Lab RISC-V to ARM - Control

CMPUT 229

About this lab

- This is the second ARM translation lab and the final lab overall.
- In this lab, you will be translating control flow instructions from RISC-V to ARM. Specifically, branches and unconditional jumps.
- After finishing this lab, you're going to be able to translate a turing complete subset of RISC-V instructions into ARM.

Your tasks in this lab

- To create a translator that turns a subset of RISC-V instructions into ARM instructions. This subset includes branch and unconditional jump instructions, in addition to the instructions you translated in the previous lab.
- To implement a function decoding the RISC-V branch immediate (i.e. shifting the immediate bits into place).

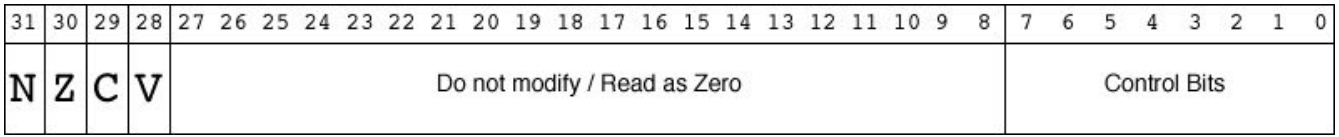
Recap

- The previous lab had you translating simple instructions, translating registers, and turning RISC-V immediates into a format usable by ARM immediate instructions.
- We glossed over the status and control bits in the last lab, requiring they be `0` and `1110` respectively.
    - These fields, however, are integral for the conditional execution in ARM, which we'll be using in this lab.

Branching in ARM

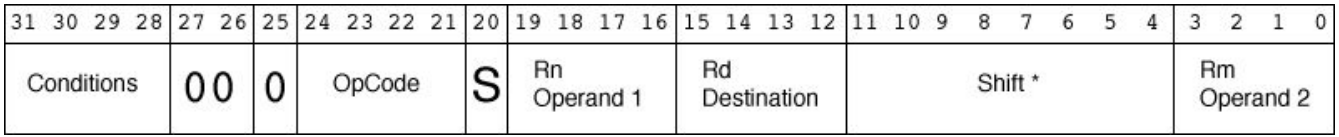
- The quantity of instructions to be dealt with in this lab is significantly smaller than what you saw in Lab RISC-V to ARM - ALU, but there's a reason for that: conditional execution of instructions.
- All instructions in ARM that have the **`Conditions`** field can be executed conditionally. Whether or not an instruction is executed depends on whether or not the condition represented by the **`Conditions`** field aligns with the CPSR register.

Current Program Status Register (CPSR)



- Though the CPSR has a variety of functions and fields, we only care about the 4 most significant bits.
- **N**, **Z**, **C** and **V** are set to 0 or 1 based on the result of instructions with status bit 1.
  - **N** will be set to 1 if the result of the instruction was negative.
  - **Z** will be set to 1 if the result of the instruction was equal to 0
  - **C** will be set to 1 if the instruction results in unsigned overflow.
  - **V** will be set to 1 if the instruction results in signed overflow.
- Once the **N**, **Z**, **C** and **V** bits are set to 1 by an instruction, they remain unchanged until another instruction with status bit 1 is executed.

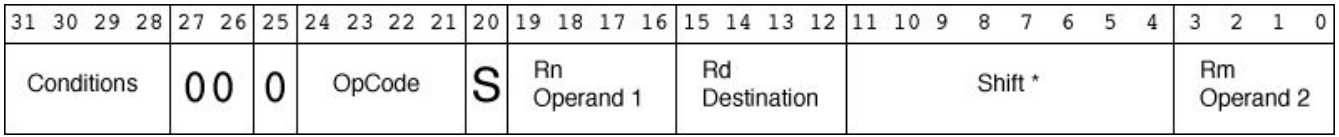
The CMP Instruction



31 30 29 28	27 26	25	24 23 22 21	20	19 18 17 16	15 14 13 12	11 10 9 8 7 6 5 4	3 2 1 0
Conditions	0 0	0	OpCode	S	Rn Operand 1	Rd Destination	Shift *	Rm Operand 2

- All instructions with an **S** bit can be made to alter the CSPR by setting the bit to 1.
- We are only interested in conditional execution for branches. As a result, we only need to have CSPR-altering instructions before branches.
- For simplicity, we're only going to use a single instruction for changing the CSPR. That instruction is the CMP instruction. That means that the CMP instruction must have its **S** bit set to 1.
- The CMP instruction is a signed subtraction (**Rn** - **Rm**) whose results aren't saved anywhere.
    - Its encoding belongs to the Data-Processing Register format, which is shown above.

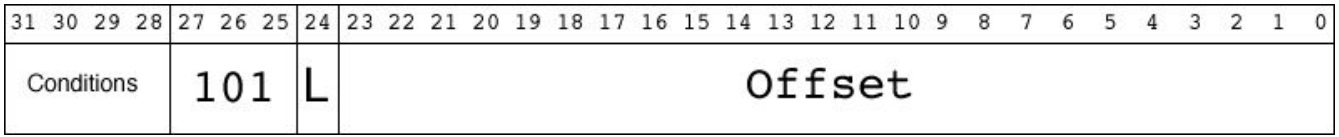
The **Conditions** field



- The **Conditions** field indicates when the instruction should be run. More specifically, it indicates the state that one or more of **N**, **Z**, **C** and **V** need to be in for the instruction to get executed.
- There are 16 possible **Conditions** field values, but for this lab, we only care about the following:

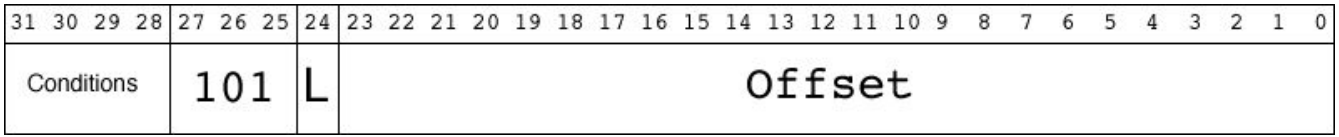
code	**flag check**	**meaning in terms of the CMP instruction**
1110	no flag check	execute no matter what
0000	**Z** is set	execute when **Rn** == **Rm** (i.e. **Rn** - **Rm** == 0)
1010	**N** equals **V**	execute when **Rn** >= **Rm** (i.e. **Rn** - **Rm** >= 0)

ARM Branch format



- The above is the branch instruction format in ARM.
- The L bit represents whether (1) or not (0) the address directly after the branch should be saved to the link register when the branch is taken.
- The Offset represents the value to be added to the current program counter in order to reach the target of the branch, however there are some quirks that must be considered.

ARM Branch format



- RISC-V branch instructions contain immediate bits 12-1, with bit 0 always implicitly set to 0. Upon instruction execution, this immediate is added to the PC.
- In ARM, the offset encoded in the instruction is shifted left by 2 bits (i.e. multiplied by 4) and then added to the PC.
- ARM, unlike RISC-V, always prefetches 2 instructions ahead, incrementing the program counter accordingly. This means that a branch at address `0x00000000` will be executed when the program counter is equal to `0x00000008`.
- Thus, since the offset is added to the program counter, it needs to be 8 smaller than the actual difference between the addresses of the branch and its target.

Translating Branches

- Branches are the only instructions that will need to be executed conditionally.
- Since each branch can be different than the one right before it, a CMP instruction needs to precede each one.
- As a result, each RISC-V branch must be translated into a CMP followed by a branch.
- This, however, creates a bit of a problem: how do we keep track of the branch offset?
    - Since there may be branch instructions between a branch and its target, the RISC-V offset alone is no longer sufficient.
    - We need an algorithm to allows us to keep track of where in the ARM binary the target is.

A two-pass approach

- Though you're free to implement any method to keep track of ARM branch targets, we suggest one involving two passes over a set of tables.
- You will use a RISC-V to ARM table to keep track of what address in the ARM binary the translated RISC-V instruction gets written to.
- You will also use a second table that, for each RISC-V branch, will keep track of the row address in the RISC-V to ARM table corresponding to its target.
- On the first pass, you will fill out the two tables, putting a 0 in the second table when an instruction isn't a branch. On the second pass, you will use the second table to track down the ARM address of the branch's target and, after calculating the correct offset for the address of the branch, fill it into the branch.

RISC-V Binary (shown disassembled)

0x00100100	`addi t0, t0, 2`
0x00100104	`addi s0, s1, -1`
0x00100108	`bge s1, t0, 4`
0x0010010C	`beq s0, t0, -8`

RISC-V to ARM Table

0x00100200	
0x00100204	
0x00100208	
0x0010020C	

Branch Table

0x00100300	
0x00100304	
0x00100308	
0x0010030C	

ARM Binary (shown disassembled)

0x00100400	
0x00100404	
0x00100408	
0x0010040C	
0x00100410	
0x00100414	

before first pass

RISC-V Binary (shown disassembled)

0x00100100	addi t0, t0, 2
0x00100104	addi s0, s1, -1
0x00100108	bge s1, t0, 4
0x0010010C	beq s0, t0, -8

RISC-V to ARM Table

0x00100200	0x00100400
0x00100204	0x00100404
0x00100208	0x00100408
0x0010020C	0x00100410

Branch Table

0x00100300	0
0x00100304	0
0x00100308	0x0010020C
0x0010030C	0x00100204

ARM Binary (shown disassembled)

0x00100400	ADD R0, R0, 2
0x00100404	SUB R3, R4, -1
0x00100408	CMP R4, R0
0x0010040C	BGE 0
0x00100410	CMP R3, R0
0x00100414	BEQ 0

after first pass

before second pass

RISC-V Binary (shown disassembled)

0x00100100	addi t0, t0, 2
0x00100104	addi s0, s1, -1
0x00100108	bge s1, t0, 4
0x0010010C	beq s0, t0, -8

RISC-V to ARM Table

0x00100200	0x00100400
0x00100204	0x00100404
0x00100208	0x00100408
0x0010020C	0x00100410

Branch Table

0x00100300	0
0x00100304	0
0x00100308	0x0010020C
0x0010030C	0x00100204

ARM Binary (shown disassembled)

0x00100400	ADD R0, R0, 2
0x00100404	SUB R3, R4, -1
0x00100408	CMP R4, R0
0x0010040C	BGE -0x04
0x00100410	CMP R3, R0
0x00100414	BEQ -0x18

after second pass

Tips

- Use the provided test cases, in addition to your own, to test for various edge cases.
- Make sure to get started early because it may take a while to wrap your head around how to implement the presented two-pass algorithm.

University of Alberta Code of Student Behavior

http://www.governance.ualberta.ca/en/CodesofConductandResidenceCommunityStandards/CodeofStudentBehaviour.aspx

30.3.2(1) Plagiarism

No Student shall submit the words, ideas, images or data of another person as the Student's own in any academic writing, essay, thesis, project, assignment, presentation or poster in a course or program of study.

30.3.2(2) Cheating

30.3.2(2) d No Student shall submit in any course or program of study, without the written approval of the course Instructor, all or a substantial portion of any academic writing, essay, thesis, research report, project, assignment, presentation or poster for which credit has previously been obtained by the Student or which has been or is being submitted by the Student in another course or program of study in the University or elsewhere

Overview

31	30	29	28	27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
N	Z	C	V	Do not modify / Read as Zero	Control Bits

31 30 29 28	27 26	25	24 23 22 21	20	19 18 17 16	15 14 13 12	11 10 9 8 7 6 5 4	3 2 1 0
Conditions	00	0	OpCode	S	Rn Operand 1	Rd Destination	Shift *	Rm Operand 2

31 30 29 28	27 26 25	24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Conditions	101	L	Offset

RISC-V Binary (shown disassembled)

0x00100100	addi t0, t0, 2
0x00100104	addi s0, s1, -1
0x00100108	bge s1, t0, 4
0x0010010C	beq s0, t0, -8

after second pass

RISC-V to ARM Table

0x00100200	0x00100400
0x00100204	0x00100404
0x00100208	0x00100408
0x0010020C	0x00100410

Branch Table

0x00100300	0
0x00100304	0
0x00100308	0x0010020C
0x0010030C	0x00100204

ARM Binary (shown disassembled)

0x00100400	ADD R0, R0, 2
0x00100404	SUB R3, R4, -1
0x00100408	CMP R4, R0
0x0010040C	BGE -0x04
0x00100410	CMP R3, R0
0x00100414	BEQ -0x18