Chapter 1:
The General Purpose Machine

Topics

Looking Ahead—Chapter 2

Explores the nature of machines and machine languages

- Relationship of machines and languages
- Generic 32-bit Simple RISC Computer—SRC
- Register transfer notation—RTN
  - The main function of the CPU is the Register Transfer
  - RTN provides a formal specification of machine structure and function
  - Maps directly to hardware
- RTN and SRC will be used for examples in subsequent chapters
- Provides a general discussion of addressing modes
- Presents a view of logic design aimed at implementing registers and register transfers

Looking Ahead—Chapter 3

- Treats 2 real machines of different types—CISC and RISC—in some depth
  - Discusses general machine characteristics and performance
  - Differences in design philosophies of
    - CISC (Complex Instruction Set Computer) and
    - RISC (Reduced Instruction Set Computer) architectures
  - CISC machine—Motorola MC68000
    - Applies RTN to the description of real machines
  - RISC machine—SPARC

Looking Ahead—Chapter 4

This *keystone chapter* describes processor design at the logic gate level

- Describes the connection between the instruction set and the hardware
- Develops alternative 1-, 2-, and 3-bus designs of SRC at the gate level
- RTN provides description of structure and function at low and high levels
- Shows how to design the control unit that makes it all run
- Describes two additional machine features:
  - implementation of exceptions (interrupts)
  - machine reset capability

Looking Ahead—Chapter 5

Important advanced topics in CPU design

- General discussion of pipelining—having more than one instruction executing simultaneously
  - requirements on the instruction set
  - how instruction classes influence design
  - pipeline hazards: detection & management
- Design of a pipelined version of SRC
- Instruction-level parallelism—issuing more than one instruction simultaneously
  - Superscalar and VLIW designs
- Microcoding as a way to implement control

Looking Ahead—Chapter 6

The arithmetic and logic unit: ALU

- Impact on system performance
- Digital number systems and arithmetic in an arbitrary radix
  - number systems and radix conversion
  - integer add, subtract, multiply, and divide
- Time/space trade-offs: fast parallel arithmetic
- Floating point representations and operations
- Branching and the ALU
- Logic operations
- ALU hardware design

Looking Ahead—Chapter 7
The memory subsystem of the computer

- Structure of 1-bit RAM and ROM cells
- RAM chips, boards, and modules
- Concept of a memory hierarchy
  - nature of different levels
  - interaction of adjacent levels
- Virtual memory
- Cache design: matching cache & main memory
- Memory as a complete system

Looking Ahead—Chapter 8

Computer input and output: I/O

- Kinds of system buses, signals and timing
- Serial and parallel interfaces
- Interrupts and the I/O system
- Direct memory access—DMA
- DMA, interrupts, and the I/O system
- The hardware/software interface: device drivers

Looking Ahead—Chapter 9

Structure, function, and performance of peripheral devices

- Disk drives
  - Organization
  - Static and dynamic properties
- Video display terminals
- Memory-mapped video
- Printers
- Mouse and keyboard
- Interfacing to the analog world

Looking Ahead—Chapter 10

Computer communications, networking, and the Internet

- Communications protocols; layered networks
- The OSI layer model
- Point to point communication: RS-232 and ASCII
- Local area networks—LANs
  - Example: Ethernet
- Internetworking and the Internet
  - TCP/IP protocol stack
  - Packet routing and routers
  - IP addresses: assignment and use
  - Nets and subnets: subnet masks
- Internet applications and futures

Chapter 1—A Perspective

- Alan Turing showed that an abstract computer, a Turing machine, can compute any function that is computable by any means
- A general purpose computer with enough memory is equivalent to a Turing machine
- Over 50 years, computers have evolved
  - from memory size of 1 kiloword (1024 words) clock periods of 1 millisecond (0.001 s)
  - to memory size of a terabyte (2^{40} bytes) and clock periods of 1 ns (10^{-9} s)
- More speed and capacity is needed for many applications, such as real-time 3D animation

Scales, Units, and Conventions

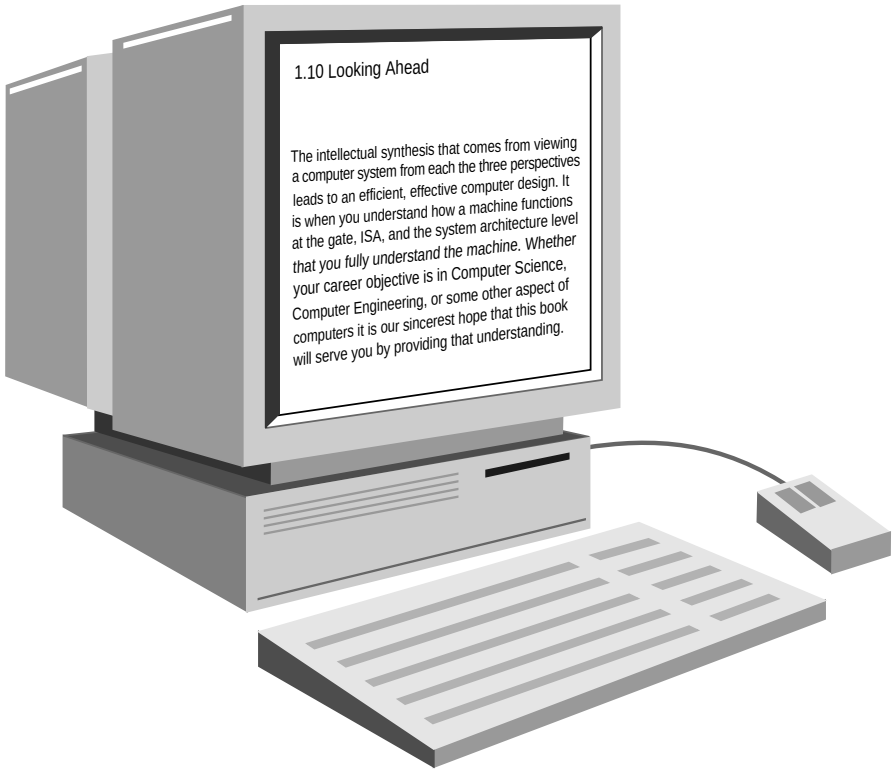
Term	Normal Usage	As a power of 2
K (kilo-)	10^3	2^{10} = 1,024
M (mega-)	10^6	2^{20} = 1,048,576
G (giga-)	10^9	2^{30} = 1,073,741,824
T (tera-)	10^{12}	2^{40} = 1,099,511,627,776

Term	Usage
m (milli-)	10^{-3}
μ (micro-)	10^{-6}
n (nano-)	10^{-9}
p (pico-)	10^{-12}

Note the differences between usages. You should commit the powers of 2 and 10 to memory.

Units: Bit (b), Byte (B), Nibble, Word (w), Double Word, Long Word, Second (s), Hertz (Hz)

Fig 1.1 The User's View of a Computer



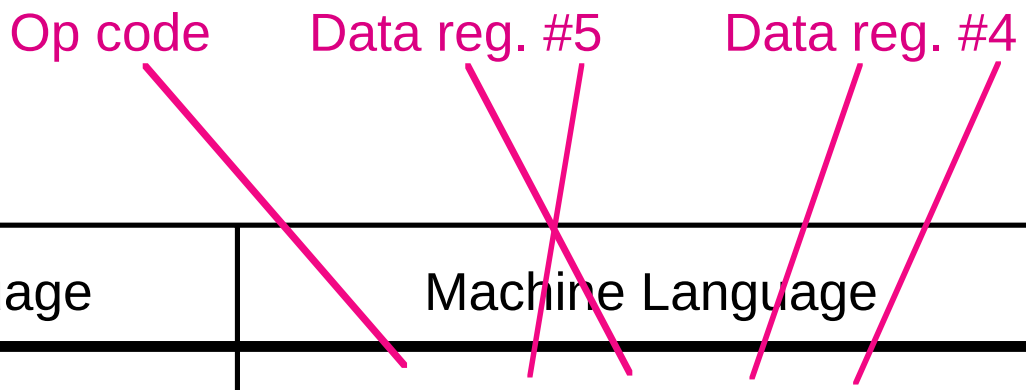
The user sees software, speed, storage capacity, and peripheral device functionality.

Machine/Assembly Language Programmer's View

- Machine language:
  - Set of fundamental instructions the machine can execute
  - Expressed as a pattern of 1's and 0's
- Assembly language:
  - Alphanumeric equivalent of machine language
  - Mnemonics more human-oriented than 1's and 0's
- Assembler:
  - Computer program that transliterates (one-to-one mapping) assembly to machine language
  - Computer's native language is machine/assembly language
  - "Programmer," as used in this course, means machine/ assembly language programmer

Machine and Assembly Language

- **The assembler converts assembly language to machine language. You must also know how to do this.**



MC68000 Assembly Language	Machine Language
MOVE.W D4, D5	0011 101 000 000 100
ADDI.W #9, D2	0000 000 010 111 100 0000 0000 0000 1001

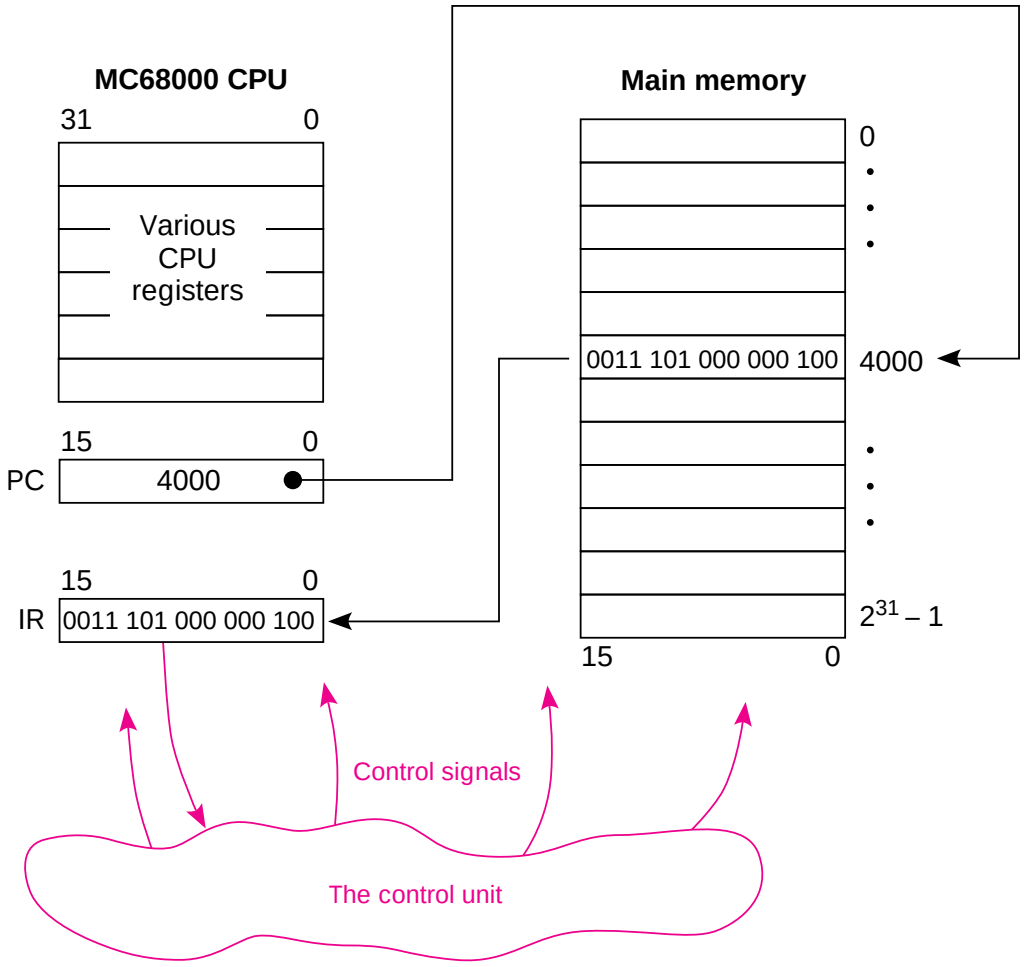
Tbl 1.2 Two Motorola MC68000 Instructions

The Stored Program Concept

The stored program concept says that the program is stored with data in the computer's memory. The computer is able to manipulate it as data—for example, to load it from disk, move it in memory, and store it back on disk.

- **It is the basic operating principle for every computer.**
- **It is so common that it is taken for granted.**
- **Without it, every instruction would have to be initiated manually.**

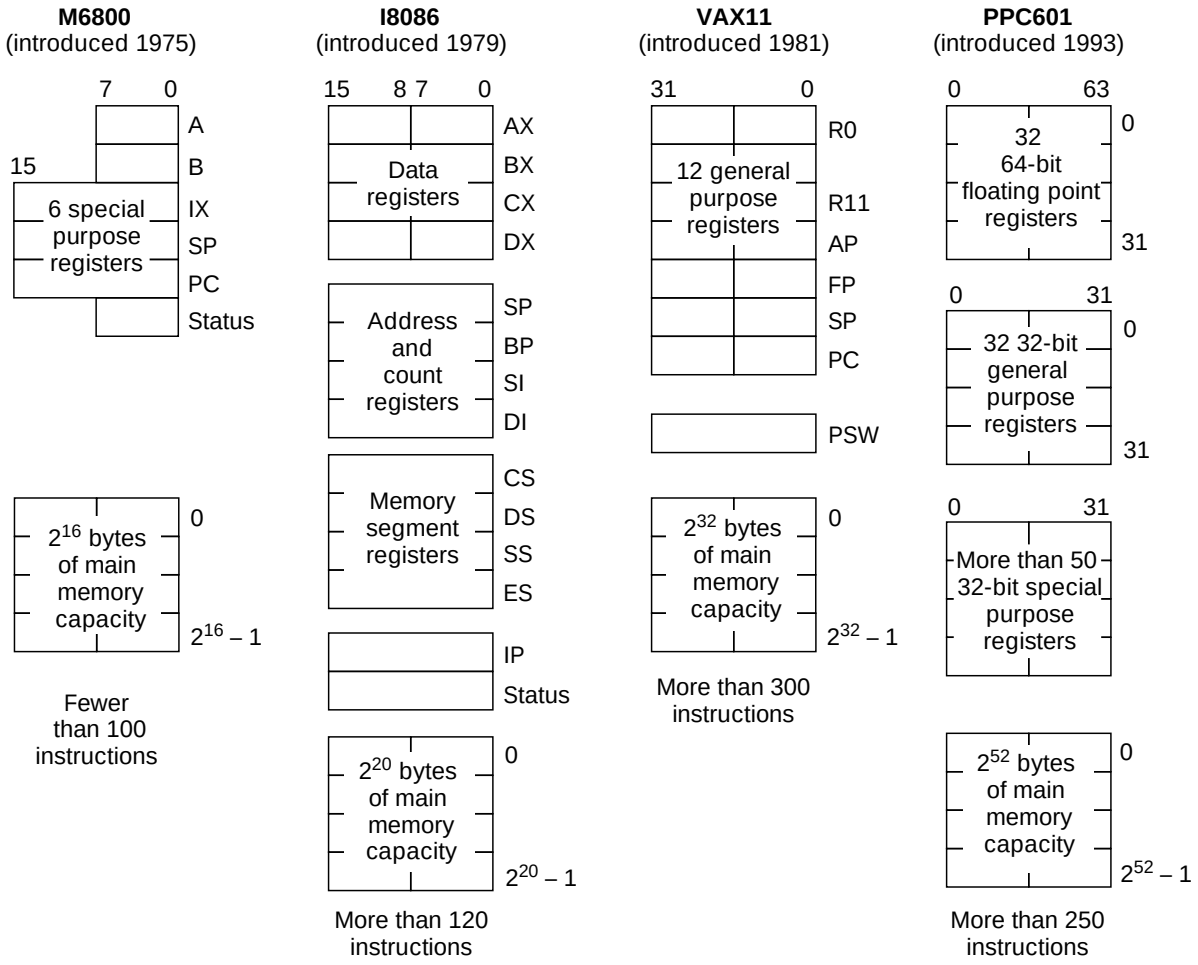
Fig 1.2 The Fetch-Execute Process



Programmer's Model: Instruction Set Architecture (ISA)

- Instruction set: the collection of all machine operations.
- Programmer sees set of instructions, along with the machine resources manipulated by them.
- ISA includes
  - Instruction set,
  - Memory, and
  - Programmer-accessible registers of the system.
- There may be temporary or scratch-pad memory used to implement some function is not part of ISA.
  - Not Programmer Accessible.

Fig 1.3 Programmer's Models of 4 Commercial Machines



Machine, Processor, and Memory State

- The Machine State: contents of all registers in system, accessible to programmer or not
- The Processor State: registers internal to the CPU
- The Memory State: contents of registers in the memory system
- “State” is used in the formal finite state machine sense
- Maintaining or restoring the machine and processor state is important to many operations, especially procedure calls and interrupts

Data Type: HLL Versus Machine Language

- HLLs provide type checking
  - Verifies proper use of variables at compile time
  - Allows compiler to determine memory requirements
  - Helps detect bad programming practices
- Most machines have no type checking
  - The machine sees only strings of bits
  - Instructions interpret the strings as a type: usually limited to signed or unsigned integers and FP numbers
  - A given 32-bit word might be an instruction, an integer, a FP number, or 4 ASCII characters

Tbl 1.3 Instruction Classes

Instruction Class	C	VAX Assembly Language
Data Movement	a = b	MOV b, a
Arithmetic/logic	b = c + d*e	MPY d, e, b ADD c, b, b
Control flow	goto LBL	BR LBL

- **This compiler:**
  - **Maps C integers to 32-bit VAX integers**
  - **Maps C assign, *, and + to VAX MOV, MPY, and ADD**
  - **Maps C goto to VAX BR instruction**
- **The compiler writer must develop this mapping for each language-machine pair**

Tools of the Assembly Language Programmer's Trade

- The assembler
- The linker
- The debugger or monitor
- The development system

Who Uses Assembly Language

- The machine designer
  - Must implement and trade off instruction functionality
- The compiler writer
  - Must generate machine language from a HLL
- The writer of time or space critical code
  - Performance goals may force program-specific optimizations of the assembly language
- Special purpose or imbedded processor programmers
  - Special functions and heavy dependence on unique I/O devices can make HLLs useless

The Computer Architect's View

- Architect is concerned with design & performance
- Designs the ISA for optimum programming utility and optimum performance of implementation
- Designs the hardware for best implementation of the instructions
- Uses performance measurement tools, such as benchmark programs, to see that goals are met
- Balances performance of building blocks such as CPU, memory, I/O devices, and interconnections
- Meets performance goals at lowest cost

Buses as Multiplexers

- Interconnections are very important to computer
- Most connections are shared
- A bus is a time-shared connection or multiplexer
- A bus provides a data path and control
- Buses may be serial, parallel, or a combination
  - Serial buses transmit one bit at a time
  - Parallel buses transmit many bits simultaneously on many wires

Fig 1.4 Simple One- and Two-Bus Architectures

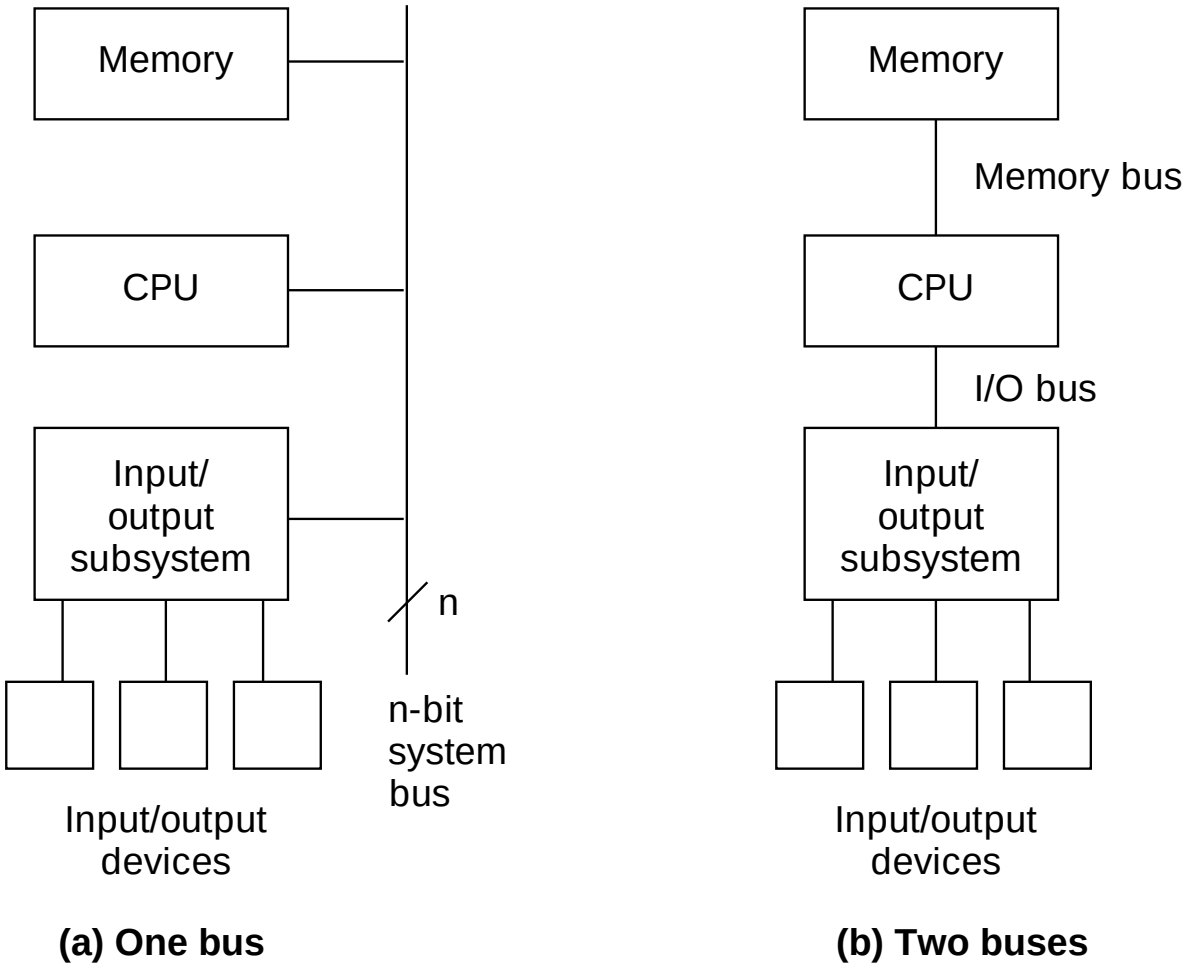


Fig 1.5 The Apple Quadra 950 Bus System (Simplified)

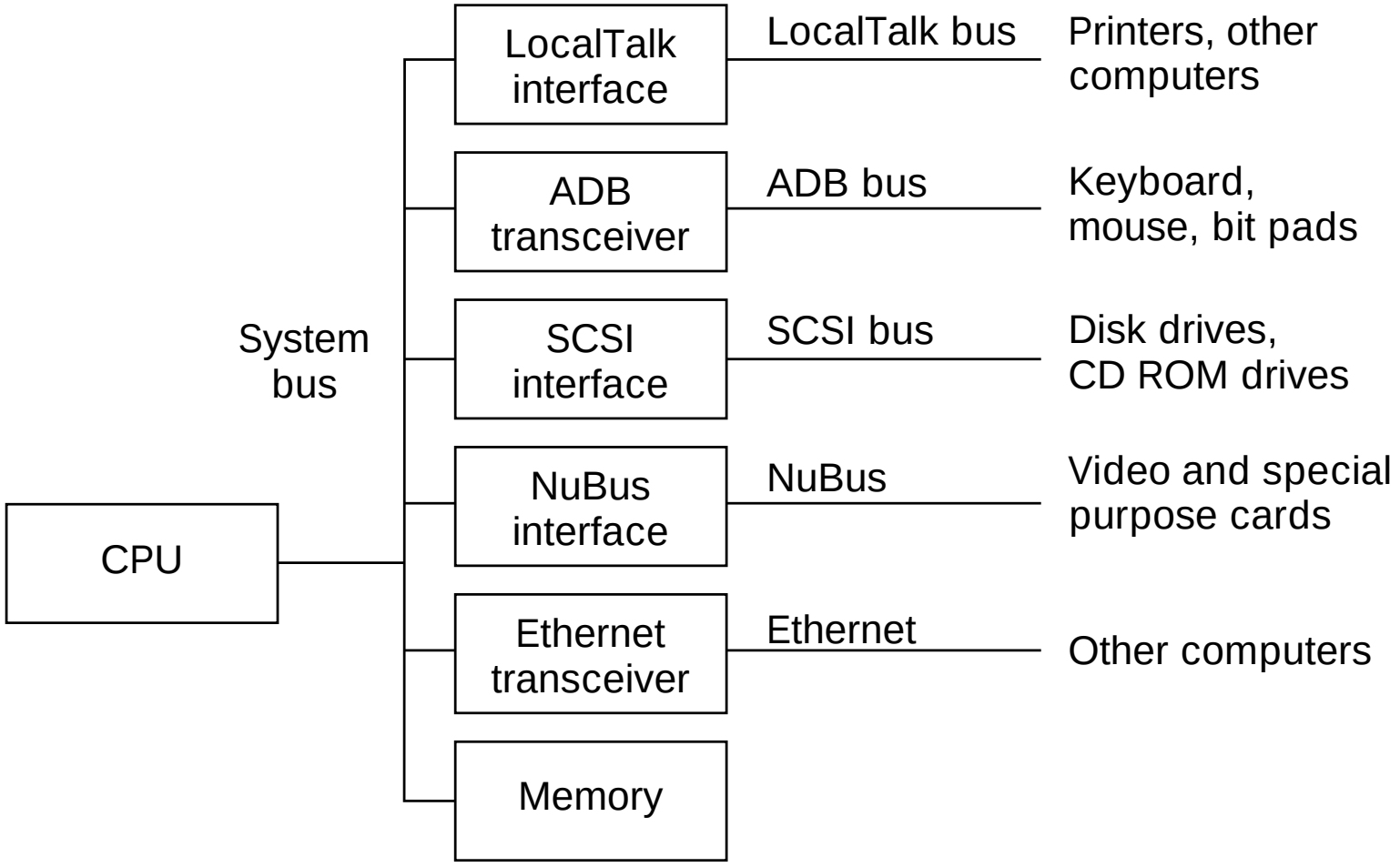
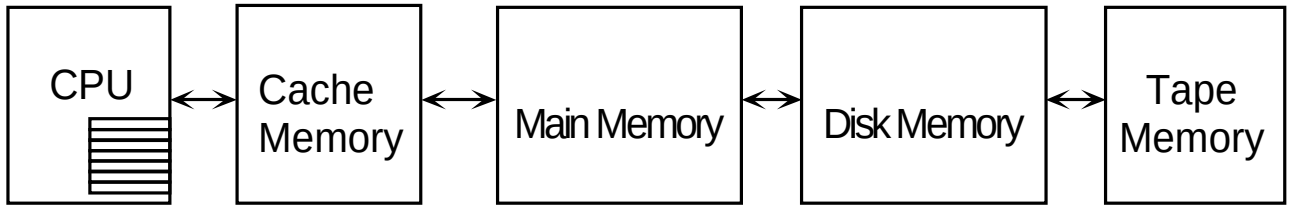


Fig 1.6 The Memory Hierarchy

- **Modern computers have a hierarchy of memories**
  - **Allows tradeoffs of speed/cost/volatility/size, etc.**
- **CPU sees common view of levels of the hierarchy.**



Tools of the Architect's Trade

- Software models, simulators and emulators
- Performance benchmark programs
- Specialized measurement programs
- Data flow and bottleneck analysis
- Subsystem balance analysis
- Parts, manufacturing, and testing cost analysis

Logic Designer's View

- Designs the machine at the logic gate level
- The design determines whether the architect meets cost and performance goals
- Architect and logic designer may be a single person or team

Implementation Domains

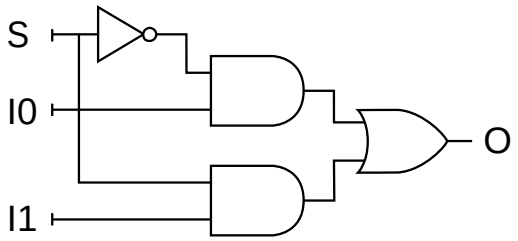
An implementation domain is the collection of devices, logic levels, etc. which the designer uses.

Possible implementation domains:

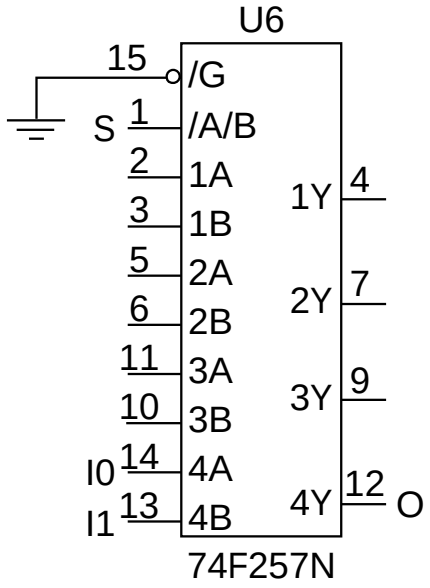
- **VLSI on silicon**
- **TTL or ECL chips**
- **Gallium arsenide chips**
- **PLAs or sea-of-gates arrays**
- **Fluidic logic or optical switches**

Fig 1.7 Three Implementation Domains for the 2-1 Multiplexer

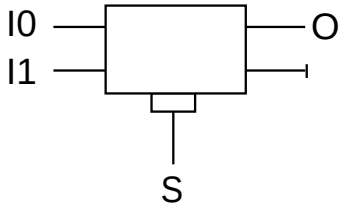
- **2-1 multiplexer in three different implementation domains**
  - **Generic logic gates (abstract domain)**
  - **National Semiconductor FAST Advanced Schottky TTL (VLSI on Si)**
  - **Fiber optic directional coupler switch (optical signals in $LiNbO_3$)**



(a) Abstract view of Boolean logic



(b) TTL implementation domain



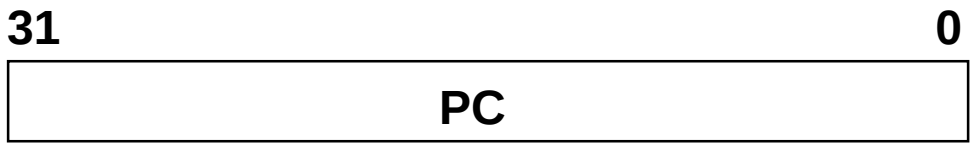
(c) Optical switch implementation

The Distinction Between Classical Logic Design and Computer Logic Design

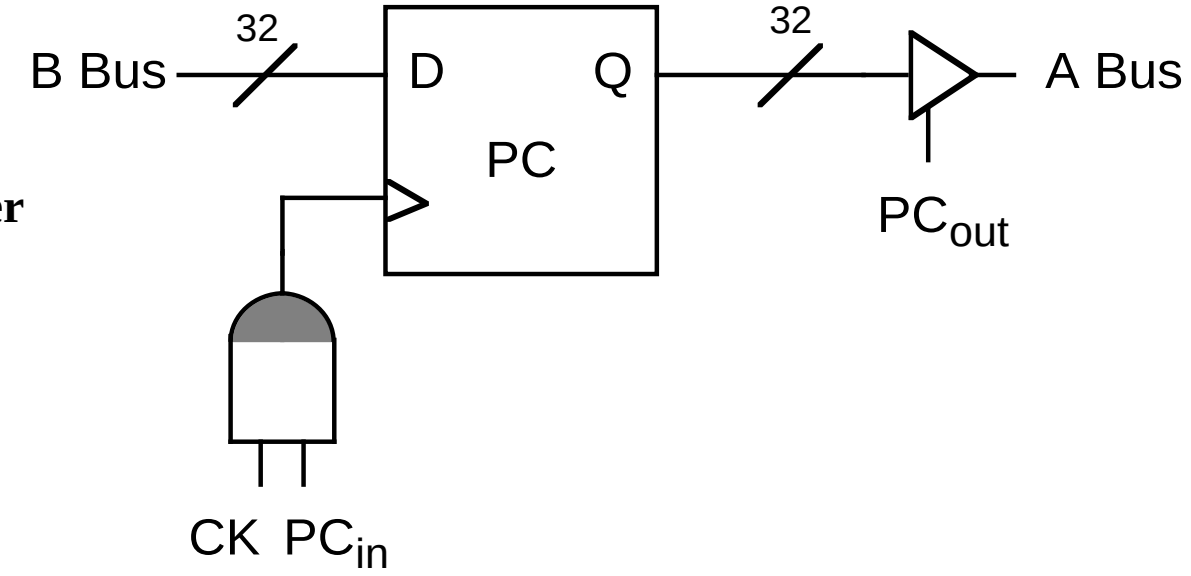
- The entire computer is too complex for traditional FSM design techniques
  - FSM techniques can be used “in the small”
- There is a natural separation between data and control
  - Data path: storage cells, arithmetic, and their connections
  - Control path: logic that manages data path information flow
- Well defined logic blocks are used repeatedly
  - Multiplexers, decoders, adders, etc.

Two Views of the CPU PC Register

Programmer:



Logic Designer (Fig 1.8):



Tools of the Logic Designer's Trade

- Computer-aided design tools
  - Logic design and simulation packages
  - Printed circuit layout tools
  - IC (integrated circuit) design and layout tools
- Logic analyzers and oscilloscopes
- Hardware development system

Historical Generations

- 1st Generation: 1946–59, vacuum tubes, relays, mercury delay lines
- 2nd generation: 1959–64, discrete transistors and magnetic cores
- 3rd generation: 1964–75, small- and medium-scale integrated circuits
- 4th generation: 1975–present, single-chip microcomputer
- Integration scale: components per chip
  - Small: 10–100
  - Medium: 100–1,000
  - Large: 1000–10,000
  - Very large: greater than 10,000

Chapter 1 Summary

- Three different views of machine structure and function
- Machine/assembly language view: registers, memory cells, instructions
  - PC, IR
  - Fetch-execute cycle
  - Programs can be manipulated as data
  - No, or almost no, data typing at machine level
- Architect views the entire system
  - Concerned with price/performance, system balance
- Logic designer sees system as collection of functional logic blocks
  - Must consider implementation domain
  - Tradeoffs: speed, power, gate fan-in, fan-out