Hardware Based Speculation

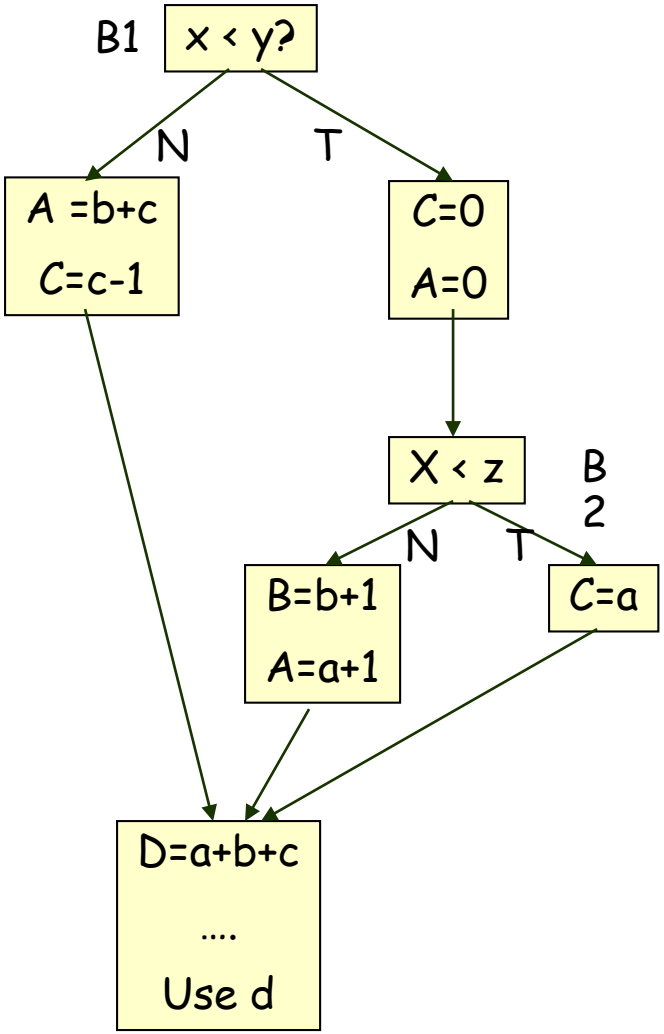
Soner Önder

Michigan Technological University, Houghton MI

Hardware Based Speculation

- **Exploiting more ILP requires that we overcome the limitation of control dependence:**
  - With branch prediction we allowed the processor continue issuing instructions past a branch based on a prediction:
    - Those fetched instructions do not modify the processor state.
    - These instructions are squashed if prediction is incorrect.
  - We now allow the processor to execute these instructions before we know if it is ok to execute them:
    - We need to correctly restore the processor state if such an instruction should not have been executed.
    - We need to pass the results from these instructions to future instructions as if the program is just following that path.

Hardware Based Speculation



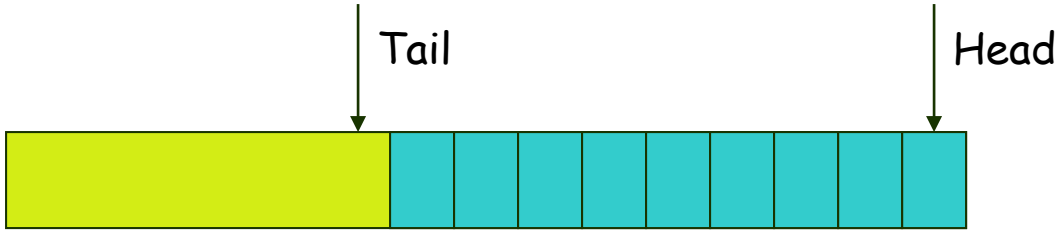
- Assume the processor predicts B1 to be taken and executes.
- What will happen if the prediction was wrong?
- What value of each variable should be used if the processor predicts B1 and B2 taken and executes instructions along the way?

Hardware Based Speculation

- In order to execute instructions speculatively, we need to provide means:
  - To roll back the values of both registers and the memory to their correct values upon a misprediction,
  - To communicate speculatively calculated values to the new uses of those values.
- Both can be provided by using a simple structure called Reorder Buffer (ROB).

Reorder Buffer

- It is a simple circular array with a head and a tail pointer:
  - New instructions is allocated a position at the tail in program order.
  - Each entry provides a location for storing the instruction's result.
  - New instructions look for the values starting from tail – back.
  - When the instruction at the head complete and becomes non-speculative the values are committed and the instruction is removed from the buffer.



Reorder Buffer

- 3 fields: instr, destination, value
  - Reorder buffer can be operand source => more registers like RS
  - Use reorder buffer number instead of reservation station when execution completes
  - Supplies operands between execution complete & commit
  - Once operand commits, result is put into register
  - Instructions commit
  - As a result, its easy to undo speculated instructions on mispredicted branches or on exceptions

Steps of Speculative Tomasulo Algorithm

1. Issue [get instruction from FP Op Queue]

    1. Check if the reorder buffer is full.
    2. Check if a reservation station is available.
    3. Access the register file and the reorder buffer for the current values of the source operands.
    4. Send the instruction, its reorder buffer slot number and the source operands to the reservation station.

Once issued, the instruction stays in the reservation station until it gets both operands.

Steps of Speculative Tomasulo Algorithm

2. Execute [operate on operands (EX)]

When both operands ready and a functional unit is available, the instruction executes.

This step checks RAW hazards and as long as operands are not ready, watches CDB for results.

Steps of Speculative Tomasulo Algorithm

3. Write result [finish execution (WB)]

Write on Common Data Bus to all awaiting FUs and
the reorder buffer; mark reservation station available.

Steps of Speculative Tomasulo Algorithm

4. Commit [update register file with reorder result]

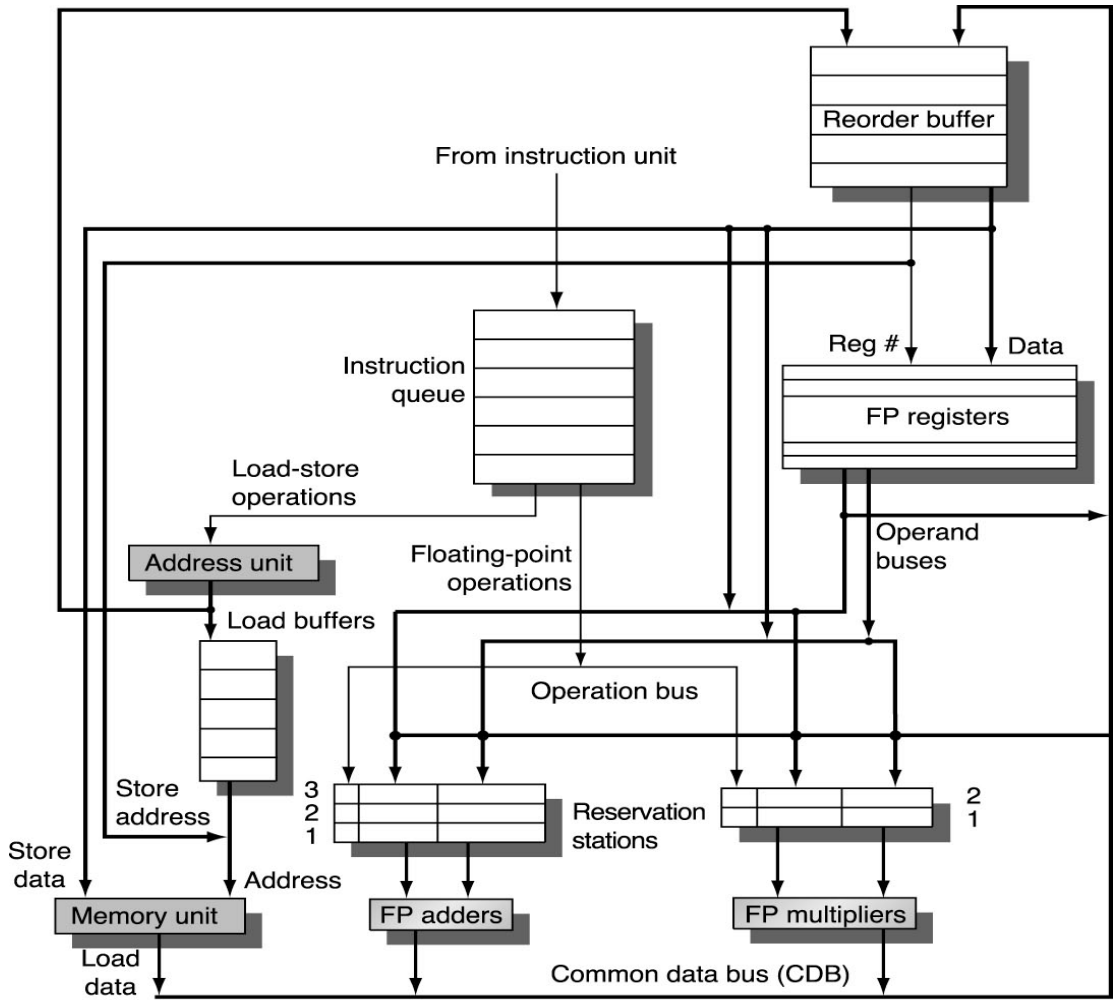
- When instruction reaches the head of reorder buffer
- The result is present
- No exceptions associated with the instruction:

The instruction becomes non-speculative:

- Update register file with result (or store to memory)
- Remove the instruction from the reorder buffer.

A mispredicted branch flushes the reorder buffer.

MIPS FP Unit



Renaming Registers

Common variation of speculative design

Reorder buffer keeps instruction information
but not the result

Extend register file with extra
renaming registers to hold speculative results

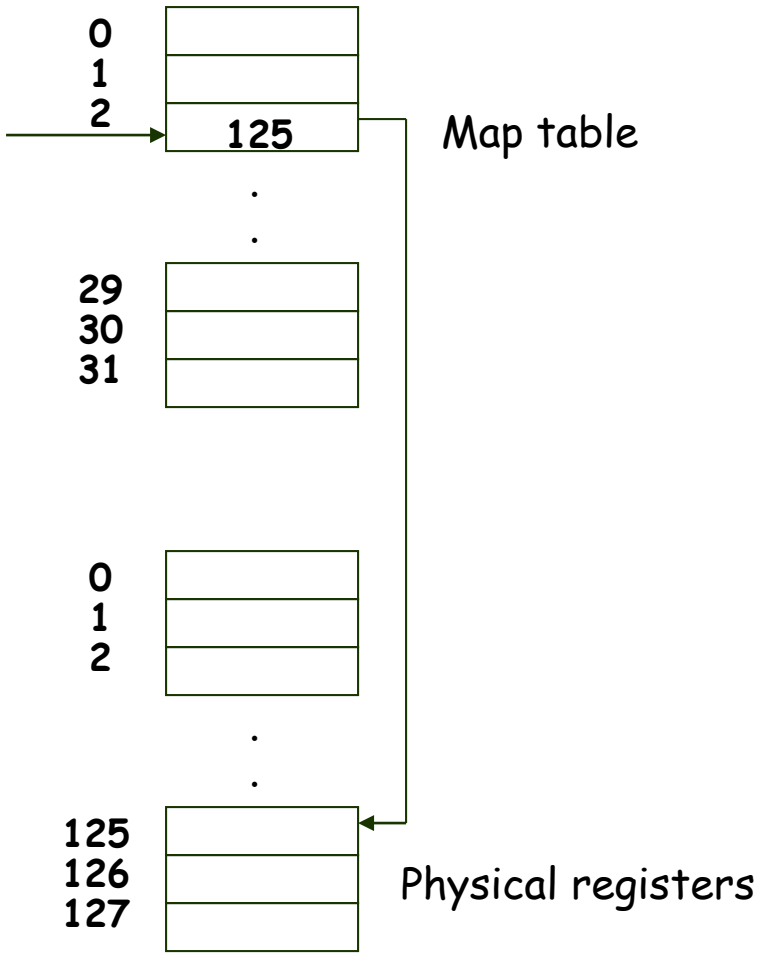
Rename register allocated at issue;
result into rename register on execution complete;
rename register into real register on commit

Operands read either from register file
(real or speculative) or via Common Data Bus

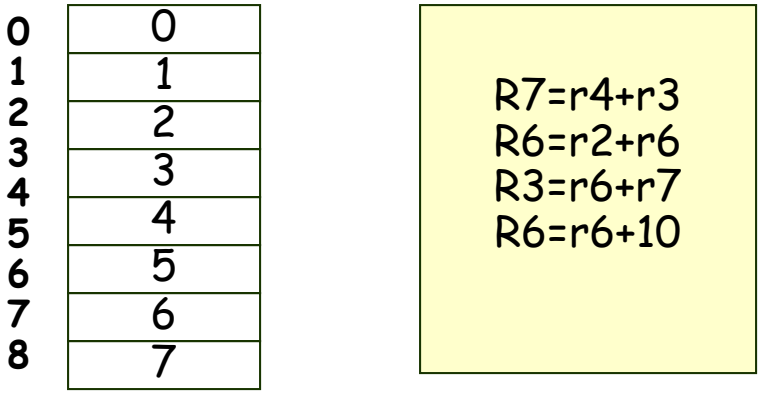
Advantage: operands are always from single source (extended register file)

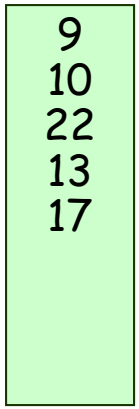
Renaming Registers

1. Index a MAP table using the source register identifiers to get the physical register number.
2. Get the previous physical register number for the destination register.
3. Allocate a free physical register and modify the MAP table by indexing it with the destination register identifier.
4. When instruction commits, return the previous physical register to the pool.

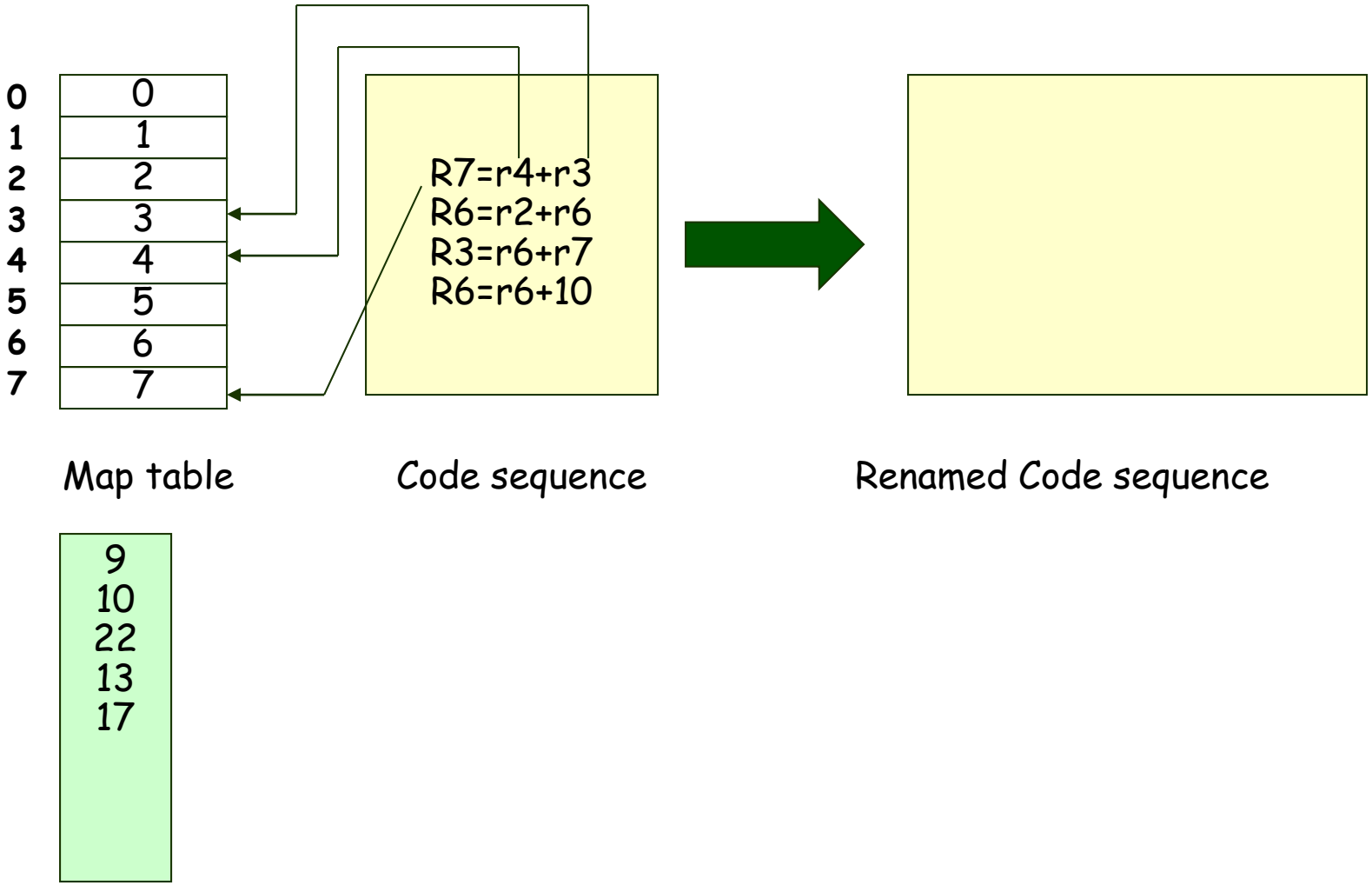


Renaming Registers

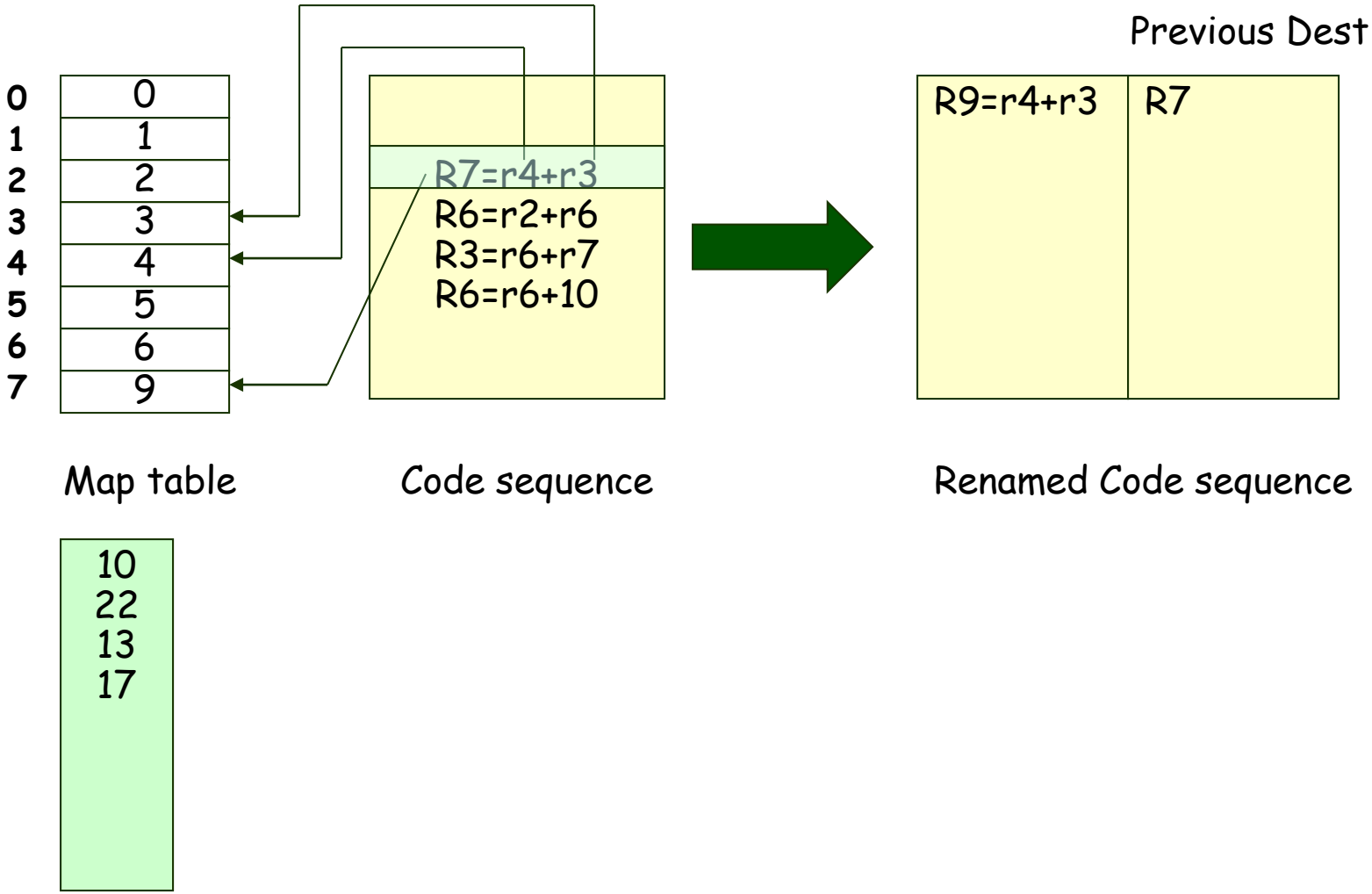




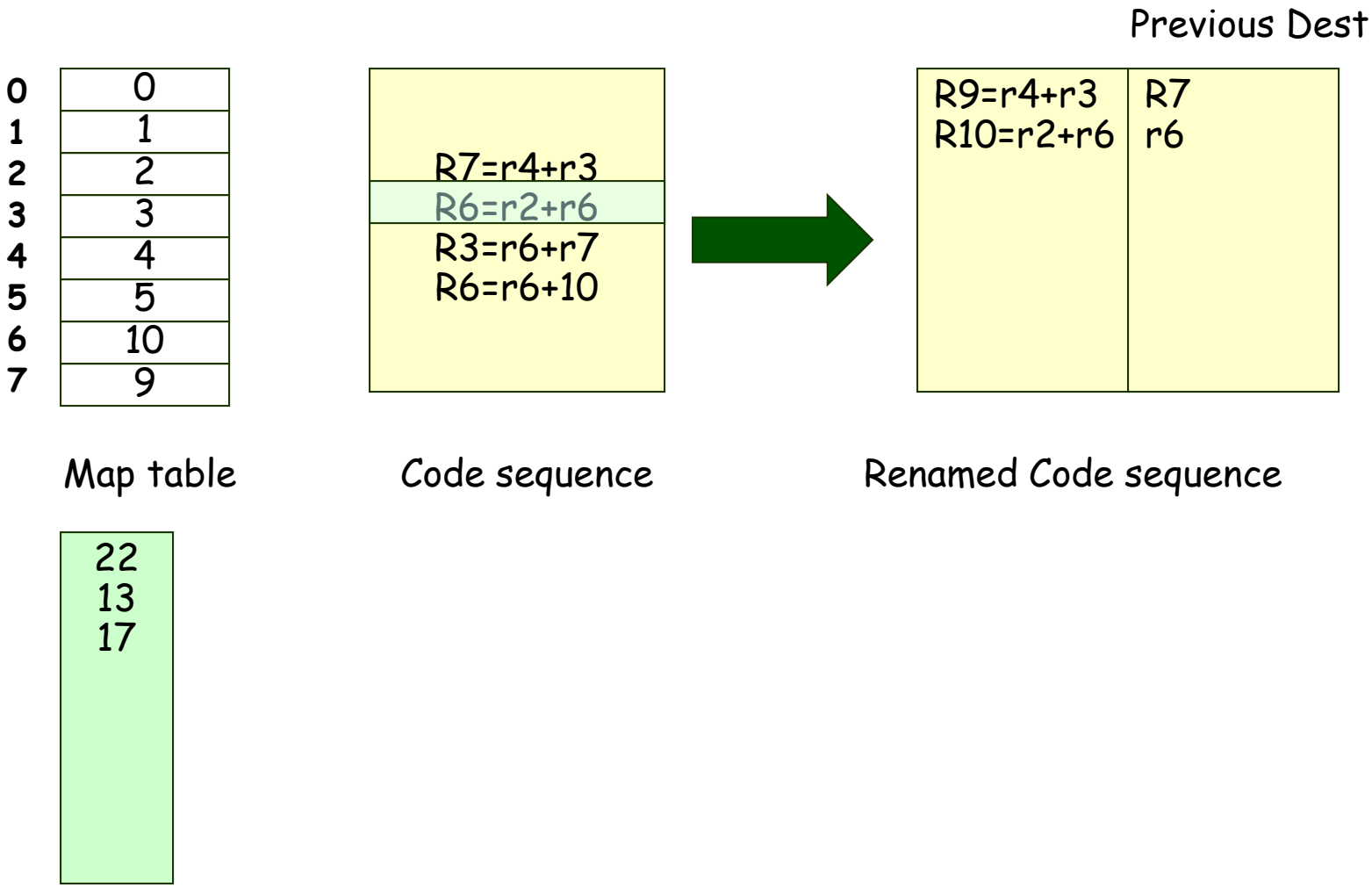
Renaming Registers



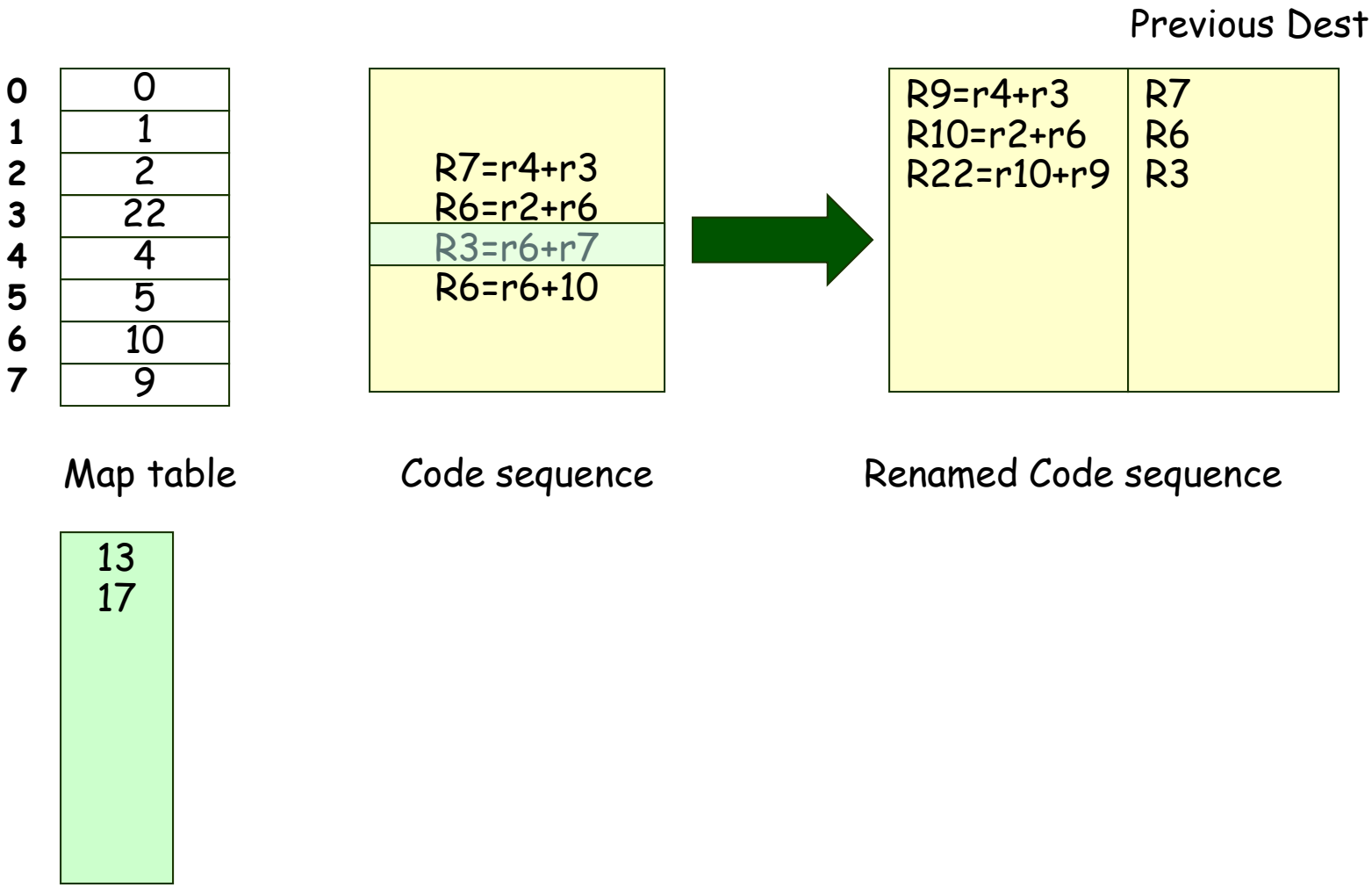
Renaming Registers



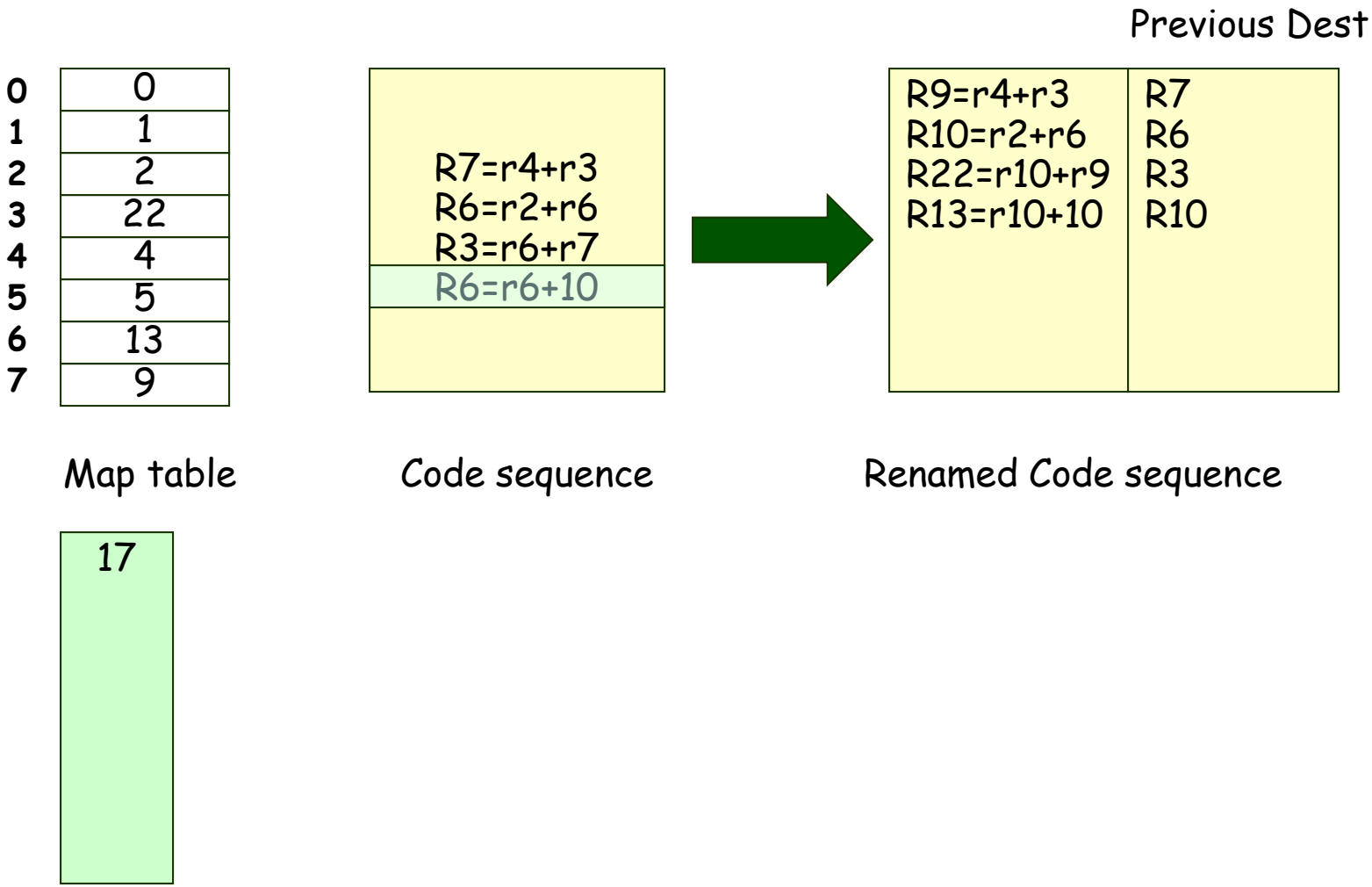
Renaming Registers



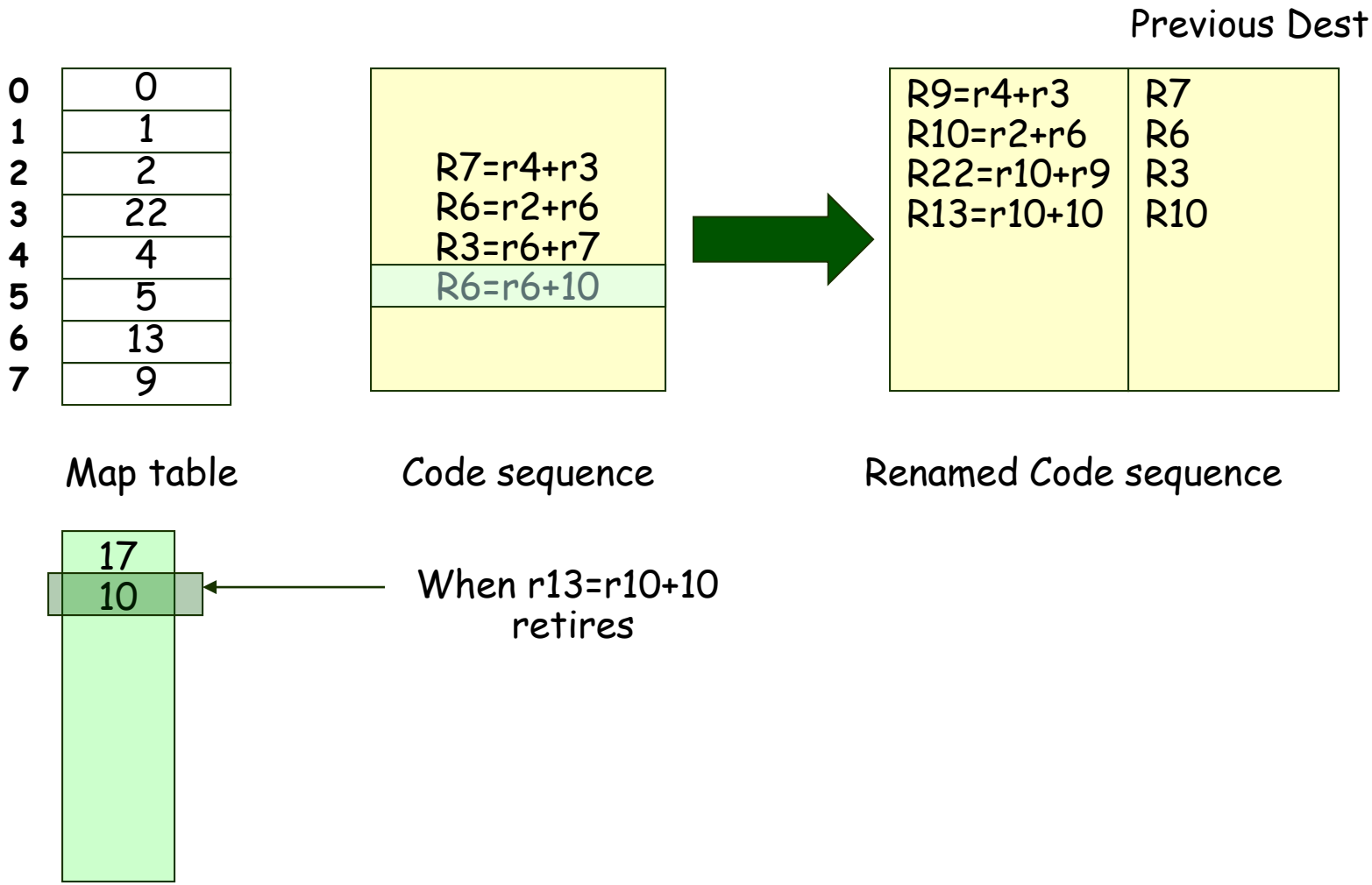
Renaming Registers



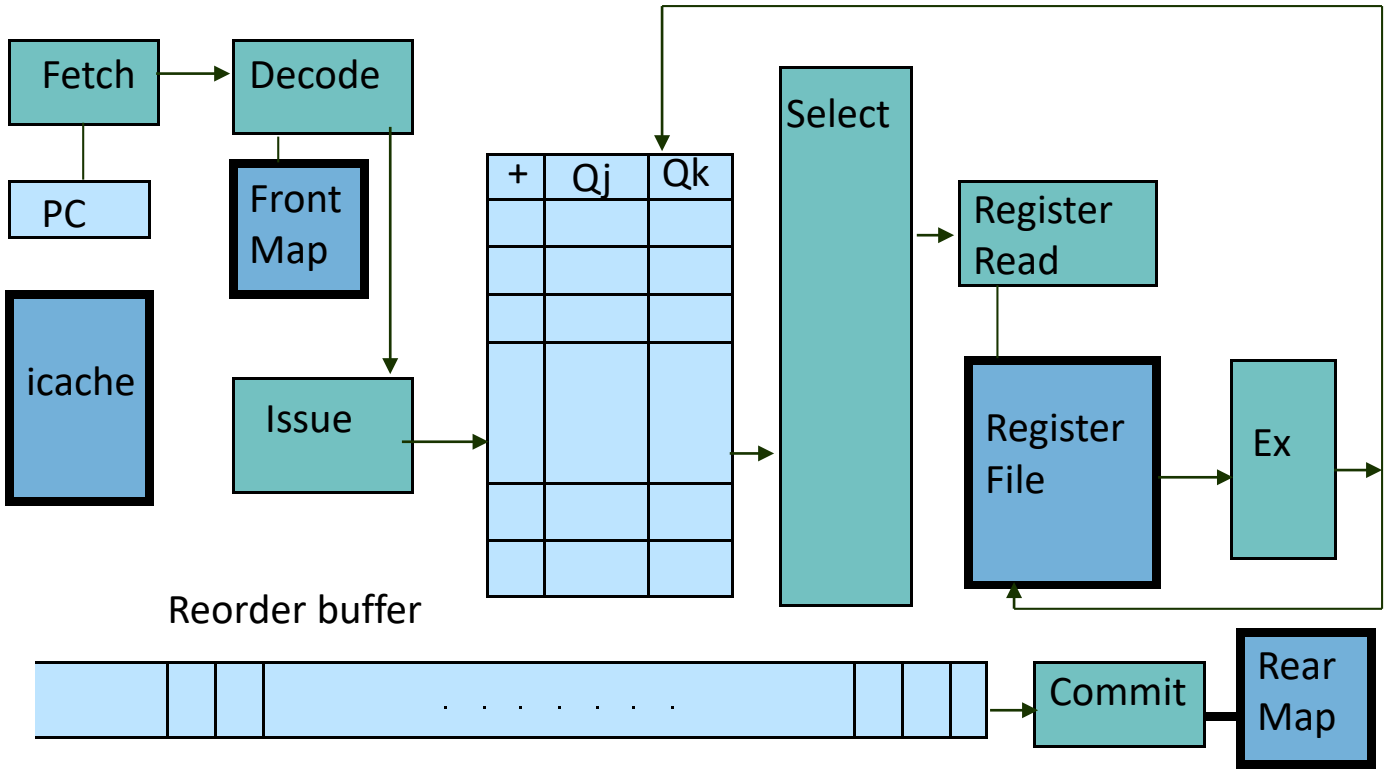
Renaming Registers



Renaming Registers



Speculative Processing with Map tables



Limits to ILP

Assumptions for ideal/perfect machine to start:

1. *Register renaming*–infinite virtual registers and all WAW & WAR hazards are avoided

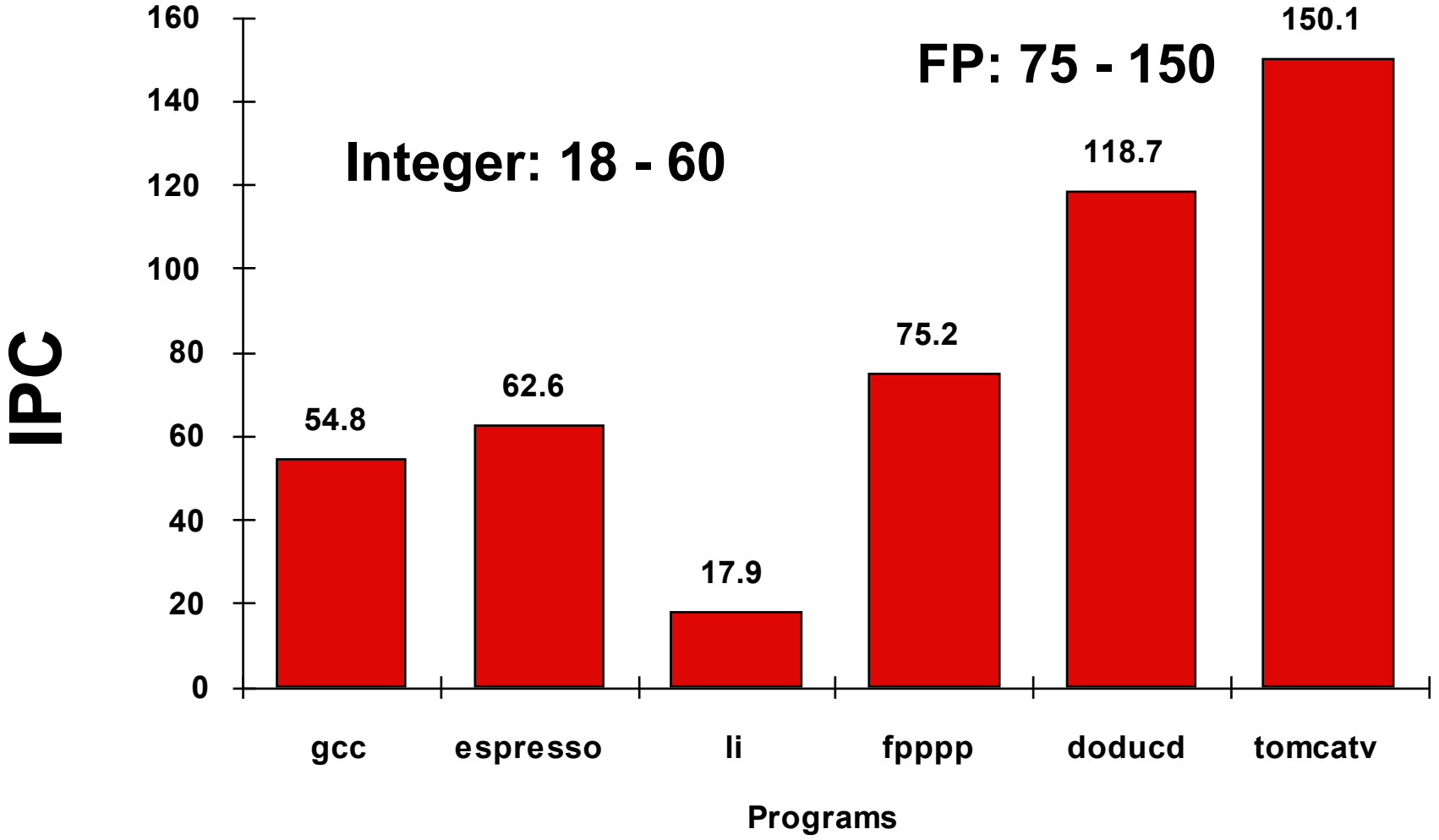
2. *Branch prediction*–perfect; no mispredictions

3. *Jump prediction*–all jumps perfectly predicted => machine with perfect speculation & an unbounded buffer of instructions available

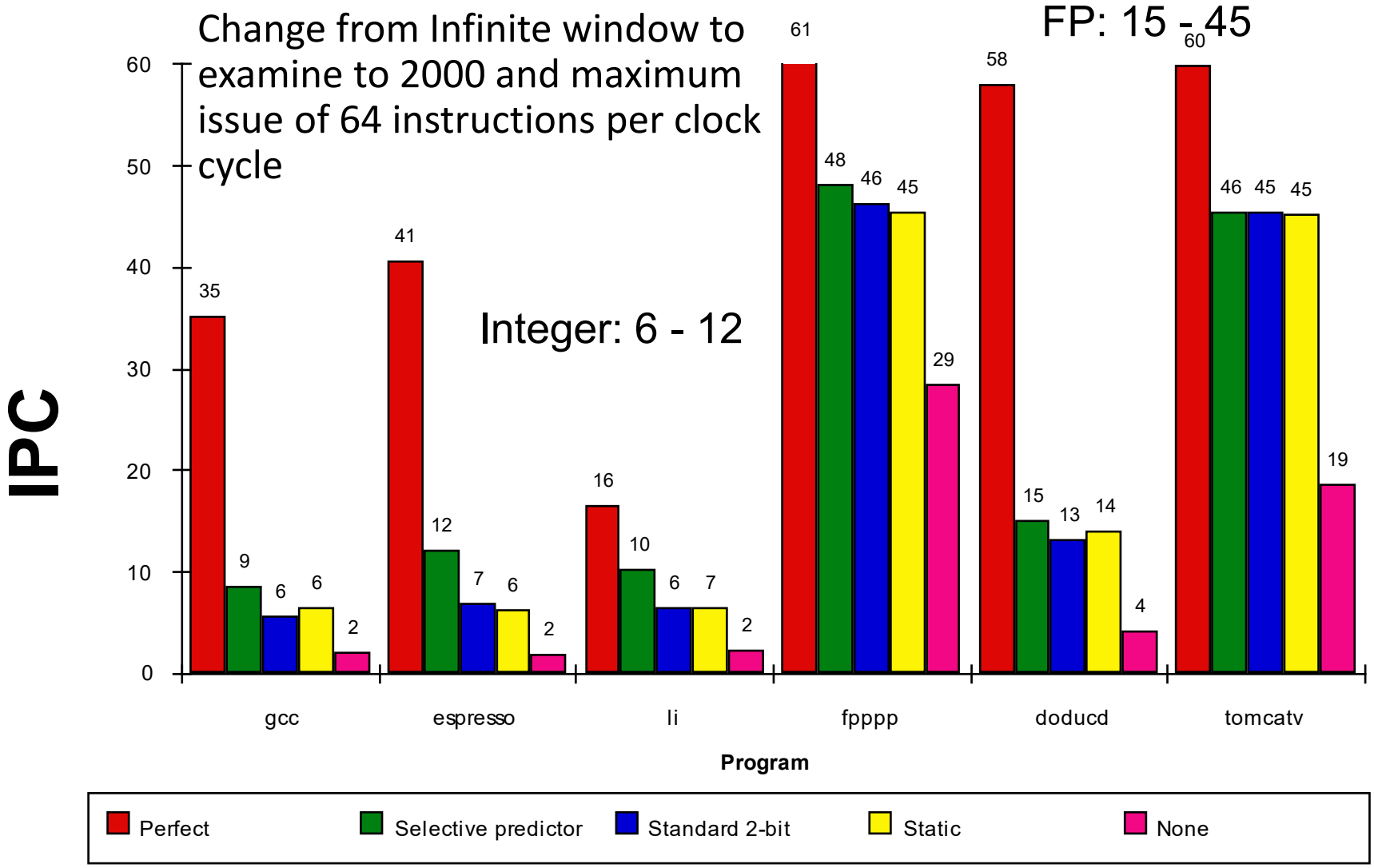
4. *Memory-address alias analysis*–addresses are known & a load can be moved before a store provided addresses not equal

1 cycle latency for all instructions; unlimited number of instructions issued per clock cycle

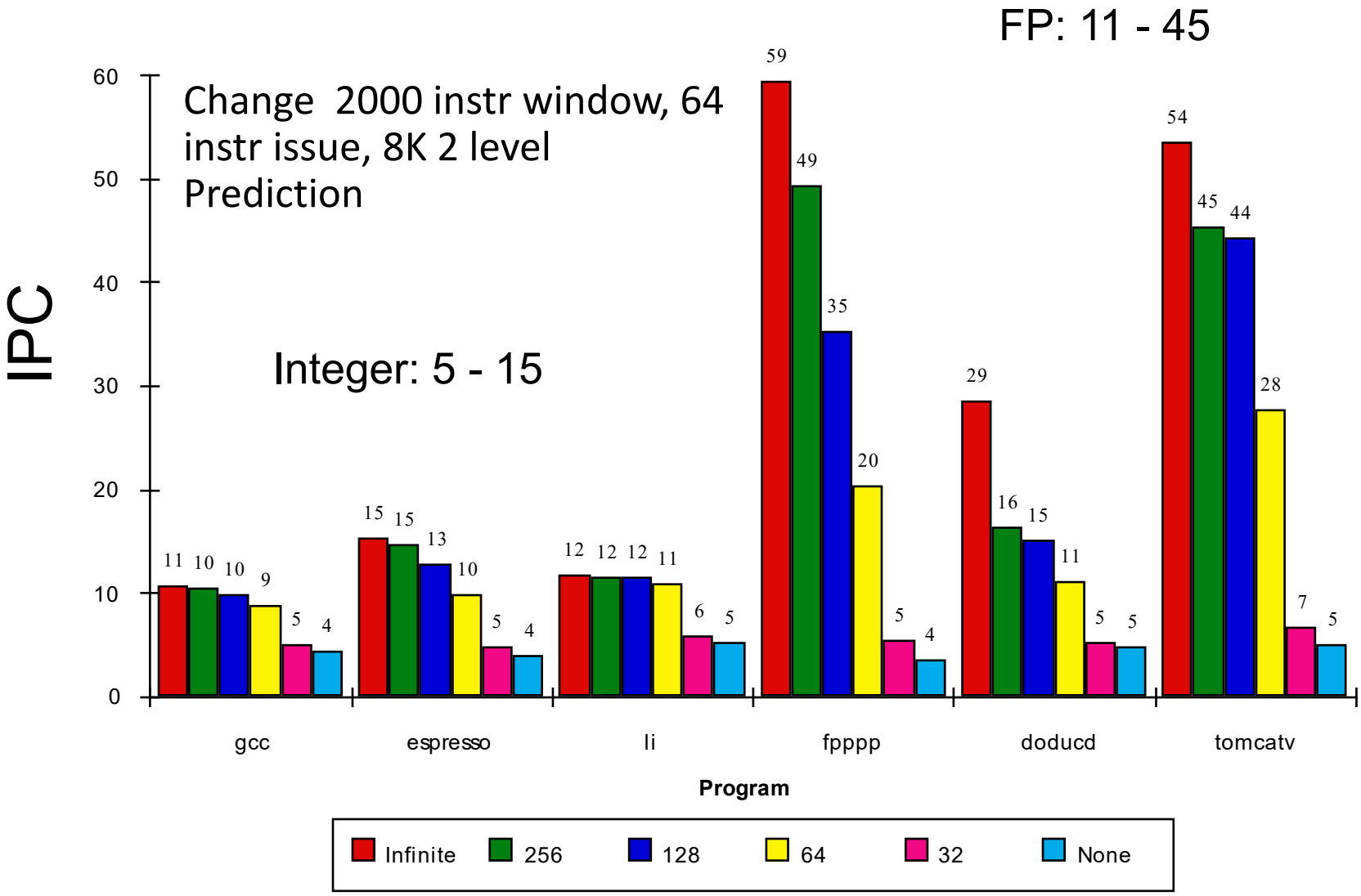
Upper Limit to ILP: Ideal Machine



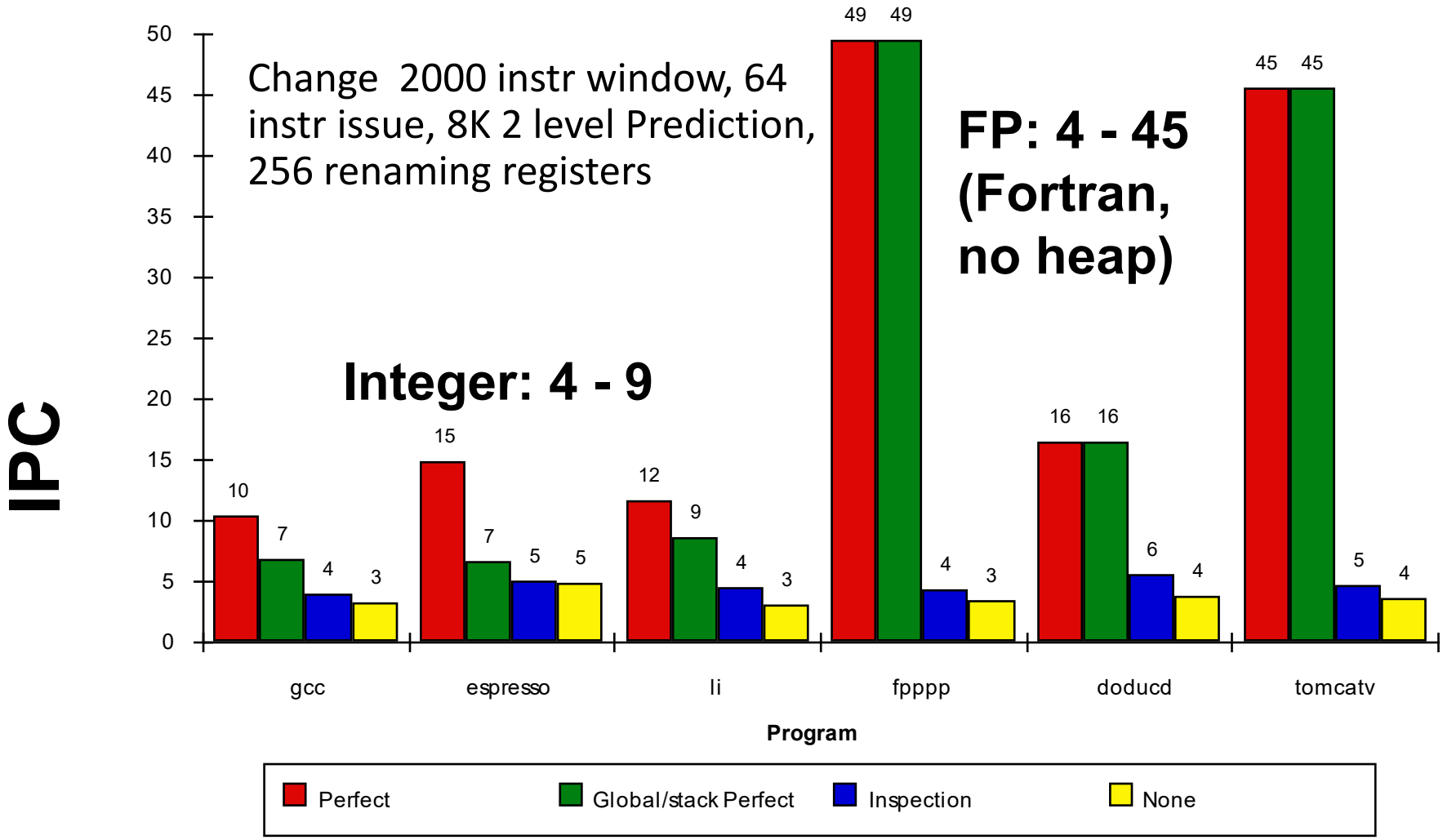
More Realistic HW: Branch Impact



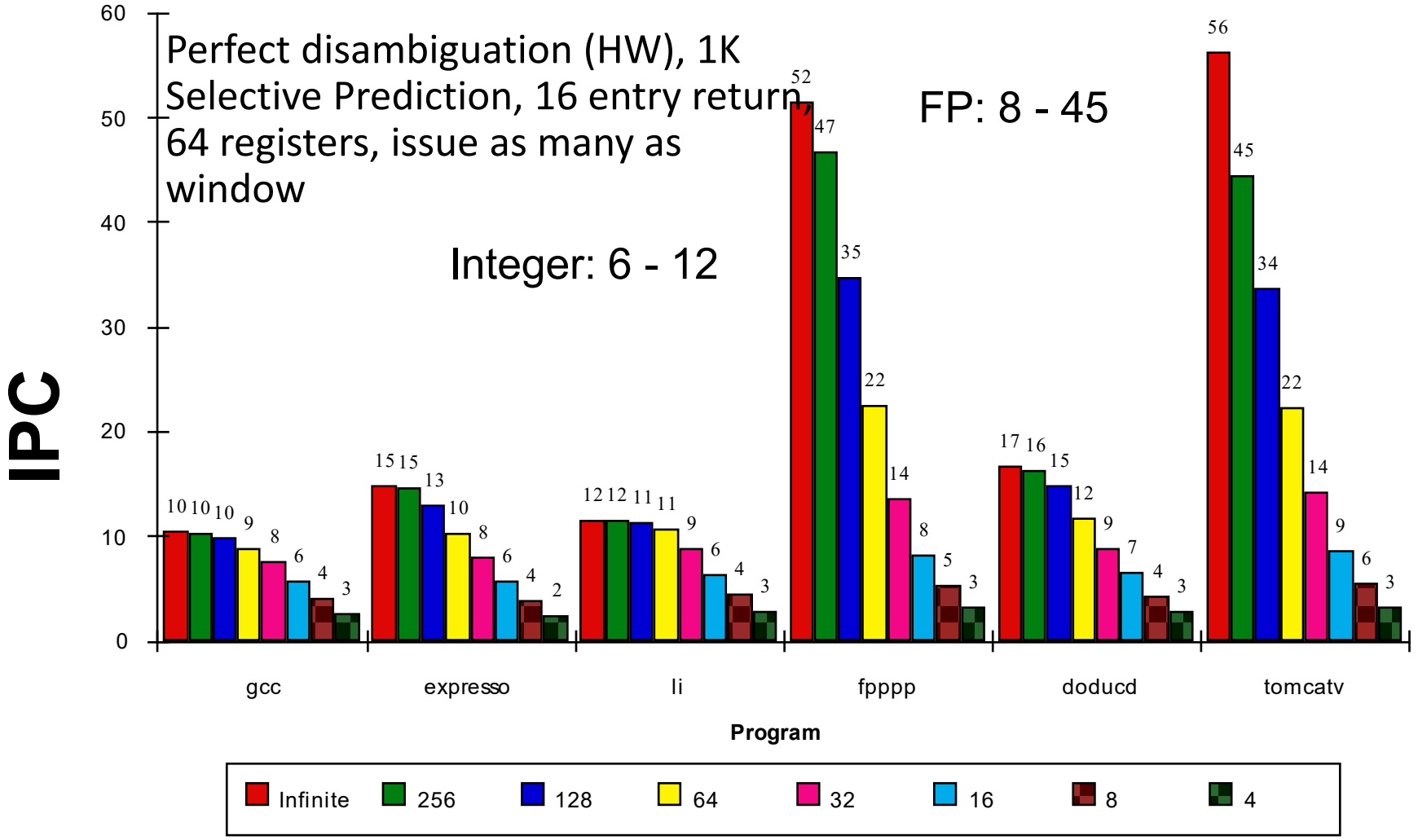
More Realistic HW: Register Impact



More Realistic HW: Alias Impact



Realistic HW for '9X: Window Impact



Speculative Processing with Map tables