The instruction set architecture (ISA)

The different aspects of ISA design

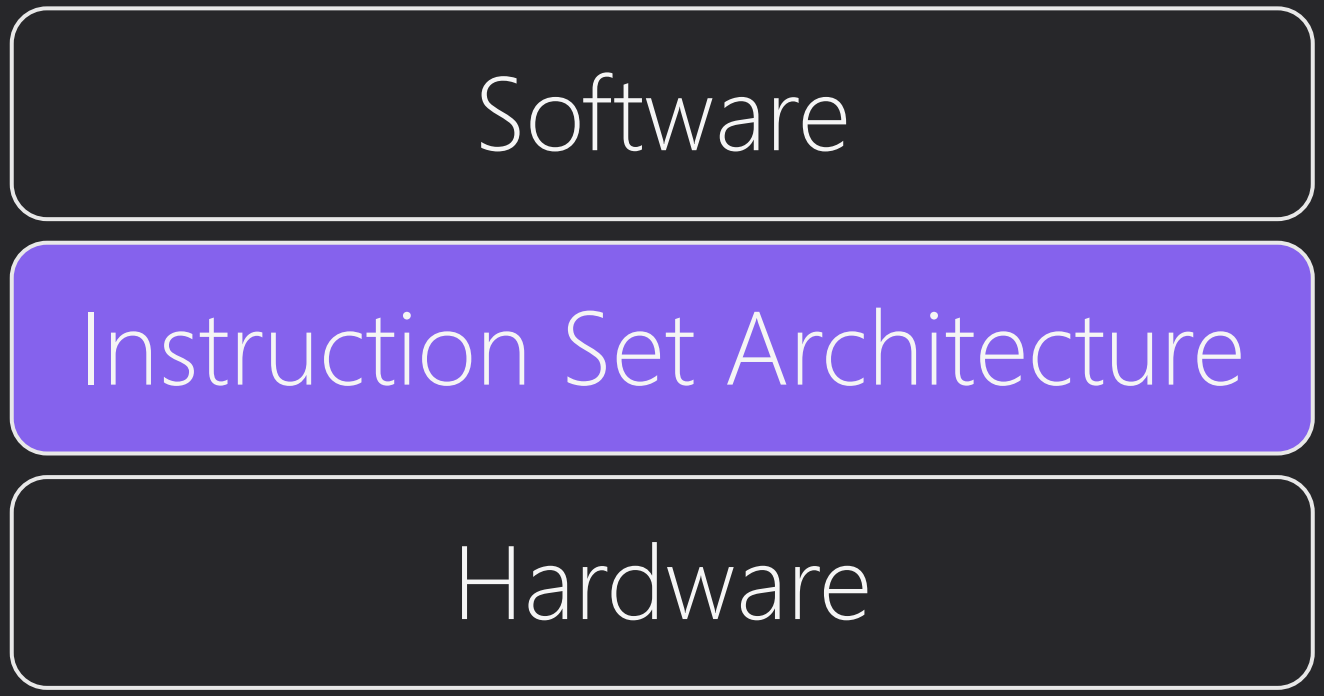
Introduction

From instruction to ISA

ISA Design

Conclusion

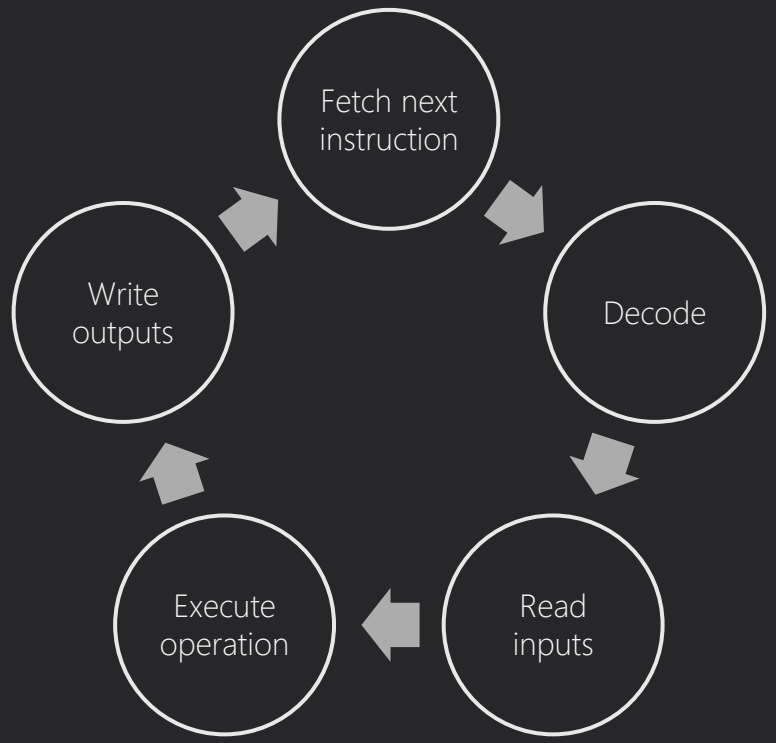
An ISA is an abstraction for computation



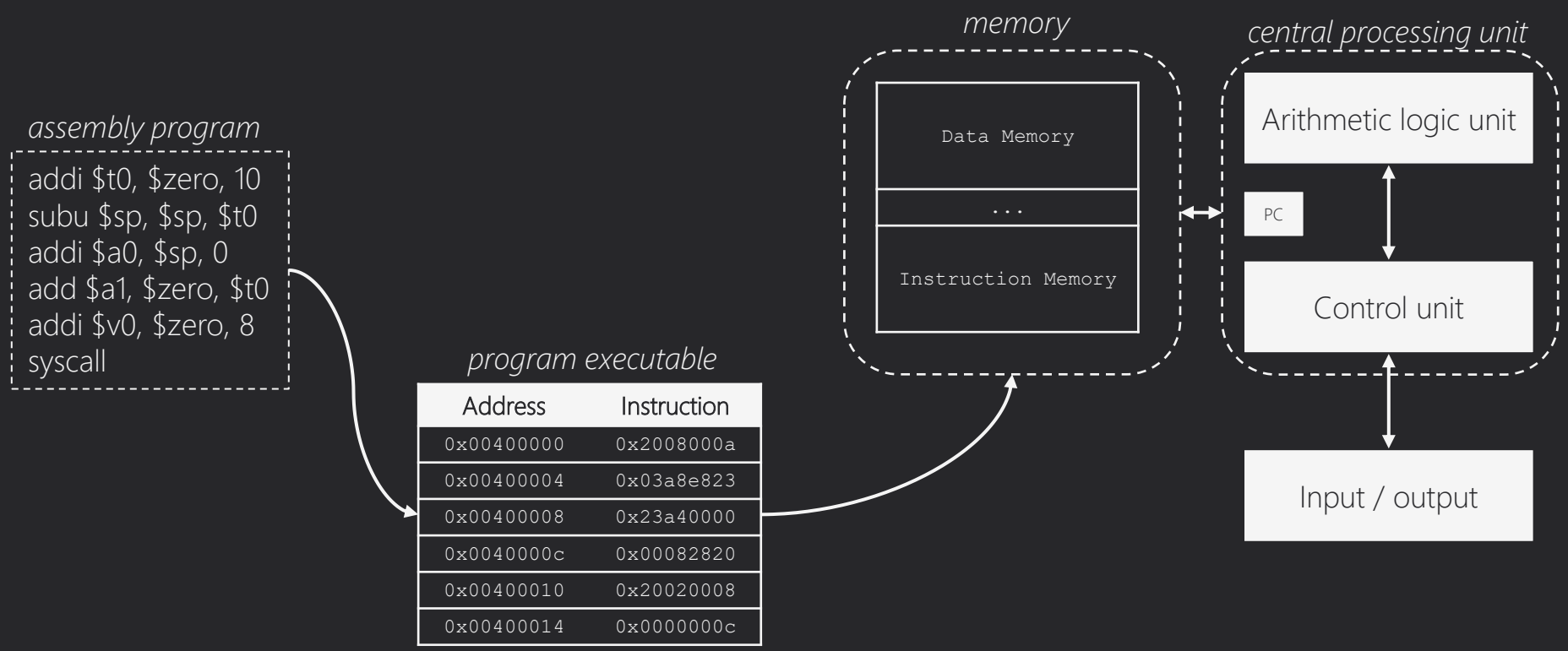
- Many different high-level languages
- Only a few major ISAs
- Many different processors on the market

The sequential state model

- The model of all major ISAs
- Processors repeat the loop (right)
- State updates are sequential
  - Instruction X finishes before X + 1
  - Matches a programmer's mental model of when memory is updated



The stored program concept



What's next

- An iterative build-up from instructions to instruction set to ISA
  - The different aspects of an ISA

- A brief comparison and look at some design decisions from existing instruction set architectures

The instruction

The instruction set

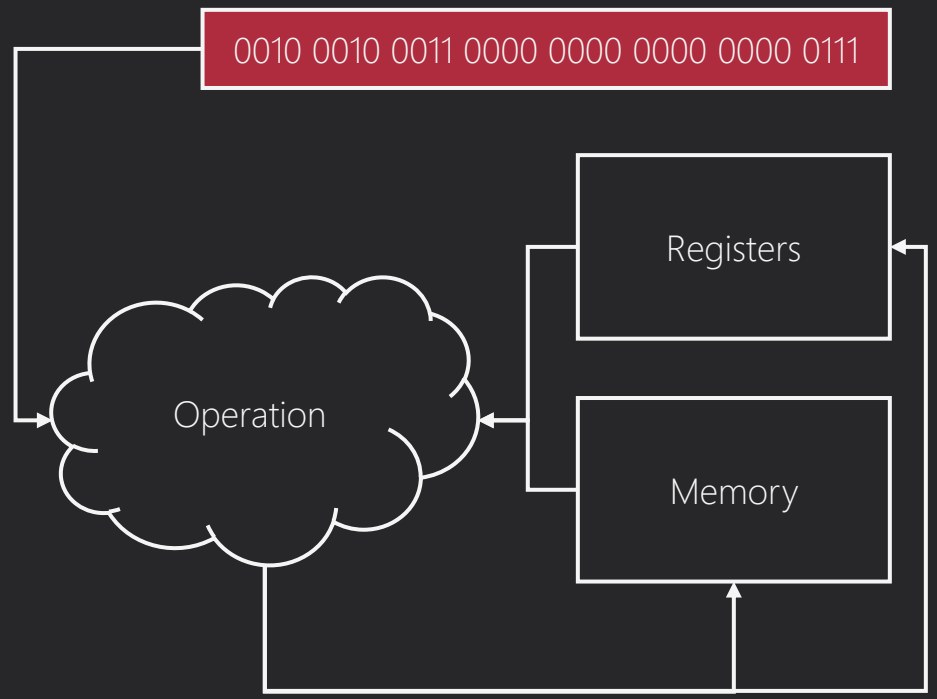
The instruction set architecture

From instruction to ISA

Some of the easier-to-see parts of an ISA

The instruction

- An *instruction* defines the
  - operation performed by the hardware
  - operands to use for that operation
- Operations mutate state
- An *operand* is an input to the operation
  - Can be a register, memory location, or immediate
  - An immediate value is data encoded into the instruction itself



The instruction set

- Simply: a set of instructions
- The instructions can be grouped by the types of operation they perform
- A subset of the instruction set is the "common case"
  - i.e., the subset makes up most of the instructions executed by software

Operation type	*MIPS examples*
Arithmetic	`add, addi, addu`
Logical	`and, or`
Control	`beq, jal`
Data transfer	`lw, sw, lui`
System	`syscall`
Floating point	`add.s, div.s`

Operand sizes (and types)

Common to most ISAs

- Sizes
  - Byte (8 bits)
  - Half word (16 bits)
  - Word (32 bits)
  - Double word (64 bits)
- Types
  - Single-precision floating point (32 bits)
  - Double-precision floating point) (64 bits)
  - Signed integers use 2's complement

Not-so-common examples

- Character strings
  - Each character is 1 byte (not so useful with Unicode)
- "Decimal" formats
  - Because not all fractions can be represented exactly in floating point

The instruction set architecture

Aspects

- Operations (earlier slide)
- Data types (earlier slide)

- Instruction format
- Operand model
- Addressing
- Control

Design philosophies

- *Reduced Instruction Set Computer* (RISC)
  - A "small" set of instructions
  - Instructions perform "simple" operations
  - A focus on the common case
- *Complex Instruction Set Computer* (CISC)
  - A large set of instructions
  - Instructions perform both simple and complex (for hardware) operations
  - Support for the uncommon case

Instruction format

Length and encoding

- Fixed-length
  - e.g., MIPS: 4 bytes
  - Three types of encodings (R-, I-, and J-type)
- Variable-length
  - e.g., x86: 1 to 15 bytes
- Hybrid
  - e.g., Arm's Thumb2: 2 or 4 bytes

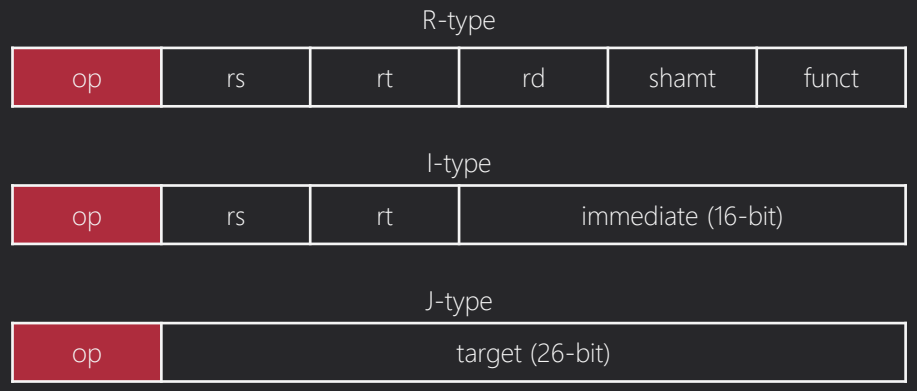
Impact

- Program size
  - Important in embedded systems
  - Fixed-length does poorly here
- Hardware complexity
  - Hardware must decode instructions before it can execute them
  - Variable-length does poorly here

MIPS instruction formats

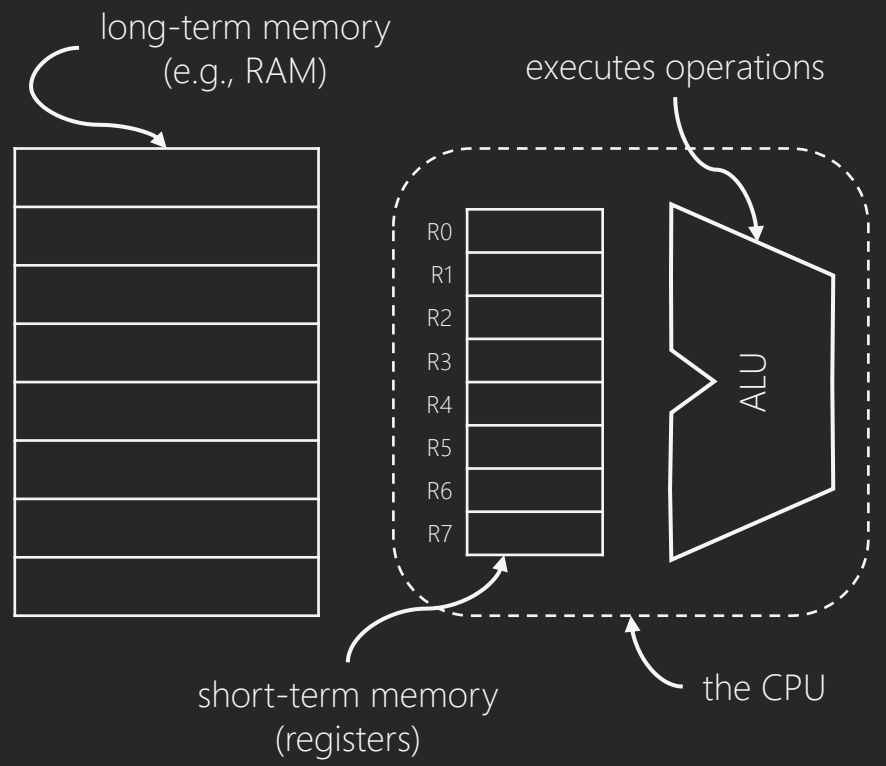
- Three formats
  - R-type (operations only use registers)
  - I-type (operations use an immediate)
  - J-type (for “jump” operations)

- Notice the regularity
  - op for all formats
  - rs and rt for R- and I-type



Classifying ISAs by operand model

- Modern ISAs define a set of general-purpose registers for "internal storage"
- A *register-memory architecture* has instructions that operate on memory or registers
  - e.g., an add operation may have one operand in memory and another in a register
- A *register-register architecture* only operates on registers
  - Data in memory must be transferred to/from registers with dedicated instructions
  - Also called a *load-store architecture*



The set of general-purpose registers

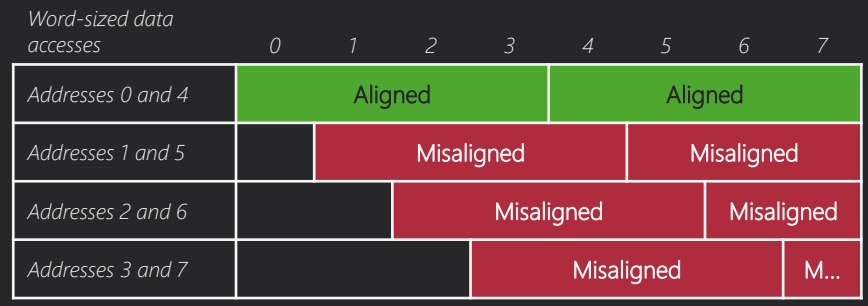
- MIPS defines thirty-two 32-bit registers
  - With conventions on how to use those registers
  - A register can be uniquely identified with 5 bits ($2^5 = 32$)

- The 32-bit version of the x86 architecture (IA-32) defines eight 32-bit general-purpose registers

Name	Address	Description
$0	0	The value 0
$at	1	A temporary value for the "assembler"
$v0-$v1	2-3	Return values for procedure calls
$a0-$a3	4-7	Arguments for procedure calls
$t0-$t7	8-15	Temporary variables
$s0-$s7	16-23	"Saved" variables
$t8-$t9	24-25	More temporary variables
$k0-$k1	26-27	Temporary variables for operating systems
$gp	28	The global pointer
$sp	29	The stack pointer
$fp	30	The frame pointer
$ra	31	The return address for procedure calls

Interpreting memory addresses

- Modern ISAs use byte-addressable memory
  - 32-bit addresses (old), 64-bit addresses (today)
- Byte-addressable memories can be *big-endian* or *little-endian*
  - i.e., byte 0 of a word contains different data on a big-endian and little-endian machine
- Memory accesses to data larger than 1 byte must (typically) be aligned
  - e.g., accesses to words (4 bytes) must use (or are preferred to use) addresses that are a multiple of 4



Addressing modes

- An *addressing mode* defines how operands are identified
- Addressing modes specify constants, registers, and locations in memory

MIPS addressing mode	The instruction specifies…	Example
Register-only	Only register addresses.	[rd] = [rs] + [rt]
Immediate	A value (not an address).	[rt] = [rs] + immediate
Base-displacement	A displacement from an address that is in a register.	address = [rs] + immediate [rt] = mem[address]
PC-relative	A value relative to the address in the PC.	address = [PC] + imm

Conditional execution (control)

Testing for conditions

- Compare-and-branch
  - e.g., MIPS' branch-if-equal (`beq`)
  - e.g., MIPS' branch-if-greater-than-zero (`bgtz`)
- Condition register
  - e.g., MIPS' set-less-than (`slt`) instruction
  - Must then use a branch instruction like `beq` or `bne`
- Condition code
  - A condition code register is set based on a previous operation
  - Branch based on value of condition code

Computing target addresses

- Need to calculate the target address to store in the PC
- Example addressing modes
  - PC-relative
  - Direct (or pseudo-direct)
  - Indirect

Well-designed ISAs

A brief comparison

x86 and implementation

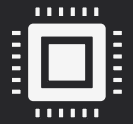
ISA Design

The driving forces behind the design of ISAs

Aspects of a well-designed ISA

Software: easy to express computation

Compiler complexity



Hardware: easy to implement

Trade-offs: performance, power, cost

Future-proof: easy to maintain compatibility

Trends in software

Trends in technology

A brief ISA comparison

- Like software, ISAs have "versions"
  - e.g., Armv8-R is a version of Arm64

Name	Operand model	Encoding	Control	General-purpose registers
MIPS32	register-register	• Fixed (32-bit)	• Compare and branch	32 32-bit
x86 (64-bit)	register-memory	• Variable (1 to 15 bytes)	• Condition code	16 64-bit
Arm32	register-register	• Fixed (32-bit)	• Condition code • Compare-and-branch	15 32-bit
Arm64	register-register	• Hybrid (32- or 64-bit)	• Condition code • Compare-and-branch	31 64-bit
RISC-V	register-register	• Fixed (base) • Variable (extensions)	• Compare-and-branch	32 64-bit

Simplifying implementation through translation

Problem and solution

- Due to its history, x86 has a set of difficult-to-implement instructions
  - Can't remove (compatibility)
- Intel's solution? "Complex" decode hardware to simplify the rest of the implementation

Example

- Original instruction:

```
push $eax                # push [eax] onto the stack
```

- Hardware translation (after decode):

```
addi $esp, $esp, -4      # make space on the stack
sw $eax, 0($esp)         # put [eax] in allocated space
```

How different are modern ISAs?

Commonalities

- Data type support
- General-purpose registers
- Support for frequently-used operations (e.g., add, multiply)
- Memory alignment restrictions

Differences

- Instruction format
- Design philosophies and historical inertia

Conclusion

Final questions and thoughts

Why define an instruction set architecture?

- One architecture has *many* different hardware implementations
  - e.g., Intel vs. AMD processors
  - e.g., Apple's M1 vs. M2 processors
- Software compiled for one architecture can run on many different processors
- Wait… What about the operating system?

Are ISAs only implemented in hardware?

- No, an ISA is just an interface
  - Can be implemented in software
  - Can be implemented in a mixture of hardware and software

- Many emulators (software) exist, especially for old architectures