Lab 6 Subroutines

I. Subroutines

In lab 4, we learned how to write a simple subroutine, that is, a subroutine without parameter passing nor a returned value. For example, in C or Java, the prototype of a simple subroutine looks like this:

```
void delay()
```

In today's lab, we are going to learn how to write subroutines with parameter(s) and returned value.

Example 1: passing two parameters by value using a stack on SRAM (the data memory), the returned value is saved in register pair sumH:sumL (r19:r18). In C or Java, the equivalent subroutine looks like this:

```
int add_num (int n1, int n2)
```

In assembly language, the parameters n1 and n2 are pushed onto the stack before calling the subroutine add_num. Download params.asm and add it to your project.

The diagram of the internal memory of the SRAM when "ldd n1, Z+9" is executed:

Address	content	details	notes
0x21FF	0xEE	parameter (Z + 9)	In the main, the caller pushes the parameters onto the stack.
0x21FE	0xCC	parameter (Z + 8)	
0x21FD	ret	return address	The return address (the address of the next command) is pushed onto the stack automatically when "call add_num" is executed.
0x21FC	ret	return address	
0x21FB	ret	return address	
0x21FA	ZL	saved register	In the subroutine, the callee pushes those registers which will be used onto the stack to protect them from being changed.
0x21F9	ZH	saved register	
0x21F8	n1	saved register	
0x21F7	n2	saved register	
0x21F6		<-SP	"0x21F6" is stored in SP (**S**tack **P**ointer)
……			
0x0200			
……			
0x01FF ~ 0x0000		General Purpose Registers and I/O Registers	.DSEG

Build params.asm and run the code, press F11 and observe the contents of the registers SP, Registers Z, r16, r1 and r0, r19:r18

Example 2: passing parameter by reference using a stack on SRAM (the data memory), no returned value.
Download lab6.asm, read void strcpy (src, dest) subroutine. Understand it, reconstruct the stack frame. Implement unsigned int strlength (str) subroutine using the two subroutines we have just learnt as examples.

Read the code in lab6.asm. Download stackFramePractice.docx and write the stack frame for strcpy using the example on the previous page.

Address	content	details	notes
0x21FF			In the main, the caller pushes the parameters onto the stack.
			The return address (the address of the next command) is pushed onto the stack automatically when "call strcpy" is executed.
			In the subroutine, the callee pushes those registers which will be used onto the stack to protect them from being changed.
0x21F1		← SP	
	……..		

Design the stack frame for strlength:

Address	content	details	notes
0x21FF			In the main, the caller pushes the parameters onto the stack.
			The return address (the address of the next command) is pushed onto the stack automatically when "call strlength" is executed.
			In the subroutine, the callee pushes those registers which will be used onto the stack to protect them from being changed.
0x21F5		← SP	
	……..		

II. Download localVariable.asm. Try to understand how local variable is used.

III. Implement *strncpy(char *dest, const char *src, size_t n). See the description copied from https://www.tutorialspoint.com/c_standard_library/c_function_strncpy.htm:

The C library function char *strncpy(char *dest, const char *src, size_t n) copies up to n characters from the string pointed to, by src to dest. In the case where the length of src is less than that of n, the remainder of dest will be padded with null bytes.