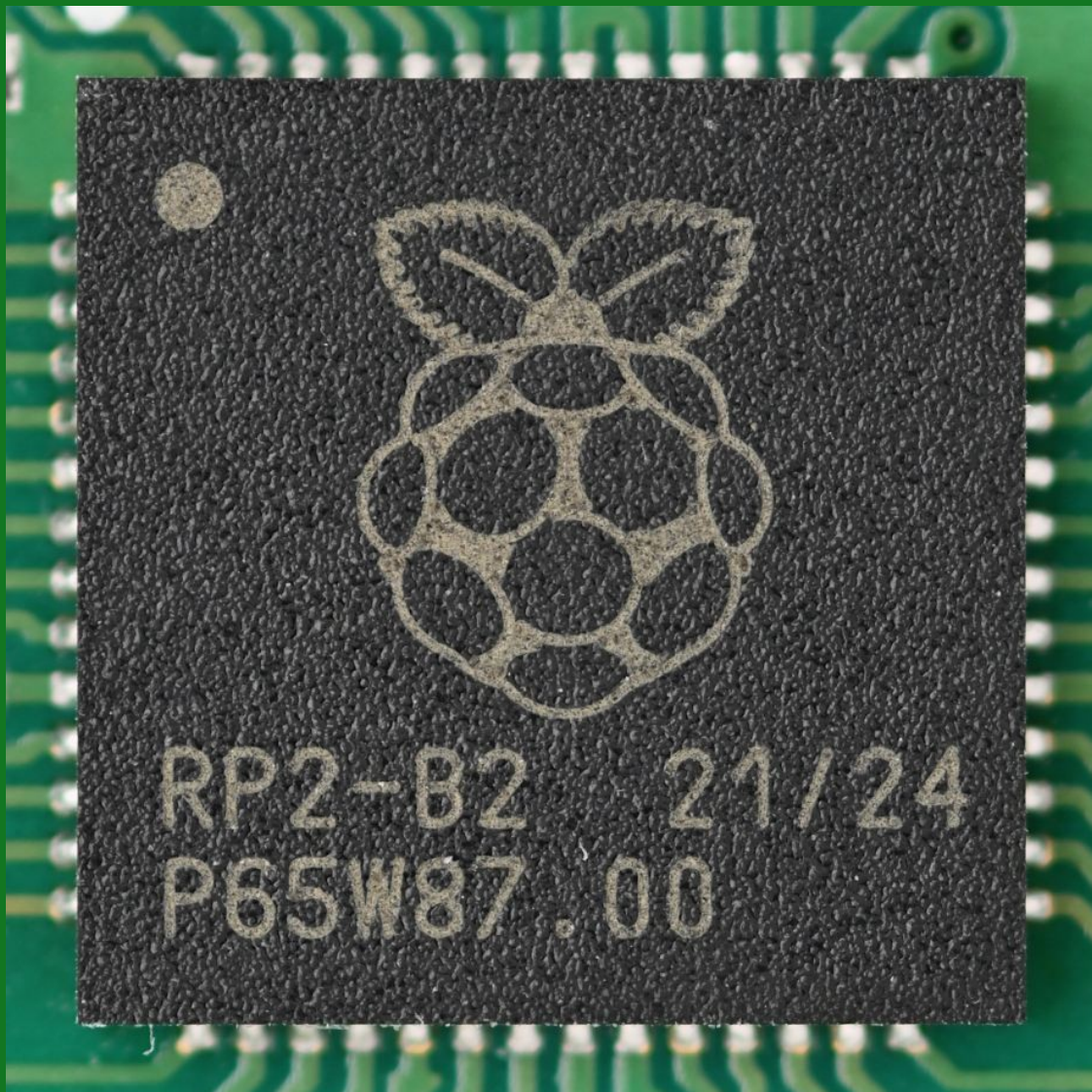


noForth T

ARM M0+ assembler



© W.J. J.J.H. & W.O.
Version 0.5

RP2040 assembler overview

Register names:

Note that! **RP** is the hardware stackpointer, **SP** is the data stackpointer!!

R0 = IP	R1 = SP	R2 = W	R3 = TOS	R4 = HOP	R5 = DAY
R6 = SUN	R7 = MOON	R8 = WW	R9 = XX	R10 = YY	R11 = ZZ
R12 = DOES	R13 = RP	R14 = LR	R15 = PC		

Two low register opcodes IP to MOON (R0 to R7):

Standard syntax

ands tos,day
neg tos,tos

is
is

Forth syntax

tos day ands,
tos tos neg,

ands,	eors,	adcs,	sbc,	rors,	tst,
neg,	cmn,	orrs,	mul,	bics,	mvns,
sxth,	sxtb,	uxth,	uxtb,	rev,	rev16,
revsh,	lsls,	lsrs,	asrs,	movs,	cmp,

Two all register opcodes (all 16 registers):

mov w,pc is w pc mov,

add, mov, cmp,

Three low register opcodes IP to MOON (R0 to R7):

ldr tos, [dsp,tos] is tos sp tos) ldr,
str day [dsp,#8] is day sp 8 x) str,

adds,	subs,	str,	strb,	strh,	ldrb,
ldrh,	ldr,	ldrsh,	ldrb,		

Two low registers and 3-bit immediate (0 to 7):

adds tos,day,#7 is tos day 7 # adds, (Literal can be 0 to 7)
rsbs tos,tos,#0 is tos tos 0 # rsbs, (Zero only!)

adds, subs, rsbs,

Two low registers and 5-bit immediate (0 to 31):

lsls day,tos,#5 is day tos 5 # lsls,

lsls,	lsrs,	asrs,	str,	strb,	strh,
ldr,	ldrb,	ldrh,			

Hardware stackpointer opcodes; register(s) & 7-bit immediate value:

add sp,#20 is rp 20 # add,

add, subs, sub,

One low register and 8-bit immediate opcode:

Standard syntax

movs tos,#127 is tos 127 # movs,
pop {tos,hop,pc} is { tos hop pc } pop,
stm w, {hop,day} is w { hop day } stm,

Forth syntax

movs,	cmp,	str, (rp)	ldr, (rp)	ldr,	add, (rp)
push,	pop,	bkpt,	udf,	svc,	adr,
stm,	ldm,	adds,	subs,		

16-bit various opcodes:

nop is nop,
bx lr is lr bx,

noop,	nop,	yield,	wfe,	wfi,	sev,
cpsie,	cpsid,	bx,	blx,		

32-bit various opcodes:

dsb is dsb,
mrs tos,control is tos control mrs,

dsb,	smb,	isb,	msr,	mrs,	bl,
------	------	------	------	------	-----

Select (virtual) addressing modes:

Standard syntax

adds tos,#0xFF is tos FF # adds,
ldrb tos,[tos,#0] is tos tos) ldrb,
ldr tos,[w,#4] is tos w 4 x) ldr,
ldr tos,[dsp,#0] adds dsp,#4 is tos sp)+ ldr,
subs dsp,dsp,#4 str tos,[dsp,#0] is tos sp -) str,
x)))+ -)

Forth syntax

Control structures:

if,	else,	then,	ahead,	
begin,	while,	repeat,	again,	until,

Conditionals:

=?	cs?	neg?	vs?	u>?	<?
>?	no				

Start & finish code definitions & using literals:

code	routine	end-code	next,	code>
align,	##	pool,	apool,	

Special register names for MSR & MRS:

apsr	iapsr	eapsr	xpsr	ipsr	epsr
iepsr	msp	psp	primask	control	

Assembler syntax examples:

Low two register opcodes:	Rd Rn and,	Rd Rn movs,
All two register opcodes:	Rd Rn add,	Rd Rn cmp,
Three low register opcodes:	Rs Rn Rm adds,	Rd Rn Rm subs,
Two low reg. & 3-bit immediate:	Rd Rn imm3 # adds,	Rd Rn imm3 # subs,
Two low reg. & 5-bit immediate:	Rd Rn imm5 # subs,	Rd Rn imm5 # ldr,
	Rd Rn imm5 # lsls,	Rd Rn imm5 # cmp,
SP & 7-bit immediate:	Rp Rp imm7 # add,	Rp Rp imm7 # subs,
Low register & 8-bit immediate	Rd imm8 # adds,	Rd imm8 # cmp,
	{ Rx .. lr } push,	{ Rx .. pc } pop,
	Rb { Rn .. Rx } ldm,	Rb { Rn .. Rx } stm,
Load & store opcodes:	Rd Rb Ro) ldr,	Rn Rb Ro) str,
	Rd Rb imm x) ldr,	Rn Rb imm x) str,
Branch	Rn bx,	
Branch & link:	Rn blx,	<address> bl,
MRS & MSR opcode:	Rd <special> mrs,	<special> Rn msr,
Zero argument opcodes:	noop, nop, yield, wfe, wfi,	
	sev, cpsie, cpsid, dsb, isb,	
	dmb,	
Control structures:	tos 0 # cmp, <? if, ... then,	\ True when negative
	tos 0 # cmp, =? no if, ... then,	\ True when not zero
	begin, ... day 10 # cmp, =? until,	\ End loop when DAY=10

First example; Forth style literal pool using LDR & LDRH :

You may define a maximum of 32 literals here! The W-register contains a pointer to the literals. As long as we do not touch the W-register these literals can be accessed. If alignment is taken into account, 16-bit values can also be read too. The word **CODE>** redirects the CFA to the real code!

```
code WORD      ( -- )
  12345678 ,    \ Literal 0
  20004000 ,    \ Literal 1
  -1 ,          \ Literal 2
code>
  tos sp -) str, \ Save TOS
  tos w 0 x) ldr, \ Literal 0 (12345678) to TOS
  hop w 4 x) ldr, \ Literal 1 (20004000) to HOP
  day w 8 x) ldr, \ Literal 2 (-1) to DAY
  ...
  next,
end-code
```

Second example; Using LDM, to read literal pool:

These constants must be fetched in one go using a LDM-opcode a maximum of 5 data cells can be read in one go in noForth! This is the most compact and fastest way to read a pool. When entering a code definition the W-register contains the address of the first literal.

Note! The registers are filled from low to high, see register names!

W = R2, TOS = R3, HOP = R4 and DAY = R5

```
code WORD      ( -- )
  12345678 ,    \ Literal 0
  20004000 ,    \ Literal 1
  -1 ,          \ Literal 2
code>
  tos sp -) str, \ Save TOS
  w { tos hop day } ldm, \ Read all three literals in one go
  ...                \ TOS = lit0, HOP = lit1, DAY = lit2
  next,
end-code
```

Alternative use, literal pool within code:

```
code WORD      ( -- )
  tos sp -) str, \ Save TOS
pool>          \ Start new pool
  87654321 ,    \ Literal 0
  10000100 ,    \ Literal 1
  true ,        \ Literal 2
then,
  tos sp -) str, \ Save TOS
  w { tos hop day } ldm, \ Read all three literals in one go
  ...                \ TOS = lit0, HOP = lit1, DAY = lit2
  next,
end-code
```

Third example; Simple literal pool:

Only single numbers can be used! The literal number is aligned & enclosed in the code word.

```
code WORD      ( -- )
  tos sp -) str,          \ Save TOS
  tos pc ) ldr, 12345678 ## \ Load 12345678 to TOS
  ...
  day pc ) ldr, -1 ##      \ Load -1 to DAY
  ...
  next,
end-code
```

This third example is expanded to this code. The word ALIGN, assembles a NOOP, when the address is not 32-bit aligned! Note that the LDR offset is increased to 4 when this was done.

```
code WORD      ( -- )
  sp sp 4 # subs,          \ Save TOS
  tos sp 0 x) str,          \ Load inline number to TOS
  tos pc 0 x) ldr,          \ Inline number
  ahead,                  \ Inline number
  12345678 ,
  then,
  ...
  day pc 4 x) ldr,          \ Load -1 to DAY
  ahead, noop, -1 , then,  \ Inline number
  ...
  ip { w } ldm,             \ The noForth NEXT routine!
  w { hop } ldm,
  pc hop mov,
end-code
```

- End -

