

13.1 a. 3 b. 15 c. 24 d. 15 e. 36 f. 36

- 13.3 a. the address field  
 b. the memory location 96  
 c. the memory location whose address is in memory location 96  
 d. register 96  
 e. the memory location whose address is in register 96

13.4		<table><tr><th>EA</th><th>Operand</th></tr><tr><td>300</td><td>900</td></tr><tr><td>101</td><td>300</td></tr><tr><td>900</td><td>1500</td></tr><tr><td>402</td><td>1002</td></tr></table>	EA	Operand	300	900	101	300	900	1500	402	1002		<table><tr><th>EA</th><th>Operand</th></tr><tr><td>400</td><td>1000</td></tr><tr><td>R1</td><td>200</td></tr><tr><td>200</td><td>800</td></tr><tr><td>200</td><td>800</td></tr></table>	EA	Operand	400	1000	R1	200	200	800	200	800
	EA	Operand																						
	300	900																						
	101	300																						
	900	1500																						
402	1002																							
EA	Operand																							
400	1000																							
R1	200																							
200	800																							
200	800																							
	a.		e.																					
	b.		f.																					
	c.		g.																					
	d.		h.																					

The autoindexing with increment is the same as the register indirect mode except that R1 is incremented to 201 after the execution of the instruction.

14.1 a. 00100100  
 00010001  
 00110101

**Carry = 0; Zero = 0; Overflow = 0; Sign = 0; Even parity = 1; Half-carry = 0.**

Even parity indicates that there is an even number of 1s in the result. The Half-Carry flag is used in the addition of packed decimal numbers. When a carry takes place out of the lower-order digit (lower-order 4 bits), this flag is set. See problem 10.1.

b. 11111110  
 00000011  
 10000001

**Carry = 1; Zero = 0; Overflow = 1; Sign = 0; Even Parity = 1; Half-Carry = 1.**

- 14.2 To perform  $A - B$ , the ALU takes the two's complement of B and adds it to A:

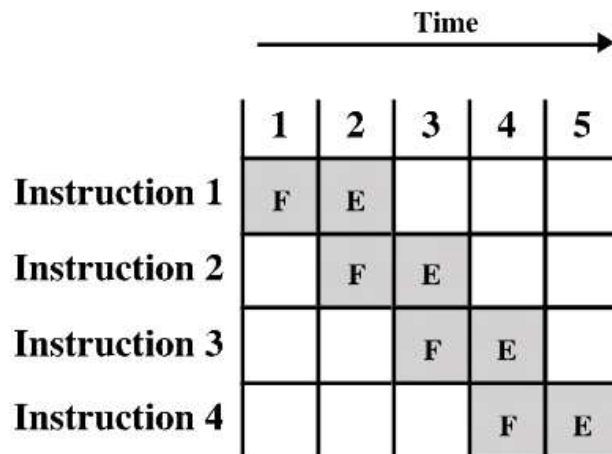
```

A:    11001100
B̄ + 1: +11001101
      10010001
  
```

**Carry = 1; Zero = 0; Overflow = 0; Sign = 1; Even parity = 0; Half-carry = 1.**

- 14.3 a. 8 GHz  
 b. 0.875 ns

14.7



This diagram distorts the true picture. The execute stage will be much longer than the fetch stage.

14.8

	1	2	3	4	5	6	7	8	9	10
I1	FI	DA	FO	EX						
I2		FI	DA	FO	EX					
I3			FI	DA	FO	EX				
I4				FI	DA	FO				
I5					FI	DA				
I6						FI				
I15							FI	DA	FO	EX

**14.9 a.** We can ignore the initial filling up of the pipeline and the final emptying of the pipeline, because this involves only a few instructions out of 3 million instructions. Therefore the speedup is a factor of four.

**b.** Two instructions are completed per clock cycle, for an throughput of 10,000 MIPS.

14.10 a. Speedup :  $9 \cdot 4 \cdot n / 10(4 + n - 1) \Rightarrow 3.6$

b. Non-pipelined processor : 5 Cycles Per an Instruction :  $10\text{GHz}/5 \Rightarrow 2000\text{MIPS}$ . Pipelined processor : Approximately one output per each clock cycle:  $9\text{GHz} \Rightarrow 9000 \text{ MIPS}$

**14.11** The number of instructions causing branches to take place is  $pqn$ , and the number that does not cause a branch is  $(1 - pq)n$ . As a good approximation, we can replace Equation (14.1) with:

$$T_k = pqnk\tau + (1 - pq)[k + (n - 1)]\tau$$

Equation (14.2) then becomes

$$S_k = \frac{T_1}{T_k} = \frac{nk\tau}{(pq)nk\tau + (1 - pq)[k + (n - 1)]\tau} = \frac{nk}{(pq)nk + (1 - pq)[k + (n - 1)]}$$