(individual assessment).

Midterm exam on Wednesday Nov. 1.

Open book, open notes, Laptop allowed

No internet access is allowed.

- ASM / ASMD chart - Algorithms to HW.
- FSM's Mealy vs Moore
- ASMD Design process
- Memory (RAM, ROM, Reg. File, Buffer)

4 per round table rest on side tables.

Division Circuit

- Design a circuit that implements the long-division algorithm:

```
      15
   ------
9 ) 140
     9
    ---
     50
     45
    ---
      5
```

(a) An example using decimal numbers

```
                 00001111   <- quotient
              ----------
divisor -> 1001 ) 10001100   <- dividend
                  1001
                 -----
                  10001
                   1001
                  -----
                   10000
                    1001
                   -----
                    1110
                    1001
                   -----
                     101   <- remainder
```

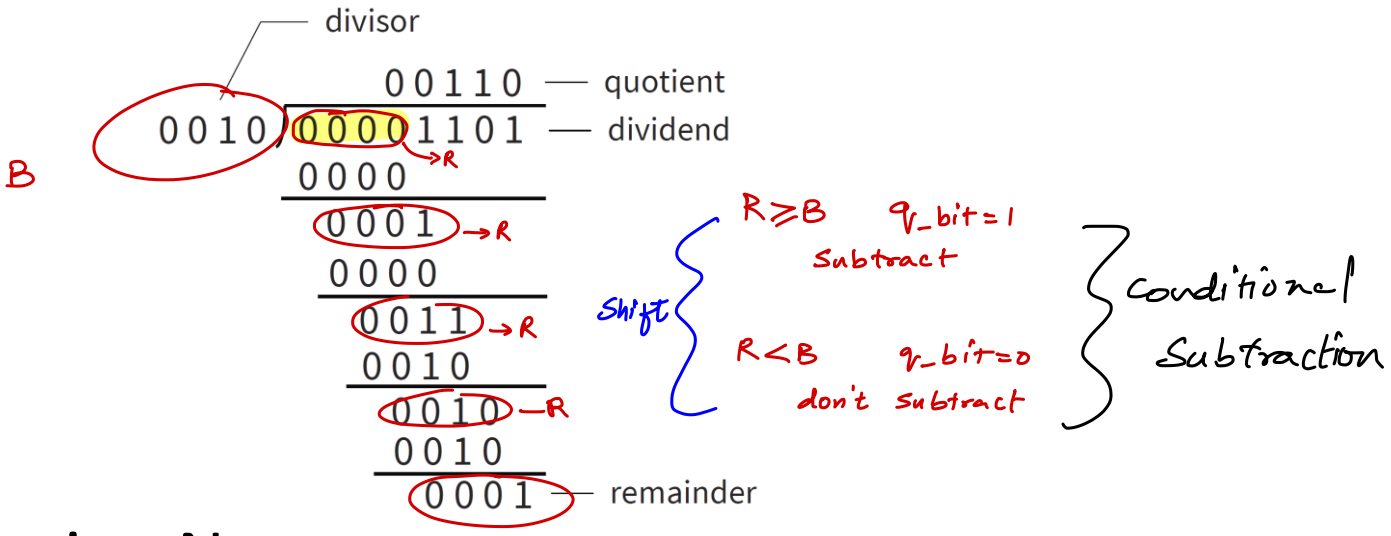
(b) Using binary numbers

- Considerations:
  - Main operations? Subtract, shift, compare
  - Stop condition? n iterations if both divisor & dividend are n-bits.

Division Circuit

- Design a circuit that implements the long-division algorithm:
  1) Double the dividend width by appending 0's in front and align the divisor to the leftmost bit of the *extended dividend*.
  2) If the corresponding dividend bits are ≥ the divisor, subtract the divisor from the dividend bits and make the corresponding quotient bit 1. Otherwise, keep the original dividend bits and make the quotient bit 0.
  3) Append one additional dividend bit to the previous result and shift the divisor to the right one position.
  4) Repeat steps 2 and 3 until all dividend bits are used.

Division Circuit



- Implementation Notes:
  - If current dividend window (R) is smaller than the divisor (B), skip subtraction
  - Instead of shifting divisor to the right, we will shift the dividend (and the quotient) *to the left*
  - We will re-use the lower half of the dividend register to store the quotient

Expected Q = 0111 R = 0001

Division Circuit Operation

DATAPATH

- A few steps of:

```
0010 | 1111
```

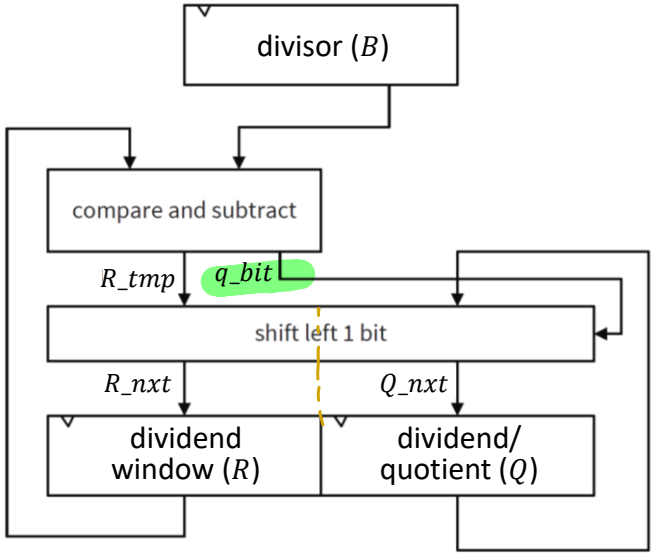
q_bit = (R >= B)

Q_nxt = {Q[n-2:0], q_bit}

R_tmp = q_bit ? R-B : R

R_nxt = {R_tmp[n-2:0], Q[n-1]}

CL compute



extended dividend.

dividend.

Op (done)	B	R	Q	q_bit	R_tmp	R_nxt	Q_nxt	P
Initialize	0010	0000	1111	0	0000 R	0001	1110	100
Compute	0010	0001	1110	0	0001 R	0011	1100	011
Compute	0010	0011	1100	1	0001 R-B	0011	1001	010
Compute	0010	0011	1001	1	0001	0011	0011	001
Compute	0010	0011	0011	1	0001	0010	0111	000
Done	0010	0001	0111	X	X	X	X	X

Division Circuit Specification

- Datapath
  - $2n$-bit *register* with bits split into n-bit R and n-bit Q
  - Divisor stored in register B, dividend stored in Q, R holds 0
  - A "compare and subtract" module outputs $\{R, 0\}$ if $R < B$ and $\{R - B, 1\}$ otherwise ($\{R\_tmp, q\_bit\}$)
  - A *shifter* left shifts q_bit into $\{R\_tmp, Q\}$ and outputs to the inputs of R and Q
  - A $\lceil \log_2(n+1) \rceil$-bit *counter* P
- Control
  - Inputs $Start$ and $Reset$, outputs $Ready$ and $Done$
  - Status signals: P_zero (all of P)
  - Control signals: Load_regs, Decr_P, Finish_RQ, Enable_RQ

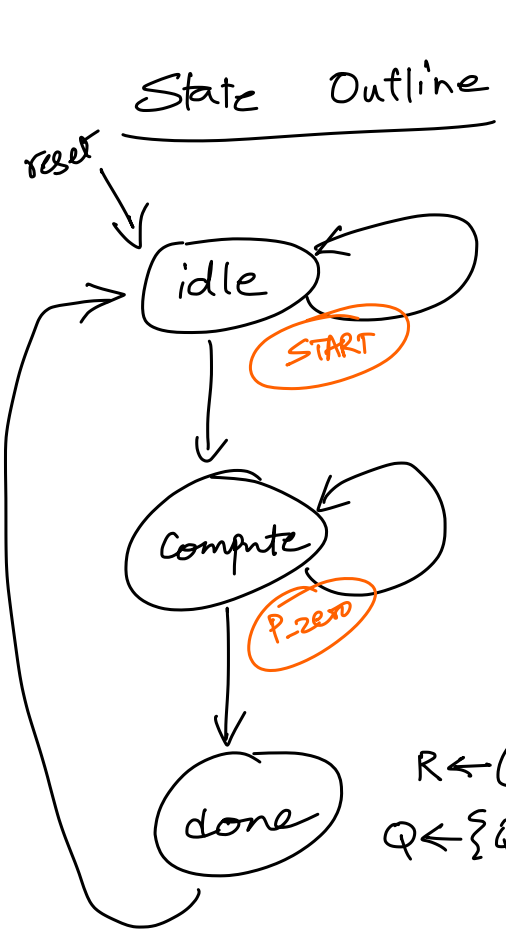
Decisions

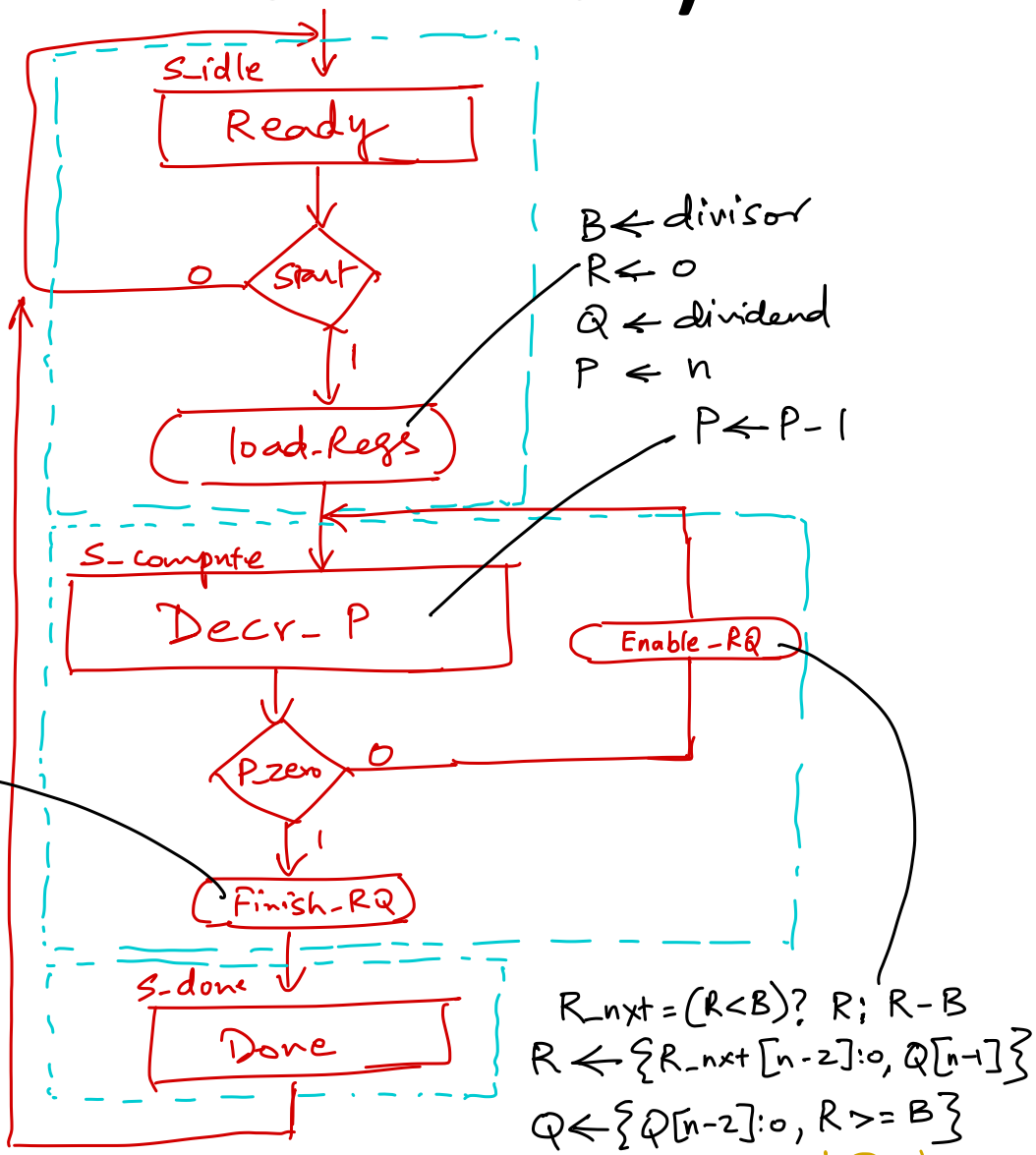
Actions

R ← R_tmp

R ← R_nxt

Division Circuit (ASM → ASMD Chart)





Division Circuit Implementation

- Controller Logic

$$Load\_regs = S\_idle \cdot Start$$

$$Enable\_RQ = S\_compute \cdot \overline{P\_zero}$$

$$Finish\_RQ = S\_compute \cdot P\_zero$$

$$Decr\_P = S\_compute$$

$$Ready = S\_idle$$

$$Done = S\_done$$

Division Circuit (SV, Datapath)

```
module datapath #(parameter WIDTH=4)
                 (Q, P, divisor, dividend, clk,
                  Load_regs, Enable_RQ, Enable_R, Decr_P);

   // port definitions
   output logic [2*WIDTH-1:0] product;
   output logic [WIDTH-1:0] Q, P;  // note: unnecessary bits for P
   input  logic [WIDTH-1:0] multiplicand, multiplier;
   input  logic clk, Load_regs, Shift_regs, Add_regs, Decr_P;

   // internal logic
   logic [WIDTH-1:0] B, R, R_tmp, R_nxt;
   logic q_bit;









endmodule
```

Division Circuit (SV, Datapath)

```
module datapath #(parameter WIDTH=4)
                 (Q, P, divisor, dividend, clk,
                  Load_regs, Enable_RQ, Enable_R, Decr_P);

   // port definitions & internal logic
   ...

   // assignments
   assign q_bit = (R >= B);
   assign R_tmp = (R < B) ? R : R-B;
   assign R_nxt = {R_tmp[WIDTH-2:0], Q[WIDTH-1]};
   assign Q_nxt = {Q[WIDTH-2:0], q_bit};

   // datapath logic
   always_ff @(posedge clk) begin
      if (Load_regs) begin
         R <= 0;       Q <= dividend;
         P <= WIDTH;   B <= divisor;
      end
      if (Decr_P)      P <= P - 1;
      if (Comp_regs)   {R, Q} <= {R_nxt, Q_nxt};
      if (Done_regs)   {R, Q} <= {R_tmp, Q_nxt};
   end  // always_ff

endmodule
```

Lab 4 Preview: Bit Counter

- Design a circuit that counts the number of bits in a register A that have the value 1

- Algorithm:

```
B = 0;  // counter
while A != 0 do
    if A[0] = 1 then
        B = B + 1
    endif
    A = A >> 1
endwhile
```

EE/CSE 371
Design of Digital Circuits and Systems

Lecture 9: Algorithms to Hardware II

Arithmetic mean

Sorting

Arithmetic Mean

Data Output M

- Design a sequential circuit that computes the mean of k n-bit numbers stored in registers
  - *e.g.*, accessing a RAM or register file with k addresses
  - To save on hardware, you can only use one n-bit adder and have a single read port RAM

k: No. of numbers; n: width of numbers

- Algorithm Pseudocode:

①
```
S = 0
for i = 0 to K-1
S = S + R[i]
end for
M = S/K
```

②
```
S = 0
for i = (K-1) to 0
S = S + R[i]
end for
M = S/K
```

③
```
M = 0
for i = K-1 to 0
M = M + R[i]/K
end for
```

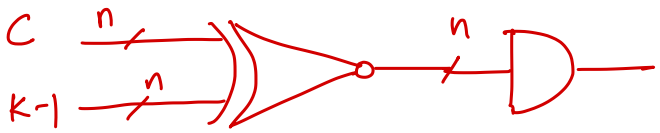
Aside: Counter Variable

equality detectors ✓ XNOR.

- Many sequential hardware algorithms utilize counters
- If both work, is there a preference?
  - How to implement $C = k - 1$ check?

$C = C_{n-1}\ C_{n-2} \cdots C_1\ C_0$

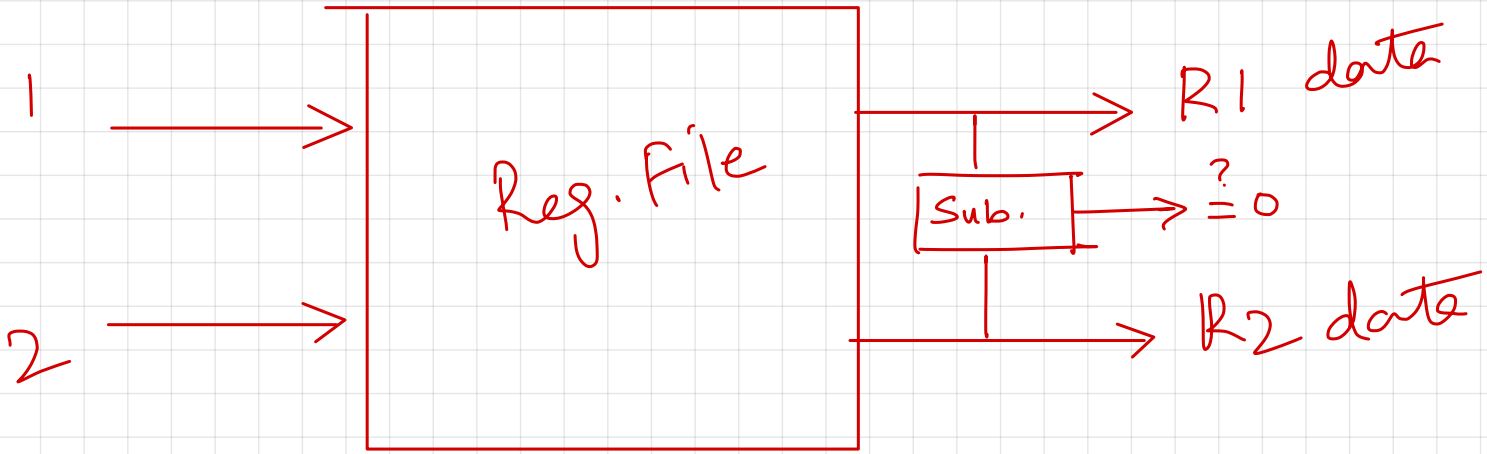
$K-1 = (K-1)_{n-1}\ (K-1)_{n-2} \cdots (K-1)_1\ (K-1)_0$

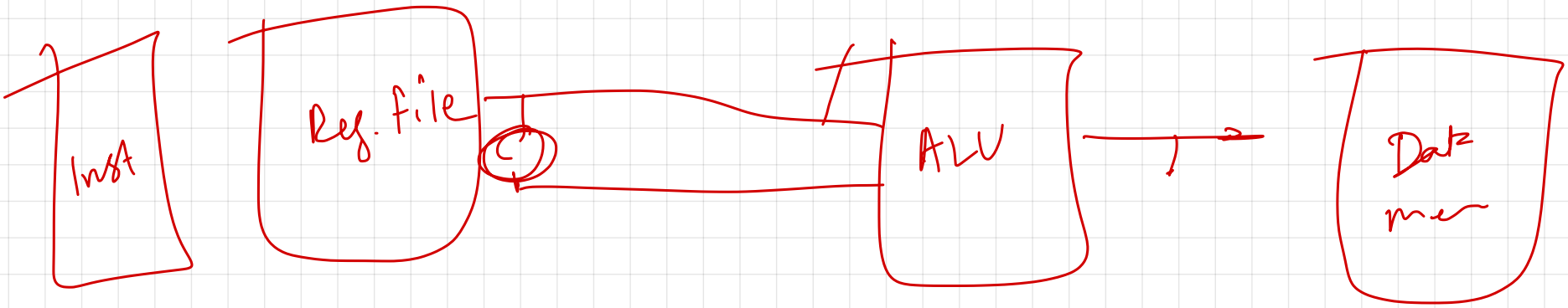


$[C - (K-1)] \stackrel{?}{=} 0$

  - How to implement $C = 0$ check?

Better


1
2
Reg. File
R1 data
Sub.
?
=0
R2 data


Inst
Reg. file
ALU
Data
mem

Arithmetic Mean Specification

- Datapath
  - A k-address *register file* (only using `r_addr` and `r_data`)
  - Reg file address stored in $\lceil \log_2(k) \rceil$ *down-counter* A
  - Sum stored in register S
  - An n-bit *divider* circuit, as discussed last lecture

- Control
  - Inputs $Start$ and $Reset$, outputs $Ready$ and $Done$
  - Status signals: A_zero , Div_done (?)
  - Control signals: Load_regs , Add , Divide , Decr_A .

Status indicators.