KansasFest 2019

assembly language
short and sweet

Mark Pilgrim
Mark Lemmert
Charles Mangin
Forrest Lowe
Michael Sternberg
Dennis Kovacich

5 minutes each plus
1 quick question

BIT: you’re holding it wrong

the humble opinions of Mark Pilgrim

BIT for switching

```
BIT $C054
BIT $C052
BIT $C057
BIT $C050
```

BIT for hiding

```
$09B9   CMP #$DE
$09BB   BEQ $09BF
$09BD   SEC
$09BE   BIT $18
```

BIT for hacking

```
JSR $8635
```

↓

```
BIT $8635
```

BIT for real

```
JSR   Has128K
ROR   $F8
JSR   HasJoystick
ROR   $F8
```

BIT by bit

```
$F8  =  11000000
        NV

BIT $F8 →?
```

BIT for branching

```
BIT  $F8
BPL  NoJoystick
BVC  No128K
```

BIT for branching

```
BIT  $F8
BMI  HasJoystick
BVS  Has128K
```

BIT of applause

Please clap

Mark Lemmert

Mark Lemmert

```
    LDA VOLUME_FLAG ;($00 = ON | >=$01 = OFF)
    CMP #$01
    BNE .NOT1
    LDA #$00
    STA VOLUME_FLAG ;($00 = ON | >=$01 = OFF)
    BEQ .EXIT ;(branch always)
.NOT1
    LDA #$01
    STA VOLUME_FLAG ;($00 = ON | >=$01 = OFF)
;Uses 19 bytes
```

Mark Lemmert

```
LDA VOLUME_FLAG ;($00 = ON | >=$01 = OFF)
EOR #$1
STA VOLUME_FLAG ;($00 = ON | >=$01 = OFF)
```

;Uses only 8 bytes

Mark Lemmert

```
;Scenario 1
00000000        ;VOLUME_FLAG = $00
00000001        ;EOR #$01
------------
00000001        ;Accumulator
```

Mark Lemmert

Scenario 2

```
00000001        ;VOLUME_FLAG = $01
00000001        ;EOR #$01
-------------
00000000        ;Accumulator
```

Charles Mangin

Starting simple: Fill the screen

Difficulty: ROM Routine

```
FILLSCREEN                                 ; 217,729 instructions
                LDA COLOR
                JSR $F832
```

Difficulty: Crib from “Compute!”

```
FILLSCREENFAST                          ; 5,403 instructions

                LDA COLOR
                LDY #$78
FILL1           DEY
                STA $400, Y
                STA $480, Y
                STA $500, Y
                STA $580, Y
                STA $600, Y
                STA $680, Y
                STA $700, Y
                STA $780, Y
                BNE FILL1
                RTS
```

COMPUTE! ISSUE 10 / MARCH 1981 / PAGE 62
Clearing The Apple II Low-Resolution Graphics Screen – Sherm Ostrowsky

Difficulty: Unroll the Loops

```
FILLSCREENREALLYFAST
                 LDA COLOR         ; 2 instructions
                                   ; +
                 STA $400          ; 4 instructions * 960 addresses
                 STA $401          ; ======
                 STA $402          ; 3,842 instructions
                 STA $403
                 STA $404
                 STA $405
                 STA $406
                 STA $407
                 STA $408
                 STA $409
                 STA $40a
                 …

                 STA $7F8
```

Multiplication and Division of large Integers

Or perhaps only 64 bit Arithmetic

Forrest Lowe
KansasFest 2019

6502 Arithmetic instructions

ADC -- Add with carry
SBC – Subtract with carry (or borrow)
CMP – Compare (a very important instruction)
ASL – Arithmetic shift Left
LSR – Logical shift Right
ROL – logical Rotate Left
ROR – logical Rotate Right
CLC – Clear Carry
SEC – Set Carry

Source -- Assembly Lines the Complete Book <shameless plug>

Note the multiply and divide instructions?
I don’t see any either, hence this talk.

Data in the 65xxxx processors are in little endian order, that is least significant byte first at the address pointed to, followed by more significant bytes in increasing value.
64 bit numbers occupy 8 consecutive bytes, unused bits set to zero.
The basic routines of arithmetic follow.
Number position address are placed in zero page locations, 2 bytes per address.

Move Number

```
     LDY   #7
M0   LDA   ($0),Y
     STA   ($2),Y
     DEY
     BPL   M0
```

Zero Number

```
     LDA   #0
     LDY   #7
Z0   STA   ($0),Y
     DEY
     BPL   Z0
```

Find highest non-zero byte

```
     LDY   #7
F0   LDA   ($0),Y
     CMP   #0
     BNE   F1
     DEY
     BMI   F0
     INY          Set Y when = 0
F1   …            Y points at MSB
```

Those were the operations that work from high to low.
Most of this has to be from low to high.

Addition ($0)+($2)=($4)

```
     LDY  #0
     CLC
A0   LDA  ($0),Y
     ADC  ($2),Y
     STA  ($4),Y
     INY
     CPY  #8
     BNE  A0
```

Subtraction ($0) – ($2) = ($4)

```
     LDY  #0
     SEC
S0   LDA  ($0),Y
     SBC  ($2),Y
     STA  ($4),Y
     INY
     CPY  #8
     BNE  A0
```

Shift left one bit

```
     LDY  #0
     LDA  ($0),Y
     ASL
     STA  ($0),Y
     INY
     …
```

Shift (continued)

```
SH0  LDA  ($0),Y
     ROL
     STA  ($0),Y
     INY
     CPY  #8
     BNE  SH0
```

That’s the easy part. For Multiply and Divide, you need to zero out the result register. Also note that Multiplier and Dividend get altered.

For multiplication, the simplest approach is to add the Multiplicand to the product then decrement the Multiplier, and repeat until the Multiplier is zero. It is also not a very good approach.
We lack the nibble instructions that could make the approach used in school useable. SO we use a binary version of that approach.

For each one bit in the multiplier, we add the multiplicand to the product, then shift for the next bit. For the zero bit in the multiplier, we just shift.

I will be calling the previous routines as ‘subroutines’ without the correcting of the zero page address, or protecting the index registers.

This will look like pseudo code to some extent.

```
Load  ZP 0,1 with multiplier addr
Load ZP 2,3 with multiplicand addr
Load ZP 4,5 with product addr
      Zero fill product
      Find highest byte of multiplier, store in high
         LDY   high
MP0  LDA   ($0),Y
         LDX   #8      bits to process
 MP1  Shift product
         CLC
         ROL         get bit
         BCS    MP2
 MP3 DEX
         BNE    MP1
         DEY
         BPL     MP0
         JMP    MPD
MP2  Add Multiplicand to product
         JMP    MP3
MPD  …        multiplication complete
```

Division is much like Multiplication. If the divisor is greater than the dividend you are done, the quotient is zero and the remainder is the dividend. Otherwise, subtract the divisor from the dividend and increment the quotient, repeat until the divisor is greater than the remains of the dividend, then the quotient is done, and the **remains** of the dividend is the **remainder**.
As with multiplication, this is simple and not very efficient.

Here is pseudo code for the first part:

Division routine:
Move Divisor ADDR to $0,1
Move Dividend ADDR to $2,3
Move zero to quotient addressed by $4,5

Find high occupied byte in divisor, store offset in HDR
Find high occupied byte in dividend , store offset in HDD

If HDR > HDD then go to DONE

Find high occupied byte in divisor, store offset in HDR
Find high occupied byte in dividend , store offset in HDD

If HDR > HDD then go to DONE

If HDR < HDD then go to Do Division

* If here HDR = HDD

IF divisor(HDR) > dividend(HDD) then go to DONE

Do Division:
Set count to zero
WHILE HDR < HDD
    {move divisor up one byte ; i.e.multiply by 256
    zero LSB
    add 8 to count
    add 1 to HDR}

```
Do Division:
Set count to zero
WHILE HDR < HDD
     {move divisor up one byte ; i.e.multiply by 256
     zero LSB
     add 8 to count
     add 1 to HDR}

DD0:
If divisor < dividend
    {shift divisor left 1 bit
     adjust HDR if needed
     add 1 to count
     go to DD0}
If divisor = dividend then DD2
```

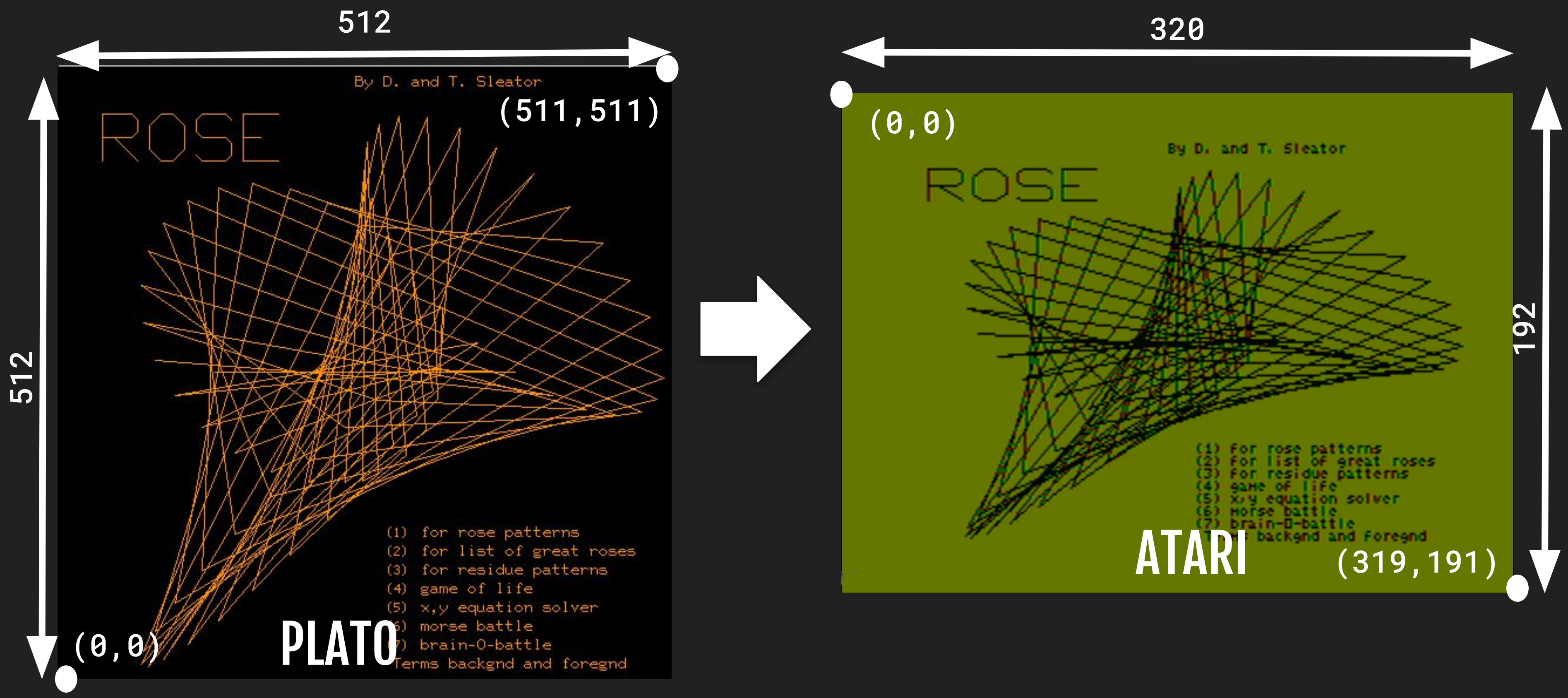
```
DD1:
If divisor > dividend
   {if count = 0  then DONE
    if count > 0
       {shift divisor right 1 bit
        adjust HDR if needed
        subtract 1 from count
        Shift quotient left 1}
    Go to DD1}
DD2:
Shift quotient left 1
Subtract divisor from dividend
Increment quotient
Go to DD1
```

DONE … At this point, quotient is set and the dividend field contains the remainder.

Cosmic Coincidence for Coordinate Scaling?

Michael Sternberg

Problem



512
512
By D. and T. Sleator
ROSE
(511,511)
(1) for rose patterns
(2) for list of great roses
(3) for residue patterns
(4) game of life
(5) x,y equation solver
morse battle
brain-O-battle
Terms backgnd and foregnd
(0,0)
PLATO
320
192
(0,0)
By D. and T. Sleator
ROSE
(1) for rose patterns
(2) for list of great roses
(3) for residue patterns
(4) game of life
(5) x,y equation solver
(6) morse battle
(7) brain-O-battle
backgnd and foregnd
ATARI
(319,191)

PLATO: Why 512 x 512?

- Purpose-built plasma display
- 0..511 = 0,0000,0000..1,1111,1111
- 16x16 Touch screen matrix fits nicely
- Bitmapped font is 8x16
  - Accommodates 64 x 32 characters

3.1.1 The PLATO terminal model (continued)

A summary of the characteristics of terminals which have been adapted for use with PLATO is included in Appendix E.

3.1.1.1 Screen resolution

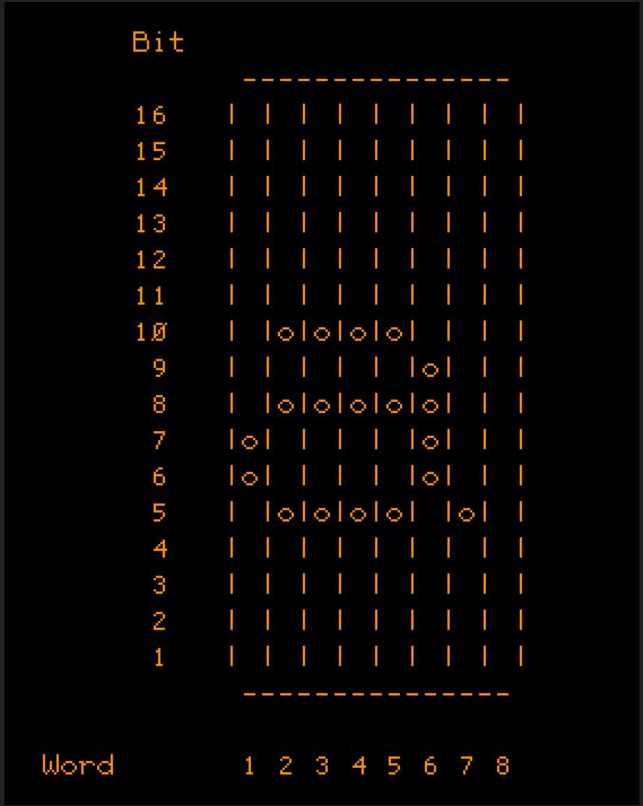
The PLATO terminal model has a screen resolution of 512 by 512 pixels with a square (1:1) aspect ratio. All screen coordinates are based on this.

The screen has 32 lines of 64 characters each.

The coordinate (0,0) is assumed to be at the lower lefthand corner of the screen. The coordinate (511,511) is at the upper righthand corner.

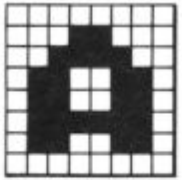
A terminal which has only a 256 vertical by 512 horizontal screen with a 2:1 aspect ratio would scale all vertical coordinates by dividing all Y coordinates by two. A terminal with 384 vertical by 512 horizontal and a 3:4 aspect ratio would multiply Y coordinates by 3/4. If a terminal has a resolution of some odd size, say, 720h by 400v and a 3:4 aspect ratio, scaling by 400/512 would not be appropriate. It would be best to scale by 3/4 and only use the center 512h by 384v area in the center of the screen.

Distortion may be unavoidable on some terminals in order to obtain the entire 512 pixel horizontal resolution. This is the recommended horizontal resolution. With it, 64



ATARI: Why 320 x 192?

- NTSC and color television constraints
- 352 visible half color clocks per scanline
- 8x8 pixel bitmap font
- 40 chars per line @ 8 pixels each = 320 pixels
- 24 chars vertically @ 8 pixels each = 192 pixels



00000000	00
00011000	18
00111100	3C
01100110	66
01100110	66
01111110	7E
01100110	66
00000000	00

Figure 3-2 Character Encoding

Scaling 512x512 to 320x192

```
512 / 320 = 0.625
512 / 192 = 0.375
```

Could use a lookup table

Bit Shifts (Doubling)

A: 0 0 0 0 1 0 0 1   8+1=9

128	64	32	16	8	4	2	1
0	0	0	0	1	0	0	1

ASL A   Arithmetic Shift Left

A: 0 0 0 1 0 0 1 0   16+2=18

128	64	32	16	8	4	2	1
0	0	0	1	0	0	1	0

Bit Shifts (Halving)

A:	0	0	0	0	1	1	0	1	8+4+1=13
	128	64	32	16	8	4	2	1	

LSR A

Logical Shift Right

A:	0	0	0	0	0	1	1	0	4+2=6	1
	128	64	32	16	8	4	2	1		Carry

Scaling 512x512 to 320x192

```
512 / 320 = 5/8
512 / 192 = 3/8
```

Scaling 512x512 to 320x192

512/320 = 5/8 = 1/2 + 1/8

512/192 = 3/8 = 1/4 + 1/8

```
;-----------------------------------------------------------------------
; Scale LSB of X coordinate for the full screen display
; X coordinate multiplied by by 5/8 (or X/2 + X/8) to get from 00..511 to 00..319
; Consider CURSOR2_X = 40.   40/2 + 40/8 = 20 + 5 = 25 = CURSOR1_X
;-----------------------------------------------------------------------
        LDA    CURSOR2_X+1    ; Start with MSB of X coordinate
        LSR    A              ; Shift 9th bit into Carry
        LDA    CURSOR2_X      ; Get LSB of X coordinate
        STA    VAR            ; Keep orig for fine coordinate TODO
        ROR    A              ; X/2 - pull 9th bit from Carry
        STA    TMP            ; Keep X/2 for later
        LSR    A              ; X/4
        LSR    A              ; X/8
        BCS    :+             ; Any bit in Carry? Yes?-->
        LSR    VAR            ;
:       ADC    TMP            ; (X/2)+(X/8) CURSX
        STA    CURSOR1_X      ; Save LSB for full screen X
;-----------------------------------------------------------------------
; Scale MSB of X coordinate for the full screen display
;-----------------------------------------------------------------------
        LDX   #$00            ;
        STX   CURSOR1_X+1     ; Initialize MSB to 0
        BCC   :+              ; --> No overflow from (X/2)+(X/8)
        INC   CURSOR1_X+1     ; Overflow. 255 <= Cursor1_X < 311
```

“The Learning Phone” ATARI 400/800 8K Cartridge (Vincent Wu, Lane Winner, Joe Miller, et al)

Dennis Kovacich

How fast can we count?

Simple AppleSoft - 3 digits

```
]list

 10  TEXT : HOME
 20  FOR X = 0 TO 1000
 30  PRINT X;
 40  HTAB 1
 50  NEXT

]
```

Applesoft - 3 digits with leading zeros

```
10  TEXT : HOME
15 Z = 176:C = 186
20  FOR X = 1024 TO 1026: POKE X,Z: NEXT
30 X = 1026
40 A =  PEEK (X):A = A + 1: POKE X,A
50  IF A < C THEN 40
60  POKE X,Z:X = X - 1
70 A =  PEEK (X):A = A + 1: POKE X,A
80  IF A < C THEN 30
90  POKE X,Z:X = X - 1
100 A =  PEEK (X):A = A + 1: POKE X,A
110  IF A < C THEN 30
120  POKE X,Z
```

Assembly without a loop - 5 digits

```
                 * COUNTER1
                          ORG    $300
                 SCREEN   EQU    $400
                 HOME     EQU    $FC58
                 ZERO     EQU    $B0
                 CARRY    EQU    $BA

0300: 20 58 FC            JSR    HOME
0303: A2 B0               LDX    #ZERO
0305: 8E 00 04            STX    SCREEN
0308: 8E 01 04            STX    SCREEN+1
030B: 8E 02 04            STX    SCREEN+2
030E: 8E 03 04            STX    SCREEN+3
0311: 8E 04 04            STX    SCREEN+4
0314: EE 04 04   ONES     INC    SCREEN+4
0317: AD 04 04            LDA    SCREEN+4
031A: C9 BA               CMP    #CARRY
031C: D0 F6               BNE    ONES
031E: 8E 04 04            STX    SCREEN+4
0321: EE 03 04            INC    SCREEN+3
0324: AD 03 04            LDA    SCREEN+3
0327: C9 BA               CMP    #CARRY
0329: D0 E9               BNE    ONES
032B: 8E 03 04            STX    SCREEN+3
032E: EE 02 04            INC    SCREEN+2
0331: AD 02 04            LDA    SCREEN+2
0334: C9 BA               CMP    #CARRY
0336: D0 DC               BNE    ONES
0338: 8E 02 04            STX    SCREEN+2
033B: EE 01 04            INC    SCREEN+1
033E: AD 01 04            LDA    SCREEN+1
0341: C9 BA               CMP    #CARRY
0343: D0 CF               BNE    ONES
0345: 8E 01 04            STX    SCREEN+1
0348: EE 00 04            INC    SCREEN
034B: AD 00 04            LDA    SCREEN
034E: C9 BA               CMP    #CARRY
0350: D0 C2               BNE    ONES
0352: 60                  RTS
```

With a loop

```
                * COUNTER2

                         ORG    $1000
                SCREEN   EQU    $400
                HOME     EQU    $FC58
                DIGITS   EQU    $06
                ZERO     EQU    $B0
                CARRY    EQU    $BA
                RDKEY    EQU    $FD0C

1000: 20 0C FD           JSR    RDKEY
1003: 29 0F              AND    #$0F
1005: 85 06              STA    DIGITS
1007: 20 58 FC           JSR    HOME
100A: A4 06              LDY    DIGITS
100C: A9 B0     CLEAR    LDA    #ZERO
100E: 99 00 04           STA    SCREEN,Y
1011: 88                 DEY
1012: 10 F8              BPL    CLEAR
1014: A4 06     START    LDY    DIGITS
1016: 20 36 10  ONES     JSR    INC
1019: B9 00 04           LDA    SCREEN,Y
101C: C9 BA              CMP    #CARRY
101E: D0 F6              BNE    ONES
1020: A9 B0     NEXT     LDA    #ZERO
1022: 99 00 04           STA    SCREEN,Y
1025: 88                 DEY
1026: 30 0D              BMI    END
1028: 20 36 10           JSR    INC
102B: B9 00 04           LDA    SCREEN,Y
102E: C9 BA              CMP    #CARRY
1030: D0 E2              BNE    START
1032: 4C 20 10           JMP    NEXT
1035: 60        END      RTS

1036: B9 00 04  INC      LDA    SCREEN,Y
1039: AA                 TAX
103A: E8                 INX
103B: 8A                 TXA
103C: 99 00 04           STA    SCREEN,Y
103F: 60                 RTS
```