is fixed and known. This type of synchronization is blind because it provides no feedback from the I/O back to the computer.

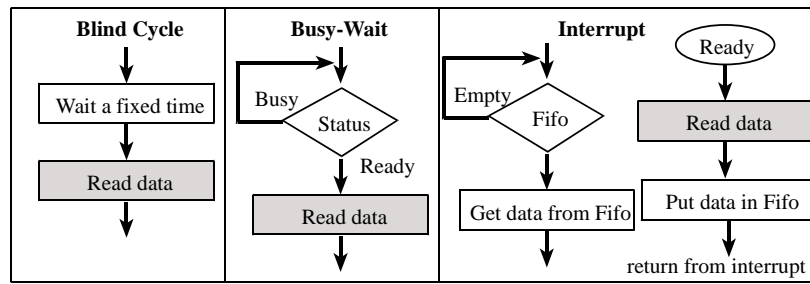
Busy Waiting Synchronization

Check busy/ready flag over and over until it is ready

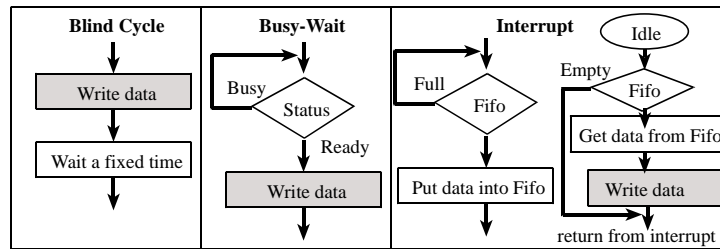
Interrupt Synchronization

Request interrupt when busy/ready flag is ready

Synchronizing with an input device



Synchronizing with an output device



8.5. Parallel Port LCD interface with the HD44780 controller

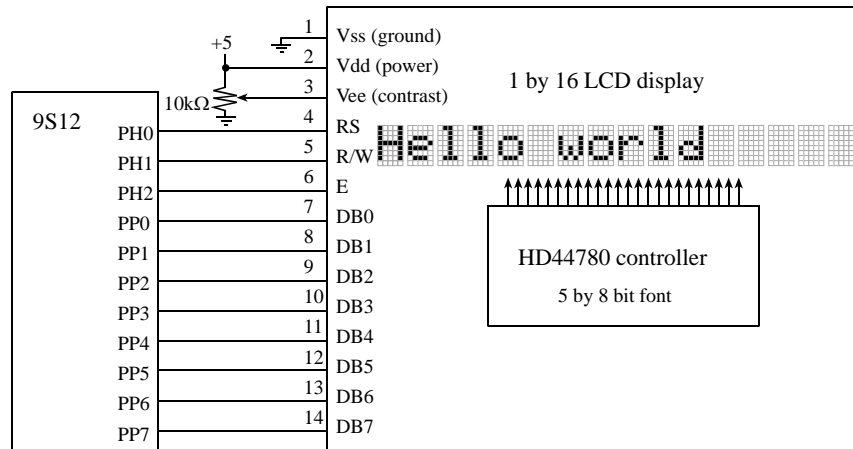


Figure 8.13. Interface of a HD44780 LCD controller.

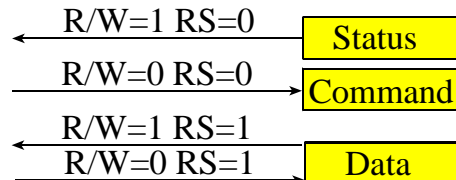
Show [LCDOptrex.pdf](#) datasheet

Show interface in PCB Artist

There are four types of access cycles to the HD44780 depending on RS and R/W

RS	R/W	Cycle
0	0	Write to Instruction Register
0	1	Read Busy Flag (bit 7)
1	0	Write data from μ P to the HD44780
1	1	Read data from HD44780 to the μ P

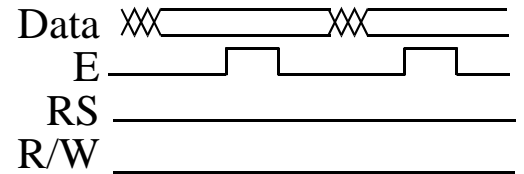
Table 8.12. Two control signals specify the type of access to the HD44780.



Execute the initialization routine using blind-cycle

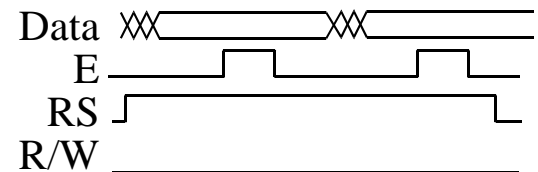
4-bit protocol write command (outCsr)

- 1) E=0, RS=0
- 2) 4-bit DB7,DB6,DB5,DB4 = most sign nibble of **command**
- 3) E=1
- 4) E=0 (latch 4-bits into LCD)
- 5) 4-bit DB7,DB6,DB5,DB4 = least sign nibble of **command**
- 6) E=1
- 7) E=0 (latch 4-bits into LCD)
- 8) blind cycle 90 us wait



4-bit protocol write ASCII data (LCD_OutChar)

- 1) E=0, RS=1
- 2) 4-bit DB7,DB6,DB5,DB4 = most significant nibble of data
- 3) E=1
- 4) E=0 (latch 4-bits into LCD)
- 5) 4-bit DB7,DB6,DB5,DB4 = least significant nibble of data
- 6) E=1
- 7) E=0 (latch 4-bits into LCD)
- 8) blind cycle 90 us wait



7.3. Local Variables

Introduction

scope => from where can this information be accessed

local means restricted to current program segment

global means any software can access it

allocation => when is it created, when is it destroyed

dynamic allocation using registers or stack

permanent allocation assigned a block of memory

Example of local variables on stack

```
org $4000
; calculate sum of numbers
; Input: RegD num
; Output: RegD Sum of 1,2,3,...,num
; Errors: may overflow
```

```

; 1) binding
num set 2 ;loop counter 1,2,3
sum set 0 ;16-bit accumulator
calc
; 2) allocation
    pshd ;allocate num
    movw #0,2,-sp ;sum=0
; 3) access
;Draw a stack picture
; SP -> sum
; SP+2 -> num
; SP+4 -> return address
loop ldd sum,sp
    addd num,sp
    std sum,sp ;sum = sum+num
    ldd num,sp
    subd #1
    std num,sp ;num = num-1
    bne loop
    ldd sum,sp ;result
; 4) deallocate
    leas 4,sp
    rts

```

Example of local variables on stack, using a stack frame

Advantage: you can use the stack for other temporary

Disadvantage: slower ties the use of a register

```

    org $4000
; calculate sum of numbers
; Input: RegD num ; Output: RegD Sum of 1,2,3,...,num
; Errors: may overflow
; 1) binding
sum set -4 ;16-bit accumulator
num set -2 ;loop counter 1,2,3
calc
; 2) allocation
    pshx ;save old frame
    tsx ;create frame
    pshd ;allocate num
    movw #0,2,-sp ;sum=0
; 3) access
;Draw a stack picture relative to frame
; X-4 -> sum
; X-2 -> num
; X -> oldX
; X+2 -> return address
loop ldd sum,x
    addd num,x
    std sum,x ;sum = sum+num
    ldd num,x
    subd #1
    std num,x ;num = num-1
    bne loop
    ldd sum,x ;result
; 4) deallocate
    txs ;restore old frame
    pulx
    rts

```

7.5. Parameter passing

input parameters

data passed from calling routine into subroutine

output parameters

data returned from subroutine back to calling routine

input/output parameters

data passed from calling routine into subroutine
data returned from subroutine back to calling routine

call by reference

how

a pointer to the object is passed

why

fast for passing lots of data
simple to implement input/output parameters
both subroutine and calling routine assess same data

call by value

how

a copy of the data is passed

why

simple for small numbers of parameters
protection of the original data from the subroutine

Parameters and locals on stack, using a stack frame

Advantage: you can pass lots of data

Disadvantage: slower

Strategy:

number of parameters?

few: use registers

a lot: use the stack

size of the parameter

1 or 2 bytes: call by value

buffers: call by reference

use call by reference for read/modify/write parameters

The bottom line

Blind, Busy-wait, Interrupt synchronization

Follow the directions when performing output to LCD

Stack frame implementation

allows you to use stack for other purposes

needs dedication use RegX or Reg Y

good on machines with a lot of registers

Call by value makes a copy of the data

Call by reference passes a pointer to the original