Performance of Single-Cycle Machine

CPI = 1.0, but what about cycle time? Unit reuse (I.e. adder for PC vs. ALU)

Arithmetic & Logic

PC	Instr. Memory	Reg Read	mux	ALU	mux	Reg Setup

Load

PC	Instr. Memory	Reg Read	mux	ALU	Data Memory	mux	Reg Setup

Store

PC	Instr. Memory	Reg Read	mux	ALU	Data Memory

Branch

PC	Instr. Memory	Reg Read	mux	ALU	Adder	mux	PC setup

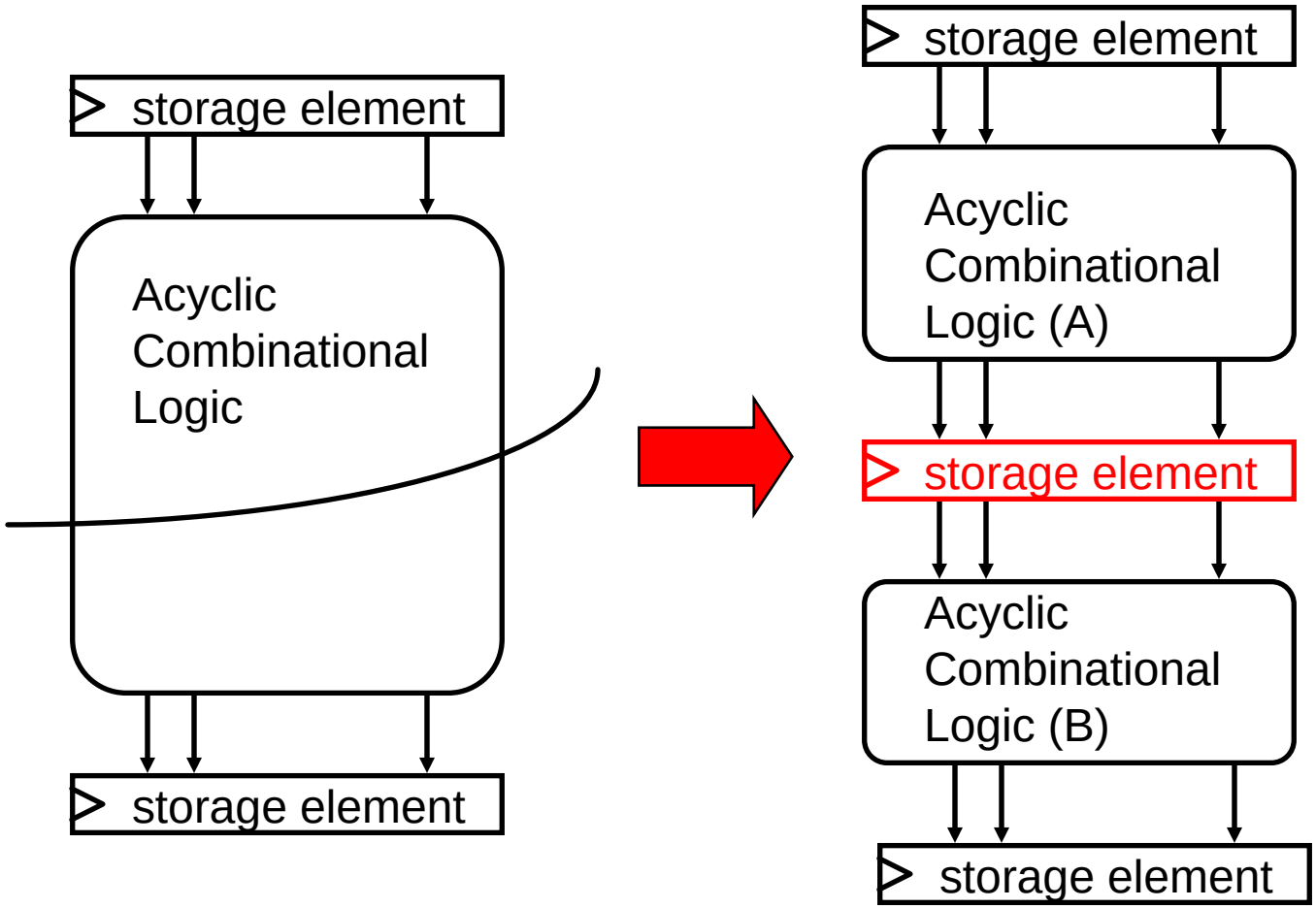
Jump

PC	Instr. Memory	mux	PC setup

Reducing Cycle Time

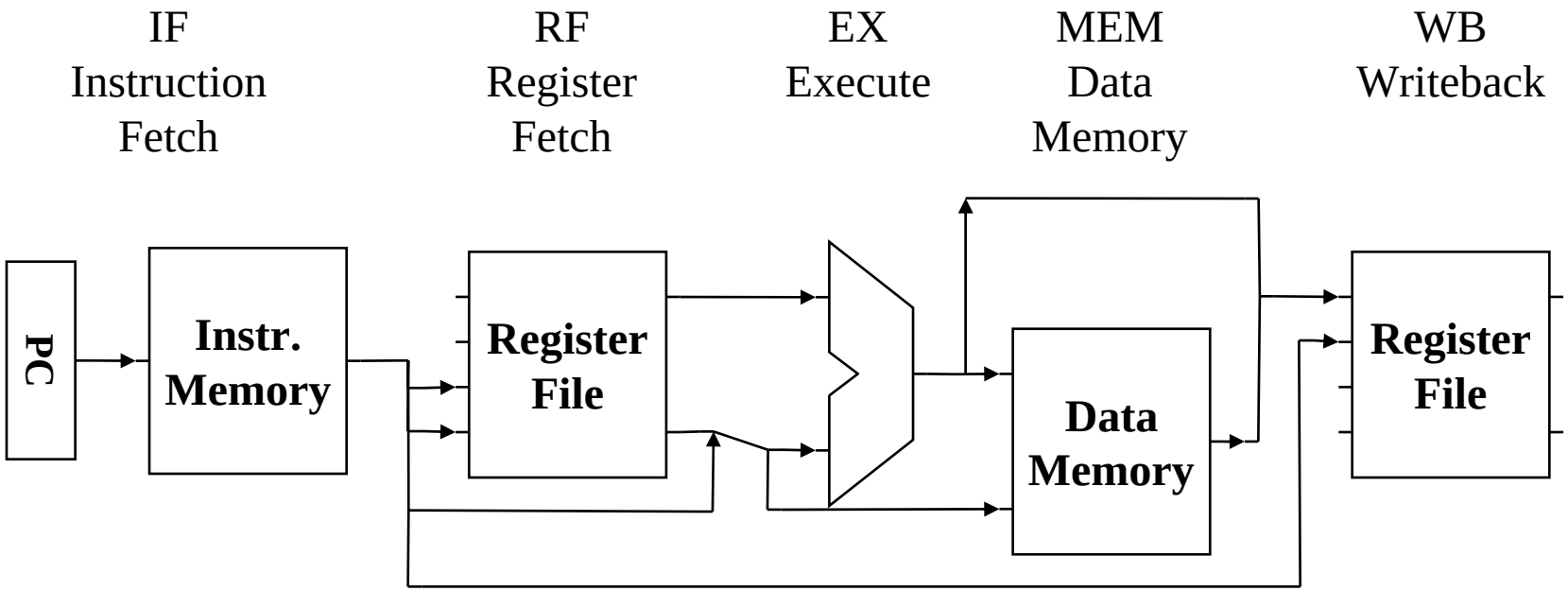
Cut combinational dependency graph and insert register / latch

Do same work in two fast cycles, rather than one slow one



Multicycle Processor Overview

Divide datapath into multiple cycles



Multicycle Processor Changes

Only one memory

- Shared between instructions and data

Only one ALU/adder

- Use ALU for instructions & PC computations

Add registers to datapath

- IR: instruction register
- MDR: Memory Data Register
- A & B: Values read from register file
- ALUout: Output of ALU

Cycle 1: Instruction Fetch

Put the instruction to execute into the Instruction Register (IR)

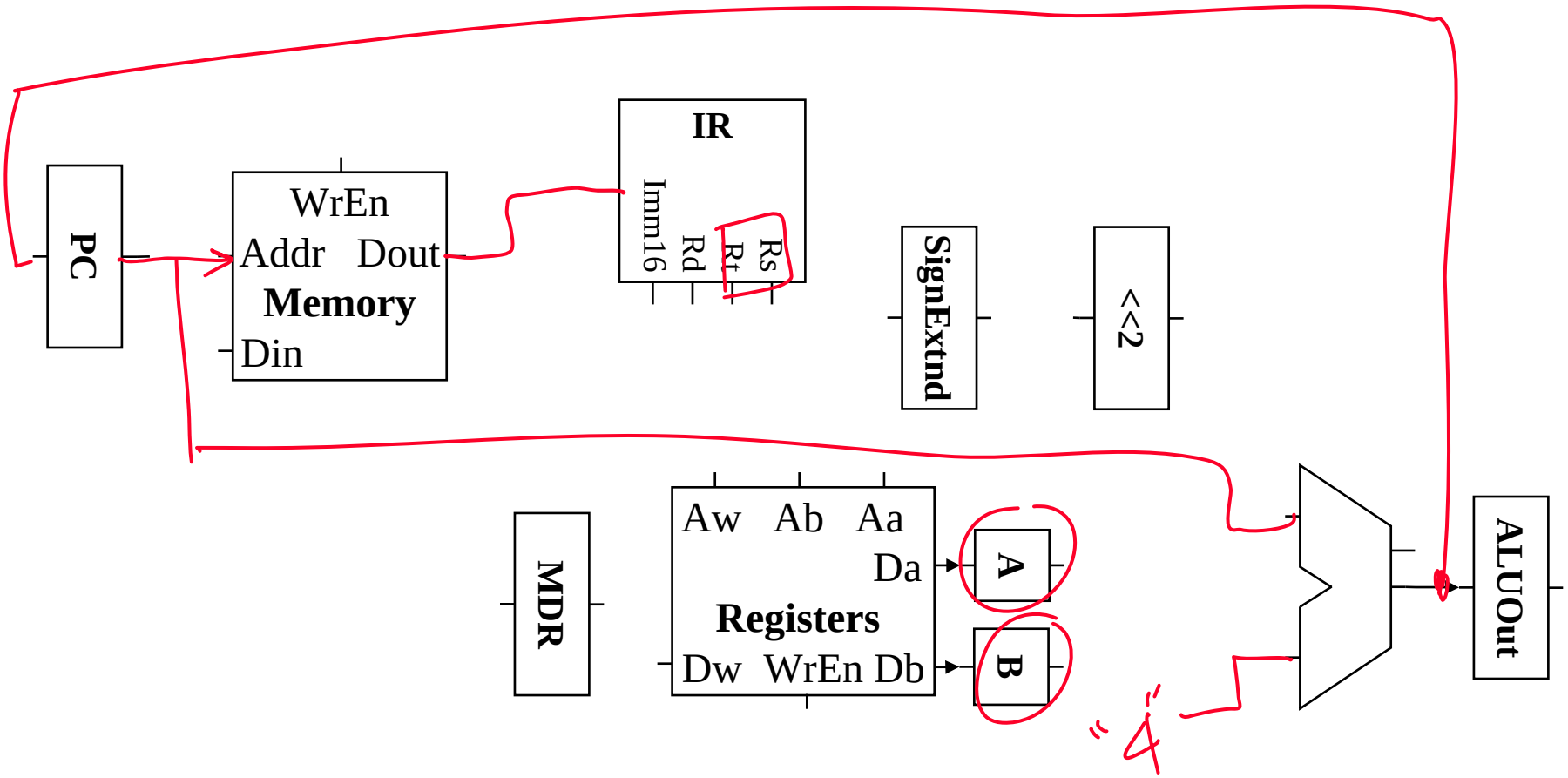
RTL: IR ← Mem [PC]

Set the PC to the next instruction (ignore branches)

RTL: PC = PC + 4

Cycle 1 Datapath

IR = Mem[PC]
PC = PC+4



Cycle 2: Instruction Decode, Register Fetch

Store the two GPR operands into registers A and B

RTL:

A = Reg [Rs]

B = Reg [Rt]

Compute Branch Target (in case it's a branch operation, won't have time later)

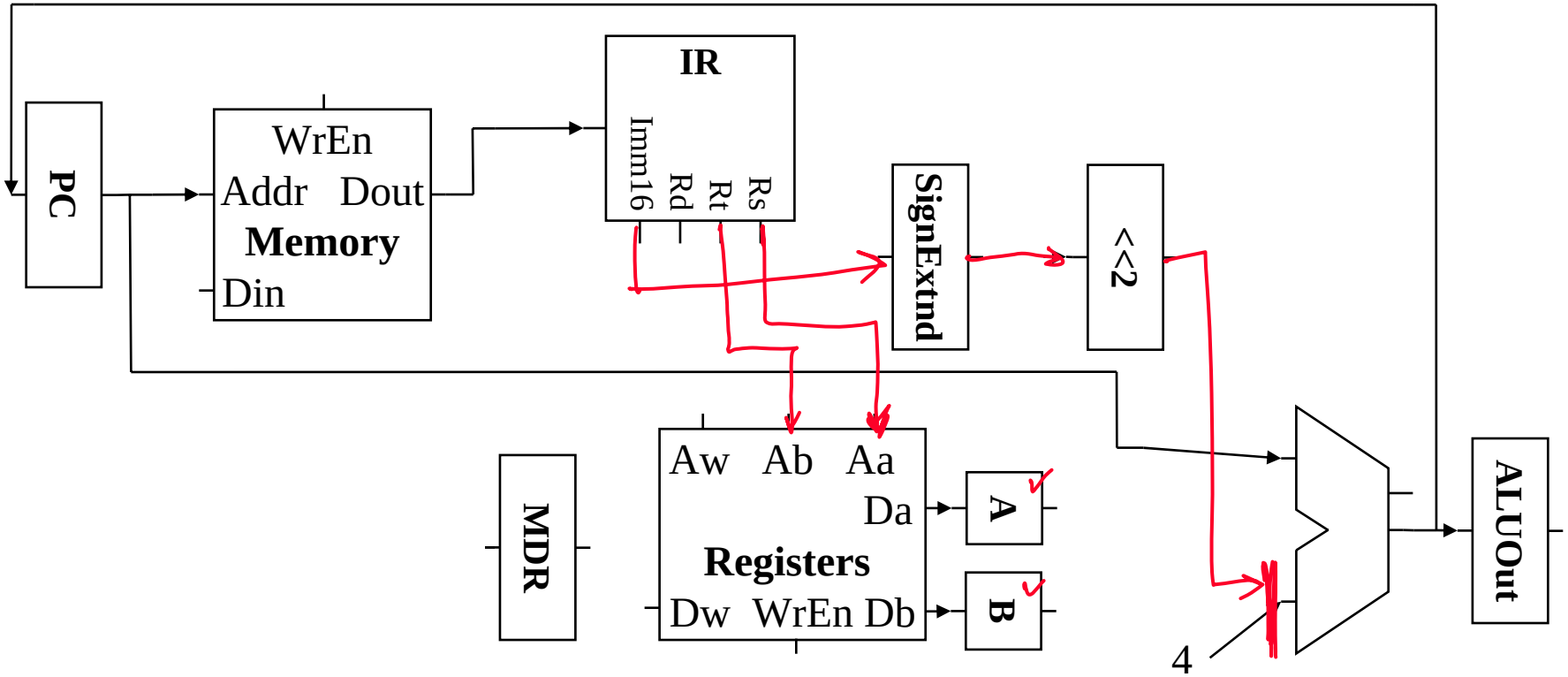
RTL:

ALUout = PC + SE (imm16) << 2

Cycle 1+2 Datapath

A = Reg [Rs]
B = Reg [Rt]

ALUOut = PC + SE(imm16) << 2



Cycle 3 (Branch)

Branch (Beq) – Branch address in ALUout. Set PC to branch address if A==B

RTL:

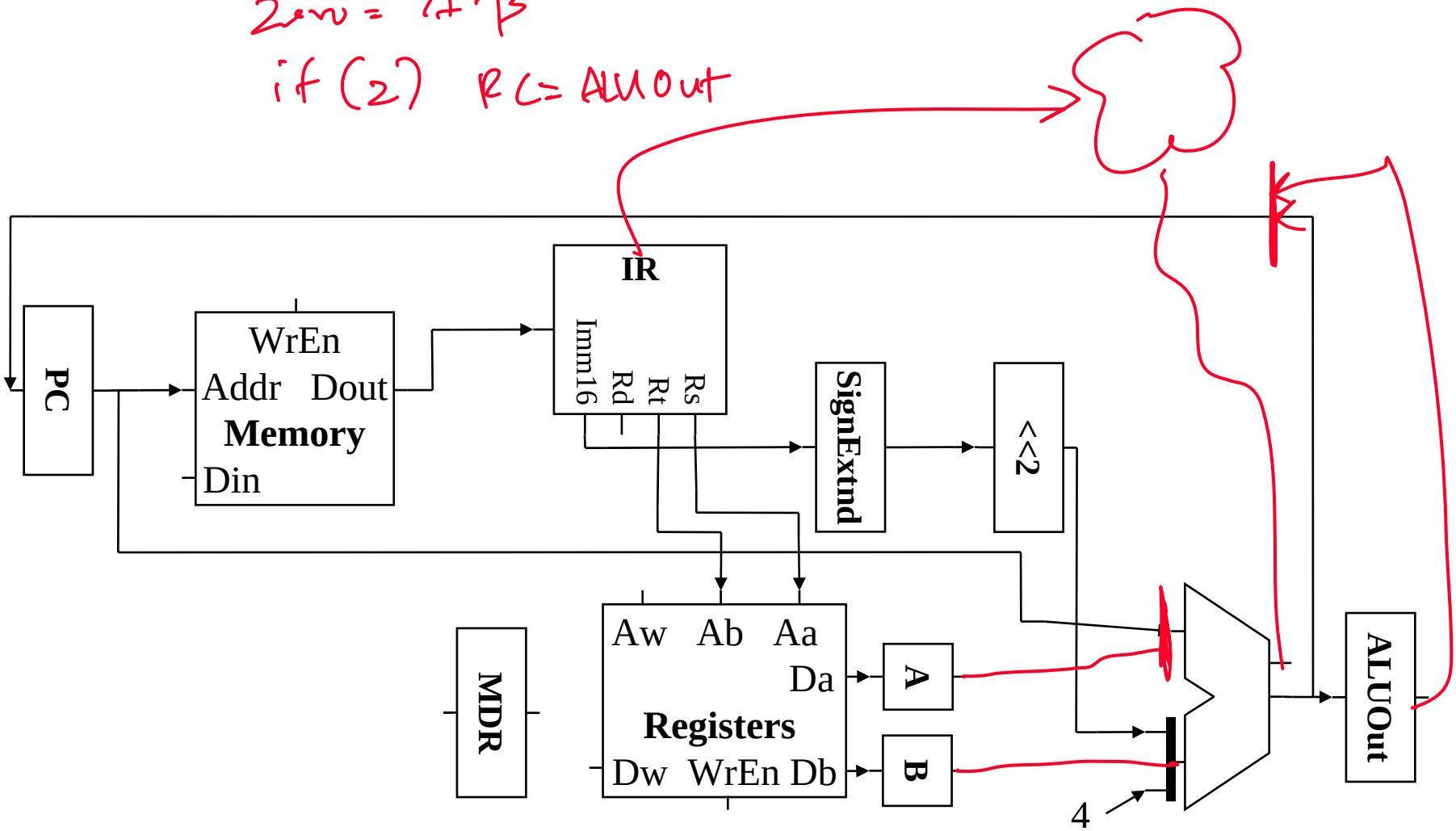
zero = A - B

if (zero) PC = ALUout

Branch Datapath

zero = A · B

if (z) PC = ALUOut



Cycle 3-4 (Add, Subtract)

Cycle 3: compute function in ALU, operands in A & B. Store in ALUout

RTL: ALUOut = A op B

Cycle 4: Write value from ALUout to destination register

RTL: Reg[Rd] = ALUout