SPARC Instructions

First of all, SPARC instructions always put the destination last. For example, add %i1, %i2, %i3 adds registers %i1 and %i2 and puts the result in %i3.

Second, in describing the instructions we use c to denote a signed 13-bit constant; that is, $-4096 \leq c \leq 4095$. And we use %r1, %r2,... to denote registers.

1 Data Movement Instructions

The following instructions load from memory or store to memory:

\[
\begin{align*}
\text{ld} & [\%r1+c], \%r2 \\
\text{ld} & [\%r1+%r2], \%r3 \\
\text{st} & \%r1, [\%r2+c] \\
\text{st} & \%r1, [\%r2+\%r3]
\end{align*}
\]

The notation $[\%r1+c]$ denotes the memory location whose address is calculated by adding the contents of register %r1 and c. Similarly, $[\%r1+%r2]$ denotes the memory location whose address is calculated by adding the contents of registers %r1 and %r2.

Movement between registers is done by

\[
\text{mov} \quad %r1, %r2
\]

and a constant is loaded into a register by

\[
\text{mov} \quad c, %r1
\]

But if n is a constant that doesn’t fit into 13 bits, then it takes a two-instruction sequence to load n into a register:

\[
\begin{align*}
\text{sethi} & \%\text{hi}(n), %r1 \\
\text{or} & \quad %r1, \%\text{lo}(n), %r1
\end{align*}
\]

The pseudo-ops %hi(n) and %lo(n) return the high 22 bits and low 10 bits of n, respectively. Also, it’s worth remembering that register %g0 always contains 0.

2 Arithmetic Instructions

The following instructions do addition; note that all operands are either registers or 13-bit constants:

\[
\begin{align*}
\text{add} & \quad %r1, %r2, %r3 \\
\text{add} & \quad %r1, c, %r2
\end{align*}
\]

Similarly, the SPARC provides sub, smul, sdiv, and, or, sll (shift left logical), srl (shift right logical), sra (shift right arithmetic), and xor.
3 Branch Instructions

The target of a branch is a label, defined as follows:

\texttt{lab:}

A function \texttt{lab} is called by

\texttt{call lab}

Functions return with \texttt{ret}; they must also do \texttt{save} and \texttt{restore} as discussed before.

Branching on the SPARC is dependent on the value of the processor condition codes. These condition codes are set by certain arithmetic operations, such as \texttt{subcc}, but for our purposes it is enough to know about

\begin{verbatim}
cmp    %r1, %r2
cmp    %r1, c
\end{verbatim}

which compare two values and set the condition codes appropriately.

Once the condition codes have been set, you can branch to a label \texttt{lab} by one of the following instructions:

\begin{verbatim}
ba    lab
be    lab
bne   lab
bg    lab
bge   lab
bl    lab
ble   lab
\end{verbatim}

The instruction \texttt{ba} branches \textit{always}; the other instructions branch if the first value was respectively =, \#, >, \#, <, or \# to the second value.

An interesting feature of the SPARC is that (because of pipelining) the instruction \texttt{after} a branch is always executed before the branch takes effect; the position after a branch is known as the delay slot. The easiest thing to do is simply to put a \texttt{nop} instruction into every delay slot, but a careful compiler can often do useful work there. For example, if we want to call a function \texttt{f}, passing it parameter values 2 and 7, we could use a code sequence like

\begin{verbatim}
mov    2, %00
call   f
mov    7, %01     ! delay slot
\end{verbatim}