Functions

Computer Organisation & Program Execution 2021

Uwe R. Zimmer - The Australian National University

3

Functions

(Greatness from ...) Small beginnings

plus_1 :: (Num a) => a -> a
plus_1 x = x + 1

int plus1 (int x) {
    return x + 1;
}

function Plus_1 (x : Integer) return Integer is (x + 1);

def plus1 (x):
    return x + 1;

pure function plus_1 (x)
    int, intent (in) :: x  int              :: plus_1
    plus_1 = x + 1;
end function;

function Plus_1 (x : integer) : integer;
begin  Plus_1 := x + 1; end;

References for this chapter

[Patterson17]
David A. Patterson & John L. Hennessy
Computer Organization and Design – The Hardware/Software Interface
Chapter 2 "Instructions: Language of the Computer"
ARM edition, Morgan Kaufmann 2017

3
(Greatness from …) Small beginnings

plus_2 : (Nat x) => a => a
plus_2 x = plus_1 (plus_1 x)

function Plus_2 (x : Integer) return Integer is (Plus_1 (Plus_1 (x)));

mov r0, #1
bl Plus_2
mov r3, r0
add r0, #1
bx lr

what is the value of G in each case?

Fib:

Fib_fact x = (fib x) + (fact x)

unsigned int fibFact (unsigned int x) {
    int, intent (in) :: x
    int              :: fib_fact
    fib_fact = fib (x) + fact (x);
    return fib_fact;
}

bl Fib
add r0, #1
bx lr

where does this lead us?

N National University
What happens to our parameters \( x \) during the function?
Functions

Components / phases of a function call:
- Values (parameters) to be passed to a function.
- Local variables inside a function.
- Values (results) to be returned from a function.

So far we:
- pushed parameter values in registers (r0-r3).
- called the function (store the return address and jump to the beginning of the function).
- pushed the return address, the previous stack frame and used registers (r0-r3).
- created a new stack frame (and addressed all local variables relative to this).
- grew the stack such that it can hold the local variables.
- did the calculations/operations based on the local variables and scratch registers.
- pushed return values in registers (r0-r3).
- restored the stack pointer (and thus de-allocated all local variables).
- jumped back to the next instruction after the original function call.
- restored the return values found in r0-r3.

ARM architecture calling practice
- r0-r3 are used for parameters.
- r0-r4 are used for return values.
- r0-r4 are not expected to be intact after a function call ...
- all other registers are expected to be intact!

- If those registers do not suffice, additional parameters and results are passed via the stack.

Parameter passing
Call by ...
- by value
- by reference (mutable)
- by reference (immutable)

Information flow
- in
- out

Analysis by structure
- by value
- by reference
Information flow

| Copy | Reference
|------|-----------
| In   | Parameter becomes a constant inside the function or is copied into an allocated variable. |
| Out  | Calling function expects the parameter value to appear in a specific space at return. |
| In & Out | Parameter becomes a constant inside the function or is copied into a local variable. |

By value

- Parameter is copied to a local variable and copied back at return.

By reference (mutable)

- Function can read and write at any time. Outside code shall not write.

By result

- Parameter is copied to a local variable and copied back at return.

By value result

- Parameter is copied to a local variable and copied back at return.

Functions

C

Full control over the parameter modes. "by value" parameters are local variables.

By copy

- No write access is allowed while the function runs (also from outside the function).

By reference

- Parameter becomes a constant inside the function or is copied into a local variable.
- No write access is allowed while the function runs (also from outside the function).

Haskell

| Copy | Reference
|------|-----------
| In   | Parameter becomes a constant inside the function or is copied into an allocated variable. |
| Out  | Calling function expects the parameter value to appear in a specific space at return. |
| In & Out | Parameter becomes a constant inside the function or is copied into a local variable. |

By value

- Parameter is copied to a local variable and copied back at return.

By reference (immutable)

- No write access is allowed while the function runs (also from outside the function).

By reference

- Parameter becomes a constant inside the function or is copied into an allocated variable.

Python

| Copy | Reference
|------|-----------
| In   | Parameter becomes a constant inside the function or is copied into an allocated variable. |
| Out  | Calling function expects the parameter value to appear in a specific space at return. |
| In & Out | Parameter becomes a constant inside the function or is copied into a local variable. |

By value

- Parameter is copied to a local variable and copied back at return.

By reference (mutable)

- No write access is allowed while the function runs (also from outside the function).

Assembly

| Copy | Reference
|------|-----------
| In   | Parameter becomes a constant inside the function or is copied into an allocated variable. |
| Out  | Calling function expects the parameter value to appear in a specific space at return. |
| In & Out | Parameter becomes a constant inside the function or is copied into a local variable. |

By value

- Parameter is copied to a local variable and copied back at return.

By reference (mutable)

- No write access is allowed while the function runs (also from outside the function).

By result

- Parameter is copied to a local variable and copied back at return.

By value result

- Parameter is copied to a local variable and copied back at return.

By reference (mutable)

- No write access is allowed while the function runs (also from outside the function).


One-way and by-copy

These are side-effect-free and hence the resulting scenarios are easy to analyze. Copying large data structures might be time consuming or infeasible. Values can be passed in registers.

<table>
<thead>
<tr>
<th>Information flow</th>
<th>Access</th>
<th>by copy</th>
<th>by reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>Parameter becomes a constant inside the function or is copied into a local variable.</td>
<td>by value</td>
<td>by reference (immutable)</td>
</tr>
<tr>
<td>out</td>
<td>Calling function expects the parameter value to appear in a specific space at return.</td>
<td>by result</td>
<td>by reference (immutable)</td>
</tr>
<tr>
<td>by result</td>
<td>Function can read and write at any time.</td>
<td>by reference (immutable)</td>
<td></td>
</tr>
<tr>
<td>by value result</td>
<td>Parameter is copied to a local variable and copied back at return.</td>
<td>by value result</td>
<td></td>
</tr>
</tbody>
</table>

Two-way and by-copy

Still side-effect-free within the function (but not on the outside). Potentially more convenient as memory space can be reused. No data has to be replicated. Values can be passed in registers.

<table>
<thead>
<tr>
<th>Information flow</th>
<th>Access</th>
<th>by copy</th>
<th>by reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>Parameter becomes a constant inside the function or is copied into a local variable.</td>
<td>by value</td>
<td>by reference (immutable)</td>
</tr>
<tr>
<td>out</td>
<td>Calling function expects the parameter value to appear in a specific space at return.</td>
<td>by result</td>
<td>by reference (immutable)</td>
</tr>
<tr>
<td>by result</td>
<td>Function can read and write at any time.</td>
<td>by reference (immutable)</td>
<td></td>
</tr>
<tr>
<td>by value result</td>
<td>Parameter is copied to a local variable and copied back at return.</td>
<td>by value result</td>
<td></td>
</tr>
</tbody>
</table>

One-way-Out and by-reference

Side-effect-free, i.e. no memory is allocated on return. Cannot be enforced on assembly level (requires compiler). Values have to be passed in memory.

<table>
<thead>
<tr>
<th>Information flow</th>
<th>Access</th>
<th>by copy</th>
<th>by reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>Parameter becomes a constant inside the function or is copied into a local variable.</td>
<td>by value</td>
<td>by reference (immutable)</td>
</tr>
<tr>
<td>out</td>
<td>Calling function expects the parameter value to appear in a specific space at return.</td>
<td>by result</td>
<td>by reference (immutable)</td>
</tr>
<tr>
<td>by result</td>
<td>Function can read and write at any time.</td>
<td>by reference (immutable)</td>
<td></td>
</tr>
<tr>
<td>by value result</td>
<td>Parameter is copied to a local variable and copied back at return.</td>
<td>by value result</td>
<td></td>
</tr>
</tbody>
</table>

One-way-In and by-reference

Side-effect-free – cannot be enforced on assembly level (requires compiler). Values have to be passed in memory.

<table>
<thead>
<tr>
<th>Information flow</th>
<th>Access</th>
<th>by copy</th>
<th>by reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>Parameter becomes a constant inside the function or is copied into a local variable.</td>
<td>by value</td>
<td>by reference (immutable)</td>
</tr>
<tr>
<td>out</td>
<td>Calling function expects the parameter value to appear in a specific space at return.</td>
<td>by result</td>
<td>by reference (immutable)</td>
</tr>
<tr>
<td>by result</td>
<td>Function can read and write at any time.</td>
<td>by reference (immutable)</td>
<td></td>
</tr>
<tr>
<td>by value result</td>
<td>Parameter is copied to a local variable and copied back at return.</td>
<td>by value result</td>
<td></td>
</tr>
</tbody>
</table>

Generic Stack-frame

Let there be some (global) data on the stack. Stack-frame (SB) is a static address, always pointing to the top of the stack. Stack Pointer (SP) points to the current top of the stack. Global variables can also be stored if required.
Functions

Generic Stack-Frame

The current code prepared to call a function:
- Push parameters on the stack.
- Work any data structures (at the stack overflow)
- Parameter passing mode.

Types, storage structures and parameter passing modes have to be agreed upon between caller and callee.

- Some languages will not have a context by default, like C or Assembly.
- If a context is required, it will be provided through the surrounding function or the hosting object.

Functions (in programming languages) have a context.
E.g. the surrounding function or the hosting object.

- The caller knows this context and provides it.
- This builds a linear chain of calls through the stack.

The context for
- The static and dynamic link might be identical in some cases.
- The static and dynamic link for function case both functions.

Dynamic link (prior frame)
- The caller of function has function c
- Or the called function has function c

Static vs. dynamic links
- Function a (x : Integer) return Integer is
  function b (y : Integer) return Integer is (x + y);
  function c (z : Integer) return Integer is (b (z));
  begin
  return c (x);
end a;
- a ': Integer -> Integer
  a x = c x

(= C expands the C standard and provides it though)

Function a (x : Integer) return Integer is
  function a (x : Integer) return Integer is (x + y);
  function b (y : Integer) return Integer is (x + y);
  function c (z : Integer) return Integer is (b (z));
  begin
  return c (x);
end a;
- a :: Integer -> Integer
  a x = c x

Some languages will not have a context by default, like C or Assembly.

Functions (in programming languages) have a context.
- E.g. the surrounding function or the hosting object.

E.g. the surrounding function or the hosting object.

- The caller knows this context and provides it.
- This builds a linear chain of calls through the stack.

The context for
- The static and dynamic link might be identical in some cases.
- The static and dynamic link for function case both functions.

Dynamic link (prior frame)
- The caller of function has function c
- Or the called function has function c

Static vs. dynamic links
- Function a (x : Integer) return Integer is
  function b (y : Integer) return Integer is (x + y);
  function c (z : Integer) return Integer is (b (z));
  begin
  return c (x);
end a;
- a :: Integer -> Integer
  a x = c x

- a :: Integer -> Integer
  a x = c x

= C expands the C standard and provides it though.

Some languages will not have a context by default, like C or Assembly.

- The context for
- The static and dynamic link might be identical in some cases.
- The static and dynamic link for function case both functions.

Dynamic link (prior frame)
- The caller of function has function c
- Or the called function has function c

Static vs. dynamic links
- Function a (x : Integer) return Integer is
  function b (y : Integer) return Integer is (x + y);
  function c (z : Integer) return Integer is (b (z));
  begin
  return c (x);
end a;
- a :: Integer -> Integer
  a x = c x

- a :: Integer -> Integer
  a x = c x

= C expands the C standard and provides it though.

Some languages will not have a context by default, like C or Assembly.

- The context for
- The static and dynamic link might be identical in some cases.
- The static and dynamic link for function case both functions.

Dynamic link (prior frame)
- The caller of function has function c
- Or the called function has function c

Static vs. dynamic links
- Function a (x : Integer) return Integer is
  function b (y : Integer) return Integer is (x + y);
  function c (z : Integer) return Integer is (b (z));
  begin
  return c (x);
end a;
- a :: Integer -> Integer
  a x = c x

- a :: Integer -> Integer
  a x = c x

= C expands the C standard and provides it though.

Some languages will not have a context by default, like C or Assembly.

- The context for
- The static and dynamic link might be identical in some cases.
- The static and dynamic link for function case both functions.

Dynamic link (prior frame)
- The caller of function has function c
- Or the called function has function c

Static vs. dynamic links
- Function a (x : Integer) return Integer is
  function b (y : Integer) return Integer is (x + y);
  function c (z : Integer) return Integer is (b (z));
  begin
  return c (x);
end a;
- a :: Integer -> Integer
  a x = c x

- a :: Integer -> Integer
  a x = c x

= C expands the C standard and provides it though.

Some languages will not have a context by default, like C or Assembly.

- The context for
- The static and dynamic link might be identical in some cases.
- The static and dynamic link for function case both functions.

Dynamic link (prior frame)
- The caller of function has function c
- Or the called function has function c

Static vs. dynamic links
- Function a (x : Integer) return Integer is
  function b (y : Integer) return Integer is (x + y);
  function c (z : Integer) return Integer is (b (z));
  begin
  return c (x);
end a;
- a :: Integer -> Integer
  a x = c x

- a :: Integer -> Integer
  a x = c x

= C expands the C standard and provides it though.

Some languages will not have a context by default, like C or Assembly.

- The context for
- The static and dynamic link might be identical in some cases.
- The static and dynamic link for function case both functions.

Dynamic link (prior frame)
- The caller of function has function c
- Or the called function has function c

Static vs. dynamic links
- Function a (x : Integer) return Integer is
  function b (y : Integer) return Integer is (x + y);
  function c (z : Integer) return Integer is (b (z));
  begin
  return c (x);
end a;
- a :: Integer -> Integer
  a x = c x

- a :: Integer -> Integer
  a x = c x

= C expands the C standard and provides it though.

Some languages will not have a context by default, like C or Assembly.

- The context for
- The static and dynamic link might be identical in some cases.
- The static and dynamic link for function case both functions.

Dynamic link (prior frame)
- The caller of function has function c
- Or the called function has function c

Static vs. dynamic links
- Function a (x : Integer) return Integer is
  function b (y : Integer) return Integer is (x + y);
  function c (z : Integer) return Integer is (b (z));
  begin
  return c (x);
end a;
- a :: Integer -> Integer
  a x = c x

- a :: Integer -> Integer
  a x = c x

= C expands the C standard and provides it though.

Some languages will not have a context by default, like C or Assembly.

- The context for
- The static and dynamic link might be identical in some cases.
- The static and dynamic link for function case both functions.
**Functions**

**Generic Stack-Frame**

- The next function call will produce the next stack frame.
- Variables and parameters from the context stay visible unless the chain of static links.

- Local variables can only be added to the current stack frame.
- Variables and parameters from the context can be added to the current stack frame.

**Functions**

**Generic Stack-Frame**

- The next function call will produce the next stack frame.
- Variables and parameters from the context stay visible unless the chain of static links.

- Note which variables and parameters are visible.

**Functions**

**Generic Stack-Frame**

- The next function call will produce the next stack frame.
- Variables and parameters from the context stay visible unless the chain of static links.

- Note which variables and parameters are visible.
Functions

Generic Stack-Frame – Caller

How to keep any memory allocation after function return?

- Allocate/identify space for the parameters
- Copy the in and out parameters to this space
- Potentially provide links
- Provide a return address ("Post_Call")
  - (usually implicit with the call itself)

Call the function

Post_Call:
- Copy the out and in-out parameters
- to local variables or registers
- Potentially restore the frame pointer
- Restore the stack to its previous state
  - (if the stack has been used)

Functions

Generic Stack-Frame – Callee

Prologue:
- Save all registers which are needed
- Inside this function to the stack
- Establish a new frame pointer
- While potentially saving the previous fp
- Allocate/identify space for local variables
- Potentially initialize local variables

Operations, which will use local and context variables and parameters
- Call the function

Epilogue:
- Potentially restore the prior frame pointer
- Restore the stack to its state at entry
- Return from function

Functions

Generic Stack-Frame – Heap

How to keep any memory allocation after function return?

- By using an out, by-reference parameter, the link to the newly allocated memory area is kept.
- ... and a local variable in the calling function can keep this link.