In the previous example, an analysis of the memory accesses that occur during the execution of the program reveals two ways to reduce the effect of the von Neumann bottleneck:

1. Reduce the length of the binary sequence representing an instruction so that it can be stored in fewer words of memory.
2. Reduce the number of operands that reference memory.

The following example illustrates how an instruction set can be designed to apply these to observations.

EXAMPLE 2: A 1-operand machine design:

Number of instructions: 16

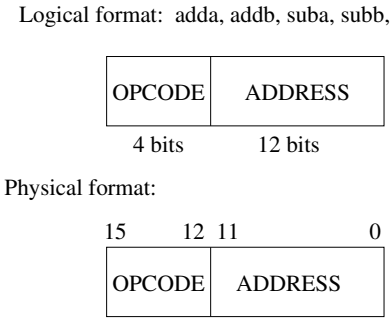
External memory, M: $2^{12} \times 16$ bits

Internal (CPU) memory, R: 2×16 bits

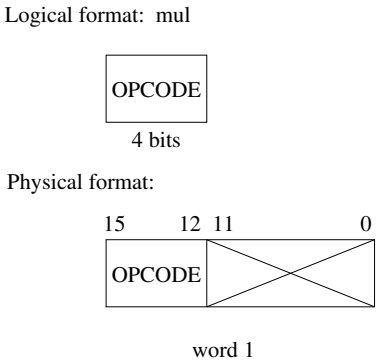
1. Proposed Instruction Set (partial):

	SYNTAX		SEMANTICS
OPCODE	INST	OPNDS	
1	ADD0	A	R[0] $\leftarrow$ R[0] + M[A]
2	ADD1	A	R[1] $\leftarrow$ R[1] + M[A]
3	SUB0	A	R[0] $\leftarrow$ R[0] $-$ M[A]
4	SUB1	A	R[1] $\leftarrow$ R[1] $-$ M[A]
5	MPY		R[0] : R[1] $\leftarrow$ R[0] $\times$ R[1]
6	LD0	A	R[0] $\leftarrow$ M[A]
7	LD1	A	R[1] $\leftarrow$ M[A]
8	ST0	A	M[A] $\leftarrow$ R[0]
9	ST1	A	M[A] $\leftarrow$ R[1]

2. The instruction format for all the instructions except "`MPY`" is:



The instruction format for "`MPY`" is:



3. Analysis of the number of memory fetches for the program is as follows:

Program to compute z = (p + q) * (p - q)

SYNTAX		SEMANTICS	accesses fetch	accesses exec
ld0	p	R[0] ← M[p]	1	1
add0	q	R[0] ← R[0] + M[q]	1	1
ld1	p	R[1] ← M[p]	1	1
sub1	q	R[1] ← R[1] − M[q]	1	1
mpy		R[0] : R[1] ← R[0] × R[1]	1	0
st0	z	M[z]← R[0]	1	1
st1	z+1	; M[z+1]← R[1]	1	1

The total number of memory accesses is 13, a significant improvement over the first program, although the programmer must write a longer program.

4. Memory Map of the Machine Language Program:

ADDR	CONTENTS
000	6A00
001	1A01
002	7A00
003	4A01
004	5000
005	8A02
006	9A03
....	
A00	8000 (value of p)
A01	0002 (value of q)
A02	0001 (value of z(31:16))
A03	0000 (value of z(15 : 0))

EXAMPLE 3: A 2-operand machine design:

Number of instructions: 16

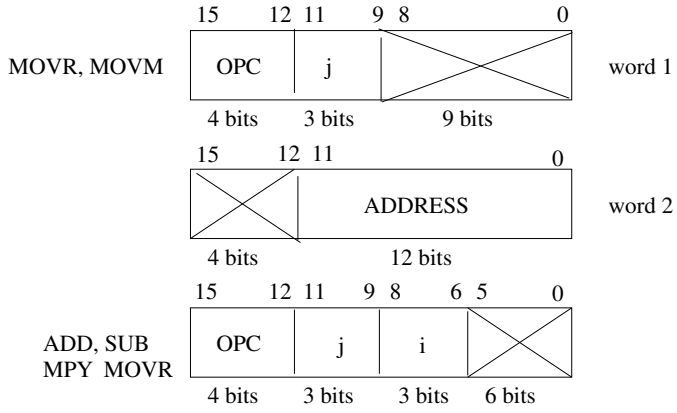
External memory, M: $2^{16} \times 16$ bits

Internal (CPU) memory, R: $2^3 \times 16$ bits

Proposed Instruction Set (partial):

	SYNTAX		SEMANTICS
OPCODE	INST	OPNDS	
		(i, j $\in$ {0,..7})	
1	ADD	Ri, Rj	R[j] $\leftarrow$ R[j] + R[i]
3	SUB	Ri, Rj	R[j] $\leftarrow$ R[j] $-$ Ri]
5	IMUL	Ri, Rj	R[i] : R[j] $\leftarrow$ R[j] $\times$ R[i]
6	MOVM	Rj, A	M[A] $\leftarrow$ R[j]
7	MOVR	A, Rj	R[j] $\leftarrow$ M[A]
8	MOVR	Ri, Rj	R[j] $\leftarrow$ R[i]

Physical Instruction Formats:



Analysis of memory accesses:

Expression evaluation: z = (p + q) * (p - q)

INSTRUCTION Syntax		Semantics	accesses fetch	exec
movr	p, R1	R[1] ← M[p]	2	1
movm	q, R2	R[2] ← M[q]	2	1
movr	R2, R3	R[3] ← R[2]	1	0
add	R1, R2	R[2]] ← R[2] + R[1]	1	0
sub	R3, R1	R[1]] ← R[1] − R[3]	1	0
imul	R2, R1	R[1] : R[2] ← R[2] × R[1]	1	0
movm	R2, z	M[z] ← R[2]	2	1
movm	R1, z+1	M[z+1] ← R[1]	2	1

A total of 16 memory accesses will occur. Examination of this and previous examples indicates that the best result that we can expect during a fetch occurs with one-word instructions. At the same time, there must be at least one memory access during execution to retrieve or store the values in external memory. Therefore, if we are to achieve an improvement, we will need to reduce the two-word instructions to one word. The way to do this is to eliminate the address operands. This is only possible if we interpret the operands differently. The way in which operands are to be interpreted is called their "addressing mode."

6.3 Addressing Modes

"Addressing modes" or "address modes" describe how the explicit operands of an instruction are to be interpreted. Among the previous examples, the following addressing modes have been introduced:

Direct Mode : The explicit operand(s) specify address(es) in memory where data values are stored. Direct mode allows for larger values to be delivered as the operands of arithmetic and logic functions, since the size of the value is determined by the space provided in memory. However, the instructions are necessarily longer, since the explicit operands are addresses.

Register Mode : The explicit operand(s) specify memory within the CPU where data values are stored. CPU's commonly include register files in contemporary architectures to provide this memory. Values stored in the register file are retrieved by providing an "address" called the *register number*. Because the size of a register file is very small, the space required in an instruction to hold a register number (that is, an address for a location in internal memory) is very much smaller than the space required to hold an address and consequently register mode instructions are much shorter than direct mode instructions.

Implied Mode : There are no operands specified (only the opcode). Implied mode instructions, as a consequence, are very short instructions and therefore require only a minimum amount of memory for their storage and a minimum number of accesses for their retrieval. There is no flexibility with regard to the operands, however, because they are all implicit.

The following addressing modes expand the ways that operands can be used to identify the actual value of and operand:

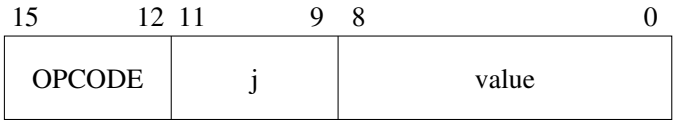
Immediate Mode : The explicit operand(s) specify the actual values of the operands. That is, the data value does not need to be retrieved from memory because it is provided as part of the instruction. Because the values are explicit operands, there is a trade-off between instruction length and the magnitude of the values that can be used.

Some examples of instructions that might be added to the instruction set of example 3:

	SYNTAX		SEMANTICS
OPC	INST	OPNDS (j ∈{0,..7})	
2	ADDI	value, Rj	R[j] ← R[j] + SE(value)
4	SUBI	value, Rj	R[j] ← R[j] − SE(value)
9	MOVI	value,Rj	R[j] ← 0x0 : SE(value)

SE(x) is a function to sign-extend an operand. For an explanation see below.

In keeping with the desire to restrict instructions to one word whenever possible, a possible physical format for this group of instructions is:



Because the operand, `value`, is 9 bits while the contents of R[j] is 16 bits, `value` must be "sign extended" to 16 bit. That is, the 9-bit value must be converted to a 16-bit value. SInce integers are represented by 2's complement codewords, to sign-extend a 9-bit codeword to a 16 bit-codeword, the sign bit (bit 8) is copied to bit positions 9 through 15 to construct the 16-bit codeword.

Register Indirect Mode : The explicit operand specifies a register within the CPU that holds the address to a value stored in memory. Indirect mode instructions are long because the explicit operand is an address. If, instead, a register is used to hold the address to a value stored in memory, then the number of bits required for the explicit operand is significantly less, since that operand specifies a register. This mode is analogous to index mode, but with the displacement always 0.

Some examples:

	SYNTAX		SEMANTICS
OPC	INST	OPNDS (i, j ∈{0,..7})	
A	MOVRI	Ri, Rj	R[j] ← M[R[i]]
B	MOVMI	Ri, Rj	M[R[j]] ← R[i]

Base + Displacement Mode : An explicit operand specifies a register within the register file where a memory address is located. A second explicit operand provides a value, called a *displacement* that, together with the contents of the register, is used to compute the address where a value is stored. This computed address is called an *effective address*. Index mode is an effective way to access a contiguous sequence of locations in memory and therefore is often used to retrieve and update the values in arrays.

Some examples:

	SYNTAX		SEMANTICS
OPC	INST	OPNDS (i, j ∈{0,..7})	
6	MOVR	value(Ri), Rj	R[j] ← M[R[i] + SE(value)]
7	MOVM	Ri, value(Rj)	M[R[j] + SE(value)] ← R[i]

Rewriting the program using index mode instructions rather than direct mode instructions to load and store:

INSTRUCTION			accesses	
Syntax		Semantics	fetch	exec
MOVI	$0x0, R7	R[7]← 0x0000	1	0
MOVR	0(R7), R1	R[1] ← M[R[7] + 0000]	1	1
MOVR	1(R7), R2	R[2] ← M[R[7] + 0001]	1	1
MOVR	R2, R3	R[3] ← R[2]	1	0
ADD	R1, R2	R[2]] ← R[2] + R[1]	1	0
SUB	R3, R1	R[1]] ← R[1] − R[3]	1	0
IMUL	R1, R2	R[1] & R[2] ← R[2] × R[1]	1	0
MOVM	R1, 2(R7)	M[R[7] + 0002] ← R[1]	1	1
MOVM	R2, 3(R7)	M[R[7] + 0003] ← R[2]	1	1

The total number of accesses is now 13 accesses.