REMINDER: we use pointers so can pass address since can’t pass values back outside of the function.

void swap(int *xp, int *yp) {
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
} //codeswap.c

swap:

pushl  %ebp
movl   %esp, %ebp
pushl  %ebx

movl   8(%ebp), %edx
movl   12(%ebp), %eax
movl   (%edx), %ecx
movl   (%eax), %ebx
movl   %ebx, (%edx)
movl   %ecx, (%eax)

popl   %ebx
popl   %ebp
ret
## Understanding Swap

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td>456</td>
</tr>
<tr>
<td>%edx</td>
<td>0x124</td>
</tr>
<tr>
<td>%ecx</td>
<td>0x120</td>
</tr>
<tr>
<td>%ebx</td>
<td>123</td>
</tr>
<tr>
<td>%esi</td>
<td></td>
</tr>
<tr>
<td>%edi</td>
<td></td>
</tr>
<tr>
<td>%esp</td>
<td>0x104</td>
</tr>
<tr>
<td>%ebp</td>
<td>0x100</td>
</tr>
</tbody>
</table>

### Move Instructions

1. Move 0x124 to %edx
2. Move 0x120 to %ecx
3. Move 123 to %ebx
4. Move 456 to %eax
5. Move 456 to M[0x124]
6. Move 123 to M[0x120]

```
pushl %ebp
movl %esp, %ebp
pushl %ebx

movl 8(%ebp), %edx  # edx = xp
movl 12(%ebp), %ecx # ecx = yp
movl (%edx), %ebx   # ebx = *xp (t0)
movl (%ecx), %eax   # eax = *yp (t1)
movl %eax, (%edx)   # *xp = t1
movl %ebx, (%ecx)   # *yp = t0
```

```
popl %ebx
popl %ebp
ret
```
## Instruction formats for swap

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Op Code</th>
<th>ModR/M</th>
<th>SIB</th>
<th>Displacement</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>&lt;swap&gt;:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0: 55</td>
<td>push %ebp</td>
<td>05</td>
<td>55</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1: 89 e5</td>
<td>mov %esp,%ebp</td>
<td>089</td>
<td>11 100 101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3: 53</td>
<td>push %ebx</td>
<td>053</td>
<td>53</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4: 8b 55 08</td>
<td>mov 0x8(%ebp),%edx</td>
<td>08b</td>
<td>01 010 101</td>
<td>0000 1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7: 8b 45 0c</td>
<td>mov 0xc(%ebp),%eax</td>
<td>08b</td>
<td>01 000 101</td>
<td>0000 1100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a: 8b 0a</td>
<td>mov (%edx),%ecx</td>
<td>08b</td>
<td>00 001 010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c: 8b 18</td>
<td>mov (%eax),%ebx</td>
<td>08b</td>
<td>00 011 000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e: 89 1a</td>
<td>mov %ebx,(%edx)</td>
<td>089</td>
<td>00 011 010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10: 89 08</td>
<td>mov %ecx,(%eax)</td>
<td>089</td>
<td>00 001 000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12: 5b</td>
<td>pop %ebx</td>
<td>05b</td>
<td>5b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13: 5d</td>
<td>pop %ebp</td>
<td>05d</td>
<td>5d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14: c3</td>
<td>ret</td>
<td>0c3</td>
<td>c3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PUSH pg 701; MOV pg 479; POP pg 637; RET pg 28
Instruction Format

- All IA-32 instruction encodings are subsets of the general instruction format shown below, in the given order.

Instructions consist of:
- optional instruction prefixes (in any order)
- 1-3 opcode bytes – determines the action of the statement
- an addressing-form specifier (if required) consisting of:
  - the ModR/M byte - addressing modes register/memory
  - sometimes the SIB (Scale-Index-Base) byte
  - a displacement (if required)
  - an immediate data field (if required).
ModR/M

<table>
<thead>
<tr>
<th>Mod</th>
<th>Reg #</th>
<th>R/M</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 bits</td>
<td>3 bits</td>
<td>3 bits</td>
</tr>
</tbody>
</table>

- **Mod=00**,
  - First operand a register, specified by Reg #
  - Second operand in memory; address stored in a register numbered by R/M.
    - That is, Memory[Reg[R/M]]
  - Exceptions:
    - R/M=100 (SP): SIB needed
    - R/M=101 (BP): disp32 needed

- **Mod=01**, same as Mod 00 with 8-bit displacement.
  - Second operand: Memory[disp8+Reg[R/M]].
  - Exception: SIB needed when R/M=100

- **Mod=10**, same as Mod 01 with 32-bit displacement

- **Mod=11**
  - Second operand is also a register, numbered by R/M.
  - Do not confuse displacement width with data width.
  - Data width is specified by the opcode.
  - For example, the use of disp8 does not imply 8-bit data.

For some opcodes, the reg# is used as an extension of the opcode.
SIB displacement and immediate

<table>
<thead>
<tr>
<th>Scale</th>
<th>Index</th>
<th>Base</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 bits</td>
<td>3 bits</td>
<td>3 bits</td>
</tr>
</tbody>
</table>

- **SIB**
  - Specify how a memory address is calculated
  - Address = Reg[base] + Reg[Index] * 2<sup>scale</sup>
  - Exceptions:
    - SP cannot be an index, and
    - BP cannot be a base

- **Displacement**
  - Can immediately follow ModR/M byte
  - 1, 2, or 4 bytes

- **Immediate**
  - Immediate operand value always follows any displacement bytes
  - 1, 2 or 4 bytes