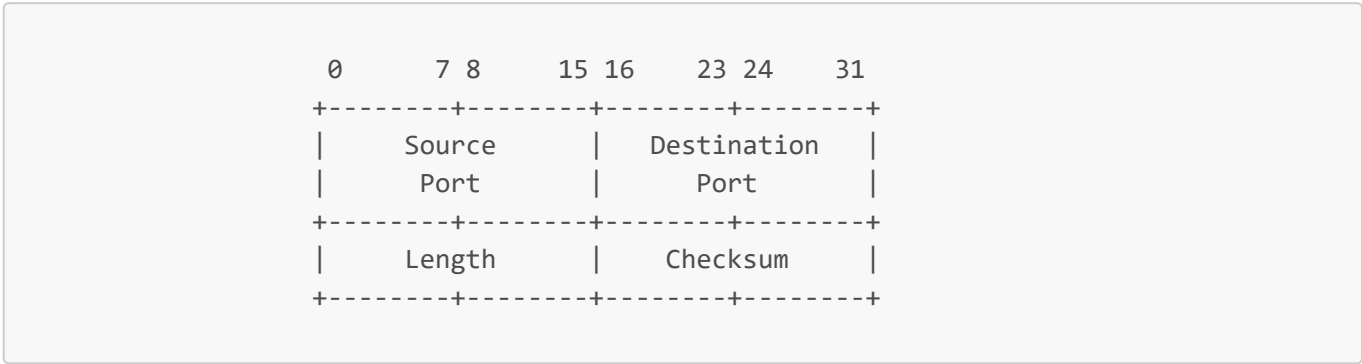
Q1 - Checksum

Introduction

In computer networking, the User Datagram Protocol (UDP) is one of the core communication protocols to send messages (transported as datagrams in packets) to other hosts on an Internet Protocol (IP) network.

A UDP datagram consists of a datagram **header** followed by a **data section** (the payload data for the application). The UDP datagram header consists of 4 fields, each of which is 2 bytes (16 bits):

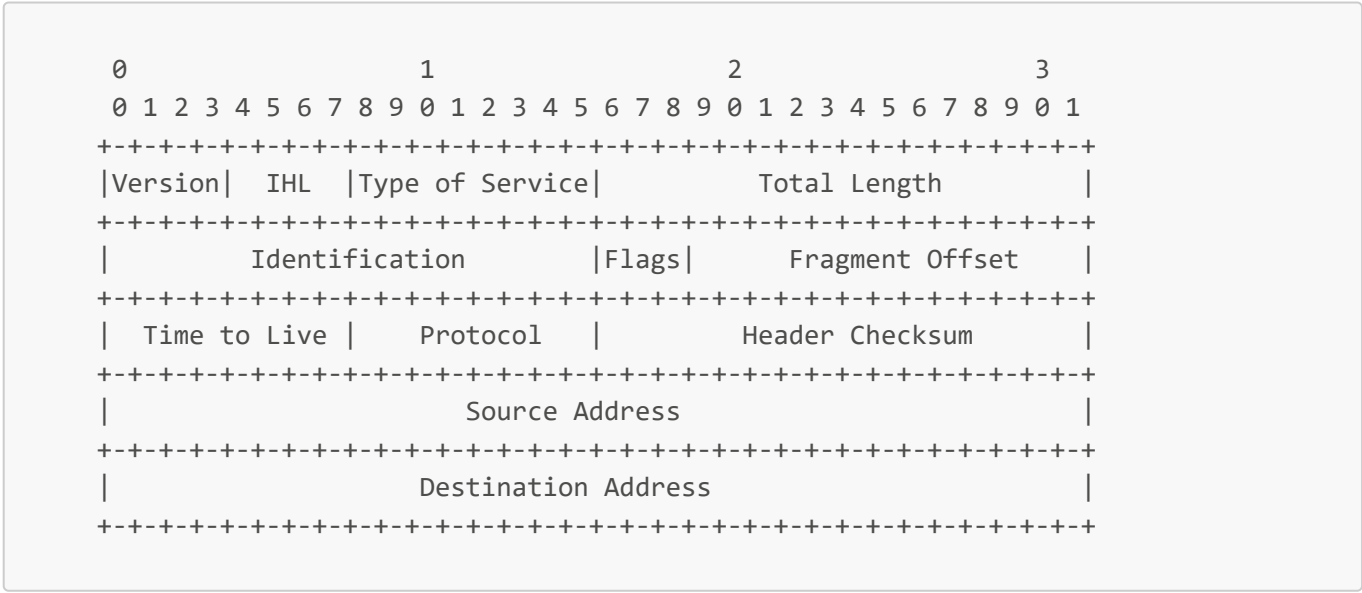


In this exercise, you need are required to calculate the `checksum` in a UDP header, as defined in RFC 768:

> Checksum is the 16-bit one's complement of the one's complement sum of a **pseudo header** of information from the **IP header**, the **UDP header**, and the **data**, padded with zero octets at the end (if necessary) to make a multiple of two octets.

For simplication purposes, you do not need to consider padding zero situations.

The contents of an IPv4 header are shown below, with each tick mark representing one bit position (20 bytes in total).



Contents of an IPv4 pseudo header on the other hand, is a simplified version of IPv4 header:

```
|          source address           |
+--------+--------+--------+--------+
|        destination address        |
+--------+--------+--------+--------+
|  zero  |protocol|   UDP length    |
+--------+--------+--------+--------+
```

However, since UDP datagram does not include an IP psuedo header, you will need to extract the information from the IP header, with the following correspondences:

- `source address` - `source address` field in IP header
- `destination address` - `destination address` field in IP header
- `protocol` - `0x11` (protocol number of UDP)
- `UDP length` - total length of the UDP **header** and **data**

Example

To extract the pseudo header, take the following IPv4 header for example (values in hexadecimal representation):

```
4500 0073 0000 4000 4011 0000 c0a8 0001 c0a8 00c7
```

We can quickly extract source address and destination address:

- `source address` - `c0a8 0001`
- `destination address` - `c0a8 00c7`

To obtain `UDP length`, suppose the UDP datagram looks like this:

```
1f90 1f91 000a 0000 0123
```

- `000a` is UDP length (8 bytes in header + 2 bytes in data)

Thus, the pseudo header can be constructed as:

```
c0a8 0001 c0a8 00c7 0011 000a
```

To calculate the checksum, we first calculate the sum of each 16 bit value, note that checksum field in the UDP header is `0000`:

```
1f90 + 1f91 + 000a + 0123 + c0a8 + 0001 + c0a8 + 00c7 + 0011 + 000a = 1c281
```

Since the checksum should be 16-bit, but the calculation result is larger than 16-bit. To handle this situation, we truncate the carry `1` and add it back to the sum (wrap-around):

```
c281 + 1 = c282
```

If the result is still larger than 16-bit, repeat wrap-around until it becomes a 16-bit number. Finally, we take the ones' complement of this result as the checksum:

```
ffff - c282 = 3d7d
```

In this exercise, given an IPv4 header and an UDP datagram (both are in hexadecimal notation, IPv4 header is in fixed 20 bytes length, no spaces in input), please **calculate** the checksum of UDP header. Output should be printed in hexadecimal notation, and you do not need to omit the zeros.

We guarantee that:

- Length of UDP datagram does not exceed 60 bytes (but should still consider how much memory space is needed to store the input).
- `checksum` field in UDP header is set to zero.
- Wrap-around happens only once .
- All alphabetical letters are in lower case.

Sample Input 1:

```
45000073000040004011000c0a80001c0a800c71f901f91000a00000123
```

Sample Output 1:

```
0x00003d7d
```

The input will be given in a `.txt` file, hence you will need to use Mars and its file I/O functions. Below is a code snippet that reads `test.txt`, and store its content to memory space marked by `buffer`. The filename is `data.txt` in this exercise. You can find more details in Mars help manual.

```
.data
    filename: .asciiz "test.txt"
    fill: .space 3
    buffer: .space 400000

.text
main:
    # open file
    li $v0, 13
```

```
    la $a0, filename
    li $a1, 0 # 0 for read
    li $a2, 0
    syscall
    move $s6, $v0 # save file descriptor

    # read file
    li $v0,14
    move $a0,$s6 # load file descriptor
    la $a1, buffer # save read content to buffer space
    li $a2, 40 # reads 40 ascii chars
    syscall
```