mu0 user manual

Title	mu0 (HDL models and programming tools for the educational MU0 processor)
Author	Nikolaos Kavvadias (C) 2010, 2011, 2012, 2013, 2014
Contact	nikos@nkavvadias.com
Website	http://www.nkavvadias.com
Release Date	19 November 2014
Version	0.0.3
Rev. history	
v0.0.3	2014-11-19 Extended the compiler/assembler to automatically produce ArchC hexadecimal files; add non-interactive mode.
v0.0.2	2014-11-18 Added more test programs/listings; minor documentation update.
v0.0.1	2014-11-17 Added preliminary version of the ArchC model for the processor. This models a byte-addressable version of MU0.
v0.0.0	2014-11-14 Initial release.

1. Introduction

The `mu0` is an educational computer taught at the University of Manchester (CS1011_MU0 and [Furber]). It is based on the SSEM computer which was one of the first computers every built - at the University (and is considered, along with the Harvard Mark 1 to be the first real computer).

The MU0 is used to illustrate basic programming concepts, and encourages thorough design due to the fact it only has 8 useful instructions (including a halting/stop instruction), albeit there is available opcode space for an additional eight instructions.

The processor can directly address 4096 words, each 16 bits long. Each word is capable of storing one fixed length command, which consists of 4 bits of opcode and 12 bits of operand, in all cases except the STOP command which takes no operand.

The only internal register is known as the accumulator (`ACC`) and this is where all processing must take place. It is 16 bits long, and is where both inputs to calculations and results must be stored. In total, an MU0 processor has three registers:

- `ACC`: the accumulator

- PC: the program counter
- IR: the instruction register.

The following table illustrates the instruction set of the MU0.

Opcode	Instruction	Effect	Syntax variant (tools)
0000	LDA S	ACC = mem[S]	ACC<= [S]
0001	STO S	mem[S] = ACC	ACC>= [S]
0010	ADD S	ACC += mem[S]	ACC+ [S]
0011	SUB S	ACC -= mem[S]	ACC- [S]
0100	JMP S	pc = S	PC<= S
0101	JGE S	if ACC>=0 pc = S	IF+VE PC<= S
0110	JNE S	if ACC!=0 pc = S	IF!=0 PC<= S
0111	STP	stop	STP

This distribution provides the following:

- Behavioral VHDL and Verilog HDL models for the mu0.
- ArchC functional simulation model for the mu0.
- Compiler (assembler) and simulator/debugger for the mu0 based on the original work of user benjy: http://everything2.com/title/MU0
- Scripts for running VHDL simulations with GHDL or Modelsim.
- Scripts for running Verilog HDL simulations with Icarus Verilog or Modelsim.
- Various test files (*.mu0, *.lst, *.hex).

Future releases will contain adapted synthesizable models, synthesis scripts for Xilinx ISE/Vivado and YOSYS and more.

The original documentation as written by benjy can be found in the /doc subdirectory in plain text, HTML and PDF formats.

2. File listing

The mu0 distribution includes the following files:

/mu0	Top-level directory
AUTHORS	List of authors.
LICENSE	The license agreement for using mu0.
README.rst	This file.
README.html	HTML version of README.
README.pdf	PDF version of README.
VERSION	Current version of the mu0 project.
rst2docs.sh	Bash script for generating the HTML and PDF versions.
/bench/verilog	Verilog HDL testbench directory

mu0_tb.v	Testbench for exercising the Verilog HDL model.
/bench/vhdl	VHDL testbench directory
mu0_tb.vhd	Testbench for exercising the VHDL model.
/doc	Documentation directory
mu0-compiler-sim.rst	Detailed documentation on the MU0 assembler and simulator (authored by user `benjy`).
mu0-compiler-sim.html	HTML version of the above.
mu0-compiler-sim.pdf	PDF version of the above.
rst2docs.sh	Bash script for generating the HTML and PDF versions.
/rtl/verilog	RTL Verilog source code directory for `mu0`
mu0_behav.v	Behavioral Verilog HDL model.
/rtl/vhdl	RTL VHDL source code directory for `mu0`
mu0_behav.vhd	Behavioral VHDL model.
/sim/archc	ArchC model files main directory
/sim/archc/src	Source directory for the model files
mu0.ac	Register and memory model for MU0.
mu0_isa.ac	Instruction encodings and assembly formats.
mu0_isa.cpp	Instruction behaviors.
/sim/archc/test	Tests subdirectory
gen-tests.sh	Bash shell script for generating ArchC hexadecimal application files for the simulator.
*.hex	ArchC hexadecimal application files for testing.
/sim/rtl_sim	RTL simulation files directory
/sim/rtl_sim/bin	RTL simulation scripts directory
mu0_behav.mk	Unix/Cygwin makefile for running a GHDL simulation.
mu0_behav_verilog.do	Modelsim `do` macro for running a Verilog simulation.
mu0_behav_vhdl.do	Modelsim `do` macro for running a VHDL simulation.
/sim/rtl_sim/out	Dumps and other useful output from RTL simulation
mu0_behavioral.vcd	VCD (Value Change Dump) file from the last simulation run.
/sim/rtl_sim/run	Files for running RTL simulations
ghdl.sh	Bash shell script for running a GHDL simulation.
iverilog.sh	Bash shell script for running an Icarus Verilog simulation.
load-program.sh	Bash shell script for loading a new program to the HDL processor model (either Verilog HDL or VHDL).
mti-verilog.sh	Bash shell script for running a Modelsim simulation of the Verilog HDL model.
mti-vhdl.sh	Bash shell script for running a Modelsim simulation of the VHDL model.
multiply.lst	Hexadecimal listing generated from `multiply.mu0` using the `mu0` compiler.

multiply.mu0	Multiplication test program.
odd_even.lst	Hexadecimal listing generated from `odd_even.mu0` using the `mu0` compiler.
odd_even.mu0	Test program for finding even numbers in a list.
prog.lst	The listing file currently visible to the processor models. Its contents are preloaded to memory before simulation starts.
test*.lst	Sample test listings.
test*.mu0	Sample test programs.
/sim/rtl_sim/run	Verilog HDL sources for running RTL simulations
/sim/rtl_sim/vhdl	VHDL source files used for running RTL simulations
std_logic_textio.vhd	Modified version of a testbench-related package.
/sw	Software utilities
Makefile	GNU Makefile for building the compiler and debugger.
compile_mu0.c	The MU0 compiler (assembler) developed by `benjy`.
execute_mu0.c	The MU0 debugger developed by `benjy`.

3. Usage

Build the MU0 compiler and debugger

Here we assume that the `/mu0` distribution directory is a subdirectory of the working directory.

```
$ cd mu0
$ cd sw
$ make clean ; make ; make tidy
```

Now the compiler (`compile_mu0.exe`) and debugger/simulator (`execute_mu0.exe`) have been generated.

Compile an MU0 application

```
$ cd ../sim/rtl_sim/run
$ ../../../sw/compile_mu0.exe
```

A command-prompt appears which looks like this:

```
COMPILE_MU0 - companion program to EXECUTE_MU0
(C) 1994 Benjy

Please enter source filename >
```

The user can enter the file name of an existing `*.mu0` assembly program such as `multiply.mu0`:

```
Please enter source filename > multiply.mu0
```

In the subsequent prompt, the user should enter the preferred filename for the listing (hexadecimal file) to be produced:

```
Please enter destination filename > multiply.lst
```

By hitting enter again, two-pass assembly will take place and the produced listing will be available for loading to the processor model(s).

Load the program

```
$ ./load-program multiply.lst
```

The above command copies the produced listing, `multiply.lst` to `prog.lst` which is the name of the listing that both the Verilog HDL and VHDL models expect to read and load to the processor's memory.

Run Verilog HDL simulation using Icarus Verilog

To run a Verilog HDL simulation using Icarus Verilog, the following script can be used. As with all simulation scripts, the user will have to edit it in order to provide the correct path to the tools (Icarus Verilog, GHDL, Modelsim) for his/her setup.

```
$ ./iverilog.sh
```

Run Verilog HDL simulation using Modelsim

```
$ ./mti-verilog.sh
```

Run VHDL simulation using GHDL

```
$ ./ghdl.sh
```

Run VHDL simulation using Modelsim

```
$ ./mti-vhdl.sh
```

Visualize simulation waveforms

For both VHDL and Verilog HDL simulations, waveform data are produced in the VCD format. VCD waveforms can be easily viewed using GTKwave.

```
$ gtkwave ../out/mu0_behavioral.vcd
```

4. ArchC model

This is the ArchC (http://www.archc.org) functional simulator model for the MU0 processor. For the time being, the architecture is modelled as a byte-addressable, as the careful reader can notice by examining the ArchC hexadecimal applications files that can be found in `/mu0/sim/archc/tests`. If the `JGE_IS_JGT` preprocessor directive is set, then the behavior of the jump if positive (`jge`) instruction is altered to

convey the meaning of jump if (strictly) larger than zero. There is no concensus about the behavior of this specific instruction, according to various sources on the MU0 processor.

Building the model

To generate the interpreted simulator, the `acsim` executable is ran:

```
$ acsim mu0.ac                          # (create the simulator)
$ make -f Makefile.archc                # (compile)
$ ./mu0.x --load=<file-path> [args] # (run an application)
```

There are two formats recognized for application <file-path>:

- ELF binary matching ArchC specifications
- hexadecimal text file for ArchC, which has currently been tested.

In order to generate the binary utilities port (`binutils` port), the `acbingen.sh` driver script must be used. This should be called as follows:

```
$ acbingen.sh -amu0 -i`pwd`/../mu0-tools/ mu0.ac
```

for generating the `binutils` port executables. This includes the following tools:

- `addr2line`
- `ar`
- `as`
- `c++filt`
- `ld`
- `nm`
- `objcopy`
- `objdump`
- `ranlib`
- `readelf`
- `size`
- `strings`
- `strip`

This feature has not yet been tested for the `mu0` model.

Alternative assembly syntax

The ArchC-based tools support a number of alternative assembly instruction syntaxes for `mu0`. The following table summarizes the differences between the syntax variations.

Instruction	Alternative syntax	
`lda`	`lda imm`	
`sto`	`sto imm`	
`add`	`add imm`	
`sub`	`sub imm`	
`jmp`	`jmp imm`	
`jge`	`jge imm`	
`jne`	`jne imm`	
`stp`	`stp`	`halt`

5. Prerequisites

- Standard UNIX-based tools (tested with gcc-4.8.1 on MinGW/x86) [optional if you use Modelsim].
  - make
  - bash (shell)

  For this reason, MinGW (http://www.mingw.org) or Cygwin (http://sources.redhat.com/cygwin) are suggested, since POSIX emulation environments of sufficient completeness.
- Icarus Verilog simulator (http://iverilog.icarus.com/). The Windows version can be downloaded from: http://bleyer.org/icarus/
- GHDL simulator (http://ghdl.free.fr) [optional if you use Modelsim]. Provides the `ghdl` executable (has several Windows versions, with 0.29.1 and 0.31 being the latest). It also installs GTKwave on Windows. Note that the latest version (0.31) from http://sourceforge.net/project/ghdl-updates/ does not include GTKwave.
- Alternatively, a commercial simulator like Mentor Modelsim (http://www.model.com) can be used.
- ArchC (http://www.archc.org) installation (tested on Cygwin/Win7-64bit and Linux) [required only for using the ArchC model]

6. Contact

You may contact me at:

Nikolaos Kavvadias <nikos@nkavvadias.com>
Independent Consultant
http://www.nkavvadias.com
Kornarou 12 Rd,
35100 Lamia, Fthiotis
Greece

References

[Furber] Stephen Furber, ARM System-on-chip Architecture, 2nd edition, Pearson Education Limited, 2000.