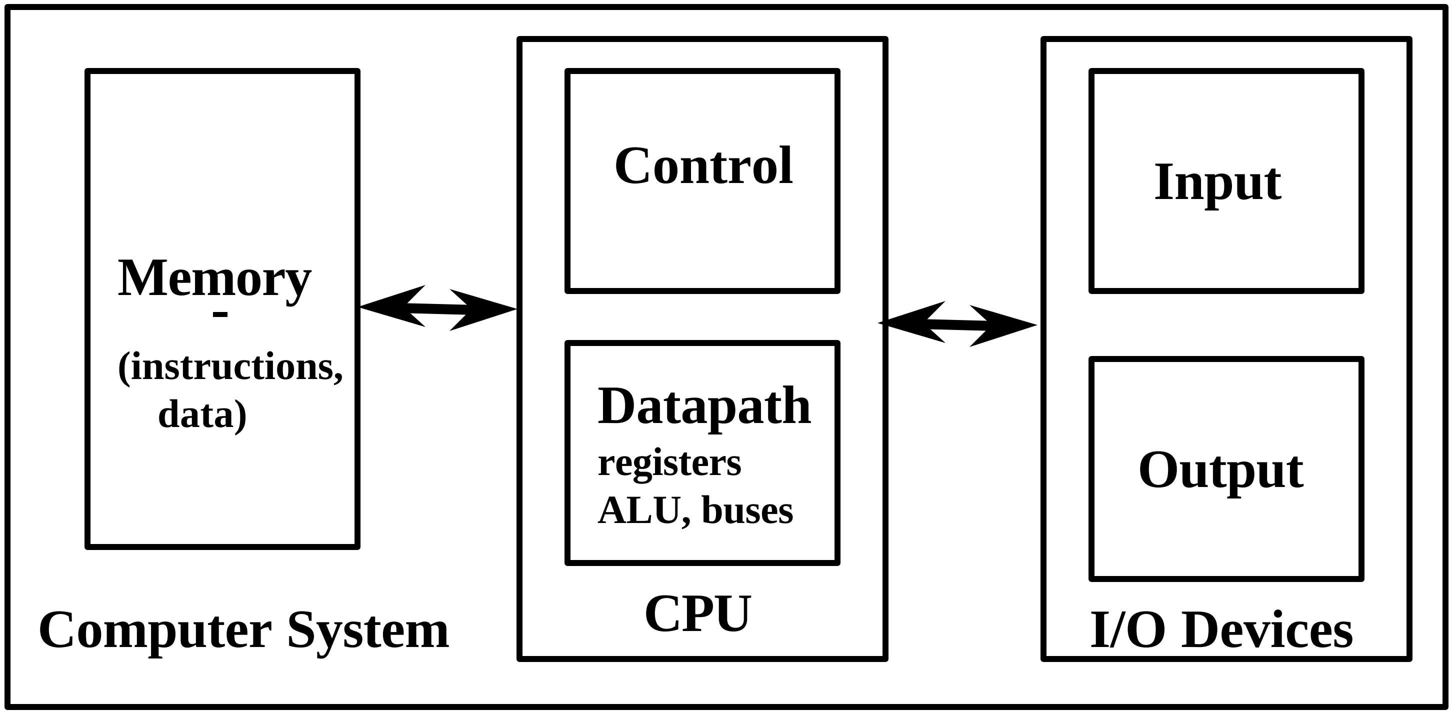
The Von-Neumann Computer Model

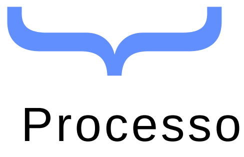
- **Partitioning of the computing engine into components:**
  - **Central Processing Unit (CPU): Control Unit (instruction decode, sequencing of operations), Datapath (registers, arithmetic and logic unit, buses).**
  - **Memory: Instruction and operand storage.**
  - **Input/Output (I/O).**
  - **The stored program concept: Instructions from an instruction set are fetched from a common memory and executed one at a time.**

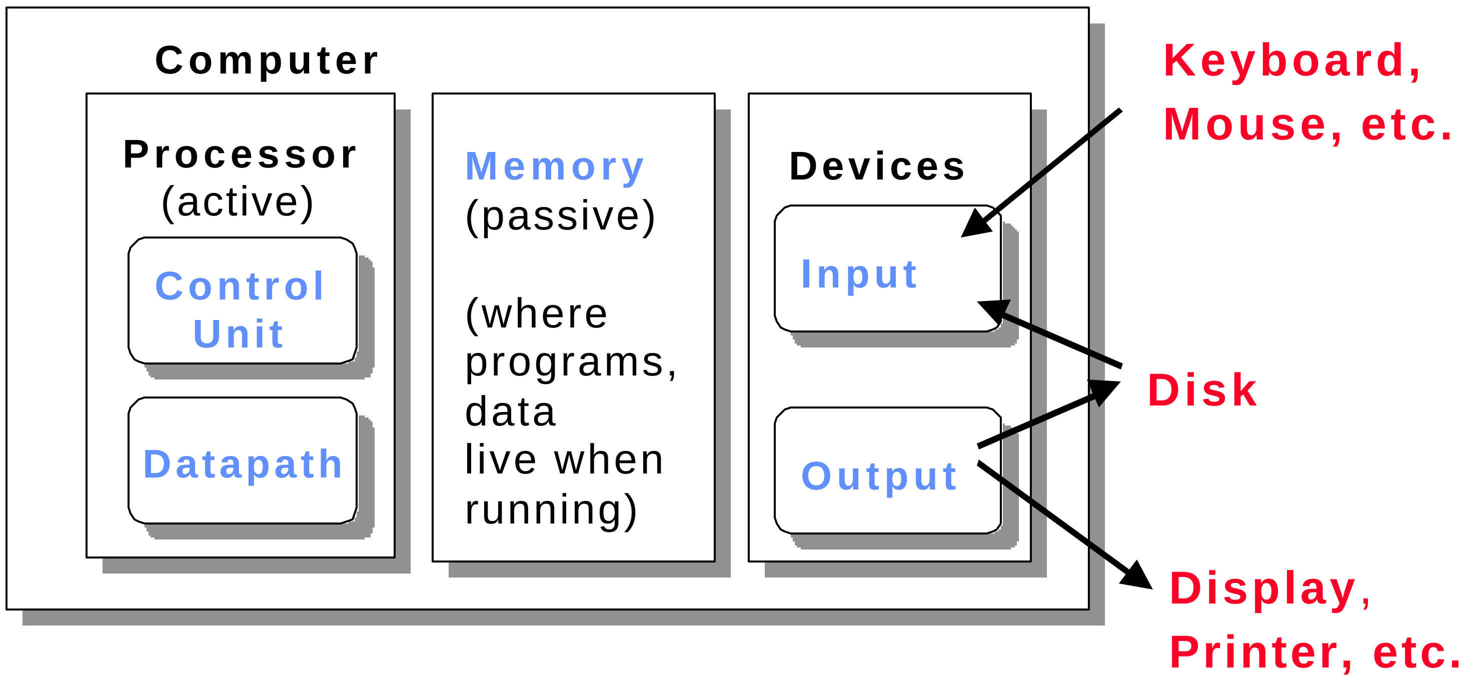


Hardware Components of Any Computer

Five classic components of all computers:

1. Control Unit; 2. Datapath; 3. Memory; 4. Input; 5. Output





CPU Organization

- **Datapath Design:**
  - **Capabilities & performance characteristics of principal Functional Units (FUs):**
  - **(e.g., Registers, ALU, Shifters, Logic Units, ...)**
  - **Ways in which these components are interconnected (buses connections, multiplexors, etc.).**
  - **How information flows between components.**
- **Control Unit Design:**
  - **Logic and means by which such information flow is controlled.**
  - **Control and coordination of FUs operation to realize the targeted Instruction Set Architecture to be implemented (can either be implemented using a finite state machine or a microprogram).**
- **Hardware description with a suitable language, possibly using Register Transfer Notation (RTN).**

Hardware Description

- Hardware visualization:
  - Block diagrams (spatial visualization):

    Two-dimensional representations of functional units and their interconnections.
  - Timing charts (temporal visualization):

    Waveforms where events are displayed vs. time.
- Register Transfer Notation (RTN):
  - A way to describe microoperations capable of being performed by the data flow (data registers, data buses, functional units) at the register transfer level of design (RT).
  - Also describes conditional information in the system which cause operations to come about.
  - A "shorthand" notation for microoperations.
- Hardware Description Languages:
  - Examples: VHDL: VHSIC (Very High Speed Integrated Circuits) Hardware Description Language, Verilog.

Register Transfer Notation (RTN)

- **Dependent RTN:** When RTN is used after the data flow is assumed to be frozen. No data transfer can take place over a path that does not exist. No statement implies a function the data flow hardware is incapable of performing.
- **Independent RTN:** Describe actions on registers without regard to nonexistence of direct paths or intermediate registers. No predefined data flow.
- The general format of an RTN statement:

  Conditional information: Action1; Action2

- The conditional statement is often an AND of literals (status and control signals) in the system (a p-term). The p-term is said to imply the action.
- Possible actions include transfer of data to/from registers/memory data shifting, functional unit operations etc.

RTN Statement Examples

$A \leftarrow B$

- **A copy of the data in entity B (typically a register) is placed in Register A**
- **If the destination register has fewer bits than the source, the destination accepts only the lowest-order bits.**
- **If the destination has more bits than the source, the value of the source is sign extended to the left.**

CTL • T0: A = B

- **The contents of B are presented to the input of combinational circuit A**
- **This action to the right of “:” takes place when control signal CTL is active and signal T0 is active.**

RTN Statement Examples

MD $\leftarrow$ M[MA]

- Memory locations are indicated by square brackets.
- Means the memory data register receives the contents of the main memory (M) as addressed from the Memory Address (MA) register.

AC(0), AC(1), AC(2),AC(3)

- Register fields are indicated by parenthesis.
- The concatenation operation is indicated by a comma.
- Bit AC(0) is bit 0 of the accumulator AC
- The above expression means AC bits 0, 1, 2, 3
- More commonly represented by AC(0-3)

E • T3: CLRWRITE

- The control signal CLRWRITE is activated when the condition E • T3 is active.

CPU Design Steps

1. Analyze instruction set operations using independent RTN => datapath <u>requirements.</u>

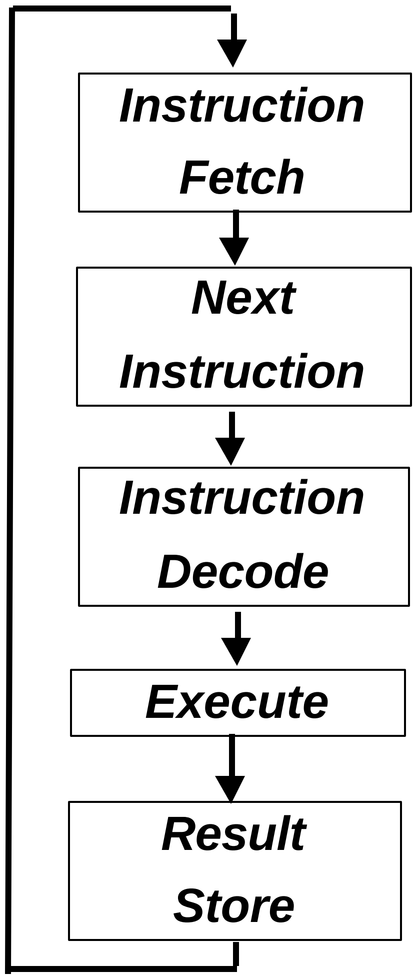
2. Select set of datapath components & establish clock methodology.

3. <u>Assemble</u> datapath meeting the requirements.

4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.

5. Assemble the control logic.

Instruction Processing Steps



Obtain instruction from program storage

Update program counter to address of next instruction

Determine instruction type
Obtain operands from registers

} Common steps for all instructions

Compute result value or status

Store result in register/memory if needed (usually called Write Back).

A Subset of MIPS Instructions

ADD and SUB:

addU rd, rs, rt

subU rd, rs, rt

31	26	21	16	11	6	0
op	**rs**	**rt**	**rd**	**shamt**	**funct**	
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

OR Immediate:

ori rt, rs, imm16

31	26	21	16	0
op	**rs**	**rt**	**immediate**	
6 bits	5 bits	5 bits	16 bits	

LOAD and STORE Word

lw rt, rs, imm16

sw rt, rs, imm16

31	26	21	16	0
op	**rs**	**rt**	**immediate**	
6 bits	5 bits	5 bits	16 bits	

BRANCH:

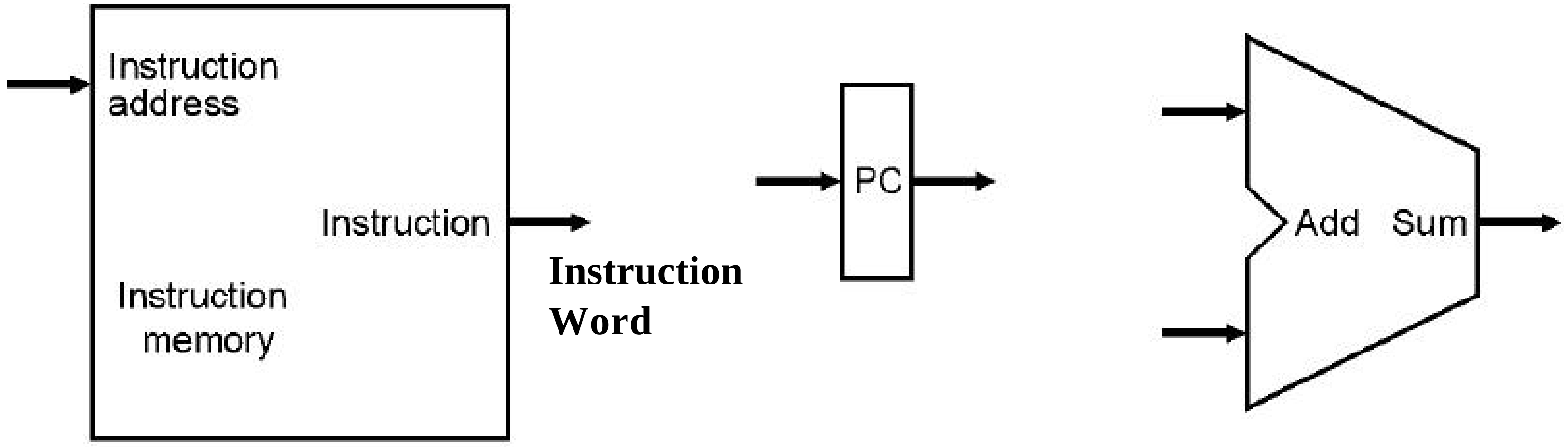
beq rs, rt, imm16

31	26	21	16	0
op	**rs**	**rt**	**immediate**	
6 bits	5 bits	5 bits	16 bits	

Overview of MIPS Instruction Micro-operations

- **All instructions go through these two steps:**
  - **Send program counter to instruction memory and fetch the instruction. (*fetch)***
  - **Read one or two registers, using instruction fields. (*decode*)**
    - **Load reads one register only.**
- **Additional instruction execution actions (*execution*) depend on the instruction in question, but similarities exist:**
  - **All instruction classes use the ALU after reading the registers:**
    - **Memory reference instructions use it for address calculation.**
    - **Arithmetic and logic instructions (R-Type), use it for the specified operation.**
    - **Branches use it for comparison.**
- **Additional execution steps where instruction classes differ:**
  - **Memory reference instructions: Access memory for a load or store.**
  - **Arithmetic and logic instructions: Write ALU result back in register.**
  - **Branch instructions: Change next instruction address based on comparison.**

Datapath Components



a. Instruction memory b. Program counter c. Adder

Two state elements needed to store and access instructions:

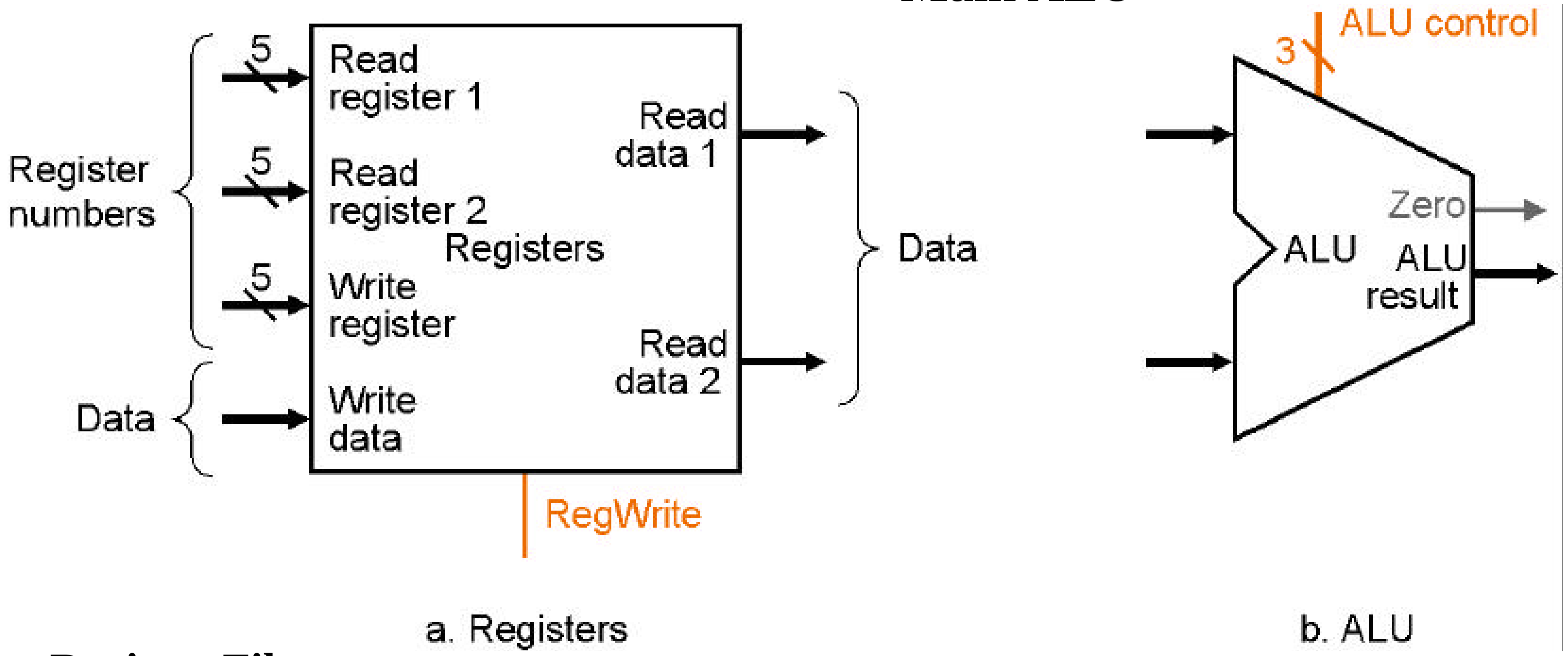
1 **Instruction memory:**
- **Only read access.**
- **No read control signal.**

2 **Program counter: 32-bit register.**
- **Written at end of every clock cycle: No write control signal.**
- **32-bit Adder: To compute the the next instruction address.**

More Datapath Components

Register File

Main ALU



Register File:

- **Contains all registers.**
- **Two read ports and one write port.**
- **Register writes by asserting write control signal**
- **Writes are edge-triggered.**
- **Can read and write to the same register in the same clock cycle.**

R-Type Example: Micro-Operation Sequence For ADDU

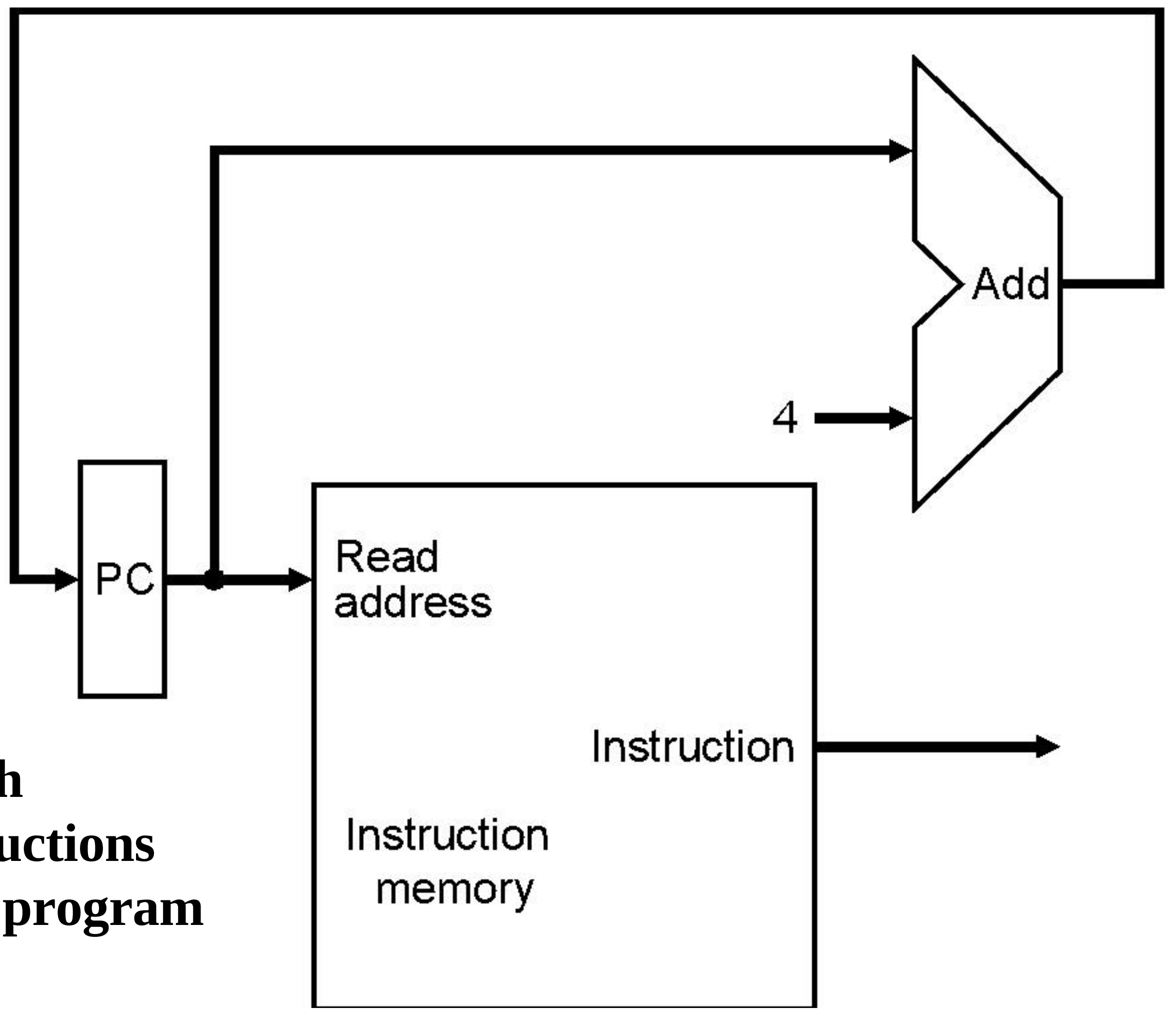
addU rd, rs, rt

OP	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Instruction Word $\leftarrow$ Mem[PC]	Fetch the instruction
PC $\leftarrow$ PC + 4	Increment PC
R[rd] $\leftarrow$ R[rs] + R[rt]	Add register rs to register rt result in register rd

Building The Datapath

Instruction Fetch & PC Update:



Portion of the datapath used for fetching instructions and incrementing the program counter.

Logical Operations with Immediate Example:

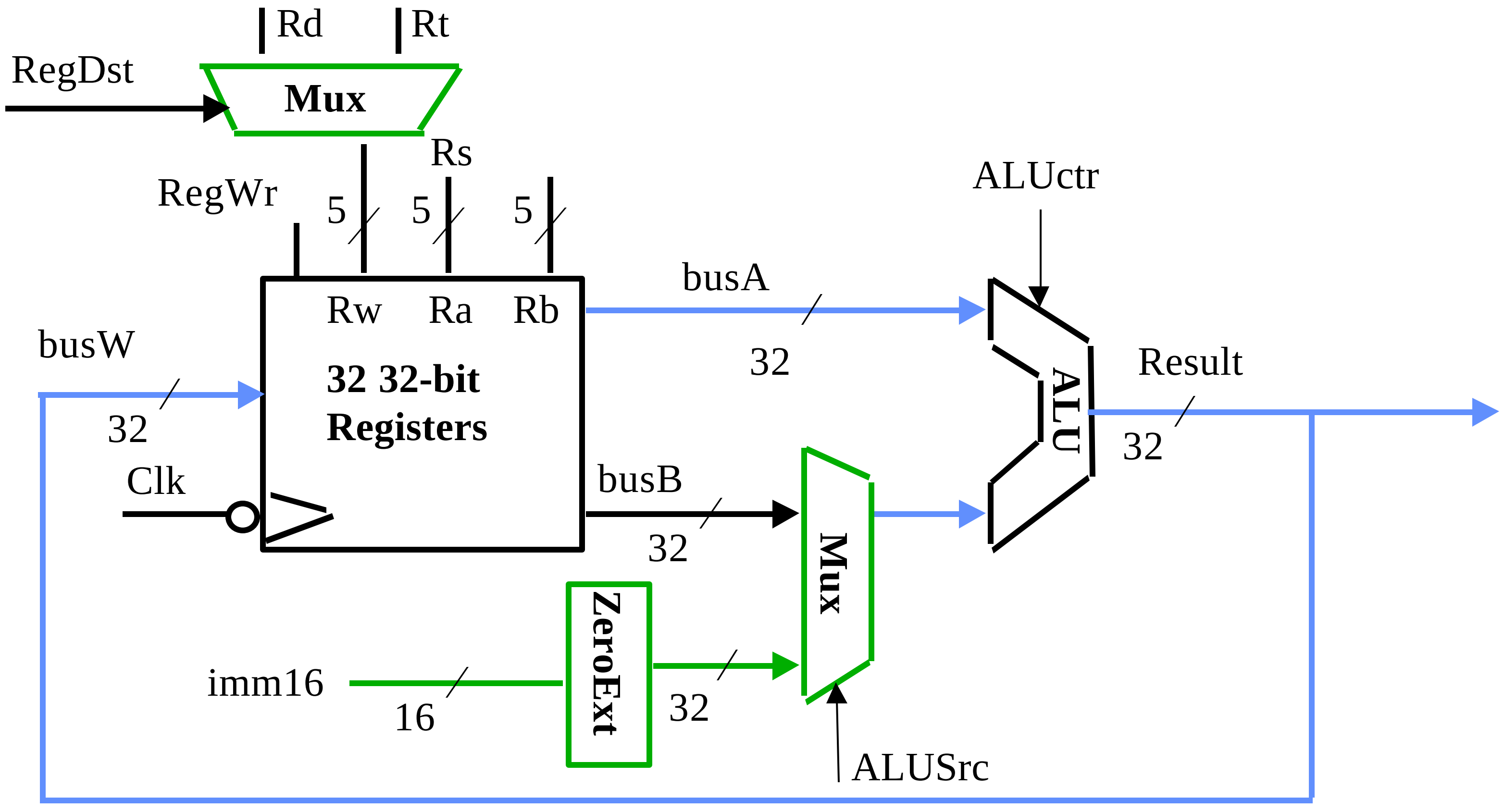
Micro-Operation Sequence For ORI

ori rt, rs, imm16

31 26	21	16	0
op	**rs**	**rt**	**immediate**
6 bits	5 bits	5 bits	16 bits

Instruction Word ← Mem[PC]	**Fetch the instruction**
PC ← PC + 4	**Increment PC**
R[rt] ← R[rs] OR ZeroExt[imm16]	**OR register rs with immediate field zero extended to 32 bits, result in register rt**

Datapath For Logical Instructions With Immediate



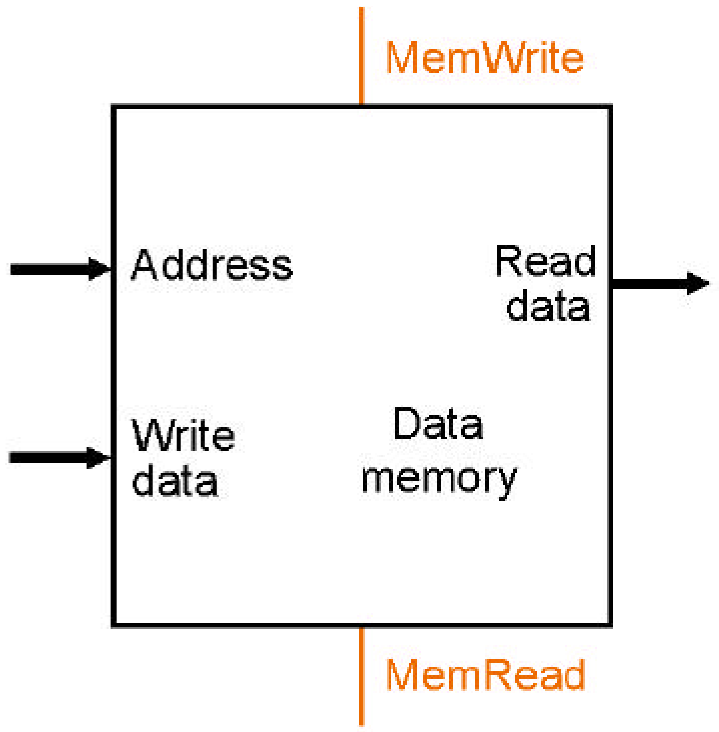
Load Operations Example:
Micro-Operation Sequence For LW

lw rt, rs, imm16

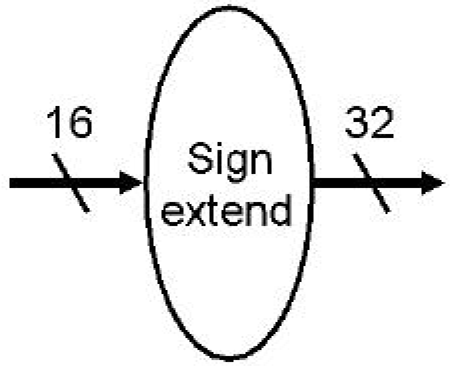
31	26	21	16	0
op	**rs**	**rt**	**immediate**	
6 bits	5 bits	5 bits	16 bits	

Instruction Word ← Mem[PC]	**Fetch the instruction**
PC ← PC + 4	**Increment PC**
R[rt] ← Mem[R[rs] + SignExt[imm16]]	**Immediate field sign extended to 32 bits and added to register rs to form memory load address, word at load address to register rt**

Additional Datapath Components For Loads & Stores



a. Data memory unit

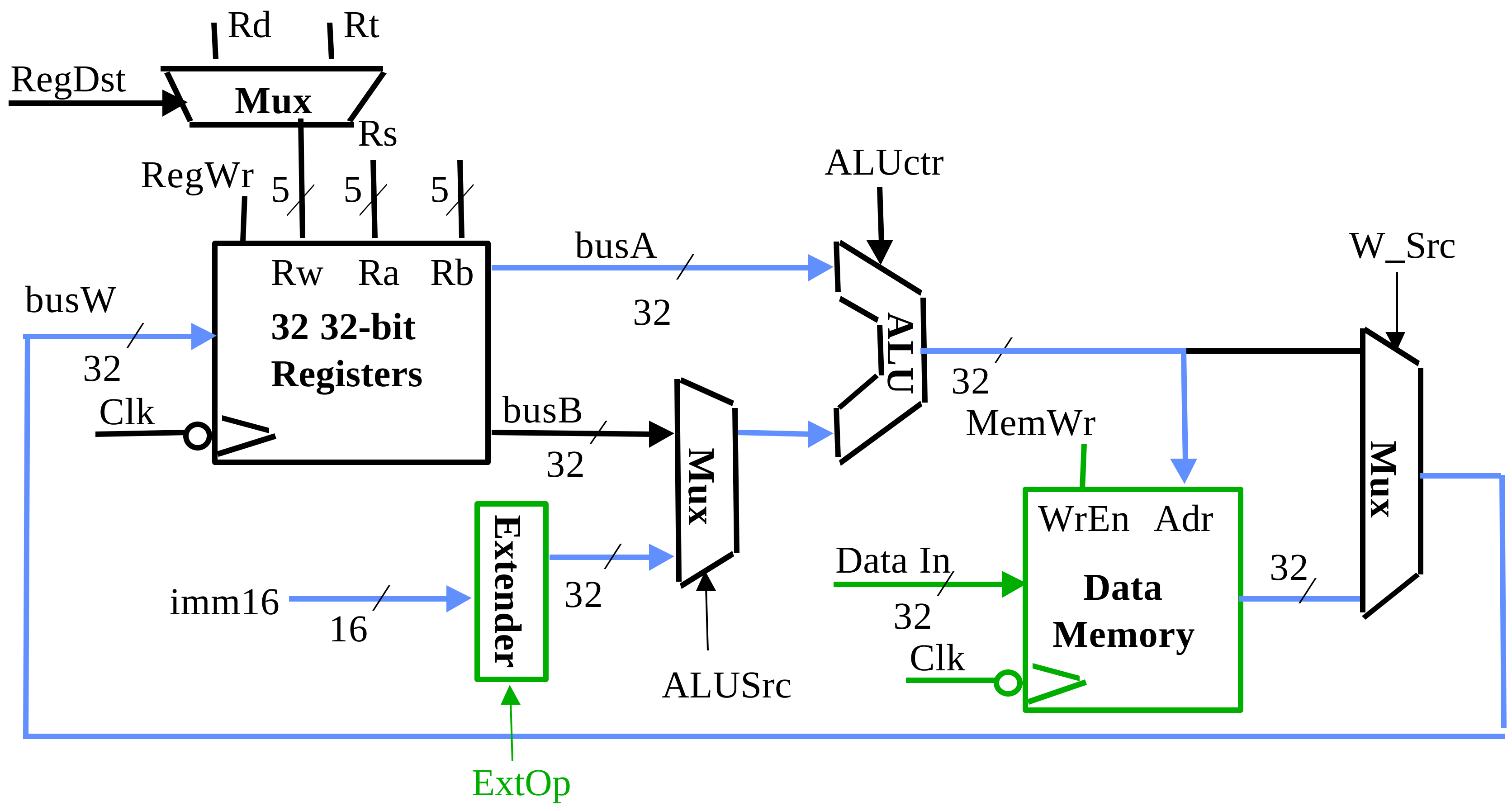


b. Sign-extension unit

Inputs for address and write (store) data
Output for read (load) result

16-bit input sign-extended into a 32-bit value at the output

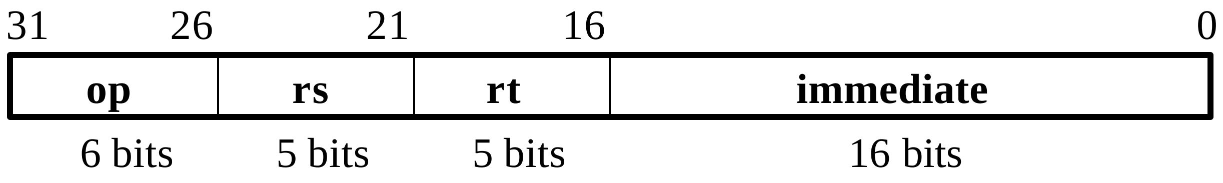
Datapath For Loads



Store Operations Example:

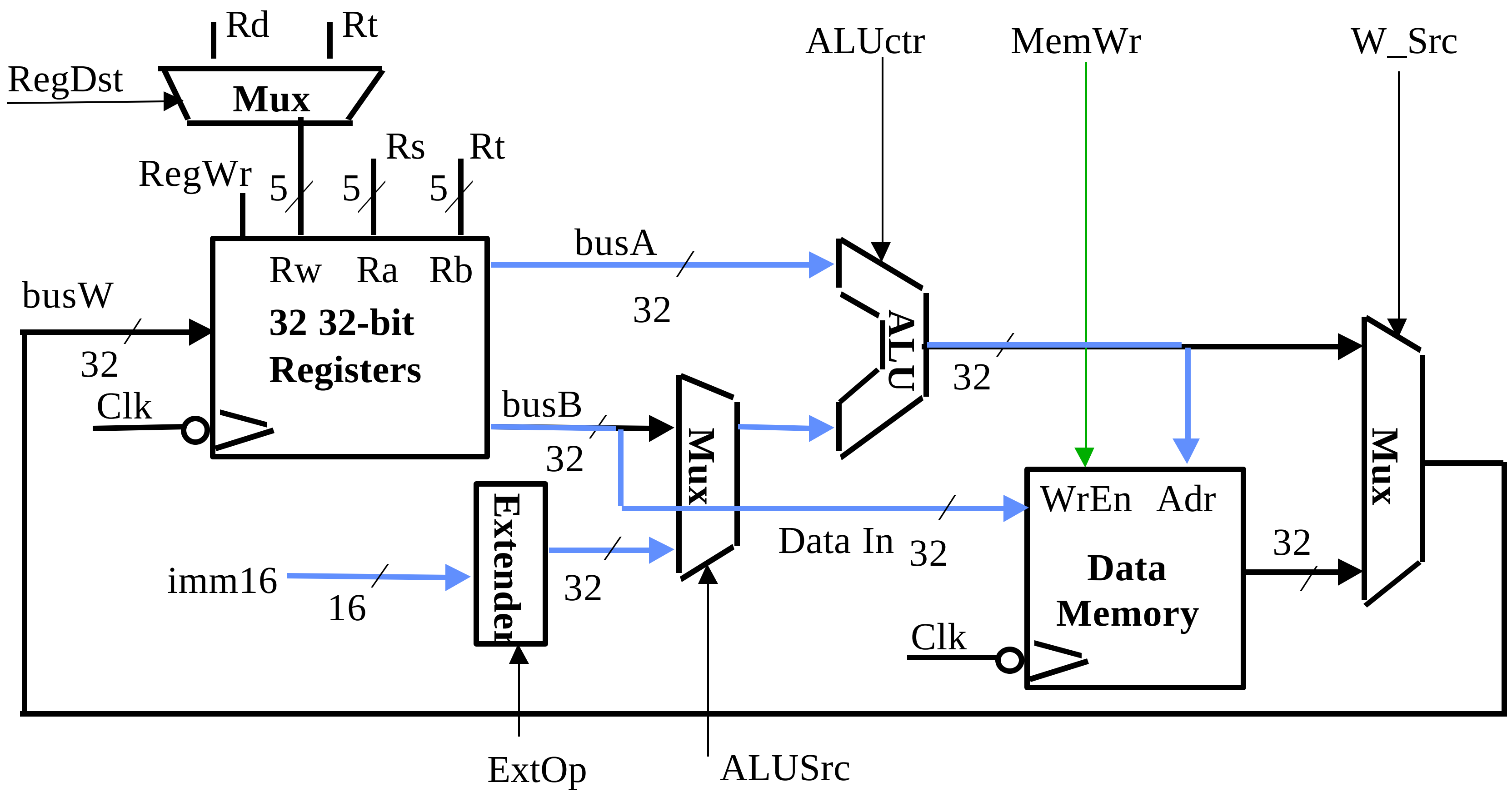
Micro-Operation Sequence For SW

sw rt, rs, imm16



Instruction Word $\leftarrow$ Mem[PC]	**Fetch the instruction**
PC $\leftarrow$ PC + 4	**Increment PC**
Mem[R[rs] + SignExt[imm16]] $\leftarrow$ R[rt]	**Immediate field sign extended to 32 bits and added to register rs to form memory store address, register rt written to memory at store address.**

Datapath For Stores



Conditional Branch Example:

Micro-Operation Sequence For BEQ

beq rs, rt, imm16

31 … 26	… 21	… 16	… 0
op	**rs**	**rt**	**immediate**
6 bits	5 bits	5 bits	16 bits

Instruction Word ← Mem[PC]	**Fetch the instruction**
PC ← PC + 4	**Increment PC**
Equal ← R[rs] == R[rt]	**Calculate the branch condition**

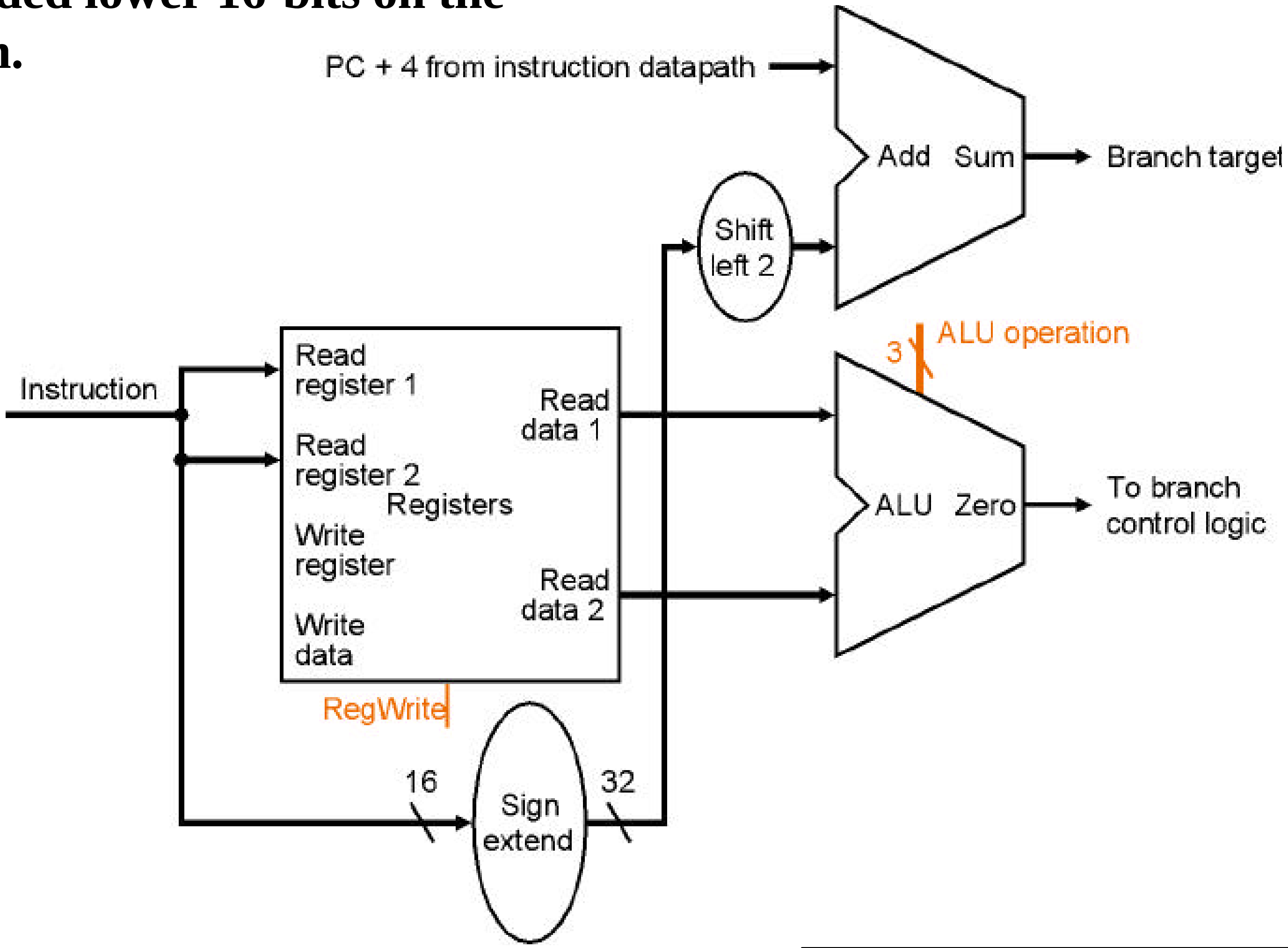
```
if (COND eq 0)
        PC ← PC + 4 + ( SignExt(imm16) x 4 )
    else
        PC ← PC + 4
```

Calculate the next instruction's PC address

Datapath For Branch Instructions

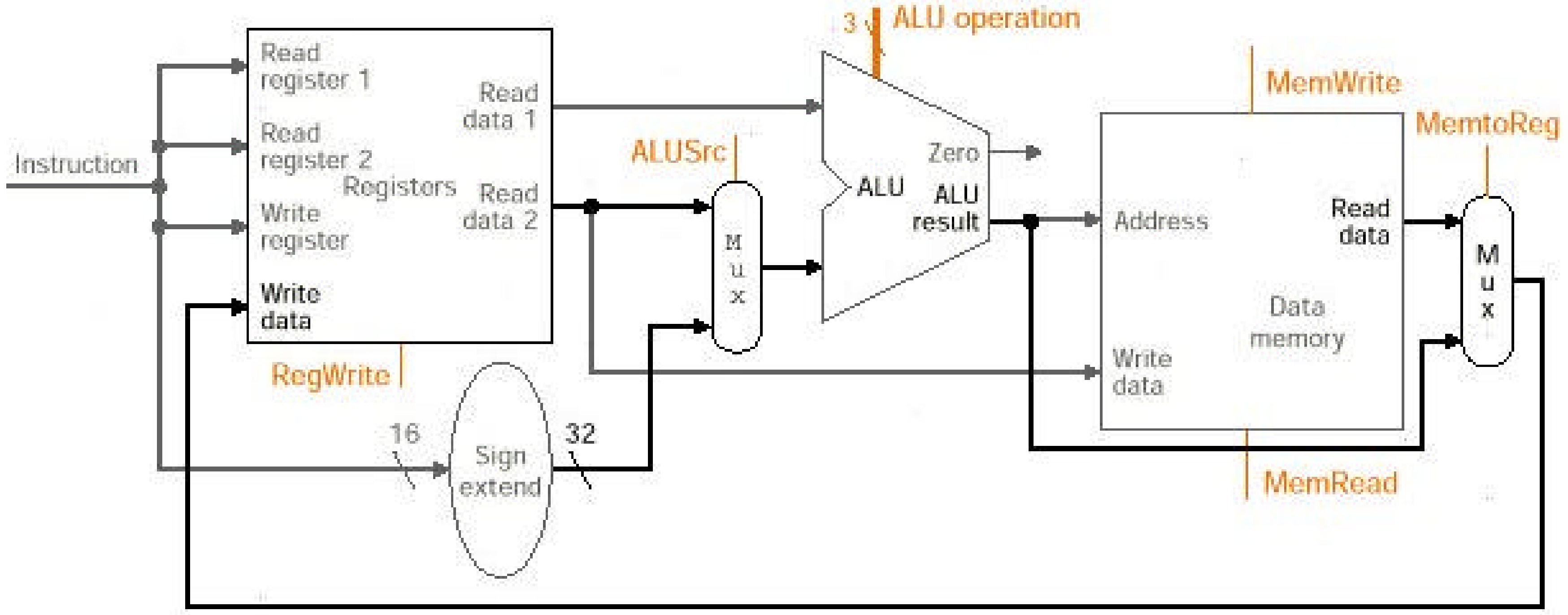
ALU to evaluate branch condition
Adder to compute branch target:

- **Sum of incremented PC and the sign-extended lower 16-bits on the instruction.**



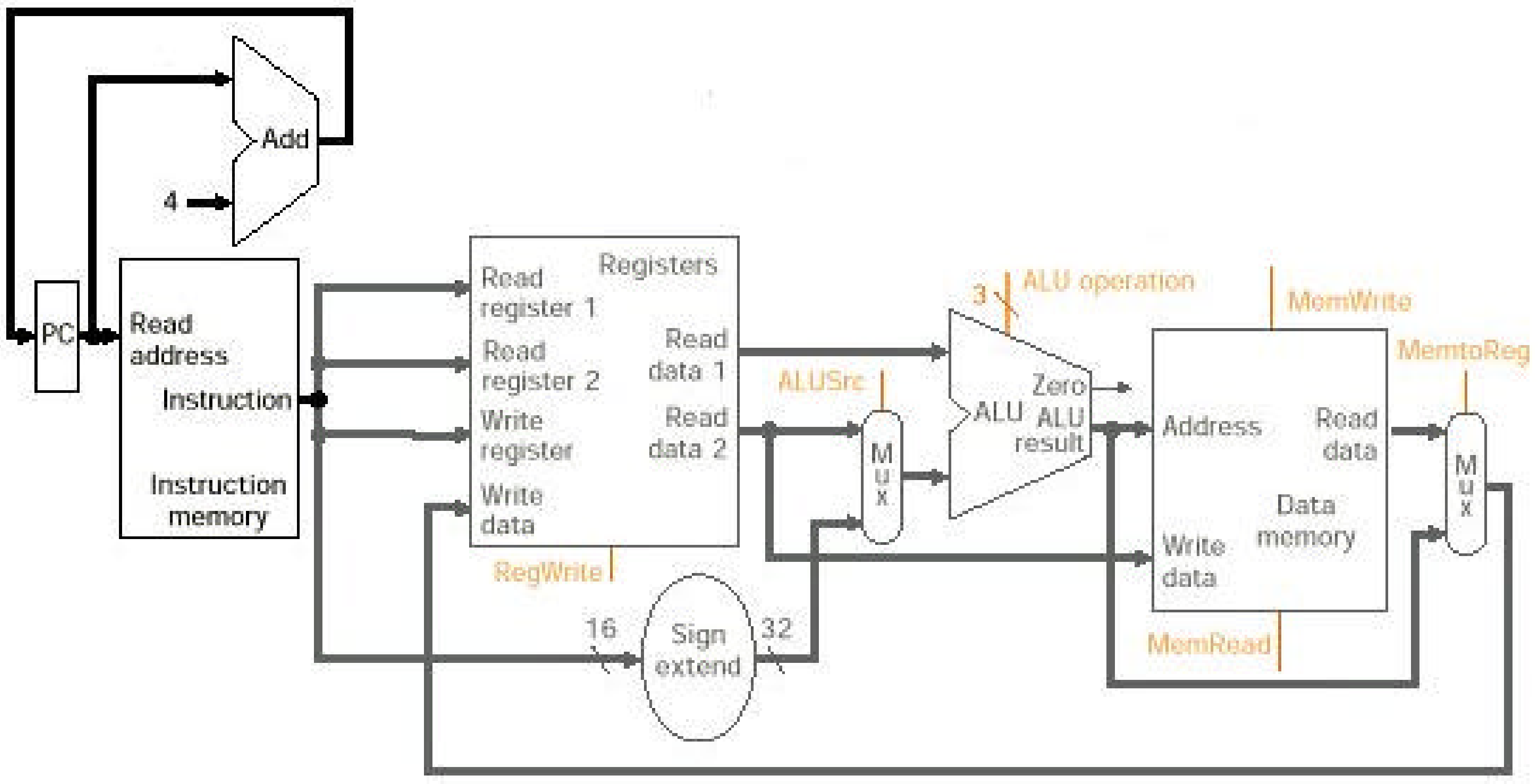
EECC250 - Shaaban

Combining The Datapaths For Memory Instructions and R-Type Instructions



Highlighted muliplexors and connections added to combine the datapaths of memory and R-Type instructions into one datapath

Instruction Fetch Datapath Added to ALU R-Type and Memory Instructions Datapath

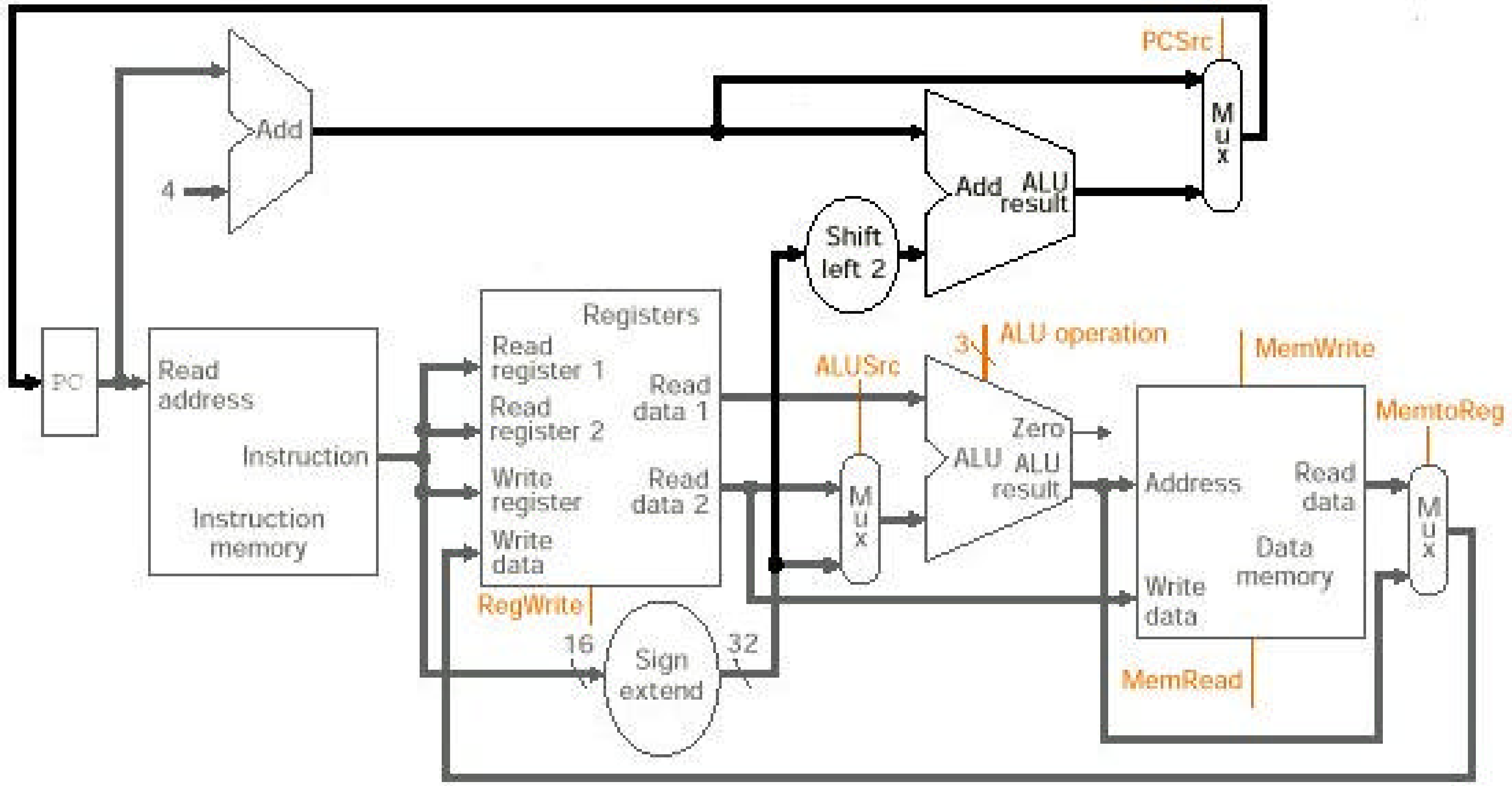


A Simple Datapath For The MIPS Architecture

Datapath of branches and a program counter multiplexor are added.

Resulting datapath can execute in a single cycle the basic MIPS instruction:

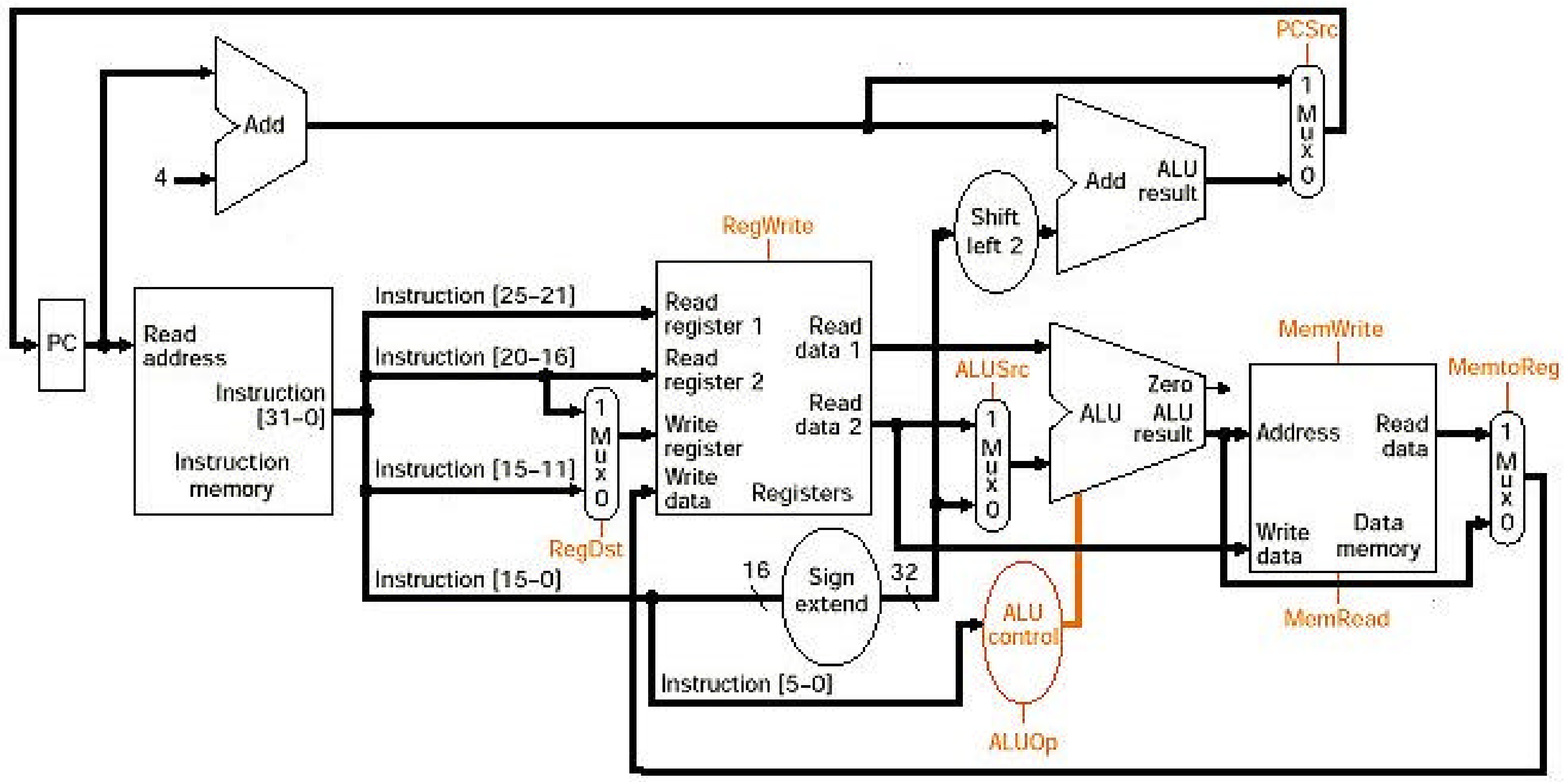
- load/store word - ALU operations - Branches



EECC250 - Shaaban

Single Cycle MIPS Datapath

Necessary multiplexors and control lines are identified here:



Adding Support For Jump:

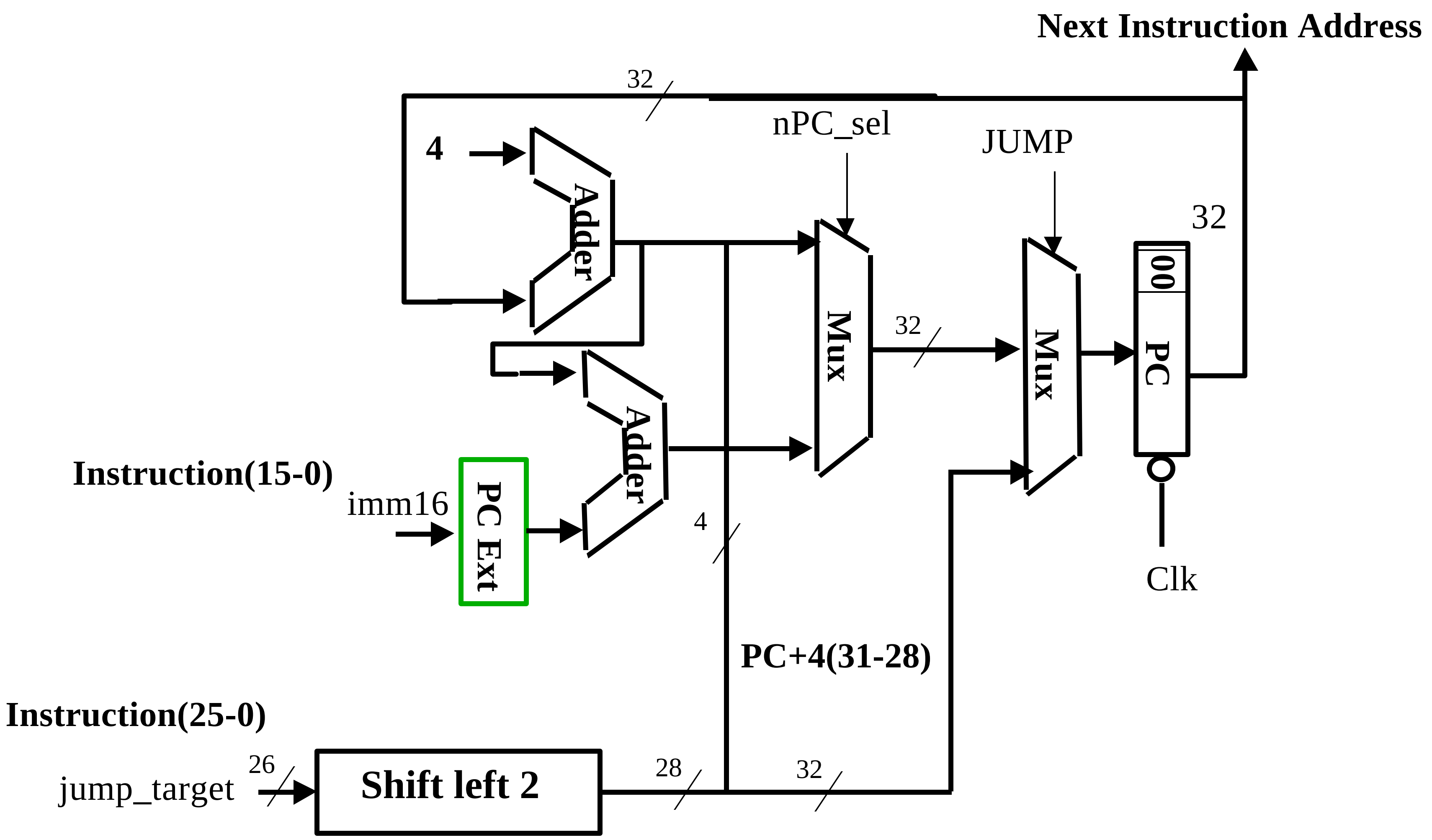
Micro-Operation Sequence For Jump: J

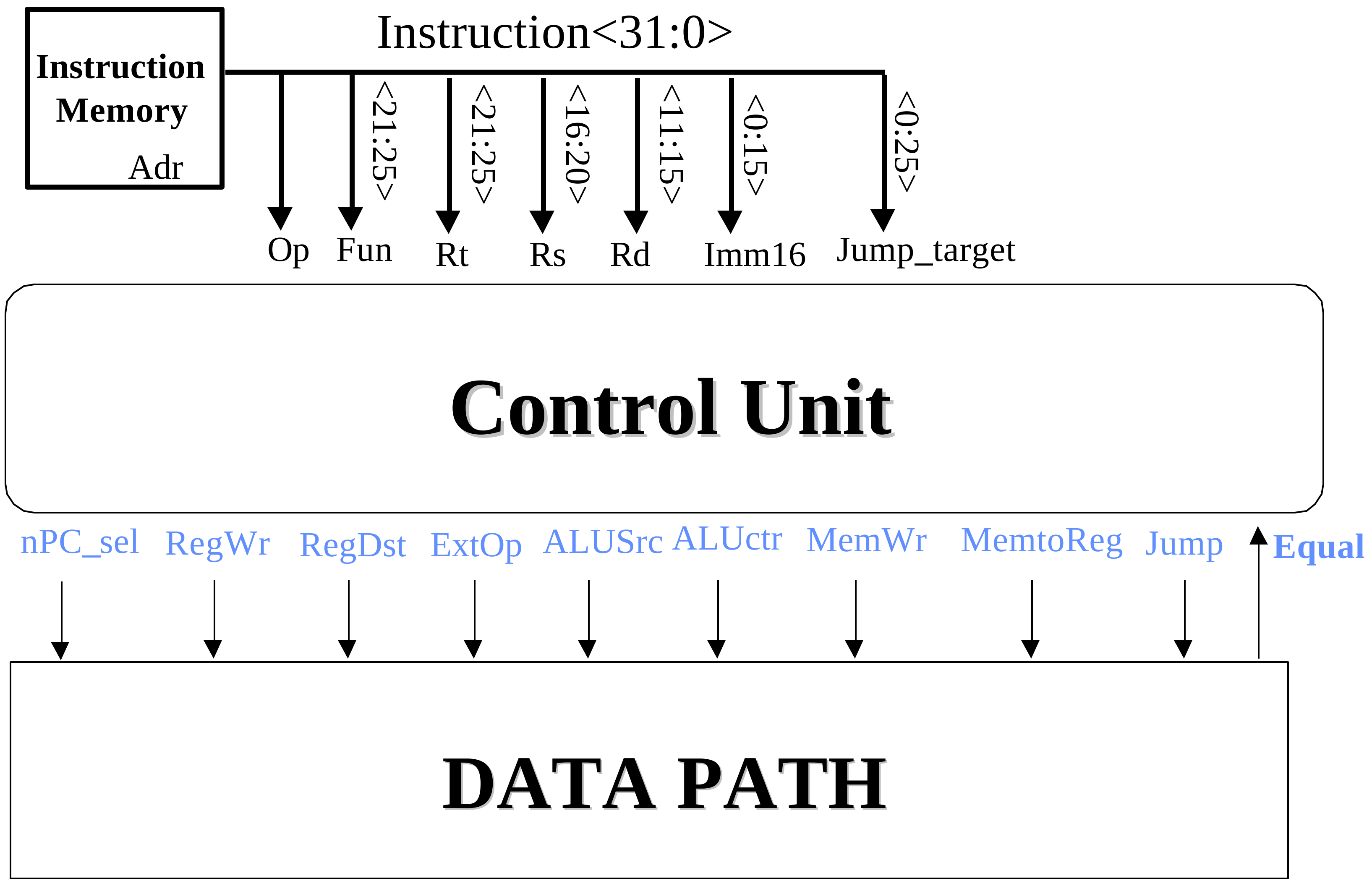
j jump_target

OP	**Jump_target**
6 bits	**26 bits**

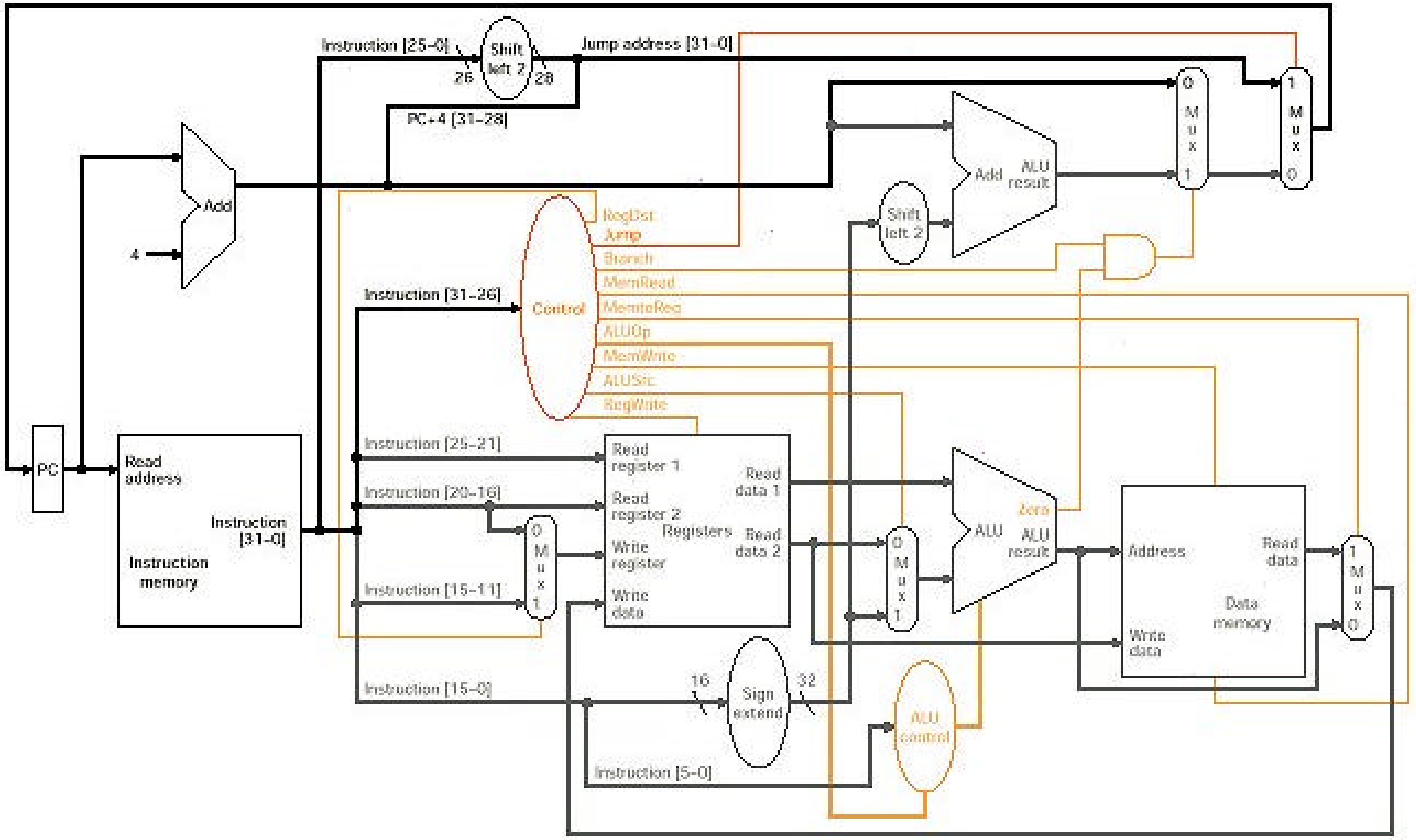
Instruction Word $\leftarrow$ Mem[PC]	**Fetch the instruction**
PC $\leftarrow$ PC + 4	**Increment PC**
PC $\leftarrow$ PC(31-28),jump_target,00	**Update PC with jump address**

Datapath For Jump




Instruction Memory
Adr
Instruction<31:0>
<21:25>
<21:25>
<16:20>
<11:15>
<0:15>
<0:25>
Op
Fun
Rt
Rs
Rd
Imm16
Jump_target
Control Unit
nPC_sel
RegWr
RegDst
ExtOp
ALUSrc
ALUctr
MemWr
MemtoReg
Jump
Equal
DATA PATH

Single Cycle MIPS Datapath Extended To Handle Jump with Control Unit Added

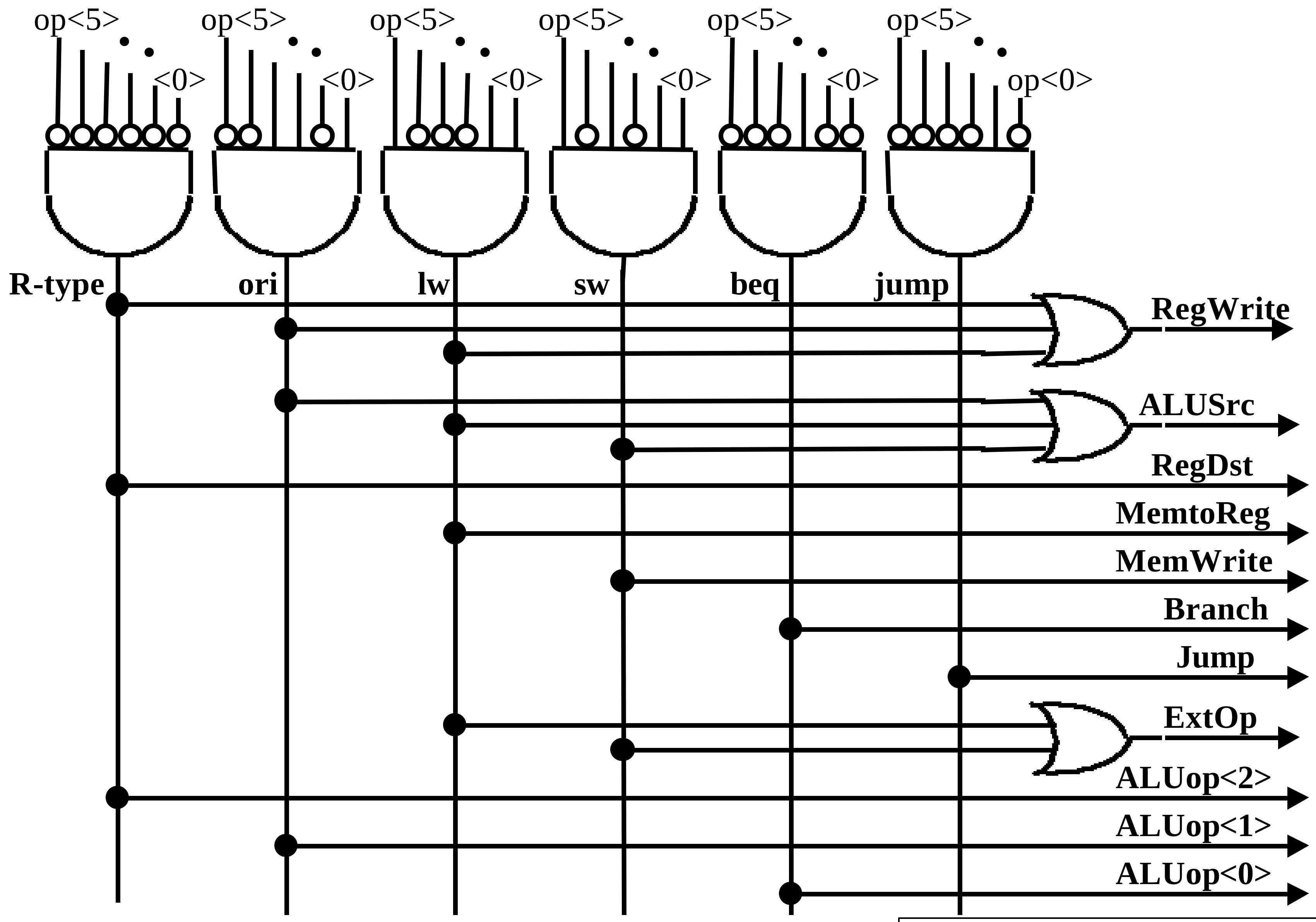


EECC250 - Shaaban

Control Signal Generation

	add	sub	ori	lw	sw	beq	jump
func	10 0000	10 0010	**Don't Care**				
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
RegDst	1	1	0	0	x	x	x
ALUSrc	0	0	1	1	1	0	x
MemtoReg	0	0	0	1	x	x	x
RegWrite	1	1	1	1	0	0	0
MemWrite	0	0	0	0	1	0	0
nPCsel	0	0	0	0	0	1	0
Jump	0	0	0	0	0	0	1
ExtOp	x	x	0	1	1	x	x
ALUctr<2:0>	Add	Subtract	Or	Add	Add	Subtract	xxx

PLA Implementation of the Main Control



EECC250 - Shaaban