

---

## Implementing a Stack

---

- A **stack** is a **data structure** with **two** operation:
  - void Push(int x): put the value "x" in the stack. The value "x" is put "at the top" of the stack.
  - int Pop(): remove the top value off the stack. The value removed is returned.

### Example:

(1) Initially:	+-----+	(empty stack)
(2) After Push(4):	+-----+   4   +-----+	
(3) After Push(9):	+-----+   9   +-----+   4   +-----+	(new value get put ON TOP)
(4) After Push(1):	+-----+   1   +-----+   9   +-----+   4   +-----+	(new value get put ON TOP again)
Suppose we have 3 variables "a", "b" and "c" defined....		
(5) After a = Pop():	+-----+   9   +-----+   4   +-----+	(top value is REMOVED !)
and a = 1		
(6) After b = Pop():	+-----+   4   +-----+	(top value is REMOVED again !)
and b = 9		
(7) After c = Pop():	+-----+	(Stack is empty)
and c = 4		

---



---

- **Stack in a program versus the *System Stack***

- The **stack** that you learned in **Java** is usually **implemented using** an **array variable**

**Example:** (CS171 material)

```
public class Stack
{
    int[] A;
    int stacktop;    // Points to the top of the stack

    public Stack(int size)
    {
        A = new int[size];    // Create the array to hold values in stack
        stacktop = -1;
    }

    public void push( int x )
    {
        A[++stacktop] = x;    // Move stack top and put x on the stack
    }

    public int pop( )
    {
        return A[stacktop--];
    }
}
```

- When a **program** is **running**, the **computer system** will need to **store** various **information items (variables)** in a **stack**

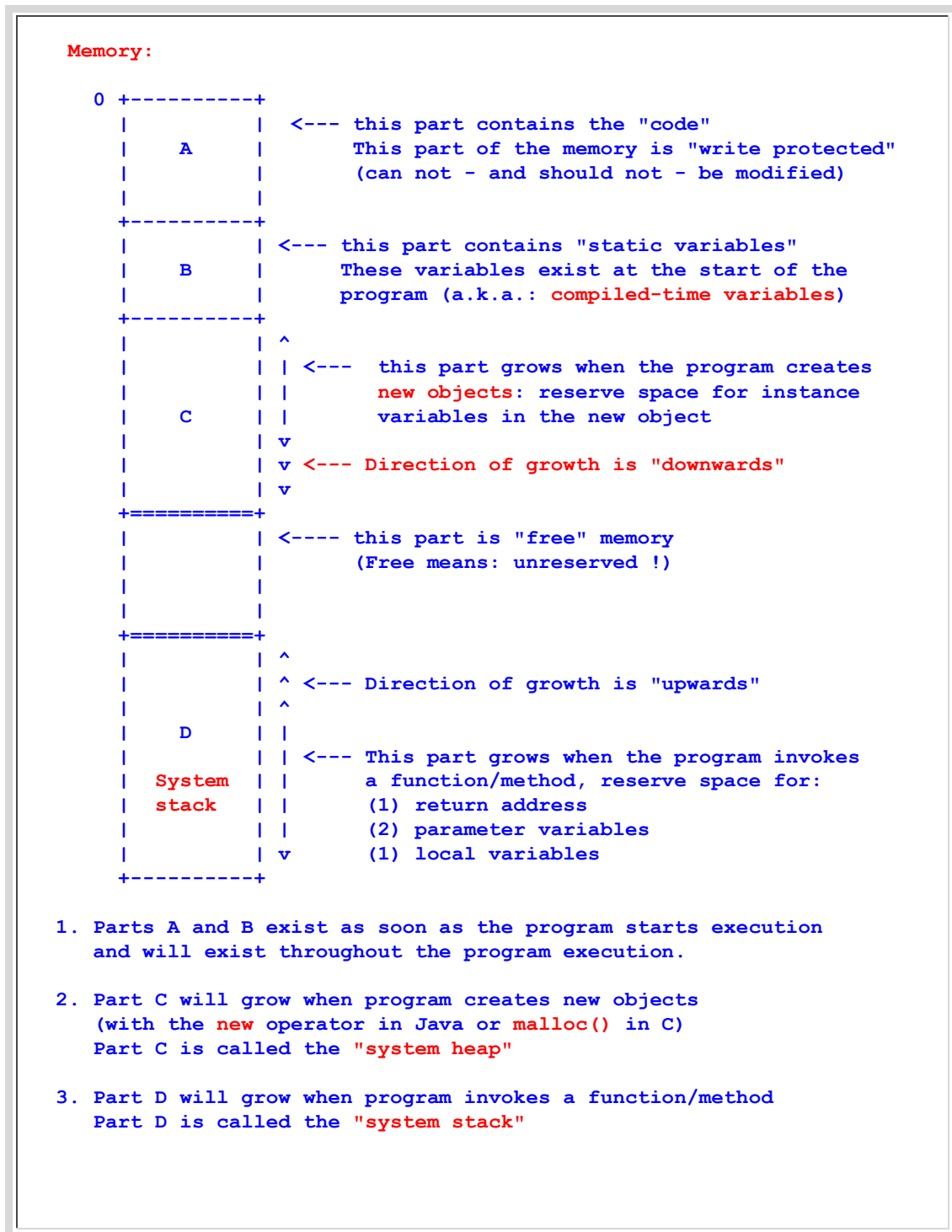
**Therefore:**

- All modern **computers** maintain a **program stack** (a.k.a. **System stack**) to manage **information**

**Information** stored in the **system stack** include:

- **function activation information** (created when a **function** is **called**  
(These **function activation information** will be **removed (popped)** when a function **returns (exits)**)

- The reason we **want** to use a **stack** is the **order** the elements enter and leave the stack is **exactly** the same order of **function activation and deactivation**.
- The **system stack** is (always) stored inside the **memory** (memory stores everything :-))
- When a **program** is **running**, the **memory** is **organized** as follows:



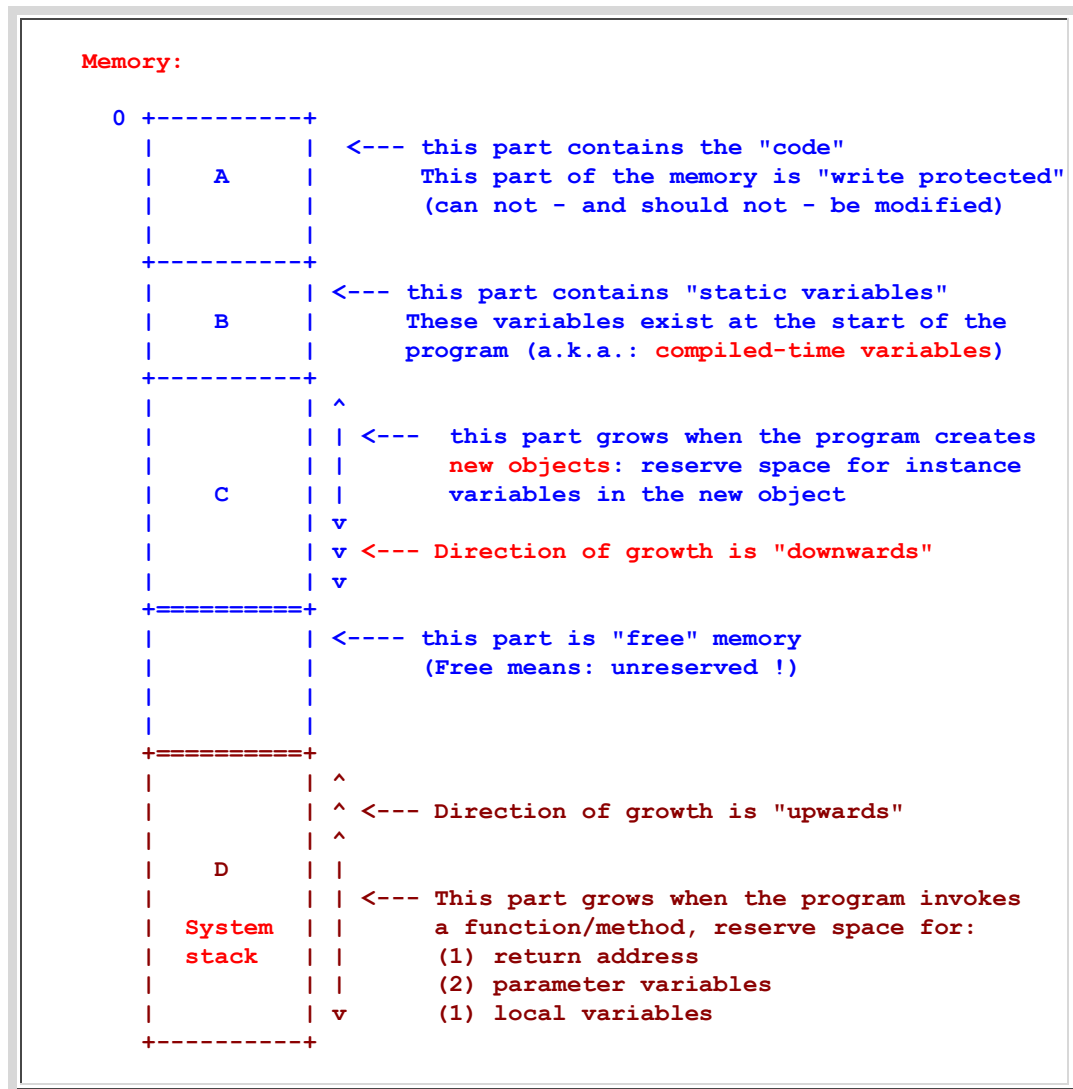
### • Implementing the System Stack

- Just like a **stack** object in **Java**, in order to **implement** a **stack**, we need:

- An **array**
- A **StackTop** index

- **Unlike** in a **programming language**, we have access to the **entire computer memory** when we program in **assembler** !!!

So the **system stack** that is **located** at the **end** of the **memory area**:



do **not** need to be **defined** (as an **array** as in **Java**)

(The **memory** is **there** and it is **reserved** for us to make a **stack**)

- All we need to **implement** the **System Stack** is:

■ A **stacktop** index

- The **system stack** is **always** implemented by:

■ Using **one** of the **registers** in the **CPU** as **stacktop** index

**Example:**

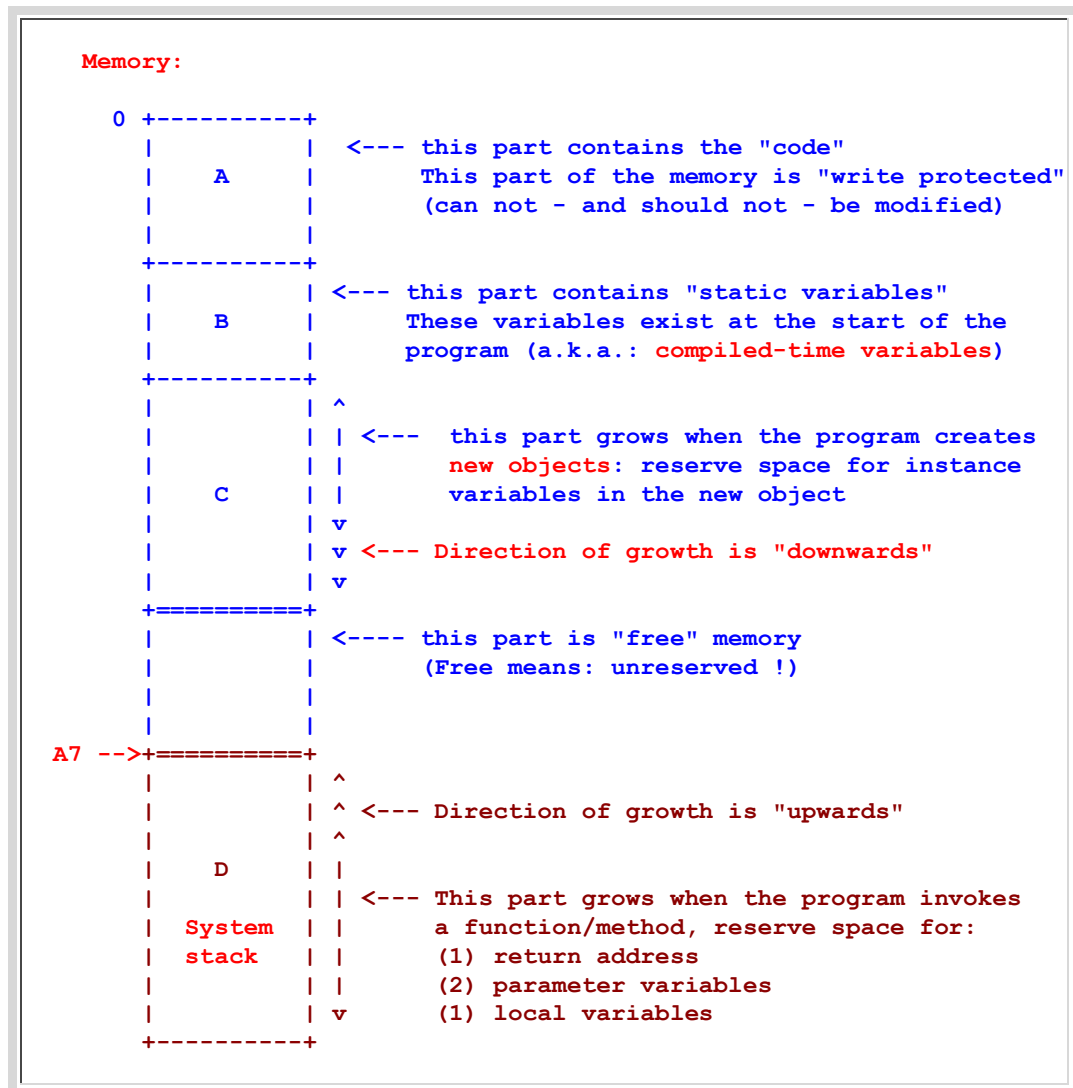
- In **M68000**, the **stacktop** index is:

■ The **address register A7**

- In **Intel's** CPU (Pentium, Core Duo...), the **stacktop** index is stored in:

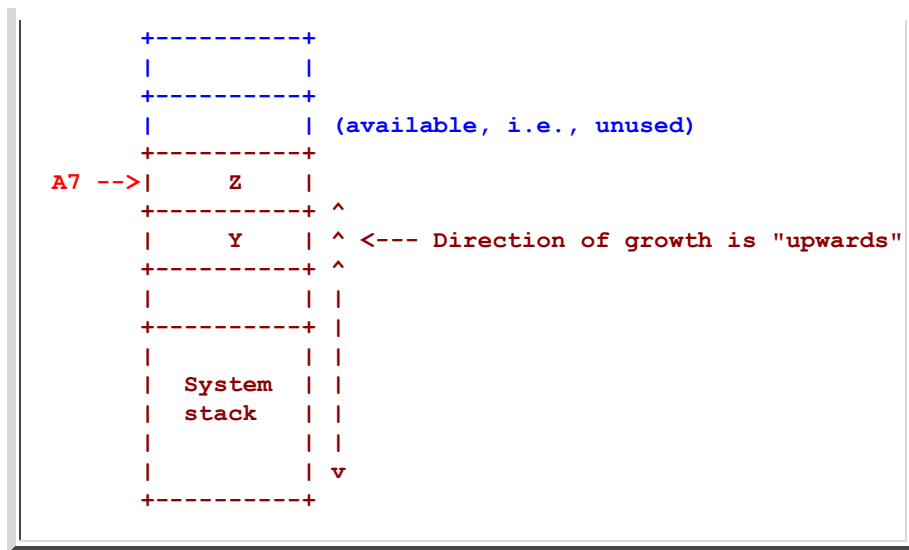
■ The **special purpose register (named) sp** (= stack pointer)

- So in **M68000**, the **system stack** is given as follows:

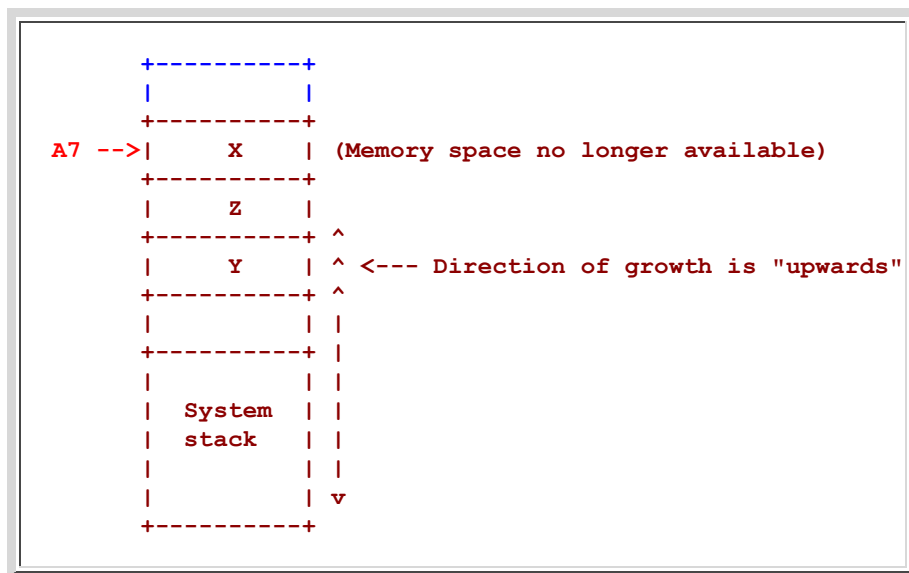


### • Pushing values onto the System Stack

- **Suppose** the **System Stack** is **as follows before** we **push** a new value:



- The **Suppose** the **System Stack** will be **like this after** we **push** a new value **X**:



(The **memory space** is **no longer available** because **all memory spaces** "at and below" the **stack pointer A7** are **used** !!!!)

- The following **assembler instructions** will **push** a **integer (4 byte)** value **X** onto the **System Stack**:

```

suba.l  #4, A7           // Move system stacktop up 4 bytes
move.l  X, (A7)         // Store X in the top of the stack

```

- **De-allocating variables from the System Stack**

- The following **assembler instructions** will **destroy (= unreserve memory)** a **integer (4 byte)** value from the **System Stack**:

```
adda.l    #4, A7           // Move system stacktop down 4 bytes
```

- **The system stack pointer**

- **Stack pointer:**

- The **stacktop index** (A7) is **more commonly known** as:

- **(System) Stack pointer**

- It is **usually** abbreviated as: **sp** (for **stack pointer**)

- **Important functions of the system stack pointer A7**

- **Function** of the **stack pointer A7:**

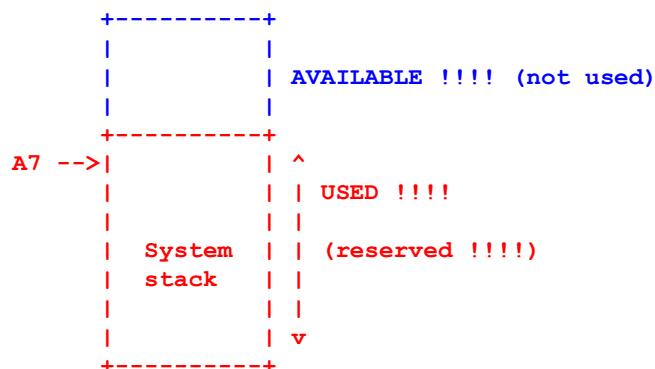
- **A7** indicate the **top** of the **stack**; as such:

- **A7** points to the **location** in the **stack** that the **push** and **pop** operations will **operate**

- **Another function** of **A7** is:

- **A7 marks** the **memory locations** that are **reserved** (= **currently used** !!!)

Specifically:



---

---

○ **Therefore:**

- When you perform a **push** operation (= **increase** the **stack size**), you will:

- **create** one or more **variables** (on the **stack !!!**)

---

- When you perform a **pop** operation (= **decrease** the **stack size**), you will:

- **destroy** one or more **variables** (on the **stack !!!**)

---

---

---