CH10 Instruction Sets: Characteristics and Functions

- Software and Hardware interface
  - Machine Instruction Characteristics
  - Types of Operands
  - Pentium II and PowerPC Data Types
  - Types of Operations
  - Pentium II and PowerPC Operation Types
  - Assembly Language

What is an instruction set?
- The complete collection of instructions that are understood by a CPU
- Machine Code
- Binary
- Usually represented by assembly codes

Elements of an Instruction
- Operation code (Op code)
  - Do this
- Source Operand reference
  - To this
- Destination (Result) Operand reference
  - Put the answer here
- Next Instruction Reference
  - When you have done that, do this...

Instruction Representation
- In machine code each instruction has a unique bit pattern
- For human consumption (well, programmers anyway) a symbolic representation is used
  - e.g. ADD, SUB, LOAD
- Operands can also be represented in this way
  - ADD A, B

Instruction Types
- Data processing
- Data storage (main memory)
- Data movement (I/O)
- Program flow control

Number of Addresses (a)
- 3 addresses
  - Operand 1, Operand 2, Result
  - a = b + c;
- May be a forth - next instruction (usually implicit)
  - Not common
  - Needs very long words to hold everything
Number of Addresses (b)
- 2 addresses
  - One address doubles as operand and result
  - $a = a + b$
  - Reduces length of instruction

Number of Addresses (c)
- 1 address
  - Implicit second address
  - Usually a register (accumulator)
  - Common on early machines

Number of Addresses (d)
- 0 (zero) addresses
  - All addresses implicit
  - Uses a stack
  - e.g. push $a$
    - push $b$
    - add
    - pop $c$
  - $c = a + b$

How Many Addresses
- More addresses
  - More complex (powerful?) instructions
  - More registers
    - Inter-register operations are quicker
  - Fewer instructions per program
- Fewer addresses
  - Less complex (powerful?) instructions
  - More instructions per program
  - Faster fetch/execution of instructions

Design Decisions (1)
- Operation repertoire
  - How many ops?
  - What can they do?
  - How complex are they?
- Data types
- Instruction formats
  - Length of op code field
  - Number of addresses

Design Decisions (2)
- Registers
  - Number of CPU registers available
  - Which operations can be performed on which registers?
- Addressing modes (later…)
- RISC v CISC
Types of Operand

- Addresses
- Numbers
  - Integer/floating point
- Characters
  - ASCII etc.
- Logical Data
  - Bits or flags
  - (Aside: Is there any difference between numbers and characters? Ask a C programmer!)

Pentium Processors

- Pentium: superscaler techniques allowing multiple instructions to execute in parallel
- Pentium Pro: branch prediction, speculative execution
- Pentium II: MMX technology to process video, audio, and graphics
- Pentium III: additional floating-point instruction to support 3D graphics software.

Pentium Data Types

- 8 bit Byte
- 16 bit word
- 32 bit double word
- 64 bit quad word
- Addressing is by 8 bit unit
- A 32 bit double word is read at addresses divisible by 4

Specific Data Types

- General - arbitrary binary contents
- Integer - single binary value
- Ordinal - unsigned integer
- Unpacked BCD - One digit per byte
- Packed BCD - 2 BCD digits per byte
- Near Pointer - 32 bit offset within segment
- Bit field
- Byte String
- Floating Point

Types of Operation

- Data Transfer
- Arithmetic
- Logical
- Conversion
- I/O
- System Control
- Transfer of Control
Data Transfer
• Specify
  › Source
  › Destination
  › Amount of data
• May be different instructions for different movements
  › e.g. IBM 370
• Or one instruction and different addresses
  › e.g. VAX

Arithmetic
• Add, Subtract, Multiply, Divide
• Signed Integer
• Floating point?
• May include
  › Increment (a++)
  › Decrement (a--)
  › Negate (-a)

Logical
• Bitwise operations
• AND, OR, NOT

Conversion
• E.g. Binary to Decimal

Input/Output
• May be specific instructions
• May be done using data movement instructions (memory mapped)
• May be done by a separate controller (DMA)

Systems Control
• Privileged instructions
• CPU needs to be in specific state
  › Ring 0 on 80386+
  › Kernel mode
• For operating systems use
Transfer of Control

- **Branch**
  - e.g. branch to x if result is zero
- **Skip**
  - e.g. increment and skip if zero
  - 1SZ Register
  - Branch xxxx
- **ADD A**
- **Subroutine call**
  - c.f. interrupt call

Common Instruction Set 1

<table>
<thead>
<tr>
<th>Type</th>
<th>Operation Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move (transfer)</td>
<td>Transfer word or block from source to destination</td>
<td></td>
</tr>
<tr>
<td>Store</td>
<td>Transfer word from processor to memory</td>
<td></td>
</tr>
<tr>
<td>Load (fetch)</td>
<td>Transfer word from memory to processor</td>
<td></td>
</tr>
<tr>
<td>Exchange</td>
<td>Swap contents of source and destination</td>
<td></td>
</tr>
<tr>
<td>Clear (reset)</td>
<td>Transfer word of 0s to destination</td>
<td></td>
</tr>
<tr>
<td>Set</td>
<td>Transfer word of 1s to destination</td>
<td></td>
</tr>
<tr>
<td>Push</td>
<td>Transfer word from source to top of stack</td>
<td></td>
</tr>
<tr>
<td>Pop</td>
<td>Transfer word from top of stack to destination</td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>Computer sum of two operands</td>
<td></td>
</tr>
<tr>
<td>Subtract</td>
<td>Compute difference of two operands</td>
<td></td>
</tr>
<tr>
<td>Multiply</td>
<td>Compute product of two operands</td>
<td></td>
</tr>
<tr>
<td>Divide</td>
<td>Compute quotient of two operands</td>
<td></td>
</tr>
<tr>
<td>Absolute</td>
<td>Replace operand by its absolute value</td>
<td></td>
</tr>
<tr>
<td>Negate</td>
<td>Change sign of operand</td>
<td></td>
</tr>
<tr>
<td>Increment</td>
<td>Add 1 to operand</td>
<td></td>
</tr>
<tr>
<td>Decrement</td>
<td>Subtract 1 from operand</td>
<td></td>
</tr>
</tbody>
</table>

Common Instruction Set 2

<table>
<thead>
<tr>
<th>Logical</th>
<th>Operation Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Perform the specified logical operation bitwise</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>Perform the specified logical operation bitwise</td>
<td></td>
</tr>
<tr>
<td>NOT (Complement)</td>
<td>Perform the specified logical operation bitwise</td>
<td></td>
</tr>
<tr>
<td>Exclusive OR</td>
<td>Perform the specified logical operation bitwise</td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td>Test specified conditions and flags based on outcome</td>
<td></td>
</tr>
<tr>
<td>Compare</td>
<td>Compare two or more operands and set flags based on outcome</td>
<td></td>
</tr>
<tr>
<td>Set Control Variables</td>
<td>Set control flags for protection purposes.</td>
<td></td>
</tr>
<tr>
<td>Shift</td>
<td>Shift the operand left or right</td>
<td></td>
</tr>
<tr>
<td>Rotate</td>
<td>Rotate the operand</td>
<td></td>
</tr>
<tr>
<td>Jump (branch)</td>
<td>Unconditional transfer; load PC with specified address</td>
<td></td>
</tr>
<tr>
<td>Jump Conditional</td>
<td>Test specified condition; either load PC with specified address or do nothing, based on condition</td>
<td></td>
</tr>
</tbody>
</table>

Common Instruction Set 3

| Jump (branch) | Unconditional transfer; load PC with specified address   |
| Jump Conditional | Test specified condition; either load PC with specified address or do nothing, based on condition |

Transfer of Control

- **Jump to Subroutine**
  - Place current program control information in known location; jump to specified address
- **Return**
  - Replace contents of PC and other registers from known location
- **Execute**
  - Fetch operand from specified location and execute as instruction. Do not modify PC
- **Skip**
  - Increment PC to skip next instruction
- **Skip Conditional**
  - Test specified condition; either skip or do nothing based on condition
- **Halt**
  - Stop program execution. Program condition is satisfied
- **Wait (hold)**
  - Stop program execution. Program condition is not satisfied; resume execution when condition is satisfied
- **No operation**
  - No operation is performed, but program execution is continued

Common Instruction Set 4

| Input (read) | Transfer data from specified (I/O) port or device to destination, e.g., main memory or processor register |
| Output (write) | Transfer data from specified source to I/O port or device       |
| Start I/O | Transfer instructions to I/O processor to initiate I/O operation |
| Test I/O | Transfer status information from I/O system to specified destination |
| Translate | Translate values in a section of memory based on a table of correspondences |
| Conversion | Convert the contents of a word from one form to another (e.g., packed decimal to binary) |

Byte Order

- **What order do we read numbers that occupy more than one byte**
- e.g. (numbers in hex to make it easy to read)
- 12345678 can be stored in 4x8bit locations as follows
## Byte Order (example)

<table>
<thead>
<tr>
<th>Address</th>
<th>Value (1)</th>
<th>Value (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>184</td>
<td>12</td>
<td>78</td>
</tr>
<tr>
<td>185</td>
<td>34</td>
<td>56</td>
</tr>
<tr>
<td>186</td>
<td>56</td>
<td>34</td>
</tr>
<tr>
<td>186</td>
<td>78</td>
<td>12</td>
</tr>
</tbody>
</table>

- i.e. read top down or bottom up?

## Byte Order Names

- The problem is called Endian
- The system on the left has the least significant byte in the lowest address
- This is called big-endian
- The system on the right has the least significant byte in the highest address
- This is called little-endian

## Standard...What Standard?

- Pentium (80x86), VAX are little-endian
- IBM 370, Motorola 680x0 (Mac), and most RISC are big-endian
- PowerPC supports both!
- Internet is big-endian
  - Makes writing Internet programs on PC more awkward!
  - WinSock provides htoi and itoh (Host to Internet & Internet to Host) functions to do the conversion