

- 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]
- revise 2006 exam Q2(d,e) (C Programming)

- 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"> <li>■ octal (o) (1 digit → 3 bits)</li> <li>o100 → 001 000 000</li> <li>o123 → 001 010 011</li> <li>o767 → 111 110 111</li> </ul>	<ul style="list-style-type: none"> <li>■ decimal (d) (n digits → 16 bits)</li> <li>d5 → 0000 0000 0000 0101</li> <li>d64 → 0000 0000 0100 0000</li> <li>d79 → 0000 0000 0100 1111</li> </ul>
<ul style="list-style-type: none"> <li>■ hexadecimal (h) (1 digit → 4 bits)</li> <li>h10 → 0001 0000</li> <li>h79 → 0111 1001</li> <li>h9D → 1001 1101</li> </ul>	<ul style="list-style-type: none"> <li>■ address (a, octal) (n digits → 10 bits)</li> <li>a5 → 0 000 000 101</li> <li>a17 → 0 000 001 111</li> <li>a167 → 0 001 110 111</li> </ul>
- examples:
  - 001 001 0 000 101 000 → o1 o1 a50 ; load 50\_8
  - 110101 0 000 000 011 → o6 o5 a3 ; trap 3
  - 1110101 000 000 000 → hE o5 a0 ; compXR

## Procedure Example: procedure-example.mli

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





## Procedure with Parameters: Example

main program:

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

START a10

AT a10

```
...
load #3
incSP #1
store AC -> mem[SP]
load #5
incsp #1
store AC -> mem[SP]
call a30
incsp #-2
trap 1
```

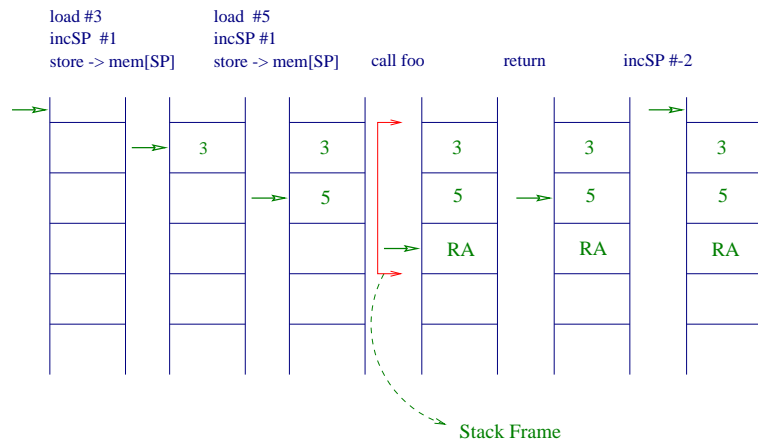
procedure:

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

AT a30

```
load mem[SP-2]
add #48 ; convert into ASCII char
trap 3 ; put character
load mem[SP-1]
add #48 ; convert into ASCII char
trap 3 ; put character
return
```

## Parameters Example Continued



- 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