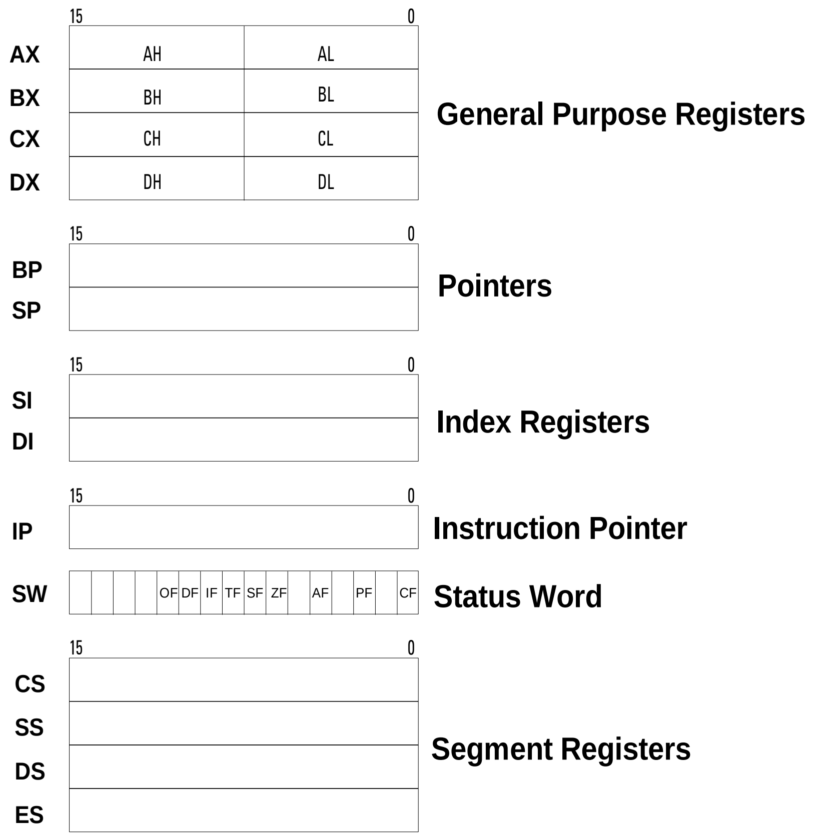
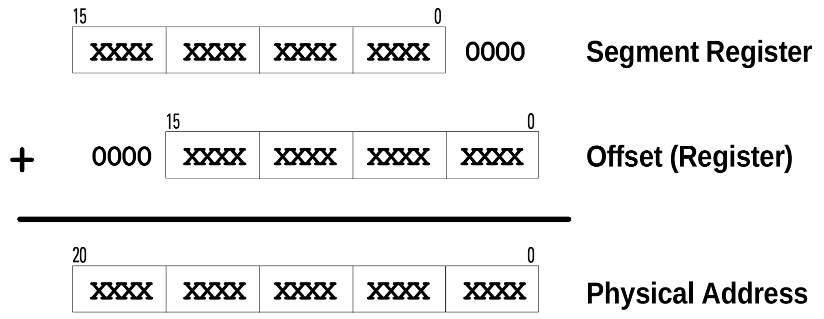
uC/OS-II Part 5: 80x86 Port

Prof. Li-Pin Chang

ESSLab@NCTU

Registers and Addressing of x86 Real Mode (or virtual 86 mode)





OS_CPU.H

```c
typedef unsigned char  BOOLEAN;
typedef unsigned char  INT8U;
typedef signed   char  INT8S;
typedef unsigned int   INT16U;
typedef signed   int   INT16S;
typedef unsigned long  INT32U;
typedef signed   long  INT32S;
typedef float          FP32;
typedef double         FP64;
```

OS_CPU.H

```
#define   OS_CRITICAL_METHOD     2

#if       OS_CRITICAL_METHOD == 1
#define   OS_ENTER_CRITICAL()   asm   CLI
#define   OS_EXIT_CRITICAL()    asm   STI
#endif

#if       OS_CRITICAL_METHOD == 2
#define   OS_ENTER_CRITICAL()   asm {PUSHF; CLI}
#define   OS_EXIT_CRITICAL()    asm   POPF
#endif

#if       OS_CRITICAL_METHOD == 3
#define   OS_ENTER_CRITICAL()   (cpu_sr = OSCPUSaveSR())
#define   OS_EXIT_CRITICAL()    (OSCPURestoreSR(cpu_sr))
#endif


#define   OS_STK_GROWTH         1

#define   uCOS                  0x80

#define   OS_TASK_SW()          asm   INT   uCOS

OS_CPU_EXT  INT8U  OSTickDOSCtr;
```

OS_CPU_C.C

- OSTaskStkInit
- OSTaskStkInit_FPE_x86
  - Borland floating point emulator
- Hook functions
  - OSTaskCreateHook()
  - OSTaskDelHook()
  - OSTaskSwHook()
  - OSTaskIdleHook()
  - OSTaskStatHook()
  - OSTimeTickHook()
  - OSInitHookBegin()

OSTaskStkInit()

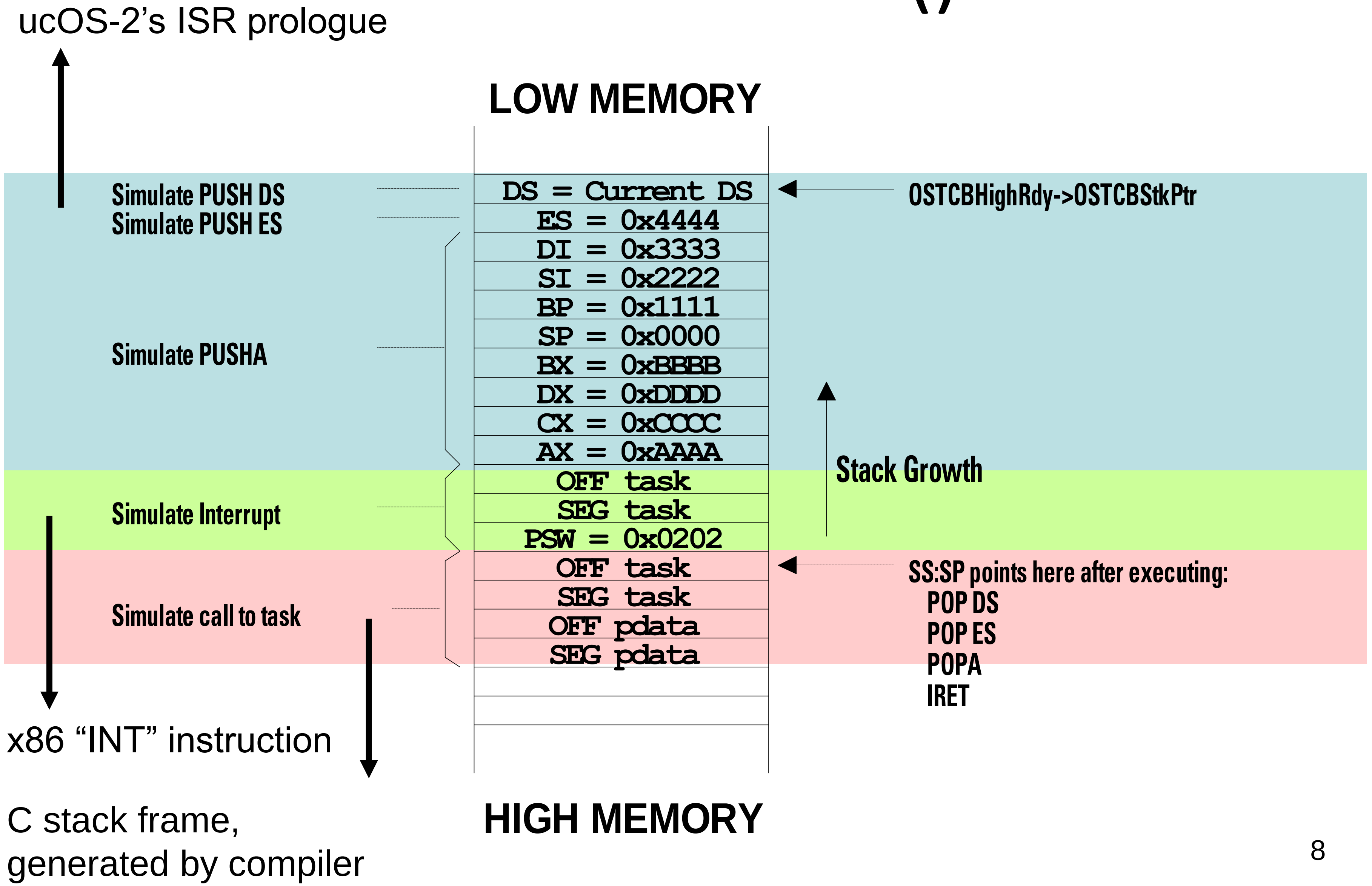
- All ready tasks are “interrupted and about to leave an ISR”
  - A newly created task is a ready task
- Emulate the stack context of a new task as if the task is “interrupted and about to leave an ISR”

```
_OSTickISR  PROC   FAR
;
       PUSHA           ; Save current task's context
       PUSH   ES       ;
       PUSH   DS       ;
…
…
```

```
_OSCtxSw    PROC   FAR
;
       PUSHA           ; Save current task's context
       PUSH   ES       ;
       PUSH   DS       ;
…
…
```

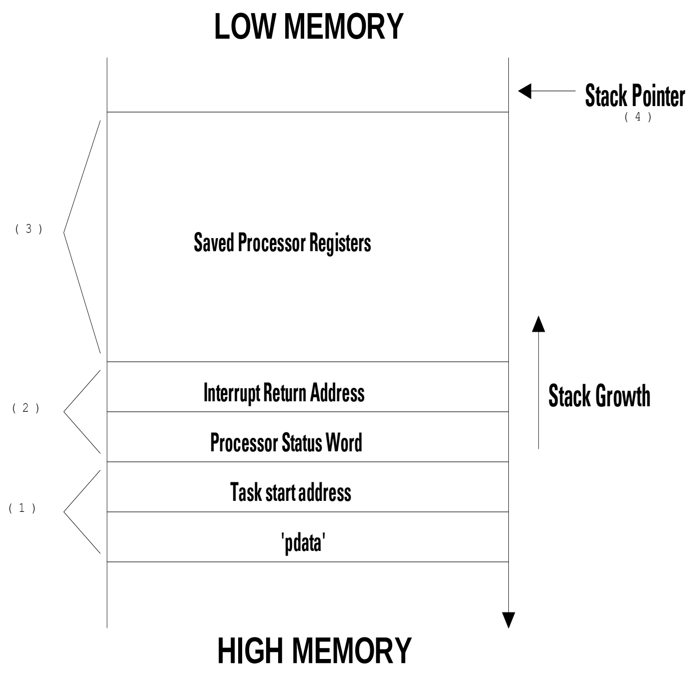
The prologue of ISR

OSTaskStkInit()



OS_CPU.C – OSStkInit()

```
OS_STK  *OSTaskStkInit (void (*task)(void *pd),
        void *pdata, OS_STK *ptos, INT16U opt) {
        INT16U *stk;
        opt     = opt;
        stk     = (INT16U *)ptos;
        *stk--  = (INT16U)FP_SEG(pdata);
        *stk--  = (INT16U)FP_OFF(pdata);
        *stk--  = (INT16U)FP_SEG(task);
        *stk--  = (INT16U)FP_OFF(task);
        *stk--  = (INT16U)0x0202;
        *stk--  = (INT16U)FP_SEG(task);
        *stk--  = (INT16U)FP_OFF(task);
        *stk--  = (INT16U)0xAAAA;
        *stk--  = (INT16U)0xCCCC;
        *stk--  = (INT16U)0xDDDD;
        *stk--  = (INT16U)0xBBBB;
        *stk--  = (INT16U)0x0000;
        *stk--  = (INT16U)0x1111;
        *stk--  = (INT16U)0x2222;
        *stk--  = (INT16U)0x3333;
        *stk--  = (INT16U)0x4444;
        *stk    = _DS;
        return ((OS_STK *)stk);
}
```



Called by OSTaskCreate()

OS_CPU_A.ASM

- OSStartHighRdy
  - Context switch on OSStart()
- OSCtxSw
  - Task-level context switch (via int 80h)
- OSIntCtxSw
  - Interrupt-level context switch
- OSTickISR
  - Clock tick ISR

OSStartHighRdy()

```
_OSStartHighRdy  PROC FAR

            MOV    AX, SEG _OSTCBHighRdy          ; Reload DS
            MOV    DS, AX                         ;
;
            CALL   FAR PTR _OSTaskSwHook          ; Call user defined task switch hook
;
            MOV    AL, 1                          ; OSRunning = TRUE;
            MOV    BYTE PTR DS:_OSRunning, AL     ;   (Indicates that multitasking has started)
;
            LES    BX, DWORD PTR DS:_OSTCBHighRdy ; SS:SP = OSTCBHighRdy->OSTCBStkPtr
            MOV    SS, ES:[BX+2]                  ;
            MOV    SP, ES:[BX+0]                  ;
;
            POP    DS                             ; Load task's context
            POP    ES                             ;
            POPA                                  ;
;
            IRET                                  ; Run task

_OSStartHighRdy  ENDP
```

Called by OSStart()

OSCtxSw()

- A task-level context switch is accomplished by triggering a software interrupt (the macro OS_TASK_SW)
- The software trap vectors to the OSCtxSw
- OSCtxSw is installed at ISR 0x80

OSCtxSw()

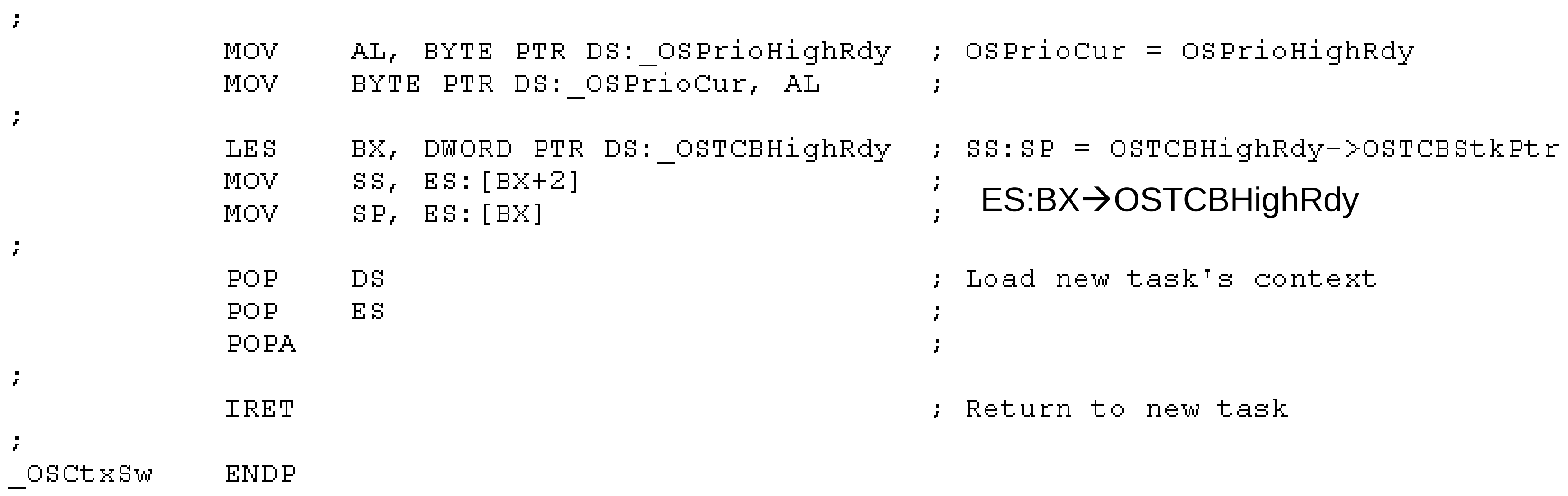
```
_OSCtxSw    PROC   FAR
;
            PUSHA                                ; Save current task's context
            PUSH   ES                            ;
            PUSH   DS                            ;
;
            MOV    AX, SEG _OSTCBCur             ; Reload DS in case it was altered
            MOV    DS, AX                        ;
;
            LES    BX, DWORD PTR DS:_OSTCBCur    ; OSTCBCur->OSTCBStkPtr = SS:SP
            MOV    ES:[BX+2], SS                 ;
            MOV    ES:[BX+0], SP                 ;
;
            CALL   FAR PTR _OSTaskSwHook         ; Call user defined task switch hook
;
            MOV    AX, WORD PTR DS:_OSTCBHighRdy+2 ; OSTCBCur = OSTCBHighRdy
            MOV    DX, WORD PTR DS:_OSTCBHighRdy  ;
            MOV    WORD PTR DS:_OSTCBCur+2, AX   ;
            MOV    WORD PTR DS:_OSTCBCur, DX     ;
```

Save the current SS:SP to TCB

Change the pointer *OSTCBCur
A pointer is of 4 bytes

OSTCBHighRdy→ task to be switched to
OSTCBCur→the currently running task

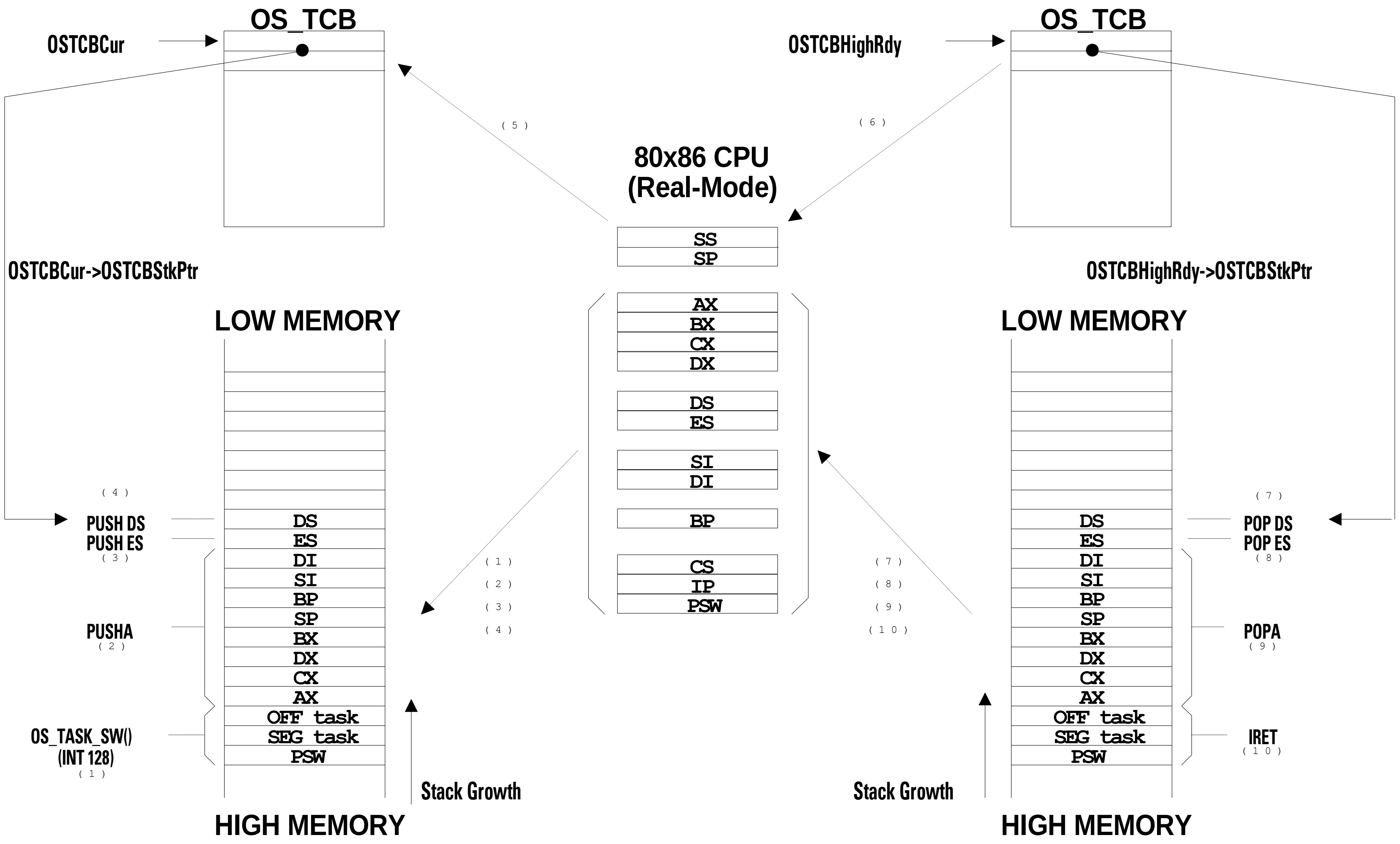
OSCtxSw()



```
;
            MOV    AL, BYTE PTR DS:_OSPrioHighRdy  ; OSPrioCur = OSPrioHighRdy
            MOV    BYTE PTR DS:_OSPrioCur, AL      ;
;
            LES    BX, DWORD PTR DS:_OSTCBHighRdy  ; SS:SP = OSTCBHighRdy->OSTCBStkPtr
            MOV    SS, ES:[BX+2]                   ;
            MOV    SP, ES:[BX]                     ;
;
            POP    DS                              ; Load new task's context
            POP    ES                              ;
            POPA                                   ;
;
            IRET                                   ; Return to new task
;
_OSCtxSw    ENDP
```

ES:BX→OSTCBHighRdy

OSPrioHighRdy: the priority of the HPT



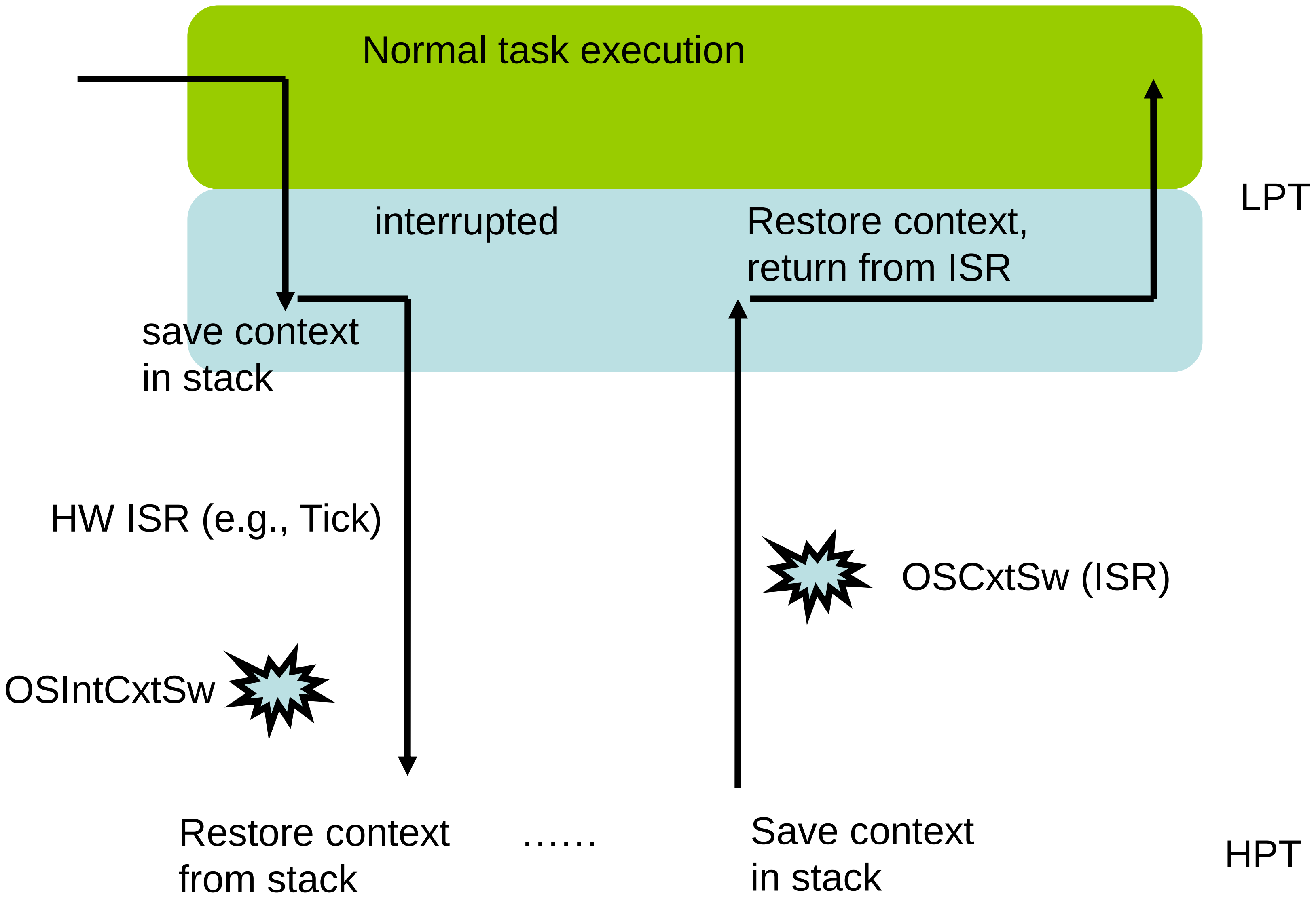
OS_TCB
OSTCBCur
OS_TCB
OSTCBHighRdy
(5)
(6)
80x86 CPU
(Real-Mode)
SS
SP
AX
BX
CX
DX
DS
ES
SI
DI
BP
CS
IP
PSW
OSTCBCur->OSTCBStkPtr
OSTCBHighRdy->OSTCBStkPtr
LOW MEMORY
LOW MEMORY
(4)
PUSH DS
PUSH ES
(3)
PUSHA
(2)
OS_TASK_SW()
(INT 128)
(1)
DS
ES
DI
SI
BP
SP
BX
DX
CX
AX
OFF task
SEG task
PSW
(1)
(2)
(3)
(4)
(7)
(8)
(9)
(10)
(7)
POP DS
POP ES
(8)
POPA
(9)
IRET
(10)
Stack Growth
Stack Growth
HIGH MEMORY
HIGH MEMORY

OSIntCtxSw()

- OSIntCtxSw is called by OSIntExit to perform a context switch when returning from an ISR
  - It is already in ISR
- All registers, including SS:SP, has been saved in the stack of the interrupted task (ISR prologue)

OSIntCtxSw()

```
_OSIntCtxSw PROC   FAR
;
            CALL   FAR PTR _OSTaskSwHook          ; Call user defined task switch hook
;
            MOV    AX, SEG _OSTCBCur              ; Reload DS in case it was altered
            MOV    DS, AX                         ;
;
            MOV    AX, WORD PTR DS:_OSTCBHighRdy+2 ; OSTCBCur = OSTCBHighRdy
            MOV    DX, WORD PTR DS:_OSTCBHighRdy  ;
            MOV    WORD PTR DS:_OSTCBCur+2, AX    ;
            MOV    WORD PTR DS:_OSTCBCur, DX      ;
;
            MOV    AL, BYTE PTR DS:_OSPrioHighRdy ; OSPrioCur = OSPrioHighRdy
            MOV    BYTE PTR DS:_OSPrioCur, AL
;
            LES    BX, DWORD PTR DS:_OSTCBHighRdy ; SS:SP = OSTCBHighRdy->OSTCBStkPtr
            MOV    SS, ES:[BX+2]                  ;
            MOV    SP, ES:[BX]                    ;
;
            POP    DS                             ; Load new task's context
            POP    ES                             ;
            POPA                                  ;
;
            IRET                                  ; Return to new task
;
_OSIntCtxSw ENDP
```


Normal task execution
interrupted
Restore context,
return from ISR
LPT
save context
in stack
HW ISR (e.g., Tick)
OSCxtSw (ISR)
OSIntCxtSw
Restore context
from stack
……
Save context
in stack
HPT

OSTickISR() – the Setup Procedure

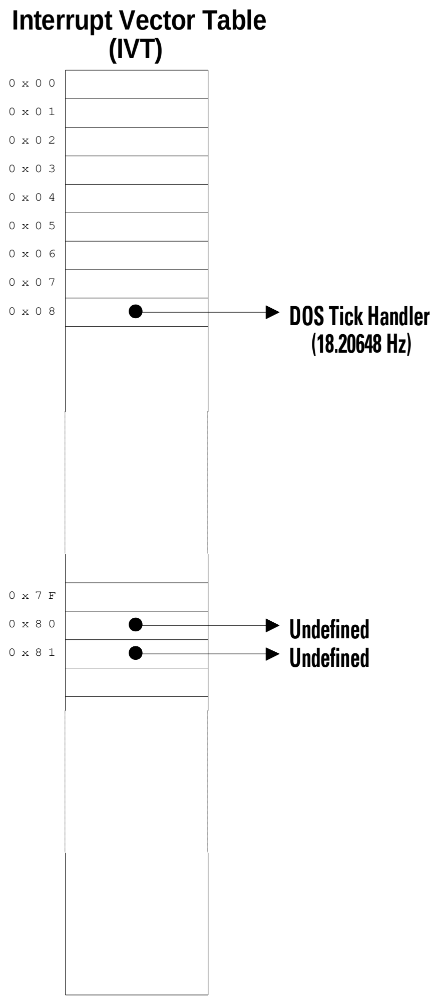
void main():

- Call OSInit
- PC_DOSSaveReturn
- PC_VecSet //install switch vector at vector 0x80
- Create at least one application task
- Call OSStart

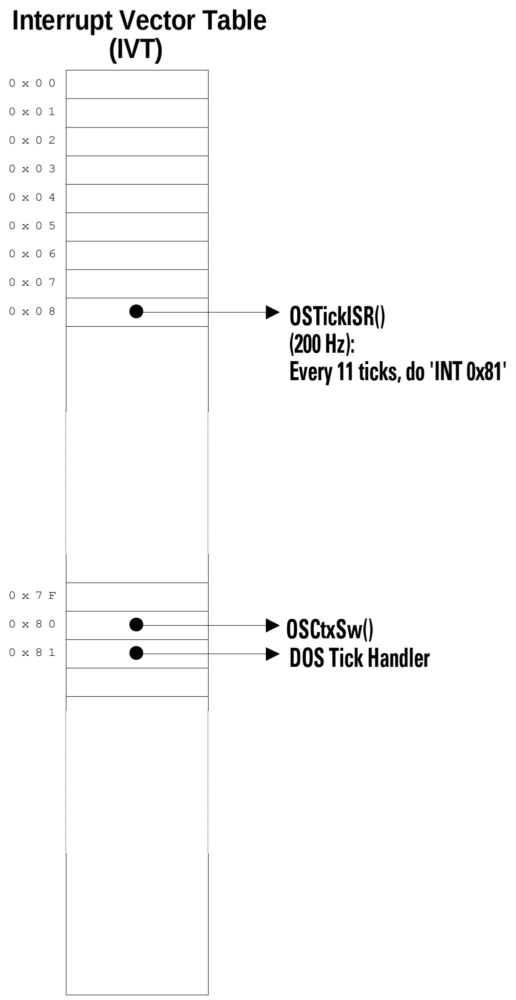
The first task:

- Install OSTickISR
- (Change the tick rate)

Before (DOS only)



After (潛/OS-II installed)


Interrupt Vector Table (IVT)
0x00
0x01
0x02
0x03
0x04
0x05
0x06
0x07
0x08
OSTickISR() (200 Hz): Every 11 ticks, do 'INT 0x81'
0x7F
0x80
OSCtxSw()
0x81
DOS Tick Handler

Pseudocode

```
void OSTickISR(void) {
        save all registers on the current task's stack
        OSIntNesting++;
        if (OSIntNesting ==1) {
                OSTCBCur->OSTCBStkPtr = SS:SP;  //We've just ran into an ISR
        }
        OSTickDosCrt--;
        if (OSTickDosCrt ==0) {
                OSTickDosCtr = 11;
                INT 81H;  //DOS will send the EOI
        } else {
                send EOI to PIC;
        }
        OSTimeTick();
        OSIntExit();
        restore all registers that were save on the current task's stack;
        return from interrupt;
}
```

OSTickISR()

```
_OSTickISR  PROC    FAR
;
            PUSHA                                   ; Save interrupted task's context
            PUSH    ES
            PUSH    DS
;
            MOV     AX, SEG(_OSIntNesting)          ; Reload DS
            MOV     DS, AX
            INC     BYTE PTR DS:_OSIntNesting       ; Notify uC/OS-II of ISR
;
            CMP     BYTE PTR DS:_OSIntNesting, 1    ; if (OSIntNesting == 1)
            JNE     SHORT _OSTickISR1
            MOV     AX, SEG(_OSTCBCur)              ;     Reload DS
            MOV     DS, AX
            LES     BX, DWORD PTR DS:_OSTCBCur      ;     OSTCBCur->OSTCBStkPtr = SS:SP
            MOV     ES:[BX+2], SS                   ;
            MOV     ES:[BX+0], SP                   ;
;
```

```
save all registers on the current task's stack
OSIntNesting++;
if (OSIntNesting ==1) {
        OSTCBCur->OSTCBStkPtr = SS:SP;
}
```

OSIntExit: check if there is a HPT becomes ready. May potentially trigger a context switch

OSTickISR()

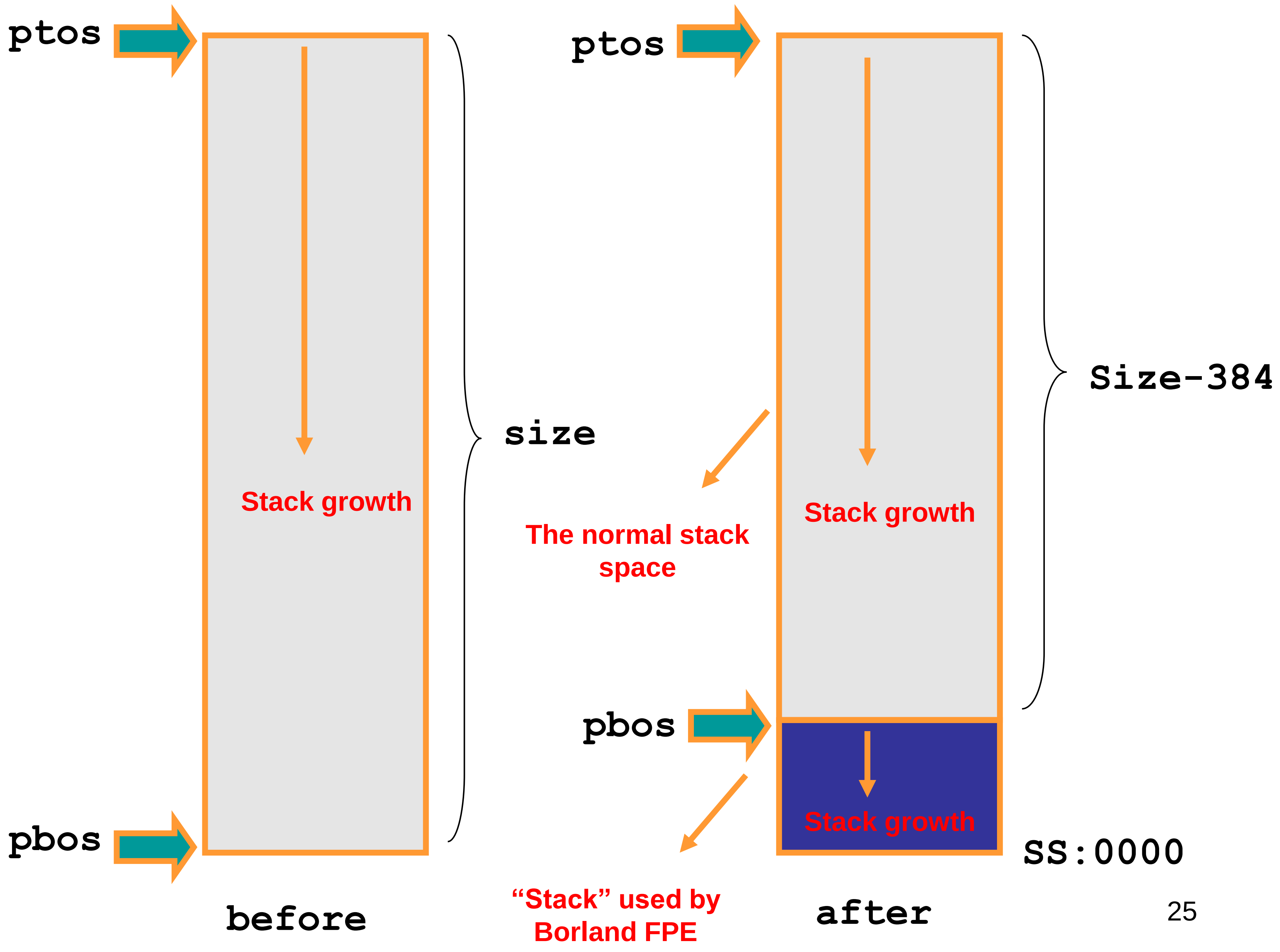
OSTimeTick: to decrement the wait timer and put ‘1’ in the bitmap if it becomes ready

```
_OSTickISR1:
            MOV     AX, SEG(_OSTickDOSCtr)            ; Reload DS
            MOV     DS, AX
            DEC     BYTE PTR DS:_OSTickDOSCtr
            CMP     BYTE PTR DS:_OSTickDOSCtr, 0
            JNE     SHORT _OSTickISR2                 ; Every 11 ticks (~199.99 Hz), chain into DOS
;
            MOV     BYTE PTR DS:_OSTickDOSCtr, 11
            INT     081H                              ; Chain into DOS's tick ISR
            JMP     SHORT _OSTickISR3

_OSTickISR2:
            MOV     AL, 20H                           ; Move EOI code into AL.
            MOV     DX, 20H                           ; Address of 8259 PIC in DX.
            OUT     DX, AL                            ; Send EOI to PIC if not processing DOS timer.
;
_OSTickISR3:
            CALL    FAR PTR _OSTimeTick
;
            CALL    FAR PTR _OSIntExit
;
            POP     DS
            POP     ES
            POPA
;
            IRET
;
_OSTickISR  ENDP
;
            END
```

```
if (OSTickDosCrt ==0) {
          OSTickDosCtr = 11;
          INT 81H;
} else {
          send EOI to PIC;
}
OSTimeTick();
OSIntExit();
```

Using OSTaskStkInit_FPE_x86()

```
OS_STK Task1Stk[1000];
OS_STK Task2Stk[1000];

void main(void) {
        OS_STK *ptos;
        OS_STK *pbos;
        OS_Init();

        ptos = &Task1Stk[999];
        pbos = &Task1st[0];
        size = 1000;

        OSTaskStkInit_FPE_x86(&ptos, &pbos, &size);
        OSTaskCreate(Task1, null, pbos, 10);

        ptos = &Task2Stk[999];
        pbos = &Task2st[0];
        size = 1000;

        OSTaskStkInit_FPE_x86(&ptos, &pbos, &size);
        OSTaskCreate(Task2, null, pbos,
                11, 11, pbos, size, null, OSTask_OPT_SAVE_FP);

        OSStart();
}
```


ptos
pbos
Stack growth
size
before
ptos
Size-384
The normal stack space
Stack growth
pbos
Stack growth
SS:0000
“Stack” used by Borland FPE
after

```c
void  OSTaskStkInit_FPE_x86 (OS_STK **pptos, OS_STK **ppbos, INT32U *psize)
{
    /* 'Linear' version of top-of-stack    address   */
    INT32U   lin_tos;
    /* 'Linear' version of bottom-of-stack address   */
    INT32U   lin_bos;
    INT16U   seg;
    INT16U   off;
    INT32U   bytes;
```

Remarks:
FP_OFF is a macro that can get or set the offset of the far pointer *p.

FP_SEG is a macro that gets or sets the segment value of the far pointer *p.

MK_FP is a macro that makes a far pointer from its component segment (seg) and offset (ofs) parts.

```c
    /* Decompose top-of-stack pointer into seg:off   */
    seg      = FP_SEG(*pptos);
    off      = FP_OFF(*pptos);
    /* Convert seg:off to linear address             */
    lin_tos  = ((INT32U)seg << 4) + (INT32U)off;
    /* Determine how many bytes for the stack        */
    bytes    = *psize * sizeof(OS_STK);
    /* Ensure paragraph alignment for BOS            */
    lin_bos  = (lin_tos - bytes + 15) & 0xFFFFFFF0L;

    /* Get new 'normalized' segment                  */
    seg      = (INT16U)(lin_bos >> 4);
    /* Create 'normalized' BOS pointer               */
    *ppbos   = (OS_STK *)MK_FP(seg, 0x0000);
    /* Copy FP emulation memory to task's stack      */
    memcpy(*ppbos, MK_FP(_SS, 0), 384);
    /* Loose 16 bytes because of alignment           */
    bytes    = bytes - 16;
    /* Determine new top-of-stack                    */
    *pptos   = (OS_STK *)MK_FP(seg, (INT16U)bytes);
    /* Determine new bottom-of-stack                 */
    *ppbos   = (OS_STK *)MK_FP(seg, 384);
    bytes    = bytes - 384;
    /* Determine new stack size                      */
    *psize   = bytes / sizeof(OS_STK);
}
```

Summary

- A port includes
  - Stack initialization
  - Context switch (task-level and interrupt level)
  - Timer ISR
- The bootloader is hardware-specific
- Refer to micrium.com for information of other ports
  - ARM, microblaze, NIOS-II…