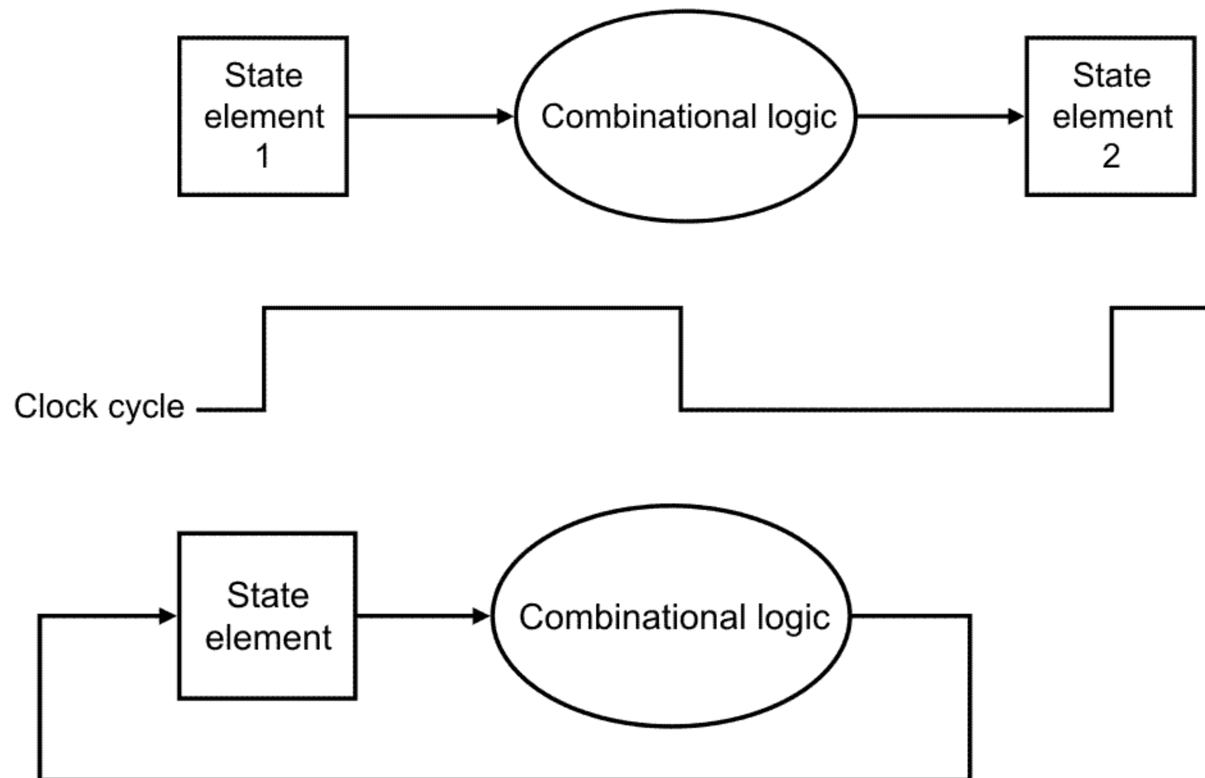
Single-Cycle Architecture

DMCS

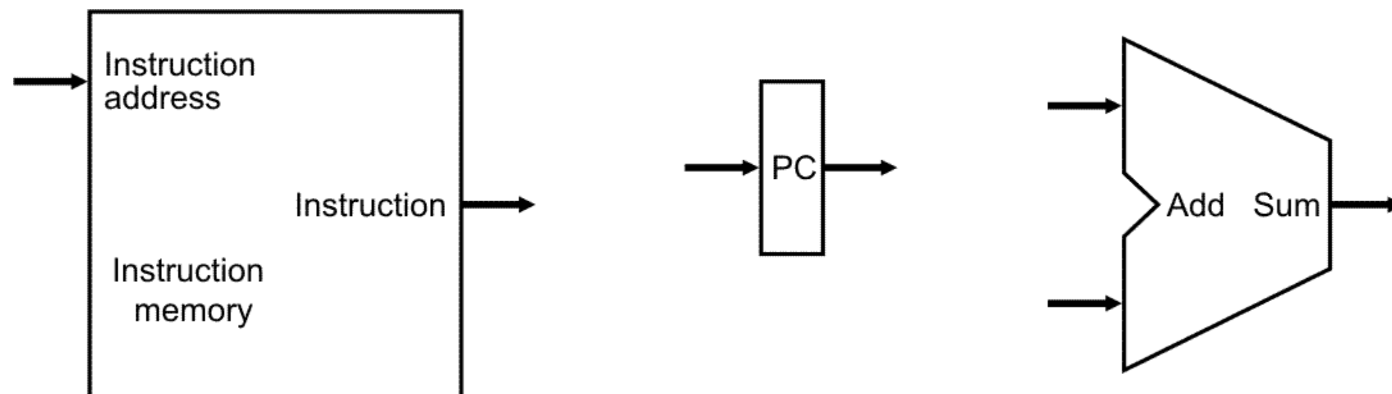
Data flow

- Data flow is synchronized with clock (edge) in sequential systems



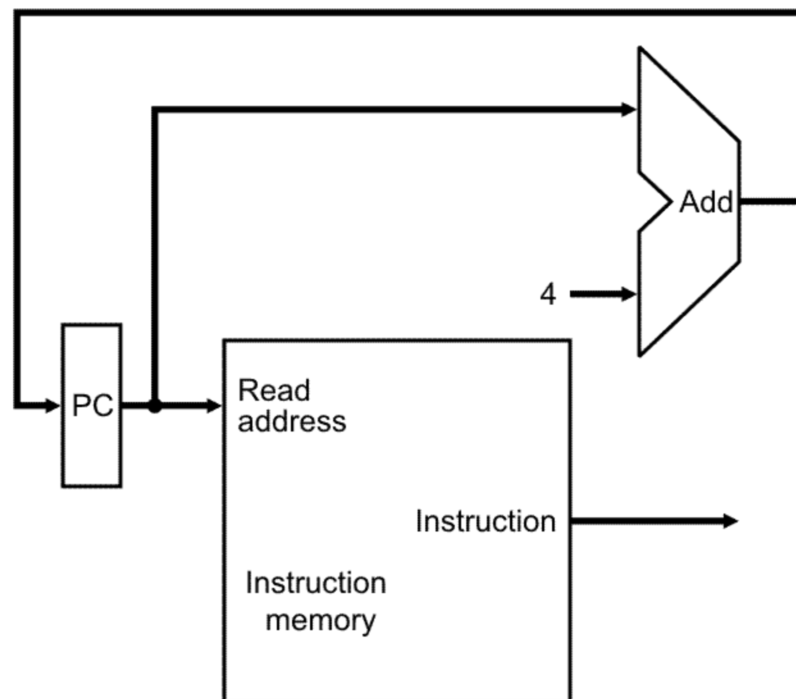
Architecture Elements - assumptions

- Program (Instruction) memory:
  - All instructions & buses are 32-bit wide (4 Bytes)
  - Instruction code is available at Instruction bus after instruction code is provided at *Instruction address* bus
- Register PC contains address to instruction
- Adder operates on 32-bit numbers



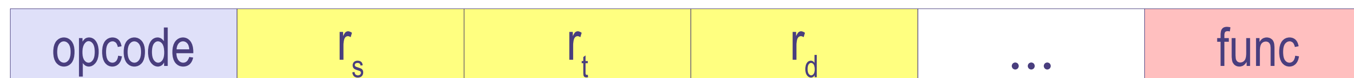
Instruction Fetch Block

- *Instruction Fetch* operation
  - PC-write operation is triggered with clock signal
  - PC is incremented by 4 in every clock cycle
  - A sequence of instructions is fetched from memory



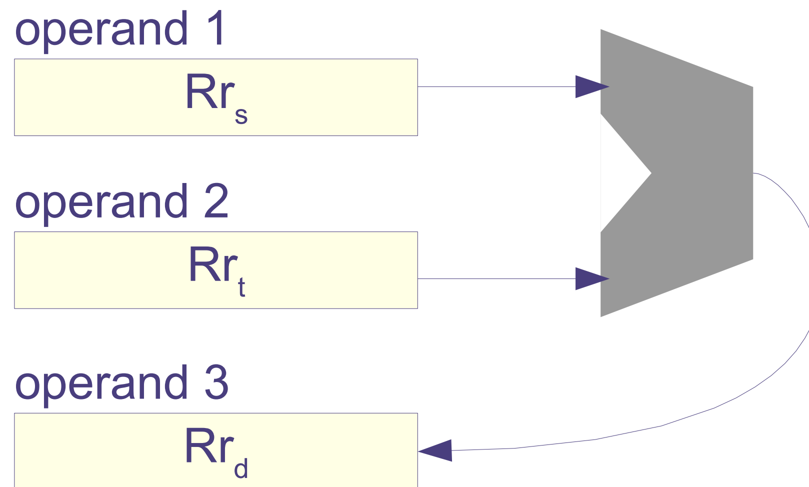
Register-type Instructions (*R-type*)

- R-type instructions perform operation only on (contents of) internal registers of processor
  - two source operands (Rr_s i Rr_t) are in internal registers
  - results is written to the internal register (Rr_d)
- R-type instruction code is composed of:
  - unique number of instruction type (*opcode*)
  - numbers of 3 registers (2x source and 1 result): r_s, r_t, r_d
  - type of arithmetical or logical operation (*func*)



Register Direct Addressing

- (Adresowanie bezpośrednie rejestrowe)
- Operands are in internal registers

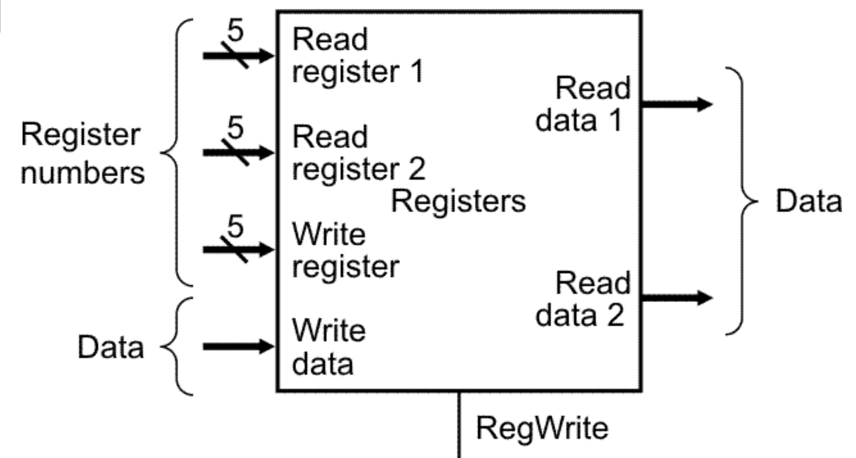


assembler:

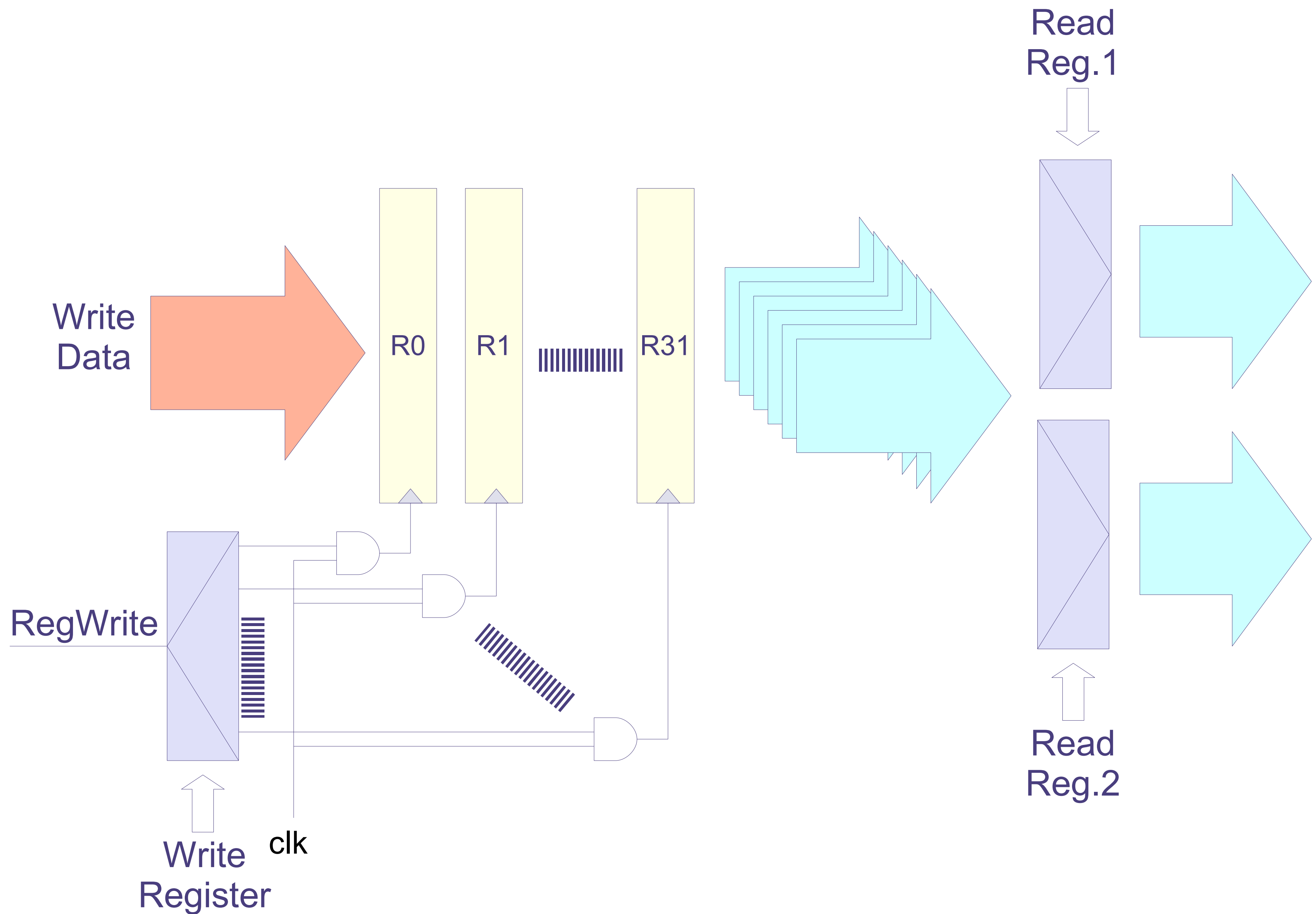
```
ADD R1,R2,R7
SUB R3,R6,R1
OR  R7,R3,R2
AND R0,R2,R5
```

Register File - assumptions

- Contains 32 registers, each 32-bit wide
- At register file output provides contents of two registers addressed by *ReadRegister1&2* input numbers
- Register numbers are 5-bit wide ($2^5 = 32$)
- Writing to a selected internal register requires: the register number (*WriteRegister*), data to be written (*WriteData*) and operation enable signal (*RegWrite*)
- Write operation is synchronized with the clock signal

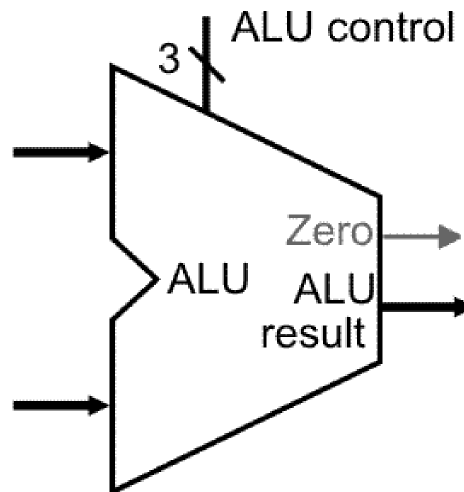


Register File – Logical Concept



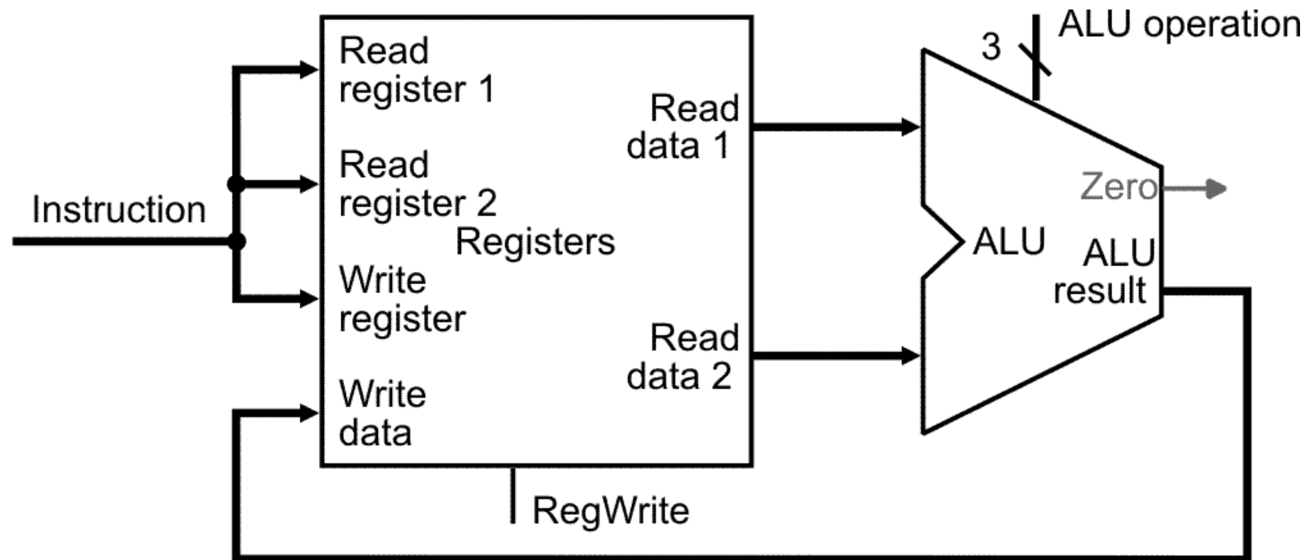
ALU - assumptions

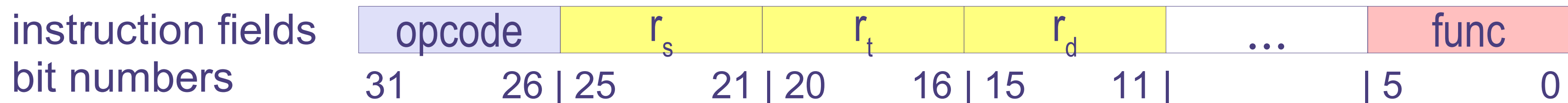
- 32-bit input & outputs (32-bit ALU)
- Operations: add, subtract, logical: AND, OR
- 3-bit ALU-control bus
- Only one output control signal: Zero (Z)



R-type Instruction Execution

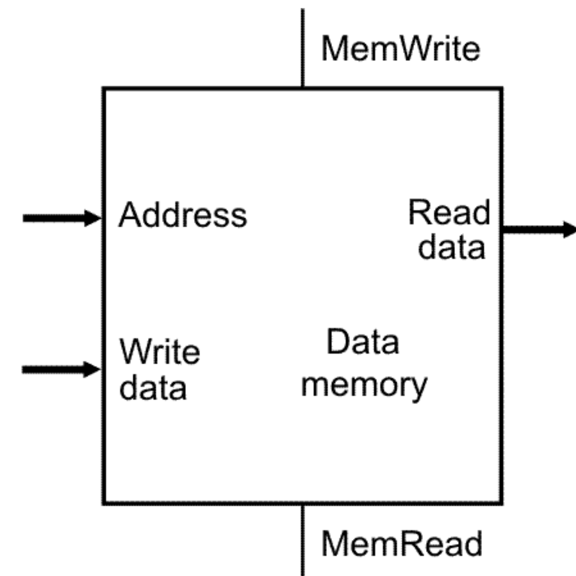
- Source operands from Register File (Rr_s i Rr_t) are selected by register numbers from instruction code
- ALU result is written back to the register selected by Rr_d at the end of clock signal cycle





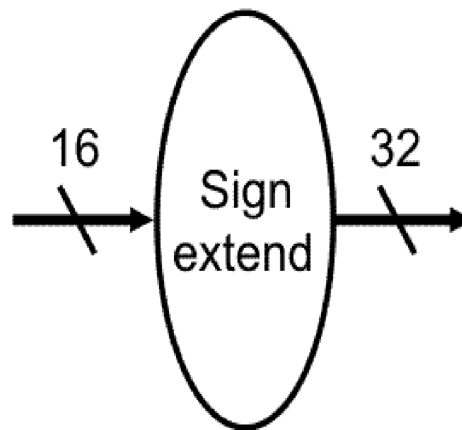
Data Memory - assumptions

- Contains data of the program, organized in 32-bit words (4B)
- Data is present at *ReadData* bus after the address is provided at *Address* bus and operation enable signal is active (*MemRead*)
- Memory modification requires the data and address to be present at *WriteData* and *Address* buses and operation enable signal active (*MemWrite*)
- Write operation is synchronized with the clock signal



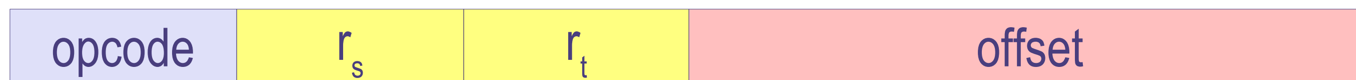
Sign Extension Unit - assumptions

- Performs conversion of 16-bit binary numbers to 32-bit representation with proper sign handling (2C)
- Combinatorial logic, no clock signal required



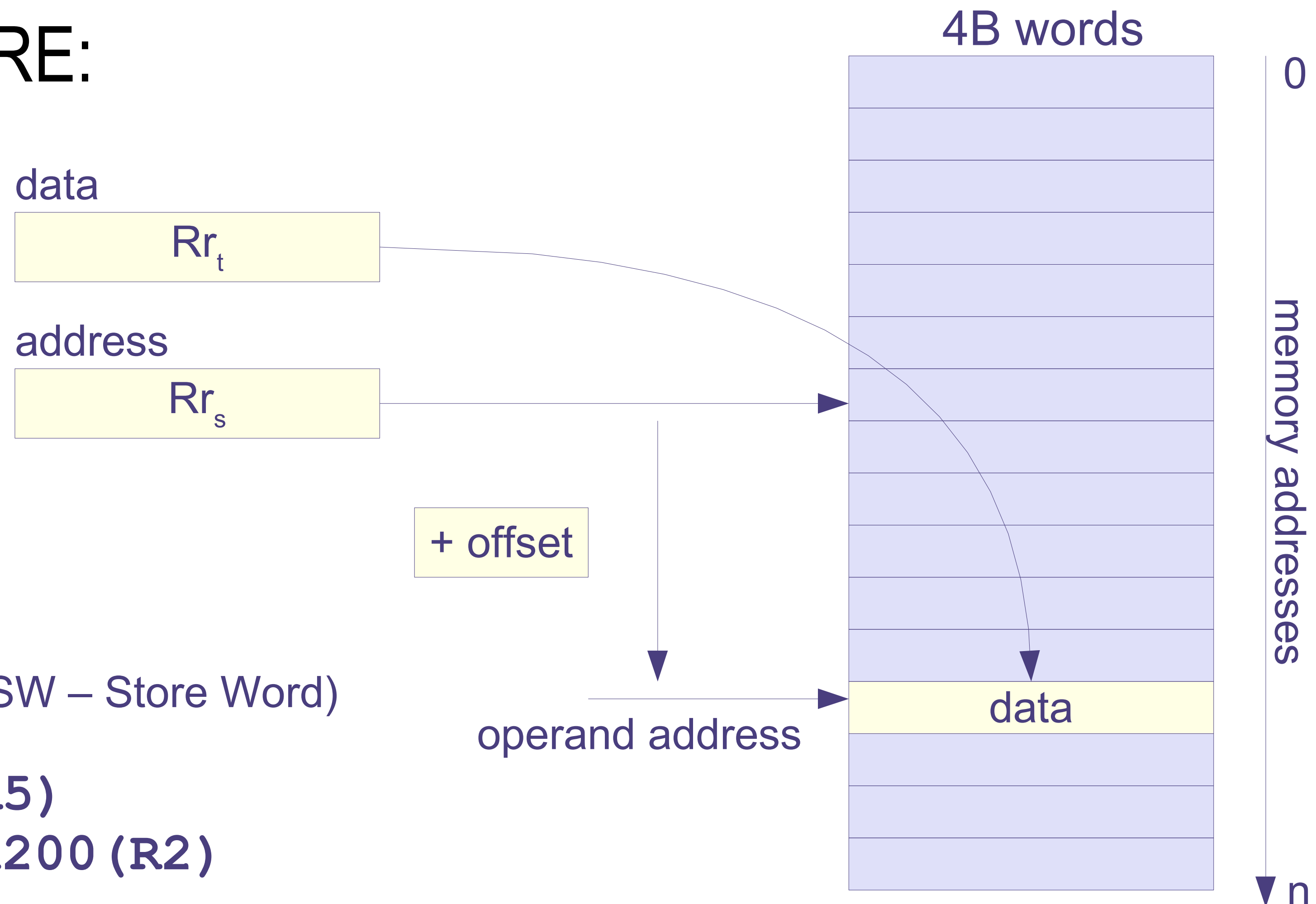
Transfer Instructions (*Load/Store*)

- Instructions perform data transfer:
  - from data memory to internal register (*Load*)
  - from internal register to data memory (*Store*)
  - transfer info: internal register (Rr_s) & memory address (Rr_t)
  - constant value (offset) extends addressing range
- Load/Store instruction code is composed of:
  - unique number of instruction type (opcode)
  - numbers of two internal registers: r_s, r_t
  - constant (*offset*), added to the memory (base) address



Register Indirect Addressing

- with Offset (Adresowanie pośrednie rejestrowe z przesunięciem)
- Offset is a signed number in 2's complement
- STORE:

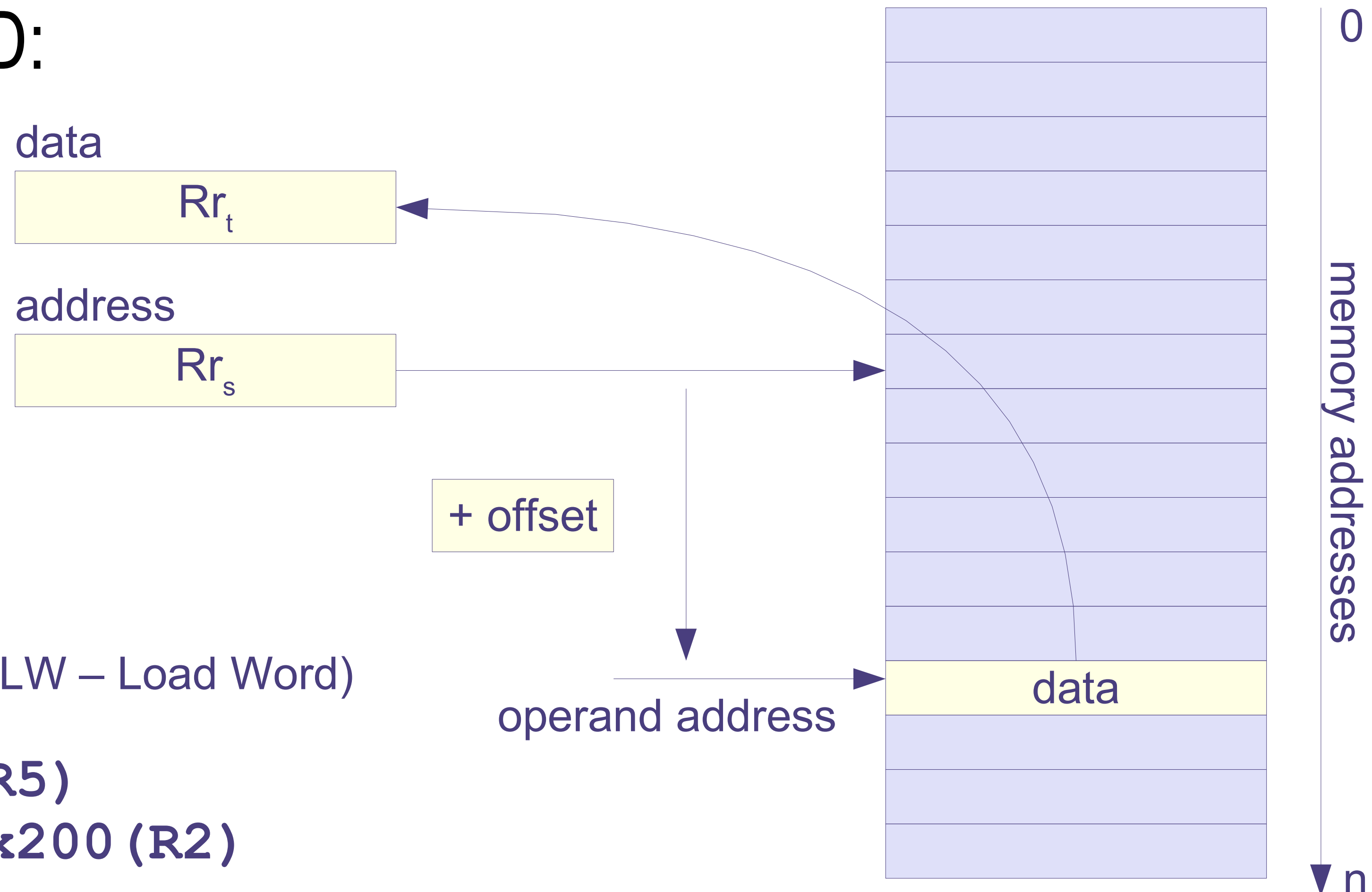


assembler: (SW – Store Word)

```
SW R7,(R5)
SW R1,0x200(R2)
```

Register Indirect Addressing

- with Offset (Adresowanie pośrednie rejestrowe z przesunięciem)
- Offset is a signed number in 2's complement
- LOAD:

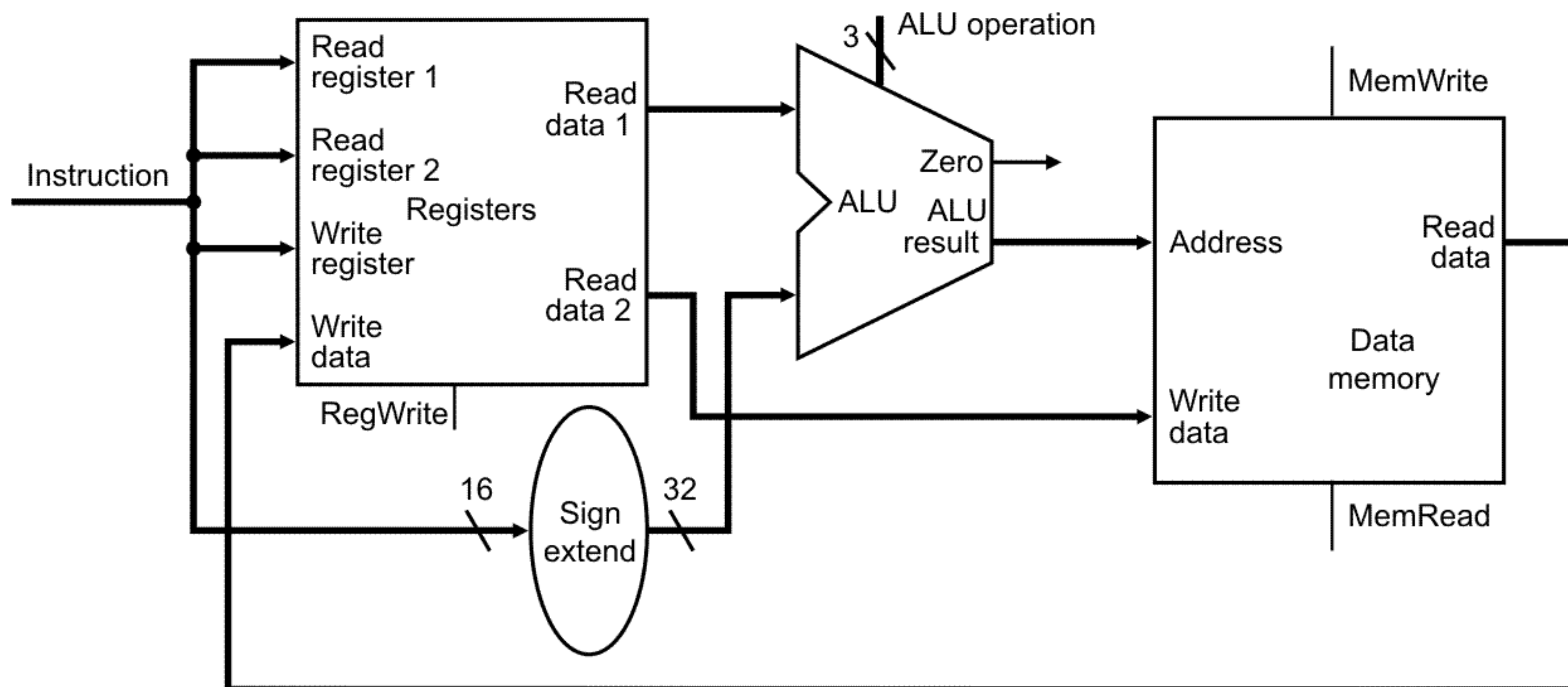



```
LW R7,(R5)
LW R1,0x200(R2)
```

Computer Architecture, IFE CS and T&CS, 4th sem

Load/Store *Execution*

- Register Rr_s + offset points to memory data (Load/Store)
- Store: Contents of register Rr_t (memory input) to be written (signal *MemWrite* active)
- Load: memory output to be written to register Rr_t (signals *MemRead and RegWrite* active)

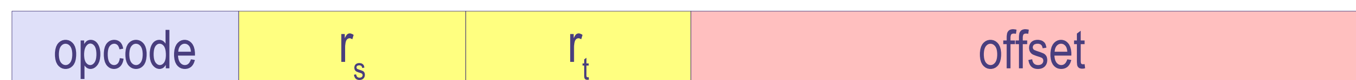


Jump/Branch Instructions

- Jump/Branch: interruption in execution of a sequence of consecutive instructions in memory
- Every Jump/Branch is a modification of the PC register
- Absolute (jumps) vs Relative (branches)
  - absolute – arbitrary new content loaded into PC
  - relative – offset added to the current value of PC
- Unconditional vs Conditional
  - unconditional – jump is always performed
  - conditional – final modification of PC depends on conditions usually provided by ALU (bits C,V,Z,N)

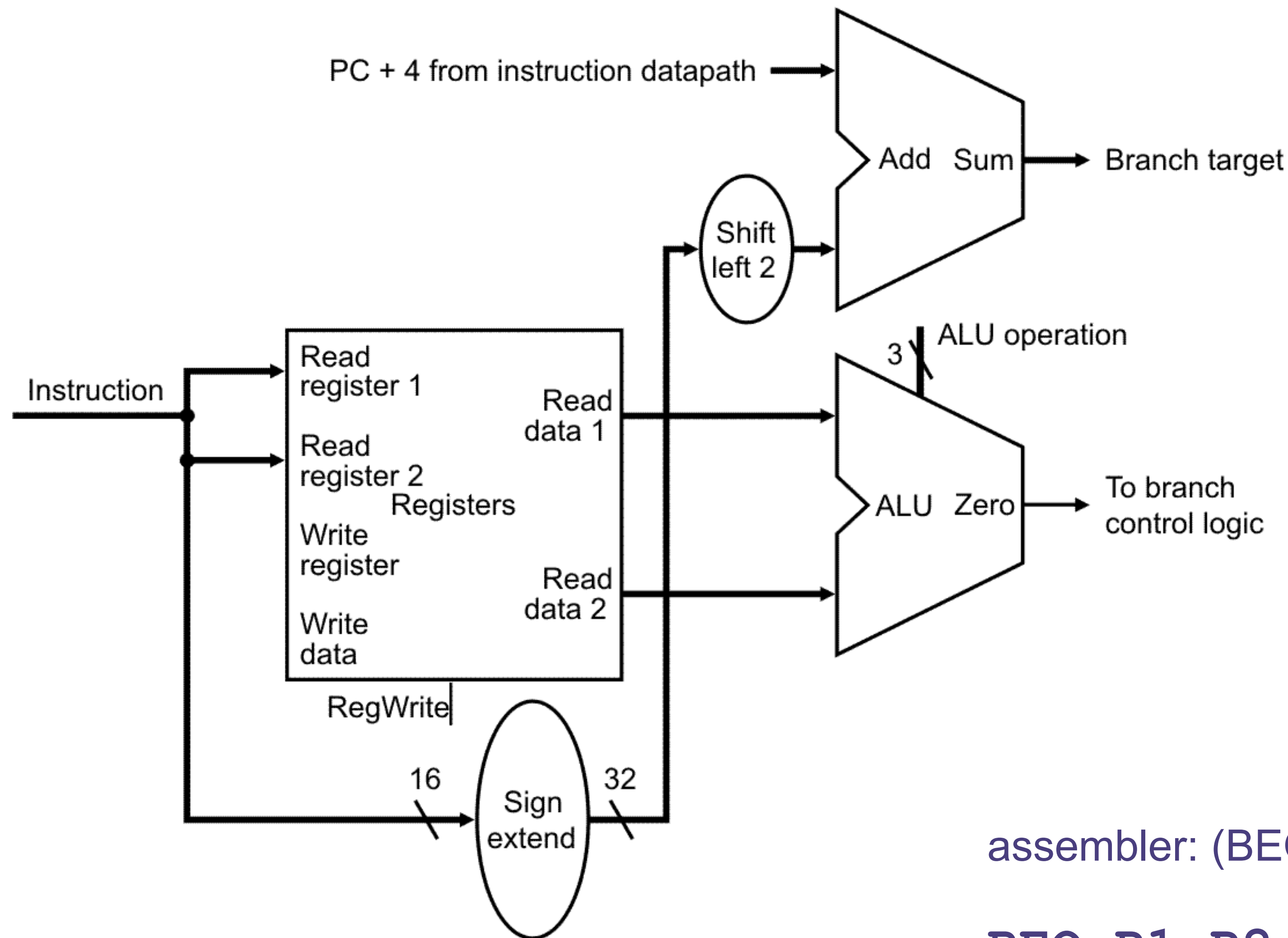
Relative Conditional Branch (BEQ)

- **`BEQ Rx,Ry,offset`** (Branch if Equal)
  – branch to address PC+offset*4 if Rx=Ry (Z=1)
- Instructions are 4B long, so branch range can be widened by pointing to every fourth byte (offset*4)
- BEQ instruction code is composed of:
  - unique number of instruction type (opcode)
  - numbers of two internal registers: r_s, r_t
  - constant (offset), added to the memory (base) address
- BEQ base address refers to instruction memory



Computer Architecture, IFE CS and T&CS, 4th sem

Relative Conditional Branch (BEQ)

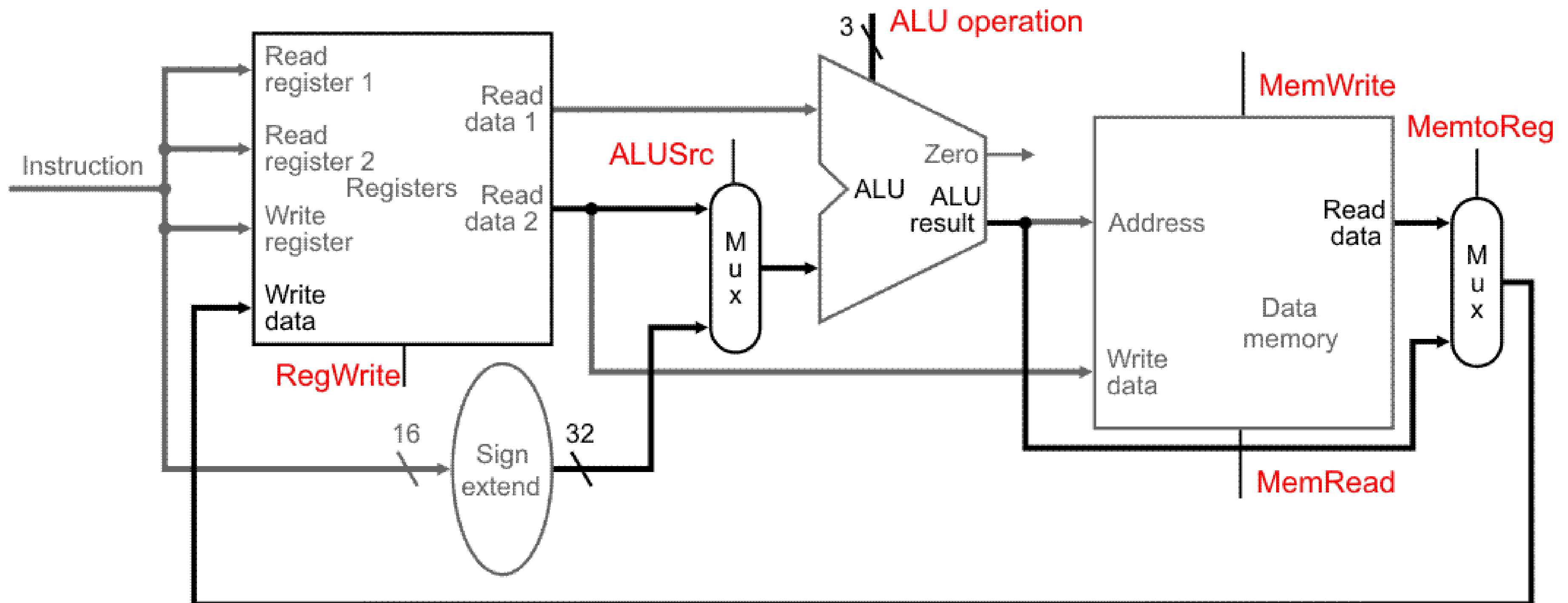


assembler: (BEQ)

```
BEQ R1,R2,0x40
BEQ R0,R7,0xFF
```

R-type and *Load/Store* Together

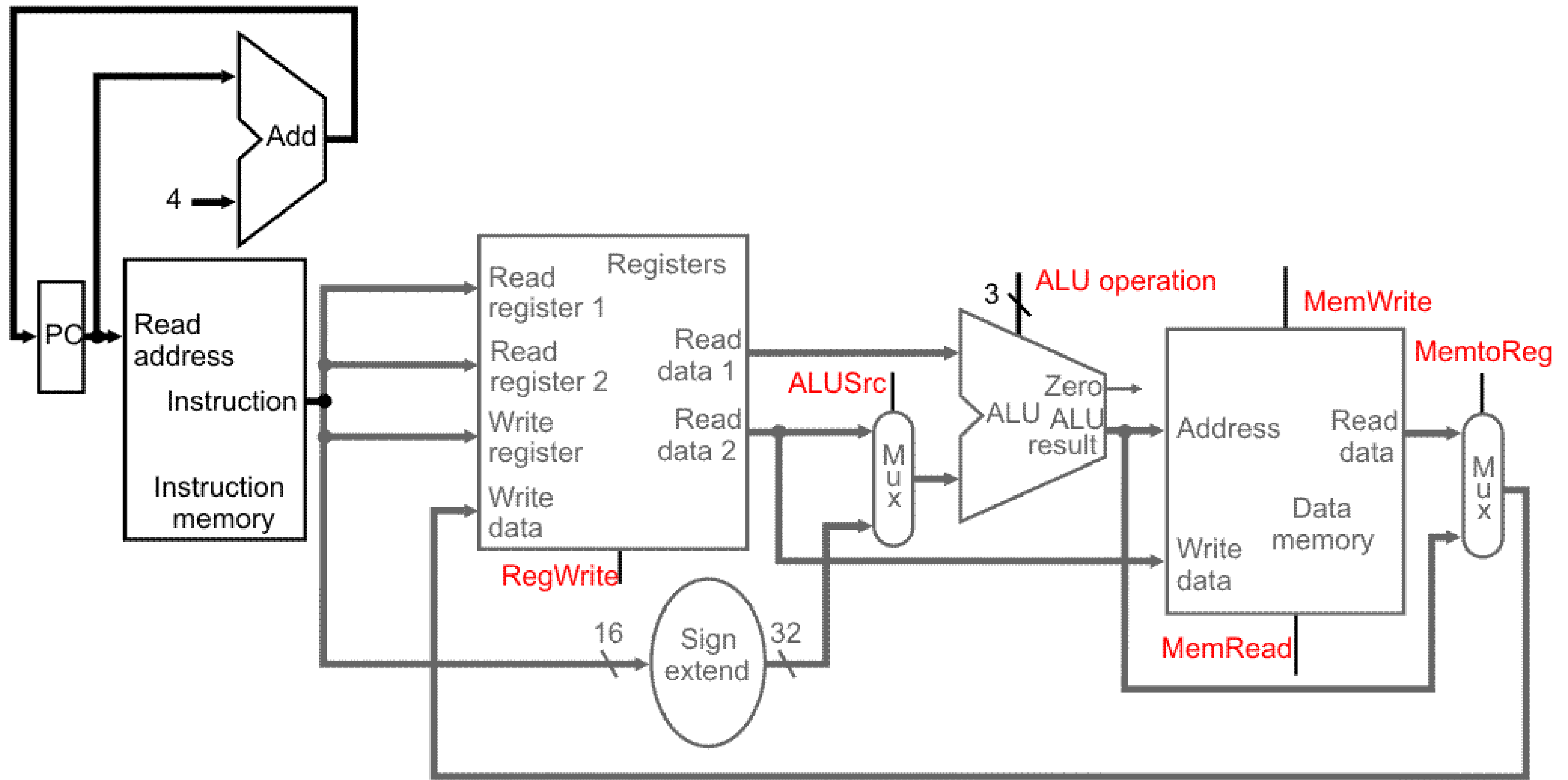
- Bus multiplexers:
  - ALUSrc: selects the second ALU operand
  - MemtoReg: selects the data to be written to a register



Computer Architecture, IFE CS and T&CS, 4th sem

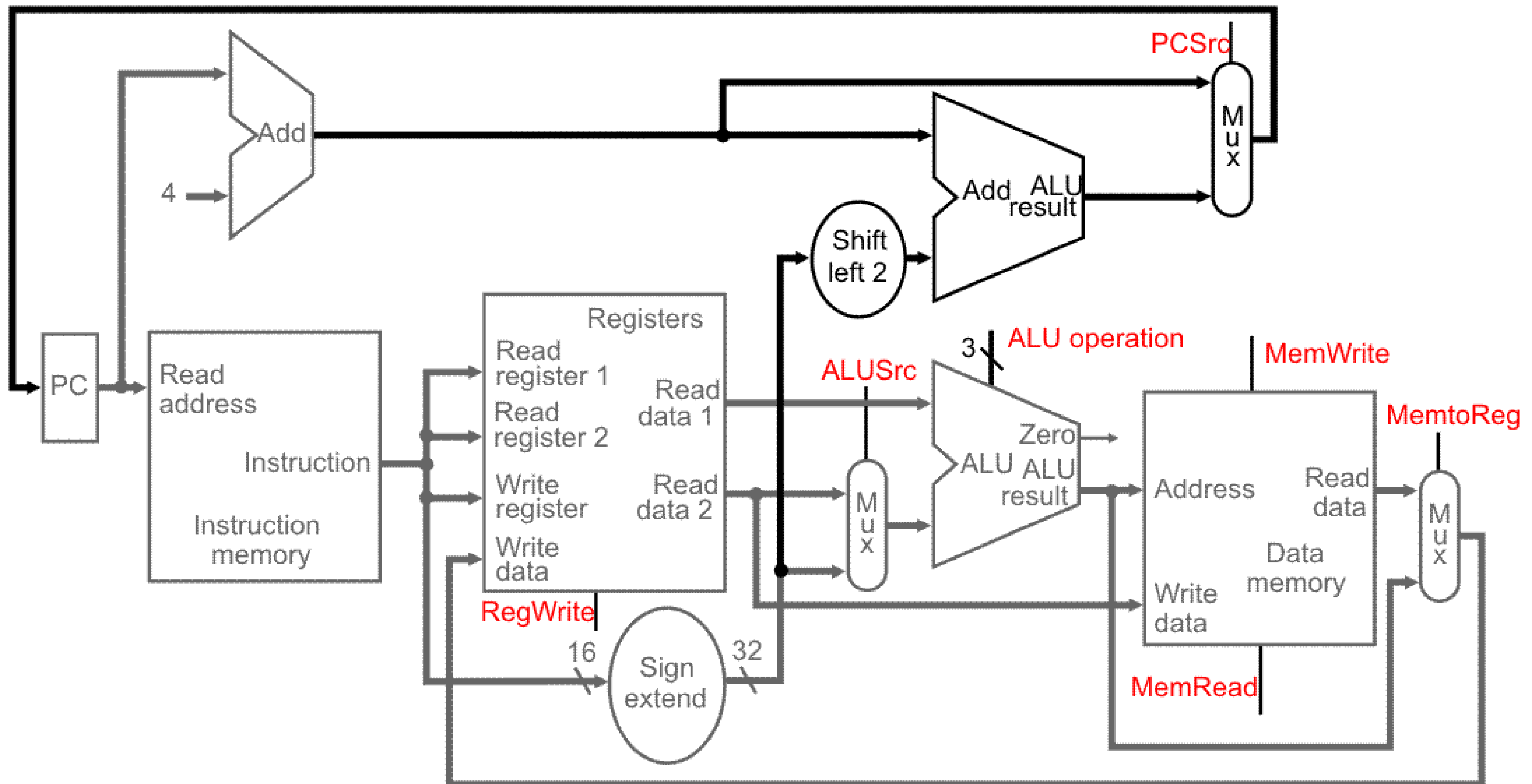
DMCS

Fetch Unit + R-type, Load/Store



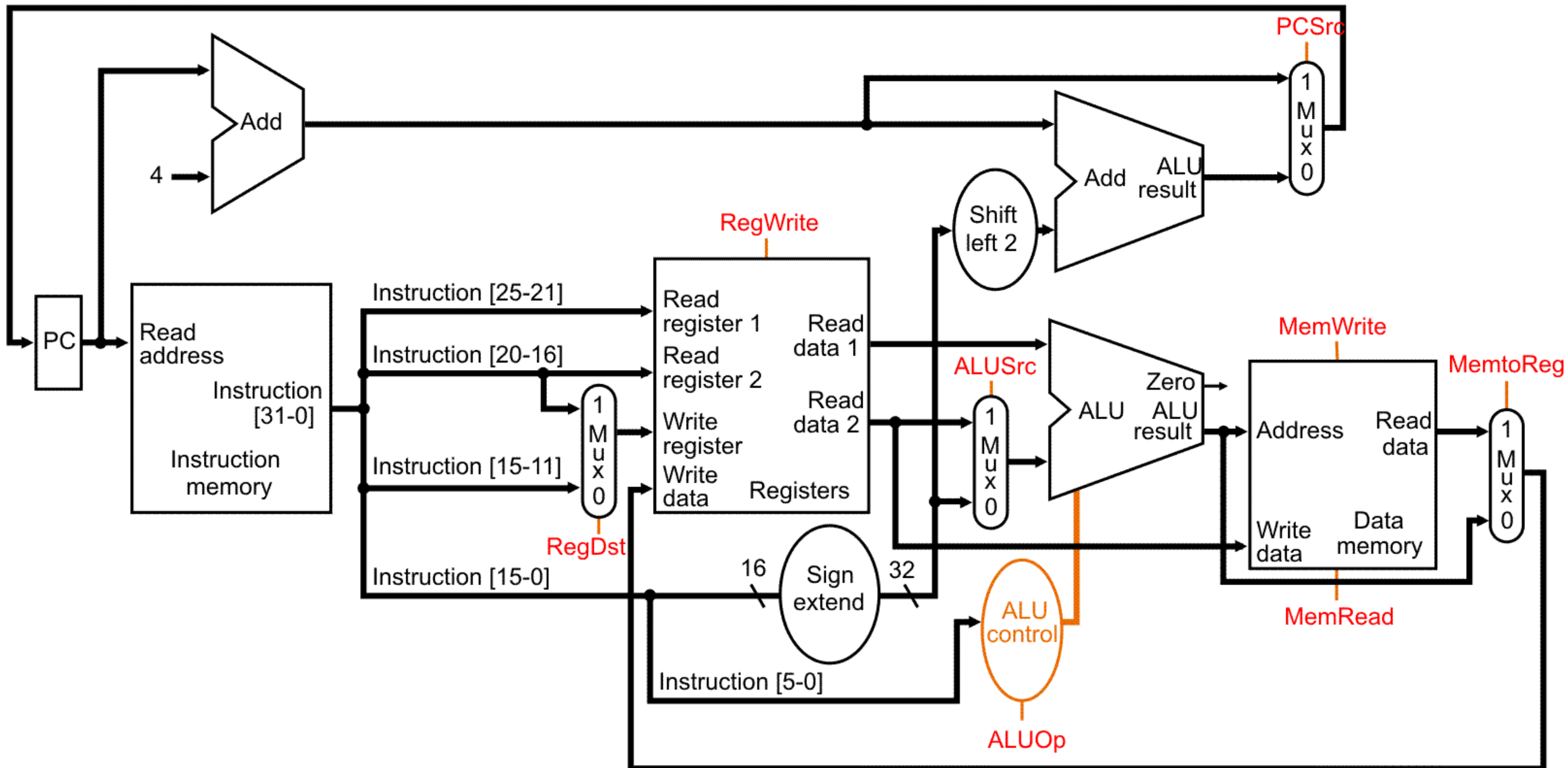
Fetch + R-type, Load/Store and Branch

- Multiplexer: PCSrc
  - selects the source for new PC value:
    PC+4 or PC+offset*4



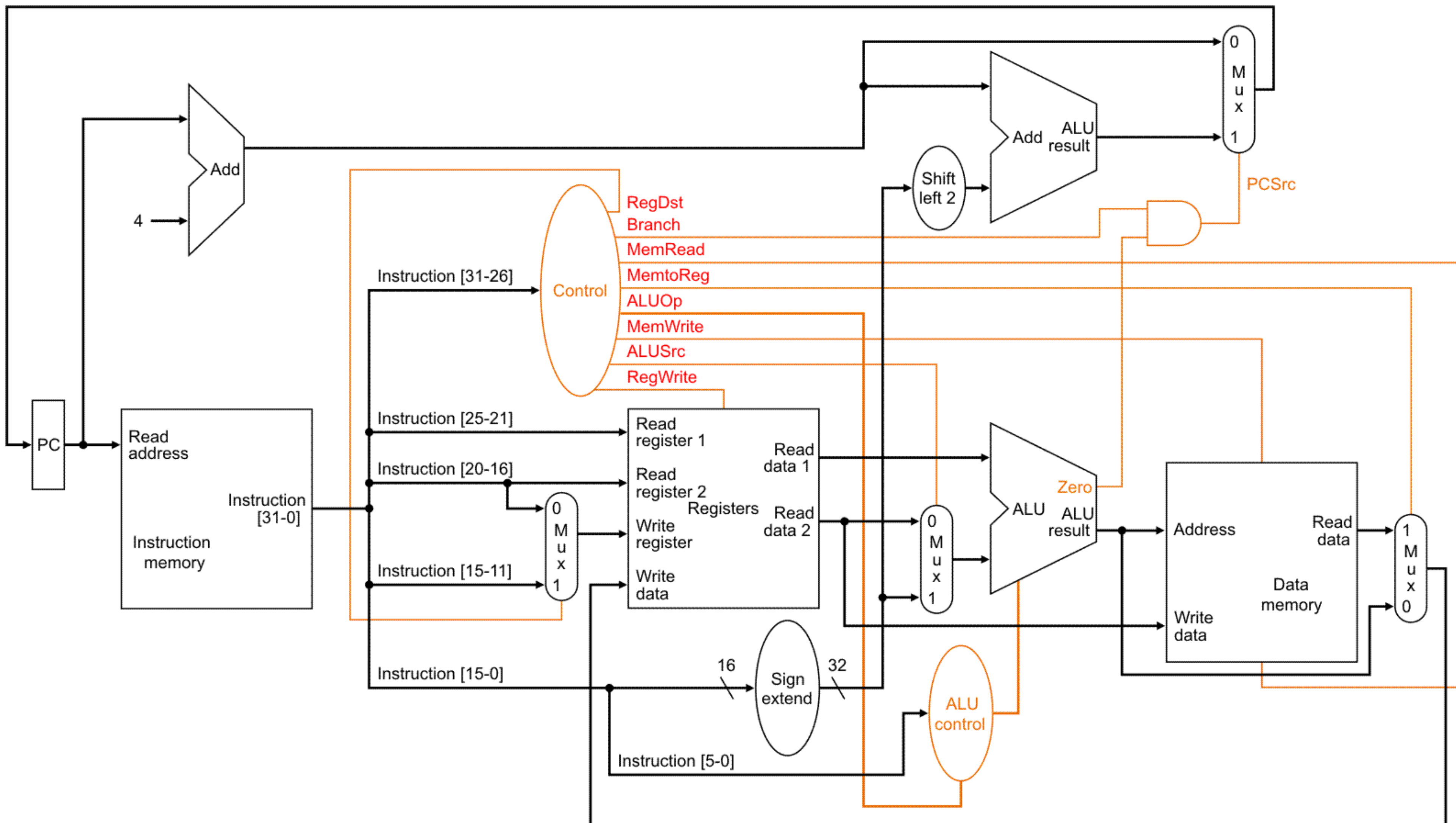
Write to Register - correction

- Multiplexer: RegDst
  - selects the correct register number to be modified: (R-type $\longrightarrow r_d$, Load $\longrightarrow r_t$)



Control Block

- Control: combinatorial logic generating all control signal

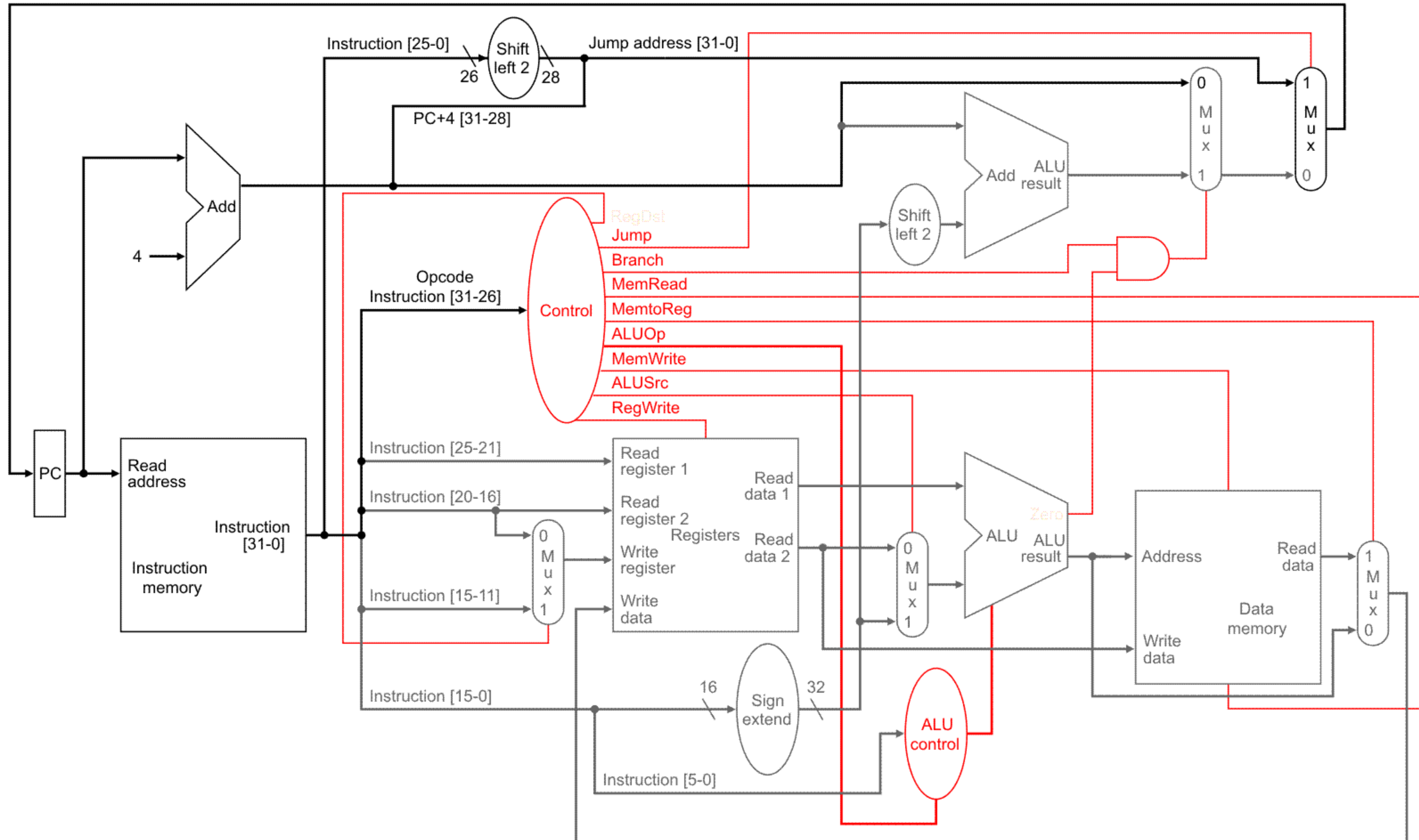


Absolute Unconditional Jump (JMP)

- New, arbitrary value loaded into PC
- No conditions checked
- Jump address multiplied by 4 – to extend jump range
- Missing 4 MSB bits complemented from current PC value (jump within a "memory segment")
- Multiplexer Jump:
  - selects the source of the next instruction address

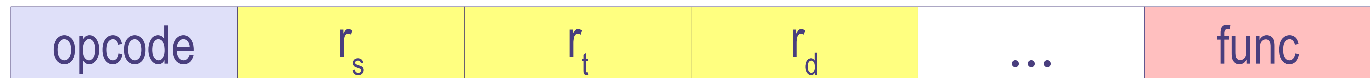
all assumptions made just for the purpose of this project, in order to keep the design simple and have all instructions of the same 32-bit size

Complete *Single-Cycle Architecture*

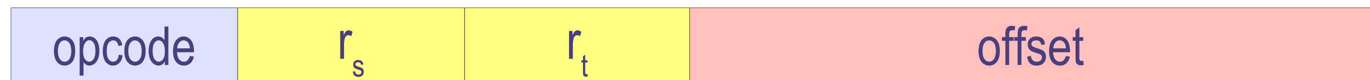


Implemented Instruction Set

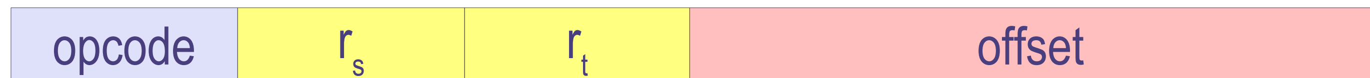
- Register (*R-type*)



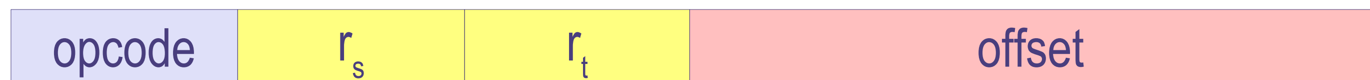
- Load



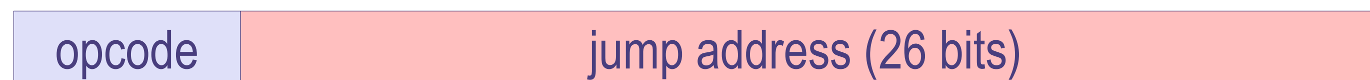
- Store



- BEQ



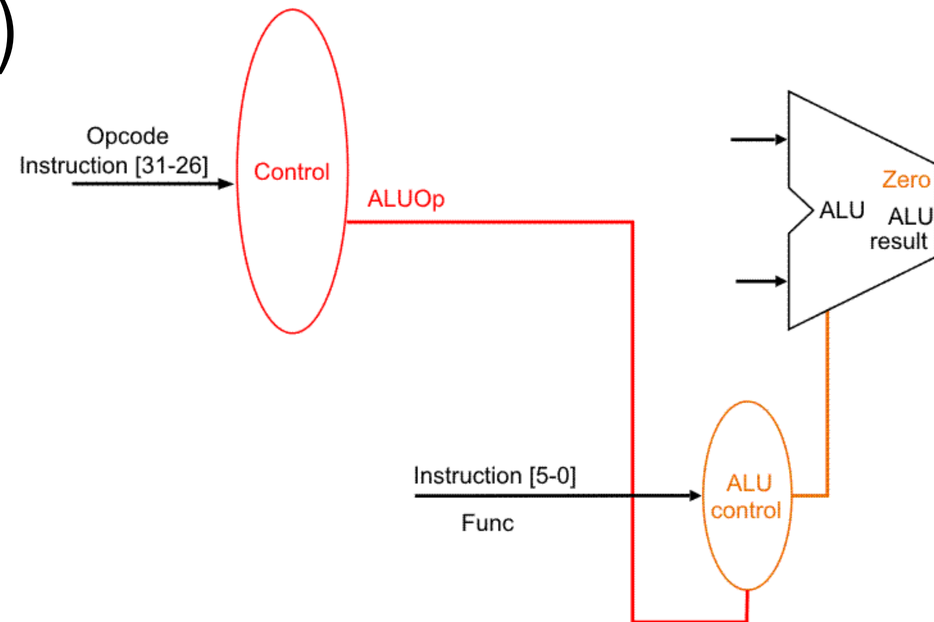
- JMP



DMCS

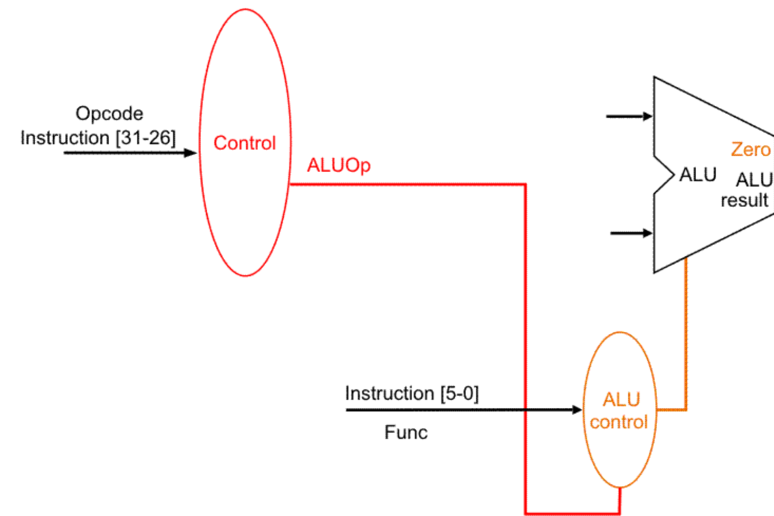
ALU Control

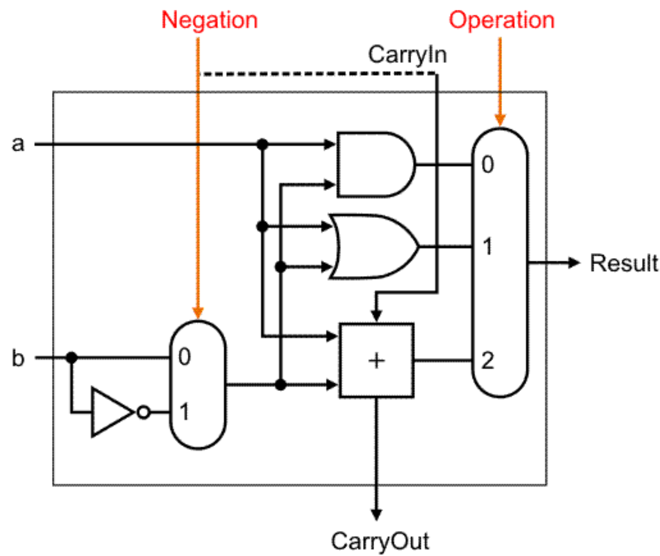
- All *R-type* instructions have identical *opcode* field, but differ in *Func* field (type of operation for ALU)
- Main control block receives only instruction opcode and generates signal *ALUop* (the same for all R-types)
- ALU Control Unit takes into account *Func* field (only for R-types) and provides direct control for ALU
- *ALUop* signal indicates instruction family, but not the actual operation

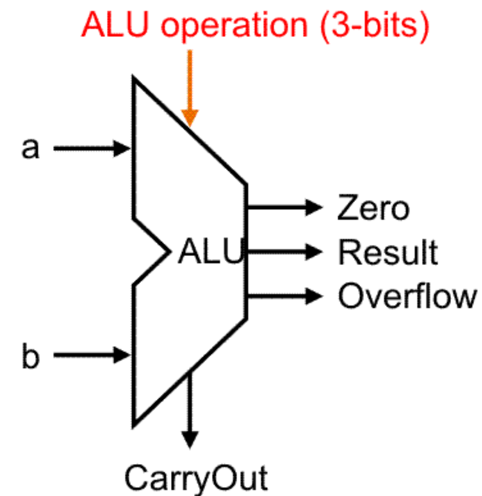


ALU Control

Func	ALU operation (Neg.+Oper)
AND	000
OR	001
ADD	010
SUB	110







ALU Control – Summary

- ALU Control is a combinatorial logic – operation can be described by truth table

Opcode	ALUOp	operation	Func field	ALU action	ALU input
LW	00	load word	xxxxxx	add	010
SW	00	store word	xxxxxx	add	010
BEQ	01	branch equal	xxxxxx	subtract	110
R-type	10	ADD	100000	add	010
R-type	10	SUB	100010	subtract	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
Jump	xx	x	xxxxxx	x	xxx