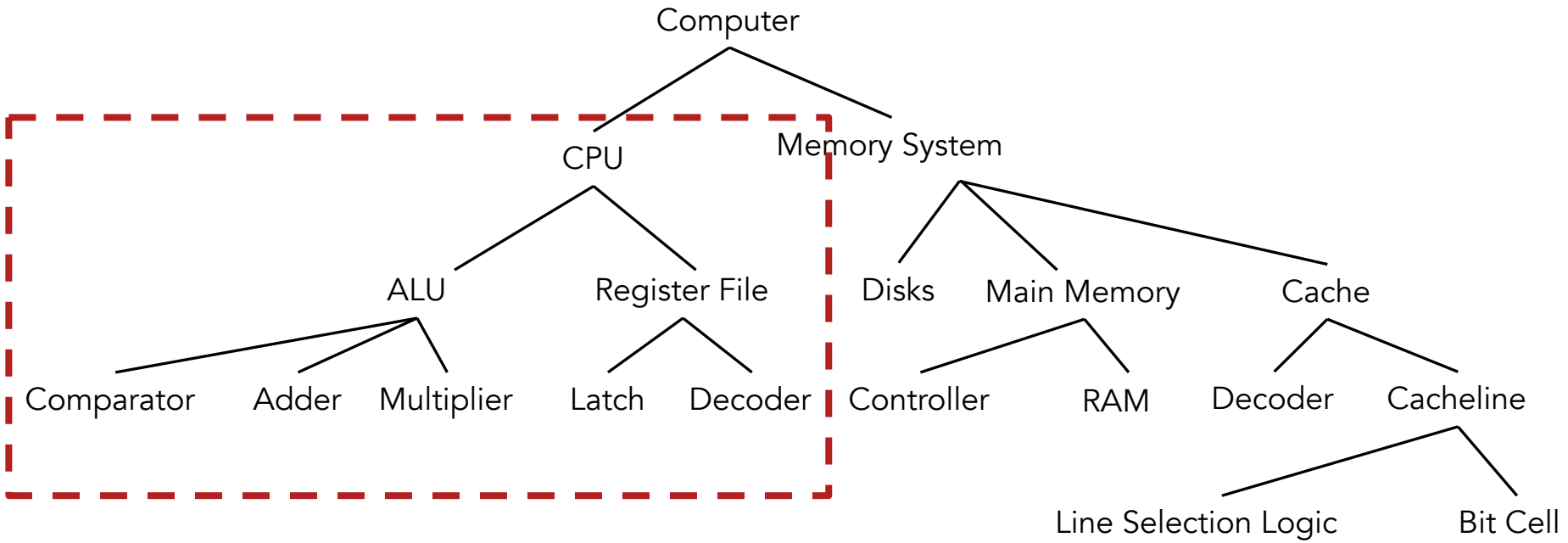
# CSE 520
# Computer Architecture II

## Review
## Central Processing Unit (CPU) Architecture Design

Prof. Michel A. Kinsy

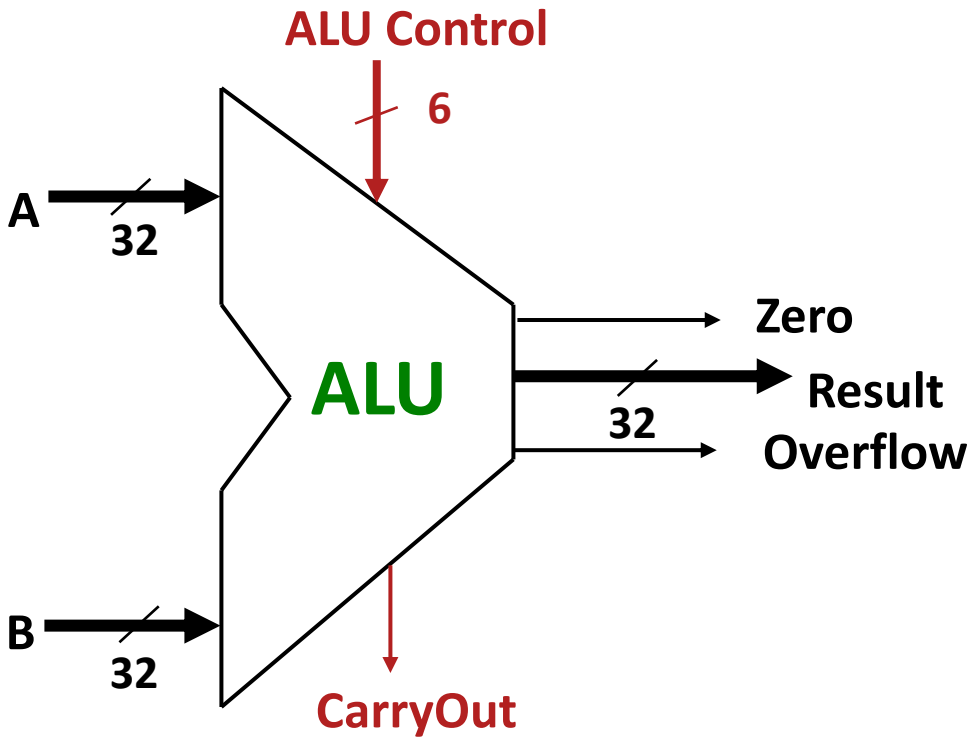
# Computing: Computer Architecture

- The DNA of Modern Computing

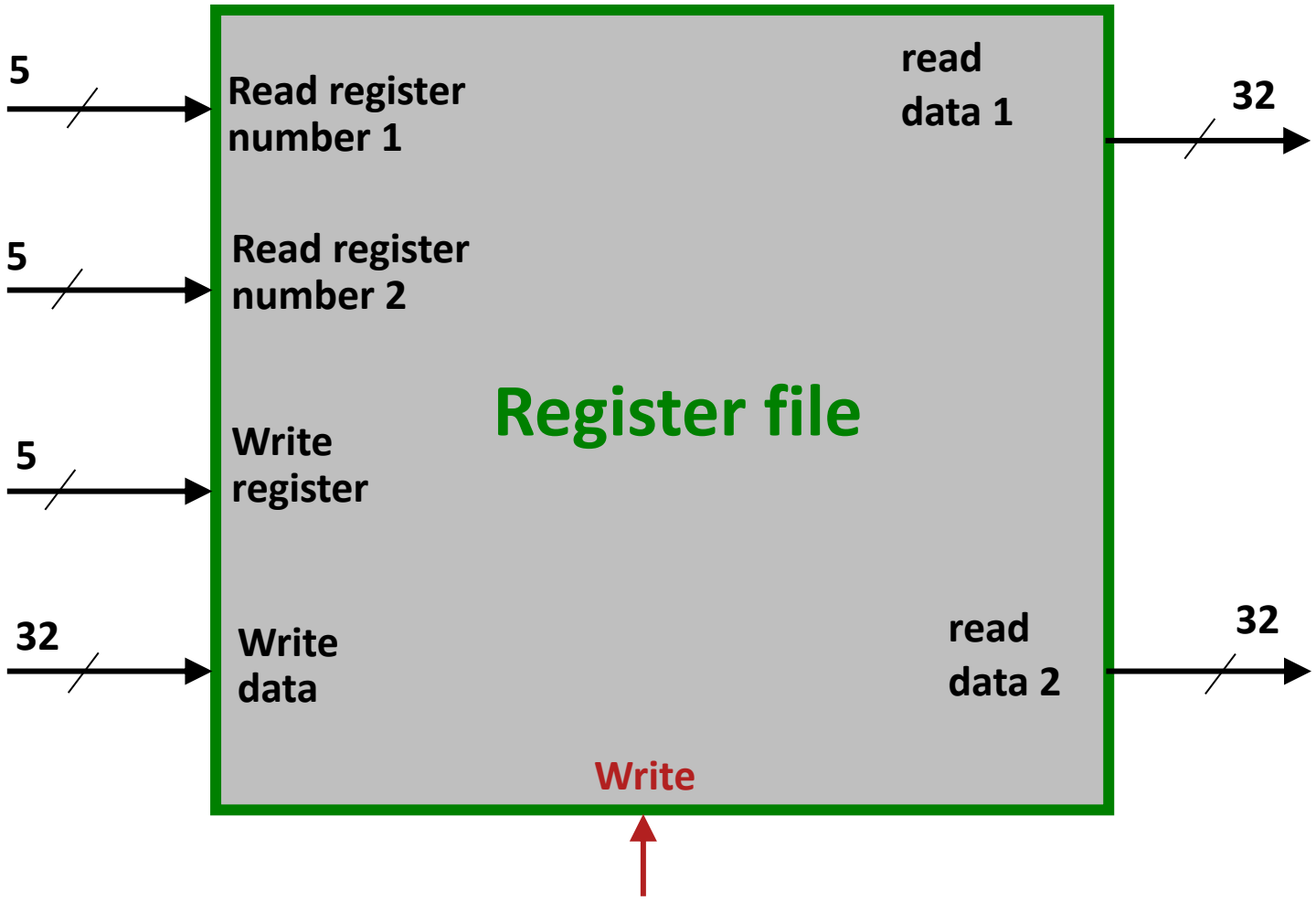


# Arithmetic Logic Unit (ALU)

- Arithmetic circuits is built in a hierarchical fashion
- Input: data and operation to perform
- Output: result of operation and status information

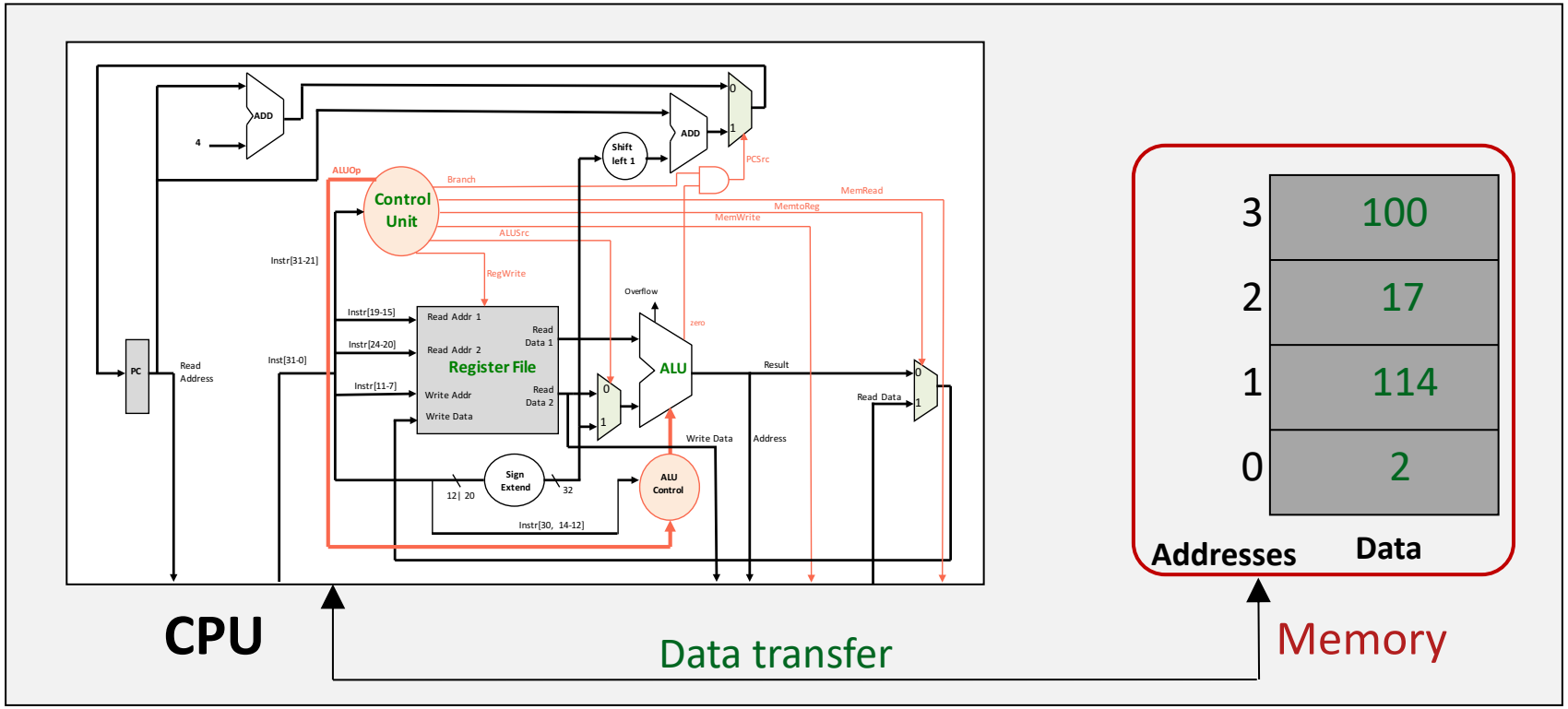


# Register File – RISC-V Support



# Computer Architecture

- Basic view of computer organization

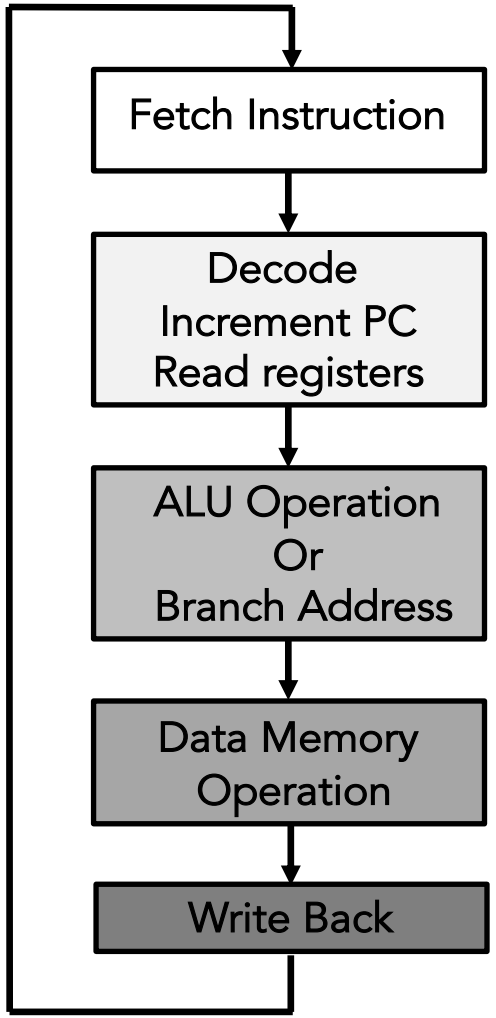


# Central Processing Unit (CPU)

- Central Processing Unit (CPU) Organization
  - CPU = Control Unit + ALU + Registers
  - Control Unit: monitors and directs sequences of instructions
  - ALU (Arithmetic-Logic Unit): performs arithmetic and logical operations

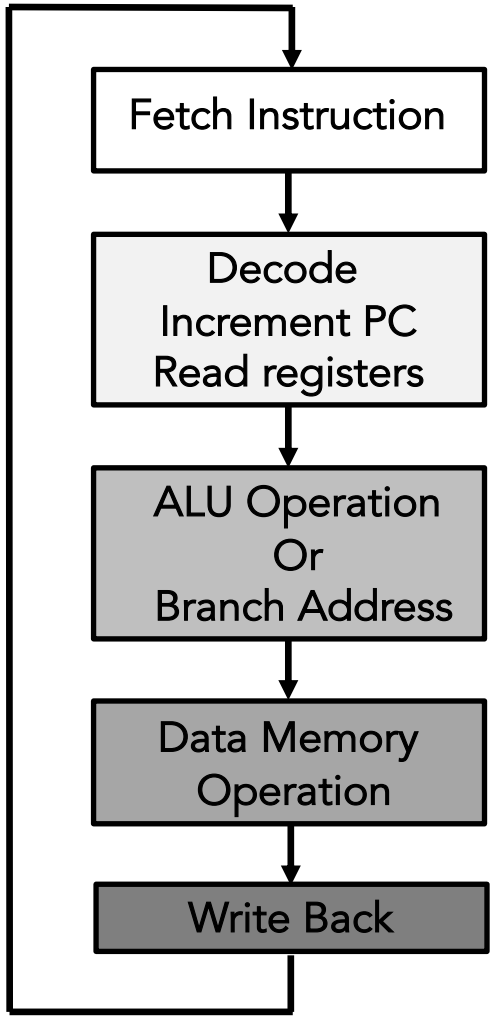
# Central Processing Unit (CPU)

- Central Processing Unit (CPU) Organization
- CPU Execution Process
  1. Fetch Instruction
  2. Decode Instruction
  3. Execute Operation
  4. Memory Operation
  5. Register Writeback Operation



# Central Processing Unit (CPU)

- Central Processing Unit (CPU) Organization
- CPU Execution Process
  1. Fetch Instruction
     - Read IM[PC]



# Instruction Fetch

- Program Counter (PC) sends current instruction address to memory to fetch instruction to execute

```
@54        // <main>
fe010113   // 00000150    addi   sp,sp,-32
00112e23   // 00000154    sw     ra,28(sp)
00812c23   // 00000158    sw     s0,24(sp)
02010413   // 0000015c    addi   s0,sp,32
40000793   // 00000160    addi   a5,zero,102
4fef42623  // 00000164    sw     a5,-20(s0)
03000793   // 00000168    addi   a5,zero,4
8fef42423  // 0000016c    sw     a5,-24(s0)
fe842583   // 00000170    lw     a1,-24(s0)
fec42503   // 00000174    lw     a0,-20(s0)
00000097   // 00000178    auipc  ra,0x0
f4c080e7   // 0000017c    jalr   ra,-180(ra) # c4 <gcd>
fea42223   // 00000180    sw     a0,-28(s0)
fe442783   // 00000184    lw     a5,-28(s0)
00078513   // 00000188    addi   a0,a5,0
01c12083   // 0000018c    lw     ra,28(sp)
01812403   // 00000190    lw     s0,24(sp)
```

# Instruction Fetch

- Program Counter (PC) sends current instruction address to memory to fetch instruction to execute

```
@54        // <main>
fe010113  // 00000150   addi   sp,sp,-32
00112e23  // 00000154   sw     ra,28(sp)
00812c23  // 00000158   sw     s0,24(sp)
02010413  // 0000015c   addi   s0,sp,32
40000793  // 00000160   addi   a5,zero,102
4fef42623  // 00000164   sw     a5,-20(s0)
03000793  // 00000168   addi   a5,zero,4
8fef42423  // 0000016c   sw     a5,-24(s0)
fe842583   // 00000170   lw     a1,-24(s0)
fec42503   // 00000174   lw     a0,-20(s0)
00000097  // 00000178   auipc  ra,0x0
f4c080e7   // 0000017c   jalr   ra,-180(ra) # c4 <gcd>
fea42223   // 00000180   sw     a0,-28(s0)
fe442783   // 00000184   lw     a5,-28(s0)
00078513   // 00000188   addi   a0,a5,0
01c12083   // 0000018c   lw     ra,28(sp)
01812403   // 00000190   lw     s0,24(sp)
```

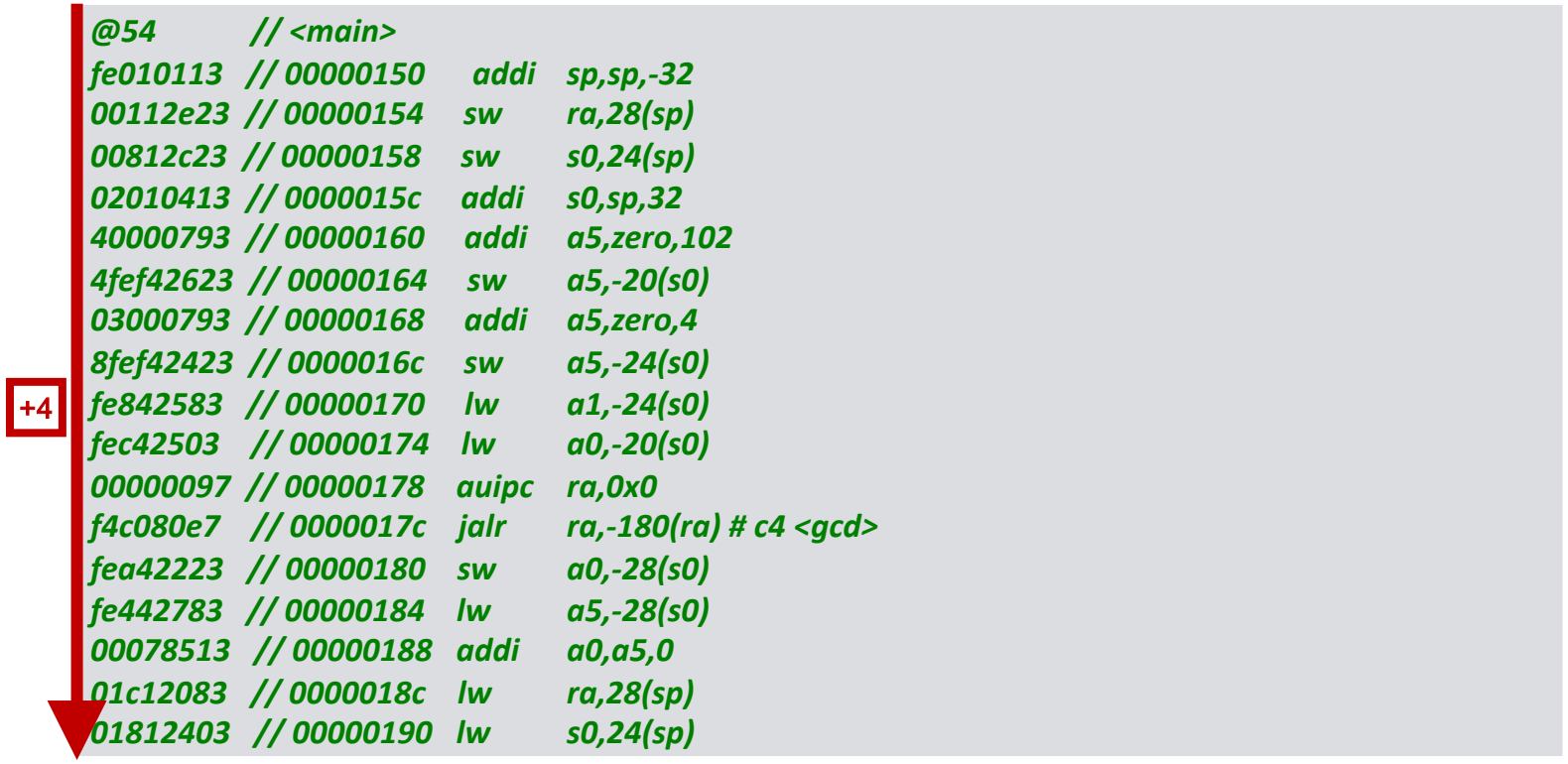
# Instruction Fetch

- Program Counter (PC) sends current instruction address to memory to fetch instruction to execute

```
@54        // <main>
fe010113   // 00000150    addi   sp,sp,-32
00112e23   // 00000154    sw     ra,28(sp)
00812c23   // 00000158    sw     s0,24(sp)
02010413   // 0000015c    addi   s0,sp,32
40000793   // 00000160    addi   a5,zero,102
4fef42623  // 00000164    sw     a5,-20(s0)
03000793   // 00000168    addi   a5,zero,4
8fef42423  // 0000016c    sw     a5,-24(s0)
fe842583   // 00000170    lw     a1,-24(s0)
fec42503   // 00000174    lw     a0,-20(s0)
00000097   // 00000178    auipc  ra,0x0
f4c080e7   // 0000017c    jalr   ra,-180(ra) # c4 <gcd>
fea42223   // 00000180    sw     a0,-28(s0)
fe442783   // 00000184    lw     a5,-28(s0)
00078513   // 00000188    addi   a0,a5,0
01c12083   // 0000018c    lw     ra,28(sp)
01812403   // 00000190    lw     s0,24(sp)
```

# Instruction Fetch

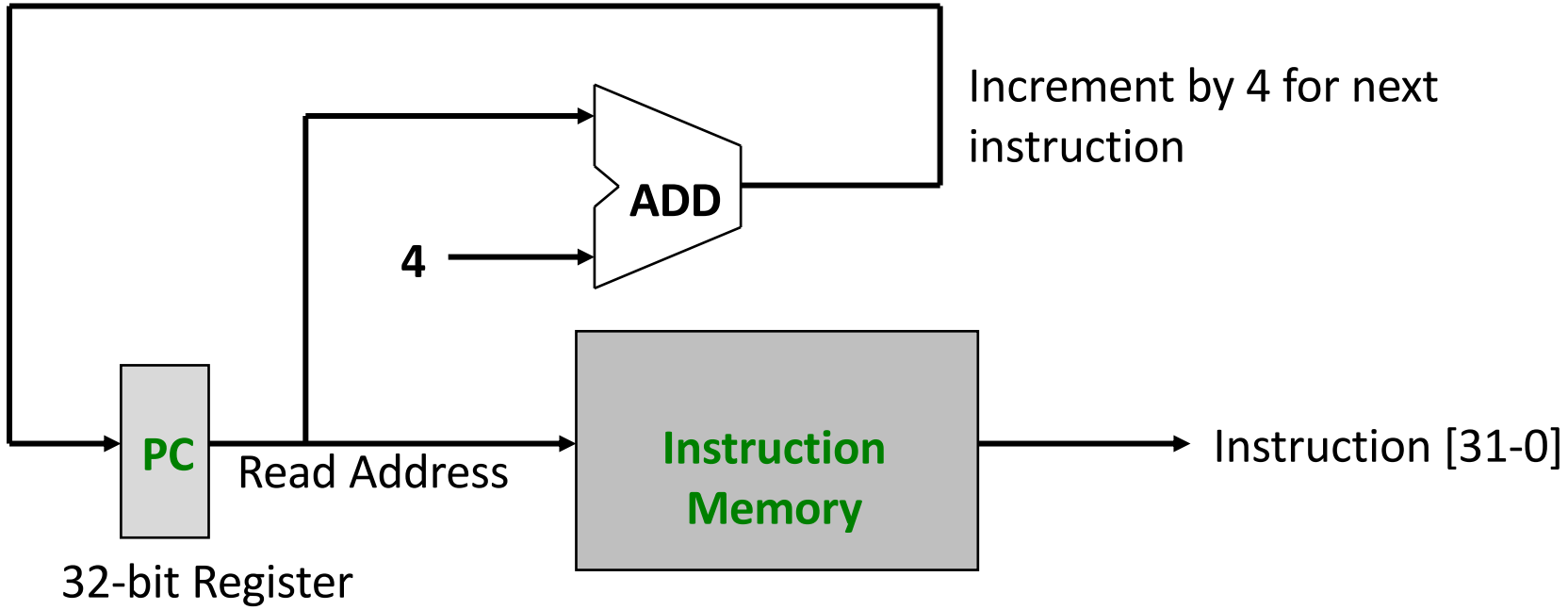
- Program Counter (PC) sends current instruction address to memory to fetch instruction to execute




```
@54          // <main>
fe010113   // 00000150    addi   sp,sp,-32
00112e23   // 00000154    sw     ra,28(sp)
00812c23   // 00000158    sw     s0,24(sp)
02010413   // 0000015c    addi   s0,sp,32
40000793   // 00000160    addi   a5,zero,102
4fef42623  // 00000164    sw     a5,-20(s0)
03000793   // 00000168    addi   a5,zero,4
8fef42423  // 0000016c    sw     a5,-24(s0)
fe842583   // 00000170    lw     a1,-24(s0)
fec42503   // 00000174    lw     a0,-20(s0)
00000097   // 00000178    auipc  ra,0x0
f4c080e7   // 0000017c    jalr   ra,-180(ra) # c4 <gcd>
fea42223   // 00000180    sw     a0,-28(s0)
fe442783   // 00000184    lw     a5,-28(s0)
00078513   // 00000188    addi   a0,a5,0
01c12083   // 0000018c    lw     ra,28(sp)
01812403   // 00000190    lw     s0,24(sp)
```

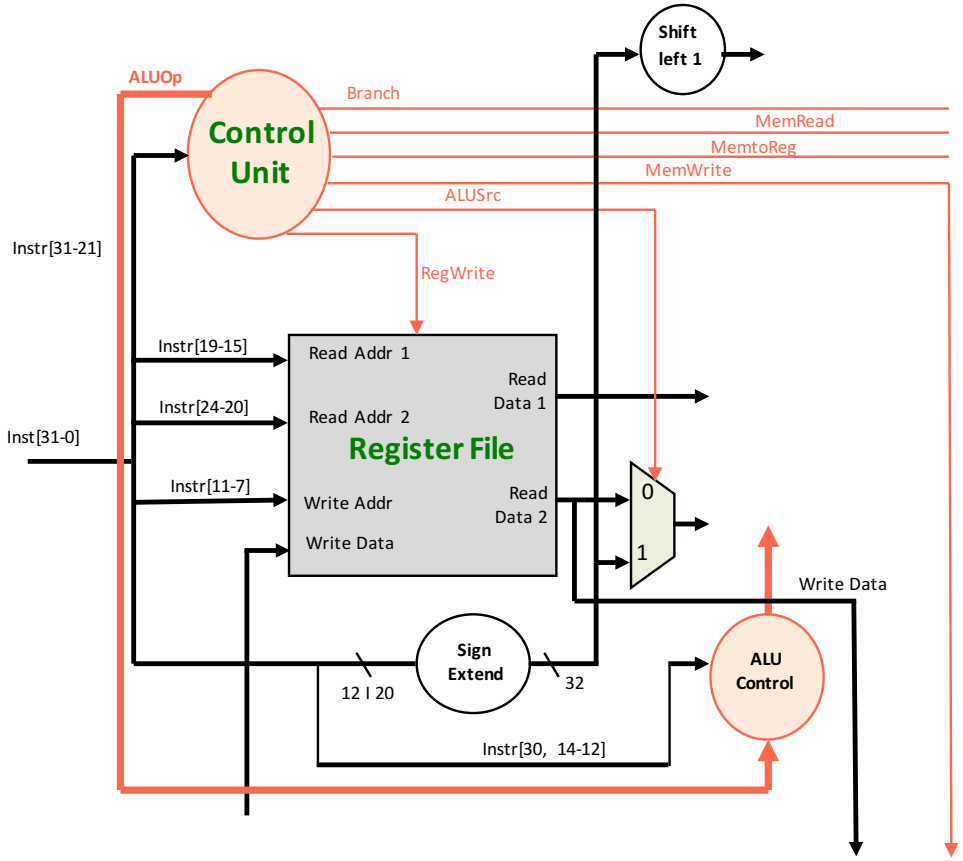
# Instruction Fetch

- Program Counter (PC) sends current instruction address to memory to fetch instruction to execute



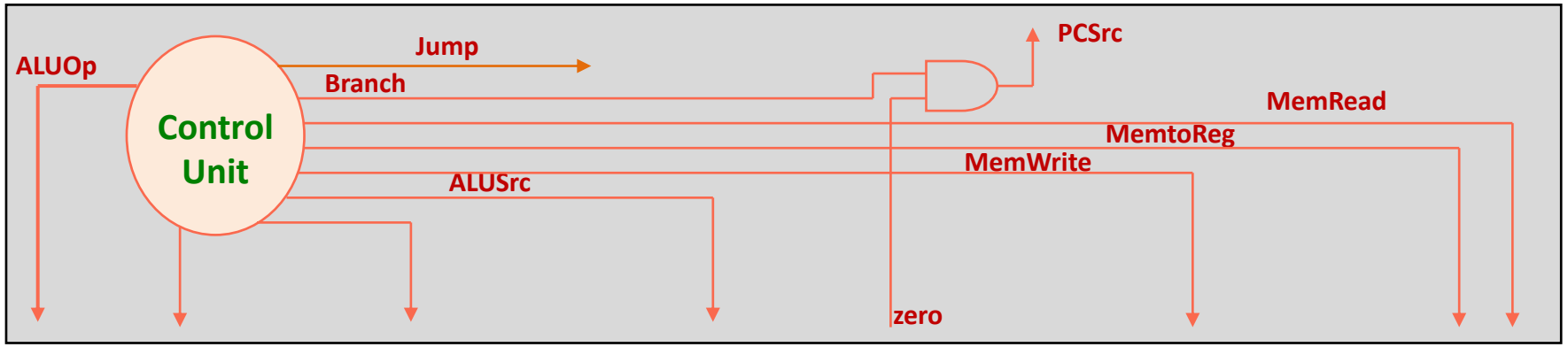
# Instruction Decode

- Identity instruction class
  - If R-Type, execution will need to:
    - Read two register operands
  - If Load/Store, execution will need to:
    - Read register operands
    - Calculate address using 16-bit offset

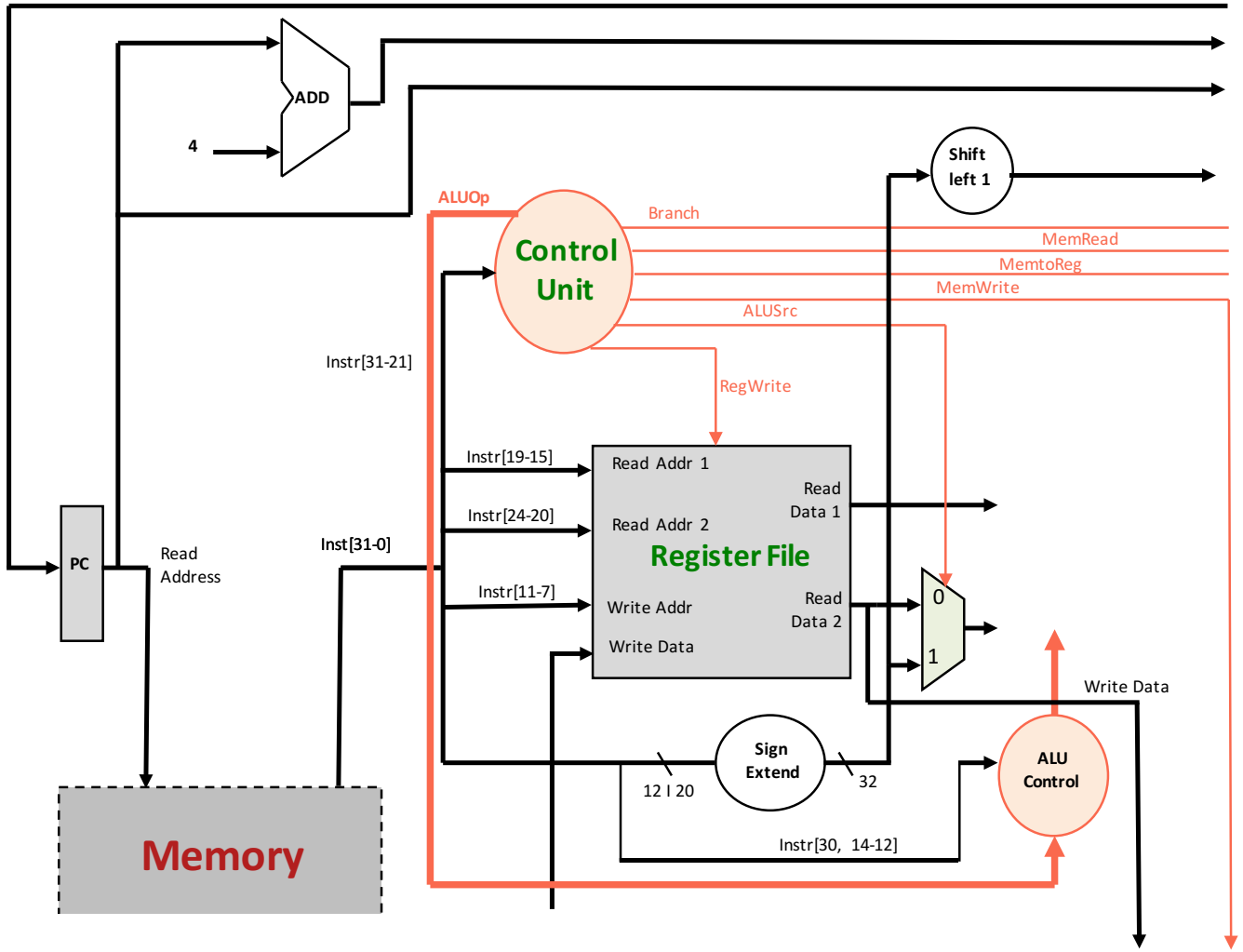


# Instruction Decode

- Identity instruction class
  - If Branch, execution will need to:
    - Read register operands
    - Compare operands
      - Use ALU to subtract and check Zero output
  - Calculate target address
    - Sign-extend displacement
    - Shift left 1 place (half-word displacement)
    - Add to PC + 4

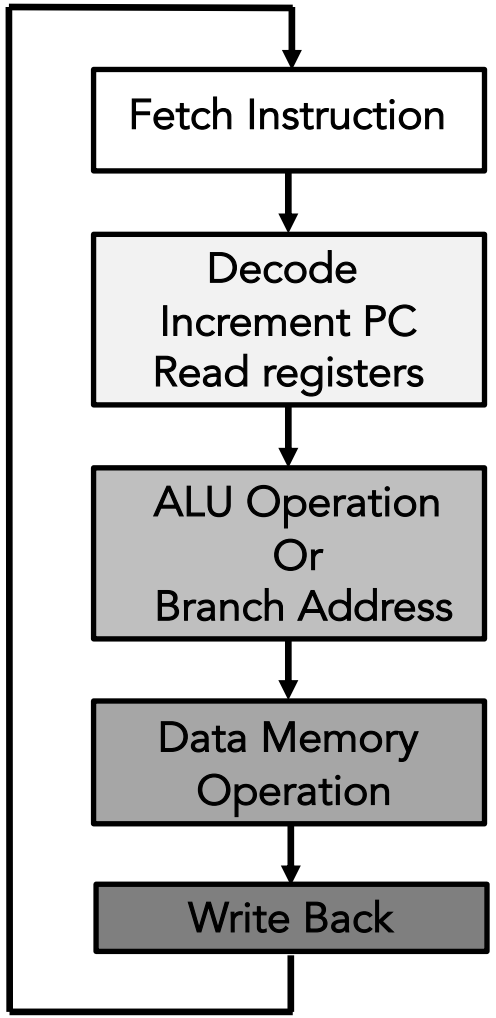


# Instruction Fetch & Decode



# Central Processing Unit (CPU)

- Central Processing Unit (CPU) Organization
- CPU Execution Process
  1. Fetch Instruction
  2. Decode Instruction
  3. Execute Operation
  4. Memory Operation
  5. Register Writeback Operation

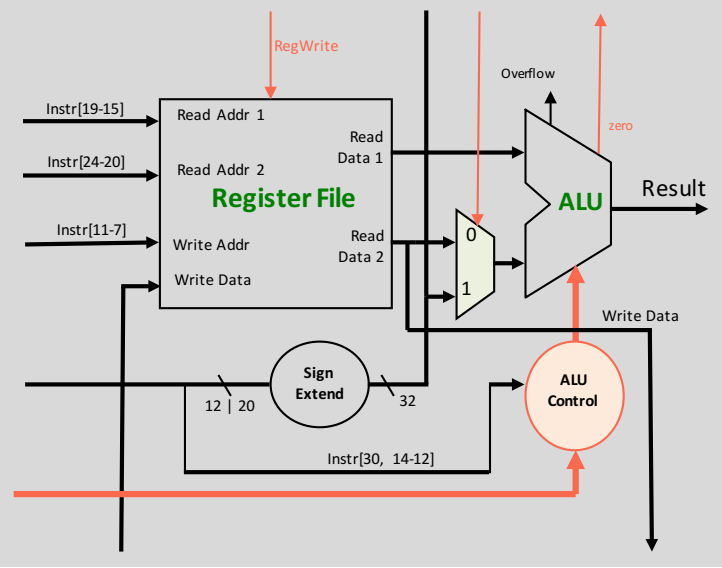


# Execute Operation

- The Arithmetic Logic Unit (ALU) is at the center of the CPU operation execution
  - ALU operation is based on instruction type and function code
    - Performs subtraction for branches (beq)
    - Performs no operation for jumps
    - Performs the operation is specified by the function field for R-type instructions
  - ALU Control unit will have the following inputs:
    - 3-bit control field called ALUOp
    - Funct3 and funct7 function fields

# Execute Operation

- The Arithmetic Logic Unit (ALU) is at the center of the CPU operation execution
  - ALU operation is based on instruction type and function code
  - ALU Control unit will have the following inputs:



# ALU Operation

| Instruction Opcode | Instruction Operation | ALU Op | funct3 | funct7 | Effective ALU Operation | ALU Control |
|---|---|---|---|---|---|---|
| LW | Load word | 100 | 010 | xxxxxxx | Add | 000000 |
| SW | Store word | 101 | 010 | xxxxxxx | Add | 000000 |
| BEQ | Branch equal | 010 | 000 | xxxxxxx | Subtract | 001000 |
| R-type | ADD | 000 | 000 | 0000000 | Add | 000000 |
| R-type | SUB | 000 | 000 | 0100000 | Subtract | 001000 |
| R-type | AND | 000 | 111 | 0000000 | and | 000111 |
| R-type | OR | 000 | 110 | 0000000 | Or | 000110 |
| R-type | SLT | 000 | 010 | 0000000 | Set-less-than | 000010 |

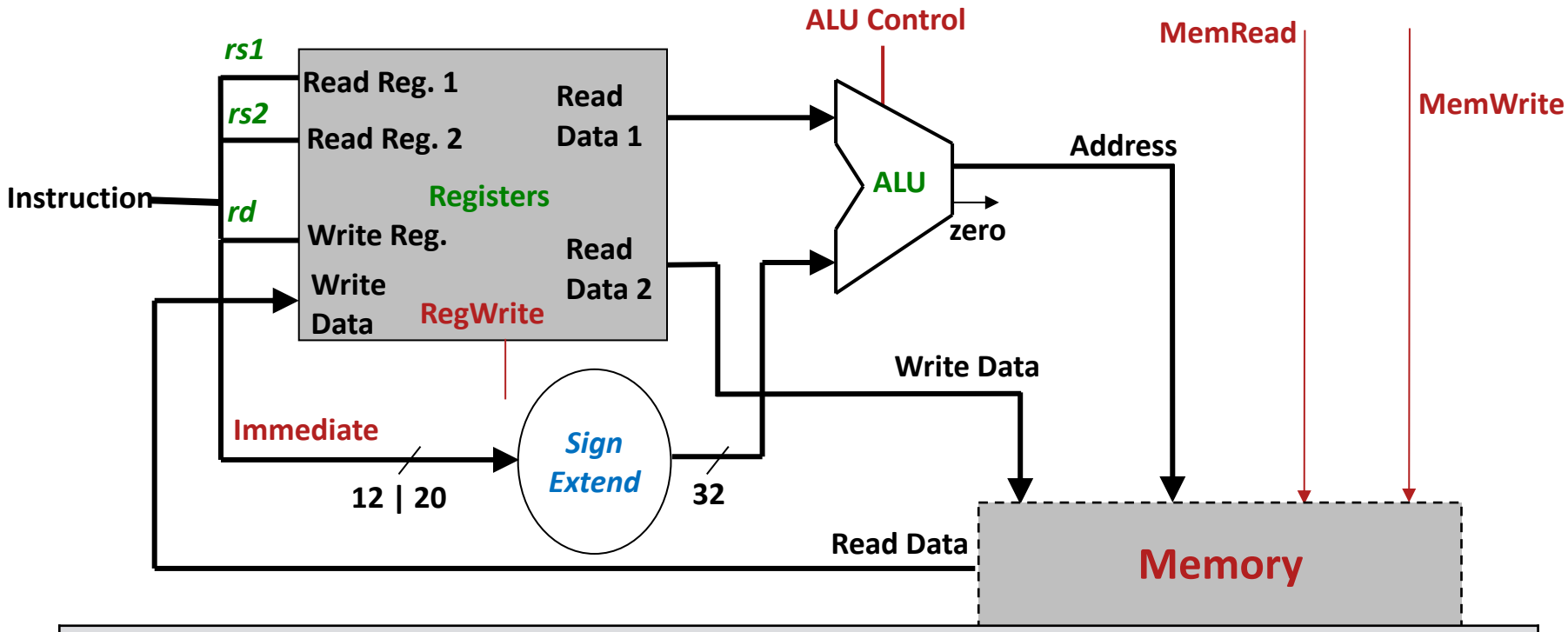
# Memory Operation

- For RISC-V Load and Store are the only two memory instructions
  - Recall:
    - RISC-V does not support memory to memory data processing operations
    - Data values must be moved into registers before using them
    - The basic load and store instructions are Load and Store Word or Byte
      - lw/lb  rd,  offset(rs1)
      - sw/sb  rs2, offset(rs1)

# Memory Operation

*lw rd, rs1, imm*

*sw rs1, rs2, imm*

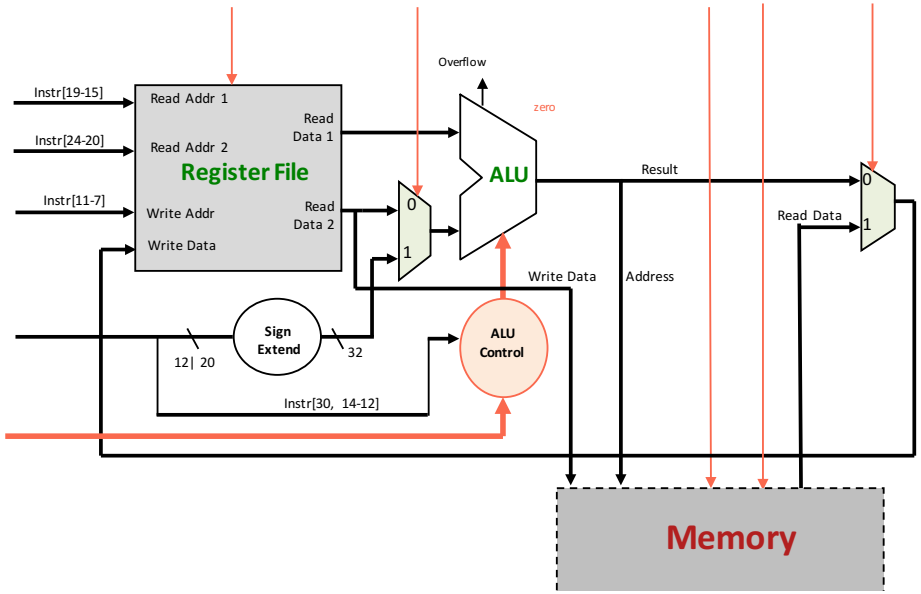



*sw rs2, offset(rs1) ; Store contents of rs2 to location pointed to by the value in rs1 plus offset*
*lw rd, offset(rs1) ; Load rd with contents of memory location pointed to by the value in rs1 plus offset*

```
fe442783   // 00000184   lw       a5,-28(s0)
```

# Register Writeback

- Write the resulting data from the instruction execution back to the register file
  - R-Type instructions
  - Load instructions
  - Some jump instruction (jal/jalr)

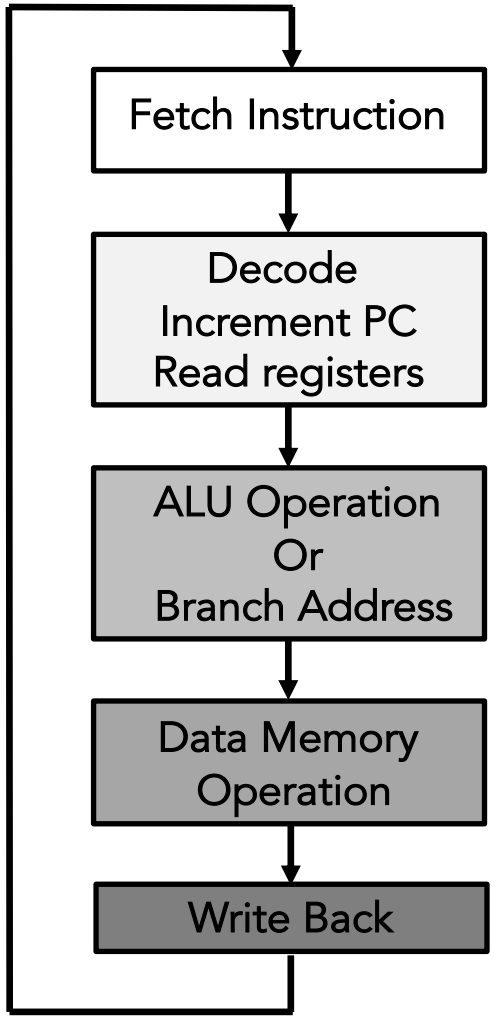


# CPU Instruction Execution Stages

| Stage | R-Type | Memory Reference | Branches | Jumps |
|---|---|---|---|---|
| Instruction Fetch | Instruction(Inst) [31:0] ← Memory[PC]<br>PC ← PC + 4 | | | |
| Instruction Decode | Read1 ← Reg. File[Inst[19:15]]<br>Read2 ← Reg. File[Inst[24:20]]<br>ALU_Result ←PC + (sign-extend(Inst[11:0]) << 1) | | | |
| Execution Operation | ALU_Result ←Read1 Op Read2 | ALU_Result ←A + sign-extend(Inst[15:0]) | If (Read1==Read2) PC ← ALU_Result | PC ← {PC[31:28], extend(Inst[20:0])} |
| Memory Access | | Load<br>Memory ← ALU_Result<br>Store<br>Memory[ALU_Result ] ← Read2 | | |
| Register Writeback | Reg. File[Inst[11:07]] ← ALU_Result | Load<br>Reg[Inst[11:07]] ← Memory[ALU_Result ] | | |

# Central Processing Unit (CPU)

- Central Processing Unit (CPU) Organization
- CPU Execution Process
  1. Fetch Instruction
  2. Decode Instruction
  3. Execute Operation
  4. Memory Operation
  5. Register Writeback Operation



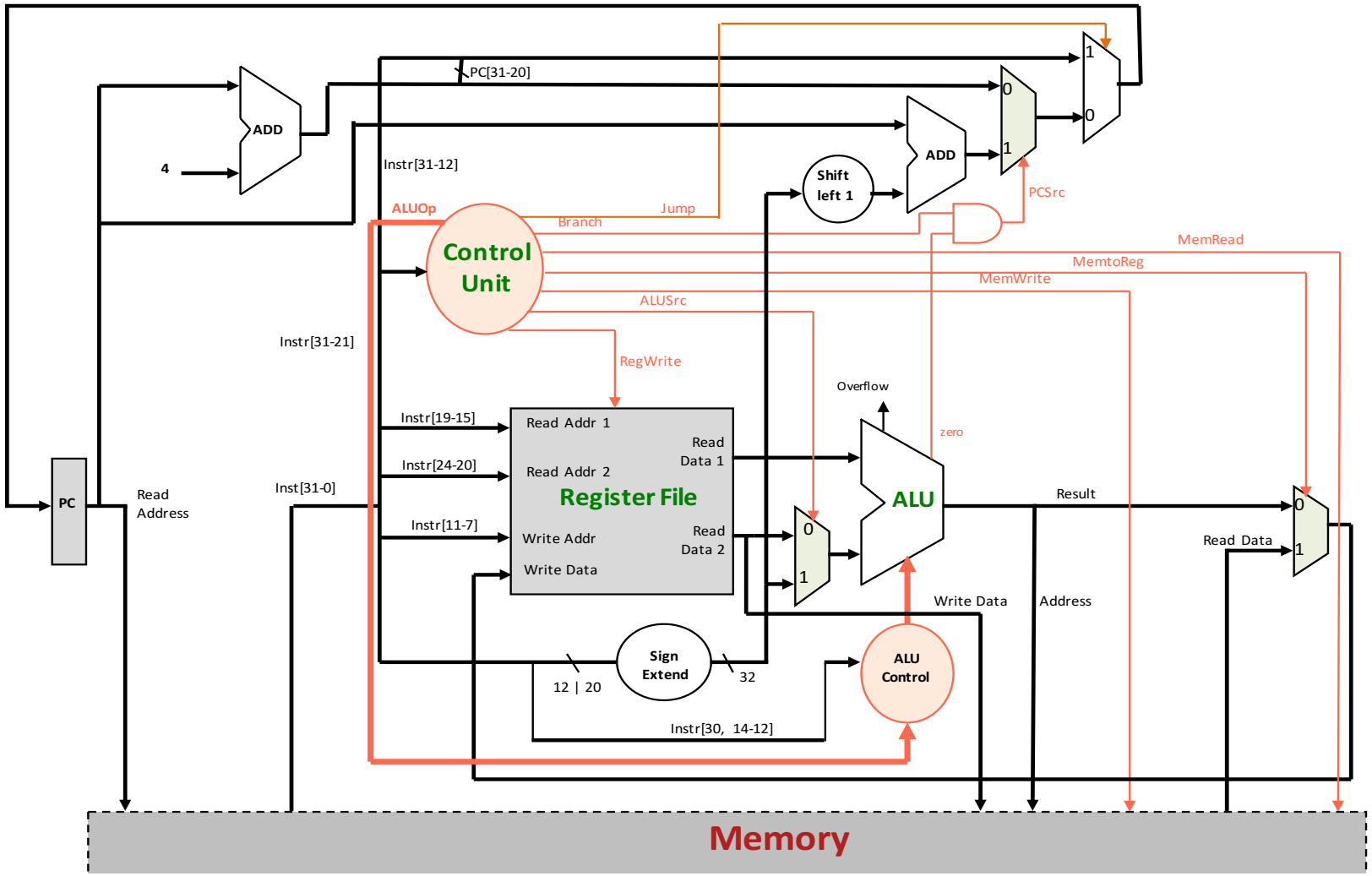
# CPU Performance

- CPU performance factors
  - Instruction count
  - Determined by ISA and compiler
  - CPI and Cycle time
  - Determined by CPU hardware
  - Longest delay determines clock period
    - Critical path: load instruction

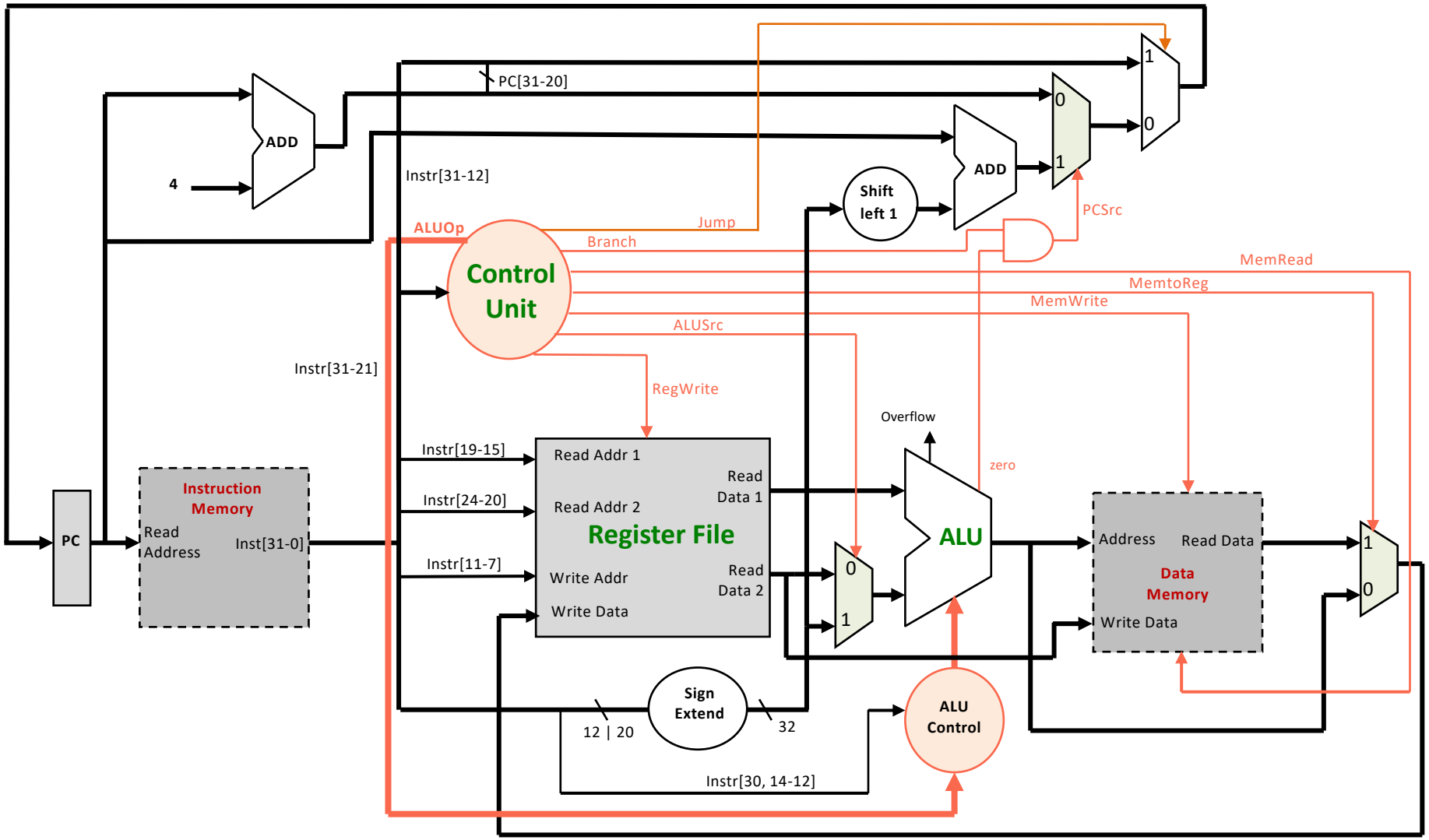
# CPU Performance

- Longest delay determines clock period
  - Critical path: load instruction
    1. Instruction memory
    2. Register file read
    3. ALU operation
    4. Data memory access
    5. Register file writeback
- Performance can be improved by pipelining

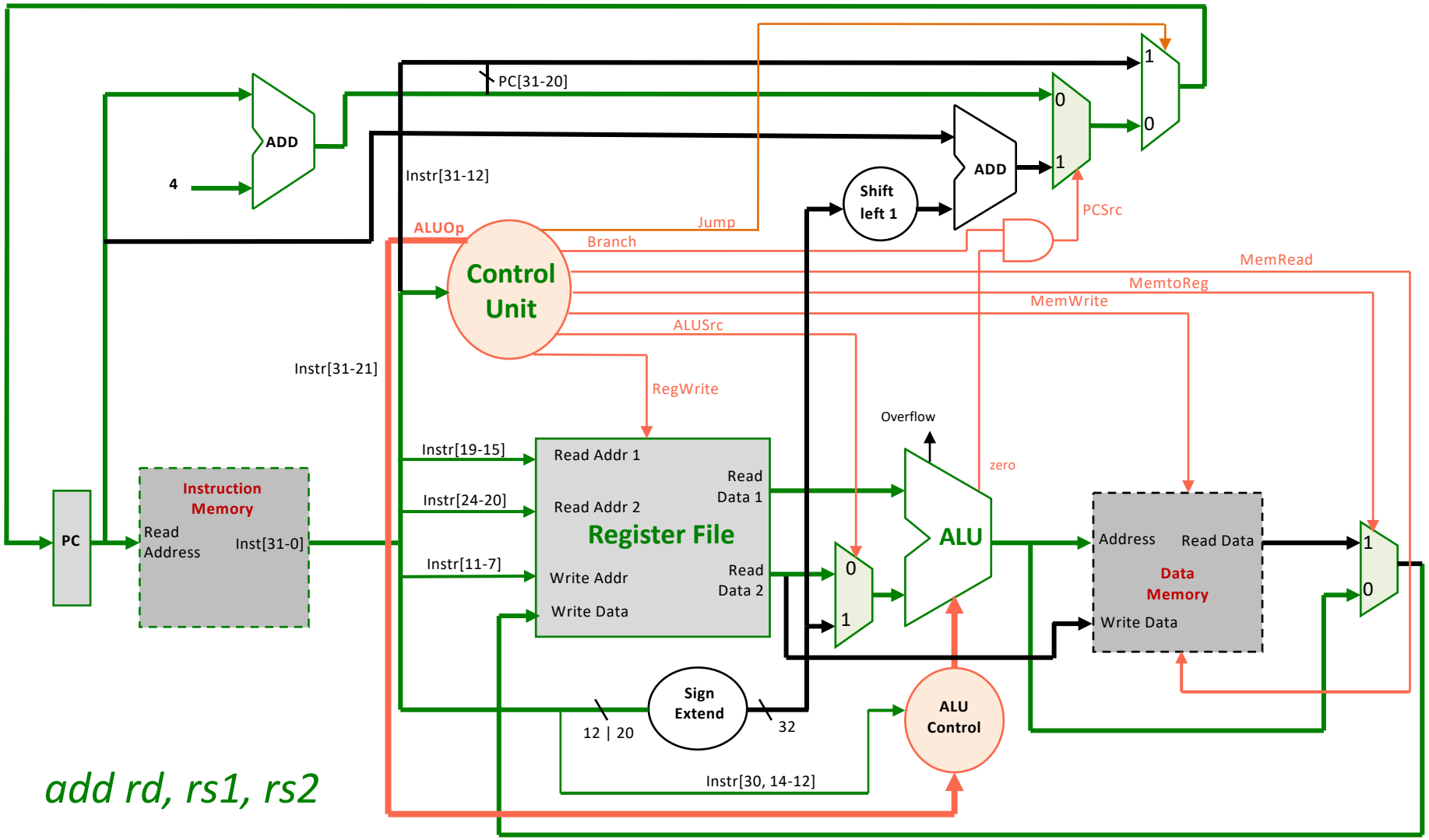
# Single Cycle RISC-V CPU



# Single Cycle RISC-V CPU

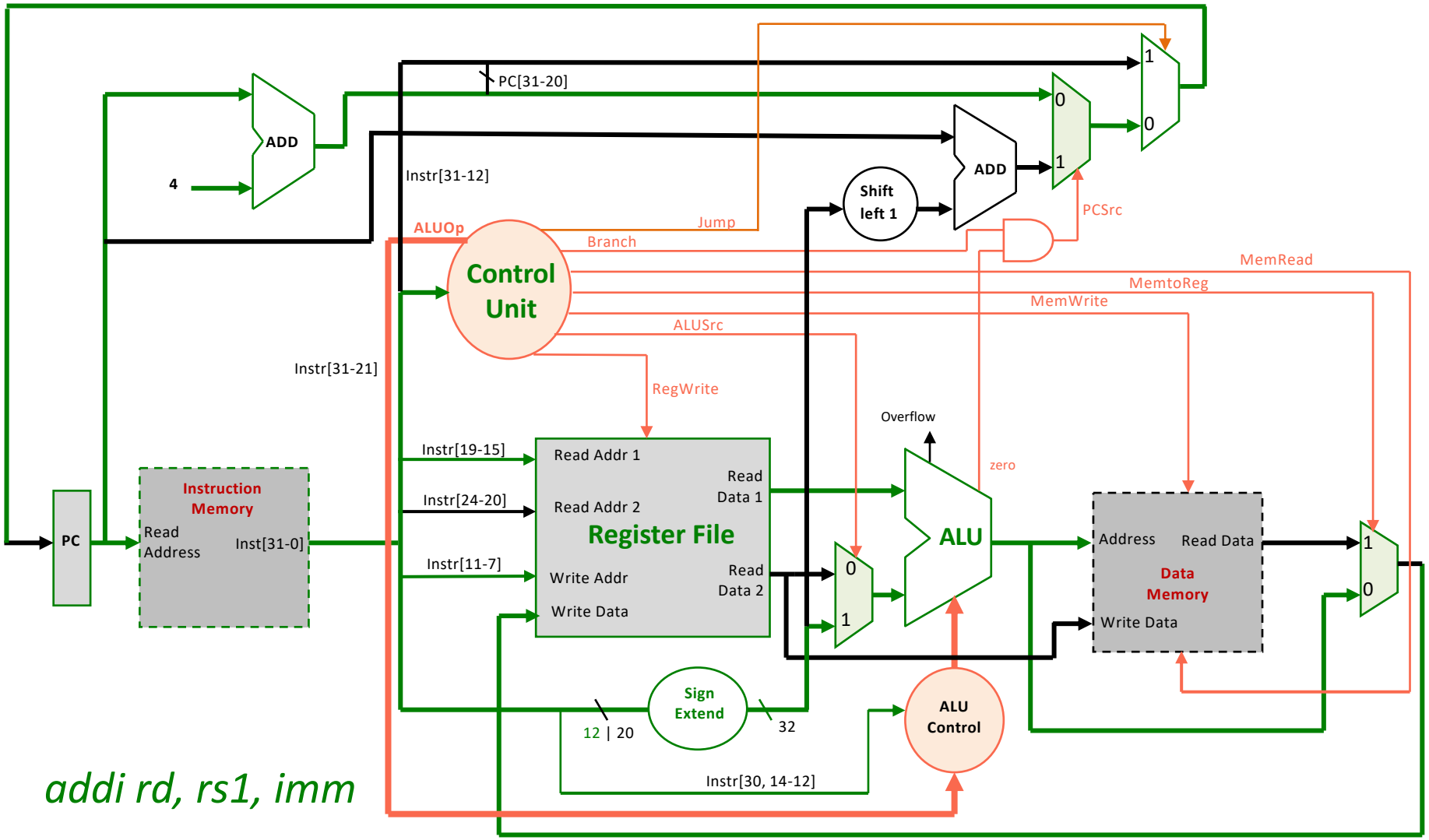


# CPU: R-Type Execution Datapath



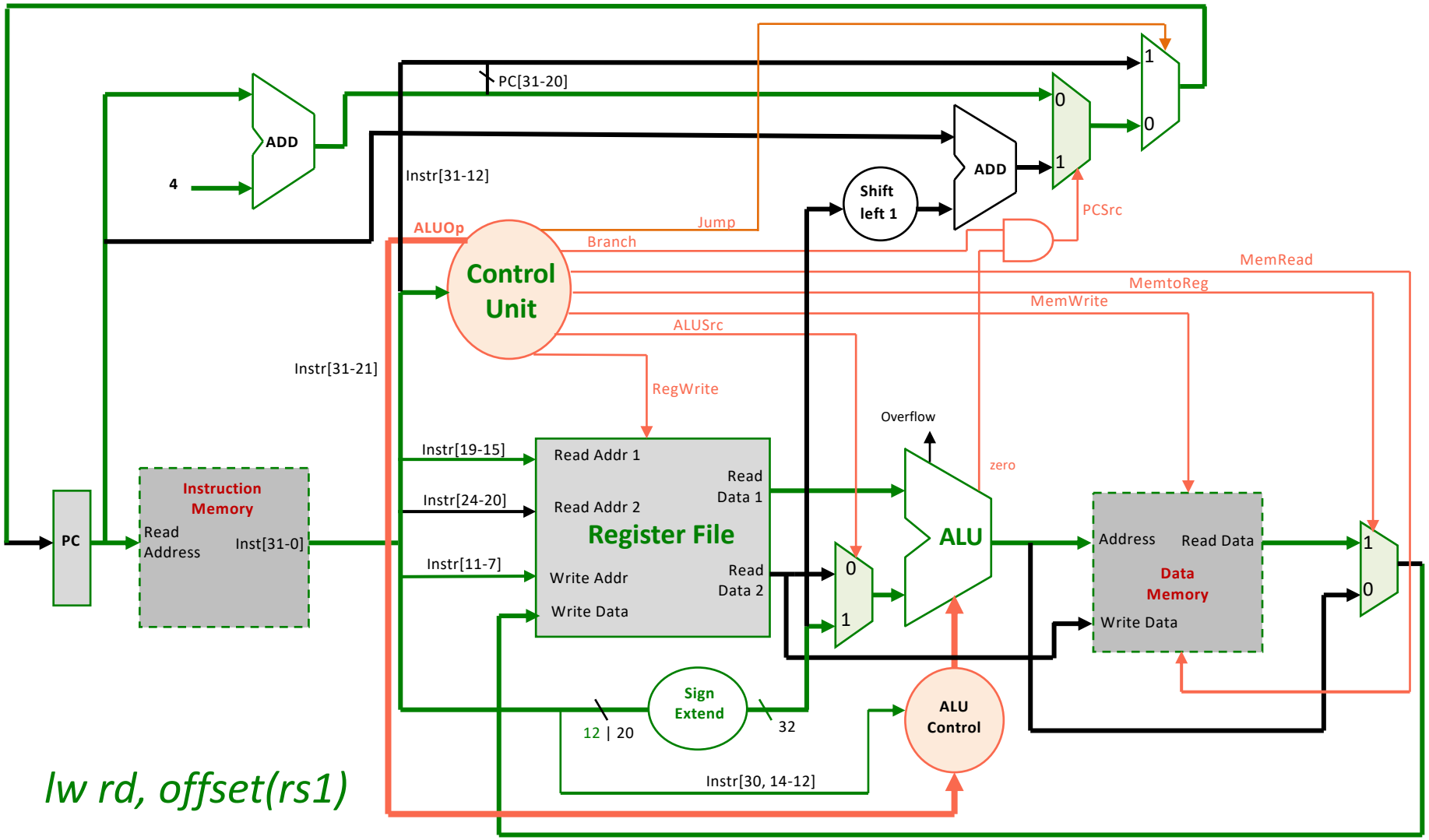

*add rd, rs1, rs2*

# CPU: I-Type Execution Datapath



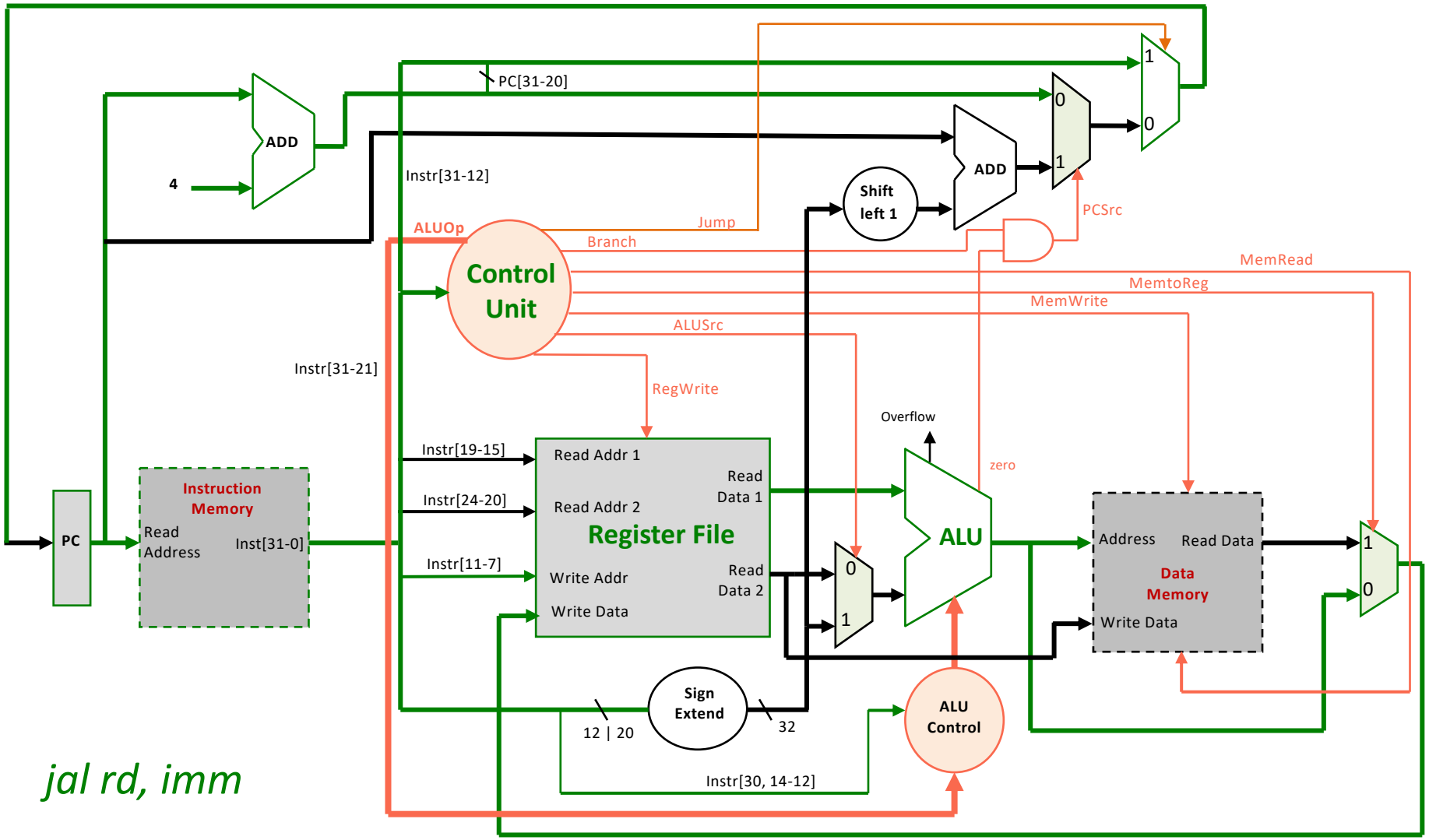

*addi rd, rs1, imm*

# CPU: I-Type Execution Datapath




*lw rd, offset(rs1)*

# CPU: U-Type Execution Datapath




*jal rd, imm*

# Instruction Execution Flows

| Instruction class | Functional Units used by the instruction class | | | | |
|---|---|---|---|---|---|
| R-type | Instruction fetch | Register access | ALU | Register access | |
| Load word | Instruction fetch | Register access | ALU | Memory access | Register access |
| Store word | Instruction fetch | Register access | ALU | Memory access | |
| Branch | Instruction fetch | Register access | ALU | | |
| Jump | Instruction fetch | | | | |

# CPU Control

| Signal name | Effect when de-asserted | Effect when asserted |
|---|---|---|
| RegDst | The destination register number comes from ***rs2*** | The destination register number comes from ***rd*** |
| RegWrite | None | Destination register is written with value on ***Write data*** |
| ALUSrc | 2nd ALU operand comes from ***Read Data 2*** | 2nd ALU operand is the sign extended, of some bits of the instruction |
| PCSrc | The ***PC*** is replaced by ***PC + 4*** | The ***PC*** is replaced by the branch target address |
| MemRead | None | Memory is read |
| MemWrite | None | Memory is written |
| MemtoReg | The value to the register ***Write data*** input comes from the ALU | The value to the register ***Write data*** input comes from the data memory |

# CPU Control

| Instruction | RegDest | ALUSrc | Memto-Reg | RegWrite | Mem Read | Mem Write | Branch |
|---|---|---|---|---|---|---|---|
| R-format | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| lw | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| sw | X | 1 | X | 0 | 0 | 1 | 0 |
| beq | X | 0 | X | 0 | 0 | 0 | 1 |

# Next Learning Module

- Processor Pipelining