

Programming in Systems (37-023)

Programming in Assembler Basics of Operating Systems Models of Computer Architecture

Lecturer today:

Prof. Thomas M. Stricker

Text-/Reference-Books:

R.P.Paul: SPARC Architecture... and C Sun SPARC V8 Manual and K&R C Reference

Topics of Today:

- Standard/Leaf Subroutines
- SPARC Calling Convention
- Arguments on the Stack
- Pointer as Arguments

17.12.2001 - 1 37-023 Systemprogrammierung

© Stricker/Alonso

Calling a subroutine (Basics)

- The address of the calling instruction in the "main program" is saved. It is the so called return address.
- The SPARC architecture saves that address into %o7 and passes it as %i7 to the subroutine (details later).
- · Calls to subroutines are delayed like branches.
- The actual destination for the return is therefore %o7 + 8 (two 32bit instructions after the call).
- If the name of the subroutine is given by a label (some fixed location in the code) the call is made with a call instruction:

call <name>

 stores the %pc in %o7 and provides a delay slot

Subroutines

- Basic tool for structured and object oriented programming.
- Functions in C procedures in Oberon -Methods in C++/Java.
- Code pieces that can be called again with different arguments:
 - -> code reuse, more compact code.
- Called with a jump instruction from main program or from subroutine itself.
- Subroutines are transparent when it comes to register use. Content of registers must be saved on the stack for subroutine call.
- Arguments to the subroutine are handled like local variables.
- Open subroutines are like a macro, where arguments are substituted.

37-023 Systemprogrammierung 17.12.2001 - 2

© Stricker/Alonso

by

impl <source_reg>, <dest_reg>

 This is necessary for calling the contents of a function variable or for calling virtual methods of classes (inheritance)

 If the address of a subroutine is calculated at run time the invocation is done

- Calls a function a the address provided by the register <source_reg>.
- Saves the %pc in <dest req>

jmpl %o0, %o7

is semantically equivalent to

call (%00)!this is illegal syntax

 The return from a subroutine is also done with the jump&link mechanism as:

jmpl %i7 + 8, %g0 /* discards the %pc */ which is equivalent to the **ret** instruction

Save / Restore (temporary definition)

save %sp, -64, %sp

- stores the registers %I0-%I7, %i0-%i7 to the stack (into the 64 bytes provided)
- copies the regs %o0-%o7- > %i0-%i7:
- does an addition (mostly used to get more space on the stack).

restore

- copies the regs %o0-%o7 <- %i0-%i7:
- load the contents of the registers %l0-%l7, %i0-%i7 back from the stack.
- also performs an add instructions which is normally left unused (except for gcc which uses it when it can!)

e.g. restore %g2,12,%g2

17.12.2001 - 5 37-023 Systemprogrammierung

© Stricker/Alonso

Passing of Arguments

- Must allow arguments that are calculated as expressions.
- Must allow recursion.

So hard coding is not a solution!

- Passing all arguments by the stack is slow since this must go by main memory (or at least by the stacks).
- Therefore the SPARC architecture uses the registers %00,...,%05 to pass arguments across subroutine calls.

note that %06 holds the stack pointer and %07 the return address.

- If more than six arguments must be passed the stack is used
- for larger data structure, the stack is used.

Typical Subroutine Call

```
main:
2
3
           call mysubr
4
                                    !or something meaningful
           nop
5
6
   mysubr:
7
           save %sp,-112,%sp
9
           ret
10
                                         !fills delay slot of ret
           restore
```

17.12.2001 - 6 37-023 Systemprogrammierung

© Stricker/Alonso

SPARC Calling Convention

- Determines the organization of the stack frame during subroutine calls.
- Allows separate compilation and linking. This permits C programs that call assembly routines.
- Stack is organized as follows:
 - **64 bytes** for the storage of the register contents at a save instruction.
 - 4 bytes for a struct pointer, that permits the return of structs
 - 24 bytes for the first six parameters that are usually passed in registers (the space is still reserved for GDB)
 - 4 bytes for padding (alignment)
 - n bytes for the local variables of the subroutines.
- Don't forget: The stack pointer must always be doubleword aligned

Example

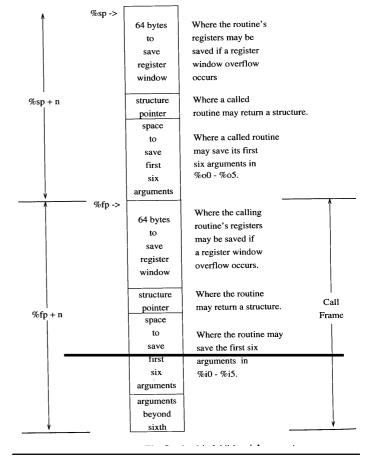
General form

```
save %sp, -(64 + 4 + 24 + n) & -8, %sp
```

Example

```
vector()
int a,b;
char d;
save %sp, -(64 + 4 + 24 + 9) & -8, %sp
```

- Addressing convention:
 - The addresses of local variables are negative with respect to %fp.
- All other data in the stack are positive with respect to %sp.



Picture of the stack

17.12.2001 - 10 37-023 Systemprogrammierung

© Stricker/Alonso

17.12.2001 - 9 37-023 Systemprogrammierung

© Stricker/Alonso

Return from a Subroutine

- Subroutines, that return arguments are called functions:
- Example in C:

```
int example(int a, int b, char c);
main() {
  int r:
  r = example(3, 5, 4);
  printf("%d\n", r);
}
int example(int a, int b, char c)
  int x, y;
  short ary[128];
  register int i, j;
  x = a + b;
  i = c + 64;
  ary[i] = c + a;
  y = x * a;
  j = x + i;
  return x + y;
```

Example in Assembler:

```
! a_r in %i0
                                              ! b_r in %i1
                                               ! c_r in %i2
   x_s = -4
   y_s = -8
   ary_s = -256
                                                !i r in %10
                                                !j_r in %l1
_example:
           %sp, -360, %sp
   save
                                                !x = a + b
   add
           %i0, %i1, %o0
   st
           %00, [%fp + x_s]
                                                !i = c + 64
   add
           %i2, 64, %10
   add
           %i0, %i2, %o0
                                             !ary[i] = c + a
   sll
           %10, 1, %o1
   add
           %fp, ary_s, %o2
   sth
           %00, [%01 + %02]
                                                !y = x * a
   ld
           [%fp + x_s], %00
   mov
           %i0, %o1
   smul
           %00, %01, %00
           %00, [%fp + y_s]
   st
   ld
           [\%fp + x_s], \%00
                                                 !j = x + i
   add
           %10, %00, %11
   14
           [\%fp + x_s], \%00
   ld
           [%fp + y_s], %o1
                                              !return x + y
   restore %00, %01, %00
```

Leaf Subroutines

- Do not contain any further subroutine calls. The call can be optimized. Save/ restore instructions can be avoided.
- Leaf subroutines must use the register set of the calling routine.
- Restrictions: Use of %00 to %05 and %g0 to %g1 is permitted.
- Call works as for subroutines but with %o7 instead of %i7. Therefore return to %o7+8!

	retl		==	jmpl	%07+8 ,	%g0
1 2 3 4 5 6		add add add add add	%01, %00, %0 %02, %00, %00 %03, %00, %00 %04, %00, %00 %05, %00, %00 [%sp + arg7_s	0	,	
7		add	%01, %00, %00)		
8		ld	[%sp + arg8_s	s], %o1		
9		retl				
10		add	%01, %00, %00)		

17.12.2001 - 13 37-023 Systemprogrammierung

© Stricker/Alonso

Register Names

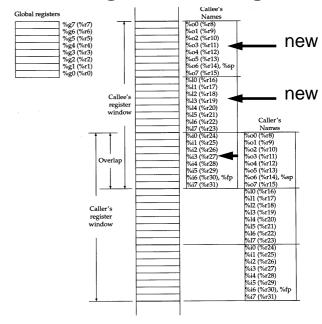
 Alternative names of registers refer to the use in subroutine calls:

```
%lx
      Local Register in Subroutine
%ix
      Input Register in Subroutine
      Output Register in Subroutine
%f0-%f31:
             %f0-%f31
             %g0-%g7
%r0-%r7:
%r8-%r15:
             %00-%07
             %10-%17
%r16-%r23:
%r24-%r31:
             %i0-%i7
             %r14, %o6
%sp:
             %r15, %o7
%ret:
%fp:
             %r30, %i6
```

Global Register

%ax

Register renaming



- Save
- Restore

17.12.2001 - 14 37-023 Systemprogrammierung

© Stricker/Alonso

SPARC calling convention

\$03 (\$r11)					
Note		-			
in %14 (%128) incoming param 5 † %13 (%127) incoming param 4 † %12 (%126) incoming param 3 † %10 (%124) incoming param 2 † %10 (%124) incoming param 1 / return value to caller † %17 (%123) local 6 † %16 (%122) local 6 † %15 (%121) local 5 † %10 (%14) with first parameters of the first paramete		%fp, %i6	(%r30)	frame pointer †	
\$13 (\$r27) incoming param 4 † \$12 (\$r26) incoming param 3 † \$10 (\$r24) incoming param 2 † \$17 (\$r23) local 7 † \$16 (\$r22) local 6 † \$15 (\$r21) local 5 † \$13 (\$r19) local 3 † \$11 (\$r17) local 1 † \$10 (\$r16) local 0 † \$10 (\$r16) local 0 † \$10 (\$r16) local 0 † \$11 (\$r17) local 1 † \$10 (\$r16) local 0 † \$11 (\$r17) local 1 † \$10 (\$r16) local 0 † \$11 (\$r17) local 1 † \$10 (\$r16) local 0 † \$11 (\$r17) local 1 † \$10 (\$r16) local 0 † \$11 (\$r17) local 1 † \$10 (\$r16) local 0 † \$11 (\$r17) local 1 † \$10 (\$r16) local 0 † \$11 (\$r17) local 1 † \$10 (\$r16) local 0 † \$11 (\$r17) local 1 † \$10 (\$r16) local 0 † \$11 (\$r17) local 1 † \$11 (\$r17) local 1 † \$12 (\$r18) local 0 † \$13 (\$r10) loutgoing param 6 † \$14 (\$r12) loutgoing param 5 † \$15 (\$r13) loutgoing param 3 † \$16 (\$r20) local 7 (\$r30) local 8 \$17 (\$r20) local 9 \$18 (\$r10) local 9 \$18 (\$r10) local 1 † \$10 (\$r16) local 0 † \$11 (\$r17) local 1 † \$11 (\$r17) local 1 † \$12 (\$r18) local 2 † \$13 (\$r10) local 9 \$14 (\$r20) local 4 † \$15 (\$r21) local 5 † \$15 (\$r21) local 5 † \$16 (\$r20) local 4 † \$17 (\$r20) local 9 † \$18 (\$r10) local 9 † \$18 (\$r10) local 9 † \$18 (\$r10) local 1 † \$12 (\$r18) local 2 † \$13 (\$r10) local 1 † \$14 (\$r20) local 4 † \$15 (\$r20) local 4 † \$15 (\$r20) local 4 † \$16 (\$r20) local 4 † \$17 (\$r10) local 2 † \$18 (\$r10) local 2 † \$18 (\$r10) local 3 † \$18 (\$r10) local 1 † \$18 (\$r10) local 2 † \$18 (\$r10) local 2 † \$18 (\$r10) local 1 † \$18 (\$r10) local 2 † \$18 (\$r		%i5	(%r29)	incoming param 6 †	
\$12 (\$r26) incoming param 3 † \$11 (\$r25) incoming param 2 † \$10 (\$r24) incoming param 1 / return value to caller † \$17 (\$r23) local 7 † \$16 (\$r22) local 6 † \$15 (\$r21) local 5 † \$10 (\$r16) local 3 † \$11 (\$r17) local 1 † \$10 (\$r16) local 0 † \$11 (\$r17) local 1 † \$10 (\$r16) local 0 † \$10 (\$r16) local 0 † \$11 (\$r17) local 1 † \$10 (\$r16) local 0 † \$11 (\$r17) local 1 † \$10 (\$r16) local 0 † \$11 (\$r17) local 1 † \$12 (\$r18) local 0 † \$13 (\$r11) loutgoing param 5 † \$14 (\$r12) loutgoing param 5 † \$15 (\$r13) loutgoing param 3 † \$16 (\$r16) loutgoing param 2 † \$17 (\$r17) loutgoing param 2 † \$18 (\$r10) loutgoing param 1 / return value from callee † \$18 (\$r10) loutgoing param 1 / return value from callee † \$18 (\$r10) loutgoing param 1 / return value from callee † \$18 (\$r10) local 3 † \$18 (\$r10) local 0 † \$18 (\$r10) local 1 † \$18 (\$r10) local 2 † \$18 (\$r10) local 2 † \$19 (\$r10) local 0 † \$10 (\$r16) local 0 †	in			incoming param 5 †	
\$11 (\$r25) incoming param 2 †		%i3	(%r27)	incoming param 4 †	
\$10 (\$r24) incoming param 1 / return value to caller † \$17 (\$r23) local 7 † \$16 (\$r22) local 6 † \$15 (\$r21) local 5 † local \$14 (\$r20) local 4 † \$13 (\$r19) local 3 † \$12 (\$r18) local 2 † \$10 (\$r16) local 0 † \$07 (\$r15) temporary value / address of CALL instruction ‡ \$05 (\$r13) outgoing param 6 ‡ \$04 (\$r12) outgoing param 6 ‡ \$05 (\$r13) outgoing param 6 ‡ \$04 (\$r12) outgoing param 4 ‡ \$02 (\$r10) outgoing param 3 ‡ \$03 (\$r11) outgoing param 2 ‡ \$04 (\$r8) outgoing param 1 / return value from callee ‡ \$27 (\$r7) global 7 (\$PARC ABI: use reserved) \$28 (\$r6) global 6 (\$PARC ABI: use reserved) \$29 (\$r2) \$2 global 3 (\$PARC ABI: global register variable §) \$30 (\$r1) \$2 global 3 (\$PARC ABI: global register variable §) \$31 (\$r3) floating-point condition codes ‡ \$10 time point value † \$10 time point value † \$10 time point valu		%i2	(%r26)	incoming param 3 †	
\$17 (\$r23)		% i1	(%r25)	incoming param 2 †	
Note		%i 0	(%r24)	incoming param 1 / return value to caller †	
State Stat		\$17	(%r23)	local 7 †	
State Stat		% 16	(%r22)	local 6 †	
\$13 (\$r19) local 3 † \$12 (\$r18) local 2 † \$10 (\$r16) local 0 † \$07 (\$r15) temporary value / address of CALL instruction ‡ \$08 (\$r14) stack pointer † \$09 (\$r13) outgoing param 6 ‡ \$01 (\$r2) outgoing param 6 ‡ \$02 (\$r10) outgoing param 3 ‡ \$02 (\$r10) outgoing param 3 ‡ \$01 (\$r9) outgoing param 2 ‡ \$03 (\$r11) outgoing param 2 ‡ \$04 (\$r2) outgoing param 1 / return value from callee ‡ \$08 (\$r6) global 7 (\$PARC ABI: use reserved) \$10 (\$r3) global 5 (\$PARC ABI: use reserved) \$10 (\$r3) global 5 (\$PARC ABI: use reserved) \$10 (\$r3) global 5 (\$PARC ABI: global register variable §) \$10 (\$r3) temporary value ‡ \$10 (\$r6 field of \$r5) (\$r6 field of \$r5) \$10 (\$r6 field of \$r5) (\$r6 field of \$r5) \$10 (\$r5) temporary value ‡ \$10 (\$		% 15	(%r21)	local 5 †	
\$12 (\$r18)	local	814	(%r20)	local 4 †	
\$11 (\$r17) local 1 † \$10 (\$r16) local 0 † \$07 (\$r15) temporary value / address of CALL instruction ‡ \$08 (\$r14) stack pointer † \$09 (\$r13) outgoing param 6 ‡ \$01 (\$r2) outgoing param 5 ‡ \$03 (\$r11) outgoing param 3 ‡ \$02 (\$r10) outgoing param 3 ‡ \$01 (\$r9) outgoing param 2 ‡ \$00 (\$r8) outgoing param 1 / return value from callee ‡ \$07 (\$r7) global 7 (\$PARC ABI: use reserved) \$09 (\$r6) global 6 (\$PARC ABI: use reserved) \$09 (\$r5) global 5 (\$PARC ABI: global register variable §) \$09 (\$r2) \$09 (\$r2) global 2 (\$PARC ABI: global register variable §) \$10 (\$r1) \$00 (\$r0) Outgoing param 1 / return value from callee ‡ \$10 (\$r6) global 6 (\$PARC ABI: use reserved) \$10 (\$r1) global 6 (\$PARC ABI: global register variable §) \$10 (\$r1) global 2 (\$PARC ABI: global register variable §) \$10 (\$r1) global 2 (\$PARC ABI: global register variable §) \$10 (\$r2) floating-point condition codes ‡ \$10 (\$r6 field of \$psr) (\$r6 field of \$fsr) (\$r7 (\$r8 (\$r8 (\$r8 (\$r8 (\$r8 (\$r8 (\$r8 (\$r8		\$13	(%r19)	local 3 †	
\$10 (\$r16) local 0 †		112	(%r18)	local 2 †	
\$\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \		*11	(%r17)	local 1 †	
Sep. %06 (%r14) Stack pointer †		\$10	(%r16)	local 0 †	
\$\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \		\$07	(%r15)	temporary value / address of CALL instruction ‡	
out %04 (%r12) outgoing param 5 ‡ %03 (%r11) outgoing param 4 ‡ %02 (%r10) outgoing param 3 ‡ %01 (%r9) outgoing param 2 ‡ %00 (%r8) outgoing param 1 / return value from callee ‡ %g7 (%r7) global 7 (SPARC ABI: use reserved) %g6 (%r6) global 6 (SPARC ABI: use reserved) %g5 (%r5) global 5 (SPARC ABI: global register variable §) %g3 (%r3) global 3 (SPARC ABI: global register variable §) %g2 (%r2) global 3 (SPARC ABI: global register variable §) %g1 (%r1) temporary value ‡ %g0 (%r0) Y register (used in multiplication/division) ‡ state %y (icc field of %fsr) (cc field of %esr) (fcc field of %esr) Floating-point condition codes ‡ Coprocessor condition codes ‡ floating-point value ‡		%sp, %o6	(%r14)	stack pointer †	
\$03 (\$r11) Outgoing param 4 ‡		% o5	(%r13)	outgoing param 6 ‡	
\$\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \	out	% 04	(%r12)		
\$\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \		%o3	(%r11)	outgoing param 4 ‡	
\$00 (\$r8)		% o2	(%r10)	outgoing param 3 ‡	
\$97 (%r7) global 7 (SPARC ABI: use reserved)		. %01	(%r9)	outgoing param 2 ‡	
\$\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \		\$00	(%r8)	outgoing param 1 / return value from callee ‡	
Second S		%g7	(%r7)	global 7 (SPARC ABI: use reserved)	
global %g4 (%r4) %g3 (%r3) %g2 (%r2) %g1 (%r1) %g0 (%r0) state %y (icc field of %psr) (fcc field of %csr) (ccc field of %csr) %f1		%g6	(%r6)	global 6 (SPARC ABI: use reserved)	
Signature Sign		%g5	(%r5)	global 5 (SPARC ABI: use reserved)	
state \$\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \	global	%g4	(%r4)	global 4 (SPARC ABI: global register variable §)	
\$g1 (\$r1) temporary value ‡		% g3	(%r3)	global 3 (SPARC ABI: global register variable §)	
\$g0 (\$r0) 0		%g2	(%r2)	global 2 (SPARC ABI: global register variable §)	
state %y		% g1	(%r1)	temporary value ‡	
(icc field of %psr) (fcc field of %fsr) (ccc field of %fsr) (ccc field of %csr) %f31 floating - point value ‡ floating - point : : : : : : : : : : : : : : : : : : :		% g0	(%r0)	0	
(fcc field of %fsr) (ccc field of %csr) %f31 floating-point condition codes ‡ Coprocessor condition codes ‡ floating-point value ‡ floating-point condition codes ‡	state	*у			
(ccc field of %csr) Coprocessor condition codes ‡ #f31 floating-point value ‡ floating : : : : : : : : : : : : : : : : : : :					
%f31 floating-point value ‡ floating : point :					
floating : : : : : : : : : : : : : : : : : : :			of %csr)		
point :		%f31		floating-point value ‡	
	floating	ig :		:	
9 f 0 floating-point value +	point	:		•	
10ating-point value ‡		% f0		floating-point value ‡	

Calling Convention

SPARC

•	%g0	NULL

- %g1 temp, caller saved
- %g2..g4 global, callee must save
- %g5..g7 global, reserved
- %l0..%l7 local, callee must save with "save" (unless leaf proc.)
- %i0..%i5 in parametes, caller must save (unless leaf proc.)
- %00..%05 local, callee must save with "save" (unless leaf proc.)
- %o6, %o7 stack-pointer, return(tmp)
- %i6,%i7 frame-pointer, ret-addr
- %f0..%f31 caller saves

17.12.2001 - 17 37-023 Systemprogrammierung

© Stricker/Alonso

Pointer as Arguments (C)

- Call by reference paradigm (as contrast to call by value)
- Need to pass pointers as arguments to subroutines.
- Classical example swap(x,y)

```
swap(int *x, int *y)
{
    int temp;

    temp = *x;
    *x = *y;
    *y = temp;
}
```

Alternatives: Simple Calling Conventions

Other processors:

- MIPS
- 68000
- iWarp

Register contents are always stored on the stack:

- Argument registers
- Caller saved registers
- Callee saved registers
- Scratch registers
- Interrupt registers

17.12.2001 - 18 37-023 Systemprogrammierung

© Stricker/Alonso

Pointer as Arguments (assembly language)

```
x s = -4
                  !int x
                                          !local variables
                  !int y
   y_s = -8
main:
                                               !bsp54.s
                                   !2 words for arguments
           %sp, -104, %sp
   save
   mov
           5, %00
           %00, [%fp + x_s]
                                                  !x = 5
   st
           7, %00
                                                  !v = 7
           00, [fp + y_s]
   st
   add
           %fp, x_s, %o0
                                      !pointer to x in %o0
   call
           swap
                                      !pointer to y in %o1
   add
           %fp, y_s, %o1
   ret
   restore
                                          ! a leaf routine
   .global _swap
swap:
                                               1\%02 = x
   ld
           [%00], %02
   ld
           [%o1], %o3
                                               1\%03 = y
           %02, [%01]
   st
   retl
           %03, [%00]
   st
```

Passing Arguments on the Stack

- Needed when more than 6 arguments are to be passed
- Caller must make space on the stack before calling subroutine

```
foo(1,2,3,4,5,6,7,8)
int foo(int al, int a2, int a3, int a4, int a5,
int a6, int a7, int a8)
{
return a1+a2+a3+a4+a5+a6+a7+a8;
}
```

Reservation of space for arg 7-8 by

```
add \$sp, -2*4 \& -8, \$sp
```

 a7 and a8 will be written at %sp+92 and %sp+96 to the stack (arg8_s =92, arg7_s=96)

17.12.2001 - 21 37-023 Systemprogrammierung

© Stricker/Alonso

Returns of "structs"

• Structs can be returned by functions:

Example in C:

```
/* bsp50.c */
struct point {
  int x, y;
}

struct point zero() {
  struct point local;

  local.x = 0;
  local.y = 0;

  return local;
}

main()
{
  struct point x1, x2;

  x1 = zero();
  x2 = zero();
}
```

Assembly Code

```
!bsp52.s
_main:
       save% sp, -96, %sp
              %sp, -2 * 4 & -8, %sp
       add
              8, %00
       mov
       st
              %00, [%sp + arg8_s]
              7, %00
       mov
              %00, [%sp + arg7_s]
       st
              6, %05
       mov
              5, %o4
       mov
       mov
              4, %03
              3, %o2
       mov
              2, %o1
       mov
              foo
       call
              1, %00
       mov
              %sp, -2 * 4 & -8, %sp
       sub
              1, %q1
       mov
       ta
_foo:
   save
          %sp, -96, %sp
           [%fp + arg8_s], %00
   ld
           [%fp + arg7_s], %ol
                                      !the 7th argument
           %00, %01, %00
   add
          %i5, %o0, %o0
   add
                                     !the sixth argument
          %i4, %o0, %o0
                                     !the fifth argument
   add
          %i3, %o0, %o0
                                    !the fourth argument
   add
           %i2, %o0, %o0
                                     !the third argument
   add
          %i1, %o0, %o0
                                !the second argument
   add
   ret
                                      !the first argument
   restore %i0, %o0, %o0
```

17.12.2001 - 22 37-023 Systemprogrammierung

© Stricker/Alonso

Assembly Code:

```
!tkisem version bsp51.s
   !local variables
   x1 = -8
   x2 = -16
main:
          %sp, -112, %sp
                                       !double aligned
   mov
          1, %11
                                       !x1.x = 1 for test
          %11, [%fp+x1]
   st
          2, %11
   mov
          %11, [%fp+x1+4]
                                       !x1.y = 2 for test
   st
   add
          %fp, x1, %o0
                                  !%fp -8 is pointer to x1
   call zero
          %00, [%sp + 64]
                                   !struct pointer adress
   add %fp, x2, %o0
   call zero
          %00, [%sp + 64]
   st
   ret
   restore
   ta
   x = 0
   y = 4
   .global zero
          %sp, -104, %sp
          [%fp + 64], %o0
                                   !get pointer into %o0
   ld
          g0, [000 + x]
   st
   st.
          %g0, [%o0 + y]
          ret
          restore
```