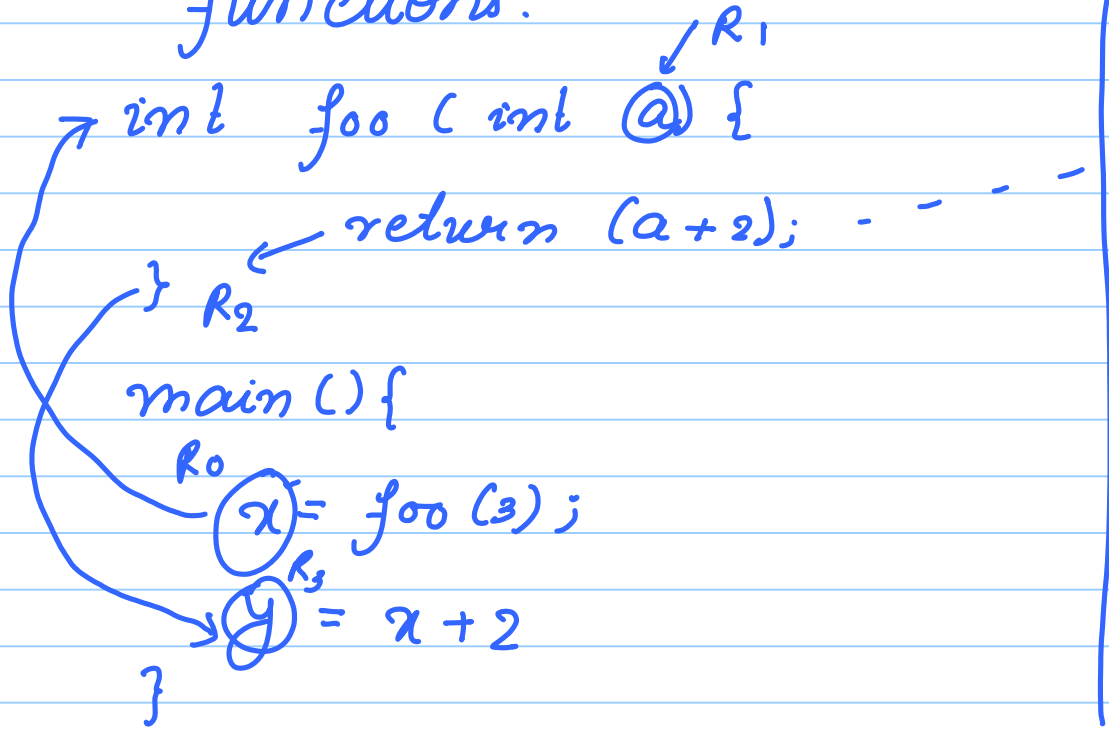
Aug - 19

Note Title 09-08-2012

Functions.

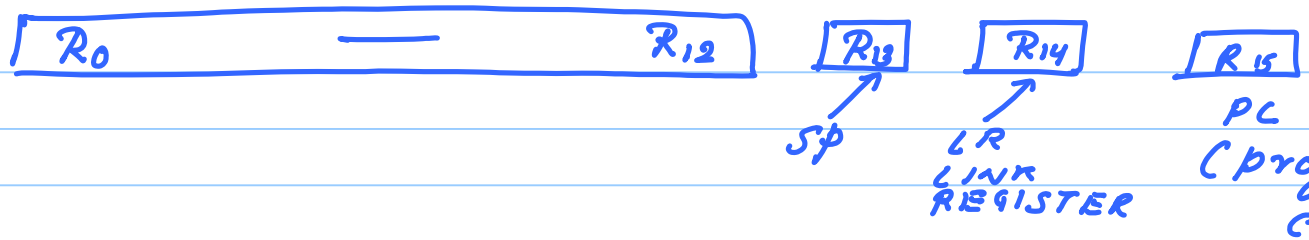


Assembly

```
.foo:
- - - ADD R2, R1, #2      { R2 = return value, R1 = arg }
      MOV PC, LR          { return }

main:
      MOV R1, #3
      BL .foo             { LR = PC + 4 }
      MOV R0, R2          [x = return value]
      ADD R3, R0, #2      [y = x + 2]
```

Register Set



Simple Cases: Arguments and return values are passed through registers.

Stack Pointer

Consider a function in C

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

} local variables.

```
    float x,y;
    :
    :
    ;
}
```

Every function in C has local variables.

Where are they stored?

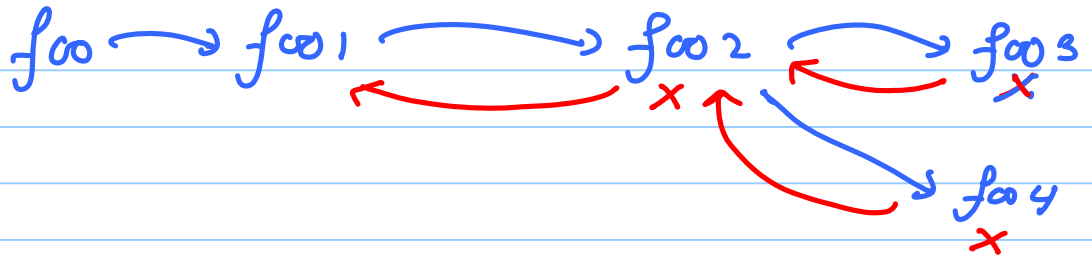
Until now: all local variables are stored in registers.

We only have 13 registers

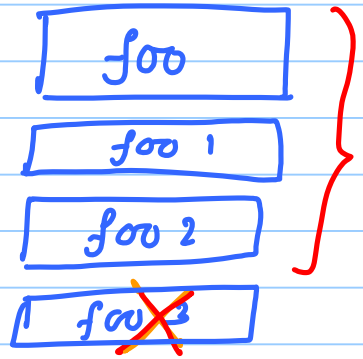
What if,

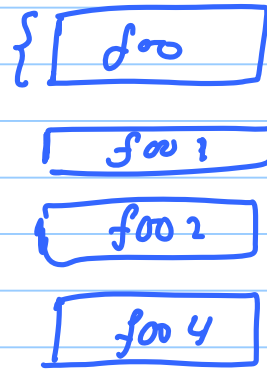
variables > # registers

Storage : Keep extra variables.



Pattern:



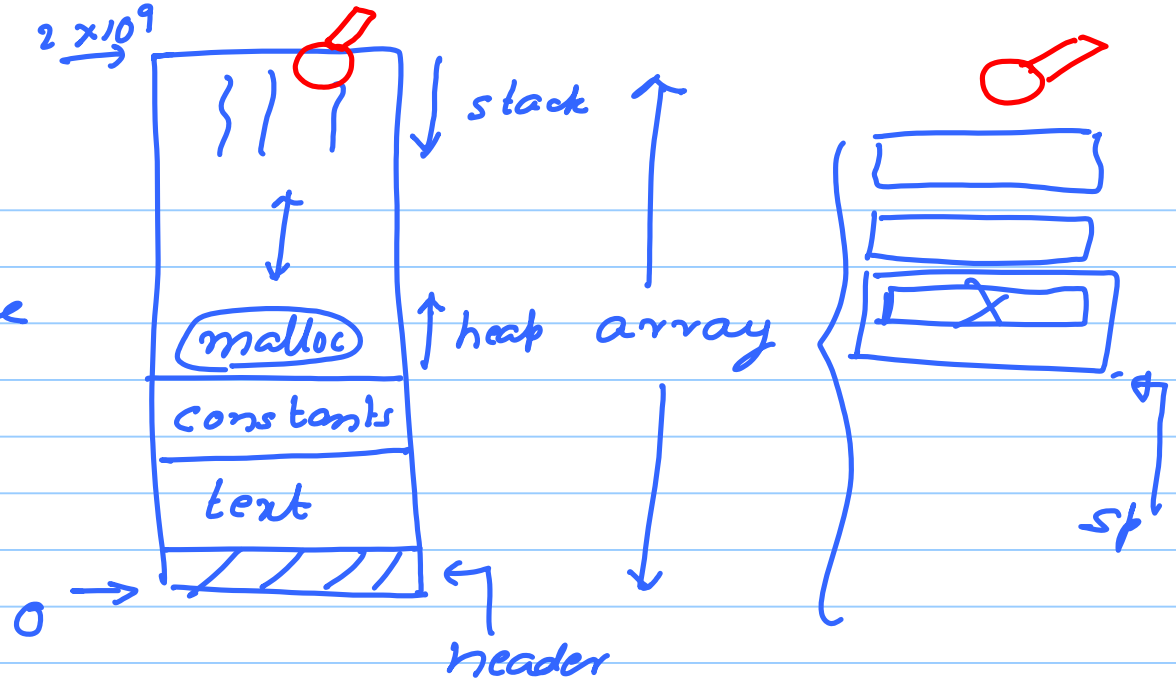


Activation Record

Behavior:
{ LIFO
{ Last In First Out
Stack.

How does this work:

Process → Running Instance of a program



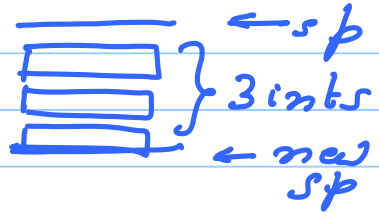
text → Instructions.

stack :
sp → points to the head of the stack

Stack

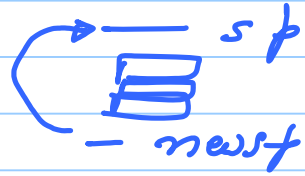
Create space on the stack

SUB sp, sp, #-12



Reclaim space on the stack

ADD sp, sp, #12



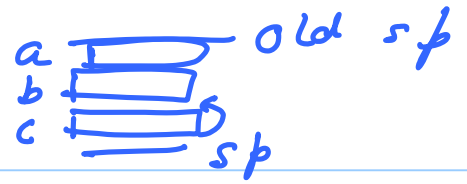

```
int foo() {                    func() {
    int a, b, c;                   ; R4 =
    a = 3; b = 4;                  ; R5 =
    func();                    }
    c = a + b;
}
```

R4 R5 — a, b are gone

R6 — c

• foo:

SUB sp, sp, #-12 [Create Space]



sp → c
sp+4 → b
sp+8 → a

```
MOV   R4, #3
STR   R4, [sp, #8]
MOV   R5, #4
STR   R5, [sp, #4]
bl    .func
```

```
LDR   R4, [sp, #8]
LDR   R5, [sp, #4]
ADD   R6, R4, R5
ADD   sp, sp, #12
(RETURN) MOV  PC, LR
```

Why not save variables in registers all the time!

✓ 1) # vars reqd. > # available regs

– 2) Calling a function, which can potentially modify registers.

[stack ⟷ temporary storage]
[SP] points to the head of the stack

function call:

Caller. Saved convention
⟶ A caller saves all the ① registers that it requires, ② and those that can be potentially overwritten (① & ② need to hold)

[saves on the stack]

Callee saved convention.

[R_1, R_2, R_3

beginning
save R_1, R_2, R_3 on the stack

Potential Exam Question

When do you use → caller saved
→ callee saved ?

Function {
label
[Create space on the stack]
[Save registers if reqd.]
{ Perform your operation
[Restore regs if required]
[Reclaim the space]
[return]

end restore R_1, R_2, R_3

Covered Uptil Now : { Loops, IF-ELSE, Arrays, Functions }

Tutorial : 1) ARM doubts

2) Solve some problems : { 1) Factorial. 2) Prime 3) GCD 4) Palindrome ×5) Function Pointers 6) LCM }

Multiplication

MUL R_1, R_2, R_3

$R_1 = R_2 \times R_3$

Division.

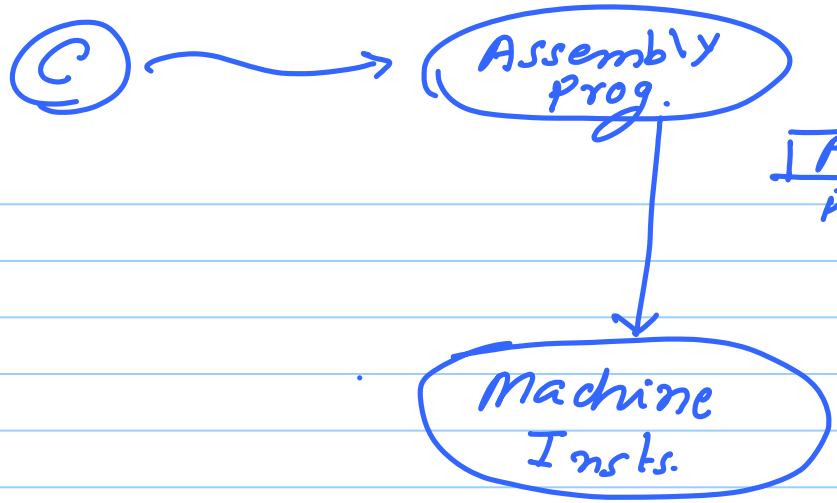
div (R_0, R_1)

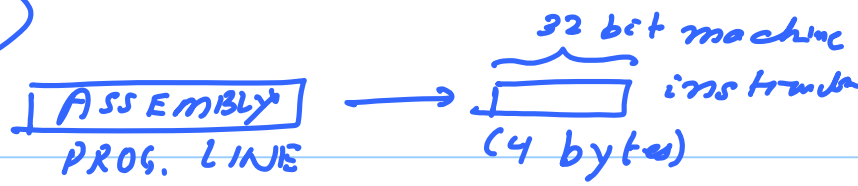
R_0 / R_1

$R_2 \rightarrow$ Quotient

$R_3 \rightarrow$ Remainder.

Next class





Next Week