Microprocessor

ET2223 Microprocessors, Microcontrollers, and Embedded Systems

Partially based on
Microprocessors and Embedded Systems, Hui Wu (2005), University of New South Wales
Introduction to VLSI Design, Steven P. Levitan (1998), University of Pittsburgh

Central Processing Unit

- The Central Processing Unit (CPU) is the “brain” of the computer.
- The CPU can be thought in terms of two basic pieces:
  - The Arithmetic Logic Unit (ALU) which modifies data by executing arithmetic and/or logical operations on it.
  - The Control Unit which takes the instructions from memory, decodes it, and then moves the data to the appropriate places and ensures that the ALU performs the desired operation.

Processor

- The CPU is often just called the processor.
- For the ENIAC and other computers of the first generation, the processor was comprised of vacuum tubes.
- The processor's individual vacuum tubes were replaced by individual transistors in the second generation of computers.
- Several transistors could be placed close together on an integrated circuit (IC) or chip. In third generation computers the processor consisted of several IC's.

Microprocessor

- Eventually the entire processor was placed on a single chip.
- When this became standard computers were said to enter the fourth generation.
- In this case, the processor is known as a microprocessor.
- What distinguishes a microprocessor from other integrated circuits is that a microprocessor can be programmed.
- So along with the idea of the microprocessor comes the idea of the instruction set – the set of actions the programmer can have the microprocessor perform.

Microprocessor Generations

- First generation: 1971-78
  - Behind the power curve
    (16-bit, <50k transistors)
- Second Generation: 1979-85
  - Becoming “real” computers
    (32-bit , >50k transistors)
- Third Generation: 1985-89
  - Challenging the “establishment”
    (Reduced Instruction Set Computer/RISC, >100k transistors)
- Fourth Generation: 1990-
  - Architectural and performance leadership
    (64-bit, > 1M transistors, Intel/AMD translate into RISC internally)

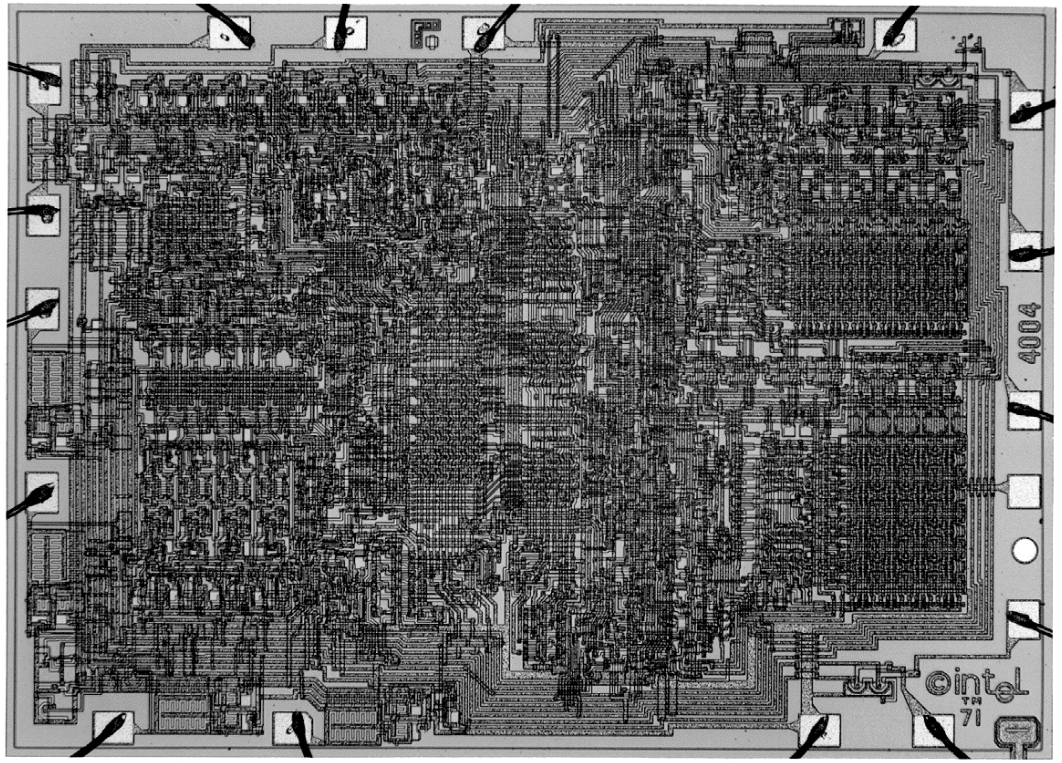
VLSI: Very Large Scale Integration

VLSI: Very Large Scale Integration

- Very large-scale integration (VLSI) is the process of integrating or embedding hundreds of thousands of transistors on a single silicon semiconductor microchip.
- VLSI technology was conceived in the late 1970s when advanced level computer processor microchips were under development.
- VLSI is a successor to large-scale integration (LSI), medium-scale integration (MSI) and small-scale integration (SSI) technologies.
- Before the introduction of VLSI technology most ICs had a limited set of functions they could perform.
- The microprocessor is a VLSI device.

VLSI: Very Large Scale Integration

- Integration: Integrated Circuits
  - multiple devices on one substrate
- How large is Very Large?
  - SSI (small scale integration)
    - 7400 series, 10-100 transistors
  - MSI (medium scale integration)
    - 74000 series 100-1000
  - LSI (large scale integration)
    - 1,000-10,000 transistors
  - VLSI (very large scale integration)
    - > 10,000 transistors



Intel 4004 Micro-Processor

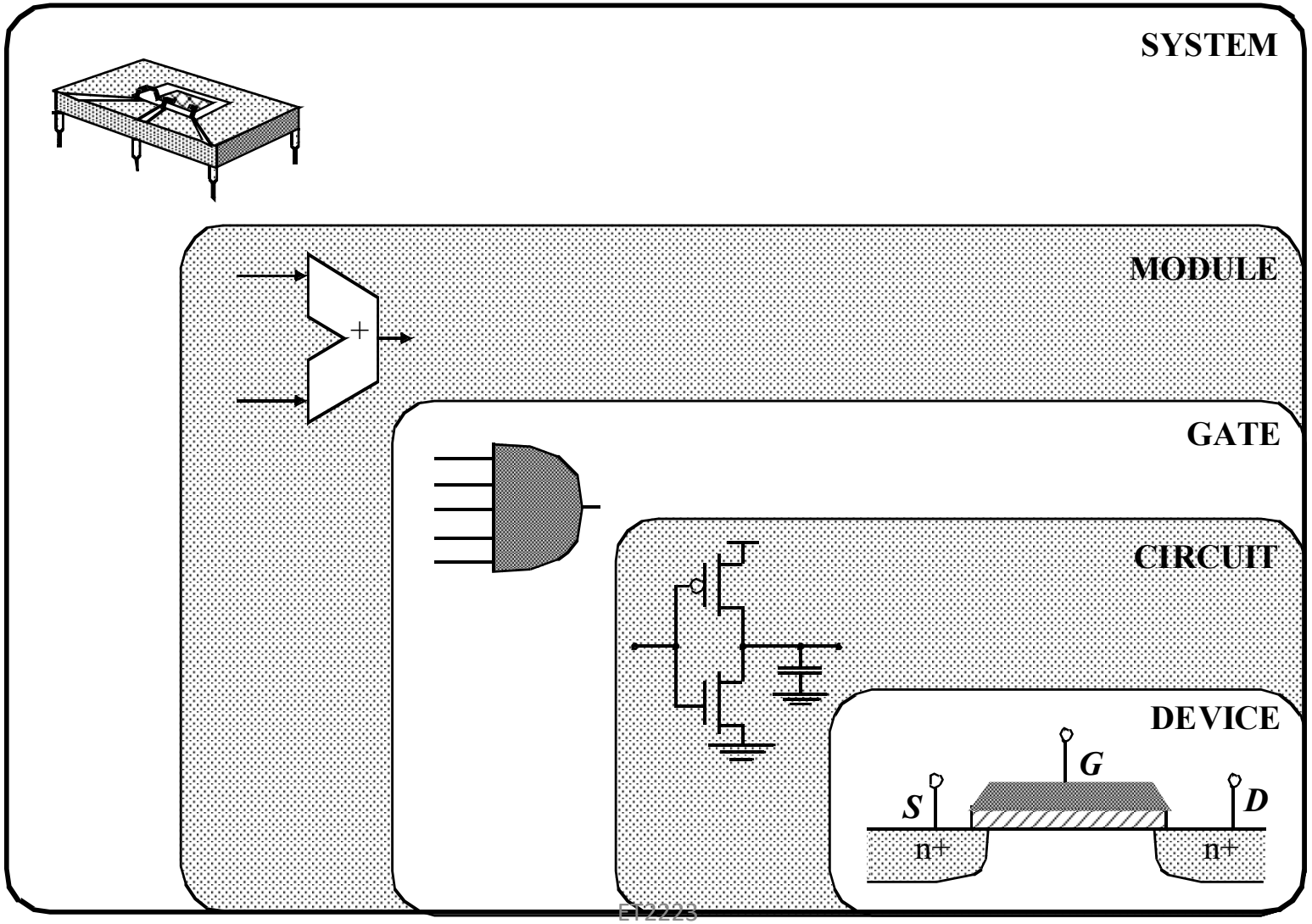
Integrated circuit classification

Name	Signification	Year	Number of Transistors	Number of Logic Gates
SSI	small-scale integration	1964	1 to 10	1 to 12
MSI	medium-scale integration	1968	10 to 500	13 to 99
LSI	large-scale integration	1971	500 to 20,000	100 to 9,999
VLSI	very large-scale integration	1980	20,000 to 1,000,000	10,000 to 99,999
ULSI	ultra-large-scale integration	1984	1,000,000 and more	100,000 and more

VLSI Design

- The real issue is that VLSI is about designing systems on chips.
- The designs are complex, and we need to use structured design techniques and sophisticated design tools to manage the complexity of the design.
- Consists of many different representations/abstractions of the system (chip) that is being designed.
- Each abstraction/view is itself a Design Hierarchy of refinements which decompose the design.

VLSI Design Abstraction Levels



VLSI Advantages

- Reduces the size of circuits
- Occupies a relatively smaller area
- Increases the operating speed of circuits
- Requires less power than discrete components
- Reduces the effective cost of the devices
- Higher reliability

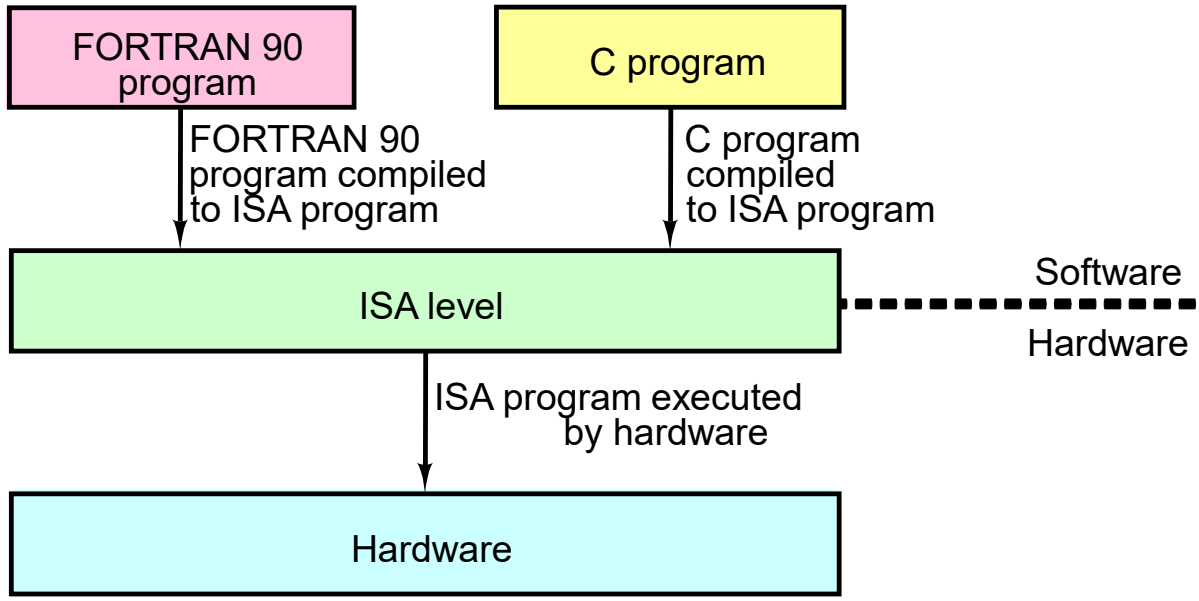
VLSI Applications

- Computers
- Voice and Data Communication networks
- Digital Signal Processing
- Commercial Electronics
- Automobiles
- Medicine
- … and many more

Instruction Set Architecture (ISA)

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
2. Registers
3. Data types
4. Instructions

ISA: Memory Models

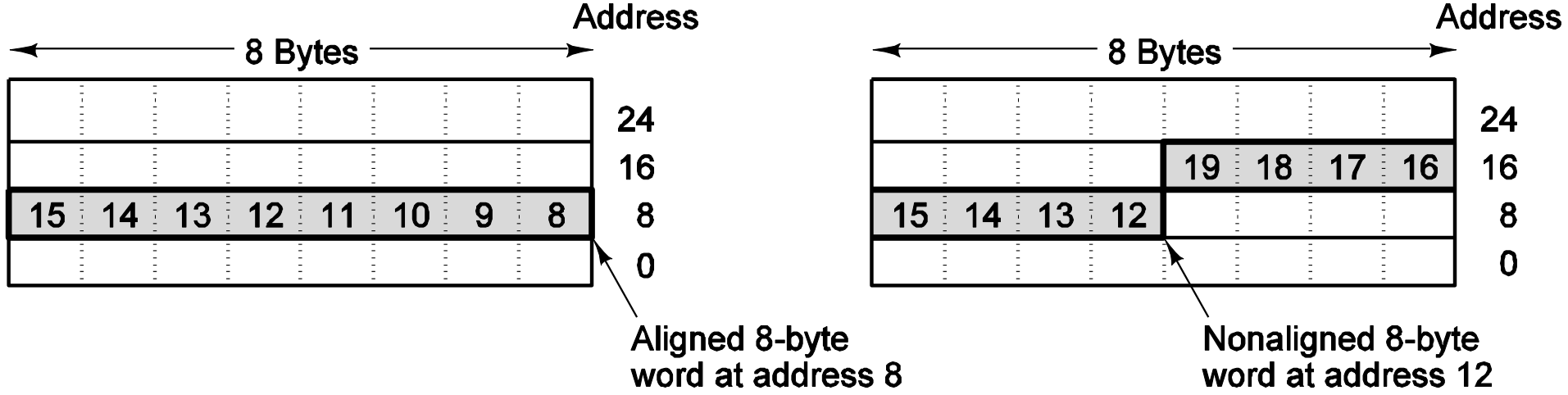
- Memory model:
  - how does memory look to CPU?
- Issues:
  - Addressable cell size
  - Alignment
  - Address spaces
  - Endianness

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

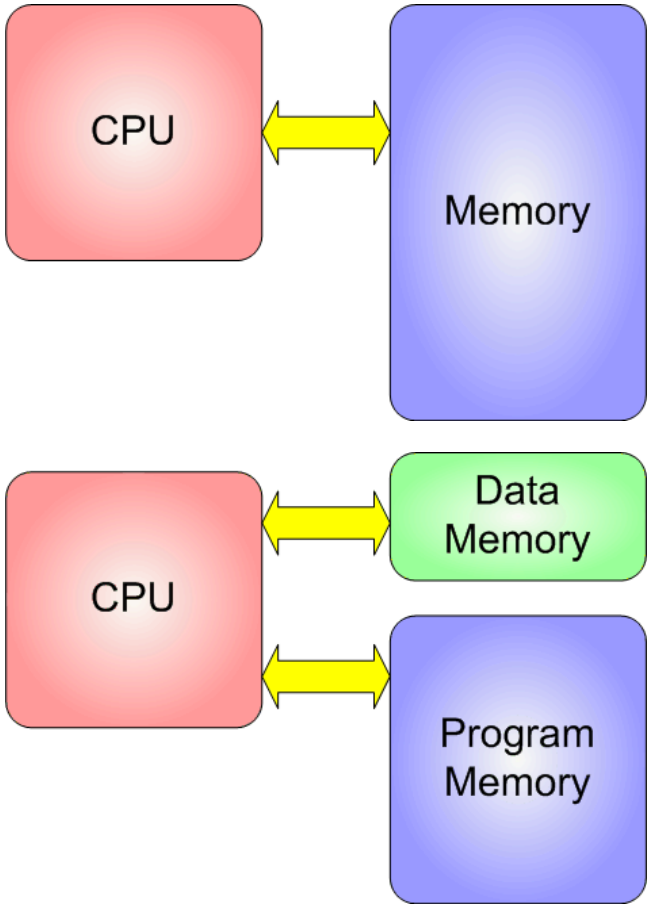
Alignment

- Many architectures require natural alignment
- Alignment often required because it is more efficient
- E.g.
  - 4-byte words starting at addresses 0,4,8, …
  - 8-byte words starting at addresses 0, 8, 16, …



Address Spaces

- Princeton architecture or Von Neumann architecture (most used).
  - A single linear address space for both instructions and data
  - e.g. 232 bytes numbered from 0 to 232 -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 216 bytes
    - Instruction address space: 212 16-bit words

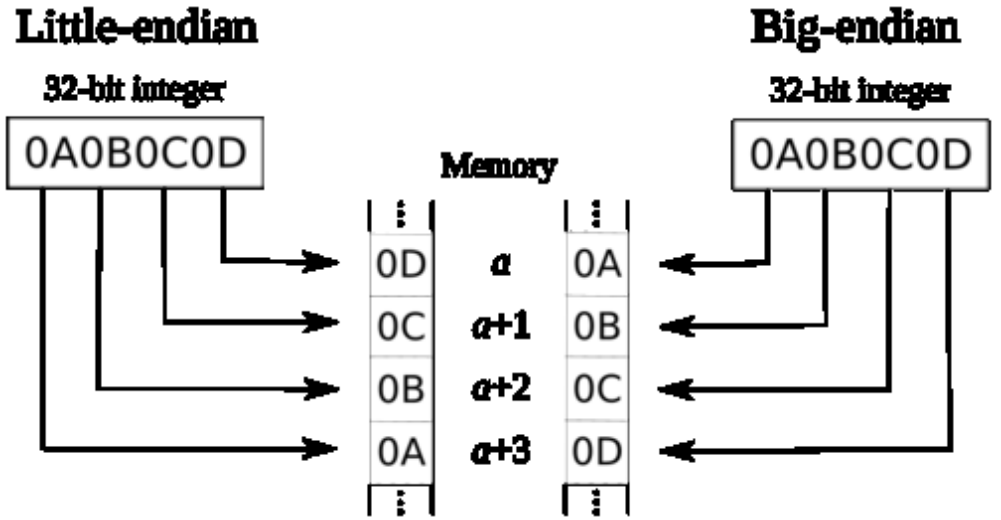


Princeton vs Harvard Architectures

Princeton Architecture	**Harvard Architecture**
Single memory to be shared by both code and data.	Separate memories for code and data.
Processor needs to fetch code in a separate clock cycle and data in another clock cycle. So it requires two clock cycles.	Single clock cycle is sufficient, as separate buses are used to access code and data.
Higher speed, thus less time consuming.	Slower in speed, thus more time-consuming.
Simple in design.	Complex in design.

Endianness

- Which bytes are most significant in multi-byte data?
- Little endian
  - little end (least significant byte) stored first (at lowest address)
  - Intel microprocessors (Pentium, etc.)
- Big endian
  - big end (most significant byte) stored first (at lowest address)
  - SPARC, Motorola microprocessors



ISA: Registers

- General purpose
  - Used for temporary results, etc.
- Special purpose
  - Program Counter (PC)
  - Stack pointer (SP)
  - Input/Output Registers
  - Status Register
- Some other registers are part of the microarchitecture NOT the ISA
  - i.e. programmer doesn't need to know about these (and can't directly change or use them)
  - Instruction Register (IR)
  - Memory Address Register (MAR)
  - Memory Data Register (MDR)

ISA: 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

- Different machines support different data types in hardware
- 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
- e.g. Pentium II:

Data Type	8 bits	16 bits	32 bits	64 bits	128 bits
Signed integer	✓	✓	✓		
Unsigned integer	✓	✓	✓		
BCD integer	✓				
Floating point			✓	✓	

- e.g. Atmel AVR:

Data Type	8 bits	16 bits	32 bits	64 bits	128 bits
Signed integer	✓				
Unsigned integer	✓				
BCD integer					
Floating point					

ISA: 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

Instructions

- Some AVR Instruction Examples
  - Addition: add r2, r1
  - Subtraction: sub r13, r12
  - Branching: breq 6
  - Load: ldi r30, $F0
  - Store: st r2, x
  - Port Read: in r25, $16; Read port B
  - Port Write: out $16, r17; Write to port B

ISA: 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

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):
    - 1001010100000011 (machine code)
    - inc r16 (assembly language, increment register 16)
- Assembly language also includes macros
  - Example:
    - .def temp = r16
    - .include “8515def.inc”

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

- 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

CISC vs. RISC

CISC	RISC
Larger set of instructions. Easy to program	Smaller set of Instructions. Difficult to program.
Simpler design of compiler, considering larger set of instructions.	Complex design of compiler.
Many addressing modes causing complex instruction formats.	Few addressing modes, fix instruction format.
Instruction length is variable.	Instruction length varies.
Higher clock cycles per second.	Low clock cycle per second.
Emphasis is on hardware.	Emphasis is on software.
Control unit implements large instruction set using micro-program unit.	Each instruction is to be executed by hardware.
Slower execution, as instructions are to be read from memory and decoded by the decoder unit.	Faster execution, as each instruction is to be executed by hardware.
Pipelining is not possible.	Pipelining of instructions is possible, considering single clock cycle.

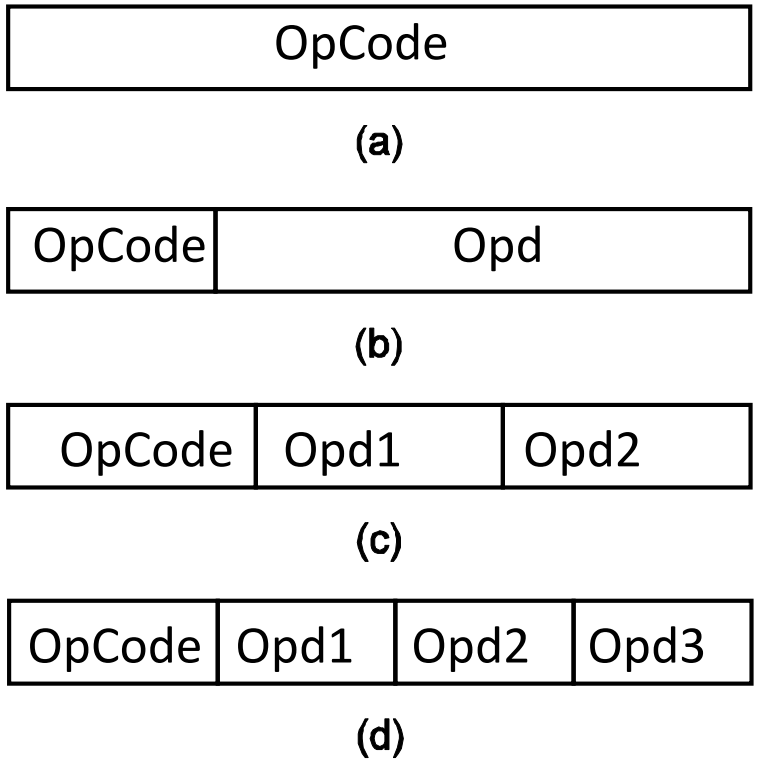
ISA vs. Microarchitecture

- 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…

Instruction Formats

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



Instruction Lengths

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

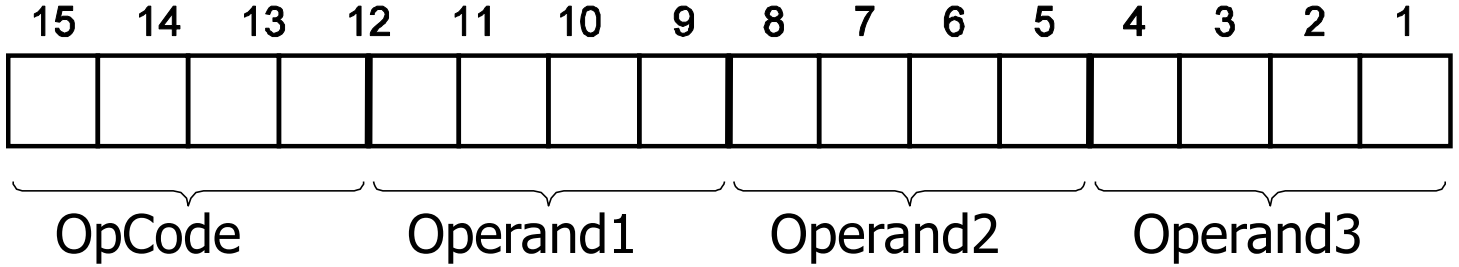
- AVR Instruction Examples
  - Almost all instructions are 16 bits long.
    - add Rd, Rr
    - sub Rd, Rr
    - mul Rd, Rr
    - brge k
  - Few instructions are 32 bits long.
    - lds Rd, k ($0 \leq k \leq 65535$)
    - loads 1 byte from the SRAM to a register.

Design Criteria for Instruction Formats

- Backwards Compatibility
  - e.g. Pentium 4 supports various instruction lengths so as to be compatible with 8086
- 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)
- Room to express operations
  - 2n operations needs at least n bits
  - Wise to allow room to add additional opcodes for next generation of CPU
- 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:



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

Summary

- Microprocessor Generations
  - CPU – Processor, Microprocessor
- Very Large Scale Integration (VLSI)
- Instruction Set Architecture (ISA)
- Instruction Formats