The ZTH1 CPU, v. 1.0

S. Morel, Zthorus-Labs, 2019-12-22

1 Introduction

The ZTH1 is a central processor unit (CPU) that can be used to build small computers around it. It can be synthesized in VHDL on a wide range of FPGA components. For instance, it has already been implemented on a Maximator board based on an Altera MAX10 FPGA.

2 General architecture

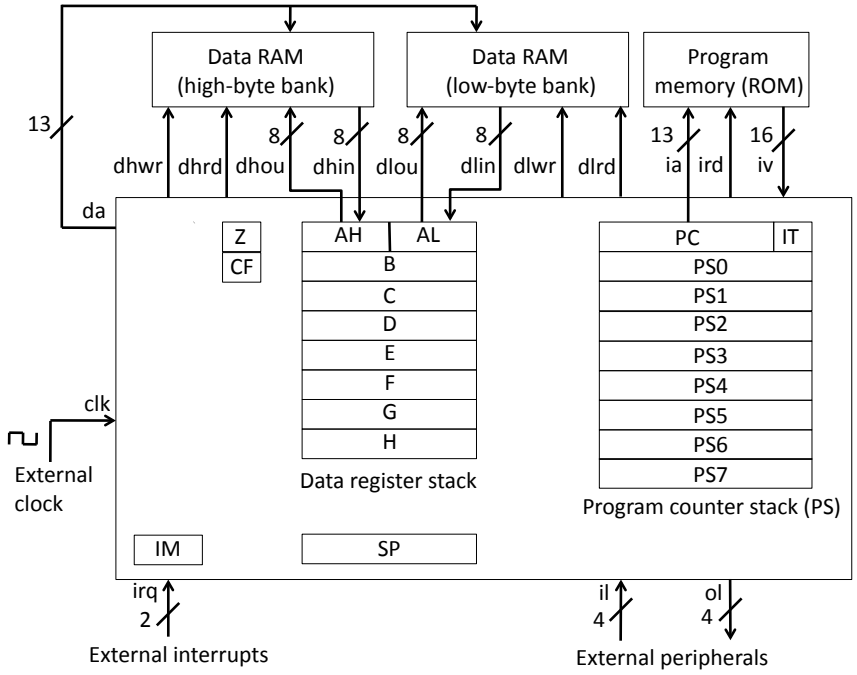
The ZTH1 is a 16-bit RISC CPU. It features an Harvard architecture: the buses to access the instruction words and the data words are not the same. The external signals of the ZTH1 component are:

Name	**Width (bits)**	Direction	**Function**
`ia`	13	Out	Instruction address bus
`iv`	16	In	Instruction value bus
`da`	13	Out	Data address bus
`dhin`	8	In	Data value high-byte in
`dhou`	8	Out	Data value high-byte out
`dlin`	8	In	Data value low-byte in
`dlou`	8	Out	Data value low-byte out
`il`	4	In	Input lines (for peripherals)
`ol`	4	Out	Output lines (for peripherals)
`irq`	2	In	Interrupt requests
`dhwr`	1	Out	Write data high-byte
`dlwr`	1	Out	Write data low-byte
`dhrd`	1	Out	Read data high-byte
`dlrd`	1	Out	Read data low-byte
`ird`	1	Out	Read instruction
`clk`	1	In	Clock signal

Note that the restriction to 13 bits of `ia` and `da` comes from the memory limitation of the Maximator board. These buses could be 16-bit width in theory.

The data have to be stored in two banks of RAM. One bank stores the high-byte parts of the 16-bit data words while the other stores the low-byte parts of the data words.

The op-codes of the ZTH1 instruction set are 8-bit. Some of the op-codes use a 8-bit argument to affect a constant to either the `AH` or the `AL` register (see next section). In that case, the op-code shall be the high-byte on `iv`, while the argument is the low-byte on `iv`. This implies the use of "NOP-padding" by the assembler (see section 7).



Architecture of the ZTH1 CPU (signals and registers)

3 Registers

To manipulate data, the ZTH1 uses a stack of registers, like in classic RPN Hewlett-Packard calculators or INMOS Transputer CPUs. The ZTH1 data stack consists of eight 16-bit registers named `A`, `B`, `C`, `D`, `E`, `F`, `G` and `H` . `A` is the register at the top of the stack (and plays the role of an accumulator), `H` is the register at the bottom of the stack. The instruction set allows to access registers `A` to `D`. Arithmetic and logic operations affect either `A` and `B` together, or `A` alone (see section 6). The `A` register is split into two 8-bit sub-registers: `AH` (high-byte part of `A`) and `AL` (low-byte part of `A`). After some instructions, the stack is either "shifted down" (`H` takes the value of `G`. `G` takes the value of `F`, ..., `B` takes the value of `A`) or "shifted up" (`A` takes the value of `B`, `B` takes the value of `C`, ..., `G` takes the value of `H`).

The 16-bit instruction words are addressed by the `PC` (Program Counter) 16-bit register. In the current implementation, the 13 least-significant bits of this register are actually used for the addressing. Since 16-bit words are used for two 8-bit instruction op-codes (except for the op-codes with a constant argument), a flag `IT` (Instruction Toggle) indicates which byte (either high or low) in the value read from `iv` has to be executed at each clock cycle. `PC` and `IT` are at the top of a stack of eight 17-bit registers (named from `PS0` to `PS7`). This stack allows to handle the call to sub-routines with some limitations due to its depth.

The results of arithmetic and boolean operations are checked by the flags tt Z (Zero) and `CF` (Carry Flag). `Z` is set to 1 whenever the result stored in `A` is zero (otherwise `Z` is set to 0). `CF` is set to 1 if the result of an addition of `A` and `B` (stored into `A`) is larger than xFFFF. `CF` can also be used for bit shifting operations on `A` (see section 6). Note that unlike other CPUs, the value of `Z` in the ZTH1 is not automatically set according to the current value of `A`. `Z` is set only after an arithmetic or boolean operation affecting `A` (or a comparison of `A` with `B`) has been executed.

There is a stack created in RAM can be used to temporary store the current value of `A` and to retrieve

it. The address in RAM of the top of this stack is given by the SP (Stack Pointer) register.

The interrupts from external peripherals (see section 5) can be masked by the IM (Interrupt Mask) register. If the bit corresponding to one of the interruptions is set to 1, this interrupt will be ignored by the ZTH1.

4 Sequencing

On a positive edge of clk, the instruction op-code available on iv (either in the high- or the low-byte part) is executed. The registers and the are updated according to this instruction. The value available at this moment on the dhin, dlin, il and irq buses can be used to modify the registers. Right after execution of the instruction, the da bus is set (usually to the value of A clipped to its 13 lowest bits) and the dhou and/or dlou buses are set (if a STH, STL ot STW instruction has to be executed. The control signals dhrd, dlrd, dhwr and dlwr are set to 0 or 1 according to the executed instruction.

On a negative edge of clk, the RAM (the high-byte bank, the low-byte bank or both banks) is either read or write. At the same time, the ROM is read if a new instruction word needs to be fetched. The ROM and RAM shall have no output data latch: the value to be read shall be immediately available (after the physical propagation time) if the control signals (dhrd, dlrd, ird) have been set for a read. This sequencing implies to have the memory components driven by a clock with the same frequency as clk but inverted (output of clk through a NOT gate).

5 Interrupts

The current implementation of ZTH1 has two interrupts. When one of these interrupt lines is set to 0 V and a positive edge is received on clk, an interrupt vector is triggered: the current values of PC and IT are stored in the PS stack, IT is set to 0 and PC is set to a specific value. This value is x0000 for the interrupt 0 which corresponds to a "reset" of the ZTH1. It is x0008 for the interrupt 1 which can be used for any purpose. The 8-word code in the interrupt vector can be used for a jump to a sub-routine.

6 Instruction set

The instruction set of the ZTH1 consists of xx op-codes represented by 8 bits that are coded by 3-character mnemonic by the assembler.

Op-code	**Mnemonic**	**Name**	**Function**	**Modify Z**	**Modify CF**	**Shift-up stack after**
00	NOP	No operation	Wait 1 cycle	No	No	No
01	LDH xx	Load high-byte	Set `AH` to *xx*	No	No	No
02	LDL xx	Load low-byte	Set `AL` to *xx*	No	No	No
03	PSH xx	Push high-byte	Shift-down stack, then set `AH` to *xx*	No	No	No
04	PSL xx	Push low-byte	Shift-down stack, then set `AL` to *xx*	No	No	No
05	GTH	Get high-byte	Set `AH` to value in RAM addressed by value of `A`	No	No	No
06	GTL	Get low-byte	Set `AL` to value in RAM addressed by `A`	No	No	No
07	GTW	Get word	Set `A` to value in RAM addressed by `A`	No	No	No
08	STH	Store high-byte	Store `AH` at RAM addressed by value of `B`	No	No	No
09	STL	Store low-byte	Store `AL` at RAM addressed by value of `B`	No	No	No
0A	STW	Store word	Store `A` at RAM addressed by value of `B`	No	No	No
0B	SWA	Swap	Permute values of `AH` and `AL`	No	No	No
0C	CLL	Clear low-byte	Set `AL` to 0	No	No	No
0D	CLH	Clear high-byte	Set `AL` to 0	No	No	No
0E	DUP	Duplicate	Shift-down stack, keep value of `A` (i.e., `B` = `A`)	No	No	No
0F	DRP	Drop	Shift-up stack (`A` will be set to `B`, etc...)	No	No	No
10	SWP	Swap	Permute values of `A` and `B`	No	No	No

Op-code	Mnemonic	Name	Function	Modify Z	Modify CF	Shift-up stack after
11	RU3	Roll-up on 3 registers	Set [A,B,C] to [B,C,A]	No	No	No
12	RU4	Roll-up on 4 registers	Set [A,B,C,D] to [B,C,D,A]	No	No	No
13	RD3	Roll-down on 3 registers	Set [A,B,C] to [C,A,B]	No	No	No
14	RD4	Roll-down on 4 registers	Set [A,B,C,D] to [D,A,B,C]	No	No	No
15	INC	Increment	Set A to A+1	Yes	Yes	No
16	DEC	Decrement	Set A to A-1	Yes	Yes	No
17	ADD	Addition	Set A to A+B	Yes	Yes	No
18	SUB	Subtraction	Set A to A-B	Yes	Yes	No
19	AND	Boolean AND	Set A to A AND B	Yes	Yes (0)	No
1A	ORR	Boolean OR	Set A to A OR B	Yes	Yes (0)	No
1B	XOR	Boolean XOR	Set A to A XOR B	Yes	Yes (0)	No
1C	NOT	Boolean NOT	Set A to NOT A	Yes	Yes (0)	No
1D	NEG	Negative	Set A to -A (=NOT(A)+1)	Yes	Yes	No
1E	CCF	Clear carry-flag	Set CF to 0	No	Yes (0)	No
1F	SCF	Set carry-flag	Set CF to 1	No	Yes (1)	No
20	RRL	Right bit-shift of AL	CF >> AL >> CF	No	Yes	No
21	RRW	Right bit-shift of A	CF >> A >> CF	No	Yes	No
22	RLL	Left bit-shift of AL	CF << AL << CF	No	Yes	No
23	RLW	Left bit-shift of A	CF << A << CF	No	Yes	No
24	BTT	Bit test	Set Z to value of bit of A indicated by B	Yes	No	No
25	CMP	Compare	Set Z to 1 if A=B and set CF to 1 if A<B	Yes	Yes	No
26	JMP	Jump	Set PC to A and IT to 0	No	No	Yes
27	JPZ	Jump if Z	Execute op-code 26 if Z=1	No	No	Yes
28	JNZ	Jump if not Z	Execute op-code 26 if Z=0	No	No	Yes
29	JPC	Jump if CF	Execute op-code 26 if CF=1	No	No	Yes
2A	JNC	Jump if not CF	Execute op-code 26 if CF=0	No	No	Yes

Op-code	Mnemonic	Name	Function	Modify Z	Modify CF	Shift-up stack after
2B	CAL	Call	Shift-down PS stack, set PS0 to PC and IT, set PC to A and IT to 0	No	No	Yes
2C	CLZ	Call if Z	Execute op-code 2B if Z=1	No	No	Yes
2D	CNZ	Call if not Z	Execute op-code 2B if Z=0	No	No	Yes
2E	CLC	Call if CF	Execute op-code 2B if CF=1	No	No	Yes
2F	CNC	Call if not CF	Execute op-code 2B if CF=0	No	No	Yes
30	RET	Return	Set PC and IT to PS0, shift-up PS stack	No	No	No
31	RTZ	Return if Z	Execute op-code 30 if Z=1	No	No	No
32	RNZ	Return if not Z	Execute op-code 30 if Z=0	No	No	No
33	RTC	Return if CF	Execute op-code 30 if CF=1	No	No	No
34	RNC	Return if not CF	Execute op-code 30 if CF=0	No	No	No
35	ENI	Enable interrupt	Set bit of IM indicated by A to 0	No	No	No
36	DSI	Disable interrupt	Set bit of IM indicated by A to 1	No	No	No
37	PU1	Push into RAM stack, step 1	Store A to RAM word addressed by SP	No	No	No
38	PU2	Push into RAM stack, step 2	Set SP to SP+1	No	No	No
39	PO1	Pop from RAM stack, step 1	Set da to SP	No	No	No
3A	PO2	Pop from RAM stack, step 2	Set A to (dhin, dlin), set SP to SP-1	No	No	No
3B	OUT	Set output signal	Set bit of ol indicated by A to CF	No	No	No
3C	INP	Get input signal	Set CF to bit of il indicated by A	No	Yes	No
3D	SSP	Set RAM stack pointer	Set SP to A	No	No	No

Notes:

- For LDH, LDL, PSH and PSL, *xx* represents an 8-bit constant that shall be coded in the low-byte of the instruction word (while the op-code shall be placed in the high-byte).

- For GTH. GTL and GTW, the 13 least-significant bits of `A` are used to address the RAM (in the current ZTH1 implementation). Likewise, for STH, STL and STW, the 13 least-significant bits of `B` are used to address the RAM.
- When executing BTT, `Z` will be set to 0 if `B` > x000F.
- The interrupt 0 ("reset") cannot be masked. Attempts to execute a DSI with `A`=0 will be ignored.
- Although all the instructions are executed in one clock cycle, the "push" and "pop" (use of the RAM stack) are split into two instructions each (PU1 and PU2 for "push", PO1 and PO2 for "pop") that shall be executed one after the other. The completion of a "push" or a "pop" requires therefore two clock cycles.
- For OUT, if `A` is greater than the number of bits of `ol` minus one, then all the bits of `ol` will be set to `CF`. Likewise for INP, if `A` is greater than the number of bits of `il` minus one, `CF` will be set to 1.

7 Rules for ZTH1 coding

Because the instruction bus of the ZTH1 is 16 bits wide, but the op-codes are 8-bit values and some of them have an 8-bit argument, some constraints exist on the coding of ZTH1 programs. First, it is recommended to group op-codes by pair (except for LDH, LDL, PSH and PSL instructions) in the code listing. On the MAX10 implementation, the code is a `.mif` file that describes the content of the instruction ROM. Each line of this file contains an address (from x0000 to x1FFF) and a 16-bit word that corresponds to a pair of op-codes or an op-code and its argument. In the former case, the op-code at the left (in the high-byte) is executed when `IT`=0, and the op-code at the right (in the low-byte) corresponds to `IT`=1.
The constraints are:

- Op-codes of LDH, LDL, PSH and PSL shall always been placed in the high-byte, with their argument in the low-byte. If the instruction just before has been put in the high-byte at the preceding address, a NOP shall be put in the low-byte at that address (this is called a "NOP-padding").
- Instructions to jump to an address (JMP, JPZ, JNZ, JPC, JNC) and to call sub-routines (CAL, CLZ, CNZ, CLC, CNC) set `IT` to 0. Therefore, if the instruction just before the branching point or the beginning of the sub-routine is put in a high-byte, a NOP has to be put in the low-byte (NOP-padding again).

8 Examples of ZTH1 code

The following is an example of `.mif` file used for to program the instruction ROM for a ZTH1. Instruction mnemonics of the op-codes have been put into comments for each line of code. Note that the presence of an interrupt vector at address 0x0008 is ignored by this code.

```
WIDTH=16;
DEPTH=8192;

ADDRESS_RADIX=HEX;
```

```
DATA_RADIX=HEX;

CONTENT BEGIN

-- Fibonacci number calculator
-- Results (from 2) are stored in RAM, infinite loop
0000 : 0100; % LDH x00 %
0001 : 0200; % LDL x00 %
0002 : 0300; % PSH x00 %
0003 : 0201; % LDL x01 %
0004 : 0E00; % DUP ; NOP %
0005 : 1713; % ADD ; RD3 %
0006 : 1510; % INC ; SWP %
0007 : 0A13; % STW ; RD3 %
0008 : 0300; % PSH x00 %
0009 : 0205; % LDL x05 %
000A : 2600; % JMP ; NOP %

[000B..1FFF] : 0000;
END;
```

___oOo___