

Chapter 8: Instruction Encoding and ISA

Mohamed M. Sabry Aly
N4-02c-92

Learning Objectives

- Describe the instruction format of ARM instruction-set architecture (ISA)
- Describe differences between fixed and variable-length ISA

Instruction Encoding

- For CPU to identify each unique assembly instruction, they must be encoded with unique binary patterns.
 - ARM 32-bit machine encode instructions in a 32-bit word
- What is included?
 - Instruction type → encoded
 - Operand(s) → encoded
 - Condition for conditional execution → encoded
 - Other flags?

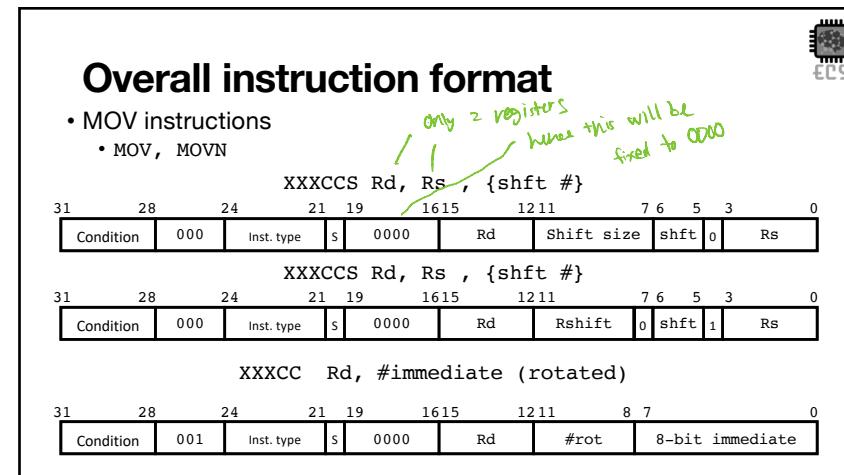
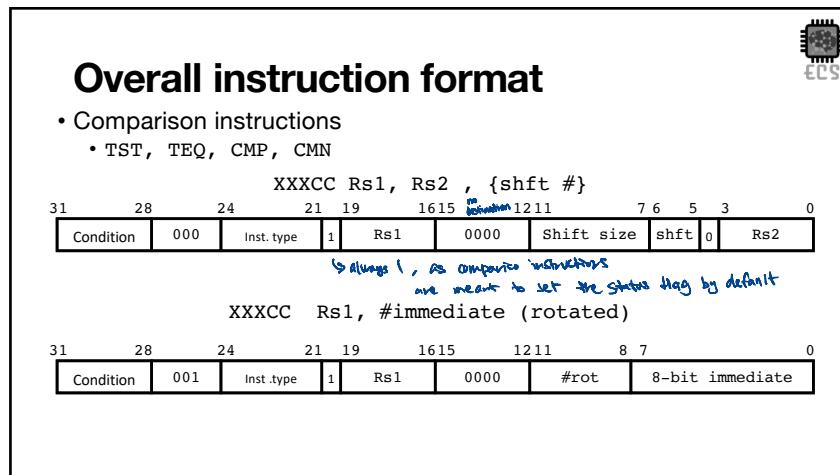
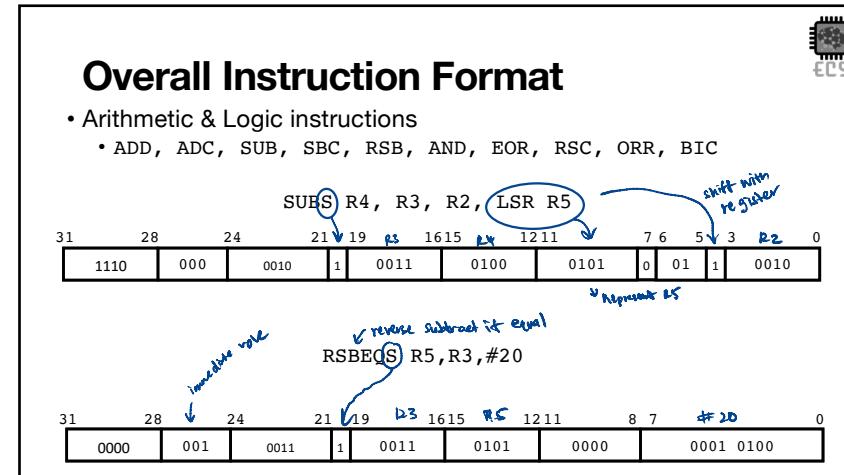
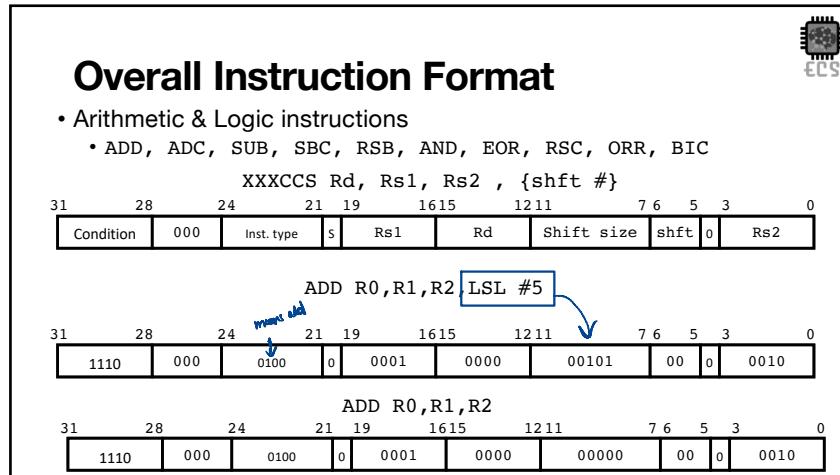
1 instruction = 32 bits

Overall Instruction Format

- Arithmetic & Logic instructions

• ADD, ADC, SUB, SBC, RSB, AND, EOR, RSC, ORR, BIC

XXXCCS Rd, Rs1, Rs2 , {shft #}												
31	rbits	28	24	21	19	1615	1211	7	6	5	3	0
Condition	000	Inst. type	s	Rs1	Rd	Shift size	shft 0				Rs2	
set source registers												
31	28	24	21	19	1615	1211	7	6	5	3	0	
Condition	000	Inst. type	s	Rs1	Rd	Rshift	shft 0	shft 1			Rs2	
0 rep shift with immediate value												
XXXCCS Rd, Rs1, Rs2 , {shift Rshift}												
31	28	24	21	19	1615	1211	7	6	5	3	0	
Condition	000	Inst. type	s	Rs1	Rd	Rshift	shft 0	shft 1			Rs2	
0 rep shift with register												
XXXCCS Rd, Rs1, #immediate (rotated)												
31	28	24	21	19	1615	1211	8	7			0	
Condition	001	Inst. type	s	Rs1	Rd	#rot	8-bit immediate					
Using immediate explicit instead of immediate value												
4 bit rotations												



Overall instruction format

- MOV instructions
 - MOV, MOVN

ns MOV R0,R5

31	28	24	21	19	1615	1211	7	6	5	3	0
1110	000	1101	0	0000	0000	(R0)	00000	00	0	0101	(R5)

MOVNE R1, R3, LSL(R2)

31	28	24	21	19	1615	R1	1211	7	6	5	3	0
0001	000	1101	0	0000	0001		0010	0	0	1	0011	

MOV R1,#0x40000000

31	28	24	21	19	1615	R1	1211	8	7	0	0
1110	001	1101	0	0000	0001		0100		0100	0000	

destination

Inst. Type field Field

4 bits for instructions → up to 16 combinations

4 bits for source and destination registers → 16 registers

4 bits

Code	Instruction
0000	AND
0001	EOR - exclusive OR
0010	SUB
0011	RSB - reverse subtract
0100	ADD
0101	ADC - add with carry
0110	SBC - subtract with carry
0111	RSC - reverse subtract with carry

Code	Instruction
1000	TST - Test
1001	TEQ - Test if equal 0
1010	CMP - compare
1011	CMN - compare negative
1100	ORR - or
1101	MOV - move
1110	BIC - bit inverse
1111	MVN - move negative

Condition Field

4 bits for condition → up to 16 difference combinations

4 bits

Code	Condition	Flags	Meaning
0000	EQ	Z = 1	Equal
0001	NE	Z = 0	Not equal
0010	CS or HS	C = 1	Higher or same, unsigned
0011	CC or LO	C = 0	Lower, unsigned
0100	MI	N = 1	Negative
0101	PL	N = 0	Positive or zero
0110	VS	V = 1	Overflow
0111	VC	V = 0	No overflow
1000	HI	C = 1 and Z = 0	Higher, unsigned
1001	LS	C = 0 or Z = 1	Lower or same, unsigned
1010	GE	N = V	Greater than or equal, signed
1011	LT	N != V	Less than, signed
1100	GT	Z = 0 and N = V	Greater than, signed
1101	LE	Z = 1 and N != V	Less than or equal, signed
1110	AL		Always.
1111	NV		Reserved (unused)

conditional assignment

What about shift instructions, what's their instruction format?

LSL, LSR, ASR, ROR, and RRX has no dedicated instructions

E.g. LSL R1, R2, #5 \equiv MOV R1, R2, LSL #5											
31	28	24	21	19	1615	1211	7	6	5	3	0
<code>1110 000 1101 S 0000 0001 00101 00 0 0010</code>											
• Specify the shift type in bits 5 and 6											
Code	Shift type										
00	LSL										
01	LSR										
10	ASR										
11	ROR/RRX										

Shift/Rotate instructions

- Instructions are encoded as MOV instructions

E.g. LSL R1, R2, #5 \equiv MOV R1, R2, LSL #5

- Specify the shift type in bits 5 and 6

Code	Shift type
00	LSL
01	LSR
10	ASR
11	ROR/RRX

rely on
use of
MOV
for shift

Branch Instructions

- For conditional branch and branch with link

31	28	23	B(L)CC	Offset	24 bits
Condition	101	1		Offset	

- 26 bits offset shifted to the right by 2 (branching to word addresses) \rightarrow calculated by the assembler
- Branch range: ± 32 MBytes

plus 1st set,
multiply by 4

Load/Store Instructions

- Load/Store word bye

LDR/STR{B}CC Rs,[RD,offset pre or post]W											
31	28	24	21	19	1615	1211	7	6	5	3	0
Condition	01	0	PUBW	L	Rs	Rd	Shift size	shft	0	Rs	

- L bit determines load (1) or store (0)
- P bit determines indexing pre (1) or post (0)
- U determines offset direction add (1) or subtract (0) offset
- B determines byte (1) or word (0) access
- W is for write back (1) of the offset

indicates

Load/Store Instructions

- Load/Store word bye

31	28	24	21	19	1615	1211	# 4	0
1110	01	1	1100	1	0010	0000	00000000100	

- L bit determines load (1) or store (0)
- P bit determines indexing pre (1) or post (0)
- U determines offset direction add (1) or subtract (0) offset
- B determines byte (1) or word (0) access
- W is for write back (1) of the offset (0, no write back)

LDR / STR : 011

Load/Store Instructions

- Load/Store word bye

STRB R0, [R2, #-4]!

31	28	24	21	19	1615	1211	0
1110	01 1	1011	0	0010	0000	00000000100	

Offset of 4

- L bit determines load (1) or store (0)
- P bit determines indexing pre (1) or post (0)
- U determines offset direction add (1) or subtract (0) offset
- B determines byte (1) or word (0) access
- W is for write back (!) of the offset (1) auto-indexing)

Load/Store Instructions

- Load/Store Multiple instructions

LDM/STMXCC RsW,{register list}

31	28	24	21	19	1615	1211	0
Condition	100	P U^ W	L	Rs	Register list		

• L bit determines load (1) or store (0)

• P bit determines indexing before (1) or after (0)

• U determines offset direction add (1) or subtract (0)

• ^ is “don’t care” → won’t care if it’s 1 or 0, as no bit transferred by words

• W is for write back (!) after operation

Load/Store Instructions

- Load/Store Multiple instructions

LDM/STMXCC RsW,{register list}

31	28	24	21	19	1615	1211	0
Condition	100	P U^ W	L	Rs	Register list		

- L bit determines load (1) or store (0)
- P bit determines indexing before (1) or after (0)
- U determines offset direction add (1) or subtract (0)
- ^ is “don’t care”
- W is for write back (!) after operation

PU	Instruction
10	STMFD
01	LDMFD
00	STMFA
11	LDMFA

- fully calculate offset and do subtraction
move down and do addition

Load/Store Instructions

- Load/Store Multiple instructions

STMFD SP!, {R0-R6}

31	28	24	21	19	1615	1211	0
1110	100	10^1	0	1101	0 0 0 0 0 0 0 0	(SP)	1 1 1 1 1 1 1 1

PC LR SP R12 R11 R10 R9 R8 R7 R6 R5 R4 R3 R2 R1 R0

- L bit determines load (1) or store (0)
- P bit determines indexing before (1) or after (0)
- U determines offset direction add (1) or subtract (0)
- ^ is “don’t care”
- W is for write back (!) after operation

✓ order does not matter

Not in Pre-lec vids (Mentioned in Lec)

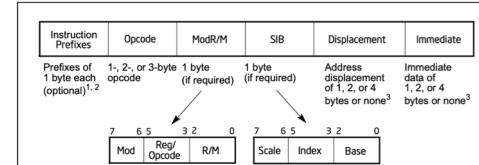
Fixed vs. Variable-Length instruction

- ARM is fixed-length
 - Operands had to be registers or immediate values
- Variable length instructions
 - Intel 80x86: Instructions vary from 1 to 17 bytes long.
 - Digital VAX: Instructions vary from 1 to 54 bytes long.
 - Require multi-step fetch and decode.
 - Allow for a more flexible (but complex) and compact instruction set.



Variable-Length: Intel Instruction Format

- Complex instructions = complex hardware
- Higher variety for smaller code and faster execution



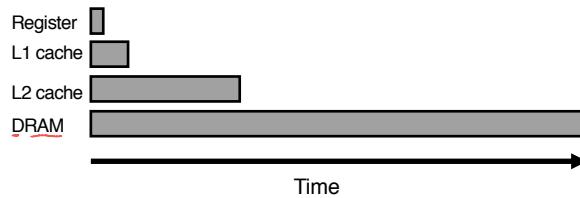
1. The REX prefix is optional, but if used must be immediately before the opicode; see Section 2.2.1, "REX-Prefixes" for additional information.
 2. For VEX encoding information, see Section 2.3, "Intel® Advanced Vector Extensions (Intel® AVX)".
 3. Some rare instructions can take an 8B immediate or 8B displacement.

Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format



Using Registers

- Operations using registers are **faster** than those involving memory.
 - Data transfer between registers and ALU is faster due to its physical proximity.
 - To speed up code execution, keep as much of your computations within **registers**.

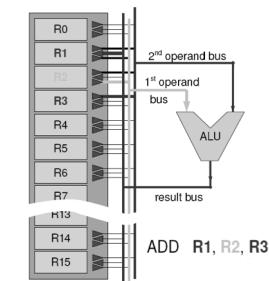


Implication of Register Count

- Implementing many registers within the CPU will incur increasing overheads.
 - They take up precious **space** in the silicon die.
 - Connections to various busses increase the **routing** and **multiplexing complexity**.
 - More registers increases the **operand size** during instruction encoding.

ADD (S)										8	6	5
11	14	13	12	11	10	9	8	6	5	3	2	1
0	0	0	1	1	0	0	Rm	Rn	Rd			
This form of ADD adds the value of one register to the value of a second register, and stores the result in a third register. The condition code flags are updated, based on the result.												
Syntax												
ADD <Rd>, <Rn>, <Rm>												
where:												
<Rd> Is the destination register for the completed operation.												
<Rn> Specifies the register containing the first value for the addition.												
<Rm> Specifies the register containing the second value for the addition.												

Instruction Encoding for ADD in Thumb-2



24

No. of pins relate to memory size not no. of registers

Orthogonality of ISA

- In a truly orthogonal ISA, every instruction is able to use every available addressing modes.
- A truly orthogonal ISA does not restrict certain instruction from using only specific registers.
- Difficult to achieve complete orthogonality due to the **large number of bit patterns** required to express all combinations of operations and **SPARC** addressing modes.
- Increase instruction length will increases the memory size required by the program.



25

Summary - Why is Good Instruction Set Architecture (ISA) Design Important?

- Good ISA design can yield benefits such as:
- Increases the execution performance of the processor.
- Increases the code density (i.e. how much memory a piece of code will occupy).
- Reduces the power consumption of the CPU.
- Reduces manufacturing cost of processor.
- Easy for programmers to write efficient programs.
- Efficient mapping of high-level programming requirements into low-level instruction sets.



26

Non - Examinable

Why Should I Care About Processor Designs and ISA?

- Designing a processor (or a co-processor) is no longer a hardware company job



25

Why Should I Care About Processor Designs and ISA?

- Designing a processor (or a co-processor) is no longer a hardware company job



The Google TPU

- Tensor Processing Unit
- Designed by Engineers in Google
- Tailored to make AI applications run faster

The diagram illustrates the internal architecture of the Google TPU. It shows the AI Chip (with Data Buffer, Computation, and Memory) connected to DRAM via a PCIe Gen3x8 interface. The AI Chip also connects to a Control Unit and a Unified Buffer Store. The Control Unit is connected to a Systolic Array Setup and a Matrix Multiply Unit. The Matrix Multiply Unit is connected to a Weight FFO (Weight Format Conversion) and Activation blocks. The Activation block is connected to Accumulators and Normalizes / Pool. The diagram also shows connections between the Control Unit, Unified Buffer Store, and the AI Chip. The total bandwidth for these connections is 147 Gbit/s.

The Google TPU

- Design by Engineers in Google to make AI applications run faster

The photograph shows a single Google TPU Dev Board on the left, which is a printed circuit board with a central chip and various components. On the right, there is a large server rack filled with multiple Google TPU units, demonstrating their use in a large-scale AI infrastructure.

Summary

- ARM is fixed-length load/store architecture
 - Operands reside in registers
- Other architectures can use variable-length format
 - E.g., Intel architecture
- Use of registers is key towards higher performance

Q1 which of the following hexadecimal encoding of ADD R2,R1,R3

~~1. 0xE00012003~~

~~2. 0xF1021003~~

~~3. 0xE00031200~~

~~4. 0x00012003~~

XXXCCS Rd, Rs1, Rs2 , {shft #}										
Condition	000	Inst. type	S	Rs1	Rd	Shift size	shft 0	shft 1	shft 2	shft 3
0010 CS or RS	0	000	0	000	000	0	0	0	0	0
0011 CC or LO	0	000	1	000	000	0	0	0	0	0

XXXCCS Rd, Rs1, Rs2 , {shft #}										
Condition	000	Inst. type	S	Rs1	Rd	Shift size	shft 0	shft 1	shft 2	shft 3
0010 CS or RS	0	000	0	000	000	0	0	0	0	0
0011 CC or LO	0	000	1	000	000	0	0	0	0	0

Code	Condition	Flags	Meaning
0000 EQ	Z = 1	Equal	
0000 NE	Z = 0	Not equal	
0010 CS or RS	C = 1	Higher or same, unsigned	
0011 CC or LO	C = 0	Lower, unsigned	
0100 HI	N = 1	Negative	
0101 PL	N = 0	Positive or zero	
0110 VS	V = 1	Overflow	
0111 VC	V = 0	No overflow	
1000 ET	C = 1 and Z = 0	Higher or same, unsigned	
1001 LT	C = 0 or Z = 1	Lower or same, unsigned	
1010 GE	N = V	Greater than or equal; signed	
1011 LT	N > V	Less than, signed	
1100 GT	Z = 0 and N < V	Greater than, signed	
1101 LE	Z = 1 and N >= V	Less than or equal, signed	
1110 AL	Z = 1	Always	
1111 MV		Reserved (unused)	

Code	Instruction
0000 AND	1000 TST
0001 EOR	1001 TEQ
0010 SUB	1010 CMP
0011 RSB	1011 CHN
0100 ADD	1100 ORR
0101 ADC	1101 MOV
0110 SBC	1110 BIC
0111 RSC	1111 MVN

Code	Instruction
0000 AND	1000 TST
0001 EOR	1001 TEQ
0010 SUB	1010 CMP
0011 RSB	1011 CHN
0100 ADD	1100 ORR
0101 ADC	1101 MOV
0110 SBC	1110 BIC
0111 RSC	1111 MVN

Address of registers:

R0 - 0000

R1 - 0001

R2 - 0010

R3 - 0011

ADD - no condition

⇒ does not mean condition = 0000

↳ cause 0000 indicates EQ

⇒ so must be condition = 1110

↳ aka E, indicates "Always"

XXXCCS Rd, Rs1, Rs2 , {shft #}										
Condition	000	Inst. type	S	Rs1	Rd	Shift size	shft 0	shft 1	shft 2	shft 3
1110	000	0100	0	1	2	0	0	0	0	3

↑ aka E,
"always"
ADD code
not ADDS

con't is
ADD,
not ADDS

no shifts

Q2 which of the following hexadecimal encoding of RSBEQS R2,R1,#5

ECS

← 8 bit immediate value

1. ~~0xF10121005~~

2. ~~0x027211005~~

3. ~~0X01321005~~

4. ~~0x012121005~~

Condition	000	Inst. type	Rel	Rd	Shift type	1615	1411	76	53	0
Condition	000	Inst. type	Rel	Rd	Shift type	1615	1411	76	53	0
Condition	000	Inst. type	Rel	Rd	Shift type	1615	1211	87	0	0

Code	Condition	Flags	Meaning
0000 EQ	Z = 1	Equal	
0001 NE	Z = 0	Not equal	
0010 CS or HS	C = 1	Higher or same, unsigned	
0011 CC or LS	C = 0	Lower, unsigned	
1000 MI	N = 1	Negative	
1010 GE	N = 0	Positive or zero	
0101 PL	V = 0	No overflow	
0110 VS	V = 1	Overflow	
0111 VC	V = 0	No overflow	
1000 HI	C = 1 and Z = 0	Higher, unsigned	
1011 LS	C = 0 and Z = 1	Lower or same, unsigned	
1010 GE	N < V	Greater than or equal, signed	
1011 LT	N > V	Less than, signed	
1100 GT	Z = 0 and N < V	Greater than, signed	
1101 LE	Z = 1 and N >= V	Less than or equal, signed	
1110 AL	Always	Reserved (unused)	
1111 NV			

Code	Instruction
0000 AND	1000 TST
0001 EOR	1001 TEQ
0010 SUB	1010 CMP
0011 RSB	1011 CNB
0100 ADD	1100 ORR
0101 ADC	1101 MOV
0110 SBC	1110 BIC
0111 RSC	1111 MVN

Code	Instruction
1000 TST	0000 AND
1001 TEQ	0001 EOR
1010 CMP	0010 SUB
1011 CNB	0011 RSB
1100 ORR	0100 ADD
1101 MOV	0101 ADC
1110 BIC	0110 SBC
1111 MVN	0111 RSC

0000 001 0011 1) 2 5
 2 7

XXXCCS Rd, Rs1, #immediate (rotated)

31	28	24	21	19	1615	1211	8	7	0
Condition	001	Inst. type	S	Rs1	Rd	#rot	8-bit immediate		

↳

0001

3rd operand

is a

register

Q3 which of the following hexadecimal encoding of MOVLE R7,R4

~~1. 0xD0D74000~~

~~2. 0xD0D70004~~

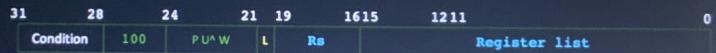
~~3. 0XD1A70004 ✓~~

~~4. 0xD1A74000~~

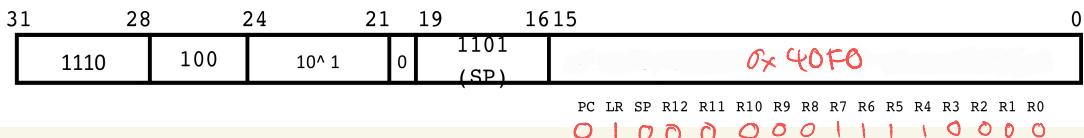
XXXCCS Rd, Rs , {shft #}										0
31	28	24	21	19	1615	1211	76	5	3	Rs
Condition	000	Inst. type	S	0000	Rd	Shift size	shft 0			
XXXCCS Rd, Rs , {shft #}										0
31	28	24	21	19	1615	1211	76	5	3	Rs
Condition	001	Inst. type	S	0000	Rd	Shift size	shft 1			
XXXCC Rd, #immediate (rotated)										0
31	28	24	21	19	1615	1211	8	7		0
Condition	001	Inst. type	S	0000	Rd	#imm	8-bit immediate			
Code	Condition	Flags	Meaning							
0000	EQ	Z = 1	Equal							
0001	NE	Z = 0	Not equal							
0010	GE or HS	C = 1	Higher or same, unsigned							
0011	LT or LE	C = 0	Lower or same, unsigned							
0100	MI	N = 1	Negative							
0101	PL	N = 0	Positive or zero							
0110	VR	V = 1	Overflow							
0111	VC	V = 0	No overflow							
1000	HI	C = 1 and Z = 0	Higher, unsigned							
1001	LS	C = 0 or Z = 1	Lower or same, unsigned							
1010	GE	N = 0	Greater than or equal, signed							
1011	LT	N != V	Less than, signed							
1100	GT	Z = 0 and N != V	Greater than, signed							
1101	LR	Z = 1 and N != V	Less than or equal, signed							
1110	AL	Z = 1	Always							
1111	MV		Reserved (unused)							

Code	Instruction
0000	AND
0001	EOR
0010	SUB
0011	RSB
0100	ADD
0101	ADC
0110	SBC
0111	RSC
1000	TST
1001	TEQ
1010	CMP
1011	CMN
1100	ORR
1101	MOV
1110	BIC
1111	MVN

Q4 If the registers list field is encoded with **0x40F0**, what register list this represents?



1. {R4, R15}
2. {R4-R7, LR} (circled)
3. {R1, R8-R11}
4. Don't know!



R4 - R7 , LR