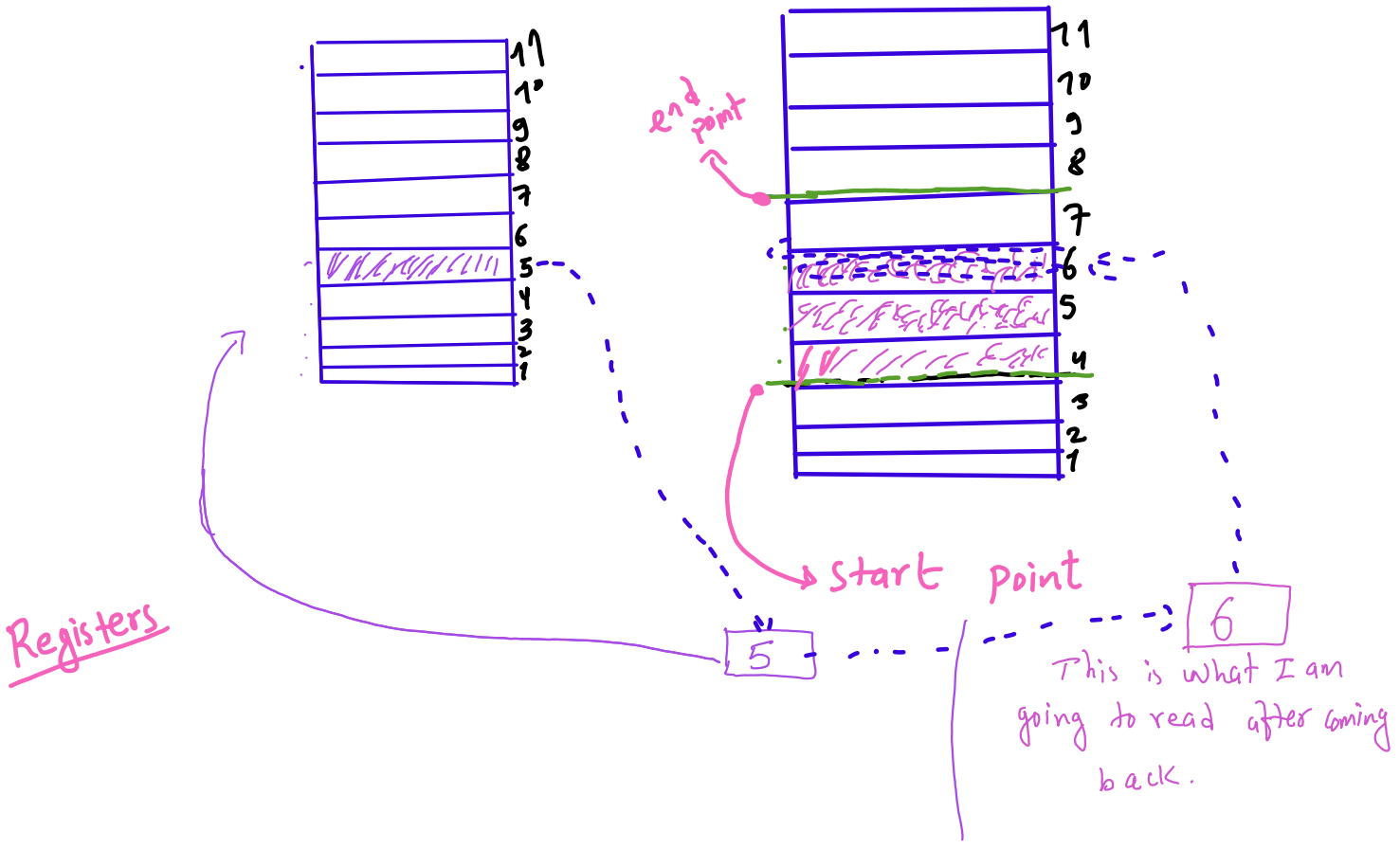
Nov 21, 2024
Lanester, France.



cCodes > learning-gcc > C stack-frame-add.c > add(int, int)

```c
#include <stdio.h>
int add(int a, int b){
    int c = a + b;
    return c;
}
int main(){
    int a = 5;
    int b = 5;
    int c = add(a,b);
    printf("%d",c);
    return 0;
}
```

cCodes > learning-gcc > C stack-frame-add.c > add(int, int)

```c
#include <stdio.h>
int add(int a, int b){
    int c = a + b;
    return c;
}
int main(){
    int a = 5;
    int b = 5;
    int c = add(a,b);
    printf("%d",c);
    return 0;
}
```

Cmd <dest^n> <source>

PUSH BP

② MOV BP, SP

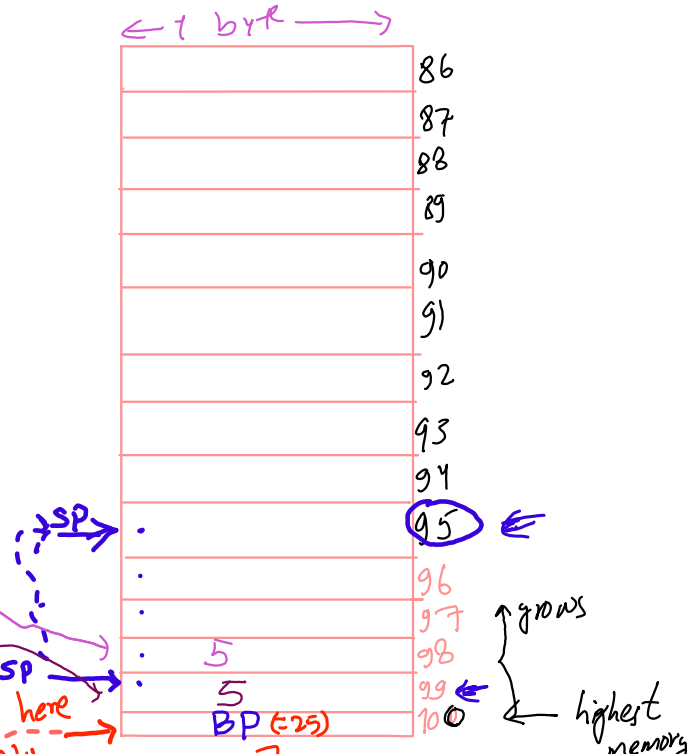
③ SUB SP, 4

MOV [BP], 5

MOV [BP-1], 5

MOV A, [BP]

MOV B, [BP-1]



memory address grows from higher addresses to lower address.

BP --> pointer that is going to be start position of the function that is going to be executed right now

SP --> the last used position in the stack, where is the top of the stack after so many push and pops?

Before pushing the BP here SP is already pointing to this address.

after ①

0 95

0 99

~~100~~ SP

~~25~~ 99 BP

(after 3), ahileko awasthama BP is holding 99 ra SP is holding 95 so the difference (BP-SP) region is going to be the stack frame that has been made only and exclusively for our function main

① After pushing BP to the TOS, SP is going to be 99 so, we have to decrement it by 1. thus, after ① SP holds 99.

② हामीले अहिले लाई पहिलोको BP त stack मा माथि सेभ्ड, त्यसैले BP ले 25 hold गरिरहनु पर्दैन। त्यो र, we have to copy current SP to the BP. hence, 2nd Instn: MOV BP, SP.

after ②, BP → 99

③ अब BP मा current SP राख्नेपछि, we need to make space for the variables, so that those spaces will not be used by system/interrupt or whatever except us or our program.

↳ we do this by subtracting SP by no. of spaces (i.e. eg. यहाँ we are taking 4 spaces

after this, SP holds 95

④ हामी, हाम्रो SP त माथी (memory को) ठुप्पामा गएर बसेको छ, र BP ले हाम्रो function को लागि बनाएको stack को start address (i.e. 99) लिएर बसेको छ। So, अब यो BP-चाहीं change नहुने हुँदा हामी यसको reference लिएर variable लाई stack मा राख्नेछौं।

∴ MOV [BP], 5

⑤ र, अब b को लागी ? We can't use [BP] as 99 वाला mem. address त पहिले नै full यसको त्यसैले What we gonna do is, [BP-1] which gives us 98th address

∴ MOV [BP-1], 5

(6) अब we can't do math operation in the stack, math operation को लागि ती value हरू CPU को registers मा हुनुपर्छ।

e.g. MOV A, [ebp] → move whatever in the address of ebp in the register A.

MOV B, [bp-1]

(7) अब, we gonna use esi, edi registers to send the values to the functions.

Let's call add

मानौं कि, हाम्रो program (main) execute गर्दै जादा add लाई call गर्ने बाटोमा पुग्यो अब के हुन्छ?

I'm using line no. as instruction address in IP.

So, 8 यदि when IP points to 9, it is going to call "add" यो र कुरा बिर्नुयो ती भाई।

अब मेरो add को लागि same steps हुनेवाला छ जुन main को लागि भयो।

So, we gonna push current BP (i.e. 99) to the current TOS (i.e. 94), & SP is decremented by 4, so it points to 90 now.

Then, we copy SP to BP,

PUSH BP ;
MOV BP, SP
SUB SP, 4
MOV [bp], SI
MOV [bp-1], DI

[SI = 5
DI = 5]

MOV A, [bp]
MOV B, [bp-1]
ADD A, B
MOV [bp-2], A

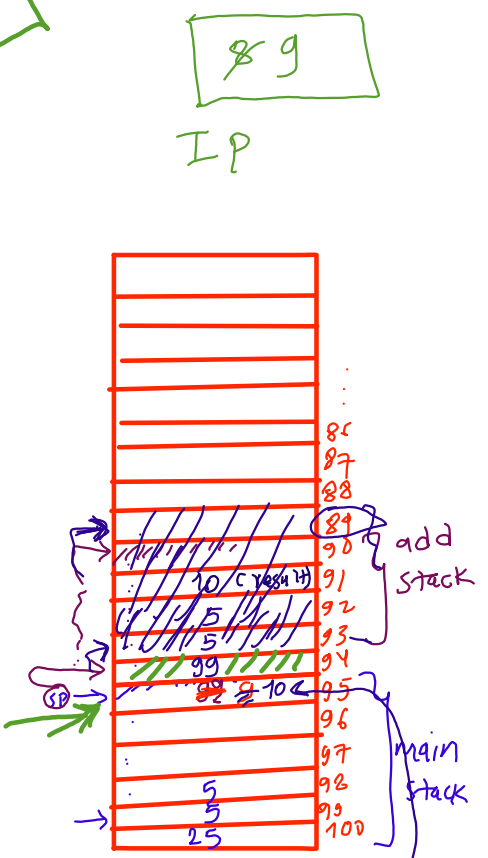
8g

~~94~~ ~~99~~ SP

~~99~~ ~~94~~ ~~93~~ BP

MOV A, [bp-2]
MOV SP, BP

POP B



10 — by call instruction.

IP

add को लागि stack बनाउनुआघि call ले current IP को value लगेर TOS मा राख्छ, त्यसपछि TOS 1 ले decremented भएर 94 मा point गर्छ, र अलि हाम्रो stack frame बनाउने काम सुरु हुन्छ।

Parameters is sent via SI & DI, त्यसैले हामीले SI र DI बाट हाम्रो Parameter bp र bp-1 मा राख्यौं।

र to perform arithmetic operation हाम्रो values हरू CPU को register मा हुन जरूरी छ। त्यसैले केटी bp location बाट A र B मा लाउन पर्यो।

MOV A, [bp]. To return result to main funct^n, we simply store result from [bp-2] to a register

How to delete stack frame & go to calling function?

Leave

↳ undo the changes we have made to the stack.

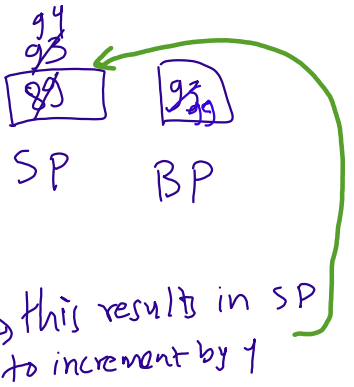
SP+4=93 or

leave ↙ does this.

MOV SP, BP

POP BP [content of TOS is going to be in BP i.e. again 99 in BP]

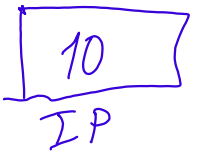
ret



this results in SP to increment by 4

ret

ले 2टा काम गर्छ,

whatever in TOS, त्यो copy गरेर IP लाई दिन्छ।

This way, we return to main again.