Computer Systems, Spring 2020

Week 8 Lab

Pointers: Records, Arrays, Procedures, I/O

The aims of this exercise are to

1. Review some basic techniques.
2. Study how to access arrays using pointers as well as indices. This technique is essential in advanced programming.
3. Learn about a more sophisticated procedure call/return mechanism, which handles stack overflow correctly.
4. Practice translating a high level algorithm to low level and assembly language.

This exercise is not assessed, but it's important to work through it. The important point is that you understand these techniques, because they will be used in the assessed exercise starting next week.

1 Review pointers

Download the program `Pointer.asm.txt`. Study the program, and step through it with the Sigma16 system. You don't need to write any new code, but you do need to understand this program: it contains many simple examples of using pointers.

2 Accessing record fields via pointer

Now we will apply the technique illustrated in the `Pointer` program to allow flexible access to records.

Download the program `RecordsEXERCISE.asm.txt`. Study the program, and step through it with the Sigma16 system. Complete the program by filling in the necessary instructions at the points labelled `; INSERT SOLUTION HERE`. The comments in the program tell you what to do.

3 PrintIntegers

PrintIntegers is a program that defines a procedure that takes an integer, converts it to a character string, and prints it. The core of the program is a procedure ShowInt.

```
procedure ShowInt (x:Int, *bufstart:Char, bufsize:Int) : Int
```

This procedure takes three parameters: an integer `x`, a pointer to a buffer in memory where the character string will be stored, and an integer `bufsize` which gives the maximum size of the string. If the number fits within bufsize

characters (e.g. 23 fits in a buffer of size 2 or more) the procedure stores the string and inserts leading spaces if needed. For example, if the buffer size is 5, then there will be three leading spaces as 23 requires only two characters. If the number simply doesn't fit (e.g 3862 doesn't fit in 3 characters) the procedure fills the buffer with hash characters. The high level algorithm is given. The problem has two parts:

1. Translate the high level algorithm into low level. Use the translation patterns, don't just write random code.

2. Translate the low level algorithm into assembly language.

Complete the program by filling in the necessary instructions at the points labelled `; INSERT SOLUTION HERE`.

Here is what the output looks like:

```
37
Cat
(    23)
(0)
(32767)
(####)
(-1)
(#)
(-32768)
(#####)
(   32)
(   17)
(  456)
(  1066)
(-30978)
(2001)
(3)
(   47)
(   13)
(19)
(  103)
(  103)
(##)
(   47)
(    48)
(   49)
(29371)
(6285)
(   264)
(##)
(  -92)
(-1)
(###)
(42)
```