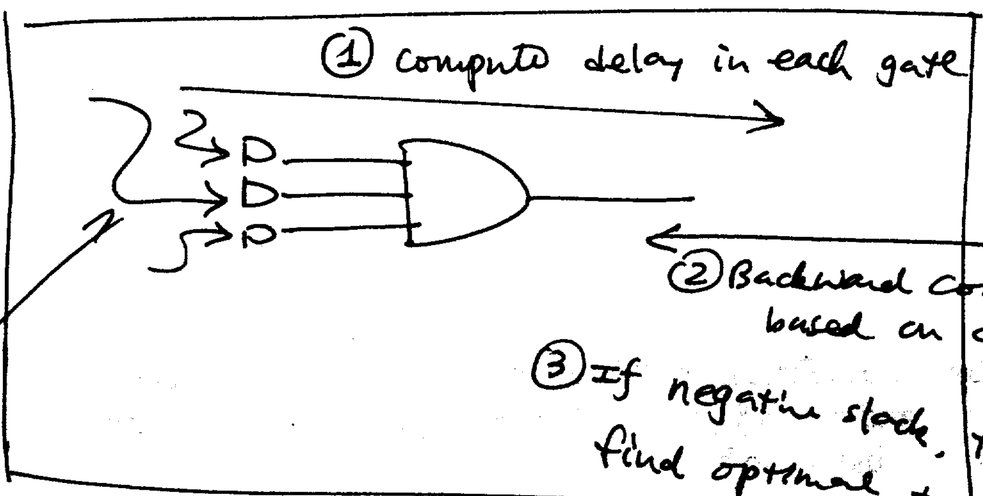
Transistor re-ordering algorithm.

Step I.



① compute delay in each gate

② Backward compute slack based on clock cycle,

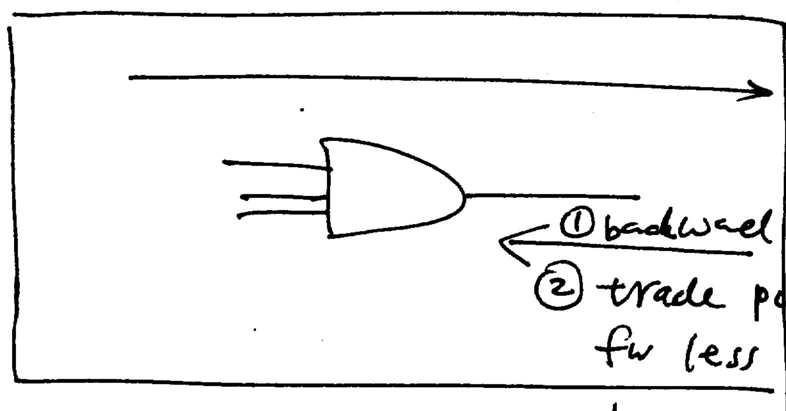
③ If negative slack, then find optimal transistor ordering. Reorder it

The latest arrival signal connected to the input with smallest delay.

④ Compute forward to update the delay.

⑤ Repeat ② ⟶ ④ until no negative slacks

Step 2:



① backward tracing.

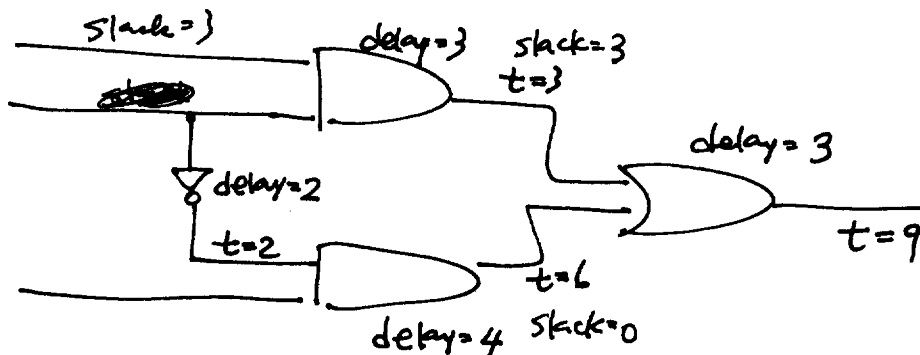
② trade positive slack for less power consumption by transistor re-ordering

③ compute slack, make sure no neg. slack

Experimental Results

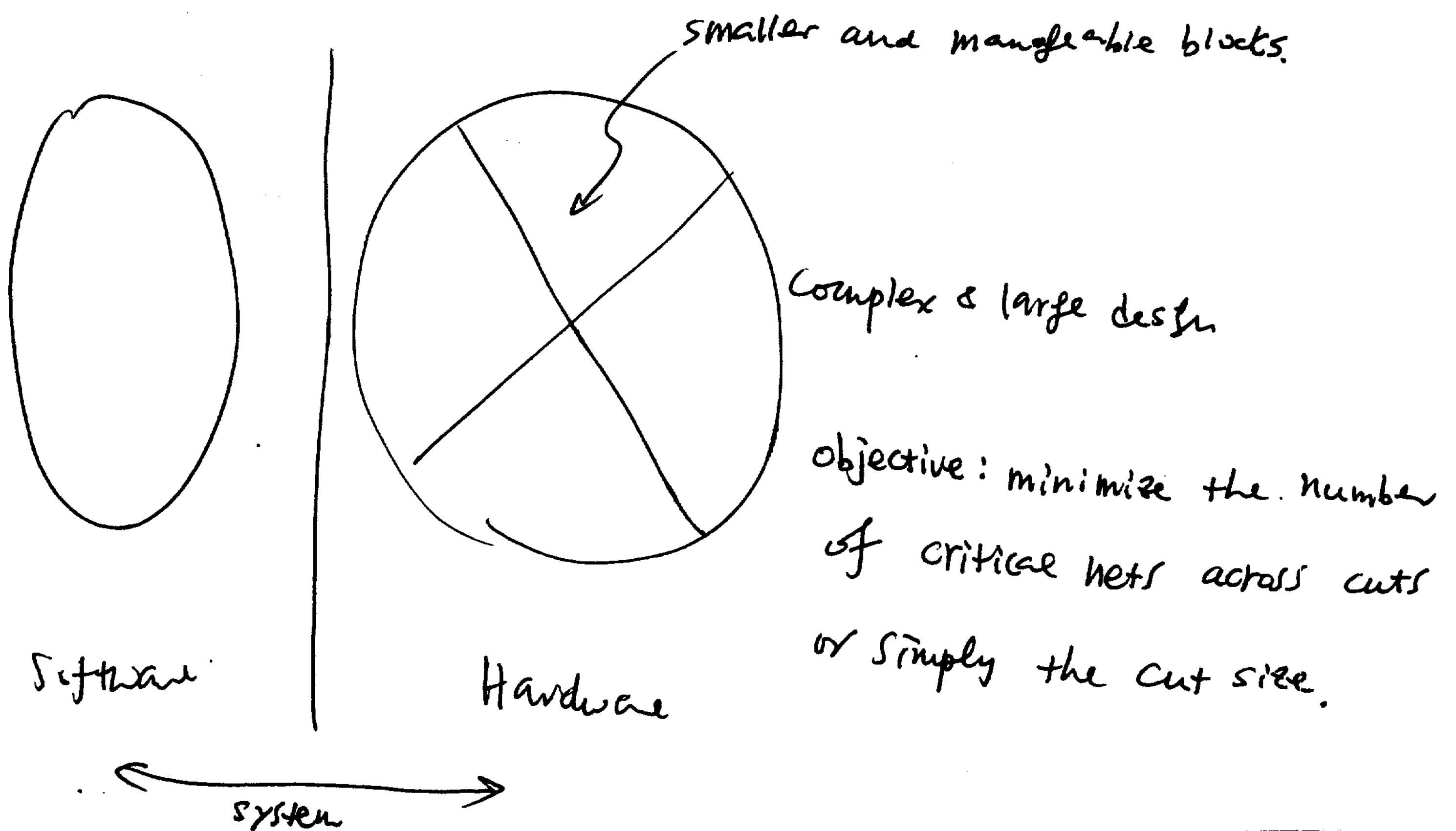
ckt	delay reduction	power reduction
vg2	9.5%	7.3%
rd84	6.9%	6.5%
alu4	8.2%	4.3%
	⋮	

Slack calculation



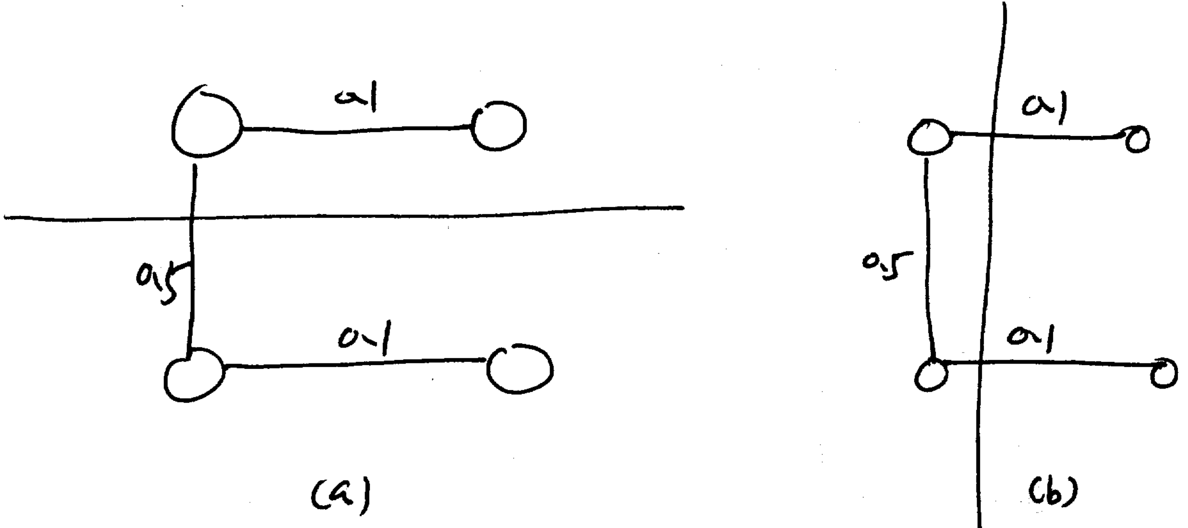
Interconnect Layout optimization

- After device sizing, the next target is interconnect.
- In deep submicron design, the interconnect capacitance dominates the gate capacitance
  ∴ Interconnect design is crucial.
- Discuss the interconnect issue on circuit partitioning, node clustering, placement and routing.
- Discuss wiresizing for delay and power optimization.
- **Circuit partitioning.**



- For low-power design, average switching frequency of the nets are important.

Example:



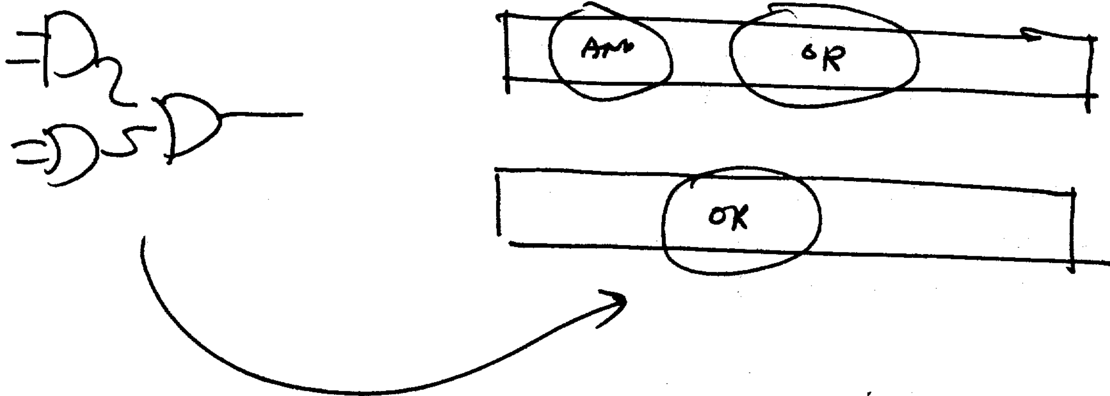
Both partitions are balanced (contains equal # of nodes).

(b) has smaller capacitive power dissipation

∵ lower switching activity across the two blocks.

- First, ~~assign~~ estimate switching activity for each net. Then, apply a partitioning method with the objective of minimizing switching capacities.

- partitioning will be further discussed in Design Automation of VLSI.



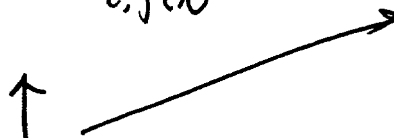
- Map the gates in a circuit onto positions on a layout surface.

- Conventional method: try to minimize the total net length of the final placement, meeting various delay constraints.

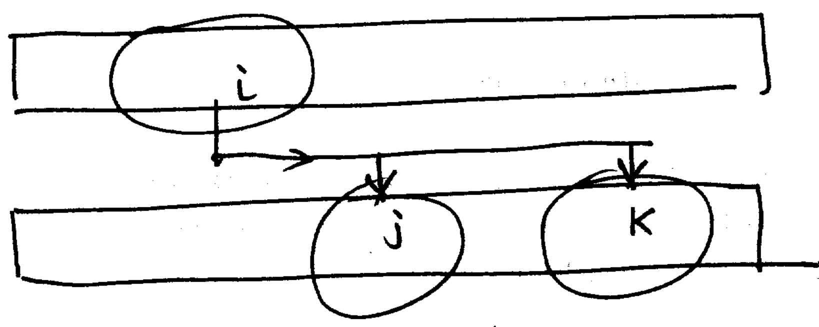
- For low-power placement: try to minimize the total effective switching capacitance, instead of total wirelength (capacitance), subject to timing constraints.

objective function:

$$L_2 = \sum_{\text{Net } N} f_N^2 \sum_{\substack{\text{cells} \\ i,j \in N}} \left\{ (x_i - x_j)^2 + (y_i - y_j)^2 \right\}$$



Quadratic net length model.



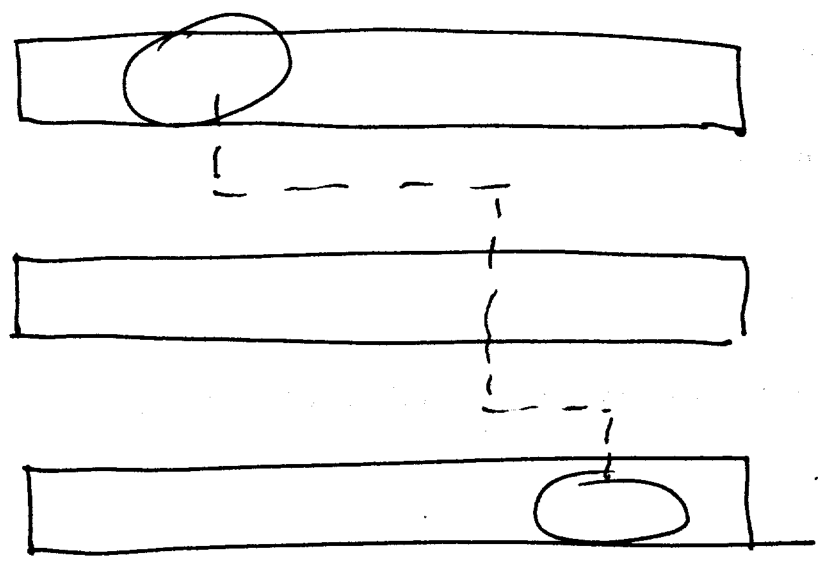
$$f_N^2\left[(x_i-x_j)^2+(y_i-y_j)^2\right]+f_N^2\left[(x_i-x_k)^2+(y_i-y_k)^2\right]$$

- The problem can be even more complex when timing delay is added as a constraint.
- Either the lumped capacitance model, the pathlength delay model or the Elmore delay model can be used to formulate the delay model.
- An average of 10% power reduction obtained when compared with the minimum net length solution.

Routing.

- Routing completes the electrical connection between cells
- Contains global routing and detailed routing.

- Global Routing: Assign each net to a set of routing regions



Objective: minimize the total wire length.

Detailed routing: Determine the actual wiring geometries and layer assignments within each region.

Objective: Minimize the channel height and total wirelength.

- Low-power routing: Minimize the total effective switching Capacitance of all nets while meeting the delay constraints.

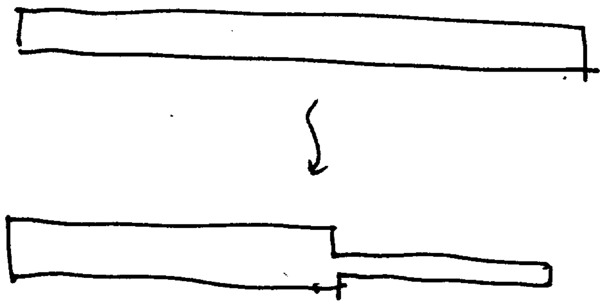
- just concentrate on global routing.

- To minimize the total effective switching capacitance, we must route high switching activity net with minimum wirelength, and allow detour for low activity nets to avoid channel congestion.
- Should avoid detour for critical nets.
- It was reported that only marginal improvement in power dissipation is observed for global routing.
- In deep submicron design, power dissipation due to coupling effect (cross talk) has become more significant.
- Try to find proper spacing and wire segment assignment ordering

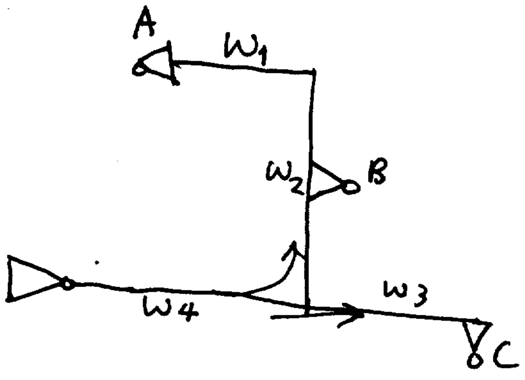
↕ Keep wider spacing for high activity nets. and assign them to diffent layers.

Wiresizing optimization.

- partitioning, placement and routing mainly concentrate on interconnect length and switching activity.
- There is room for optimization for a fixed-length interconnect by sizing.



- Very important for deep submicron technology, since the delay of interconnect dominates.
- Definitions:



routing Tree: T

sink (T) : the set of sinks in T (A, B, C)

W: the wiresizing solution (wire width assignment for each segment of T).

$t_i(W)$: the Elmore delay from the source to sink s_i under W.

Objective function for Wiresizing:

$$t(W) = \sum_{s_i \in sink(T)} \lambda_i t_i(W)$$

λ_i: measures the criticality of sink s_i.

E.g. large λ_i for timing critical sinks.

Tradeoff between routing area (also interconnect capacitance and power dissipation) and performance.

$$\alpha \, Area(W) + \beta \cdot t(W)$$

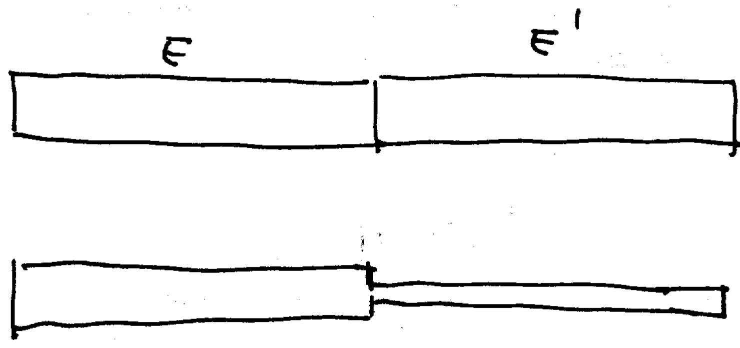
α, β are weights to indicate the relative importance of area and performance.

- Consider either $t(W)$ or $\alpha \, Area(W) + \beta \, t(W)$, there are three important properties for Wiresizing:

  Monotone, separability, and dominance

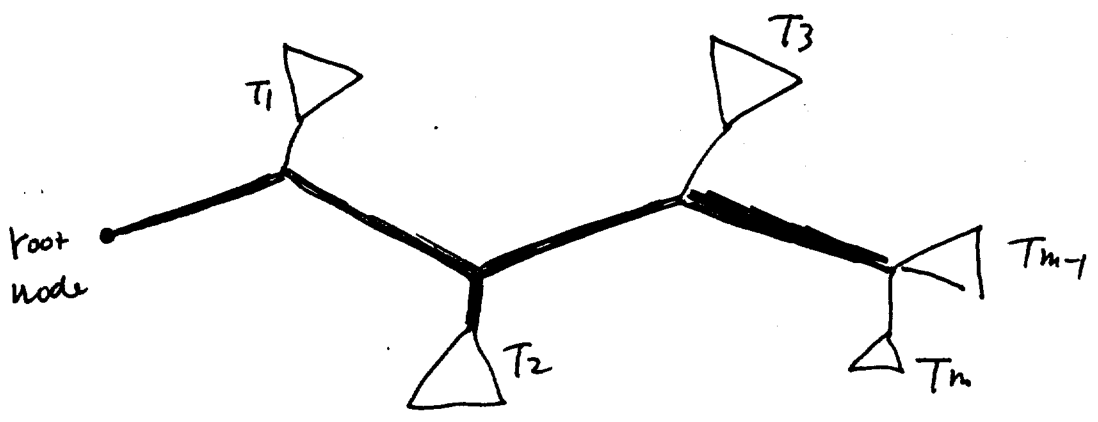
- There exists an optimal wiresizing solution which is

Monotone: Given any pair of wire segments E and E' such that E' is the downstream of E, the width of E is no smaller than that of E'.



- A very natural result since E drives more devices, so should have larger width, in general.

Separability: If the width assignment of a path P originated from the source is given, the optimal width assignment for each tree branching off P can be carried out independently.

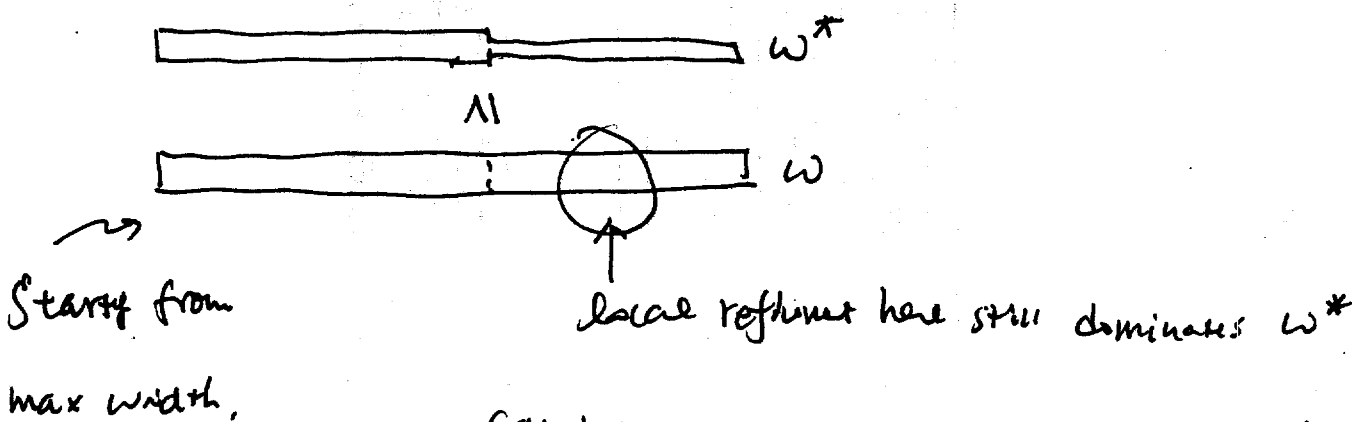


- In the above figure, based on the separability property, the optimal width assignment for $T_1, T_2, \dots, T_m$ can be computed independently. if the width assignment of P is given.
- This property is extremely important in simplifying the wiresizing problem.

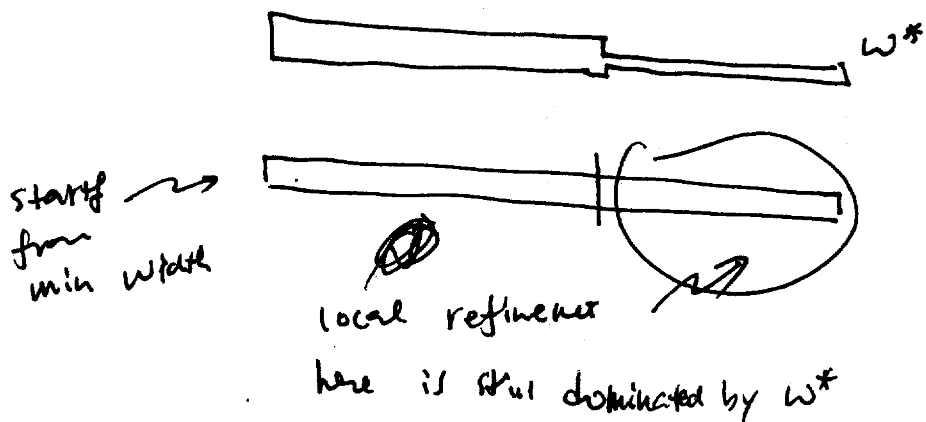
Def. Given two wire width assignments W and W', W dominates W' if for any segment E, the width assignment of E in W is greater than or equal to that of E in W'.

Def. Given a routing tree T, a wire width assignment W on T, and any particular segment $E \in T$, A local refinement of W with respect to E is the operation of optimizing the width of E while keeping the wire width assignment of other segments in W unchanged.

Thm: Let w^* be an optimal width assignment. If a width assignment w dominates w^*, then any local refinement of w still dominates w^*. Similarly, if a width assignment w is dominated by w^*, then any local refinement of w is dominated by w^*.



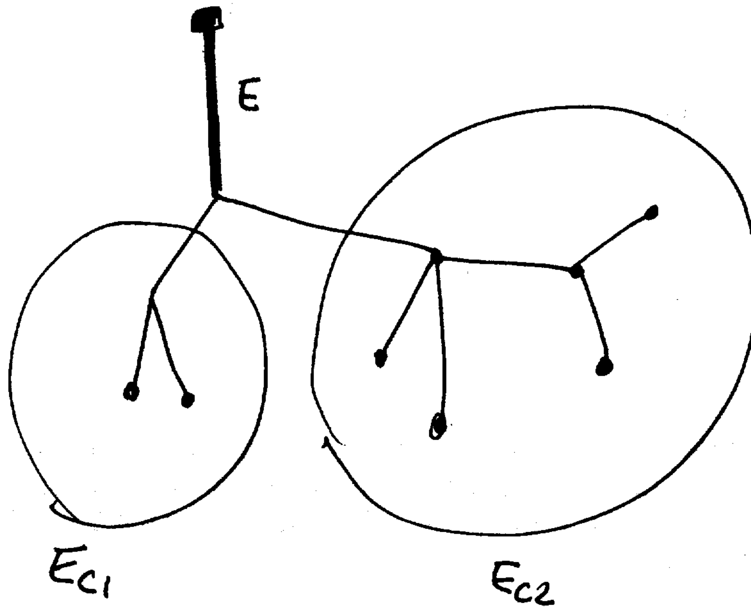
Can be used to find the upper bound of the optimal wire width.



∴ can be used to find the lower bound of the optimal wire width.

Based on monotone and separability properties:

Optimal wiresizing Algorithm:



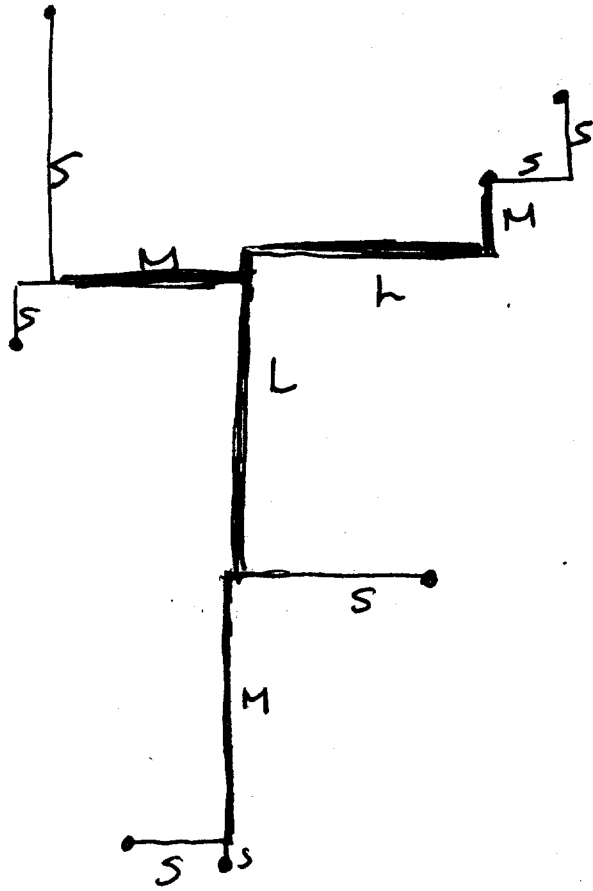
How to determine the width of E?

possible widths: $w_1, w_2, \ldots, w_r$.

For each possible width assignment w_k of E ($1 \le k \le r$), we determine the optimal assignment for each subtree $E_{c1}, \ldots E_{cz}$. independently by recursively applying the same procedure to each E_{c_i} with $\{w_1, w_2, \ldots, w_k\}$ as the set of possible widths. The optimal assignment for E is the one which gives the smallest total delay.

(using Elmore delay model)

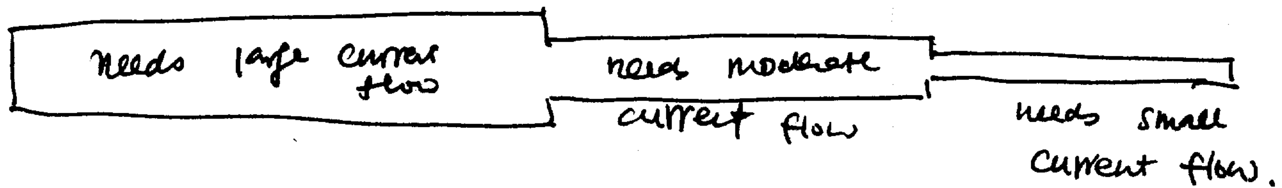
A possible optimal wiresizing.



3 wiresizes

L. M. S.

- Without this technique, designers generally use conservative design: End up using oversized, uniform-width wires
  - ∴ Waste too much area & power, causes longer signal delay
- Optimal wiresizing can be used to "taper" wires for area and power dissipation without performance degradation



- A computer simulation result demonstrates:

  area reduction: 10% compared with manually tappered design.
  (reduced by)

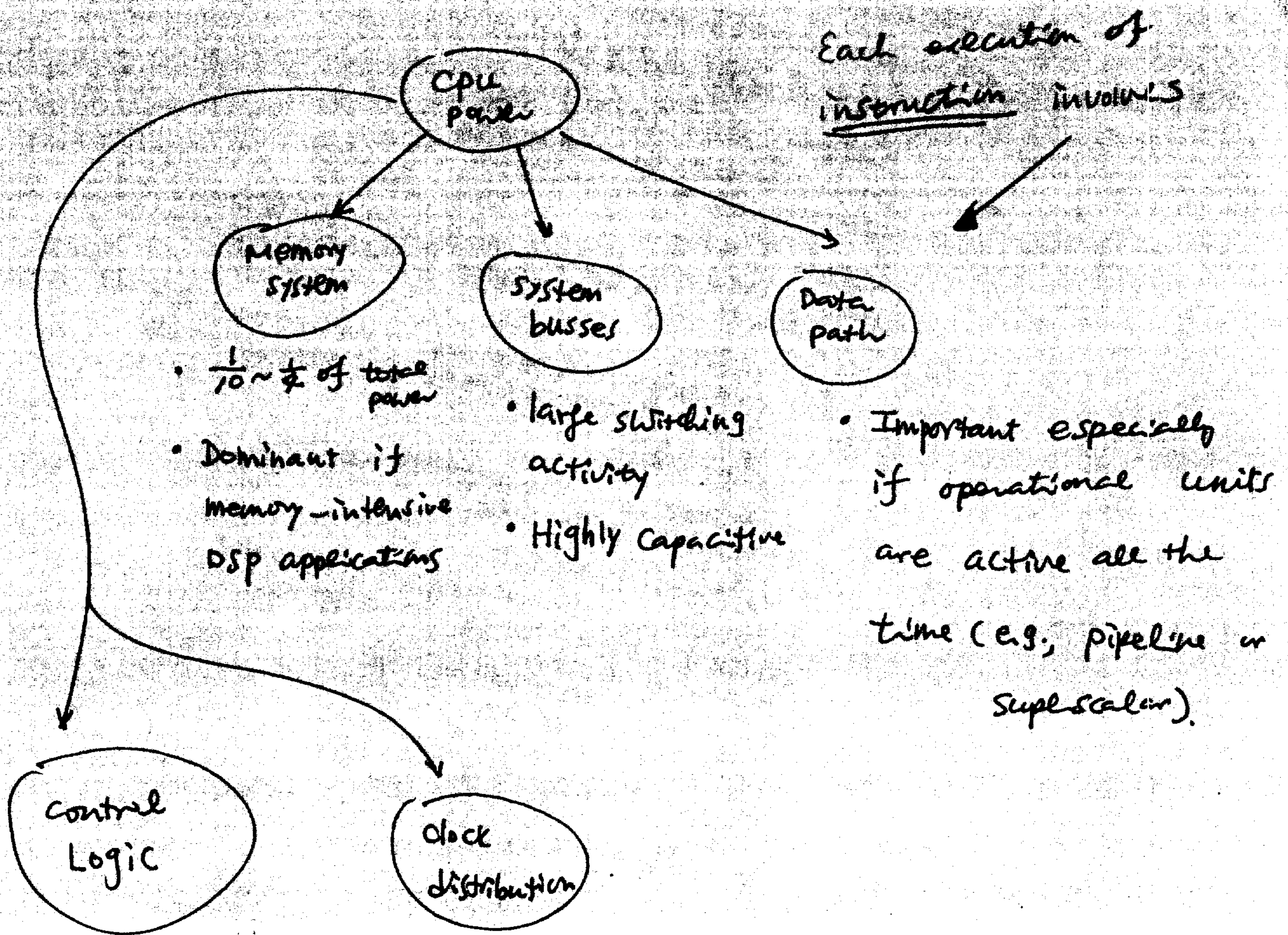
  clock power reduced by 6%.

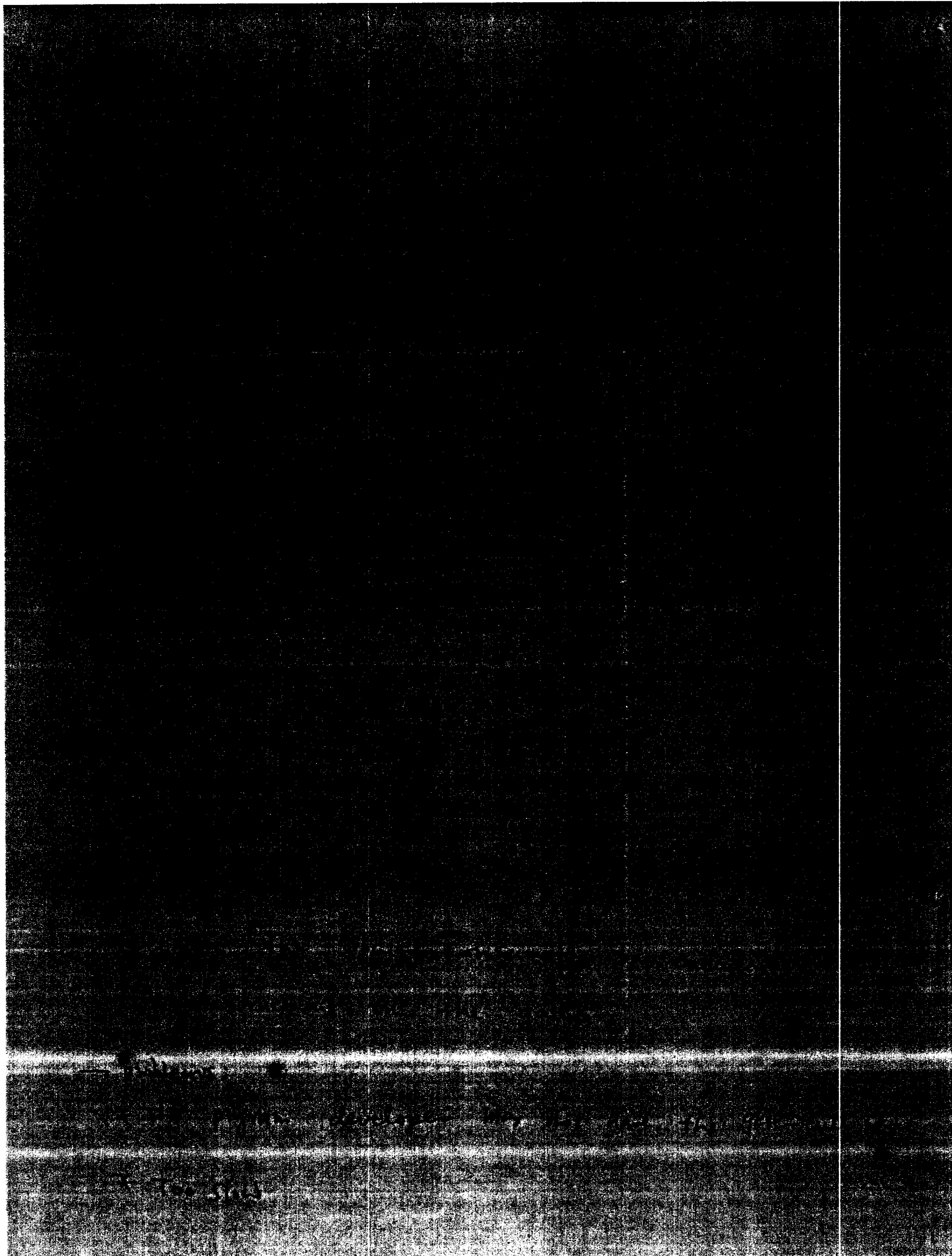
  average delay is delay is reduced by 3%.

Software Design for Low power

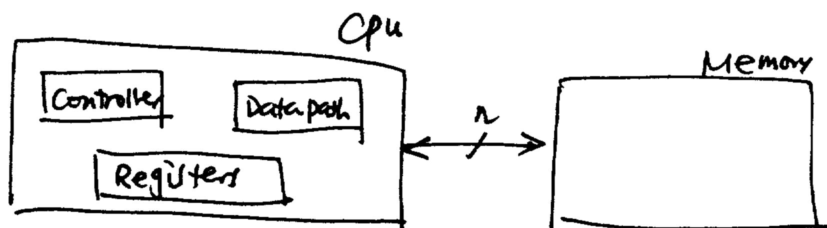
- Major emphases:
  1. Minimize memory accesses
  2. optimal selection and sequencing of machine instructions
  3. exploit the low power features of some processors.

- Sources of software power dissipation





• Architectural-Level power Estimation.



Model: ① power dissipation at each major component

② which components will be activated by the program.

— Less precise, but much faster than gate-level estimation.

• Bus Switching Activity

- use bus activity to represent the overall switching activity in a processor

— Requires knowledge of:

* Bus architecture, * op-codes for the instruction set,

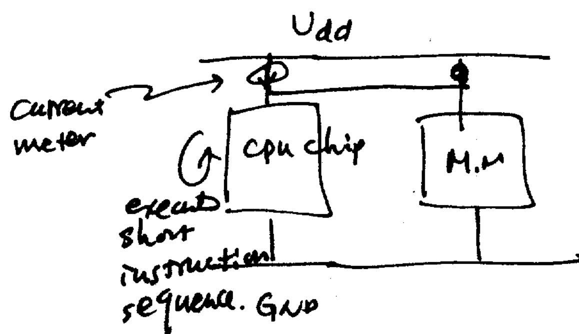
* representative input data to a program,

* program code and data mapping to M.M.

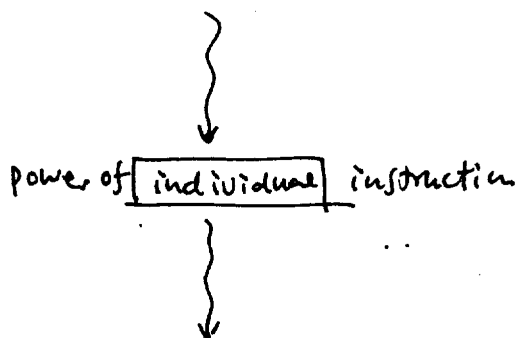
* Simulation for switching at different buses.

• Instruction-level power analysis.

- Use an empirical method for power characterization of short instruction sequences.
- Then, use the results to estimate the power dissipation of a program.
- Basic idea

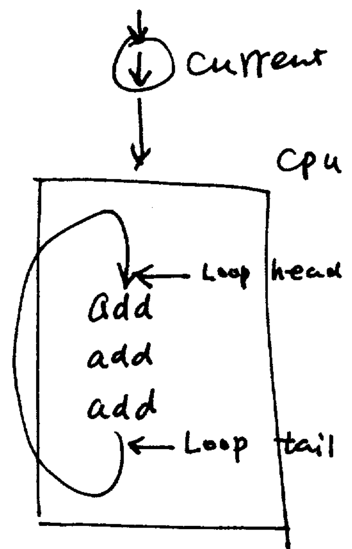


- Choice of instruction sequences for characterization is critical
- Try to estimate the Base cost of each individual instruction.
- Base cost: The portion of power that is independent of the prior state of the processor.



- avoid estimating pipeline stalls, cache misses, bus switching ... due to consecutive instructions - - - -.

- How ?



Give several instances of the instruction and execute an infinite loop.

- The loop should be as long as possible

  minimize estimation error due to loop overhead.

- The loop cannot be too long.

  Avoid cache misses.

- Power ~~Energy~~ : power supply voltage × average current draw.

- The effect of prior processor state should also be considered.

  Exaple: Pipeline stall, buffer stall, cache misses.

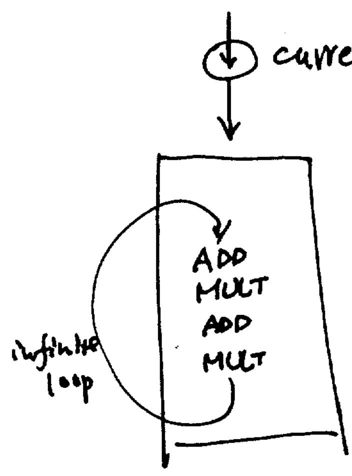
  Called Circuit state effect.

ADD A ← B, C

MULT C ← A, B

① Instruction (op-code) bus switch ~~change~~
② switch of control signals
③ mode change within ALU
④ Switch of data lines between ALU & registers

- How to measure circuit state effect ?



problem: we are measuring the effect of ADD to MULT. But, it is impossible to separate

ADD → MULT &

MULT → ADD.

- Instruction-level power estimate

$$E_p = \sum_i (B_i \times N_i) + \sum_{i,j} (O_{i,j} \times N_{i,j}) + \sum_k E_k$$

- E_p: overall energy cost of a program
- B_i: Base cost for type i instruction
- N_i: # of type i instructions
- $O_{i,j}$: Circuit state effect of type i instruction followed by type j. ① — ④ above
- $N_{i,j}$: #
- E_k: each occurrence of other effects (e.g. stalls, cache misses..)

- Example:

DLOAD A ← x, B ← y

LOAD C ← z, MULT D ← A, B

ADD A ← C, D

	Base Cost	Circuit state effect			
		ADD	MULT	DLOAD	LOAD MULT
DLOAD	2.37	1.19	- - - -	-	
LOAD, MULT	2.25			1.06	
ADD	0.99				0.99

Table 1 of pp. 439

∴

	Base Cost	Circuit state
DLOAD	2.37	1.19
LOAD-MULT	2.25	1.06
ADD	0.99	0.99
total	5.61	3.24

Each instruction requires 25 ns.

∴ total Energy ~~power~~ = 8.85 PJ.

Ave. power = 8.85 PJ / 75 ns = 118 uW.

- **Automatic analysis process**

① Basic blocks are extracted from a program.

② power for each block is totalled (Base + circuit state effect).

③ The frequency and cost of stalls for each block are estimated and added into the block.

④ The program profiler determines the # of time each block executes + accumulate total cost

⑤ A cache simulator is used to estimate the frequency of cache misses, and add the power to the total program power.

- Software power optimizations

- By Instruction selection & ordering

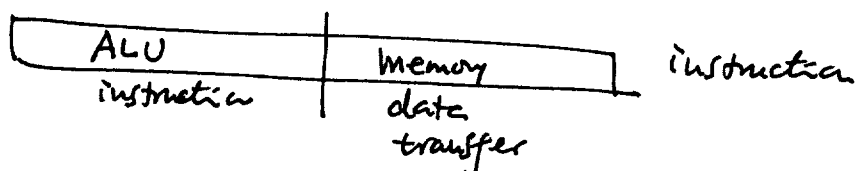
- Cache performance

  Large code and small cache ⟶ frequent cache misses ⟶ high power penalty

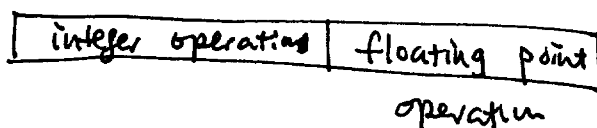
  ∴ should maximize code density for embedded processors.

- Instruction packing

  Example: Fujitsu DSP

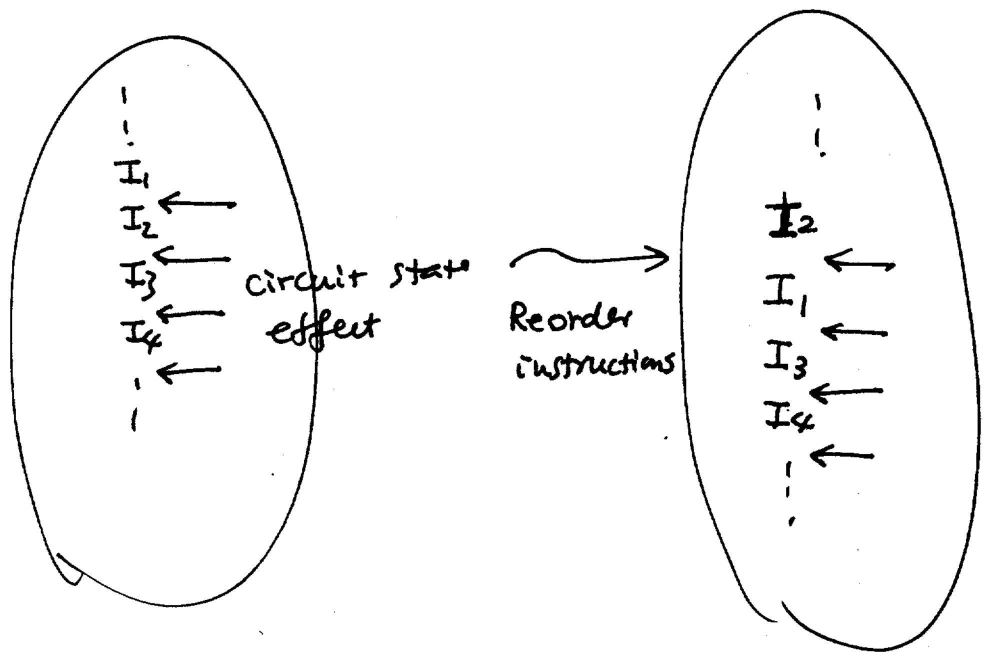


* 50% of energy reduced if each instruction fetch overhead is not duplicated



- This method is also used in VLIW & Superscalar architectures.

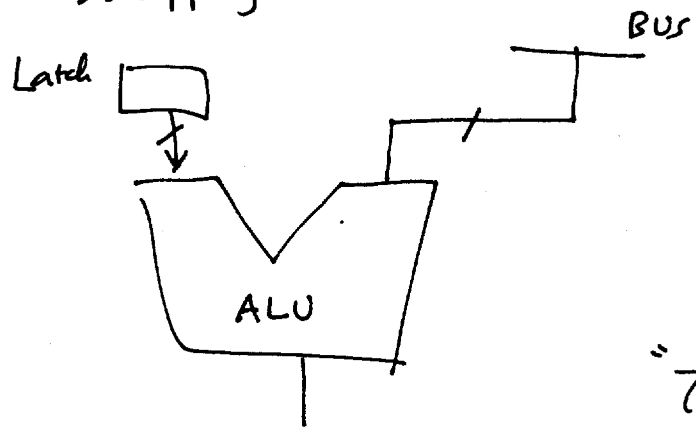
- Instruction ordering



to minimize the circuit state power

* This technique is more significant for DSP's than general-purpose architectures.

- operand swapping

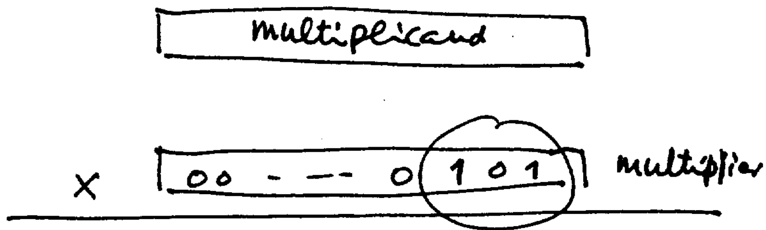


$x+7$

$y+7$

"7" should be placed in the latch.

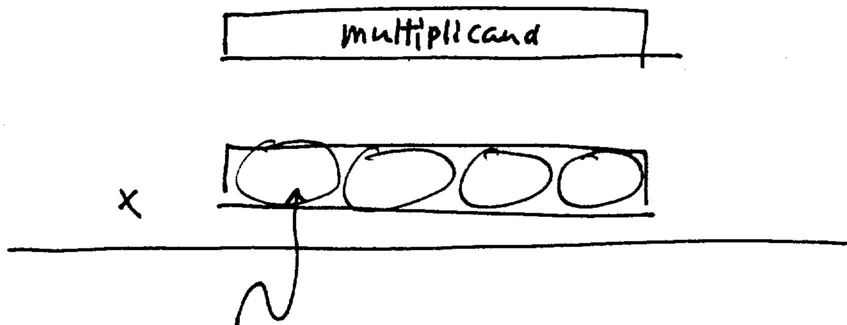
Another example: ~~Booth~~ Shift multiplier



only very limited # of add & shift operations are required.

⇒ Reorder small weight # to the second operand (multiplier)

Booth multiplier



* Different bit pattern determines the # of additions & subtractions
* The # of additions & subtractions ~~for each bit pattern~~ is called recoding weight
* place the value with low recoding weight to the 2nd position.

Minimizing Memory Access Cost

- Memory is both a power and performance bottleneck.

  ∵ It is slow & power hungry.

- Try to
  - Minimize the # of memory accesses required by an algorithm.
  - Make memory accesses as close to the process as possible.

    register ⟶ cache ⟶ RAM
  - Minimize the total ~~number~~ memory required by an algorithm
  - Make the most efficient use of available memory bandwidth.

    e.g., try to use multiple word, parallel loads.

<u>Example</u>:

```
For i=1 to N do
    B[i] = f(A[i]);

for i = 1 to N do
    C[i] = g ( B[i]) ;
```

⟹

```
For i=1 to N do
 begin
    B[i]= f(A[i]);
    C[i]= g (B[i]);
 end
```

↑ If B array is large,
register ⟷ memory transfer
Waste power !!
2N memory transfers

↑ compiler can use a register to store B[i].
N memory transfers only.

Exploiting Low power Features of Hardware

- Software control over power management (instead of gated clock).

* Software designer can use instructions to power-down some components

* Example:

  Standby mode: CPU core is stopped, but other operations are maintained

  Sleep mode: all operations except the real time clock stops.

- Advantage:
  - software designer and compiler can better determine when to remove or slow down a clock.
  - pure hardware power down may make incorrect decision (based on processor activity) ⟶ hurt performance & power ∵ system restoring cost.

- Disadvantage:

  Needs program execution cycles,

  more costly than hardware-controlled power-down.