Computer Architecture

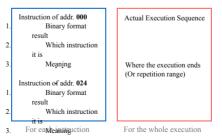
Assignment #1: ARM Instruction Analysis

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



inst_data.mif

address (HEX)

```
FA000006;
000
001
         EAFFFFFE;
002
         EA0000A7;
003
                EAFFFFFE;
     0051
006
         EA0000A4;
                                          EA000006
007
         EAFFFFFE;
008
         E59F2EC8;
                                           instruction
009
         E3A00040;
00A
         E5820010;
                                             (HEX)
00B
         E5820014;
00C
         E5820018;
00D
         E582001C;
00E
         E5820020;
00F
         E5820024;
010
         E3A0003F;
011
         F5820028;
012
         E3A00008;
013
         E582002C
014
         E59F3E9C;
015
         E59F1E9C;
016
         E5831000;
017
         E59F9E98;
018
019
01A
01B
01C
01D
01E
         E5898014;
01F
         E5898018;
020
         E59FDE78;
021
         E5931200;
022
         E3510001;
023
         0A000000;
024
         EAFFFFFB;
```

arm_architecture_reference_manual.pdf

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

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.

processing immediate shift	cond [1]	0	0	0		ро	ode	,	s		Rn			Rd			shift amount				nt shift			0		Rr	n	
Miscellaneous instructions: See Figure A3-4	cond [1]	0	0	0	1	0	×	×	0	x	хх	x	х	x >	()	()	()	()	ĸ	x	х	х	x	0	x	x	x	x
processing register shift [2]	cond [1]	0	0	0	١,	opc	ode	,	s	Г	Rn			Rd		Τ		Rs		T	0	sh	ift	1		Rr	n	٦
Miscellaneous instructions: See Figure A3-4	cond [1]	0	0	0	1	0	x	x	0	x	хх	х	х	x >	()	()	()		ĸ	x	0	х	x	1	×	x	x	х
Multiplies: See Figure A3-3 ad/stores: See Figure A3-5	cond [1]	0	0	0	×	×	×	×	×	×	хх	×	x	x >		()		()	<	x	1	x	х	1	×	×	х	×
a processing immediate [2]	cond [1]	0	0	1	opcode				s	Rn			Rd				rotate				imm			me	nediate			
Undefined instruction	cond [1]	0	0	1	1	0	×	0	0	×	хх	×	×	x >	()	()	()	<	×	x	×	x	×	x	x	x	×	x
mmediate to status register	cond [1]	0	0	1	1	0	R	1	0	Г	Mask		Г	SBC)	Τ	ro	otat	e	I			im	me	diat	e		
oad/store immediate offset	cond [1]	0	1	0	P U B W L Rn Rd				Rd		immediate																	
Load/store register offset	cond [1]	0	1	1	Р	U	В	w	L		Rn		Г	Rd		T	shi	ft a	ma	un	t	sh	ift	0		Rr	n	٦
Media instructions [4]: See Figure A3-2	cond [1]	0	1	1	×	x	×	×	×	х	× ×	x	x	×	()	()	()	()	x	x	x	x	x	1	x	x	x	x
Architecturally undefined	cond [1]	0	1	1	1	1	1	1	1	x	хх	×	x	x :	K 3	< :	ĸ	×	×	×	1	1	1	1	x	x	×	x
Load/store multiple	cond [1]	1	0	0	Р	U	s	w	L		Rn		Г						reg	iste	er I	ist						
ranch and branch with link	cond [1]	1	1 0 1 👢 24-bit offset												٦													
essor load/store and double register transfers	cond [3]	1	1	0	Р	U	N	w	L		Rn			CR	i	Τ	ср	nu	ım	T			8-	bit (offse	et		
processor data processing	cond [3]	1	1	1	0	0 opcode		1	CRn			CRd			T	cp_num			-	opcode2		0		CRm		٦		
rocessor register transfers	cond [3]	1	1	1	0	ор	coc	le1	L	Г	CRn		Г	Rd		T	ср	nı	ım		оро	ode	е2	1		CF	ζm	٦
Software interrupt	cond [1]	1	1	1	1	1 swi number									٦													
Unconditional instructions: See Figure A3-6	1 1 1 1	×	x	x	×	×	×	×	×	х	хх	x	х	x >	()	()	()	()	K	x	х	х	x	x	x	×	x	x
	processing register shift [2] Miscellaneous instructions: See Figure A3-4 Multiplies: See Figure A3-4 addistores: See Figure A3-3 conditional instructions (A1-2 Andistore register offset Media instructions: [4]- See Figure A3-2 Architecturally undefined Load/store multiple tranch and branch with link ssor load/store and double register transfers processor register transfers Software interrupt Unconditional instructions:	Miscellaneous instructions: See Figure A3-3 cond [1] cond	Miscellaneous instructions: See Figure AS4 See Figure AS4 Multiplies: See Figure AS4 Multiplies: See Figure AS4 Multiplies: See Figure AS4 Cond [1] 0 Multiplies: See Figure AS4 addistores: See Figure AS4 a processing immediate [2] Undefined instruction Undefined instruction Cond [1] 0 Cond [1] 1 Cond [1] 0 Cond [1] 1	Miscellaneous instructiona: See Figure A3-3 cond [1] 0 0 discellaneous instructiona: See Figure A3-3 cond [1] 0 0 cond [1] 0 1 cond [1] 0 1 discellaneous felse cond [1] 0 1 cond [1] 1 0 cond [1] 1 1 cond cond cond cond cond cond cond cond	Miscellaneous instructions: See Figure AS-3 Cond [1] 0 0 0 0 See Figure AS-3 Cond [1] 0 0 0 0 Multipleis: See Figure AS-3 Cond [1] 0 0 0 0 Multipleis: See Figure AS-3 Cond [1] 0 0 0 0 Multipleis: See Figure AS-3 Cond [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Miscellaneous instructions: See Figure A3-3 cond (†) 0 0 0 0 1 1 cond (†) 0 0 0 0 0 1 cond (†) 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Miscellaneous instructions:	Miscellaneous instructions: See Figure A3-4 Soe Figure A3-3 Soe Figure A3-3	Miscellaneous instructions: See Figure AS-3 Cond [1] 0 0 0 1 0 x x Cond [1] 0 0 0 0 1 0 x x Cond [1] 0 0 0 0 1 0 x x Cond [1] 0 0 0 0 1 0 x x Cond [1] 0 0 0 0 1 0 x x Cond [1] 0 0 0 0 1 0 x x Cond [1] 0 0 0 0 1 0 x x Cond [1] 0 0 0 1 0 x x x Cond [1] 0 0 0 1 0 x x x x Cond [1] 0 0 0 1 0 x x x x Cond [1] 0 0 0 1 0 x x x x Cond [1] 0 0 0 1 0 0 x x x x Cond [1] 0 0 0 1 0 0 x x x x x Cond [1] 0 0 0 1 0 0 x x x x x Cond [1] 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Miscellaneous instructions See Figure AS-4 Cond [1] 0 0 0 1 0 x x 0	Miscellaneous instructions: See Figure AS-3 cond [1] 0 0 0 1 0 x x 0 0 x corporate See Figure AS-3 cond [1] 0 0 0 1 0 x x 0 0 x corporate See Figure AS-3 cond [1] 0 0 0 1 0 x x 0 0 x corporate See Figure AS-3 cond [1] 0 0 0 1 0 x x x x x x x x x x x x x x x	Miscellaneous instructions: See Figure A.34 Cond [1] 0 0 0 1 0 x x 0 x x x x processing register with [2] Cond [1] 0 0 0 0 pccde S Rn	Miscellaneous instructions See Figure A3-4 Cond	Miscellaneous instructions: See Figure AS-3 Cond [1] 0 0 0 0 Docode S Rn	Miscellaneous instructions: See Figure A3-3 cond [1] 0 0 0 1 0 x x 0 x x x x x x	Miscellaneous instructions:	Miscellaneous instructions:	Miscellaneous instructions:	Miscellaneous instructions: Soe Figure A3-4 Cond [1] 0 0 0 0 0 0 0 0 0	Miscellaneous instructions: See Figure A3-3 See Figure A3-3	Miscellaneous instructions: See Figure A3-3 cond [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Miscellaneous instructions: See Figure A-34 Cond [1] 0 0 0 1 0 x x 0 x x x x x x x x x x x x x x x	Miscellaneous instructions: cond [1] 0 0 0 1 0 x x 0 x x x x x x x x x x x x	Miscellaneous instructions: See Figure A-34 Cond [1] 0 0 0 1 0 x x 0 x x x x x x x x x x x x x x	Miscellaneous instructions: Spee Figure A3-3 cond [1] 0 0 0 1 0 0 x 0 0 x x x x x x x x x x x	Miscellaneous instructions: See Figure A-3d Cond [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Miscellaneous instructions: cond [1] 0 0 0 1 0 x x 0 x x x x x x x x x x x x	Miscellaneous instructions: See Figure A-34 Cond [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

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.

- 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; Syntax
 - **Describe what instruction means**

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

Move to the address <u>current address</u>*4<u>+8</u>+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

A4.1.5 B, BL

EAFFFFFE

Change instructions to binary format

 $1110\ \underline{101}0\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110\ (2)$

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

Step 1: Switching 0 to 1, 1 to 0

```
B #001;
cf. -2 = 1111 1111 1111 1111 1110 1110
= 2's complement of 0000 0000 0000 0000 0000 0010
```

0000 0000 0000 0000 0000 0010 1111 1111 1111 1111 1101 Step 2: Add 1 to the result of 1st Step

Describe what instruction means

Sign-extending the 24-bit signed(two's complement) immediate to 301 bits 10

1111 1111 1111 1111 1111 1110 -> <u>11 1111</u> 1111 1111 1111 1111 1110

Shifting the result **left two** bits to form a **32-bit** value

1111 1111 1111 1111 1111 1111 1111 10 $\underline{00}$ = -210 * 4 = -810

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

E59FDE78

Change instruction to binary format

1110 <u>010</u>1 1001 <u>1111</u> 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

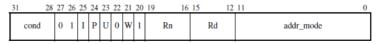
Describe what instruction means

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



- E1A0F00E
- Change instruction to binary format

 - 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

3	1 2	82	27	26	25	24	23	22	21	20	19	16	15	12	-11	l .	0
	cond		0	0	Ι	1	1	0	1	S	SBZ		Rd			shifter_operand	

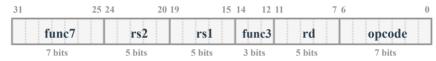
Hint

- 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

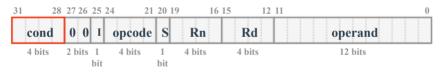
Differences in the

ARM and RISC-V instructions

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



ARM



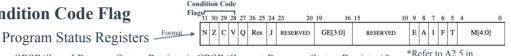
What is "cond" field?

Condition Field

Refer to 'arm_architecture_reference_manual.pdf' A3.2.1

		7	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	НІ	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 $(Z == 1 \text{ or } N != V)$
1110	AL	Always (unconditional)	-
1111	-	See Condition code 0b1111	-





- 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)

SPSR(Saved Progam Status Register), CPSR(Current Program Status Register)*

- 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



The 1st value == The 2nd value The 1st value - The 2nd value =

manual

==> Z is set.

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.

Execution Flow

Example

Consider following ARM assembly

MOV	r0, #2	MOV >> R0 is 2												
CMP	r0, #3	CMP >> 'N' is set after this instruction because 'r0 - 3 < 0' is true.												
ADDLT	r0, r0, #1	ADDLT >> This line is executed because 'cond' field is met; r0 is no												
CMP	r0, #3	3. CMP												
ADDLT	r0, r0, #1	>> 'Z' is set because 'r0 == 3' is true ADDLT												
BEQ	Somewhere	>>> 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.

Synte

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

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-

- XRM

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 ~.
```