Microprocessor Power Analysis by Labeled Simulation

Cheng-Ta Hsieh, Kevin Chen and Massoud Pedram

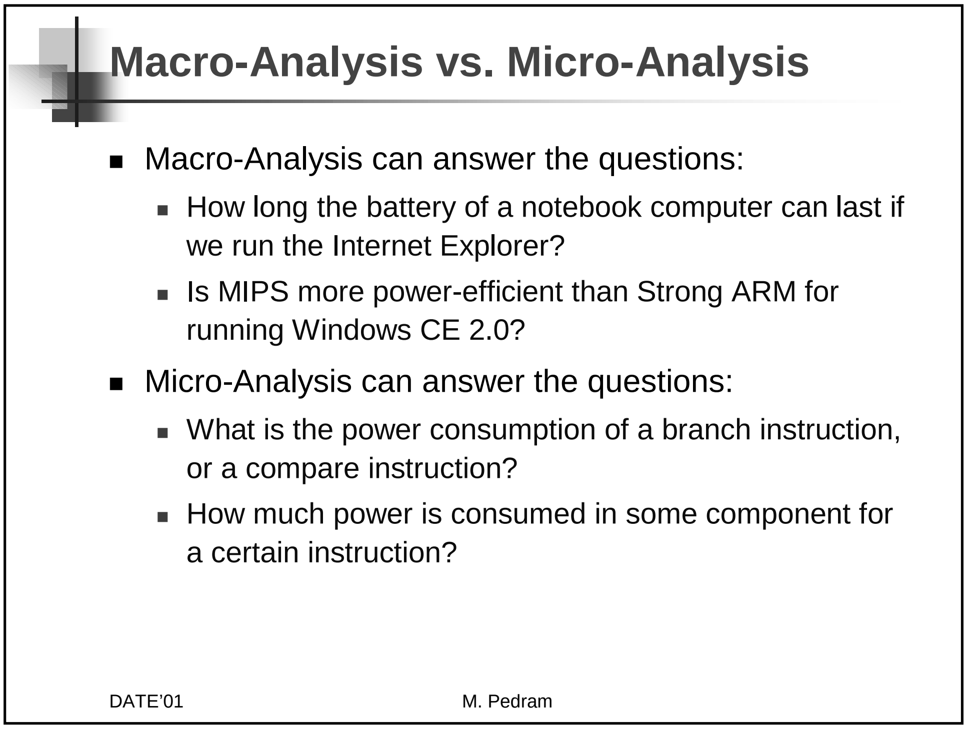
University of Southern California

Dept. of EE-Systems

Los Angeles CA 90089

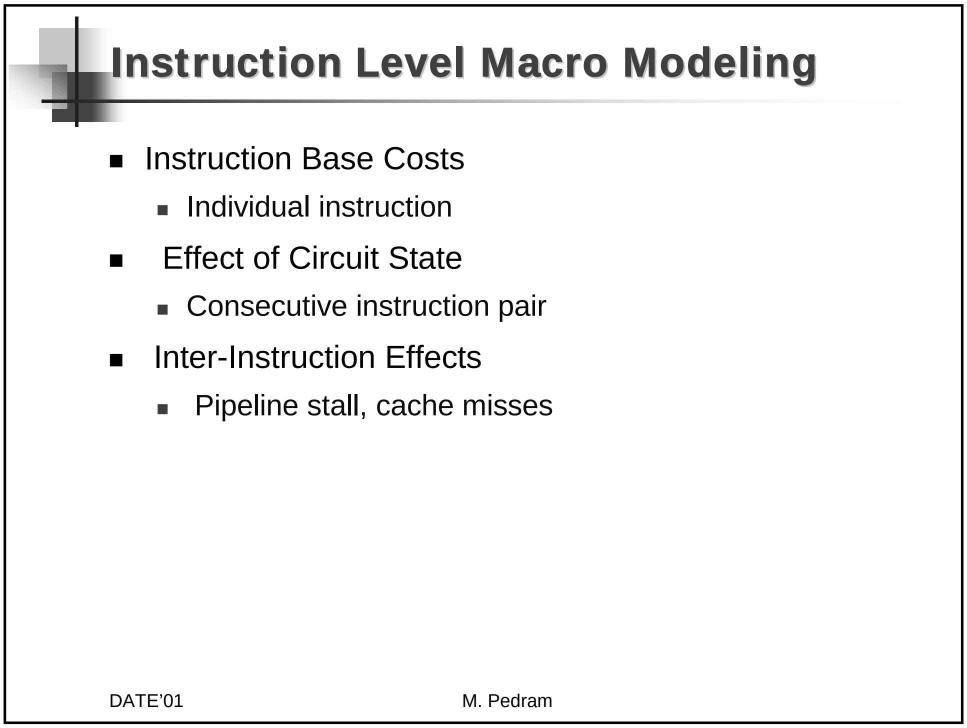
Outline

- Introduction
- Problem Formulation
  - Source and Sink
  - Architecture Patterns
- Propagation Rules
- Generalizations
- Conclusions



Macro-Analysis vs. Micro-Analysis

- Macro-Analysis can answer the questions:
  - How long the battery of a notebook computer can last if we run the Internet Explorer?
  - Is MIPS more power-efficient than Strong ARM for running Windows CE 2.0?
- Micro-Analysis can answer the questions:
  - What is the power consumption of a branch instruction, or a compare instruction?
  - How much power is consumed in some component for a certain instruction?



Instruction Level Macro Modeling

- Instruction Base Costs
  - Individual instruction
- Effect of Circuit State
  - Consecutive instruction pair
- Inter-Instruction Effects
  - Pipeline stall, cache misses

Review of Instruction Level Model

Power Macro Model

=

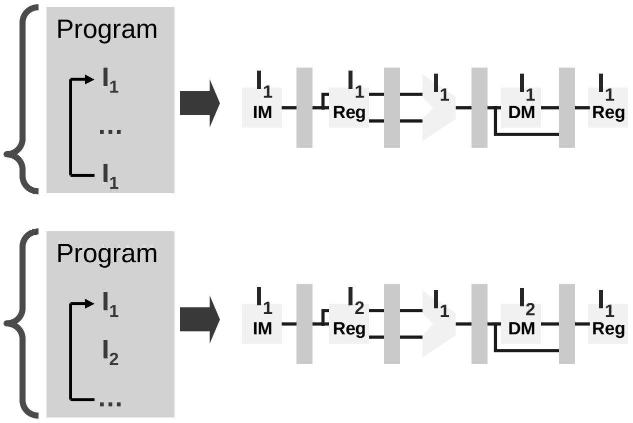
Instruction Base Power

+

Consecutive Instruction Pair Power

+

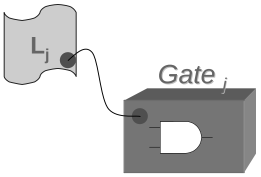
Instruction Stall and Cache Miss Power etc.



$$E_P = \underbrace{\sum_i (B_i \times N_i)}_{\text{Base}} + \underbrace{\sum_{i,j} (O_{i,j} \times N_{i,j})}_{\text{Pair}} + \underbrace{\sum_k E_k}_{\text{Other}}$$

Problem Formulation

- Given *N* gates, $g_1, g_2, \dots g_n$ in a processor and *k* active instructions $I_1, I_2, \dots, I_k$
- For each gate g_i, find an instruction set or a labeling, $L_j=\{I_1,\dots I_m\}$, such that the energy consumption of the gate in the current clock cycle is caused by instructions in L_i
- Calculate instruction power consumption by label propagation

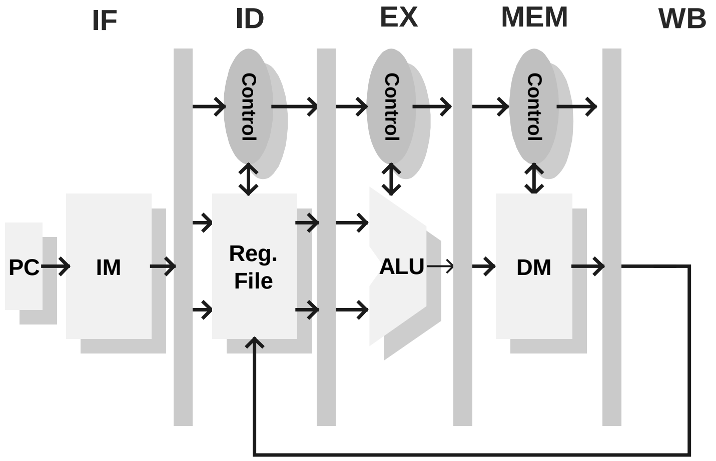


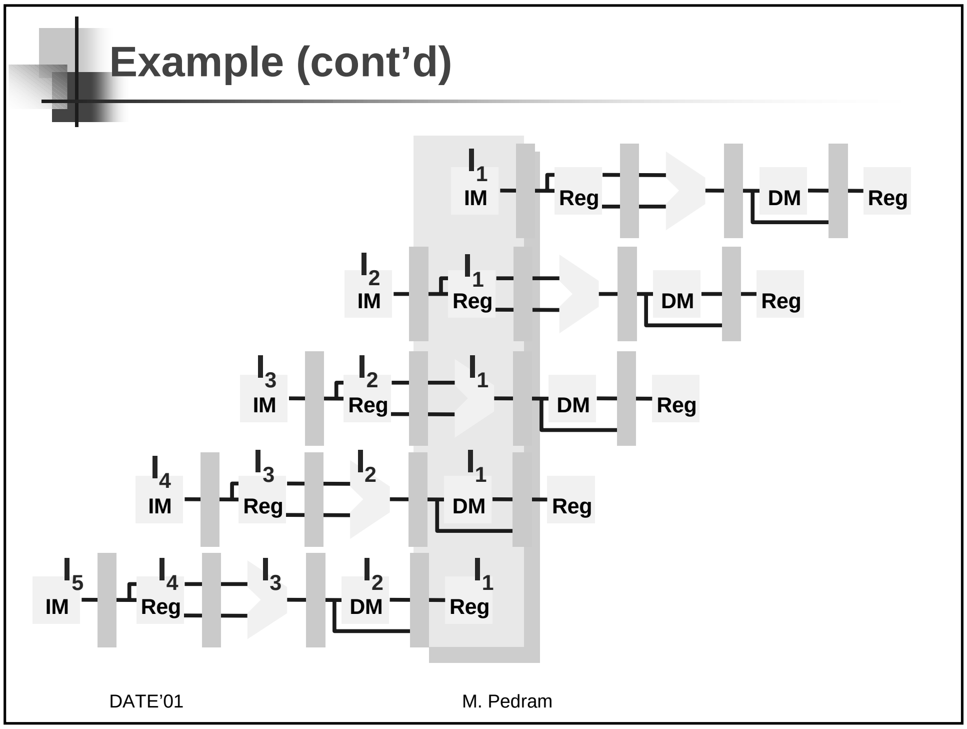
Cycle-Accurate Energy Calculation

- Let *gates(I)* denote the set of label indices that contain instruction *I*
- Energy consumed by instruction *I* in the current clock cycle is:

$$E(I) = \frac{1}{2} \sum_{j \in gates(I)} \frac{1}{|L_j|} C_j V_{dd}^2 sw_j$$

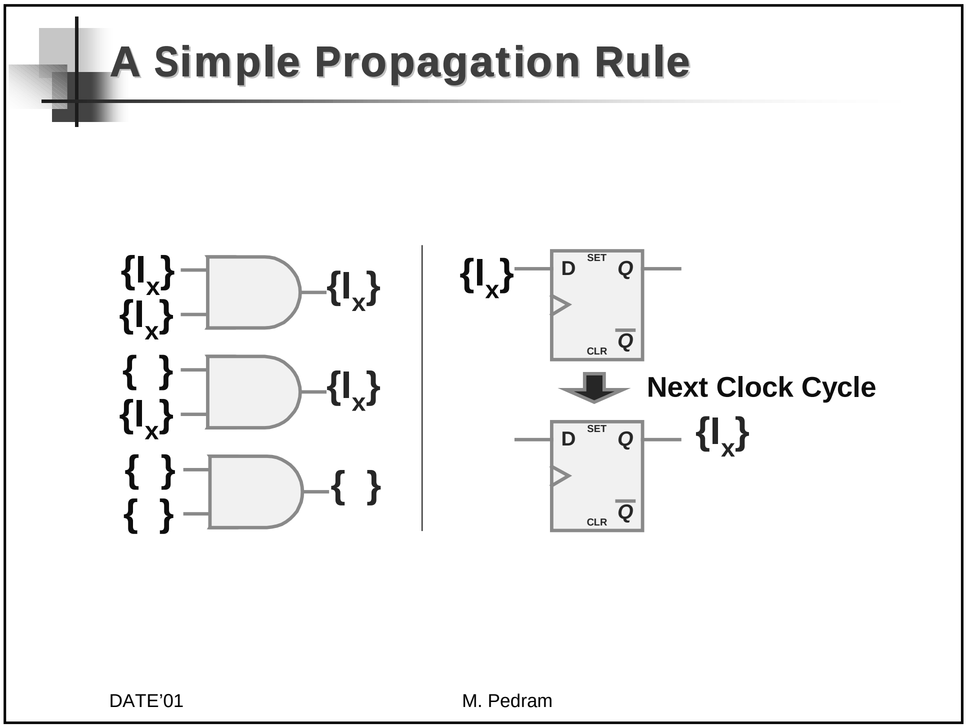
An Example Pipeline





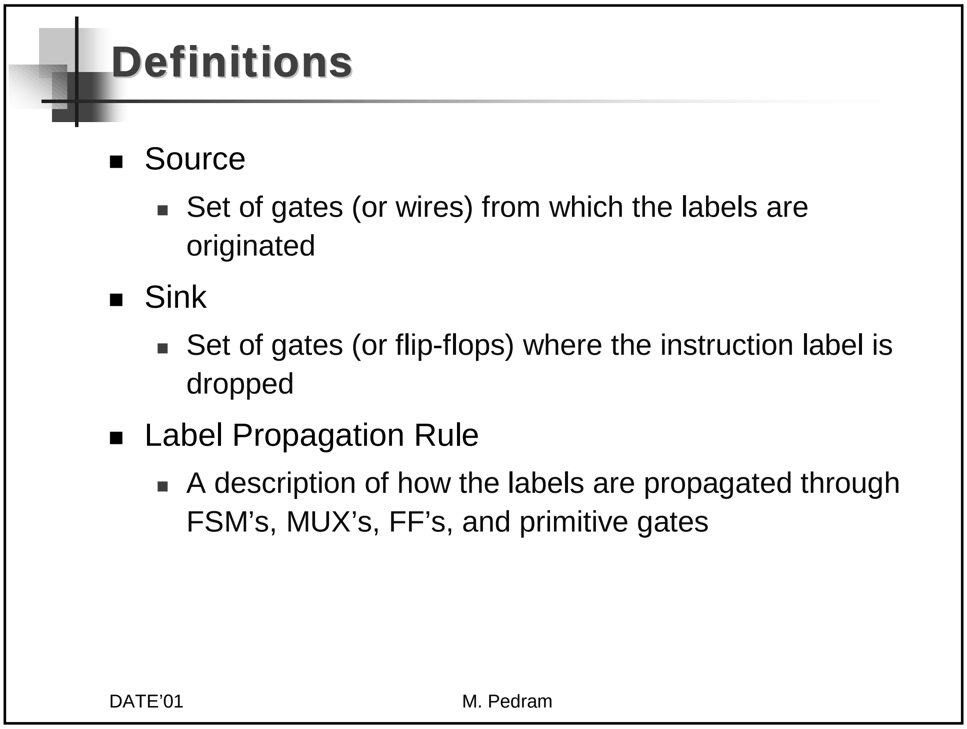
Example (cont’d)

DATE’01 M. Pedram



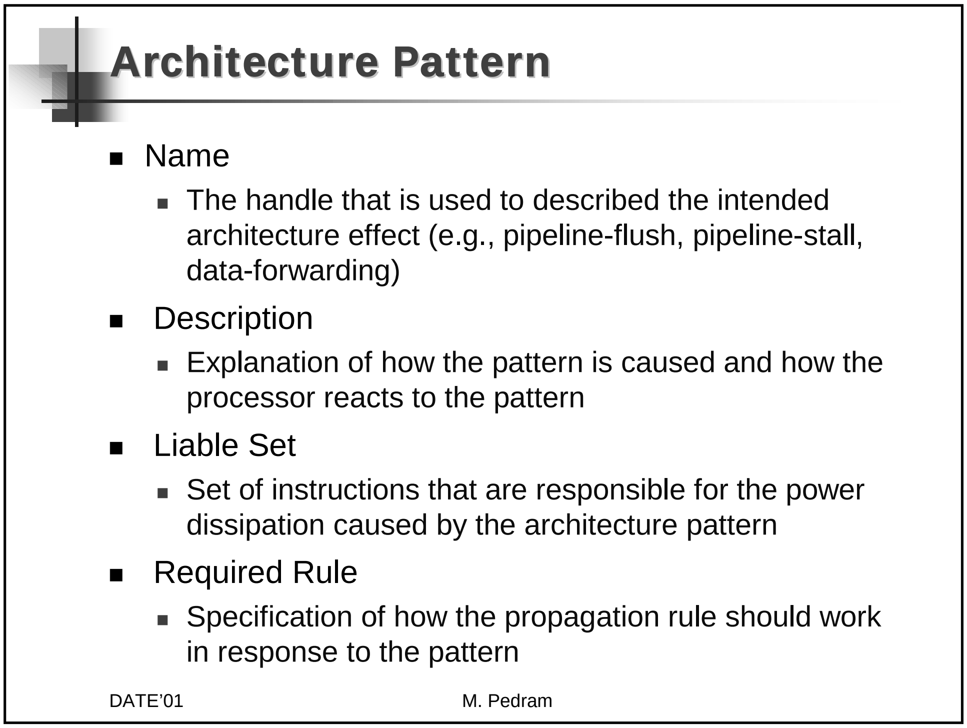
A Simple Propagation Rule

DATE’01 M. Pedram



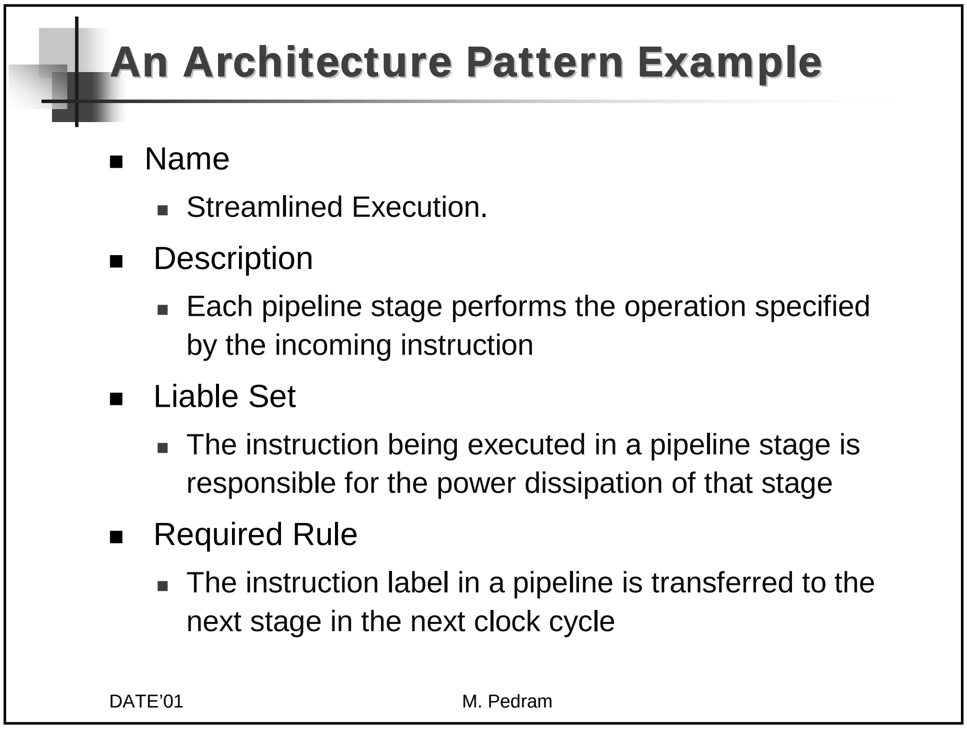
Definitions

- Source
  - Set of gates (or wires) from which the labels are originated
- Sink
  - Set of gates (or flip-flops) where the instruction label is dropped
- Label Propagation Rule
  - A description of how the labels are propagated through FSM’s, MUX’s, FF’s, and primitive gates



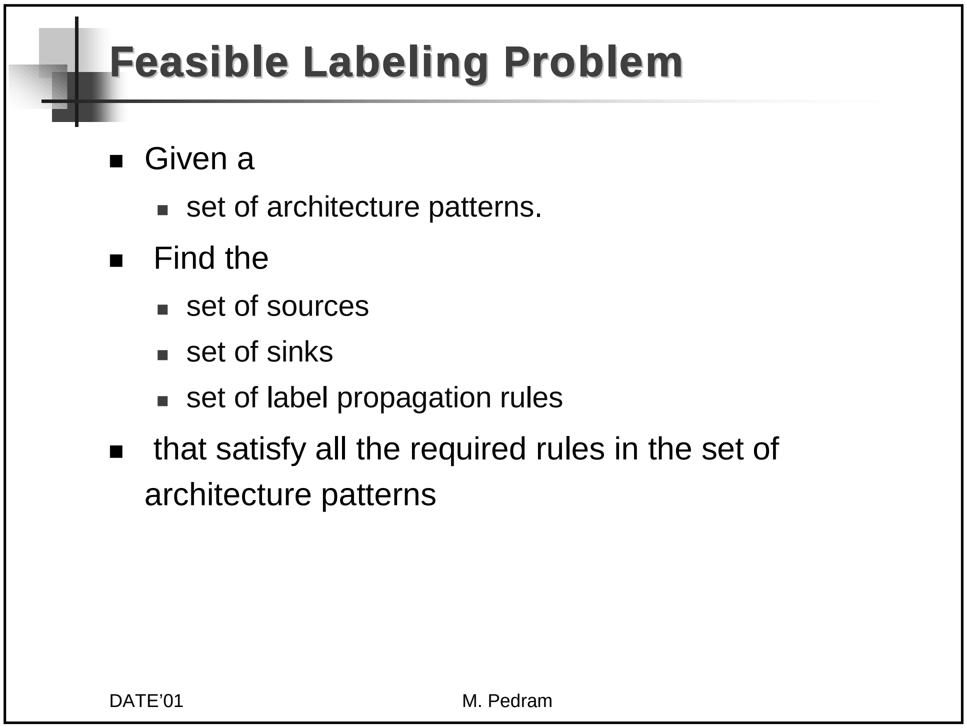
Architecture Pattern

- Name
  - The handle that is used to described the intended architecture effect (e.g., pipeline-flush, pipeline-stall, data-forwarding)
- Description
  - Explanation of how the pattern is caused and how the processor reacts to the pattern
- Liable Set
  - Set of instructions that are responsible for the power dissipation caused by the architecture pattern
- Required Rule
  - Specification of how the propagation rule should work in response to the pattern



An Architecture Pattern Example

- Name
  - Streamlined Execution.
- Description
  - Each pipeline stage performs the operation specified by the incoming instruction
- Liable Set
  - The instruction being executed in a pipeline stage is responsible for the power dissipation of that stage
- Required Rule
  - The instruction label in a pipeline is transferred to the next stage in the next clock cycle



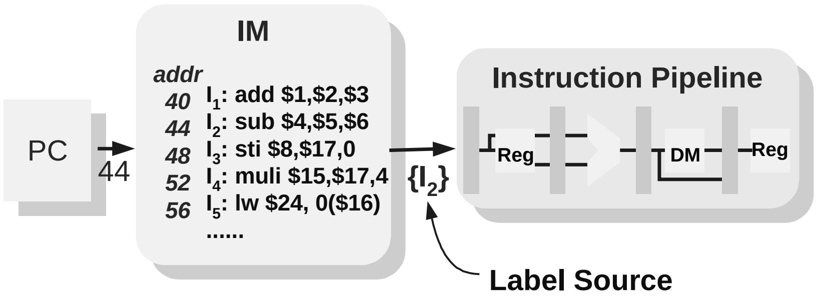
Feasible Labeling Problem

- Given a
  - set of architecture patterns.
- Find the
  - set of sources
  - set of sinks
  - set of label propagation rules
- that satisfy all the required rules in the set of architecture patterns

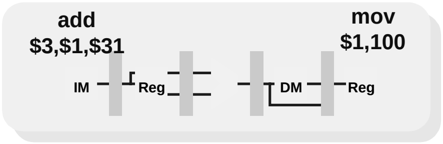
Implication by Domination

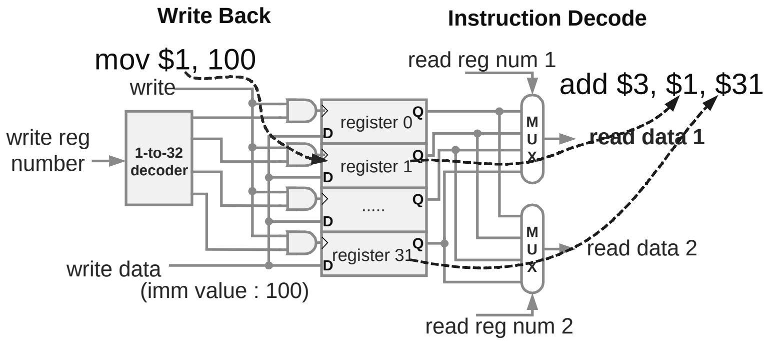
- An architecture pattern is dominated by a combination of other patterns if its required rules are covered by the required rules of these architecture patterns
- A labeling scheme is feasible for a target processor if all of the architecture patterns of that processor are dominated by the architecture patterns that are captured by the labeling scheme

Source



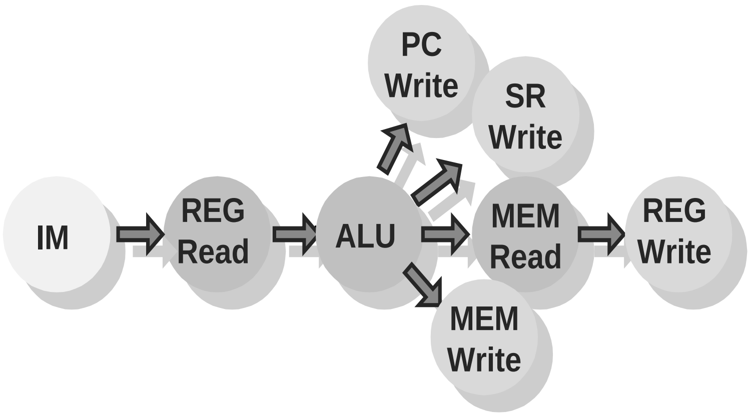
Sink



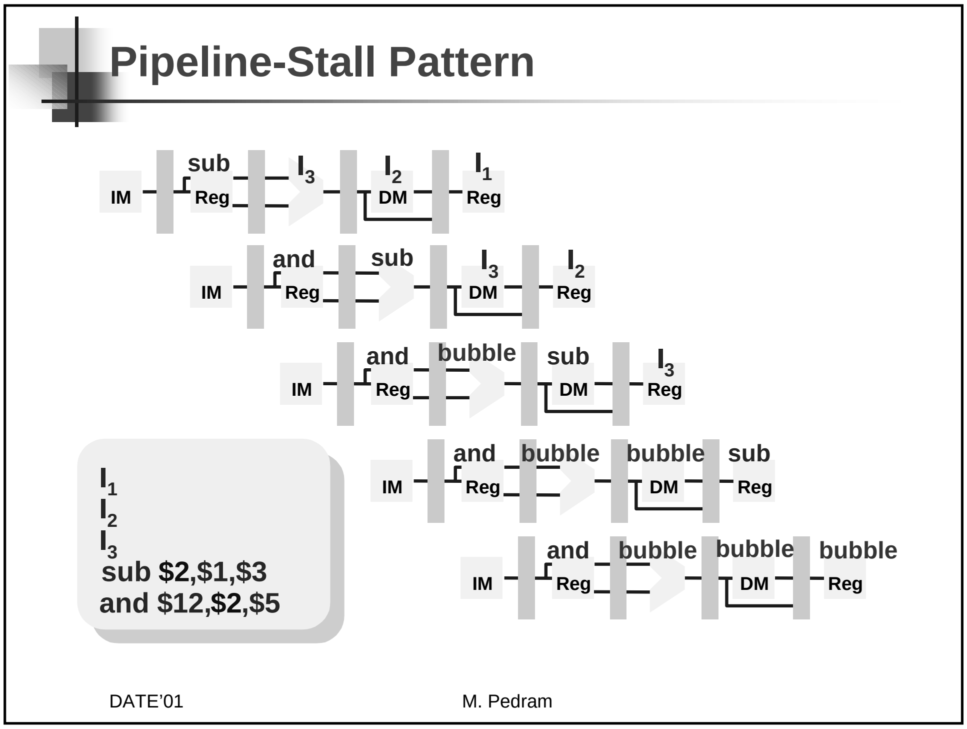


DATE'01 M. Pedram

Journey of a MIPS Instruction

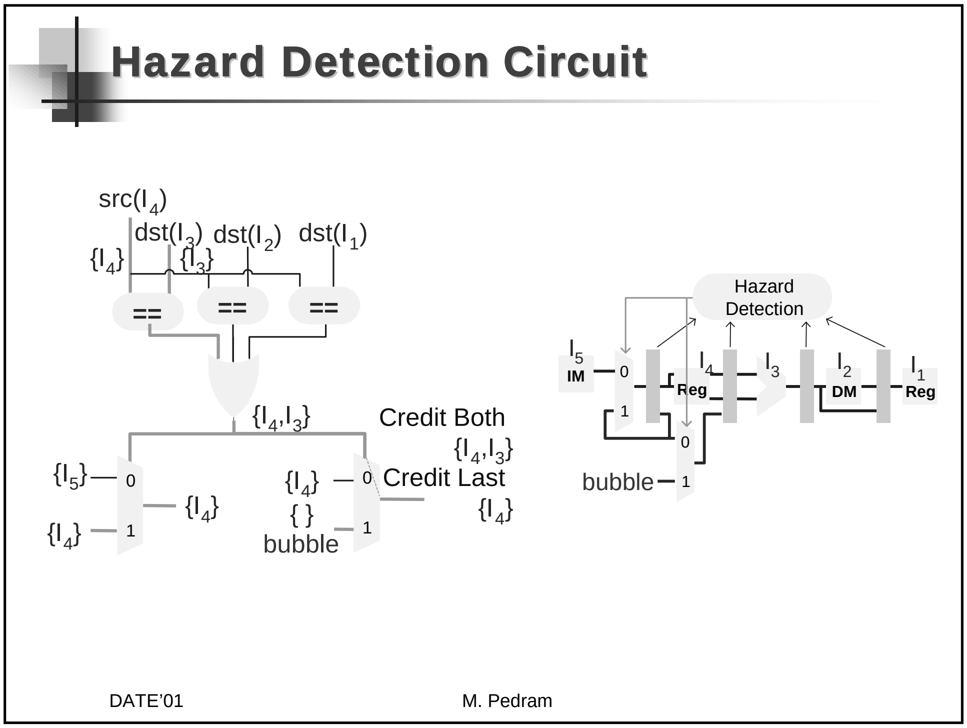


DATE'01 M. Pedram



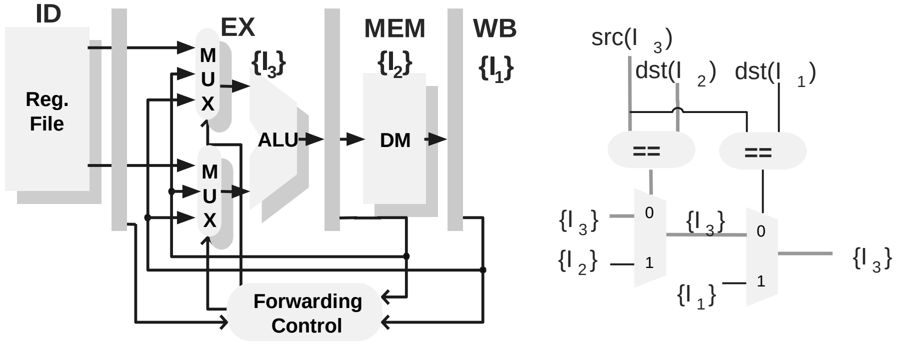
Pipeline-Stall Pattern

I_1
I_2
I_3
sub \$2,\$1,\$3
and \$12,\$2,\$5



Hazard Detection Circuit

Data Forwarding Pattern



Pipeline Flush Pattern

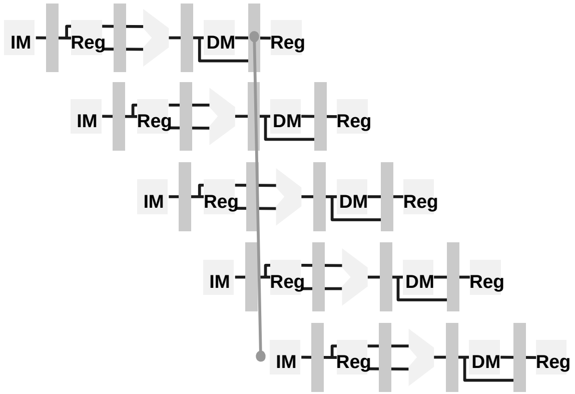
40 beq $1,$3,28

44 and $12,$2,$5

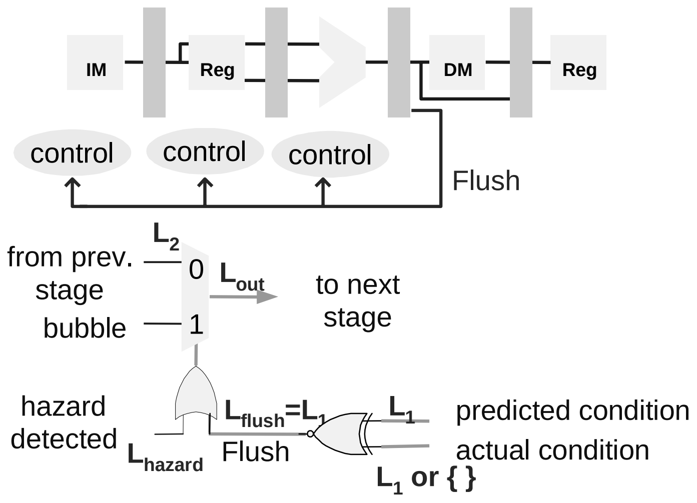
48 or $13,$6,$2

52 add $14,$2,$2

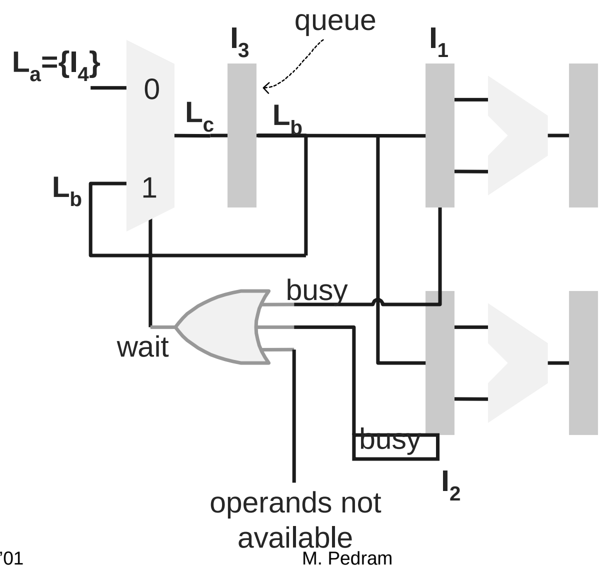
72 lw $4, 50($7)

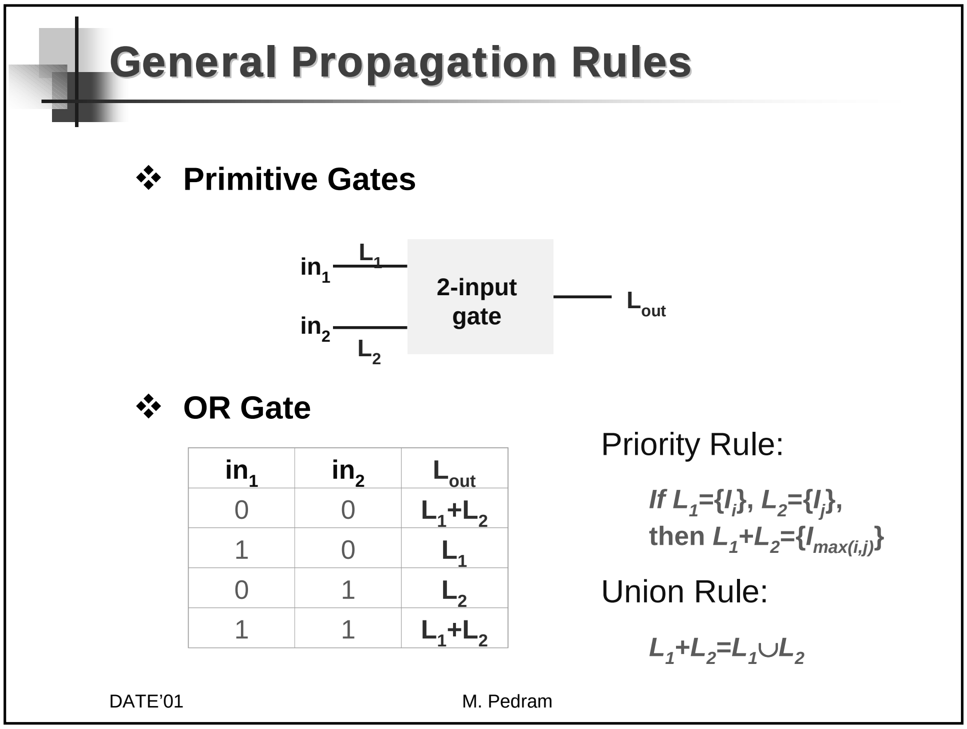


Pipeline Flush Control Circuit



Queuing Pattern





General Propagation Rules

- ❖ **Primitive Gates**

in_1
L_1
2-input gate
in_2
L_2
L_{out}

- ❖ **OR Gate**

in_1	in_2	L_{out}
0	0	L_1+L_2
1	0	L_1
0	1	L_2
1	1	L_1+L_2

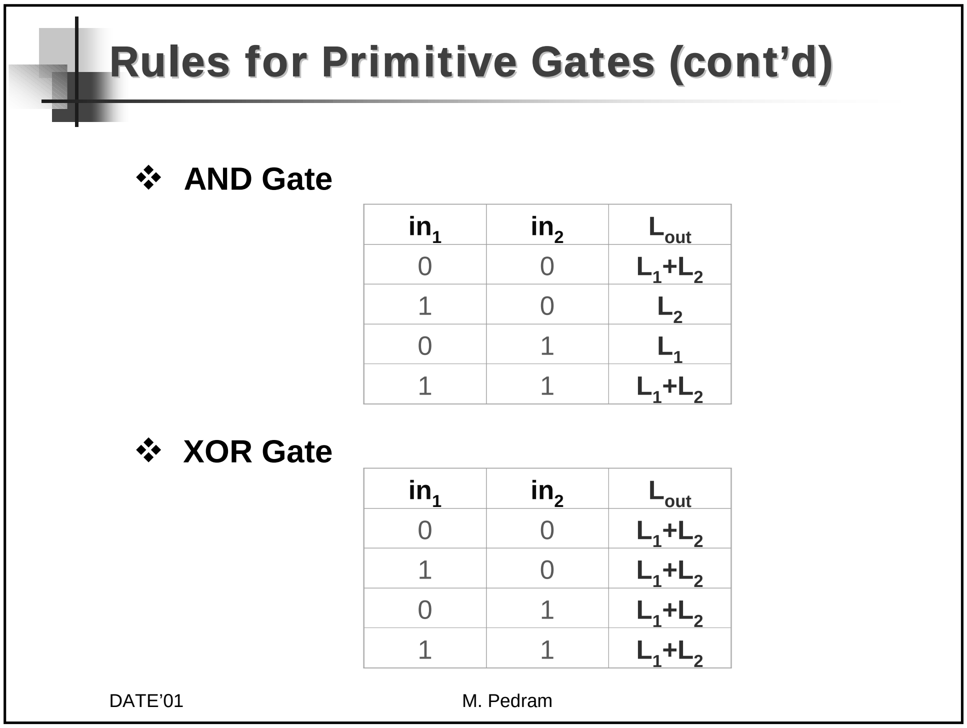
Priority Rule:

If $L_1=\{l_i\}$, $L_2=\{l_j\}$, then $L_1+L_2=\{l_{max(i,j)}\}$

Union Rule:

$L_1+L_2=L_1 \cup L_2$

DATE'01 M. Pedram



Rules for Primitive Gates (cont'd)

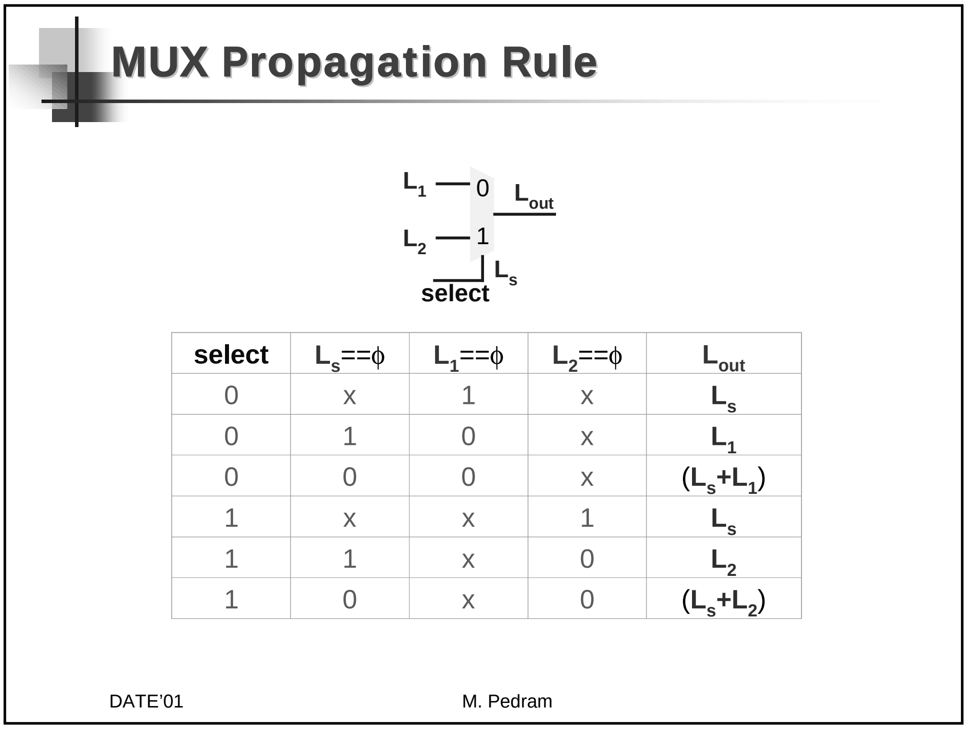
- ❖ **AND Gate**

in_1	in_2	L_{out}
0	0	L_1+L_2
1	0	L_2
0	1	L_1
1	1	L_1+L_2

- ❖ **XOR Gate**

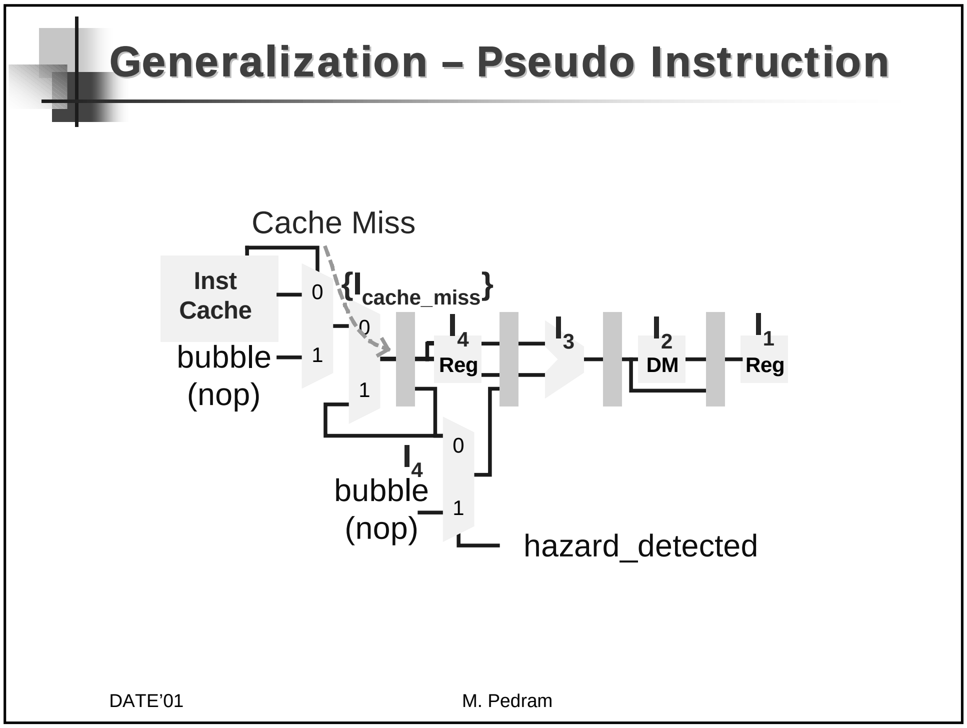
in_1	in_2	L_{out}
0	0	L_1+L_2
1	0	L_1+L_2
0	1	L_1+L_2
1	1	L_1+L_2

DATE'01 M. Pedram



MUX Propagation Rule

select	$L_s==\phi$	$L_1==\phi$	$L_2==\phi$	L_{out}
0	x	1	x	L_s
0	1	0	x	L_1
0	0	0	x	(L_s+L_1)
1	x	x	1	L_s
1	1	x	0	L_2
1	0	x	0	(L_s+L_2)



Generalization – Pseudo Instruction

Generalization - Instruction Splitting

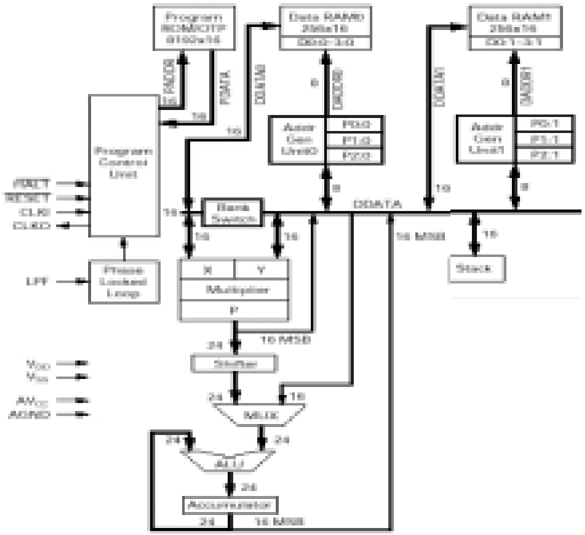
An ARM arithmetic instruction

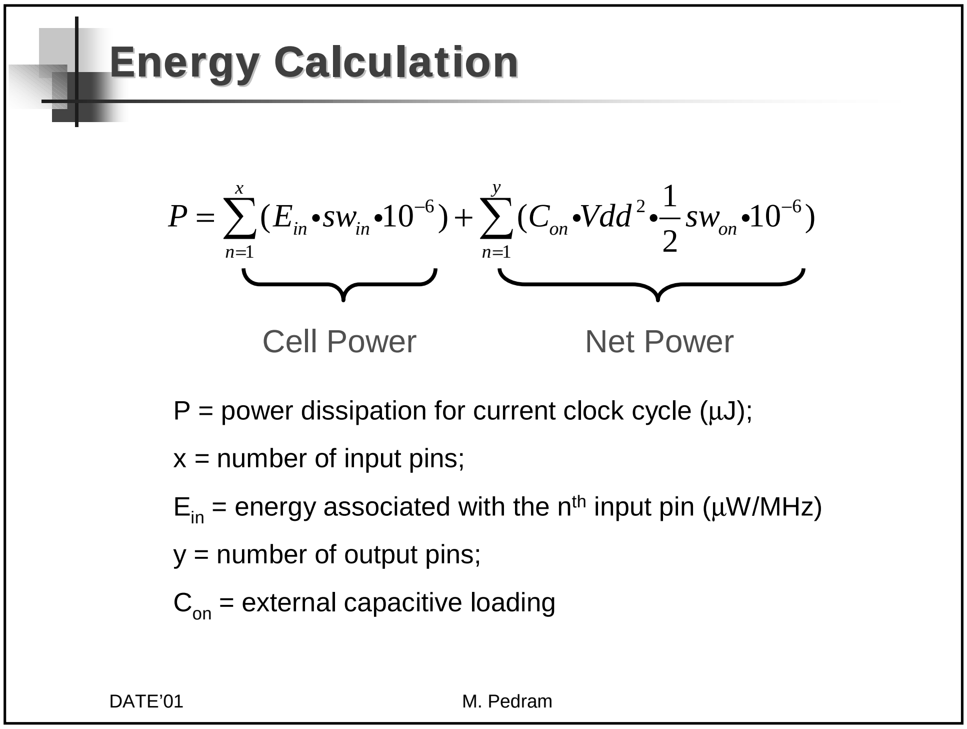
31 30 29 28	27 26 25	24 23 22 21	20	19 18 17 16	15 14 13 12	11 10 9 8 7 6 5 4 3 2 1 0
cond	**0 0 1**	**op**	**S**	**Rd**	**Rn**	**shifter_operand**

Operation

```
if ConditionPassed (<cond>) then
        Rd = Rn <op> <shifter_operand>
        if S == 1 and Rd == R15 then
                CPSR = SPSR
        else if S == 1 then
                N Flag = Rd[31]
                Z Flag = if Rd == 0 then 1 else 0
                C Flag = CarryFrom (Rn + <shifter_operand>)
                V Flag = OverflowFrom (Rn + <shifter_operand>)
```

A Design Example: ZILOG DSP CORE





Energy Calculation

$$P = \underbrace{\sum_{n=1}^{x} (E_{in} \bullet sw_{in} \bullet 10^{-6})}_{\text{Cell Power}} + \underbrace{\sum_{n=1}^{y} (C_{on} \bullet Vdd^{2} \bullet \frac{1}{2} sw_{on} \bullet 10^{-6})}_{\text{Net Power}}$$

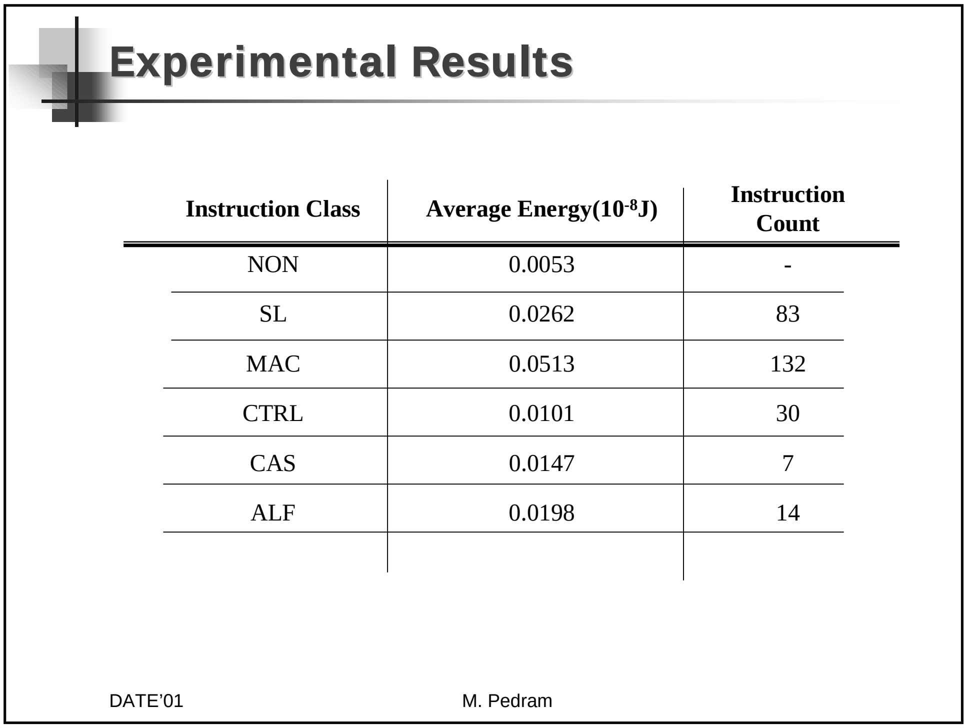
P = power dissipation for current clock cycle (μJ);

x = number of input pins;

E_{in} = energy associated with the nth input pin (μW/MHz)

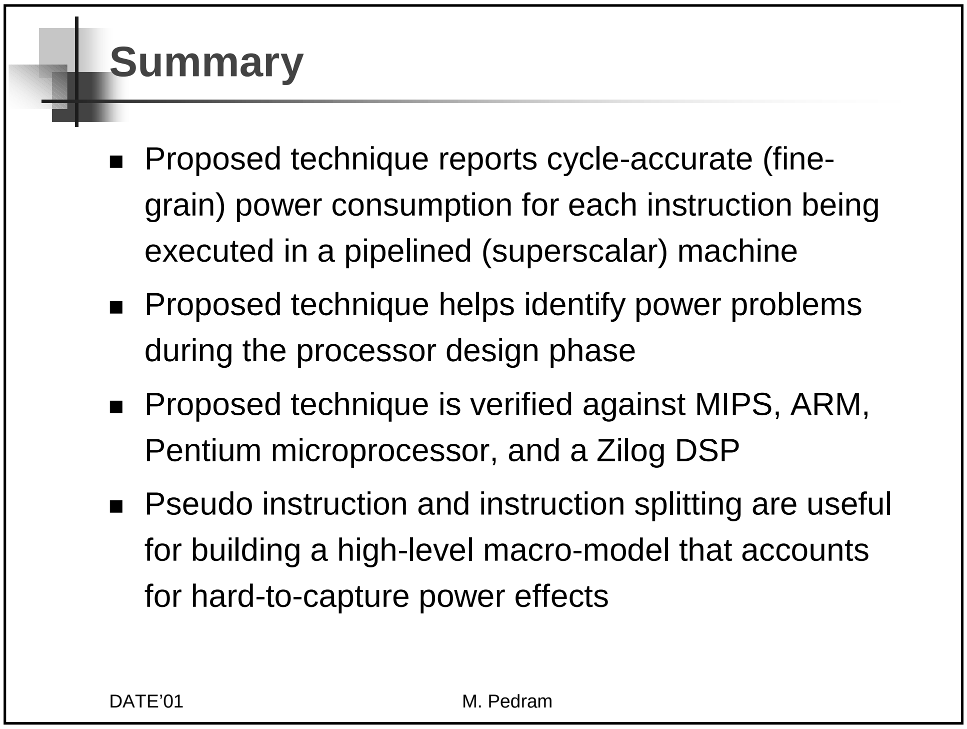
y = number of output pins;

C_{on} = external capacitive loading



Experimental Results

Instruction Class	Average Energy(10^{-8}J)	Instruction Count
NON	0.0053	-
SL	0.0262	83
MAC	0.0513	132
CTRL	0.0101	30
CAS	0.0147	7
ALF	0.0198	14



Summary

- Proposed technique reports cycle-accurate (fine-grain) power consumption for each instruction being executed in a pipelined (superscalar) machine
- Proposed technique helps identify power problems during the processor design phase
- Proposed technique is verified against MIPS, ARM, Pentium microprocessor, and a Zilog DSP
- Pseudo instruction and instruction splitting are useful for building a high-level macro-model that accounts for hard-to-capture power effects