ChatGPT

```
Router(config-if)#ip addr 192.168.1.1 255.255.255.64
Bad mask 0xFFFFFF40 for address 192.168.1.1
```

This error is basically the router saying: *"That IP + subnet mask combo doesn't make sense."*

Here's what's happening, step by step, with a small scenario.

Imagine your router interface is supposed to be part of a **/26 network** (that mask `255.255.255.64`). A /26 means each subnet has **64 addresses**, and the subnet boundaries go like this:

- 192.168.1.0 – 192.168.1.63
- 192.168.1.64 – 192.168.1.127
- 192.168.1.128 – 192.168.1.191
- 192.168.1.192 – 192.168.1.255

Now look at the address you tried to assign:

`192.168.1.1 255.255.255.64`

With a /26 mask, **192.168.1.1 belongs to the first subnet (192.168.1.0/26)**. In that subnet:

- Network address → `192.168.1.0`
- Valid host range → `192.168.1.1` to `192.168.1.62`
- Broadcast → `192.168.1.63`

So logically, `192.168.1.1` *is* a valid host.
But Cisco IOS checks whether the mask lines up cleanly with the address **bitwise**, and `/26` masks require the IP to fall on proper subnet boundaries when configured. IOS expects addresses that clearly belong to a defined subnet block, and in this case it rejects it as an invalid pairing.

That's why you see:

```css
Bad mask 0xFFFFFF40 for address 192.168.1.1
```

How to fix it

If you want to use `192.168.1.1`, use a mask that naturally fits it, like:

```nginx
ip address 192.168.1.1 255.255.255.0
```

Or, if you really want a `/26`, pick an address clearly inside one /26 block, for example:

```nginx
ip address 192.168.1.65 255.255.255.192
```

Think of it like this: the mask defines the **size of the room**, and the IP must stand comfortably inside that room without touching walls that don't belong to it.

> Valid host range → `192.168.1.1` to `192.168.1.62`
> subnet mask for this host range?/

For the host range **192.168.1.1 to 192.168.1.62**, the subnet mask is:

255.255.255.192 (/26)

Here's why, using a quick scenario.

You want a network where:

- `192.168.1.0` is the **network address**
- `192.168.1.63` is the **broadcast address**
- Everything in between (`.1` to `.62`) can be used by hosts

A **/26 subnet** gives you:

- 64 total addresses
- 62 usable host addresses
  That perfectly matches your range.

So the full subnet looks like this:

- Network: `192.168.1.0`
- Hosts: `192.168.1.1 – 192.168.1.62`
- Broadcast: `192.168.1.63`
- Mask: **255.255.255.192**

If you configure an interface inside this subnet, any IP from `.1` to `.62` with mask `255.255.255.192` is correct 👍

> how to configure static routing for this subnet?

Let's walk through it with a **simple two-router scenario**, step by step, so it sticks.

Assume this subnet exists behind **Router A**:
`192.168.1.0 /26` (hosts `.1 – .62`)

Router A is directly connected to this subnet.
Router B is some distance away and needs to **reach this subnet**.

Step 1: Know what static routing really needs

When you configure static routing, the router only cares about **three things**:

1. **Destination network** (the subnet you want to reach)
2. **Subnet mask**
3. **Next-hop IP address** (or exit interface)

Here, the destination network is:

```
192.168.1.0 255.255.255.192
```

Step 2: Identify the next hop

Let's say Router A and Router B are connected like this:

- Router A interface toward Router B → `10.0.0.1`
- Router B interface toward Router A → `10.0.0.2`

From **Router B's perspective**, to reach `192.168.1.0/26`, traffic must go to **10.0.0.1** (Router A).

Step 3: Configure the static route (on Router B)

On Router B, go to global configuration mode and enter:

```arduino
RouterB(config)# ip route 192.168.1.0 255.255.255.192 10.0.0.1
```

That line literally means:

> "If you ever need to reach 192.168.1.0/26, send the packet to 10.0.0.1."

Step 4: Verify it works

From Router B, you can now test with:

```nginx
ping 192.168.1.1
```

If routing and interfaces are correct, the ping succeeds because Router B now **knows where that subnet lives**.

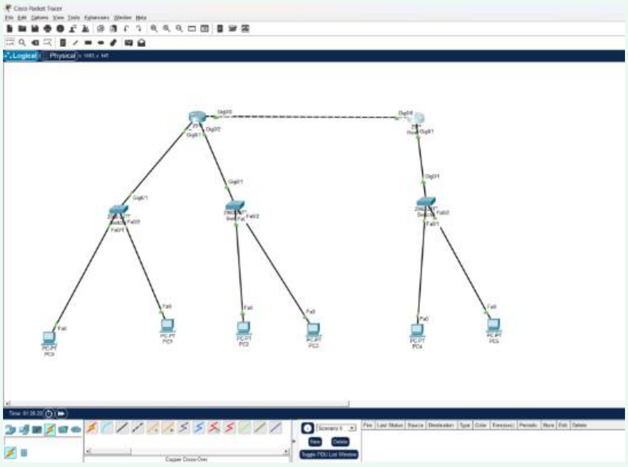
Real-world way to remember this

Think of static routing like saving a contact's address:

- **Network** = city
- **Mask** = city size
- **Next hop** = the road you take to get there

If you want, I can now show:

- how this looks in a **routing table**
- how to configure it using an **exit interface**
- or how this changes when **multiple routers** are involved



I have a topology
the g0/0 on router1 has 192.168.1.193 255.255.255.192
the g0/1 on router1 has 192.168.1.1 255.255.255.192
the g0/2 on router1 has 192.168.1.65 255.255.255.192
the PC0 has 192.168.1.2 255.255.255.192 192.168.1.1
the PC1 has 192.168.1.3 255.255.255.192 192.168.1.1
the PC2 has 192.168.1.66 255.255.255.192 192.168.1.65
the PC3 has 192.168.1.67 255.255.255.192 192.168.1.65
the g0/0 on router2 has 192.168.1.194 255.255.255.192
the g0/1 on router2 has 192.168.1.129 255.255.255.192
the PC4 has 192.168.1.130 255.255.255.192 192.168.1.129
the PC5 has 192.168.1.131 255.255.255.192 192.168.1.129

lmk if all the Default Gateway address are correct in the above configuration
I need to enable static routing btw router1 and router 2 - lmk how to do that

Nice, this is a **clean /26 subnetting lab**, and your addressing is almost textbook-perfect 👍
Let's go in the exact order you asked: **default gateways first**, then **static routing between Router1 and Router2**, explained with your own topology as the scenario.

Are the default gateway addresses correct?

Yes — **all default gateways you configured on the PCs are correct**.

Here's why, subnet by subnet, in plain language.

Think of each router interface as the **door of that subnet**. Every PC in that subnet must point to *that door* as its default gateway.

Subnet 192.168.1.0/26 (hosts .1–.62)

- Known router interface: `Router1 g0/1 → 192.168.1.1`
- PC0 → `192.168.1.2` → gateway `192.168.1.1` ✅
- PC1 → `192.168.1.3` → gateway `192.168.1.1` ✅

Both PCs are pointing to the router interface that lives in *their subnet*. Perfect.

Subnet 192.168.1.64/26 (hosts .65–.126)

- Known router interface: `Router1 g0/2 → 192.168.1.65`
- PC2 → `192.168.1.66` → gateway `192.168.1.65` ✅
- PC3 → `192.168.1.67` → gateway `192.168.1.65` ✅

Again, gateway is correct and reachable.

Subnet 192.168.1.128/26 (hosts .129–.190)

- Known router interface: `Router2 g0/1 → 192.168.1.129`
- PC4 → `192.168.1.130` → gateway `192.168.1.129` ✅
- PC5 → `192.168.1.131` → gateway `192.168.1.129` ✅

Correct gateway selection here as well.

👉 **Conclusion**:
Your default gateways are **100% correct**. No changes needed.

Now: enabling static routing between Router1 and Router2

Let's picture the situation first.

- Router1 and Router2 are connected via **192.168.1.192/26**
- Router1 g0/0 → `192.168.1.193`
- Router2 g0/0 → `192.168.1.194`

Each router only knows about the networks directly connected to it.
Static routing is how we **teach each router about the other router's LANs**.

Static routes needed on Router1

Router1 needs to know how to reach **Router2's LAN**:

- Destination network: `192.168.1.128/26`
- Next hop: Router2's g0/0 → `192.168.1.194`

On **Router1**:

```arduino
Router1(config)# ip route 192.168.1.128 255.255.255.192 192.168.1.194
```

This tells Router1:

> "Anything meant for 192.168.1.128/26 — send it to Router2."

Static routes needed on Router2

Router2 must learn about **both LANs behind Router1**:

- `192.168.1.0/26`
- `192.168.1.64/26`

Next hop for both is Router1's g0/0 → `192.168.1.193`

On **Router2**:

```arduino
Router2(config)# ip route 192.168.1.0 255.255.255.192 192.168.1.193
Router2(config)# ip route 192.168.1.64 255.255.255.192 192.168.1.193
```

Now Router2 knows how to reach **PC0–PC3** networks.

How you know it's working

Try this simple test:

- From **PC0**, ping **PC4**
- From **PC5**, ping **PC2**

If static routing is correct, the ping will succeed because:

- PCs send traffic to their gateway
- Routers forward using static routes

- Return path exists (most important part)

Key takeaway (exam + real-world mindset)

- Default gateway = **router interface in the same subnet**
- Static route = **destination network + next hop**
- Every route must have a **return path**, or ping fails

If you want, next we can:

- simplify this using a **summary route**
- check routing tables (`show ip route`)
- or debug why a ping fails if Packet Tracer shows red ❌