CS47 - Lecture 18

Kaushik Patra
(kaushik.patra@sjsu.edu)

1

- Addition
- Subtraction
- Logic Add/Sub in Assembly

Reference Books:

1) *Chapter 3 of 'Computer Organization & Design by Hennessy, Patterson*
2) *Chapter 4 of 'Logic and Computer Design Fundamentals' by Mano, Kime*

Addition Operation ...

2

Binary Addition Process

Binary Two Single Bit Addition Result

Bit 1 (A)	Bit 2 (B)	Sum Bit (Y)	Carry Bit (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half Addition

Example

CI		1		0		1
A		1		0		0
B		1		1		1
	1	1	0	1	1	0
	CO	Y	CO	Y	CO	Y

Binary Three Single Bit Addition Result

Bit 1 (CI) Carry In	Bit 2 (A)	Bit 3 (B)	Sum Bit (Y)	Carry Bit (CO) Carry Out
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

Full Addition

Calculation

1 + 1 + 1 = 10 + 1 = 11

0 + 0 + 1 = 00 + 1 = 01

1 + 0 + 1 = 01 + 1 = 10

3

- Each binary addition of two single bits produces two bits of result – sum bit and carry bit. Addition considering just two input bits is known as half addition process. This is half of the addition process since it does not consider the incoming carry bit from addition of previous bit position.
- Full addition involves two input bits and a carry bit from the addition operation from the previous bit position.
- Binary addition of 1 + 1 + 1 can be derived as following.
  - 1 + 1 + 1 = (1 + 1) + 1 = 10 + 1 = 11

Binary Addition Process

Binary Two Single Bit Addition Result

Bit 1 (A)	Bit 2 (B)	Sum Bit (Y)	Carry Bit (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half Addition

Example

CI		1		0		1
A		1		0		0
B		1		1		1
	1	1	0	1	1	0
	CO	Y	CO	Y	CO	Y

Binary Three Single Bit Addition Result

Bit 1 (CI) Carry In	Bit 2 (A)	Bit 3 (B)	Sum Bit (Y)	Carry Bit (CO) Carry Out
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

Full Addition

Calculation

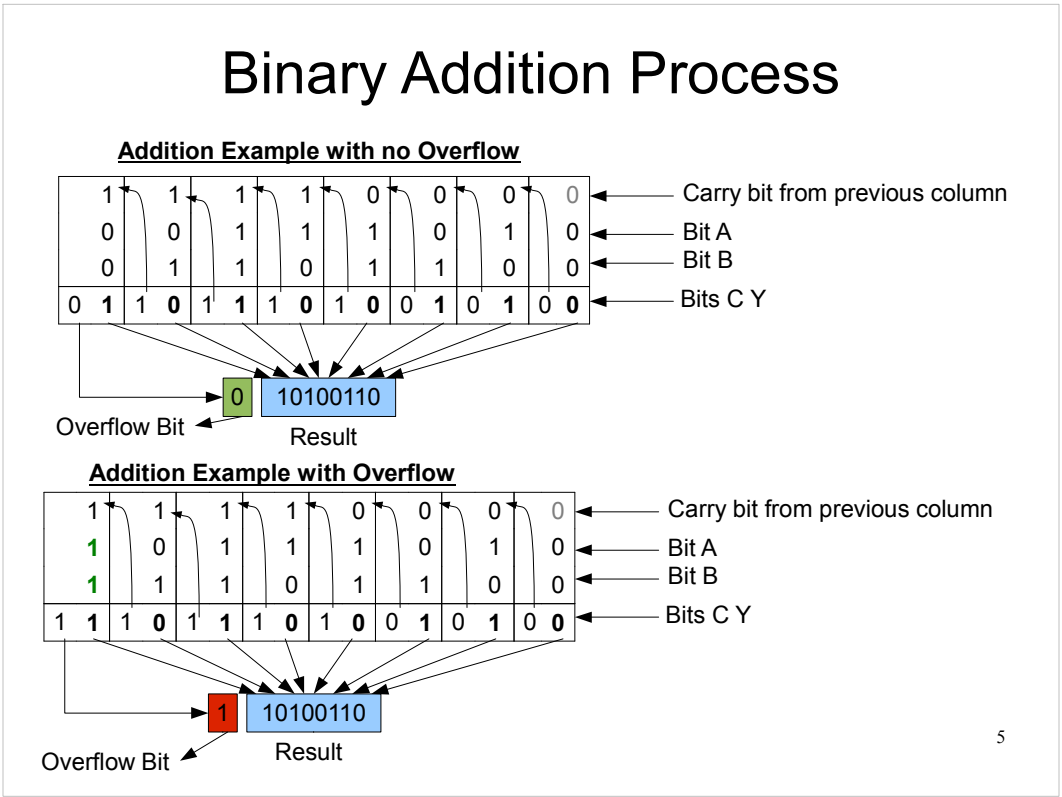
1 + 1 + 1 = 10 + 1 = 11

0 + 0 + 1 = 00 + 1 = 01

1 + 0 + 1 = 01 + 1 = 10

4

- Each binary addition of two single bits produces two bits of result – sum bit and carry bit. Addition considering just two input bits is known as half addition process. This is half of the addition process since it does not consider the incoming carry bit from addition of previous bit position.
- Full addition involves two input bits and a carry bit from the addition operation from the previous bit position.
- Binary addition of 1 + 1 + 1 can be derived as following.
  - 1 + 1 + 1 = (1 + 1) + 1 = 10 + 1 = 11



- We assume a carry in bit of 0 for the addition of first bit position. A full addition is performed starting from the 1st bit position and the corresponding carry bit is used as carry in bit for the successive higher bit position.

- Addition operation may result in overflow. A overflow condition is a condition when the sum of two numbers can not be represented with the fixed number of bits. For example if we represent unsigned number in 4 bits then 1111 + 1 can not be represented with 4 bits (since the sum results in 10000, which is 5 bit long). For a signed representation with 4 bits 0111 + 1 will result in overflow since the sum is 1000, which is well within the bit range (4 bit), but since 1000 represents -ve number (-8 in decimal) the actual sum result +8 can not be represented with 4-bit. Thus an overflow condition occurs.

- For a n-bit number represented in 2's complement, n^{th} bit is the sign bit. Hence any addition of two +ve numbers resulting altering the nth bit will be an overflow condition.

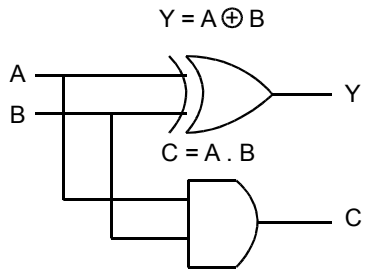
- For n-bit unsigned numbers, if the n^{th} stage addition causes a carry out value 1, it is an overflow condition.

Half Adder

Binary Two Single Bit Addition Result

Bit 1 (A)	Bit 2 (B)	Sum Bit (Y)	Carry Bit (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half Addition



6

- A half adder implements the half addition involving two input bits only. No carry in bit is considered. It results in one sum bit and one carryout bit.

- The sum bit is logic is an ex-OR operation between two input bit and the the carry out bit is AND operation between two input bits.

Full Adder

Binary Three Single Bit Addition Result

	Bit 1 (CI) Carry In	Bit 2 (A)	Bit 3 (B)	Sum Bit (Y)	Carry Bit (CO) Carry Out
m0	0	0	0	0	0
m1	0	0	1	1	0
m2	0	1	0	1	0
m3	0	1	1	0	1
m4	1	0	0	1	0
m5	1	0	1	0	1
m6	1	1	0	0	1
m7	1	1	1	1	1

Full Addition

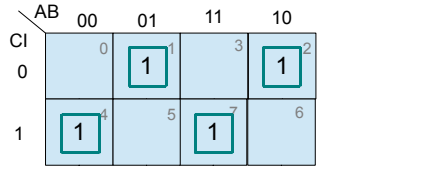
$Y = \Sigma m(1,2,4,7)$

$CO = \Sigma m(3,5,6,7)$

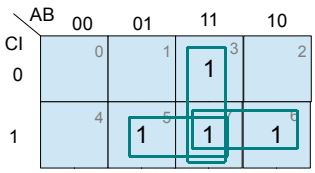
7

- For a full addition implementation, we uses a traditional way of combinational circuit implementation.
- For each of the output, we identify the minterm which causes output to be 1 and express the output equation in POS format.

Full Adder



Y = Σm(1,2,4,7)
(1) = CI.A'B' + CI'.A'.B + CI.A.B + CI'.A.B'
(2) = CI'.(A'.B + A.B') + CI.(A.B + A'.B')
(3) = CI'.(A ⊕ B) + CI. (A ⊕ B)'
(4) = CI ⊕ A ⊕ B



CO = Σm(3,5,6,7)
(1) = CI.B + CI.A + A.B
(2) = CI. (A + B) + A.B
(3) = CI.(A'.B+A.B') + A.B
(4) = CI.(A ⊕ B) + A.B

8

- Once POS format of the output equation is determined, we uses K-map to reduce the equation. However, we do not reduce them to full extent deliberately to reuse some of the intermediate results in both of the output equations.

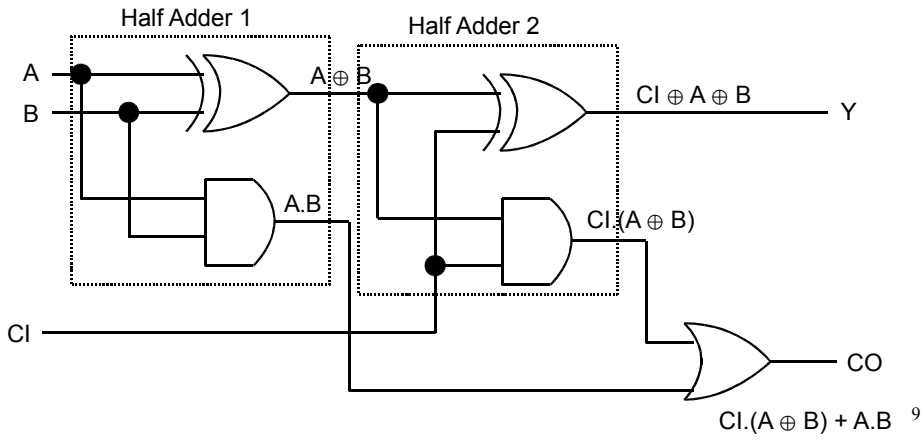
- For output Y determination, step-2 has a term (AB + A'B') which is reduced to (A ⊕ B)' in step-3. (AB + A'B') is true when both A, B are 1 or 0. This means (A'B + AB') is false if (AB + A'B') is true. Hence this is equivalent to express (AB + A'B') as (A'B + AB')', which is (A ⊕ B)'.

- For output CO determination, step-2 has a term CI.(A + B) is reduced (technically expanded) to CI.(A'B + AB'). This is not very intuitive. In this case, instead of combining minterms (5,7) we take the minterm 5. On the other hand, combining minterms (7,6), we take the minterm 6. The minterm 7 is already covered by the minterm combination (3,7). This has been done to reuse (A ⊕ B) computed in Y.

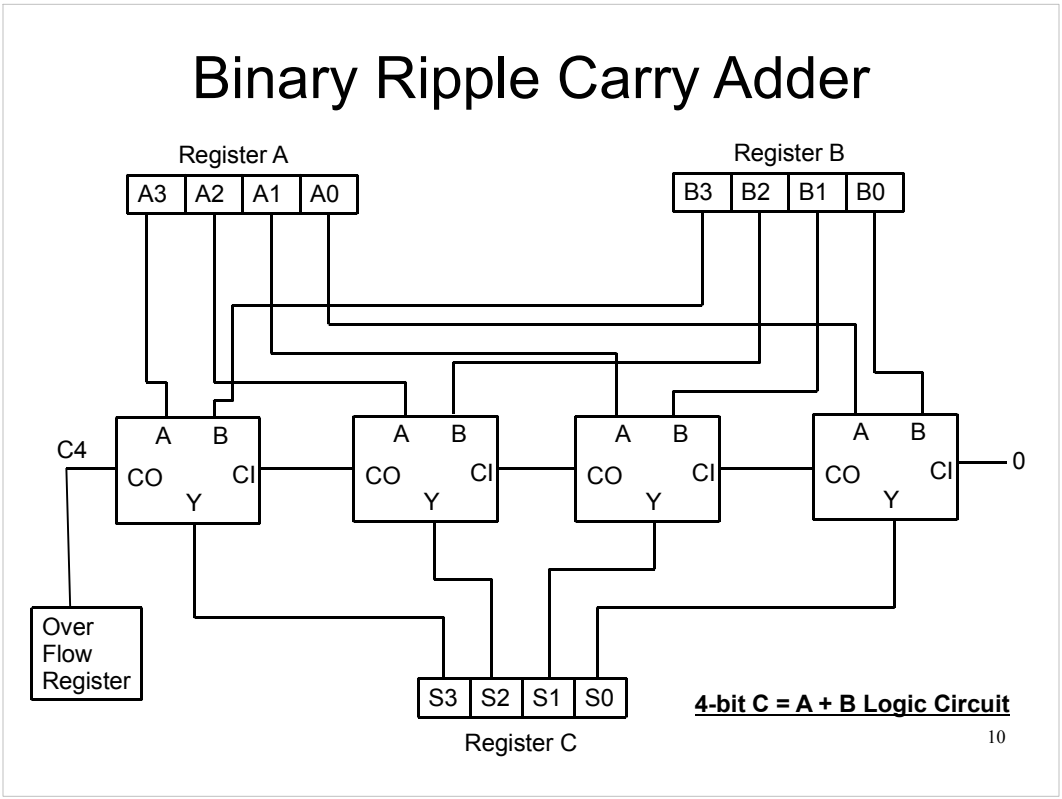
Full Adder

$Y = CI \oplus (A \oplus B)$

$CO = CI.(A \oplus B) + A.B$



- Once the equation is determined, digital implementation is done using half-adder, since half adder gives intermediate terms of the equation. We can also think of (CI + A + B) addition operation as (A + B) + CI. (A+B) is performed using one half-adder. Then CI + (A+B) is performed by another half adder. The two carry out from the two half-adders are then combined using OR gate to get the final carry out.



- Multiple single bit full-adders can be stitched together to form a complete adder for multi-bit numbers. For a n-bit adder we stitch n one bit full-adders. Carry out bit from one bit position adder is connected to carry in bit of the next higher bit position adder. The carry in bit for the 1st bit position adder is set to 0. The last carry out bit indicates the overflow in addition operation.

- Since the carry bit is rippled up from lower bit to higher bit position adder, this type of adder is called ripple carry adder. There are faster addition implementation, like carry lookahead adder. However, those speciality adders are beyond scope of computer architecture discussion.

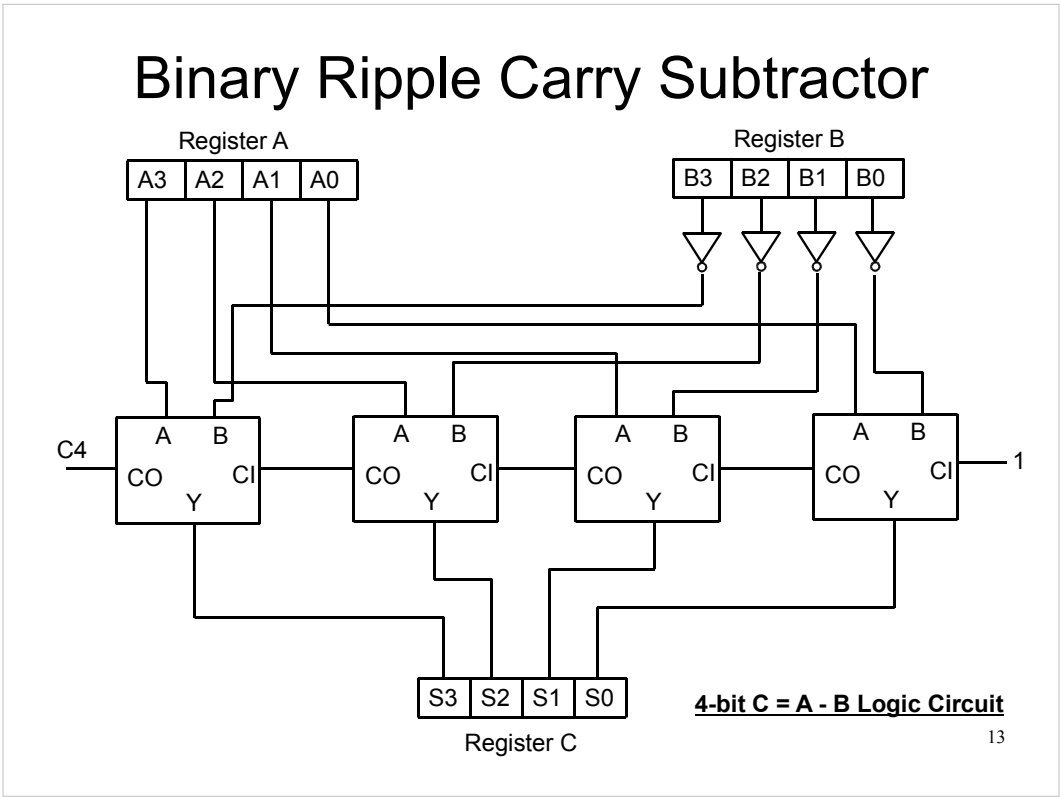
Subtraction Operation ...

Subtraction Using 2's Complement

- Using 2's complement format, there is no need for additional subtraction circuit. We use the following formula.
  - S = A – B = A + INV(B) + 1
    - Both A and B are in 2's complement format.
    - INV(B) is the inversion operation of each individual bit of B which gives 1's complement to B.
    - Addition of 1 to INV(B) will give 2's complement of B.

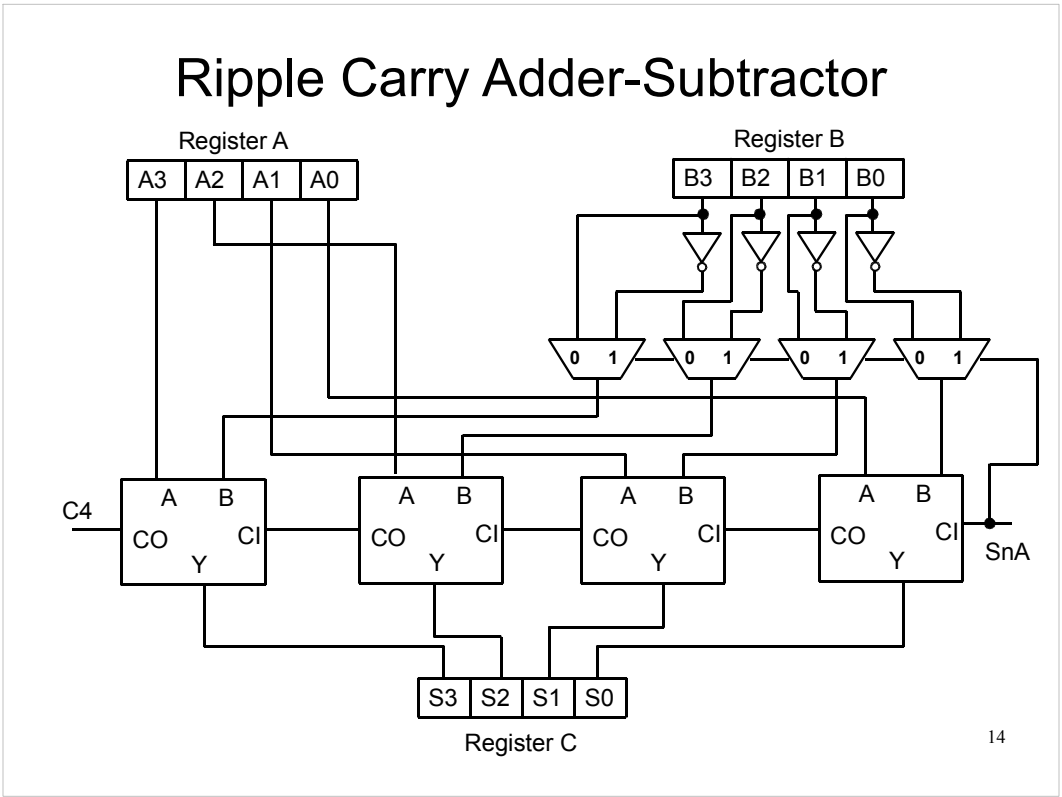
12

- Number representation in 2's complement form gives ability to use adder circuit for subtraction, instead of having dedicated subtractor circuit.



- For a logic circuit implementing C = A -B, where A, B and C are in 2's complement form, we implement C = A + (B' + 1). In 2's complement form, -B is (B' + 1).

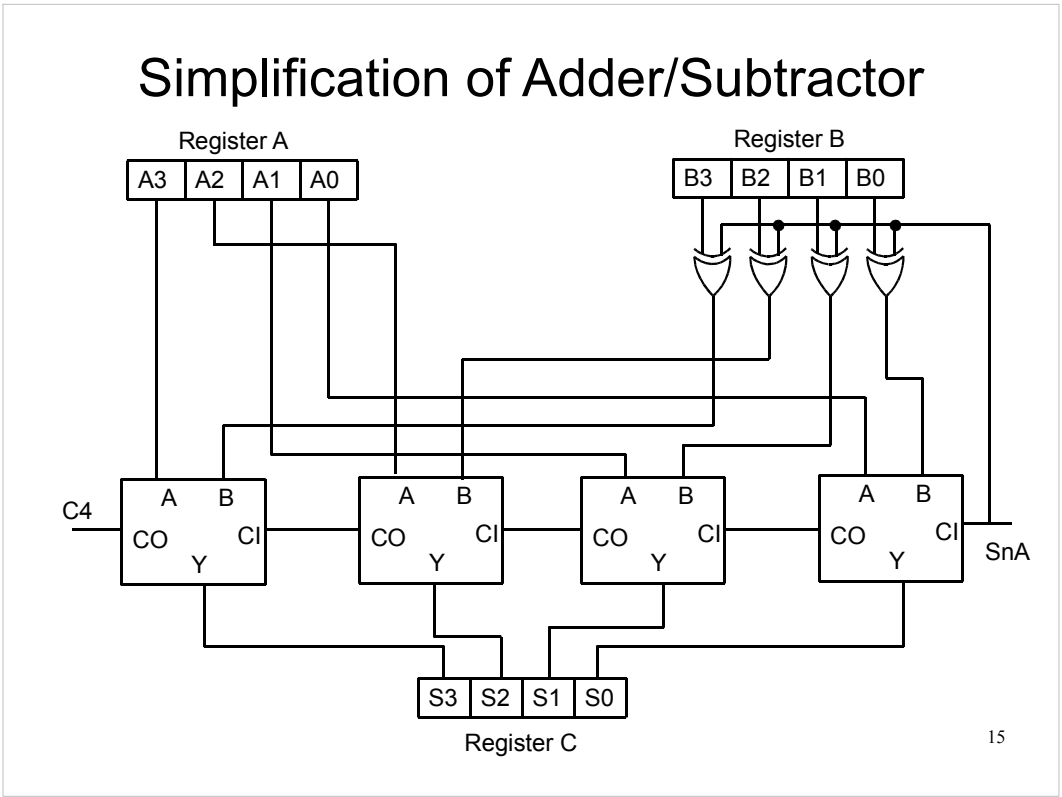
- The term B' is created using NOT gates and the +1 term is created by setting the carry in bit to 1 in the input carry of the 1st bit position adder.



- Careful inspection of ripple carry adder and subtractor reveals more similarities than difference. This gives opportunity to merge these two functionality of addition and subtraction.

- The difference between adder and the subtracter are as following.
  - B is inverted for subtraction, while it is not for addition.
  - Input carry to 1st bit position is 1 incase of subtraction, where it is 0 for addition.

- If there is a control signal SnA, meaning subtraction on SnA = 1 and addition on SnA = 0, then we can connect this 'mode selection' signal into the carry in bit for the 1st bit position adder. We also can use this control signal to select B or B' depending on the addition or subtraction mode setting. This modification will create a generic circuit for addition or subtraction with a control signal SnA.



- We can further eliminate the need of MUX since the choice is either B or B' to be fed into the ripple carry adder circuit. Simple use of XOR gate with one input as the control signal SnA will do the trick. If the controlling signal SnA is set to 0, XOR will pass whatever the other input value is. Hence at SnA=0 term B is fed into the adder. Once SnA = 1, XOR will invert the value of the other input. Hence in this condition of SnA=1, B' will be passed into the adder.

Overflow Detection ...

16

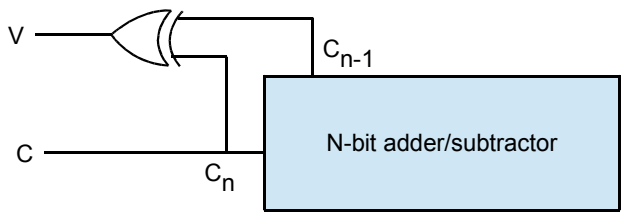
Overflow Condition

- If an n-bit operation results in a quantity that can not be held within a n-bit register, the condition is called a overflow.
- For two unsigned numbers, only addition can create overflow.
- For two signed numbers, overflow can occur with addition if both the numbers are positive or negative.

17

- For unsigned number, only addition can result in a bigger number than the involving operand values. Subtraction result will be either equal or less than that of the operand values. Hence subtraction of unsigned can not produce an overflow condition.
- For signed number representation, overflow can not occur if we add one +ve and one -ve number (since they will create lesser magnitude than the involved operand values). Overflow only can occur if both of the operands are +ve or both of them are -ve (only these conditions can result in higher magnitude than the original operand values).

Overflow Detection



- For signed number, addition overflow will be indicated by n-1th carry bit (C_{n-1}) and subtraction overflow will be indicated by nth carry bit (C_n). The result V will indicate overflow for signed number operation.
- For unsigned number nth carry bit (C_n) will indicate any overflow.

18

- For signed numbers, overflow condition of addition of two positive numbers can be detected by (n-1)th carry bit. If its value is 1, an overflow occurred. For any +ve number of n-bit length, represented in 2's complement format, nth bit is always 0. Hence if there is carry out bit set to 1 at (n-1)th bit position full adder, it will alter the nth bit to 1, which no longer represents a +ve number. However addition of two +ve number should always give a +ve number. Hence we determine a overflow condition.

- Similarly, addition of two -ve numbers will cause overflow if the result can not be held within n bits. Any such subtraction operation will result a carry out value of 1 from the nth bit position of the full adder and it should also have a carry bit out of the n-1th position of the full adder. If it does not, the resultant number becomes +ve, which is clearly a wrong result. Hence detecting absence of n-1th carry-bit, but present of nth carry bit will detect overflow in addition of two -ve numbers represented in 2's complement format.

- It can also be observed that overflow test condition for +ve number addition and -ve number addition are mutually exclusive to each other. Hence if we connect these two carry out bits C_n and C_{n-1} through XOR gate, we'll have overflow detection bit V for signed operation.

- For unsigned operation, as discussed earlier, detecting Cn is sufficient to detect overflow condition.

Logical Add / Sub in Assembly ...

19

Write Utility Macros

- Macro '*extract_nth_bit*' to extract nth bit from a bit pattern.
  - .macro extract_nth_bit($regD, $regS, $regT)
    - $regD : will contain 0x0 or 0x1 depending on nth bit being 0 or 1
    - $regS: Source bit pattern
    - $regT: Bit position n (0-31)
  - Example usage : extact_nth_bit($t0, $s1, $t1)
    - $t1 contains bit position
    - $s1 is the bit pattern
    - $t0 will be 0x0 or 0x1

20

- You need to rigth shift number in $regS by $regT and mask it for 1st bit position value only. Assign this masked result to $regD.

Write Utility Macros

- Macro '*insert_to_nth_bit*' to insert bit 1 at nth bit to a bit pattern.
  - .macro insert_one_to_nth_bit ($regD, $regS, $regT, $maskReg)
    - $regD : This the bit pattern in which 1 to be inserted at nth position
    - $regS: Value n, from which poistion the bit to be extracted (0-31)
    - $regT: Register that contains 0x1 or 0x0 (bit value to insert)
    - $maskReg: Register to hold temporary mask
  - Example usage : insert_to_nth_bit($t0, $s1, $t1, $t9)
    - $t1 contains 0x1
    - $s1 is bit position n
    - $t0 bit pattern will be inserted with 1 at nth position

21

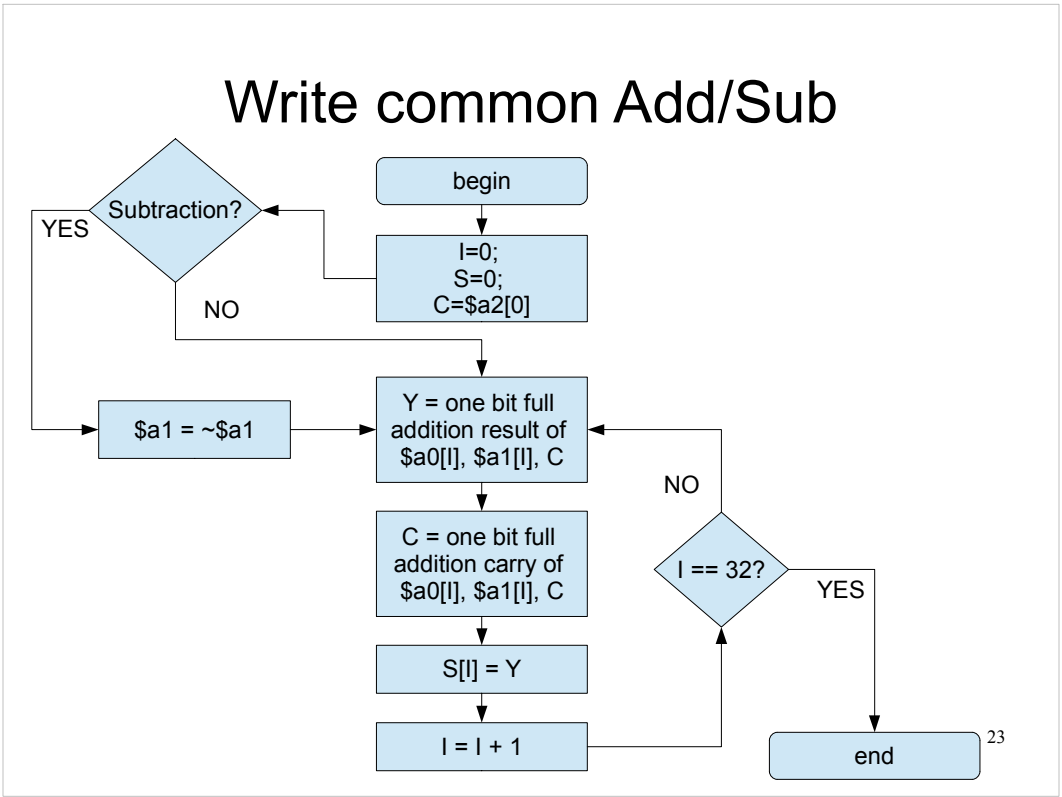
- Prepare a mask in $maskReg by shifting 0x1 for $regS amount and then inverting it. Mask $regD with $maskReg. Now, shift left register $regT by amount in $regS and then logically OR this resultant pattern to $regD to insert the bit at the nth position.

Write common Add/Sub

- Write common procedure add_sub_logical
  - $a0 : First number
  - $a1 : Second number
  - $a2 : Mode
    - 0x00000000 is addition
    - 0xFFFFFFFF is subtraction
  - Returns $a0+$a1 in $v0 in addition mode and $a0-$a1 in $v0 in subtraction mode.
  - About 41 line of code including frame creation and restore.
- You can use 'add' / 'sub' for non-result computing parts
  - To set $s0 to 0 you can use 'add $s0, $zero, $zero'
    - Alternate is to use 'or $s0, $zero, $zero'
  - To increment index you can use 'addi $t1, $t1, 0x1'

22

- Similar to hardware, make a common procedure for addition and subtraction .



- I is the index from 0 to 31, S is the result of the operation, C is the starting Carry bit value for full addition at 1st bit position. Set the starting carry bit as the 1st bit of the mode value in $a2.

- Test $a2 if it is a subtraction operation. If it is, invert the value in $a1.

- For each bit position, compute sum bit using bit at I-th position in $a0 and $a1 (use the extract bit position macro) and the C (the carry) bit. Compute the next carry bit C using C, $a0[I], $a1[I]. Use the logical equation we have derived for one bit full adder. Insert the sum bit (Y) into Ith position of the result (use the insert bit macro).

Use common Add/Sub

- Write procedure add_logical
  - $a0 : First number
  - $a1 : Second number
  - Returns $a0+$a1 in $v0
  - Calls add_sub_logical with $a2 = 0x00000000
- Write procedure sub_logical
  - $a0 : First number
  - $a1 : Second number
  - Returns $a0-$a1 in $v0
  - Calls add_sub_logical with $a2 = 0xFFFFFFFF

24

- Now write separate procedures for add and sub which calls the common add/sub procedure passing the arguments $a0, $a1 with appropriate mode in $a2.

CS47 - Lecture 18

Kaushik Patra
(kaushik.patra@sjsu.edu)

25

- Addition
- Subtraction
- Logic Add/Sub in Assembly

Reference Books:

1) *Chapter 3 of 'Computer Organization & Design by Hennessy, Patterson*
2) *Chapter 4 of 'Logic and Computer Design Fundamentals' by Mano, Kime*