RUNNING RUBY ON THE APPLE][

COLIN FULTON

TWITTER: @PETERQUINES
GITHUB: JUSTCOLIN
EMAIL: JUSTCOLIN@GMAIL.COM

FIRST, A MEA CULPA

"COME SEE RUBY RUNNING WHERE
IT HAS NEVER RUN BEFORE ..."

RUBY ISN'T RUNNING ON THE APPLE][YET.
I'M SORRY.

"... LEARN HOW SUCH A RICH LANGUAGE
CAN BE SQUEEZED DOWN TO FIT
ON THE HUMBLE APPLE][."

THIS YOU WILL SEE.

PROGRAM LIKE IT'S 1977 ON THE APPLE][

WOZ STYLE

RUBY WOULD BE SO MUCH BETTER

LET'S COMPILE AND RUN CRUBY!
- CRUBY EXPECTS THINGS...
  LIKE A FILE SYSTEM
  AND UNICODE
  ... AND ASCII
- CRUBY BINARY IS OVER 3MB IN SIZE!
  WOULD BE LARGER EVEN ON THE 6502

MRUBY TO THE RESCUE!
- STILL TOO BIG
- NOT WELL OPTIMIZED FOR 8-BIT CPU'S
- C IS TOO HIGH LEVEL

ALL HOPE IS LOST?

FOR THE
APPLE][

THE JOY OF RUBY, BUT REALLY SMALL
REALLY REALLY SMALL

WRITTEN IN ASSEMBLY

NANO-RUBY?

AN "N" IS HALF AN "M"

NRUBY DESIGN

- FLEXIBLE RUBY SYNTAX
- PURE OBJECT ORIENTED
- FEATURES WE ALL LOVE:
  - MODULES
  - BLOCKS
  - IRB
  - ENUMERABLES
  - EVAL
- MORE DYNAMIC THAN ANYONE NEEDS
- DYNAMIC MEMORY ALLOCATION AND GC
- MAKE AS MANY OBJECTS AS YOU WANT!
  AS LONG AS YOU WANT 256, OR FEWER
  256 OBJECTS IS ENOUGH, RIGHT?

ASSEMBLY?!

LESS SCARY THAN YOU MAY THINK:

YOU CAN DO A THING TO A REGISTER
YOU CAN DO A THING TO A BYTE IN MEMORY
YOU CAN JUMP TO ANOTHER PART OF THE CODE

LABEL	MNEMONIC	ARGUMENT
	LDA	#$00
	CLC	
LOOP	ADC	#$01
	CMP	COUNT
	BNE	LOOP
	BRK	

QUICK INTRO TO THE 6502

- 8-BIT PROCESSOR
- "A" REGISTER IS YOUR ACCUMULATOR
- "X" AND "Y" REGISTERS ARE INDEXES
- "FLAG" REGISTER
- "STACK POINTER" REGISTER
- PROGRAM COUNTER
- INSTRUCTIONS ARE 1 TO 3 BYTES
- 56 MNEMONICS FOR INSTRUCTIONS
- 16-BIT ADDRESS SPACE

  . . . HOW DOES THAT WORK?
- PAGES ARE 256 BYTES LONG
- PAGE 1 HAS THE STACK
- PAGE 0 IS SPECIAL

MEMORY MANAGEMENT: GARBAGE COLLECTION

- CRUBY USES A MARK AND SWEEP GC
- THIS HAS MANY ADVANTAGES,
  BUT IT IS VERY COMPLICATED

- REFERENCE COUNTING IS MUCH SIMPLER
- HOWEVER, IT HAS SOME SERIOUS ISSUES

MEMORY MANAGEMENT: DYNAMIC MEMORY

```
 PAGE 0      PAGE 1      PAGE 2     ...
----------------------------------------
[SLOT 0]   [SLOT 16]  [SLOT 32]   ...
[SLOT 1]   [SLOT 17]  [SLOT 33]   ...
[SLOT 2]   [SLOT 18]  [SLOT 34]   ...
[SLOT 3]   [SLOT 19]  [SLOT 35]   ...
[SLOT 4]   [SLOT 20]  [SLOT 36]   ...
[SLOT 5]   [SLOT 21]  [SLOT 37]   ...
[SLOT 6]   [SLOT 22]  [SLOT 38]   ...
[SLOT 7]   [SLOT 23]  [SLOT 39]   ...
[SLOT 8]   [SLOT 24]  [SLOT 40]   ...
[SLOT 9]   [SLOT 25]  [SLOT 41]   ...
[SLOT 10]  [SLOT 26]  [SLOT 42]   ...
[SLOT 11]  [SLOT 27]  [SLOT 43]   ...
[SLOT 12]  [SLOT 28]  [SLOT 44]   ...
[SLOT 13]  [SLOT 29]  [SLOT 45]   ...
[SLOT 14]  [SLOT 30]  [SLOT 46]   ...
[SLOT 15]  [SLOT 31]  [SLOT 47]   ...
```

SLOT MEMORY LAYOUT

```
 0   REFERENCE COUNT
 1   CLASS ID
 2   OBJECT SIZE
 3   DATA
 4   DATA
 5   DATA
 6   DATA
 7   DATA
 8   DATA
 9   DATA
10   DATA
11   DATA
12   DATA
13   DATA
14   DATA
15   NEXT SLOT ID
```

NEXT SLOTS MEMORY LAYOUT

0	REFERENCE COUNT
1	DATA
2	DATA
3	DATA
4	DATA
5	DATA
6	DATA
7	DATA
8	DATA
9	DATA
10	DATA
11	DATA
12	DATA
13	DATA
14	DATA
15	NEXT SLOT ID

CALCULATING OBJECT ID

```
 PAGE 0      PAGE 1      PAGE 2     ...
---------------------------------------
[SLOT 0]   [SLOT 16]  [SLOT 32]   ...
[SLOT 1]   [SLOT 17]  [SLOT 33]   ...
[SLOT 2]   [SLOT 18]  [SLOT 34]   ...
[SLOT 3]   [SLOT 19]  [SLOT 35]   ...
[SLOT 4]   [SLOT 20]  [SLOT 36]   ...
[SLOT 5]   [SLOT 21]  [SLOT 37]   ...
[SLOT 6]   [SLOT 22]  [SLOT 38]   ...
[SLOT 7]   [SLOT 23]  [SLOT 39]   ...
[SLOT 8]   [SLOT 24]  [SLOT 40]   ...
[SLOT 9]   [SLOT 25]  [SLOT 41]   ...
[SLOT 10]  [SLOT 26]  [SLOT 42]   ...
[SLOT 11]  [SLOT 27]  [SLOT 43]   ...
[SLOT 12]  [SLOT 28]  [SLOT 44]   ...
[SLOT 13]  [SLOT 29]  [SLOT 45]   ...
[SLOT 14]  [SLOT 30]  [SLOT 46]   ...
[SLOT 15]  [SLOT 31]  [SLOT 47]   ...
```

CALCULATING OBJECT ID

LET'S SAY WE HAVE AN OBJECT ID OF $CF
$CF == BYTE 240 ($F0) OF PAGE 12 ($0C)

```
OBJ_ID_TO_ADDR TAX
               ASL
               ASL
               ASL
               ASL
               STA    OBJ_ADDR_BYTE
               TXA
               LSR
               LSR
               LSR
               LSR
               CLC
               ADC    #>OBJ_SPACE
               STA    OBJ_ADDR_PAGE
               RTS
```

WE CAN GET CLEVER:

PAGE 0	PAGE 1	PAGE 2	...
[SLOT 0]	[SLOT 16]	[SLOT 32]	...
[SLOT 1]	[SLOT 17]	[SLOT 33]	...
[SLOT 2]	[SLOT 18]	[SLOT 34]	...
[SLOT 3]	[SLOT 19]	[SLOT 35]	...
[SLOT 4]	[SLOT 20]	[SLOT 36]	...
[SLOT 5]	[SLOT 21]	[SLOT 37]	...
[SLOT 6]	[SLOT 22]	[SLOT 38]	...
[SLOT 7]	[SLOT 23]	[SLOT 39]	...
[SLOT 8]	[SLOT 24]	[SLOT 40]	...
[SLOT 9]	[SLOT 25]	[SLOT 41]	...
[SLOT 10]	[SLOT 26]	[SLOT 42]	...
[SLOT 11]	[SLOT 27]	[SLOT 43]	...
[SLOT 12]	[SLOT 28]	[SLOT 44]	...
[SLOT 13]	[SLOT 29]	[SLOT 45]	...
[SLOT 14]	[SLOT 30]	[SLOT 46]	...
[SLOT 15]	[SLOT 31]	[SLOT 47]	...

WE CAN GET CLEVER:

```
 PAGE 0      PAGE 1      PAGE 2       ...
----------------------------------------
[SLOT 0]    [SLOT 1]    [SLOT 2]      ...
[SLOT 16]   [SLOT 17]   [SLOT 18]     ...
[SLOT 32]   [SLOT 33]   [SLOT 34]     ...
[SLOT 48]   [SLOT 49]   [SLOT 50]     ...
[SLOT 64]   [SLOT 65]   [SLOT 66]     ...
[SLOT 80]   [SLOT 81]   [SLOT 82]     ...
[SLOT 96]   [SLOT 97]   [SLOT 98]     ...
[SLOT 112]  [SLOT 113]  [SLOT 114]    ...
[SLOT 128]  [SLOT 129]  [SLOT 130]    ...
[SLOT 144]  [SLOT 145]  [SLOT 146]    ...
[SLOT 160]  [SLOT 161]  [SLOT 162]    ...
[SLOT 176]  [SLOT 177]  [SLOT 178]    ...
[SLOT 192]  [SLOT 193]  [SLOT 194]    ...
[SLOT 208]  [SLOT 209]  [SLOT 210]    ...
[SLOT 224]  [SLOT 225]  [SLOT 226]    ...
[SLOT 240]  [SLOT 241]  [SLOT 242]    ...
```

CALCULATING OBJECT ID

```
$CF == BYTE 192 ($C0) OF 15 PAGE ($0F)
       ($0FC0 INSTEAD OF $0CF0)

OBJ_ID_TO_ADDR TAX
               AND    #$F0
               STA    OBJ_ADDR_BYTE
               TXA
               AND    #$0F
               CLC
               ADC    #>OBJ_SPACE
               STA    OBJ_ADDR_PAGE
               RTS
```

```
OBJ_ID_TO_ADDR TAX
               ASL
               ASL
               ASL
               ASL
               STA    OBJ_ADDR_BYTE
               TXA
               LSR
               LSR
               LSR
               LSR
               CLC
               ADC    #>OBJ_SPACE
               STA    OBJ_ADDR_PAGE
               RTS
```

```
OBJ_ID_TO_ADDR TAX
               AND    #$F0
               STA    OBJ_ADDR_BYTE
               TXA
               AND    #$0F
               CLC
               ADC    #>OBJ_SPACE
               STA    OBJ_ADDR_PAGE
               RTS
```

PARSING RUBY SYNTAX USING ASSEMBLY
MUST BE REALLY HARD, RIGHT?

LET'S WALK THROUGH TURNING RUBY CODE
INTO NRUBY'S VM BYTE CODE.

PARSING RUBY SYNTAX USING ASSEMBLY

SYMBOLS:

FIRST WE NEED A WAY TO TURN SYMBOLS (TOKENS OR ATOMS) INTO A UNIQUE INTEGER

TABLE IN MEMORY: "PUTS DONE DEF"

```
DEF DO_THING(ARG_1, ARG_2)
```

PARSING RUBY SYNTAX USING ASSEMBLY

ASSIGNMENT:

X = 10

```
    OP_LITERAL_INT
    $0A
    $00
    OP_SET_LOCAL
    $4C  (X'S ID)
```

PARSING RUBY SYNTAX USING ASSEMBLY

AMBIGUOUS SYNTAX:

```
SMORP

    OP_CALL_OR_LOCAL
    $E7  (SMORP'S ID)
```

PARSING RUBY SYNTAX USING ASSEMBLY

DOT METHOD:

FOO.BAR(BAZ)

```
    OP_CALL_OR_LOCAL
    $DE  FOO
    OP_DOT_CALL
    $6A  BAR
    OP_CALL_OR_LOCAL
    $08  BAZ
    OP_ARG_APPEND
    OP_ARG_END
```

PARSING RUBY SYNTAX USING ASSEMBLY

```
INTERPOLATED STRINGS:
"HI #{NAME}!"

     OP_LITERAL_STRING
     $C8  H
     $C9  I
     $A0
     $A7  '
     OP_DOT_CALL
     $03  +
     OP_CALL_OR_LOCAL
     $77  NAME
     OP_DOT_CALL
     $03  +
     OP_LITERAL_STRING
     $A1  !
     $A0  '
```

PARSING RUBY SYNTAX USING ASSEMBLY

INTEGERS:

1977

```
0. SET RUNNING TOTAL TO $0000
1. MULTIPLY RUNNING TOTAL BY 10
   1A. COPY RUNNING TOTAL
   1B. SHIFT RUNNING TOTAL LEFT
   1C. SHIFT RUNNING TOTAL LEFT AGAIN
   1D. ADD COPY OF RUNNING TOTAL
   1E. SHIFT RUNNING TOTAL LEFT AGAIN
   X * 10 = ((X * 2 * 2) + X) * 2
2. CONVERT CHARACTER TO INTEGER BYTE
   "1" == $B1, SO JUST SUBTRACT $B0
3. ADD NEW DIGIT TO RUNNING TOTAL
4. MOVE TO NEXT DIGIT
5. REPEAT!
... AND DO OVERFLOW CHECKS AT EVERY STEP
```

PARSING RUBY SYNTAX USING ASSEMBLY

INTEGERS:

1977

```
    OP_LITERAL_INT
    $B9
    $07
```

ASSEMBLING, RUNNING, AND TESTING CODE

- YOU CAN DO TDD IN ASSEMBLY!
- DEVELOPING ON VINTAGE HARDWARE IS NOT A GREAT IDEA
- VIRTUAL][EMULATOR
- MERLIN32 ASSEMBLER
  BASED ON VINTAGE APPLE][ASSEMBLER
  CC65 IS ANOTHER OPTION
- APPLECOMMANDER TO CREATE DSK IMAGES
- ADTPRO TO TRANSFER DISK IMAGES TO A REAL APPLE][USING A SERIAL CABLE

WHAT'S NEXT?

- FINISH NRUBY V0.1
- OTHER PROCESSORS, LIKE THE MSP430
- OTHER COMPUTERS RUNNING THE 6502... LIKE THE NES (2KB OF RAM AND 40KB ROM)

QUESTIONS?

COLIN FULTON

TWITTER: @PETERQUINES
GITHUB: JUSTCOLIN
EMAIL: JUSTCOLIN@GMAIL.COM

THANK YOU!

COLIN FULTON

TWITTER: @PETERQUINES
GITHUB: JUSTCOLIN
EMAIL: JUSTCOLIN@GMAIL.COM