# Organization of Digital Computer Lab
# EECS 112L/CSE 132L

**Lab 2 - Single-cycle ARM Datapath and Control** - Complete

prepared by: Pooria M.Yaghini

University of California, Irvine

February, 6, 2017

Due on February $19^{th}$, 11pm. Note: this is a two-week assignment

## 1 Goal

In this lab, you will complete designing the single-cycle ARM processor.

### 1.1 Lab Description

In this lab you will complete and assess your previously designed single-cycle ARM single-cycle processor. In the first week, all data-processing instructions plus Branch and Link (BL), and Byte-size Store/Load instructions should be supported by the datapath and the control unit. Then in the second week, you need to replace the memories with synthesizable memory models, and synthesize you designed processor and report the Area and the Frequency of your processor. When is doubt refer to the reference textbook for further information. Then you will load a test program and confirm that the system works.

Here are the instructions need to be added in this lab.

Table 1: Instructions that need to be supported in Lab-3

| **Branches** | **Memory operation** | **Data Processing** | **Data Processing** |
|---|---|---|---|
| B | LDR | AND | BIC |
| BL | STR | EOR | MVN |
| | LDRB | SUB | CMN |
| | STRB | RSB | ORR |
| | LDRSB | ADD | MOV |
| | STRSB | ADC | LSR |
| | LDRSH | SBC | LSL |
| | STRSH | RSC | ROR |
| | | TST | ASR |
| | | TEQ | RRX |
| | | CMP | |

## 1.2 Branch & Link

"'Branch instructions use a single 24-bit signed immediate operand, imm24. As with data-processing and memory instructions, branch instructions begin with a 4-bit condition field and a 2-bit op, which is $10_2$. The funct field is only 2 bits. The upper bit of funct is always 1 for branches. The lower bit, L, indicates the type of branch operation: 1 for BL and 0 for B. The remaining 24-bit two's complement imm24 field is used to specify an instruction address relative to PC + 8."' Please read Section 6.4.3 of the supplemental reference book.

## 1.3 Immediate-shifted & Register-Shifted Register Instructions

ARM supports different operand addressing modes. One of the addressing modes is called Immediate-shifted or Register-shifted which can be applied either to Base or Register. Read it in Figure 6.17, and Table 6.12 of the supplemental reference book.
As you can see in the following example, R1 is scaled (shifted left by two) then added to the base address (R0). Thus, the memory address is R0 + (R1  4).

LDR R3, [R0, R1, LSL #2] ; R3 $\Leftarrow$ MEM[R0 + R1 $\times$ 4]

In addition to scaling the index register, ARM provides offset, pre-indexed, and post-indexed addressing to enable dense and efficient code for array accesses and function calls. This (pre-indexed, and post-indexed addressing), if implemented, will get bonus points. Please read page 314 of the supplemental reference book to get more information about this addressing mode.

### 1.3.1 Memory

In order to access data memory, design your processor to support 32-bit and 8-bit Store and Load instructions. Half-word Load/Store instruction implementation will get bonus points. The data & instruction memory should be a 512-word×32-bit array for now.

Code 1: Memory wrapper

```
module ram
    (
        input          clk    ,
        input          we     ,
        input  [ 3:0] be     , // Byte-enable
        input  [31:0] addr   ,
        input  [31:0] dataI  ,
        output [31:0] data0
 endmodule;
```

## 1.4 Store Byte and Store Half

The STRB instruction stores the low-byte of a register to memory. The STRH instruction is similar, except it stores the low two bytes of a register to memory. Use the provided data memory wrapper which is modified to support the STRB and STRH instructions. The data memory (ram) component includes a 4-bit port called *be* (Byte-enable) which each bit corresponds to the valid byte to be written in the memory. You will have to modify your control unit to set *be* signal appropriately. Refer to page 315 and 316 of supplemental reference book for more information.

## 1.5 Load Byte and Load Half

The LDRB instruction returns one byte which is zero-extended. In case of LDRSB (Load Signed Byte), the read byte data should sign-extend. In other words, if I request the byte at address 0x03 and the word at that location is 0x80000000, It should return 0xFFFFFF80. The instruction lets you specify any byte within the four-byte word (using the last 2 bits of the address). Refer to page 315 and 316 of supplemental reference book for more information.

### 1.5.1 ALU

Table 2 lists the operations it can perform and what value of *Function* each corresponds to. Refer to Appendex-B of supplemental reference book to get the complete information.

Table 2: Data-Processing supported instructions (ALU operations)

| cmd | Name | Description | Operation |
|---|---|---|---|
| 0000 | AND Rd, Rn, Src2 | Bitwise AND | Rd ← Rn & Src2 |
| 0001 | EOR Rd, Rn, Src2 | Bitwise XOR | Rd ← Rn ∧ Src2 |
| 0010 | SUB Rd, Rn, Src2 | Subtract | Rd ← Rn − Src2 |
| 0011 | RSB Rd, Rn, Src2 | Reverse Subtract | Rd ← Src2 − Rn |
| 0100 | ADD Rd, Rn, Src2 | Add | Rd ← Rn + Src2 |
| 0101 | ADC Rd, Rn, Src2 | Add with Carry | Rd ← Rn + Src2 + C |
| 0110 | SBC Rd, Rn, Src2 | Subtract with Carry | Rd ← Rn − Src2 − C |
| 0111 | RSC Rd, Rn, Src2 | Reverse Sub w/ Carry | Rd ← Src2 − Rn − C |
| 1000 (S : 1) | TST Rd, Rn, Src2 | Test | Set flags based on Rn & Src2 |
| 1001 (S : 1) | TEQ Rd, Rn, Src2 | Test Equivalence | Set flags based on Rn ∧ Src2 |
| 1010 (S : 1) | CMP Rn, Src2 | Compare | Set flags based on Rn − Src2 |
| 1011 (S : 1) | CMN Rn, Src2 | Compare Negative | Set flags based on Rn + Src2 |
| 1100 | ORR Rd, Rn, Src2 | Bitwise OR | Rd ← Rn \| Src2 |
| 1101 | Shifts | MOV, LSR, LSL, ROR, ASR, ARX | Rd ← Src2 |
| 1110 | BIC Rd, Rn, Src2 | Bitwise Clear | Rd ← Rn & Src2 |
| 1111 | MVN Rd, Rn, Src2 | Bitwise NOT | Rd ← Rn |

## 1.6 Assignment Deliverables

Your submission should include the following:

- Block diagram of your processor design.
- SystemVerilog Testbench capable of preloading memories and verifying the correct functionality of the processor using couple of simple and complicated test ARM program.
- A comprehensive report of the Design and testbench architecture. In the report, include the information on how you have make sure the processor works fine, and if you have found any bugs in your design.
- In your report describe after running the sample program provided in this document, what was the expected/rea output of the program.
- Synthesize your processor and report the Area, Power, and the maximum frequency your processor operates.

**IMPORTANT**: only **ONE** submission per group is required and the group leader should be the submitter.

**Note1**: Compress all your files in "zip" or "tar" format and then submit the compressed file.

**Note2**: The compressed file should include **design, sim, verif, doc, syn** directories and the corresponding files inside each directory. Your report's tex (if prepared with Latex), drawing, and pdf files are expected to be inside **doc** directory. The synthesis files and reports should be located in **syn** directory. Only include the text files and remove the other unnecessary files when submitting.

**Note3**: Remember to include your Group ID and the name and student ID of each group member in the report. When in doubt about your group ID, refer to the spreadsheet uploaded on the course website.

**Note4**: Assuming that the parent directory that you are working under that is named `Lab-2`, the following command will compress it for you:

```
$  tar cvf lab-2.tar lab-2
```