August 24

Note Title 24-08-2011

Lab Timing: No Timing

🚩 Students can work from home.

3rd Sept: csl211.ropar@gmail.com

www.gnuarm.com

TA

Abhishek Sagar (DD) Kapil Khanna (M)

Harsh Kumar (DD) Gayathri Ananthanarayanan (Ph.D)

Assembly Programming - II

C Language:

register, automatic, static, volatile
(default) ✓

gives a suggestion to the compiler to keep a variable in a register

✓ static int a;
✓ int a;
✓ register int a;
volatile int a;

```
void foo() {

    static int a=0;
    a = a+1;
    printf("%d \n"
                , a);
}
```

You come back to the function once again

(1) a retains its value.

foo(): If I would have had (int a):

1

2 would have printed:

.

.

10

1

1

1

1

A static value is like a global.

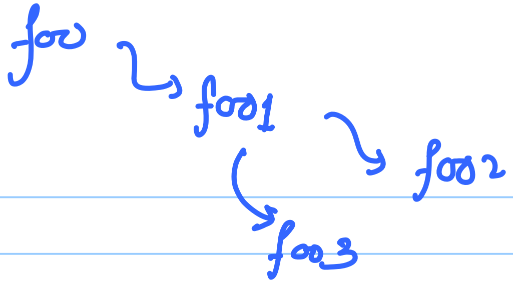
The value is saved across function invocations

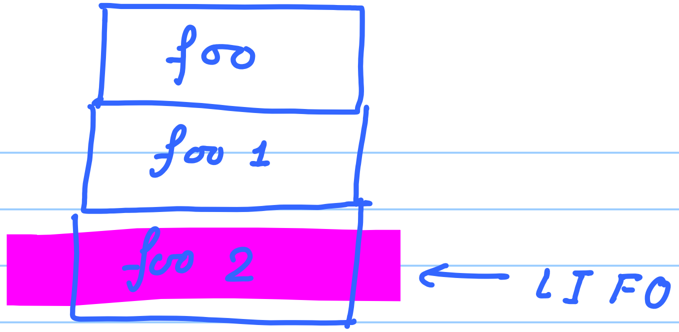
whereas,

a regular automatic variable is not saved across function invocations.

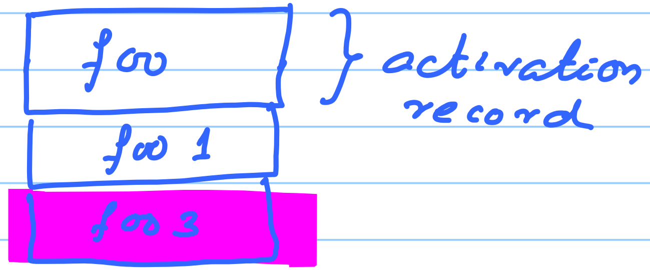
volatile: A volatile variable can be changed by entities like I/O devices that are external to a program.

Functions have a lot of automatic variables which lose their identity after the function finishes.

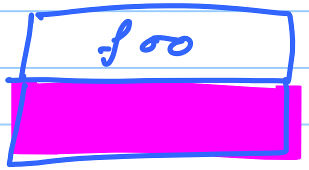


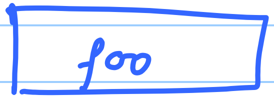


When foo2 finishes



After foo3 finishes



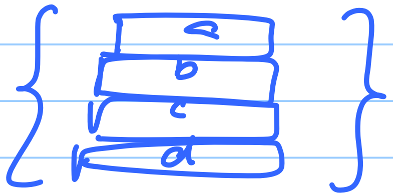


```
void    foo() {
            int a, b, c, d;
}
```

automatic temporary variables

Each "int" is 4 bytes

Total: 16 bytes of storage.



activation record of foo.

LIFO → Stack.

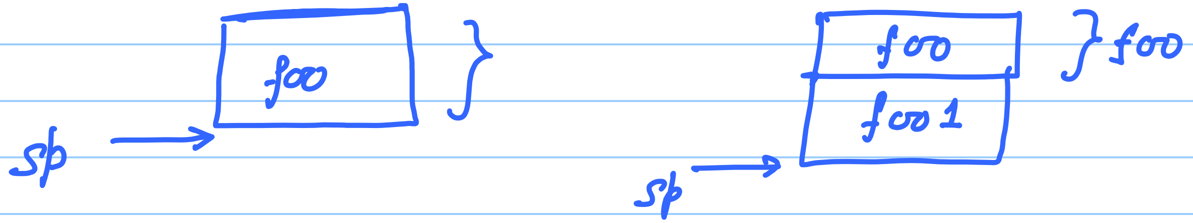
(We want to maintain a stack)

We designate an area in main memory that can act as a stack.

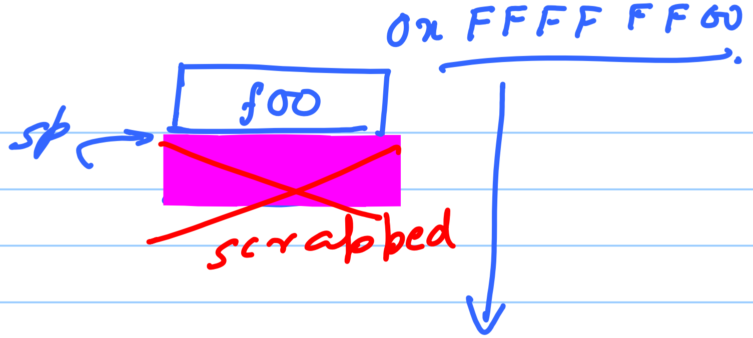
↓

downward growing
grows towards lower memory addresses.

sp → stack pointer (points to the top of the stack)
(r_{13})



When foo1 exits



sp decreases → function invocation

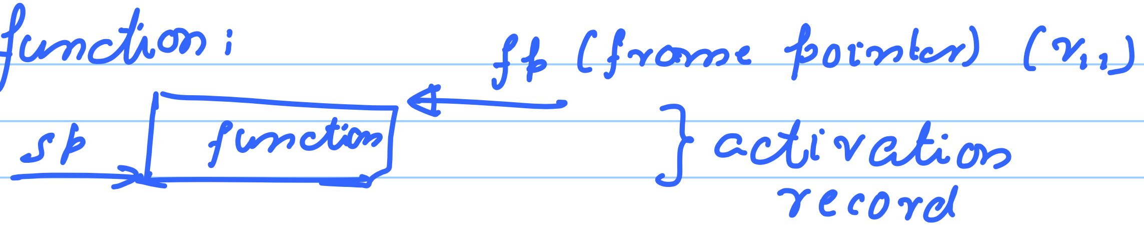
sp increases → function returns.

activation record:

a) store all the local automatic variables

b) arguments to the function

Starting a function:



When a function starts:

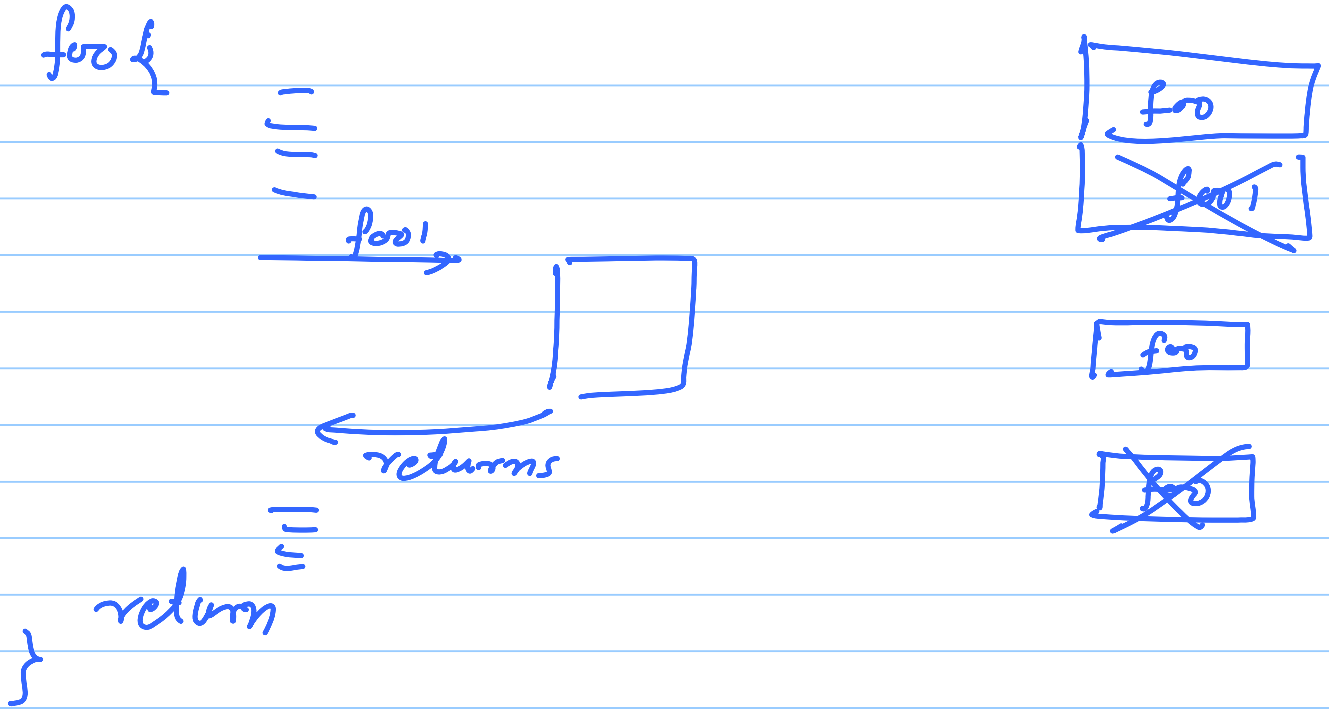
$(fp == sp)$

$sp -= $ (size of activation record)

when a function ends:

$sp = fp$.

return to the previous function (correctly set the value of the fp register)



foo1 can overwrite registers used by foo

We need to do a save & restore

foo ← caller

foo1 ← callee

Two paradigms

Caller saved

foo saves & restores all the registers that it needs.

callee saved

foo1 saves & restores all the registers that it overwrites.

Register Usage Convention in arm

not saved

- r_0, r_1 } used to pass [arguments & return values] [to/from] a function
- r_2, r_3 } temporary registers (not saved)

{ r_4 – r_{11} (saved by caller/callee)

(r_{11} ← fp)

not saved { r_{12} → ip intra-procedural scratch registers

- r_{13} → sp (saved)
- r_{14} → lr (saved)
- r_{15} → pc (not saved)

What do we know :

(1) Data Processing
Data Transfer
Control } Instructions

(2) Basic Instruction Types

ADD, SUB, LDR, STR, B, BL

(3) Stack & registers

(4) In the Tut. session
written a program to
compute a factorial.

Next Step:

2
1) Conditional Instructions
2) Complex Addressing Modes
3) Instruction Format

Chapters

1
One lecture

3
1) ADDITION
ii) Multiplication / Division
iii) Floating Point.

Mid Term

Last week
of Sept

(8 classes
left)

4 [Processor Design (Half of it)

Sept 3rd: Dead line for 1st HW.

Sept 11th: Dead line for HW 2.

Sept 21st: HW 3