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
$F8 = 11000000$

NV

BIT $F8 \rightarrow$?
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
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

    LDA COLOR
    JSR $F832

; 217,729 instructions
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
Difficulty: Unroll the Loops

FILLSCREENREALLYFAST

LDA COLOR ; 2 instructions
            ; +
STA $400    ; 4 instructions * 960 addresses
STA $401
STA $402
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 65xxx 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.

<table>
<thead>
<tr>
<th>Move Number</th>
<th>Zero Number</th>
<th>Find highest non-zero byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDY #7</td>
<td>LDA #0</td>
<td>LDY #7</td>
</tr>
<tr>
<td>M0 LDA ($0),Y</td>
<td>LDY #7</td>
<td>F0 LDA ($0),Y</td>
</tr>
<tr>
<td>STA ($2),Y</td>
<td>Z0 STA ($0),Y</td>
<td>CMP #0</td>
</tr>
<tr>
<td>DEY</td>
<td>DEY</td>
<td>BNE F1</td>
</tr>
<tr>
<td>BPL M0</td>
<td>BPL Z0</td>
<td>BMI F0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set Y when = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Y points at MSB</td>
</tr>
</tbody>
</table>
Those were the operations that work from high to low. Most of this has to be from low to high.

<table>
<thead>
<tr>
<th>Addition ($0) + ($2) = ($4)</th>
<th>Subtraction ($0) - ($2) = ($4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDY #0</td>
<td>LDY #0</td>
</tr>
<tr>
<td>CLC</td>
<td>SEC</td>
</tr>
<tr>
<td>A0  LDA ($0),Y</td>
<td>S0   LDA ($0),Y</td>
</tr>
<tr>
<td>ADC ($2),Y</td>
<td>SBC ($2),Y</td>
</tr>
<tr>
<td>STA ($4),Y</td>
<td>STA ($4),Y</td>
</tr>
<tr>
<td>INY</td>
<td>INY</td>
</tr>
<tr>
<td>CPY #8</td>
<td>CPY #8</td>
</tr>
<tr>
<td>BNE  A0</td>
<td>BNE  A0</td>
</tr>
</tbody>
</table>

Shift left one bit

<table>
<thead>
<tr>
<th>Shift (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDY #0</td>
</tr>
<tr>
<td>LDA ($0),Y</td>
</tr>
<tr>
<td>ASL</td>
</tr>
<tr>
<td>STA ($0),Y</td>
</tr>
<tr>
<td>INY</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
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 \textit{remains} of the dividend is the \textit{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

ROSE

By D. and T. Sleator

(511, 511)

PLATO

(0, 0)

(1) for rose patterns
(2) for list of great roses
(3) for residue patterns
(4) game of life
(5) x,y equation solver
(6) mose battle
(8) brain-o-battle
Terms backend and foreground

ATARI

(319, 191)

(0, 0)

(1) For rose patterns
(2) For list of great roses
(3) For residue patterns
(4) Game of life
(5) X,Y equation solver
(6) Horse battle
(8) Brain-O-battle
Background and Foreground
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
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
Scaling 512x512 to 320x192

\[
\frac{512}{320} = 0.625, \quad \frac{512}{192} = 0.375
\]

Could use a lookup table
Bit Shifts (Doubling)

A: 0 0 0 0 0 1 0 0 1 1

128  64  32  16  8  4  2  1

8 + 1 = 9

ASL A

Arithmetic Shift Left

A: 0 0 0 0 1 0 0 0 1 0

128  64  32  16  8  4  2  1

16 + 2 = 18
Bit Shifts (Halving)

A: 0 0 0 0 0 1 1 0 1

128 64 32 16 8 4 2 1

8 + 4 + 1 = 13

LSR A

Logical Shift Right

A: 0 0 0 0 0 0 1 1 0

128 64 32 16 8 4 2 1

4 + 2 = 6

Carry

1
Scaling 512x512 to 320x192

512 / 320 = 5/8
512 / 192 = 3/8
Scaling $512 \times 512$ to $320 \times 192$

\[
\frac{512}{320} = \frac{5}{8} = \frac{1}{2} + \frac{1}{8}
\]

\[
\frac{512}{192} = \frac{3}{8} = \frac{1}{4} + \frac{1}{8}
\]
; Scale LSB of X coordinate for the full screen display
; X coordinate multiplied 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 ; Initialize MSB to 0
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)
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

```
0300: 20 58 FC  8             JSR   HOME
0303: A2 B0     9             LDX   #ZERO
0305: 8E 00 04 10            STX   SCREEN
0308: 8E 01 04 11            STX   SCREEN+1
030B: 8E 02 04 12            STX   SCREEN+2
030E: 8E 03 04 13            STX   SCREEN+3
0311: 8E 04 04 14            STX   SCREEN+4
0314: EE 04 04 15 ONES       INC   SCREEN+4
0317: AD 04 04 16            LDA   SCREEN+4
031A: C9 BA  17             CMP   #CARRY
031C: D0 F6  18             BNE   ONES
031E: 8E 04 04 19            STX   SCREEN+4
0321: EE 03 04 20            INC   SCREEN+3
0324: AD 03 04 21            LDA   SCREEN+3
0327: C9 BA  22             CMP   #CARRY
0329: D0 E9  23             BNE   ONES
032B: 8E 03 04 24            STX   SCREEN+3
032E: EE 02 04 25            INC   SCREEN+2
0331: AD 02 04 26            LDA   SCREEN+2
0334: C9 BA  27             CMP   #CARRY
0336: D0 DC  28             BNE   ONES
0338: 8E 02 04 29            STX   SCREEN+2
033B: EE 01 04 30            INC   SCREEN+1
033E: AD 01 04 31            LDA   SCREEN+1
0341: C9 BA  32             CMP   #CARRY
0343: D0 CF  33             BNE   ONES
0345: 8E 01 04 34            STX   SCREEN+1
0348: EE 00 04 35            INC   SCREEN
034B: AD 00 04 36            LDA   SCREEN
034E: C9 BA  37             CMP   #CARRY
0350: D0 C2  38             BNE   ONES
0352: 60 C2  39             RTS
```

---

1  * COUNTER1
2             ORG   $300
3    SCREEN   EQU   $400
4    HOME     EQU   $FC58
5    ZERO     EQU   $B0
6    CARRY    EQU   $BA
7
0300: 20 58 FC  8             JSR   HOME
0303: A2 B0     9             LDX   #ZERO
0305: 8E 00 04 10            STX   SCREEN
0308: 8E 01 04 11            STX   SCREEN+1
030B: 8E 02 04 12            STX   SCREEN+2
030E: 8E 03 04 13            STX   SCREEN+3
0311: 8E 04 04 14            STX   SCREEN+4
0314: EE 04 04 15 ONES       INC   SCREEN+4
0317: AD 04 04 16            LDA   SCREEN+4
031A: C9 BA  17             CMP   #CARRY
031C: D0 F6  18             BNE   ONES
031E: 8E 04 04 19            STX   SCREEN+4
0321: EE 03 04 20            INC   SCREEN+3
With a loop

1  * COUNTER2
2  ORG $1000
3  SCREEN EQU $400
4  HOME EQU $FC58
5  DIGITS EQU $06
6  ZERO EQU $B0
7  CARRY EQU $BA
8  RDKEY EQU $FD0C
9  
10

1000: 20 0C FD  JSR RDKEY
1003: 29 0F  AND #$0F
1005: 85 06  STA DIGITS
1007: 20 58 FC  JSR HOME
100A: 20 36 10 1016: 20 36 10  ONES JSR INC
100B: B9 00 04 21 1019: B9 00 04  LDA SCREEN,Y
1010: C9 BA 22 101C: C9 BA  CMP #CARRY
1011: D0 F6 23 101E: D0 F6  BNE ONES
1014: A9 B0 24 1020: A9 B0  LDA #ZERO
1017: 99 00 04 25 1022: 99 00 04  STA SCREEN,Y
101A: 88 27 1025: 88  DEY
101C: 30 0D 28 1026: 30 0D  BMI END
101E: 20 36 10 29 1028: 20 36 10  LDA SCREEN,Y
1021: 20 36 10 30 102B: B9 00 04  JSR INC
1024: A9 B0 31 102E: C9 BA  CMP #CARRY
1027: 99 00 04 32 1030: D0 E2  BNE START
102A: 88 33 1032: 4C 20 10  JMP NEXT
102D: 30 0D 34 1035: 30 0D  END RTS
102E: 20 36 10 35 1038: 20 36 10  LDA SCREEN,Y
1031: 20 36 10 36 103B: 20 36 10  INC LDA SCREEN,Y
1034: A9 B0 37 103E: A9 B0  TAX
1037: 99 00 04 38 103A: E8 38
103C: 99 00 04 40 103B: 8A 39
103F: 60 41

1040: B9 00 04 39 1043: AA 37
1046: 88 38 1049: E8 38
104C: 8A 39
104F: 60 41

1050: B9 00 04 40 1053: 8A 39
1056: 60 41

1059: B9 00 04 42
105C: AA 37
105F: E8 38

1062: 8A 39
1065: 60 41