City University London

IN2009

Language Processors

Session 10

Stack Frames

Igor Siveroni

Schedule

- **Revision class & Test**
  Monday 26th April 2010 (11.00-13.00)
  Room C340
  – 11:00-12:00 Revision part 1
  – 12:00-12:30 Test3
    Type-checking and stack frames.
    Sample questions on Wednesday.
  – 12:30-13:00 Test3 solution
- All results ready on May 3.

Schedule

- **Exam**
  June 2, 14:30-16:00
  – Choose 2 out of 3 questions. 100 points each.
  – Based on test1, test2 and test3.
  – … more during revision class.

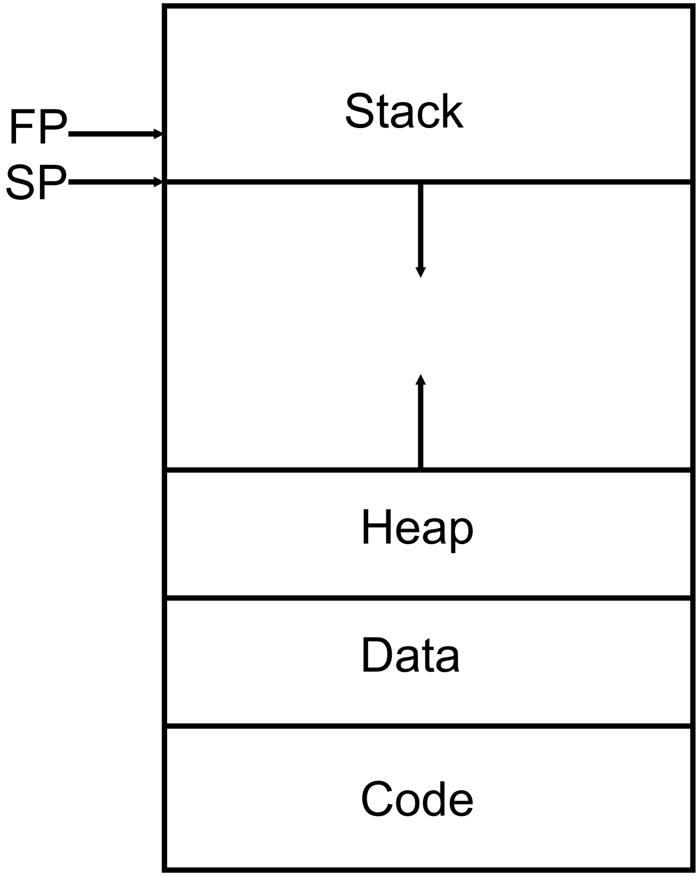
- No Lab today.

Session Plan

Session 6: Stack Frames

- Memory Model
- Stack frames
  - layout
  - frame pointer and stack pointer
  - parameter passing
- TPL representation
- Example

Layout in memory



possible format of a code file before it is loaded into memory:

header [magic number,sizes,entry point]
text [the code]
data [global variable space]

symbol table
[variable & method names etc]
string table
[the text of names in symbol table]

'memory model'

Stack Frames

- Functions/procedures may have *local* variables
- Several invocations at same time each with own instantiations of local variables - e.g. recursive calls,and arguments.
- Local variables destroyed on method return
- LIFO behaviour (implemented with stack data structure)

```
int f(int x) {
   int y = x+x;
   if y < 10
     return f(y);
   else
     return y-1;
}
```

- New instantiation of x created & initialised by f’s caller each time f called
- Recursive calls - many x’s exist simultaneously
- New instantiation of y created each time body of f entered

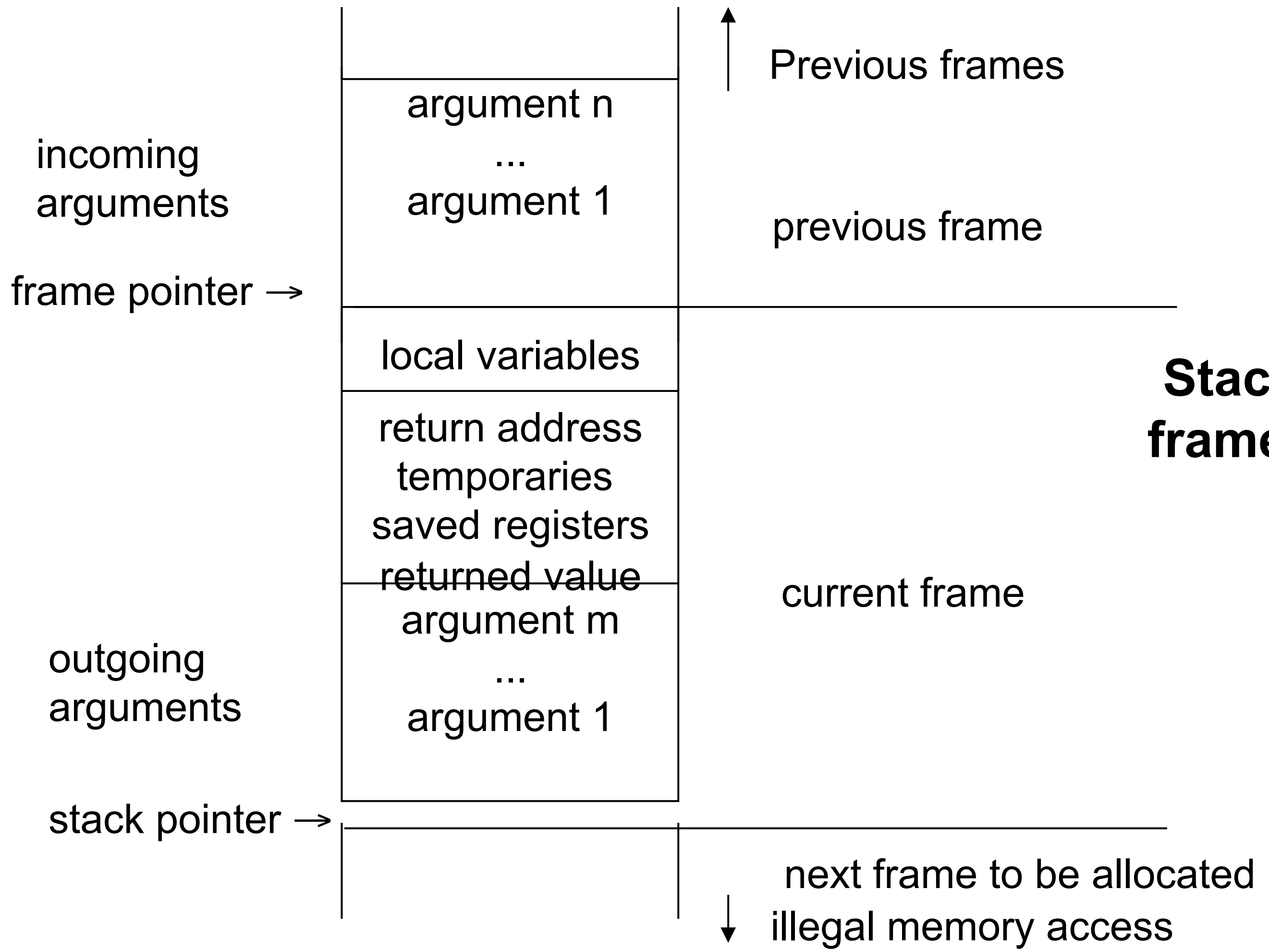
Stack frames

- **Stack frames**: Needed due to the existence of procedures and recursion in a programming language
- Stack frames save the state of execution of a procedure/function: A snapshot of a procedure's state.
- Frame layout design
  - Takes into account particular features of instruction set architecture and programming language being compiled

Stack frames

- Usually a standard frame layout prescribed by manufacturer
  - not necessarily convenient for compiler writers, but...
  - functions/methods written in one language can call functions/methods written in another, so...
  - gain programming language interoperability
  - can combine modules/classes compiled from different languages in same running program

Stack frames



Stack frames

Stack frame layout

- Set of *incoming arguments* (part of previous frame) passed (stored) by caller code
- Return address: location where execution resumes after method return.
- Local variables (those not in registers)
- Temporaries - locations where code temporarily saves intermediate values (if registers not available)

Stack frame layout

- Saved registers: Area for values held in registers that need to be saved when a procedure call is made. Registers are restored when procedure returns.
- Returned value: Place where the callee stores the value returned by the procedure/function.
- Outgoing argument space: to pass (store) parameters when procedure calls other procedures.

Parameter passing

- Pre-1960: passed in statically allocated memory blocks - no recursive functions or methods
- 1970s machines: function arguments passed on the stack
- But program analysis shows that very few functions/methods have >4 arguments, and almost none >6.

Parameter passing

- So on most modern machines
  - first k arguments (k=4 or 6) are passed in registers $r_p,\ldots,r_{p+k-1}$ and the rest passed in memory on the stack
- But in our case (SPL and TPL) we will pass all arguments on the stack.

Procedure call. Steps:

Caller *g(...)* calls callee *f(*a_1*,...,*a_n*):*

- Calling code in *g* puts arguments to *f* at end of *g* frame
- Registers (if necessary) are stored in g's frame.
- Return address is saved in g's frame
- On entry to *f*,
  - SP points to first argument *g* passes to *f*
  - old SP becomes current *frame pointer* FP
  - *f* then allocates frame by setting SP=(SP + framesize)

Procedure call…steps

- Old SP becomes current *frame pointer* FP
- Many implementations have FP as a separate register
  - so method code:
    - has incoming arguments referenced by FP-an offset
    - has local variables referenced by FP+an offset
    - has saved registers, return address and outgoing arguments referenced by FP+an offset orSP-an offset

Procedure return

- Callee saves returned value in caller's frame.
- On exit from *f* : SP = FP, removing frame.
- Code in g restores registers (if any).
- Execution (in g) resumes from return address.

Implementation in TPL

- Procedure call:

  `call` ***`label, framesize`***

Where label is the location of the callee's code, and framesize is the size of the callee's stack frame

- Method return:

  `return`

  variation: **`return Arg`**

With return value.

Implementation in TPL

- We will assume that the size (in memory) of values are:
  - Integer, boolean and addresses: 1 word
  - Float: 2 words
- The size of values is important to determine the total size of the frame.
- We will assume that there's a dedicated FP register.

Implementation in TPL

- Given a variable x located at an offset *offset* from FP, we can access the variable by combining FP with *offset*.
- For example:
  - **`store 20, FP(offset)`**
    stores 20 to variable x
  - **`store FP(offset), R1`**
    stores the contents of x into register R1.

Example

```
int f(int x) {
  float y, z;
  int w;
  y = (float) (10*x);
  z = g(10, y);
  // more code…
  // no more calls
  return w;
}


float g(int a,
        float b) {

// g’s body
  return 10.0;
}
```

- We’ll assume that there’s enough registers (no need to save them)
- **f** stack frame. size = 11
  - location 0: var y (size 2)
  - location 2: var z
  - location 4: var w
  - location 5: return address
  - location 6: g’s returned value
  - location 8: g param b, offset 3
  - location 10: g param a, offset 1

Example

- From f:
  - `STORE 5, FP(-1)`
    stores 5 to parameter x (integer)
  - `STORE FP(2), R2`
    stores value of local variable z into register R2 (float)
- From g:
  - `ADDI FP(-1), 2, R3`
    adds 2 to value of parameter `a` (integer), and stores result in register R3
  - `STORE 10.0, FP(-5)`
    `stores 10.0 in location` reserved for returned value of g.

Example

- Test 3 will:
  - Contain a similar question: given a code sample, provide the stack frame structure.
  - Ask you to list type errors present in a piece of code.
  - Ask you to define the type checks for a new statement or expression.