Script **generated by TTT**

Title: Petter: Compiler Construction (02.07.2020) - 57: Caller-Callee Interaction

Date: Fri Jul 03 13:33:20 CEST 2020

Duration: 12:29 min

Pages: 6

Split of Obligations

Definition

Let f be the current function that calls a function g.

- f is dubbed *caller*
- g is dubbed *callee*

The code for managing function calls has to be split between caller and callee.
This split cannot be done arbitrarily since some information is only known in that caller or only in the callee.

Observation:

The space requirement for parameters is only know by the caller:
Example: `printf`

Principle of Function Call and Return

actions taken on entering g:

1. compute the start address of g
2. compute actual parameters in globals
3. backup of caller-save registers
4. backup of FP
5. set the new FP
6. back up of PC and jump to the beginning of g
7. copy actual params to locals

$\}$ saveloc
$\}$ mark
$\}$ call
$\}$ are in f
$\}$ … $\}$ is in g

actions taken on leaving g:

1. compute the result into R_0
2. restore FP, SP
3. return to the call site in f, that is, restore PC
4. restore the caller-save registers

$\}$ return $\}$ are in g
$\}$ restoreloc $\}$ is in f

Managing Registers during Function Calls

The two register sets (global and local) are used as follows:

- automatic variables live in *local* registers R_i
- intermediate results also live in *local* registers R_i
- parameters live in *global* registers R_i (with $i \leq 0$)
- global variables:

Managing Registers during Function Calls

The two register sets (global and local) are used as follows:

- automatic variables live in *local* registers R_i
- intermediate results also live in *local* registers R_i
- parameters live in *global* registers R_i (with $i \leq 0$)
- global variables: let's suppose there are none

convention:

Managing Registers during Function Calls

The two register sets (global and local) are used as follows:

- automatic variables live in *local* registers R_i
- intermediate results also live in *local* registers R_i
- parameters live in *global* registers R_i (with $i \leq 0$)
- global variables: let's suppose there are none

convention:

- the ith argument of a function is passed in register $\boxed{R_{-i}}$
- the result of a function is stored in $\boxed{R_0}$
- local registers are saved before calling a function

Managing Registers during Function Calls

The two register sets (global and local) are used as follows:

- automatic variables live in *local* registers R_i
- intermediate results also live in *local* registers R_i
- parameters live in *global* registers R_i (with $i \leq 0$)
- global variables: let's suppose there are none

convention:

- the ith argument of a function is passed in register R_{-i}
- the result of a function is stored in R_0
- local registers are saved before calling a function

Definition

Let f be a function that calls g. A register R_i is called

- *caller-saved* if f backs up R_i and g may overwrite it
- *callee-saved* if f does not back up R_i, and g must restore it before returning