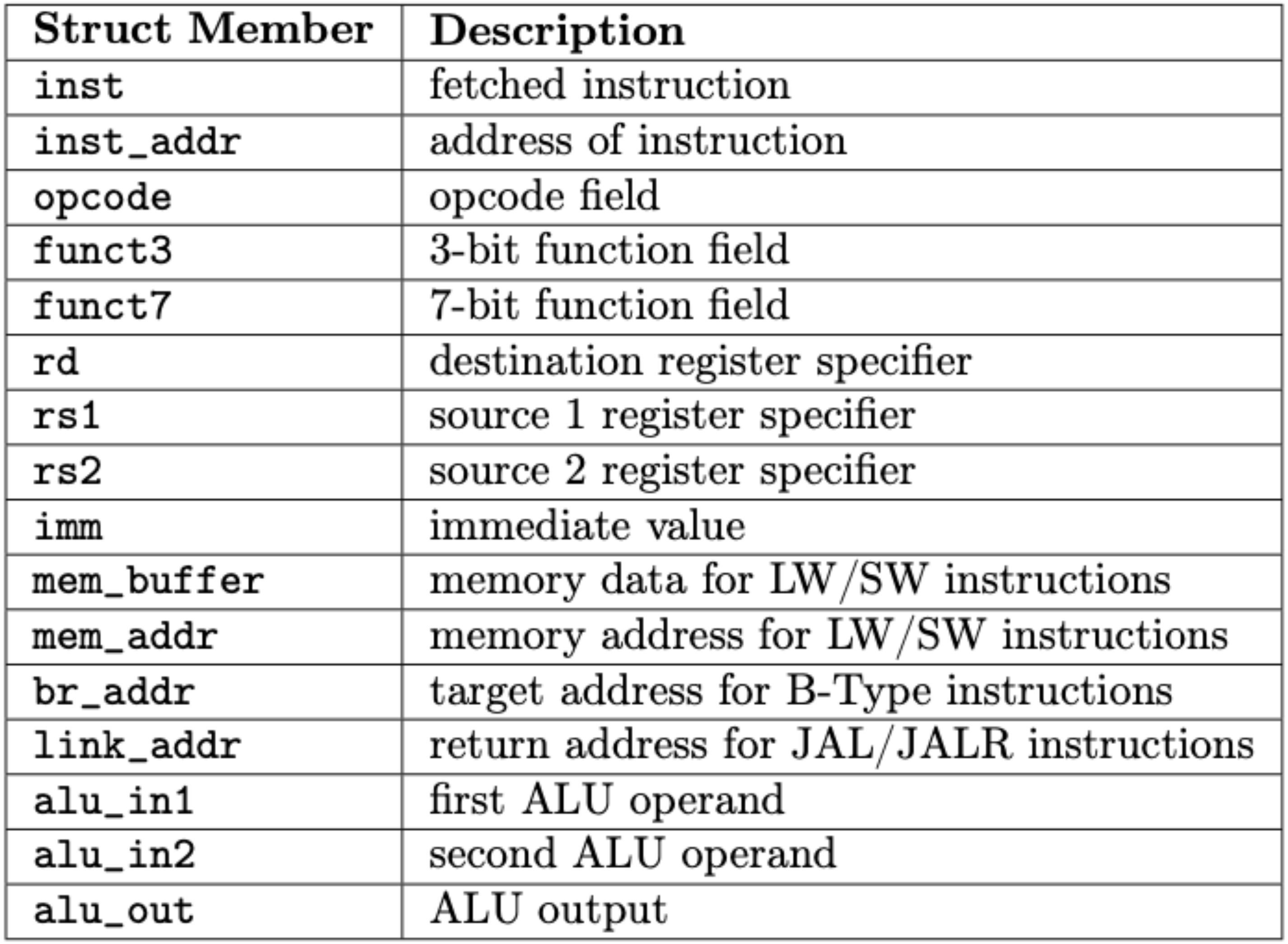
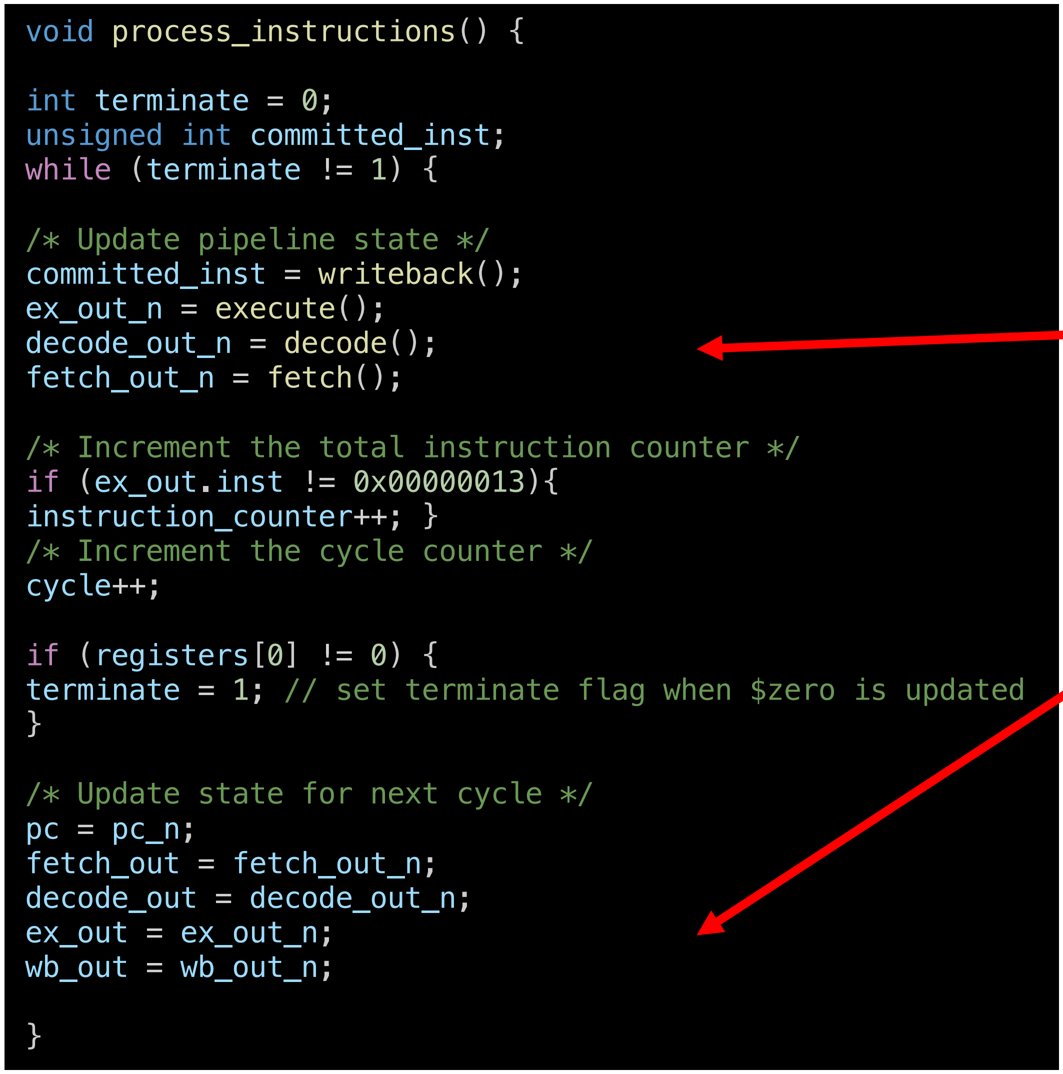
PA0: 4-Stage Pipelining



Struct Member	Description
`inst`	fetched instruction
`inst_addr`	address of instruction
`opcode`	opcode field
`funct3`	3-bit function field
`funct7`	7-bit function field
`rd`	destination register specifier
`rs1`	source 1 register specifier
`rs2`	source 2 register specifier
`imm`	immediate value
`mem_buffer`	memory data for LW/SW instructions
`mem_addr`	memory address for LW/SW instructions
`br_addr`	target address for B-Type instructions
`link_addr`	return address for JAL/JALR instructions
`alu_in1`	first ALU operand
`alu_in2`	second ALU operand
`alu_out`	ALU output

- PA0 only supports ADDI instruction type
- Does not support handling of instruction dependencies on data and control flow
- Does not support structural hazards

process_instruction in sim_core.c



```
void process_instructions() {

int terminate = 0;
unsigned int committed_inst;
while (terminate != 1) {

/* Update pipeline state */
committed_inst = writeback();
ex_out_n = execute();
decode_out_n = decode();
fetch_out_n = fetch();

/* Increment the total instruction counter */
if (ex_out.inst != 0x00000013){
instruction_counter++; }
/* Increment the cycle counter */
cycle++;

if (registers[0] != 0) {
terminate = 1; // set terminate flag when $zero is updated
}

/* Update state for next cycle */
pc = pc_n;
fetch_out = fetch_out_n;
decode_out = decode_out_n;
ex_out = ex_out_n;
wb_out = wb_out_n;

}
```

- while loop runs until terminate flag is asserted
- Pipeline stages are updated in each cycle in reverse order
- Once all *_n state structs are updated for all pipeline stages, the state structs are updated

PA0: Fetch stage

```c
/**
 * Fetch stage implementation.
 */
struct State fetch() {

    fetch_out_n.inst = memory[pc / 4];
    fetch_out_n.inst_addr = pc;
    advance_pc(fetch_out_n.inst_addr + 4);

    //Return the instruction
    return fetch_out_n;
}
```

- Fetch the instruction from memory[pc/4]
- Advance PC by +4
- Return the fetch_out_n state struct, which is passed to the Decode stage in the next cycle
  - See process_instructions function in sim_core.c for how the pipeline stages are processed across cycles

PA0: Decode stage

```
struct State decode() {
    // read the fetch_out state and start processing decode functionality
    decode_out_n = fetch_out;
    decode_fields(&decode_out_n);
    decode_out_n.alu_in1 = registers[decode_out_n.rs1];
    decode_out_n.alu_in2 = decode_out_n.imm;
    return decode_out_n;
}
```

- In a given cycle, the current state of fetch_out state struct is available for the decode stage to process
- decode_fields function in decode_fields.h the decoding of the RISC-V instructions to populate
  - opcode, funct3, funct7 fields
  - rd, rs1, rs2 register specifiers
  - imm field for the appropriate instruction type… follow the switch-case statement to see an example of decoding
  - alu_in1 and alu_in2 are specific for the ADDI instruction that is supported in PA0. For PA1, this logic will need to take into account the appropriate instruction type to populate these fields
- Return the decode_out_n state struct, which is passed to the Execute stage in the next cycle
  - See process_instructions function in sim_core.c for how the pipeline stages are processed across cycles

PA0: Execute stage

```
struct State execute() {
    // read the decode_out state and start processing execute stage's functionality
    ex_out_n = decode_out;
    ex_out_n.alu_out = ex_out_n.alu_in1 + ex_out_n.alu_in2;
    return ex_out_n;
}
```

- In a given cycle, the current state of decode_out state struct is available for the execute stage to process
- alu_in1 and alu_in2 fields from the state struct that were populated in the decode stage are operated on using the ALU and the alu_out state filed is populated
- Return the ex_out_n state struct, which is passed to the writeback stage in the next cycle
  – See process_instructions function in sim_core.c for how the pipeline stages are processed across cycles

PA0: Writeback stage

```
unsigned int writeback() {

    wb_out_n = ex_out;
    registers[wb_out_n.rd] = wb_out_n.alu_out;
    return wb_out_n.inst;
}
```

- In a given cycle, the current state of ex_out state struct is available for the writeback stage to process
- alu_out state filed is used to write the ADDI output in the register file using the rd destination register from the state
- Return the wb_out_n.inst state field, which represents the commited_inst in the sim_core.c