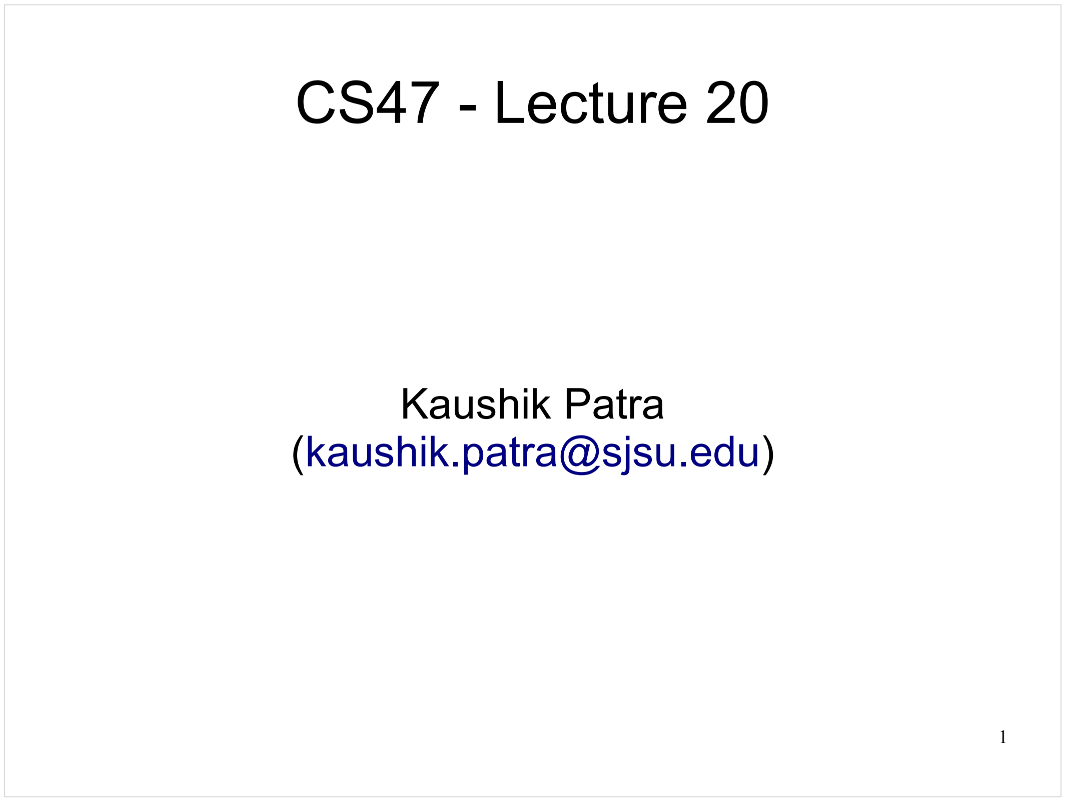


CS47 - Lecture 20

Kaushik Patra
(kaushik.patra@sjsu.edu)

1

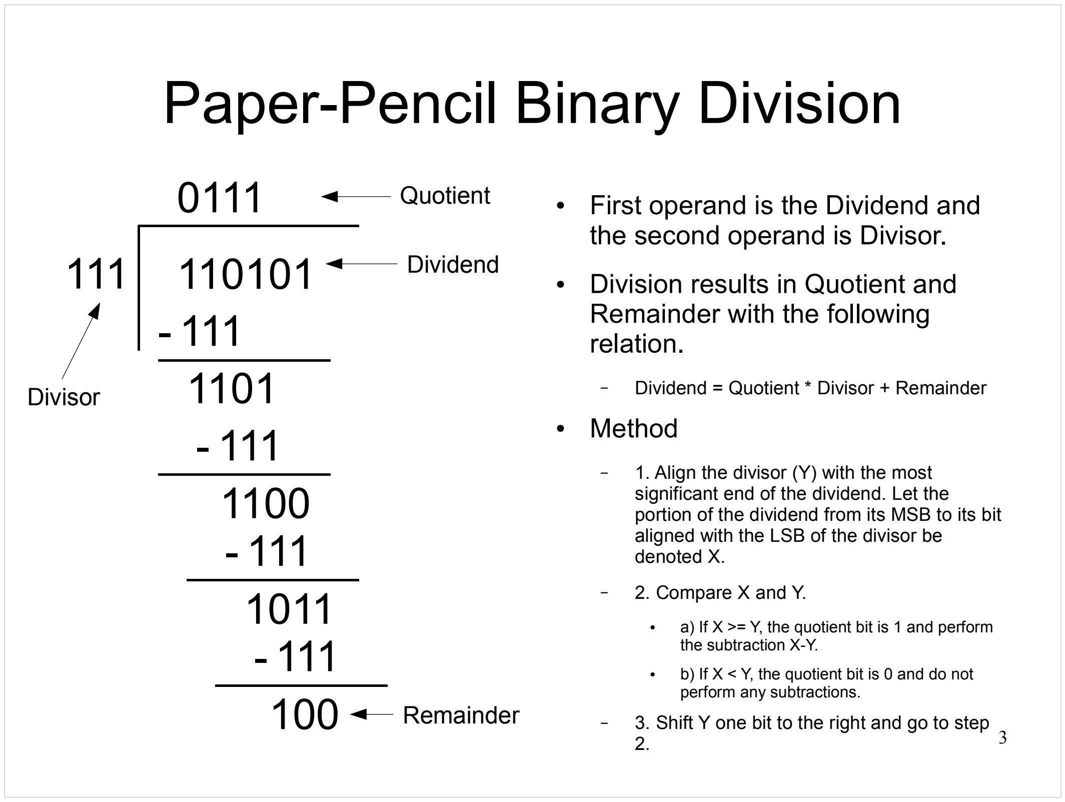
- Division

Reference Books:

1) *Chapter 3 of 'Computer Organization & Design by Hennessy, Patterson*

Division Operation ...

2



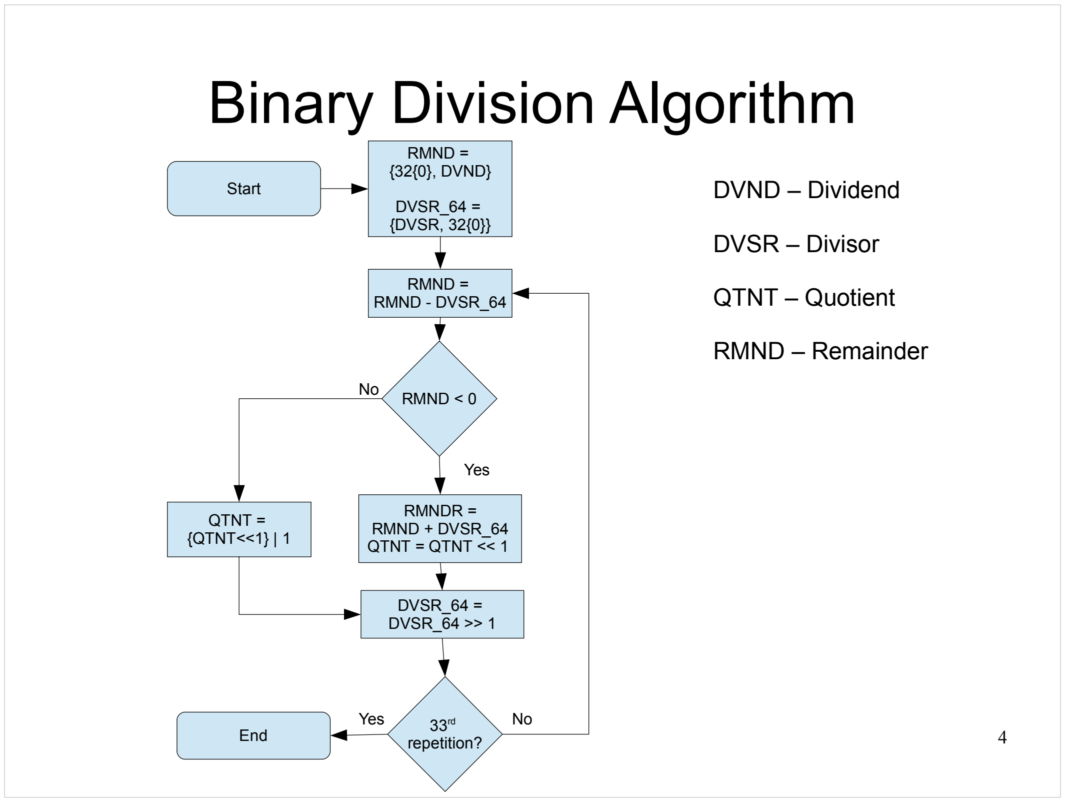
Paper-Pencil Binary Division

```
       0111      <-- Quotient
     ______
111 | 110101     <-- Dividend
    | -111
     ------
      1101
      -111
     ------
       1100
       -111
      ------
        1011
        -111
       ------
         100     <-- Remainder
```

- First operand is the Dividend and the second operand is Divisor.
- Division results in Quotient and Remainder with the following relation.
  - Dividend = Quotient * Divisor + Remainder
- Method
  - 1. Align the divisor (Y) with the most significant end of the dividend. Let the portion of the dividend from its MSB to its bit aligned with the LSB of the divisor be denoted X.
  - 2. Compare X and Y.
    - a) If X >= Y, the quotient bit is 1 and perform the subtraction X-Y.
    - b) If X < Y, the quotient bit is 0 and do not perform any subtractions.
  - 3. Shift Y one bit to the right and go to step 2.

3

- For a paper-pencil method, the divisor is to placed below the dividend at the most significant bit (MSB) side such that the position of MSB of divisor and dividend matches. If at this position part of dividend is greater than divisor, we place 1 in the quotient bit (which grows form left to right – MSB to LSB position) and subtract divisor from the dividend part. We place 0 in the quotient bit otherwise and do not perform the subtraction. We keep on shifting the divisor to cover entire dividend and each step we subtract the divisor (resulting quotient bit 1 for that iteration) or no (resulting quotient bit 0 for that iteration). Once we consume all the dividend bits, we finalize the quotient and the result of the last subtraction is the the remainder of the division operation.



Binary Division Algorithm

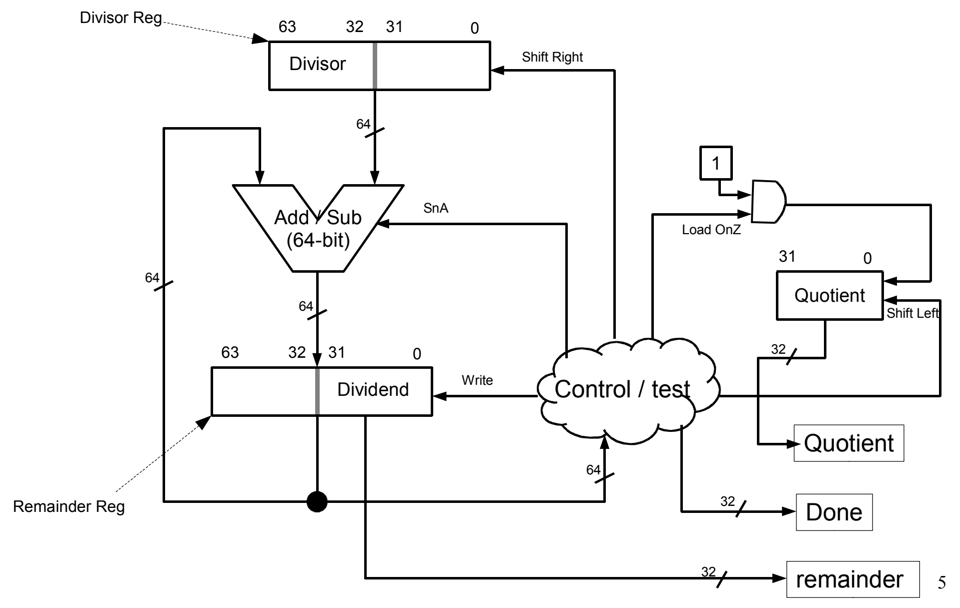
DVND	Dividend
DVSR	Divisor
QTNT	Quotient
RMND	Remainder

4

- As an algorithmic step, we place the 32-bit divisor in the upper half of a 64-bit register and place the dividend in the lower half of the 64-bit remainder register. Once placing them in this manner we do the following steps.

  1) Subtract divisor from remainder.
  2) If the subtraction result is -ve, we add back the divisor and shift quotient register one bit left with 0 inserted in LSB. If the subtraction result is +ve, we keep the result and shift quotient register one bit left with 1 inserted in LSB.
  3) Repeat from step-1 for 32 times.

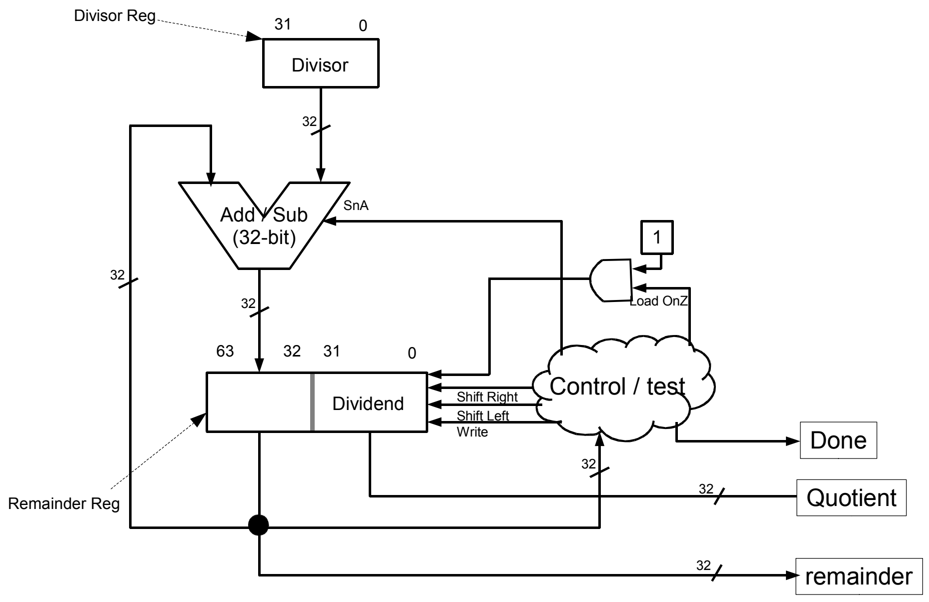
Binary Division Circuit



- This implementation directly reflects the algorithm described in slide 12.
  - For divisor a parallel load, parallel out right shift 64-bit register is used.
  - For remainder a parallel load, parallel out 64-bit register is used. The divisor is loaded into the upper half of the register.
  - For quotient a serial in, parallel out 32-bit left shift register is used. The dividend is loaded into the lower half of the register.
  - A 64-bit adder/ subtractor is used. One input is the divisor register value and other input is the remainder register value.

- The control unit for the division operation first issue right shift for the divisor register. Then subtraction operation is issued and the result in the remainder register is written successively. If the the remainder content is -ve, addition operation is issued and the result is written back in the remainder reg. In this condition, OnZ signal is set to 0 and quotient reg is shifted left for one bit. If the remainder content is positive, no addition operation is issued and the OnZ signal is set to one with shifting the quotient reg one bit to left. This process is done for 32 times and then an optional flag DONE is turned on. The quotient reg contains the quotient and the remainder reg contains the remainder of the division operation at the end of the division operation.

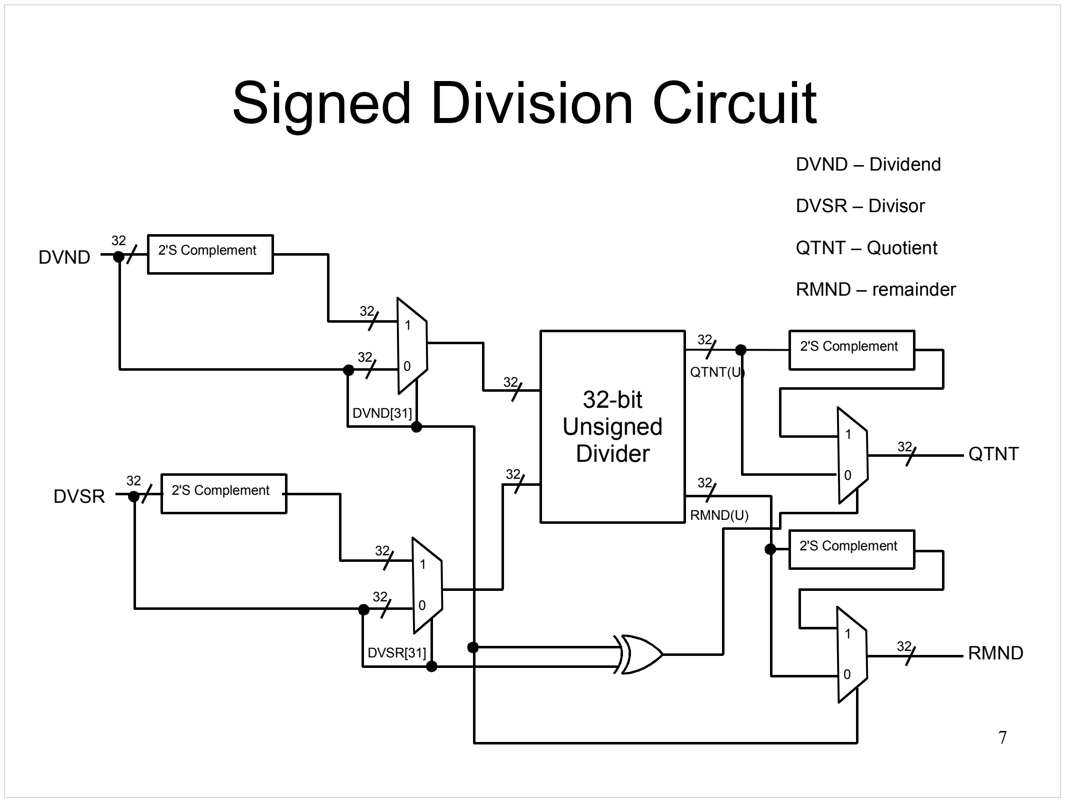
Simplified Binary Division Circuit



6

- There is a scope to further optimize the division circuit by reusing the remainder register to accommodate quotient in steps. We can also modify the operation in a way instead of shifting divisor to right, we can shift the dividend to left. That way the addition / subtraction can be done using 32-bit. In this case divisor register can be a fixed 32-bit register, since it does not need shifting operation.

- The 64-bit register, however, needs to have both left and right shift operation. The controller will shift the dividend to left and perform subtraction. While shifting left, the quotient bit is created at the LSB in the register with 0 (Assuming OnZ is set to 0). Now if the subtraction result is +ve, the register is shifted right one bit and then shifted left to insert 1 a quotient bit. This will fix the result. At the end of the operation the upper half will contain the remainder and the lower half will contain the quotient.



Signed Division Circuit

DVND – Dividend

DVSR – Divisor

QTNT – Quotient

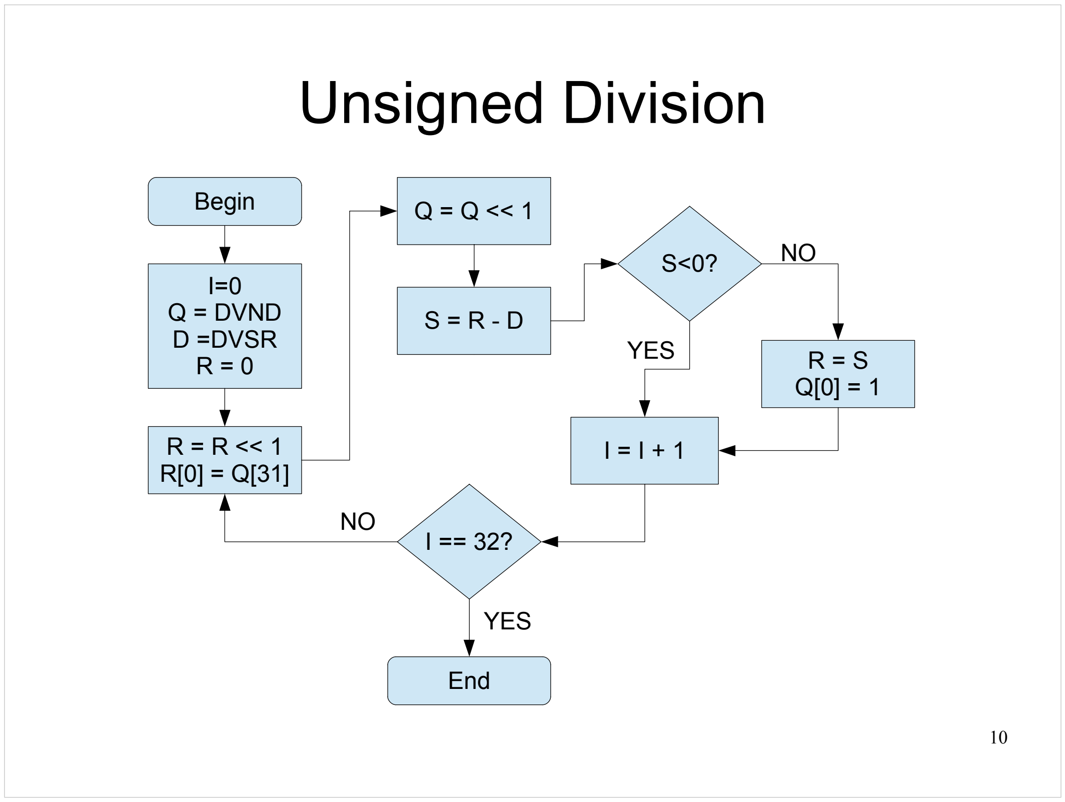
RMND – remainder

- The signed division circuit is similar to signed multiplier circuit. The sign of the quotient bit will be +ve if sign bits are same for dividend and the divisor, it is -ve otherwise. The remainder sign will be same as the dividend sign.

Division in MIPS Assembly ...

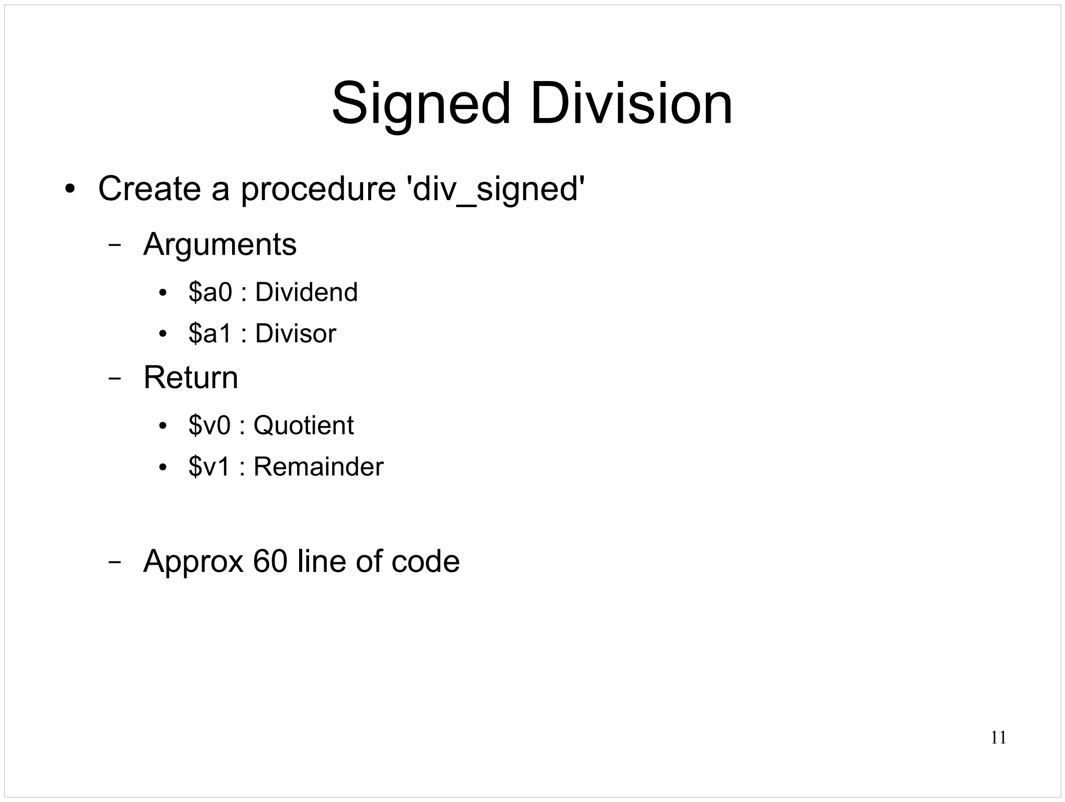
Unsigned Division

- Create a procedure 'div_unsigned'
  - Arguments
    - $a0 : Dividend
    - $a1 : Divisor
  - Return
    - $v0 : Quotient
    - $v1 : Remainder
  - Approx 65 line of code



Unsigned Division

- Use sub_logical for subtraction
- Use extract and insert macro for { R[0] = Q [31]; and Q[0]=1 }
- You can use add to increment I.



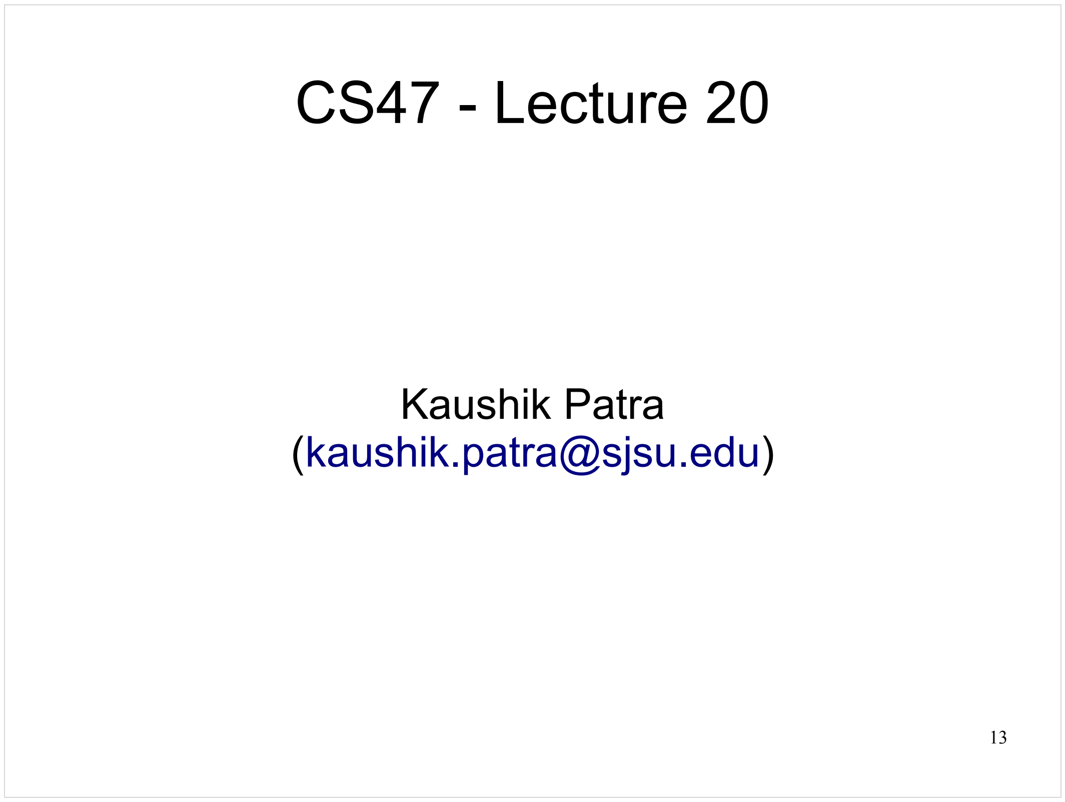
Signed Division

- Create a procedure 'div_signed'
  - Arguments
    - $a0 : Dividend
    - $a1 : Divisor
  - Return
    - $v0 : Quotient
    - $v1 : Remainder
  - Approx 60 line of code

Signed Division

- Steps
  - N1 = $a0, N2 = $a1
  - Make N1 two's complement if negative
  - Make N2 two's complement if negative
  - Call unsigned Division using N1, N2. Say the result is Q and R
  - Determine sign S of Q
    - Extract $a0[31] and $a1[31] bits and xor between them. The xor result is S.
    - If S is 1, use the 'twos_complement' to determine two's complement form of Q.
  - Determine sign S of R
    - Extract $a0[31] and assign this to S
    - If S is 1, use the 'twos_complement' to determine two's complement form of R.

12



CS47 - Lecture 20

Kaushik Patra
(kaushik.patra@sjsu.edu)

13

- Division

Reference Books:

1) *Chapter 3 of 'Computer Organization & Design by Hennessy, Patterson*