

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)

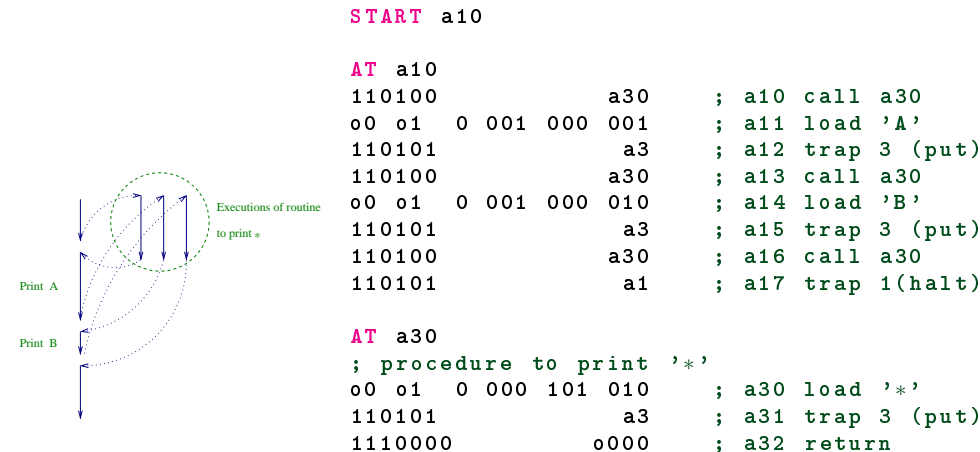
Using Other Bases in .mli files

- writing all instructions in binary can be tedious, although is often clearer
- notation:

<ul style="list-style-type: none"> ■ octal (o) (1 digit → 3 bits) <ul style="list-style-type: none"> o100 → 001 000 000 o123 → 001 010 011 o767 → 111 110 111 ■ hexadecimal (h) (1 digit → 4 bits) <ul style="list-style-type: none"> h10 → 0001 0000 h79 → 0111 1001 h9D → 1001 1101 	<ul style="list-style-type: none"> ■ decimal (d) (n digits → 16 bits) <ul style="list-style-type: none"> d5 → 0000 0000 0000 0101 d64 → 0000 0000 0100 0000 d79 → 0000 0000 0100 1111 ■ address (a, octal) (n digits → 10 bits) <ul style="list-style-type: none"> a5 → 0 000 000 101 a17 → 0 000 001 111 a167 → 0 001 110 111
---	--
- examples:
 - 001 001 0 000 101 000 → o1 o1 a50 ; load a50 (direct)
 - 110101 0 000 000 011 → o6 o5 a3 ; trap 3
 - 1110101 000 000 000 → hE o5 o000 ; compXR

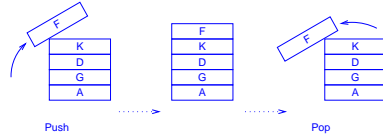
Procedure Example: procedure-example.mli

- write a program that prints out *A*B*

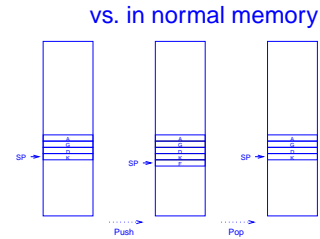


Nested Procedures (Procedures within Procedures)

- 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)



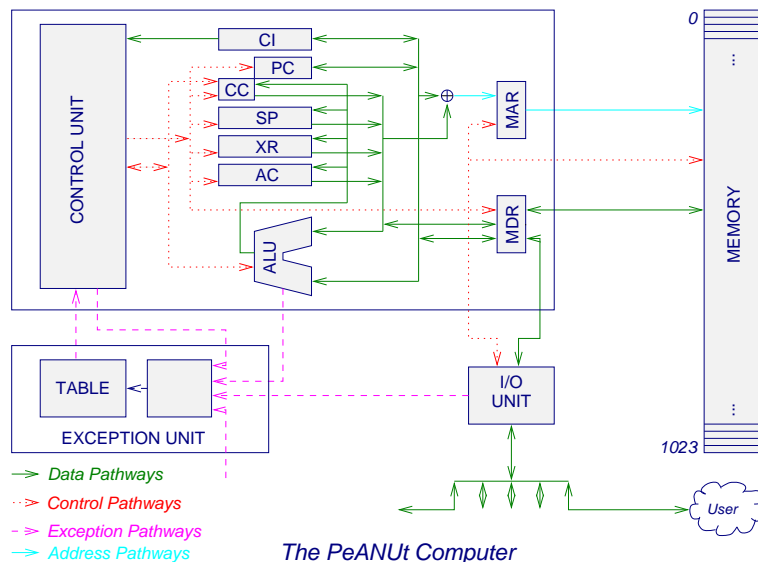
- how can we make a stack?
 - in special hardware (complex, finite size)



The Stack Pointer (SP) Register

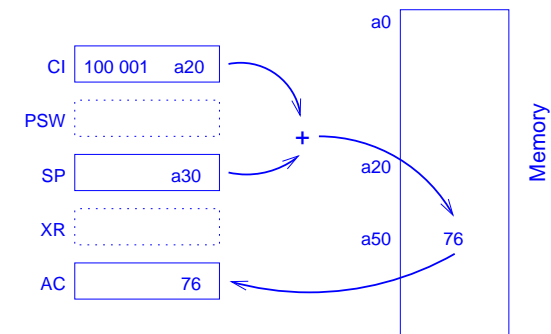
- points to (contains address of) top of the stack
 - automatically increases when a CallProcedure is made
 - automatically decreases when a Return is made
- can also be manually incremented or decremented
- PC values are stored on the stack (to allow return from procedures)
 - procedure nesting is thus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

Stack Support in the PeANUt Architecture



Stack Addressing Mode (mode bits 100)

- the address of the operand is given by adding the contents of the stack pointer (SP) and the opspec
- i.e. the operand is at $\text{mem}[\text{SP} + \text{opspec}]$



- similar to indexed addressing mode (but in what sense is the opspec different?)

