Machine Architecture
CS 217

Computer Organization

Memory Hierarchy

- Registers
  ~128, 1-5ns access time (CPU cycle time)
- Cache
  1KB – 4MB, 20-100ns (multiple levels)
- Memory
  64MB – 1GB, 200ns
- Disk
  1GB – 20GB, 10ms
- Long-term Storage
  1TB, 1-10s
Source to Binary

- Source code in some high-level language
  \[ x = a + b; \]
- Assembly language
  
  ```
  ld a, %r1
  ld b, %r2
  add %r1, %r2, %r3
  st %r3, x
  ```
- Machine language
  
  32-bit instructions (bit patterns) executed by the machine

Instruction Formats

- Each machine instruction is composed of...
  - **opcode**: operation to be performed
  - **operand(s)**: data that is operated upon
- Each machine supports a few formats...
  
  ```
  opcode
  opcode dst
  opcode src dst
  opcode src1 src2 dst
  ```

Instruction Execution

- CPU’s control unit executes a loop
  - fetch: fetch at PC; increment PC
  - decode: interpret instruction format
  - operand fetch: load operands into registers
  - execute: perform instruction opcode
  - store: write results to memory
Addressing Memory

- 8-bit byte is the smallest addressable unit
- 32-bit addresses; thus 32-bit address space
- Doubleword too
- Sparc is big-endian

\[ A \quad A+1 \quad A+2 \quad A+3 \]

Sparc Registers

- 32 x 32-bit general-purpose registers
  \( r_0 \ldots r_{31} \)
- Register map
  \[
  \begin{array}{ccc}
  g0 & \ldots & g7 \\
  o0 & \ldots & o7 \\
  i0 & \ldots & i17 \\
  l0 & \ldots & l17 \\
  s0 & \ldots & s15 \\
  f0 & \ldots & f31 \\
  \end{array}
  \]
  global
  output
  local
  input
- Some registers have dedicated uses
  \%sp (\%r14, \%o6) stack pointer
  \%fp (\%r30, \%i6) frame pointer
  \%r15 temporary
  \%r31 return address
  \%r0 (\%r0) always 0

Sparc Registers (cont)

- Special-purpose registers
  manipulated by special instructions
  floating point registers (\%f0 \ldots \%f31)
  program counter (PC)
  next program counter (nPC)
  PSR, TBR, WIM, Y
**Sparc Instruction Set**

- Instruction groups
  - load/store instructions
  - integer arithmetic and bit-wise logical instructions
  - control transfer instructions
  - special instructions (used by OS)
  - floating point arithmetic

**Instruction Set (cont)**

- Format 1 (op = 1): `call`
  - `op disp30`

- Format 2 (op = 0): `sethi` and branches
  - `op rd op2 imm22`
  - `op a cond op2 disp22`

**Instruction Set (cont)**

- Format 3 (op = 2 or 3): remaining instructions
  - `op rd op3 rs1 sh shift simm13`
  - `op rd op3 rs1 sh simm13`
  - `op rd op3 rs1 `
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 to machine language
  mapping is 1-to-1
- Compilers map from source to assembly
  mapping is 1-to-many

Assembly vs Machine (cont)

- Example
  \[ \text{add } \%\text{i1}, 360, \%\text{o2} \]

\[
\begin{array}{cccccc}
2 & 10 & 0 & 26 & 1 & 360 \\
2 & 12 & 0 & 31 & 1 & 550 \\
\end{array}
\]

(decimal)

(octal)

100101000000110011000101101000

Addressing Modes

- Two modes to yield effective address
  - add contents of two registers
    \[ \text{ld } \%\text{o1}, \%\text{o2} \] register indirect
    \[ \text{st } \%\text{o1}, [\%\text{o2}, \%\text{o3}] \] register indexed
  - add contents of register and immediate
    \[ \text{ld } \%\text{o1+10}, \%\text{o2} \] base displacement
### Addressing Modes (cont)

- **Assembly language syntax**

<table>
<thead>
<tr>
<th>Address</th>
<th>Synonym</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg</td>
<td>reg + %g0</td>
</tr>
<tr>
<td>reg + reg</td>
<td>reg + N</td>
</tr>
<tr>
<td>reg + N</td>
<td>N + reg</td>
</tr>
<tr>
<td>N + reg</td>
<td>reg + N</td>
</tr>
<tr>
<td>N</td>
<td>%g0 + N</td>
</tr>
</tbody>
</table>

  where $N$ is a 13-bit, signed, integer constant