

CSE140 - Processors



8-bit adder



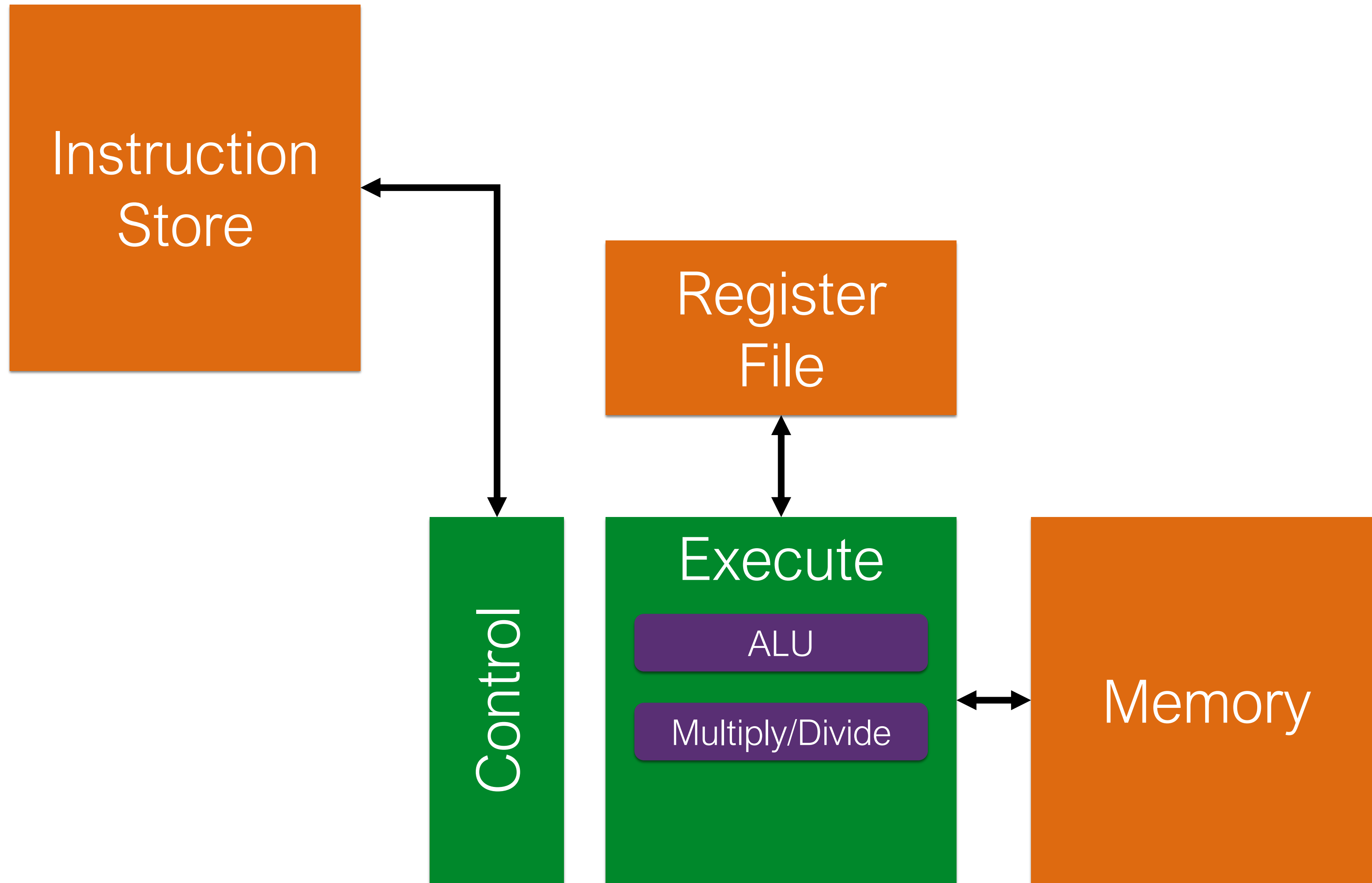
Edited file from Bryan Chin

<http://www.megaprocessor.com/>

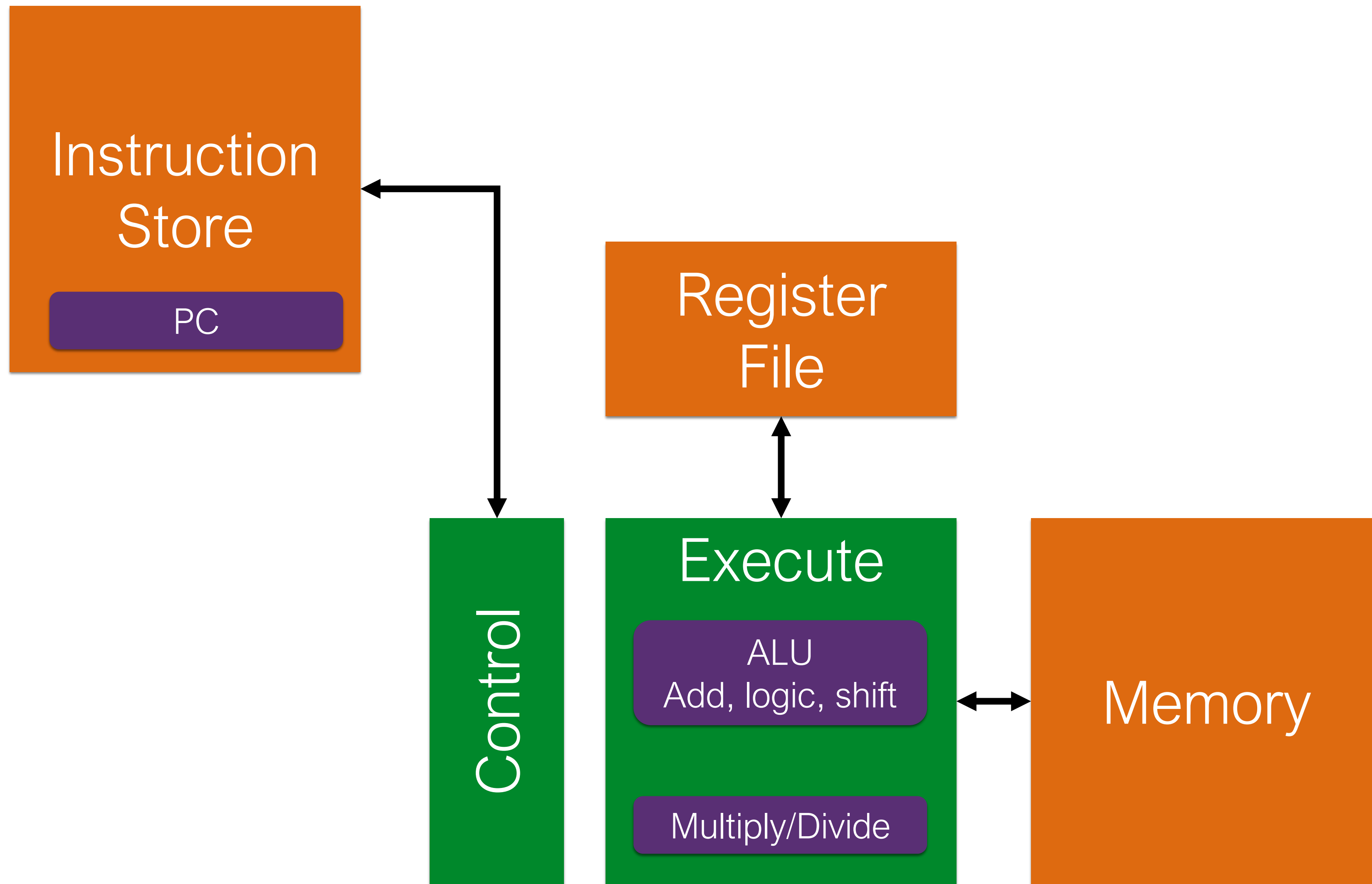
Navigation Check

- Simple (RISC) Processor

Simplified Digital Computer



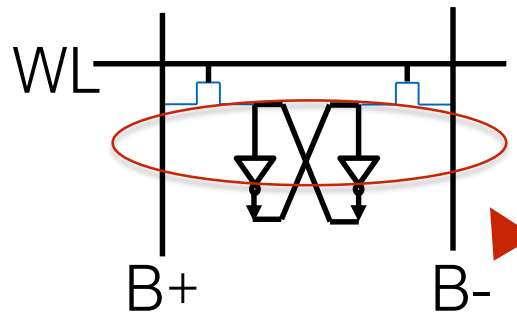
Simplified Digital Computer



Instruction Types

- Load
- Store
- Control Flow
- Execute

SRAM (single port)

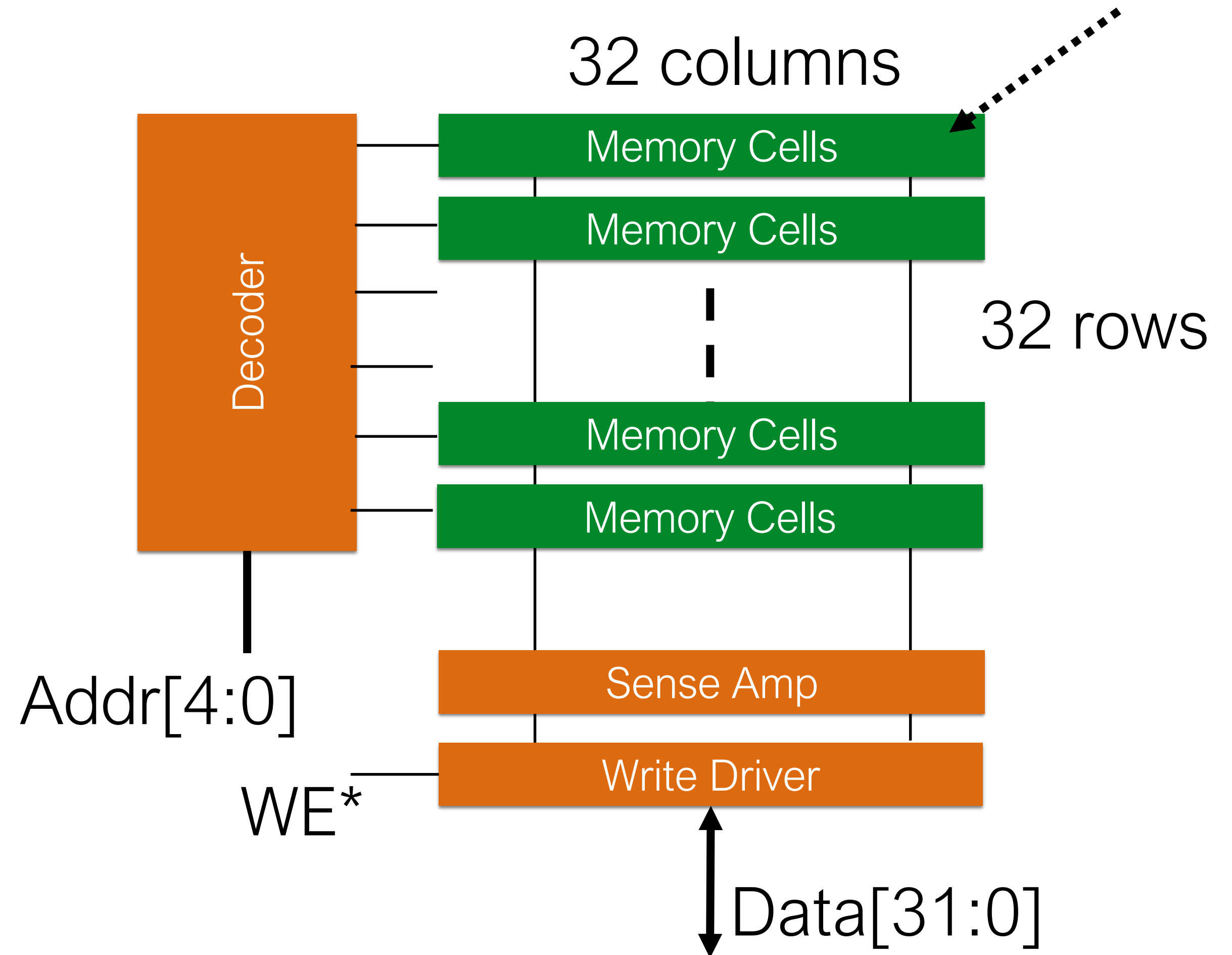


- 32 rows, 32 columns
- Can read or write one row at a time (32 bits wide)
- Decoder – takes in 5 bits, generates 1 of 32 (one-hot output)

Processor Register file

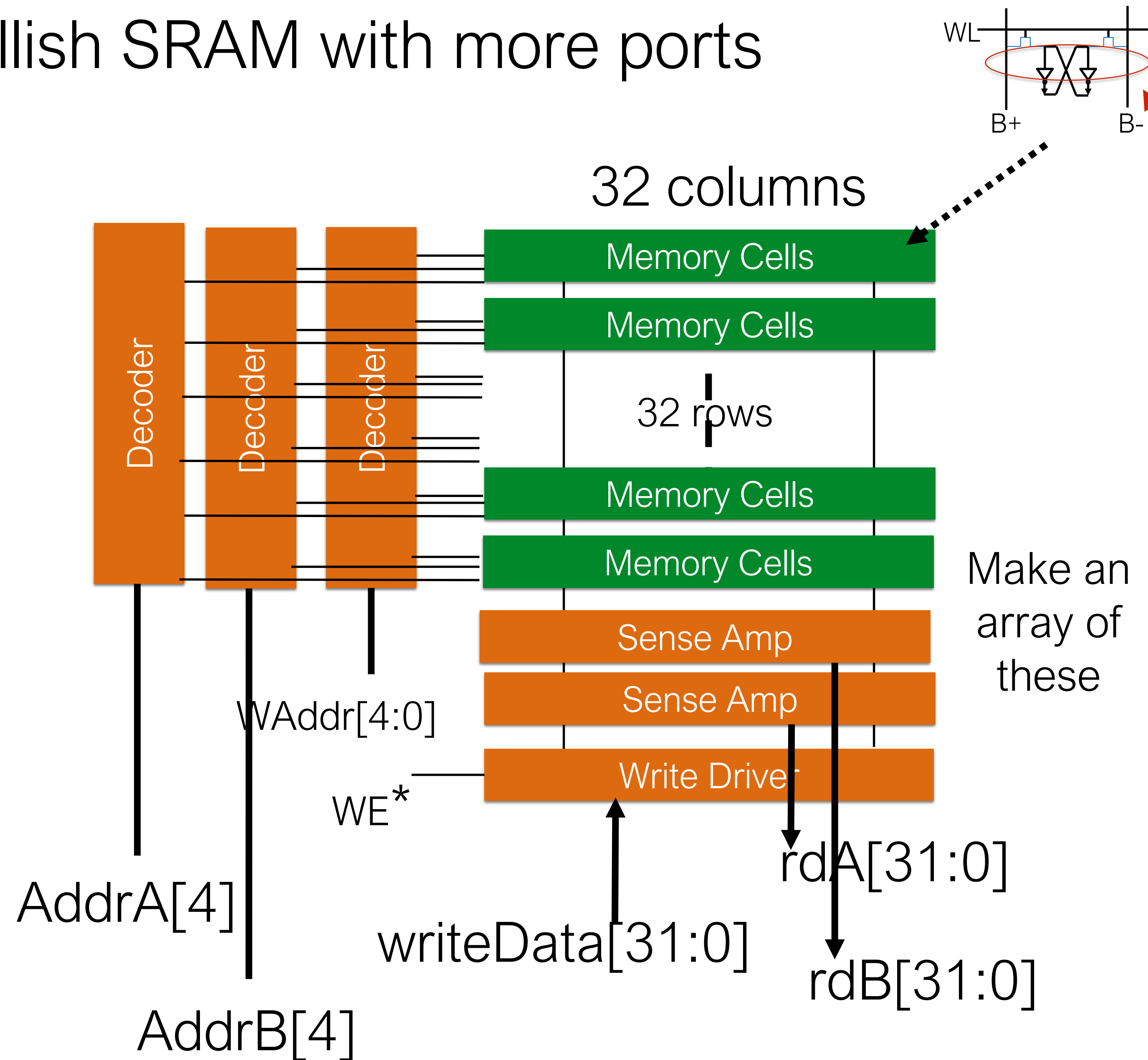
MIPS I - 32 registers, 32 bits

Need to read 2 operands and write 1
3 ports



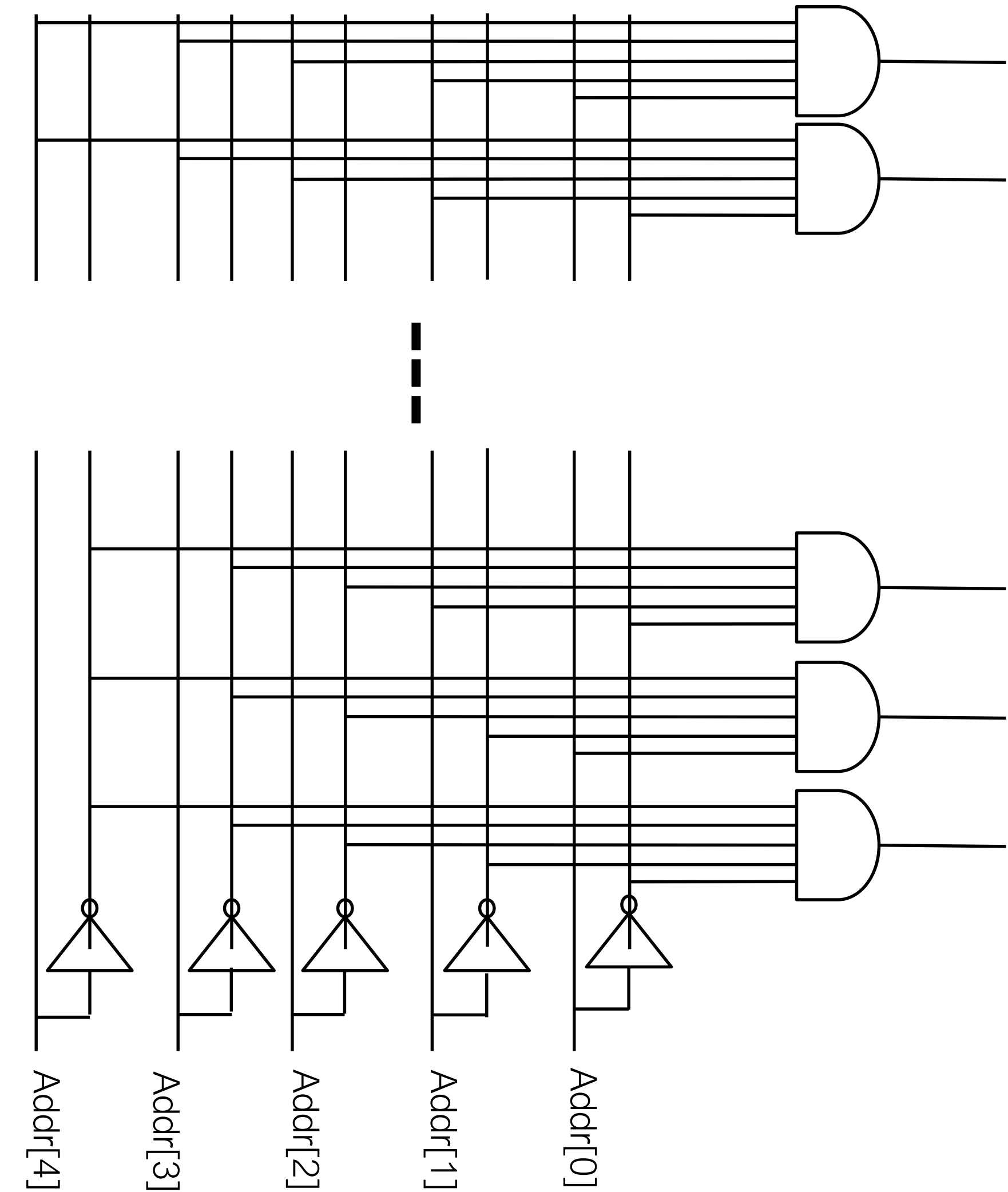
Register File – smallish SRAM with more ports

- Processor Register file
 - MIPS I - 32 registers, 32 bits
 - Need to read 2 operands and write 1
 - 3 ports
- Modern processor register files may have lots of ports (e.g 14 R and 7 W)



Row Decoder

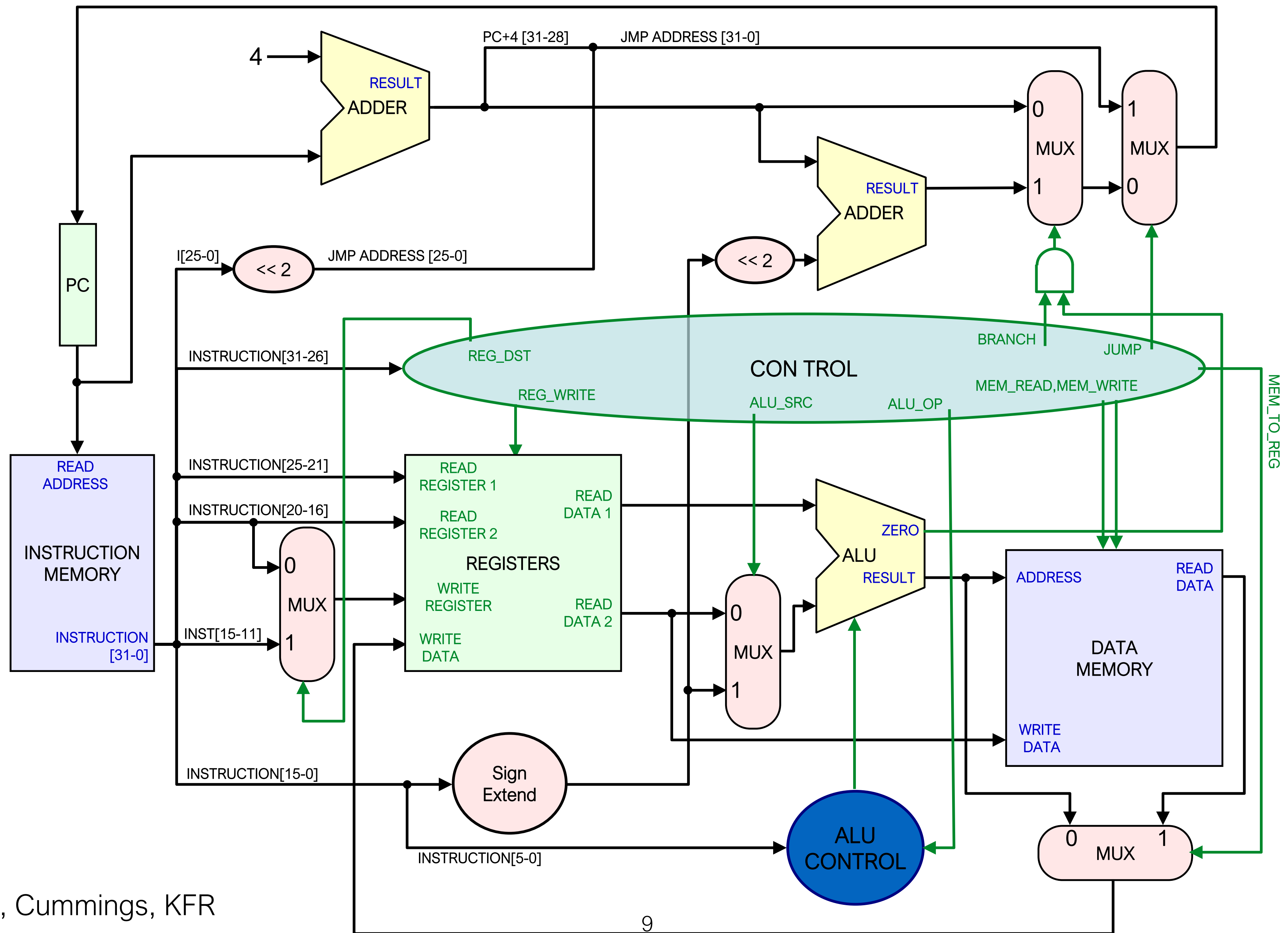
- 5- $\rightarrow$ 32 decoder choses which row
- One decoder per port



A register file is 128 bits wide and contains a total of 1 KB of storage.
How many address bits will be required by the row decoder?

- A. 10 bits
- B. 9 bits
- C. 64 bits
- D. 6 bits
- E. None of the above

* - B means byte, b means bit



Computer Program

- MIPS (example RISC) architecture
- Architecture
 - Instruction Set
 - Programming Resources (registers)
 - View of memory (how to read and write it)

Assembly

```
Label:    addi r8, r8, -4  
          sw   r0, 0(r8)  
          bne  r3, r0, Label
```

PC : Instruction

```
00400000: 2108fffc ;  
00400004: ac000000 ;  
00400008: 1400fffd ;
```

Simple Single Cycle CPU

- Assume each instruction executes in a single cycle
- Instruction Format – MIPS RISC Instructions.
 - Fixed instruction size (all instructions are 4 bytes (32 bits)).
 - Load/Store, Compute and Control

R-Type

Opcode	Rs	Rt	Rd	Shift (shamt)	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

e.g. add, sub, sll, ...
`add r6, r2, r3`

I-Type

Opcode	Rs	Rt	Immediate
6 bits	5 bits	5 bits	12 bits

e.g. ori, andi, bne, ld, st, ...
`ld r6, 0x100(r7)`
`bne r3, r0, target`

J Type

Opcode	Rs	Rt	Rd	Sa	Funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

e.g. jr, jalr
`jr target`

CPU Components – Single Cycle Execution

Assumptions:

- Every machine language instruction happens in 1 Clock Cycle
- MIPS architecture
 - *Microprocessor without interlocked pipeline stages*
 - reg-reg architecture: all operands must be in registers (total 32)
 - 3 Instruction Formats; each instruction 32 bits long
 1. R-type: all data in registers (most arithmetic and logical)
e.g. **add \$s1, \$s2, \$s3**
 2. I-type: branches, memory transfers, constants
e.g. **beq \$s1, \$s2, Label; lw \$s1, 32(\$s2)**
 3. J-type: jumps and calls
e.g. **j Label;**

add \$s0, \$s1, \$s2

0000000	10001	10010	10000	000000	1000000
---------	-------	-------	-------	--------	---------

Src: TSR, Cummings, KFR

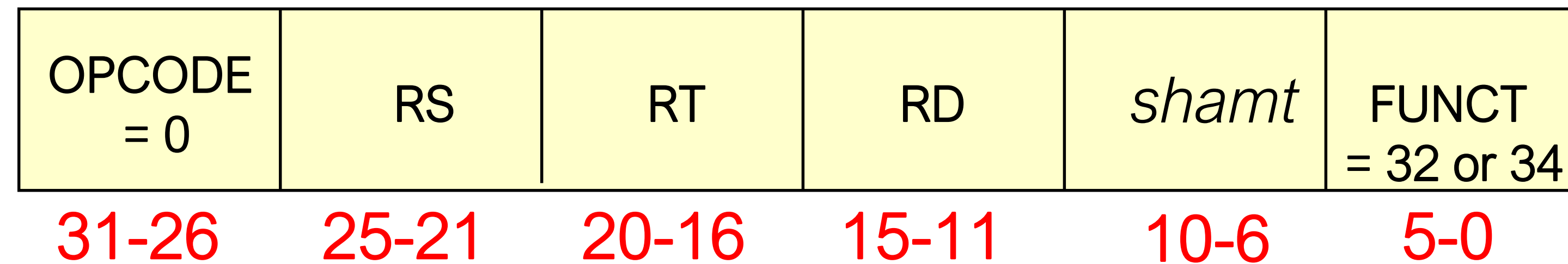
0	17	18	16	0	32
---	----	----	----	---	----

R-type Instruction: reg-reg ALU ops (e.g. add, and)

Tells operation to be performed

Tells specific variant of operation (e.g. add/sub have same opcode)

R-type Instruction



ADD r5, r7, r2

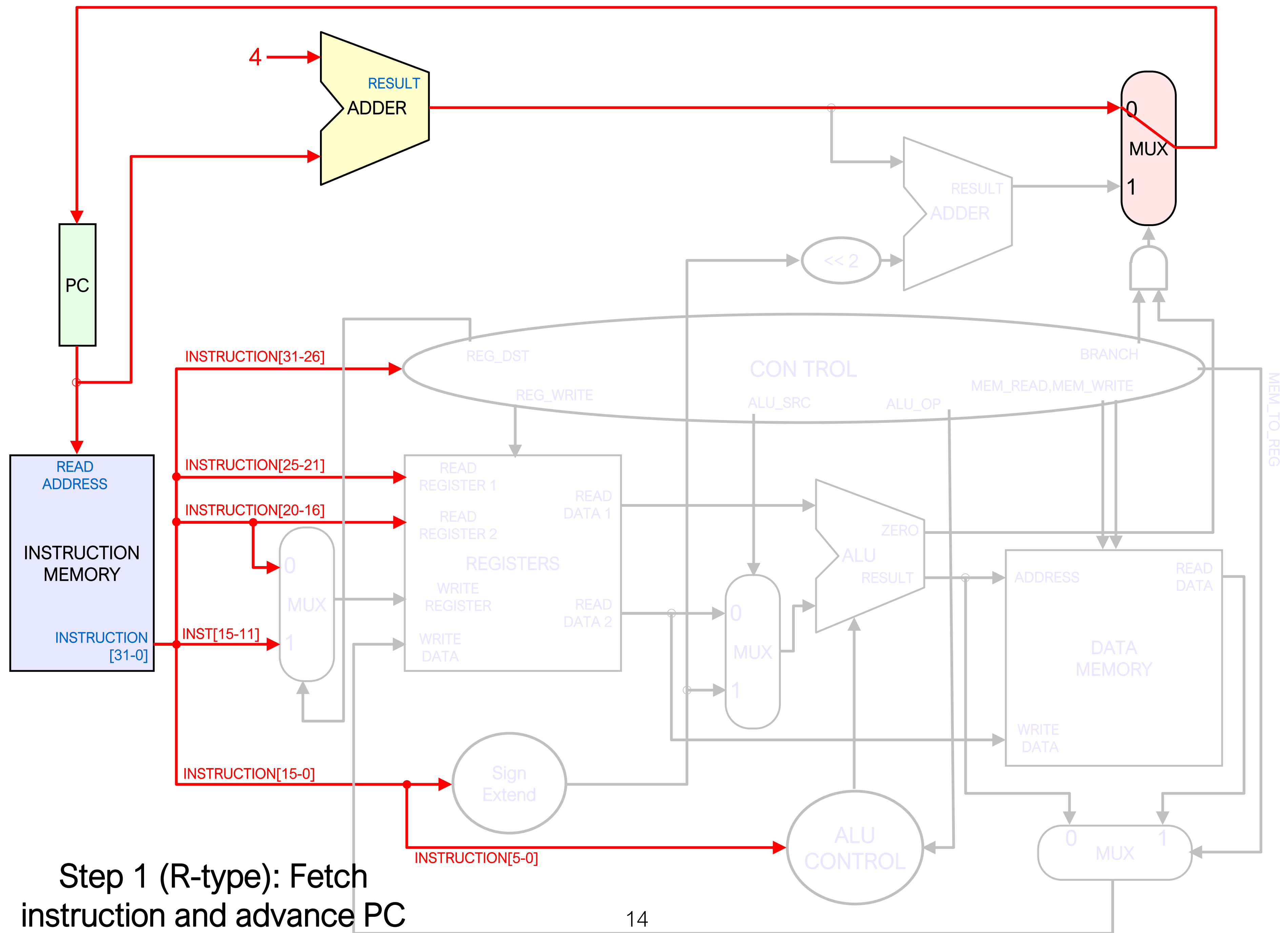
ADD RD, RS, RT

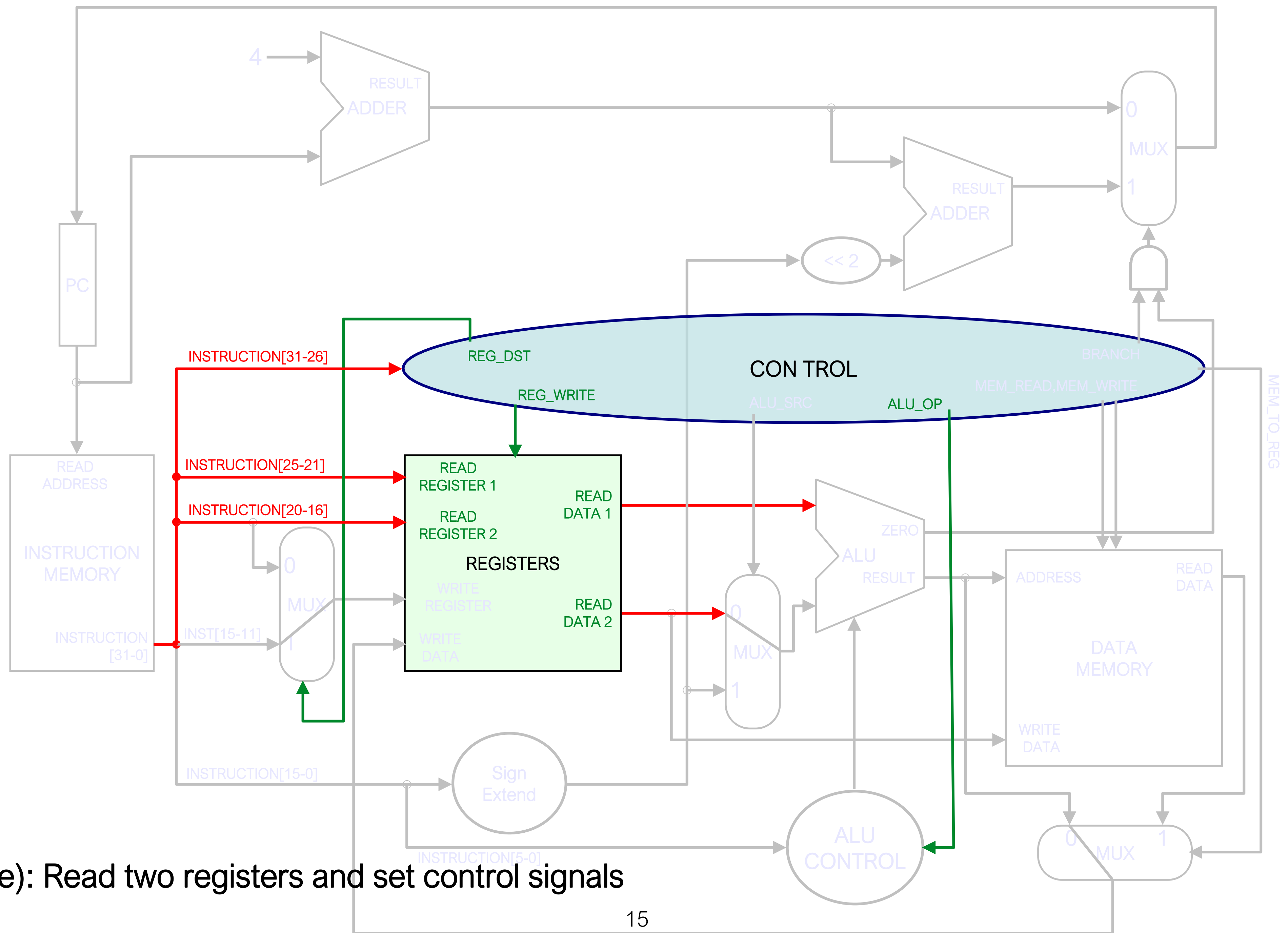
Shift amount (for sll, srl etc.)

Source Register 1
(attached to "Read Register 1" input)

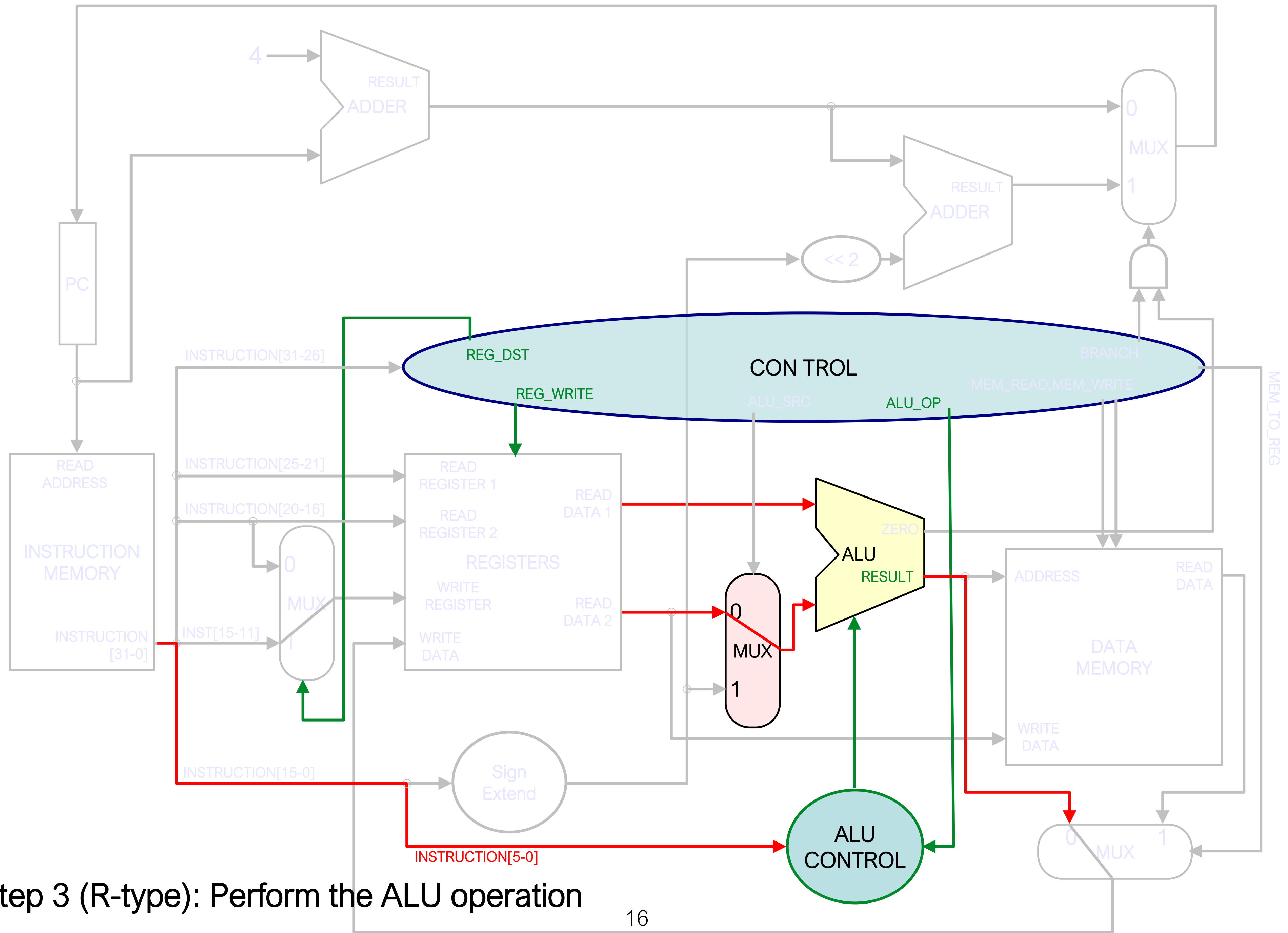
Source Register 2
(attached to "Read Register 2" input)

Destination Register
(attached to "Write Register" input)

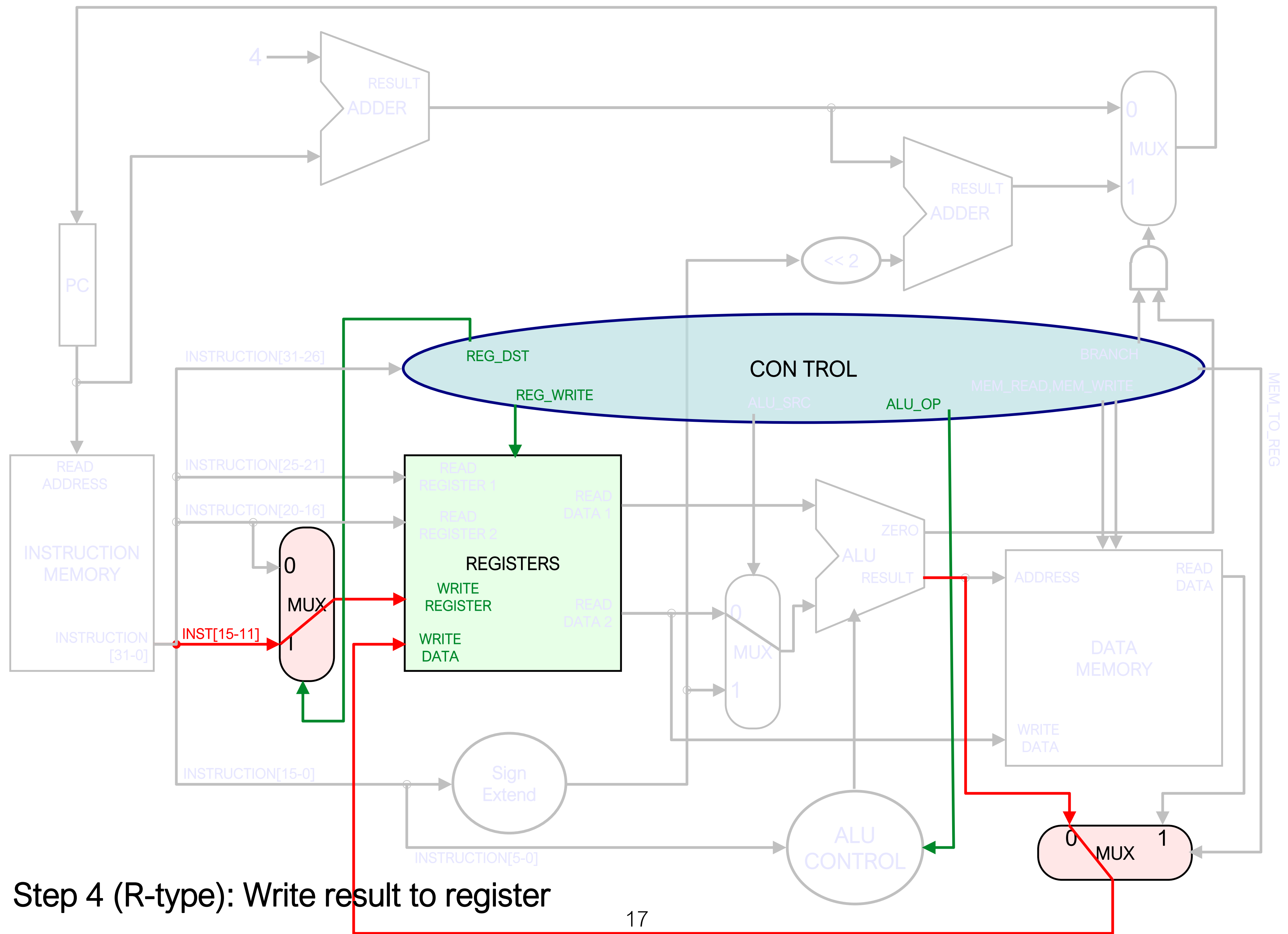




Step 2 (R-type): Read two registers and set control signals



Step 3 (R-type): Perform the ALU operation

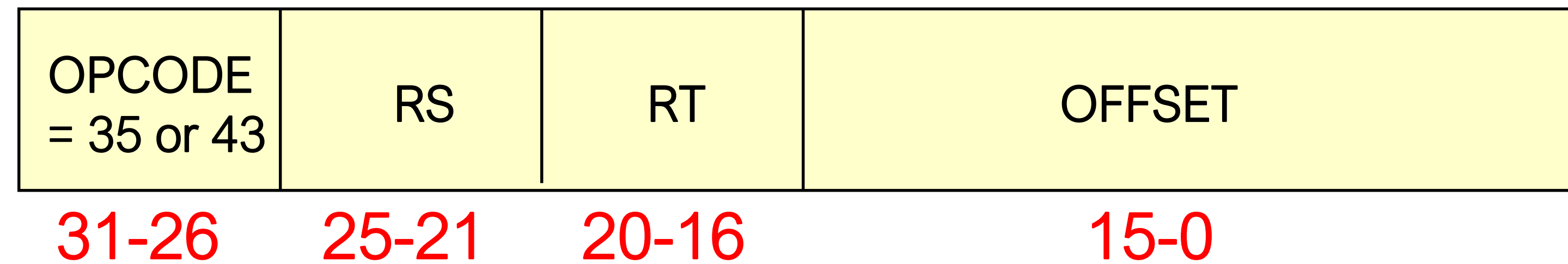


Step 4 (R-type): Write result to register

I-Type: Store Instruction

Tells operation to
be performed

Store
Instruction



Base Address Register
(attached to “Read
Register 1” input)

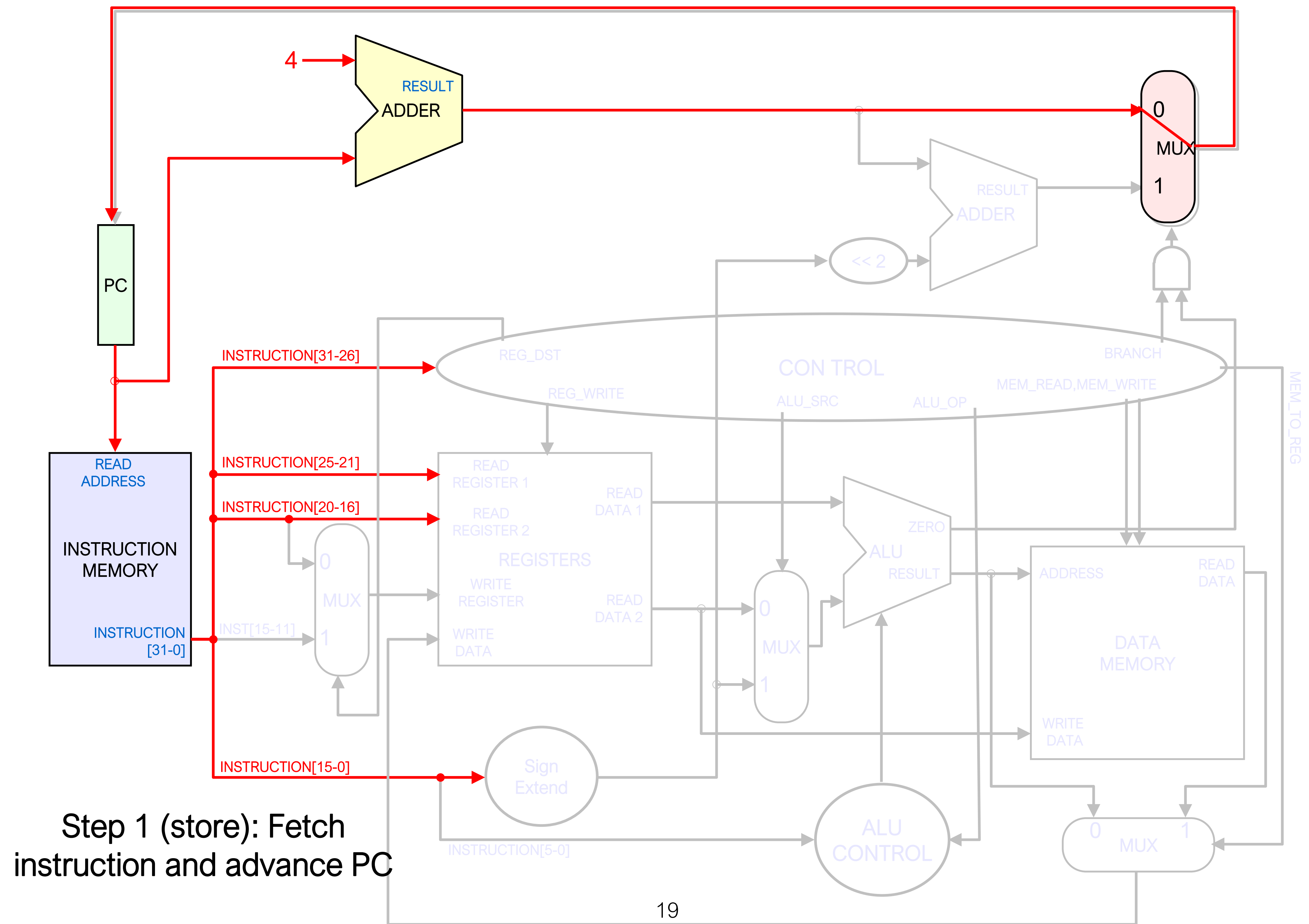
Source register
whose value will be
stored to memory
(attached to “Read
Register 2” input)

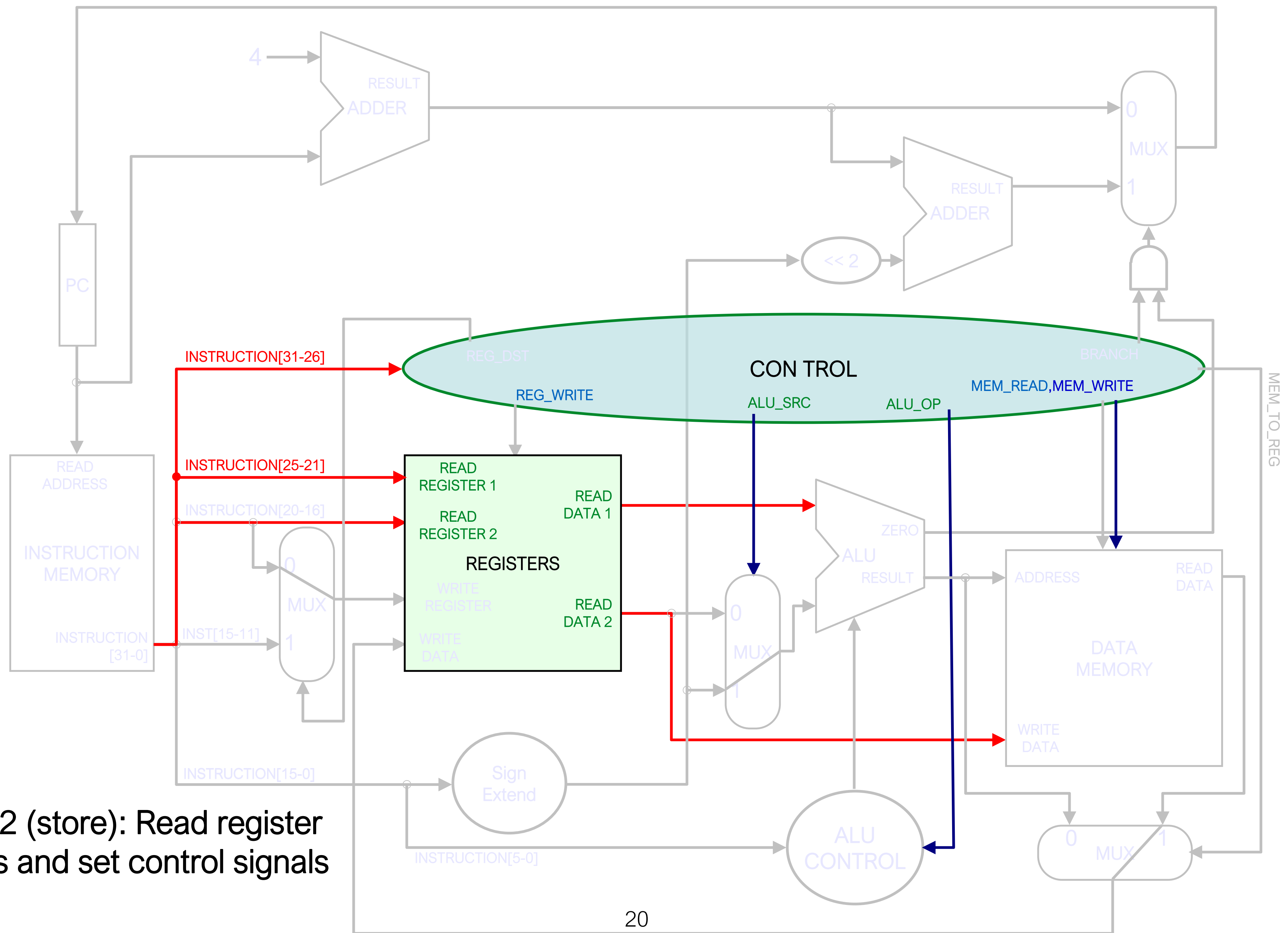
Constant offset
(added to the base
address in RS)

SW r0, 32(r5)

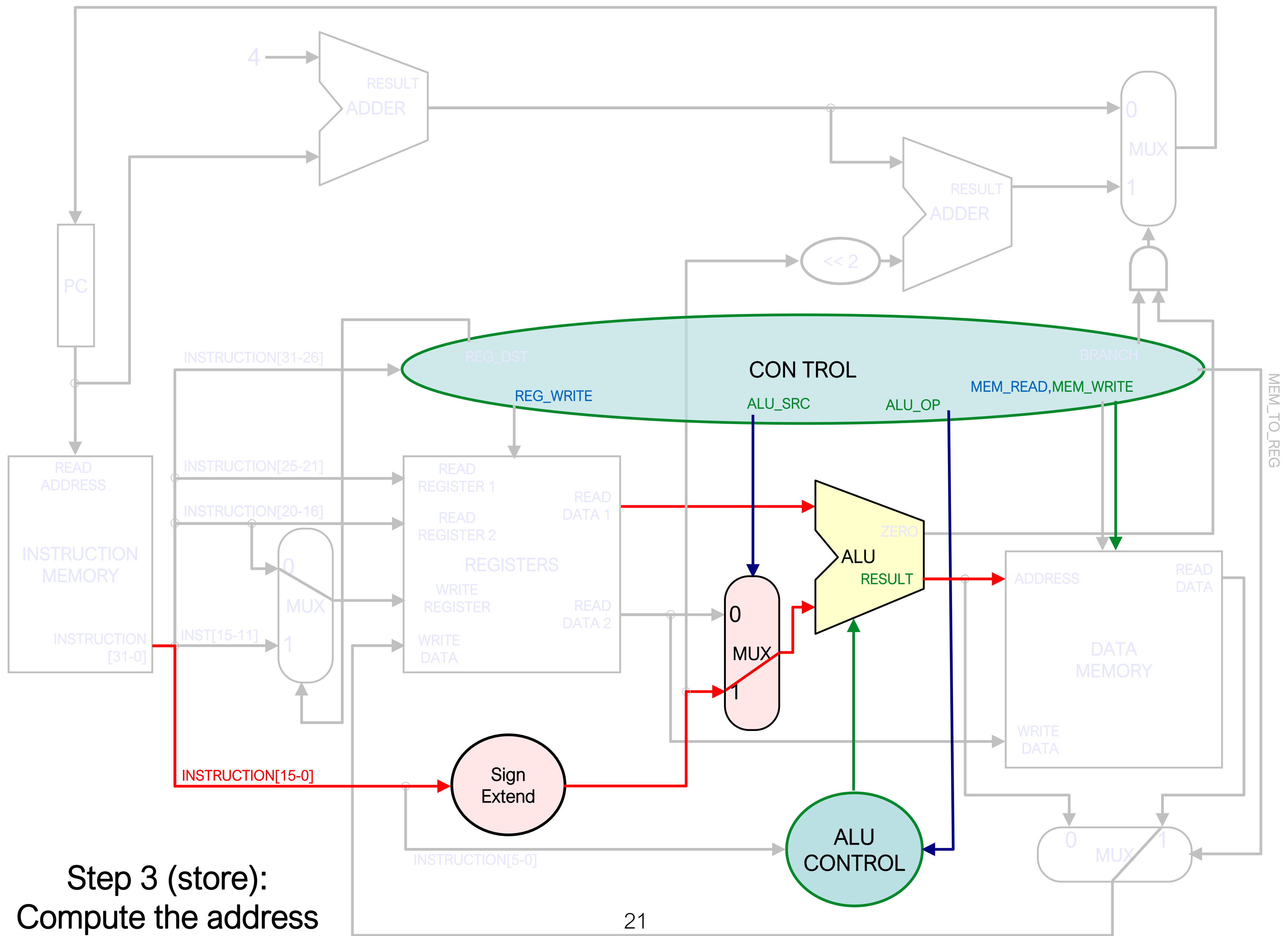
SW RT, #(RS)

*Note: same as x86
MOV [ebx+32], eax*

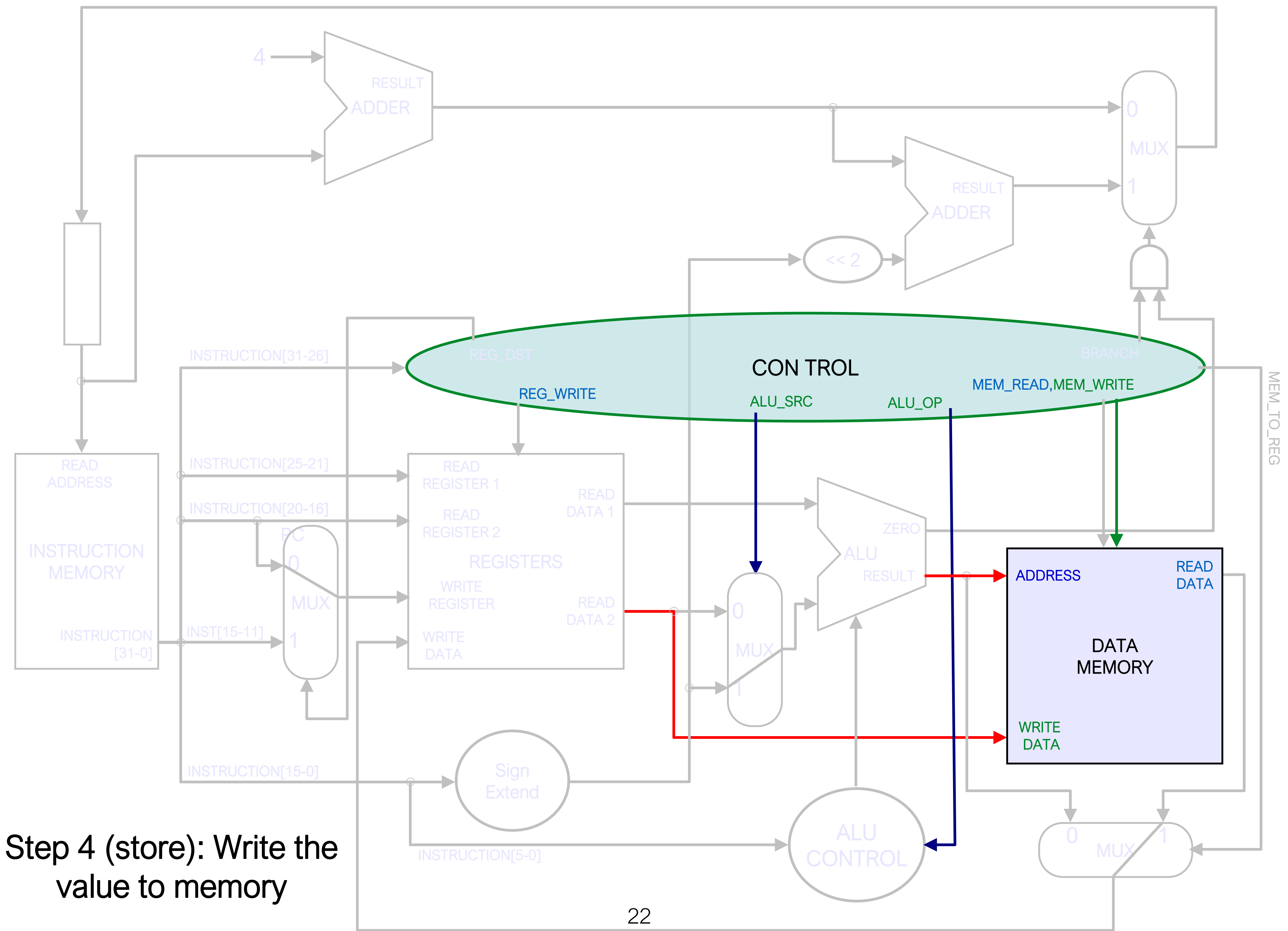




Step 2 (store): Read register values and set control signals

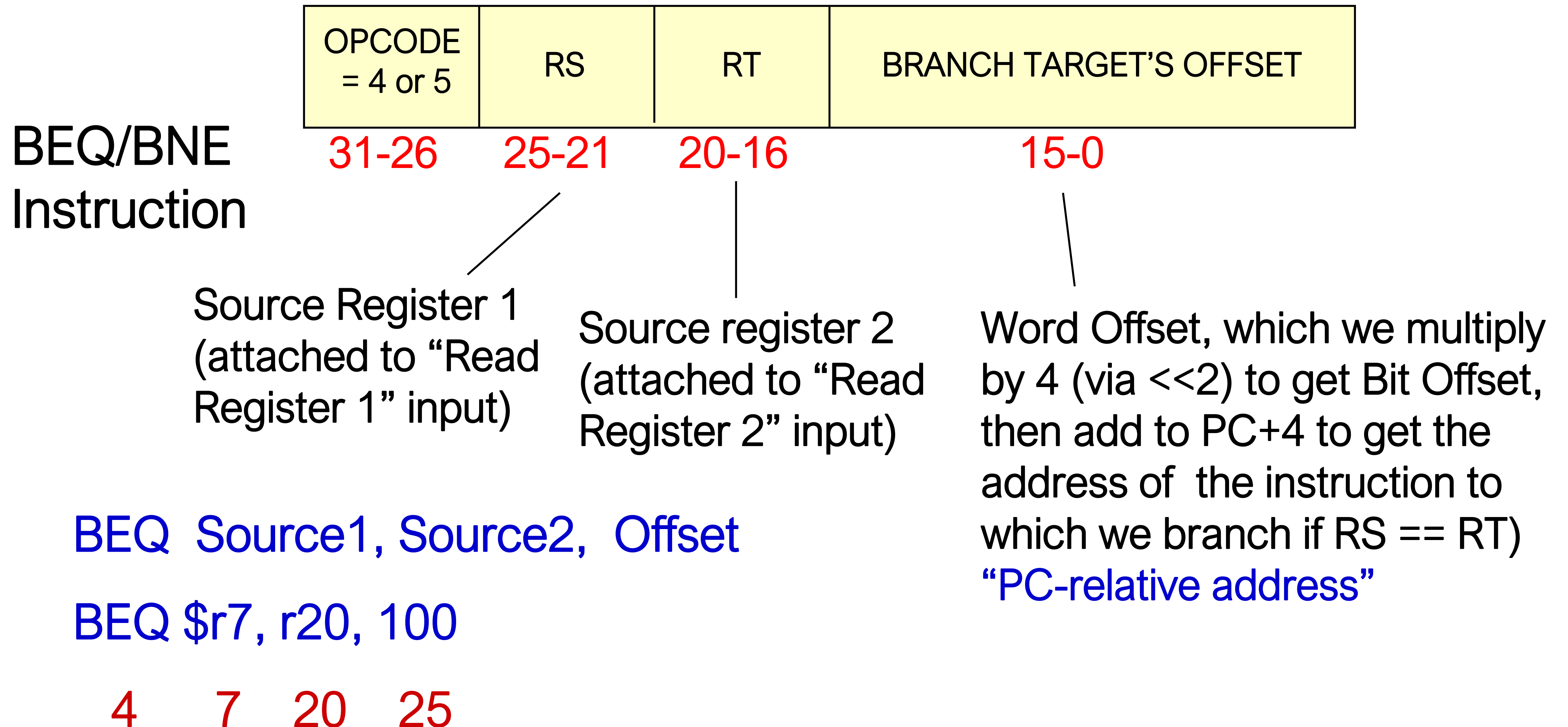


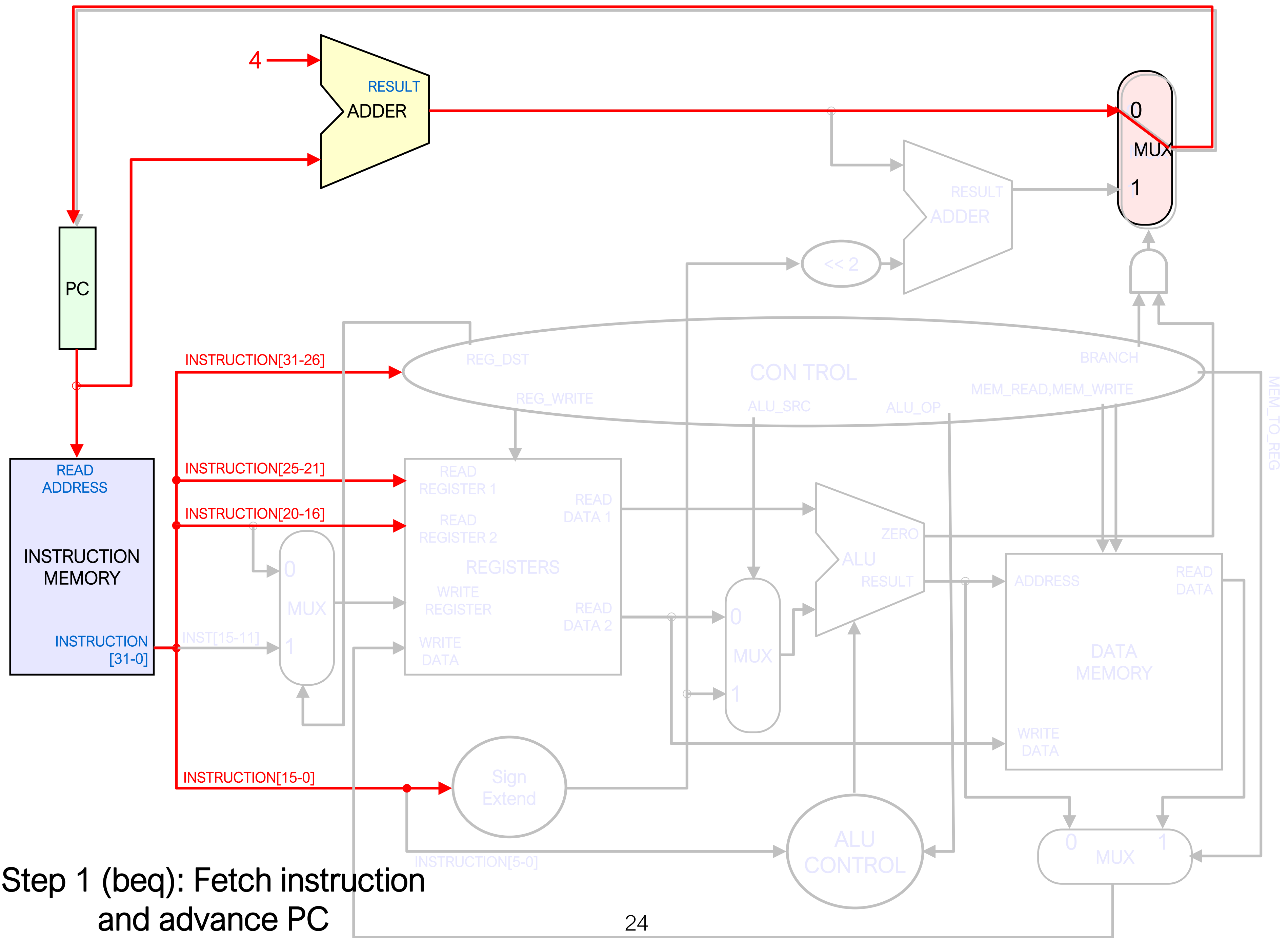
Step 3 (store):
Compute the address



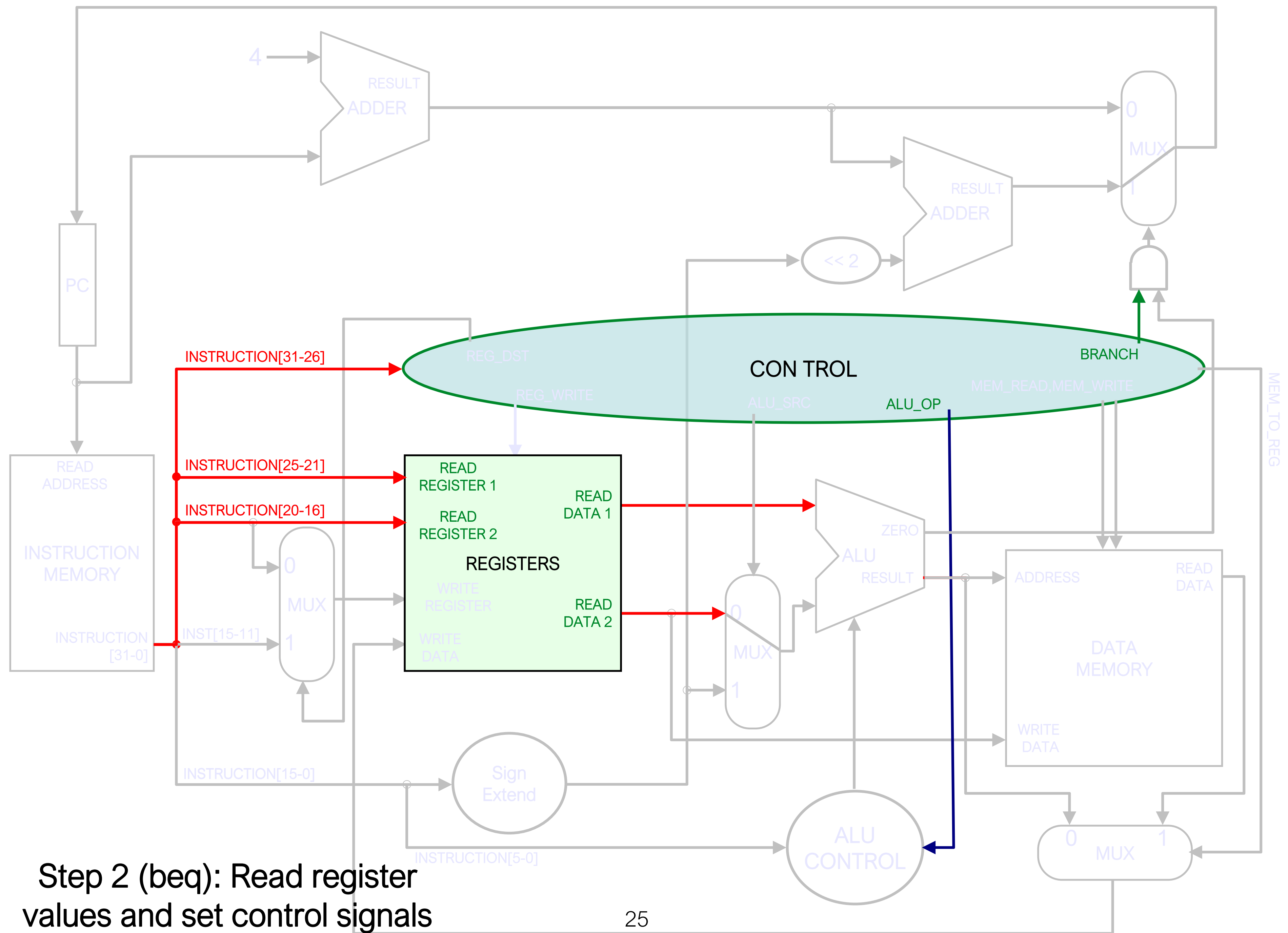
Step 4 (store): Write the value to memory

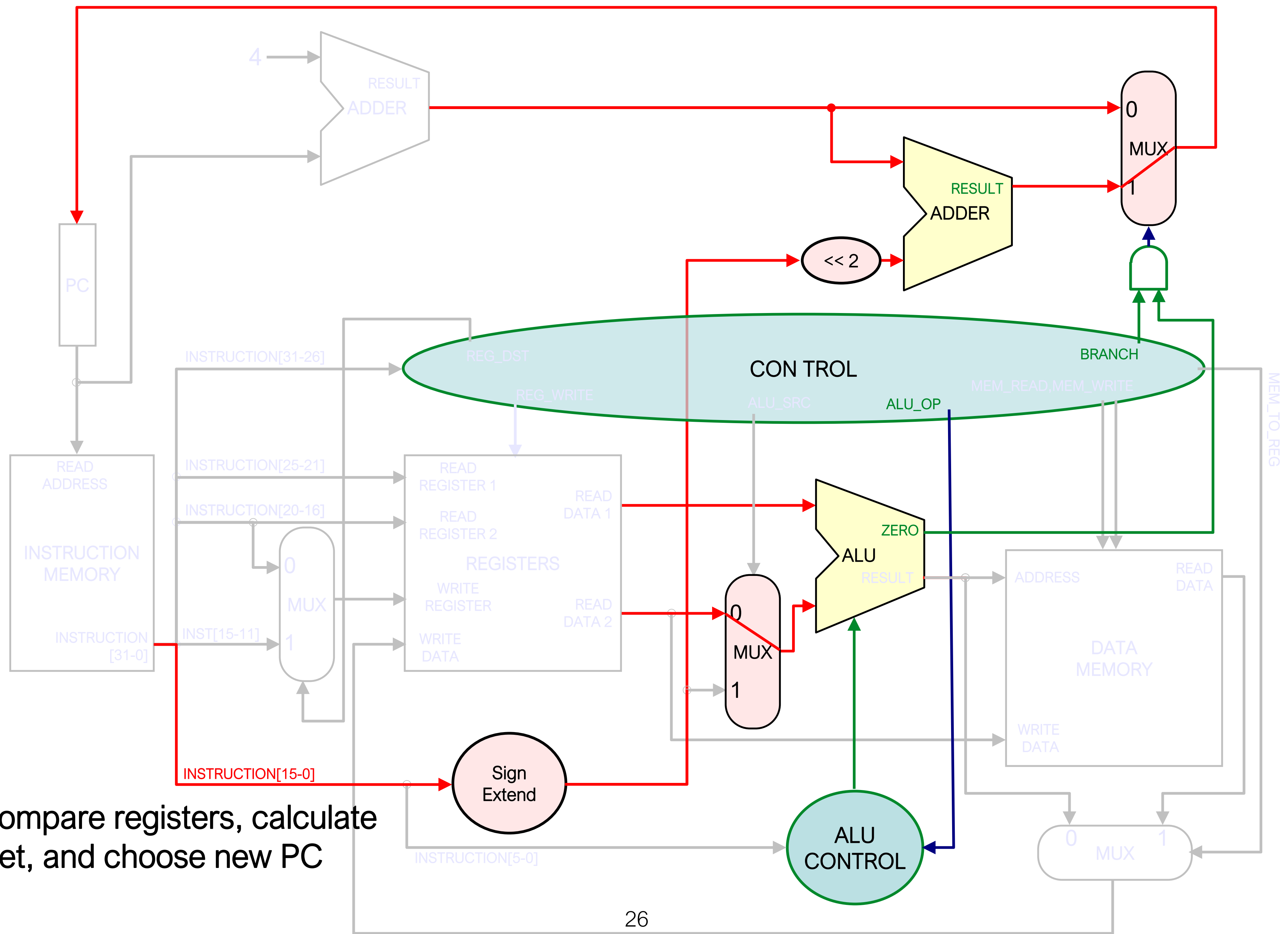
I-Type: Conditional Branch





Step 1 (beq): Fetch instruction
and advance PC





Step 3 (beq): Compare registers, calculate branch target, and choose new PC

J-Type: Unconditional Branch

JMP/JAL
Instruction

OPCODE = 2 or 3	BRANCH TARGET ADDRESS
--------------------	-----------------------

31-26

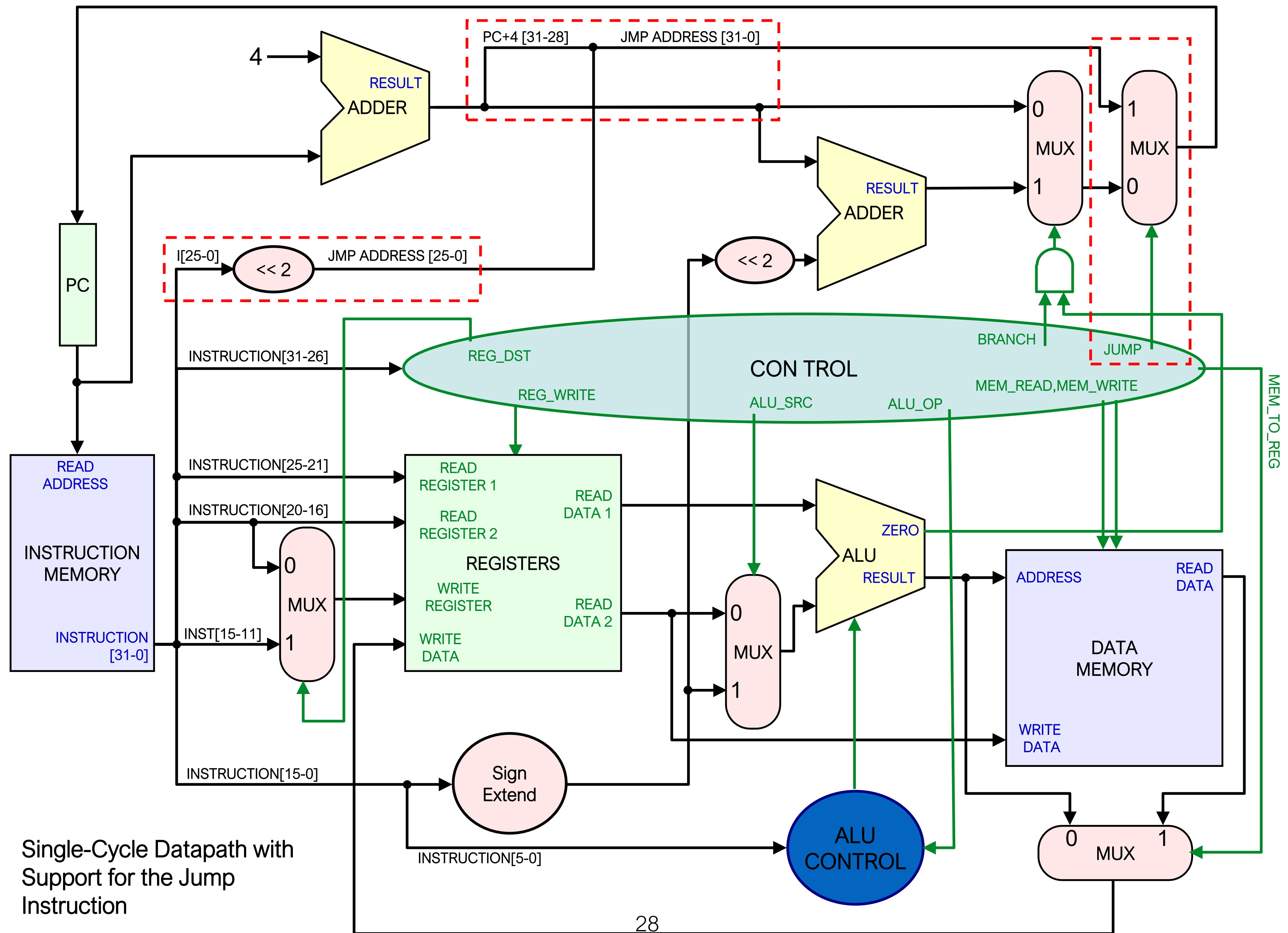
25-0

J Offset

J 10000

2 2500

Actual Address (in words) which we multiply by 4 ($\ll 2$) to get 28-Bit Address, then concatenate to upper 4 bits of PC+4 to get the 32-bit address of instruction to which we branch unconditionally



Single-Cycle Datapath with Support for the Jump Instruction