A2
A Programming Language for the Apple II

Taeber Rapczak <taeber@rapczak.com>
a2lang.com

KansasFest 2022 @ Rockhurst University
Friday 22 July 2022

About Taeber

From Florida

Stand-up comedian

About Taeber

From Florida

Software Engineer

University of Florida (Go Gators!)

Likes programming languages

Wrote a2asm—the most popular 6502 assembler in my entire house!

https://github.com/taeber/a2asm

Obligatory first program

RFC-0X42

...

When presenting examples of ANY computer language, the first program SHALL BE one that prints the words "hello, world".

...

The hello, world according to K&R

C

```
#include <stdio.h>


main()
{
    printf("hello, world\n");
}
```

Kernighan, Brian W. The C Programming Language, 2nd Edition. 1988. ISBN 0-13-110362-8

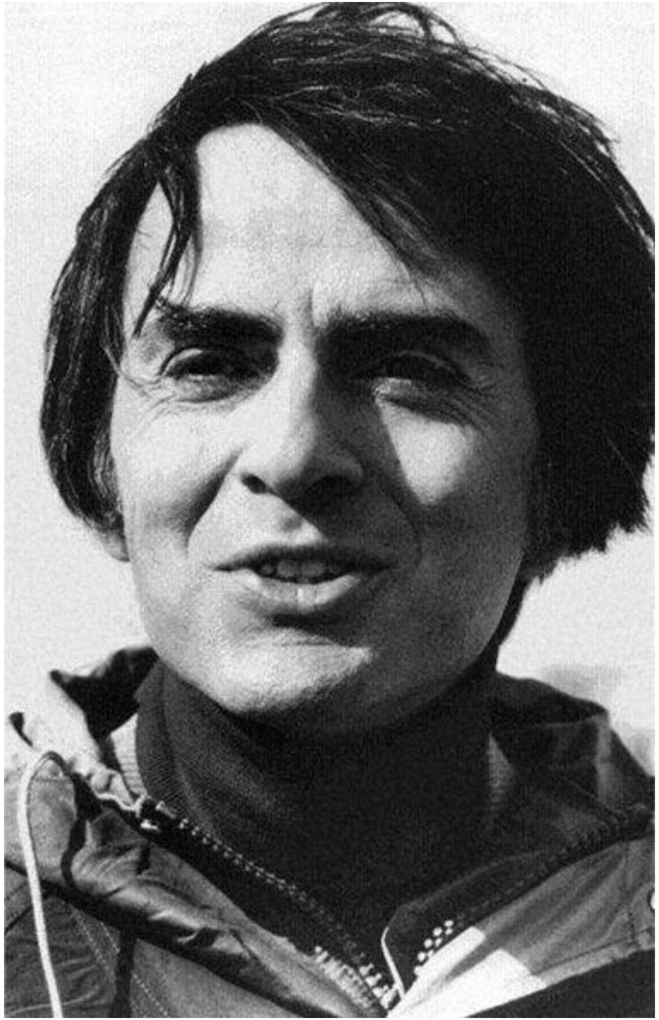
Hello?

A2

```
let main = sub {
   Println("hello, world")
}
```

```
$ a2 compile hello.a2

fatal: unknown symbol: Println
```



If you wish to make an apple pie from scratch, you must first invent the universe.

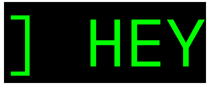
Carl Sagan, *Cosmos (1980)*

https://en.wikipedia.org/wiki/Carl_Sagan#/media/File:Carl_Sagan_-_1980.jpg

```
] HEY
```

ASM

```
       ORG $8000
COUT   EQU $FDED
       LDA #"H"
       JSR COUT
       LDA #"E"
       JSR COUT
       LDA #"Y"
       JSR COUT
```



```
] HEY
```

ASM

```
        ORG $8000
COUT    EQU $FDED
        LDA #"H"
        JSR COUT
        LDA #"E"
        JSR COUT
        LDA #"Y"
        JSR COUT
```

COUT

Apple Monitor Subroutine

Outputs a character

Lives in ROM at memory location $FDED

Expects character to be in Accumulator.

Apple Monitor Subroutine Resources

Assembly Lines (Appendix D)

APPLE2.ROM Disassembly

https://www.6502disassembly.com/a2-rom/APPLE2.ROM.html

Hey, A2!

A2

```
use COUT: sub
    <- [ch: char @ A]
    -> []
    @ $FDED

COUT(`H)
COUT(`E)
COUT(`Y)
```

Hey, A2!

A2

```
use COUT: sub
    <- [ch: char @ A]
    -> []
    @ $FDED


COUT(`H)
COUT(`E)
COUT(`Y)
```

Declaration

use *Name* :*Type*

Parameters

Optional; can be omitted.

Input	**<-** **[...]**
Output	**->** **[...]**

Binding

@ *Address* or *Register*

Hey, A2!

```
use COUT: sub
    <- [ch: char @ A]
    -> []
    @ $FDED


COUT(`H)
COUT(`E)
COUT(`Y)
```

A2

<u>Call</u>

SubroutineName ***(…)***

<u>Character Literals</u>

char type

Backtick-prefixed

Only some printable ASCII [!, ~]

Hey, A2!

A2

```
use COUT: sub
    <- [ch: char @ A]
    -> []
    @ $FDED

COUT(`H)
COUT(`E)
COUT(`Y)
```

ASM

```
COUT    EQU $FDED
        LDA #"H"
        JSR COUT
        LDA #"E"
        JSR COUT
        LDA #"Y"
        JSR COUT
```

Example: *6502 Print U16 as Decimal*

ASM

```
*-------------------------------
* DecPrint - 6502 print 16 bits
* Merlin 8/16/32 assembler
*
* by Michael T. Barry 2017.07.07. Free to
* copy, use and modify, but without warranty
*
* Optimized by J.Brooks & qkumba 7/8/2017
*-------------------------------
     lst off
     org $0300

ZpDecWord = $45 ;U16 being printed

RomPlaPrHex = $FDE2 ;PLA then PrHexZ
RomSave = $FF4A ;A->$45, X->$46, Y->$47

*-------------------------------

Demo ldx #$12
     lda #$34
* 4 byte demo falls into DecPrintU16
```

ASM

```
*-------------------------------
* Print U16 as decimal via COUT
* IN: A=hi, X=lo
* OUT: X=$00, Y=$FF
*-------------------------------
DecPrintU16
       jsr RomSave    ;Save A,X to $45,$46
:DoDigit lda #0       ;Remainder=0
       clv            ;V=0 means div result = 0
       ldx #16        ;16-bit divide
:Div10 cmp #10/2      ;Calc ZpDecWord/10
       bcc :Under10
       sbc #10/2+$80  ;Remove digit & set V=1 to
                      ; show div result > 0
       sec            ;Shift 1 into div result
:Under10 rol ZpDecWord ;Shift /10 result into ZpDecWord
       rol ZpDecWord+1
       rol    ;Shift bits of input into acc (input mod 10)
       dex
       bne :Div10     ;Continue 16-bit divide
       pha            ;Push low digit 0-9 to print
       lda #>RomPlaPrHex-1
       pha            ;Push address of ROM nibble print
       lda #<RomPlaPrHex-1
       pha
       bvs :DoDigit   ;If V=1, result of /10 was > 0
                      ; & do next digit
       rts
       lst off
```

https://groups.google.com/g/comp.sys.apple2/c/_y27d_TxDHA?pli=1

Example: *6502 Print U16 as Decimal*

ASM

```
*-----------------------------
* DecPrint - 6502 print 16 bits
*
...
*
* Print U16 as decimal via COUT
* IN: A=hi, X=lo
* OUT: X=$00, Y=$FF
*-----------------------------

DecPrintU16
...
```

A2

```
; DecPrintU16 prints the
; specified word using COUT
use DecPrintU16: sub
    <- [high: byte @ X
         low: byte @ A]

; Prints $1234 as "4660"
DecPrintU16($12, $34)
```

https://groups.google.com/g/comp.sys.apple2/c/_y27d_TxDHA?pli=1

Example: *6502 Print U16 as Decimal*

ASM

```
*-------------------------------
* DecPrint - 6502 print 16 bits
*
...
*
* Print U16 as decimal via COUT
* IN: A=hi, X=lo
* OUT: X=$00, Y=$FF
*-------------------------------

DecPrintU16
...
```

A2

```
; DecPrintU16 prints the
; specified word using COUT
use DecPrintU16: sub
    <- [high: byte @ X
         low: byte @ A]

; Prints $1234 as "4660"
DecPrintU16($12, $34)
```

https://groups.google.com/g/comp.sys.apple2/c/_y27d_TxDHA?pli=1

Example: *6502 Print U16 as Decimal*

ASM

```
*-------------------------------
* DecPrint - 6502 print 16 bits
*
...
*
* Print U16 as decimal via COUT
* IN: A=hi, X=lo
* OUT: X=$00, Y=$FF
*-------------------------------

DecPrintU16
...
```

A2

```
; DecPrintU16 prints the
; specified word using COUT
use DecPrintU16: sub
    <- [value: word @ XA]



DecPrintU16($1234)
```

https://groups.google.com/g/comp.sys.apple2/c/_y27d_TxDHA?pli=1

Example: *Filling memory with a value*

ASM

```
;======================
; memset
;======================
; a=value
; x=length
; MEMPTRL/MEMPTRH is address
memset:
    ldy    #0
memset_loop:
    sta    MEMPTRL,Y
    iny
    dex
    bne    memset_loop
    rts
```

A2

```
var [MEMPTRL: byte @ $E7
     MEMPTRH: byte @ $E8]
; memset writes length
; bytes of value to memory
; starting at addr.
use memset: sub <- [
    value : byte @ A
    length: int  @ X
    addr  : word @ MEMPTRL
]

memset(addr=$300, value=1,
       length=16)
```

https://github.com/deater/dos33fsprogs/blob/master/asm_routines/memset.s

A2 Declarations as Documentation

Each one of the previous examples had their own documentation format.

A2 declarations provide a consistent subroutine signature.

Initial motivation for A2.

TypeScript and JavaDocs

A2 declarations are machine parseable which could lead to smarter assemblers.

Printing a line of text

A2

```
let main = sub {
   Println("hello, world")
}
```

Printing a line of text

```
use Println: sub
    <- [text: ???]


let main = sub {
    Println("hello, world")
}
```

Text Literals and Arrays

```
use Println: sub
    <- [text: char^13]

let main = sub {
    Println("hello, world")
}
```

Text literals

Sequence of char(acters)

NUL-terminated (`h e l l o ,   w o r l d \0`)

Array

Type ^ Size

ArrayName _ Offset (in bytes)

Not array[index]!

Pointers

```
use Println: sub
    <- [text: char^ @ $00FE]

let main = sub {
    Println("hello, world")
}
```

Pointers

Memory Address stored as a *word*

Required to be in Zero Page

Indirect Indexed Mode LDA ($FE),Y

Println, the definition

```
let Println = sub
    <- [txt: char^ @ $FE]
{
    var i: int @ Y
    i := 0
    while txt_i <> 0 {
        COUT(txt_i)
        i += 1
    }
    CROUT()
}
```

Println, the definition

```
let Println = sub
    <- [txt: char^ @ $FE]
{
    var i: int @ Y
    i := 0
    while txt_i <> 0 {
        COUT(txt_i)
        i += 1
    }
    CROUT()
}
```

Definitions

let *Name* = *Value|Type|Subroutine|Number…*

Cannot be redefined or changed

Subroutine definitions without declaration

Variables

```
let Println = sub
    <- [txt: char^ @ $FE]
{
    var i: int @ Y
    i := 0
    while txt_i <> 0 {
        COUT(txt_i)
        i += 1
    }
    CROUT()
}
```

Variable Declaration

var Name : Type

Like other declarations

Can be bound

"Static" lifetime

No recursion for you!

Assignment and Arithmetic

```
let Println = sub
    <- [txt: char^ @ $FE]
{
    var i: int @ Y
    i := 0
    while txt_i <> 0 {
        COUT(txt_i)
        i += 1
    }
    CROUT()
}
```

<u>Assignment</u>

Assign or reassign using :=

<u>Arithmetic Assignment</u>

Add, subtract += -=

Other languages: val1 = val1 + val2

Logic (and, or, exclusive-or) &= |=

No multiply, divide

Repetition Repetition Repetition

```
let Println = sub
   <- [txt: char^ @ $FE]
{
   var i: int @ Y
   i := 0
   while txt_i <> 0 {
      COUT(txt_i)
      i += 1
   }
   CROUT()
}
```

Loops

while *Condition* **{ … }**

repeat same as *continue* in C

stop same as *break* in C

Condition requires a binary operation

Loops

C

```
while (1) {
}

for (i = 0; i < len; i++) {
}


do {
} while (isTrue());
```

A2

```
while 1 <> 0 {
}

i := 0
while i < len {
    i += 1
}

done := 0
while done <> 0 {
    done := isTrue()
}
```

Challenge: Loop compilation

A2

```
let Println = sub
    <- [txt: char^ @ $FE]
{
    var i: int @ Y
    i := 0
    while txt_i <> 0 {
        COUT(txt_i)
        i += 1
    }
    CROUT()
}
```

ASM

```
Println.txt EQU $FE
Println
        LDY #$00
Println._0
        LDA (Println.txt),Y
        BNE Println._1
        JMP Println._2
Println._1
        LDA (Println.txt),Y
        JSR COUT
        INY
        JMP Println._0
Println._2
        JMP CROUT
```

Where to start?

A2

```
asm {
       ORG $800
       JSR main
       JMP $3D0
}
```

ASM

```
       ORG $800
       JSR main
       JMP $3D0
```

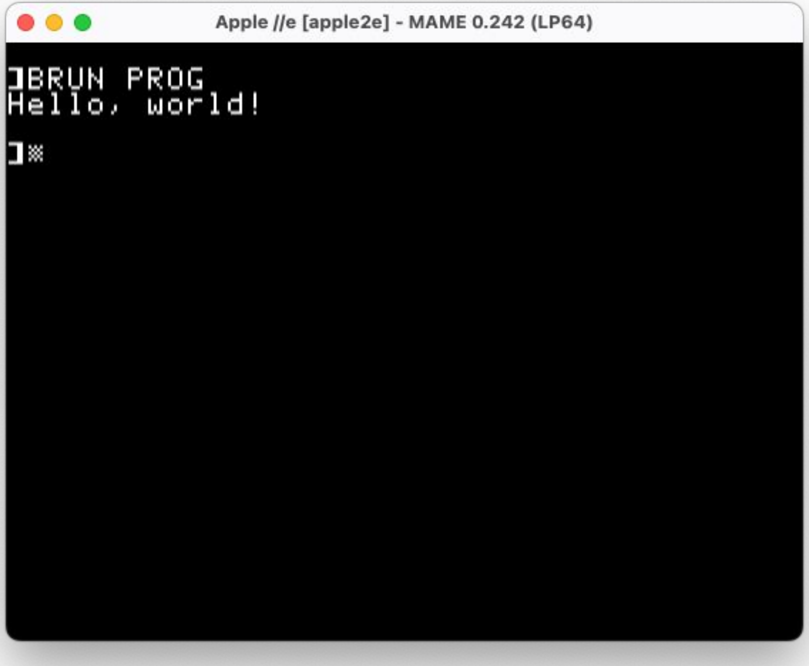
```
asm {
    ORG $800
    JSR main
    JMP $3D0
}

use [
    COUT : sub <- [ch: char @ A] @ $FDED
    CROUT: sub @ $FD8E
]

var PTR: word @ $06

let Println = sub <- [txt: text @ PTR] {
    var i: int @ Y
    i := 0
    while txt_i <> 0 {
        COUT(txt_i)
        i  += 1
    }
    CROUT()
}

let main = sub {
    CROUT()
    Println("hello, world")
}
```



```
]BRUN PROG
Hello, world!

]
```

hello.a2 running in Ample
https://github.com/ksherlock/ample

WARSHIPS

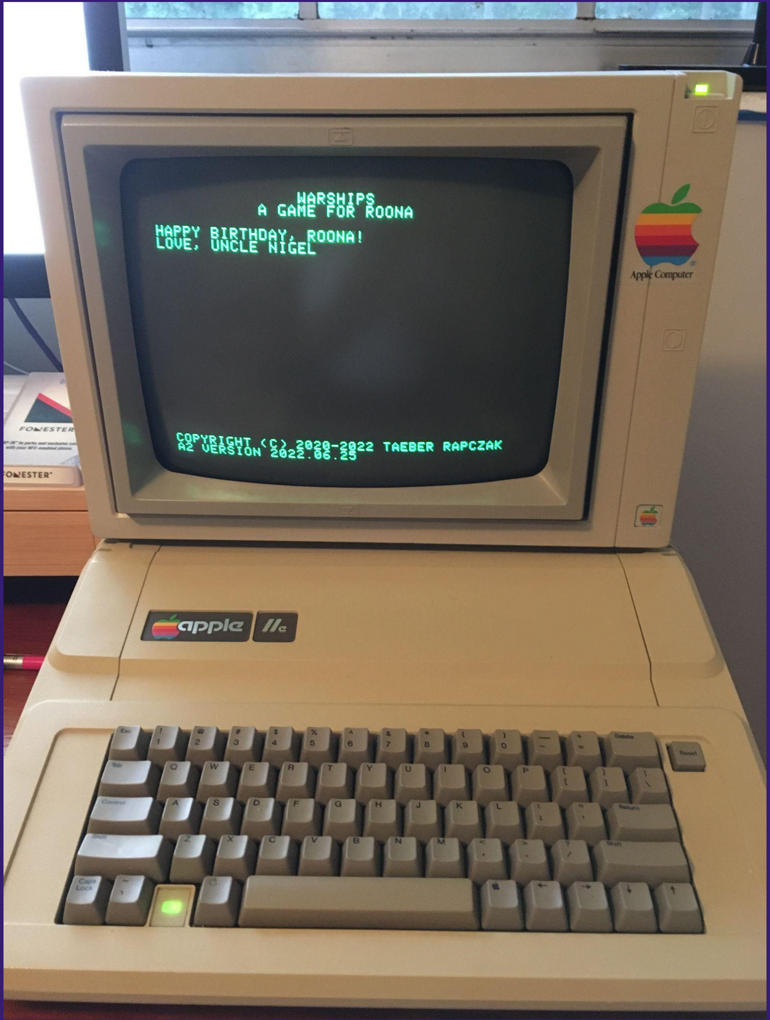
Wrote a Battleships-inspired game in assembly during the development of a2asm

https://github.com/taeber/warships

Rewrote it in A2 to get a sense of the development experience

https://github.com/taeber/a2lang/blob/main/samples/warships.a2

	Line Count	Compiled Size
warships.a2	1052	4131 bytes
warships.asm	913	2219 bytes

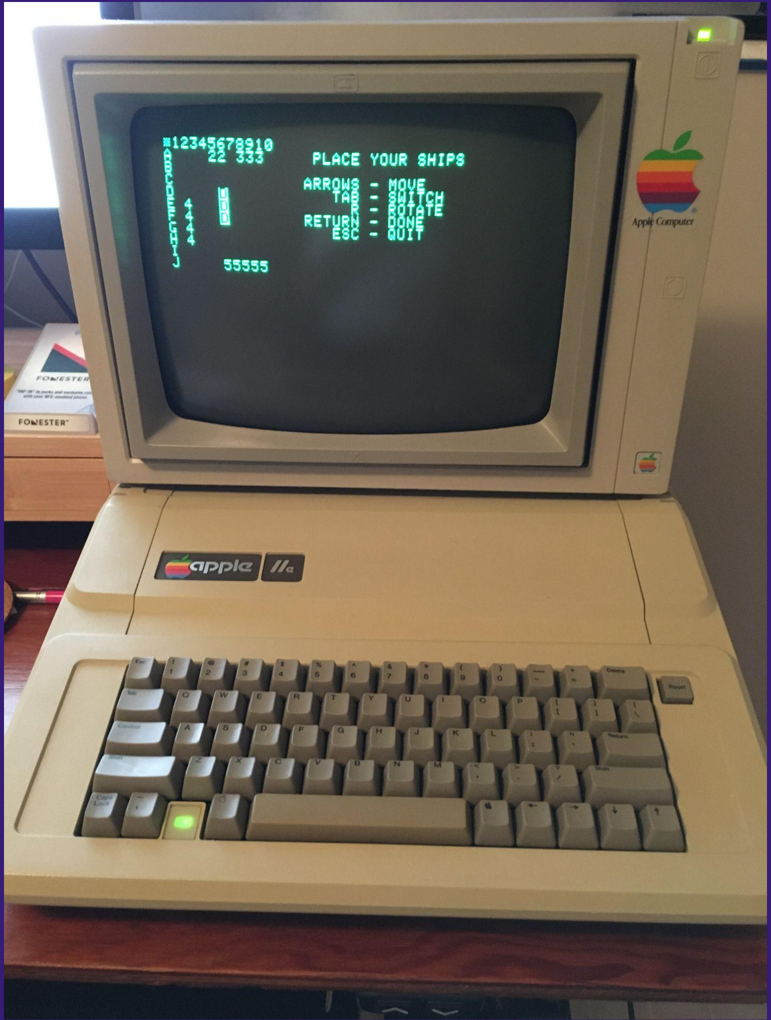


```
          WARSHIPS
      A GAME FOR ROONA

HAPPY BIRTHDAY, ROONA!
LOVE, UNCLE NIGEL



COPYRIGHT (C) 2020-2022 TAEBER RAPCZAK
A2 VERSION 2022.06.25
```



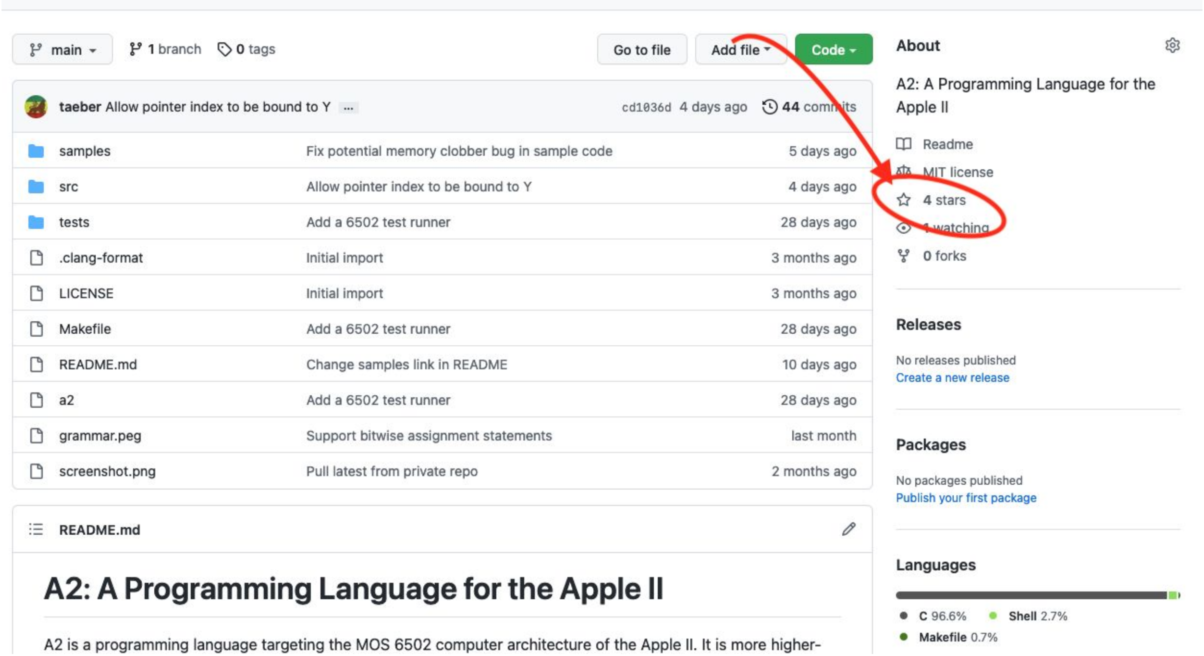
```
▒12345678910
A   22 333        PLACE YOUR SHIPS
B
C                ARROWS - MOVE
D     █             TAB - SWITCH
E 4   █               R - ROTATE
F 4   █          RETURN - DONE
G 4                 ESC - QUIT
H 4
I
J     55555
```

Help!

Star the repo on GitHub!

https://github.com/taeber/a2lang

http://a2lang.com



taeber Allow pointer index to be bound to Y ... cd1036d 4 days ago 44 commits

samples	Fix potential memory clobber bug in sample code	5 days ago
src	Allow pointer index to be bound to Y	4 days ago
tests	Add a 6502 test runner	28 days ago
.clang-format	Initial import	3 months ago
LICENSE	Initial import	3 months ago
Makefile	Add a 6502 test runner	28 days ago
README.md	Change samples link in README	10 days ago
a2	Add a 6502 test runner	28 days ago
grammar.peg	Support bitwise assignment statements	last month
screenshot.png	Pull latest from private repo	2 months ago

README.md

A2: A Programming Language for the Apple II

A2 is a programming language targeting the MOS 6502 computer architecture of the Apple II. It is more higher-

About

A2: A Programming Language for the Apple II

Readme

MIT license

4 stars

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Languages

- C 96.6%
- Shell 2.7%
- Makefile 0.7%

Future work

Finish implementing grammar (multiple returns, subroutine pointers, inline subs)

Add more useful compiler error messages

Carry flag as error indicator (`err: bool @ CF`)

Columnar layout alternative for arrays (struct-of-arrays vs array-of-struct)

Stack-based parameters and variables (`arg1: byte @ stack`)

Compile-time functions (`sizeof!() len!() multiply!()`)

Standard library

Rewrite a2asm in C (or add C as target architecture)

Compiler Explorer plugin

Thank you, KansasFest!!

Taeber Rapczak

taeber@rapczak.com

a2lang.com

`Apple II Forever!!`