Principles of Computers
12th Lecture

Pavel Ježek, Ph.D.
pavel.jezek@d3s.mff.cuni.cz
# Examples of CPU Architectures

<table>
<thead>
<tr>
<th>CPU arch.</th>
<th>CPU name</th>
<th>Logical address width</th>
<th>Current instruction register(s)</th>
<th>Physical address width</th>
<th>Special mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit</td>
<td>MOS 6502</td>
<td><strong>16-bit</strong></td>
<td>PC</td>
<td>16-bit (64 kB)</td>
<td></td>
</tr>
<tr>
<td>x86-16</td>
<td>Intel 8088</td>
<td>16 + 16 bit</td>
<td>CS:IP</td>
<td>20-bit (1 MB)</td>
<td></td>
</tr>
<tr>
<td>x86-16</td>
<td>Intel 8086</td>
<td>16 + 16 bit</td>
<td>CS:IP</td>
<td>24-bit (16 MB)</td>
<td>protected 16 (+ real) mode</td>
</tr>
<tr>
<td>x86-32</td>
<td>Intel 80286</td>
<td>16 + 16 bit</td>
<td>CS:IP</td>
<td>32-bit (4 GB)</td>
<td>protected 32 mode</td>
</tr>
<tr>
<td>x86-32</td>
<td>Intel 80386</td>
<td><strong>32-bit</strong></td>
<td>EIP</td>
<td>36-bit (64 GB)</td>
<td>PAE</td>
</tr>
<tr>
<td>IA-32</td>
<td>Intel Pentium Pro</td>
<td>32-bit</td>
<td>EIP</td>
<td>40-bit (1 TB)</td>
<td>long mode</td>
</tr>
<tr>
<td>AM64</td>
<td>AMD Opteron (Intel Pentium 4)</td>
<td>64-bit</td>
<td>RIP</td>
<td>40-bit (1 TB)</td>
<td>long mode</td>
</tr>
<tr>
<td>EM64T</td>
<td>2015 current (e.g. Core i7)</td>
<td>64-bit</td>
<td>RIP</td>
<td>AMD: 48b → 256 TB Intel: 46b → 64 TB</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>32-bit ARM</th>
<th>64-bit ARM</th>
<th>32b MIPS</th>
<th>MIPS64</th>
<th>PowerPC (PPC)</th>
<th>32b Motorola 68000 (68k)</th>
</tr>
</thead>
</table>

### Notes:
- **ARM**: 32-bit and 64-bit architectures.
- **MIPS**: 32-bit and 64-bit architectures.
- **MIPS64**: 64-bit architectures.
- **PowerPC (PPC)**: 32-bit and 64-bit architectures.
- **32b Motorola 68000 (68k)**: 16-bit architecture.
# Basic Instructions (6502 vs. x86)

<table>
<thead>
<tr>
<th>6502 machine code</th>
<th>Intel x86 (IA-32) machine code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>← Offset from instruction’s start (base) address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>← Actual bytes of instruction’s machine code</td>
</tr>
</tbody>
</table>
## Basic Instructions (6502 vs. x86)

<table>
<thead>
<tr>
<th>6502 machine code</th>
<th>Intel x86 (IA-32) machine code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $EA</td>
<td>0 $90</td>
<td>Offset from instruction’s start (base) address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Actual bytes of instruction’s machine code</td>
</tr>
</tbody>
</table>

*No operation (just do nothing)*
# Basic Instructions (6502 vs. x86)

<table>
<thead>
<tr>
<th>6502 machine code</th>
<th>Intel x86 (IA-32) machine code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $EA</td>
<td>0 $90</td>
<td>← Offset from instruction’s start (base) address</td>
</tr>
<tr>
<td>PC := PC + 1</td>
<td>EIP := EIP + 1</td>
<td>← Actual bytes of instruction’s machine code</td>
</tr>
</tbody>
</table>

*No operation (just do nothing and continue to next instruction)*
## Basic Instructions (6502 vs. x86)

<table>
<thead>
<tr>
<th>6502 machine code</th>
<th>Intel x86 (IA-32) machine code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $EA</td>
<td>0 $90</td>
<td>← Offset from instruction’s start (base) address&lt;br&gt;← Actual bytes of instruction’s machine code&lt;br&gt;&lt;i&gt;No operation (just do nothing and continue to next instruction)&lt;/i&gt;</td>
</tr>
<tr>
<td>PC := PC + 1</td>
<td>EIP := EIP + 1</td>
<td></td>
</tr>
<tr>
<td>$4C xx_0 xx_1</td>
<td>$E9 xx_0 xx_1 xx_2 xx_3</td>
<td>← Offset from instruction’s start (base) address&lt;br&gt;← Actual bytes of instruction’s machine code&lt;br&gt;&lt;i&gt;Direct jump to target address x&lt;/i&gt;</td>
</tr>
<tr>
<td>PC := $xx_3xx_2xx_1xx_0</td>
<td>EIP := $xx_3xx_2xx_1xx_0</td>
<td></td>
</tr>
</tbody>
</table>

**Jump/branch instruction**
## Basic Instructions (6502 vs. x86)

<table>
<thead>
<tr>
<th>6502 machine code</th>
<th>Intel x86 (IA-32) machine code</th>
<th>Comment</th>
</tr>
</thead>
</table>
| 0 $EA             | 0 $90                           | ← Offset from instruction’s start (base) address  
                    |                                 | ← Actual bytes of instruction’s machine code  
                    |                                 | No operation (just do nothing and continue to next instruction) |
| PC := PC + 1      | EIP := EIP + 1                  |         |
|                   |                                 | 16-bit PC → 2 byte argument  
                   |                                 | 32-bit EIP → 4 byte argument |
| 0 $4C xx₀ xx₁     | 0 $E9 xx₀ xx₁ xx₂ xx₃           | ← Offset from instruction’s start (base) address  
                    |                                 | ← Actual bytes of instruction’s machine code  
                    |                                 | Direct jump to target address x |
| PC := $xx₁xx₀     | EIP := $xx₃xx₂xx₁xx₀            |         |

**Jump/branch instruction**
## Basic Instructions (6502 vs. x86)

<table>
<thead>
<tr>
<th>6502 machine code</th>
<th>Intel x86 (IA-32) machine code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $EA</td>
<td>0 $90</td>
<td>← Offset from instruction’s start (base) address</td>
</tr>
<tr>
<td></td>
<td>EIP := EIP + 1</td>
<td>← Actual bytes of instruction’s machine code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No operation (just do nothing and continue to next instruction)</td>
</tr>
<tr>
<td>0 $4C xx&lt;sub&gt;0&lt;/sub&gt; xx&lt;sub&gt;1&lt;/sub&gt;</td>
<td>0 $E9 xx&lt;sub&gt;0&lt;/sub&gt; xx&lt;sub&gt;1&lt;/sub&gt; xx&lt;sub&gt;2&lt;/sub&gt; xx&lt;sub&gt;3&lt;/sub&gt;</td>
<td>← Offset from instruction’s start (base) address</td>
</tr>
<tr>
<td></td>
<td>PC := $xx&lt;sub&gt;1&lt;/sub&gt;xx&lt;sub&gt;0&lt;/sub&gt;</td>
<td>← Actual bytes of instruction’s machine code</td>
</tr>
<tr>
<td></td>
<td>EIP := $xx&lt;sub&gt;3&lt;/sub&gt;xx&lt;sub&gt;2&lt;/sub&gt;xx&lt;sub&gt;1&lt;/sub&gt;xx&lt;sub&gt;0&lt;/sub&gt;</td>
<td>Direct jump to target address x</td>
</tr>
</tbody>
</table>

**Jump/branch instruction**

6502 is LE CPU arch.

x86 is LE CPU arch.
## Basic Instructions (6502 vs. x86)

<table>
<thead>
<tr>
<th>6502 machine code</th>
<th>Intel x86 (IA-32) machine code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $EA</td>
<td>0 $90</td>
<td>Offset from instruction’s start (base) address</td>
</tr>
<tr>
<td>PC := PC + 1</td>
<td>EIP := EIP + 1</td>
<td>Actual bytes of instruction’s machine code</td>
</tr>
<tr>
<td><strong>6502 assembler:</strong></td>
<td><strong>Intel assembler:</strong></td>
<td><strong>No operation (just do nothing and continue to next instruction)</strong></td>
</tr>
<tr>
<td>NOP</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0 1 2 $4C xx₀ xx₁</td>
<td>0 1 2 3 4 $E9 xx₀ xx₁ xx₂ xx₃</td>
<td>Offset from instruction’s start (base) address</td>
</tr>
<tr>
<td>PC := $xx₁xx₀</td>
<td>EIP := $xx₃xx₂xx₁xx₀</td>
<td>Actual bytes of instruction’s machine code</td>
</tr>
<tr>
<td><strong>6502 assembler:</strong></td>
<td><strong>Intel assembler:</strong></td>
<td><strong>Direct jump to target address x</strong></td>
</tr>
<tr>
<td>JMP $xx₁xx₀</td>
<td>JMP $xx₃xx₂xx₁xx₀h</td>
<td></td>
</tr>
</tbody>
</table>
# Basic Instructions (6502 vs. x86)

<table>
<thead>
<tr>
<th>6502 machine code</th>
<th>Intel x86 (IA-32) machine code</th>
<th>Comment</th>
</tr>
</thead>
</table>
| 0 $EA             | 0 $90                         | $EA ← Offset from instruction’s start (base) address  
|                   |                               | $90 ← Actual bytes of instruction’s machine code  |

*6502 assembler:*  
NOP

*Intel assembler:*  
NOP

| 0 1 2             | 0 1 2 3 4               | $EA ← Offset from instruction’s start (base) address  
| $4C $x_0 $x_1    | $E9 $x_0 $x_1 $x_2 $x_3 | $90 ← Actual bytes of instruction’s machine code  |

*6502 assembler:*  
JMP $xx_1xx_0

*Intel assembler:*  
JMP xx_3xx_2xx_1xx_0h

| 0 1 2             | 0 1 2 3 4 5             | $EA ← Offset from instruction’s start (base) address  
| $6C $x_0 $x_1    | $FF $25 $x_0 $x_1 $x_2 $x_3 | $90 ← Actual bytes of instruction’s machine code  |

*Comment:*  
No operation (just do nothing and continue to next instruction)

Direct jump to target address x
## Basic Instructions (6502 vs. x86)

<table>
<thead>
<tr>
<th>6502 machine code</th>
<th>Intel x86 (IA-32) machine code</th>
<th>Comment</th>
</tr>
</thead>
</table>
| $EA$              | $90$                            | $\leftarrow$ Offset from instruction’s start (base) address  
|                   |                                 | $\leftarrow$ Actual bytes of instruction’s machine code |
| $\text{PC} := \text{PC} + 1$ | $\text{EIP} := \text{EIP} + 1$ | *No operation (just do nothing and continue to next instruction)* |
| $\text{6502 assembler: NOP}$ | $\text{Intel assembler: NOP}$ |         |
| $4C$ $xx_0$ $xx_1$ | $E9$ $xx_0$ $xx_1$ $xx_2$ $xx_3$ | $\leftarrow$ Offset from instruction’s start (base) address  
|                   | $\text{EIP} := \text{EIP} + 1$ | $\leftarrow$ Actual bytes of instruction’s machine code |
| $\text{6502 assembler: JMP }$ $xx_1$ $xx_0$ | $\text{Intel assembler: JMP }$ $xx_3$ $xx_2$ $xx_1$ $xx_0$ $h$ | *Direct jump to target address* $x$ |
| $6C$ $xx_0$ $xx_1$ | $FF$ $25$ $xx_0$ $xx_1$ $xx_2$ $xx_3$ | $\leftarrow$ Offset from instruction’s start (base) address  
|                   | $\text{EIP} := (^\text{longword}(xx_3xx_2xx_1xx_0))$ | $\leftarrow$ Actual bytes of instruction’s machine code |
| $\text{6502 assembler: JMP (}xx_1xx_0\text{)}$ | $\text{Intel assembler: JMP [}xx_3xx_2xx_1xx_0h\text{]}$ | *Indirect jump to target address that is stored at address* $x$ |
# Basic Instructions (6502 vs. x86)

<table>
<thead>
<tr>
<th>6502 machine code</th>
<th>Intel x86 (IA-32) machine code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $EA</td>
<td>0 $90</td>
<td>← Offset from instruction’s start (base) address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>← Actual bytes of instruction’s machine code</td>
</tr>
<tr>
<td>PC := PC + 1</td>
<td>EIP := EIP + 1</td>
<td><strong>No operation (just do nothing and continue to next instruction)</strong></td>
</tr>
<tr>
<td><strong>6502 assembler:</strong></td>
<td><strong>Intel assembler:</strong></td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0 1 2</td>
<td>0 1 2 3 4</td>
<td>← Offset from instruction’s start (base) address</td>
</tr>
<tr>
<td>$4C xx_0 xx_1</td>
<td>$E9 xx_0 xx_1 xx_2 xx_3</td>
<td>← Actual bytes of instruction’s machine code</td>
</tr>
<tr>
<td>PC := $xx_1xx_0</td>
<td>EIP := $xx_3xx_2xx_1xx_0</td>
<td><strong>Absolute direct jump to target address x</strong></td>
</tr>
<tr>
<td><strong>6502 assembler:</strong></td>
<td><strong>Intel assembler:</strong></td>
<td></td>
</tr>
<tr>
<td>JMP $xx_1xx_0</td>
<td>JMP xx_3xx_2xx_1xx_0h</td>
<td></td>
</tr>
<tr>
<td>0 1 2</td>
<td>0 1 2 3 4 5</td>
<td>← Offset from instruction’s start (base) address</td>
</tr>
<tr>
<td>$6C xx_0 xx_1</td>
<td>$FF $25 xx_0 xx_1 xx_2 xx_3</td>
<td>← Actual bytes of instruction’s machine code</td>
</tr>
<tr>
<td>PC := (^word($xx_1xx_0))^</td>
<td>EIP := (^longword($xx_3xx_2xx_1xx_0))^</td>
<td><strong>Indirect jump to target address that is stored at address x</strong></td>
</tr>
<tr>
<td><strong>6502 assembler:</strong></td>
<td><strong>Intel assembler:</strong></td>
<td></td>
</tr>
<tr>
<td>JMP ($xx_1xx_0)</td>
<td>JMP [xx_3xx_2xx_1xx_0h]</td>
<td></td>
</tr>
</tbody>
</table>
## Basic Instructions (6502 vs. x86)

<table>
<thead>
<tr>
<th>6502 machine code</th>
<th>Intel x86 (IA-32) machine code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $EA</td>
<td>0 $90</td>
<td>No operation (just do nothing and continue to next instruction)</td>
</tr>
<tr>
<td>PC := PC + 1</td>
<td>EIP := EIP + 1</td>
<td></td>
</tr>
<tr>
<td>6502 assembler: NOP</td>
<td>Intel assembler: NOP</td>
<td></td>
</tr>
<tr>
<td>0 1 2</td>
<td>0 1 2 3 4</td>
<td></td>
</tr>
<tr>
<td>$4C xx₀ xx₁</td>
<td>$E9 xx₀ xx₁ xx₂ xx₃</td>
<td></td>
</tr>
<tr>
<td>PC := $xx₁xx₀</td>
<td>EIP := EIP + longint($xx₃xx₂xx₁xx₀)</td>
<td></td>
</tr>
<tr>
<td>6502 assembler: JMP $xx₁xx₀</td>
<td>Intel assembler: JMP xx₃xx₂xx₁xx₀h</td>
<td></td>
</tr>
<tr>
<td>0 1 2</td>
<td>0 1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>$6C xx₀ xx₁</td>
<td>$FF $25 xx₀ xx₁ xx₂ xx₃</td>
<td></td>
</tr>
<tr>
<td>PC := (^word($xx₁xx₀))^</td>
<td>EIP := (^longword($xx₃xx₂xx₁xx₀))</td>
<td></td>
</tr>
<tr>
<td>6502 assembler: JMP ($xx₁xx₀)</td>
<td>Intel assembler: JMP [xx₃xx₂xx₁xx₀h]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Basic Instructions (6502 vs. x86)

<table>
<thead>
<tr>
<th>6502 machine code</th>
<th>Intel x86 (IA-32) machine code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $EA</td>
<td>0 $90</td>
<td>$EA</td>
</tr>
<tr>
<td>PC := PC + 1</td>
<td>EIP := EIP + 1</td>
<td>No operation (just do nothing and continue to next instruction)</td>
</tr>
<tr>
<td>6502 assembler:</td>
<td>Intel assembler:</td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0 1 2 $4C xx₀ xx₁</td>
<td>0 1 2 3 4 $E9 xx₀ xx₁ xx₂ xx₃</td>
<td></td>
</tr>
<tr>
<td>PC := $xx₁xx₀</td>
<td>EIP := EIP + 5 + longint($xx₃xx₂xx₁xx₀)</td>
<td></td>
</tr>
<tr>
<td>6502 assembler:</td>
<td>Intel assembler:</td>
<td></td>
</tr>
<tr>
<td>JMP $xx₁xx₀</td>
<td>JMP xx₃xx₂xx₁xx₀h</td>
<td></td>
</tr>
<tr>
<td>0 1 2 $6C xx₀ xx₁</td>
<td>0 1 2 3 4 $FF $25 xx₀ xx₁ xx₂</td>
<td></td>
</tr>
<tr>
<td>PC := (^word($xx₁xx₀))^</td>
<td>EIP := (^longword($xx₃xx₂xx₁xx₀))^</td>
<td></td>
</tr>
<tr>
<td>6502 assembler:</td>
<td>Intel assembler:</td>
<td></td>
</tr>
<tr>
<td>JMP ($xx₁xx₀)</td>
<td>JMP [xx₃xx₂xx₁xx₀h]</td>
<td></td>
</tr>
</tbody>
</table>

- **Absolute** direct jump to target address x
- **Relative** direct jump to target address that is x byte far from current instruction
- **Indirect** jump to target address that is stored at address x
- x86 is using address relative to instruction’s end (i.e. not relative to instruction’s start) → **instruction size** (5 bytes for this $E9 jump instruction) is added to x
## Basic Instructions (6502 vs. x86)

<table>
<thead>
<tr>
<th>6502 machine code</th>
<th>Intel x86 (IA-32) machine code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $EA</td>
<td>0 $90</td>
<td></td>
</tr>
<tr>
<td>$EA</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC := PC + 1</td>
<td>EIP := EIP + 1</td>
<td></td>
</tr>
<tr>
<td>0 1 2</td>
<td>0 1 2</td>
<td></td>
</tr>
<tr>
<td>$4C xx_0 xx_1</td>
<td>$E9 xx_0 xx_1 xx_2 xx_3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC := $xx_1xx_0</td>
<td>EIP := EIP + 5 + longint($xx_3xx_2xx_1xx_0)</td>
<td></td>
</tr>
</tbody>
</table>

**6502 assembler:**
- NOP

**Intel assembler:**
- NOP

- **Absolute direct jump to target address**
- **Relative direct jump to target address**
- **Indirect jump to target address**

---

Single instruction unbreakable infinite cycle doing nothing :-)  

*in Intel assembler:*  
JMP -5  
=  
*in Intel assembler:*  
JMP FFFFFFFFBh  
=  
*in x86 machine code:*  
E9 FB FF FF FF

---

x86 is using address relative to instruction’s end (i.e. not relative to instruction’s start) → **instruction size** (5 bytes for this $E9 jump instruction) is added to x
## Basic Instructions (6502 vs. x86)

<table>
<thead>
<tr>
<th>6502 machine code</th>
<th>Intel x86 (IA-32) machine code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $EA</td>
<td>0 $90</td>
<td>No operation (just do nothing and continue to next instruction)</td>
</tr>
<tr>
<td>PC := PC + 1</td>
<td>EIP := EIP + 1</td>
<td></td>
</tr>
<tr>
<td><em>6502 assembler:</em></td>
<td><em>Intel assembler:</em></td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>0 1 2</td>
<td>0 1 2 3 4</td>
<td>Absolute direct jump to target address x</td>
</tr>
<tr>
<td>$4C xx₀ xx₁</td>
<td>$E9 xx₀ xx₁ xx₂ xx₃</td>
<td>Relative direct jump to target address that is x byte far from current instruction</td>
</tr>
<tr>
<td>PC := $xx₁xx₀</td>
<td>EIP := EIP + 5 + longint($xx₃xx₂xx₁xx₀)</td>
<td></td>
</tr>
<tr>
<td><em>6502 assembler:</em></td>
<td><em>Intel assembler:</em></td>
<td></td>
</tr>
<tr>
<td>JMP $xx₁xx₀</td>
<td>JMP xx₃xx₂xx₁xx₀h</td>
<td></td>
</tr>
<tr>
<td>0 1 2</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>$6C xx₀ xx₁</td>
<td>$EB xx₀</td>
<td></td>
</tr>
<tr>
<td>PC := (^word($xx₁xx₀))^</td>
<td>EIP := EIP + 2 + longint(shortint($xx₀))</td>
<td>Instruction size = 2 bytes</td>
</tr>
<tr>
<td><em>6502 assembler:</em></td>
<td><em>Intel assembler:</em></td>
<td></td>
</tr>
<tr>
<td>JMP ($xx₁xx₀)</td>
<td>JMP xx₀h</td>
<td></td>
</tr>
</tbody>
</table>

- $EA: Actual bytes of instruction’s start (base) address
- $90: Actual bytes of instruction’s machine code
- $xx₀, $xx₁: Actual bytes of instruction’s machine code
program PascalProgram;

type
  PProc = procedure;

procedure P1;
begin
  α
end;

procedure P2;
begin
  β
end;

var
  i : word;
  ptr : PProc;
  j : word;

begin
  γ₁
  ptr := @P1;
  ptr;
  P2;
  γ₂
end.
program PascalProgram;

type
  PProc = procedure;

procedure P1;
begin
  \(\alpha\)
end;

procedure P2;
begin
  \(\beta\)
end;

var
  i : word;
  ptr : PProc;
  j : word;

begin
  \(V_1\)
  ptr := @P1;

  ptr; \rightarrow \text{indirect jump}
  P2; \rightarrow \text{direct jump}

  \(V_2\)
end.
program PascalProgram;

type
  PProc = procedure;

procedure P1;
begin
  \[ \alpha \]
  jmp back
end;

procedure P2;
begin
  \[ \beta \]
  jmp back
end;

var
  i : word;
  ptr : PProc;
  j : word;

begin
  \[ V_1 \]
  ptr := @P1;
  ptr; \rightarrow \text{indirect jump}
  P2; \rightarrow \text{direct jump}
  \[ V_2 \]
  end.
\]
program PascalProgram;

type
    PProc = procedure;

procedure P1;
begin
    Alpha
end;  jmp back

procedure P2;
begin
    Beta
end; jmp back

var
    i : word;
    ptr : PProc;
    j : word;

begin
    V1
    ptr := @P1;

    ptr;  P2;

    V2
end.

main program

procedure P2

procedure P1

var
    i : word;
    ptr : PProc;

begin
    V1
    ptr := @P1;

    ptr;  P2;

    V2
end.

main program
program PascalProgram;

type
    PProc = procedure;

procedure P1;
begin
  α
end;  jmp back  A

procedure P2;
begin
  β
end;  jmp back  B

var
  i : word;
  ptr : PProc;
  j : word;

begin
  V1
  ptr := @P1;
  ptr;
  P2;
  V2
end.  main program  C1

variable j

variable ptr

padding

variable i

procedure P2

procedure P1

begin
  γ
  ptr := @P1;
  ptr;
  P2;
  γ
  end  C2
program PascalProgram;

type
  PProc = procedure;

procedure P1;
begin
  α
end;  jmp back

procedure P2;
begin
  β
end;  jmp back

var
  i : word;
  ptr : PProc;
  j : word;

begin
  γ
  ptr := @P1;
  ptr;  P2;
  γ
end.

main program

Variable j
variable ptr
padding
variable i
procedure P2
procedure P1

begin
  jmp E9
  jmp FF
end.
program PascalProgram;
type
  PProc = procedure;
procedure P1;
begin
  \( \alpha \) \text{ jmp back} \end{end}
end;
procedure P2;
begin
  \( \beta \) \text{ jmp back} \end{end}
end;
var
  i : word;
  ptr : PProc;
  j : word;
begin
  \( \gamma_1 \) \text{ ptr := @P1;} \end{end}
  \( \gamma_2 \) \text{ ptr;} \end{end}
  P2; \end{end}
  \( \gamma_3 \) \text{ end.} \end{end}
program PascalProgram;

type
  PProc = procedure;

procedure P1;
begins
  α
end; /* jmp back */

procedure P2;
begins
  β
end; /* jmp back */

var
  i : word;
  ptr : PProc;
  j : word;

begin
  V₂
  ptr := @P1;
  P2;
  V₁
end.

$00007A08
$00007A04
$00007A02
$00007A00

$00001306
$00001300

C1:

C2:

C1:

$00001000

A

$00002000

procedure P1;
begins
  jmp back
end;

procedure P2;
begins
  jmp back
end;

variablé i
variablé ptr
padding
variablé j
procedure P2
procedure P1

begin
  $00007A00
  0D
  $00007A02
  F5
  jmp $00002100

  jmp $00001306

  $00002100 ← $00002100 − ($001306 + 5)
  = $00002100 − $0000130B = $0000DF5

  jmp $00001300
  25
  jmp $00001306

  $00001000

Examples of CPU Architectures

<table>
<thead>
<tr>
<th>CPU arch.</th>
<th>CPU name</th>
<th>Logical address width</th>
<th>Current instruction register(s)</th>
<th>Physical address width</th>
<th>Special mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit 6502</td>
<td>MOS 6502</td>
<td>16-bit</td>
<td>PC</td>
<td>16-bit (64 kB)</td>
<td></td>
</tr>
<tr>
<td>16-bit x86-16 x86</td>
<td>Intel 8088</td>
<td>16 + 16 bit</td>
<td>CS:IP</td>
<td>20-bit (1 MB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Intel 8086</td>
<td>16 + 16 bit</td>
<td>CS:IP</td>
<td>24-bit (16 MB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Intel 80286</td>
<td>16 + 16 bit</td>
<td>CS:IP</td>
<td>24-bit (16 MB)</td>
<td>protected 16 (+ real) mode</td>
</tr>
<tr>
<td>32-bit x86 IA-32 INTEL32</td>
<td>Intel 80386</td>
<td>32-bit</td>
<td>EIP</td>
<td>32-bit (4 GB)</td>
<td>protected 32 mode</td>
</tr>
<tr>
<td></td>
<td>Intel Pentium Pro</td>
<td>32-bit</td>
<td>EIP</td>
<td>36-bit (64 GB)</td>
<td>PAE</td>
</tr>
<tr>
<td>64-bit x64 x86-64 AMD64 INTEL64 EM64T</td>
<td>AMD Opteron (Intel Pentium 4)</td>
<td>64-bit</td>
<td>RIP</td>
<td>40-bit (1 TB)</td>
<td>long mode</td>
</tr>
<tr>
<td></td>
<td>2015 current (e.g. Core i7)</td>
<td>64-bit</td>
<td>RIP</td>
<td>AMD: 48b → 256 TB Intel: 46b → 64 TB</td>
<td></td>
</tr>
</tbody>
</table>
## Examples of CPU Architectures

<table>
<thead>
<tr>
<th>CPU arch.</th>
<th>CPU name</th>
<th>Logical address width</th>
<th>Current instruction register(s)</th>
<th>Physical address width</th>
<th>Special mode</th>
<th>Stack top</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit 6502</td>
<td>MOS 6502</td>
<td>16-bit</td>
<td>PC</td>
<td>16-bit (64 kB)</td>
<td>01 S</td>
<td>SS:SP</td>
</tr>
<tr>
<td>16-bit x86-16</td>
<td>Intel 8088</td>
<td>16 + 16 bit</td>
<td>CS:IP</td>
<td>20-bit (1 MB)</td>
<td>SS:SP</td>
<td></td>
</tr>
<tr>
<td>x86-16</td>
<td>Intel 8086</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Intel 80286</td>
<td>16 + 16 bit</td>
<td>CS:IP</td>
<td>24-bit (16 MB)</td>
<td>SS:SP</td>
<td></td>
</tr>
<tr>
<td>32-bit x86 IA-32 INTEL32</td>
<td>Intel 80386</td>
<td>32-bit</td>
<td>EIP</td>
<td>32-bit (4 GB)</td>
<td>ESP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Intel Pentium Pro</td>
<td>32-bit</td>
<td>EIP</td>
<td>36-bit (64 GB)</td>
<td>ESP</td>
<td></td>
</tr>
<tr>
<td>64-bit x64 x86-64 AMD64 INTEL64 EM64T</td>
<td>AMD Opteron (Intel Pentium 4)</td>
<td>64-bit</td>
<td>RIP</td>
<td>40-bit (1 TB)</td>
<td>RSP</td>
<td></td>
</tr>
<tr>
<td>AMD: 48b → 256 TB Intel: 46b → 64 TB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2015 current (e.g. Core i7)</td>
<td>64-bit</td>
<td>RIP</td>
<td></td>
<td></td>
<td>RSP</td>
<td></td>
</tr>
</tbody>
</table>
### Push and Pop Variants on x86 (IA-32)

<table>
<thead>
<tr>
<th>Machine code</th>
<th>Intel assembler</th>
<th>Comment</th>
</tr>
</thead>
</table>
| 68 xx xx xx xx | PUSH xxxxxxxxxxh  
(or PUSH DWORD PTR xxxxxxxxxxh) | push 32-bits of x  (x = immediate) |
## Push and Pop Variants on x86 (IA-32)

<table>
<thead>
<tr>
<th>Machine code</th>
<th>Intel assembler</th>
<th>Comment</th>
</tr>
</thead>
</table>
| 68 xx xx xx xx | PUSH xxxxxxxxh (or PUSH DWORD PTR xxxxxxxxh) | push 32-bits of x \( x = \text{immediate} \)  
SP := SP – 4 |

\( = \text{sizeof}(x) \)
### Push and Pop Variants on x86 (IA-32)

<table>
<thead>
<tr>
<th>Machine code</th>
<th>Intel assembler</th>
<th>Comment</th>
</tr>
</thead>
</table>
| 68 xx xx xx xx | PUSH xxxxxxxxxh (or PUSH DWORD PTR xxxxxxxxxh) | push 32-bits of $x$ ($x = \text{immediate}$)  
$SP := SP - 4$  
$(^\text{longword}(SP))^ := x$ = sizeof($x$) |
### Push and Pop Variants on x86 (IA-32)

<table>
<thead>
<tr>
<th>Machine code</th>
<th>Intel assembler</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>68 xx xx xx xx</td>
<td>PUSH xxxxxxxxxh</td>
<td>push 32-bits of x (x = immediate)</td>
</tr>
<tr>
<td></td>
<td>(or PUSH DWORD PTR xxxxxxxxxh)</td>
<td>SP := SP - 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(^longword(SP))^ := x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EIP := EIP + 5</td>
</tr>
</tbody>
</table>

- = sizeof(x)
- = sizeof(68 PUSH instruction)
# Push and Pop Variants on x86 (IA-32)

<table>
<thead>
<tr>
<th>Machine code</th>
<th>Intel assembler</th>
<th>Comment</th>
</tr>
</thead>
</table>
| 68 xx xx xx xx | PUSH xxxxxxxxh (or PUSH DWORD PTR xxxxxxxxh) | push 32-bits of x \( (x = \text{immediate}) \)  
\[ \text{SP} := \text{SP} - 4 \]  
\[ (^{\text{longword}}(\text{SP}))^ := x \] |
| 66 68 xx xx | PUSH xxxxh (or PUSH WORD PTR xxxxh) | push 16-bits of x  
\[ \text{SP} := \text{SP} - 2 \]  
\[ (^{\text{word}}(\text{SP}))^ := x \] |

\[ \text{EIP} := \text{EIP} + \text{sizeof(PUSH)} \]
## Push and Pop Variants on x86 (IA-32)

<table>
<thead>
<tr>
<th>Machine code</th>
<th>Intel assembler</th>
<th>Comment</th>
</tr>
</thead>
</table>
| 68 xx xx xx xx | PUSH xxxxxxxxh (or PUSH DWORD PTR xxxxxxxxh) | push 32-bits of x \( (x = \text{immediate}) \)  
\( \text{SP} := \text{SP} - 4 \)  
\( (^\text{longword}(\text{SP}))^ := x \) |
| 66 68 xx xx | PUSH xxhh (or PUSH WORD PTR xxhh) | push 16-bits of x  
\( \text{SP} := \text{SP} - 2 \)  
\( (^\text{word}(\text{SP}))^ := x \) |
| FF 35 xx xx xx xx | PUSH [xxxxxxxxxh] (or PUSH DWORD PTR [xxxxxxxxxh]) | push 32-bits from 32-bit address x \( (x = \text{absolute address}) \)  
\( \text{SP} := \text{SP} - 4 \)  
\( (^\text{longword}(\text{SP}))^ := (^\text{longword}(x))^ \) |

- \( ^\text{longword}(x)^ \) = sizeof(x)
- \( ^\text{longword}(\text{SP})^ \) = sizeof(longword)
## Push and Pop Variants on x86 (IA-32)

<table>
<thead>
<tr>
<th>Machine code</th>
<th>Intel assembler</th>
<th>Comment</th>
</tr>
</thead>
</table>
| 68 xx xx xx xx | PUSH xxxxxxxhh  
(or PUSH DWORD PTR xxxxxxxh) | push 32-bits of x  
(x = immediate)  
SP := SP - 4  
(^longword(SP))^ := x |
| 66 68 xx xx | PUSH xxxxxh  
(or PUSH WORD PTR xxxxxh) | push 16-bits of x  
SP := SP - 2  
(^word(SP))^ := x |
| FF 35 xx xx xx xx | PUSH [xxxxxxh]  
(or PUSH DWORD PTR [xxxxxxh]) | push 32-bits from 32-bit address x  
(x = absolute address)  
SP := SP - 4  
(^longword(SP))^ := (^longword(x))^ |
| 66 FF 35 xx xx xx xx | PUSH WORD PTR [xxxxxxh] | push 16-bits from 32-bit address x  
SP := SP - 2  
(^word(SP))^ := (^word(x))^ |
### Push and Pop Variants on x86 (IA-32)

<table>
<thead>
<tr>
<th>Machine code</th>
<th>Intel assembler</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>68 xx xx xx xx</strong></td>
<td><strong>PUSH xxxxxxxxxh</strong>&lt;br&gt;(or <strong>PUSH DWORD PTR xxxxxxxxxh</strong>)</td>
<td>push 32-bits of (x) ((x = \text{immediate}))&lt;br&gt;(SP := SP - 4)&lt;br&gt;((^\text{longword}(SP)))^ := (x)</td>
</tr>
<tr>
<td><strong>66 68 xx xx</strong></td>
<td><strong>PUSH xxxxxh</strong>&lt;br&gt;(or <strong>PUSH WORD PTR xxxxxh</strong>)</td>
<td>push 16-bits of (x)&lt;br&gt;(SP := SP - 2)&lt;br&gt;((^\text{word}(SP)))^ := (x)</td>
</tr>
<tr>
<td><strong>FF 35 xx xx xx xx</strong></td>
<td><strong>PUSH [xxxxxxxxxh]</strong>&lt;br&gt;(or <strong>PUSH DWORD PTR [xxxxxxxxxh]</strong>)</td>
<td>push 32-bits from 32-bit address (x) ((x = \text{absolute address}))&lt;br&gt;(SP := SP - 4)&lt;br&gt;((^\text{longword}(SP)))^ := ((^\text{longword}(x)))^</td>
</tr>
<tr>
<td><strong>66 FF 35 xx xx xx xx</strong></td>
<td><strong>PUSH WORD PTR [xxxxxxxxxh]</strong></td>
<td>push 16-bits from 32-bit address (x)&lt;br&gt;(SP := SP - 2)&lt;br&gt;((^\text{word}(SP)))^ := ((^\text{word}(x)))^</td>
</tr>
<tr>
<td><strong>8F 05 xx xx xx xx</strong></td>
<td><strong>POP [xxxxxxxxxh]</strong>&lt;br&gt;(or <strong>POP DWORD PTR [xxxxxxxxxh]</strong>)</td>
<td>pop 32-bits and save them to 32-bit address (x) ((x = \text{absolute address}))&lt;br&gt;((^\text{longword}(x)))^ := ((^\text{longword}(SP)))^&lt;br&gt;(SP := SP + 4)</td>
</tr>
<tr>
<td>Machine code</td>
<td>Intel assembler</td>
<td>Comment</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
<td>---------</td>
</tr>
</tbody>
</table>
| 68 xx xx xx xx | PUSH xxxxxxxxh  
(or PUSH DWORD PTR xxxxxxxxh) | push 32-bits of x  
(x = immediate)  
SP := SP - 4  
(^longword(SP))^ := x |
| 66 68 xx xx | PUSH xxxh  
(or PUSH WORD PTR xxxh) | push 16-bits of x  
SP := SP - 2  
(^word(SP))^ := x |
| FF 35 xx xx xx xx | PUSH [xxxxxxxxxh]  
(or PUSH DWORD PTR [xxxxxxxxxh]) | push 32-bits from 32-bit address x  
(x = absolute address)  
SP := SP - 4  
(^longword(SP))^ := (^longword(x))^ |
| 66 FF 35 xx xx xx xx | PUSH WORD PTR [xxxxxxxxxh] | push 16-bits from 32-bit address x  
SP := SP - 2  
(^word(SP))^ := (^word(x))^ |
| 8F 05 xx xx xx xx | POP [xxxxxxxxxh]  
(or POP DWORD PTR [xxxxxxxxxh]) | pop 32-bits and save them to 32-bit address x  
(x = absolute address)  
(^longword(x))^ := (^longword(SP))^  
SP := SP + 4 |
| 66 8F 05 xx xx xx xx | POP WORD PTR [xxxxxxxxxh] | pop 16-bits and save them to 32-bit address x  
(^word(x))^ := (^word(SP))^  
SP := SP + 2 |
program PascalProgram;

type
    PProc = procedure;

procedure P1;
begin
  α
  jmp back
end;

procedure P2;
begin
  β
  jmp back
end;

var
  i : word;
  ptr : PProc;
  j : word;

begin
  \( \upsilon_1 \) ptr := @P1;
  ptr;
  P2;
  \( \upsilon_2 \) end.

...
program PascalProgram;

type
  PProc = procedure;

procedure P1;
begin
  \(\alpha\) \quad \text{jmp back } \equiv \text{ret}
end;

procedure P2;
begin
  \(\beta\) \quad \text{jmp back } \equiv \text{ret}
end;

var
  i : word;
  ptr : PProc;
  j : word;

begin
  \(V_1\)
  ptr := @P1;
  \(V_2\)
  end.

jmp back = \text{ret}

program PascalProgram;

type
  PProc = procedure;

procedure P1;
begin
  α
end;  jmp back ≡ ret

procedure P2;
begin
  β
end;  jmp back ≡ ret

var
  i : word;
  ptr : PProc;
  j : word;

begin
  V1
  ptr := @P1;

  ptr;
  P2;

  V2
end.

...
program PascalProgram;

type
    PProc = procedure;

procedure P1;
begin
    α
    jmp back = ret
end;

procedure P2;
begin
    β
    jmp back = ret
end;

var
    i : word;
    ptr : PProc;
    j : word;

begin
    \( V_1 \)
    ptr := @P1;
    ptr;
    \( V_2 \)
    P2;
    end.

CALL E8
$00001306
00
00
00
7A
04

CALL 15
$00001300
00
00
00
7A
00

E8 = relative call (E9 = relative jump)

$000001300 ← $00002100 − (001306 + 5)
= $00002100 − $0000130B = $00000DF5

FF 15 = indirect call (FF 25 = indirect jump)

$000001300 → $00F00018
$00F00017
$00F00016
$00F00015
$00F00014
$00F00013
$00F00012
$00F00011
$00F00010
$00F0000F
$00F0000E
$00F0000D
$00F0000C
$00F0000B
$00F0000A
$00F00009
$00F00008
$00F00007
$00F00006
$00F00005
$00F00004
$00F00003
$00F00002
$00F00001
$00F00000
program PascalProgram;

type
  PProc = procedure;

procedure P1;
begin
  \(\alpha\)
end;  \(\text{jmp back} \equiv \text{ret}\)

procedure P2;
begin
  \(\beta\)
end;  \(\text{jmp back} \equiv \text{ret}\)

var
  i : word;
  ptr : PProc;
  j : word;

begin
  \(V_1\)
  ptr := @P1;
  ptr;
  P2;
  \(V_2\)
end.

\(C_1\)

\(C_2\)

\(\text{procedure P1;\hspace{1cm}procedure P2;\hspace{1cm}variable i;\hspace{1cm}variable j;\hspace{1cm}variable ptr;\hspace{1cm}padding;\hspace{1cm}variable i;\hspace{1cm}procedure P1;\hspace{1cm}procedure P2;\hspace{1cm}main program}\)
program PascalProgram;

type
  PProc = procedure;

procedure P1;
begin
  α
end;  jmp back = ret

procedure P2;
begin
  β
end;  jmp back = ret

var
  i : word;
  ptr : PProc;
  j : word;

begin
  V1
  ptr := @P1;
  ptr;
  P2;
  V2
end.

procedure P1;
begin
  CALL E8
end.

procedure P2;
begin
  CALL indir FF
end.

$00001306 ← $00002100 - (\$001306 + 5) = $00002100 - \$0000130B = \$00000DF5

E8 = relative call (E9 = relative jump)

FF 15 = indirect call (FF 25 = indirect jump)
program PascalProgram;

type
    PProc = procedure;

procedure P1;
begin
    α
end; jmp back ⇔ ret

procedure P2;
begin
    β
end; jmp back ⇔ ret

var
    i : word;
    ptr : PProc;
    j : word;

begin
    γ
    ptr := @P1;
    γ
    ptr;
    P2;
    γ
end.

V1

V2
program PascalProgram;

type
  PProc = procedure;

procedure P1;
begin
  α
end;  jmp back ≡ ret

procedure P2;
begin
  β
end;  jmp back ≡ ret

var
  i : word;
  ptr : PProc;
  j : word;

begin
  \( V_1 \)
  ptr := @P1;

  \( V_2 \)
  P2;
end.

procedure P1;
begin
end

procedure P2;
begin
end

main program

var
  i : word;
  ptr : PProc;
  j : word;

begin
  \( V_1 \)
  ptr := @P1;

  \( V_2 \)
  P2;
end.

CALL E8

CALL 15

FF 15 = indirect call (FF 25 = indirect jump)

E8 = relative call (E9 = relative jump)
program PascalProgram;
type
  PProc = procedure;
procedure P1;
begin
  \( \alpha \)
end;  
  \( \text{jmp \ back} \equiv \text{ret} \)
procedure P2;
begin
  \( \beta \)
end;  
  \( \text{jmp \ back} \equiv \text{ret} \)
var
  i : word;
  ptr : PProc;
  j : word;
begin
  \( V_1 \)
  \( \text{ptr} := @P1; \)
  \( \text{ptr; P2;} \)
  \( V_2 \)
  \( \text{end.} \)
end.
program PascalProgram;

type
  PProc = procedure;

procedure P1;
begin
  α
  jmp back = ret
end;

procedure P2;
begin
  β
  jmp back = ret
end;

var
  i : word;
  ptr : PProc;
  j : word;

begin
  γ
  ptr := @P1;
  jmp back = ret
end.

procedure P1;
begin
  α
end;

end.

procedure P2;
begin
  β
end.

var
  i : word;
  ptr : PProc;
  j : word;

begin
  γ
  ptr := @P1;
  jmp back = ret
end.

procedure P1;
begin
  α
end;

end.
procedure P1(a : word;
    b : longword);
begin
    ...
    P1($AABB, $11223344);
    ...
end;

begin
    ...
    P1(5, 7);
    ...
end.
procedure P1(a : word;
           b : longword);
$0A50: ...
        push 11223344h
        push AABBh
        call 00000A50h
        ...  
        ret

begin
    ...  
    push 00000007h ← IP
    push 0005h
$0900: call 00000A50h
$0905: nop
    ...  
end.

P1($AABB, $11223344);
var { global variables }
x : longword;
y : longword;

procedure P1(a : longword;
    b : longword;
    var c : longword);

$0A50: ... 
  ... 
  ret

begin { main program }
  ...
  ...
  P1(5, x, y);
  ...
  ...
end.
var { global variables }
x : longword;
y : longword;

procedure P1(a : longword;
  b : longword;
  var c : longword);
$0A50: ... ...
  ret
begin { main program }
  ...  
  push ?
  push ?
  push ?
$0900: call 00000A50h
$0905: ...
  ...
end.
var \{ \textit{global variables} \}
\begin{align*}
\textit{x} & : \text{longword;} \\
\textit{y} & : \text{longword;}
\end{align*}

procedure \texttt{P1}(\textit{a} : \text{longword};
\textit{b} : \text{longword};
\textit{c} : \text{longword});

\$\texttt{0A50}: \ldots$
\begin{align*}
\ldots & \text{ret} \\
\text{begin \{ main program \}} \\
\ldots & \text{push ?} \\
\ldots & \text{push ?} \\
\ldots & \text{push 00000005h} \\
\$\texttt{0900}: \text{call 0000A50h}$
\$\texttt{0905}: \ldots$
\end{align*}

end.
var { global variables }
    x : longword;
    y : longword;

procedure P1(a : longword;
    b : longword;
    var c : longword);
$0A50: ... 
... 
begin { main program } 
... 
    ret 
begin { main program } 
... 
    push ? 
    push [00001004h] 
    push 00000005h 
$0900: call 00000A50h 
$0905: ... 
... 
end.
var { global variables }
    x : longword;
    y : longword;

procedure P1(a : longword;
    b : longword;
    var c : longword);

$0A50: ... ...
    ... ret

begin { main program }
    ... push 00001008h
    push [00001004h]
    push 00000005h
    $0900: call 00000A50h
$0905: ... ...
end.
procedure P1(a : word;
  b : longword);

$0A50: ...$
push 11223344h
push AABBh
call 00000A50h
...
ret

begin...
push 00000007h ← IP
push 0005h
$0900: call 00000A50h
$0905: nop
...
end.
procedure P1(a : word;
    b : longword);

$0A50: ... 
    push 11223344h
    push AABBh
    call 00000A50h
    ... 
    ret

begin 
    ... 
    push 00000007h ← IP
    push 0005h
$0900: call 00000A50h
$0905: nop
    ... 
end.
procedure P1(a : word;
    b : longword);
$0A50: ...}
push 11223344h
push AABBh
call 00000A50h
...
ret

begin
...
push 00000007h
push 0005h ← IP
$0900: call 00000A50h
$0905: nop
...
end.
procedure P1(a: word;
    b: longword);

$0A50: ...
    push 11223344h
    push AABBh
    call 00000A50h
    ...
    ret

begin ...
    push 00000007h
    push 0005h ← IP
$0900: call 00000A50h
$0905: nop ...
end.
procedure P1(a : word;
    b : longword);

$0A50: ... push 11223344h
        push AABBH
        call 00000A50h
        ...
        ret

begin
    ...
    push 00000007h
    push 0005h
$0900: call 00000A50h ← IP
$0905: nop
    ...
end.
procedure P1(a : word;
    b : longword);
$0A50: ...  
    push 11223344h  
    push AAB Bh  
    call 00000A50h  
    ...  
    ret

begin  
    ...  
    push 00000007h  
    push 0005h  
$0900: call 00000A50h ← IP
$0905: nop
    ...  
end.
procedure P1(a : word;
b : longword);

$0A50: ... ← IP
push 11223344h
push AAB Bh
call 00000A50h
...
ret

begin
...
push 0000007h
push 0005h
$0900: call 00000A50h
$0905: nop
...
end.
procedure P1(a : word;
     b : longword);

$0A50: ... push 11223344h ← IP
             push AABBh
             call 00000A50h
... ret

begin ...
             push 00000007h
             push 0005h
$0900: call 00000A50h
$0905: nop ...
end.
procedure P1(a : word;  
    b : longword);

$0A50: ...  
    push 11223344h ← IP  
    push AABBh  
    call 00000A50h  
    ...  
    ret

begin  
    ...  
    push 00000007h  
    push 0005h  
    $0900: call 00000A50h  
    $0905: nop  
    ...  
end.
procedure P1(a : word;
  b : longword);

$0A50$: ...
  push 11223344h
  push AAB Bh ← IP
  call 00000A50h
  ...
  ret

begin
  ...
  push 00000007h
  push 0005h
$0900$: call 00000A50h
$0905$: nop
$0905$: ...
end.
procedure P1(a : word;
    b : longword);

$0A50: ... push 11223344h
push AABBh ← IP
call 00000A50h
...
ret

begin ...
push 00000007h
push 0005h
$0900: call 00000A50h
$0905: nop ...
end.
procedure P1(a : word;
       b : longword);
$0A50: ... push 11223344h
      push AABBh
      call 00000A50h ← IP
$0A70: ... ret
begin ...
      push 00000007h
      push 0005h
$0900: call 00000A50h
$0905: nop
      ...
end.
procedure P1(a : word;
    b : longword);

$0A50: ...
    push 11223344h
    push AABBh
    call 00000A50h ← IP
$0A70: ...
    ret
begin
    ...
    push 00000007h
    push 0005h
$0900: call 00000A50h
$0905: nop
    ...
end.
procedure P1(a : word;
    b : longword);
$0A50: ... ← IP
    push 11223344h
    push AABBh
    call 00000A50h
$0A70: ...
    ret
begin
    ...
    push 00000007h
    push 0005h
$0900: call 00000A50h
$0905: nop
    ...
end.
procedure P1(a : word;  
    b : longword);

$0A50: ...  
    push 11223344h  
    push AABBh  
    call 00000A50h  

$0A70: ...  
    ret  ← IP

begin  
    ...  
    push 00000007h  
    push 0005h  

$0900: call 00000A50h  
$0905: nop  
    ...  
end.
procedure P1(a : word;
  b : longword);

$0A50: ...  
  push 11223344h
  push AABBh
  call 00000A50h

$0A70: ...  ← IP
ret

begin
  ...
  push 00000007h
  push 0005h
$0900: call 00000A50h
$0905: nop
  ...  
end.
procedure P1(a : word;
           b : longword);
$0A50: ...  
    push 11223344h
    push AABBh
    call 00000A50h
$0A70: ...  
    ret ← IP

begin ...
    push 00000007h
    push 0005h
$0900: call 00000A50h
$0905: nop ...
end.
procedure P1(a : word;
b : longword);
$0A50: ... push 122344h
push ABBh
call 00000A50h
$0A70: ... ret

begin
... push 0000007h
push 0005h
$0900: call 00000A50h
$0905: nop
...
end.
procedure P1(a : word;
   b : longword);

$0A50: ...$
   push 11223344h
   push AABBh
   call 00000A50h
$SP := SP + (4+2) ← IP$

$0A70: ...
   ret

begin
   ...
   push 00000007h
   push 0005h
$0900: call 00000A50h
$0905: SP := SP + (4+2) ← IP
   nop
   ...
end.
This code is just pseudocode with Pascal-like syntax! The “? := ? + (? + ?)” command is in general not a valid instruction. It is also not a valid Pascal program, because we cannot directly access SP register in Pascal code (or any other CPU register).

The whole $SP := SP + (4 + 2)$ expression has to be further compiled into an actual instruction sequence yet – see lectures later in this semester.
procedure P1(a : word;
    b : longword);
$0A50: ... push 11223344h
push AABBh
call 00000A50h
SP := SP + (4+2) ← IP
$0A70: ...
    ret

begin
    ...
push 00000007h
push 0005h
$0900: call 00000A50h
$0905: SP := SP + (4+2)
nop ...
end.
procedure P1(a : word;
   b : longword);
$0A50: ... ...
push 11223344h
push AAB Bh
call 00000A50h
SP := SP + (4+2)
$0A70: ...
ret ← IP
begin
... ...
push 00000007h
push 0005h
$0900: call 00000A50h
$0905: SP := SP + (4+2)
nop ...
end.
procedure P1(a : word;
   b : longword);

$0A50: ... 
   push 11223344h
   push AABBh
   call 00000A50h
   SP := SP + (4+2)
$0A70: ...
   ret

begin ...
   push 00000007h
   push 0005h
$0900: call 00000A50h
$0905: SP := SP + (4+2) ← IP
   nop ...
end.
procedure P1(a : word;
    b : longword);
$0A50: ...$0A50h
    push 112344h
    push AABZh
    call 00000A50h
    SP := SP + (4+2)
$0A70: ...$0A70h
    ret

begin
  ...
    push 0000007h
    push 0005h
$0900: call 00000A50h
$0905: SP := SP + (4+2) ← IP
    nop
  ...
end.
procedure P1(a : word;
    b : longword);
$0A50: ... push 11223344h
    push AAB Bh
    call 00000A50h
    SP := SP + (4+2)
$0A70: ...
    ret

begin ...
    push 00000007h
    push 0005h
$0900: call 00000A50h
$0905: SP := SP + (4+2)
    nop ← IP
    ... end.