Lecture 09

SFU

# A SINGLE CYCLE PROCESSOR
# PART 2

# Instruction Classes

SFU

There are 59 instructions in rv64I.

I have ignored 8 of these instructions. { fence, fenci.i } (thread synchronization) &

{ csrrs, csrrsi, csrrw, csrrwi, csrrc, csrrci } (control/status register - read/write, read/set, read/clear)

We will design a processor that can execute the 51 remaining instructions.

| | R-Format | I-Format | S-Format | SB-Format | U-Format | UJ-Format |
|---|---|---|---|---|---|---|
| Flow Control (10) | | ebreak, ecall, jalr | | bne, beq, blt, bge, bltu, bgeu | | jal |
| Execution (30) | add, addw, sub, subw | addi, addiw | | | lui, auipc | |
| | and, or, xor | andi, ori, xori | | | | |
| | slt, sltu | slti, sltiu | | | | |
| | sll, srl, sra, sllw, srlw, sraw | slli, srli, srai, slliw, srliw, sraiw | | | | |
| Memory (11) | | lb, lh, lw, ld, lbu, lhu, lwu | sd, sw, sh, sb | | | |

Execution: (**30**)

- Logic: (6) → 3R-format + 3I-format
- Arithmetic: (12) → 6R-format + 4I-format +2U-format
- Shift: (12) → 6R-format + 6I-format

Flow Control: (**10**)

- Branch: (6) → 6SB-format
- Jump: (2) → 2UJ-format
- Environment: (2) → 2I-format

Memory: (**11**)

- Load: (7) → 7I-format
- Store: (4) → 4S-format

# The (51) Instructions

SFU

| | funct6 | funct3 | opcode |
|---|---|---|---|
| sllw | 0000000 | 001 | 0111011 |
| srlw | 0000000 | 101 | 0111011 |
| sraw | 0100000 | 101 | 0111011 |
| slliw | 0000000 | 001 | 0011011 |
| srliw | 0000000 | 101 | 0011011 |
| sraiw | 0100000 | 101 | 0011011 |

| | funct6 | funct3 | opcode |
|---|---|---|---|
| sll | 0000000 | 001 | 0110011 |
| srl | 0000000 | 101 | 0110011 |
| sra | 0100000 | 101 | 0110011 |
| slli | 0000000 | 001 | 0010011 |
| srli | 0000000 | 101 | 0010011 |
| srai | 0100000 | 101 | 0010011 |

| | funct7 | funct3 | opcode |
|---|---|---|---|
| xor | 0000000 | 100 | 0110011 |
| or | 0000000 | 110 | 0110011 |
| and | 0000000 | 111 | 0110011 |
| xori | - | 000 | 0010011 |
| ori | - | 010 | 0010011 |
| andi | - | 011 | 0010011 |

| | funct7 | funct3 | opcode |
|---|---|---|---|
| add | 0000000 | 000 | 0110011 |
| addw | 0000000 | 000 | 0111011 |
| sub | 0100000 | 000 | 0110011 |
| subw | 0100000 | 000 | 0111011 |
| slt | 0000000 | 010 | 0110011 |
| sltu | 0000000 | 011 | 0110011 |

| | funct7 | funct3 | opcode |
|---|---|---|---|
| addi | - | 000 | 0010011 |
| addiw | - | 000 | 0011011 |
| slti | - | 010 | 0010011 |
| sltiu | - | 011 | 0011011 |
| lui | - | - | 0110111 |
| auipc | - | - | 0010111 |

| | funct7 | funct3 | opcode |
|---|---|---|---|
| lb | - | 000 | 0000011 |
| lh | - | 001 | 0000011 |
| lw | - | 010 | 0000011 |
| ld | - | 011 | 0000011 |
| lbu | - | 100 | 0000011 |
| lhu | - | 101 | 0000011 |
| lwu | - | 110 | 0000011 |

| | funct7 | funct3 | opcode |
|---|---|---|---|
| sb | - | 000 | 0100011 |
| sh | - | 001 | 0100011 |
| sw | - | 010 | 0100011 |
| sd | - | 011 | 0100011 |

| | funct7 | funct3 | opcode |
|---|---|---|---|
| beq | - | 000 | 1100011 |
| bne | - | 001 | 1100011 |
| blt | - | 100 | 1100011 |
| bge | - | 101 | 1100011 |
| bltu | - | 110 | 1100011 |
| bgeu | - | 111 | 1100011 |

| | | | |
|---|---|---|---|
| ebreak | imm=0x000 | 000 | 1110011 |
| ecall | imm=0x001 | 000 | 1110011 |

| | | | |
|---|---|---|---|
| jal | - | - | 1101111 |
| jalr | - | 000 | 1100111 |

# The Primary Elements of a Processor Datapath

SFU

So far we have the two most important elements of a Processor Datapath.

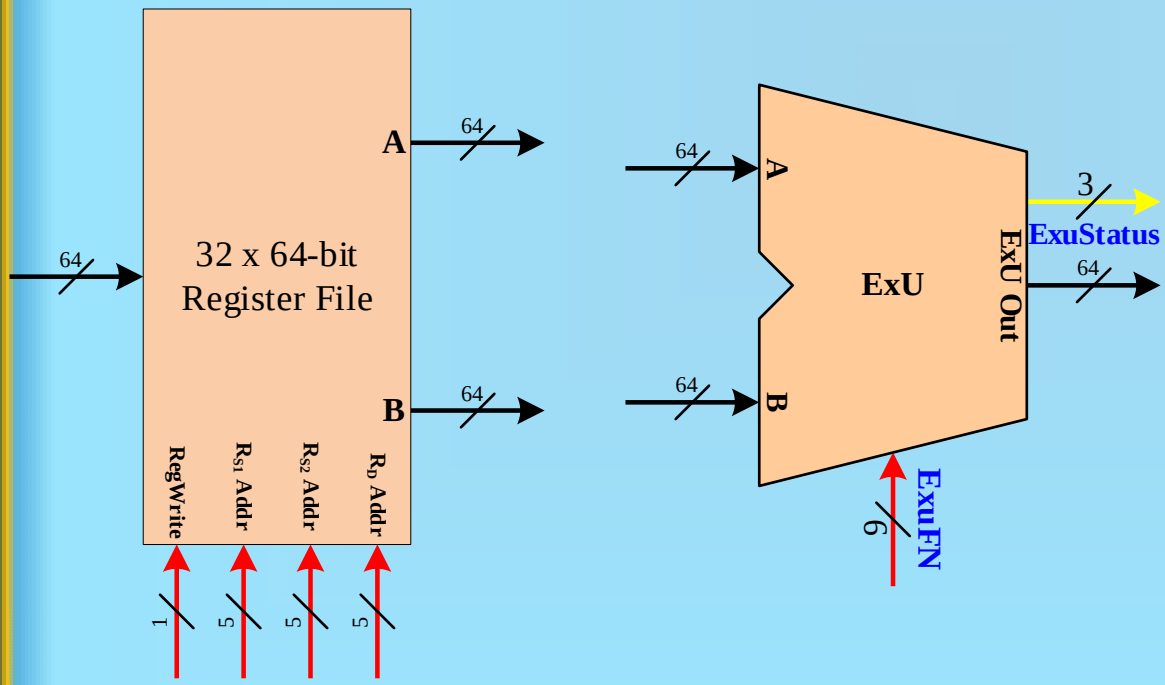
The Execution Unit (**ExU**) and the Register File (**RegF**)

There are 9-bits that control the behaviour of the execution unit, FuncClass, LogicFN, ShiftFN, ExtWord, AddnSub and NotA. (We don't actually need NotA )

Let us define a 9-bit signal,


```
ExuFN <= NotA & FuncClass & LogicFN & ShiftFN & AddnSub & ExtWord;
```

```
ExuStatus <= Cout,& Ovfl & Zero;
```


| FuncClass | | operation |
|---|---|---|
| 0 | 0 | sltu |
| 0 | 1 | slt |
| 1 | 0 | shift/arith |
| 1 | 1 | logic |

| LogicFn | | operation |
|---|---|---|
| 0 | 0 | Lui |
| 0 | 1 | A xor B |
| 1 | 0 | A or B |
| 1 | 1 | A and B |

| ShiftFN | | operation |
|---|---|---|
| 0 | 0 | arith |
| 0 | 1 | sll |
| 1 | 0 | srl |
| 1 | 1 | sra |

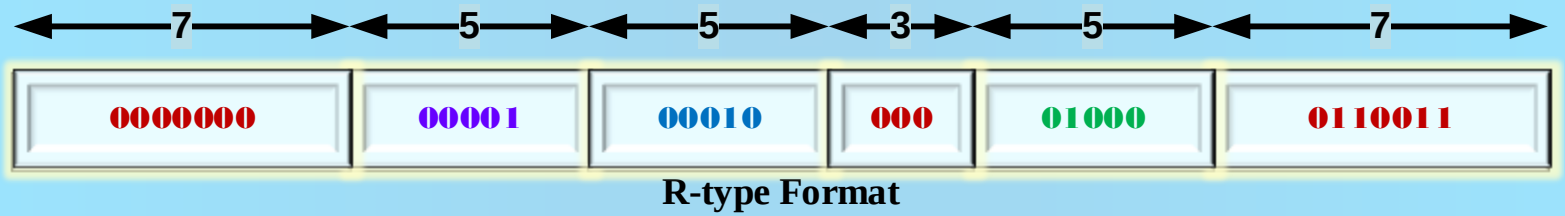
# Simplified Datapath

High-Level Language Statement **a = b + c;**

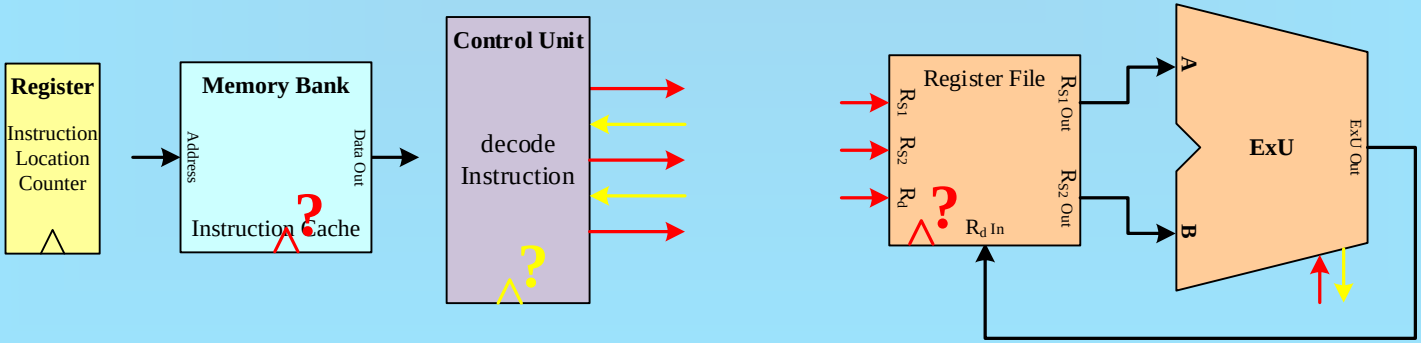
Assembly Language Instruction **add x8, x2, x1**

Machine Language Instruction **0000000 00001 00010 000 01000 0110011**

0x **0011 0433**




**R-type Format**

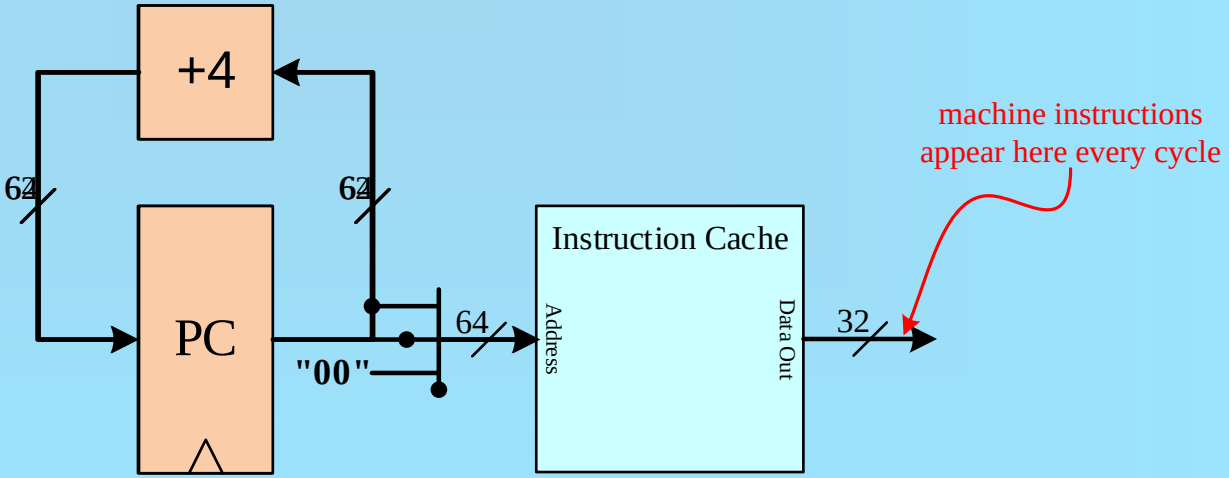


# The Instruction Cycle

The Datapath contains a special Register called the **Program Counter, PC**.

- This register holds a 64-bit value that represents a memory address of a location in the Instruction Memory Unit. ( The **Lowest bit** is always **‘0’** because the RISC-V specification requires that Instructions must be **16-bit halfword-aligned** )
- Instructions are only read from the Instruction Memory Unit and thus the behaviour is similar to that of a combinational circuit. (The clocking scheme is not an issue.)

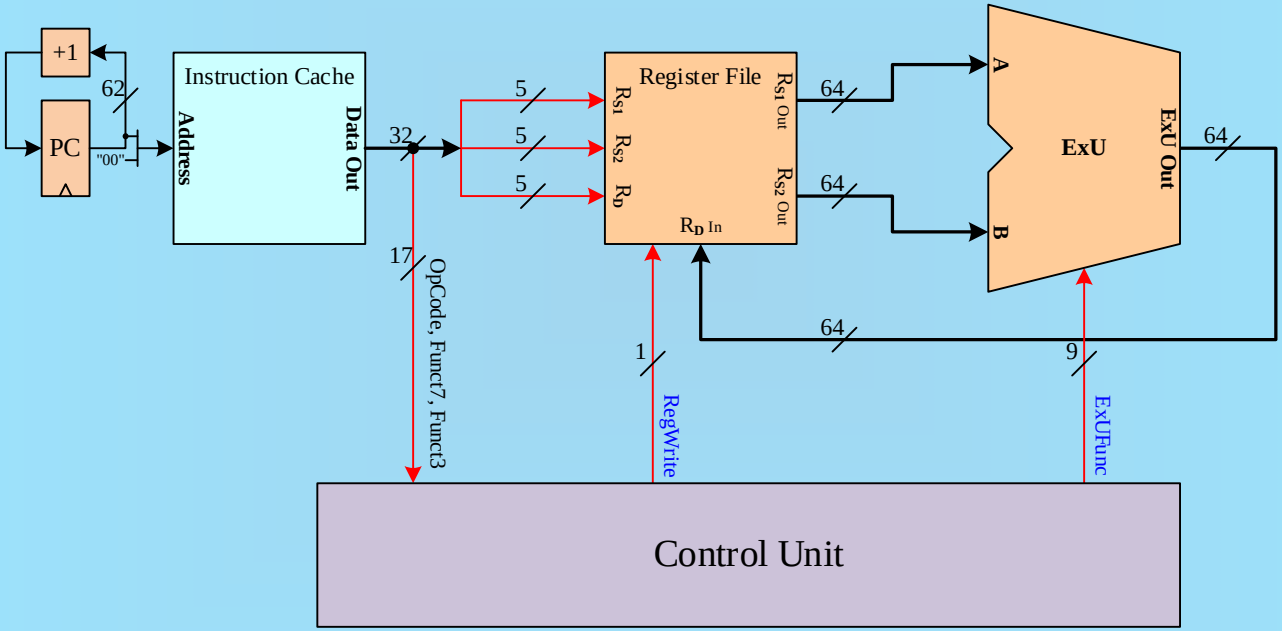
The value inside the PC is used to calculate a new value for the PC each cycle. The new value represents the location of the instruction to be processed in the next cycle.
The new value will be PC ← PC + 4, for all instructions except Flow Control Instructions



# R-Format "Execute" Instructions

Execute Instructions: (30 instructions)

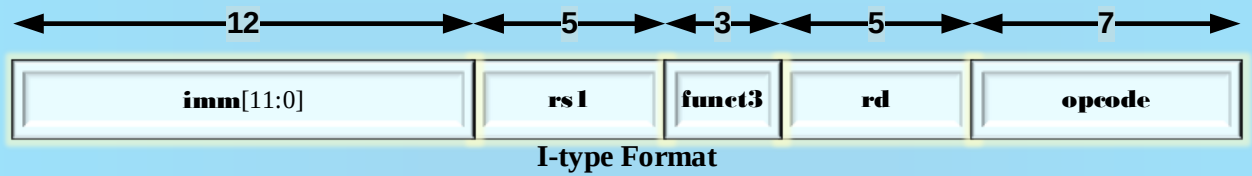
- pass operand data through an execution unit and store a result back in a destination register.
- 15 of these instructions are **R-Format**; with operands coming from two source registers.



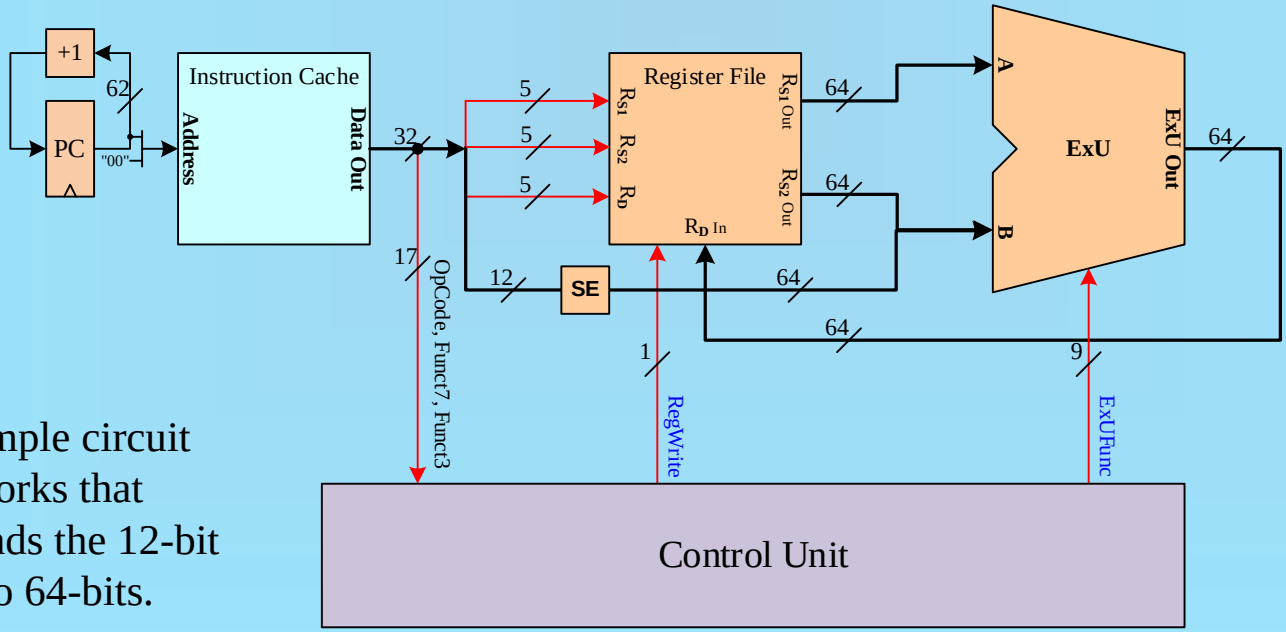
# I-Format "Execute" Instructions

Execute Instructions: (30 instructions)

- pass operand data through an execution unit and store a result back in a destination register.
- 13 of these instructions are **I-Format**; with operands coming from one source registers , and a12-bit sign-extended immediate constant.

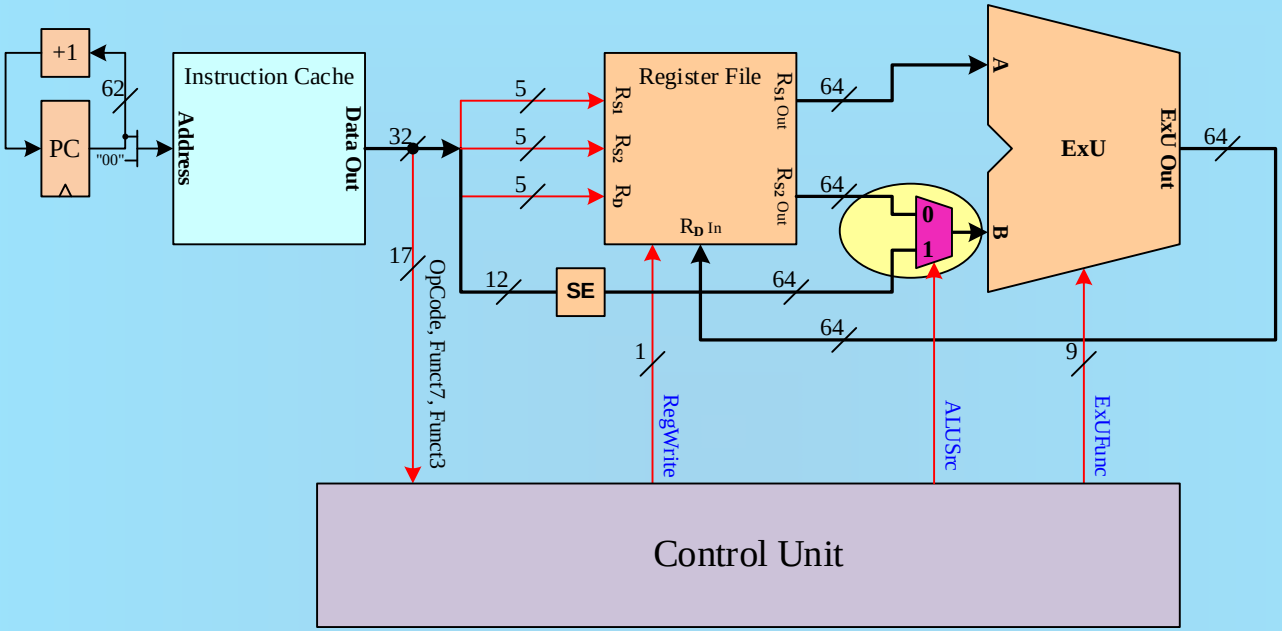



I-type Format




SE is a simple circuit contains forks that sign-extends the 12-bit constant to 64-bits.

# Execution Unit Source MUX

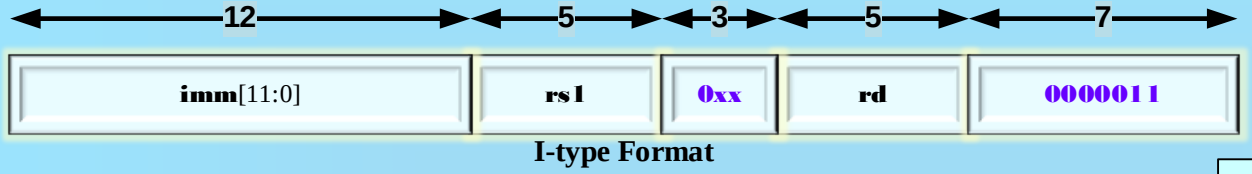



We have now introduced one more control signal, AluSrc, into the design so that the datapath can execute both **R format** and **I format** instructions.

# I-Format Data Memory "Read" Access

Memory Instructions: (7 load and 4 store instructions)

- Load instructions are **I-Format** instructions. **ld** $R_D \leftarrow$ mem( $R_{S1}$ + imm )

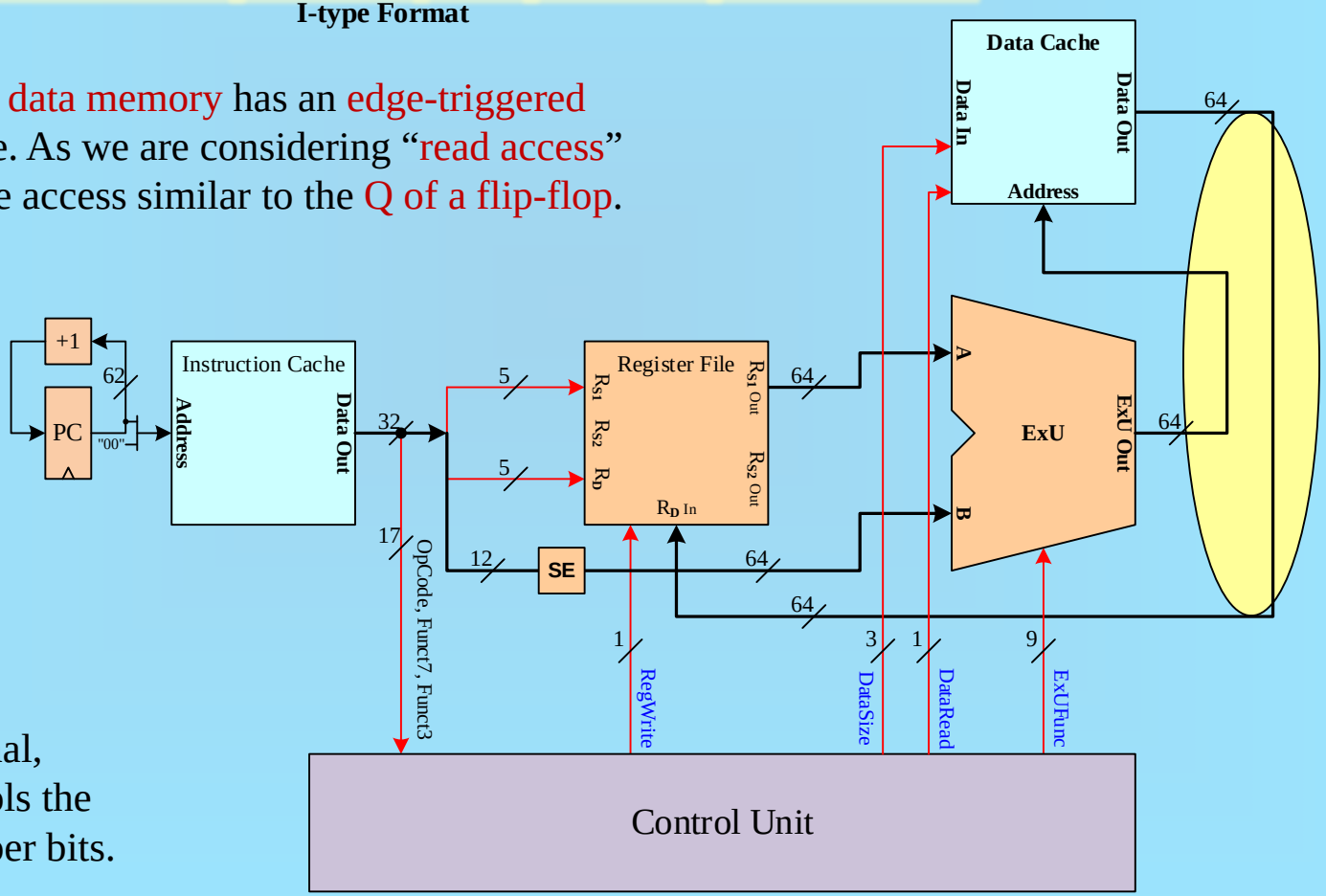



**I-type Format**

Assume that the data memory has an edge-triggered clocking scheme. As we are considering "read access" we may view the access similar to the Q of a flip-flop.

| DataSize | | | Read Op |
|---|---|---|---|
| 0 | 0 | 0 | Ubyte |
| 0 | 0 | 1 | Uhalf |
| 0 | 1 | 0 | Uword |
| 0 | 1 | 1 | double |
| 1 | 0 | 0 | Sbyte |
| 1 | 0 | 1 | Shalf |
| 1 | 1 | 0 | Sword |
| 1 | 1 | 1 | double |

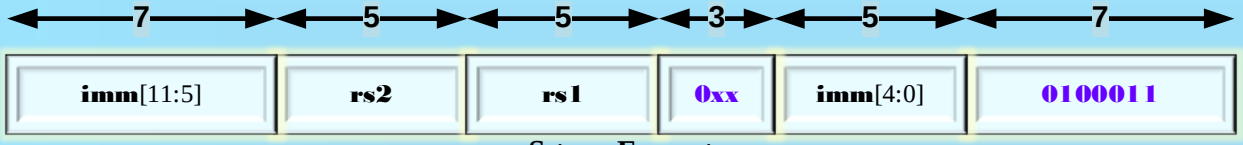
A new 3-bit signal, DataSize, controls the extension of upper bits.



# S-Format Data Memory "Write" Access

Memory Instructions: (7 load and 4 store instructions)

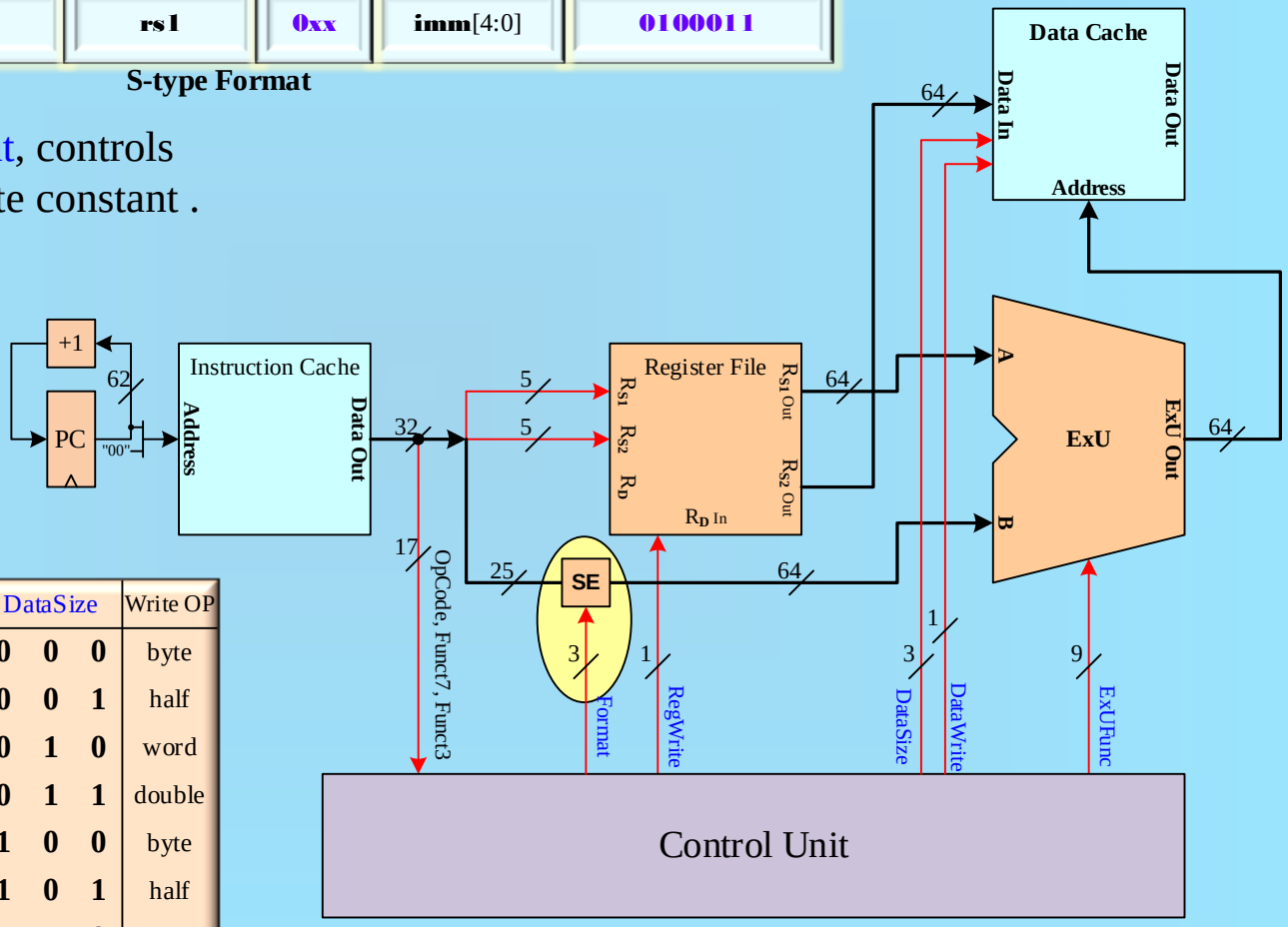
- Store instructions are **S-Format** instructions. **sd** mem( $R_{S1}$ + imm ) ← $R_{S2}$




**S-type Format**

A new 3-bit signal, Format, controls formation of the immediate constant .

| Format | | | operation |
|---|---|---|---|
| 0 | 0 | 0 | R-type |
| 0 | 0 | 1 | I-type |
| 0 | 1 | 0 | S-type |
| 0 | 1 | 1 | SB-type |
| 1 | 0 | 0 | U-type |
| 1 | 0 | 1 | UJ-type |
| 1 | 1 | 0 | xx |
| 1 | 1 | 1 | xx |




| DataSize | | | Write OP |
|---|---|---|---|
| 0 | 0 | 0 | byte |
| 0 | 0 | 1 | half |
| 0 | 1 | 0 | word |
| 0 | 1 | 1 | double |
| 1 | 0 | 0 | byte |
| 1 | 0 | 1 | half |
| 1 | 1 | 0 | word |
| 1 | 1 | 1 | double |

Assume that the Data Memory Unit ignores the **msb** of DataSize for data write operations.

# Adding The Destination Register MUX

The Data Memory Operations can be combined with the Execution Operations by inserting a MUX that selects the source for storing to the Register File.
This MUX is controlled by a control signal called RegInSrc.