# Computer Organization
(Architecture Design Methodology)

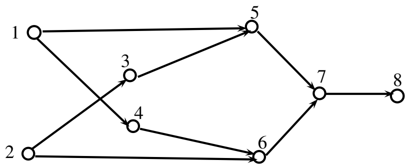
KR Chowdhary
Professor & Head
*Email: kr.chowdhary@gmail.com*
*webpage: krchowdhary.com*

Department of Computer Science and Engineering
MBM Engineering College, Jodhpur

November 14, 2013

## Design Methodology

- Objective: To design a information processing system, with a function to transform a set $A$ of input information items (program and its data sets) into output information $B$; the transformation is $f : A \rightarrow B$. And $f(a) = b$, if $f$ maps $a \in A$ to $b \in B$. Also $b \leftarrow f(a)$.
- If input of truth table is $a$ then $f(a)$ is output.
- System Modeling: System is modeled using directed graph, with set of objects $V$ (vertices) and set of edges (ordered pairs of nodes).



## Design Problem

- Data and Control: Divides the system into: (1) data processing and (2) control. The CPU is as (1), and memory content as (2). $f : A \rightarrow B$ can be decomposed into *data* and *control* part. Let $A \subseteq A_c \times A_d$, $B \subseteq B_c \times B_d$, $(c, d$ for control and data).
- Also, $f_c : A_c \rightarrow B_c$, $f_d : (A_d \times A_c) \rightarrow B_d$.

## Data and control

Figure below is a idealized portion:

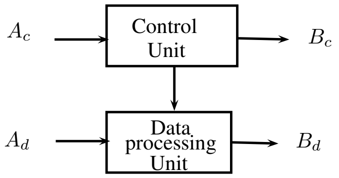
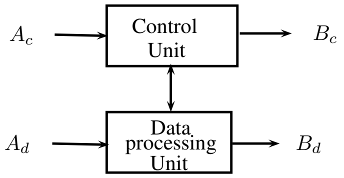


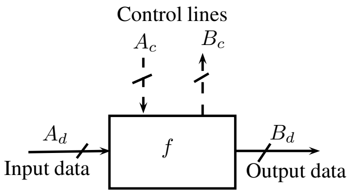
Figure below is usual partition



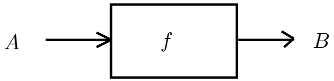
- The boundary between data and control is arbitrary and subjective (same signals may be viewed as control, and other time as data).
- System classification: by the way the control function is implemented. Control unit($f_c$) is fixed logic circuits, then system is hard-wired,
- If $f_c$ is implemented by storing information, then control function resides in software, i.e. programmable, called microprogrammed control.

## Descriptive Methods

There is no standard notation for distinguishing data and control signals.



Component in a block:



- The behaviour of machine is as a finite state machine, determined by *states*. Total input state $A$ is in two parts: (1) signals $A$ applied from external source, and (2) internal states $Y$ (from information stored in the machine).
- Total output states $B$: (1) output part $Z$, (2) new internal part $Y$, i.e.,

  $f : X \times Y \rightarrow Y \times Z$. If $(x, y) \in X \times Y$, $(y', z) \in Y \times Z)$, then $y', z \leftarrow f(x, y)$.
- $f$ is *state-table*, with rows and columns are $Y, X$, and its entries are total $Y \times Z$. Entries in state-table are represented by *state-diagram*. Nodes are internal states and edges are transition between them. (implies that it is finite state machine.)

## Descriptive Methods

- Other way to represent state behaviour: state transitions/equations:

$$
\begin{aligned}
y_1, z_1 &\leftarrow f(x_0, y_0) \\
y_2, z_2 &\leftarrow f(x_1, y_1) \\
&\ldots\ldots\ldots\ldots\ldots\ldots \\
y_p, z_p &\leftarrow f(x_r, y_s)
\end{aligned}
$$

- Each transition can be viewed as statement about the machine (in grammatical terms - statement of what machine does or should do)
- Transitions are viewed as *instructions* (of smallest size)
- Sequence of these instructions = program
- Above is basis for: *design of languages, hardware description languages, register transfer languages.*

## Design Levels

| Design level | Components | IC type | unit of Info |
|---|---|---|---|
| Processor | CPU, IOPs, memories, IO devices | LSI, VLSI | Blocks of words |
| Register | Register, combinational, ckts simple sequential ckst | MSI | Words |
| Gate | logic gates, flip-flops | SSI | Bits |

- Computer centre manager's view: Processor level
- Assembly language programmer's view: Register level
- Gate level: Classical twitching theory

## Hierarchical Design

- To be Low / high level? Should we proceed L to H or reverse?
- One component at level $L_i$ is equal to a network of components at level $L_{i-1}$
- Specify the processor level components, then register level components, then gate level
- Components at each level should be independent of other levels, and standard interfaces.
- It is most natural to proceed from higher level to low level for design.
  1. Specify processor level structure
  2. Specify register level structure (for 1 above)
  3. Specify gate level structure (for 2 above)
- Theoretical design exists at gate level, but higher level it is more of an art.

## Gate Level Design

- Switching theory deals with binary variables $\{x_i\} \in \{0, 1\}$
- A combinational switching function:
  $z = B^k \rightarrow B, B \in \{0, 1\}, B^k$ is $2^n$ binary tuples
- Combinational circuit can be designed by truth tables, I/P $=$ $(x_1, \ldots, x_n)$, O/P $z(x_1, \ldots, x_n)$

| $n$ | $x_0$ | $y_0$ | $c_i$ | $z_0$ | $c_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 2 | 0 | 1 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 | 0 | 1 |
| 4 | 1 | 0 | 0 | 1 | 0 |
| 5 | 1 | 0 | 1 | 0 | 1 |
| 6 | 1 | 1 | 0 | 0 | 1 |
| 7 | 1 | 1 | 1 | 1 | 1 |