Co-routines and "Threads"

```
Coroutine1::
    DoSomeWork();
    Resume(Coroutine2);
    DoSomeMoreWork();
    Resume(Coroutine2);
    exit();

Coroutine2::
    DoSomeWork();
    Resume(Coroutine1);
    DoSomeMoreWork2();
    Resume(Coroutine1);
    exit();
```

State of computation (process, thread)

For an executing program, state is:

- All registers (including IP, SP, PSW)
- Local variables and arguments
- Other variables (global): saved per process.
- Other state (files, devices): partially saved per process.

If all state is saved, program can be suspended and then resumed without adverse effect.

To implement threads, can ignore global variables and IO state

Thread State = all registers + the stack

Code that can be interrupted and safely called again "in parallel" (re-entered) is called **re-entrant**

Reentrant code

Reentrant code may not hold any static (or global) non-constant data.

Reentrant code may not modify its own code.

Reentrant code may not call non-reentrant computer programs or routines.

But reentrant code usually does modify local variables!

Therefore need:
Local variables and other local state
separate for each activation

Using stack for activation frame, code that changes only local variables is **reentrant**

As a special case, reentrant code supports recursion

Implementing co-routines

Each co-routine (thread) has its own stack.

Co-routines are initialized, then can be SUSPENDED and RESUMED at any point. (Synonyms: co-init and co-call)

Co-routines can call procedures normally.

Keep a struct for each co-routine, with:

- Initial entry point
- Stack pointer
- (Optionally) base pointer
- (Optionally) initialization flag
- Actual stack

For threads need additional information, e.g. for scheduling.

Using several stacks

Some processors have multiple SPs

Motorola 680X0: USP, ISP, MSP
Also, any An can be a SP

In general case can save SP,
then re-load SP (Intel 80X86):

```
mov  [spsave1], esp  ; Save esp
mov  esp, [spsave2]  ; Restore other esp
```

Data structure for coroutines

```
numco:  dd      3
CORS:   dd      CO1
        dd      CO2
        dd      CO3

 STKSZ   equ     16*1024
 CODEP   equ     0  ; constant offsets
 FLAGSP  equ     4
 SPP     equ     8

; Structure for first co-routine
CO1:    dd      CO1code
Flags1: dd      0
SP1:    dd      STK1+STKSZ

STK1:   resb    STKSZ
```

Code for CO-INIT

```
  ; Assuming EBX is pointer to COn
co_init:
  pusha
  bts    dword [EBX+FLAGSP],0 ; initialized?
  jc     init_done
  mov    EAX,[EBX+CODEP] ; Get initial IP
  mov    [SPT], ESP
  mov    ESP,[EBX+SPP]   ; Get initial SP
  mov    EBP, ESP        ; Also use as EBP
  push   EAX ; Push initial "return" address
  pushf      ; and flags
  pusha      ; and all other regs
  mov    [EBX+SPP],ESP   ; Save new SP
  mov    ESP, [SPT]      ; Restore old SP
init_done:
  popa
  ret
```

Code for CO-CALL (RESUME)

EBX: pointer to co-init struct of co-routine
to be resumed.

CURR: pointer to co-init structure of the curent
co-routine.

```
resume:
  pushf     ; Save state of caller
  pusha
  mov   EDX, [CURR]
  mov   [EDX+SPP],ESP  ; Save current SP
do_resume:
  mov   ESP, [EBX+SPP] ; Load SP (resumed co)
  mov   [CURR], EBX
  popa   ; Restore resumed co-routine state
  popf
  ret    ; "return" to resumed co-routine!
```