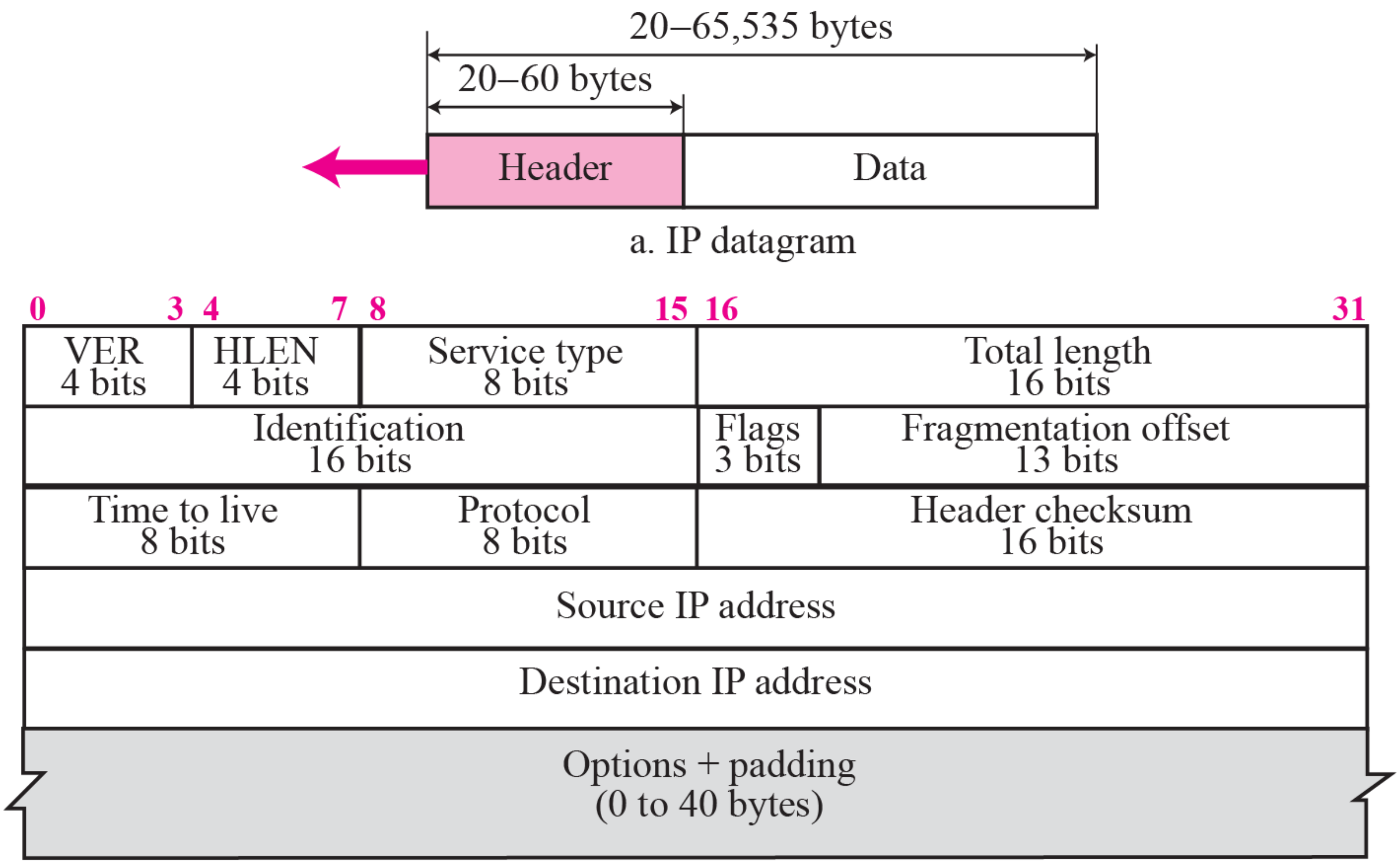
# Figure 7.2 *IP datagram*




b. Header format

## **Figure 7.3** *Service type*

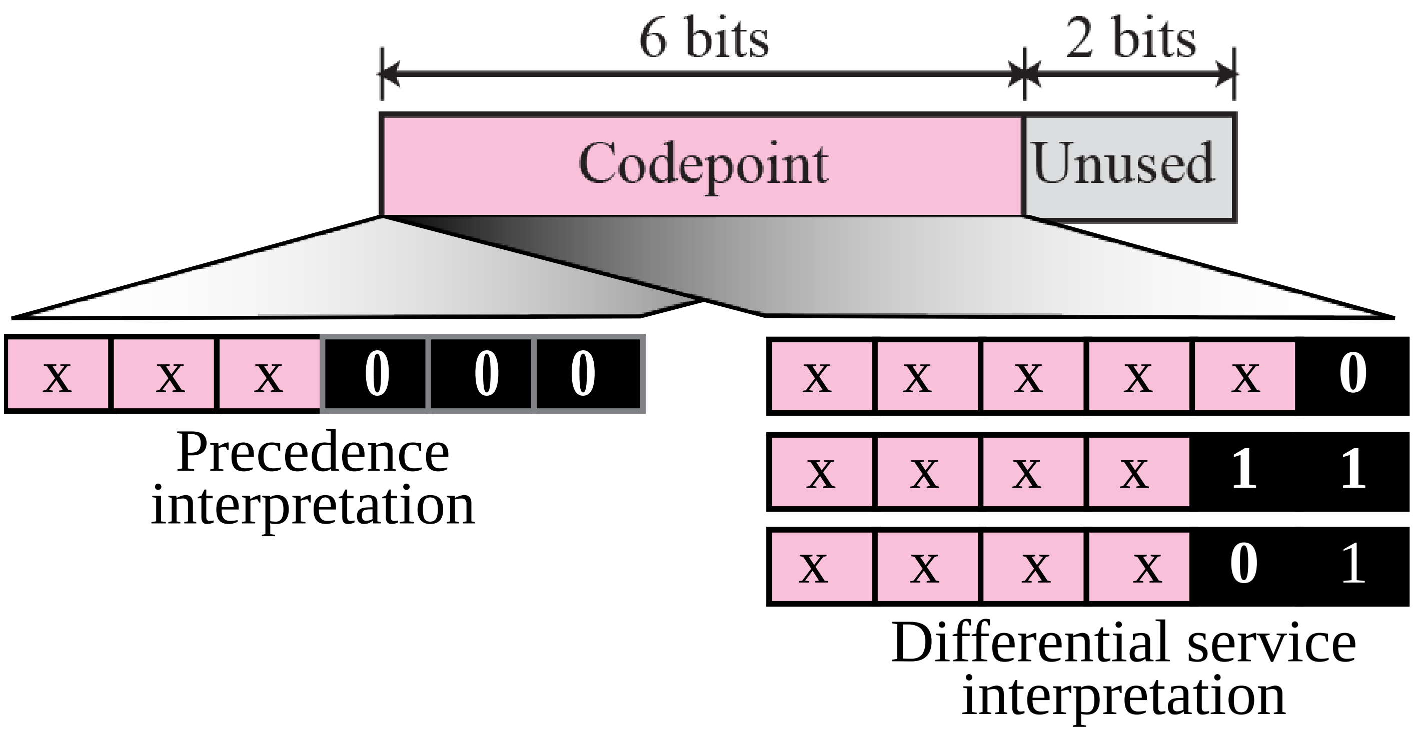


**Table 7.1** *Values for codepoints*

| *Category* | *Codepoint* | *Assigning Authority* |
|---|---|---|
| 1 | XXXXX0 | Internet |
| 2 | XXXX11 | Local |
| 3 | XXXX01 | Temporary or experimental |

## Figure 7.4 *Encapsulation of a small datagram in an Ethernet frame*

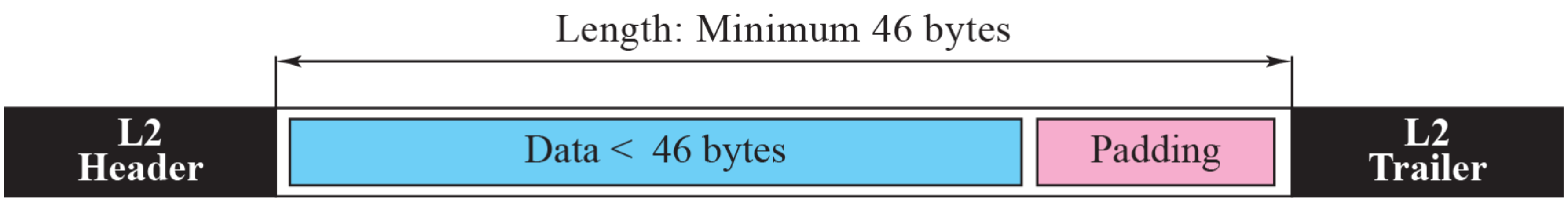


**Table 7.2** *Protocols*

| *Value* | *Protocol* | *Value* | *Protocol* |
|---|---|---|---|
| 1 | ICMP | 17 | UDP |
| 2 | IGMP | 89 | OSPF |
| 6 | TCP | | |

# Example 7.1

An IP packet has arrived with the first 8 bits as shown:

| |
|---|
| 01000010 |

The receiver discards the packet. Why?

*Solution*

There is an error in this packet. The 4 left-most bits (0100) show the version, which is correct. The next 4 bits (0010) show the wrong header length ($2 \times 4 = 8$). The minimum number of bytes in the header must be 20. The packet has been corrupted in transmission.

# Example 7.2

In an IP packet, the value of HLEN is 1000 in binary. How many bytes of options are being carried by this packet?

*Solution*

The HLEN value is 8, which means the total number of bytes in the header is $8 \times 4$ or 32 bytes. The first 20 bytes are the base header, the next 12 bytes are the options.

# Example 7.3

In an IP packet, the value of HLEN is $5_{16}$ and the value of the total length field is $0028_{16}$. How many bytes of data are being carried by this packet?

*Solution*

The HLEN value is 5, which means the total number of bytes in the header is $5 \times 4$ or 20 bytes (no options). The total length is 40 bytes, which means the packet is carrying 20 bytes of data $(40 - 20)$.

# Example 7.4

An IP packet has arrived with the first few hexadecimal digits as shown below:

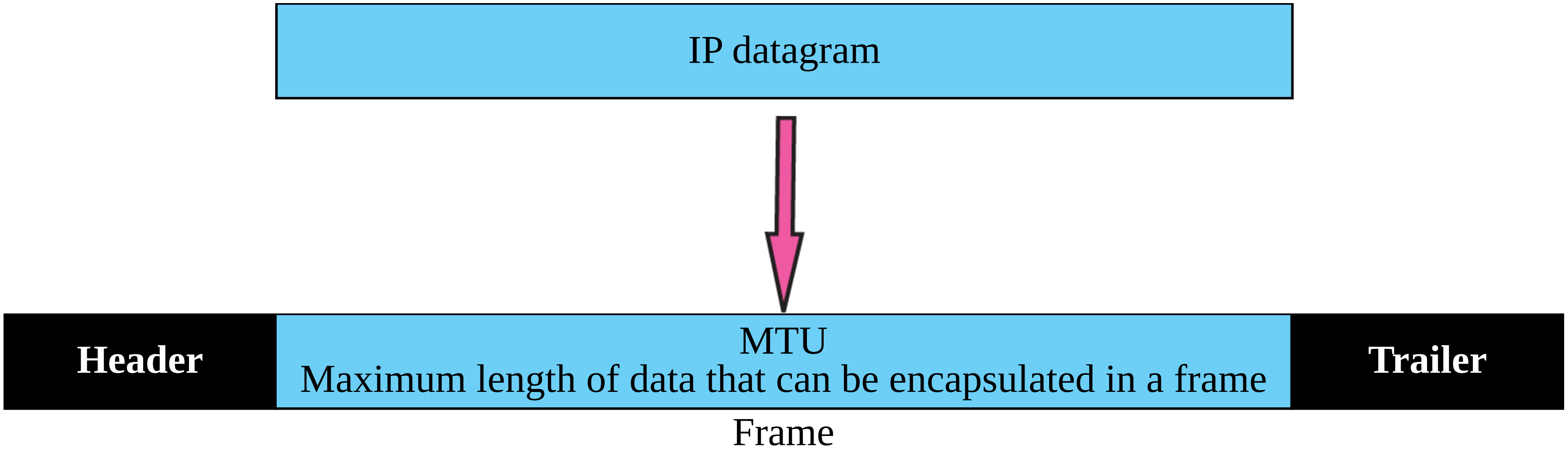
45000028000100000102 . . .

How many hops can this packet travel before being dropped? The data belong to what upper layer protocol?

*Solution*

To find the time-to-live field, we skip 8 bytes (16 hexadecimal digits). The time-to-live field is the ninth byte, which is 01. This means the packet can travel only one hop. The protocol field is the next byte (02), which means that the upper layer protocol is IGMP (see Table 7.2)

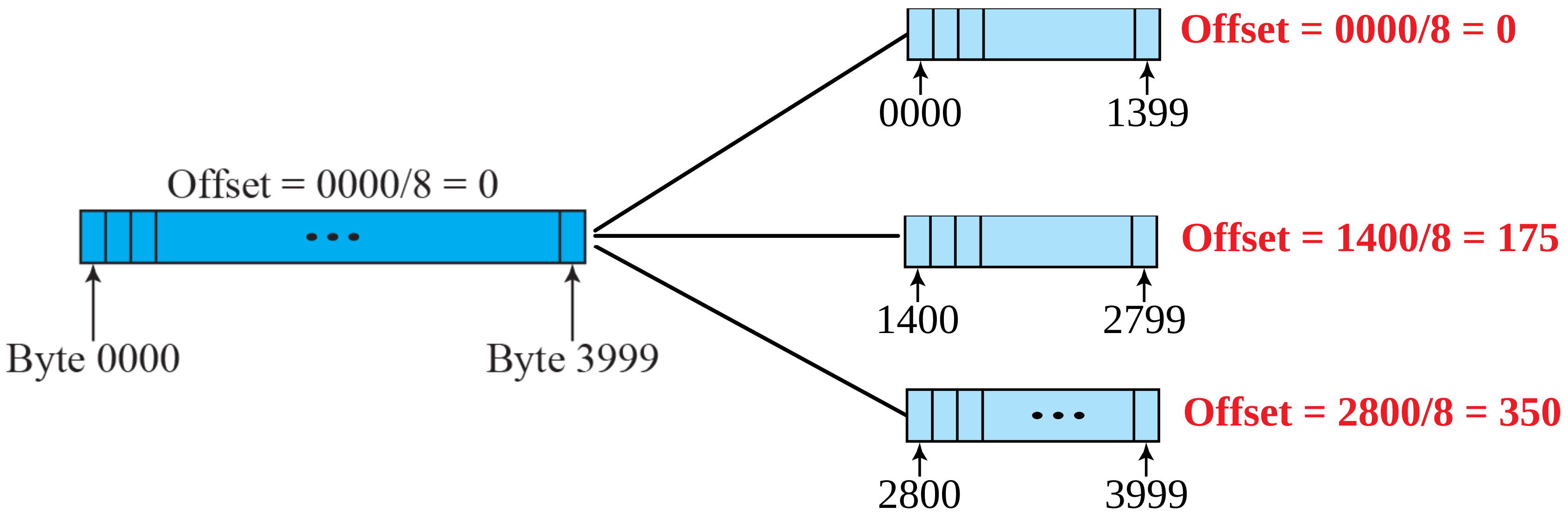
## Figure 7.6 *MTU*



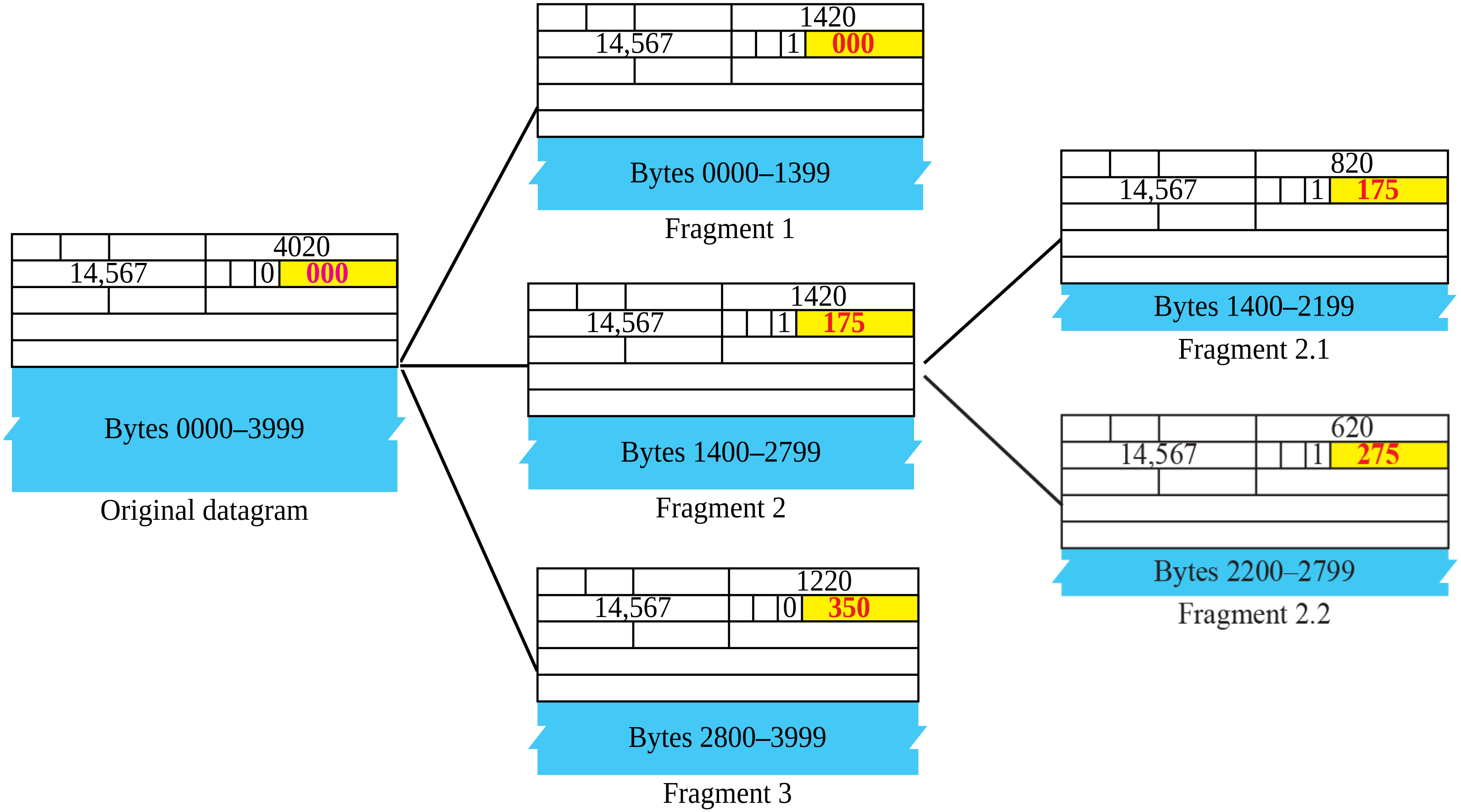
## Figure 7.7 *Flags field*

D: Do not fragment
M: More fragments

## Figure 7.8 *Fragmentation example*



# Figure 7.9 *Detailed fragmentation example*



# Example 7.5

A packet has arrived with an M bit value of 0. Is this the first fragment, the last fragment, or a middle fragment? Do we know if the packet was fragmented?

*Solution*

If the M bit is 0, it means that there are no more fragments; the fragment is the last one. However, we cannot say if the original packet was fragmented or not. A nonfragmented packet is considered the last fragment.

# Example 7.6

A packet has arrived with an M bit value of 1. Is this the first fragment, the last fragment, or a middle fragment? Do we know if the packet was fragmented?

*Solution*

If the M bit is 1, it means that there is at least one more fragment. This fragment can be the first one or a middle one, but not the last one. We don’t know if it is the first one or a middle one; we need more information (the value of the fragmentation offset). See also the next example.

# Example 7.7

A packet has arrived with an M bit value of 1 and a fragmentation offset value of zero. Is this the first fragment, the last fragment, or a middle fragment?

*Solution*

Because the M bit is 1, it is either the first fragment or a middle one. Because the offset value is 0, it is the first fragment.

# Example 7.8

A packet has arrived in which the offset value is 100. What is the number of the first byte? Do we know the number of the last byte?

*Solution*

To find the number of the first byte, we multiply the offset value by 8. This means that the first byte number is 800. We cannot determine the number of the last byte unless we know the length of the data.

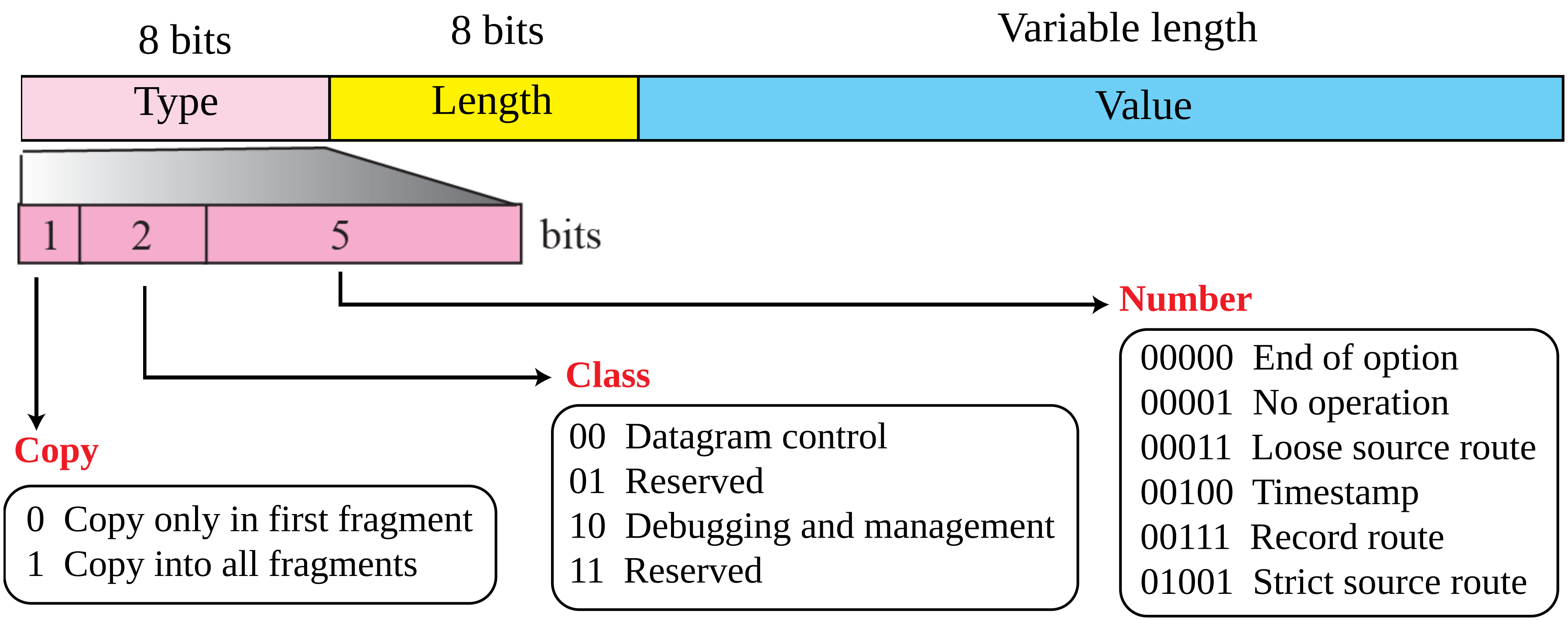
# Example 7.9

A packet has arrived in which the offset value is 100, the value of HLEN is 5 and the value of the total length field is 100. What is the number of the first byte and the last byte?

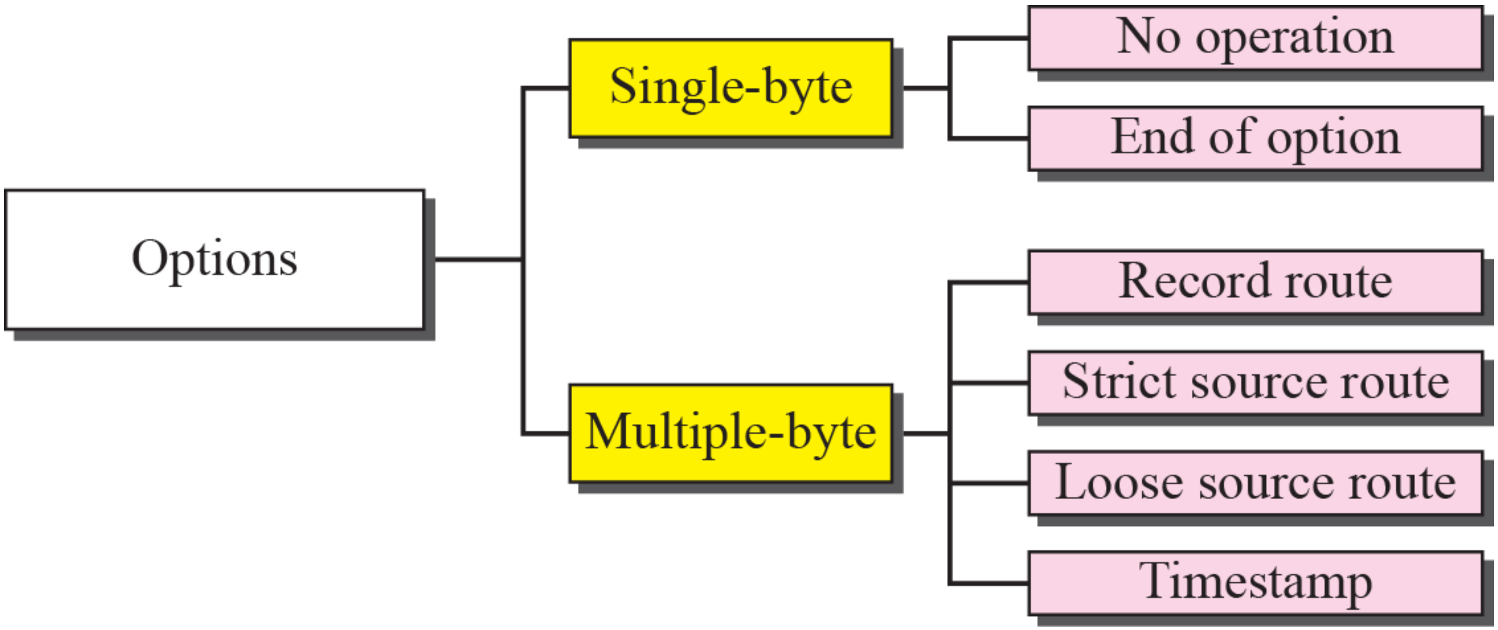
*Solution*

The first byte number is $100 \times 8 = 800$. The total length is 100 bytes and the header length is 20 bytes ($5 \times 4$), which means that there are 80 bytes in this datagram. If the first byte number is 800, the last byte number must be 879.

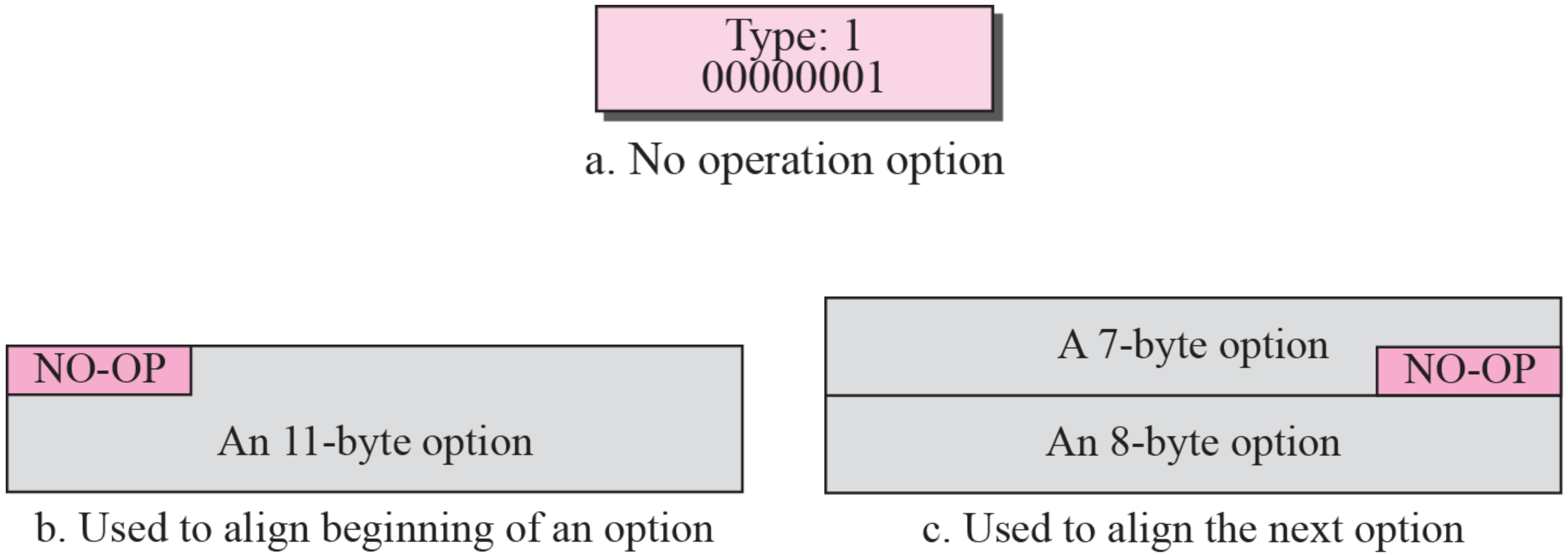
## Figure 7.10 *Option format*



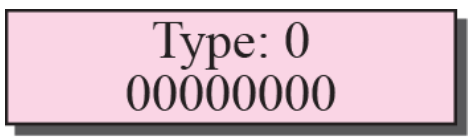
# **Figure 7.11** *Categories of options*



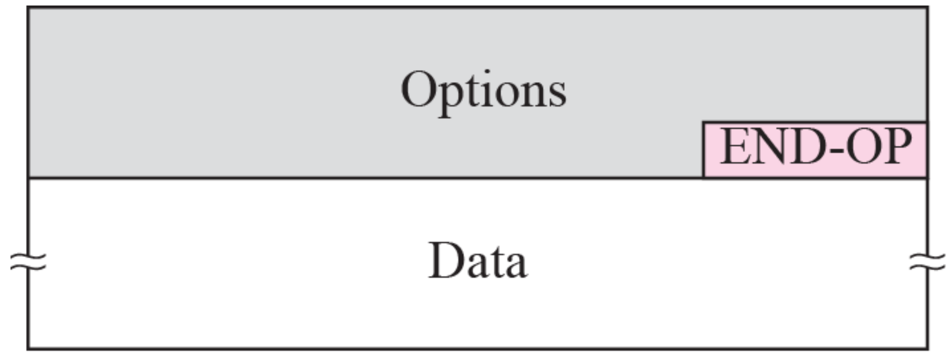
## Figure 7.12 *No operation option*



## Figure 7.13 *Endo-of-option option*

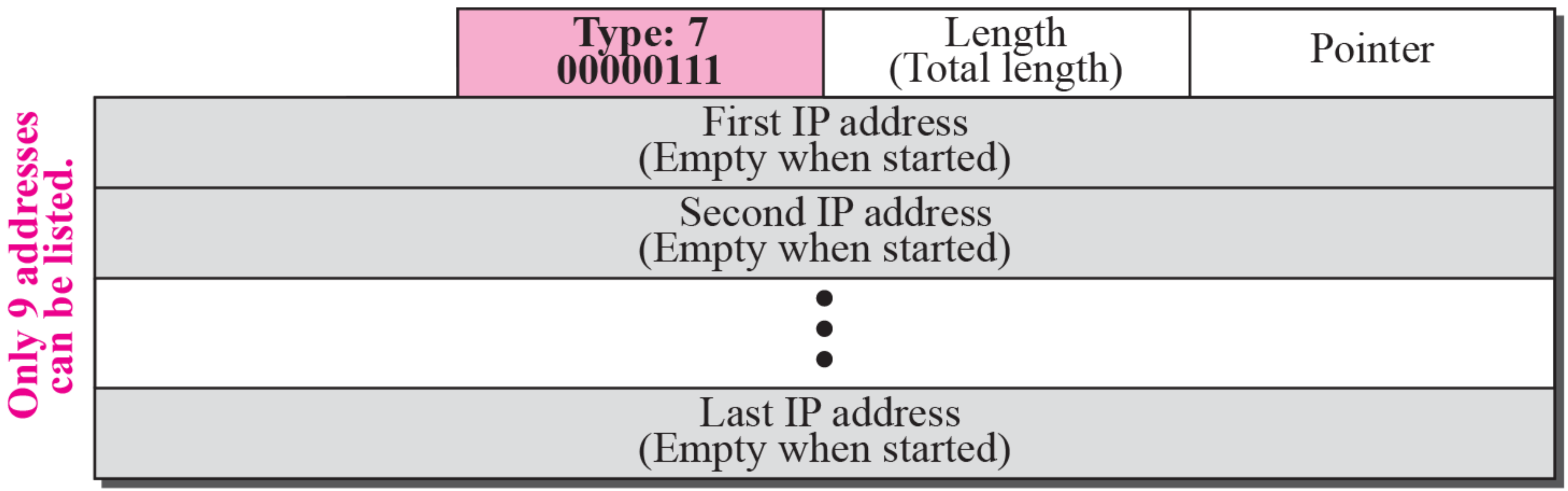



a. End of option

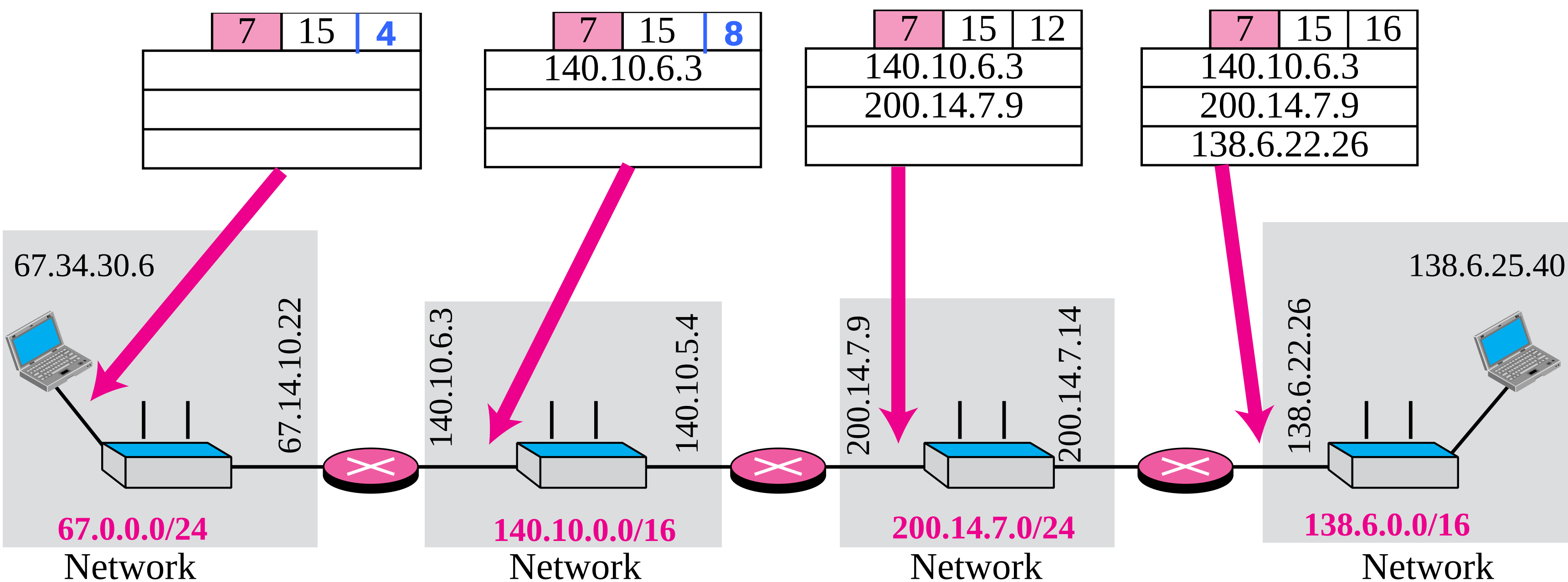



b. Used for padding

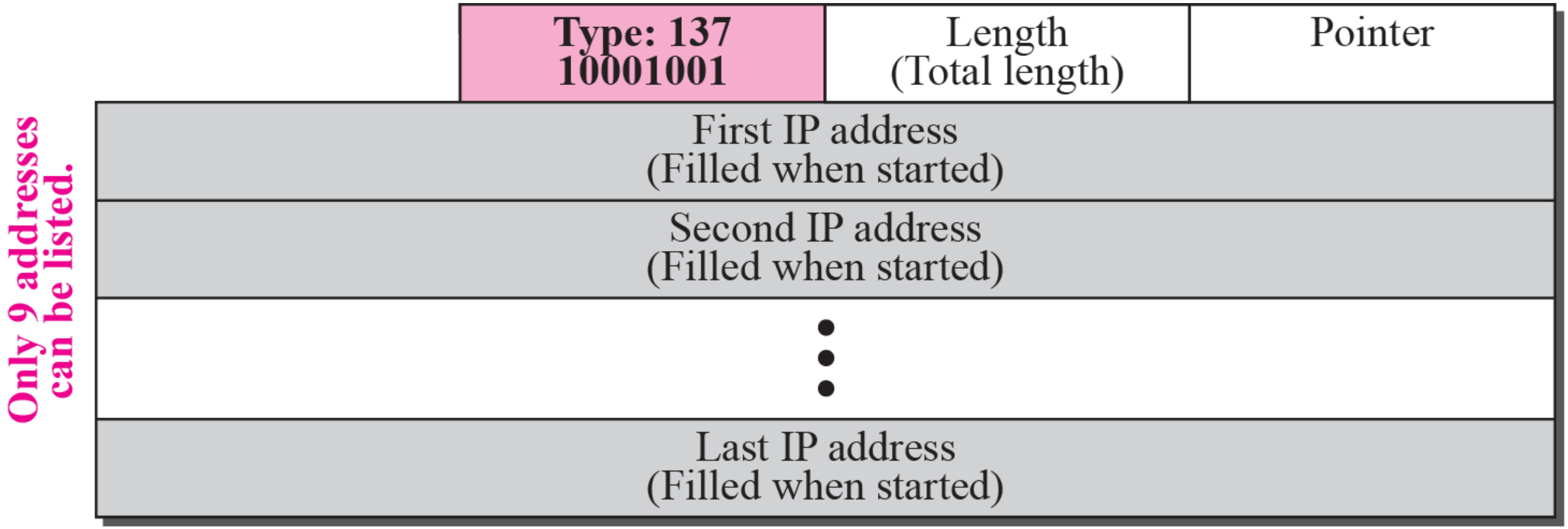
## Figure 7.14 *Record-route option*



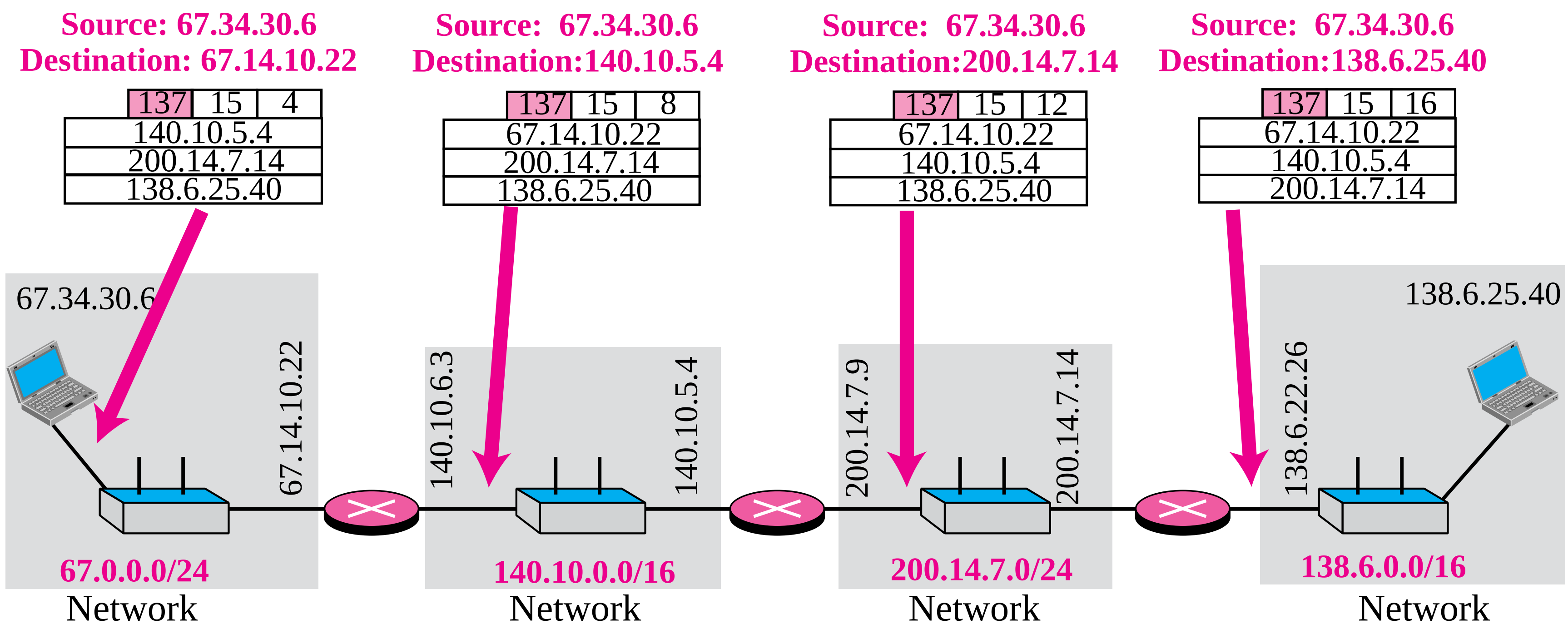
# Figure 7.15 *Record-route concept*



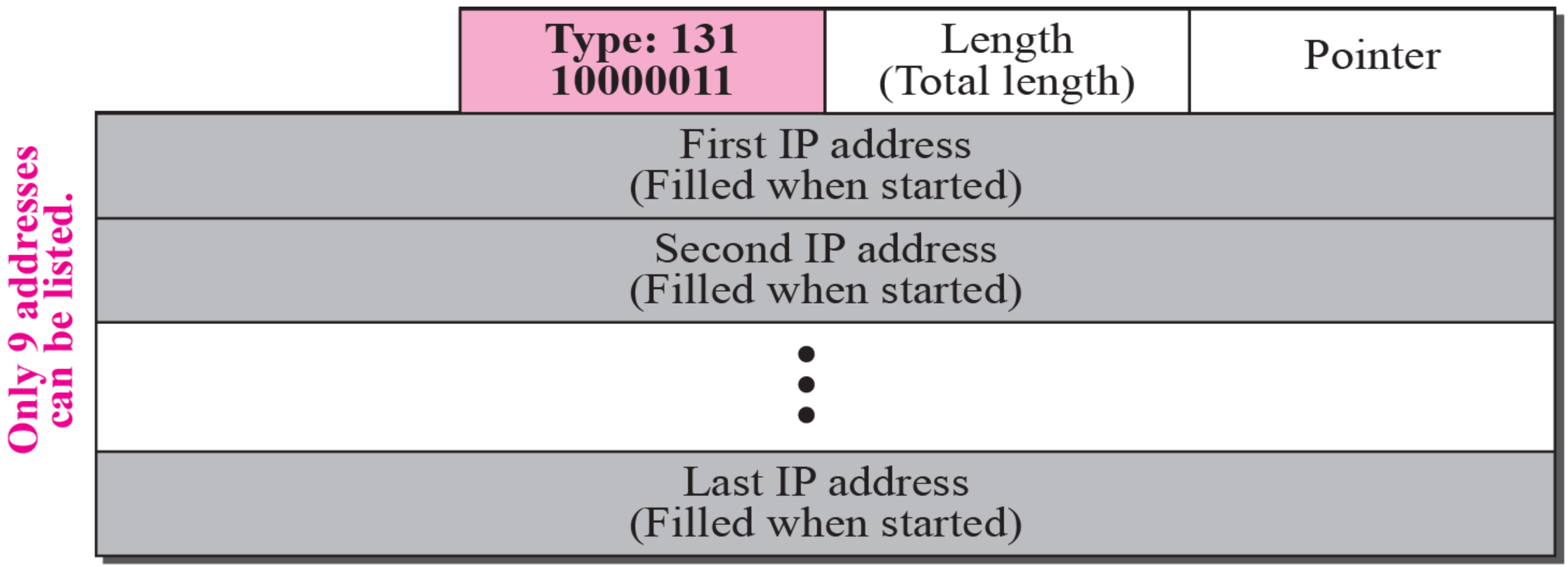
# Figure 7.16 *Strict-source-route option*



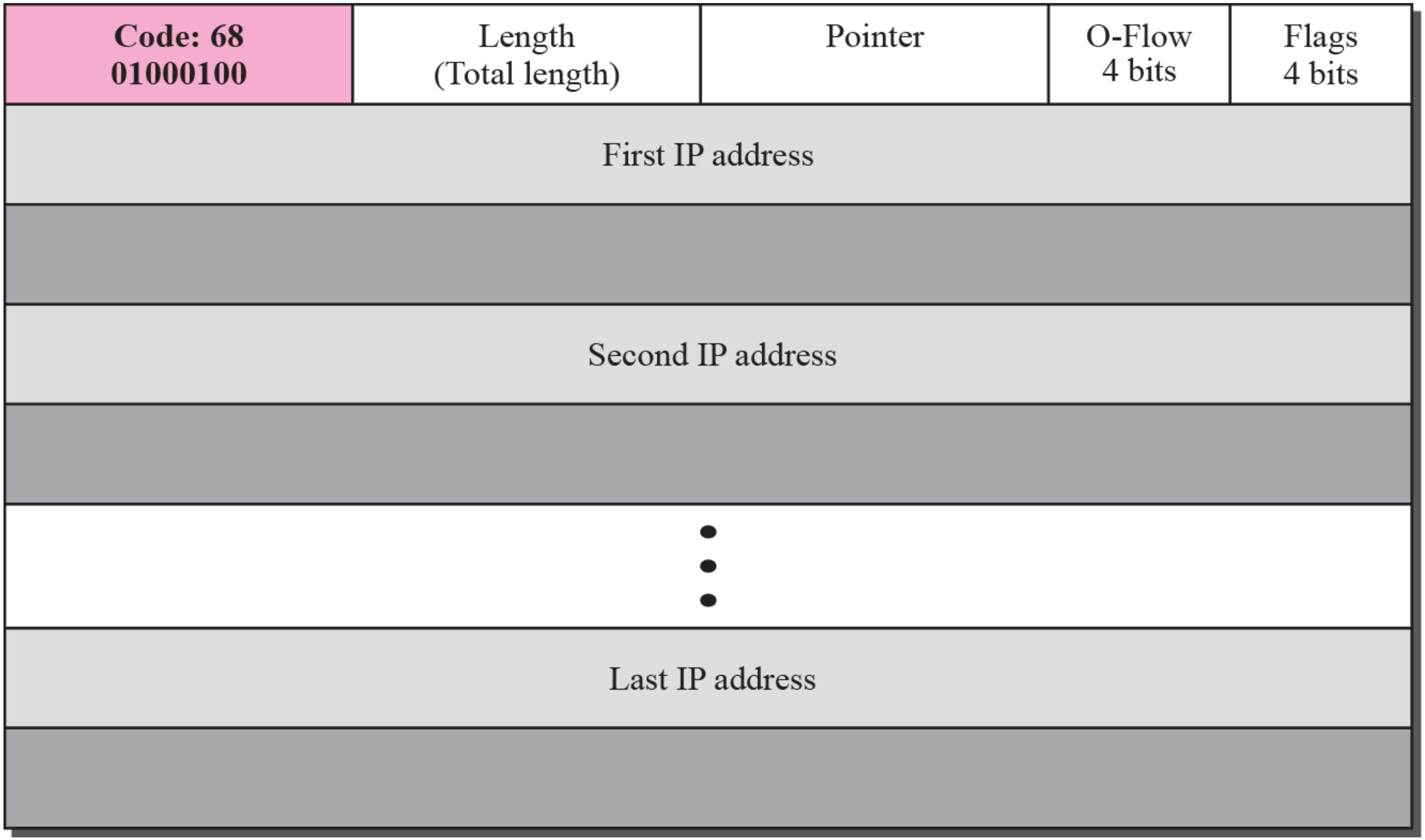
# Figure 7.17 *Strict-source-route option*



## Figure 7.18 *Loose-source-route option*

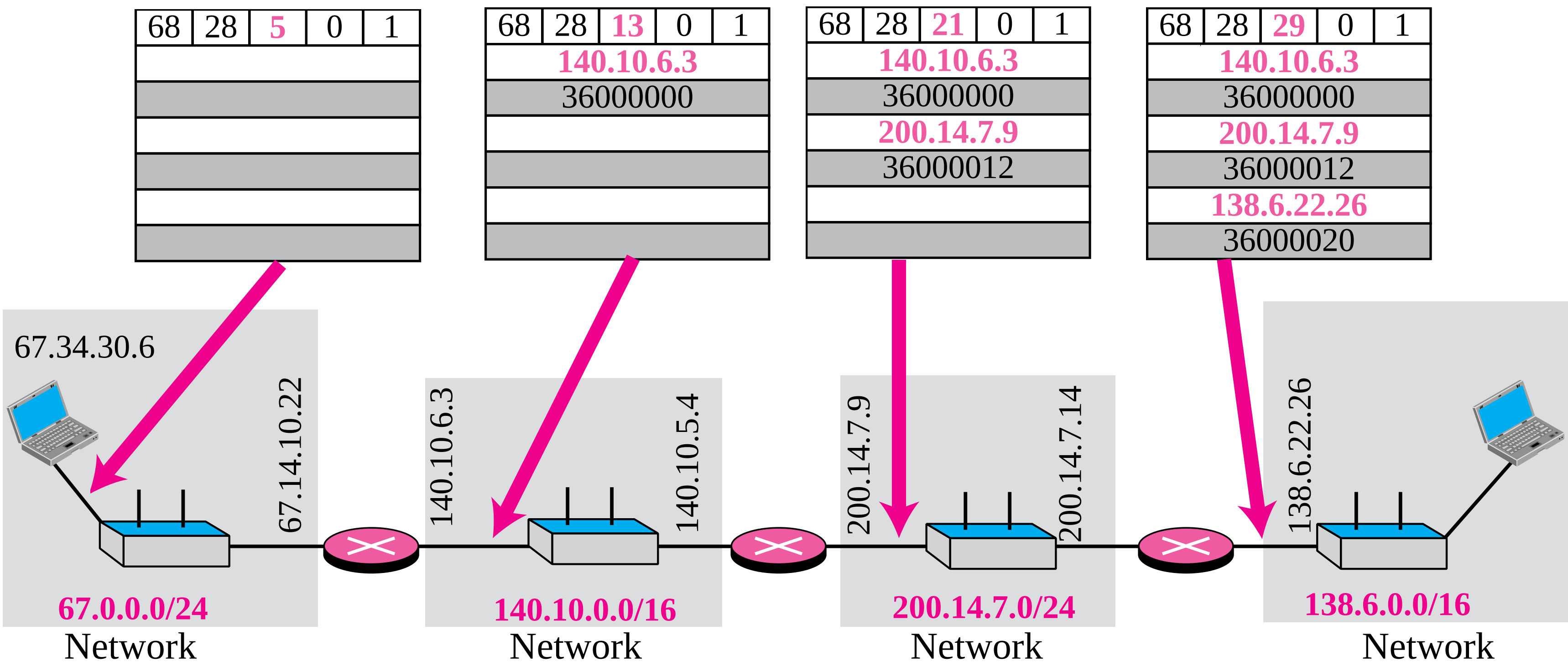


## **Figure 7.19** *Time-stamp option*



# Figure 7.20 *Use of flags in timestamp*

## Figure 7.21 *Timestamp concept*



# Example 7.10

Which of the six options must be copied to each fragment?

*Solution*

We look at the first (left-most) bit of the type for each option.

a. No operation: type is 00000001; not copied.
b. End of option: type is 00000000; not copied.
c. Record route: type is 00000111; not copied.
d. Strict source route: type is 10001001; copied.
e. Loose source route: type is 10000011; copied.
f. Timestamp: type is 01000100; not copied.

# Example 7.11

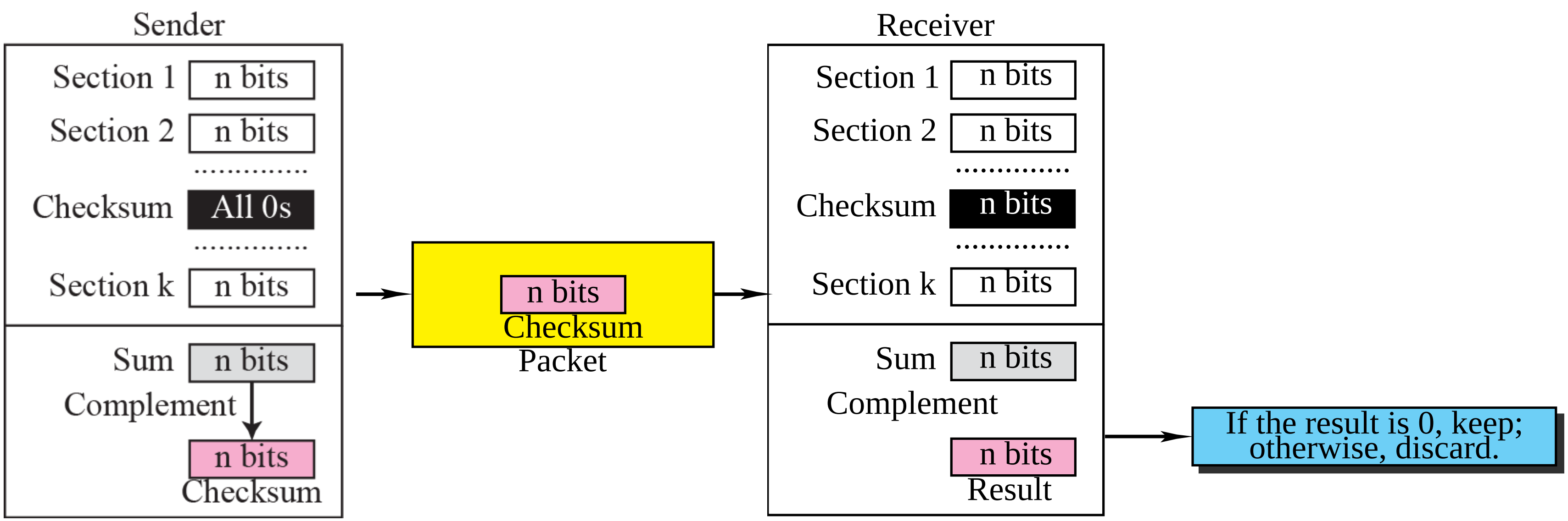
Which of the six options are used for datagram control and which for debugging and managements?

*Solution*

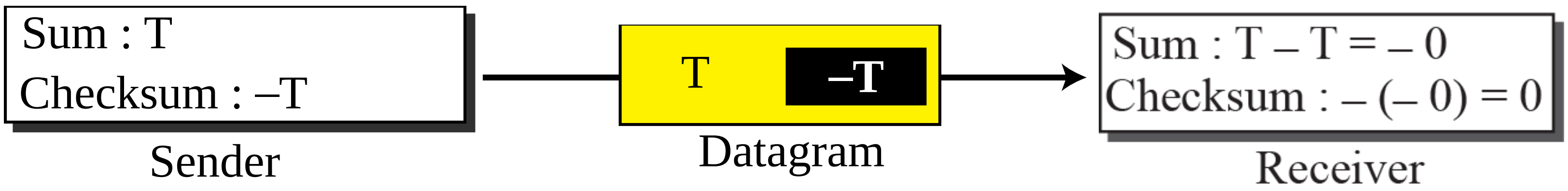
We look at the second and third (left-most) bits of the type.

a. No operation: type is 00000001; datagram control.
b. End of option: type is 00000000; datagram control.
c. Record route: type is 00000111; datagram control.
d. Strict source route: type is 10001001; datagram control.
e. Loose source route: type is 10000011; datagram control.
f. Timestamp: type is 01000100; debugging and management control.

# Figure 7.22 *Checksum concept*



## **Figure 7.23** *Checksum in one's complement arithmetic*



**Note**

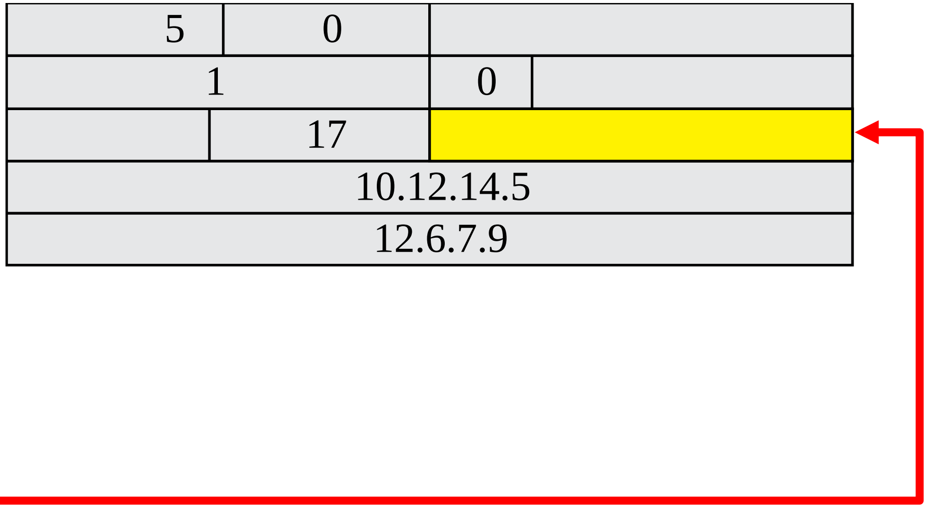
***Checksum in IP covers only the header, not the data.***

# Example 7.17

Figure 7.24 shows an example of a checksum calculation at the sender site for an IP header without options. The header is divided into 16-bit sections. All the sections are added and the sum is complemented. The result is inserted in the checksum field.

# **Figure 7.24** *Example of checksum calculation at the sender*

| | | |
|---|---|---|
| 4, 5, and 0 → | 01000101 | 00000000 |
| 28 → | 00000000 | 00011100 |
| 1 → | 00000000 | 00000001 |
| 0 and 0 → | 00000000 | 00000000 |
| 4 and 17 → | 00000100 | 00010001 |
| 0 → | 00000000 | 00000000 |
| 10.12 → | 00001010 | 00001100 |
| 14.5 → | 00001110 | 00000101 |
| 12.6 → | 00001100 | 00000110 |
| 7.9 → | 00000111 | 00001001 |
| Sum → | **01110100** | **01001110** |
| Checksum → | 10001011 | 10110001 |



# Example 7.18

Figure 7.25 shows the checking of checksum calculation at the receiver site (or intermediate router) assuming that no errors occurred in the header. The header is divided into 16-bit sections. All the sections are added and the sum is complemented. Since the result is 16 0s, the packet is accepted.

# Figure 7.25 *Example of checksum calculation at the receiver*

| | | | |
|---|---|---|---|
| 4 | 5 | 0 | 28 |
| 1 | | 0 | 0 |
| 4 | 17 | 35761 | |
| 10.12.14.5 | | | |
| 12.6.7.9 | | | |

| | | |
|---|---|---|
| 4, 5, and 0 → | 01000101 | 00000000 |
| 28 → | 00000000 | 00011100 |
| 1 → | 00000000 | 00000001 |
| 0 and 0 → | 00000000 | 00000000 |
| 4 and 17 → | 00000100 | 00010001 |
| Checksum → | **10001011** | **10110001** |
| 10.12 → | 00001010 | 00001100 |
| 14.5 → | 00001110 | 00000101 |
| 12.6 → | 00001100 | 00000110 |
| 7.9 → | 00000111 | 00001001 |
| Sum → | **1111 1111** | **1111 1111** |
| Checksum → | **0000 0000** | **0000 0000** |

# Topics Discussed in the Section

- ✓ **Header-Adding Module**
- ✓ **Processing Module**
- ✓ **Queues**
- ✓ **Routing Table**
- ✓ **Forwarding Module**
- ✓ **MTU Table**
- ✓ **Fragmentation Module**
- ✓ **Reassembly Table**
- ✓ **Reassembly Module**

# Figure 7.29 *IP components*

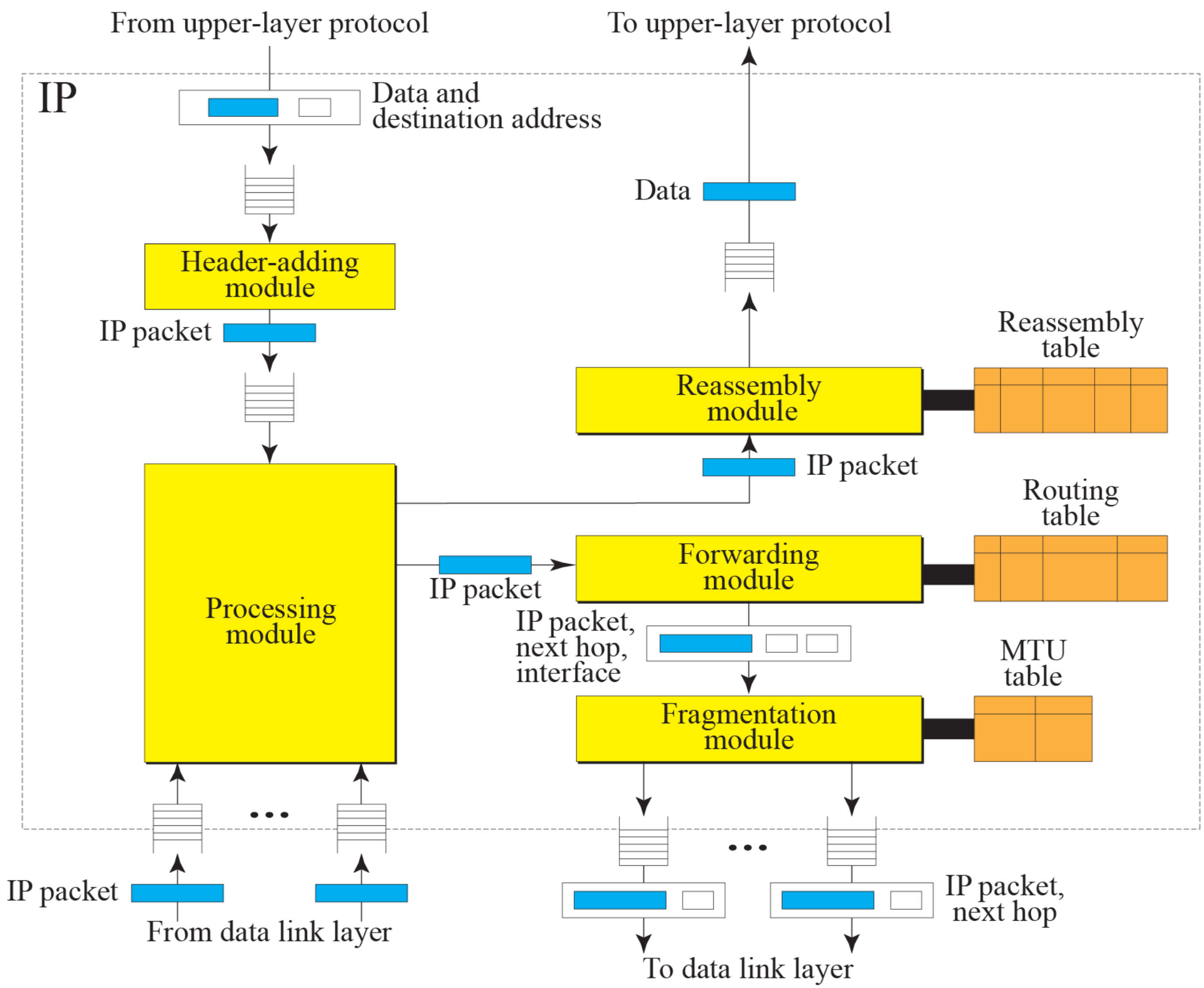


**Table 7.3** *Adding module*

```
IP_Adding_Module (data, destination_address)
{
    Encapsulate data in an IP datagram
    Calculate checksum and insert it in the checksum field
    Send data to the corresponding queue
    Return
}
```

**Table 7.4** *Processing module*

```
IP_Processing_Module (Datagram)
{
        Remove one datagram from one of the input queues.
        If (destination address matches a local address)
        {
            Send the datagram to the reassembly module.
            Return.
        }
        If (machine is a router)
        {
            Decrement TTL.
        }
        If (TTL less than or equal to zero)
        {
            Discard the datagram.
            Send an ICMP error message.
            Return.
        }
        Send the datagram to the forwarding module.
        Return.
}
```

**Table 7.5** *Fragmentation module*

```
IP_Fragmentation_Module (datagram)
{
    Extract the size of datagram
    If (size > MTU of the corresponding network)
    {
        If (D bit is set)
        {
            Discard datagram
            Send an ICMP error message
            return
        }
        Else
        {
            Calculate maximum size
            Divide the segment into fragments
            Add header to each fragment
            Add required options to each fragment
```

**Table 7.5** *Fragmentation module (continued)*

```
            Send fragment
            return
        }
    }
    Else
    {
        Send the datagram
    }
    Return.
}
```

## Figure 7.30 *Reassembly table*

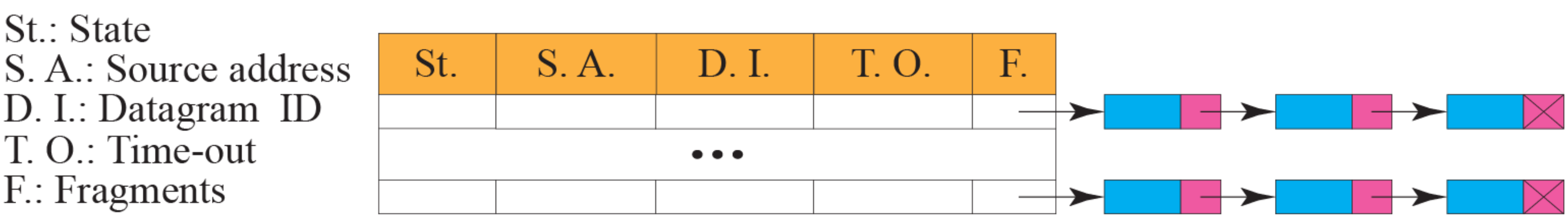


Table 7.6 *Reassembly module*

```
IP_Reassembly_Module (datagram)
{
      If (offset value = 0 AND M = 0)
      {
            Send datagram to the appropriate queue
            Return
      }
      Search the reassembly table for the entry
      If (entry not found)
      {
            Create a new entry
      }
      Insert datagram into the linked list
      If (all fragments have arrived)
      {
            Reassemble the fragment
            Deliver the fragment to upper-layer protocol
            return
      }
      Else
      {
            If (time-out expired)
            {
                  Discard all fragments
                  Send an ICMP error message
            }
      }
      Return.
}
```