

Procedures and Functions in PeANUt

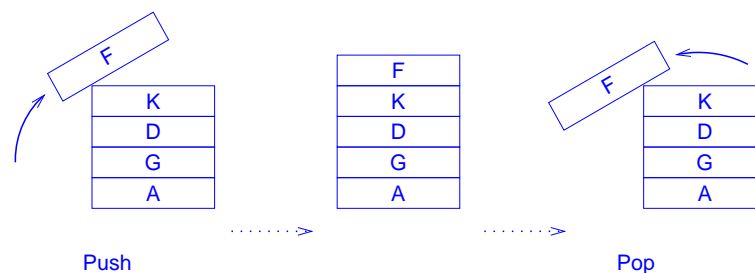
- number systems (bases) in `.mli` files
- procedure / function calls
- nested procedures
- the stack:
 - stack pointer register
 - stack addressing mode
 - the stack frame
- ref: [PeANUt Spec,]; additional reading: [O'H&Bryant, sect 3.7]
- other matters:
 - Assignment 1: getting the output right and understanding `sdiff` output
 - MSE & spare Peanuts
 - revise pointers and dynamically allocated memory P2 (p12-13), P3 (p5-6)
also questions in `pointerQs.txt`

Simple Procedure Calls

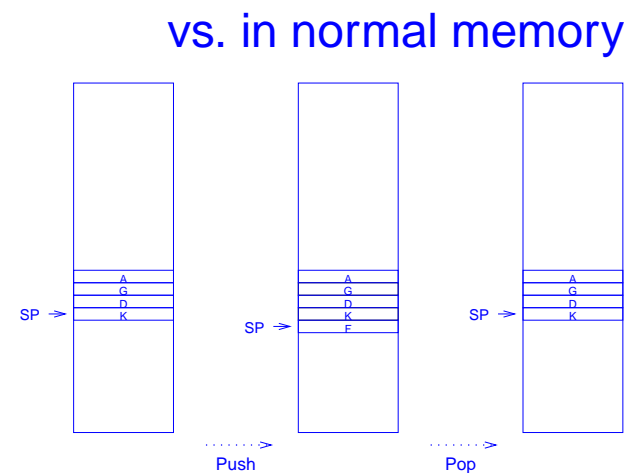
- motivation:
 - often, the instruction set does not include some operation that is regularly required
 - the user can effectively extend the instruction set by using procedures / functions
 - procedures can be written in PeANUt (like functions in C)
- PeANUt procedures:
 - the instructions CallProcedure and Return are important
 - CallProcedure allows the PC to be *remembered*, and a new PC value is given (so that execution can continue from a different place)
 - Return retrieves the *remembered* PC value and resets the PC to this value (so that execution continues where it left off)

Nested Procedures (Procedures within Procedures)

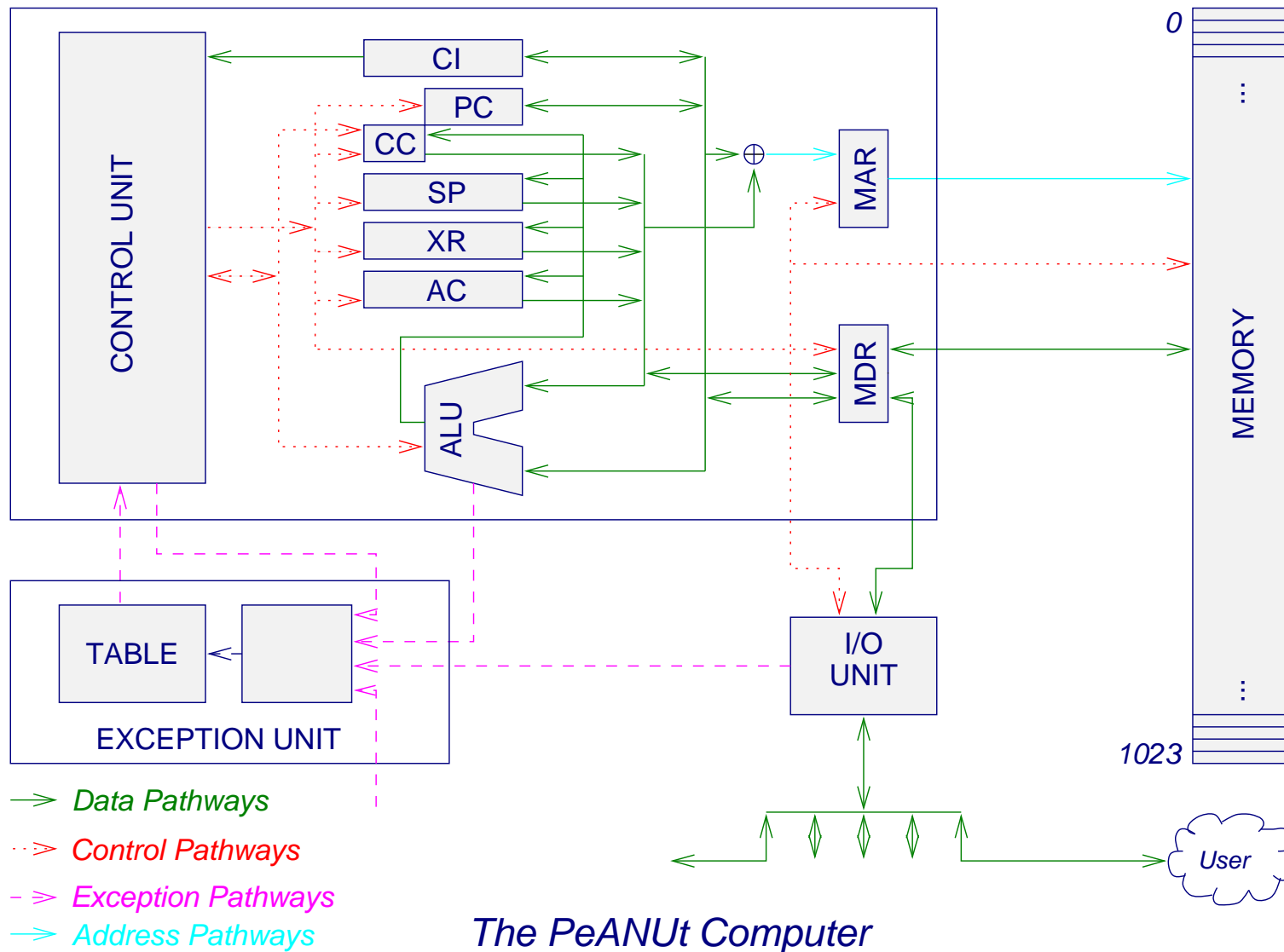
- is it possible to nest procedures?
 - can $>$ one PC value be saved? if so, how to do so in an organised way?
- what is a stack? Consider a pile of books:
 - LIFO (Last In, First Out) - compare with a queue (FIFO)



- how can we make a stack?
 - in special hardware (complex, finite size)



Stack Support in the PeANUt Architecture



The Stack Pointer (SP) Register

- points to (contains address of) top of the stack
 - automatically increases when a CallProcedure is made
 - automatically decreases when a Return is made
- can also be manually incremented or decremented
- PC values are stored on the stack (to allow return from procedures)
 - procedure nesting is thus limited only by potential stack size
 - stack size limited only by available memory
- the stack can be used to pass parameters

Using the PeANUt's Stack

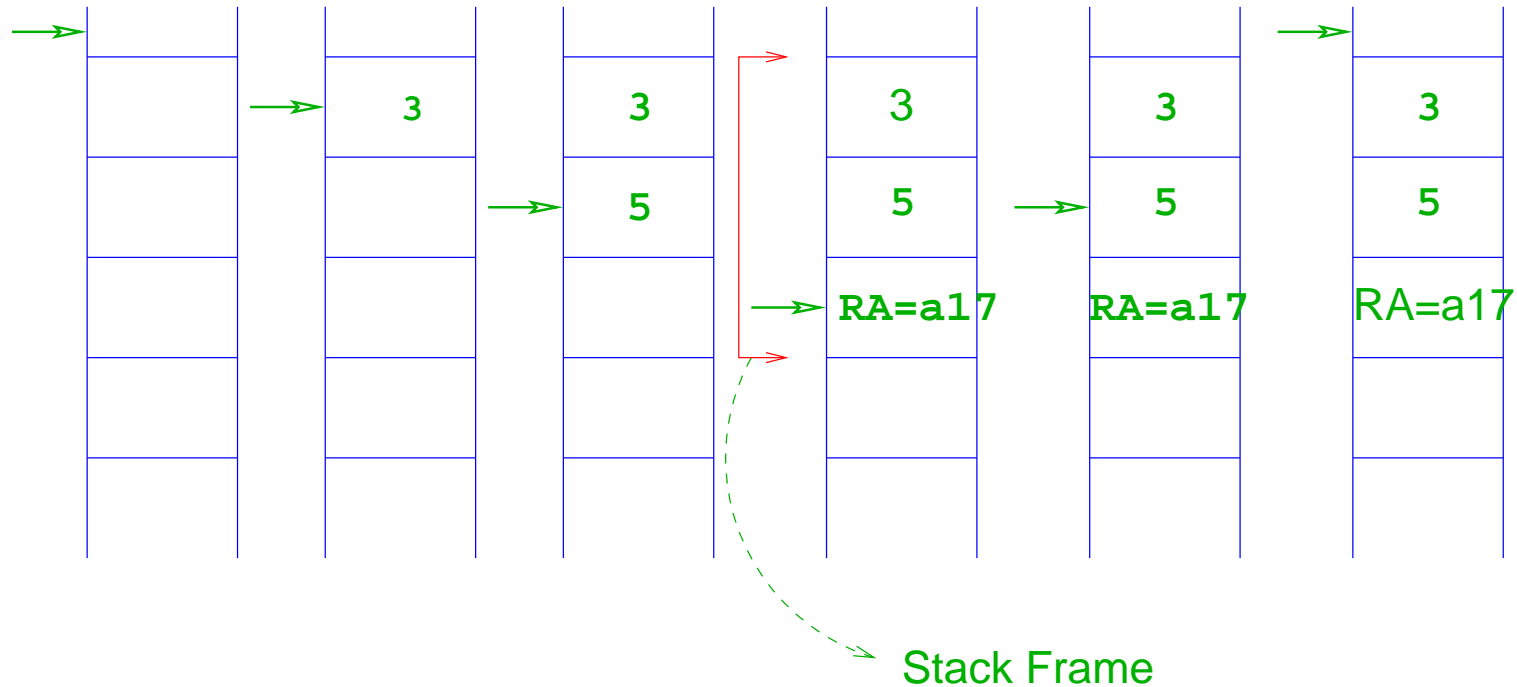
- the PeANUt convention: stack frames are constructed in the following order
 - return value *(later...)*
 - parameters
 - return address
 - local variables *(later...)*
- for procedure calls with no parameters:
 - to call the procedure, just use the CallProcedure instruction, which:
 - ◆ increments SP by 1
 - ◆ the current PC value is placed on top of the stack
 - ◆ the procedure's start address is placed in the PC
 - to return from a procedure, use the Return instruction, which:
 - ◆ places the value on top of the stack into the PC
 - ◆ decrements SP by 1
 - previous example (printing of *A*B*)

Procedure Calls: Parameters

- unlike the return address, parameters must be *manually* placed on, and removed, from the stack
- before calling a procedure, parameters must be placed on the stack
 1. increment SP
 2. store value to the top of the stack→ repeat for each parameter
- after returning from a procedure, space on the stack for the parameters must be deallocated
 1. decrement SP by the number of parameters

Parameters Example Continued

```
load 3      load 5  
incSP 1     incSP 1  
st mem[SP]  st mem[SP] call foo  return  incSP -2
```



- something to think about: value vs reference parameters
- writing code of this complexity is getting increasingly difficult in machine language!
- next: translating C into PeANUt assembly language