Computer Science & Engineering Department
I. I. T. Kharagpur

Operating System: CS33007

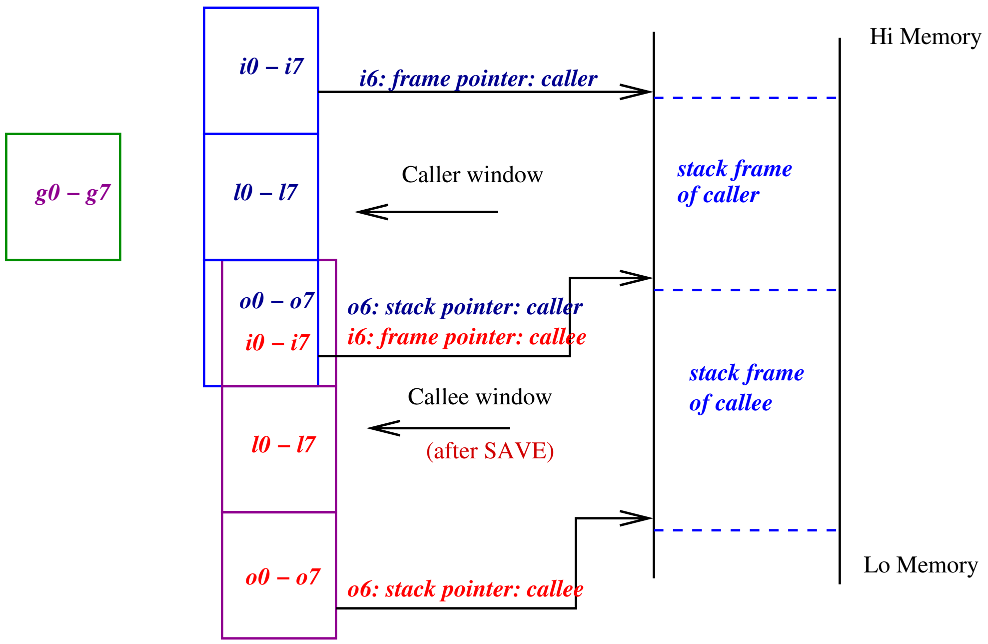
3rd Year CSE: 5th Semester (*Autumn 2006 - 2007*)
Lecture I (**SPARC Register Windows**)

Instructors: PDG and GB *Date:* 20th July, 2006

1. 32 GPRs (r_0 to r_{31}): each of size 64-bits, but lower 32-bits are used to store 32-bit integer or 32-bit address.
2. r_0 to r_7 are same as g_0 to g_7 - global registers. The register r_0 or g_0 is always zero (0).
3. For the current window:
   - *out registers*: r_8 to r_{15} are same as o_0 to o_7.
   - *local registers*: r_{16} to r_{23} are same as l_0 to l_7.
   - *in registers*: r_{24} to r_{31} are same as i_0 to i_7.
   - First six integer size parameters can be passed through the out register o_0 to o_5. Additional parameters are passed through the stack.
   - The output register o_6 (r_{14}) is used as the stack pointer (sp). The in register i_6 (r_{30}) is used as the frame pointer.
   - A 'call' instruction (pseudo instruction using jmpl - jump and link) puts its address (not the address of the next instruction) in o_7. The actual return address is 8-byte more from the content of o_7. Every call instruction is followed by a *delay slot* instruction, so the return address is 8-byte away.
4. In the called procedure, if the 'save' instruction is executed, there is a shift in the current register window, current window pointer (CWP) is decremented. All the 'out' registers of the caller are available as the 'in' registers of the callee. The 'restore' instruction at the end of the callee, restores the register window for the caller.
5. The new stack pointer of the callee is updated by the stack frame size:

   ```
   save %sp, -C, %sp !  sp (new) = sp (old) - C
   ```

   The caller's stack pointer (old o_6) is the callee's frame pointer (new i_6).



6. The old o_7 (of caller) or the new i_7 (of callee) holds the address of the call instruction. The actual return address is $i_7 + 8$.

7. The integer (integer, pointer) size return value is saved in callee's i_0, which the caller gets in o_0.

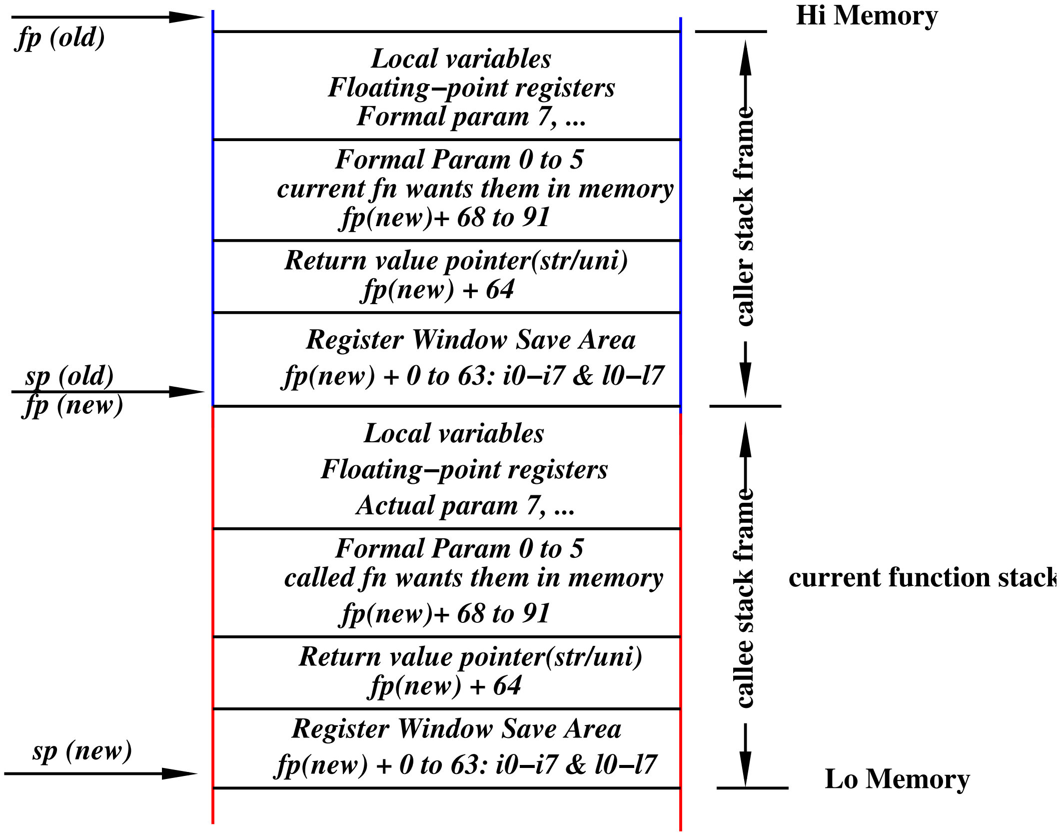
8. If the return value is larger in size, such as a structure or a union, the caller provides space for the return value to copy. The address of the space is kept in the stack frame of the caller at *fp*(new)+64 or *sp*(old)+64.

   Called function uses the address to write the value. It also writes the content of *fp*(new)+64 to its i_0 (o_0 of caller).

   There is more in this story - When the return value is larger than integer size, the compiler writes an `unimp` (unimplemented instruction)[1] after the delay instruction following the `call`. There is an immediate field of `unimp` that holds the size of the return object expected by the caller. The callee checks it with its own return value size. If there is a match, the value is written in proper place using the pointer at *fp*(new)+64 and the control is transfered to $i_7 + 12$ (skipping the `unimp`). If there is mismatch, the control is transfered to $i_7 + 8$ causing an illegal instruction trap.

9. It may be necessary to save the formal parameters in the stack. As an example if the address of a formal parameter is needed, then it cannot be in a register.

[1] The *illegal instruction trap* is generated whenever this instruction is executed.


fp (old)
Hi Memory
Local variables
Floating−point registers
Formal param 7, ...
Formal Param 0 to 5
current fn wants them in memory
fp(new)+ 68 to 91
Return value pointer(str/uni)
fp(new) + 64
Register Window Save Area
fp(new) + 0 to 63: i0−i7 & l0−l7
caller stack frame
sp (old)
fp (new)
Local variables
Floating−point registers
Actual param 7, ...
Formal Param 0 to 5
called fn wants them in memory
fp(new)+ 68 to 91
Return value pointer(str/uni)
fp(new) + 64
Register Window Save Area
fp(new) + 0 to 63: i0−i7 & l0−l7
callee stack frame
current function stack
sp (new)
Lo Memory

10. SPARC assembly language - an example: C program **merge.c**

```
/****************** merge.c *******************/
#include <stdio.h>
#define MAX 100
#define DATA1 9

int data1[MAX] = {10, 20, 30, 40, 50, 60, 70, 80, 90} ;
int mergedData[MAX] ;
void merge(int [], int) ;

int main(){
     int n, i, data2[MAX] ;

     printf("Enter a positive integer:") ;
     scanf("%d", &n) ;
     printf("\nEnter %d integer data in ascending order:\n", n) ;
     for(i=0; i<n; ++i) scanf("%d", &data2[i]) ;

     merge(data2, n) ;
     for(i=0; i<DATA1+n; ++i) printf("%d ", mergedData[i]) ;
     printf("\n") ;

     return 0;
}
void merge(int dat[], int m){
     int i, j, k ;

     for(i=0, j=DATA1+m-1; i<m; ++i, --j) data1[j] = dat[i] ;
     for(i=k=0, j=DATA1+m-1; k<DATA1+m; ++k)
          if(data1[i] < data1[j]) mergedData[k] = data1[i++] ;
          else mergedData[k] = data1[j--] ;
}
```

11. The assembly language program generated by the gcc compiler - **merge.s**. Comments are written manually.

```
     .file     "merge.c"          ! File name of C program
gcc2_compiled.:
     .global data1                ! data1 is a global name.
.section     ".data"              ! .data section to store global data
     .align 4                     ! The section is aligned to 4-byte
     .type      data1,#object     ! boundary. The type of data1 is data object.
     .size      data1,400         ! Allocation for data1 is 100*4 bytes
data1:
     .uaword     10               ! data1[0] initialized to 10
     .uaword     20               ! data1[1] initialized to 20
     .uaword     30
     .uaword     40
     .uaword     50
     .uaword     60
     .uaword     70
     .uaword     80
     .uaword     90               ! data1[8] initialized to 90
     .skip 364                    ! No initialization for 400 - 9*4 = 364
                                  ! bytes.
.section     ".rodata"            ! Read-only data section
     .align 8                     ! The section is aligned to 8-byte
                                  ! boundary.
.LLC0:                            ! Label for the string constant
     .asciz     "Enter a positive integer:"     ! for printf parameter,
     .align 8                     ! aligned to 8-byte boundary (why?).
.LLC1:
     .asciz     "%d"              ! scanf parameter string.
     .align 8
.LLC2:
     .asciz     "\nEnter %d integer data in ascending order:\n"
                                  ! 2nd printf parameter string.
     .align 8
.LLC3:
     .asciz     "%d "             ! 3rd printf parameter string.
     .align 8
.LLC4:
     .asciz     "\n"              ! 4th printf parameter string.
.section     ".text"          ! The code section .text starts.
     .align 4                 ! The section is aligned to 4-byte boundary.
     .global main             ! main is a global symbol
     .type      main,#function     ! The type of main is function.
     .proc     04
main:                         ! Label for main - main starts here.
     !#PROLOGUE# 0
```

```
        save     %sp, -520, %sp  ! Stack frame of size 520 bytes is created,
        !#PROLOGUE# 1            ! 400 byte is for the local array.
                                 ! Register window shift by save.
                                 ! sp (new) <-- sp (old) - 520.
                                 ! old sp is the new fp.
        sethi     %hi(.LLC0), %o1     ! o1 <-- higher order 22-bits of
                                 ! memory address corresponding to
                                 ! label .LLC0, address of "Enter ...
        or     %o1, %lo(.LLC0), %o0     ! lo extracts lower order 10 bits of
                                 ! .LLC0, and 'or' with o1
                                 ! o0 <-- memory address of .LLC0.
        call     printf, 0       ! call printf, the only parameter is
                                 ! in o0
        nop                      !
        add     %fp, -20, %o1    ! Address of n is in o1
        sethi     %hi(.LLC1), %o2   ! First parameter, address of format string
                                 ! for scanf
        or     %o2, %lo(.LLC1), %o0 ! in o0.
        call     scanf, 0        ! Call scanf with two parameters in o0 and o1.
        nop                      !
        sethi     %hi(.LLC2), %o1   ! Format string for printf
        or     %o1, %lo(.LLC2), %o0 !
        ld     [%fp-20], %o1     ! o1 <-- Memory[fp-20], value of n in o1
        call     printf, 0       ! Call to printf with two parameters.
        nop
        st     %g0, [%fp-24]     ! Memory[fp - 24] <-- g0, (i = 0), g0 is
                                 ! always zero. fp - 24 is the
                                 ! location (address) for i.
.LL3:
        ld     [%fp-24], %o0     ! o0 <-- Memory[fp-24] (i)
        ld     [%fp-20], %o1     ! o1 <-- Memory[fp - 20] (n)
        cmp     %o0, %o1         ! Compare o0 (i) and o1 (n)
        bl     .LL6              ! if i < n then goto .LL6 (loop),
        nop                      ! delay slot - no operation,
        b     .LL4               ! otherwise goto .LL4 (out of loop)
        nop                      ! delay slot - no operation
.LL6:                            ! Loop starts
        add     %fp, -424, %o0   ! o0 <-- fp - 424, the address of data2[0]
                                 ! i.e. the value of data2
        ld     [%fp-24], %o1     ! o1 <-- Memory[fp-i] (i)
        mov     %o1, %o2         ! o2 <-- o1 (i)
        sll     %o2, 2, %o3      ! Shift-left o2 twice: o3 <-- 4*o2 (4*i)
        add     %o0, %o3, %o1    ! o1 <-- o0 + o3 (data2 + 4*i)
                                 ! Address of the ith element of data2
        sethi     %hi(.LLC1), %o2     ! Address of the format string
        or     %o2, %lo(.LLC1), %o0   !
        call     scanf, 0        ! Call to scanf to read data2[i]
```

```
        nop                         !
.LL5:
        ld      [%fp-24], %o0      ! o0 <-- i
        add      %o0, 1, %o1       ! o1 <-- o0 (i) + 1
        st      %o1, [%fp-24]      ! i <-- o1 (++i)
        b      .LL3                ! goto .LL3 to continue loop
        nop                        ! delay slot (no-op)
.LL4:
        add      %fp, -424, %o1    ! o1 <-- data2
        mov      %o1, %o0          ! o0 <-- o1 (data2), 1st parameter
        ld      [%fp-20], %o1      ! o1 <-- n, 2nd parameter
        call      merge, 0         ! Call merge
        nop                        !
        st      %g0, [%fp-24]      ! i <-- g0 (0)
.LL7:
        ld      [%fp-20], %o1      ! o1 <-- Memory[fp-24] (n)
        add      %o1, 9, %o0       ! o0 <-- o1 (n) + 9 (DATA1)
        ld      [%fp-24], %o1      ! o1 <-- i
        cmp      %o1, %o0          ! Compare i and n+9 (n+DATA1)
        bl      .LL10              ! if i < n+9 goto loop .LL10
        nop
        b      .LL8                ! else go out of loop .LL8
        nop
.LL10:
        sethi      %hi(mergedData), %o0    ! Starting address of the array
        or      %o0, %lo(mergedData), %o1  ! mergedData[]. o1 <-- mergedData
        ld      [%fp-24], %o0       ! o0 <-- i
        mov      %o0, %o3           ! o3 <-- o0
        sll      %o3, 2, %o2        ! o2 <-- 4*i
        sethi      %hi(.LLC3), %o3 ! Address of the format string
        or      %o3, %lo(.LLC3), %o0
        ld      [%o1+%o2], %o1      ! o1 <-- o1 + o2
                                    ! o1 <-- mergedData + 4*i
        call      printf, 0         ! Call printf
        nop
.LL9:
        ld      [%fp-24], %o0       ! o0 <-- i
        add      %o0, 1, %o1        ! o1 <-- o0 (i) + 1
        st      %o1, [%fp-24]       ! i <-- o1 (i+1)
        b      .LL7                 ! go .LL7 to continue loop
        nop
.LL8:
        sethi      %hi(.LLC4), %o1
        or      %o1, %lo(.LLC4), %o0
        call      printf, 0         ! Call to the last printf
        nop
        mov      0, %i0             ! Return value 0 is loaded
```

```
     b     .LL2
     nop
.LL2:
     ret                     ! Return
     restore                 ! Restores the stsck frame and
                             ! register window
.LLfe1:
     .size      main,.LLfe1-main  ! The size is the difference of labels
     .align 4
     .global merge                ! merge is a global symbol.
     .type      merge,#function   ! The type of merge is function.
     .proc     020
merge:
     !#PROLOGUE# 0
     save     %sp, -128, %sp     ! Shift the register window and
     !#PROLOGUE# 1               ! create the stack frame
     st     %i0, [%fp+68]        ! The two input parameters are saved
     st     %i1, [%fp+72]        ! i0 (dat) and i1 (m)
     st     %g0, [%fp-20]        ! i <-- 0
     ld     [%fp+72], %o0        ! o0 <-- m
     add     %o0, 8, %o1         ! o1 <-- m + 8 (DATA1 - 1)
     st     %o1, [%fp-24]        ! j <-- m + 8
.LL12:
     ld     [%fp-20], %o0        ! o0 <-- i
     ld     [%fp+72], %o1        ! o1 <-- m
     cmp     %o0, %o1            ! Compare i and m
     bl     .LL15                ! if i < m goto loop .LL15
     nop
     b     .LL13                 ! go out of loop .LL13
     nop
.LL15:
     sethi     %hi(data1), %o1   ! Store the address of data1
     or     %o1, %lo(data1), %o0 ! in o0
     ld     [%fp-24], %o1        ! o1 <-- j
     mov     %o1, %o2            ! o2 <-- o1 (j)
     sll     %o2, 2, %o1         ! o1 <-- 4*o2 (4*j)
     ld     [%fp-20], %o2        ! o2 <-- i
     mov     %o2, %o3            ! o3 <-- o2 (i)
     sll     %o3, 2, %o2         ! o2 <-- 4*o3 (4*i)
     ld     [%fp+68], %o3        ! o3 <-- dat
     add     %o2, %o3, %o2       ! o2 <-- dat + o2 (dat + 4*i)
     ld     [%o2], %o3           ! o3 <-- Memory[o2] (dat[i])
     st     %o3, [%o0+%o1]       ! Memory[o0+o1] <-- o3
                                 ! (data1[j] <-- dat[i])
.LL14:
     ld     [%fp-20], %o0        ! o0 <-- i
     add     %o0, 1, %o1         ! o1 <-- o0 + 1 (i+1)
```

```
    st     %o1, [%fp-20]        ! i <-- o1 (i+1)
    ld     [%fp-24], %o0        ! j <-- o0
    add     %o0, -1, %o1        ! o1 <-- o0 - 1 (j - 1)
    st     %o1, [%fp-24]        ! j <-- o1 (j - 1)
    b     .LL12                 ! go back to loop .LL12
    nop
.LL13:
    nop
    st     %g0, [%fp-28]      ! k <-- g0 (0)
    st     %g0, [%fp-20]      ! i <-- g0 (0)
    ld     [%fp+72], %o0      ! o0 <-- m
    add     %o0, 8, %o1       ! o1 <-- o0 + 8 (m + DATA1 - 1)
    st     %o1, [%fp-24]      ! j <-- o1 (m + DATA1 - 1)
.LL16:
    ld     [%fp+72], %o1      ! o1 <-- m
    add     %o1, 9, %o0       ! o0 <-- o1 + 9 (m + DATA1)
    ld     [%fp-28], %o1      ! o1 <-- k
    cmp     %o1, %o0          ! Compare k and m
    bl     .LL19              ! if k < m goto loop .LL19
    nop
    b     .LL17               ! else go out of loop .LL17
    nop
.LL19:
    sethi     %hi(data1), %o1 !
    or     %o1, %lo(data1), %o0   ! o0 <-- data1
    ld     [%fp-20], %o1          ! o1 <-- i
    mov     %o1, %o2              ! o2 <-- o1 (i)
    sll     %o2, 2, %o1           ! o1 <-- 4*o2 (4*i)
    sethi     %hi(data1), %o3     !
    or     %o3, %lo(data1), %o2   ! o2 <-- data1
    ld     [%fp-24], %o3          ! o3 <-- j
    mov     %o3, %o4              ! o4 <-- o3
    sll     %o4, 2, %o3           ! o3 <-- 4*o4 (4*j)
    ld     [%o0+%o1], %o0         ! o0 <-- Memory[o0+o1] (data1[i])
    ld     [%o2+%o3], %o1         ! o1 <-- Memory[o2+o3] (data1[j])
    cmp     %o0, %o1              ! Compare o0 and o1
                                  ! (data1[i] and data1[j])
    bge     .LL20                 ! if data[i] >= data1[j] goto else .LL20
    nop
    sethi     %hi(mergedData), %o1     ! if part
    or     %o1, %lo(mergedData), %o0   ! o0 <-- mergedData
    ld     [%fp-28], %o1           ! o1 <-- k
    mov     %o1, %o2               !
    sll     %o2, 2, %o1            ! o1 <-- 4*k
    sethi     %hi(data1), %o2      !
    or     %o2, %lo(data1), %o4    ! o4 <-- data1
    add     %fp, -20, %o2          ! o2 <-- fp - 20 (&i)
```

```
    ld     [%o2], %o3              ! o3 <-- Memory[o2] (i)
    mov     %o3, %o5               ! o5 <-- o3 (i)
    sll     %o5, 2, %g2            ! g2 <-- 4*o5
    ld     [%o4+%g2], %o4          ! o4 <-- Memory[o4+g2] (data1[i])
    st     %o4, [%o0+%o1]          ! Memory[o0+o1] <-- o4
                                   ! mergedData[k] <-- data1[i]
    add     %o3, 1, %o3            ! o3 <-- o3 + 1 (i+1)
    st     %o3, [%o2]              ! Memory[o2] <-- 03 (i++)
    b     .LL18                    ! end of if part goto .LL18 to continue
    nop
.LL20:                             ! else part
    sethi     %hi(mergedData), %o1
    or     %o1, %lo(mergedData), %o0 ! o0 <-- mergedData
    ld     [%fp-28], %o1           ! o1 <-- k
    mov     %o1, %o2               ! o2 <-- o1 (k)
    sll     %o2, 2, %o1            ! o1 <-- 4*k
    sethi     %hi(data1), %o2      !
    or     %o2, %lo(data1), %o4    ! o4 <-- data1
    add     %fp, -24, %o2          ! o2 <-- fp - 24 (&j)
    ld     [%o2], %o3              ! o3 <-- Memory[o2] (j)
    mov     %o3, %o5               ! o5 <-- o3 (j)
    sll     %o5, 2, %g2            ! g2 <-- 4*j
    ld     [%o4+%g2], %o4          ! o4 <-- data1[j]
    st     %o4, [%o0+%o1]          ! mergedData[k] <-- o4 (data[j])
    add     %o3, -1, %o3           ! o3 <-- o3 - 1 (j - 1)
    st     %o3, [%o2]              ! Memory[o2] <-- o3 (j--)
.LL21:
.LL18:
    ld     [%fp-28], %o0           ! o0 <-- k
    add     %o0, 1, %o1            ! o1 <-- o0 + 1 (k + 1)
    st     %o1, [%fp-28]           ! k <-- o1 (++k)
    b     .LL16                    ! go back to loop .LL16
     nop
.LL17:
.LL11:
    ret                            ! return
    restore                        ! restore stack and register window
.LLfe2:
    .size      merge,.LLfe2-merge  ! Size of the function merge
    .common     mergedData,400,4   ! 400 bytes are reserved for uninit
                                   ! mergedData in .common area (aligned to
                                   ! 4 byte boundary.
    .ident     "GCC: (GNU) 2.95.3 20010315 (release)"
```

12. If the return value is larger than int size, its space is allocated in the caller and a pointer is placed at the callers stack frame at `fp (new) + 64`.

```
/****************************************
 * Large return value: largeRetVal.c      *
 * **************************************/
#include <stdio.h>

typedef struct {
     int a, b, c, d, e, f ;
} what ;

what but() {
     what t ;

     t.a = 10 ; t.b = 20 ; t.c = 30 ; t.d = 40 ;
     return t ;
}

int main() {
     what a ;

     a = but() ;
     printf("%d\n", a.a) ;
     return 0 ;
}
```

```
! Return value is larger than int
     .file     "largeRetVal.c"
gcc2_compiled.:
.section     ".text"
     .align 4
     .global but
     .type      but,#function
     .proc     010
but:
     !#PROLOGUE# 0
     save     %sp, -136, %sp
     !#PROLOGUE# 1
     ld     [%fp+64], %o0
     mov     10, %o1
     st     %o1, [%fp-40]
     mov     20, %o1
     st     %o1, [%fp-36]
     mov     30, %o1
     st     %o1, [%fp-32]
     mov     40, %o1
     st     %o1, [%fp-28]
     ld     [%fp-40], %o1
     st     %o1, [%o0]
     ld     [%fp-36], %o1
     st     %o1, [%o0+4]
     ld     [%fp-32], %o1
     st     %o1, [%o0+8]
     ld     [%fp-28], %o1
     st     %o1, [%o0+12]
     ld     [%fp-24], %o1
     st     %o1, [%o0+16]
     ld     [%fp-20], %o1
     st     %o1, [%o0+20]
     b     .LL2
      nop
.LL2:
     mov     %o0, %i0
     jmp     %i7+12
     restore
.LLfe1:
     .size      but,.LLfe1-but
.section     ".rodata"
     .align 8
.LLC0:
     .asciz     "%d\n"
.section     ".text"
     .align 4
```

```
	.global main
	.type	 main,#function
	.proc	04
main:
	!#PROLOGUE# 0
	save	%sp, -136, %sp
	!#PROLOGUE# 1
	add	%fp, -40, %o0
	st	%o0, [%sp+64]
	call	but, 0
	nop
	unimp	24		! Unimplemented instruction with
				! immediate data 24 sizeof(what).
	sethi	%hi(.LLC0), %o1
	or	%o1, %lo(.LLC0), %o0
	ld	[%fp-40], %o1
	call	printf, 0
	nop
	mov	0, %i0
	b	.LL3
	nop
.LL3:
	ret
	restore
.LLfe2:
	.size	 main,.LLfe2-main
	.ident	"GCC: (GNU) 2.95.3 20010315 (release)"
```

13. First six int size formal parameters are saved in the caller's stack frame from `fp (new) + 68` to `fp (new) + 91` if it is necessary to have them in memory. For example the address of the formal pointer is required and that is not available if it is in the register.

```
/**********************************************
 * Address of the formal parameter is required   *
 *          addressOfFormalParam.c              *
 **********************************************/
#include <stdio.h>

extern int abc(int *p) ;
void xyz(int a) {
      int n = abc(&a) ;
}
```

```
     .file     "aOFP.c"
gcc2_compiled.:
.section     ".text"
     .align 4
     .global xyz
     .type      xyz,#function
     .proc     020
xyz:
     !#PROLOGUE# 0
     save     %sp, -120, %sp
     !#PROLOGUE# 1
     st     %i0, [%fp+68]
     add     %fp, 68, %o0          ! Address of the parameter is to be
                                   ! used (passed as parameter)
     call     abc, 0
     nop
     st     %o0, [%fp-20]
.LL2:
     ret
     restore
.LLfe1:
     .size      xyz,.LLfe1-xyz
     .ident     "GCC: (GNU) 2.95.3 20010315 (release)"
```