Chapter 16 -- Architectures

```
PERSPECTIVE ON ARCHITECTURE DESIGN
----------------------------------

factors in computer design:
   speed
       as fast as possible, of course
       dependent on technology and cost
   cost/price
       profit, non-profit, mass market, single use
   useablility
       shared/single user, size of machine, OS/software issues,
       power requirements
       depends on intended use!
   intended market
       mass market, scientific research, home use, multiple users,
       instructional, application specific
   technology
```

```
price/performance curve

       ^  |
 perf. |  |         x
          |           x          Want to be to the "left"  of these,
          |      x               to have higher performance for the
          |                      price.  "More bang for the buck."
          |    x   x
           ------------------

                    price ->
```

```
technology -- a perspective
----------

    electromechanical (1930) -- used mechanical relays

    vacuum tubes (1945)
      space requirement:  room
          Manchester Mark 1 (late 1940s)

    transistors
      discrete (late 1950s)
         space requirement:  a large cabinet to a room
         Examples:
            CDC 6600
            B 5000
            Atlas (?)
```

```
            PDP 11/10

      SSI,    MSI (mid-late 1960s)
      1-10   10-100 transistors
         space requirement:  a cabinet
         Examples:
           Cray 1
           VAX 11/780

      LSI   (early 1970s)
      100-10,000 transistors
         space requirement:  a board
         Examples:

      VLSI  (late 1970s - today)
      >10,000 transistors
         space requirement:  a chip, or chip set, board
         Examples:
            MIPS R2000
            Intel 386 (~275,000 transistors)
            Pentium (~4 million transistors?)
            Sparc
            PowerPC (millions of transistors)
```

RISC vs. CISC

RISC - Reduced Instruction Set Computer

The term was first used to name a research architecture at Berkeley: the RISC microprocessor. It has come to (loosely) mean a single chip processor that has the following qualities:

1. load/store architecture
2. very few addressing modes
3. simple instructions
4. pipelined implementation
5. small instruction set -- easily decoded instructions
6. fixed-size instructions

CISC - Complex Instruction Set Computer

This term was coined to distinguish computers that were not RISC. It generally is applied to computers that have the following qualities:

1. complex instructions
2. large instruction set
3. many addressing modes

difficulties with these terms

- not precisely defined
- term introduced/applied to earlier machines
- "RISC" has become a marketing tool

single chip constraint

As technologies advanced, it became possible to put a processor on a single VLSI chip. Designs became driven by how much (how many transistors) could go on the 1 chip.

Why? 1. The time it takes for an electrical signal to cross a chip are significantly less than the time for the signal to get driven off the chip to somewhere else.
2. The number of pins available was limited.

So, the desire is to have as little interaction of the chip with the outside world as possible. It cannot be eliminated, but it can be minimized.

The earliest of single processors on a chip had to carefully pick and choose what went on the chip. Cutting-edge designs today can fit everything but main memory on the chip.

how the world has changed
---

earliest computers had their greatest difficulties in getting the hardware to work --
- technology difficulties
  - space requirements
  - cooling requirements

given a working computer, scientists would jump through whatever hoops necessary to use it.

as hardware has gotten (much) faster and cheaper, attention has been diverted to software.
- OS
- compilers
- optimizers
- IPC (inter-process communication)

1 instruction at a time isn't enough. The technology isn't "keeping up." So, do more than one instruction at a time:

parallelism
---

instruction level (ILP) -- pipelining
superscalar -- more than one instruction at a time

multis
VLIW
supercomputer

WHICH OF THESE IS "BEST" and "FASTEST" DEPENDS ON WHAT PROGRAM

```
  IS BEING RUN -- THE INTENDED USEAGE.
```

on the 68000 Family

- released in the late 1970's
- an early "processor on a chip"
- a lot of its limitations have to do with what could fit on a VLSI chip in the late 1970's
- big early competitor of Intel

INSTRUCTIONS

- a relatively simple set, but NOT a load/store arch.
- a two-address architecture
- most instructions are specified in 16 bits -- fixed size.
- tight encoding, it is difficult to distinguish opcode from operands, but the m.s. 4 bits are always part of the opcode.

integer arithmetic
  different opcode for varying size data
  (add.b add.w add.l)
   8-bit 16-bit 32-bit

logical
  different opcode for varying size data

control instructions
  conditional branches, jumps
  (condition code mechanism used -- where most instructions had the effect of setting the condition codes)

procedure mechanisms
  call and return instructions

floating point ??
  (I guess not!)

decimal string
  arithmetic presuming representation of binary coded decimal

REGISTERS

16 32-bit general purpose registers,
only one is not general purpose (it is a stack pointer)
the PC is not part of the general purpose registers

the registers are divided up into two register files of 8,
  one is called the D (data) registers, and the other
  is called the A (address) registers. This is a distinction
  similar to the CRAY 1.

A7 is the stack pointer.

DATA TYPES

byte
word (16 bits)
longword (32 bits)

addresses are really their own data type.

arithmetic on A registers is 32 bit arithmetic. However, pin limitations on the VLSI chip required a reduced size of address. Addresses that travel on/off chip are 24 bits -- and the memory is byte-addressable. So a 24-bit address specifies one of 16Mbyte memory locations.

each instruction operates on a fixed data type

OPERAND ACCESS

the number of operands for each individual instructions is fixed

like the VAX, the addressing mode of an operand does not depend on the instruction. To simplify things, one of the operands (of a 2 operand instruction) must usually come from the registers (like the Pentium).

the number/type of addressing modes is much larger than the MIPS, but fewer than the VAX, similar to Pentium.

PERFORMANCE
?, they got faster as new technologies got faster.

ORIGINAL SIZE
1 64-pin VLSI chip (a huge number of pins at that time)

all about the Cray 1

There has always been a drive to design the best, fastest computer in the world. Whatever computer is the fastest has generally been called a supercomputer.

The Cray 1 earned this honor, and was the fastest for a relatively long period of time.

The man who designed the machine, Semour Cray, was a bit of an eccentric, but he can get away with it because he was so good. The Cray 1 has an exceptionally "clean" design, and that makes it fast. (This is probably a bit exaggerated due to my bias -- the Cray 1 is probably my favorite computer.)

Mostly my opinion:
To make the circuitry as fast as possible, the Cray 1 took 2 paths

1. Physical -- a relatively "time-tested" technology was used, but much attention was paid to making circuits physically close (Semour was aware of the limits imposed by the speed of light.) and the technology was pushed to its limits.
2. Include only what was necessary, but on that, a "don't spare the horses" philosophy was used.
   This means that extra hardware was used (not paying attention to the cost) wherever it could to make the machine faster. And, at the same time, any functionality that wasn't necessary (in Semour's

```
      opinion) was left out.
 Just remember:
   if something seems out of place to you, or some functionality of a
   computer that you think is essential and was not included in the Cray 1,
   it wasn't necessary!  And, leaving something out made the machine
   faster.

 What the Cray 1 was good for:
   It was designed to be used for scientific applications that required
   lots and lots of floating point manipulations.  It wouldn't make
   a good instructional machine (don't want to hook lotsa terminals up
   to it!), and it wouldn't be much fun to try to implement a modern
   operating system on.

 How it is used:
   most often, a separate (not as fast/powerful) computer was hooked up
   as what was commonly called a host computer.  The host is where you do
   all your editing and debugging of programs.  The host also maintains a
   queue of jobs to be run on the Cray.  One by one the jobs are run, so
   the only thing that the Cray is doing is running the final jobs -- often
   with LOTS of data.  Although its operating system would allow it,
   the "multi-tasking" (more than 1 program running simultaneously)
   ability was not often used.




instruction set
  fixed length instructions
    either 16 or 32 bit (no variability that depends on
     the number of operands)
  number of operands possible for an instruction
    0-3  (it is a 3-address instruction set)
  number and kind of instructions
            op codes are 7 bits long -- giving 128 instrucitons
             This includes complete integer and floating point
             instructions.

             Notice that missing from the instruction set are:
                 character (byte) manipulation, duplicates
                 of anything (!), integer divide

  Data representation is vastly simpified from what we've
     seen so far!  There are ONLY 64-bit 2's complement integers,
     24-bit addresses, and floating point numbers!

     ALL accesses to memory are done in WORD chunks.  A word
     on the Cray 1 is 64 bits.  All instructions operate on
     a single size of data -- Either a 64 bit word, or on an
     address (24 bits).

  addressing modes (strikingly similar to MIPS)
    Register Mode.  An instruction (op code) specifies exactly where
          the data is.
    Base Displacement Mode.  Used only for load and store instructions.



 REGISTERS:
   There are an ENORMOUS number of registers.
   There are 5 types of registers.
```

S registers -- 'S' stands for scalar. These are 64 bit regs. They are used for all sorts of data, but not addresses. There's 8 of them.

T registers --
These are 64 64-bit backup registers for the S registers. If you were to do some heavy programming on the Cray 1, you'd find these registers very useful. This is partially because you run out of S registers quickly, so you need temporary storage, but don't want your program to store to main memory (slow!). There's also an instruction that allows you to load a block of memory to the T registers. That's 1 instruction to do a bunch of loads.

A registers -- 'A' stands for address. These are 24 bit regs. They are used for addresses, and to a rather limited extent, integer counters.

B registers --
These are backups to the A regs and are used in the same manner as the T regs.

V registers -- 'V' stands for vector.
There are 8 sets of V regs. Each set has 64 64-bit registers! That is a lot! They are used mainly for processing large quantities of "array" data. Their use makes the Cray 1 very fast. A single instruction that uses a vector register (1 set) will cause something to happen to each of the 64 registers within that set.
(SIMD)

hardware stack
no support for stack accesses at all! There is no special stack pointer register.

cache
none. There's so many registers that there isn't really a need for one.

size of machine
A bit bigger than 2 refrigerators.

speed of machine
Significantly faster than the VAX and 68000.
For a while, it was the fastest machine around.

price of machine
As an analogy to some very pricey restaurants:
If you need to see the prices on the menu, you can't afford to eat there.

Probably about $3 million for the basic machine when they first came out.

A Cray 1 came with a full time hardware engineer, (a field service person). Why? Down time on a Cray is very expensive due to the way they are expected to be used. Waiting for field service to come was considered too expensive.

how many instructions get executed at one time

its debatable. There can be more than 1 instruction at some point in its execution at 1 time. It is a pipelined machine. This can only go so far (only 1 new instruction can be started each clock cycle).

complexity of ALU

There are actually quite a few alu's in the machine. Cray calls them functional units. Each one is a specialized piece of hardware that does its own job as fast as can be done. Each of them could conceivably be working at the same time.

on the VAX

The VAX was a popular and commercially successful computer put out in the early 1970's by DEC (Digital Equipment Corp).

It might be characterized by the term CISC.
RISC (Reduced Instruction Set Computer)
CISC (Complex Instruction Set Computer)

A CISC computer is often characterized by

1. many instructions
2. lots of addressing modes
3. (this one is debatable) variable length instructions
4. memory-to-memory architecture

Some details:

LOTS OF INSTRUCTIONS (like Pentium)
integer arithmetic
different opcode for varying size data
logical
different opcode for varying size data
address manipulations
bit manipulations
control instructions
conditional branches, jumps, looping instructions
procedure mechanisms
call and return instructions (there were more than 1!)
floating point (on more than one representation)
character string manipulations
crc (Cyclic Redundancy Check)
decimal string
arithmetic presuming representation of binary coded decimal
string edit

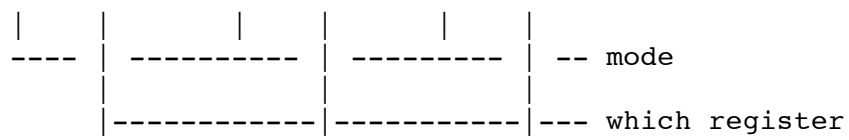
overall: more than 200 instructions

opcodes were of variable length, but always a multiple of 8 -- most opcodes were specified in the first 8 bits of an instruction.

REGISTERS
16 32-bit general purpose registers,

```
  except that they really weren't all general purpose
    R15 is the PC -- note that the programmer can change the
                     PC at will!
    R14 is a stack pointer
    R13 is a frame pointer
    R12 is an argument pointer (address of where a procedure's
          parameters are stored -- sometimes on the stack,
          and sometimes in main memory)

DATA TYPES
 byte
 word (16 bits)
 longword (32 bits)
 quadword (64 bits)
 octaword (128 bits)

 F floating point (32 bits -- 7 bits of exponent)
 D floating point (64 bits -- 7 bits of exponent)
 G floating point (64 bits -- 10 bits of exponent)
 H floating point (128 bits -- 15 bits of exponent)

 character string (consecutive bytes in memory, specified always
                   by a starting address and the length in bytes)
 numeric string  (the ASCII codes that represent an integer)
 packed decimal string (consecutive sequence of bytes in memory
     that represent a BCD integer.  BCD digits are each in
     4-bit quantities (a "nibble")

       example:  the integer +123 is represented by
               0001     0010   0011       1100
               (1)      (2)    (3)        (+)
   numbering   a<7-4> a<3-0>  a+1<7-4>  a+1<3-0>


each instruction operates on a fixed data type



OPERAND ACCESS

 the number of operands for each individual instructions
 is fixed

 a 3-address instruction set

 the location of operands is definitely not fixed,
   they can be in memory, or registers, and the variety
   of addressing modes that specify the location of an
   operand is large!

 equivalent of  Pentium    mov  EAX, ECX
                           add  EAX, EDX  ; EAX <- ECX + EDX

    addl3  R3, R4, R2
       ^^
       ||-- 3 operands
       |
       |--- operate on a 32 bit quantity
 ( there is also  addb3, addw3,  addb2, addw2, addl2)

 (and this is just for 2's complement addition!)
```

```
 This instruction does  R3 + R4 -> R2

 This is a VERY simple use of addressing modes.
 The syntax of operand specification allows MANY possible
   addressing modes -- every one presented this semester,
   plus more!

   for example
       addl3 (R3), R4, R2
         uses Register Direct addressing mode for the first
         operand --
         operation
           the address of the first operand is in R3,
           load the operand at the address, add to the
           contents of R4, and place the result into R2

  The addressing mode for each operand can (an often is)
  be different! NO restrictions (unlike Pentium and Motorola 68000)



One type addressing mode sticks out --
auto-increment and auto-decrement
  They have the side effect of changing the address used to get
  an operand, as well as specifying an address. (Motorola 68000
  has this addressing mode.)

    addl3 (R3)+, R4, R2
    operation
      the address of the first operand is in R3,  load
      the operand at the address, then increment the contents
      of R3 (the address), then add data loaded from memory
      to the contents of R4 and place the result into R2

      the amount added to the contents of R3 depends on the
      size of the data being operated on.  In this case, it
      will be 4 (longwords are 4 bytes)



MACHINE CODE

Together with each operand is an addressing mode specification.
Each operand specification requires (at least) 1 byte.

Format for the simple  addl3 R3, R4, R2

   8-bit opcode   0101 0011    0101 0100   0101 0010
                   ^    ^       ^    ^      ^    ^
                   |    |       |    |      |    |
                   ---- | ---------- | --------- | -- mode (register = 5)
                        |            |           |
                        |------------|-----------|--- which register


Format for the     addl3 (R3), R4, R2

      same
   8-bit opcode   0110 0011    0101 0100   0101 0010
                   ^    ^       ^    ^      ^    ^
```



Each instruction has an 8-bit opcode.
There will be 1 8-bit operand specifier for each operand that
the instruction specifies.

Because of the large number and variety of addressing modes,
an operand specification can be much more than 1 byte.
Example: Immediates are placed directly after their specification
within the code.

PERFORMANCE

the term MIPS (millions of instructions per second) really came
from the VAX --
the VAX 11 780 ran at just about 1 MIPS
note that this term is misleading --
Instructions take variable times to fetch and execute,
so the performance depends on the program

SIZE
one version: the VAX11 750 was about the size of a large-capacity
washing machine
another version: the VAX11 780 was about the size of 2 refridgerators,
standing side by side

the SPARC architecture

Scalar Processor ARChitecture
- - ---

developed in late 1980s by Sun Microsystems

some details:

-- single chip processor
-- load store architecture
-- machine code instructions are fixed size: 32 bits
-- control instructions based on condition code bits (like Pentium)
-- design goal: make procedure call and return efficient

So, the big difference between this architecture and others
is that it uses/assigns registers differently.

Think about the way that we write programs. There are
LOTS of procedures, and therefore lots of procedure calls
within a program. For each procedure call, parameters
have to be placed on the stack. Within the procedure,
the parameters are copied back out of the stack into
registers, in order to be used. Also, return values from

the procedure (function, really) must be placed somewhere.
This somewhere was usually on the stack.

What if . . .the current activation record at the top of
the stack was in registers, instead of memory? It would
make the program go faster, since accesses to the activation
record would really be to registers, NOT to memory.

This is what the SPARC processor does. There are a very
large number of registers. They are divided into overlapping
sets called WINDOWS. One window contains a procedure's
activation record.

See manuscript, page 320 for a diagram of this.
A program places parameters to a procedure in the
current window's OUTs. The procedure call switches
windows, such that the new window receives the
parameters in its INS.