```
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
// IP parser code:

header_type ipv4_base_t {
    fields {
        bit<4>  version;
        bit<4>  ihl;
        bit<8>  diffserv;
        bit<16> totalLen;
        bit<16> identification;
        bit<3>  flags;
        bit<13> fragOffset;
        bit<8>  ttl;
        bit<8>  protocol;
        bit<16> hdrChecksum;
        bit<32> srcAddr;
        bit<32> dstAddr;
    }
}

#define OPTION_A_TYPE 0xAA
header_type ipv4_option_A {
    fields {
        // Boilerplate
        bit<8> type;
        bit<8> length;

        // Option A fields
        bit<32> foo;
    }
}

#define OPTION_B_TYPE 0xBB
header_type ipv4_option_B {
    fields {
        // Boilerplate
        bit<8> type;
    }
}

#define OPTION_PADDING_TYPE 0x00
header_type ipv4_option_padding {
    fields {
        bit<8> pad;
    }
}

struct_type ipv4_t {
```

```
    header ipv4_base_t base;
    header ipv4_option_A option_A;
    header ipv4_option_B option_B;
    header ipv4_option_padding padding[4];
}

module_type ipv4_parser_t (struct ipv4_t ipv4) {

    ////////////////////////////////////////
    ///// Core machinery

    local bit<8> ipv4_byte_counter;
    local bit test_result;

    // Entrypoint:
    parser parse {
        extract (ipv4.base);
        set_metadata(locals.ipv4_byte_counter, ipv4.base.ihl*4 - 20);
        return select(ipv4.base.ihl) {
            0: reject;
            1: reject;
            2: reject;
            3: reject;
            4: reject;

            5: accept;

            default: options_loop;
        }
    }

    parser options_loop {
        set_metadata(locals.test_result, ipv4_byte_counter<1);
        return select(locals.test_result) {
            true: accept;
            false: decode_option_type;
        }
    }

    parser decocde_option_type {
        return select(current(0,8)) {
            OPTION_A_TYPE : parse_option_a;
            OPTION_B_TYPE : parse_option_b;
            OPTION_PADDING_TYPE : parse_option_padding;
            default : reject;
        }
    }

    ////////////////////////////////////////
    ///// Individual option states
```

```
    parser parse_option_a {
        extract(ipv4.option_A);
        set_metadata(locals.ipv4_byte_counter,ipv4_byte_counter-6);
        return options_loop;
    }

    parser parse_option_b {
        extract(ipv4.option_B);
        set_metadata(locals.ipv4_byte_counter,ipv4_byte_counter-1);
        return options_loop;
    }

    parser parse_option_padding {
        extract(ipv4.padding[next]);
        set_metadata(locals.ipv4_byte_counter,ipv4_byte_counter-1);
        return options_loop;
    }

}

/////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////
// Main program:

header ethernet_t ethernet;
struct ipv4_t ipv4;
header tcp_t tcp;
header udp_t udp;

parser example_entry {
    extract(ethernet);
    return select (ethernet.etherType) {
        ETHERTYPE_IPV4 : parse_ipv4;
        default : accept;
    }
}

module ipv4_parser_t ipv4_parser (ipv4);
parser parse_ipv4 {
    ipv4_parser.parse();
    return select (ipv4.base.protocol) {
        IP_PROTO_UDP : parse_udp;
        IP_PROTO_TCP : parse_tcp;
        default: accept
    }
}

parser parse_udp {
    extract(udp);
```

```
    return accept;
}

parser parse_tcp {
    extract(tcp);
    return accept;
}
```