16 seg display and more like matrix

david vajda

February 10, 2026

1 14seg27xxx02022026.bak.txt

```
-- (c) david vajda
-- 02/02/2026
-- 14 segment anzeige fuer buchstaben

-- sorry - laut abbildung ist es 16 segment...

A:      a, b, c, d, h, g, u, p
B:      a, b, c, d, e, f, g, h, u, p
C:      a, b, e, f, g, h
D:      a, b, c, d, e, f, g, h
E:      a, b, e, f, g, h, u, p
F:      a, b, g, h, u, p
G:      a, b, d, e, f, g, h, p
H:      c, d, g, h, u, p
I:      m, s
J:      c, d, e, f, g
K:      d, g, h, n, u, p
L:      e, f, g, h
M:      c, d, f, g, k, m
N:      c, d, g, h, k, r
O:      a, b, c, d, e, f, g, h
P:      a, b, c, g, h, u, p
Q:      a, b, c, d, e, f, g, h, r
S:      a, b, d, e, f, h
T:      a, b, m, s
U:      c, d, e, f, g, h, r
V:      c, d, e, f, g, h
W:      c, d, r, s, g, h
X:      k, n, t, r
Y:      c, d, e, f, k, p
Z:      a, b, e, f, n, t
```

	a	b	c	d	e	f	g	h	k	m	n	p	s	r	t	u
A	H	H	H	H	L	L	H	H	L	L	L	H	L	L	L	H
B	H	H	H	H	H	H	H	H	L	L	L	L	L	L	L	H
C	H	H	L	L	H	H	H	H	L	L	L	L	L	L	L	L
D	H	H	H	H	H	H	H	H	L	L	L	L	L	L	L	L

E	H	H	L	L	H	H	H	H	L	L	L	H	L	L	L	H
F	H	H	L	L	L	L	H	H	L	L	L	H	L	L	L	H
G	H	H	L	H	H	H	H	H	L	L	L	H	L	L	L	L
H	L	L	H	H	L	L	H	H	L	L	L	H	L	L	L	H
I	L	L	L	L	L	L	L	L	L	H	L	L	H	L	L	L
J	L	L	H	H	H	H	H	L	L	L	L	L	L	L	L	L
K	L	L	L	H	L	L	H	L	L	L	H	H	L	L	L	H
L	L	L	L	L	H	H	H	H	L	L	L	L	L	L	L	L
M	L	L	H	H	L	H	H	L	H	H	L	L	L	L	L	L
N	L	L	H	H	L	L	H	H	H	L	L	L	L	H	L	L
O	H	H	H	H	H	H	H	H	L	L	L	L	L	L	L	L
P	H	H	H	L	L	L	H	H	L	L	L	H	L	L	L	H
Q	H	H	H	H	H	H	H	H	L	L	L	L	L	H	L	L
R	H	H	H	L	L	L	H	H	L	L	L	H	L	H	L	L
S	H	H	L	H	H	H	L	H	L	L	L	L	L	L	L	L
T	H	H	L	L	L	L	L	L	L	H	L	L	H	L	L	L
U	L	L	H	H	H	H	H	H	L	L	L	L	L	H	L	L
V	L	L	H	H	H	H	H	H	L	L	L	L	L	L	L	L
W	L	L	H	H	L	L	H	H	L	L	L	L	H	H	L	L
X	L	L	L	L	L	L	L	L	H	L	H	L	L	H	H	L
Y	L	L	H	H	H	H	L	L	H	L	L	H	L	L	L	L
Z	H	H	L	L	H	H	L	L	L	L	H	L	L	L	H	L

A: *a, *b, *c, *d, h, g, u, p
B: *a, *b, *c, *d, e, f, g, h, u, p
C: *a, *b, e, f, g, h
D: *a, *b, *c, *d, e, f, g, h
E: *a, *b, e, f, g, h, u, p
F: *a, *b, g, h, u, p
G: *a, *b, *d, e, f, g, h, p
H: *c, *d, g, h, u, p
I: m, s
J: *c, *d, e, f, g
K: *d, g, h, n, u, p
L: e, f, g, h
M: *c, *d, f, g, k, m
N: *c, *d, g, h, k, r
O: *a, *b, *c, *d, e, f, g, h
P: *a, *b, *c, g, h, u, p
Q: *a, *b, *c, *d, e, f, g, h, r
R: *a, *b, *c, g, h, u, p, r
S: *a, *b, *d, e, f, h
T: *a, *b, m, s
U: *c, *d, e, f, g, h, r
V: *c, *d, e, f, g, h
W: *c, *d, r, s, g, h
X: k, n, t, r
Y: *c, *d, e, f, k, p
Z: *a, *b, e, f, n, t

```
cat tmp1.txt | sed 's/[A-Z]\ \ \ \(.*\)/\1/g' | sed 's/H/1/g' | sed 's/L/0/g' | sed 's/
```

```
A   H   H   H   H   L   L   H   H   L   L   L   H   L   L   L   H
B   H   H   H   H   H   H   H   H   L   L   L   L   L   L   L   H
C   H   H   L   L   H   H   H   H   L   L   L   L   L   L   L   L
D   H   H   H   H   H   H   H   H   L   L   L   L   L   L   L   L
E   H   H   L   L   H   H   H   H   L   L   L   H   L   L   L   H
F   H   H   L   L   L   L   H   H   L   L   L   H   L   L   L   H
G   H   H   L   H   H   H   H   H   L   L   L   H   L   L   L   L
H   L   L   H   H   L   L   H   H   L   L   L   H   L   L   L   H
I   L   L   L   L   L   L   L   L   L   H   L   L   H   L   L   L
J   L   L   H   H   H   H   H   L   L   L   L   L   L   L   L   L
K   L   L   L   H   L   L   H   L   L   L   H   H   L   L   L   H
L   L   L   L   L   H   H   H   H   L   L   L   L   L   L   L   L
M   L   L   H   H   L   H   H   L   H   H   L   L   L   L   L   L
N   L   L   H   H   L   L   H   H   H   L   L   L   L   H   L   L
O   H   H   H   H   H   H   H   H   L   L   L   L   L   L   L   L
P   H   H   H   L   L   L   H   H   L   L   L   H   L   L   L   H
Q   H   H   H   H   H   H   H   H   L   L   L   L   L   H   L   L
R   H   H   H   L   L   L   H   H   L   L   L   H   L   H   L   L
S   H   H   L   H   H   H   L   H   L   L   L   L   L   L   L   L
T   H   H   L   L   L   L   L   L   L   H   L   L   H   L   L   L
U   L   L   H   H   H   H   H   H   L   L   L   L   L   H   L   L
V   L   L   H   H   H   H   H   H   L   L   L   L   L   L   L   L
W   L   L   H   H   L   L   H   H   L   L   L   L   H   H   L   L
X   L   L   L   L   L   L   L   L   H   L   H   L   L   H   H   L
Y   L   L   H   H   H   H   L   L   H   L   L   H   L   L   L   L
Z   H   H   L   L   H   H   L   L   L   L   H   L   L   L   H   L
```

uebersetzt:

```
1111001100010001
1111111100000001
1100111100000000
1111111100000000
1100111100010001
1100001100010001
1101111100010000
0011001100010001
0000000001001000
0011111000000000
0001001000110001
0000111100000000
0011011011000000
0011001110000100
1111111100000000
1110001100010001
```

```
1111111100000100
1110001100010100
1101110100000000
1100000001001000
0011111100000100
0011111100000000
0011001100001100
0000000010100110
0011110010010000
1100110000100010

cat tmp1.txt | sed 's/[A-Z]\ \ \ \(.*\)/\1/g' | sed 's/H/1/g' | sed 's/L/0/g' | sed 's/

0b
0b1111001100010001
0b1111111100000001
0b1100111100000000
0b1111111100000000
0b1100111100010001
0b1100001100010001
0b1101111100010000
0b0011001100010001
0b0000000001001000
0b0011111000000000
0b0001001000110001
0b0000111100000000
0b0011011011000000
0b0011001110000100
0b1111111100000000
0b1110001100010001
0b1111111100000100
0b1110001100010100
0b1101110100000000
0b1100000001001000
0b0011111100000100
0b0011111100000000
0b0011001100001100
0b0000000010100110
0b0011110010010000
0b1100110000100010
0b
0b
0buebersetzt:
0b
0b
0b1111001100010001
0b1111111100000001
0b1100111100000000
0b1111111100000000
0b1100111100010001
```

```
0b1100001100010001
0b1101111100010000
0b0011001100010001
0b0000000001001000
0b0011111000000000
0b0001001000110001
0b0000111100000000
0b0011011011000000
0b0011001110000100
0b1111111100000000
0b1110001100010001
0b1111111100000100
0b1110001100010100
0b1101110100000000
0b1100000001001000
0b0011111100000100
0b0011111100000000
0b0011001100001100
0b0000000010100110
0b0011110010010000
0b1100110000100010
```

```
 cat tmp1.txt | sed 's/[A-Z]\ \ \ \(.*\)/\1/g' | sed 's/H/1/g' | sed 's/L/0/g' | sed 's/
```

```
# (c) david vajda
# 02/02/2026
# python3 - 14 seg - ...

for x in 0b1111001100010001,0b1111111100000001,0b1100111100000000,0b1111111100000000,0b110
    print(hex(x))
```

```
0xf311
0xff01
0xcf00
0xff00
0xcf11
0xc311
0xdf10
0x3311
0x48
0x3e00
0x1231
0xf00
0x36c0
0x3384
0xff00
0xe311
0xff04
0xe314
```

```
0xdd00
0xc048
0x3f04
0x3f00
0x330c
0xa6
0x3c90
0xcc22

david@www001:~$ python3 python02022026seg16eeprom.py > python02022026seg16eeprom.c
david@www001:~$ python3 python02022026seg16eeprom.py > python02022026seg16eeprom.c.tmp
david@www001:~$


david@www001:~$ hexdump c02022026seg16eeprom.bin
0000000 11f3 01ff 00cf 00ff 11cf 11c3 10df 1133
0000010 4800 003e 3112 000f c036 8433 00ff 11e3
0000020 04ff 14e3 00dd 48c0 043f 003f 0c33 a600
0000030 903c 22cc
0000034
david@www001:~$


// (c) david vajda
// 02/02/2026
// c eeprom 27xxx 16 seg

#include <stdio.h>

int main (void) {
int seg16code [] =  {0xf311,0xff01,0xcf00,0xff00,0xcf11,0xc311,0xdf10,0x3311,0x48,0x3e00,0

int i;
for (i = 0;   i < 26;   i++)
    printf ("%c%c", (char)((seg16code[i]>>8) & 0xff), (char)(seg16code[i] & 0xff));

return 0;
}


// (c) david vajda
// 02/02/2026
// c eeprom 27xxx 16 seg

#include <stdio.h>

int main (void) {
int seg16code [] =  {0xf311,0xff01,0xcf00,0xff00,0xcf11,0xc311,0xdf10,0x3311,0x48,0x3e00,0

int i;
```

```
for (i = 0;   i < 26;   i++)
    printf ("%c%c", (char)((seg16code[i]) & 0xff), (char)((seg16code[i] >> 8) & 0xff));

return 0;
}

david@www001:~$ gcc c02022026seg16eeprom.c -o c02022026seg16eeprom
david@www001:~$ ./c02022026seg16eeprom > c02022026seg16eeprom.bin
david@www001:~$ hexdump c02022026seg16eeprom.bin
0000000 f311 ff01 cf00 ff00 cf11 c311 df10 3311
0000010 0048 3e00 1231 0f00 36c0 3384 ff00 e311
0000020 ff04 e314 dd00 c048 3f04 3f00 330c 00a6
0000030 3c90 cc22
0000034
david@www001:~$


0000000 f311 ff01 cf00 ff00 cf11 c311 df10 3311
0000010 0048 3e00 1231 0f00 36c0 3384 ff00 e311
0000020 ff04 e314 dd00 c048 3f04 3f00 330c 00a6
0000030 3c90 cc22
0000034
```

2 14seg27xxx02022026.txt

```
-- (c) david vajda
-- 02/02/2026
-- 14 segment anzeige fuer buchstaben

-- sorry - laut abbildung ist es 16 segment...

A:      a, b, c, d, h, g, u, p
B:      a, b, c, d, e, f, g, h, u, p
C:      a, b, e, f, g, h
D:      a, b, c, d, e, f, g, h
E:      a, b, e, f, g, h, u, p
F:      a, b, g, h, u, p
G:      a, b, d, e, f, g, h, p
H:      c, d, g, h, u, p
I:      m, s
J:      c, d, e, f, g
K:      d, g, h, n, u, p
L:      e, f, g, h
M:      c, d, f, g, k, m
N:      c, d, g, h, k, r
O:      a, b, c, d, e, f, g, h
P:      a, b, c, g, h, u, p
Q:      a, b, c, d, e, f, g, h, r
S:      a, b, d, e, f, h
```

T: a, b, m, s
U: c, d, e, f, g, h, r
V: c, d, e, f, g, h
W: c, d, r, s, g, h
X: k, n, t, r
Y: c, d, e, f, k, p
Z: a, b, e, f, n, t

	a	b	c	d	e	f	g	h	k	m	n	p	s	r	t	u
A	H	H	H	H	L	L	H	H	L	L	L	H	L	L	L	H
B	H	H	H	H	H	H	H	H	L	L	L	L	L	L	L	H
C	H	H	L	L	H	H	H	H	L	L	L	L	L	L	L	L
D	H	H	H	H	H	H	H	H	L	L	L	L	L	L	L	L
E	H	H	L	L	H	H	H	H	L	L	L	H	L	L	L	H
F	H	H	L	L	L	L	H	H	L	L	L	H	L	L	L	H
G	H	H	L	H	H	H	H	H	L	L	L	H	L	L	L	L
H	L	L	H	H	L	L	H	H	L	L	L	H	L	L	L	H
I	L	L	L	L	L	L	L	L	L	H	L	L	H	L	L	L
J	L	L	H	H	H	H	H	L	L	L	L	L	L	L	L	L
K	L	L	L	H	L	L	H	L	L	L	H	H	L	L	L	H
L	L	L	L	L	H	H	H	H	L	L	L	L	L	L	L	L
M	L	L	H	H	L	H	H	L	H	H	L	L	L	L	L	L
N	L	L	H	H	L	L	H	H	H	L	L	L	L	H	L	L
O	H	H	H	H	H	H	H	H	L	L	L	L	L	L	L	L
P	H	H	H	L	L	L	H	H	L	L	L	H	L	L	L	H
Q	H	H	H	H	H	H	H	H	L	L	L	L	L	H	L	L
R	H	H	H	L	L	L	H	H	L	L	L	H	L	H	L	L
S	H	H	L	H	H	H	L	H	L	L	L	L	L	L	L	L
T	H	H	L	L	L	L	L	L	L	H	L	L	H	L	L	L
U	L	L	H	H	H	H	H	H	L	L	L	L	L	H	L	L
V	L	L	H	H	H	H	H	H	L	L	L	L	L	L	L	L
W	L	L	H	H	L	L	H	H	L	L	L	L	H	H	L	L
X	L	L	L	L	L	L	L	L	H	L	H	L	L	H	H	L
Y	L	L	H	H	H	H	L	L	H	L	L	H	L	L	L	L
Z	H	H	L	L	H	H	L	L	L	L	H	L	L	L	H	L

A: *a, *b, *c, *d, h, g, u, p
B: *a, *b, *c, *d, e, f, g, h, u, p
C: *a, *b, e, f, g, h
D: *a, *b, *c, *d, e, f, g, h
E: *a, *b, e, f, g, h, u, p
F: *a, *b, g, h, u, p
G: *a, *b, *d, e, f, g, h, p
H: *c, *d, g, h, u, p
I: m, s
J: *c, *d, e, f, g
K: *d, g, h, n, u, p
L: e, f, g, h
M: *c, *d, f, g, k, m
N: *c, *d, g, h, k, r

```
O:      *a, *b, *c, *d, e, f, g, h
P:      *a, *b, *c, g, h, u, p
Q:      *a, *b, *c, *d, e, f, g, h, r
R:      *a, *b, *c, g, h, u, p, r
S:      *a, *b, *d, e, f, h
T:      *a, *b, m, s
U:      *c, *d, e, f, g, h, r
V:      *c, *d, e, f, g, h
W:      *c, *d, r, s, g, h
X:      k, n, t, r
Y:      *c, *d, e, f, k, p
Z:      *a, *b, e, f, n, t
```

```
cat tmp1.txt | sed 's/[A-Z]\ \ \ \(.*\)/\1/g' | sed 's/H/1/g' | sed 's/L/0/g' | sed 's/
```

```
A   H   H   H   H   L   L   H   H   L   L   L   H   L   L   L   H
B   H   H   H   H   H   H   H   H   L   L   L   L   L   L   L   H
C   H   H   L   L   H   H   H   H   L   L   L   L   L   L   L   L
D   H   H   H   H   H   H   H   H   L   L   L   L   L   L   L   L
E   H   H   L   L   H   H   H   H   L   L   L   H   L   L   L   H
F   H   H   L   L   L   L   H   H   L   L   L   H   L   L   L   H
G   H   H   L   H   H   H   H   H   L   L   L   H   L   L   L   L
H   L   L   H   H   L   L   H   H   L   L   L   H   L   L   L   H
I   L   L   L   L   L   L   L   L   L   H   L   L   H   L   L   L
J   L   L   H   H   H   H   H   L   L   L   L   L   L   L   L   L
K   L   L   L   H   L   L   H   L   L   L   H   H   L   L   L   H
L   L   L   L   L   H   H   H   H   L   L   L   L   L   L   L   L
M   L   L   H   H   L   H   H   L   H   H   L   L   L   L   L   L
N   L   L   H   H   L   L   H   H   H   L   L   L   L   H   L   L
O   H   H   H   H   H   H   H   H   L   L   L   L   L   L   L   L
P   H   H   H   L   L   L   H   H   L   L   L   H   L   L   L   H
Q   H   H   H   H   H   H   H   H   L   L   L   L   L   H   L   L
R   H   H   H   L   L   L   H   H   L   L   L   H   L   H   L   L
S   H   H   L   H   H   H   L   H   L   L   L   L   L   L   L   L
T   H   H   L   L   L   L   L   L   L   H   L   L   H   L   L   L
U   L   L   H   H   H   H   H   H   L   L   L   L   L   H   L   L
V   L   L   H   H   H   H   H   H   L   L   L   L   L   L   L   L
W   L   L   H   H   L   L   H   H   L   L   L   L   L   H   H   L   L
X   L   L   L   L   L   L   L   L   H   L   H   L   L   H   H   L
Y   L   L   H   H   H   H   L   L   H   L   L   H   L   L   L   L
Z   H   H   L   L   H   H   L   L   L   L   H   L   L   L   H   L
```

```
uebersetzt:
```

```
1111001100010001
1111111100000001
1100111100000000
```

```
1111111100000000
1100111100010001
1100001100010001
1101111100010000
0011001100010001
0000000001001000
0011111000000000
0001001000110001
0000111100000000
0011011011000000
0011001110000100
1111111100000000
1110001100010001
1111111100000100
1110001100010100
1101110100000000
1100000001001000
0011111100000100
0011111100000000
0011001100001100
0000000010100110
0011110010010000
1100110000100010
```

```
cat tmp1.txt | sed 's/[A-Z]\ \ \ \(.*\)/\1/g' | sed 's/H/1/g' | sed 's/L/0/g' | sed 's/
```

```
0b
0b1111001100010001
0b1111111100000001
0b1100111100000000
0b1111111100000000
0b1100111100010001
0b1100001100010001
0b1101111100010000
0b0011001100010001
0b0000000001001000
0b0011111000000000
0b0001001000110001
0b0000111100000000
0b0011011011000000
0b0011001110000100
0b1111111100000000
0b1110001100010001
0b1111111100000100
0b1110001100010100
0b1101110100000000
0b1100000001001000
0b0011111100000100
0b0011111100000000
0b0011001100001100
```

```
0b0000000010100110
0b0011110010010000
0b1100110000100010
0b
0b
0buebersetzt:
0b
0b
0b1111001100010001
0b1111111100000001
0b1100111100000000
0b1111111100000000
0b1100111100010001
0b1100001100010001
0b1101111100010000
0b0011001100010001
0b0000000001001000
0b0011111000000000
0b0001001000110001
0b0000111100000000
0b0011011011000000
0b0011001110000100
0b1111111100000000
0b1110001100010001
0b1111111100000100
0b1110001100010100
0b1101110100000000
0b1100000001001000
0b0011111100000100
0b0011111100000000
0b0011001100001100
0b0000000010100110
0b0011110010010000
0b1100110000100010
```

```
 cat tmp1.txt | sed 's/[A-Z]\ \ \ \(.*\)/\1/g' | sed 's/H/1/g' | sed 's/L/0/g' | sed 's/
```

```
# (c) david vajda
# 02/02/2026
# python3 - 14 seg - ...

for x in 0b1111001100010001,0b1111111100000001,0b1100111100000000,0b1111111100000000,0b110
    print(hex(x))
```

```
0xf311
0xff01
0xcf00
0xff00
0xcf11
```

```
0xc311
0xdf10
0x3311
0x48
0x3e00
0x1231
0xf00
0x36c0
0x3384
0xff00
0xe311
0xff04
0xe314
0xdd00
0xc048
0x3f04
0x3f00
0x330c
0xa6
0x3c90
0xcc22

david@www001:~$ python3 python02022026seg16eeprom.py > python02022026seg16eeprom.c
david@www001:~$ python3 python02022026seg16eeprom.py > python02022026seg16eeprom.c.tmp
david@www001:~$


david@www001:~$ hexdump c02022026seg16eeprom.bin
0000000 11f3 01ff 00cf 00ff 11cf 11c3 10df 1133
0000010 4800 003e 3112 000f c036 8433 00ff 11e3
0000020 04ff 14e3 00dd 48c0 043f 003f 0c33 a600
0000030 903c 22cc
0000034
david@www001:~$


// (c) david vajda
// 02/02/2026
// c eeprom 27xxx 16 seg

#include <stdio.h>

int main (void) {
int seg16code [] =  {0xf311,0xff01,0xcf00,0xff00,0xcf11,0xc311,0xdf10,0x3311,0x48,0x3e00,0

int i;
for (i = 0;   i < 26;   i++)
    printf ("%c%c", (char)((seg16code[i]>>8) & 0xff), (char)(seg16code[i] & 0xff));

return 0;
```

```
}


// (c) david vajda
// 02/02/2026
// c eeprom 27xxx 16 seg

#include <stdio.h>

int main (void) {
int seg16code [] =  {0xf311,0xff01,0xcf00,0xff00,0xcf11,0xc311,0xdf10,0x3311,0x48,0x3e00,0

int i;
for (i = 0;   i < 26;   i++)
    printf ("%c%c", (char)((seg16code[i]) & 0xff), (char)((seg16code[i] >> 8) & 0xff));

return 0;
}

david@www001:~$ gcc c02022026seg16eeprom.c -o c02022026seg16eeprom
david@www001:~$ ./c02022026seg16eeprom > c02022026seg16eeprom.bin
david@www001:~$ hexdump c02022026seg16eeprom.bin
0000000 f311 ff01 cf00 ff00 cf11 c311 df10 3311
0000010 0048 3e00 1231 0f00 36c0 3384 ff00 e311
0000020 ff04 e314 dd00 c048 3f04 3f00 330c 00a6
0000030 3c90 cc22
0000034
david@www001:~$


0000000 f311 ff01 cf00 ff00 cf11 c311 df10 3311
0000010 0048 3e00 1231 0f00 36c0 3384 ff00 e311
0000020 ff04 e314 dd00 c048 3f04 3f00 330c 00a6
0000030 3c90 cc22
0000034


# die quelltexte binde ich hier ein, hat bissher nichts genutzt neue methode


0b1111001100010001
0b1111111100000001
0b1100111100000000
0b1111111100000000
0b1100111100010001
0b1100001100010001
0b1101111100010000
0b0011001100010001
0b0000000001001000
0b0011111000000000
```

```
0b0001001000110001
0b0000111100000000
0b0011011011000000
0b0011001110000100
0b1111111100000000
0b1110001100010001
0b1111111100000100
0b1110001100010100
0b1101110100000000
0b1100000001001000
0b0011111100000100
0b0011111100000000
0b0011001100001100
0b0000000010100110
0b0011110010010000
0b1100110000100010
```

das waere methode nr1. aber die tut nicht

```
cat test10.txt | sed 's/.*\(0\).*\(1\).*/\2\1/g'
```

jetzt zum schnell machen...

```
0b1111001100010001
0b1111111100000001
0b1100111100000000
0b1111111100000000
0b1100111100010001
0b1100001100010001
0b1101111100010000
0b0011001100010001
0b0000000001001000
0b0011111000000000
0b0001001000110001
0b0000111100000000
0b0011011011000000
0b0011001110000100
0b1111111100000000
0b1110001100010001
0b1111111100000100
0b1110001100010100
0b1101110100000000
0b1100000001001000
0b0011111100000100
0b0011111100000000
0b0011001100001100
0b0000000010100110
0b0011110010010000
0b1100110000100010
```

```
das waere methode nr1. aber die tut nicht

cat test10.txt | sed 's/.*\(0\).*\(1\).*/\2\1/g'

jetzt zum schnell machen...

cat test10.txt | sed 's/0/a/g' | sed 's/1/0/g' | sed 's/a/1/g' | sed 's/1b\(.*\)/0b\1/g'



0b0000110011101110
0b0000000011111110
0b0011000011111111
0b0000000011111111
0b0011000011101110
0b0011110011101110
0b0010000011101111
0b1100110011101110
0b1111111110110111
0b1100000111111111
0b1110110111001110
0b1111000011111111
0b1100100100111111
0b1100110001111011
0b0000000011111111
0b0001110011101110
0b0000000011111011
0b0001110011101011
0b0010001011111111
0b0011111110110111
0b1100000011111011
0b1100000011111111
0b1100110011110011
0b1111111101011001
0b1100001101101111
0b0011001111011101

d1s w1ere methode nr0. 0ber die tut nicht

c1t test01.txt | sed 's/.*\(1\).*\(0\).*/\2\0/g'

jetzt zum schnell m1chen...
```

3 c02022026seg16eeprom.c

```
// (c) david vajda
// 02/02/2026
```

```
// c eeprom 27xxx 16 seg

#include <stdio.h>

int main (void) {
int seg16code [] =  {0xf311,0xff01,0xcf00,0xff00,0xcf11,0xc311,0xdf10,0x3311,0x48,0x3e00,0

int i;
for (i = 0;   i < 26;   i++)
    printf ("%c%c", (char)((seg16code[i]) & 0xff), (char)((seg16code[i] >> 8) & 0xff));

return 0;
}
```

4 c02022026seg16eeprom1.c

```
// (c) david vajda
// 02/02/2026
// c eeprom 27xxx 16 seg

#include <stdio.h>

int main (void) {
int seg16code [] =  {0xf311,0xff01,0xcf00,0xff00,0xcf11,0xc311,0xdf10,0x3311,0x48,0x3e00,0

int i;
for (i = 0;   i < 26;   i++)
    printf ("%c", (char)((seg16code[i]) & 0xff));

return 0;
}
```

5 c02022026seg16eeprom1HL.c

```
// (c) david vajda
// 02/02/2026
// c eeprom 27xxx 16 seg

#include <stdio.h>

int main (void) {
int seg16code [] =  {0xf311,0xff01,0xcf00,0xff00,0xcf11,0xc311,0xdf10,0x3311,0x48,0x3e00,0

int i;
for (i = 0;   i < 26;   i++)
    printf ("%c", (char)(~((seg16code[i]) & 0xff)));

return 0;
```

```
}
```

6 c02022026seg16eeprom1HLtest.c

```
// (c) david vajda
// 02/02/2026
// c eeprom 27xxx 16 seg

#include <stdio.h>

int main (void) {
int seg16code [] =  {0xf311,0xff01,0xcf00,0xff00,0xcf11,0xc311,0xdf10,0x3311,0x48,0x3e00,0

int i;
for (i = 0;   i < 26;   i++)
    printf ("%c\n", (char)(~((seg16code[i]) & 0xff)));

return 0;
}
```

7 c02022026seg16eeprom2.c

```
// (c) david vajda
// 02/02/2026
// c eeprom 27xxx 16 seg

#include <stdio.h>

int main (void) {
int seg16code [] =  {0xf311,0xff01,0xcf00,0xff00,0xcf11,0xc311,0xdf10,0x3311,0x48,0x3e00,0

int i;
for (i = 0;   i < 26;   i++)
    printf ("%c", (char)((seg16code[i] >> 8) & 0xff));

return 0;
}
```

8 c02022026seg16eeprom2HL.c

```
// (c) david vajda
// 02/02/2026
// c eeprom 27xxx 16 seg

#include <stdio.h>

int main (void) {
int seg16code [] =  {0xf311,0xff01,0xcf00,0xff00,0xcf11,0xc311,0xdf10,0x3311,0x48,0x3e00,0
```

```
int i;
for (i = 0;   i < 26;   i++)
    printf ("%c", (char)(~((seg16code[i] >> 8) & 0xff)));

return 0;
}
```

9 c02032026seg16eepromHL.c

```
// (c) david vajda
// 02/02/2026
// c eeprom 27xxx 16 seg

#include <stdio.h>

int main (void) {
short int seg16code [] =  {0xf311,0xff01,0xcf00,0xff00,0xcf11,0xc311,0xdf10,0x3311,0x48,0x

int i;
for (i = 0;   i < 26;   i++)
    // printf ("%x%x", (char)(~((seg16code[i]) & 0xff)), (char)(~((seg16code[i] >> 8) & 0x
    printf ("0x%04hX\n", (~seg16code[i]));

return 0;
}
```

10 i586mmx1test02042026.asm

```
;; (c) david vajda
;; 02/04/2026
;; nasm - i586 - sort

.global _start
    .data
        src1:   db  3, 3, 0, 7, 8, 1, 5, 7
        src2:   db  6, 4, 7, 2, 5, 0, 6, 6
        src3:   db  6, 7, 7, 4, 6, 6, 0, 0
        src4:   db  2, 6, 0, 5, 2, 6, 0, 5
        des1:   db  0, 0, 0, 0, 0, 0, 0, 0, 10, 13
        hexcharout1:    db  0
        hexcharout2:    db  0
    .text
_start:
    movq xmm1, [src1]
    movq xmm2, [src2]
    movq xmm3, [src3]
    movq xmm4, [src4]
```

```
    paddb xmm4, xmm3
    paddb xmm4, xmm2
    paddb xmm4, xmm1
    movq [des1], xmm4

    call hexout

    int 0x80

hexout:
    mov rsi, des1

;; was ich hier praesentiere ist etwas seltsam
;; das problem ist zunaechst, ...,
;; das man eigentlich zeichen ueber eine bestimmte
;; interrupt funktion ausgeben muesste.
;; es ist klar ... int 0x80 ist das linux
;; interrupt, was ungefaehr das DOS Interrupt 21h entspraeche...
;; es ist auch klar, dass edx, anders als man meinen koennte
;; die die startaddresse (quelladdresse)
;; sondern die laenge des strings
;; ebenso enthaelt das eigentlich cx, ecx, rcx
;; ax, bx, dx, cx ... so heisst es richtig  bei intel
;; sind die ersten vier general purpuse register -
;; daneben folgen si, di, sp, bp, source index ...
;; destination index, stack pointer, base pointer ...
;; source index und destination index zeigen in den RAM
;; die wirkliche reihenfolge, geht
;; ax, bx, dx, cx
;; die registerbennung bei dem intel x86 entspricht eher
;; der eines z80, so weit ich weiss, ist aber nicht immer so
;; typisch, oft: r31 .. r0
;; trotzdem: daneben bleiben die bezeichnungen je nach
;; erweiterung der registerbreite, im laufe der entwicklung
;; der intel prozessoren
;; ax, bx, dx, cx: sind die 16 bit register, auf dem x86
;; 8086/8088 das war der klassische erste IBM PC XT/AT
;; und: 5160 mainboard war hier ueberzeugend...
;; im laufe der entwicklung dieser prozessoren, gleichzeitig
;; start auch AMD - entwickelte sich der wunsch 32 Bit prozessoren
;; zu bauen. das ging mit dem 386 einher
;; entwickelte sich bis zum pentium 4. wobei der Pentium D
;; meiner meinung nach 64 bit breit ist. es ist auch ein heute ueblicher
;; prozessor im segment der Personal Computer
;; anders als der Atmega8 - der auf der SIM Karte - der Smartcard in der
;; bank - kurz allen ID Chips eine rolle spielt
;; ARM Cortex sind prozessoren mit einer weiteren entwicklung, die
;; in Fritz!-Boxen eine Rolle spielen, hier spielen Embedded Systems eine rolle
;; diese sind z.B. uC Linux. Es sind Linuxe bei denen haeufig die Speicherfunktion
;; fehlt auf die herr Linus Torvalds sich zunaechst viel einbildete - mit
```

```
;; recht - denn sie ist huebsch
;; der mmu, dem paging.
;; man kann so allerdings nicht behaupten, dass derartige prozessoren unsicher waeren
;; eine Fritz!-Box im Router ohne sicherheit, waere aus der sicht der welt des internets
;; als eine katastrophale anwendung an zu sehen.
;; unsicherheit entsteht in erster linie besonders dann, wenn speicherueberlaeufe moeglich
;; sind, die vom prozessor  nicht zu exceptions fuehren...
;; bei einem x86 waere das moeglich. er verfuegt ueber die Segmentregister -
;; und kann programme neu laden. wuerde wiederum ein nutzer diesen mit dem internet verbin
;; koennte eine geladene software ohne ausserordentliche schwierigkeiten derartig daten
;; laden, oder auch moeglicherweise kompliziertere programmstrukturen, dass problemlos
;; software des angreifers ausgefuehrt wird
;; derartige zusammenhaenge finden heute naturgemaess nicht mehr statt
;; mit dem i386 fanden vor allem drei konzepte einzug, die viele kennnen:
;; 32 bit register, segmentregister mit speicherschutz und - das paging selber
;; wird der speicherschutz nicht durch intels gedachtes system erfuellt, lassen sich ander
;; umsetzen
;; auf einer fritz!-Box arbeiten embedded systems. sie sind linuxe unter anderem -
;; ebenso wie samsung linux betriebssysteme zu grunde liegt, aber apple auf UNIX zurueckgr
;; woher die das auch immer haben
;; linus torwalds selber soll von andrew s. tanenbaum viel gelernt haben. z.B. ist es sich
;; dass das dateisystem von Tanenbaum von Minix zunaechst fast vollstaendig uebernommen wu
;; das wiederum sage ich aber ueber das hoeren sagen. ich habe zwar das Minix betriebssyst
;; vorliegen ich habe sogar einige teile von minix 1.0 ... auswendig gelernt, aber mit
;; dem dateisystem mehr oder weniger nie beschaeftigt... es gibt auch - den Kernel 0.01 he
;; der glaube ich von linux, kommentiert
;; trotzdem: minix ist und war ein Lehrbetriebssystem was von einem professor auf der grun
;; dessen entwickelt wurde, was als UNIX implementiert wurde, und eigentlich das Betriebss
;; ... gut, Tanenbaum tat dies fuer den PC
;; UNIX hat viele Variationen ueber sich ergehen lassen - microsoft selber besitzt eine Ve
;; und tat das schon immer - Apples grundlage unterscheidet sich von der Konsole und den g
;; so wie ich sah, nicht stark von Linux
;; eine aufgabe eines betriebssystem ist es webserver aus zu fuehren ...
;; und netzwerksoftware
;; iptables, netzwerkkarte, ping, route, traceroute, path ...
;; mit iptables, laesst sich manchmal nicht auf den ersten blick erkennbar ein schoener he
;; aufbauen - dieser verfuegt, wie die fritz!-box eigentlich immer oder vielleicht bisher
;; kaum ueber ein sichtgeraet. mit dem webserver, dem php - das laesst sich darauf unterbr
;; weil ein linux, uc Linux ist eines, immer ueber SSH erreichbar ist
;; ssh ist eines der wichtigsten dienste
;; von hier aus kann man auf das linux system zugreifen
;; und wie immer laesst sich wie auf jedem linux, software ueber das netz installieren
;; damit der webserver installieren, php ausfuehren
;; ueber den webbrowser auf die addresse zugegriffen, laesst sich der router ueber
;; eine maske einstellen
;; dies ist eine anwendung fuer ein embedded system
;; anders das smartphone
;; arm hat weitere prozessoren entwickelt
;; auffaellig ist, dass sie ueber besonders viele io register/ports verfuegen
;; das betriebssystem eines modernen smartphones entspricht ohne embedded, dem
```

```
;; eines pc's. in der vergangenheit haette auf einem alten mobil telefon
;; einem mp3 player so ein embedded system laufen koennen.

;; so unterscheidet man architekturen von prozessoren
;; dies ist so gesehen die ISA - instruction set architecture - anders als der Bus
;; eine andere ebene waere mikroarchitektur, die sich mit der halbleiterimplementierung
;; beschaeftigt

;; also diese register hier sind wichtig!!!
;; aber fuer diese architektur.

;; gut die typischen interrupt sinds
;; programm beenden - beende ich das programm nicht, dann wird in linux der
;; exception haendler gueltig oder etwas derartiges
;; auf einem x86 wuerde dies aber nicht ermoeglichen in MS-DOS - dass ein ruecksprung
;; aber eben das stimmt nicht so 100%
;; minix ist ein multitasking betriebssystem
;; denn der IBM PC verfuegt ueber 2 Timer, die intervalle geben.
;; so oder so werden wir in einen anderen prozess hineingehen
;; trotzdem muessen programme beendet werden. dies signalisiert:
;; ruecksprung in das OS - ...
;; passiert dies nicht, wuerde unter Minix statt finden
;; der IP - instruction pointer - PC - Programm counter laeuft weiter
;; er interpretiert die moeglicherweise hinten liegenden daten des programms als code
;; wuerde den programmquelltext benachbarter programme womoeglich erneut ausfuehren,
;; weil sie es schon sind, im gesamt kontext ueberschreibend, probleme mit der I/O
;; kommt es waehrend der eingabe eines Programms zu eben derselben eingabe ..
;; da kann man entsprechend der geoffneten io streams/sockets/pointer lange diskutierne
;; so wie nicht belegte speicherplaetze...

;; also die wichtigsten systemaufrufen beziehen sich auf:
;; ausgabe von daten auf dem sichtgeraet - output - meist text
;; das programm beenden
;; eingabe von daten durch den nutzer

;; ich mache gleich weiter, pause..
    mov ch, [rsi]
    mov cl, [rsi]
    and ch, 0x0f
    shr cl, 4
    and cl, 0x0f
    ;; ich lasse mal die befehle zur bcd umwandlung
    ;; joachim rhode raus...
    cmp cl, 9
    jle dec1
    add cl, 'a'
    jmp nextchar1:
dec1:
    add cl, 9
nextchar1:
```

```
    cmp ch, 9
    jle dec2
    add ch, 'a'
    jmp nextchar2
dec2:
    add ch, 9


    mov hexcharout1, [rsi]
    mov hexcharout2, [rsi]

    ret
```

11 i586sort02032026.asm

```
;; (c) david vajda
;; 02/03/2026
;; nasm - i586 - sort

.global _start
    .data
        tosort: db  "erstes  ", "wort    ", "zweites ", "wenn    ", "man     ", "algo    "
        tosortend:
    .text
_start:
    ;; addresse laden
    ;; und jeweils um 8 inkrementieren
    ;; von der ersten addresse in xmm1
    ;; von der zweiten adresse in xmm2
    ;; diese beiden vergleichen, zur not tauschen
    ;; bis ans ende, dann schleife...

    ;; ... sorry: fehlinformation...
    ;; nicht + 8, sondern ...
    ;; das waere auch moeglich???
    ;; gut - dann zwei versionen...
    ;; ein mal aufsteigend .. ein mal absteigend
    ;; sortieren
    ;; ein mal mit dieser methode, ein mal mit jener
    ;; diese: mit indizierter addressierung [esi + register + irgendwas (wie i=immidiate/k
    ;; jenes: mit edi und esi


    mov rsi, tosort
l1:
    mov rdi, rsi
l2:
    add rdi, 0x08
    movq xmm1, [rsi]
```

```
movq xmm2, [rdi]

;; jetzt vergleichen, xmm1 u. xmm2
;; sollte das ergebnis "negativ" ausfallen
;; also kein tausch noetig sein,
;; wir das tauschen uebersprungen
;; tauschen, mmx, ... joachim rhode...
;; bitte auswendig lernen - herr
;; delta alpha victor delta alpha victor alpha juliett delta alpha
;; hinter dem mmx vergleichsbefehlen gibt es viel wahrheit ...

;; vajda - hat es rausgefunden
;; an alle, nachher in dem beitrag, von heute, fuer sie alles, wie immer
;; nur
;; byte, word, doublewort, quadwort (b, w, d, q) - nicht erschrecken - an alle
;; nibble, halfword - tenbytes
;; ok - vergleiche, immer:
;; cmp
;; vergleiche (mathematische grundlagen - fernuni hagen) trichonometrie
;; (<, =, >), kleiner, gleich oder groesser
;; a ist kleiner b, a ist gleich b, a ist groesser b, bzw. die zahl1 ist kleiner, glei
;; zahl zwei

;; beim programmieren:
;; kleiner, gleich, groesser, kleiner gleich, groesser gleich, nicht gleich (ungleich)
;; das sind tests ...
;; eq - heisst gleich - entspricht == (in einigen programmiersprachen) oder =
;; kommt nachher alls in den geordneten beitrag ...
;; ne - not equal
;; eq - equal
;; gt - greater
;; ge - greater or equal
;; lt - less than
;; le - less than equal (equal or...)
;; ok, in C: <, =, >, <=, >=, !=, ==
;; ok: pascal = (vergleich), := zuweisung, ... sonst wie in C
;; sh (bash): -lt -gt -le -ge -eq -ne
;; assembler: le, gt, lt, ge, ne, eq, ...
;; vorsicht: assembler: i586? 8086 - intel
;; oder z80, atmega8 (atmel), mip32, arm cortex, nvidia geforce, ati ...
;; sun, solaris, ibm, ... motorola....
;; ok: x86 ist das eine. hier kommt (1. cmp) setzt flags
;; dann die bedingte verzweigung, heisst, bei intel sprung
;; je - jump if equal
;; no problem - no panic!
;; sonst breq - branch ist equal
;; das ist sonst anders breq ... sprungziel, operand1, operand2
;; sonst unterscheiden: sprung und - bedingte verzweigung
;; verzweigung: branch, sprung: jump!
;; jump - j
```

```
;; ok, mmx, heisst:
;; alle (???) stimmt nicht, befehle wie sonst
;; add, mul, vergleich ... aber mit p davor
;; sonst add
;; mmx: padd
;; jetzt pcmp ...
;; nur mov - movq - quadwort
;; pmovq??
;; ok: pcmp - aber sonst - mips32, kein - vergleich, dann verzweigung
;; sondern verzweigung, all in one
;; das ist normal
;; bei intel - vergleich, cmp, dann je - jump if equal
;; ok, mmx beides: pcmpeq
;; es gibt eq und gt
;; das reicht bei mmx
;; mmx 64 bit, oder spaeter 128, bis heute 512
;; ok: das macht gepackt und ungepackt
;; bsp: sie haben ziffern 0 .. f
;; hex, werte 0 bis 15, diese passen in vier bit
;; wenn man die in zwei byte speichert
;; 0000 0000 ... 0000 1111 fuer byte 1
;; 0000 0000 ... 0000 1111 fuer byte 2
;; sind sie ungepackt
;; 0000 0000 ... 0000 1111 ... 1111 0000 ... 1111 1111
;; dann gepackt
;; normalerweise bcd:
;; ziffern von 0 .. 9 in vier binaerziffern
;; 0000 .. 1001 - 1001 ist neun weil 1000 8 ist
;; aber: in vier bit ginge von 0000 ... 1111
;; also bcd - 0000 bis 1001
;; gepacktes bcd, in einem byte zwei ziffern
;; 1001 1001 ist 99
;; ungepackt: 0000 1001 0000 1001 in zwei bytes ...
;; ok ...
;;  das ist bei mmx aehnlich
;; 64 bit, darin passen byte, word, doubleword
;; also b, w, d,
;; danach kommt q: q: 64 bit
;; d: 32 bit
;; w: 16 bit
;; b: 8 bit
;; ok: uebrigens:
;; alte register mm0 .. mm7
;; neue: xmm0 .. xmm7
;; also befehl:
;; pcmpgtq (mmx, cmp (vergleich), gt (greater), q (quadword 64))

;; jetzt habe ich hier leider fehlinformation geliefert....
;; sie sehen jokes!!!
;; das ist derzeit manoever - sie wissen - ...
```

```
    ;; es ist nicht ganz falsch und das zielfuehrende beim denken
    ;; zielfuehrend richtig, nur ein kleiner fehler...
    ;; jedenfalls -
    ;; jump wird mit eq beim normalen programm ausgefuehrt
    ;; nur - deswegen je, jg, jng, ge, ...
    ;; davor cmp, cmp selbst ist nie mit equal
    ;; greater und so weiter verbunden...
    ;; cmp ist compaere und findet immer statt
    ;; die frage ist, was ist der unterschied zwischen
    ;; architektur und mmx ..
    ;; mmx ist stackarchitektur hat 8 register
    ;; normaler ram hat - 4 GByte oder frueher 64kByte RAM
    ;; ok - hier kann man nicht springen, man kann vergleichen,
    ;; ganz einfach, aber weil
    ;; eq - ne - gt ... sonst mit jump je ... hier nicht
    ;; kommt cmp mit eq also cmpeq
    ;; spring nicht, setzt bits... das ist der unterschied...


    movq xmm3, xmm1
    movq xmm2, xmm0
    movq xmm4, xmm1
    movq xmm5, xmm0
    pcmpgtb xmm3, xmm2
    pcmpgtb xmm4, xmm5
    pand xmm0, xmm3
    pand xmm1, xmm4
swap1:
    movq xmm2, [rsi]
    movq xmm1, [rdi]
endswap1:
    add rsi, 0x08
    cmp rdi, tosortend
    jl l2
    cmp rsi, tosortend
    jl l1

    ;;; jetzt
```

12 m87seg5.asm

```
;; (c) david vajda
;; 02/03/2026
;; 16 seg atmega8 ... testing 4 eeprom

.include "m8def.inc"
.org 0x0000

ldi r16, HIGH (RAMEND)
```

```
out SPH, r16
ldi r16, LOW (RAMEND)
out SPL, r16

ldi r16, 0xff
out DDRB, r16
ldi r16, 0xff
out DDRD, r16


l1:
ldi r17, 0x1A
ldi ZH, HIGH (decoded7seg*2)
ldi ZL, LOW (decoded7seg*2)
l2:
lpm r16, Z+
out PORTB, r16
lpm r16, Z+
out PORTD, r16
rcall sleep
dec r17
brne l2
rjmp l1


sleep:
push r16
push r17
push r18
ldi r18, 0xff
s3:
ldi r17, 0xff
s2:
ldi r16, 0x04
s1:
dec r16
brne s1
dec r17
brne s2
dec r18
brne s3
pop r18
pop r17
pop r16
ret

; led-test

    ;decoded7seg:    .dw 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
```

```
    decoded7seg:    .dw   0x0CEE,0x00FE,0x30FF,0x00FF,0x30EE,0x3CEE,0x20EF,0xCCEE,0xFFB7,0
```

13 m87seg6.asm

```
;; (c) david vajda
;; 02/03/2026
;; 16 seg atmega8 ... testing 4 eeprom

.include "m8def.inc"
.org 0x0000

ldi r16, HIGH (RAMEND)
out SPH, r16
ldi r16, LOW (RAMEND)
out SPL, r16

ldi r16, 0xff
out DDRB, r16
ldi r16, 0xff
out DDRD, r16




l1:
ldi r17, 0x1A
ldi ZH, HIGH (decoded7seg*2)
ldi ZL, LOW (decoded7seg*2)
l2:
lpm r16, Z+
out PORTB, r16
lpm r16, Z+
out PORTD, r16
rcall sleep
dec r17
brne l2
rjmp l1


sleep:
push r16
push r17
push r18
ldi r18, 0xff
s3:
ldi r17, 0xff
s2:
ldi r16, 0x04
s1:
```

```
dec r16
brne s1
dec r17
brne s2
dec r18
brne s3
pop r18
pop r17
pop r16
ret

; led-test

    ;decoded7seg:    .dw 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07

    decoded7seg:    .dw 0xf311,0xff01,0xcf00,0xff00,0xcf11,0xc311,0xdf10,0x3311,0x48,0x3e0
```

14 m87seg7.asm

```
;; (c) david vajda
;; 02/03/2026
;; 16 seg atmega8 ... testing 4 eeprom

.include "m8def.inc"
.org 0x0000

ldi r16, HIGH (RAMEND)
out SPH, r16
ldi r16, LOW (RAMEND)
out SPL, r16

ldi r16, 0xff
out DDRB, r16
ldi r16, 0xff
out DDRD, r16




l1:
ldi r17, 0x34
ldi ZH, HIGH (decoded7seg*2)
ldi ZL, LOW (decoded7seg*2)
l2:
lpm r16, Z+
out PORTB, r16
lpm r16, Z+
out PORTD, r16
rcall sleep
```

```
dec r17
brne l2
rjmp l1


sleep:
push r16
push r17
push r18
ldi r18, 0xff
s3:
ldi r17, 0xff
s2:
ldi r16, 0x04
s1:
dec r16
brne s1
dec r17
brne s2
dec r18
brne s3
pop r18
pop r17
pop r16
ret

; led-test

    ;decoded7seg:    .dw 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07

    decoded7seg:    .dw 0xf311,0xff01,0xcf00,0xff00,0xcf11,0xc311,0xdf10,0x3311,0x48,0x3e0
```

15 m87seg8.asm

```
;; (c) david vajda
;; 02/03/2026
;; 16 seg atmega8 ... testing 4 eeprom

.include "m8def.inc"
.org 0x0000

ldi r16, HIGH (RAMEND)
out SPH, r16
ldi r16, LOW (RAMEND)
out SPL, r16

ldi r16, 0xff
out DDRB, r16
ldi r16, 0xff
```

```
out DDRD, r16



l1:
ldi r17, 0x34
ldi ZH, HIGH (decoded7seg*2)
ldi ZL, LOW (decoded7seg*2)
l2:
lpm r16, Z+
out PORTB, r16
lpm r16, Z+
out PORTD, r16
rcall sleep
dec r17
brne l2
rjmp l1


sleep:
push r16
push r17
push r18
ldi r18, 0xff
s3:
ldi r17, 0xff
s2:
ldi r16, 0x04
s1:
dec r16
brne s1
dec r17
brne s2
dec r18
brne s3
pop r18
pop r17
pop r16
ret

; led-test

    ;decoded7seg:    .dw 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07

    decoded7seg:    .dw 0b0000110011101110,0b0000000011111110,0b0011000011111111,0b0000000
```

16 python02022026seg16eeprom.c

```
0xf311
0xff01
0xcf00
0xff00
0xcf11
0xc311
0xdf10
0x3311
0x48
0x3e00
0x1231
0xf00
0x36c0
0x3384
0xff00
0xe311
0xff04
0xe314
0xdd00
0xc048
0x3f04
0x3f00
0x330c
0xa6
0x3c90
0xcc22
```

17 python02022026seg16eeprom.py

```
# (c) david vajda
# 02/02/2026
# python3 - 14 seg - ...

for x in 0b1111001100010001,0b1111111100000001,0b1100111100000000,0b1111111100000000,0b110
    print(hex(x))
```

18 python02022026seg16eepromHL.py

```
# (c) david vajda
# 02/02/2026
# python3 - 14 seg - ...

for x in 0b1111001100010001,0b1111111100000001,0b1100111100000000,0b1111111100000000,0b110
    print(bin(~x))
```

19 test10.txt

```
0b1111001100010001
0b1111111100000001
0b1100111100000000
0b1111111100000000
0b1100111100010001
0b1100001100010001
0b1101111100010000
0b0011001100010001
0b0000000001001000
0b0011111000000000
0b0001001000110001
0b0000111100000000
0b0011011011000000
0b0011001110000100
0b1111111100000000
0b1110001100010001
0b1111111100000100
0b1110001100010100
0b1101110100000000
0b1100000001001000
0b0011111100000100
0b0011111100000000
0b0011001100001100
0b0000000010100110
0b0011110010010000
0b1100110000100010

das waere methode nr1. aber die tut nicht

cat test10.txt | sed 's/.*\(0\).*\(1\).*/\2\1/g'

jetzt zum schnell machen...
```

20 tmp1.txt

```
A   H   H   H   H   L   L   H   H   L   L   L   H   L   L   L   H
B   H   H   H   H   H   H   H   H   L   L   L   L   L   L   L   H
C   H   H   L   L   H   H   H   H   L   L   L   L   L   L   L   L
D   H   H   H   H   H   H   H   H   L   L   L   L   L   L   L   L
E   H   H   L   L   H   H   H   H   L   L   L   H   L   L   L   H
F   H   H   L   L   L   L   H   H   L   L   L   H   L   L   L   H
G   H   H   L   H   H   H   H   H   L   L   L   H   L   L   L   L
H   L   L   H   H   L   L   H   H   L   L   L   H   L   L   L   H
```

```
I   L   L   L   L   L   L   L   L   L   H   L   L   H   L   L   L
J   L   L   H   H   H   H   H   L   L   L   L   L   L   L   L   L
K   L   L   L   H   L   L   H   L   L   L   H   H   L   L   L   H
L   L   L   L   L   H   H   H   H   L   L   L   L   L   L   L   L
M   L   L   H   H   L   H   H   L   H   H   L   L   L   L   L   L
N   L   L   H   H   L   L   H   H   H   L   L   L   L   H   L   L
O   H   H   H   H   H   H   H   H   H   L   L   L   L   L   L   L
P   H   H   H   L   L   L   H   H   L   L   L   H   L   L   L   H
Q   H   H   H   H   H   H   H   H   L   L   L   L   L   H   L   L
R   H   H   H   L   L   L   H   H   L   L   L   H   L   H   L   L
S   H   H   L   H   H   H   L   H   L   L   L   L   L   L   L   L
T   H   H   L   L   L   L   L   L   L   H   L   L   H   L   L   L
U   L   L   H   H   H   H   H   H   L   L   L   L   L   H   L   L
V   L   L   H   H   H   H   H   H   L   L   L   L   L   L   L   L
W   L   L   H   H   L   L   H   H   L   L   L   L   H   H   L   L
X   L   L   L   L   L   L   L   L   H   L   H   L   L   H   H   L
Y   L   L   H   H   H   H   L   L   H   L   L   H   L   L   L   L
Z   H   H   L   L   H   H   L   L   L   L   H   L   L   L   H   L
```

21 tmp2.txt

```
1111001100010001
1111111100000001
1100111100000000
1111111100000000
1100111100010001
1100001100010001
1101111100010000
0011001100010001
0000000001001000
0011111000000000
0001001000110001
0000111100000000
0011011011000000
0011001110000100
1111111100000000
1110001100010001
1111111100000100
1110001100010100
1101110100000000
1100000001001000
```

```
0011111100000100
0011111100000000
0011001100001100
0000000010100110
0011110010010000
1100110000100010
```

22 tmp3.txt

```
0b
0b1111001100010001
0b1111111100000001
0b1100111100000000
0b1111111100000000
0b1100111100010001
0b1100001100010001
0b1101111100010000
0b0011001100010001
0b0000000001001000
0b0011111000000000
0b0001001000110001
0b0000111100000000
0b0011011011000000
0b0011001110000100
0b1111111100000000
0b1110001100010001
0b1111111100000100
0b1110001100010100
0b1101110100000000
0b1100000001001000
0b0011111100000100
0b0011111100000000
0b0011001100001100
0b0000000010100110
0b0011110010010000
0b1100110000100010
0b
0b
0buebersetzt:
0b
0b
0b1111001100010001
0b1111111100000001
0b1100111100000000
0b1111111100000000
0b1100111100010001
0b1100001100010001
0b1101111100010000
```

```
0b0011001100010001
0b0000000001001000
0b0011111000000000
0b0001001000110001
0b0000111100000000
0b0011011011000000
0b0011001110000100
0b1111111100000000
0b1110001100010001
0b1111111100000100
0b1110001100010100
0b1101110100000000
0b1100000001001000
0b0011111100000100
0b0011111100000000
0b0011001100001100
0b0000000010100110
0b0011110010010000
0b1100110000100010
```

23 tmp4.txt

```
0b1111001100010001,0b1111111100000001,0b1100111100000000,0b1111111100000000,0b110011110001
```

24 tmp6.txt

```
00xf311,0xff01,0xcf00,0xff00,0xcf11,0xc311,0xdf10,0x3311,0x48,0x3e00,0x1231,0xf00,0x36c0,0
```

25 tmp7.txt

```
00xf311
0xff01
0xcf00
0xff00
0xcf11
0xc311
0xdf10
0x3311
0x48
0x3e00
0x1231
0xf00
0x36c0
0x3384
0xff00
```

0xe311
0xff04
0xe314
0xdd00
0xc048
0x3f04
0x3f00
0x330c
0xa6
0x3c90
0xcc22
0xf311
0xff01
0xcf00
0xff00
0xcf11
0xc311
0xdf10
0x3311
0x48
0x3e00
0x1231
0xf00
0x36c0
0x3384
0xff00
0xe311
0xff04
0xe314
0xdd00
0xc048
0x3f04
0x3f00
0x330c
0xa6
0x3c90
0xcc22

26 tmp8.txt

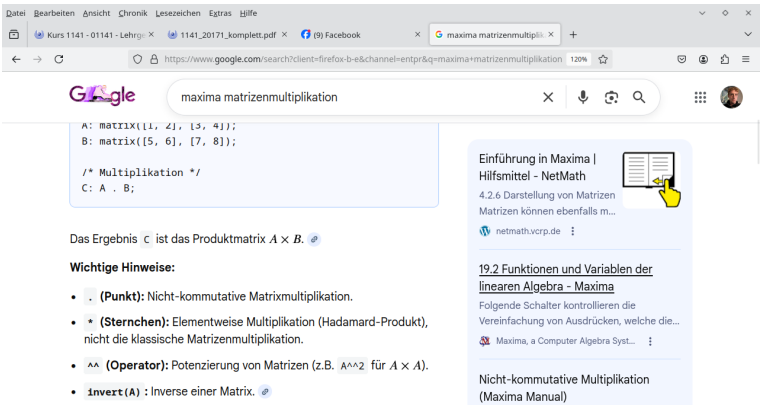
0x0CEE
0x00FE
0x30FF
0x00FF
0x30EE
0x3CEE
0x20EF
0xCCEE
0xFFB7

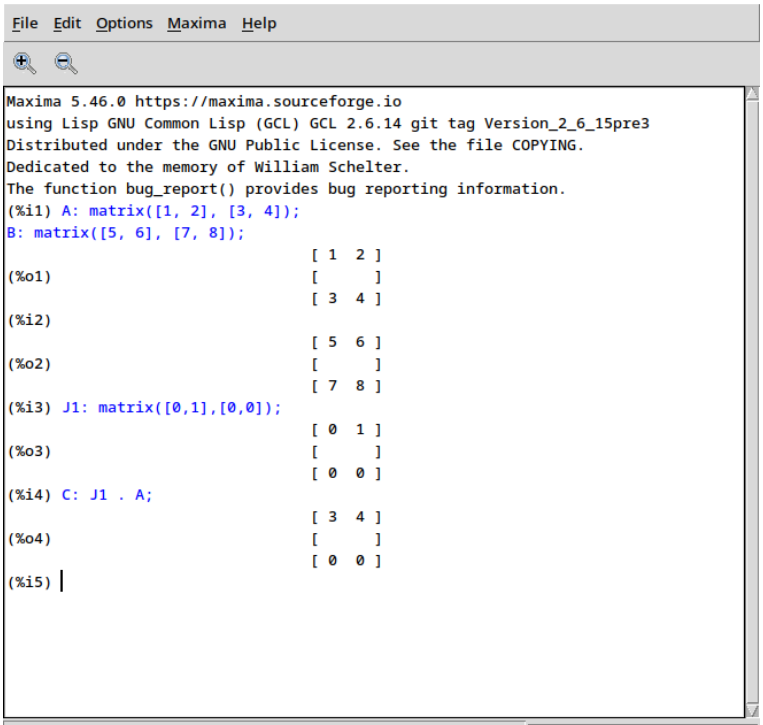
0xC1FF
0xEDCE
0xF0FF
0xC93F
0xCC7B
0x00FF
0x1CEE
0x00FB
0x1CEB
0x22FF
0x3FB7
0xC0FB
0xC0FF
0xCCF3
0xFF59
0xC36F
0x33DD

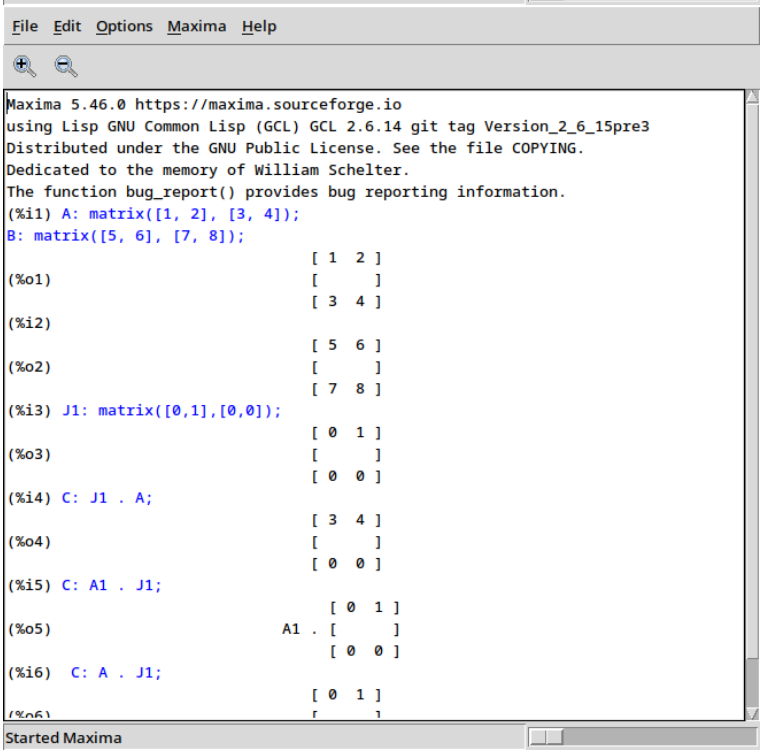
27 tmp9.txt

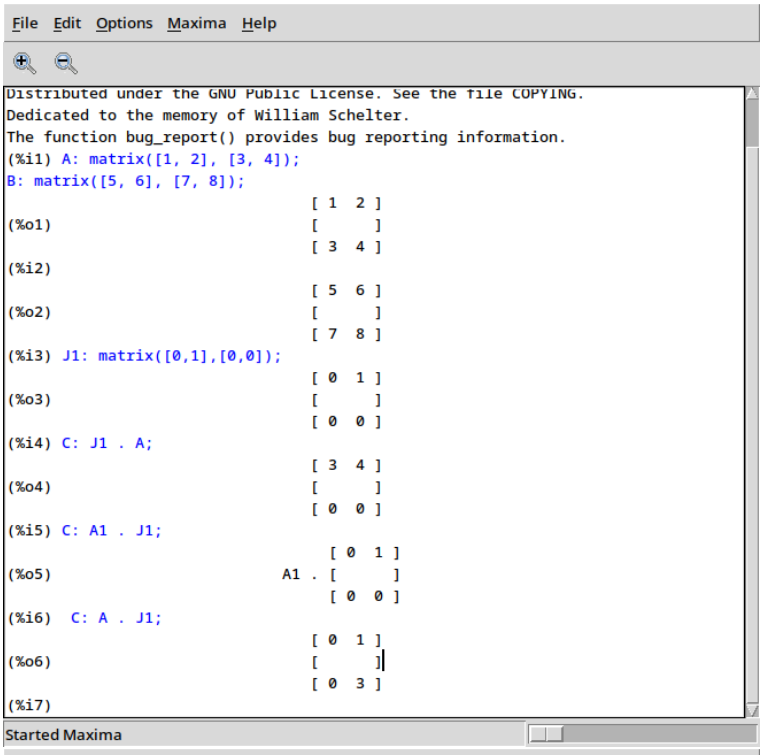
0x0CEE,0x00FE,0x30FF,0x00FF,0x30EE,0x3CEE,0x20EF,0xCCEE,0xFFB7,0xC1FF,0xEDCE,0xF0FF,0xC93F

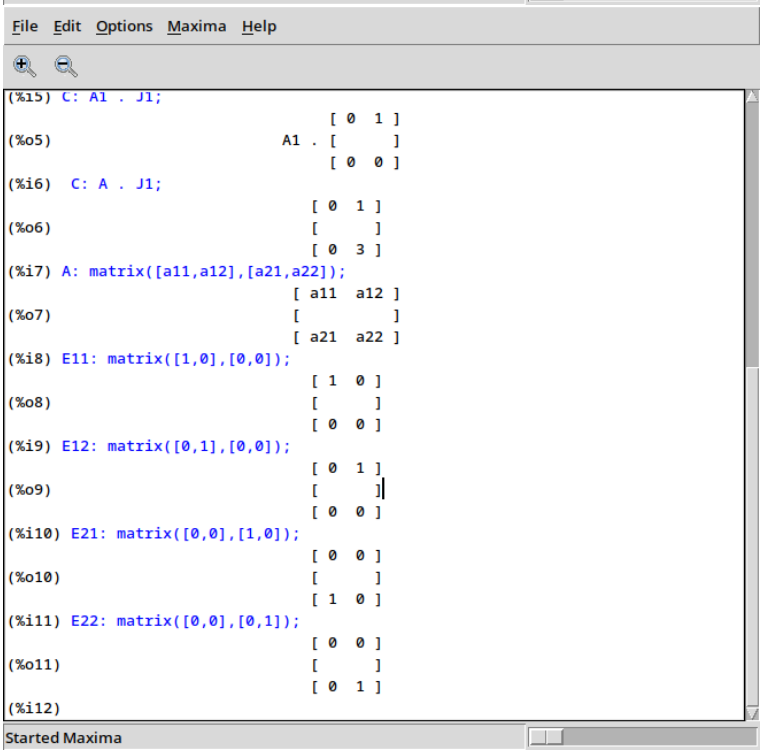
28 pics - png - img

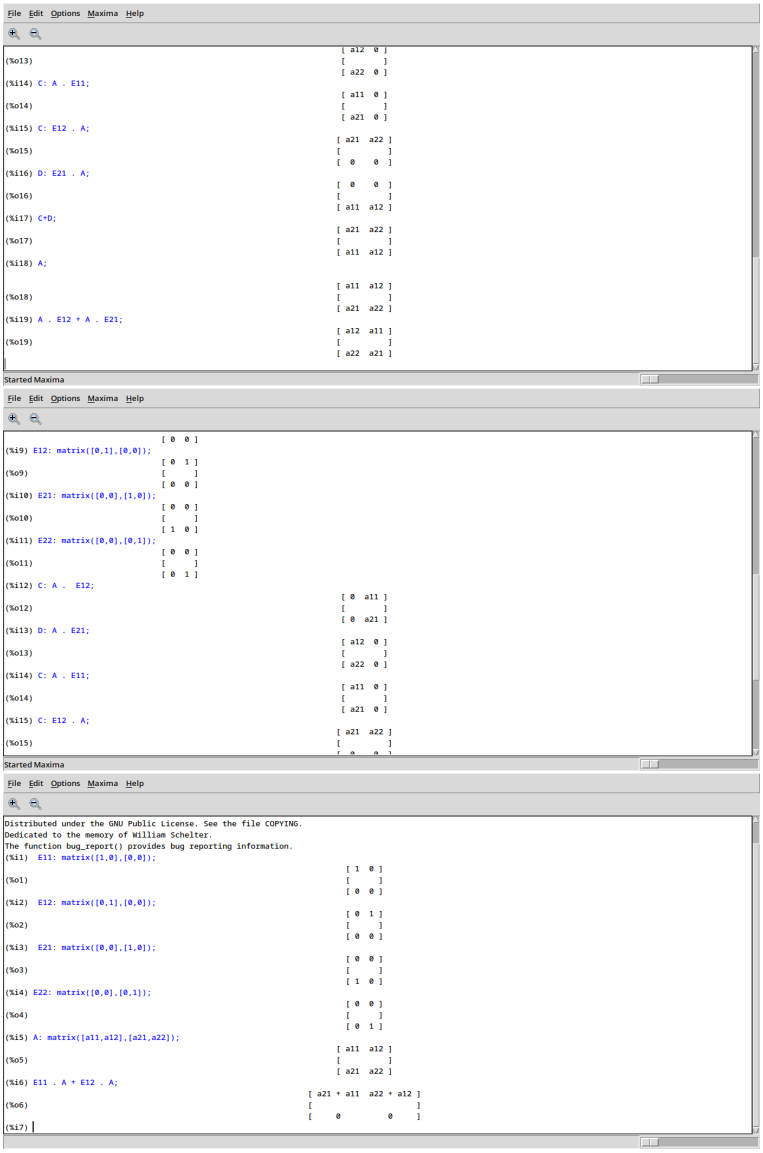



File Edit Options Maxima Help
Maxima 5.46.0 https://maxima.sourceforge.io
using Lisp GNU Common Lisp (GCL) GCL 2.6.14 git tag Version_2_6_15pre3
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting information.
(%i1) A: matrix([1, 2], [3, 4]);
B: matrix([5, 6], [7, 8]);
[1 2]
(%o1) [ ]
[3 4]
(%i2)
[5 6]
(%o2) [ ]
[7 8]
(%i3) J1: matrix([0,1],[0,0]);
[0 1]
(%o3) [ ]
[0 0]
(%i4) C: J1 . A;
[3 4]
(%o4) [ ]
[0 0]
(%i5)
Started Maxima


File Edit Options Maxima Help
Maxima 5.46.0 https://maxima.sourceforge.io
using Lisp GNU Common Lisp (GCL) GCL 2.6.14 git tag Version_2_6_15pre3
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting information.
(%i1) A: matrix([1, 2], [3, 4]);
B: matrix([5, 6], [7, 8]);
[1 2]
(%o1) [ ]
[3 4]
(%i2)
[5 6]
(%o2) [ ]
[7 8]
(%i3) J1: matrix([0,1],[0,0]);
[0 1]
(%o3) [ ]
[0 0]
(%i4) C: J1 . A;
[3 4]
(%o4) [ ]
[0 0]
(%i5) C: A1 . J1;
[0 1]
(%o5) A1 . [ ]
[0 0]
(%i6) C: A . J1;
[0 1]
Started Maxima


File Edit Options Maxima Help
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting information.
(%i1) A: matrix([1, 2], [3, 4]);
B: matrix([5, 6], [7, 8]);
(%o1)
(%i2)
(%o2)
(%i3) J1: matrix([0,1],[0,0]);
(%o3)
(%i4) C: J1 . A;
(%o4)
(%i5) C: A1 . J1;
(%o5) A1 .
(%i6) C: A . J1;
(%o6)
(%i7)
Started Maxima


File Edit Options Maxima Help
(%i5) C: A1 . J1;
(%o5) A1 .
(%i6) C: A . J1;
(%o6)
(%i7) A: matrix([a11,a12],[a21,a22]);
(%o7)
(%i8) E11: matrix([1,0],[0,0]);
(%o8)
(%i9) E12: matrix([0,1],[0,0]);
(%o9)
(%i10) E21: matrix([0,0],[1,0]);
(%o10)
(%i11) E22: matrix([0,0],[0,1]);
(%o11)
(%i12)
Started Maxima


File Edit Options Maxima Help
(%o13)
(%i14) C: A . E11;
(%o14)
(%i15) C: E12 . A;
(%o15)
(%i16) D: E21 . A;
(%o16)
(%i17) C+D;
(%o17)
(%i18) A;
(%o18)
(%i19) A . E12 + A . E21;
(%o19)
Started Maxima
File Edit Options Maxima Help
(%i9) E12: matrix([0,1],[0,0]);
(%o9)
(%i10) E21: matrix([0,0],[1,0]);
(%o10)
(%i11) E22: matrix([0,0],[0,1]);
(%o11)
(%i12) C: A . E12;
(%o12)
(%i13) D: A . E21;
(%o13)
(%i14) C: A . E11;
(%o14)
(%i15) C: E12 . A;
(%o15)
Started Maxima
File Edit Options Maxima Help
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting information.
(%i1) E11: matrix([1,0],[0,0]);
(%o1)
(%i2) E12: matrix([0,1],[0,0]);
(%o2)
(%i3) E21: matrix([0,0],[1,0]);
(%o3)
(%i4) E22: matrix([0,0],[0,1]);
(%o4)
(%i5) A: matrix([a11,a12],[a21,a22]);
(%o5)
(%i6) E11 . A + E12 . A;
(%o6)
(%i7)