Computer Architecture and Organization

Module-1

- Introduction and overview of computer architecture

  - Introduction to computer systems
  - Over view of organization and architecture
  - Functional components of a computer
  - Registers and register files
  - Interconnection of components
  - Organization of Von Neumann machine
  - Harvard architecture
  - Performance of processor

Why this subject?

- To acquire some understanding and appreciation of a computer systems functional components, their characteristics, performance and interactions
- Need to understand computer architecture in order to structure a program so that it runs more efficiently on a real machine
- In selecting a system to use, they should be able to understand the trade of among various components such as CPU clock speed vs memory

Module 1

Introduction and Overview of Computer Architecture

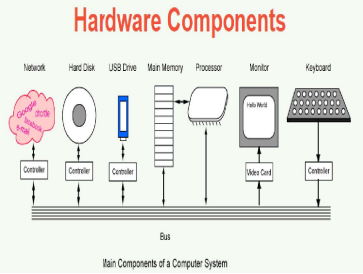
Introduction to Computer Systems

A computer is an electronic data processing machine that accepts data and instructions through input unit (or from memory) and processes data as per the instruction, to get desired output.

A computer system is a complex system consists of both hardware and software.

Hardware and Software

- The **hardware** components of a computer system are the electronic and mechanical parts
- The **software** components of a computer system are the data and the computer programs.



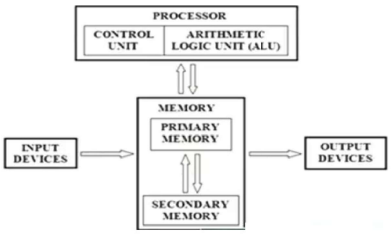
Main Components of a Computer System

- Computer is a digital device, which works on two levels of signal: High or Low
- High level signal (5v or 12v) and low level signal (0v)
- To make it convenient, 0v represented by L (Low) or 0
  1v represented by H(High) or 1
- All the functionalities of the computer system are represented with 0 and 1
- Binary number system is used to represent information and manipulation of information in computer

Overview of Computer Organization and Architecture

- Organization
  - Design of the components and functional blocks using which computers are built
  - Analogy: Civil Engineer task during building construction (cement, bricks, iron rods and other building materials during construction)
- Architecture
  - How to integrate the components to build a computer system to achieve a desired level of performance
  - Analogy: Architect task during the planning of the building (overall layout, floor plan, etc.)

Functional Components of a Computer



Input Unit

Memory

- Physical device to store programs or data
- Two types: Main memory (Physical memory) and Secondary memory
- Memory vs storage
- Main memory vs Secondary memory

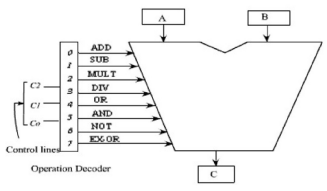
- **Main memory:**
  - closely connected to the processor.
  - stored data are quickly and easily changed.
  - holds the programs and data that the processor is actively working with.
  - interacts with the processor millions of times per second.
  - needs constant electric power to keep its information.
- **Secondary memory:**
  - connected o main memory through the bus and a controller.
  - stored data are easily changed, but changes are slow compared to main memory.
  - used for long-term storage of programs and data.
  - before data and programs can be used, they must be copied from secondary memory into main memory.
  - does not need electric power to keep its information.

Main memory vs Secondary memory

Central Processing Unit (CPU)

Arithmetic and Logic Unit (ALU)

- Most computer operations are performed at ALU
- Example



Block Diagram of the ALU

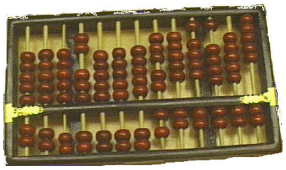
Control Unit

- Operations of all the units are coordinated by control unit
- Sends control signal to all the units
- Timing signals that govern the data transfer, are generated by control circuits
- Timing signals are signals that determine when a given action takes place

Output Unit

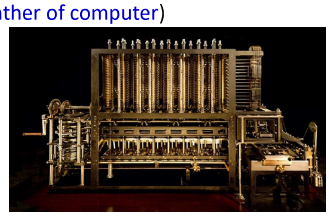
Evolution of computer system

- The beginning of computing – Abacus 3000 BC
- Calculating tool that was in use centuries before the adoption of the written modern numeral system
- Still widely used by traders, merchants and clerks in Asia and Africa



Babbage's Difference Engine (1823)

- He was a mathematician, philosopher and a mechanical engineer
- He is best remembered now for originating the concept of a programmable computer
- He is credited with inventing the first mechanical computer (So considered as father of computer)



ENIAC (1943-1946)

- ENIAC (Electronic Numeric Integrator And Computer)
- Designed by Mauchly and Eckert (University of Pennsylvaniya)
- First general purpose electronic computer
- Response to WW2 need to calculate trajectory tables for weapons
- ENIAC details in decimal (not binary)
- 20 accumulators for 10 digits
- Programmed manually by switches
- 18,000 vacuum tubes, 30 tons
- 15,000 sft, 140 KW power consumption
- 5000 additions per second

ENIAC (1943-1946)

- Vacuum tubes

ENIAC (1943-1946)

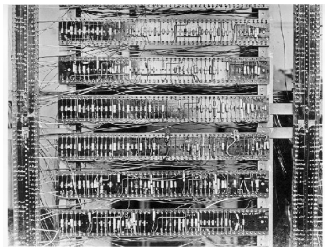
- **To reprogram the ENIAC you had to rearrange the patch cards that you can observe on the left in the photo, and the setting of 3000 switches that you can observe on the right**

Transistor Based Computers

- Transistors replaced vacuum tubes
- Smaller, cheaper and less heat dissipation
- Made from silicon (sand)
- Invented at Bell labs, 1947
- Commercial Transistor based computers: NCR & RCA produced small transistor machines, IBM 7000, DEC – 1957 (PDP -1)

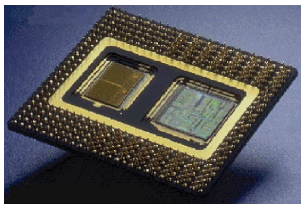
Transistor Based Computers

- First transistor based computer – Manchester University (1953)



Integrated Circuits

- A single self contained transistor is called a discrete component
- Transistor based computers – discrete components manufactured separately, packed in their own containers, and soldered or wired together onto circuit boards
- The transistor based computers contained about 10000 transistors – but grew up to hundreds of thousands
- Integrated circuits revolutionized electronics
- Silicon chip - collection of tiny transistors



Generation of computers

- Vacuum tubes – 1946 – 1957 (one bit – size of hand)
- Transistors - 1958 -1964 (one bit – size of a finger nail)
- Small Scale Integration (SSI) – 1965 onwards (up to 100 of devices on a single chip)
- Medium Scale Integration (MSI) –up to 1971 (100 to 3000 devices on chip
- Large Scale Integration (LSI) – 1971 -1977 (3000 to 100,000 devices on a chip
- Very Large Scale Integration (VLSI) – 1978 – Till date (100,000 to 100,000,000 devices on a chip
- Ultra Large Scale Integration (ULSI) (over 100,000,000 devices on a chip

Computer Generations

Generation	Period	Technology
First	1945 - 1954	Vacuum Tubes
Second	1955 - 1964	Transistors
Third	1965 -1974	Integrated Circuits (SSI, MSI)
Fourth	1975 - ?	Integrated Circuits (LSI, VLSI)
FIFTH	?	AI, Neural Network, Web Computing etc.

Registers

- To speed up the processor operations, the processor includes some internal memory storage locations called registers
- Memory hierarchy
  - Registers
  - Cache memory
  - Main memory
  - Secondary memory
- At higher level of hierarchy, memory is faster, smaller and more expensive
- Registers top level of hierarchy
- Two roles
  - User visible registers – referenced by machine instruction
  - Control and Status registers – employed to control the operation

User visible registers

- General purpose register
- Data registers
- Address registers
- Condition code registers

User visible registers

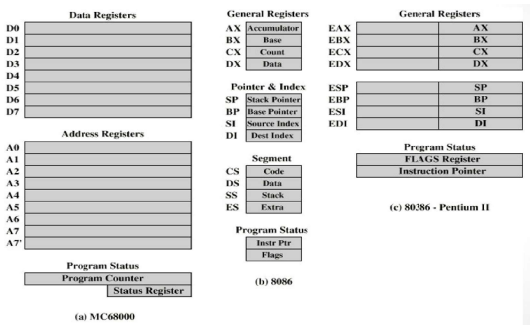
- General-purpose register ($R_0 – R_{n-1}$)
- Data register – used only to hold data and not for any other purpose
- Address Register
  - Segment register – holds the address of the base of the segment
  - Index register – Used for indexed addressing
  - Stack Pointers – points to the top of the stack
- Condition Codes (Flags) – are bits set by the processor as a result of an operation

Control and Status registers

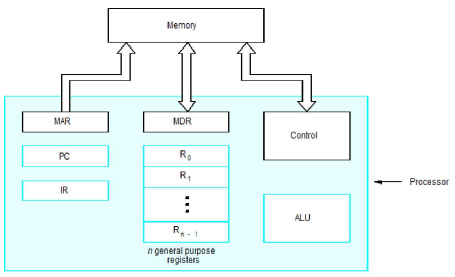
- Not visible under user mode

1. Instruction Register (IR) – used to store the instructions
2. Program Counter (PC) – gives the address of next instruction to be executed
3. Memory address register (MAR) – holds the address of the memory to read/write data
4. Memory data register (MDR) – holds the data/instruction fetched from the memory

Examples of Register Organization



Connection Between the Processor and the Memory

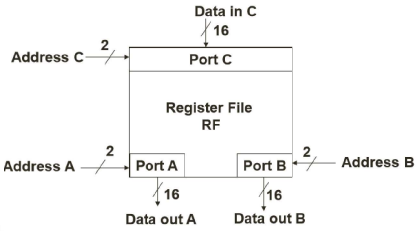


Register Files

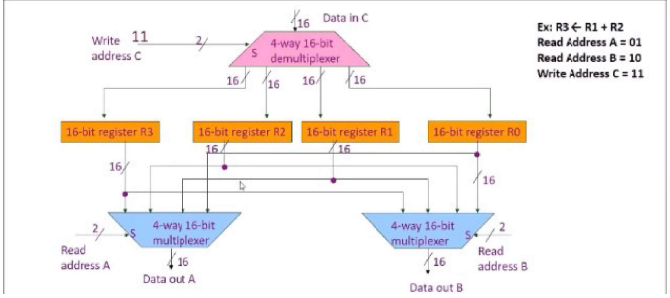
- The register file is the component that contains all the general purpose registers of the microprocessor.
- It is an array of processor register in a CPU
- It can be implemented using SRAM with multiple ports
- SRAMs are distinguished by having dedicated read and write ports, whereas ordinary multiported SRAMs will usually read and write through the same ports.

Register Files

- A simple register file is a set of registers and a decoder. The register file requires an address and a data input
- To read two values at once and write one value back in a single cycle. Consider the following equation: C = A + B



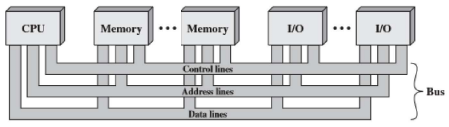
A Register File with three access ports – logic diagram

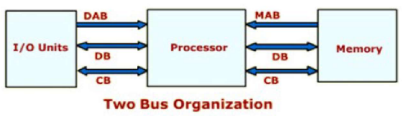


Interconnection of Components

- Functional components of the computer needs to be interconnected
- A group of lines/wires that serves as a connecting path for several devices is called a Bus
- The collection of paths connecting the various modules is called the interconnection structure
- Bus that connects the major components are called System Bus
- Bus is a shared transmission medium

Bus Structures





Two Bus Organization

Speed Issue

- Different devices have different transfer/operate speed.
- If the speed of bus is bounded by the slowest device connected to it, the efficiency will be very low.
- How to solve this?
- A common approach – use buffers.

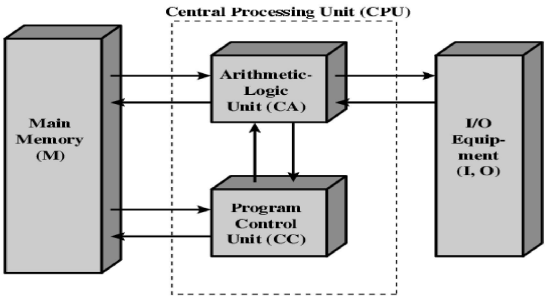
Computer Architecture

- Broadly can be classified into two types
  - Von Neumann architecture
  - Harvard architecture

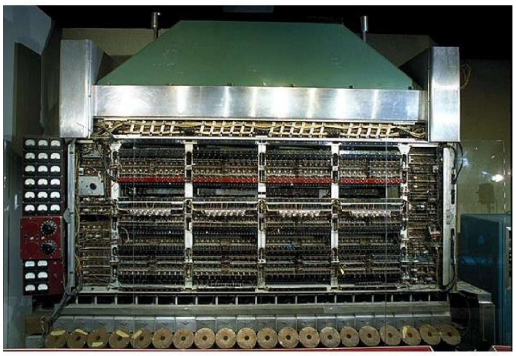
Von Neumann Architecture

- The stored program concept is given by a mathematician Von Neumann
- In 1946, Von Neumann and his colleagues began the design of a new stored program computer, referred as IAS computer at the Princeton Institute for Advanced Studies
- General structure of IAS computer
  - Instructions and data are stored in the same memory module
  - More flexible and easier to implement
  - Suitable for most of the general purpose processors
  - Disadvantage: All instructions and data are moved back and forth through the same pipe

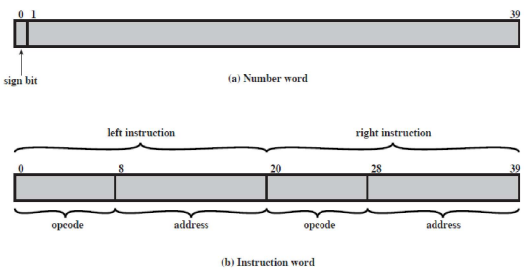
Structure of Von Neumann Machine



The IAS Computer, 1952



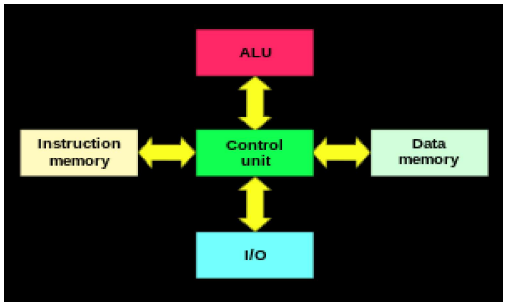
IAS Memory Format



IAS Memory Format

- 1000 x 40 bit words (1000 storage locations of 40 binary bits each)
  - Both instructions and data stored here
- Number format
  - Each number is represented by a sign bit and a 39 bit value
- Instruction format
  - A word may contain two 20 bit instructions
  - 8 bit operation code (opcode) : specifying the operation to be performed and
  - 12 bit address : designating one of the word in memory (0 to 999)

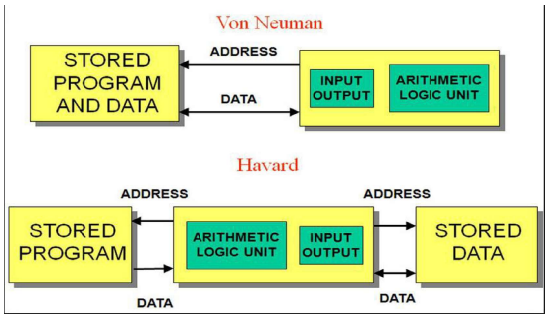
Harvard Architecture



Harvard Architecture

- The Harvard architecture is a computer architecture with physically separate memory
- Separate memory for program and data
  - Instruction are stored in program memory
  - Data stored in data memory
- In Harvard architecture, there is no need to make the two memories shared characteristics
- In particular, word width and memory address structure can differ
- In some systems, instructions can be stored in read-only memory whereas the data stored in read-write memory
- In some systems, there is much more instruction memory than data memory so instruction addresses is wider than data addresses

Von Neumann vs Harvard architecture



Pipeline in Executing Instructions

- Pipeline is the concept to speedup the execution
- Pipeline means overlap execution
- Instruction execution is typically divided into 5 stages
  - Instruction Fetch (IF)
  - Instruction Decode (ID)
  - ALU operation (EX)
  - Memory Access (MEM)
  - Write Back result to register file (WB)
- These five stages can be executed in an overlapped fashion in pipeline architecture

Basic 5-stage Pipelining Diagram

Instruction	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

Performance

- The most important measure of a computer is how quickly it can execute programs.
- Three factors affect performance:
  - Hardware design
  - Instruction set
  - Compiler

Performance

- Processor time to execute a program depends on the hardware involved in the execution of individual machine instructions.
- The processor and a relatively small cache memory can be fabricated on a single integrated circuit chip.
- Speed
  - The internal speed of performing the basic steps of instruction processing on cache is very high
- Cost
- Memory management

Processor Clock

- Clock, clock cycle, and clock rate
- Processor circuits are controlled by a timing signal called Clock
- The clock defines regular time intervals called clock cycles
- To execute a machine instruction, the processor divides the action to be performed into sequence of basic steps such that each step can be completed in one clock cycle
- Let P be the length of one clock cycle. Its inverse R=1/P is called clock rate, which is measured in cycles per second.

Basic Performance Equation

- T – processor time required to execute a program that has been prepared in high-level language
- N – number of actual machine language instructions needed to complete the execution (note: loop)
- S – average number of basic steps needed to execute one machine instruction. Each step completes in one clock cycle
- R – clock rate

$$T = \frac{N \times S}{R}$$

How to improve T?

- Instructions are not necessarily executed one after another.
- The value of S doesn't have to be the number of clock cycles to execute one instruction.
- Pipelining – overlapping the execution of successive instructions.
- Add R1, R2, R3
- Superscalar operation – multiple instruction pipelines are implemented in the processor.
- Goal – reduce S (could become <1!)

Clock Rate

- Increase clock rate
  - Improve the integrated-circuit (IC) technology to make the circuits faster
  - Reduce the amount of processing done in one basic step (however, this may increase the number of basic steps needed)
- Increases in R that are entirely caused by improvements in IC technology affect all aspects of the processor's operation equally except the time to access the main memory.

Compiler

- A compiler translates a high-level language program into a sequence of machine instructions.
- To reduce N, we need a suitable machine instruction set and a compiler that makes good use of it.
- Goal – reduce N×S
- A compiler may not be designed for a specific processor; however, a high-quality compiler is usually designed for, and with, a specific processor.

Performance Measurement

- T is difficult to compute.
- Measure computer performance using benchmark programs.
- System Performance Evaluation Corporation (SPEC) selects and publishes representative application programs for different application domains, together with test results for many commercially available computers.
- Compile and run (no simulation)
- Reference computer

$$SPEC\ \ rating = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test}}$$

$$SPEC\ \ rating = \left(\prod_{i=1}^{n} SPEC_i\right)^{\frac{1}{n}}$$

Instruction formats

Instruction Formats

Instruction sets are differentiated by the following:

- Number of bits per instruction.
- Stack-based or register-based.
- Number of explicit operands per instruction.
- Operand location.
- Types of operations.
- Type and size of operands.

Instruction Formats

Instruction set architectures are measured according to:

- Main memory space occupied by a program.
- Instruction complexity.
- Instruction length (in bits).
- Total number of instructions in the instruction set.

Instruction Formats

In designing an instruction set, consideration is given to:

- Instruction length.
  - Whether short, long, or variable.
- Number of operands.
- Number of addressable registers.
- Memory organization.
  - Whether byte- or word addressable.
- Addressing modes.
  - Choose any or all: direct, indirect or indexed.

Instruction Formats

- Byte ordering, or *endianness*, is another major architectural consideration.
- If we have a two-byte integer, the integer may be stored so that the least significant byte is followed by the most significant byte or vice versa.
  - In *little endian* machines, the least significant byte is followed by the most significant byte.
  - *Big endian* machines store the most significant byte first (at the lower address).

Instruction Formats

- As an example, suppose we have the hexadecimal number 12345678.
- The big endian and small endian arrangements of the bytes are shown below.

Address ⟶	00	01	10	11
Big Endian	12	34	56	78
Little Endian	78	56	34	12

Instruction Formats

- Big endian:
  - Is more natural.
  - The sign of the number can be determined by looking at the byte at address offset 0.
  - Strings and integers are stored in the same order.
- Little endian:
  - Makes it easier to place values on non-word boundaries.
  - Conversion from a 16-bit integer address to a 32-bit integer address does not require any arithmetic.

Instruction Formats

- The next consideration for architecture design concerns how the CPU will store data.
- We have three choices:
  1. A stack architecture
  2. An accumulator architecture
  3. A general purpose register architecture.
- In choosing one over the other, the tradeoffs are simplicity (and cost) of hardware design with execution speed and ease of use.

Instruction Formats

- In a stack architecture, instructions and operands are implicitly taken from the stack.
  - A stack cannot be accessed randomly.
- In an accumulator architecture, one operand of a binary operation is implicitly in the accumulator.
  - One operand is in memory, creating lots of bus traffic.
- In a general purpose register (GPR) architecture, registers can be used instead of memory.
  - Faster than accumulator architecture.
  - Efficient implementation for compilers.
  - Results in longer instructions.

Instruction Formats

- Most systems today are GPR systems.
- There are three types:
  - Memory-memory where two or three operands may be in memory.
  - Register-memory where at least one operand must be in a register.
  - Load-store where no operands may be in memory.
- The number of operands and the number of available registers has a direct affect on instruction length.

Instruction Formats

- Stack machines use one- and zero-operand instructions.
- **`LOAD`** and **`STORE`** instructions require a single memory address operand.
- Other instructions use operands from the stack implicitly.
- **`PUSH`** and **`POP`** operations involve only the stack's top element.
- Binary instructions (e.g., **`ADD`**, **`MULT`**) use the top two items on the stack.

Instruction Formats

- Stack architectures require us to think about arithmetic expressions a little differently.
- We are accustomed to writing expressions using *infix* notation, such as: Z = X + Y.
- Stack arithmetic requires that we use *postfix* notation: Z = XY+.
  - This is also called *reverse Polish notation*, (somewhat) in honor of its Polish inventor, Jan Lukasiewicz (1878 - 1956).

Instruction Formats

- The principal advantage of postfix notation is that parentheses are not used.
- For example, the infix expression,

  Z = (X × Y) + (W × U),

  becomes:

  Z = X Y × W U × +

  in postfix notation.

Instruction Formats

- In a stack ISA, the postfix expression,

  Z = X Y × W U × +

  might look like this:

```
PUSH X
PUSH Y
MULT
PUSH W
PUSH U
MULT
ADD
PUSH Z
```

Note: The result of a binary operation is implicitly stored on the top of the stack!

Instruction Formats

- In a one-address ISA, like MARIE, the infix expression,

  Z = X × Y + W × U

  looks like this:

```
LOAD X
MULT Y
STORE TEMP
LOAD W
MULT U
ADD TEMP
STORE Z
```


Instruction Formats

- In a two-address ISA, (e.g.,Intel, Motorola), the infix expression,

  Z = X × Y + W × U

  might look like this:

```
LOAD R1,X
MULT R1,Y
LOAD R2,W
MULT R2,U
ADD  R1,R2
STORE Z,R1
```

Note: One-address ISAs usually require one operand to be a register.

Instruction Formats

- With a three-address ISA, (e.g.,mainframes), the infix expression,

  Z = X × Y + W × U

  might look like this:

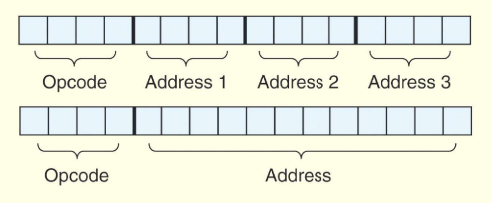
```
MULT R1,X,Y
MULT R2,W,U
ADD  Z,R1,R2
```

Instruction Formats

- We have seen how instruction length is affected by the number of operands supported by the ISA.
- In any instruction set, not all instructions require the same number of operands.
- Operations that require no operands, such as `HALT`, necessarily waste some space when fixed-length instructions are used.
- One way to recover some of this space is to use expanding opcodes.

Instruction Formats

- A system has 16 registers and 4K of memory.
- We need 4 bits to access one of the registers. We also need 12 bits for a memory address.
- If the system is to have 16-bit instructions, we have two choices for our instructions:



Instruction Formats

- If we allow the length of the opcode to vary, we could create a very rich instruction set:

```
0000 R1   R2   R3      } 15 3-address codes
1110 R1   R2   R3      }

1111 0000 R1   R2      } 14 2-address codes
1111 1101 R1   R2      }

1111 1110 0000 R1      } 31 1-address codes
1111 1111 1110 R1      }

1111 1111 1111 0000    } 16 0-address codes
1111 1111 1111 1111    }
```


Instruction types

Instructions fall into several broad categories that you should be familiar with:

- Data movement.
- Arithmetic.
- Boolean.
- Bit manipulation.
- I/O.
- Control transfer.
- Special purpose.

Addressing

- Addressing modes specify where an operand is located.
- They can specify a constant, a register, or a memory location.
- The actual location of an operand is its *effective address*.
- Certain addressing modes allow us to determine the address of an operand dynamically.

Addressing

- *Immediate addressing* is where the data is part of the instruction.
- *Direct addressing* is where the address of the data is given in the instruction.
- *Register addressing* is where the data is located in a register.
- *Indirect addressing* gives the address of the address of the data in the instruction.
- *Register indirect addressing* uses a register to store the address of the address of the data.

Addressing

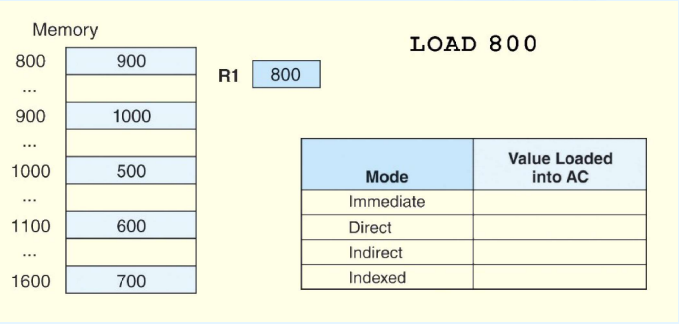
- *Indexed addressing* uses a register (implicitly or explicitly) as an offset, which is added to the address in the operand to determine the effective address of the data.
- *Based addressing* is similar except that a base register is used instead of an index register.
- The difference between these two is that an index register holds an offset relative to the address given in the instruction, a base register holds a base address where the address field represents a displacement from this base.

Addressing

- In *stack addressing* the operand is assumed to be on top of the stack.
- There are many variations to these addressing modes including:
  - Indirect indexed.
  - Base/offset.
  - Self-relative
  - Auto increment - decrement.
- We won't cover these in detail.

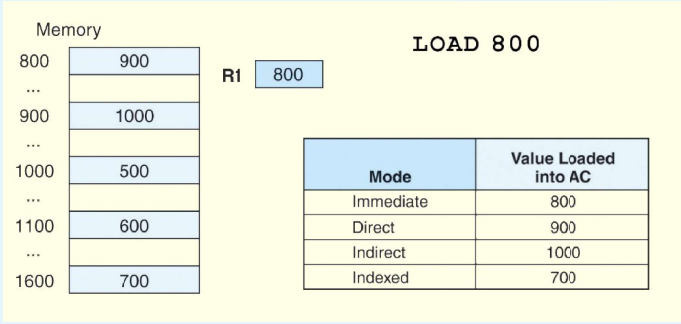
Addressing

- For the instruction shown, what value is loaded into the accumulator for each addressing mode?



Addressing

- These are the values loaded into the accumulator for each addressing mode.



Calculate the effective address for the following addressing modes?

Immediate addressing mode

Add A, #100

Effective address = 100

1) R=1000, [1000]= 50

2) Register Direct Addressing Mode

Add R, #50 ; R ← R + 50 R← 1000+50

EA = R

Add [R],50 R ← 50+50

EA =[R]

- Register Indirect Add [[R]], 50

  R = 1000, [1000] = [2000] = 40

  EA =[[R]]

Memory Direct : Add [1000], 50

[1000] = 40

EA =[1000]

Memory Indirect : Add[[1000], 50

[1000] = [2000] = 60

EA = [[1000]]

- Index addressing mode (SI , DI)

EA = BX / memory location + SI/DI + offset

Bx= 1000, SI = 20 , offset = 3

EA = 1000+ 20 + 3 = 1023 → 50

Relative addressing mode (PC) Pc ← 5000

EA = Pc + SI/DI + offset

Pc =5000, Si = 200, offset = 5

EA = [5205]

Auto increment : [R]+ R ← R +1

R = 400= 401

Auto Decremnt : [R]- R ← R-1

R= 399

- Program starts from the location 1000.
- 1000 : Add A, 100
- Add [A], 1
- Add [R], 2

Add[[R]], 2

Add [1008], 3

Add [[1008]], 4

Add [Bx+Si+offset], 3

add [Pc], 2

Memory location details

Memory	Data
0009	6
1002	1008
1008	1100
1100	50
1101	7
1102	9
1103	8

R = 1102, Bx = 1000, SI = 100, offset = 2

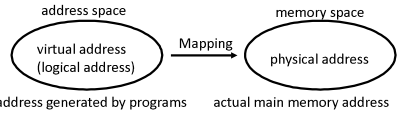
Effective address of all the addressing mode.

Virtual Memory & Replacement Algorithm

VIRTUAL MEMORY

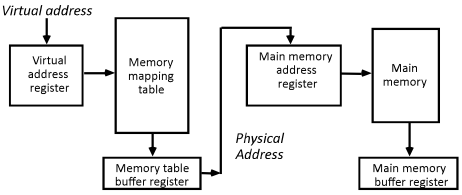
Give the programmer the illusion that the system has a very large memory, even though the computer actually has a relatively small main memory

Address Space(Logical) and Memory Space(Physical)



Address Mapping

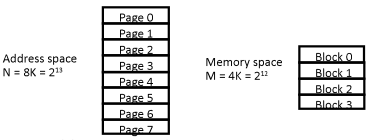
Memory *Mapping Table* for *Virtual Address -> Physical Address*



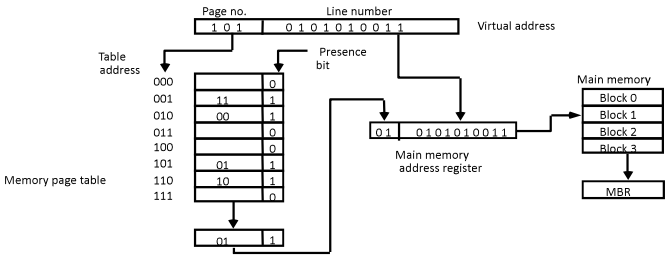
ADDRESS MAPPING

Address Space and Memory Space are each divided into fixed size group of words called *blocks* or *pages*

1K words group



Organization of memory Mapping Table in a paged system



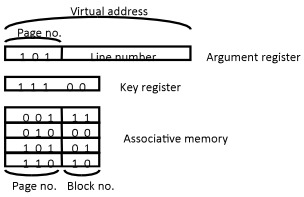
- Memory Page table consist of Eight words ,one for each page.
- The address in the page table denotes the Page number, and the content of the word gives the block number where the page is stored.
- A presence bit indicates whether the page has been transferred from auxiliary memory in to main memory
- If the page is not there call to the operating system is then generated to fetch the required page from auxiliary memory to main memory

ASSOCIATIVE MEMORY PAGE TABLE

Assume that
Number of Blocks in memory = m
Number of Pages in Virtual Address Space = n

Page Table
- Straight forward design -> n entry table in memory
Inefficient storage space utilization
<- n-m entries of the table is empty

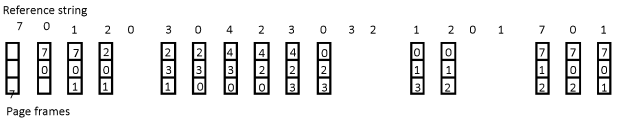
- More efficient method is m-entry Page Table
Page Table made of an Associative Memory
m words; (Page Number: Block Number)



Page Fault
Page number cannot be found in the Page Table

PAGE REPLACEMENT ALGORITHMS

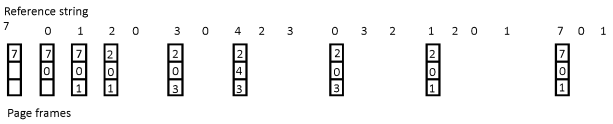
FIFO



FIFO algorithm selects the page that has been in memory the longest time
Using a queue - every time a page is loaded, its
identification is inserted in the queue
-
Easy to implement
May result in a frequent page fault

Optimal Replacement (OPT) - Lowest page fault rate of all algorithms

Replace that page which will not be used for the longest period of time

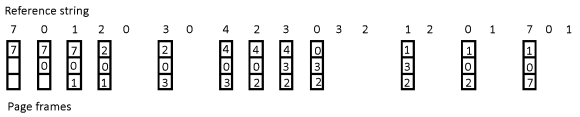


PAGE REPLACEMENT ALGORITHMS

LRU

- OPT is difficult to implement since it requires future knowledge
- LRU uses the recent past as an approximation of near future.

Replace that page which has not been used for the longest period of time



- LRU may require substantial hardware assistance
- The problem is to determine an order for the frames defined by the time of last use

Problems in Page Replacement

- Consider a paging system in which M1 has a capacity of three pages. The execution of a program Q requires reference to five distinct pages Pi, where i= 1,2,3,4,5 and i is the page address. The page address stream formed by executing Q is

2 3 2 1 5 2 4 5 3 2 5 2

Problem2

- A virtual memory system has an address space of 8K words, a memory space of 4K words, and page and block sizes of 1K words. The following page reference chanses occur during a given time interval.

4 2 0 1 2 6 1 4 0 1 0 2 3 5 7

Determine the four pages that are resident in main memory after each page reference change if the replacement algorithm used is FIFO, LRU and OPT.

Memory Portions

- Key Characteristics – William Stallings
- Main memory Organization (1-D,2-D,Static and dynamic RAM)–hayes[6], ppt, william Stallings.
- Access methods – William Stallings
- Design Problems – PPT, Morris Mano, Hayes
- Memory Hierarchy – Morris Mano(12.1), Hayes (6.1)
- Cache Memories –Morris Mano (12.5),PPT, William Stallings.
- Virtual Memory and replacement Alg –Morris Mano (12.6)

Unit –II reference Materials

Hardware and software implementation of arithmetic unit for common arithmetic operations: addition, subtraction, multiplication, division(Fixed point and floating point) = Refer Morris Mano chapter 10.

Data Representation = ppt.

Representation of non-numeric data (character codes, graphical data) = ppt, morris mano pg:383

COMPUTER ARITHMETIC

Digital Hardware Algorithms

- Arithmetic operations
- Addition, subtraction, multiplication, division
- Data types
- Fixed-point binary
- Signed-magnitude representation
- Signed-2's complement representation
- Floating-point binary
- Binary-coded decimal (BCD)

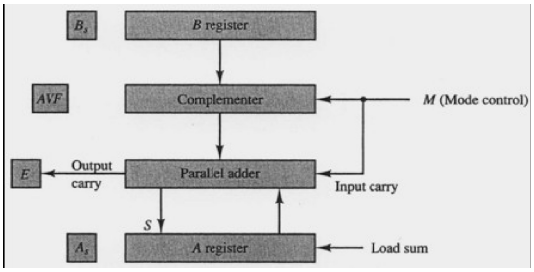
Algorithm

- When the sign of A & B are identical add the two magnitudes and attach the sign of A to the result.
- When the sign of A & B are different compare the magnitudes and subtract the smaller no from the larger.
- Choose the sign of the result to be same as A if A>B or the complement of the sign of A if A<B.
- If two magnitudes are equal , subtract B from A and make the sign of the result positive.

Add / Subtract Signed-Magnitude

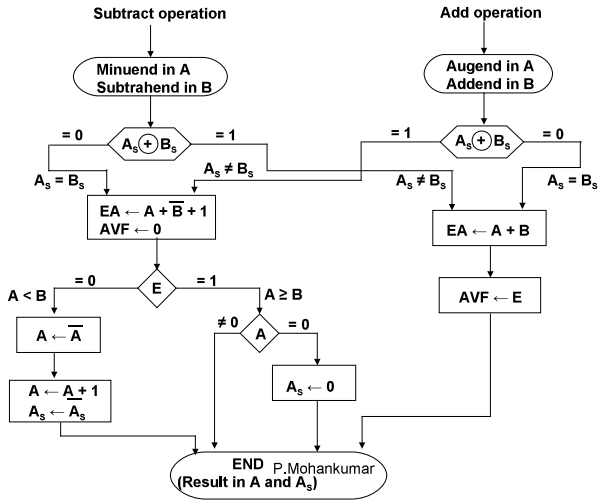
Operation	Add Magnitudes	Subtract Magnitudes: When $A > B$	When $A < B$	When $A = B$
$(+A) + (+B)$	$+(A + B)$			
$(+A) + (-B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(-A) + (+B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$
$(-A) + (-B)$	$-(A + B)$			
$(+A) - (+B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(+A) - (-B)$	$+(A + B)$			
$(-A) - (+B)$	$-(A + B)$			
$(-A) - (-B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$

Hardware

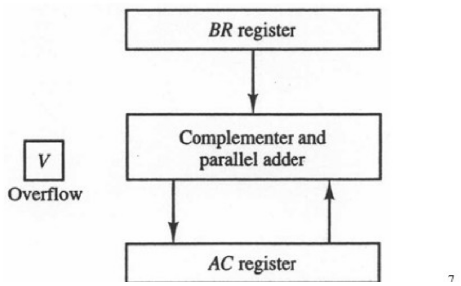


- Description
- AS Sign of A
- BS Sign of B
- AS & A Accumulator
- AVF Overflow bit for $A + B$
- E Output carry for parallel adder

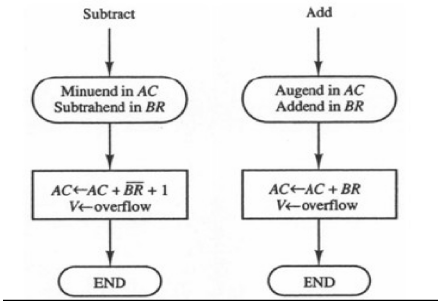
Flow chart for add and subtract operations



Add / Sub Signed-2's Complement



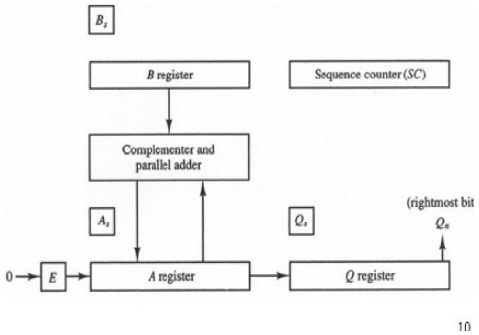
Algorithm



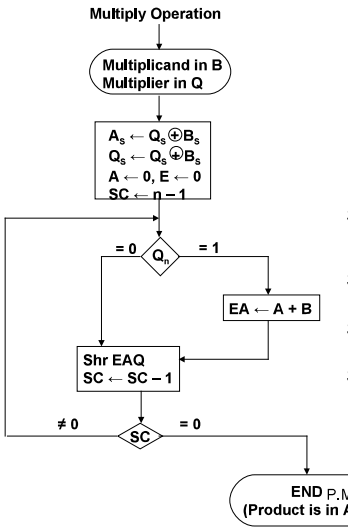
Multiply Signed-Magnitude

- Series of successive shift and add operations
- 23 10111 Multiplicand
  19 x 10011 Multiplier
  10111
  10111
  00000
  00000
  10111 +
  437 110110101 Product

Hardware



Flow chart for Multiplication (Signed Magnitude Representation)



23 × 19 = 437
Q = 19 (10011), B = 23 (10111)

	E	A	Q	SC
	0	00000	10011	101
Add		10111		
		10111		
Shift	0	01011	11001	100
Add		10111		
	1	00010		
Shift	0	10001	01100	011
Shift	0	01000	10110	010
Shift	0	00100	01011	001
Add		10111		
		11011		
Shift	0	01101	10101	000

Description

- Q multiplier
- B multiplicand
- A 0
- SC number of bits in multiplier
- E overflow bit for A
- Do SC times
  - If low-order bit of Q is 1
    - $A \leftarrow A + B$
  - Shift right EAQ
- Product is in AQ

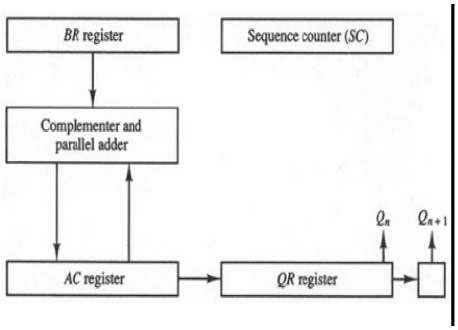
Multiply Signed-2's Complement

- Booth algorithm
- QR multiplier
- Q_n least significant bit of QR
- Q_{n+1} previous least significant bit of QR
- BR multiplicand
- AC 0
- SC number of bits in multiplier

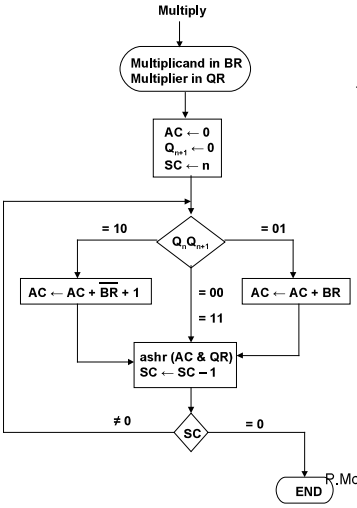
Algorithm

- Do $SC + 1$ times
  - $Q_nQ_{n+1} = 10$
    - $AC \leftarrow AC + \overline{BR} + 1$
  - $Q_nQ_{n+1} = 01$
    - $AC \leftarrow AC + BR$
  - Arithmetic shift right AC & QR
  - $SC \leftarrow SC - 1$

Hardware



Flowchart for Booth Multiplication



Example: -19 × -23 = 437
BR = 10111, $\overline{BR}$ + 1 = 01001

Comment	AC	QR	Q_{n+1}	SC
	00000	10011	0	5
Subtract BR	01001			
	01001			
Ashr	00100	11001	1	4
Ashr	00010	01100	1	3
Add BR	10111			
	11001			
Ashr	11100	10110	0	2
Ashr	11110	01011	0	1
Subtract BR	01001			
	00111			
Ashr	00011	10101	1	0

Binary Division

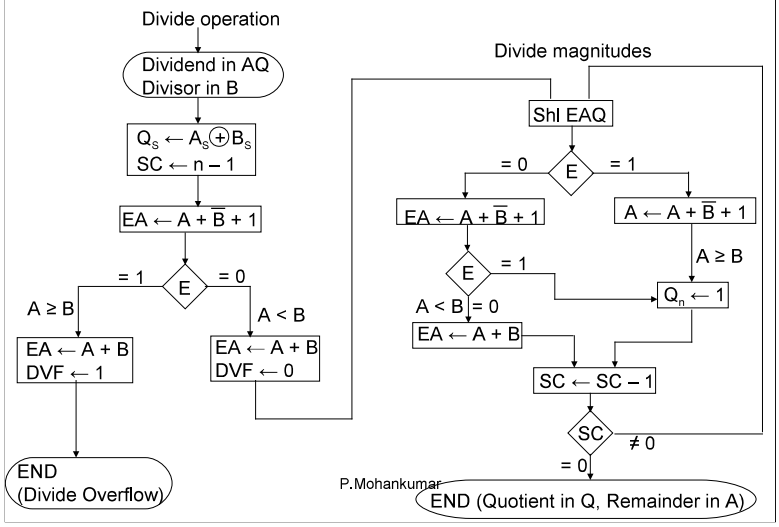
Example (Paper and Pencil Method)

Divisor: B = 1 0 0 0 1

```
          1 1 0 1 0          Quotient = Q
        ---------------
1 0 0 0 1 | 0 1 1 1 0 0 0 0 0   Dividend = A
          | 0 1 1 1 0          5 bits of A < B, quotient has 5 bits
          | 0 1 1 1 0 0       6 bits of A ≥ B
          |   1 0 0 0 1        Shift right B and subtract; enter 1 in Q
          | 0 1 0 1 1 0        6 bits of remainder ≥ B
          |   1 0 0 0 1        Shift right B and subtract; enter 1 in Q
          | 0 0 1 0 1 0        Remainder < B, enter 0 in Q; shift right B
          | 0 0 1 0 1 0 0      Remainder ≥ B
          |     1 0 0 0 1      Shift right B and subtract; enter 1 in Q
          | 0 0 0 1 1 0        Remainder < B, enter 0 in Q
```

Final remainder: 000110

Signed magnitude Division Flow chart



Signed – Magnitude Division

Divisor B = 10001
$\overline{B}$ + 1 = 01111

Comment	E	A	Q	SC
Dividend:		0 1 1 1 0	0 0 0 0 0	5
Shl EAQ	0	1 1 1 0 0	0 0 0 0 0	
Add $\overline{B}$ + 1		0 1 1 1 1		
E = 1	1	0 1 0 1 1		
Set Q_n = 1	1	0 1 0 1 1	0 0 0 0 1	4
Shl EAQ	0	1 0 1 1 0	0 0 0 1 0	
Add $\overline{B}$ + 1		0 1 1 1 1		
E = 1	1	0 0 1 0 1		
Set Q_n = 1	1	0 0 1 0 1	0 0 0 1 1	3
Shl EAQ	0	0 1 0 1 0	0 0 1 1 0	
Add $\overline{B}$ + 1		0 1 1 1 1		
E = 0, leave Q_n = 0	0	1 1 0 0 1	0 0 1 1 0	
Add B		1 0 0 0 1		

Divisor B = 10001
$\overline{B}$ + 1 = 01111

Comment	E	A	Q	SC
Restore remainder	1	0 1 0 1 0		2
Shl EAQ	0	1 0 1 0 0	0 1 1 0 0	
Add $\overline{B}$ + 1		0 1 1 1 1		
E = 1	1	0 0 0 1 1		
Set Q_n = 1	1	0 0 0 1 1	0 1 1 0 1	1
Shl EAQ	0	0 0 1 1 0	1 1 0 1 0	
Add $\overline{B}$ + 1		0 1 1 1 1		
E = 0, leave Q_n = 0	0	1 0 1 0 1	1 1 0 1 0	
Add B		1 0 0 0 1		
Restore remainder	1	0 0 1 1 0	1 1 0 1 0	0
Neglect E				
Remainder in A:		0 0 1 1 0		
Quotient in Q			1 1 0 1 0	

10-10(b) $\frac{1111}{0011} = 0101$ $B = 0011$ $\overline{B}+1 = 1101$

	E	A	Q	SC
Dividend in Q, A=0		0000	1111	100
shl EAQ		0001	1110	
add $\overline{B}$+1		1101		
E=0, leave Q_n=0	0	1110	1110	
add B		0011		
restore partial remainder		0001		011
shl EAQ		0011	1100	
add $\overline{B}$+1		1101		
E=1, set Q_n to 1	1	0000	1101	010
shl EAQ		0001	1010	
add $\overline{B}$+1		1101		
E=0, leave Q_n=0	0	1110	1010	
add B		0011		
restore partial remainder		0001		001
shl EAQ		0011	0100	
add $\overline{B}$+1		1101		
E=1, set Q_n to 1	1	0000	0101	000
		remainder	quotient	

10-11

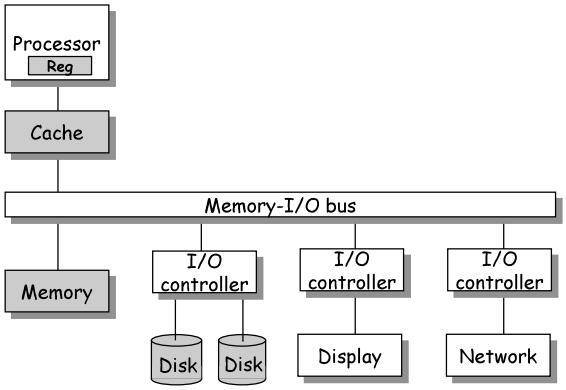
P.Mohankumar

Memory Technology

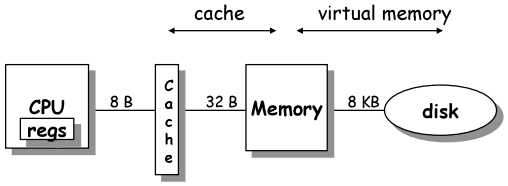
Topics

- Memory Hierarchy Basics
- Static RAM
- Dynamic RAM
- Magnetic Disks
- Access Time Gap

Computer System



Levels in Memory Hierarchy



	Register	Cache	Memory	Disk Memory
size:	200 B	32KB - 4MB	128 MB	20 GB
speed:	2 ns	4 ns	60 ns	8 ms
$/Mbyte:		$100/MB	$1.50/MB	$0.05/MB
block size:	8 B	32 B	8 KB	

larger, slower, cheaper →

Scaling to 0.1μm

- **Semiconductor Industry Association, 1992 Technology Workshop**
  - Projected future technology based on past trends

	1992	1995	1998	2001	2004	2007
Feature size:	0.5	0.35	0.25	0.18	0.12	0.10
DRAM capacity:	16M	64M	256M	1G	4G	16G
Chip area (cm^2):	2.5	4.0	6.0	8.0	10.0	12.5

- *Industry is slightly ahead of projection*
- *Doubles every 1.5 years*
- *Prediction on track*
- *Way off! Chips staying small*

Static RAM (SRAM)

Fast
- ~4 nsec access time

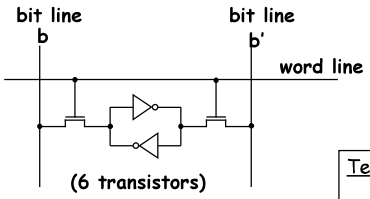
Persistent
- as long as power is supplied
- no refresh required

Expensive
- ~$100/MByte
- 6 transistors/bit

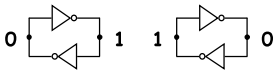
Stable
- High immunity to noise and environmental disturbances

Technology for caches

Anatomy of an SRAM Cell



Stable Configurations



Terminology:
- *bit line:* carries data
- *word line:* used for addressing

Write:
1. set bit lines to new data value
   - 'b' is set to the opposite of **b**
2. raise word line to "high"

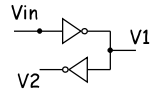
⇒ sets cell to new state (may involve flipping relative to old state)

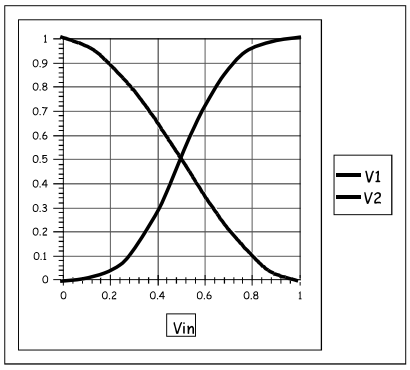
Read:
1. set bit lines high
2. set word line high
3. see which bit line goes low

SRAM Cell Principle

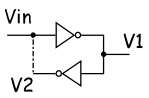
Inverter Amplifies
- Negative gain
- Steps = -1 in middle
- Saturates at ends

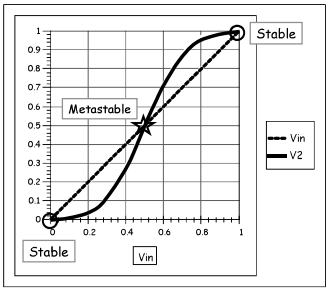
Inverter Pair Amplifies
- Positive gain
- Steps = 1 in middle
- Saturates at ends





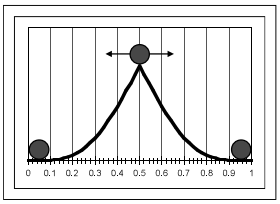
Bistable Element



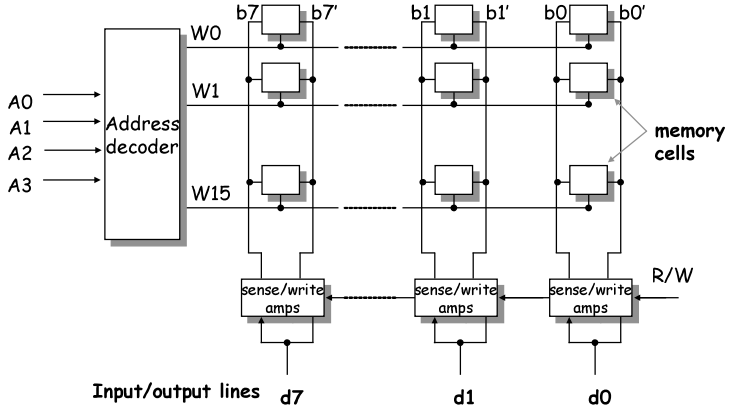


Stability
- **Require Vin = V2**
- **Stable at endpoints**
  - recover from pertubation
- **Metastable in middle**
  - Fall out when perturbed

Ball on Ramp Analogy



Example SRAM Configuration (16 x 8)



Dynamic RAM (DRAM)

Slower than SRAM
- access time ~60 nsec

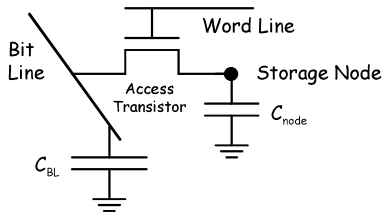
Nonpersistant
- every row must be accessed every ~1 ms (refreshed)

Cheaper than SRAM
- ~$1.50 / MByte
- 1 transistor/bit

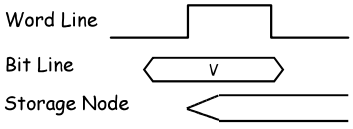
Fragile
- electrical noise, light, radiation

Workhorse memory technology

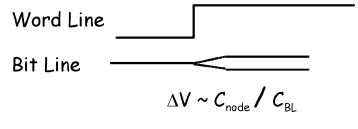
Anatomy of a DRAM Cell



Writing



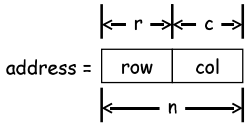
Reading



Addressing Arrays with Bits

Array Size
- R rows, R = 2^r
- C columns, C = 2^c
- N = R * C bits of memory

Addressing
- Addresses are n bits, where N = 2^n
- row(address) = address / C
  - leftmost r bits of address
- col(address) = address % C
  - rightmost bits of address

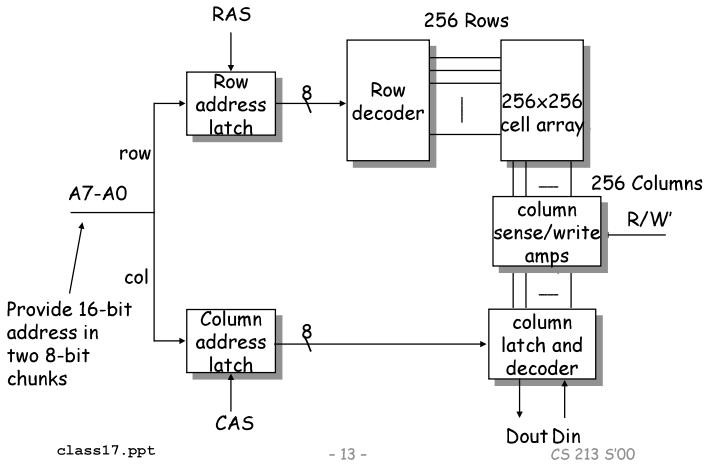


Example
- R = 2
- C = 4
- address = 6

	0	1	2	3
0	000	001	010	011
1	100	101	110	111

row 1 col 2

Example 2-Level Decode DRAM (64Kx1)



DRAM Operation

Row Address (~50ns)
- Set Row address on address lines & strobe RAS
- Entire row read & stored in column latches
- Contents of row of memory cells destroyed

Column Address (~10ns)
- Set Column address on address lines & strobe CAS
- Access selected bit
  - READ: transfer from selected column latch to Dout
  - WRITE: Set selected column latch to Din

Rewrite (~30ns)
- Write back entire row

Observations About DRAMs

Timing
- Access time (= 60ns) < cycle time (= 90ns)
- Need to rewrite row

Must Refresh Periodically
- Perform complete memory cycle for each row
- Approximately once every 1ms
- Sqrt(n) cycles
- Handled in background by memory controller

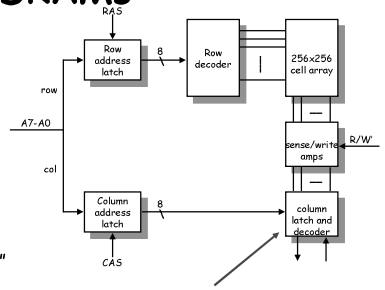
Inefficient Way to Get a Single Bit
- Effectively read entire row of Sqrt(n) bits

Enhanced Performance DRAMs

Conventional Access
- Row + Col
- RAS CAS RAS CAS ...

Page Mode
- Row + Series of columns
- RAS CAS CAS CAS ...
- Gives successive bits

Other Acronyms
- EDORAM
  - "Extended data output"
- SDRAM
  - "Synchronous DRAM"



Typical Performance

row access time	col access time	cycle time	page mode cycle time
50ns	10ns	90ns	25ns

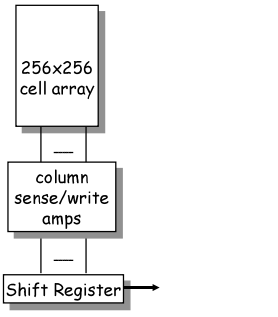
Video RAM

Performance Enhanced for Video / Graphics Operations
- Frame buffer to hold graphics image

Writing
- Random access of bits
- Also supports rectangle fill operations
  - Set all bits in region to 0 or 1

Reading
- Load entire row into shift register
- Shift out at video rates

Performance Example
- 1200 X 1800 pixels / frame
- 24 bits / pixel
- 60 frames / second
- 2.8 GBits / second



DRAM Driving Forces

Capacity

- 4X per generation
  - Square array of cells
- Typical scaling
  - Lithography dimensions 0.7X
    - » Areal density 2X
  - Cell function packing 1.5X
  - Chip area 1.33X
- Scaling challenge
  - Typically C_{node} / C_{BL} = 0.1–0.2
  - Must keep C_{node} high as shrink cell size

Retention Time

- Typically 16–256 ms
- Want higher for low-power applications

DRAM Storage Capacitor

Planar Capacitor

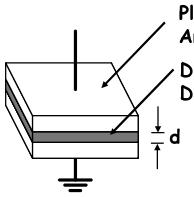
- Up to 1Mb
- C decreases linearly with feature size

Trench Capacitor

- 4–256 Mb
- Lining of hole in substrate

Stacked Cell

- > 1Gb
- On top of substrate
- Use high ε dielectric

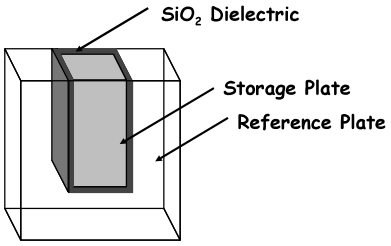


$C = \varepsilon A/d$

Trench Capacitor

Process

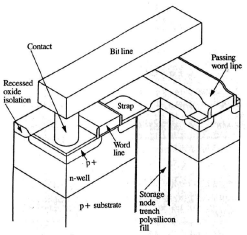
- Etch deep hole in substrate
  - Becomes reference plate
- Grow oxide on walls
  - Dielectric
- Fill with polysilicon plug
  - Tied to storage node



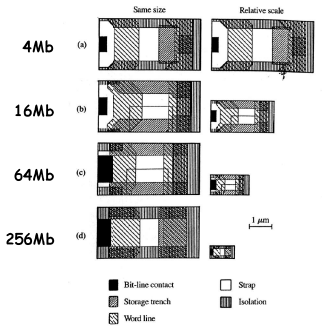
IBM DRAM Evolution

- IBM J. R&D, Jan/Mar '95
- Evolution from 4 – 256 Mb
- 256 Mb uses cell with area 0.6 μm²

4 Mb Cell Structure



Cell Layouts



Mitsubishi Stacked Cell DRAM

- IEDM '95
- Claim suitable for 1 – 4 Gb

Technology

- 0.14 μm process
  - Synchrotron X-ray source
- 8 nm gate oxide
- 0.29 μm² cell

Storage Capacitor

- Fabricated on top of everything else
- Rubidium electrodes
- High dielectric insulator
  - 50X higher than SiO_2
  - 25 nm thick
- Cell capacitance 25 femtofarads

Cross Section of 2 Cells

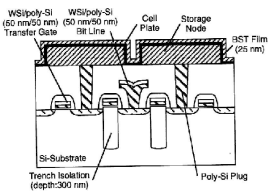


Fig. 2 Schematic cross-sectional view of DRAM memory cells with Ru/BST/Ru stacked capacitors.

Mitsubishi DRAM Pictures

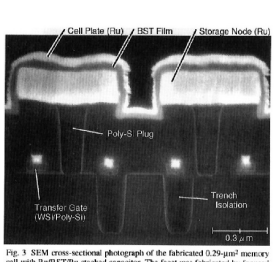


Fig. 3 SEM cross-sectional photograph of the fabricated 0.29-μm² memory cell with Ru/BST/Ru stacked capacitor. The facet was fabricated by focused ion beam etching.

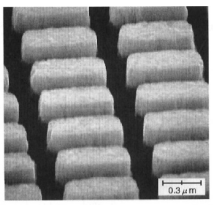


Fig. 8 SEM photograph of a Ru-metal storage node array with a projection a height of 0.2 μm.

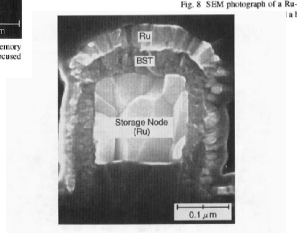
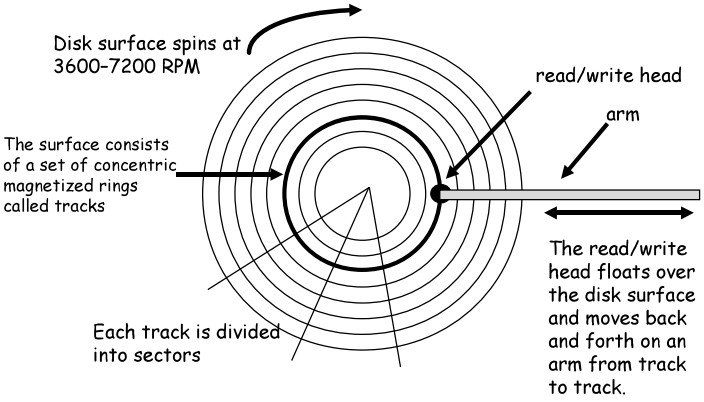


Fig. 10 SEM cross-sectional view of a Ru/BST/Ru capacitor cell. The facet shown is a cleaved facet.

Magnetic Disks



Disk Capacity

Parameter	18GB Example
• Number Platters	12
• Surfaces / Platter	2
• Number of tracks	6962
• Number sectors / track	213
• Bytes / sector	512
Total Bytes	**18,221,948,928**

Disk Operation

Operation
- Read or write complete sector

Seek
- Position head over proper track
- Typically 6-9ms

Rotational Latency
- Wait until desired sector passes under head
- Worst case: complete rotation
  10,025 RPM ⇒ 6 ms

Read or Write Bits
- Transfer rate depends on # bits per track and rotational speed
- E.g., 213 * 512 bytes @10,025RPM = 18 MB/sec.
- Modern disks have external transfer rates of up to 80 MB/sec
  - DRAM caches on disk help sustain these higher rates

Disk Performance

Getting First Byte
- Seek + Rotational latency = 7,000 – 19,000 µsec

Getting Successive Bytes
- ~ 0.06 µsec each
  - *roughly 100,000 times faster than getting the first byte!*

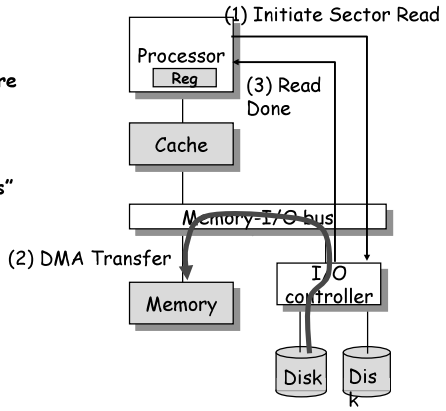
Optimizing Performance:
- Large block transfers are more efficient
- Try to do other things while waiting for first byte
  - switch context to other computing task
  - processor is interrupted when transfer completes

Disk / System Interface

1. Processor Signals Controller
- Read sector X and store starting at memory address Y

2. Read Occurs
- "Direct Memory Access" (DMA) transfer
- Under control of I/O controller

3. I/O Controller Signals Completion
- Interrupts processor
- Can resume suspended process



Magnetic Disk Technology

Seagate ST-12550N Barracuda 2 Disk

• Linear density	52,187.	bits per inch (BPI)
– Bit spacing	0.5	microns
• Track density	3,047.	tracks per inch (TPI)
– Track spacing	8.3	microns
• Total tracks	2,707.	tracks
• Rotational Speed	7200.	RPM
• Avg Linear Speed	86.4	kilometers / hour
• Head Floating Height	0.13	microns

Analogy:
- put the Sears Tower on its side
- fly it around the world, *2.5cm* above the ground
- each complete orbit of the earth takes *8 seconds*

CD Read Only Memory (CDROM)

Basis

- **Optical recording technology developed for audio CDs**
  - 74 minutes playing time
  - 44,100 samples / second
  - 2 X 16-bits / sample (Stereo)
  - ⇒ Raw bit rate = 172 KB / second
- **Add extra 288 bytes of error correction for every 2048 bytes of data**
  - Cannot tolerate any errors in digital data, whereas OK for audio

Bit Rate

- **172 * 2048 / (288 + 2048) = 150 KB / second**
  - For 1X CDROM
  - N X CDROM gives bit rate of N * 150
  - E.g., 12X CDROM gives 1.76 MB / second

Capacity

- **74 Minutes * 150 KB / second * 60 seconds / minute = 650 MB**

Storage Trends

SRAM

metric	1980	1985	1990	1995	2000	***2000:1980***
$/MB	19,200	2,900	320	256	100	***190***
access (ns)	300	150	35	15	2	***100***

DRAM

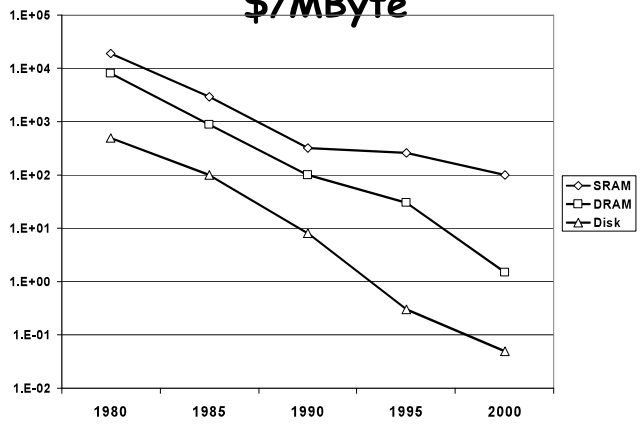
metric	1980	1985	1990	1995	2000	***2000:1980***
$/MB	8,000	880	100	30	1.5	***5,300***
access (ns)	375	200	100	70	60	***6***
typical size(MB)	0.064	0.256	4	16	64	***1,000***

Disk

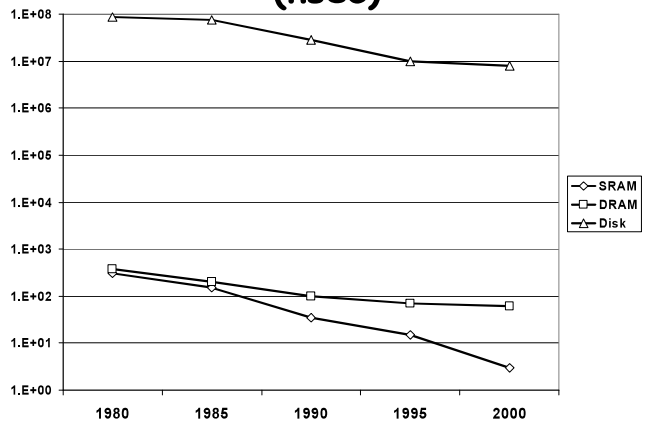
metric	1980	1985	1990	1995	2000	***2000:1980***
$/MB	500	100	8	0.30	0.05	***10,000***
access (ms)	87	75	28	10	8	***11***
typical size(MB)	1	10	160	1,000	9,000	***9,000***

(Culled from back issues of Byte and PC Magazine)

Storage Price: $/MByte



Storage Access Times (nsec)



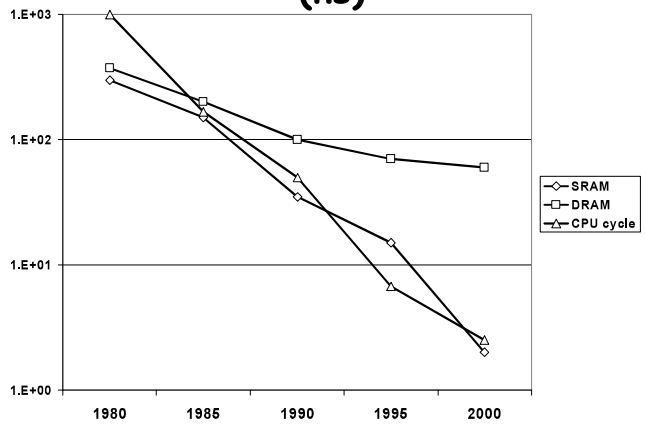
Processor clock rates

Processors

metric	1980	1985	1990	1995	2000	***2000:1980***
typical clock(MHz)	1	6	20	150	600	**600**
processor	8080	286	386	Pentium	P-III	

culled from back issues of Byte and PC Magazine

The CPU vs. DRAM Latency Gap (ns)



Memory Technology Summary

Cost and Density Improving at Enormous Rates

Speed Lagging Processor Performance

Memory Hierarchies Help Narrow the Gap:

- Small fast SRAMS (cache) at upper levels
- Large slow DRAMS (main memory) at lower levels
- Incredibly large & slow disks to back it all up

Locality of Reference Makes It All Work

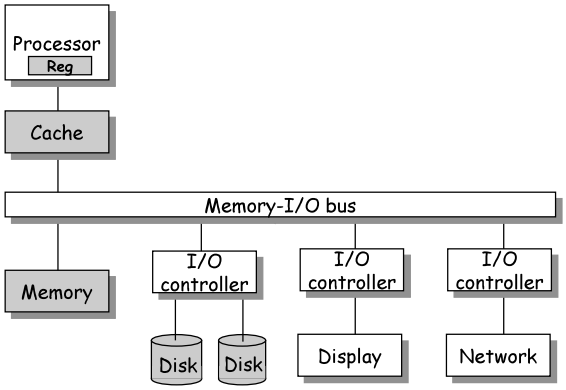
- Keep most frequently accessed data in fastest memory

Memory Technology

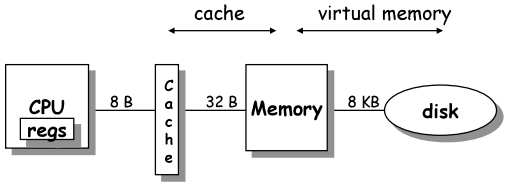
Topics

- Memory Hierarchy Basics
- Static RAM
- Dynamic RAM
- Magnetic Disks
- Access Time Gap

Computer System



Levels in Memory Hierarchy



	Register	Cache	Memory	Disk Memory
size:	200 B	32KB - 4MB	128 MB	20 GB
speed:	2 ns	4 ns	60 ns	8 ms
$/Mbyte:		$100/MB	$1.50/MB	$0.05/MB
block size:	8 B	32 B	8 KB	

larger, slower, cheaper →

Scaling to 0.1µm

- **Semiconductor Industry Association, 1992 Technology Workshop**
  - Projected future technology based on past trends

	1992	1995	1998	2001	2004	2007
Feature size:	0.5	0.35	0.25	0.18	0.12	0.10
- *Industry is slightly ahead of projection*						
DRAM capacity:	16M	64M	256M	1G	4G	16G
- *Doubles every 1.5 years*						
- *Prediction on track*						
Chip area (cm^2):	2.5	4.0	6.0	8.0	10.0	12.5
- *Way off! Chips staying small*						

Static RAM (SRAM)

Fast

- ~4 nsec access time

Persistent

- as long as power is supplied
- no refresh required

Expensive

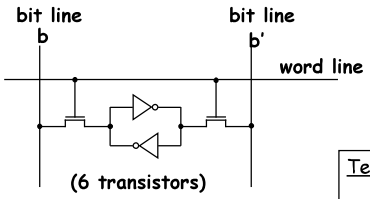
- ~$100/MByte
- 6 transistors/bit

Stable

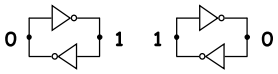
- High immunity to noise and environmental disturbances

Technology for caches

Anatomy of an SRAM Cell



Stable Configurations



Terminology:
- *bit line:* carries data
- *word line:* used for addressing

Write:
1. set bit lines to new data value
   - 'b' is set to the opposite of **b**
2. raise word line to "high"

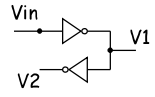
⇒ sets cell to new state (may involve flipping relative to old state)

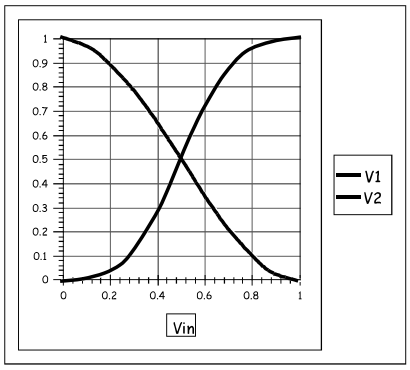
Read:
1. set bit lines high
2. set word line high
3. see which bit line goes low

SRAM Cell Principle

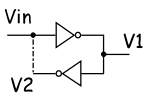
Inverter Amplifies
- Negative gain
- Steps = -1 in middle
- Saturates at ends

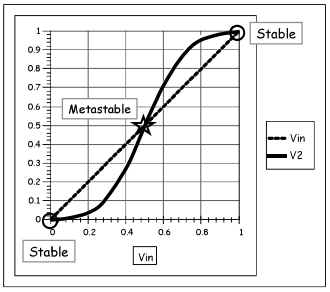
Inverter Pair Amplifies
- Positive gain
- Steps = 1 in middle
- Saturates at ends





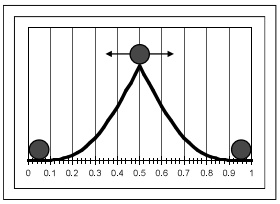
Bistable Element



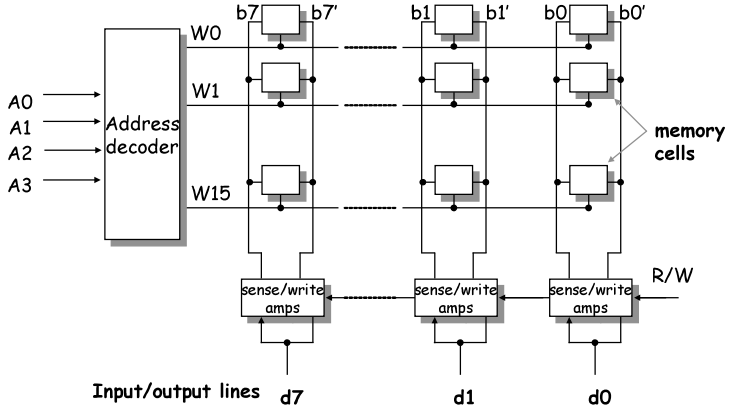


Stability
- **Require Vin = V2**
- **Stable at endpoints**
  - recover from pertubation
- **Metastable in middle**
  - Fall out when perturbed

Ball on Ramp Analogy



Example SRAM Configuration (16 x 8)



Dynamic RAM (DRAM)

Slower than SRAM
- access time ~60 nsec

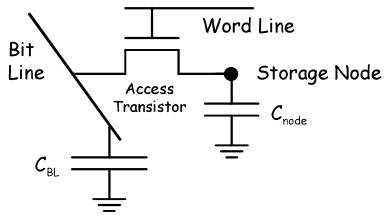
Nonpersistant
- every row must be accessed every ~1 ms (refreshed)

Cheaper than SRAM
- ~$1.50 / MByte
- 1 transistor/bit

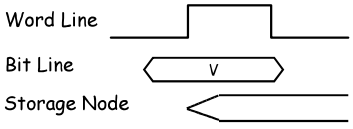
Fragile
- electrical noise, light, radiation

Workhorse memory technology

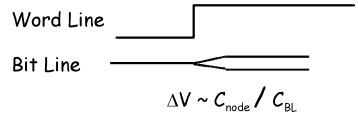
Anatomy of a DRAM Cell



Writing



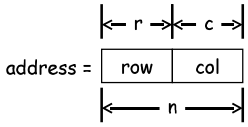
Reading



Addressing Arrays with Bits

Array Size
- R rows, R = 2^r
- C columns, C = 2^c
- N = R * C bits of memory

Addressing
- Addresses are n bits, where N = 2^n
- row(address) = address / C
  - leftmost r bits of address
- col(address) = address % C
  - rightmost bits of address

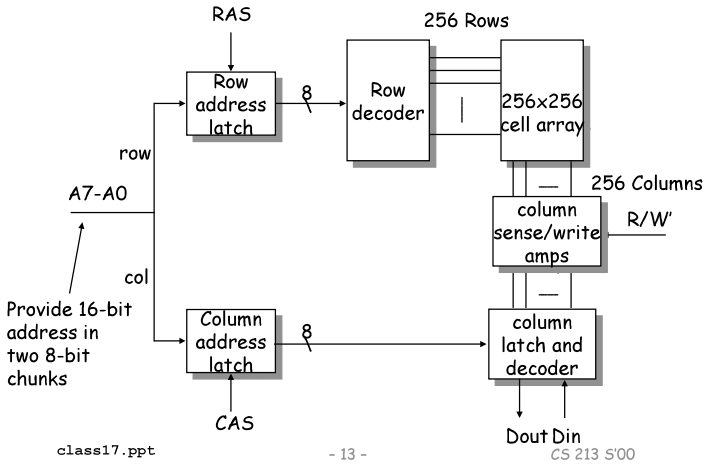


Example
- R = 2
- C = 4
- address = 6

	0	1	2	3
0	000	001	010	011
1	100	101	110	111

row 1 col 2

Example 2-Level Decode DRAM (64Kx1)



DRAM Operation

Row Address (~50ns)
- Set Row address on address lines & strobe RAS
- Entire row read & stored in column latches
- Contents of row of memory cells destroyed

Column Address (~10ns)
- Set Column address on address lines & strobe CAS
- Access selected bit
  - READ: transfer from selected column latch to Dout
  - WRITE: Set selected column latch to Din

Rewrite (~30ns)
- Write back entire row

Observations About DRAMs

Timing
- Access time (= 60ns) < cycle time (= 90ns)
- Need to rewrite row

Must Refresh Periodically
- Perform complete memory cycle for each row
- Approximately once every 1ms
- Sqrt(n) cycles
- Handled in background by memory controller

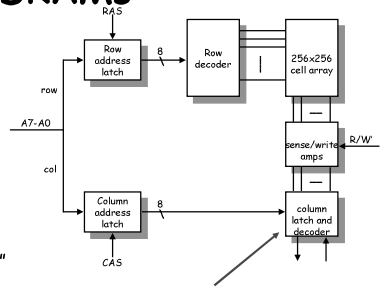
Inefficient Way to Get a Single Bit
- Effectively read entire row of Sqrt(n) bits

Enhanced Performance DRAMs

Conventional Access
- Row + Col
- RAS CAS RAS CAS ...

Page Mode
- Row + Series of columns
- RAS CAS CAS CAS ...
- Gives successive bits

Other Acronyms
- EDORAM
  - "Extended data output"
- SDRAM
  - "Synchronous DRAM"



Typical Performance

row access time	col access time	cycle time	page mode cycle time
50ns	10ns	90ns	25ns

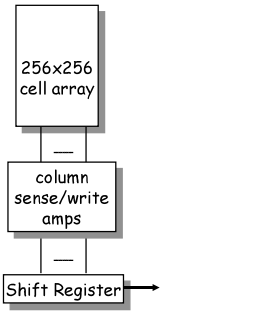
Video RAM

Performance Enhanced for Video / Graphics Operations
- Frame buffer to hold graphics image

Writing
- Random access of bits
- Also supports rectangle fill operations
  - Set all bits in region to 0 or 1

Reading
- Load entire row into shift register
- Shift out at video rates

Performance Example
- 1200 X 1800 pixels / frame
- 24 bits / pixel
- 60 frames / second
- 2.8 GBits / second



DRAM Driving Forces

Capacity

- 4X per generation
  - Square array of cells
- Typical scaling
  - Lithography dimensions 0.7X
    - » Areal density 2X
  - Cell function packing 1.5X
  - Chip area 1.33X
- Scaling challenge
  - Typically C_{node} / C_{BL} = 0.1–0.2
  - Must keep C_{node} high as shrink cell size

Retention Time

- Typically 16–256 ms
- Want higher for low-power applications

DRAM Storage Capacitor

Planar Capacitor

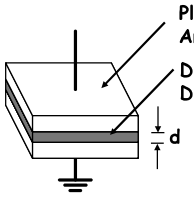
- Up to 1Mb
- C decreases linearly with feature size

Trench Capacitor

- 4–256 Mb
- Lining of hole in substrate

Stacked Cell

- > 1Gb
- On top of substrate
- Use high ε dielectric

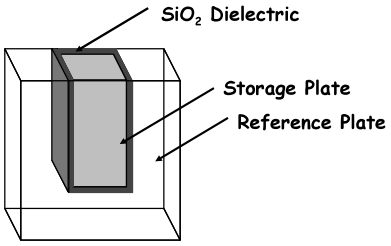


$C = \varepsilon A/d$

Trench Capacitor

Process

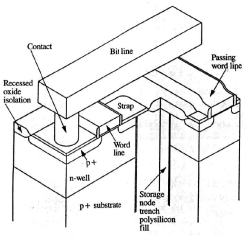
- Etch deep hole in substrate
  - Becomes reference plate
- Grow oxide on walls
  - Dielectric
- Fill with polysilicon plug
  - Tied to storage node



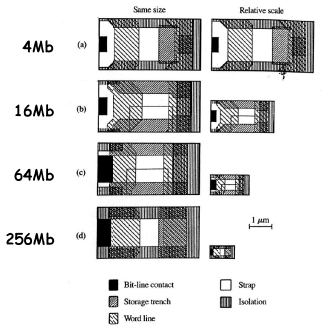
IBM DRAM Evolution

- IBM J. R&D, Jan/Mar '95
- Evolution from 4 – 256 Mb
- 256 Mb uses cell with area 0.6 μm²

4 Mb Cell Structure



Cell Layouts



Mitsubishi Stacked Cell DRAM

- IEDM '95
- Claim suitable for 1 – 4 Gb

Technology

- 0.14 μm process
  - Synchrotron X-ray source
- 8 nm gate oxide
- 0.29 μm² cell

Storage Capacitor

- Fabricated on top of everything else
- Rubidium electrodes
- High dielectric insulator
  - 50X higher than SiO_2
  - 25 nm thick
- Cell capacitance 25 femtofarads

Cross Section of 2 Cells

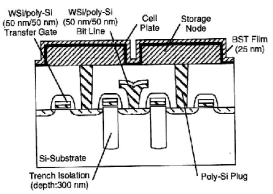


Fig. 2 Schematic cross-sectional view of DRAM memory cells with Ru/BST/Ru stacked capacitors.

Mitsubishi DRAM Pictures

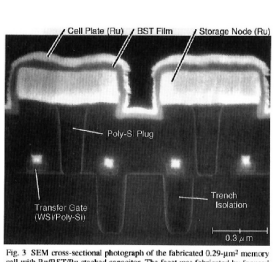


Fig. 3 SEM cross-sectional photograph of the fabricated 0.29-μm² memory cell with Ru/BST/Ru stacked capacitor. The facet was fabricated by focused ion beam etching.

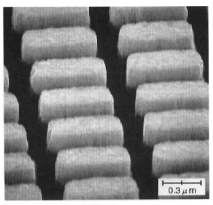


Fig. 8 SEM photograph of a Ru-metal storage node array with a projection a height of 0.2 μm.

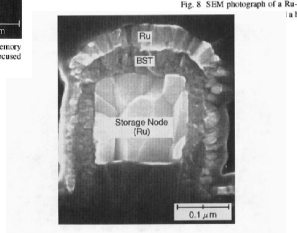
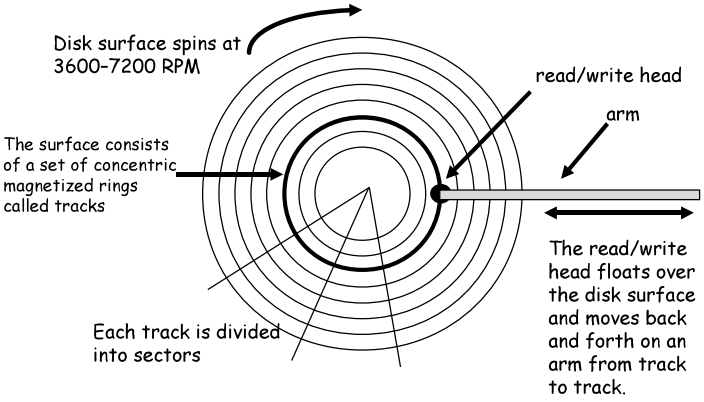


Fig. 10 SEM cross-sectional view of a Ru/BST/Ru capacitor cell. The facet shown is a cleaved facet.

Magnetic Disks



Disk Capacity

Parameter	18GB Example
• Number Platters	12
• Surfaces / Platter	2
• Number of tracks	6962
• Number sectors / track	213
• Bytes / sector	512
Total Bytes	**18,221,948,928**

Disk Operation

Operation
- Read or write complete sector

Seek
- Position head over proper track
- Typically 6-9ms

Rotational Latency
- Wait until desired sector passes under head
- Worst case: complete rotation
  10,025 RPM ⇒ 6 ms

Read or Write Bits
- Transfer rate depends on # bits per track and rotational speed
- E.g., 213 * 512 bytes @10,025RPM = 18 MB/sec.
- Modern disks have external transfer rates of up to 80 MB/sec
  - DRAM caches on disk help sustain these higher rates

Disk Performance

Getting First Byte
- Seek + Rotational latency = 7,000 – 19,000 µsec

Getting Successive Bytes
- ~ 0.06 µsec each
  - *roughly 100,000 times faster than getting the first byte!*

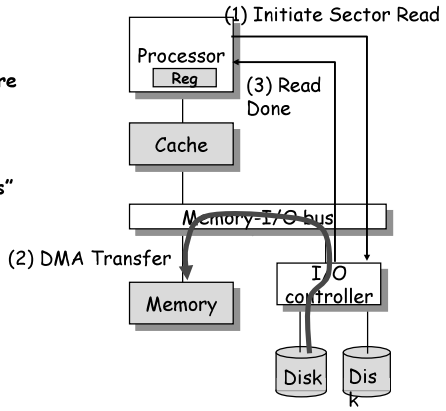
Optimizing Performance:
- Large block transfers are more efficient
- Try to do other things while waiting for first byte
  - switch context to other computing task
  - processor is interrupted when transfer completes

Disk / System Interface

1. Processor Signals Controller
- Read sector X and store starting at memory address Y

2. Read Occurs
- "Direct Memory Access" (DMA) transfer
- Under control of I/O controller

3. I/O Controller Signals Completion
- Interrupts processor
- Can resume suspended process



Magnetic Disk Technology

Seagate ST-12550N Barracuda 2 Disk

• Linear density	52,187.	bits per inch (BPI)
– Bit spacing	0.5	microns
• Track density	3,047.	tracks per inch (TPI)
– Track spacing	8.3	microns
• Total tracks	2,707.	tracks
• Rotational Speed	7200.	RPM
• Avg Linear Speed	86.4	kilometers / hour
• Head Floating Height	0.13	microns

Analogy:
- put the Sears Tower on its side
- fly it around the world, *2.5cm* above the ground
- each complete orbit of the earth takes *8 seconds*

CD Read Only Memory (CDROM)

Basis

- **Optical recording technology developed for audio CDs**
  - 74 minutes playing time
  - 44,100 samples / second
  - 2 X 16-bits / sample (Stereo)
  - ⇒ Raw bit rate = 172 KB / second
- **Add extra 288 bytes of error correction for every 2048 bytes of data**
  - Cannot tolerate any errors in digital data, whereas OK for audio

Bit Rate

- **172 * 2048 / (288 + 2048) = 150 KB / second**
  - For 1X CDROM
  - N X CDROM gives bit rate of N * 150
  - E.g., 12X CDROM gives 1.76 MB / second

Capacity

- **74 Minutes * 150 KB / second * 60 seconds / minute = 650 MB**

Storage Trends

SRAM

metric	1980	1985	1990	1995	2000	***2000:1980***
$/MB	19,200	2,900	320	256	100	***190***
access (ns)	300	150	35	15	2	***100***

DRAM

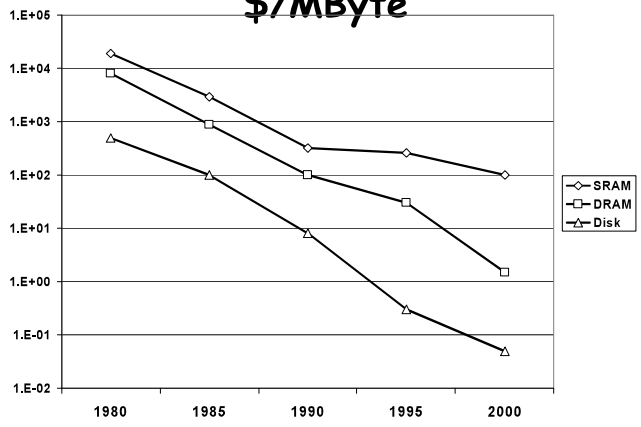
metric	1980	1985	1990	1995	2000	***2000:1980***
$/MB	8,000	880	100	30	1.5	***5,300***
access (ns)	375	200	100	70	60	***6***
typical size(MB)	0.064	0.256	4	16	64	***1,000***

Disk

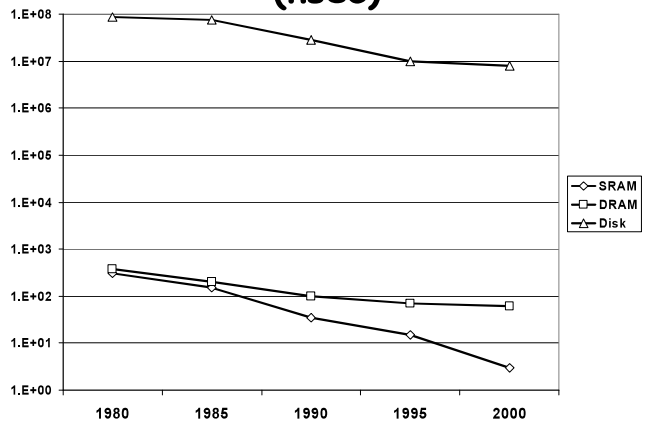
metric	1980	1985	1990	1995	2000	***2000:1980***
$/MB	500	100	8	0.30	0.05	***10,000***
access (ms)	87	75	28	10	8	***11***
typical size(MB)	1	10	160	1,000	9,000	***9,000***

(Culled from back issues of Byte and PC Magazine)

Storage Price: $/MByte



Storage Access Times (nsec)



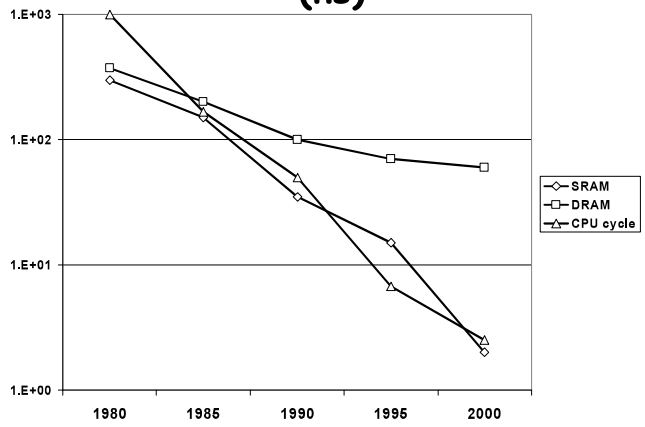
Processor clock rates

Processors

metric	1980	1985	1990	1995	2000	***2000:1980***
typical clock(MHz)	1	6	20	150	600	**600**
processor	8080	286	386	Pentium	P-III	

culled from back issues of Byte and PC Magazine

The CPU vs. DRAM Latency Gap (ns)



Memory Technology Summary

Cost and Density Improving at Enormous Rates

Speed Lagging Processor Performance

Memory Hierarchies Help Narrow the Gap:

- Small fast SRAMS (cache) at upper levels
- Large slow DRAMS (main memory) at lower levels
- Incredibly large & slow disks to back it all up

Locality of Reference Makes It All Work

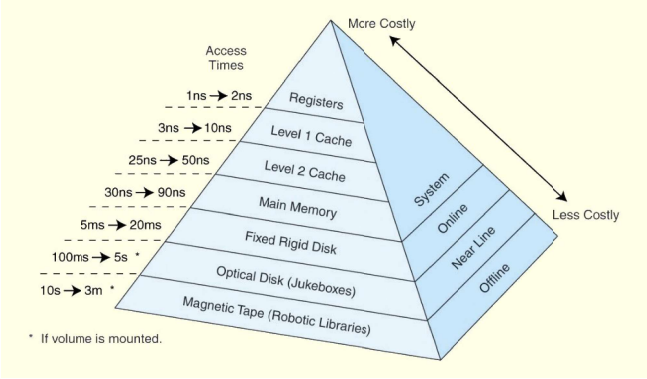
- Keep most frequently accessed data in fastest memory

UNIT IV

CACHE MEMORY – MAPPING FUNCTIONS

The Memory Hierarchy

- This storage organization can be thought of as a pyramid:



Hierarchy List

- Registers
- L1 Cache
- L2 Cache
- Main memory
- Disk cache
- Disk
- Optical
- Tape

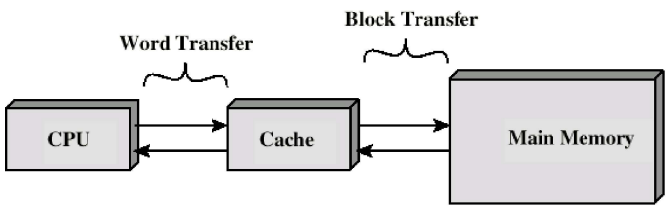
- As one goes down the hierarchy
  - Decreasing cost per bit
  - Increasing capacity
  - Increasing access time
  - Decreasing frequency of access of the memory by the processor – locality of reference

Locality of Reference

- Temporal Locality
  - Programs tend to reference the same memory locations at a future point in time
  - Due to loops and iteration, programs spending a lot of time in one section of code
- Spatial Locality
  - Programs tend to reference memory locations that are near other recently-referenced memory locations
  - Due to the way contiguous memory is referenced, e.g. an array or the instructions that make up a program
  - Sequential Locality
    - Instructions tend to be accessed sequentially
- Locality of reference does not always hold, but it usually holds

Cache

- Small amount of fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or module
  - An entire blocks of data is copied from memory to the cache because the principle of locality tells us that once a byte is accessed, it is likely that a nearby data element will be needed soon.



Cache operation - overview

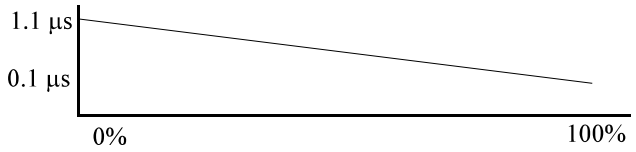
- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU
- Cache includes tags to identify which block of main memory is in each cache slot

Cache Definitions

- This leads us to some definitions.
  - A *hit* is when data is found at a given memory level.
  - A *miss* is when it is not found.
  - The *hit rate* is the percentage of time data is found at a given memory level.
  - The *miss rate* is the percentage of time it is not.
  - Miss rate = 1 - hit rate.
  - The *hit time* is the time required to access data at a given memory level.
  - The *miss penalty* is the time required to process a miss, including the time that it takes to replace a block of memory plus the time it takes to deliver the data to the processor.

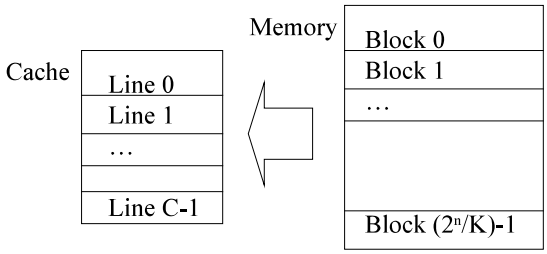
Cache Example

- Consider a Level 1 cache capable of holding 1000 words with a 0.1 μs access time. Level 2 is memory with a 1 μs access time.
- If 95% of memory access is in the cache:
  - T=(0.95)*(0.1 μs) + (0.05)*(0.1+1 μs) = 0.15 μs
- If 5% of memory access is in the cache:
  - T=(0.05)*(0.1 μs) + (0.95)*(0.1+1 μs) = 1.05 μs
- Want as many cache hits as possible!



Cache Design

- If memory contains 2^n addressable words
  - Memory can be broken up into blocks with K words per block. Number of blocks = $2^n / K$
  - Cache consists of C **lines** or **slots**, each consisting of K words
  - C << M
  - How to map blocks of memory to lines in the cache?



Cache Design

- Size
- Mapping Function
- Replacement Algorithm
- Write Policy
- Block Size
- Number of Caches

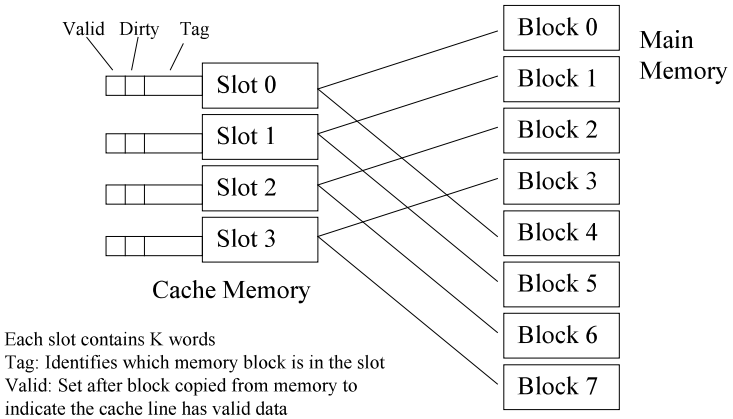
Direct Mapping

- Simplest mapping technique - each block of main memory maps to only one cache line
  - i.e. if a block is in cache, it must be in one specific place
- Formula to map a memory block to a cache line:
  - i = j mod c
    - i=Cache Line Number
    - j=Main Memory Block Number
    - c=Number of Lines in Cache
  - i.e. we divide the memory block by the number of cache lines and the remainder is the cache line address

Direct Mapping with C=4

- Shrinking our example to a cache line size of 4 slots (each slot/line/block still contains 4 words):
  - Cache Line — Memory Block Held
    - 0 — 0, 4, 8, …
    - 1 — 1, 5, 9, …
    - 2 — 2, 6, 10, …
    - 3 — 3, 7, 11, …
  - In general:
    - 0 — 0, C, 2C, 3C, …
    - 1 — 1, C+1, 2C+1, 3C+1, …
    - 2 — 2, C+2, 2C+2, 3C+2, …
    - 3 — 3, C+3, 2C+3, 3C+3, …

Direct Mapping with C=4



Each slot contains K words
Tag: Identifies which memory block is in the slot
Valid: Set after block copied from memory to indicate the cache line has valid data

Direct Mapping Address Structure

- There is an easy way to compute
  i = j mod c
  based upon the bits in the address we are trying to access
- Address is in three parts
  - Least Significant `w` bits identify unique word within a cache line
    - `w` must be enough bits to address a specific word in a cache line; e.g. if 4 words per cache line, then we need 2 bits for `w`
  - Next Significant `s` bits specify which line this address maps into
    - `s` must be enough bits to address a specific line in the cache; e.g. if 16 cache lines, then we need 4 bits for `s`
  - Remaining `t` bits used as a tag to identify the memory block

Tag t	Line or Slot s	Word w
2	4	2

8 bit address

Direct Mapping Address 64K Cache Example

cache only | memory address

V	D	Tag t	Line or Slot s	Word w
1	1	8	14	2

- Given a 24 bit address (to access 16Mb)
- 2 bit word identifier (4 byte block)
- Need 14 bits to address the cache slot/line
- Leaves 8 bits left for tag (=22-14)

- No two blocks in the same line have the same Tag field
- Check contents of cache by finding line and checking Tag
- Also need a Valid bit and a Dirty bit
  - Valid – Indicates if the slot holds a block belonging to the program being executed
  - Dirty – Indicates if a block has been modified while in the cache. Will need to be written back to memory before slot is reused for another block

Direct Mapping Example, 64K Cache

Cache Memory

	Addr	Tag	W0	W1	W2	W3
0		00	F1	F2	F3	F4
1		1B	11	12	13	14
2						
3						
4						
5						
..						
..						
2^{14}-1						

Main Memory

	Addr (hex)	Data
Line 0	000000	F1
	000001	F2
	000002	F3
	000003	F4
Line 1	000004	AB
	…	
Line 1	1B0004	11
	1B0005	12
	1B0006	13
	1B0007	14

1B0007 = 0001 1011 0000 0000 0000 0111
Word = 11, Line = 0000 0000 0000 01, Tag= 0001 1011

Cache Example

- The website for the textbook includes a link to CAMERA, a cache simulator
- Example for direct-mapped cache

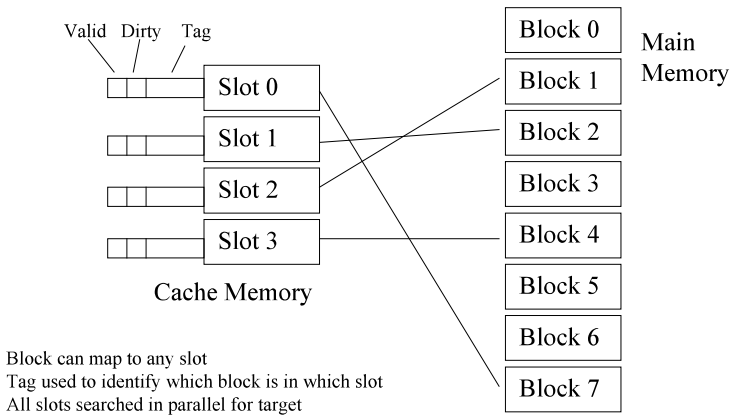
Direct Mapping pros & cons

- Simple
- Inexpensive
- Fixed location for given block
  - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high – condition called **thrashing**

Fully Associative Mapping

- A fully associative mapping scheme can overcome the problems of the direct mapping scheme
  - A main memory block can load into any line of cache
  - Memory address is interpreted as tag and word
  - Tag uniquely identifies block of memory
  - Every line's tag is examined for a match
  - Also need a Dirty and Valid bit
- But Cache searching gets expensive!
  - Ideally need circuitry that can simultaneously examine all tags for a match
  - Lots of circuitry needed, high cost
- Need replacement policies now that anything can get thrown out of the cache (will look at this shortly)

Associative Mapping Example



Block can map to any slot
Tag used to identify which block is in which slot
All slots searched in parallel for target

Associative Mapping 64K Cache Example

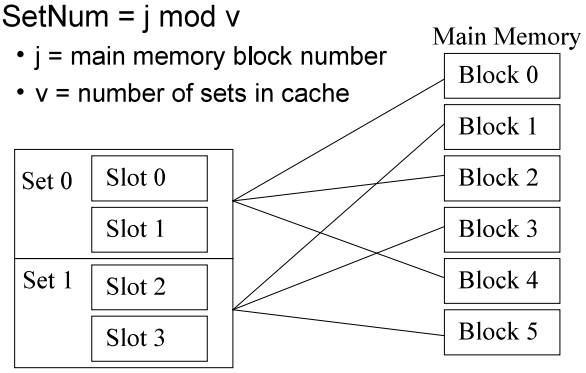
- 22 bit tag stored with each slot in the cache – no more bits for the slot line number needed since all tags searched in parallel
- Compare tag field of a target memory address with tag entry in cache to check for hit
- Least significant 2 bits of address identify which word is required from the block, e.g.:
  - Address: FFFFFC = 1111 1111 1111 1111 1111 1100
    - Tag: Left 22 bits, truncate on left:
      - 11 1111 1111 1111 1111 1111
      - 3FFFFF
  - Address: 16339C = 0001 0110 0011 0011 1001 1100
    - Tag: Left 22 bits, truncate on left:
      - 00 0101 1000 1100 1110 0111
      - 058CE7

Set Associative Mapping

- Compromise between fully-associative and direct-mapped cache
  - Cache is divided into a number of sets
  - Each set contains a number of lines
  - A given block maps to any line in a specific set
    - Use direct-mapping to determine which set in the cache corresponds to a set in memory
    - Memory block could then be in any line of that set
  - e.g. 2 lines per set
    - 2 way associative mapping
    - A given block can be in either of 2 lines in a specific set
  - e.g. K lines per set
    - K way associative mapping
    - A given block can be in one of K lines in a specific set
    - Much easier to simultaneously search one set than all lines

Set Associative Mapping

- To compute cache set number:
  - SetNum = j mod v
    - j = main memory block number
    - v = number of sets in cache

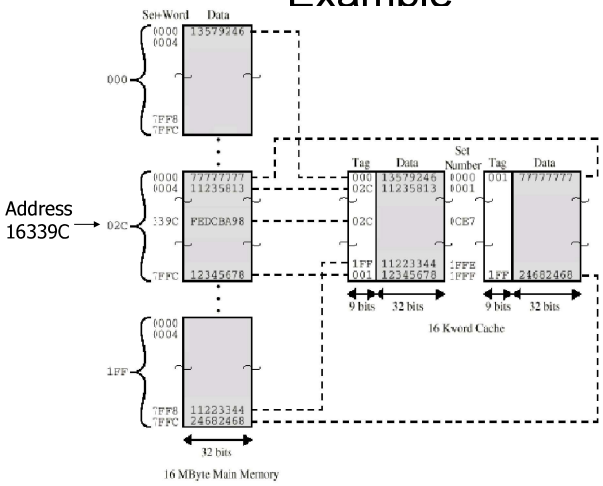


Set Associative Mapping 64K Cache Example

Tag 9 bit	Set 13 bit	Word 2 bit

- E.g. Given our 64Kb cache, with a line size of 4 bytes, we have 16384 lines. Say that we decide to create 8192 sets, where each set contains 2 lines. Then we need 13 bits to identify a set (2^{13}=8192)
- Use set field to determine cache set to look in
- Compare tag field of all slots in the set to see if we have a hit, e.g.:
  - Address = 16339C = 0001 0110 0011 0011 1001 1100
    - Tag = 0 0010 1100 = 02C
    - Set = 0 1100 1110 0111 = 0CE7
    - Word = 00 = 0
  - Address = 008004 = 0000 0000 1000 0000 0000 0100
    - Tag = 0 0000 0001 = 001
    - Set = 0 0000 0000 0001 = 0001
    - Word = 00 = 0

Two Way Set Associative Example



K-Way Set Associative

- Two-way set associative gives much better performance than direct mapping
  - Just one extra slot avoids the thrashing problem
- Four-way set associative gives only slightly better performance over two-way
- Further increases in the size of the set has little effect other than increased cost of the hardware!

Replacement Policy (1)

- The replacement policy is the technique we use to determine which line in the cache should be thrown out when we want to put a new block in from memory
- Direct mapping
  - No choice
  - Each block only maps to one line
  - Replace that line

Replacement Algorithms (2) Associative & Set Associative

- Algorithm must be implemented in hardware (speed)
- Least Recently used (LRU)
  - e.g. in 2 way set associative, which of the 2 block is LRU?
    - For each slot, have an extra bit, USE. Set to 1 when accessed, set all others to 0.
  - For more than 2-way set associative, need a time stamp for each slot - expensive
- First in first out (FIFO)
  - Replace block that has been in cache longest
  - Easy to implement as a circular buffer
- Least frequently used
  - Replace block which has had fewest hits
  - Need a counter to sum number of hits
- Random
  - Almost as good as LFU and simple to implement

Write Policy

- So far we've only discussed reading memory, we may also write back to memory
- If a memory write only updates the cache, then the cache is now inconsistent with main memory
  - Could cause problems
  - Reading in another memory block that maps to the same cache line
  - I/O device or another processor might directly try to read/write to memory

Write through

- Simplest technique to handle the cache inconsistency problem - All writes go to main memory as well as cache.
- Multiple CPUs must monitor main memory traffic (snooping) to keep local cache local to its CPU up to date in case another CPU also has a copy of a shared memory location in its cache
- Simple but Lots of traffic
- Slows down writes

Write Back

- Updates initially made in cache only
  - Dirty bit is set when we write to the cache, this indicates the cache is now inconsistent with main memory
- Dirty bit for cache slot is cleared when update occurs
- If cache line is to be replaced, write the existing cache line to main memory if dirty bit is set before loading the new memory block

Cache Performance

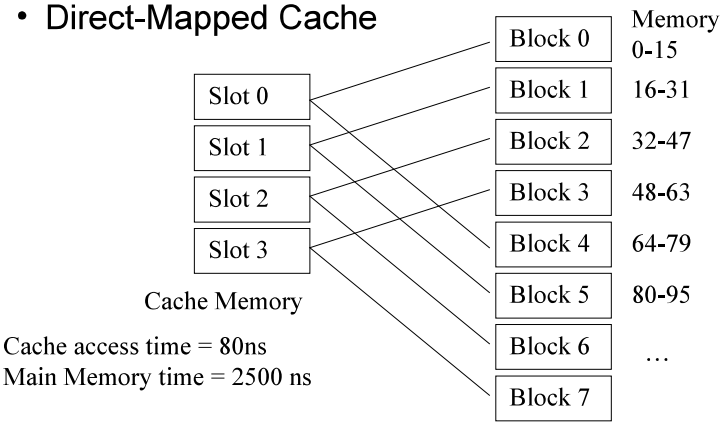
- Two measures that characterize the performance of a cache are the **hit ratio** and the **effective access time**

$$\text{Hit Ratio} = \frac{\text{(Num times referenced words are in cache)}}{\text{(Total number of memory accesses)}}$$

$$\text{Eff. Access Time} = \frac{\text{(\# hits)(TimePerHit)+(\# misses) (TimePerMiss)}}{\text{(Total number of memory accesses)}}$$

Cache Performance Example

- Direct-Mapped Cache

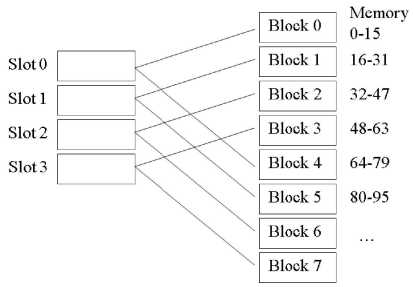


Cache access time = 80ns
Main Memory time = 2500 ns

Cache Performance Example

- Sample program executes from memory location 48-95 once. Then it executes from 15-31 in a loop ten times before exiting.

Event	Location	Time	Comment
1 miss	48	2500ns	Memory block 3 to cache slot 3
15 hits	49-63	80ns×15=1200ns	
1 miss	64	2500ns	Memory block 4 to cache slot 0
15 hits	65-79	80ns×15=1200ns	
1 miss	80	2500ns	Memory block 5 to cache slot 1
15 hits	81-95	80ns×15=1200ns	
1 miss	15	2500ns	Memory block 0 to cache slot 0
1 miss	16	2500ns	Memory block 1 to cache slot 1
15 hits	17-31	80ns×15=1200ns	
9 hits	15	80ns×9=720ns	Last nine iterations of loop
144 hits	16-31	80ns×144=12,240ns	Last nine iterations of loop
Total hits = 213		Total misses = 5	



Once: 48-95

Ten Times: 15-31

Event	Location	Time	Comment
1 miss	48	2500ns	Memory block 3 to cache slot 3
15 hits	49-63	80ns×15=1200ns	
1 miss	64	2500ns	Memory block 4 to cache slot 0
15 hits	65-79	80ns×15=1200ns	
1 miss	80	2500ns	Memory block 5 to cache slot 1
15 hits	81-95	80ns×15=1200ns	
1 miss	15	2500ns	Memory block 0 to cache slot 0
1 miss	16	2500ns	Memory block 1 to cache slot 1
15 hits	17-31	80ns×15=1200ns	
9 hits	15	80ns×9=720ns	Last nine iterations of loop
144 hits	16-31	80ns×144=12,240ns	Last nine iterations of loop
Total hits = 213		Total misses = 5	

Cache Performance Example

- Hit Ratio: 213 / 218 = 97.7%
- Effective Access Time: ((213)*(80ns)+(5)(2500ns)) / 218 = 136 ns

- Although the hit ratio is high, the effective access time in this example is 75% longer than the cache access time due to the large amount of time spent during a cache miss
- What sequence of main memory block accesses would result in much worse performance?

Parameters of Cache memory

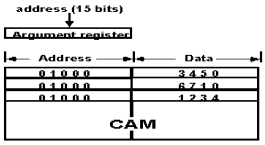
- Cache Hit
  - A referenced item is found in the cache by the processor
- Cache Miss
  - A referenced item is not present in the cache
- Hit ratio
  - Ratio of number of hits to total number of references => number of hits/(number of hits + number of Miss)
- Miss penalty
  - Additional cycles required to serve the miss

Parameters of Cache Memory

- Time required for the cache miss depends on both the latency and bandwidth
- Latency – time to retrieve the first word of the block
- Bandwidth – time to retrieve the rest of this block

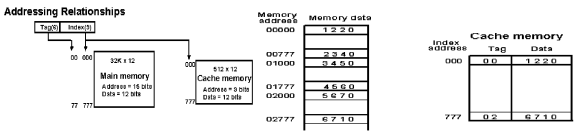
Associative Mapping

- Fastest, most flexible but very expensive
- Any block location in cache can store any block in memory
- Stores both the address and the content of the memory word
- CPU address of 15 bits is placed in the argument register and the associative memory is searched for a matching address
- If found data is read and sent to the CPU else main memory is accessed.
- CAM – content addressable memory



Direct Mapping

- N-bit CPU memory address – k bits index field and (n-k) bits tag field
- Index bits are used to access the cache
- Each word in cache consists of data word and its associated tag



On memory request, index field is used to access the cache. Tag field of CPU address is compared with the tag in the word read in the cache

If match then there is hit

Else miss – word is read from the memory

It is then stored in the cache together with the new tag replacing the previous value

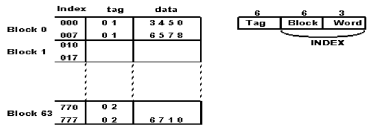
Disadvantage: hit ratio drops if 2 or more words with same index but different tags are accessed repeatedly.

When memory is divided into blocks of words, index field – block field and word field

Ex: 512 words cache – 64 blocks of 8words each – block field (6bits) and words field (3bits)

- Tags within the block are same.

Direct Mapping with block size of 8 words

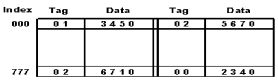


- When a miss occurs, entire block is transferred from main memory to cache.
- It is time consuming but improves hit ratio because of the sequential nature of programs.

Set-Associative Mapping

- Each word of cache can store 2 or more words of memory under the same index address

Set Associative Mapping Cache with set size of two



The comparison logic is done by an associative search of the tags in the set similar to an associative memory search, thus the name "Set-Associative"

- Hit ratio increases as the set size increases but more complex comparison logic is required when number of bits in words of cache increases
- When a miss occurs and set is full, one of tag-data items are replaced using block replacement policy

Problems

- A set associative cache consists of 64 lines or slots, divided into four line sets. Main memory consists 4k blocks of 128 words each. Show the format of main memory addresses.

The cache is divided into 16 sets of 4 lines each. Therefore, 4 bits are needed to identify the set number. Main memory consists of $4K = 2^{12}$ blocks. Therefore, the set plus tag lengths must be 12 bits and therefore the tag length is 8 bits. Each block contains 128 words. Therefore, 7 bits are needed to specify the word.

	TAG	SET	WORD
Main memory address =	8	4	7

Problem 2

- A two-way set associative cache has lines of 16 bytes and a total size of 8k bytes. The 64-Mbyte main memory is byte addressable. Show the format of main memory addresses.

There are a total of 8 kbytes / 16 bytes = 512 lines in the cache. Thus the cache consists of 256 sets of 2 lines each. Therefore 8 bits are needed to identify the set number. For the 64-Mbyte main memory, a 26-bit address is needed. Main memory consists of 64-Mbyte / 16 bytes = 2^{22} blocks. Therefore, the set plus tag lengths must be 22 bits, so the tag length is 14 bits and the word field length is 4 bits.

	TAG	SET	WORD
Main memory address =	14	8	4

Block Replacement

- **Least Recently Used: (LRU)**

  Replace that block in the set that has been in the cache longest with no reference to it.
- **First Come First Out: (FIFO)**

  Replace that block in the set that has been in the cache longest.
- **Least Frequently Used: (LFU)**

  Replace that block in the set that has experienced the fewest references

Update Policies- Write Through

- Update main memory with every memory write operation
- Cache memory is updated in parallel if it contains the word at specified address.
- Advantage: main memory always contains the same data as the cache
- It is important during DMA transfers to ensure the data in main memory is valid
- Disadvantage: slow due to memory access time

Write Back

- Only cache is updated during write operation and marked by flag. When the word is removed from the cache, it is copied into main memory
- Memory is not up-to-date, i.e., the same item in cache and memory may have different value

Update policies

- **Write-Around**
  - **correspond to items not currently in the cache (i.e. write misses) the item could be updated in main memory only without affecting the cache.**
- **Write-Allocate**
  - **update the item in main memory and bring the block containing the updated item into the cache.**

Performance analysis

- Look through: The cache is checked first for a hit, and if a miss occurs then the access to main memory is started.
- Look aside: access to main memory in parallel with the cache lookup;

•Look through
TA = TC + (1-h)*TM
TC is the average cache access time
TM is the average access time
(Mean memory access time)

•Look aside
TA = h*TC + (1-h)*TM

$$\text{•hit ratio } h = \frac{\text{number of references found in the cache}}{\text{total number of memory references}}$$

• **Miss Ratio m=(1-h)**

Example: assume that a computer system employs a cache with an access time of 20ns and a main memory with a cycle time of 200ns. Suppose that the hit ratio for reads is 90%,

a) what would be the average access time for reads if the cache is a "look-through" cache?

The average read access time (**TA) = TC + (1-h)*TM**
20ns + 0.10*200ns = 40ns

b) what would be the average access time for reads if the cache is a "look-Aside" cache?

The average read access time in this case (**TA)**
= h*TC + (1-h)*TM= 0.9*20ns + 0.10*200ns = 38ns

Problem

- Consider a memory system with T_c = 100ns and T_m = 1200ns. If the effective access time is 10% greater than the cache access time, what is the hit ratio H in look-through cache?

$\Rightarrow T_A = T_C + (1-h)*T_M$

$\Rightarrow 1.1\ T_c = T_c + (1-h)*T_M$

$\Rightarrow 0.1T_C = (1-h)*\ T_M$

$\Rightarrow 0.1 * 100 = (1-h) *1200$

$\Rightarrow 1-h = 10/1200$

$\Rightarrow h = 1190/1200$

Sources of cache Misses

- Compulsory Misses
- Capacity Misses
- Cold Misses
- Conflict Misses

Sources of Cache Misses

- **Compulsory Misses:** These are misses that are caused by the cache being empty initially.
- **Cold Misses :** The very first access to a block will result in a miss because the block is not brought into cache until it is referenced.
- **Capacity Misses :** If the cache cannot contain all the blocks needed during the execution of a program, capacity misses will occur due to blocks being discarded and later retrieved.
- **Conflict Misses:** If the cache mapping is such that multiple blocks are mapped to the same cache entry

Cache organization

- Split cache
  - Separate caches for instructions and data
  - I-cache (Instruction) – mostly accessed sequentially
  - D-cache (data) – mostly random access
- Unified cache
  - Same cache for instruction and data
- Higher hit rate for unified cache as it balances between instruction and data
- Split caches eliminate contention for cache between the instruction processor and the execution unit – used for pipelining processes

Multilevel caches

- The penalty for a cache miss is the extra time that it takes to obtain the requested item from central memory.
- One way in which this penalty can be reduced is to provide another cache, the secondary cache, which is accessed in response to a miss in the primary cache.
- The primary cache is referred to as the L1 (level 1) cache and the secondary cache is called the L2 (level 2) cache.
- Most high-performance microprocessors include an L2 cache which is often located off-chip, whereas the L1 cache is located on the same chip as the CPU.
- With a two-level cache, central memory has to be accessed only if a miss occurs in both caches.

Example:

- A computer system employs a write-back cache with a 70% hit ratio for writes. The cache operates in look-aside mode and has a 90% read hit ratio. Reads account for 80% of all memory references and writes account for 20%. If the main memory cycle time is 200ns and the cache access time is 20ns, what would be the average access time for all references (reads as well as writes)?

The average access time for reads = 0.9*20ns + 0.1*200ns = 38ns.

The average write time = 0.7*20ns + 0.3*200ns = 74ns

Hence the overall average access time for combined reads and writes is

0.8*38ns + 0.2*74ns = 45.2ns

References

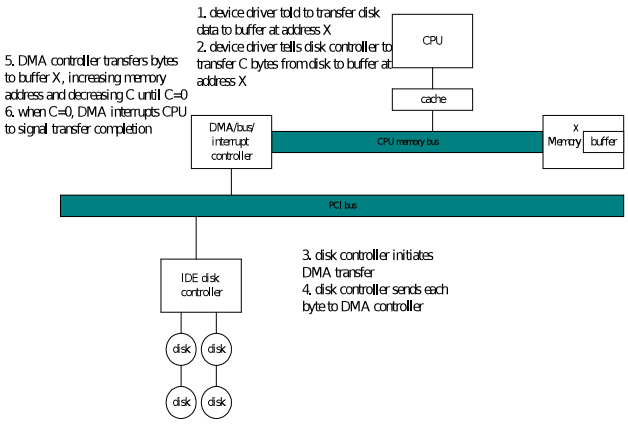
- J. L. Hennessy & D.A. Patterson, Computer architecture: A quantitative approach, Fourth Edition, Morgan Kaufman, 2004.

Introduction of Direct Memory Access (DMA)

Why is DMA?

- It is wasteful to feed data into a controller register 1 bytes at a time. (PIO)
- The DMA unit is word.
- In the high loading environment, a system with DMA has better improvement.

DMA transfer



DMA Progress

- To initiate a DMA transfer, the host writes a DMA command into memory:
  - A pointer to the source of a transfer
  - A count of the number of bytes to be transferred
  - ……..
- The CPU writes the address of the DMA command block to the DMA controller.

DMA Progress (cont.)

- The DMA controller proceeds to operate the memory bus directly without CPU help.
- Handshaking exists between DMA controller and device controller.
- When the entire transfer is finished, the DMA controller will interrupts the CPU.

Handshaking

- DMA-request and DMA_acknowledge
  - When a word of data is available, the device controller places a signal on the *DMA-request* wire.
  - The signal causes the DMA controller to seize the memory bus,
    - To place the desired address on the memory-address wire
    - To place a signal on the DMA-acknowledge wire

Handshaking (cont.)

- When the device controller receives the DMA-acknowledge signal,
  - it transfer the word of data to memory
  - and remove the DMA_request signal.

Input/Output Organization

Chapter 19

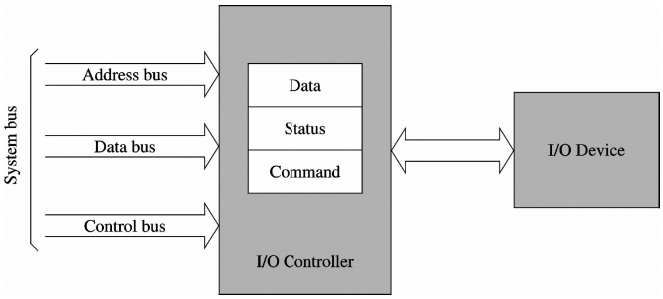
Outline

- Introduction
- Accessing I/O devices
- An example I/O device
  - Keyboard
- I/O data transfer
  - Programmed I/O
  - DMA
- Error detection and correction
  - Parity encoding
  - Error correction
  - CRC
- External interface
  - Serial transmission
  - Parallel interface
- USB
  - Motivation
  - USB architecture
  - USB transactions
- IEEE 1394
  - Advantages
  - Transactions
  - Bus arbitration
  - Configuration

Introduction

- I/O devices serve two main purposes
  - To communicate with outside world
  - To store data
- I/O controller acts as an interface between the systems bus and I/O device
  - Relieves the processor of low-level details
  - Takes care of electrical interface
- I/O controllers have three types of registers
  - Data
  - Command
  - Status

Introduction (cont'd)



Introduction (cont'd)

- To communicate with an I/O device, we need
  - Access to various registers (data, status,…)
    - » This access depends on I/O mapping
      - – Two basic ways
        - ✈Memory-mapped I/O
        - ✈Isolated I/O
  - A protocol to communicate (to send data, …)
    - » Three types
      - – Programmed I/O
      - – Direct memory access (DMA)
      - – Interrupt-driven I/O

Accessing I/O Devices

- I/O address mapping
  - Memory-mapped I/O
    - » Reading and writing are similar to memory read/write
    - » Uses same memory read and write signals
    - » Most processors use this I/O mapping
  - Isolated I/O
    - » Separate I/O address space
    - » Separate I/O read and write signals are needed
    - » Pentium supports isolated I/O
      - – 64 KB address space
        - ✈Can be any combination of 8-, 16- and 32-bit I/O ports
      - – Also supports memory-mapped I/O

Accessing I/O Devices (cont'd)

- Accessing I/O ports in Pentium
  - Register I/O instructions

    `in accumulator, port8` ; direct format
    - – Useful to access first 256 ports

    `in accumulator,DX` ; indirect format
    - – DX gives the port address
  - Block I/O instructions
    - » `ins` and `outs`
      - – Both take no operands---as in string instructions
    - » `ins`: port address in DX, memory address in ES:(E)DI
    - » `outs`: port address in DX, memory address in ES:(E)SI
    - » We can use `rep` prefix for block transfer of data

An Example I/O Device

- Keyboard
  - Keyboard controller scans and reports
    - – Key depressions and releases
    - » Supplies key identity as a scan code
      - – Scan code is like a sequence number of the key
        - ✈Key's scan code depends on its position on the keyboard
        - ✈No relation to the ASCII value of the key
  - Interfaced through an 8-bit parallel I/O port
    - » Originally supported by 8255 programmable peripheral interface chip (PPI)

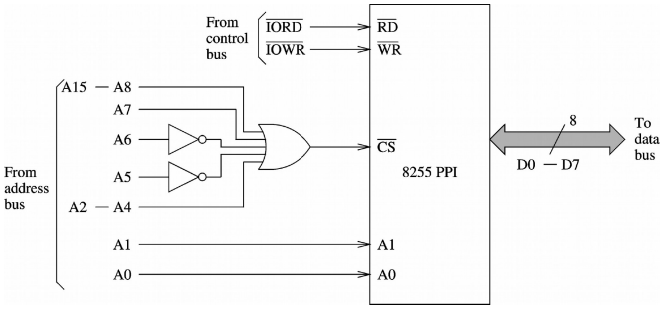
An Example I/O Device (cont'd)

- 8255 PPI has three 8-bit registers
  - » Port A (PA)
  - » Port B (PB)
  - » Port C (PC)
  - These ports are mapped as follows

8255 register	Port address
PA (input port)	60H
PB (output port)	61H
PC (input port)	62H
Command register	63H

An Example I/O Device (cont'd)

Mapping of 8255 I/O ports



An Example I/O Device (cont'd)

- Mapping I/O ports is similar to mapping memory
  - Partial mapping
  - Full mapping
    - » See our discussion in Chapter 16
- Keyboard scan code and status can be read from port 60H
  - 7-bit scan code is available from
    - » PA0 – PA6
  - Key status is available from PA7
    - » PA7 = 0 – key depressed
    - » PA0 = 1 – key released

I/O Data Transfer

- Data transfer involves two phases
  - A data transfer phase
    - » It can be done either by
      - – Programmed I/O
      - – DMA
  - An end-notification phase
    - » Programmed I/O
    - » Interrupt
- Three basic techniques
  - Programmed I/O
  - DMA
  - Interrupt-driven I/O (discussed in Chapter 20)

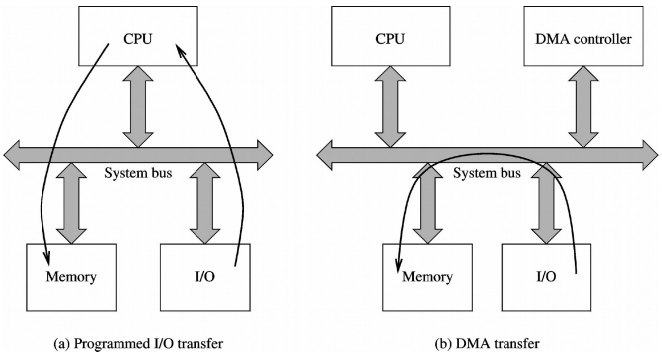
I/O Data Transfer (cont'd)

- Programmed I/O
  - Done by busy-waiting
    - » This process is called ***polling***
- Example
  - Reading a key from the keyboard involves
    - » Waiting for PA7 bit to go low
      - – Indicates that a key is pressed
    - » Reading the key scan code
    - » Translating it to the ASCII value
    - » Waiting until the key is released
  - Program 19.1 uses this process to read input from the keyboard

I/O Data Transfer (cont'd)

- Direct memory access (DMA)
  - Problems with programmed I/O
    - » Processor wastes time polling
      - – In our example
        - ✈Waiting for a key to be pressed,
        - ✈Waiting for it to be released
    - » May not satisfy timing constraints associated with some devices
      - – Disk read or write
  - DMA
    - » Frees the processor of the data transfer responsibility

I/O Data Transfer (cont'd)



I/O Data Transfer (cont'd)

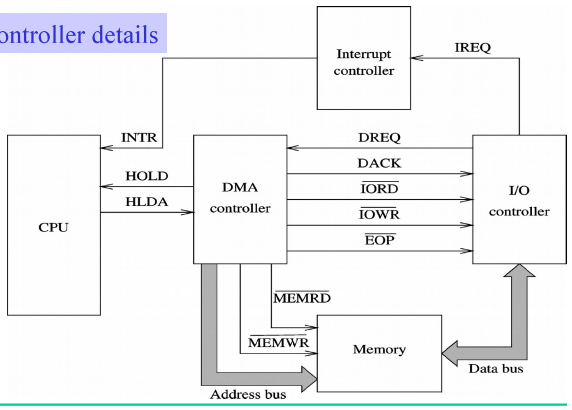
- DMA is implemented using a DMA controller
  - DMA controller
    - » Acts as slave to processor
    - » Receives instructions from processor
    - » Example: Reading from an I/O device
      - – Processor gives details to the DMA controller
        - ✈I/O device number
        - ✈Main memory buffer address
        - ✈Number of bytes to transfer
        - ✈Direction of transfer (memory → I/O device, or vice versa)

I/O Data Transfer (cont'd)

- Steps in a DMA operation
  - Processor initiates the DMA controller
    - » Gives device number, memory buffer pointer, …
      - – Called ***channel initialization***
    - » Once initialized, it is ready for data transfer
  - When ready, I/O device informs the DMA controller
    - » DMA controller starts the data transfer process
      - – Obtains bus by going through bus arbitration
      - – Places memory address and appropriate control signals
      - – Completes transfer and releases the bus
      - – Updates memory address and count value
      - – If more to read, loops back to repeat the process
  - Notify the processor when done
    - » Typically uses an interrupt

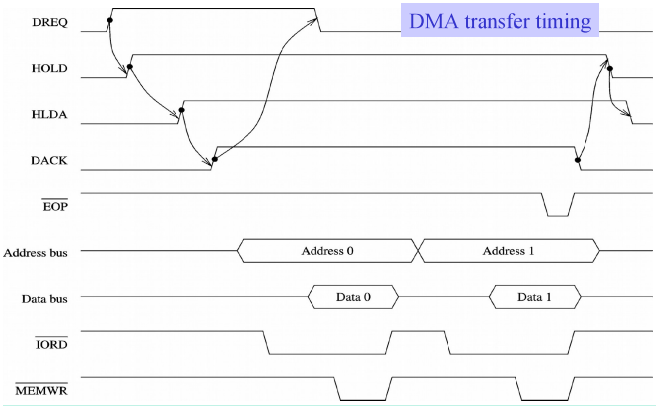
I/O Data Transfer (cont'd)

DMA controller details



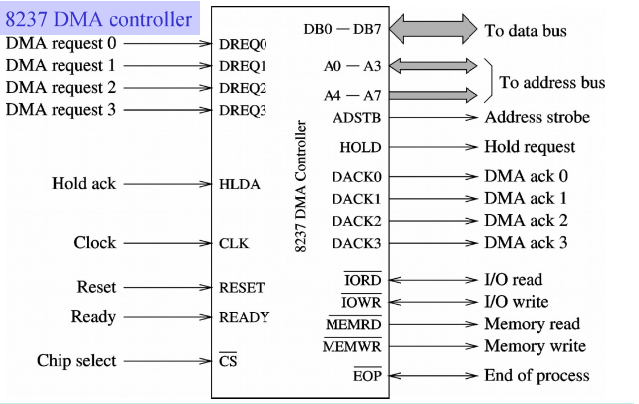
I/O Data Transfer (cont'd)

DMA transfer timing



I/O Data Transfer (cont'd)

8237 DMA controller



I/O Data Transfer (cont'd)

- 8237 supports four DMA channels
- It has the following internal registers
  - Current address register
    - » One 16-bit register for each channel
    - » Holds address for the current DMA transfer
  - Current word register
    - » Keeps the byte count
    - » Generates terminal count (TC) signal when the count goes from zero to FFFFH
  - Command register
    - » Used to program 8257 (type of priority, …)

I/O Data Transfer (cont'd)

- Mode register
  - » Each channel can be programmed to
    - – Read or write
    - – Autoincrement or autodecrement the address
    - – Autoinitialize the channel
- Request register
  - » For software-initiated DMA
- Mask register
  - » Used to disable a specific channel
- Status register
- Temporary register
  - » Used for memory-to-memory transfers

I/O Data Transfer (cont'd)

- 8237 supports four types of data transfer
  - Single cycle transfer
    - » Only single transfer takes place
    - » Useful for slow devices
  - Block transfer mode
    - » Transfers data until TC is generated or external EOP signal is received
  - Demand transfer mode
    - » Similar to the block transfer mode
    - » In addition to TC and EOP, transfer can be terminated by deactivating DREQ signal
  - Cascade mode
    - » Useful to expand the number channels beyond four

Error Detection and Correction

- Parity encoding
  - » Simplest mechanism
  - » Adds rudimentary error detection capability
  - Add a parity bit such that the total number of 1s is
    - » odd (odd-parity)
    - » even (even-parity)
  - Advantage:
    - » Simple to implement
  - Disadvantage
    - » Can be used to detect single-bit errors
      - – Cannot detect even number of bit errors

Error Detection and Correction (cont'd)

- Error correction
  - Need to know the error bit position
    - » To correct, simply flip the bit
  - To correct single-bit errors in ***d*** data bits
    - » Add ***p*** parity bits
    - » Codeword $C = d + p$ bits
  - How many parity bits do we need?
    - » Depends on ***d***
    - » Hamming distance between codewords
      - – Number of bit positions in which the two codewords differ
    - » Hamming distance of code
      - – Smallest Hamming distance between any pair of codewords in the code

Error Detection and Correction (cont'd)

- Constructing codewords to correct single bit errors
  - Count bit positions from left to right starting from 1
  - Parity bits are in positions that are a power of 2
    - » Parity bits are called ***check bits***
    - » Example for 8-bit data ($d = 8$)
      - – We need 4 check bits
      - – Codeword is 12 bits long

P1	P2	D7	P4	D6	D5	D4	P8	D3	D2	D1	D0
		1		0	1	0		1	0	1	0
1	2	3	4	5	6	7	8	9	10	11	12

Error Detection and Correction (cont'd)

- Check bits are derived as in the parity scheme
  - » Uses even parity
  - Each check bit is responsible for checking certain bits
    - » P1: checks bits 1, 3, 5, 7, 9, and 11
    - » P2: checks bits 2, 3, 6,7, 10, and 11
    - » P4: checks bits 4,5,6,7, and 12
    - » P8: checks bits 8,9,10,11, and 12
  - Example

P1	P2	D7	P4	D6	D5	D4	P8	D3	D2	D1	D0
1	1	1	1	0	1	0	0	1	0	1	0
1	2	3	4	5	6	7	8	9	10	11	12

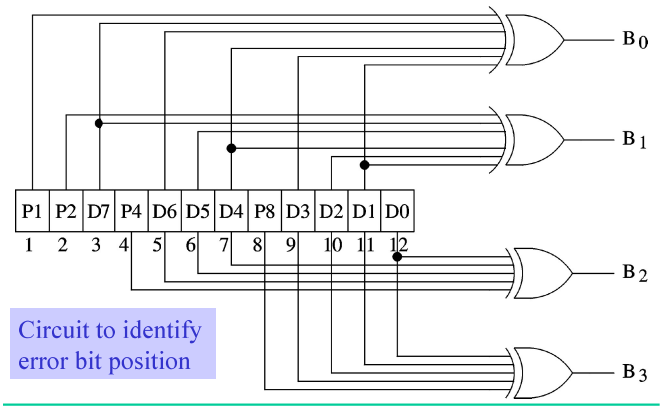
Error Detection and Correction (cont'd)

- How is error bit position computed?

  Error bit position = sum of weights of check bits in error
  - Suppose P1, P2, and P8 are in error but not P4
    - » Error bit position = $1 + 2 + 8 = 11^{th}$ bit
- What is the logic?
  - Write the numbers 1,2, 3,4,… in binary
    - » P1 checks those bit positions for which the rightmost column has 1 (i.e., with weight $2^0 = 1$)
    - » P2 check those bits positions for which the second rightmost column has 1 (i.e., with weight $2^1 = 2$)
    - » . . .

```
0: 0000
1: 0001
2: 0010
3: 0011
4: 0100
5: 0101
6: 0110
7: 0111
8: 1000
```

Error Detection and Correction (cont'd)



Error Detection and Correction (cont'd)

- SECDED
  - Single-error correction and double error detection
    - » Often used in high-performance systems
  - Previous scheme gives single-error detection and correction capability
  - To incorporate double error detection
    - » Add an additional parity bit
      - – This bit is added as the leftmost bit P0 that is not used by the error correction scheme
    - » Example
      - – Previous example would have 8 data bits and ***5*** check bits for SECDED capability

Error Detection and Correction (cont'd)

- CRC
  - Cyclic redundancy check
    - » Computed for a block of data
    - » Widely used to detect burst errors
    - » Uses fixed number of bits
      - – Mostly 16 or 32 bits depending on the block size
  - Basic idea: If

$$\frac{D}{G} = Q + R$$

$$\frac{D-R}{G} = Q$$

    - » Based on integer division

Error Detection and Correction (cont'd)

- CRC uses a polynomial of degree ***n***
  - Example:
    - » USB polynomial for data packets

$$x^{16} + x^{15} + x^2 + 1$$

    - » USB polynomial for token packets

$$x^5 + x^2 + 1$$

    - » Polynomial identifies the 1 bit positions
      - – USB data polynomial: `1100000000000101`
      - – USB token polynomial: `100101`
    - » Such polynomials are called ***polynomial generators***

Error Detection and Correction (cont'd)

- A bit of theory behind CRC

  C = (***d*** + ***n***)-bit codeword

  D = ***d***-bit data

  R = ***n***-bit remainder (i.e., CRC code)

  G = degree ***n*** polynomial generator

  - Goal: To generate ***C*** such that
    - » Remainder of (C/G) = 0
  - Since we append ***n*** bits to the right

$$C = D \times 2^n \oplus R$$

  $\oplus$ = XOR operation

Error Detection and Correction (cont'd)

- We generate R as

$$\frac{D \times 2^n}{G} = Q \oplus \frac{R}{G}$$

- Add the remainder R to generate the codeword
- When this codeword is divided by G, we get remainder as zero

$$\frac{C}{G} = \frac{D \times 2^n \oplus R}{G}$$

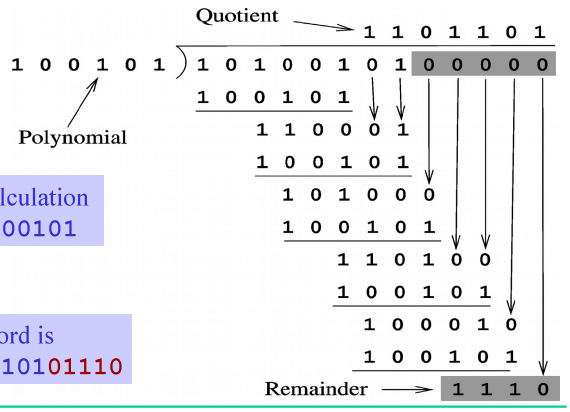
  - » From above, we get

$$\frac{C}{G} = Q \oplus \frac{R}{G} \oplus \frac{R}{G} = Q \oplus \frac{R \oplus R}{G} = Q$$

zero

Error-free condition

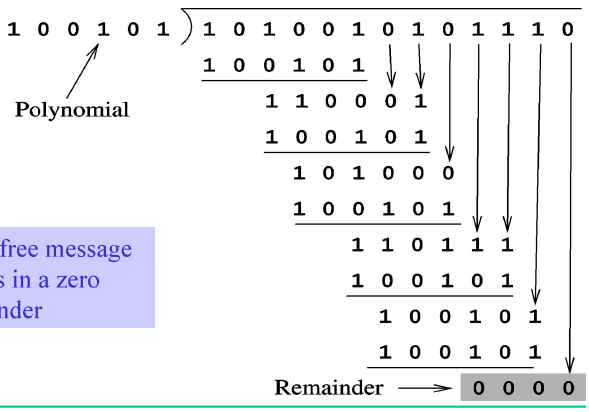
Error Detection and Correction (cont'd)



CRC calculation for 10100101

Codeword is 1010010101110

Error Detection and Correction (cont'd)

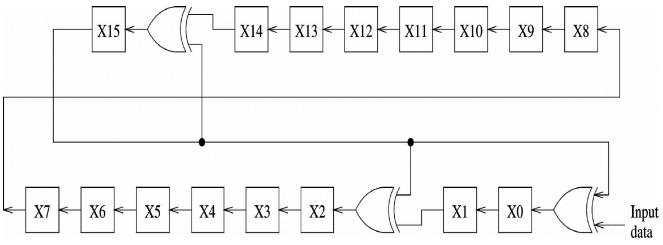


Error free message results in a zero remainder

Error Detection and Correction (cont'd)

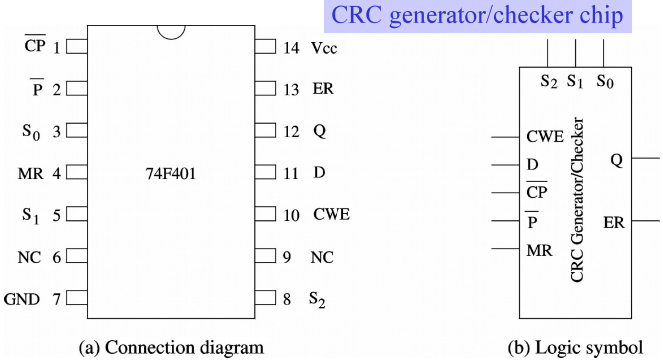
- A serial CRC generator circuit
  - Uses polynomial generator

$$x^{16} + x^{15} + x^2 + 1$$



Error Detection and Correction (cont'd)

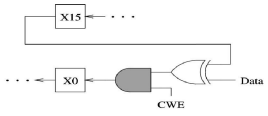
CRC generator/checker chip



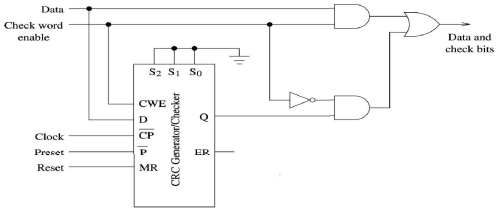
Error Detection and Correction (cont'd)

Using 74F401 to generate CRC for the generator

$$x^{16} + x^{15} + x^2 + 1$$



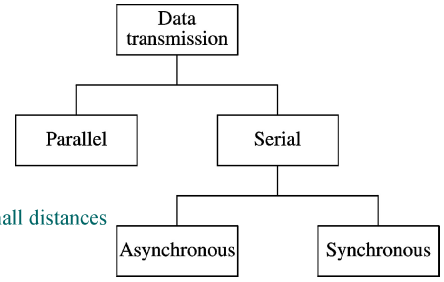
(a) Gated Code Word Enable input



(b) Check word generation

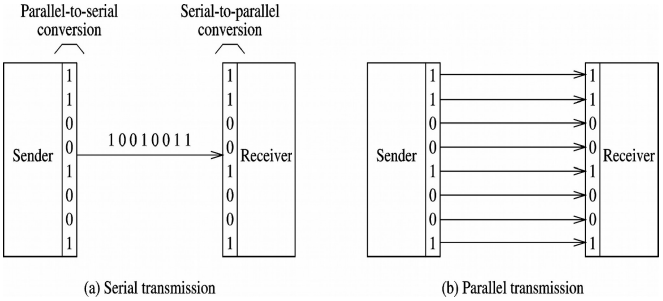
External Interface

- Two ways of interfacing I/O devices
  - Serial
    - » Cheaper
    - » Slower
  - Parallel
    - » Faster
    - » Data skew
    - » Limited to small distances



External Interface (cont'd)

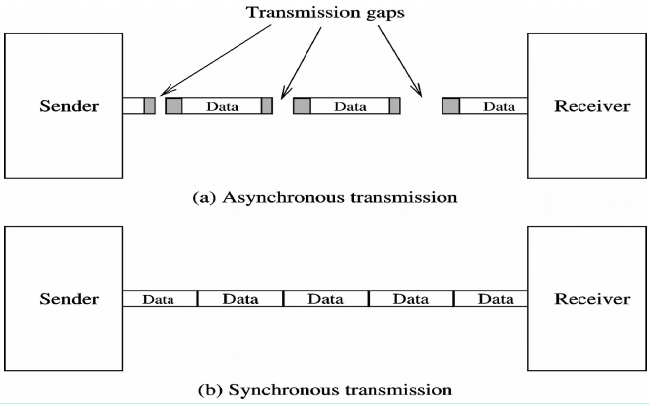
Two basic modes of data transmission



External Interface (cont'd)

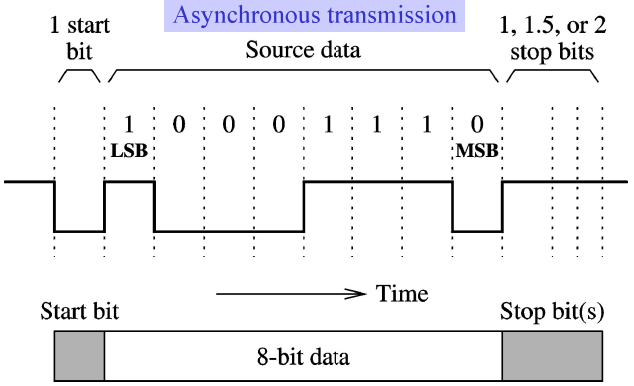
- Serial transmission
  - Asynchronous
    - » Each byte is encoded for transmission
      - – Start and stop bits
    - » No need for sender and receiver synchronization
  - Synchronous
    - » Sender and receiver must synchronize
      - – Done in hardware using phase locked loops (PLLs)
    - » Block of data can be sent
    - » More efficient
      - – Less overhead than asynchronous transmission
    - » Expensive

External Interface (cont'd)



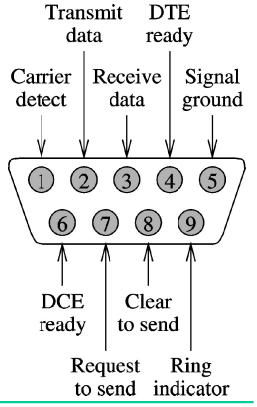
External Interface (cont'd)

Asynchronous transmission



External Interface (cont'd)

- EIA-232 serial interface
  - Low-speed serial transmission
  - Adopted by Electronics Industry Association (EIA)
    - » Popularly known by its predecessor RS-232
  - It uses a 9-pin connector DB-9
    - » Uses 8 signals
  - Typically used to connect a modem to a computer



External Interface (cont'd)

- Transmission protocol uses three phases
  - Connection setup
    - » Computer A asserts DTE Ready
      - – Transmits phone# via Transmit Data line (pin 2)
    - » Modem B alerts its computer via Ring Indicator (pin 9)
      - – Computer B asserts DTE Ready (pin 4)
      - – Modem B generates carrier and turns its DCE Ready
    - » Modem A detects the carrier signal from modem B
      - – Modem A alters its computer via Carrier Detect (pin 1)
      - – Turns its DCE Ready
  - Data transmission
    - » Done by handshaking using
      - – request-to-send (RTS) and clear-to-send (CTS) signals
  - Connection termination
    - » Done by deactivating RTS

External Interface (cont'd)

- Parallel printer interface
  - A simple parallel interface
  - Uses 25-pin DB-25
    - » 8 data signals
      - – Latched by strobe (pin 1)
    - » Data transfer uses simple handshaking
      - – Uses acknowledge (CK) signal
        - ✈After each byte, computer waits for ACK
    - » 5 lines for printer status
      - – Busy, out-of-paper, online/offline, autofeed, and fault
    - » Can be initialized with INIT
      - – Clears the printer buffer and resets the printer

External Interface (cont'd)

Table 19.3 Parallel printer interface signals

Pin #	Signal	Signal direction	Signal function
1	STROBE	PC ⟹ printer	Clock used to latch data
2	Data 0	PC ⟹ printer	Data bit 0 (LSB)
3	Data 1	PC ⟹ printer	Data bit 1
4	Data 2	PC ⟹ printer	Data bit 2
5	Data 3	PC ⟹ printer	Data bit 3
6	Data 4	PC ⟹ printer	Data bit 4
7	Data 5	PC ⟹ printer	Data bit 5
8	Data 6	PC ⟹ printer	Data bit 6
9	Data 7	PC ⟹ printer	Data bit 7 (MSB)
10	ACK	printer ⟹ PC	Printer acknowledges receipt of data
11	BUSY	printer ⟹ PC	Printer is busy
12	POUT	printer ⟹ PC	Printer is out of paper
13	SEL	printer ⟹ PC	Printer is online
14	AUTO FEED	printer ⟹ PC	Autofeed is on
15	FAULT	printer ⟹ PC	Printer fault
16	INIT	PC ⟹ printer	Clears printer buffer and resets printer
17	SLCT IN	PC ⟹ printer	TTL high level
18–25	Ground	N/A	Ground reference

External Interface (cont'd)

- SCSI
  - Pronounced "scuzzy"
  - Small Computer System Interface
    - » Supports both internal and external connection
  - Comes in two bus widths
    - » 8 bits
      - – Known as ***narrow SCSI***
      - – Uses a 50-pin connector
      - – Device id can range from 0 to 7
    - » 16 bits
      - – Known as ***wide SCSI***
      - – Uses a 68-pin connector
      - – Device id can range from 0 to 15

External Interface (cont'd)

Table 19.4 Types of SCSI

SCSI type	Bus width (bits)	Transfer rate MB/s
SCSI 1	8	5
Fast SCSI	8	10
Ultra SCSI	8	20
Ultra 2 SCSI	8	40
Wide Ultra SCSI	16	40
Wide Ultra 2 SCSI	16	80
Ultra 3 (Ultra 160) SCSI	16	160
Ultra 4 (Ultra 320) SCSI	16	320

External Interface (cont'd)

Table 19.5 Narrow SCSI signals

Description	Signal	Pin	Pin	Signal	Description
Twisted pair ground	GND	1	26	D0	Data 0
Twisted pair ground	GND	2	27	D1	Data 1
Twisted pair ground	GND	3	28	D2	Data 2
Twisted pair ground	GND	4	29	D3	Data 3
Twisted pair ground	GND	5	30	D4	Data 4
Twisted pair ground	GND	6	31	D5	Data 5
Twisted pair ground	GND	7	32	D6	Data 6
Twisted pair ground	GND	8	33	D7	Data 7
Twisted pair ground	GND	9	34	DP	Data parity bit
Ground	GND	10	35	GND	Ground
Ground	GND	11	36	GND	Ground
Reserved		12	37		Reserved
No connection		13	38	TermPwr	Termination power (+5 V)

cont'd

External Interface (cont'd)

Reserved		14	39		Reserved
Ground	GND	15	40	GND	Ground
Twisted pair ground	GND	16	41	ATN	Attention
Ground	GND	17	42	GND	Ground
Twisted pair ground	GND	18	43	BSY	Busy
Twisted pair ground	GND	19	44	ACK	Acknowledge
Twisted pair ground	GND	20	45	RST	Reset
Twisted pair ground	GND	21	46	MSG	Message
Twisted pair ground	GND	22	47	SEL	Selection
Twisted pair ground	GND	23	48	C/D	Command/data
Twisted pair ground	GND	24	49	REQ	Request
Twisted pair ground	GND	25	50	I/O	Input/output

External Interface (cont'd)

- SCSI uses client-server model
  - Uses terms ***initiator*** and ***target*** for client and server
    - » Initiator issues commands to targets to perform a task
      - – Initiators are typically SCSI host adaptors
    - » Targets receive the command and perform the task
      - – Targets are SCSI devices like disk drives
- SCSI transfer proceeds in phases
  - Command
  - Message in
  - Message out
  - Data in
  - Data out
  - Status

IN and OUT from the initiator point of view

External Interface (cont'd)

- SCSI uses asynchronous mode for all bus negotiations
  - » Uses handshaking using REQ and ACK signals for each byte of data
- On a synchronous SCSI
  - » Data are transferred synchronously
  - » REQ-ACK signals are not used for each byte
  - » A number of bytes (e.g., 8) can be sent without waiting for ACK
    - – Improves throughput
    - – Minimizes adverse impact of cable propagation delay

USB

- Universal Serial Bus
  - Originally developed in 1995 by a consortium including
    - » Compaq, HP, Intel, Lucent, Microsoft, and Philips
  - USB 1.1 supports
    - » Low-speed devices (1.5 Mbps)
    - » Full-speed devices (12 Mbps)
  - USB 2.0 supports
    - » High-speed devices
      - – Up to 480 Mbps (a factor of 40 over USB 1.1)
    - » Uses the same connectors
      - – Transmission speed is negotiated on device-by-device basis

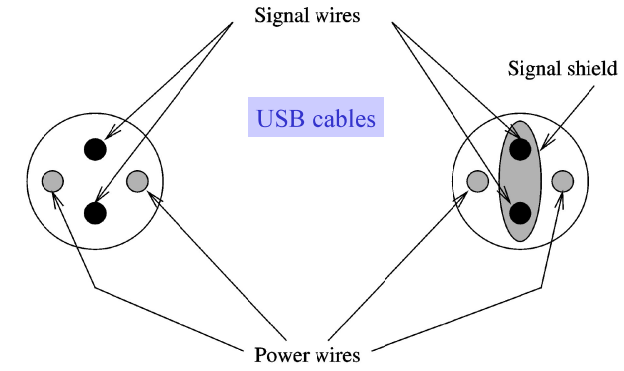
USB (cont'd)

- Motivation for USB
  - Avoid device-specific interfaces
    - » Eliminates multitude of interfaces
      - – PS/2, serial, parallel, monitor, microphone, keyboard,…
  - Avoid non-shareable interfaces
    - » Standard interfaces support only one device
  - Avoid I/O address space and IRQ problems
    - » USB does not require memory or address space
  - Avoid installation and configuration problems
    - » Don't have to open the box to install and configure jumpers
  - Allow hot attachment of devices

USB (cont'd)

- Additional advantages of USB
  - Power distribution
    - » Simple devices can be bus-powered
      - – Examples: mouse, keyboards, floppy disk drives, wireless LANs, …
  - Control peripherals
    - » Possible because USB allows data to flow in both directions
  - Expandable through hubs
  - Power conservation
    - » Enters suspend state if there is no activity for 3 ms
  - Error detection and recovery
    - » Uses CRC

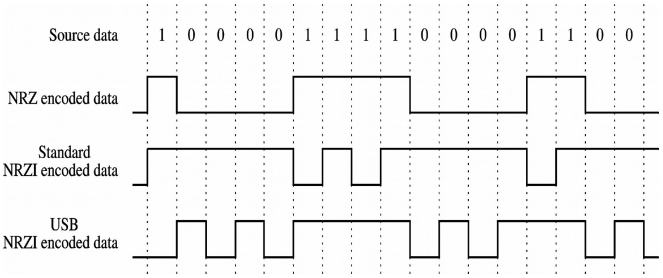
USB (cont'd)



(a) Low-speed cable (b) Full-speed/High-speed cable

USB (cont'd)

- USB encoding
  - Uses NRZI encoding
    - » Non-Return to Zero-Inverted

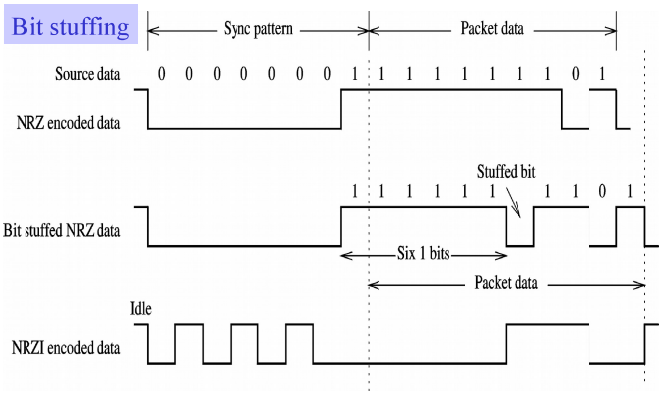


USB (cont'd)

- NRZI encoding
  - A signal transition occurs if the next bit is zero
    - » It is called ***differential encoding***
  - Two desirable properties
    - » Signal transitions, not levels, need to be detected
    - » Long string of zeros causes signal changes
  - Still a problem
    - » Long strings of 1s do not causes signal change
  - To solve this problem
    - » Uses ***bit stuffing***
      - – A zero is inserted after every six consecutive 1s

USB (cont'd)

Bit stuffing



USB (cont'd)

- Transfer types
  - » Four types of transfer
  - Interrupt transfer
    - » Uses polling
      - – Polling interval can range from 1 ms to 255 ms
  - Isochronous transfer
    - » Used in real-time applications that require constant data transfer rate
      - – Example: Reading audio from CD-ROM
    - » These transfers are scheduled regularly
    - » Do not use error detection and recovery

USB (cont'd)

- Control transfer
  - » Used to configure and set up USB devices
  - » Three phases
    - – Setup stage
      - ✈Conveys type of request made to target device
    - – Data stage
      - ✈Optional stage
      - ✈Control transfers that require data use this stage
    - – Status stage
      - ✈Checks the status of the operation
  - » Allocates a guaranteed bandwidth of 10%
  - » Error detection and recovery are used
    - – Recovery is by means of retries

USB (cont'd)

- Bulk transfer
  - » For devices with no specific data transfer rate requirements
    - – Example: sending data to a printer
  - » Lowest priority bandwidth allocation
  - » If the other three types of transfers take 100% of the bandwidth
    - – Bulk transfers are deferred until load decreases
  - » Error detection and recovery are used
    - – Recovery is by means of retries

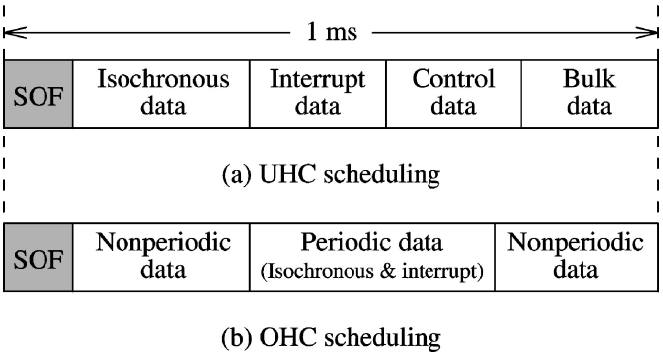
USB (cont'd)

- USB architecture
  - USB host controller
    - » Initiates transactions over USB
  - Root hub
    - » Provides connection points
  - Two types of host controllers
    - » Open host controller (OHC)
      - – Defined by Intel
    - » Universal host controller (UHC)
      - – Specified by National Semiconductor, Microsoft, Compaq
    - » Difference between the two
      - – How they schedule the four types of transfers

USB (cont'd)

- UHC scheduling
  - Schedules periodic transfers first
    - » Periodic transfers: isochronous and interrupts
    - » Can take up to 90% of bandwidth
  - These transfers are followed by control and bulk transfers
    - » Control transfers are guaranteed 10% of bandwidth
  - Bulk transfers are scheduled only if there is bandwidth available

USB (cont'd)



USB (cont'd)

- OHC scheduling
  - Different from UHC scheduling
  - Reserves space for non-periodic transfers first
    - » Non-periodic transfers: control and bulk
    - » 10% bandwidth reserved
  - Next periodic transfers are scheduled
    - » Guarantees 90% bandwidth
  - Left over bandwidth is allocated to non-periodic transfers

USB (cont'd)

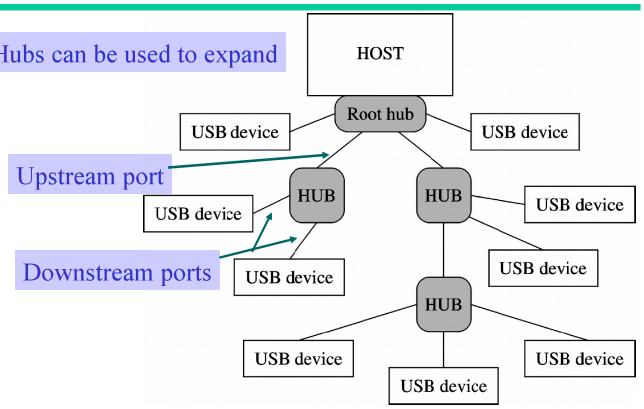
- Bus powered devices
  - Low-power
    - » Less than 100 mA
    - » Can be bus-powered
  - High-powered
    - » Between 100 mA and 500 mA
      - – Full-powered ports can power these devices
    - » Can be designed to have their own power
    - » Operate in three modes
      - – Configured (500 mA)
      - – Unconfigured (100 mA)
      - – Suspended (about 2.5 mA)

USB (cont'd)

- USB hubs
  - Bus-powered
    - » No extra power supply required
    - » Must be connected to an upstream port that can supply 500 mA
    - » Downstream ports can only supply 100 mA
      - – Number of ports is limited to four
      - – Support only low-powered devices
  - Self-powered
    - » Support 4 high-powered devices
    - » Support 4 bus-powered USB hubs
  - Most 4-port hubs are dual-powered

USB (cont'd)

Hubs can be used to expand

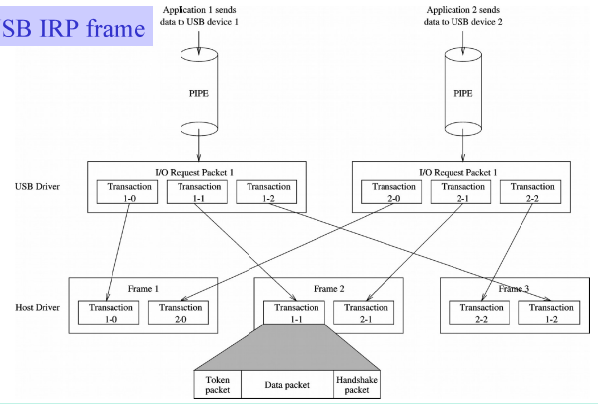


USB (cont'd)

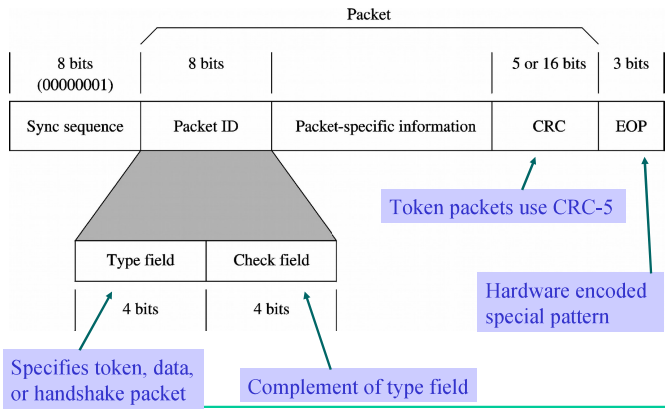
- USB transactions
  - Transfers are done in one or more transactions
    - » Each transaction consists of several packets
  - Transactions may have between 1 and 3 phases
    - » Token packet phase
      - – Specifies transaction type and target device address
    - » Data packet phase (optional)
      - – Maximum of 1023 bytes are transferred
    - » Handshake packet phase
      - – Except for isochronous transfers, others use error detection for guaranteed delivery
      - – Provides feedback on whether data has been received without error

USB (cont'd)

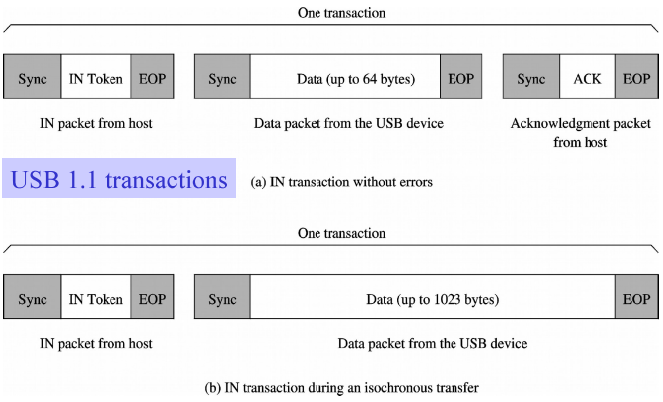
USB IRP frame



USB (cont'd)



USB (cont'd)



USB 1.1 transactions

(a) IN transaction without errors

(b) IN transaction during an isochronous transfer

USB (cont'd)

- USB 2.0
  - USB 1.1 uses 1 ms frames
  - USB 2.0 uses 125 µs frames
    - » 1/8 of USB 1.1
  - Supports 40X data rates
    - » Up to 480 Mbps
  - Competitive with
    - » SCSI
    - » IEEE 1394 (FireWire)
  - Widely available now

IEEE 1394

- Apple originally developed this standard for high-speed peripherals
  - Known by a variety of names
    - » Apple: FireWire
    - » Sony: i.ILINK
  - IEEE standardized it as IEEE 1394
    - » First released in 1995 as IEEE 1394-1995
    - » A slightly revised version as 1394a
    - » Next version 1394b
  - Shares many of the features of USB

IEEE 1394 (cont'd)

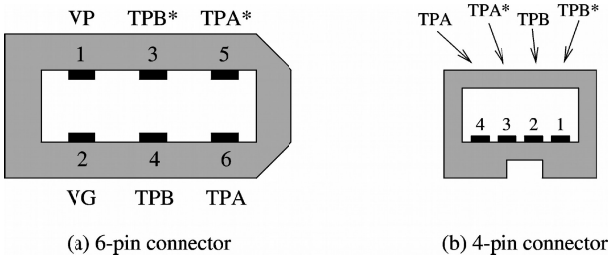
- Advantages
  - High speed
    - » Supports three speeds
      - – 100, 200, 400 Mbps
        - ✈Competes with USB 2.0
      - – Plans to boost it to 3.2 Gbps
  - Hot attachment
    - » Like USB
    - » No need to shut down power to attach devices
  - Peer-to-peer support
    - » USB is processor-centric
    - » Supports peer-to-peer communication without involving the processor

IEEE 1394 (cont'd)

- Expandable bus
  - » Devices can be connected in daisy-chain fashion
  - » Hubs can used to expand
- Power distribution
  - » Like the USB, cables distribute power
    - – Much higher power than USB
      - ✈Voltage between 8 and 33 V
      - ✈Current an be up to 1.5 Amps
- Error detection and recovery
  - » As in USB, uses CRC
  - » Uses retransmission in case of error
- Long cables
  - » Like the USB

IEEE 1394 (cont'd)

IEEE 1394 6-pin and 4-pin connectors

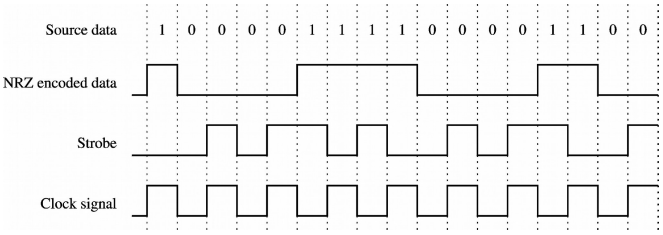


(a) 6-pin connector (b) 4-pin connector

4-pin connector does not distribute power

IEEE 1394 (cont'd)

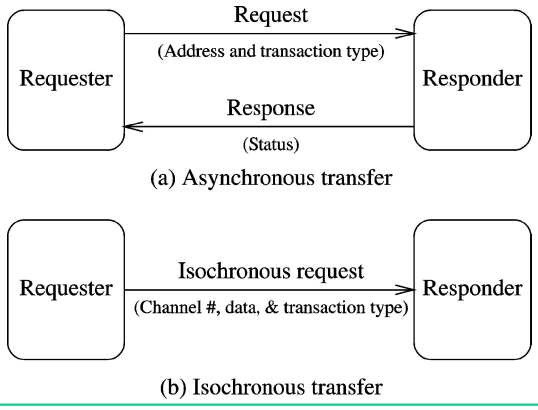
- Encoding
  - Uses a simple NRZ encoding
  - Strobe signal is encoded
    - » Changes the signal even if successive bits are the same



IEEE 1394 (cont'd)

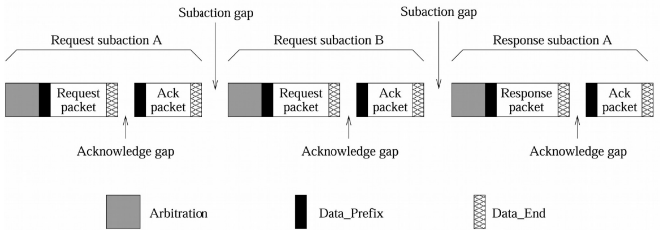
- Transfer types
  - Asynchronous
    - » For applications that require correct delivery of data
      - – Example: writing a file to a disk drive
    - » Uses an acknowledgement to confirm delivery
    - » Guaranteed bandwidth of 20%
  - Isochronous
    - » For real-time applications
    - » No acknowledgement
    - » Up to 80% of bandwidth allocated
  - Bandwidth allocation on a cycle-by-cycle basis
    - » Cycle time: 125 μs

IEEE 1394 (cont'd)



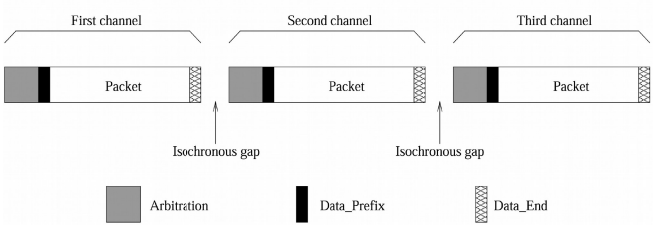
IEEE 1394 (cont'd)

- Transactions
  - Follow request and reply format
  - Each packet is encapsulated between `Data_Prefix` and `Data_end`



IEEE 1394 (cont'd)

- Isochronous transactions
  - Similar to asynchronous transactions
  - Main difference:
    - » No acknowledgement packets



IEEE 1394 (cont'd)

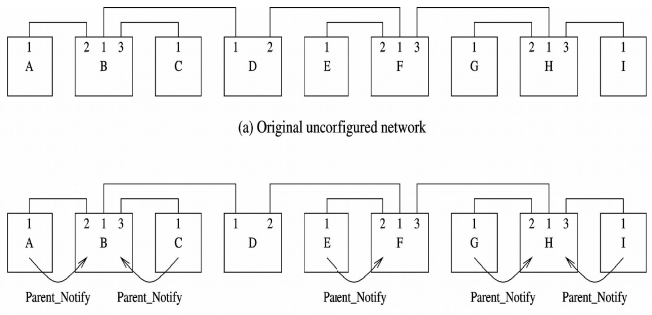
- Bus arbitration
  - Needed because of peer-to-peer communication
  - Arbitration must respect
    - » Bandwidth allocation to isochronous channels
    - » Fairness-based allocation for asynchronous channels
  - Uses fairness interval
    - » During each interval
      - – All nodes with pending asynchronous transaction are allowed bus ownership once
  - Nodes with pending isochronous transactions go through arbitration during each cycle
  - IRM is used for isochronous bandwidth allocations

IEEE 1394 (cont'd)

- Configuration
  - Does not require the host system
  - Consists of two main phases
    - » Tree identification
      - – Used to find the network topology
      - – Uses two special signals
        - → `Parent_notify` and `Child_Notify`
    - » Self-identification
      - – Done after the tree identification
      - – Assigns unique ids to nodes

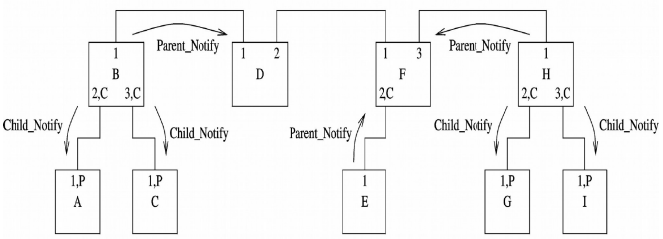
IEEE 1394 (cont'd)

Tree identification



IEEE 1394 (cont'd)

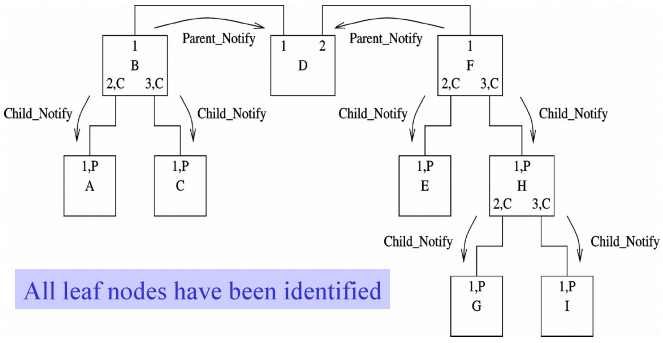
Tree identification



(c) Topology starts to take shape with nodes A, C, E, G, and I identified as leaf nodes

IEEE 1394 (cont'd)

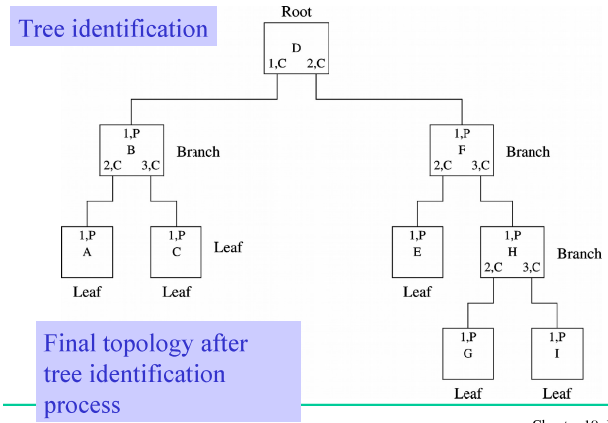
Tree identification



All leaf nodes have been identified

IEEE 1394 (cont'd)

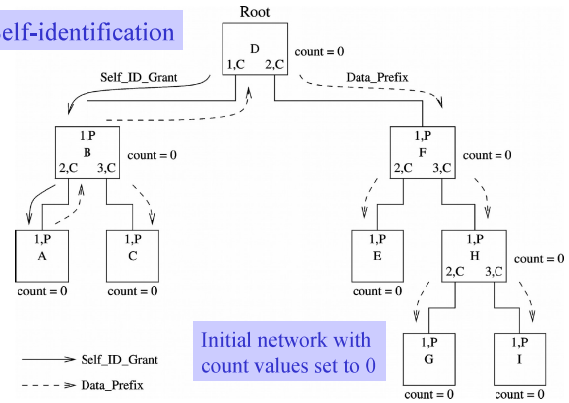
Tree identification



Final topology after tree identification process

IEEE 1394 (cont'd)

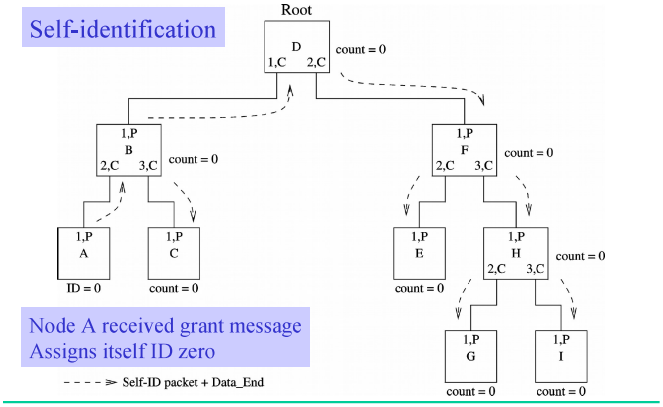
Self-identification



Initial network with count values set to 0

IEEE 1394 (cont'd)

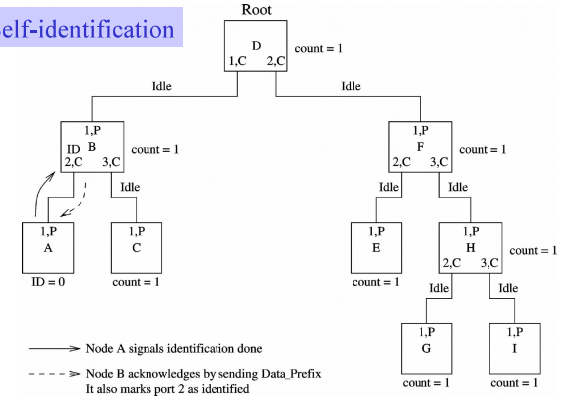
Self-identification



Node A received grant message
Assigns itself ID zero

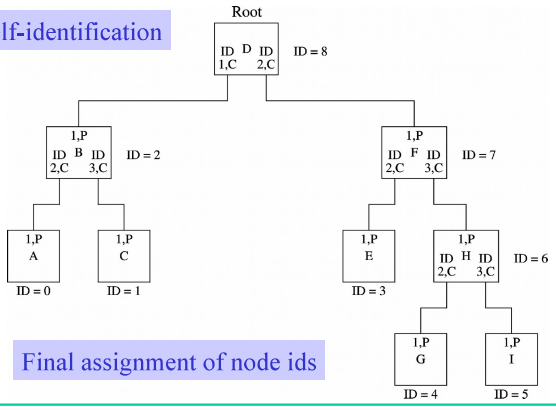
IEEE 1394 (cont'd)

Self-identification



IEEE 1394 (cont'd)

Self-identification



Final assignment of node ids

Bus Wars

- SCSI is dominant in disk and storage device interfaces
  - Parallel interface
  - Its bandwidth could go up to 640 MB/s
- IEEE 1394
  - Serial interface
  - Supports peer-to-peer applications
  - Dominant in video applications
- USB
  - Useful in low-cost, host-to-peripheral applications
  - USB 2.0 provides high-speed support

Last slide

Parameters of Cache memory

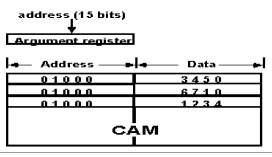
- Cache Hit
  - A referenced item is found in the cache by the processor
- Cache Miss
  - A referenced item is not present in the cache
- Hit ratio
  - Ratio of number of hits to total number of references => number of hits/(number of hits + number of Miss)
- Miss penalty
  - Additional cycles required to serve the miss

Parameters of Cache Memory

- Time required for the cache miss depends on both the latency and bandwidth
- Latency – time to retrieve the first word of the block
- Bandwidth – time to retrieve the rest of this block

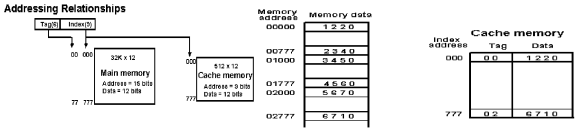
Associative Mapping

- Fastest, most flexible but very expensive
- Any block location in cache can store any block in memory
- Stores both the address and the content of the memory word
- CPU address of 15 bits is placed in the argument register and the associative memory is searched for a matching address
- If found data is read and sent to the CPU else main memory is accessed.
- CAM – content addressable memory



Direct Mapping

- N-bit CPU memory address – k bits index field and (n-k) bits tag field
- Index bits are used to access the cache
- Each word in cache consists of data word and its associated tag



On memory request, index field is used to access the cache. Tag field of CPU address is compared with the tag in the word read in the cache

If match then there is hit

Else miss – word is read from the memory

It is then stored in the cache together with the new tag replacing the previous value

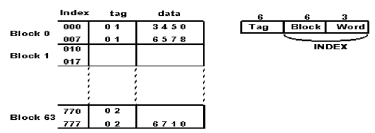
Disadvantage: hit ratio drops if 2 or more words with same index but different tags are accessed repeatedly.

When memory is divided into blocks of words, index field – block field and word field

Ex: 512 words cache – 64 blocks of 8words each – block field (6bits) and words field (3bits)

- Tags within the block are same.

Direct Mapping with block size of 8 words



- When a miss occurs, entire block is transferred from main memory to cache.
- It is time consuming but improves hit ratio because of the sequential nature of programs.

Set-Associative Mapping

- Each word of cache can store 2 or more words of memory under the same index address

Set Associative Mapping Cache with set size of two

Index	Tag	Data	Tag	Data
000	0 1	3 4 5 0	0 2	5 6 7 0
777	0 2	6 7 1 0	0 0	2 3 4 0

The comparison logic is done by an associative search of the tags in the set similar to an associative memory search, thus the name "Set-Associative"

- Hit ratio increases as the set size increases but more complex comparison logic is required when number of bits in words of cache increases
- When a miss occurs and set is full, one of tag-data items are replaced using block replacement policy

Problems

- A set associative cache consists of 64 lines or slots, divided into four line sets. Main memory consists 4k blocks of 128 words each. Show the format of main memory addresses.

The cache is divided into 16 sets of 4 lines each. Therefore, 4 bits are needed to identify the set number. Main memory consists of 4K = 2^{12} blocks. Therefore, the set plus tag lengths must be 12 bits and therefore the tag length is 8 bits. Each block contains 128 words. Therefore, 7 bits are needed to specify the word.

	TAG	SET	WORD
Main memory address =	8	4	7

Problem 2

- A two-way set associative cache has lines of 16 bytes and a total size of 8k bytes. The 64-Mbyte main memory is byte addressable. Show the format of main memory addresses.

There are a total of 8 kbytes/16 bytes = 512 lines in the cache. Thus the cache consists of 256 sets of 2 lines each. Therefore 8 bits are needed to identify the set number. For the 64-Mbyte main memory, a 26-bit address is needed. Main memory consists of 64-Mbyte/16 bytes = 2^{22} blocks. Therefore, the set plus tag lengths must be 22 bits, so the tag length is 14 bits and the word field length is 4 bits.

	TAG	SET	WORD
Main memory address =	14	8	4

Block Replacement

- **Least Recently Used: (LRU)**

  Replace that block in the set that has been in the cache longest with no reference to it.
- **First Come First Out: (FIFO)**

  Replace that block in the set that has been in the cache longest.
- **Least Frequently Used: (LFU)**

  Replace that block in the set that has experienced the fewest references

Update Policies- Write Through

- Update main memory with every memory write operation
- Cache memory is updated in parallel if it contains the word at specified address.
- Advantage: main memory always contains the same data as the cache
- It is important during DMA transfers to ensure the data in main memory is valid
- Disadvantage: slow due to memory access time

Write Back

- Only cache is updated during write operation and marked by flag. When the word is removed from the cache, it is copied into main memory
- Memory is not up-to-date, i.e., the same item in cache and memory may have different value

Update policies

- **Write-Around**
  - **correspond to items not currently in the cache (i.e. write misses) the item could be updated in main memory only without affecting the cache.**
- **Write-Allocate**
  - **update the item in main memory and bring the block containing the updated item into the cache.**

Performance analysis

- Look through: The cache is checked first for a hit, and if a miss occurs then the access to main memory is started.
- Look aside: access to main memory in parallel with the cache lookup;

- **Look through**

TA = TC + (1-h)*TM
TC is the average cache access time
TM is the average access time
(Mean memory access time)

- **Look aside**

TA = h*TC + (1-h)*TM

- hit ratio $h = \frac{\text{number of references found in the cache}}{\text{total number of memory references}}$

- **Miss Ratio m=(1-h)**

Example: assume that a computer system employs a cache with an access time of 20ns and a main memory with a cycle time of 200ns. Suppose that the hit ratio for reads is 90%,

a) what would be the average access time for reads if the cache is a "look-through" cache?

The average read access time (**TA) = TC + (1-h)*TM**
20ns + 0.10*200ns = 40ns

b) what would be the average access time for reads if the cache is a "look-Aside" cache?

The average read access time in this case (**TA)**
= h*TC + (1-h)*TM= 0.9*20ns + 0.10*200ns = 38ns

Problem

- Consider a memory system with T_c = 100ns and T_m = 1200ns. If the effective access time is 10% greater than the cache access time, what is the hit ratio H in look-through cache?

$\Rightarrow T_A = T_C + (1-h)*T_M$
$\Rightarrow 1.1\ T_c = T_c + (1-h)*T_M$
$\Rightarrow 0.1T_C = (1-h)*\ T_M$
$\Rightarrow 0.1 * 100 = (1-h) *1200$
$\Rightarrow 1-h = 10/1200$
$\Rightarrow h = 1190/1200$

Sources of cache Misses

- Compulsory Misses
- Capacity Misses
- Cold Misses
- Conflict Misses

Sources of Cache Misses

- **Compulsory Misses:** These are misses that are caused by the cache being empty initially.
- **Cold Misses :** The very first access to a block will result in a miss because the block is not brought into cache until it is referenced.
- **Capacity Misses :** If the cache cannot contain all the blocks needed during the execution of a program, capacity misses will occur due to blocks being discarded and later retrieved.
- **Conflict Misses:** If the cache mapping is such that multiple blocks are mapped to the same cache entry

Cache organization

- Split cache
  - Separate caches for instructions and data
  - I-cache (Instruction) – mostly accessed sequentially
  - D-cache (data) – mostly random access
- Unified cache
  - Same cache for instruction and data
- Higher hit rate for unified cache as it balances between instruction and data
- Split caches eliminate contention for cache between the instruction processor and the execution unit – used for pipelining processes

Multilevel caches

- The penalty for a cache miss is the extra time that it takes to obtain the requested item from central memory.
- One way in which this penalty can be reduced is to provide another cache, the secondary cache, which is accessed in response to a miss in the primary cache.
- The primary cache is referred to as the L1 (level 1) cache and the secondary cache is called the L2 (level 2) cache.
- Most high-performance microprocessors include an L2 cache which is often located off-chip, whereas the L1 cache is located on the same chip as the CPU.
- With a two-level cache, central memory has to be accessed only if a miss occurs in both caches.

Example:

- A computer system employs a write-back cache with a 70% hit ratio for writes. The cache operates in look-aside mode and has a 90% read hit ratio. Reads account for 80% of all memory references and writes account for 20%. If the main memory cycle time is 200ns and the cache access time is 20ns, what would be the average access time for all references (reads as well as writes)?

The average access time for reads = 0.9*20ns + 0.1*200ns = 38ns.

The average write time = 0.7*20ns + 0.3*200ns = 74ns

Hence the overall average access time for combined reads and writes is

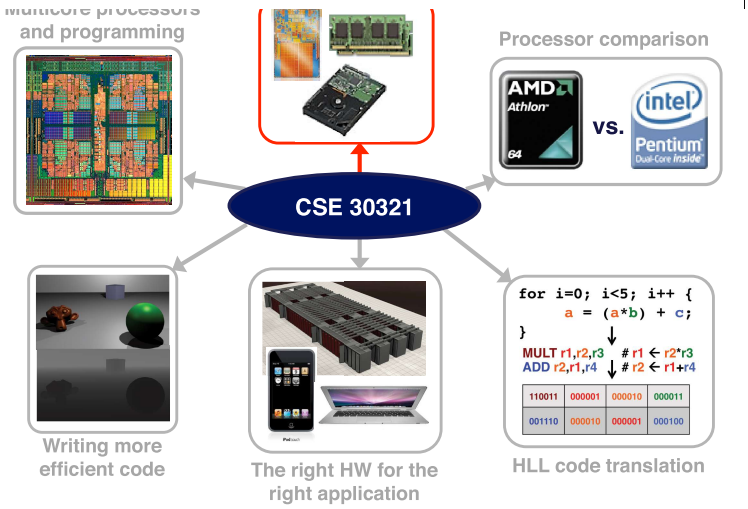
0.8*38ns + 0.2*74ns = 45.2ns

References

- J. L. Hennessy & D.A. Patterson, Computer architecture: A quantitative approach, Fourth Edition, Morgan Kaufman, 2004.

Lecture 13-14: Pipelines Hazards

Suggested reading:
(HP Chapter 4.5—4.7)



Fundamental lesson(s)

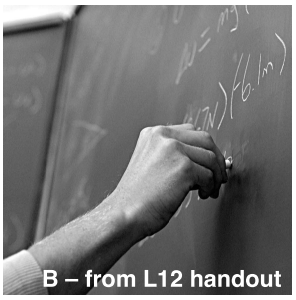
- **Pipelining changes the timing as to when the result(s) of an instruction are produced**
- **Additional HW is needed to ensure that the correct program results are produced while maintaining the speedups offered from the introduction of pipelining**
- **We must also account for the efficient pipelining of control instructions (e.g. beq) to preserve performance gains *and* program correctness**

Why it's important…

- **If you're a hardware designer OR a complier writer, you need to be aware of how HLL is mapped to assembly instructions AND what HW is used to execute a sequence of assembly instructions**
- **Otherwise, it is quite possible to always have built in inefficiencies**

On the board…

- Let's look at hazards…
  - …and how they (generally) impact performance.



B – from L12 handout

Pipelining hazards

- Pipeline hazards prevent next instruction from executing during designated clock cycle
- There are 3 classes of hazards:
  - Structural Hazards:
    - Arise from resource conflicts
    - HW cannot support all possible combinations of instructions
  - Data Hazards:
    - Occur when given instruction depends on data from an instruction ahead of it in pipeline
  - Control Hazards:
    - Result from branch, other instructions that change flow of program (i.e. change PC)

How do we deal with hazards?

- Often, pipeline must be stalled
- Stalling pipeline usually lets some instruction(s) in pipeline proceed, another/others wait for data, resource, etc.
- A note on terminology:
  - If we say an instruction was "issued later than instruction x", we mean that it was issued after instruction x and is not as far along in the pipeline
  - If we say an instruction was "issued earlier than instruction x", we mean that it was issued before instruction x and is further along in the pipeline

Stalls and performance

- Stalls impede progress of a pipeline and result in deviation from 1 instruction executing/clock cycle
- Pipelining can be viewed to:
  - Decrease CPI or clock cycle time for instruction
  - Let's see what affect stalls have on CPI…
- CPI pipelined =
  - Ideal CPI + Pipeline stall cycles per instruction
  - 1 + Pipeline stall cycles per instruction

Stalls and performance

- Ignoring overhead and assuming stages are balanced:

$$Speedup = \frac{CPI\ unpipelined}{1 + pipeline\ stall\ cycles\ per\ instruction}$$

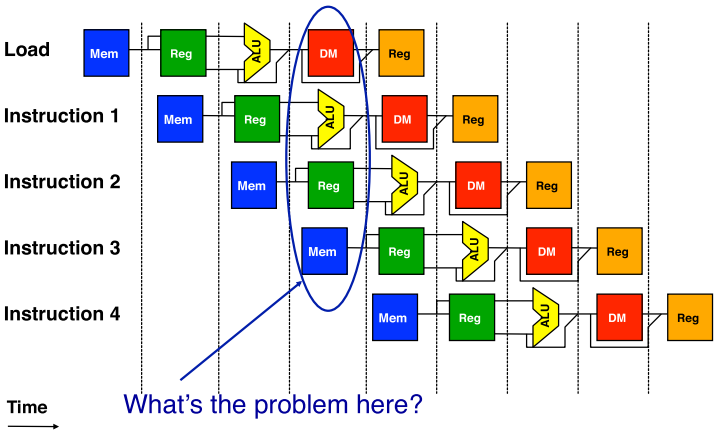
- If no stalls, speedup equal to # of pipeline stages in ideal case

STRUCTURAL HAZARDS

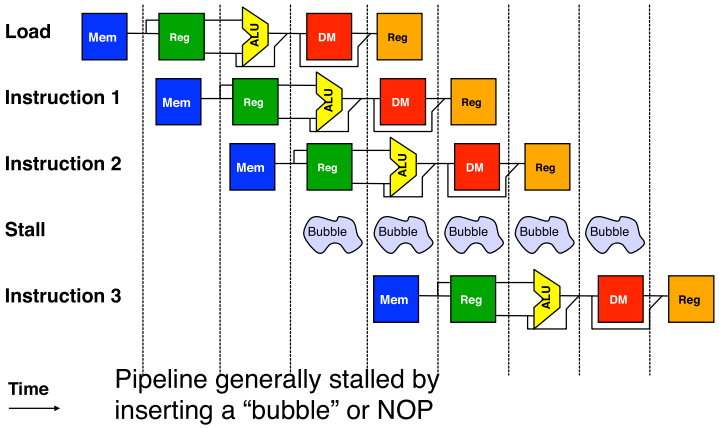
Structural hazards

- Avoid structural hazards by duplicating resources
  - e.g. an ALU to perform an arithmetic operation and an adder to increment PC
- If not all possible combinations of instructions can be executed, structural hazards occur
- Pipelines stall result of hazards, CPI increased from the usual "1"

An example of a structural hazard



How is it resolved?



Or alternatively…

Clock Number

Inst. #	1	2	3	4	5	6	7	8	9	10
LOAD	IF	ID	EX	MEM	WB					
Inst. i+1		IF	ID	EX	MEM	WB				
Inst. i+2			IF	ID	EX	MEM	WB			
Inst. i+3				stall	IF	ID	EX	MEM	WB	
Inst. i+4						IF	ID	EX	MEM	WB
Inst. i+5							IF	ID	EX	MEM
Inst. i+6								IF	ID	EX

- LOAD instruction "steals" an instruction fetch cycle which will cause the pipeline to stall.
- Thus, no instruction completes on clock cycle 8

What's the realistic solution?

- Answer: Add more hardware.
  - (especially for the memory access example – i.e. the common case)
    - CPI degrades quickly from our ideal '1' for even the simplest of cases…

DATA HAZARDS

Data hazards

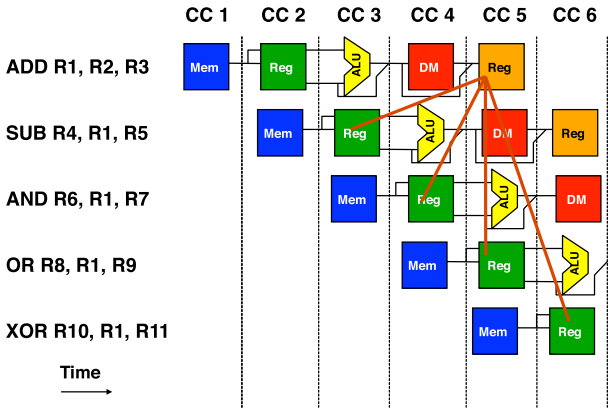
- These exist because of pipelining
- Why do they exist???
  - Pipelining changes when data operands are read, written
  - Order differs from order seen by sequentially executing instructions on un-pipelined machine
- Consider this example:
  - ADD R1, R2, R3
  - SUB R4, R1, R5
  - AND R6, R1, R7
  - OR R8, R1, R9
  - XOR R10, R1, R11

All instructions after ADD use result of ADD

ADD writes the register in WB but SUB needs it in ID.

This is a data hazard

Illustrating a data hazard

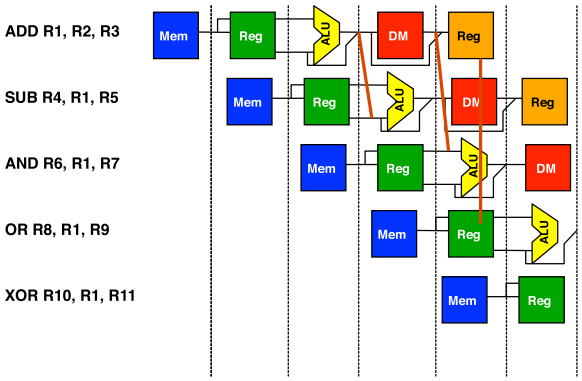


ADD instruction causes a hazard in next 3 instructions b/c register not written until after those 3 read it.

Forwarding

- Problem illustrated on previous slide can actually be solved relatively easily – with forwarding
- Can we move the result from EX/MEM register to the beginning of ALU (where SUB needs it)?
  - Yes!
- Generally speaking:
  - Forwarding occurs when a result is passed directly to functional unit that requires it.
  - Result goes from output of one unit to input of another

When can we forward?



SUB gets info. from EX/MEM pipe register

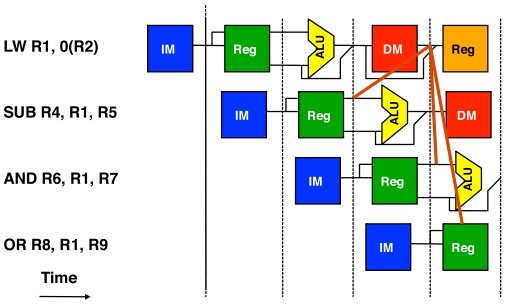
AND gets info. from MEM/WB pipe register

OR gets info. by forwarding from register file

Rule of thumb:

If line goes "forward" you can do forwarding.
If its drawn backward, it's physically impossible.

Forwarding doesn't always work

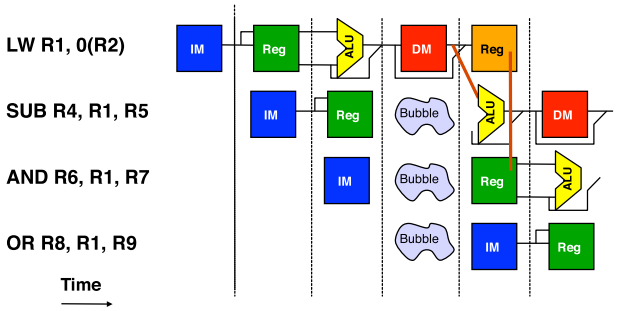


Load has a latency that forwarding can't solve.

Pipeline must stall until hazard cleared (starting with instruction that wants to use data until source produces it).

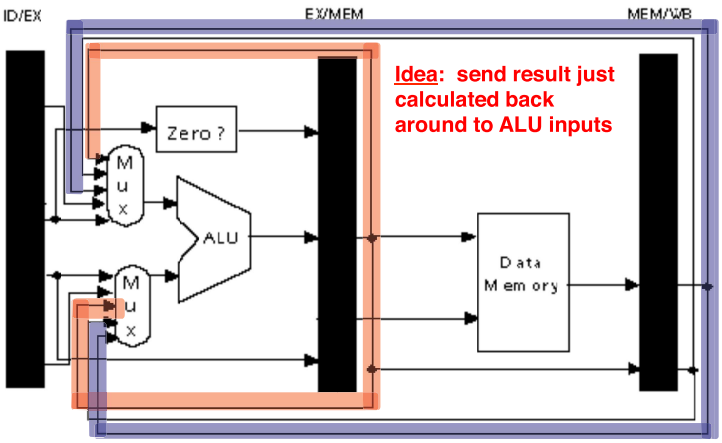
Can't get data to subtract b/c result needed at beginning of CC #4, but not produced until end of CC #4.

The solution pictorially



Insertion of bubble causes # of cycles to complete this sequence to grow by 1

HW Change for Forwarding



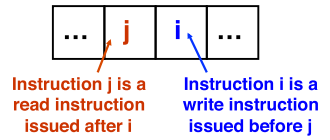
Send output of memory back to ALU too…

Data hazard specifics

- There are actually 3 different kinds of data hazards:
  - Read After Write (RAW)
  - Write After Write (WAW)
  - Write After Read (WAR)
- With an in-order issue/in-order completion machine, we're not as concerned with WAW, WAR

Read after write (RAW) hazards

- With RAW hazard, instruction j tries to read a source operand before instruction i writes it.
- Thus, j would incorrectly receive old or incorrect value
- Graphically/Example:



i: ADD R1, R2, R3
j: SUB R4, R1, R6

- Can use stalling or forwarding to resolve this hazard

Memory Data Hazards

- Seen register hazards, can also have memory hazards
  - RAW:
    - store R1, 0(SP)
    - load R4, 0(SP)

	1	2	3	4	5	6
Store R1, 0(SP)	F	D	EX	M	WB	
Load R1, 0(SP)		F	D	EX	M	WB

  - In simple pipeline, memory hazards are easy
    - In order, one at a time, read & write in same stage
  - In general though, more difficult than register hazards

Data hazards and the compiler

- Compiler should be able to help eliminate some stalls caused by data hazards
- i.e. compiler could not generate a LOAD instruction that is immediately followed by instruction that uses result of LOAD's destination register.

What about control logic?

- For MIPS integer pipeline, all data hazards can be checked during ID phase of pipeline
- If data hazard, instruction stalled before its issued
- Whether forwarding is needed can also be determined at this stage, controls signals set
- If hazard detected, control unit of pipeline must stall pipeline and prevent instructions in IF, ID from advancing

Some example situations

Situation	Example	Action

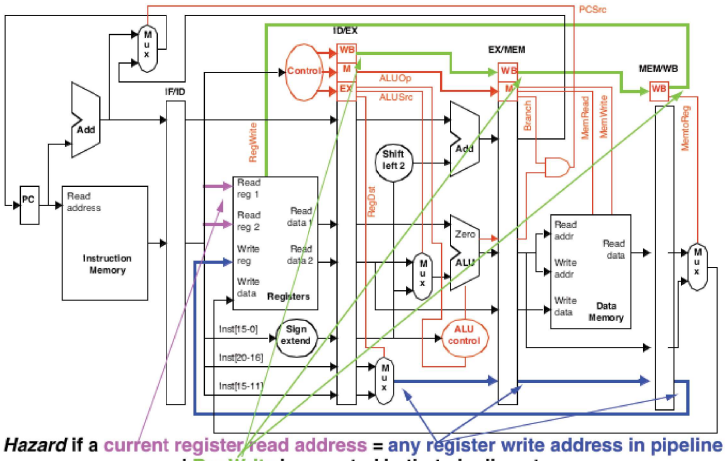
No Dependence	LW R1, 45(R2)
	ADD R5, R6, R7
	SUB R8, R6, R7
	OR R9, R6, R7
		No hazard possible because no dependence exists on R1 in the immediately following three instructions.

Dependence requiring stall	LW R1, 45(R2)
	ADD R5, R1, R7
	SUB R8, R6, R7
	OR R9, R6, R7
		Comparators detect the use of R1 in the ADD and stall the ADD (and SUB and OR) before the ADD begins EX

Dependence overcome by forwarding	LW R1, 45(R2)
	ADD R5, R6, R7
	SUB R8, R1, R7
	OR R9, R6, R7
		Comparators detect the use of R1 in SUB and forward the result of LOAD to the ALU in time for SUB to begin with EX

Dependence with accesses in order	LW R1, 45(R2)
	ADD R5, R6, R7
	SUB R8, R6, R7
	OR R9, R1, R7
		No action is required because the read of R1 by OR occurs in the second half of the ID phase, while the write of the loaded data occurred in the first half.

Detecting Data Hazards

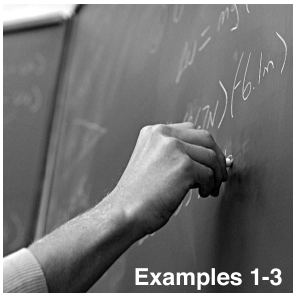


Hazard if a current register read address = any register write address in pipeline and RegWrite is asserted in that pipeline stage

Hazards vs. Dependencies

- dependence: fixed property of instruction stream
  - (i.e., program)
- hazard: property of program and processor organization
  - implies potential for executing things in wrong order
    - potential only exists if instructions can be simultaneously "in-flight"
    - property of dynamic distance between instructions vs. pipeline depth
- For example, can have RAW dependence with or without hazard
  - depends on pipeline

Examples…

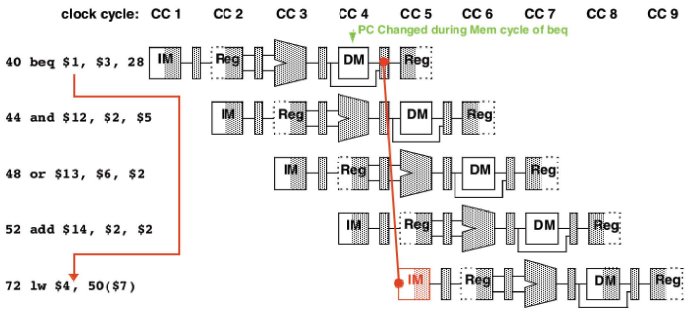


CONTROL HAZARDS

Branch / Control Hazards

- So far, we've limited discussion of hazards to:
  - Arithmetic/logic operations
  - Data transfers
- Also need to consider hazards involving branches:
  - Example:
    - 40: beq $1, $3, 28 # (28 leads to address 72)
    - 44: and $12, $2, $5
    - 48: or $13, $6, $2
    - 52: add $14, $2, $2
    - 72: lw $4, 50($7)
- How long will it take before the branch decision takes effect?
  - What happens in the meantime?

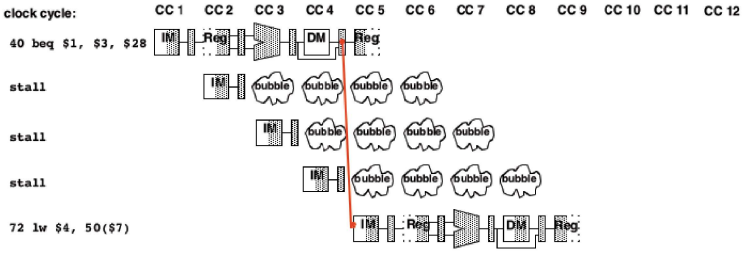
How branches impact pipelined instructions



- If branch condition true, must skip 44, 48, 52
  - But, these have already started down the pipeline
  - They will complete unless we do something about it
- How do we deal with this?
  - We'll consider 2 possibilities

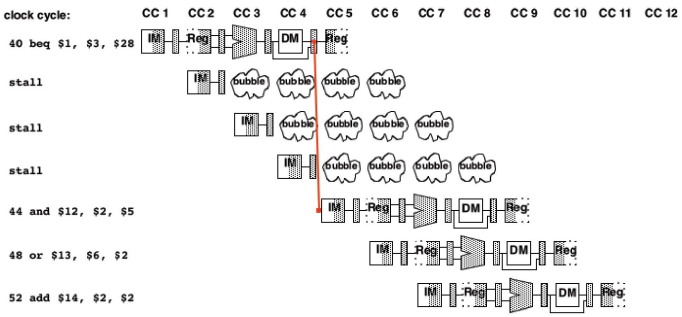
Dealing w/branch hazards: always stall

- Branch taken
  - Wait 3 cycles
  - No proper instructions in the pipeline
  - Same delay as without stalls (no time lost)



Dealing w/branch hazards: always stall

- Branch not taken
  - Still must wait 3 cycles
  - Time lost
  - Could have spent CCs fetching, decoding next instructions

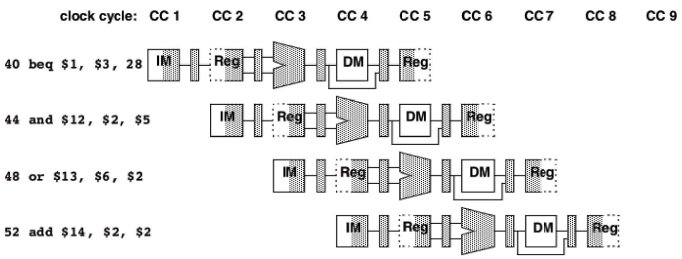


Dealing w/branch hazards

- On average, branches are taken ½ the time
  - If branch not taken…
    - Continue normal processing
  - Else, if branch is taken…
    - Need to flush improper instruction from pipeline
- One approach:
  - Always assume branch will NOT be taken
    - Cuts overall time for branch processing in ½
  - If prediction is incorrect, just flush the pipeline

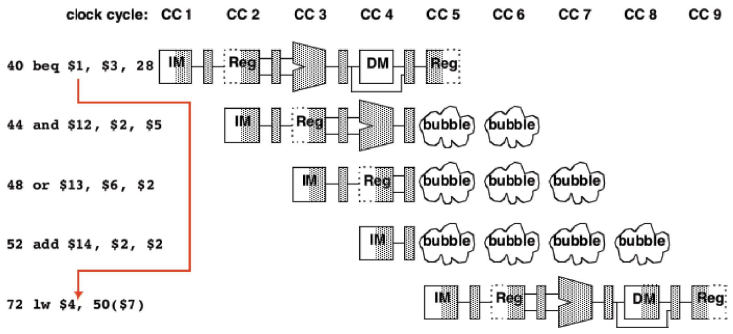
Impact of "predict not taken"

- Execution proceeds normally – no penalty

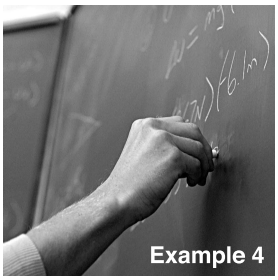


Impact of "predict not taken"

- Bubbles injected into 3 stages during cycle 5



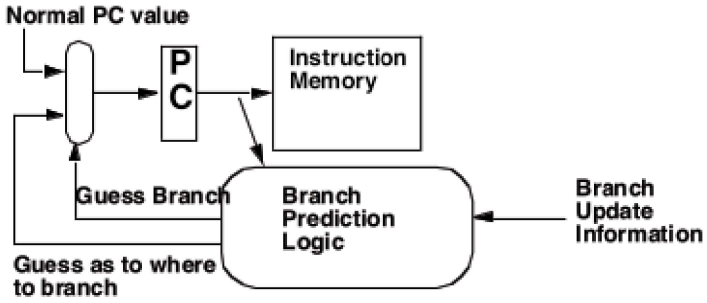
Branch Penalty Impact



Branch Prediction

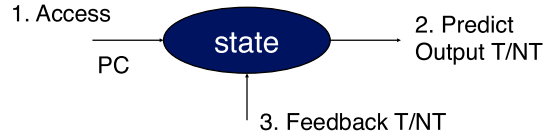
- Prior solutions are "ugly"
- Better (& more common): guess possible outcome
  - Technique is called "branch predicting"; needs 2 parts:
    - "Predictor" to guess where / if instruction will branch
      - (and to where)
    - "Recovery Mechanism":
      - i.e. a way to fix your mistake
  - Prior strategy:
    - Predictor: always guess branch never taken
    - Recovery: flush instructions if branch taken
  - Alternative: accumulate info. in IF stage as to…
    - Whether or not for any particular PC value a branch was taken next
    - To where it is taken
    - How to update with information from later stages

A Branch Predictor

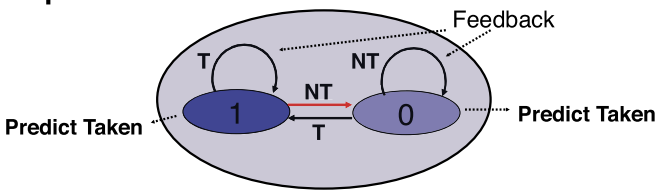


Predictor for a Single Branch

General Form



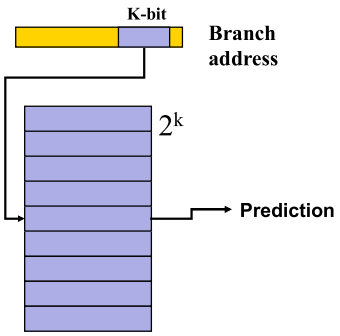
1-bit prediction



Branch History Table of 1-bit Predictor

BHT also called branch prediction buffer

- Can use only one 1-bit predictor, but accuracy is low
- BHT: use a table of simple predictors, indexed by bits from PC
- More entries, more cost, but less conflicts, higher accuracy
- BHT can contain complex predictors



1 bit weakness

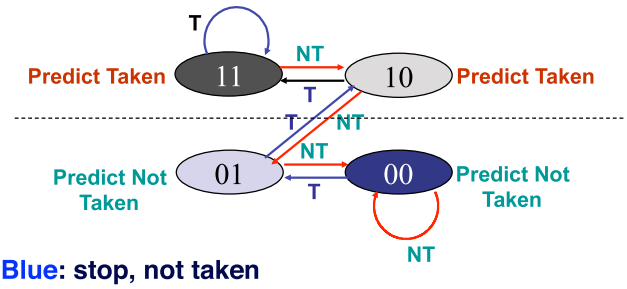
- Example:
  - in a loop, 1-bit BHT will cause 2 mispredictions
- Consider a loop of 9 iterations before exit:

```
for (…){
  for (i=0; i<9; i++)
    a[i] = a[i] * 2.0;
}
```

  - End of loop case, when it exits instead of looping as before
  - First time through loop on *next* time through code, when it predicts *exit* instead of looping
  - Only 80% accuracy even if loop 90% of the time

2-bit saturating counter

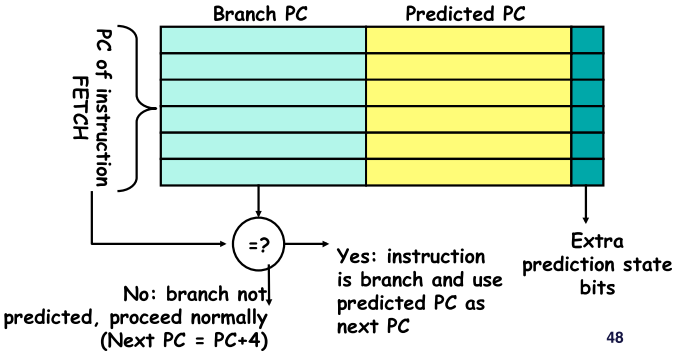
- Solution: 2-bit scheme where change prediction only if get misprediction *twice:*



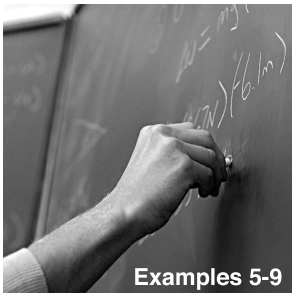
- Blue: stop, not taken
- Gray: go, taken
- Adds *hysteresis* to decision making process

Branch target buffer

- Branch Target Buffer (BTB): Address of branch index to get prediction AND branch address (if taken)
  - Note: must check for branch match now, since can't use wrong branch address
- Example: BTB combined with BHT



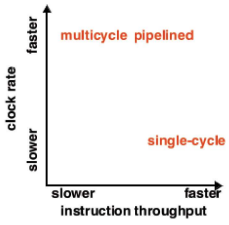
Examples…

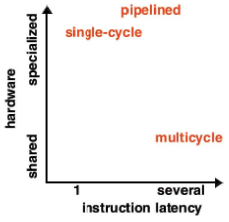


Discussion

- How does instruction set design impact pipelining?
- Does increasing the depth of pipelining always increase performance?

Comparative performance





- Throughput: instructions per clock cycle = 1/cpi
  - Pipeline has fast throughput and fast clock rate
- Latency: inherent execution time, in cycles
  - High latency for pipelining causes problems
    - Increased time to resolve hazards