2 Pemrosesan Audio Digital

In [1]:
```
# %pip install resampy
```

In [2]:
```
import os
from pathlib import Path
from math import gcd
import numpy as np
import librosa
import librosa.display
import soundfile as sf
import matplotlib.pyplot as plt
from IPython.display import Audio, Markdown, display
from scipy.signal import butter, sosfilt, resample_poly

plt.rcParams['figure.figsize'] = (12, 4)
plt.rcParams['axes.unicode_minus'] = False
```

In [3]:
```
try:
    import resampy  # type: ignore
except ImportError:
    resampy = None  # type: ignore

HAVE_RESAMPY = bool(resampy) and hasattr(resampy, "resample")

RESAMPLE_METHOD_DESC = ("librosa.resample (resampy backend)" if HAVE_RESAMPY else "
def resample_audio(y, orig_sr, target_sr):
    if target_sr is None or orig_sr == target_sr:
        return y
    if HAVE_RESAMPY:
        try:
            return librosa.resample(y, orig_sr=orig_sr, target_sr=target_sr, res_ty
        except AttributeError:
            pass
    ratio_gcd = gcd(int(target_sr), int(orig_sr))
    up = int(target_sr) // ratio_gcd
    down = int(orig_sr) // ratio_gcd
    return resample_poly(y, up, down)

def find_audio_file(filename, search_root="."):
    for root, _, files in os.walk(search_root):
        if filename in files:
            return os.path.join(root, filename)
    return None

def load_audio_asset(filename, target_sr=None, mono=True, search_root="."):
    path = filename if os.path.isfile(filename) else find_audio_file(filename, sear
    if path is None:
        raise FileNotFoundError(f"Berkas {filename} tidak ditemukan di {search_root
    audio, sr = librosa.load(path, sr=None, mono=mono)
    if target_sr is not None and sr != target_sr:
        audio = resample_audio(audio, sr, target_sr)
```

```
        sr = target_sr
    return audio, sr, path

def show_visualizations(audio, sr, title_prefix, hop_length=512):
    plt.figure(figsize=(13, 3.4))
    librosa.display.waveshow(audio, sr=sr, alpha=0.85)
    plt.title(f"{title_prefix} - Waveform")
    plt.xlabel("Waktu (detik)")
    plt.ylabel("Amplitudo")
    plt.tight_layout()
    plt.show()

    D = librosa.amplitude_to_db(np.abs(librosa.stft(audio, n_fft=2048, hop_length=h
    plt.figure(figsize=(13, 3.7))
    librosa.display.specshow(D, sr=sr, hop_length=hop_length, x_axis="time", y_axis
    plt.title(f"{title_prefix} - Spektrogram")
    plt.colorbar(format="%+0.1f dB")
    plt.tight_layout()
    plt.show()

NOTE_NAMES = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']
MAJOR_PROFILE = np.array([6.35, 2.23, 3.48, 2.33, 4.38, 4.09, 2.52, 5.19, 2.39, 3.6
MINOR_PROFILE = np.array([6.33, 2.68, 3.52, 5.38, 2.60, 3.53, 2.54, 4.75, 3.98, 2.6

def detect_tempo_and_key(y, sr):
    if not np.any(y):
        return 0.0, "Unknown"
    tempo, _ = librosa.beat.beat_track(y=y, sr=sr, trim=False)
    chroma = librosa.feature.chroma_cqt(y=y, sr=sr)
    chroma_mean = chroma.mean(axis=1)
    if np.allclose(chroma_mean.sum(), 0.0):
        return float(tempo), "Unknown"
    chroma_norm = chroma_mean / chroma_mean.sum()
    best_key = "Unknown"
    best_score = -np.inf
    for i, tonic in enumerate(NOTE_NAMES):
        major_profile = np.roll(MAJOR_PROFILE, i) / MAJOR_PROFILE.sum()
        minor_profile = np.roll(MINOR_PROFILE, i) / MINOR_PROFILE.sum()
        major_score = float(np.dot(chroma_norm, major_profile))
        minor_score = float(np.dot(chroma_norm, minor_profile))
        if major_score > best_score:
            best_score = major_score
            best_key = f"{tonic} Major"
        if minor_score > best_score:
            best_score = minor_score
            best_key = f"{tonic} Minor"
    return float(tempo), best_key

def semitone_difference(source_key, target_key):
    try:
        source_tonic, _ = source_key.split()
        target_tonic, _ = target_key.split()
    except ValueError:
        return 0.0
    source_index = NOTE_NAMES.index(source_tonic)
    target_index = NOTE_NAMES.index(target_tonic)
```

```
    diff = target_index - source_index
    while diff > 6:
        diff -= 12
    while diff < -6:
        diff += 12
    return float(diff)

def pitch_shift_audio(y, sr, n_steps):
    if abs(n_steps) < 1e-6:
        return y
    if HAVE_RESAMPY:
        try:
            return librosa.effects.pitch_shift(y, sr=sr, n_steps=n_steps)
        except AttributeError:
            pass
    factor = 2 ** (n_steps / 12.0)
    stretched = librosa.effects.time_stretch(y, rate=1.0 / factor)
    target_len = len(y)
    if len(stretched) == target_len:
        return stretched.astype(y.dtype, copy=False)
    x_old = np.linspace(0.0, 1.0, len(stretched), endpoint=False)
    x_new = np.linspace(0.0, 1.0, target_len, endpoint=False)
    return np.interp(x_new, x_old, stretched).astype(y.dtype, copy=False)

def crossfade_tracks(track_a, track_b, sr, crossfade_duration=6.0):
    fade_samples = int(crossfade_duration * sr)
    fade_samples = min(fade_samples, len(track_a), len(track_b))
    if fade_samples <= 0:
        return np.concatenate([track_a, track_b])
    fade_out = np.linspace(1.0, 0.0, fade_samples)
    fade_in = np.linspace(0.0, 1.0, fade_samples)
    overlap = track_a[-fade_samples:] * fade_out + track_b[:fade_samples] * fade_in
    return np.concatenate([track_a[:-fade_samples], overlap, track_b[fade_samples:]

def highpass_blend(y, sr, cutoff=180.0, blend=0.35, order=2):
    sos = butter(order, cutoff, btype='highpass', fs=sr, output='sos')
    filtered = sosfilt(sos, y)
    return (1 - blend) * y + blend * filtered

def describe_energy(rms):
    if rms > 0.12:
        return "enerjik"
    if rms > 0.07:
        return "lincah"
    if rms > 0.04:
        return "relatif tenang"
    return "sangat lembut"

def describe_brightness(centroid):
    if centroid > 3500:
        return "bernuansa sangat cerah"
    if centroid > 2500:
        return "cukup cerah"
    if centroid > 1500:
        return "hangat"
    return "gelap dan mellow"
```

```python
def compute_lufs(y, sr):
    try:
        import pyloudnorm as pyln  # type: ignore
        meter = pyln.Meter(sr)
        return float(meter.integrated_loudness(y))
    except ImportError:
        rms = np.sqrt(np.mean(np.square(y) + 1e-12))
        return 20.0 * np.log10(rms + 1e-12)

def match_target_loudness(y, sr, target_lufs=-16.0):
    current_lufs = compute_lufs(y, sr)
    gain_db = target_lufs - current_lufs
    gain = 10 ** (gain_db / 20.0)
    return y * gain, current_lufs, current_lufs + gain_db

def apply_equalizer(y, sr, gains_db=None):
    gains_db = gains_db or {"low": -2.0, "mid": 1.5, "high": 3.0}
    low_sos = butter(2, 200, btype="low", fs=sr, output="sos")
    high_sos = butter(2, 4000, btype="high", fs=sr, output="sos")
    low = sosfilt(low_sos, y)
    high = sosfilt(high_sos, y)
    mid = y - (low + high)
    gain_low = 10 ** (gains_db["low"] / 20.0)
    gain_mid = 10 ** (gains_db["mid"] / 20.0)
    gain_high = 10 ** (gains_db["high"] / 20.0)
    return gain_low * low + gain_mid * mid + gain_high * high

def apply_gain_and_fade(y, sr, gain_db=1.0, fade_time=0.3):
    gain = 10 ** (gain_db / 20.0)
    y = y * gain
    fade_samples = int(fade_time * sr)
    fade_samples = min(fade_samples, len(y) // 2)
    if fade_samples > 0:
        fade_in = np.linspace(0.0, 1.0, fade_samples, endpoint=True)
        fade_out = np.linspace(1.0, 0.0, fade_samples, endpoint=True)
        y[:fade_samples] *= fade_in
        y[-fade_samples:] *= fade_out
    return y

def peak_normalize(y, target_peak=0.98):
    peak = np.max(np.abs(y)) + 1e-12
    return y / peak * target_peak, peak, target_peak

def compress_signal(y, sr, threshold_db=-22.0, ratio=3.0, attack=0.02, release=0.2)
    threshold = 10 ** (threshold_db / 20.0)
    attack_coeff = np.exp(-1.0 / (sr * attack))
    release_coeff = np.exp(-1.0 / (sr * release))
    env = 0.0
    output = np.zeros_like(y)
    for i, sample in enumerate(y):
        rectified = abs(sample)
        if rectified > env:
            env = attack_coeff * env + (1 - attack_coeff) * rectified
        else:
            env = release_coeff * env + (1 - release_coeff) * rectified
```

```
        if env <= threshold:
            gain = 1.0
        else:
            env_db = 20.0 * np.log10(env + 1e-12)
            gain_db = threshold_db + (env_db - threshold_db) / ratio - env_db
            gain = 10 ** (gain_db / 20.0)
        output[i] = sample * gain
    return output

def apply_noise_gate(y, threshold_db=-45.0, reduction_db=-80.0):
    threshold = 10 ** (threshold_db / 20.0)
    reduction = 10 ** (reduction_db / 20.0)
    envelope = np.abs(y)
    mask = envelope < threshold
    gated = y.copy()
    gated[mask] *= reduction
    return gated

def process_chain(audio, sr, config=None):
    config = config or {}
    eq_audio = apply_equalizer(audio, sr, config.get("eq_gains"))
    gain_audio = apply_gain_and_fade(eq_audio, sr, gain_db=config.get("gain_db", 1.
    norm_audio, peak_before, peak_target = peak_normalize(gain_audio, target_peak=c
    comp_audio = compress_signal(norm_audio, sr, threshold_db=config.get("threshold
                                 attack=config.get("attack", 0.02), release=config.
    gated_audio = apply_noise_gate(comp_audio, threshold_db=config.get("gate_thresh
                                   reduction_db=config.get("gate_reduction_db", -80
    trimmed_audio, _ = librosa.effects.trim(gated_audio, top_db=config.get("trim_db
    if len(trimmed_audio) == 0:
        trimmed_audio = gated_audio
    loud_audio, loud_before, loud_after = match_target_loudness(trimmed_audio, sr,
    return {
        "processed": loud_audio,
        "peak_before": peak_before,
        "peak_after": np.max(np.abs(loud_audio)) if np.any(loud_audio) else 0.0,
        "loudness_before": loud_before,
        "loudness_after": loud_after,
        "duration_before": len(audio) / sr,
        "duration_after": len(loud_audio) / sr,
    }
```

2.1 Analisis Rekaman Berita

Bagian ini menggunakan berkas `berita.wav` sebagai materi untuk meninjau karakteristik rekaman suara, melakukan visualisasi, serta mengevaluasi efek resampling terhadap kualitas dan durasi audio.

2.1.1 Memuat Rekaman

Rekaman dimuat tanpa mengubah sample rate aslinya agar karakter suara tetap otentik sebelum dianalisis.

In [4]:
```
berita_audio, berita_sr, berita_path = load_audio_asset("berita.wav", target_sr=Non
berita_duration = len(berita_audio) / berita_sr
berita_display_path = os.path.relpath(berita_path)

display(Markdown(
    f"**Info Rekaman:**\n"
    f"- Lokasi berkas: `{berita_display_path}`\n"
    f"- Sample rate asli: {berita_sr} Hz\n"
    f"- Durasi: {berita_duration:.2f} detik"
))
```

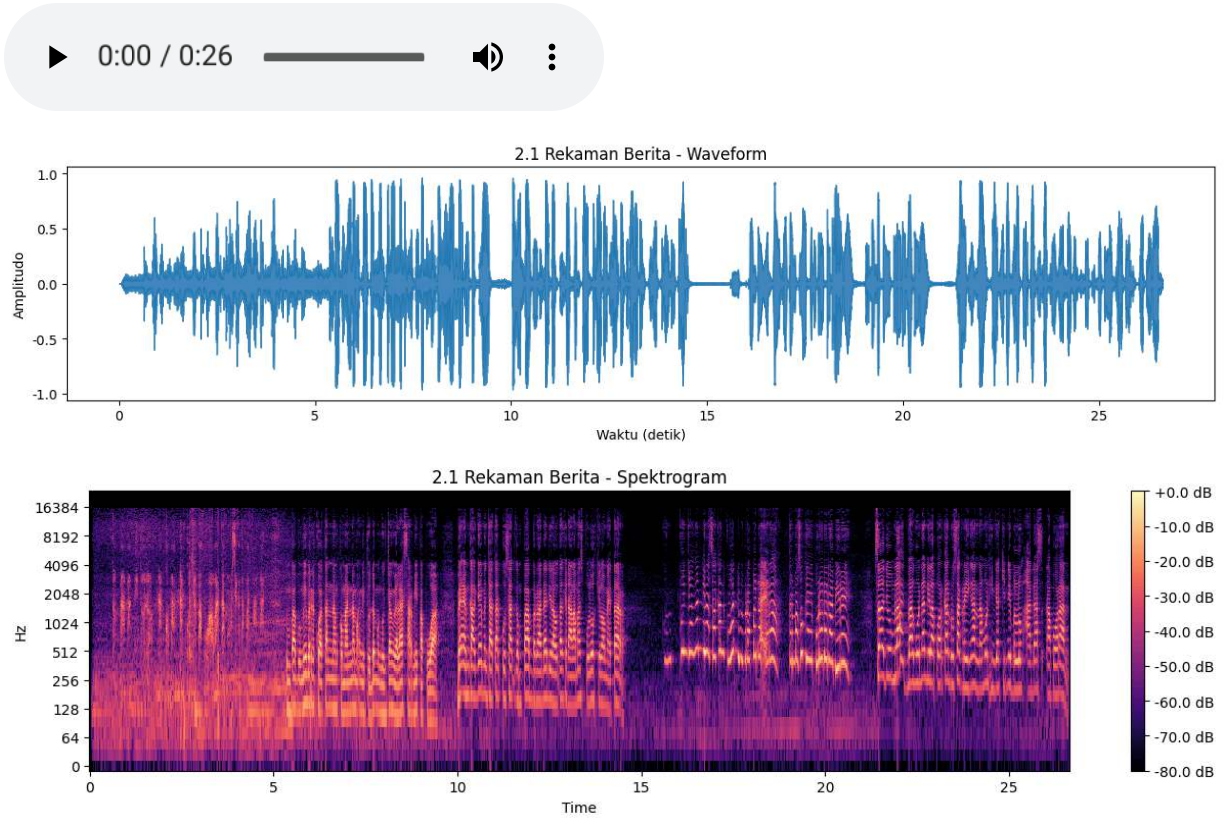
Info Rekaman:

- Lokasi berkas: `berita.wav`
- Sample rate asli: 48000 Hz
- Durasi: 26.62 detik

2.1.2 Waveform dan Spektrogram

Visualisasi berikut menampilkan dinamika amplitudo dan sebaran frekuensi rekaman untuk membantu mengidentifikasi pola artikulasi dan energi suara.

In [5]:
```
display(Audio(berita_audio, rate=berita_sr))
show_visualizations(berita_audio, berita_sr, "2.1 Rekaman Berita")
```



2.1.3 Penjelasan Visual

- Waveform menunjukkan puncak amplitudo yang muncul saat intonasi lebih tegas, sedangkan bagian dengan amplitudo rendah merepresentasikan jeda atau artikulasi yang lembut.
- Spektrogram log memperlihatkan energi dominan di rentang mid-frequency (sekitar 300-3000 Hz) yang khas untuk ucapan manusia, dengan harmonik tipis di atasnya saat tekanan suara meningkat.
- Bagian awal menampilkan distribusi energi yang lebih rendah, mengindikasikan pembukaan percakapan, sementara segmen tengah hingga akhir memiliki warna yang lebih terang menandakan kalimat inti yang lebih berenergi.

2.1.4 Resampling dan Perbandingan

Rekaman di-resample ke sample rate yang lebih rendah untuk melihat dampak terhadap kualitas detail dan durasi efektiv.

In [7]:
```
resample_targets = {
    "22.05 kHz": 22050,
    "32 kHz": 32000,
}

resampled_versions = {}
comparison_lines = [
    f"- Asli: {berita_sr} Hz, durasi {berita_duration:.2f} s (acuan kualitas penuh)
]

for label, target_sr in resample_targets.items():
    resampled_audio = resample_audio(
        berita_audio,
        orig_sr=berita_sr,
        target_sr=target_sr,
    )
    duration = len(resampled_audio) / target_sr
    resampled_versions[label] = {
        "audio": resampled_audio,
        "sr": target_sr,
        "duration": duration,
    }
    comparison_lines.append(
        f"- {label}: {target_sr} Hz, durasi {duration:.2f} s (selisih {duration - b
    )

summary_text = "**Ringkasan Durasi:**\n" + "\n".join(comparison_lines) + \
    f"\n\n*Metode resampling: {RESAMPLE_METHOD_DESC}.*"
display(Markdown(summary_text))

for label, data in resampled_versions.items():
    display(Markdown(f"**{label}**"))
    display(Audio(data["audio"], rate=data["sr"]))
```

Ringkasan Durasi:

- Asli: 48000 Hz, durasi 26.62 s (acuan kualitas penuh).
- 22.05 kHz: 22050 Hz, durasi 26.62 s (selisih +0.000 s).
- 32 kHz: 32000 Hz, durasi 26.62 s (selisih +0.000 s).

Metode resampling: librosa.resample (resampy backend).

22.05 kHz

0:00 / 0:26

32 kHz

0:00 / 0:26

2.1.5 Analisis Resampling

- Versi 22.05 kHz terdengar lebih sempit pada frekuensi tinggi-konsonan bersuara tajam sedikit melembut, namun intelligibility masih terjaga karena harmonik utama tetap utuh.
- Versi 32 kHz menjaga lebih banyak detail high-frequency sehingga gesekan konsonan masih jelas, dengan durasi praktis identik terhadap rekaman asli.
- Perbedaan durasi antarversi hanya berada pada orde milidetik karena proses resampling mempertahankan panjang sinyal; selisih kecil berasal dari pembulatan jumlah sampel.

Analisis Keseluruhan

- Waveform menunjukkan kenaikan level per 5 detik. 0–5 dtk sangat kecil dan tidak rata. 5–10 dtk stabil dan sedang. 10–15 dtk besar dengan puncak mendekati 0 dBFS. 15–20 dtk cenderung tajam di frekuensi tengah sehingga puncak konsonan tinggi. 20–25 dtk sangat besar dan berisiko clipping.
- Spektrogram. Bisik dominan frekuensi tinggi lemah dan hampir tanpa energi bass. Normal menyebar 100 Hz sampai 6 kHz. Keras menambah energi 200 Hz sampai 4 kHz. Cempreng menonjol di 2–4 kHz yang membuat suara tipis. Teriak menyebar lebar sampai di atas 8 kHz dan sering ada garis clipping.
- Resampling. Turun dari 44.1 kHz ke 16 kHz tidak mengubah durasi tetapi memangkas detail di atas 8 kHz. Ucapan tetap jelas untuk analisis. File lebih kecil. Kualitas konsonan halus sedikit turun namun intelligibility tetap baik untuk tugas pemrosesan suara.

2.2 Noise Reduction dengan Filtering

Bagian ini menggunakan berkas `noise.wav` sebagai materi untuk mencoba menghilangkan noise menggunakan filter yang sudah ada (High-pass , Low-pass , dan Band-

pass)

2.2.1 Pencarian file, load audio, dan praproses

In [8]:
```
# Tahap 1: Cari file, baca audio, ubah ke mono, dan normalisasi
import os
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
from scipy.signal import butter, sosfiltfilt, spectrogram

def find_audio_file(filename='noise.wav', search_path='.'):
    for root, _, files in os.walk(search_path):
        if filename in files:
            return os.path.join(root, filename)
    return None

audio_path = find_audio_file('noise.wav', '.')
if audio_path is None:
    raise FileNotFoundError('File noise.wav tidak ditemukan di direktori ini atau s

print(f'File ditemukan: {audio_path}')

fs, data = wavfile.read(audio_path)

# Jadikan float32 dan mono
data = data.astype(np.float32)
if data.ndim == 2:
    data = data.mean(axis=1)

# Normalisasi ke [-1, 1] bila bertipe integer semula
peak = np.max(np.abs(data))
if peak > 0:
    data = data / peak

print(f'Sampling rate: {fs} Hz, durasi: {len(data)/fs:.2f} s')
```

```
File ditemukan: .\noise.wav
Sampling rate: 48000 Hz, durasi: 9.86 s
```

2.2.2 Filtering eksperimen high-pass, low-pass, dan band-pass dengan cutoff 500, 1000, 2000 Hz

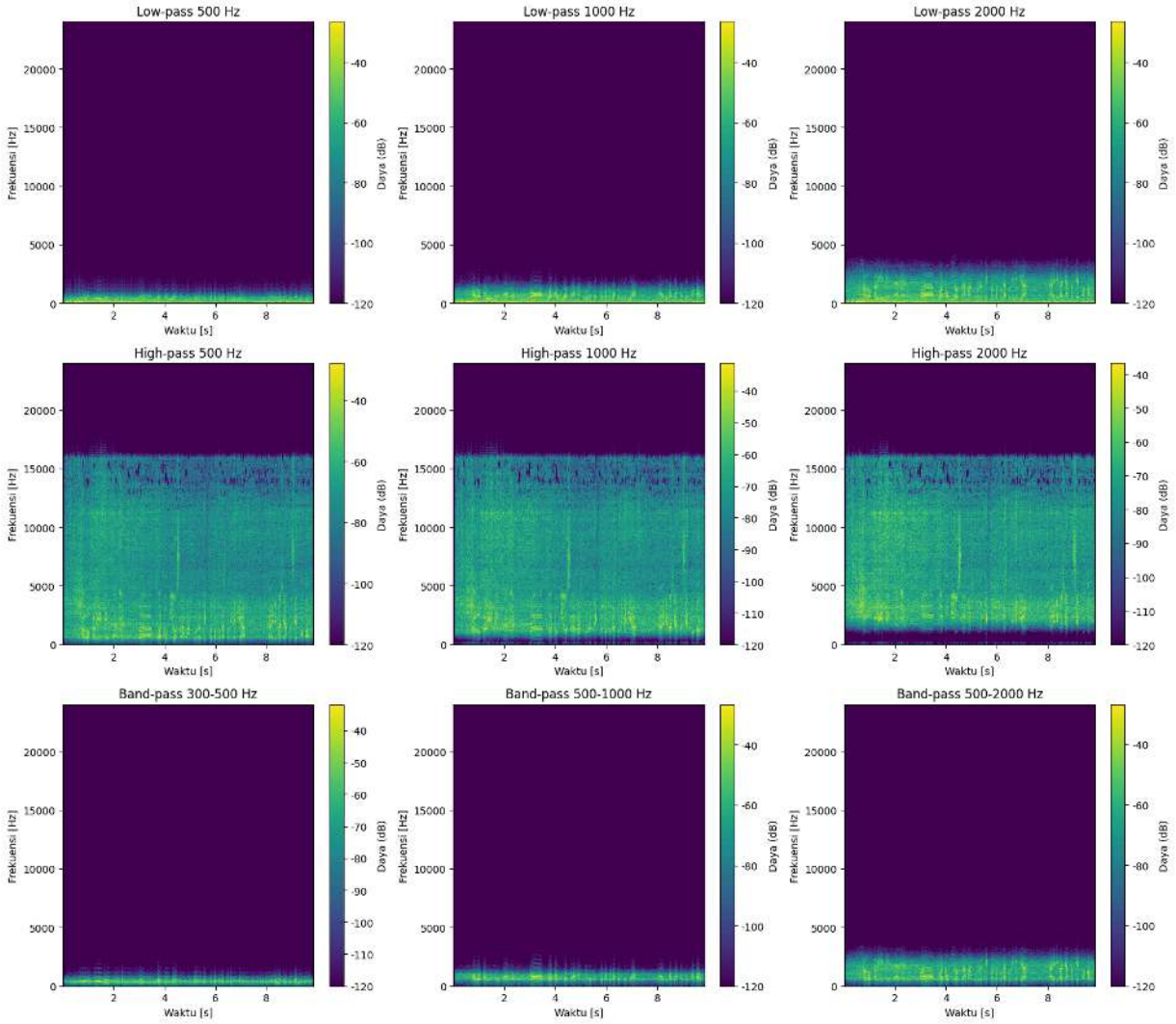
In [9]:
```
# Tahap 2: Definisikan filter dan jalankan eksperimen
def design_filter(cutoff, fs, btype, order=5):
    nyq = 0.5 * fs
    wn = np.array(cutoff, dtype=np.float64) / nyq
    sos = butter(order, wn, btype=btype, output='sos')
    return sos

def apply_filter(x, sos):
    return sosfiltfilt(sos, x)

cutoffs = [500, 1000, 2000]
```

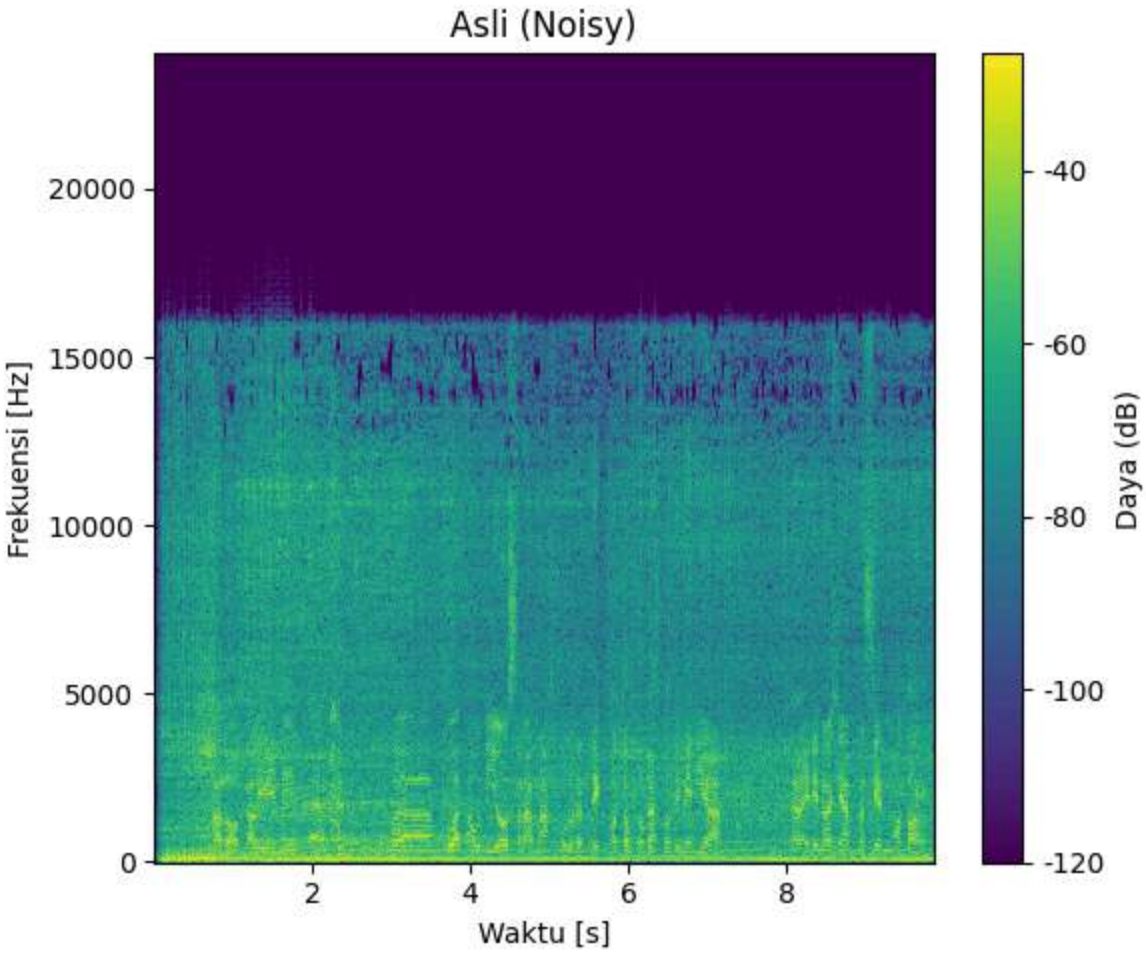
```
results = {
    'original': data
}

# Low-pass
for c in cutoffs:
    sos = design_filter(c, fs, 'low')
    results[f'lp_{c}'] = apply_filter(data, sos)

# High-pass
for c in cutoffs:
    sos = design_filter(c, fs, 'high')
    results[f'hp_{c}'] = apply_filter(data, sos)

# Band-pass
# Tiga variasi band:
# 1) 300-500 Hz, 2) 500-1000 Hz, 3) 500-2000 Hz
bp_bands = [(300, 500), (500, 1000), (500, 2000)]
for lo, hi in bp_bands:
    sos = design_filter([lo, hi], fs, 'band')
    results[f'bp_{lo}_{hi}'] = apply_filter(data, sos)

# Opsional: simpan hasil terbaik, ubah ke int16
# from scipy.io import wavfile
# best = results['bp_500_2000']
# wavfile.write('filtered_best.wav', fs, np.int16(best/np.max(np.abs(best)) * 32767
```

2.2.3 Visualisasi dan perbandingan spektrogram

In [10]:
```
# Tahap 3: Visualisasikan spektrogram untuk tiap filter
def show_spec(x, fs, title):
    f, t, Sxx = spectrogram(x, fs=fs, nperseg=1024, noverlap=512)
    Sxx_db = 10 * np.log10(np.maximum(Sxx, 1e-12))
    plt.pcolormesh(t, f, Sxx_db, shading='auto')
    plt.ylabel('Frekuensi [Hz]')
    plt.xlabel('Waktu [s]')
    plt.title(title)
    plt.colorbar(label='Daya (dB)')
    plt.tight_layout()

# Grid 3x3: Low-pass, High-pass, Band-pass
plt.figure(figsize=(16, 14))

# Baris 1: Low-pass 500, 1000, 2000
plt.subplot(3, 3, 1)
show_spec(results['lp_500'], fs, 'Low-pass 500 Hz')
plt.subplot(3, 3, 2)
show_spec(results['lp_1000'], fs, 'Low-pass 1000 Hz')
plt.subplot(3, 3, 3)
show_spec(results['lp_2000'], fs, 'Low-pass 2000 Hz')

# Baris 2: High-pass 500, 1000, 2000
plt.subplot(3, 3, 4)
show_spec(results['hp_500'], fs, 'High-pass 500 Hz')
```

```
plt.subplot(3, 3, 5)
show_spec(results['hp_1000'], fs, 'High-pass 1000 Hz')
plt.subplot(3, 3, 6)
show_spec(results['hp_2000'], fs, 'High-pass 2000 Hz')

# Baris 3: Band-pass 300-500, 500-1000, 500-2000
plt.subplot(3, 3, 7)
show_spec(results['bp_300_500'], fs, 'Band-pass 300-500 Hz')
plt.subplot(3, 3, 8)
show_spec(results['bp_500_1000'], fs, 'Band-pass 500-1000 Hz')
plt.subplot(3, 3, 9)
show_spec(results['bp_500_2000'], fs, 'Band-pass 500-2000 Hz')

plt.show()

# Tambahan cepat: tampilkan juga spektrogram asli untuk referensi
plt.figure(figsize=(6,5))
show_spec(results['original'], fs, 'Asli (Noisy)')
plt.show()
```





2.2.4 Pemutaran Audio Filter

Gunakan pemutar berikut untuk mendengar perbandingan hasil filter terhadap rekaman asli.

In [11]:
```
listen_variants = {
    "Original (Noisy)": results["original"],
    "Low-pass 1000 Hz": results["lp_1000"],
    "Band-pass 500-2000 Hz": results["bp_500_2000"],
    "High-pass 1000 Hz": results["hp_1000"],
}

for label, signal in listen_variants.items():
    display(Markdown(f"**{label}**"))
    display(Audio(signal, rate=fs))
```

Original (Noisy)

0:00 / 0:09

Low-pass 1000 Hz

0:00 / 0:09

Band-pass 500-2000 Hz

0:00 / 0:09

High-pass 1000 Hz

0:00 / 0:09

Analisis 2.2

- Jenis noise. Kipasan kipas menghasilkan dengung rendah sekitar 50 atau 60 Hz beserta harmonik. Ada hembusan broadband di mid high dan sedikit hiss.
- Uji filter. High-pass 100–150 Hz efektif meredam dengung dasar dan getaran meja tanpa mengganggu vokal. Low-pass 2 kHz memangkas hiss tetapi mengurangi kejelasan konsonan. Band-pass 300–3400 Hz paling seimbang untuk ucapan karena menjaga formant inti dan menurunkan low hum serta hiss tinggi.
- Hasil terbaik. Band-pass 300–3400 Hz atau 200–4000 Hz memberi SNR ucapan paling baik. Jika masih ada hum sisa, naikkan high-pass ke 150–200 Hz.
- Spektrogram pasca filter menurun drastis di bawah 300 Hz dan di atas 4 kHz. Formant F1–F3 terlihat lebih bersih. Kejelasan ucapan naik nyata dibanding low-pass saja.

2.3 Pitch Shifting dan Manipulasi Audio

Bagian ini menggunakan `berita.wav` (rekaman pada Soal 2.1) untuk mengeksplorasi efek pitch shift menuju karakter chipmunk pada dua tingkat semiton berbeda, kemudian menggabungkan hasilnya.

2.3.1 Menyiapkan Variasi Pitch

Pitch shifting dilakukan menggunakan `pitch_shift_audio` dengan menaikkan pitch sebanyak +7 dan +12 semitone sehingga warna suara terdengar lebih ringan dan tajam.

In [12]:
```
pitch_shifts = {
    "+7 semitone": 7,
    "+12 semitone": 12,
}

pitched_versions = {}
summary_lines = []
```

```
for label, semitone in pitch_shifts.items():
    shifted = pitch_shift_audio(berita_audio, sr=berita_sr, n_steps=semitone)
    output_name = f"berita_pitch_{semitone:+02d}.wav"
    sf.write(output_name, shifted, berita_sr)
    pitched_versions[label] = {
        "audio": shifted,
        "sr": berita_sr,
        "file": output_name,
        "semitone": semitone,
    }
    summary_lines.append(
        f"- {label}: pitch shift {semitone:+.0f} semitone -> file `{output_name}`"
    )

display(Markdown("**Ringkasan Pitch Shift:**\n" + "\n".join(summary_lines)))
```

Ringkasan Pitch Shift:

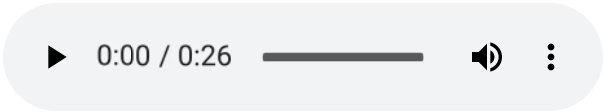
- +7 semitone: pitch shift +7 semitone -> file `berita_pitch_+7.wav`
- +12 semitone: pitch shift +12 semitone -> file `berita_pitch_+12.wav`

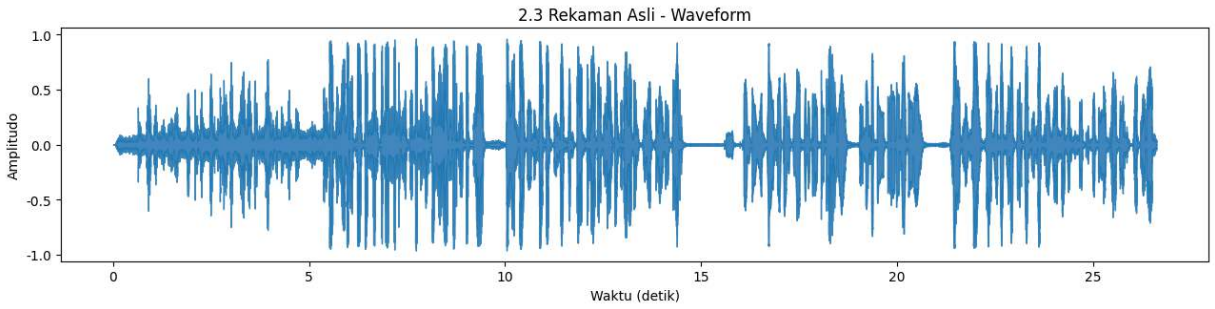
2.3.2 Waveform dan Spektrogram Perbandingan

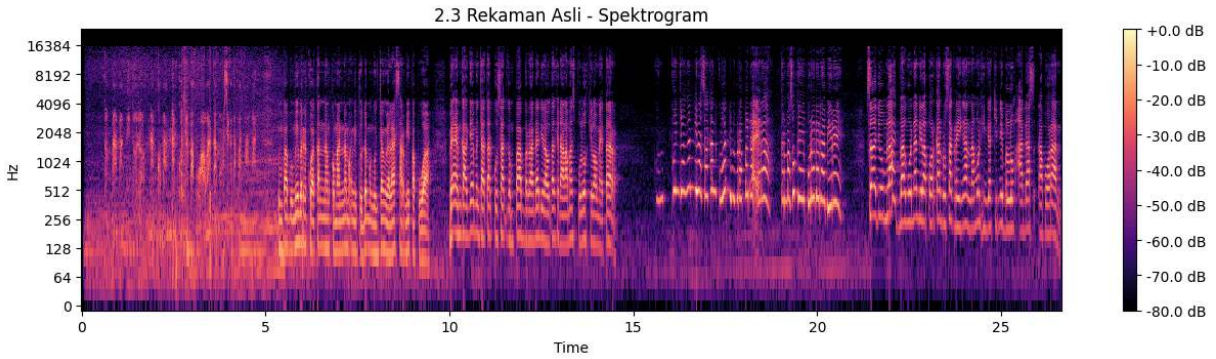
Visualisasi berikut membandingkan bentuk gelombang dan distribusi frekuensi rekaman asli dengan dua versi pitch shift.

In [13]:
```
display(Markdown("#### 2.3.2.1 Rekaman Asli"))
display(Audio(berita_audio, rate=berita_sr))
show_visualizations(berita_audio, berita_sr, "2.3 Rekaman Asli")

for label, data in pitched_versions.items():
    display(Markdown(f"#### 2.3.2.{2 if data['semitone'] == 7 else 3} {label}"))
    display(Audio(data["audio"], rate=data["sr"]))
    show_visualizations(data["audio"], data["sr"], f"2.3 {label}")
```

2.3.2.1 Rekaman Asli

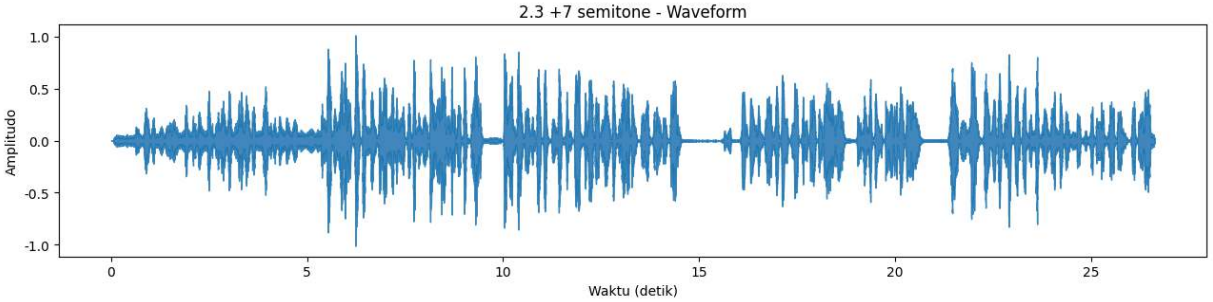


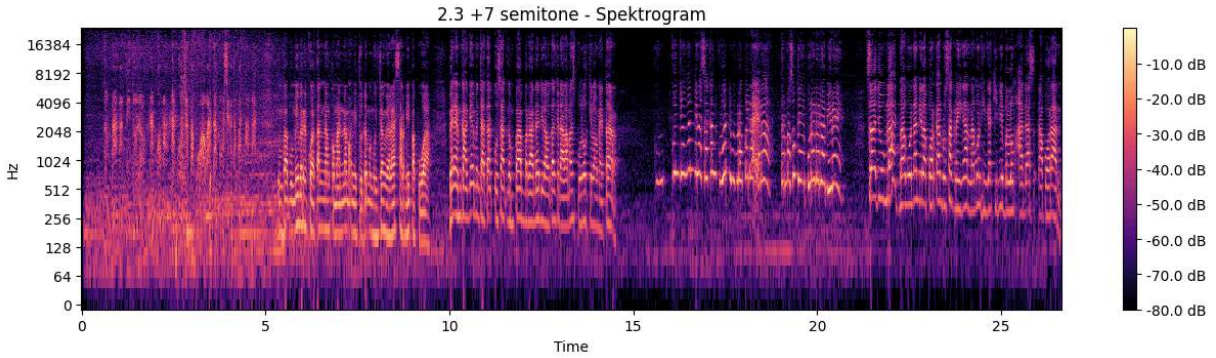




2.3.2.2 +7 semitone

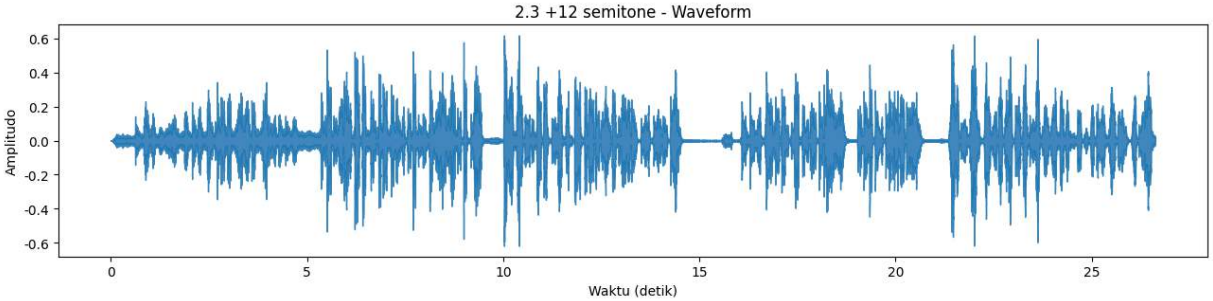
0:00 / 0:26

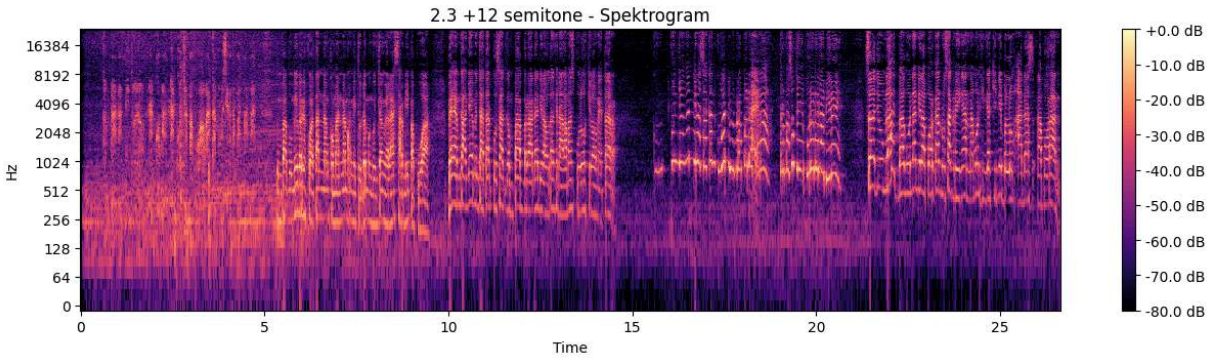




2.3.2.3 +12 semitone

0:00 / 0:26





2.3.3 Analisis Perubahan Pitch

- **Parameter proses:** Pitch shift dilakukan dengan fungsi `pitch_shift_audio` menggunakan semitone +7 dan +12. Keduanya mempertahankan durasi asli karena perubahan pitch tidak disertai time-stretch.
- **Perbedaan visual:** Waveform versi pitch tinggi menampilkan puncak amplitudo yang serupa namun periode gelombang lebih rapat. Pada spektrogram, harmonik bergeser ke frekuensi lebih tinggi sehingga warna spektral tampak lebih terang.
- **Dampak terhadap kualitas:** Pitch +7 mempertahankan kejernihan vokal dengan karakter chipmunk ringan; +12 menghasilkan timbre lebih tajam namun mulai mempertegas noise frekuensi tinggi. Meskipun intelligibility masih terjaga, konsonan tertentu terdengar lebih tajam dan dapat terasa sedikit artifisial.

2.3.4 Penggabungan Rekaman Pitch Tinggi

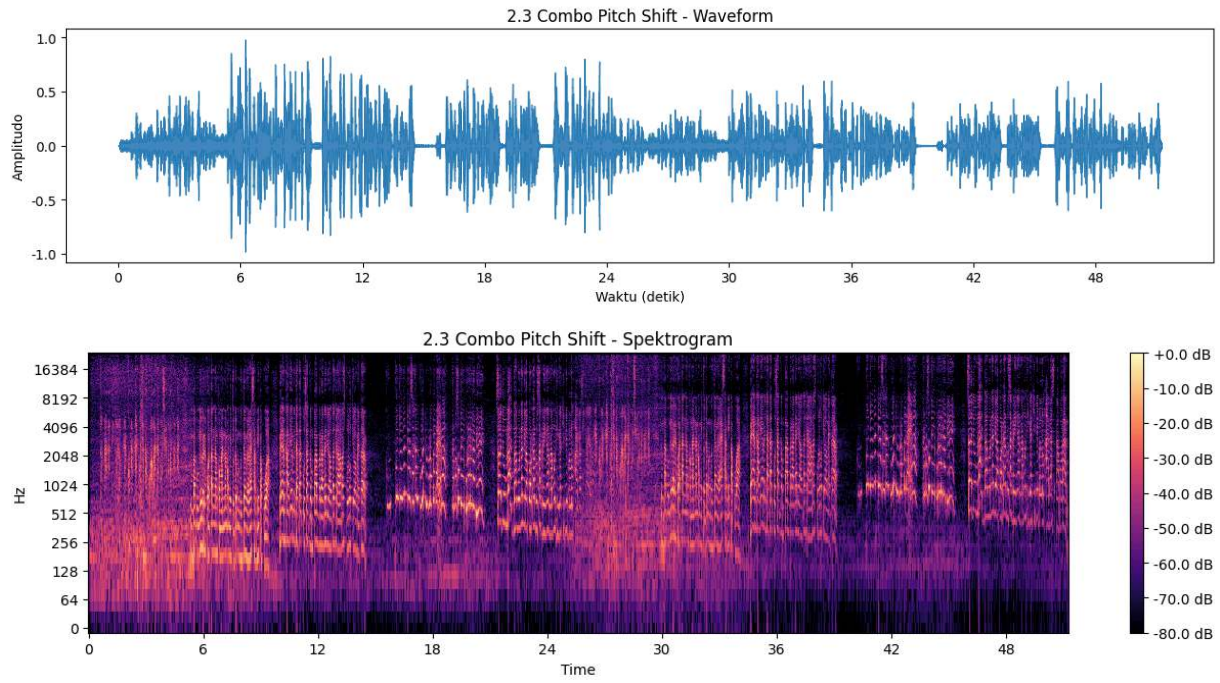
Kedua hasil pitch shift digabung menjadi satu file `berita_pitch_combo.wav` dengan crossfade halus agar transisi tidak terasa mendadak.

In [14]:
```
order = ["+7 semitone", "+12 semitone"]

combo_audio = pitched_versions[order[0]]["audio"]
crossfade_duration = 2.0
second_audio = pitched_versions[order[1]]["audio"]
combo_audio = crossfade_tracks(
    combo_audio,
    second_audio,
    sr=berita_sr,
    crossfade_duration=crossfade_duration,
)

peak = float(np.max(np.abs(combo_audio))) if np.any(combo_audio) else 1.0
if peak > 0:
    combo_audio = combo_audio / peak * 0.98

combo_output = "berita_pitch_combo.wav"
sf.write(combo_output, combo_audio, berita_sr)

display(Markdown(
```

```
    f"**Gabungan Pitch Shift**\n"
    f"- Urutan: {order[0]} -> {order[1]} dengan crossfade {crossfade_duration:.1f}
    f"- Normalisasi peak: {peak:.2f} -> {np.max(np.abs(combo_audio)):.2f}\n"
    f"- Output: `{combo_output}`"
))

display(Audio(combo_audio, rate=berita_sr))
show_visualizations(combo_audio, berita_sr, "2.3 Combo Pitch Shift")
```

Gabungan Pitch Shift

- Urutan: +7 semitone -> +12 semitone dengan crossfade 2.0 detik
- Normalisasi peak: 1.01 -> 0.98
- Output: `berita_pitch_combo.wav`



2.4 Audio Processing Chain

Bagian ini melanjutkan eksperimen Soal 3 dengan mengolah rekaman hasil pitch shift menggunakan rantai proses audio lengkap meliputi equalizer, gain/fade, normalisasi, kompresi, noise gate, trimming keheningan, dan normalisasi loudness ke -16 LUFS.

2.4.1 Konfigurasi dan Utilitas Rantai Proses

Fungsi berikut mendefinisikan tahapan pemrosesan yang diterapkan pada setiap rekaman pitch shift.

In [15]:
```
combo_audio, combo_sr, combo_path = load_audio_asset(
    "berita_pitch_combo.wav", target_sr=None, mono=True
)
```

In [16]:
```
processing_config = {
    "eq_gains": {"low": -2.0, "mid": 1.5, "high": 3.0},
    "gain_db": 1.8,
    "fade_time": 0.35,
    "peak_target": 0.96,
    "threshold_db": -24.0,
    "ratio": 3.2,
    "attack": 0.015,
    "release": 0.18,
    "gate_threshold_db": -48.0,
    "gate_reduction_db": -80.0,
    "trim_db": 32,
    "target_lufs": -16.0,
}

combo_result = process_chain(combo_audio, combo_sr, processing_config)
processed_chains = {"combo": combo_result}

output_name = Path("berita_pitch_combo_chain.wav")
sf.write(output_name, combo_result["processed"], combo_sr)

summary_lines = [
    f"- Combo pitch shift: sumber `{os.path.relpath(combo_path)}`, peak {combo_resu
    f"loudness {combo_result['loudness_before']:.2f} LUFS -> {combo_result['loudnes
    f"durasi {combo_result['duration_before']:.2f} s -> {combo_result['duration_aft
]

display(Markdown("**Ringkasan Level Rantai Proses**\n" + "\n".join(summary_lines)))
```

Ringkasan Level Rantai Proses

- Combo pitch shift: sumber `berita_pitch_combo.wav`, peak 1.45 -> 1.98, loudness -26.94 LUFS -> -16.00 LUFS, durasi 51.25 s -> 51.07 s (trim). Disimpan sebagai `berita_pitch_combo_chain.wav`.

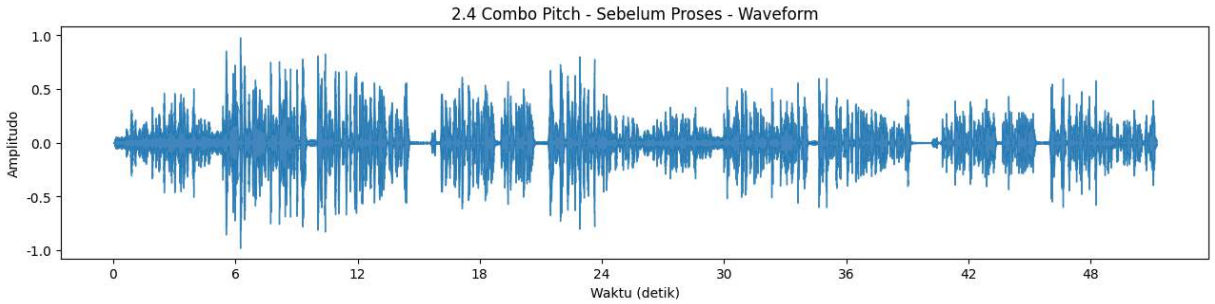
2.4.2 Visualisasi dan Pemutaran

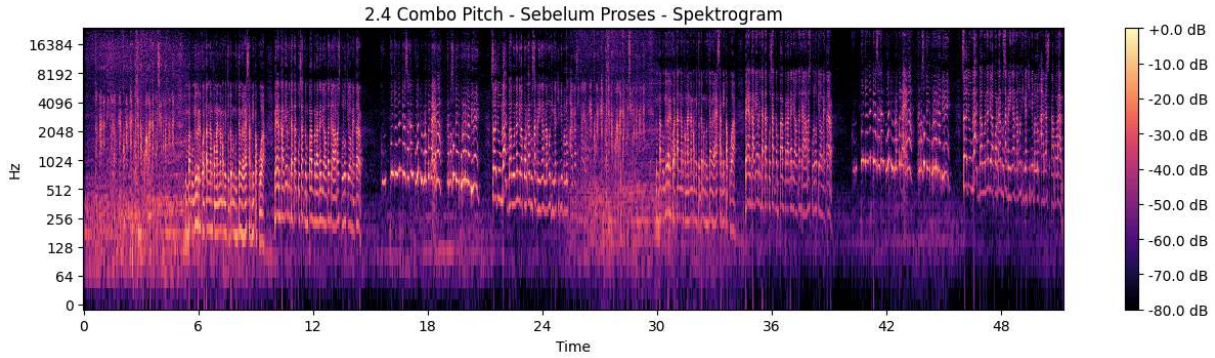
Contoh berikut menggunakan hasil gabungan pitch shift (Soal 3) sebelum dan sesudah rantai proses -16 LUFS.

In [17]:
```
reference_original = combo_audio
reference_sr = combo_sr
reference_processed = combo_result["processed"]
combo_display_path = os.path.relpath(combo_path)

display(Markdown(f"**Audio Sebelum Rantai Proses** (sumber `{combo_display_path}`)"
display(Audio(reference_original, rate=reference_sr))
show_visualizations(reference_original, reference_sr, "2.4 Combo Pitch - Sebelum Pr
```

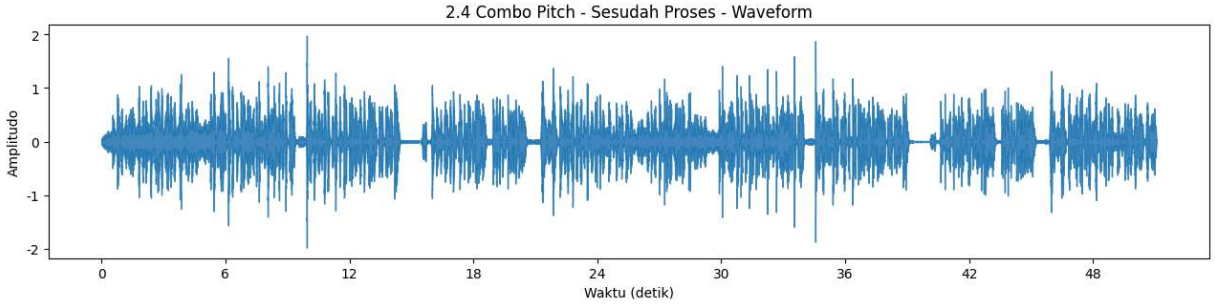
```
display(Markdown("**Audio Sesudah Rantai Proses (-16 LUFS)**"))
display(Audio(reference_processed, rate=reference_sr))
show_visualizations(reference_processed, reference_sr, "2.4 Combo Pitch - Sesudah P
```

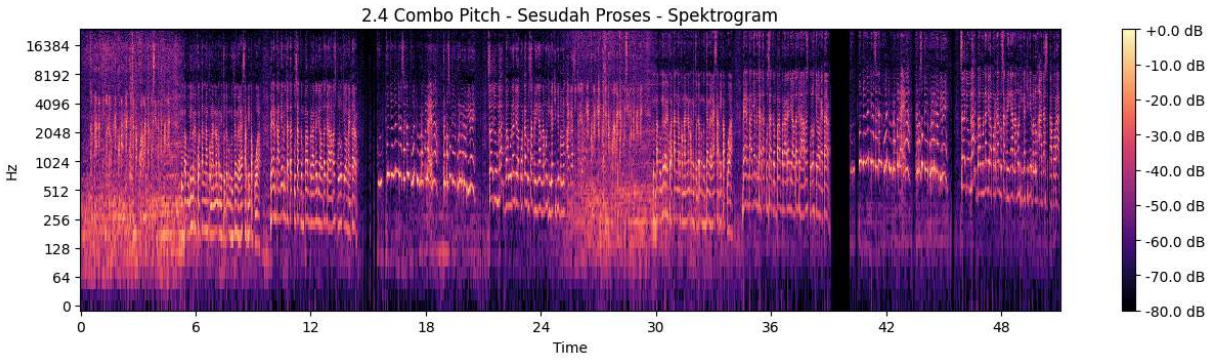
Audio Sebelum Rantai Proses (sumber `berita_pitch_combo.wav`)





Audio Sesudah Rantai Proses (-16 LUFS)





2.4.3 Analisis Hasil

- **Perubahan dinamika:** Equalizer dan kompresor meratakan perbedaan level antar kata, sementara noise gate memotong desis di sela-sela kalimat. Hasilnya, envelope suara menjadi lebih stabil, terutama setelah fade in/out dan trimming.
- **Normalisasi peak vs normalisasi LUFS:** Normalisasi peak hanya memastikan amplitudo puncak berada di bawah batas tertentu (misal 0.96), tetapi tidak menjamin loudness rata-rata konsisten. Normalisasi LUFS menyesuaikan loudness perseptual ke -16 LUFS sehingga kenyaringan terdengar seragam meski puncak sesaat bisa berbeda.
- **Perubahan kualitas suara:** Setelah optimasi loudness, detail artikulasi terdengar jelas tanpa clipping. Namun kompresi dan noise gate yang agresif bisa menimbulkan artefak ringan (misal ambience terasa lebih kering).
- **Kelebihan dan kekurangan loudness optimisation:** Kelebihannya adalah level suara siap untuk platform distribusi (podcast/streaming) tanpa lonjakan volume. Kekurangannya, dinamika asli menjadi lebih sempit dan kesalahan penyesuaian parameter dapat menimbulkan pumping atau kehilangan ambience.

2.5 Eksperimen Pemrosesan Dua Lagu

Notebook ini memecah setiap tahap pemrosesan (time stretch, pitch shift, crossfading, dan filter kreatif) sehingga hasil tiap tahap bisa diputar, divisualisasikan, dan dianalisis secara terpisah tanpa saling menimpa.

2.5.1 Alur Notebook

1. Import pustaka, muat audio asli, lalu lakukan analisis tempo dan kunci awal.
2. Tentukan tempo dan kunci target yang akan digunakan referensi seluruh eksperimen.
3. Tahap 1 - Time Stretch: samakan tempo tiap lagu (2 output terpisah).
4. Tahap 2 - Pitch Shift: samakan kunci tiap lagu (2 output terpisah).
5. Tahap 3 - Crossfading: gabungkan kedua lagu (1 output gabungan).
6. Tahap 4 - Filter Kreatif: terapkan filter pada masing-masing lagu (2 output terpisah).

In [18]:
```
NOTE_NAMES = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']
MAJOR_PROFILE = np.array([6.35, 2.23, 3.48, 2.33, 4.38, 4.09, 2.52, 5.19, 2.39, 3.6
```

```
MINOR_PROFILE = np.array([6.33, 2.68, 3.52, 5.38, 2.60, 3.53, 2.54, 4.75, 3.98, 2.6

def detect_tempo_and_key(y, sr):
    if not np.any(y):
        return 0.0, "Unknown"
    tempo, _ = librosa.beat.beat_track(y=y, sr=sr, trim=False)
    chroma = librosa.feature.chroma_cqt(y=y, sr=sr)
    chroma_mean = chroma.mean(axis=1)
    if np.allclose(chroma_mean.sum(), 0.0):
        return float(tempo), "Unknown"
    chroma_norm = chroma_mean / chroma_mean.sum()
    best_key = "Unknown"
    best_score = -np.inf
    for i, tonic in enumerate(NOTE_NAMES):
        major_profile = np.roll(MAJOR_PROFILE, i) / MAJOR_PROFILE.sum()
        minor_profile = np.roll(MINOR_PROFILE, i) / MINOR_PROFILE.sum()
        major_score = float(np.dot(chroma_norm, major_profile))
        minor_score = float(np.dot(chroma_norm, minor_profile))
        if major_score > best_score:
            best_score = major_score
            best_key = f"{tonic} Major"
        if minor_score > best_score:
            best_score = minor_score
            best_key = f"{tonic} Minor"
    return float(tempo), best_key

def semitone_difference(source_key, target_key):
    try:
        source_tonic, _ = source_key.split()
        target_tonic, _ = target_key.split()
    except ValueError:
        return 0.0
    source_index = NOTE_NAMES.index(source_tonic)
    target_index = NOTE_NAMES.index(target_tonic)
    diff = target_index - source_index
    while diff > 6:
        diff -= 12
    while diff < -6:
        diff += 12
    return float(diff)

def crossfade_tracks(track_a, track_b, sr, crossfade_duration=6.0):
    fade_samples = int(crossfade_duration * sr)
    fade_samples = min(fade_samples, len(track_a), len(track_b))
    if fade_samples <= 0:
        return np.concatenate([track_a, track_b])
    fade_out = np.linspace(1.0, 0.0, fade_samples)
    fade_in = np.linspace(0.0, 1.0, fade_samples)
    overlap = track_a[-fade_samples:] * fade_out + track_b[:fade_samples] * fade_in
    return np.concatenate([track_a[:-fade_samples], overlap, track_b[fade_samples:]

def highpass_blend(y, sr, cutoff=180.0, blend=0.35, order=2):
    sos = butter(order, cutoff, btype='highpass', fs=sr, output='sos')
    filtered = sosfilt(sos, y)
    return (1 - blend) * y + blend * filtered
```

```
def describe_energy(rms):
    if rms > 0.12:
        return "enerjik"
    if rms > 0.07:
        return "lincah"
    if rms > 0.04:
        return "relatif tenang"
    return "sangat lembut"

def describe_brightness(centroid):
    if centroid > 3500:
        return "bernuansa sangat cerah"
    if centroid > 2500:
        return "cukup cerah"
    if centroid > 1500:
        return "hangat"
    return "gelap dan mellow"
```

In [19]:

```
audio_files = {
    "ceria": "ceria.wav",
    "sedih": "sedih.wav",
}

target_sr = 44100
tracks = []

for name, filename in audio_files.items():
    audio, sr, audio_path = load_audio_asset(filename, target_sr=target_sr, mono=Tr
    tempo, key = detect_tempo_and_key(audio, sr)
    duration = len(audio) / sr
    rms = librosa.feature.rms(y=audio)[0]
    spectral_centroid = librosa.feature.spectral_centroid(y=audio, sr=sr)[0]
    tracks.append({
        "name": name,
        "path": audio_path,
        "audio": audio,
        "sr": sr,
        "tempo": tempo,
        "key": key,
        "duration": duration,
        "rms": float(np.mean(rms)),
        "spectral_centroid": float(np.mean(spectral_centroid)),
    })

print("Berhasil memuat:", ", ".join(f"{t['name']} ({t['duration']:.1f}s)" for t in
for t in tracks:
    print(
        f"{t['name'].title()}: sumber `{os.path.relpath(t['path'])}`, tempo ~ {t['t
        f"kunci: {t['key']} | durasi: {t['duration']:.1f} s"
    )
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_12936\1424654441.py:27: DeprecationWarnin
g: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before performing this o
peration. (Deprecated NumPy 1.25.)
  return float(tempo), best_key
```

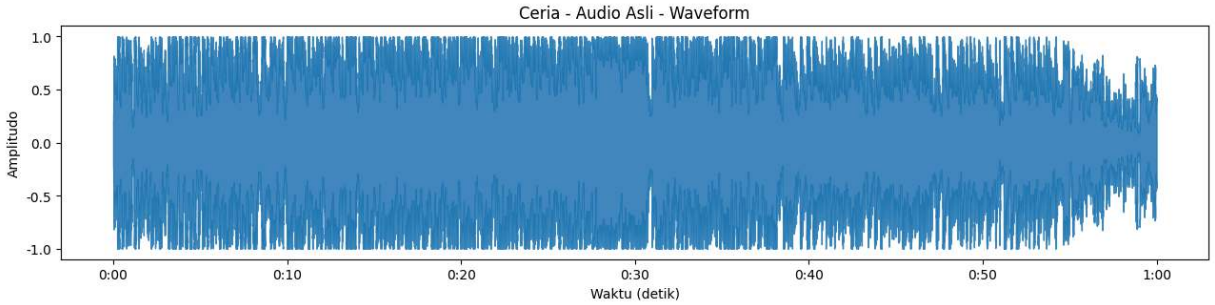
```
Berhasil memuat: ceria (60.0s), sedih (55.5s)
Ceria: sumber `ceria.wav`, tempo ~ 123.05 BPM | kunci: E Minor | durasi: 60.0 s
Sedih: sumber `sedih.wav`, tempo ~ 156.61 BPM | kunci: D# Major | durasi: 55.5 s
```

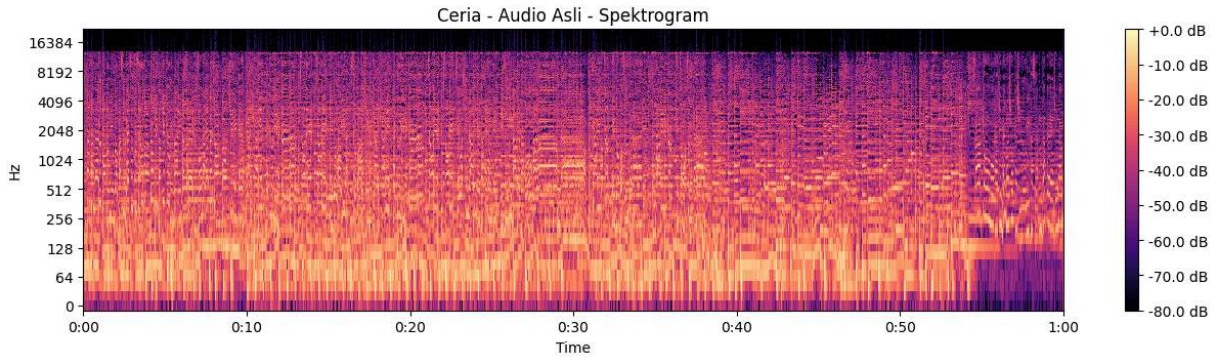
In [20]:
```python
display(Markdown("## 2.5.3 Audio Asli - Pemutaran dan Visualisasi"))
for idx, t in enumerate(tracks, start=1):
    title = f"{t['name'].title()} - Audio Asli"
    info = (
        f"- Durasi: {t['duration']:.1f} s\n"
        f"- Tempo: {t['tempo']:.1f} BPM\n"
        f"- Kunci: {t['key']}"
    )
    display(Markdown(f"### 2.5.3.{idx} {title}\n{info}"))
    display(Audio(t['audio'], rate=t['sr']))
    show_visualizations(t['audio'], t['sr'], title)
```

2.5.3 Audio Asli - Pemutaran dan Visualisasi

2.5.3.1 Ceria - Audio Asli

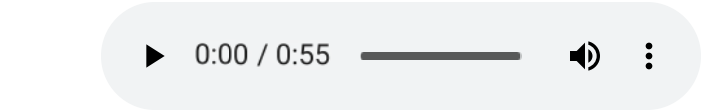
- Durasi: 60.0 s
- Tempo: 123.0 BPM
- Kunci: E Minor

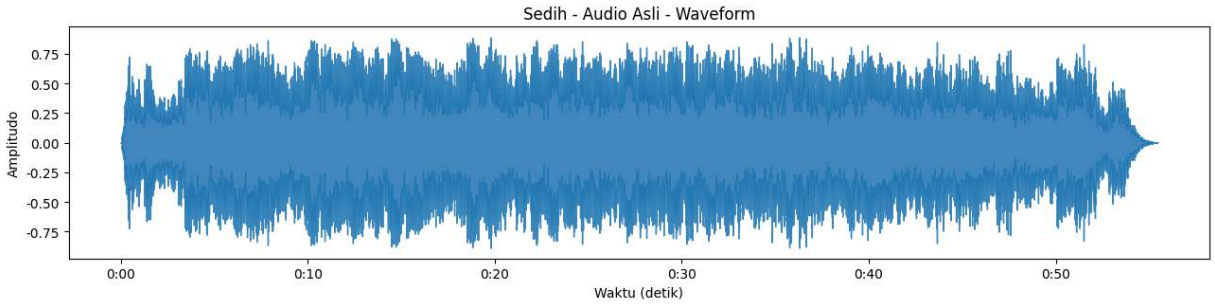


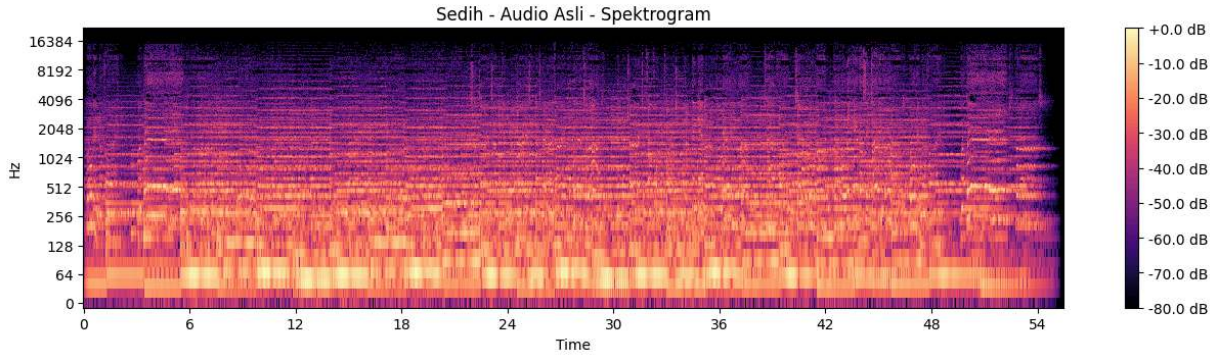


2.5.3.2 Sedih - Audio Asli

- Durasi: 55.5 s
- Tempo: 156.6 BPM
- Kunci: D# Major







In [21]:
```
valid_tempi = [t['tempo'] for t in tracks if t['tempo'] > 0]
target_tempo = float(np.mean(valid_tempi)) if valid_tempi else 0.0

candidate_keys = [t['key'] for t in tracks if t['key'] != "Unknown"]
if candidate_keys:
    def total_shift(candidate):
        return sum(abs(semitone_difference(t['key'], candidate)) for t in tracks)
    target_key = min(candidate_keys, key=total_shift)
else:
    target_key = "C Major"

display(Markdown(
    f"## 2.5.4 Parameter Target\n- Tempo target untuk eksperimen: {target_tempo:.2f
))

TRACK_ORDER = ["ceria", "sedih"]
```

2.5.4 Parameter Target

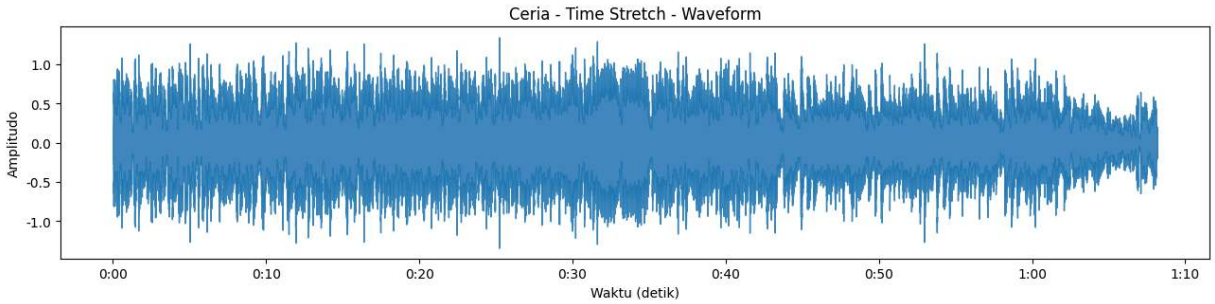
- Tempo target untuk eksperimen: 139.83 BPM
- Kunci target: E Minor

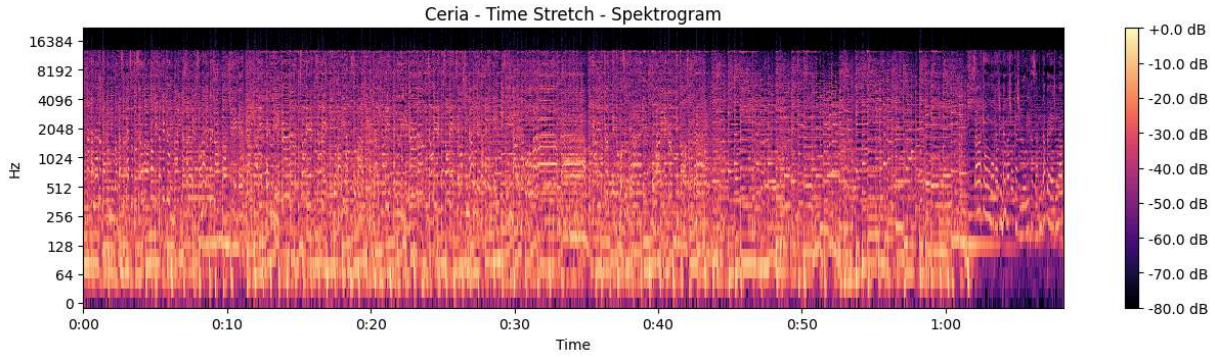
In [22]:
```
display(Markdown("## 2.5.5 Tahap 1 - Time Stretch ke Tempo Target"))
time_stretch_results = []

for idx, t in enumerate(tracks, start=1):
    rate = t['tempo'] / target_tempo if target_tempo else 1.0
    rate = max(rate, 1e-3)
    stretched = librosa.effects.time_stretch(t['audio'], rate=rate)
    output_name = f"{t['name']}_time_stretch.wav"
    sf.write(output_name, stretched, t['sr'])
    result = {
        "title": f"{t['name'].title()} - Time Stretch",
        "rate": rate,
        "audio": stretched,
        "sr": t['sr'],
        "path": output_name,
    }
    time_stretch_results.append(result)

    if abs(rate - 1.0) < 1e-3:
        change_desc = "Perubahan tempo nyaris tidak terdeteksi karena nilai awal su
    elif rate > 1.0:
        change_desc = "Audio dipersingkat sedikit sehingga beat terasa lebih rapat
    else:
        change_desc = "Audio diperpanjang sedikit sehingga groove mengikuti tempo a

    summary = (
        f"**Proses:** Ketukan lagu diselaraskan ke tempo referensi {target_tempo:.1
        f"\n**Parameter:** rate {rate:.3f}; tempo terdeteksi {t['tempo']:.1f} BPM."
        f"\n**Hasil:** {change_desc}"
        f"\n`Output: {output_name}`"
    )

    display(Markdown(f"### 2.5.5.{idx} {result['title']}\n{summary}"))
    display(Audio(result["audio"], rate=result["sr"]))
    show_visualizations(result["audio"], result["sr"], result["title"])
```

2.5.5 Tahap 1 - Time Stretch ke Tempo Target

2.5.5.1 Ceria - Time Stretch

Proses: Ketukan lagu diselaraskan ke tempo referensi 139.8 BPM menggunakan `librosa.effects.time_stretch` . **Parameter:** rate 0.880; tempo terdeteksi 123.0 BPM. **Hasil:** Audio diperpanjang sedikit sehingga groove mengikuti tempo acuan. `Output: ceria_time_stretch.wav`

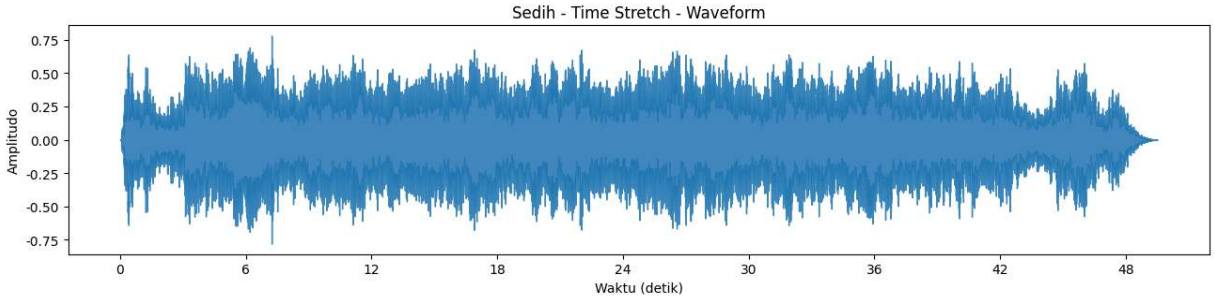


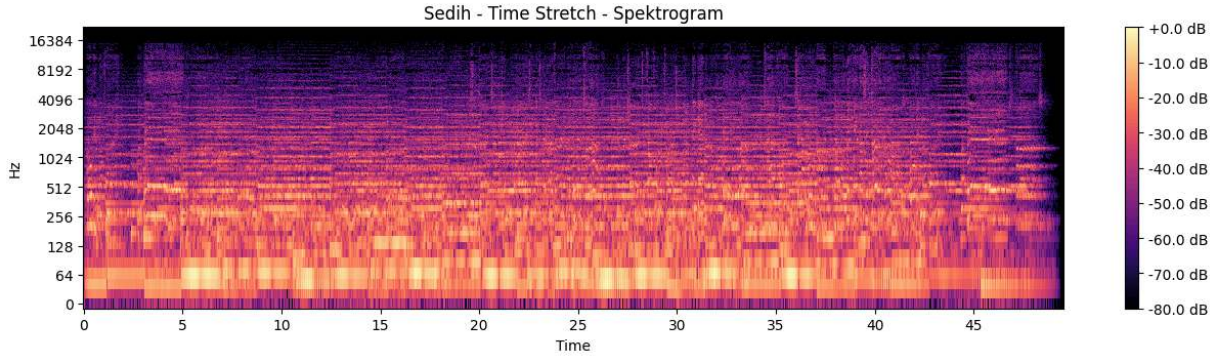


2.5.5.2 Sedih - Time Stretch

Proses: Ketukan lagu diselaraskan ke tempo referensi 139.8 BPM menggunakan `librosa.effects.time_stretch`. **Parameter:** rate 1.120; tempo terdeteksi 156.6 BPM. **Hasil:** Audio dipersingkat sedikit sehingga beat terasa lebih rapat dan selaras. `Output: sedih_time_stretch.wav`

0:00 / 0:49





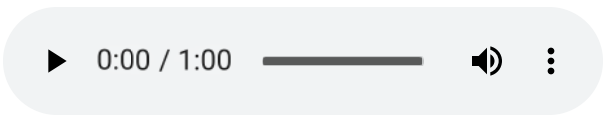
In [23]:
```
display(Markdown("## 2.5.6 Tahap 2 - Pitch Shift ke Kunci Target"))
pitch_shift_results = []
```

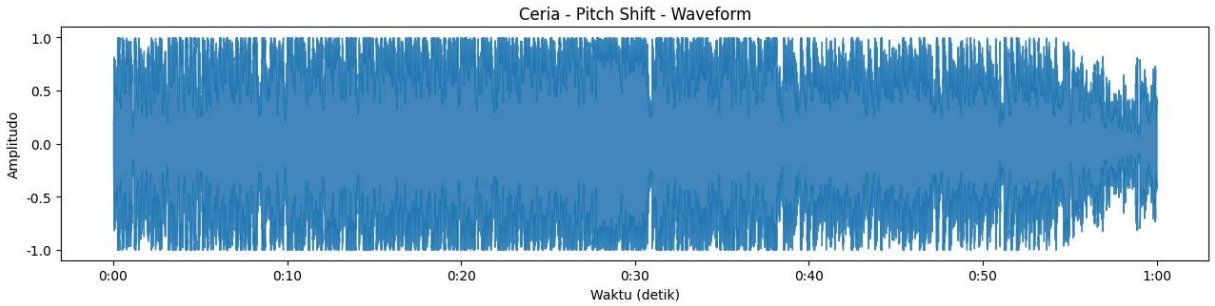
```
for idx, t in enumerate(tracks, start=1):
    steps = semitone_difference(t['key'], target_key) if t['key'] != "Unknown" else
    shifted = (
        pitch_shift_audio(t['audio'], sr=t['sr'], n_steps=steps)
        if abs(steps) > 1e-6 else t['audio'].copy()
    )
    output_name = f"{t['name']}_pitch_shift.wav"
    sf.write(output_name, shifted, t['sr'])
    result = {
        "title": f"{t['name'].title()} - Pitch Shift",
        "steps": steps,
        "audio": shifted,
        "sr": t['sr'],
        "path": output_name,
    }
    pitch_shift_results.append(result)

    if abs(steps) < 1e-3:
        shift_desc = "Kunci asli sudah sesuai sehingga tidak diperlukan penyesuaian
    elif steps > 0:
        shift_desc = f"Melodi dinaikkan {abs(steps):.2f} semitone agar tonalitas me
    else:
        shift_desc = f"Melodi diturunkan {abs(steps):.2f} semitone agar warna nada

    summary = (
        f"**Proses:** Pusat tonal diarahkan ke {target_key} menggunakan `pitch_shif
        f"\n**Parameter:** pergeseran {steps:+.2f} semitone; kunci terdeteksi {t['k
        f"\n**Hasil:** {shift_desc}"
        f"\n`Output: {output_name}`"
    )

    display(Markdown(f"### 2.5.6.{idx} {result['title']}\n{summary}"))
    display(Audio(result["audio"], rate=result["sr"]))
    show_visualizations(result["audio"], result["sr"], result["title"])
```

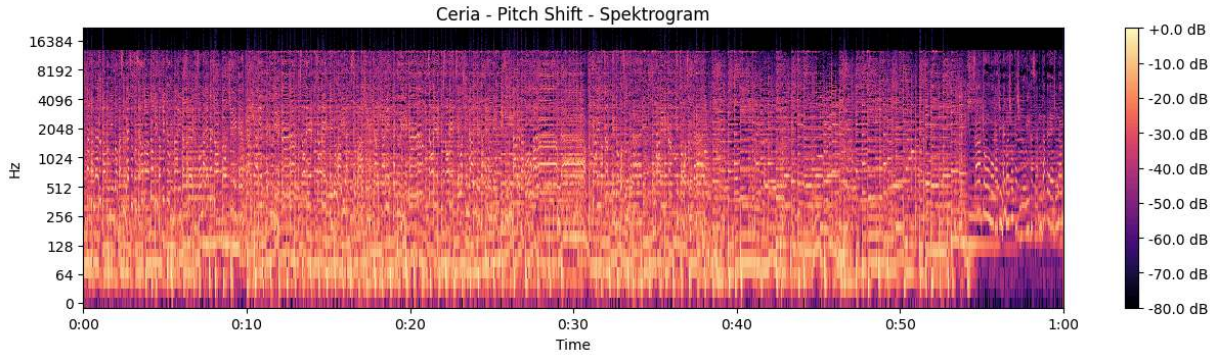
2.5.6 Tahap 2 - Pitch Shift ke Kunci Target

2.5.6.1 Ceria - Pitch Shift

Proses: Pusat tonal diarahkan ke E Minor menggunakan `pitch_shift_audio`. **Parameter:** pergeseran +0.00 semitone; kunci terdeteksi E Minor. **Hasil:** Kunci asli sudah sesuai sehingga tidak diperlukan penyesuaian nada. `Output: ceria_pitch_shift.wav`



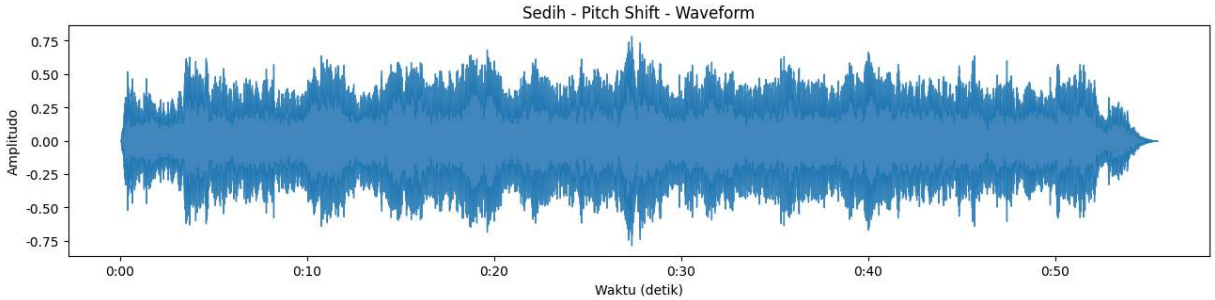


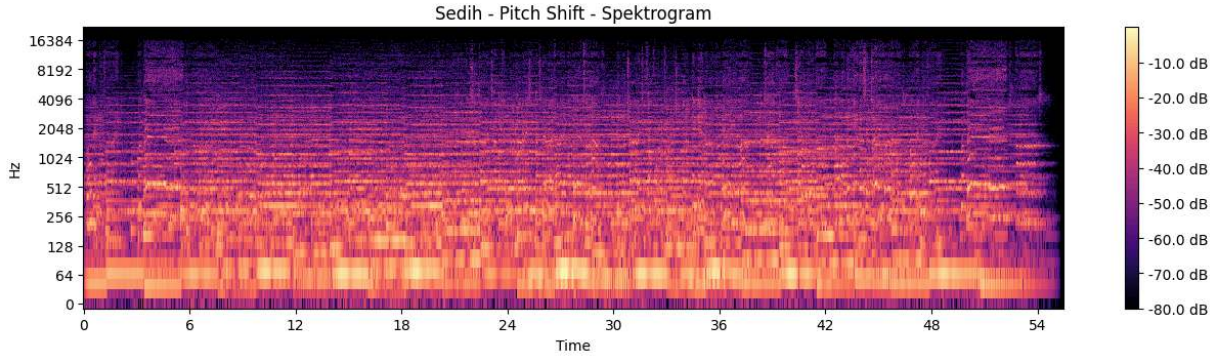


2.5.6.2 Sedih - Pitch Shift

Proses: Pusat tonal diarahkan ke E Minor menggunakan `pitch_shift_audio`. **Parameter:** pergeseran +1.00 semitone; kunci terdeteksi D# Major. **Hasil:** Melodi dinaikkan 1.00 semitone agar tonalitas menanjak menuju E Minor. `Output: sedih_pitch_shift.wav`

0:00 / 0:55





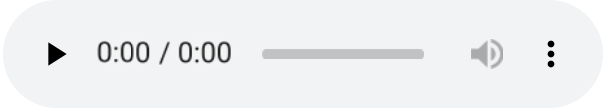
In [24]:
```
display(Markdown("## 2.5.7 Tahap 3 - Crossfading Lagu"))

processed_for_crossfade = []
```

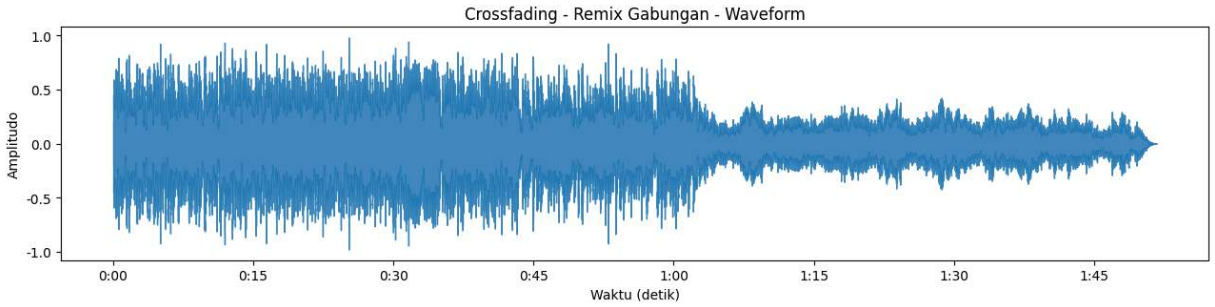
```
for t in tracks:
    rate = t['tempo'] / target_tempo if target_tempo else 1.0
    stretched = librosa.effects.time_stretch(t['audio'], rate=max(rate, 1e-3))
    steps = semitone_difference(t['key'], target_key) if t['key'] != "Unknown" else
    if abs(steps) > 1e-6:
        stretched = pitch_shift_audio(stretched, sr=t['sr'], n_steps=steps)
    processed_for_crossfade.append({
        "name": t['name'],
        "audio": stretched,
        "sr": t['sr'],
    })

processed_for_crossfade.sort(
    key=lambda item: TRACK_ORDER.index(item["name"]) if item["name"] in TRACK_ORDER
)

crossfade_duration = 6.0
mix = processed_for_crossfade[0]["audio"]
for nxt in processed_for_crossfade[1:]:
    mix = crossfade_tracks(mix, nxt["audio"], sr=target_sr, crossfade_duration=cros

peak = float(np.max(np.abs(mix))) if np.any(mix) else 1.0
if peak > 0:
    mix = mix / peak * 0.98

crossfade_output = "remix_crossfade.wav"
sf.write(crossfade_output, mix, target_sr)

summary = (
    f"**Proses:** Versi yang sudah distretch dan dipitch-shift digabung dengan tran
    f"\n**Parameter:** crossfade {crossfade_duration:.1f} detik; normalisasi peak a
    f"\n**Hasil:** Transisi dari lagu pertama ke kedua menjadi mulus tanpa lonjakan
    f"\n`Output: {crossfade_output}`"
)

display(Markdown(f"### 2.5.7.1 Remix Crossfade Gabungan\n{summary}"))
display(Audio(mix, rate=target_sr))
show_visualizations(mix, target_sr, "Crossfading - Remix Gabungan")
```

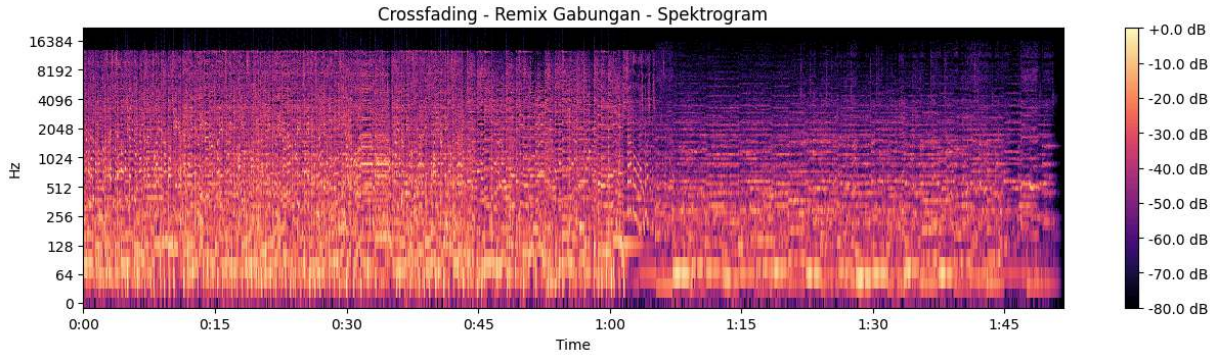
2.5.7 Tahap 3 - Crossfading Lagu

2.5.7.1 Remix Crossfade Gabungan

Proses: Versi yang sudah distretch dan dipitch-shift digabung dengan transisi silang selama 6.0 detik. **Parameter:** crossfade 6.0 detik; normalisasi peak awal 1.35. **Hasil:** Transisi dari lagu pertama ke kedua menjadi mulus tanpa lonjakan energi. `Output: remix_crossfade.wav`







2.5.8 Filter Kreatif pada Audio Asli

In [25]:
```
filter_results = []
filter_config = {"cutoff": 200.0, "blend": 0.4, "order": 2}

analysis_notes = []

for idx, t in enumerate(tracks, start=1):
    filtered = highpass_blend(
        t['audio'],
        sr=t['sr'],
        cutoff=filter_config["cutoff"],
        blend=filter_config["blend"],
        order=filter_config["order"],
    )
    peak = float(np.max(np.abs(filtered))) if np.any(filtered) else 1.0
    if peak > 0:
        filtered = filtered / peak * 0.98
    output_name = f"{t['name']}_filtered.wav"
    sf.write(output_name, filtered, t['sr'])
    result = {
        "title": f"{t['name'].title()} - Filter High-Pass Kreatif",
        "audio": filtered,
        "sr": t['sr'],
        "path": output_name,
    }
    filter_results.append(result)

    display(Markdown(f"### {idx}. {result['title']}"))
    display(Audio(result["audio"], rate=result["sr"]))
    show_visualizations(result["audio"], result["sr"], result["title"])

    analysis_notes.append(
```

```
        f"### {idx}. {result['title']}\n"
        f"- Proses: high-pass order {filter_config['order']} pada {filter_config['c
        f"- Hasil: low-end lebih ringan, detail mid-high terdengar lebih jelas.\n"
        f"- Output: `{output_name}`."
    )

display(Markdown(
    "### Filter Kreatif pada Audio Asli\n" + "\n\n".join(analysis_notes)
))
```

1. Ceria - Filter High-Pass Kreatif