Fundamentals of Computer Architecture

Assembly Instruction Encoding

Teacher: Lobar Asretdinova

How to Encode Instructions?

- **Design Principle 1: Regularity supports design simplicity**
  - 64-bit data, 32-bit instructions
  - For design simplicity, would prefer a single instruction format but…
  - Instructions have different needs

Design Principle 4

Good design demands good compromises

- Multiple instruction formats allow flexibility
  - `ADD`, `SUB`: use 3 register operands
  - `LDR`, `STR`: use 2 register operands and a constant
- Number of instruction formats kept small
  - to adhere to design principles 1 and 3 (regularity supports design simplicity and smaller is faster)

Machine Language

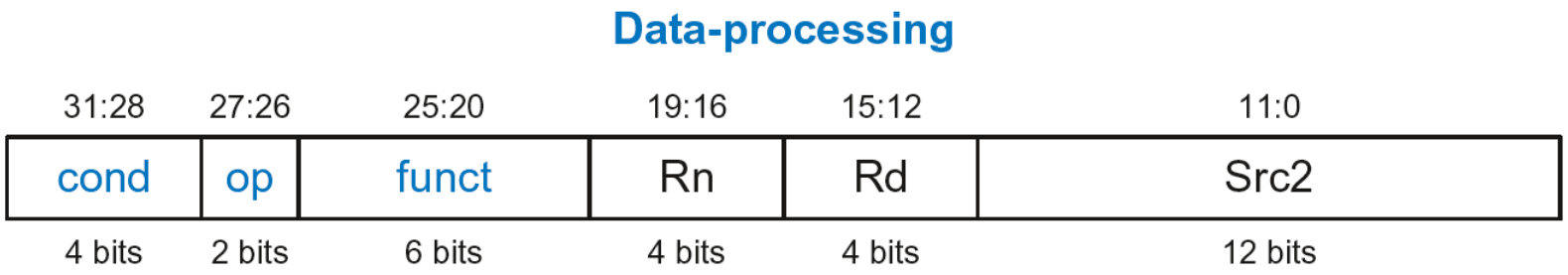
- **Binary representation of instructions**
- Computers only understand **1's and 0's**
- **32-bit instructions**
  - Simplicity favors regularity: 32-bit data & instructions
- **3 instruction formats:**
  - Data-processing
  - Memory
  - Branch

Instruction Formats

- **Data-processing**
- Memory
- Branch

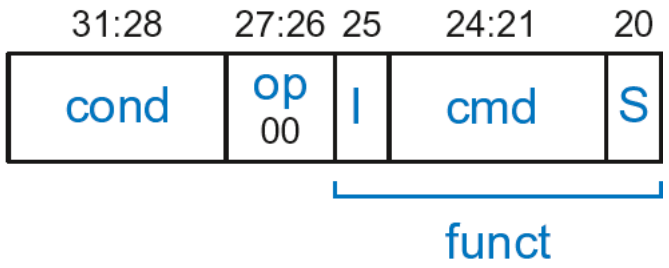
Data-processing Instruction Format

- **Operands:**
  - *Rn*: first source register
  - *Src2*: second source – register or immediate
  - *Rd*: destination register
- **Control fields:**
  - *cond*: specifies conditional execution
  - *op*: the *operation code* or *opcode*
  - *funct*: the *function*/operation to perform



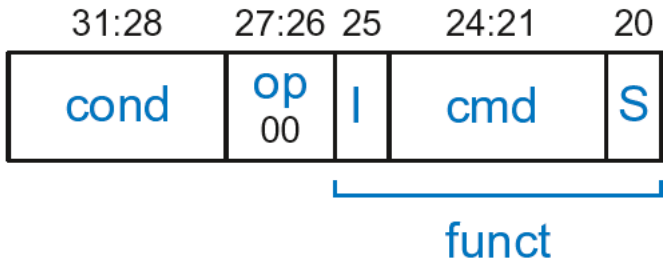
Data-processing Control Fields

- *op* = 00_2 for data-processing (DP) instructions
- *funct* is composed of *cmd*, *I*-bit, and *S*-bit



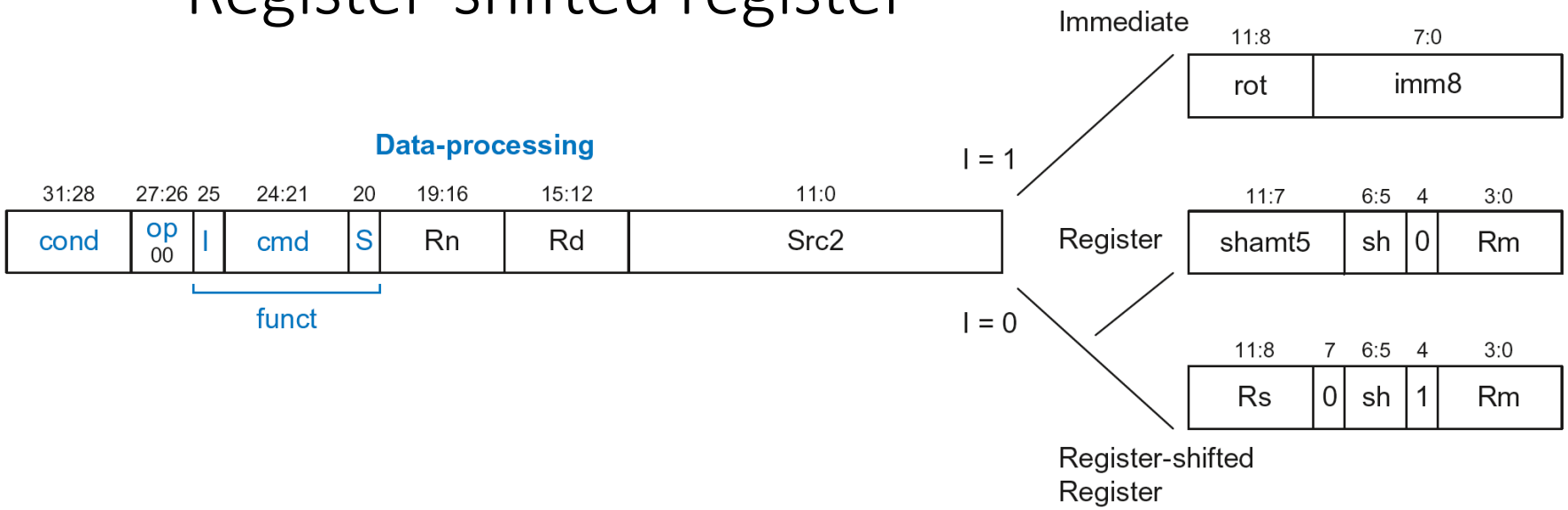
Data-processing Control Fields

- *op* = 00_2 for data-processing (DP) instructions
- *funct* is composed of *cmd*, *I*-bit, and *S*-bit
  - *cmd:* specifies the specific data-processing instruction. For example,
    - *cmd* = 0100_2 for `ADD`
    - *cmd* = 0010_2 for `SUB`
  - *I*-bit
    - *I* = 0: *Src2* is a register
    - *I* = 1: *Src2* is an immediate
  - *S*-bit: 1 if sets condition flags
    - ***S*** = **0:** `SUB  R0, R5, R7`
    - ***S*** = **1:** `ADDS R8, R2, R4` **or** `CMP R3, #10`



Data-processing *Src2* Variations

- *Src2* can be:
  - Immediate
  - Register
  - Register-shifted register



DP Instruction with Immediate *Src2*

```
ADD R0, R1, #42
```

- ***cond*** = 1110_2 (14) for unconditional execution
- ***op*** = 00_2 (0) for data-processing instructions
- ***cmd*** = 0100_2 (4) for ADD
- ***Src2*** is an immediate so ***I*** = 1
- ***Rd*** = 0, ***Rn*** = 1
- ***imm8*** = 42, ***rot*** = 0

Field Values

31:28	27:26	25	24:21	20	19:16	15:12	11:8	7:0
1110_2	00_2	1	0100_2	0	1	0	0	42
cond	op	I	cmd	S	Rn	Rd	shamt5	sh Rm
1110	00	1	0100	0	0001	0000	0000	00101010

DP Instruction with Immediate *Src2*

```
ADD R0, R1, #42
```

- ***cond*** = 1110_2 (14) for unconditional execution
- ***op*** = 00_2 (0) for data-processing instructions
- ***cmd*** = 0100_2 (4) for ADD
- ***Src2*** is an immediate so ***I*** = 1
- ***Rd*** = 0, ***Rn*** = 1
- ***imm8*** = 42, ***rot*** = 0

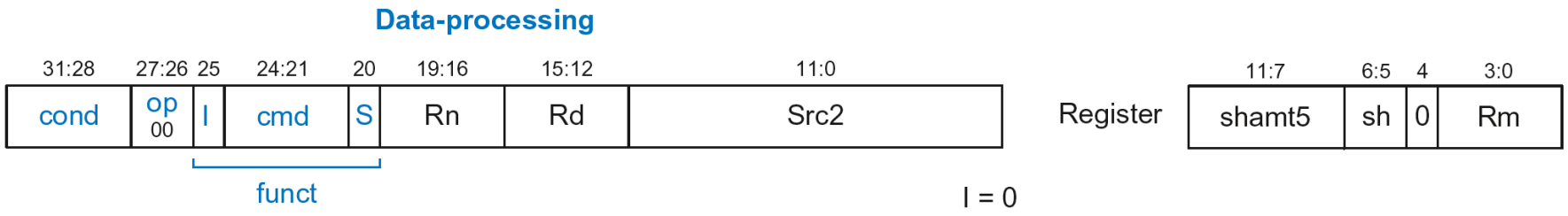
Field Values

31:28	27:26	25	24:21	20	19:16	15:12	11:8	7:0
1110_2	00_2	1	0100_2	0	1	0	0	42
cond	op	I	cmd	S	Rn	Rd	shamt5	sh Rm
1110	00	1	0100	0	0001	0000	0000	00101010

0xE281002A

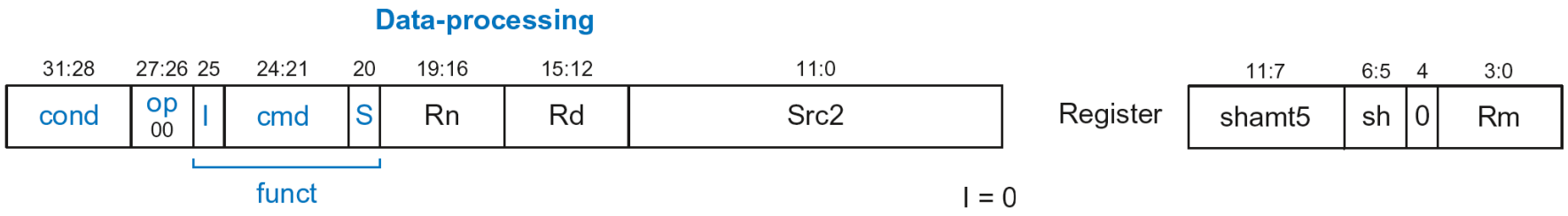
DP Instruction with Register *Src2*

- *Src2* can be:
  - Immediate
  - **Register**
  - Register-shifted register



DP Instruction with Register *Src2*

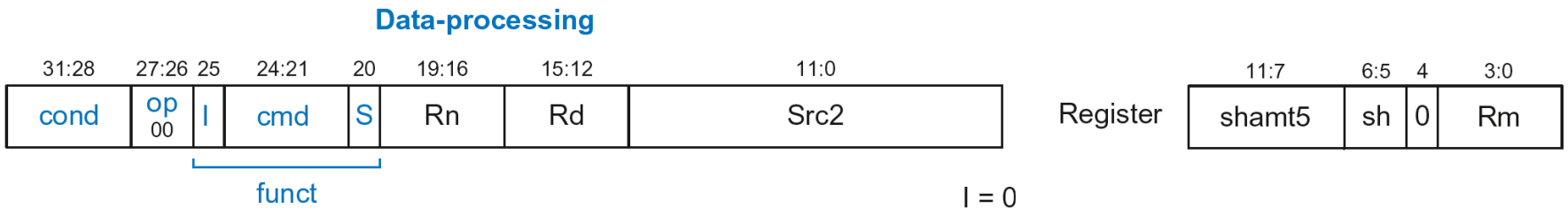
- ***Rm***: the second source operand
- ***shamt5***: the amount Rm is shifted
- ***sh***: the type of shift (i.e., >>, <<, >>>, ROR)



DP Instruction with Register *Src2*

- ***Rm***: the second source operand
- ***shamt5***: the amount rm is shifted
- ***sh***: the type of shift (i.e., >>, <<, >>>, ROR)

First, consider unshifted versions of *Rm* (*shamt5*=0, *sh*=0)



DP Instruction with Register *Src2*

`ADD R5, R6, R7`

- ***cond*** = 1110_2 (14) for unconditional execution
- ***op*** = 00_2 (0) for data-processing instructions
- ***cmd*** = 0100_2 (4) for `ADD`
- ***Src2*** is a register so ***I***=0
- ***Rd*** = 5, ***Rn*** = 6, ***Rm*** = 7
- ***shamt*** = 0, ***sh*** = 0

DP Instruction with Register *Src2*

`ADD R5, R6, R7`

- ***cond*** = 1110_2 (14) for unconditional execution
- ***op*** = 00_2 (0) for data-processing instructions
- ***cmd*** = 0100_2 (4) for `ADD`
- ***Src2*** is a register so ***I***=0
- ***Rd*** = 5, ***Rn*** = 6, ***Rm*** = 7
- ***shamt*** = 0, ***sh*** = 0

Field Values

31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0
1110_2	00_2	0	0100_2	0	6	5	0	0	0	7
cond	op	I	cmd	S	Rn	Rd	shamt5	sh		Rm
1110	00	0	0100	0	0110	0101	00000	00	0	0111

DP Instruction with Register *Src2*

`ADD R5, R6, R7`

- ***cond*** = 1110_2 (14) for unconditional execution
- ***op*** = 00_2 (0) for data-processing instructions
- ***cmd*** = 0100_2 (4) for `ADD`
- ***Src2*** is a register so ***I***=0
- ***Rd*** = 5, ***Rn*** = 6, ***Rm*** = 7
- ***shamt*** = 0, ***sh*** = 0

Field Values

31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0
1110_2	00_2	0	0100_2	0	6	5	0	0	0	7
cond	op	I	cmd	S	Rn	Rd	shamt5	sh		Rm
1110	00	0	0100	0	0110	0101	00000	00	0	0111

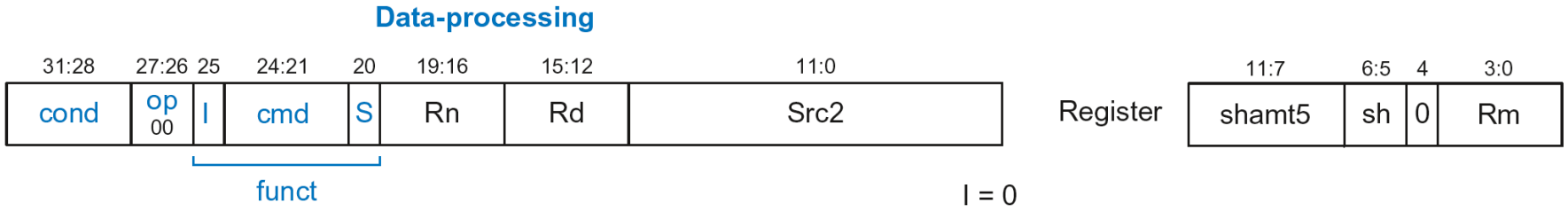
0xE0865007

DP Instruction with Register *Src2*

- ***Rm***: the second source operand
- ***shamt5***: the amount Rm is shifted
- ***sh***: the type of shift

Shift Type	*sh*
LSL	00_2
LSR	01_2
ASR	10_2
ROR	11_2

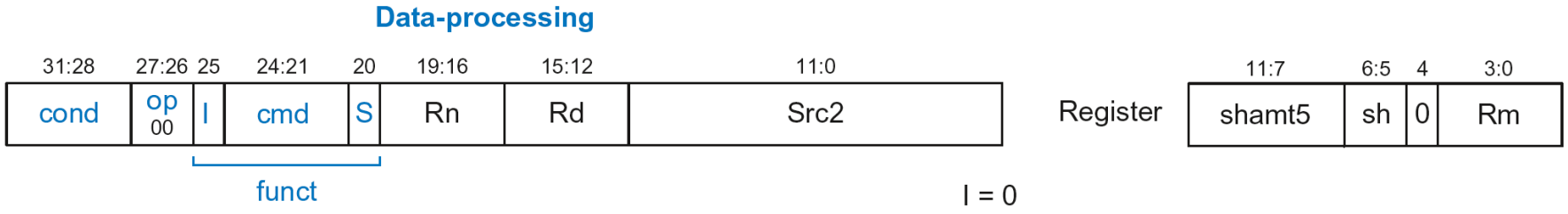
Now, consider shifted versions.



DP Instruction with Register *Src2*

`ORR R9, R5, R3,` **`LSR #2`**

- **Operation:** R9 = R5 OR (R3 >> 2)
- ***cond*** = 1110_2 (14) for unconditional execution
- ***op*** = 00_2 (0) for data-processing instructions
- ***cmd*** = 1100_2 (12) for `ORR`
- ***Src2*** is a register so ***I***=0
- ***Rd*** = 9, ***Rn*** = 5, ***Rm*** = 3
- ***shamt5*** = 2, ***sh*** = 01_2 (LSR)

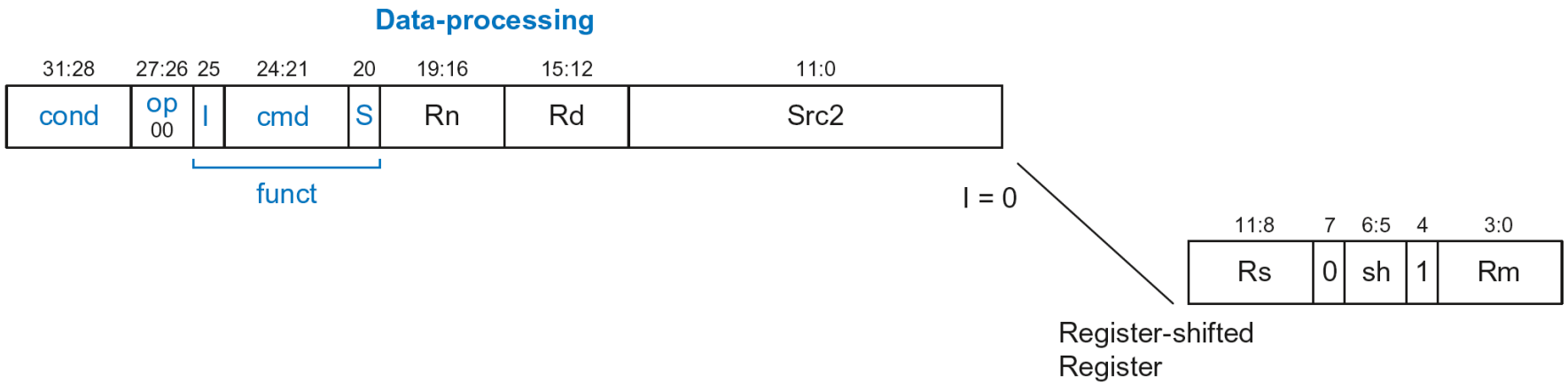


1110 00 0 1100 0 0101 1001 00010 01 0 0011

0xE1859123

DP with Register-shifted Reg. *Src2*

- *Src2* can be:
  - Immediate
  - Register
  - **Register-shifted register**



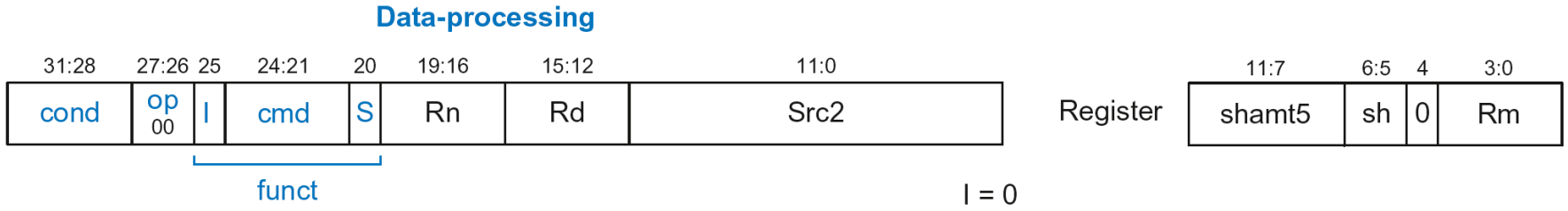
Shift Instructions Encoding

Shift Type	*sh*
LSL	00_2
LSR	01_2
ASR	10_2
ROR	11_2

Shift Instructions: Immediate shamt

```
ROR R1, R2, #23
```

- **Operation:** R1 = R2 ROR 23
- ***cond*** = 1110_2 (14) for unconditional execution
- ***op*** = 00_2 (0) for data-processing instructions
- ***cmd*** = 1101_2 (13) for all shifts (LSL, LSR, ASR, and ROR)
- ***Src2*** is an immediate-shifted register so ***I***=0
- ***Rd*** = 1, ***Rn*** = 0, ***Rm*** = 2
- ***shamt5*** = 23, ***sh*** = 11_2 (ROR)



1110 00 0 1101 0 0000 0001 10111 11 0 0010

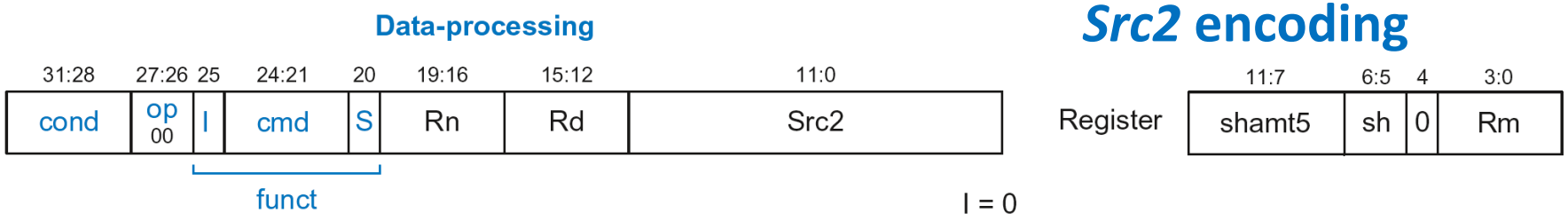
0xE1A01BE2

Shift Instructions: Immediate shamt

```
ROR R1, R2, #23
```

- **Operation:** R1 = R2 ROR 23
- ***cond*** = 1110_2 (14) for unconditional execution
- ***op*** = 00_2 (0) for data-processing instructions
- ***cmd*** = 1101_2 (13) for all shifts (LSL, LSR, ASR, and ROR)
- ***Src2*** is an immediate-shifted register so ***I***=0
- ***Rd*** = 1, ***Rn*** = 0, ***Rm*** = 2
- ***shamt5*** = 23, ***sh*** = 11_2 (ROR)

Uses (immediate-shifted) register *Src2* encoding



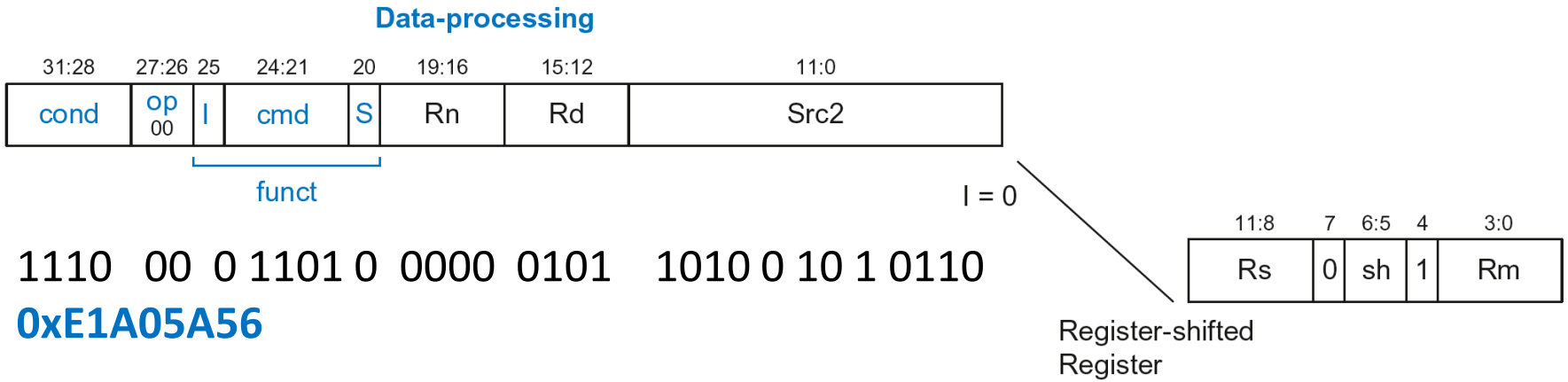
1110 00 0 1101 0 0000 0001 10111 11 0 0010

0xE1A01BE2

Shift Instructions: Register shamt

```
ASR R5, R6, R10
```

- **Operation:** R5 = R6 >>> $R10_{7:0}$
- ***cond*** = 1110_2 (14) for unconditional execution
- ***op*** = 00_2 (0) for data-processing instructions
- ***cmd*** = 1101_2 (13) for all shifts (LSL, LSR, ASR, and ROR)
- ***Src2*** is a register so ***I***=0
- ***Rd*** = 5, ***Rn*** = 0, ***Rm*** = 6, ***Rs*** = 10
- ***sh*** = 10_2 (ASR)

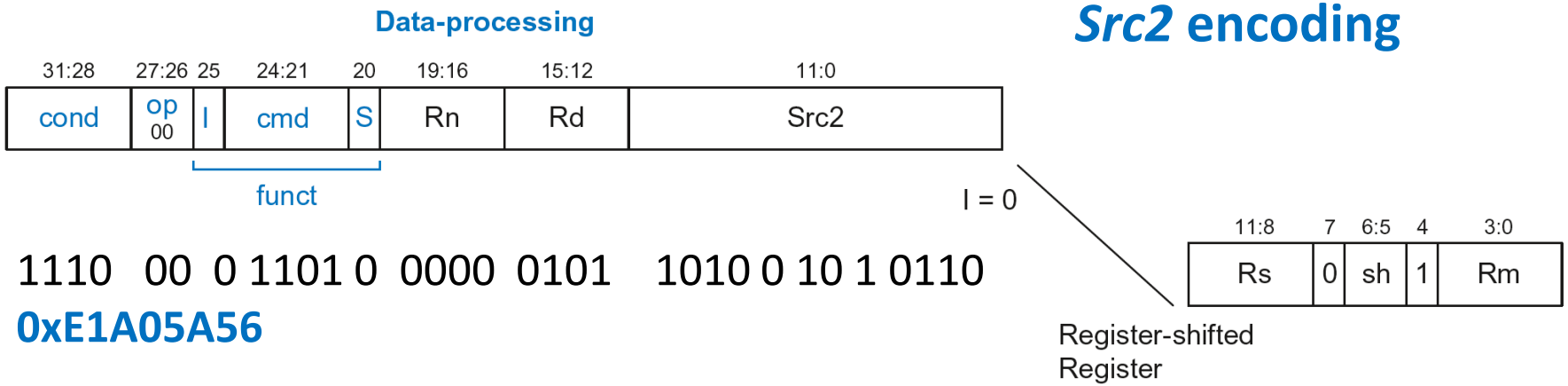


Shift Instructions: Register shamt

```
ASR R5, R6, R10
```

- **Operation:** R5 = R6 >>> $R10_{7:0}$
- ***cond*** = 1110_2 (14) for unconditional execution
- ***op*** = 00_2 (0) for data-processing instructions
- ***cmd*** = 1101_2 (13) for all shifts (LSL, LSR, ASR, and ROR)
- ***Src2*** is a register so ***I***=0
- ***Rd*** = 5, ***Rn*** = 0, ***Rm*** = 6, ***Rs*** = 10
- ***sh*** = 10_2 (ASR)

Uses register-shifted register *Src2* encoding

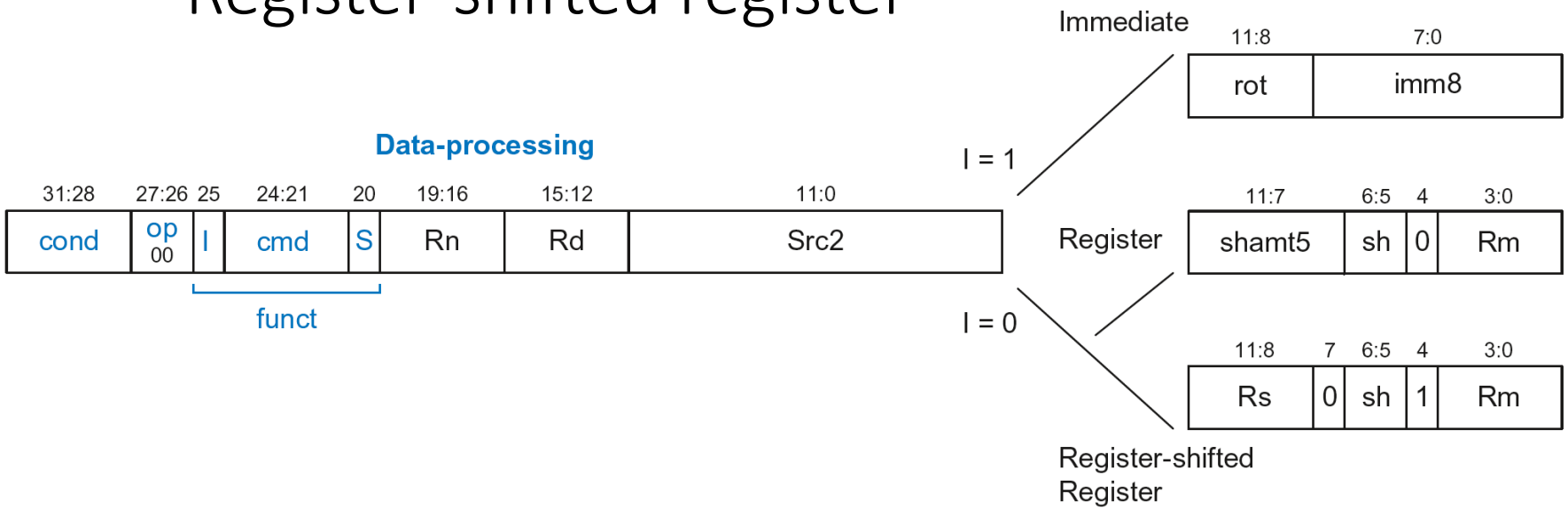


1110 00 0 1101 0 0000 0101 1010 0 10 1 0110

0xE1A05A56

Review: Data-processing Format

- *Src2* can be:
  - Immediate
  - Register
  - Register-shifted register



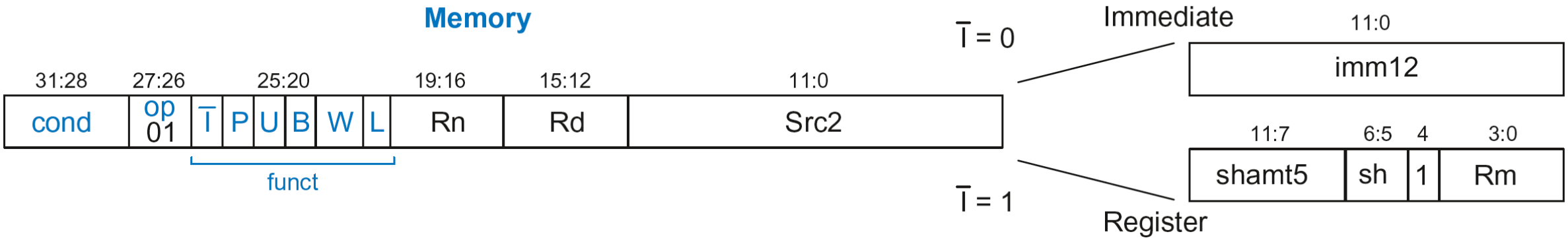
Instruction Formats

- Data-processing
- **Memory**
- Branch

Memory Instruction Format

Encodes: `LDR`, `STR`, `LDRB`, `STRB`

- *op* = 01_2
- *Rn* = base register
- *Rd* = destination (load), source (store)
- *Src2* = offset
- *funct* = 6 control bits



Offset Options

Recall: Address = Base Address + Offset

Example: `LDR R1, [R2, #4]`

Base Address = R2, Offset = 4

Address = (R2 + 4)

- Base address always in a register
- The offset can be:
  - an immediate
  - a register
  - or a scaled (shifted) register

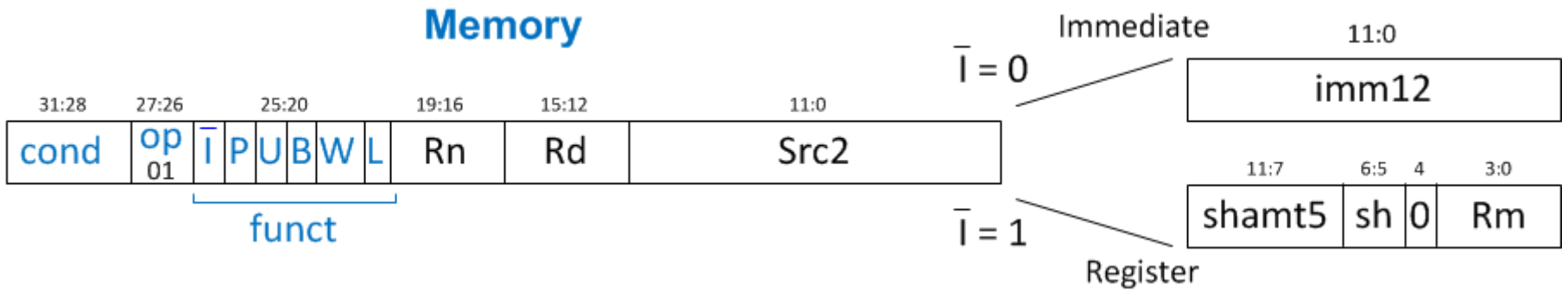
Offset Examples

ARM Assembly	Memory Address
`LDR R0, [R3, #4]`	R3 + 4
`LDR R0, [R5, #-16]`	R5 – 16
`LDR R1, [R6, R7]`	R6 + R7
`LDR R2, [R8, -R9]`	R8 – R9
`LDR R3, [R10, R11, LSL #2]`	R10 + (R11 << 2)
`LDR R4, [R1, -R12, ASR #4]`	R1 – (R12 >>> 4)
`LDR R0, [R9]`	R9

Memory Instruction Format

Encodes: `LDR`, `STR`, `LDRB`, `STRB`

- *op* = 01_2
- *Rn* = base register
- *Rd* = destination (load), source (store)
- *Src2* = offset: register (optionally shifted) or immediate
- *funct* = 6 control bits



Indexing Modes

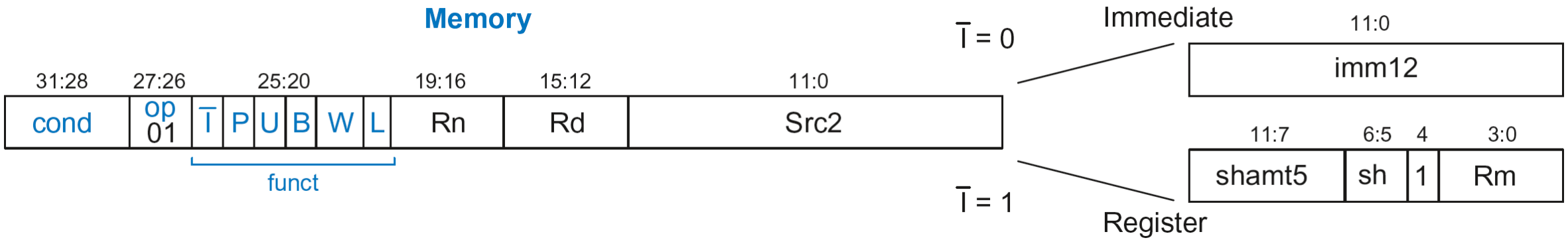
Mode	Address	Base Reg. Update
Offset	Base register ± Offset	No change
Preindex	Base register ± Offset	Base register ± Offset
Postindex	Base register	Base register ± Offset

Examples

- Offset:
```
LDR R1, [R2, #4]     ; R1 = mem[R2+4]
```
- Preindex:
```
LDR R3, [R5, #16]!   ; R3 = mem[R5+16]
                     ; R5 = R5 + 16
```
- Postindex:
```
LDR R8, [R1], #8     ; R8 = mem[R1]
                     ; R1 = R1 + 8
```

Memory Instruction Format

- *funct*:
  - $\overline{I}$: Immediate bar
  - *P*: Preindex
  - *U*: Add
  - *B*: Byte
  - *W*: Writeback
  - *L*: Load



Memory Format *funct* Encodings

Type of Operation

L	*B*	Instruction
0	0	`STR`
0	1	`STRB`
1	0	`LDR`
1	1	`LDRB`

Memory Format *funct* Encodings

Type of Operation

L	*B*	Instruction
0	0	`STR`
0	1	`STRB`
1	0	`LDR`
1	1	`LDRB`

Indexing Mode

P	*W*	Indexing Mode
0	1	Not supported
0	0	Postindex
1	0	Offset
1	1	Preindex

Memory Format *funct* Encodings

Type of Operation

L	*B*	Instruction
0	0	`STR`
0	1	`STRB`
1	0	`LDR`
1	1	`LDRB`

Indexing Mode

P	*W*	Indexing Mode
0	1	Not supported
0	0	Postindex
1	0	Offset
1	1	Preindex

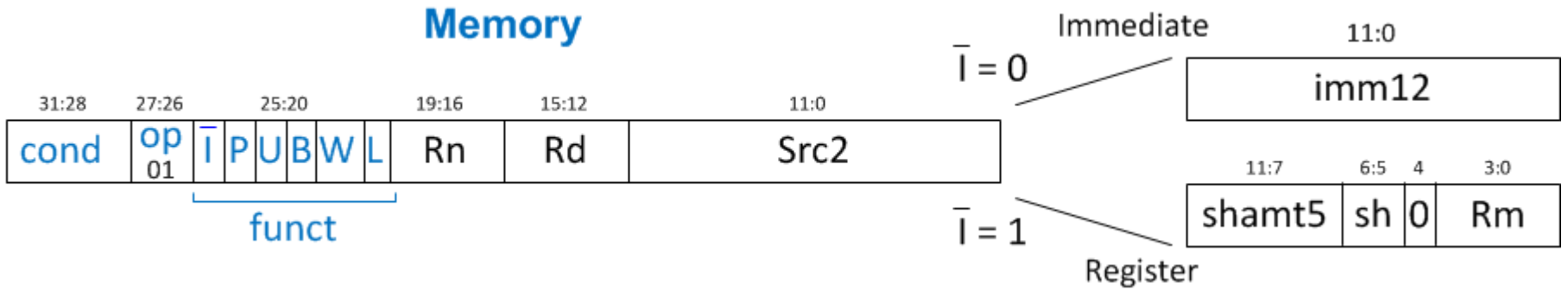
Add/Subtract Immediate/Register Offset

Value	$\overline{I}$	*U*
0	**Immediate** offset in *Src2*	**Subtract** offset from base
1	**Register** offset in *Src2*	**Add** offset to base

Memory Instruction Format

Encodes: `LDR`, `STR`, `LDRB`, `STRB`

- *op* = 01_2
- *Rn* = base register
- *Rd* = destination (load), source (store)
- *Src2* = offset: immediate or register (optionally shifted)
- *funct* = $\bar{I}$ (immediate bar), *P* (preindex), *U* (add), *B* (byte), *W* (writeback), *L* (load)



Memory Instr. with Immediate *Src2*

```
STR R11, [R5], #-26
```

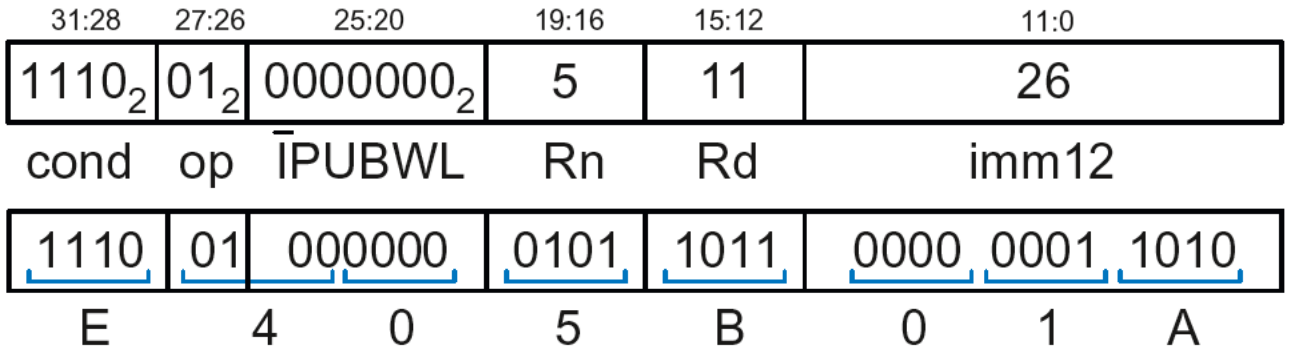
- **Operation:** mem[R5] <= R11; R5 = R5 - 26
- ***cond*** = 1110_2 (14) for unconditional execution
- ***op*** = 01_2 (1) for memory instruction
- ***funct*** = 0000000_2 **(0)**
  $\overline{I}$ = 0 (immediate offset), ***P*** = 0 (postindex),
  U = 0 (subtract), ***B*** = 0 (store word), ***W*** = 0 (postindex),
  L = 0 (store)
- ***Rd*** = 11, ***Rn*** = 5, ***imm12*** = 26

Memory Instr. with Immediate *Src2*

```
STR R11, [R5], #-26
```

- **Operation:** mem[R5] <= R11; R5 = R5 - 26
- ***cond*** = 1110_2 (14) for unconditional execution
- ***op*** = 01_2 (1) for memory instruction
- ***funct*** = 0000000_2 **(0)**
  $\overline{I}$ = 0 (immediate offset), ***P*** = 0 (postindex),
  U = 0 (subtract), ***B*** = 0 (store word), ***W*** = 0 (postindex),
  L = 0 (store)
- ***Rd*** = 11, ***Rn*** = 5, ***imm12*** = 26

Field Values

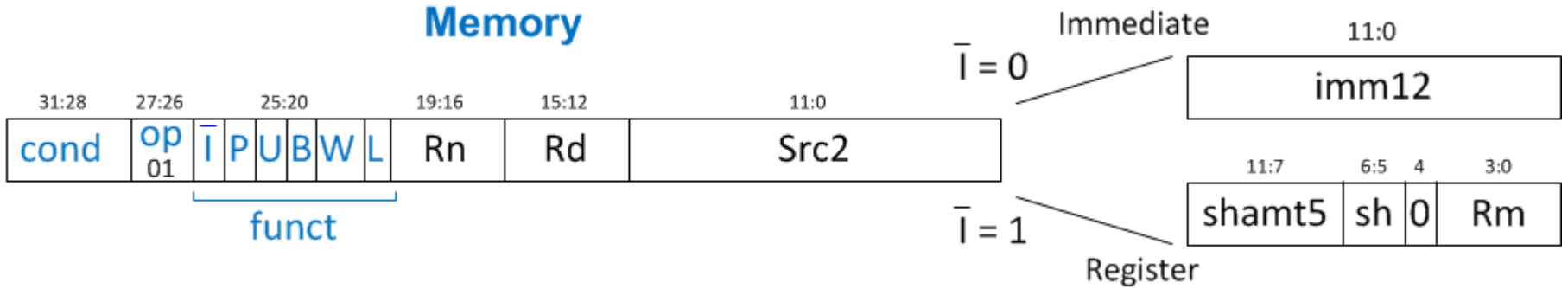


Memory Instr. with Register *Src2*

```
LDR R3, [R4, R5]
```

- **Operation:** R3 <= mem[R4 + R5]
- ***cond*** = 1110_2 (14) for unconditional execution
- ***op*** = 01_2 (1) for memory instruction
- ***funct*** = 111001_2 **(57)**
  $\overline{I}$ = 1 (register offset), ***P*** = 1 (offset indexing),
  U = 1 (add), ***B*** = 0 (load **word**), ***W*** = 0 (offset indexing),
  L = 1 (load)
- ***Rd*** = 3, ***Rn*** = 4, ***Rm*** = 5 (***shamt5*** = 0, ***sh*** = 0)

1110 01 111001 0100 0011 00000 00 0 0101 = **0xE7943005**

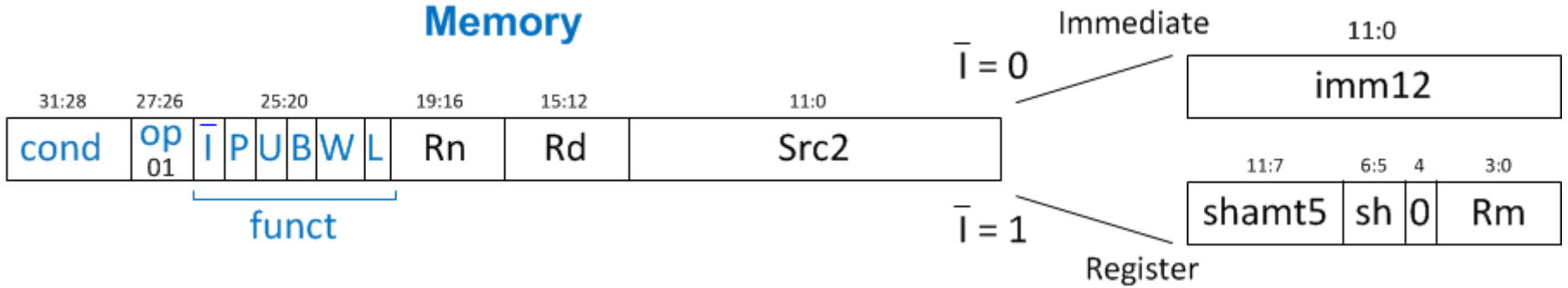


Memory Instr. with Scaled Reg. *Src2*

```
STR R9, [R1, R3, LSL #2]
```

- **Operation:** mem[R1 + (R3 << 2)] <= R9
- ***cond*** = 1110_2 (14) for unconditional execution
- ***op*** = 01_2 (1) for memory instruction
- ***funct*** = 111000_2 **(0)**
  $\overline{I}$ = 1 (register offset), ***P*** = 1 (offset indexing),
  U = 1 (add), ***B*** = 0 (store **word**), ***W*** = 0 (offset indexing),
  L = 0 (store)
- ***Rd*** = 9, ***Rn*** = 1, ***Rm*** = 3, ***shamt*** **= 2**, ***sh*** = 00_2 (LSL)

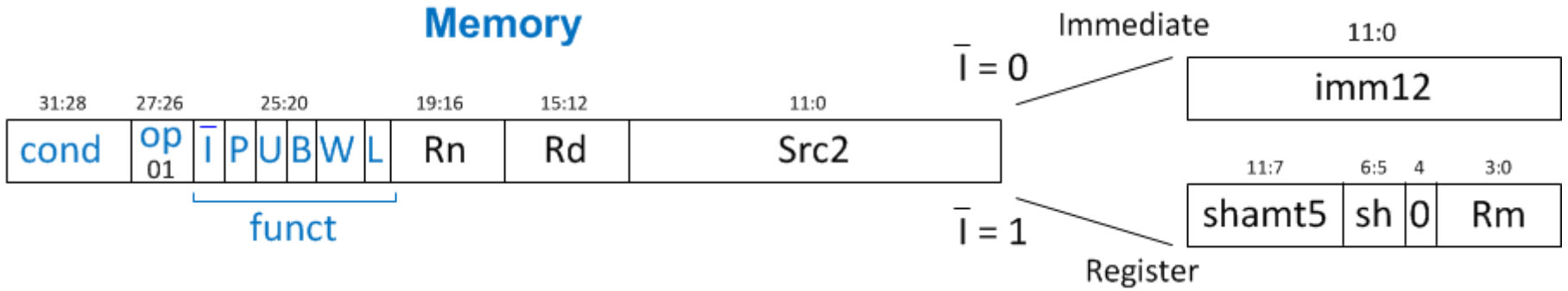
1110 01 111000 0001 1001 00010 00 0 0011 = **0xE7819103**



Review: Memory Instruction Format

Encodes: `LDR`, `STR`, `LDRB`, `STRB`

- *op* = 01_2
- *Rn* = base register
- *Rd* = destination (load), source (store)
- *Src2* = offset: register (optionally shifted) or immediate
- *funct* = $\bar{I}$ (immediate bar), *P* (preindex), *U* (add), *B* (byte), *W* (writeback), *L* (load)



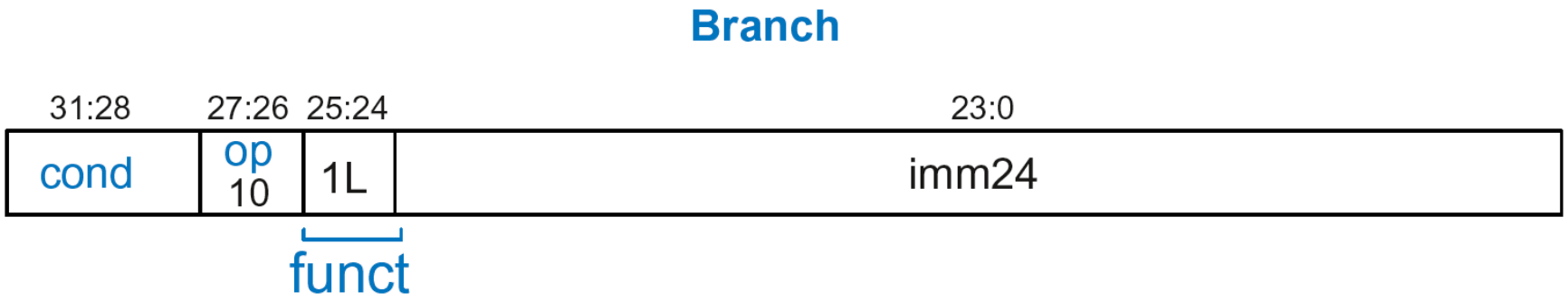
Instruction Formats

- Data-processing
- Memory
- **Branch**

Branch Instruction Format

Encodes `B` and `BL`

- ***op*** = 10_2
- ***imm24*****:** 24-bit immediate
- ***funct*** = $1L_2$: ***L*** = 1 for `BL`, ***L*** = 0 for `B`



Encoding Branch Target Address

- ***Branch Target Address*** **(BTA)*:*** Next PC when branch taken
- BTA is relative to current PC + 8
- *imm24* encodes BTA
- ***imm24*** = # of words BTA is away from PC+8

Branch Instruction: Example 1

ARM assembly code

```
0xA0          BLT THERE          ← PC
0xA4          ADD R0, R1, R2
0xA8          SUB R0, R0, R9     ← PC+8
0xAC          ADD SP, SP, #8
0xB0          MOV PC, LR
0xB4  THERE   SUB R0, R0, #1     ← BTA
0xB8          BL  TEST
```

- PC = 0xA0
- PC + 8 = 0xA8
- THERE label is 3 instructions past PC+8
- So, *imm24* = 3

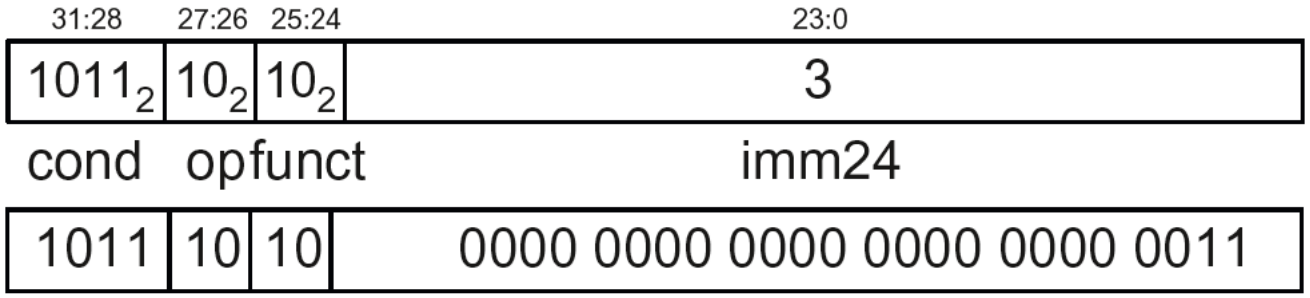
Branch Instruction: Example 1

ARM assembly code

```
0xA0          BLT THERE          ← PC
0xA4          ADD R0, R1, R2
0xA8          SUB R0, R0, R9     ← PC+8
0xAC          ADD SP, SP, #8
0xB0          MOV PC, LR
0xB4  THERE   SUB R0, R0, #1     ← BTA
0xB8          BL  TEST
```

- PC = 0xA0
- PC + 8 = 0xA8
- THERE label is 3 instructions past PC+8
- So, *imm24* = 3

Field Values



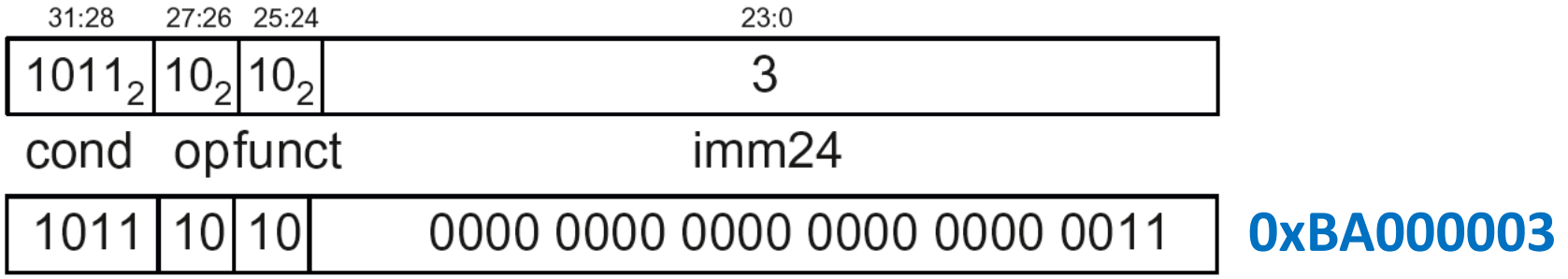
Branch Instruction: Example 1

ARM assembly code

```
0xA0          BLT THERE          ← PC
0xA4          ADD R0, R1, R2
0xA8          SUB R0, R0, R9     ← PC+8
0xAC          ADD SP, SP, #8
0xB0          MOV PC, LR
0xB4  THERE   SUB R0, R0, #1     ← BTA
0xB8          BL  TEST
```

- PC = 0xA0
- PC + 8 = 0xA8
- THERE label is 3 instructions past PC+8
- So, *imm24* = 3

Field Values



0xBA000003

Branch Instruction: Example 2

ARM assembly code

```
0x8040 TEST   LDRB R5, [R0, R3]   ← BTA
0x8044        STRB R5, [R1, R3]
0x8048        ADD  R3, R3, #1
0x8044        MOV  PC, LR
0x8050        BL   TEST           ← PC
0x8054        LDR  R3, [R1], #4
0x8058        SUB  R4, R3, #9     ← PC+8
```

- PC = 0x8050
- PC + 8 = 0x8058
- `TEST` label is 6 instructions before PC+8
- So, *imm24* = -6

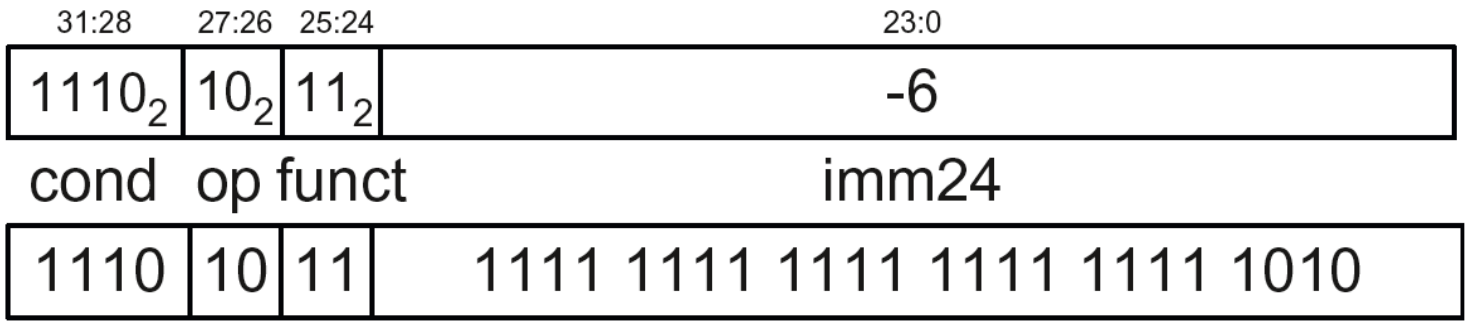
Branch Instruction: Example 2

ARM assembly code

```
0x8040 TEST   LDRB R5, [R0, R3]   ← BTA
0x8044        STRB R5, [R1, R3]
0x8048        ADD  R3, R3, #1
0x8044        MOV  PC, LR
0x8050        BL   TEST           ← PC
0x8054        LDR  R3, [R1], #4
0x8058        SUB  R4, R3, #9     ← PC+8
```

- PC = 0x8050
- PC + 8 = 0x8058
- `TEST` label is 6 instructions before PC+8
- So, *imm24* = -6

Field Values



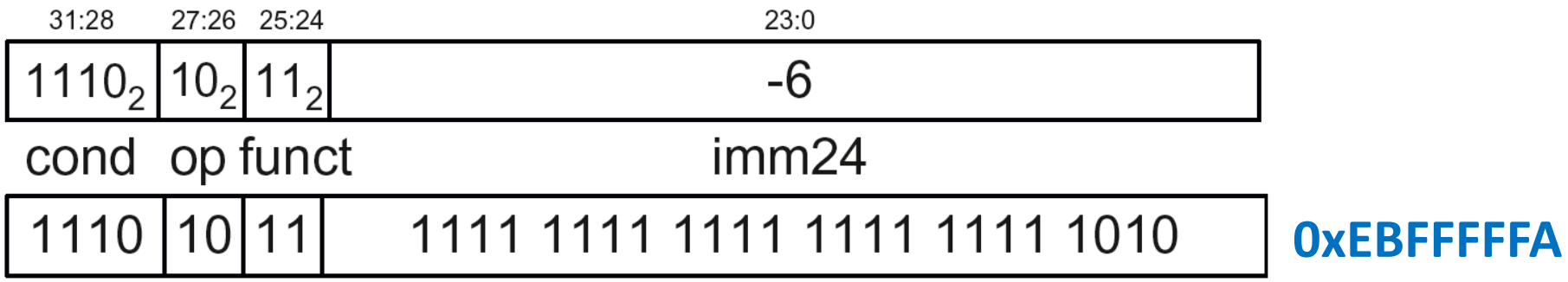
Branch Instruction: Example 2

ARM assembly code

```
0x8040 TEST   LDRB R5, [R0, R3]   ← BTA
0x8044        STRB R5, [R1, R3]
0x8048        ADD  R3, R3, #1
0x8044        MOV  PC, LR
0x8050        BL   TEST           ← PC
0x8054        LDR  R3, [R1], #4
0x8058        SUB  R4, R3, #9     ← PC+8
```

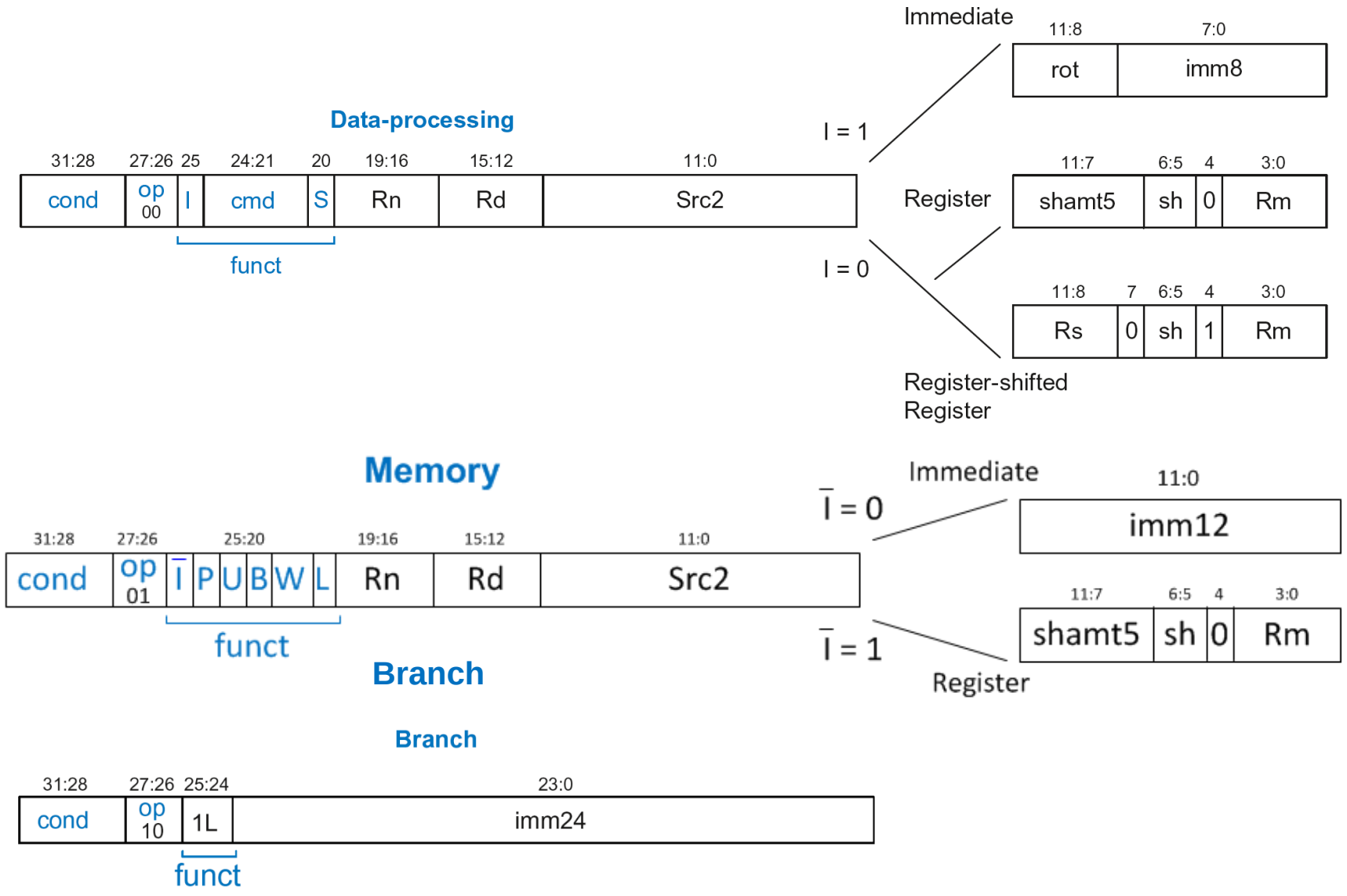
- PC = 0x8050
- PC + 8 = 0x8058
- `TEST` label is 6 instructions before PC+8
- So, *imm24* = -6

Field Values



0xEBFFFFFA

Review: Instruction Formats



Conditional Execution

Encode in *cond* bits of machine instruction

For example,

```
ANDEQ R1, R2, R3      (cond = 0000)
ORRMI R4, R5, #0xF    (cond = 0100)
SUBLT R9, R3, R8      (cond = 1011)
```

Review: Condition Mnemonics

cond	Mnemonic	Name	CondEx
0000	EQ	Equal	Z
0001	NE	Not equal	$\bar{Z}$
0010	CS / HS	Carry set / Unsigned higher or same	C
0011	CC / LO	Carry clear / Unsigned lower	$\bar{C}$
0100	MI	Minus / Negative	N
0101	PL	Plus / Positive of zero	$\bar{N}$
0110	VS	Overflow / Overflow set	V
0111	VC	No overflow / Overflow clear	$\bar{V}$
1000	HI	Unsigned higher	$\bar{Z}C$
1001	LS	Unsigned lower or same	$Z\ OR\ \bar{C}$
1010	GE	Signed greater than or equal	$\overline{N \oplus V}$
1011	LT	Signed less than	$N \oplus V$
1100	GT	Signed greater than	$\bar{Z}(\overline{N \oplus V})$
1101	LE	Signed less than or equal	$Z\ OR\ (N \oplus V)$
1110	AL (or none)	Always / unconditional	ignored

Conditional Execution: Machine Code

Assembly Code

Field Values

Assembly Code	31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0
	cond	op	I	cmd	S	rn	rd	shamt5	sh		rm
SUBS R1, R2, R3	14	0	0	2	1	2	1	0	0	0	3
ADDEQ R4, R5, R6	0	0	0	4	0	5	4	0	0	0	6
ANDHS R7, R5, R6	2	0	0	0	0	5	7	0	0	0	6
ORRMI R8, R5, R6	4	0	0	12	0	5	8	0	0	0	6
EORLT R9, R5, R6	11	0	0	1	0	5	9	0	0	0	6

Machine Code

31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0	
cond	op	I	cmd	S	rn	rd	shamt5	sh		rm	
1110	00	0	0010	0	0010	0001	00000	00	0	0011	(0xE0421003)
0000	00	0	0100	0	0101	0100	00000	00	0	0110	(0x00854006)
0010	00	0	0000	0	0101	0111	00000	00	0	0110	(0x20057006)
0100	00	0	1100	0	0101	1000	00000	00	0	0110	(0x41858006)
1011	00	0	0001	0	0101	1001	00000	00	0	0110	(0xB0259006)

Interpreting Machine Code

- **Start with *op*:** tells how to parse rest

  op = 00 (Data-processing)

  op = 01 (Memory)

  op = 10 (Branch)

- ***I*-bit:** tells how to parse ***Src2***
- **Data-processing instructions:**

  If *I*-bit is 0, bit 4 determines if ***Src2*** is a register (bit 4 = 0) or a register-shifted register (bit 4 = 1)

- **Memory instructions:**

  Examine ***funct*** bits for indexing mode, instruction, and add or subtract offset

Interpreting Machine Code: Example 1

0xE0475001

Interpreting Machine Code: Example 1

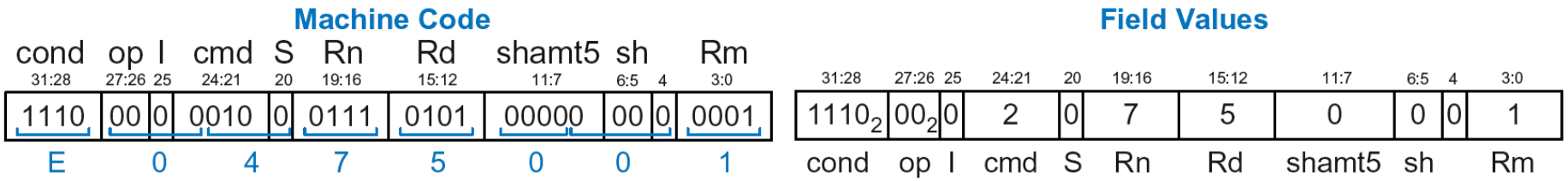
0xE0475001

- **Start with *op*:** 00_2, so data-processing instruction

Interpreting Machine Code: Example 1

0xE0475001

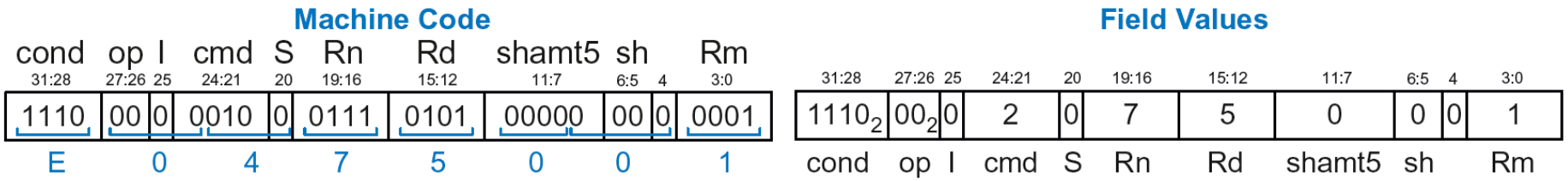
- **Start with *op*:** 00_2, so data-processing instruction
- ***I*-bit:** 0, so *Src2* is a register
- **bit 4:** 0, so *Src2* is a register (optionally shifted by *shamt5*)



Interpreting Machine Code: Example 1

0xE0475001

- **Start with *op*:** 00_2, so data-processing instruction
- ***I*-bit:** 0, so *Src2* is a register
- **bit 4:** 0, so *Src2* is a register (optionally shifted by *shamt5*)
- ***cmd*:** 0010_2 (2), so SUB
- **Rn**=7, **Rd**=5, **Rm**=1, ***shamt5*** = 0, ***sh*** = 0
- So, instruction is: **`SUB R5,R7,R1`**



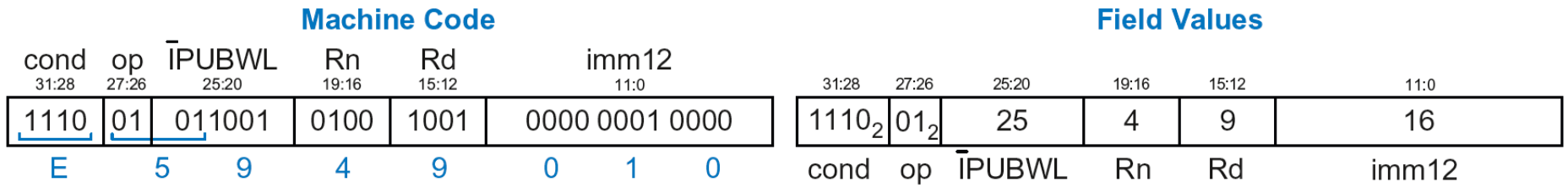
Interpreting Machine Code: Example 2

0xE5949010

Interpreting Machine Code: Example 2

0xE5949010

- **Start with *op*:** 01_2, so memory instruction
- ***funct*:** *B*=0, *L*=1, so `LDR`; *P*=1, *W*=0, so offset indexing; *I*=0, so immediate offset, *U*=1, so add offset
- **Rn**=4, **Rd**=9, ***imm12*** = 16
- So, instruction is: **`LDR R9,[R4,#16]`**



Addressing Modes

How do we address operands?

- Register
- Immediate
- Base
- PC-Relative

Addressing Modes

How do we address operands?

- **Register Only**
- Immediate
- Base
- PC-Relative

Register Addressing

- Source and destination operands found in registers
- Used by data-processing instructions
- **Three submodes:**
  - Register-only
  - Immediate-shifted register
  - Register-shifted register

Register Addressing Examples

- **Register-only**

  Example: `ADD R0, R2, R7`

- **Immediate-shifted register**

  Example: `ORR R5, R1, R3, LSL #1`

- **Register-shifted register**

  Example: `SUB R12, R9, R0, ASR R1`

Addressing Modes

How do we address operands?

- Register Only
- **Immediate**
- Base
- PC-Relative

Immediate Addressing

- Operands found in registers **and** immediates

  Example: `ADD R9, R1, #14`

- Uses data-processing format with *I*=1
  - Immediate is encoded as
    - 8-bit immediate (*imm8*)
    - 4-bit rotation (*rot*)
  - 32-bit immediate = *imm8* ROR (*rot* x 2)

Addressing Modes

How do we address operands?

- Register Only
- Immediate
- **Base**
- PC-Relative

Base Addressing

- Address of operand is:

  base register + offset

- Offset can be a:
  - 12-bit Immediate
  - Register
  - Immediate-shifted Register

Base Addressing Examples

- **Immediate offset**

  Example: `LDR R0, [R8, #-11]`

  (R0 = mem[R8 - 11])

- **Register offset**

  Example: `LDR R1, [R7, R9]`

  (R1 = mem[R7 + R9])

- **Immediate-shifted register offset**

  Example: `STR R5, [R3, R2, LSL #4]`

  (R5 = mem[R3 + (R2 << 4)])

Addressing Modes

How do we address operands?

- Register Only
- Immediate
- Base
- **PC-Relative**

PC-Relative Addressing

- Used for branches
- Branch instruction format:
  - Operands are PC and a signed 24-bit immediate (*imm24*)
  - Changes the PC
  - New PC is relative to the old PC
  - *imm24* indicates the number of words away from PC+8
- PC = (PC+8) + (SignExtended(*imm24*) x 4)

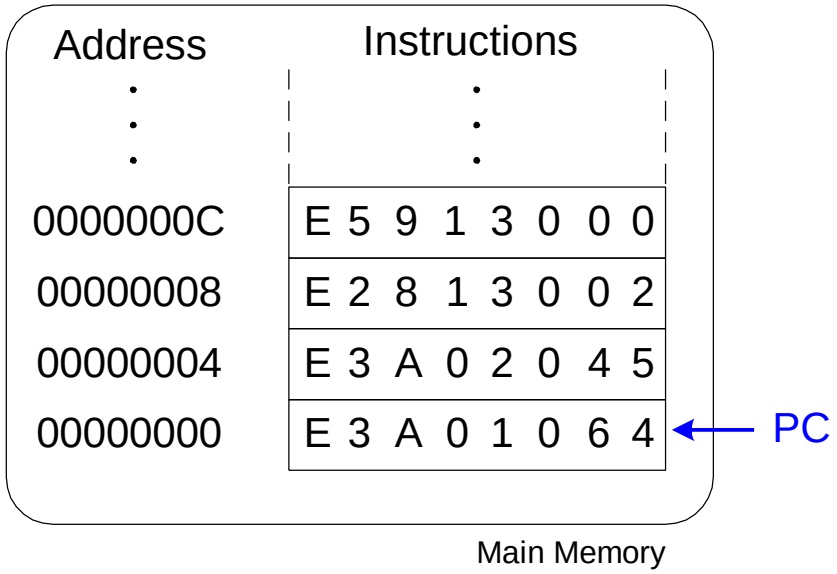
Power of the Stored Program

- **32-bit instructions & data** stored in memory
- **Sequence of instructions:** only difference between two applications
- **To run a new program:**
  - No rewiring required
  - Simply store new program in memory
- **Program Execution:**
  - Processor *fetches* (reads) instructions from memory in sequence
  - Processor performs the specified operation

The Stored Program

Assembly Code	Machine Code
`MOV R1, #100`	`0xE3A01064`
`MOV R2, #69`	`0xE3A02045`
`ADD R3, R1, R2`	`0xE2813002`
`STR R3, [R1]`	`0xE5913000`

Stored Program



Program Counter (PC): keeps track of current instruction