Comparative architecture
COMSM1302 Overview of Computer Architecture

John Lapinskas, University of Bristol

What to look for in a new ISA

- Size of words in memory. (A “64-bit” CPU means a 64-bit word size.)
- Address space of memory.
- Instruction length (normally a multiple of word size, may be variable!)
- Design philosophy: Harvard vs von Neumann, RISC vs CISC.
- Registers.
- Addressing modes.
- Available arithmetic, logical, and branching operations.
- Hardware interrupts.
- The stack (covered later in the unit).

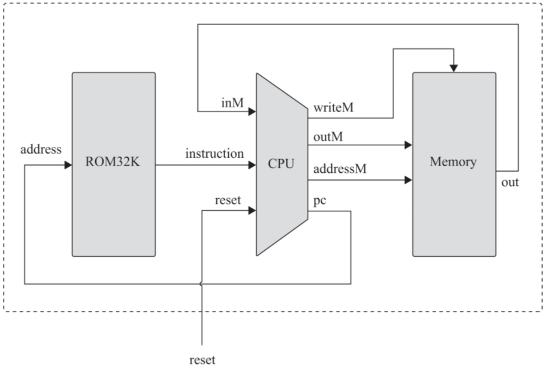
What to look for in a new ISA

- Size of words in memory. (A "64-bit" CPU means a 64-bit word size.)
- Address space of memory.
- Instruction length (normally a multiple of word size, may be variable!)
- Design philosophy: Harvard vs von Neumann, RISC vs CISC.
- Registers.
- Addressing modes.
- Available arithmetic, logical, and branching operations.
- Hardware interrupts.
- The stack (covered later in the unit).

Note that most assembly languages use a slightly different-looking syntax to Hack, putting the operand first. For example, in MIPS, the Hack command `A=D+M` would be written `add A,D,M`.

This is cosmetic — one line of assembly still corresponds to one machine code instruction. Hack's syntax is because it only has two instructions!

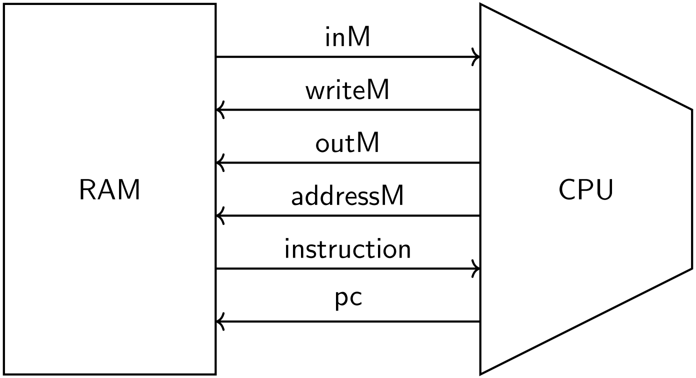
Harvard vs von Neumann



Source: Nisan and Schocken

Hack reads instructions from ROM and stores data in RAM. Having two separate memory banks makes it an example of **Harvard architecture**.

Harvard vs von Neumann



Most modern ISAs instead follow **von Neumann architecture**, storing both instructions and data in the same memory, which is RAM.

(The main advantage of Harvard architecture is simpler hardware!)

RISC vs CISC

Reduced Instruction Set Computing (**RISC**)	**Complex Instruction Set Computing** (**CISC**)
Requires simple microarchitecture	Requires complex microarchitecture
Assembly uses many simple instructions	Assembly uses few complex instructions
Less efficient memory use	More efficient memory use
Fixed-length instructions	Variable-length instructions
Instructions mostly take 1 cycle each	Instructions take any number of cycles
ISA has “standard” features only	ISA has “extra” features (e.g. execution modes)

RISC and CISC are not absolute or rigorous definitions, but two ends of a spectrum. Some ISAs are more RISC-like, and others are more CISC-like.

Usually for modern CPUs, CISC runs code faster than RISC. RISC is still widely-used for low-power, low-cost applications (e.g. embedded hardware).

Historically, this has gone back and forth — in 2000, RISC was better for high-speed applications as well.

Registers

In Hack:

- PC is the programme counter;
- M is the memory register;
- A the address register that controls M.

All these are **special-purpose registers**. They have special roles in the hardware and limited access to ALU operations. Typically these are ISA-specific — e.g. non-Hack ISAs don't have equivalents of A and M.

Almost all ISAs have a PC, but many call it the **instruction register** (**IR**).

Registers

In Hack:

- *PC* is the programme counter;
- *M* is the memory register;
- *A* the address register that controls *M*.

All these are **special-purpose registers**. They have special roles in the hardware and limited access to ALU operations. Typically these are ISA-specific — e.g. non-Hack ISAs don't have equivalents of *A* and *M*.

Almost all ISAs have a *PC*, but many call it the **instruction register** (**IR**).

D is the opposite, a **general-purpose register**. These can fill any role in ALU operations, and most architectures have 32 or more.

All general-purpose registers in an ISA behave the same way, and they're direct analogues of *D*.

Addressing modes

An **addressing mode** defines how an ISA maps the operands of an instruction to data. Hack has the three most common:

- **Immediate addressing** interprets the operand as data.
  - `@511` $\leftrightarrow$ `0x01FF` interprets the operand 511 as the number 511.
  - In ARM7, `LDR R0, #0xDEADBEEF` loads the value `0xDEADBEEF` into register `R0`.

Addressing modes

An **addressing mode** defines how an ISA maps the operands of an instruction to data. Hack has the three most common:

- **Immediate addressing** interprets the operand as data.
- **Direct addressing** interprets the operand as the data's location.
    - `D=A+D` $\leftrightarrow$ `0xE090` interprets the `comp` operand to mean that the values to add are A and D, and the `dest` operand to mean that the result should be stored in D.
    - In ARM7, `LDR R0, 0xDEADBEEF` loads the value stored at memory address `0xDEADBEEF` into register `R0`.

Addressing modes

An **addressing mode** defines how an ISA maps the operands of an instruction to data. Hack has the three most common:

- **Immediate addressing** interprets the operand as data.
- **Direct addressing** interprets the operand as the data's location.
- **Indirect addressing** interprets the operand as the location of a *pointer* to the data.
    - `M=D+1` $\leftrightarrow$ `0xE7C8` interprets the `dest` operand to mean that $D+1$ should be stored at the memory address stored in A. (The $D+1$ part of the instruction is an example of direct addressing.)
    - In ARM7, `LDR R0, [R1]` reads register `R1`, reads the memory at that address, and stores the result in `R0`.

Addressing modes

An **addressing mode** defines how an ISA maps the operands of an instruction to data. Hack has the three most common:

- **Immediate addressing** interprets the operand as data.
- **Direct addressing** interprets the operand as the data's location.
- **Indirect addressing** interprets the operand as the location of a *pointer* to the data.

CISC ISAs often have other addressing modes for increased efficiency. For example, ARM7 supports **indexed indirect addressing**.

Given the instruction `LDR R0, [R1, #0xBEEF]`, ARM7 will read the address stored in `R1`, add `0xBEEF` to it, then store the value stored at *that* memory address in `R0`.

This is useful for arrays — if `R1` is the address of an array `arr` in C, then `LDR R0, [R1, #0xBEEF]` stores `arr[0xBEEF]` in `R0` in one instruction!

Common ISAs: x64

Source: PCMag (Pictured: AMD Ryzen 9 7900 with fan.)

Most modern 64-bit desktop and laptop computers run the **x64** ISA, a.k.a. **x86-64** and **AMD64**.

(Intel did develop their own 64-bit architecture, **IA-64** a.k.a. **Itanium**, but it was discontinued in 2019 and they now use x64 too.)

x64 is far towards CISC. CPUs running it are typically very fast and efficient, but also expensive, power-hungry, and need a lot of cooling.

Common ISAs: ARM

Source: Apple (Pictured: iPhone 13)

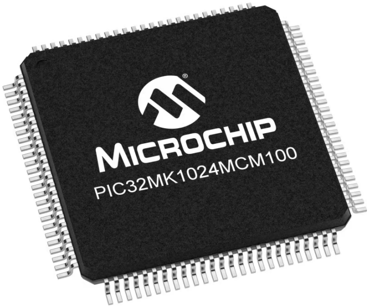
ARM is a family of 64-bit and 32-bit ISAs geared towards portable use.

ARM is further towards RISC than x64. Chipsets using it usually have lower power and cooling requirements, but are slower and less efficient.

Almost all modern mobiles and tablets use ARM. Modern Macs also use a variant of ARM developed by Apple for higher-power applications.

The company responsible, Arm Holdings, is actually based in Bristol!

Common ISAs: MIPS

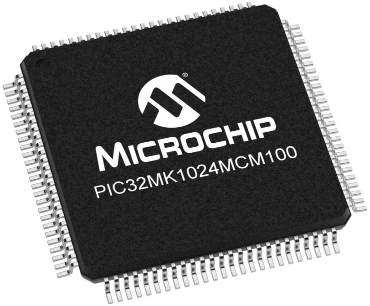


Source: Microchip Technology Incorporated

MIPS is a family of simple RISC ISAs used in embedded applications.

For example, PIC chips (pictured) typically cost £3–£10 per unit, run at 15–120MHz, and at the low end have power drain comparable to a LED.

Common ISAs: MIPS



Source: Microchip Technology Incorporated

Source: Super Mario Wiki
(Screenshot from Super Mario 64)

MIPS is a family of simple RISC ISAs used in embedded applications.

For example, PIC chips (pictured) typically cost £3–£10 per unit, run at 15–120MHz, and at the low end have power drain comparable to a LED.

The Nintendo 64 ran a variant of MIPS as well — hence the bunny's name!

Advanced feature: Hardware interrupts

Recall Hack input works by polling — reading RAM[`0x6000`] returns a value representing the key being pressed at that moment (if any).

Advanced feature: Hardware interrupts

Recall Hack input works by polling — reading RAM[`0x6000`] returns a value representing the key being pressed at that moment (if any).

Problem: Polling sucks! It wastes a huge number of clock cycles and (at lower clock speeds) risks missing inputs.

Advanced feature: Hardware interrupts

Recall Hack input works by polling — reading RAM[0x6000] returns a value representing the key being pressed at that moment (if any).

Problem: Polling sucks! It wastes a huge number of clock cycles and (at lower clock speeds) risks missing inputs.

Solution: All modern ISAs use **interrupts** for input:

- The CPU has one or more dedicated pins for an interrupt signal on the hardware level, e.g. from a key being pressed.
- On receiving an interrupt signal, the CPU stops what it's doing, saves its current PC value, and immediately branches to a new location containing code to deal with that interrupt.
- After handling the input, the CPU restores its original PC value and picks up execution where it left off.

Advanced feature: Hardware interrupts

Recall Hack input works by polling — reading RAM[0x6000] returns a value representing the key being pressed at that moment (if any).

Problem: Polling sucks! It wastes a huge number of clock cycles and (at lower clock speeds) risks missing inputs.

Solution: All modern ISAs use **interrupts** for input:

- The CPU has one or more dedicated pins for an interrupt signal on the hardware level, e.g. from a key being pressed.
- On receiving an interrupt signal, the CPU stops what it's doing, saves its current PC value, and immediately branches to a new location containing code to deal with that interrupt.
- After handling the input, the CPU restores its original PC value and picks up execution where it left off.

Interrupts are also used for other things, like attempts to access protected memory (a.k.a. why you get segfaults when your pointers go wrong).