Foundations of Programming Languages

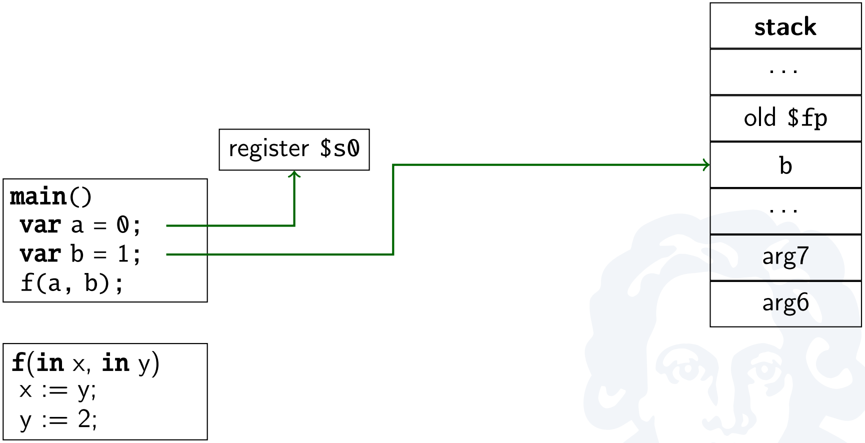
Implementing Parameter Passing

Prof. Dr. Christoph Reichenbach

Fachbereich 12 / Institut für Informatik

29. Oktober 2014

Implementing Subprograms



Implementing Pass-By-Value

```
main()
 var a = 0;
 var b = 1;
 f(a, b);
```

```
main: li     $s0, 0              ; init
      li     $t0, 1
      sd     $t0, -8($fp)
              ; prepare call
      move   $a0, $s0
      ld     $a1, -8($fp)
      jal    f
```

stack
...
old $fp
b
...
arg7
arg6

```
f(in x, in y)
 x := y;
 y := 2;
```

```
f: move   $a0, $a1
   li     $a1, 2
   jreturn
```

- a ($s0), b($fp+8) preserved across call
- f updates local copies ($a0, $a1)

Updates in f invisible to main

Implementing Pass-By-Value

- Common parameter passing mode
- Matches standard subroutine parameter passing methods

Nothing new here, really

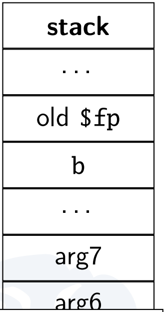
Implementing Pass-By-Result

```
main()
 var a = 0;
 var b = 1;
 f(a, b);
```

```
f(out x, out y)
 x := 4;
 y := 5;
```

```
main: li      $s0, 0             ; init
      li      $t0, 1
      sd      $t0, -8($fp)
              ; call
      jal     f
              ; postprocess call
      move    $s0, $a0
      sd      $a1, -8($fp)
```

```
f: li      $a0, 4
   li      $a1, 5
   jreturn
```



- No need to pass actual values in arguments
- Use parameter storage for passing results out

Implementing Pass-By-(Value-)Result

- Passing results out in parameter storage
- Pass-By-Value-Result:
  - Pass-By-Value for input
  - Pass-By-Result for output

$$\boxed{\textbf{Other schemes are conceivable, but this one is simple}}$$

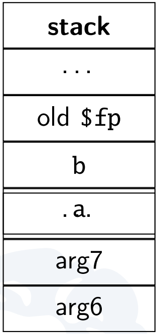
Implementing Pass-By-Reference

```
main()
 var a = 0;
 var b = 1;
 f(a, b);
```

```
f(ref x, ref y)
 x := y;
 y := 2;
```

```
main: li      $s0, 0               ; init
      li      $t0, 1
      sd      $t0, -8($fp)
            ; prepare call
      move    $a1, $fp
      subi    $a1, 8
      sd      $s0, -16($fp)
      move    $a0, $fp
      subi    $a0, 16
      jal     f
            ; postprocess call
      ld      $s0, -16($fp)
```

```
f: ld      $t0, 0($a1)
   sd      $t0, 0($a0)
   li      $t0, 2
   sd      $t0, 0($a1)
   jreturn
```



- Must store a in memory
- Read/write involves memory accesses

Passing By Name / Need

```
var count = 0;
subprogram next(a)
begin
  count := count + a;
  return count;
end

subprogram f(name b)
begin
  print("b1=", b, ", b2=", b);
end
f(next(2));     (* print("b1=2, b2=4") *)
```

- Parameter side effects triggered at every use
  $\Rightarrow$ Must call evaluation code
- Implementation uses *closures* / *thunks*

Summary

- Pass-By-Value:
  As with regular subroutine invocations
- Pass-By-Result:
  Use registers, stack storage to pass results *out*
- Pass-By-Value-Result:
  Combine Pass-By-Value, Pass-By-Result
- Pass-By-Reference:
  - Pass memory address, rather than memory contents
  - Store registers in memory, if needed
- Pass-By-Name/Need:
  Needs more advanced techniques