Computer Architecture

Assignment #1: ARM Instruction Analysis

Purpose of Assignment #1

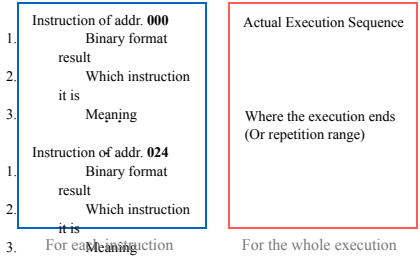
- To understand hardware instructions and assembly language
- To understand ARM instructions through the ARM reference manual
- To understand how instructions are executed at the system and hardware level

About report

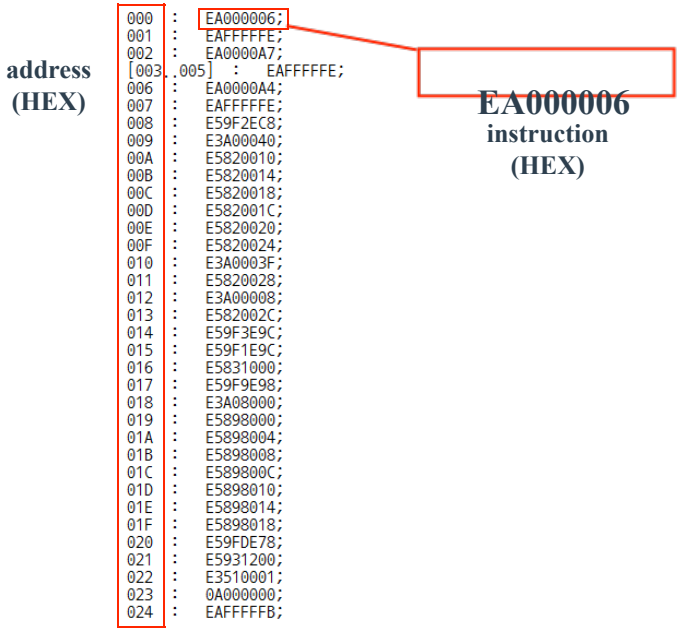
- **ARM instruction analysis report**

- **Analyze instruction of address 000~024 in instruction file(inst_data.mif)**

1. Change hex instructions to binary format and translate the binary instructions to assembly codes by referring to the ARM reference manual
2. Explain the meaning of each instruction
3. Explain the actual execution flow of the instructions
4. Explain where the execution ends (if it doesn't end, explain the range of repetition in detail)



inst_data.mif



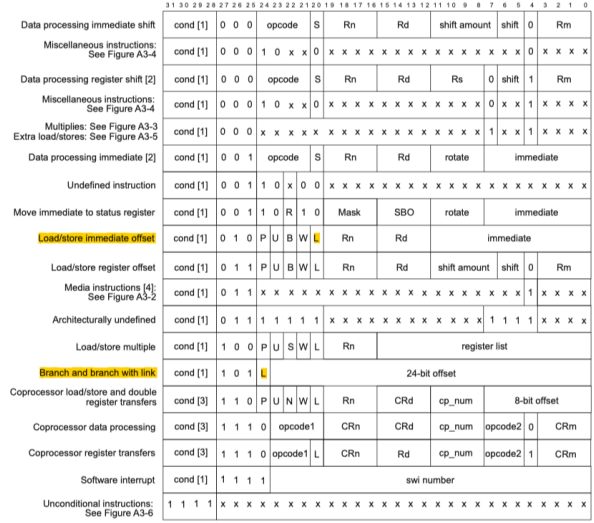
arm_architecture_reference_manual.pdf

A3.1 Instruction set encoding

Figure A3-1 shows the ARM instruction set encoding.

All other bit patterns are UNPREDICTABLE or UNDEFINED. See *Extending the instruction set* on page A3-32 for a description of the cases where instructions are UNDEFINED.

An entry in square brackets, for example [1], indicates that more information is given after the figure.



Chapter A4
ARM Instructions

This chapter describes the syntax and usage of every ARM® instruction, in the sections:

- *Alphabetical list of ARM instructions* on page A4-2
- *ARM instructions and architecture versions* on page A4-286.

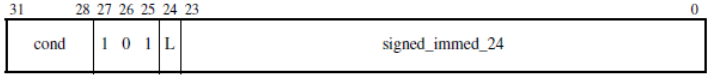
Example

- **EA000006**
  - **Change instructions to binary format**
    - 1110 **101**0 0000 0000 0000 0000 0000 0110 (2)
  - **Translate the binary instructions to assembly codes by referring to the reference file**
    - **B #008**;
  - **Describe what instruction means**
    - Move to the address current address*4+8+6x4 (In the case of RISC-V, add 4 to target address)
      - Because the address unit is 4 Byte (= 1 Word)
    - Therefore, move to the address (0 + 8 + 24) / 4 = 008
      - Next instruction will be E59F2EC8 at the address 008

Syntax

```
B{L}{<cond>}  <target_address>
```

A4.1.5 B, BL



Example

- **EAFFFFFE**

➢ **Change instructions to binary format**

- 1110 **101**0 1111 1111 1111 1111 1111 1110 (2)

➢ **Translate the binary instructions to assembly codes by referring to the reference file**

- B #**001**;

cf. -2 = 1111 1111 1111 1111 1111 1110

= 2's complement of 0000 0000 0000 0000 0000 0010

Step 1: Switching 0 to 1, 1 to 0
0000 0000 0000 0000 0000 0010
1111 1111 1111 1111 1111 1101
Step 2: Add 1 to the result of 1st Step
1111 1111 1111 1111 1111 1101
\+ 1
1111 1111 1111 1111 1111 1110

➢ **Describe what instruction means**

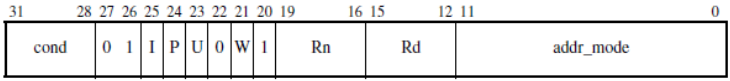
1. **Sign-extending** the **24-bit** signed(two's complement) immediate to **30 bits**
   - 1111 1111 1111 1111 1111 1110 —> **11 1111** 1111 1111 1111 1111 1111 1110
2. **Shifting** the result **left two** bits to form a **32-bit** value
   - 1111 1111 1111 1111 1111 1111 1111 10**00** = -210 * 4 = **-8**10

- Adding this to the **PC**, which contains the address of the branch instruction(current address) plus 8 bytes
  - Make '4' by adding the current instruction address '**(1*4)+8**' and '**-8**'
  - Divide '4' into 4 so that it branches at first among the word-unit instructions : 4/4 = 1
  - Because it branches to the same instruction, the same instruction repeats indefinitely

Example

- **E59FDE78**
  - ➢ **Change instruction to binary format**
  - 1110 **010**1 1001 **1111 1101 1110 0111 1000** (2)
  - ➢ **Translate the binary instructions to assembly codes by referring to the reference file**
  - LDR **$13**, [**$15**, **#0xE78**];
  - Mark it as # because I bit, the 25th bit, is 0
  - ➢ **Describe what instruction means**
  - Calculate the address of memory by adding the value stored in the **15th register** with **#0xE78**
  - Access to the calculated memory address and load the data into **the 13th register**
  - Read the value saved in address [**$15** + **#0xE78**] of memory and store it **$13**

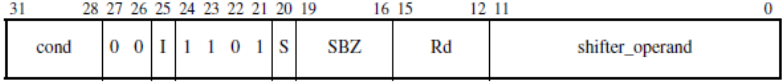
A4.1.23 LDR



Example

- **E1A0F00E**
  - **Change instruction to binary format**
    - 1110 0001 1010 0000 1111 0000 0000 1110 (2)
  - **Translate the binary instructions to assembly codes by referring to the reference file**
    - MOV **$15**, **$14**;
  - **Describe what instruction means**
    - Store **the value of the 14th register** at the **15th register**

A4.1.35 MOV



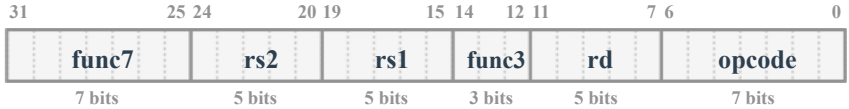
Hint

- **Condition Code (e.g., 0(0000) = Equal, E(1110) = Always)**
  - Refer to ARM Reference Manual p.112
- **Data Processing Instructions (e.g., MOV, CMP)**
  - **Opcode**: Refer to Manual p.115
  - **Instruction Encoding**: Refer to Manual p.116
- **Load and Store Instructions (e.g., LDR, STR)**
  - **Address Mode**: Refer to Manual p.129
  - **Instruction Encoding**: Refer to Manual p.130
  - **Examples**: Refer to Manual p.131
- **Branch Instructions (e.g., B, BL)**
  - **Instruction Encoding**: Refer to Manual p.160
  - **Examples**: Refer to Manual p.114

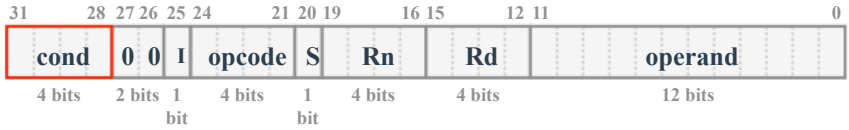
Differences in the ARM and RISC-V instructions

Differences Between ARM and RISC-V

- Different instruction field format is used
- RISC-V (R-type)



- ARM



What is "cond" field?

Differences Between ARM and RISC-V

- **Condition Field**

Refer to 'arm_architecture_reference_manual.pdf' A3.2.1

Table A3-1 Condition codes

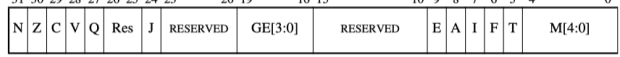
Opcode [31:28]	Mnemonic extension	Meaning	Condition flag state
0000	EQ	Equal	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set/unsigned higher or same	C set
0011	CC/LO	Carry clear/unsigned lower	C clear
0100	MI	Minus/negative	N set
0101	PL	Plus/positive or zero	N clear
0110	VS	Overflow	V set
0111	VC	No overflow	V clear
1000	HI	Unsigned higher	C set and Z clear
1001	LS	Unsigned lower or same	C clear or Z set
1010	GE	Signed greater than or equal	N set and V set, or N clear and V clear (N == V)
1011	LT	Signed less than	N set and V clear, or N clear and V set (N != V)
1100	GT	Signed greater than	Z clear, and either N set and V set, or N clear and V clear (Z == 0,N == V)
1101	LE	Signed less than or equal	Z set, or N set and V clear, or N clear and V set (Z == 1 or N != V)
1110	AL	Always (unconditional)	-
1111	-	See *Condition code 0b1111*	-

Differences Between ARM and RISC-V

- **Condition Code Flag**
  - Program Status Registers

Condition Code Flags

Format

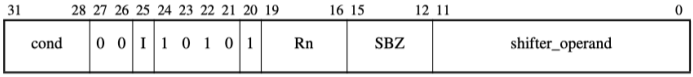


*Refer to A2.5 in manual

  - SPSR(Saved Progam Status Register), CPSR(Current Program Status Register)*
  - **N** - set if the result is negative
  - **Z - set if the result is zero**
  - **C** - set if carry exists in the results of addition, subtraction, shift (unsigned operation)
  - **V** - set if the result overflows from addition or subtraction (signed operation)
  - Operation with "S" suffix
    - ADDS, SUBS (ADD, SUB doesn't update flags)**

**Refer to A4.1.3 in manual

A4.1.15 CMP



CMP (Compare) compares two values. The first value comes from a register. The second value can be either an immediate value or a value from a register, and can be shifted before the comparison.

CMP updates the condition flags, based on the result of subtracting the second value from the first.

The 1st value == The 2nd value
The 1st value - The 2nd value = 0
==> **Z is set.**

Execution Flow

- **Example**
  - Consider following ARM assembly

MOV	r0, #2
CMP	r0, #3
ADDLT	r0, r0, #1
CMP	r0, #3
ADDLT	r0, r0, #1
BEQ	Somewhere

Syntax

```
MOV{<cond>}{S}  <Rd>, <shifter_operand>
CMP{<cond>}  <Rn>, <shifter_operand>
ADD{<cond>}{S}  <Rd>, <Rn>, <shifter_operand>
B{L}{<cond>}  <target_address>
```

r0	N	Z	C	V

MOV
>> R0 is 2
CMP
>> 'N' is set after this instruction because 'r0 - 3 < 0' is true.
ADDLT
>> This line is executed because **'cond' field** is met; r0 is now 3.
CMP
>> 'Z' is set because 'r0 == 3' is true
ADDLT
>> This line is not executed because the **'cond' field** is not met.
BEQ
>> This line is executed because 'Z' is met.

N	Is set to bit 31 of the result of the instruction. If this result is regarded as a two's complement signed integer, then N = 1 if the result is negative and N = 0 if it is positive or zero.
Z	Is set to 1 if the result of the instruction is zero (this often indicates an *equal* result from a comparison), and to 0 otherwise.

Differences Between ARM and RISC-V

- **Branch instruction**
  - **RISC-V**
    - Branching condition is tested and executed in one instruction
  - **ARM**
    - Branching condition is embedded in condition(cond) field
    - Branch is executed **only if "cond" condition is met**

Instruction Set Manual of ARM and RISC-V

- **ARM**
- https://iitd-plos.github.io/col718/ref/arm-instructionset.pdf

- **RISC-V**
- https://riscv.org/wp-content/uploads/2019/06/riscv-spec.pdf

Announcement

About Assignment #1

- **Report: 5 points**
- Analyzing instruction at 000~024 address in the instruction file (inst_data.mif)
- **Report should include the following**
- Student number and name
- For each instruction,
- Change the instruction from HEX to Binary (Score: 1)
- Translate the binary instructions to assembly codes by referring to the ARM reference manual and explain the meaning of each instruction (Score: 2)
- Explain the actual execution flow of the instructions (Score: 1)
- Specify where the execution ends (if not, specify the range repeated in detail) (Score: 1)

About Assignment #1

- **Write your answers by editing blue colored part.**

Address 001 | Instruction EAFFFFFE

a)→Change to binary format: YOUR ANSWER
b)→Write assembly code: YOUR ANSWER
c)→Describe why you wrote the assembly code like above: YOUR ANSWER
d)→What is the meaning of the instruction? : YOUR ANSWER

Address 002 | Instruction EA0000A7

a)→Change to binary format: YOUR ANSWER
b)→Write assembly code: YOUR ANSWER
c)→Describe why you wrote the assembly code like above: YOUR ANSWER
d)→What is the meaning of the instruction? : YOUR ANSWER

Explain the actual execution flow of the instructions(Address 000~024)

YOUR ANSWER

Specify where the execution ends (If not, specify the range repeated in detail)

YOUR ANSWER

About Execution Flow of Assignment #1

- As you cannot know which value is stored in memory, you may need to assume arbitrary (or temporary) values for the data transfer instructions, such as STR and LDR.

- Similarly, as you cannot know the exact value that may be stored in memory and/or registers, you may also need to consider both directions for conditional branch instructions, such as B and BL.

e.g., If the condition is satisfied, it may ~.
Otherwise, it may ~.
If branch will be executed, it may ~.