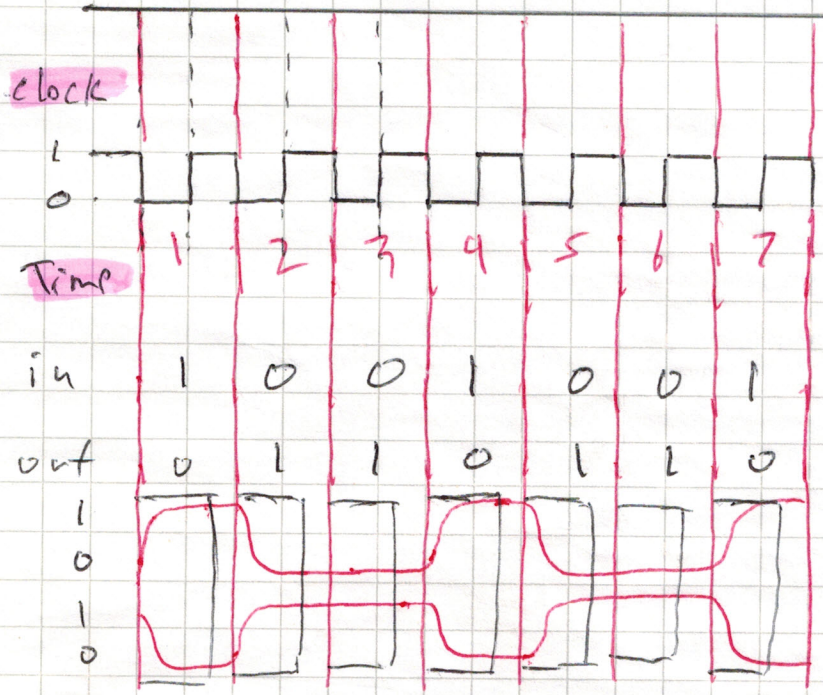
Unit 3.1 - Sequential Logic

Fri, Apr 24, 2020

- so far ignored issue of time, previous units used given inputs to calculate output → Combinatorial Logic

The Clock

Physical time



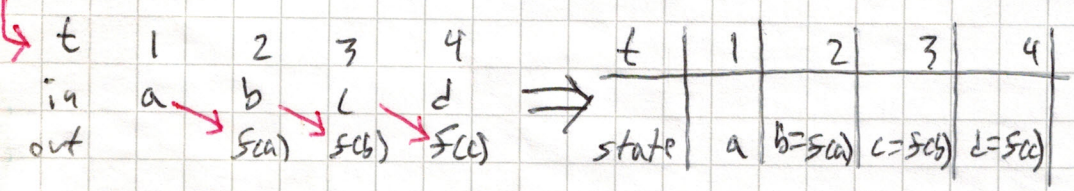
} NOT gate over time

- physical constraints of signal mean delay between signal change
- clock cycle needs to be tuned to not include wave-like ramp up period → "stabilization of state"

Combinatorial Logic vs. Sequential Logic

Combinatorial: $out[t] = f(in[t])$

Sequential: $out[t] = f(in[t-1])$



- input/output are the same bits/busses

Fri, Apr 24, 2020

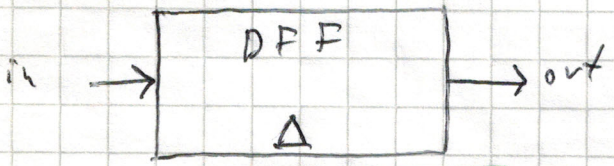
Unit 3.2 - Flip Flops

review: $state[t] = f(state[t-1])$

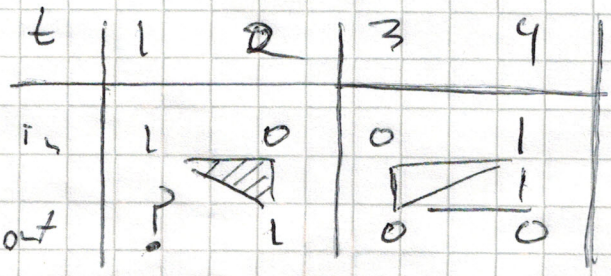
Remembering state with Flip Flops

- A flip-flop can flip between the two states of 0 & 1
- It is used to remember state at time $t-1$ so it can be used at time t

The Clocked Data Flip Flop Controlled Chaos



$out[t] = in[t-1]$

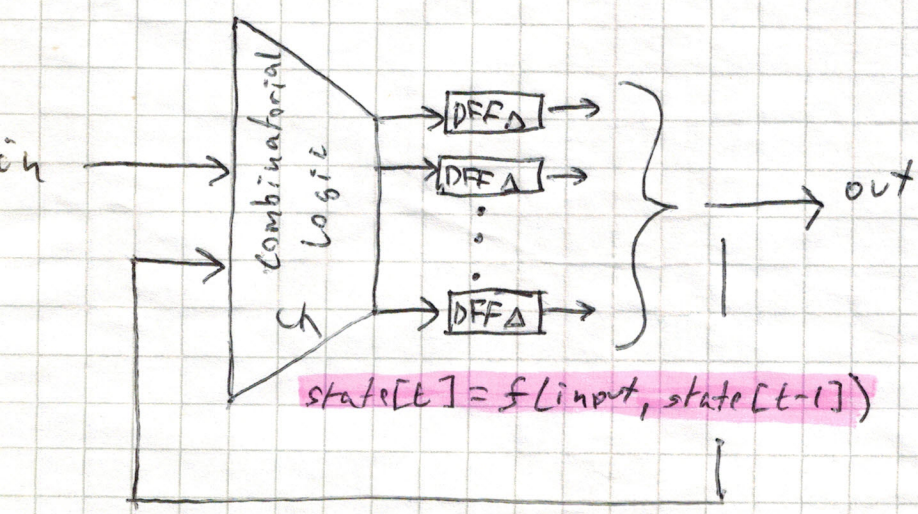


- DFF can be considered a primitive - like NAND gate
- important to separate combinatorial logic from sequential logic

Physical Implementation using NAND gates

1) create a "loop" achieving an "unclocked" flip-flop
2) Isolation across time steps using "master-slave" setup

Sequential Logic Implementation



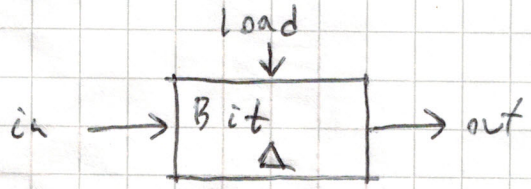
$state[t] = f(input, state[t-1])$

Fri, Apr 24, 2020

Unit 3.2 - Flip Flops (Continued)

1-bit Register

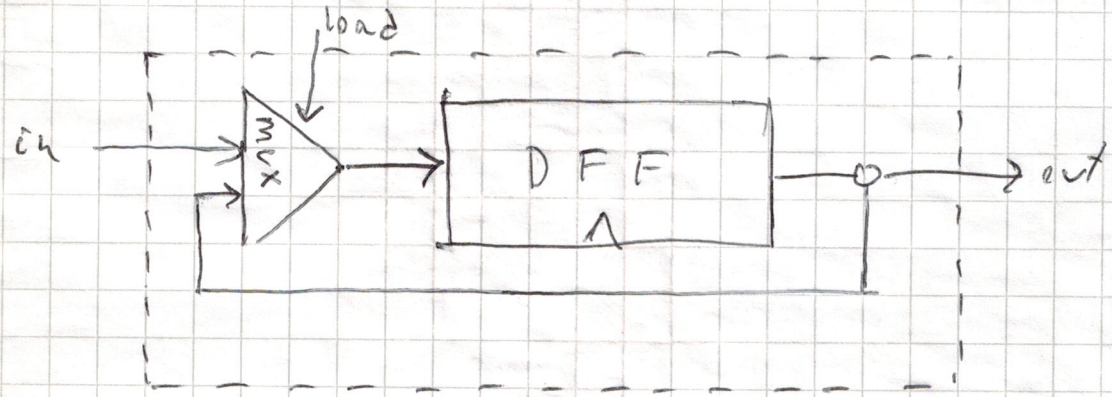
- remembering an input bit "forever": until requested to load a new value



if load(t-1) then out(t) = in(t-1)
else out(t) = out(t-1)

example

load	1	0	0	1	0
in	1	0	0	0	0
out	?	1	1	1	0



Fri, April 24, 2020

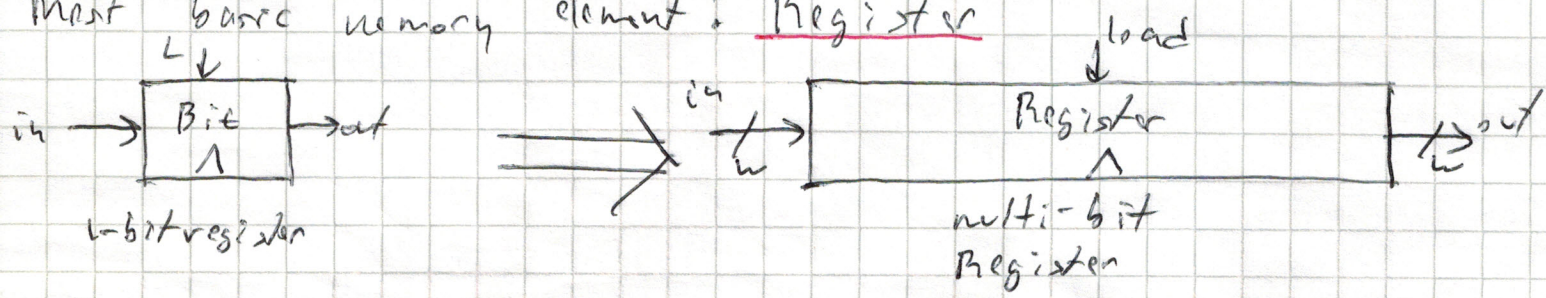
Unit 3.3 - Memory Units

memory can be:
- main memory: RAM...
- secondary memory: disks...
- volatile / non-volatile

Perspective:
- Physical
- logical

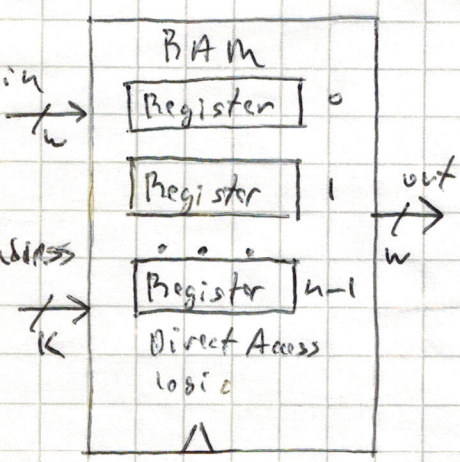
RAM:
- Data
- Instructions

Most basic memory element: Register



- w = word width → 16-bit, 32, 64...
- Register state: value currently stored in register
- to read a Register: probe out, out emits Register's state
- write logic: set in = v, load = 1 (assert load bit)
  ↳ next cycle onward, state becomes v, out emits v

RAM Unit



RAM Abstraction - sequence of n addressable registers, with addresses 0 to n-1

At any given point in time, only one Register is selected

k (width of address input): $k = \log_2 n$

w (word width): No impact on RAM logic

RAM is a sequential chip, with clocked behaviour

Fri, Apr 24, 2020

Unit 3.3 - Memory Units (continued)

To read Register i:
set address = i

Result:
out emits the state of Register i

Read Logic

To set Register i to 2:
set address = i
set in = 2
set load = 1

Result:
- state of Register i becomes 2
- next cycle onward, out emits 2

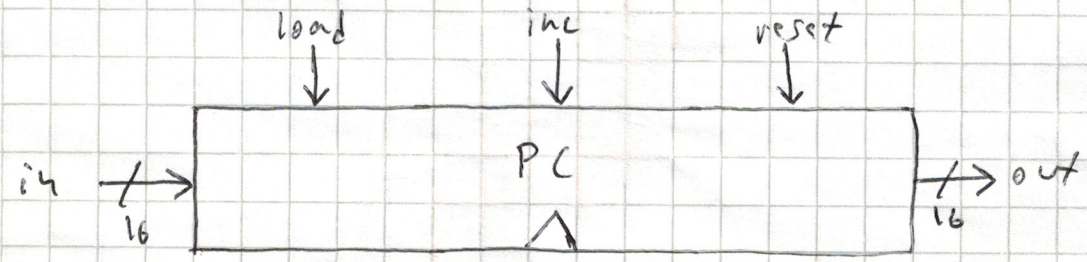
Write logic

Why the name "Random Access Memory"?
↳ because irrespective of RAM size, any register can be accessed in the same access time - instantaneously

Fri, April 24, 2020

Unit 3.4 - Counters

- Computer must keep track of which instruction should be fetched & executed next
- This control mechanism can be realized by a Program Counter
- The PC contains the address of the instruction that will be fetched & executed next
- Three possible control settings:
  - **Reset**: fetch first instruction PC = 0
  - **Next**: fetch the next instruction PC ++
  - **Goto**: fetch instruction n PC = n
- A counter is a chip that realizes this abstraction




```
if (reset[t]==1)  out[t+1]=0
else if (load[t]==1)  out[t+1] = in[t]
     else if (inc[t]==1)  out[t+1] = out[t] +1
          else out[t+1] = out[t]
```

- reset counter to 0
- setting counter = value
- incrementing counter ++
- counter does not change