Chapter #5
Internal Memory

Internal Memory Types

<table>
<tr><th>Memory Type</th><th>Category</th><th>Erasure</th><th>Write Mechanism</th><th>Volatility</th></tr>
<tr><td>Random-access memory (RAM)</td><td>Read-write memory</td><td>Electrically, byte-level</td><td>Electrically</td><td>Volatile</td></tr>
<tr><td>Read-only memory (ROM)</td><td rowspan="2">Read-only memory</td><td rowspan="2">Not possible</td><td>Masks</td><td rowspan="5">Nonvolatile</td></tr>
<tr><td>Programmable ROM (PROM)</td><td rowspan="4">Electrically</td></tr>
<tr><td>Erasable PROM (EPROM)</td><td rowspan="3">Read-mostly memory</td><td>UV light, chip-level</td></tr>
<tr><td>Electrically Erasable PROM (EEPROM)</td><td>Electrically, byte-level</td></tr>
<tr><td>Flash memory</td><td>Electrically, block-level</td></tr>
</table>

Random Access Memory

- RAM
  - —Misnamed as all semiconductor memory is random access
  - —Read/Write
  - —Volatile
  - —Temporary storage
  - —Static or dynamic

Dynamic RAM

- Storage:
  - —Bits stored as charge in capacitors
- Adv:
  - —Simpler construction
  - —Smaller per bit
  - —Less expensive
- Disadv:
  - —Charges leak
  - —Need refreshing even when powered
  - —Slower

Static RAM

- Storage:
  - —Bits stored as on/off switches
- Adv:
  - —Charges don’t leak
  - —No refreshing needed when powered
  - —Faster
- Disadv:
  - —More complex construction
  - —Larger per bit
  - —More expensive

Read Only Memory (ROM)

- Written during manufacture
  - —Very expensive for small runs
- Programmable (once)
  - —PROM
  - —Needs special equipment to program
- Read “mostly”
  - —Erasable Programmable (EPROM)
    - – Erased by UV
  - —Electrically Erasable (EEPROM)
    - – Takes much longer to write than read
  - —Flash memory
    - – Erase whole memory electrically

Error Detection & Correction

- Detection:
  - — Find one or more erroneous bits
  - — Report & resend data
- Correction:
  - —Find one or more erroneous bits
  - —Report and fix the error(s)
- Methods:
  - —Parity bit code
  - —Hamming code

Types of Errors

- Hard failure
  - —Permanent defect
  - —Eg. broken wire, burned transistor
- Soft Error
  - —Random, non-destructive
  - —No permanent damage to memory
  - —Eg. Noise on bus, weak signal (weak 0, weak 1)

Parity Bit Code

- Bit appended to data to determine even/odd number of 1's in data
- Parity bit sent is compared to parity bit received to determine possible error
- Types:
  - —Odd parity: parity bit is set to 1 if total number of 1's in data is even, 0 otherwise
  - —Even parity: parity bit is set to 1 if total number of 1's in data is odd, 0 otherwise
- Capability:
  - —Detect odd number of errors
  - —Correct no errors

Hamming Code

- Length of code:

  M original data bits + *K* parity bits

- Relation between *K* and *M*:

  $K \geq \lceil \log_2 (M + \lceil \log_2 M \rceil + 1) \rceil$

- Capability:

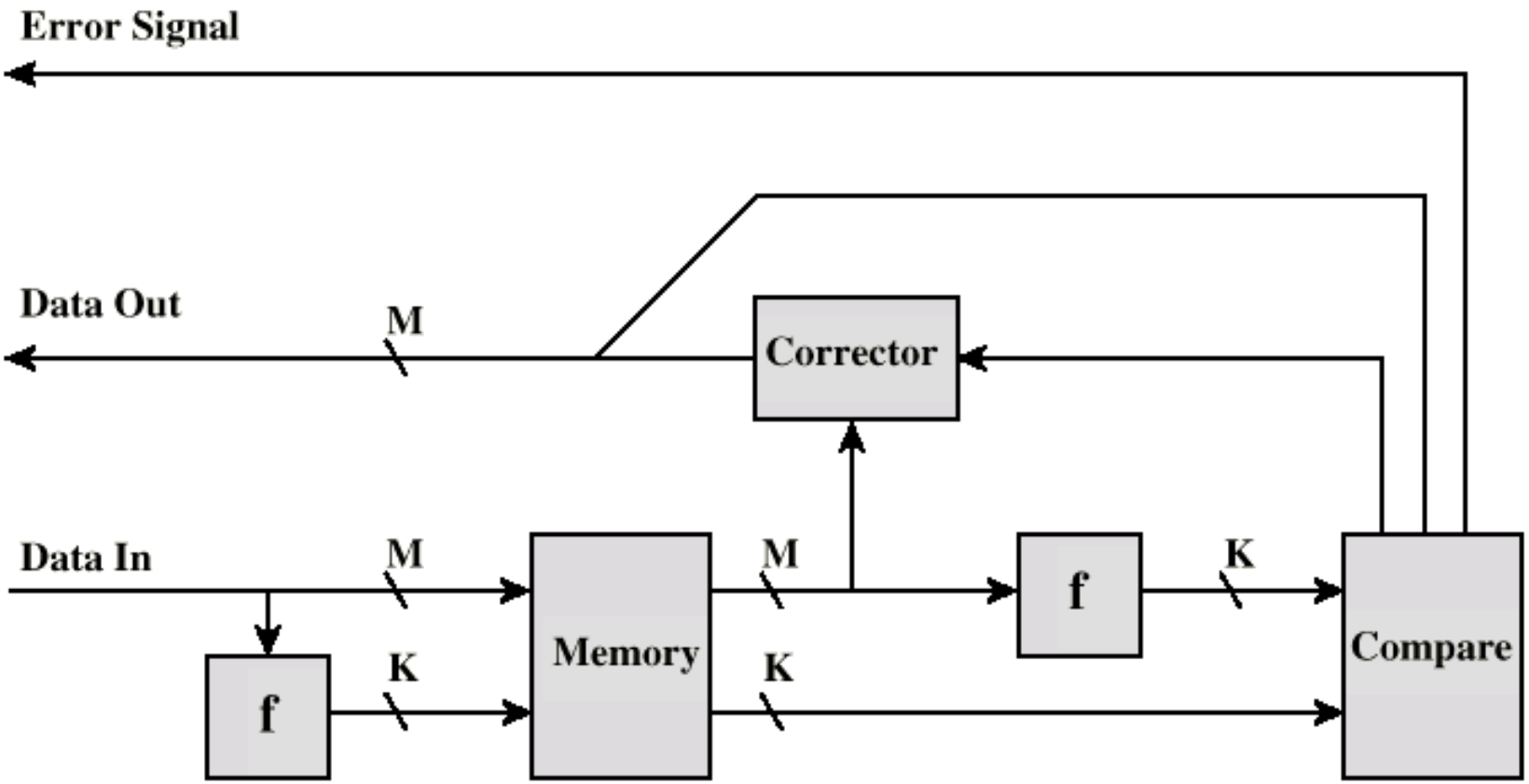
  —Detect multiple number of errors
  (2 errors always, >2 errors sometimes)

  —Correct one error

Example Hamming Code Lengths

Data bits	Check bits	Increase
8	4	50%
16	5	31.25%
32	6	18.75%
64	7	10.94%
128	8	6.25%
256	9	3.52%

Hamming Code Implementation



Hamming Code Layout

Bit Position	12	11	10	9	8	7	6	5	4	3	2	1
Position Number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data Bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check Bit					C8				C4		C2	C1

Figure 5.9 Layout of Data Bits and Check Bits

Hamming Code Correction

- Hamming code:

  8 original data bits: D_8 to D_1
  4 check bits: C_8, C_4, C_2, C_1

- Recalculate parity bits:

  $C_1' = D_1 \oplus D_2 \oplus D_4 \oplus D_5 \oplus D_7$ (every other bit position)
  $C_2' = D_1 \oplus D_3 \oplus D_4 \oplus D_6 \oplus D_7$ (every other 2 bit positions)
  $C_4' = D_2 \oplus D_3 \oplus D_4 \oplus D_8$ (every other 4 bit positions)
  $C_8' = D_5 \oplus D_6 \oplus D_7 \oplus D_8$ (every other 8 bit positions)

- Compare original & new parity bits (difference is bit in error):

  C_8 C_4 C_2 C_1
  $\oplus$ C_8' C_4' C_2' C_1'

Hamming Code Correction Example

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check bit					C8				C4		C2	C1
Word stored as	0	0	1	1	0	1	0	0	1	1	1	1
Word fetched as	0	0	1	1	0	1	1	0	1	1	1	1
Position Number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Check Bit					0				0		0	1

Figure 5.10 Check Bit Calculation