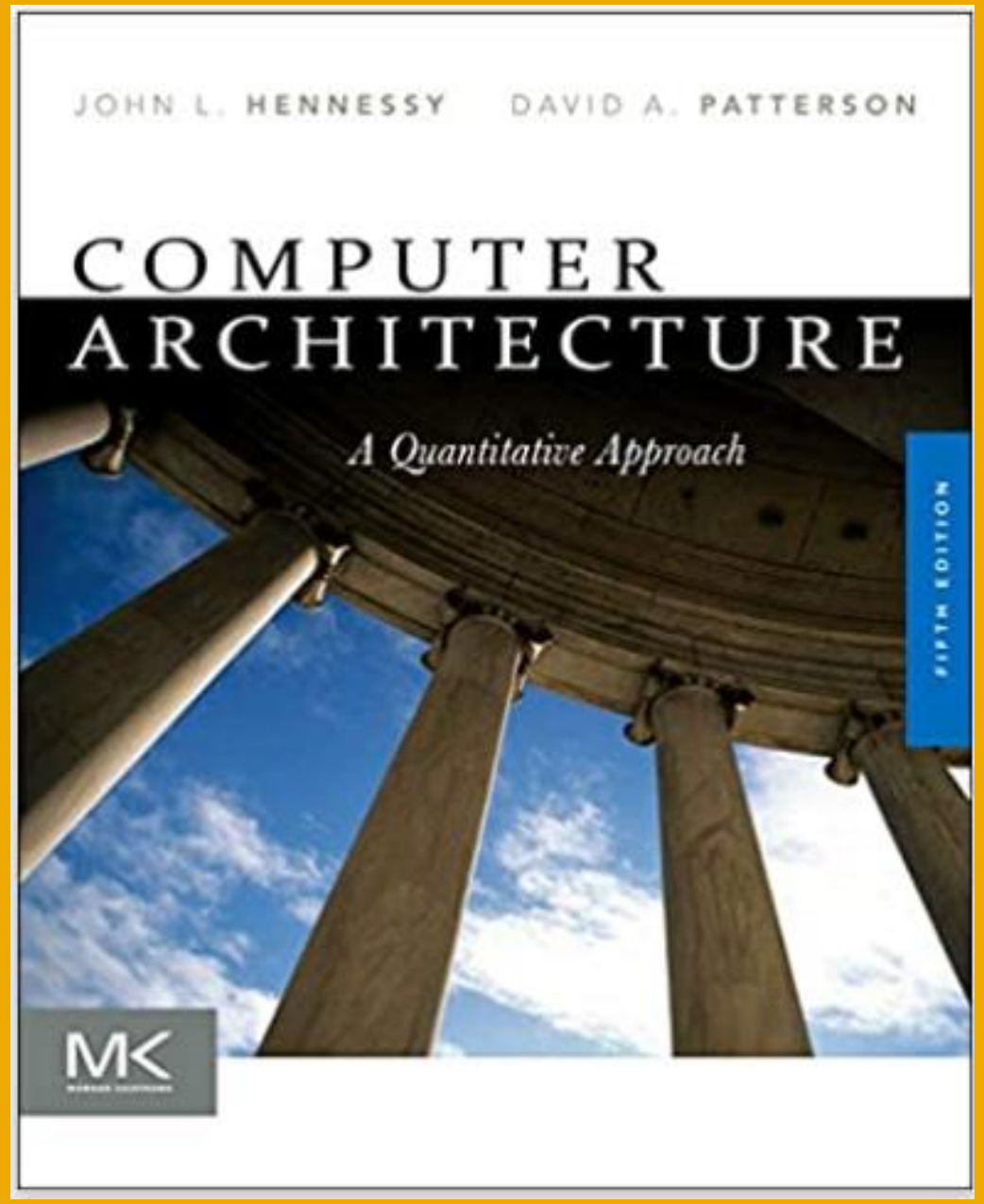
Advanced Microarchitecture

Can we make processors faster

- Biggest tradeoffs:
  - Power, cost, and speed
- Biggest advantage:
  - Transistors are becoming faster and smaller by the day
  - Billions of transistors on one chip
  - Build more complex systems than ever.
- Key elements of performance are still the same:
  - Number of instructions
  - Clock per instruction
  - Period of the clock

Advanced Microarchitecture

- Deep Pipelining
- Micro-operations
- Branch Prediction
- Superscalar Processors
- Out of Order Processors
- Register Renaming
- SIMD
- Multithreading
- Multiprocessors


JOHN L. HENNESSY
DAVID A. PATTERSON
COMPUTER ARCHITECTURE
A Quantitative Approach
FIFTH EDITION
MK

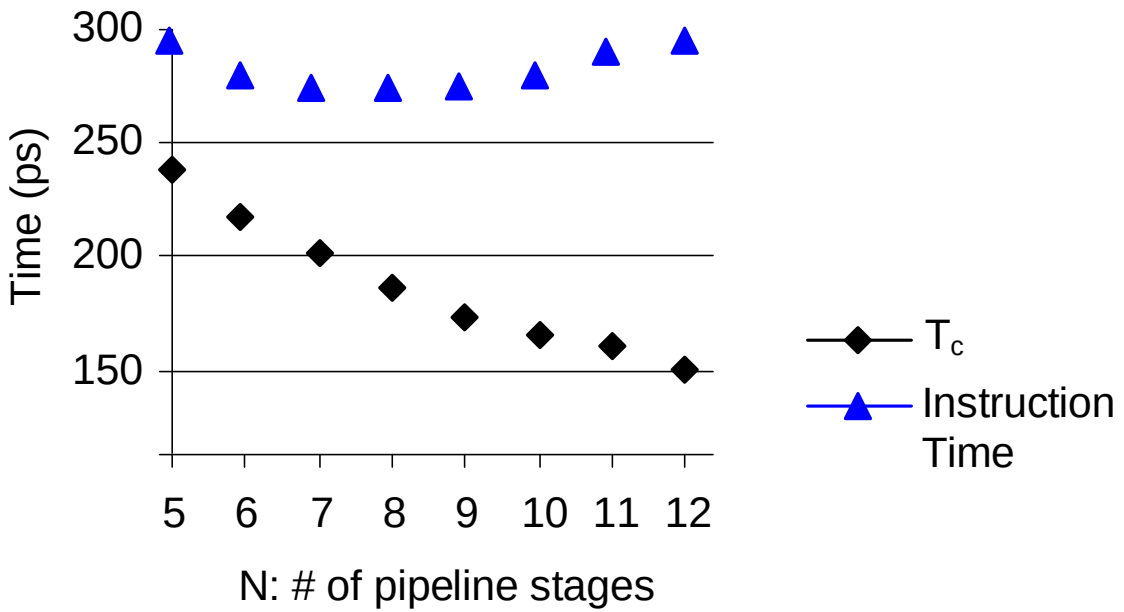
Deep Pipelining

- **10-20 stages typical**
- Number of stages limited by:
  - Pipeline hazards
  - Sequencing overhead
  - Power
  - Cost

Analysis Example

- Single cycle propagation delay is 750ps
- Sequencing overhead is 90ps (setup and hold time)
- CPI for five stages is 1.25.
- Assumptions:
  - CPI increases by 0.1 with every extra stage
  - The clock period is inversely proportional to the number of stages plus the sequencing overhead
- Instruction time is defined as: $CPI \times T_c$
  - $CPI = 1.25 + 0.1(N - 5)$
  - $T_c = \frac{750}{N} + 90$ ps

Analysis Example



- For N=8 the instruction time is 281 ps versus 295 ps for five stages
- The tradeoff is not good: too much power and cost for minimal improvement
- SWeRV EH1 (open source RISC-V processor developed by Western digital) has 9 stages

Micro-operations

- Decompose complex instructions into series of simple instructions called ***micro-operations*** (*micro-ops* or *µ-ops*)
- **At run-time**, complex instructions are decoded into one or more micro-ops
- Used heavily in **CISC** (complex instruction set computer) architectures (e.g., x86)

Complex Op

```
lw s1, 0(s2), postincr 4
```

Micro-op Sequence

```
lw   s1, 0(s2)
addi s2, s2, 4
```

Without µ-ops, would need 2nd write port on the register file

Micro-operations: Example 2

- ADD [ESP], [EDX+80+EDI*2] =>

```
Slli t2, EDI, 1
add t1, EDX, t2
lw t1, 80(t1)
lw t2, 0(ESP)
add t1, t2, t1
sw t1, 0 (ESP)
```

- Instead of requiring more register files being read, more ports on memory, more executables being done at the same time.
- Transform the CISC into set of micro-operations during run time.

Branch Prediction

- **Guess** whether branch will be taken
  - Backward branches are usually taken (loops)
  - Consider history to improve guess
- Good prediction **reduces fraction of branches requiring a flush**

Branch Prediction

- Ideal pipelined processor: CPI = 1
- Branch misprediction increases CPI
- **Static branch prediction:**
  - Check direction of branch (forward or backward)
  - If backward, predict taken
  - Else, predict not taken
- **Dynamic branch prediction:**
  - Keep **history** of last several hundred (or thousand) branches in *branch target buffer*, record:
    - Branch destination
    - Whether branch was taken

Dynamic Branch Prediction

- 1-bit branch predictor
- 2-bit branch predictor
- Both should be done in the fetch stage
- Good dynamic branch predictors can achieve a 90% accuracy.

Branch Prediction Example

```
    addi s1, zero, 0       # s1 = sum
    addi s0, zero, 0       # s0 = i
    addi t0, zero, 10      # t0 = 10

For:                       # for (i=0; i<10; i=i+1)
    bge  s0, t0, Done
    add  s1, s1, s0        # sum = sum + i
    addi s0, s0, 1         # i = i + 1
    j    For

Done:
```

1-Bit Branch Predictor

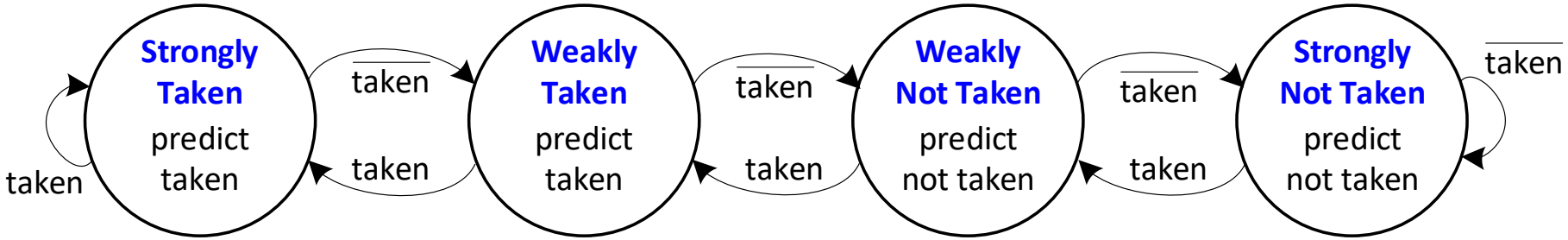
- **Remembers** whether branch was taken the last time and **does the same thing**
- Mispredicts first and last branch of loop

```
  addi s1, zero, 0      # s1 = sum
  addi s0, zero, 0      # s0 = i
  addi t0, zero, 10     # t0 = 10

For:                    # for (i=0; i<10; i=i+1)
  bge  s0, t0, Done
  add  s1, s1, s0       # sum = sum + i
  addi s0, s0, 1        # i = i + 1
  j    For

Done:
```

2-Bit Branch Predictor




```
    addi s1, zero, 0      # s1 = sum
    addi s0, zero, 0      # s0 = i
    addi t0, zero, 10     # t0 = 10

For:                      # for (i=0; i<10; i=i+1)
    bge  s0, t0, Done
    add  s1, s1, s0       # sum = sum + i
    addi s0, s0, 1        # i = i + 1
    j    For

Done:
```

Only mispredicts **last branch** of loop