

RUNNING RUBY ON THE APPLE II

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?



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

SLOT MEMORY LAYOUT

	REFERENCE COUNT
0	CLASS ID
1	OBJECT SIZE
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	DATA
16	DATA
17	DATA
18	DATA
19	DATA
20	DATA
21	DATA
22	DATA
23	DATA
24	DATA
25	DATA
26	DATA
27	DATA
28	DATA
29	DATA
30	DATA
31	DATA
32	DATA
33	DATA
34	DATA
35	DATA
36	DATA
37	DATA
38	DATA
39	DATA
40	DATA
41	DATA
42	DATA
43	DATA
44	DATA
45	DATA
46	DATA
47	DATA
48	DATA
49	DATA
50	DATA
51	DATA
52	DATA
53	DATA
54	DATA
55	DATA
56	DATA
57	DATA
58	DATA
59	DATA
60	DATA
61	DATA
62	DATA
63	DATA
64	DATA
65	DATA
66	DATA
67	DATA
68	DATA
69	DATA
70	DATA
71	DATA
72	DATA
73	DATA
74	DATA
75	DATA
76	DATA
77	DATA
78	DATA
79	DATA
80	DATA
81	DATA
82	DATA
83	DATA
84	DATA
85	DATA
86	DATA
87	DATA
88	DATA
89	DATA
90	DATA
91	DATA
92	DATA
93	DATA
94	DATA
95	DATA
96	DATA
97	DATA
98	DATA
99	DATA
100	DATA
101	DATA
102	DATA
103	DATA
104	DATA
105	DATA
106	DATA
107	DATA
108	DATA
109	DATA
110	DATA
111	DATA
112	DATA
113	DATA
114	DATA
115	DATA
116	DATA
117	DATA
118	DATA
119	DATA
120	DATA
121	DATA
122	DATA
123	DATA
124	DATA
125	DATA
126	DATA
127	DATA
128	DATA
129	DATA
130	DATA
131	DATA
132	DATA
133	DATA
134	DATA
135	DATA
136	DATA
137	DATA
138	DATA
139	DATA
140	DATA
141	DATA
142	DATA
143	DATA
144	DATA
145	DATA
146	DATA
147	DATA
148	DATA
149	DATA
150	DATA
151	DATA
152	DATA
153	DATA
154	DATA
155	DATA
156	DATA
157	DATA
158	DATA
159	DATA
160	DATA
161	DATA
162	DATA
163	DATA
164	DATA
165	DATA
166	DATA
167	DATA
168	DATA
169	DATA
170	DATA
171	DATA
172	DATA
173	DATA
174	DATA
175	DATA
176	DATA
177	DATA
178	DATA
179	DATA
180	DATA
181	DATA
182	DATA
183	DATA
184	DATA
185	DATA
186	DATA
187	DATA
188	DATA
189	DATA
190	DATA
191	DATA
192	DATA
193	DATA
194	DATA
195	DATA
196	DATA
197	DATA
198	DATA
199	DATA
200	DATA
201	DATA
202	DATA
203	DATA
204	DATA
205	DATA
206	DATA
207	DATA
208	DATA
209	DATA
210	DATA
211	DATA
212	DATA
213	DATA
214	DATA
215	DATA
216	DATA
217	DATA
218	DATA
219	DATA
220	DATA
221	DATA
222	DATA
223	DATA
224	DATA
225	DATA
226	DATA
227	DATA
228	DATA
229	DATA
230	DATA
231	DATA
232	DATA
233	DATA
234	DATA
235	DATA
236	DATA
237	DATA
238	DATA
239	DATA
240	DATA
241	DATA
242	DATA
243	DATA
244	DATA
245	DATA
246	DATA
247	DATA
248	DATA
249	DATA
250	DATA
251	DATA
252	DATA
253	DATA
254	DATA
255	DATA
256	DATA
257	DATA
258	DATA
259	DATA
260	DATA
261	DATA
262	DATA
263	DATA
264	DATA
265	DATA
266	DATA
267	DATA
268	DATA
269	DATA
270	DATA
271	DATA
272	DATA
273	DATA
274	DATA
275	DATA
276	DATA
277	DATA
278	DATA
279	DATA
280	DATA
281	DATA
282	DATA
283	DATA
284	DATA
285	DATA
286	DATA
287	DATA
288	DATA
289	DATA
290	DATA
291	DATA
292	DATA
293	DATA
294	DATA
295	DATA
296	DATA
297	DATA
298	DATA
299	DATA
300	DATA
301	DATA
302	DATA
303	DATA
304	DATA
305	DATA
306	DATA
307	DATA
308	DATA
309	DATA
310	DATA
311	DATA
312	DATA
313	DATA
314	DATA
315	DATA
316	DATA
317	DATA
318	DATA
319	DATA
320	DATA
321	DATA
322	DATA
323	DATA
324	DATA
325	DATA
326	DATA
327	DATA
328	DATA
329	DATA
330	DATA
331	DATA
332	DATA
333	DATA
334	DATA
335	DATA
336	DATA
337	DATA
338	DATA
339	DATA
340	DATA
341	DATA
342	DATA
343	DATA
344	DATA
345	DATA
346	DATA
347	DATA
348	DATA
349	DATA
350	DATA
351	DATA
352	DATA
353	DATA
354	DATA
355	DATA
356	DATA
357	DATA
358	DATA
359	DATA
360	DATA
361	DATA
362	DATA
363	DATA
364	DATA
365	DATA
366	DATA
367	DATA
368	DATA
369	DATA
370	DATA
371	DATA
372	DATA
373	DATA
374	DATA
375	DATA
376	DATA
377	DATA
378	DATA
379	DATA
380	DATA
381	DATA
382	DATA
383	DATA
384	DATA
385	DATA
386	DATA
387	DATA
388	DATA
389	DATA
390	DATA
391	DATA
392	DATA
393	DATA
394	DATA
395	DATA
396	DATA
397	DATA
398	DATA
399	DATA
400	DATA
401	DATA
402	DATA
403	DATA
404	DATA
405	DATA
406	DATA
407	DATA
408	DATA
409	DATA
410	DATA
411	DATA
412	DATA
413	DATA
414	DATA
415	DATA
416	DATA
417	DATA
418	DATA
419	DATA
420	DATA
421	DATA
422	DATA
423	DATA
424	DATA
425	DATA
426	DATA
427	DATA
428	DATA
429	DATA
430	DATA
431	DATA
432	DATA
433	DATA
434	DATA
435	DATA
436	DATA
437	DATA
438	DATA
439	DATA
440	DATA
441	DATA
442	DATA
443	DATA
444	DATA
445	DATA
446	DATA
447	DATA
448	DATA
449	DATA
450	DATA
451	DATA
452	DATA
453	DATA
454	DATA
455	DATA
456	DATA
457	DATA
458	DATA
459	DATA
460	DATA
461	DATA
462	DATA
463	DATA
464	DATA
465	DATA
466	DATA
467	DATA
468	DATA
469	DATA
470	DATA
471	DATA
472	DATA
473	DATA
474	DATA
475	DATA
476	DATA
477	DATA
478	DATA
479	DATA
480	DATA
481	DATA
482	DATA
483	DATA
484	DATA
485	DATA
486	DATA
487	DATA
488	DATA
489	DATA
490	DATA
491	DATA
492	DATA
493	DATA
494	DATA
495	DATA
496	DATA
497	DATA
498	DATA
499	DATA
500	DATA
501	DATA
502	DATA
503	DATA
504	DATA
505	DATA
506	DATA
507	DATA
508	DATA
509	DATA
510	DATA
511	DATA
512	DATA
513	DATA
514	DATA
515	DATA
516	DATA
517	DATA
518	DATA
519	DATA
520	DATA
521	DATA
522	DATA
523	DATA
524	DATA
525	DATA
526	DATA
527	DATA
528	DATA
529	DATA
530	DATA
531	DATA
532	DATA
533	DATA
534	DATA
535	DATA
536	DATA
537	DATA
538	DATA</

CALCULATING OBJECT ID

PAGE 0	PAGE 1	PAGE 2
FF FF FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00 00 00 00 FF FF FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00 00 00 00 FF FF FF FF FF FF FF FF FF FF 11 11 11 11 11 11 11 11 11 11 44 44 44 44 44 44 44 44 44 44 FF FF FF FF	FF FF FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00 00 00 00 FF FF FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00 00 00 00 FF FF FF FF FF FF FF FF FF FF 66 66 66 66 66 66 66 66 66 66 11 11 11 11 11 11 11 11 11 11 FF FF FF FF FF FF FF FF FF FF	FF FF FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00 00 00 00 FF FF FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00 00 00 00 FF FF FF FF FF FF FF FF FF FF 44 44 44 44 44 44 44 44 44 44 11 11 11 11 11 11 11 11 11 11 FF FF FF FF FF FF FF FF FF FF

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	
	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