Digital Techniques 2

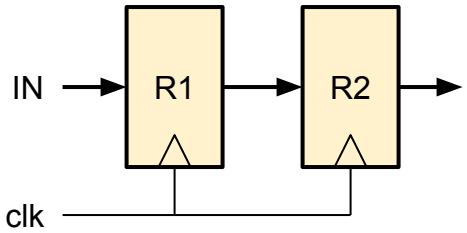
L11: Finite-State Machine with Datapath Architecture

UNIVERSITY OF OULU
CIRCUITS and SYSTEMS

Reminder: Register Transfers and Micro-Operations

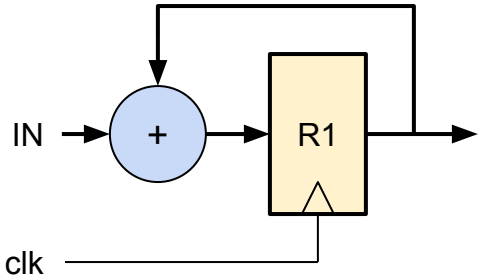
DIRECT TRANSFERS (CONCURRENT)

`R1 ← IN, R2 ← R1`



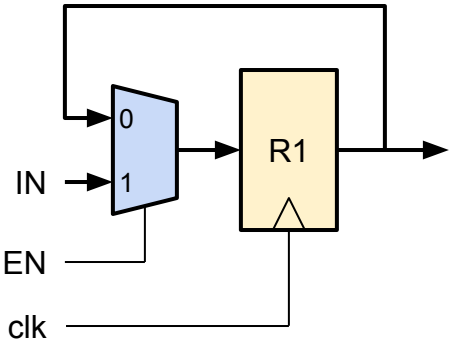
TRANSFER WITH MICRO-OPERATION

`R1 ← R1 + IN`



CONDITIONAL TRANSFER

`if (EN) R1 ← IN`



The RTL Architecture Design Problem

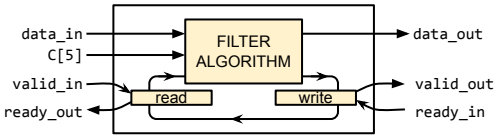
We have:

- Combinational resources (adders, multipliers, "random" logic blocks etc)
- Sequential resources (registers, register banks, counters)
- Control resources (multiplexers for data-flow control, finite-state machines for scheduling control)

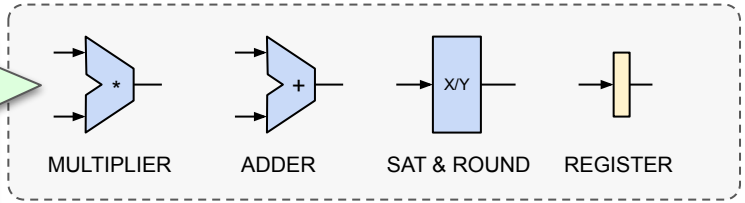
Question: How do we build data processing circuits using these?

Example: FIR Filter Algorithm

```
forever {
  data_in = read();
  for (int i=4; i > 0; --i)
    d[i] = d[i-1];
  d[0] = data_in;
  acc = 0;
  for (int i=0; i < 5; ++i)
    acc = acc + d[i] * C[i];
  data_out = saturate_and_round(acc);
  write(data_out);
}
```

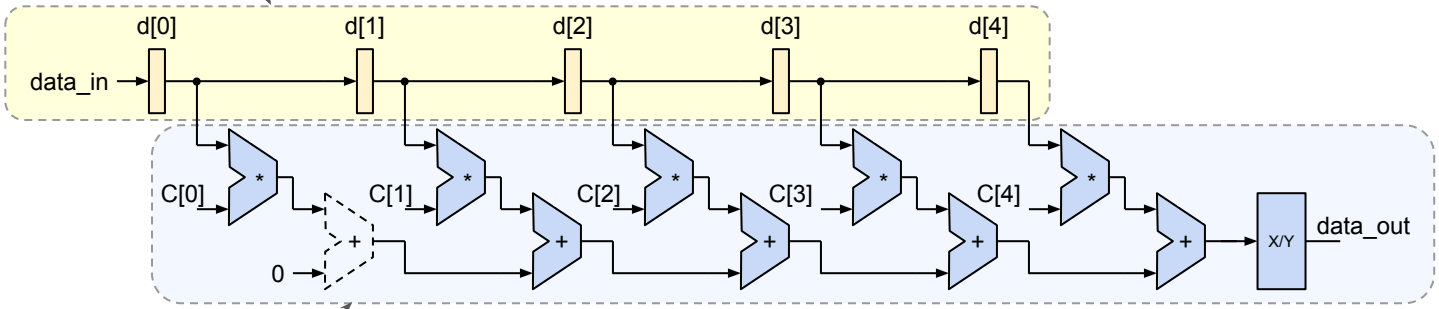


We need at these resources, but how many, and how do we organize everything?



Direct Implementation Based on Algorithm's Operations and Data Flow

```
for (int i=4; i > 0; --i)
    d[i] = d[i-1];
  d[0] = data_in;
```

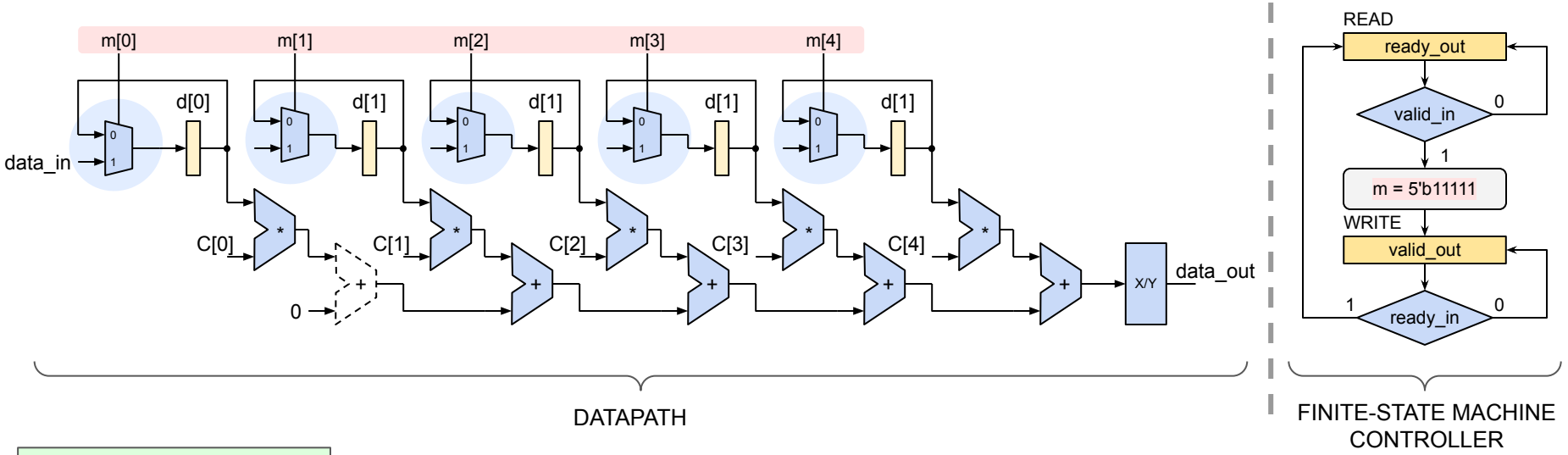



```
acc = 0;
for (int i=0; i < 5; ++i)
  acc = acc + d[i] * C[i];
data_out = saturate_and_round(acc);
```

We can allocate resources based on computation and storage requirements, and connect them according to the algorithm's internal dataflow…

…but this solution does not stop to read and write data. Control logic must be added.

Control Is Implemented by Adding Multiplexers and a Finite-State Machine



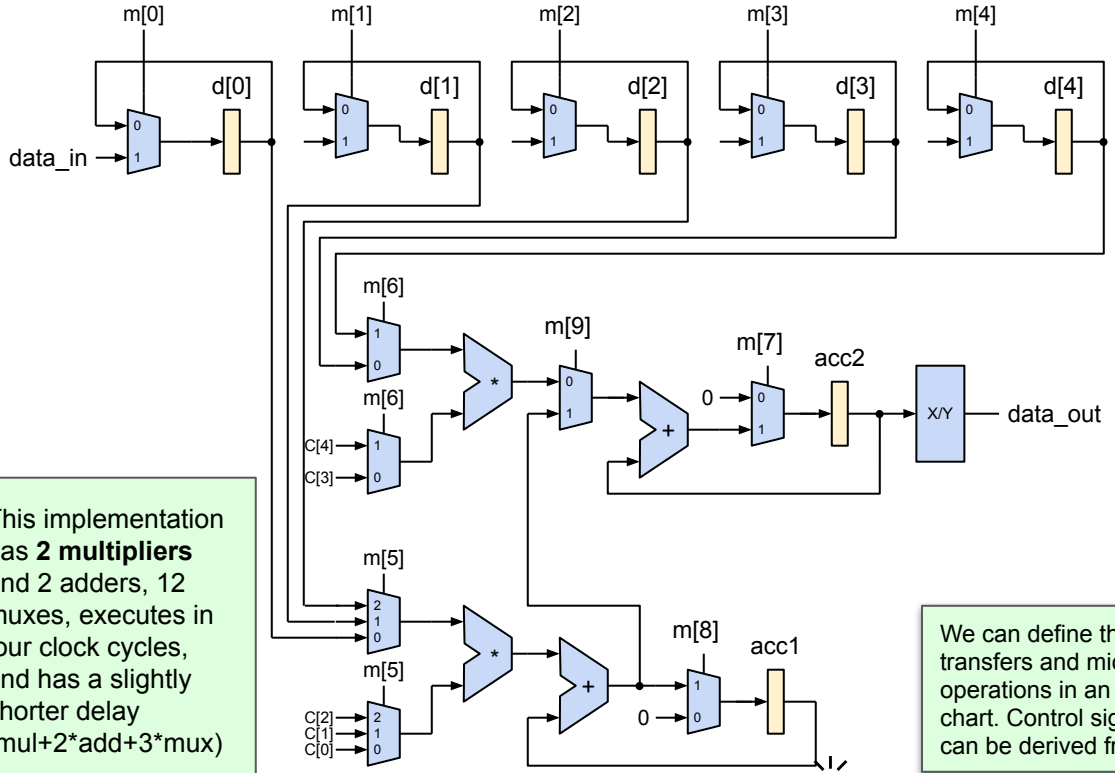
This direct implementation has **5 multipliers**, 4 adders and 5 muxes, executes in one clock cycle, and has a long delay (mul + 4*add + sat/round).

In this **finite-state machine with datapath** (FSMD) architecture, the datapath contains computation, storage and data-flow control resources (muxes).

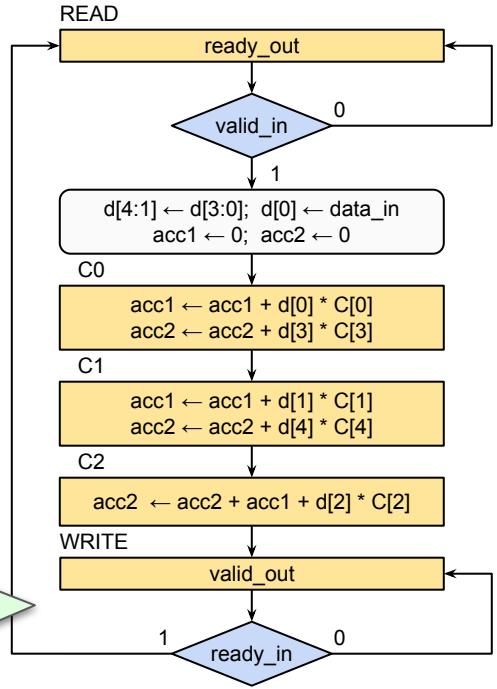
The FSM controls the schedule of the register transfers and micro-operations in clock cycles.

The RTL Architecture Must Usually Be Optimized for Some Criteria

silicon area
latency in cycles
logic delay



This implementation has **2 multipliers** and 2 adders, 12 muxes, executes in four clock cycles, and has a slightly shorter delay (mul+2*add+3*mux)



We can define the register transfers and micro-operations in an ASM chart. Control signal values can be derived from them.

Summary of FSMD Architecture Design

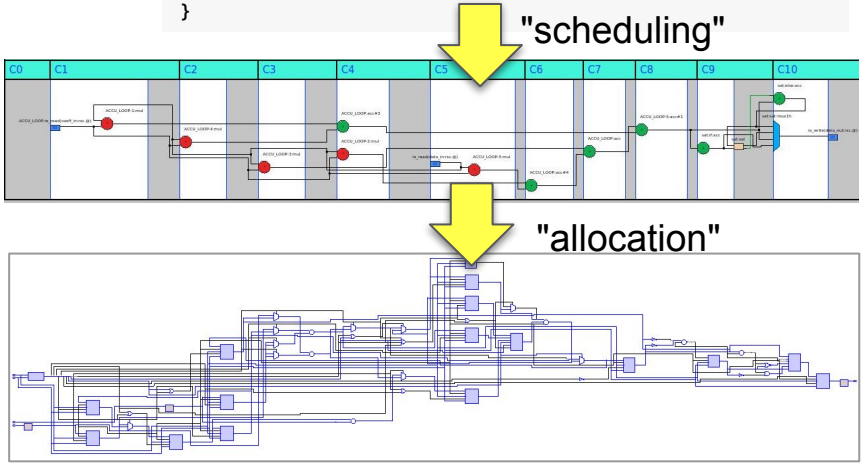
Steps:

1. Figure out exactly the storage and micro-operations that take place on each clock cycle (= scheduling of the algorithm)
2. Allocate combinational resources, multiplexers and registers that are required to implement the schedule
3. Define the control state machine

This problem is not easy!

Some EDA tools that do this automatically are called **high-level synthesis** tools =>

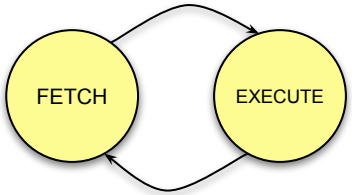
```
void fir (ac_int<DATABITS> *data_in, ac_int<DATABITS> C[TAPS],
          ac_int<DATABITS> *data_out)
{
  static ac_int<DATABITS> d[TAPS];
  ac_int<ACCBITS>         acc = 0;

 SHIFT_LOOP: for (int i = TAPS-1; i > 0; --i)
    d[i] = d[i-1];
  d[0] = *data_in;

 ACCU_LOOP: for (int i = TAPS-1; i >= 0; --i)
      acc = acc + d[i] * C[i];

  *data_out = sat(acc);
}
```

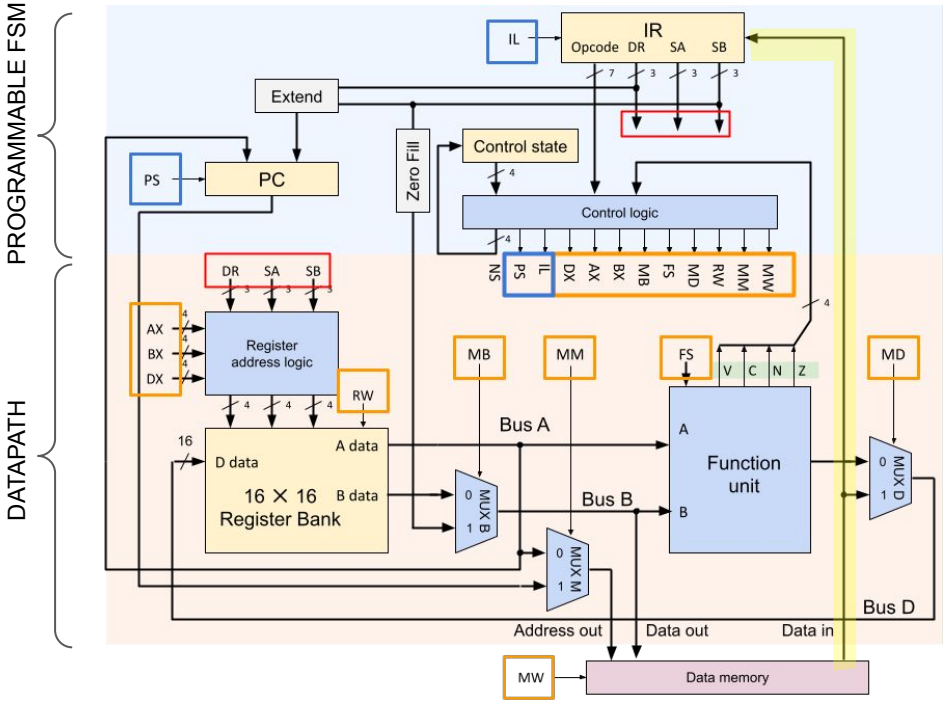


Instruction Set Processor Architecture

The ISP is a generalization of the FSMD architecture.

- General-purpose datapath support a wide range of register transfers and micro-operations, not just algorithm-specific
- Control FSM's actions are based also on data (instructions) from read from an external memory, which allows changing the control algorithm
- The FSM can be very simple: one state for fetching the instruction and another for executing it. The "real" control is in the instruction data.





Source: Mano, M. M., Kime, C. R. & Martin, T. (2014) "Logic and computer design fundamentals"

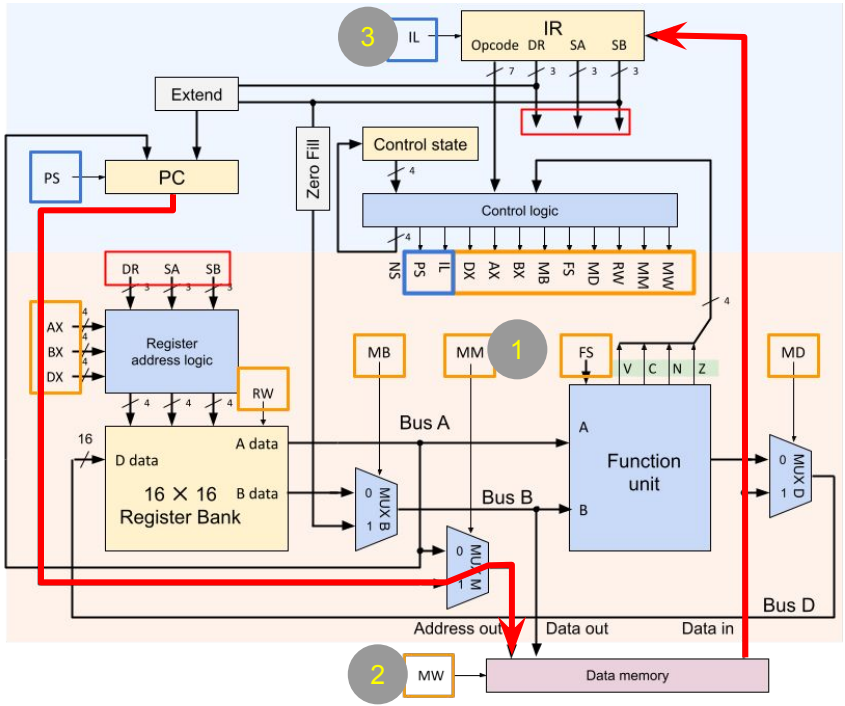
How Does IT Work: The Fetch Cycle

During the **fetch cycle** the active controls are:

1. The memory mux is set to pass the value of the **program counter** (PC) onto memory address bus with MM
2. The memory is set to read mode with MW
3. Loading of the instruction register is enabled with IL

With these settings, on the next clock edge, the instruction word is copied from the memory into the **instruction register** (IR):

IR ← MEMORY[PC]



How Does IT Work: Instruction Execution Cycle

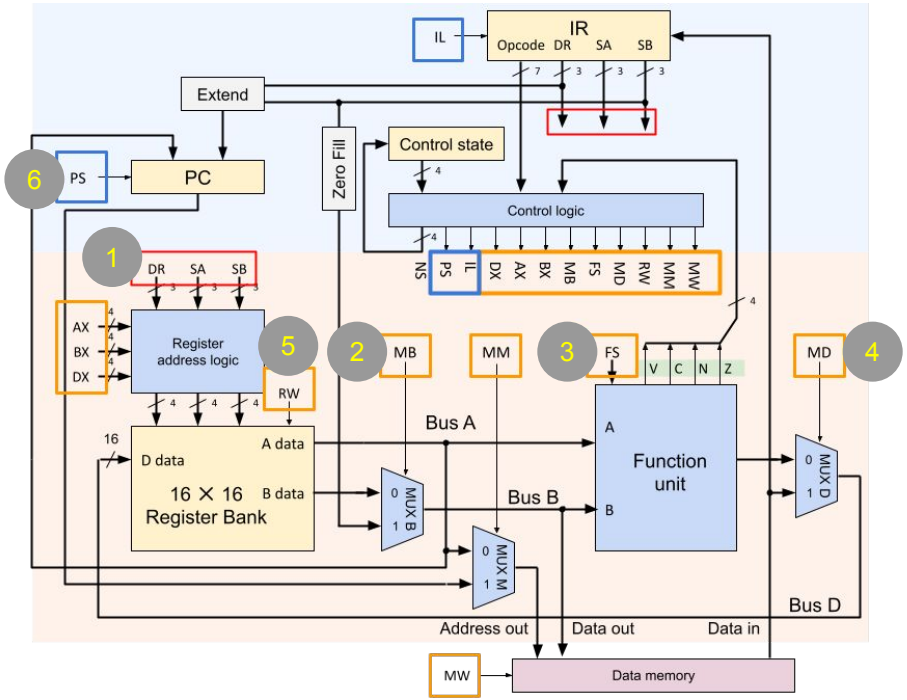
The control logic ("instruction decoder") sets the control signals so that they enable the register transfers and micro-operations defined by the instruction stored in the IR. PC is also updated.

Example: *"Add immediate value 3 included in the instruction to register R3 and save the result in R0"*

```
ADI R0,R3,3: 1000010 000 011 011
                ADI   R0  R3   3
```

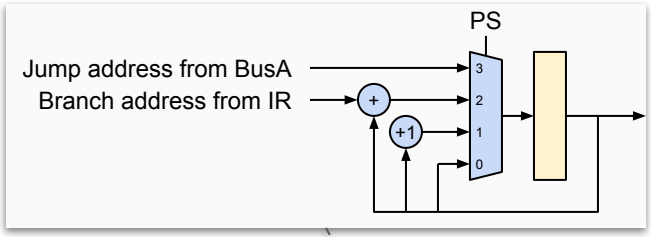
Active controls: 1) Register selects (DR=0, SA=3), 2) MUX B select (MB=1), 3) function select (FS="ADD"), 4) MUX D select (MD=0), 5) register write enable, 6) program counter mode select (PS="increment by one"). The effect is:

R0 ← R3 + IR[2:0], PC ← PC+1

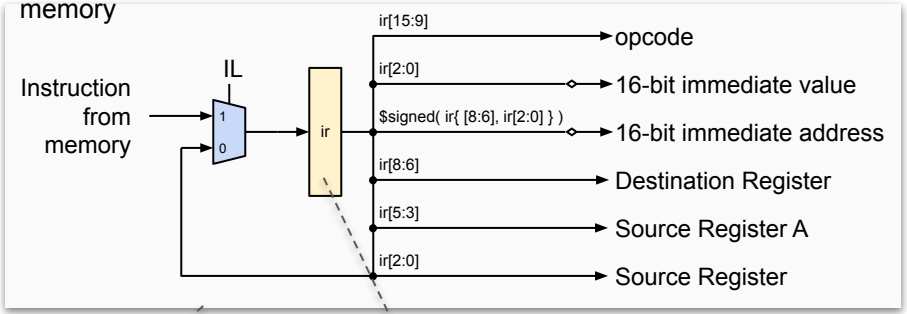


The Datapath of an ISP is Fairly Regular. The Controller is Interesting.

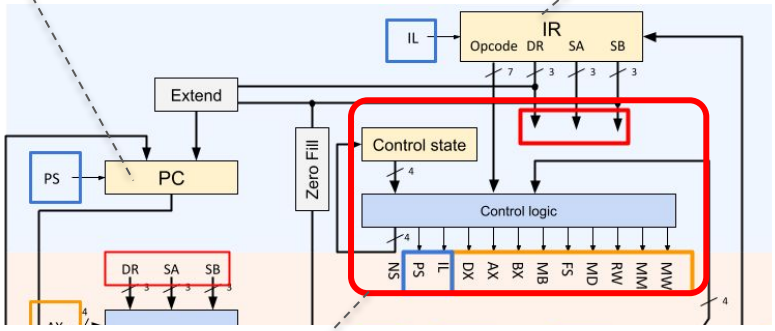
The **Program Counter** (PC) contains the memory address of the next instruction.

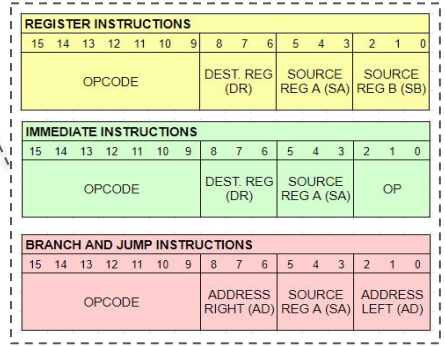


The **Instruction Register** (IR) stores the instruction fetched from memory



The **FSM** decodes control signals for the PC, IR and the datapath. The decoding logic of the FSM can be very complex.

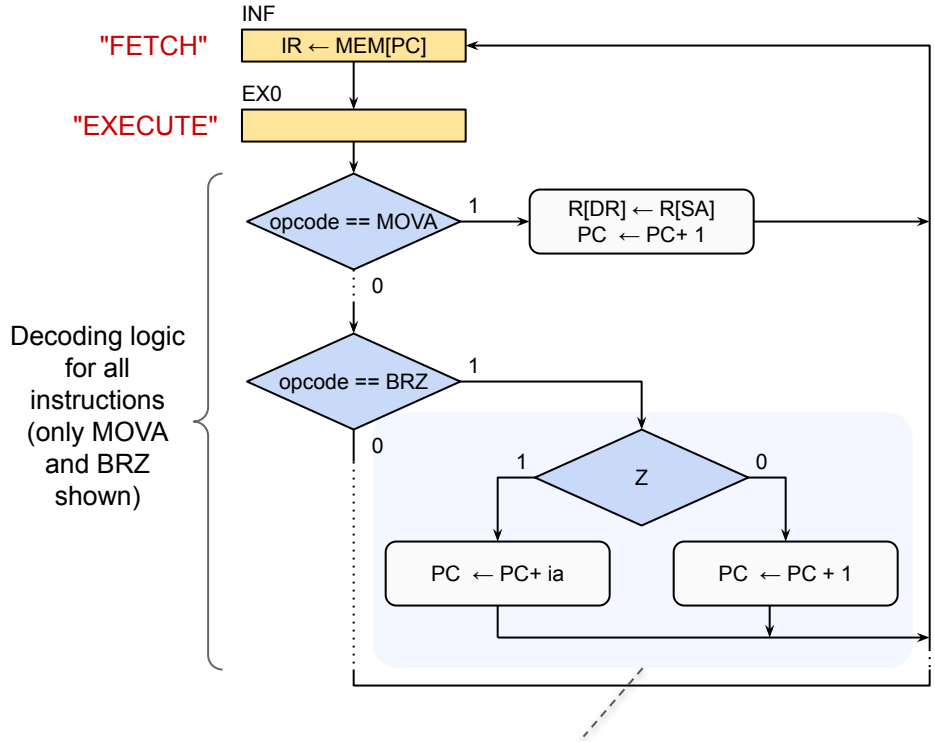




Instruction formats

Control FSM of the ISP

- In the "FETCH" state (INF), the FSM enables loading of data from memory to the IR (sets IL=1, MW=0 and controls muxes)
- In the "EXECUTE" state (EX0), the FSM controls the datapath and PC so that the register transfers and operations defined by the instruction are execute.
- The number of states can be very small (like 2), but the decoding logic can be very complex
- If the ISP has multi-cycle instructions, more states are needed.



Here you can see how status data coming back from the datapath (the Z flag) can be used to determine the controller's actions. (ia = immediate address)

Summary

- The FSMD architecture is a good starting point for any design project
- Despite the clear principle, actual design is not easy in practice as the algorithm can be complex and the constraints can challenging
- Only experience and maybe high-level synthesis tools can help
- More advanced architecture principles exist