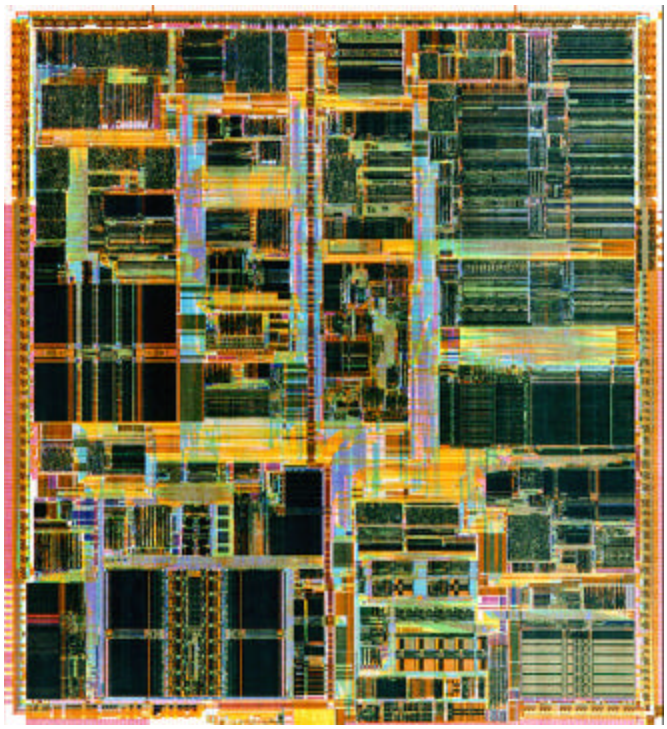
Inside the CPU

- **how does the CPU work?**
  - what operations can it perform?
  - how does it perform them? on what kind of data?
  - where are instructions and data stored?
- **a toy machine to illustrate the basics**
  - a simulator program for the toy machine so we can run programs for the toy machine

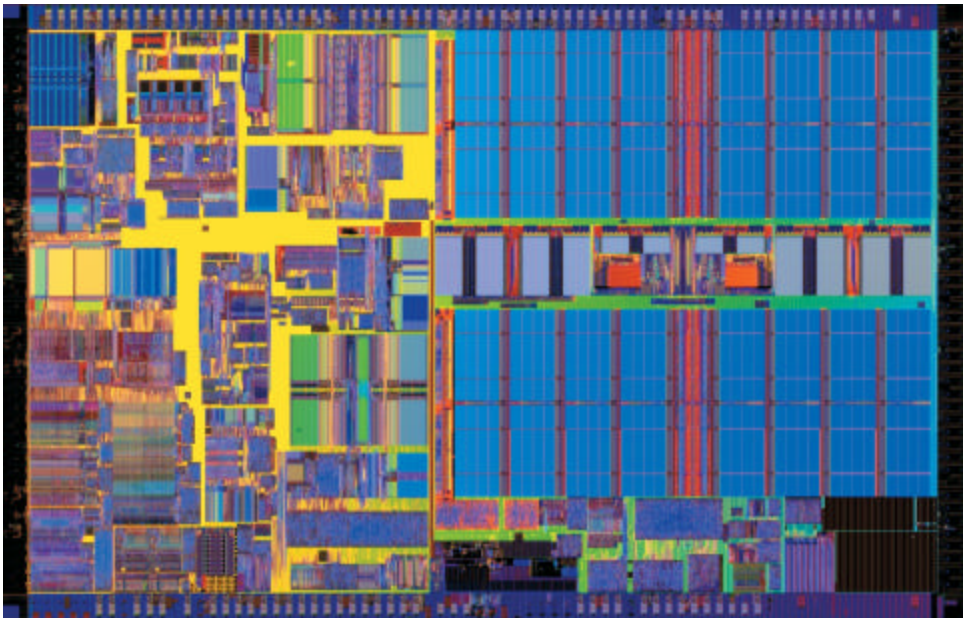
- **computer architecture: real machines**
- **caching**
  - making things seem faster than they are

- **how chips are made**
- **Moore's law**

Pentium II

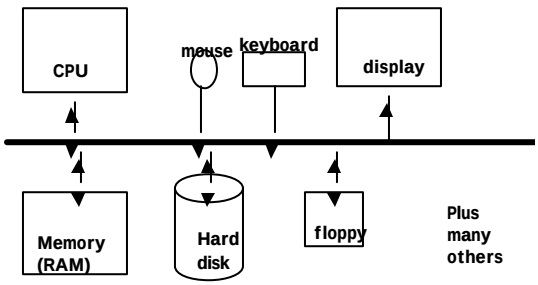


Pentium III



Block diagram of computer

- **CPU can perform a small set of basic operations**
  - arithmetic: add, subtract, multiply, divide, …
  - memory access: fetch data from memory, store results back in memory
  - decision making: compare numbers, letters, …, and decide what to do next according to result
  - control the rest of the machine
- **operates by performing sequences of very simple operations *very* fast**



A really dumb computer

- **design:**
  - a shelf that holds a set of 3x5 cards numbered 1, 2, ...
  - each card is big enough to hold one number or one instruction
  - one card is set aside as a scratchpad
- **instructions include:**
  - GET a number from keyboard and put it on the scratchpad
  - PRINT the number that's on the scratchpad (scratchpad doesn't change)
  - STORE the number that's on the scratchpad on a specific card (scratchpad doesn't change)
  - LOAD the number that's on a specific card onto the scratchpad (specified card doesn't change)
  - ADD the number on a specified card to the scratchpad value, and put the result on the scratchpad (specified card doesn't change)
  - STOP: don't do anything more
- **programming: write instructions on cards**
- **execution: start with the first card, do what it says, then go to the next card**

A program to add any two numbers

```
    GET          get first number from keyboard into scratchpad
    STORE  N     save it on card named "N"
    GET          get second number from keyboard into scratchpad
    ADD  N       add value from card N (first number) to scratchpad
    PRINT        print the result (from scratchpad)
    STOP
N   blank card
```

- **how would you extend this to adding three numbers?**
- **how would you extend this to adding 1000 numbers?**

Looping and testing and branching

- **we need a way to re-use instructions**
- **add a new instruction to CPU's repertoire:**
  - GOTO start with instruction on a specified card, instead of using next card
- **this lets us repeat a sequence of instructions indefinitely**

- **how do we stop the repetition?**
- **add another new instruction:**
  - IFZERO if scratchpad value is zero, go to specified card instead of using next card

- **these two instructions let us write programs that repeat instructions until a specified condition becomes true**
- **the CPU can change the course of a computation according to the results of previous computations**

Add a lot of numbers

```
start   GET             get a number
        IFZERO print    if number was zero, go to "print"
        ADD SUM         add SUM so far to new number
        STORE SUM       store it back in SUM so far
        GOTO start      go back to "start" to get another number
print   LOAD SUM        load sum onto scratchpad
        PRINT
        STOP
SUM     card with 0     has to be 0 to start
```

How does this relate to a real computer?

- **the "shelf" corresponds to RAM**
  - each card corresponds to one location in memory
  - memory locations are numbered 1, 2, …
- **each memory location holds an instruction or a data value**
- **instructions are encoded numerically (so they look the same as data)**
  - e.g., GET = 1, PRINT = 2, LOAD = 3, STORE = 4, …
- **we can't tell whether a specific memory location holds an instruction or a data value (except by context)**
  - everything looks like numbers

- **CPU operates by a simple cycle**
  - FETCH: get the next instruction from memory
  - DECODE: figure out what it does
  - EXECUTE: do the operation
    move operands to and from memory, scratchpad, etc.
  - go back to fetch

A simulator for the toy computer

- **simulator (a program) reads a program for the dumb computer**
  - converts instructions and names into numbers
  - simulates what the dumb computer would do
- **instruction repertoire:**

01	**`get`**	**read a number from the keyboard into scratchpad**
02	**`print`**	**print contents of scratchpad**
03	**`load M`**	**load scratchpad with contents of memory location M**
04	**`store M`**	**store contents of scratchpad into memory location M**
05	**`add M`**	**add contents of location M to scratchpad**
06	**`sub M`**	**subtract contents of location M from scratchpad**
07	**`goto M`**	**go to location M unconditionally**
08	**`ifpos M`**	**go to location M if scratchpad is positive**
09	**`ifzero M`**	**go to location M if scratchpad is zero**
10	**`stop`**	**stop execution**
	`init V`	**initialize this memory location to value V (once, before simulation starts)**

Program:

- **read two numbers, print sum:**

```
     get             gets the first number into scratchpad
     store num       store it in a memory location
     get             get the second number into scratchpad
     add num         add previously-read and -stored number
     print           print the sum from the scratchpad
     stop
num init  0          initial value for sum
```

Branching and looping

- **read numbers until one of them is zero, then print sum:**

```
# print sum of input numbers (terminated by zero)

start get             # read a number
      ifzero   done   # no more input if number is zero
      add      sum    # add in accumulated sum
      store    sum    # store new value back in sum
      goto     start  # go back and read another number

done  load     sum    # print sum
      print
      stop

sum   init     0
```