

# Chapter 6

## Architecture

Word Address	Data	
⋮	⋮	⋮
00000003	4 0 F 3 0 7 8 8	Word 3
00000002	0 1 E E 2 8 4 2	Word 2
00000001	F 2 F 1 A C 0 7	Word 1
00000000	A B C D E F 7 8	Word 0

**Figure 6.1 Word-addressable memory**

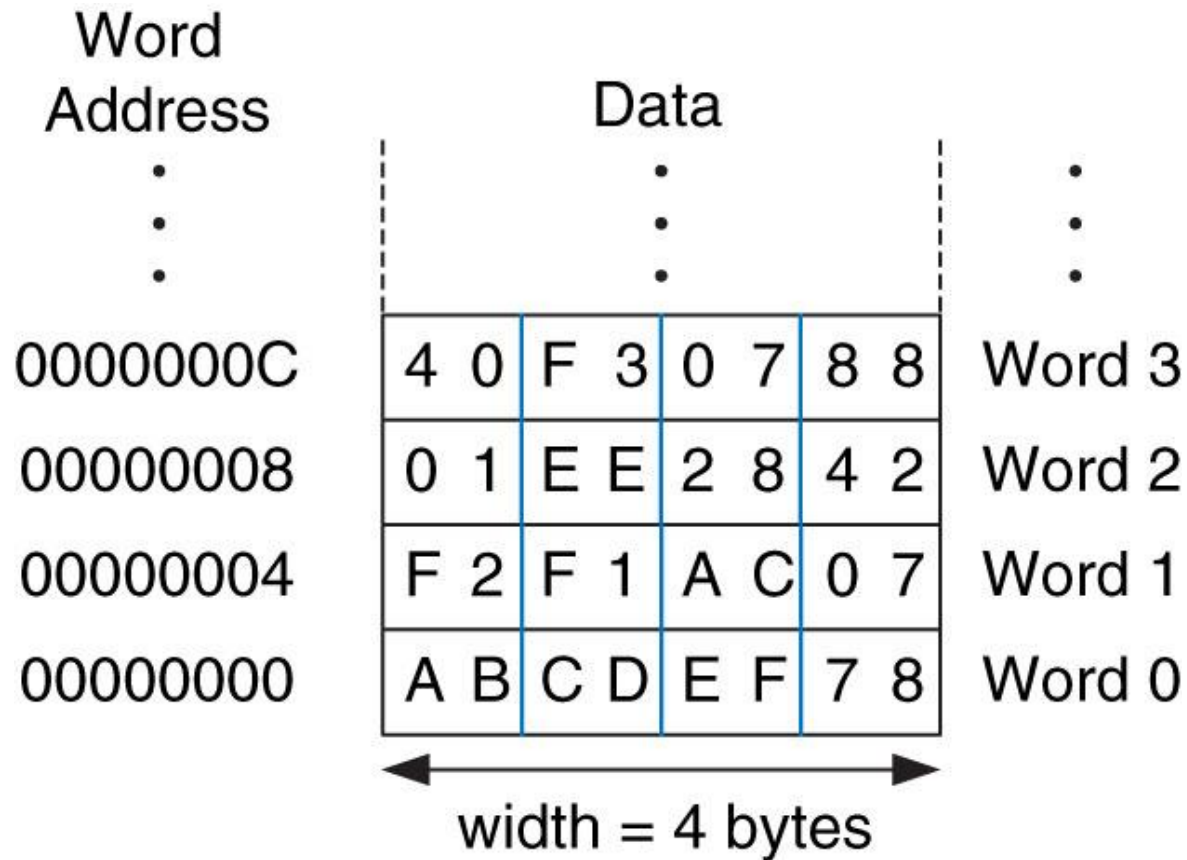
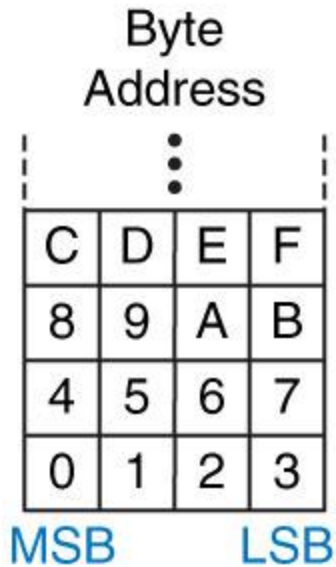


Figure 6.2 Byte-addressable memory

## Big-Endian



## Little-Endian

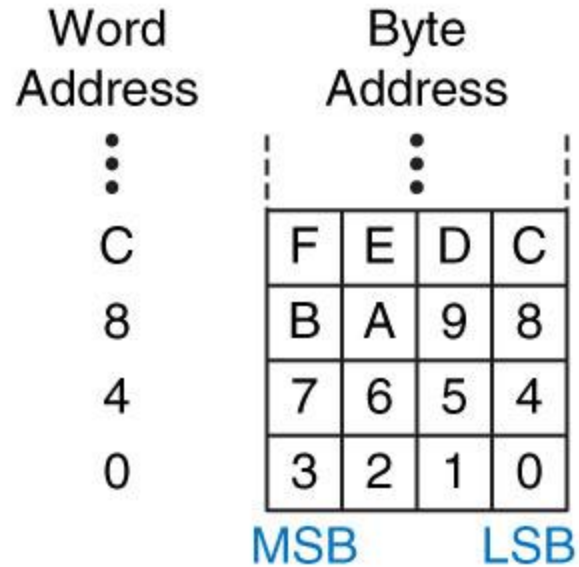


Figure 6.3 Big- and little-endian memory addressing

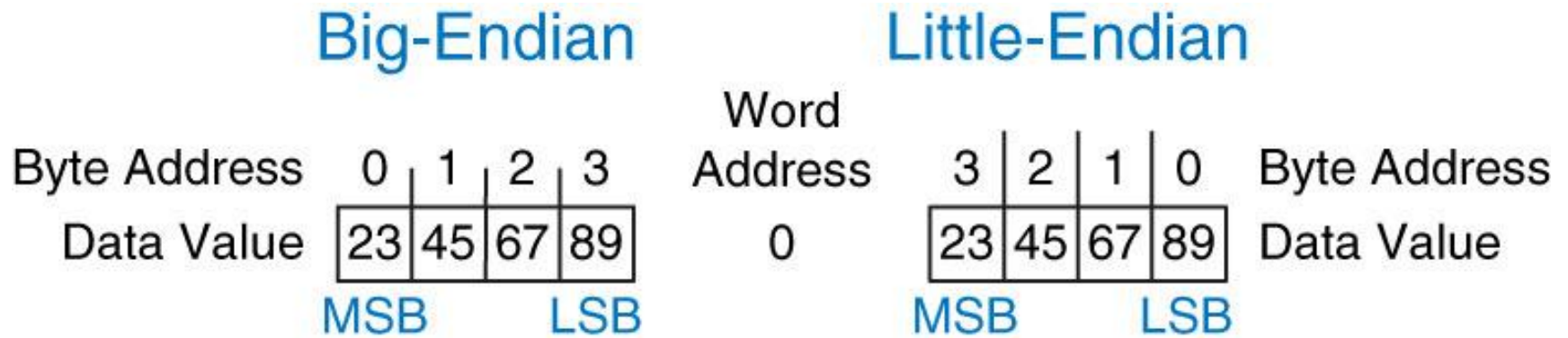


Figure 6.4 Big-endian and little-endian data storage

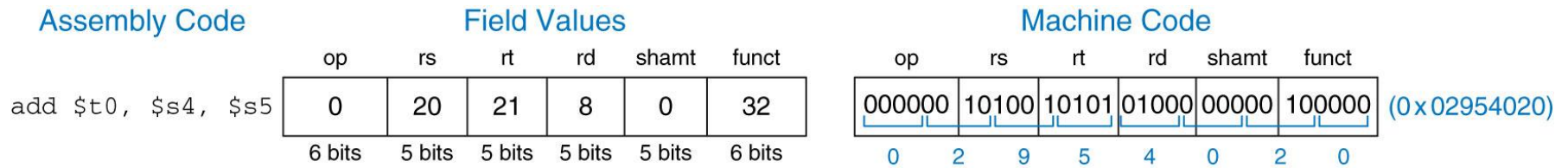
## R-type



Figure 6.5 R-type machine instruction format

Assembly Code		Field Values						Machine Code					
		op	rs	rt	rd	shamt	funct						
add \$s0, \$s1, \$s2		0	17	18	16	0	32	000000	10001	10010	10000	00000	100000 (0x02328020)
sub \$t0, \$t3, \$t5		0	11	13	8	0	34	000000	01011	01101	01000	00000	100010 (0x016D4022)
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

**Figure 6.6 Machine code for R-type instructions**



**Figure 6.7 Machine code for the R-type instruction of Example 6.3**



## I-type

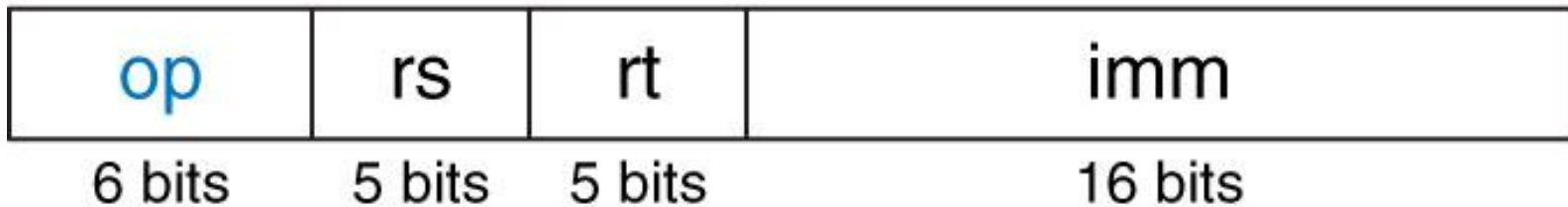
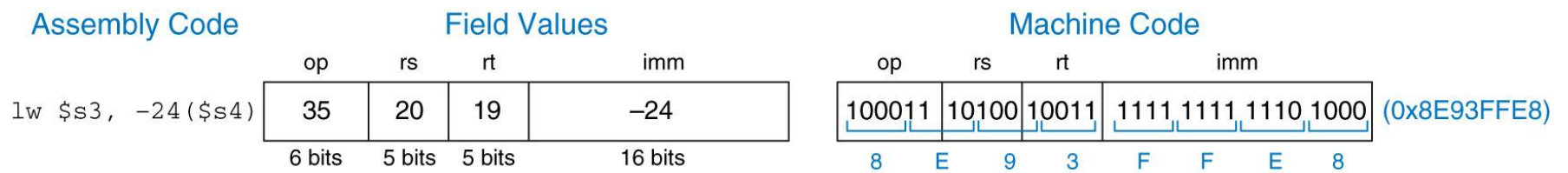


Figure 6.8 I-type instruction format

Assembly Code		Field Values				Machine Code			
		op	rs	rt	imm	op	rs	rt	imm
addi \$s0, \$s1, 5		8	17	16	5	001000	10001	10000	0000 0000 0000 0101 (0x22300005)
addi \$t0, \$s3, -12		8	19	8	-12	001000	10011	01000	1111 1111 1111 0100 (0x2268FFF4)
lw \$t2, 32(\$0)		35	0	10	32	100011	00000	01010	0000 0000 0010 0000 (0x8C0A0020)
sw \$s1, 4(\$t1)		43	9	17	4	101011	01001	10001	0000 0000 0000 0100 (0xAD310004)
		6 bits	5 bits	5 bits	16 bits	6 bits	5 bits	5 bits	16 bits

**Figure 6.9 Machine code for I-type instructions**



**Figure 6.10 Machine code for an I-type instruction**

## J-type

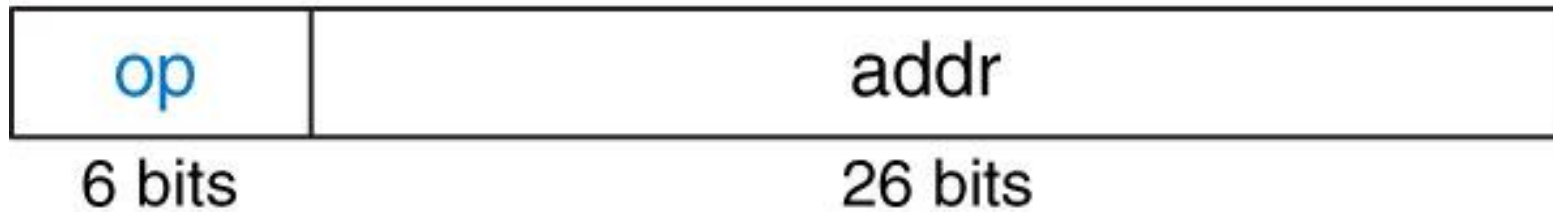
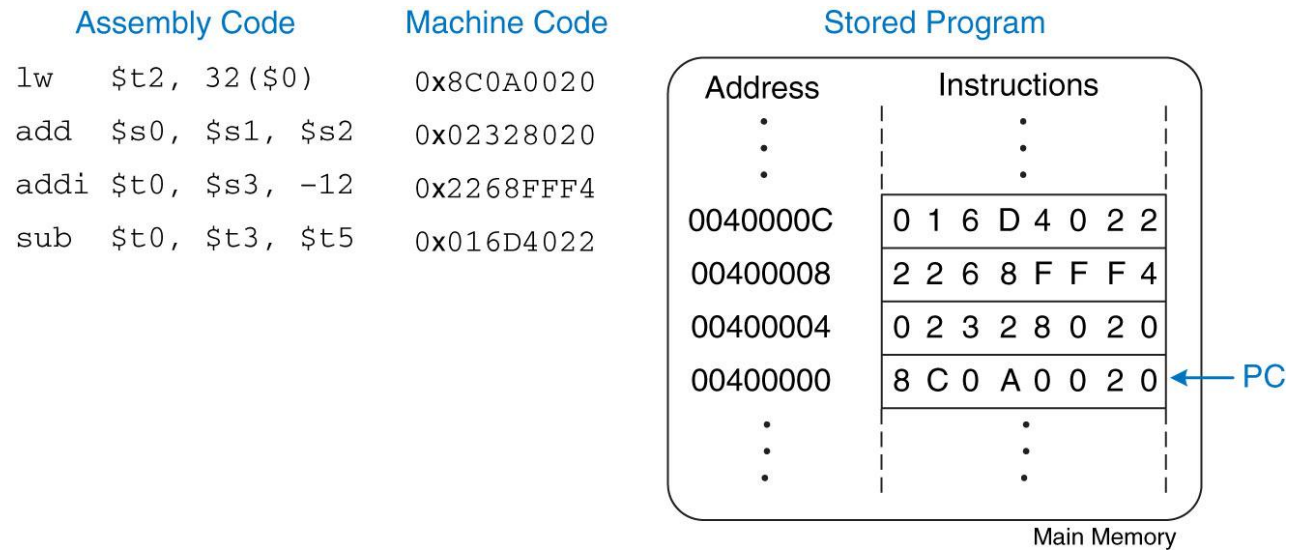


Figure 6.11 J-type instruction format

	Machine Code				Field Values				Assembly Code
	op	rs	rt	imm	op	rs	rt	imm	
(0x2237FFF1)	001000	10001	10111	1111 1111 1111 0001	8	17	23	-15	addi \$s7, \$s1, -15
	2	2	3	7 F F F 1					
	op	rs	rt	rd	shamt	funct			
(0x02F34022)	000000	10111	10011	01000	00000	100010			
	0	2	F	3	4	0	2	2	
	op	rs	rt	rd	shamt	funct			
	0	23	19	8	0	34			sub \$t0, \$s7, \$s3

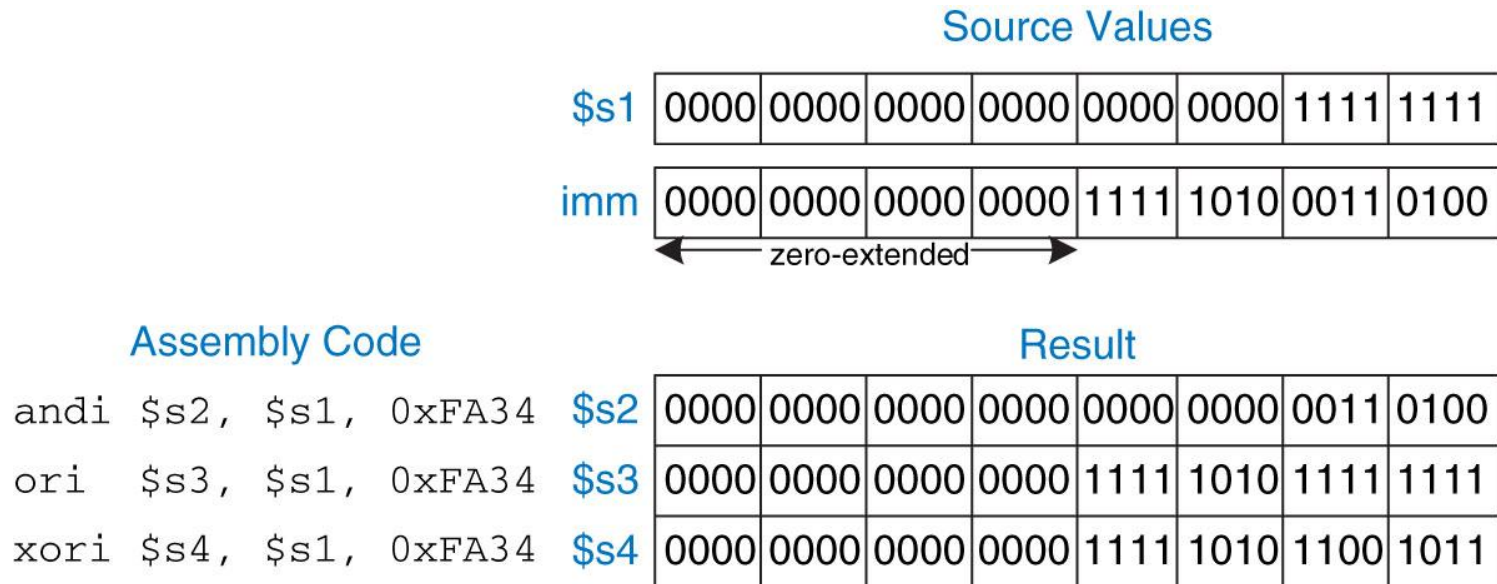
**Figure 6.12 Machine code to assembly code translation**



**Figure 6.13 Stored program**

		Source Registers							
	\$s1	1111	1111	1111	1111	0000	0000	0000	0000
	\$s2	0100	0110	1010	0001	1111	0000	1011	0111
Assembly Code		Result							
and \$s3, \$s1, \$s2	\$s3	0100	0110	1010	0001	0000	0000	0000	0000
or \$s4, \$s1, \$s2	\$s4	1111	1111	1111	1111	1111	0000	1011	0111
xor \$s5, \$s1, \$s2	\$s5	1011	1001	0101	1110	1111	0000	1011	0111
nor \$s6, \$s1, \$s2	\$s6	0000	0000	0000	0000	0000	1111	0100	1000

**Figure 6.14 Logical operations**

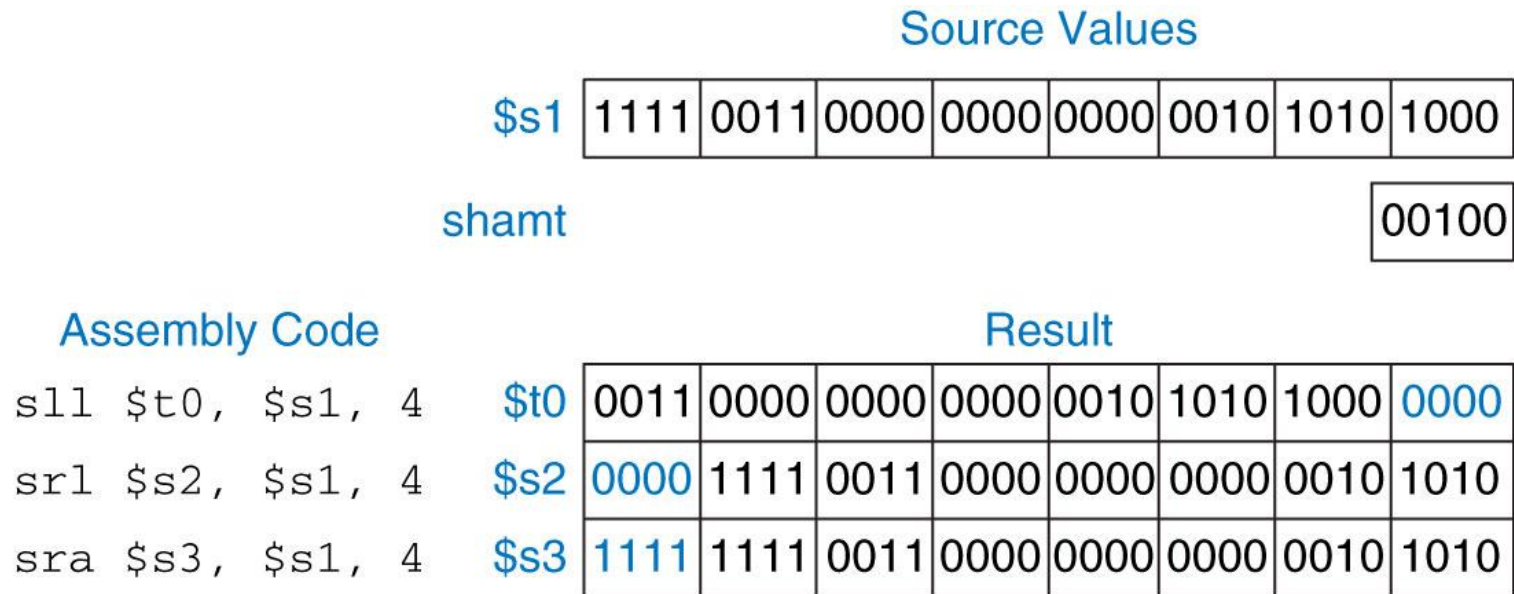


**Figure 6.16 Shift instruction machine code**



Assembly Code		Field Values						Machine Code					
		op	rs	rt	rd	shamt	funct	op	rs	rt	rd	shamt	funct
sll \$t0, \$s1, 4		0	0	17	8	4	0	000000	00000	10001	01000	00100	000000 (0x00114100)
srl \$s2, \$s1, 4		0	0	17	18	4	2	000000	00000	10001	10010	00100	000010 (0x00119102)
sra \$s3, \$s1, 4		0	0	17	19	4	3	000000	00000	10001	10011	00100	000011 (0x00119903)
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

**Figure 6.16 Shift instruction machine code**



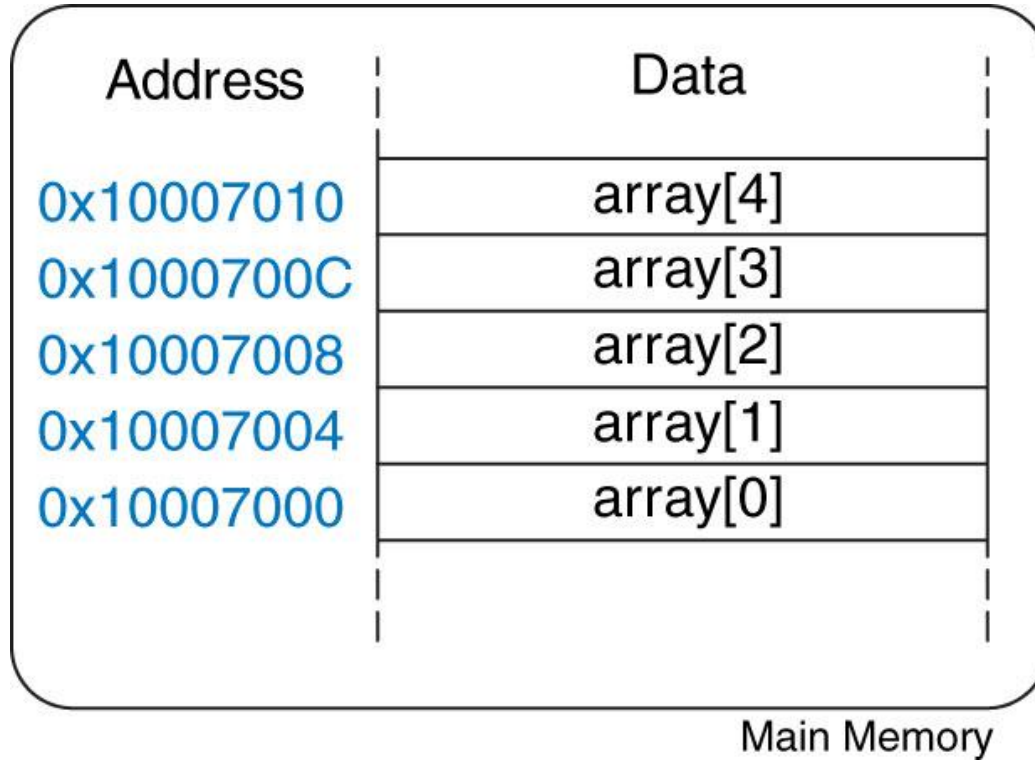
**Figure 6.17 Shift operations**

Assembly Code		Field Values						Machine Code					
		op	rs	rt	rd	shamt	funct	op	rs	rt	rd	shamt	funct
sllv \$s3, \$s1, \$s2		0	18	17	19	0	4	000000	10010	10001	10011	00000	000100 (0x02519804)
srlv \$s4, \$s1, \$s2		0	18	17	20	0	6	000000	10010	10001	10100	00000	000110 (0x0251A006)
srav \$s5, \$s1, \$s2		0	18	17	21	0	7	000000	10010	10001	10101	00000	000111 (0x0251A807)
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

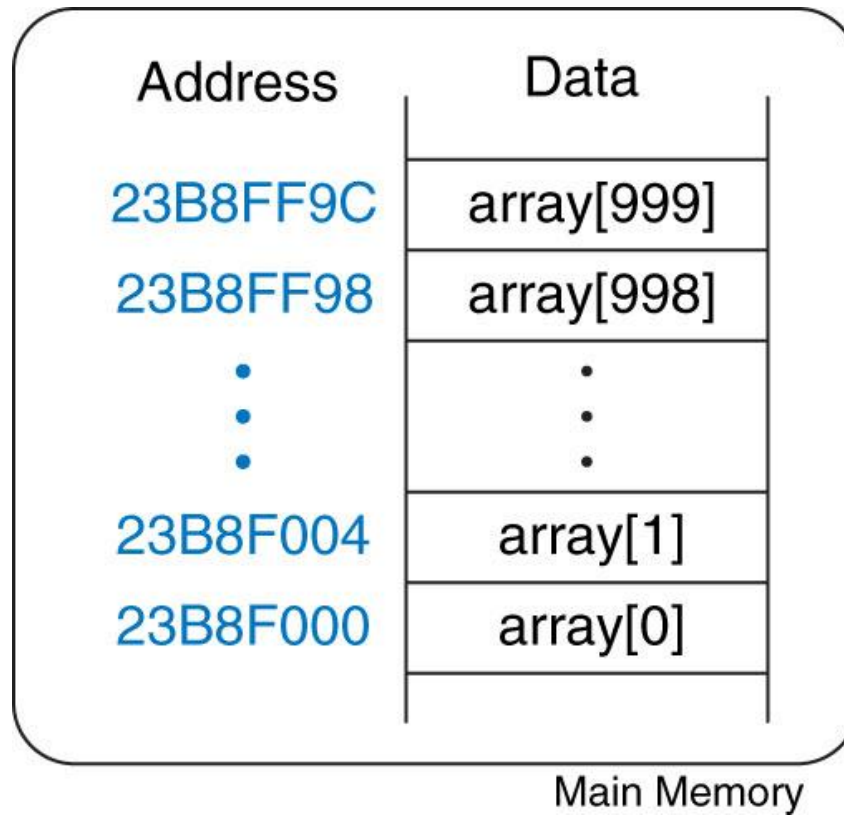
**Figure 6.18 Variable-shift instruction machine code**

		Source Values							
		\$s1	1111	0011	0000	0100	0000	0010	1010 1000
		\$s2	0000	0000	0000	0000	0000	0000	0000 1000
Assembly Code		Result							
sllv \$s3, \$s1, \$s2	\$s3	0000	0100	0000	0010	1010	1000	0000	0000
srlv \$s4, \$s1, \$s2	\$s4	0000	0000	1111	0011	0000	0100	0000	0010
srav \$s5, \$s1, \$s2	\$s5	1111	1111	1111	0011	0000	0100	0000	0010

**Figure 6.19 Variable-shift operations**



**Figure 6.20 Five-entry array with base address of 0x10007000**



**Figure 6.21** Memory holding `array[1000]` starting at base address `0x23B8F000`

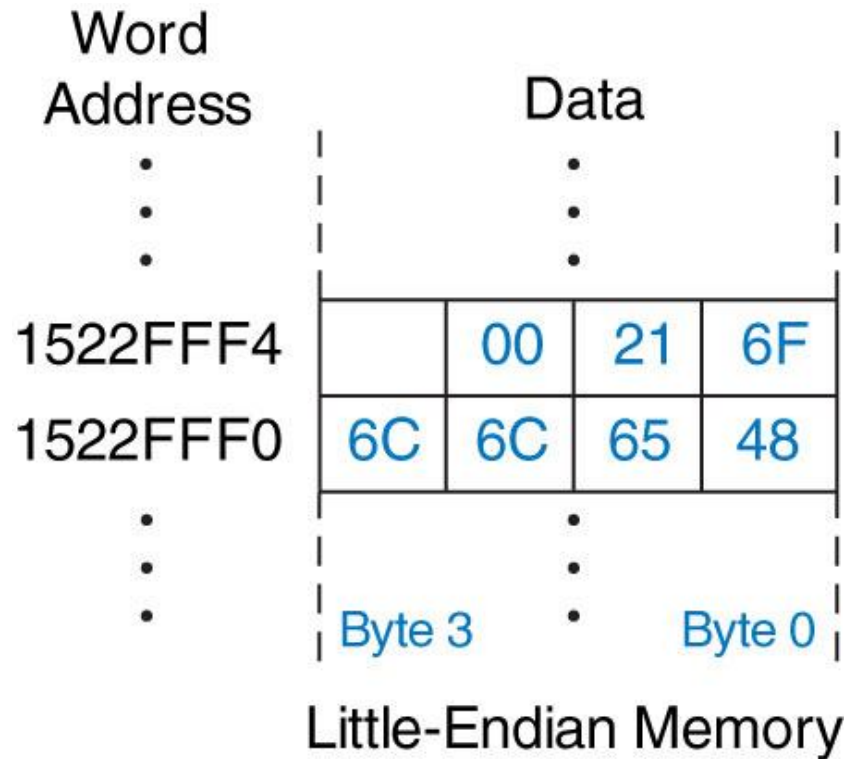
## Little-Endian Memory

Byte Address	3	2	1	0
Data	F7	8C	42	03

## Registers

\$s1	00	00	00	8C	lbu	\$s1, 2(\$0)
\$s2	FF	FF	FF	8C	lb	\$s2, 2(\$0)
\$s3	XX	XX	XX	9B	sb	\$s3, 3(\$0)

Figure 6.22 Instructions for loading and storing bytes



**Figure 6.23** The string “Hello!” stored in memory



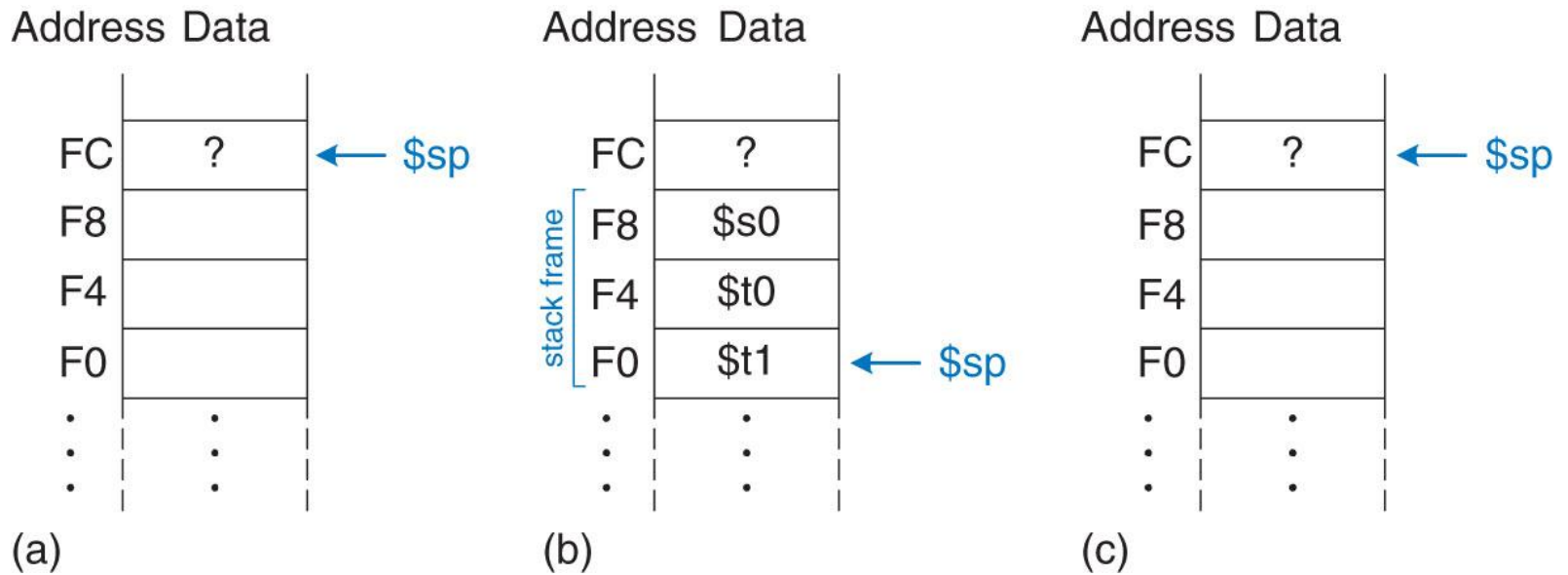
Address	Data
7FFFFFFC	12345678 ← \$sp
7FFFFFF8	
7FFFFFF4	
7FFFFFF0	
⋮	⋮

**(a)**

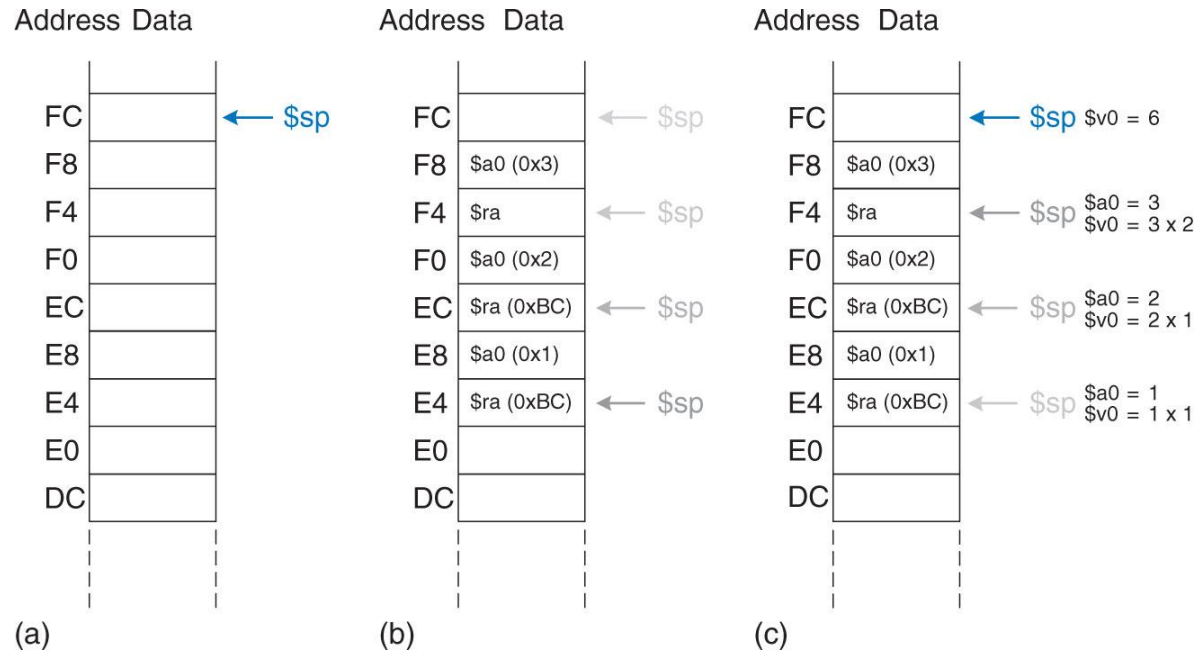
Address	Data
7FFFFFFC	12345678
7FFFFFF8	AABBCCDD
7FFFFFF4	11223344 ← \$sp
7FFFFFF0	
⋮	⋮

**(b)**

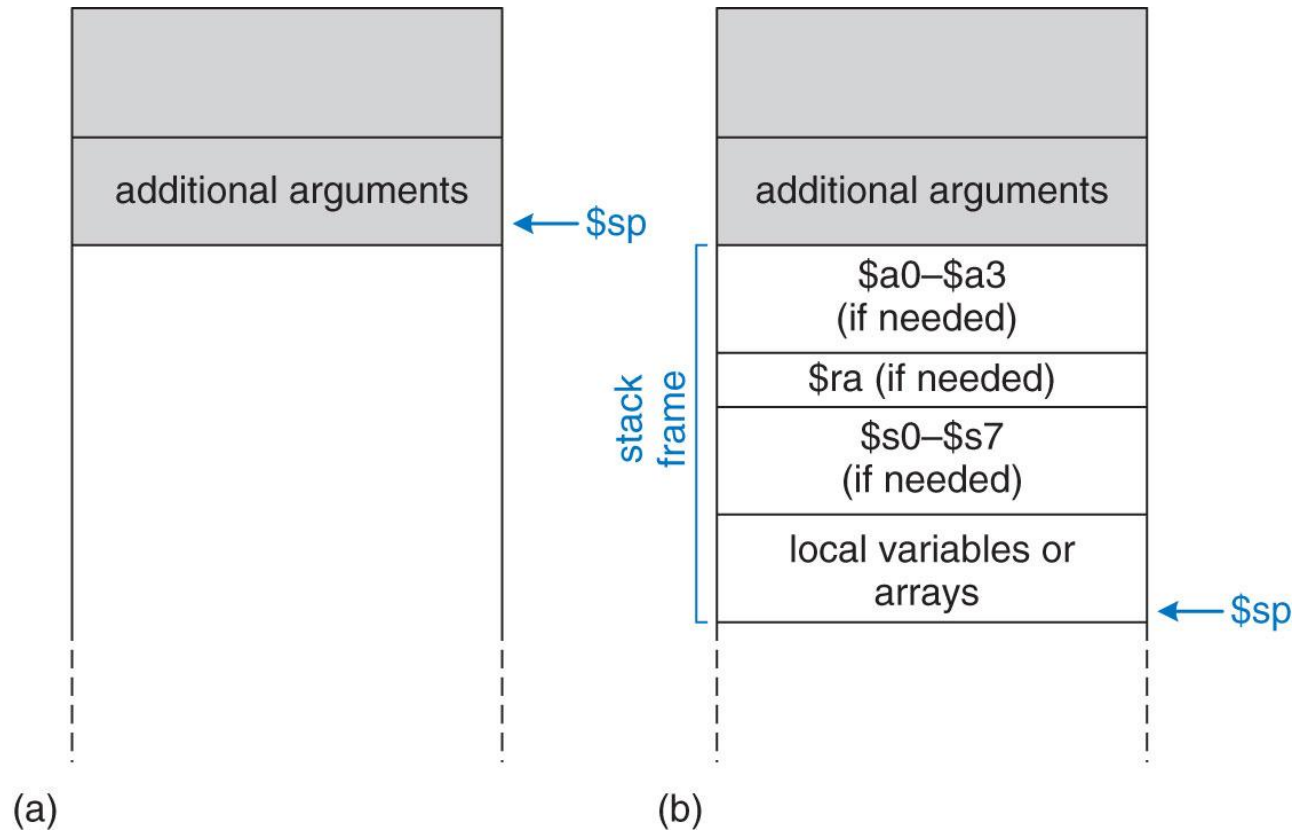
**Figure 6.24 The stack**



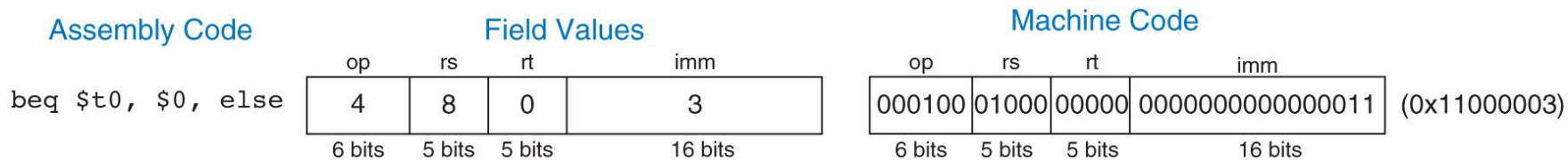
**Figure 6.25** The stack (a) before, (b) during, and (c) after `diffosums` function call



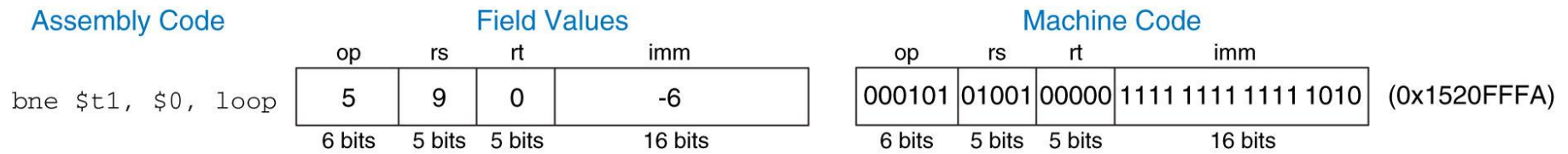
**Figure 6.26 Stack during factorial function call when  $n = 3$ : (a) before call, (b) after last recursive call, (c) after return**



**Figure 6.27 Stack usage: (a) before call, (b) after call**



**Figure 6.28 Machine code for beq**



**Figure 6.29 bne machine code**

## Assembly Code

jal sum

## Field Values

op	addr
3	0x0100028
6 bits	26 bits

## Machine Code

op	addr
000011	00 0001 0000 0000 0000 0010 1000
6 bits	26 bits

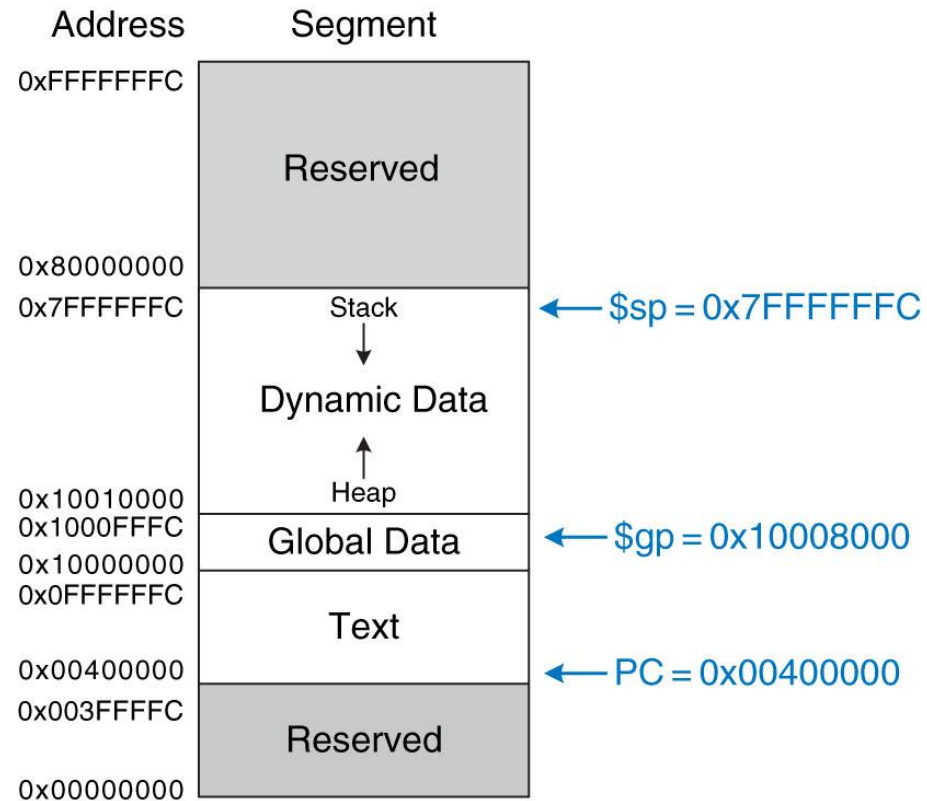
(0x0C100028)

JTA 0000 0000 0100 0000 0000 0000 1010 0000 (0x004000A0)

26-bit addr 0000 0000 0100 0000 0000 0000 1010 0000 (0x0100028)

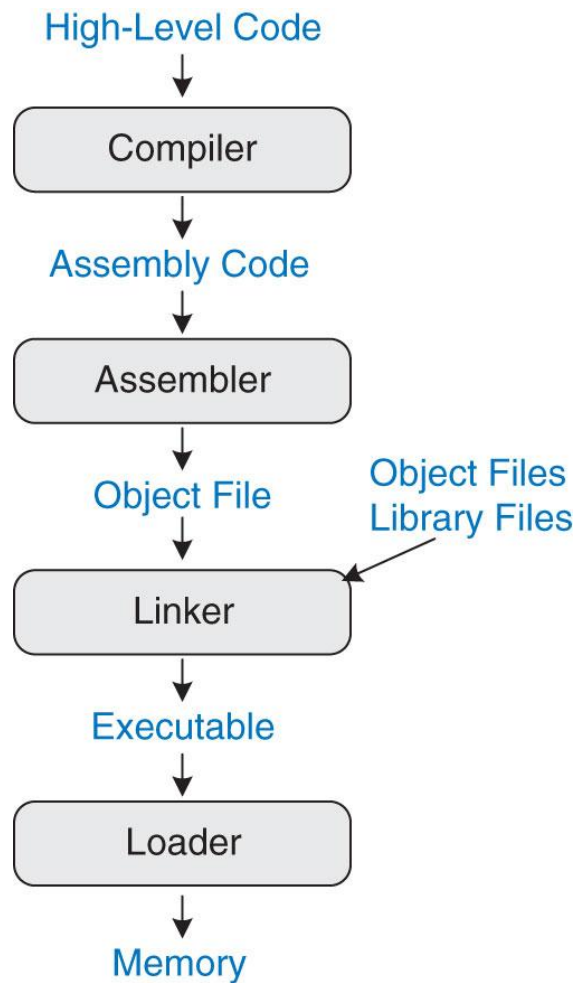
0 1 0 0 0 2 8

**Figure 6.30 jal machine code**



**Figure 6.31 MIPS memory map**





**Figure 6.32 Steps for translating and starting a program**

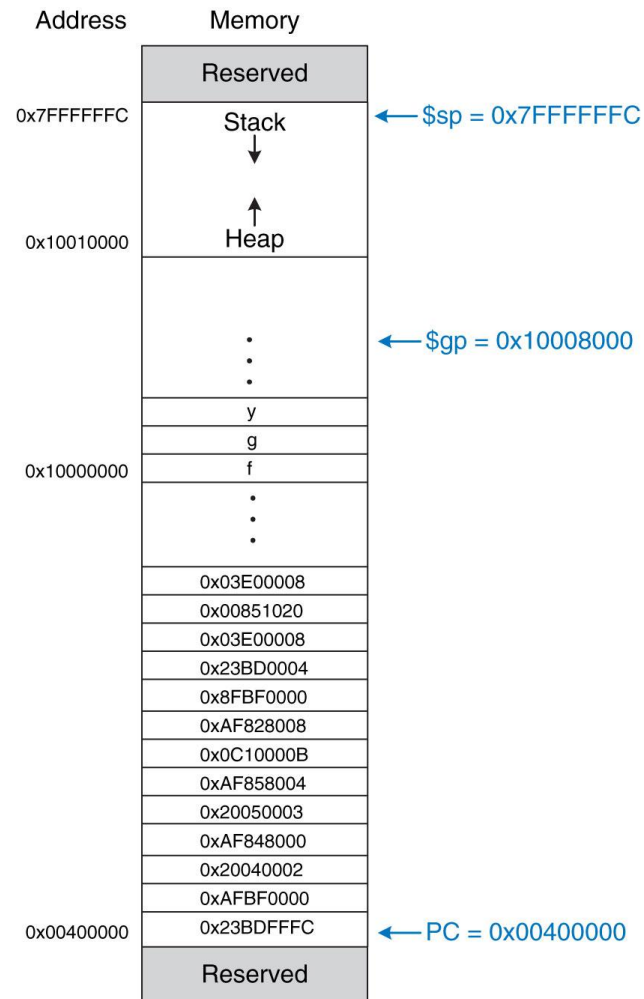
Executable file header	Text Size	Data Size
	0x34 (52 bytes)	0xC (12 bytes)
Text segment	Address	Instruction
	0x00400000	0x23BDFFFC
	0x00400004	0xAFBF0000
	0x00400008	0x20040002
	0x0040000C	0xAF848000
	0x00400010	0x20050003
	0x00400014	0xAF858004
	0x00400018	0x0C10000B
	0x0040001C	0xAF828008
	0x00400020	0x8FBF0000
	0x00400024	0x23BD0004
	0x00400028	0x03E00008
	0x0040002C	0x00851020
	0x00400030	0x03E00008
Data segment	Address	Data
	0x10000000	f
	0x10000004	g
	0x10000008	y

```

addi $sp, $sp, -4
sw  $ra, 0($sp)
addi $a0, $0, 2
sw  $a0, 0x8000($gp)
addi $a1, $0, 3
sw  $a1, 0x8004($gp)
jal  0x0040002C
sw  $v0, 0x8008($gp)
lw  $ra, 0($sp)
addi $sp, $sp, -4
jr  $ra
add $v0, $a0, $a1
jr  $ra

```

**Figure 6.33 Executable**



**Figure 6.34 Executable loaded in memory**

## F-type

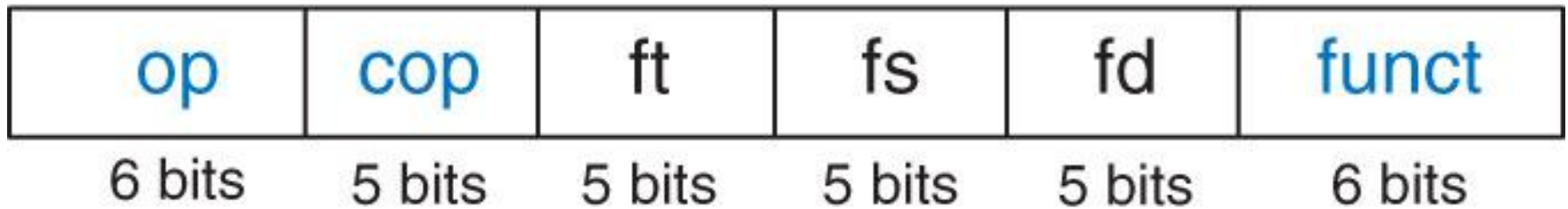
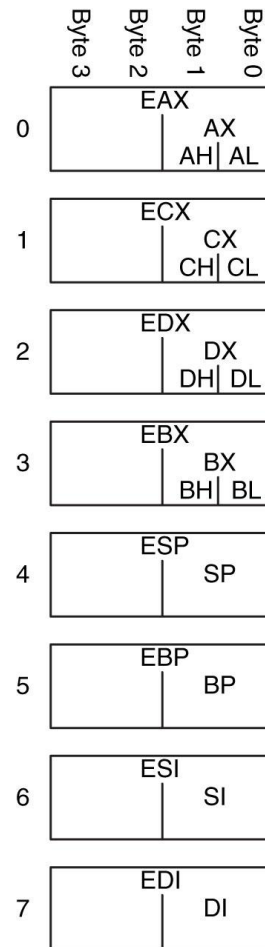
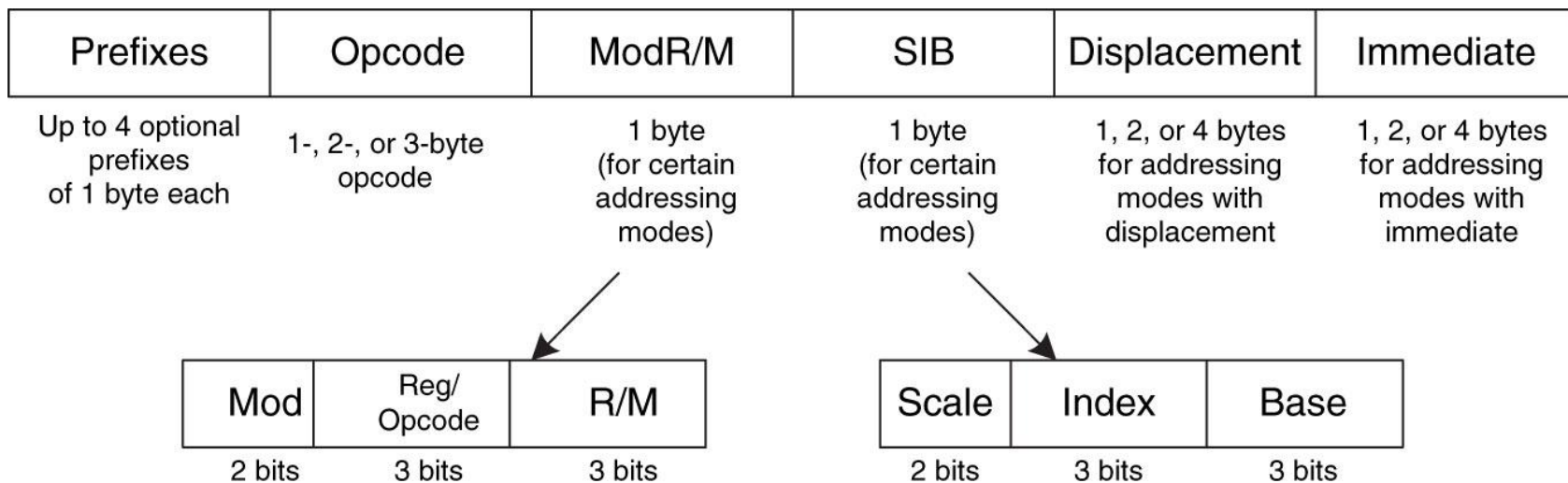


Figure 6.35 F-type machine instruction format



**Figure 6.36 x86 registers**



**Figure 6.37 x86 instruction encodings**

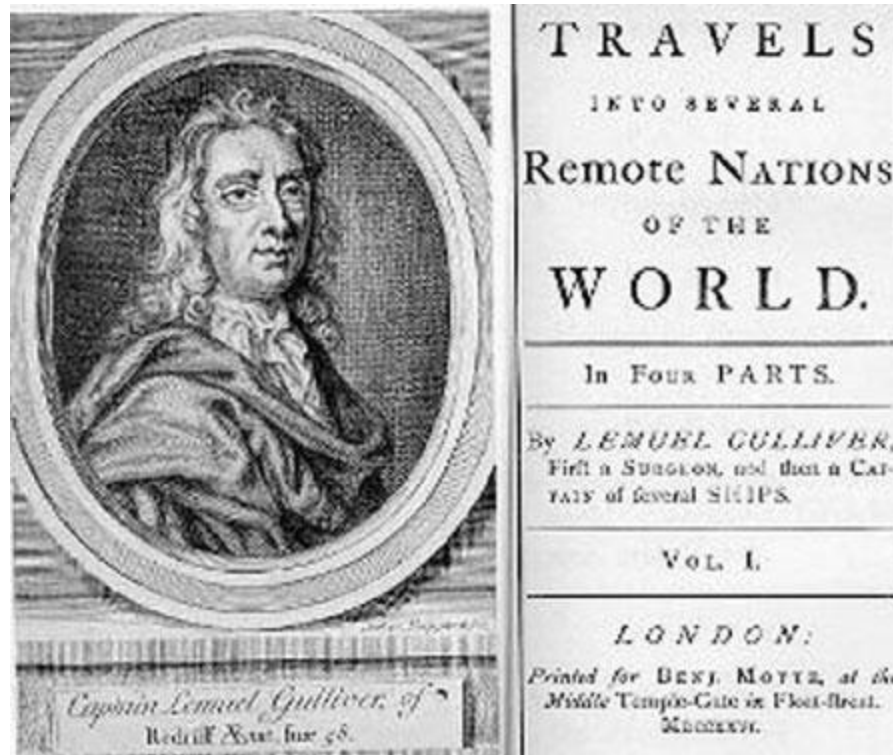


Figure M 01



**Figure M 02**

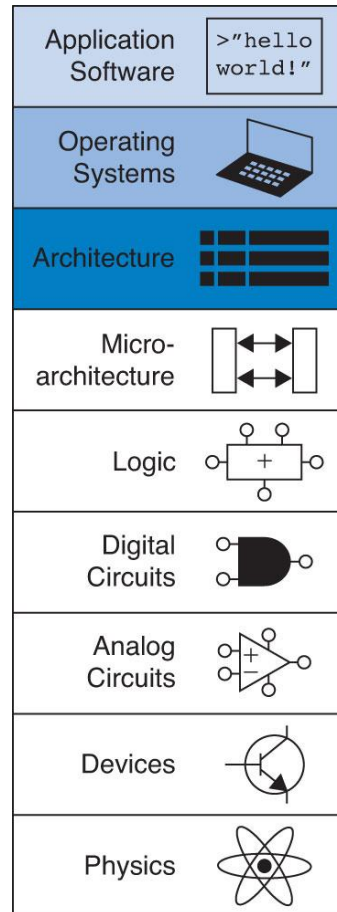




**Figure M 03**



**Figure M 04**



**UNN Figure 1**