SPARC Architecture

• 8-bit cell (byte) is smallest addressable unit
• 32-bit addresses, i.e., 32-bit virtual address space
• Larger sizes: at address $A$

\[
\begin{array}{c}
A \\
7 \quad 0
\end{array}
\quad \text{byte}
\]

\[
\begin{array}{c}
A \quad A+1 \\
15
\end{array}
\quad \text{halfword}
\]

\[
\begin{array}{c}
A \quad A+1 \quad A+2 \quad A+3 \\
31
\end{array}
\quad \text{word}
\]

\[
\begin{array}{c}
A \quad A+1 \quad A+2 \quad A+3 \quad A+4 \quad A+5 \quad A+6 \quad A+7 \\
63
\end{array}
\quad \text{doubleword}
\]

• SPARC is a \textit{big-endian} (big end, or most-significant end, first) machine

SPARC Registers

• 32, 32-bit wide general-purpose registers

\[
\begin{array}{c}
%r0 \ldots \%r31 \\
%g0 \ldots %g7 \quad %r0 \ldots %r7 \quad \text{global} \\
%o0 \ldots %o7 \quad %r8 \ldots %r15 \quad \text{output} \\
%10 \ldots %17 \quad %r16 \ldots %r23 \quad \text{local} \\
%i0 \ldots %i7 \quad %r24 \ldots %r31 \quad \text{input}
\end{array}
\]

• The groups relate to procedure calling conventions

• Some registers have \textit{dedicated uses}

%sp (%r14) \quad \text{stack pointer}
%fp (%r30) \quad \text{frame pointer}
%r15 \quad \text{temporary}
%r31 \quad \text{return address}

• Register %g0 always has the value 0 when read; writing it has no effect

• Other special registers (manipulated by special instructions):
  - floating point registers (%f0...%f31)
  - program counter (pc)
  - next program counter (npc)
  - PSR, TBR, WIM, Y
### SPARC Register Map

- See §2.2 in Paul

<table>
<thead>
<tr>
<th>Register</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%i7</code>, <code>%r31</code></td>
<td>return address - 8</td>
</tr>
<tr>
<td><code>%i6</code>, <code>%fp</code>, <code>%r30</code></td>
<td>frame pointer</td>
</tr>
<tr>
<td><code>%i5</code>, <code>%r29</code></td>
<td>incoming parameter 6</td>
</tr>
<tr>
<td><code>...</code></td>
<td><code>...</code></td>
</tr>
<tr>
<td><code>%i0</code>, <code>%r24</code></td>
<td>incoming parameter 1/return value to caller</td>
</tr>
<tr>
<td><code>%i7</code>, <code>%r23</code></td>
<td>local 7</td>
</tr>
<tr>
<td><code>...</code></td>
<td><code>...</code></td>
</tr>
<tr>
<td><code>%i0</code>, <code>%r16</code></td>
<td>local 0</td>
</tr>
<tr>
<td><code>%o7</code>, <code>%r15</code></td>
<td>temporary value/address of call instruction</td>
</tr>
<tr>
<td><code>%o6</code>, <code>%sp</code>, <code>%r14</code></td>
<td>stack pointer</td>
</tr>
<tr>
<td><code>%o5</code>, <code>%r13</code></td>
<td>outgoing parameter 6</td>
</tr>
<tr>
<td><code>...</code></td>
<td><code>...</code></td>
</tr>
<tr>
<td><code>%o0</code>, <code>%r8</code></td>
<td>outgoing parameter 1/return value from caller</td>
</tr>
<tr>
<td><code>%g7</code>, <code>%r7</code></td>
<td>global 7</td>
</tr>
<tr>
<td><code>...</code></td>
<td><code>...</code></td>
</tr>
<tr>
<td><code>%g1</code>, <code>%r1</code></td>
<td>temporary value</td>
</tr>
<tr>
<td><code>%g0</code>, <code>%r0</code></td>
<td>0</td>
</tr>
</tbody>
</table>

### SPARC Register Map, cont’d

- Other registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%y</code></td>
<td>Y register</td>
</tr>
<tr>
<td><code>%psr</code></td>
<td>integer condition codes</td>
</tr>
<tr>
<td><code>%fsr</code></td>
<td>floating point condition codes</td>
</tr>
<tr>
<td><code>%csr</code></td>
<td>coprocessor condition codes</td>
</tr>
<tr>
<td><code>%f31</code></td>
<td>floating point value</td>
</tr>
<tr>
<td><code>...</code></td>
<td><code>...</code></td>
</tr>
<tr>
<td><code>%f0</code></td>
<td>floating point value</td>
</tr>
</tbody>
</table>

- Register save conventions: what happens across calls?

- `%g2` ... `%g7` saved?
- `%g1` destroyed
- `%o0` ... `%o5`, `%o7` destroyed
- `%o6` saved
- `%10` ... `%17` saved
- `%i0` ... `%i7` saved
- `%f0` ... `%f31` saved
SPARC Instruction Set

- **Instruction groups**
  - load/store instructions
  - integer arithmetic and bitwise logical instructions
  - control instructions (branches, calls)
  - special instructions (operating system)
  - floating point arithmetic and conversion

- **Instruction formats (see Ch. 8 in Paul)**
  - format 1 ($op = 1$): **call**
    
    \[
    \begin{array}{c|c}
    op & \text{disp30} \\
    \hline
    31 & 29 \\
    \end{array}
    \]
  
  - format 2 ($op = 0$): **sethi** and branches (**bicc**, **fbfcc**, **cbccc**)
    
    \[
    \begin{array}{c|c|c|c}
    \text{op} & \text{a} & \text{cond} & \text{op2} \\
    \hline
    \text{disp22} & 24 & 21 \\
    \end{array}
    \]
  
  - format 3 ($op = 2$ or $3$): remaining instructions
    
    \[
    \begin{array}{c|c|c|c|c|c|c}
    \text{op} & \text{rd} & \text{op3} & \text{rs1} & \text{i=0} & \text{asi} & \text{rs2} \\
    \hline
    \text{simm13} & 18 & 13 & 12 & 4 \\
    \end{array}
    \]

Assembly vs. Machine Language

- **Machine language** is the **bit patterns** that represent instructions

- **Assembly language** is a **symbolic representation** of machine language

- **Assemblers** translate from assembly language to machine language
  
  \[\text{add } \%i1,360,\%o2\] is a format 3 instruction: **022401460550**

- **Disassemblers** translate from machine language to an assembly language

- Assemblers: mapping an assembly instruction to a machine instruction (1-to-1)

- Compilers: mapping a statement to 1 or many assembly instructions
Load Instructions

- Load: move data from memory to a register

  ![Load Instruction Diagram]

  alternate address space (privileged)

  \[ \text{ld} \{ \text{address} \}, \text{reg} \]

  signed or unsigned

  byte, halfword, word, double

- Fetched byte or halfword appears right-justified in the 32-bit register
- Leftmost bits are zero-filled or sign-extended
- A double word is loaded into a register \textit{pair and reg} must be even
  - the most-significant word lands in \textit{reg}
  - the least-significant word in \textit{reg} + 1
- Addresses must be \textit{aligned} for address \textit{A}
  - halfword \( A \mod 2 = 0 \)
  - word \( A \mod 4 = 0 \)
  - double word \( A \mod 8 = 0 \)

Store Instructions

- Move data from a register to memory

  ![Store Instruction Diagram]

  alternate address space (privileged)

  \[ \text{st} \{ \text{address} \}, \text{reg} \]

  byte, halfword, word, doubleword

- Storing bytes and halfwords
  - the rightmost bits are stored
  - the leftmost bits are ignored

- Storing double words
  - \textit{reg} must be even
Addressing Modes

• SPARC has two addressing modes to yield an effective address:
  1. add the contents of two registers
  2. add the contents of a register and a signed, 13-bit number

• Common names
  1. register indirect or deferred
     \texttt{ld} \%o1,\%o2
  1. register indexed (above is a special case that uses \%g0)
     \texttt{st} \%o1,\%o2+\%o3
  2. register displacement or based
     \texttt{ld} \%o1+10,\%o2

• Assembly-language syntax: \(N\) is a 13-bit integer constant

<table>
<thead>
<tr>
<th>address</th>
<th>synonym</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{reg}</td>
<td>\texttt{reg + %g0}</td>
</tr>
<tr>
<td>\texttt{reg + reg}</td>
<td>\texttt{reg + N}</td>
</tr>
<tr>
<td>\texttt{reg +} (N)</td>
<td>\texttt{%g0 + N}</td>
</tr>
</tbody>
</table>