중간고사 관련

- **가반**
  - 4월 28일 (수) 13:30 - 14:45
- **나반**
  - 4월 28일 (수) 15:00 – 16:15
- **다반**
  - 4월 28일 (수) 시간 선택 투표 (수요일이 가능한 학생)
  - 같은 주 다른 날 선택 투표 (수요일이 불가능한 학생)

숭실대학교
Soongsil University

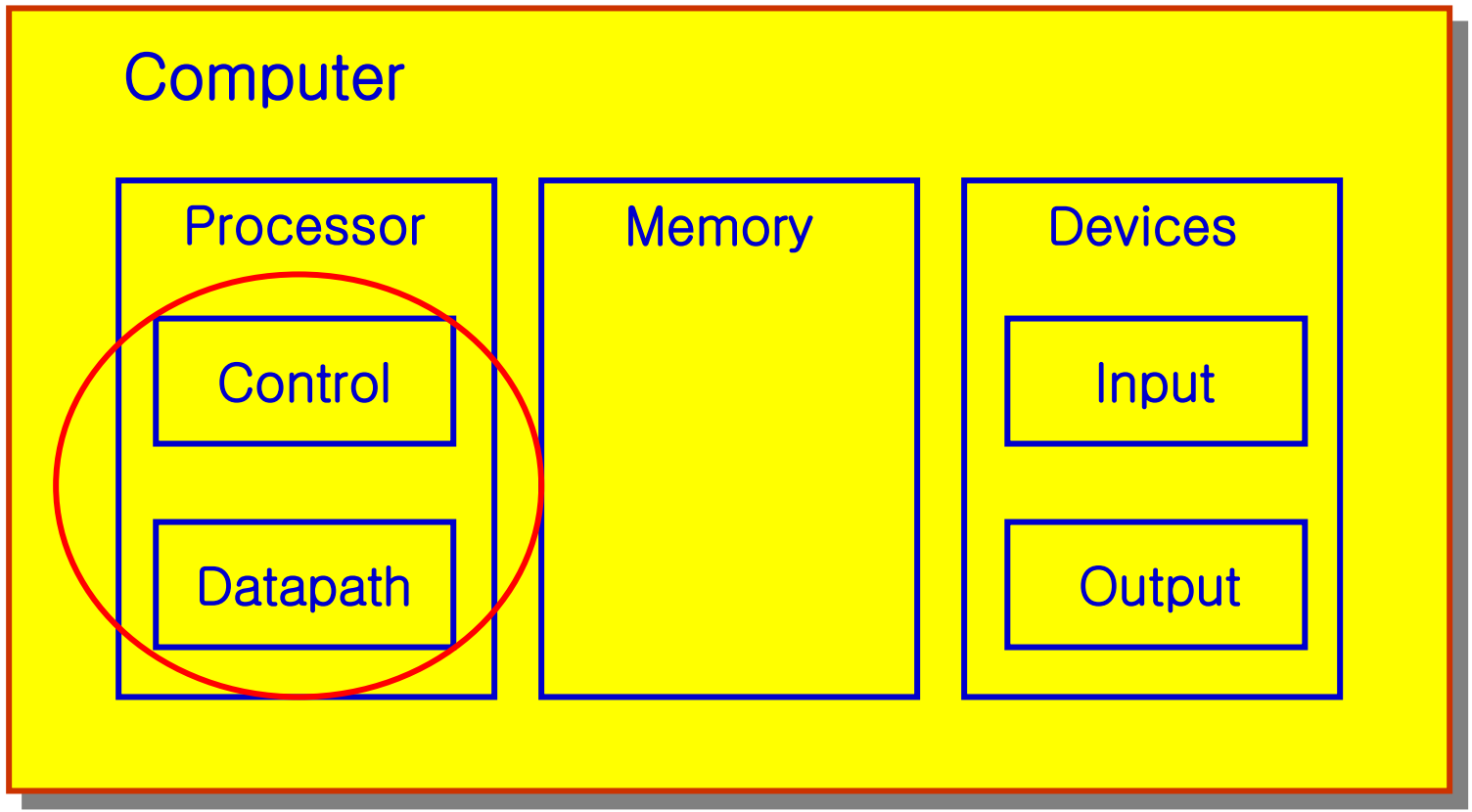
Computer Architecture

강의 #4: The Processor: Datapath and Control (1)

2021년 1학기

Young Geun Kim (김영근)

컴퓨터 구조


Computer
Processor
Control
Datapath
Memory
Devices
Input
Output

Big Picture: 프로세서의 구현

- **핵심 아이디어**
  - Datapath와 Control의 개념
  - 명령어와 데이터 bits가 어떻게 이동하는지

- **접근 방법**
  - 간단한 구현부터 시작해서,
    기능을 하나씩 추가하는 방식으로 살펴볼 예정

명령어 중 일부

- **간단한 프로세서 디자인을 살펴보기 위해, MIPS 명령어 중 일부에 집중함**
  - 메모리 접근: *lw* and *sw*
  - 산술 연산: *add*, *sub*, *and*, *ori*, *slt*
  - Branch: *beq*, *j*

MIPS 명령어 Format 다시보기

- **R-Format**
  - add rd, rs, rt
  - sub rd, rs, rt

- **Bit 구성**

Fields					
6bits	5bits	5bits	5bits	5bits	6bits
op	rs	rt	rd	shamt	funct
	1st source register	2nd source register	result register	shift amount	function code

MIPS 명령어 Format 다시보기 (Cont'd)

- **I-Format**
  - lw rt, rs, imm
  - sw rt, rs , imm
  - beq rs, rt, imm
  - ori rt, rs, imm
- **Branch 명령어 관련 주의사항**
  - Branch 명령어는 PC Relative Addressing을 사용함: PC + 4 + (4 * imm)
- **Bit 구성**

Name	Fields					
Field size	6bits	5bits	5bits	5bits	5bits	6bits
I–format	op	rs	rt	address/immediate		

1st source register

2nd source register

immediate

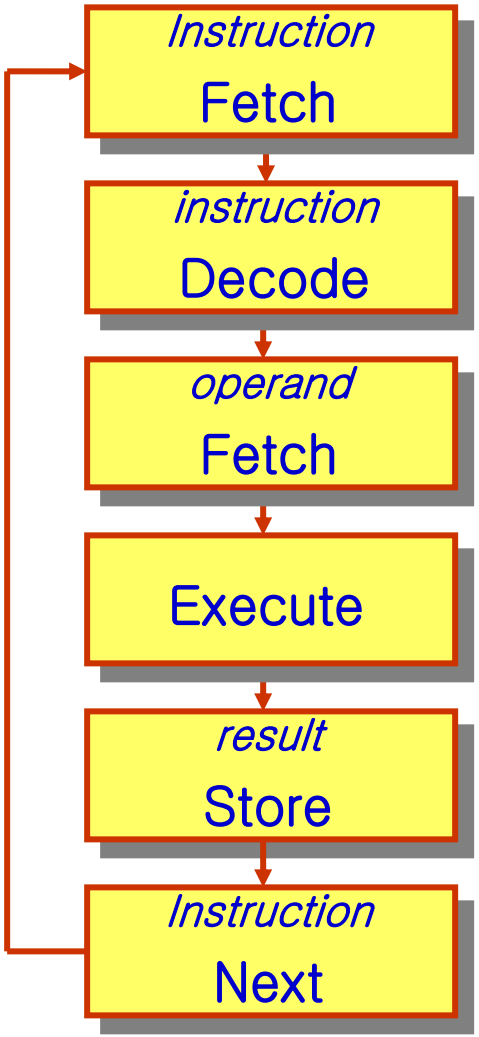
MIPS 명령어 다시보기 (Cont'd)

- **J-Format**
  - j target
- **Jump 명령어 관련 주의사항**
  - Jump 명령어는 PC Pseudo-Direct Addressing을 사용함:
    - Target * 4 를 통해 28 bits를 생성함
    - 나머지 상위 4 bits는 PC에서 가져와서 32 bits 주소를 생성함
- **Bit 구성**

Name	Fields					
Field size	6bits	5bits	5bits	5bits	5bits	6bits
J-format	op	target address				

실행 주기

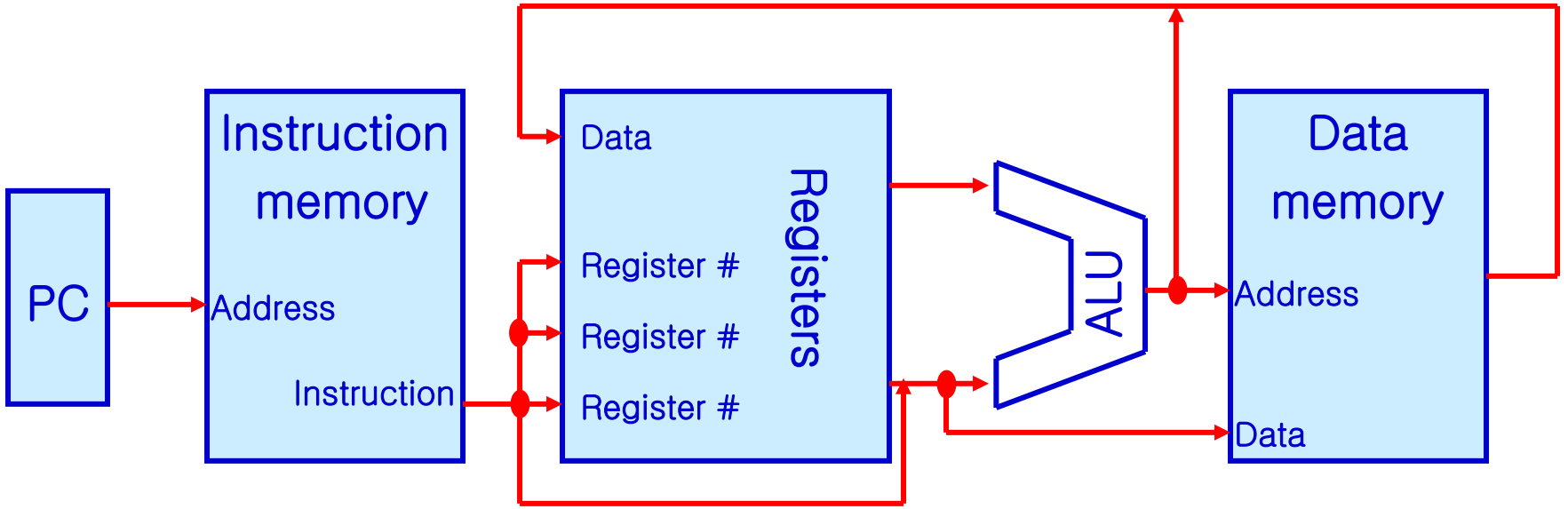
- 프로세서에서 명령어가 실행되는 순서는 다음과 같음



- *Obtain instruction from program storage*
- *Determine required actions and instruction size*
- *Locate and obtain operand data*
- *Compute result value or status*
- *Deposit results in storage for later use*
- *Determine the successor instruction*

구현 Overview

- Data는 Memory에서 Functional Unit (= 계산을 위한 Unit)으로 이동함

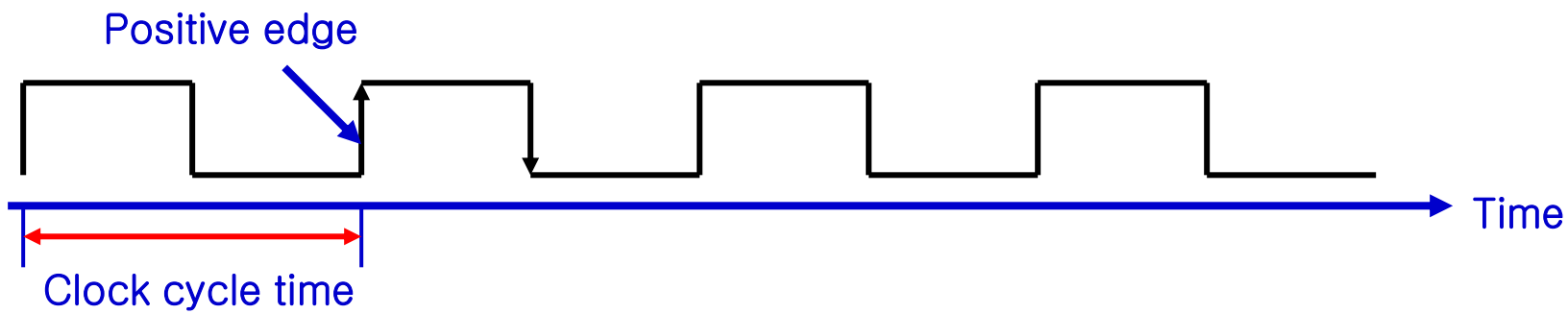


간단한 논리 회로 관련 지식

- **두 가지 중요한 정의**
  - Combinational Logic
    - 출력값이 오직 입력값에 의해서 결정됨
    - E.g., ALU
  - Sequential Logic
    - Clock에 따라 값을 저장함
    - E.g., Registers

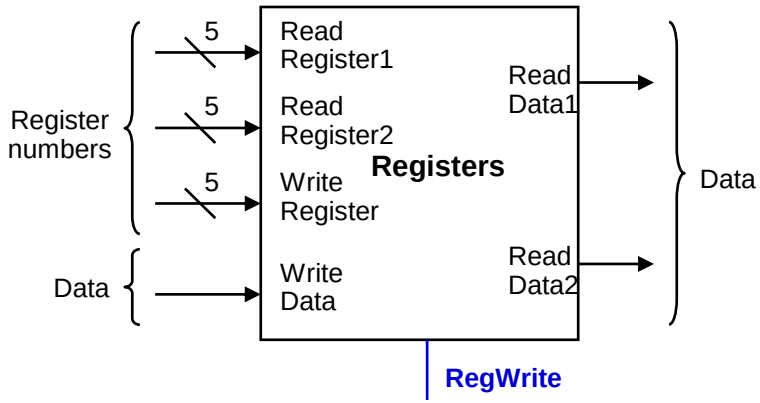
저장 장치: Register

- **Register**
  - D Flip Flop과 비슷함
    - N-bit의 입력과 출력
    - Write Enable가 추가적인 입력
  - Write Enable:
    - 0: Data Out이 변하지 않음
    - 1: Data In이 Data Out으로 나감
  - 저장된 값은 오직 Clock Edge에서만 변함



저장 장치: Register File

- **Register File은 32개의 Register로 구성:**
  - 두 개의 32-bit 출력 버스:
    Read Data 1 and Read Data 2
  - 하나의 32-bit 입력 버스:
    Write Data

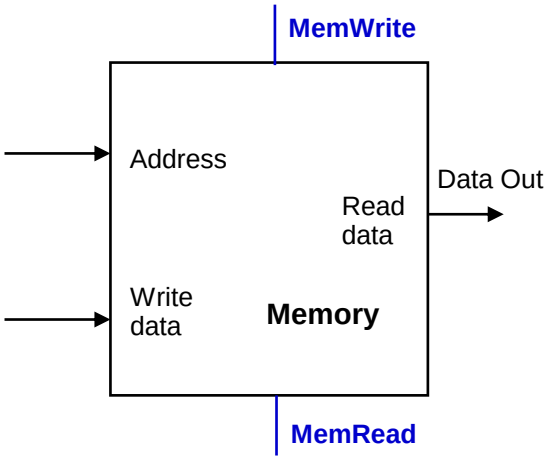


- **Register는 다음과 같이 선택됨:**
  - Read Register 1이 Read Data1로 값을 내보낼 Register를 선택
  - Read Register 2가 Read Data 2로 값을 내보낼 Register를 선택
  - Write Register는 RegWrite가 1일 때 Write Data를 쓸 Register를 선택

- **Clock Input (CLK)**
  - CLK 입력은 Write하고만 연관이 있음
  - 읽을 때는 마치 Combinational Logic처럼 동작함

저장 장치: Memory

- **Memory는 2개의 버스를 갖고 있음:**
  - 하나의 출력 버스: Read Data (Data Out)
  - 하나의 입력 버스: Write Data (Data In)



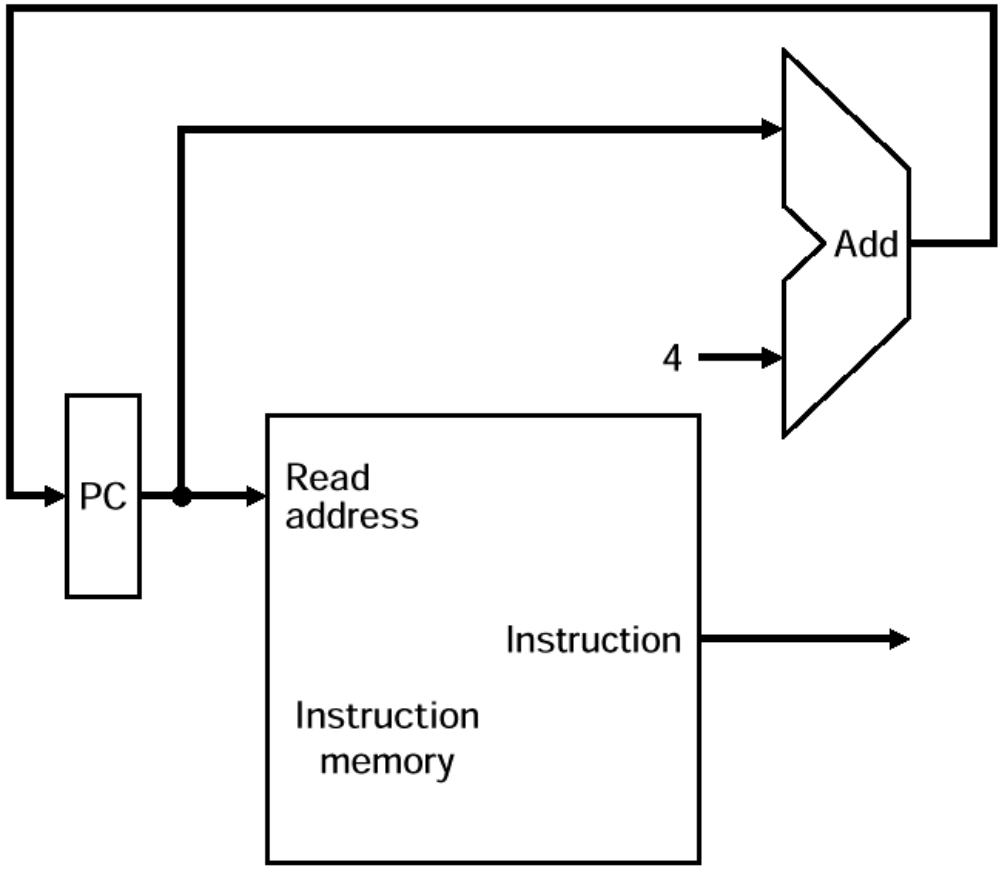
- **주소**
  - MemRead가 1일 때, Data Out으로 내보내기 위한 Data의 주소를 선택
  - MemWrite가 1일 때는 Data In으로 들어오는 Data가 저장될 곳의 주소를 선택

- **Clock Input (CLK)**
  - CLK 입력은 Write하고만 연관이 있음
  - 읽을 때는 마치 Combinational Logic처럼 동작함

Instruction Fetch (IF)

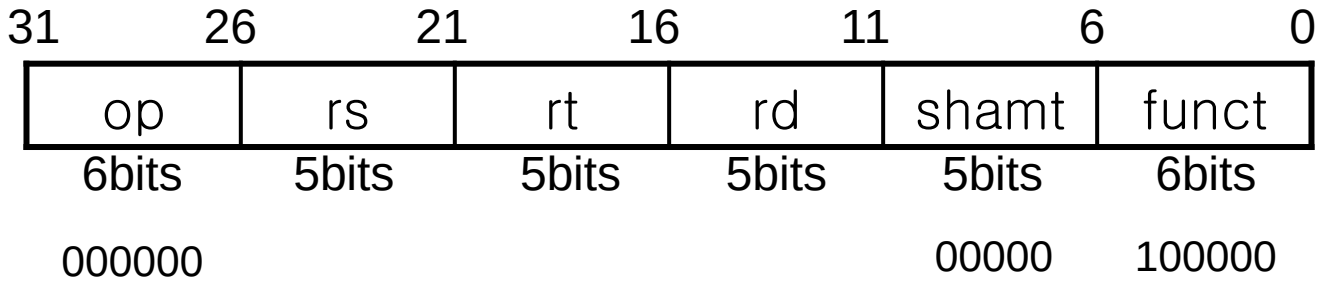
- **일반적인 동작**
  - Fetch 명령어
    - Mem[PC]; // PC (= 이번에 실행할 명령어의 주소를 저장한 Register) 에 저장된 주소에 접근하여 명령어를 불러옴
  - Update PC
    - PC = PC + 4; // 다음 명령어를 실행하기 위해 PC를 PC+4로 Update

Datapath: IF Unit



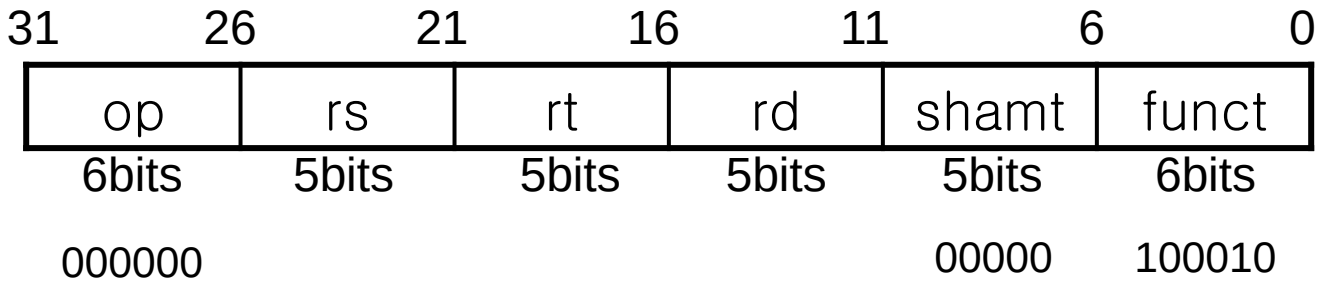
산술 연산: Add

- **Add 명령어**
  - add rd, rs, rt
    - Mem[PC]; // PC (= 이번에 실행할 명령어의 주소를 저장한 Register) 에 저장된 주소에 접근하여 명령어를 불러옴
    - R[rd] = R[rs] + R[rt] // Register rs와 Register rt에서 불러온 값을 더하여 Register rd에 저장함
    - PC = PC + 4 // 다음 명령어를 실행하기 위해 PC를 PC+4로 Update



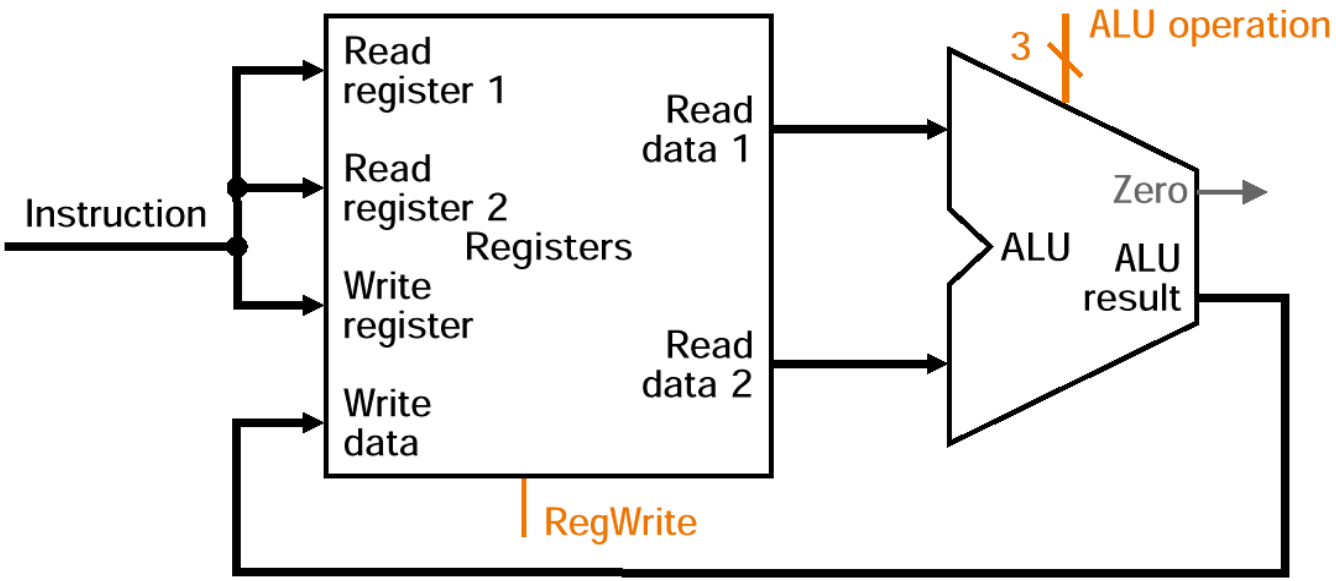
산술 연산: Sub

- **Sub 명령어**
  - sub rd, rs, rt
    - Mem[PC]; // PC (= 이번에 실행할 명령어의 주소를 저장한 Register) 에 저장된 주소에 접근하여 명령어를 불러옴
    - R[rd] = R[rs] - R[rt] // Register rs와 Register rt에서 불러온 값을 빼서 Register rd에 저장함
    - PC = PC + 4 // 다음 명령어를 실행하기 위해 PC를 PC+4로 Update



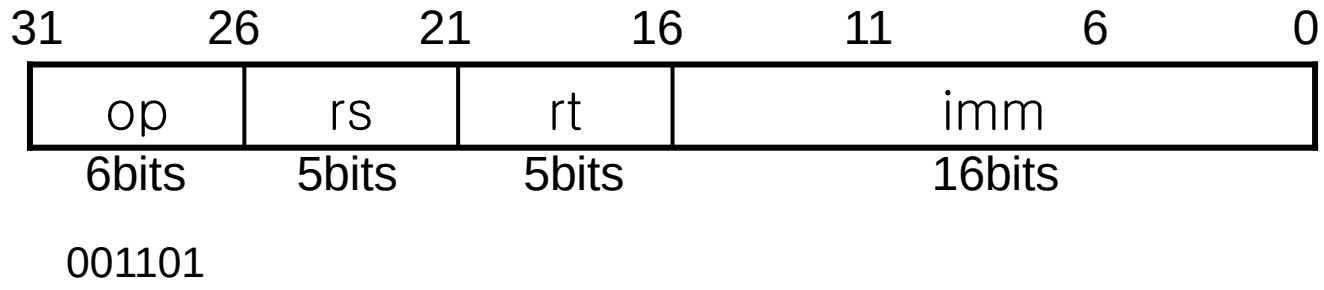
Datapath: Reg/Reg Operations

- **R[rd] = R[rs] op R[rt];**
  - ALU (Arithmetic/Logical Unit) Control과 RegWrite는 명령어의 opcode와 funct에 따라 결정됨
  - Read Register1, Read Register2, 그리고 Write Register는 rs, rt, rd Field에 의해서 결정됨



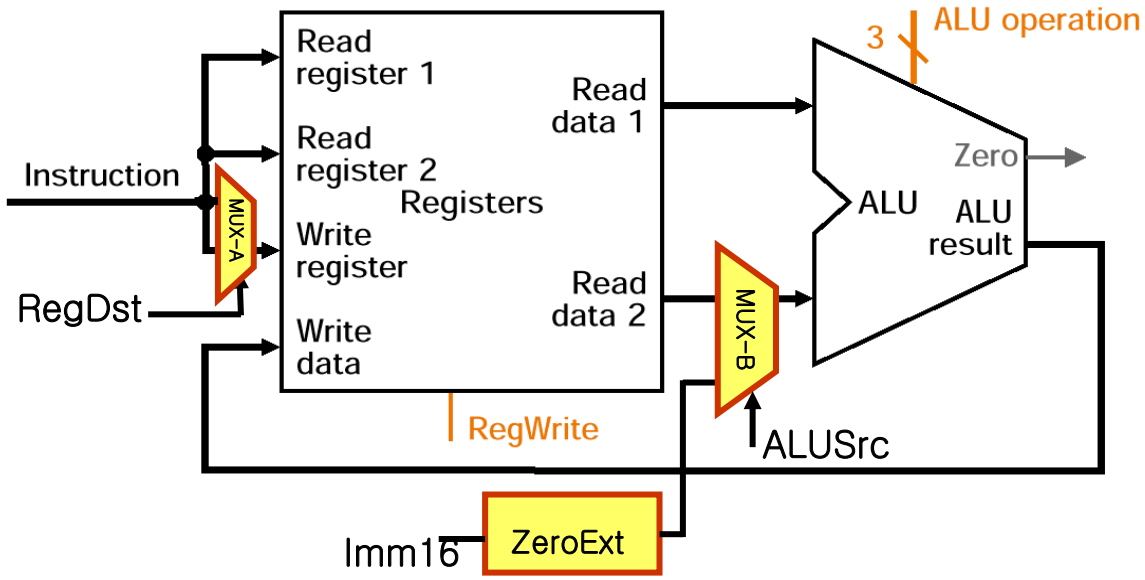
논리 연산: OR Immediate

- **OR Immediate 명령어**
  - ori rt, rs, imm
    - Mem[PC]; // PC (= 이번에 실행할 명령어의 주소를 저장한 Register) 에 저장된 주소에 접근하여 명령어를 불러옴
    - R[rt] = R[rs] OR ZeroExt(imm) // Register rs에서 불러온 값과 Zero-Extended imm 값의 OR 결과를 Register rt에 저장함
    - PC = PC + 4 // 다음 명령어를 실행하기 위해 PC를 PC+4로 Update



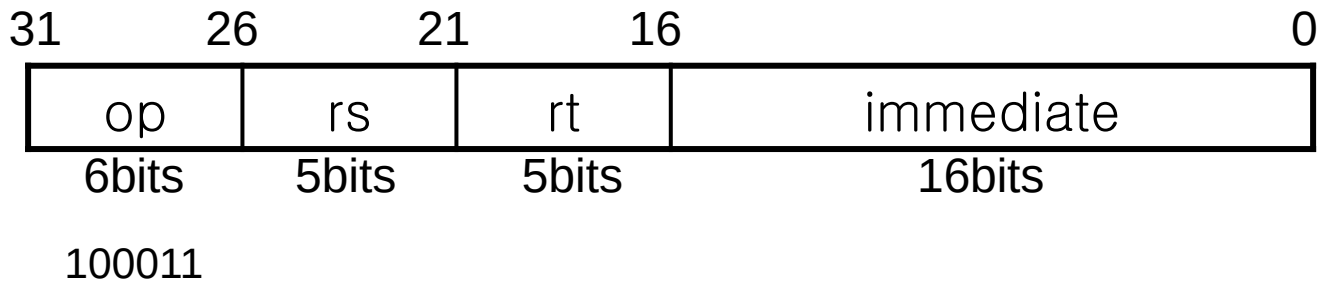
Datapath: Immediate Operations

- **2 Muxes and 1 ZeroExt 가 추가되어야 함**
  - 첫 번째 Mux: RegDst에 따라, Write Register를 rd 또는 rt 중 하나로 선택
  - 두 번째 Mux: ALUSrc에 따라, ALU의 두번째 입력값을 Read Data2 또는 ZeroExt(Imm) 중 하나로 선택



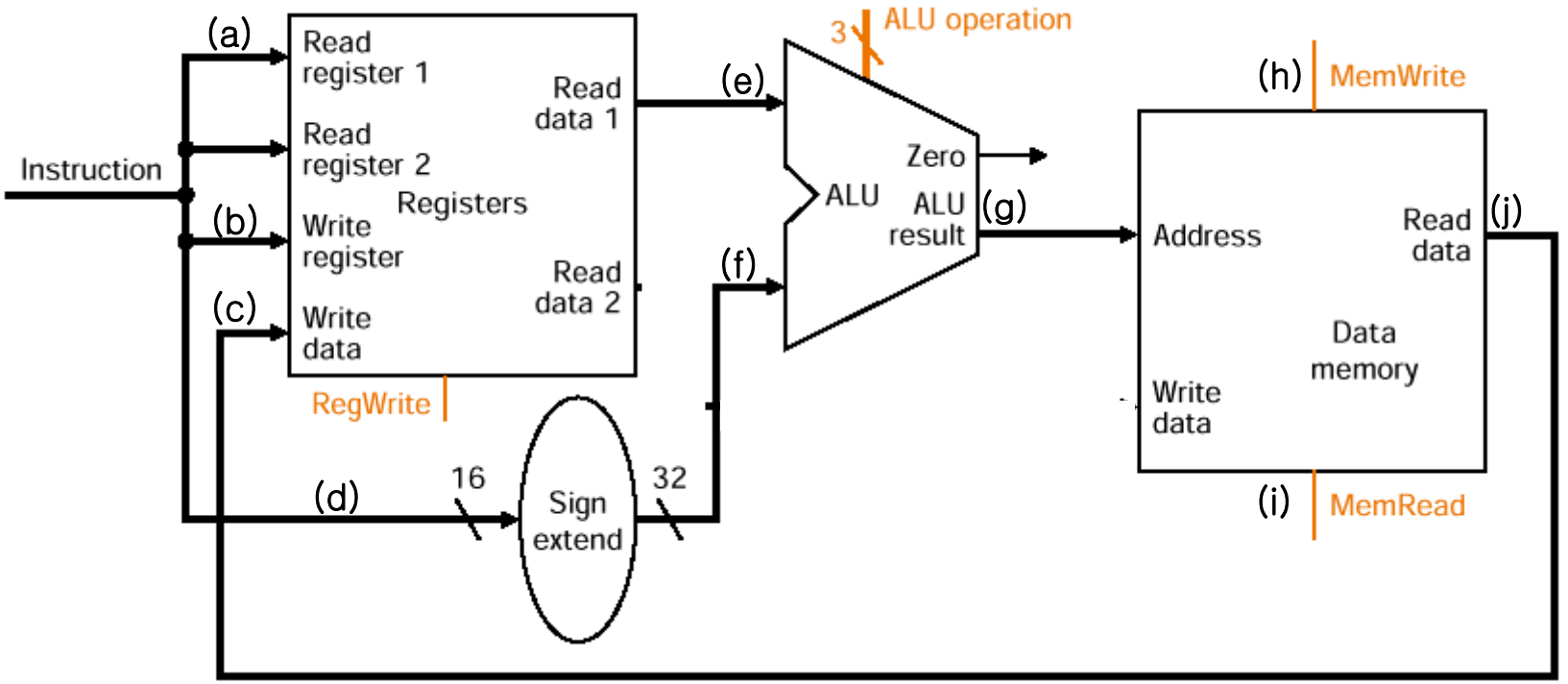
메모리 접근: Load

- **Load 명령어**
  - lw rt, rs, imm
    - Mem[PC]; // PC (= 이번에 실행할 명령어의 주소를 저장한 Register) 에 저장된 주소에 접근하여 명령어를 불러옴
    - Addr = R[rs] + SignExt(imm) // Register rs에서 불러온 값과 Sign-Extended imm 값을 더해 접근할 주소를 계산함
    - R[rt] = Mem[Addr]; // 메모리의 Addr 주소에 접근하여 불러온 Data를 Register rt에 저장함
    - PC = PC + 4 // 다음 명령어를 실행하기 위해 PC를 PC+4로 Update



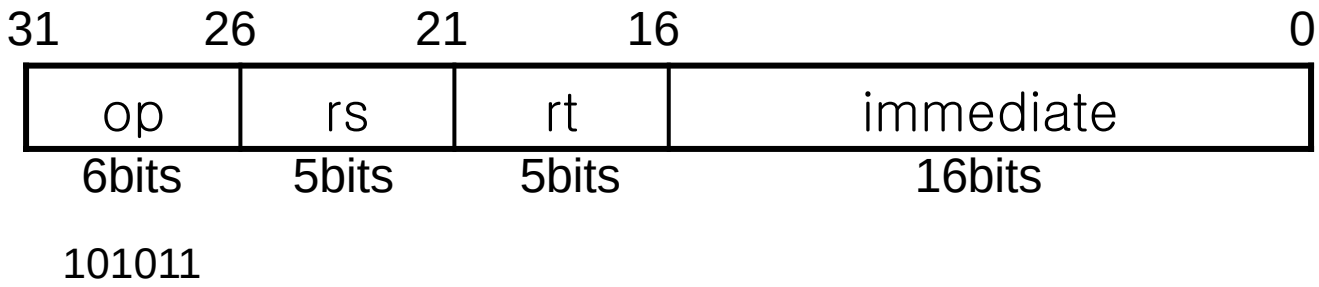
Datapath: Load

- **Sign Extension이 추가됨**
  - Offset이 양수나 음수 모두 가능하기 때문임
  - E.g., lw $r1, 100($r2) / lw $r1, -100($r2)



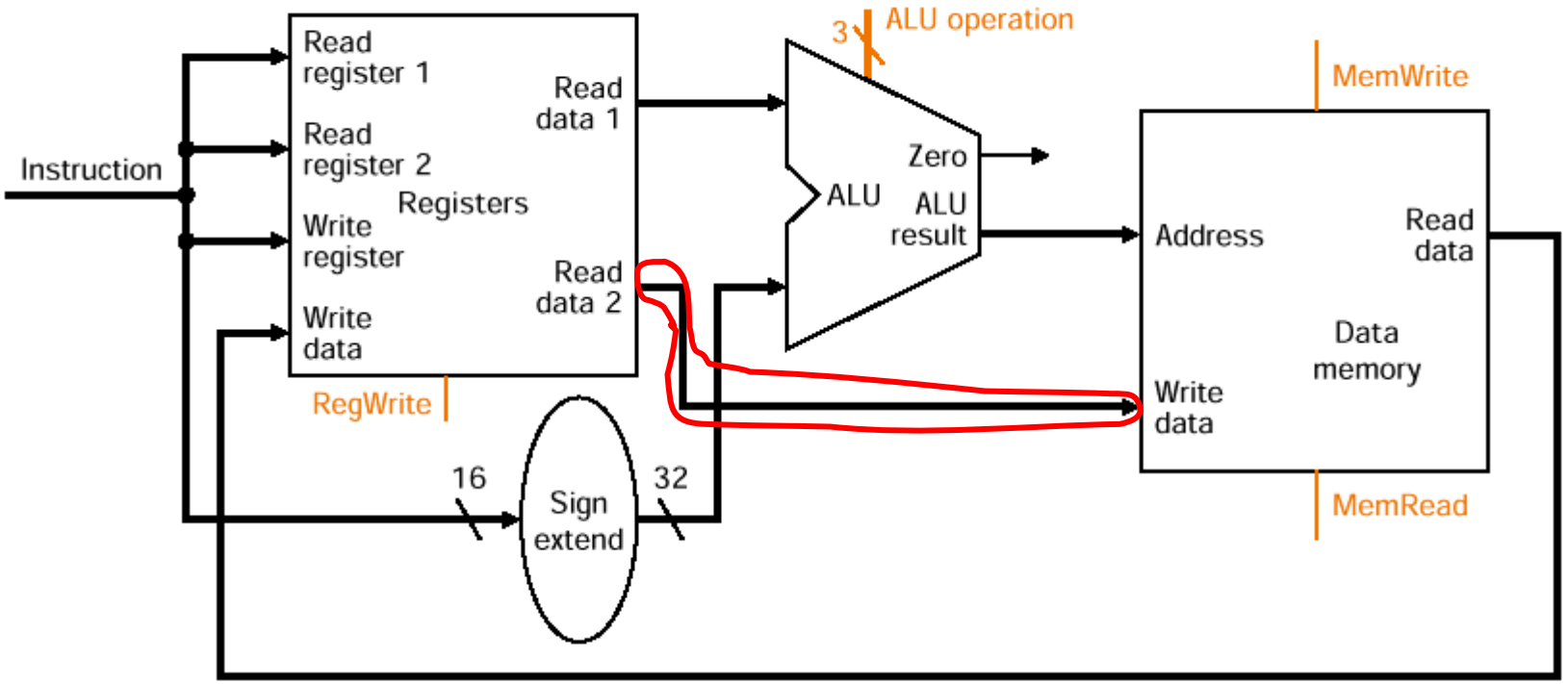
메모리 접근: Store

- **Store 명령어**
  - lw rt, rs, imm
    - Mem[PC]; // PC (= 이번에 실행할 명령어의 주소를 저장한 Register) 에 저장된 주소에 접근하여 명령어를 불러옴
    - Addr = R[rs] + SignExt(imm) // Register rs에서 불러온 값과 Sign-Extended imm 값을 더해 접근할 주소를 계산함
    - Mem[Addr] = R[rt]; // 메모리의 Addr 주소에 Register rt에서 불러온 값을 저장함
    - PC = PC + 4 // 다음 명령어를 실행하기 위해 PC를 PC+4로 Update



Datapath: Store

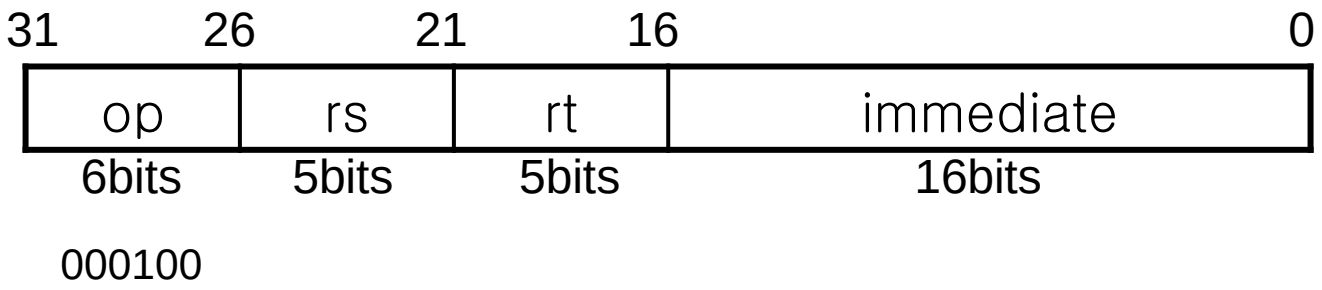
- Register에서 Memory의 Write Data로 가는 Path가 추가됨



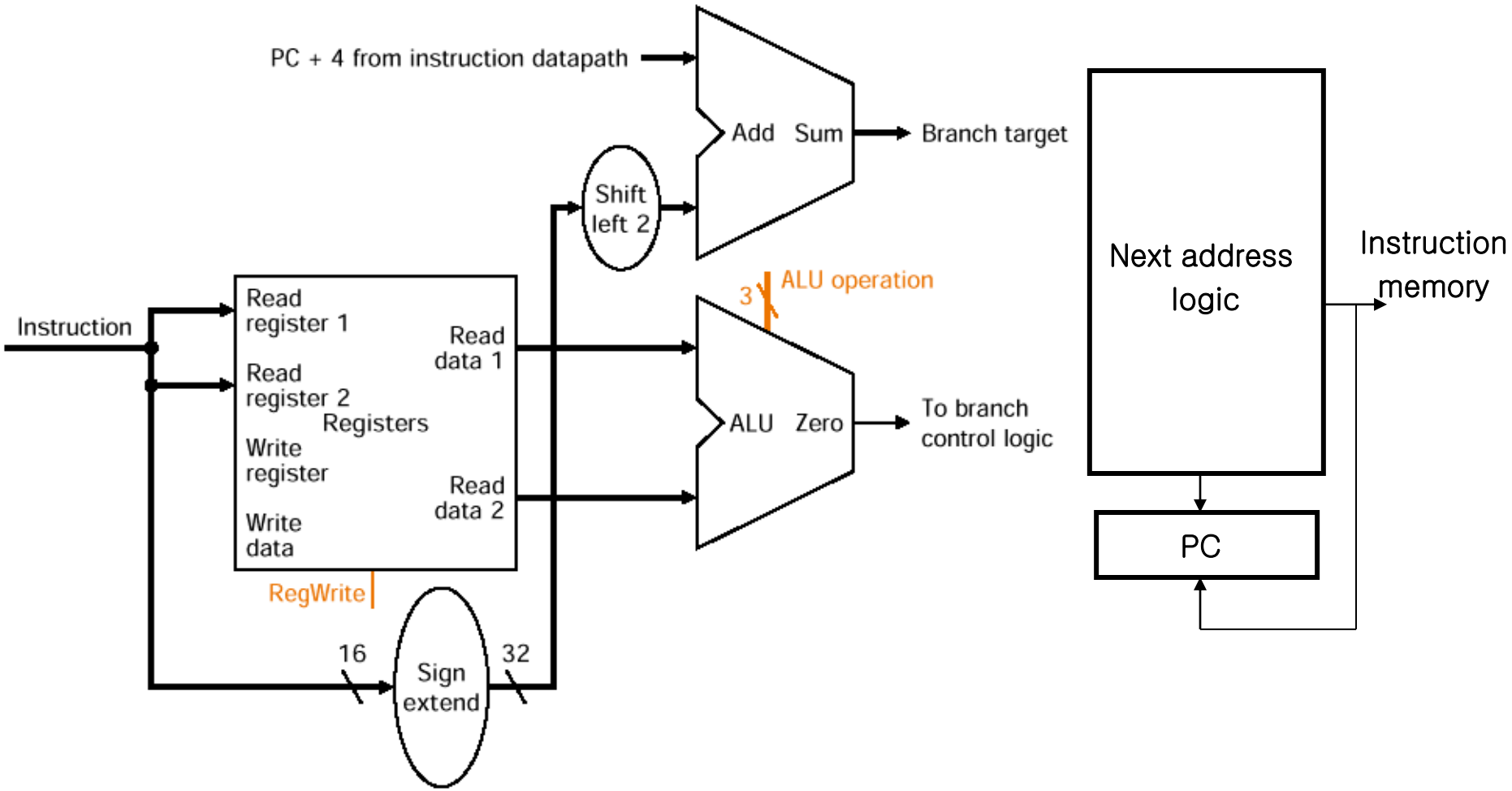
Branch

- **Branch 명령어**
  - beq rt, rs, imm
    - Mem[PC]; // PC (= 이번에 실행할 명령어의 주소를 저장한 Register) 에 저장된 주소에 접근하여 명령어를 불러옴
    - Zero = R[rs] – R[rt] // R[rt]와 R[rs]의 값이 같으면 0을 Zero에 저장함
    - If (Zero == 0) then
      PC = PC + 4 + (SignExt(imm) << 2); // Cond가 0이면, PC + 4 + (imm * 4) 값을 PC에 저장함

      else
      PC = PC + 4 // Cond가 0이 아니면, 다음 명령어가 실행될 수 있도록 PC + 4를 PC에 저장함



Datapath: Branch



다음에 실행될 주소

- **PC는 명령어 메모리에 접근하기 위한 Byte단위 주소를 저장함**
  - 일반적인 경우
    - PC[31:0] = PC[31:0] + 4
  - Branch 하는 경우
    - PC[31:0] = PC[31:0] + 4 + SignExt(imm) * 4

Datapath: 빠르지만 비싼 회로

- PC는 일반적인 경우를 위해 "미리" 계산됨
- Branch 명령어일 때, Zero 값을 확인하기 위한 ALU연산과 동시에
  - SignExt(imm) * 4를 PC + 4에 더해줌