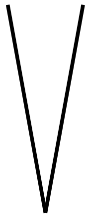
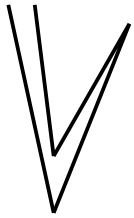
The paper folding number

Assign digits to folding directions: 1= left turn; 2 = right turn



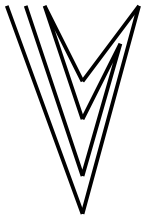
1

Only **one fold**



110

Three folds:
The first one stays, but beforeand behind we add new folds in the same and opposite direction.



1101100

Seven folds:
Now we have the first three folds like in the step before, than a left fold and then three folds in the opposite direction and inverse order of the preceeding step.

This leads to the iteration formula:

$S_0 = 1$
$S_1 = 110 = S_0 1 \underline{S}_0$
$S_2 = 1101100 = S_1 1 \underline{S}_1$
$S_3 = 110110011100100 = S_2 1 \underline{S}_2$
...
$S_{n+1} = S_n 1 \underline{S}_n$, with $\underline{S}$ = S where 1<>0 is exchanged and direction inversed

If you indicate the digits from left to right from 0 to $m=2^{n+1}-2$
then you can write:
$S_n = a^n_0 a^n_1 ... a^n_m$

Because you add the new folds between the old ones you get
$S_{n-1} = a^n_1 a^n_3 a^n_5 ...$

Find out by yourself, that every second digit alternates between 0 and 1. Then find out, that of the other digits every second alternates between 0 and 1 and so on...

So i came up with the idea of following algorithm:

```
for i = 0 to 2^(n+1)-2
  if i is even then {
    if int(i/2) is even then a^n_i = 1 else a^n_i = 0}
  else {if int(i/2) is even then {
           if int(i/4) is even then a^n_i = 1 else a^n_i = 0}}
         else{if int(i/4) is even then if int(i/8) ....
  endif
next
```

This is, what the short assembler routine testbits.s does.
The rest is done in c.