NeoCore16x32: A Dual-Issue 7-Stage In-Order CPU Core for ECP5 FPGAs

Dulat Sarbassov
Independent project

February 18, 2026

Abstract

NeoCore16x32 is a synthesizable SystemVerilog CPU core with 16-bit general-purpose registers, 32-bit addressing, variable-length instructions, and a dual-issue in-order pipeline. The implemented pipeline is `IF1` → `IF2` → `IB` → `ID` → `EX` → `MEM` → `WB`, where IB is a clocked 6-entry instruction queue and part of the architectural stage count. The memory model is logically unified (von Neumann) and physically dual-ported through one 128-bit instruction-fetch port and one 32-bit data port into a single memory space. This document summarizes the ISA, microarchitecture, memory conventions, RTL organization, verification workflow, measured simulation behavior, and FPGA implementation results from the repository.

1 Introduction

NeoCore16x32 is developed as a full-stack CPU project: ISA definition, assembler/linker tooling, synthesizable RTL, and simulation/FPGA flows. This paper describes the implemented architecture directly and uses measured data from the project verification flow.

2 Architectural Summary

2.1 Programmer-visible state

NeoCore16x32 exposes:

- 16 general-purpose registers (R0–R15), each 16 bits,
- a 32-bit program counter,
- status flags Z (zero) and V (overflow).

R0 is not hardwired to zero. Software conventions may assign special roles (for example R14 as stack pointer), but hardware treats all GPRs as general-purpose.

2.2 Data model and endianness

The architecture is big-endian for both instructions and data. Native data sizes are byte (8), halfword (16), and word (32) with byte addressing over a 32-bit address space.

2.3 Top-level pipeline view

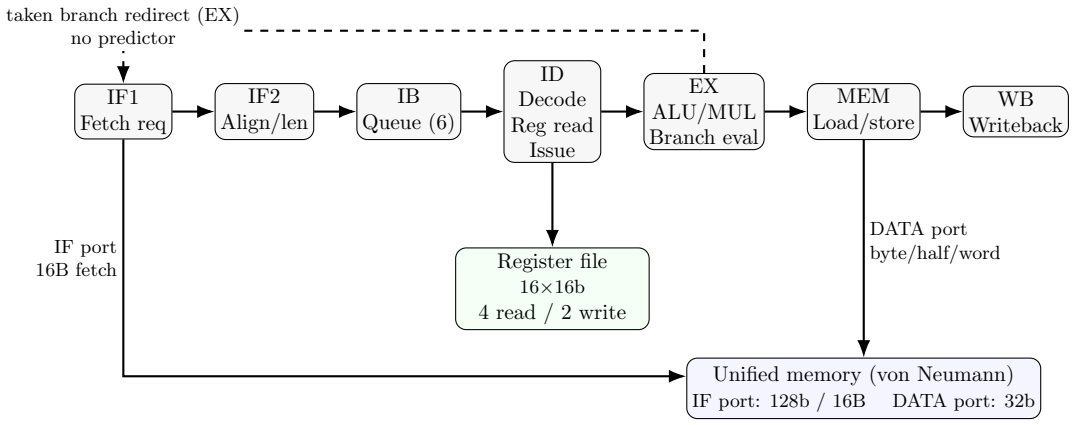

Figure 1: NeoCore16x32 block diagram. The 7-stage in-order pipeline is dual-issue; branches are resolved in EX (no predictor).

3 Instruction Set and Encoding

3.1 ISA scope

The ISA defines 26 opcodes, organized across ALU, MOV, branch/control flow, multiply, stack, subroutine, and control classes. Instruction length is variable from 2 to 9 bytes.

3.2 Common instruction prefix

All instructions begin with:

- Byte 0: **specifier** (mode/variant and length behavior),
- Byte 1: **opcode**.

Remaining bytes encode register fields and big-endian immediates/addresses.

3.3 Concrete encoding example

A representative register-format ALU instruction is:

Byte 0	Byte 1	Byte 2	Byte 3
Specifier	Opcode	`rd`	`rn`

Register IDs are carried in the low nibble of Byte 2 (`rd`) and Byte 3 (`rn`).

Example (`ADD` register mode): `ADD R1, R2` is encoded as bytes `[0x01][0x01][0x01][0x02]`, where:

- Byte 0 `0x01`: register-mode specifier,
- Byte 1 `0x01`: `ADD` opcode,
- Byte 2 `0x01`: `rd = R1`,
- Byte 3 `0x02`: `rn = R2`.

For this mode, the implemented arithmetic ordering is `rd := rn + rd`, so the example executes as `R1 := R2 + R1`.

4 Pipeline and Issue Microarchitecture

4.1 7-stage pipeline

NeoCore16x32 uses:

`IF1` → `IF2` → `IB` → `ID` → `EX` → `MEM` → `WB`

Stage	Purpose
IF1	Fetch request / PC steering
IF2	Fetch output / length predecode
IB	6-entry instruction queue
ID	Decode, register read, issue decision
EX	ALU/multiply/branch execution
MEM	Load/store path
WB	Architectural writeback

4.2 Frontend design

The fetch unit maintains a 64-byte sliding window using four 16-byte buffers (`buf_hi`, `buf_lo`, `buf_pf`, `buf_p2`) to support variable-length extraction and reduce boundary bubbles.

4.3 Dual-issue rules (current RTL)

Two instructions may issue together when all restrictions pass:

- no data-port structural conflict,
- no write-port conflict,
- no inter-pair data dependency that must serialize,
- no dual-branch pair (branch+non-branch is allowed),
- no UMULL/SMULL pairing restriction,
- no halt restriction.

Important current behavior: branch pairing is blocked only when *both* instructions are branches. If slot 0 is a taken branch and slot 1 was issued, execute-stage logic squashes slot 1 side effects.

4.4 Hazards and forwarding

Hazard handling combines:

- register-file same-cycle forwarding,
- EX operand forwarding with priority EX/MEM > MEM/WB > WB,
- automatic stalls when forwarding cannot satisfy correctness.

A load-use dependency incurs a one-cycle stall in the current timing model. Load data is produced at the end of MEM and can be forwarded to EX in the following cycle.

4.5 Control flow

There is no branch predictor. Branches resolve in EX and taken branches flush frontend/decode state and redirect the fetch PC. The nominal taken-branch recovery is 3 cycles in the current configuration: after EX resolves the branch, IF1/IF2/IB are flushed and refilled, and the first valid target-path instruction reaches the frontend/decode boundary three cycles later.

5 Unified Memory System

The core uses a unified address space with two physical ports to one memory array:

- IF port: 128-bit (16-byte) fetch,

- DATA port: 32-bit load/store path with byte/halfword/word operations.

Default testbench memory size is 64 KiB. Access semantics are big-endian for all supported widths.

6 Implementation Organization

The implementation is partitioned into stage-local modules:

- frontend (fetch and instruction buffering),
- decode/issue (instruction decode, dependency checks, pairing),
- execution (ALU, multiply, branch evaluation),
- backend (memory access, writeback, register file),
- explicit pipeline-register boundaries between stages.

7 Toolchain and Verification

7.1 Assembler/linker context

The documentation references companion tools `nc16x32-as` and `nc16x32-ld` and a custom relocatable object format (LF02). The assembler flow is described as preprocess, lex, parse/semantic passes, encode, and serialize; linker flow performs section placement and relocation.

7.2 RTL verification flow

Primary verification steps are:

- `make unit-tests`
- `make core-tests`
- `make run_any PROGRAM=mem/<file>.hex`

For profiling, the integration testbench exposes per-cycle counters with `+PROFILE`.

8 Measured Simulation Performance

Results below were collected from `make run_any ... VVPARGS=+PROFILE` in `tb/core_any_tb.sv`.

Workload	Cycles	Issued IPC	Dual-issue cycles	Taken branches / 100 issued inst.
`mem/splitmix.hex`	25	1.600	76.0%	0.0
`mem/rle.hex`	1276	1.047	43.2%	7.3
`mem/test_mixed_lengths.hex`	174587	0.729	18.5%	13.6

Additional profile highlights from `mem/test_mixed_lengths.hex`:

- Fetch IPC (valid): 1.063, IB IPC (valid): 0.987
- Issue rejects (both valid, no dual): 45153 (25.9% of cycles)
- Main reject contributors: branch restrict 32.6% of rejects, data dependency 29.2%, write conflict 28.9%
- Stall cycles: mem 6.1%, hazard 0.0% (rounded)

Reject counters are not mutually exclusive; more than one reject condition can be true in the same cycle.

Theoretical peak throughput is 2.0 IPC; the measured workloads above span 36% to 80% of that peak.

The core runs a Dhrystone-like kernel with performance around 23 MIPS, 0.825 dMIPS at 25 MHz.

8.1 Dhrystone-style kernel report

The project also tracks a Dhrystone-like run (1000 iterations) with the following measured counters:

Metric	Value
Program halt PC	`0x000000f7`
Total cycles	68009
IPC (issued)	0.912
Dual-issue cycles	19009 (28.0%)
Fetch IPC (valid)	1.338
IB IPC (valid)	1.235
Branch taken cycles	6000 (8.8%)
Issue rejects (both valid, no dual)	22000 (32.3%)
Stall cycles	mem=7001 (10.3%), hazard=1000 (1.5%)

Frontend behavior

- Fetch dual-valid cycles: 43011 (63.2%); IB dual-valid cycles: 41009 (60.3%)
- Fetch bubbles (`valid0=0`): 20000 (29.4%)

- IB blocked (`valid0 & accept=0`): 12002 (17.6%)
- Fetch `mem_req` cycles: 39004 (57.4%); fetch `mem_ack` cycles: 39003 (57.3%)
- Fetch no-HI-valid cycles: 10001 (14.7%); fetch inst0-fits stalls: 3999 (5.9%)

Issue reject breakdown

- mem conflict: 999 (1.5% of cycles, 4.5% of rejects)
- data dependency: 13000 (19.1% of cycles, 59.1% of rejects)
- write conflict: 8000 (11.8% of cycles, 36.4% of rejects)
- branch restrict: 4000 (5.9% of cycles, 18.2% of rejects)
- mul restrict: 0 (0.0% of cycles, 0.0% of rejects)
- halt restrict: 2001 (2.9% of cycles, 9.1% of rejects)

9 FPGA Implementation Results (ECP5)

Synthesis/place-and-route artifacts in this repository target ULX3S-85F (ECP5-85F, CABGA381) via Yosys/nextpnr.

From `build/nextpnr_report.json`:

- Achieved $f_{\max}$: 25.0388 MHz (constraint 25 MHz, pass)
- `DP16KD`: 64 / 208 (30%)
- `MULT18X18D`: 3 / 156 (1%)
- `TRELLIS_COMB`: 12449 / 83640 (14%)
- `TRELLIS_FF`: 2225 / 83640 (2%)
- `TRELLIS_IO`: 17 / 365 (4%)

The utilization profile is BRAM-heavy relative to LUT/FF, consistent with the wide fetch path and unified memory structure.

10 Conclusion

NeoCore16x32 demonstrates a practical dual-issue in-order core with variable-length encoding, deterministic 7-stage execution, and an FPGA-friendly implementation strategy. The design achieves useful IPC improvements from dual issue while keeping control and hazard behavior explicit and testable. The repository includes enough architecture, RTL, and profiling infrastructure to treat this core as a complete reference implementation for its ISA generation.

Reproducibility Commands

- `make unit-tests`
- `make run_any PROGRAM=mem/splitmix.hex VVPARGS=+PROFILE`
- `make run_any PROGRAM=mem/rle.hex VVPARGS=+PROFILE`
- `make run_any PROGRAM=mem/test_mixed_lengths.hex VVPARGS=+PROFILE`
- `make fpga` (for `build/nextpnr_report.json`)