Registers

General-Purpose Registers

Program Status and Control Register

Segment Registers
<table>
<thead>
<tr>
<th>31</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
<th>16-bit</th>
<th>32-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AH</td>
<td>AL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AX</td>
<td>EAX</td>
</tr>
<tr>
<td>BH</td>
<td>BL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BX</td>
<td>EBX</td>
</tr>
<tr>
<td>CH</td>
<td>CL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CX</td>
<td>ECX</td>
</tr>
<tr>
<td>DH</td>
<td>DL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DX</td>
<td>EDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BP</td>
<td>EBP</td>
</tr>
<tr>
<td>SI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SI</td>
<td>ESI</td>
</tr>
<tr>
<td>DI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DI</td>
<td>EDI</td>
</tr>
<tr>
<td>SP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SP</td>
<td>ESP</td>
</tr>
</tbody>
</table>
General registers

- EAX — Accumulator for operands and results data
- EBX — Pointer to data in the DS segment
- ECX — Counter for string and loop operations
- EDX — I/O pointer
- ESI — Pointer to data in the segment pointed to by the DS register; source pointer for string operations
- EDI — Pointer to data (or destination) in the segment pointed to by the ES register; destination pointer for string operations
- ESP — Stack pointer (in the SS segment)
- EBP — Pointer to data on the stack (in the SS segment)
Address translation
Segmentation
Logical address

- Segment selector (16 bit) + offset (32 bit)
**Segment descriptors**

- **Base address**
  - 0 – 4 GB
- **Limit (size)**
  - 0 – 4 GB
- **Access rights**
  - Executable, readable, writable
  - Privilege level (0 - 3)

---

<table>
<thead>
<tr>
<th>Access</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Address</td>
<td></td>
</tr>
</tbody>
</table>
## Segment descriptors

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base 31:24</td>
<td>Base address</td>
</tr>
<tr>
<td>G</td>
<td>Granularity</td>
</tr>
<tr>
<td>D/B</td>
<td>Default operation size (0 = 16-bit segment; 1 = 32-bit segment)</td>
</tr>
<tr>
<td>AVL</td>
<td>Available for use by system software</td>
</tr>
<tr>
<td>Seg. Limit 19:16</td>
<td>Segment limit</td>
</tr>
<tr>
<td>P</td>
<td>Segment present</td>
</tr>
<tr>
<td>DPL</td>
<td>Descriptor privilege level</td>
</tr>
<tr>
<td>S</td>
<td>Descriptor type (0 = system; 1 = code or data)</td>
</tr>
<tr>
<td>Type</td>
<td>Segment type</td>
</tr>
<tr>
<td>Base 23:16</td>
<td>Base address</td>
</tr>
<tr>
<td>Base Address 15:00</td>
<td>Base address</td>
</tr>
<tr>
<td>Segment Limit 15:00</td>
<td>Segment limit</td>
</tr>
</tbody>
</table>

* L — 64-bit code segment (IA-32e mode only)

---

L — 64-bit code segment (IA-32e mode only)
AVL — Available for use by system software
BASE — Segment base address
D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
DPL — Descriptor privilege level
G — Granularity
LIMIT — Segment Limit
P — Segment present
S — Descriptor type (0 = system; 1 = code or data)
TYPE — Segment type
Segment registers

• Hold 16 bit segment selectors
  • Pointers into a special table
  • Global or local descriptor table
• Segments are associated with one of three types of storage
  • Code
  • Data
  • Stack
Code segment

- Code
  - CS register
  - EIP is an offset inside the segment stored in CS
- Can only be changed with
  - procedure calls,
  - interrupt handling, or
  - task switching
Data segment

• Data
  • DS, ES, FS, GS
  • 4 possible data segments can be used at the same time
Stack segment

- Stack
  - SS
- Can be loaded explicitly
  - OS can set up multiple stacks
  - Of course, only one is accessible at a time
Flat model

- Hide segmentation mechanism
- But allows access to nonexistent memory
Protected flat

Segment Descriptors

Access | Limit
---|---
Base Address

Linear Address Space (or Physical Memory)

- Code
- Not Present
- Memory I/O
- Data and Stack

Segment Registers
- CS
- ES
- SS
- DS
- FS
- GS
Stack
Stack

- **SS**
  - Specifies stack segment

- **ESP**
  - Contains the address of the data that would be removed from the stack

- **PUSH/POP**
  - Insert/remove data on the stack
  - Subtract/add 4 to ESP
Example: PUSH

Stack Growth

Before Pushing Doubleword

31 0

n
n - 4
n - 8

After Pushing Doubleword

31 0

Doubleword Value

ESP

ESP
Example: POP

Before Popping Doubleword

<table>
<thead>
<tr>
<th>31</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td></td>
</tr>
<tr>
<td>n - 4</td>
<td></td>
</tr>
<tr>
<td>n - 8 Doubleword Value</td>
<td></td>
</tr>
</tbody>
</table>

After Popping Doubleword

<table>
<thead>
<tr>
<th>31</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP</td>
<td></td>
</tr>
</tbody>
</table>

Stack Growth
Setting up stack

- Create a stack descriptor
  - Base, limit, access rights
- Load stack selector into SS register
  - MOV, POP, or LSS
- Load the stack pointer into ESP
  - MOV, POP, or LSS
Call/return

- Stack is used to implement function invocations
- CALL
  - Makes an unconditional jump to a subprogram and pushes the address of the next instruction on the stack
- RET
  - Pops off an address and jumps to that address
Stack bottom pointer

Initially parameter is
- \([ESP + 4]\)

Later as the function pushes things on the stack it changes, e.g.
- \([ESP + 8]\)
- Use dedicated register \texttt{EBP}\n
\begin{table}
\begin{tabular}{|c|c|}
\hline
ESP & Parameter \\hline
ESP + 4 & Return address \\hline
ESP + 8 & Parameter \\hline
ESP + 4 & Return address \\hline
ESP & subprogram data \\hline
\end{tabular}
\end{table}
Prologue/epilogue

subprogram_label:
    push ebp            ; save original EBP value on stack
    mov ebp, esp       ; new EBP = ESP
; subprogram code
    pop ebp            ; restore original EBP value
    ret

• Example invocation

    push dword 1       ; pass 1 as parameter
    call fun
    add esp, 4        ; remove parameter from stack
Calling conventions

- Goal: reentrant programs
  - Conventions differ from compiler, optimizations, etc.
- Call/return are used for function invocations
- Parameters passed on the stack
  - Pushed onto the stack before the CALL instruction
Local variables

• Stored right after the saved EBP value in the stack
• Allocated by subtracting the number of bytes required from ESP

subprogram_label:
    push ebp                ; save original EBP value on stack
    mov ebp, esp           ; new EBP = ESP
    sub esp, LOCAL_BYTES ; = # bytes needed by locals

; subprogram code
    mov esp, ebp           ; deallocate locals
    pop ebp                ; restore original EBP value
    ret
Parameter passing

- Registers
- On the stack
- Through memory
  - Pass a pointer to the parameter list in one of the registers
Saving state

- Processor doesn't save registers
  - General purpose, segment, flags
- Calling convention is needed
  - Agreement on what gets saved by a callee and caller
Interrupts
INT X

• Transfers control to the handler number X in a special table
  • Interrupt descriptor table
• IDT can be anywhere in the linear address space
  • Located with the IDTR register
Interrupt descriptor

Interrupt Gate

Offset 31..16

Segment Selector

P DPL 0 D 1 1 0 0 0 0

4

0

Offset 15..0
Interrupt descriptor

Interrupt Gate

Offset 31..16

Segment Selector

Offset 15..0
Interrupt descriptor

Interrupt Gate

Offset 31..16

Segment Selector

Offset 15..0
Stack Usage with No Privilege-Level Change

Interrupted Procedure’s and Handler’s Stack

- ESP Before Transfer to Handler
- ESP After Transfer to Handler

EFLAGS
CS
EIP
Error Code
Stack Usage with Privilege-Level Change

Interrupted Procedure’s Stack

Handler’s Stack

ESP Before Transfer to Handler

SS
ESP
EFLAGS
CS
EIP
Error Code

ESP After Transfer to Handler
1. check that CPL <= DPL in the descriptor (but only if INT instruction).

2. save ESP and SS in a CPU-internal register (but only if target segment selector's PL < CPL).

3. load SS and ESP from TSS ("")

4. push user SS ("")

5. push user ESP ("")

6. push user EFLAGS

7. push user CS

8. push user EIP

9. clear some EFLAGS bits

10. set CS and EIP from IDT descriptor's segment selector and offset