Stack Instructions

- **PUSH op**
  - the stack pointer ESP is decremented by the size of the operand
  - the operand is copied to [ESP]
- **POP op**
  - the reverse of PUSH
  - [ESP] is copied to the destination operand
  - ESP is incremented by the size of the operand
Stack Instructions

- Where is the stack?
  - The stack has its own section
  - Linux processes wake up with ESP initialized properly
  - The stack grows "upward" – toward smaller addresses
  - Memory available to the stack set using "limit"
### POP—Pop a Value from the Stack

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>Load the 16 bits from the memory location specified by the base register (bp) into ra.</td>
</tr>
<tr>
<td>000100</td>
<td>Load the 16 bits from the memory location specified by the base register (bp) into rb.</td>
</tr>
<tr>
<td>001000</td>
<td>Load the 16 bits from the memory location specified by the base register (bp) into rc.</td>
</tr>
<tr>
<td>010000</td>
<td>Load the 16 bits from the memory location specified by the base register (bp) into rd.</td>
</tr>
<tr>
<td>011000</td>
<td>Load the 16 bits from the memory location specified by the base register (bp) into re.</td>
</tr>
<tr>
<td>100000</td>
<td>Load the 16 bits from the memory location specified by the base register (bp) into rf.</td>
</tr>
</tbody>
</table>

### POP—Pop a Value from the Stack (Continued)

The POP instruction pops a value from the stack and stores it in the register specified by `Rt`. The POP instruction is used to pop the value off the stack and store it in the register specified by `Rt`. The POP instruction is similar to the POP instruction in other versions of the MIPS architecture. The POP instruction is used to pop the value off the stack and store it in the register specified by `Rt`. The POP instruction is similar to the POP instruction in other versions of the MIPS architecture.
Subroutine Instructions

- **CALL label**
  - Used to call a subroutine
  - PUSHes the instruction pointer (EIP) on the stack
  - jump to the label
  - does NOTHING else

- **RET**
  - reverse of CALL
  - POPs the instruction pointer (EIP) off the stack
  - execution proceeds from the instruction after the CALL instruction

- **Parameters?**