RV32I RISC-V Core

Instruction Fetch Stage

rv32i_fetch.v — RTL Design Documentation

Pipeline Stage: Stage 1 of 5 | Language: Verilog RTL | ISA: RV32I

Project: Angelo Jacobo (Open-Source Reference) | Study & Learning Documentation

1. Module Overview

rv32i_fetch is the **first stage (IF)** of the classic 5-stage RISC-V pipeline. Its sole responsibility is to **present one valid, correctly-addressed instruction** to the downstream Decode stage on every cycle it is able to. Everything else — stalls, flushes, PC redirection — exists to protect that goal.

The module sits at the boundary between the **pipeline control domain** and the **instruction memory interface**. It drives a Wishbone-style strobe/acknowledge handshake (`o_stb_inst` / `i_ack_inst`) to the instruction memory and manages a single pipeline-register output (`o_ce` + `o_inst` + `o_pc`).

1.1 Role in the 5-Stage Pipeline

Stage	Module	Key Input from IF	Key Output to IF
IF (Stage 1)	rv32i_fetch.v	—	o_inst, o_pc, o_ce
ID (Stage 2)	rv32i_decoder.v	o_inst, o_ce	i_stall
EX (Stage 3)	rv32i_alu.v	—	i_alu_change_pc, i_alu_next_pc
MEM (Stage 4)	rv32i_memoryaccess.v	—	i_stall (propagated)
WB (Stage 5)	rv32i_writeback.v	—	i_writeback_change_pc, i_writeback_next_pc

1.2 High-Level Feature Summary

Five core capabilities are implemented in this module:

- **PC Management —** Maintains and advances the Program Counter; handles reset, sequential increment, and redirects from ALU (branch/jump) and Writeback (trap/MRET).
- **Instruction Fetch Handshake —** Issues o_stb_inst to instruction memory; waits for i_ack_inst; supports multi-cycle memory (no fixed-latency assumption).
- **Pipeline Bubble Generation —** On any PC change, forces o_ce = 0 for one or more cycles to flush wrongly-fetched instructions from entering Decode.
- **Stall Handling —** Freezes its own outputs when downstream stages assert i_stall. Saves the in-flight instruction and PC before the stall and restores them afterwards.
- **Flush —** An explicit i_flush input drives o_ce low immediately to invalidate any instruction currently at the IF/ID boundary.

2. Port List & Signal Descriptions

Port Name	Dir	Width	Group	Description
i_clk	in	1	System	Global clock. All sequential logic is posedge-triggered.
i_rst_n	in	1	System	Active-low asynchronous reset. Clears all pipeline registers.
o_iaddr	out	32	Mem I/F	Byte address sent to instruction memory. Always word-aligned.
o_pc	out	32	Pipeline	PC value associated with the instruction in o_inst. One-cycle delayed from o_iaddr to align with the data path.
i_inst	in	32	Mem I/F	32-bit instruction word returned by instruction memory when i_ack_inst is high.
o_inst	out	32	Pipeline	Registered instruction forwarded to the Decode stage.
o_stb_inst	out	1	Mem I/F	Wishbone-style strobe. High when the fetch stage wants a new instruction. Driven directly by internal CE register.
i_ack_inst	in	1	Mem I/F	Acknowledgment from instruction memory. Instruction on i_inst is valid when high.
i_writeback_change_pc	in	1	PC Ctrl	Asserted by WB stage to redirect PC (trap entry, MRET, or other privileged PC change).
i_writeback_next_pc	in	32	PC Ctrl	Target PC value from WB stage (trap handler address or MEPC).
i_alu_change_pc	in	1	PC Ctrl	Asserted by EX stage when a taken branch or JAL/JALR resolves.
i_alu_next_pc	in	32	PC Ctrl	Target PC value from EX stage (branch target or JALR result).
o_ce	out	1	Pipeline	Clock-enable output to Decode stage. When low, ID stage must treat its inputs as invalid (pipeline bubble).
i_stall	in	1	Pipeline	Global stall from downstream stages. When high, IF must freeze all outputs and not advance the PC.
i_flush	in	1	Pipeline	When high and not stalled, forces o_ce low at the next clock edge, discarding the current instruction.

3. Internal Signals & Registers

Signal	Type	Description
iaddr_d	wire/reg (comb.)	Next value of o_iaddr, computed in the combinational always block. Either PC+4, i_alu_next_pc, or i_writeback_next_pc.
ce	reg (seq.)	Internal fetch-enable register. High when the fetch stage is active. Drives o_stb_inst directly.
ce_d	wire (comb.)	Combinational next-state of ce. Cleared during PC-change cycles to create a bubble; otherwise follows ce.
stall_bit	wire (comb.)	OR of all stall conditions. When high, the pipeline register update is blocked. See Section 4.4 for full decode.
stall_fetch	reg (comb.)	Per-module stall flag. Set when a wait-state is required for memory or PC changes. Combined into stall_bit.
stall_q	reg (seq.)	Registered copy of the previous stall state. Used to detect the rising edge of a stall (first cycle of stall).
prev_pc	reg (seq.)	Delayed copy of o_iaddr. Provides one-cycle alignment so o_pc corresponds to the instruction on i_inst.
stalled_pc	reg (seq.)	Snapshot of prev_pc captured at the first cycle of a stall. Restored to o_pc when the stall ends.
stalled_inst	reg (seq.)	Snapshot of i_inst captured at the first cycle of a stall. Restored to o_inst when the stall ends.

4. Detailed Functional Description

4.1 PC Management

The program counter lives in `o_iaddr` and is updated every cycle that `stall_bit` is de-asserted. The combinational block computes `iaddr_d` (the next PC), and the sequential block latches it into `o_iaddr`.

Priority of PC sources (highest first):

- **Writeback redirect (`i_writeback_change_pc`) —** Traps and MRET. Writeback has architectural authority over all other stages; it wins even if ALU also requests a change.
- **ALU redirect (`i_alu_change_pc`) —** Taken branches (BEQ, BNE, BLT, BGE, BLTU, BGEU) and unconditional jumps (JAL, JALR). Resolved in the EX stage.
- **Sequential (PC + 4) —** Default when no redirect is active.

Note that `prev_pc` tracks `o_iaddr` with a one-cycle delay. This compensates for the memory round-trip: when the address has been issued, the data comes back one cycle later, so `o_pc` must report the address of that returned instruction, not the next fetch address.

4.2 Memory Handshake (Wishbone-like)

`o_stb_inst` is driven directly from `ce` — the fetch stage requests a new instruction whenever it is active. The memory responds by asserting `i_ack_inst`. Until acknowledgment arrives, `stall_bit` stays high and the pipeline register is frozen.

o_stb_inst	i_ack_inst	stall_bit contribution	Effect
0	—	High (no request)	No instruction needed this cycle; pipeline is already stalled/bubbled.
1	0	High (pending)	Waiting for memory; pipeline holds current outputs.
1	1	Low	Instruction valid on i_inst; pipeline advances normally.

4.3 Pipeline Bubble Generation

Whenever the PC must change (either source), the combinational block sets `ce_d = 0`. On the next clock edge, `ce` becomes 0, which means `o_stb_inst = 0`. The sequential block then propagates `o_ce = 0` to the Decode stage — this is the bubble. The bubble persists for exactly as many cycles as it takes to fetch the new instruction and receive `i_ack_inst`.

Important: The bubble is only inserted when the stage is **not already stalled** (`!(i_stall || stall_fetch)`). If a stall is active at the same time as a PC change, the redirect is queued implicitly because `iaddr_d` still receives the new target and it will be latched as soon as the stall clears.

4.4 Stall Decode — stall_bit

The central stall wire is the OR of four independent conditions:

```
wire stall_bit = stall_fetch           // module-level wait
               || i_stall              // downstream stall
               || (o_stb_inst && !i_ack_inst)  // memory not ready
```

```
                || !o_stb_inst;          // not even requesting
```

Condition	Meaning	Action when true
stall_fetch	Module decided it must wait (e.g., during PC change)	Hold all pipeline registers; do not advance PC.
i_stall	Downstream stage (ID/EX/MEM/WB) cannot accept new data	Freeze outputs; save current instruction in stalled_inst.
o_stb_inst && !i_ack_inst	Request issued but memory has not replied yet	Wait; same effect as above.
!o_stb_inst	Fetch stage is disabled (ce = 0 / pipeline bubble)	Nothing to deliver; hold stage.

4.5 Stall Save & Restore (stall_q, stalled_pc, stalled_inst)

When a stall begins (`stall_bit` rises for the first time, detected by `stall_bit && !stall_q`), the module captures a snapshot:

- `stalled_pc` ← `prev_pc` (the PC of the instruction currently on i_inst)
- `stalled_inst` ← `i_inst` (the raw instruction word from memory)

On the cycle after the stall ends (`stall_q` goes low), `o_pc` and `o_inst` are driven from these saved registers rather than from the live `prev_pc` and `i_inst`. This is necessary because `i_inst` may have changed (memory bus moved on) even though the pipeline was frozen.

4.6 Flush

When `i_flush` is asserted and the pipeline is not stalled (`!stall_bit`), `o_ce` is forced to 0 on the next clock edge regardless of `ce_d`. The **flush takes precedence over ce_d** but yields to a stall (flushing a stalled stage makes no sense — the stalled instruction has not yet committed).

4.7 ce Register — Sequential Always Block

A separate always block manages the internal `ce` register:

```
always @(posedge i_clk, negedge i_rst_n) begin
    if (!i_rst_n)
        ce <= 0;
    else if ((i_alu_change_pc || i_writeback_change_pc)
              && !(i_stall || stall_fetch))
        ce <= 0;   // create bubble when PC changes (and not stalled)
    else
        ce <= 1;   // fetch stage is active in all other cases
end
```

Notice `ce` never stays 0 beyond one cycle naturally — it returns to 1 on the very next clock unless another PC-change is still active. The memory handshake (`stall_bit`) keeps the pipeline frozen while the new instruction is being fetched.

5. Timing & Pipeline Waveform Diagrams

5.1 Normal Sequential Fetch (Single-Cycle Memory)

Below is the cycle-by-cycle behaviour when instruction memory always acknowledges in one cycle and no stalls or PC changes occur.

Cycle	o_iaddr	o_stb_inst	i_ack_inst	i_inst (from mem)	o_pc	o_inst	o_ce
N+0 (reset done)	0x00	1	1	INSTR_A	—	—	0→1
N+1	0x04	1	1	INSTR_B	0x00	INSTR_A	1
N+2	0x08	1	1	INSTR_C	0x04	INSTR_B	1
N+3	0x0C	1	1	INSTR_D	0x08	INSTR_C	1

5.2 Branch Taken — PC Redirect from ALU

At cycle T, the EX stage resolves a taken branch and asserts `i_alu_change_pc`. The fetch stage must:

- Set ce = 0 (bubble) so the two wrongly-fetched instructions never reach Decode.
- Load iaddr_d with i_alu_next_pc so that the target is fetched on the next cycle.

Cycle	i_alu_change_pc	i_alu_next_pc	o_iaddr (next)	ce / o_stb	o_ce (to ID)	Comment
T−1	0	—	PC+4	1	1	Normal fetch
T	1	TARGET	TARGET	0	0	Bubble inserted; target loaded
T+1	0	—	TARG+4	1	0	Memory fetch of TARGET in flight; ce back to 1
T+2	0	—	TARG+8	1	1	TARGET instruction valid at ID

5.3 Load-Use Stall (i_stall from downstream)

When ID detects a load-use hazard it asserts `i_stall`. The fetch stage freezes, saves the in-flight instruction, and waits.

Cycle	i_stall	stall_bit	stall_q	Action	o_inst	o_pc
K	0	0	0	Normal flow	INSTR_P	PC_P
K+1	1	1	0	First stall cycle — save stalled_inst	INSTR_Q	PC_Q

Cycle	i_stall	stall_bit	stall_q	Action	o_inst	o_pc
K+2	1	1	1	Stall continues — outputs frozen	INSTR_Q	PC_Q
K+3	0	0	1	Stall ends — restore from stalled regs	INSTR_Q	PC_Q
K+4	0	0	0	Normal flow resumes	INSTR_R	PC_R

6. Annotated RTL Code

The complete **`rv32i_fetch.v`** source is reproduced below with inline commentary. Each logical block is labelled to correspond to the sections above.

```
// ═══════════════════════════════════════════════════════════════════════
//  rv32i_fetch.v  —  Instruction Fetch Stage  (Stage 1 / 5)
//  RV32I 5-Stage Pipeline  |  Angelo Jacoana Open-Source Reference
// ═══════════════════════════════════════════════════════════════════════
`timescale 1ns / 1ps
`default_nettype none
`include "rv32i_header.vh"

module rv32i_fetch #(parameter PC_RESET = 32'h00_00_00_00) (
    // ── System ──────────────────────────────────────────────────────
    input  wire        i_clk, i_rst_n,
    // ── Instruction Memory Interface (Wishbone-like) ────────────────
    output reg  [31:0] o_iaddr,       // fetch address → instruction memory
    output reg  [31:0] o_pc,          // PC aligned to instruction in o_inst
    input  wire [31:0] i_inst,        // instruction word from memory
    output reg  [31:0] o_inst,        // instruction forwarded to Decode
    output wire        o_stb_inst,    // request strobe to memory
    input  wire        i_ack_inst,    // acknowledge from memory
    // ── PC Control Inputs ───────────────────────────────────────────
    input  wire        i_writeback_change_pc,  // redirect: trap / MRET
    input  wire [31:0] i_writeback_next_pc,
    input  wire        i_alu_change_pc,         // redirect: branch / jump
    input  wire [31:0] i_alu_next_pc,
    // ── Pipeline Control ────────────────────────────────────────────
    output reg         o_ce,          // clock-enable to Decode stage
    input  wire        i_stall,       // downstream stall
    input  wire        i_flush        // flush current instruction
);

    // ── Internal Registers & Wires ──────────────────────────────────
    reg  [31:0] iaddr_d;          // combinational next-PC
    reg  [31:0] prev_pc;          // one-cycle delayed o_iaddr (PC alignment)
    reg  [31:0] stalled_inst;     // snapshot of i_inst at stall entry
    reg  [31:0] stalled_pc;       // snapshot of prev_pc at stall entry
    reg         ce, ce_d;         // fetch CE: registered and combinational
    reg         stall_fetch;      // module-level stall flag (combinational)
    reg         stall_q;          // previous-cycle stall state

    // ── Section 4.4 : Stall Logic ───────────────────────────────────
    wire stall_bit = stall_fetch
                   || i_stall
                   || (o_stb_inst && !i_ack_inst)
                   || !o_stb_inst;

    // o_stb_inst = ce : request memory whenever the stage is active
    assign o_stb_inst = ce;

    // ── Section 4.7 : CE Register ───────────────────────────────────
    always @(posedge i_clk, negedge i_rst_n) begin
        if (!i_rst_n)
            ce <= 0;
        else if ((i_alu_change_pc || i_writeback_change_pc)
                  && !(i_stall || stall_fetch))
            ce <= 0;   // bubble: PC is changing, invalidate next fetch
        else
            ce <= 1;   // fetch stage is enabled
    end
```

```
// ── Sections 4.1 / 4.3 / 4.5 / 4.6 : Main Sequential Block ──────────
always @(posedge i_clk, negedge i_rst_n) begin
    if (!i_rst_n) begin
        o_ce         <= 0;
        o_iaddr      <= PC_RESET;
        prev_pc      <= PC_RESET;
        stalled_inst <= 0;
        o_pc         <= 0;
    end
    else begin
        // ── Pipeline register update ──────────────────────────────
        // Advance when:
        //   (a) stage enabled and not stalled, OR
        //   (b) stage enabled, stalled, but output is currently 0
        //       (first valid instruction after reset), OR
        //   (c) writeback forces a PC change (trap redirect)
        if ((ce && !stall_bit)
            || (stall_bit && !o_ce && ce)
            || i_writeback_change_pc) begin
            o_iaddr <= iaddr_d;
            // restore saved values if returning from stall
            o_pc   <= stall_q ? stalled_pc   : prev_pc;
            o_inst <= stall_q ? stalled_inst : i_inst;
        end

        // ── Clock-enable output to Decode ─────────────────────────
        if (i_flush && !stall_bit)
            o_ce <= 0;           // Section 4.6: flush wins
        else if (!stall_bit)
            o_ce <= ce_d;        // normal: follow combinational next
        else if (stall_bit && !i_stall)
            o_ce <= 0;           // fetch stalled but ID is free → bubble

        // ── Stall edge detection & snapshot ───────────────────────
        stall_q <= i_stall || stall_fetch;
        if (stall_bit && !stall_q) begin
            stalled_pc   <= prev_pc;   // save before bus moves on
            stalled_inst <= i_inst;
        end

        // ── PC alignment delay ────────────────────────────────────
        prev_pc <= o_iaddr;
    end
end

// ── Section 4.1 / 4.3 : Combinational PC & CE Logic ─────────────────
always @* begin
    iaddr_d     = 0;
    ce_d        = 0;
    stall_fetch = i_stall;   // propagate downstream stall

    if (i_writeback_change_pc) begin      // Priority 1: trap / MRET
        iaddr_d = i_writeback_next_pc;
        ce_d    = 0;          // bubble
    end
    else if (i_alu_change_pc) begin        // Priority 2: branch / jump
        iaddr_d = i_alu_next_pc;
        ce_d    = 0;          // bubble
    end
    else begin                             // Priority 3: sequential
        iaddr_d = o_iaddr + 32'd4;
        ce_d    = ce;
    end
end
```

```
endmodule
```

7. Key Design Decisions & Observations

7.1 Why is ce separate from o_ce?

`ce` is the **internal enable** that drives `o_stb_inst`. `o_ce` is the **pipeline clock-enable** forwarded to the Decode stage. They differ because the Decode stage needs to see the bubble one clock edge later than when the fetch stage itself stops requesting. Separating them lets the designer control both edges independently.

7.2 Writeback priority over ALU

In the combinational block, `i_writeback_change_pc` is tested before `i_alu_change_pc`. This reflects architectural priority: a trap or MRET must always win against a speculative branch result. Both cannot legally be high simultaneously in a correct implementation, but the priority encoding protects against unexpected corner cases during bring-up.

7.3 stall_fetch initialized to i_stall

The combinational block sets `stall_fetch = i_stall` as the base case before any other logic. This ensures `stall_fetch` is never X (undefined) and that a downstream stall is always reflected even if none of the PC-change branches is taken.

7.4 The third update condition in the sequential block

The condition `(stall_bit && !o_ce && ce)` handles the corner case right after reset: `o_ce` is 0 (no instruction has yet been delivered to Decode) and `ce` is 1 (fetch is active). Without this arm, the very first instruction would be lost because `stall_bit` could be high (memory not yet acked) while `o_ce` is still 0.

7.5 No compressed instruction (RVC) support

The PC is always incremented by exactly 4 (`o_iaddr + 32'd4`). This design is RV32I-only. Adding RVC (C extension) would require a half-word alignment check and a two-byte increment for 16-bit instructions, plus a 32-bit instruction assembler across fetch boundaries.

8. Testbench Guidance

Since you are writing testbenches for each module, here are the essential test scenarios for rv32i_fetch:

Test Case	Stimulus	Expected Observable
Reset behaviour	Assert i_rst_n = 0 for 2 cycles then release	o_iaddr = PC_RESET, o_ce = 0, o_stb_inst = 0 during reset.
Sequential fetch	i_ack_inst = 1 every cycle, no stalls	o_iaddr increments by 4 each cycle; o_pc = o_iaddr delayed 1; o_ce = 1.
Memory wait-state	Hold i_ack_inst = 0 for 3 cycles then assert	o_iaddr freezes; o_inst / o_pc frozen; o_ce = 0. Advance after ack.
Branch taken	Assert i_alu_change_pc with target for 1 cycle	o_ce = 0 (bubble) for 1–2 cycles; o_iaddr jumps to target.
Trap (writeback redirect)	Assert i_writeback_change_pc with handler address	Same as branch but handler address used; writeback wins if both asserted.
Downstream stall	Assert i_stall for 3 cycles during normal fetch	o_inst and o_pc frozen; on release, same instruction presented.
Flush	Assert i_flush for 1 cycle	o_ce = 0 next cycle; instruction discarded.
Stall + branch overlap	Assert i_stall while i_alu_change_pc is also high	Branch target loaded but no bubble until stall releases.

Tip: Use a simple memory model in the testbench that returns `32'h0000_0013` (the `NOP` instruction, ADDI x0, x0, 0) as the default instruction. This gives you a legal, executable word without needing a real instruction ROM during unit testing.

9. Glossary

Term	Definition
Pipeline Bubble	A cycle in which no valid instruction is presented to the Decode stage (o_ce = 0). Used to flush wrongly-fetched instructions after a PC redirect.
PC Redirect	Any event that changes the Program Counter to a non-sequential address. Sources: taken branch (ALU), JAL/JALR (ALU), trap entry, MRET (Writeback).
Stall	A condition in which one or more pipeline stages freeze their outputs for one or more cycles. Can be caused by memory latency, load-use hazards, or structural hazards.
Flush	Explicit invalidation of the instruction currently in the fetch stage, forcing o_ce = 0 without necessarily changing the PC.
Clock Enable (CE)	A signal that gates the update of pipeline registers. When CE = 0, the register holds its previous value (equivalent to inserting a NOP from the next stage's perspective).
Strobe (STB)	In Wishbone terminology, a signal from master to slave indicating a valid transaction is being requested this cycle.
Acknowledge (ACK)	In Wishbone terminology, a signal from slave to master indicating the transaction has completed and data is valid.
stall_q	One-cycle delayed version of the stall condition used to detect the first cycle of a new stall (rising edge detection).
prev_pc	A one-cycle delayed copy of o_iaddr that aligns the PC value with the instruction data coming back from memory.
RV32I	The base integer instruction set for 32-bit RISC-V processors. Includes 47 instructions across R, I, S, B, U, and J encoding formats.