

Chapter 1

CENTRAL PROCESSING UNIT

This section describes the hardware description of a basic Central Processing Unit (CPU) that operates with Reduced Instruction Set Computer (RISC) instruction set.

The section describes the fixed-point instruction types classified according to their interaction with fixed-point Arithmetic Logic Unit (ALU), Program Counter (PC) and data memory.

A dedicated hardware, called data-path, is designed to execute each RISC instruction. Instruction-specific data-paths are combined in a single platform to execute a variety of user programs.

RISC INSTRUCTION FORMATS

In a CPU, all instructions start with an Operation Code (OPC) field which instructs the processor what to do with a particular instruction.

OPC field is followed by one or more operand fields, which either correspond to register addresses in the Register File (RF) or immediate values used by the CPU to process data.

There are three types of instructions in a RISC CPU: register-to-register type, immediate type and jump type.

Register-to-register instruction, whose format is shown below, fetches the contents of the first and second source registers, Reg [RS1] and Reg [RS2], from the RF, combines them according to the OPC, and writes the result back to the RF as the contents of the destination register, Reg [RD], as described below by its operational equation.

OPC RS1, RS2, RD

Reg [RS1] (OPC) Reg [RS2] → Reg [RD]

The immediate type instruction combines the contents of the source register, Reg [RS], with a sign-extended immediate value according to the OPC, and writes the result back to the RF as the contents of the destination register, Reg [RD]. The format of this type is shown below along with its operational equation.

OPC RS, RD, Immediate Value

Reg [RS] (OPC) Immediate Value \rightarrow Reg [RD]

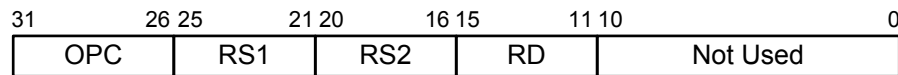
The jump type instruction uses only immediate field as shown below and processes this data according to its OPC. This type is commonly used to modify the Program Counter (PC) of the instruction memory.

OPC Immediate Value

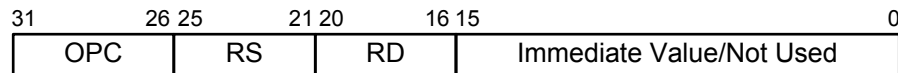
Immediate Value → PC

All three types of instructions are mapped onto the 32-bit wide instruction memory as shown below. In this figure, the numbers on top each field format correspond to the bit positions that define the borders of OPC or a particular operand field.

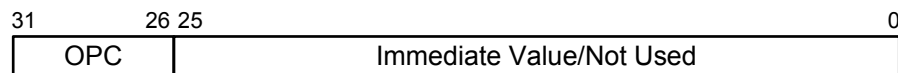
Register-to-Register Type



Immediate Type



Jump Type



REGISTER-TO-REGISTER TYPE ALU INSTRUCTIONS

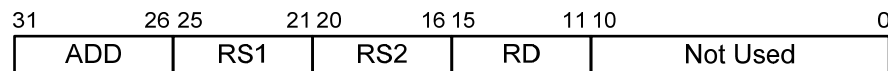
Register-to-register type ALU instructions operate between the functional units of the ALU and the RF.

The Add (ADD) instruction, whose format and operational equation shown below, fetches data from the source addresses RS1 and RS2 in the RF, adds them and returns the result to the address RD. Reg [RS1], Reg [RS2] and Reg [RD] in the operational equation refer to the 32-bit data values at RF addresses, RS1, RS2 and RD, respectively.

The field format of this instruction in the instruction memory is shown below.

ADD RS1, RS2, RD

Reg [RS1] + Reg [RS2] → Reg [RD]

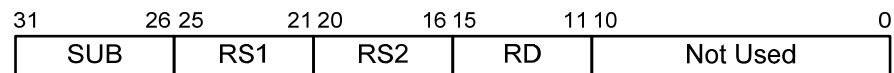


Similar to ADD instruction, the Subtract (SUB) instruction subtracts the 32-bit data at RS1 from the data at RS2 and returns the result to RD.

This instruction's format and operational equation are shown below.

SUB RS1, RS2, RD

Reg [RS1] - Reg [RS2] → Reg [RD]

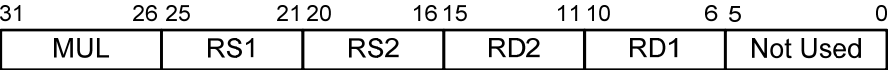


Since a 32-bit multiplication generates a 64-bit result, the fixed-point Multiply (MUL) instruction multiplies the contents of RS1 and RS2 and writes the less significant 32 bits of the result into RD1 and the more significant bits into RD2 as shown below.

The field format of this instruction is shown below.

MUL RS1, RS2, RD1, RD2

Reg [RS1] * Reg [RS2] → {Reg [RD2], Reg [RD1]}



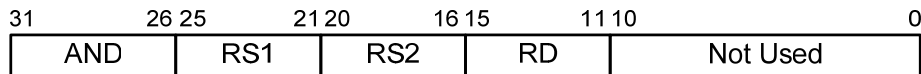
Logical operations also utilize the functional units in the ALU.

The And (AND) instruction, whose format is shown below, bitwise-ANDs the contents of RS1 and RS2 and returns the result to RD as suggested by its operational equation in which the “&” sign indicates the AND operation.

The field format of this instruction is also shown below.

AND RS1, RS2, RD

Reg [RS1] & Reg [RS2] → Reg [RD]



Similar to the And instruction, the Or (OR), the exclusive Or (XOR) and their complemented variants, the Nand (NAND), the Nor (NOR) and the Exclusive Nor (XNOR), instructions also follow the same instruction format as shown below.

OR RS1, RS2, RD
Reg [RS1] | Reg [RS2] → Reg [RD]

XOR RS1, RS2, RD
Reg [RS1] ^ Reg [RS2] → Reg [RD]

NAND RS1, RS2, RD
Reg [RS1] ~& Reg [RS2] → Reg [RD]

NOR RS1, RS2, RD
Reg [RS1] ~| Reg [RS2] → Reg [RD]

XNOR RS1, RS2, RD
Reg [RS1] ~^ Reg [RS2] → Reg [RD]

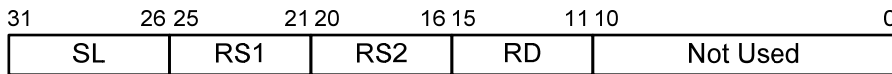
Another important set of register-to-register instructions that use the functional units of the ALU are the shift operations.

Shift Left (SL) instruction shifts the contents of RS1 to the left by the amount stored at the address RS2, and returns the result to RD. The format and the operational equation of this instruction are shown below.

The “<<” sign in the equation indicates left-shift operation. This instruction’s field format is similar to the previous register-to-register type instructions and is shown below.

SL RS1, RS2, RD

Reg [RS1] << Reg [RS2] → Reg [RD]

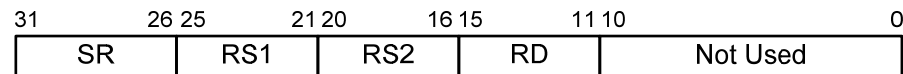


Shift Right (SR) instruction and its operational equation is similar to the SL instruction except the contents of RS1 is shifted right by the amount in RS2.

The “>>” sign in the equation corresponds to SR operation.

SR RS1, RS2, RD

Reg [RS1] >> Reg [RS2] → Reg [RD]



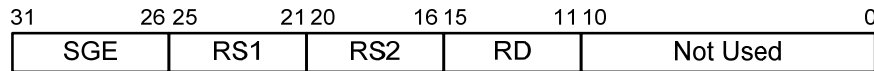
The next category of instructions is set instructions which are used in decision-making situations where two source register values are compared with each other prior to branching off the program to a different location.

The SGE instruction and its operational equation shown below sets the contents of RD to 1 if the contents of RS1 are found to be Greater than or Equal (SGE) to the contents of RS2.

If the comparison fails, then the contents of RD are set to 0. The field format of this instruction is given in Fig. 6.

SGE RS1, RS2, RD

If Reg [RS1] \geq Reg [RS2] then 1 \rightarrow Reg [RD] else 0 \rightarrow Reg [RD]



Similar to SGE, there are other conditional set instructions where the contents of two source registers are compared in a variety of different conditions before the destination register is set.

In this list, SGT, SLE, SLT, SEQ and SNE correspond to Set Greater Than, Set Less than or Equal, Set Less Than, Set Equal and Set Not Equal, respectively.

SGT RS1, RS2, RD

If Reg [RS1] > Reg [RS2] then 1 → Reg [RD] else 0 → Reg [RD]

SLE RS1, RS2, RD

If Reg [RS1] ≤ Reg [RS2] then 1 → Reg [RD] else 0 → Reg [RD]

SLT RS1, RS2, RD

If Reg [RS1] < Reg [RS2] then 1 → Reg [RD] else 0 → Reg [RD]

SEQ RS1, RS2, RD

If Reg [RS1] = Reg [RS2] then 1 → Reg [RD] else 0 → Reg [RD]

SNE RS1, RS2, RD

If Reg [RS1] ≠ Reg [RS2] then 1 → Reg [RD] else 0 → Reg [RD]

FIXED-POINT IMMEDIATE TYPE ALU INSTRUCTIONS

Immediate ALU instructions accommodate user data to be included in the instruction.

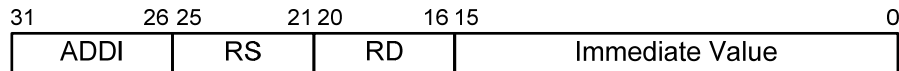
However, these instructions still refer to the RF to fetch part the other source operand to be combined with the user data.

The Add Immediate instruction (ADDI) adds the contents of RS with the user-supplied 16-bit immediate value and returns the result to RD in the RF.

The instruction and its operational equation below show this process. The field format of this instruction in the instruction memory is also given below.

ADDI RS, RD, Imm Value

Reg [RS] + Immediate Value → Reg [RD]



Subtract immediate instruction (SUBI) behaves similarly to ADDI, but subtracts the immediate value from the contents of RS and returns the result to RD as illustrated below.

SUBI RS, RD, Imm Value

Reg [RS] - Immediate Value → Reg [RD]

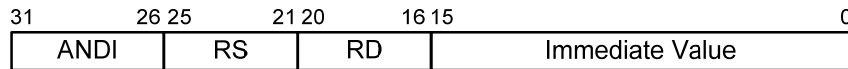
There are also immediate logical instructions that operate on the user supplied data.

AND immediate (ANDI) instruction, for example, bitwise ANDs the contents of RS with the immediate value and returns the result to RD as shown below.

The field format of this instruction is given below.

ANDI RS, RD, Imm Value

Reg [RS] & Immediate Value → Reg [RD]



Similar to ANDI, the other logical operations operate on an immediate value.

The instructions ORI, XORI, NANDI, NORI, XNORI and their operational equations are listed below.

ORI RS, RD, Imm Value
Reg [RS] | Immediate Value → Reg [RD]

XORI RS, RD, Imm Value
Reg [RS] ^ Immediate Value → Reg [RD]

NANDI RS, RD, Imm Value
Reg [RS] ~& Immediate Value → Reg [RD]

NORI RS, RD, Imm Value
Reg [RS] ~| Immediate Value → Reg [RD]

XNORI RS, RD, Imm Value
Reg [RS] ~^ Immediate Value → Reg [RD]

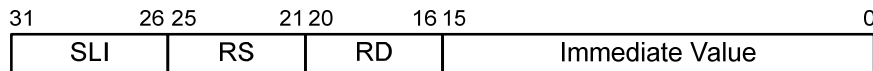
The immediate Shift Left (SLI) and Shift Right (SRI) instructions use the same functional units in the ALU as SL and SR instructions.

SLI instruction fetches the contents of RS, shifts left by an immediate value and stores the result in RD as indicated by its operational equation below.

This instruction's field format is shown below.

SLI RS, RD, Imm Value

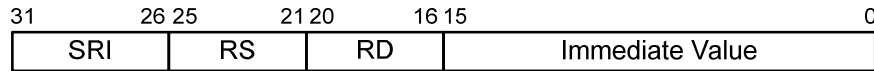
Reg [RS] << Immediate Value → Reg [RD]



Similar to SLI instruction, SRI instruction shifts the contents of RS by an immediate value and stores the result in RD as shown below.

SRI RS, RD, Imm Value

Reg [RS] >> Immediate Value → Reg [RD]

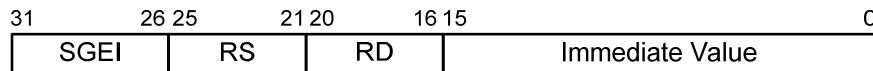


Immediate set and reset instructions compare the contents of RS with an immediate value before setting or resetting the register RD.

Set Greater than or Equal Immediate (SGEI) instruction sets the contents of RD if it finds the contents of RS is greater than or equal to the immediate value. The instruction, its operational equation and field format are shown below.

SGEI RS, RD, Imm Value

If Reg [RS] \geq Immediate Value \Rightarrow 1 \rightarrow Reg [RD]



Similarly, the instructions SGTI, SLEI, SLTI, SEQI and SNEI and their operational equations are given below.

SGTI RS, RD, Imm Value

If Reg [RS] > Immediate Value then 1 → Reg [RD] else 0 → Reg [RD]

SLEI RS, RD, Imm Value

If Reg [RS] ≤ Immediate Value then 1 → Reg [RD] else 0 → Reg [RD]

SLTI RS, RD, Imm Value

If Reg [RS] < Immediate Value then 1 → Reg [RD] else 0 → Reg [RD]

SEQI RS, RD, Imm Value

If Reg [RS] = Immediate Value then 1 → Reg [RD] else 0 → Reg [RD]

SNEI RS, RD, Imm Value

If Reg [RS] ≠ Immediate Value then 1 → Reg [RD] else 0 → Reg [RD]

DATA MOVEMENT INSTRUCTIONS

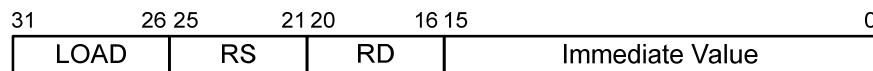
The instruction that moves data from a data memory location to a register is the Load (LOAD) instruction.

As the instruction and its operational equation below state, this instruction adds the contents of RS to an immediate value to form a memory address; it then uses this address to download data from the data memory to RD.

The field format of this instruction in the instruction memory is given below.

LOAD RS, RD, Imm Value

mem [Reg [RS] + Immediate Value] → Reg [RD]

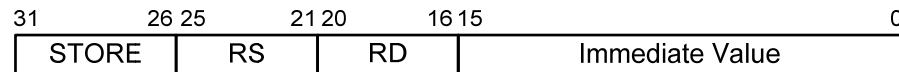


The Store (STORE) instruction shown below performs the opposite of the Load instruction.

This instruction uses the contents of RD and an immediate value to form a data memory address to upload the contents of RS as described by its operational equation. This instruction's field format is shown below.

STORE RS, RD, Imm Value

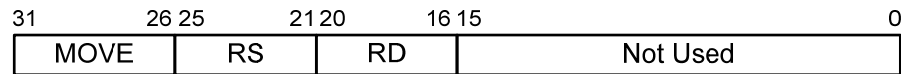
Reg [RS] \rightarrow mem [Reg [RD] + Immediate Value]



The Move (MOVE) instruction does not interact with data memory; nevertheless, it moves data from one register location to another in the RF as described by its operational equation below. The field format of move instruction is shown below.

MOVE RS, RD

Reg [RS] → Reg [RD]

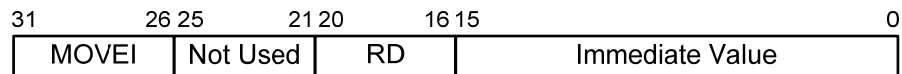


The Move Immediate (MOVEI) instruction moves an immediate value to a destination register as shown below by its operational equation.

The field format of this instruction is shown below.

MOVEI Imm Value, RD

Immediate Value → Reg [RD]



PROGRAM CONTROL INSTRUCTIONS

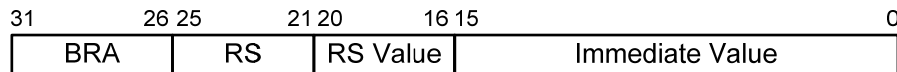
The Branch (BRA) instruction is one of the most essential instructions that control the program flow as shown below with its operational equation.

This instruction first compares the contents of RS against a 5-bit RS Value and then re-directs the program to a new PC address if the comparison is successful.

The new PC address is calculated by incrementing the old PC address by an immediate value. If the comparison is not successful, the program proceeds to the next instruction. This instruction's field format is shown below.

BRA RS, RS Value, Imm Value

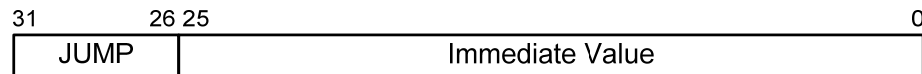
If Reg [RS] = RS Value then PC + Immediate Value → PC else PC → PC + 1



The Jump (JUMP) instruction simply replaces the current PC value with unsigned immediate value as shown below. Its field format is shown below.

JUMP Imm Value

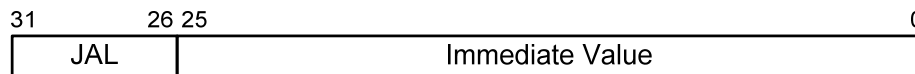
Immediate Value → PC



The Jump-And-Link (JAL) instruction stores the next instruction address to the special register R31 before it replaces the PC value with an immediate value as shown below. Its field format is given below.

JAL Imm Value

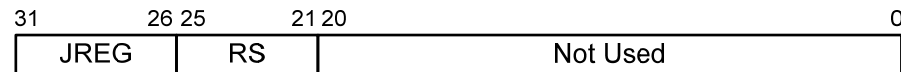
(PC + 1) → Reg [R31] followed by Immediate Value → PC



The Jump Register (JREG) instruction is similar to Jump instruction; however, it uses the contents of RS to replace the current PC value instead of an immediate value as shown below. This instruction's field format is given below.

JREG RS

Reg [RS] → PC

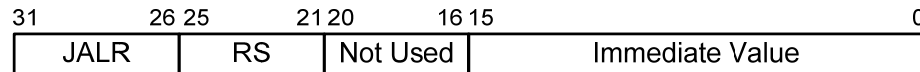


The operation of Jump-And-Link Register (JALR) instruction is also similar to Jump-And-Link instruction.

It stores the next PC value in the special register R31 before it overwrites the PC with the contents of RS as described by its operational equation below. The field format of this instruction is also shown below.

JALR RS

$(PC + 1) \rightarrow \text{Reg [R31]}$ followed by $\text{Reg [RS]} \rightarrow PC$

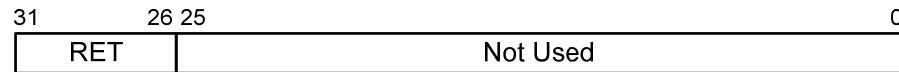


Return instruction (RET) below works with Jump-And-Link or Jump-And-Link Register instructions.

It retrieves the old program address stored earlier at R31 and replaces this value with the current PC value in order to go back to the original program location as described by its operational equation. This instruction's field format is given below.

RET

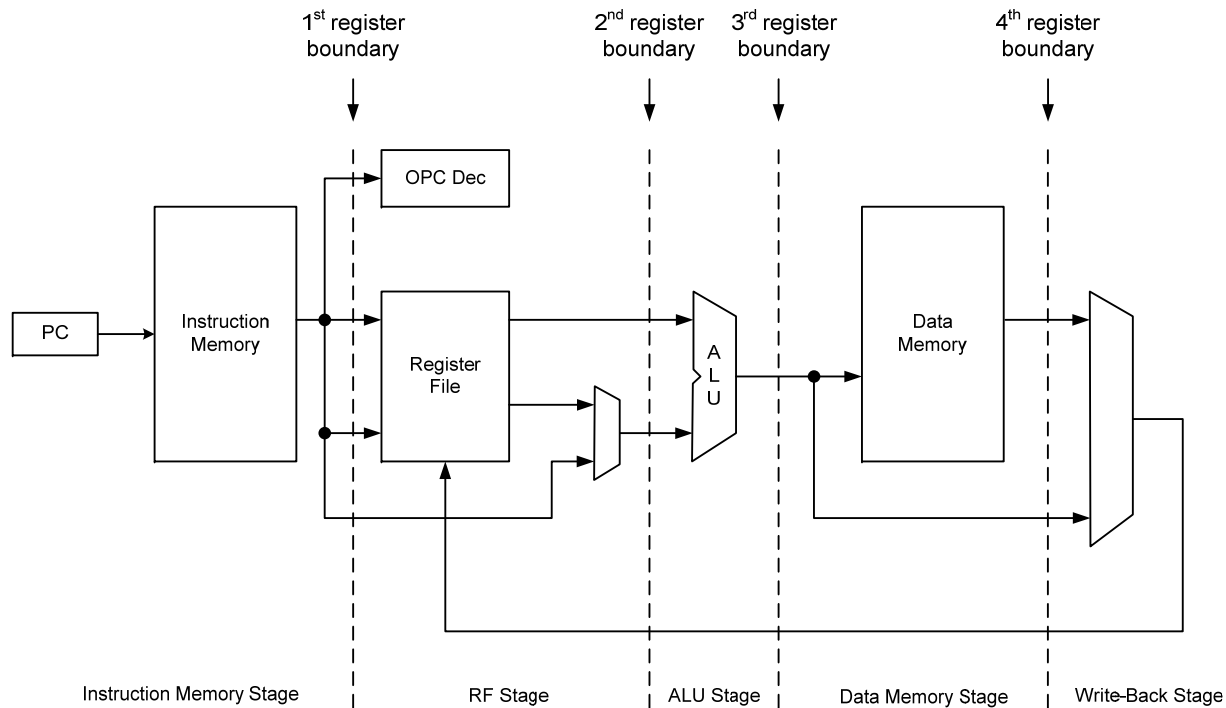
Reg [R31] → PC



CPU DATA-PATH

The RISC instructions examined earlier are executed in a 5-stage CPU data-path called CPU pipeline.

Each stage in this data-path is separated from its neighboring stage by a register boundary that stores processed data for one cycle.



The **instruction memory** stage is where the instructions are fetched from the instruction memory with the PC value used as the instruction memory address. After the instruction is fetched from the instruction memory, it is stored in the instruction register which corresponds to the first register boundary.

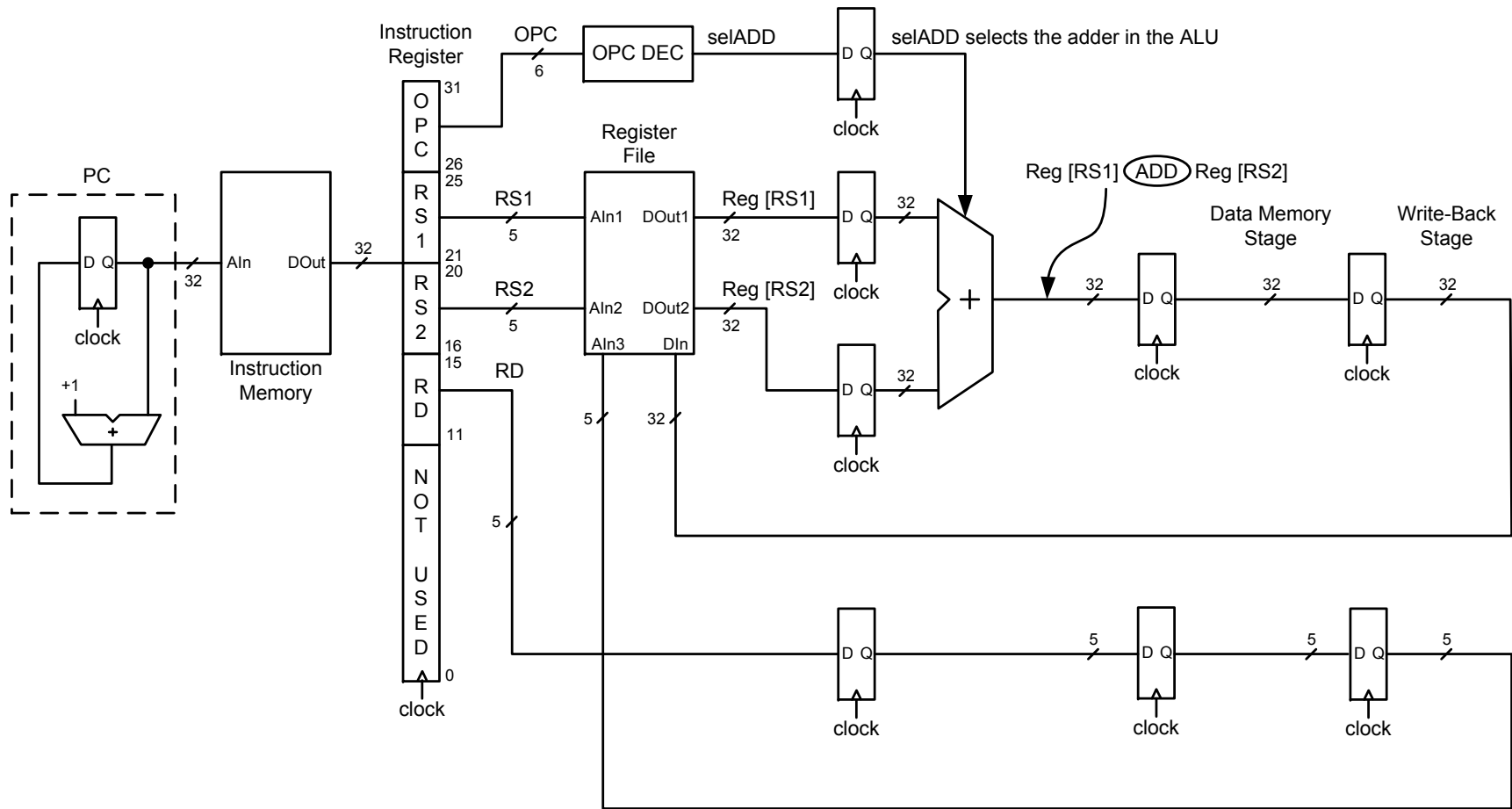
The **next stage is the RF stage** where the instruction OPC is separated from its operands. The OPC is decoded to control the data routing for the rest of the pipeline; the operand fields are either the address values used to access the stored data in the RF or immediate data supplied by the user.

The **third stage is the ALU stage**. The data from the source registers in the RF and the immediate data from the instruction (if any) are combined in this stage according to the OPC and subsequently loaded to the third register boundary.

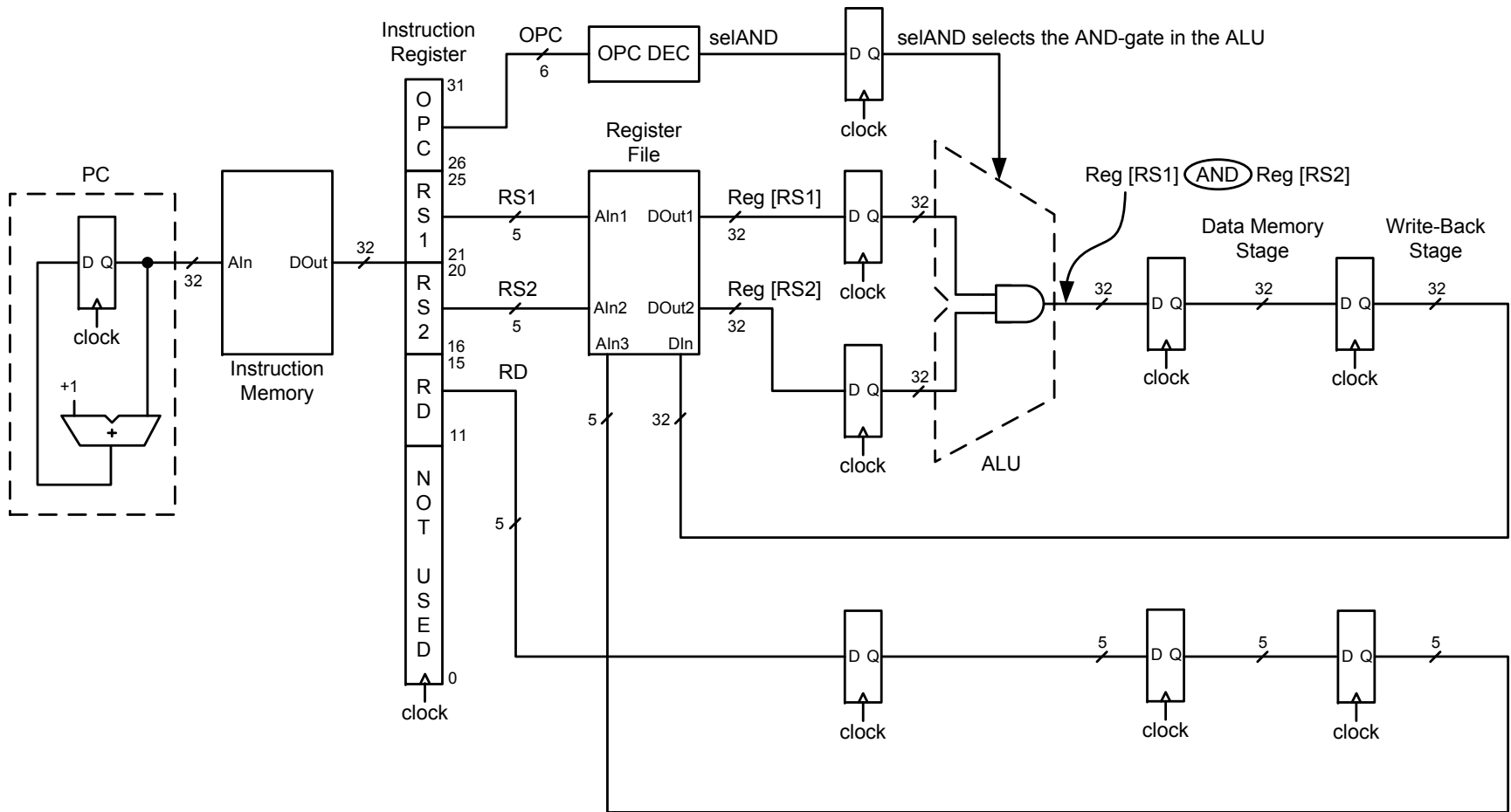
The processed data entering **the data memory stage** either provides an effective address for the data memory or completely bypasses this unit. If the instruction calls for loading or storing data to the data memory, the ALU calculates the data memory address to access the memory contents. Otherwise, the ALU result simply bypasses the data memory and is stored at the fourth register boundary.

The last and the fifth stage of the pipeline is **the write-back stage**. In this stage, data is either routed from the output of the data memory or from the bypass path around the data memory to the RF. As a result, the RF is updated with new data at a destination register whose address is provided in the instruction.

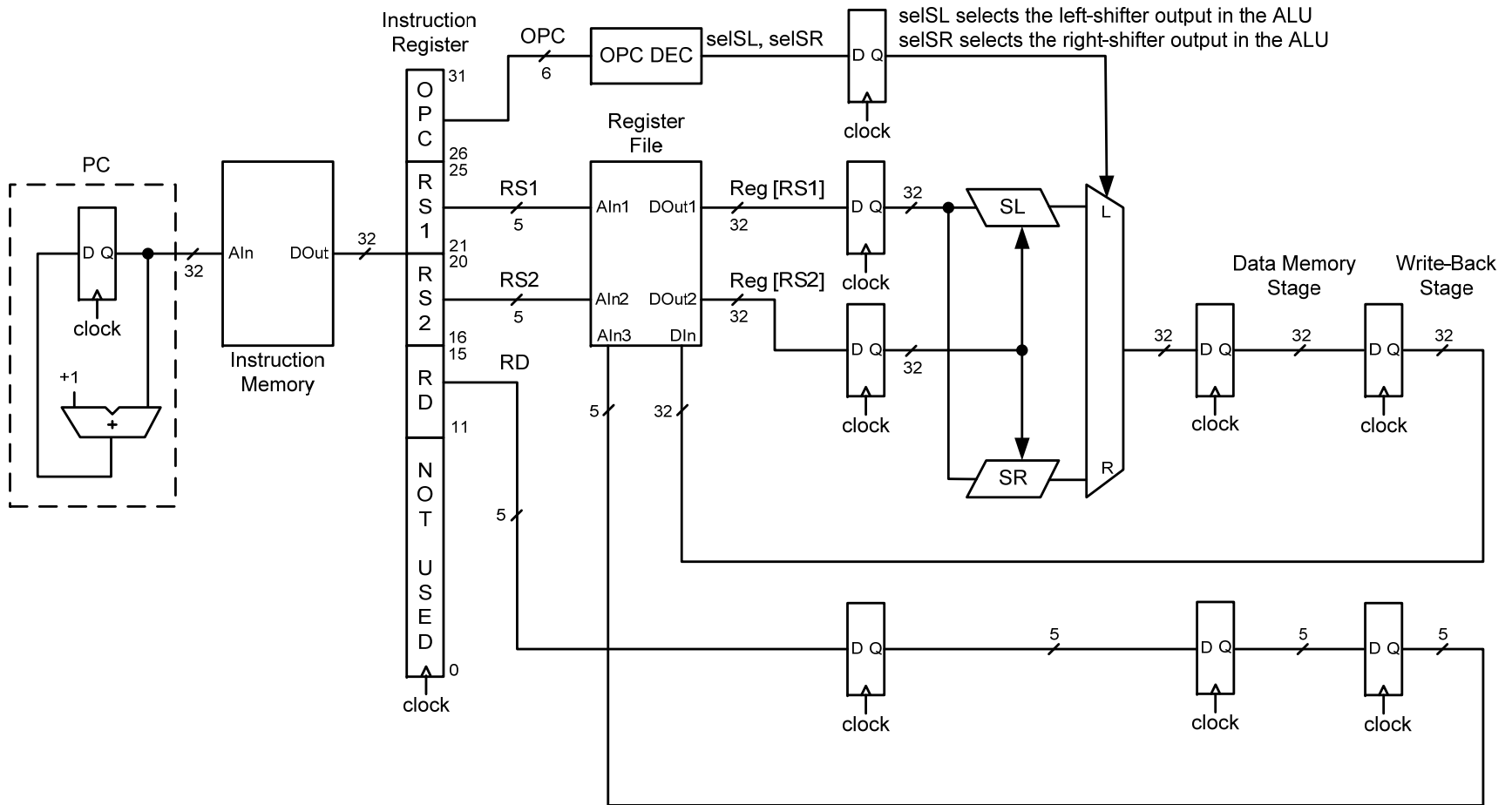
DATA-PATHS FOR REGISTER-TO-REGISTER ALU INSTRUCTIONS



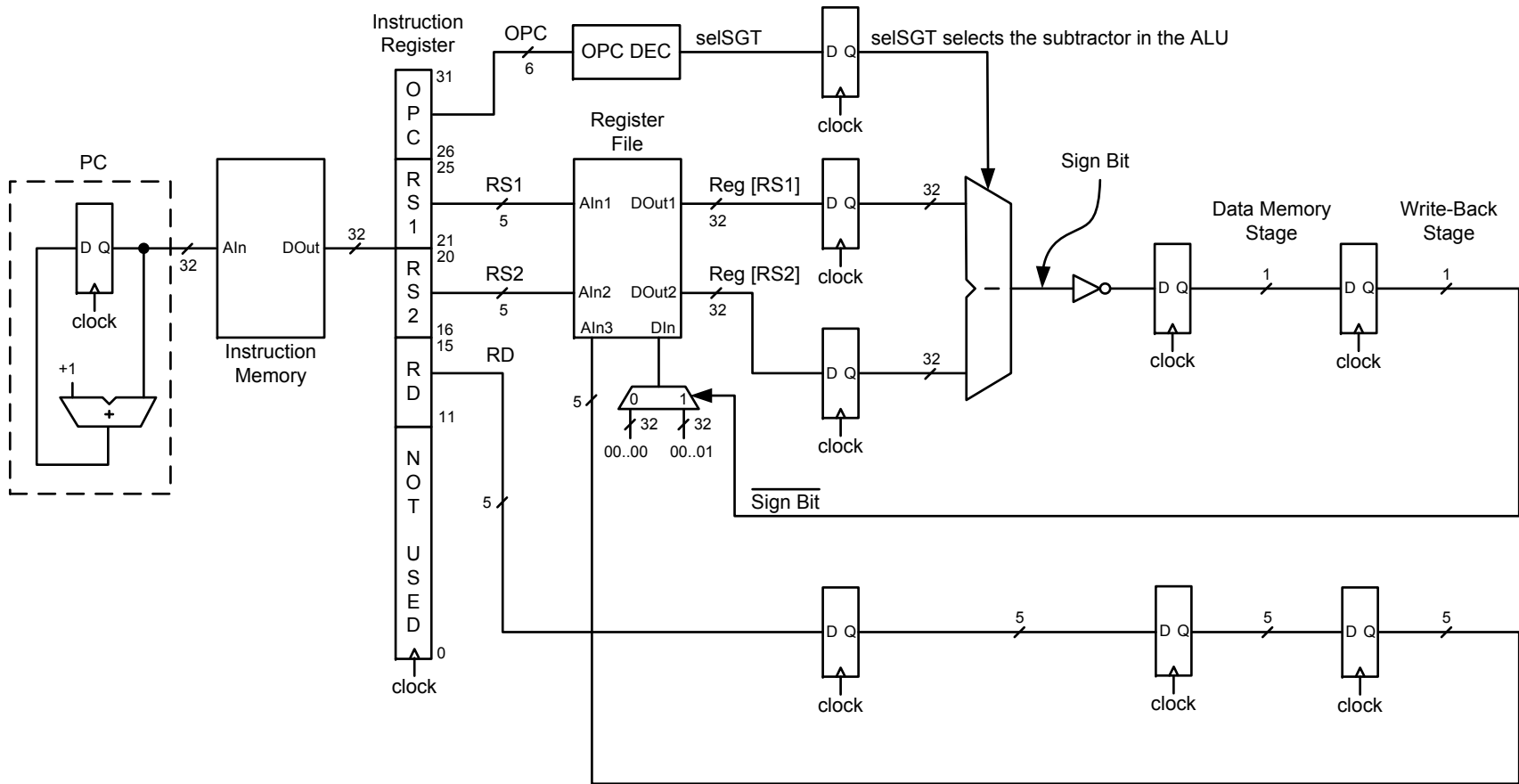
ADD instruction data-path



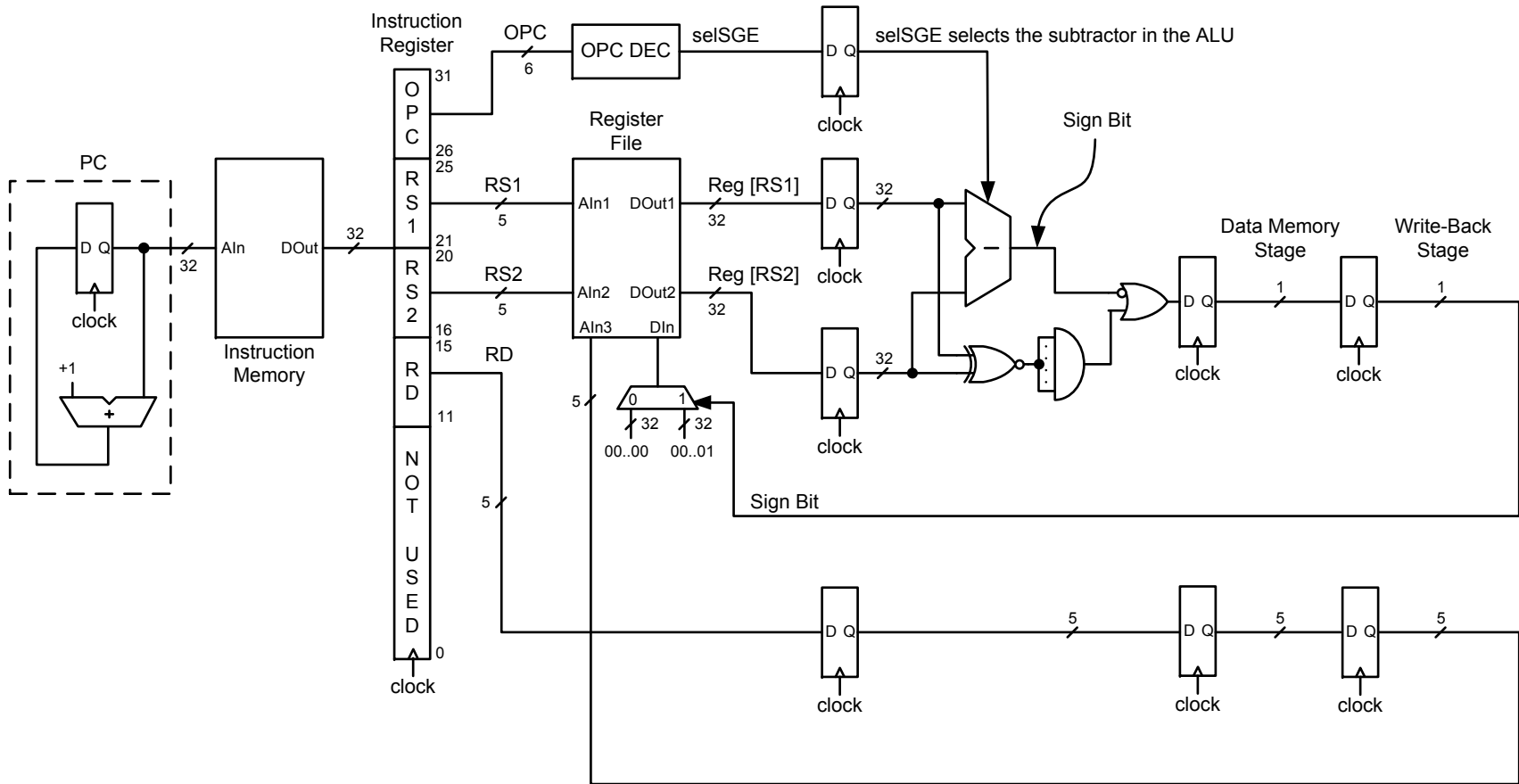
AND instruction data-path



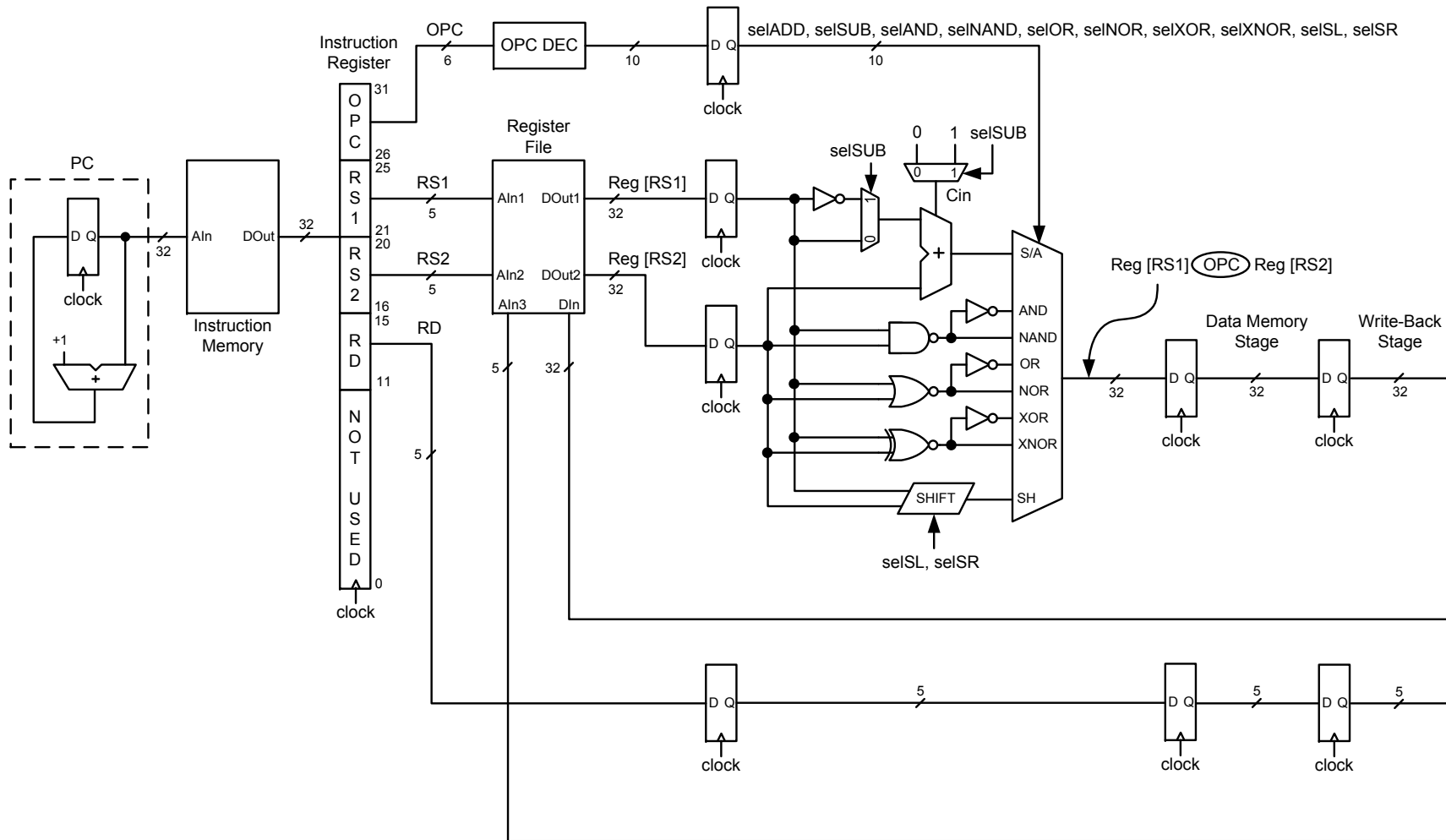
Shift instruction data-path



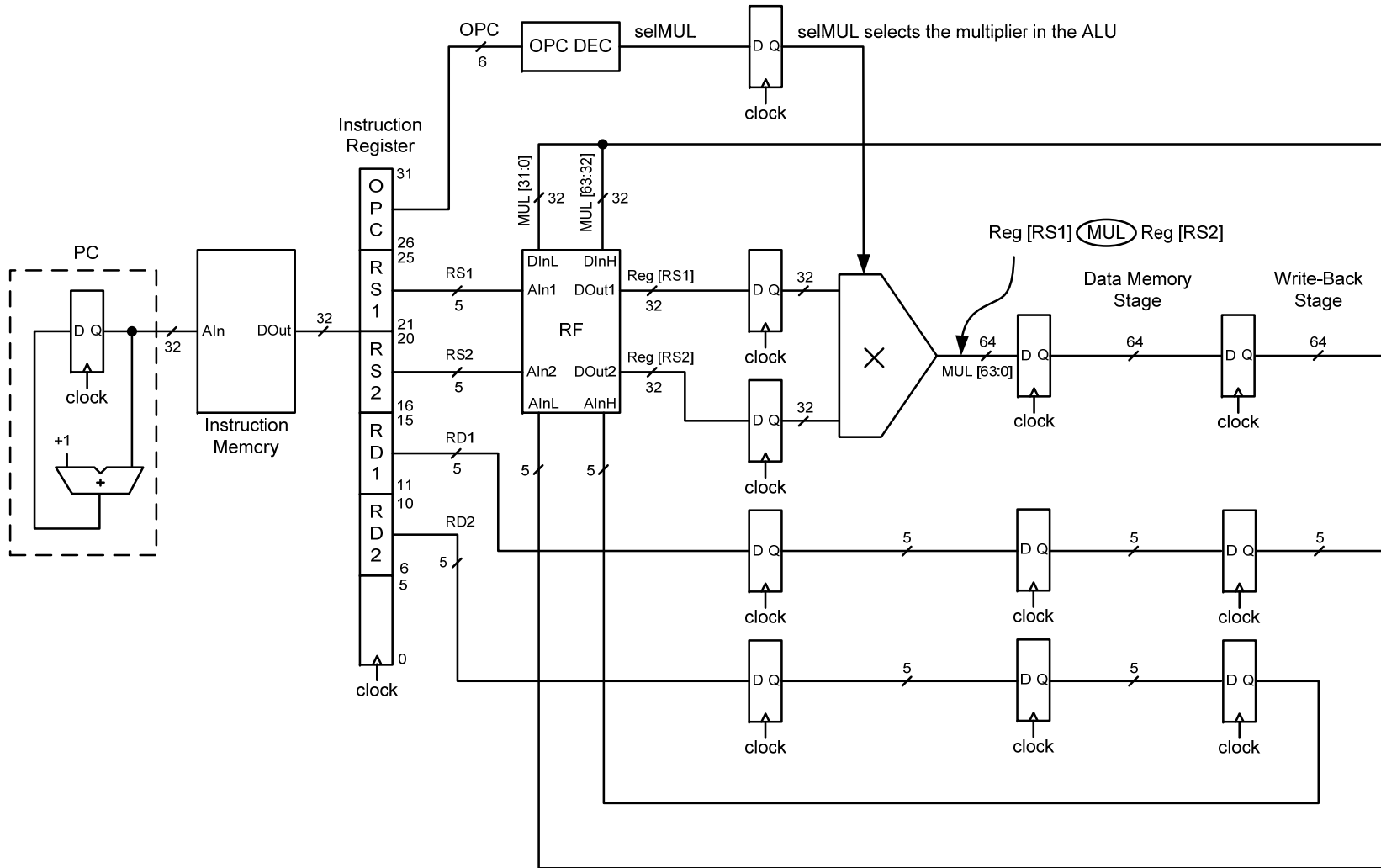
SGT instruction data-path



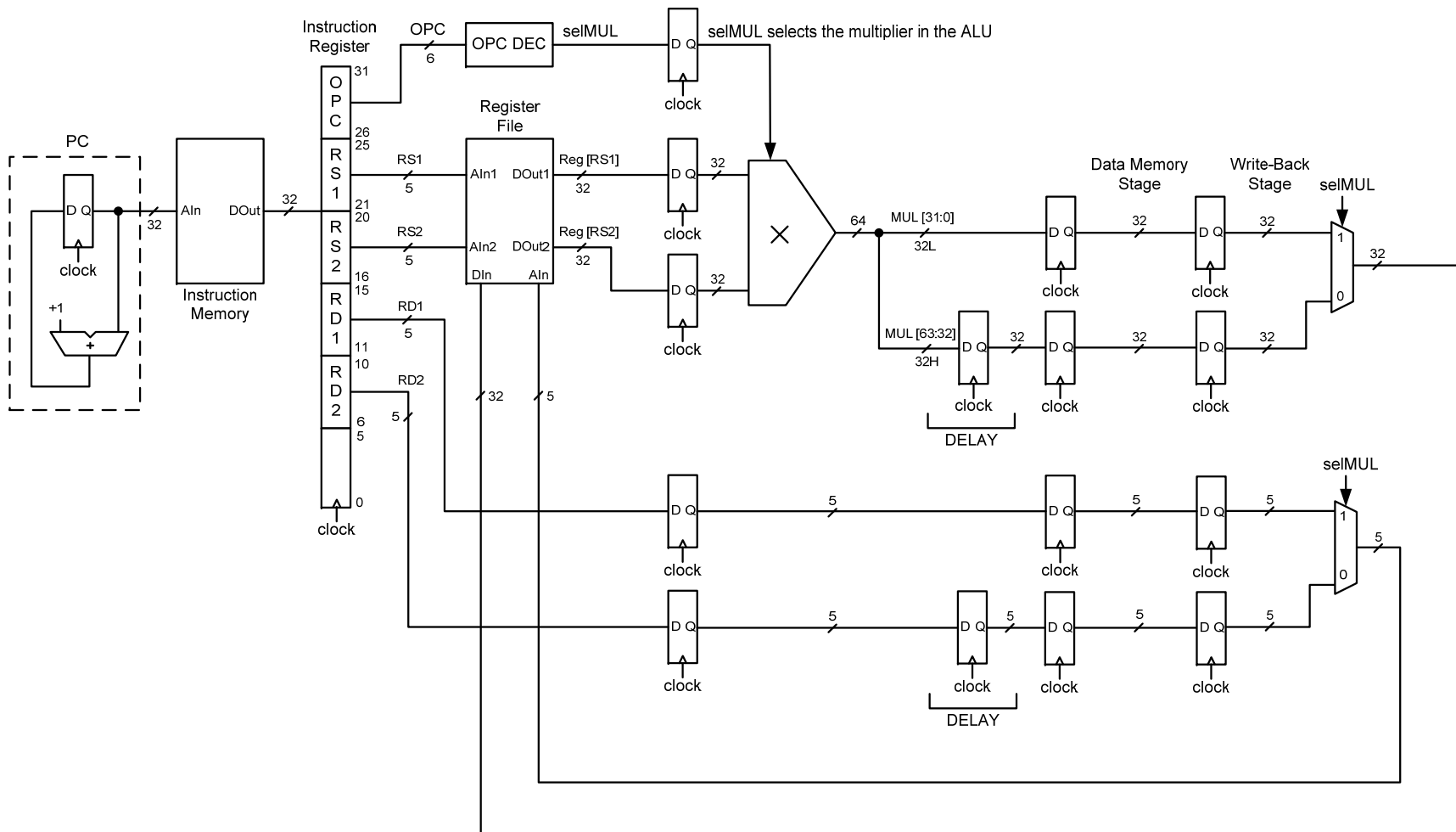
SGE instruction data-path



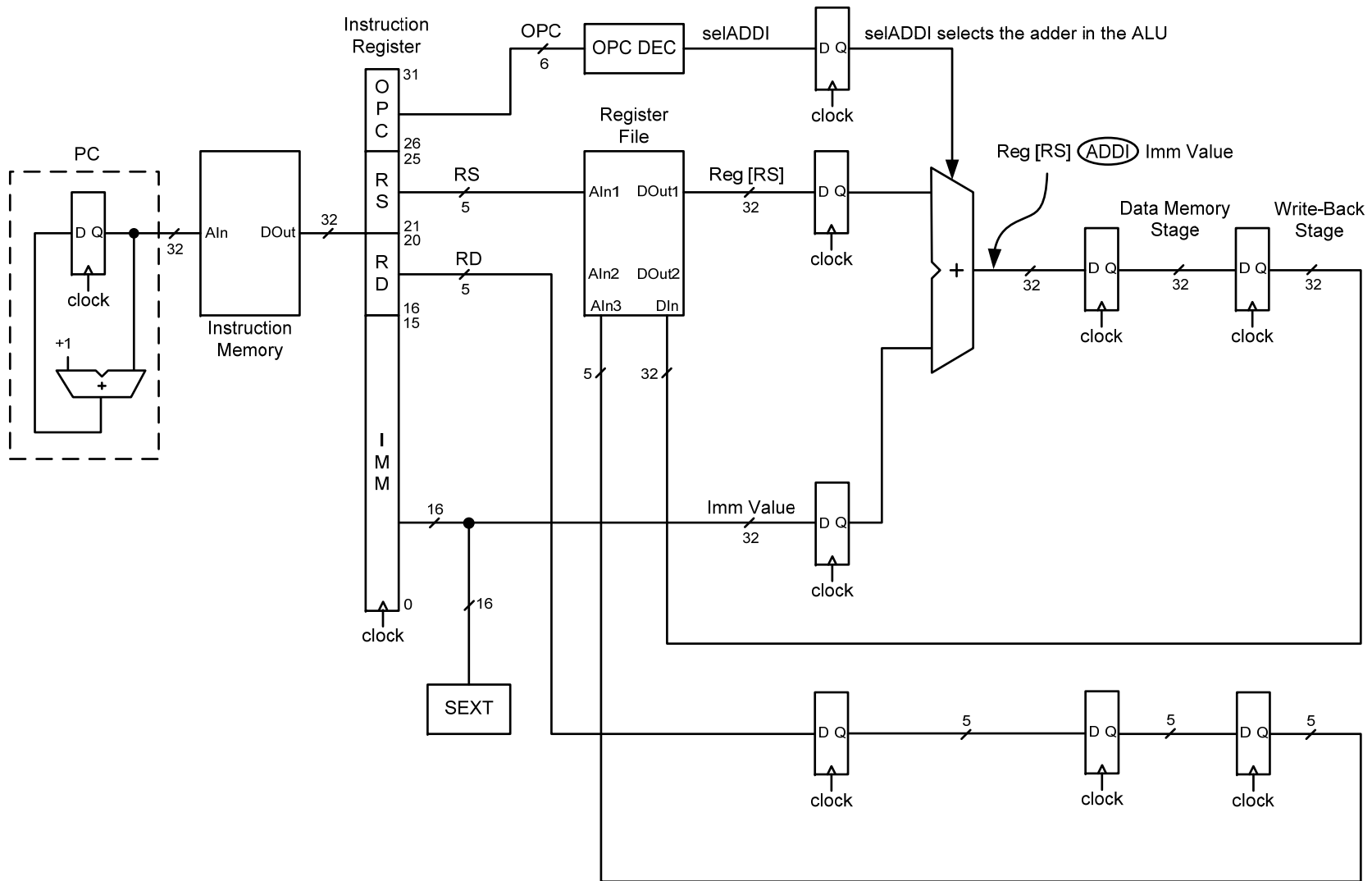
Combined register-to-register ALU instruction data-paths



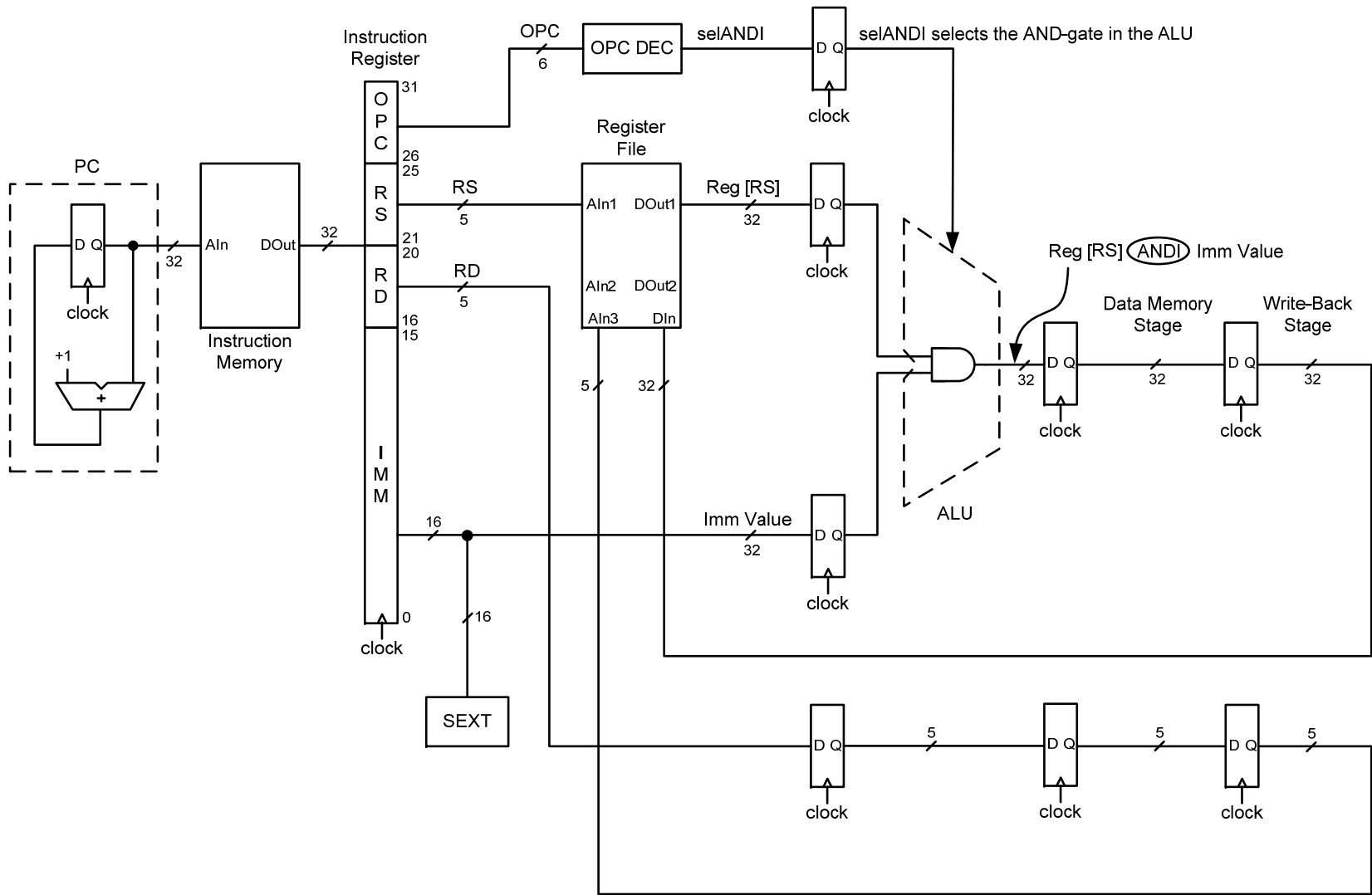
The data-path for fixed-point multiplication using 2 write-back ports



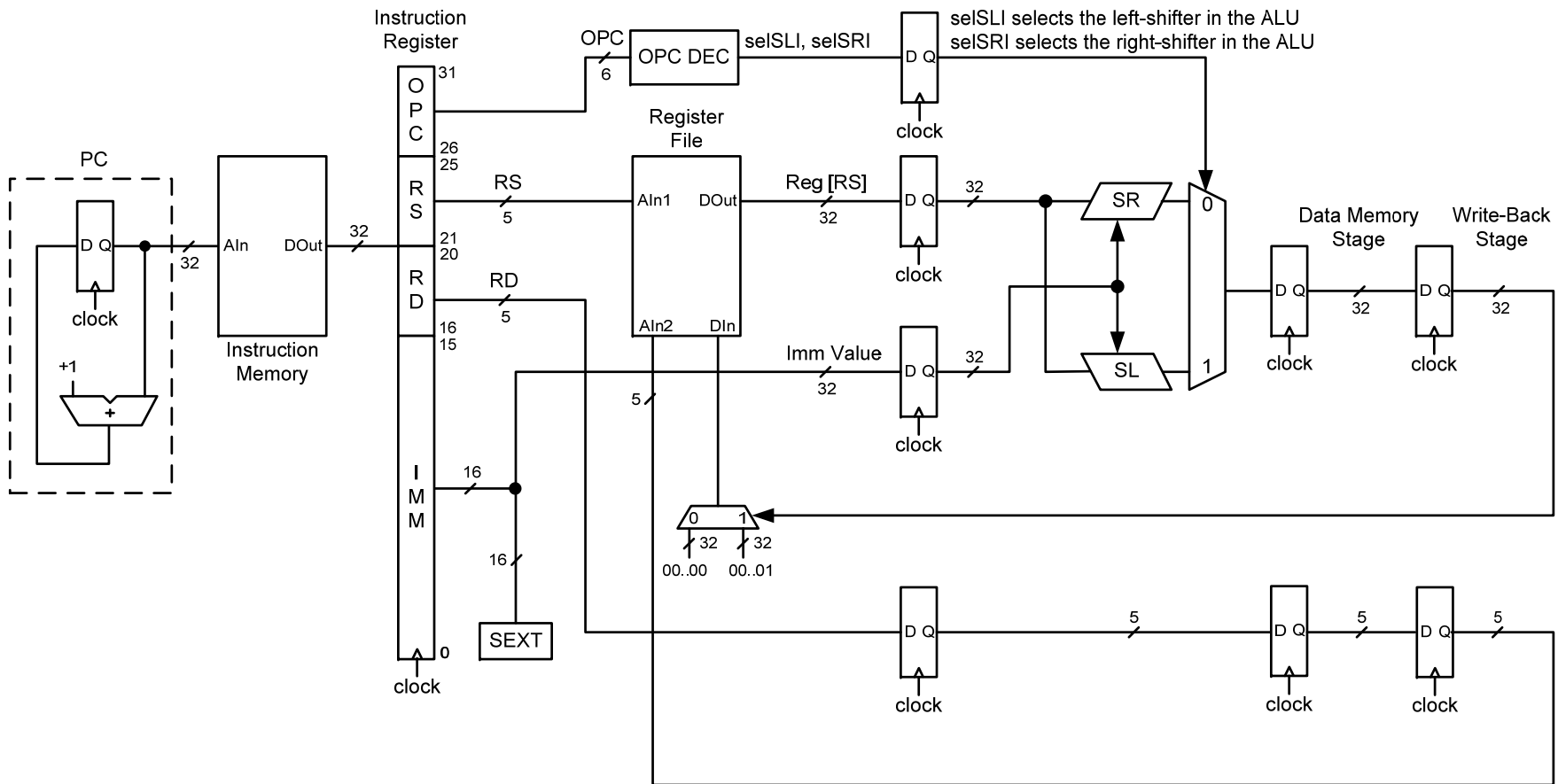
The data-path for fixed-point multiplication using a single write-back port



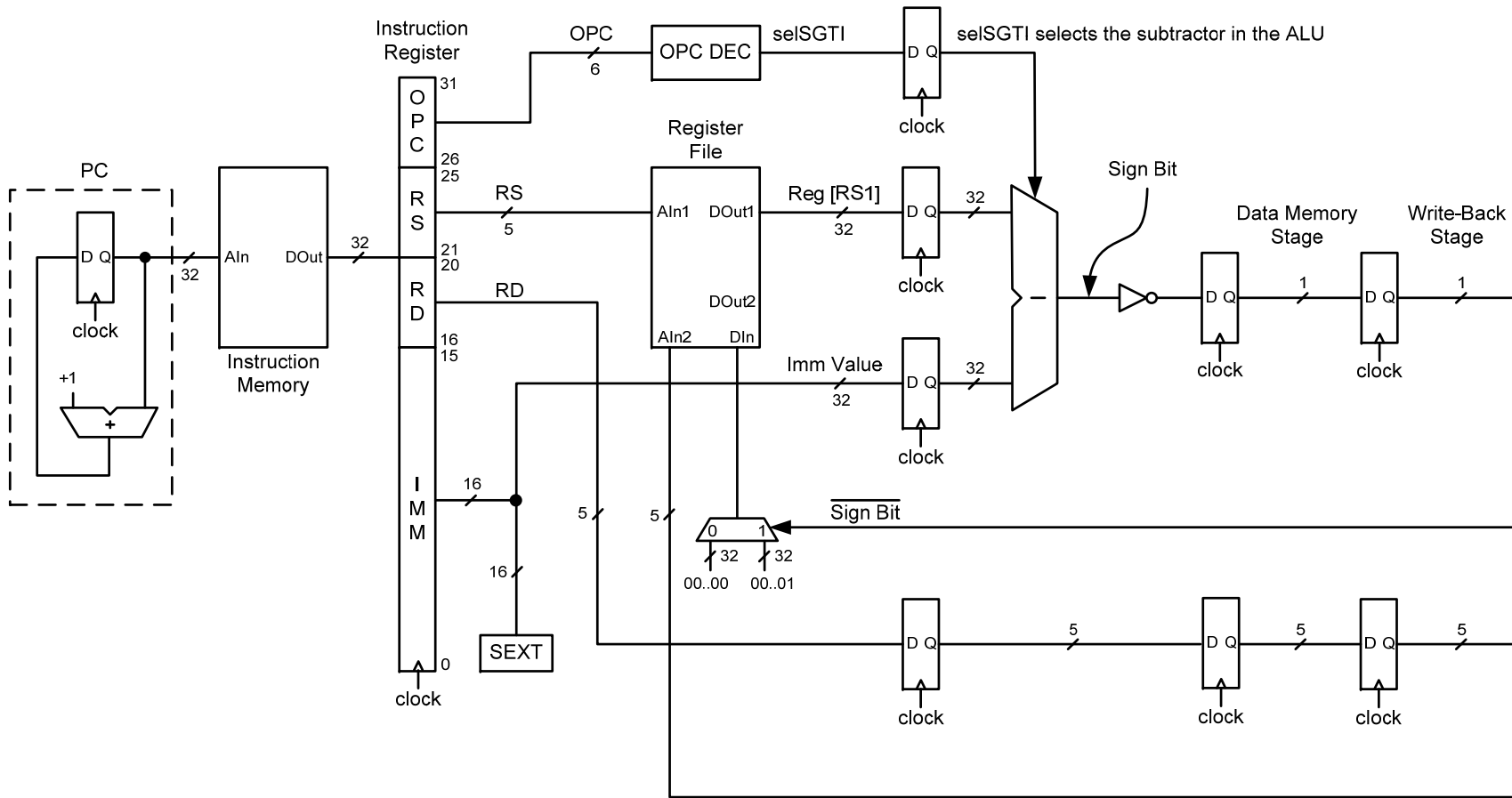
ADDI instruction data-path



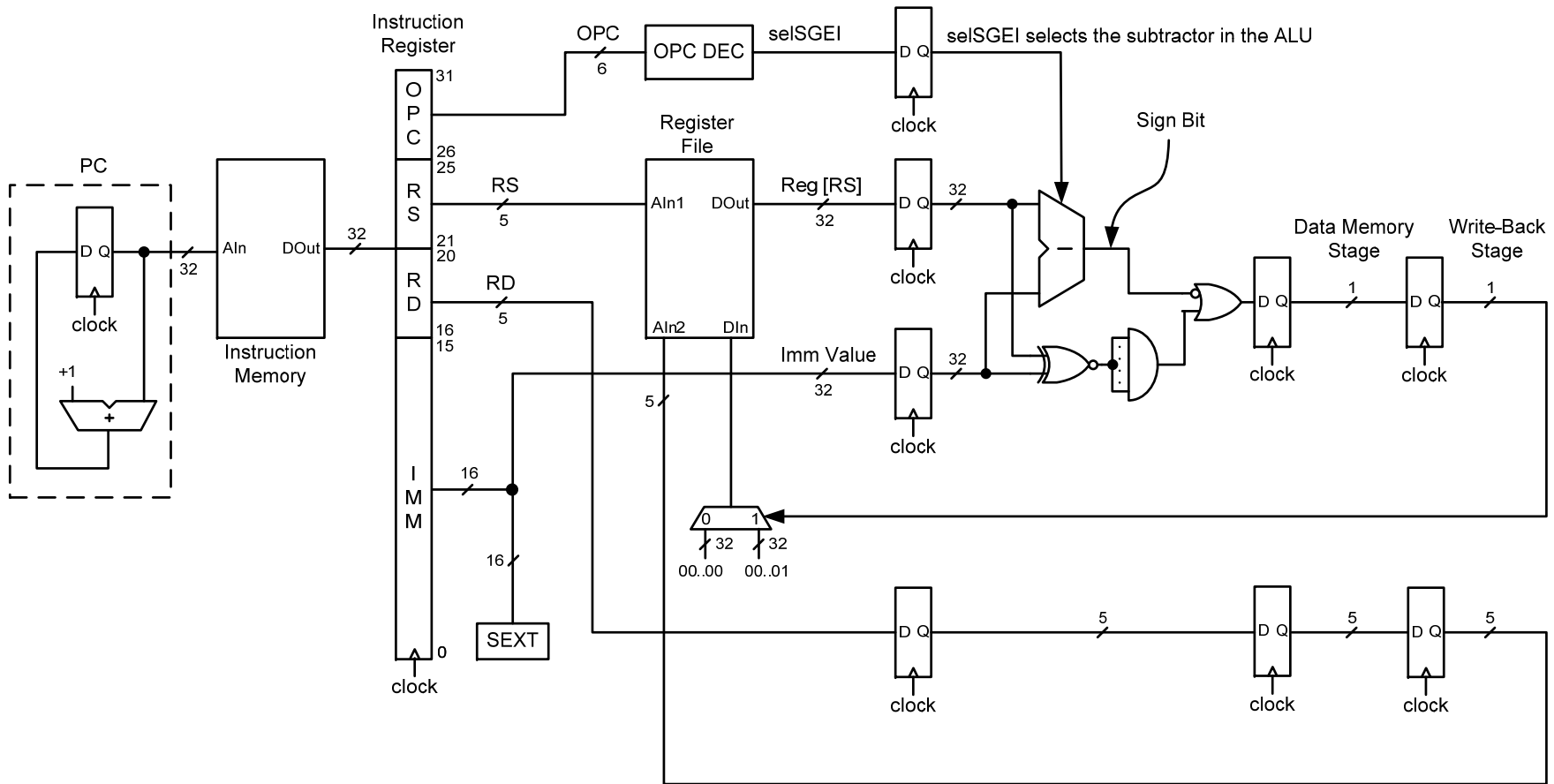
ANDI instruction data-path



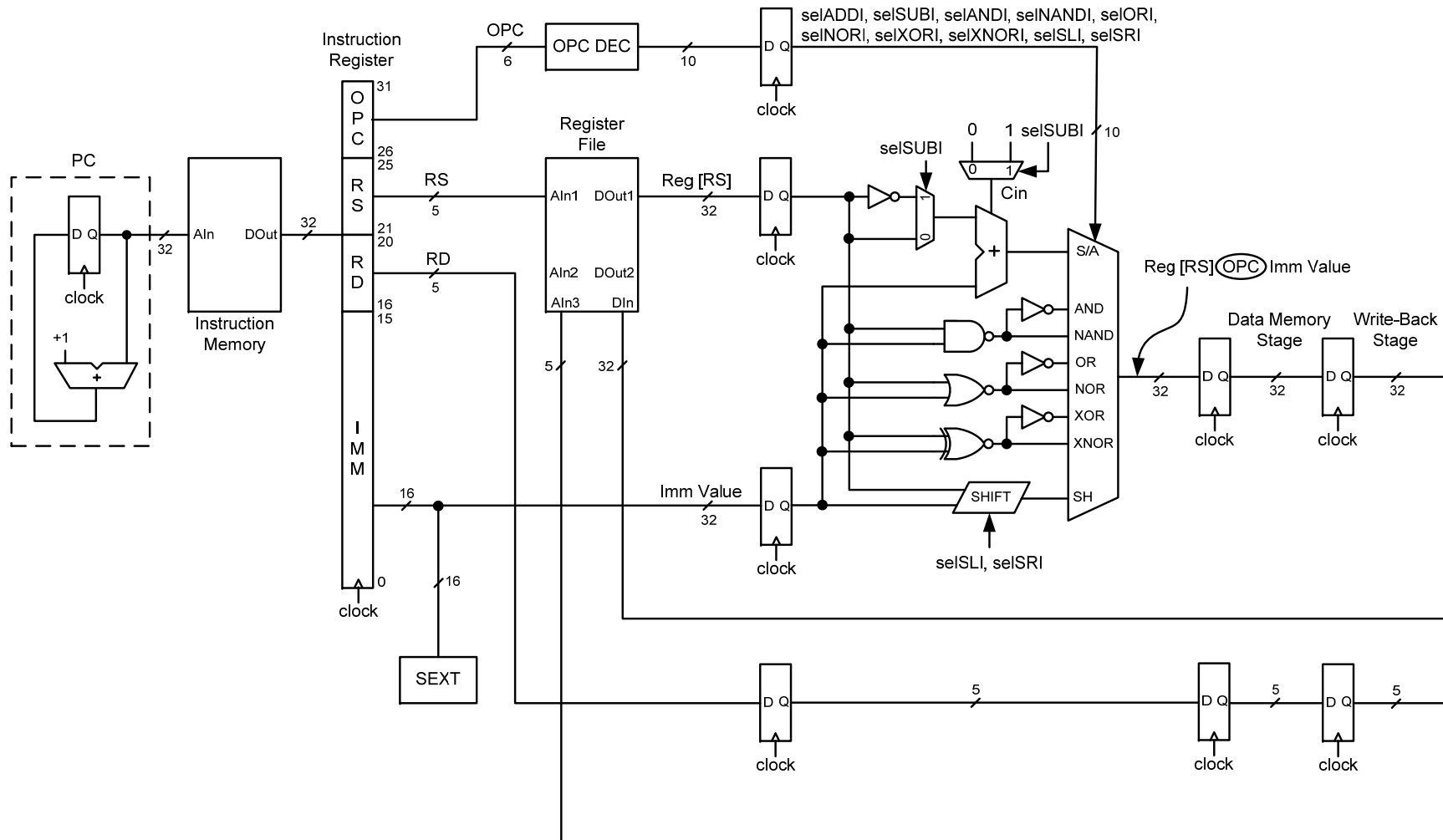
SLI and SRI instruction data-paths



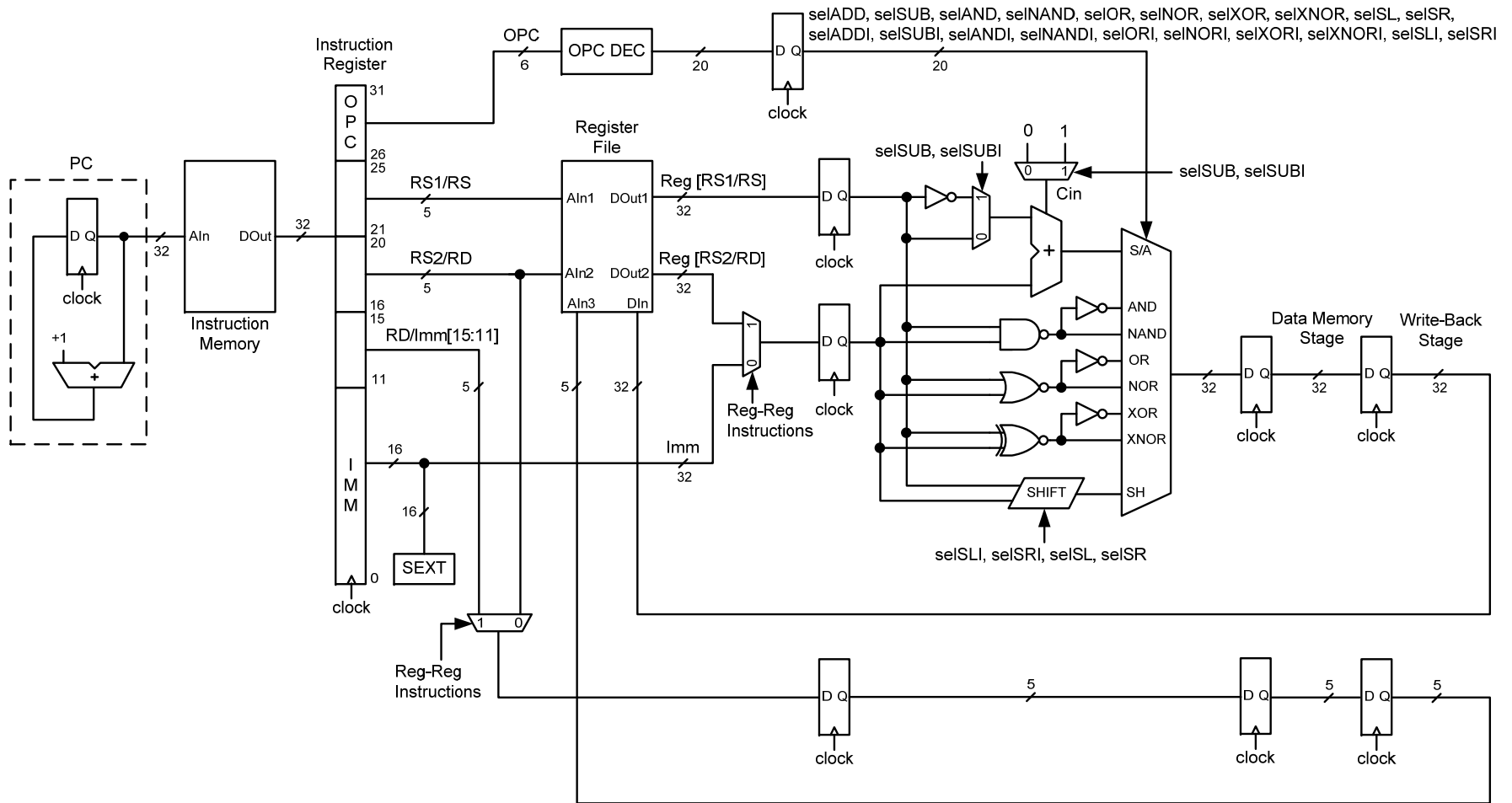
SGTI instruction data-path



SGEI instruction data-path

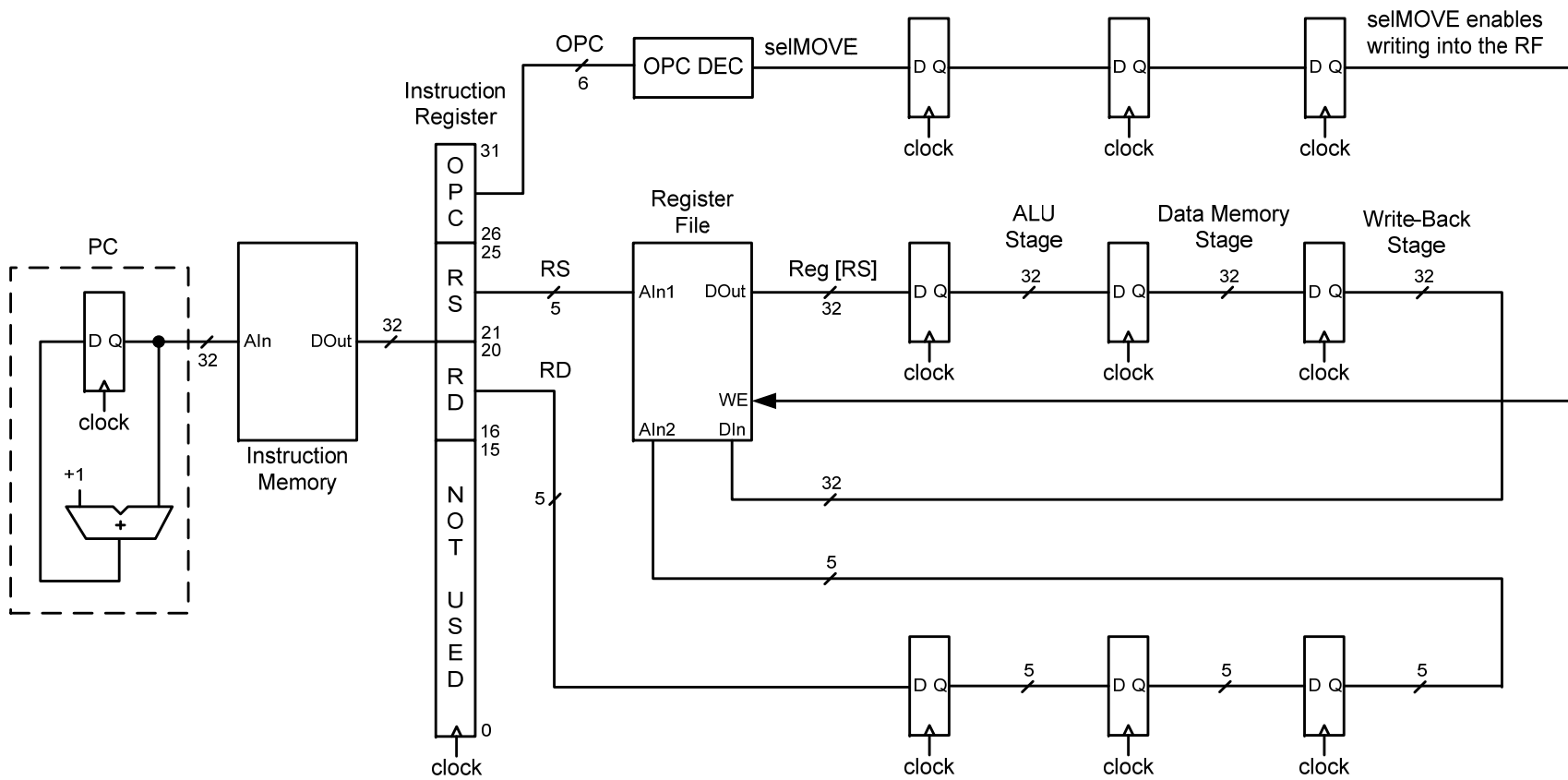


Combined immediate ALU instruction data-paths

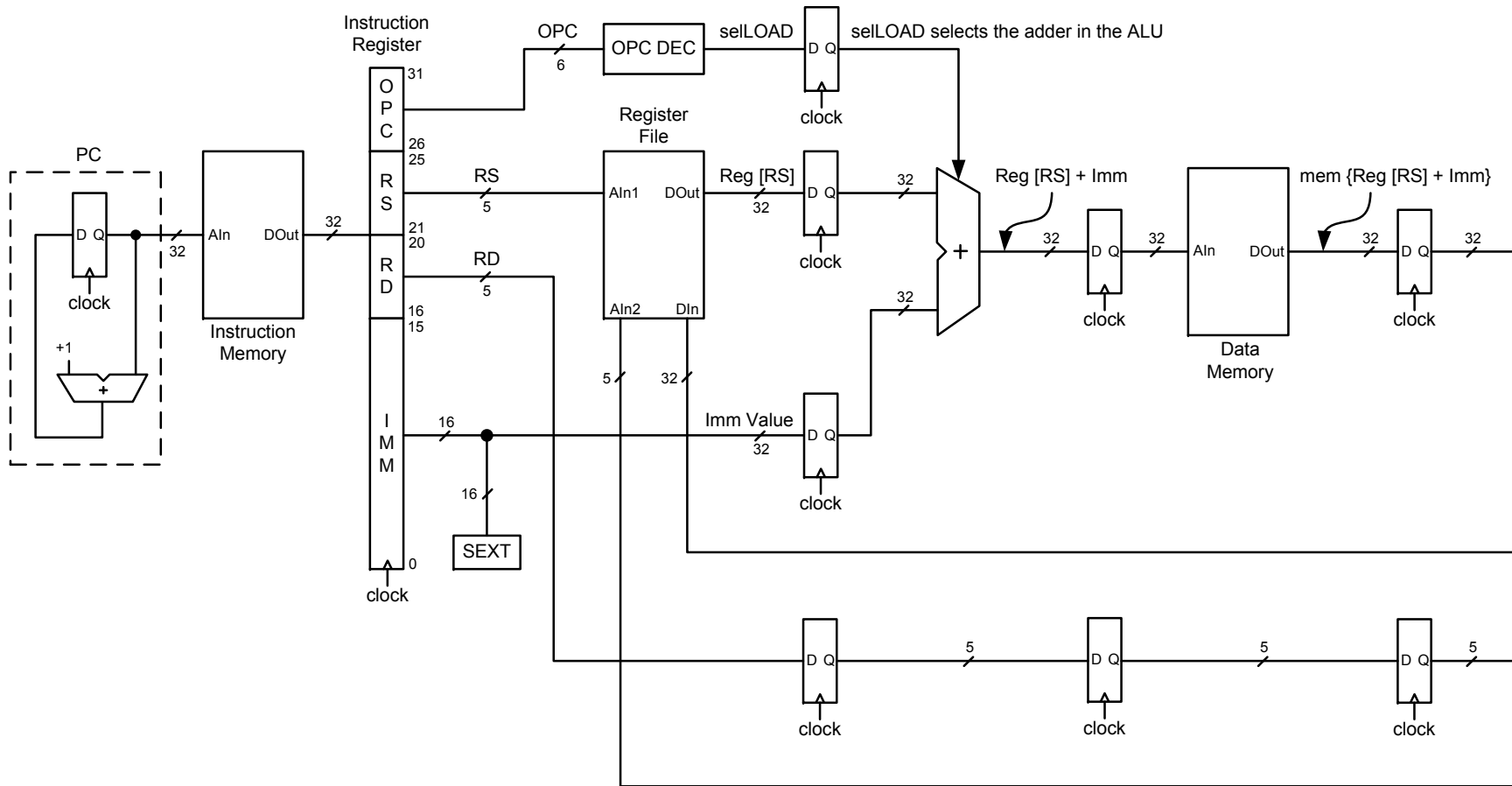


Register-to-register and immediate ALU instruction data-paths

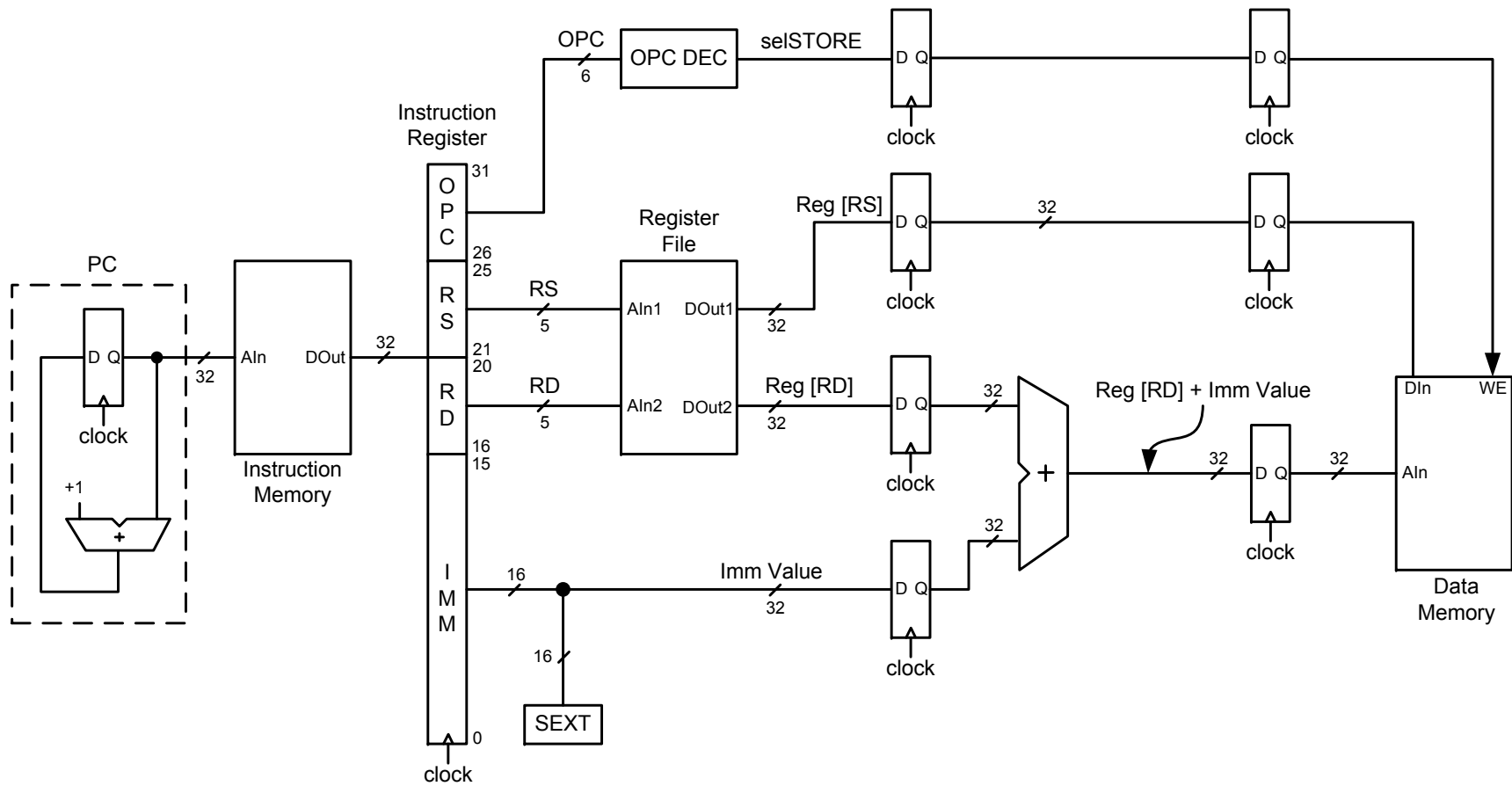
DATA-PATHS FOR DATA MOVEMENT INSTRUCTIONS



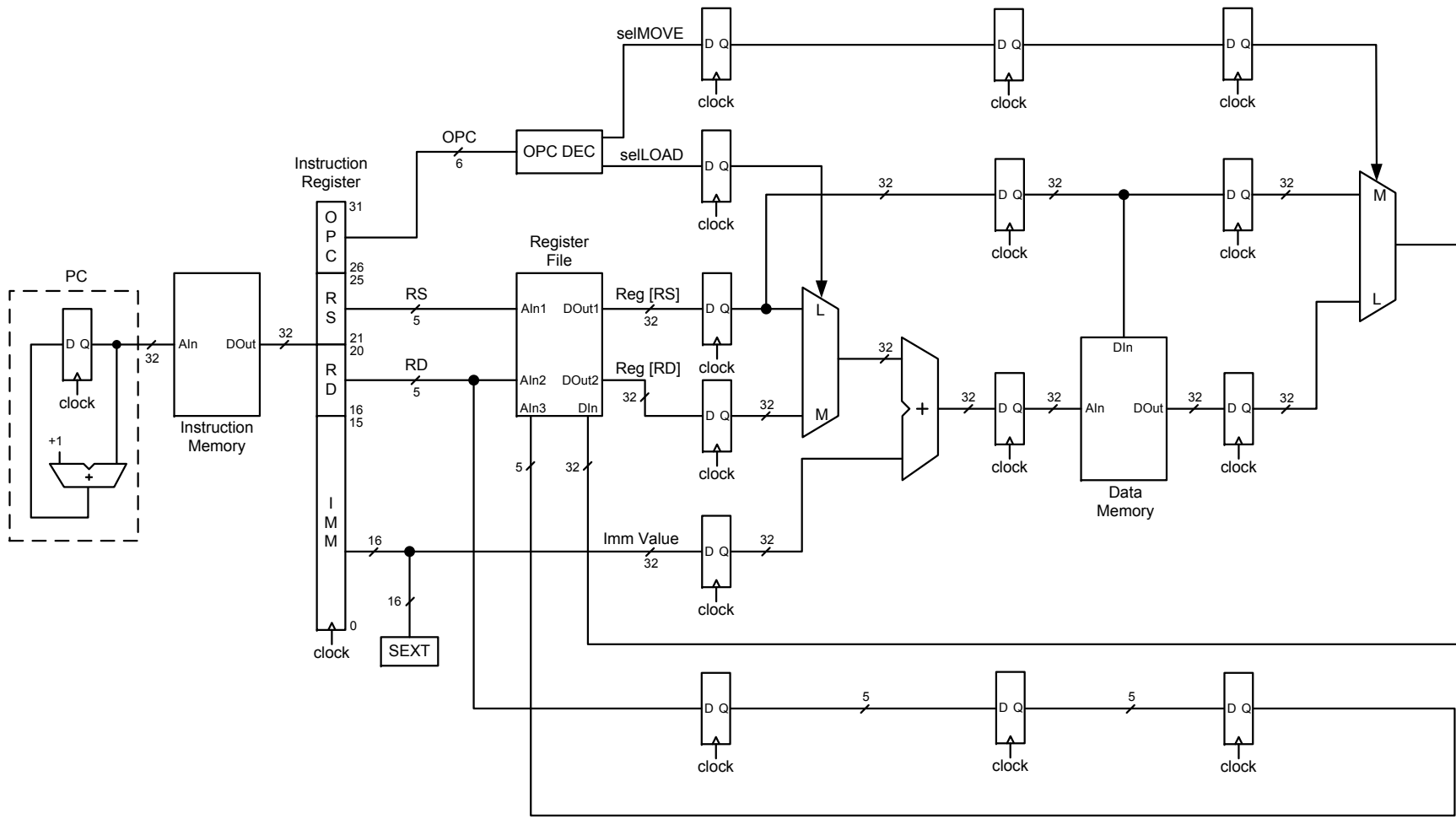
MOVE instruction data-path



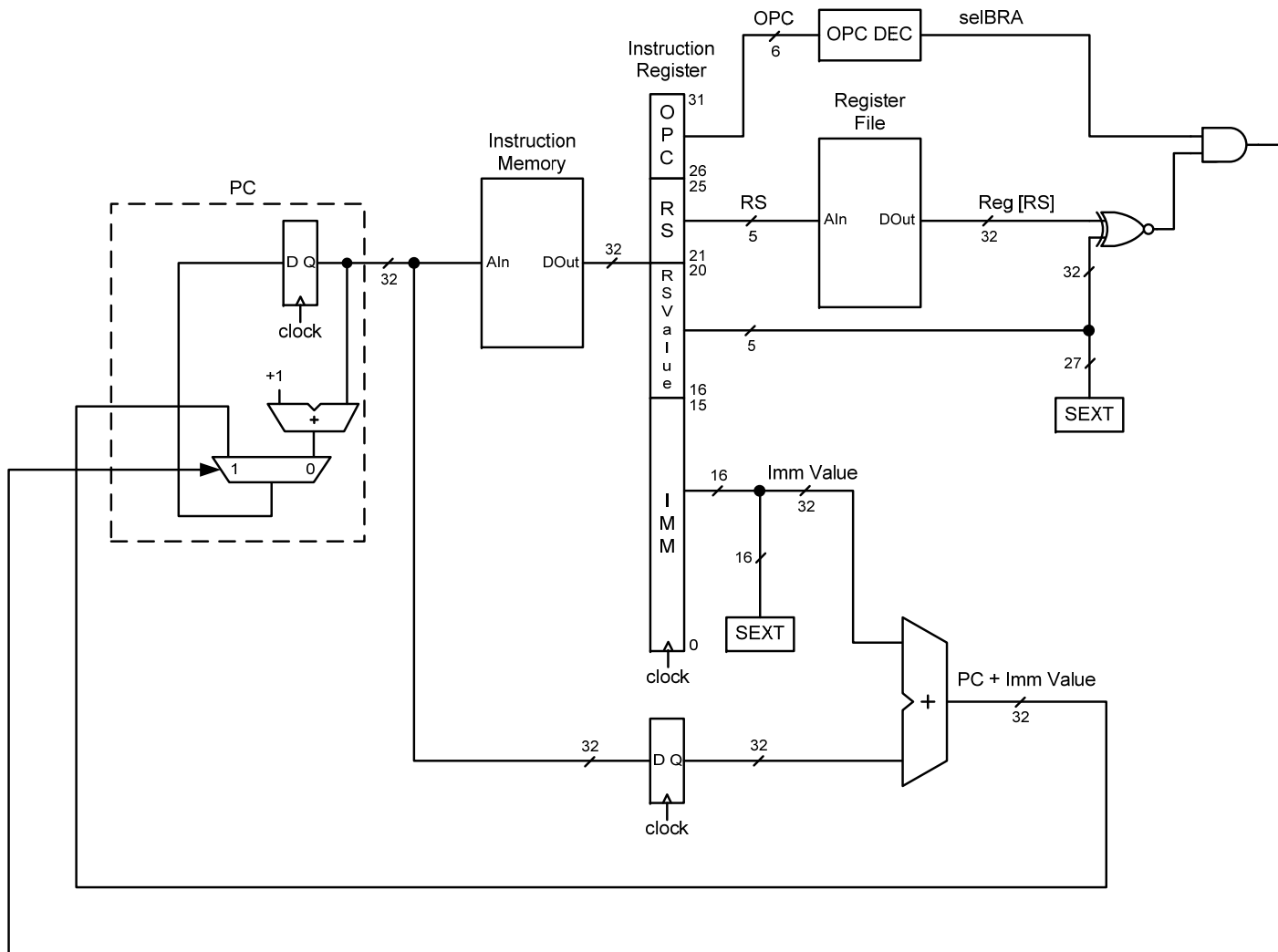
LOAD instruction data-path



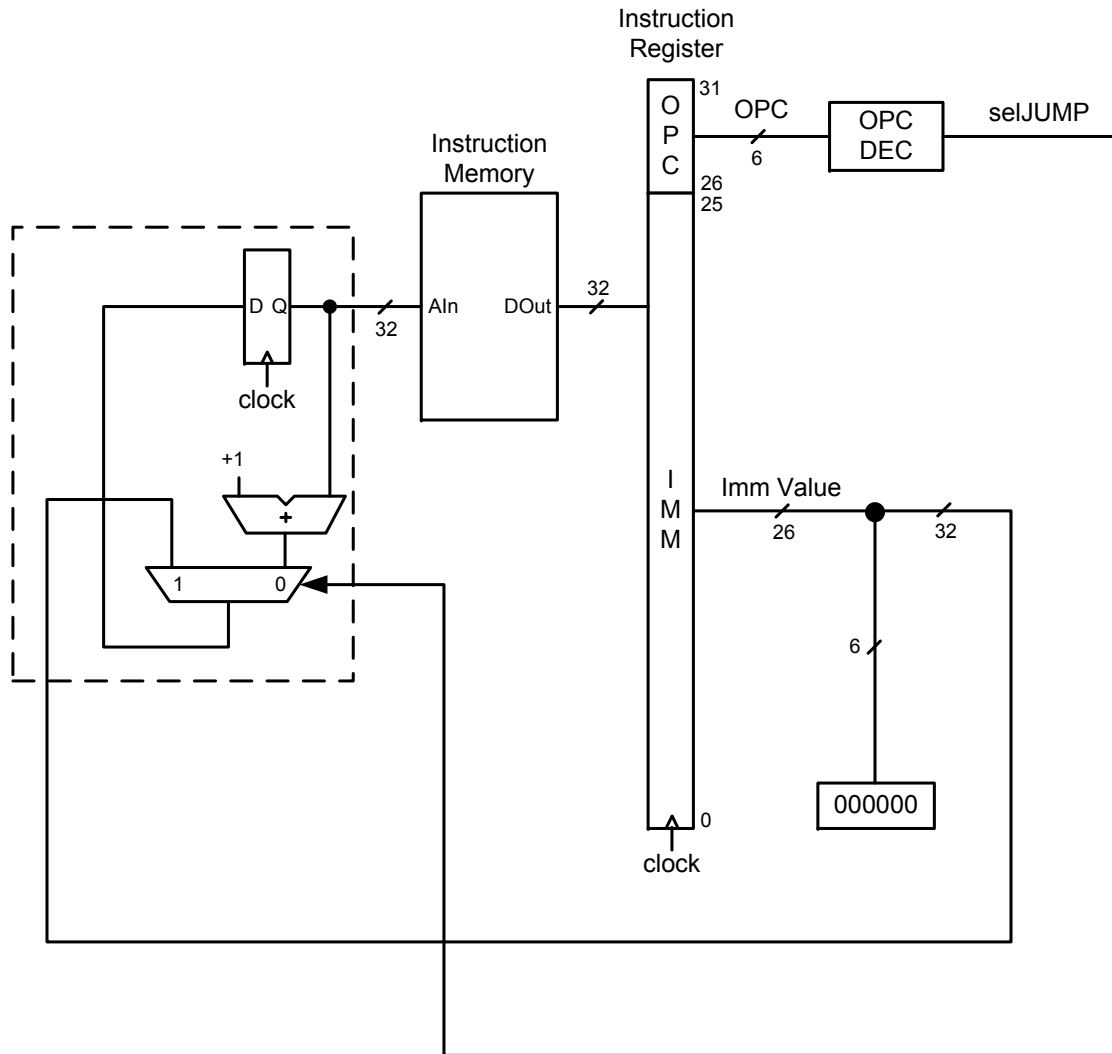
STORE instruction data-path



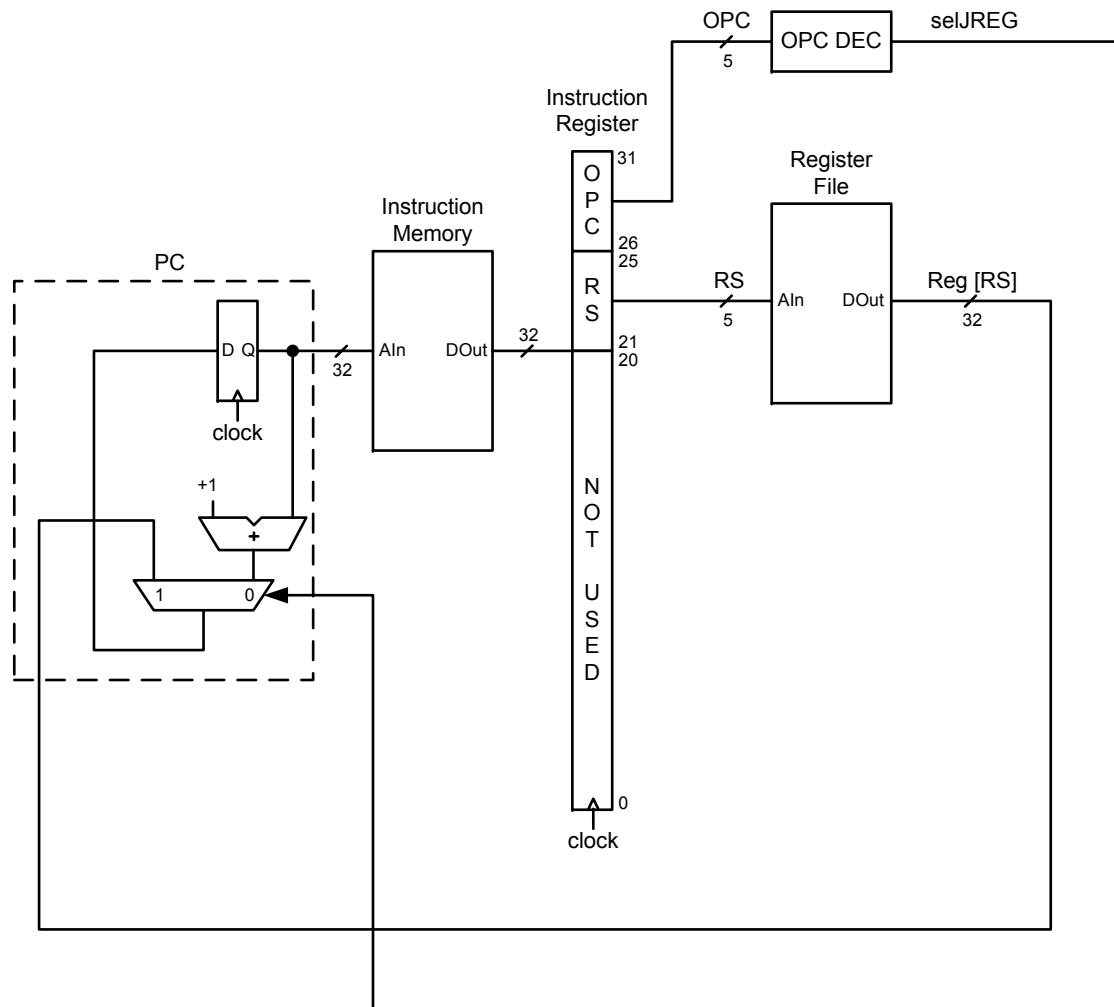
Data movement instruction data-paths (MOVE, LOAD and STORE)



BRA instruction data-path



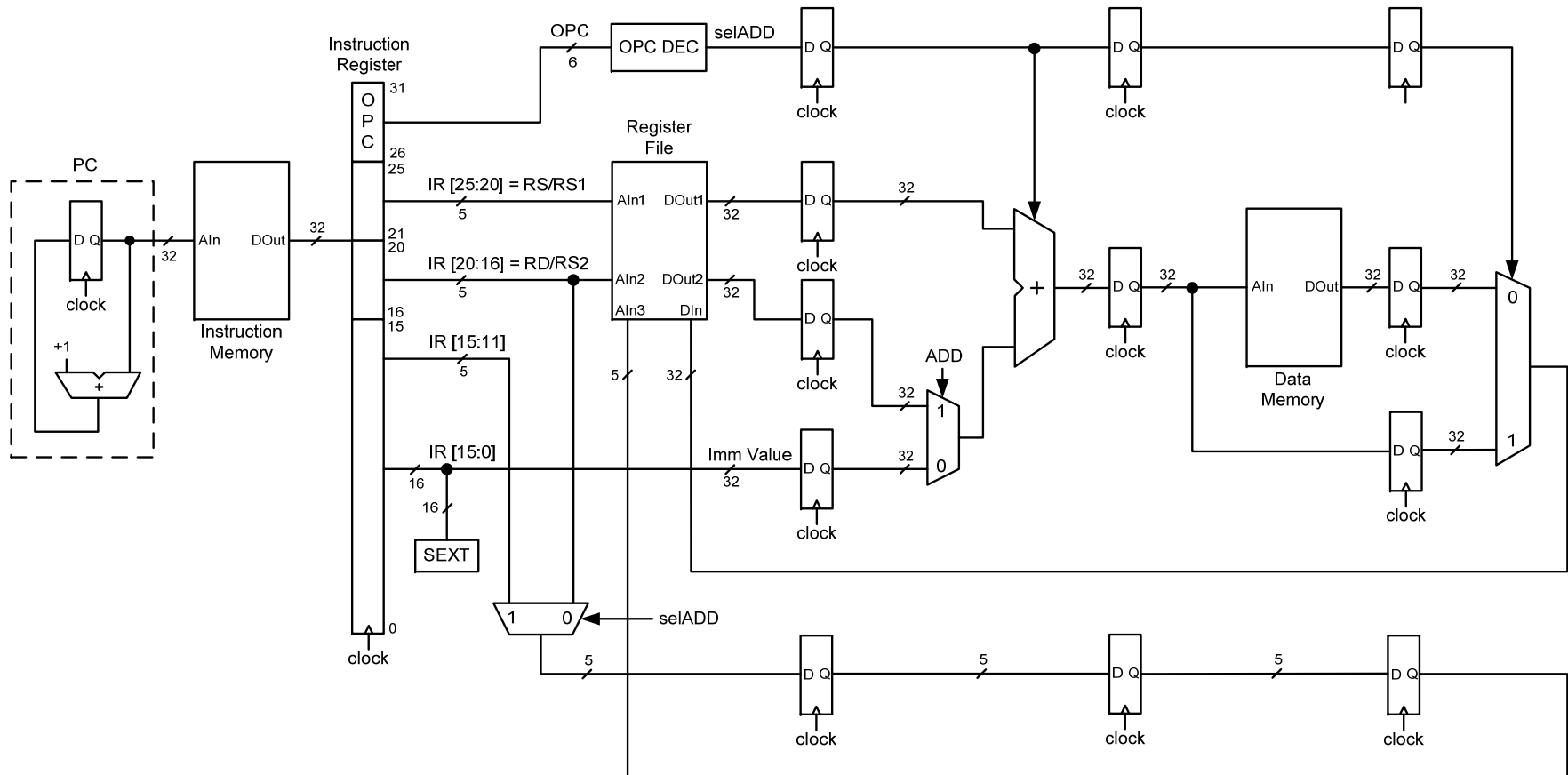
JUMP instruction data-path



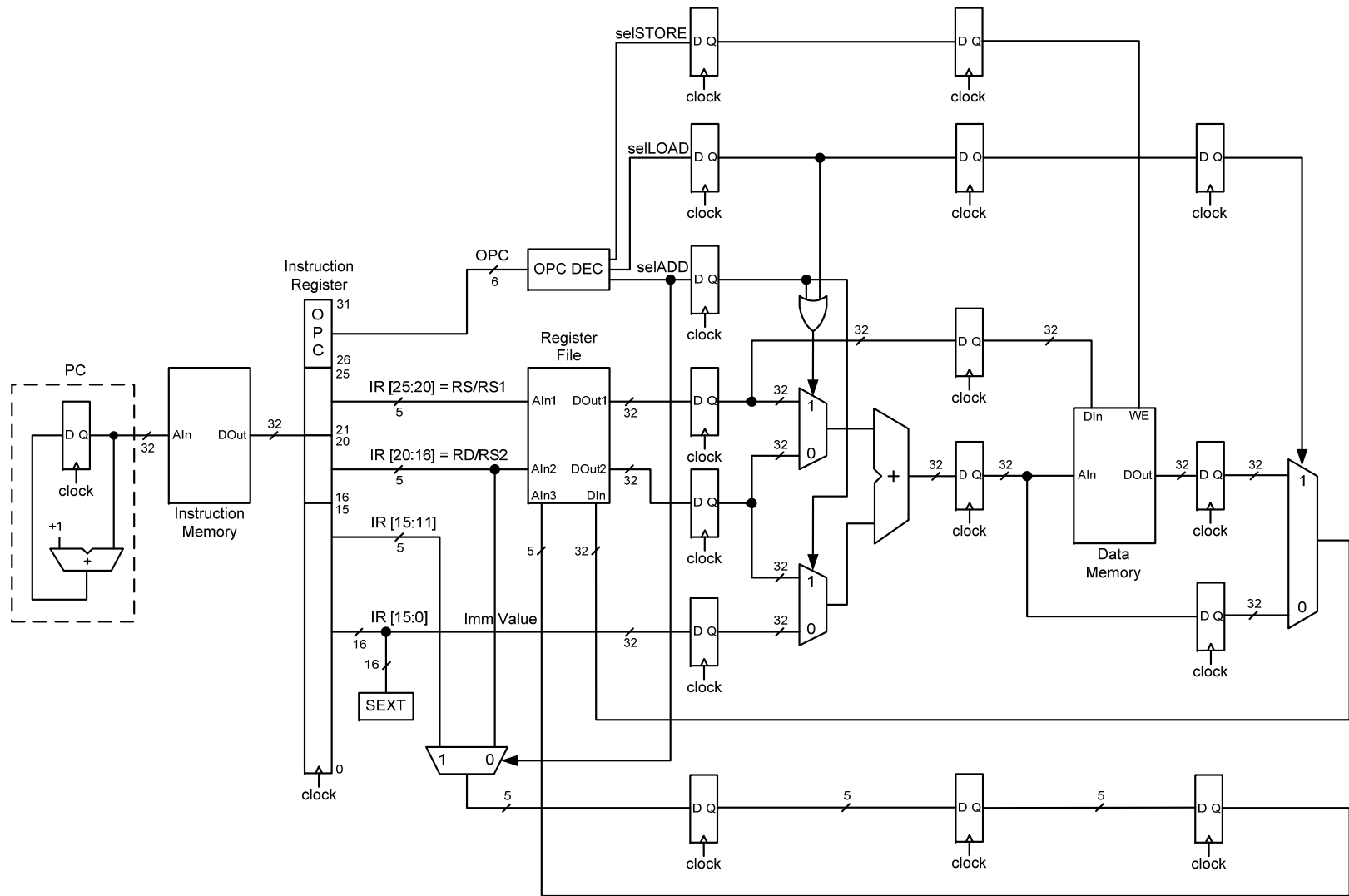
JREG instruction data-path

A DESIGN EXAMPLE: A RISC CPU WITH 4 INSTRUCTIONS

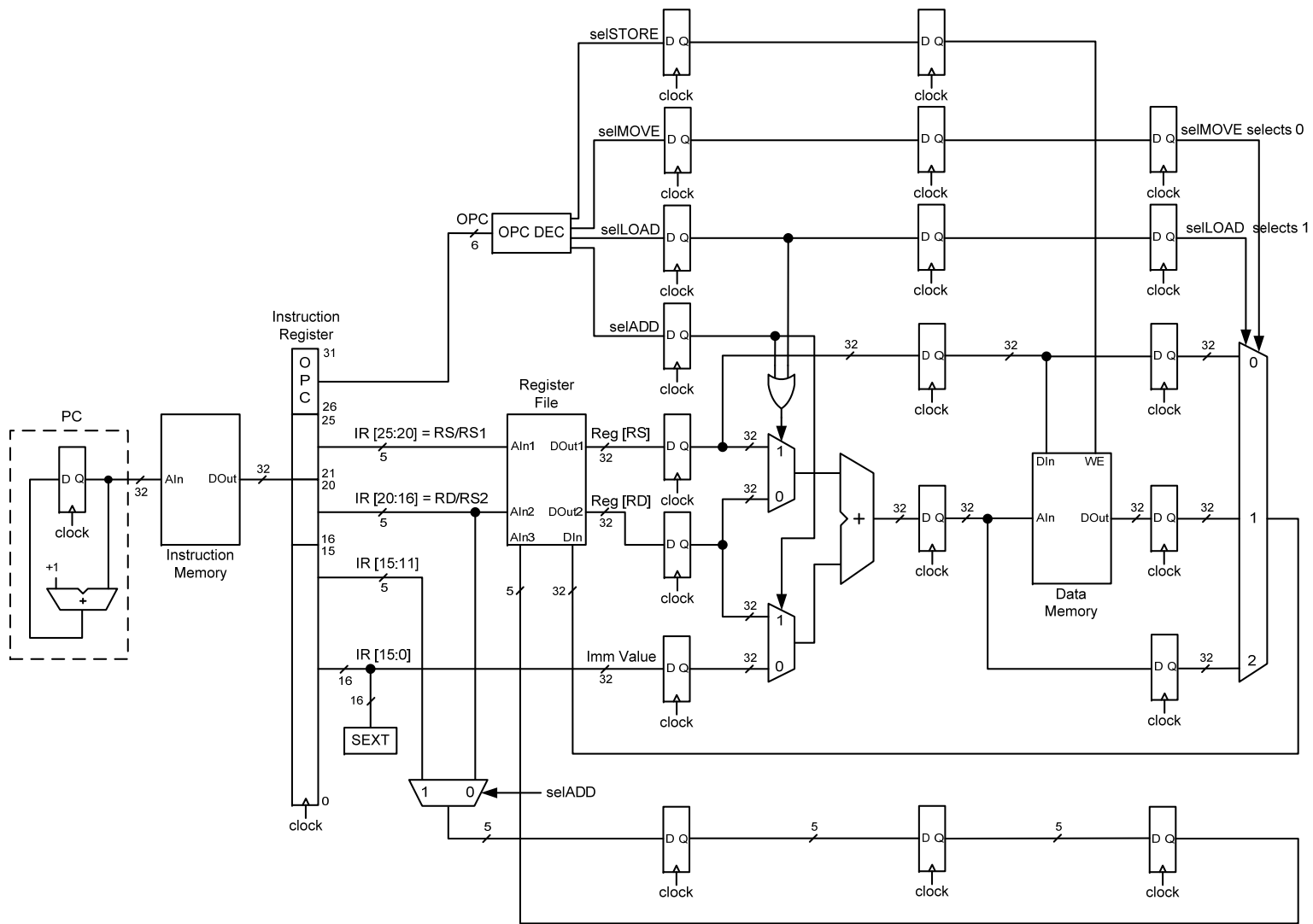
This CPU design example explains how to design the data-path of a RISC CPU that can execute ADD, LOAD, STORE and MOVE instructions.



First step: Construct RISC CPU data-path with ADD and LOAD instructions



Second step: Add STORE instruction to ADD and LOAD instructions



Third step: Add MOVE instruction to ADD, LOAD and STORE instructions