Introducing a Subroutine Call-Return Mechanism

- How can we have a different delay for each call of the subroutine?
  - We must somehow convey the delay duration to the subroutine from the calling location.
  - There are 2 ways of doing this
    1. We simply set the value of the bl register to the desired delay before making the call; the subroutine will use this value as before.
    2. We put the value on the stack before making the call and add appropriate code to the subroutine to access it correctly.
  - Method 1 is equivalent to having a global variable
  - Method 2 is in the spirit of passing a parameter to the subroutine.
  - Both methods have advantages and disadvantages depending on your perspective and the abstraction level that you are working on.

172

Illustrating Method 1

```
;Traffic Lights with a various delay durations using subroutines

start:
  mov al, 84    ; 84 corresponds to Red-Green on the Traffic Lights
  out 01        ; Write to Traffic Lights Port
  mov bl, fa    ; initialize bl with a value representing the delay
  call 70       ;

  mov al, 48    ; 48 corresponds to Amber-Amber on the Traffic Lights
  out 01        ; Write to Traffic Lights Port
  mov bl, fe    ; initialize bl with a value representing the delay
  call 70

  mov al, 30    ; 30 corresponds to Green-Red on the Traffic Lights
  out 01
  mov bl, ef    ; initialize bl with a value representing the delay
  call 70

  mov al, 48    ; 48 corresponds to Amber-Amber on the Traffic Lights
  out 01        ; Write to Traffic Lights Port
  mov bl, fe    ; initialize bl with a value representing the delay
  call 70
jmp start

org 70 ;
 loop:
 inc bl
 cmp bl, 00    ; check to see if bl has overflowed
 jnz loop     ; if not continue incrementing and checking
```

173

Illustrating Method 2

```
;Traffic Lights with a various delay durations using subroutines

start:
  mov al, 84    ; 84 corresponds to Red-Green on the Traffic Lights
  out 01        ; Write to Traffic Lights Port
  mov bl, fa    ; initialize bl with a value representing the delay
  push bl       ; place parameter on stack
  call 70       ;

  mov al, 48    ; 48 corresponds to Amber-Amber on the Traffic Lights
  out 01        ; Write to Traffic Lights Port
  mov bl, fe    ; initialize bl with a value representing the delay
  push bl       ; place parameter on stack
  call 70

  mov al, 30    ; 30 corresponds to Green-Red on the Traffic Lights
  out 01
  mov bl, ef    ; initialize bl with a value representing the delay
  push bl       ; place parameter on stack
  call 70

  mov al, 48    ; 48 corresponds to Amber-Amber on the Traffic Lights
  out 01        ; Write to Traffic Lights Port
  mov bl, fe    ; initialize bl with a value representing the delay
  push bl       ; place parameter on stack
  call 70
jmp start
```

174

Illustrating Method 2 - continued

```
org 70 ; Subroutine
 pop cl  ;  this will be the return address.
 pop bl  ;  this is the parameter.
 push cl ;  put return address back on stack
 loop:
 inc bl
 cmp bl, 00    ; check to see if bl has overflowed
 jnz loop     ; if not continue incrementing and checking
 ret
end
```

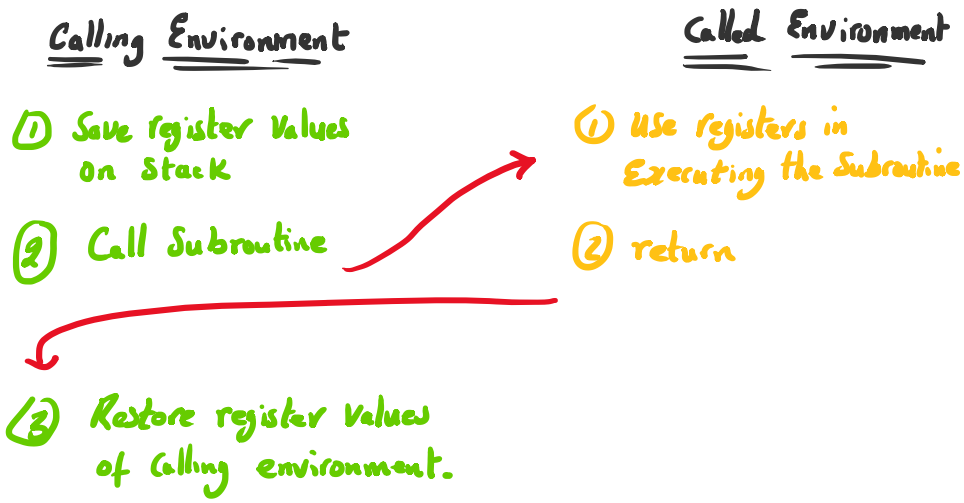
175

Why Method 2?

- Method 2 looks to be a lot more complicated
  - It is, but it is also more flexible
  - In general, we not only use the stack to pass parameters; we also use it to store the values of all the registers from the calling environment.
  - This, *context switch*, allows us to reuse all the registers in processing the code of the subroutine without fear of overwriting their values from another context.
  - When the subroutine ends, the context can be switched again – with the stored values of the calling environment registers being restored to the appropriate registers.

176

Why Method 2? –Context Switching



177

```
mov al, 30
mov bl, 31
mov cl, 32
mov dl, 33

push al    ;save the context of the calling env
push bl
push cl
push dl

call 70

pop dl  ; restore the context of thecalling env
pop cl
pop bl
pop al


mov [c0], al
mov [c1], bl
mov [c2], cl
mov [c3], dl


org 70
add al, bl
add cl, dl
add al, cl
mov [d0], al

ret
```

use registers in calling Env

Save state of calling Env

Call Subroutine

restore state of calling Env

Print state

Reuse Registers to Implement functionality of Subroutine

178

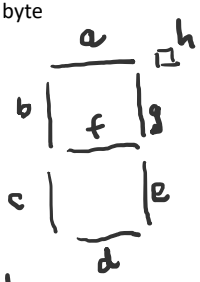
Creating a Translation Table

- The ASCII table allows us to translate between symbols and numbers. We can view the ASCII table as a translation table.
- To translate between numbers and the 7-segment display (for example) we also need to create an appropriate translation table.
-

179

Creating a Translation Table

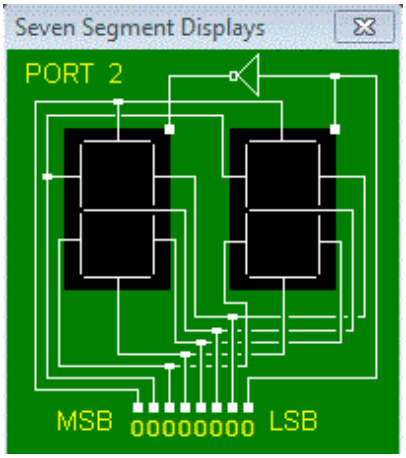
- Consider the 7-segment display:
- We could label all of the LEDs, comprising the 7 segments, in order from the most significant bit (MSB) to the least significant bit (LSB) in the associated byte



a b c d e f g h

msb

LSB = Select

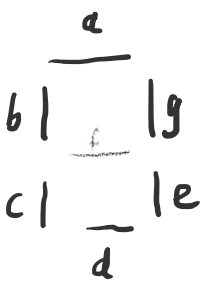


Select = 1 ⇒ R.H.S. Display active
Select = 0 ⇒ L.H.S. Display active

180

Creating a Translation Table

- To display 0, for example, on a display requires
- a, b, c, d, e, g = 1 and f = 0
- The value of h will determine the display



181

Creating a Translation Table

Image	a	b	c	d	e	f	g	h	LHS	RHS
0	1	1	1	1	1	0	1	0|1	FA	FB
1	0	0	0	0	1	0	1	0|1	0A	0B
2	1	0	1	1	0	1	1	0|1	B6	B7
3	1	0	0	1	1	1	1	0|1	9E	9F
4	0	1	0	0	1	1	1	0|1	4E	4F
5	1	1	0	1	1	1	0	0|1	DC	DD
6	1	1	1	1	1	1	0	0|1	FC	FD
7	1	0	0	0	1	0	1	0|1	8A	8B
8	1	1	1	1	1	1	1	0|1	FE	FF
9	1	1	0	0	1	1	1	0|1	CE	CF

182

Creating a Translation Table

- In Samphire, we can use the Assembler Directives, DB and ORG to place a specific byte into memory starting at a specific memory location.
- Thus,

```
org 50
DB  FA
```

Will place the byte `FA` into memory at address 50.

Each subsequent `DB` directive will place a byte in subsequent memory addresses.

183

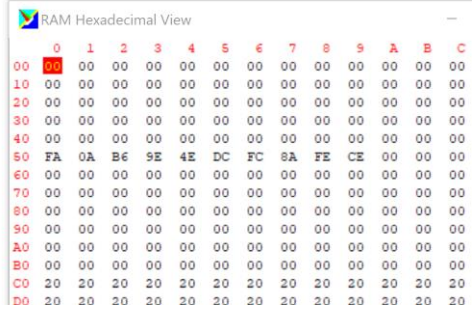
Creating a Translation Table

- Thus, we can place the complete translation table for the LHS of the 7-segment display into RAM starting at address 50, as follows:

```
org 50

DB FA
DB 0A
DB B6
DB 9E
DB 4E
DB DC
DB FC
DB 8A
DB FE
DB CE

end
```



Note that there is no need to put both tables into memory, since the RHS display values can be calculated by adding 1 to the LHS display values

184

Seven Segment Display

```
;The following code displays a numberic key from the keyboard on the 7-segment display

loop:
  in 00         ; kbd input put in al
  sub al, 30    ; subtract ascii value of 0 to get numberic value of key
  add al, 50    ; index into the translation table by adding the numberic kbd key
                ; value to the base address of the translation table.
  mov al, [al] ; copy the 7-seg code at address given in al into al
  out 02        ; write to 7-seg display
  jmp loop

org 50

DB FA ; 7-seg code for symbol for 0
DB 0A ; 7-seg code for symbol for 1
DB B6 ; 7-seg code for symbol for 2
DB 9E ; 7-seg code for symbol for 3
DB 4E ; 7-seg code for symbol for 4
DB DC ; 7-seg code for symbol for 5
DB FC ; 7-seg code for symbol for 6
DB 8A ; 7-seg code for symbol for 7
DB FE ; 7-seg code for symbol for 8
DB CE ; 7-seg code for symbol for 9

end
```

185