I’m Benjamin Misell

@benjaminmisell
github.com/benjaminmisell

Tales of C, the 6502 and the BBC

With added python

Because python is awesome

I’m not

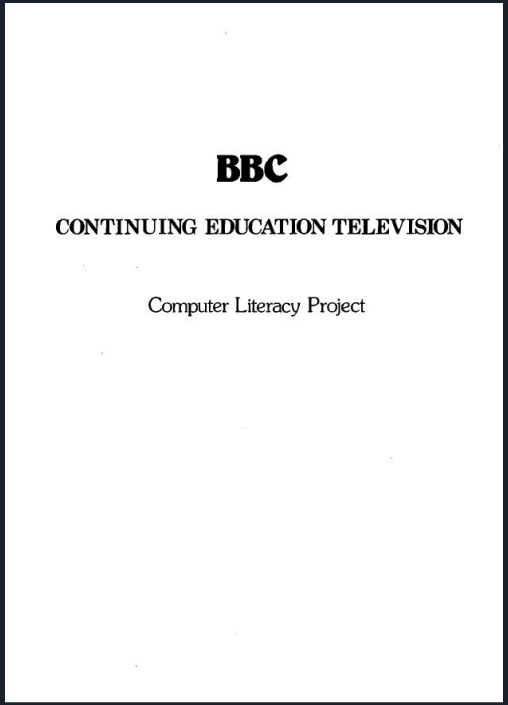
From the 80’s

BACK TO THE 80s

It all starts with

With 80’s appropriate logo

The computer literacy project



BBC

CONTINUING EDUCATION TELEVISION

Computer Literacy Project

The Acorn BBC Micro

6502 assembly

ADC....add with carry
AND....and (with accumulator)
ASL....arithmetic shift left
BCC....branch on carry clear
BCS....branch on carry set
BEQ....branch on equal (zero set)
BIT....bit test
BMI....branch on minus (negative set)
BNE....branch on not equal (zero clear)
BPL....branch on plus (negative clear)
BRK....interrupt
BVC....branch on overflow clear
BVS....branch on overflow set
CLC....clear carry
CLD....clear decimal
CLI....clear interrupt disable
CLV....clear overflow

CMP .... compare (with accumulator)
CPX .... compare with X
CPY .... compare with Y
DEC .... decrement
DEX .... decrement X
DEY .... decrement Y
EOR .... exclusive or (with accumulator)
INC .... increment
INX .... increment X
INY .... increment Y
JMP .... jump
JSR .... jump subroutine
LDA .... load accumulator
LDY .... load X
LDY .... load Y
LSR .... logical shift right
NOP .... no operation
ORA .... or with accumulator
PHA .... push accumulator
PHP .... push processor status (SR)
PLA .... pull accumulator

PLP .... pull processor status (SR)
ROL .... rotate left
ROR .... rotate right
RTI .... return from interrupt
RTS .... return from subroutine
SBC .... subtract with carry
SEC .... set carry
SED .... set decimal
SEI .... set interrupt disable
STA .... store accumulator
STX .... store X
STY .... store Y
TAX .... transfer accumulator to X
TAY .... transfer accumulator to Y
TSX .... transfer stack pointer to X
TXA .... transfer X to accumulator
TXS .... transfer X to stack pointer
TYA .... transfer Y to accumulator

An if (passing)

- LDA #$42 (Load accumulator)
- CMP #$44 (Compare value in accumulator with absolute value)
- BMI thing (Branch on minus)
- JMP elsething
- .thing *** *Do thing* ***
- JMP endthing
- .elsething *** *Do other thing* ***
- .endthing
- *** *Rest of program* ***

An if (failing)

- LDA #$42 (Load accumulator)
- CMP #$40 (Compare value in accumulator with absolute value)
- BMI thing (Branch on minus)
- JMP elsething

.thing *** *Do thing* ***

JMP endthing

- .elsething *** *Do other thing* ***
- .endthing
- *** *Rest of program* ***

I can compilez


EHH. . .NO

How does one compile?

What does a compiler do

Code

C in this case

Other code

Assembly in this case

What does a compiler do

Code		Some things		Other code
C in this case	→	Lexer, Parser, AST etc	→	Assembly in this case

Step 1

A lexer

A lexwhaaaat?

```
int main() {
    return 1 + 2;
}
```

```
Token(INT, ‘int‘)
Token(ID, ‘main‘)
Token(LBRACK, ‘(‘)
Token(RBRACK, ‘)‘)
Token(LBRACE, ‘{‘)
Token(RETURN, ‘return‘)
Token(INT, 1)
Token(PLUS, ‘+‘)
Teken(INT, 2)
Token(SEMI, ‘;‘)
Token(RBRACE, ‘}‘)
```

Step 2

A parser and AST

Autonomous Sugary Topiary?

```
Token(INT, 'int')
Token(ID, 'main')
Token(LBRACK, '(')
Token(RBRACK, ')')
Token(LBRACE, '{')
Token(RETURN, 'return')
Token(INT, 1)
Token(PLUS, '+')
Teken(INT, 2)
Token(SEMI, ';')
Token(RBRACE, '}')
```

```
TranslationUnit
  Function<main:<IntegerCType(2:True)>>()
    Compound
      Return
        Plus
          Number<1>
          Number<2>
```

Step 3

Assembly, finally

Compiled Assembly (No pun this time)

```
NEW
0 FOR opt%=0 TO 2 STEP 2
1 P%=&E00
2 [
3 OPT opt%
4 .__putchar
5 LDY #01
6 LDA (&8E),Y
7 JSR &FFEE
8 STA &71
9 LDA #00
10 STA &70
11 RTS
12
13 .__getchar
14 JSR &FFE0
15 STA &71
16 LDA #00
17 STA &70
18 RTS
19
20 .__osbyte
21 LDY #05
22 LDA (&8E),Y
23 TAY
24 LDY #03
25 LDA (&8E),Y
26 TAX
27 LDY #01
28 LDA (&8E),Y
29 JSR &FFF4
30 STX &71
31 LDA #00
32 STA &70
33 RTS
34
35 \ Routines
36 ._bbcc_pusha PHA
37 LDA &8E
38 BNE _bbcc_pusha_1
39 DEC &8F
40 ._bbcc_pusha_1
41 DEC &8E
42 PLA
43 LDY #00
44 STA (&8E),Y
45 RTS
46 ._bbcc_pulla LDY #0
47 LDA (&8E),Y
48 INC &8E
49 BEQ _bbcc_pulla_1
50 RTS
51 ._bbcc_pulla_1
52 INC &8F
53 RTS
54
55 \ Label
56 ._setup_global
57
58 \ Return
59 RTS
60
61 \ Function
62 ._start
63
64 \ Set
65 LDA #&00
66 STA &8E
67 LDA #&18
68 STA &8F
69
70 \ JmpSub
71 JSR _setup_global
72
73 \ CallFunction
74 JSR __main
75
76 \ Return
77 RTS
78
79 \ Function
80 .__main
81
82 \ Add
83 CLC
84 LDA #&01
85 ADC #&02
86 STA &71
87 LDA #&00
88 ADC #&00
89 STA &70
90
91 \ Return
92 RTS
93 ]
94 NEXT opt%
95 CALL _start
96 RH=&70
97 RL=&71
98 PRINT ~?RH,~?RL
```

Demo time

- ~~Or it be if I could SSH into a RBC Micro~~
- Scrap that! Yes I can!

SSH into a what?

One final thing

How do variables work?

A Symbol Table

A diagram (anyone spot a theme)

ID	Type	Location
a	INT	0x2FA
b	CHAR	0x2FC
c	INT[4]	0x2FD
d	CHAR[3]	0x305
e	INT	0x308

Diagram #I've lost count

Global Scope

ID	Type	Location
a	INT	0x2FA
b	CHAR	0x2FC

Local Scope 1

c	INT[4]	SP+0
d	CHAR[3]	SP+8

Local Scope 2

e	INT	SP+0

Thank you

I've been Benjamin Misell

Questions?

@benjaminmisell
github.com/benjaminmisell/