Instruction Set Architecture (ISA)

- ISA is the interface between hardware and software
- For (machine language) programmers (and compiler writers)
  - Don’t need to know (much) about microarchitecture
  - Just write or generate instructions that match the ISA
- For hardware (microarchitecture) designers
  - Don’t need to know about the high level software
  - Just build a microarchitecture that implements the ISA

What makes an ISA?

#1: Memory Models

- Memory model: how does memory look to CPU?
- Issues
  1. Addressable cell size
  2. Alignment
  3. Address spaces
  4. Endianness

What makes an ISA?

- Memory models
- Registers
- Data types
- Instructions
1. Addressable Cell Size

- Memory has cells, each of which has an address
- Most common cell size is 8 bits (1 byte)
- But not always!
  - AVR Instruction memory has 16 bit cells
- Note – the data bus may be wider
  - i.e. retrieve several cells (addresses) at once

2. Alignment

- Many architectures require **natural alignment**, e.g.
  - 4-byte words starting at addresses 0,4,8, …
  - 8-byte words starting at addresses 0, 8, 16, …

3. Address Spaces

- Princeton architecture or Von Neumann architecture (most used).
  - A single linear address space for both instructions and data
  - e.g. $2^{32}$ bytes numbered from 0 to $2^{32} - 1$
    » (may not be bytes – depends on addressable cell size)
- Harvard architecture
  - Separate address spaces for instructions and data
  - AVR AT90S8515
    - Data address space: up to $2^{16}$ bytes
    - Instruction address space: $2^{12}$ 16-bit words
### AVR Address Spaces

**Program Memory**
- Program Flash Memory (1K bytes~128K bytes)

**Data Memory**
- 32 General purpose Working Registers
- 64 Input/Output Registers
- Internal SRAM (128~4K bytes)
- External SRAM

#### End Address
- 0x0000
- 0x1F
- 0x20
- 0x5F
- 0x60

#### End Address
- 8 bits

### AVR Address Spaces (cont.)

**Data EEPROM Memory**
- EEPROM Memory (64~4K bytes)

#### End address
- 8 bits

### 4. Endianness

- Different machines may support different byte orderings
- Two orderings:
  - Little endian – little end (least significant byte) stored first (at lowest address)
    - Intel microprocessors (Pentium etc)
  - Big endian – big end stored first
    - SPARC, Motorola microprocessors
- Most CPUs produced since ~1992 are “bi-endian” (support both)
  - some switchable at boot time
  - others at run time (i.e. can change dynamically)

### What makes an ISA?

#### #2: Registers

- Two types
  - General purpose
    - Used for temporary results etc
  - Special purpose, e.g.
    - Program Counter (PC)
    - Stack pointer (SP)
    - Input/Output Registers
    - Status Register
Registers (cont.)

- Some other registers are part of the microarchitecture NOT the ISA
  - Instruction Register (IR)
  - Memory Address Register (MAR)
  - Memory Data Register (MDR)
  - i.e. programmer doesn’t need to know about these (and can’t directly change or use them)

AVR Registers

- General purpose registers are quite regular
  - Exception: a few instructions work on only the upper half (registers 16-31)
    - Bit limitations in some instructions (e.g. only 4 bits to specify which register)
  - There are many I/O registers
    - Not to be confused with general purpose registers
    - Some instructions work with these, others with general purpose registers – don’t confuse them
  - When X is needed as an index register, R26 and R27 are not available as general registers.
  - In AVR devices without SRAM, the registers are also the only memory – can be tricky to manage

What makes an ISA? #3: Data Types

- Numeric
  - Integers of different lengths (8, 16, 32, 64 bits)
    - Possibly signed or unsigned
  - Floating point numbers, e.g. 32 bits (single precision) or 64 bits (double precision)
  - Some machines support BCD (binary coded decimal) numbers
- Non-numeric
  - Boolean (0 means false, 1 means true) – stored in a whole byte or word
  - Bit-map (collection of booleans, e.g. 8 in a byte)
  - Characters
  - Pointers (memory addresses)

Data types (cont.)

- Different machines support different data types in hardware
  - e.g. Pentium II:
    - Signed integer
    - Unsigned integer
    - BCD integer
    - Floating point

  - e.g. Atmel AVR:
    - Signed integer
    - Unsigned integer
    - BCD integer
    - Floating point
Data types (cont.)

- Other data types can be supported in software
  - e.g. 16-bit integer operations can be built out of 8-bit operations
  - Floating point operations can be built out of logical and integer arithmetic operations

What makes an ISA?

#4: Instructions

- This is the main feature of an ISA
- Instructions include
  - Load/Store – move data from/to memory
  - Move – move data between registers
  - Arithmetic – addition, subtraction
  - Logical – Boolean operations
  - Branching – for deciding which instruction to perform next

Some AVR Instruction Examples

- Addition: \textit{add} \( r2, r1 \)
- Subtraction: \textit{sub} \( r13, r12 \)
- Branching: \textit{breq} 6
- Load: \textit{ldi} \( r30, $F0 \)
- Store: \textit{st} \( r2, x \)
- Port Read: \textit{in} \( r25, $16 \); Read port B
- Port Write: \textit{out} \$16, \( r17 \); Write to port B

ISA vs. Assembly Language

- ISA defines machine code (or machine language)
  - 1’s and 0’s that make up instructions
- Assembly language is a textual representation of machine language
  - Example (Atmel AVR instruction):
    \[
    \begin{align*}
    &\text{1001010100000111} &\text{(machine code)} \\
    &\text{inc r16} &\text{(assembly language, increment register 16)}
    \end{align*}
    \]
- Assembly language also includes macros
  - Example:
    \[
    \begin{align*}
    &.\text{def temp} = \text{r16} \\
    &.\text{include “8515def.inc”}
    \end{align*}
    \]
Summary: What makes an ISA?

- Memory models
- Registers
- Data types
- Instructions
- If you know all these details, you can
  - Write machine code that runs on the CPU
  - Build the CPU

Backwards Compatibility

- Many modern ISAs are constrained by backwards compatibility
  - Pentium ISA is backwards compatible to the 8088 (1978)
    - Echoes back to the 8080 (1974)
  - Problem: Pentium family is a poor target for compilers (register poor, irregular instruction set)
- AMD has defined a 64-bit extension to the Pentium architecture
  - Implemented by the Hammer family of CPUs

CISC vs. RISC

- How complex should the instruction set be? Should you do everything in hardware?
- 2 “styles” of ISA design
- **CISC = Complex Instruction Set Computer**
  - Lots of complex instructions – many of which take many clock cycles to execute
  - Examples: 8086 to 80386
  - Classic example: VAX had a single instruction to evaluate a polynomial equation
- **RISC = Reduced Instruction Set Computer**
  - Fewer, simpler instructions which can execute quickly (often one clock cycle)
  - Lots of registers
  - More complex operations built out of simpler instructions
  - Examples: SPARC, MIPS, PowerPC

CISC vs. RISC (cont.)

- Originally (80s)
  - CISC – 200+ instructions
  - RISC – ~50 instructions
- Today
  - Number of instructions irrelevant
  - Many “CISC” processors use RISC techniques
    - e.g. 80486 … Pentium IV
  - Better to look at use of registers/memory
    - CISC – often few registers, many instructions can access memory
    - RISC – many registers, only load/store instructions access memory
- Atmel AVR is a RISC processor
ISA vs. Microarchitecture

• An Instruction Set Architecture (ISA) can be implemented by many different microarchitectures

• Examples
  – 8086 ISA is implemented by many processors – in different ways
  – Pentium ISA is implemented by
    • Pentium … Pentium IV (in different ways)
    • Various AMD devices …
    • Other manufacturers also…

Reading Material

1. Chap 2 in *Microcontrollers and Microcomputers.*