Tine Alpha
Programmer's manual

Dominik Salvet

January 2017

Contents

1 Characteristics of Tine Alpha

Tine Alpha is processor architecture with the following characteristics:

- 8-bit width,
- Harvard type,
- MISC type,
- can address up to 256 B memory of each instructions and data,
- has instruction pipelining.

1.1 Memory model

For both instruction and data memory is used physically addressed flat model - every address used in instruction is equal to address that will appear on the address bus. The data memory address space includes I/O devices.

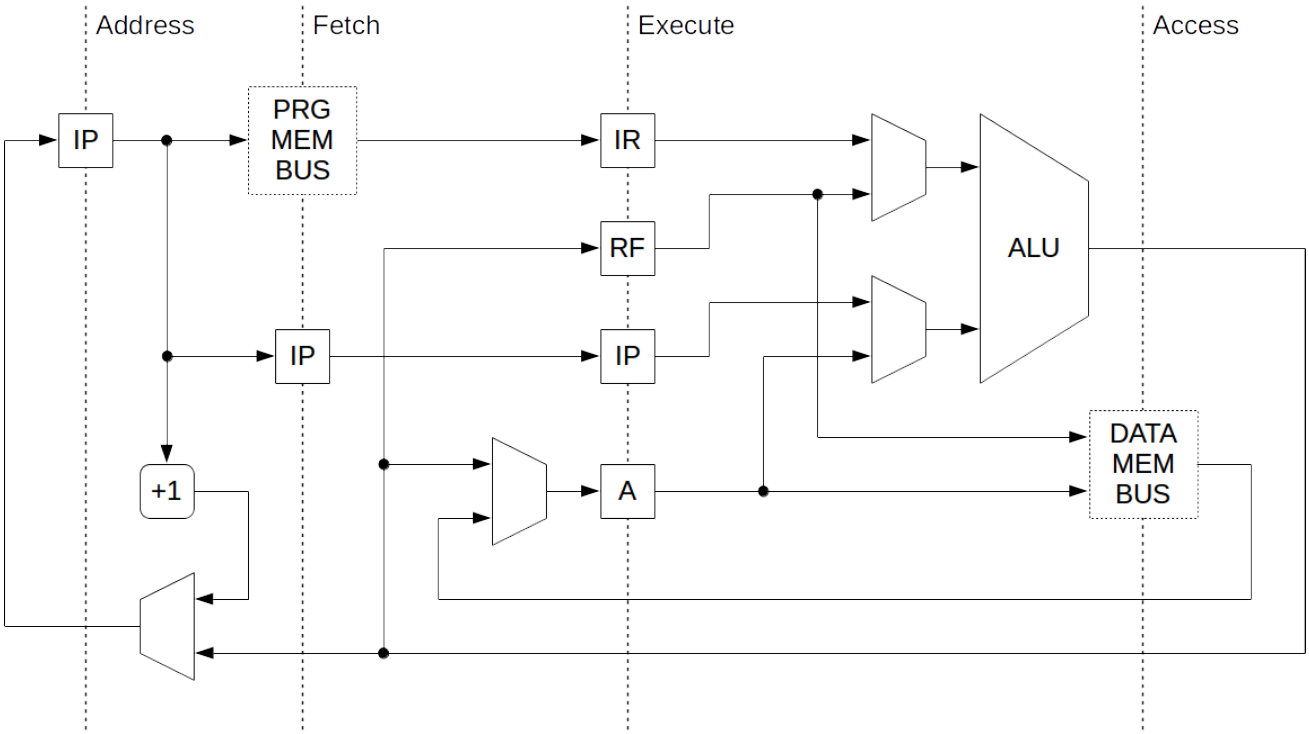
1.2 Register file

Architecture is accumulator-based and so defines one main register known as accumulator (A is short and symbolic variant), which is used in every instruction except JMP instruction. Also there are file of four registers, known as RF (next time it stands for one of these four registers), that are used as second operand of most instructions.

1.3 List of instructions

- `ADD` - addition
- `SUB` - subtraction
- `SL` - set if less
- `SLU` - set if less unsigned
- `NOR` - negation of logical disjunction
- `AND` - logical conjunction
- `SLL` - shift left logical
- `SRL` - shift right logical
- `LI` - load immediate value
- `LIS` - load immediate value scaled
- `CPA` - copy to accumulator
- `CPR` - copy to register
- `LDA` - load to accumulator
- `STA` - store accumulator
- `JMP` - jump
- `JMPA` - jump to accumulator
- `JWL` - jump with link
- `JWLA` - jump with link to accumulator
- `SKNV` - skip never
- `SKIP` - skip the following instruction
- `SKE` - skip if equal
- `SKNE` - skip if not equal
- `SKL` - skip if less
- `SKLE` - skip if less or equal
- `SKG` - skip if greater
- `SKGE` - skip if greater or equal

1.4 Architecture schematic



1.5 Input and output

CLK	in	clock signal source
RST	in	reset signal
IMEM_IN[8]	in	instruction memory read data
IMEM_RD	out	instruction memory read enable
IMEM_ADDR[8]	out	instruction memory address
DMEM_IN[8]	in	data memory read data
DMEM_WR	out	data memory write enable
DMEM_ADDR[8]	out	data memory address
DMEM_OUT[8]	out	data memory written data

2 Instruction set

This chapter contains description of every executable instruction. Every description of instruction is explaining dependencies of instruction word and its final binary form as well as behavior of individual instructions. Every instruction lasts one clock signal, unless otherwise stated.

2.1 Used conventions

The following labels `addr4`, `imm4`, `uimm4` that will be used later, are representing any natural number in the range:

- `addr4` - from -2^3 to $2^3 - 1$,
- `imm4` - from -2^3 to $2^3 - 1$,
- `uimm4` - from 0 to $2^4 - 1$.

These numbers can be expressed in various radix:

- Decimal - <digits(0-9)> or <digits(0-9)>D or <digits(0-9)>d:
  - it is possible to demonstrate the value sign, even if it is positive,
- Hexadecimal - <digits(0-F or 0-f)>H or <digits(0-F or 0-f)>h,
- Octal - <digits(0-7)>O or <digits(0-7)>o,
- Binary - <digits(0,1)>B or <digits(0,1)>b.

2.2 Instructions with immediate value

This type of instructions is using 4-bit immediate value contained inside its instruction word as a operand. This value is always extended to 8-bit width before enter ALU. So the immediate value can be interpreted as follows:

- signed value (always two’s complement format):
  - `addr4`, `imm4`,
- unsigned value (always direct binary value):
  - `uimm4`.

2.2.1 Arithmetic and logic

The most common instructions that perform arithmetic and logic computing. The first operand is always A register, also the result of operation is always written here. The second operand is largely 4-bit signed value. Note that logic operations don’t make the difference, logic operations always execute on signed extended value. The only exception that is using unsigned value of immediate value is SLU instruction.

Instruction word structure

7	6	5	4	3	2	1	0
1	type			(u)imm4			

List of instructions

	type		mnemonic	C-like language
0	0	0	AND imm4	A = A & imm4;
0	0	1	NOR imm4	A = ~(A | imm4);
0	1	0	SLL imm4	A = A << (imm4 & 0x07);
0	1	1	SRL imm4	A = A >> (imm4 & 0x07);
1	0	0	SLU uimm4	A = A < uimm4;
1	0	1	SL imm4	A = A < imm4;
1	1	0	SUB imm4	A = A - imm4;
1	1	1	ADD imm4	A = A + imm4;

Description of instructions

`AND imm4`

- Performs bitwise logical conjunction between A register and 4-bit signed value extended to 8-bit width, the result is written to A register.

`NOR imm4`

- Performs logical negation of bitwise logical disjunction between A register and 4-bit signed value extended to 8-bit width, the result is written to A register.

`SLL imm4`

- Performs logical shift to the left of value in A register by unsigned value of 3 lower bits of 4-bit immediate value operand, the result is written to A register.

`SRL imm4`

- Performs logical shift to the right of value in A register by unsigned value of 3 lower bits of 4-bit immediate value operand, the result is written to A register.

`SLU uimm4`

- Performs value compare of unsigned arithmetics with 4-bit unsigned value and A register. If A register value is less, LSB of A register is set and other bits are cleared. Otherwise, all bits of A register are cleared.

`SL imm4`

- Performs value compare of signed arithmetics with extended 4-bit signed value and A register. If A register value is less, LSB of A register is set and other bits are cleared. Otherwise, all bits of A register are cleared.

`SUB imm4`

- Performs arithmetic subtraction of 4-bit signed value from A register, the result is written to A register.

`ADD imm4`

- Performs arithmetic addition of 4-bit signed value and A register, the result is written to A register.

2.2.2 Loading constants

Instructions that load 4-bit unsigned values to A register. There are two mods for that. The first one loads 4-bit binary string from instruction word to lower part of A and clears the higher. The second, scaled mode, loads 4-bit binary string to higher part of A and clears the lower.

Instruction word structure

7	6	5	4	3	2	1	0
0	1	1	t	uimm4			

List of instructions

type	mnemonic	C-like language
0	`LI uimm4`	`A = uimm4;`
1	`LIS uimm4`	`A = uimm4 << 4;`

Description of instructions

`LI uimm4`

- Loads immediate value to A register. 4-bit unsigned value is 4-bit lower part and it is extended by 4-bit higher part of zeros. These parts are composed together and written to A register.

`LIS uimm4`

- Loads scaled immediate value to A register. 4-bit lower part is containing zeros and 4-bit higher part is represented by 4-bit unsigned value. These parts are composed together and written to A register.

2.2.3 Unconditional jumps

Unconditional jumps instructions provide a way to change address of executed instructions. It can be also linked the address of instruction that follows jump instruction. This can be used for calling functions. Immediate value is interpreted as signed value that represents address displacement from address of currently executing jump instruction.

Instruction word structure

7	6	5	4	3	2	1	0
0	1	0	t	imm4			

List of instructions

type	mnemonic	C-like language
0	`JWL imm4`	`A = IP + 1; IP = IP + imm4;`
1	`JMP imm4`	`IP = IP + imm4;`

Description of instructions

`JWL imm4`

- Performs unconditional jump to change the following instruction's address. The final address is calculated by adding address of actual instruction (IP register value) and signed 4-bit value. Since the immediate value is signed, it is also possible to perform jump backwards, as it is required. The following instruction's address (IP + 1) is written to A register.
- This instruction lasts 3 clock signals to perform.

`JMP imm4`

- Performs unconditional jump to change the following instruction's address. The final address is calculated by adding address of actual instruction (IP register value) and signed 4-bit value. Since the immediate value is signed, it is also possible to perform jump backwards, as it is required.
- This instruction lasts 3 clock signals to perform.

2.3 Instructions working with registers

This type of instructions is using only registers to achieve the result.

2.3.1 Arithmetic and logic

The most common instructions that perform arithmetic and logic computing. The first operand is always A register, also the result of operation is always written here. The second operand is RF register.

Instruction word structure

7	6	5	4	3	2	1	0
0	0	1	type			Ri	

List of instructions

type			mnemonic	C-like language
0	0	0	AND Ri	A = A & Ri;
0	0	1	NOR Ri	A = ~(A | Ri);
0	1	0	SLL Ri	A = A << (Ri & 0x07);
0	1	1	SRL Ri	A = A >> (Ri & 0x07);
1	0	0	SLU Ri	A = A < Ri;
1	0	1	SL Ri	A = (signed) A < Ri;
1	1	0	SUB Ri	A = A - Ri;
1	1	1	ADD Ri	A = A + Ri;

Description of instructions

`AND Ri`

- Performs bitwise logical conjunction between A register and RF register, the result is written to A register.

`NOR Ri`

- Performs logical negation of bitwise logical disjunction between A register and RF register, the result is written to A register.

`SLL Ri`

- Performs logical shift to the left of A register by value of 3 lower bits of RF register, the result is written to A register.

`SRL Ri`

- Performs logical shift to the right of A register by value of 3 lower bits of RF register, the result is written to A register.

`SLU Ri`

- Performs value compare of unsigned arithmetics between A register and RF register. If A register value is less, LSB of A register is set and other bits are cleared. Otherwise, all bits of A register are cleared.

`SL Ri`

- Performs value compare of signed arithmetics between A register and RF register. If A register value is less, LSB of A register is set and other bits are cleared. Otherwise, all bits of A register are cleared.

`SUB Ri`

- Performs arithmetic subtraction of RF register from A register, the result is written to A register.

`ADD Ri`

- Performs arithmetic addition of RF register and A register, the result is written to A register.

2.3.2 Register value transferring

The instructions that provide to move register values between themselves. The transfer can be realized only by using A register.

Instruction word structure

7	6	5	4	3	2	1	0
0	0	0	1	1	t	Ri	

List of instructions

type	mnemonic	C-like language
0	`CPA Ri`	`A = Ri;`
1	`CPR Ri`	`Ri = A;`

Description of instructions

`CPA Ri`

- Copies the value of RF register to A register.

`CPR Ri`

- Copies the value of A register to RF register.

2.3.3 Accessing the memory

The only way to communicate with the memory and I/O devices. RF register is always used as a pointer to the memory. When storing value, A register is holding the value that will be written to memory. When loading value, read value will be written to A register.

Instruction word structure

7	6	5	4	3	2	1	0
0	0	0	1	0	t	Ri	

List of instructions

type	mnemonic	C-like language
0	`LDA Ri`	`A = *Ri;`
1	`STA Ri`	`*Ri = A;`

Description of instructions

`LDA Ri`

- Load value from memory pointed by RF register to A register.
- This instruction lasts 2 clock signals to perform.

`STA Ri`

- Store value of A register to memory pointed by RF register.

2.3.4 Unconditional jumps

Unconditional jumps instructions provide a way to change address of executed instructions. It can be also linked the address of instruction that follows jump instruction. This can be used for calling functions. Jump will be performed to address pointed by value in A register.

Instruction word structure

7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	t

List of instructions

type	mnemonic	C-like language
0	JWLA	T = A; A = IP + 1; IP = T;
1	JMPA	IP = A;

Description of instructions

`JWLA`

- Performs unconditional jump to address pointed by A register value. Also the following instruction's address (IP + 1) is written to A register.
- This instruction lasts 3 clock signals to perform.

`JMPA`

- Performs unconditional jump to address pointed by A register value.
- This instruction lasts 3 clock signals to perform.

2.3.5 Conditional instruction skips

Conditional instruction skips provide the only way to achieve conditional program execution. There are several arithmetic conditions to provide conditional instruction skip. These conditions are always testing the A register value compared to zero and if condition is met, the following instruction will not be executed. Also there is SKNV instruction that can represent no operation instruction.

Instruction word structure

7	6	5	4	3	2	1	0
0	0	0	0	0	type		

List of instructions

type			mnemonic	C-like language
0	0	0	SKNV	`if (0) IP = IP + 1;`
0	0	1	SKIP	`if (1) IP = IP + 1;`
0	1	0	SKNE	`if (A != 0) IP = IP + 1;`
0	1	1	SKE	`if (A == 0) IP = IP + 1;`
1	0	0	SKL	`if (A < 0) IP = IP + 1;`
1	0	1	SKLE	`if (A <= 0) IP = IP + 1;`
1	1	0	SKG	`if (A > 0) IP = IP + 1;`
1	1	1	SKGE	`if (A >= 0) IP = IP + 1;`

Description of instructions

All following instructions last 2 clock signals to perform if they successfully skipped the following instruction, otherwise 1 clock signal is required to complete.

`SKNV`

- Never skips the following instruction. It can represents no operation.

`SKIP`

- Skips the following instruction.

`SKNE`

- Skips the following instruction if A register value is not zero.

`SKE`

- Skips the following instruction if A register value is zero.

`SKL`

- Skips the following instruction if A register value is less than zero.

`SKLE`

- Skips the following instruction if A register value is less than zero or zero.

`SKG`

- Skips the following instruction if A register value is greater than zero.

`SKGE`

- Skips the following instruction if A register value is greater than zero or zero.

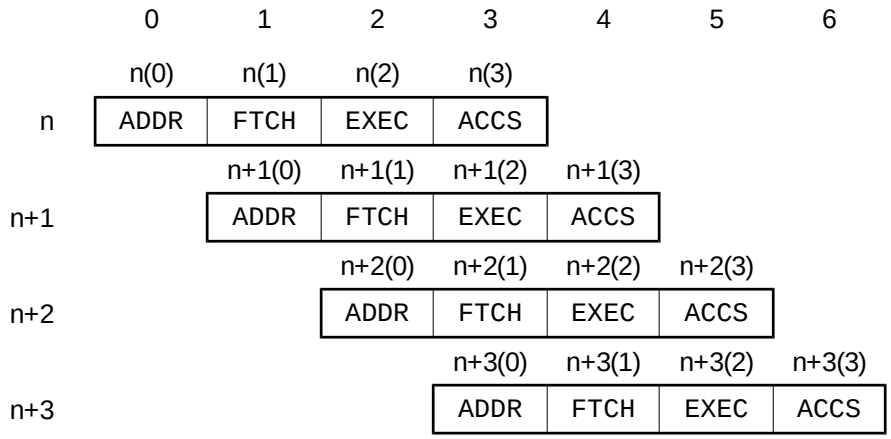
2.4 Program example

```
; == Unsigned software multiply ==
; perform multiply of unsigned numbers
; R1 = R2 * R3

;address  mnemonic  binary         comment

 00H:     LI 13     01101101
 01H:     CPR R3    00011111       ; load R3 with 13
 02H:     LI 7      01100111
 03H:     CPR R2    00011110       ; load R2 with 7
 04H:     LI 0      01100000
 05H:     CPR R1    00011101       ; load R1 with 0

 06H:     CPA R2    00011010
 07H:     SKNE      00000010       ; test if R2 is 0
 08H:     JMP +5    01010101       ; if so, calculation complete (1)

 09H:     SLL 7     10100111       ; else
 0AH:     SKNE      00000010       ; check if R2 is even
 0BH:     JMP +7    01010111       ; if so, skip addition (2)
 0CH:     JMP +3    01010011       ; else, jump to addition (3)

 0DH:     LIS 2     01110010       ; (1) load jump complete address
 0EH:     JMPA      00001111       ; jump to it

 0FH:     CPA R1    00011001       ; (3)
 10H:     ADD R3    00111111       ; add R1 and R3
 11H:     CPR R1    00011101       ; write back to R1

 12H:     CPA R3    00011011       ; (2)
 13H:     SLL 1     10100001       ; double R3
 14H:     CPR R3    00011111       ; write back
 15H:     CPA R2    00011010
 16H:     SRL 1     10110001       ; half of R2
 17H:     CPR R2    00011110       ; write back
 18H:     LI 6      01100110       ; load jump address to iterate
 19H:     JMPA      00001111       ; iterate

 20H:     JMP 0     01010000       ; calculation is complete
```

The program is only for demonstrate the possibilities. Used algorithm is not intended to be optimized.

3 Instruction processing

Each instruction takes four clock signals to perform and is overlapped with three others instructions. Then under ideal conditions every clock signal is completed one instruction. This instruction processing is known as pipelining.



3.1 Instruction stages

Every instruction must go through four stages of pipeline to be performed. This stages are pretty independent between themselves and so it is possible to process four instructions at the moment at most. These are the stages:

- ADDR - calculating the address of the following instruction,
- FTCH - fetching the instruction from instruction memory,
- EXEC - decoding and performing arithmetic and logic operations,
- ACCS - accessing to data memory.

3.2 Pipeline stalls

Some instructions can cause pipeline stalls. It means that it is not possible to perform them in one clock signal and it is required some additional handling. Pipelined instruction processing can negatively affect these instructions:

- Any jump instruction (3 clock signals to perform):
  - it is required to fetch new instruction from just calculated address and wait to fill up the pipeline,
- Load data from memory (2 clock signals to perform):
  - at the first it is required to write data’s address on data memory address bus and then it is required to fetch the data,
- Any skip instruction (1 or 2 clock signals to perform)
  - if skip is successful, the following instruction is is replaced with SKNV (no operation) and it is performed, then the real effect comes when it is fetched the next instruction.

List of Symbols and Abbreviations

MISC	Minimal Instruction Set Computer
I/O	Input/Output
A	Accumulator
RF	Register File
ALU	Arithmetic Logic Unit
LSB	Least Significant Bit
IP	Instruction Pointer