# Lecture # 39

- Instruction Set Architecture
- CISC vs. RISC

# Electronic Discrete Variable Automatic Computer (EDVAC)

- ENIAC's programming system was external
  - –Sequences of instructions were executed independently of the results of the calculation
  - –Human intervention required to take instructions "out of order"
- Eckert, Mauchly, John von Neumann and others designed EDVAC (1944) to solve this problem
  - –Solution was the *stored program computer*
    - ® *"program can be manipulated as data"*
- *First Draft of a report on EDVAC* was published in 1945, but just had von Neumann's signature!
  - –In 1973 the court of Minneapolis attributed the honor of *inventing the computer* to John Atanasoff

# And then there was IBM 701

IBM 701 -- 30 machines were sold in 1953-54

IBM 650 -- a cheaper, drum based machine,
more than 120 were sold in 1954
and there were orders for 750 more!
- eventually sold about 2000 of them

*Users stopped building their own machines.*

Why was IBM late getting into computer technology?

*IBM was making too much money!*
Even without computers, IBM revenues were doubling every 4 to 5 years in 40's and 50's.

## Into the 60's…:

# Compatibility Problem at IBM

By early 60's, *IBM had 4 incompatible lines of computers!*

| | | |
|---|---|---|
| 701 | □ | 7094 |
| 650 | □ | 7074 |
| 702 | □ | 7080 |
| 1401 | □ | 7010 |

Each system had its own

- Instruction set
- Peripherals: magnetic tapes, drums and disks
- Programming tools: assemblers, compilers, libraries,...
- market niche: business, scientific, etc....

® *IBM 360*

# IBM 360 : Design Premises

## *Amdahl, Blaauw and Brooks, 1964*

http://www.research.ibm.com/journal/rd/441/amdahl.pdf

- Breaks the link between ***programmer and hardware***
- Upward and downward, machine-language compatibility across a family of machines
- General purpose machine organization, general I/O interfaces,

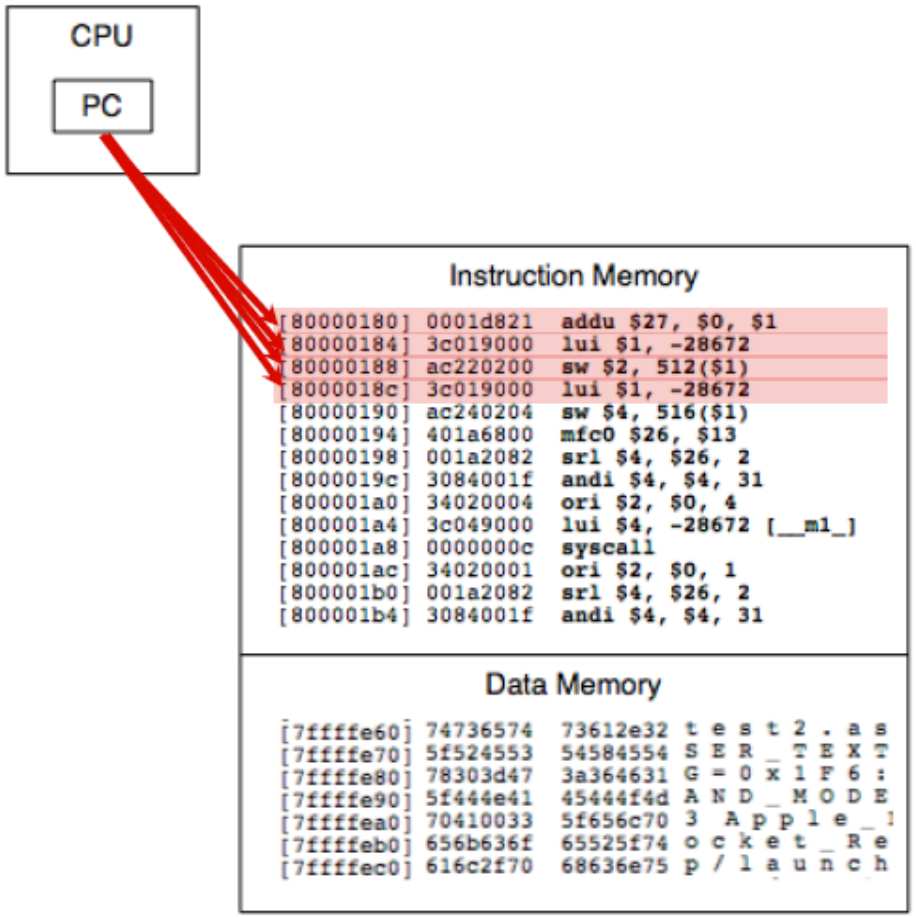
  storage > 32K
- Easier to use (answers-per-month vs. bits-per-second)
- Machine must be capable of *supervising itself* without manual intervention →OS/360 (simple OS's in IBM 700/7000)
- Built-in *hardware fault checking* and locating aids to reduce down time

… **the use of the "ISA" as a compatibility layer was a $175 billion project (2011 dollars)**

The Amdahl .. from Amdahl's Law.
The Brooks .. from The Mythical Man-Month.

# The Stored Program Computer

- The program is *data*
  - It is a series of bits
  - It lives in memory
  - A series of discrete "instructions"
- The program counter (PC) control execution
  - It points to the current instruction
  - Advances through the program



# The Instruction Set Architecture (ISA)

- The ISA is the set of instructions a computer can execute
- All programs are combinations of these instructions
- It is an abstraction that programmers (and compilers) use to express computations
  - The ISA defines a set of operations, their semantics, and rules for their use.
  - The software agrees to follow these rules.
- The hardware can implement those rules IN ANY WAY IT CHOOSES!
  - Directly in hardware
  - Via a software layer (i.e., a virtual machine)
  - Via a trained monkey with a pen and paper
  - Via a software simulator (like SPIM)

# RISC vs CISC

# In the Beginning…

- 1964 -- The first ISA appears on the IBM System 360
- In the "good" old days
  - Initially, the focus was on usability by humans.
  - Lots of "user-friendly" instructions (remember the x86 addressing modes).
  - Memory was expensive, so code-density mattered.
  - Many processors were *microcoded* -- each instruction actually triggered the execution of a builtin function in the CPU. Simple hardware to execute complex instructions (but CPIs are very, very high)
- …so…
  - Many, many different instructions, lots of bells and whistles
  - Variable-length instruction encoding to save space.
- … their success had some downsides…
  - ISAs evolved organically.
  - They got messier, and more complex.

# Things Changed

- In the modern era
  - Compilers write code, not humans.
  - Memory is cheap.  Code density is unimportant.
  - Low CPI should be possible, but only for simple instructions
  - We learned a lot about how to design ISAs, how to let them evolve gracefully, etc.
- So, architects started with with a clean slate...

# Reduced Instruction Set Computing (RISC)

- Simple, regular ISAs, mean simple CPUs, and simple CPUs can go fast.
  - Fast clocks.
  - Low CPI.
  - Simple ISAs will also mean more instruction (increasing IC), but the benefits should outweigh this.
- Compiler-friendly, not user-friendly.
  - Simple, regular ISAs, will be easy for compilers to use
  - A few, simple, flexible, fast operations that compiler can combine easily.
  - Separate memory access and data manipulation
    - Instructions access memory *or* manipulate register values. Not both.
    - "Load-store architectures" (like MIPS)

# RISC Characteristics of MIPS

- All instructions have
  - <= 1 arithmetic op
  - <= 1 memory access
  - <= 2 register reads
  - <= 1 register write
  - <= 1 branch
  - It needs a small, fixed amount of hardware.
- Instructions operate on memory *or* registers *not both*
  - “Load/Store Architecture”
- Decoding is easy
  - Uniform opcode location
  - Uniform register location
  - Always 4 bytes -> the location of the next PC is to know.
- Uniform execution algorithm
  - Fetch
  - Decode
  - Execute
  - Memory
  - Write Back
- Compiling is easy
  - No complex instructions to reason about
  - No special registers
- The HW is simple
  - A skilled undergrad can build one in 10 weeks.
  - 33 instructions can run complex programs.

# CISC: x86

- x86 is the prime example of CISC (there were many others long ago)
  - Many, many instruction formats.  Variable length.
  - Many complex rules about which register can be used when, and which addressing modes are valid where.
  - Very complex instructions
  - Combined memory/arithmetic.
  - Special-purpose registers.
  - Many, many instructions.
- Implementing x86 correctly is almost intractable

# x86 ISA Caveats

- x86 is a poorly-designed ISA
  - It breaks almost every rule of good ISA design.
  - There is nothing “regular” or predictable about its syntax.
  - We don’t have time to learn how to write x86 with any kind of thoroughness.
- It is the most widely used ISA in the world today.
  - It is the ISA you are most likely to see in the “real world”
  - So it’s useful to study.
- Intel and AMD have managed to engineer (at considerable cost) their CPUs so that this ugliness has relatively little impact on their processors’ performance (more on this later)