Evolution of ISA's

ISA has changed over computer "generations"

A traditional way to look at ISA complexity encompasses

- Number of addresses per instruction
- Regularity of instruction formats
- Number of addressing types

Number of addresses per instruction

- Started with one address and "implied" accumulator
- Then 1 address + index registers
- Followed by 1 address + general registers
- Then 2 or 3 addresses and general registers
- Then N addresses (VAX)
- Also stack computers (0-address)

Addresses per instruction

In RISC machines:

- 3 addresses but the operands are registers (or 2 registers and 1 immediate)
- 2 addresses for load-store with 1 being a register

In CISC machines like IBM System/360 family and Intel x86 family

- 2 addresses (source $\leftarrow$ source op. destination)
- 1 operand is a register the other can be either register, memory adress, or immediate
- Some instructions (manipulating strings) can have 2 memory addresses as operands

Regularity of instruction formats

- Started with fixed format (ease of programming)
- Then more flexibility (assembly language/compiler) Three or four instruction formats not necessarily the same length.
- Then strive for memory compactness. Encoding of variable length instructions.
- Back to regular instruction sets (memory is cheap; encoding must be simple for instructions to be executed fast)

Number of instruction formats

In RISC machines, all instructions have the same length. Very few instruction formats (3 or 4)

In CISC machines, instructions of various lengths

- In System 360, instructions can be 2,4, or 6 bytes long
- In Intel x86, they can be from 1 to 17 bytes

In CISC machines, larger number of instruction formats:

- In System 360, 5 instruction formats
- In Intel x86, ...many .... with lots of variations due to specifiers.

Example: IBM 360, 370, 30xx, 43yy

ISA defined mid-60's.

Two-address; 4+ instruction formats of length either 2,4, or 6 bytes.

RR format

OPC	R1	R2

R1 <- R1 op. R2

RX format

OPC	R1	X2	B2	displacement

R1 <- R1 op. mem[X2 + R2 + displ]

RS format

OPC	R1	R3	B2	displacement

mem[B2 + displ] <- [R1-R3]

SI format

OPC	value	B2	displacement

mem[B2 + displ] <- mem[B2+displ] op. value

SS format

OPC	count	R1	address 1	R2	address 2

mem[address2 + R2 + i] <- mem [address 1 + r1 +i], 0<=i<count

Addressing modes

- Start with immediate, direct, indirect
- Then index registers
- Then index + base (i.e., allow the sum of 2 registers to be an address. Some RISC machines, e.g., IBM PowerPC have this feature)
- CISC's have all kinds of addressing

Example: VAX-11, the ultimate CISC

ISA defined late 70's.

Over 200 instructions

- Variable length instructions
- "Powerful" instructions: e.g., polynomial evaluation, "add, compare, and branch", procedure call with register saving and frame set-up etc.

Complex addressing modes

VAX addressing modes: a sample

1. Immediate (with even some small f-p constants) [MOVL constant, register]
2. Direct (register) [CLRL register] (one for each data type)
3. Indirect (or deferred) [CLRL (register)] Clear the memory location whose address in the register
4. Autoincrement [CLRL (register)+] as above but after the CLRL the register is incremented by 4 (or 1 if CLRB etc...)
5. Autodecrement as above but the register is decremented before access
6. Autoincrement deferred [CLRL @(register)+] now register points to the address of the address
7. Displacement [CLRL constant(register)] the constant has a modifier for bytes, half-words, words etc.
8. Displacement deferred [CLRL @constant(register)]
9. Index mode (see next slide)

VAX indexing mode

The index register [Rx] is multiplied by the size of the operand in bytes to yield the "adjustement index". Then the adjustment index is added to the address computed according to the other modes.

Example:

```
CLRL    A[R1]          #Clear the word at address A + 4*R1
CLRL    (R4)+[R1]      #Clear the word at address R4 + 4*R1
                       #Increment R4 by 4 after the address
                       #is computed
CLRL    12(R4)[R1]     #Clear the word at address R4 +12*4 +R1*4
```

Intel x86: the largest number of CPU's in the world

Compatibility hurts! Only 8 general registers.

Has an ISA for 16-bit and an extended one for 32 bits

Addressing relies on segments (code, stack and 4 data segments)

Lots of different addressing modes (but fewer than in VAX)

Instruction encoding in Intel x86

Quite complex.

Opcode 1 or 2 bytes. One bit in opcode for size of operand.

First byte following opcode is operand specifier (can specify 2 registers, or one register and the next byte specifies base and address register, and possibly next (1,2,4) byte is a displacement etc...)

No regularity in instruction set.

But .. more than 300 millions x86 in the world today (about 85% of the market for computers; less if you count embedded processors where Motorola has a significant share)

MIPS is not the only RISC

RISC trend for general-purpose computing started in early 80's as a reaction to ever more complex CISC's.

- Research at IBM for the IBM 801 that became RT/PC that led to RS6000 and now to PowerPC
- Research at Berkeley for the RISC I and II that led to SUN SPARC processors
- Research at Stanford for the MIPS that led to MIPS Co., now bought by SGI, MIPS Rxxxx
- HP Precision architecture
- DEC Alpha (the fastest –clock-wise– microprocessor today)

How about the new Intel IA-64 architecture?

Code name Merced

Intel-HP collaboration

Upward compatible with x86?
Details of the ISA have not yet been disclosed.

A review of basic RISC ideas

- Fixed length, easy to decode (regular), instructions
- Load-store architecture
- Few addressing modes

with the consequences:

- Decoding and control hardwired
- Striving for small cycle time and small (equal for all instructions) number of cycles per instruction (CPI)
- "Expose" the architecture to the compiler/programmer

Current RISC are not “that restricted” !!

- special instructions for graphics, multimedia, managing some hardware resources (but these instructions are still “simple”)
- super-scalar machines
- speculative execution
- out-of-order issue
- etc..

However, the Ultimate driving force is Speed!

$$Exec.time = Number\ of\ instructions \times CPI \times clock\ cycle\ time$$

(cf. Chapter 2)