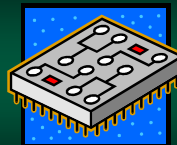




Part 8

Design Principles

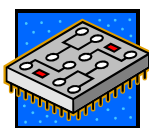


Technological Trends

The Constant Evolution

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



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

3

Integrated Circuits Improved In...

- Density – 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

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

4

Rate of Improvement

- The increase in performance does not increase at a linear rate
- Speed and 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

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

5

Rate of Improvement

- On average...
 - number of transistors that can be fabricated on a silicon chip **increases** by about **50%** per year
 - transistor speed **increases** for basic logic gates (AND, OR, etc.) by **13%** per year

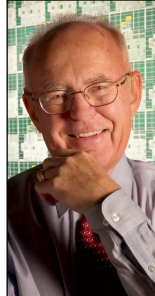
5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

6

Moore's Law

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



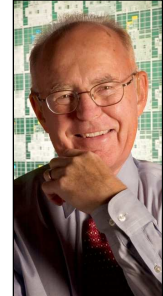
5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

7

Moore's Law

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



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

8

Intel Processors Over Time

Processor	Year	Speed (KHz)	Transistors	Technology
4004	1971	108	2,300	10
8008	1972	800	3,500	10
8080	1974	2,000	4,500	6
8086	1978	5,000	29,000	3
8088	1979	5,000	29,000	3
80286	1982	6,000	134,000	1.5
80386	1985	16,000	275,000	1.5
80486	1989	25,000	1,200,000	1

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

9

Intel Processors Over Time

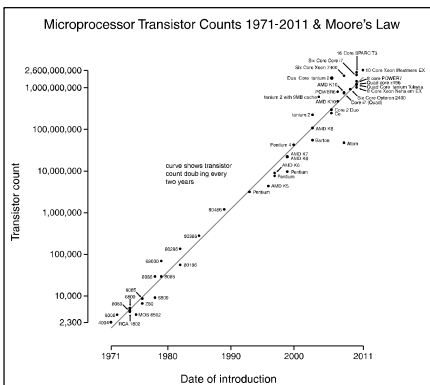
Processor	Year	Speed (KHz)	Transistors	Technology
Pentium	1993	66,000	3,100,000	0.8
Pentium Pro	1995	200,000	5,500,000	0.6
Pentium II	1997	300,000	7,500,000	0.25
Pentium III	1999	500,000	9,500,000	0.18
Pentium 4	2000	1,500,000	42,000,000	0.18
Pentium M	2002	1,700,000	55,000,000	0.13
Pentium D	2005	3,200,000	291,000,000	0.065

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

10

Microprocessor Transistor Counts 1971-2011 & Moore's Law



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

11


The Late 1990's

- Processor performance (per unit energy dissipation) has also improved *exponentially* rather than linearly
- This has made feasible
 - smart phones
 - tablets
 - handheld consoles

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

12




von Neumann Architecture

The Information Super Highway

von Neumann Machine Architecture


- Modern computers are based on the design of John von Neumann
- His design greatly simplified the construction of (and use) computers



5/5/2017 Sacramento State - Cook - CSc 35 - Spring 2017 14

Some von Neumann Attributes

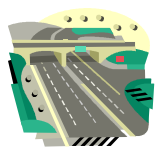
- Programs are stored and executed **in memory**
- Separation** of processing from storage
- Different system components communicate over **shared buses**



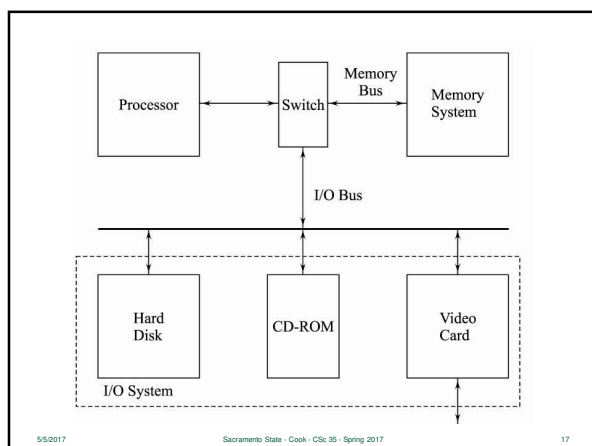
5/5/2017 Sacramento State - Cook - CSc 35 - Spring 2017 15

The Bus

- Electronic pathway that transports data between components
- Think of it as a "highway"
 - data moves on shared paths
 - otherwise, the computer would be very complex




5/5/2017 Sacramento State - Cook - CSc 35 - Spring 2017 16



System Bus

- Interconnects the processor with the memory
- Also called the "system bus" since it interconnects the subsystems)



5/5/2017 Sacramento State - Cook - CSc 35 - Spring 2017 18

System Bus

- The information sent on the memory bus falls into **3** categories
- Three sets of signals
 - address bus
 - data bus
 - control bus



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

19

Address Bus

- Used by the processor to access a specific piece of data
- This "address" can be
 - a specific byte in memory
 - unique IO port
 - etc...



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

20

Address Bus Characteristics

- Total number of bits used in the address limits the total number of bytes that can be accessed
- For an address-size of n bits, you have 2^n memory addresses

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

21

Address Bus Size Examples

- 8-bit \rightarrow 256 bytes
- 16-bit \rightarrow 64 KB (65,536 bytes)
- 32-bit \rightarrow 4 GB (4,294,967,296 bytes)
- 64-bit \rightarrow 18 EB (18,446,744,073,709,551,616)



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

22

Historic Address Sizes

- Intel 8086
 - original 1982 IBM PC
 - 20-bit address bus (1 MB)
 - only 640 KB usable for programs
- MOS 6502 computers
 - Commodore 64, Apple II, Nintendo, etc...
 - 16-bit address bus (64 KB)

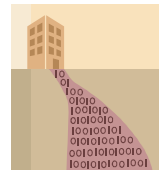
5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

23

Data Bus

- The actual data travels over the **data bus**
- An integer that has the same number of bits as the system is called a **word**



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

24

Data Bus

- Different processors use a different amount of bytes to store and manipulate data
- Example:
 - 8-bit system uses 8 bit integers for data
 - 16-bit system uses 16 bits (2 bytes) for data
 - 32-bit system uses 32 bits (4 bytes) for data
 - etc...

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

25

Data Bus

- Often the processor's address bus and data bus use different bit counts
- Examples:
 - MOS 6502 – 8 bit data, 16 bit address bus
 - Intel 8086 – 16 bit data, 20 bit bus (well 16, but expanded to 20 using a trick)

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

26

Control Bus

- The *control bus* controls the timing and synchronizes the subsystems
- Specifies what is happening
 - read data
 - write data
 - reset
 - etc...

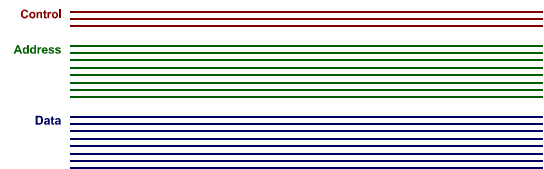


5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

27

The Bus

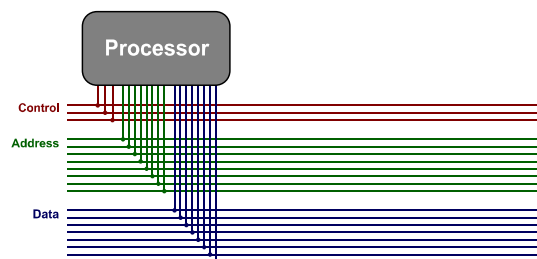


5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

28

Processor

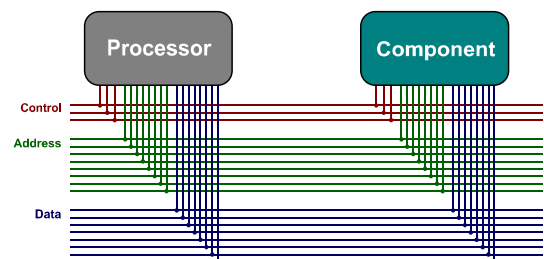


5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

29

Components Are On the Bus



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

30

Reading Data

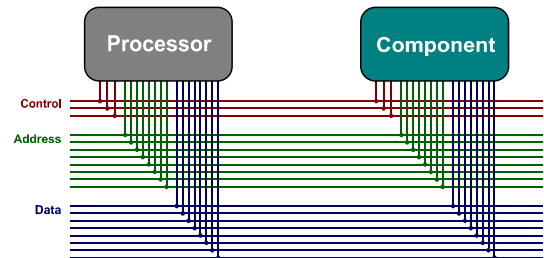
- Processor has the address, but needs data
- Actions:
 1. processor puts the address on the bus
 2. signals the component to read
 3. component reads the address
 4. component puts the data on the bus
 5. processor stores the data

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

31

Reading Data

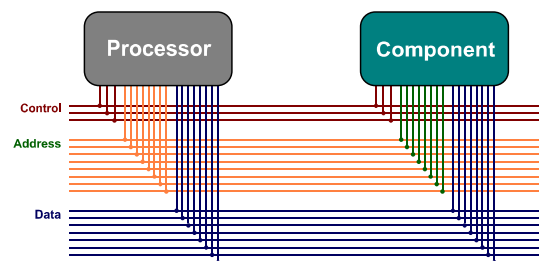


5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

32

Read: Put Address on Bus

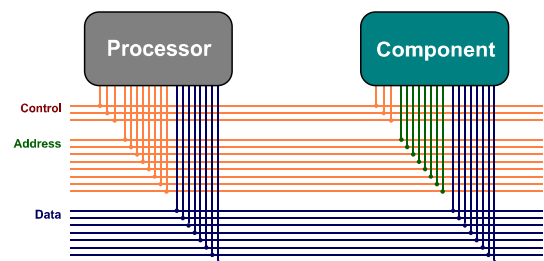


5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

33

Read: Signal Component

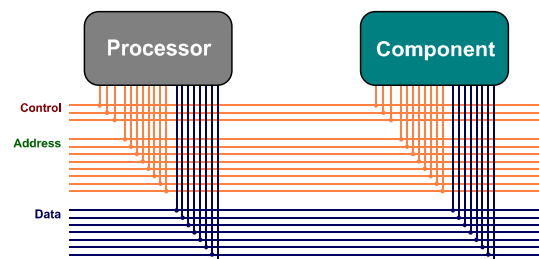


5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

34

Read: Component Gets Address

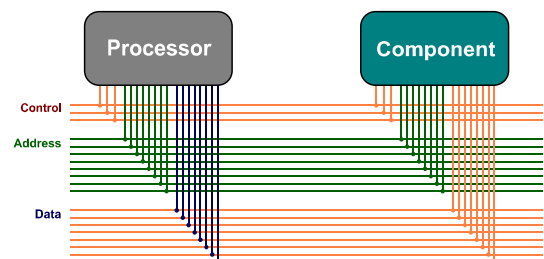


5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

35

Read: Component Returns Data

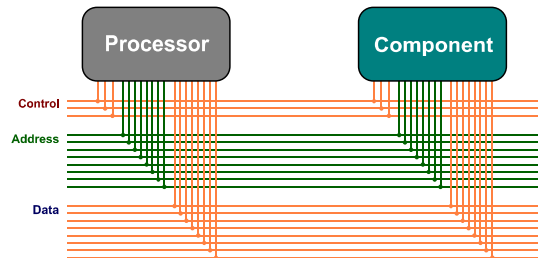


5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

36

Read: Processor Gets Data



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

37

Writing Data

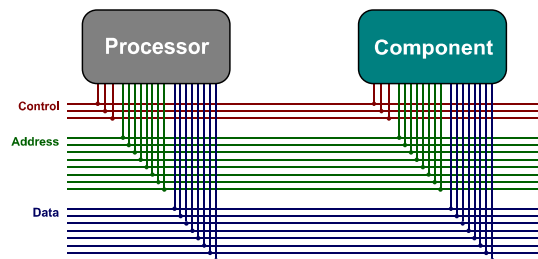
- Processor has the address and data
- Actions:
 1. processor puts the address and data on bus
 2. signals the component to write
 3. component takes the data and then stores it

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

38

Writing Data

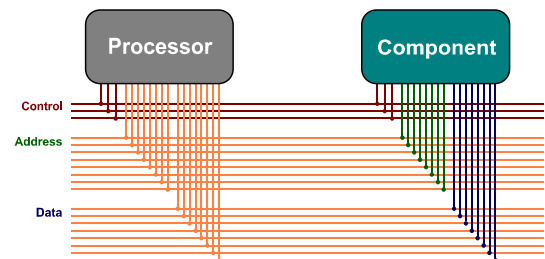


5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

39

Write: Put Address + Data on Bus

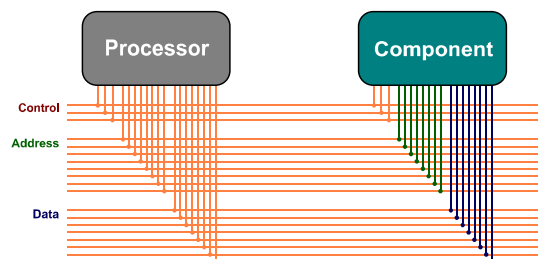


5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

40

Write: Signal Component

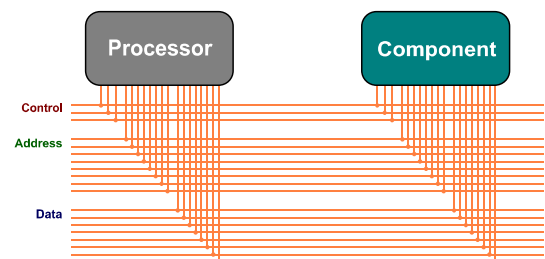


5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

41

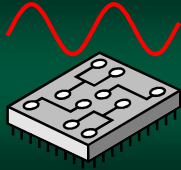
Write: Component Stores Data



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

42

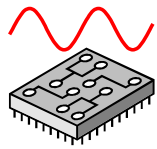


Processor Speed

Let's rock around the clock tonight

The Clock

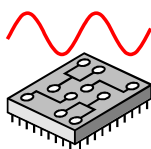
- The rate in which instructions are executed is controlled by the CPU clock
- The faster the clock rate, the faster instructions will be executed
- Measured in Hertz – number of oscillations per second



5/5/2017 Sacramento State - Cook - CSc 35 - Spring 2017 44

The Clock

- Computers are typically (and generically) labeled on the processor clock rate
- In the early 80's it was about 1 Megahertz – million clocks per second
- Now, it is terms of Gigahertz – billion clocks per second



5/5/2017 Sacramento State - Cook - CSc 35 - Spring 2017 45

Clock and Instructions

- Not all instructions are "equal"
- Some require multiple clock cycles to execute
- For example:
 - a simple add can take a single clock
 - but floating-point math could require a dozen
- Some processors can also execute several instructions at a time

5/5/2017 Sacramento State - Cook - CSc 35 - Spring 2017 46




CISC vs. RISC

Highlander: "there only can be one"
(well, not really)

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



5/5/2017 Sacramento State - Cook - CSc 35 - Spring 2017 48

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 is a design philosophy with a large "gray" area between extremes



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

49

CISC

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



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

50

The Semantic Gap

- Pre-1980's focused on reducing the "semantic gap" between languages and the processor
- *So, can we make instructions more like high-level languages?*



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

51

The Semantic Gap

- In high level languages...
 - blocks are common, but there are no "blocks" in assembly
 - if statements are common, but there is no "if" instruction
 - while statements are common, but there is no "while" instruction



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

52

CISC Reasoning

1. Results in better performance
 - each instruction does more
 - reduces the number of instructions required to implement a program
2. Easier to compile high-level languages
 - statements can be mapped directly into instructions
 - compilers will be simpler and result in more consistent machine code

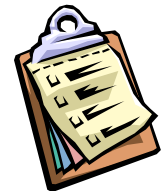
5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

53

CISC Characteristics

- Very few general purpose registers – memory access is emphasized
- Some special-purpose registers
- Instructions can take multiple cycles – depending on how complex



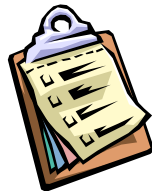
5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

54

CISC Characteristics

- Operands are *generalized*
 - each can access different resources – memory, immediates or registers
 - one typically is the destination
- This allows combinations like:
 - register to register
 - register to memory
 - memory to register



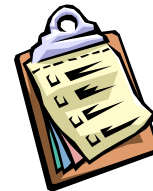
5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

55

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



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

56

CISC Today

- CISC architectures became increasingly complex (some even having case blocks)
- After the 1980's...
 - CISC architectures attempted to have a middle ground between flexibility and complexity
 - dropped instructions that were not used often
 - complex instructions had to justify their implementation (inclusion in the instruction set)

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

57

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



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

58

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...

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

59

Moore's Law and CISC

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



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

60

Memory is the Bottleneck

- CISC can access memory with nearly every instruction
- But, 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



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

61

RISC

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



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

62

RISC

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



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

63

RISC Reasoning

1. Results in higher performance
 - simple instructions can execute at higher clock rates than CISC
 - memory access is limited, ending the bottleneck
2. Easy to compile high-level languages
 - compiler only needs to understand a few instructions
 - compilers can create blocks of instructions fairly robotically

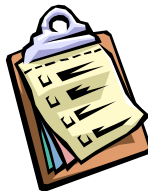
5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

64

RISC Characteristics

- Access to memory is restricted to load/store instructions – that only can be used with a register
- All other instructions only work with registers



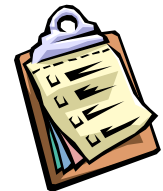
5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

65

RISC Characteristics

- Since registers are used to hold more data, RISC processors typically have many
- Instructions typically take one clock cycle each



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

66

RISC Advantages

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

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

67

Example RISC Processors

- ARM
 - dominant processor used by smartphones - iPhone and Droid
 - designed to reduce transistors
 - which reduces cost, creates less heat, and uses less power



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

68

Example RISC Processors

- IBM PowerPC 601
 - developed in by IBM, Apple, and Motorola (AIM)
 - used by 1990's Macintosh computers

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

69

Addressing: RISC vs. CISC

- There are a large number of possible addressing modes
- RISC tends to limit the number of addressing modes – to about 4 or 5
- CISC tends to have more – sometimes exceeding a dozen or more



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

70

RISC vs. CISC Comparison

CISC	RISC
Emphasis on hardware complexity	Emphasis on software complexity
Operands are generalized	Load/Store instructions
Low number of registers	Higher number of registers
Instructions tend towards multiple clock cycles	Instructions tend towards one per clock cycle

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

71

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

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

72



Instruction Operands

How much data does each need?

Instruction Operand

- The number of operands used in an instruction varies greatly by processor
- More operands give greater functionality, but require more bits to store in memory
- Typically processors contain 1, 2 or 3 operands



Single Operand Processors

- Single operand processors are also known as *accumulators*
- Operates similar to your hand calculator
- The accumulator register
 - used for all mathematical computations
 - other registers simply are used to compare and hold temporary data
- Examples: MOS 6502

Single Operand Instruction

```
# z = 50 - (x + y)

load x
add y      # x + y
store temp

load $50
sub temp   # 50 - temp

store z
```

Two Operand Processors

- Allows two operands to be specified
- For computations, both operands are typically treated as input and one is used to store the result
- Examples:
 - x86 processors
 - PowerPC

Two Operand Instruction

```
# z = 50 - (x + y)

mov x, %R1
add y, %R1    # x + y

mov $50, %R2
sub %R1, %R2   # 50 - R1

mov %R2, z
```

Three Operand Processors

- Allows two input values like before, but also can specify a third output operand
- The third operand can also be used as a index for simple addressing
- Examples:
 - ARM
 - Intel Itanium

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

79

Three Operand Instruction

```
# z = 50 - (x + y)

add x, y, %R1    # x + y
sub $50, %R1, z
```

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

80

Instruction Encoding

How Instructions are Really Stored

Instruction Encoding

- Each instruction on a computer is *encoded* into 1's and 0's
- All information that needs to be stored, has to be converted to bits
- Instructions can either be stored using a variable-length or fixed number of bytes

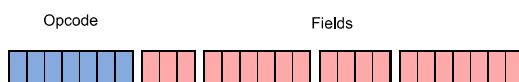
5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

82

Typical Instruction Format

- The *opcode* contains a unique value that indicates the operation to be performed
- It is typically followed by various fields containing register codes, addressing data, etc...



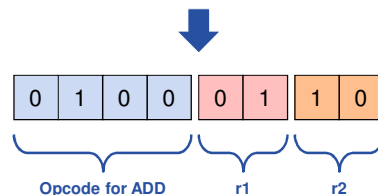
5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

83

Machine Code Example (not x86)

ADD %r1, %r2



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

84

Variable-Length Instructions



- Changes size depending on what features are being used
- Typical in CISC – since operands have multiple features that may or not be present
- Instruction itself must contain data about its length

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

85

Variable-Length Instructions

- Advantages
 - can reduce the space taken up by a program
 - easily extendable – "override" codes can be added to tell the decoder there are additional bytes
- Disadvantages
 - complex decoder circuitry – the different parts of the instruction (e.g. operands) can be in different bit locations – depending on the instruction
 - hardware can't predict the address of the next instruction until the current one is decoded

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

86

Fixed-Length Instructions



- Are always the same size in bytes
- Typical in RISC
 - doesn't have versatile operands
 - and each instruction does less – meaning special features are spread over instructions

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

87

Fixed-Length Instructions

- Advantages
 - simple to decode – faster & less circuitry
 - easily predict the location of the next instruction (assuming that the current is not a jump)
 - easier for the processor to use pipelining
- Disadvantages
 - not easily extendable
 - architects must take this into account and possibly provide some mechanism

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

88



Let's Make a Processor

Appreciate the encoding by making one

Let's Make a Processor

- When a processor is designed, the architects needs to balance features and simplicity
- Generally, architects want to find an encoding that is both compact and simple



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

90

Design Considerations

- A compact encoding:
 - takes as little space, as possible, to store instructions
 - programs require less storage
- A simple encoding:
 - requires little logic to decode
 - minimizes the circuitry on the processor and reduces cost



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

91

Let's Make a Processor

- Assume we are creating a simple processor
- It will have a total of 4 general purpose registers: R0, R1, R2, R3
- The processor will support two operand instructions



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

92

Assigning Bits

- Since there are 4 registers, we need enough bits store a code for each
- $2^2 = 4$, so each register code can be stored with just two bits
- So, to store both operands, we will need:
2 operands \times 2 bits \rightarrow 4 bits

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

93

Our Opcodes

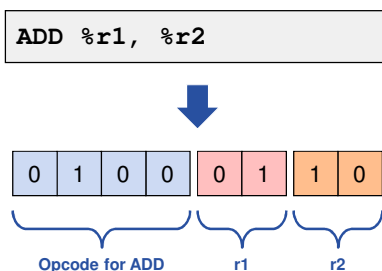
- If we want to store each instruction in a single byte
 - we have $8 - 4 \rightarrow 4$ bits left
 - so, the opcode will be 4 bits
 - this will allow a total of $2^4 \rightarrow 16$ instructions
- Now, we assign opcode bit values for each operation we want the processor to perform

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

94

Sample Encoding



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

95

Limits of this encoding

- With only 4 bits for an opcode, it is limited to 16 distinct processor instructions
- This is simply not enough for all the features that we would like – addressing etc...
- Essentially, the more bits used in the opcode, the more features we can add to the processor

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

96

Two Bytes

- If the instruction set is expanded to 2 bytes, we have far more complexity
- An entire byte can be used for the opcode – giving 256 unique instructions
- The second byte could contain two 4-bit fields – allowing 16 registers!

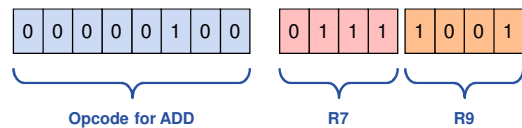
5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

97

Sample Encoding: 2 Byte

ADD %R7, %R9



5/5/2017

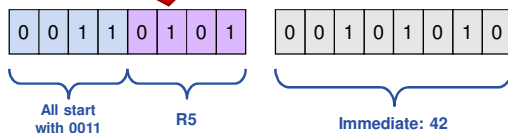
Sacramento State - Cook - CSc 35 - Spring 2017

98

Bits Can Be Used Differently

mov \$42, %R5

Its like we created 16 instructions:
One for each register!



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

99



x86
Instruction
Format

Variable length, but quite structured

x86 Instruction Format

- The x86 instruction set evolved over decades from the original 8086
- Instructions are variable-length (including the opcode)
- These actually helped the x86 to evolve



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

101

x86 Instruction Format

- x86 evolved for 30+ years
 - instructions contain overrides
 - and extensions
- For example, opcode field
 - ran out of unique values
 - added "extension" codes (0Fh)

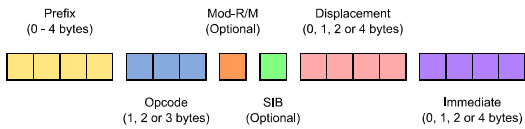


5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

102

x86 Instruction Format



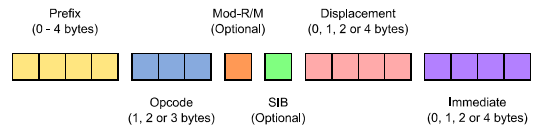
- x86 instructions range from 1 byte up to **17** bytes
- They start with 0 to 4 prefix bytes which contain information how the instruction will be handled

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

103

x86 Instruction Format



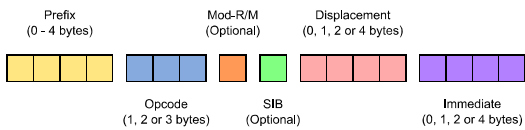
- Opcode contains either one, two or three bytes
- Originally 1 byte until the total opcodes exceeded 255

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

104

x86 Instruction Format



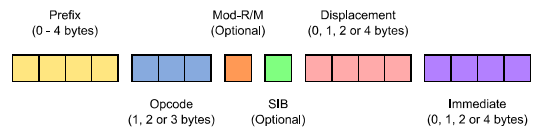
- Mod-R/M and SIB bytes are optional depending the opcode
- These contain all addressing mode information

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

105

x86 Instruction Format



- The instruction ends with an optional Displacement and/or Immediate value
- These are dependent on the opcode **and** Mod-R/M

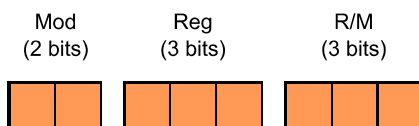
5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

106

Mod-R/M Byte

- Consists of a 2-bit modifier, a 3-bit register code and a 3-bit R/M field
- RM and Mod together define the addressing mode
- **Complex** – due to the extension from 8-bit to 16 to 32



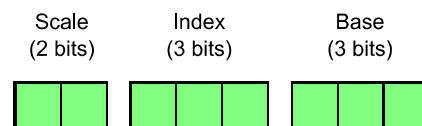
5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

107

SIB Byte

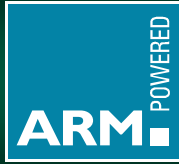
- The Scale-Index-Base byte is fairly straight forward
- Scale byte contains a code for 1, 2, 4 or 8
- Index contains a register number 0..7
- Base contains a register number 0..7



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

108



ARM Instruction Format

What is in your phone
(which better be in your pocket)

ARM Instruction Format

- The ARM Processor was designed for smart phones and other devices that must conserve power
- Reduces transistors...
 - which reduces cost
 - creates less heat
 - uses less power



5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

110

ARM Instruction Format

- Fixed-sized at 32-bit each
- Format is very consistent
- Three operands
- 16 general purpose registers

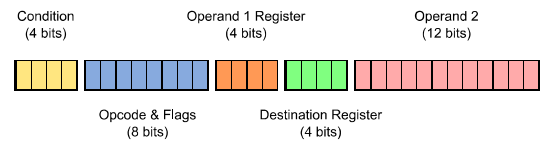


5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

111

ARM Instruction Format



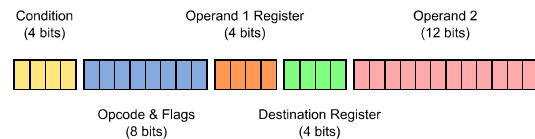
- Instructions start with 4 bit "condition" field we will learn about later
- It allows instructions to be skipped rather than jumped over

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

112

ARM Instruction Format



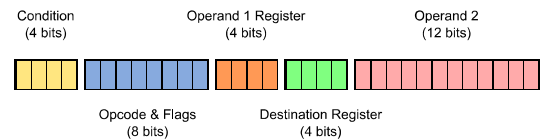
- The opcode field is complex
- Opcodes and flags are mixed together – but, in a very logical way

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

113

ARM Instruction Format



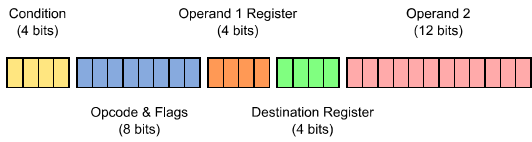
- The first operand is always a register
- The destination of a computation is always a register

5/5/2017

Sacramento State - Cook - CSc 35 - Spring 2017

114

ARM Instruction Format



- The second operand contains various types of data – depending on the opcode
- e.g. literals, registers, addressing, and more...

5/5/2017

Sacramento State - Cook - CSs 35 - Spring 2017

115

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond	0	0	1																												
Cond	0	0	0	0	0	0	A	S		Rd		Rn																			
Cond	0	0	0	0	0	1	U	A	S	RdHi		RdLo		Rn		Rs	1	0	0	1											
Cond	0	0	0	0	1	0	B	0	0		Rn		Rd		0	0	0	0	1	0	0	1									
Cond	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1								
Cond	0	0	0	P	U	0	W	L		Rn		Rd		0	0	0	0	1	S	H	1										
Cond	0	0	0	P	U	1	W	L		Rn		Rd																			
Cond	0	1	1	P	U	B	W	L		Rn		Rd																			
Cond	0	1	1																												
Cond	1	0	0	P	U	S	W	L		Rn																					
Cond	1	0	1	L																											
Cond	1	1	0	P	U	N	W	L		Rn		CRd		CP#																	
Cond	1	1	1	0	CP	Opc				CRn		CRd		CP#		CP	0														
Cond	1	1	1	0	CP	Opc	L			CRn		Rd		CP#		CP	1														
Cond	1	1	1	1																											

5/5/2017

Sacramento State - Cook - CSs 35 - Spring 2017

116