Overview of 32-bit ARM Architecture

```
        0    AND R0,R1, #0          //Clear R0
        4    OR R1,R0, #40          //Set R1 to numbers address
        8    LDRB R2, [R1,R0]       //Load multiplicand
        12   LDRB R3, [R1, #2]      //Load multiplier
        16   ADD R5,R0,R0           //Clear R5
Loop    20   ADD R5,R2,R5           //Add R2 and R5 cumulative
        24   SUBS R3,R3, #1         //Subtract multiplier
        28   BNE Loop               //Repeat iteration
        32   STRB R5, [R1,#3]       //Store result
        36   B End                  //Branch to Loc 44
        40
End     44   B End                  //Endless loop
```

Objective of this ARM Summary

- The objective of this summary is to provide an understanding of the function of the different fields of the binary code that correspond to the most relevant ARM instructions.
- This summary will be useful to understand how these fields control different components of the Pipelined Processing Unit described in this lesson.

Memory

- 2^{32} bytes
- Each instruction requires four bytes of memory and must begin on even addresses, multiples of four
- Integer Data Types
  - Byte
  - Halfword (16 bits)
  - Word (32 bits)
  - Doubleword (64 bits)
- Support both little-endian and big-endian format

Registers

- General Purpose R0 to R15
  - R14 – **Link Register** (return address for subroutines and exceptions)
  - R15 – **Program Counter**
- CPSR – current program status register (holds condition flags)

Instruction Set

- Data Processing
- Load/Store (Addressing Mode 2)
- Branches/Subroutines

Conditional Execution of Instructions

- All instructions specified a condition that has to be met in order to be executed.
- If the condition is not met the execution of the instruction is omitted. It become a no operation instruction (no-op in short)
- The condition is based on the Condition Flags of CPSR.
- The condition is specified with the most significant four bits of the word specifying the instruction

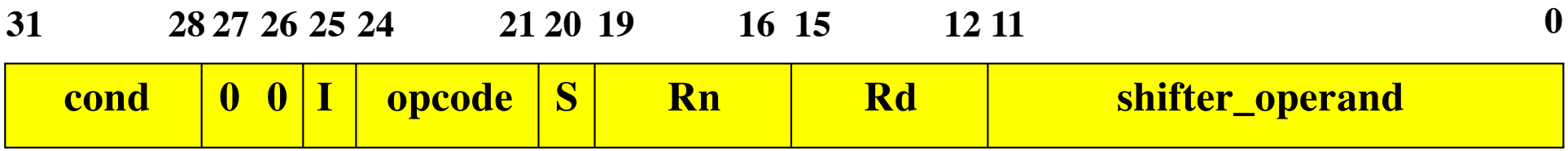
Instruction Condition Codes

Code	Suffix	Description	Flags tested
0000	EQ	Equal	Z=1
0001	NE	Not equal	Z=0
0010	CS/HS	Unsigned higher or same	C=1
0011	CC/LO	Unsigned lower	C=0
0100	MI	Minus	N=1
0101	PL	Positive or Zero	N=0
0110	VS	Overflow	V=1
0111	VC	No overflow	V=0
1000	HI	Unsigned higher	C=1 & Z=0
1001	LS	Unsigned lower or same	C=0 or Z=1
1010	GE	Greater or equal	N=V
1011	LT	Less than	N!=V
1100	GT	Greater than	Z=0 & N=V
1101	LE	Less than or equal	Z=1 or N=!V
1110	AL	Always	

Data-Processing Instructions

Opcode	Mnemonic	Operation	Action
0000	AND	Logical AND	Rd := Rn AND shifter_operand
0001	EOR	Logical Exclusive OR	Rd := Rn EOR shifter_operand
0010	SUB	Subtract	Rd := Rn - shifter_operand
0011	RSB	Reverse Subtract	Rd := shifter_operand – Rn
0100	ADD	Add	Rd := Rn + shifter_operand
0101	ADC	Add with Carry	Rd := Rn + shifter_operand + Carry
0110	SBC	Subtract with Carry	Rd := Rn - shifter_operand - NOT(Carry)
0111	RSC	Reverse Subtract with Carry	Rd := shifter_operand - Rn - NOT(Carry)
1000	TST	Test	Update flags after Rn AND shifter_operand
1001	TEQ	Test Equivalence	Update flags after Rn EOR shifter_operand
1010	CMP	Compare	Update flags after Rn - shifter_operand
1011	CMN	Compare Negated	Update flags after Rn + shifter_operand
1100	ORR	Logical Or	Rd := Rn OR shifter_operand
1101	MOV	Move	Rd := shifter_operand (no first operand)
1110	BIC	Bit Clear	Rd := Rn AND NOT(shifter_operand)
1111	MVN	Move Not	Rd := NOT shifter_operand (no first operand)

Instruction Encoding



cond ☛ condition code that must be satisfied for the instruction to executed

I ☛ Indicates if the shifter_operand is an immediate value or the content of a register

opcode ☛ the operation code of the instruction

S ☛ S = 1: condition codes can be modified

S = 0: condition codes can't be modified

Rn ☛ First source operand

Rd ☛ Destination register

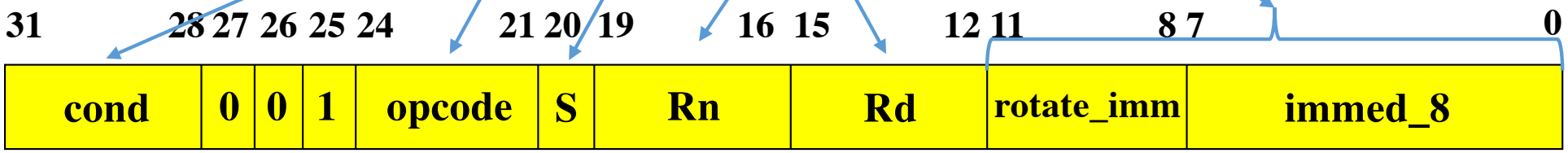
shifter_operand ☛ Second source operand (immediate number or content of register Rm)

32-bit Immediate Shifter Operand

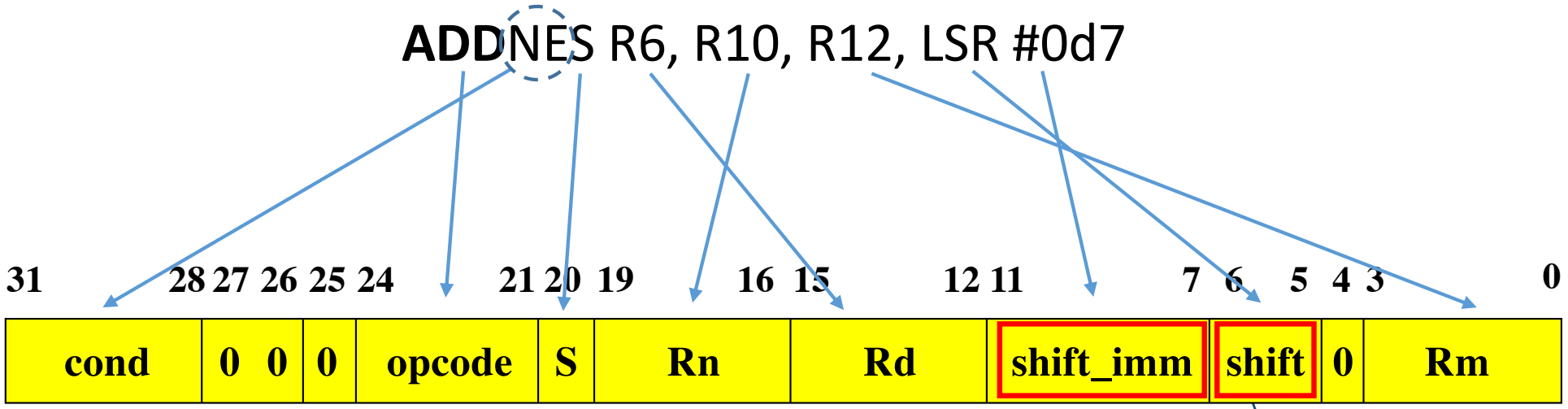
ADD R6, R10, #0d25

ADDNE R6, R10, #0d25

ADDNES R6, R10, #0d25



Shift by Immediate Shifter Operand



shifter_operand = content of ***Rm*** shifted ***shift_imm*** times

00 := LSL
01 := LSR
10 := ASR
11 := ROR

Load/Store Instructions (Addressing Mode 2)

- **Load/Store Word and Unsigned Byte**

  LDR - Load Word

  LDRB - Load Unsigned Byte

  LDRBT - Load Unsigned Byte with User Mode Privilege

  LDRT - Load Word with User Mode Privilege

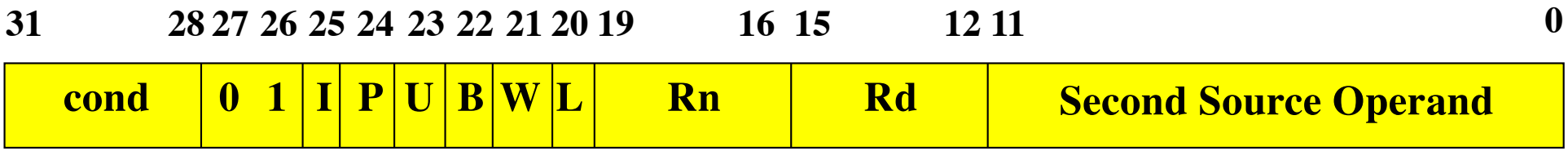
  STR - Store Word

  STRB Store Byte

  STRBT - Store Byte with User Mode Privilege

  STRT - Store Word with User Mode Privilege

Instruction Format



Bits **26** and **27** identify the instruction as a load/store instruction.

Bit **I** determines if the second source operand is an immediate number (**I**=0) or the content of register Rm (**I**=1)

Bit **L** determines if it is a Load (**L**=1) or a Store (**L**=0)

Bit **B** determines if the data type is a byte (**B**=1) or a word (**B**=0)

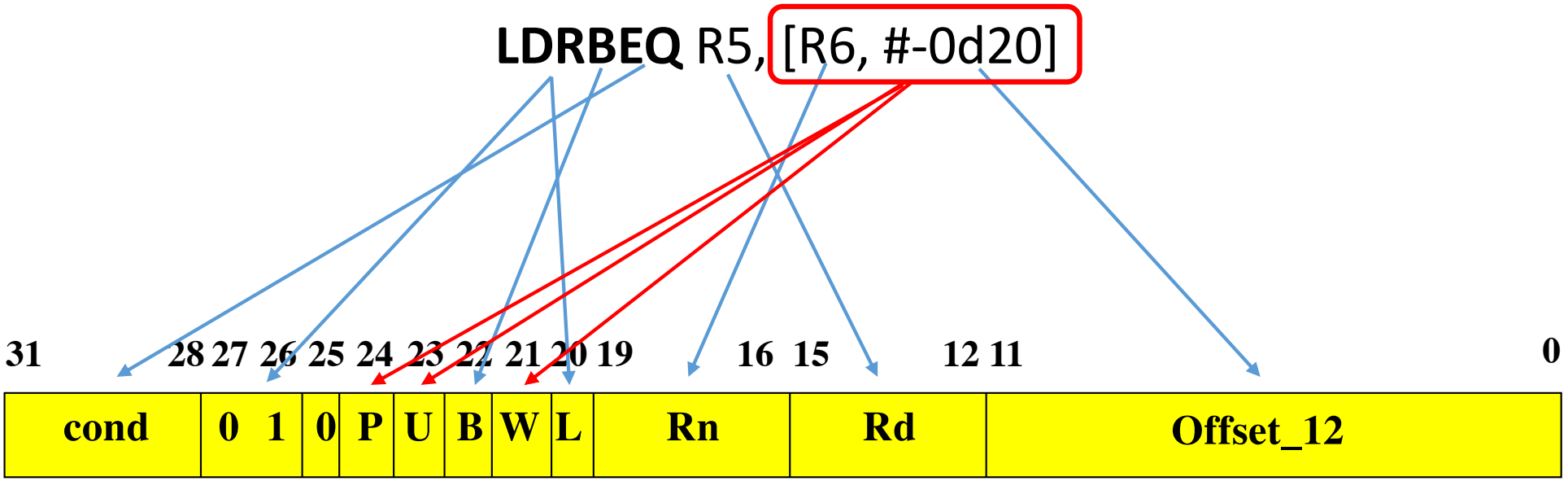
Bits **P**, **U** and **W** determine the addressing mode

Addressing Modes of Loads/Stores

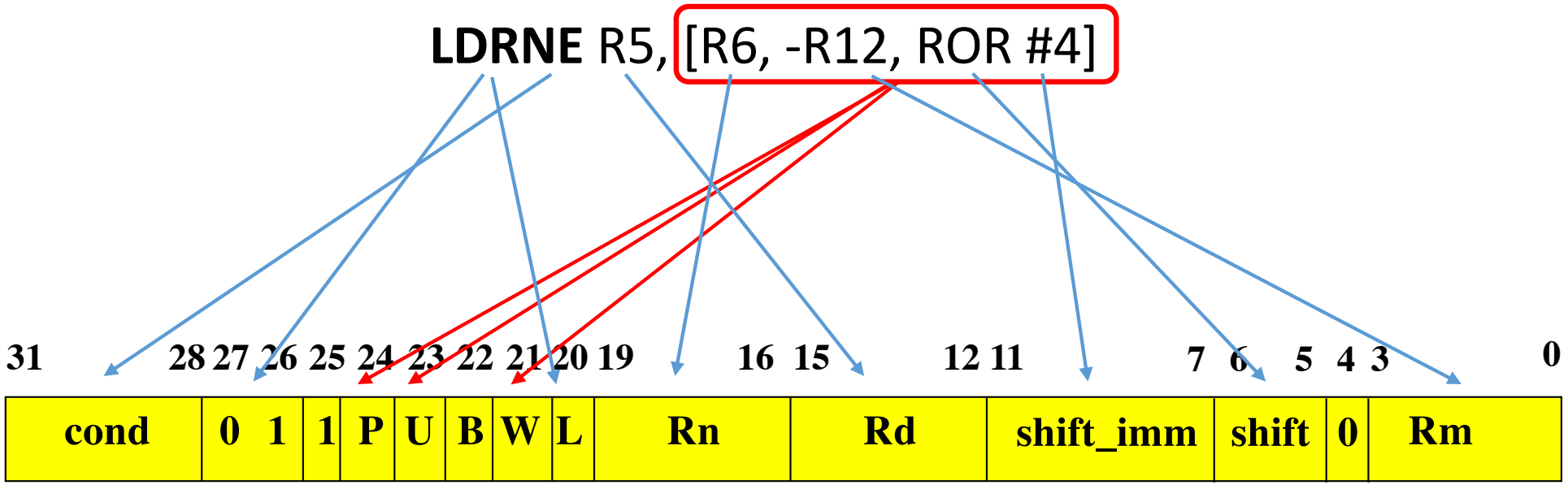
Determine how the second source operand is generated and how the base Rn is updated.

1. [<Rn>, #+/-<offset_12>]	*Immediate offset*
2. [<Rn>, +/-<Rm>]	*Register offset*
3. [<Rn>, +/-<Rm>, <shift> #<shift_imm>]	*Scaled register offset*
4. [<Rn>, #+/-<offset_12>]!	*Immediate pre-indexed*
5. [<Rn>, +/-<Rm>]!	*Register pre-indexed*
6. [<Rn>, +/-<Rm>, <shift> #<shift_imm>]!	*Scaled register pre-indexed*
7. [<Rn>], #+/-<offset_12>	*Immediate post-indexed*
8. [<Rn>], +/-<Rm>	*Register post-indexed*
9. [<Rn>], +/-<Rm>, <shift> #<shift_imm>	*Scaled register post-indexed*

Immediate Offset/Index

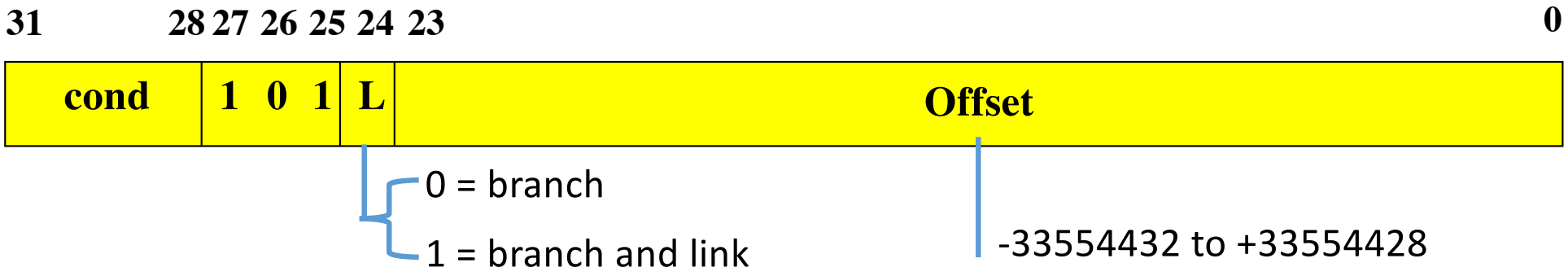


Scaled Register Offset/Index



Branch and Branch and Link Instruction Format

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



if Condition == true then

    if L == 1 then

        LR <= PC + 4

    PC <= target_address = PC + 8 + 4 x Offset (sign extended)

Else

    PC <= PC + 4 (NOP)