Overview

- Instruction format
- AVR instruction format examples
- PowerPC instruction format examples

Instruction Formats

- Instructions typically consist of
  - Opcode (Operation code)
    - defines the operation (e.g. addition)
  - Operands
    - what’s being operated on (e.g. particular registers or memory address)
- There are many different formats for instructions

<table>
<thead>
<tr>
<th>Instruction Formats</th>
<th>Instruction Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Format Diagram" /></td>
<td><img src="image" alt="Format Diagram" /></td>
</tr>
<tr>
<td>(a)</td>
<td>(b)</td>
</tr>
<tr>
<td>(c)</td>
<td>(d)</td>
</tr>
</tbody>
</table>

- Instructions typically have 0, 1, 2 or 3 operands
  - Could be memory addresses, constants, register addresses (i.e. register numbers)
AVR Instruction Examples

• Clear register.
  Syntax: clr Rd
  Operand: $0 \leq d \leq 31$
  Operation: $Rd \leftarrow 0$

• Instruction format.
  $$\begin{array}{cccccccccccc}
  0 & 0 & 1 & 0 & 0 & 1 & d & d & d & d & d & d & 0 \\
  \end{array}$$
  – OpCode uses 6 bits (bit 9 to bit 15).
  – The only operand uses the remaining 10 bits (only 5 bits (bit 0 to bit 4) are actually needed).

• Subtraction with carry.
  Syntax: sbc Rd, Rr
  Operation: $Rd \leftarrow Rd - Rr - C$
  Rd: Destination register. $0 \leq d \leq 31$
  Rr: Source register. $0 \leq r \leq 31$
  C: Carry

• Instruction format.
  $$\begin{array}{cccccccccccc}
  0 & 0 & 0 & 0 & 1 & 0 & r & d & d & d & d & d & d & r & r & r & 0 \\
  \end{array}$$
  – OpCode uses 6 bits (bit 9 to bit 15).
  – Two operands share the remaining 10 bits.

Instruction Lengths

• On some machines – instructions are all the same length
• On other machines – instructions can have different lengths

AVR Instruction Examples

• Almost all instructions are 16 bits long.
  – add Rd, Rr
  – sub Rd, Rr
  – mul Rd, Rr
  – brge k

• Few instructions are 32 bits long.
  – lds Rd, k ($0 \leq k \leq 65535$)
    loads 1 byte from the SRAM to a register.
Design Criteria for Instruction Formats

1. Backwards Compatibility
   - e.g. Pentium 4 supports various instruction lengths so as to be compatible with 8086

2. Instruction Length
   - Ideally (if you’re starting from scratch)
     - All instructions same length
     - Short instructions are better (less memory needed to store programs and can read instructions in from memory faster)

Instruction Design Criteria (cont.)

3. Room to express operations
   - $2^n$ operations needs at least $n$ bits
   - Wise to allow room to add additional opcodes for next generation of CPU

4. Number of operand bits in instruction
   - Do you address bytes or words?

OpCode ↔ Operand Tradeoffs

- Instructions can tradeoff number of OpCode bits against number of operand bits
- Example:
  - 16 bit instructions
  - 16 registers (i.e. 4-bit register addresses)
  - Instructions could be formatted like this:

```
  15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

- OpCode  Operand1  Operand2  Operand3

- But what if we need more instructions and some instructions only operate on 0, 1 or 2 registers?

Expanding OpCodes

- Some OpCodes can mean “look elsewhere in the instruction for the real OpCode”
  - e.g. if first 4 bits are 1111, OpCode is really contained in next 4 bits (i.e. effectively an 8 bit OpCode), and so on
Expanding OpCodes

- Other combinations are possible
- Exercise (two minutes)

For a 16 bit instruction machine with 16 registers, design OpCodes that allow for

- 14 3-operand instructions
- 30 2-operand instructions
- 30 1-operand instructions
- 32 0-operand instructions

PowerPC Examples

- PowerPC ISA defines OpCode as the first six bits
  - This specifies type of instruction (operation)
  - OpCode specifies format of the rest of the instruction

PowerPC Machine Instruction Example 1

- OpCode (001110 two or 14) tells us that this instruction is an integer addition:
  
  destination-register = source-register + value
  
  \[ r5 = r12 + (-1) \]

PowerPC Machine Instruction Example 2

- OpCode (111111 two or 63) tells us this is a double precision floating point instruction
  - But it does not tell us what it actually does!
  - We need to look at a Secondary OpCode
PowerPC Machine Instruction Example 2

- Secondary OpCode (42) tells us this is a double precision floating point addition
  - destination-reg = register-A + register-B
  - \( fr13 = fr26 + fr6 \)