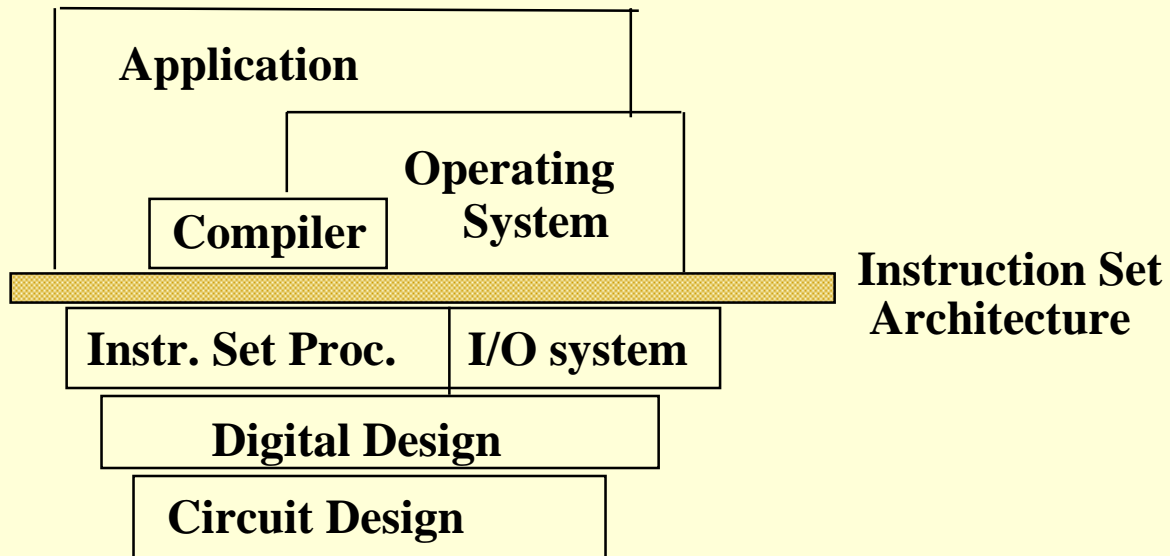


Instruction Set Architectures

Part 1



Some ancient history

- Earliest (1940's) computers were one-of-a-kind.
- Early commercial computers (1950's), each new model had entirely different instruction set.
- Programmed at machine code or assembler level
- 1957 – IBM introduced FORTRAN
 - Much easier to write programs.
 - Remarkably, code wasn't much slower than hand-written.
 - Possible to use a new machine without reprogramming.
 - Algol, PL/I, and other “high-level languages” followed
 - performance not quite as good as Fortran

Impact of High-Level Languages

- Customers were delighted
- Computer makers weren't so happy
 - Needed to write new compilers (and OS's) for each new model
 - Written in assembly code
 - Portable compilers didn't exist

IBM 360 architecture

- The first ISA used for multiple models
 - IBM invested \$5 billion
 - 6 models introduced in 1964
 - Performance varied by factor of 50
 - 24-bit addresses (huge for 1964)
 - largest model only had 512 KB memory
 - Huge success!
 - Architecture still in use today
 - Evolved to 370 (added virtual addressing) and 390 (32 bit addresses).

“Let’s learn from our successes” ...

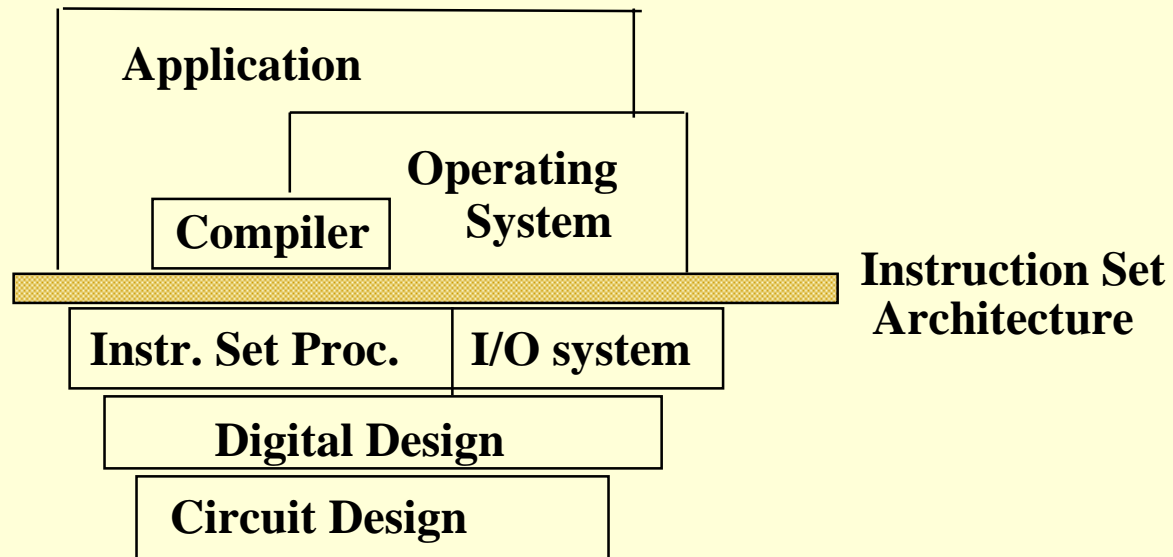
- Early 70’s, IBM took another big gamble
- “FS” – a new layer between ISA and high-level language
 - Put a lot of the OS function into hardware
- Huge failure

Moral: Getting right abstraction is hard!

The Instruction Set Architecture

The agreed-upon interface between:

the software that runs on a computer and
the hardware that executes it.



The Instruction Set Architecture

- that part of the architecture that is visible to the programmer
 - instruction formats
 - opcodes (available instructions)
 - number and types of registers
 - storage access, addressing modes
 - exceptional conditions

Overall goals of ISA

- Can be implemented by simple hardware
- Can be implemented by fast hardware
- Instructions do useful things
- Easy to write (or generate) machine code

Key ISA decisions

instruction length

- are all instructions the same length?

how many registers?

where do operands reside?

- e.g., can you add contents of memory to a register?

instruction format

- which bits designate what??

operands

- how many? how big?
- how are memory addresses computed?

operations

- what operations are provided??

Running examples

We'll look at four example ISA's:

- Digital's VAX (1977) - elegant
 - Intel's x86 (1978) - ugly, but successful (IBM PC)
 - MIPS - focus of text, used in assorted machines
 - PowerPC - used in Mac's, IBM supercomputers, ...
- VAX and x86 are CISC ("Complex Instruction Set Computers")
 - MIPS and PowerPC are RISC ("Reduced Instruction Set Computers")
 - almost all machines of 80's and 90's are RISC
 - including VAX's successor, the DEC Alpha

Instruction Length

Variable:

x86 – Instructions vary from 1 to 17 Bytes long

VAX – from 1 to 54 Bytes

Fixed:

MIPS, PowerPC, and most other RISC's:

all instructions are 4 Bytes long

Instruction Length

- Variable-length instructions (x86, VAX):
 - require multi-step fetch and decode.
 - + allow for a more flexible and compact instruction set.
- Fixed-length instructions (RISC's)
 - + allow easy fetch and decode.
 - + simplify pipelining and parallelism.
 - instruction bits are scarce.

What's going on??

- How is it possible that I SA's of 70's were much more complex than those of 90's?
 - Doesn't everything get more complex?
 - Today, transistors are much smaller & cheaper, and design tools are better, so building complex computer should be easier.
- How could I BM make two models of 370 I SA in the same year that differed by 50x in performance??

Microcode

- Another layer - between ISA and hardware
 - 1 instruction → sequence of microinstructions
 - μ -instruction specifies values of individual wires
 - Each model can have different micro-language
 - low-end (cheapest) model uses simple HW, long microprograms.
- We'll look at rise and fall of microcode later
- Meanwhile, back to ISA's ...

How many registers?

All computers have a small set of registers

Memory to hold values that will be used soon

Typical instruction will use 2 or 3 register values

Advantages of a small number of registers:

It requires fewer bits to specify which one.

Less hardware

Faster access (shorter wires, fewer gates)

Faster context switch (when all registers need saving)

Advantages of a larger number:

Fewer loads and stores needed

Easier to do several operations at once

In 141, “load” means moving data from memory to register, “store” is reverse

How many registers?

VAX – 16 registers

R15 is program counter (PC)

Elegant! Loading R15 is a jump instruction

x86 – 8 general purpose regs Fine print – some restrictions apply

Plus floating point and special purpose registers

Most RISC's have 32 int and 32 floating point regs

Plus some special purpose ones

- PowerPC has 8 four-bit “condition registers”, a “count register” (to hold loop index), and others.

Itanium has 128 fixed, 128 float, and 64 “predicate” registers

Where do operands reside?

Stack machine:

“Push” loads memory into 1st register (“top of stack”), moves other regs down

“Pop” does the reverse.

“Add” combines contents of first two registers, moves rest up.

Accumulator machine:

Only 1 register (called the “accumulator”)

Instruction include “store” and “acc ← acc + mem”

Register-Memory machine :

Arithmetic instructions can use data in registers and/or memory

Load-Store Machine (aka Register-Register Machine):

Arithmetic instructions can only use data in registers.

Load-store architectures

can do:

```
add r1=r2+r3
```

```
load r3, M(address)
```

```
store r1, M(address)
```

⇒ forces heavy dependence on registers, which is exactly what you want in today's CPUs

can't do:

```
add r1=r2+M(address)
```

- more instructions
- + fast implementation (e.g., easy pipelining)

Where do operands reside?

VAX: register-memory

Very general. 0, 1, 2, or 3 operands can be in registers

x86: register-memory ...

But floating-point registers are a stack.

Not as general as VAX instructions

RI SC machines:

Always load-store machines

I'm not aware of any accumulator machines in last 20 years.
But they may be used by embedded processors, and might
conceivable be appropriate for 141L project.

Comparing the Number of Instructions

Code sequence for $C = A + B$

<u>Stack</u>	<u>Accumulator</u>	<u>Register-Memory</u>	<u>Load-Store</u>
Push A	Load A	Add C, A, B	Load R1, A
Push B	Add B		Load R2, B
Add	Store C		Add R3, R1, R2
Pop C			Store C, R3

Alternate I SA's

$$A = X * Y + X * Z$$

Stack

Accumulator

Reg-Mem

Load-store

'Computer" of the day

Jacquard loom

late 1700's

for weaving silk

"Program" on punch cards

"Microcode": each hole
lifts a set of threads

"Or gate": thread lifted if
any controlling hole punched



Card Punch

- Early “programmers” were well-paid (compared to loom operators)

