**이미지 변환 함수 동작 방식 - Memory relocation**

첫번째 방식 – 포인터 주소를 이미지의 시작점으로 지정해준 후, 헤더를 떼어낸 부분을 RGBA 배열에 저장, 그 후 RGBA배열에서 Alpha 채널만 건너뛴 후 RGB배열에 RGB가 순환하며 저장

텍스트, 스크린샷, 폰트이(가) 표시된 사진

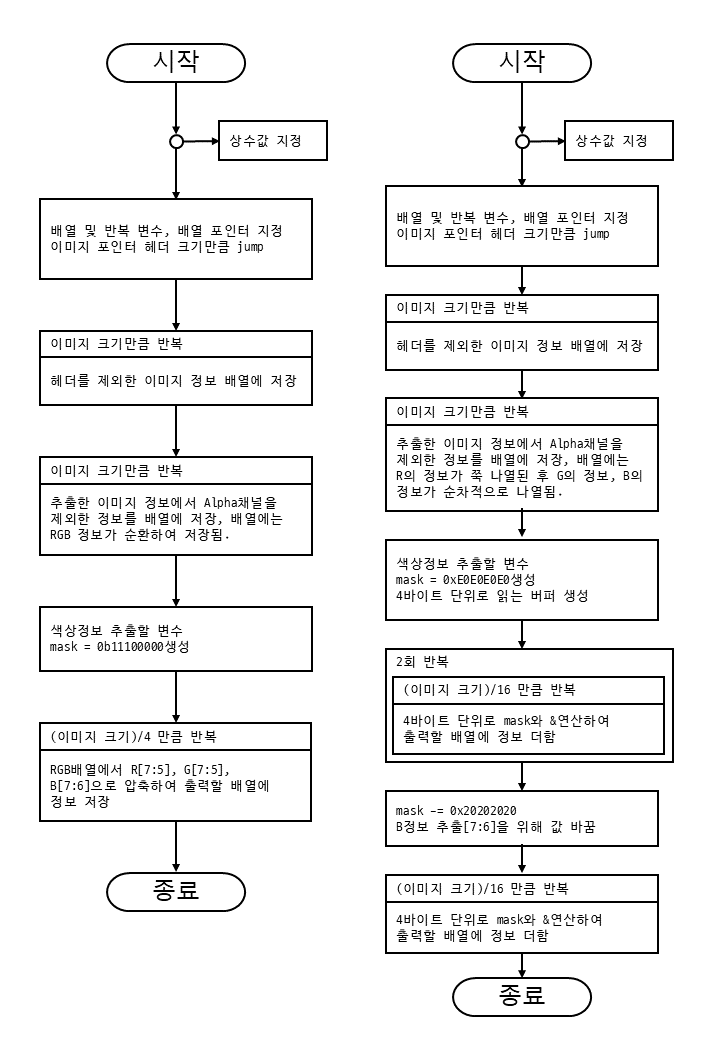
자동 생성된 설명

두번째 방식 – 첫번째 방식과 동일하나 이미지의 크기를 고려하여 한 색상의 크기(COLOR\_SIZE = IMAGE\_SIZE/4)만큼 건너뛰어 저장함. 이런 방식으로 저장하면 배열에는 RGB 배열에 R전체가 순서대로 나열된 후 G, B도 순서대로 나열됨. 크기는 첫번째 방식의 RGB배열과 동일하나 RGB가 순환하는지, 한가지 색상의 전체 정보가 나온 후 다음 색상 정보가 나오는 것에서 차이를 가짐. 기존에는 반복문을 하나 더 사용하여 RGBA배열을 세 번 순환하여 RGB배열에 색상 값 정보를 배치하도록 구성했다가, 3번이나 배열을 순환하면 처리하는데 많은 시간이 소요되어 이미지 크기를 기반으로 배열을 건너뛰어 처리시간을 줄임.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**순서도**



**C/ARM코드**

Github 참조

**실행 후 memory map 사진**

Case1

텍스트, 스크린샷, 직사각형, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 번호, 직사각형이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 직사각형, 번호이(가) 표시된 사진

자동 생성된 설명

Case2

텍스트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

\*나머지는 case1과 동일

**성능비교**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **load** | **comp** | **total** |
| **case1** | 34.817ms | 14.339ms | 49.156ms |
| **case2** | 33.281ms | 7.299ms | 40.580ms |

Case1 대비 case2는 이미지 압축 시 4바이트 단위로 움직일 수 있기 때문에 처리가 훨씬 빠르다. RGB가 반복하여 나열된 경우에는 이미지 정보(RGB 3바이트)가 레지스터 크기(8바이트)와 정렬하지 못해 어쩔 수 없이 1바이트씩 접근하여 압축된 RGB 1바이트로 저장할 수밖에 없음. 그러나 R~R,G~G,B~B로 저장된 경우 3바이트 RGB를 1바이트 RGB로 압축할 때 필요한 정보만 빼내서 더해가는 식으로 8바이트 단위로 접근하면 첫번째 방법보다 훨씬 빠르게 처리할 수 있다.