Chapter 7: Microarchitecture

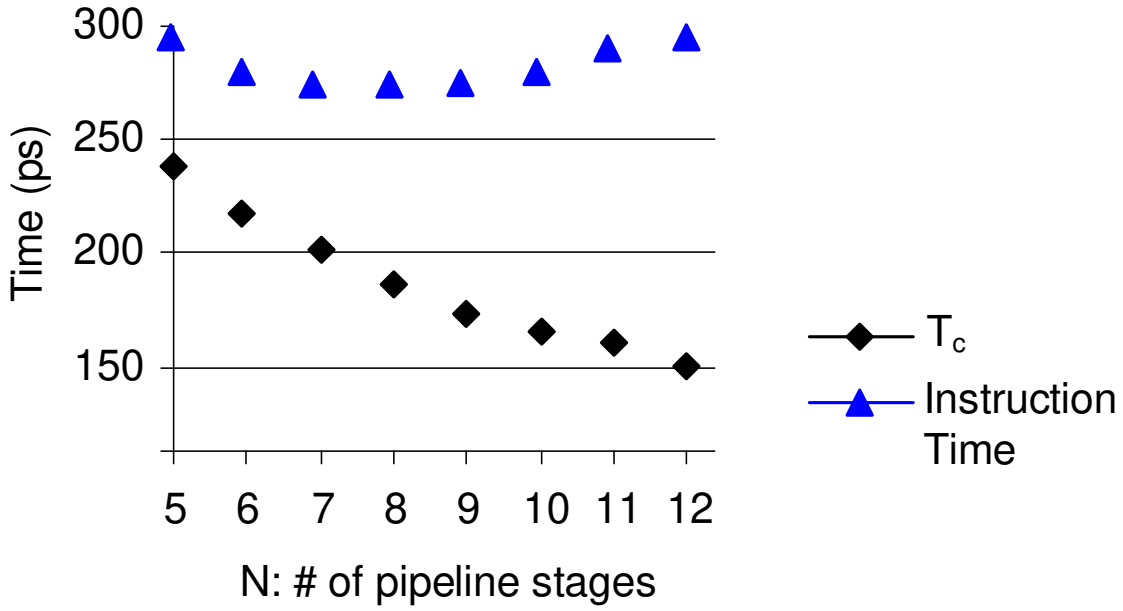
Advanced Microarchitecture

Advanced Microarchitecture

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

Deep Pipelining

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



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

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

Branch Prediction Example

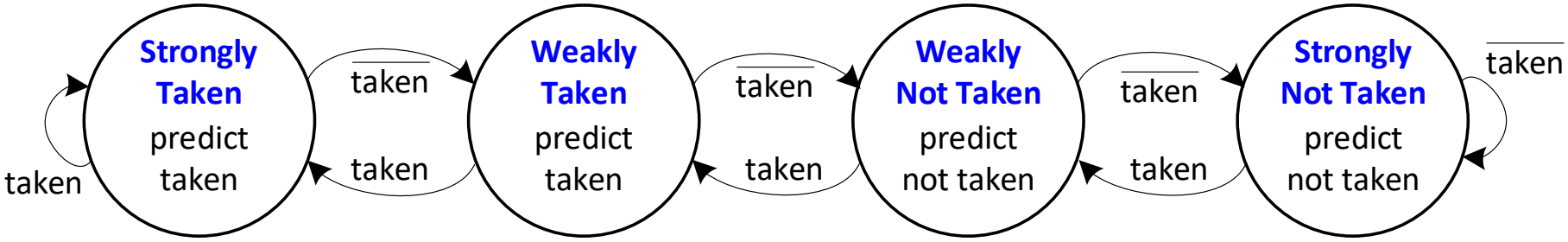
```
    addi s1, zero, 0        # s1 = sum
    addi s0, zero, 0        # s0 = i
    addi t0, zero, 10       # t0 = 10

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:
```

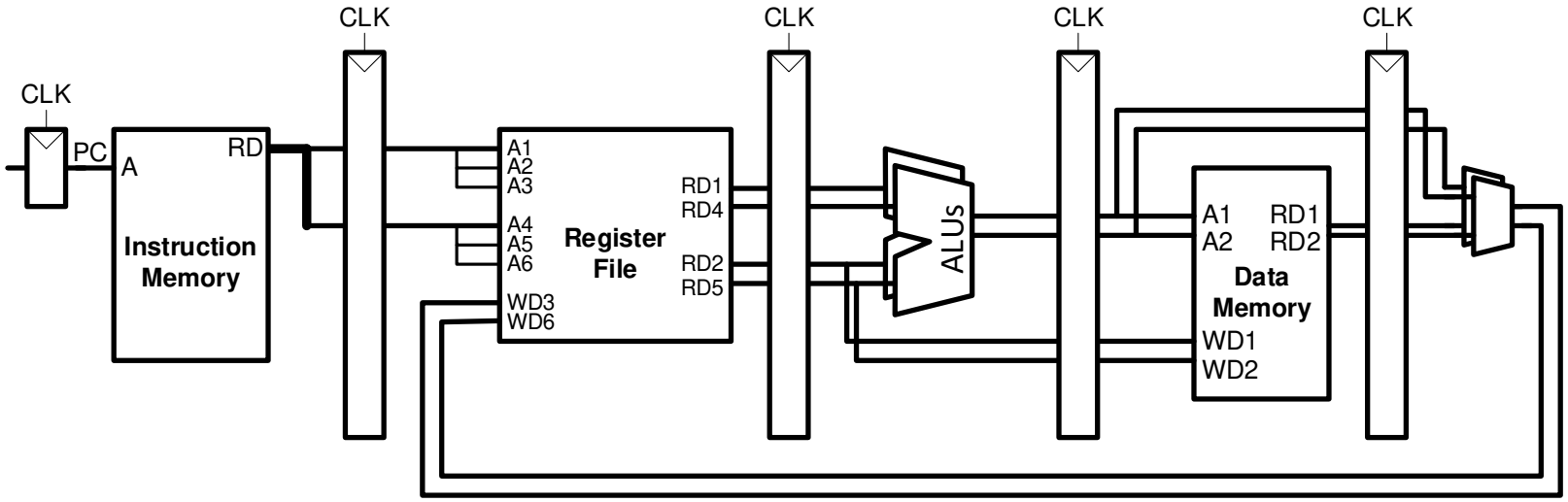
Only mispredicts **last branch** of loop

Chapter 7: Microarchitecture

Superscalar & Out of Order Processors

Superscalar Processors

- Multiple copies of datapath execute multiple instructions at once
- Dependencies make it tricky to issue multiple instructions at once

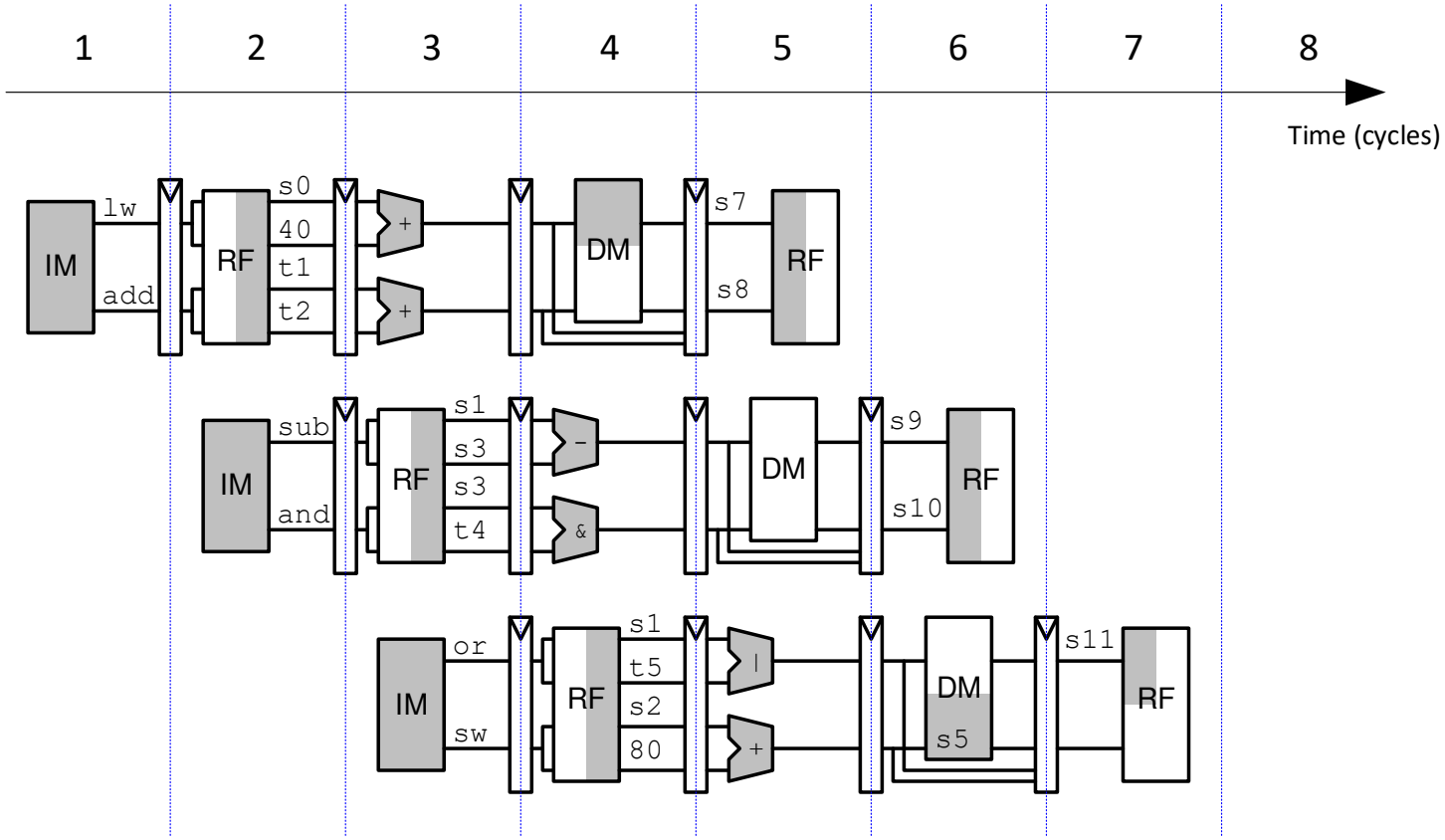


Superscalar Example

Ideal IPC: **2**

Actual IPC: **2**

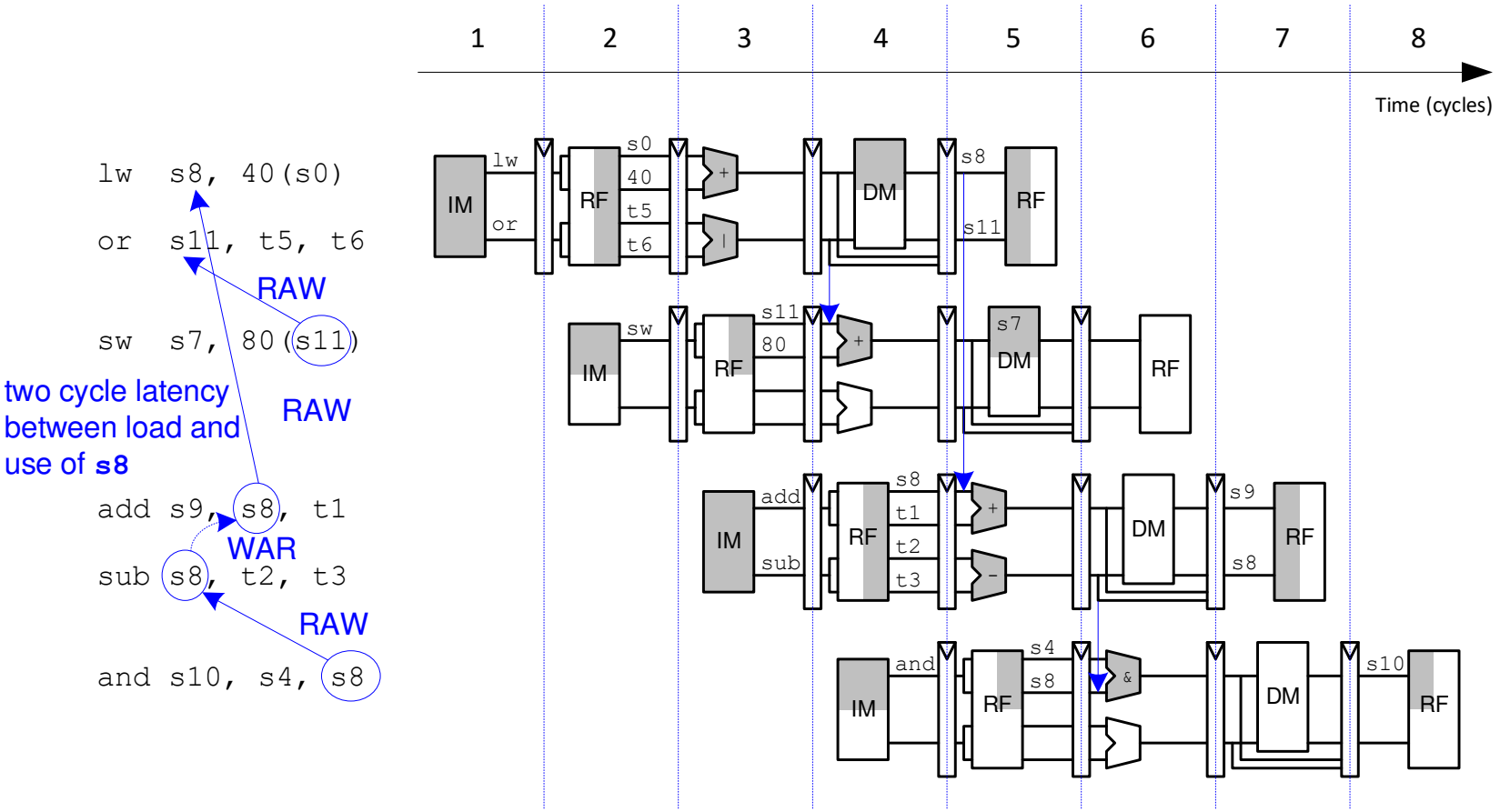
```
lw   s7, 40(s0)
add  s8, t1, t2

sub  s9, s1, s3
and  s10, s3, t4

or   s11, s1, t5
sw   s5, 80(s2)
```



Superscalar with Dependencies

Ideal IPC: **2**

Actual IPC: **6/5 = 1.2**



Out of Order (OOO) Processor

- Looks ahead across multiple instructions
- Issues as many instructions as possible at once
- Issues instructions out of order (as long as no dependencies)
- **Dependencies:**
  - **RAW** (read after write): one instruction writes, later instruction reads a register
  - **WAR** (write after read): one instruction reads, later instruction writes a register
  - **WAW** (write after write): one instruction writes, later instruction writes a register

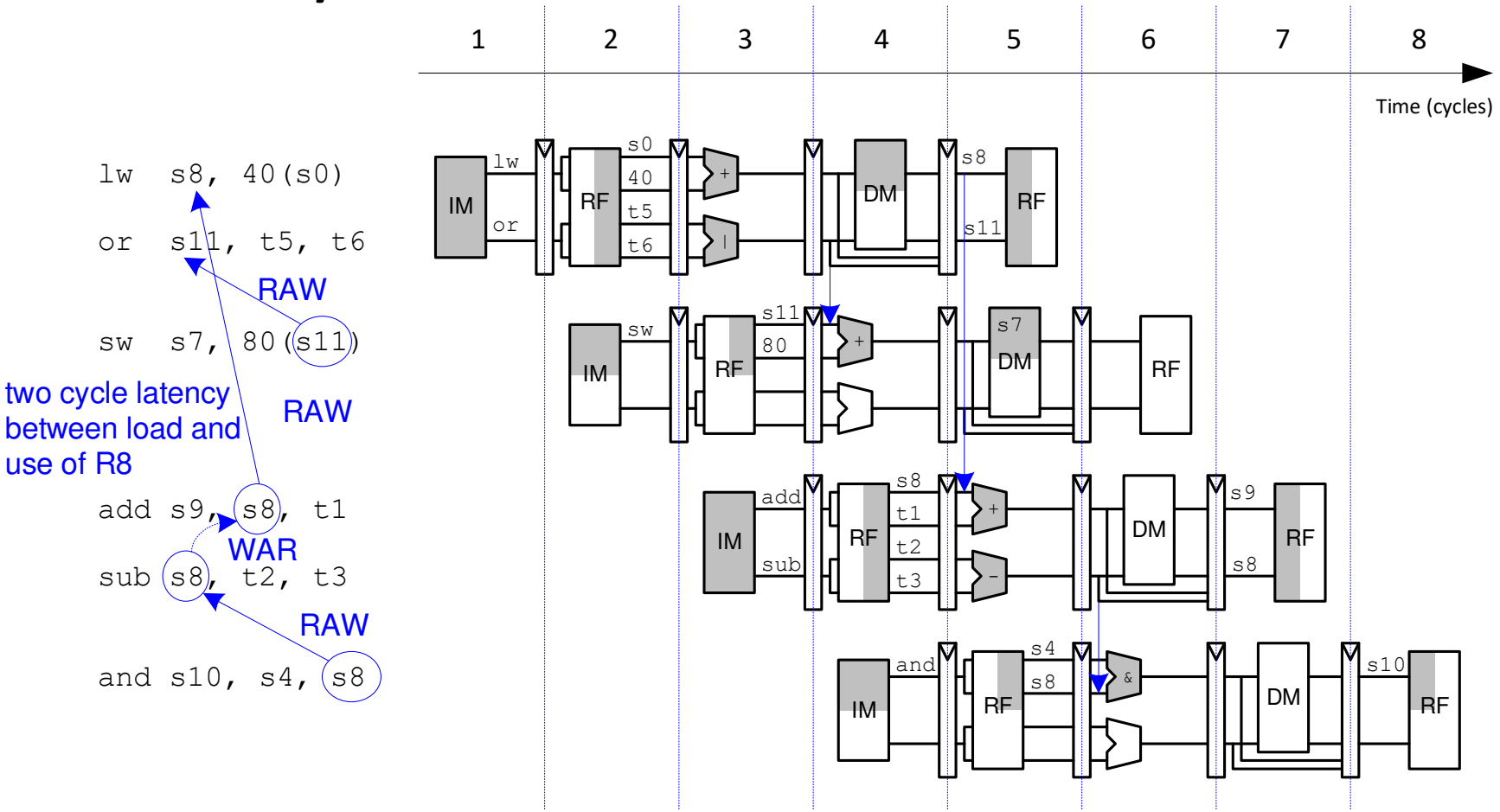
Out of Order (OOO) Processor

- **Instruction level parallelism (ILP):** number of instruction that can be issued simultaneously (average < 3)
- **Scoreboard:** table that keeps track of:
  - Instructions waiting to issue
  - Available functional units
  - Dependencies

Out of Order Processor Example

Ideal IPC: **2**

Actual IPC: **6/4 = 1.5**



Register Renaming

Ideal IPC: **2**

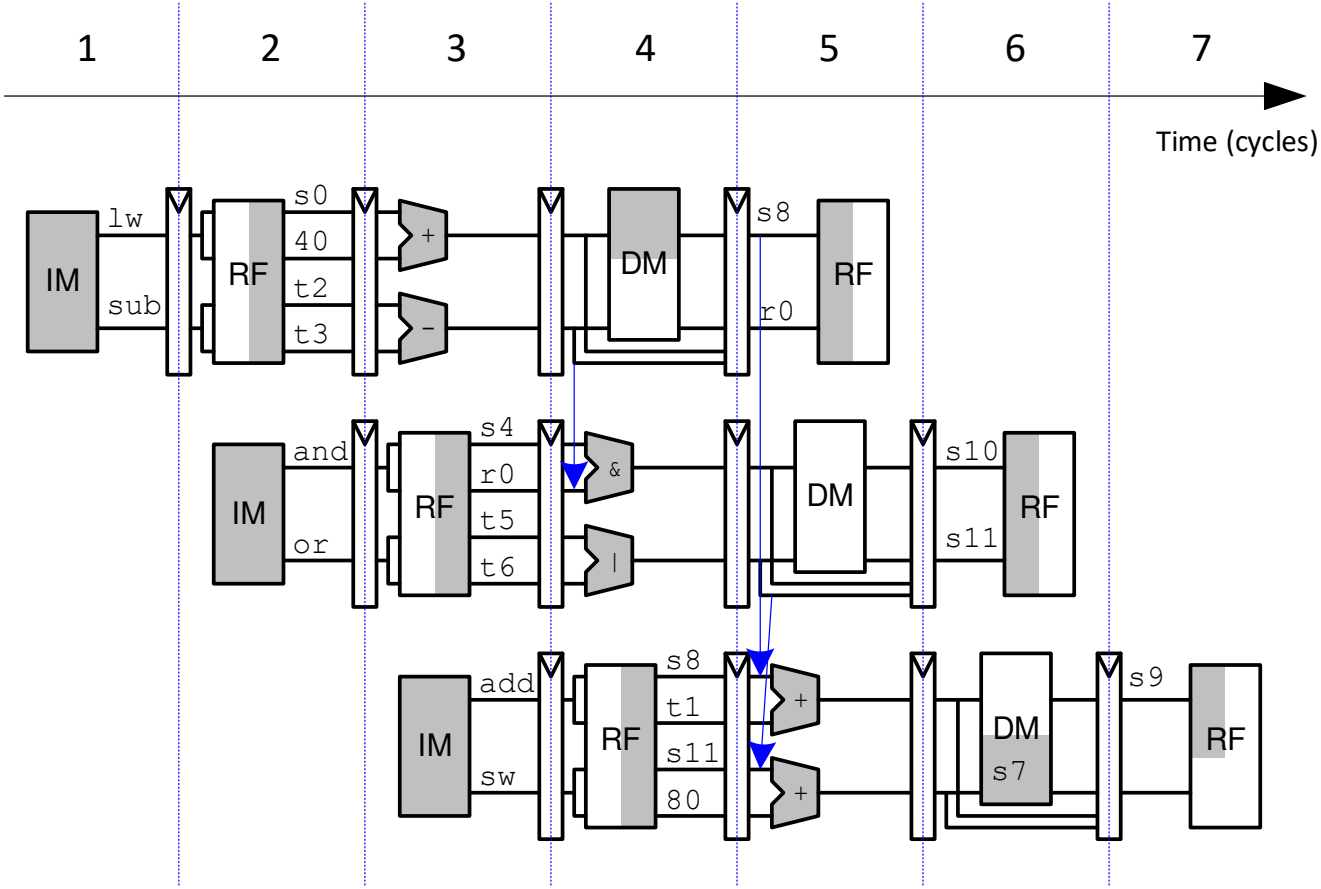
Actual IPC: **6/3 = 2**

```
lw   s8, 40(s0)
sub  r0, t2, t3
and  s10, s4, r0
or   s11, t5, t6
add  s9, s8, t1
sw   s7, 80(s11)
```

2-cycle RAW

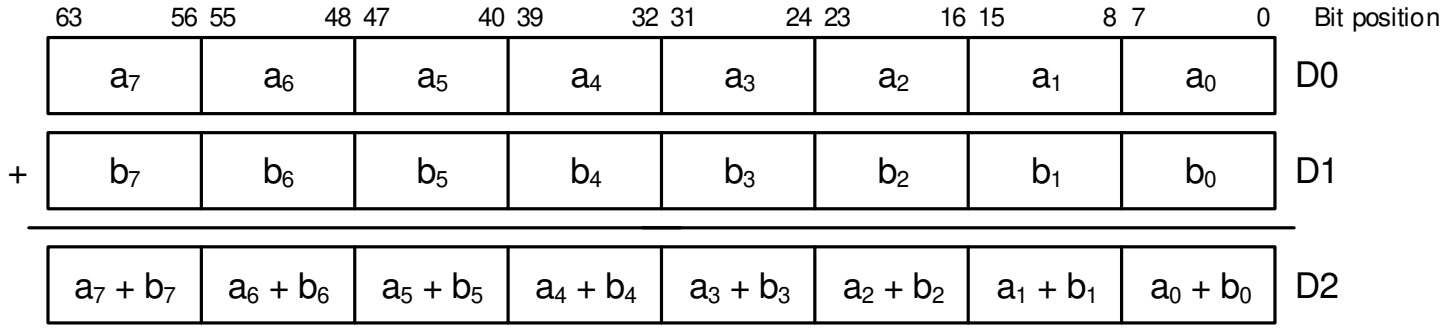
RAW

RAW



SIMD

- **Single Instruction Multiple Data** (SIMD)
  - Single instruction **acts on multiple pieces of data at once**
  - Common application: **graphics**
  - Can apply to short arithmetic operations (also called *packed arithmetic*)
- For example, add eight 8-bit elements



Chapter 7: Microarchitecture

Multithreading & Multiprocessors

Advanced Architecture Techniques

- **Multithreading**
  - Wordprocessor: thread for typing, spell checking, printing
- **Multiprocessors**
  - Multiple processors (cores) on a single chip

Threading: Definitions

- **Process:** program running on a computer
  - Multiple processes can run at once: e.g., surfing Web, playing music, writing a paper
- **Thread:** part of a program
  - Each process has multiple threads: e.g., a word processor may have threads for typing, spell checking, printing

Threads in a Conventional Processor

Single-core system:

- One thread runs at once
- When one thread stalls (for example, waiting for memory):
  - Architectural state of that thread stored
  - Architectural state of waiting thread loaded into processor and it runs
  - Called **context switching**
- Appears to user like all threads running simultaneously

Multithreading

- Multiple copies of architectural state
- Multiple threads **active** at once:
  - When one thread stalls, another runs immediately
  - If one thread can't keep all execution units busy, another thread can use them
- Does not increase instruction-level parallelism (ILP) of single thread, but increases throughput

Intel calls this "hyperthreading"

Multiprocessors

- Multiple processors (cores) with a method of communication between them
- Types:
  - **Homogeneous:** multiple cores with shared main memory
  - **Heterogeneous:** separate cores for different tasks (for example, DSP and CPU in cell phone)
  - **Clusters:** each core has own memory system

About these Notes

Digital Design and Computer Architecture Lecture Notes

© 2021 Sarah Harris and David Harris

These notes may be used and modified for educational and/or non-commercial purposes so long as the source is attributed.