The State Machine

The CPU program:
while (TRUE)
{
  fetch the instruction
  execute the instruction
  (update the instruction pointer)
}

What instructions do:
• Change the state of CPU
• ALU operation
• Jumps
• Read data from memory
• Write data to memory

X86 assembly quick tutorial
(real mode)
Wei Dong
CPU State: Register Set

General-purpose registers 16bit  32bit

<table>
<thead>
<tr>
<th>AH</th>
<th>AL</th>
<th>AX</th>
<th>EAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>BH</td>
<td>BL</td>
<td>BX</td>
<td>EAX</td>
</tr>
<tr>
<td>CH</td>
<td>CL</td>
<td>CX</td>
<td>ECX</td>
</tr>
<tr>
<td>DH</td>
<td>DL</td>
<td>DX</td>
<td>EDX</td>
</tr>
<tr>
<td>BP</td>
<td></td>
<td>EBP</td>
<td></td>
</tr>
<tr>
<td>SI</td>
<td></td>
<td>ESI</td>
<td></td>
</tr>
<tr>
<td>DI</td>
<td></td>
<td>EDI</td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td></td>
<td>ESP</td>
<td></td>
</tr>
</tbody>
</table>

Segment registers (16bit)

<table>
<thead>
<tr>
<th>CS</th>
<th>DS</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES</td>
<td>FS</td>
<td>GS</td>
</tr>
</tbody>
</table>

Address the register by: %ax, %ebx, etc

A little bit on EFLAGS

Function of EFLAGS:

• Control the behavior of CPU
• Save the status of last instruction

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CF: carry flag</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ZF: zero flag</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>SF: sign flag</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>IF: interrupt</td>
<td>sti; cli;</td>
</tr>
<tr>
<td>10</td>
<td>DF: direction</td>
<td>std; cld;</td>
</tr>
<tr>
<td>11</td>
<td>OF: overflow</td>
<td></td>
</tr>
</tbody>
</table>
Memory Addressing

Format:

\[
\text{segment:displacement(base, index)}
\]

Offset = Base + Index + Displacement

Address = (Segment \(\ll 4\)) + Offset

Displacement: constant
Base: \%bx, \%bp
Index: \%si, \%di
Segment: \%cs, \%ds, \%ss, \%es

Memory Addressing (data)

\[
\text{segment:displacement(base, index)}
\]

- The components are all optional
- Default segment:
  - \%bp: \%ss
  - \%bx, \%si, \%di: \%ds
- Examples
  
  \[
  \begin{align*}
  100 & = \%ds:(%si) \\
  (%bp) & = \%ss:(%bp) \\
  (%bx,%si) & = \%ds:(%bx,%si) \\
  -10(%bp) & = \%ss:-10(%bp) \\
  \%ds:-10(%bx, %si) & 
  \end{align*}
  \]
Instructions: arithmetic & logic

- add/sub{l,w,b} source, dest
- inc/dec/neg{l,w,b} dest
- cmp{l,w,b} source, dest
- and/or/xor{l,w,b} source, dest

...  

Restrictions

- No more than one memory operand

Instructions: Data Transfer

- mov{lwb} source, dest
- xchg{lwb} source, dest
  - Segment registers can only appear with registers
- movsb
  - movb %ds:(%si) %es:(%di)
  - %si ← %si + inc  
    - If DF = 0 then inc = 1
    - else inc = -1
  - %di ← %di + inc
  - %cx ← %cx – 1
  - Often used with %cx to move a number of bytes
Example

- Move 0x200 bytes from 0x0100:0x0000 to 0x0080:0x0000
  movw $0x0100, %ax
  movw %ax, %ds /* setup %ds */
  movw $0x0080, %ax
  movw %ax, %es /* setup %es */
  movw $0, %ax
  movw %ax, %si /* setup %si */
  movw %ax, %di /* setup %di */
  movw $0x200, %cx
  cld /* setup direction flag */
  repeat:
    movsb
    cmp $0, %cx
    jnz repeat

Example (cont.)

- Move 0x200 bytes from 0x0100:0x0000 to 0x0080:0x0000
  movw $0x0100, %ax
  movw %ax, %ds /* setup %ds */
  movw $0x0080, %ax
  movw %ax, %es /* setup %es */
  movw $0, %ax
  movw %ax, %si /* setup %si */
  movw %ax, %di /* setup %di */
  movw $0x200, %cx
  cld /* setup direction flag */
  rep movsb
Instructions: stack access

- pushw source
  - %sp ← %sp – 2
  - %ss:(%sp) ← source
- popw dest
  - dest ← %ss:(%sp), dest
  - %sp ← %sp + 2
- Setup up the stack before you actually use it

Instructions: unconditional jump

- jmp label
  - %ip ← label
- ljmp NEW_CS, offset
  - %ip ← label; %cs ← NEW_CS
- call label
  - push %ip + ? (address of call instruction)
  - %ip ← label
- ret
  - pop %ip
- Also lcall and lret
Instructions: conditional jump

- j* label: jump to label if flag * is 1
- jn* label: jump to label if flag * is 0
- *: bits of %eflags
  Examples: js, jz, jc, jns, jnz, jnc, ...

BIOS Service

- Use BIOS service through interruption
  - Store the parameters to the registers
  - Call the interruption
- int INT_NUM
Example: BIOS INT 0x13 Function 2

- ah = 2
- al = number of sectors to read
- ch = cylinder number bits 0-8
- cl, bits 6&7 = cylinder number bits 8-9.
- bits 0-5 = starting sector number, 1 to 63
- dh = starting head number, 0 to 255
- dl = drive number
- es:bx = pointer where to place information read from diskette

Returns:
- ah = return status (0 if successful)
- carry = 0 successful, = 1 if error occurred

Note for project 1

- In our project, the bootloader is working in real mode (16 bits).
- Bootloader code is loaded by BIOS, so it did not have %ds, %ss, %sp setup properly when it is loaded.
- In bootloader, all the code and data share the same 512 bytes. So data will have the same segment as code.