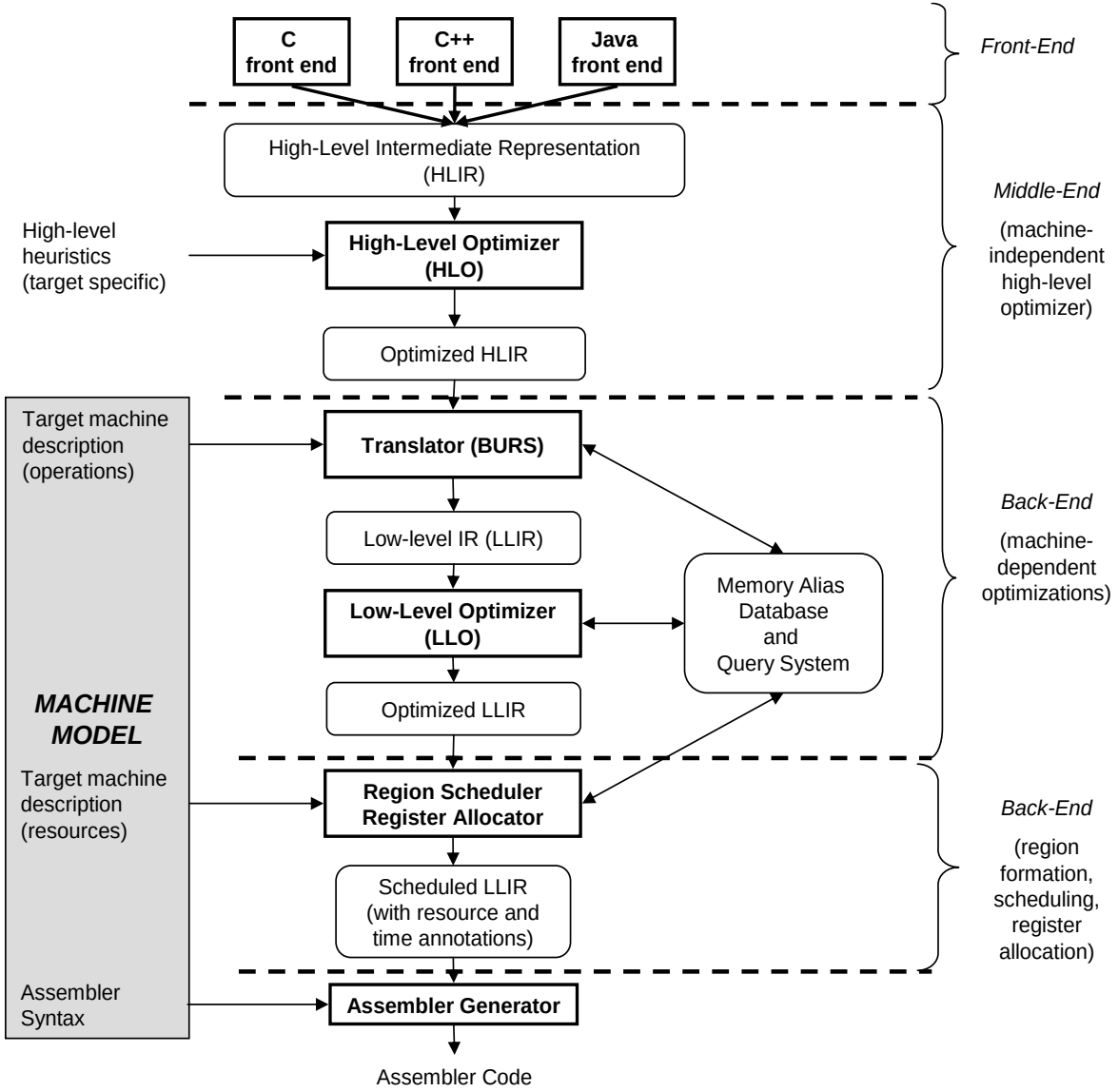
Instruction Level Parallelism

Compilers for High Performance Architectures

Instruction Level Parallelism

- Introduction
- Basic Optimizations
- Control Flow
- Loop Optimizations
- Instruction Scheduling
- Loop Scheduling
- Register Allocation

Structure of a ILP compiler



VLIW/EPIC Architectures

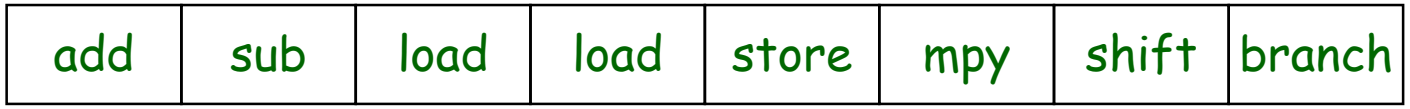
- Best way to start a compiler class is to talk about hardware!!
- Our target processor for this class is a VLIW/EPIC
  - VLIW = Very Long Instruction Word
  - EPIC = Explicitly Parallel Instruction Computing

- Examples:
  - IA-64: aka Itanium I and II, Itanic, Merced, McKinley
  - Embedded processors –
    - All high-performance embedded CPUs are VLIW
    - TI-C6x, Philips Trimedia, Starcore, ST200, Qualcomm Hexagon

VLIW/EPIC Philosophy

- Compiler creates complete plan of run-time execution
  - At what time and using what resource
  - POE communicated to hardware via the instruction set
  - Processor obediently follows POE
  - No dynamic scheduling, out of order execution (these second guess the compilers plan)
- Compiler allowed to play the statistics
  - Many types of info only available at run-time (branch directions, locations accessed via pointers)
  - Traditionally compilers behave conservatively → handle worst case possibility
  - Allow the compiler to gamble when it believes the odds are in its favor
    - Profiling
- Expose microarchitecture to the compiler
  - branch execution, number and type of units, latencies, etc

Defining Feature I - MultiOp



- Superscalar
  - Operations are sequential
  - Hardware figures out resource assignment, time of execution
- MultiOp instruction
  - Set of independent operations that are to be issued simultaneously (no sequential notion within a MultiOp)
  - 1 instruction issued every cycle – provides notion of time
  - Resource assignment indicated by position in MultiOp
  - POE communicated to hardware via MultiOps

Defining Feature II - Exposed Latency

- Superscalar
  - Sequence of atomic operations
  - Sequential order defines semantics
    - Unit assumed latency (UAL)
  - Each conceptually finishes before the next one starts
- VLIW/EPIC – non-atomic operations
  - Register reads/writes for 1 operation separated in time
  - Semantics determined by relative ordering of reads/writes
- Assumed latency (NUAL if > 1 for at least one op)
  - Contract between the compiler and hardware
  - Instruction issuance provides common notion of time

What if I don’t care about VLIWs?

- How do we compile for superscalars?
- How do we compile for RISCs?
- All the basic compiler analyses and transformations are the same for all processor types
  - They were developed for RISCs
- Superscalar compilers work by pretending the processor is a VLIW
  - But must worry about hardware undoing what the compiler did
  - Other resources to worry about (ie reorder buffer, reserv stations, etc.)
  - Not all hardware features available

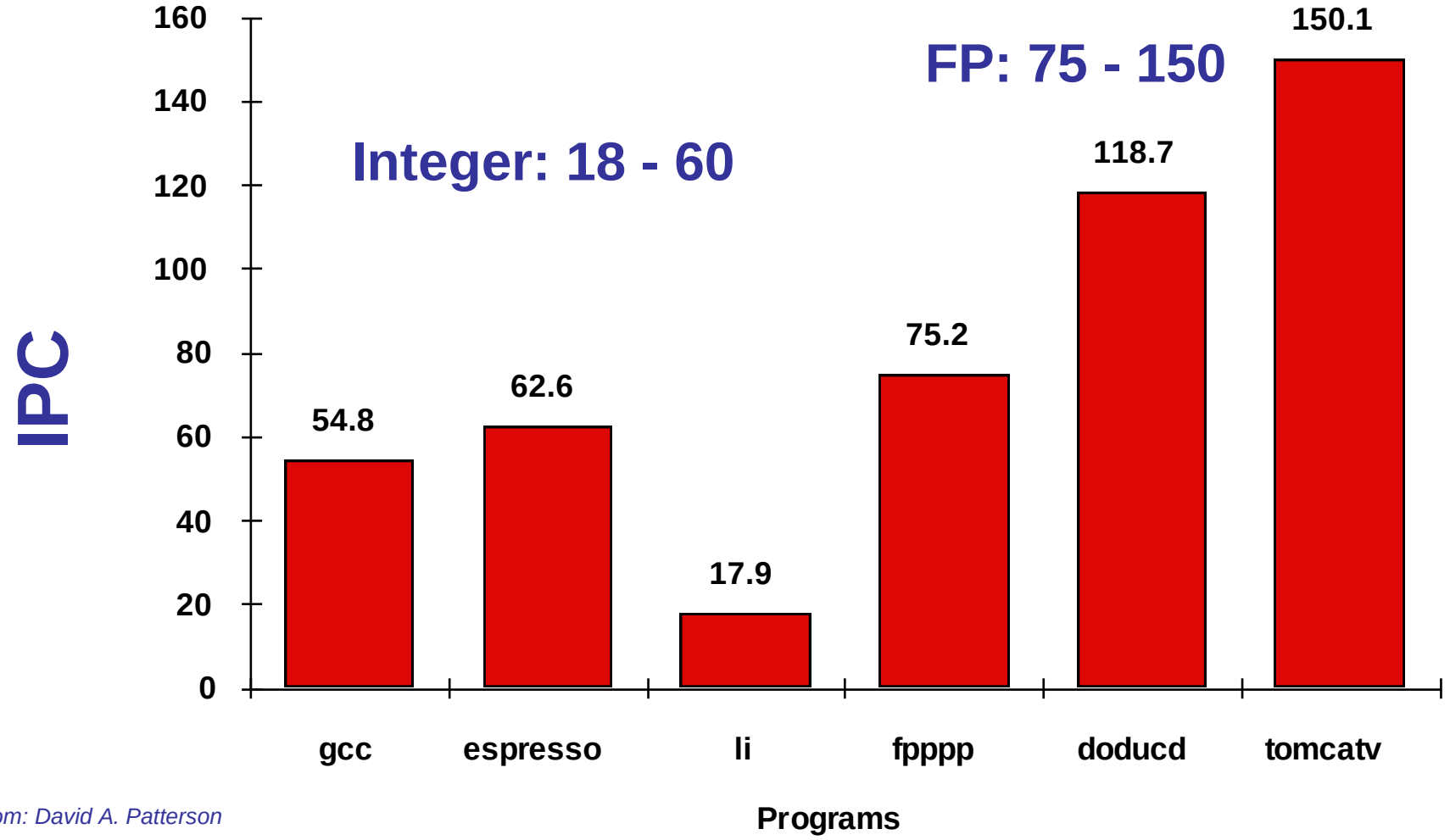
Limits of ILP (1)

- Many Conflicting studies (1980s/90s) of amount of ILP
  - Debated issues
    - Benchmarks (FP Vectorizable Fortran vs. Integer C)
    - Hardware sophistication
    - Compiler sophistication
- How much ILP is available using existing mechanisms with increasing HW budgets?
- What other HW/SW mechanisms do we need to invent to keep on processor performance curve?

Limits of ILP (2)

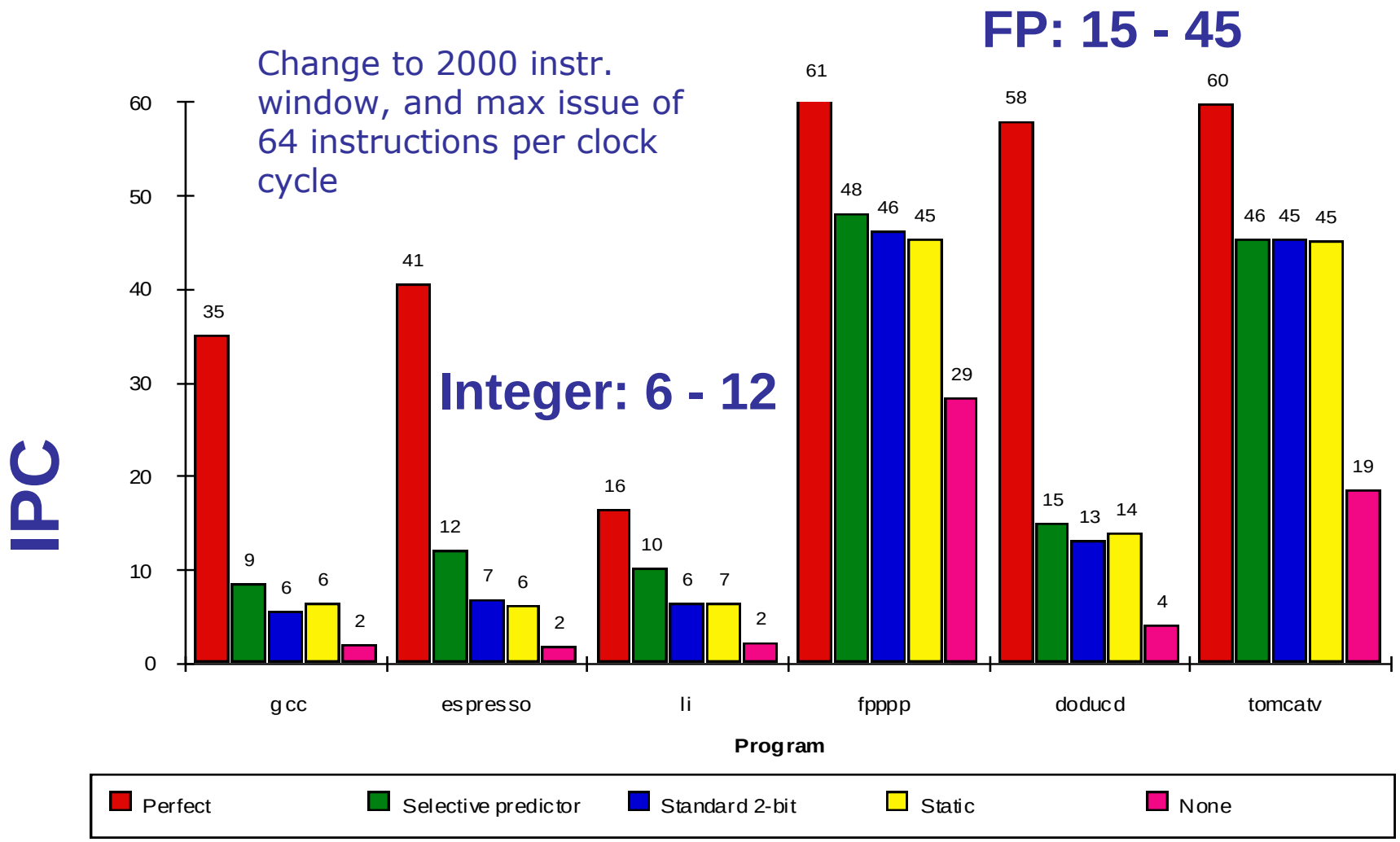
- Initial Model
  - MIPS-like RISC compiler and architecture
- Assumptions for ideal/perfect machine to start:
  - Register renaming–infinite virtual registers and all WAW & WAR hazards are avoided
  - Branch & Jump prediction–perfect; no mispredictions:
    - Machine with perfect speculation & an unbounded buffer of instructions available
  - Memory-address alias analysis–addresses are known & a store can be moved before a load provided addresses not equal
- One-cycle latency for all instructions
- Unlimited number of instructions issued per clock cycle

Ideal Machine



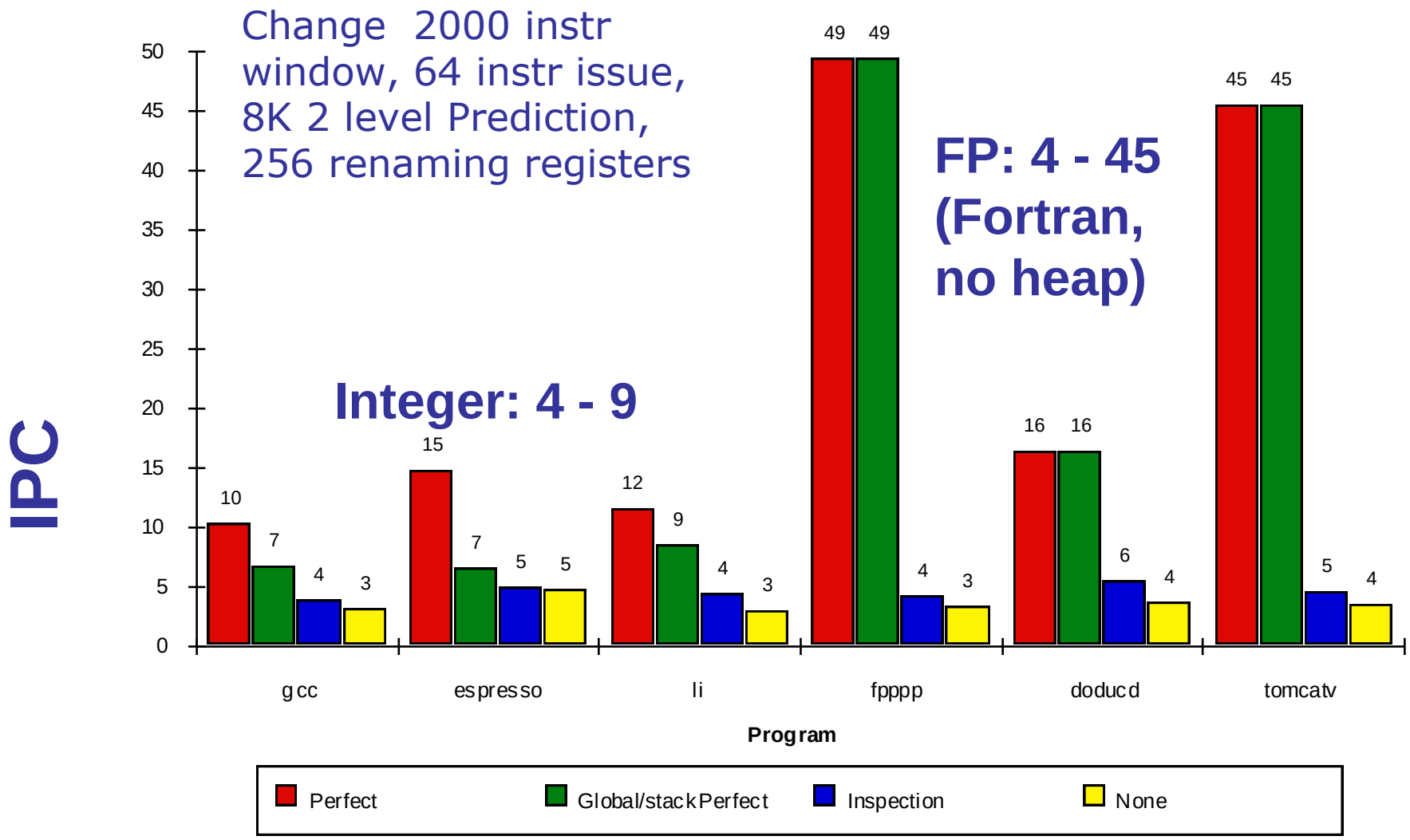
From: David A. Patterson
UCB, CS252 Lecture Notes

Realistic HW: Branch Impact



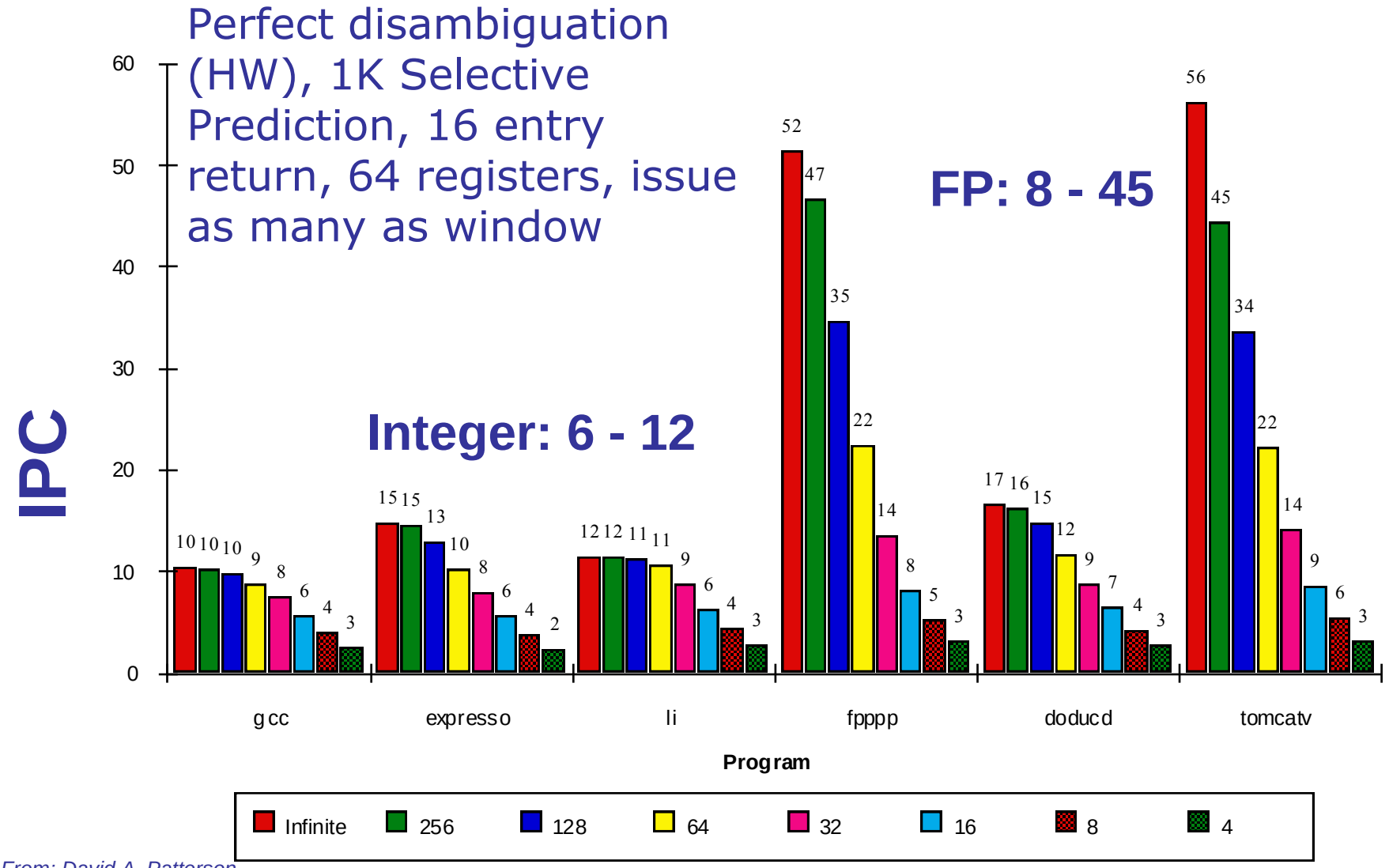
From: David A. Patterson
UCB, CS252 Lecture Notes

Realistic HW: Register Impact



From: David A. Patterson
UCB, CS252 Lecture Notes

Realistic HW: Fetch Window Impact



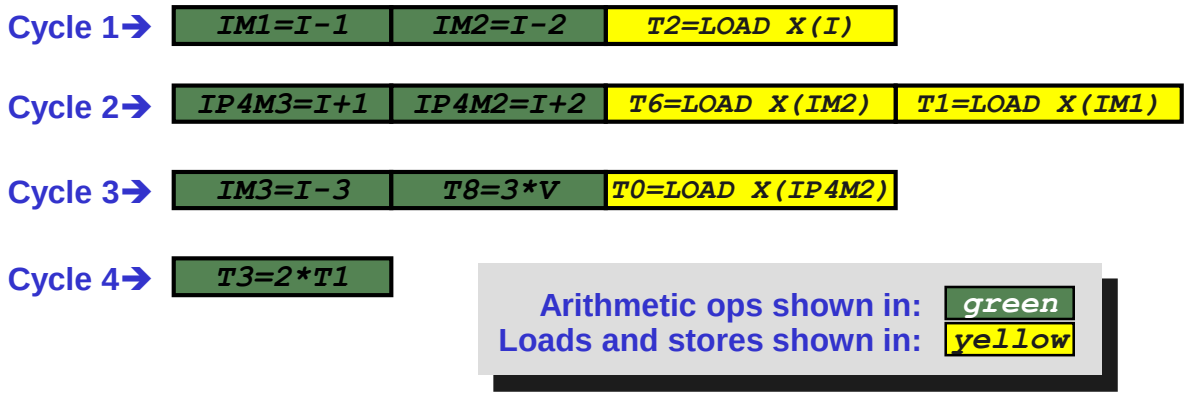
From: David A. Patterson UCB, CS252 Lecture Notes

Instruction-Level Parallelism

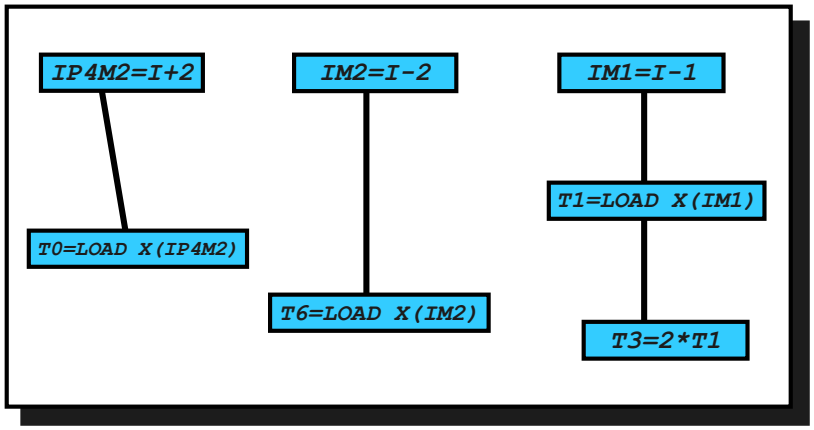
Source Code:

```
IM1  = I-1
IM2  = I-2
IM3  = I-3
T1   = LOAD X(IM1)
T2   = LOAD X(I)
T3   = 2*T1
T6   = LOAD X(IM2)
T8   = 3*V
IP4  = I+4
IP4M2 = I+2
IP4M3 = I+1
T0   = LOAD X(IP4M2)
```

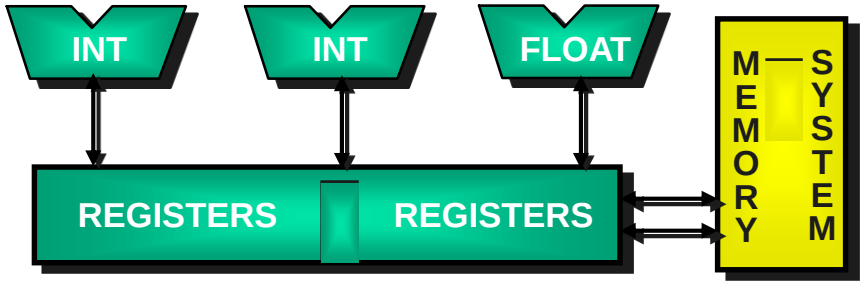
Might Execute Like This:



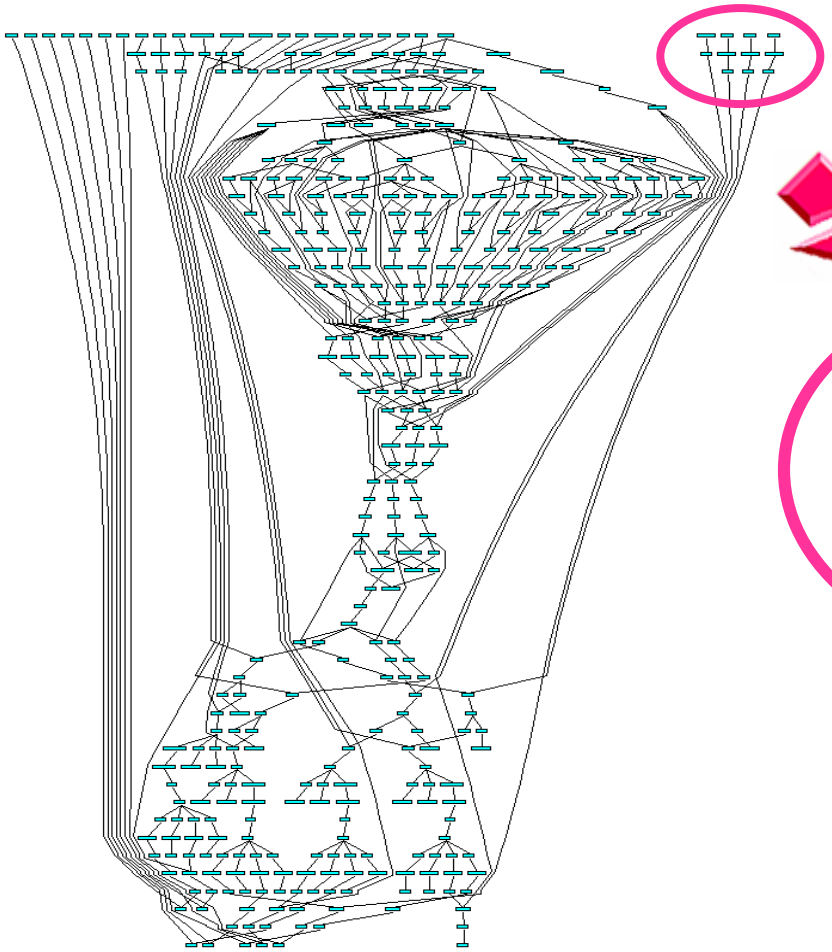
Data Dependences:

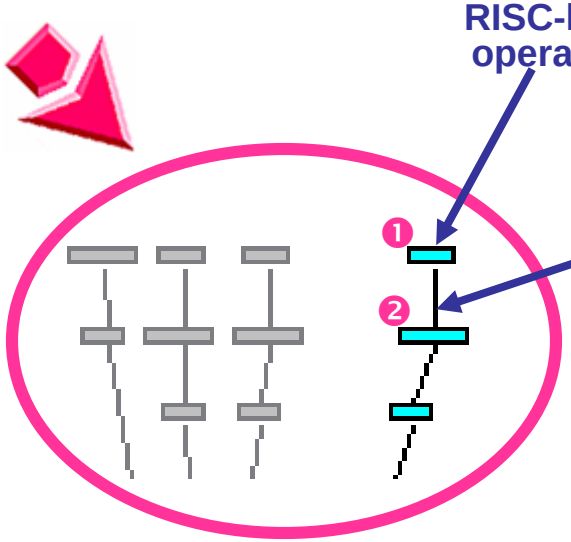


Using Extra ILP Hardware:



Example: imaging pipeline one pixel computation



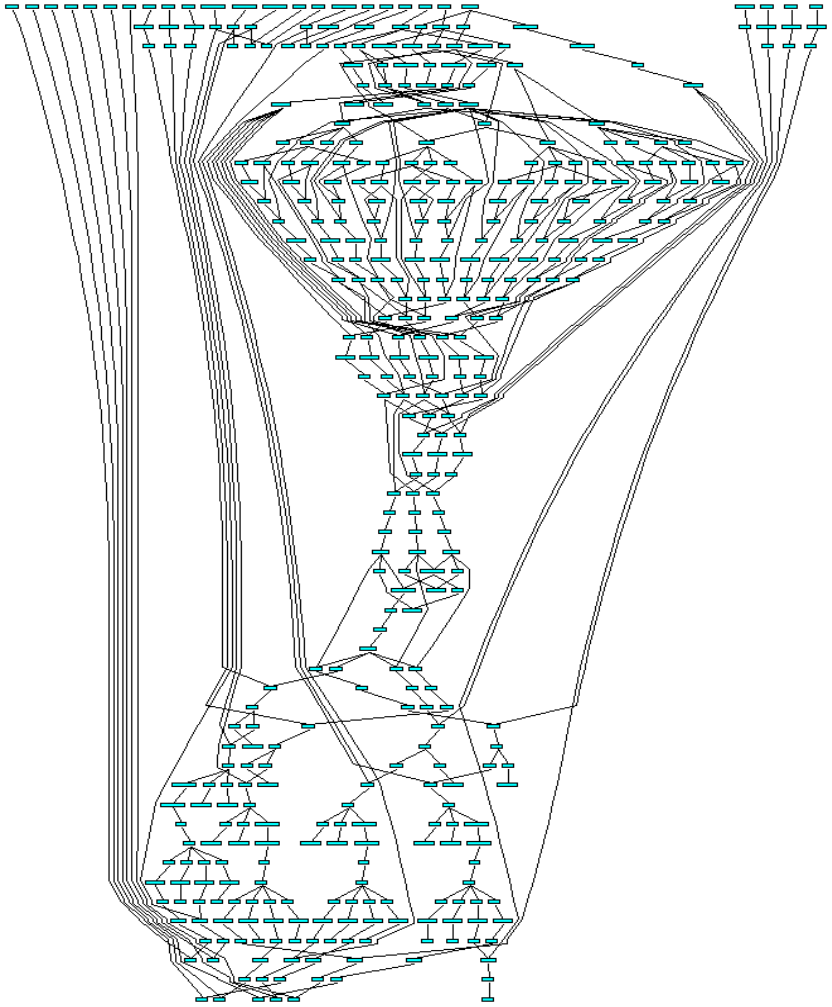


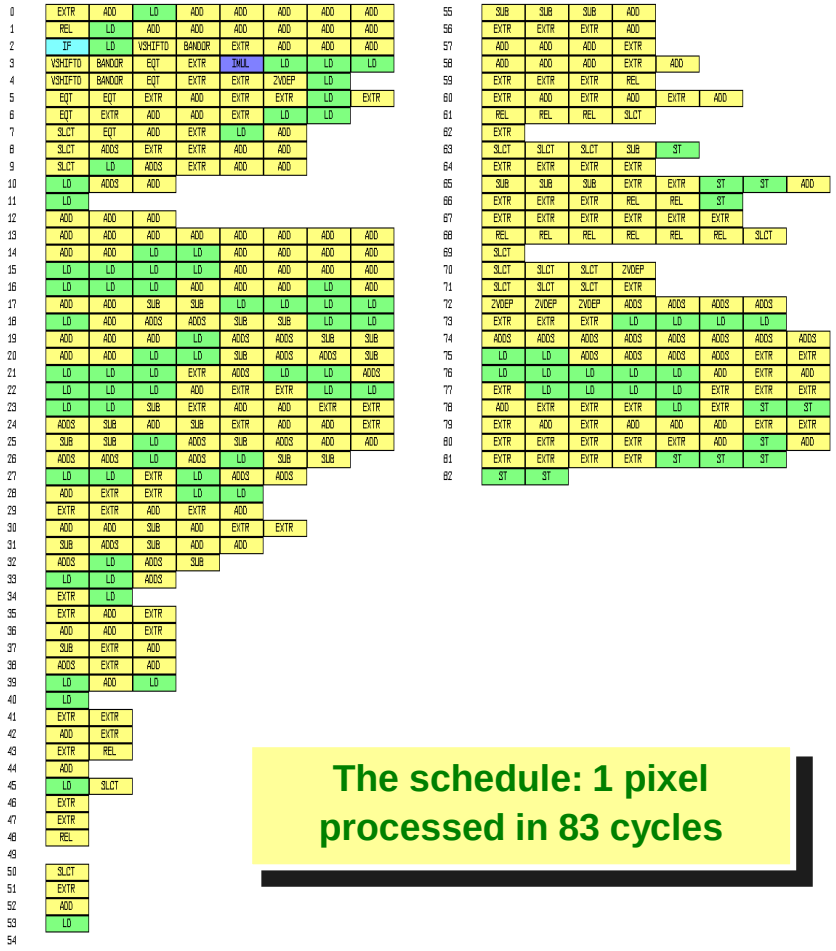
This is a single RISC-level operation

This line means that the operation labeled ❶ must be scheduled before the operation labeled ❷

Example algorithm: typical color printer imaging pipeline (upscaling, colorspace conversion, halftoning)

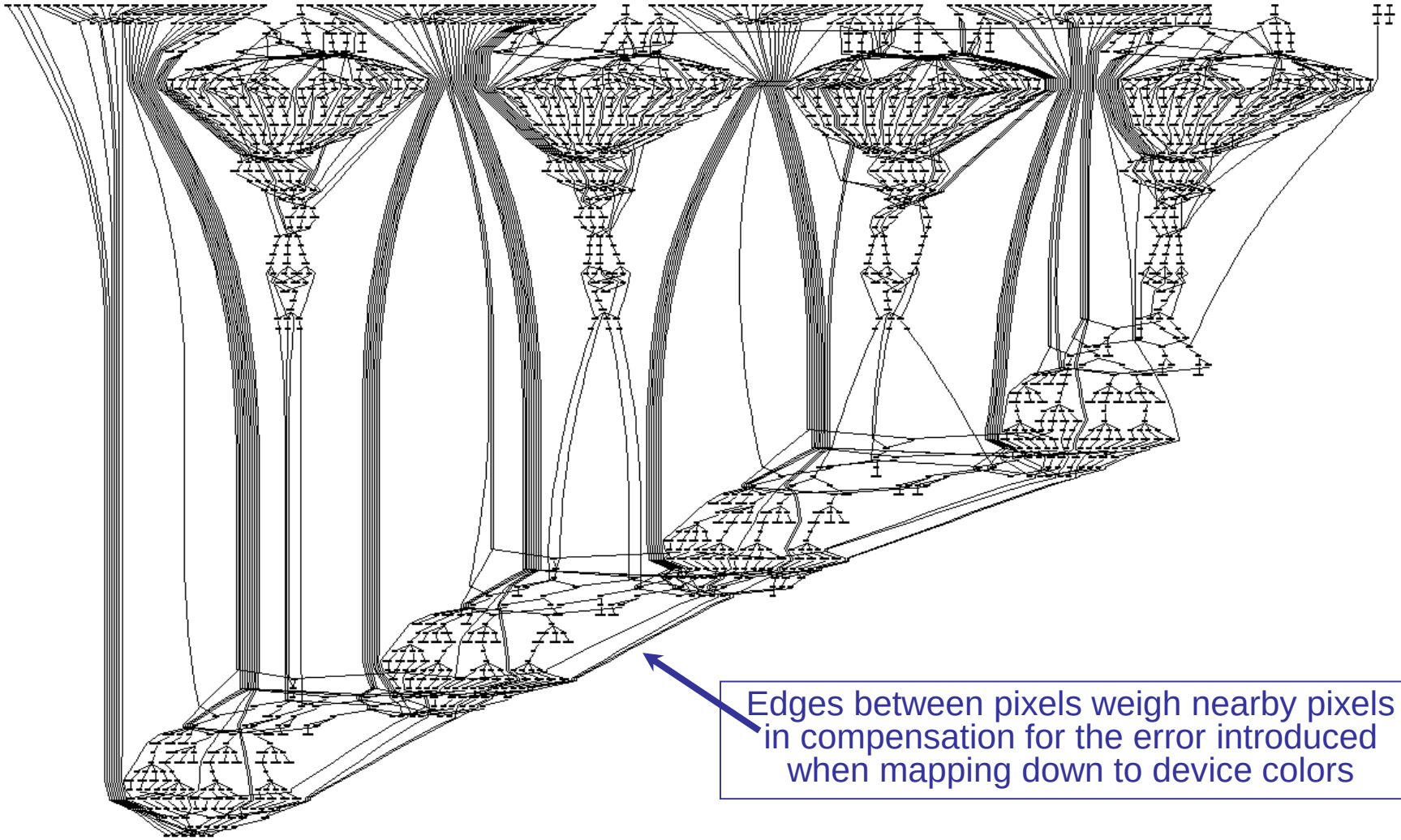
Example: imaging pipeline one pixel dependences and schedule





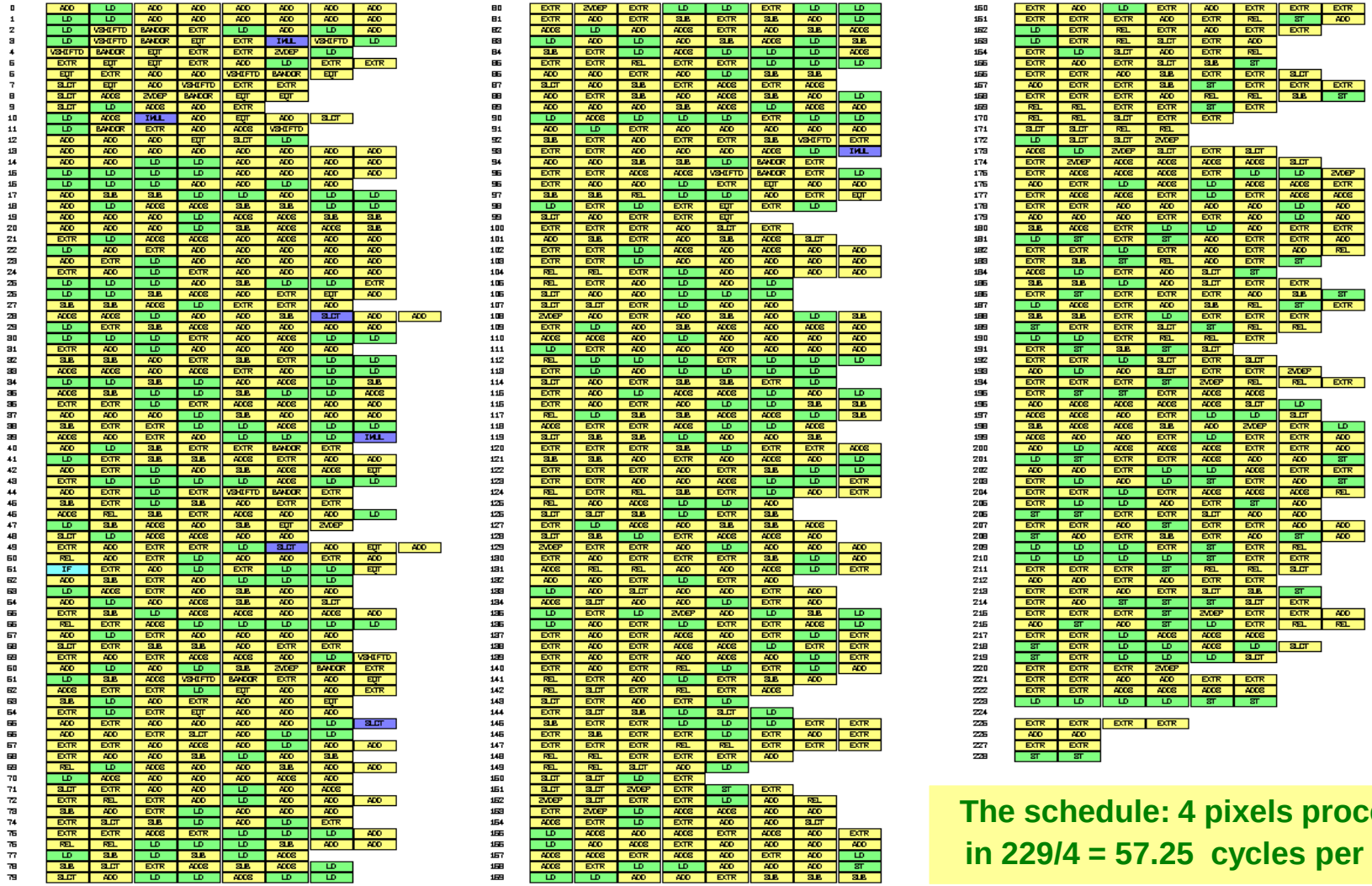
The schedule: 1 pixel processed in 83 cycles

Example: imaging pipeline four pixels unrolled



Edges between pixels weigh nearby pixels in compensation for the error introduced when mapping down to device colors

Example: imaging pipeline four pixels scheduled



The schedule: 4 pixels processed in 229/4 = 57.25 cycles per pixel.