Disclaimer: "The contents of this document are scribe notes for The University of Texas at Austin EE382V Spring 2007, Computer Architecture: User System Interplay". The notes capture the class discussion and may contain erroneous and unverified information and comments.

Terminology

Lecture #2: Wednesday, 24 Jan 2007
Lecturer: Mattan Erez
Scribe: Min Kyu Jeong
Reviewer: Mattan Erez

1 Users

Someone whose requirements system must satisfy. There is a variety of users and requirements vary greatly. A single system may have multiple types of users, for example a PC should meet the requirements of end-users at the same time that of developers.

1.1 Consumers

End users, general public who simply buy the system and use it with software that *developers* build. Systems such as PCs (General Purpose), home appliances, cars, web servers, game consoles and mobile gadgets must be built to meet the needs of this type of users. Engineers and scientists are in between consumers and developers because they both use existing software as well as do programming.

1.2 Developers

Users who develop software to run on the system. They build PC software, server software, development tools, etc. Programmability of the system is an important requirement for this type of user.

1.3 Service Providers

Service providers such as web-portals, search engine and banks use web servers, transaction processing systems, DB systems and etc.

1.4 Vertical Systems

A vertical system is which designers control all aspects of the system, from hardware to application software. Therefore, they do not consider much about extendability or portability. Car, signal processors or government systems are examples of this type of systems.

2 Requirements

Different users have different requirements. Some requirements apply to all types of users (ex. cost), but some may not be an issue for certain types of users (ex. compatibility for vertical system users). Even for the same requirement, standards differ (ex. cost for supercomputer and mobile devices)

- **Cost** - How much does it cost to build, sell and maintain?
- **Performance** - Meaning of performance may differ depending on applications of the system. ex. latency constraints of embedded systems, throughput for transaction systems.
- Ease of use - How easy is the interface to use considering expected users and environments.
- Aesthetics -
- Reliability, Lifetime
- Availability - The proportion of time a system is in a functioning condition. "Server downtime costs $108,000 a minute in lost brokerage operations." source: Contingency Planning Research (http://www.contingencyplanningresearch.com), a Division of Eagle Rock Alliance
- Serviceability - *RAS* is a common acronym for Reliability, Availability and Serviceability.
- Size, Portability - ex. PCs should be in stackable for suppliers.
- **Power** - How much power does the system consume. ex. power budget for battery operated systems
- Security - How system is secure against malicious users.
- Compatibility - How compatible with different generations of the system or other components of a bigger system.
- Scalability - Is it possible/How much does it cost to scale the system?

- Flexibility, Programmability, Reusability, Functionality - ex. camera/mp3 capability for cell phones
- Business aspects, Branding, Market forces, Licensing - ex. Best selling IPod is not much better than other mp3 players in terms of cost or performance.
- Environment - environment under which the system will operate. ex - military systems, space, underwater, etc.

Many requirements mentioned above are more related to consumer users.

2.1 Functionality Partitioning

There are multiple ways to meet given requirements. For example, scientific applications require answers to be correct. Designer can replicate 3 identical systems and compare results (microarchitecture level), or run an algorithm that have redundancy in itself (application level). Latter solution obviously has advantage in cost, but performance may well be better with the former solution. Solutions can be at different levels of system (Application/ System Software/ Architecture/ Microarchitecture/ Circuits/ Devices/ Packaging). The problem is to determine pros/cons of each solution.

3 Cost

If you are willing to pay, you can build just about anything. Cost is the most prohibitive factor in general.

$$\text{Cost} = \text{System ?!}$$

The above claim basically implies that building a system is a tradeoff of cost and capabilities (capabilities are answers to user requirements)

One of the best ways to evaluate a system is by estimating how much it would cost to support the requirements of the user. A different way of saying it is that we can develop metrics for evaluating a system that will let us a the end draw conclusions about its cost and make fair comparisons.

- Supply chain - cost for transportation, storage and inventory cost
- Engineering and manufacturing effort - actual manufacturing cost such as labor, equipment depreciation and etc.
- Research and design effort

- Part cost

  die area, volume, process technology, complexity, design for manufacturing, verification and testing

  formal verification method is generally applicable verification step-by-step procedure, but cannot use it for very complex design

- Power - power cost, special packaging, cooling devices
- Business (Marketing, Branding, etc)
- Maintenance
- Application Software - in general, the most expensive part in system. Software are a lot more complex than hardware in many cases. ex. probably code size for MS word bigger than that of processors.

4 General Purpose Processor - Next Class

4.1 What characterize GPP?

The definition of what general purpose processors is somewhat vague, but there are some characteristics identifiable from them. First of all, they have good programmability which makes them 'general purpose'. However, programmability itself cannot define GPP, as that is the quality many other types of processors share too. For example, FPGA is programmable, but it is not considered as GPP.

They have many off-the-shelf applications available. Because the software are sold in volume, it may not be very expensive to buy, but is very expensive and time consuming to develop. However, the most important feature that characterize GPP is **Backward compatibility**. Backward compatible processors can run legacy codes, which is a very desirable characteristics for users who want to improve performance of their system, but to keep the software they have been using. As stated above, there are multiple ways to achieve this backward compatibility requirement.

4.1.1 Binary Level Compatibility

Binary level compatible processors can run old legacy binary as is. It can be achieved by hardware, or combination of hardware and software. Hardware techniques keep ISA the same while change microarchitecture incorporating new technologies. A good example of this is Intel's superscalar processors which internally execute multiple translated uOps out of order, but giving the illusion of sequential, one-at-a-time execution of x86 instructions. Hardware/software combination techniques uses a layer of binary translation software that translates old ISA instructions to underlying architecture's instructions on the fly.

Transmeta Crusoe processors is this type of processors which runs x86 instructions on VLIW processor.

4.1.2 Source Level Compatibility

Source level compatibility means new processors can run old programs by recompiling the source code. Although execution model of new processor might be radically different from that of old processors, compilers produce binaries from previously written source codes. Execution models of TRIPS or WaveScalar are radically different from that of conventional Von Neumann machine, but their compilers accept C or Fortran code and generates binary for them.

4.1.3 Programming Model Compatibility

Sequential programming model, Algorithm level compatibility