BEE 425

Project Milestone One

Section AA
Spring 2020

Prepared by
Benjamin Coltrane
Young Beum Cho
Chris Gatata
04.14.2020

Instructor
Joseph Decuir

Abstract

In this lab, we will begin planning for designing a simple single-cycle microprocessor CPU. We will decide which hardware extensions to add, and provide a rough plan for implementing and testing this basic CPU and its extensions.

Introduction

In this lab, we will begin planning for the design of a simple single-cycle microprocessor CPU. A single-cycle microprocessor is one which executes a single instruction in a single cycle. Instructions could include data processing such as ADD or SUB, or AND and ORR. They could also include memory access instructions such as LDR or conditional instructions such as BL. In a subsequent lab, we will add extension instructions to a simple single-cycle ARM processor. In this lab, we discuss which instructions we will add, and our plan for implementing this.

Procedures and Planning

The extension we intend to introduce to our simple single-cycle microprocessor is we will add shift and rotate extensions. We will add a barrel shifter to the ALU data paths, and add instructions to the ARM microprocessor simulation to implement Logical Shift Right(LSR), Logical Shift Left(LSL), Arithmetic Shift Right(ASR), and Rotate Right(ROR).

In order to test the new extension instructions, we will modify our Lab 3 test program *memfile.asm* to test the output of the included instruction. At this time, we intend to test our program by taking advantage of the fact that shifting in binary is equivalent to multiplication. Further details are included in the **Resources** section.

When we finally implement our extension instructions, the logic we intend to include in our System Verilog program will allow for shifting along a binary number to properly implement the instructions LSR, LSL, ASR, and ROR.

As our group continues to refine our ARM assembly skills, we continue to meet approximately once per week to maintain our momentum in this class.

Conclusion

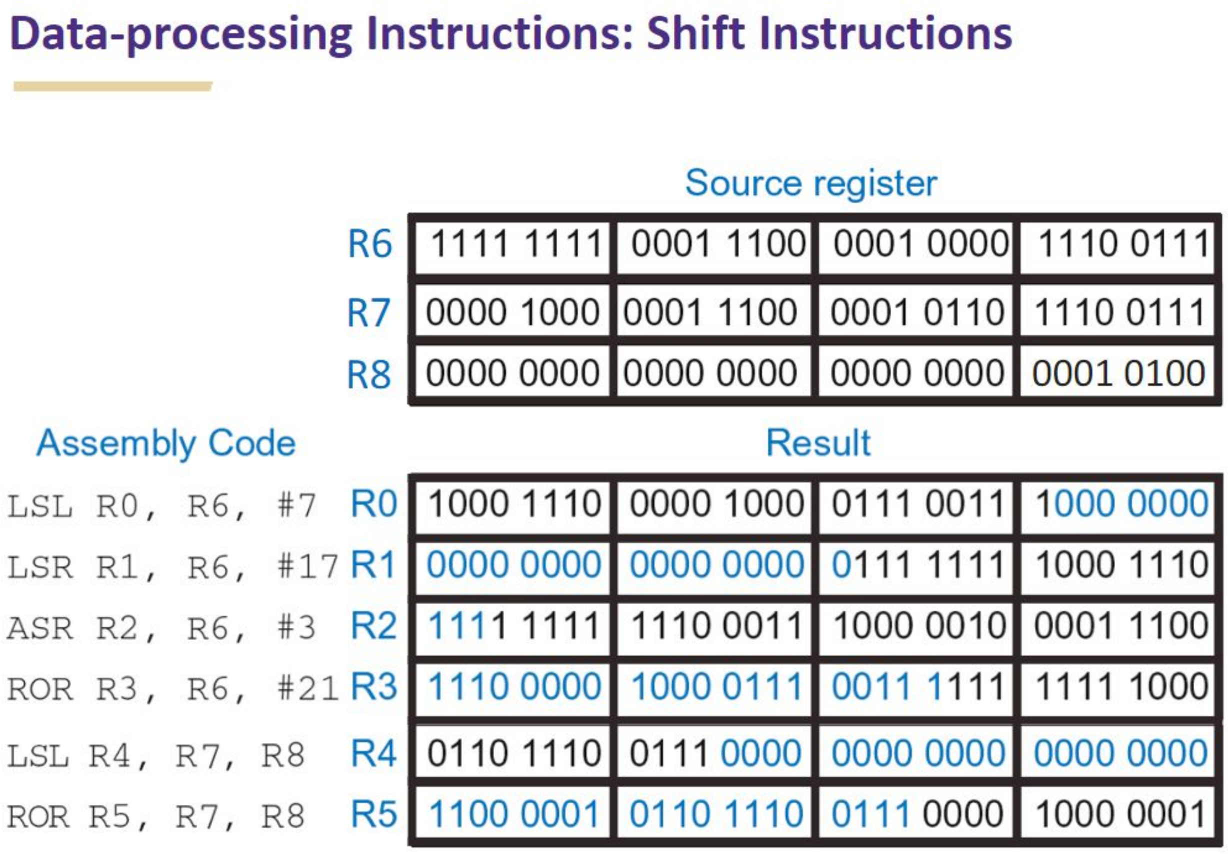
In this lab, we discussed how we intend to progress in Lab 3 and Lab 4. We decided which instruction extensions we will be using in Lab 3, and briefly discussed our SystemVerilog implementation in Lab 4.

Suggestions for Future Students

The main goal of this project for us is to use LSL, LSR, ASR, ROR functions implemented in a ARM assembly code to achieve particular purpose (e.g. calculating X * (2^n)).

It would be helpful for future students to have understanding of what various functions are capable of; shifting can be used for multiplying variable X by 2^n. Also, it would be helpful to reference simple and short codes which demonstrates scalabilities of functions.

Resources



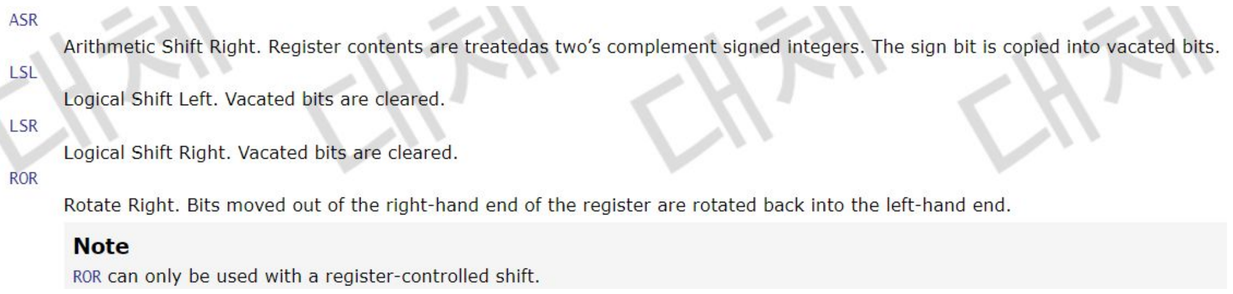
Data-processing Instructions: Shift Instructions

Source register

R6	1111 1111	0001 1100	0001 0000	1110 0111
R7	0000 1000	0001 1100	0001 0110	1110 0111
R8	0000 0000	0000 0000	0000 0000	0001 0100

Assembly Code	Result				
LSL R0, R6, #7	R0	1000 1110	0000 1000	0111 0011	1000 0000
LSR R1, R6, #17	R1	0000 0000	0000 0000	0111 1111	1000 1110
ASR R2, R6, #3	R2	1111 1111	1110 0011	1000 0010	0001 1100
ROR R3, R6, #21	R3	1110 0000	1000 0111	0011 1111	1111 1000
LSL R4, R7, R8	R4	0110 1110	0111 0000	0000 0000	0000 0000
ROR R5, R7, R8	R5	1100 0001	0110 1110	0111 0000	1000 0001

BEE 425 Lecture 7 Slide



ASR

Arithmetic Shift Right. Register contents are treatedas two's complement signed integers. The sign bit is copied into vacated bits.

LSL

Logical Shift Left. Vacated bits are cleared.

LSR

Logical Shift Right. Vacated bits are cleared.

ROR

Rotate Right. Bits moved out of the right-hand end of the register are rotated back into the left-hand end.

Note

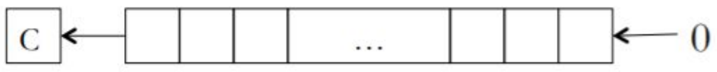
ROR can only be used with a register-controlled shift.

대체

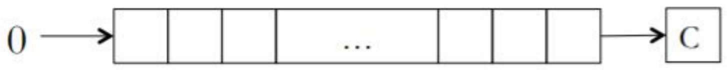
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0068b/BABEHCFA.html

---

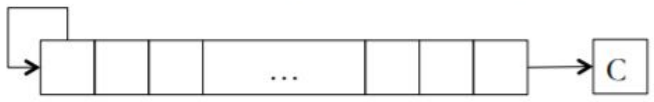
- LSL – logical shift by n bits – multiplication by 2^n



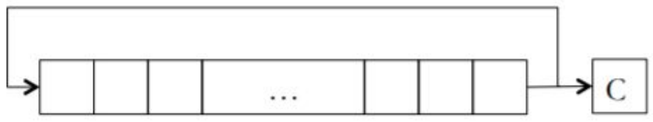
- LSR – logical shift by n bits – unsigned division by 2^n



- ASR – arithmetic shift by n bits – signed division by 2^n



- ROR – logical rotate by n bits – 32 bit rotate



https://cseweb.ucsd.edu/classes/wi14/cse30-c/lectures/PI_WI_14_CSE30_lecture_8_post.pdf

---

Sample Testing Algorithm

Process/Pseudocode:

```
Given, R1 = $2_{10}$, R2 = $32_{10}$, R3 = $-32_{10}$
Calculate $(2_{10} * 2^{2}) + (32_{10} * 2^{-2}) - (-32_{10} * 2^{-2})$. Assign the result as R5
                ↑LSL          ↑LSR           ↑ASR
From above, R5 should equal to $32_{10}$
Then, given/assigning R7 = 0000 1111 1111 1000 1111 1010 1111 1010
Then, the operation:  ROR R7, R7, R5
Should be the equivalent to R7 as before.
Thus, if the final output is,  R7 = 0000 1111 1111 1000 1111 1010 1111
1010
Then, LSL, LSR, ASR, ROR all works perfectly!
```