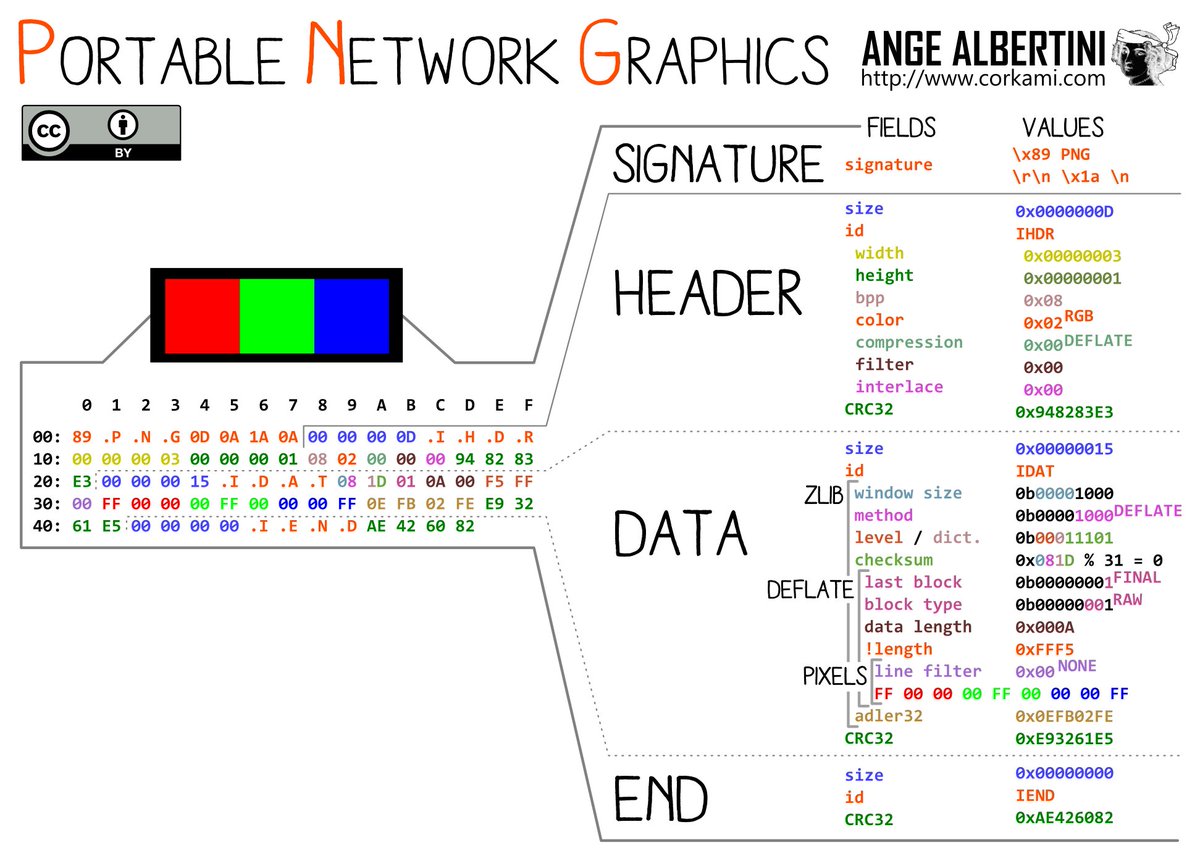
마이크로프로세서 function 3 진행상황 5/14 02:44

사전지식



PNG의 구조이며

맨 앞은 PNG임을 알리는 Signature가 있으며

Header인 IDHR은 PNG의 정보를 나타내는데,

파일의 width , height의 크기

RGB에서 각 파일의 value의 크기(8bit or 16bit)를 결정하는 bpp

Grayscale 이미지인지 RGBA이미지인지 등,color의 type을 결정하는 colortype

압축방식으로 고정된 방식인 Deflate ( Huffman & LZ77 )를 사용하는 compression

필터의 방식을 나타내는 0(고정) 1(빼기) …. filter type

Interlace 사용여부를 결정하는 0(사용x) 1(사용o) interlace

파일의 데이터의 오류가 있는지 확인해주는 crc

로 구성 되어있다.

다음 영역인 IDAT는 PNG의 pixel data를 나타내는데, 이 데이터는 filtering과 compressing이 되어있기 때문에 defiltering과 decompressing하여 Raw Data값을 구하여 계산할 수 있다.

마지막 영역인 IEND영역은 IDAT의 영역이 끝나고 파일의 마지막 부분임을 알 수 있게 해준다.

이외에도 PLTE chunk와 같이 여러 chunk가 존재하지만 RGBA.png에는 없으므로 고려하지 않는다.

//이구조를 이용하여 PNG파일을 해석하는 여러 함수들을 만들 수 있다.

텍스트, 도표, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명 텍스트, 폰트, 화이트, 영수증이(가) 표시된 사진

자동 생성된 설명

Function3 32bit RGBA image -> binary (black/white) image의 순서도

Raw Data : PNG를 Decoding하여 만들어 낸다.

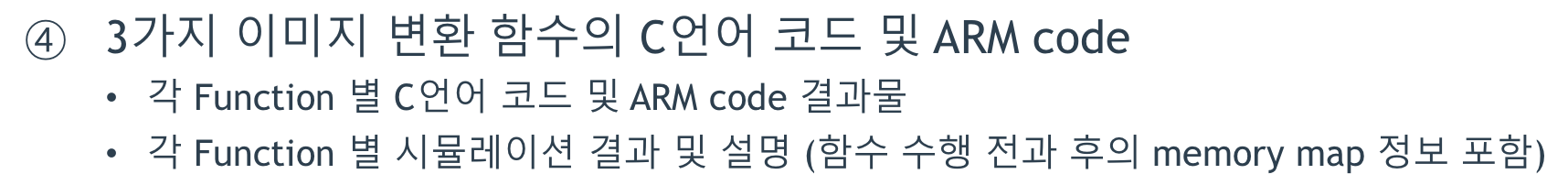
Skip Signature함수를 통해 PNG 시그니쳐를 skip한다.

IHDR Chunk 함수로 바뀐 파일정보를 수정해주며

FindIDATChunk함수를 통하여 IDATChunk를 찾아내며

그 이후 Data의 배열방식에 따라 R G B의 값을 찾아내며 그 값의 평균이 127을 넘기면 1 넘기지 못하면 0으로 처리하고 필요한 pixels 수만큼 반복한다. 마지막으로 원래크기 pixel 의 1/4로 처리해야 하는 데이터가 줄어들었으므로 마지막 IEND Chunk함수로 IEND Chunk를 따로 만들어 주어야 한다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

SkipHeader : 처음에 바로 사용하는 함수로 PNG의 시그니처부분을 건너뛰는 함수이다. 시그니처함수는 16byte 길이를 갖고 있으므로 i = 0으로 설정하여 16이 되기전까지 16번 반복하여 1byte씩 읽고 쓴다. R1는 ReadPointer로 주어진 값을 읽을떄마다 1씩 증가하게 하고 R2는 WritePointer로 값을 STR할떄마다 1씩 증가하게 해야한다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

IHDRChunk 함수는 PNG의 정보를 담고있는 IHDRChunk의 값을 수정하여 write를 한다.

IHDRChunk의 첫 정보는 width 와 height 로 C++코드를 보면 width와 height를 parameter로 가져 주어진 값을 8byte 16진수로 변환하는 함수인 DecimaltoHex는 다음그림과 같다.

텍스트, 폰트, 번호, 라인이(가) 표시된 사진

자동 생성된 설명

256의 세제곱으로 나눈 값을 write하고 그 나머지를 256의 제곱으로 나누고 또 write한다 다음과정을 반복하여 표현할 수 있다. Square함수는 제곱 함수이다.

그 이외 bpp와 color compression 등 여러 정보를 입력하여 넣을 수 있게 이루어져있다.

ARM Code의 경우 주어진 Width를 자동으로 16진수로 변환되어 레지스터에 들어가게 되는데,

STRB로 저장할 경우 LSB의 2Byte만 저장되기 때문에 반대로 저장이 된다. ( C0 03 00 00 순으로)

따라서 주어진 값을 ROR로 right shift를 24bit씩 하여 오른쪽으로 6byte씩 이동하여 (00 00 03 C0)

로 저장하게 한다.height도 이와 같이 적용되며 다른 정보들도 주어진 상수값을 쓰도록 되어있다.

IHDR Chunk 이후 값을 읽어야 하므로 R1도 write한만큼 1씩 계속 증가하게 하였다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

FindIDATChunkAndCopyPaste 함수는 함수명 그대로 2byte씩 읽어서 IDAT Chunk를 찾는 함수이다.

IDAT를 찾기위해서는 C++에서는 data[0] data[1]이, ARM Code에서는 R3,R4가 아스키코드를 역변환하여 IDAT를 10진법으로 73 68 65 84를 찾게되고 IDAT의 T까지 복사를 하여 Write를 한다.

텍스트, 라인, 도표, 스크린샷이(가) 표시된 사진

자동 생성된 설명

IDAT는 두가지 Case로 찾을 수 있다. R3가 I와 같거나 R4가 I가 되어서 IDAT를 찾는다.

따라서 If문으로 나머지 DAT도 R3,R4와 같은 지 확인을 하고 write를 한다.

찾지 못하였을 경우 ARM code에서는 NotFound LABEL로 이동을 하여 R3와 R4를 STA하고 다시 찾을 때 까지 반복을 한다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

RGBAtoBinary함수는 function3에 해당하는 함수로

4byte씩 데이터를 읽어서 RGB를 추출한 뒤 세 개의 값의 평균을 구하고 그 값이 127보다 크면 1 작으면 0으로 처리해서 write를 하는 함수이다.픽셀수 만큼 반복해야 하므로 960x160인 153600번 반복하게 되어 있다. ARM Code의 경우 나눗셈을 구현하면 복잡하고 오래 걸리기 떄문에 127을 381로 수정하여 3을 나누지 않고 비교하도록 되어 있다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

IENDChunk 함수는 PNG의 마지막부분을 표현하는 Chunk로 00 00 00 00 I E N D AE 42 60 82로 되어있으며 c++와 ARM code 모두 해당하는 값을 직접 넣었다. ARM Code의 경우 LSB의 2byte씩 STR하므로 D N E I 순서로 값을 지정하고 STR하고 4Byte ROR하고 STR하는 식으로 코드를 구현하였다.

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

ARM code 결과 Memory Map PNG 파일 Memory Map

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

C Code 결과

ARM code와 C code 의 결과가 동일함을 알 수 있다.

텍스트, 폰트, 화이트, 영수증이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명RGB 순의 배열의 경우 Alpha 값이 없으므로 3Byte씩 읽어서 그대로 계산하므로 코드의 큰 변화는 없다.

텍스트, 폰트, 번호, 스크린샷이(가) 표시된 사진

자동 생성된 설명  
RRR…GGG…BBB…배열의 경우

R과 G와 B사이에 Pixels의 개수인 153600개가 존재하므로 153600번지 만큼 건너뛰면서 R,G,B의 값을 LDA하는 부분이 다르다.

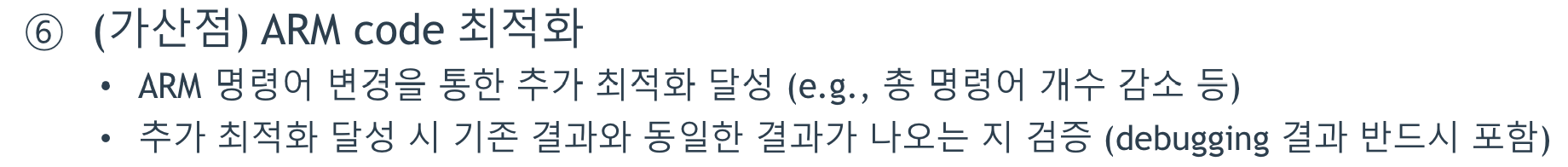
텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

RGBtoBinary배열의 Memory Map RRGGBBtoBinary배열의 Memory Map

두 Memory Map 모두 기존 RGBAtoBinary의 Memory Map과 같다.



최적화한 코드

STR/LDR RD,[RM] STR/LDR RD,[RM],#1

ADD RM,#1

LDA와 STR을 사용함과 동시에 Rm에 Immediate value를 더하여 최적화

폰트, 텍스트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명  
 변경전 변경이후

텍스트, 전자제품, 스크린샷, 디스플레이이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 화이트이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

Memory map을 확인할 시 동일한 결과를 확인 할 수 있다.

483.743ms -> 406.933ms 15.89% 속도 향상

205줄 -> 156줄 23.9% 총 명령어 수 감소

Branch 명령어에서 S를 사용하여 CPSR을 동시에 설정하여 코드수와 시간 최적화



텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

앞 사진과 비교하여 결과 같음을 알 수 있다.

406.933ms -> 391.571ms로 3.78% 성능향상 총 명령어 개수 1개 감소

반복되는 STRB ROR 구간을 SUB routine을 만들어서 한꺼번에 계산

IHDR Chunk에서 ADD r1,#1을 IHDR의 길이는 고정임을 이용하여 ADD r1,#13으로 한꺼번에 처리

변경전텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 번호, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

IHDRChunk와

IENDChunk

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명Sub routine 으로 4번 ROR하고 STRB하므로 r4에 #4부여

SUB1에 가기전 r5에 저장할 데이터 r6에 rotation value를 넣고 ROR을 사용하고 저장하는 함수이다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명  
변경후 IDHR Chunk와 IEND Chunk 구간

BL SUB1로 가기전에 R14에 돌아갈 위치를 설정하였고,

R5와 R6에 값을 넣어주고 레지스터는 유지해도 되므로 R14만 스택에 저장하도록 하였다.

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

391.571ms -> 391.582ms로 0.011ms 증가 하였으나

총 명령어 155줄 -> 129줄로 26줄 감소하여

전체적인 면에서 향샹을 확인 할 수 있음

RGBA에서 Alpha data는 Gray 계산에 필요로 하지 않는다.

따라서 R6는 Alpha 채널 Bit Data이므로 LDRB를 할 필요없이 r5에서 r1을 #2씩 증가시켜 계산

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

391.582ms ->345.502ms로 13.3%향상과 총 명령어 1개 감소로 성능 향상

텍스트, 라인, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 라인, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 폰트, 라인, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

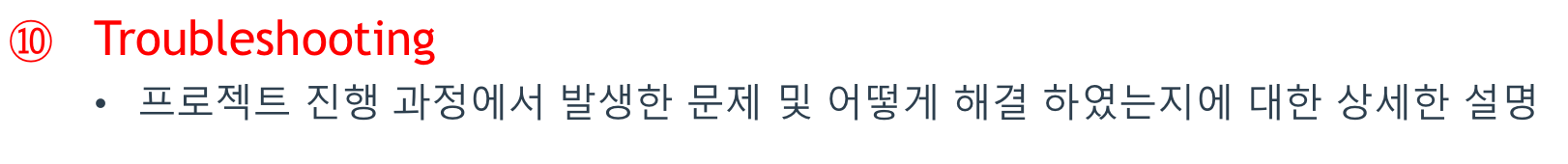
최적화 이전

RGBA 배열 488.743 ms

RGB 배열 422.303ms

RRGGBB 배열 599.096ms  
로 RRGGBB배열이 가장 오래 걸렸다. 아마도 R과 G와 B사이의 값이 멀리 있어서 세 값을 찾아서 계산하려면 복잡하기 때문인 것 같았다.

그와 반대로 RGB배열은 Alpha 채널이 없기 때문에 RGBA 배열보다 더 빠른 속도를 보여주었다.

1. C언어에서 파일을 읽었을 때 나온 데이터 값이 binary형태여서 C언어는 이를 더하거나 계산하는 것이 불가능해서 다른 type으로 변환 필요

int R = data[0]의 형태로 int값으로 받아들여서 계산이 가능하여 해결

2. ARM code에서 파일을 STA하거나 LDA할 때 32bit단위로 STA하거나 LDA하여서 데이터를 1byte씩 읽지 않아서 문제 발생

STRB,LDAB 명령어를 통해 byte 단위로 읽어내는 것이 가능하여 해결

3. IHDR chunk에서 주어진 width와 height를 바르게 고쳐야 하는데 주어진 값을 4byte(32bit)크기의 hex로 변환하는 과정이 까다로움

해결

텍스트, 폰트, 영수증, 화이트이(가) 표시된 사진

자동 생성된 설명

IHDR영역을 찾는 함수 (o)

IHDR영역을 function에 맞게 수정하는 함수(o)

IDAT영역을 찾는 함수 (o)

IEND영역을 만드는 함수 (o)

ㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡ

시간 남으면 하는 것들

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

PNG Encoding Decoding 하는 함수 , library 참고하면서 해야함. Filtering은 없으므로 x