Computron VM is virtual computer machine, supported by Computron interface, which looks as follows

Computron has two periferals:

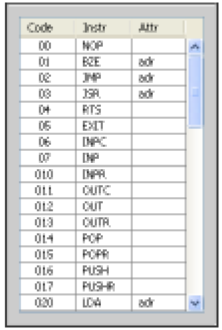
- Alphanumeric screen, and
- ODFC Programming Device

Computron Interface and Screen are sufficient for binary programming.

ODFC Programming device is interface extension, which allows to enter data in Octal, Decimal, Floating Point and Character formats, as well as to load and save binary programs.

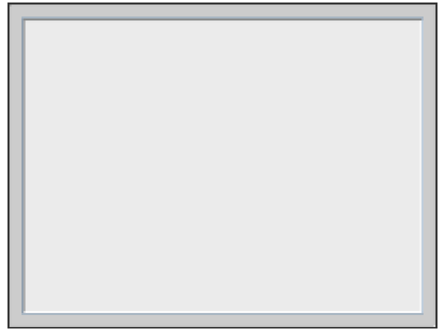
Table of instructions is helpful when octal code of an instruction is needed, using ODFC device. The meaning – operational semantics of instructions is defined in detail in Help file, i.e. in this file.

ODFC Programming Device



Code	Instr	Attr
00	NOP	
01	BZE	adr
02	JMP	adr
03	JSR	adr
04	RTS	
05	EXIT	
06	INPC	
07	INP	
010	INPR	
011	OUTC	
012	OUT	
013	OUTR	
014	POP	
015	POPR	
016	PUSH	
017	PUSHR	
020	LDA	adr

Assembler Syntactic Help



Computron Screen

Computron Architecture

is word (1 word=16bits) oriented architecture, which consists of Central Processing Unit (**CPU**) and main Memory (**M**).

CPU has four single-word registers

PC	Program Counter
SP	Stack Pointer
A	Accumulator
X	Index Register

and one double-word register

R Real Numbers Register: RH is the higher and RL is the lower word of R.

Computron Memory **M** is the array of 64K, i.e 65536 single-word registers M[0], M[1] … M[177777].

Note: The indices of array M are memory addresses. The lowest address is 0, and the highest address is 65635 (177777 in octal representation). In CPU, a memory register M[PC] is accessed automatically based on the value of an address stored in PC. A real number (in float type representation) is stored in two subsequent registers (cells) of memory, RL at address addr and RH at address addr+1.

Floating point numbers are in range

-Infinity > -3.4028235E38 ... -1.4E-45 , 0 , 1.4E45 ... 3.4028235E38 < Infinity

For a floating point value limits stored in R, RH and RL octal values are as follows.

R	RH	RL
- Infinity	177600	000000
-3.4028235E38	177577	177777
-1.4E-45	100000	000001
0	000000	000000
1.4E-45	000000	000001
3.4028235E38	077577	177777
Infinity	077600	000000

Note: Floating-point numbers use the IEEE (Institute of Electrical and Electronics Engineers) format. Single-precision values with float type have 4 bytes, consisting of a sign bit, an 8-bit excess-127 binary exponent, and a 23-bit mantissa. The mantissa represents a number between 1.0 and 2.0. Since the high-order bit of the mantissa is always 1, it is not stored in the number. This representation gives a range of approximately 3.4E–38 to 3.4E+38 for type float.

Computron Instruction Set

Instruction Categories

1. No Operation Instruction
2. Control Flow Instructions
3. Input from keyboard Instructions
4. Output to screen Instructions
5. Stack PUSH and POP Instructions (single and double word)
6. Load accumulator A from memory
7. Load register R from memory
8. Store accumulator A and register R to memory
9. Load/Store index register X and stack pointer SP from/to memory
10. Boolean operations
11. Comparisons of words using register A and double words using register R
12. Arithmetic operations on integer and real numbers

Note: If instruction operational code is at address PC, then attribute (adr or val) of each attributed instruction is at address PC+1, and then adr = M[PC + 1], or val = M[PC + 1].

No Operation Instruction			
Octal Code	Name	Attrib	Operational Semantics
000000	NOP		PC := PC + 1;
No operation			

Control Flow Instructions			
Octal Code	Name	Attrib	Operational Semantics
000001	BZE	adr	if (A = 0) then PC := adr else PC := PC + 2;
Branch if zero: conditional jump to address adr			
Octal Code	Name	Attrib	Operational Semantics
000002	JMP	adr	PC := adr;
Non-conditional jump to address adr			
Octal Code	Name	Attrib	Operational Semantics
000003	JSR	adr	M[SP] := PC + 2; SP := SP + 1; PC := adr;
Jump to subroutine			
Octal Code	Name	Attrib	Operational Semantics
000004	RTS		SP := SP – 1; PC := M[SP];
Return from subroutine			
Octal Code	Name	Attrib	Operational Semantics
000005	EXIT		running_signal := false;
Exit – stops machine execution			

Input from keyboard Instructions			
Octal Code	Name	Attrib	Operational Semantics
000006	INPC		A := typed_character_ascii_code(); PC := PC + 1;
Input character from keyboard			
Octal Code	Name	Attrib	Operational Semantics
000007	INP		A := typed_integer_value(); PC := PC + 1;
Input integer from keyboard			
Octal Code	Name	Attrib	Operational Semantics
000010	INPR		R := typed_floating_point_value(); PC := PC + 1;
Input real number from keyboard			

Output to screen Instructions			
Octal Code	Name	Attrib	Operational Semantics
000011	OUTC		display_character_of_code_in (A); PC := PC + 1;
Output character to screen			
Octal Code	Name	Attrib	Operational Semantics
000012	OUT		display_integer_value_in (A); PC := PC + 1;
Output integer to screen			
Octal Code	Name	Attrib	Operational Semantics
000013	OUTR		display_floating_point_value_in (R); PC := PC + 1;
Output real number to screen			

Stack PUSH and POP Instructions (single and double word)			
Octal Code	Name	Attrib	Operational Semantics
000014	POP		SP := SP – 1; A := M[SP]; PC := PC + 1;
Pop word from stack to accumulator A			
Octal Code	Name	Attrib	Operational Semantics
000015	POPR		SP := SP – 2; R := (M[SP], M[SP + 1]); PC := PC + 1;
Pop real number from stack to register R;			
Octal Code	Name	Attrib	Operational Semantics
000016	PUSH		M[SP] := A; SP := SP + 1; PC := PC + 1;
Push word in A on the stack			
Octal Code	Name	Attrib	Operational Semantics
000017	PUSHR		(M[SP], M[SP + 1]) := R; SP := SP + 2; PC := PC + 1;
Push real number in R on the stack			

Load accumulator A from memory			
Octal Code	Name	Attrib	Operational Semantics
000020	LDA	adr	A := M[adr]; PC := PC + 2;
Load value to A from memory cell at address adr			
Octal Code	Name	Attrib	Operational Semantics
000021	LDA’M	val	A := val; PC := PC + 2;
Load value val to A from memory address PC+1 (immediate addressing)			
Octal Code	Name	Attrib	Operational Semantics
000022	LDA’I	adr	A := M[M[adr]]; PC := PC + 2;
Load value to A from memory cell at address M[adr] (indirect addressing)			
Octal Code	Name	Attrib	Operational Semantics
000023	LDA’X	val	A := M[val + X]; PC := PC + 2;
Load value to A from memory cell at address val+X (indexed addressing)			

<table>
<tr><td colspan="4">Load register R from memory</td></tr>
<tr><td>Octal Code</td><td>Name</td><td>Attrib</td><td>Operational Semantics</td></tr>
<tr><td>000024</td><td>LDR</td><td>adr</td><td>R := (M[adr], M[adr + 1]);   PC := PC + 2;</td></tr>
<tr><td colspan="4">Load value from memory cell at adr to RL, and value from cell at addr+1 to RH</td></tr>
<tr><td>Octal Code</td><td>Name</td><td>Attrib</td><td>Operational Semantics</td></tr>
<tr><td>000025</td><td>LDR’I</td><td>adr</td><td>R := (M[M[adr]], M[M[adr] + 1]);   PC := PC + 2;</td></tr>
<tr><td colspan="4">Load value from memory cell at M[adr] to RL, and from M[adr] + 1] to RH (indirect addressing)</td></tr>
<tr><td colspan="4">Store accumulator A and register R to memory</td></tr>
<tr><td>Octal Code</td><td>Name</td><td>Attrib</td><td>Operational Semantics</td></tr>
<tr><td>000026</td><td>STA</td><td>adr</td><td>M[adr] := A;   PC := PC + 2;</td></tr>
<tr><td colspan="4">Store value from accumulator A to memory cell at address adr</td></tr>
<tr><td>Octal Code</td><td>Name</td><td>Attrib</td><td>Operational Semantics</td></tr>
<tr><td>000027</td><td>STA’I</td><td>adr</td><td>M[M[adr]] := A;   PC := PC + 2;</td></tr>
<tr><td colspan="4">Store value from accumulator A to memory cell at address M[adr] (indirect addressing)</td></tr>
<tr><td>Octal Code</td><td>Name</td><td>Attrib</td><td>Operational Semantics</td></tr>
<tr><td>000030</td><td>STR’I</td><td>adr</td><td>(M[M[adr]], M[M[adr] + 1]) := R;   PC := PC + 2;</td></tr>
<tr><td colspan="4">Load value from RL to cell at address M[adr], and from RH to M[adr] + 1] (indirect addressing)</td></tr>
</table>

Load/Store index register X and stack pointer SP from/to memory			
Octal Code	Name	Attrib	Operational Semantics
000031	LDX	adr	X := M[adr]; PC := PC + 2;
Load from memory to index register X			
Octal Code	Name	Attrib	Operational Semantics
000032	STX	adr	M[adr] := X; PC := PC + 2;
Store from index register X to memory			
Octal Code	Name	Attrib	Operational Semantics
000033	LDS	adr	SP := M[adr]; PC := PC + 2;
Load fro memory to stack pointer SP			
Octal Code	Name	Attrib	Operational Semantics
000034	STS	adr	M[adr] := SP; PC := PC + 2;
Store from stack pointer SP to memory			

Boolean operations			
Octal Code	Name	Attrib	Operational Semantics
000035	OR	adr	if ((A = 0) && (M[adr] = 0)) then A := 0 else A := 1; PC := PC + 2;
Disjunction operation			
Octal Code	Name	Attrib	Operational Semantics
000036	AND	adr	if ((A = 1) && (M[adr] = 1)) then A := 1 else A := 0; PC := PC + 2;
Conjunction operation			
Octal Code	Name	Attrib	Operational Semantics
000037	NOT		if (A = 1) then A := 0 else A := 1; PC := PC + 1;
Negation operation			

Comparisons of words using register A and double words using register R			
Result is stored in A, where false is represented by 0 and true by 1.			
Octal Code	Name	Attrib	Operational Semantics
000040	EQ	adr	if (A = M[adr]) then A := 1 else A := 0; PC := PC + 2;
000041	NE	adr	if (A != M[adr]) then A := 1 else A := 0; PC := PC + 2;
000042	LT	adr	if (int (A) < int (M[adr])) then A := 1 else A := 0; PC := PC + 2;
000043	LE	adr	if (int (A) <= int (M[adr])) A := 1 else A = 0; PC := PC + 2;
000044	GT	adr	if (int (A) > int (M[adr])) A := 1 else A := 0; PC := PC + 2;
000045	GE	adr	if (int (A) >= int (M[adr])) A := 1 else A := 0; PC := PC + 2;
000046	EQR	adr	if (R = (M[adr], M[adr + 1])) then A := 1 else A := 0; PC := PC + 2;
000047	NER	adr	if (R != (M[adr], M[adr + 1])) then A := 1 else A := 0; PC := PC + 2;
000050	LTR	adr	if (R < (M[adr], M[adr + 1])) then A := 1 else A := 0; PC := PC + 2;
000051	LER	adr	if (R <= (M[adr], M[adr + 1])) A := 1 else A := 0; PC := PC + 2;
000052	GTR	adr	if (R > (M[adr], M[adr + 1])) A := 1 else A := 0; PC := PC + 2;
000053	GER	adr	if (R >= (M[adr], M[adr + 1])) then A := 1 else A := 0; PC := PC + 2;

Arithmetic operations on integer and real numbers			
Octal Code	Name	Attrib	Operational Semantics
000054	ADD	adr	A = int (A) + int (M[adr]); PC := PC + 2;
000055	ADD'M	val	A := int(A) + int (val); PC := PC + 2;
000056	SUB	adr	A := int (A) – int (M[adr]); PC = PC + 2;
000057	SUB'M	val	A := int (A) – int (val); PC := PC + 2;
000060	MUL	adr	A := int (A) * int (M[adr]); PC := PC + 2;
000061	DIV	adr	A := int (A) / int (M[adr]); PC := PC + 2;
000062	NEG		A := - int (A); PC := PC + 1;
000063	ADDR	adr	R := R + (M[adr], M[adr + 1]); PC := PC + 2;
000064	SUBR	adr	R := R - (M[adr], M[adr + 1]); PC := PC + 2;
000065	MULR	adr	R := R * (M[adr], M[adr + 1]); PC := PC + 2;
000066	DIVR	adr	R := R / (M[adr], M[adr + 1]); PC := PC + 2;
000067	NEGR		R := - R; PC := PC + 1;

Generating Computron code in C/C++

```
// CVMDataGen.cpp : Defines the entry point for the console application.
//
// Computron Data Generation
//
// -----------------------------------------------------------
// Representative Example, how to generate
//
// 1. operation code of instructions
// 2. (code of) characters, cardinal and octal numbers
// 3. signed integer numbers, and
// 4. real numbers in floating point representation
//
// Output NON EXECUTABLE binary file
// 1. CAN BE LOADED in Computron memory from any initial address
// 2. Binary representation of all data can be checked by incrementing initial address.
// ------------------------------------------------------------

#include "stdafx.h"
#include <stdio.h>
#include <conio.h>
#include <string.h>
```

```
FILE *outstream; // output file stream

char outf[30]; errno_t err; int items;

// -----------------------------------------------------------
// Operation Code OPvalue is a value of enum OP_Codes          -
// converted to unsigned short type representation             -
// -----------------------------------------------------------

enum OP_Code
	{	NOP, BZE, JMP, JSR, RTS, EXIT,
		INPC, INP, INPR, OUTC, OUT, OUTR,
		POP, POPR, PUSH, PUSHR,
		LDA, LDAM, LDAI, LDAX,
		LDR, LDRI, STA,
		STAI, STRI,
		LDX, STX, LDS, STS,
		OR, AND, NOT,
		EQ, NE, LT, LE, GT, GE,
		EQR, NER, LTR, LER, GTR, GER,
		ADD, ADDM, SUB, SUBM,
		MUL, DIV, NEG,
		ADDR, SUBR, MULR, DIVR, NEGR
	};
unsigned short OPvalue;
```

```
// -----------------------------------------------------------
// Character codes Cvalue (0..255),
// Octal numbers Ovalue (\0 .. \177777) and
// Cardinal numbers Uvalue (0 .. 65535)
// are represented by unsigned short type
// -----------------------------------------------------------

// Examples:

unsigned short Cvalue = (unsigned short) 'a';
unsigned short Ovalue = (unsigned short) 0177777;
unsigned short Uvalue = 65535;

// -----------------------------------------------------------
// Signed integers Ivalue (-32768 .. 32767),
// are represented by short type and
// converted to unsigned short type
// -----------------------------------------------------------

unsigned short Ivalue = -32767;
```

```c
// --------------------------------------------------------------
// Real Numbers --- represented by Real_Type ---------------------
// --------------------------------------------------------------
// Real_Type is polymorph type for conversion of real value Rvalue
// in Floating Point Format (Rvalue.R) and Double Word Format
// (Rvalue.Dword.RL, Rvalue.Dword.RH). The next equation holds:
//
//              Rvalue = (Rvalue.Dword.RL, Rvalue.Dword.RH)
// --------------------------------------------------------------

typedef union RealRegister {
        float R;
        struct {
                unsigned short RL;
                unsigned short RH;
        } Dword;
} Real_Type;

Real_Type Rvalue;

// Example of two possible ways, how to determine real number
// Rvalue.R = (float) 2.5e31 ;
// or
// Rvalue.Dword.RL=65535; Rvalue.Dword.RH=32767;
```

```
// =============================================================
// Concluding, we need just two procedures for generating Computron code:
//
// void putWord(unsigned short argument);
// and
// void putReal(float argument);
//

void putWord(unsigned short arg)
{
	items = fwrite(&arg, sizeof(short), 1, outstream);
};

void putReal(float arg)
{	Real_Type Rvalue;
	Rvalue.R = arg;
	putWord(Rvalue.Dword.RL);
	putWord(Rvalue.Dword.RH);
};
```

```
int _tmain(int argc, _TCHAR* argv[])
{ char c;
// ==================================================================
// Generating Computron data to output binary file
// 1. Binary file is a file of unsigned short values
// 2. and it can be loaded by ODFC Programming device
// ==================================================================
// Open output file - file of unsigned short values
        printf("\nOutput binary file name: "); scanf("%20s", &outf);
        if( (err = fopen_s( &outstream, outf, "w+b" )) != 0 )
                printf( "Output file %s was not opened\n", outf );
        else {
                // Generating Characters
                putWord('A'); putWord('B'); putWord('Z');
                putWord(012); // newline: (12 octal, 10 decimal)

                // Generating Instruction Operations (without arguments)
                putWord(NOP); putWord(LDAM); putWord(NEGR);

                // Generating Octal Numbers
                putWord(0377); putWord(0177777);

                // Generating Cardinal (Unsigned Decimal) Numbers
                putWord(0); // Cardinal Minimum
                putWord(255); // Cardinal Sample
                putWord(65535); // Cardinal Maximum
```

```
		// Generating Integer Numbers
		putWord(-32768); // Integer Minimum   100000 octal
		putWord(-1);     // Integer Sample    111111 octal
		putWord(0);      // Integer Sample    000000 octal
		putWord(1);      // Integer Sample    000001 octal
		putWord(32767);  // Integers Maximum  011111 octal

		// Generating Real Numbers in Floating Point Representation
		putWord(0000000); putWord(0177600); // -Infinity
		putReal((float) -3.4028235E38); // Minimum Real Number
		putReal((float) -1.4E-45); // Negative Real Number nearest to Zero
		putReal((float) 0.0);
		putReal((float) 1.4E-45); // Positive Real Number nearest to Zero
		putReal((float) 3.4028235E38); // Maximum Real Number
		putWord(0000000); putWord(0077600); // Infinity

		fclose(outstream);
	}
	printf("\nOutput binary file generated\n");
	return 0;
}
```

ODFC Programming Device

ODFC Input Data Formats

Octal: '0' {Octal_Digit}
Decimal: [- |+] Decimal_Digit { Decimal_Digit }
Floating Point: Decimal . [Decimal] [e [+|-] Decimal] | Decimal e [+|-] Decimal
Character-printable: Cx{x}, where C is printable character entered and x is any other character,
non printable: octal or unsigned decimal number, in range 0.. 255

Loading CPU and Memory Registers

After entering a data value, press Enter and then one of accessible Computron registers, as follows.

Data Form	PC	A	SP	X	RH	RL	R	M	M-R	Begin	End
Octal	●	●	●	●	●	●		●		●	●
Decimal	●	●	●	●				●			
Floating Point							●		●		
Character		●						●			

Note: M-R is double word memory at M[PC,PC+1]

<u>Loading and saving binary programs</u>

1. To load a binary program generated by some compiler, set starting memory address *addrB* – using Begin button, press Load button and you can load a program from external binary file to Computron memory beginning at address *addrB*.

2. To save a binary program stored in Computron memory from *addrB* to *addrE*, set *addrB* using Begin button and *addrE* using End Button. Then press Save Button and save your binary program to external file (total of *addrE-addrB+1* words in length).

Note: ODFC Buttons Incr, Decr, and Run have the same effect as those in Computron interface.
However, ODFC Load button loads a whole program (or binary data) from external file, while Computron Load loads just one word from memory to a selected register.

Have a good luck!