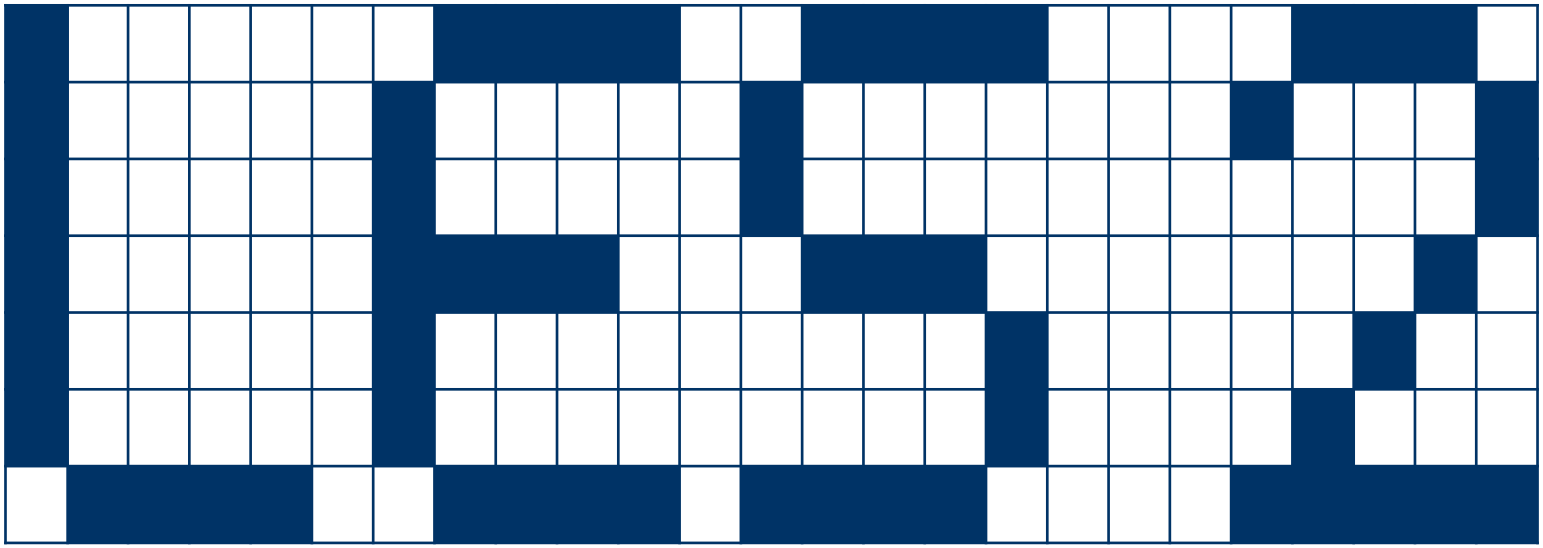
Hogeschool van Arnhem en Nijmegen

EVD1 – Vision operators



HAN

Performance optimization

Op de laptop hebben we genoeg rekenkracht, maar op het embedded systeem??

Dus: hoe sneller de algoritmes, des te beter!!

Bijvoorbeeld:
Contrast stretching is een bewerkelijke operatie doordat:
- alle pixels (25344) lezen (laagste en hoogste bepalen)
- stretch factor berekenen
- alle pixels schrijven (berekeningen met een kommagetal)

Wat kunnen we doen?

Performance optimization

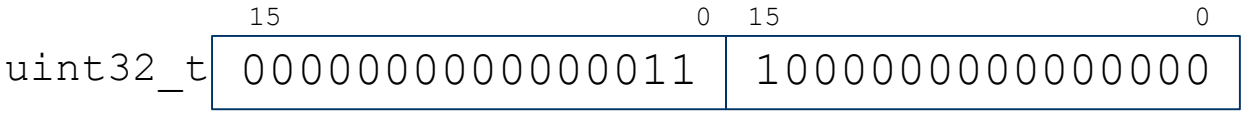
1. De microcontroller heeft een FPU!

Deze wordt enabled in de compiler opties
(is enabled in het gegeven project)

Zet de Floating Point calculations eens op softwarematig en bekijk de performance verschillen!!
(via Project - Options for Target ‘evdk’… - tabblad Target)

Ga je nu ‘double’ of ‘float’ gebruiken?

Alternatief: implementeer Fixed Point calculations



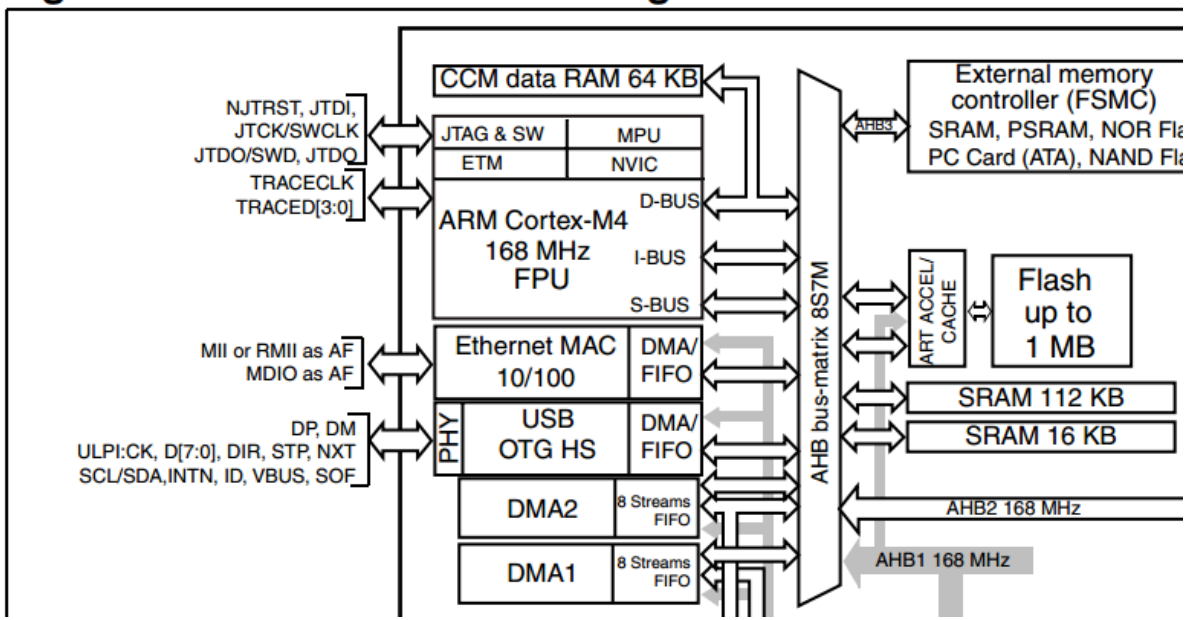

```
Binair: 229376

Fixed point : 3,5 !!
```

Performance optimization

2. Gebruik register variabelen !!

Er is vééééééééééééééél minder memory access nodig

Figure 5. STM32F40x block diagram



Bron: STM32F407xx datasheet

Performance optimization

3. Vergelijk met nul in loops

```
i = src->width * src->height;
while(i-- > 0)
{
   ...
}
```

```
for(i=0; i < (src->width * src->height); i++)
{
   ...
}
```

1 keer

```
   353:   i = src->width * src->height;
0x08005212 8807      LDRH          r7,[r0,#0x00]
0x08005214 F8B08002  LDRH          r8,[r0,#0x02]
0x08005218 FB07F708  MUL           r7,r7,r8
0x0800521C B23B      SXTH          r3,r7
```

25344 keer

```
   354:   while(i-- > 0)
0x08005232 1E1F      SUBS          r7,r3,#0
0x08005234 F1A30801  SUB           r8,r3,#0x01
0x08005238 FA0FF388  SXTH          r3,r8
0x0800523C DCF0      BGT           0x08005220
```

```
   364:   for(i=0; i < (src->width * src->height); i++)
0x0800525C F10C0701  ADD           r7,r12,#0x01
0x08005260 FA0FFC87  SXTH          r12,r7
0x08005264 8807      LDRH          r7,[r0,#0x00]
0x08005266 F8B08002  LDRH          r8,[r0,#0x02]
0x0800526A FB07F708  MUL           r7,r7,r8
0x0800526E 4567      CMP           r7,r12
0x08005270 DCEB      BGT           0x0800524A
```

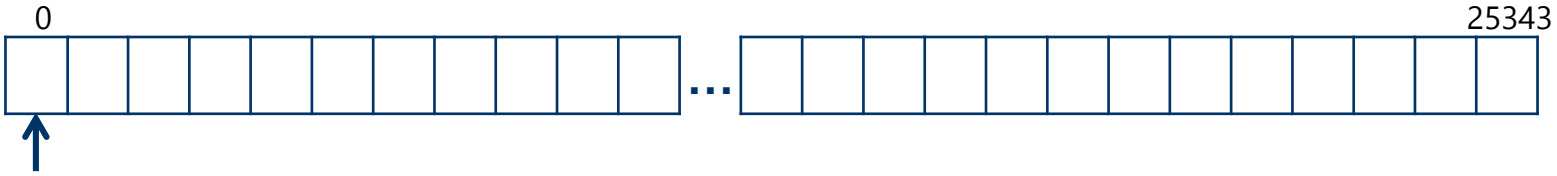
Performance optimization

4. Gebruik pointers naar de source en destination

- pre- en/of post increment/decrement
- geen loops-in-loops

```
for(y = 0; y < src->height; y++)
{
  for(x = 0; x < src->width; x++)
  {
    ...
  }
}
```


```
uint8_t *s = (uint8_t *)src->data;
```




```
uint8_t *d = (uint8_t *)dst->data;
```

Performance optimization

5. Loop unrolling

Het testen van de conditie in een loop kost instructies en dus executietijd

0 keer

```
lPixel = 255;
hPixel = 0;
uint8_t *s = (uint8_t *)src->data;

if (*s < lPixel) {lPixel = *s;}
if (*s > hPixel) {hPixel = *s;}
s++;

if (*s < lPixel) {lPixel = *s;}
if (*s > hPixel) {hPixel = *s;}
s++;

...

if (*s < lPixel) {lPixel = *s;}
if (*s > hPixel) {hPixel = *s;}
s++;
```

Maar veel meer flash nodig…

25344 keer

```
lPixel = 255;
hPixel = 0;
uint16_t i = (src->width * src->height);
uint8_t *s = (uint8_t *)src->data;

while(i)
{
  if(*s < lPixel) {lPixel = *s;}
  if(*s > hPixel) {hPixel = *s;}
  s++; i--;
}
```

25344 / 2 keer

```
lPixel = 255;
hPixel = 0;
uint16_t i = (src->width * src->height);
uint8_t *s = (uint8_t *)src->data;

while(i)
{
  if(*s < lPixel) {lPixel = *s;}
  if(*s > hPixel) {hPixel = *s;}
  s++; i--;

  if(*s < lPixel) {lPixel = *s;}
  if(*s > hPixel) {hPixel = *s;}
  s++; i--;
}
```

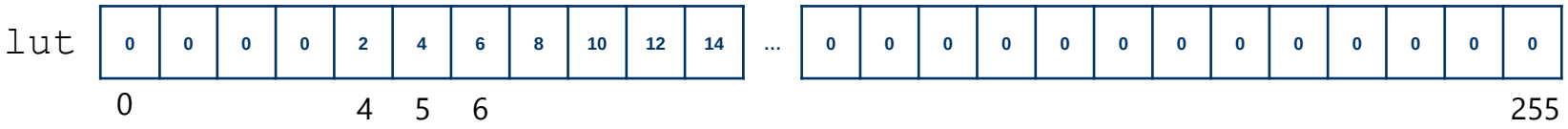
Performance optimization

6. Construeer een lookup table

Uitgangspunt: alle src pixels met een bepaalde grijswaarde, krijgen na een berekening allemaal dezelfde nieuwe grijswaarde (bijvoorbeeld bij ContrastStretch)

$$p_{dst} = \left((p_{src} - lpixel) \cdot factor\right) + 0{,}5 + bottom$$

- ‘Complexe’ berekeningen slechts maximaal 256 keer uitvoeren

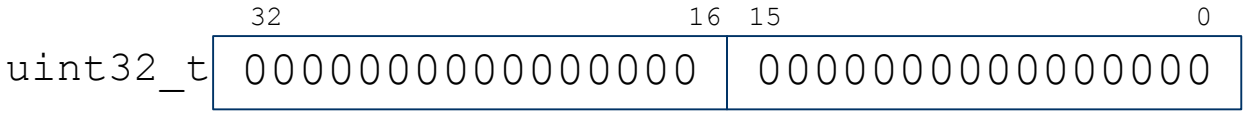


- Uitkomst gebruiken om een waarde toe te kennen

```
dst->data[y][x] = lut[ src->data[y][x] ];
```

Performance optimization

7. Gebruik voor een (lokale) teller 32-bit variabelen !



Stel je moet tellen van 0 tot 100

```
uint8_t i = 0;

while(i > 100)
{
  …
  i++;
}
```

Niet efficiënt voor 32-bit microcontrollers!
Registers en RAM-geheugen zijn namelijk 32-bit.

Gebruik je een uint8_t, dan moet de compiler er zeker van zijn dat 255 + 1 = 0.

Dit is een extra instructie (welke?) per optelling!!

Performance optimization

8. Lees/schrijf zo min mogelijk het RAM geheugen !

Alle operaties worden op de registers uitgevoerd.
Geheugen access is dan dus een extra operatie.
Lees/schrijf als het mogelijk is 4 pixels tegelijk!

```
register uint32_t *a = (uint32_t *)src->data;
register uint32_t *b = (uint32_t *)dst->data;
register int32_t i;

i = (src->width * src->height)/4;
while(i-- > 0)
{
  *b++ = *a++;
}
```

TIP: een 32-bit variabele bit-shiften kost minder instructies dan lezen uit het geheugen

Performance optimization

Opdracht

Implementeer de functie:

```
void vContrastStretchFast(image_t *src,
                          image_t *dst);
```

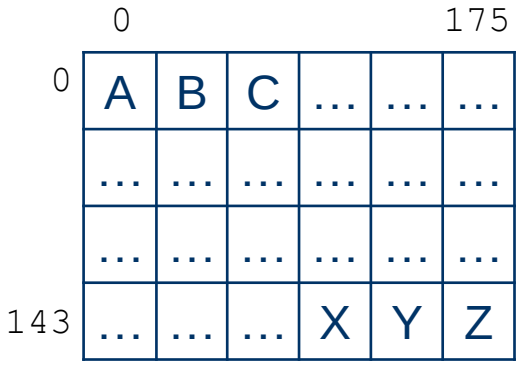
Uitgangspunten:
- stretch altijd van 0 t/m 255
- het algoritme combineert meerdere van de behandelde technieken
- algoritme heeft op de target een benchmark tijd van maximaal 4 ms

Record: minder dan 2,5 ms !

Performance optimization

Rotate 180

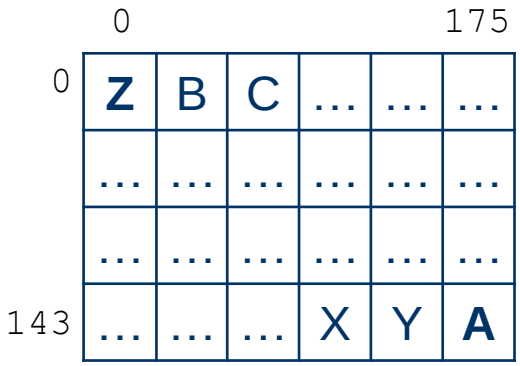
Links boven wordt rechts onder en vice versa.



Performance optimization

Rotate 180

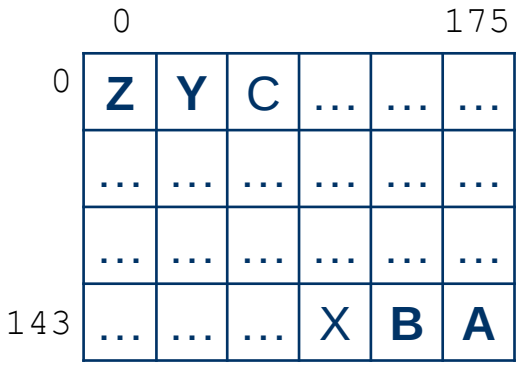
Links boven wordt rechts onder en vice versa.



Performance optimization

Rotate 180

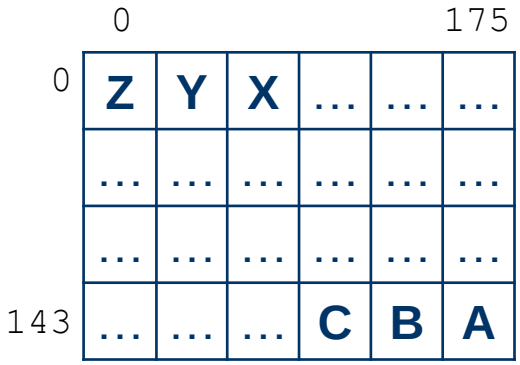
Links boven wordt rechts onder en vice versa.



Performance optimization

Rotate 180

Links boven wordt rechts onder en vice versa.

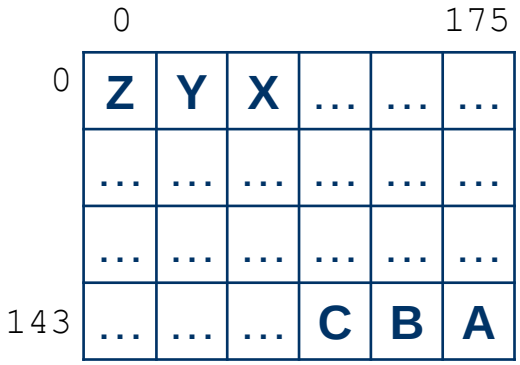


Performance optimization

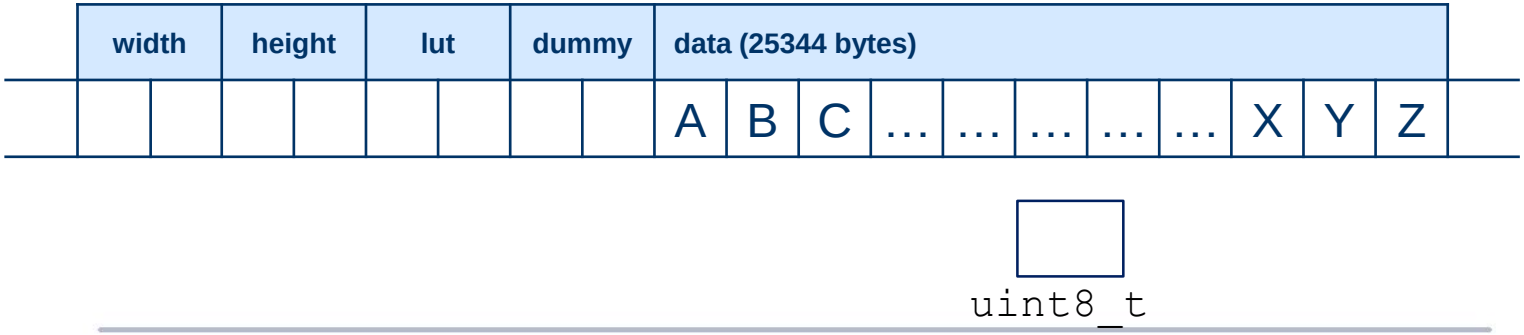
Rotate 180

Links boven wordt rechts onder en vice versa.

Oplossen dmv tijdelijke variabele:



Geheugen:

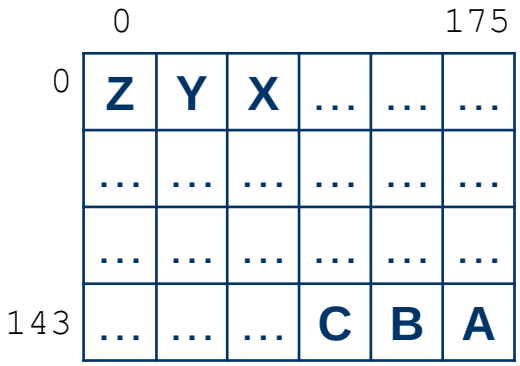


Performance optimization

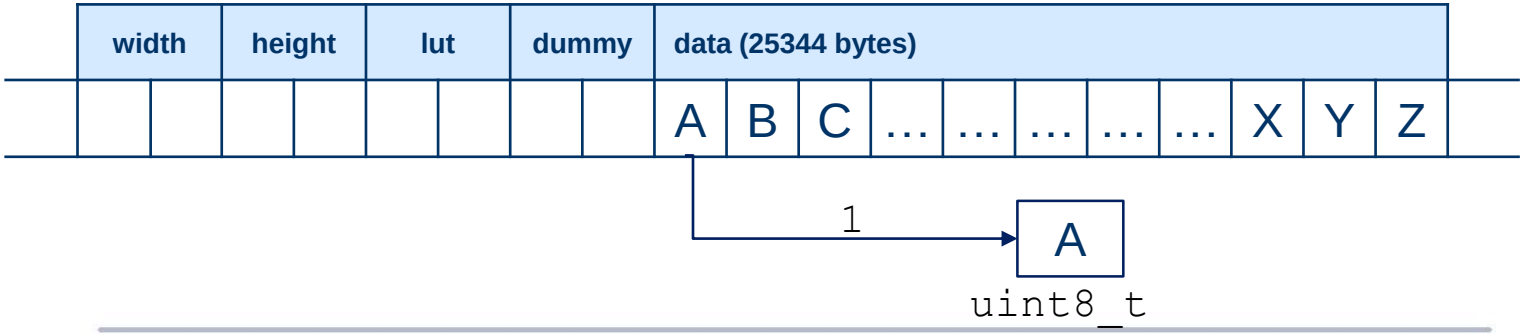
Rotate 180

Links boven wordt rechts onder en vice versa.

Oplossen dmv tijdelijke variabele:



Geheugen:

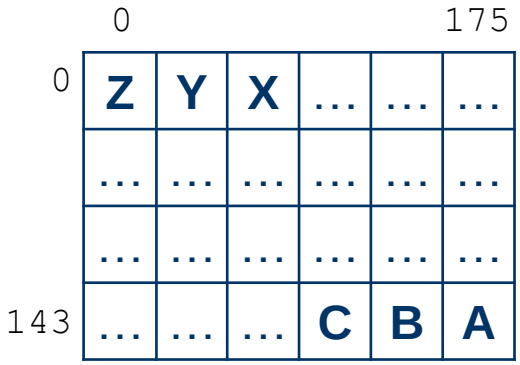


Performance optimization

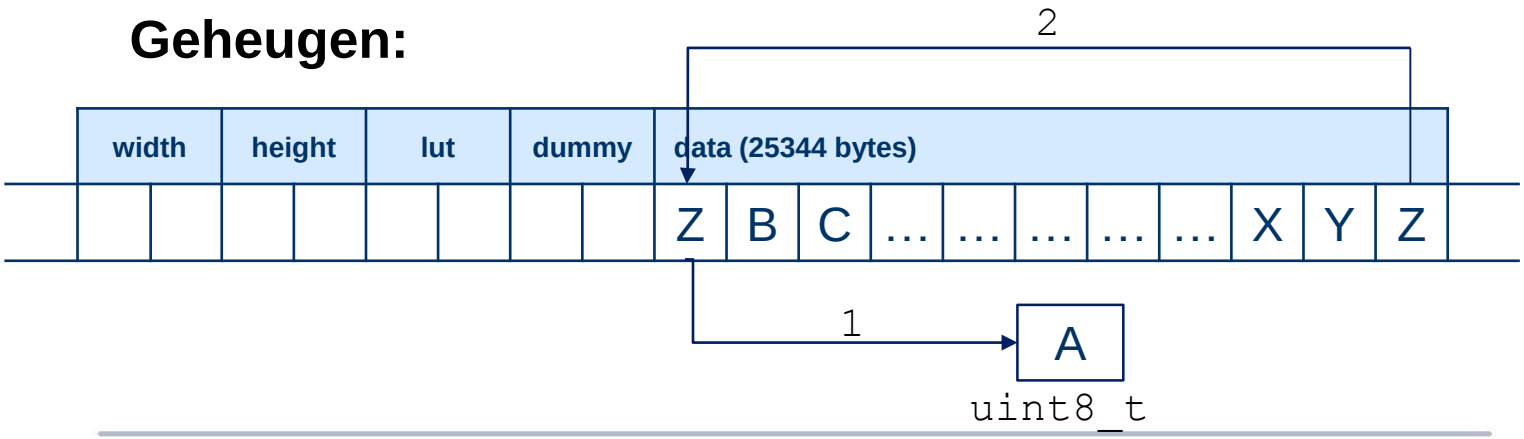
Rotate 180

Links boven wordt rechts onder en vice versa.

Oplossen dmv tijdelijke variabele:



Geheugen:

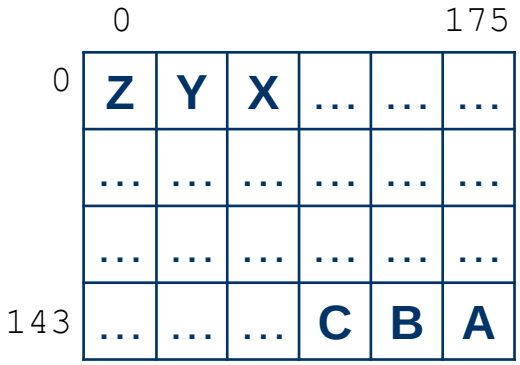


Performance optimization

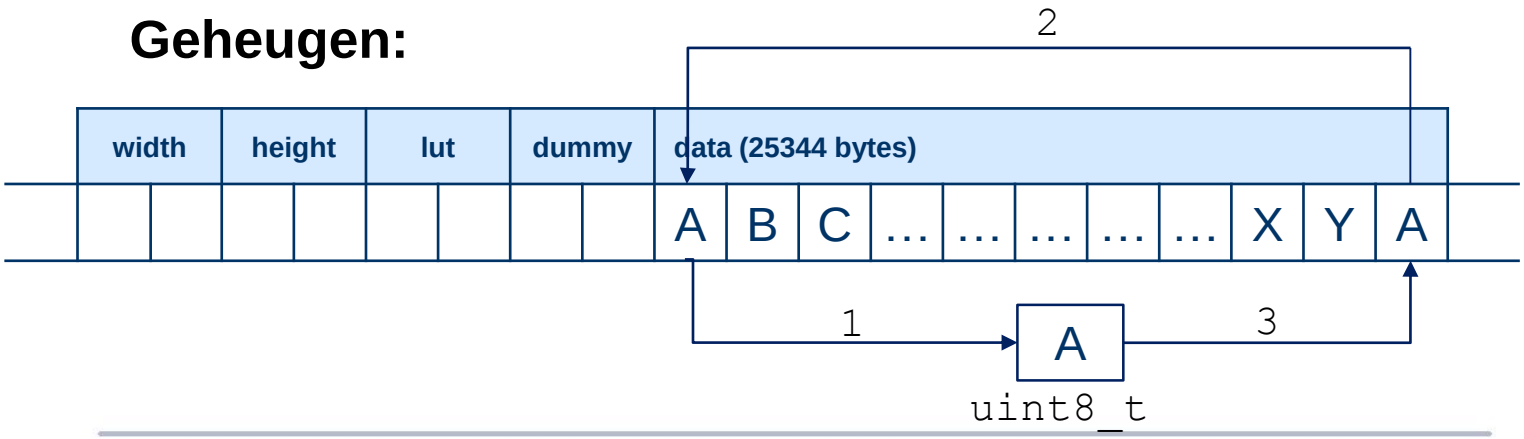
Rotate 180

Links boven wordt rechts onder en vice versa.

Oplossen dmv tijdelijke variabele:



Geheugen:



Performance optimization

Opdracht

Implementeer de functie:

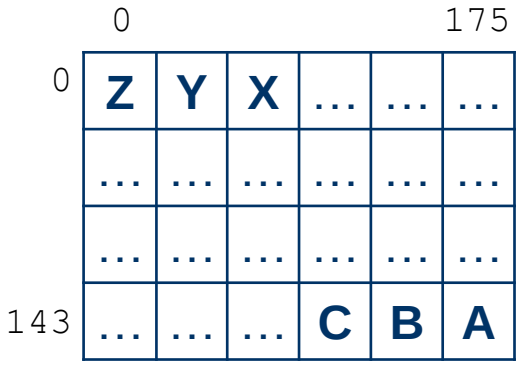
```
void vRotate180(image_t *img);
```

Uitgangspunten:
- **Implementatie zoals hiervoor beschreven**
- **Werking testen in Qt**
- **Werking testen op target**
- **Noteer benchmark tijd in je logboek**

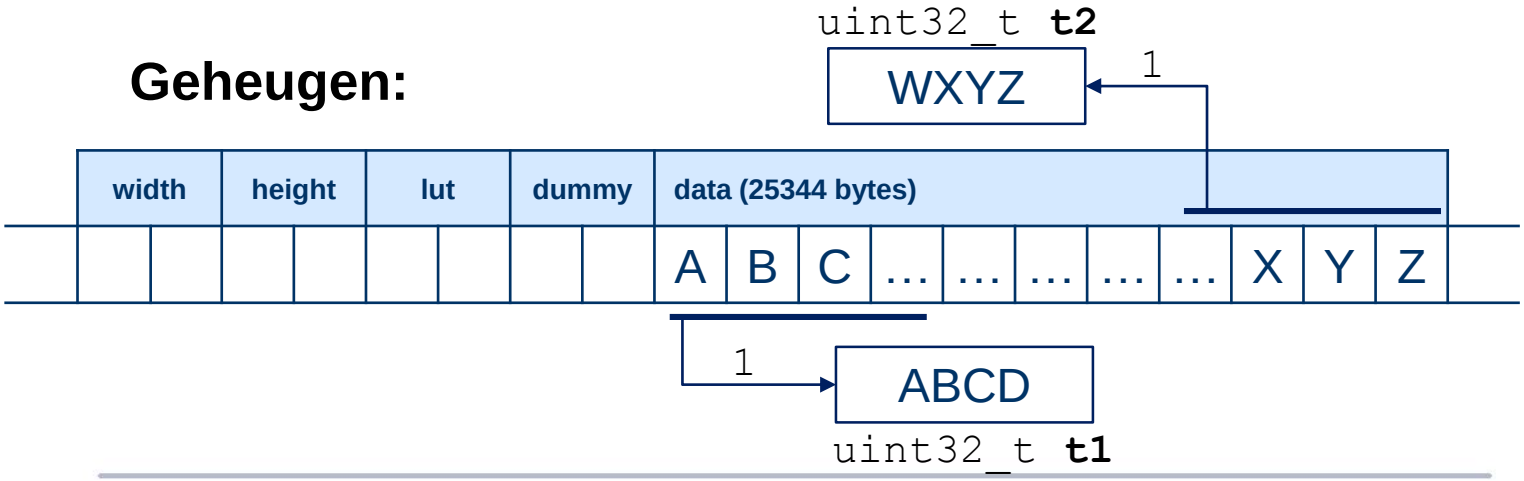
Performance optimization

Rotate 180

*Maar we kunnen op de **target** enorme tijdwinst behalen door 4 pixels tegelijk te lezen in een uint32_t …*



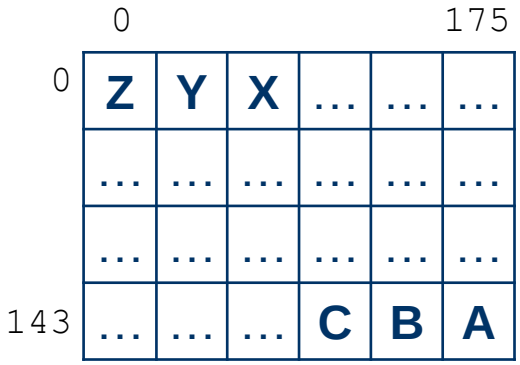
Geheugen:



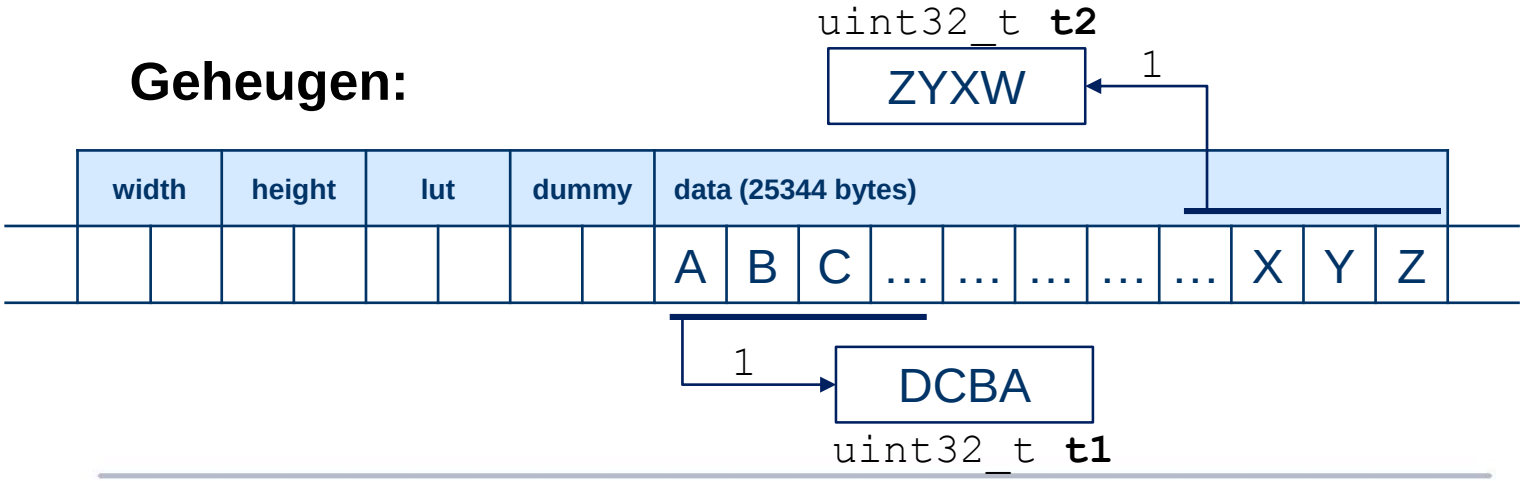
Performance optimization

Rotate 180

… de volgorde met één instructie te wijzigen …



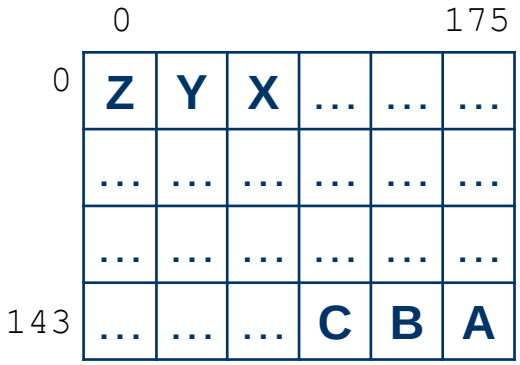
Geheugen:



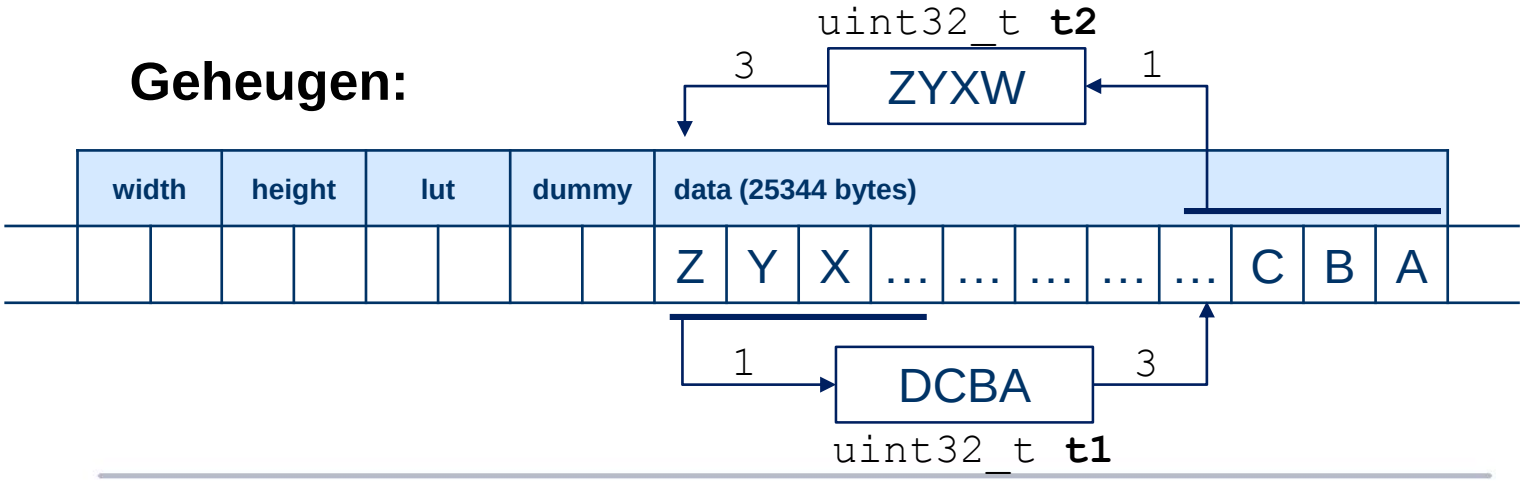
Performance optimization

Rotate 180

… en het resultaat met 4 pixels tegelijk terug te schrijven!



Geheugen:



Performance optimization

Rotate 180

Enkele tips:

```
#ifdef QDEBUG_ENABLE
  ...
#else
  ...
#endif

// Point to first four pixels
register uint32_t *a = (uint32_t *)img->data;
// Point to last four pixels
register uint32_t *b = (uint32_t *)&img->data[img->height-1][img->width-4];


// Read pixels
t1 = *a;
t2 = *b;


// Reverse pixels
__asm
{
  // Reverse 32-bit byte order : b3 b2 b1 b0 -> b0 b1 b2 b3
  REV t1, t1;
  REV t2, t2;
}
```

Performance optimization

Opdracht

Implementeer de functie:

```
void vRotate180(image_t *img);
```

Uitgangspunten:
- voor de target met inline assembly
- het algoritme combineert meerdere van de behandelde technieken
- algoritme heeft op de target een benchmark tijd van maximaal 1 ms

Record: minder dan 0,2 ms !