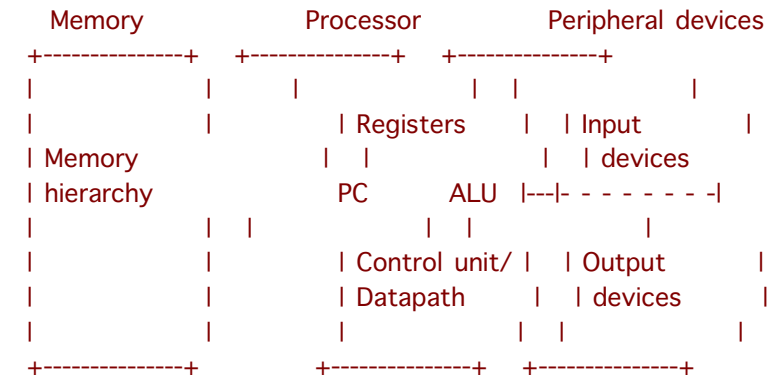
inside processor



less physical (more abstract),imagine point-to-point interconnect on left and (ultimately) shared I/O bus on the right.Look carefully,4block.

1)processor (CPU) contains

a)ALU : performs arithmetic and logical operations
b)file of registers whose main function is to serve as high-speed storage of operands,
c)control unit/datapath(two are inseparable) that interprets instructions and causes them to be executed
d)program counter(PC) lives inside datapath and indicates address of next instruction to be fetched.
Datapath executes instructions, while control unit specifies how this work should be carried out.

control unit abstract"program" that has been burned into silicon chip;call hard-wired.current soln. Previously,control unit was actual (software) program that ran on smaller implementing processor inside "real" processor, i.e.,one visible to compiler,assembler,programmer. old term "microprogramming".

Note: Modern terminology would distinguish

i) pipeline,
ii) register
iii) individual functional units
iv) on-chip caches.

2)**memory**: stores instructions, data,intermediate and final results.Memory is now implemented as a hierarchy. Memory is typically byte addressed.

3) **A set of peripheral input devices**: send (transmit) data and instructions sometimes from themselves, sometimes from outside world to memory. Disk (storage device) functions as an input device, and so do I/O perpherals such as the keyboard and the network-interface chip.

4) **A set of peripheral output devices:** receive (transmit) final results and messages from memory---sometimes to themselves, sometimes to outside world. Again,disk functions as an output device, but so do I/O peripherals such as the monitor ("screen") and the printer.

Recall the fetch-execute cycle in more detail.

1) fetch engine ("f-box") in datapath fetches an instruction from memory as specified by PC.

2) PC is unconditionally incremented by length of an instruction in bytes,assuming,memory is byte addressed. Unless otherwise specified,consider all instructions to be encoded in 32 bits. If fetched instruction is a conditional or an unconditional branch, further modification of PC may take place, by some other box, at a later time.

3)Decode engine ("d-box") in datapath decodes instruction, guided by control unit. This is first time we know what kind of instruction we have fetched. (We will beef up the d-box later).

4)For register-register instruction,execution engine ("x-box") in datapath executes instruction's operation. But this is one of several types of instruction. For example, it could equally well be a load instruction transferring data from a memory location to a register,store instruction transferring data from a register to a memory location, or conditional-branch instruction. So, we need to add at least a load-store engine("m-box") to the datapath. Actually, we need to add one more. We will list all datapath engines later. (called pipeline boxes).

5) And repeat until shutdown (or whatever).

Note: The details about exact sequencing of actions will make sense much later when we consider instruction pipelining by datapath.

2016,this picture is somewhat dated.Ex:
we don't have ALUs any more;replaced by sets of functional units.
Portions of thememory hierarchy(L1$,L2$,L3$) integrated into processor chip.(processor caches).
multicore means several processors(now called cores),each with its private low-level cache, have been integrated onto a single "processor" chip.
Finally, although this is more advanced, sometimes we deviate from strict program order deep inside datapath. This deviation is hidden from running program.This advanced technique is called dynamic instruction scheduling and conceptually makes use of dataflow ideas, thus temporarily breaking away from the von Neumann model of pure control-flow scheduling.

processor performance.

Historically, 2 factors with greatest impact on performance have been

i) increases in clock frequency
ii) increases in number of transistors(hence, gates) that can be packed onto a single chip. Peak performance has been strongly affected by product "number of logic transistors-Hz".

How have these two factors evolved over the years?

Moore's Law

>*From 1971 to 2002, clock speeds increased at an exponential rate (roughly* doubling every 2.5 years). After 2003,frequencies stabilize in 3-GHz range(otherwise chips would burn up:unless cooled them with Freon).

But other factor is still going strong. The number of transistors that can be put on a chip rose at the same pace (or better) as the clock frequency over this period, but without any leveling off in 2003.
Moore's law has had several incarnations. Initially, it was exponential increase in number of memory transistors per square centimeter, and per dollar (Moore's memory law). Later, it was about a similar exponential increase in number of logic transistors per square centimeter, and per dollar (Moore's processor law). for a while,it was about combined exponential increase in both number of logic transistors and clock frequency.
Number of logic transistors-Hz isn't everything, but it's damn important for performance.

In 2016, Moore's law has reverted back to mean an exponential increase in number of logic transistors per square centimeter, and hence in the number of logic transistors per processor chip. How long can Moore's law continue?That's one of the $64,000 questions.

there might be one Moore's law for memory,another Moore's law for processors. But why does clock frequency enter picture and then suddenly depart? Logic transistors were, and still are, getting smaller. For a time, there was a concurrent phenomenon: _Dennard scaling_.
When Dennard scaling held,following was true: as transistors got smaller,power density was constant---so if there was a reduction in a transistor's linear size by two,power it used fell by four (with voltage and current both falling by two).As a result,total chip power for a given chip-area size stayed same from one VLSI-process generation to the next. At the same time, feature sizes shrank, transistor count doubled,and the clock frequency increased by 40% every two years.However, when feature sizes dropped below 65 nm, Dennard scaling could no longer be sustained,bcz of exponential growth of the leakage current. In short, although today we still have an exponentially increasing number of logic transistors per processor chip, we can no longer afford power to turn them all on,& no longer tolerate heat of clocking them faster.

Again, we have billions and billions of transistors, but can no longer afford power to turn them all on.
Multicore is a different path to steadily increasing performance that deliberately underclocks cores to stay within power budget.
Multicore has the potential to restore power efficiency,which is number of operations/second per Watt.

Broader significance for future of computing is this.It used to be that sequential processors had steadily increasing performance; that let programmers carry on with business as usual.Now it is case that only parallel processors will steadily increasing performance;programmers will have to get off their asses and learn to program the new machines.

I personally find it unlikely that programming systems, such as Google's Map-Reduce, will allow parallel-oblivious programmers to survive in the new era. Using a tool to program for you is not exactly a comprehensive solution to problem of enabling general-purpose parallel programming.

Speaking as a computer architect, it appears that memory is the critical bottleneck for multicore: until we have major increases in memory bandwidth,multicore growth will be stunted. Indeed, even highest-level shared cache is a bottleneck, because of unrelenting contention for it among cores on chip.
Note: GPUs need high memory bandwidth; CPUs need low memory latency. As a general rule, when too many threads share the same cache, the performance crashes.

Again, Intel and other manufacturers capped their clock frequencies at their2003 level.IBM has been a bit more daring.

Amdahl's Law

famous law : how efforts to improve performance (or reduce power or whatever) sometimes lead to disappointments,or at least to suprises. This is Amdahl's law.
Consider a program with one portion that is perfectly sequential, another perfectly parallel portion that can be made as parallel as we like.Suppose sequential portion accounts for 5% of run time when the program is run sequentially. What happens if program is run on a 1000-core processor?

We draw little diagrams. If you have any brains, you will avoid formulas.

```
  5 /1 +   95 /1000 =  100
  5  +  0.1  =  5.1   su = 19.6 (= 100/5.1)
speedup isn't 1,000; it's barely 20.
```

In general terms, Amdahl's law says that optimizing one part of the system that contributes the fraction 'p' of the quantity being minimized can yield at best_ an improvement of 1/(1 - p). Example: When p = 0.95, the best result is a factor of 20. This works for time, power, etc.

Example: Gene Amdahl observed that the less parallel portion of a program can limit performance on a parallel computer. Thus, one might consider reserving part of a multicore chip for a larger core specialized in single-thread performance.

Suppose we ignore this suggestion and build 100 identical cores. Program P has portion A that uses 10% of the sequential time, and gets no parallel speed up, and portion B that uses 90% of the sequential time, and gets a parallel speed up equal to the number of cores. What is the run time of P on this parallel computer?

```
10  +  90  =  100
/1   /100
--   ----     ---

10 +  0.9  = 10.9     su = 9.2
```

Now, let us cannibalize the resources required to build 10 cores to build one larger core that runs single threads twice as fast, leaving 90 cores of the original design. Program P is the same, with its 10%/90% split. What is the run time of P on this new parallel computer?

```
10  +  90  =  100
/2    /90
--    ---     ---

 5  +   1  =    6     su = 16.7
```

Example: In a 100-Watt sequential circuit,combinational logic dissipates 20 Watts, while the (clocked) state elements dissipate 80 Watts.In combinational logic, power is proportional to voltage, but in state elements, power is proportional to square of clock frequency.

The designers propose to reduce the clock frequency by a factor of 10, and the voltage by a factor of 8. After the change, how much power is dissipated by the circuit?

```
20  +  80  =  100 Watts
/8   /100
--   ----     ---

2.5  + 0.8  =  3.3 Watts
```

New plan. Starting from the original circuit, the designers now propose to reduce the clock frequency by a factor of 15, and the voltage by a factor of 5. After the change, how much power is dissipated by the circuit?

```
20  +   80  =  100 Watts
/5    /225
--    ----     ---

 4  + 0.36  = 4.36 Watts
```

Exercise: On a uniprocessor, perfectly serial portion A of program P consumes 25 s, while perfectly parallel portion B consumes 75 s, for total uniprocessor run time of 100 s. On 1,000-P multiprocessor, however, program P's run time falls to 25 + 0.075 = 25.075 s. How many processors are required to achieve at least 75% of the 1,000-P speedup?

Ans: 75% of speedup translates into a contribution of 4/3 * 25.075= 33.433 - 25 = 8.433 s from portion B. 8.9 processors is enough for this (8.8932806), but that's crazy. Nine processors gives us 8 1/3 sas B's contribution, which is less than 8.433 s, so nine processors is the right answer.

At present,I don't plan to cover in any detail two low-level implementation topics.
At lowest level, there is circuit design.key words:"wires", "transistors", "resistors", "diodes", and so on; this is domain of electrical engineers. The next level up is logic design,at which logic gates and wires are put together to build combinational circuits such as ALUs and PLAs, and at which stable-storage primitives such as flip-flops and latches are combined to implement registers and hard-wired control. We will do some logic design.

Signed and Unsigned Numbers

Computers need to represent numbers.Since computers are electrical machines,it is natural to represent numbers in hardware as a series of high and low electronic signals. This gives us only two digits, 0 and 1, so these numbers are called binary numbers. A single binary digit is a bit.

We will work mostly in base 2 and base 16, but numbers may be represented in any base. If 'd' is value of the ith digit,contribution to number from that digit position is d*base^i, where 'i' starts at 0 and increases from right to left.

Mathematicians have a set of numbers they call natural numbers(nonnegative integers including 0). The use of binary bit patterns to represent natural numbers follows what is called natural-number semantics. (Computer jargon for natural number is "unsigned number").

Example: In natural-number semantics, the bit pattern 1011 represents
(1 * 2^3) + (0 * 2^2) + (1 * 2^1) + (1 * 2^0) = 11.

The bit corresponding to exponent 0 is the least significant bit. bit corresponding to largest exponent is most significant bit.

Suppose our registers are 32-bits long.Now, there are 2^32 distinct 32-bit bit patterns.If we interpret registers using natural-number semantics, these 2^32 bit patterns will represent natural numbers from 0 to 2^32 - 1.

Example: In a 3-bit register,using natural-number semantics, we have following interpretations:
000 = 0, 001 = 1, 010 = 2, 011 = 3, 100 = 4,101 = 5, 110 = 6, and 111 = 7.

Hardware can be designed to add,subtract,multiply,and divide numbers represented by these bit patterns.

How do we represent integers,which may be negative?After a bit of confusion,sensible people decided that integers ("signed numbers" in the jargon) should be represented using two's complement semantics. Basically, leading 0s mean positive, while leading 1s mean negative. A slightly larger example is necessary.

Example: In a 4-bit register, using two's complement semantics, we have following interpretations:
0000 = 0, 0001 = 1, 0010 = 2, 0011 = 3, 0100 = 4,0101 = 5, 0110 = 6, 0111 = 7,

1000 = -8, 1001 = -7, 1010 = -6, 1011 = -5,1100 = -4, 1101 = -3, 1110 = -2, and 1111 = -1.

Since 0 occupies a position as a positive number,we wind up with one more nonzero negative number than nonzero positive number.Notice:look at the most significant bit to decide if a number is positive or negative. This bit is called the sign bit.

Addition is addition. Let's add a few 4-bit numbers in 4-bit registers and ignore _carry out_. Then, using two's-complement semantics, we will determine whether we got the right answer for that semantics in the 4-bit register.

```
Example:  1100 = -4        Example:  0100 =  4
         +1100 = -4                  +0100 =  4
          ----                      ----
        1|1000 = -8                 1000 = -8

Example:  1011 = -5        Example:  0101 =  5
         +1011 = -5                  +0101 =  5
          ----                      ----
        1|0110 =  6                 1010 = -6
```

If we discard most negative integer, we observe that every (two's complement) integer fits into a 4-bit register if and only if its negation fits into a 4-bit register. There is nothing special about 4.
The same assertion holds for (two's complement) integers and n-bit registers.

BTW,positive(two's complement) integer 'x' fits into an n-bit register if and only if x <= 2^(n-1) - 1.

We say that overflow has occurred if the register is too small to contain the correct result, including the sign bit. Note that a carry out of the register is not in itself a sign of overflow. (See first example).

Trick number 1: How to negate a two's complement number.

```
1) Flip every bit.      0101 =  5    0001 =  1      1000 = -8
                        1010         1110           0111
2) Add one.             1011 = -5    1111 = -1      1000 = -8
```

not every two's complement number can be negated.Why?there is no 4-bit +8.correct answer must fit in the register.

Trick number 2: How to place an n-bit two's complement number in a register with more than 'n' bits.

1) Copy the number on the right. Example: 1000 = -8 (4 bits)

2) Replicate the sign bit on the left. 1111 1000 = -8 (8 bits)

Now, let's learn how to write bit patterns in hexadecimal. This is just shorthand; it is not new semantics.

0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	8	-8
9	1001	9	-7
a	1010	10	-6
b	1011	11	-5
c	1100	12	-4
d	1101	13	-3
e	1110	14	-2
f	1111	15	-1

We can convert "hex" patterns into bit patterns, and bit patterns into "hex" paterns.

If the length of the bit pattern is not a multiple of 4, go from right to left.

A hex digit corresponds to 4 bits.

In this table,columns are: i) hex digit, ii) bit pattern, iii) natural-number semantics,iv) two's complement semantics.