# Developer's Guide – Full Edition

## 0. Table of contents



## 1. The data-driven architecture – PI architeture overview

**What does it mean that this assembly is data-driven?** It means that the programmer simply manipulates the data and moves it around – computer is the one conducting the operations.

**What does it mean that this assembly is lightweight?** It means that it will only take you a couple of minutes to read the full Developer's Guide and get familiar with the architecture.

### Core principles:

1. Simplicity – all CPU **registers are co-addressed as memory**, giving the programmer maximum wiggle room in manipulating the data, with **just 2 simple assembly instructions**.

2. Complexity on the programmer's side – you, as the programmer are the best equipped to know how data should be manipulated by your code. That's why there's **no complex multi-step instructions**. there is no micro-code – **PI assembly itself is essentially the micro-code**. This way you have maximum control over what exactly is being done – isn't this exactly what coding in assembly is about?

3. Speed – **Every instruction takes one clock cycle** and one clock cycle only.

### Arthmetical-Logical Unit (ALU):

This most crucial element of the CPU conducts all Arthimetical and Logical operations.
In the 4-bit PI architecture CPU, the **ALU has two 4-bit inputs and one 4-bit output.**

The output is either an **arithmetical sum of the two numbers given on input** or **NAND of all bits from the inputs**. Output cannot be enabled or disabled. **ALU is always listening to inputs and always transmitting it's output.**

Whenever ,,ALU input" and ,,ALU output" is mentioned in this document, it is referring to exactly those values.
ALU also always stores the carry bit from the last operation.

## 2. Data and Flags

All registers and memory cells are 4-bit wide.

### Memory:

All CPU registers are co-addressed with memory. Meaning that they all can be accesed just like normal memory cells.

| | Adress: | Annotation: |
|---|---|---|
| **Registers** | 0000 | **00** registry – Instruction Pointer |
| | 0001 | **01** registry - Alpha |
| | 0010 | **10** registry - Beta |
| | 0011 | **11** registry - Accumulator |
| **Regular memory** | 0100 | |
| | … | |
| | 1110 | |
| | 1111 | |

As you may observe, 2 last bits of the adress correspond to the registry number.

**Instruction Pointer (IP)** – shows instruction that is currently being executed. 2 is added automatically after each instruction to this registry. IP can only store even numbers.

**Alpha and Beta** – two registers wired directly to the ALU at all times (always enabled).

**Accumulator** – hard wired to ALU, after each instruction in which Alpha or Beta are changed the Accumulator is automatically updated with ALU output.

Memory containing instructions shall always be aligned by 2. First 4 bits of the instruction are **to be stored under an even address**. Next 4 bits are to be stored in the next memory cell (with an odd adress). As shown below:

| Adress: | Content: |
|---|---|
| XXX0 | Instruction 1 |
| XXX1 | |
| XXX0 | Instruction 2 |
| XXX1 | |

Because all registers serve as memory, there is nothing preventing you from storing instructions inside registers.

### Flags:

| Flag number: | Flag name: | Description: |
|---|---|---|
| 00 | HF – Halt Flag | When the flag is set to 1, all activity of the CPU is stopped. When the flag is set to 0, CPU works normally. |
| 01 | SF – Skip Flag | When the flag is set to 1, all instructions are skipped, except instructions that would change the value of this flag. When the flag is set to 0 - all instructions are executed normally. |
| 10 | AF – ALU Flag | When the flag is set to 0, ALU output is sum of Alpha and Beta. Otherwise each bit of ALU output is NAND of respective bits of Alpha and Beta |

## 3. Instructions

All instructions are 8-bit long, meaning they take 2 memory cells each.

### MOV:

The most important and basic instruction of the PI assembly. It is used to move data from a registry to memory (or reverse). Note, that since all registers are co-addressed with memory – you can also move data between registers (**MOV 0 01 0011** will move data from 11 registry to 01 registry)

| **0** | **R** | **REG** | **MEM** |
|---|---|---|---|
| 1 bit | 1 bit | 2 bits | 4 bits |
| Instruction code | Reverse Flag | Registry number | Memory address |

When R is 0, data is transferred from memory (MEM) to a regisry (REG). When R is 1, the direction is reversed.

### FLAG:

This is an instruction, that serves to alter the control flow of the program, by changing flags (described on page 2).

| **1** | **FLAG** | **VALUE** | **CARRY** | **ACC** | **NEG** | **OP** |
|---|---|---|---|---|---|---|
| 1 bit | 2 bits | 1 bit | 1 bit | 1 bit | 1 bit | 1 bit |
| Instruction code | Flag number | Value inserted into the flag | Indicating if to include carry | Indicating if to include accumulator | If set to 1 – final value inserted into the flag will be negated | Indicating operation performed on the inputs (0 – OR, 1 - AND) |

Flag indicated by the argument **FLAG** will be set.

Other arguments determine what will the input be.

**VALUE** is given directly to be put into the FLAG

**CARRY and ACC** indicate if **Carry from the ALU (from the last operation)** and l**ogical sum (OR) of all bits in the Accumulator** will be combined with the value given in argument VALUE.

**OP** indicates in what way all included values will be combined (AND / OR).

**NEG** gives the opportunity to negate the whole input (after including VALUE, CARRY and ACC, as well as conducting operation specified by OP)

## 4. Input and output operations

CPU and memory compliant with the PI architecture should provide **5 output pins and 10 input pins,** that allow for communication with peripheal devices, external sensors, etc. as well as programming of the computer.

In general, PI CPU works in one of **2 availible modes: programming mode and running mode.**

Pins are outlined below. As you can see, the programming mode allows you to directly alter all memory and registers. Meanwhile the running mode only allows data to be input into the Accumulator.

| Size and type: | Name: | Operations in programming mode: | Operations in running mode: |
|---|---|---|---|
| 1-bit input | Mode switch | Pin for switching between modes<br>1 – running mode<br>0 – programming mode | |
| 1-bit input | Input confirmation pin | Sets the memory cell specified by the **Adress input** to the value specifiec in the **Data input** | If Halt Flag equals 1, sets Accumulator to the value specified in **Data input.** Otherwise does nothing. |
| 1-bit input | Resume pin | The behaviour is undefined. | 0 – does nothing<br>1 – Sets the **Halt Flag** to 0, if **HF** equals 0, then does nothing. |
| 4-bit input | Address input | Wired directly to address bus | The behaviour is undefined. |
| 4-bit input | Data input | Wired directly to the data bus. | Wired directly to **Accumulator** registry (0011 memory cell) input. |
| 4-bit output | Data output | The behaviour is undefined. | Those pins are always set to the value of the **Accumulator.** |
| 1-bit output | Halt indicator | The behaviour is undefined. | This pin is always set to the value of the **HF** |