

# **MODULE-7**

## **ARM Instruction Set**

**V.PRAKASH**

Asst. Professor(Sr.), SENSE,  
VIT Chennai

# MODULE-7

## ARM Instruction Set

ARM Instruction- data  
processing instructions, branch  
instructions, load store  
instructions, SWI Instruction,  
Loading instructions,  
conditional Execution,  
Assembly Programming.

# ARM INSTRUCTION SET

# ARM INSTRUCTION SET

## FEATURES

- Load-store architecture
- Conditional execution of every instruction
- Possible to load/store multiple registers at once
- Possible to combine shift and ALU operations in a single instruction
- 3-address instructions
- 2 or 3 operand field
- Number representation: Hexa-0x, Binary-0b
- The processor operations is illustrated using examples with pre- and post-conditions, describing registers and memory before and after the instruction are executed

*PRE* <pre-conditions>

<instruction/s>

*POST* <post-conditions>

# ARM INSTRUCTION SET

## ARM Instruction Set (ARMv5E) Classification

1. Load store instructions : LDR, STR
2. Data processing instructions
  - a) Move instructions : MOV, MVN
  - b) Barrel shift instructions : LSL, LSR, ASR, ROR, RRX
  - c) Arithmetic instructions : ADD, ADC, SUB, SBC, RSB, RSC
  - d) Multiply instructions : MUL, MLA
  - e) Logical instructions : AND, ORR, EOR, BIC
  - f) Comparison instructions : CMP, CMN
3. Branch instructions : B, BL
4. Conditional execution instructions : EQ, NE, CS, CC, MI, PL, VS, VC, HI, LS, GE, LT, GT, LE, AL
5. Software interrupts instruction
6. Status register transfer instruction
7. Coprocessor instructions



# ARM INSTRUCTION SET

## 1. LOAD STORE INSTRUCTIONS

- Load-store instructions transfer data between memory and processor registers or immediately load the value to registers.
- There are three types of load-store instructions:
  1. **Single-register transfer** - used for moving a single data item in and out of a register
  2. **Multiple-register transfer** - enable transfer of large quantities of data
  3. **Swap** - allow exchange between a register and memory in one instruction
- However, Single-Register Transfer instructions are most commonly used to perform load store operation.

**Syntax:** <LDR | STR>{<cond>}{B} Rd,addressing1

Instruction	Meaning	Example	Operation
LDR	Load word into a register	LDR r0, =0x0000000f	r0 := 0x0000000f
		LDR r0, [r1]	r0 := mem32[r1]
STR	Store word into a register	STR r0, [r1]	mem32[r1] := r0
		STR r2, [r5], #8	mem32[r5] = r2, r5 = r5 + 8

# ARM INSTRUCTION SET

## 2. DATA PROCESSING INSTRUCTIONS (a) MOVE INSTRUCTIONS

- Data processing instructions are **manipulate data within registers**.
- **Move copies N into a destination register Rd**, where N is a register or immediate value.
- This instruction is **useful for setting initial values** and transferring data between registers.
- If **suffix S used** on a data processing instruction, then it updates the flags in the CPSR.

**Syntax:** <instruction>{<cond>}{S} Rd, N

Instruction	Meaning	Operation	Example	Description
MOV	Move a 32-bit value into register	Rd=N	<b>PRE</b> r5=0x00000005, r7=0x00000008 MOV r7, r5 <b>POST</b> r5=0x00000005, r7=0x00000005	Move r5 value into r7
MVN	Move the NOT of the 32-bit value into register	Rd=~N	<b>PRE</b> r5=0x00000005, r7=0x00000008 MVN r7, r5 <b>POST</b> r5=0x00000005, r7=0xFFFFF0FA	NOT r5 value then move to r7

# ARM INSTRUCTION SET



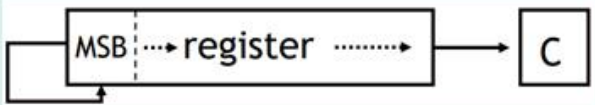
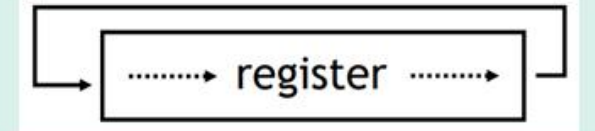

## 2. DATA PROCESSING INSTRUCTIONS (b) BARREL SHIFT INSTRUCTIONS

- A unique and powerful feature of the ARM processor is the ability to shift the 32-bit binary pattern in one of the source registers left or right by a specific number of positions before it enters the ALU.
- This shift increases the power and flexibility of many data processing operations.
- This is particularly useful for loading constants into a register and achieving fast multiplies or division by a power of 2.
- Most data processing instructions can process one of their operands using the barrel shifter.
- It is possible to directly specify the number of bits to be shifted in the instruction itself.



# ARM INSTRUCTION SET

## 2. DATA PROCESSING INSTRUCTIONS (b) BARREL SHIFT INSTRUCTIONS

Instruction	Meaning	Shift	Example
LSL	Logical Shift Left		<b>PRE</b> R2=0x00000030 MOV R0, R2, LSL #2 ; R0:=R2<<2 <b>POST</b> R0=0x000000C0, R2=0x00000030
LSR	Logical Shift Right		<b>PRE</b> R2=0x00000030 MOV R0, R2, LSR #2 ; R0:=R2>>2 <b>POST</b> R0=0x0000000C, R2=0x00000030
ASR	Arithmetic Shift Right		<b>PRE</b> R2=0x80000030 MOV R0, R2, ASR #2 ; R0:=R2>>2 <b>POST</b> R0=0xE000000C, R2=0x80000030
ROR	ROtate Right		<b>PRE</b> R2=0x00000031 MOV R0, R2, ROR #2 ; R0:=R2 rotate 2 <b>POST</b> R0=0x4000000C, R2=0x00000031
RRX	Rotate Right eXtended		<b>PRE</b> R2=0x00000031, C=1 MOV R0, R2, RRX ; R0:=R2 rotate 1, C <b>POST</b> R0=0x80000018, C=1, R2=0x00000031

# ARM INSTRUCTION SET

## 2. DATA PROCESSING INSTRUCTIONS (c) ARITHMETIC INSTRUCTIONS

- The arithmetic instructions implement **addition and subtraction of 32-bit values**.

**Syntax:** <instruction>{<cond>}{S} Rd, Rn, N

Instruction	Meaning	Operation	Example
ADD	Add two 32-bit values	$Rd = Rn + N$	<b>PRE</b> r1=0x00000005 ADD r0, r1, r1, LSL #1 ; r0=(r1<<1)+r1 <b>POST</b> r0=0x0000000F, r1=0x00000005
ADC	Add two 32-bit values and carry	$Rd = Rn + N + C$	<b>PRE</b> r1=0x00000005, C=1 ADC r0, r1, r1, LSL #1 ; r0=(r1<<1)+r1+C <b>POST</b> r0=0x00000010, r1=0x00000005
SUB	Subtract two 32-bit values	$Rd = Rn - N$	<b>PRE</b> r1=0x00000002, r2=0x00000001 SUB r0, r1, r2 ; r0=r1-r2 <b>POST</b> r0=0x00000001

N is the result of the shifter operation

# ARM INSTRUCTION SET

## 2. DATA PROCESSING INSTRUCTIONS (c) ARITHMETIC INSTRUCTIONS

Instruction	Meaning	Operation	Example
SBC	Subtract with carry of two 32-bit values	$Rd = Rn - N - !C$	<b>PRE</b> $r1=0x00000002, r2=0x00000001, C=0$ SBC $r0, r1, r2$ ; $r0=r1-r2-!C$ <b>POST</b> $r0=0x00000000$
RSB	Reverse Subtract two 32-bit values	$Rd = N - Rn$	<b>PRE</b> $r1=0x00000002, r2=0x00000005$ RSB $r0, r1, r2$ ; $r0=r2-r1$ <b>POST</b> $r0=0x00000003$
RSC	Reverse Subtract with carry of two 32-bit values	$Rd = N - Rn - !C$	<b>PRE</b> $r1=0x00000002, r2=0x00000005, C=0$ RSC $r0, r1, r2$ ; $r0=r2-r1-!C$ <b>POST</b> $r0=0x00000002$

N is the result of the shifter operation



# ARM INSTRUCTION SET

## 2. DATA PROCESSING INSTRUCTIONS (d) MULTIPLY INSTRUCTIONS

- The multiply instructions **multiply the contents of a pair of registers** and, depending upon the instruction, accumulate the results in with another register.

**Syntax:** **MUL**{<cond>}{S} Rd, Rm

**MLA**{<cond>}{S} Rd, Rm, Rs, Rn

Instruction	Meaning	Operation	Example
MUL	Multiply two 32-bit values	$Rd = Rm * Rs$	<b>PRE</b> r1=0x00000002, r2=0x00000002 MUL r0, r1, r2 ; r0=r1*r2 <b>POST</b> r0=0x00000004, r1=0x00000002, r2=0x00000002
MLA	Multiply two 32-bit values and accumulate	$Rd = (Rm * Rs) + Rn$	<b>PRE</b> r1=0x00000002, r2=0x00000002, r3=0x00000005 MLA r0, r1, r2, r3 ; r0=(r1*r2)+r3 <b>POST</b> r0=0x00000009, r1=0x00000002, r2=0x00000002, r3=0x00000005



# ARM INSTRUCTION SET

## 2. DATA PROCESSING INSTRUCTIONS (e) LOGICAL INSTRUCTIONS

- Logical instructions perform **bitwise logical operations** on the two source registers.

**Syntax:** <instruction>{<cond>}{S} Rd, Rn, N

Instruction	Meaning	Operation	Example
AND	Logical bitwise AND of two 32-bit values	$Rd = Rn \& N$	<b>PRE</b> r1=0x0000002F, r2=0x00000005 AND r0, r1, r2 ; r0=r1&r2 <b>POST</b> r0=0x00000005
ORR	Logical bitwise OR of two 32-bit values	$Rd = Rn   N$	<b>PRE</b> r1=0x0000002F, r2=0x00000005 ORR r0, r1, r2 ; r0=r1 r2 <b>POST</b> r0=0x0000002F
EOR	Logical bitwise XOR of two 32-bit values	$Rd = Rn \wedge N$	<b>PRE</b> r1=0x0000002F, r2=0x00000005 EOR r0, r1, r2 ; r0=r1^r2 <b>POST</b> r0=0x0000002A
BIC	Logical bit clear (AND NOT)	$Rd = Rn \& \sim N$	<b>PRE</b> r1=0x0000002F, r2=0x00000005 BIC r0, r1, r2 ; r0=r1&~r2 <b>POST</b> r0=0x0000002A

# ARM INSTRUCTION SET

## 2. DATA PROCESSING INSTRUCTIONS (f) COMPARISON INSTRUCTIONS

- The comparison instructions are used to **compare or test a register with a 32-bit value**.
- They **update the CPSR flag bits according to the result**, but do not affect register content.
- After the bits have been set, the information **can then be used to change program flow by using conditional execution**

**Syntax:** <instruction>{<cond>} Rn, N

Instruction	Meaning	Operation	Example	Description
CMP	Compare	CPSR flag sets as a result of Rn-N	<b>PRE</b> CPSR=nzcvqiFt_USER, r0=0x00000008, r9=0x00000008 CMP r0, r9 <b>POST</b> CPSR=nZcvqiFt_USER	Since r0-r9 result is zero, "Z" flag is SET
CMN	Compare negate	CPSR flag sets as a result of Rn+N	<b>PRE</b> CPSR=nzcvqiFt_USER, r0=0xA0000000, r9=0x80000000 CMN r0, r9 <b>POST</b> CPSR=nzCvqiFt_USER	Since r0+r9 result generate carry, "C" flag is SET

# ARM INSTRUCTION SET

## 3. BRANCH INSTRUCTIONS

- A branch instruction changes the flow of execution or is used to call a routine which forces the program counter (pc) to point to a new address.
- Branch instructions allows programs to have subroutines, if-then-else structures, and loops.

Syntax: B{<cond>} label

BL{<cond>} label

Instruction	Meaning	Operation	Example	Description
B	Branch	PC=label	B forward ..... forward .....	Branching forward
BL	Branch with Link	PC=Label LR=Address of the next instruction after BL	BL calc ..... <subroutine code> MOV pc, lr	Branching but after executing it return from subroutine using link register copied to PC



# ARM INSTRUCTION SET

## 4. CONDITIONAL EXECUTION INSTRUCTIONS

- A beneficial feature of the ARM architecture is that instructions can be made to execute conditionally.
- In ARM state, all instructions are conditionally executed according to the state of the CPSR condition codes and the instruction's condition field.
- If the state of the C, N, Z and V flags fulfils the conditions encoded by the field, the instruction is executed, otherwise it is ignored.
- There are 16 possible conditions, each represented by a two-character suffix that can be appended to the instruction's mnemonic. In practice, 15 different conditions may be used.
- For example, a Branch (B in assembly language) becomes BEQ for "Branch if Equal", which means the Branch will only be taken if the Z flag is set.



# ARM INSTRUCTION SET

## 4. CONDITIONAL EXECUTION INSTRUCTIONS

Code	Suffix	Flags	Meaning
0000	EQ	Z set	equal
0001	NE	Z clear	not equal
0010	CS	C set	unsigned higher or same
0011	CC	C clear	unsigned lower
0100	MI	N set	negative
0101	PL	N clear	positive or zero
0110	VS	V set	overflow
0111	VC	V clear	no overflow
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear or Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal to V	less than
1100	GT	Z clear AND (N equals V)	greater than
1101	LE	Z set OR (N not equal to V)	less than or equal
1110	AL	(ignored)	always

# ARM INSTRUCTION SET

## 4. CONDITIONAL EXECUTION INSTRUCTIONS

```
CMP    r0, #5      ; if (a == 5)
MOVEQ  r0, #10
BLEQ   fn          ; fn(10)
```

Assume a is in R0. Compare R0 to 5. The next two instructions will be executed only if the compare returns EQual. They move 10 into R0, then call 'fn' (branch with link, BL)

```
CMP    r0, #0      ; if (x <= 0)
MOVLE  r0, #0      ; x = 0;
MOVGT  r0, #1      ; else x = 1;
```

Assume x is in R0. Compare R0 to 0. The next instruction MOVLE will be executed only if the compare returns less than or equal. Later, check for Greater than condition and execute MOVGT instruction.

```
CMP    r0, #'A'    ; if (c == 'A')
CMPNE  r0, #'B'    ; || c == 'B')
MOVEQ  r1, #1      ; y = 1;
```

Assume A is in R0. Compare R0 to 'A' The next instruction CMPNE will be executed only if the compare returns Not Equal. They move 'B' into R0, then check for EQUAL condition for MOVEQ instruction execution.

# ASSEMBLY PROGRAMMING

# ASSEMBLY PROGRAMMING

## EXAMPLE – 1

- Write an ARM assembly language program to compute sum of first “n” numbers using the formula  $[n(n+1)/2]$ . Assume “n” as 5.

### PROGRAM

```
start      area sample,code, READONLY
           entry

           LDR r0, =0x00000005 ; N=5
           ADD r2, r0, #1
           MUL r3, r2, r0
           MOV r3, r3, LSR #1

stop       B stop
           END
```

### OUTPUT

Register	Value
Current	
R0	0x00000005
R1	0x00000000
R2	0x00000006
R3	0x0000000F
R4	0x00000000
R5	0x00000000
R6	0x00000000
--	-----



# ASSEMBLY PROGRAMMING

## EXAMPLE – 2

- Write an ARM assembly language program to compute addition of two 64-bit number.

### PROGRAM

```
area sample,code, READONLY
entry
CODE32

start

    LDR r0,=0x1111ffff ; LSB 32-bit 1st number
    LDR r1,=0x11110000 ; MSB 32-bit 1st number
    LDR r2,=0x1111ffff ; LSB 32-bit 2nd number
    LDR r3,=0x11110000 ; MSB 32-bit 2nd number
    ADDS r4,r2,r0       ; Add LSBs
    ADC r5,r3,r1        ; Add MSBs with carry

stop
    B stop
END
```

### OUTPUT

Register	Value
<b>Current</b>	
R0	0x1111FFFF
R1	0x11110000
R2	0x1111FFFF
R3	0x11110000
R4	0x2223FFFE
R5	0x22220000
R6	0x00000000
R7	0x00000000

# ASSEMBLY PROGRAMMING

## EXAMPLE – 3

- Write an ARM assembly language program to find the largest of two 32-bit numbers.

### PROGRAM

```
area sample, code, READONLY
entry

start

    LDR r0, =0x00000006    ; a
    LDR r1, =0x00000005    ; b
    CMP r0, r1
    MOVLT r4, r1
    MOVGT r4, r0

stop
    B stop
END
```

### OUTPUT

Register	Value
Current	
R0	0x00000006
R1	0x00000005
R2	0x00000000
R3	0x00000000
R4	0x00000006
R5	0x00000000
R6	0x00000000

# ASSEMBLY PROGRAMMING

## EXAMPLE – 4

- Write an equivalent ARM assembly language program for the “for loop function” given *for(i=0;i<15;i++) {j=j+j}.*

### PROGRAM

```
area sample,code, READONLY
entry

start
    LDR r0,=0x00000000    ; i
    LDR r1,=0x00000001    ; j
    MOV r2,#15            ; count=15
loop
    ADD r1,r1,r1          ; j=j+j
    ADD r0,r0,#1          ; i++
    CMP r2,r0             ; r2-r0
    BGT loop              ; i<15?
stop
    B stop
END
```

### OUTPUT

Register	Value
Current	
R0	0x0000000F
R1	0x00008000
R2	0x0000000F
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000

# ASSEMBLY PROGRAMMING

## EXAMPLE – 5

- Write an equivalent ARM assembly language program for the “if else function” given *if(a>5){x=5;y=c+d} else x=c-d*

### PROGRAM

```
area sample,code, READONLY
entry
start
    LDR r0,=0x00000003    ; a
    LDR r4,=0x00000001    ; c
    LDR r5,=0x00000002    ; d
    MOV r1,#5             ; count=5
    CMP r0,r1             ; r0-r1
    MOVLT r2,#5           ; x
    ADDLT r3,r4,r5        ; y=c+d
    SUBGT r2,r4,r5        ; y=c-d
stop
    B stop
END
```

### OUTPUT

Register	Value
Current	
R0	0x00000003
R1	0x00000005
R2	0x00000005
R3	0x00000003
R4	0x00000001
R5	0x00000002
R6	0x00000000



**THANK YOU**

THANK YOU



by  
*prakash v*