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
Using Other Bases in .mli files

● writing all instructions in binary can be tedious, although is often clearer

● notation:
  ■ octal (o) (1 digit → 3 bits)
    o100  →  001 000 000
    o123  →  001 010 011
    o767  →  111 110 111
  ■ hexadecimal (h) (1 digit → 4 bits)
    h10  →  0001 0000
    h79  →  0111 1001
    h9D  →  1001 1101
  ■ decimal (d) (n digits → 16 bits)
    d5  →  0000 0000 0000 0101
    d64 →  0000 0000 0100 0000
    d79 →  0000 0000 0100 1111
  ■ address (a, octal) (n digits → 10 bits)
    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 0000 ; compXR
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)
Procedure Example: procedure-example.mli

- write a program that prints out \( *A*B* \)

```
START a10

AT a10
110100 a30 ; a10 call a30
00 o1 0 001 000 001 ; a11 load 'A'
110101 a3 ; a12 trap 3 (put)
110100 a30 ; a13 call a30
00 o1 0 001 000 010 ; a14 load 'B'
110101 a3 ; a15 trap 3 (put)
110100 a30 ; a16 call a30
110101 a1 ; a17 trap 1(halt)

AT a30
; procedure to print '∗'
00 o1 0 000 101 010 ; a30 load '∗'
110101 a3 ; a31 trap 3 (put)
1110000 o000 ; a32 return
```
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) vs. in normal memory
Stack Support in the PeANUt Architecture

The PeANUt Computer
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 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 mem[SP + opspec]

- similar to indexed addressing mode (but in what sense is the opspec different?)
Procedure Context: The Stack Frame

- what information is relevant for the duration of the execution of a procedure / function (and not at any other time)?
  - return address
  - parameters (sometimes)
  - return value (sometimes)
  - local variables (sometimes)

- the stack is used to store all the information associated with the execution of a procedure: this is known as a stack frame

- conventions define the order of data within a stack frame

- lifetime of stack frames:
  - when the procedure is called, a new stack frame is created (parameters are initialised)
  - when the procedure returns, its stack frame is removed from the stack

- allows variables associated with all currently executing procedures to be accessible, without having them permanently allocated
Using the PeANUUt’s Stack

- **the PeANUUt 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
Procedure with Parameters: Example

main program:

```c
int main() {
    ...
    foo(3,5);
    ...
}
```

procedure:

```c
void foo(int a, int b) {
    printf("%d", a);
    printf("%d", b);
}
```

START a10
AT a10 ; main program
00 01 a3 ; load 3 (imm)
110011 a1 ; inc SP by 1
04 02 a0 ; store mem[SP]
110011 a1 ; inc SP by 1
04 02 a0 ; store mem[SP]
110100 a30 ; call a30
110011 a1776 ; inc SP by -2
110101 a1 ; trap 1 (halt)

AT a30 ; foo()
04 01 a1776 ; load mem[SP-2]
00 03 a60 ; add 48 (imm)
110101 a3 ; trap 3 (put)
04 01 a1777 ; load mem[SP-1]
00 03 a60 ; add 48 (imm)
110101 a3 ; trap 3 (put)
111000 a0 ; return

COMP2300 P4: Procedures in PeANUt 2009
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