Memory Move Instructions

Store

```
st Ri,[Rj]            [Rj] := Ri
st Ri,[Rj+C]          [Rj+C] := Ri
```

Load

```
ld [Ri],Rj            Rj := [Ri]
ld [Ri+C],Rj          Rj := [Ri+C]
```

Move

The source register may also be replaced by a constant (i.e. `mov 5,R1`)

```
mov Ri,Rj             Rj := Ri
```

Mathematical Operations

The source registers may be replaced by constants (i.e. `add R1,5,R2`)

```
add Ri,Rj,Rk          Rk := Ri + Rj
sub Ri,Rj,Rk          Rk := Ri - Rj
mul Ri,Rj,Rk          Rk := Ri * Rj
div Ri,Rj,Rk          Rk := Ri / Rj
```

Branching Instructions

```
cmp Ri,Rj
```

Just like the mathematical operators, constants may be used as operands.

Branching instructions

```
b L
bg L
bge L
bl L
ble L
bne L
```

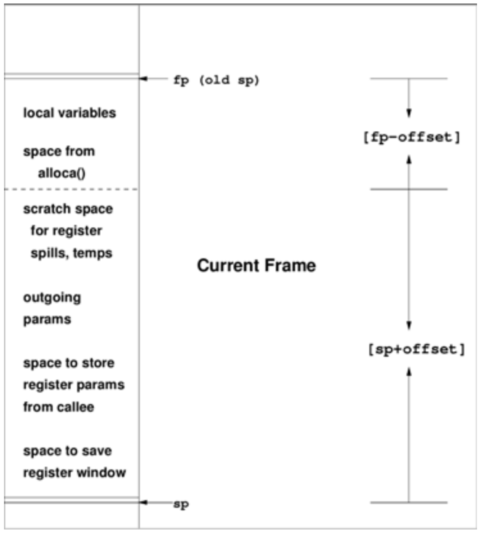
To express `if R1 <= 0 goto L1` we write

```
cmp R1,0
ble L1
```

Other Special Instructions

```
call L                R15:=pc; pc:=L
save sp,-C,sp         save registers,
                      allocating C bytes
                      on the stack
restore               restore registers
ret                   pc:=R15+8
nop                   do nothing
```

Stack Frame


fp (old sp)
local variables
space from alloca()
scratch space for register spills, temps
outgoing params
space to store register params from callee
space to save register window
sp
Current Frame
[fp-offset]
[sp+offset]

Stack Frames

- Store the function call hierarchy and the respective program memory;
- `sp` and `fp` point to stack frames;
- When a function is called a new stack frame is created:

  `push fp; fp := sp; sp := sp + C;`
- When a function returns, the top stack frame is popped:

  `sp := fp; fp = pop;`
- Local variables are stored relative to `fp`;
- The figure shows additional features of the SPARC architecture.

Writing VirtualRISC Code

```
int fact(int n) {
   int i, sum;
   sum = 1;
   i = 2;
   while (i <= n) {
      sum = sum * i;
      i = i + 1;
   }
   return sum;
}
```

Writing VirtualRISC Code

```
int fact(int n) {
   int i, sum;
   sum = 1;
   i = 2;
   while (i <= n) {
      sum = sum * i;
      i = i + 1;
   }
   return sum;
}
```

```
_fact:
  save sp,-112,sp     // save stack frame
  st R0,[fp+68]       // save arg n in frame of CALLER
  mov 1,R0            // R0 := 1
  st R0,[fp-16]       // [fp-16] is location for sum
  mov 2,R0            // RO := 2
  st RO,[fp-12]       // [fp-12] is location for i
L3:
  ld [fp-12],R0       // load i into R0
  ld [fp+68],R1       // load n into R1
  cmp R0,R1           // compare R0 to R1
  ble L5              // if R0 <= R1 goto L5
  b L4                // goto L4
L5:
  ld [fp-16],R0       // load sum into R0
  ld [fp-12],R1       // load i into R1
  mul R0,R1,R0        // R0 := R0 * R1
  st R0,[fp-16]       // store R0 into sum
  ld [fp-12],R0       // load i into R0
  add R0,1,R1         // R1 := R0 + 1
  st R1,[fp-12]       // store R1 into i
  b L3                // goto L3
L4:
  ld [fp-16],R0       // put return value of sum into R0
  restore             // restore register window
  ret                 // return from function
```