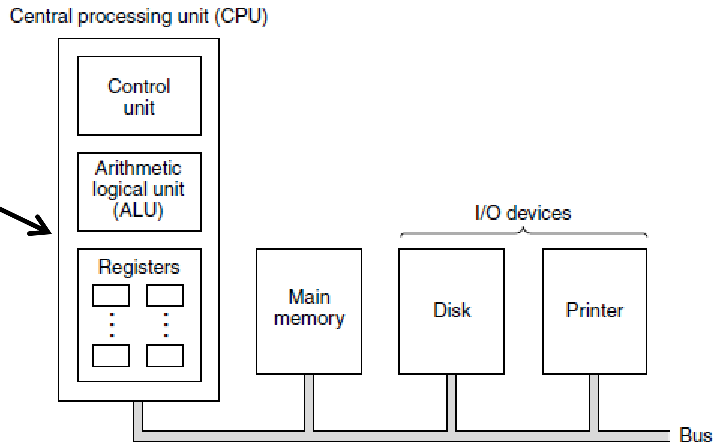
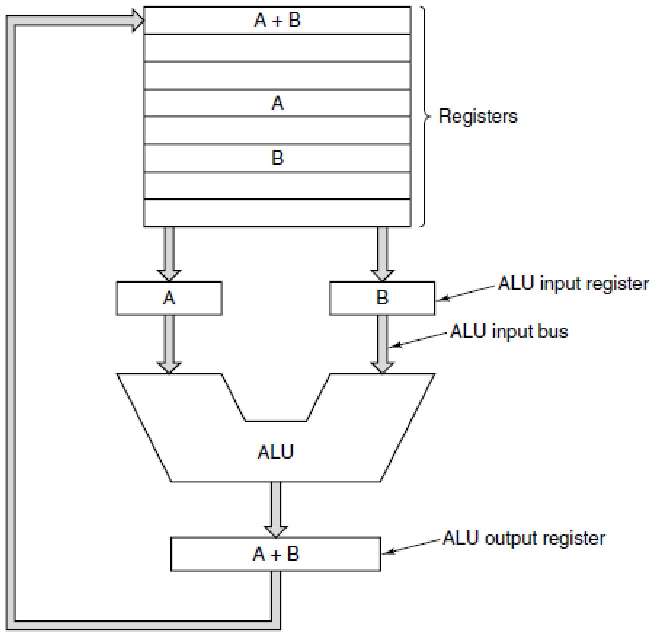
<u>Last lecture recap</u> Processors

CPU:
- Registers
  e.g. PC, IR
- Arithmetic Logistical Unit (ALU)
  e.g. NOT, OR, AND, and so on
- Control unit

Internal bus



External bus



Type of instructions:
- Register-Memory
- Register-Register

Last lecture recap Processors

Instruction Execution

Fetch-Decode-Execute cycle

Machine codes

Human readable language

Memory

```
0ADF:0300  32 C0 AA C3 BE 08 98 BF-CE 92 B4 60 CD 21 BE 77   2..........`.!.w
0ADF:0310  92 BF CE 92 E8 CB E3 C3-33 C0 89 3E 32 99 A2 35   ........3..>2..5
0ADF:0320
```

Instructions

```
0ADF:0200 A10003       MOV     AX,[0300]
0ADF:0203 89C2         MOV     DX,AX
0ADF:0205 050400       ADD     AX,0004
```

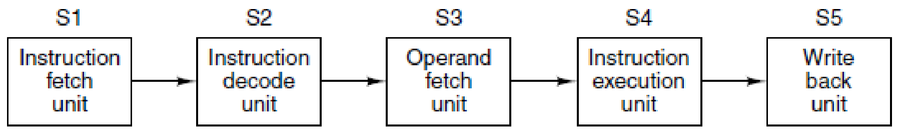
Execution

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0ADF  ES=0ADF  SS=0ADF  CS=0ADF  IP=0200   NV UP EI PL NZ NA PO NC
0ADF:0200 A10003        MOV     AX,[0300]                       DS:0300=C032

AX=C032  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0ADF  ES=0ADF  SS=0ADF  CS=0ADF  IP=0203   NV UP EI PL NZ NA PO NC
0ADF:0203 89C2          MOV     DX,AX

AX=C032  BX=0000  CX=0000  DX=C032  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0ADF  ES=0ADF  SS=0ADF  CS=0ADF  IP=0205   NV UP EI PL NZ NA PO NC
0ADF:0205 050400        ADD     AX,0004

AX=C036  BX=0000  CX=0000  DX=C032  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0ADF  ES=0ADF  SS=0ADF  CS=0ADF  IP=0208   NV UP EI NG NZ NA PE NC
0ADF:0208 0480          ADD     AL,80
```

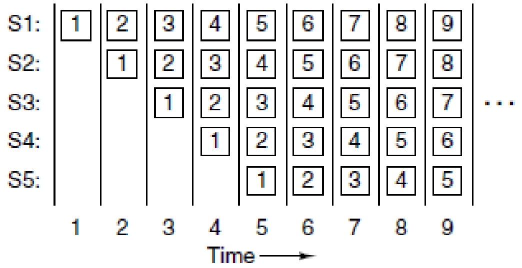
Last lecture recap Processors

Instruction-Level Parallelism
Pipelining

Pipelining



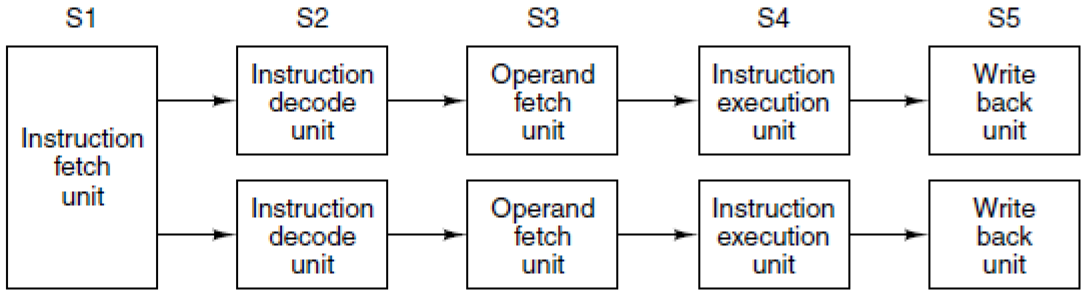
(a)



(b)

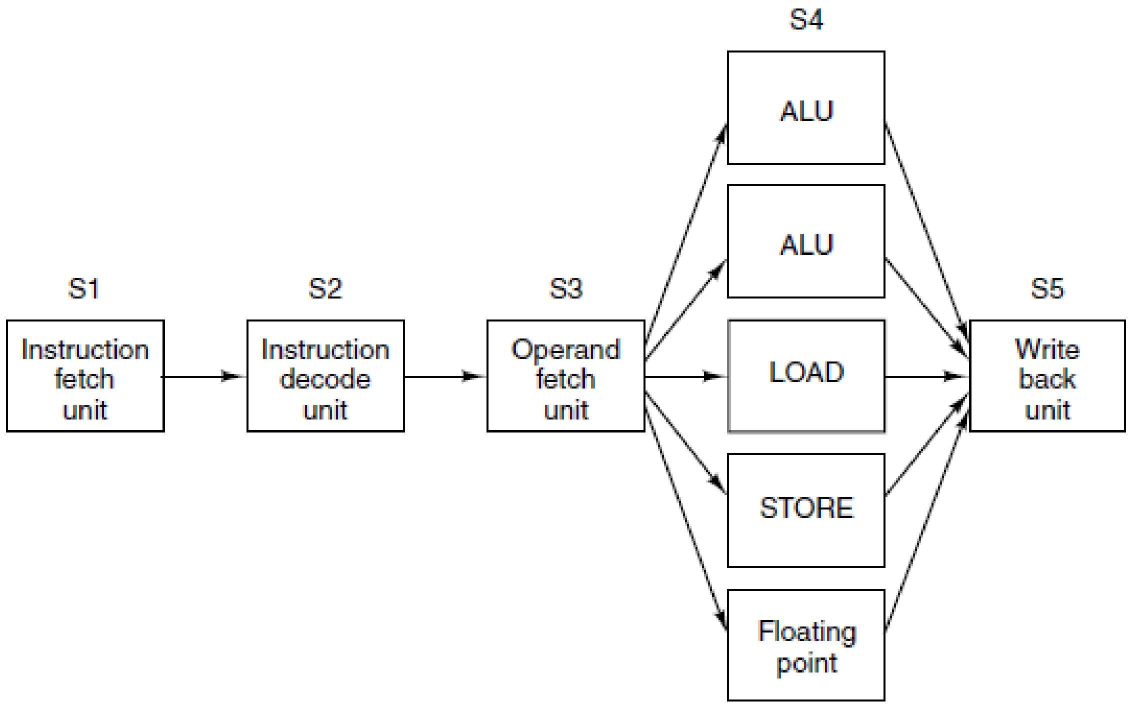
Superscalar Architectures

Dual five-stage pipeline



Last lecture recap Processors

Instruction-Level Parallelism



COMPUTER SYSTEMS ORGANIZATION

Processors
Primary Memory
Secondary Memory
Input/Output

PRIMARY MEMORY

Bits

- Memory is part of computer
- Programs and data are stored in memory
- The basic unit of memory is **bit {0, 1}**
- Other Units

    1 byte = 8 bits

    word = 1, 2, .. Byte

    e.g. 32 bits -> word = 4 bytes

    64 bits -> word = 8 bytes

PRIMARY MEMORY

Memory Addresses

Memories contains a number of cells

Cell or location

- Each cell can store a piece of information
- Each cell has a number called its address
- A cell with k bits can hold one of 2^k different bit combinations
  e.g. k = 3, 2^3 = 8, 000, 001, 010, 011, 100, 101, 110, 111

- A memory with n cells will have addresses 0 to n-1
- Adjacent cells have consecutive addresses (by definition)
- If an address has m bits, the maximum number of cells addressable is 2^m

- The number of bits in the address determines the maximum number of directly addressable cells in the memory and independent of the number of bits per cell

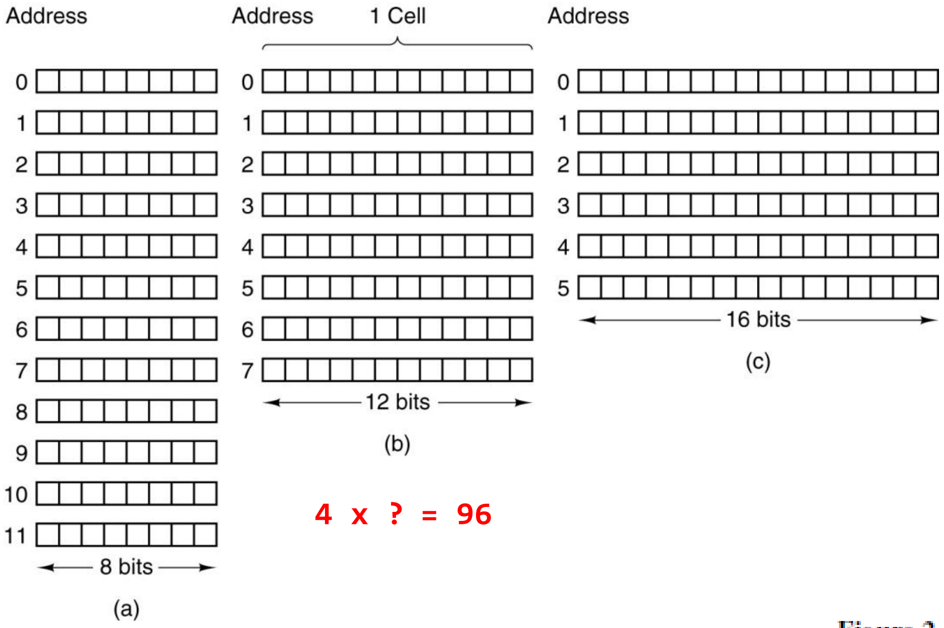
PRIMARY MEMORY

Memory Addresses

Memory size defined by cell size and number of cells

Three ways of organizing a 96-bit memory

12 x 8 = 96   8 x 12 = 96   6 x 16 = 96



4 x ? = 96

Computer	Bits/cell
Burroughs B1700	1
IBM PC	8
DEC PDP-8	12
IBM 1130	16
DEC PDP-15	18
XDS 940	24
Electrologica X8	27
XDS Sigma 9	32
Honeywell 6180	36
CDC 3600	48
CDC Cyber	60

Figure 2-10. Number of bits per cell for some historically interesting commercial computers.

PRIMARY MEMORY

Byte ordering

Little Endian vs. Big Endian

Storing data in a 32 bits memory (4 bytes)

260_{10} = 0X104 = 0X00000104 = $0x16^7$+ … + $1x16^2$ + $0x16^1$ + $4x16^0$

Most significant ← Least significant

Option 1	00	00	01	04
Option 2	04	01	00	00
Memory address	16	17	18	19

PRIMARY MEMORY

Byte ordering

Little Endian vs. Big Endian

Big Endian (most significant byte stored first)

- First most significant byte in smallest address
- second most significant byte in second smallest address
- . . . . . .

Little Endian (least significant byte stored first)

- First least significant byte in smallest address
- second least significant byte in second smallest address
- . . . . . .

Big Endian	00	00	01	04
Little Endian	04	01	00	00
Memory address	16	17	18	19

PRIMARY MEMORY

Byte ordering

Little Endian vs. Big Endian

Example (32 bits):

$$0x12345678 = 1x16^7 + \ldots + 6x16^2 + 7x16^1 + 8x16^0$$

Big Endian	12	34	56	78
Little Endian	78	56	34	12
Memory address	0x00401	0x00402	0x00403	0x00404

PRIMARY MEMORY

Error detections and corrections

- **Why Error Correcting Codes**
  1. Computer memories can make errors due to voltage spikes or others
  2. Error Correcting Codes are used to guard against such errors
- **Codeword**

  An n-bit unit containing m data and r check bits where $n = m + r$
- **Hamming distance between two Codewords**

  Defined as the number of bit positions in which two Codewords differ

Example: What is Hamming distance between 10101001 and 11001101 ?

Hamming distance = 3

10101001

11001101

PRIMARY MEMORY

Error detections and corrections
Hamming code

- **Error Correcting Properties**

1. To detect **s** single-bit errors, you need a distance **d = s + 1** code Why? because with such a code there is no way that s single-bit errors can change a valid codeword into another valid codeword.
2. To correct **s** single-bit errors, you need a distance **d = 2s+1** code.

PRIMARY MEMORY

Error detections and corrections
Hamming code

d = 2

	Data word	Code word		
0	000	0000	0000	0
1	001	0011	0001	1
2	010	0101	0010	2
3	011	0110	0011	3
4	100	1001	0100	4
5	101	1010	0101	5
6	110	1100	0110	6
7	111	1111	0111	7
			1000	8
			1001	9
			1010	10
			1011	11
			1100	12
			1101	13
			1110	14
			1111	15

Green color = valid
Red color = Not valid

Example:

single-bit change
0110 --> 1110 (14 not valid and detectable)

two-bit change
0110 --> 1100 (12 valid but not detectable)

PRIMARY MEMORY

Error detections and corrections
Hamming code

Example:

A code scheme has a Hamming distance d = 5. What is the error detection and correction capability of this scheme?

Error detection capability : $d = s + 1 \rightarrow s = 5-1 = 4$
Error correction capability : $d = 2s + 1 \rightarrow s = (5-1)/2 = 2$

PRIMARY MEMORY

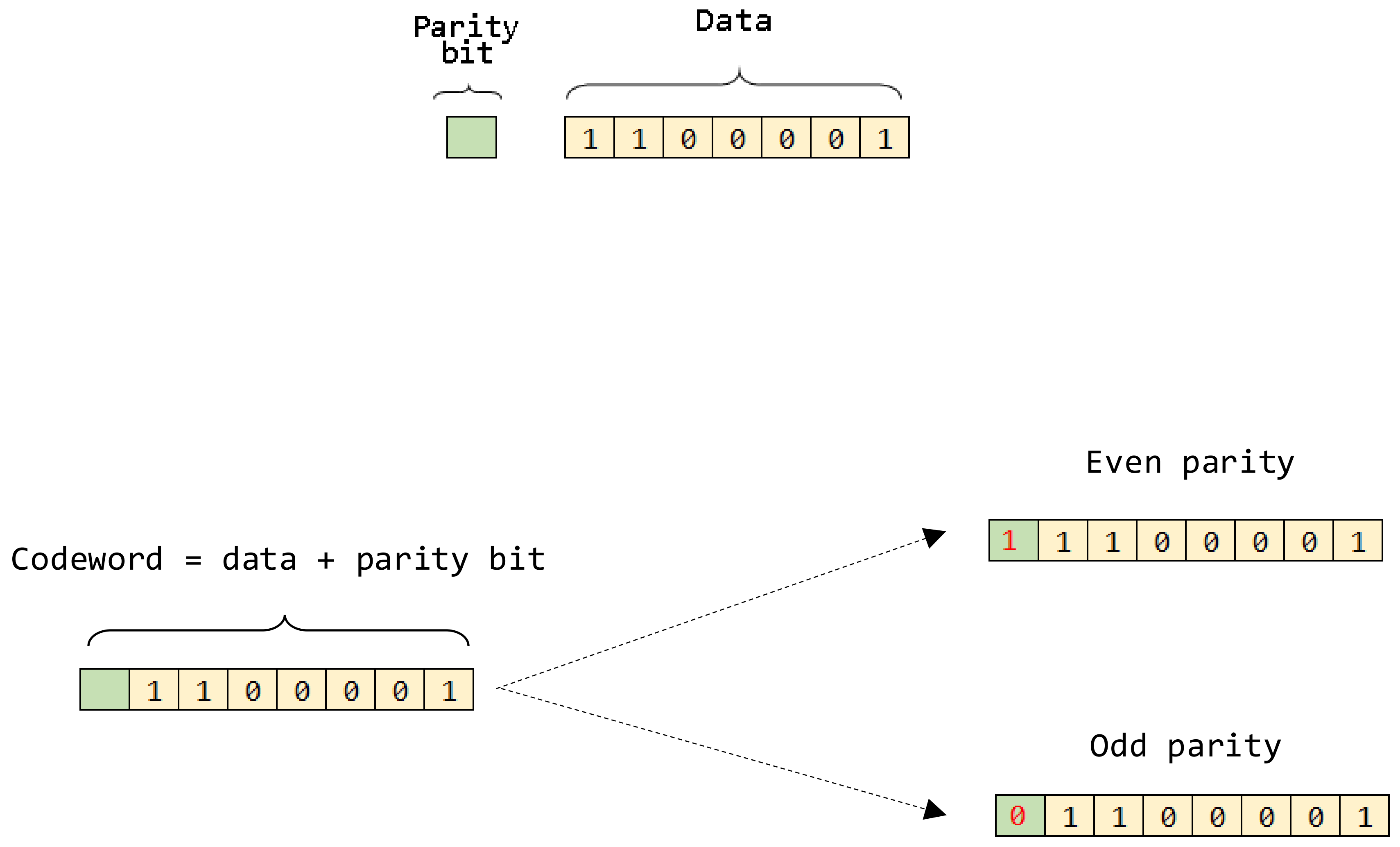
Error detections and corrections

Parity bit
Cyclic Redundancy Check (CRC)
Hamming code

Error detection
Parity bit

A **parity bit**(or **check bit)**, is a single bit added a group of bits. It is set to either **1** or **0** to make the total number of **1**-bits either even ("even parity") or odd ("odd parity").

Data is transferred from source to destination, so an error is occurred if a parity change is observed.


Parity bit
Data
1 1 0 0 0 0 1
Codeword = data + parity bit
1 1 0 0 0 0 1
Even parity
1 1 1 0 0 0 0 1
Odd parity
0 1 1 0 0 0 0 1

PRIMARY MEMORY

Error detections
Parity bit

Example:

Even Parity	BCD		Odd Parity	BCD	
0	0 0 0 0	00000	1	0 0 0 0	10000
1	0 0 0 1	10001	0	0 0 0 1	00001
1	0 0 1 0	10010	0	0 0 1 0	00010
0	0 0 1 1	00011	1	0 0 1 1	10011
1	0 1 0 0	10100	0	0 1 0 0	00100
0	0 1 0 1	00101	1	0 1 0 1	10101
0	0 1 1 0	00110	1	0 1 1 0	10110
1	0 1 1 1	10111	0	0 1 1 1	00111
1	1 0 0 0	11000	0	1 0 0 0	01000
0	1 0 0 1	01001	1	1 0 0 1	11001

PRIMARY MEMORY

Error detections
Parity bit

Example: What type of parity is used? Find if any error is occured?

10101 10110 00111 01000 11001	Odd or even Parity?	10101 10110 00101 01000 11000

Source → Destination

01000
11001

Writing

Reading

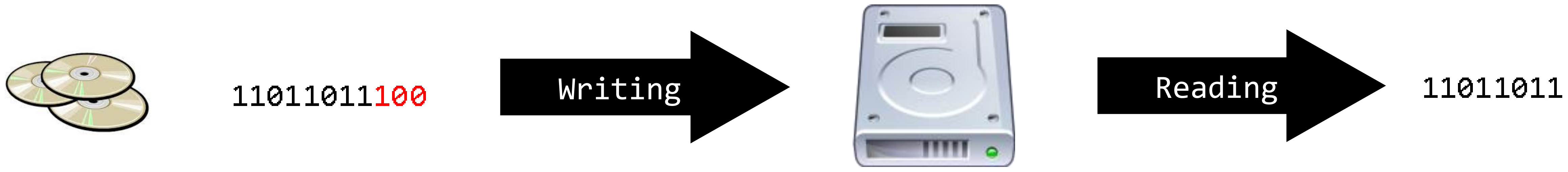
01000
11000

PRIMARY MEMORY

Error detections
CRC

The cyclic redundancy check (CRC) used for error detection when digital data are transferred from source to destination e.g. copying data from CD or DVD to a hard drive.

A certain number of bits (sometimes called a checksum) is added to the original data and then the data is sent to the destination. The received data in the destination is tested for errors using CRC.



PRIMARY MEMORY

CRC

Modulo-2 Arithmetic (XOR ⊕)

Modulo-2 Arithmetic (⊕) is applied digit by digit on binary numbers

$0 \oplus 0 = 0$

$1 \oplus 0 = 1$

$0 \oplus 1 = 1$

$1 \oplus 1 = 0$

Example:

```
  1011010
⊕ 0111101
  -------
  1100111
```

PRIMARY MEMORY

CRC

1) Define an agreed-upon divisor (generator)
2) Add the length of generator – 1 zeros to the data from the right side
3) Divide the extended data by the generator using ⊕ operation
4) If the reminder is not zero the zeros added in part 2 must be replaced by the reminder
5) Now the data is ready to be sent to the receiver

Example:

Data = 1101
Generator = 110

Codeword = 110100

```
110 | 110100
      110
      ----------
      000100
       000
      ----------
       00100
        000
      ----------
        0100
         110
      ----------
         010
```

Sender

Codeword = 110110

Ready to be sent to the receiver

```
110 | 110110
      110
      ----------
      000110
       000
      ----------
       00110
        000
      ----------
        0110
         110
      ----------
         000
```

Receiver

PRIMARY MEMORY

CRC

Example:

Data = 1101
Generator = 100

Codeword = 110100

Sender

```
100 | 110100
      100
      ----------
      010100
       100
      ----------
       00100
        000
      ----------
        0100
         100
      ----------
          000
```

Codeword = 110100

Ready to be sent to the receiver

Receiver

```
100 | 110100
      100
      ----------
      010100
       100
      ----------
       00100
        000
      ----------
        0100
         100
      ----------
          000 ✓
```

PRIMARY MEMORY

CRC

Example:

Data = 11011011
Generator = 1101

Codeword = 11011011000

Reminder = 100

Codeword = 11011011100

Ready to be sent to the receiver

```
1101 | 11011011000
       1101
       -----------------------
       0001011000
       0000
       -----------------------
        001011000
        0000
       -----------------------
         01011000
         0000
       -----------------------
          1011000
          1101
       -----------------------
           110000
           1101
       -----------------------
            00100
            0000
       -----------------------
             0100
             0000
       -----------------------
              100
```

PRIMARY MEMORY

Error detections and corrections
Hamming code

Number of check bits for a code that can correct **a single error**

$$(m + r + 1) \leqslant 2^r$$

m word size and r check bits

$m = 8, r = 1 \Rightarrow 10 \leq 2$
$m = 8, r = 2 \Rightarrow 11 \leq 4$
$m = 8, r = 3 \Rightarrow 12 \leq 8$
$\underline{m = 8, r = 4 \Rightarrow 13 \leq 16}$

Word size	Check bits	Total size	Percent overhead
8	4	12	50
16	5	21	31
32	6	38	19
64	7	71	11
128	8	136	6
256	9	265	4
512	10	522	2

Figure 2-13. Number of check bits for a code that can correct a single error.

PRIMARY MEMORY

Error detections and corrections

Hamming code(m = 4, r = 3)

$$(m + r + 1) \leqslant 2^r$$

m = 4, r = 3 $\Rightarrow$ 8 $\leq$ 8

Data word = abcd

Parity bits = xyz

Codeword = xyazbcd

1	2	3	4	5	6	7
x	y	a	z	b	c	d
2^0	2^1		2^2			

PRIMARY MEMORY

Error detections and corrections

Hamming code(m = 4, r = 3)

1	2	3	4	5	6	7
x	y	a	z	b	c	d
2^0	2^1		2^2			

		3 a	5 b	6 c	7 d
1	x	✓	✓	×	✓
2	y	✓	×	✓	✓
4	z	×	✓	✓	✓

X(1)

$a = 3 = (11)_2 = 2 + 1 = 2 + x$

$b = 5 = (101)_2 = 4 + 1 = 4 + x$

$c = 6 = (110)_2 = 2 + 4$

$d = 7 = (111)_2 = 4 + 2 + 1 = 4 + 2 + x$

PRIMARY MEMORY

Error detections and corrections

Hamming code(m = 4, r = 3)

1	2	3	4	5	6	7
x	y	a	z	b	c	d
2^0	2^1		2^2			

		3	5	6	7
		a	b	c	d
1	x	✓	✓	×	✓
2	y	✓	×	✓	✓
4	z	×	✓	✓	✓

Y(2)

$a = 3 = (11)_2 = 2 + 1 = Y + 1$

$b = 5 = (101)_2 = 4 + 1$

$c = 6 = (110)_2 = 4 + 2 = 4 + Y$

$d = 7 = (111)_2 = 4 + 2 + 1 = 4 + Y + 1$

PRIMARY MEMORY

Error detections and corrections
Hamming code(m = 4, r = 3)

1	2	3	4	5	6	7
x	y	a	z	b	c	d
2^0	2^1		2^2			

		3 a	5 b	6 c	7 d
1	x	✓	✓	×	✓
2	y	✓	×	✓	✓
4	z	×	✓	✓	✓

Z(4)

$a = 3 = (11)_2 = 2 + 1$

$b = 5 = (101)_2 = 4 + 1 = Z + 1$

$c = 6 = (110)_2 = 4 + 2 = Z + 2$

$d = 7 = (111)_2 = 4 + 2 + 1 = Z + 2 + 1$

PRIMARY MEMORY

Error detections and corrections

Hamming code(m = 4, r = 3)

Data word = abcd

Parity bits = xyz

Codeword = xyazbcd

$x = a \oplus b \oplus d$

$y = a \oplus c \oplus d$

$z = b \oplus c \oplus d$

PRIMARY MEMORY

Error detections and corrections
Hamming code(m = 4, r = 3)

Example:

Imagine you have received the codeword 1011010. Do you think the codeword is corrupted or not?

$$\text{Codeword} = \begin{matrix} 1011010 \\ xyazbcd \end{matrix}$$

$$x = a \oplus b \oplus d \qquad 1 = 1 \oplus 0 \oplus 0 \quad ✓$$
$$y = a \oplus c \oplus d \qquad 0 = 1 \oplus 1 \oplus 0 \quad ✓$$
$$z = b \oplus c \oplus d \qquad 1 = 0 \oplus 1 \oplus 0 \quad ✓$$

Not corrupted

PRIMARY MEMORY

Error detections and corrections
Hamming code(m = 8, r = 4)

$$(m + r + 1) \leqslant 2^r$$

$\underline{m = 8,\ r = 4 \Rightarrow 13 \leq 16}$

Data word = **D3 D5 D6 D7 D9 D10 D11 D12**
Parity bits = **P1 P2 P4 P8**

1	2	3	4	5	6	7	8	9	10	11	12
P1	P2	D3	P4	D5	D6	D7	P8	D9	D10	D11	D12

	D3	D5	D6	D7	D9	D10	D11	D12
P1	✓	✓	✗	✓	✓	✗	✓	✗
P2	✓	✗	✓	✓	✗	✓	✓	✗
P4	✗	✓	✓	✓	✗	✗	✗	✓
P8	✗	✗	✗	✗	✓	✓	✓	✓

PRIMARY MEMORY

Error detections and corrections
Hamming code(general case)

Hamming’s algorithm

1. r parity bits are added to an m-bit word, forming a new word of length m + r bits.
2. The bits are numbered starting at 1, not 0, with bit 1 the leftmost (high-order) bit.
3. All bits whose bit number is a power of 2 are parity bits; the rest are used for data.
4. Each parity bit checks specific bit positions;

For example, with a 16-bit word, 5 parity bits are added $(16+r+1) \leq 2^r \Rightarrow r = 5$
Bits 1, 2, 4, 8, and 16 are parity bits, and all the rest are data bits. In all, the memory word has 21 bits (16 data, 5 parity).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
2^0	2^1		2^2				2^3								2^4					

PRIMARY MEMORY

Error detections and corrections
Hamming code

Hamming's algorithm

The bit positions checked by the parity bits are (here even parity is considered):

Bit 1 checks bits 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21.

Bit 2 checks bits 2, 3, 6, 7, 10, 11, 14, 15, 18, 19.

Bit 4 checks bits 4, 5, 6, 7, 12, 13, 14, 15, 20, 21.

Bit 8 checks bits 8, 9, 10, 11, 12, 13, 14, 15.

Bit 16 checks bits 16, 17, 18, 19, 20, 21.

PRIMARY MEMORY

Error detections and corrections
Hamming code

Example : Error-Correcting: **001001100000101101110** (16 data and 5 check bits)

Bit 1 checks bits 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21.

Bit 2 checks bits 2, 3, 6, 7, 10, 11, 14, 15, 18, 19.

Bit 4 checks bits 4, 5, 6, 7, 12, 13, 14, 15, 20, 21.

Bit 8 checks bits 8, 9, 10, 11, 12, 13, 14, 15.

Bit 16 checks bits 16, 17, 18, 19, 20, 21.

The position of wrong bit:
4 (bit 4) + 1 (bit 1) = 5

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	0	1	0	0	1	1	0	0	0	0	0	1	0	1	1	0	1	1	1	0

Bit 1	☒	five 1's	0	0	1	0	0	1	1	0	0	0	0	0	1	0	1	1	0	1	1	1	0
Bit 2	☑	six 1's	0	0	1	0	0	1	1	0	0	0	0	0	1	0	1	1	0	1	1	1	0
Bit 4	☒	five 1's	0	0	1	0	0	1	1	0	0	0	0	0	1	0	1	1	0	1	1	1	0
Bit 8	☑	two 1's	0	0	1	0	0	1	1	0	0	0	0	0	1	0	1	1	0	1	1	1	0
Bit 16	☑	four 1's	0	0	1	0	0	1	1	0	0	0	0	0	1	0	1	1	0	1	1	1	0

0	0	1	0	1	1	1	0	0	0	0	0	1	0	1	1	0	1	1	1	0

Corrected code word

PRIMARY MEMORY

Error detections and corrections
Hamming code

Example : Construction of the Hamming code for the memory word **1111000010101110** (16 data bits so we need and 5 check bits).

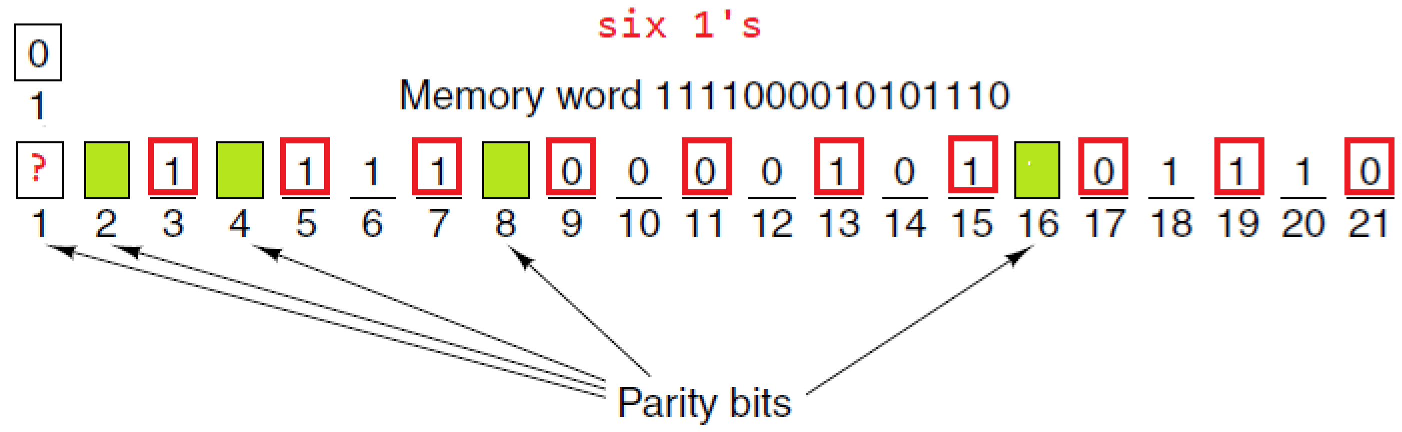
Bit 1 checks bits 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21.

Bit 2 checks bits 2, 3, 6, 7, 10, 11, 14, 15, 18, 19.

Bit 4 checks bits 4, 5, 6, 7, 12, 13, 14, 15, 20, 21.

Bit 8 checks bits 8, 9, 10, 11, 12, 13, 14, 15.

Bit 16 checks bits 16, 17, 18, 19, 20, 21.



Bit 1 checks bits 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21.

PRIMARY MEMORY

Error detections and corrections
Hamming code

Example : Construction of the Hamming code for the memory word **1111000010101110** (16 data bits so we need and 5 check bits).

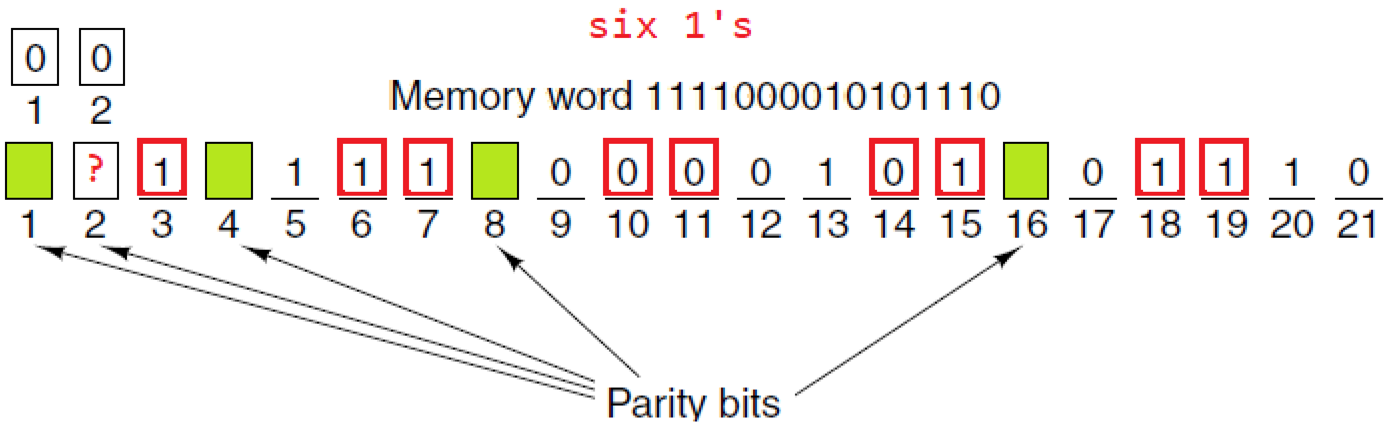
Bit 1 checks bits 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21.

Bit 2 checks bits 2, 3, 6, 7, 10, 11, 14, 15, 18, 19.

Bit 4 checks bits 4, 5, 6, 7, 12, 13, 14, 15, 20, 21.

Bit 8 checks bits 8, 9, 10, 11, 12, 13, 14, 15.

Bit 16 checks bits 16, 17, 18, 19, 20, 21.



Bit 2 checks bits 2, 3, 6, 7, 10, 11, 14, 15, 18, 19.

PRIMARY MEMORY

Error detections and corrections
Hamming code

Example : Construction of the Hamming code for the memory word **1111000010101110** (16 data bits so we need and 5 check bits).

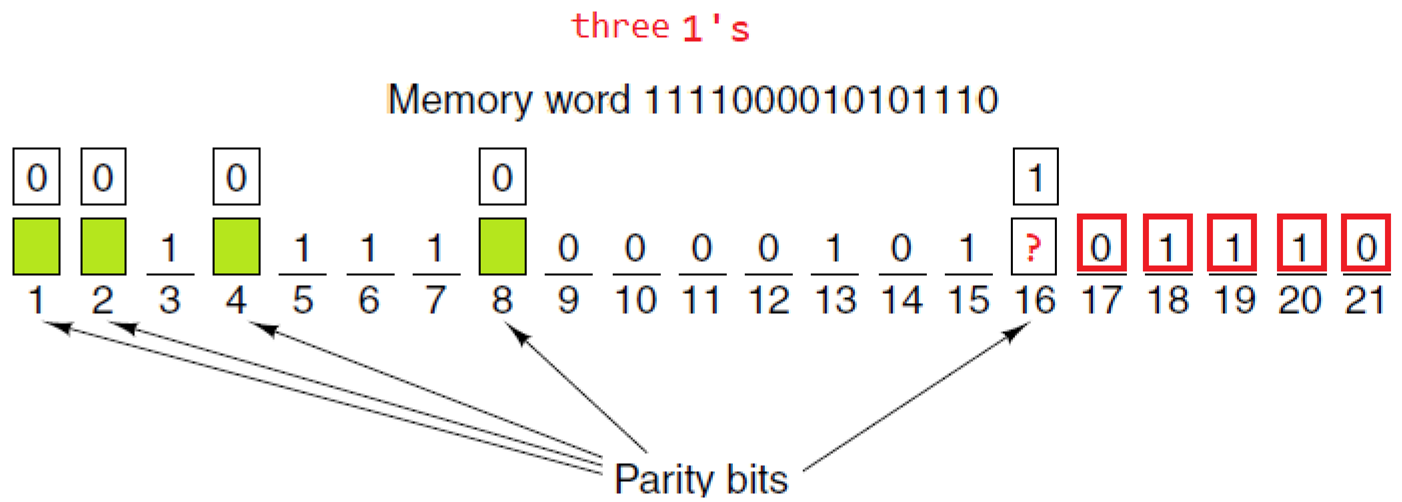
Bit 1 checks bits 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21.

Bit 2 checks bits 2, 3, 6, 7, 10, 11, 14, 15, 18, 19.

Bit 4 checks bits 4, 5, 6, 7, 12, 13, 14, 15, 20, 21.

Bit 8 checks bits 8, 9, 10, 11, 12, 13, 14, 15.

Bit 16 checks bits 16, 17, 18, 19, 20, 21.



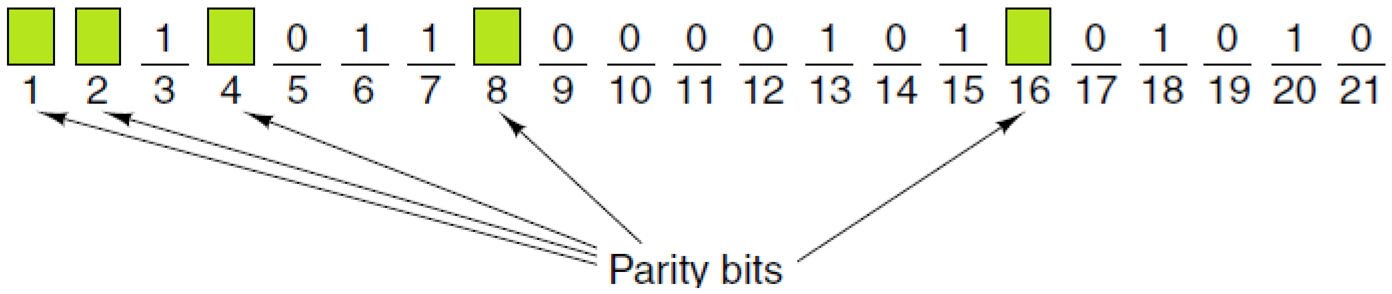
Bit 16 checks bits 16, 17, 18, 19, 20, 21.

PRIMARY MEMORY

Error detections and corrections
Hamming code

Example : Construction of the Hamming code for the memory word **1011000010101010** (? data bits so we need and ? check bits)?

Memory word **1011000010101010**



Bit 1 checks bits 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21.

Bit 2 checks bits 2, 3, 6, 7, 10, 11, 14, 15, 18, 19.

Bit 4 checks bits 4, 5, 6, 7, 12, 13, 14, 15, 20, 21.

Bit 8 checks bits 8, 9, 10, 11, 12, 13, 14, 15.

Bit 16 checks bits 16, 17, 18, 19, 20, 21.