18-447 Lecture 4: Development of ISAs

James C. Hoe
Department of ECE
Carnegie Mellon University

Housekeeping

- Your goal today
  - understand how ISAs got to be the way they are
- Notices
  - Lab 1, Part A, due this week
  - Lab 1, Part B, due next week
  - HW1, due 2/8
- Readings
  - P&H Ch 2 (optional P&H Appendix E)
  - optional (in supplemental handout on Blackboard)
    - 1946 von Neumann paper
    - 1964 IBM 360 paper
  - P&H Ch 1.6, 1.7 and 1.9 for next time
Computer Architecture is Engineering

- An applied discipline of finding and optimizing solutions under the joint constraints of demand, technology, economics, and ethics
- Thus, instances of what we practice evolve continuously
- Need to learn the principles that govern how to develop solutions to meet constraints

Don’t memorize what you see; ask why it is

Deep Influence of von Neumann Arch.

- By far the most common architectural paradigm
- Memory holds both program and data
  - instructions and data in a linear memory array
  - instructions can be modified just like data
- Sequential instruction processing
  1. program counter (PC) identifies the current instruction
  2. instruction is fetched from memory
  3. instruction execution causes some state (e.g. memory) to be updated as a specific function of current state
  4. program counter is advanced (according to instruction)
  5. repeat

* atomic
* sequential
* inorder
An Early ISA: EDSAC

- Single accumulator architecture, i.e. $\text{ACC} \leftarrow \text{ACC} \oplus \text{M}[n]$
- Instruction examples
  - $\text{A n}$: add $\text{M}[n]$ into $\text{ACC}$ (also $\text{S}, \text{R}, \text{L}$)
  - $\text{T n}$: transfer the contents of $\text{ACC}$ to $\text{M}[n]$ and clear
  - $\text{E n}$: If $\text{ACC} > -1$, branch to $\text{M}[n]$ or proceed serially
  - $\text{I n}$: Read the next character from paper tape, and store it as the least significant 5 bits of $\text{M}[n]$
  - $\text{Z}$: Stop the machine and ring the warning bell

Notice: only absolute addressing mode in memory access and control transfer

Let’s try some basic things

- Function call

```
......
a: Ef
......
......
b: Ef
......
f: ......
......
......
E a+1
```

- Array access in a loop

```
......
a: ......
......
......
......
E a
......
```

What is the hacker way?
What is the proper fix?
Evolution of Register Architecture

◆ Accumulator
- a legacy from the “adding” machine days
  Ever wonder about that “AC” button on your calc?
◆ Accumulator + address registers
- need register indirection
- initially address registers were special-purpose,
  i.e., can only be loaded with an address for indirection
- eventually arithmetic on address registers
◆ General purpose registers (GPR)
- all registers good for all purposes
- grew from a few registers to 32 (common for RISC) to
  128 in Intel Itanium

What drove the changes?

Operand Sources?

◆ Number of Specified Operands
  Niladic Op (e.g. Burroughs)
  Monadic OP in2 (e.g. EDSAC)
  Dyadic OP inout, in2 (e.g. IBM 360)
  Triadic OP out, in1, in2 (e.g. MIPS)

◆ Can ALU operands be in memory?
  Yes! e.g. x86/VAX/“CISC”
  No! e.g. MIPS/“RISC”/load-store architectures

◆ How many different formats and addressing modes?
  a very few e.g. MIPS / “RISC”
  a lot e.g. x86
  everything goes e.g. VAX
Memory Addressing Modes

- **Absolute**
  LW rt, 10000
  use immediate value as address

- **Register Indirect**
  LW rt, \( r_{\text{base}} \)
  use GPR[\( r_{\text{base}} \)] as address

- **Displaced or based**
  LW rt, offset(\( r_{\text{base}} \))
  use offset+GPR[\( r_{\text{base}} \)] as address

- **Indexed**
  LW rt, \( (r_{\text{base}}, r_{\text{index}}) \)
  use GPR[\( r_{\text{base}} \)]+GPR[\( r_{\text{index}} \)] as address

- **Memory Indirect**
  LW rt \( ((r_{\text{base}})) \)
  use value at \( M[GPR[r_{\text{base}}]] \) as address

- **Auto inc/decrement**
  LW Rt, \( r_{\text{base}} \)
  use GPR[\( r_{\text{base}} \)] as address, and inc. or dec. GPR[\( r_{\text{base}} \)]

- Anything else you like to see ......

VAX-11: ISA in mid-life crisis

- First commercial 32-bit machine considered an important milestone

- Ultimate in “orthogonality” and “completeness”
  All of the above addressing modes x \{ 7 integer and 2 floating point formats\} x \{more than 300 opcodes\}

- Opcode in excess
  - 2-operand and 3-operand versions of ALU ops
  - INS/(REM)QUE (for circular doubly-linked list)
  - “polyf": 4th-degree polynomial solve

- Encoding
  addl3 r1,737(r2),(r3)[r4] 7-byte, sequential decode

The first VAX11-780 was installed at CMU!!
"RISC"

- Simple operations
  - 2-input, 1-output arithmetic and logical operations
  - few alternatives for accomplishing the same thing
- Simple data movements
  - ALU ops are register-to-register (need a large register file)
  - “load-store” architecture, 1 addressing mode
- Simple branches
  - limited varieties of branch conditions and targets
- Simple instruction encoding
  - all instructions encoded in the same number of bits
  - few, simple encoding formats

An ISA philosophy motivated by and intended for compilers rather than assembly programmers

Evolution of ISAs

- Why were the earlier ISAs so simple? e.g., EDSAC
  - technology
  - precedence
- Why did it get so complicated later? e.g., VAX11
  - assembly programming
  - lack of memory size and performance
  - microprogrammed implementation
- Why did it become simple again? e.g., RISC
  - memory size and speed (cache!)
  - compilers
- Why is x86 still so popular?
  - technical merit vs. {SW base, psychology, deep pocket}
- Why has ARM thrived while other RISC ISAs decimated
- Why RISC-V?
Intel IA-64/Itanium Architecture

- Late 90’s attempt to counter RISC in servers market
- IA-64 Instruction “Bundle”
  - three IA-64 instructions (aka syllables)
  - template bits specify dependencies within a bundle and between bundles
  - group=collection of dependence-free bundles

A efficient and flexible way to encode instruction parallelism explicitly

<table>
<thead>
<tr>
<th>inst₁</th>
<th>inst₂</th>
<th>inst₃</th>
<th>temp</th>
</tr>
</thead>
</table>

- “Thin” abstraction to make hardware fast and simple
  - shift from dynamic HW to compiler static analysis and/or profile-driven
  - expose inst-by-inst performance mechanisms to SW
  - very hard to produce high performing code by hand

Example: Rotating Registers

- 128 general purpose physical integer registers
- Register names R0 to R31 are static; refer to the first 32 physical GPRs
- Register names R32 to R127 are “rotating registers”; renamed onto the remaining 96 physical registers by an offset
- Simplifies register use on function call/return and on loop optimizations (when register names are reused in code)
Example: Predicated Execution

- 64 one-bit predicate register file
  - each instruction carries a 6-bit predicate operand field
  - instruction has no effect if predicate operand is false
- A way to realize conditionals without control flow

Good tradeoff when (1) very high cost in control flow, and
(2) an excess of resources to absorb the “extra” work

Example: Exposed Memory Hierarchies

- ISA included the concept of cache hierarchy
  - multiple levels
  - separate “temporal” vs “non-temporal”
- Memory instructions give hints where best to cache
  As hints, microarchitecture does not have to comply
Intel iAPX 432, circa 1980

- Was to be a follow-on to 8080
  contemporary development with RISC and x86
- Way off the beaten path
  - bit-aligned variable length instruction objects
  - stack instead of GPRs
- High-level abstraction in HW
  - object-oriented (down to instructions)
  - capability-based memory protection
  - HW garbage collection
- Architectural vision too aggressive for technology and commercial demand of the time

How much should ISA still matter?
Binary Translation

- Generate a new executable in a **target ISA** with same functional behavior as the original in a **source ISA**
  - not the same as interpretation or VM
  - not easy but doable (for the right source and target)
  - static vs dynamic
- Holy grail
  - all software run on the ISA/processor I sell
  - all processors can run the software I sell
- “Architecture” need not be the HW/SW contract
  - binary compatibility by translation virtualization
  - ISA and processor can become commodity (like DRAM) and faster to evolve
  - old software and ISA can live on for ever (and get faster overtime)

What is CUDA?

Transmeta Crusoe & Code Morphing

<table>
<thead>
<tr>
<th>Complete x86 Abstraction</th>
<th>x86 applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86 OS</td>
<td>x86 OS</td>
</tr>
<tr>
<td>x86 BIOS</td>
<td>x86 BIOS</td>
</tr>
</tbody>
</table>

Code Morphing
Dynamic Binary Translation

Crusoe VLIW Processor
(**with superset of x86 ISA state)**

- Crusoe boots “Code Morpher” from ROM at power-up
- Crusoe+Code Morphing == x86 processor
  x86 software (including BIOS) cannot tell the difference

BTW, this really worked in the early 2000s
Code Morphing Software (CMS)

- Begins execution at power-up
  - fetches first-time x86 basic block from memory
  - translates basic block into Crusoe VLIW and caches the translation for reuse
  - jumps to the generated Crusoe code for execution
  - continue directly from block to block if translation already cached; CMS regains control on new basic blocks
  
  basic block with “unsafe” x86 instructions not translated

- Re-optimize a translated block after collecting profiling information

- The only native SW for Crusoe ISA
  
  Crusoe processors do not need to be binary compatible between generations

Not really so different from Intel’s own
von Neumann abstraction is not free

- Significant transistor and energy overhead in presenting the simplifying abstraction
  - per-instruction access to program memory
  - dataflow through reading/writing of registers and memory state
  - need to unwind sequentiality and atomicity
- In fact, von Neumann processors are mostly overhead
  - ~90% energy [Hameed, ISCA 2010, Tensilica core]
  - ~95% energy [Balfour, CAL 2007, embedded RISC]
- ISA future?
  - move away from von Neumann as doctrine?
  - do away with ISAs (lower-level, more explicit HW)?

A lot depends on what languages and compilers can do

What if you had to design an ISA?
Architecture*

- “The term architecture is used here to describe the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flow and controls, the logical design, and the physical implementation.”


Open-Ended Design: extrapolation and anticipation

“a dependable base for a decade of customer planning and customer programming, and continuing laboratory development...” Typical life expectancy 15~20 years

- “Asynchronous” operation of components
  - abstract out exact time, performance etc. to allow changing technology and relative speed of components
  - Note: this doesn’t say to build them as asynchronous logic

- Parameterization of storage capacity, multi CPU, multi I/O, etc.

- Permit future extensions by “reserving” spare bits in instruction encoding

- Standard interfaces for expansion sub-systems

[Amdahl, Blaauw and Brooks, 1964]
General Purpose

- General Purpose = effective support for “large and small, separate and mixed applications” in many domains (e.g., commercial, scientific, real-time....)
- How
  - code-independent operation
    - no special interpretation of bit pattern in data
      - e.g. ASCII character has no special significance per se
    - except where essential
      - e.g., integer, floating point, etc.
  - support full generality of logic manipulation on bit and data entities
  - fine-grain memory addressability (down to small units of bits)

[Amalgh, Blaauw and Brooks, 1964]

Inter-Model Compatibility

- Strict program compatibility = “a valid program whose logic will not depend implicitly upon time of execution and which runs upon configuration A, will also run on configuration B if the latter includes at least the required storage, at least the required I/O devices ....”
- Invalid programs [...] are not constrained to yield the same result
  - “invalid program” means a program that violates the architecture manual and not that it generates exceptions
  - exceptional conditions are part of the architecture
- By virtualization of logical structures and functions
- The King of Binary Compatibility: Intel x86, IBM 360
  - software base
  - performance scalability

[Amalgh, Blaauw and Brooks, 1964]
Curio: Single-Instruction ISAs

- What is the simplest single instruction ISA that is Turing-complete?

- E.g.
  
  Subtract-Branch-If-Negative a1 a2 dest
  
  $M[a1] \leftarrow M[a1] - M[a2]$
  
  if $M[a1] < 0$ then goto $M[\text{dest}]$
  
  else go to next

Universality check list
conditioned branch
memory read/write
universal logic function