RISC-V assembly for cross compiling Forth

Using Mac/Linux/PC

Rob Probin, Oct/Nov 2024

Why RISC-V?

[Background]

- ARM licensing costs?
- Allow differentiation?
  - ARM only peripherals for ***most*** licensees
- Becoming popular?

Pronounced “Risk Five”

RISC-V

- RV32I = Base integer instruction set, 32-bit
- RV32E = Base integer instruction set (embedded), 32-bit,
  - 16 registers
- RV64I = Base integer instruction set, 64-bit
- RV64E = Base integer instruction set (embedded), 64-bit
- RV128I = Base integer instruction set, 128-bit
- RVWMO = Weak memory ordering

Common RISC-V Extensions

- C = compressed instructions (16-bit)
- M = Standard extension for integer multiplication and division
- F = Standard extension for single-precision floating-point
- B = Standard extension for bit manipulation
- Many more!!!

But **also** allows manufacturers to extend instruction set!

Registers

Width is 32, 64 or 128 bits

REGISTER	NAME	USE	SAVER
x0	zero	The constant value 0	N.A.
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	--
x4	tp	Thread pointer	--
x5-x7	t0-t2	Temporaries	Caller
x8	s0/fp	Saved register/Frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Function arguments/Return values	Caller
x12-x17	a2-a7	Function arguments	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller
f0-f7	ft0-ft7	FP Temporaries	Caller
f8-f9	fs0-fs1	FP Saved registers	Callee
f10-f11	fa0-fa1	FP Function arguments/Return values	Caller
f12-f17	fa2-fa7	FP Function arguments	Caller
f18-f27	fs2-fs11	FP Saved registers	Callee
f28-f31	ft8-ft11	R[rd] = R[rs1] + R[rs2]	Caller

X0 - X15 on embedded I32.

X8 - X15 in compressed instructions

Example: CH32V003

Smallest RISC-V from WCH

- 8 pin to 20 pin packages (TSSOP20, QFN20, SOP16, SOP8)
- 48MHz, 16K Flash, 2K RAM, 6-18 GPIO, 6-8 10-bit ADC, 1 OPA, 1 USART, 12C, SPI, 3.3/5.0, 1×16-bit advanced-control timer, 1×16-bit general-purpose timer, 2 WDOG, 1×32-bit SysTick, Multiple low-power modes: Sleep, Standby, Power on/off reset, programmable voltage detector, 1 group of 1-channel general-purpose DMA controller, 64-bit chip unique ID, QingKe RISC-V2A, 1-wire serial debug interface (SDI)
- **RV32I subset RV32E**
- **RV32EC - compressed instructions**
- **Custom XW extensions**: supports half-word and byte operation compression instructions.
    - xw.c.lbusp: Load Unsigned Byte at address relative to SP
    - xw.c.sbsp: Store Byte at address relative to SP
    - xw.c.lbu: Load Unsigned Byte
    - xw.c.sb: Store Byte
    - xw.c.lhusp: Load Unsigned Half at address relative to SP
    - xw.c.shsp: Store Half at address relative to SP
    - xw.c.lhu: Load Unsigned Half
    - xw.c.sh: Store Half
- Hardware Prologue/Epilogue (HPE), Vector Table Free (VTF), a more streamlined single-wire serial, debug interface (SDI), and support for "WFE" instructions.

Instructions

- Go to https://www.cl.cam.ac.uk/teaching/1617/ECAD+Arch/files/docs/RISCVGreenCardv8-20151013.pdf

Pseudo Instructions

MNEMONIC	NAME	DESCRIPTION	USES
beqz	Branch = zero	if(R[rs1]==0) PC=PC+{imm,1b'0}	beq
bnez	Branch ≠ zero	if(R[rs1]!=0) PC=PC+{imm,1b'0}	bne
fabs.s,fabs.d	Absolute Value	F[rd] = (F[rs1]< 0) ? –F[rs1] : F[rs1]	fsgnx
fmv.s,fmv.d	FP Move	F[rd] = F[rs1]	fsgnj
fneg.s,fneg.d	FP negate	F[rd] = –F[rs1]	fsgnjn
j	Jump	PC = {imm,1b'0}	jal
jr	Jump register	PC = R[rs1]	jalr
la	Load address	R[rd] = address	auipc
li	Load imm	R[rd] = imm	addi
mv	Move	R[rd] = R[rs1]	addi
neg	Negate	R[rd] = –R[rs1]	sub
nop	No operation	R[0] = R[0]	addi
not	Not	R[rd] = ~R[rs1]	xori
ret	Return	PC = R[1]	jalr
seqz	Set = zero	R[rd] = (R[rs1]== 0) ? 1 : 0	sltiu
snez	Set ≠ zero	R[rd] = (R[rs1]!= 0) ? 1 : 0	sltu

Sample Code

```
main:

sp_init:
        li sp, STACK

        li x10,R32_RCC_APB2PCENR
        lw x11,0(x10)
        li x7,((1<<2)|(1<<4)|(1<<5))
        or x11,x11,x7
        sw x11,0(10)
```

```
delay:
        li x6,20000000
dloop:
        addi x6,x6,-1
        bne x6,zero,dloop
        ret
```

```
0000 37110020 H.     li sp, DSTACK_INIT
     13010180
0008 37040020 H.     li x8, RSTACK_INIT
     13040478
0010 B7010020 H.     la gp, SRAM_start
```

Lots of Good information

- Good summary - Green Card - https://www.cl.cam.ac.uk/teaching/1617/ECAD+Arch/files/docs/RISCVGreenCardv8-20151013.pdf
- https://riscv.org/technical/specifications/
- https://riscv.org/
- https://lf-riscv.atlassian.net/wiki/spaces/HOME/pages/16154769/RISC-V+Technical+Specifications
- https://en.wikipedia.org/wiki/RISC-V
- https://www.eecs.yorku.ca/course_archive/2018-19/F/2021SUP/REF/RISCV-GreenCard.pdf
- https://msyksphinz-self.github.io/riscv-isadoc
- https://michaeljclark.github.io/asm.html

Assembler based development

- Tools based on this project:
  - https://github.com/cnlohr/ch32v003fun
    - Originally for a simpler (than WCH) C environment
- Install via https://github.com/cnlohr/ch32v003fun/wiki/Installation - for Mac, Linux, Windows
- Just use binutils RISC-V assembler - but via gcc to get preprocessor (specifically #define)

Assembling

```
riscv64-unknown-elf-gcc -march=rv32ec -mabi=ilp32e forth.S -o ../
output/forth.elf -Wa,-alms=../output/forth.lst -nostdlib -T ../
ch32v003.ld -g

riscv64-unknown-elf-objcopy -O binary ../output/forth.elf ../output/
forth.bin
```

Downloading Code

to CH32V003

- SWDIO (PD1), NRST, 3V3, GND

```
minichlink -a -w output/forth.bin 0x08000000 -b
```

- Also debug over this interface…
- Also use serial terminal over this interface…

Interface options

- CH32V003 has UART, I2C, SPI but we can also use via SWDIO (since wire on PD1).

Debugging

Using VSCode

- https://github.com/cnlohr/ch32v003fun/blob/master/examples/template/.vscode/launch.json
- Ubuntu:
  - sudo apt-get install gdb-multiarch
- MacOS:
  - brew install riscv64-elf-gdb
  - but doesn’t have svd_load….
- To connect to your GDBStub running, you can:
  - gdb-multiarch -ex 'target remote :2000' ./blink.elf
  - riscv64-elf-gdb -ex 'target remote :2000' ./blink.elf

SVD

Peripheral Views

- Allows you to view peripheral registers, etc.
- svd_load ch32v003fun-master/misc/CH32V003xx.svd
- https://zuendmasse.de/gdb-svd.html

Using GDB from command line

- https://cs.baylor.edu/~donahoo/tools/gdb/tutorial.html
- https://shakti.org.in/docs/RISC-V-GDB-tutorial.pdf

- Can setup to work via VSCode
- tui enable
- layout reg
- https://dev.to/irby/making-gdb-easier-the-tui-interface-15l2
- https://sourceware.org/gdb/current/onlinedocs/gdb.html/TUI-Commands.html
- https://stackoverflow.com/questions/9970636/view-both-assembly-and-c-code