

---

## Modern programming languages

---

- **Modern programming languages**

- **Fact:**

- **All modern programming languages** supports *recursion*
- 
- 
- 

- **Detecting recursion**

- **Fact:**

- It is **very hard** to **detect** if a **function** is *recursive* due to the *indirect recursion* phenomenon
- 

- **Indirect recursion:**

```
void A(...)  
{  
    ....  
    B(...);  
    ....  
}  
  
void C(...)  
{  
    ....  
    D(...);  
    ....  
}  
  
void D(...)  
{  
    ....  
    A(...);  
    ....  
}
```

The **call chain** can be **arbitrary deep** !!!

---

---

---

- **Compilers of modern languages**

- **Fact:**

- The **compiler** of **modern programming language** will **not** try to **detect** if some **function/method** is **recursive** or not
- 
- The **compiler** of **modern programming language** will **assume** the **worst case scenario**:
    - It will **provide support** for **recursion** for **every function** that it **compiles.....**

- **Recursion and storing local variable in memory variables**

- The technique to use **memory variables** created with **ds** to store **local variables** will **only work** for **non-recursive** of **subroutine call**:

<pre>main() {     int a, b, c;      c = sum(a, b); }</pre>	<pre>int sum(int x, int y) {     x = x + 1;     y = y + 1;      return (x*x + y*y); }</pre>
--	---

- Storing **local variables** in **memory variable** using **ds** will not easily with **recursive calls**:

<pre>main() {     c = func_1(a, b); }</pre>	<pre>int func(int x, int y) {     int k, l, m;     ....      m = func(k, l); }</pre>
---	--

- **Reason:**

- When the function **func** is called a **second time**, **another copy of the local variable must be created** that will be used by the **second function call !!!**

The **ds** assembler directive can **only** create (= reserve space) for **one copy** of local

**variable !!!**

---

---

---

---

---

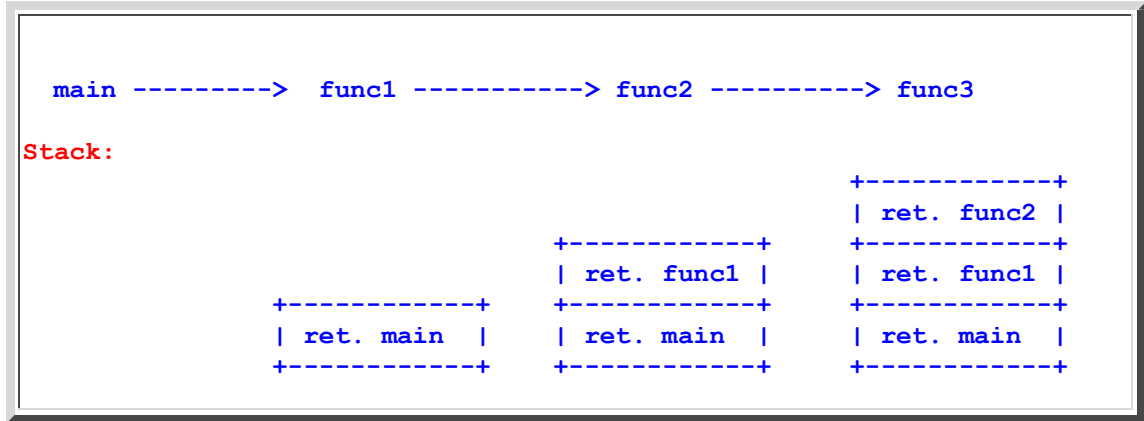
---

---

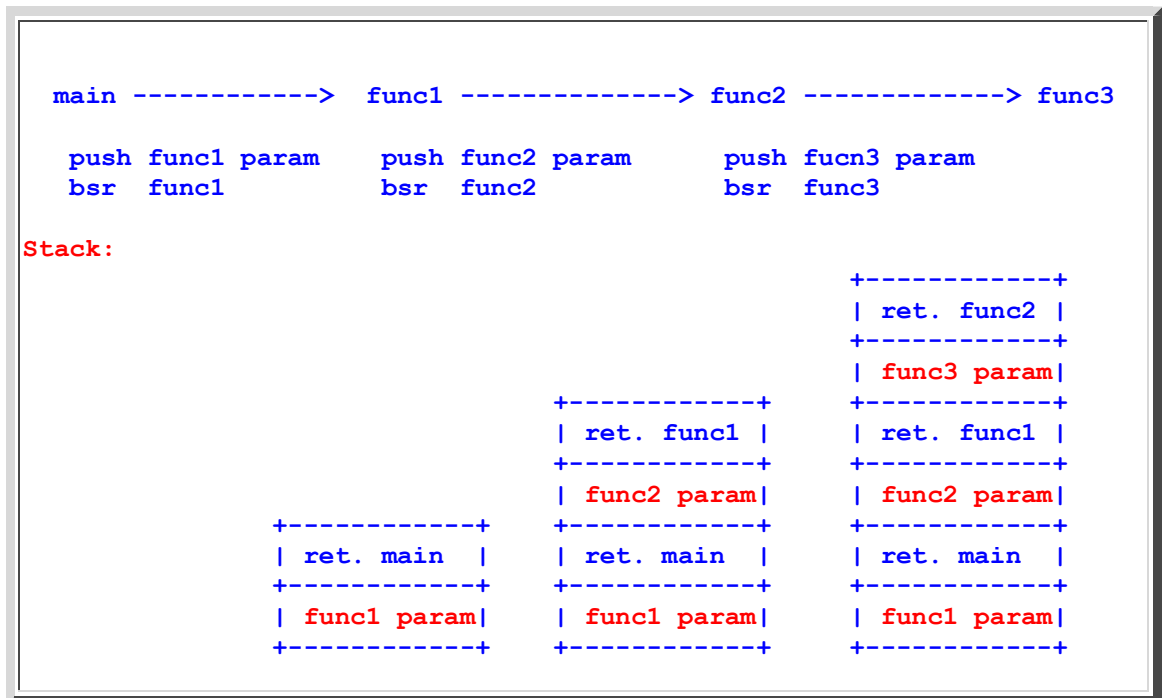
---

• **Using the system stack to pass parameters**

- It is **natural** to pass parameters using the stack because the way that the functions are activated and de-activated:



- The following figure shows how each function passes **ONE** parameter its own callee function
  - **main** pass one parameter (**func1 param**) to **func1**
  - **func1** pass one parameter (**func2 param**) to **func2**
  - **func2** pass one parameter (**func3 param**) to **func3**




---

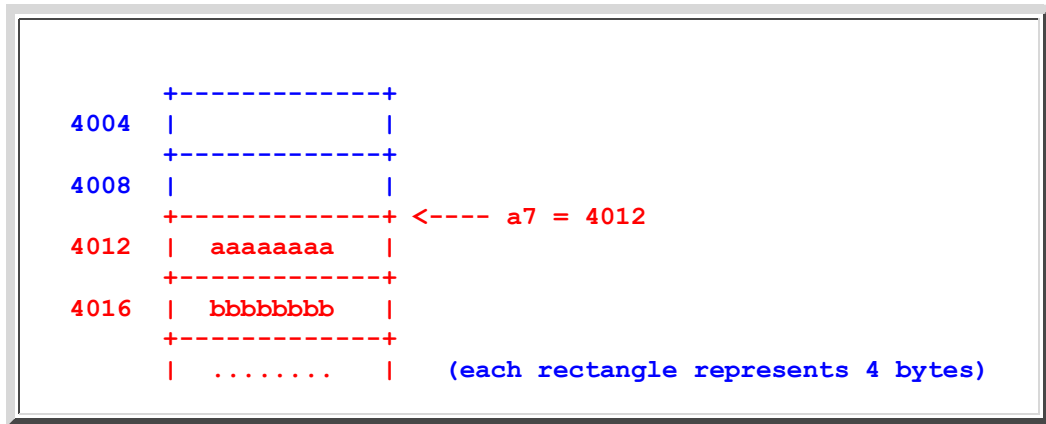
---

---

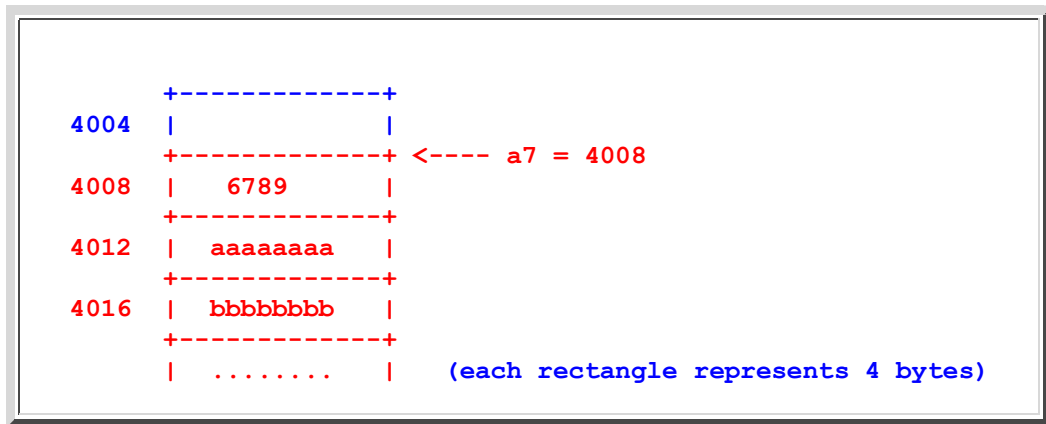
---

- **How to pass a value on the stack**

- Recall that the M68000 system stack is implemented using the address register **a7**
- Suppose the system stack is initially as follows



- After you push an **integer** value (say 6789) on the stack, the stack will look like this:



- You can achieve this result using the following 2 instruction:

```

suba.l #4, a7
move.l #6789, (a7)
  
```

- Because pushing values on the system stack is a frequently used operation, M68000 has provided a special **addressing mode** to perform the **push** operation:

```

move.l <ea>, -(a7)    is same as:  suba.l #4, a7
                        move.l <ea>, (a7)

move.w <ea>, -(a7)    is same as:  suba.l #2, a7
                        move.w <ea>, (a7)
  
```

- So when you push a **long (4 bytes)**, the stack pointer **a7** is decremented by 4.
  - But when you push a **word (2 bytes)**, the stack pointer **a7** is decremented by 2 !!!
  - This address mode is called "indirect with **pre-increment**"
- 
- 
- 

- **Order of discussion**

- The **topic** of **passing parameter** and **storing local variables** using the **system stack** is **pretty complex**
- 

- Therefore, I will **discuss** the **topic** in a **piece meal fashion**:

(I.e., I try to break down this complex topic into a number of simpler topics --- hope this will help you understand the complex topic)

1. I will **first** show you **how to pass parameters** and **store local variables** using the **system stack**

**And access** the variable using the **stack pointer**

---

2. **Finally** I will show you **how to pass parameters** and **store local variables** using the **system stack**

**And access** the variable using the **frame pointer**

The **2nd method** is the **goal** of this **piece meal treatment** of this **complex topic**:

- **Do not** use the **technique** explained in **step 1** !!!
- 

- The **goal** of the **course** is to **teach** you the **technique** in **step 2** !!!
- 
- 
-