Computer organization and architecture

Lesson 10

Machine language

Assembly language is convenient for humans to read.

Digital circuits understand only 1’s and 0’s.

A program written in assembly is translated to machine language.

ARM uses 32-bit instructions.

ARM defines three main instruction formats:

– Data-processing instructions

– Memory instructions

– Branch instructions

Data-processing instructions

Data-processing instructions have

– a destination register

– a 1st source register

– a 2nd source that is either an immediate or a register, possibly shifted

6 fields: cond, op, funct, Rn, Rd, Src2

31:28	27:26	25:20	19:16	15:12	11:0
cond	op	funct	Rn	Rd	Src2
4 bits	2 bits	6 bits	4 bits	4 bits	12 bits

op – the opcode (operation code)

op is 00 for data-processing instructions

31:28	27:26	25:20	19:16	15:12	11:0
cond	op	funct	Rn	Rd	Src2
4 bits	2 bits	6 bits	4 bits	4 bits	12 bits

cond – conditional execution based on flags

cond = 1110 for unconditional instructions

funct – function code

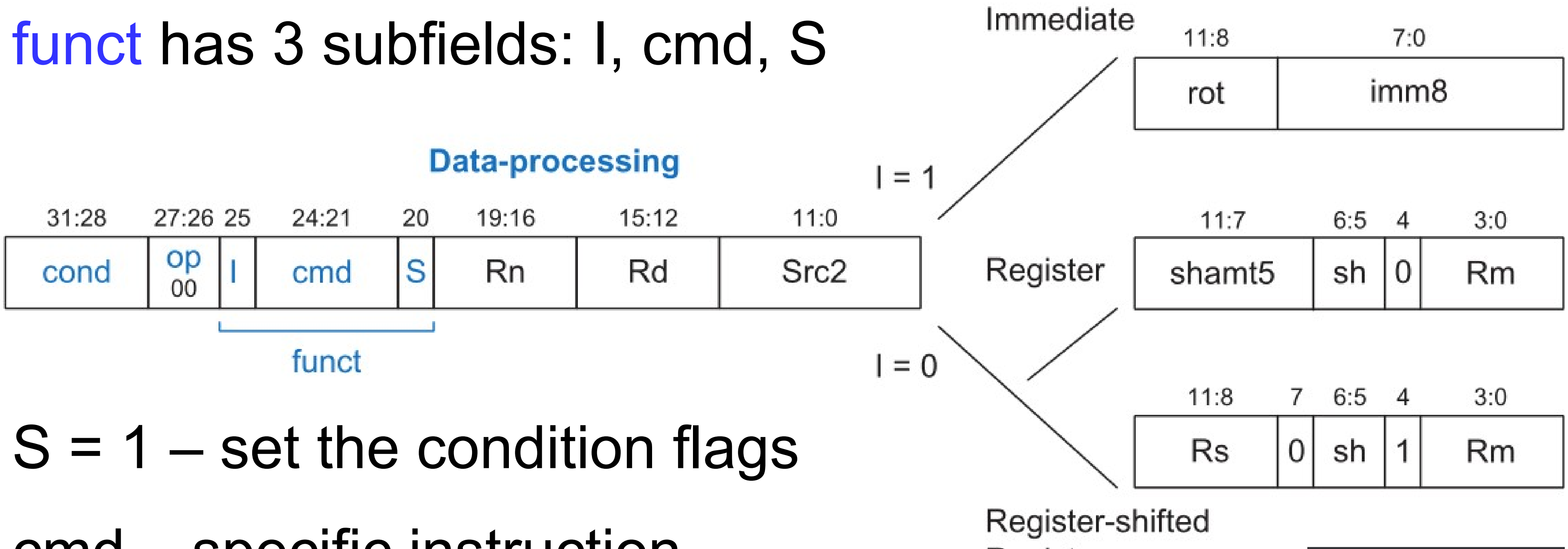
Rn – the 1st source register

Src2 – the 2nd source

Rd – the destination register

cond	Mnemonic
0000	EQ
0001	NE
0010	CS/HS
0011	CC/LO
0100	MI
0101	PL
0110	VS
0111	VC
1000	HI
1001	LS
1010	GE
1011	LT
1100	GT
1101	LE
1110	AL (or none)

funct has 3 subfields: I, cmd, S



S = 1 – set the condition flags

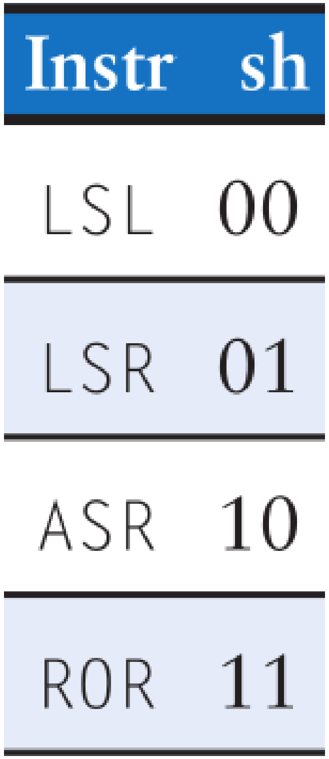
cmd – specific instruction

Src2 can be

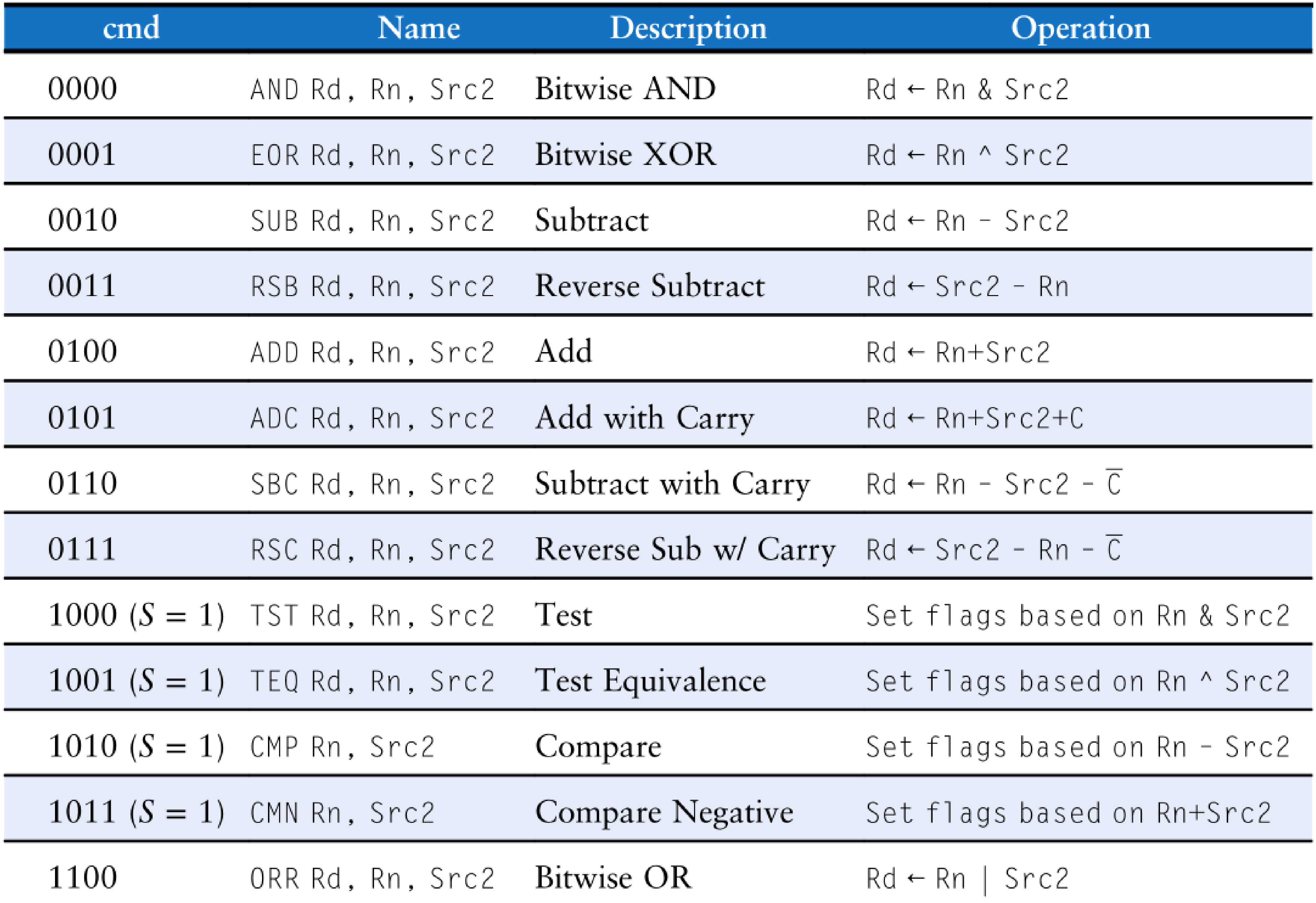
1) an immediate

2) a register Rm optionally shifted by a constant shamt5

3) a register Rm shifted by another register Rs



Instr	sh
LSL	00
LSR	01
ASR	10
ROR	11



cmd	Name	Description	Operation
0000	AND Rd, Rn, Src2	Bitwise AND	Rd ← Rn & Src2
0001	EOR Rd, Rn, Src2	Bitwise XOR	Rd ← Rn ^ Src2
0010	SUB Rd, Rn, Src2	Subtract	Rd ← Rn - Src2
0011	RSB Rd, Rn, Src2	Reverse Subtract	Rd ← Src2 - Rn
0100	ADD Rd, Rn, Src2	Add	Rd ← Rn+Src2
0101	ADC Rd, Rn, Src2	Add with Carry	Rd ← Rn+Src2+C
0110	SBC Rd, Rn, Src2	Subtract with Carry	Rd ← Rn - Src2 - $\overline{C}$
0111	RSC Rd, Rn, Src2	Reverse Sub w/ Carry	Rd ← Src2 - Rn - $\overline{C}$
1000 (S = 1)	TST Rd, Rn, Src2	Test	Set flags based on Rn & Src2
1001 (S = 1)	TEQ Rd, Rn, Src2	Test Equivalence	Set flags based on Rn ^ Src2
1010 (S = 1)	CMP Rn, Src2	Compare	Set flags based on Rn - Src2
1011 (S = 1)	CMN Rn, Src2	Compare Negative	Set flags based on Rn+Src2
1100	ORR Rd, Rn, Src2	Bitwise OR	Rd ← Rn | Src2

cont. on the next page...

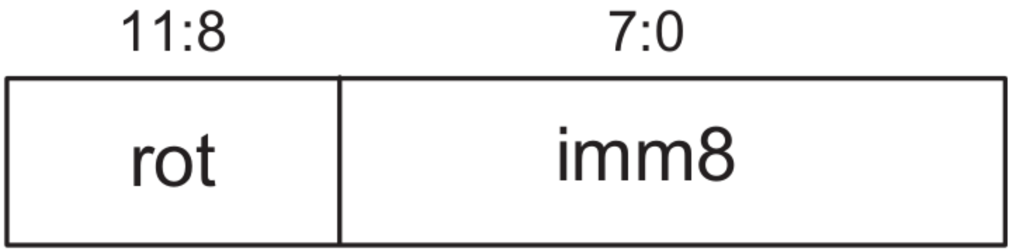
cmd	Name	Description	Operation
1101	Shifts:		
$I = 1$ OR ($\text{instr}_{11:4} = 0$)	MOV Rd, Src2	Move	Rd ← Src2
$I = 0$ AND ($sh = 00$; $\text{instr}_{11:4} \neq 0$)	LSL Rd, Rm, Rs/shamt5	Logical Shift Left	Rd ← Rm << Src2
$I = 0$ AND ($sh = 01$)	LSR Rd, Rm, Rs/shamt5	Logical Shift Right	Rd ← Rm >> Src2
$I = 0$ AND ($sh = 10$)	ASR Rd, Rm, Rs/shamt5	Arithmetic Shift Right	Rd ← Rm>>>Src2
$I = 0$ AND ($sh = 11$; $\text{instr}_{11:7,\ 4} = 0$)	RRX Rd, Rm, Rs/shamt5	Rotate Right Extend	{Rd, C} ← {C, Rd}
$I = 0$ AND ($sh = 11$; $\text{instr}_{11:7} \neq 0$)	ROR Rd, Rm, Rs/shamt5	Rotate Right	Rd ← Rn ror Src2
1110	BIC Rd, Rn, Src2	Bitwise Clear	Rd ← Rn & ~Src2
1111	MVN Rd, Rn, Src2	Bitwise NOT	Rd ← ~Rn

Multiply instructions: 3-bit cmd field

31:28	27:26	25:24	23:21	20	19:16	15:12	11:8	7:4	3:0
cond	op 00	00	cmd	S	Rd	Ra	Rm	1001	Rn
4 bits	2 bits	6 bits			4 bits	4 bits	4 bits	4 bits	4 bits

cmd	Name	Description	Operation
000	MUL Rd, Rn, Rm	Multiply	Rd ← Rn × Rm (low 32 bits)
001	MLA Rd, Rn, Rm, Ra	Multiply Accumulate	Rd ← (Rn × Rm)+Ra (low 32 bits)
100	UMULL Rd, Rn, Rm, Ra	Unsigned Multiply Long	{Rd, Ra} ← Rn × Rm (all 64 bits, Rm/Rn unsigned)
101	UMLAL Rd, Rn, Rm, Ra	Unsigned Multiply Accumulate Long	{Rd, Ra} ← (Rn × Rm)+{Rd, Ra} (all 64 bits, Rm/Rn unsigned)
110	SMULL Rd, Rn, Rm, Ra	Signed Multiply Long	{Rd, Ra} ← Rn × Rm (all 64 bits, Rm/Rn signed)
111	SMLAL Rd, Rn, Rm, Ra	Signed Multiply Accumulate Long	{Rd, Ra} ← (Rn × Rm)+{Rd, Ra} (all 64 bits, Rm/Rn signed)

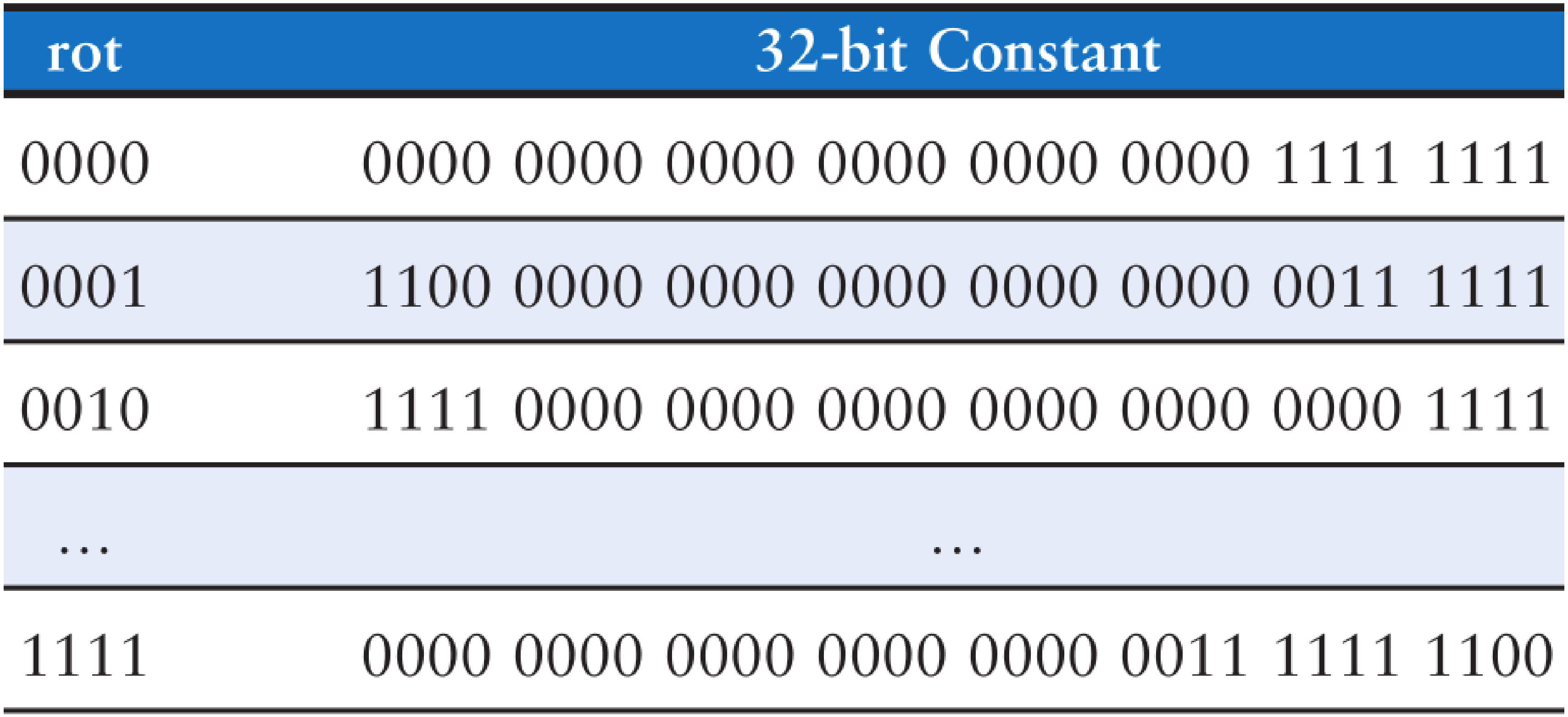
Representation of immediates



11:8	7:0
rot	imm8

imm8 is rotated right by 2 × rot to create a 32-bit constant

Example: Immediate rotations and resulting 32-bit constant for imm8 = 0xFF



rot	32-bit Constant
0000	0000 0000 0000 0000 0000 0000 1111 1111
0001	1100 0000 0000 0000 0000 0000 0011 1111
0010	1111 0000 0000 0000 0000 0000 0000 1111
. . .	. . .
1111	0000 0000 0000 0000 0000 0011 1111 1100

Memory instructions

Memory instructions have three operands:

- – a register that is the destination on an `LDR` and a source on an `STR`
- – a base register
- – an offset that is either an immediate or an optionally shifted register

The same six overall fields: cond, op, funct, Rn, Rd, Src2

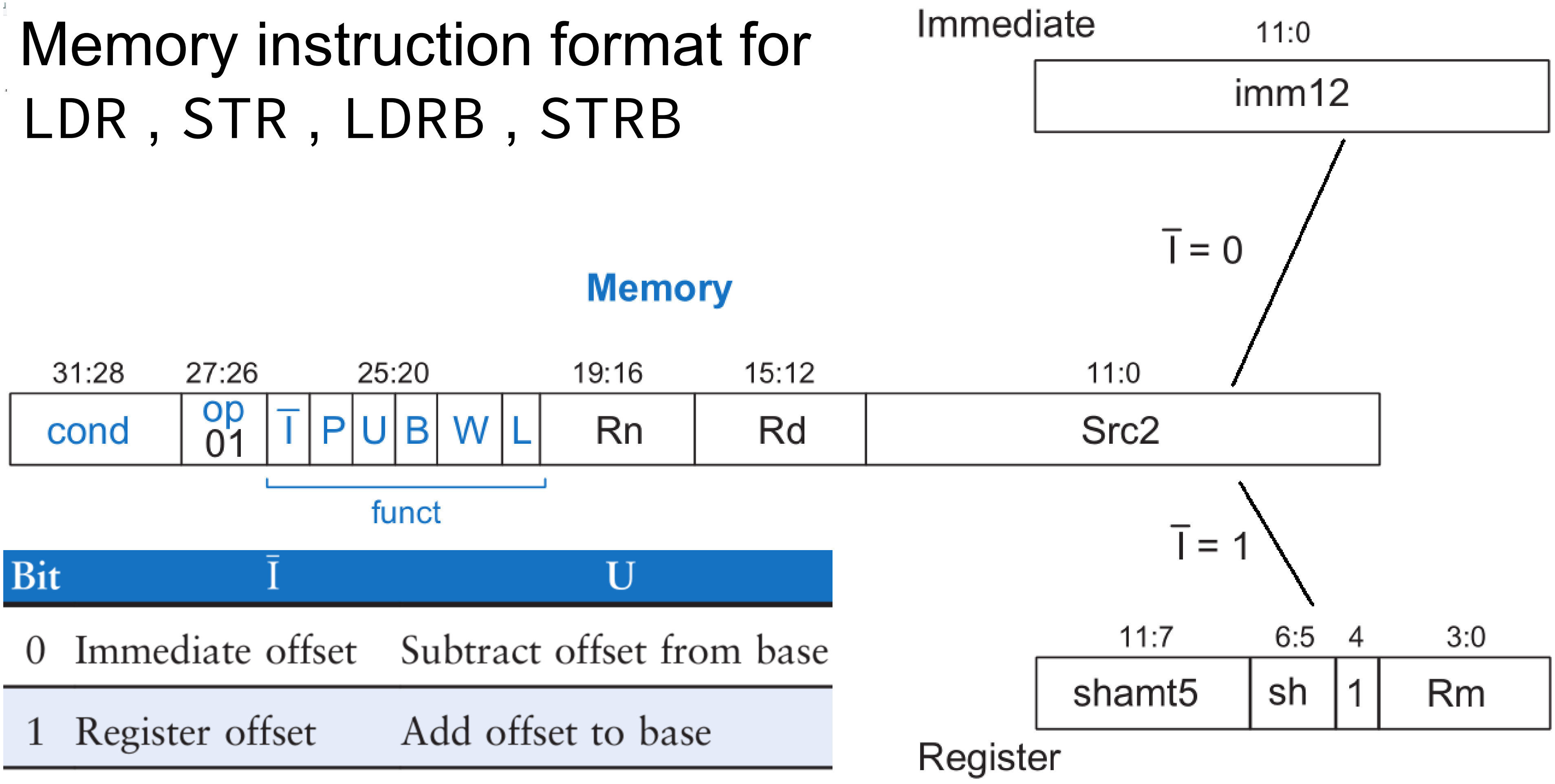
Rn – the base register

Src2 – the offset

Rd – the destination register in a load or
the source register in a store

op is 01 for memory instructions

Memory instruction format for LDR , STR , LDRB , STRB



Bit	Ī	U
0	Immediate offset	Subtract offset from base
1	Register offset	Add offset to base

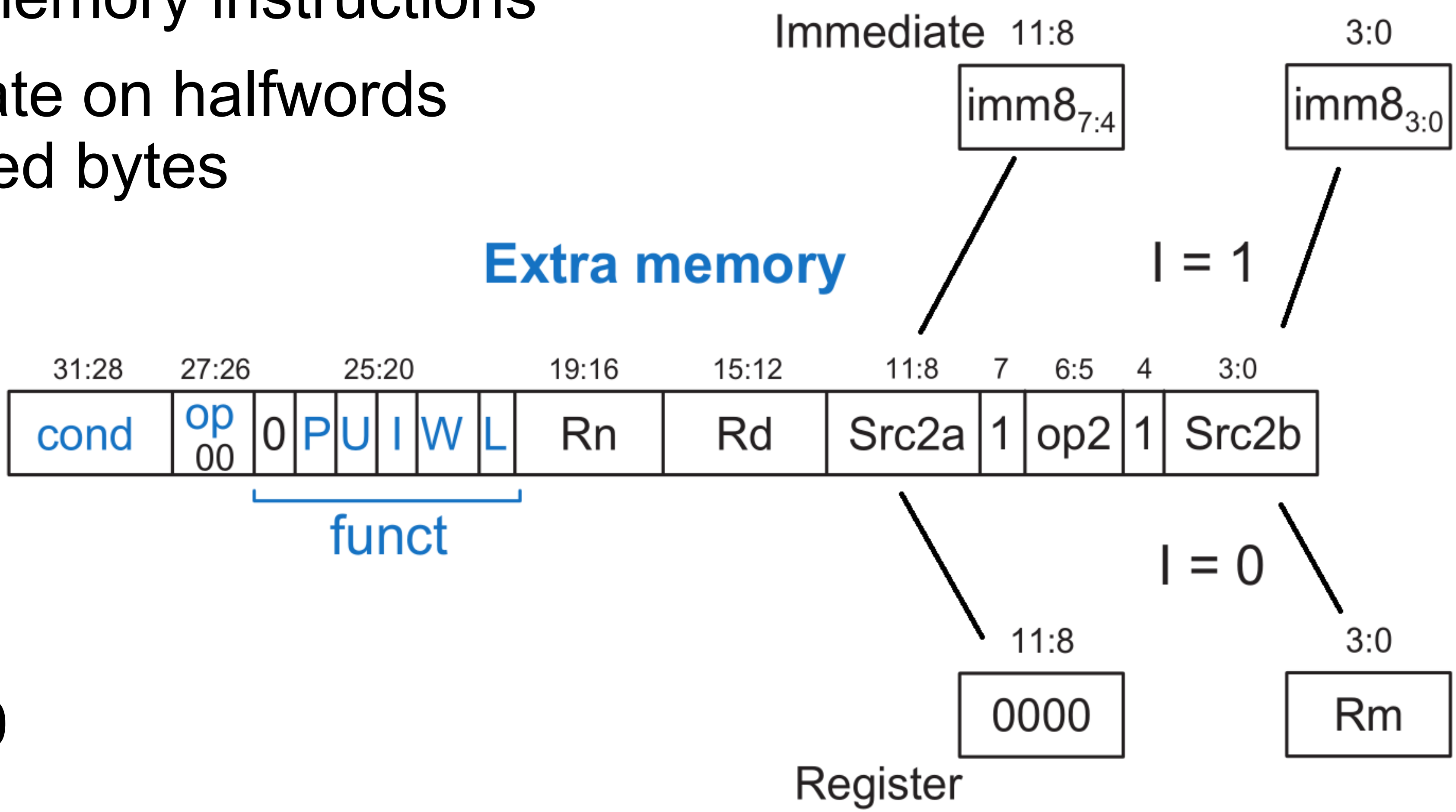
P	W	Index Mode
0	0	Post-index
0	1	Not supported
1	0	Offset
1	1	Pre-index

L	B	Instr
0	0	STR
0	1	STRB
1	0	LDR
1	1	LDRB

The offset is either a 12-bit unsigned immediate imm12 or a register Rm that is optionally shifted by a constant shamt5.

Extra memory instructions

– operate on halfwords
or signed bytes

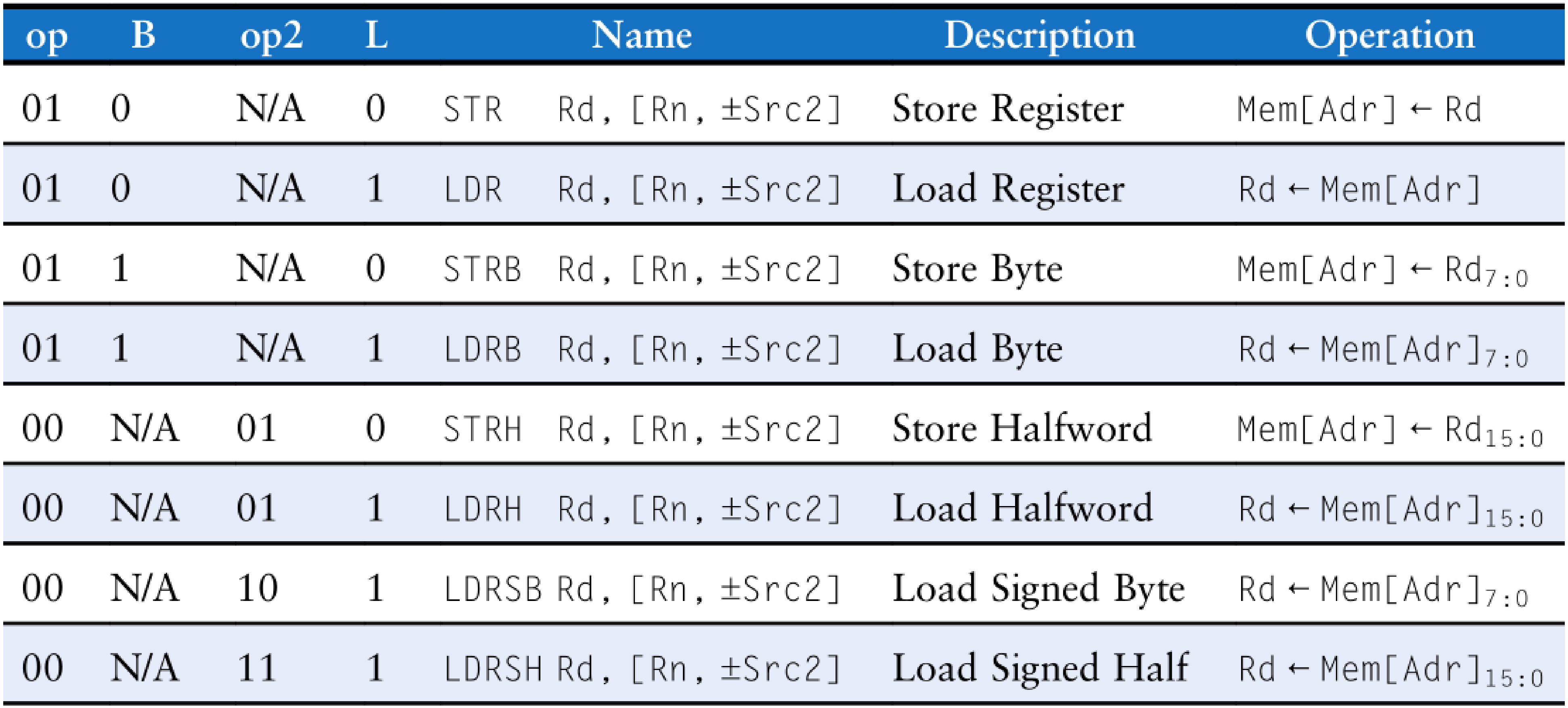


op = 00

The immediate offset is only 8 bits and the register offset cannot be shifted.

LDRB and LDRH zero-extend the bits to fill a word, while LDRSB and LDRSH sign-extend the bits.

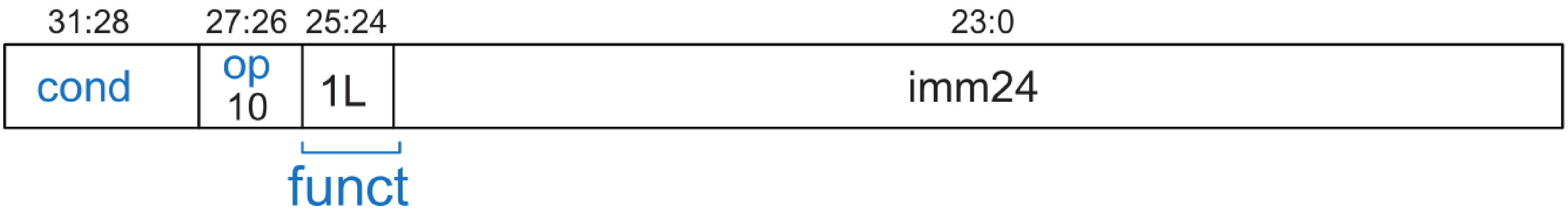
Memory instructions



op	B	op2	L	Name	Description	Operation
01	0	N/A	0	STR Rd, [Rn, ±Src2]	Store Register	Mem[Adr] ← Rd
01	0	N/A	1	LDR Rd, [Rn, ±Src2]	Load Register	Rd ← Mem[Adr]
01	1	N/A	0	STRB Rd, [Rn, ±Src2]	Store Byte	Mem[Adr] ← $Rd_{7:0}$
01	1	N/A	1	LDRB Rd, [Rn, ±Src2]	Load Byte	Rd ← $Mem[Adr]_{7:0}$
00	N/A	01	0	STRH Rd, [Rn, ±Src2]	Store Halfword	Mem[Adr] ← $Rd_{15:0}$
00	N/A	01	1	LDRH Rd, [Rn, ±Src2]	Load Halfword	Rd ← $Mem[Adr]_{15:0}$
00	N/A	10	1	LDRSB Rd, [Rn, ±Src2]	Load Signed Byte	Rd ← $Mem[Adr]_{7:0}$
00	N/A	11	1	LDRSH Rd, [Rn, ±Src2]	Load Signed Half	Rd ← $Mem[Adr]_{15:0}$

Branch instructions

Branch instructions use a single 24-bit signed immediate operand, imm24



op is 10 for memory instructions

The funct field is 2 bits.

The upper bit of funct is always 1 for branches.

The lower bit, L: 1 for BL and 0 for B

The remaining 24-bit two’s complement imm24 field is used to specify an instruction address relative to PC+8

```
      BLT THERE
      ADD R0, R1, R2
      SUB R0, R0, R9
      ADD SP, SP, #8
      MOV PC, LR
THERE SUB R0, R0, #1
      ADD R3, R3, #0x5
```

Machine code for BLT

31:28	27:26	25:24	23:0
1011	10	10	0000 0000 0000 0000 0000 0011
cond	op	funct	imm24

The **branch target address** (BTA) is the address of the next instruction to execute if the branch is taken.

The value in the immediate field (imm24) of BLT is 3 because the BTA is three instructions past PC+8.

Addressing modes

ARM uses four main modes:

- – register addressing
- – immediate addressing
- – base addressing
- – PC-relative addressing

Data-processing instructions use register or immediate addressing.

Memory instructions use base addressing.

Branches use PC-relative addressing in which the branch target address is computed by adding an offset to PC + 8.

Operand Addressing Mode	Example	Description
Register		
Register-only	ADD R3, R2, R1	R3 ← R2 + R1
Immediate-shifted register	SUB R4, R5, R9, LSR #2	R4 ← R5 − (R9 >> 2)
Register-shifted register	ORR R0, R10, R2, ROR R7	R0 ← R10 | (R2 ROR R7)
Immediate	SUB R3, R2, #25	R3 ← R2 − 25
Base		
Immediate offset	STR R6, [R11, #77]	mem[R11+77] ← R6
Register offset	LDR R12, [R1, −R5]	R12 ← mem[R1 − R5]
Immediate-shifted register offset	LDR R8, [R9, R2, LSL #2]	R8 ← mem[R9 + (R2 << 2)]
PC-Relative	B LABEL1	Branch to LABEL1

Interpreting machine language code

All three formats start with a 4-bit condition field and a 2-bit op.

The best place to begin is to look at the op.

If it is 00, then the instruction is a data-processing instruction.

If it is 01, then the instruction is a memory instruction.

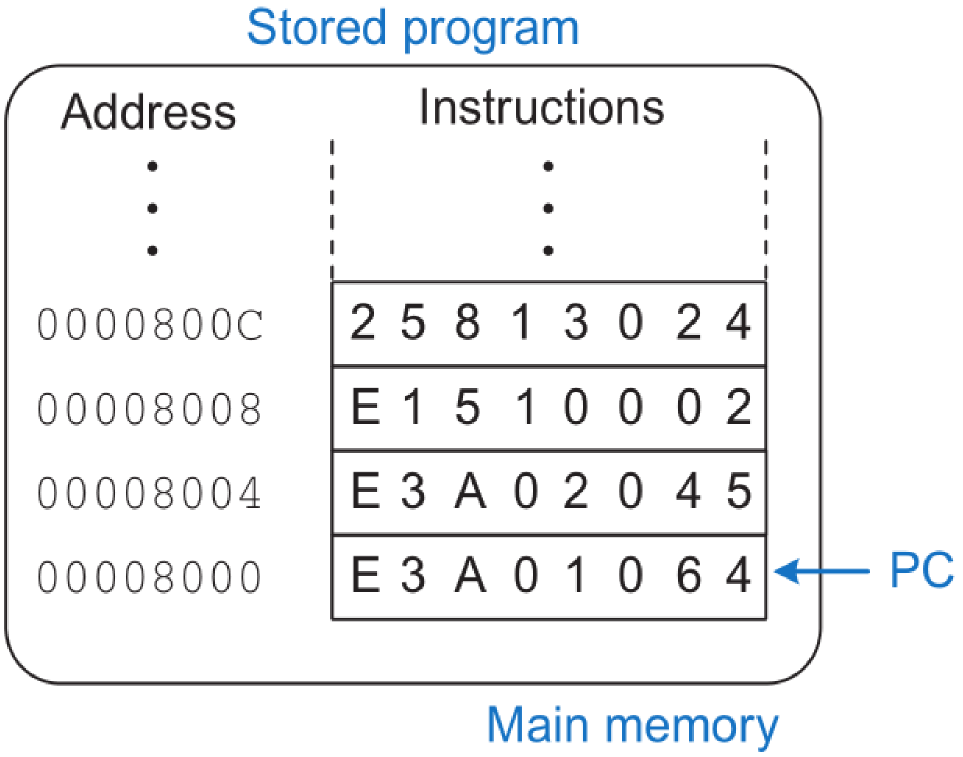
If it is 10, then it is a branch instruction.

Based on that, the rest of the fields can be interpreted.

A program written in machine language is a series of 32-bit numbers representing the instructions.

It is stored in memory.

Assembly code	Machine code
`MOV   R1, #100`	`0xE3A01064`
`MOV   R2, #69`	`0xE3A02045`
`CMP   R1, R2`	`0xE1510002`
`STRHS R3, [R1, #0x24]`	`0x25813024`



Stored program

Address	Instructions
⋮	⋮
0000800C	2 5 8 1 3 0 2 4
00008008	E 1 5 1 0 0 0 2
00008004	E 3 A 0 2 0 4 5
00008000	E 3 A 0 1 0 6 4

Main memory

The architectural state

The architectural state of a microprocessor holds the state of a program.

For ARM, the architectural state includes the register file and status registers.

If the operating system (OS) saves the architectural state at some point in the program, it can interrupt the program, do something else, and then restore the state such that the program continues properly, unaware that it was ever interrupted.

Exercise 10.1 In this lesson we studied what is the machine code for the instructions like

```
MOV R1, #0x000000AB
MOV R1, #0x0000AB00
MOV R1, #0xAB000000
```

Each of them will be of the form E3A… with the cmd field equal to 1101

What will be the form of machine instructions corresponding to

```
MOV R1, #0xFFFFFFAB
MOV R1, #0xFFFFABFF
MOV R1, #0xABFFFFFF
```

What will be their cmd field?

Which instruction it represents?

Exercise 10.2 Calculate the immediate field and show the machine code for the branch instruction in the following assembly program.

```
      BLT THERE
      ADD R0, R1, R2
      SUB R0, R0, R9
      ADD SP, SP, #8
      MOV PC, LR
THERE SUB R0, R0, #1
      ADD R3, R3, #0x5
```

Explain each field in the machine code for the branch instruction.

Exercise 10.3 Consider the following code that contains two branch instructions

```
  MOV R0, #1        ; pow = 1
  MOV R1, #0        ; x = 0
WHILE
  CMP R0, #128      ; pow != 128 ?
  BEQ DONE          ; if pow == 128, exit loop
  LSL R0, R0, #1    ; pow = pow * 2
  ADD R1, R1, #1    ; x = x + 1
  B WHILE           ; repeat loop
DONE
```

Obtain the machine code for each brach instruction.

Explain every field in each machine code.

Exercise 10.4 Find the machine code that corresponds to the instruction

```
stop  B stop
```

Explain this machine code.

Exercise 10.5 Run the code for the exercises from the lesson 7. Explain the machine code for memory instructions in those exercises.

Exercise 10.6 Explain the machine code for each instruction in the recursive function that calculates the factorial in the lesson 9.

You have to complete the exercise 9.2 before doing this exercise.