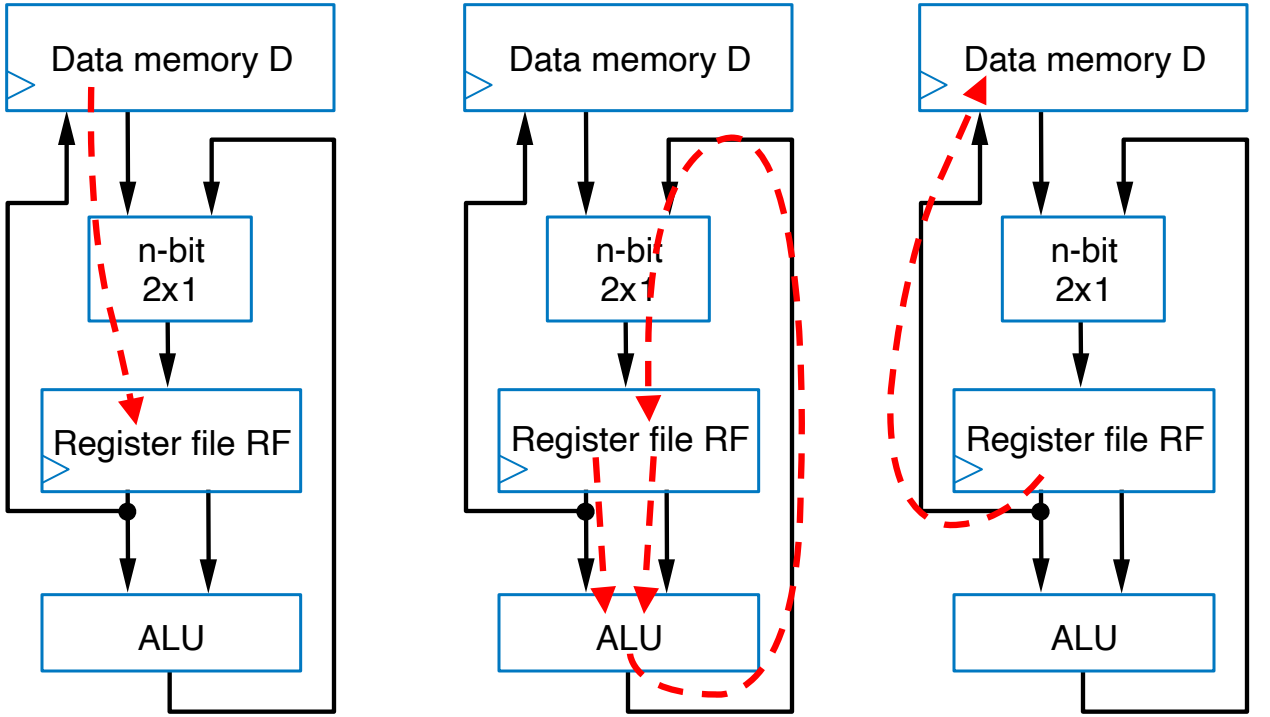
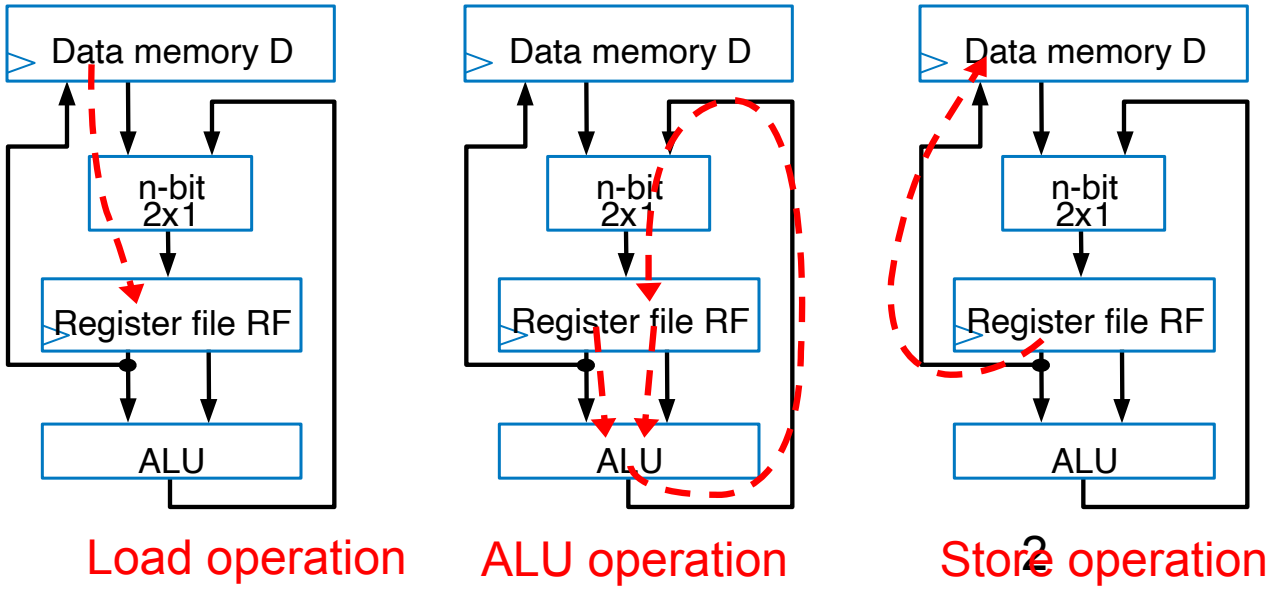
Basic Datapath Operations

- Load operation: Load data from data memory to RF
- ALU operation: Transforms data by passing one or two RF register values through ALU, performing operation (ADD, SUB, AND, OR, etc.), and writing back into RF.
- Store operation: Stores RF register value back into data memory
- Each operation can be done in one clock cycle



Basic Datapath Operations

- Q: Which are valid *single-cycle operations* for given datapath?
  - Move D[1] to RF[1] (i.e., RF[1] = D[1])
    - A: YES – That's a load operation
  - Store RF[1] to D[9] and store RF[2] to D[10]
    - A: NO – Requires two separate store operations
  - Add D[0] plus D[1], store result in D[9]
    - A: NO – ALU operation (ADD) only works with RF. Requires two load operations (e.g., RF$_a$[0]=D[0]; RF[1]=D[1], an ALU operation (e.g., RF[2]=RF[0]+RF[1]), and a store operation (e.g., D[9]=RF[2])

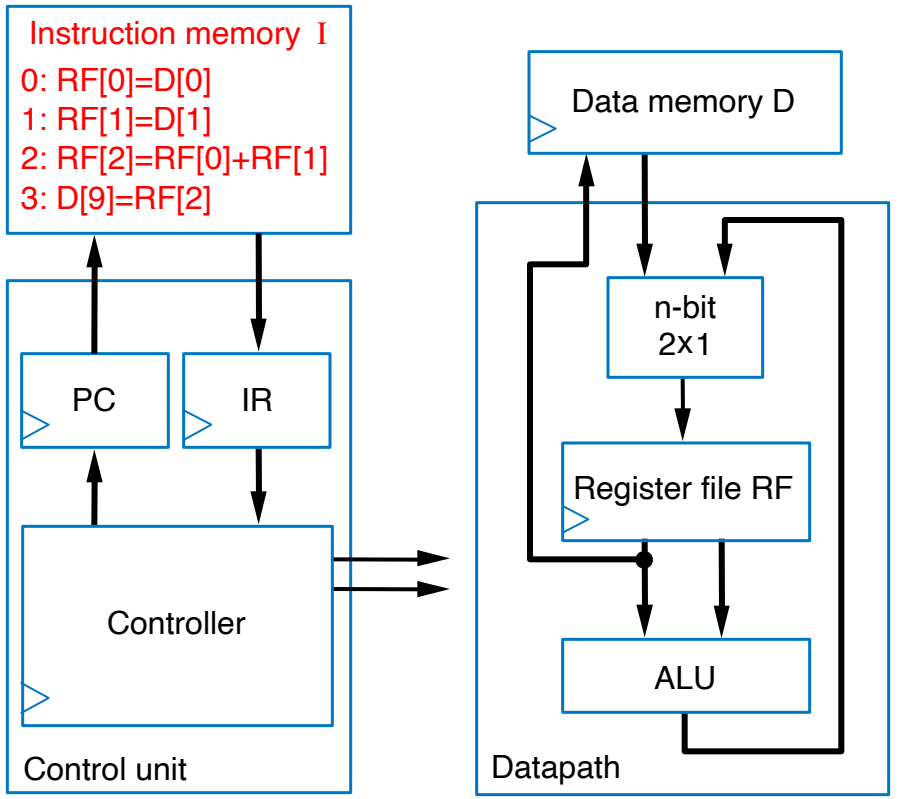


Basic Architecture – Control Unit

- D[9] = D[0] + D[1] – requires a sequence of four datapath operations:

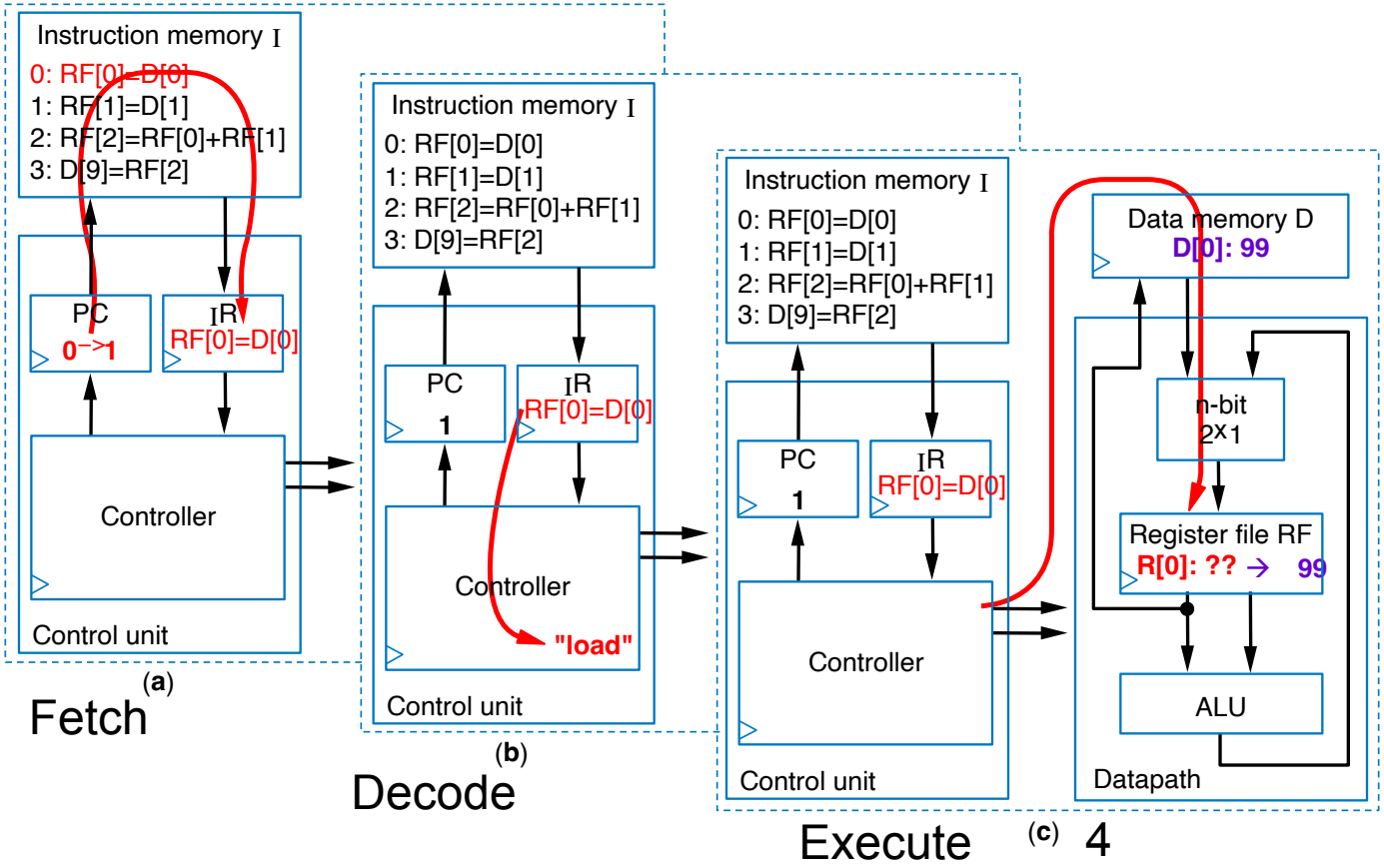
  0: RF[0] = D[0]
  1: RF[1] = D[1]
  2: RF[2] = RF[0] + RF[1]
  3: D[9] = RF[2]

- Each operation is an *instruction*
  - Sequence of instructions – *program*
  - Looks cumbersome, but that's the world of programmable processors – Decomposing desired computations into processor-supported operations
  - Store program in *Instruction memory*
  - *Control unit* reads each instruction and executes it on the datapath
    - PC: Program counter – address of current instruction
    - IR: Instruction register – current instruction



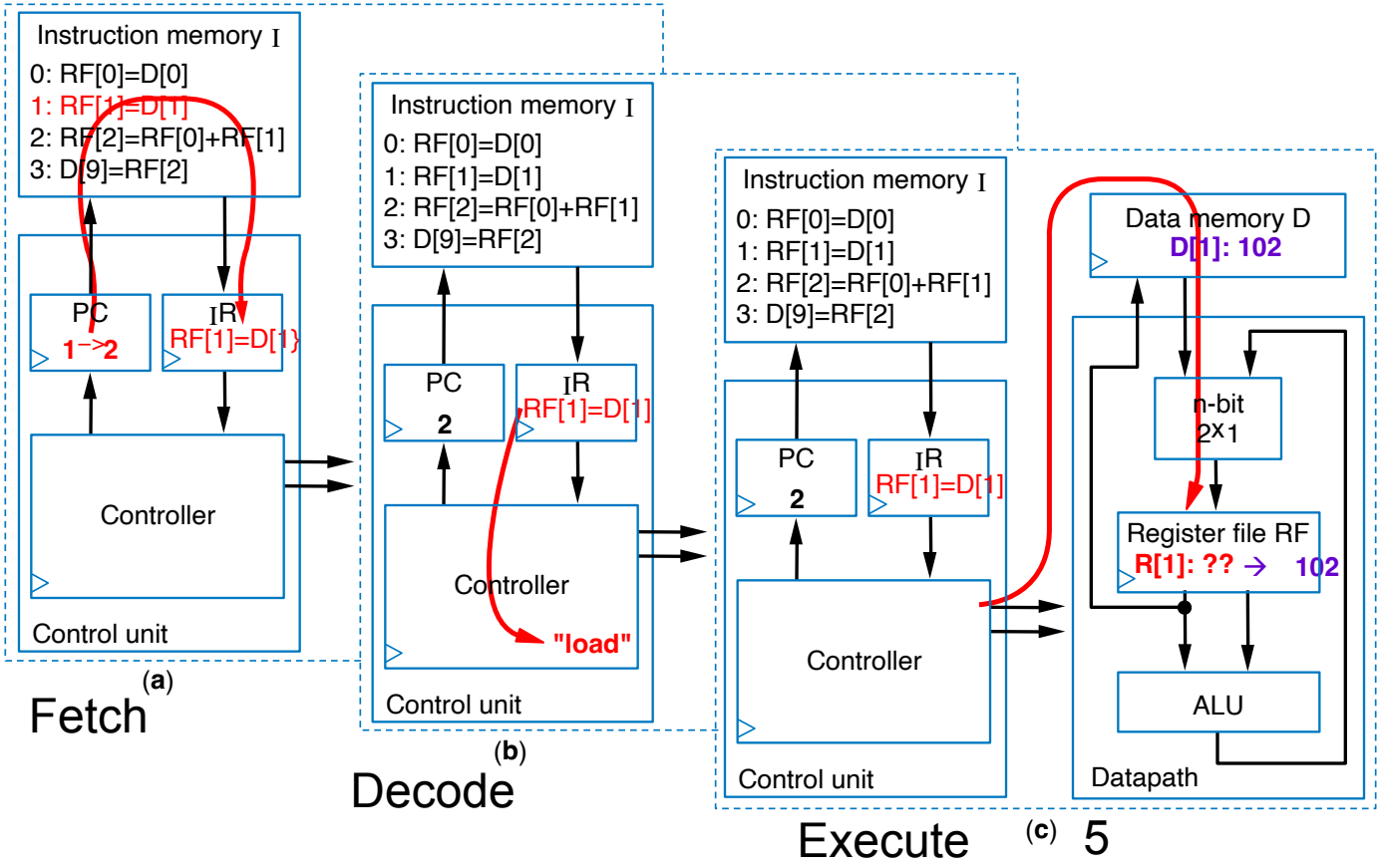
Basic Architecture – Control Unit

- To carry out *each instruction*, the control unit must:
  - Fetch – Read instruction from inst. mem.
  - Decode – Determine the operation and operands of the instruction
  - Execute – Carry out the instruction's operation using the datapath



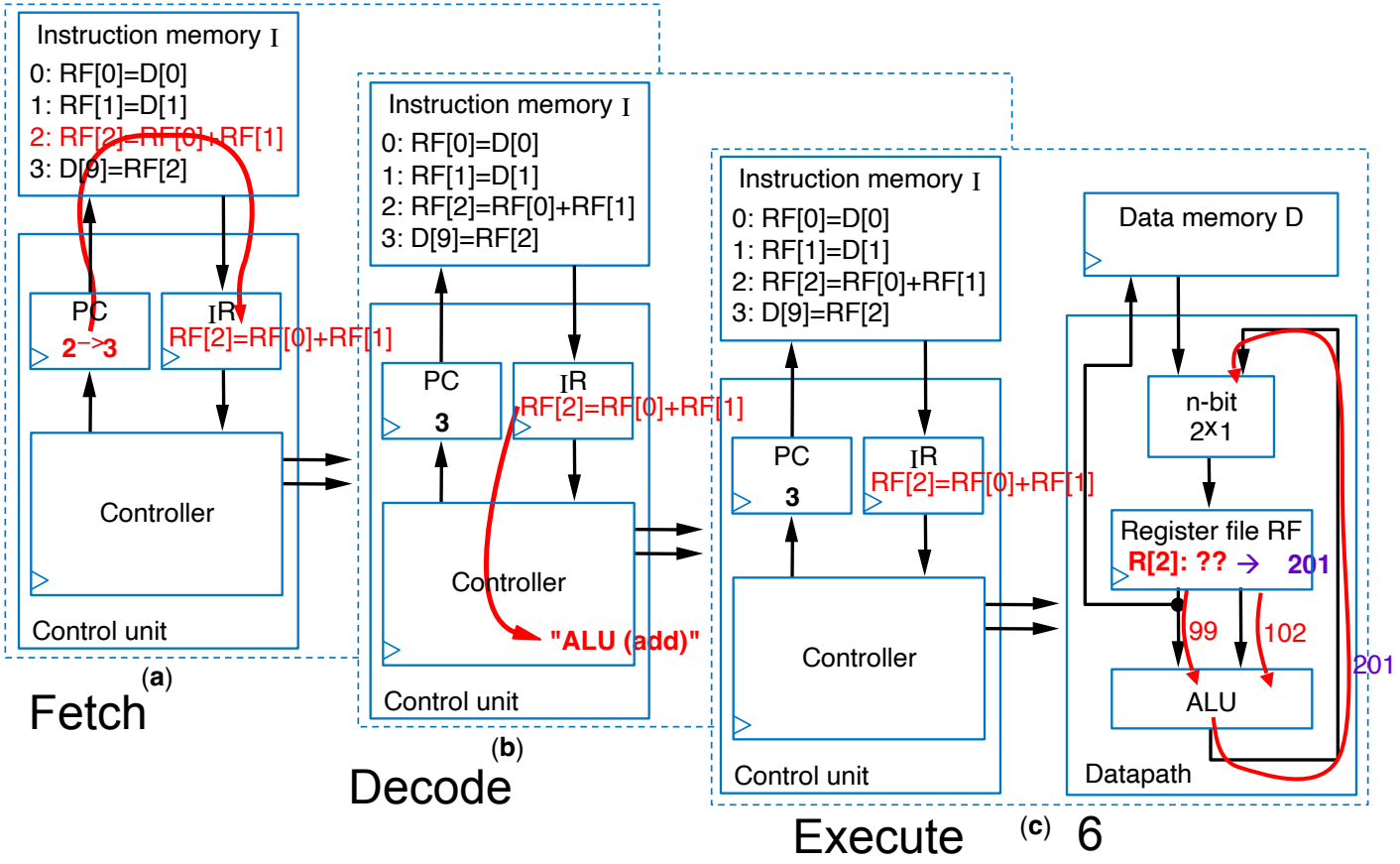
Basic Architecture – Control Unit

- To carry out *each instruction*, the control unit must:
  - Fetch – Read instruction from inst. mem.
  - Decode – Determine the operation and operands of the instruction
  - Execute – Carry out the instruction's operation using the datapath



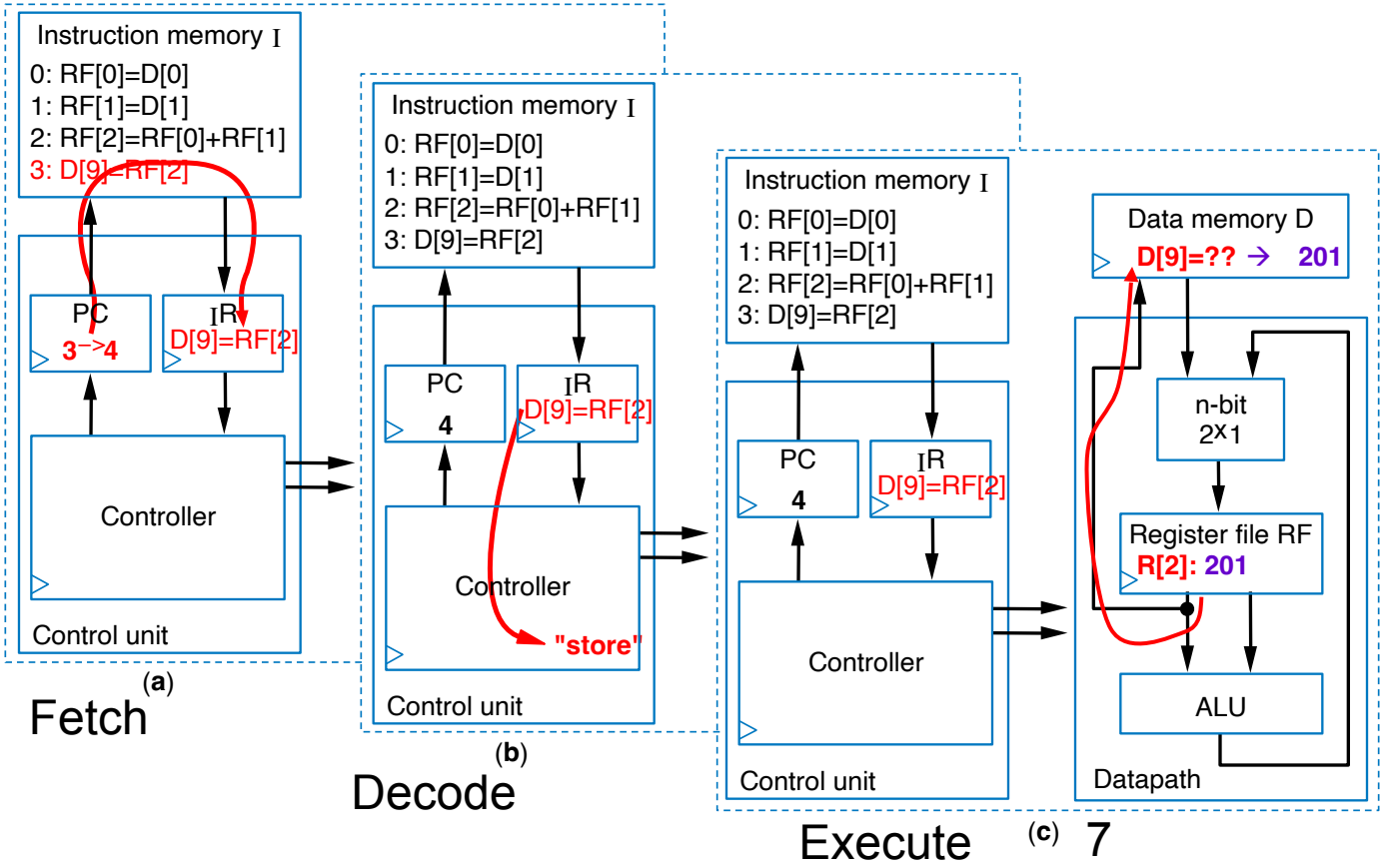
Basic Architecture – Control Unit

- To carry out *each instruction*, the control unit must:
  - Fetch – Read instruction from inst. mem.
  - Decode – Determine the operation and operands of the instruction
  - Execute – Carry out the instruction's operation using the datapath



Basic Architecture – Control Unit

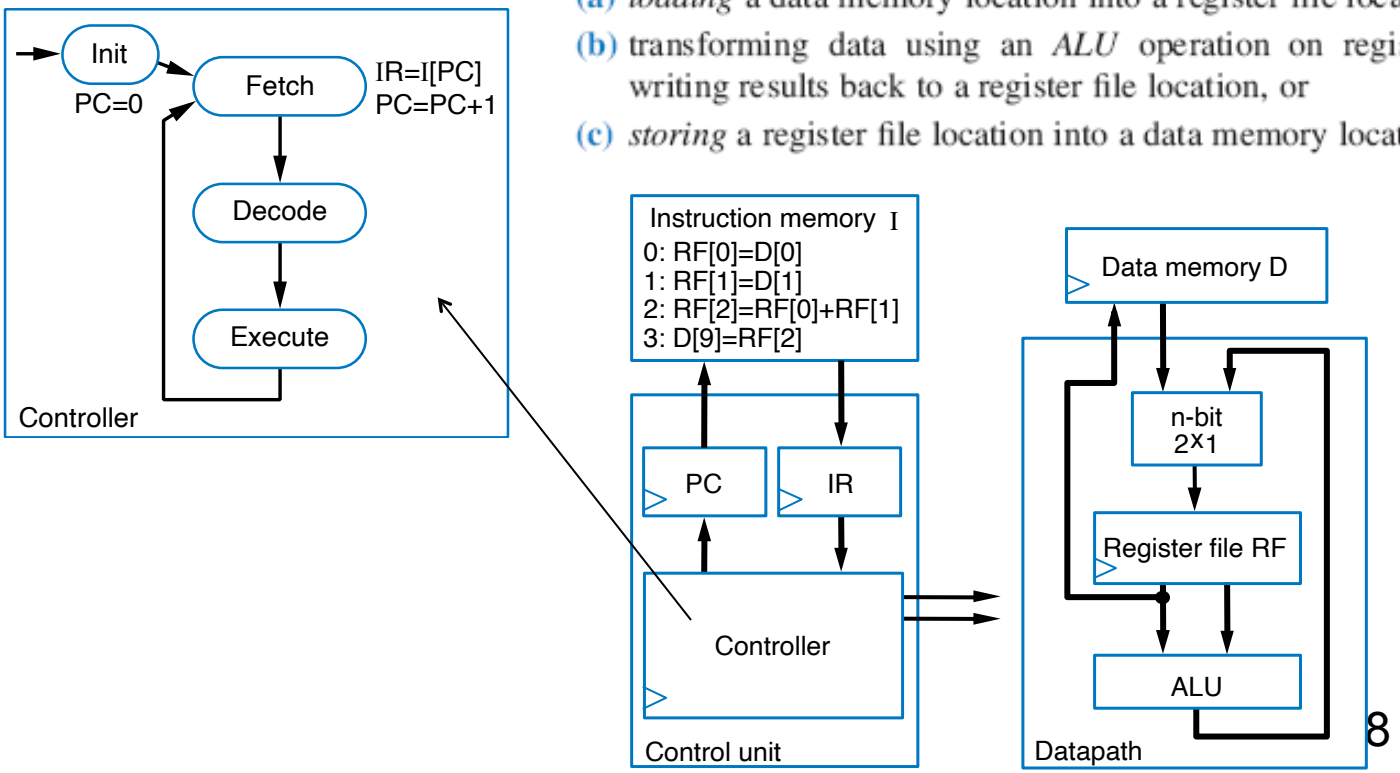
- To carry out *each instruction*, the control unit must:
  - Fetch – Read instruction from inst. mem.
  - Decode – Determine the operation and operands of the instruction
  - Execute – Carry out the instruction's operation using the datapath



Basic Architecture – Control Unit

To summarize, the control unit processes each instruction in three stages:

1. first *fetching* the instruction by loading the current instruction into *IR* and incrementing the *PC* for the next fetch,
2. next *decoding* the instruction to determine its operation, and
3. finally *executing* the operation by setting the appropriate control lines for the datapath, if applicable. If the operation is a datapath operation, the operation may be one of three possible types:
   (a) *loading* a data memory location into a register file location,
   (b) transforming data using an *ALU* operation on register file locations and writing results back to a register file location, or
   (c) *storing* a register file location into a data memory location.



Creating a Sequence of Instructions

- Q: Create sequence of instructions to compute D[3] = D[0]+D[1]+D[2] on earlier-introduced processor
- A1: One possible sequence
  - First load data memory locations into register file
    - R[3] = D[0]
    - R[4] = D[1]
    - R[2] = D[2]

    (Note arbitrary register locations)
  - Next, perform the additions
    - R[1] = R[3] + R[4]
    - R[1] = R[1] + R[2]
  - Finally, store result
    - D[3] = R[1]
- A2: Alternative sequence
  - First load D[0] and D[1] and add them
    - R[1] = D[0]
    - R[2] = D[1]
    - R[1] = R[1] + R[2]
  - Next, load D[2] and add
    - R[2] = D[2]
    - R[1] = R[1] + R[2]
  - Finally, store result
    - D[3] = R[1]

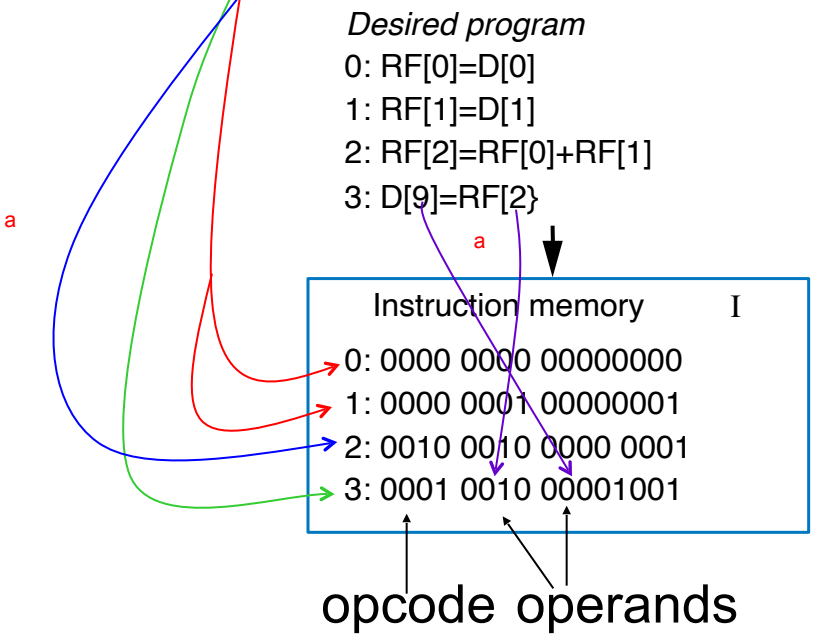
Number of Cycles

- Q: How many cycles are needed to execute six instructions using the earlier-described processor?

- A: Each instruction requires 3 cycles – 1 to fetch, 1 to decode, and 1 to execute
  - Thus, 6 instr * 3 cycles/instr = 18 cycles

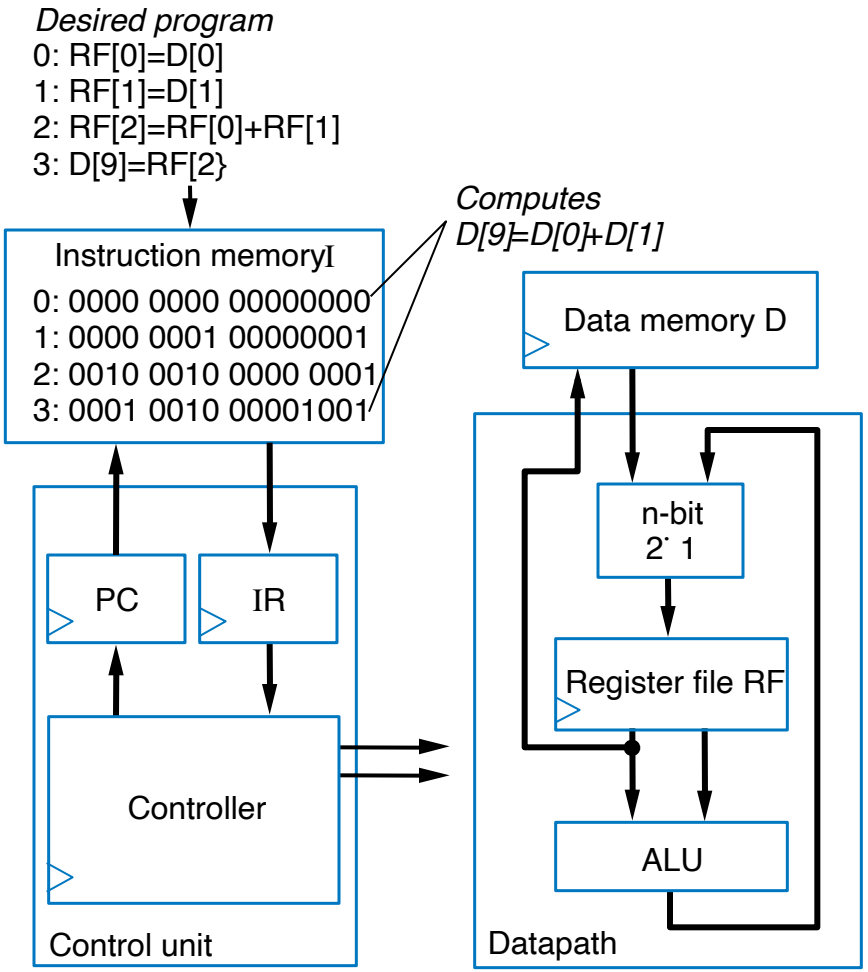
Three-Instruction Programmable Processor

- Instruction Set – List of allowable instructions and their representation in memory, e.g.,
  - ***Load*** instruction — **0000 $r_3r_2r_1r_0$ $d_7d_6d_5d_4d_3d_2d_1d_0$**
  - ***Store*** instruction — **0001 $r_3r_2r_1r_0$ $d_7d_6d_5d_4d_3d_2d_1d_0$**
  - ***Add*** instruction — **0010 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$**



Instructions in 0s and 1s – *machine code*

Program for Three-Instruction Processor



Program for Three-Instruction Processor

- Another example program in machine code
  - Compute D[5] = D[5] + D[6] + D[7]

```
0: 0000 0000 00000101  // RF[0] = D[5]
1: 0000 0001 00000110  // RF[1] = D[6]
2: 0000 0010 00000111  // RF[2] = D[7]
3: 0010 0000 0000 0001  // RF[0] = RF[0] + RF[1]
                        // which is D[5]+D[6]
4: 0010 0000 0000 0010  // RF[0] = RF[0] + RF[2]
                        // now D[5]+D[6]+D[7]
5: 0001 0000 00000101  // D[5] = RF[0]
```

- ***Load*** instruction — **0000 $r_3r_2r_1r_0$ $d_7d_6d_5d_4d_3d_2d_1d_0$**
- ***Store*** instruction — **0001 $r_3r_2r_1r_0$ $d_7d_6d_5d_4d_3d_2d_1d_0$**
- ***Add*** instruction — **0010 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$**

- Machine code (0s and 1s) hard to work with
- Assembly code – Uses mnemonics
  - ***Load*** instruction—**MOV Ra, d**
    - specifies the operation *RF[a]=D[d]*. *a* must be 0,1, ..., or 15—so *R0* means *RF[0]*, *R1* means *RF[1]*, etc. *d* must be 0, 1, ..., 255
  - • ***Store*** instruction—**MOV d, Ra**
    - specifies the operation *D[d]=RF[a]*
  - • ***Add*** instruction—**ADD Ra, Rb, Rc**
    - specifies the operation *RF[a]=RF[b]+RF[c]*

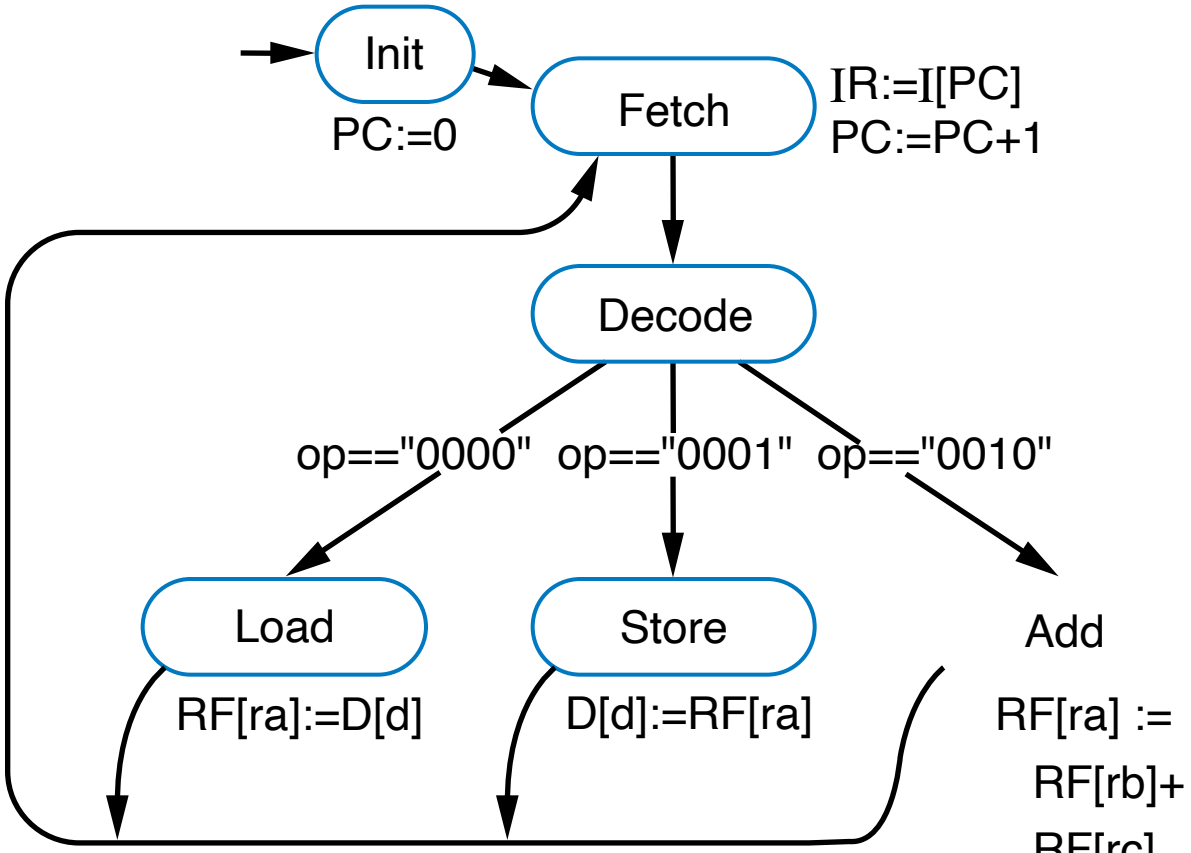
Desired program

0: RF[0]=D[0]
1: RF[1]=D[1]
2: RF[2]=RF[0]+RF[1]
3: D[9]=RF[2]

machine code	assembly code
0: 0000 0000 00000000	0: MOV R0, 0
1: 0000 0001 00000001	1: MOV R1, 1
2: 0010 0010 0000 0001	2: ADD R2, R0, R1
3: 0001 0010 00001009	3: MOV 9, R2

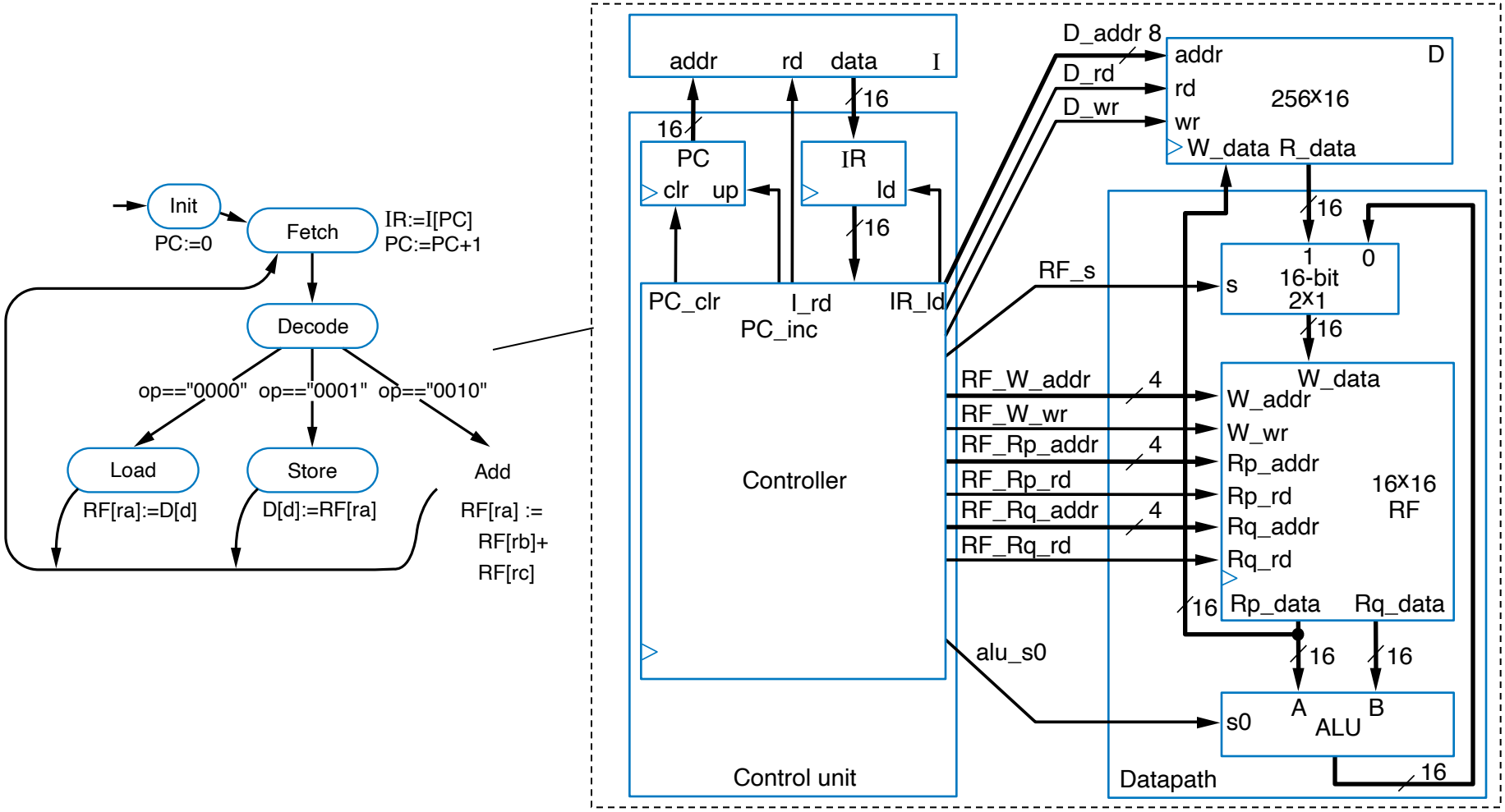
Control-Unit and Datapath for Three-Instruction Processor

- To design the processor, we can begin with a high-level state machine description of the processor's behavior



Control-Unit and Datapath for Three-Instruction Processor

- Create detailed connections among components



Control-Unit and Datapath for Three-Instruction Processor