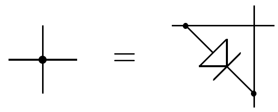
11. μ-programs

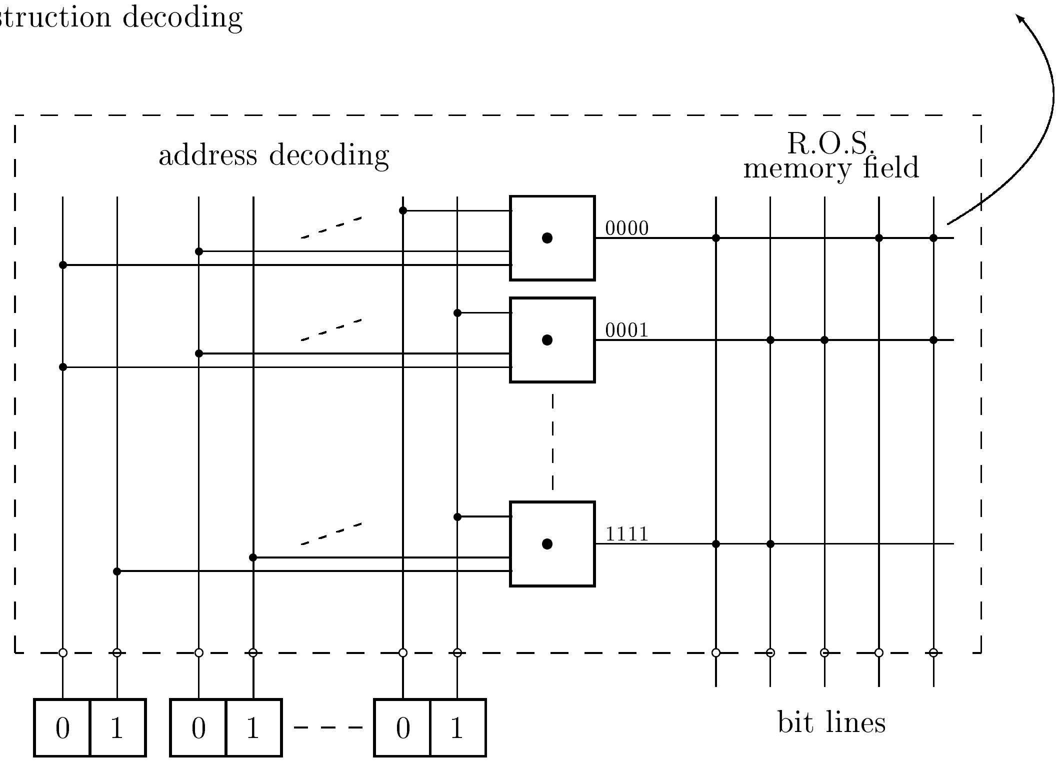
We have seen one function of a μ-computer, namely addition and subtraction in the ALU. This is done in hardware because of the speed. Other functions are multiply and shift, which can be used in combined forms (multiply = shift + add operations; note that all arithmetic functions can be done by using only the + function). We also but briefly discussed registers/buffers.

A μ-computer (processor itself, no peripherals) consists (or may consist) of only a few elements: ALU, registers and control logic.
The control logic takes care of data flow and function commands. This is defined in so-called μ-programs which define the μ-orders for performing basic tasks.
Control logic consists of/does:

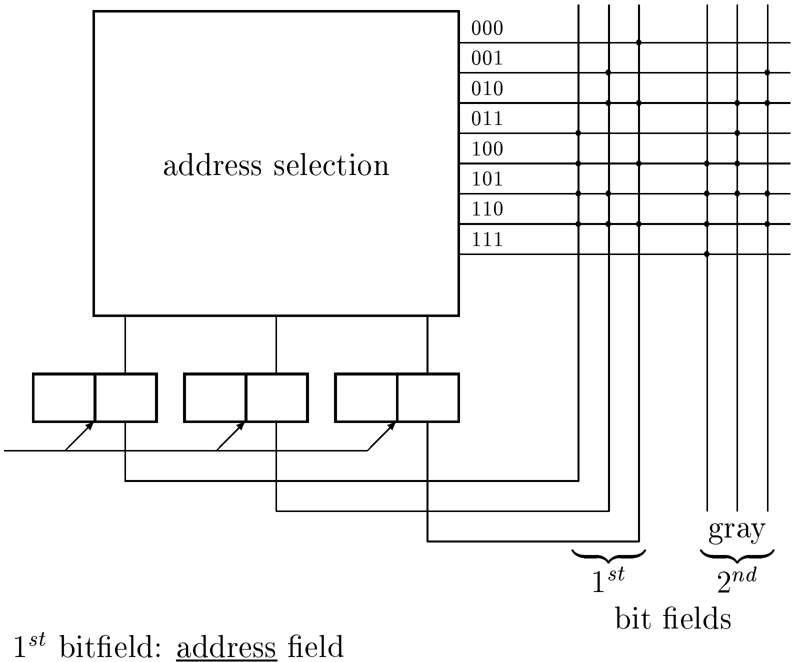
- a non-volatile memory (ROS = read only store)
- b address selection
- c address modification
- d address buffer
- e instruction decoding



a+b:



11.1 Applications of the addressing mechanism



- feedback of bitlines to address buffer generates a new address
- other bitlines can be used for generating a specific code (like gray-mirror code)

  NOTE We also could count directly in the gray code!

1^{st} bitfield: address field
2^{nd} bitfield: code field

An address field can be complete (all lines are used for address) or incomplete (some address lines may be stored in a register which can sometimes be changed)

- Here only complete address fields are considered for simplicity.

Depending on a function to be performed, a μ-program can be modified.
Example: multiplying words of 32 bits. The signs give four possibilities: $++$, $+-$, $-+$, $--$.
If both numbers are made positive first, the same multiplication method (μ-program routine) can be used in all 4 cases:

1) examine signs
2) multiply
3) put sign bit

In such a case, some parts of a μ-program can be skipped. This is done by jumping to another address. This is called "address modification."

One could use the two least significant bits for jumps.

Example `JOHN` ~~`xxxxx xx`~~ the next address will
`PETER 00` be `PETER 00`

We can now change the two `0`'s (depending on the problem) such that the following addresses are

```
PETER 00 - no modification
PETER 01
PETER 10
PETER 11
```

On these 4 addresses, we can put a jump instruction or a jump command, such that the μ-program goes to a specific piece of code.

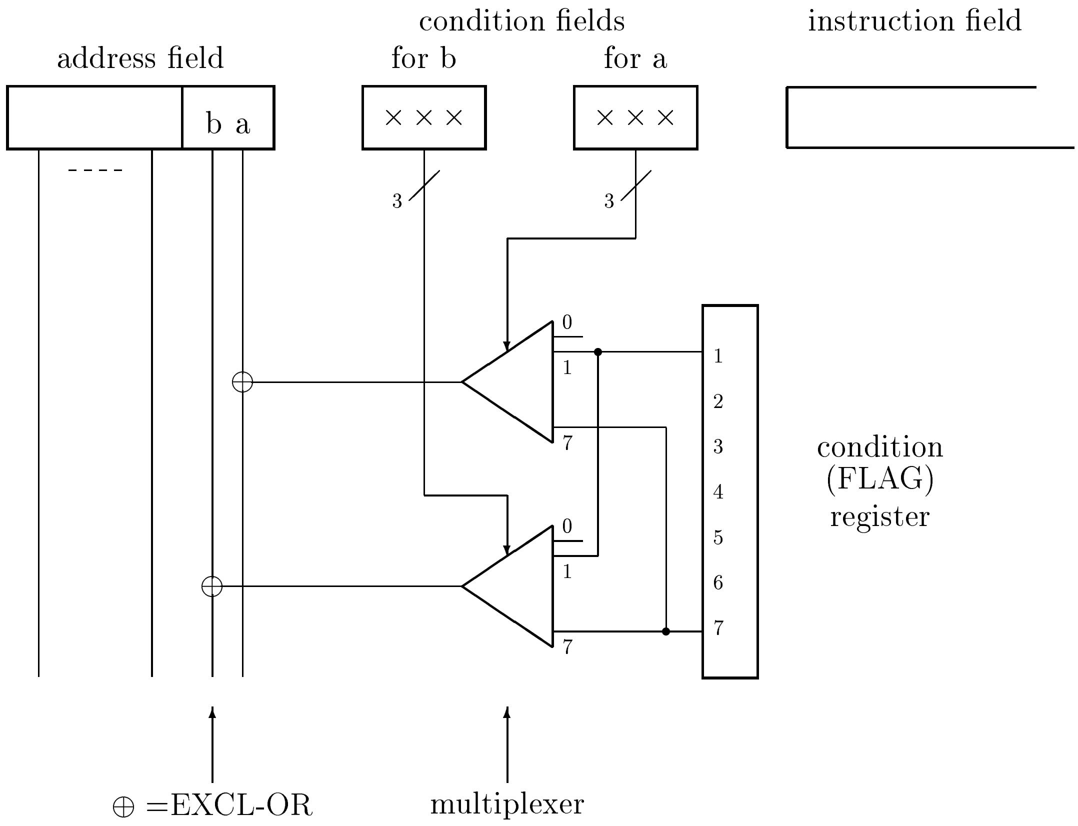
Example

LABEL	address	instruction
`JOHN`	`PETER 00`	
`JOHN+1`	⋮ rest μ-program	
	⋮ part	
`PETER 00`	`MARY`	
`PETER 01`	`ANN`	
`MARY`	`MARY+1`	
	`+2`	
	⋮	
`MARY+m`	`JOHN+1`	
`ANN`	`ANN+1`	
	`+2`	
	`+3`	
	⋮	
`ANN+n`	`MARY`	

Address modifications are often made on the basis of the contents of a condition register (or FLAG register)

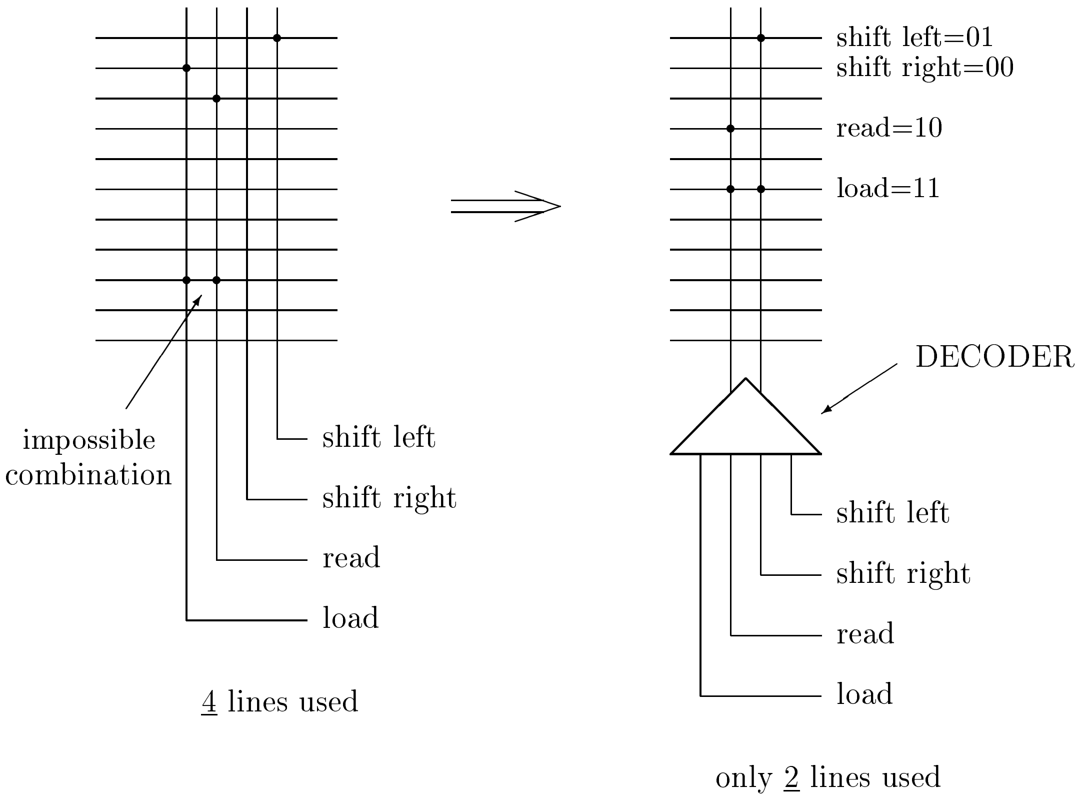
$$\left.\begin{array}{r}\text{carry}\\ \text{sign}\\ \text{zero}\\ \text{parity}\\ \text{overflow}\end{array}\right\}\ \text{modified upon certain instructions like additions}$$

A condition field is used to define which bit(s) of the condition register is (are) used in the address modification:



This <u>instruction field</u> has often <u>groups</u> of instructions.
Some instructions are mutually exclusive, like shift-left and shift-right.
Instructions can be <u>coded</u>; later on again <u>decoded</u>.

Example



- Designers collect mutually exclusive instructions. These are combined in specific instruction sub-fields. In this way, the total field becomes smaller and cheaper. However, to be added are decoding/demultiplexing costs, such that an economic minimum must be found.