Hardware and Software Laboratory Project 3 (Hardware)

Implementation of the SIMPLE Architecture Processor

Hardware and Software Laboratory Project 3

In Charge of Hardware

Computer Science Course, School of Informatics and Mathematical Science, Faculty of Engineering, Kyoto University

2020/04 rev

Experiment 3
Hardware Details and Objectives

- Details
  - Microprocessor system design, logic design
  - Run application programs on an FPGA
- Objectives
  - Understand the operating principles behind processors
  - Learn about circuit design, optimization, and running tests
  - Get hands-on learning with various extension methods and optimization techniques for processors
- References
  - Shinji Tomita, Hiroshi Nakashima: Computer Hardware
  - D.A. Patterson and J.L. Hennessy (authors), Mitsuaki Narita (translation): Computer Organization and Design 1 and 2, etc.

SIMPLE Processor Architecture

Overview of SIMPLE

➠ <u>SI</u>xteen-bit <u>M</u>icro<u>P</u>rocessor for <u>L</u>aboratory <u>E</u>xperiment

- 😐Simple instruction set
- ☺Equipped with essentially all the basic functionalities

- Characteristics
  - 16-bit fixed length instructions
  - 8 general purpose registers
  - 16-bit × 64 K word main memory
  - Load/store architecture
  - 2 operand format instruction set (Rd op Rs -> Rd)

Architecture Description

- Architecture
  - Computer configuration
    - Processor, memory, I/O, etc.
  - Configuration of the main memory and registers will be included here
- Instruction set architecture
  - Instruction configuration
  - The aforementioned load/store architecture is one of the instruction set formats
- Microarchitecture
  - Implementation of the architecture at the circuit level

Main memory and register

Components that represent the status of the computer

1. Main memory
   - 16-bit × 64 K words (word address format)
   - However, the maximum size that can be ensured on the FPGA used in this experiment is about 33 K words
2. General purpose register
   - 16-bit × 8 words
3. Program counter (PC)
   - 16bit
4. Condition codes
   - S Sign
   - Z Zero
   - C Carry
   - V Overflow

Instruction Set

Components that change the state of the computer

1. Operation instructions
    - Arithmetic logic operation instructions
    - Shift instructions
2. Load/store instructions
3. Branch instruction
    - Unconditional branch instruction
    - Conditional branch instruction
4. Other
    - Input-output instruction
    - Stop instruction

Operation instructions

- Arithmetic logic operation instructions
  - **r[Rd] = r[Rd] op3 r[Rs]**

- Shift instructions
  - **r[Rd] = shift_op3(r[Rd], d)**
- Note: sets condition codes after running

11	Rs	Rd	op3	D

15 13 10 7 3 0

Load/Store Instruction (1)

- Load instruction **(op1 : 00)**
  - **r[Ra] = *(r[Rb] + sign_ext(d))**

- Store instruction **(op1 : 01)**
  - ***(r[Rb] + sign_ext(d)) = r[Ra]**

op1	**Ra**	**Rb**	**d**	
15	13	10	7	0

Load/Store Instruction (2)

- Load immediate instruction
  - **r[Rb] = sign_ext(d)**

- Any 16-bit value can be stored to the register using two load immediate instructions and a shift instruction

10	**001**	**Rb**	**d**

15 13 10 7 0

Branch Instruction (1)

- Unconditional branch instruction (B: Branch)
  - **PC = PC + 1 + sign_ext(d)**

10	**100**		**d**
15	13	10	7 0

Branch Instruction (2)

- Conditional branch instruction
  - **if (cond) PC = PC + 1 + sign_ext(d)**
  - Branches according to the condition code
    - Condition codes are set when executing the operation instruction

10	**111**	**cond**	**d**

15 13 10 7 0

Other Instructions

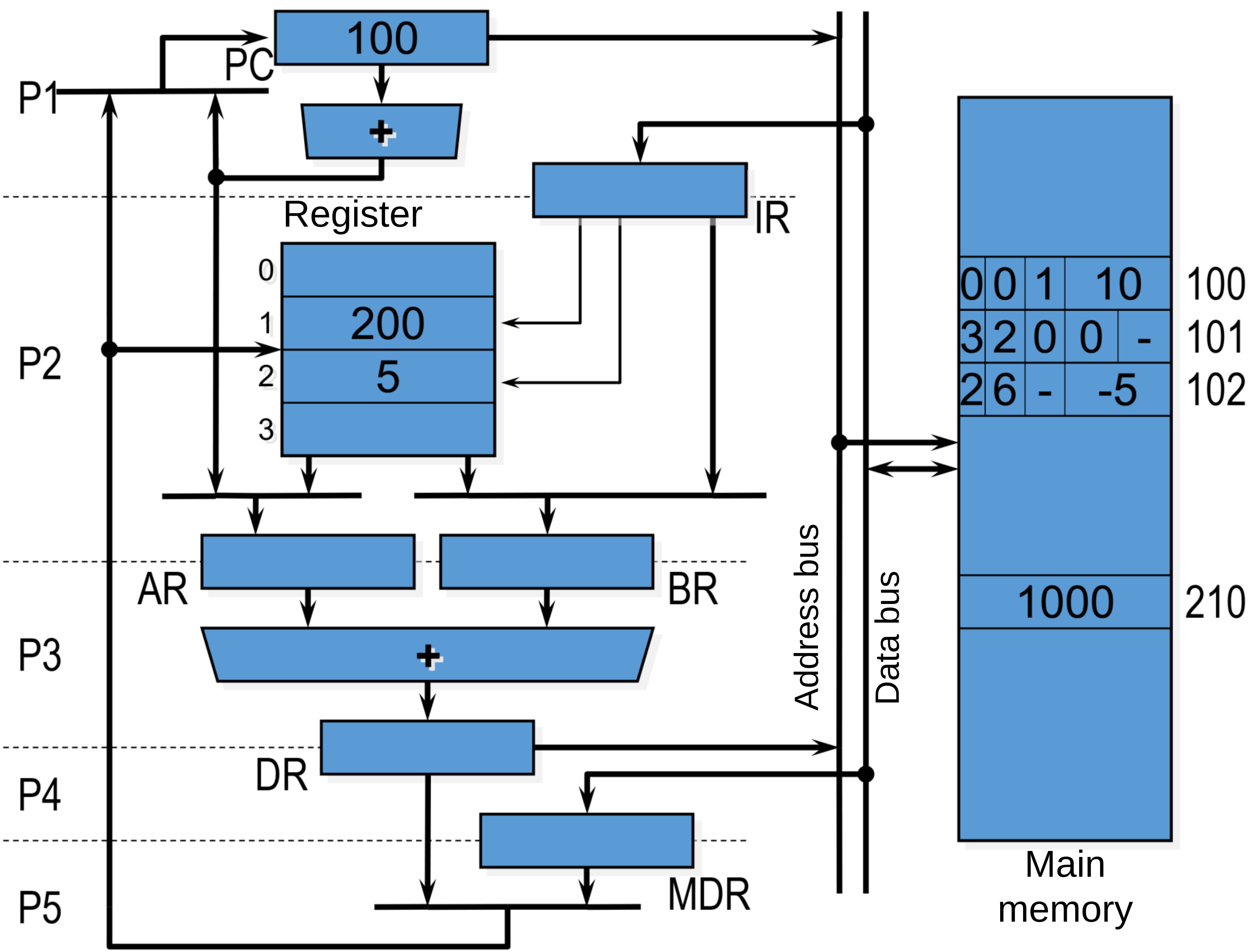
- Stop instruction **(op3 : 1111)**
- Input instruction **(op3 : 1100)**
  - **r[Rd] = input**
  - Input destination is switches on the board, etc.
- Output instruction **(op3 : 1101)**
  - **output = r[Rs]**
  - Output destination is the board's LED/7SEG LED, etc.

11	**Rs**	**Rd**	**op3**	**d**

15 13 10 7 3 0

Basic Implementation SIMPLE/B

- Position the functional units, registers, and data buses as shown in the following slide
- Activate the 5 phases one after another as done with the sequential circuits in Experiment 2
  - P1 Instruction fetch
  - P2 Instruction decoding, register readout
  - P3 Operation
  - P4 Main memory access
  - P5 Register writing/PC updating
- The phases are activated by a controller
  - (Updates the register in which the data input into the phase is retained)
  - Switches between selectors within phases as appropriate
  - Updates the register in which the data output from the phase is retained



P1
PC
100
+
IR
Register
0
1
200
2
5
3
P2
AR
BR
P3
+
DR
P4
MDR
P5
Address bus
Data bus
0 0 1 10
3 2 0 0 -
2 6 - -5
1000
100
101
102
210
Main memory

Sample Instruction for Execution

- Load instruction: program counter 100
  - LD R0, 10(R1)

Abbreviation

0	0	1	10

00	Ra (000)	Rb (001)	d (00001010)

15 13 10 7 0

- Addition instruction: program counter 101
  - ADD R0, R2

Abbreviation

3	2	0	0	-

11	Rs (010)	Rd (000)	op3 (0000)	d

15 13 10 7 3 0

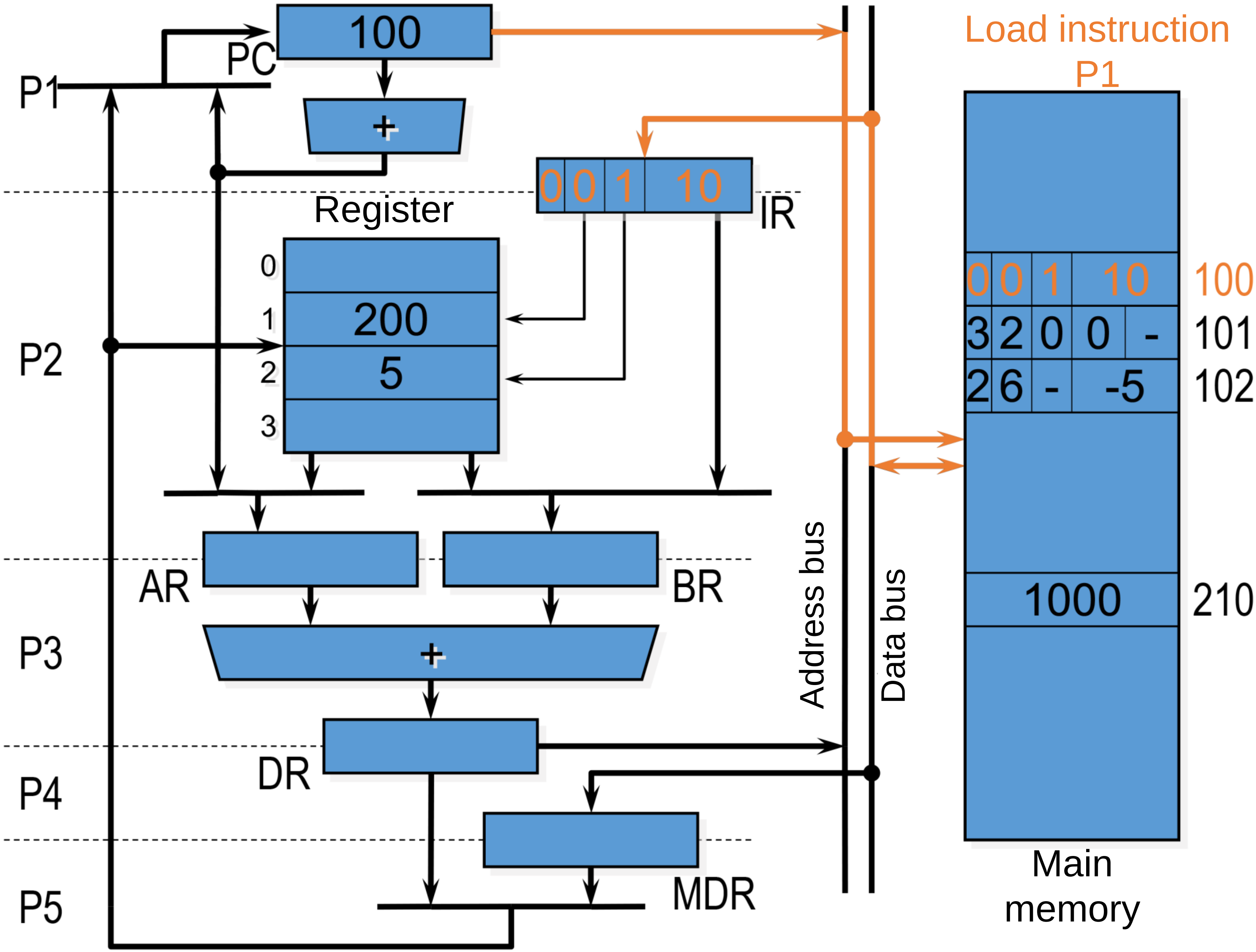
Sample Instruction for Execution

- Unconditional branch instruction: program counter 102
  - B -5

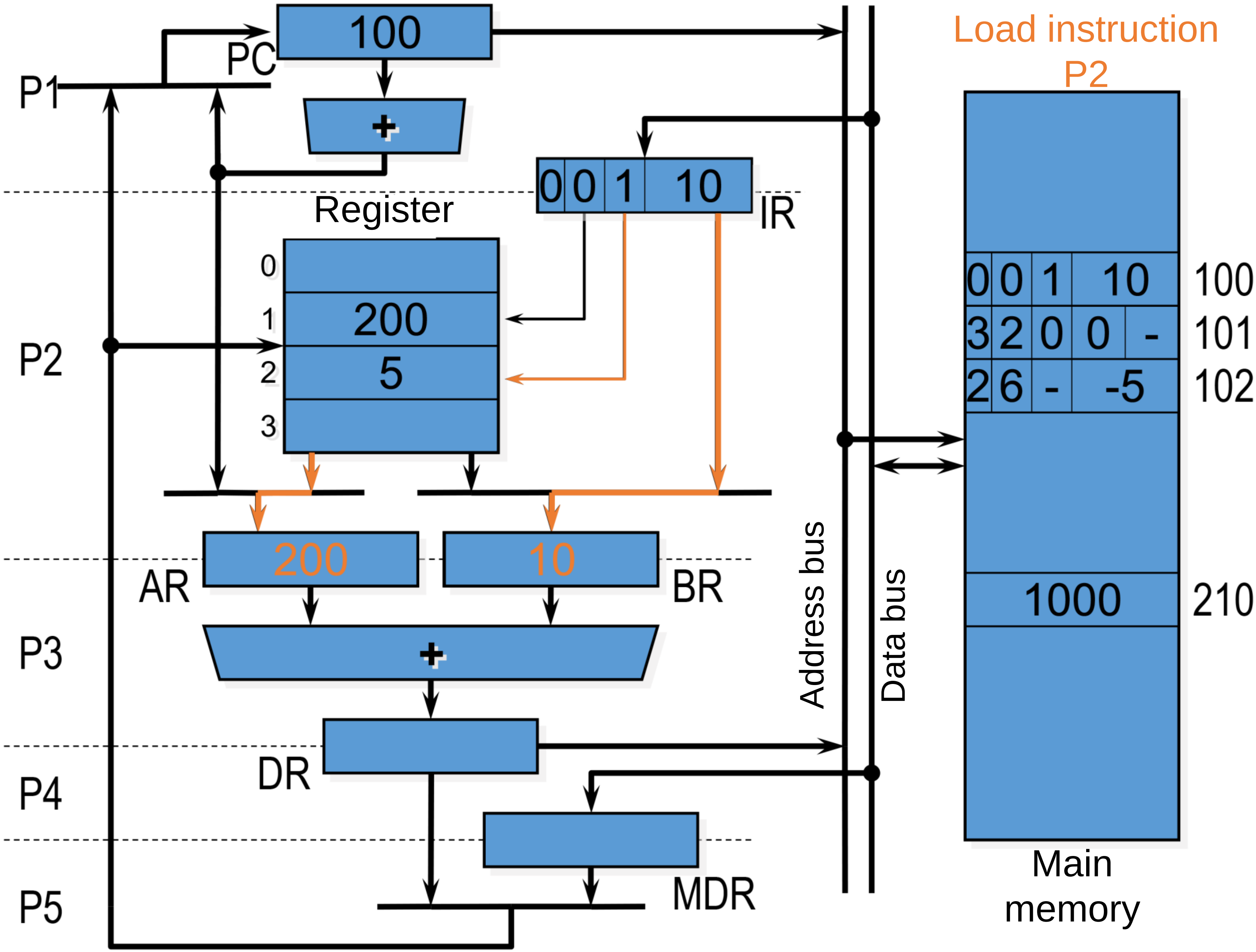
Abbreviation

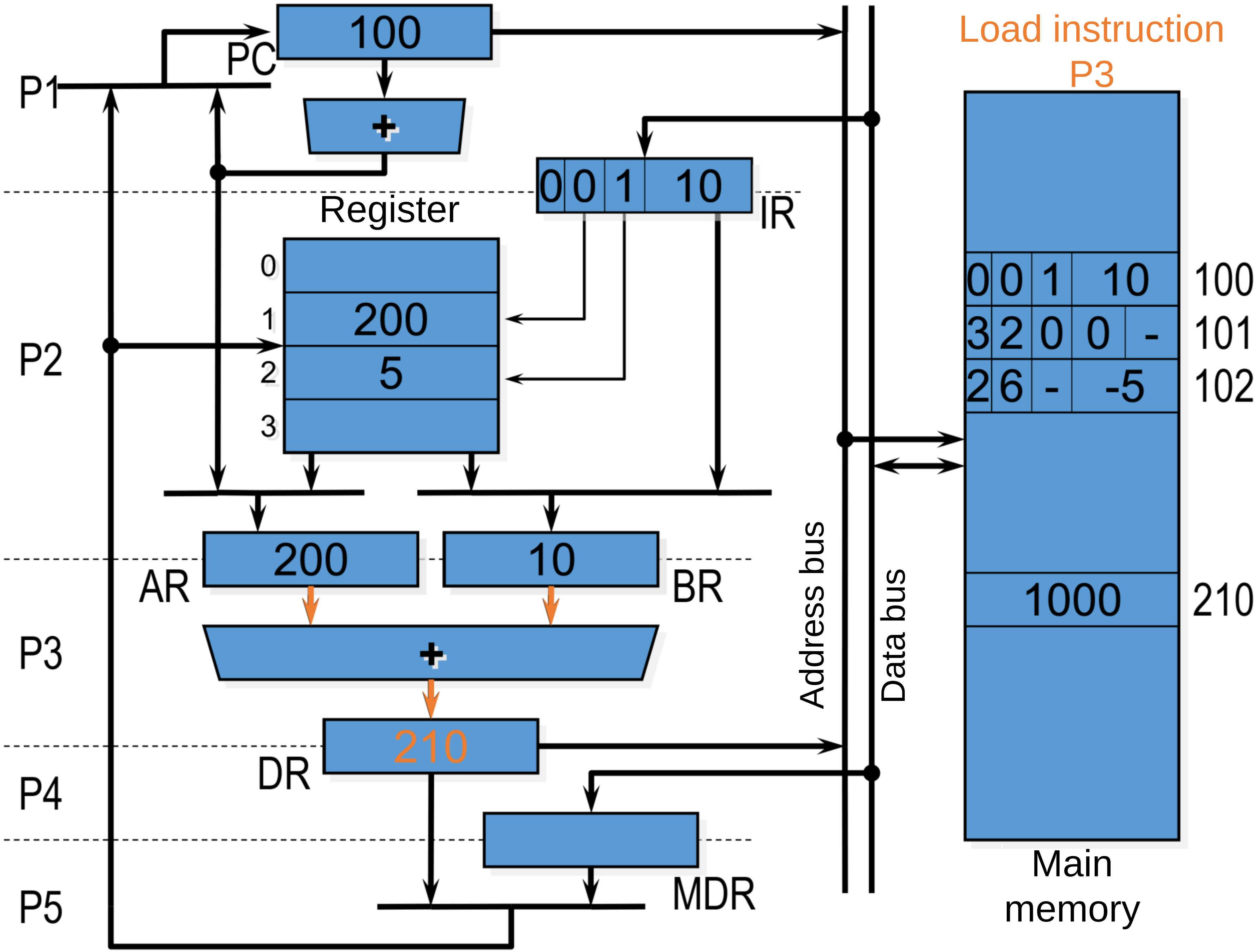
2	6	-	-5

10	**op2 (110)**		**d (11111011)**
15	13	10	7 … 0

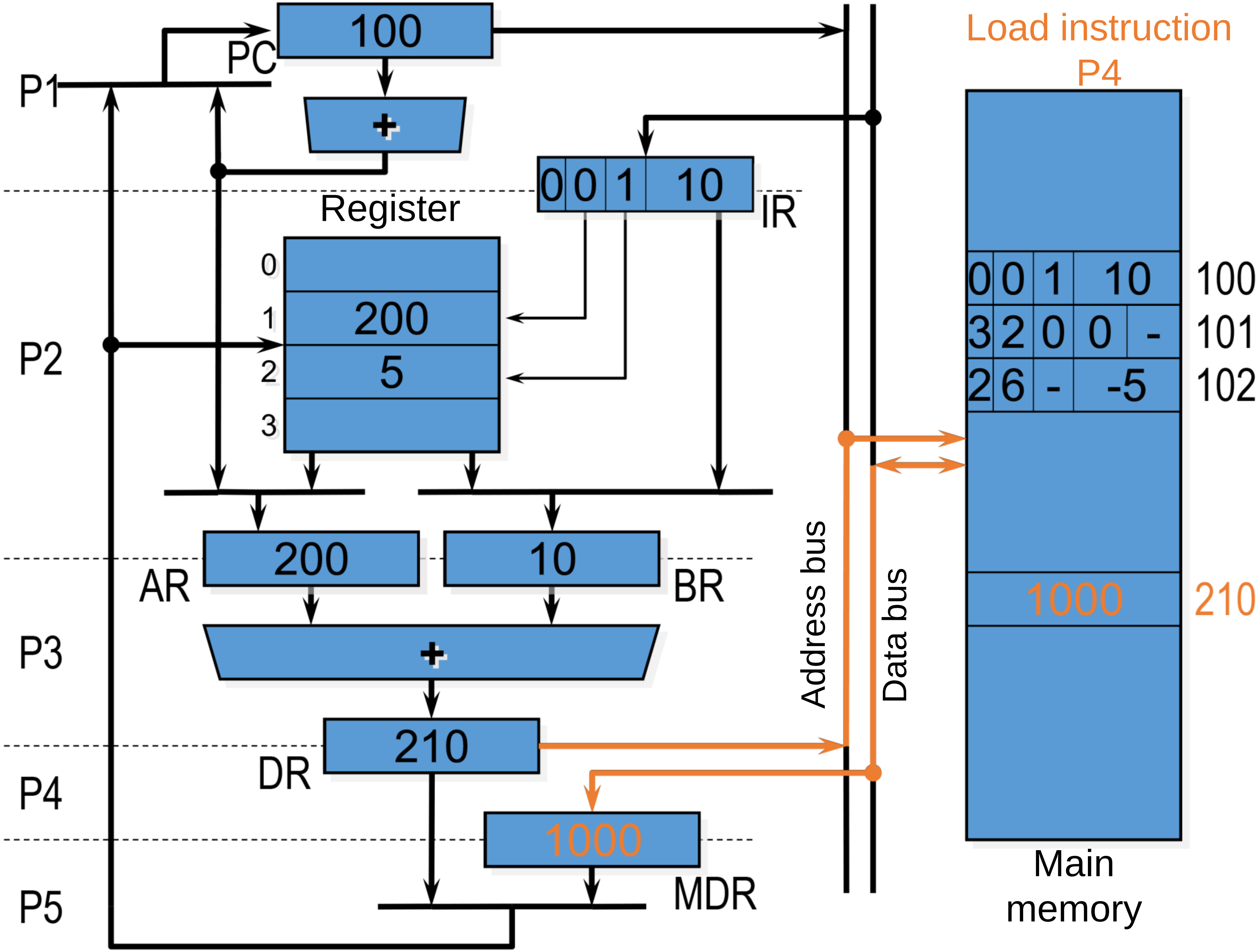


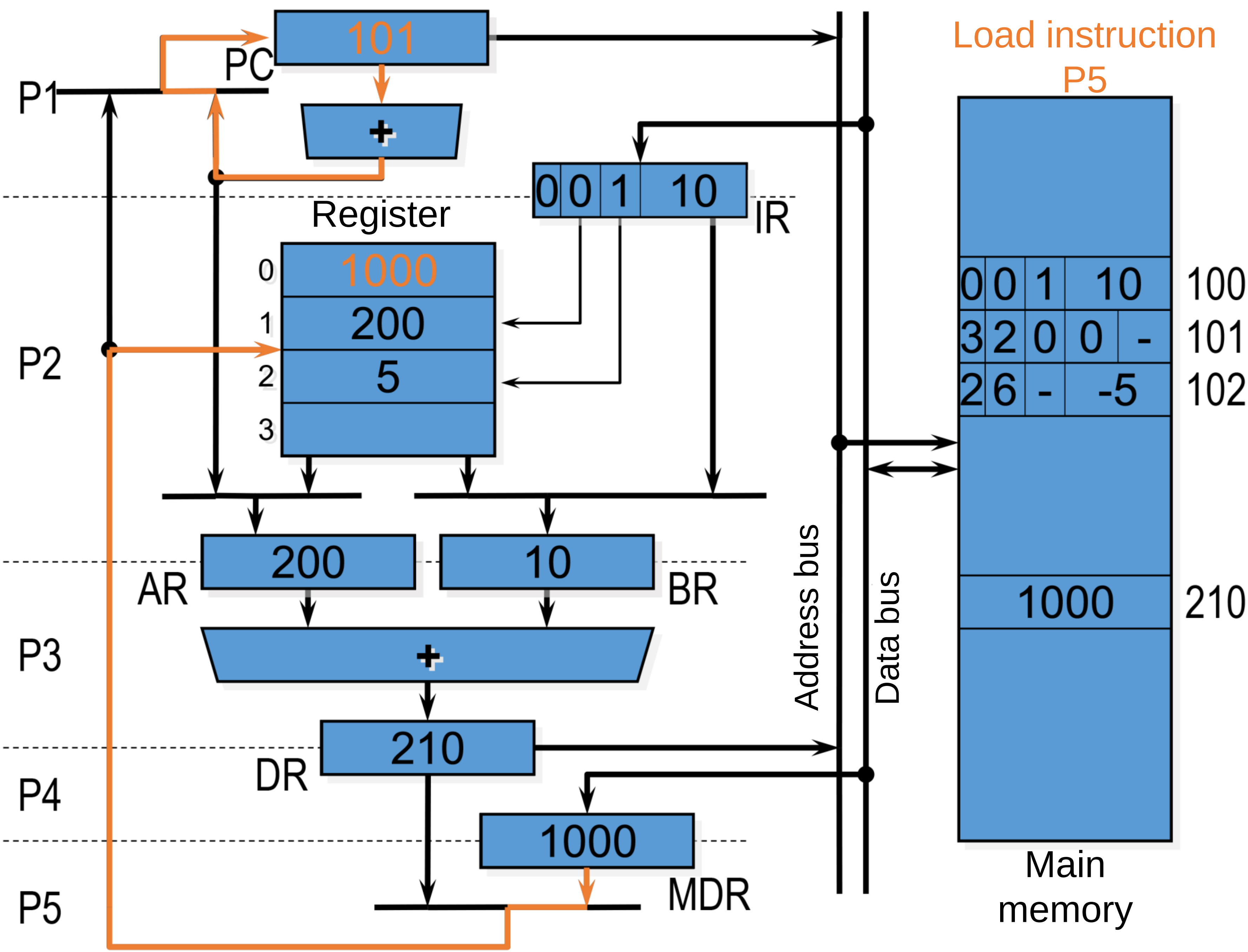
Load instruction
P1
PC
100
P1
+
Register
0
1
200
2
5
3
0 0 1 10
IR
P2
AR
BR
+
P3
DR
P4
MDR
P5
Address bus
Data bus
0 0 1 10
100
3 2 0 0 -
101
2 6 - -5
102
1000
210
Main
memory


Load instruction
P2
PC
100
P1
+
0 0 1 10
IR
Register
0
1
2
3
200
5
P2
AR
200
10
BR
P3
+
DR
P4
MDR
P5
Address bus
Data bus
0 0 1 10
3 2 0 0 -
2 6 - -5
100
101
102
1000
210
Main
memory

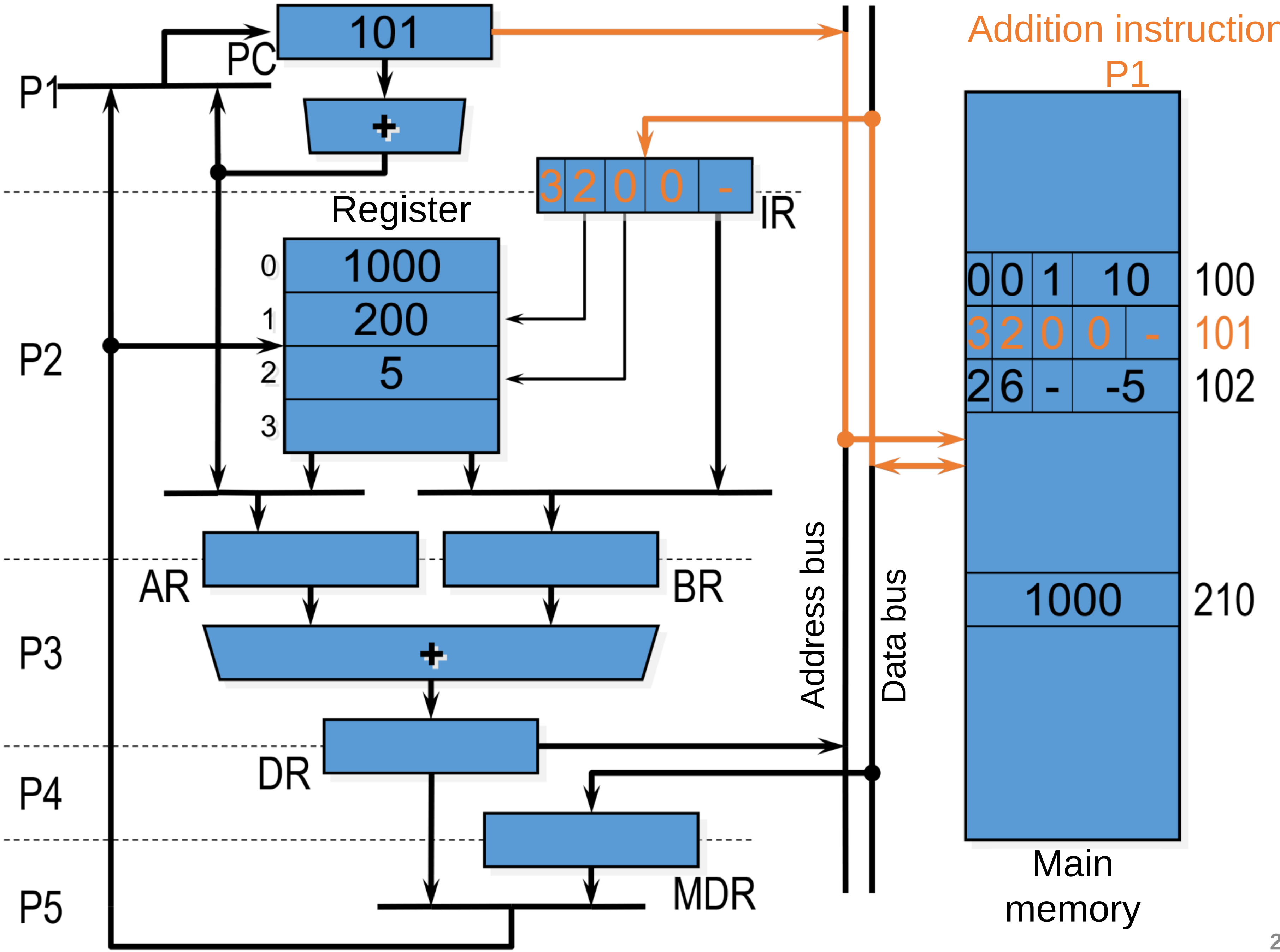


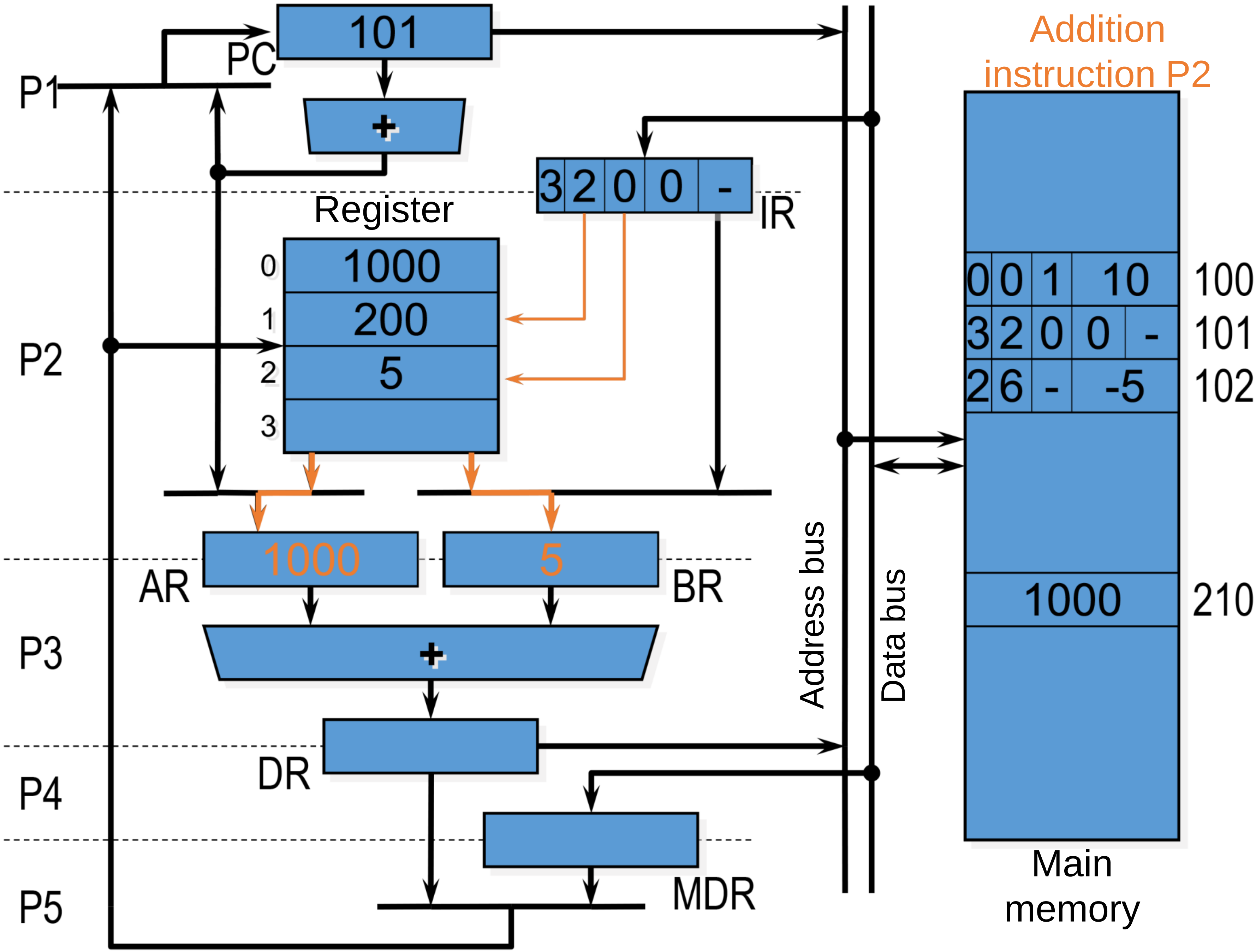
Load instruction
P3
100
PC
P1
+
0 0 1 10
IR
Register
0
1
2
3
200
5
P2
200
10
AR
BR
+
P3
210
DR
P4
MDR
P5
Address bus
Data bus
0 0 1 10
3 2 0 0 -
2 6 - -5
100
101
102
1000
210
Main
memory


Load instruction
P4
PC
100
P1
+
0 0 1 10
IR
Register
0
1 200
2 5
3
P2
AR
200
10
BR
P3
+
DR
210
P4
1000
MDR
P5
Address bus
Data bus
0 0 1 10 100
3 2 0 0 - 101
2 6 - -5 102
1000 210
Main
memory



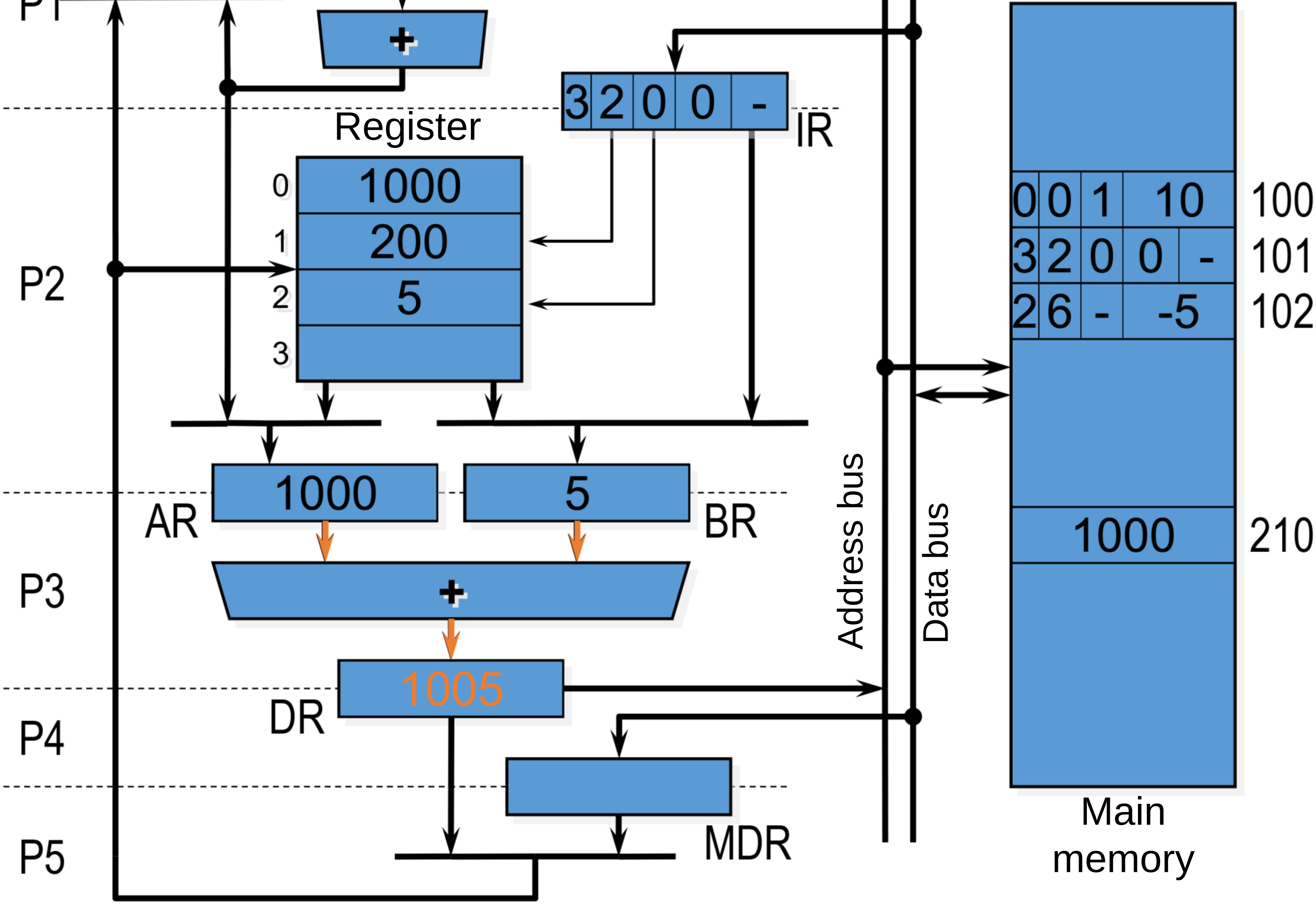
Load instruction
P5
101
PC
P1
+
0 0 1 10
IR
Register
0
1000
1
200
2
5
3
P2
0 0 1 10
100
3 2 0 0 -
101
2 6 - -5
102
200
AR
10
BR
P3
+
Address bus
Data bus
1000
210
210
DR
P4
1000
MDR
P5
Main
memory

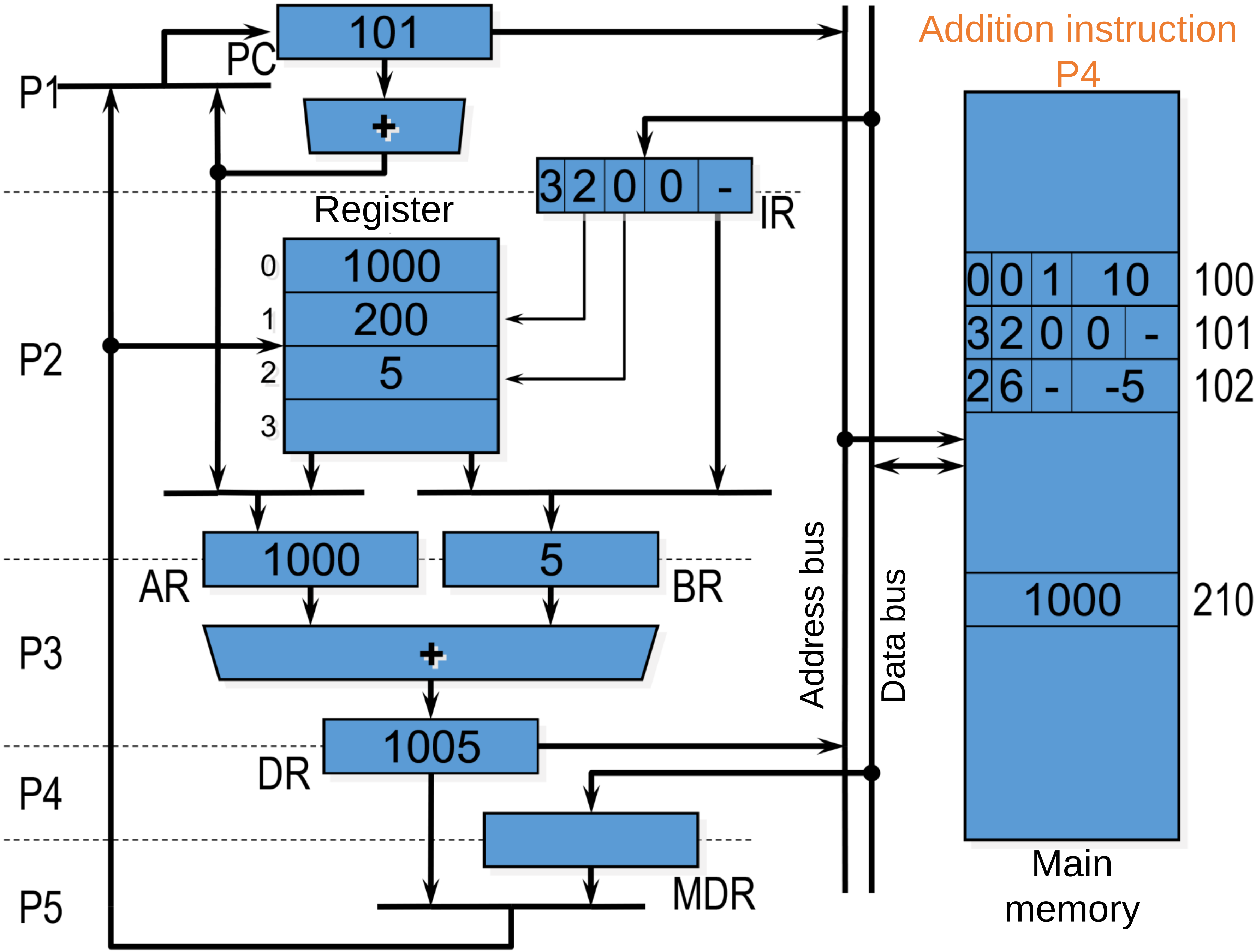

Addition instruction
P1
PC
101
P1
+
3 2 0 0 -
IR
Register
0
1000
1
200
2
5
3
P2
AR
BR
P3
+
DR
P4
MDR
P5
Address bus
Data bus
0 0 1 10
100
3 2 0 0 -
101
2 6 - -5
102
1000
210
Main memory



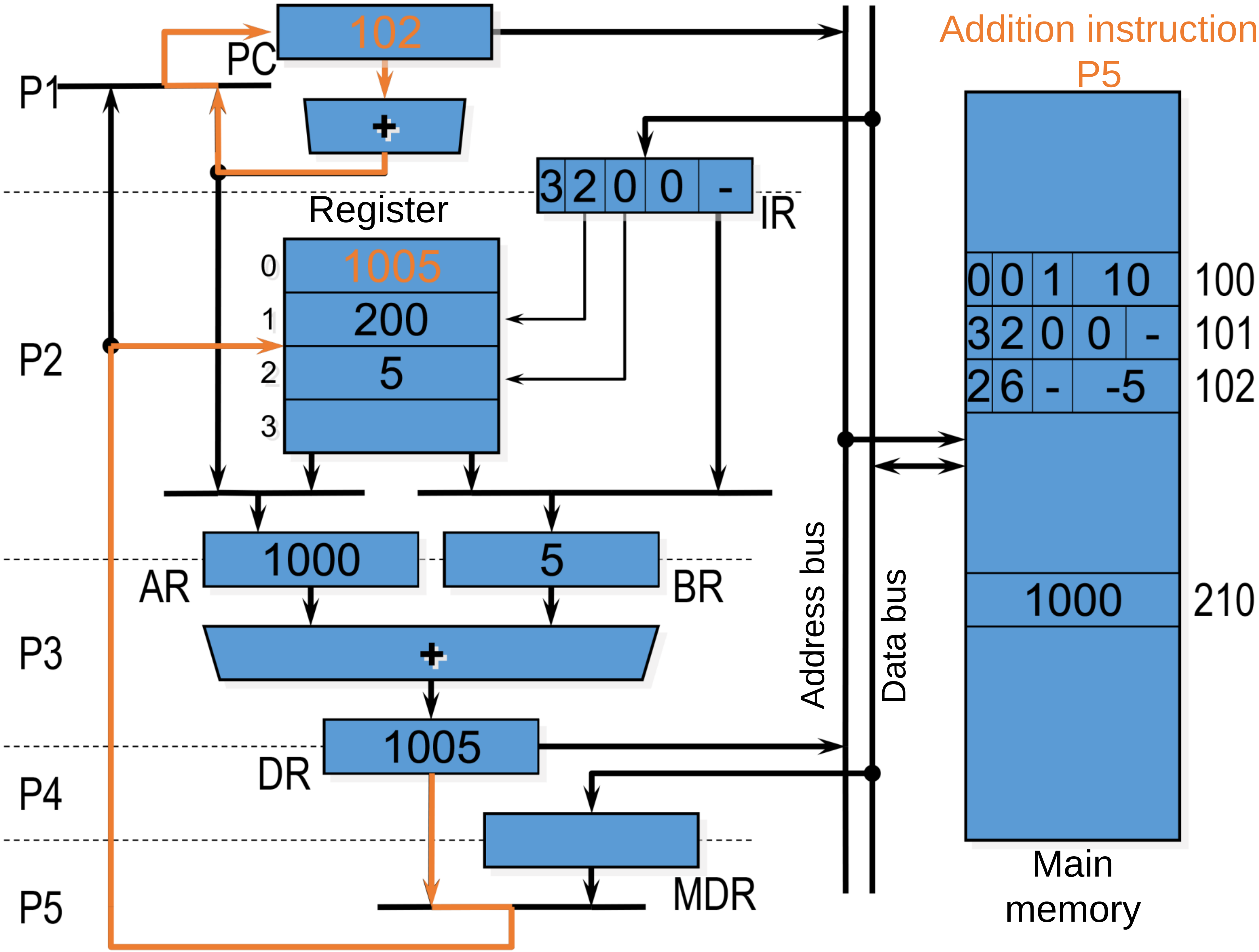
Addition instruction P2
P1
PC
101
+
3 2 0 0 -
IR
Register
0
1000
1
200
2
5
3
P2
AR
1000
5
BR
P3
+
DR
P4
MDR
P5
Address bus
Data bus
0 0 1 10
100
3 2 0 0 -
101
2 6 - -5
102
1000
210
Main memory

Addition instruction P3

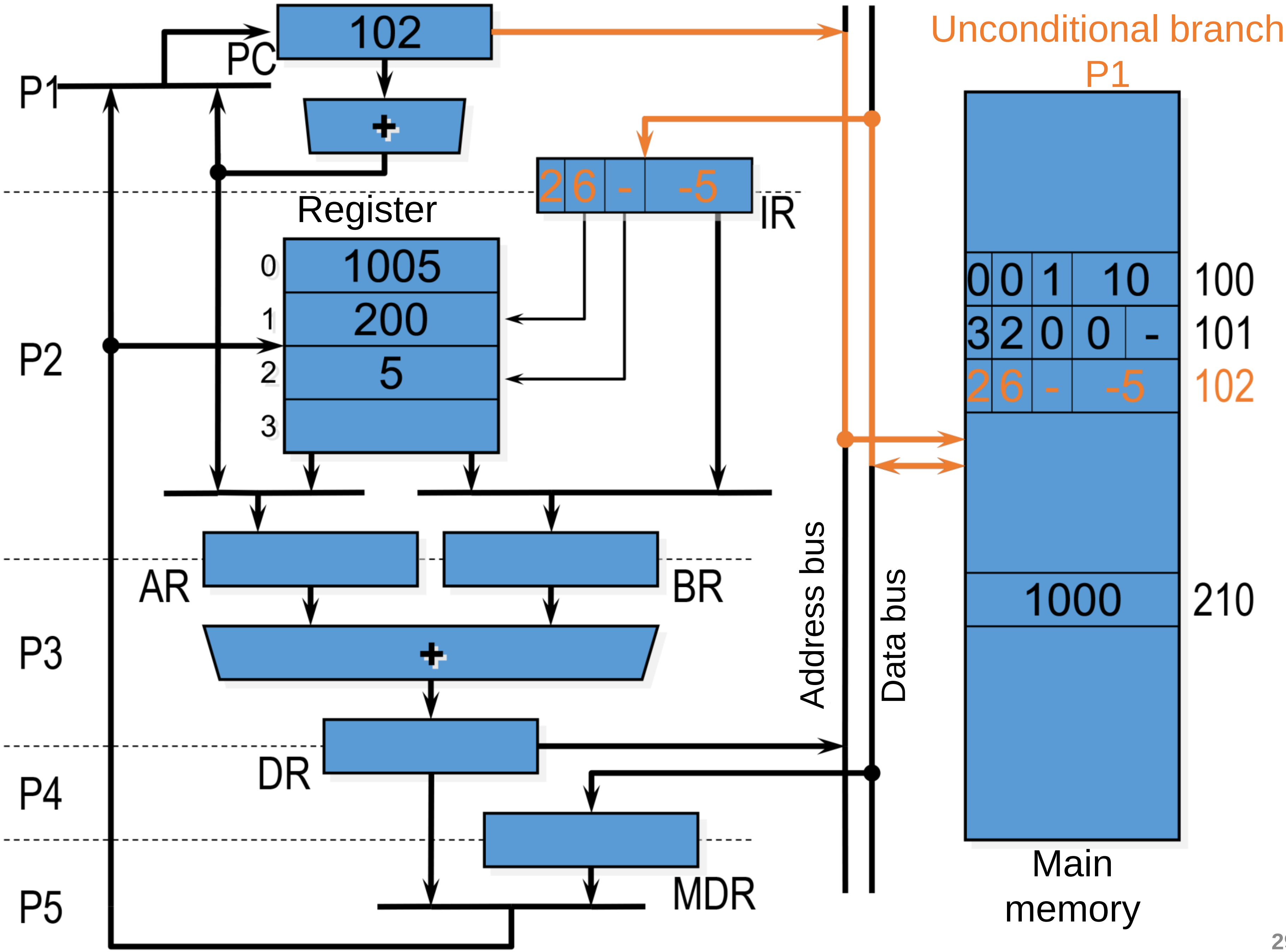




Addition instruction
P4
P1
PC
101
+
P2
Register
0
1000
1
200
2
5
3
3 2 0 0 -
IR
AR
1000
5
BR
P3
+
DR
1005
P4
MDR
P5
Address bus
Data bus
0 0 1 10
100
3 2 0 0 -
101
2 6 - -5
102
1000
210
Main
memory

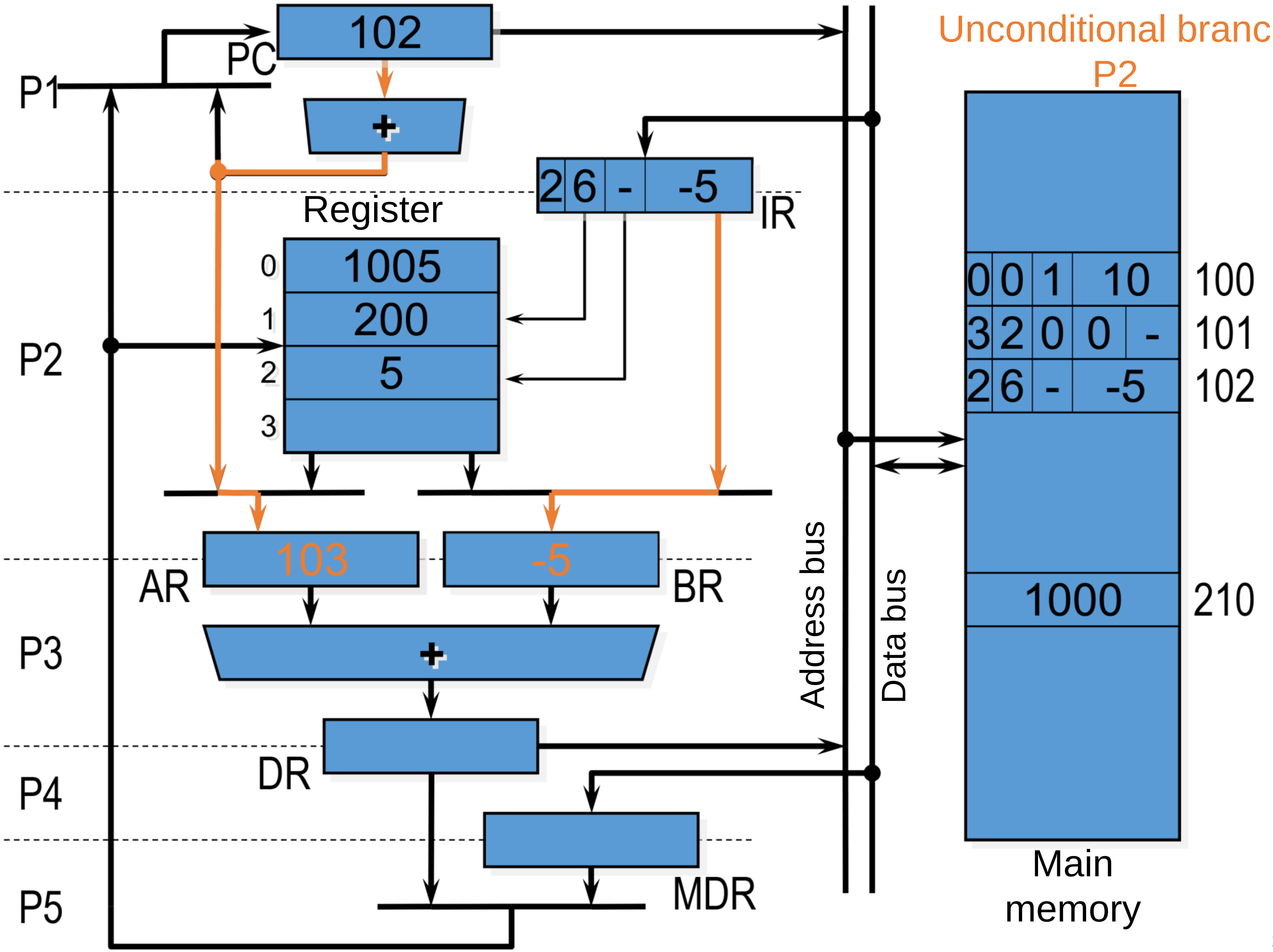


Addition instruction
P5
102
PC
P1
+
3 2 0 0 -
IR
Register
0
1005
1
200
2
5
3
P2
0 0 1 10
100
3 2 0 0 -
101
2 6 - -5
102
1000
AR
5
BR
Address bus
Data bus
1000
210
P3
+
1005
DR
P4
MDR
P5
Main
memory

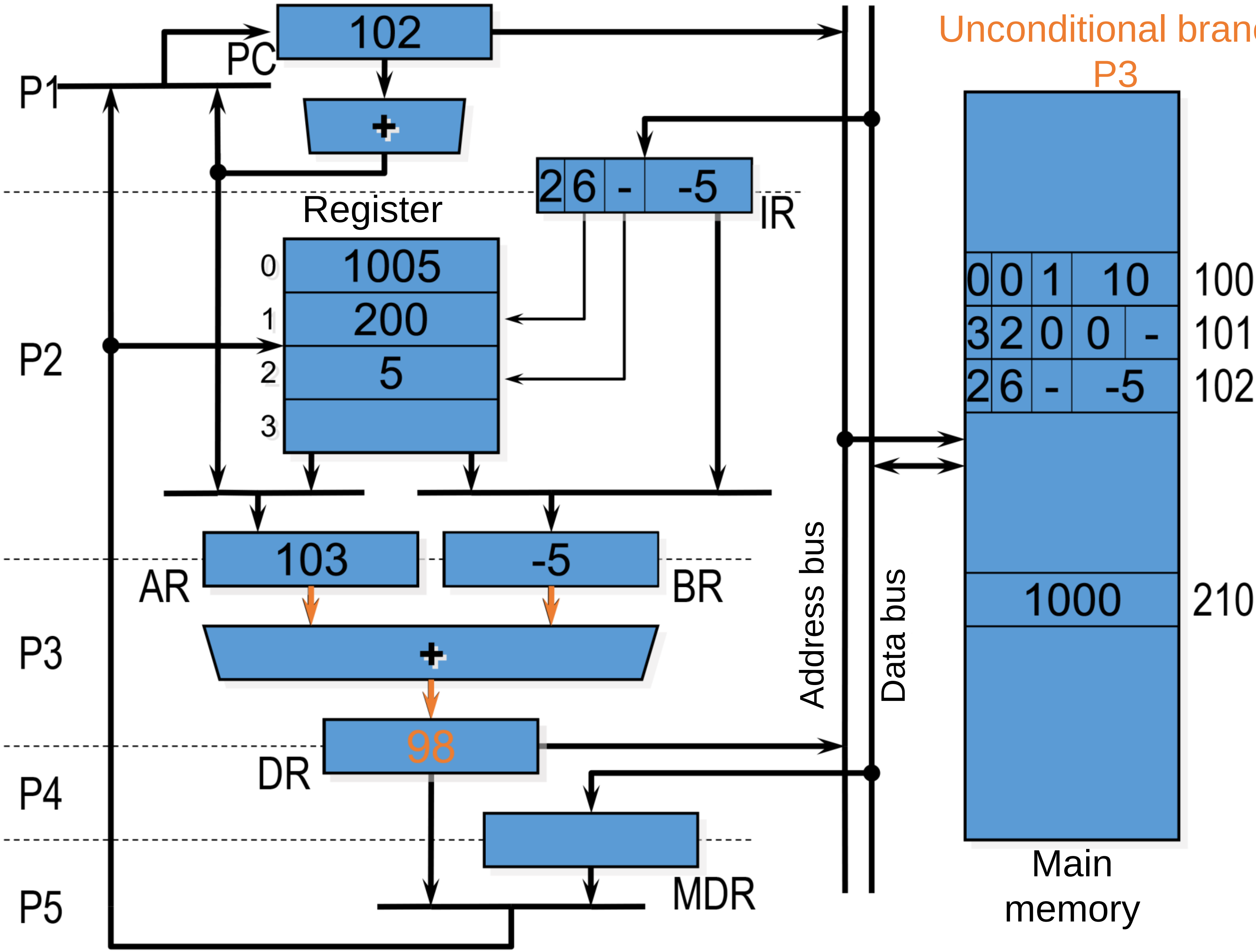


Unconditional branch
P1
102
PC
P1
P2
P3
P4
P5
Register
0
1
2
3
1005
200
5
2 6 - -5
IR
AR
BR
DR
MDR
Address bus
Data bus
0 0 1 10
3 2 0 0 -
2 6 - -5
1000
100
101
102
210
Main
memory

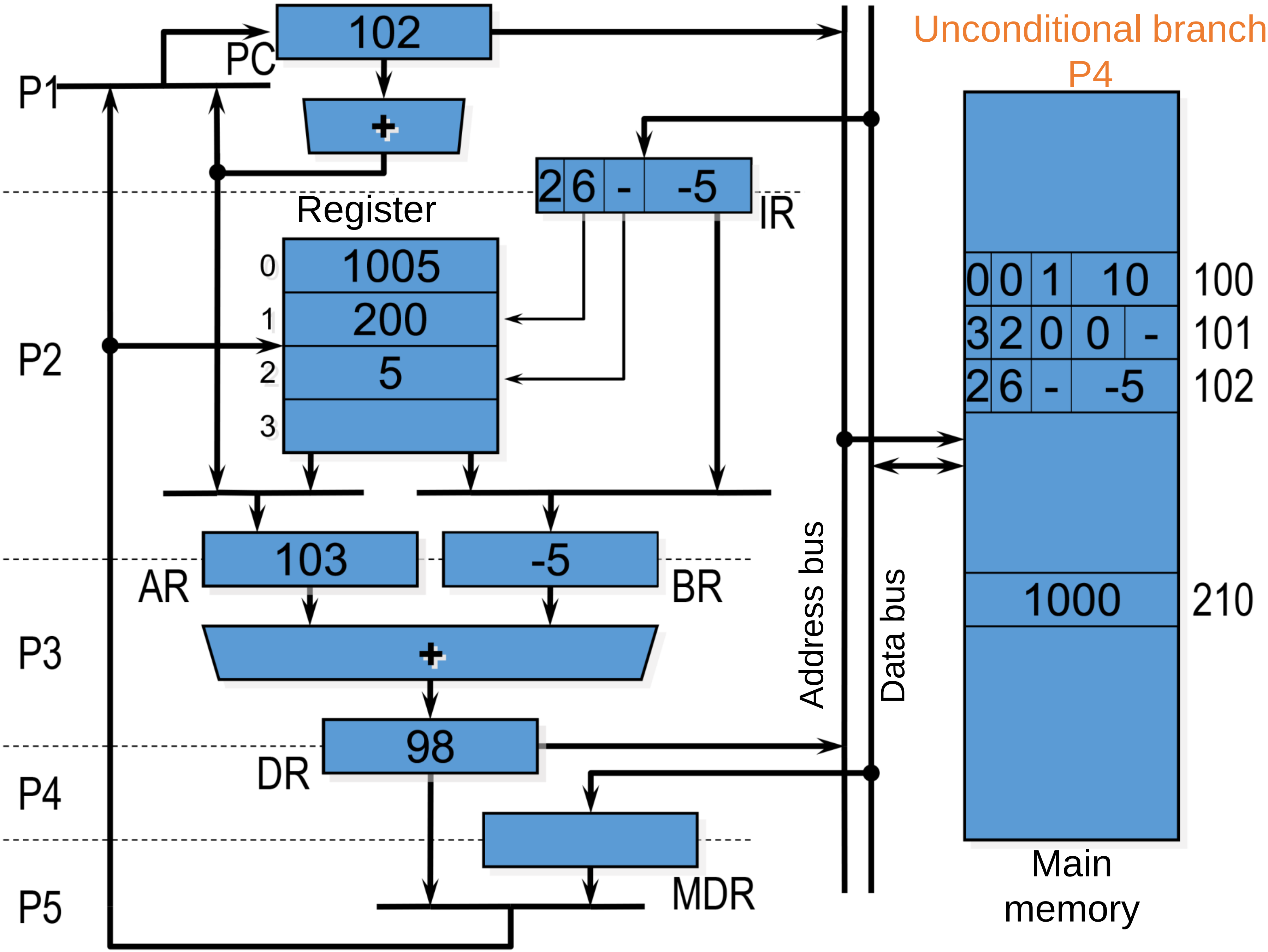
Unconditional branch
P2



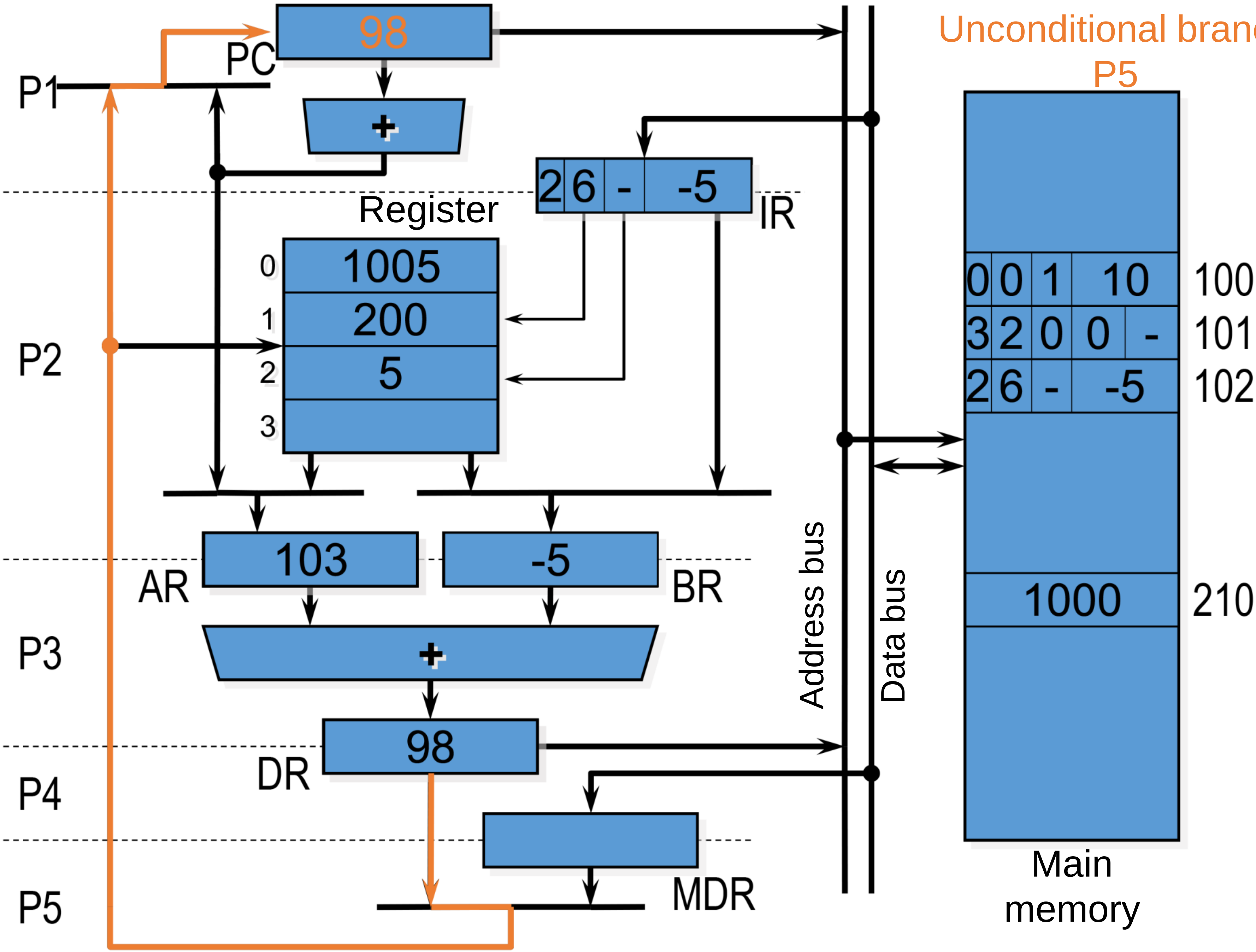
Unconditional branch
P3



Unconditional branch P4



Unconditional branch
P5



Hints for Designing

Module Configuration

- Divide the whole into sub-designs
  - Which logic to make into a unit?
  - Verilog HDL module units? Functional block units?
  - Which unit will each register belong to?
- Allotment of work
  - Control system and data path system?
  - Sub-design, top design and interface?
  - Basic functionalities and extended functionalities?
  - Design different versions of the same functional block separately?

Test Environment

- Simulation test bench
  - Automate so that the manual work needed for each simulation is reduced
  - **Automate from an early stage**
- Machine testing
  - Probe internal signals using the board's switches or LED
  - Configure a testing circuit (probe and switch, display driver) onto the exterior of the processor itself
  - **Create a test environment from an early stage**

Progress Management and Schedule

- Top down? Bottom up?
  - Create from parts or get the top design ready first (using dummy parts)?

- Prototyping, milestones, gantt charts
  - How to structure a schedule and functions such that it can run some instructions by the time of the midterm report (taking into account time for testing)?
  - Start with the simplest functionalities and configuration first? Or first think of a configuration with extensions?
  - When to decide the specifications of the final deliverable? When to re-examine?

Assignments and Demonstrations

Assignment: Functionality Extensions and Performance Evaluation

- Add some kind of extension and **perform an evaluation comparing before and after the extension was included**
  - How much faster are programs running?
    - Max clock frequency, number of instructions executed, number of execute cycles
  - What extra hardware was needed?
    - Number of gates (number of LUTs)
- Examples of potential extensions (Refer to Chapter 4 of SIMPLE Design Resources)
  - Improvements to the instruction set architecture: improvements to existing instructions, adding new instructions, adding support for interrupts
  - Improvements to the microarchitecture: parallel execution of phases (pipelining), parallel execution of instructions (superscalar)

Contest

- To you who wants to prove that your processor is the best and leave your name in history...
- A race to see who's processor can sort data the fastest (changes planned?)
  - Data
    - 1024 16-bit signed integers
    - Three types of data: random, sorted in ascending order, and sorted in descending order
  - Definition of time: number of cycles until completed × clock frequency
  - An average value of the processing time for each three types of data
- Please participate and try to break the current record
  http://isle3hw.kuis.kyoto-u.ac.jp /contest /index.html