3 Sequential Logic [p41]

3.1 Background [p42]

3.1.1 Sequential VS. Combinational Logic

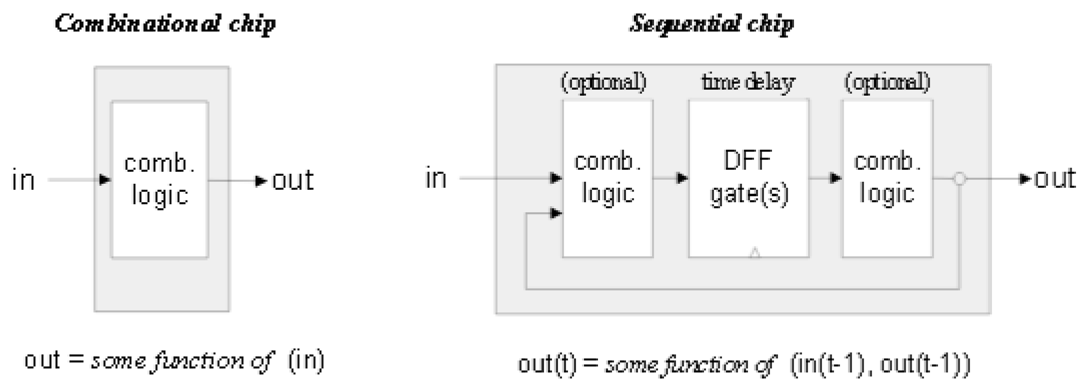
All the chips discussed and built so far are combinational chips:

Combinational devices operate on data only; compute functions that depend solely on *combinations of their values.* This type of chip cannot store and recall values.

Sequential devices operate on data and a clock signal; can be *state-aware* and provide storage and synchronisation services.

The low-level behaviour of sequential gates is **tricky**; although

Theorem 3.1. *Every sequential chip can be based upon the "**data flip flop**" or **DFF** sequential gate.*



3.1.2 Memory [p42]

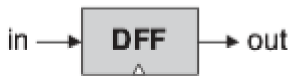
The act of "remembering something" is *time-dependent* (you remember now what has been committed to memory before). So for chip to "remember", we require s means for representing the progression of time.

Hierarchy of memory chips:

- Flip-flop gates.
- Binary cells.
- Registers.
- RAM.

The Clock Most computers have a master clock that delivers a continuous train of alternating signals between two phases labelled "0" and "1". The elapsed time between "0" and "1" is the *cycle*, and each clock cycle represent on discrete time unit.

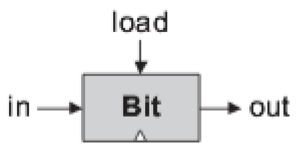
Flip-Flops We consider a variant of a flip-flop called the *data flip-flop*, whose interface consists of a single-bit data input/output, and a *clock* (continuously changing) according to the master clock's signal: $out(t) = in(t-1)$ where in/out are the input/output values and t is the current clock cycle.



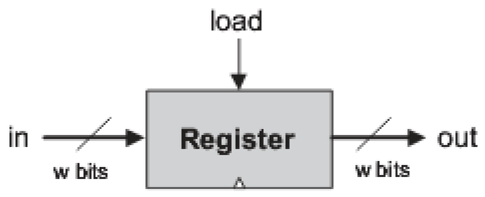
```
Chip name: DFF
Inputs:    in
Outputs:   out
Function:  out(t)=in(t-1)
Comment:   This clocked gate has a built-in
           implementation and thus there is
           no need to implement it.
```

Figure 3.1: API for a *data flip-flop* gate; c.f. Nand gate regarding primitive nature.

Registers a storage device that can "remember" a value over time: $out\,(t) = out\,(t-1)$; note, a DFF can <u>only</u> output its previous input (i.e. $out\,(t) = in\,(t-1)$).



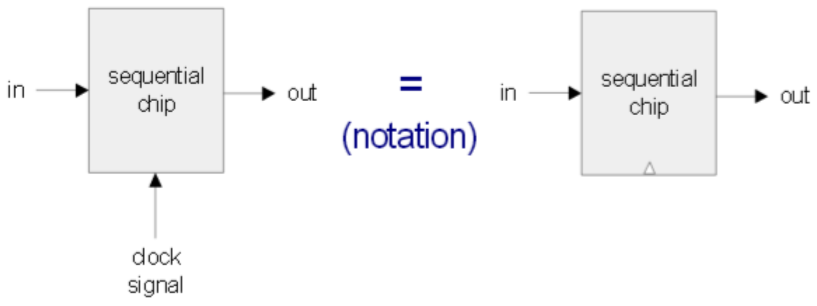
```
Chip name: Bit
Inputs:    in, load
Outputs:   out
Function:  If load(t-1) then out(t)=in(t-1)
           else out(t)=out(t-1)
```



```
Chip name: Register
Inputs:    in[16], load
Outputs:   out[16]
Function:  If load(t-1) then out(t)=in(t-1)
           else out(t)=out(t-1)
Comment:   "=" is a 16-bit operation.
```

Figure 3.2: Top: API for a single-bit register (a *Bit*, or *binary cell*) chip.
Bottom: API for a 16-bit register: to **read**, we identify the output; to **write**, we write a new data value $in = d$ into the register and assert (set to 1) the *load* input, and in the next clock cycle the register commits to the new data (outputting d).

Notation 3.2. Represent the clock signal of a sequential chip:



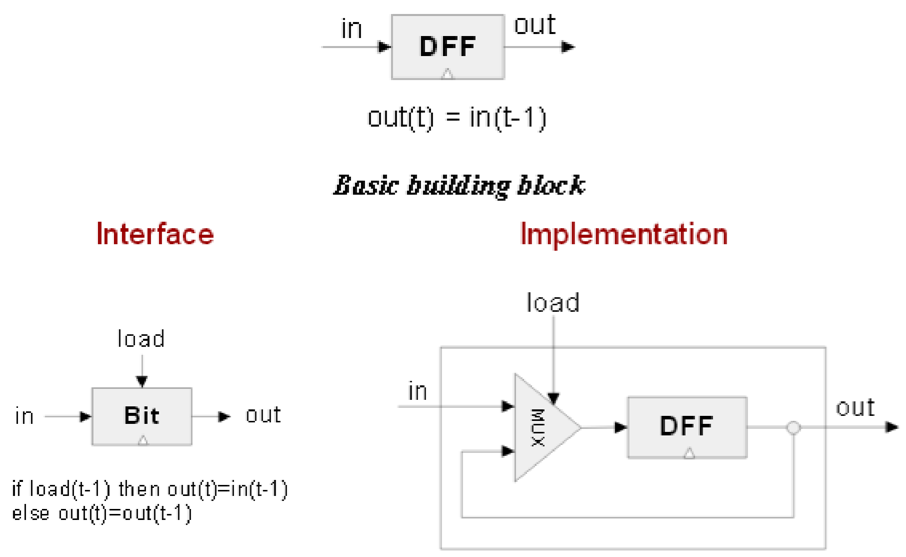
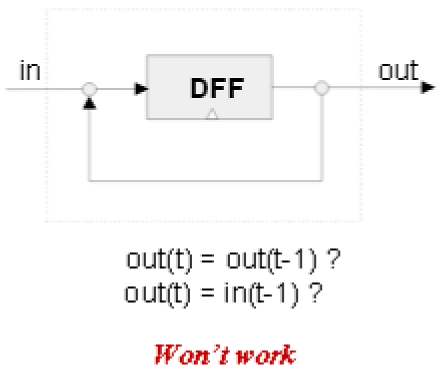


Figure 3.3: From a DFF to a single-bit register: to start storing a new value, set *load* = 1, and keep storing until *load* = 0.

Remark (WARNING:). The following 1-bit register description from a DFF is invalid:



It is not clear how we'll never load new data into this because should we draw input from the *in* or *out* wire? More generally, chips design dictates that *internal pins must have a fan-in of 1*.

Once we have the mechanism for remembering a single bit with time, we construct n-bit wide registers by forming an array of as many single-bit registers as needed.

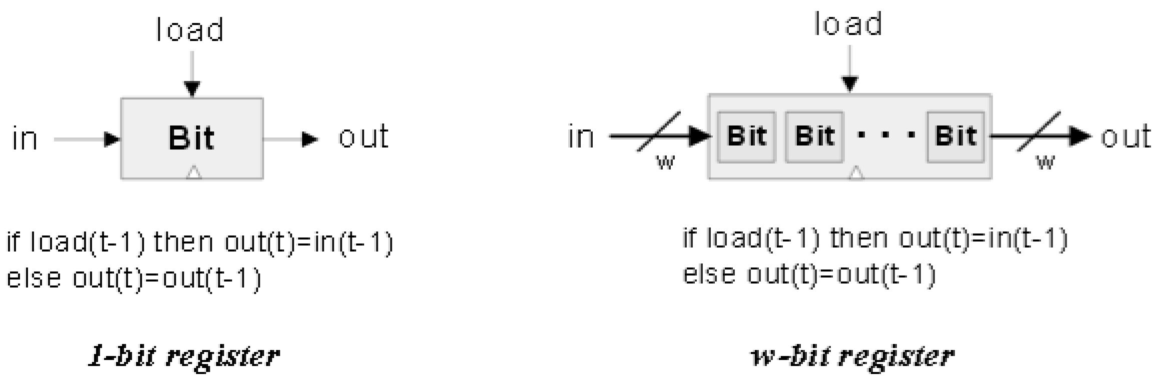


Figure 3.4: From single-bit to multi-bit registers.

The *width* of a register is the number of bits it holds (e.g. 16,32,64, ...).
The multi-bit contents of a register are the *words*.

Memories Stacking many register form a *Random Access Memory (RAM)* unit; this name is derived from the requirement that read/write operations of any randomly chosen word on RAM should be accessed directly (in equal speed, irrespective of it physical location). Hence, a classical RAM device accepts three inputs: *data*, *address*, and *load*. *Address* specifies which RAM register to access in the current time unit. If $load = 0$ (a read operation) the RAM's $out = value\, of\, selected\, register$; otherwise (a write operation, $load = 1$) the selected memory register commits to the input value.

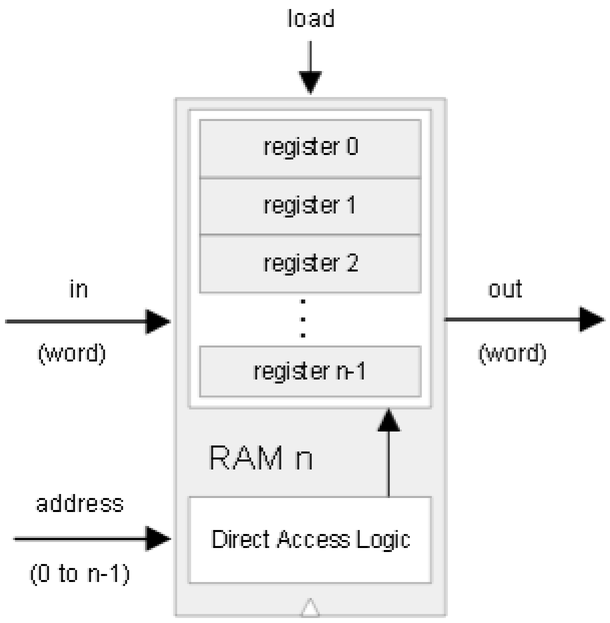
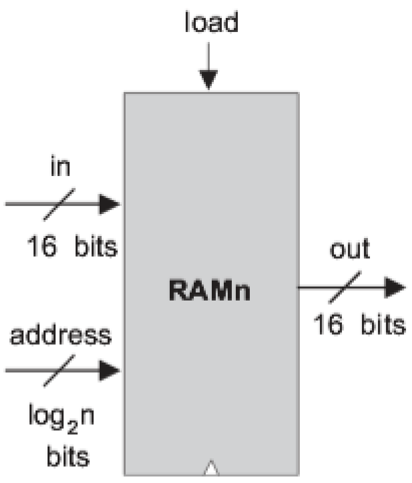
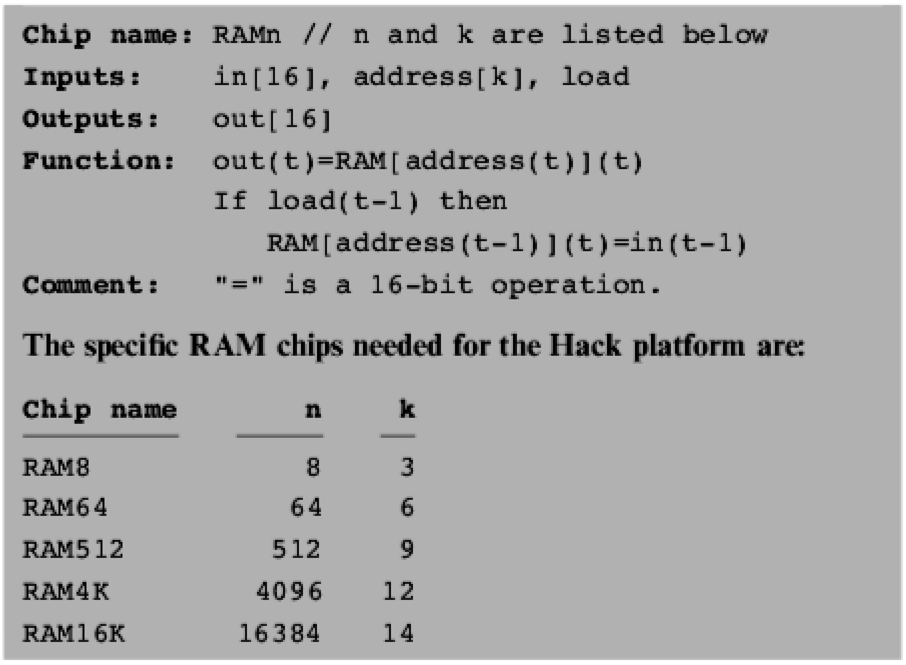


Figure 3.5: RAM.

The *width* of RAM of one of its registers (32- or 64-bit wide RAM).
The *size* of RAM is the number of registers in the RAM.





```
Chip name: RAMn // n and k are listed below
Inputs:    in[16], address[k], load
Outputs:   out[16]
Function:  out(t)=RAM[address(t)](t)
           If load(t-1) then
              RAM[address(t-1)](t)=in(t-1)
Comment:   "=" is a 16-bit operation.
```

The specific RAM chips needed for the Hack platform are:

Chip name	n	k
RAM8	8	3
RAM64	64	6
RAM512	512	9
RAM4K	4096	12
RAM16K	16384	14

Figure 3.6: API for16-bits wide RAM with various sizes (RAM8, RAM64, RAM512, RAM4K and RAM16").
Read: to read register number m, input $address = m$ outputting the value of this register (combinational operation independtent of the clock).
Write: to write new data d to register number m, input $address = m$,$in = d$ and asser the *load* input. In the next clock cycle, the selected register commits the new value d, as well as the RAM outputting this.

Counters a sequential chip whose state is $c \in \mathbb{Z}$ (typically 1) that increments every time unit, effecting $out\,(t) = out\,(t-1) + c$. Use for tasks that require such a measure.

3.2 Specification [p47]

Hierarchy of sequential chips:

- Data Flip-flops (DFFs).
- Registers (based on DFFs).
- Memory banks (based on registers).
- Coutner chips (based on registers).

3.3 Implementation [p50]

3.4 Perspective [p52]

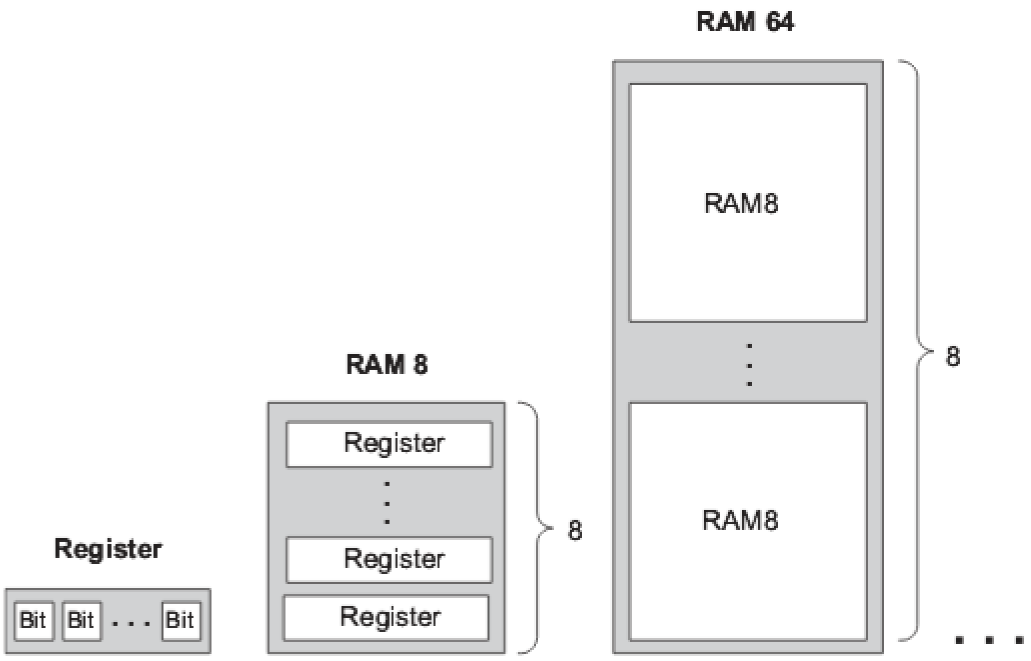


Figure 3.8: Construction of memory banks by recursive ascent.