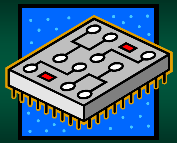
Design Principles

Part 10

1



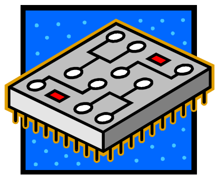
Technological Trends

Change is fast

2

Technological Trends

- Since the design of the integrated circuit, computers have advanced *dramatically*
- Home computer's today have more power than mainframes did 30 years ago
- A hand calculator has more power than the computer that took us to the Moon



3

Integrated Circuits Improved In…

- *Density* – total number transistors and wires can be placed in a fixed area on a silicon chip
- *Speed* – how quickly basic logic gates and memory devices operate
- *Area* – the physical size of the largest integrated circuit that can be fabricated

4

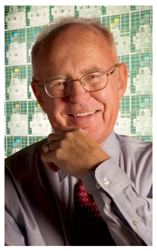
Rate of Improvement

- The increase in performance does not increase at a linear rate
- Speed & Density improves *exponentially*
  - from one year to the next… it has been a relatively constant fraction of the previous year's performance
  - …rather than constant absolute value

5

Moore's Law

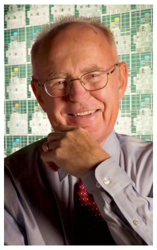
- Gordon Moore is one of the co-founders of Intel
- He first observed (and predicted) computer performance improves *exponentially*, not linearly



6

Moore's Law

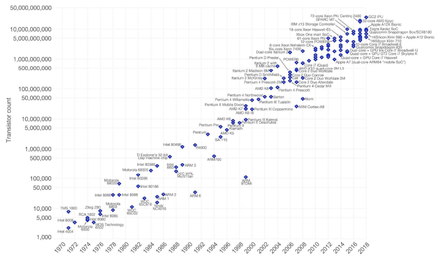
- *Moore's Law* states the performance doubles every 18 months
- This law has held for nearly 50 years



7

Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

8

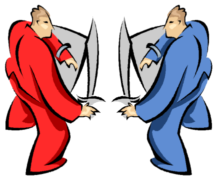
CISC vs. RISC

How do we tip the scales?

9

CISC vs. RISC

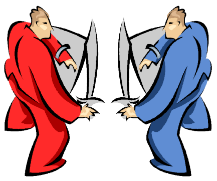
- There is, an often contentious, debate on how to design a processor
- For instance:
  - how is memory going to be accessed
  - what instructions are needed
  - how to encode/structure them



10

CISC vs. RISC

- Typically the debate comes down to CISC vs. RISC
- Processors are typically put into these two categories
- Rarely is a processor "pure" RISC or CISC
- It's a *design philosophy* with a large "gray" area

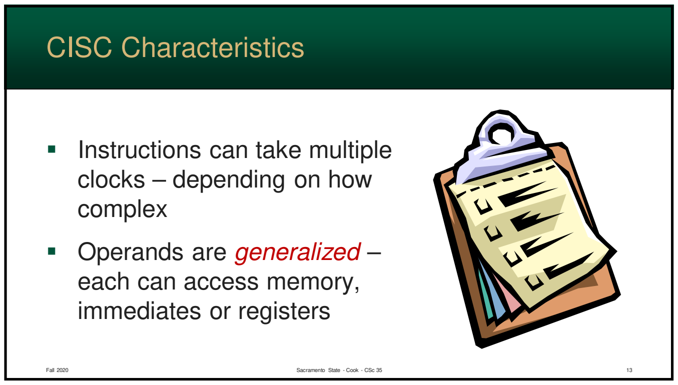


11

CISC

- Complex Instruction Set Computer (CISC) emphasizes flexibility in instructions
- Hardware should contain the complexity rather than the software

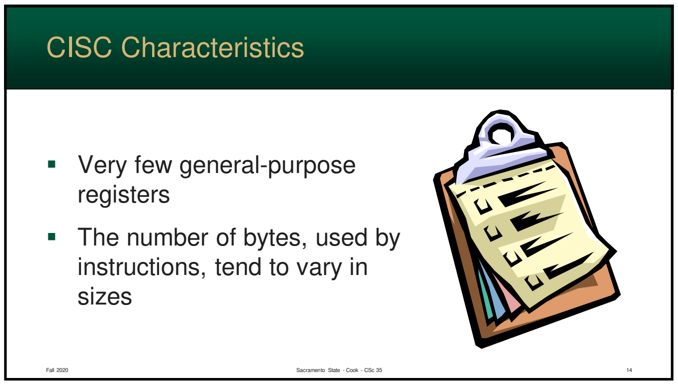
12



CISC Characteristics

- Instructions can take multiple clocks – depending on how complex
- Operands are *generalized* – each can access memory, immediates or registers

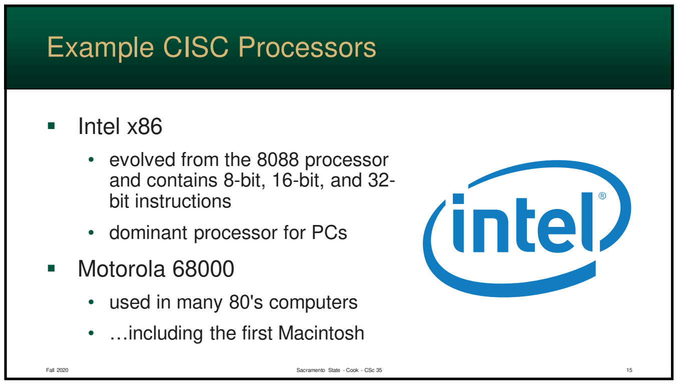
13



CISC Characteristics

- Very few general-purpose registers
- The number of bytes, used by instructions, tend to vary in sizes

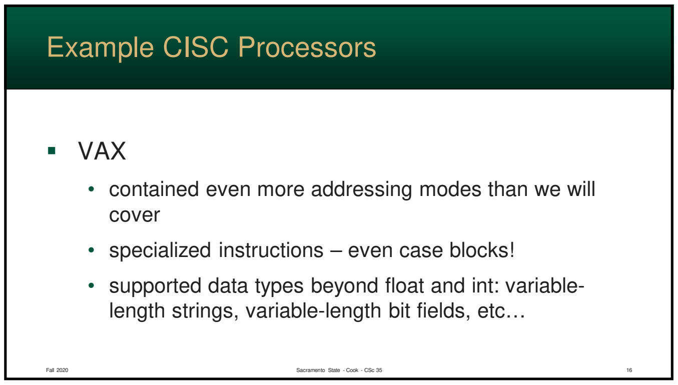
14



Example CISC Processors

- Intel x86
  - evolved from the 8088 processor and contains 8-bit, 16-bit, and 32-bit instructions
  - dominant processor for PCs
- Motorola 68000
  - used in many 80's computers
  - …including the first Macintosh

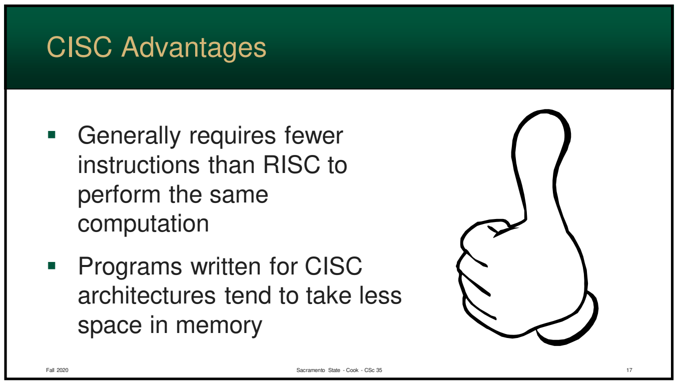
15



Example CISC Processors

- VAX
  - contained even more addressing modes than we will cover
  - specialized instructions – even case blocks!
  - supported data types beyond float and int: variable-length strings, variable-length bit fields, etc…

16



CISC Advantages

- Generally requires fewer instructions than RISC to perform the same computation
- Programs written for CISC architectures tend to take less space in memory

17

Please Wait

CSC 35 will start shortly

Lecture today.
Review next class.

18

Moore's Law and CISC

- Computer speed through the 1980's grew exponentially
- However, …
  - rate of processor growth has been far greater than memory
  - so, memory *relative to the processor's speed* has gotten much slower

19

Memory is the Bottleneck

- CISC can access memory with nearly every instruction
- Memory is slow compared to register-to-register operations
- It is far more efficient (now) to do all work on the processor and use memory **only** when absolutely necessary

20

RISC

- Reduced Instruction Set Computer (RISC) emphasizes simplicity
- Software should contain the complexity rather than hardware

21

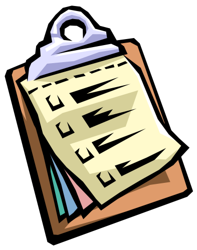
RISC

- So, RISC contains fewer instructions than CISC – only the minimum needed to work
- Minimalize memory access
  - only a few instructions can access memory
  - usually limited to register load and store instructions

22

RISC Characteristics

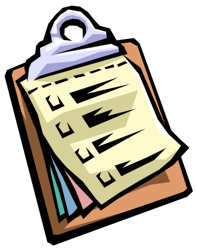
- Access to memory is restricted to load/store instructions
- Many registers – since all instructions can only use registers for calculations



23

RISC Characteristics

- Instructions typically take one clock cycle each
- The number of bytes, used by instructions, tend to fixed in size (all 32-bit, for example)



24

Example RISC Processors

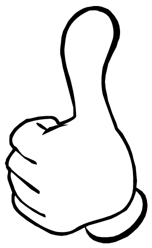
- ARM
  - dominant processor used by smartphones
  - designed to reduce transistors
  - less cost, less heat, less power
- IBM PowerPC 601
  - developed in by IBM, Apple, and Motorola (AIM)
  - used by 1990's Macs

ARM POWERED

25

RISC Advantages

- Simpler instructions make it easier to implement on different processors – and make them more efficient
- Easier to program and master – less to learn
- Memory access is minimalized



26

RISC vs. CISC Comparison

CISC	RISC
Emphasis on hardware complexity	Emphasis on software complexity
Most operands can access memory	Load/Store instructions can access memory
Low number of registers	Higher number of registers
Instructions can have multiple clock cycles	Instructions tend towards one per clock cycle
Encoded instructions vary in size	Encoded instructions are all the same size.

27

Latest Approach

- After the 1990s, RISC architectures have incorporated some of most useful complex instructions from CISC architectures
- Rely on their micro-architecture to implement these instructions with little impact on the clock cycle

28

CISC Example (not x86)

```
# n  =  a * (b + c) - d

mov    R1, b
add    R1, c        # b + c

mul    R1, a        # a * (b + c)
sub    R1, d        # a * (b + c) - d

mov    n, R1
```

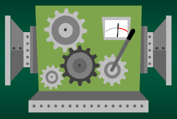
29

RISC Example (not x86)

```
# n  =  a * (b + c) - d

load   R1, b
load   R2, c
add    R2, R1       # b + c
load   R3, a
mul    R2, R3       # a * (b + c
load   R4, d
sub    R2, R4       # a * (b + c) - d
store  n, R2
```

30



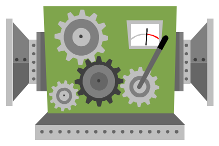
Instruction Execution

The Steps Involved

31

Instruction Execution

- When an instruction executes on a processor, a number of different tasks take place
- Typically, these tasks can be broken into 5 different steps



32

1. Fetch the Instruction

- First, the processor fetches the instruction from the memory
- The result is stored in the Instruction Register
- Formally known as *Instruction Fetch (IF)*

Fetch

33

2. Decode the Instruction

- Second, the instruction is *decoded* to determine what it is and its operands
- Signals are sent to the execution unit as input
- Formally known as *Instruction Decode (ID)*

Decode

34

3. Read Inputs

- Execution Unit then reads the values of the instruction
- These be Immediates, registers, and from memory
- Formally known as *Memory Access (Mem)*

Read

35

4. Execute the Instruction

- Third, values are passed to the ALU
- Depending on the complexity of the instruction, some computations require multiple clock cycles
- Formally: *Execute (EX)*

Execute

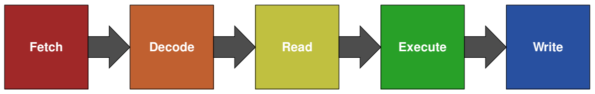
36

5. Write Result

- Finally, the result is written back into the register / memory
- Processor also updates flags and other state information such as the Program Counter
- Formally known as *Write Back (WB)*

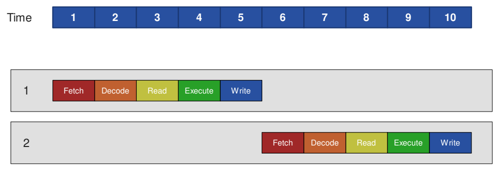
37

Instruction Execution

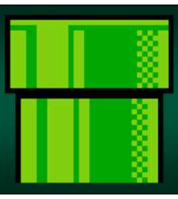


38

Execution of 2 Instructions



39



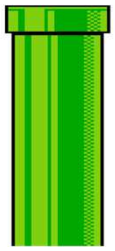
Pipelining

Do multiple things at once

40

Pipelining

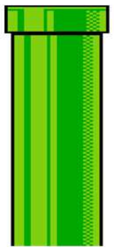
- *Pipelining* is a technique where multiple instructions are executed at the simultaneously
- This greatly improves the speed (and efficiency) of a system



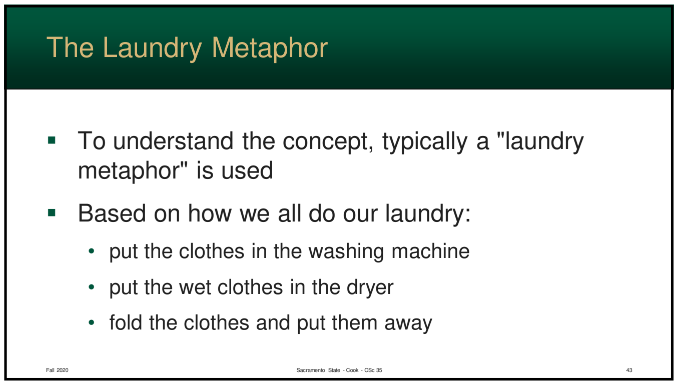
41

Pipelining

- It is invisible to the programmer… implemented by the hardware
- Pipelining is different from multi-core processors!
- Pipeline happens on one core



42

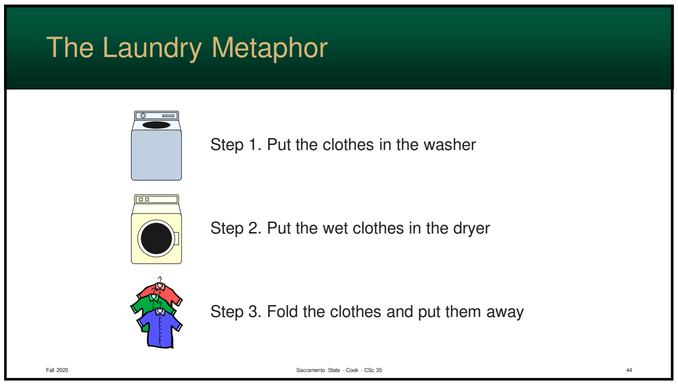


The Laundry Metaphor

- To understand the concept, typically a "laundry metaphor" is used
- Based on how we all do our laundry:
  - put the clothes in the washing machine
  - put the wet clothes in the dryer
  - fold the clothes and put them away

Fall 2020 Sacramento State - Cook - CSc 35 43

43



The Laundry Metaphor

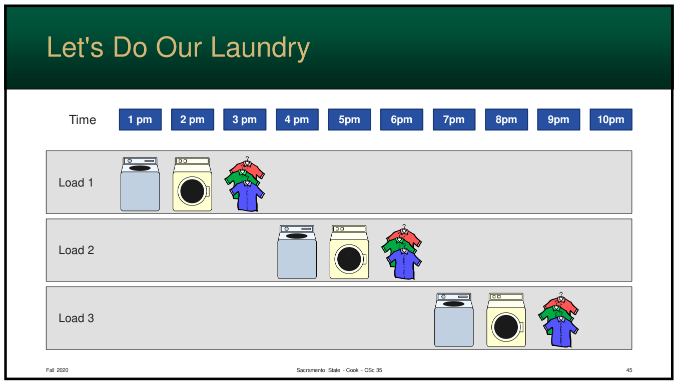
Step 1. Put the clothes in the washer

Step 2. Put the wet clothes in the dryer

Step 3. Fold the clothes and put them away

Fall 2020 Sacramento State - Cook - CSc 35 44

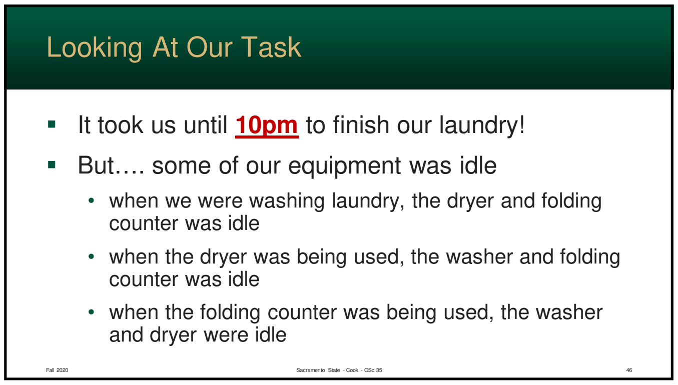
44



Let's Do Our Laundry

Fall 2020 Sacramento State - Cook - CSc 35 45

45

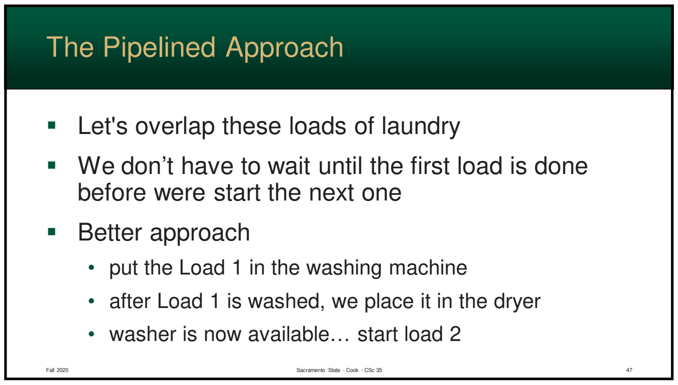


Looking At Our Task

- It took us until **10pm** to finish our laundry!
- But…. some of our equipment was idle
  - when we were washing laundry, the dryer and folding counter was idle
  - when the dryer was being used, the washer and folding counter was idle
  - when the folding counter was being used, the washer and dryer were idle

Fall 2020 Sacramento State - Cook - CSc 35 46

46

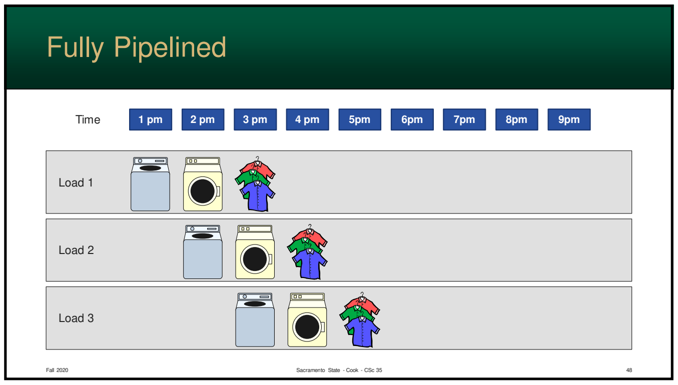


The Pipelined Approach

- Let's overlap these loads of laundry
- We don't have to wait until the first load is done before were start the next one
- Better approach
  - put the Load 1 in the washing machine
  - after Load 1 is washed, we place it in the dryer
  - washer is now available… start load 2

Fall 2020 Sacramento State - Cook - CSc 35 47

47

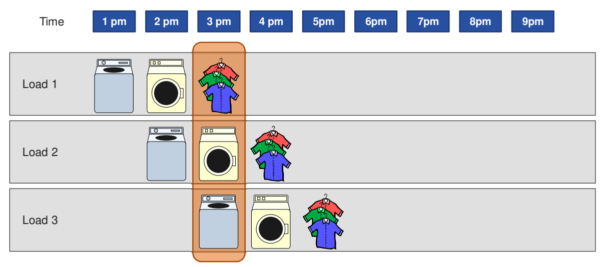


Fully Pipelined

Fall 2020 Sacramento State - Cook - CSc 35 48

48

All Three Components in Use



49

So, is it faster?

- Note: the actual time required to finish a load of laundry has *not* changed
- It still requires the same 3 steps and each takes one hour
- The speedup occurs because the different loads are executed in *parallel*

50

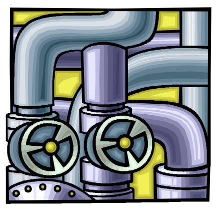
Start-up and Wind-down

- At the beginning of the work load, the pipeline wasn't completely full
- So, the pipeline has to fill before we get our full speedup
- As the number of Loads increases…
  - pipeline is full for a larger fraction of the time
  - so the start-up and wind-down (finishing the last load) becomes meaningless

51

Pipelined Instructions

- Just like the laundry metaphor, processors have different components that can used at the same time
- On modern processors, practically all the hardware is in continuous use



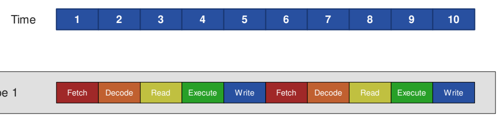
52

Pipelined Instructions

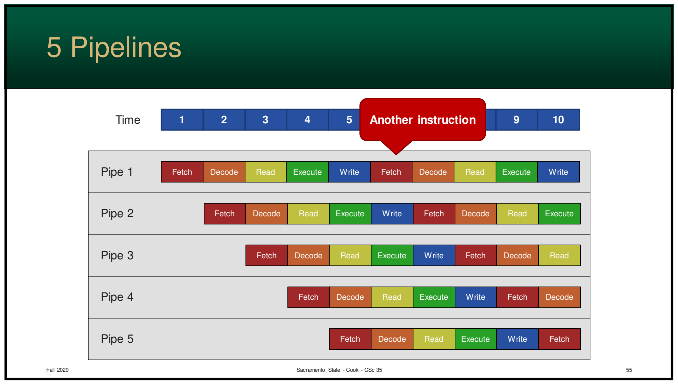
- Ideally, *nothing is ever idle!*
- Different stages of execution are pipelined
  - fetch
  - decode
  - read
  - execution
  - write back

53

Unpipelined Instructions



54



5 Pipelines

Time	1	2	3	4	5				9	10
Pipe 1	Fetch	Decode	Read	Execute	Write	Fetch	Decode	Read	Execute	Write
Pipe 2		Fetch	Decode	Read	Execute	Write	Fetch	Decode	Read	Execute
Pipe 3			Fetch	Decode	Read	Execute	Write	Fetch	Decode	Read
Pipe 4				Fetch	Decode	Read	Execute	Write	Fetch	Decode
Pipe 5					Fetch	Decode	Read	Execute	Write	Fetch

55