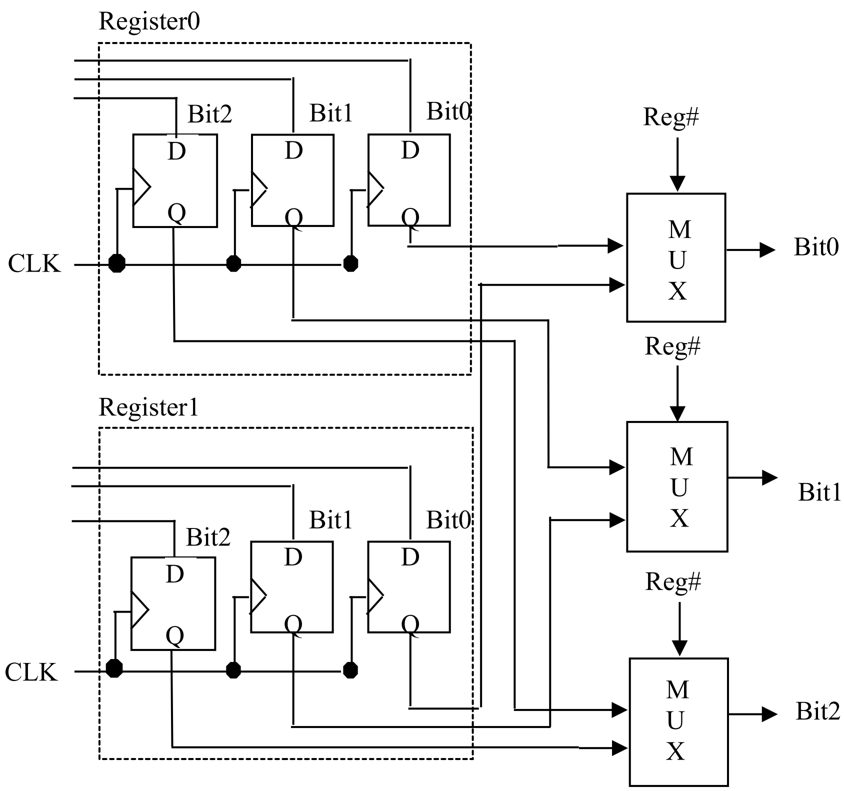
CS151B/EE116C Spring 2017 – Solutions to Homework #1

Problem (1)



Problem (2)

The base address of `b:3880220`, in hexadecimal, is `0x003B351C`.

```
lui   $2,   0x003B
ori   $2,   $2,   0x351C      #$2 = b
add   $3,   $5,   $8          #$3 = i + j
sll   $3,   $3,   2           #$3 = 4(i + j)
add   $3,   $2,   $3          #$3 = b + $3 = b + 4(i + j)
lw    $12,  0($3)             #$12 = value at loc. b + 4(i + j) = b[i+j]
sub   $12,  $12,  $13         #$12 = $12 - x = b[i+j] - x
sll   $5,   $5,   2           #$5 = 4 * $5 = 4i
sub   $2,   $2,   $5          #$2 = b - 4i = b[-i]
sw    $12,  48($2)            #b[12-i] = b[i+j] - x
```

Problem (3)

```
sll   $12, $12, 6
srl   $12, $12, 6     # bits 26-31 of $12 are now 0
srl   $2,  $11, 11
sll   $2,  $2,  26    # bits 26-31 of $2 <- bits 11-16 of $11
or    $12, $12, $2
```

Problem (4)

`414` decimal = `0x019E`

In a Big Endian machine, memory contains:

24	25	26	27
01	9E	00	00

We then obtain `$1 = 0x9E` and `$2 = 0x00.`

In a Little Endian machine, memory contains:

27	26	25	24
01	9E	00	00

`We then obtain $1 = 0x00 and $2 = 0x9E`

Problem (5)

op						rs					rt					offset															
1	0	1	0	0	1	1	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0

```
op = 0x29 = 101001          rt     = $4 = 00100
rs = $27  = 11011           offset = 36 = 100100
```

Problem (6)

address	op	rs	rt	rd	shamt	func	opcode
0000 0001 1100 0100	001000	00000	01111	0000 0000 0010 0001			addi
0000 0001 1100 1000	000000	00000	00101	00100	00010	000000	sll
0000 0001 1100 1100	000000	01011	00100	00100	00000	100000	add
0000 0001 1101 0000	100011	00100	00100	0000 0000 0000 0000			lw
0000 0001 1101 0100	000100	00100	01111	0000 0000 0000 0111			beq
0000 0001 1101 1000	001000	00101	00101	0000 0000 0000 0001			addi
0000 0001 1101 1100	000000	00111	00100	00011	00000	101010	slt
0000 0001 1110 0000	000100	00011	00000	0000 0000 0000 0010			beq
0000 0001 1110 0100	000000	01100	00100	01100	00000	100000	add
0000 0001 1110 1000	000010	00 0000 0000 0000 0000 0111 0010					j
0000 0001 1110 1100	000000	01100	00111	01100	00000	100010	sub
0000 0001 1111 0000	000010	00 0000 0000 0000 0000 0111 0010					j

Problem (7)

```
sltu $8, $22, $13     #$8 will be 1 if $22 < $13, 0 if $13 <= $22
xori $8, $8,  1
```

Problem (8)

For explanation purposes, the comments assume that the following values are in memory:

240	241	242	243
AA	BB	CC	DD

`lhwrdu $7, 240($14)` should load the zero-extended half word contained in addresses 240-241. Thus, $7 should contain 00 00 AA BB after execution.

Assume that the register $1 can be used by the assembler for temporary values.

```
lbu  $7, 240($14)     #$7 contains 00 00 00 AA
sll  $7, 7, 8         #$7 contains 00 00 AA 00
lbu  $1, 241($14)     #$1 contains 00 00 00 BB
or   $7, $7, $1
```

Problem (9)

```
addi $7,  $0,  1
sw   $7,  8($0)
lb   $7,  11($0)
sb   $7,  149($0)
```

Big Endian: Memory[11] = 1 after the sw

8	9	10	11
00	00	00	01

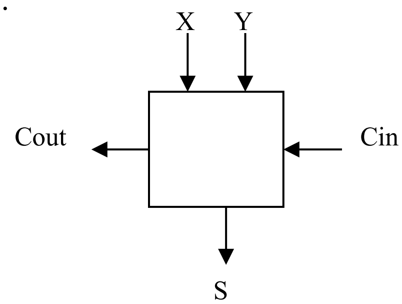
Little Endian: Memory[11] = 0 after the sw

11	10	9	8
00	00	00	01

Solutions to Homework #1 Practice Problems

Problem (10)

A modular 1-bit adder:



Truth table for the 1-bit adder:

X	Y	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Karnaugh map for S:

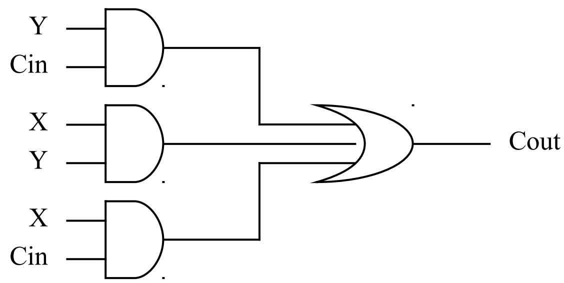
S		YCin			
		00	01	11	10
X	0	0	1	0	1
	1	1	0	1	0

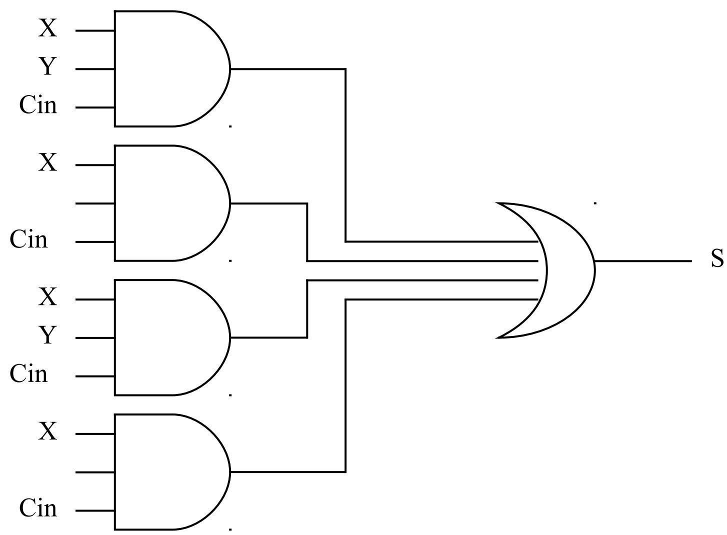
$$S = \overline{X}\,\overline{Y}Cin + \overline{X}Y\overline{Cin} + X\overline{Y}\,\overline{Cin} + XYCin$$

Karnaugh map for Cout:

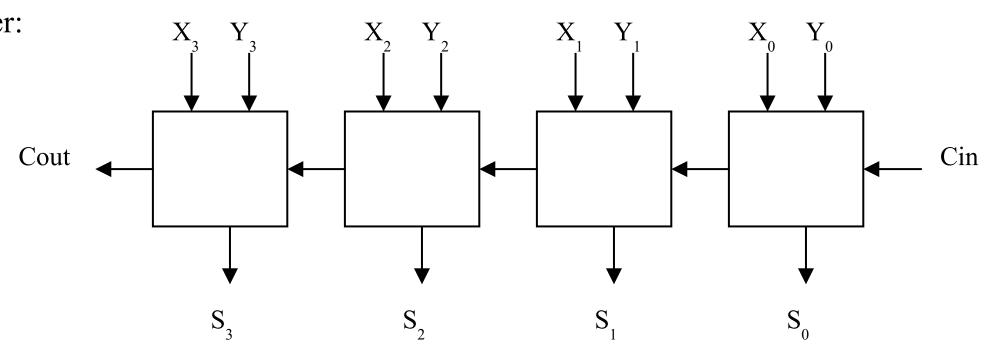
Cout		YCin			
		00	01	11	10
X	0	0	0	1	0
	1	0	1	1	1

Cout = YCin + XY + XCin

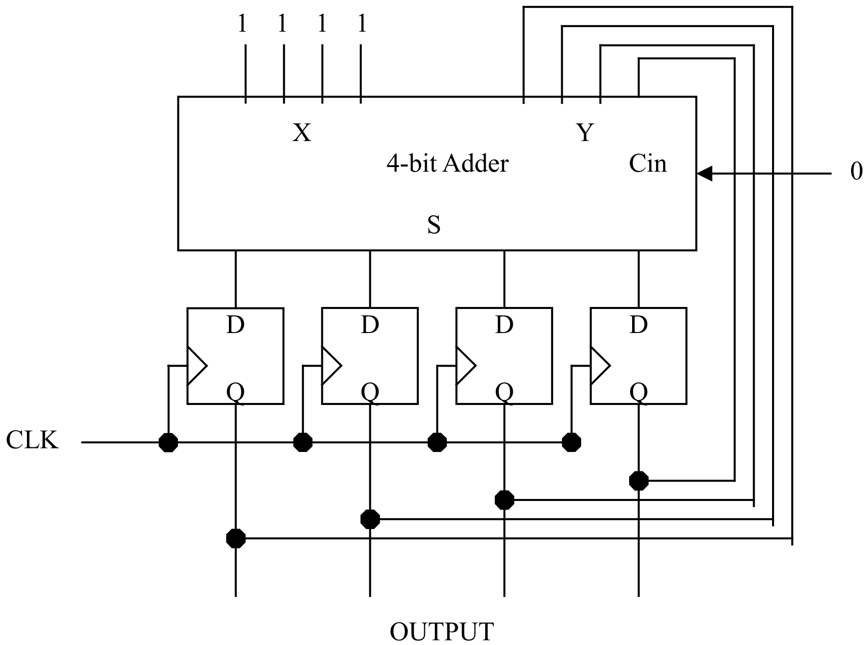




4-bit adder:



Problem (11)



Problem (12)

In binary numbers multiplication by 2 can be accomplished by a left shift of 1 bit if ignoring any overflow. Multiplication by 5 is the same as a multiplication by 4 (left shift of 2 bits) followed by an addition of the multiplicand itself. In MIPS:

```
sll   $17, $16, 2          # $17 = 4 * $16
add   $17, $16, $17        # $17 = $17 + $16
```

Address	Op	Rs	Rt	Rd	Shamt	Func	Inst
10000000	000000	00000	10000	10001	00010	000000	sll
10000100	000000	10000	10001	10001	00000	100000	add

Problem (13)

```
add   $11, $4,  $0
bgez  $4,  1
sub   $11, $0,  $11
```

Problem (14)

Assume that the register $1 can be used by the assembler for temporary values. This matches the standard MIPS register usage conventions (Register $1 is reserved for assembler)

```
slt   $1, $11, $7      # $1=1 if $7>$11, $1=0 otherwise
beq   $1, $0,  3       # don't jump if $7<=$11
lui   $1, 0x3A01       # upper 16 bits of jump dest
ori   $1, $1, 0x5432   # lower 16 bits of jump dest
jr    $1               # jump to the dest
```

Problem (15)

Assume that the register $1 can be used by the assembler for temporary values.

```
lui   $1, 0xDCBA
ori   $1, $1, 0x9886
lbu   $1, 0($1)
sb    $1, 0x6ADB($0)
```

Problem (16)

Assume that the register $1 can be used by the assembler for temporary values.

```
sll   $8,   $3,   16
srl   $1,   $3,   16
or    $8,   $8,   $1
```

Problem (17)

101 000	10110	00101	0000 0000 0000 1100
store byte	rs	rt	address/immediate

So the instruction is

```
sb    $5, 12($22)
```

which stores the low byte of register $5 into Mem[R[$22]+12].
State changes:

```
PC ✶ PC + 4;
Mem[0x0003002E]byte ✶ 0x82
```

Problem (18)

Address	Op	Rs	Rt	Rd	Shamt	Func	Inst
10001000	000000	00000	10011	01001	00010	000000	sll
10001100	000000	01001	10110	01001	00000	100000	add
10010000	100011	01001	01000	0000 0000 0000 0000			lw
10010100	000101	01000	10101	0000 0000 0000 0010			bne
10011000	001000	10011	10011	0000 0000 0000 0001			addi
10011100	000010	0000 0000 0000 0000 0000 1000 10					j