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

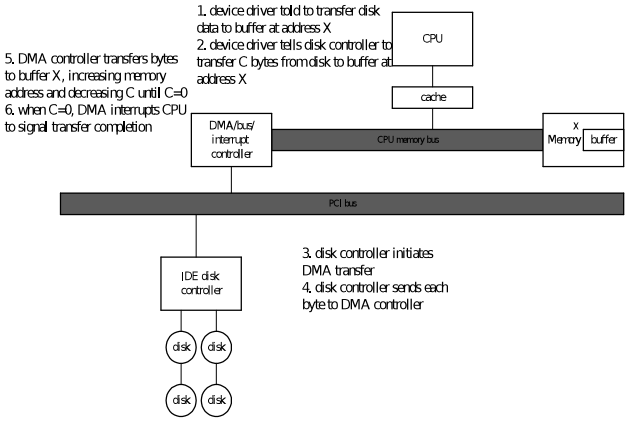
- J. L. Hennessy & D.A. Patterson, Computer architecture: A quantitative approach, Fourth Edition, Morgan Kaufman, 2004.

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

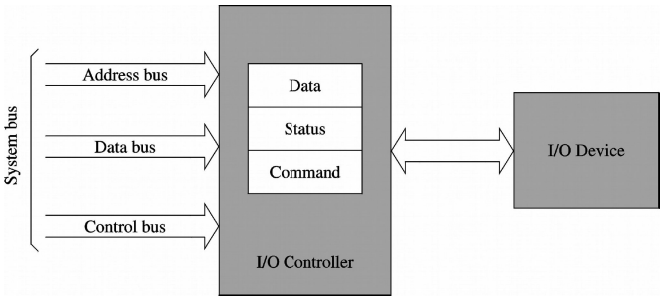
Outline

- 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)

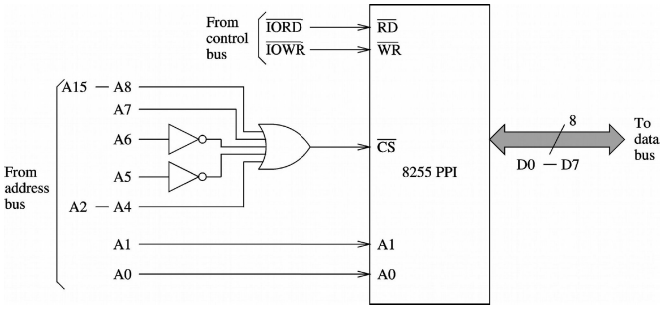
An Example I/O Device (cont'd)

- 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)

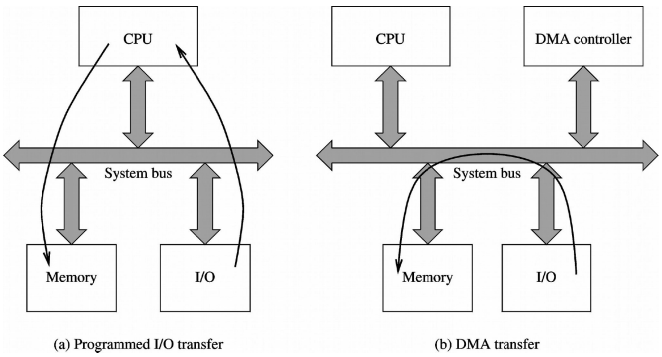
I/O Data Transfer (cont'd)

- 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)

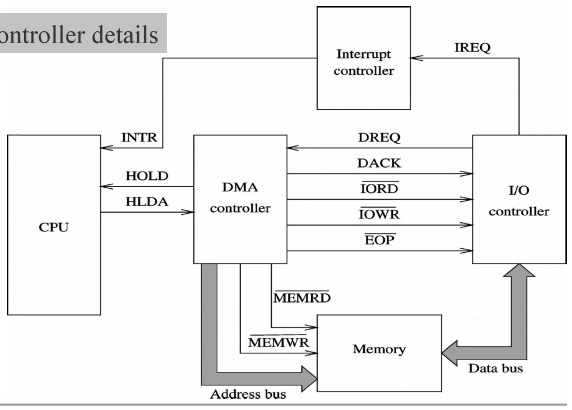
- DMA is implemented using a DMA controller
  - DMA controller
    - » Acts as slave to processor
    - » Receives instructions from processor
    - » Example: Reading from an I/O device
      - – Processor gives details to the DMA controller
        - →I/O device number
        - →Main memory buffer 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

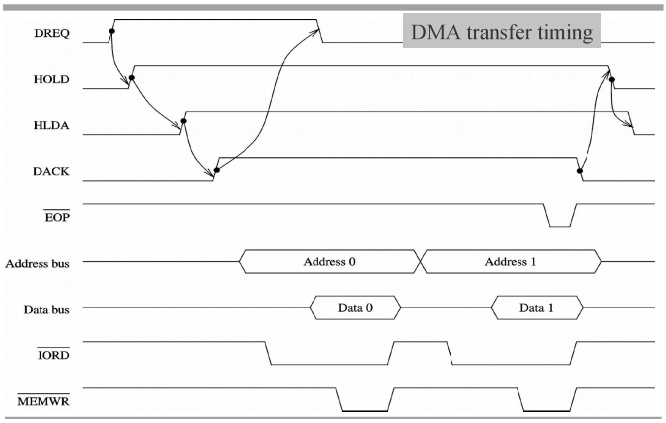
I/O Data Transfer (cont'd)

DMA controller details



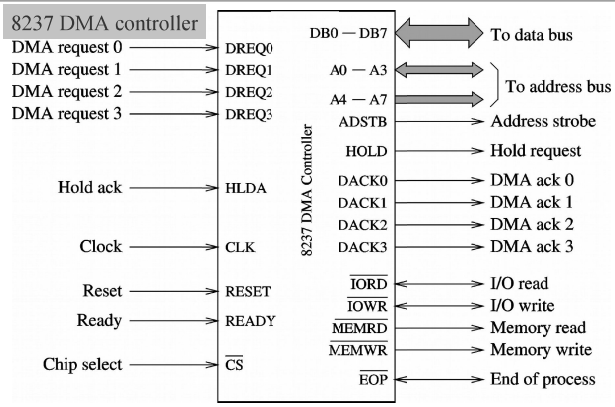
I/O Data Transfer (cont'd)

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

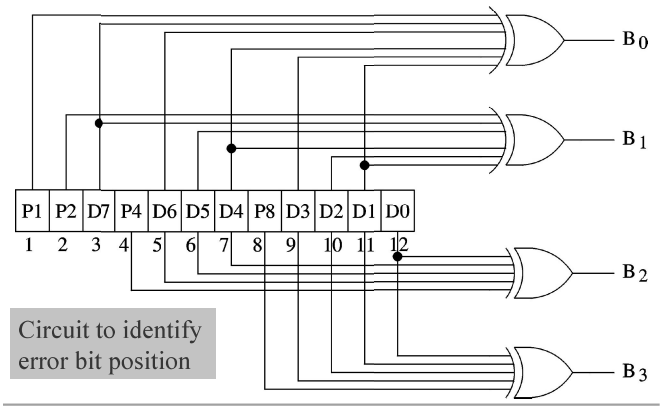
Error Detection and Correction (cont'd)

- 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 = 11th 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: `11000000000000101`
      - – 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

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

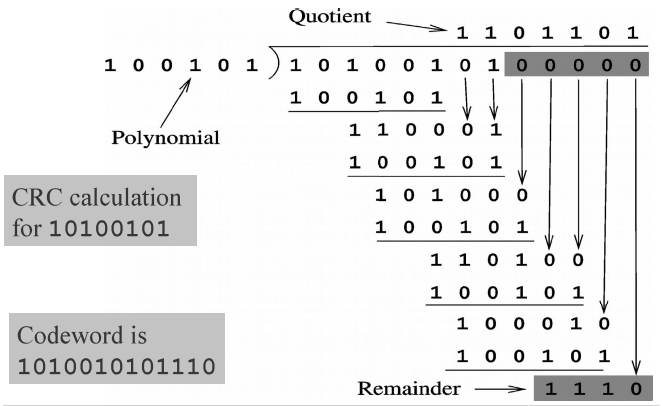
  - » From above, we get

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

zero

Error-free condition

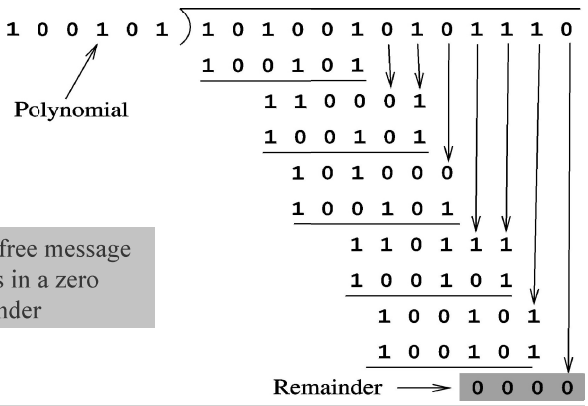
Error Detection and Correction (cont'd)



CRC calculation for `10100101`

Codeword is `1010010101110`

Error Detection and Correction (cont'd)

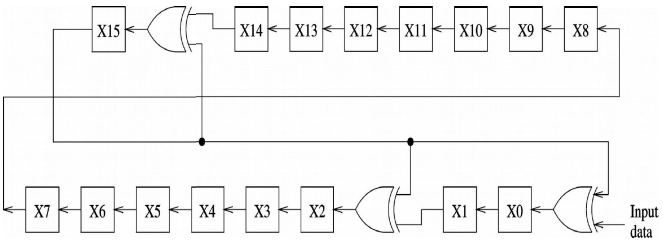


Error free message results in a zero remainder

Error Detection and Correction (cont'd)

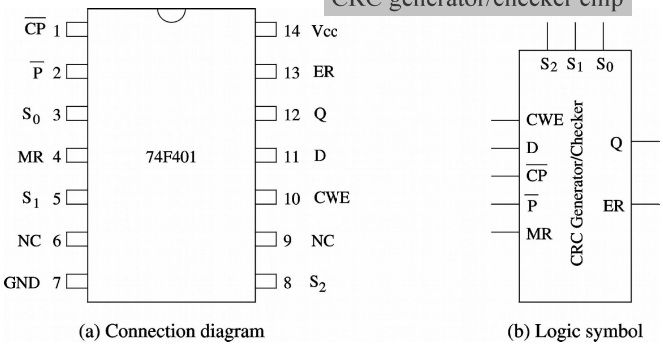
- A serial CRC generator circuit
  - Uses polynomial generator

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



Error Detection and Correction (cont'd)

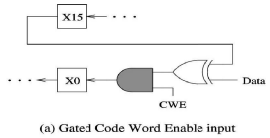
CRC generator/checker chip



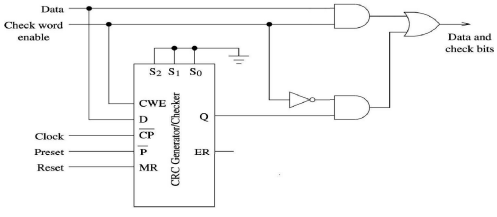
Error Detection and Correction (cont'd)

Using 74F401 to generate CRC for the generator

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



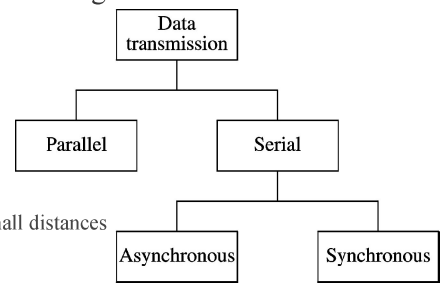
(a) Gated Code Word Enable input



(b) Check word generation

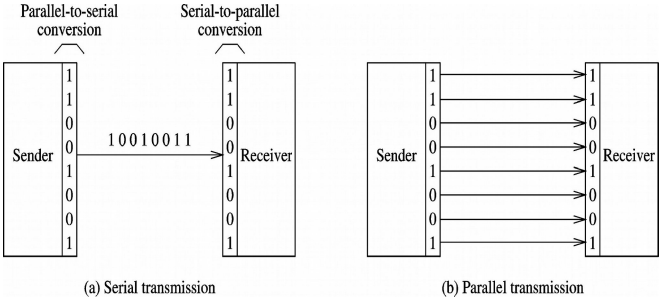
External Interface

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



External Interface (cont'd)

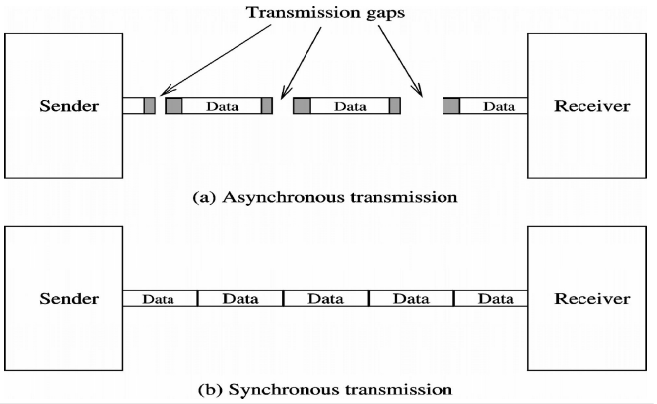
Two basic modes of data transmission



External Interface (cont'd)

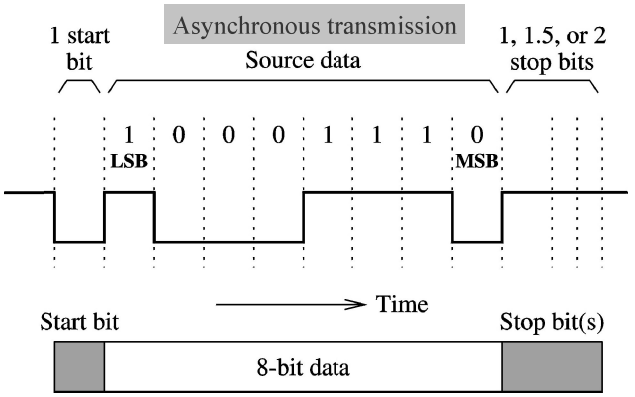
- Serial transmission
  - Asynchronous
    - » Each byte is encoded for transmission
      - – Start and stop bits
    - » No need for sender and receiver synchronization
  - Synchronous
    - » Sender and receiver must synchronize
      - – Done in hardware using phase locked loops (PLLs)
    - » Block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

External Interface (cont'd)



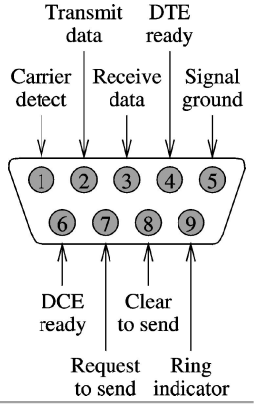
External Interface (cont'd)

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

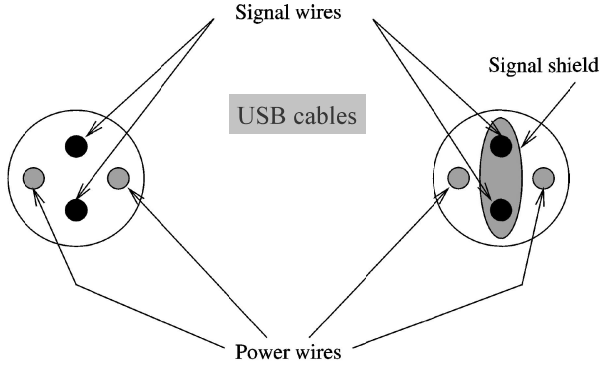
USB (cont'd)

- 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)

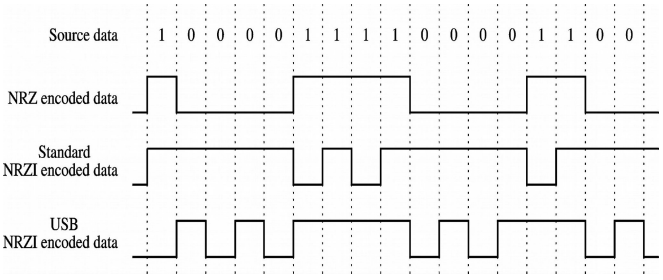
- Additional advantages of USB
  - Power distribution
    - » Simple devices can be bus-powered
      - – Examples: mouse, keyboards, floppy disk drives, wireless LANs, …
  - Control peripherals
    - » Possible because USB allows data to flow in both directions
  - Expandable through hubs
  - Power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

USB (cont'd)



USB (cont'd)

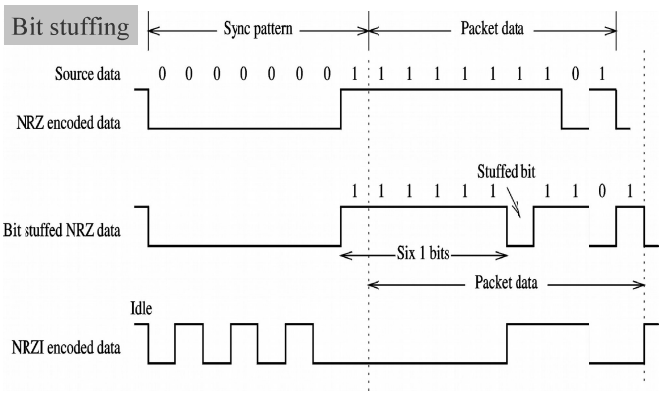
- USB encoding
  - Uses NRZI encoding
    - » Non-Return to Zero-Inverted



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)



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

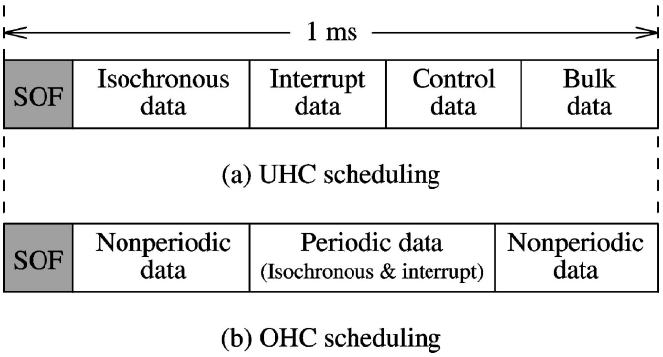
USB (cont'd)

- 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

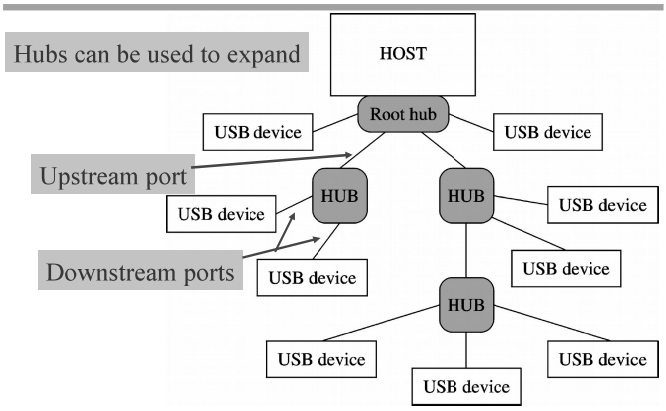
USB (cont'd)

- 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)

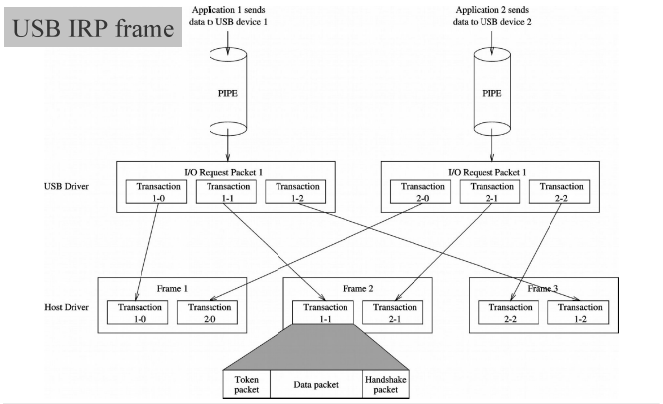


USB (cont'd)

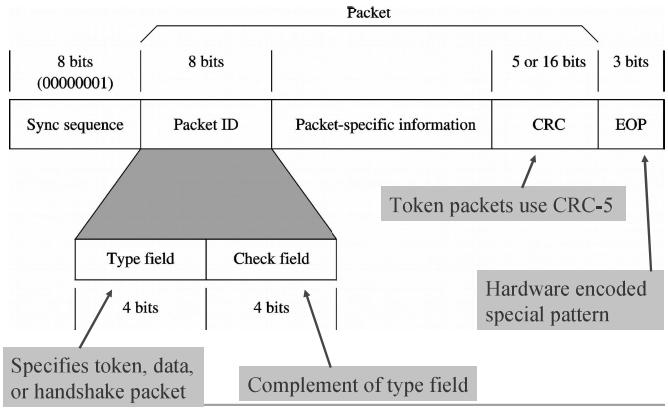
- USB transactions
  - Transfers are done in one or more transactions
    - » Each transaction consists of several packets
  - Transactions may have between 1 and 3 phases
    - » Token packet phase
      - – Specifies transaction type and target device address
    - » Data packet phase (optional)
      - – 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)

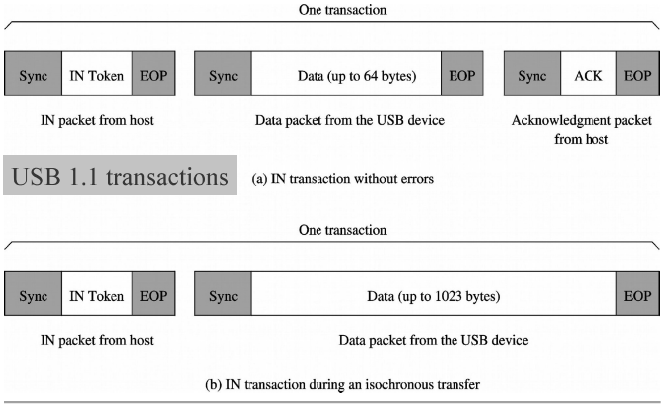
USB IRP frame



USB (cont'd)



USB (cont'd)



USB (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)

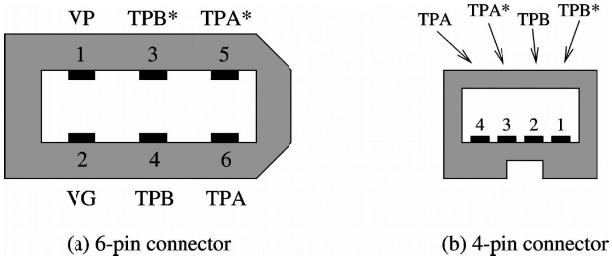
- Advantages
  - High speed
    - » Supports three speeds
      - – 100, 200, 400 Mbps
        - →Competes with USB 2.0
      - – Plans to boost it to 3.2 Gbps
  - Hot attachment
    - » Like USB
    - » No need to shut down power to attach devices
  - Peer-to-peer support
    - » USB is processor-centric
    - » 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

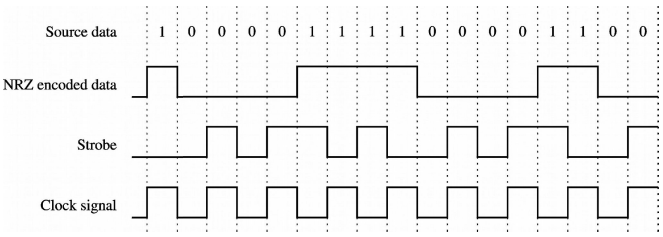


(a) 6-pin connector (b) 4-pin connector

4-pin connector does not distribute power

IEEE 1394 (cont'd)

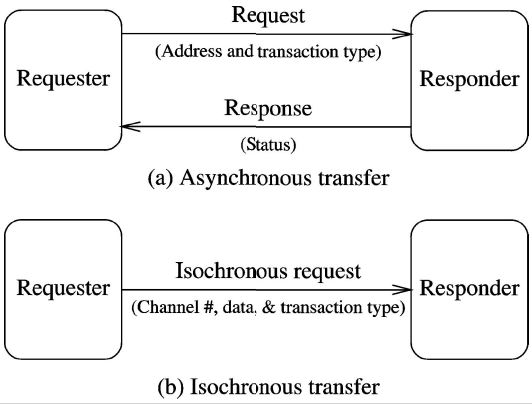
- Encoding
  - Uses a simple NRZ encoding
  - Strobe signal is encoded
    - » Changes the signal even if successive bits are the same



IEEE 1394 (cont'd)

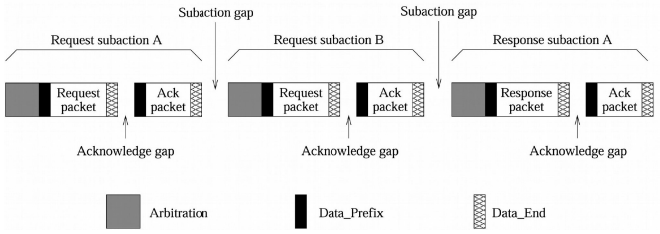
- Transfer types
  - Asynchronous
    - » For applications that require correct delivery of data
      - – Example: writing a file to a disk drive
    - » Uses an acknowledgement to confirm delivery
    - » Guaranteed bandwidth of 20%
  - Isochronous
    - » For real-time applications
    - » No acknowledgement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

IEEE 1394 (cont'd)



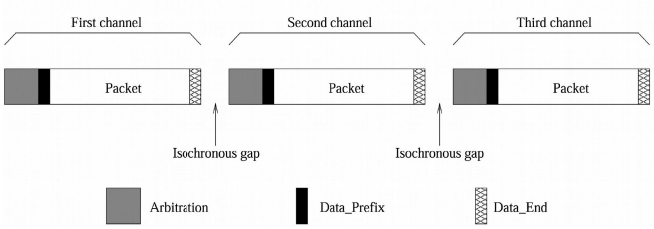
IEEE 1394 (cont'd)

- 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)

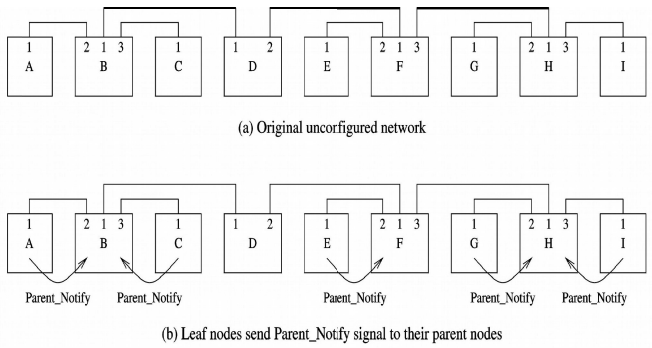
- Bus arbitration
  - Needed because of peer-to-peer communication
  - Arbitration must respect
    - » Bandwidth allocation to isochronous channels
    - » Fairness-based allocation for asynchronous channels
  - Uses fairness interval
    - » During each interval
      - – All nodes with pending asynchronous 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

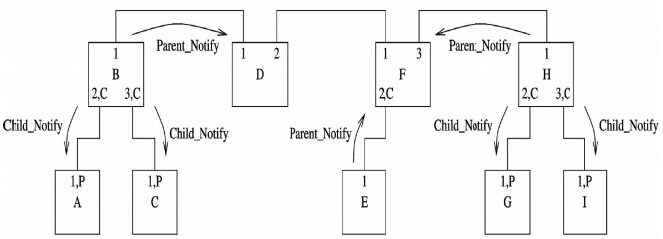
IEEE 1394 (cont'd)

Tree identification



IEEE 1394 (cont'd)

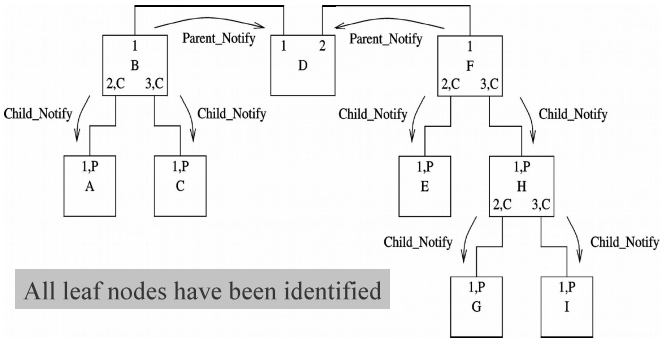
Tree identification



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

IEEE 1394 (cont'd)

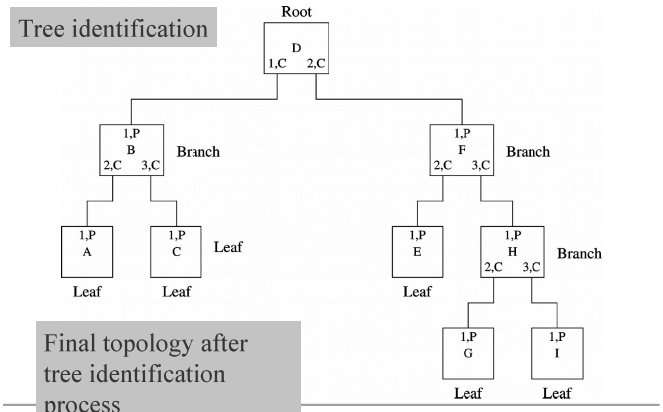
Tree identification



All leaf nodes have been identified

IEEE 1394 (cont'd)

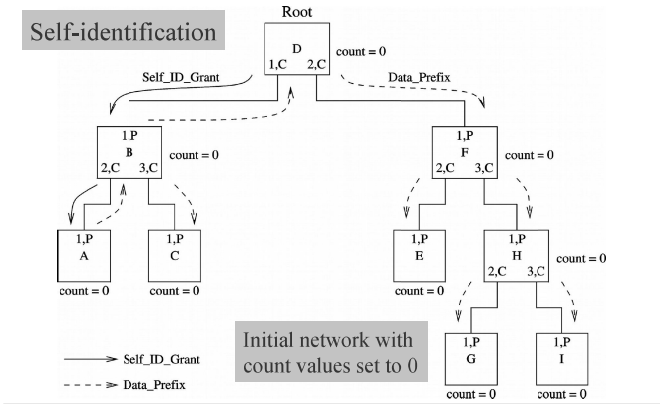
Tree identification



Final topology after tree identification process

IEEE 1394 (cont'd)

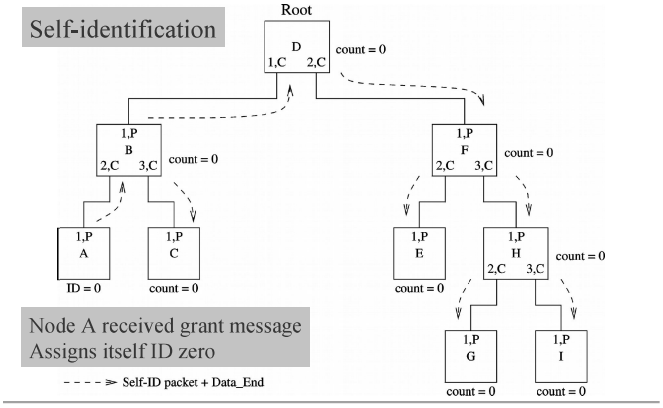
Self-identification



Initial network with count values set to 0

IEEE 1394 (cont'd)

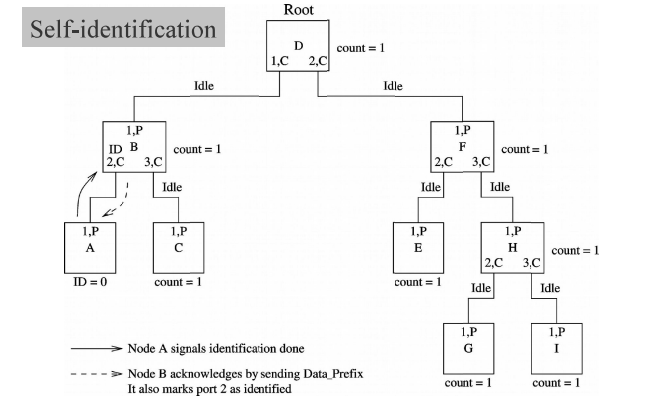
Self-identification



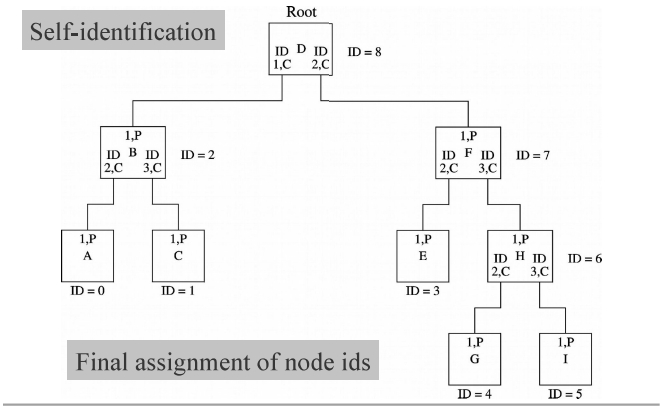
Node A received grant message
Assigns itself ID zero

IEEE 1394 (cont'd)

Self-identification



IEEE 1394 (cont'd)



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

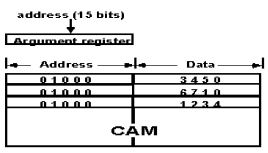
- Cache Hit
  - A referenced item is found in the cache by the processor
- Cache Miss
  - A referenced item is not present in the cache
- Hit ratio
  - Ratio of number of hits to total number of references => number of hits/(number of hits + number of Miss)
- Miss penalty
  - Additional cycles required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

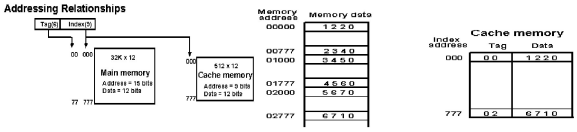
Associative Mapping

- 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

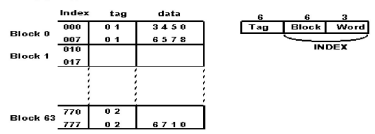
Disadvantage: hit ratio drops if 2 or more words with same index but different tags are accessed repeatedly.

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

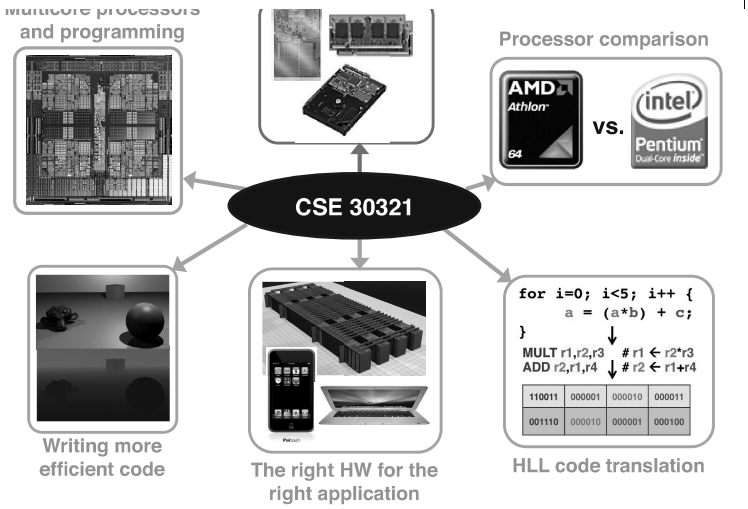
0.8*38ns + 0.2*74ns = 45.2ns

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)

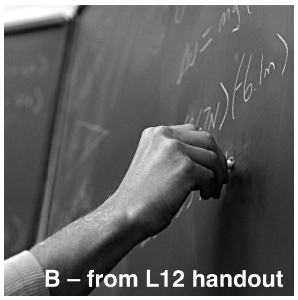
- **Pipelining changes the timing as to when the result(s) of an instruction are produced**
- **Additional HW is needed to ensure that the correct program results are produced while maintaining the speedups offered from the introduction of pipelining**
- **We must also account for the efficient pipelining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



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:

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

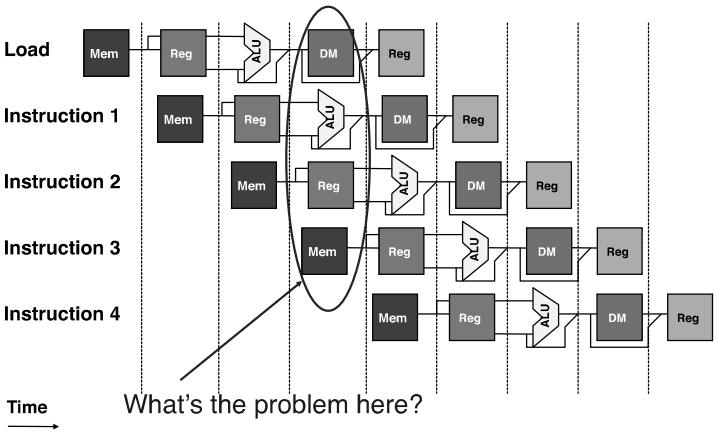
- If no stalls, speedup equal to # of pipeline stages in ideal case

STRUCTURAL HAZARDS

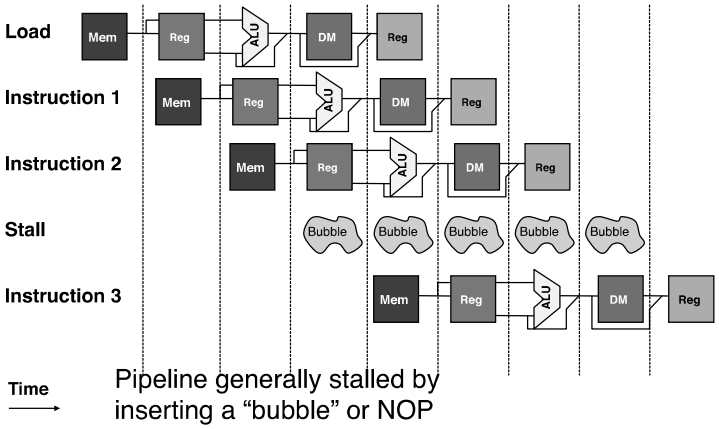
Structural hazards

- **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

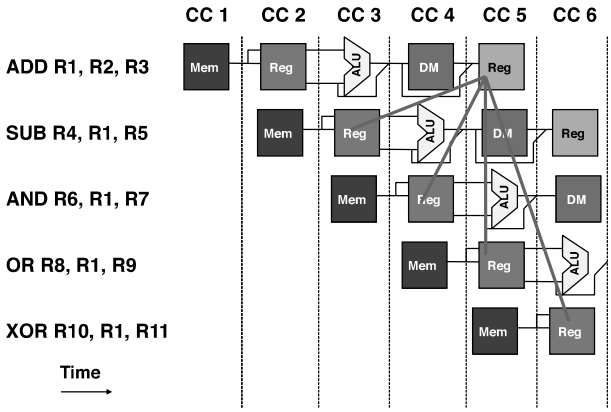
- These exist because of pipelining
- **Why do they exist???**
  - **Pipelining changes when data operands are read, written**
  - **Order differs from order seen by sequentially executing instructions on un-pipelined machine**
- **Consider this example:**
  - **ADD R1, R2, R3**
  - **SUB R4, R1, R5**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

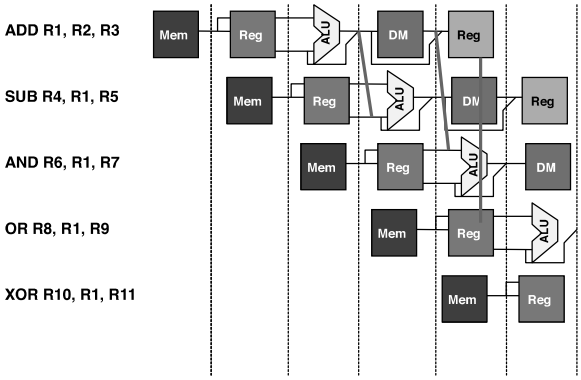


ADD instruction causes a hazard in next 3 instructions b/c register not written until after those 3 read it.

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

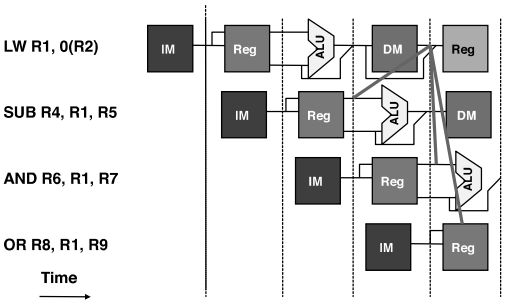
AND gets info. from MEM/WB pipe register

OR gets info. by forwarding from register file

Time

Rule of thumb: **If line goes "forward" you can do forwarding. If its drawn backward, it's physically impossible.**

Forwarding doesn't always work

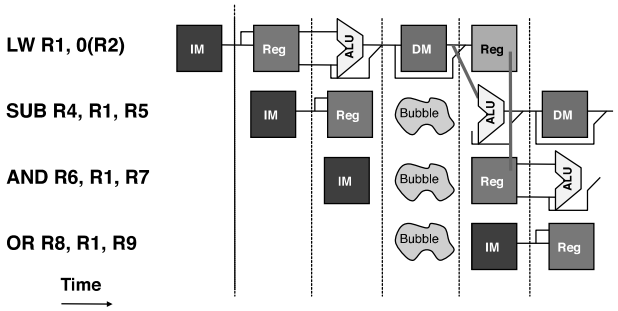


Load has a latency that forwarding can't solve.

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

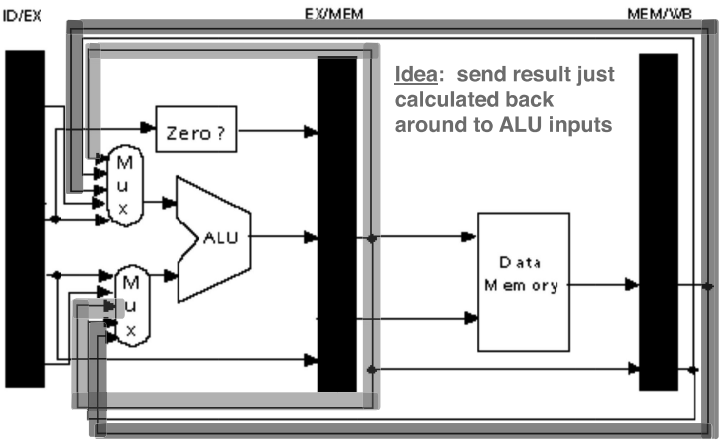
Can't get data to subtract b/c result needed at beginning of CC #4, but not produced until end of CC #4.

The solution pictorially



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

HW Change for Forwarding



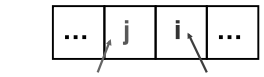
Send output of memory back to ALU too…

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:



Instruction j is a read instruction issued after i

Instruction i is a write instruction issued before j

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

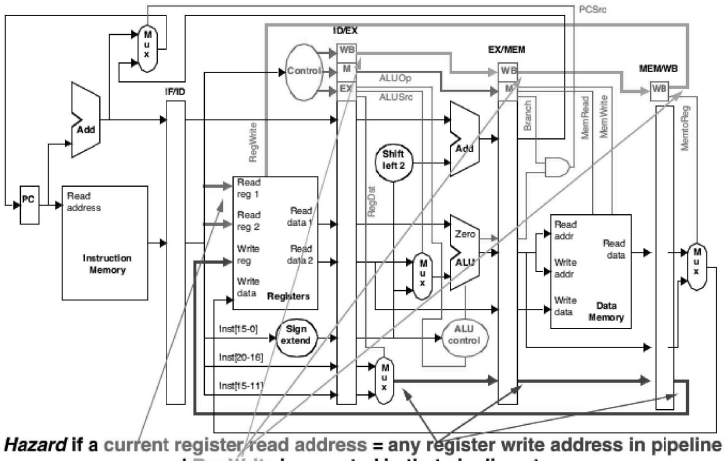
No Dependence	LW R1, 45(R2)
	ADD R5, R6, R7
	SUB R8, R6, R7
	OR R9, R6, R7
		No hazard possible because no dependence exists on R1 in the immediately following three instructions.

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

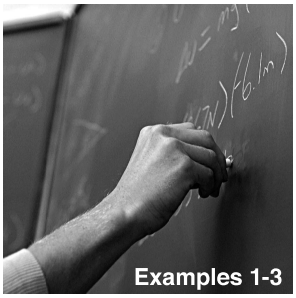


Hazard **if a current register read address = any register write address in pipeline and RegWrite is asserted in that pipeline stage**

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…

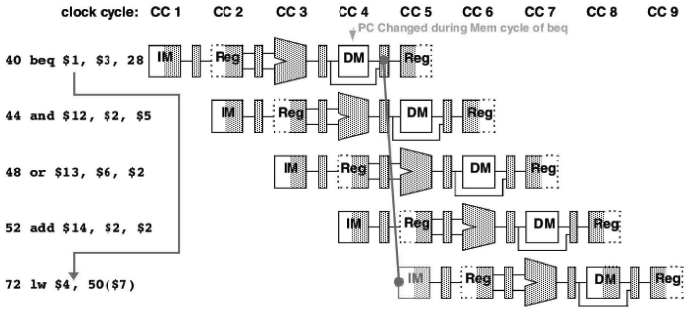


CONTROL HAZARDS

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?

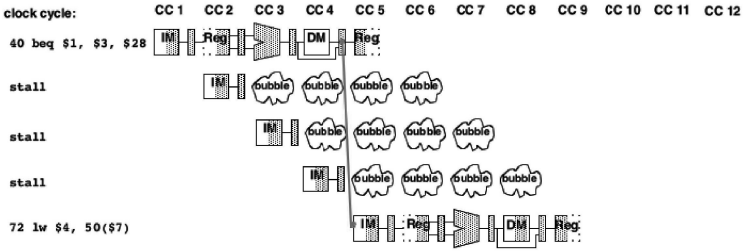
How branches impact pipelined instructions



- 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

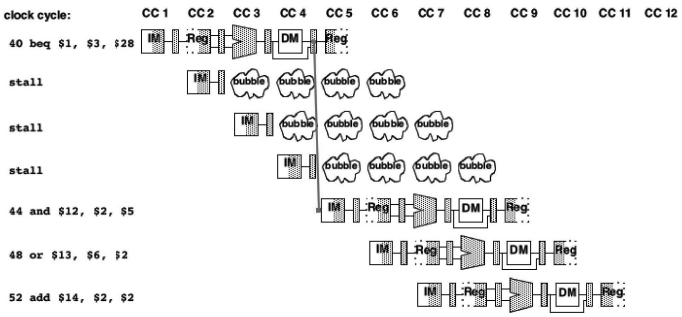
Dealing w/branch hazards: always stall

- 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

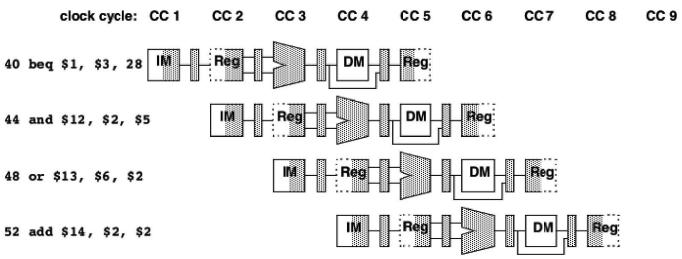


Dealing w/branch hazards

- 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

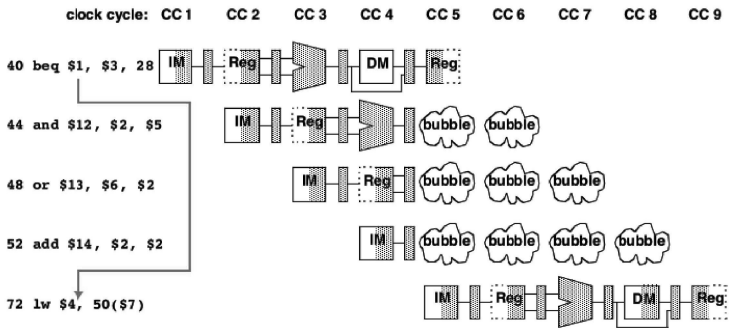
Impact of "predict not taken"

- Execution proceeds normally – no penalty

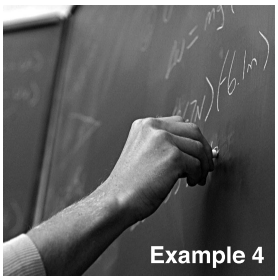


Impact of "predict not taken"

- Bubbles injected into 3 stages during cycle 5



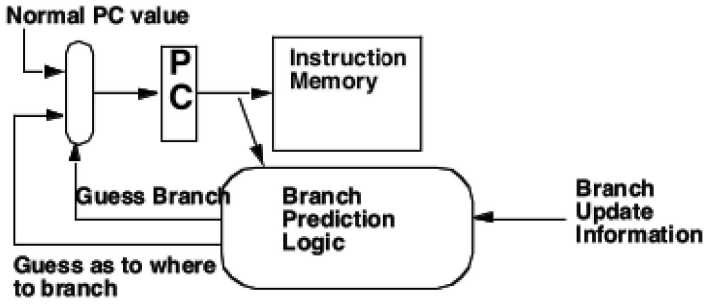
Branch Penalty Impact



Branch Prediction

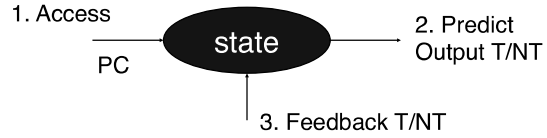
- Prior solutions are "ugly"
- Better (& more common): guess possible outcome
  - Technique is called "branch predicting"; needs 2 parts:
    - "Predictor" to guess where / if instruction will branch
      - (and to where)
    - "Recovery Mechanism":
      - i.e. a way to fix your mistake
  - Prior strategy:
    - 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

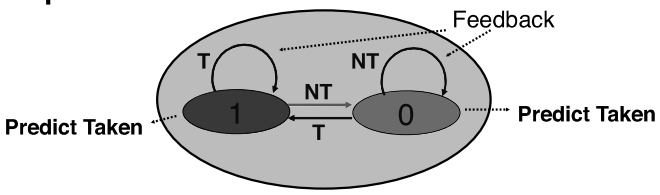


Predictor for a Single Branch

General Form



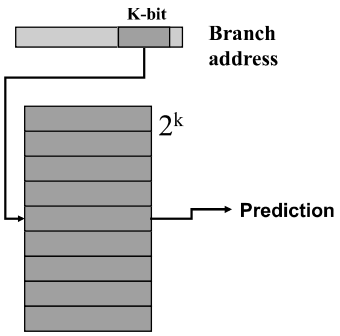
1-bit prediction



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

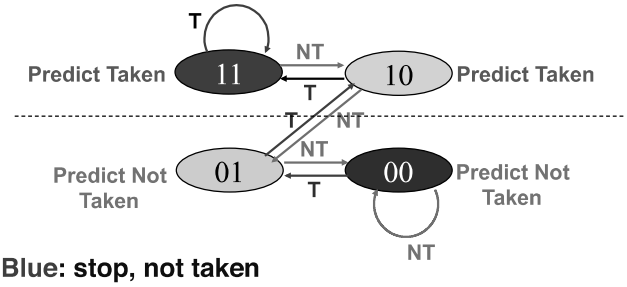
- Example:
  - in a loop, 1-bit BHT will cause 2 mispredictions
- Consider a loop of 9 iterations before exit:

```
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

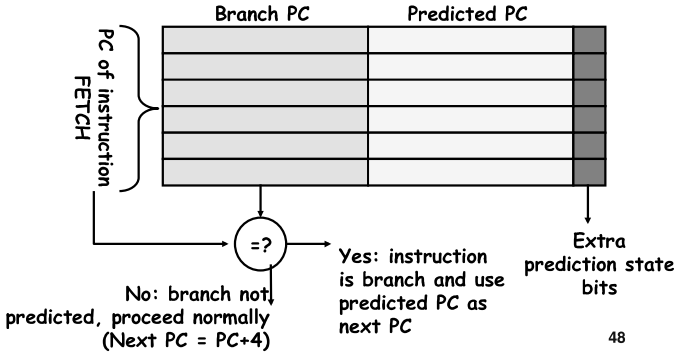
- Solution: 2-bit scheme where change prediction only if get misprediction *twice:*



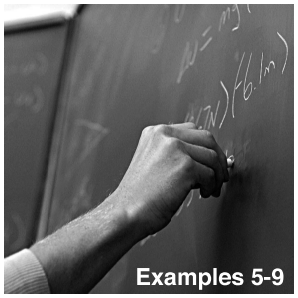
- Blue: stop, not taken
- Gray: go, taken
- Adds *hysteresis* to decision making process

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



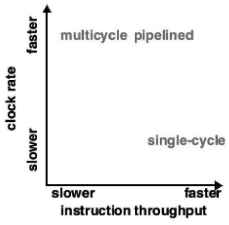
Examples…

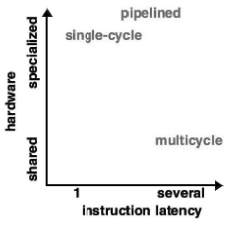


Discussion

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

Comparative performance



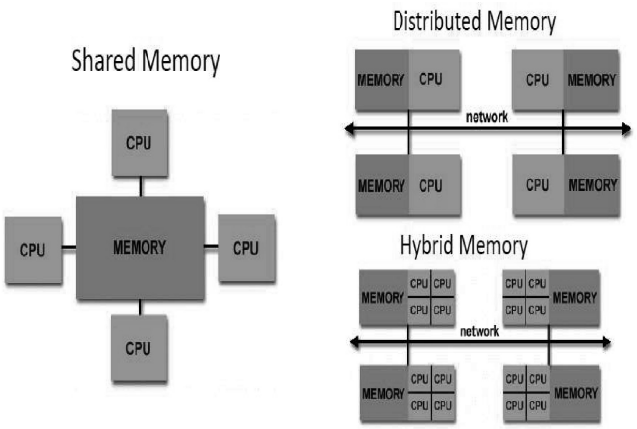


- Throughput: instructions per clock cycle = 1/cpi
  - Pipeline has fast throughput and fast clock rate
- Latency: inherent execution time, in cycles
  - High latency for pipelining causes problems
    - Increased time to resolve hazards

- Parallel computers are those that emphasize the parallel processing between the operations in some way.
- Various Varieties
  - Shared Memory
  - Distributed Memory
  - Hybrid Memory

Instruction Cycle

- Start
- Calculate the address of the instruction to be executed
- Fetch the instruction
- Decode the instruction
- Calculate the operand address
- Fetch the operands
- Execute the instructions
- Store the results
- If more instructions to be executed, go to step 2 else stop.



Flynn's Classification

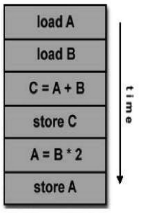
- SISD: Single Instruction Single Data
  - Classical Von-Neumann architecture
- SIMD: Single Instruction Multiple data
- MISD: Multiple Instructions Single Data
- MIMD: Multiple Instructions Multiple Data
  - Most common and general parallel machine

Flynn's Taxonomy

- This classification was first studied and proposed by Michael Flynn in 1972.
- Flynn did not consider the machine architecture for classification of parallel computers
- He introduced the concept of instruction and data streams for categorizing of computers.
- Instruction stream - a flow of instructions from main memory to the CPU
- Data stream - a flow of operands between processor and memory
- All the computers classified by Flynn are not parallel computers
- Let I_s and D_s are minimum number of streams flowing at any point in the execution

Single Instruction Single Data stream

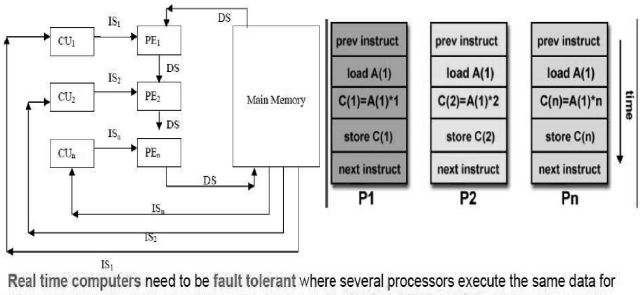
- A serial computer
- SISD machines are conventional serial computers that process only one stream of instructions and one stream of data.
- $I_s = D_s = 1$
- Examples
  - CDC 6600 which is unpipelined but has multiple functional units.
  - CDC 7600 which has a pipelined arithmetic unit.
  - Amdhal 470/6 which has pipelined instruction processing.
  - Cray-1 which supports vector processing.



Single Instruction Multiple Data stream

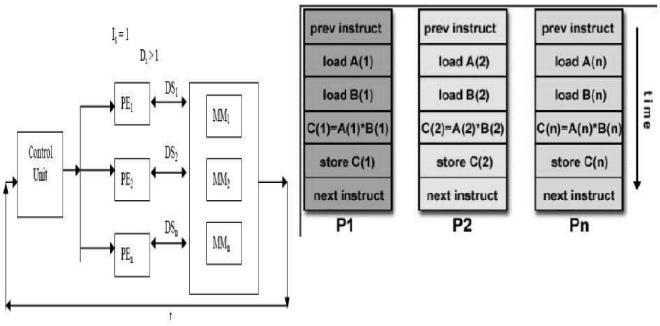
- Multiple processing elements work under the control of a single control unit.
- It has one instruction and multiple data stream
- Main memory can also be divided into modules for generating multiple data streams acting as a distributed memory
- Examples of SIMD organisation are ILLIAC-IV, PEPE, BSP, STARAN, MPP, DAP and the Connection Machine (CM-1).
- A type of parallel computer
- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element

MISD



Real time computers need to be fault tolerant where several processors execute the same data for producing the redundant data. This is also known as N- version programming. All these redundant data are compared as results which should be same; otherwise faulty unit is replaced. Thus MISD machines can be applied to fault tolerant real time computers.

SIMD



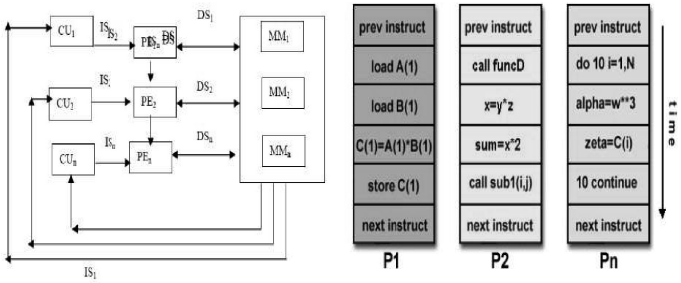
Multiple Instruction Multiple Data stream

- Multiple instruction streams operate on multiple data streams
- The processors work on their own data with their own instructions.
- Tasks executed by different processors can start or finish at different times.
- This classification actually recognizes the parallel computer.
- Examples include; C.mmp, Burroughs D825, Cray-2, S1, Cray X-MP, HEP, Pluribus, IBM 370/168 MP, Univac 1100/80, Tandem/16, IBM 3081/3084, C.m*, BBN Butterfly, Meiko Computing Surface (CS-1), FPS T/40000, iPSC.
- MIMD organization is the most popular for a parallel computer.
- In the real sense, parallel computers execute the instructions in MIMD mode.
- $I_s > 1$, $D_s > 1$

Multiple Instruction Single Data stream

- Multiple processing elements are organised under the control of multiple control units.
- Each control unit is handling one instruction stream and processed through its corresponding processing element.
- A single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently via independent instruction streams.
- But each processing element is processing only a single data stream at a time
- All processing elements are interacting with the common shared memory.

MIMD



What is RAID

- Redundant Array of Independent (Inexpensive) Disks
- A set of disk stations treated as one logical station
- Data are distributed over the stations
- Redundant capacity is used for parity allowing for data repair

Parity

- Way to do error checking and correction
- Add up all the bits that are 1
  - if even number, set parity bit to 0
  - if odd number, set parity bit to 1
- To actually implement this, do an exclusive OR of all the bits being considered
- Consider the following 2 bytes

byte	parity
10110011	1
01101010	0

- If a single bit is bad, it is possible to correct it

RAID (Redundant Array of Independent Disks)

- Redundant array of inexpensive disks
- Multiple disk database design
- Set of physical disk drives viewed by the OS as a single logical drive
- Data are distributed across the physical drives of an array
- Improve access time and improve reliability
  - large storage capacity
  - redundant data
  - 7 levels (6 levels in common use)

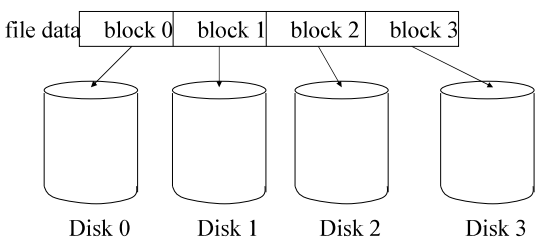
  - differing levels of redundancy, error checking, capacity, and cost

Mirroring

- Keep two copies of data on two separate disks
- Gives good error recovery
  - if some data is lost, get it from the other source
- Expensive
  - requires twice as many disks
- Write performance can be slow
  - have to write data to two different spots
- Read performance is enhanced
  - can read data from file in parallel

Striping

- Take file data and map it to different disks
- Allows for reading data in parallel



Levels of RAID

- 6 levels of RAID (0-5) have been accepted by the industry
- Other kinds have been proposed in literature
- Level 2 and 4 are not commercially available, they are included for clarity

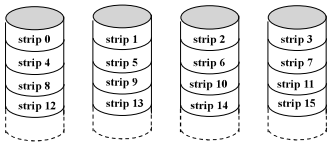
RAID 0

- All data (user and system) are distributed over the disks so that there is a reasonable chance for parallelism
- Disk is logically a set of strips (blocks, sectors,…). Strips are numbered and assigned consecutively to the disks (see picture.)

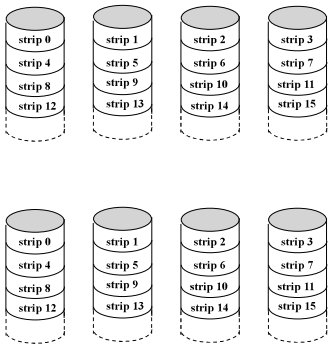
RAID 0

- Block level stripping
- Without parity or mirroring
- No redundancy, No backup, No fault tolerance
- Improved performance, faster, as it is uses block level stripping
- Maximum use of storage space as there is no backup
- Any drive failure destroys the array

Raid 0 (No redundancy)



Raid 1 (mirrored)



Data mapping Level 0

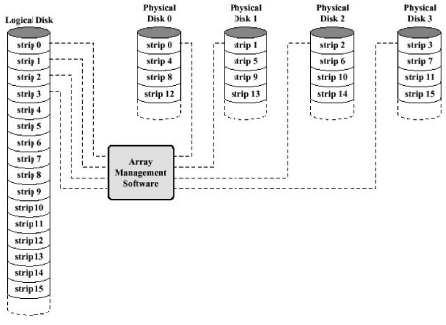


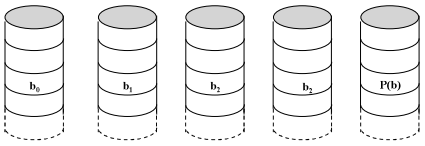
Figure 11.9 Data Mapping for a RAID Level 0 Array [MASS94]

RAID 1

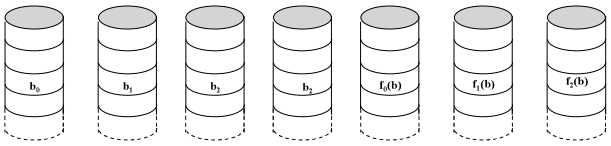
- RAID 1 does not use parity, it simply mirrors the data to obtain reliability
- Plus:
  - Reading request can be served by any of the two disks containing the requested data (minimum search time)
  - Writing request can be performed in parallel to the two disks: no "writing penalty"
  - Recovery from error is easy, just copy the data from the correct disk
- Minus:
  - Price for disks is doubled
  - Will only be used for system critical data that must be available at all times

- Mirroring
- Without parity
- Data is written identically to two drives (parallel write)
- Not slower, not faster (Write)
- Read can be faster by parallel access
- Fault tolerance
- 50% storage space can be used

RAID 3 (bit-interleaved parity)



Raid 2 (redundancy through Hamming code)



RAID 3

- Level 2 needs $\log_2$(number of disks) parity disks
- Level 3 needs only one, for one parity bit
- In case one disk crashes, the data can still be reconstructed even on line ("reduced mode") and be written (X1-4 data, P parity):

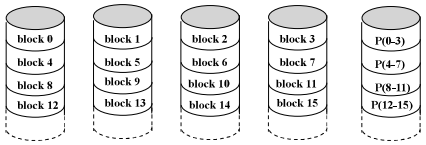
  P = X1+X2+X3+X4

  X1=P+X2+X3+X4
- RAID 2-3 have high data transfer times, but perform only one I/O at the time so that response times in transaction oriented environments are not so good

RAID 2

- Small strips, one byte or one word
- Synchronized disks, each I/O operation is performed in a parallel way
- Error correction code (Hamming code) allows for correction of a single bit error
- Controller can correct without additional delay
- Is still expensive

RAID 4 (block-level parity)



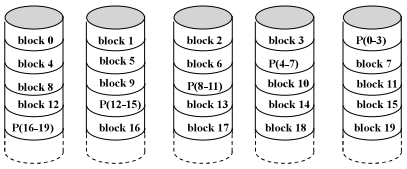
RAID 4

- Larger strips and one parity disk
- Blocks are kept on one disk, allowing for parallel access by multiple I/O requests
- Writing penalty: when a block is written, the parity disk must be adjusted (e.g. writing on X1):

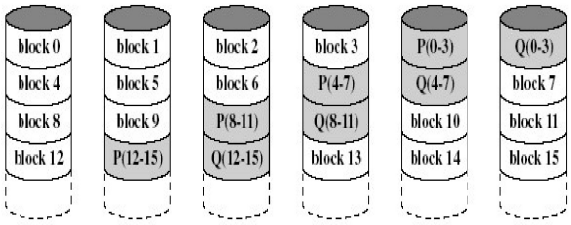
  P =X4+X3+X2+X1
  P'=X4+X3+X2+X1'
  =X4+X3+X2+X1'+X1+X1
  =P+X1+X1'
- Parity disk may be a bottleneck
- Good response times, less transfer rates

- Faster- Block level stripping
- Fault tolerance by Distributed parity (backup of data)
- Distribution of the parity strip to avoid the bottle neck.
- The array is not destroyed by the failure of single hard drive
- Minimum three disks
- Loose one hard disk space for parity
- 75% space

RAID 5 (block-level distributed parity)



RAID 6



RAID 5

- Distribution of the parity strip to avoid the bottle neck.

RAID Level 6

- Two different parity calculations are carried out and stored in separate blocks on different disks.
  - Example: XOR and an independent data check algorithm => makes it possible to regenerate data even if two disks containing user data fail.
- No. of disks required = N + 2 (where N = number of disks required for data).
- Provides HIGH data availability.
- Incurs substantial write penalty as each write affects two parity blocks.
- Three disks would have to fail within MTTR (mean time to repair) interval to cause data to be lost

- Faster-Block level stripping
- Provides fault tolerance up to two failed drives Fault tolerance- double distributed parity (backup of backup of data)
- High availability systems
- Failure of hard drive, slow down the performance of the system
- Loose two hard disk space for parity
- Incurs substantial write penalty

Overview Raid 0-2

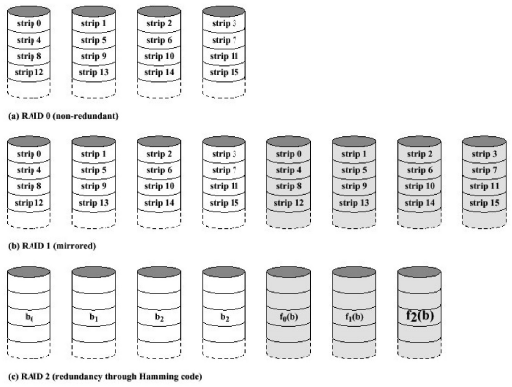


Figure 11.8 RAID Levels (page 1 of 2)

Overview Raid 3-5

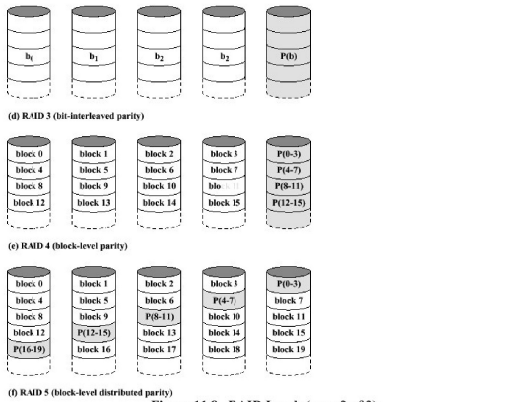


Figure 11.8 RAID Levels (page 2 of 2)