Processor control flow

Simon Hollis, COMS12600

Controlling what?

- So far, we have a processor and an ISA, so what is missing to produce a useful system?
- One of the features that has made modern architectures so versatile is the ability to:
  - React to changes in data values
  - Support modular code

Recap: data and control

- You can think of a processor’s function as being dictated by two separate influences:
    - The ***control*** information (which tells it what to do)
    - The ***data*** information being operated on (which tells it with what to produce the result)
- These two influences form two paths into the processor logic

Part 1

Controlling program execution

Controlling control

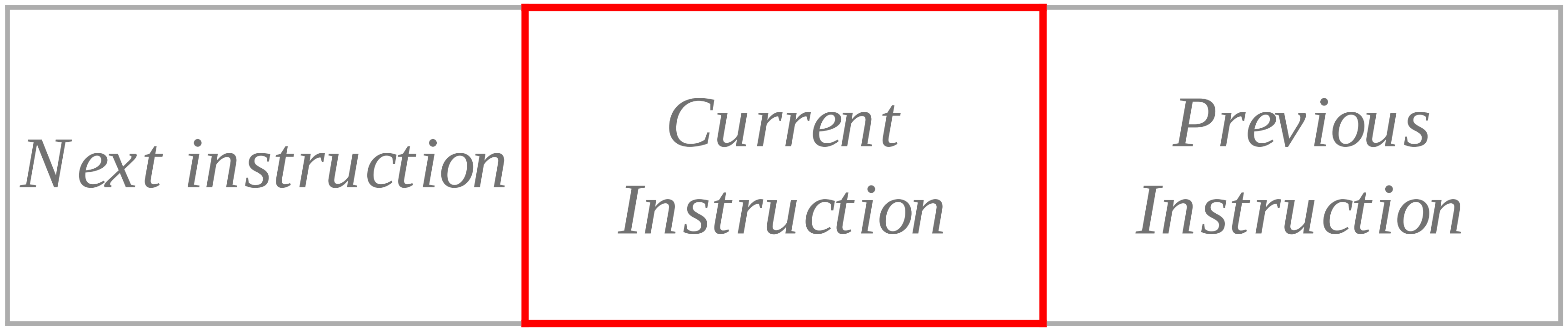
- The main source for control flow in a modern processor is the ***program counter (PC)***
- The PC causes things to ‘happen’ by being the source of context for the processor.
- It also provides the address for instruction fetches.

The Program Counter (PC)

- The program counter is a register
- It contains a number, normally equal to the word-length of the processor.
- The number is interpreted as an address.
- It normally increments by one every cycle of execution.

Execution

- We can model a series of steps of execution as a *sequence of instructions*.



Next instruction

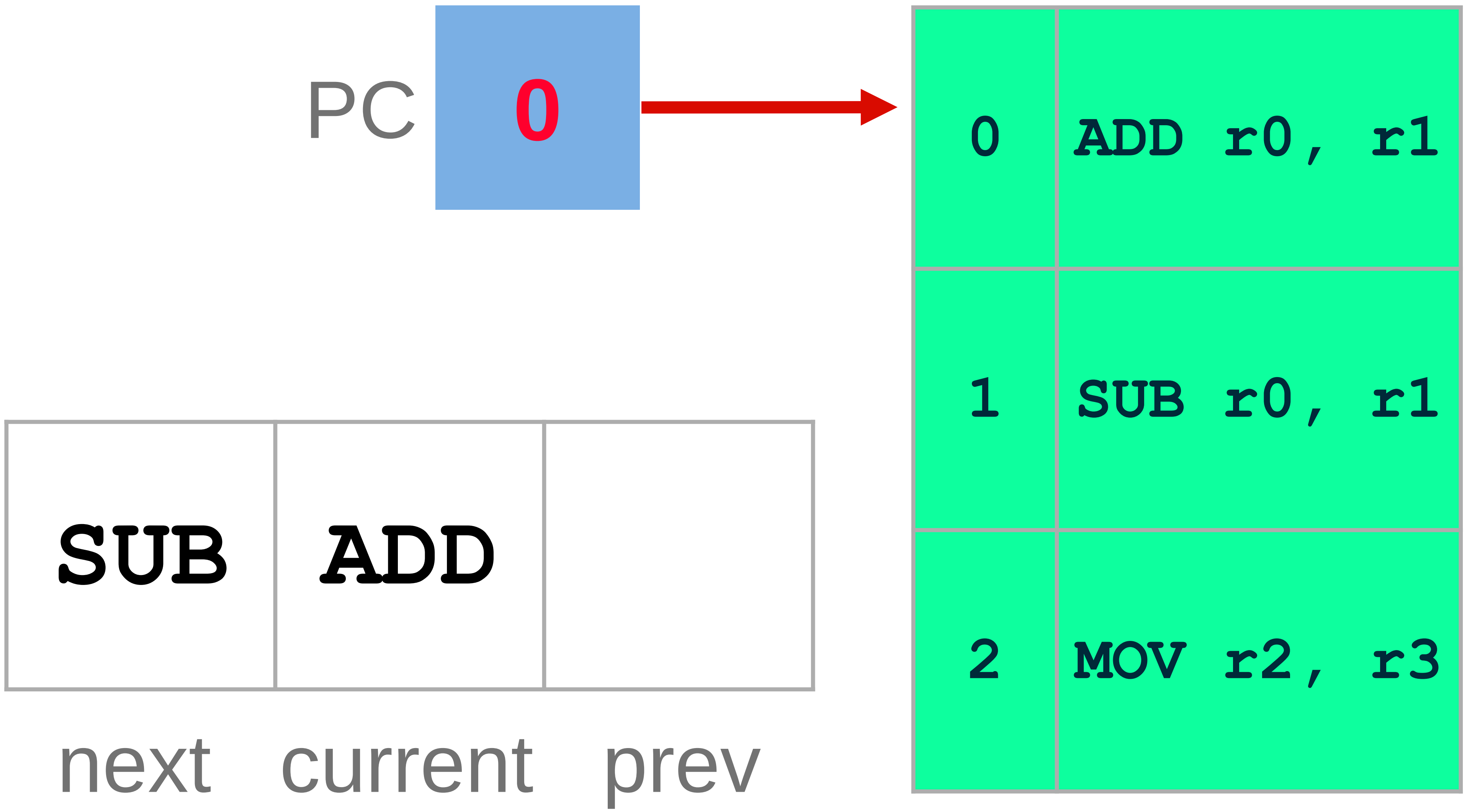
Current Instruction

Previous Instruction

  - The sequence can be real or conceptual.
  - Modelling the sequence is useful for seeing instruction effects and data dependencies (more on this soon)

PC by default

- Under normal program flow, the PC increments by one instruction address per instruction fetched.



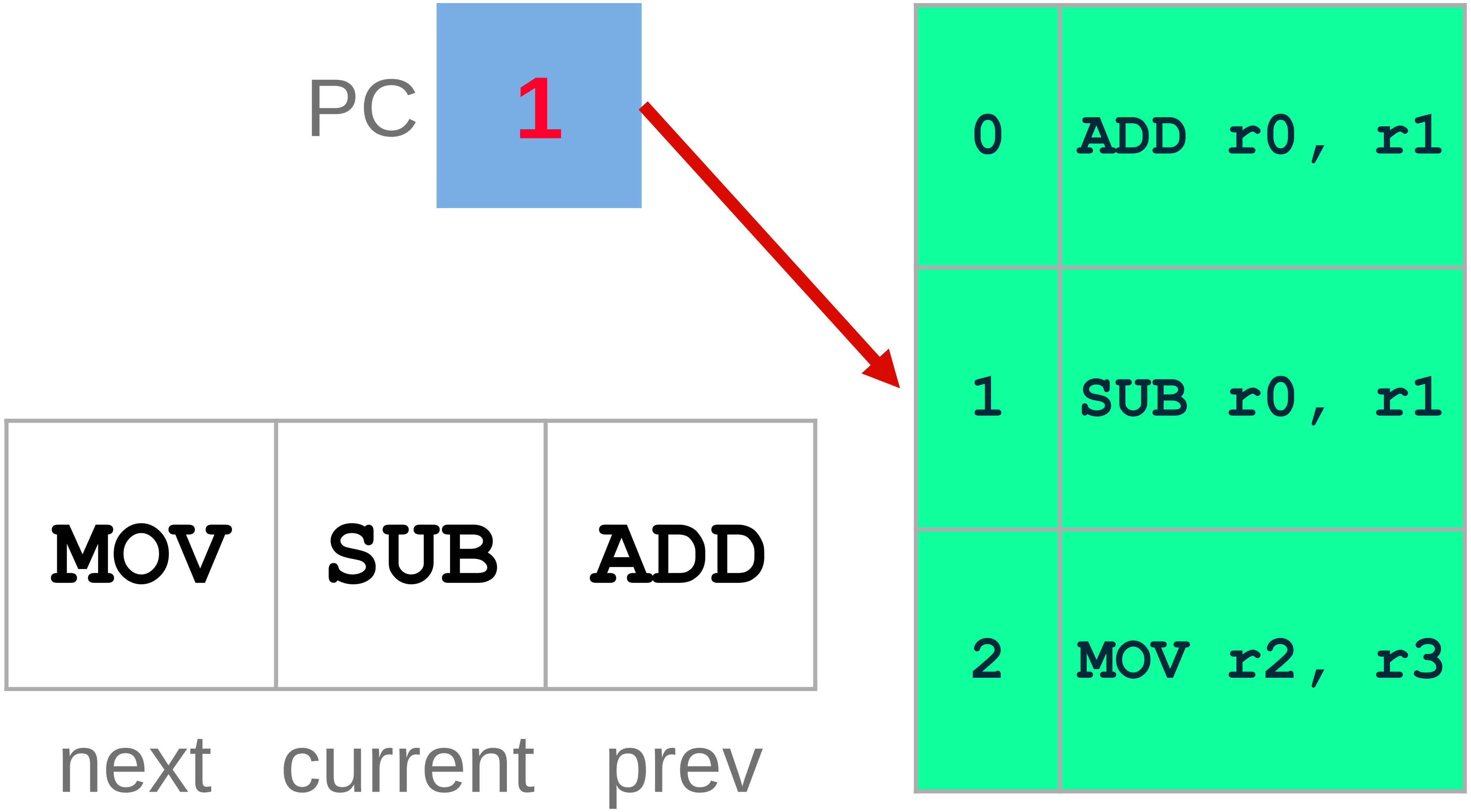
PC 0

0	ADD r0, r1
1	SUB r0, r1
2	MOV r2, r3

next	current	prev
SUB	ADD	

PC by default

- Under normal program flow, the PC increments by one instruction address per instruction fetched.



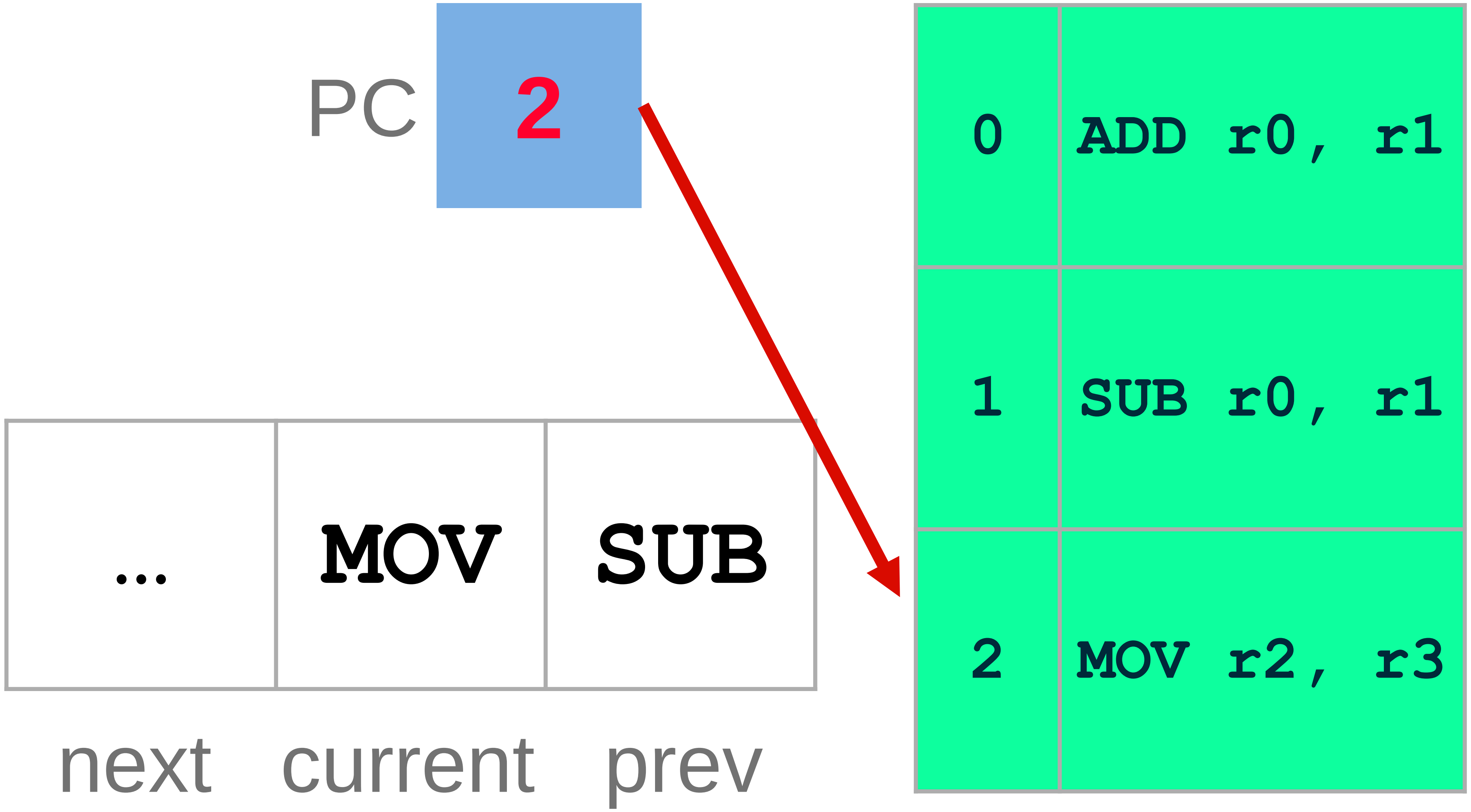
PC 1

0	`ADD r0, r1`
1	`SUB r0, r1`
2	`MOV r2, r3`

MOV	SUB	ADD
next	current	prev

PC by default

- Under normal program flow, the PC increments by one instruction address per instruction fetched.

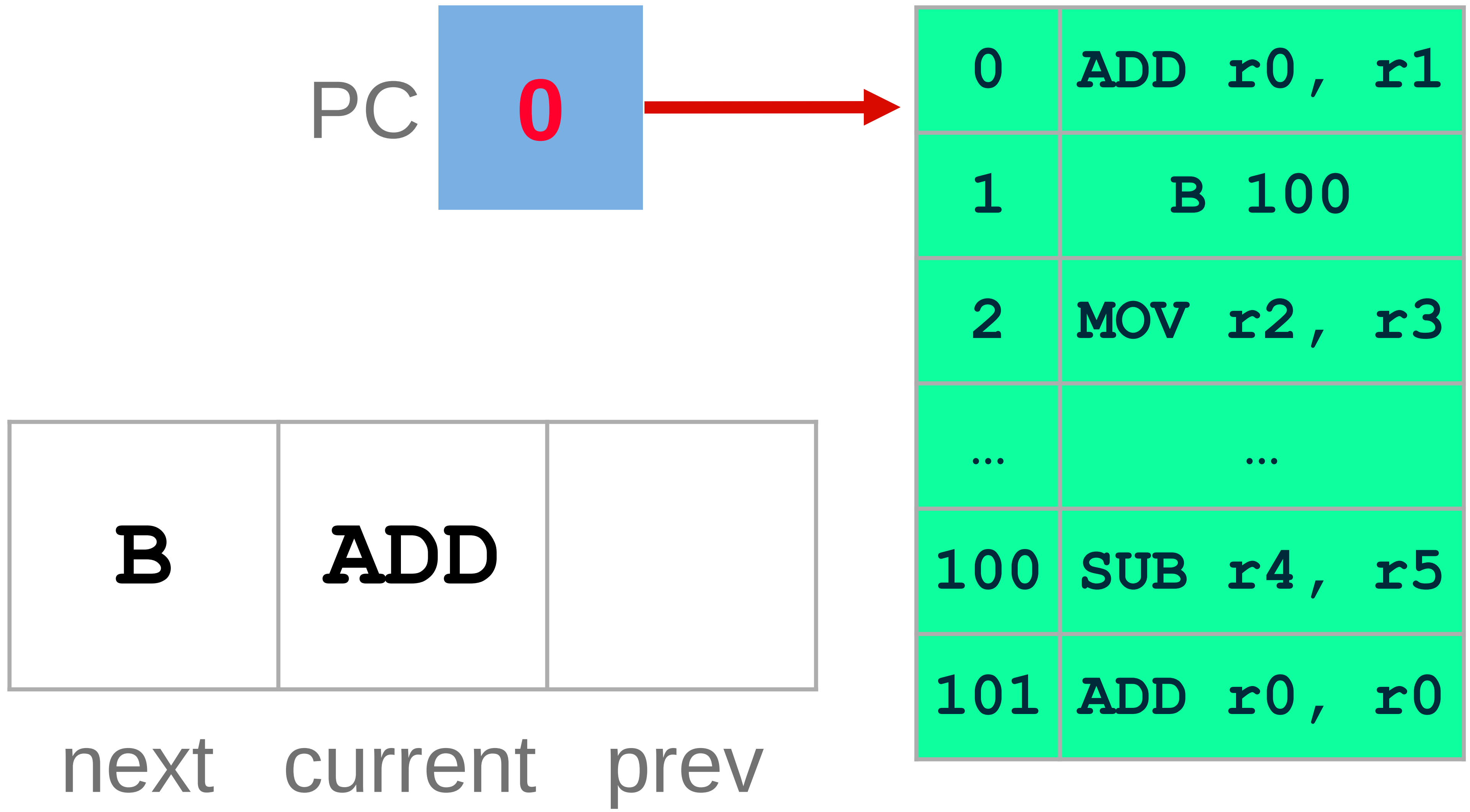


0	ADD r0, r1
1	SUB r0, r1
2	MOV r2, r3

next	current	prev
…	MOV	SUB

Taking a branch

- Under normal program flow, the PC increments by one instruction address per instruction fetched.



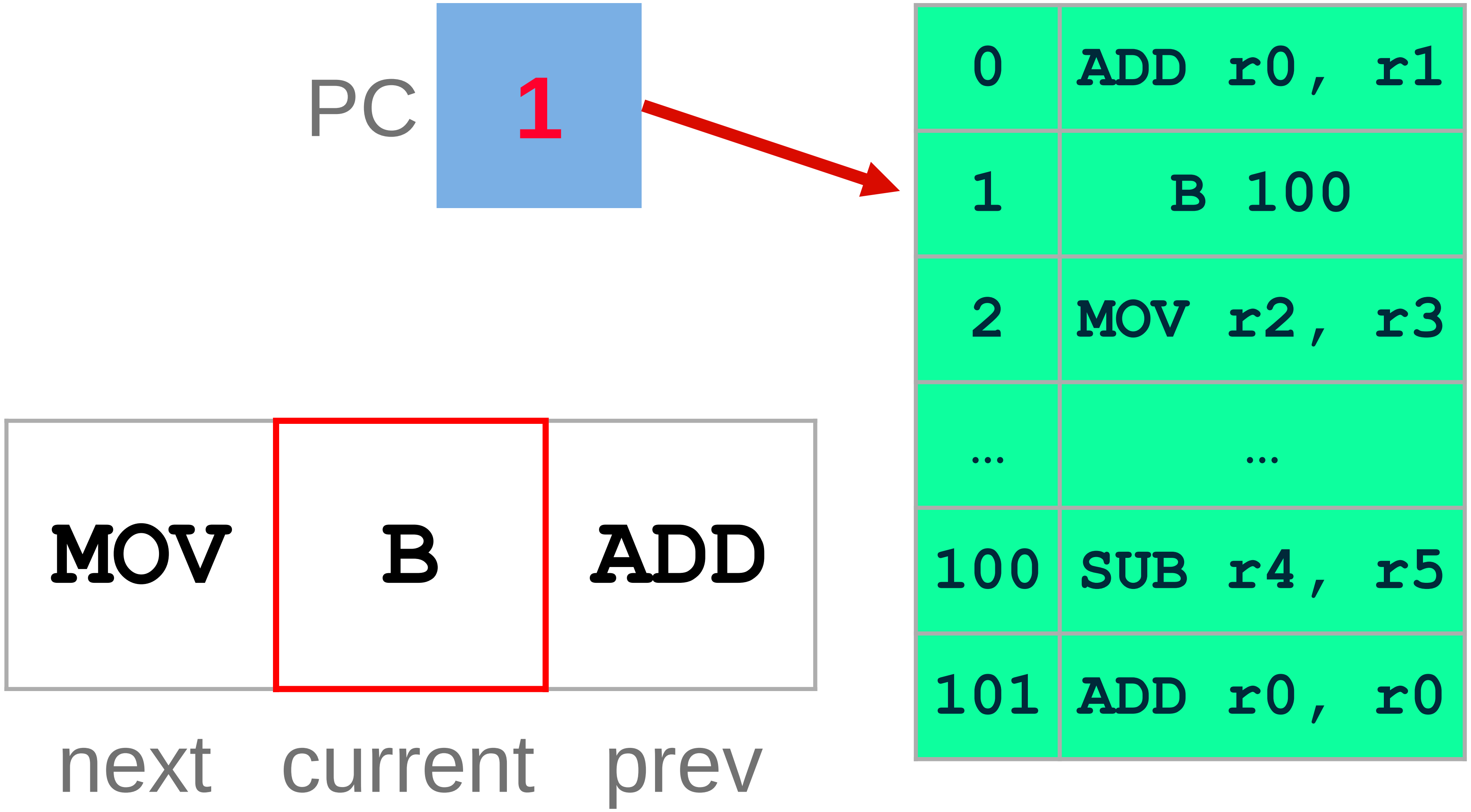
PC 0

0	`ADD r0, r1`
1	`B 100`
2	`MOV r2, r3`
…	…
100	`SUB r4, r5`
101	`ADD r0, r0`

next	current	prev
`B`	`ADD`	

Taking a branch

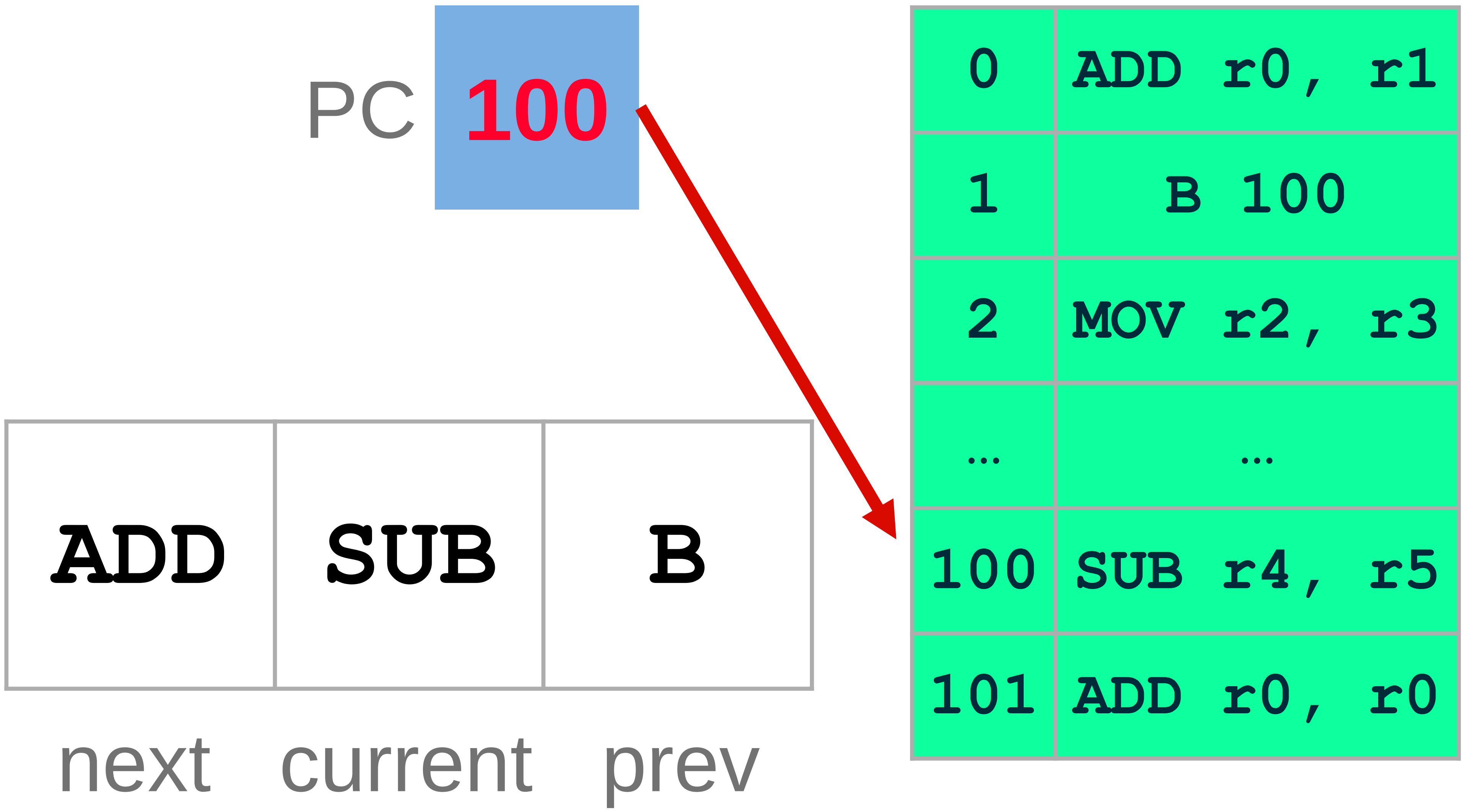
- Under normal program flow, the PC increments by one instruction address per instruction fetched.



0	ADD r0, r1
1	B 100
2	MOV r2, r3
…	…
100	SUB r4, r5
101	ADD r0, r0

Taking a branch

- Under normal program flow, the PC increments by one instruction address per instruction fetched.

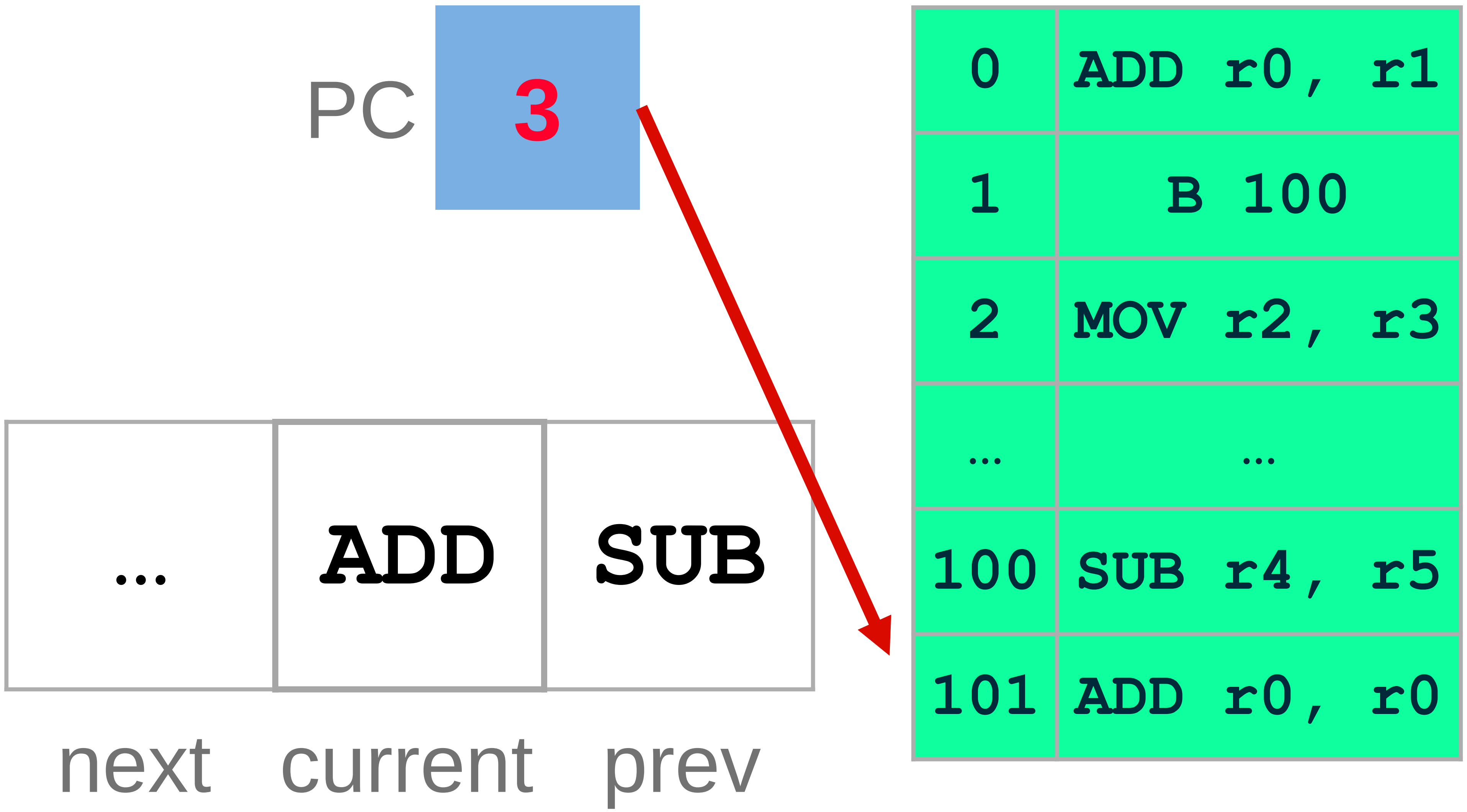


ADD	SUB	B
next	current	prev

0	ADD r0, r1
1	B 100
2	MOV r2, r3
…	…
100	SUB r4, r5
101	ADD r0, r0

Taking a branch

- Under normal program flow, the PC increments by one instruction address per instruction fetched.



PC 3

0	ADD r0, r1
1	B 100
2	MOV r2, r3
…	…
100	SUB r4, r5
101	ADD r0, r0

next	current	prev
…	ADD	SUB

Conditional control flow

- We can make the control flow conditional on data values
  - These values are normally those previously seen by the processor.
  - Can also be based on operands to the conditional instructions (architecture dependent)

Conditional branching

- Most often, we make branches the target of conditional execution
  - Allows us to redirect the flow of a program at run-time.
  - Introduces ***non-determinism*** in the program
    - Increases the programmer’s power :)
    - Explodes the complexity of analysis :(

Conditional example

- C code: `If (a == 0) then a = 1 else a = 2`
- Accumulator machine code

If	then	else
`LOAD a`	`MOVE 1`	`MOVE 2`
`COMPARE 0`	`STORE a`	`STORE a`
`BIFEQ then`	`B end`	`B end`
`BIFNE else`		

Condition codes

- How do the tests for the branches work?
  - Where does the information come from?
  - There was no argument supplied to the instruction.
- The information comes from the *condition code* or *flag* register

The flag register

- Architecture specific, but normally contains most of the following:

Mnemonic	**EQ**	**Z**	**N**	**C**
Meaning	**Equal**	**Zero**	**Negative**	**Carry out**

- Most other conditions can be synthesised from a combination of the above

Summary

- We have seem how basic control flow is handled in processors, in the form of *branches*.
- We have also see how control flow behaviour may be made non-deterministic or data-controlled, using *conditional branching*.
- Next time, we will look at more complex control flow and support for procedure calls / sub-routines.