DESIGN OF DIGITAL CIRCUITS AND SYSTEMS

ASM with Datapath III

Instructor: Justin Hsia

Teaching Assistants:

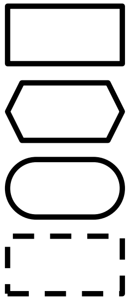
Colton Carroll
Hemil Patel
Rasya Fawwaz
Grace Zhou
Quinlyn Donohue
Rose Maresh

Relevant Course Information

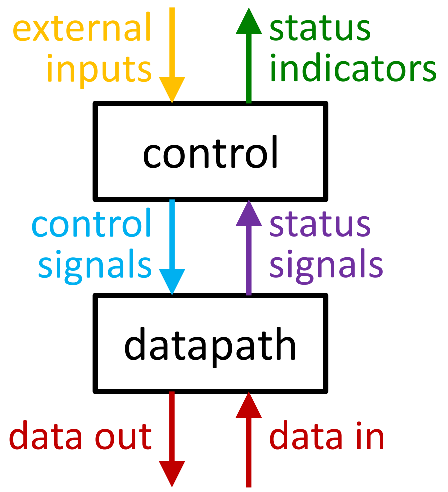
- Homework 3 due tomorrow
- Homework 4 released today and due 5/4
  - ASMDs and algorithm implementation debugging
- Quiz 2 (ROM, RAM, Reg files) @ 11:50 AM
- Lab 3 reports due next Friday (5/1)
  - Ideally finish by early next week so you can start Lab 4
- Lab 4 released today and due 5/8
  - Implementing bit counting and binary search algorithms

ASMD Chart Review Questions

state box:
decision box:
conditional output box:
ASM block:



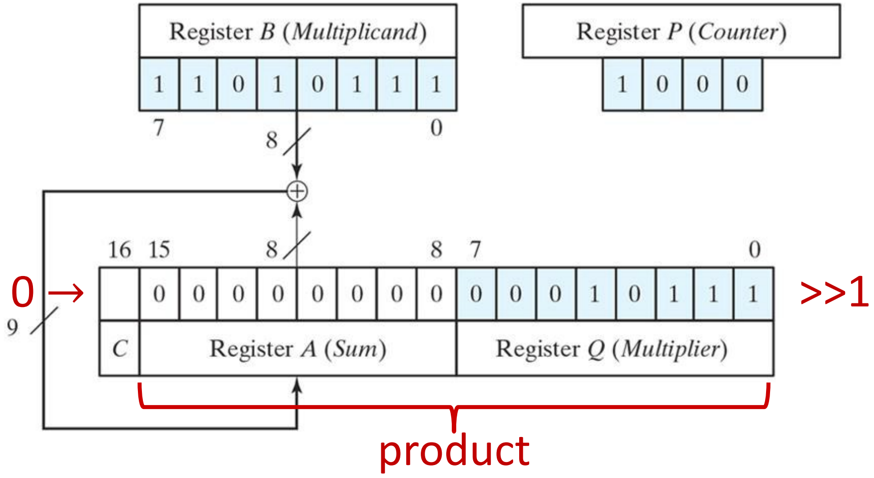
- Circle all that apply:
  - Where can **control signals** be found? outputs
    - State boxes   Decision boxes   Conditional output boxes
  - Where can **status signals** be found? inputs
    - State boxes   Decision boxes   Conditional output boxes
  - Where can **external input signals** be found? inputs
    - State boxes   Decision boxes   Conditional output boxes
  - What is the *first* thing a path should encounter in an **ASM block**?
    - State boxes   Decision boxes   Conditional output boxes
  - What can be found outside of **ASM blocks**? None!
    - State boxes   Decision boxes   Conditional output boxes
  - What can **RTL operations** be attached to? for datapath
    - Control signals   Status signals   External output signals



Sequential Binary Multiplier Operation

- A few steps of:

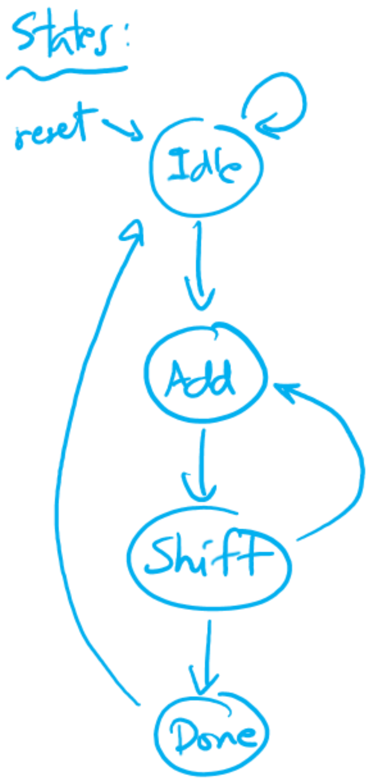
```
   11010111
x  00010111
```

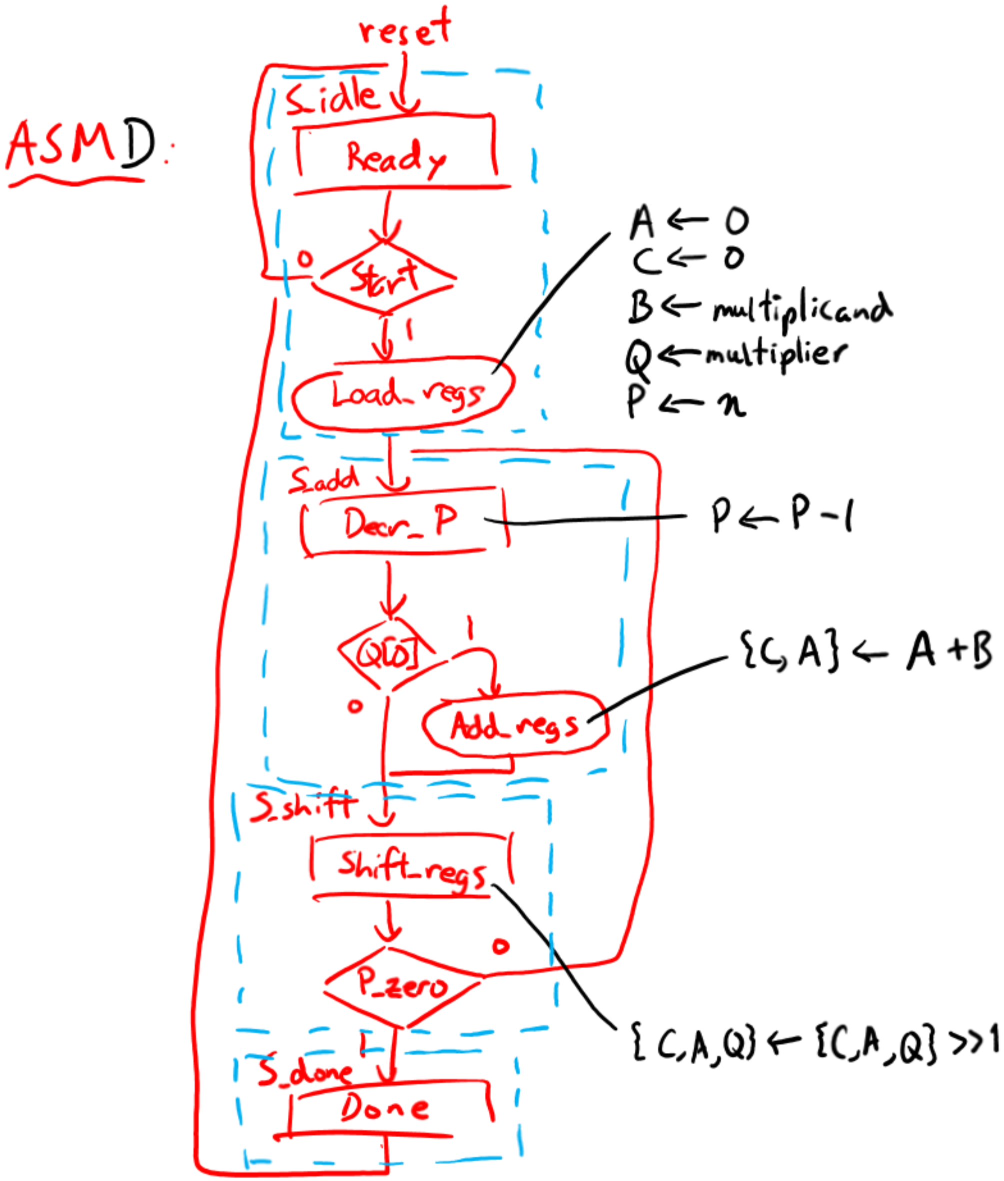


Operation (completed)	C	A	Q	P
Initialize computation	0	00000000	00010111	1000
Add (Q[0]=1)	0	11010111	0001011**1**	1000
Shift	0	01101011	10001011	0111
Add (Q[0]=1)	1	01000010	1000101**1**	0111
Shift	0	10100001	01000101	0110
Add (Q[0]=1)	1	01111000	0100010**1**	0110
Shift	0	10111100	00100010	0101

-1
-1
-1

Binary Multiplier: ASMD Chart





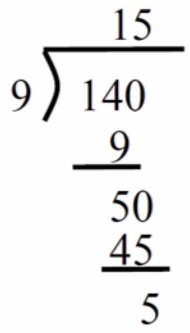
Justin's ASMD Process

1) Identify datapath components, control signals, and status signals from description or pseudocode.
2) [*optional*] Create control-datapath circuit diagram.
3) [*optional*] Create state outline to plan out states and transitions between them.
4) Draw out ASM state boxes, decision boxes, and paths between them.
5) Augment state boxes with Moore-type outputs and add conditional output boxes with Mealy-type outputs.
6) Add ASM blocks to organize states.
7) Add RTL operations to control signals.
8) Double-check decision box edge cases and timing of operations (*i.e.*, debug).

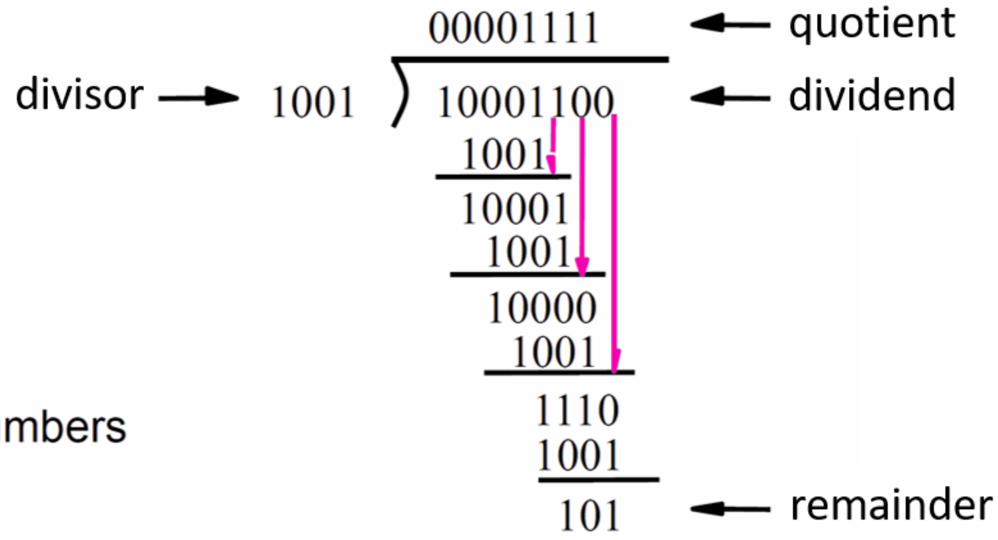
SHORT TECH BREAK

Division Circuit: Examples

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



(a) An example using decimal numbers



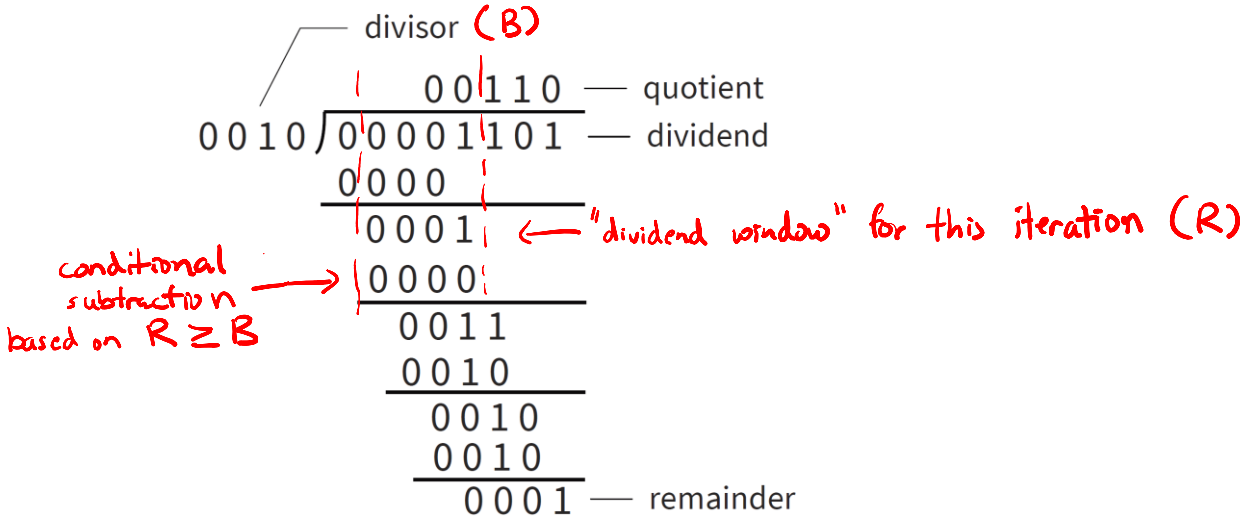
(b) Using binary numbers

- Considerations:
  - Main operations? shift, subtract, compare
  - Stop condition? n iterations if both divisor and dividend are n bits

Division Circuit: Algorithm Description

- 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



- Implementation Notes:
  - If current dividend window is smaller than the divisor, 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

Division Circuit: Operation

- A few steps of:

final Q → 0111 (7)

(2) 0010 ⟌ 00001111 (15)

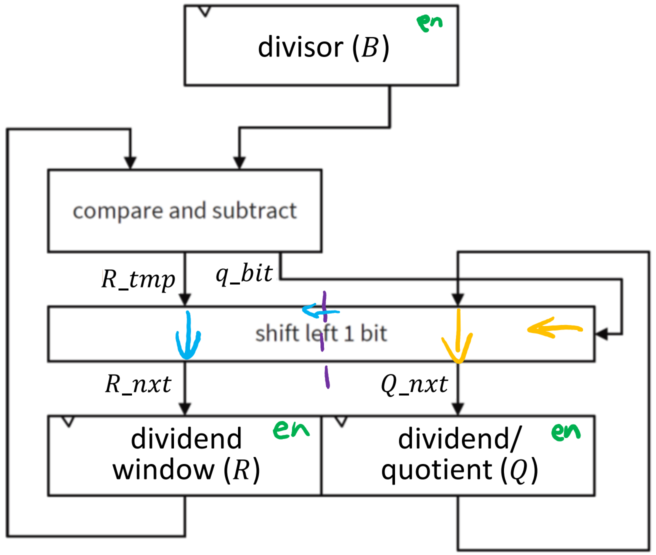
-10
11
-10
11
-10
final R → 0001

$q\_bit = (R >= B)$

$Q\_nxt = \{Q[n-2:0], q\_bit\}$

$R\_tmp = q\_bit \; ? \; R-B : R$

$R\_nxt = \{R\_tmp[n-2:0], Q[n-1]\}$



registers signals

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	↓R_nxt 0001	↓Q_nxt 1110	0	0001 (R)	0011	1100	011
Compute	0010	↓R_nxt 0011	↓Q_nxt 1100	1	0001 (R-B)	0011	1001	010
Compute	0010	↓R_nxt 0011	↓Q_nxt 1001	1	0001 (R-B)	0011	0011	001
Compute	0010	↓R_nxt 0011	↓Q_nxt 0011	1	0001 (R-B)	0010	0111	000
Done	0010	↓R_tmp 0001	↓Q_nxt 0111	X	X	X	X	X

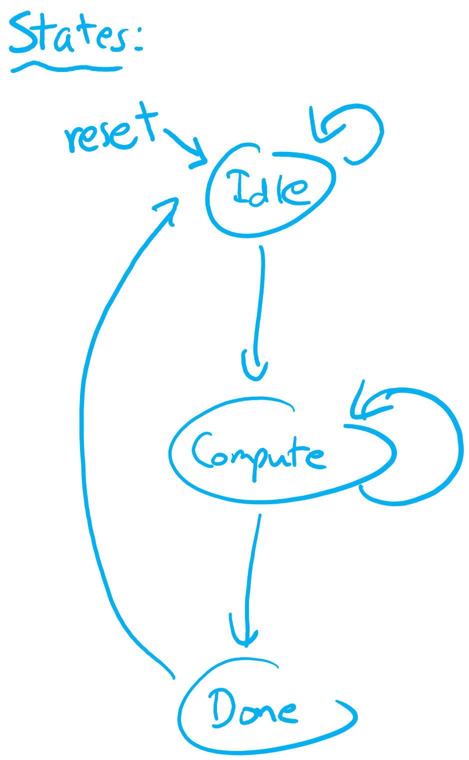
Division Circuit: Specification

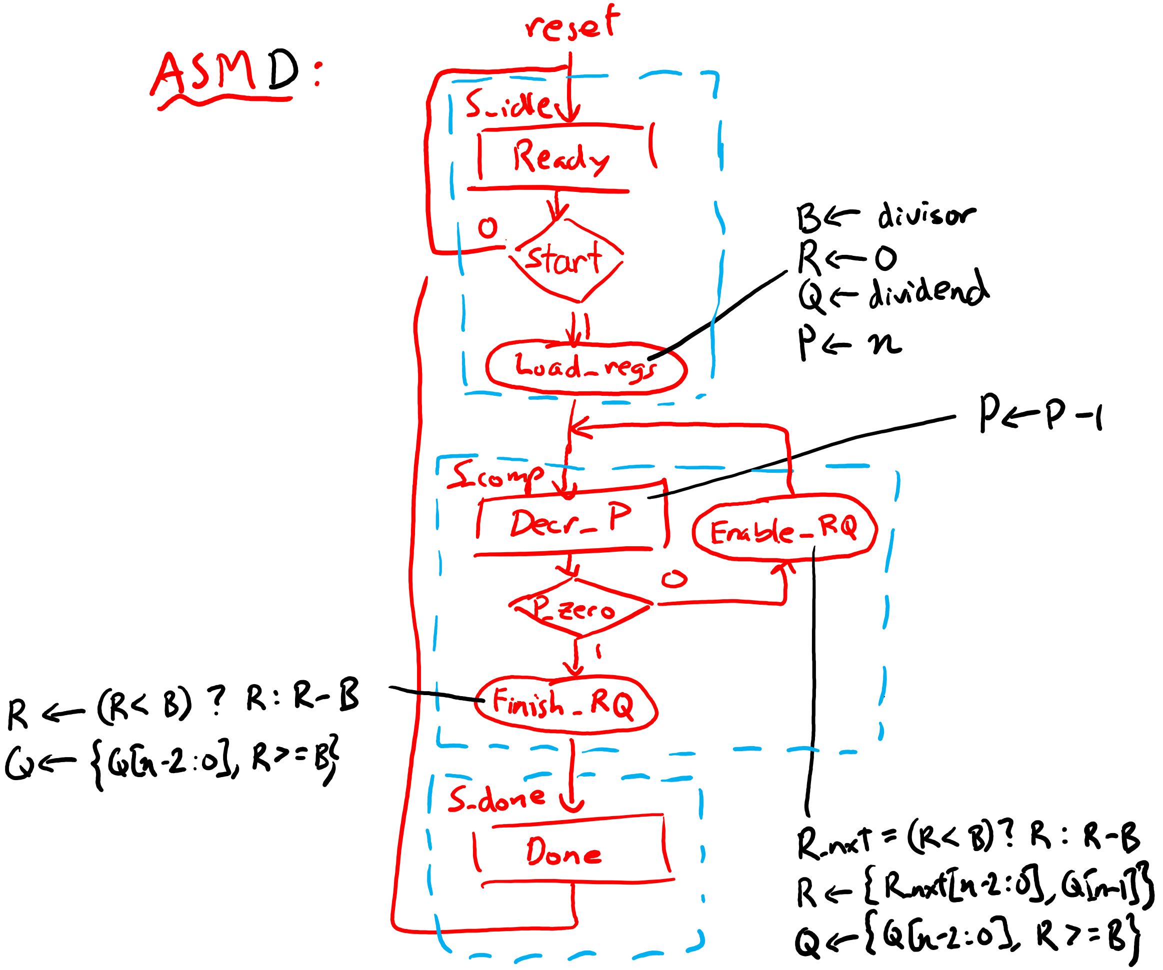
- Datapath
  - $2n$-bit *register* with bits split into n-bit R and n-bit Q
  - Divisor stored in register B, dividend stored in Q, R holds 0
  - A "compare and subtract" module outputs $\{R, 0\}$ if $R < B$ and $\{R - B, 1\}$ otherwise ($\{R\_tmp, q\_bit\}$)
  - A *shifter* left shifts q_bit into $\{R\_tmp, Q\}$ and outputs to the inputs of R and Q

combinational logic only

  - A $\lceil \log_2(n+1) \rceil$-bit *counter* P
- Control
  - Inputs $Start$ and $Reset$, outputs $Ready$ and $Done$
  - Status signals: P_zero (or all of P)
  - Control signals: Load_regs, Enable_RQ, Finish_RQ, Decr_P

Division Circuit: ASMD Chart





Division Circuit: Controller Logic

- Controller Logic

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

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

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

$$Decr\_P = S\_comp$$

$$Ready = S\_idle$$

$$Done = S\_done$$

Division Circuit: Datapath Code (1/2)

```
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: Datapath Code (2/2)

```
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
```