CH11 Instruction Sets:
Addressing Modes and Formats

- Software and Hardware interface
  - Addressing
  - Pentium and PowerPC Addressing Modes
  - Instruction Formats
  - Pentium and PowerPC Instruction Formats

Addressing Modes

- Immediate
- Direct
- Indirect
- Register
- Register Indirect
- Displacement (Indexed)
- Stack

Immediate Addressing

- Operand is part of instruction
- Operand = address field
- e.g. ADD 5
  - Add 5 to contents of accumulator
  - 5 is operand
- No memory reference to fetch data
- Fast
- Limited range

Immediate Addressing Diagram

```
<table>
<thead>
<tr>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
</tr>
</tbody>
</table>
```

Direct Addressing

- Address field contains address of operand
- Effective address (EA) = address field (A)
- e.g. ADD A
  - Add contents of cell A to accumulator
  - Look in memory at address A for operand
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space

Direct Addressing Diagram

```
<table>
<thead>
<tr>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
</tr>
</tbody>
</table>
```

Memory

Operand
Indirect Addressing (1)

- Memory cell pointed to by address field contains the address of (pointer to) the operand
- EA = (A)
  - Look in A, find address (A) and look there for operand
- e.g. ADD (A)
  - Add contents of cell pointed to by contents of A to accumulator

Indirect Addressing (2)

- Large address space
- $2^n$ where $n =$ word length
- May be nested, multilevel, cascaded
  - e.g. EA = (((A)))
    - Draw the diagram yourself
- Multiple memory accesses to find operand
- Hence slower

Register Addressing (1)

- Operand is held in register named in address filed
- EA = R
- Limited number of registers
- Very small address field needed
  - Shorter instructions
  - Faster instruction fetch

Register Addressing (2)

- No memory access
- Very fast execution
- Very limited address space
- Multiple registers helps performance
  - Requires good assembly programming or compiler writing
  - N.B. C programming
    - register int a;
- c.f. Direct addressing

Indirect Addressing Diagram

Register Addressing Diagram
Register Indirect Addressing
- C.f. indirect addressing
- EA = (R)
- Operand is in memory cell pointed to by contents of register R
- Large address space (2^n)
- One fewer memory access than indirect addressing

Displacement Addressing
- EA = A + (R)
- Address field hold two values
  - A = base value
  - R = register that holds displacement
  - or vice versa

Relative Addressing
- A version of displacement addressing
- R = Program counter, PC
- EA = A + (PC)
- i.e. get operand from A cells from current location pointed to by PC
- c.f locality of reference & cache usage

Base-Register Addressing
- A holds displacement
- R holds pointer to base address
- R may be explicit or implicit
- e.g. segment registers in 80x86
Indexed Addressing
- A = base
- R = displacement
- EA = A + R
- Good for accessing arrays
  - EA = A + R
  - R++

Combinations
- Postindex
  - EA = (A) + (R)
- Preindex
  - EA = (A+(R))
  - (Draw the diagrams)

Stack Addressing //
- Operand is (implicitly) on top of stack
- e.g.
  - ADD Pop top two items from stack and add

Pentium Addressing Modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>Operand = A</td>
</tr>
<tr>
<td>Register</td>
<td>LA = R</td>
</tr>
<tr>
<td>Displacement</td>
<td>LA = (SR) + A</td>
</tr>
<tr>
<td>Base</td>
<td>LA = (SR) + (R)</td>
</tr>
<tr>
<td>Base with Displacement</td>
<td>LA = (SR) + (B) + A</td>
</tr>
<tr>
<td>Scaled Index with Displacement</td>
<td>LA = (SR) + (I) x S + A</td>
</tr>
<tr>
<td>Base with Index and Displacement</td>
<td>LA = (SR) + (I) + (I) + A</td>
</tr>
<tr>
<td>Base with Scaled Index and Displacement</td>
<td>LA = (SR) + (I) x S + (B) + A</td>
</tr>
<tr>
<td>Relative</td>
<td>LA = (PC) + A</td>
</tr>
</tbody>
</table>

PowerPC Addressing Modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load/Store Addressing</td>
<td></td>
</tr>
<tr>
<td>Indirect</td>
<td>EA = (BR) + D</td>
</tr>
<tr>
<td>Indirect Indexed</td>
<td>EA = (BR) + (IR)</td>
</tr>
<tr>
<td>Absolute</td>
<td>EA = 1</td>
</tr>
<tr>
<td>Relative</td>
<td>EA = (PC) + 1</td>
</tr>
<tr>
<td>Indirect</td>
<td>EA = (UCR)</td>
</tr>
<tr>
<td>Fixed-Point Computation</td>
<td></td>
</tr>
<tr>
<td>Register</td>
<td>EA = GPR</td>
</tr>
<tr>
<td>Immediate</td>
<td>Operand = 1</td>
</tr>
<tr>
<td>Floating-Point Computation</td>
<td></td>
</tr>
<tr>
<td>Register</td>
<td>EA = FPR</td>
</tr>
</tbody>
</table>
**Instruction Length**

- Affected by and affects:
  - Memory size
  - Memory organization
  - Bus structure
  - CPU complexity
  - CPU speed
- Trade off between powerful instruction repertoire and saving space

**Allocation of Bits**

- Number of addressing modes
- Number of operands
- Register versus memory
- Number of register sets
- Address range
- Address granularity

**Pentium Instruction Format**

(a) Instruction

**PowerPC Instruction Formats**

(a) Branch Instructions

(b) Condition Register Logical Instructions

(c) Load/Store Instructions

**PowerPC Instruction Formats 2**

(d) Integer Arithmetic, Logical, and Shift/rotate Instructions

(e) Floating-Point Arithmetic Instructions

**Foreground Reading**

- Stallings chapter 10
- Intel and PowerPC Web sites