GAS144 - A GA144 Simulator-Project in newLISP

Introduction:

This is the first release of my Green-Arrays GA144 chip simulator in newLISP. A proof of concept, it is possible to simulate one node with it, see also www.ForthSeen.de for more.
The development is in progress and the first step towards a complete IDE for the GA144.

Contact:

I am happy about every kind of feedback: mailto:Admin@dmemos.de

Quick Start:

1. Install newLISP. You can download it here: http://www.newlisp.org
2. I assume you have already the „Java run time environment“ from : http://www.java.com
3. The GAS144a source is here: http://www.ForthSeen.de/stuff/GAS144a.zip
   or you can copy it from the following appendix
4. For Windows and for simplicity copy the 7 small files in a new directory C:\GAS144\ ,
   but you can use any other directories if you change the path in line 3 in file „gas-7.lsp“
5. Start newLISP-GS and load and run gas-7.lsp

Some hints for experts who can't wait:

- The hex code in the example is from the Practical Example No. 9

- The program generates a csv file, you can import and analyze it with Excel or LibreOfficeCalc

- gas-7.lsp the main-loop
  line 10: true: generate csv file false: print in newLISP-GS output window
  line 17: iterates max 1000 times, you can fill in any number, 10000 is ok, too
  line 25: you can fill in any or multiple condition (eg: prints the first 500 iterations)
  line 14: define here all variable you like to print

- after the program stops with „bye..“
  - you can look at all variables in the lower monitor window of the newLISP-GS
  - example, the data stack with TSDS or return-stack with RRS
  - you can execute every command like (gand) or (gor) or (g2*)

Why newLISP:

- I use newLISP when I want to enjoy programming
- Free and open source - you can download it from http://www.newlisp.org
- Very powerful, you'll find probably everything you'll need, look into the manual..
- Very small, about 0.3MB - a fraction of what a mp3 song needs
- A scripting language, but really fast
- There is an easy to use interface to Java, if you need a Graphical-User-Interface
- Experts can do magical things with newLISP
- But also amateurs like me or you can use it without problems

Programming Guide Lines:

- Not more than 5 to 10 lines per function
- Not more than one print-page per unit
- Target is simplicity. Slowly, not tons of code
- Always asking what else could be left out
- In doubt function before speed
- Not too tricky, so that I still can understand it after a few weeks
- At best a self- explaining code

Basic Concept:

- For every F18A Instruction there is one function
- The function names needed a preceding “g” - “**!**” is implemented as **(g!)**
- Single registers are single variables
- Stacks and Memory are lists (arrays and list are quite the same in newLISP)
- The combination of all variables and lists build a list, too – one node, one list
- 144 nodes build one GA144 chip
- The nodes can be saved and loaded, example: (GAsave 5) … (GAload 5)
- The whole chip can be saved in a text-file, example: (save “G144.txt” 'GA144)
- At one point in time only one node is active and simulated
- There is one Program-Counter, one Instruction-Register and so on - no indexing!

Planed next steps:

- Simulation of the communication between two or more nodes
- Small Compiler or Assembler to generate own Hex code for testing
- More testing
- Serial Link to the GA144 chip

---

gas-1.lsp Basic-Declarations, Register, Variables and Lists

reglist:
A list of symbols for the used variables and lists - for automatic assignments later

TSDS /RRS:
One list as combination of T-register S-register and Circular-Data-Stack, size 10 elements
One list as combination of R-register and Circular-Return-Stack, size 9 elements

F18cmd:
A list of F18-command strings, from command-no-zero ";" to command-no-31 "a!"
Already used in my disassembler see www.ForthSeen.de

IO-SYM:
Only the existing IO-addresses are implemented. A list of key-value elements.

GA-NODE
A list of all elements of a node - besides the B-register and the I-Register all values are initial zero.

GA144
A list of 144 nodes. The status of one complete chip.

---

gas-2.lsp Internal functions

A specialty in newLISP is the easy access to list elements via indices:

- (alist 0) returns the first element OR (alist -1) returns the last element
- (0 2 alist) returns a sublist, starting at element 0, 2 elements long
- preceding indices return sublist OR attached indices return elements

DSpush / DSpop and Rspush / RSpop
The Stacks are implemented as lists, so that the fast standard push and pop functions can be used,
Normally the first element are popped or pushed, with an attached "-1" the last.

(rd adr) / (wr adr x)
RAM, ROM and IO's are implemented as lists aka arrays. The access is done via indices .
Another specialty of the F18A chip is that RAM and ROM are partly mirrored.

The other functions are needed due to some special features of the F18A-core:

- General limitation to 18 bits (dezimal 262143)
- Limitation to 9 bits of the B-register
- Only 7bit incrementer of the A-register
- Only 7bit incrementer of the Program-Counter or P-Register
- No incrementing of P IO-address-range

---

gas-3.lsp GA144 Commands for Arithmetic, Logic and Register Manipulation

Based on the mentioned concept the implementation of the basic functions is easy.
I think the code is now self-explaining. A lot of pushing and popping and some simple arithmetic.

For some functions the ALU is used, one extra functions limits the ALU-result to 18bit and pushes it to the TSDS Datastack.

For the extended arithmetic a Carry register was needed, maybe the EXT-Flag can be left out later.

---

gas-4.lsp GA144 Commands for Memory Read and Write

As before with the help of the (rd adr) / (wr adr x) functions it's no problem to implement this commands – self explaining code.

" T into [A] " is the short form of " Register A is the address of a target, store the value of Register-T to this target "

FA18 special: The "@p" command increments the program-counter additionally.

---

gas-5.lsp GA144 Commands for Program Control

This is so far the most tricky part of this project. A lot of F18A special functions are considered:

- One 18bit instructions word can have up to 4 slots and up to 4 commands
- Besides the p-counter, there is a slot-counter
- Example P=0 aslot=0 .. aslot=1 .. aslot=2 .. aslot=3 ; P=1 aslot=0 and so on
- Program Control commands can discontinue the execution of rest-slots
- A jump command uses the rest-slots for an address-delta
- The address-delta is added to the Program-Counter
- The "unext" command is a micro-loop in one instruction word
- If the program-counter points to the IO-range it is not auto incremented
- Commands ar x-ored with 0x15555 – addresses and literal numbers not
- Some special bits in the program counter are used as flag for extended arithmetic
- Only slot0 jumps can reach the IO-range – others only RAM and ROM

Continuation: gas-5.lsp GA144 Commands for Program Control

The main-loop (explained later in gas-7.lsp) consist of three loops :

- an outer iteration loop - practically a time counter (count-variable ix)
- a program-counter loop underneath (count-variable p)
- an inner slot-counter loop (count variable aslot)

With this concept the commands for program control can alter the slot-counter and the program-counter at any time.

(get-slot-command)
Returns the command-string decoded from the IREG and depending on the slot counter aslot.

(set-jump)
Gets the delta-address depending on the slot from which the jump starts and changes the program-counter.

---

gas-6.lsp GA144 Debug and Print Functions

Another example for the power of newLISP – with a few lines you can have a debugging tool for the most functions you know from standard tools.

Example:

(set 'my-symbol-list '(P B A IREG TSDS))
Make a list of variables and registers your are interested in

(pr-csv-header-list my-symbol-list)
Prints the comma separated symbol names for the column header

(pr-csv-list my-symbol-list)
Prints the symbol values as comma separated values

(pr-csv-list my-symbol-list (< P 10))
The last function can be extended with a condition, prints then only if condition is true

(pr-csv-list my-symbol-list (and (> ix 7000) (< ix 7500))
The condition can have any complexity you need, it can be a combination

You see one other example in the following gas-7.lsp

For a first simulation I inserted the hex-dump of the practical example you found in chapter 9 of this document: http://www.forthseen.de/stuff/DB004-131030-aFUSER.pdf

gas-7.lsp GA144 The main simulation loop - The happy end - so far …

```
08: ( set 'P 0 'ix 0 )

17:( while ( < ix 1000 )
18: ( set 'IREG ( rd P ) )
19: ( incP )
20: ( set 'aslot 0 )
21:  ( while ( < aslot 4 )
22:   ( set 'xcmd-str ( append "(g" ( get-slot-command ) ")" ))
23:   ( eval-string xcmd-str )
24:   ( ++ aslot )
25:   ( pr-csv-list mysyms (< ix 500) ) ; print 500 iterations
26:  ( ++ ix )))
```

That's all you need for simulating this example:

- The program counter starts at zero – a pointer to the first RAM address (08:)
- The IREG is set to the RAM value at address zero (18:)
- P is incremented (19:)
- The slot-counter is set to 0 (20:)
- The slot0 command is decoded, the string (g@b) is generated (22:)
- The string (g@b) is evaluated – that means the function is executed (23:)
- The slot counter is incremented (24:)
- The csv-list is printed (25:)
- The iteration loop counter ix is incremented

An excerpt from the resulting output file – imported in LibreOfficeCalc :

test.txt * gas-project dmemos.de * Sat Dec 13 16:50:15 2014

xcmd-str	ix	aslot	P	A	B	IREG	TSDS	TSDS1	TSDS2	TSDS3
(g@b)	0	1	1	0	349	7674	0	0	0	0
(g@p)	1	2	2	0	349	7674	8192	0	0	0
(gand)	2	3	2	0	349	7674	0	8192	0	0
(g.)	3	4	2	0	349	7674	0	8192	0	0
(gif)	4	1	8	0	349	103432	0	8192	0	0
(gex)	5	5	0	0	349	103432	0	8192	0	0
(g@b)	6	1	1	0	349	7674	0	0	8192	0
(g@p)	7	2	2	0	349	7674	8192	0	0	8192

So that's what I planed to explain - more details on request - see Introduction.

```
; gas-7.lsp   dmemos  24.December.2014  GAS144a

( change-dir "C:/GAS144/" )
( load "gas-6.lsp" ) ;

( GAload 0 )
( set 'RAM ( 0 63 ( append hexdump1 RAM ))) ; put hexdump in RAM
( set 'P 0 'ix 0 )

( set 'write-to-file true )

( if write-to-file ( openfile "test.txt" )) ; add timestring if name ends with "-"
eg. "test-.txt"

( set 'mysyms '( xcmd-str ix aslot P A B IREG  TSDS RRS  ))
( pr-csv-header-list mysyms )

( while ( < ix 1000 )
 ( set 'IREG ( rd P ) )
 ( incP )
 ( set 'aslot 0 )
 ( while ( < aslot 4 )
   ( set 'xcmd-str ( append "(g" ( get-slot-command ) ")" ))
   ( eval-string xcmd-str )
   ( ++ aslot )
   ( pr-csv-list mysyms (< ix 500) ) ; print list to csv if condition is true
 ( ++ ix )))

( if write-to-file ( close (device )))
( println "gas-7.lsp * bye! " )
```

```
1  ; gas-6.lsp   dmemos  24.December.2014 GAS144a
2  ( load "gas-5.lsp" ) ;
3
4  ; debug-print-functions  - print content as csv ...
5
6  ( define (pr-csv-header-el x )
7    ( print (string x) ",")
8    ( if ( list? ( set 'y ( eval x )) )
9      ( dolist ( i (sequence 1 (--(length y)))) (print (string x) i "," ))))
10 ; input variable or list-symbol, out names and fieldnames  as csv
11
12 ( define (pr-csv-header-list xl)
13   ( dolist ( xe xl ) (pr-csv-header-el xe )) ( println "" ) )
14 ; inupt list of variable or list-symbols, out names and fieldnames as csv
15
16 ( define (pr-csv-el x )
17   ( if ( list? x )
18     ( dolist ( xi x ) (print xi "," ))
19     ( print x "," )))
20 ; input variable or list-symbol, out content as csv
21
22 ( define ( pr-csv-list xl condition )
23   ( if condition ( begin ( dolist ( x xl )  ( pr-csv-el ( eval x) ))( println "" ))))
24 ; input list of variables and list-symbols,  out content as csv if condition is true
25 ; example: ( pr-csv-list mysyms (and (> ix 7000) (< ix 7500)) )
26
27 ( set 'hexdump1 '(
28   0x01dfa 0x02000 0x19408 0x04b02 0x00115 0x2f455 0x04bb2 0x0015d
29   0x3bdfa 0x1ffff 0x209f2 0x1b40f 0x05bb2 0x07530 0x11400 0x05bb2
30   0x00000 0x11400 0x3a8ef 0x00100 0x26cb2 0x11406 0x134a9 0x256aa 0x24D4A ))
31 ; Example hexdump as test-input
32
33 ( define (openfile fname )
34    ( replace "-." fname ( append "-" (string(apply date-value (now))) "." ))
35    ( set 'outfile ( device ( open  fname "write" )))
36    ( println fname " * gas-project dmemos.de * " (date) ))
37 ;  close with ( close (device) )  ; print to file ; if -. add time-string
38
39 ( define ( timestr ) ( string ( apply date-value (now))))
40 ; time-string seconds since 1.1.1970  00:00:00
41
42
```

```
; gas-5.lsp   dmemos  24.December.2014 GAS144a
( load "gas-4.lsp" ) ;
; program control commands
( define ( g; )  ; ";" or return set P to return address from R-stack
  ( set 'P ( RSpop ))
  ( set 'aslot 4))

( define ( gex )   ; execute  R <> P
  ( set 'x (RSpop) )
  ( RSpush P)
  ( set 'P x )
  ( set 'aslot 4))

( define ( gname; ) ; name;
   ( set-jump ))

( define ( gname ) ;  call
   ( RSpush P )
   ( set-jump ))

( define ( gunext ) ; unext
  ( if ( = ( RRS 0) 0 )
     ( begin ( RSpop ) ( set aslot 4))
     ( begin ( ( RSpush (-- ( RSpop ))) ( set 'aslot 0 )))))

( define ( gnext) ; next
  ( if ( = ( RRS 0) 0 )
    ( begin ( RSpop ) ( set aslot 4))
     ( begin ( RSpush (-- ( RSpop )) ( set-jump )))))

( define ( gif )
  ( if ( = 0 (TSDS 0 ) ( set-jump)(aslot 4))))

( define ( g-if )
  ( if ( = 0 ( & (TSDS 0) 0x10000))( set-jump)( aslot 4)))

( define ( get-slot-command )
 ( set 'x (^ 0x15555 IREG ))
 ( case aslot
  ( 3 (F18cmd (<< ( & 7 x) 2 )))
  ( 2 (F18cmd (>> ( & 0xF8 x)3)))
  ( 1 (F18cmd (>> ( & 0x1F00 x)8)))
  ( 0 (F18cmd (>> ( & 0x3E000 x)13)))))
; eg: IREG=0x01dfa case aslot 0->"@b", 1->"@p", 2->"and", 3->"."

( define ( set-jump )
  ( case aslot
     ( 2 ( set 'P (| (^ P (& P 0x107)) (& IREG 0x07 ))))
     ( 1 ( set 'P (| (^ P (& P 0x1FF)) (& IREG 0xFF ))))
     ( 0 ( set 'P (| (^ P (& P 0x3FF)) (& IREG 0x3FF))))
     ( true ( println "error - no jump from slot3 !"  ))))
; replaces the last 3/8/10 P-bits with IREG-bits, if 3/8 set bit P8 to 0

```

```
; gas-4.lsp   dmemos 24.December.2014 GAS144a
( load "gas-3.lsp" )
; Memory Read and Write commands
( define ( g@p )  ;fetch-P , read [P] into T
  ( if ( set 'x ( rd P ))
    (begin ( DSpush x ) ( incP ))
    ( println "error!" )))

( define ( g@ ) ; fetch - read [A] to T
  ( if ( set 'x ( rd A ))
    ( begin ( DSpush x ))
    ( println "error!" )))

( define ( g@+ ) ; fetch-plus, reads [A] to T
  ( if ( set 'x ( rd A ))
    ( begin ( DSpush x ) ( incA ))
    ( println "error!" )))

( define ( g!b  ) ;  T into [B]
  ( wr B ( DSpop)))

( define ( g@b ) ; fetch-B, read [B] to T
  ( if ( set 'x ( rd B ))
    ( begin ( DSpush x ))
    ( println "error!" )))

( define ( g!  ) ;  T into [A]
  ( wr A ( DSpop)))

( define ( g!+  ) ;  T into [A], increment A
  ( wr A ( DSpop))
  ( incA ))

( define ( g!b ) ;  T into [B]
  ( wr B ( DSpop))
)

( define ( g! ) ;  T into [A]
  ( wr A ( DSpop))
)
```

```
1  ; gas-3.lsp   dmemos  24.December.2014 GAS144a
2  ( load "gas-2.lsp" )
3  ; GA144 commands for Arithmetic, Logic and Register Manipulation
4  ( define ( gdrop ) ( DSpop ))
5
6  ( define ( gdup ) ( DSpush (TSDS 0 )))
7
8  ( define ( gpop ) ( DSpush ( RSpop )))
9
10 ( define ( gover ) ( set 'ALU (DSpop)) ( set 'x (DSpop))
11    (DSpush ALU)(DSpush x ))
12
13 ( define ( ga ) ( DSpush A )    )
14
15 ( define ( g. )  ) ; nop
16
17 ( define ( gpush ) ( RSpush ( DSpop ))     )
18
19 ( define ( gb! ) ( setB ( DSpop ))   )
20
21 ( define ( ga! ) ( set 'A ( DSpop ))   )
22
23 ; DSop: operations with T and S, result in ALU
24 ; limits result to 18 bit;  to T-S-DS ; replaces T with ALU-Result
25 ; Push ALU to TSDS, limit to 18 bit
26 ( define ( ALUpush ) ( DSpush ( B18 ALU ) TSDS ))
27
28 ( define ( DSop x )
29   ( set 'ALU ( apply x  ( 0 2 TSDS )))
30   ( ALUpush ))
31
32 ( define ( gand ) ( DSop & ))
33 ( define ( gor )  ( DSop ^ ))
34
35 ; Top: operations only with T
36 ( define ( g- )
37   ( set 'ALU ( ~ ( DSpop)))
38   ( ALUpush ))
39
40 ( define ( g2* )  ; shift logical left 1 bit
41   ( set 'ALU (<< ( DSpop) 1 ))
42   ( ALUpush ))
43
44 ( define ( g2/ )  ; Two-Slash, shift-right one bit arithmetically, signed
45   ( set 'ALU ( >> (DSpop) 1))
46   ( if ( != 0 (& ALU 0x10000)) ( set 'ALU (| ALU 0x20000)))
47   ( ALUpush))
48
49 ; "+"  T = T + S ; DS > S ; Add with carry if extended
50 ( define ( g+ )
51   ( if ( = EXT 0 ) ( set 'CY 0 ))
52   ( set 'ALU ( + (DSpop) (DSpop) CY ))
53   ( if (!= 0 (& ALU 0x40000) ( set 'CY 1 )( set 'CY 0 )))
54   ( ALUpush )   CY )
55
```

```
; gas-2.lsp   dmemos  24.December.2014 GAS144a
( load "gas-1.lsp" ) ;
; some internal functions

( define ( DSpush x ) ; push to datastack ==  T-Reg,S-Reg and Datastack
  ( push x TSDS )
  ( pop TSDS -1 )
)

( define ( RSpush x )  ; push to returnstack == R-Reg and Returnstack
  ( push x RRS )
  ( pop RRS -1 )
)

( define ( DSpop )
  ( push ( TSDS 3 ) TSDS -1)
  ( pop TSDS )
)

( define ( RSpop )
  ( push ( RRS 2 ) RRS -1)
  ( pop RRS )
)

; read from address
( define ( rd adr )
  ( if ( <= adr 0x7F ) ( RAM ( & adr 0x3f ))
   ( if ( <= adr 0xFF ) ( ROM ( & adr 0x3f ))
     ( if ( lookup adr IO-SYM ) ( IO ( lookup adr IO-SYM ))
        ( println " error-address unknown:"  adr )))))

; write to address
( define ( wr adr x )
  ( if ( <= adr 0x7F ) ( setf ( RAM ( & adr 0x3f )) x )
   ( if ( <= adr 0xFF ) ( setf ( ROM ( & adr 0x3f )) x )
     ( if ( lookup adr IO-SYM ) ( setf ( IO ( lookup adr IO-SYM )) x )
        ( println " error-address unknown:"  adr )))))


( define ( B18 x ) ( & x 0x3FFFF ))  ; 18bit == 262143 max ; limitation to 18bit

( define ( setB x ) ( set 'B ( & x 0x1FF ))) ; limitation to 9 bit Register B only

( define ( incA )  ; 7-bit incrementer
  ( set 'lbits ( & A 0x7f ))
  ( set 'hbits ( ^ A lbits))
  ( set 'A ( + hbits ( & (++ lbits) 0x7f))))

( define ( incP )  ; 7-bit increment, if not IO
   ( if ( = (& P 0x100) 0 )  ; if P not in IO-range
     ( begin
     ( set 'lbits ( & P 0x7f ))
     ( set 'hbits ( ^ P lbits))
     ( set 'P ( + hbits ( & (++ lbits) 0x7f))))))

```

```
; gas-1.lsp   dmemos  24.December.2014 GAS144a

; GA144 internal Register and Memory Names
( set 'reglist '( TSDS RRS RAM ROM IO P A B ALU EXT CY IREG))

( set 'F18cmd '(
  ";" "ex" "name;" "name" "unext" "next" "if" "-if" "@p" "@+"
  "@b" "@" "!p" "!+" "!b" "!"
  "+*" "2*" "2/" "not" "+" "and" "or" "drop" "dup" "pop"
  "over" "a" "." "push" "b!" "a!" )) ; F18-commands: 0..9, A..F, 10..19, 1A..1F

 ( set 'IO-SYM  '(  (0x15D "io" 0 )
  (0x141 "data" 1 ) (0x145 "---u" 2 ) (0x175 "--l-" 3 ) (0x165 "--lu" 4 )
  (0x115 "-d--" 5 ) (0x105 "-d-u" 6 ) (0x135 "-dl-" 7 ) (0x125 "-dlu" 8 )
  (0x1D5 "r---" 9 ) (0x1C5 "r--u" 10) (0x1F5 "r-l-" 11) (0x1E5 "r-lu" 12)
  (0x195 "rd--" 13) (0x185 "rd-u" 14) (0x1B5 "rdl-" 15) (0x1A5 "rdlu" 16)))

 ( set  'GA-NODE  ( list
  ( array 10 '(0) ) ; T - S - Data-Stack
  ( array 9 '(0) )  ; R - Return-Stack
  ( array 64 '(0) ) ; RAM
  ( array 64 '(0) ) ; ROM
  ( array 17 '(0) ) ; IO
   0 0 0x15D 0 0 0 0x15555 ; registers: P A B ALU EXT CY IREG
 ))

( set 'GA144 '() )
  ( for ( x 0 143 )
   (setf ( GA-NODE 10 ) x)
   (push GA-NODE GA144 -1))

  ; Load all registers of one node
( define ( GAload x )
  ( set 'node-list ( GA144 x ))
  ( dolist ( rx reglist )
    ( set rx ( pop node-list))))

; Save all registers of one node
( define ( GAsave x )
   ( set 'node-list '() )
   ( dolist ( rx reglist )
    ( push (eval rx) node-list -1 ))
   ( setf ( GA144 x) node-list )
)
```