Computer architecture

prepared for Programmable Logic Lessons / 1.4 by ~skmp

Kindly hosted by hackerspace.gr

Structure of a computer

- Units connected to a bus
  - CPU, RAM, GPU, I/O ports
- There are usually many buses depending on the requirements
  - ex. SATA, PCI-E, USB
  - Chips internally are similar in design, but use mostly custom protocols

Typical design flow

- Design an ISA (Instruction Set Architecture)
- Implement the ISA in software for testing
  - Iterate early a lot (to improve the ISA – changes are much more expensive in later stages of development)
- Break down the cpu into blocks than when connected together implement the desired functionality
  - These blocks can be worked on and tested partly independently of each other
- Do functional simulation using the verilog/vhdl model
  - Mostly a simplified model
- Optimize/re-design the verilog/vhdl model to meet performance, area and power requirements
  - Make sure everything still behaves as expected
- Maintain a -big- test suite that thoroughly tests everything. FPGAs are easy to update, but actual hardware isn't
  - Finding the right corner cases to test (to avoid test-bloat) is a very delicate task

Why so many implementations ?

- ISA is specified/implemented many times
  - Text, High level Simulator, HDL for simulation, HDL for implementation
- Each “stage” should behave identically but adds more details to the specification
  - Eg, a high level simulator doesn't have as much concurrency as an HDL implementation
- The more detail you have, the harder it is to modify the design
- Detailed cross-validation/verification is usually possible within “one” complexity 'step'
- (most of the cost in a modern design is verification)
- Wikipedia has a nice article

ISA Details (1)

- Typically, cpus have internal memory called registers
  - Implemented as flip flops/SRAM
  - Just a few (8 to 128 on modern cpus)
- Operations are usually done between registers
  - Some architectures can operate directly on memory as well, eg x86
- The register size defines the largest possible amount of data that can be processed at once
  - 32 bit is most common, with 64 bits gaining popularity
  - Usually there are also dedicated “media processing”/SIMD registers that are larger but they can't be used for general computation
    - SSE on x86, NEON or arm, etc
- Example
  - Mov r0, 10 // move the value 10 to register 0
  - Mov r1, 30 // move the value 30 to register 1
  - Add r2, r1, r0 // compute r1 + r0 and store it to r2
  - Store r2, r3 // write to the memory, at address r3 the contents of r2

ISA Details (2)

- CPU Commands are encoded into bytes called opcodes (operation codes)
- Some architectures have variable length encodings, and others fixed length
  - X86 is variable length, arm, mips and ppc are fixed length .. mostly :)
- There used to be a clear distinction between RISC and CISC designs, but with the growing amount of complexity it doesn't make that much of a difference
  - RISC – few simple instructions that work fast
  - CISC – more complicated instructions that take more time
- There's also VLIW that has expresses much more details about the cpu at the expense of much longer instructions
  - 256 bit instructions are common in VLIW world
  - RISC/CISC break down the instructions usually to an internal VLIW format
- Instruction encoding is a trade-off between space efficiency and how 'expressive' the opcodes can be
  - So, its like a form of compression to save memory and bandwidth

ISA Details (3)

- Typical opcodes
  - I/O (typically Load / Store ↔ memory)
  - ALU (add, cmp, xor, not, and, etc)
  - Branches (change the “flow” of the program)
- Typical register names r0, r1, .. rN
  - Some architectures name their registers, eg x86 has eax,ecx,ecx, … but in newer versions they have rN-like aliases as well
- Sometimes sub-parts of registers can be accessed to perform eg 16 bit operations on 32 bit registers

A simple cpu

- 6502 Is a primitive and simple cpu you can use to learn
  - It was used at NES, an old 8 bit console
- 6502 low-level implementation
  - Pretty descriptive, almost schematic-level detail
  - Taken from pagetable
    - Amazing article, wonderful site. Go read it
- Online assembler/emulator
  - Another one
- Actual silicon/VLSI/chip level emulation in your brower

Modern architectures

- You can study some existing ISAs to get a general idea
  - X86 (X86-64 is the 64 bit ver, examples, video)
    - Home computers, PS4, XBOX ONE
  - ARM (ARMv4, ARMv7a are common, examples)
    - Most mobile devices, routers, etc
    - GBA/NDS
  - MIPS (mips32, mostly, examples)
    - Mobile devices, psp, ps1, ps2, N64
  - PPC (ppc32, ppc64, example)
    - Xbox360, ps3, wii

Emulation/Simulation

- During the design phase, most cpus are implement first in C/C++ or a similar language
  - Faster to work with than verilog
  - Can be used to run programs on the pc, for testing and evaluation
- Typical structure is a fetch, decode, execute loop
  - eg. for (;;) { int op = read(pc); pc ++; switch(pc) { ….} }
- Take a look at the repo for a simple emulator

Thanks !

Next week we'll get into more details about cpu ISA and inner workings !

Feel free to drop by #hsgr @ freenode

... or the hsgr mailing list

... and use the wiki !

(or, send direct feedback – skmp@emudev.org)