

Материалы этой лекции доступны здесь: <https://github.com/crazzypeter/zx-spectrum-from-scratch>

Входим в разработку под Sinclair ZX Spectrum (Pentagon 128)



@crazzzyPeter aka Петр Безумный

ZX Spectrum 48 (ZX 82)

Год: 1982

Процессор: Zilog Z80, 3.5 МГц

ОЗУ: 48к

ПЗУ: 16к

Видео: 256x192, 15 цветов

Звук: Бипер

ОС: BASIC

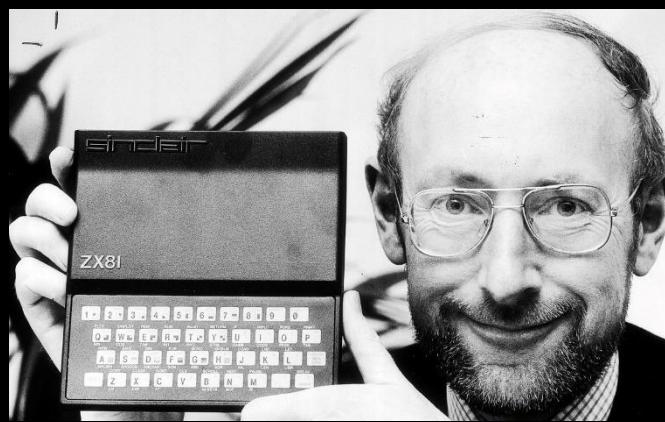
Разработка:

Создатель: Clive Sinclair

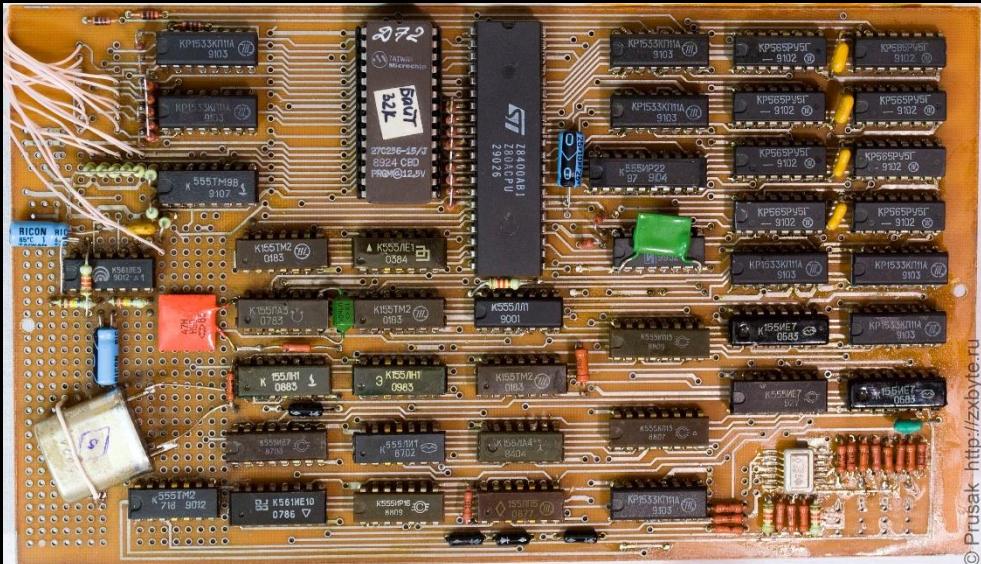
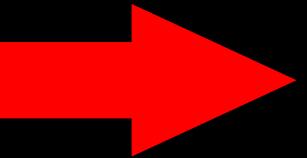
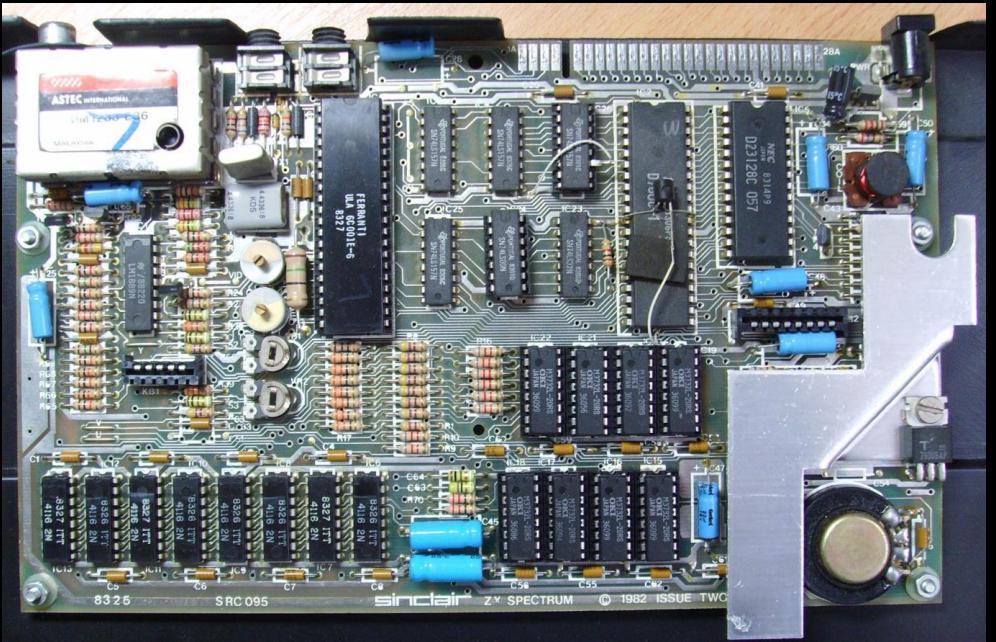
Железо: Richard Altwasser

Софт: Steven Vickers

Дизайн: Rick Dickinson



Клон ZX Spectrum 48 - Ленинград 48



1982г

1987г (Сергей Зонов)



ZX Spectrum 128 (тостер)

Год: 1986

Процессор: Zilog Z80, 3.5 МГц

ОЗУ: 128к

ПЗУ: 64к

Видео: 256x192, 15 цветов
(Две видеостраницы)

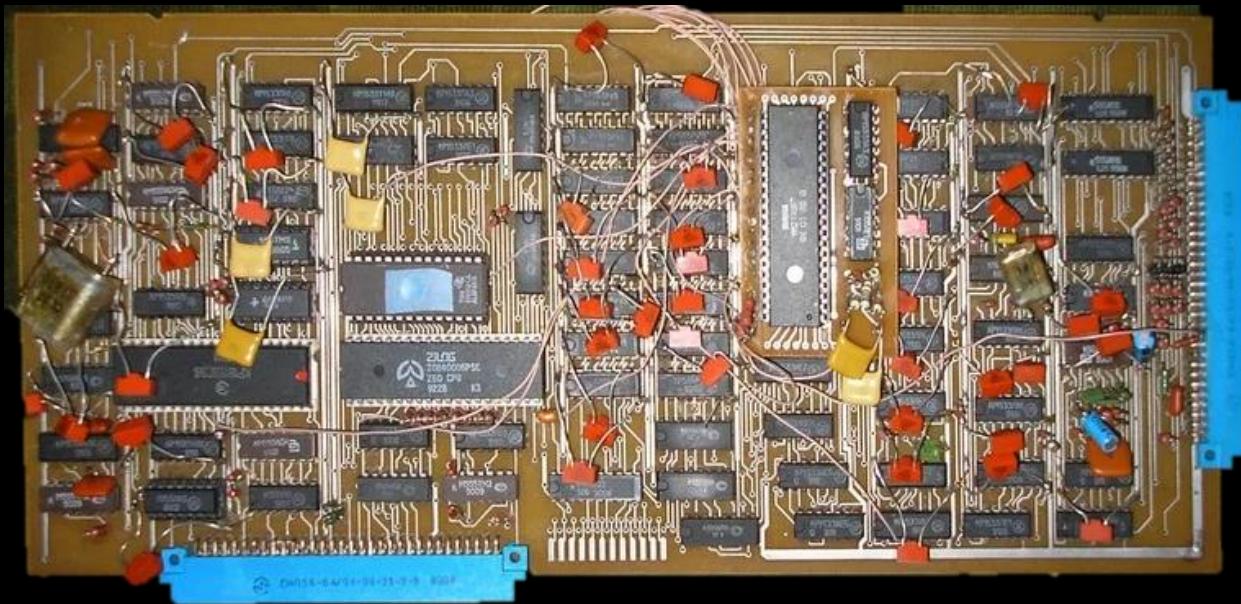
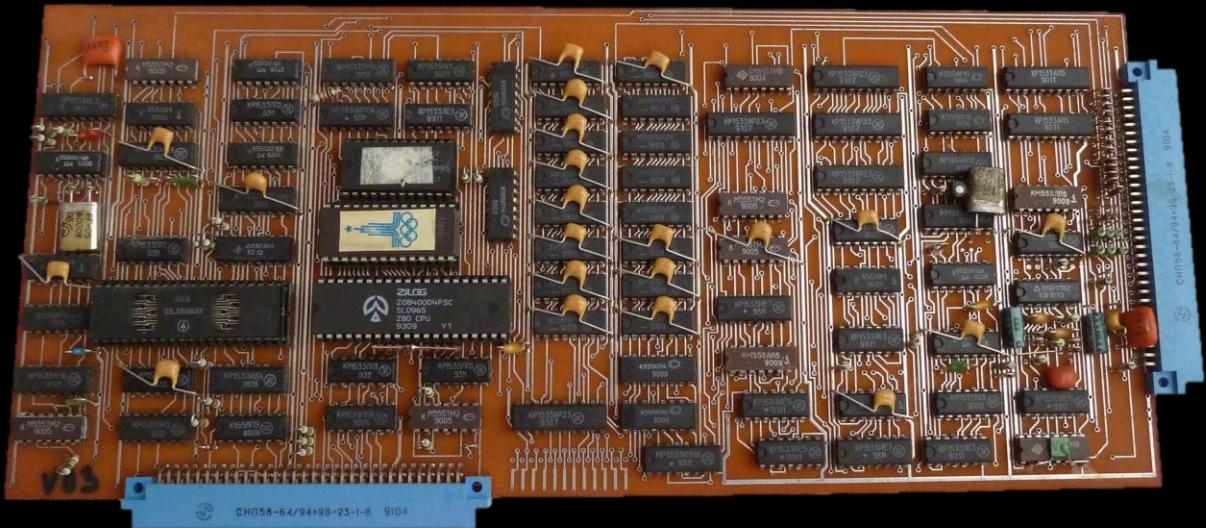
Звук: Бипер + AY-3-8912
(3 канала)

ОС: BASIC



Pentagon 128

1991 - Автор неизвестен



Pentagon 128 (ex ussr)

Год: 1991

Процессор: Zilog Z80, 3.5 Мгц

ОЗУ: 128к

ПЗУ: 64к

Видео: 256x192, 15 цветов

(Две видеостраницы)

Звук: Бипер + АY-3-8912

(3 канала)

ОС: BASIC

Количество тактов на кадр: 71680

Про такты: <http://hype.retroscene.org/773.html>

**Хранение данных: + BDI (Beta Disk Interface) + TR-DOS
(Дисковая ОС, дискеты на 640кб)**





(c) zx-byte.ru

Zilog Z80



8-битный CISC процессор

Разработан в 1976 году

158 инструкций

16 разрядная шина адреса, 64 кб адресуемой памяти



Распределение памяти

Нижняя
память



Верхняя
память

Адрес	Описание
#0000	ПЗУ BASIC/BASIC128
#4000	Экран (6144 байта) 256x192
#5800	Атрибуты (768 байта) 32x24
#5B00	Буфер принтера (256 байт), если не нужен 128ROM, можно использовать
#5C00	Системные переменные BASIC
#5CB6...#5D26	Системные переменные TR-DOS
#5D3B/#5CCB	Начало BASIC программы (в зависимости от того, проинициализирован tr-dos или нет)
#5E3C	Начало BASIC программы, при работе tr-dos с диском (tr-dos использует 257 байт под буфер сектора)
...	Дно стека (стек растет на встречу BASIC программе)
~#6100...#FFFF	***Ваша программа*** До этого адреса память лучше не трогать, если вам важна работоспособность tr-dos и basic

Для сохранения работоспособности BASIC в регистре IY должно быть #5C3A

Структура экрана (#4000)

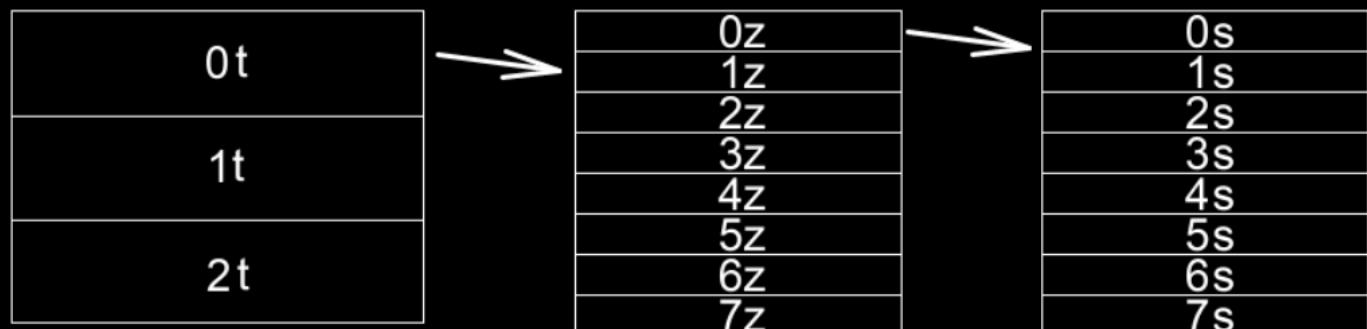
+-----H-----+	+-----L-----+
-+-----+-----+-----+-----+	
0 1 0 t t s s z z z x x x x	
-+-----+-----+-----+-----+-----+	
++ +---+ +---+ +-----+ +-----+	
1 2 3 4	

1 (t) Номер трети 0..2

2 (s) Смещение внутри знакоместа 0..7

3 (z) Номер ряда 0..7

4 (x) Номер столбца



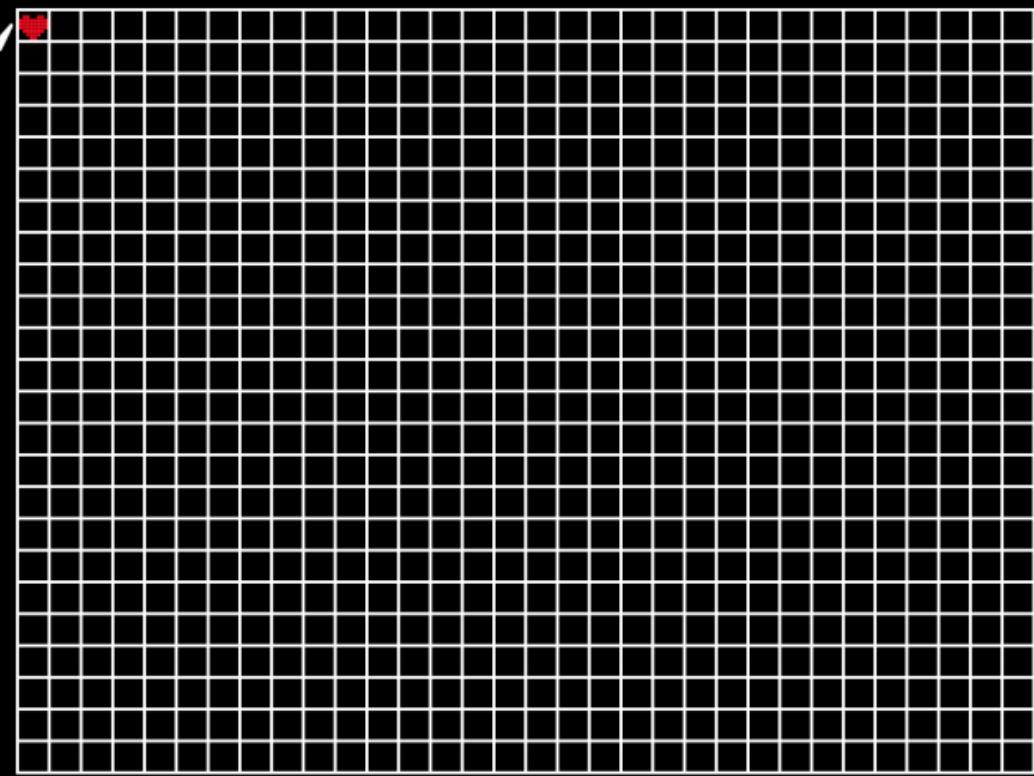
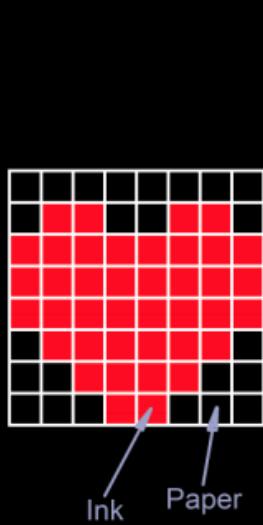
Про экран:

<http://zxpress.ru/article.php?id=18217>

<http://hype.retroscene.org/blog/dev/130.html>

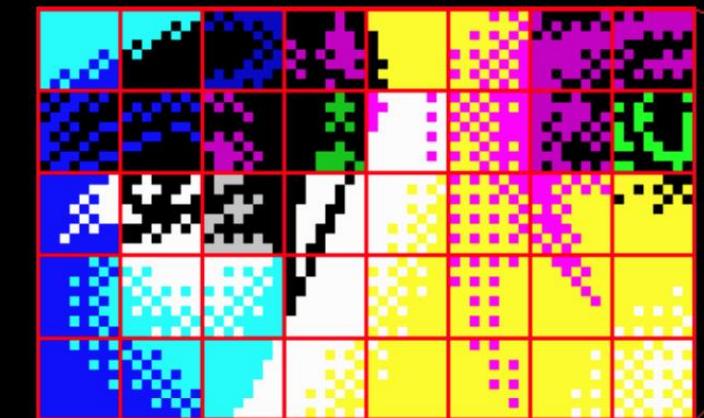
Цвета (#5800)

32



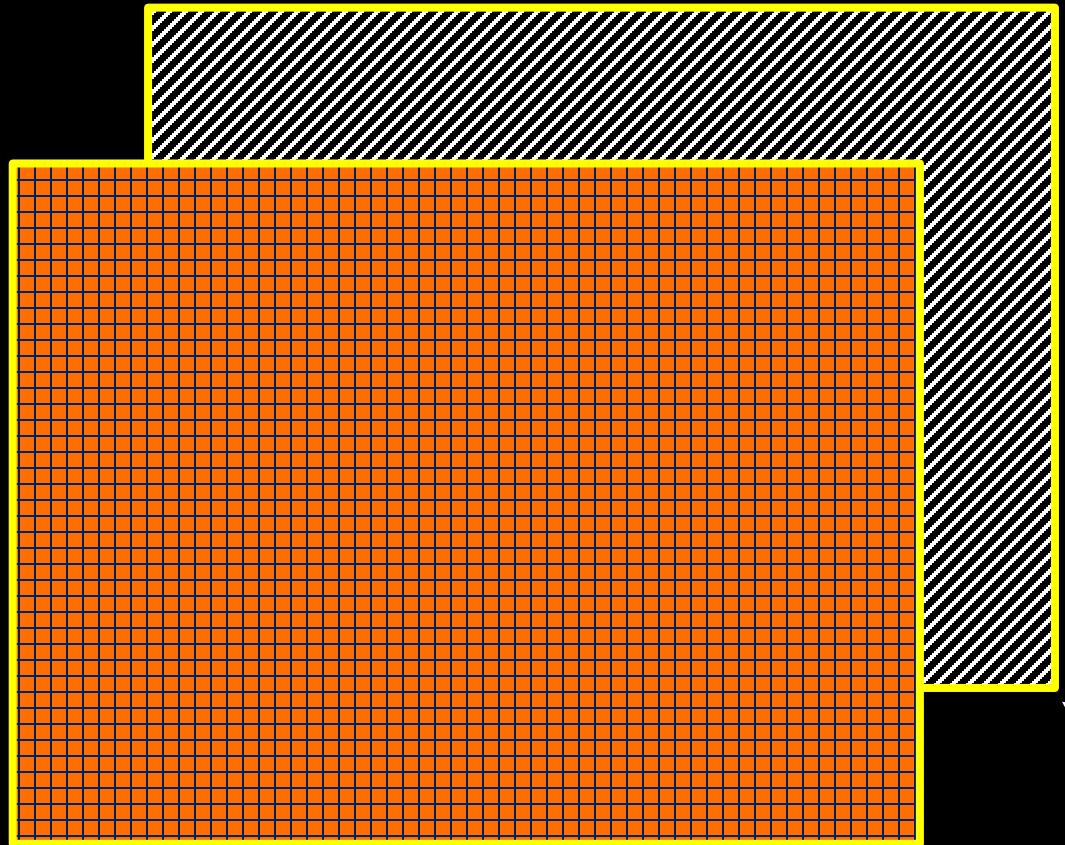
24

B0	B1	Номер	Название
		0	Black
Blue	Blue	1	Blue
Red	Red	2	Red
Magenta	Magenta	3	Magenta
Green	Green	4	Green
Cyan	Cyan	5	Cyan
Yellow	Yellow	6	Yellow
White	White	7	White



- D0..D2 - цвет "чернил" (INK)
- D3..D5 - цвет "бумаги" (PAPER)
- D6 - бит яркости (BRIGHT)
- D7 - бит мерцания (FLASH)

spectrum?



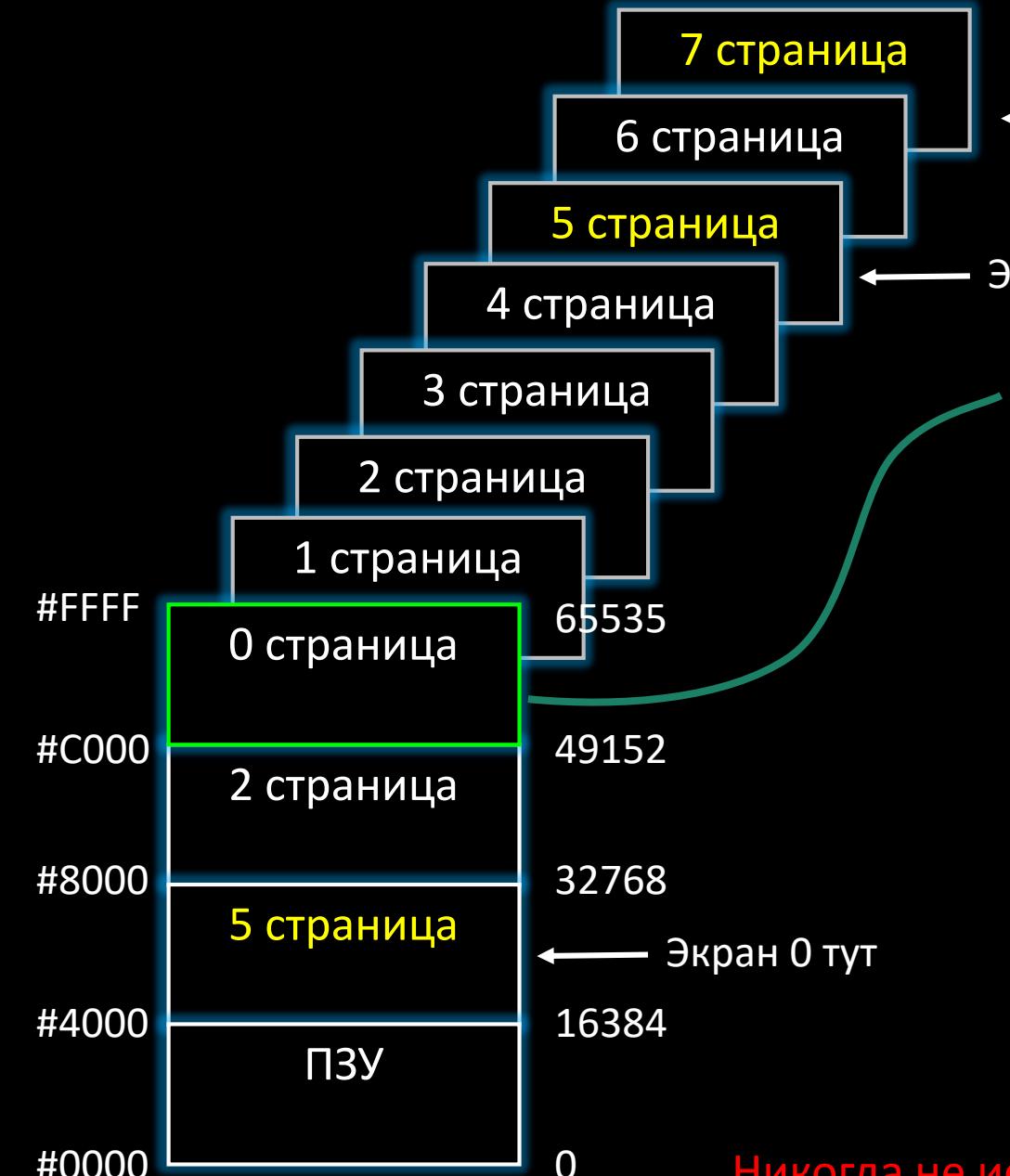
Экран (192x256)
Атрибуты (32x24)



Порт #FE

Чтение	Запись
D0-D4 - отображают состояние определённого полуряда клавиатуры ZX Spectrum. Порты полурядов - #7FFE, #BFFE, #DFFE, #EFFE, #F7FE, #FBFE, #FDDE и #FEFE.	D0-D2 - определяют цвет бордюра. D3 - управляет состоянием выхода записи на магнитофон MIC. D4 - управляет внутренним динамиком (бипером).
D6 - отображает состояние магнитофонного входа.	D5-D7 - обычно не используются.
D5, D7 - обычно не используются.	

Переключение страниц ОЗУ/экранов. Порт #7FFD (порт управляющий отображением страниц)



D0-D2 - номер страницы ОЗУ, подключенной в верхние 16 КБ памяти (**с адреса #C000**)

D3 - выбор отображаемой видеостраницы.

0 - страница в **банке 5**, 1 - в **банке 7. (по умолчанию 0)**

D4 - номер страницы ПЗУ.

0 - BASIC128, 1 - BASIC48. (**обычно используется 1**)

D5 - запрет расширенной памяти (48К защёлка).

(нам надо 0)

При установке бита управление расширенной памятью будет невозможно до сброса компьютера.

Никогда не используйте порт #FD!

AY



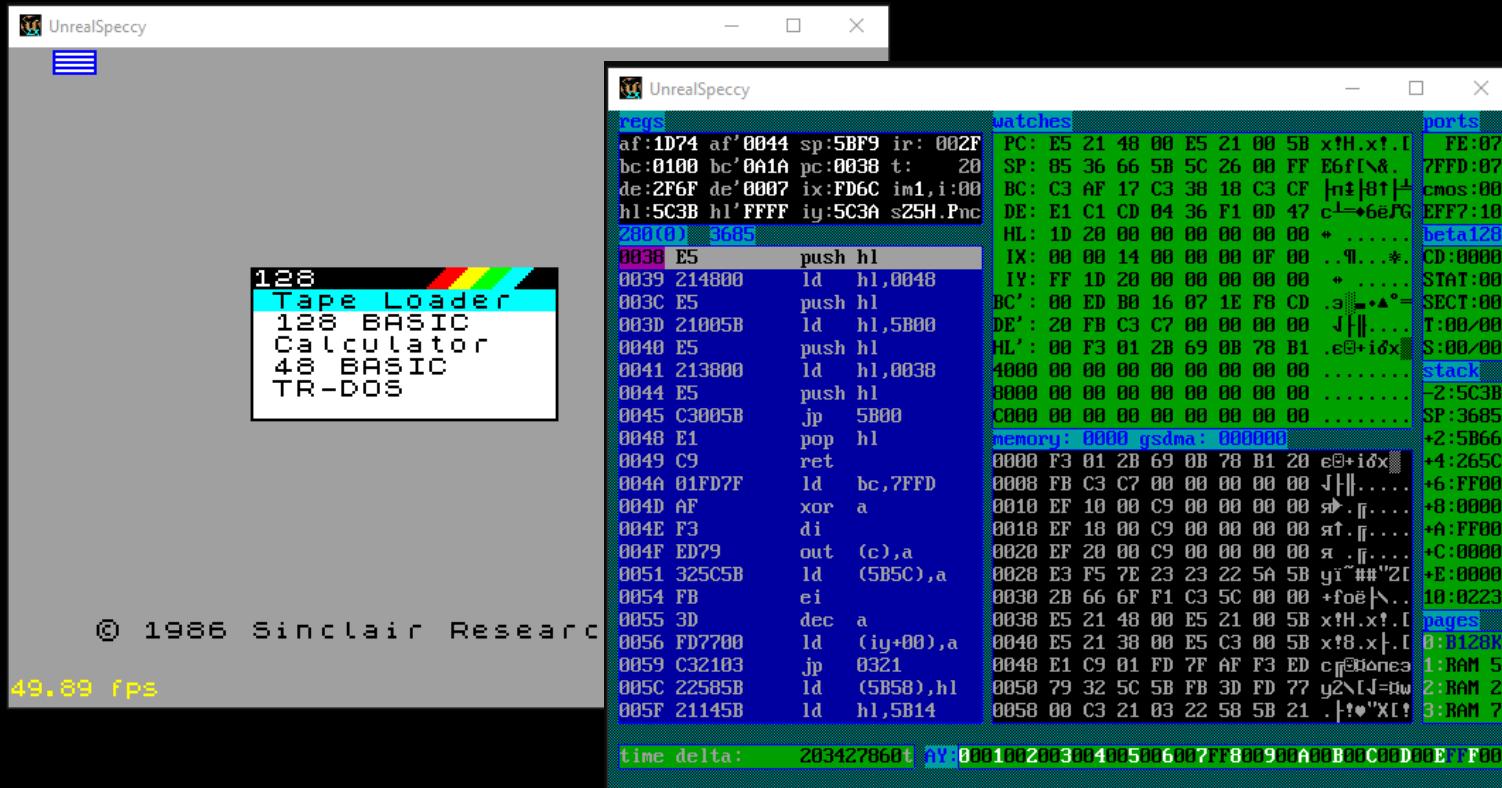
#FFFFD - регистр адреса AY-3-8910

#BFFD - регистр данных AY-3-8910

Используемый софт и рабочая среда



Эмулятор Unreal Speccy (Unreal)



Запуск: unreal <file_name>

Эмулятор умеет отображать пользовательские метки из
файла user.l, при нажатии Ctrl+L.

Файл user.l должен лежать в папке с эмулятором

Загрузить: <http://dlcorp.nedopc.com/viewforum.php?f=8>



Кросс-ассемблер sjAsmPlus

```
----- Compile -----
SjASMPplus Z80 Cross-Assembler v1.18.2 (https://github.com/z00m128/sjasmplus)
Pass 1 complete (0 errors)
Pass 2 complete (0 errors)
> code size: 258
Pass 3 complete
Errors: 0, warnings: 0, compiled: 270 lines, work time: 0.063 seconds
----- Running -----
Скопировано файлов: 1.
UnrealSpeccy 0.39.0 by SMT, Jan 18 2019
```

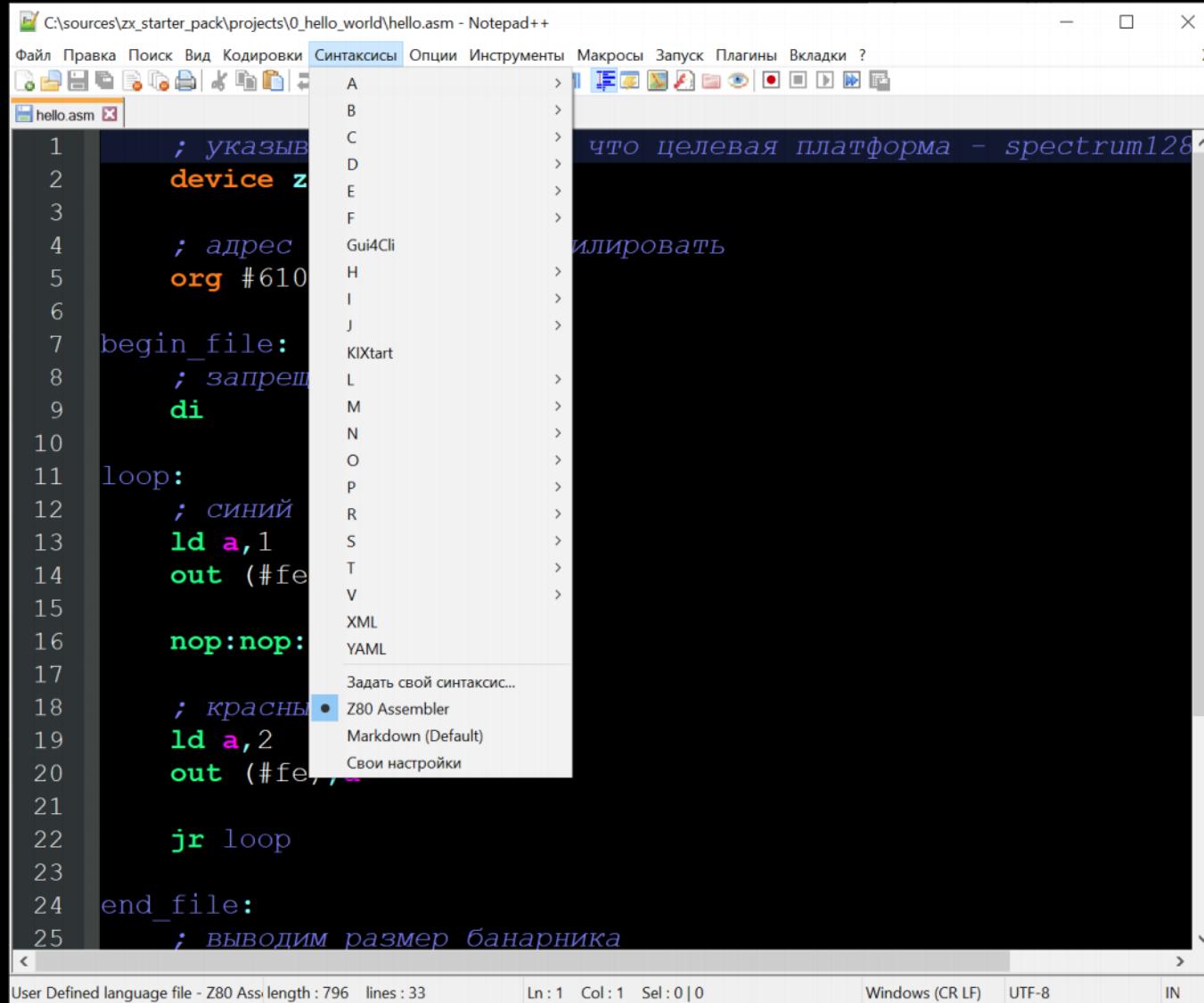
Запуск: sjasmplus <asm_file_name> --syntax=F

Параметр --syntax=F запрещает фейковые инструкции такие как LD HL,DE => LD H,D: LD L,E.

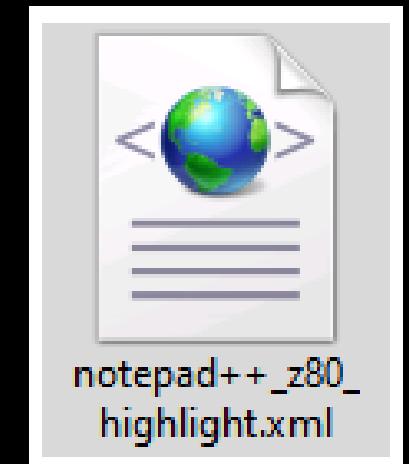
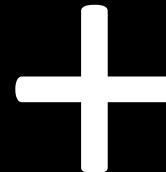
Крайне рекомендую не использовать фейковые инструкции!

Загрузить: <https://github.com/z00m128/sjasmplus>

Редактор Notepad++



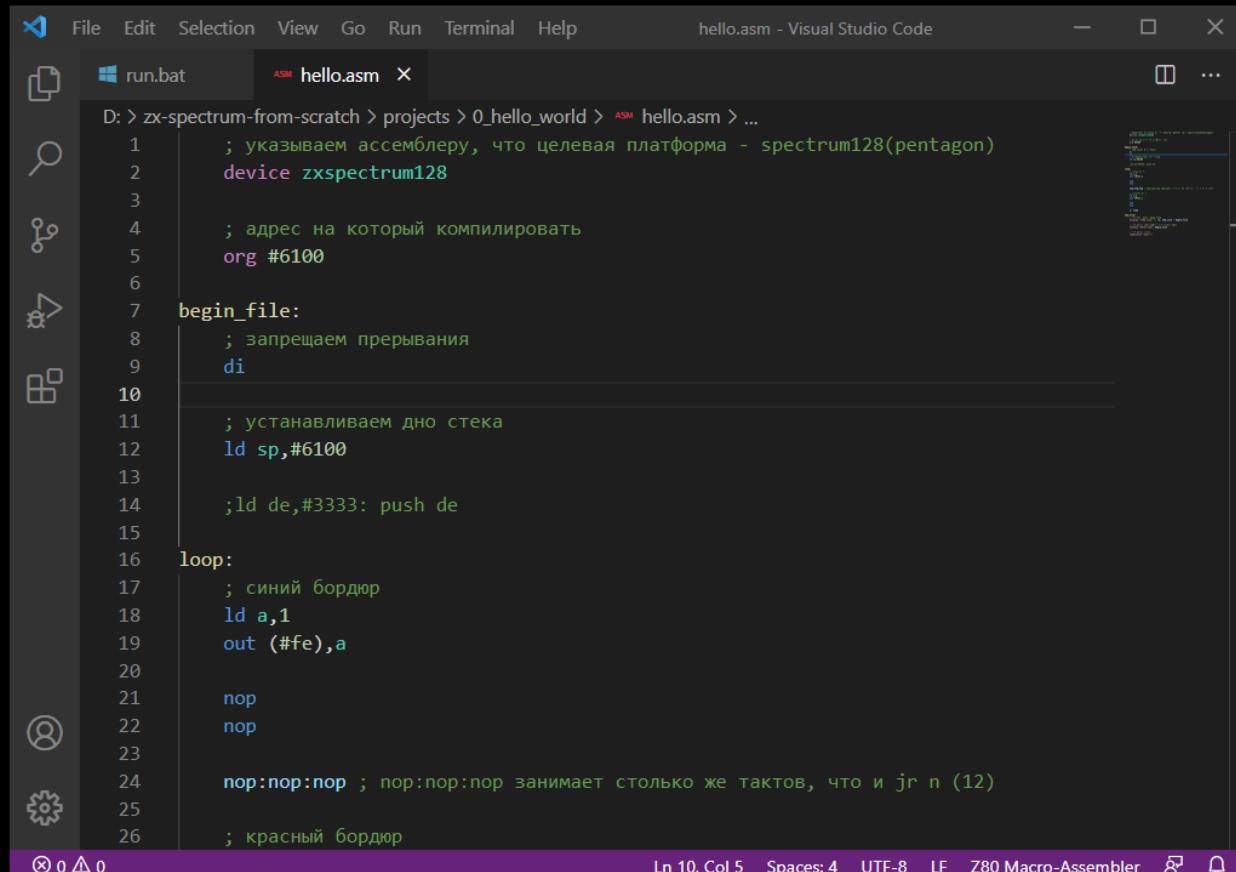
Screenshot of Notepad++ showing a file named "hello.asm". The code is written in Z80 assembly language. The syntax highlighting is custom-made for Z80 assembly. The menu "Синтаксисы" (Syntax) is open, showing various language options like A, B, C, D, E, F, Gui4Cli, H, I, J, KIXtart, L, M, N, O, P, R, S, T, V, XML, and YAML. The option "Z80 Assembler" is selected. The status bar at the bottom shows "User Defined language file - Z80 Ass length : 796 lines : 33" and "Windows (CR LF) UTF-8 IN".



Кастомная подсветка z80 asm

Загрузить: <https://notepad-plus-plus.org/>

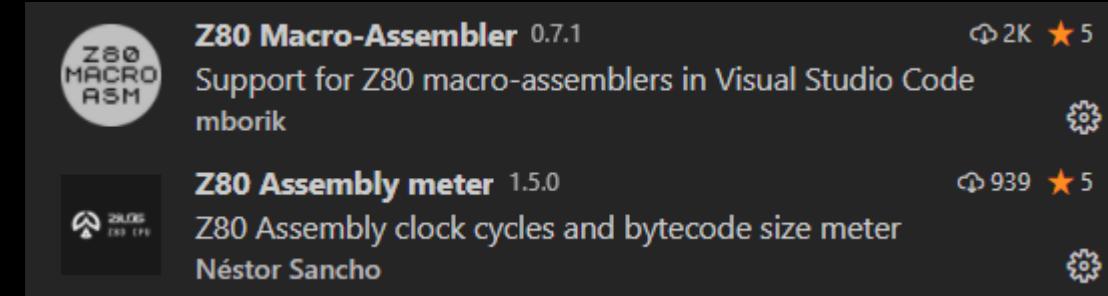
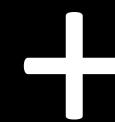
Редактор Visual Studio Code



A screenshot of the Visual Studio Code interface. The title bar says "hello.asm - Visual Studio Code". The left sidebar has icons for File, Edit, Selection, View, Go, Run, Terminal, and Help. There are two tabs open: "run.bat" and "hello.asm". The "hello.asm" tab shows assembly code:

```
D: > zx-spectrum-from-scratch > projects > 0_hello_world > ASM hello.asm > ...
1 ; указываем ассемблеру, что целевая платформа - spectrum128(pentagon)
2 device zxspectrum128
3
4 ; адрес на который компилировать
5 org #6100
6
7 begin_file:
8 ; запрещаем прерывания
9 di
10
11 ; устанавливаем дно стека
12 ld sp,#6100
13
14 ;ld de,#3333: push de
15
16 loop:
17 ; синий бордюр
18 ld a,1
19 out (#fe),a
20
21 nop
22 nop
23
24 nop:nop:nop ; nop:nop:nop занимает столько же тактов, что и jr n (12)
25
26 ; красный бордюр
```

The status bar at the bottom shows "Ln 10, Col 5 Spaces: 4 UTF-8 LF Z80 Macro-Assembler ⚡ ⚡".

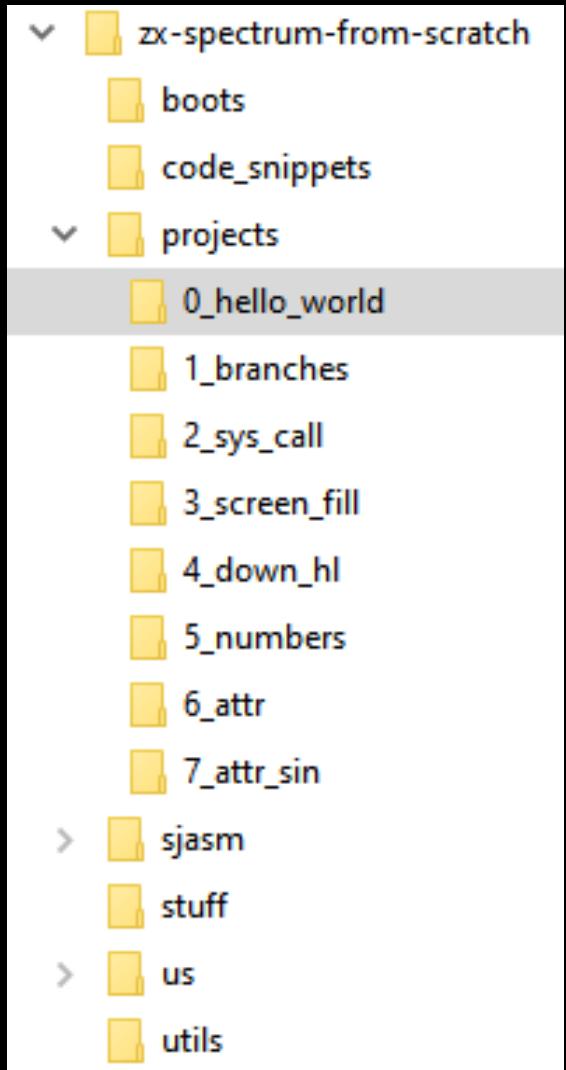


Необходимые расширения

Загрузить: <https://code.visualstudio.com/>

Структура файлов

```
zx-spectrum-from-scratch
|
+---- boots (загрузчики)
|
+---- code_snippets (полезные кусочки кода)
|
+---+ sjasm (папка с ассемблером)
|   |
|   +---- sjasmplus.exe (ассемблер)
|
+---+ us (папка с эмулятором)
|   |
|   +---- unreal.exe (эмуплятор)
|
+---- stuff (разные полезные файлы)
|
+---- utils (полезные утилиты)
|
+---+ projects (ваши проекты)
|   |
|   +---+ project1 (проект 1)
|       |
|       +---- hello.asm (главный ассемблерный файл)
|       |
|       +---- run.bat (скрипт сборки и запуска в эмуляторе)
|       |
|       ... Прочие файлы
|
|   +---- project2 (проект 2)
|
|   ... Прочие проекты
```



Загрузить всю среду: <https://github.com/crazzypeter/zx-spectrum-from-scratch>

Простейшая программа

```
; указываем ассемблеру, что целевая платформа - spectrum128 (pentagon)
device zxspectrum128
; адрес на который компилировать
org #6100
begin_file:
    ; запрещаем прерывания
    di

    ; устанавливаем дно стека
    ld sp,#6100

loop:
    ; синий бордюр
    ld a,1
    out (#fe),a

    ; красный бордюр
    ld a,2
    out (#fe),a

    jr loop

end_file:
    ; выводим размер банарника
    display "code size: ", /d, end_file - begin_file
    ; сохраняем sna(снапшот состояния) файл
    savesna "hello.sna", begin_file
    ; сохраняем метки
    labelslist "user.l1"
```

Указываем, что компиляция для ZX Spectrum 128 (Pentagon 128)

Указываем, в какой адрес компилировать

Циклически меняем цвет бордюра

Сохраняем файл sna для эмулятора,
Указываем, что исполнение начинается с метки begin_file

Сохраняем метки для удобства отладки в эмуляторе
Пример: .\projects\0_hello_world\hello.asm

Желтое – метки
Бирюзовое – директивы компилятора «исполняются» при компиляции (Compile time)
Белое – команды z80
Исполняются при выполнении (Run time)

Метки в ассемблере jsAsmPlus

label: - объявление метки

Пример - бесконечный цикл:

loop:

```
jp loop; jp = goto
```

.label: - объявление локальной метки (существует до следующей метки)

Пример - цикл:

some_proc:

```
...
```

.loop:

```
...
```

```
djnz .loop
```

```
ret
```

name equ value - присвоение name значения value

Пример:

disp **equ** #4000; теперь обращаться к экрану можно через disp.

```
ld hl,disp
```

macro name, args - объявление макроса, аналог **#define**

endm

Пример:

```
macro load_a_plus_b num_a, num_b  
ld a, num_a+num_b  
endm
```

Директивы ассемблера jsAsmPlus

device machine – установка целевой машины

Пример:

```
device zxspecrum128
```

org address – по какому адресу размещать код

Пример:

```
org #6100; размещаем код начиная с адреса #6100
```

display text – отобразить текст/число в консоли

Пример:

```
display "current: ", $; $ – текущий адрес
```

savesna "name",addr – сохранить «состояние» в файл name, с точкой входа addr

Пример:

```
savesna "hello.sna", #6100
```

labelslist "name" – сохранить метки для отладки в файл name

Пример:

```
labelslist "user.l"
```

.N command – повторить (вставить в бинарник) command N раз

Пример:

```
.3 inc hl; три инкремента hl
```

dup N – повторить (вставить в бинарник) code N раз

code

edup

Типы данных в ассемблере z80

122 – десятичное число

#10 – шестнадцатеричное число

%10111011 – двоичное число

'A' – символ (char)

db 1,2,3 – объявить «массив» из 3х **байт**

dw #1000,#2123,#1454 – объявить «массив» из трех **слов**

defb "hello\0" – объявить «строку» с завершающим 0

Байт – 8 бит/1 регистр

Слово – 16 бит/регистровая пара

Простейший скрипт для сборки sna файла

```
@echo off  
  
rem Params  
set name=hello ← Имя проекта  
  
echo -----  
rem Compile main file  
..\.\\.\\siasm\\sjasmplus %name%.asm --syntax=F ← Компиляция asm файла,  
                                                названного как имя проекта  
  
echo -----  
rem Copy labels to emulator  
copy "user.l" "..\\.\\us\\" ← Копирование меток в папку эмулятора,  
del user.l ← Удаляем мусор  
                                                для удобства отладки  
  
rem Run emulator  
..\\.\\us\\unreal %name%.sna ← Запуск эмулятора, с загрузкой созданного sna файла
```



Пример: .\projects\0_hello_world\run.bat

Регистры Z80

Основные регистры		Альтернативные регистры	
A Аккумулятор	F Флаги	A' Аккумулятор	F' Флаги
B	C	B'	C'
D	E	D'	E'
H	L	H'	L'

EX AF,AF
 $AF \Leftrightarrow AF'$

EXX
 $BC,DE,HL \Leftrightarrow BC',DE',HL'$

EX DE,HL
 $DE \Leftrightarrow HL$

BC, DE, HL - регистровые пары

Специальные регистры	
IX (16 бит)	Индексный регистр
IY (16 бит)	Индексный регистр
SP (16 бит)	Указатель стека
PC (16 бит)	Program counter (текущая инструкция)
I	Вектор прерывания
R	Регенерация памяти

IXL, IXH / IYL, IYH - половинки индексных регистров

ЧИЛИМ

Самая простая инструкция, как правило есть в любом ассемблере:

NOP

Данная команда ничего не делает.

Регистры/доступ к памяти

EX AF, AF – обменять AF и AF' ($AF \leftrightarrow AF'$)

EXX – обменять BC,DE,HL и BC',DE',HL' ($BC,DE,HL \leftrightarrow BC',DE',HL'$)

EX DE, HL – обменять DE и HL ($DE \leftrightarrow HL$)

LD R1, R2 – загрузить значение регистра R2 в регистр R1

Например: LD A,B

LD R1, N – загрузить в регистр R1 значение N

Например: LD A, 100

LD RR, NN – загрузить значение NN в регистровую пару RR

Например: LD HL, 3000

LD (RR), R1 - загрузить в память, по адресу в регистровой паре RR, значение из регистра R1

Например: LD (HL), C

LD R1, (RR) - загрузить в регистр R1 значение из памяти, по адресу в регистровой паре RR

Например: LD C, (HL)

LD R, (NN) - загрузить в регистр A / регистровую пару RR значение из памяти, по адресу NNNN

Например: LD A, (#4000) / LD HL, (#4000)

LD (NN), R - загрузить в память, по адресу NNNN, значение регистра A / регистровой пары RR

Например: LD (#4000), A / LD HL, (#4000)

Aka доступ по указателю

Логические операции

OR R – операция OR между регистром A и регистром R

Например: OR B

OR N – операция OR между регистром A и N

Например: OR %00001111

AND R – операция AND между регистром A и регистром R

Например: AND B

AND N – операция AND между регистром A и N

Например: AND %00001111

XOR R – операция XOR между регистром A и регистром R

Например: XOR B

XOR N – операция XOR между регистром A и N

Например: XOR %00001111

CPL – инвертировать регистр A

Например: CPL

ВАЖНО:

Практически всегда операции производятся с регистром A
(Аккумулятором)

Арифметические операции

ADD R – сложить A с значением регистра R

Например: ADD C

ADD N – сложить A с числом N

Например: ADD 7

SUB R – вычесть из регистра A значение регистра N

Например: SUB C

SUB N – вычесть из регистра A число N

Например: SUB 89

INC R – увеличить значение в регистре R

Например: INC D

DEC R – уменьшить значение в регистре R

Например: DEC D

CP R – сравнить значение в регистре A с значением в регистре R

Например: CP B

CP N – сравнить значение в регистре A с значением N

Например: CP 20

ВАЖНО:

Практически всегда операции производятся с регистром А
(Аккумулятором)

Переходы

JP NN – перейти по адресу NN

Например: JP START

JP <NZ,NC,PO,P,Z,PE,C,M>, NN - перейти по адресу NN, в зависимости от условия

Например: JP Z, CLEARED

JR N – относительный переход на -128...+127 байт (на небольшое расстояние)

Например: JR LOOP

JR N – относительный переход на -128...+127 байт, в зависимости от условия

Например: JR C, LOOP

DJNZ N – цикл, уменьшает В, и если В != 0, то переход на N

Например: DJNZ LOOP

Условие	Флаг	Описание
Z	Z	Результат равен 0
NZ	Z	Результат не равен 0
C	C	Произошел перенос
NC	C	Перенос не произошел
P	S	Результат положителен
M	S	Результат отрицателен
PO	P/V	Произошла смена знака
PE	P/V	Знак не поменялся

ВАЖНО:

Флаги устанавливают логические/арифметические операции и команда CP (ComPare)

Подробнее об условиях:

<https://retrocomputing.stackexchange.com/questions/9163/comparing-signed-numbers-on-z80-8080-in-assembly>



Стек/подпрограммы

CALL NN – вызвать подпрограмму по адресу NN,

При этом адрес возврата (куда надо вернуться когда подпрограмма завершиться) будет сохранен в стеке.

(есть условные версии: CALL NZ,NC,PO,PZ,C,PE,M)

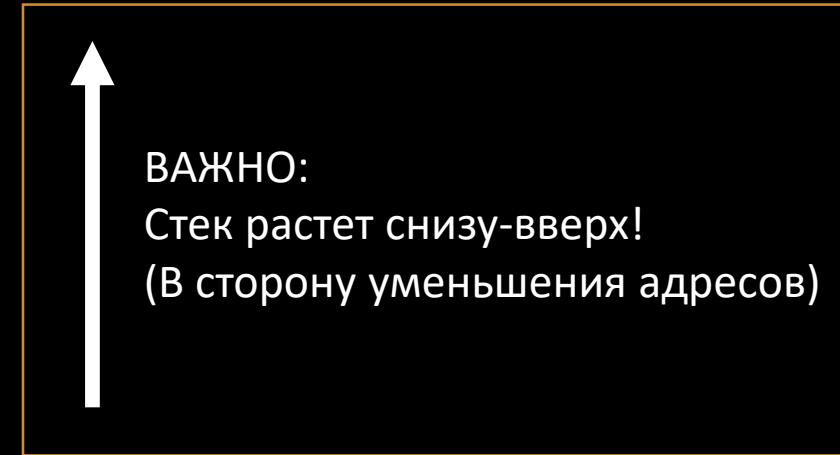
Например: CALL DOWN_HL

RET – возврат

При этом адрес возврата будет извлечен из стека.

(есть условные версии: RET NZ,NC,PO,PZ,C,PE,M)

Например: RET



PUSH RR – загрузить «запушить» в стек значение из регистровой пары RR

Например: PUSH DE

POP RR – загрузить значение из стека в регистровую пару RR

Например: POP DE

Сдвиги

Сдвиги вправо:

SRL R – сдвиг R вправо [0>>]

SRA R – арифметический сдвиг R вправо [0/1>>]

RR R – сдвиг вправо с учетом флага переноса [C>>]



Сдвиги влево:

SLA R – сдвиг R влево [<<0]

SLI R – арифметический сдвиг R влево [<<1]

RL R – сдвиг влево с учетом флага переноса [<<C]

Циклические сдвиги:

RRCA R - циклический сдвиг вправо [>>]

RLCA R - циклический сдвиг влево [<<]

Быстрые сдвиги:

RLA – сдвиг аккумулятора влево

RRA – сдвиг аккумулятора вправо

Порты ввода/вывода

IN A, (N) – ввод значения из порта N в регистр A

Пример: IN A, (#FE)

OUT (N), A – вывод значения из регистра A в порт N

Пример: OUT (#FE), A

IN R, (C) – ввод значения из порта BC в регистр R. (16 бит адреса)

Пример: IN A, (C)

OUT (C), A – вывод значения из регистра A в порт BC. (16 бит адреса)

Пример: OUT (C), A

Пример:

LD A, #10

LD BC, #7FFD

OUT (C), A

Всегда используйте порт #7FFD, никогда
не используйте порт #FD!



Прерывания

Вектор прерываний – регистр R.

Младшая часть адреса берётся с Шины Данных. (в режиме IM2)

Управление прерываниями:

DI – Запрет прерываний

EI – Разрешение прерываний

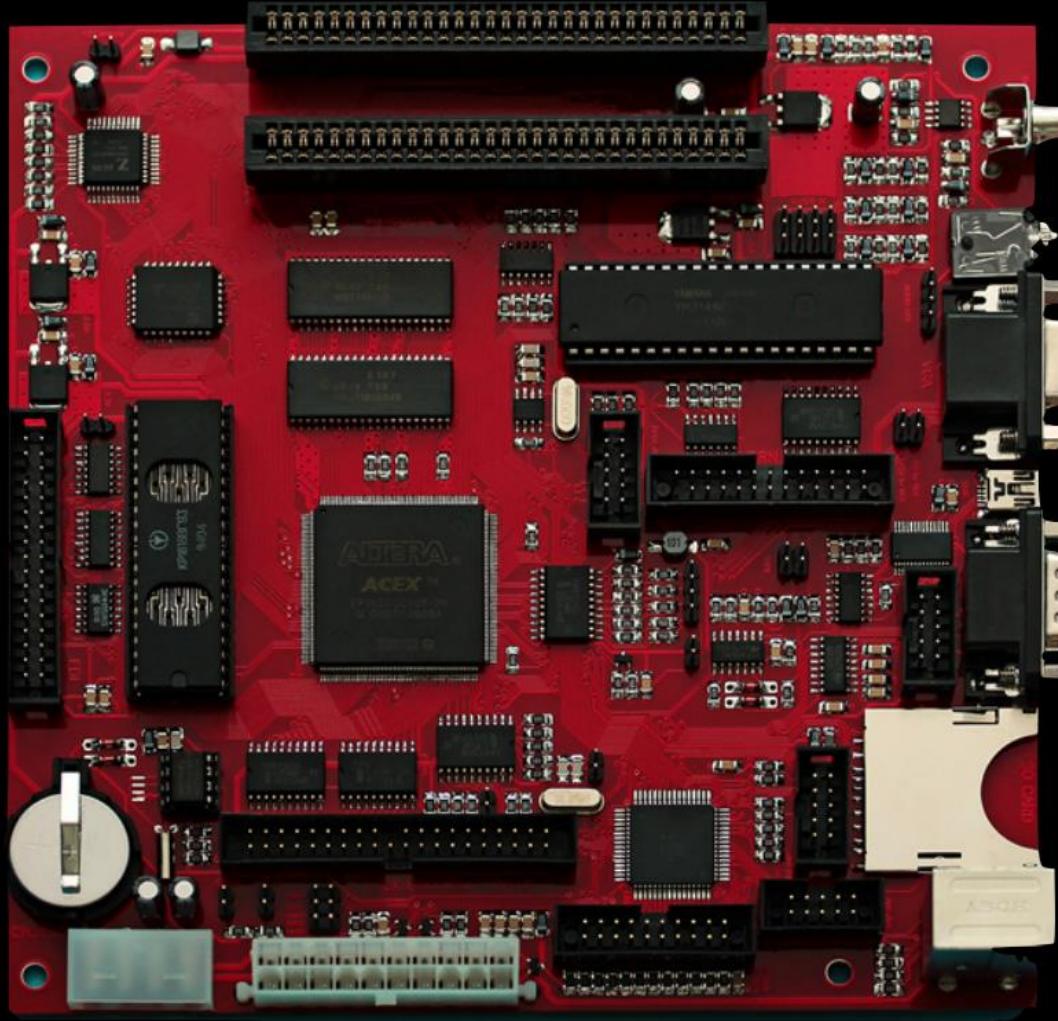
HALT – Ожидание прерывания

Подсказка:

DI: HALT – приводит к умышленному « зависанию » Z80 (полезно для отладки)

В ZX Spectrum прерывания происходят каждый новый кадр => 50 раз в секунду.

Где взять реал? (ZX Evolution)



Информация: <http://www.nedopc.com/zxevo/zxevo.php>
Купить: <http://tetroid.nedopc.com>

Литература

Z80:

Описание команд микропроцессора Z80: [./doc/Полное_описание_команд_микропроцессора_Z80.PDF](#) (в репозитории)

Интерактивная таблица команд Z80: <https://clrhome.org/table/>

ЦЕНТРАЛЬНЫЙ ПРОЦЕССОР Z80CPU: [./doc/Z80-Central-Processor-Unit.pdf](#) (в репозитории)

Документация на sjAsmPlus: <http://z00m128.github.io/sjasmplus/documentation.html>

Математика на Z80:

<http://map.grauw.nl/sources/external/z80bits.html>

<http://z80-heaven.wikidot.com/math>

http://map.grauw.nl/articles/mult_div_shifts.php

<http://zxpress.ru/article.php?id=11746>

Знаковые сравнения:

<https://retrocomputing.stackexchange.com/questions/9163/comparing-signed-numbers-on-z80-8080-in-assembly>

ZX Spectrum:

Огромное количество статей про программирование под ZX Spectrum: <https://zxpress.ru/tag.php?id=13>

Архив книг по ZX Spectrum: <https://vtrd.in/>

Что с фоном?

Литература (продолжение)

Блог про ретросцену:

<https://hype.retroscene.org>

Лекции Яндекса про программирование под ZX Spectrum:

<https://youtu.be/Y-xPRiXO57g>

Коллекция zx-арта, музыки, демок для ZX Spectrum:

<https://zxart.ee/>

Огромная коллекция демок под ZX Spectrum (Pentagon 128):

<https://zxaan.net/>

Большой форум о ZX Spectrum:

<https://zx-pk.ru/>

Контакты

YouTube: <https://youtube.com/c/crazzypeter>

Twitch: <https://www.twitch.tv/crazzypeter>

GitHub: <https://github.com/crazzypeter>

Telegram: @crazzypeter

Instagram: @crazzypeter

Поддержать/donate: <https://www.donationalerts.com/r/crazzypeter>

Материалы этой лекции доступны здесь:

<https://github.com/crazzypeter/zx-spectrum-from-scratch>