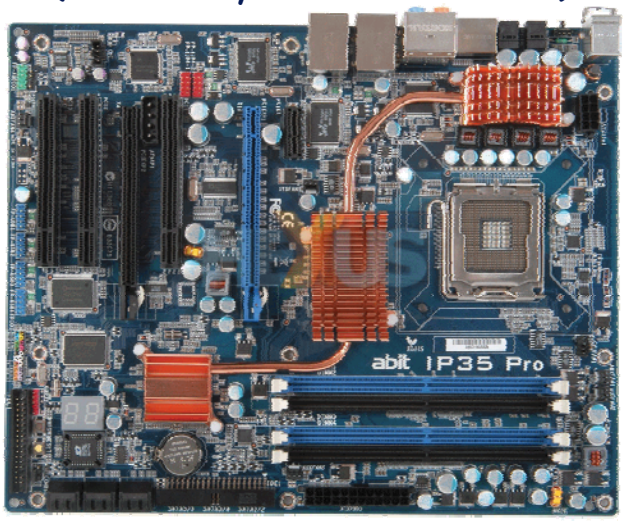
Computer Organization/Design Overview
(KiKi's toy)

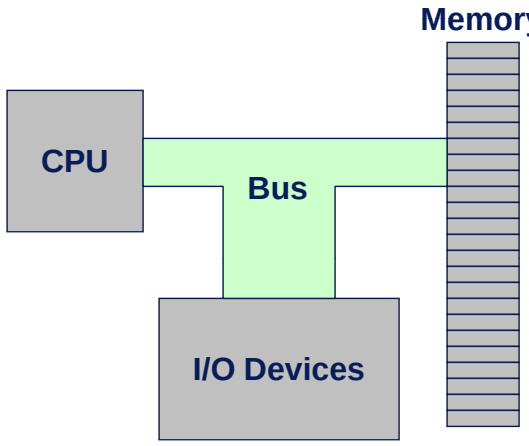
- Anticipated components (when complete):
  - "Da Box"
  - Monitor
  - Keyboard
  - Mouse
  - Joystick
  - Printer
  - Scanner

Computer Organization/Design Overview
(KiKi's toy's motherboard)



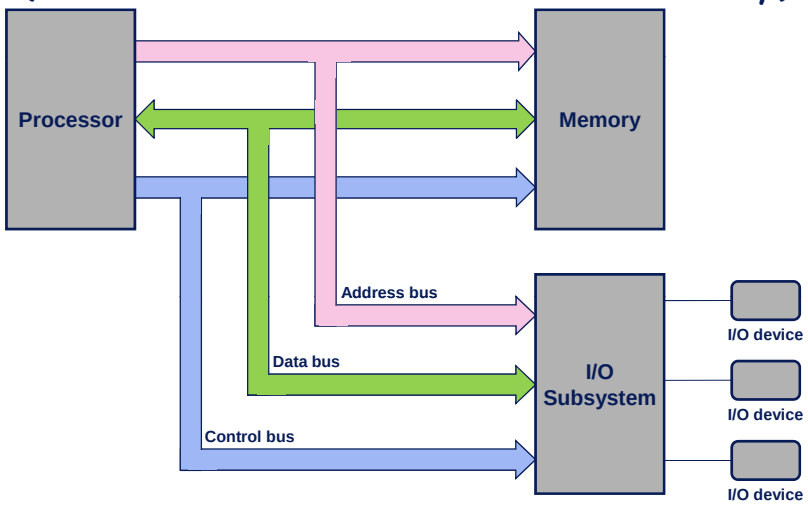
Computer Organization/Design Overview

(an architect's view of KiKi's toy)

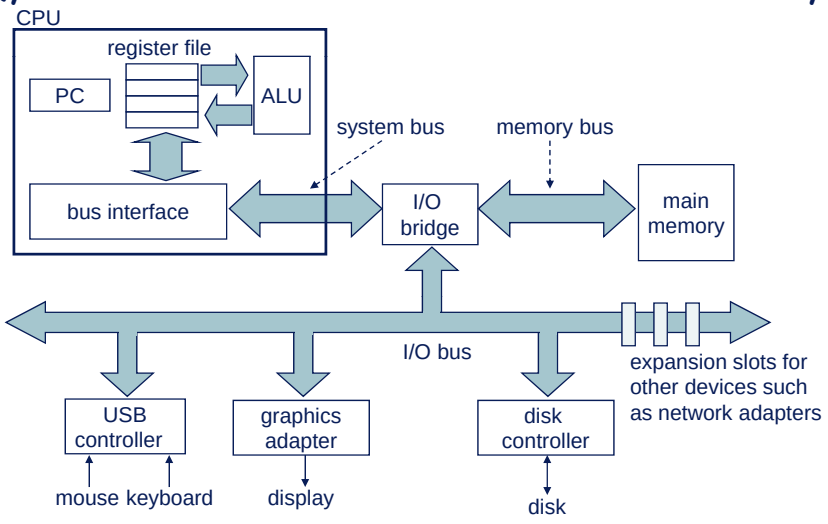


Computer Organization/Design Overview

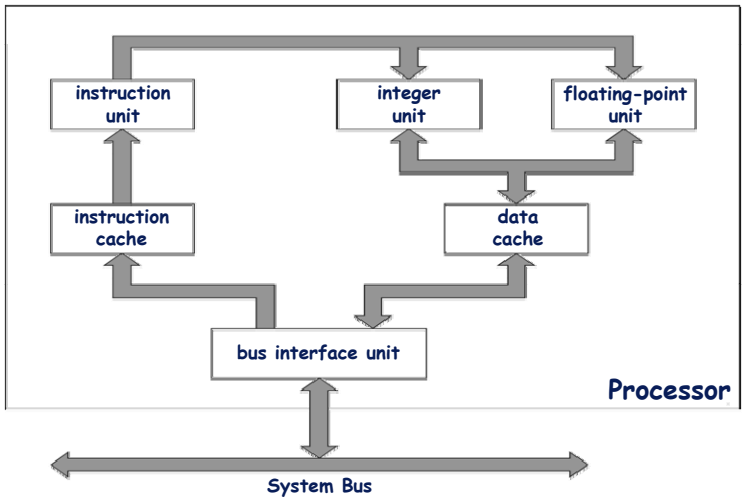
(another architect's view of KiKi's toy)



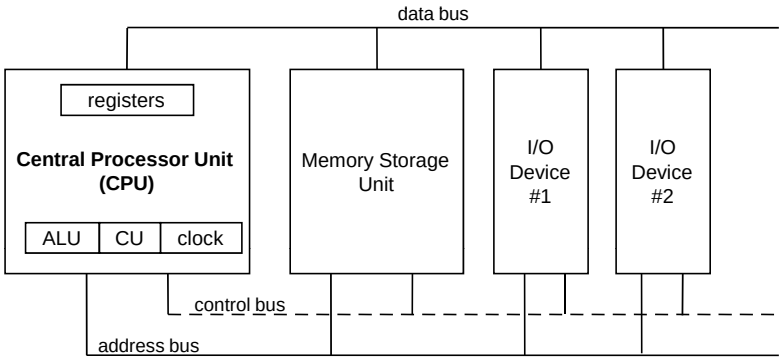
Computer Organization/Design Overview
(yet another architect's view of KiKi's toy)



Computer Organization/Design Overview
(abstraction almost always at work)



Computer Organization/Design Overview
(still yet another architect's view of KiKi's toy)



Computer Organization/Design Overview
(registers)

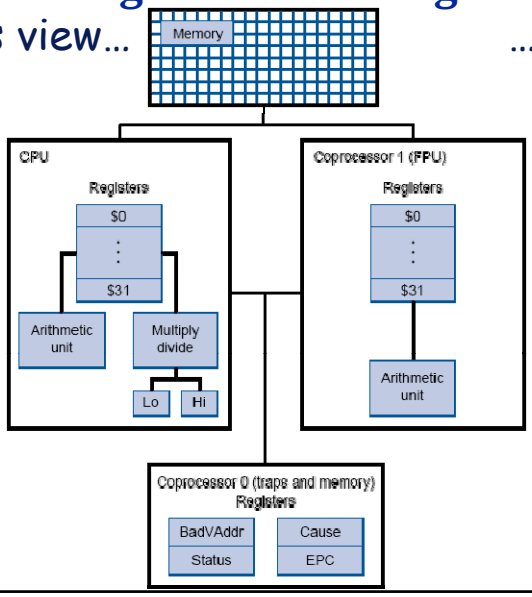
- High-speed memory locations
  - Located on-chip
  - Can be accessed very quickly by processor
  - Processor's "scratchpad"
- Various uses
  - General-purpose (multi-purpose) vs special-purpose
    - ***Register file***: group of similarly-purposed registers
  - Hold data and results of operations
  - Hold addresses that point to locations in memory
  - Hold status information about results of operations
  - ...

Computer Organization/Design Overview
(registers within memory hierarchy)

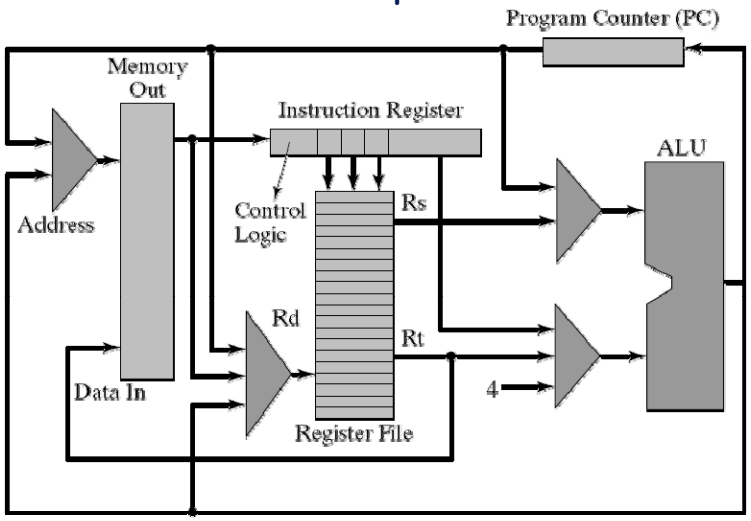
Component	Access Speed (time for data to be returned)	Size of Component
Registers	1 cycle (0.3 ns)	8
L1 Cache	3 cycles (1 ns)	Data: 16K Instruction: 16K
L2 Cache	20 cycles (7 ns)	256K
L3 Cache	40 cycles (13 ns)	4096K
Main Memory (RAM)	300 cycles (100 ns)	16G
Disk	30,000,000 cycles (10 ms)	400G

(numbers given for an actual Intel Pentium 4, 3.2 GHz Server)

Computer Organization/Design Overview
(CS2318's view… …of MIPS)



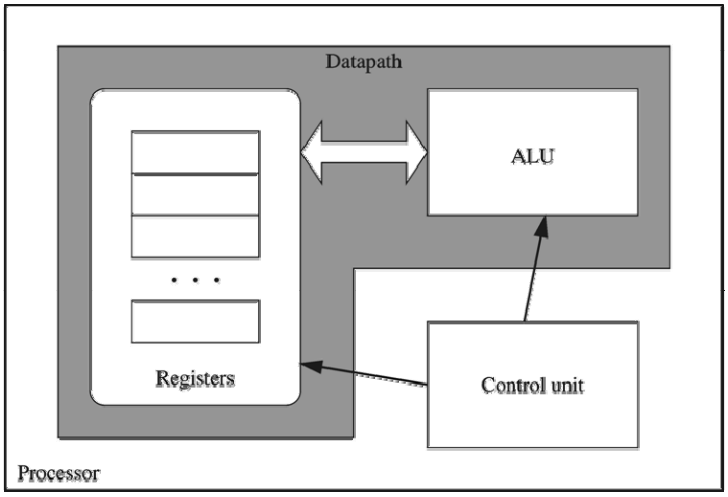
Computer Organization/Design Overview
(textbook author's simplified view of MIPS)



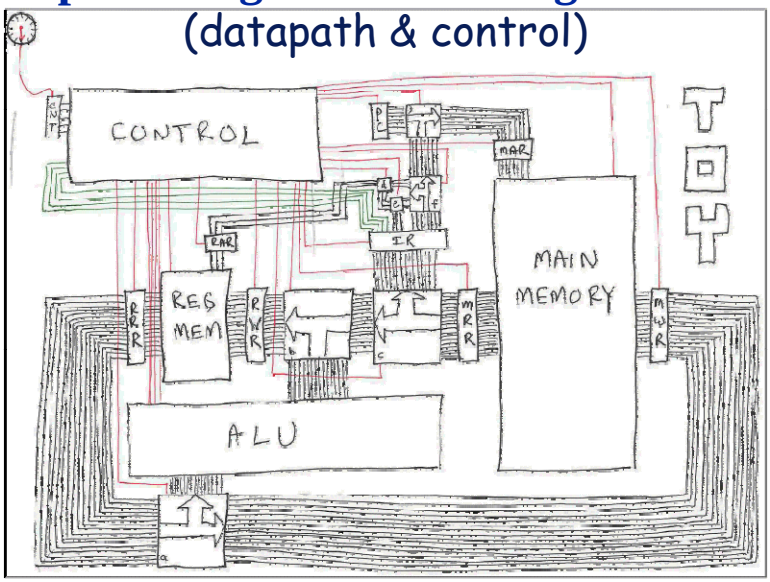
Computer Organization/Design Overview
(datapath & control)

- Datapath
  - Collection of *state elements, computation elements* and *interconnections* that together…
  - …provide *conduit for flow and transformation of data*…
  - …*in processor* during execution
- Control
  - Component of processor that…
  - …commands ("controls") datapath, memory and I/O devices…
  - …according to program instructions

Computer Organization/Design Overview
(datapath & control)



Computer Organization/Design Overview
(datapath & control)



Computer Organization/Design Overview
(instruction & instruction set)

- Instruction:
  - Command to instruct machine to perform certain operation
- Instruction set:
  - Vocabulary of commands (set of all available commands)
- Parts of instruction:
  - Opcode ("verb") – what operation to perform
  - Operand ("noun") – what to operate on
    - Source operands – where values come from
    - Destination operand – where to place result

Computer Organization/Design Overview
(addressing mode)

- Big term referring to *how operands are specified*
  - An operand specifies either (what) data or (which) code
    - Data specification: ***directly*** or ***via address*** (of data in memory)
    - Code specification: always ***via address*** (of code in memory)
    - (an operand may be a data operand or an address operand)
  - At AL level, data may in general exist in 1 of 3 places
    - In a ***register***, in (as part of) an ***instruction***, in ***memory***
  - MIPS supports only *in-register* and *in-instruction* data operands
    - Does not support *in-memory* data operands
      - Special *load/store* instructions for *memory-register* data movements
      - (*load/store* architecture as opposed to *memory-to-memory* architecture)
  - MIPS supports address operands in limited # of ways (modes)
- Next slide → an early glimpse of MIPS addressing modes

meaning data-access or address-calc

Computer Organization/Design Overview
(early glimpse @ MIPS addressing modes)

- Register addressing ← what data or at which address
  - Operand is in a register
- Immediate addressing ← what data
  - Operand is a constant within instruction itself

- Base or displacement addressing ← at which address
  - Operand is at memory location whose *address* is sum of a register and a constant in instruction
- PC-relative addressing ← at which address
  - Branch *address* is sum of PC and a constant in instruction
- Pseudodirect addressing ← at which address
  - Jump *address* is 26 bits of instruction combined with upper bits of PC

Computer Organization/Design Overview
(four processor categories)

- 3-operand ("3-address") machines
  - 2 for source operands, 1 for result
- 2-operand ("2-address") machines
  - 1 of the addresses serves double duty (as source and result)
- 1-operand ("1-address") machines
  - Accumulator machines
  - Accumulator used as 1 source and result
- 0-operand ("0-address") machines
  - Stack machines
  - Operands are taken from stack and result goes back onto stack

Computer Organization/Design Overview
(3-operand machine illustration)

- C++ expression:

```
A = B + C * D – E + F + A
```

- Equivalent instructions:

```
mult    T,C,D   ;T = C*D
add     T,T,B   ;T = B+C*D
sub     T,T,E   ;T = B+C*D-E
add     T,T,F   ;T = B+C*D-E+F
add     A,T,A   ;A = B+C*D-E+F+A
```


Computer Organization/Design Overview
(2-operand machine illustration)

- C++ expression:

```
A = B + C * D – E + F + A
```

- Equivalent instructions:

```
load    T,C   ;T = C
mult    T,D   ;T = C*D
add     T,B   ;T = B+C*D
sub     T,E   ;T = B+C*D-E
add     T,F   ;T = B+C*D-E+F
add     A,T   ;A = B+C*D-E+F+A
```

Computer Organization/Design Overview
(1-operand machine illustration)

- C++ expression:

```
A = B + C * D – E + F + A
```

- Equivalent instructions:

```
load   C   ;load C into accum
mult   D   ;accum = C*D
add    B   ;accum = C*D+B
sub    E   ;accum = B+C*D-E
add    F   ;accum = B+C*D-E+F
add    A   ;accum = B+C*D-E+F+A
store  A   ;store accum contents in A
```


Computer Organization/Design Overview
(0-operand machine illustration)

- C++ expression:

```
A = B + C * D – E + F + A
```

- Equivalent instructions:

```
push   E          sub
push   C          push   F
push   D          add
mult              push   A
push   B          add
add               pop    A
```

Computer Organization/Design Overview
(implications of # of operands)

- With fewer operands
  - Instructions more primitive → requires less complex processor
  - Instructions shorter in length
  - More instructions per program
    - More instructions needed to perform a given task
  - Longer execution times (generally)
- What do you think is…
  - Major disadvantage of 1-operand machine (instruction set)
    - ***Hint:*** Accessing memory is slow (compared to accessing register)

Computer Organization/Design Overview
(load-store architecture) (e.g.: MIPS)

- Only *load/store* instructions have to access **memory**
  - All other instructions access things in **register** &/or **instruction**
  - *Load/store* instructions enable *memory-register* data movements
  - Low clock cycles for instructions (except load/store instructions)
  - Reduces instruction length (*how?*)
- Sample MIPS instructions

pseudocode description

```
lw $t1, 8($t0)      # $t1 = Mem[$t0 + 8]
sw $t1, 0($t0)      # Mem[$t0 + 0] = $t1
add $t2, $t1, $t0   # $t2 = $t1 + $t0
sub $t2, $t1, $t0   # $t2 = $t1 - $t0
```

NOTE: **`$t0`**, **`$t1`**, **`$t2`** are *registers*, **`lw`** = *load* word, **`sw`** = *store* word

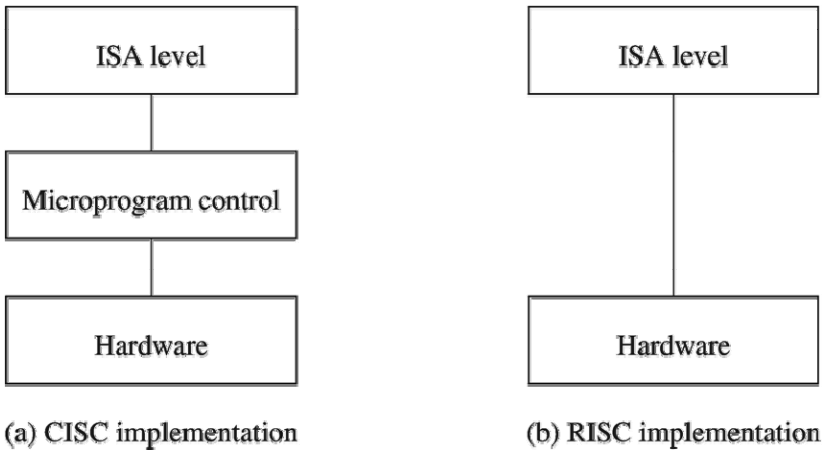
Computer Organization/Design Overview (RISC vs CISC)

- Reduced Instruction Set Computer
  - Uses simple instructions
  - Operands assumed in registers
    - Not in memory
    - Simplifies design (e.g.: fixed instruction size)
- Complex Instruction Set Computer
  - Uses complex instructions
  - Operands can be in registers or memory
    - Instruction size varies
  - Typically uses microprogam

Computer Organization/Design Overview (RISC design principles)

- Simple operations
  - Simple instructions that can execute in 1 cycle
    - Operations can be hardwired (no need for microprogram control)
- Register-to-register operations
  - Only load and store instructions access memory
- Simple addressing modes
  - Few addressing modes
- Large number of registers
  - Needed to support register-to-register operations
  - Minimize procedure call and return overhead

Computer Organization/Design Overview
(RISC vs CISC)



(a) CISC implementation (b) RISC implementation

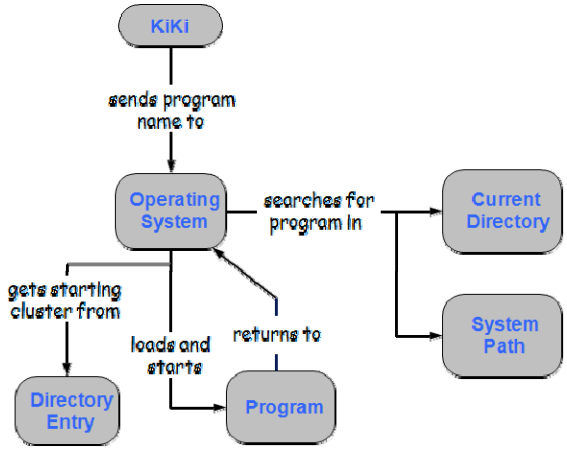
Computer Organization/Design Overview
(RISC vs CISC)

Aspect	CISC		RISC
	VAX 11/780	Intel 486	MIPS R4000
# of instructions	303	235	94
# of addressing modes	22	11	1
size of instruction (bytes)	2-57	1-12	4
# of general purpose registers	16	8	32

Computer Organization/Design Overview
(oversimplified historical perspective)

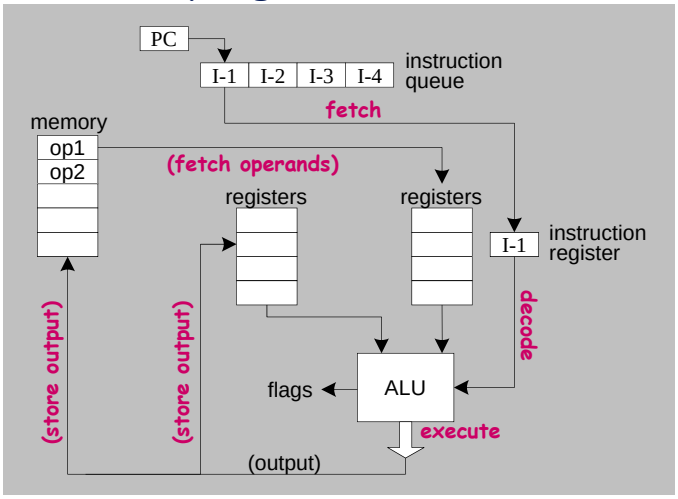
- Early generations
  - Difference engine of Charles Babbage
- Vacuum tube generation
  - Around 1940's and 1950's
- Transistor generation
  - Around 1950's and 1960's
- IC generation
  - Around 1960's and 1970's
- VLSI generation
  - Since 1970's

Computer Organization/Design Overview
(running KiKi's favorite program)

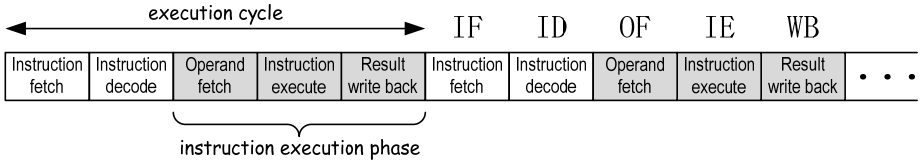


Computer Organization/Design Overview
(KiKi's favorite program on the run)

- **Fetch**
- **Decode**
- (Fetch operands)
- **Execute**
- (Store output)



Computer Organization/Design Overview
(KiKi's favorite program keeps on running)



Computer Organization/Design Overview
(KiKi's grandpa still praises his 486)

Stages

Cycles	S1	S2	S3	S4	S5	S6
1	I-1					
2		I-1				
3			I-1			
4				I-1		
5					I-1	
6						I-1
7	I-2					
8		I-2				
9			I-2			
10				I-2		
11					I-2	
12						I-2

(non-pipelined)

Stages

Cycles	S1	S2	S3	S4	S5	S6
1	I-1					
2	I-2	I-1				
3		I-2	I-1			
4			I-2	I-1		
5				I-2	I-1	
6					I-2	I-1
7						I-2

(pipelined)

(instructions can be executed in parallel)

Computer Organization/Design Overview
(GiGi, KiKi's best friend)