
EE 4504
Computer Organization

Dr. N. J. Davis

Spring 1997

Course Objectives

- ◆ Review development of computer systems
- ◆ Examine the operation of the major building blocks of a computer system
- ◆ Investigate performance enhancements for each component

EE 4504
Computer Organization

Section 1
Introduction to Computer Systems

Objectives

- ◆ Review historical development of computer systems
- ◆ Identify design levels for computer system development
- ◆ Discuss descriptive and design tools for each design level
- ◆ Compare and contrast various performance metrics for computer systems

Computer Architecture

- ◆ Baer: “The design of the integrated system which provides a useful tool to the programmer”
- ◆ Hayes: “The study of the structure, behavior and design of computers”
- ◆ Abd-Alla: “The design of the system specification at a general or subsystem level”
- ◆ Foster: “The art of designing a machine that will be a pleasure to work with”
- ◆ Hennessy and Patterson: “The interface between the hardware and the lowest level software”

- ◆ Common themes:
 - Design / structure
 - Art
 - System
 - Tool for programmer and application
 - Interface
- ◆ Thus, computer architecture refers to those attributes of the system that are visible to a programmer -- those attributes that have a direct impact on the execution of a program
 - Instruction sets
 - Data representations
 - Addressing
 - I/O

Computer Organization

- ◆ Synonymous with “architecture” in many uses and textbooks
- ◆ We will use it to mean the underlying implementation of the architecture
- ◆ Transparent to the programmer
- ◆ An architecture can have a number of organizational implementations
 - Control signals
 - Technologies
 - Device implementations

What is a computer?

- ◆ Historically, a computer was a job title, not a piece of equipment!
- ◆ Requirements of a computer:
 - Process data
 - Store data
 - Move data between the computer and the outside world
 - Control the operation of the above

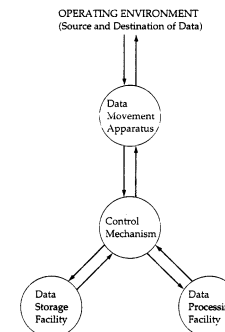


Figure 1.1 Functional view of a computer

History of Computers

- ◆ Mechanical Era (1600s-1940s)
 - Wilhelm Schickhard (1623)
 - » Astronomer and mathematician
 - » Automatically add, subtract, multiply, and divide
 - Blaise Pascal (1642)
 - » Mathematician
 - » Mass produced first working machine (50 copies)
 - » Could only add and subtract
 - » Maintenance and labor problems
 - Gottfried Leibniz (1673)
 - » Mathematician and inventor
 - » Improved on Pascal's machine
 - » Add, subtract, multiply, and divide

- Charles Babbage (1822)
 - » Mathematician
 - » “Father of modern computer”
 - » Wanted more accuracy in calculations
 - » Difference engine
 - ◆ Government / science agreement
 - ◆ Automatic computation of math tables
 - » Analytic engine
 - ◆ Perform any math operation
 - ◆ Punch cards
 - ◆ Modern structure: I/O, storage, ALU
 - ◆ Add in 1 second, multiply in 1 minute
 - » Both engines plagued by mechanical problems
- George Boole (1847)
 - » Mathematical analysis of logic
 - » Investigation of laws of thought

-
- Herman Hollerith (1889)
 - » Modern day punched card machine
 - » Formed Tabulating Machine Company (became IBM)
 - » 1880 census took 5 years to tabulate
 - » Tabulation estimates
 - ◆ 1890: 7.5 years
 - ◆ 1900: 10+ years
 - » Hollerith's tabulating machine reduced the 7.5 year estimate to 2 months
 - Konrad Zuse (1938)
 - » Built first working mechanical computer, the Z1
 - » Binary machine
 - » German government decided not to pursue development -- W.W.II already started
 - Howard Aiken (1943)
 - » Designed the Harvard Mark I
 - » Implementation of Babbage's machine
 - » Built by IBM

◆ Mechanical era summary

- Mechanical computers were designed to reduce the time required for calculations and increase accuracy of the results
- Two drawbacks
 - » Speed of operation limited by the inertia of moving parts (gears and pulleys)
 - » Cumbersome, unreliable, and expensive

The Electronic Era

- ◆ Generation 1 (1945 - 1958)
 - ENIAC
 - » Developed for calculating artillery firing tables
 - » Designed by Mauchly and Eckert of the University of Pennsylvania
 - » Generally regarded as the first electronic computer
 - ◆ Colossus probably the first, but was classified until recently
 - » BIG!
 - ◆ 18,000 tubes
 - ◆ 70,000 resistors
 - ◆ 10,000 capacitors
 - ◆ 6,000 switches
 - ◆ 30 x 50 feet
 - ◆ 140 kW of power
 - » Decimal number system used
 - » Programmed by manually setting switches

- IAS (Institute for Advanced Studies)
 - » von Neumann and Goldstone
 - » Took idea of ENIAC and developed concept of storing a program in the memory
 - » This architecture came to be known as the “von Neumann” architecture and has been the basis for virtually every machine designed since then
 - » Features
 - ◆ Data and instructions (programs) are stored in a single read-write memory
 - ◆ Memory contents are addressable by location, regardless of the content itself
 - ◆ Sequential execution
- Lots of initial and long-term fighting over patents, rights, credits, firsts, etc.

◆ **Generation 2 (1958 - 1964)**

- Technology change
- Transistors
- High level languages
- Floating point arithmetic

◆ **Generation 3 (1964 - 1974)**

- Introduction of integrated circuits
- Semiconductor memory
- Microprogramming
- Multiprogramming

◆ **Generation 4 (1974 - present)**

- Large scale integration / VLSI
- Single board computers

◆ **Generation 5 (? - ?)**

- VLSI / ULSI
- Computer communications networks
- Artificial intelligence
- Massively parallel machines

Summary of Generations

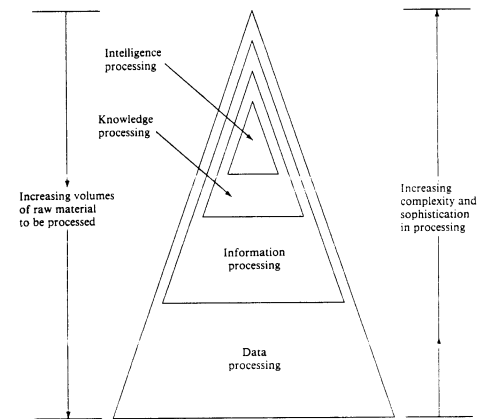
Generation	Example Machines	Hardware	Software	Performance
1	ENIAC, UNIVAC I, IBM 700	Vacuum tubes, magnetic drums	Machine code, stored programs	2 Kb memory, 10 KIPS
2	IBM 7094	Transistors, core memory	High level languages	32 Kb memory, 200 KIPS
3	IBM 360 370, PDP 11	ICs, semiconductor memory, microprocessors	Timesharing, graphics, structured programming	2 Mb memory, 5 MIPS
4	IBM 3090, Cray XMP, IBM PC	VLSI, networks, optical disks	Packaged programs, object-oriented languages, expert systems	8 Mb memory, 30 MIPS
5	Sun Sparc, Intel Paragon	ULSI, GaAs, parallel systems	Parallel languages symbolic processing, AI	64 Mb memory, 10 GFLOPS

Applications that Drive Computer Performance

- ◆ Weather forecasting
- ◆ Oceanography
- ◆ Seismic/petroleum exploration
- ◆ Medical research and diagnosis
- ◆ Aerodynamics and structure analysis
- ◆ nuclear physics
- ◆ Artificial intelligence
- ◆ Military/defense
- ◆ Socio-economics

Trends in Computer Usage

- ◆ 4 levels of ascending sophistication



Computer processing spaces [HwB84]

Four Levels of Computer Description

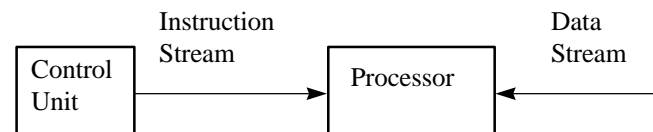
- ◆ Global system structure
 - Overall system structure is defined
 - Major components identified
 - » Processors
 - » Control modules
 - » Memory modules
 - » Interconnection structure
 - Mostly a static description -- “black box” approach
- ◆ Processor level
 - Architectural Features specified
 - » Interfaces
 - » Instruction sets
 - » Data Representation
 - More detailed individual component specification

- ◆ Register level
 - Specify internal operation of processor-level components at the word level
 - Primitives:
 - » Registers
 - » Counters
 - » Memories
 - » ALUs
 - » Clocks
 - » Combinational logic
- ◆ Gate level
 - Specify operations at the individual bit level
 - Gates are primitive elements
 - Very cumbersome to do manually (logic minimization, etc.)

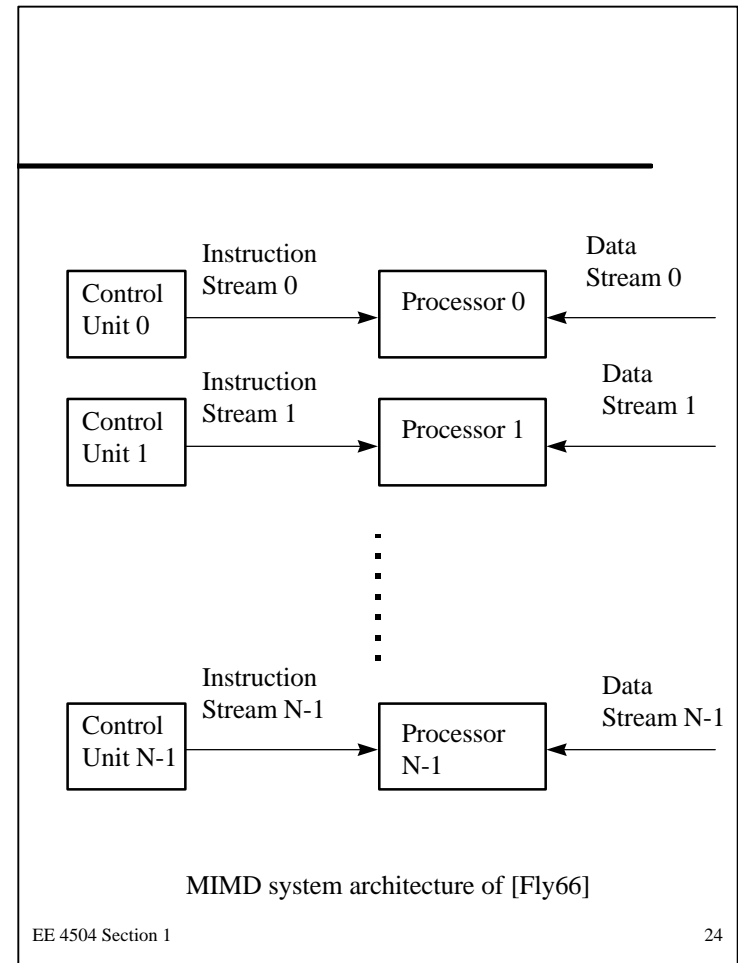
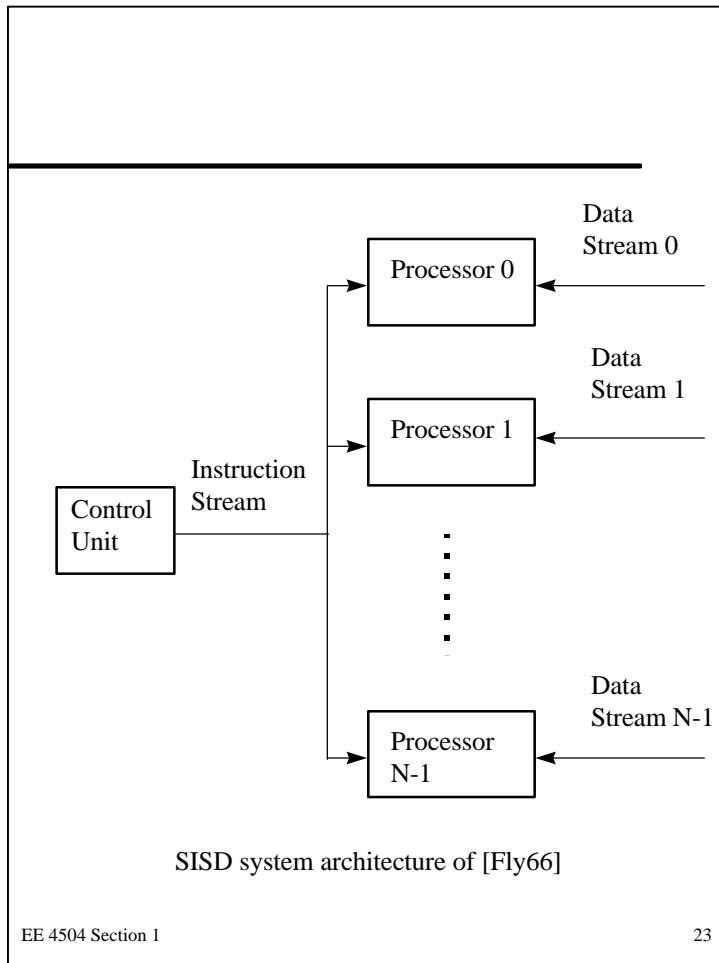
Global Descriptive Tools

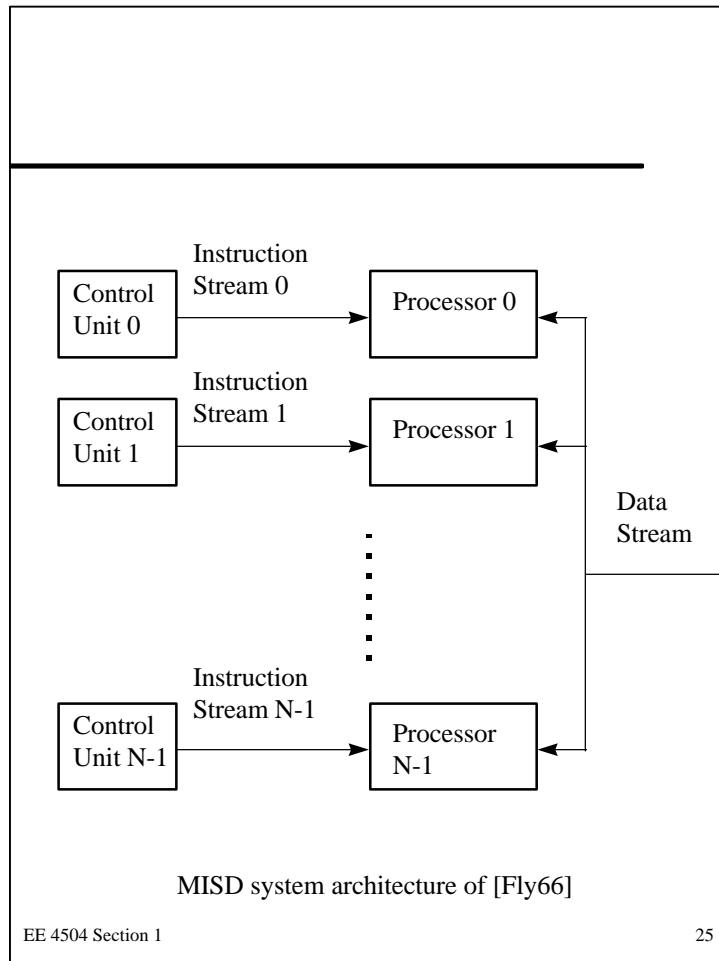
◆ Flynn's Taxonomy

- The most universally accepted method of classifying computer systems
- Relies on a block diagram approach
- Published in the Proceedings of the IEEE in 1966
- Any computer can be placed in one of 4 broad categories
 - » SISD: Single instruction stream, single data stream
 - » SIMD: Single instruction stream, multiple data streams
 - » MIMD: Multiple instruction streams, multiple data streams
 - » MISD: Multiple instruction streams, single data stream

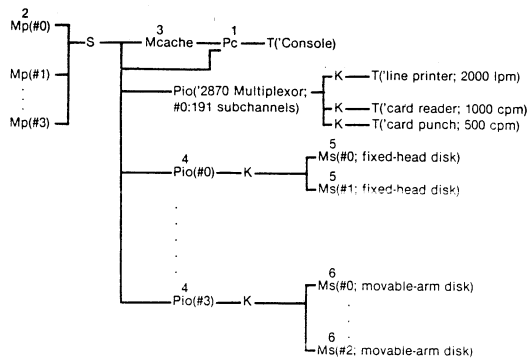


SISD system architecture of [Fly66]





- Advantages of Flynn
 - » Universally accepted
 - » Compact Notation
 - » Easy to classify a system (?)
 - Disadvantages of Flynn
 - » Very coarse-grain differentiation among machine systems
 - » Comparison of different systems is limited
 - » Interconnections, I/O, memory not considered in the scheme
 - ◆ Other global level tools
 - Tendency to rely on block diagrams and very coarse performance measures
 - » Processor-memory-switch notation of [BeN71] uses block diagrams with 7 basic component types
- EE 4504 Section 1 26



1. Pc (Model 155; cycle-time : 115ns., microprogrammed 72 bit hoz.; 8b/byte; data-path: 4 bytes; (2, 4|6) bytes/instr.; Mps - 88 bytes)
2. Mp (core; 256 KBytes; t.cycle: 2.1 μs; 16 bytes/transfer)
3. Mcache (monolithic; 8KBytes; t.cycle: 230ns.; 4 bytes/transfer)
4. Pio (2880 Block Multiplexor; transfer rate: 3MB/sec)
5. Ms (2305 fixed-head disk; capacity: 5MBytes; t.access: 5ms.; transfer rate: 1.5 MB/sec)
6. Ms (3330 movable-arm disk; (2|4|6|8) drives; capacity : 100MB/drive; t.seek : 10-55 ms.; t.access ~ 8.4ms.; transfer rate : 806 KB/sec)

PMS description of IBM 370/155 [Bae80]

Processor Level Descriptive Tools

- ◆ At this level, the operation of the global level components and their interfaces must be defined
- ◆ Items to be specified include:
 - Data formats
 - » Word lengths
 - » Instruction formats
 - » Data representation
 - Memory accessing
 - Instruction set and its operation
- ◆ Specification takes the appearance of a software program
 - Permits direct simulation of the machine's operation
- ◆ Typical tool is ISP -- Instruction Set Processor

Pc State

R[0:15](0:31)	General registers
R0 := R[0]	Register 0
XR[1:7] := R[1:7]	Index registers
PSD(0:83)	Program status doubleword
CC(1:4) := PSD(0:3)	Condition codes
FMC(0:2) := PSD(5:7)	Floating point mode control
FS := FMC(0)	Floating significance
FZ := FMC(1)	Floating zero
FN := FMC(2)	Floating normalize
NS := PSD(3)	Master-mode-slave-mode
DM := PSD(10)	Decimal mask
AM := PSD(11)	Arithmetic mask
IA := PSD(15:31)	Instruction address
WK(0:1) := PSD(34:35)	Write-key
IIS(0:2) := PSD(37:39)	Interrupt inhibits
CI := IIS(0)	
IJ := IIS(1)	
EI := IIS(2)	
RSP := PSD(56:59)	Register block pointer

MWP(0:255)(0:1)
 IR(0:33) := Instruction □00

Memory write-protect registers
 Instruction register

Mp State

Mp(0:3)[0:1FFF₁₆](0:31)

4 Modules of Mp

Instruction formats

i(0:31)	Instruction
ib := i(0)	Indirect bit
op(0:6) := i(1:7)	Operation code
gr(0:3) := i(8:11)	General register field
ir(0:2) := i(12:14)	Index register field
address(0:16) := i(15:31)	Operand address
value(10:19) := i(12:31)	Immediate value
hop(0:3) := op(0:3)	Half opcode

Effective address calculation

EA := (ba - Z(15:33); ha - Z(15:32);	Effective address length
wa - Z(15:31); dwa - Z(15:30))	
Z(15:33) := (!ib - Z;	Indirect addressing
ib - (address(0:16) - Mp[address](15:31);	
next Z')	
Z'(15:33) := (X0 - address □00;	Indexing
!X0 - address □00 + (ba - R[ir];	with register
ha - R[ir]□0; wa - R[ir]□00;	shifting
dwa - R[ir]□000)	

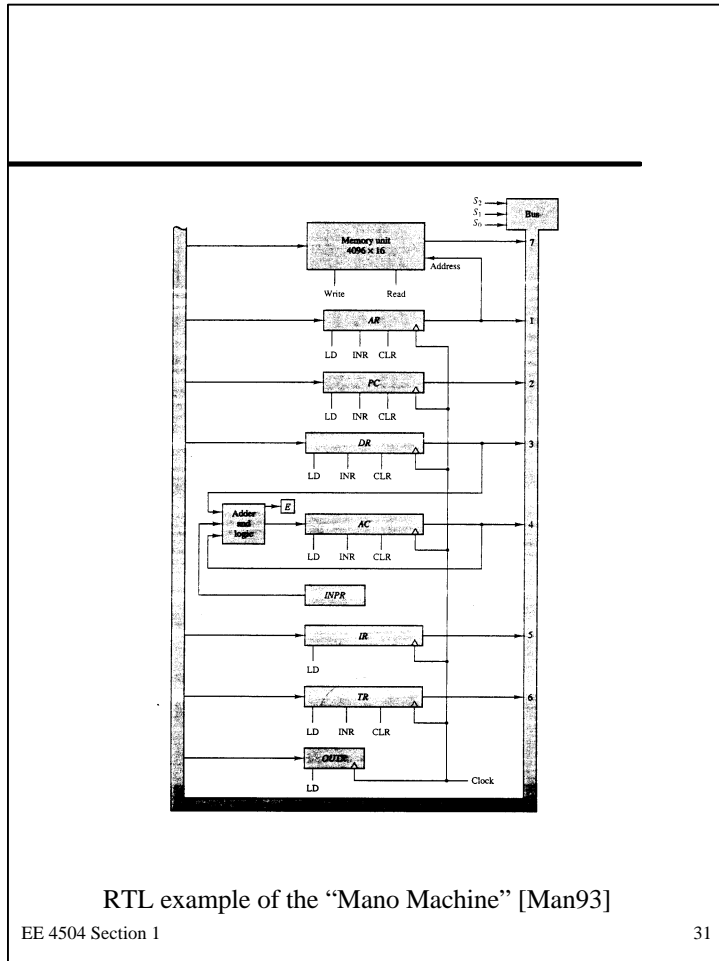
ISP specification of Sigma 5 machine [Bae80]

EE 4504 Section 1 29

Register Level Descriptive Tools

- ◆ Internal operation of each processor level component is defined
- ◆ Defines the actual hardware of the system in terms of data flow (at the word (register) level) and the associated control mechanisms
- ◆ Any number of “hardware design languages” (CDLs, HDLs, and RTLs) can be used
 - An example is the RTL from the Mano machine in EE 2504

EE 4504 Section 1 30



Fetch	$R^*T_0: AR \leftarrow PC$
	$R^*T_1: IR \leftarrow M[AR], PC \leftarrow PC + 1$
Decode	$R^*T_2: D_0, \dots, D_7 \leftarrow \text{Decode } IR(12-14),$ $AR \leftarrow IR(0-11), I \leftarrow IR(15)$
Indirect	$D^*IT_3: AR \leftarrow M[AR]$
Interrupt:	
	$I^*T_4, T_5: (IEN) \wedge (FGI \wedge FGO): R \leftarrow 1$
	$R^*T_6: AR \leftarrow 0, TR \leftarrow PC$
	$R^*T_7: M[AR] \leftarrow TR, PC \leftarrow 0$
	$R^*T_8: PC \leftarrow PC + 1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$
Memory-reference:	
AND	$D^*T_0: DR \leftarrow M[AR]$ $D^*T_1: AC \leftarrow AC \wedge DR, SC \leftarrow 0$
ADD	$D^*T_0: DR \leftarrow M[AR]$ $D^*T_1: AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$
LDA	$D^*T_0: DR \leftarrow M[AR]$ $D^*T_1: AC \leftarrow DR, SC \leftarrow 0$
STA	$D^*T_0: M[AR] \leftarrow AC, SC \leftarrow 0$
BUN	$D^*T_0: PC \leftarrow AR, SC \leftarrow 0$
BSA	$D^*T_0: M[AR] \leftarrow PC, AR \leftarrow AR + 1$ $D^*T_1: PC \leftarrow AR, SC \leftarrow 0$
ISZ	$D^*T_0: DR \leftarrow M[AR]$ $D^*T_1: DR \leftarrow DR + 1$ $D^*T_2: M[AR] \leftarrow DR, \text{ if } (DR = 0) \text{ then } (PC \leftarrow PC + 1), SC \leftarrow 0$
Register-reference:	
	$D^*T_0: r = r$ (common to all register-reference instructions)
	$IR(i) = B_i (i = 0, 1, 2, \dots, 11)$
	$r: SC \leftarrow 0$
CLA	$rB_0: AC \leftarrow 0$
CLE	$rB_0: E \leftarrow 0$
CMA	$rB_0: AC \leftarrow \overline{AC}$
CME	$rB_0: E \leftarrow \overline{E}$
CIR	$rB_0: AC \leftarrow \text{shr } AC, AC(15) \leftarrow E, E \leftarrow AC(0)$
CIL	$rB_0: AC \leftarrow \text{shl } AC, AC(0) \leftarrow E, E \leftarrow AC(15)$
INC	$rB_0: AC \leftarrow AC + 1$
SFA	$rB_0: \text{if } (AC(15) = 0) \text{ then } (PC \leftarrow PC + 1)$
SNA	$rB_0: \text{if } (AC(15) = 1) \text{ then } (PC \leftarrow PC + 1)$
SZA	$rB_0: \text{if } (AC = 0) \text{ then } (PC \leftarrow PC + 1)$
SZE	$rB_0: \text{if } (E = 0) \text{ then } (PC \leftarrow PC + 1)$
HLT	$rB_0: S \leftarrow 0$
Input-output:	
	$D^*IT_0: p = p$ (common to all input-output instructions)
	$IR(i) = B_i (i = 6, 7, 8, 9, 10, 11)$
	$p: SC \leftarrow 0$
INP	$pB_0: AC(0-7) \leftarrow INPR, FGI \leftarrow 0$
OUT	$pB_0: OUTR \leftarrow AC(0-7), FGO \leftarrow 0$
SKI	$pB_0: \text{if } (FGI = 1) \text{ then } (PC \leftarrow PC + 1)$
SKO	$pB_0: \text{if } (FGO = 1) \text{ then } (PC \leftarrow PC + 1)$
ION	$pB_0: IEN \leftarrow 1$
IOF	$pB_0: IEN \leftarrow 0$

RTL example of the “Mano Machine” [Man93] (cont)

EE 4504 Section 1 32

Gate Level Descriptive Tools

- ◆ At the gate level, descriptive tools rely on combinational and sequential design techniques as in EE 2504 and EE 4505/6
- ◆ To specify a complete computer system at this level is a staggering task!
- ◆ Current automated design tools have replaced the manual methods of state tables, truth tables, etc.

VHDL -- A Universal Design Tool

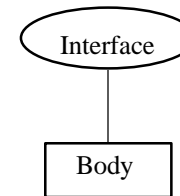
- ◆ Background
 - DoD sponsored the VHSIC Hardware Design Language (VHDL) program in the 1980s to promote the rapid insertion of advanced microelectronic components into operational systems -- cut design and development time
 - Speed process by:
 - » Increasing communication among defense contractors
 - » Increasing efficiency of CAD/CAM capabilities
 - » Improving functioning of multi-contractor teams
- ◆ Objectives
 - Support hardware technologies and design methodologies
 - Support design styles and automation tools
 - Support management of design data

◆ Implementation

- Main focus is on chip-level designs at the gate and transistor level
- Designs supported hierarchical design decomposition
- VHDL design specifications resemble “programs” -- very similar to Ada in style
- Structured programming mechanisms enable top-down hierarchical decomposition of the design process
- As a result of the programming nature of the specification, simulation of a design is possible
- VHDL has been used to span all design levels and is now being used to specify full multiprocessor systems from the global level all the way down to actual implementation in hardware

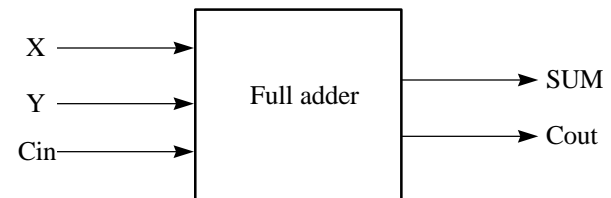
◆ Modeling computer hardware

- A hardware component is represented using a “design entity”
- Entities consist of 2 parts
 - » Interface -- containing externally visible information
 - » Bodies -- Describing one or more implementations of the entity



Design entity

-
- ◆ Entity interface contains / defines externally visible items
 - Ports
 - Data parameters
 - Generic parameters
 - Declarations and assertions
 - ◆ Entity body describes alternative implementations of the entity
 - Architectural
 - » Data flow approach where body statements are executed in parallel
 - » Function decomposition composed of other entities
 - Behavioral
 - » Control flow approach -- statements executed sequentially
 - » No information on structural decomposition



architecture architecture_view of full_adder is view:

```
block
begin
    SUM <= X xor Y xor Cin
    Cout <= (X and Y) or (Y and Cin) or
            (Cin and X)
end block;
end architecture_view
```

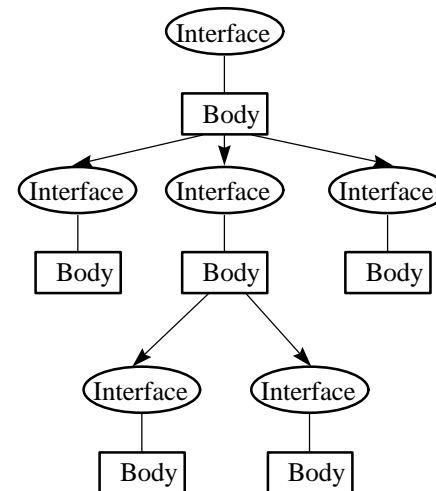
architecture behavior_view of full_adder is view:

```
block
begin

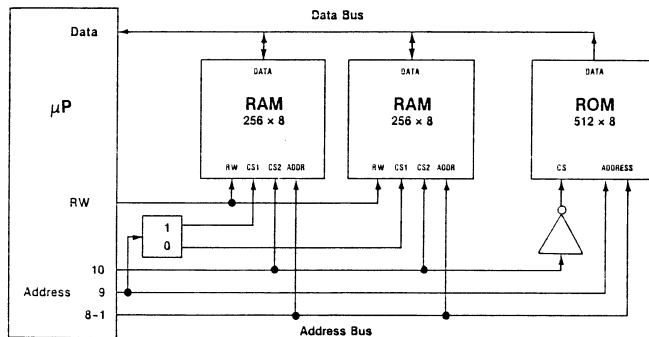
    process (X, Y, Cin)
        variable S: bit_vector (1 to 3):= X&Y&Cin;
        variable NUM: integer range 0 to 3 :=0;
    begin
        for i:= 1 to 3 loop
            if S(i) = '1' then
                NUM := NUM+1;
            end if;
        end loop;
        case NUM is
            when 0 => Cout<=0; SUM<=0;
            when 1 => Cout <=0; SUM<=1;
            when 2 => Cout<=1; SUM<=0;
            when 3=> Cout<=1; SUM<=1;
        end case;
    end process;
end block;
end behavior_view
```

◆ Architectural description can contain references to other entities

- Designs can use libraries of entities of previously designed “parts” to speed the process



- ◆ Thus, the full adder could be defined as consisting of interconnected XOR, AND, and OR gates which are themselves defined as entities in a library
- ◆ Consider a simple computer system:



```

block
  — data type declarations
  type mvl is ('0', '1', 'Z');
  type tristate is array (7 downto 0) of
    atomic one_at_a_time mvl;
  — local component declarations
  component MICROPROCESSOR
    port (Address : out BIT_VECTOR;
          RW      : out BIT;
          Data    : inout tristate);
  component RAM
    port (CS1, CS2 : in BIT;
          RW       : in BIT;
          Addr    : in BIT_VECTOR;
          Data    : inout tristate);
  component ROM
    port (CS : in BIT;
          Address : in BIT_VECTOR;
          Data : out tristate);
  component DECODER
    port (input : in BIT;
          output : out BIT_VECTOR);
  component INVERTER
    port (input : in BIT;
          output : out BIT);
  — local signal declarations
  signal Address_Bus : BIT_VECTOR (10 downto 1);
  signal RW : BIT;
  signal Ram_CS : BIT_VECTOR (0 to 1);
  signal Rom_CS : BIT;
  signal Data_bus : tristate;

begin
  — component instantiation statements
  CPU : MICROPROCESSOR port (Address_Bus, RW, Data_Bus);
  SEL : DECODER port (Address_Bus(9), Ram_CS);
  M0 : RAM port (Ram_CS(0), Address_Bus(10), RW,
                Address_Bus(8 downto 1), Data_Bus);
  M1 : RAM port (Ram_CS(1), Address_Bus(10), RW,
                Address_Bus(8 downto 1), Data_Bus);
  INV : INVERTER port (Address_Bus(10), Rom_CS);
  M2 : ROM port (Rom_CS, Address_Bus(9 downto 1), Data_Bus);

end block;

```

Computer Architecture Performance Measures

◆ Introduction

- In the last section, representative descriptive tools for each design level were presented
- We still have the problem of assessing one or more differing architectures in a dynamic sense -- how well (fast) will the machine work?
- In this section, we will discuss common ways of gauging a system's value in terms of its cost and its performance
- Observation: An increase in a machine's performance is viewed in one of two (competing) ways:
 - » Reduced response time to an individual job
 - » Increase in overall throughput

- Which of the following increases throughput, reduces response time, or both?
 - » Faster clock cycle time
 - » Multiple processors for separate tasks
 - » Parallel processing of scientific problems
- Recalling that architects design machines to run programs, improved performance is a total system process as embodied by Amdahl's Law:
 - » The performance improvement to be gained from using some faster mode of execution is limited by the fraction of time the faster mode can be used

- Consider the problem of leaving West Virginia for civilization in Virginia. Assume that you must walk out of the mountains and that portion of the trip takes 20 hours. Once in Virginia, 200 miles are traversed in one of several modes of travel.

Compute the total travel time for each mode and the speedup compared to walking the entire trip.

Modes:

- » Walk at 4 MPH
- » Ride a bike at 10 MPH
- » Drive a Honda Excel at 50 MPH
- » Drive a Ferrari at 120 MPH
- » Drive a rocket car at 600 MPH

Vehicle Used in VA	Hrs for Trip in VA	Speedup in VA	Hrs for Entire Trip	Speedup for Trip
Walk	50	1	70	1
Bike	20	2.5	40	1.8
Honda	4	12.5	24	2.9
Ferrari	1.67	30	21.67	3.2
Rocket Car	0.33	150	20.33	3.4

- Moral 1: One must be careful as an architect in assessing the impact of a particular performance enhancement on the system as a whole
- Moral 2: When evaluating a machine's performance, one must be very careful in the use of narrowly defined test/benchmark data

◆ Memory Bandwidth

- Memory bandwidth is the maximum rate in bits per second at which information can be transferred to or from main memory
- Imposes a basic limit on the processing power of a system
- Weakness is that it is not related in any way to actual program execution
- Not one of the current "in" figures of merit

◆ MIPS

- Defined as millions of instructions per second

$$MIPS = \frac{InstructionCount}{ExecTime \times 10^6} = \frac{ClockRate}{CPI \times 10^6}$$

- In general, faster machines will have higher MIPS ratings and appear to have better performance
- Advantage in use: easy to "understand," easy to market systems with this measure of performance
- Problems:
 - » Rating of a machine is based on its instruction set -- how do you compare machines with very different instruction sets? Apples and Oranges

- MIPS rating can vary on a single computer based on program being executed
- MIPS can vary inversely to performance! -- increase in performance with a decrease in MIPS rating
- Example 1:

Use of a floating point unit vs. S/W routines for floating point operations:

FPU uses less time and less instructions, S/W uses many simple integer instructions leading to what seems like higher MIPS rating -- even though it takes more time than the FPU to perform the task

- Example 2: Impact of optimizing compiler

Assume the following program makeup:

Operation	Freq.	Clock Cycles
ALU	43%	1
Load	21%	2
Store	12%	2
Branch	24%	2

Assume a 20 ns clock, optimizing compiler eliminates 50% of all ALU operations

NOT Optimized:

$$\text{Ave CPI} = 0.43 \times 1 + 0.21 \times 2 + 0.12 \times 2 + 0.24 \times 2 = 1.57$$

$$\text{MIPS} = 50 \text{ MHz} / 1.57 \times 10^6 = 31.8$$

Optimized:

$$\text{Ave CPI} = (.43/2 \times 1 + .21 \times 2 + .12 \times 2 + .24 \times 2) / (1 - .43/2) = 1.73$$

$$\text{MIPS} = 50 \text{ MHz} / 1.73 \times 10^6 = 28.9$$

-
- To attempt to standardize MIPS ratings across machines, we now have native vs normalized MIPS
 - » Use a reference machine to standardize the ratings
 - » VAX 11-780 1 MIP machine has been the standard for a number of years

$$\text{Norm. MIPS} = \frac{\text{TimeReference}}{\text{TimeNewMachine}} \times \text{MIPS.Ref}$$

◆ MFLOPS

- Millions of floating point operations per second
- Useful in comparing performance of "scientific applications" machines
- Intended to provide a "fair" comparison between such machines since a flop is the same on all machines
- Problems
 - » While a flop is a flop, not all machines implement the same set of flops -- some operations are synthesized from more primitive flops
 - » MFLOP rating varies with floating point instruction mix -- the idea of fast vs. slow floating point opns

◆ Programs for performance analysis

- To accurately gauge system performance, applications programs must be considered
- Synthetic Benchmarks
 - » Programs that attempt to match average frequency of operations and operands over a large program base
 - » Don't do any real work
 - » Whetstone -- based on 1970's Algol programs
 - » Dhrystone -- Based on a composite of HLL statements for 1980's -- targeted to test CPU and compiler performance
 - » Designer can get around these benchmarks

- Toy Benchmarks

- » 10-100 lines of easily produced code with known computational result
- » Small, easy to write (compiler may not be produced yet) and evaluate
- » Hennessy and Patterson assert best use is for "beginning programming assignments" since the benchmarks do not reflect computer's performance for normal sized applications

- Kernels

- » Small, key pieces of real programs
- » Livermore loops and Linpack are good examples
- » Tend to focus on a specific aspect of the overall performance rather than the entire system

- Real Programs

- » Actual applications with real I/O
- » Best measure of a machine's total capability
- » Often, the hardest to judge based on limited access to machine -- have not bought it yet!
- » Typical suite would include compilers, word processors, math applications, etc.
- » SPEC (System Performance Evaluation Cooperative) is a good example of workstation benchmark (started by HP, Sun, DEC, and MIPS)

- ◆ Other considerations

- COST!!!
 - » Design cost
 - » Purchase cost
 - » Components: component, direct, indirect
- Compatibility
- S/W availability
- Maintenance

Section Summary

- ◆ This section has briefly presented a history of the electronic computer
 - Generations
 - Technologies
 - Applications
- ◆ Surveyed
 - Classification and descriptive levels of computer design
 - Associated design tools
- ◆ Introduced performance measures for computer evaluation and comparison