

Central Processing Unit Simulation

Version v2.5 (July 2005)

Charles André
University Nice-Sophia Antipolis



1 Table of Contents

1	Table of Contents	3
2	Overview	5
3	Installation	7
4	The CPU Simulator (v2.0)	9
4.1	Launching the application.....	9
4.2	Selecting	9
4.3	The Control Window	10
4.3.1	Menus	11
4.3.2	Registers	12
4.3.3	Control button.....	13
4.3.4	Preferences.....	13
4.4	The Memory Window	14
4.5	The Performance Windows	14
4.6	Symbol Table Window.....	15
5	Manual Execution	17
6	Accumulator-based Machine	19
6.1	Arithmetic and Logical Unit.....	19
6.2	Instruction Set	19
6.3	Example of the gcd computation	20
7	Stack-based Machine	23
7.1	Arithmetic and Logical Unit.....	23
7.2	Instruction Set	23
7.2.1	Example of the gcd computation	24

2 Overview

Purpose: Educational.

- Initiation to low-level programming and CPU principles.
- Explore various architectures of CPU through their programming and their step-by-step execution.
- Deliberate limitation: it supports elementary programs only.

Realization:

- Pure Tcl-Tk implementation.
- Requires Tcl-Tk, version 8.4 or better, and the BWidget ToolKit, version 1.7 or better.
- Tested on Window2000, Windows XP and Linux.

Contact: Written by Charles ANDRÉ, Electrical Engineering Department, Faculty of Sciences, University of NICE-SOPHIA ANTIPOLIS. andre@unice.fr

Caveats: The look and Feel of the widgets is dependent on the operating system. The pictures given in this text are screen copies captured under Windows 2000, and have been generated by version 2.0 of the software.

3 Installation

Be sure that Tcl-Tk, version 8.4 or higher, and BWidget, version 1.7 or higher are installed.

1. If the environment variable **SOFTDIR** is defined, go to 5
2. Create a directory in which all software from C. André will be installed
3. Set the environment variable **SOFTDIR** to the path of this directory
4. Prepend your **PATH** environment variable with **\$SOFTDIR/bin** (or **%SOFTDIR%\bin** for Windows)
5. Copy the distribution file **archi.2.x.tgz** in **\$SOFTDIR** (replace **x** by the current minor version number)
6. Execute: **cd \$SOFTDIR**
7. Execute: **gunzip -c archi.2.x.tgz | tar xvp -** (overwrite existing files, if any)
This creates (or updates) the Directory tree:
\$SOFTDIR
 bin
 ARCHI
 bin
 TCL
 doc
 EXAMPLES
8. For the **first installation only**:
 execute: **cd \$SOFTDIR/bin**
 execute: **platform.tcl**
 This copies configuration files in your home directory depending on your OS

- 9) **optional**: in **\$SOFTDIR/bin** make a symbolic link:
 CPUSimulator to **../ARCHI/bin/CPUSimulator.tcl**

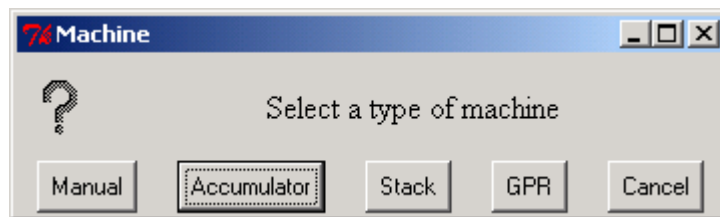
- 10) That's All Folks!

4 The CPU Simulator (v2.0)

4.1 Launching the application

Execute `CPUSimulator`. This is a symbolic link, present in the `$SOFTDIR/bin` directory and pointing to the actual application:
`$SOFTDIR/ARCHI/bin/CPUSimulator.tcl`.

The dialog box below appears.



Four machines are proposed:

- **Manual:** not a programmable machine, just an ALU and a Memory. Data are fetched from and written to the memory by “drag-and-drop” moves. The operation performed by the ALU is selected by the user.
- **Accumulator:** an accumulator-based machine. Associated assembly language source files are suffixed by `.amp` (Accumulator Machine Program).
- **Stack:** a stack-based machine. Associated assembly language source files are suffixed by `.smp` (Stack Machine Program).
- **GPR:** a General-Purpose-Register machine, more precisely a register-register architecture: only load and store instructions access memory locations; arithmetic and logical instructions are performed only on registers. Associated assembly language source files are suffixed by `.gmp` (GPR Machine Program).

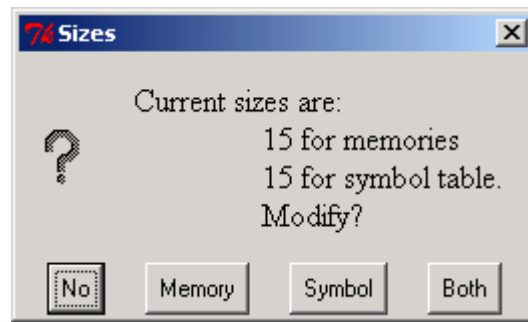
Remark: GPR machine are not yet implemented.

These machines are very small machines: They have tiny memories and a limited set of instructions. For simplicity, data memory and instruction memory are separated.

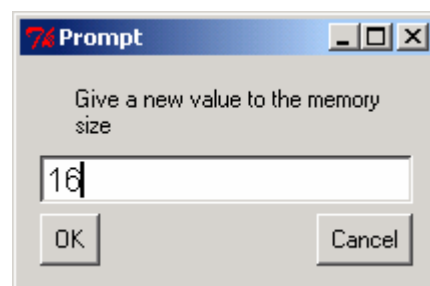
4.2 Selecting

Click on the corresponding button to select the desire machine type.

The default sizes of the (small) memories of the machines are stored in a configuration file (`.CPUSimulator`, in your home directory). This size is initially set to 15. A dialog box proposes to change these sizes. You can answer “No”, since the actual sizes could be enlarged later.



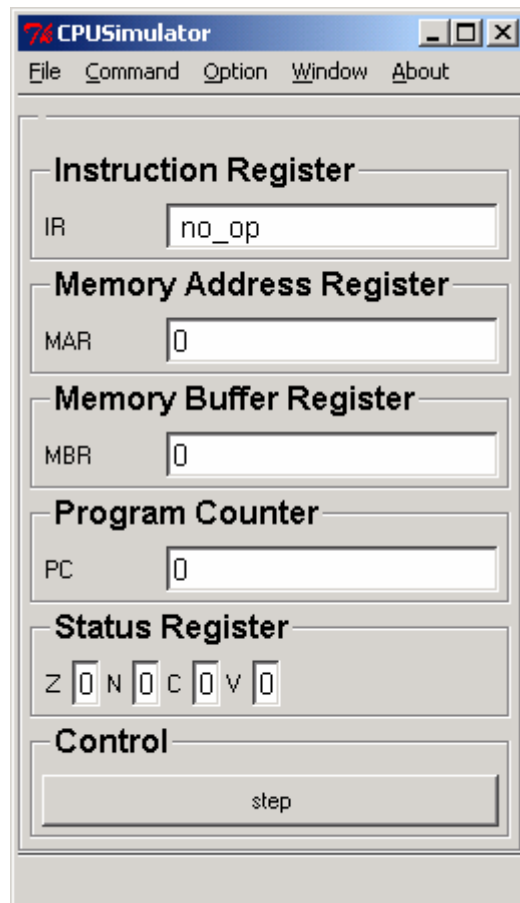
If a modification is selected, another dialog box pops up and asks for a new value. “Memory” applies only to Data and Instruction Memories; “Symbol” concerns the size of the Table of Symbols; “Both” applies to both. Prefer this latter option to the previous two choices.



4.3 The Control Window

The control window is the main panel. It offers

- a menu bar,
- various registers,
- and a control button



4.3.1 Menus

The menu bar presents a set of button-like menu entries to users. When you drag your mouse over the menu bar, the different menus are displayed. A click on a button posts (i.e., displays) the associated menu.

File

Load	open a browser for loading
Re-load	load the previously loaded file
Save	open a browser for saving
Exit	exit the application

Command

Reset	reset the simulation (memory and register cleanup)
<hr/>	
Extend Memory	increment the size of the data and instruction memory by 1

Option

Preferences	configure your personal environment with a dialog box
<hr/>	
Fonts	show and select fonts
<hr/>	
Help font: 8pt	set the size of font in balloon to 8
Help font: 10pt	set the size of font in balloon to 10
Help font: 12pt	set the size of font in balloon to 12

<p>-----</p> <p>Step</p> <p>Fetch/Execute</p> <p>Detail</p>	<p>running mode: step by step (a full instruction at a time)</p> <p>running mode: separate instruction fetch and execution</p> <p>running mode: details (i-fetch, decode, d-fetch, operate, store)</p>
--	--

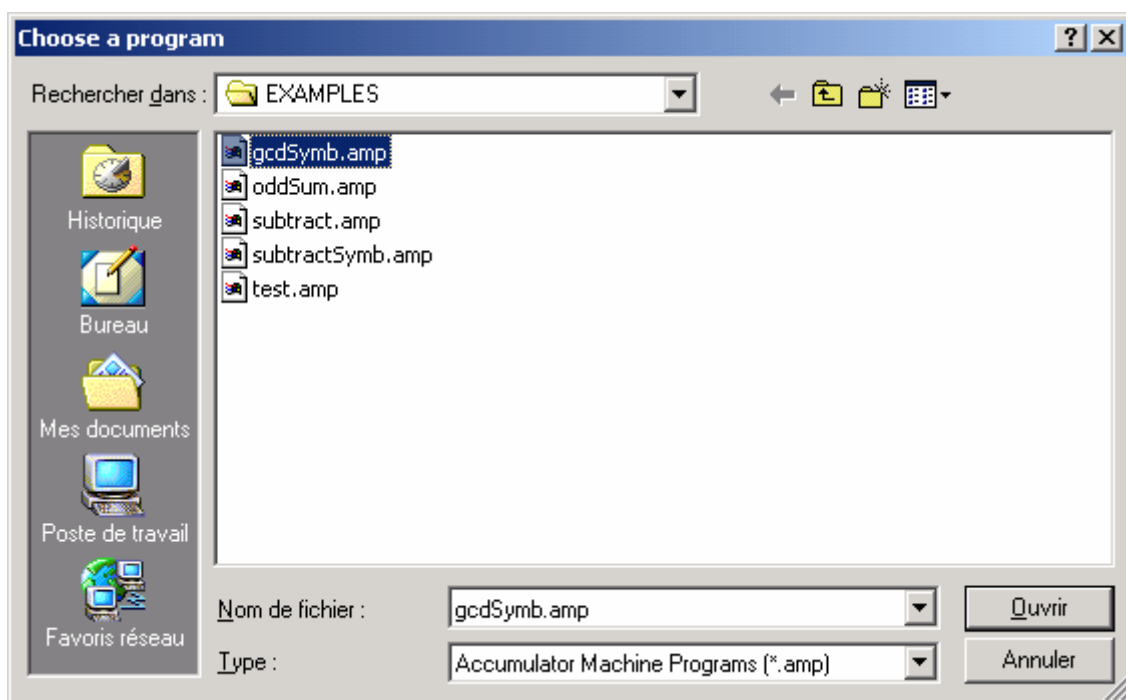
Window check buttons selecting windows to show/hide

Show ALU
Show Symbol Table
Show Pseudo Console
Instruction Set
Performances
Raise Memory

About miscellaneous information

Author
Address
Version

File loading dialog box



File Saving Dialog Box

This dialog box is similar to the file loading dialog box. Contrary to the file loading box, creating a new file is possible.

4.3.2 Registers

Registers displayed in the control window are non user-programmable registers. They are useful for tracing program executions and understanding data paths.

- The **Instruction Register** (IR) contains, in a symbolic form, the instruction to be decoded and executed.
- The **Memory Address Register** (MAR) points either an Instruction Memory Cell (for i-fetch), or a Data Memory Cell (for d-fetch or store).
- The **Memory Buffer Register** (MBR) contains data from the memory (read-memory cycle) or to the memory (write-memory cycle).
- The **Program Counter** (PC) points to the current instruction. All programs start from address 0.
- The **Status Register** consists of 4 individual flags:
 - Zero (**Z**) set to 1 when the result of a logical or an arithmetic operation is 0
 - Negative (**N**) set to 1 when the most significant bit is 1
 - Carry (**C**) set to 1 by addition or subtraction overflows, and by logical shift operations.
 - oVerflow (**V**) set to 1 in case of an algebraic overflow.

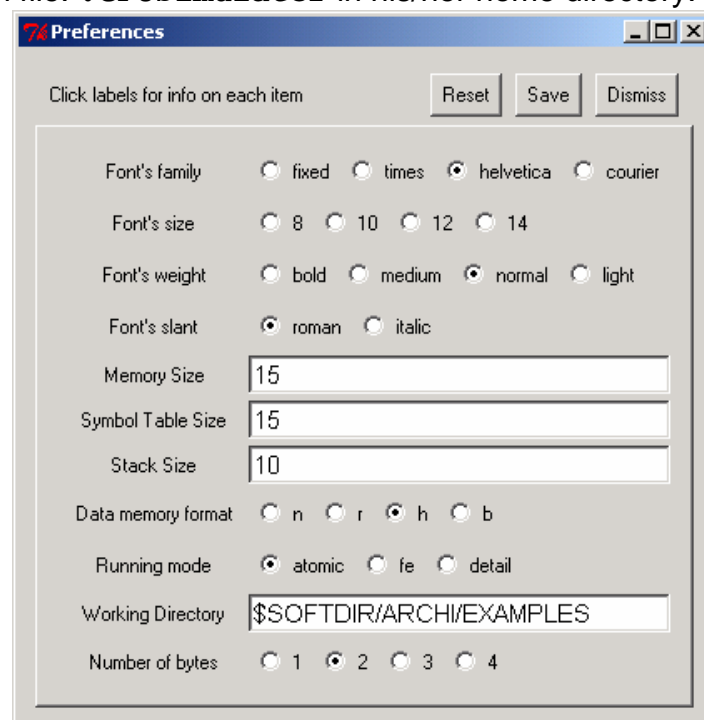
4.3.3 Control button

This button indicates what is the next thing to be done by the simulator. Its label depends on the current running mode.

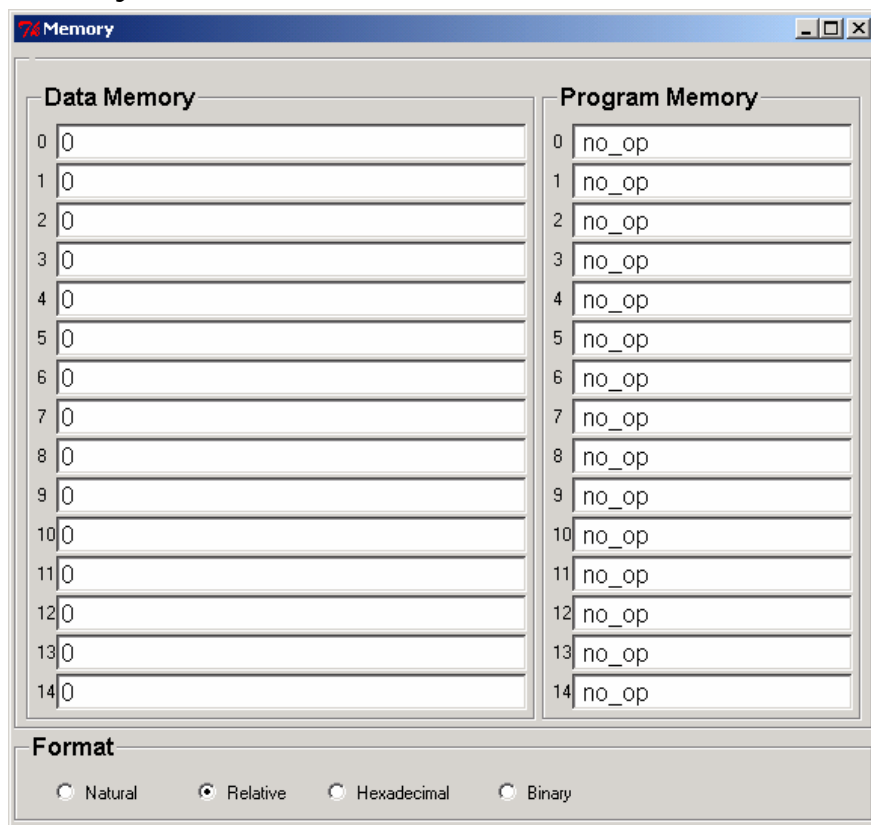
- In the step-by-step (or **Atomic**) mode, the label is always “Step”.
- In the **Fetch/Execute** (or fe) mode, the label alternates “i-fetch” and “Execute”.
- In the **Detail** mode, the label may be “i-fetch”, “decode”, “d-fetch”, “operate”, “d-write”. The exact succession depends on the instruction.

4.3.4 Preferences

This dialog box allows the user to customize its environment. His/her choices are saved in a hidden file: **.CPUSimulator** in his/her home directory.



4.4 The Memory Window

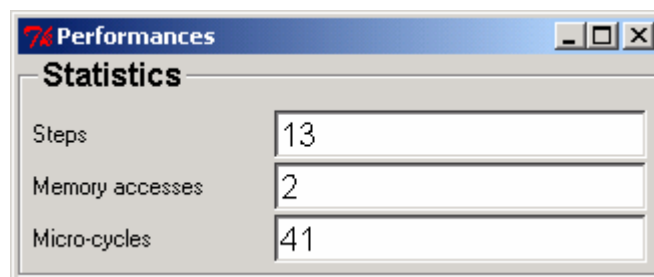


The left side of the window represents the Data Memory, the right side the Program Memory. The addresses are written on the left of each memory entry. The bottom frame, entitled "Format", allows the user to select a display format.

4.5 The Performance Windows

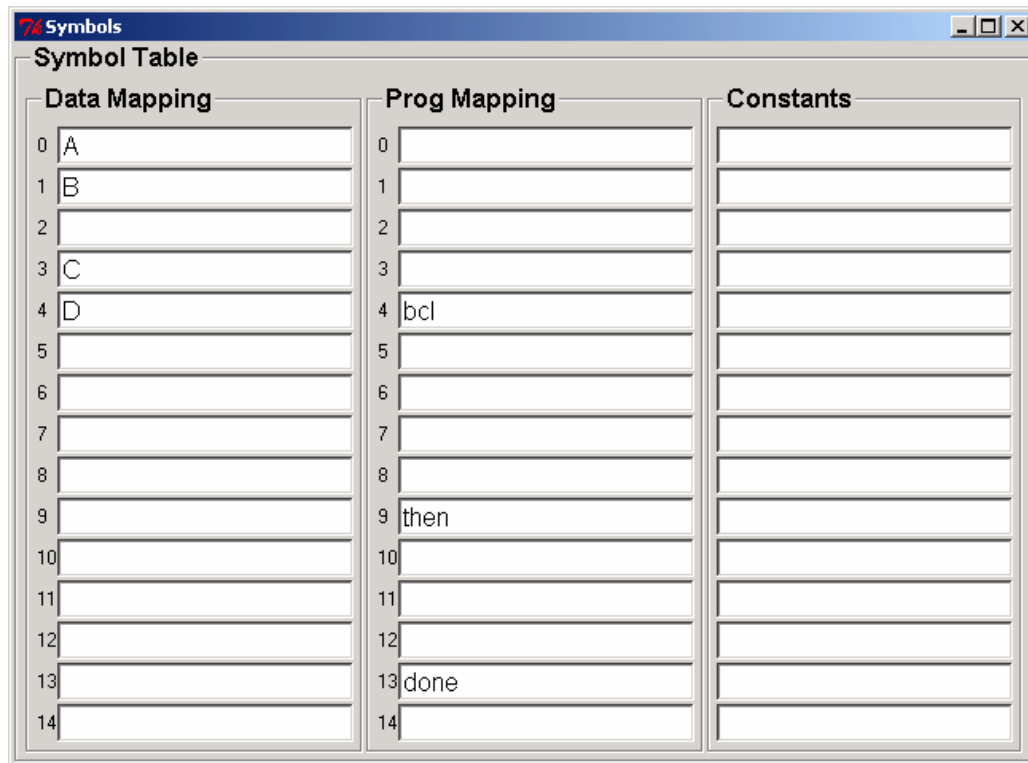
While a program is executed, statistics are computed. The number of steps (i.e., the number of executed instructions), the number of Memory Accesses, and the number of microcycles are automatically updated. All the counters are reset to 0 by the Reset command, or by a new program loading.

The number of microcycles by instruction is computed by taking one unit for each phase: i-fetch, decode, d-fetch, operate, d-write. This is a coarse approximation. With modern fast CPUs, external accesses should be given a relative cost much higher than 1.



4.6 Symbol Table Window

When your program uses symbolic names, you have to fill up the symbol table (You do what is usually done by an assembler!).



The screenshot shows a window titled "74 Symbols" with a "Symbol Table" section. It contains three columns: "Data Mapping", "Prog Mapping", and "Constants". Each column has 15 rows, indexed from 0 to 14. The "Data Mapping" column contains the labels A, B, C, and D at addresses 0, 1, 3, and 4 respectively. The "Prog Mapping" column contains the labels bcl, then, and done at addresses 4, 9, and 13 respectively. The "Constants" column is empty.

Symbol Table		
Data Mapping	Prog Mapping	Constants
0 A	0	
1 B	1	
2	2	
3 C	3	
4 D	4 bcl	
5	5	
6	6	
7	7	
8	8	
9 then	9 then	
10	10	
11	11	
12	12	
13 done	13 done	
14	14	

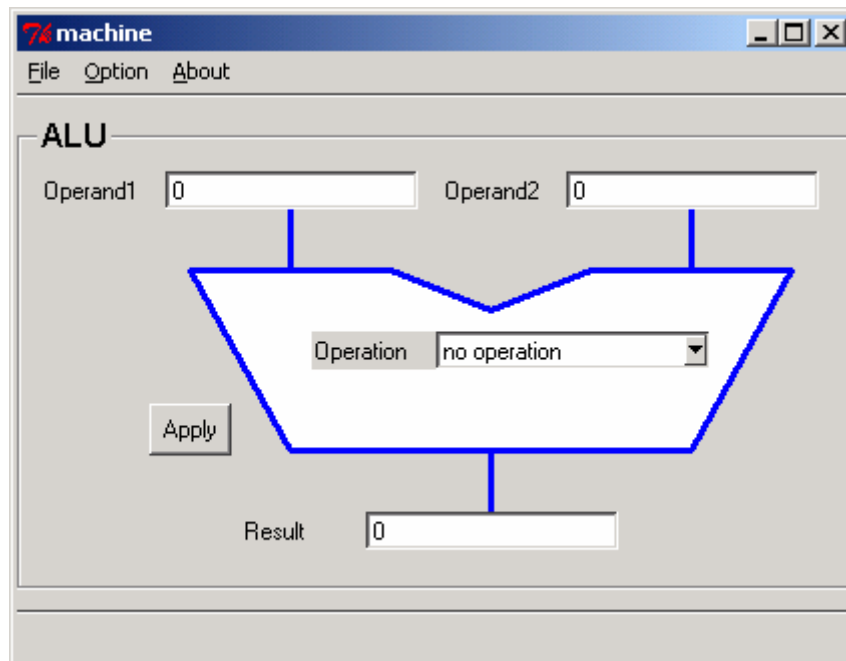
The above table corresponds to a program that makes reservation for variables A, B, gcd, C, and D at the respective addresses 0, 1, 2, 3, 4 in the Data memory. The program uses symbolic labels "bcl:", "then:", "done:" at respective locations 2, 4, and 13 in the Program Memory.

Aliases are possible: each entry may contain a space-separated list of identifiers.

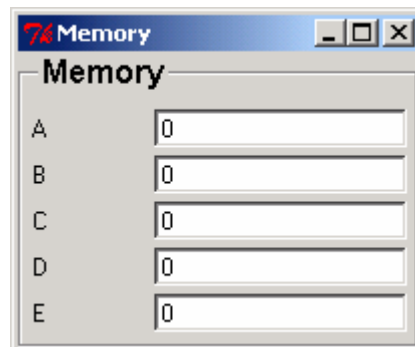
Syntactic Remark: symbolic labels are followed by a colon in their definition; the colon is omitted in the table and in the branch instructions.

5 Manual Execution

As explained above, this is not a programmable machine. The user has to drag and drop the data, select the operation of the ALU, apply the operation and collect the result.



The screenshot shows a window titled "74 machine" with a menu bar containing "File", "Option", and "About". The main area is labeled "ALU". It features two input fields: "Operand1" and "Operand2", both containing the value "0". Below these is a large, white, trapezoidal shape with a blue outline, representing the ALU. Inside this shape is a dropdown menu labeled "Operation" with the text "no operation". To the left of the ALU shape is a button labeled "Apply". Below the ALU shape is a "Result" field containing the value "0".



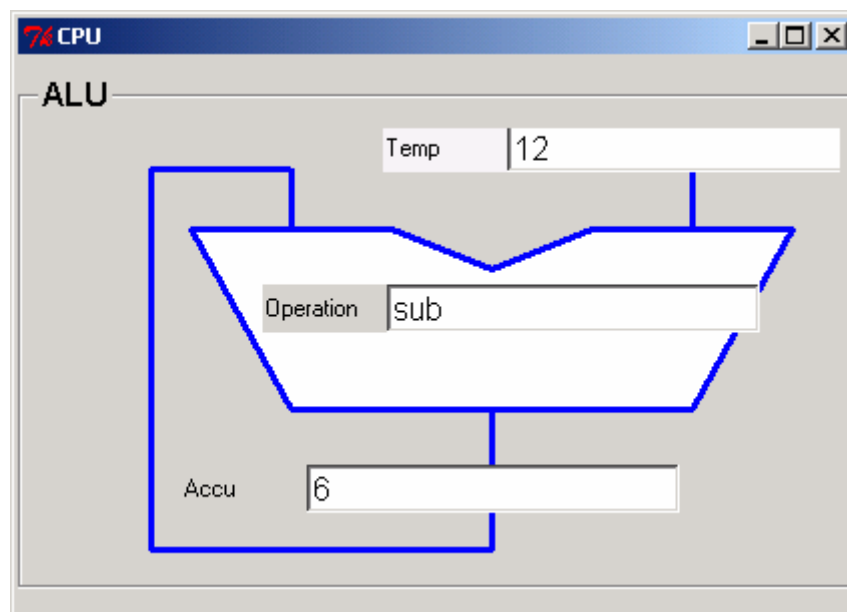
The screenshot shows a window titled "74 Memory" with a menu bar containing "File", "Option", and "About". The main area is labeled "Memory". It features a list of five memory locations: A, B, C, D, and E. Each location has a corresponding input field, all of which contain the value "0".

6 Accumulator-based Machine

6.1 Arithmetic and Logical Unit

This machine has an accumulator, used as both the first operand of the operation performed by the ALU, and the result of the operation. The second operand, if any, comes from the Memory or the Instruction Register (for immediate operations). The entry named “Temp” shows the value of this operand.

The operation entry is set during the decode phase according to the contents of the Instruction Register.



6.2 Instruction Set

74 Instructions			
Instruction Set			
Data Transfer	Arithmetic Operations	Logical Operations	Control Transfer
load	add	and	beqz
store	sub	or	bnez
loadi	neg	not	bltz
	addi	asr	bgez
	subi	lsl	bc
		lsr	bnc
		andi	bt
		ori	

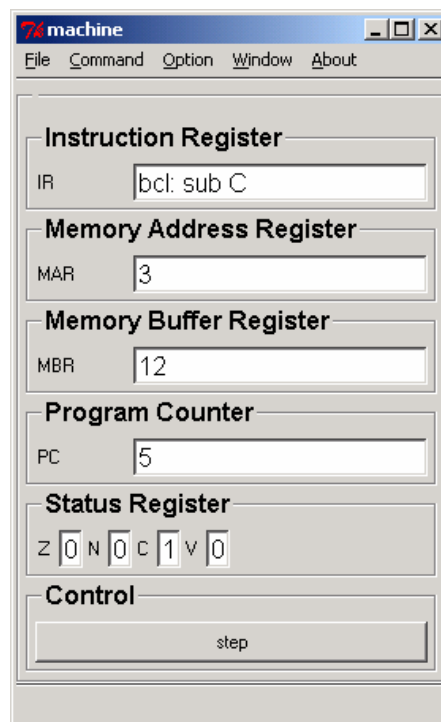
The (simplified) instruction set is available in the Instruction Set Window. The semantics of each instruction is popped-up when the mouse stays over the label for a while.

6.3 Example of the gcd computation

This program computes the Greatest Common Divider of two positive integers A and B. This algorithm is the original one, using only subtraction, not the remainder of the integer division.

Gcd is the result, A and B are the input data preserved during the computation. C and D are auxiliary variables.

	load	A	
	store	C	; C is the dynamic value of A
	load	B	
	store	D	; D is the dynamic value of B
bcl:	sub	C	; D - C
	beqz	done	; If Z then go to done
	bltz	then	; If D < C then go to then
	store	D	; D > C => B <- B - A
	bt	bcl	; iterate
then:	neg		; C > D; take the opposite of D - C
	store	C	; C <- C - A
	load	D	; Before looping, put D in the accumulator
	bt	bcl	; iterate
done:	load	C	; Here C=D=gcd
	store	gcd	



The figure above shows the control window *before* executing the instruction at location 5 (i.e., the contents of PC). In the Program Memory Window, the last

executed instruction (`bcl: sub C`) is highlighted (painted red) and still present in the Instruction Register. The Memory Address Register points to (data) memory cell 3 (i.e., the location of C, the last accessed memory cell). The Memory Buffer Register contains the value of C (i.e., 12).

The highlighted Data Memory cell is the more recently written memory cell (Here, the cell that stores the value of D).

Remind that correspondences between memory locations and symbolic names are given by the Table of Symbols.

The screenshot shows a window titled "74 Memory" with two main sections: "Data Memory" and "Program Memory".

Data Memory: A table with 15 rows (0 to 14) and one column. The values are: 0: 12, 1: 18, 2: 0, 3: 12, 4: 18 (highlighted in red), 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0, 12: 0, 13: 0, 14: 0.

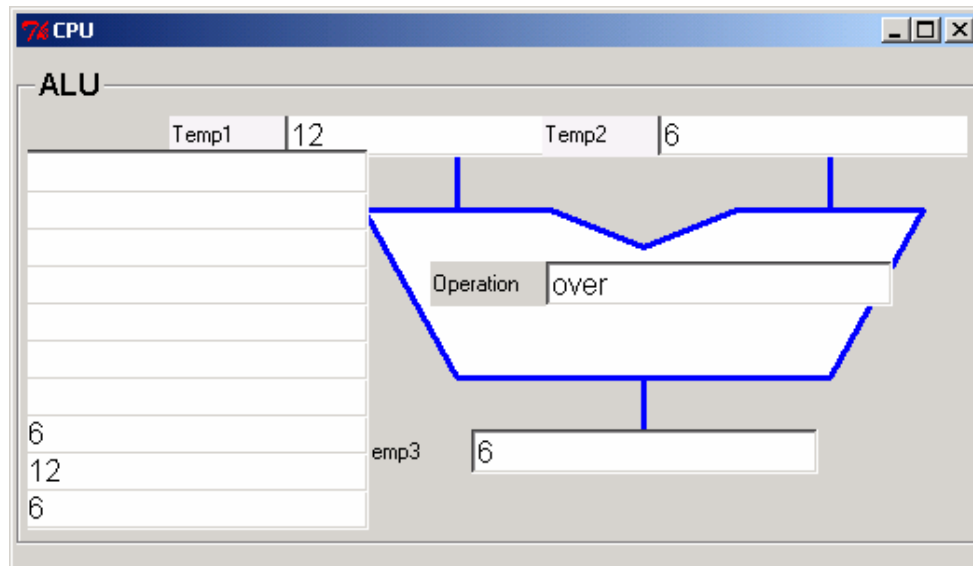
Program Memory: A table with 15 rows (0 to 14) and one column. The instructions are: 0: load A, 1: store C, 2: load B, 3: store D, 4: bcl: sub C (highlighted in red), 5: beqz done, 6: bltz then, 7: store D, 8: bt bcl, 9: then: neg, 10: store C, 11: load D, 12: bt bcl, 13: done: load C, 14: store gcd.

Format: A section at the bottom with four radio buttons: Natural, Relative (selected), Hexadecimal, and Binary.

7 Stack-based Machine

7.1 Arithmetic and Logical Unit

The UAL has two operands and a result. Sources and destination are the stack. The stack itself is represented on the left of the picture. The stack grows upwards.



7.2 Instruction Set

The instruction set is available in the Instruction set Window. The semantics of each instruction is popped-up when the mouse stays over the label for a while.

The screenshot shows the '74 Instructions' window with the 'Instruction Set' tab selected. The window displays a table of instructions categorized into five groups: Data Transfer, Stack Operations, Arithmetic Operations, Logical Operations, and Control Transfer.

Data Transfer	Stack Operations	Arithmetic Operations	Logical Operations	Control Transfer
push	dup	add	and	beqz
pop	swap	sub	or	bnez
pushi	drop	neg	not	bltz
	over	cmp	asr	bgez
	roll		lsl	bc
			lsr	bnc
				bt

7.2.1 Example of the gcd computation

	push	A	;	A	-	
	push	B	;	A	B	-
bcl :	cmp		;	A	B	- Set N and Z
	beqz	done	;	Branch to done if A == B		
	bltz	else	;	Branch to else if A < B		
	swap		;	B	A	-
	over		;	B	A	B -
	sub		;	B	A-B	-
	bt	bcl	;	Branch to bcl with new A and old B		
else:	over		;	A	B	A -
	sub		;	A	B-A	-
	bt	bcl	;	Branch to bcl with new B and old A		
done:	drop		;	A	-	Discard B
	pop	gcd	;	-		

74Memory
⌵ ⌶ ⌵

Data Memory

0	12
1	18
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0

Program Memory

0	push A
1	push B
2	bcl: cmp
3	beqz done
4	bltz else
5	swap
6	over
7	sub
8	bt bcl
9	else: over
10	sub
11	bt bcl
12	done: drop
13	pop gcd
14	no_op

Format

☐ Natural
☒ Relative
☐ Hexadecimal
☐ Binary

74 machine

File Command Option Window About

Instruction Register

IR over

Memory Address Register

MAR 6

Memory Buffer Register

MBR over

Program Counter

PC 7

Status Register

Z 0 N 0 C 1 V 0

Control

step