# Department of Electrical and Computer Engineering

### The University of Texas at Austin

EE 460N, Fall 2014
Lab Assignment 3
Due: Tuesday, September 30th, 11:59 pm
Yale Patt, Instructor
Stephen Pruett, Emily Bragg, Siavash Zangeneh, TAs

# Introduction

For this assignment, you will write a cycle-level simulator for the LC-3b. The simulator will take two input files:

1. A file entitled **`ucode3`** which holds the control store.
2. A file entitled **`isaprogram`** which is an assembled LC-3b program.

The simulator will execute the input LC-3b program, using the microcode to direct the simulation of the microsequencer, datapath, and memory components of the LC-3b.

Note: The file **`isaprogram`** is the output file from Lab Assignment 1. This file should consist of 4 hex characters per line. Each line of 4 hex characters should be prefixed with '0x'. For example, the instruction `NOT R1, R6` would have been assembled to `1001001110111111`. This instruction would be represented in the **`isaprogram`** file as `0x93BF`. The file **`ucode3`** is an ASCII file that consists of 64 rows and 35 columns of zeros and ones.

The simulator is partitioned into two main sections: the shell and the simulation routines. We are providing you with the shell. Your job is to write the simulation routines.

# The Shell

The purpose of the shell is to provide the user with commands to control the execution of the simulator. In order to extract information from the simulator, a file named **`dumpsim`** will be created to hold information requested from the simulator. The shell supports the following commands:

1. **`go`** – simulate the program until a HALT instruction is executed.
2. **`run <n>`**– simulate the execution of the machine for n cycles
3. **`mdump <low> <high>`** – dump the contents of memory, from location low to

location high to the screen and the dump file. For hex addresses, put "**0x**" in front of the address, eg. **mdump 0x3000 0x3001**

4. **rdump** – dump the current cycle count, the contents of R0-R7, IR, PC, MAR, MDR, and other status information to the screen and the dump file.
5. **?** – print out a list of all shell commands.
6. **quit** – quit the shell

# The Simulation Routines

The simulation routines carry out the cycle-by-cycle simulation of the input LC-3b program. The simulation of any cycle is based on the contents of the current latches in the system. The simulation consists of two concurrently executing phases:

The microsequencer phase uses 9 bits from the microinstruction register and appropriate literals from the datapath to determine the next microinstruction. Function `eval_micro_sequencer` evaluates this phase.

The datapath phase uses 26 bits of the microinstruction to manipulate the data in the datapath. Each microinstruction must be literally interpreted. For example, if the `GateMDR` bit is asserted, then data must go from the MDR onto the bus. You must also establish an order for events to occur during a machine cycle. For example, data should be gated onto the bus first, and loaded into a register at the end of the cycle. Simulate these events by writing the code for functions `eval_bus_drivers`, `drive_bus` and `latch_datapath_values`.

We will assume a memory operation takes five cycles to complete. That is, the ready bit is asserted at the end of the fourth cycle. Function `cycle_memory` emulates memory.

# What To Do

The shell has been written for you. From your ECE LRC account, copy the following file to your work directory:

**lc3bsim3.c**

At present, the shell reads in the microcode and input program and initializes the machine. It is your responsibility to write the correct microcode file and to complete the simulation routines that simulate the activity of the LC-3b microarchitecture. In particular, you will be writing the five functions described above (`eval_micro_sequencer`, `cycle_memory`, `eval_bus_drivers`, `drive_bus`, and `latch_datapath_values`).
Add your code to the end of the shell code. Do not modify the shell code.

The accuracy of your simulator is your main priority. Specifically, make sure the correct microarchitectural structures sample the correct signals.

It is your responsibility to verify that your simulator is working correctly. You should write programs using all of the LC-3b instructions and execute them one cycle at a time (run 1). You can use the **`rdump`** and **`mdump`** commands to verify that the state of the machine is updated correctly after the execution of each cycle.

Because we will be evaluating your code on linux, you must be sure your code compiles on an ECE linux machine using gcc with the **`-ansi`** flag. This means that you need to write your code in C such that it conforms to the ANSI C standard. You should also make sure that your code runs correctly on one of the ECE linux machines.

# What To Turn In

Please submit your lab assignment electronically. You will submit the following files:

1. **`lc3bsim3.c`** – adequately documented source code of your simulator
2. **`ucode3`** – your microcode file

# Important

1. Please make sure that you have made the following correction to the ISA handout.
2. In Appendix A, please correct the operation of the JSR/JSRR instruction to read:
3. 
```
TEMP = PC†
if (bit(11)==0)
    PC = BaseR;
else
    PC = PC† + LSHF(SEXT(PCoffset11), 1);
R7 = TEMP;

* PC†: incremented PC
```
4. Please note that LEA **does NOT** set condition codes.
5. LC-3b registers are 16 bits wide. However, when you perform arithmetic or bitwise operations in C on `int` data types on the Linux x86 machines you are using 32 bits. Therefore, you must be careful about not keeping the higher 16 bits of the results in the architectural state. The shell code includes a macro called `Low16bits` that you can use to avoid this problem.
6. The control signals in your ucode3 file **must** be encoded according to Tables C.1 and C.2 in Appendix C. For each signal, the first (leftmost or the topmost) signal value must be encoded as 0, the next value as 1, the value after that as 2 (binary 10) and so

on. For example, Table C.1 lists the signal values for the two bit signal ALUK as follows: ADD, AND, XOR, PASSA. This means that ADD must be encoded as binary 00, AND as 01, XOR as 10, and PASSA as 11. Please use a 0 whenever a particular signal is a don't care.

# Lab Assignment 3 Clarifications

**NOTE: FAQ's for this semester will be posted here. Please check back regularly.**

1. What could be the cause of "Warning: Extra bit(s) in control store file ucode3."?
This means that the ucode3 file has more than 35 columns. One reason why this can happen is if you've transferred the ucode3 file from Windows to Unix. Run **`dos2unix`** program on UNIX machines to remove the extra control characters inserted by Windows. You can do this by typing:

```
dos2unix ucode3
```

on any LRC Linux machine. If this doesn't work, check to make sure that your ucode3 file has at most 35 columns.

2. What is `CYCLE_COUNT`? Is this the counter for memory access?
`CYCLE_COUNT` is a global variable used by the simulator to count the number of machine cycles elapsed since the program started execution. Do not change this variable. You will need to use another variable for simulating memory latency.

3. How do we handle Memory Mapped I/O for this lab?
You do not need to implement Memory Mapped I/O for this lab.

4. You do not have to implement the RTI instruction for this lab. You can assume that the input file to your simulator will not contain any RTI instructions.

5. For this assignment, you can assume that the programmer will always give aligned addresses, and your simulator does not need to worry about unaligned cases.

6. You may assume that the code running on your simulator has been assembled correctly and that the instructions your simulator sees comply with the ISA specifications, i.e. all instructions are valid and there are no unaligned accesses.

7. In your code that you write for lab 3, do not assign the current latches to the next latches. This is already done in the shell code.

8. I am getting values like `xFFFFFFFF` in my registers when they should be `xFFFF` instead, why is this?
The variables in your c program are 32 bit values, so the nubmer -1 is `xFFFFFFFF`. You need to make sure that when you store values in a variable you mask them properly. For

instance you would need to assign `var1` to `var2` using the statement `var1 = var2 & 0xFFFF`, or its equivalent `var1 = Low16bits(var2)`. This zeroes out the top 16 bits before writing `var2` into `var1`.

9. Do we need to implement the TRAP *routines*?
No. Whenever a TRAP instruction is processed, after the last state, PC will be set to 0, if you implement the TRAP instruction correctly. The simulator halts whenever PC becomes 0. **You are still implementing states 15, 28, and 30 associated with the TRAP instruction.**

10. Why am I not getting the result I expect from a C expression?
Please read and make sure you understand the precedence of C operators. Examples:

- `a & b == 0` means `a & (b == 0)`, therefore you might want to write `(a & b) == 0`
- `a >> 1 + b` means `a >> (1 + b)`, therefore you might want to write `(a >> 1) + b`
- `a = b + c? d : e` means `a = (b + c)? d : e`, therefore you might want to write `a = b + (c? d : e)`
- there are other examples from some lab 2 implementations... If you do not want to remember or to think too much about this, just use parentheses!

11. How do I convert my control store spreadsheet into the **`ucode3`** file to use in my simulator?

- Windows machine: Once you have filled in the control store spreadsheet, select only the cells that contain the 0s and 1s that form the microinstructions (rows 2-65, columns B-AJ). Choose “copy” from the Edit menu. Open up a new Word document and choose “paste special” from the Edit menu. Then, choose “unformatted text” and click on OK. Finally, select “replace” from the Edit menu. In the “Find what” box, type “^t” (without the quotes); leave the “Replace with” box empty. Click on the “Replace All” button. Save your file as a plain text file with filename “ucode3.” To use this file on a linux machine with the simulator, you will need to change the filename from “ucode3.txt” to “ucode3”. You will also need to run dos2unix on this file (see #7 on How to port code from Windows to sunfire).
- Linux machine: Use the OpenOffice spreadsheet program (oocalc) to open and fill in the spreadsheet. Select the cells that contain the 0s and 1s that form the microinstructions (rows 2-65, columns B-AJ). Choose “copy” from the Edit menu. Open up a text editor (eg. gedit, gvim, oowriter) and choose “paste” from the Edit menu (with OpenOffice oowriter, select “paste special” and choose “unformatted text”). Do a search and replace, searching for “\t” (without the quotes), and leaving the replace field empty. Save your file as “ucode3”.

12. If none of the tri-state buffers (GatePC, GateALU, etc.) are driving the bus in a given cycle, please put zero onto the bus in that cycle.