september 2020

RISC-V assembler notation in noForth

For RISC-V instructions see riscv-spec.pdf on https://riscv.org
or search internet for "risc-v green card".

1. Registers

In noForth assembler the registers have special names.

```
noForth
zero    R0
link    R1
rp      R2      return stack pointer
ram     R6      start of RAM section
nxt     R7      address of next routine
tos     R8      top of data stack
sp      R9      data stack pointer
ip      R10     instruction pointer
w       R11
hop     R12
day     R13
sun     R14
moon    R15
```

w, hop, day, sun and moon (R11..R15) are local scratch regeisters. NoForth uses them, but you may also use them within your code definitions. As soon as you leave the definition their value becomes uncertain.

Other registers

The registers R3..R5 and R16..R31 are not defined and not used in noForth. It is very easy to define them if you need them:

```
8005 constant R5    (hex 8000 + register number)
801F constant R31
   etc.
```

2. Assembler code

The noForth RISC-V assembler is in forth style. This means:
1. First the operands, then the instruction name
2. Spaces between operands, instead of commas

```
noForth style
tos day sun ADD     \ ADD tos,day,sun
tos day 2 ADDI      \ ADDI tos,day,2
```

3. Compressed code

Drop the 'c' from compressed instruction names, the dot remains.

```
noForth style
tos day .add        \ c.add tos,day
tos -1 .addi        \ c.addi tos,-1
```

4. Memory addressing

```
noForth style
tos day ) .lw       \ c.lw tos,day,0
tos day ) LB        \ LB tos,day,0
day sun ) SB        \ SB day,sun,0
day 4 sun x) .sw    \ c.sw day,sun,4
tos 3 day x) LB     \ LB tos,day,3
day 1 sun x) SB     \ SB day,sun,1
```

5. Decisions (branches)

```
.0=?  .0<>?          \ 1 operand
=?    <>?            \ 2 operands
>?    <EQ?
U>?   U<EQ?
```

Use these conditions before IF, WHILE, UNTIL,

```
tos .0=? IF, .. THEN,
sun moon <EQ? IF, .. ELSE, .. THEN,
BEGIN, .. tos .0<>? WHILE, .. REPEAT,
BEGIN, .. sun moon >? UNTIL,
AHEAD, .. THEN,
BEGIN, .. AGAIN,
```

6. Macros

The macro **.mov** handles .lw .sw .mw .lwsp and .swsp
It accepts source)+ and destination -), also with RP
* **Mind the order of the operands (dest src) in the store operations**

```
macro               result
sun day .mov        sun day .mw
tos sp ) .mov       tos sp ) .lw
tos sp )+ .mov      tos sp ) .lw   sp 4 .addi
sp ) tos .mov   *   tos sp ) .sw
rp -) tos .mov  *   rp -4 .addi   tos rp ) .swsp
```

The macros **BMOV** and **HMOV** function similarly, they handle LBU, SB, LHU and SH, but they cannot be used with RP

```
macro               result
tos day )+ BMOV     tos day ) LBU   day 1 .addi
day ) tos HMOV  *   tos day ) SH
```

The macro **LI** loads any 32 bit number in a register

```
macro               result
tos 1234ABCD LI     tos 1234B000 LUI   tos tos -433 ADDI
tos 500 LI          tos zero 500 ADDI
tos -3 LI           tos -3 .li
```

7. Error messages

```
MSG from ?.REG      illegible register or register not allowed
MSG from ?REG       illegible register
MSG from ?R0        R0 not allowed
MSG from ?MODUS     -) or )+ not allowed
MSG from ?RANGE.U   unsigned immediate range error
MSG from ?RANGE.S   signed immediate range error
MSG from IF,        illegible condition
MSG from THEN,      unbalanced
MSG from UNTIL,     unbalanced, or illegible condition
MSG from AGAIN,     unbalanced
```

8. Code examples

```
noForth                     result

code DROP
    tos sp )+ .mov          tos sp ) .lw
                            sp 4 .addi
    next end-code           nxt .jr

 code 2DROP ( x y -- )
    tos 4 sp x) .mov        tos 4 sp x) .lw
    sp 8 .addi              sp 8 .addi
    next end-code           nxt .jr

code DUP ( x -- x x )
    sp -) tos .mov          sp -4 .addi
                            tos sp ) .sw
    next end-code           nxt .jr

code >R ( x -- )
    rp -) tos .mov          rp -4 .addi
                            tos rp ) .swsp
    tos sp )+ .mov          tos sp ) .lw
                            sp 4 .addi
    next end-code           nxt .jr


code >DIG ( n -- ch )
    day 0A li               day A .li
    day tos U<EQ?           tos day 6 BLTU
    if, tos 7 .addi         tos 7 .addi
    then,
    day char 0 li           day zero 30 ADDI
    tos day .add            tos day .add
    next end-code           nxt .jr
```

9. noForth assembler words

```
.ADD .ADDI .AND .ANDI .JALR .JR .LI .MW .OR .SLLI .SRAI .SRLI .SUB .XOR
ADD ADDI ANDI AUIPC J LB LBU LH LHU LUI MRET ORI SB SH SLL SLT SLTI
   SLTIU SLTU SRA SRL SUB WFI XORI
DIV DIVU MUL MULH MULHSU MULHU REM REMU   CSRRC CSRRCI CSRRS CSRRSI CSRRW CSRRWI
Macros: .MOV BMOV HMOV LI NEXT   Mem.addr: ) X) -) )+
Decisions: .0<>? .0=?   <>? <EQ? =? >? U<EQ? U>?
AGAIN, AHEAD, BEGIN, ELSE, IF, REPEAT, THEN, UNTIL, WHILE,
Registers: DAY HOP IP LINK MOON NXT RAM RP SP TOS SUN W ZERO
```