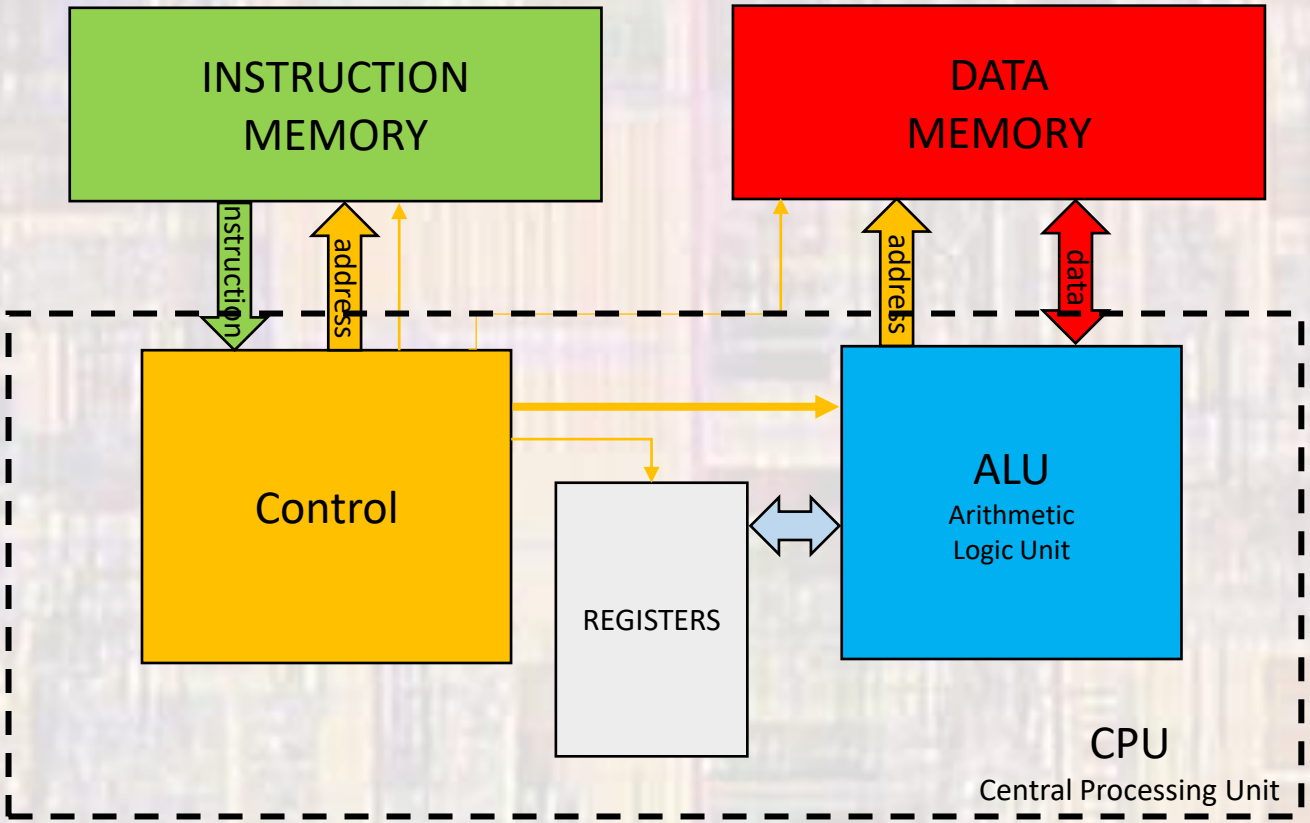
Linear Program Execution Example

Last updated 6/25/24

These slides show an example of linear program flow

Linear Program Execution

- Processor Architecture
  - Harvard – separate Instruction and Data memory paths



Linear Program Execution

- RISC Instruction set
  - 2 basic types of instructions
    - Register based instructions
    - Memory instructions
  - Register Instructions
    - Only allow access to the internal registers
      - Arithmetic
      - Logical
      - Control
  - Memory Operations
    - Read or write to memory/registers

Linear Program Execution

- Instruction Execution
  - These steps happen in hardware – we do not control them directly

	Fetch		get next instruction from instruction memory
	Decode		determine what the instruction is
depend on instruction	Execute		if necessary – do what the instruction requires
	Load	Mem	if necessary – get value from data memory
	Store		if necessary – place value in data memory
	Write Back		if necessary – store result in register

Linear Program Execution

- Instruction Sequencing
  - Program Counter (PC)
    - Register that holds the NEXT instruction memory location to be fetched
    - Provides the address for the instruction memory read
    - Typically the register is incremented each clock cycle
      - Incremented by the size of an instruction
      - e.g. for a 16 bit instruction word the PC would be incremented by 2
        - 0x1234 to 0x1236 since each instruction uses up 2 bytes

Linear Program Execution

- Instruction Sequencing
  - Program control
    - Linear flow – increment PC normally

Linear Program Execution

The compiler has assigned
x to memory location 0x0C
y to memory location 0x10
z to memory location 0x14

- 1 line of code - complete

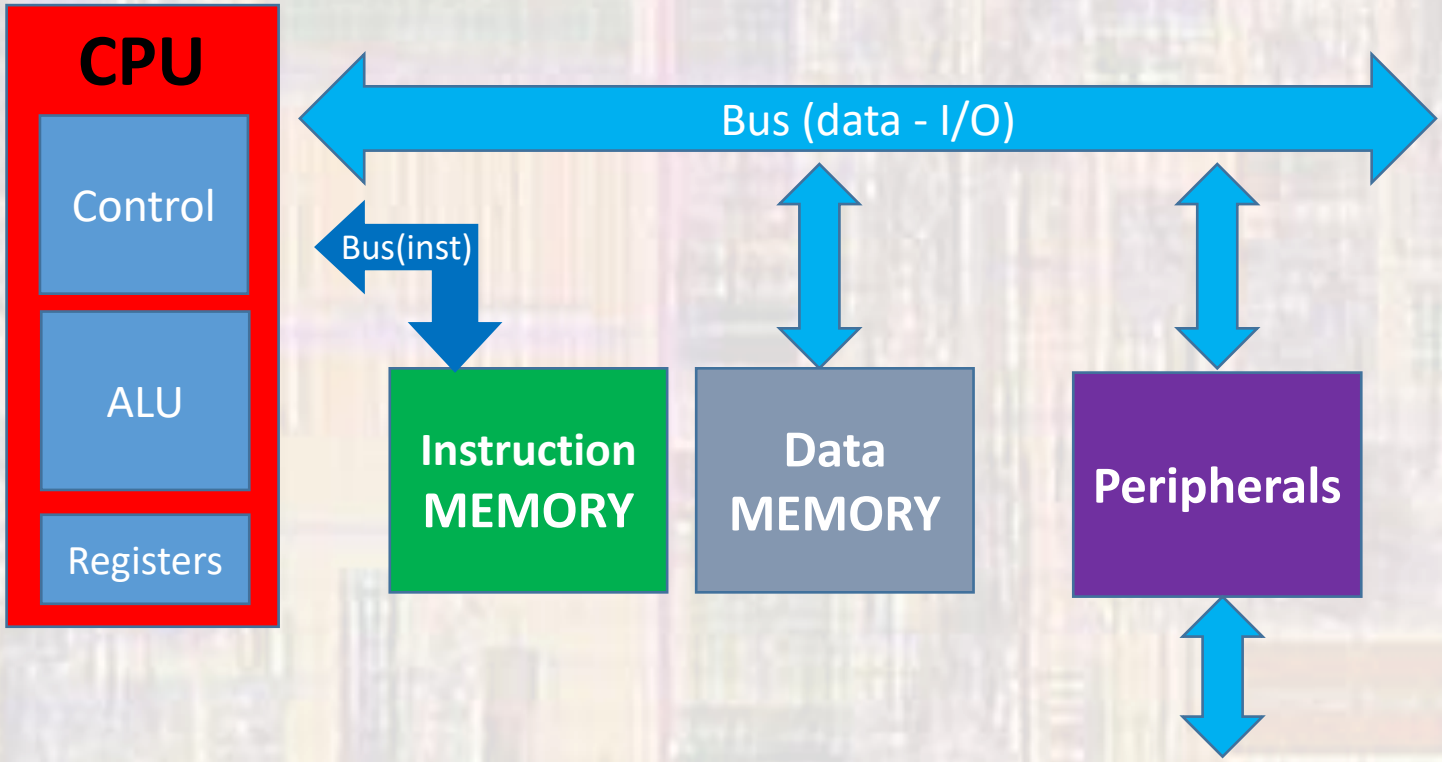
$$z = x + y;$$

The compiler turned the single line into 7 instructions

Mem loc	Instruction	Encoding	action
1000	ldi RA, 12	0xC00C	Load loc for x into RA
1002	ld RA, RB	0x8040	Put value at loc for x in RB ld RB, mem(RA)
1004	ldi RA, 16	0xC010	Load loc for y into RA
1006	ld RA, RC	0x8080	Put value at loc for y in RC ld RC, mem(RA)
1008	add RB, RC, RD	0x46C0	RD <- RB + RC
100A	ldi RA, 20	0xC014	Load loc for z into RA
100C	st RA, RD	0x9300	Put value of RD into loc for z st mem(RA), RD

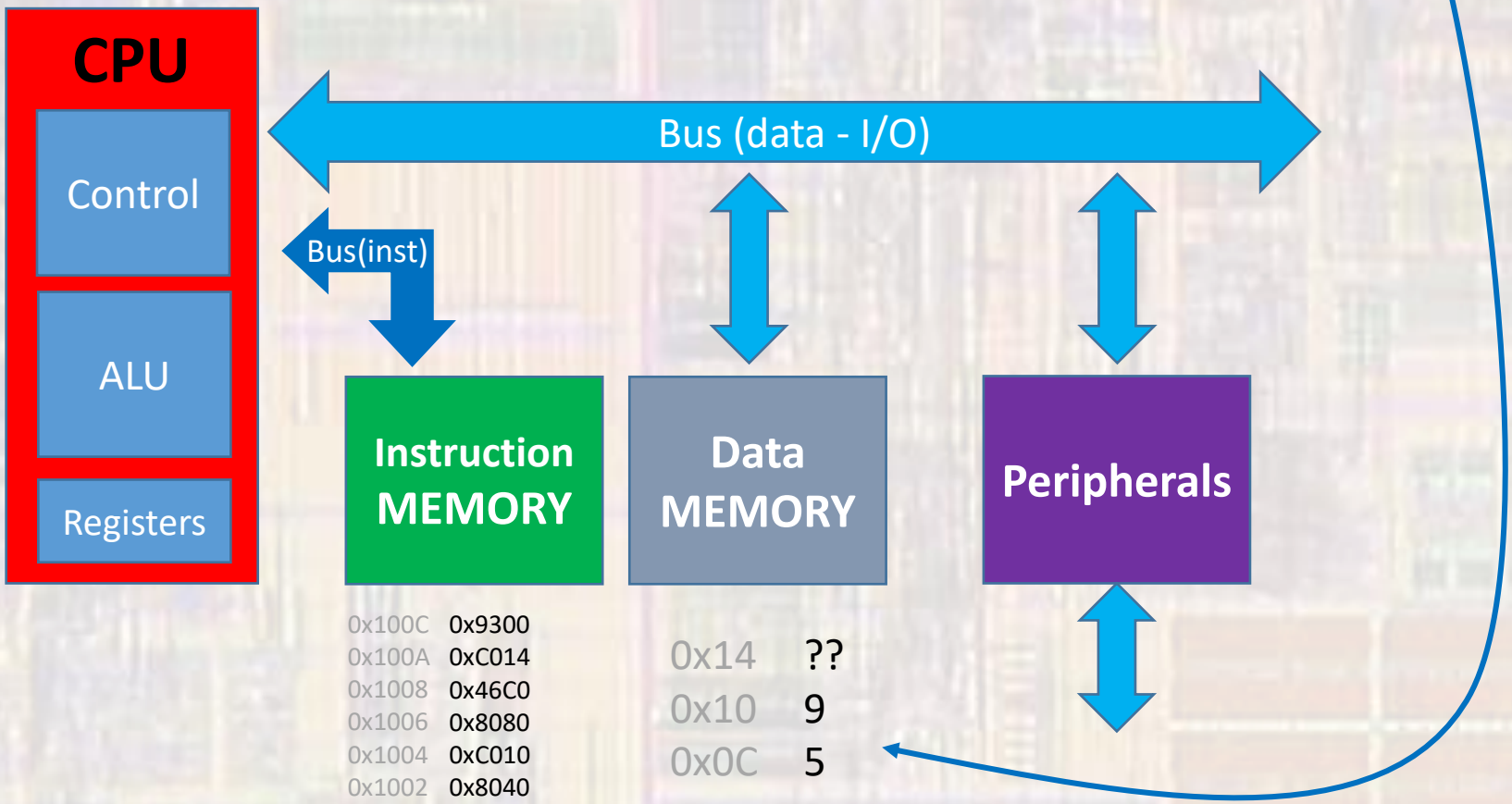
Linear Program Execution

- Simplified Block Diagram



Linear Program Execution

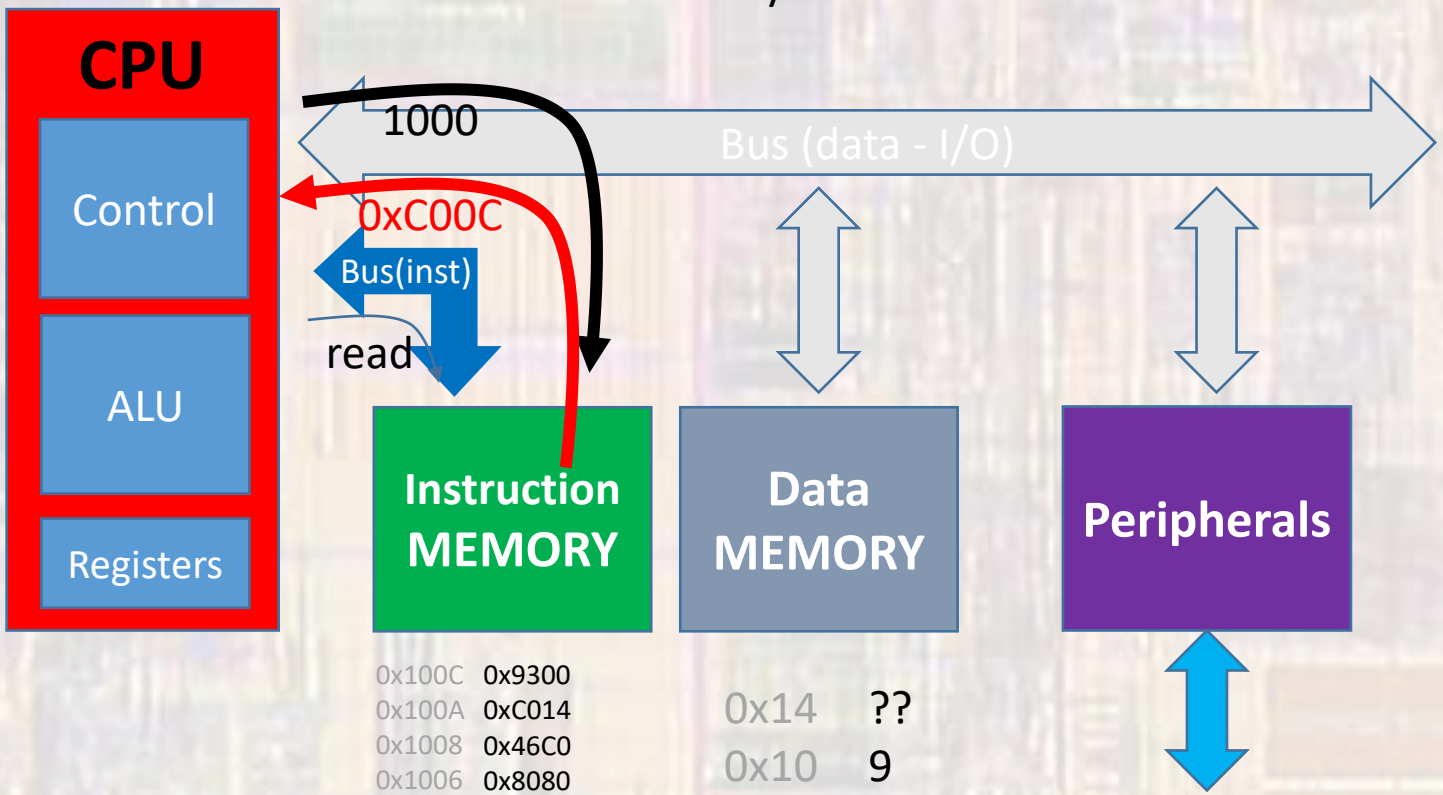
- Status
  - Data locations filled by previous commands
  - PC currently pointing to Instruction memory location 1000



Linear Program Execution

- First Instruction (fetch)

Control puts a memory location (1000)
on the address bus along with a read signal
Instruction memory returns the value at that location (0xC00C)

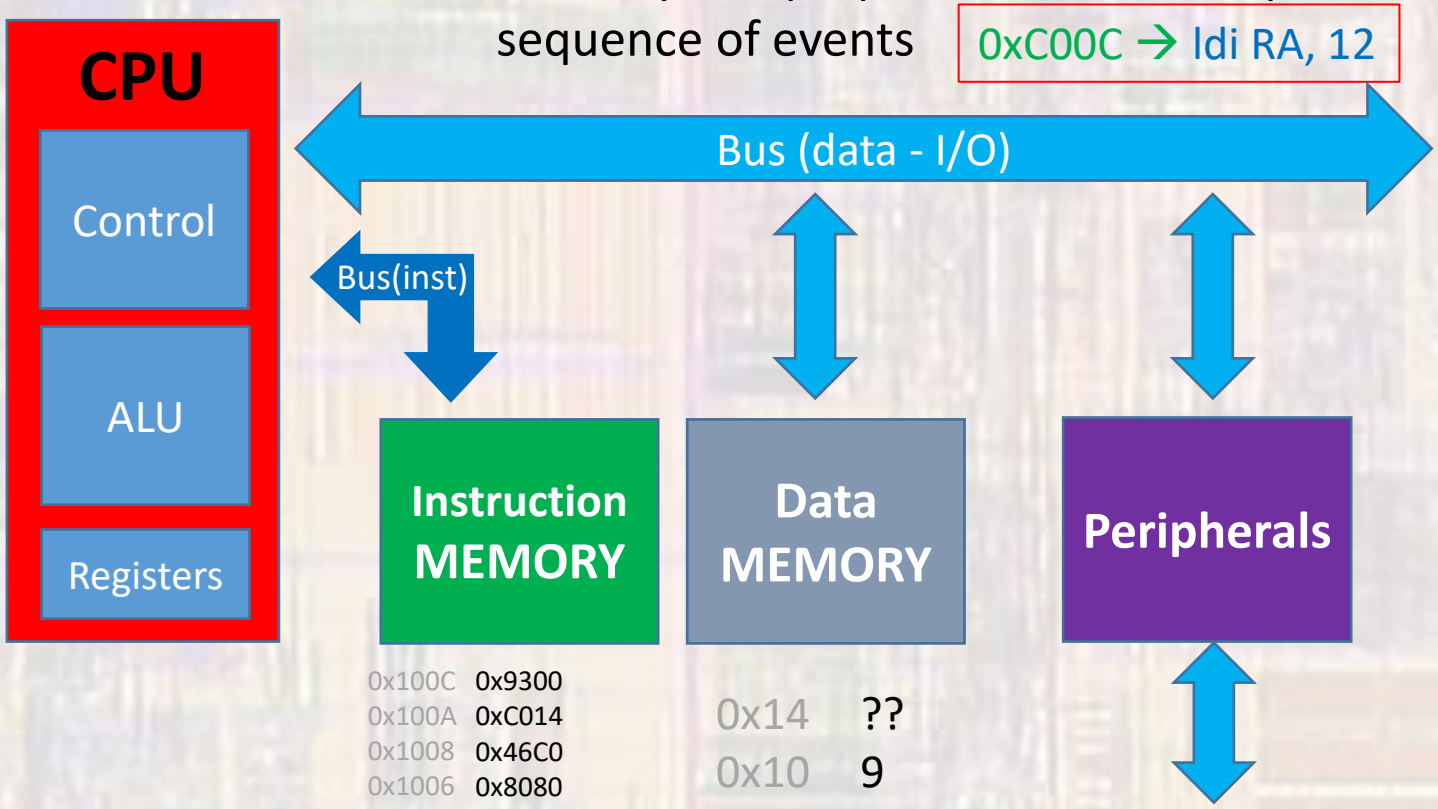


Linear Program Execution

- First Instruction (decode)

Control decodes the word returned by the memory and prepares to execute a pre-defined sequence of events

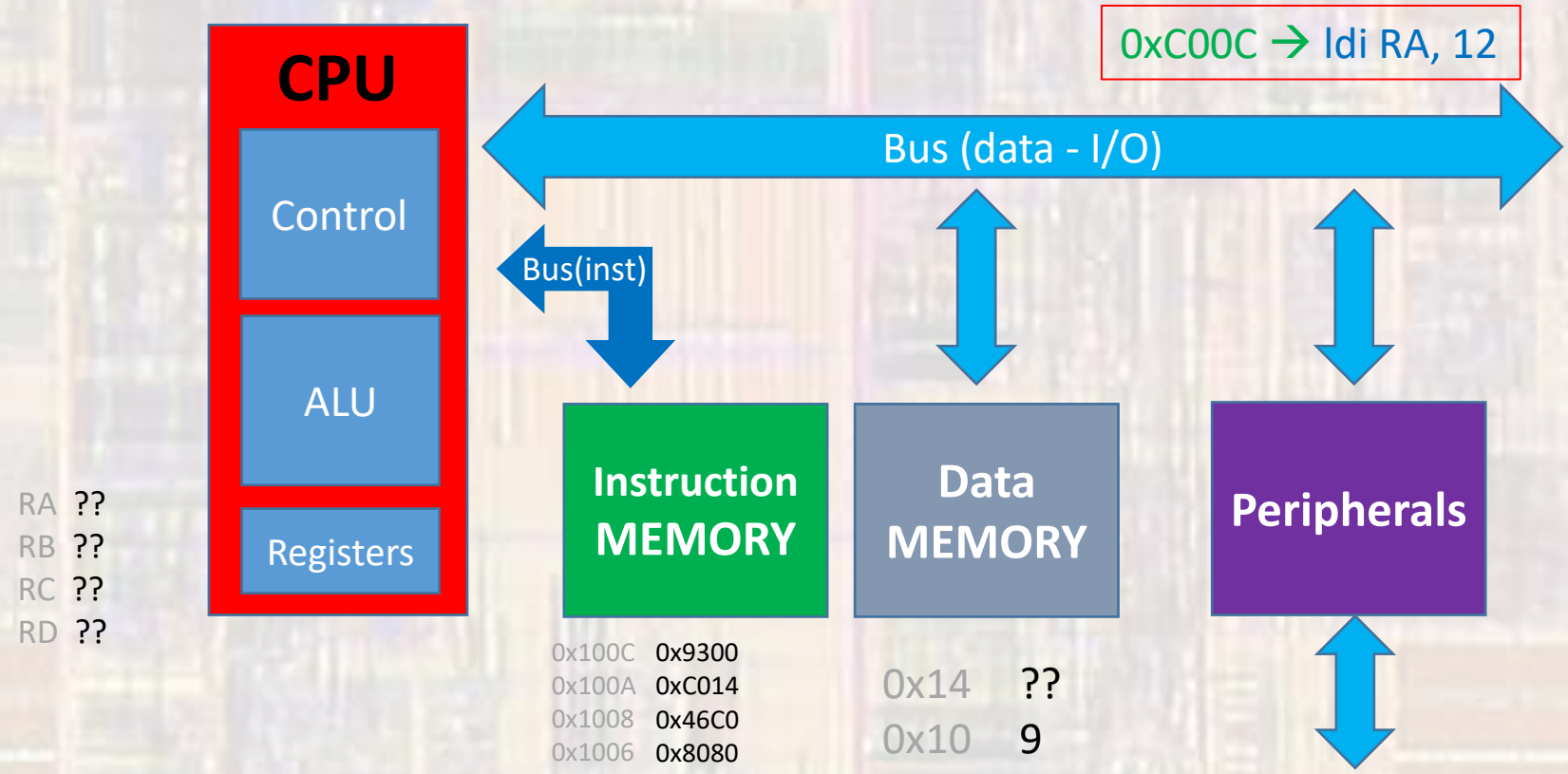
0xC00C → ldi RA, 12



Linear Program Execution

- First Instruction (execute)

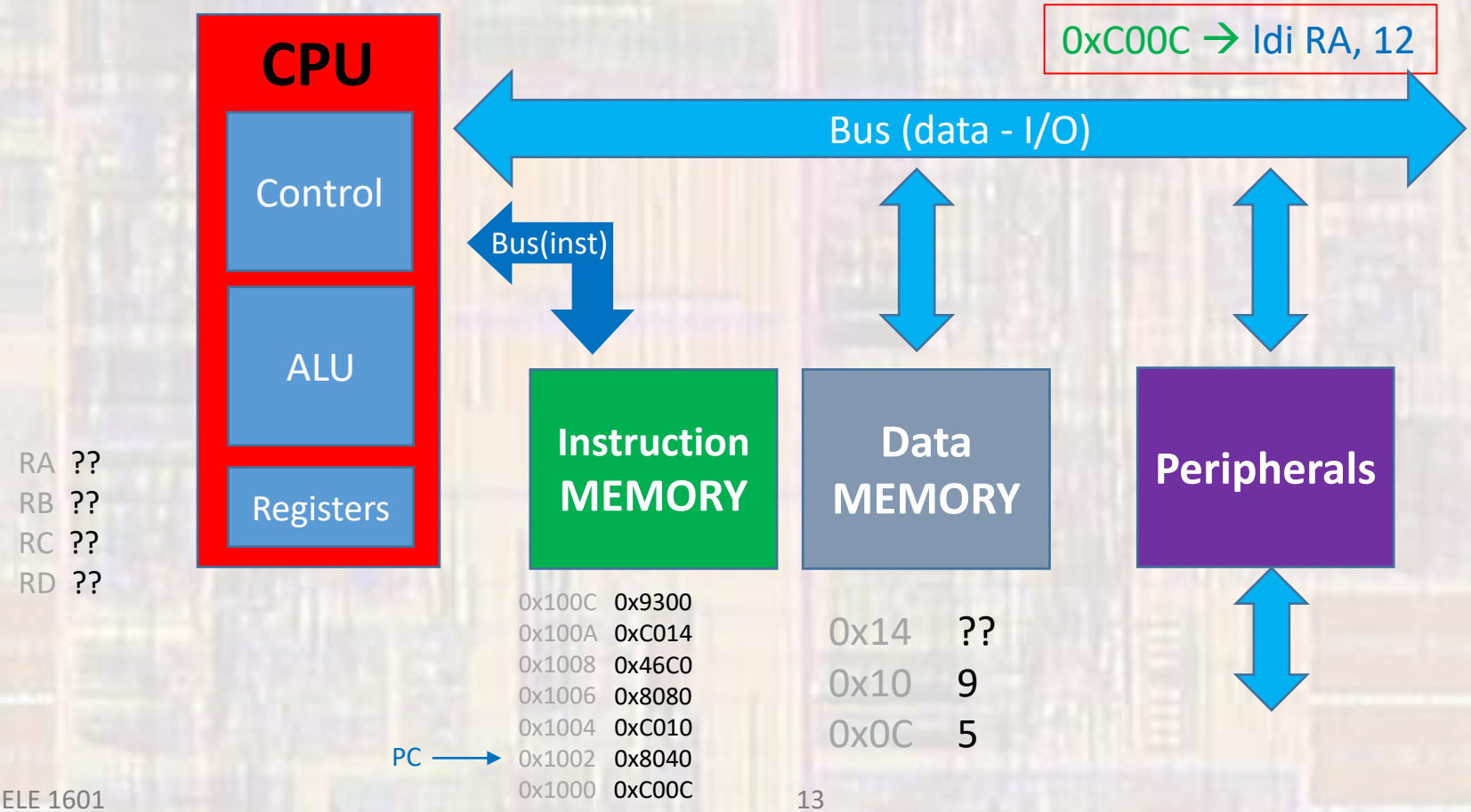
Does nothing for this instruction



Linear Program Execution

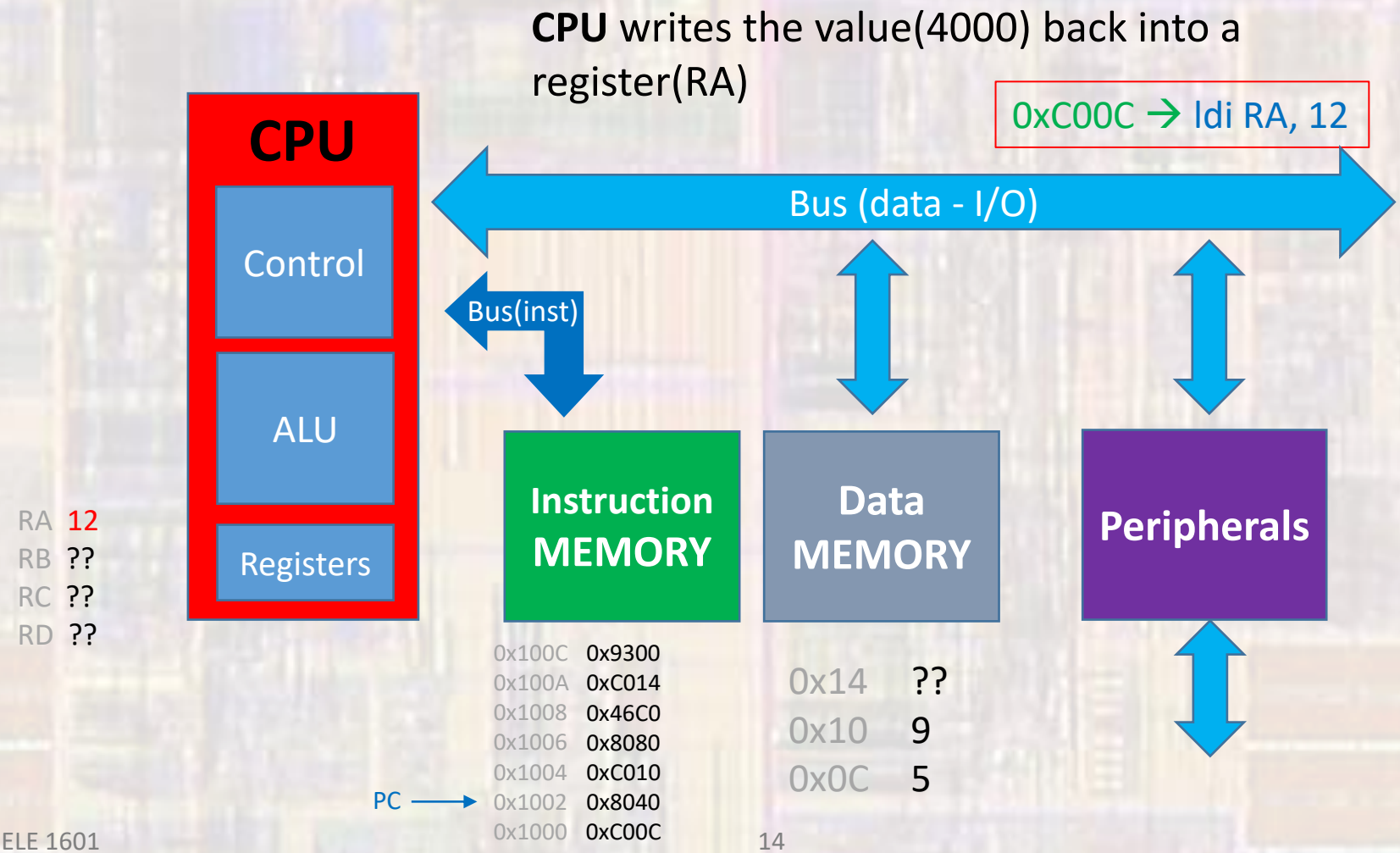
- First Instruction (mem)

Does nothing for this instruction



Linear Program Execution

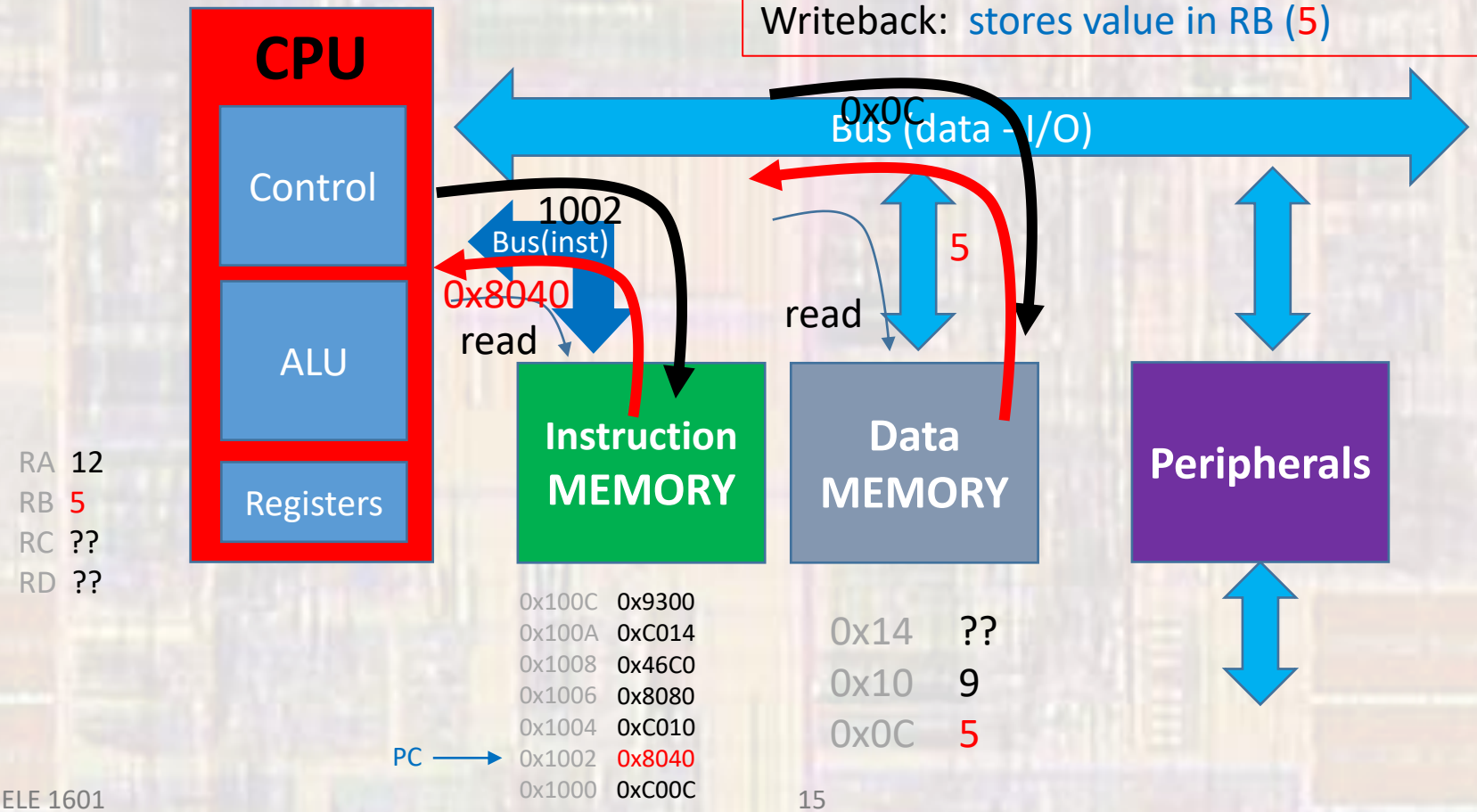
- First Instruction (write back)



Linear Program Execution

- New Fetch

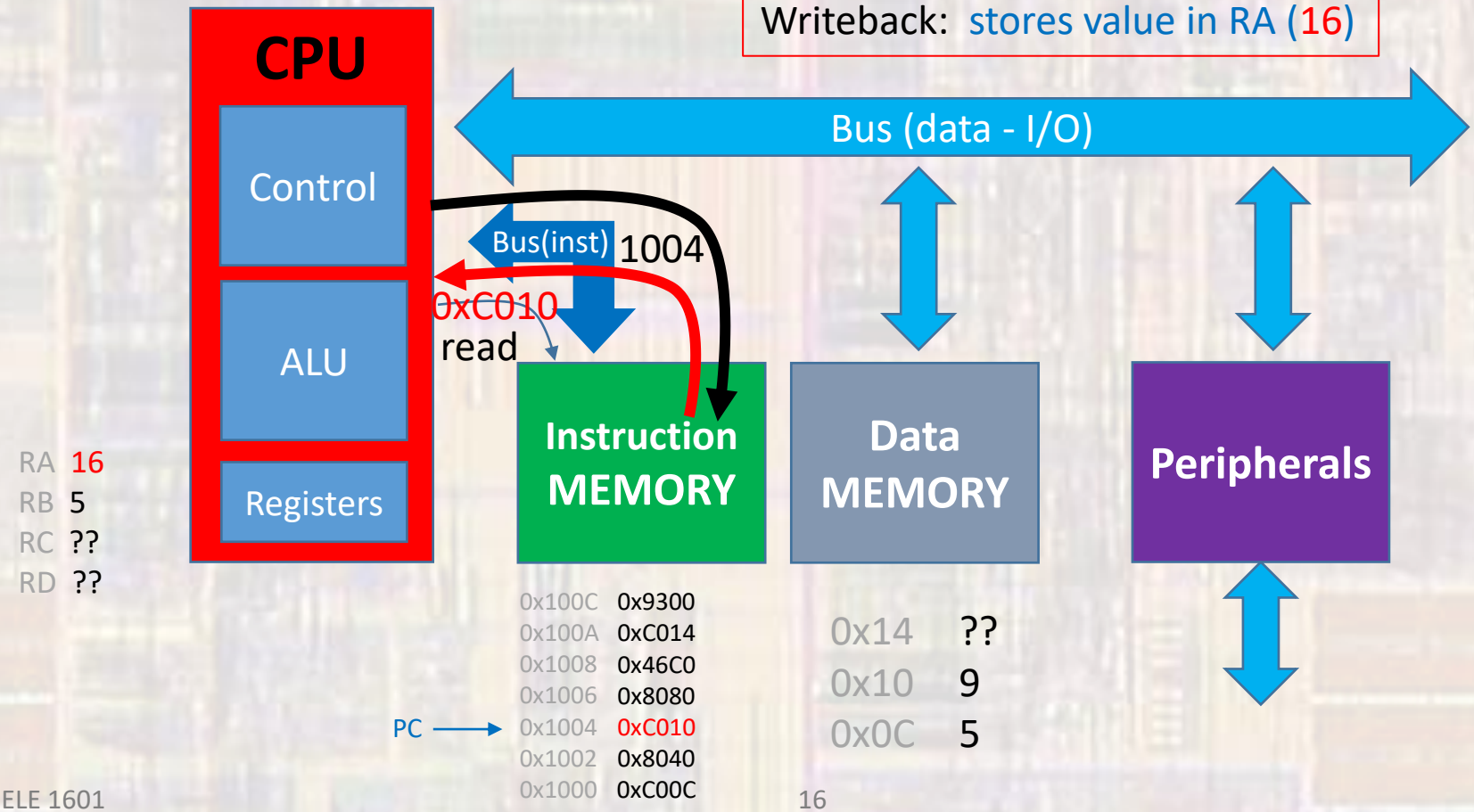
Fetch: → 1002 ← 0x8040
Decode: 0x8040 → ld RA, RB RB ← mem(RA)
Execute: idle
MEM: value at location in RA(12) = (5)
Writeback: stores value in RB (5)



Linear Program Execution

- New Fetch

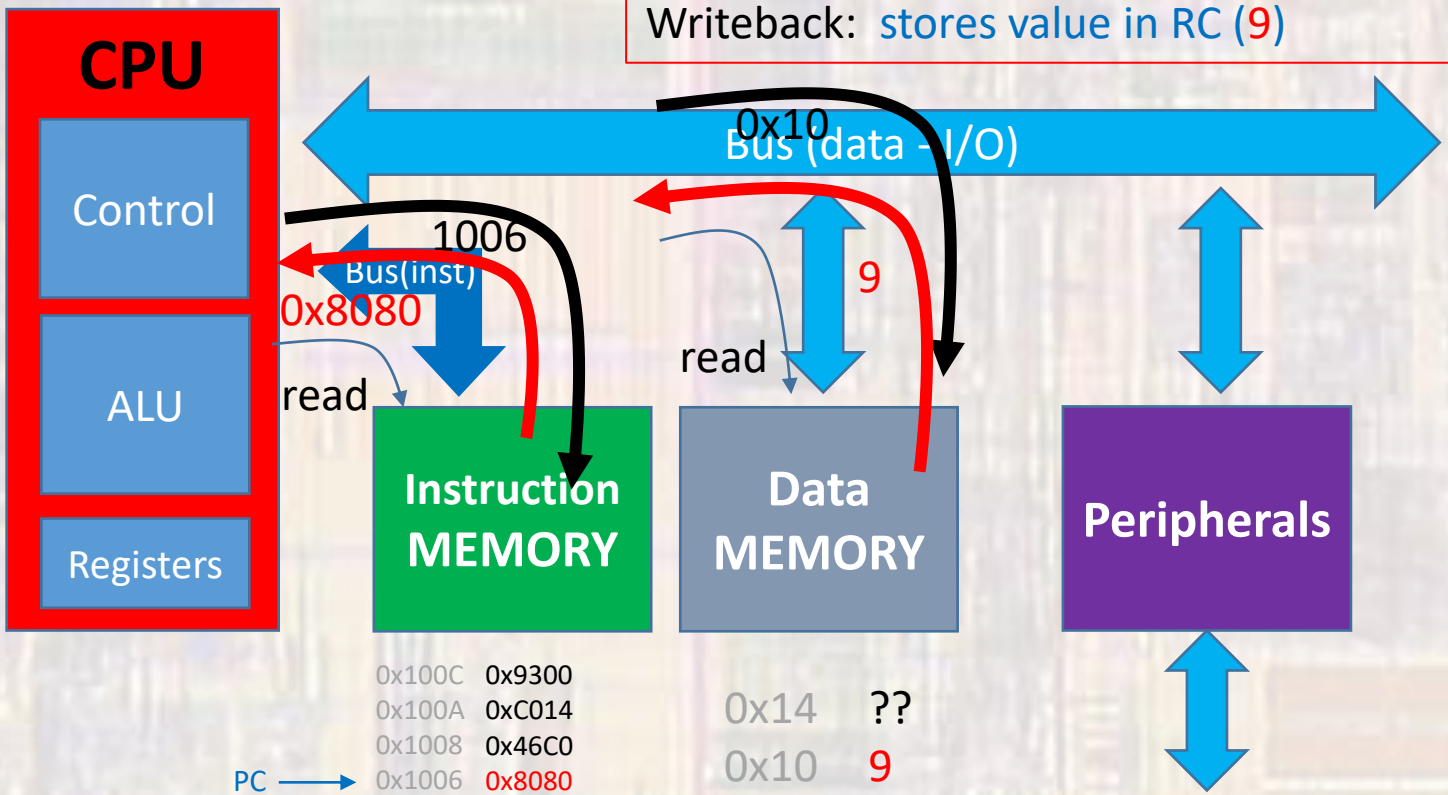
Fetch: → 1004 ← 0xC010
Decode: 0xC010 → ldi RA, 16
Execute: idle
MEM: idle
Writeback: stores value in RA (16)



Linear Program Execution

- New Fetch

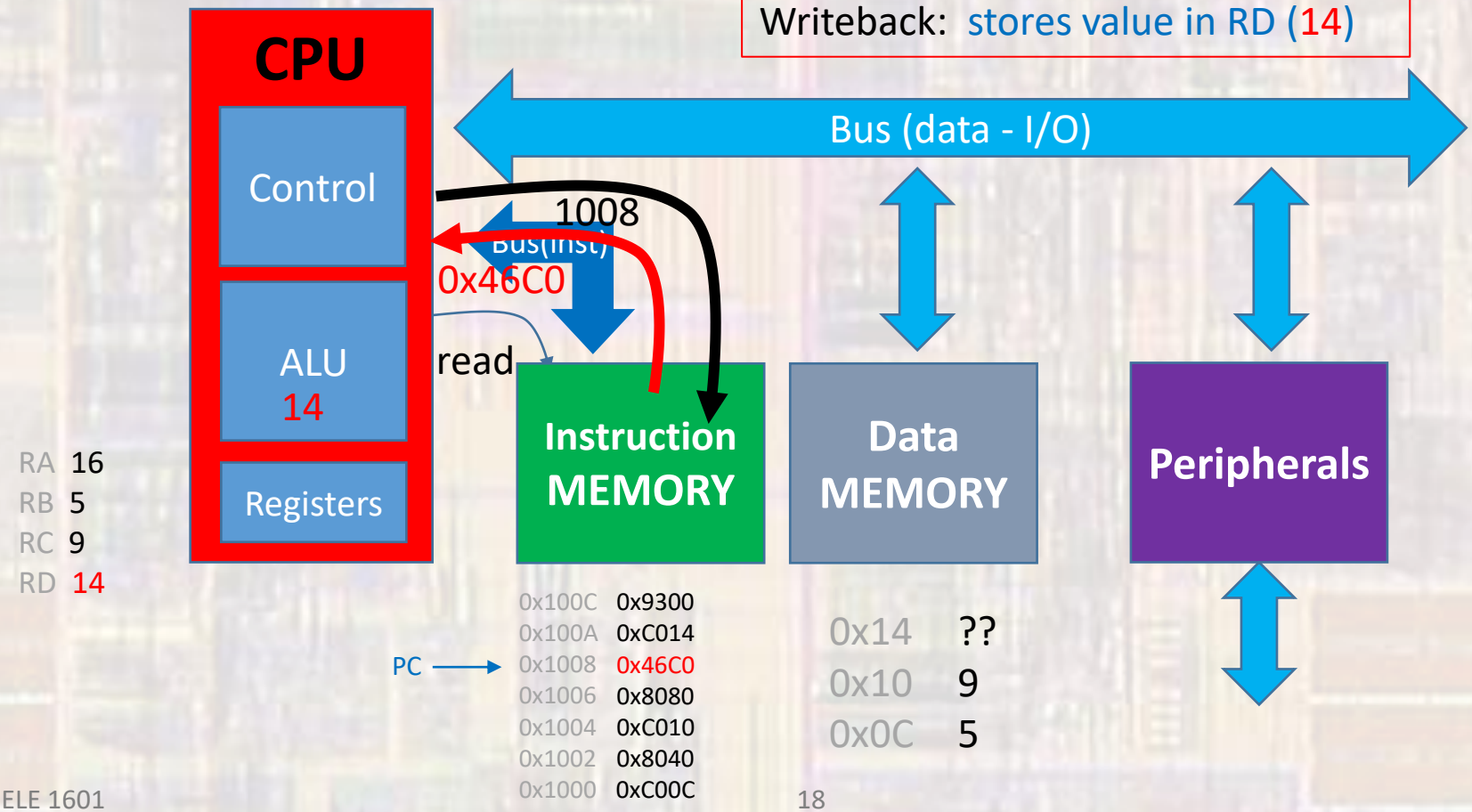
Fetch: → 1006 ← 0x8080
Decode: 0x8080 → ld RA, RC RC ← mem(RA)
Execute: idle
MEM: value at location in RA(16) = (9)
Writeback: stores value in RC (9)



Linear Program Execution

- New Fetch

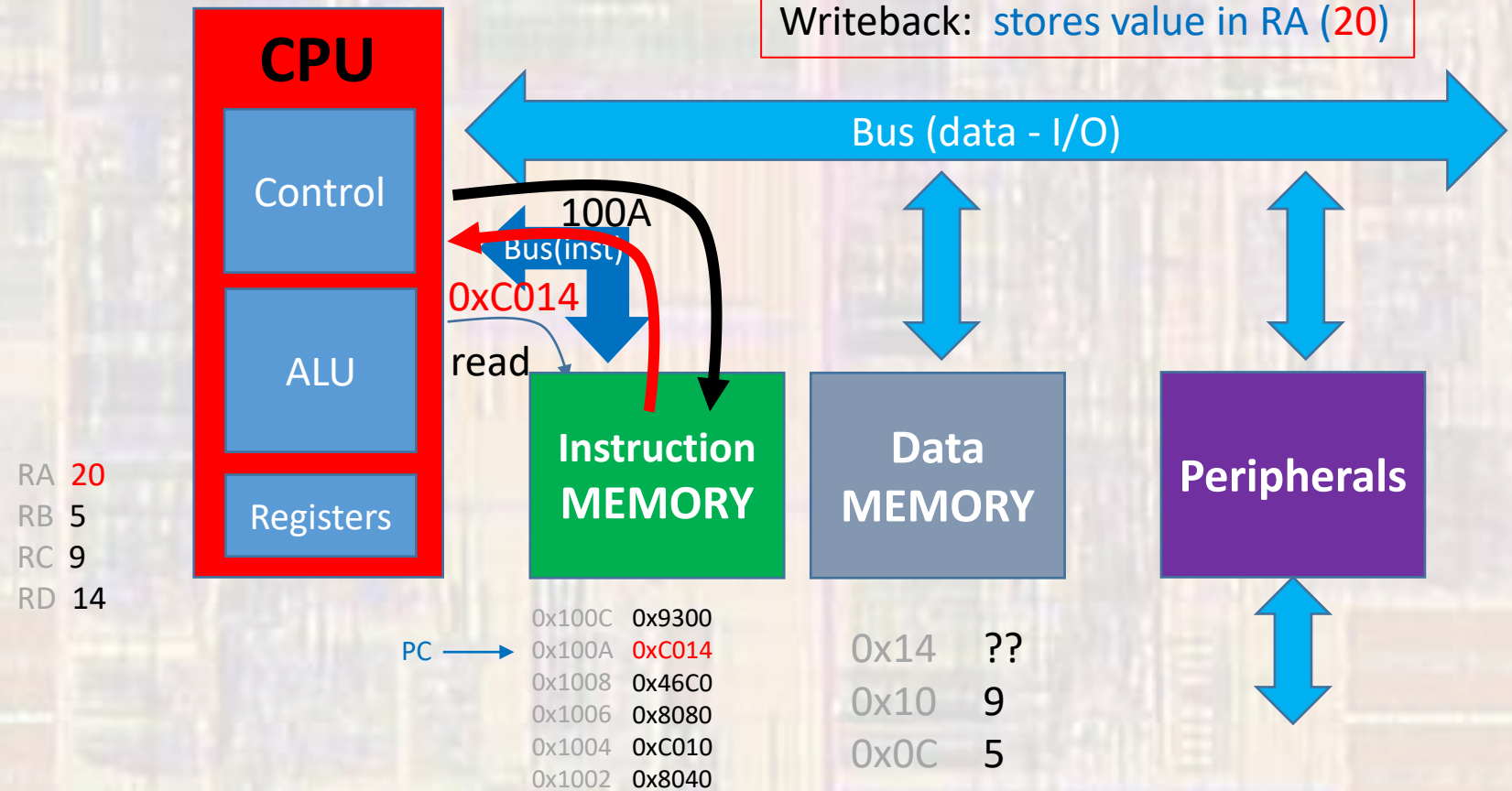
Fetch: → 1008 ← 0x46C0
Decode: 0x46C0 → add RB, RC, RD
Execute: adds RB + RC → 14
MEM: idle
Writeback: stores value in RD (14)



Linear Program Execution

- New Fetch

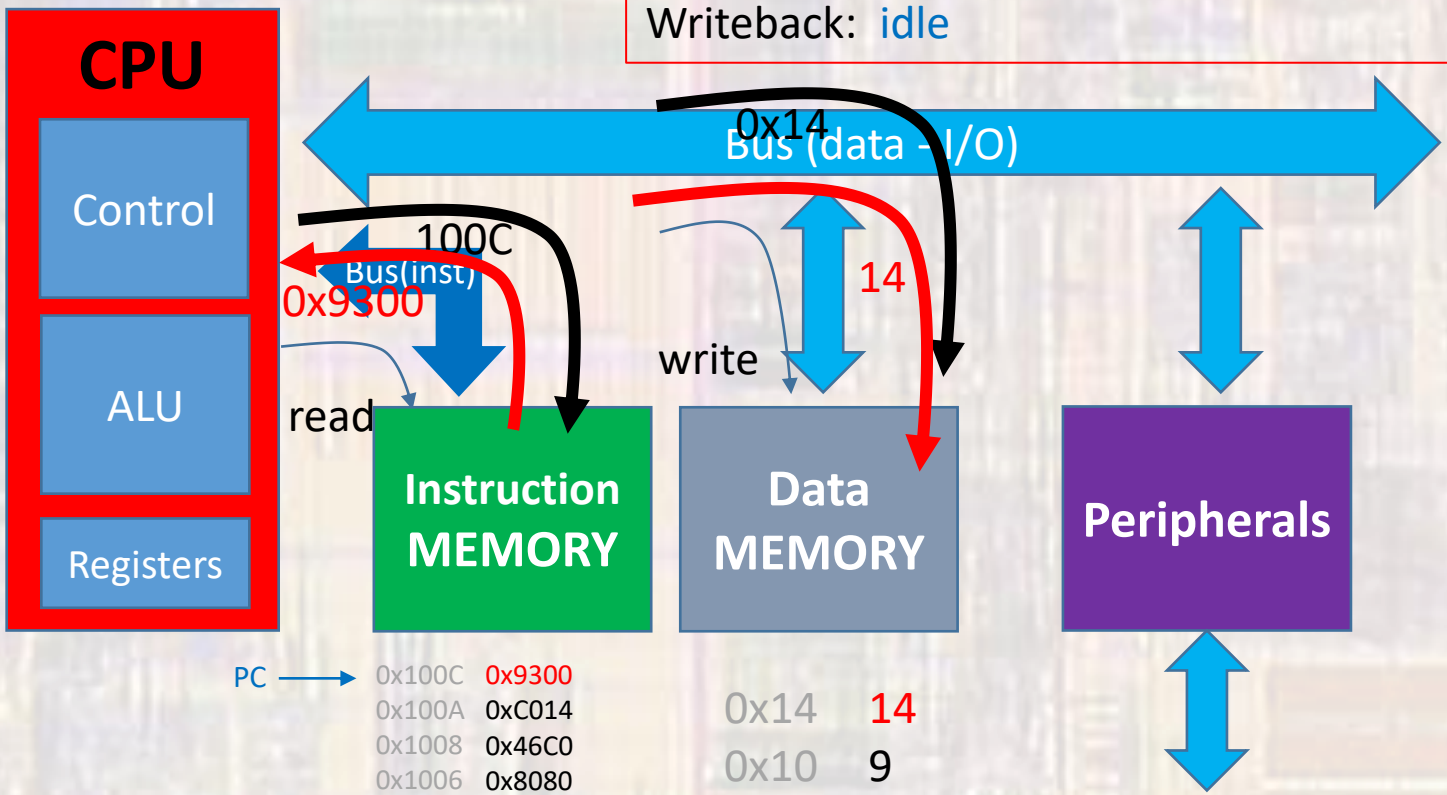
Fetch: → 100A ← 0xC014
Decode: 0xC014 → ldi RA, 20
Execute: idle
MEM: idle
Writeback: stores value in RA (20)



Linear Program Execution

- New Fetch

Fetch: → 100C ← 0x9300
Decode: 0x9300 → st RA, RD mem(RA) ← RD
Execute: idle
MEM: RD(14) → location in RA(20)
Writeback: idle



RA 20
RB 5
RC 9
RD 14

PC →

0x100C	0x9300
0x100A	0xC014
0x1008	0x46C0
0x1006	0x8080
0x1004	0xC010
0x1002	0x8040
0x1000	0xC00C

0x14	14
0x10	9
0x0C	5

Linear Program Execution

- Timing and memory

z = x + y; ⇔

Mem loc	Instruction	Encoding	action
1000	ldi RA, 12	0xC00C	Load loc for x into RA
1002	ld RA, RB	0x8040	Put value at loc for x in RB ld RB, mem(RA)
1004	ldi RA, 16	0xC010	Load loc for y into RA
1006	ld RA, RC	0x8080	Put value at loc for y in RC ld RC, mem(RA)
1008	add RB, RC, RD	0x46C0	RD <- RB + RC
100A	ldi RA, 20	0xC014	Load loc for z into RA
100C	st RA, RD	0x9300	Put value of RD into loc for z st mem(RA), RD

- 1 line of code → 6 instructions
  - 6 Instruction memory words
- 1 line of code → 6 clock cycles
  - Lines of code and clock cycles are not easily correlated