Procedure Declaration

The syntax of procedure declaration is the following:

```
PROC name type
; body of procedure
ret
ENDP name
```

type can be `NEAR` (in same segment) or `FAR` (in a different segment) -- if omitted, `NEAR` is assumed

The CALL Instruction

CALL invokes a procedure

CALL has two forms, *direct*

```
call name
```

where *name* is the name of a procedure, and *indirect*

```
call address_expression
```

(not generally recommended)

where *address_expression* specifies a register or memory location containing the address of a procedure

Executing a CALL

The return address to the calling program (the current value of the IP) is saved on the stack

IP get the offset address of the first instruction of the procedure (this transfers control to the procedure)

FAR procedures must process CS:IP instead of just IP

The RET instruction

To return from a procedure, the instruction

```
ret pop_value
```

is executed

The integer argument *pop_value* is optional

ret causes the stack to be popped into IP

If *pop_value* N is specified, it is added to SP -- in effect removes *N* additional bytes from the stack

Parameter Passing

Parameters can be passed to the called procedure using any of the following ways

. Through Registers

. Through the stack

. Through shared memory

The values returned if any by the procedure can also be returned in the same way.

Far Procedure

A procedure which is called from the other segment is known as far procedure. An intersegment CALL instruction is used to call a far procedure which save both the segment base and the offset of the next instruction onto the stack. The far procedure uses the far RET to return from the procedure by retrieving the offset and the segment address from the stack to resume the execution from correct point in the segment of the caller.