What is the decimal form of the 4-bit, signed, 2's complement number `1101`?

A: -4
B: -2
C: 1
D: -3
E: Something else

What is the decimal form of the 8-bit, signed, 2's complement number `4f`?

A: `81`

B: `79`

C: `55`

D: `-77`

E: Something else

```
  01100101
+ 00100010
----------
  10000111
```

01100101	**Operand A**
00100010	**Operand B**
10000111	**Result**
+	***Operation***

This must run somehow on a physical machine!

Where (physically) are Op A/B, the Result, and the Instruction?

And the ***Operation*** had better be encoded in binary!

Registers

some fixed width in bits

r0	
r1	01100101
r2	00100010
r3	

Instructions

"add r1 to r2 and store in r0"

00 00 01 10

Which operation?
opcode

Where are operands?
source registers

Where to store result?
destination register

To run a program, the machine uses

The content of ***Registers***

r0	
r1	01100101
r2	00100010
r3	

A list of ***Instructions***, with a ***Current Instruction***

```
00 01 10 11
01 10 10 11
10 10 10 11
...
```

The rules for the ***Instruction Set***

1. If the opcode is 00, add the contents of the operand registers, store the result in the destination register.
2. If the opcode is 01, …

The machine runs each instruction in order

```
00 01 10 11
```

Which operation?
opcode

Where are operands?
source registers

Where to store result?
destination register

Registers

r0	~~00001000 (8)~~ 00100000 (32)
r1	00000110 (6)
r2	00000010 (2)
r3	00000100 (4)

```
00 00 01 10   “put r1 + r2 in r0”
01 11 01 10   “put r1 - r2 in r3”
10 00 00 11   “put r0 * r3 in r0”
```

Instructions (The Program!)

Instruction Set

1. If the opcode is `00`, **add** the contents of the source registers, store the result in the destination register.
2. If the opcode is `01`, **subtract** the contents of the second source from the first, store the result in the destination register.
3. If the opcode is `10`, **multiply** the contents of the source registers, store the result in the destination register.

Registers

`r0`	`00001111 (15)`
`r1`	`00010100 (20)`
`r2`	`00000101 (5)`
`r3`	

```
00 11 00 01
00 11 11 10
```

What is in `r3` after this runs?

A: `00100011 (35)`

B: `00010100 (20)`

C: `00101000 (40)`

D: `00011001 (25)`

E: Something else

Instruction Set

1. If the opcode is `00`, **add** the contents of the source registers, store the result in the destination register.
2. If the opcode is `01`, **subtract** the contents of the second source from the first, store the result in the destination register.
3. If the opcode is `10`, **multiply** the contents of the source registers, store the result in the destination register.

Registers

r0	00001111 (15)
r1	00010100 (20)
r2	00000101 (5)
r3	

01 11 01 00
00 11 11 10
10 11 11 10

What is in **r3** after this runs?

A: 01100100 (100)
B: 00010100 (20)
C: 00101000 (40)
D: 00110010 (50)
E: Something else

Instruction Set

1. If the opcode is 00, **add** the contents of the source registers, store the result in the destination register.
2. If the opcode is 01, **subtract** the contents of the second source from the first, store the result in the destination register.
3. If the opcode is 10, **multiply** the contents of the source registers, store the result in the destination register.

Registers

r0	00000010 (2)
r1	00010100 (20)
r2	00000101 (5)
r3	

Which of these programs ends with 60 in **r3**?

A
```
10 10 10 10
00 01 10 01
00 11 01 10
```

C
```
00 11 10 10
00 11 10 01
00 11 11 11
```

B
```
00 10 10 10
00 11 10 01
10 11 00 11
```

D
```
00 10 10 10
00 11 10 01
00 11 11 11
```

Instruction Set

1. If the opcode is 00, **add** the contents of the source registers, store the result in the destination register.
2. If the opcode is 01, **subtract** the contents of the second source from the first, store the result in the destination register.
3. If the opcode is 10, **multiply** the contents of the source registers, store the result in the destination register.

Registers

r0	
r1	
r2	
r3	

```
11 00 00 10
11 01 01 00
00 10 01 00
```

What is in **r2** after this runs?

A: `00000110` (6)
B: `00000000` (0)
C: `00001000` (8)
D: `00010110` (22)
E: Something else

Instruction Set

1. If the opcode is `00`, **add** the contents of the source registers, store the result in the destination register.
2. If the opcode is `01`, **subtract** the contents of the second source from the first, store the result in the destination register.
3. If the opcode is `10`, **multiply** the contents of the source registers, store the result in the destination register.
4. **If the opcode is `11`, interpret the last 4 bits as a constant, and move them into the destination register**

```
00 Rd Rn Rm    put Rn + Rm in Rd
01 Rd Rn Rm    put Rn - Rm in Rd
10 Rd Rn Rm    put Rn * Rm in Rd
11 Rd Imm4     put Imm4 in Rd
```

ARM: (just a few) more opcodes and options

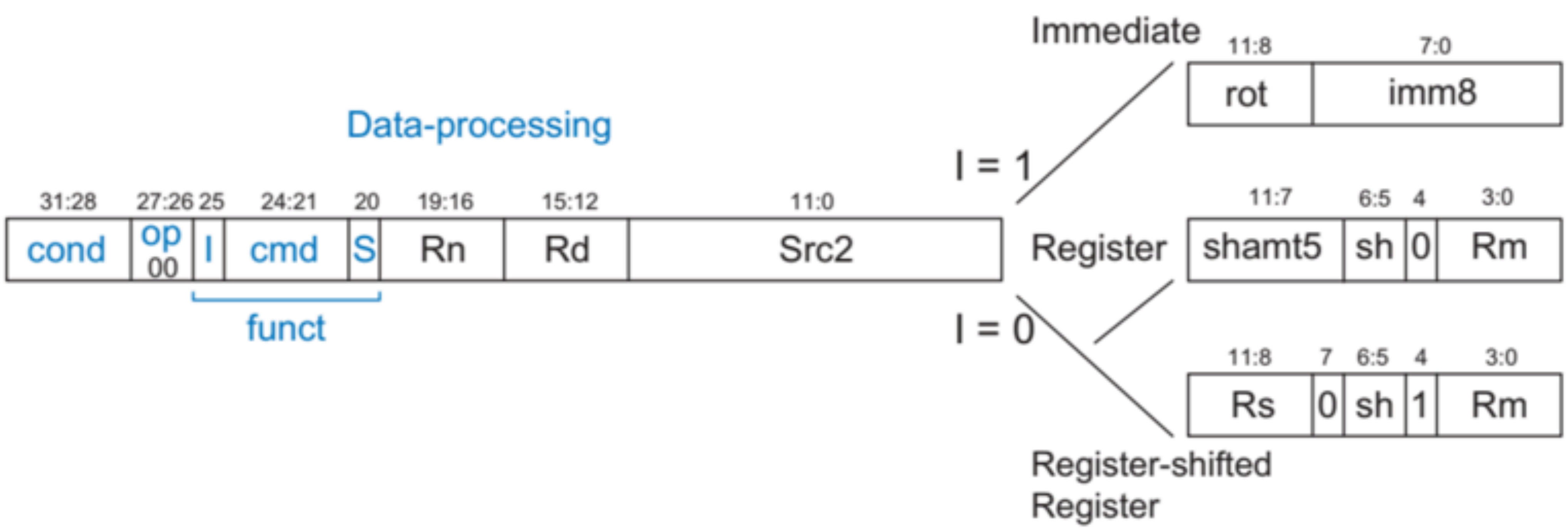

Figure B.1 **Data-processing instruction encodings**

Harris et al, *Digital Design and Computer Architecture, Second Edition*. 2016. Appendix B

Some definitions:

word/word size: The number of bits in an instruction/register. In this lecture, we had 8-bit words. Our pi-cluster ARM will use 32-bit words.

opcode: The part of the instruction that determines which operation to perform. Our pi-cluster ARM will have many more opcodes (4 bits and more!)

instruction: A word that the processor interprets to perform a step of computation. Usually this means changing the value in some register. There are *lots* of instructions we'll talk about this quarter.

register: A word of quick-access memory, where most computation happens. Data is stored here briefly before being stored elsewhere. Our pi-cluster ARM uses 16 registers.

immediate value: A constant number that appears as part of an instruction.

Instruction Set

1. If the opcode is `00`, **add** the contents of the source registers, store the result in the destination register.
2. If the opcode is `01`, **subtract** the contents of the second source from the first, store the result in the destination register.
3. If the opcode is `10`, **multiply** the contents of the source registers, store the result in the destination register.
4. If the opcode is `11`, interpret the last 4 bits as a constant, and move them into the destination register