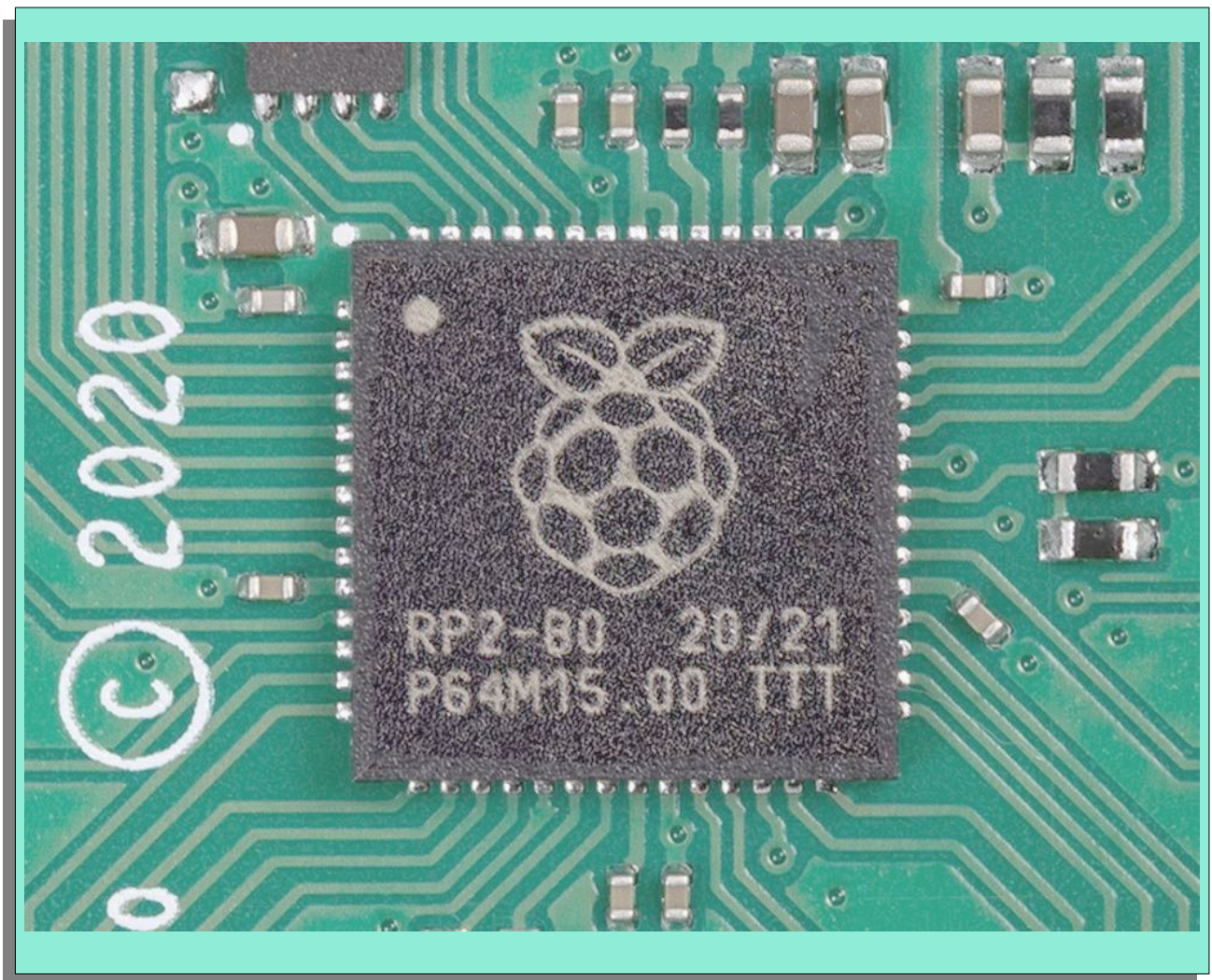


noForth T

ARMv6-M, Thumb-2 assembler



© W.J. J.J.H. & W.O.
for version 0.6

RP2040 assembler overview

More info on the RP2040 instruction set: [ARMv6-M Architecture Reference Manual](#) and the: [ARM and Thumb-2 Instruction Set Quick Reference Card](#)

1. Registers

Note that! **RP** is the hardware stackpointer, **SP** is the data stackpointer!
All scratch registers may be used in your code definitions.

noForth	RP2040	Description
IP	R0	NoForth instruction pointer
SP	R1	Data stack pointer
W	R2	Scratch register used by NEXT
TOS	R3	Top of data stack
HOP	R4	Scratch register used by NEXT
DAY	R5	Scratch register used by code words
SUN	R6	Scratch register used by code words
MOON	R7	Scratch register used by code words
WW	R8	Free register
XX	R9	Free register
YY	R10	Free register
ZZ	R11	Free register
DOES	R12	Address of DOES routine
RP	SP	Return stack pointer
LR	LR	Link register
PC	PC	Program counter

2. Forth style assembler

The noForth Thumb2 assembler is in forth style. This means:

1. First the operands, then the instruction name
2. Spaces between operands, instead of commas

noForth style	ARM style
tos day ands,	ands tos, day
tos tos neg,	neg tos, tos

3. Memory addressing

noForth style	ARM style
tos sp tos) ldr,	ldr tos, [dsp,tos]
day sp 8 x) str,	str day [dsp,#8]
tos w tos) ldhr,	ldrh tos, [w,tos]
day w 8 x) strh,	strh day [w,#8]
tos sun tos) ldrb,	ldr btos, [sun,tos]
day sun 8 x) strb,	strb day [sun,#8]
tos moon tos) ldrsh,	ldr sh tos, [moon,tos]
day moon 8 x) ldsrb,	ldrsb day [moon,#8]
w { tos day sun } ldm,	ldm w, {tos,day,sun}
hop { sun moon } stm,	stm hop {sun,moon}
{ tos hop pc } pop,	pop {tos,hop,pc}
{ tos hop lr } push,	push {tos,hop,lr}
tos w 400 x) adr,	adr tos, [w,#400]

Note that: The memory addressing may also be used relative to the program counter (PC). This is used for the ARM-style literal pools (using inline literal numbers). The last chapter shows examples of those literal pools.

4. Decisions (branches)

Testing of status flags							
=?	CS?	NEG?	VS?	U>?	<?	>?	Condition flags
NO							Invert condition flag

Examples of control structures, condition flags have been added where applicable.

=? IF, .. THEN,
U>? NO IF, .. ELSE, .. THEN,
BEGIN, .. =? NO WHILE, .. REPEAT,
BEGIN, .. >? UNTIL,
AHEAD, .. THEN,
BEGIN, .. AGAIN,

5. Opcodes that set the status flags

When an S is added to an opcode, it means that it also sets the status flags.

ADCS,	ADDs,	SUBS,	RSBS,	SBCS,	NEG,	MULS	ASRS,	ANDS,	EORS,
ORRS,	RORS,	LSLS,	LSRS,	BICS,	TST,	CMP,	CMN,	MVNS,	MOVS,

6. Opcodes that do not set the status flags

ADD,	SUB,	MOV,	REV,	REV16,	REVSH,	SXTB,	SXTH,	UXTB,	UXTH,
------	------	------	------	--------	--------	-------	-------	-------	-------

7. Opcodes ordered by arguments

7.0 Opcodes that use two low registers

ANDS,	EORS,	ADCS,	SBCS,	RORS,	TST,	NEG,	CMN,	ORRS,	MULS,
BICS,	MVNS,	SXTH,	SXTB,	UXTH,	UXTB,	REV,	REV16,	REVSH,	LSLS,
LSRS,	ASRS,	MOVS,	CMP,						

7.1 Opcodes that use two 'all' registers

ADD,	MOV,	CMP,
------	------	------

7.2 Opcodes that use three low registers

ADDs,	SUBS,	STR,	STRB,	STRH,	LDRB,	LDRH,	LDR,	LDRSB,
LDRSH,								

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

ADDs,	SUBS,	RSBS,
-------	-------	-------

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

LSLS,	LSRS,	ASRS,	STR,	STRB,	STRH,	LDR,	LDRB,	LDRH,
-------	-------	-------	------	-------	-------	------	-------	-------

7.5 Opcodes that work on the hardware stack

ADD, SUBS, SUB, LDR, STR, POP, PUSH,

Using **LDR**, to read from the hardware stack & **ADD**, on hardware stack calculations.

```
code LEAVE
  ip rp 8 x) ldr,          \ 2 - Read loop address
  ip 4 # subs,             \ 1 - Go one cell back
  ip { ip } ldm,           \ 2 - Read there the unloop address
  rp rp 0C # add,          \ 1 - Remove all loop data
  next,                   \ 6
end-code
```

7.6 Opcodes with one low register and an 8-bit number

MOVS, CMP, STR, LDR, LDR, ADD, PUSH, POP, BKPT,
UDF, SVC, ADR, STM, LDM, ADDS, SUBS,

7.7 16-bit various opcodes

NOOP, NOP, YIELD, WFE, WFI, SEV, CPSIE, CPSID, BX, BLX,

7.8 32-bit various opcodes

DSB, SMB, ISB, MSR, MRS, BL,

8. Addressing modes

#	Immediate addressing
x)	Indexed addressing with offset
)	Indexed addressing with offset zero or register
)+	Indexed addressing with auto increment
-)	Indexed addressing with auto decrement
{ }	Multiple register addressing

Usage examples

noForth style	ARM style
tos FF # adds,	adds tos,#0xFF
tos w 4 x) ldr,	ldr tos,[w,#4]
tos tos) ldrb	ldrb tos,[tos,#0]
tos sp)+ ldr,	ldm sp,{tos}
tos sp -) str,	subs sp,sp,#4 str tos,[sp,#0]
w { tos day sun } ldm,	ldm w, {tos,day,sun}

9. Start & finish code definitions & using large numbers

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

Usage examples for, **next**, **code** & **end-code** are showed in chapter 12.

The word **code> align**, **pool**, **apool** and **##** in chapter 14, and finally the word **routine** is used to write interrupt- and subroutines, it leaves the address of the routine when called.

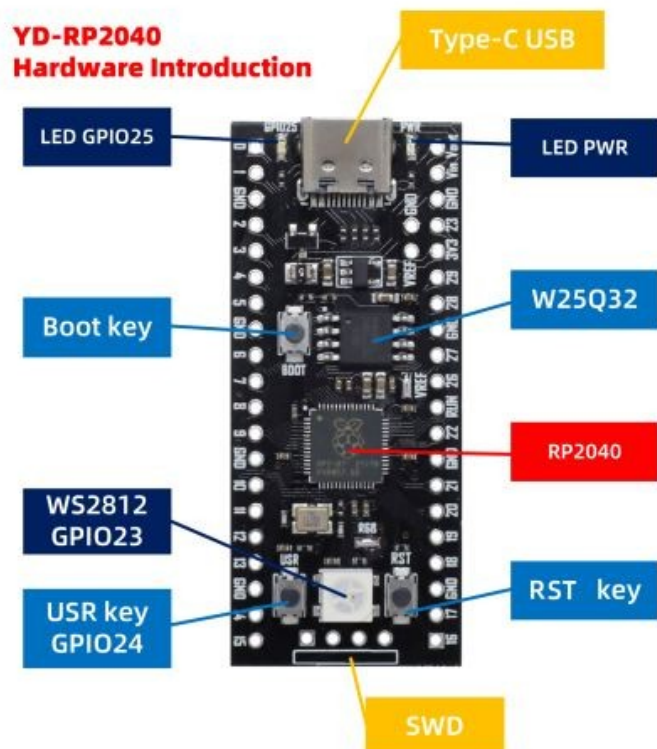
```
Routine BLOPS ( -- a )
    xx yy add,
    lr bx,
end-code
```

10. Special register names for MSR & MRS

```
apshr     iapshr    eapshr    xpsr      ipsr       epsr
iepsr     msp       psp       primask   control
```

11. Sample Pico boards

Basic Pico board



12. 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 number	Rd Rn imm3 # adds,	Rd Rn imm3 # subs,
Two low reg. & 5-bit number	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 number	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 and exchange	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,	
	tos 0 # cmp, =? no if, ... then,	
	begin, ... day 10 # cmp, =? until,	

```
code +!      ( x a -- )
  day sp )+ ldr,      \ 2 - Pop 'x' to DAY
  sun tos ) ldr,      \ 2 - Read contents of a to SUN
  day day sun adds,   \ 1 - Add contents of a to DAY
  day tos ) str,      \ 2 - Store result to address a
  tos sp )+ ldr,      \ 2 - Replace TOS
  next,              \ 6
end-code
```

```
code R>      ( -- x )
  tos sp -) str,      \ 3 - Save TOS
  { tos } pop,        \ 3 - Pop top RP to TOS
  next,              \ 6
end-code
```

13. Error messages

Msg from ?REG	Invalid register used
Msg from ?PAIR	Unbalanced control structure
Msg from ?RANGE-U	Unsigned number too large
Msg from ?DIST	Branch out of range
Msg from }	Invalid register (order) used
Msg from IF,	Unbalanced control structure
Msg from UNTIL,	Unbalanced control structure
Msg from ?SPECIAL	Invalid special register used
Msg from #)	Inline number used on wrong instruction

14.0 Forth style literal pool using LDR or 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 AWORD      ( -- )
  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
```

14.1 Using LDM, to read multiple literal pool

These constants are 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 AWORD      ( -- )
  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
```


14.2 Alternative use, literal pool embedded within code

```
code AWORD      ( -- )
  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
```

14.3 Poor mans literal pool

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

```
code AWORD      ( -- )
  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 AWORD      ( -- )
  sp sp 4 # subs,      \ Save TOS
  tos sp 0 x) str,
  tos pc 0 x) ldr,      \ Load inline number to TOS
  ahead,
  12345678 ,           \ Inline number
  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
```

More info on the RP2040 instruction set: ARMv6-M Architecture Reference Manual
and the: ARM and Thumb-2 Instruction Set Quick Reference Card
ARM site: <https://developer.arm.com/documentation/ddi0484/c/CHDCICDF>

ADC, ADD	Add with carry, add
ADR	Load program or register-relative address (short range)
AND	Logical AND
ASR	Arithmetic shift right
B	Branch
BIC	Bit Clear
BKPT	Software breakpoint
BL	Branch with Link
BLX	Branch with Link, change instruction set
BX	Branch, change instruction set
CMN, CMP	CMP Compare negative, compare
CPSID	Disable interrupts
CPSIE	Enable interrupts
DMB, DSB	DSB Data Memory Barrier, Data Synchronization Barrier
EOR	Exclusive OR
ISB	Instruction Synchronization Barrier
LDM	Load multiple registers (low eight)
LDR	Load register with word
LDRB	Load register with byte
LDRH	Load register with halfword
LDRSB	Load register with signed byte
LDRSH	Load register with signed halfword
LSL, LSR	Logical shift left, logical shift right
MOV	Move
MRS	Move from PSR to register
MSR	Move from register to PSR
MUL	Multiply
NEG	Two's complement
NOP	No operation
ORR	Logical OR
PUSH, POP	PUSH registers to stack, POP registers from stack (low eight & LR or PC)
REV	Reverse bytes in word
REV16, REVSH	Reverse bytes in halfword, reverse byte in halfword & sign extend
ROR	Rotate right register
SBC	Subtract with carry
SEV	Set event in multiprocessor system.
STM	Store multiple registers (low eight)
STR	Store register with word
STRB	Store register with byte
STRH	Store register with halfword
SUB	Subtract
SVC	Supervisor call
SXTB, SXTB	Sign extend
TST	Test
UXTB, UXTB	Unsigned extend
WFE, WFI	Wait for event, wait for interrupt
YIELD	Yield control to alternative thread