

Machine Code and Data

Lecture 5

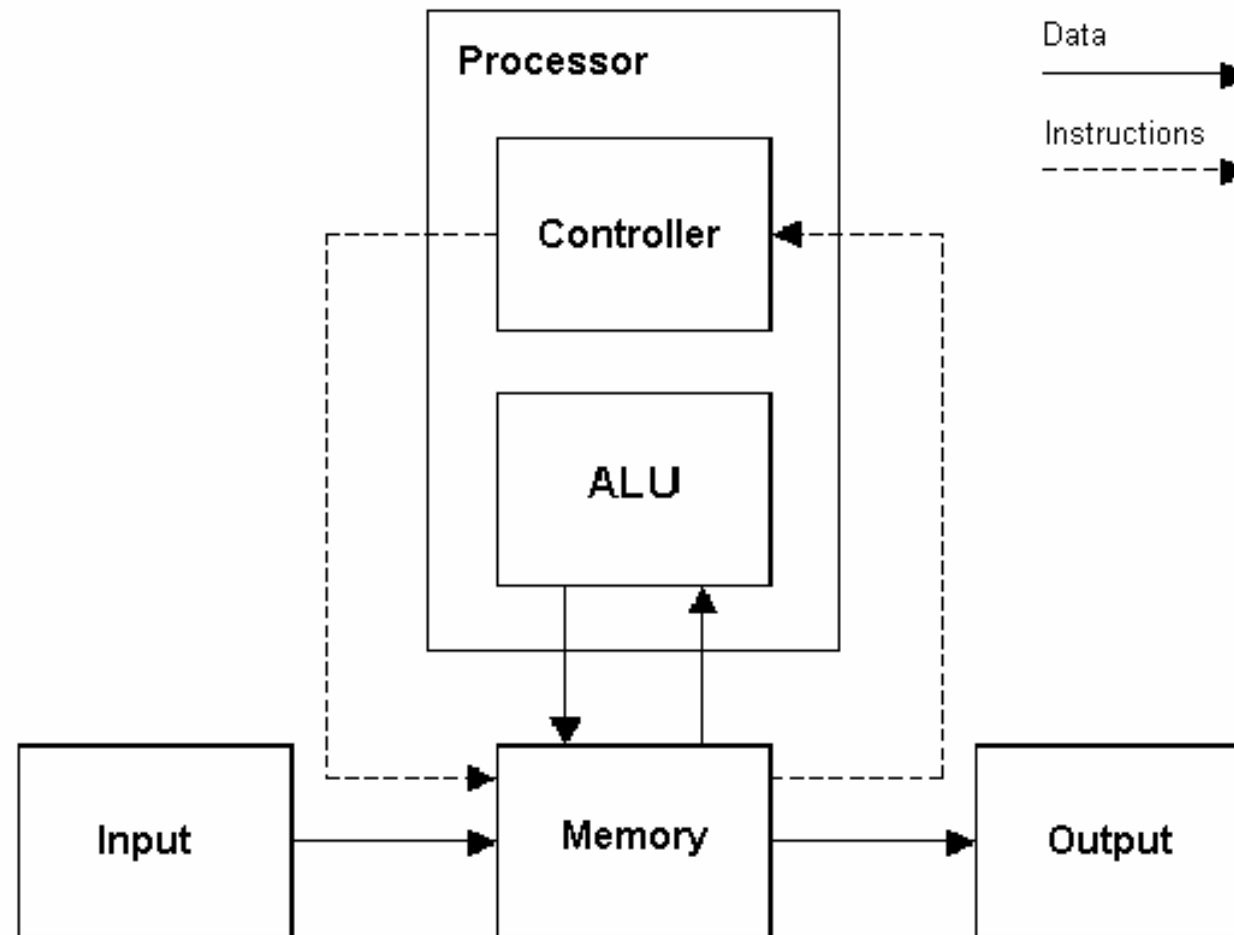
Basic cycle of a simple von Neumann computer

R. Lórencz, J. Hlaváč

Outline

- Description of the architecture (memory; CPU: ALU, CU, registers; bus)
- Memory, cooperation with other units
- Operation of computer units during IF, ID, OF, IE, WB
- Memory and machine code (demonstrating on given architecture)
- Operation and significance of PC, IR in the basic cycle
- Jumps (absolute, conditional)

von Neumann architecture, revisited



Simple von Neumann type processor (1)

CPU (Central Processing Unit)

- ALU (Arithmetic/Logic Unit)
- CU (Control Unit) – Controller
- **Registers**

Memory

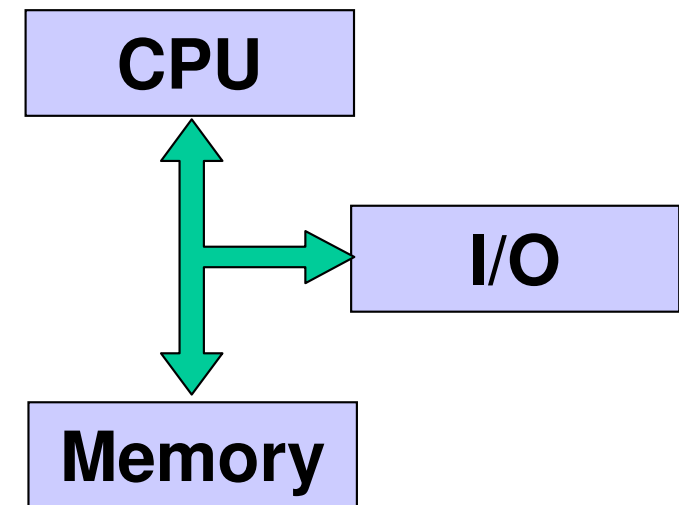
- Data
- Instructions – program

I/O (Input/Output)

- Disk drives
- Graphics units
- Ports, etc.

Microprocessor bus

- Communication between CPU, I/O and Memory



Simple von Neumann type processor (2)

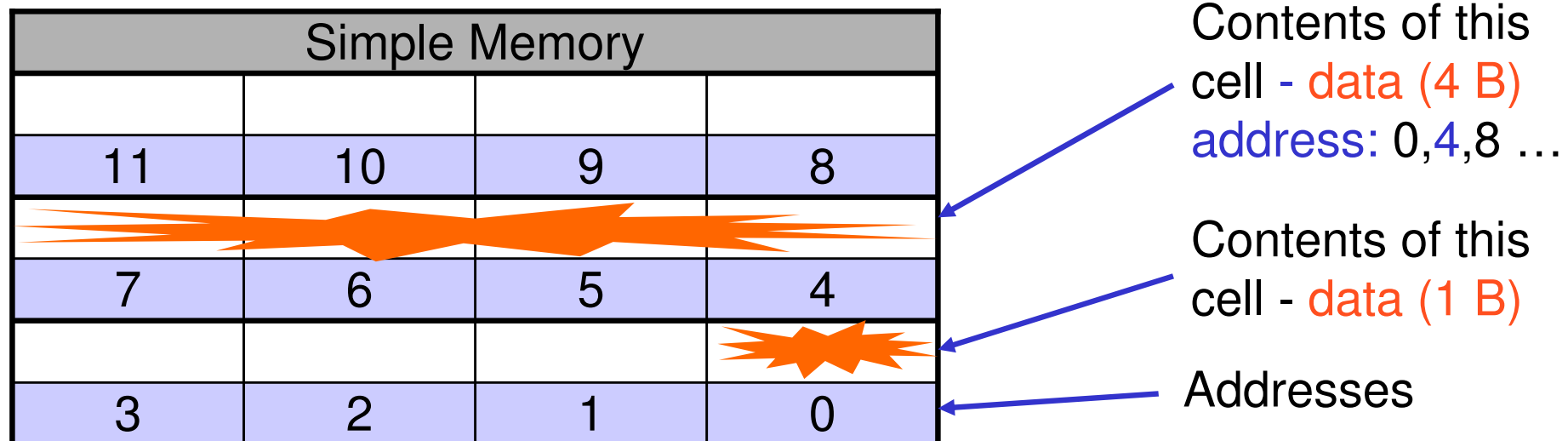
Memory

- Divided into **cells** (memory locations)
- Cells are **addressable** with nonnegative integers
- Size of a memory cell (CPU dependent):

Word, e.g. 16b, 32b, 64b,...

Byte, usually 1B = 8b

Notation to indicate the memory cell at the address a : $[a]$ or $\langle a \rangle$.



Simple von Neumann type processor (3)

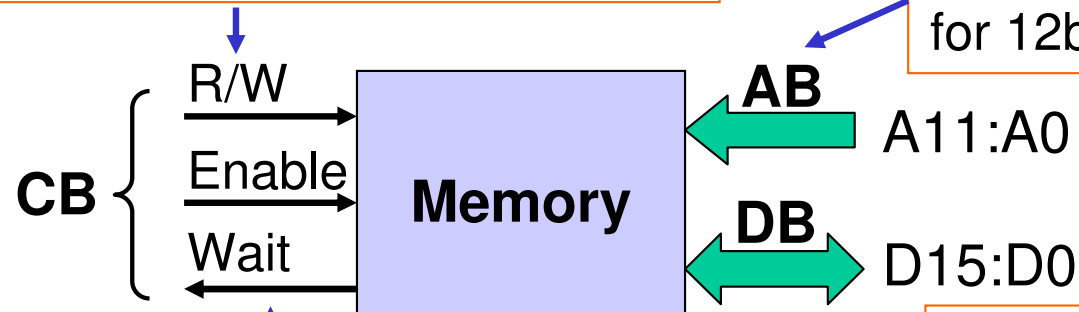
Memory

CPU communicates with the memory using 3 types of busses

- Address bus (AB), determines the memory cell (memory location)
- Data bus (DB), used to transfer the contents of a cell (data)
- Control bus (CB), controls the transfers, etc.

R/W – determines whether the memory is Read from or Written into

The AB width determines memory size;
for 12b AB => $2^{12}=4096$ cells

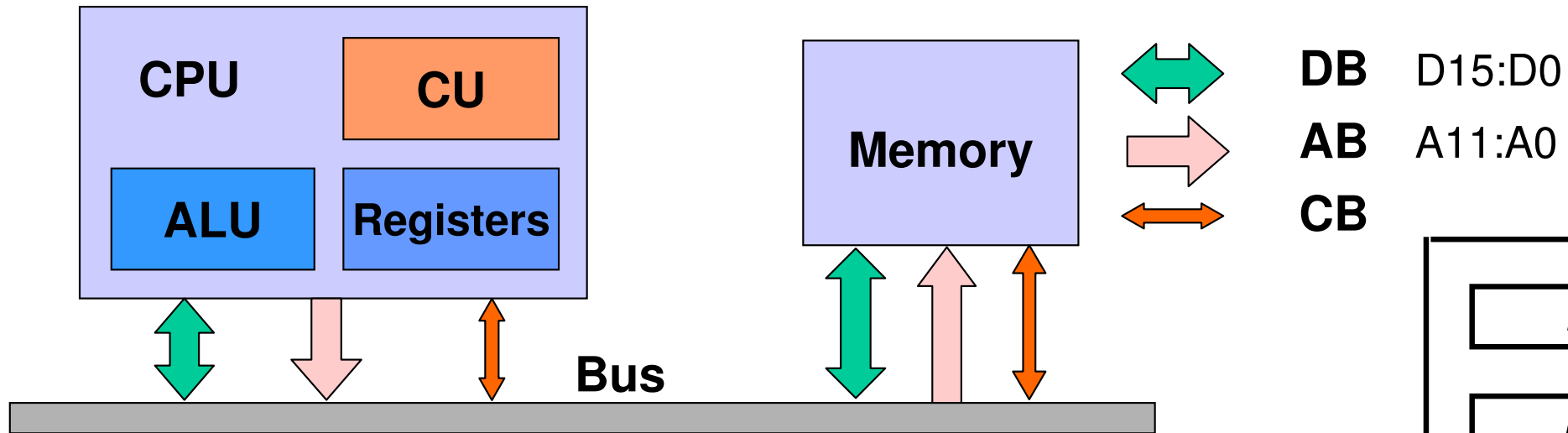


Enable – activates the memory
Wait – informs CPU if the memory is ready to be read or written

DB width determines the size of the content of one addressable location
If DB is 16b => 1 address points to 16b of memory contents

Simple von Neumann type processor (4)

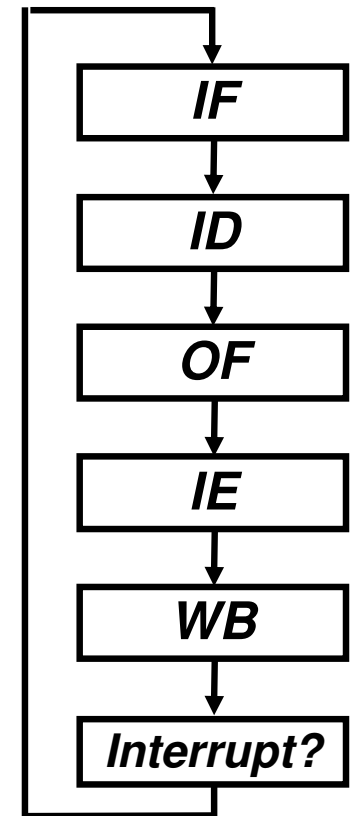
Memory, CPU



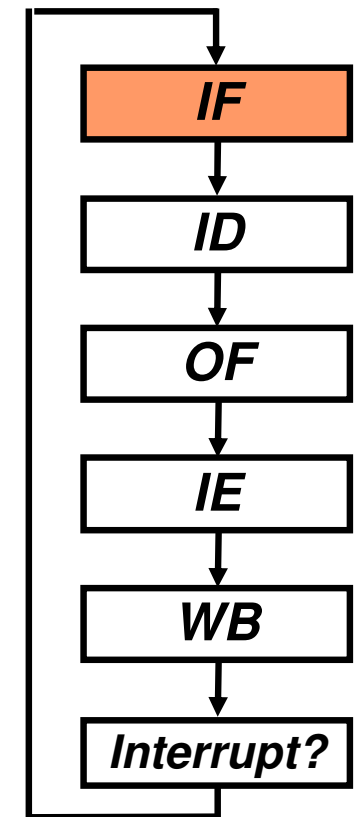
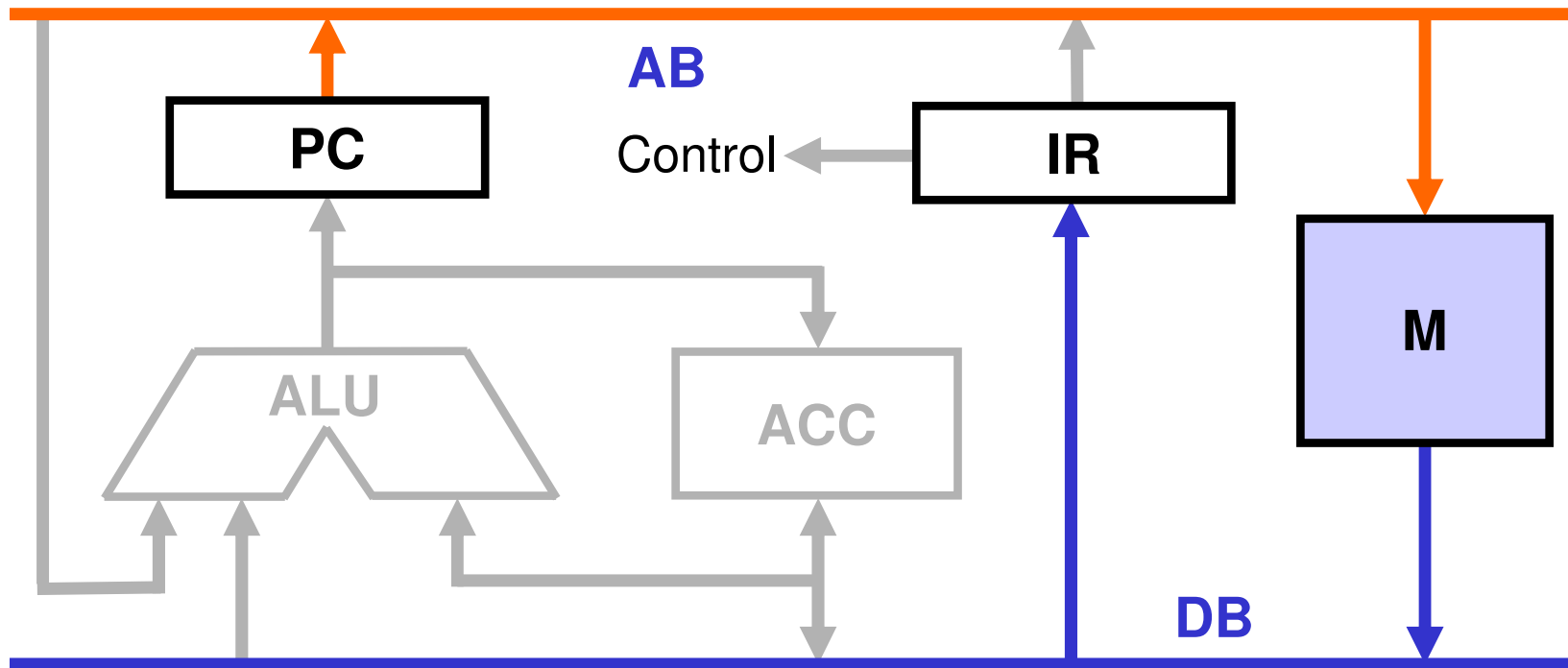
CPU must contain **ALU**, **CU** and **registers** for temporary storage of data, instructions and temporary variables

To execute a program, **CPU** repeats the **basic computer cycle**:

1. Instruction Fetch **IF**
2. Instruction Decode **ID**
3. Operand Fetch **OF**
4. Instruction Execution **IE**
5. Write Back **WB** (also Result Store)



Simple processor – P0, IF (Instruction Fetch)

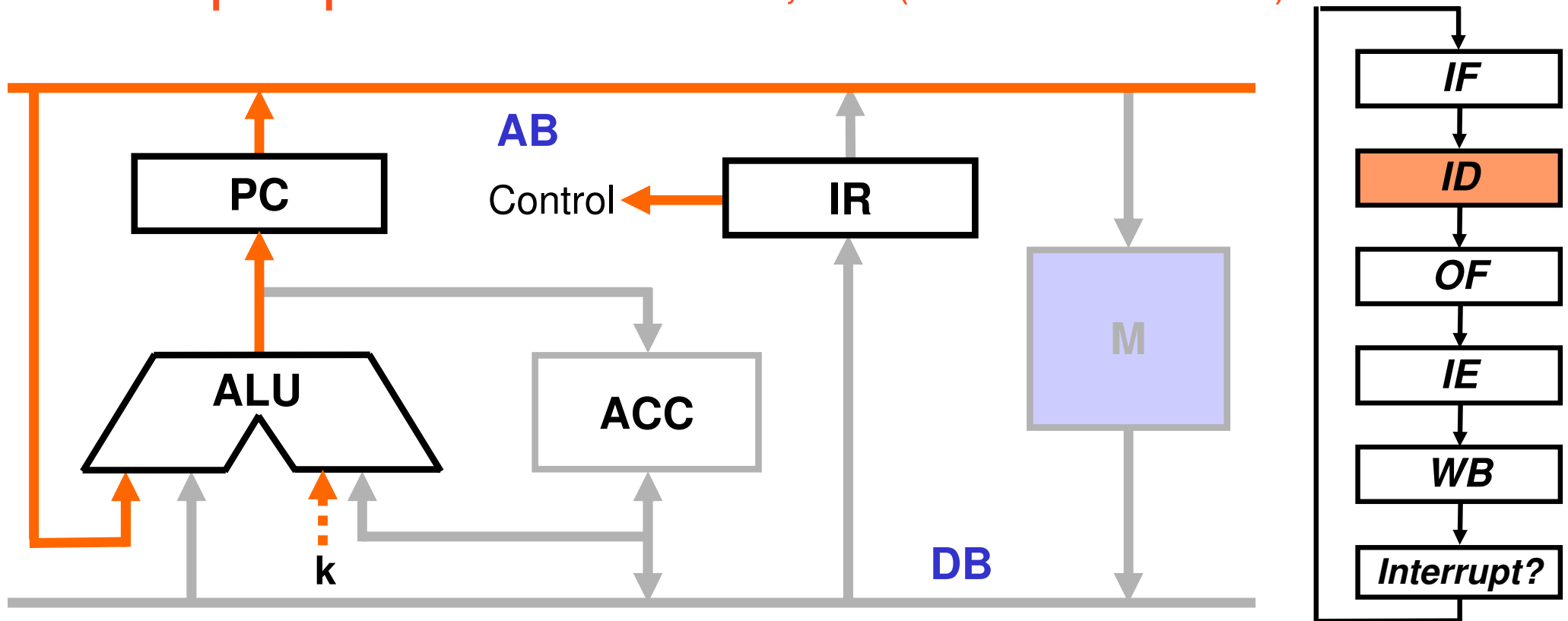


P0 – puts the address of the instruction in PC to AB

M – puts the contents of the memory location (instruction) to DB

IR – instruction code at DB is stored to IR

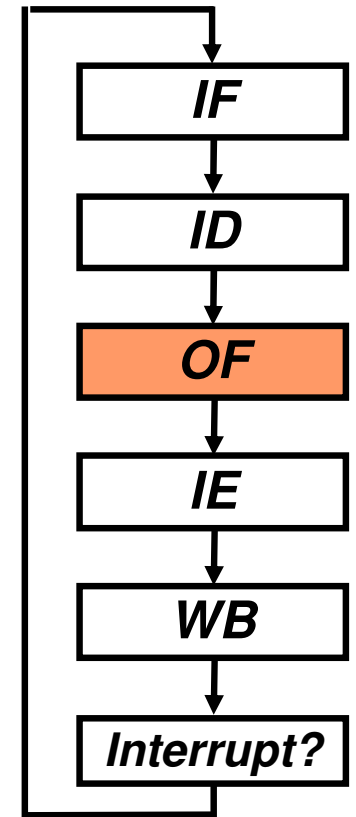
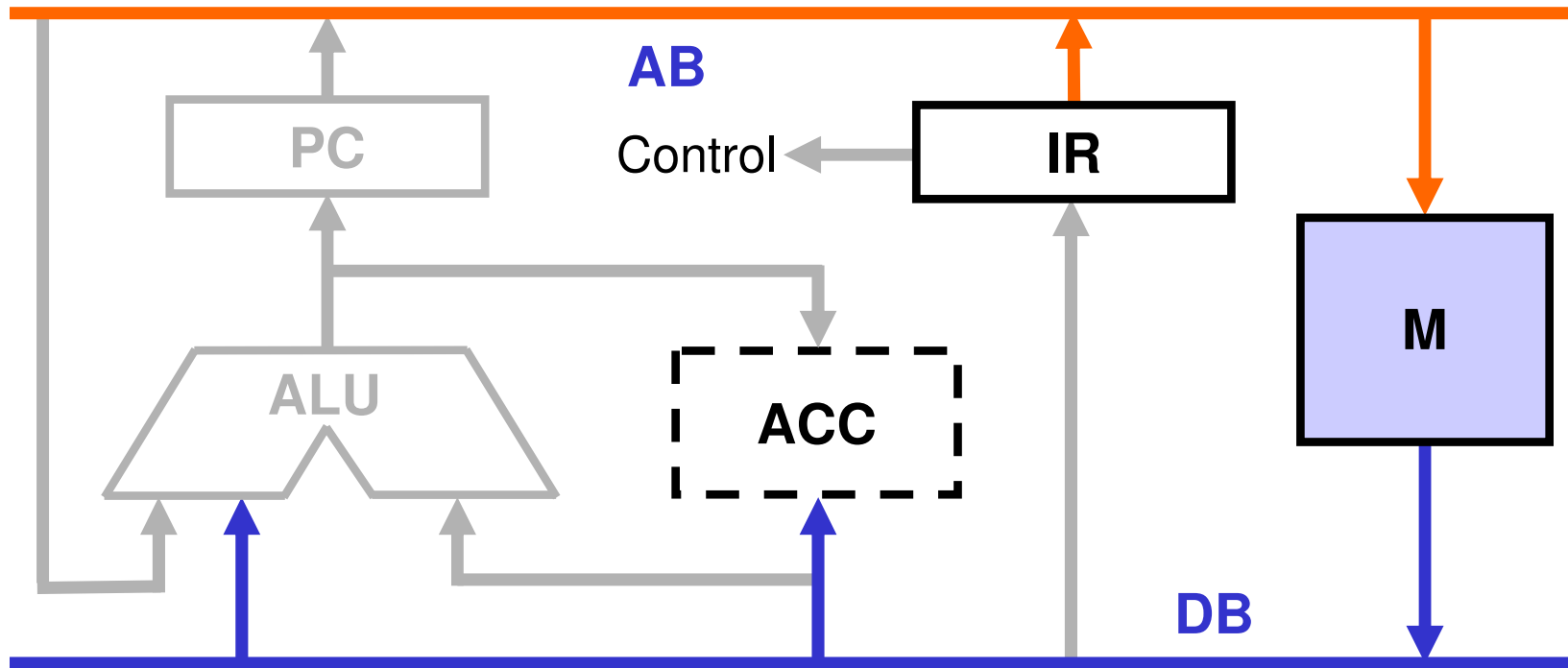
Simple processor – P0, ID (Instruction Decode)



IR – instruction code in IR is decoded by the internal logic (decoder), control signals for the ALU as well as other internal P0 circuitry are generated.

PC – program counter puts the value to AB, ALU increases this value by k and stores it back to PC (assuming pc is the original content of PC, then $pc \leftarrow pc+k$); k is determined by decoding the instruction - for sequential processing $k=1$, for a jump k is an integer.

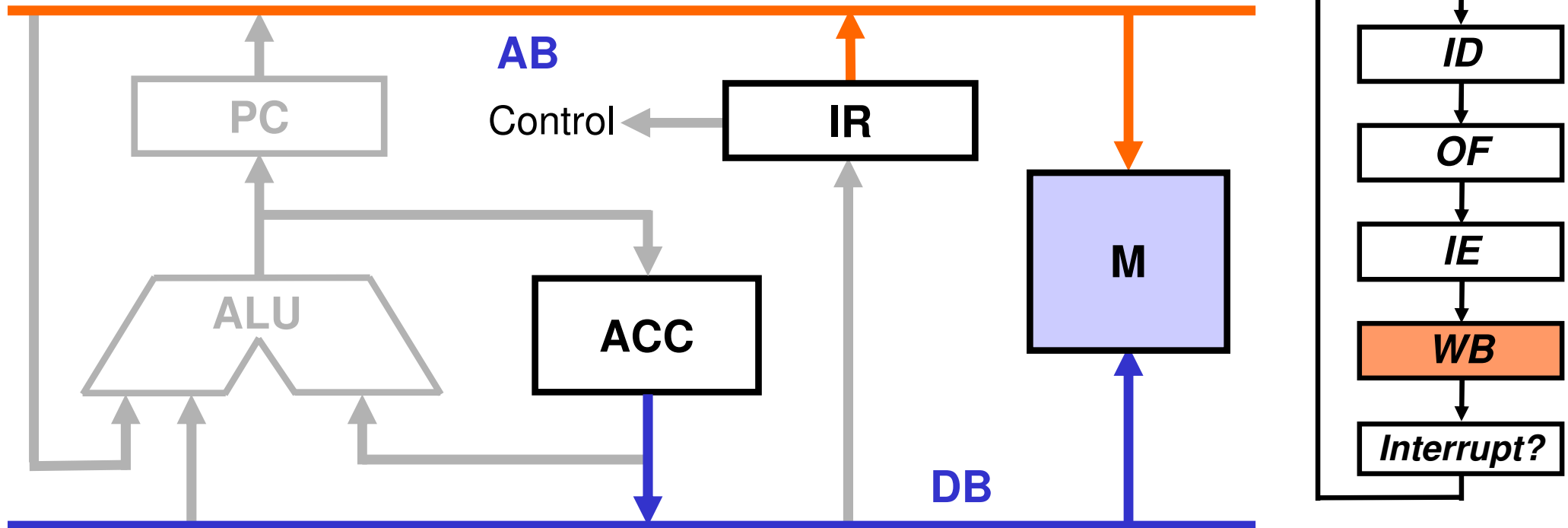
Simple processor – P0, OF (Operand Fetch)



IR – puts the operand address to AB (the operand which is needed for executing an instruction)

M – Memory puts the operand value to DB, making the operand ready for processing in the ALU or ACC

Simple processor – P0, WB (Write Back) not always executed



IR – puts the memory address where the value in ACC is to be stored to AB

ACC – puts the value it contains to DB

M – the value at DB is written to the memory

This phase of the basic cycle may be skipped. In some processors, it is performed as a separate instruction.

Simple processor – P0, instructions (1)

Let us assume that P0:

- Has only 8 instructions
- Can address only 4 kB of memory (2^{12}) => AB is 12b wide
- Length of **all** instructions is **16b**
- 16b instruction code, **machine code**, has this format:



where **OP** is the operation code (“opcode”)
and **S** is either:

- address of an operand whose value is in memory (or is to be stored to the memory), or
- value of an immediate operand

Simple processor – P0, instructions (2)

P0 Instruction Set

Instruction	OP	Description
LDA S	0000	$ACC \leftarrow M[S]$
STO S	0001	$M[S] \leftarrow ACC$
ADD S	0010	$ACC \leftarrow ACC + M[S]$
SUB S	0011	$ACC \leftarrow ACC - M[S]$
JMP S	0100	$PC \leftarrow S$
JGE S	0101	If $ACC \geq 0$, $PC \leftarrow S$
JNE S	0110	If $ACC \neq 0$, $PC \leftarrow S$
STP	0111	Stop ($PC \leftarrow FE0$)

Instruction types:

2 load/store: LDA, STO

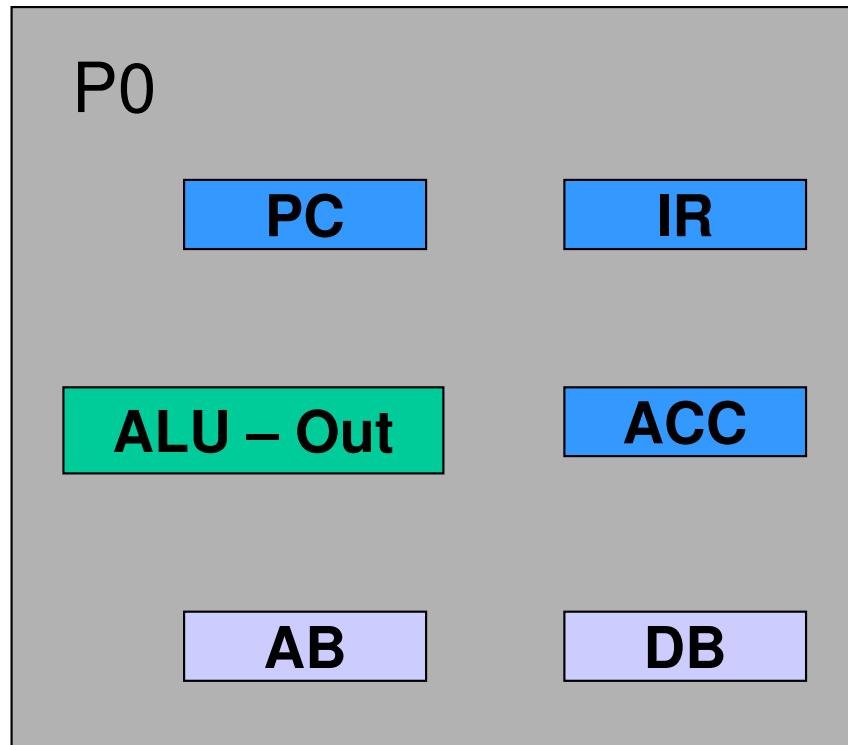
2 ALU: ADD, SUB

3 branch: JMP, JGE, JNE

1 control: STP

Simple processor – P0, program (1)

Program:



Address	Mnemonic	Machine code
000	LDA 02E	002E
001	ADD 02F	202F
002	STO 030	1030
003	STP	7FE0
--	--	--
02E	ABCD	ABCD
02F	4321	4321
030	--	--

What are the contents of PC, IR, ACC registers, values at the ALU output (ALU – Out) and at the AB and DB buses during individual phases of the basic P0 computer cycle?

Simple processor – P0, program (2)

After reset, all values are set to 0.

Phase	PC	IR	ACC	AB	DB	ALU-Out
IF	000	0	0	000	002E	X
ID	000	002E	0	000	X	001
OF	001	002E	0	02E	ABCD	X
IE	001	002E	ABCD	X	X	X

Addr.	Mnemonic	Machine code
000	LDA 02E	002E
001	ADD 02F	202F
002	STO 030	1030
003	STP	7FE0
--	--	--
02E	ABCD	ABCD
02F	4321	4321
030	--	--

WB is (of course) not executed for LDA instructions. We continue with the next instruction whose address is in PC, that is, the instruction at the address **001: ADD 02F**

Note that IE can be joined with IF of the following instruction.

Simple processor – P0, program (3)

Continue by executing the instruction at address **001: ADD 02F**

Phase	PC	IR	ACC	AB	DB	ALU-Out
IF	001	002E	ABCD	001	202F	X
ID	001	202F	ABCD	001	X	002
OF	002	202F	ABCD	02F	4321	EEEE
IE	002	202F	EEEE	02F	4321	X

Addr.	Mnemonic	Machine code
000	LDA 02E	002E
001	ADD 02F	202F
002	STO 030	1030
003	STP	7FE0
--	--	--
02E	ABCD	ABCD
02F	4321	4321
030	--	--

WB - again not performed for ADD instructions.

The execution continues with the next instruction whose address is in PC, that is, the instruction at the address

002: STO 030

Simple processor – P0, program (4)

Continue by executing the instruction at address **002: STO 030**

Ph.	PC	IR	ACC	AB	DB	ALU-Out
IF	002	202F	EEEE	002	1030	X
ID	002	1030	EEEE	002	X	003
WB	003	1030	EEEE	030	EEEE	X

Addr.	Mnemonic	Machine code
000	LDA 02E	002E
001	ADD 02F	202F
002	STO 030	1030
003	STP	7FE0
--	--	--
02E	ABCD	ABCD
02F	4321	4321
030	EEEE	EEEE

When storing ACC to the memory, OF and IE can be skipped.
Final phase is WB (memory store).

Execution continues with the instruction whose address is in the PC, that is, the instruction at address **003: STP**

Simple processor – P0, program (5)

Continue by executing the instruction at address **002: STP**

Ph.	PC	IR	ACC	AB	DB	ALU-Out
IF	003	1030	EEEE	003	7FE0	X
ID	003	7FE0	EEEE	003	X	X
IE	FE0	7FE0	EEEE	X	X	X

Addr.	Mnemonic	Machine code
000	LDA 02E	002E
001	ADD 02F	202F
002	STO 030	1030
003	STP	7FE0
--	--	--
02E	ABCD	ABCD
02F	4321	4321
030	EEEE	EEEE

This instruction is only read from memory and PC is overwritten in the IE phase with a new address, **FE0**, where an interrupt handler can reside.

It can, for example, wait for a new program to be loaded.

Simple processor – P0, program (6)

In our example, the STP instruction terminates program execution by jumping to the address FE0, which should contain e.g. an interrupt handling subroutine.

The JMP S instruction performs the same action; however, the target address is given by an immediate operand S.

For example, instruction **006: JMP 020** results in:

Ph.	PC	IR	ACC	AB	DB	ALU- Out
IF	006	X	X	006	4020	X
ID	006	4020	X	006	X	X
IE	020	4020	X	X	X	X

Simple processor – P0, program (7)

Conditional branch instructions **JGE S** and **JNE S** jump to the address **S** only if a condition is true. For **JGE S**, it is: **ACC ≥ 0**. For **JNE S**, it is: **ACC ≠ 0**.

The **ACC ≥ 0** test can be realized by testing the MSB of ACC.

The **ACC ≠ 0** test can be realized by the logic “OR” applied to all bits of ACC (a part of ALU).

It is advantageous to have another register containing the flags that relate to the ACC or to the result of the last ALU operation.

The bits in this register can be for example:

- flag indicating zero contents of ACC
- flag indicating negative contents of ACC
- overflow flag, ...