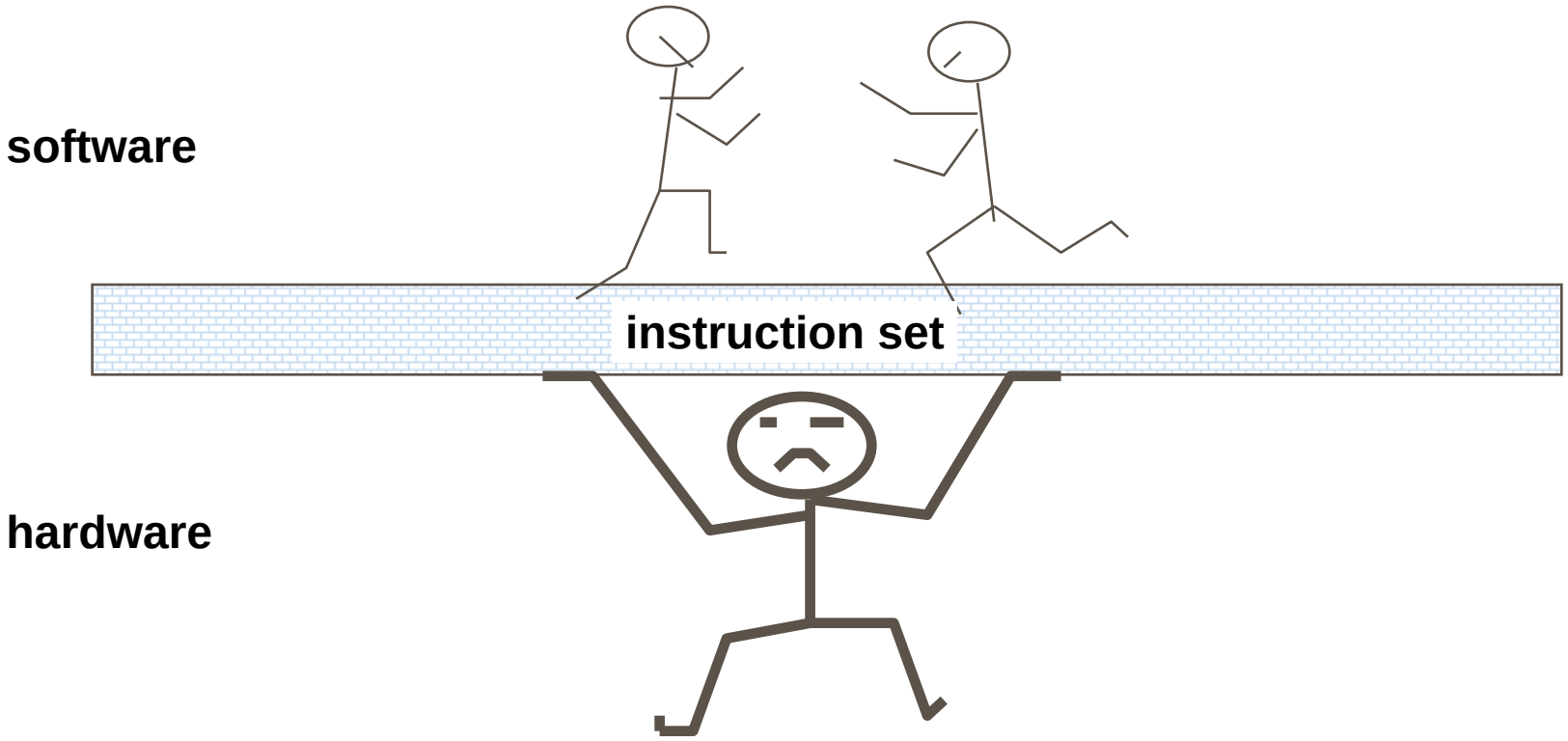
Instruction Set Architecture

Instruction Set Architecture (ISA)

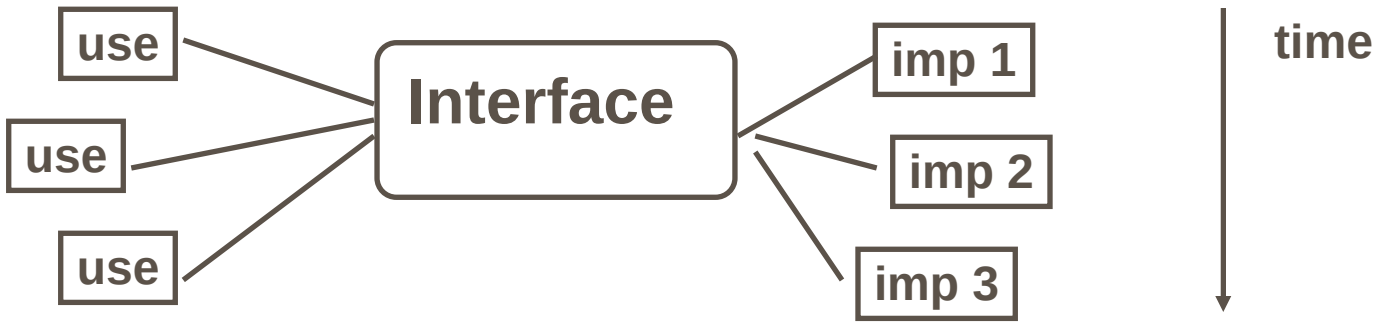


ISA is the interface between the hardware and software. The Instruction Set Architecture is that portion of the machine visible to the assembly level programmer or to the compiler writer.

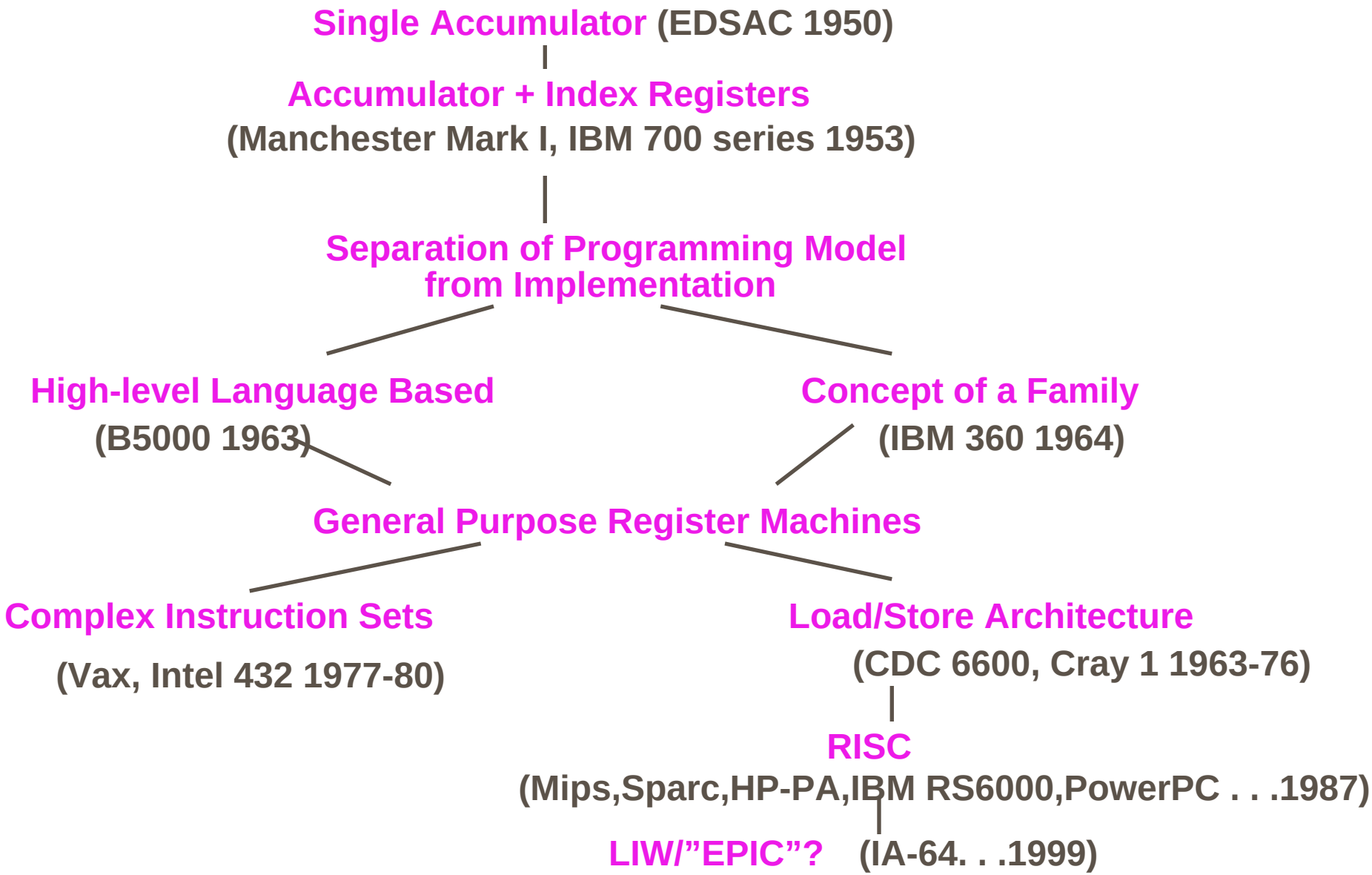
Interface Design

A good interface:

- Lasts through many implementations (portability, compatibility)
- Is used in many different ways (generality)
- Provides **convenient** functionality to higher levels
- Permits an **efficient** implementation at lower levels



Evolution of Instruction Sets



Evolution of Instruction Sets

- Major advances in computer architecture are typically associated with landmark instruction set designs
  - Ex: Stack vs. GPR (System 360)
- Design decisions must take into account:
  - Technology
  - Machine organization
  - Programming languages
  - Compiler technology
  - Operating systems
- And they in turn influence these

Classifying Instruction Set Architectures

The major choices which differ basically by the type of Internal Storage in a processor are :

- Stack
- Accumulator
- General-purpose register (GPR)
  - Register–Memory
  - Register-register/load-store
  - Memory – Memory (not used now)

Classification is based on:

- Number of memory operands in an ALU operation
- Total number of operands in an ALU operation

Comparison of ISAs

Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R1,B	Load R2,B
Add	Store C	Store C, R1	Add R3,R1,R2
Pop C			Store C,R3

Code sequence for (C = A + B) for four classes of instruction sets.

Registers are the class that win. The more registers on the CPU, the better.

GPR Architectures

Number of memory addresses	Maximum number of operands allowed	Type of Architecture	Examples
0	3	Register-register	MIPS, SPARC
1	2	Register-memory	IBM360/370, Intel 80x86
2	2	Memory-memory	VAX
3	3	Memory-memory	VAX

Advantages and Disadvantages of the 3 most common types of GPR computers

Type	Advantages	Disadvantages
Register-register (0, 3)	Simple, fixed-length instruction encoding. Simple code generation model. Instructions take similar numbers of clocks to execute	Higher instruction count than architectures with memory references in instructions. More instructions and lower instruction density leads to larger programs.
Register-memory (1, 2)	Data can be accessed without a separate load instruction first. Instruction format tends to be easy to encode and yields good density.	Operands are not equivalent since a source operand in a binary operation is destroyed. Encoding a register number and a memory address in each instruction may restrict the number of registers. Clocks per instruction may vary by operand location.
Memory-memory (2, 2) or (3, 3)	Most compact. Doesn't waste registers for temporaries.	Large variation in instruction size, Especially for three-operand instructions. In addition, large variation in work per instruction. Memory accesses create memory bottleneck. (Not used today.)

Features to be considered while designing the ISA

- **Types of instructions (Operations in the Instruction set)**
- **Types and size of operands**
- **Addressing Modes**
- **Addressing Memory**
- **Encoding and Instruction Formats**
- **Compiler related issues**

Types of instructions

Guidelines for choosing the types of instructions:

- Choose commonly used instructions
- Make them faster
- Specialized instructions depending upon the application of the processor
  - Eg. **Media and Signal Processing** which involve operations like
    - Partitioned add instructions
    - Single-instruction multiple-data (SIMD) or vector instructions
    - Paired single operations
    - Saturating arithmetic operations
    - Multiple-accumulate (MAC) instructions

Categories of instruction operators and examples

Operator Type	Examples
Arithmetic and logical	Integer arithmetic and logical operations: add, subtract, and, or, multiply, divide
Data transfer	Loads-stores (move instructions on computers with memory addressing)
Control	Branch, jump, procedure call and return, traps
System	Operating system call, virtual memory management instructions
Floating point	Floating-point operations:add, multiply, divide, compare
Decimal	Decimal add, decimal multiply, decimal-to-character conversions
String	String move, string compare, string search
Graphics	Pixel and vertex operations, compression/decompression operations

Top 10 instructions for Intel 80x86

Rank	80x86 Instruction	Integer average (% total executed)
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
Total		96%

Type and Size of operands

Encoding in the opcode, designates the type of an operand.

Common operand types and their sizes are:

- **Character (8 bits)**
- **Half word (16 bits)**
- **Word (32 bits)**
- **Single Precision Floating Point (1 Word)**
- **Double Precision Floating Point (2 Words)**

Integers are two's complement binary numbers.

Characters are usually in ASCII.

Floating point commonly follows the IEEE Standard 754.

Packed and Unpacked Decimal

Tradeoffs:

Memory Addressing

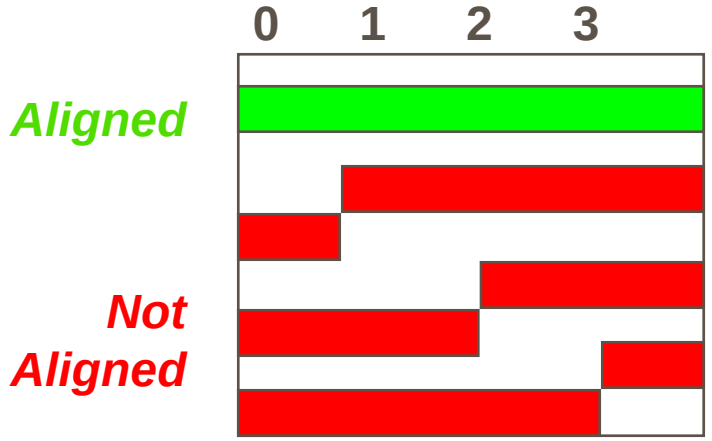
Memory Addressing Includes:

- **Interpreting Memory Addresses**
- **Addressing Modes**

Interpreting Memory Addresses

- **Big Endian:** address of *most* significant byte = word address
  (x…x000 = Big End of word)
  - IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA
- **Little Endian:** address of *least* significant byte = word address
  (x…x000 = Little End of word)
  - Intel 80x86, DEC Va, DEC Alpha (Windows NT)

Alignment: Data must be aligned on a boundary equal to its size. An access to an object of size *s* bytes at byte address *A* is aligned if *A* mod *s* = 0. Misalignment may result in multiple access which is normally handled by memory controller.



Addressing Modes

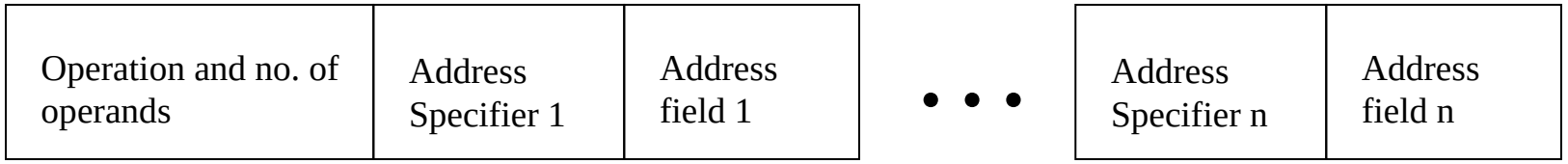
This table shows the most common modes.

Addressing Mode	Example Instruction	Meaning	When Used
Register	Add R4, R3	R[R4] <- R[R4] + R[R3]	When a value is in a register.
Immediate	Add R4, #3	R[R4] <- R[R4] + 3	For constants.
Displacement	Add R4, 100(R1)	R[R4] <- R[R4] + M[100+R[R1]]	Accessing local variables.
Register Deferred	Add R4, (R1)	R[R4] <- R[R4] + M[R[R1]]	Using a pointer or a computed address.
Absolute	Add R4, (1001)	R[R4] <- R[R4] + M[1001]	Used for static data.

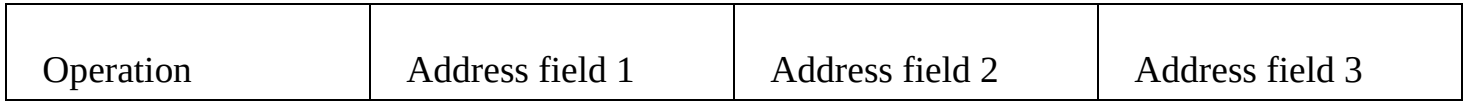
Instruction Encoding

Three basic variations in instruction encoding are **Variable length, fixed length and hybrid**.

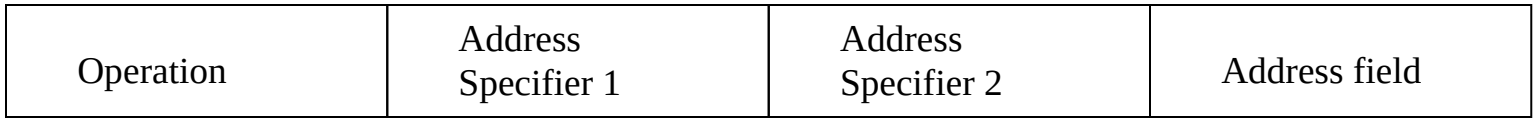
(a) **Variable** (e.g., VAX, Intel 80x86)



(b) **Fixed** (e.g., Alpha, ARM, MIPS, PowerPC, SPARC, SuperH)



(c) **Hybrid** (e.g., IBM360/370, MIPS16, Thumb, TI TMS320C54x)



The Role of Compilers

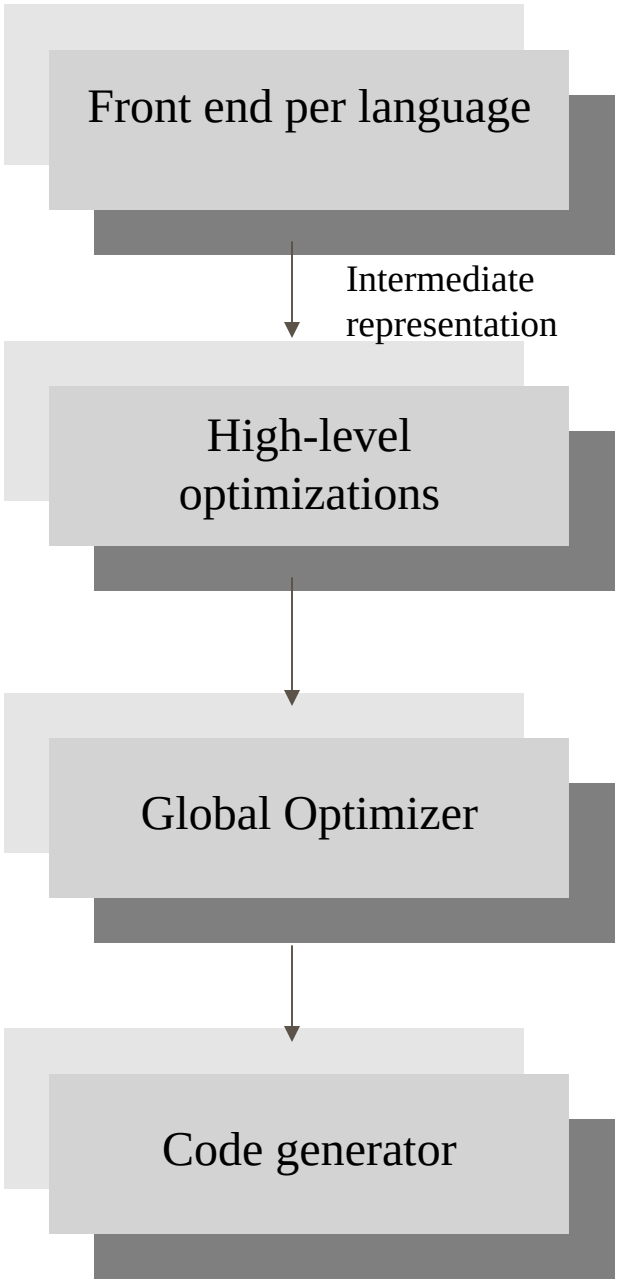
Compiler **goals**:

- All correct programs execute correctly
- Most compiled programs execute fast (optimizations)
- Fast compilation
- Debugging support

Role of compilers (Contd..)

Dependencies

Function



Language dependent;

Machine independent

Transform language to common intermediate form

Somewhat language dependent;

Largely machine independent

For example, loop transformations and procedure inlining (also called procedure integration)

Small language dependencies;

Machine dependencies slight

(e.g., rgister counts/types)

Including global and local optimizations + register allocation

Highly machine dependent;

Language independent

Detailed Instruction selection and machine-dependent optimizations; may include or be followed by assembler

Steps in Compilation

Parsing --> intermediate representation

Jump Optimization

Loop Optimizations

Register Allocation

Code Generation --> assembly code

Common Sub-Expression

Procedure in-lining

Constant Propagation

Strength Reduction

Pipeline Scheduling

OPTIMIZATIONS

Optimizations performed by modern compilers can be classified by the style of the transformation, as follows:

- ***High-level optimizations*** are often done on the source with output fed to later optimization passes.
- ***Local optimizations*** optimize code only within a straight-line code fragment (called *basic block* by compiler people).
- ***Global optimizations*** extend the local optimizations across branches and introduce a set of transformations aimed at optimizing loops.
- ***Register allocation*** associates registers with operands.
- ***Processor-dependent optimizations*** attempt to take advantage of specific architectural knowledge.

Architect's Help To The Compiler Writer

With respect to Instruction Set
- Regularity
- Orthogonality
- Composability

Compilers perform a giant case analysis
- too many choices make it hard

Orthogonal instruction sets
- operation, addressing mode, data type

One solution or all possible solutions
- 2 branch conditions - eq, lt
- or all six - eq, ne, lt, gt, le, ge
- **not** 3 or 4

There are advantages to having instructions that are primitives.

Let the compiler put the instructions together to make more complex sequences.

The MIPS Architecture – A Study

MIPS Characteristics

32-bit byte addresses aligned
Load/store - only displacement addressing
Standard data types
3 fixed length formats
32 32-bit GPRs (r0 = 0)
16 64-bit (32 32-bit) FPRs
FP status register
No Condition Codes

There's MIPS – 64 – the current arch.
Standard datatypes
4 fixed length formats (8,16,32,64)
32 64-bit GPRs (r0 = 0)
64 64-bit FPRs

Addressing Modes

- Immediate
- Displacement
- (Register Mode used only for ALU)

Data transfer

- load/store word, load/store byte/halfword signed?
- load/store FP single/double
- moves between GPRs and FPRs

ALU

- add/subtract signed? immediate?
- multiply/divide signed?
- and,or,xor immediate?, shifts: ll, rl, ra immediate?
- sets immediate?

MIPS Characteristics (Contd..)

Control

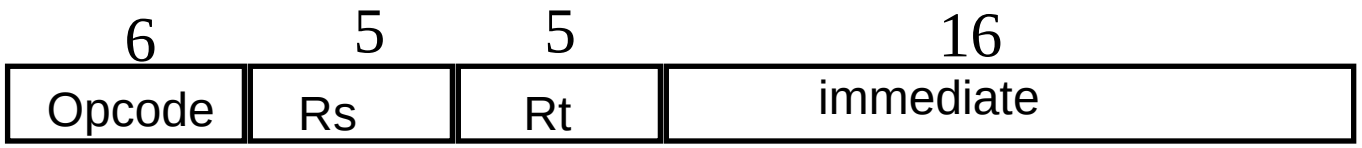
- branches == 0, <> 0
- conditional branch testing FP bit
- jump, jump register
- jump & link, jump & link register
- trap, return-from-exception

Floating Point

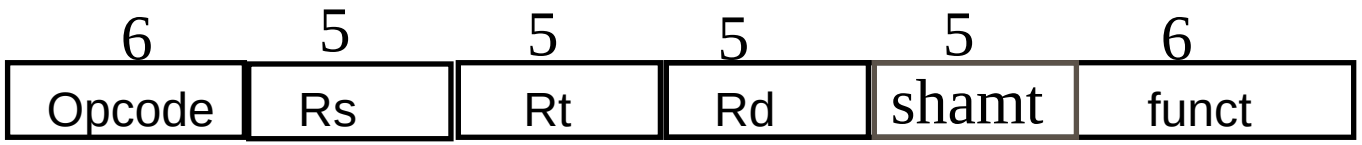
- add/sub/mul/div
- single/double
- fp converts, fp set

MIPS - Instruction formats

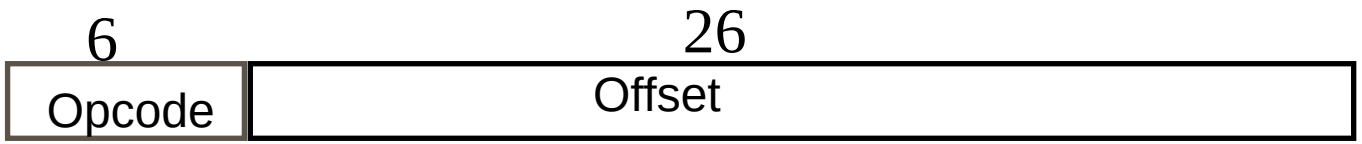
I-type instruction



R-type instruction

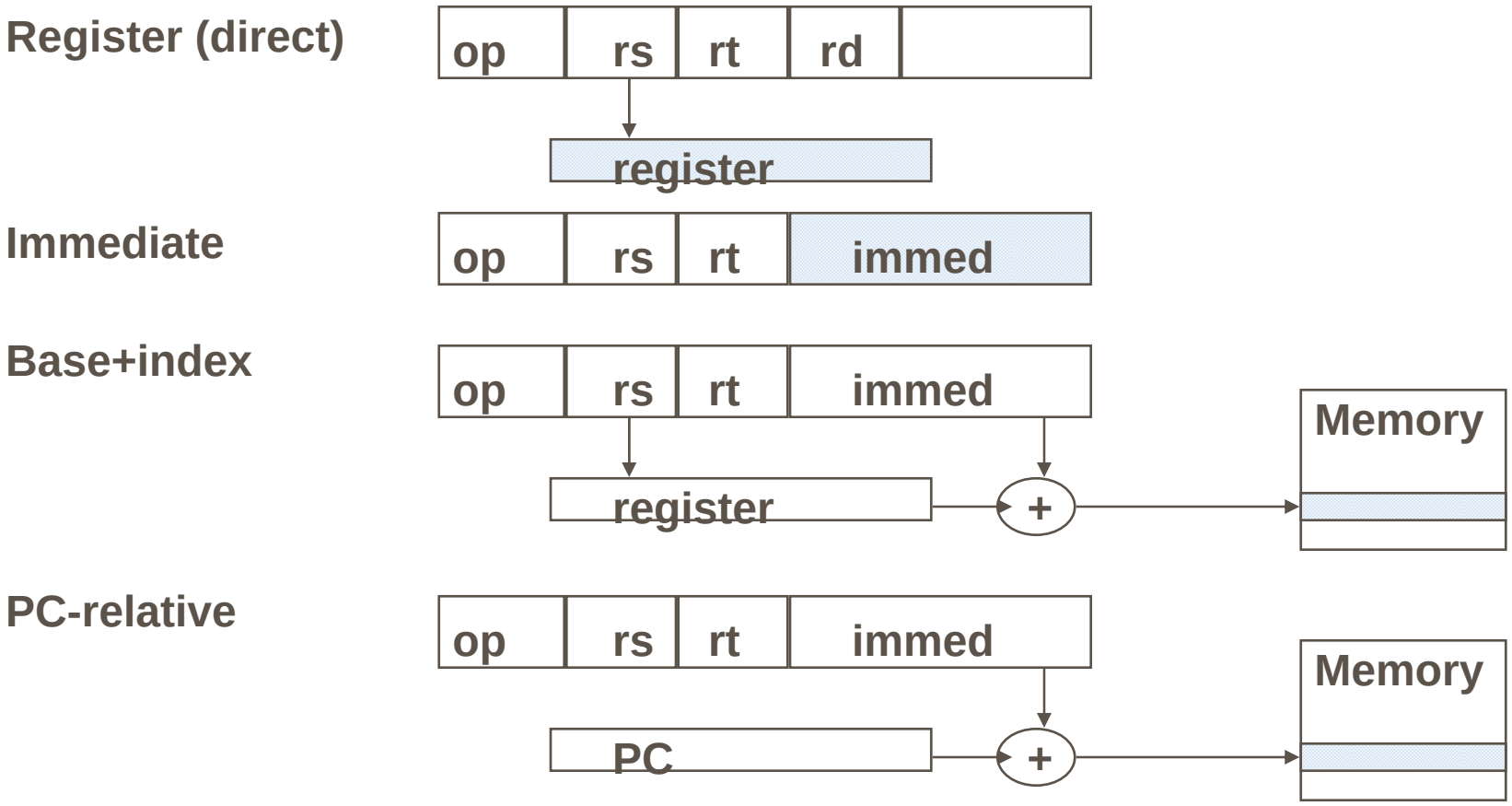


J-type instruction



MIPS Addressing Modes & Formats

- Simple addressing modes
- All instructions 32 bits wide



- Register Indirect?

Summary

- ISA forms an important part of the design
- Many issues need to be decided while designing the ISA