Evan Smith

CSE 661

HW 1

1 – Summary of papers:

Design of the B5000 System, Lonergan and King

- Overall goals of system were increased compute performance and ease of change for both software and hardware.
- Wanted to separate software concepts like data, program, and memory locations from the specifics of the hardware configuration. Additionally, it should work in reverse such that changes to the hardware should not warrant software updates, given that the initial interfaces were respected.
- Nominal flow is to run a master control program which seems to act as an OS. To execute programs or make other actions, and interrupt is used to queue work.
- The system's instruction set has both static and dynamically sized instructions, depending on the opcode. These modes were called word mode and character mode.
- Word mode used settings that enabled a more streamlined representation of fractional values and increased the range of expressible numbers.
- Character mode was intended to improve performance for replacement of characters in data, was able to store more information per unit of memory and was helpful for lists.
- Programs for the system were made up of instruction words, with each word consisting of 4 "syllables" of 12 bits each.
- The B5000 used a pushdown stack, which uses a program counter of sorts to keep track of the top of the stack and moves the two most recent values into two on-processor registers. These special registers also had flags to denote if they were empty or not, which prevented pushing operands to memory if they would just be immediately pulled back up into the registers.
- The system used right-hand polish notation to make the operator precedence obvious.
- Independent addressing allowed for automatic program segmentation, allowing for easier programming and up to 60,000 instructions for a minimum system.
- Branching within programs was relative, not absolute, which allowed jumps to move large distances and the required program memory to be fetched as required.
- The system supported all four types of direct and indirect addressing in operands.

The Case for the Reduced Instruction Set Computer, Patterson and Ditzel

- This paper interrogates the idea that increased complexity in the design of computers is cost-effective. They argue that in some cases a reduced complexity increases cost-effectiveness, specifically RISC vs CISCs.
- The first case study brought up was the addition of a few more higher-level instructions when there was a clear and notable improvement in speed required and achieved, since the processor was much faster than memory for subroutines.
- They further show that as microprogrammed control meant that control memories were in powers of 2, and therefore adding common subroutines into instruction architecture was essentially free, since the instruction space was already allocated.
- When memory was expensive, complex instruction sets allowed for more dense code. This benefit may have evaporated now that memory cost is smaller, since reducing instructions simplifies CPU performance.
- They also claim that the easy way to maintain compatibility in later versions is to add features, making removal or rework of previous, more basic instructions difficult and expensive to develop.
- When interrupts are involved, a CISC system has more information to store as its state vs a RISC one.
- As higher-level languages were introduced, the number of programmers writing assembly dropped, and compilers generally only use a small set of instructions.
- Technological improvements meant that often a compiler-style usage of many lower-level instructions was faster than a single, higher-level instruction.
- Design time and difficulty increases as instructions increase.
- RISC allowed single-chip implementation with less technological advancement since the architecture is necessarily less complex.
- As more instructions are added, the system cycle is slowed down due to the additional gates, and so the added instruction should be able to justify the clock slow down.
- As the goal of the computers turned to using exclusively high-level languages, it seems clear that everything will be compiled. The best compilers are possible with low-level instructions.
- Several different labs were working on a RISC architecture that was better positioned for the future.
- In conclusion, the authors ask architects to consider the system-wide costs of adding an additional instruction to promote cost-effectiveness, and posit that RISCs are the best path forward.

Comments on "The Case for the Reduced Instruction Set Computer", Clark and Strecker

- The authors contest the previous paper's claims that RISC architectures are as cost-effective as CISC.
- They claim that to prove such an assertion would require fully building or simulating the processors, compilers, and even OS.
- They push on the comparison between reduced and complex instruction sets. They point out that adding additional instructions that simply accommodate a datatype do not really make the set more complex, just more complete.
- They push for proof of the claim that the RISC can achieve equally compact assembly code as a CISC and discard the idea that less dense code is no issue since memory cost decreases, as there is certainly a savings if less memory is required.
- They note that more instructions help provide the small number of instructions needed by several different high-level languages. They use FORTRAN and COBOL as foils, where the top instructions in FORTRAN are not used nearly as often in COBOL.
- The measurement of usage should be CPU time used, not frequency of calculation since that is often where improvements can be made.
- They claim that the increase in microcode size cannot be written off as negative, since the example used in the paper has increased performance in the processor with more microcode while supporting more instruction sets.
- They discard the development time argument, as the requirements of larger-scale production at larger companies will fundamentally take longer.
- They point out that the increase in errors is inherent in the problem of writing more complex code and programs, whether those errors occur in the compiler for a RISC or the microcode for a CISC.
- They also state that a CISC provides more cases for using specialized hardware solutions for some instructions, while using RISC instructions would reduce those cases.
- The authors refute the idea that marketing strategy drives instruction set complexity.

Architecture of the IBM System/360

- Objectives of the system design are focus on business data, IO, compilers, magnetic memory, rapid storage improvements, and rise of real-time systems.
- As new features were designed, they were sure to plan for future systems in the family. They used open-ended components to help promote that compatibility.
- The format of data was a key decision, including 4/8 character sizes, 32/64 length floating point words, hex floating point representations, twos-complement sign representation, variable length decimal fields, storage-storage operation, length field specification, support for both BCD and ASCII (which wasn't yet approved as the standard!), and boundary rules based on the field length.
- Instruction decisions included using an addressed registers scheme vs a stack, binary addressing, multiple accumulators, and separate accumulators.
- IO used channels as distinct operating entities to allow for IO usage of many different hardware types depending on scale and scope of the system in use.

1.1 A) Yield for IBM Power5 given
Die size = 389 mm^2, Defect Rate 0.03 defects / cm^2,
Man. size = 130 nm, and 276 mil ~~resist~~ transistors.

$$\text{Die Yield} = \text{Wafer Yield} \times \frac{1}{(1 + \text{defect/area} \times \text{area})^N}$$

$$= 1.0 \times \frac{1}{(1 + 0.0003 \times 389)^{13.5}}$$

$$= \frac{1}{(1.1167)^{13.5}}$$

$$= \frac{1}{4.438}$$

$$= 22.5\%$$

1.5 A) Given each server has an Intel Pentium 4, 1 GB of 240 pin DRAM, an a 7200 RPM Hard Drive, the range of power consumption per server is $\left.\begin{matrix} 48.9 + 2.3 + 4.0 = 55.2\ W \\ 66.0 + 2.3 + 7.9 = 76.2\ W \end{matrix}\right\}$ per server.

With a cooling door handling 14,000 W, one door can handle $14{,}000 / 76.2\ W = 183.7 \rightarrow$ **183 servers**