CFA for Dummies

Slides on the Canonical Frames Address

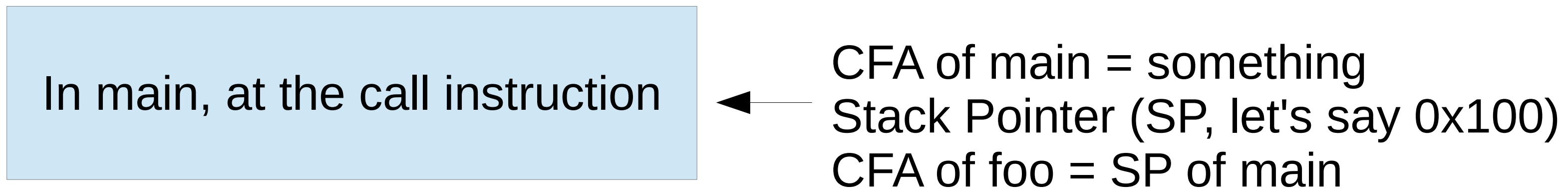
Author : Xavier Pouyollon with the help of
- Eugene Tarassov
- Francis Lynch
- Wikipedia : http://stackoverflow.com/questions/7534420/gas-explanation-of-cfi-def-cfa-offset

CFA Definition

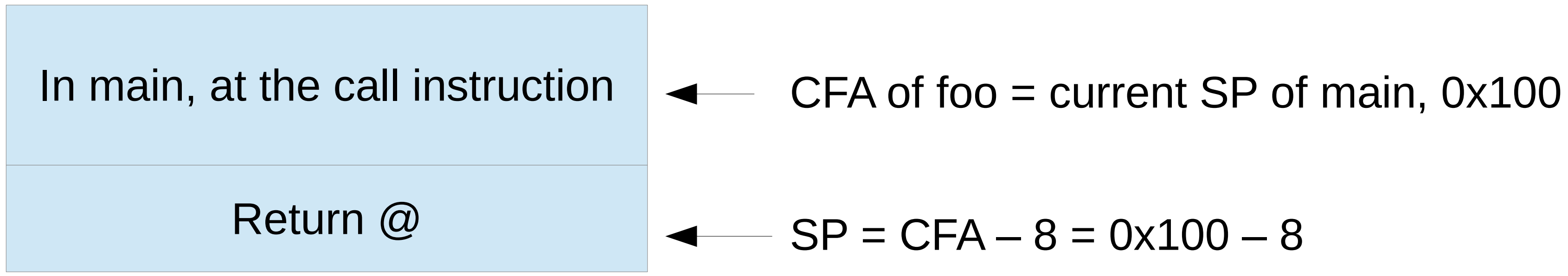
- CFA means Canonical Frame Address
  - Typically, the CFA is defined to be the value of the stack pointer at the call site in the previous frame (which may be different from its value on entry to the current frame) (Dwarf Specs)
  - Typically, the CFA is defined to be the value of the stack pointer at the call site. (Eugene Tarassov)
  - Whatever calls the function, stack pointer at the moment of the call is CFA of the <u>**callee**</u>
  - **CFA, by definition, should be same at any instruction of a function.**
  - **However rules to compute CFA can be different at different places in the function.**

Pentium Sample

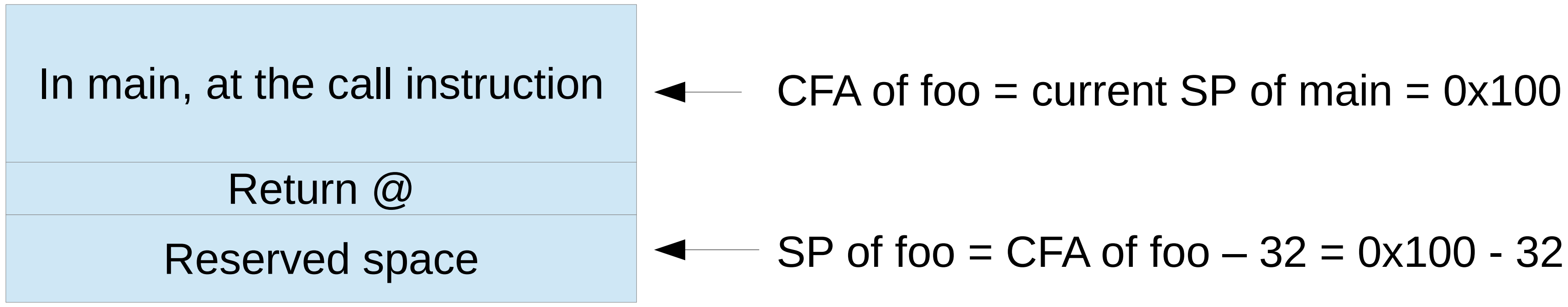
- Main() calling foo()


In main, at the call instruction
CFA of main = something
Stack Pointer (SP, let's say 0x100)
CFA of foo = SP of main

- At the first assembly instruction of foo()


In main, at the call instruction
Return @
CFA of foo = current SP of main, 0x100
SP = CFA – 8 = 0x100 – 8

- After prolog been executed


In main, at the call instruction
Return @
Reserved space
CFA of foo = current SP of main = 0x100
SP of foo = CFA of foo – 32 = 0x100 - 32

TCF stepping-over stragegy

- Before the step-over:
  - Record CFA of caller
  - Compute the C-line range
  - Step until out-of-range
- When out-of-range:
  - Compute the current CFA
    - Either by using debug_infos or processor plugin
  - CFA != CFA of caller ?
    - Yes, we are in another function, compute return @, put a breakpoint there and continue
    - No, we are in the same function, step-over done.

Case study

- RH850
  - Jarl is the call instruction
  - Jarl does not change SP
    - CFA of callee == SP of caller
    - CFA always points to the value of the stack pointer in the previous frame
    - Can it be a problem ?
      - Usually not, if caller do “prepare” a frame
      - Non-leaf function should always prepare a frame.

Case Study

- Caller function:

```
0010003e:   prepare 2,20
```

→ CFA of caller is something. Let's say CFA = 0x100. SP=0x100 (see previous slide why)

→ Prepare has changed SP. SP is now 0xEC

```
00100042:   jarl    100032,r31
```

→ CFA value of of caller does NOT change. However, it's computation rule DO change CFA = SP + 0x14.

CFA of caller is 0x100
CFA of callee is SP at the call site (jarl) so 0xEC

Case study

- It's possible to have a leaf function on a target like PPC or RH850 that does not change the stack pointer.
- In that case, its CFA is the stack pointer from the calling function, but since that calling function cannot be a leaf function, it will have saved at least its return address on the stack, and therefore its CFA will not be equal to the stack pointer, and so it will still be true that the CFA in the leaf function is not the same as the calling function's CFA.

Case study

- It's possible that the caller of the leaf function might be optimized to save the return address in another register rather than storing it on the stack. In that case:
  - The CFA logic would not be correct because the stack pointer would be the same in both functions.
  - This probably shouldn't happen for debug compiles, but an aggressive optimizer might do this.
- You could handle this case by adding a test to see if (assuming you have .debug_frame info) the return address is at the same location as the function where you started the step-over. For the case I have just described, the frame info would indicate the return address in some other register when the call is made but in the leaf function it would show that it is still in the link register.
- So the stepping logic would become: if not within the source line range, check the CFA values; if equal, check the RTN locations; if both are equal you are back in the original function. If either one is not the same, you would need to continue stepping.

Case Study

- When the jarl is executed
  - CFA of callee is different from CFA of caller

    (0xEC != 0x100)
  - Stepping-logic will work

- Conclusion:
  - Non-leaf function should always prepare a frame.
    - If not, on arch like RH850, CFA's callee == CFA's caller and this would break step-over logic.