Introduction

Muhammad Tahir

Lecture 1-2

Electrical Engineering Department
University of Engineering and Technology Lahore

Contents

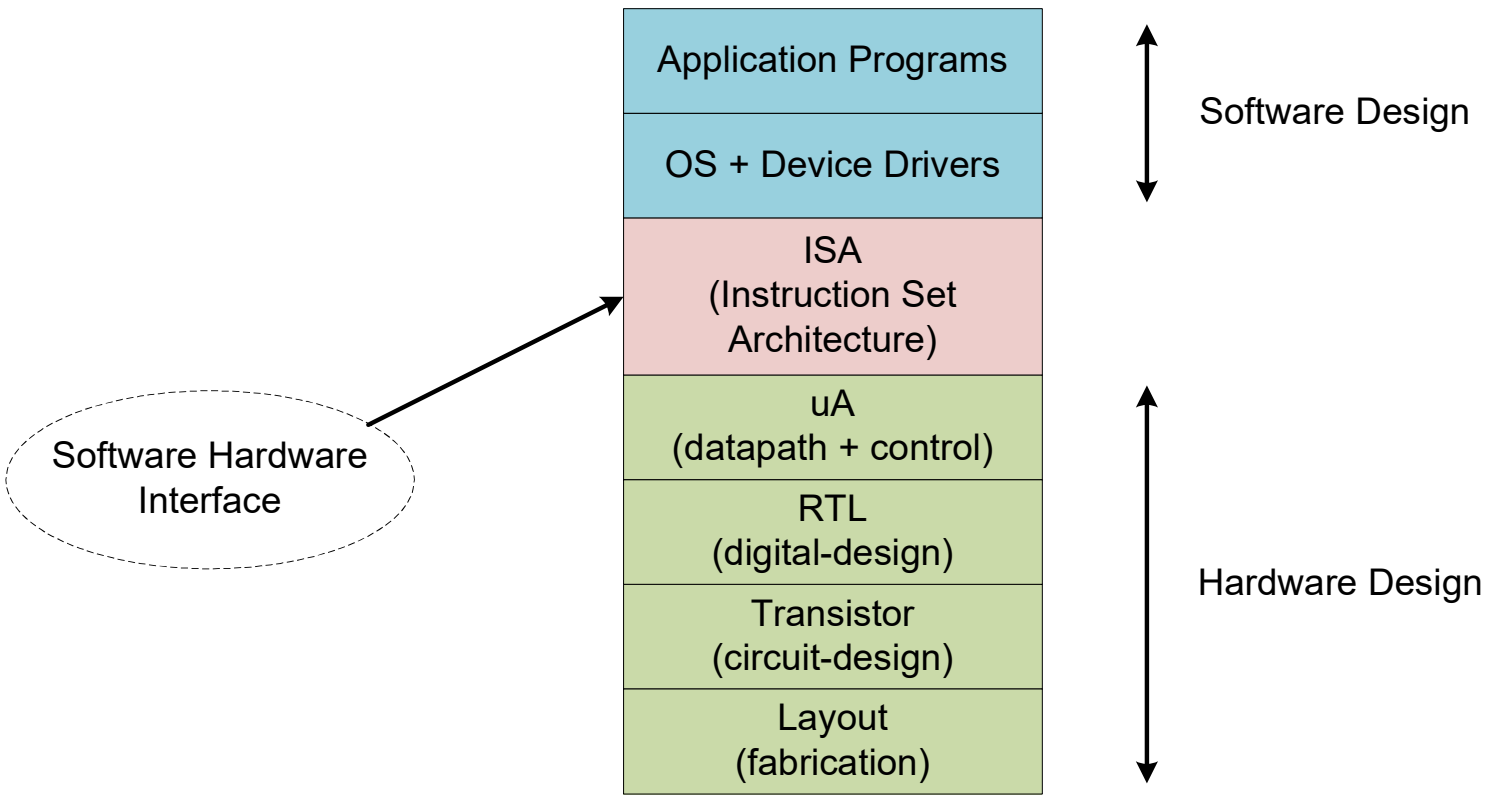
What is Computer Architecture?

Computer Architecture defines computer `functionality` and its `organization` while meeting certain `implementation` specific performance attributes

- Functionality – Instruction set architecture (ISA) design
- Organization – Microarchitecture design
- Implementation –
  - Low power
  - High performance
  - Scalability
  - Throughput

Instruction Set Architecture

- Instruction Set Architecture (ISA) acts as an `interface` between hardware and software



ISA Design

An ISA is precisely the computer design from the `Programmer's Perspective`

- Which instructions to have?
- Which data types to be supported?
- What should those instructions do?
- How instruction operands are specified and accessed
- How complex an instruction can be?

ISA Design Cont'd

- An ISA is required to be complete
- Four principles to be followed when designing an ISA
  - `Regularity` $\sim$ Leads to simplicity
  - `Modularity` $\sim$ Smaller is faster
  - `Optimized Tradoff` $\sim$ Ease of compilation and implementation
  - `Performance` $\sim$ Make the common case fast (exploit principle of locality)

ISA Attributes

- Instruction Set is the vocabulary of the language that a computer understands
- An Instruction Set Architecture defines:
  - The data types to be supported
  - Set of states that are visible to the programmer
  - Instruction operations, operand types as well as operand size and sequencing
  - Instruction format and its encoding
- Example ISAs: ARM Thumb2, x86, MIPS, RISC-V

ISA as HW-SW Co-Design

Consider an example high-level instruction

```
// vector multiplication
result = result + a[i] * b[j];
```

ISA as HW-SW Co-Design

Consider an example high-level instruction

```
// vector multiplication
result = result + a[i] * b[j];
```

Simple assembly instructions

```
// vector multiplication
MUL  Rt, Rs1, Rs2   // Rt = Rs1 * Rs2
ADD  Rd, Rd, Rt     // Rd = Rd + Rt
```

ISA as HW-SW Co-Design

Consider an example high-level instruction

```
// vector multiplication
result = result + a[i] * b[j];
```

Simple assembly instructions

```
// vector multiplication
MUL Rt, Rs1, Rs2   // Rt = Rs1 * Rs2
ADD Rd, Rd, Rt     // Rd = Rd + Rt
```

Complex assembly instruction (requires a different C-compiler)

```
MLA Rd, Rd, Rs1, Rs2    // Rd = Rd + Rs1 * Rs2
```

Optimizing ISA Implementation

- Each `ISA` can be realized using different micro-architectures
- Each `micro-architecture` can have many different implementations
- Each `implementation` can be laid out in many different ways

Processing Machine Performance

$$\text{Execution Time} = N \times CPI \times \frac{time}{cycle}$$

where,

$$\begin{aligned} N &\sim \text{instruction count} \\ CPI &\sim \frac{cycles}{instruction} \end{aligned}$$

Processing Machine Performance Cont'd

$$\text{Execution Time} = \sum_i (N_i \times CPI_i) \times \frac{time}{cycle}$$

where,

$i \quad \sim \quad$ instruction type index

How ISA Affects Performance?

- Compiled program number of instructions depends on:
  - Source program (high level language)
  - Compiler optimization capability
  - `Instruction Set Architecture`
- Cycles Per Instruction (CPI) depends on:
  - `Instruction Set Architecture`
  - Machine micro-architecture
- Time per cycle depends on:
  - Machine micro-architecture
  - Fabrication process

Amdahl's Law

$$T_o = T_o(1 - p) + pT_o$$

T_o $\sim$ Execution time for original implementation.

p $\sim$ Fraction of execution time for which performance enhancement can be achieved.

Amdahl's Law

$$T_o = T_o(1 - p) + pT_o$$

T_o $\sim$ Execution time for original implementation.

p $\sim$ Fraction of execution time for which performance enhancement can be achieved.

$$T_n = T_o(1 - p) + \frac{p}{s}T_o$$

T_n $\sim$ Execution time for new implementation.

s $\sim$ Speedup factor.

Amdahl's Law Cont'd

$$
\begin{aligned}
Speedup &= \frac{T_o}{T_n} \\
&= \frac{1}{(1-p) + \frac{p}{s}}
\end{aligned}
$$

Amdahl's Law Cont'd

$$\begin{aligned} Speedup &= \frac{T_o}{T_n} \\ &= \frac{1}{(1-p) + \frac{p}{s}} \end{aligned}$$

If 50% of task execution can be enhanced by a factor of 2 then

$$\begin{aligned} Speedup &= \frac{1}{(1-0.5) + \frac{0.5}{2}} \\ &= 1.33 \end{aligned}$$

Amdahl's Law Cont'd

$$\begin{aligned} Speedup &= \frac{T_o}{T_n} \\ &= \frac{1}{(1-p) + \frac{p}{s}} \end{aligned}$$

If 50% of task execution can be enhanced by a factor of 2 then

$$\begin{aligned} Speedup &= \frac{1}{(1-0.5) + \frac{0.5}{2}} \\ &= 1.33 \end{aligned}$$

$$\lim_{s \to \infty} Speedup = 2 \quad (\texttt{Limiting behavior})$$

Accessing The Memory

- `Object Size` $\sim$ byte, half-word, word, double-word
- `Byte Ordering` $\sim$ little-endian, big-endian
- `Alignment Issues` $\sim$
  - an access to an object of size s bytes at an address A is aligned if $A \mod s = 0$
  - misalignment causes hardware complications and in general requires more time to access the specified memory location
- `Addressing Modes` $\sim$ register, immediate, displacement/offset, register indirect etc.

Accessing The Memory: Displacement Addressing Mode

- `Relative Addressing`
  - The displacement is relative to instruction address (also termed as PC-relative addressing)
  - Used by flow control instructions for control transfer

Accessing The Memory: Displacement Addressing Mode

- `Relative Addressing`
  - The displacement is relative to instruction address (also termed as PC-relative addressing)
  - Used by flow control instructions for control transfer
- `Register Offset Addressing`
  - The base register contains a memory address
  - Address field contains displacement from memory address
  - Useful for segmented memory accesses

Accessing The Memory: Displacement Addressing Mode

- `Relative Addressing`
  - The displacement is relative to instruction address (also termed as PC-relative addressing)
  - Used by flow control instructions for control transfer
- `Register Offset Addressing`
  - The base register contains a memory address
  - Address field contains displacement from memory address
  - Useful for segmented memory accesses
- `Indexed Addressing`
  - Address field contains a memory address
  - The register contains displacement from memory address
  - Useful for iterative operations

Computer Architecture Classification

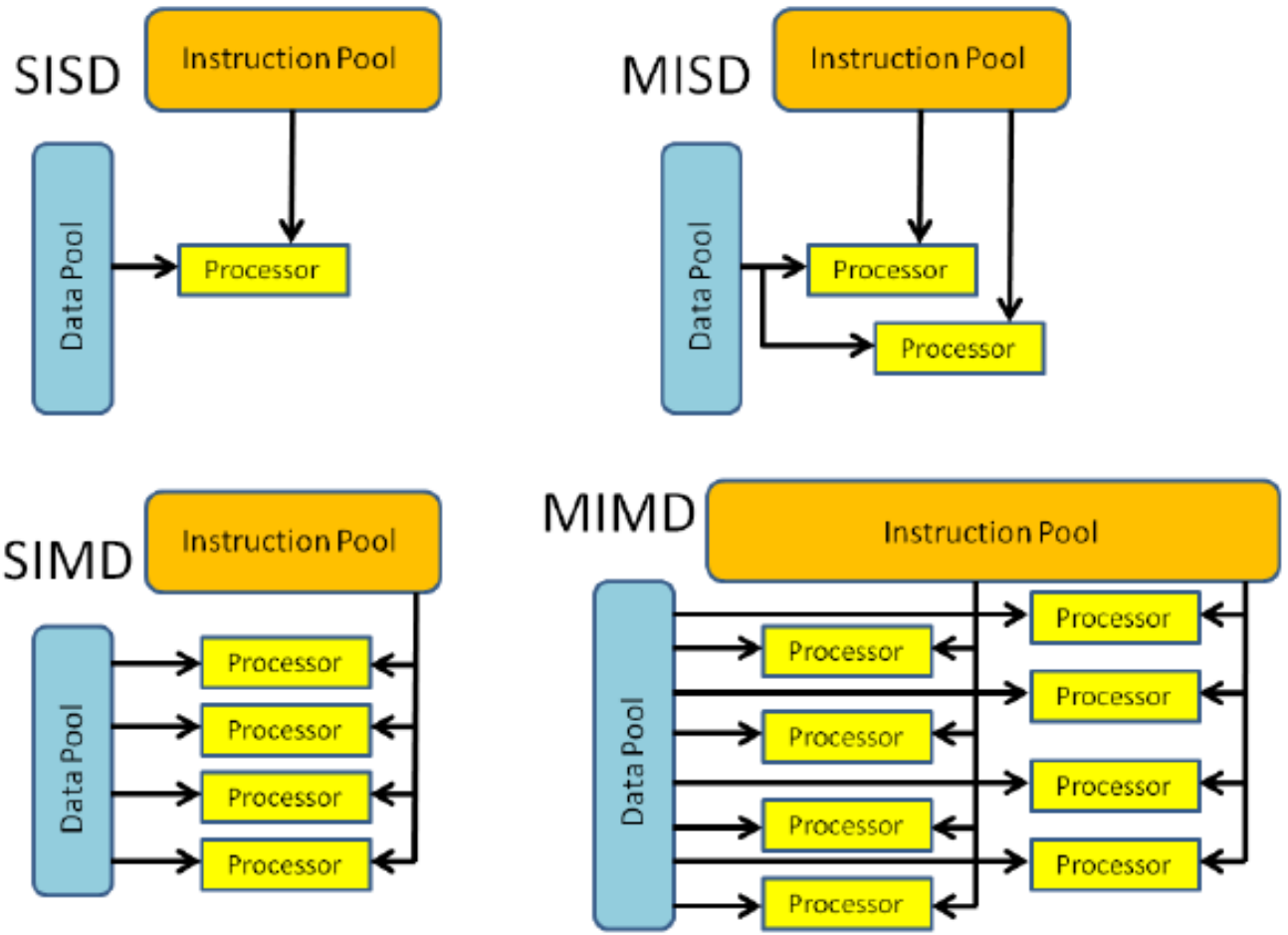


Figure: Flynn's taxonomy of computer architecture [Flynn, 1972].

Computer Architecture Classification Cont'd

- SISD –
  - Uni-processor (single core) system
  - Exploits instruction level parallelism
- MISD – No commercial products, managed by MIMD
- SIMD –
  - Array/vector processor, GPU
  - Exploits data level parallelism
- MIMD –
  - Multi-core processor system
  - Exploits thread level parallelism

Parallelism in Computer Architecture

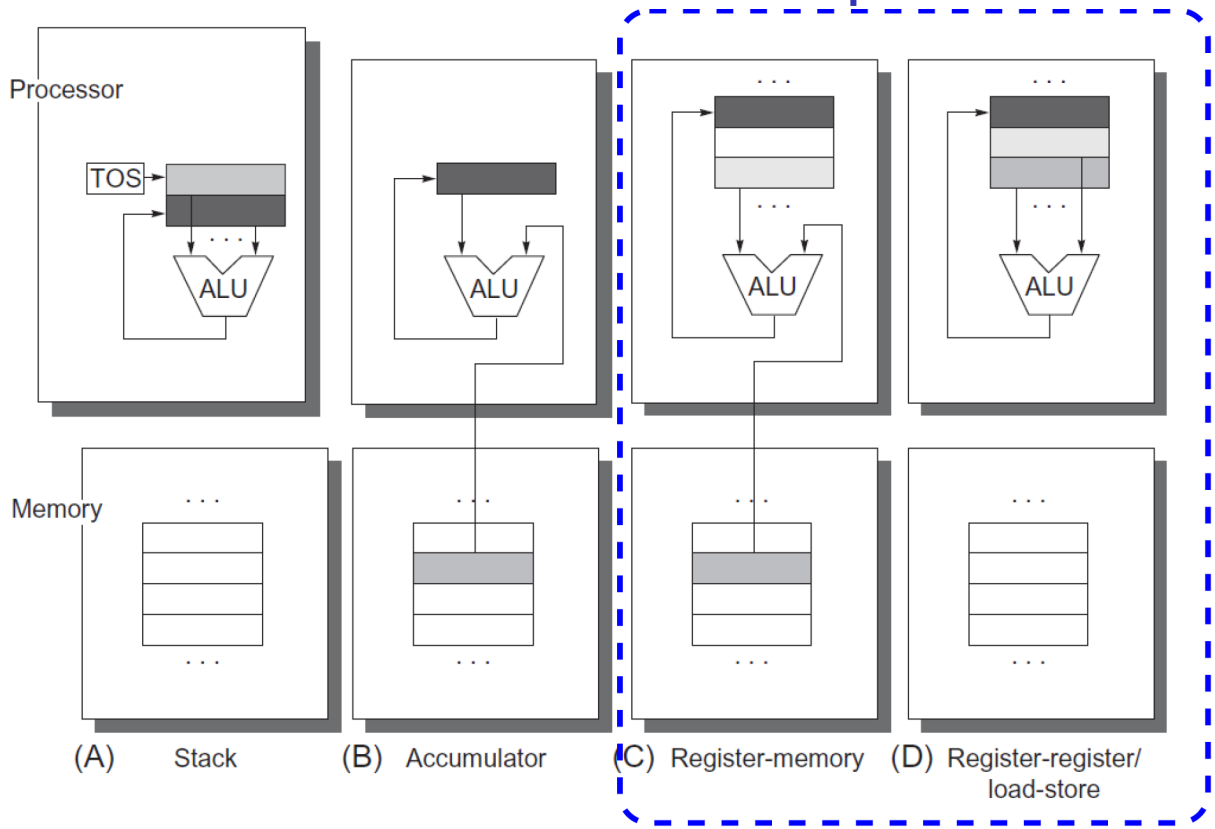
Four types of parallelism in computer architecture

- Bit level – 16-bit, 32-bit, 64-bit processor
- Instruction level – Pipelining, in-order, out-or-order multiple issue
- Data level – Array/vector processing
- Thread level – Multi core processor

ISA Classification

- Instruction set architecture can be classified based on the following attributes:
  - Instruction operand location/type
  - Instruction complexity

ISA Classification: Operand Placement



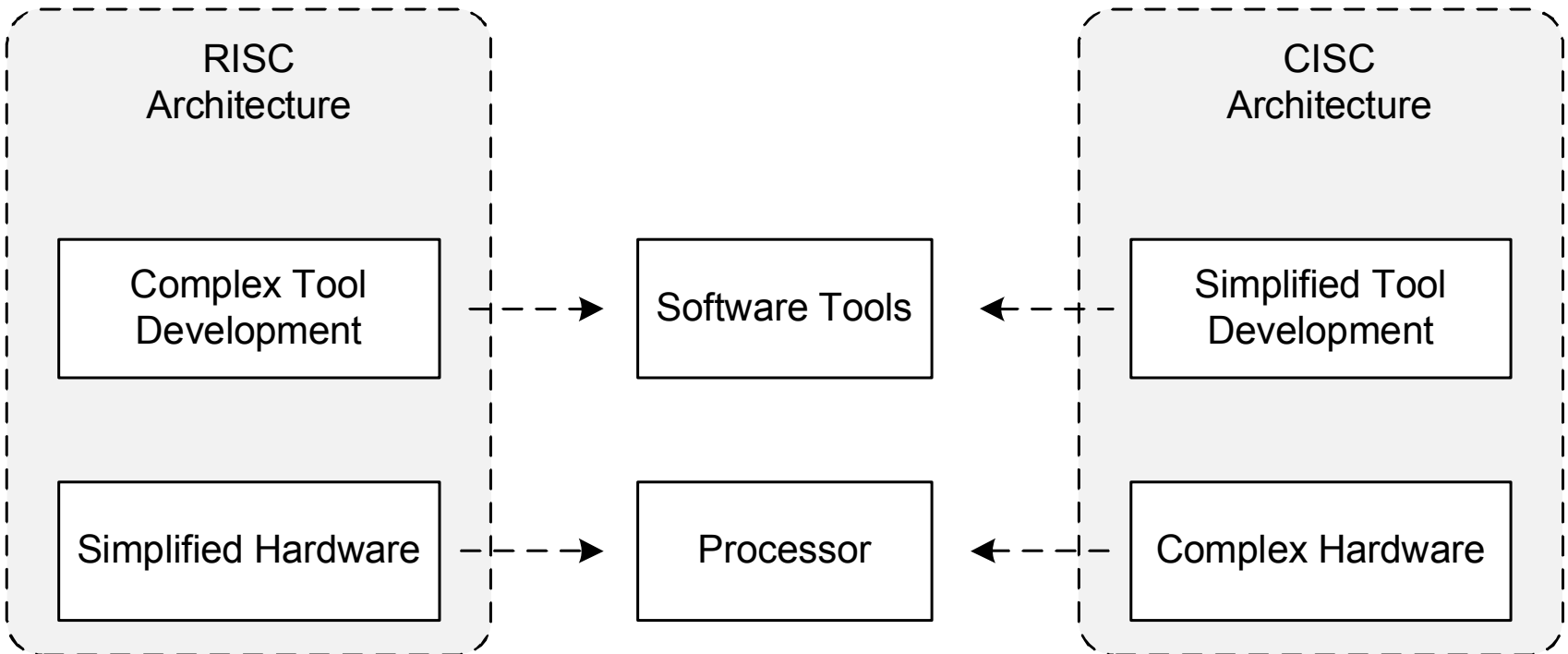
Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R3,R1,B	Load R2,B
Add	Store C	Store R3,C	Add R3,R1,R2
Pop C			Store R3,C

(Source: Appendix A, [Patterson and Hennessy, 2019])

ISA Classification: Instruction Complexity

- Instruction set architecture classification based on complexity
  - Complex instruction set computers (CISC)
  - Reduced instruction set computers (RISC)

CISC vs RISC


RISC
Architecture
Complex Tool
Development
Simplified Hardware
Software Tools
Processor
CISC
Architecture
Simplified Tool
Development
Complex Hardware

CISC Features

- A single complex instruction can perform multiple operations
- Complex instructions, with varying instruction length, require many processor clock cycles to complete
- A program running on a CISC architecture based machine involves a relatively smaller number of complex instructions, thus providing high code density
- In CISC the complexity is embedded in the processor hardware, making the compilation tools design simpler

RISC Features

- Simplified instructions used by the architecture
- Each complex operation is broken into multiple simplified operations e.g., load and store instructions are provided for memory read and write operations. Other instructions cannot access memory directly.
- RISC computer requires a relatively larger number of simplified instructions and results in low code density
- An associated advantage is fewer memory addressing modes resulting in reduced complexity
- The simplicity in the hardware architecture in RISC is complemented by the increased complexity in the generation of assembly code by the tools (compiler)

Instruction Encoding

Operation and no. of operands	Address specifier 1	Address field 1	...	Address specifier *n*	Address field *n*

(A) Variable (e.g., Intel 80x86, VAX)

Operation	Address field 1	Address field 2	Address field 3

(B) Fixed (e.g., RISC V, ARM, MIPS, PowerPC, SPARC)

Operation	Address specifier	Address field

Operation	Address specifier 1	Address specifier 2	Address field

Operation	Address specifier	Address field 1	Address field 2

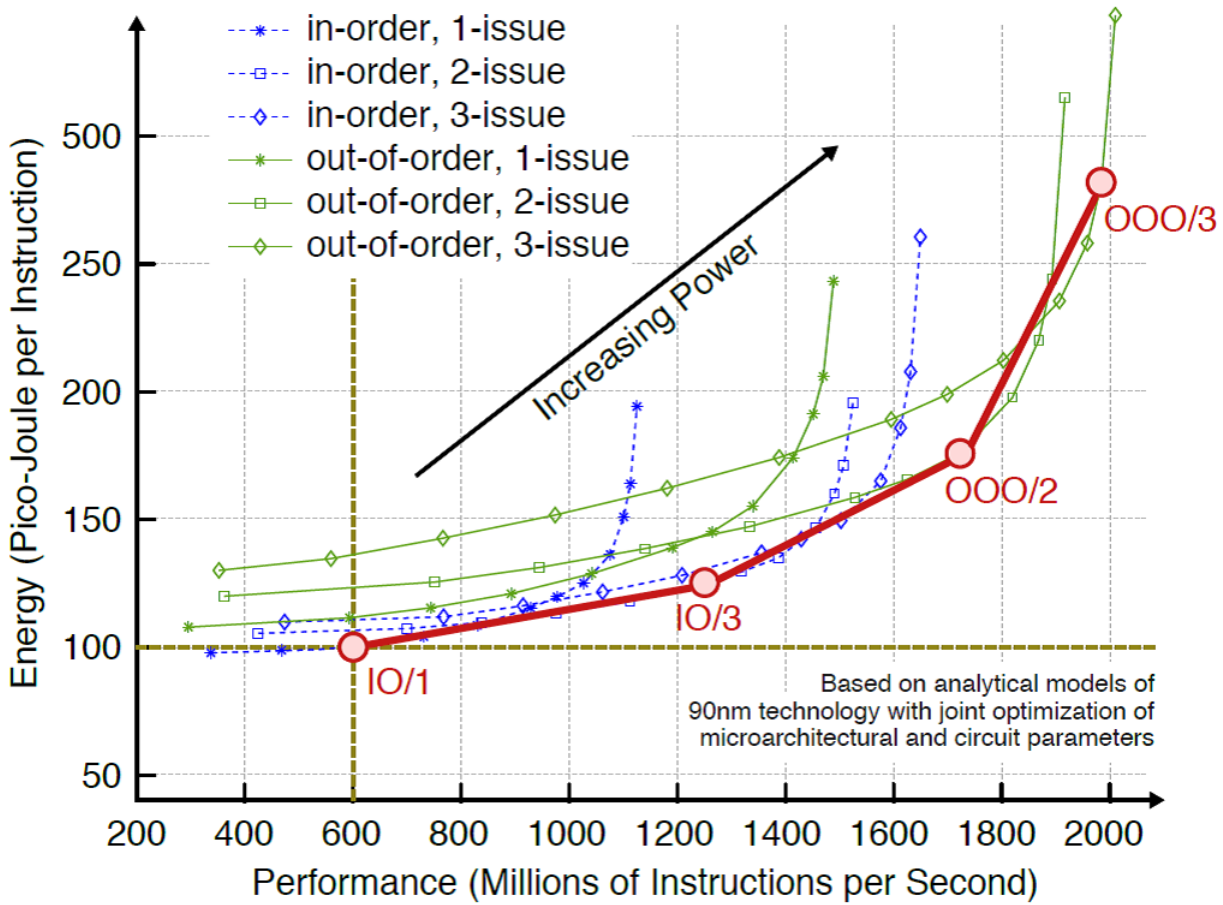
(C) Hybrid (e.g., RISC V Compressed (RV32IC), IBM 360/370, microMIPS, Arm Thumb2)

(Source: Appendix A, [Patterson and Hennessy, 2019])

Computer Architecture Trends

- A consequence of `application demands` versus `technological limitations`
- Key Trends
  - Diverse applications ranging from tiny IoT devices to cloud compute
  - High power density motivating new designs
  - Frequency scaling limitation leading to multi-core designs
  - Demanding applications leading to heterogeneous system(s)-on-chip
  - Limitations/challenges due to technology scalability motivating custom solutions

Power versus Performance



Adpated from O. Azizi et al. "Energy-Performance Tradeoffs ..." ISCA, 2010.

Heterogeneous System-on-Chip

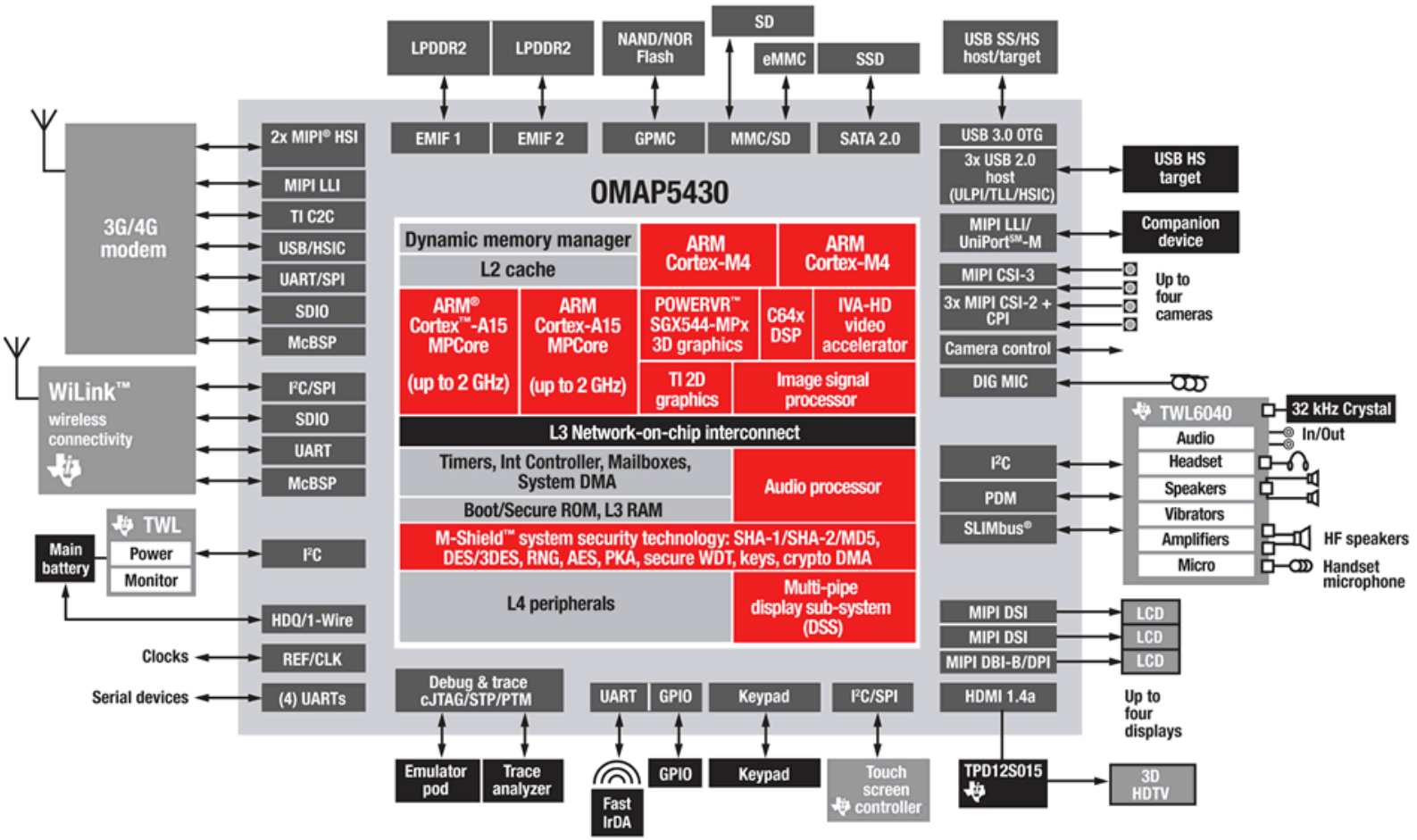
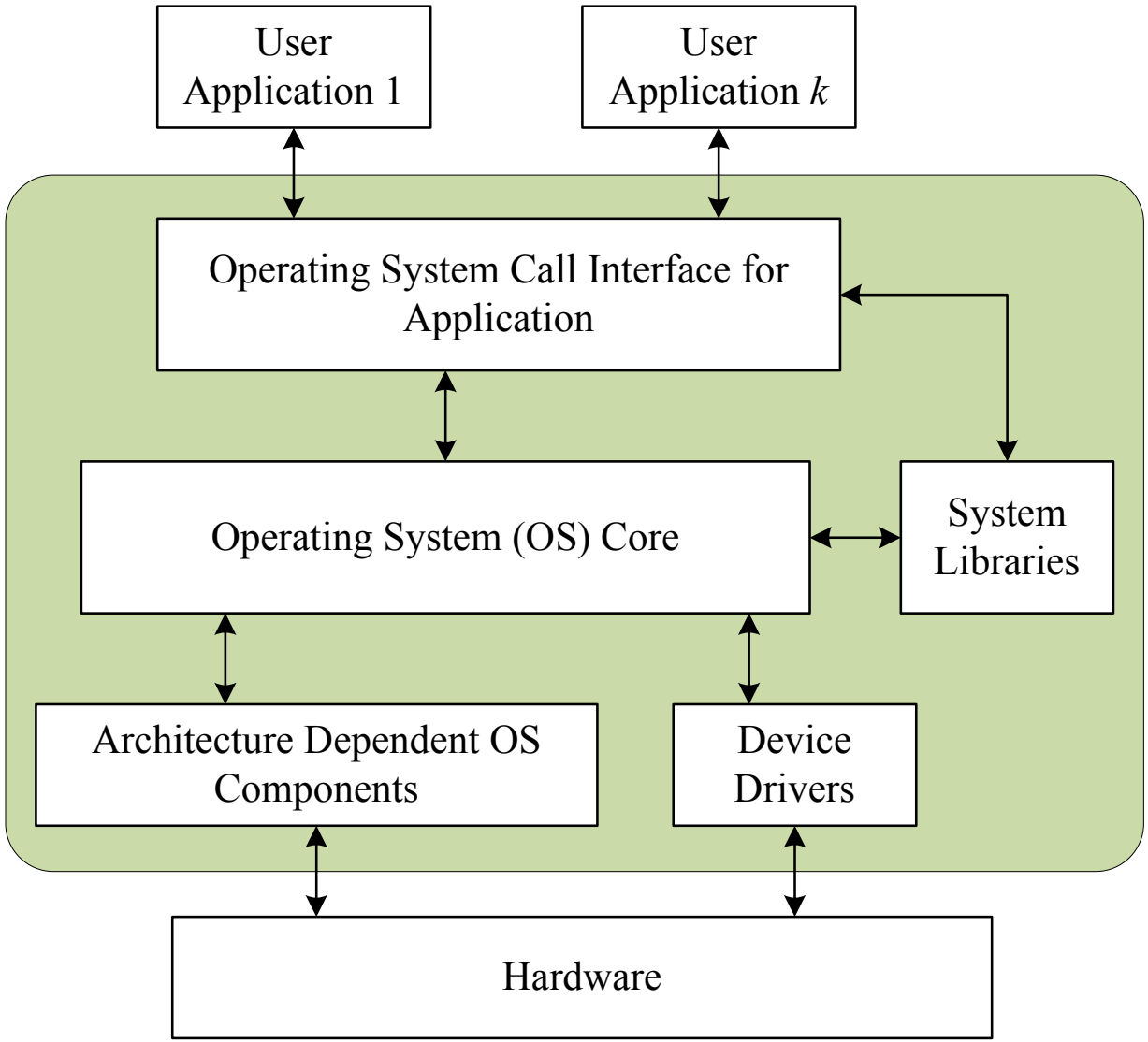
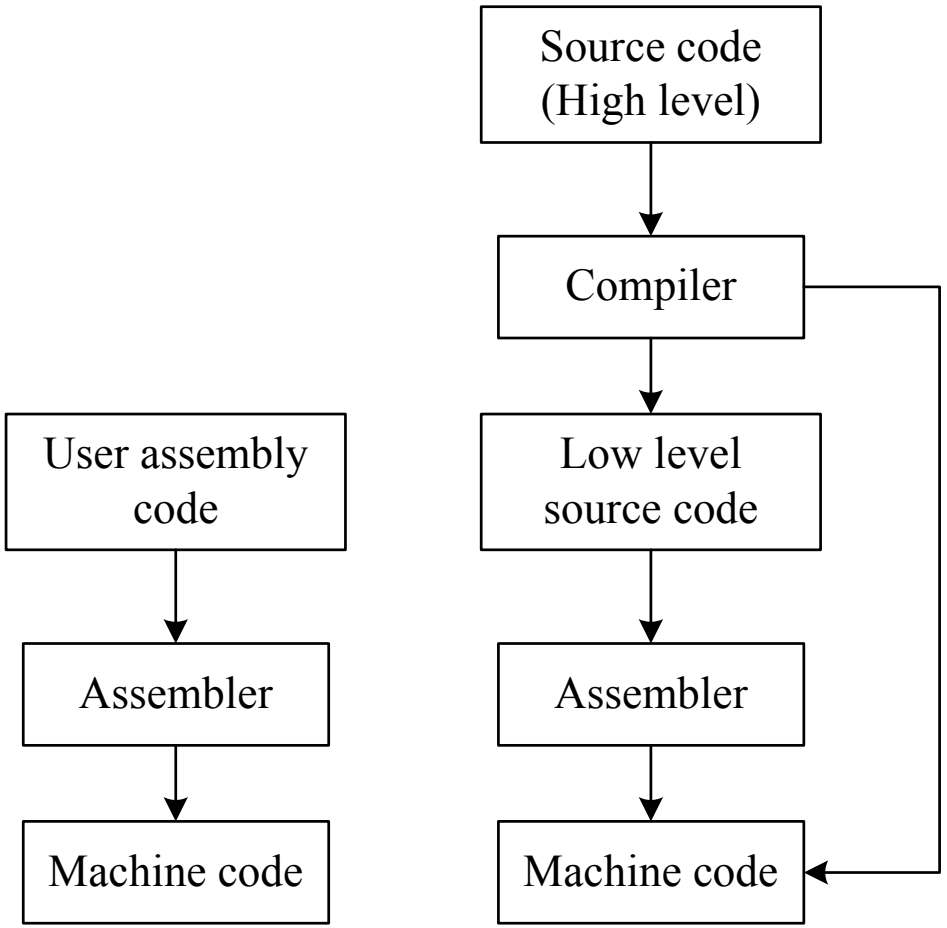


Figure: OMAP5 – Highly integrated SoC from Texas Instruments.

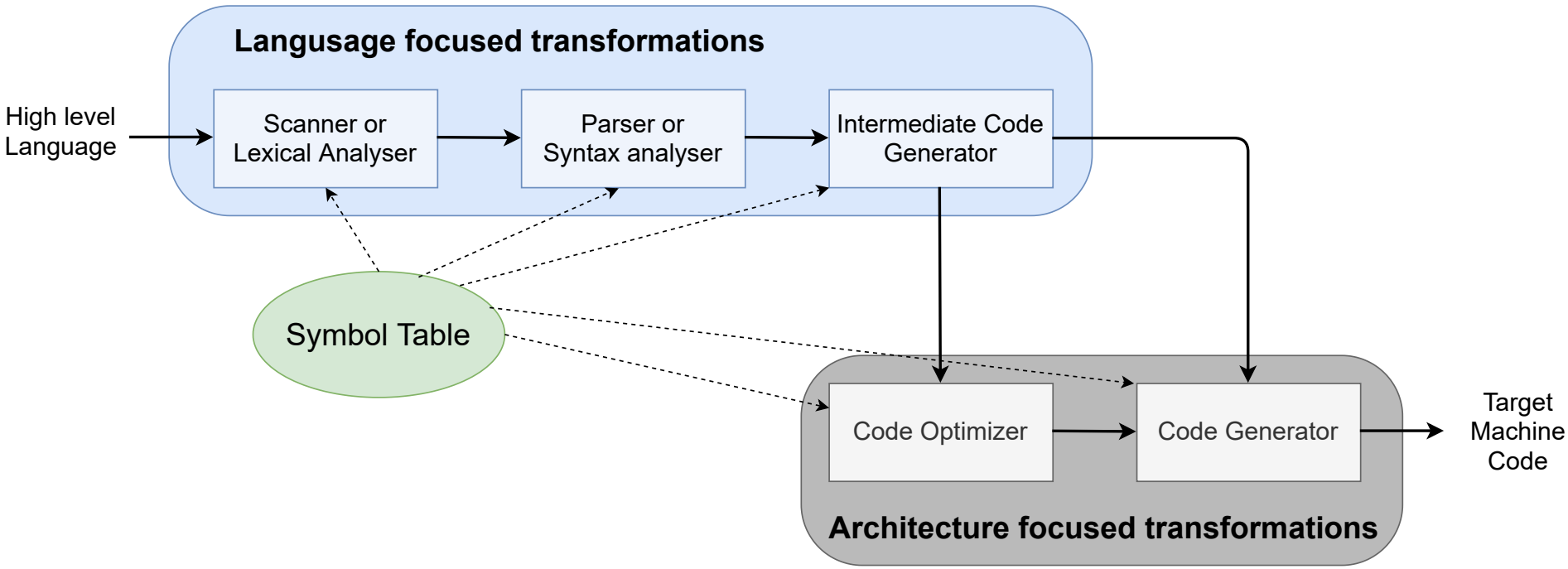
Software System



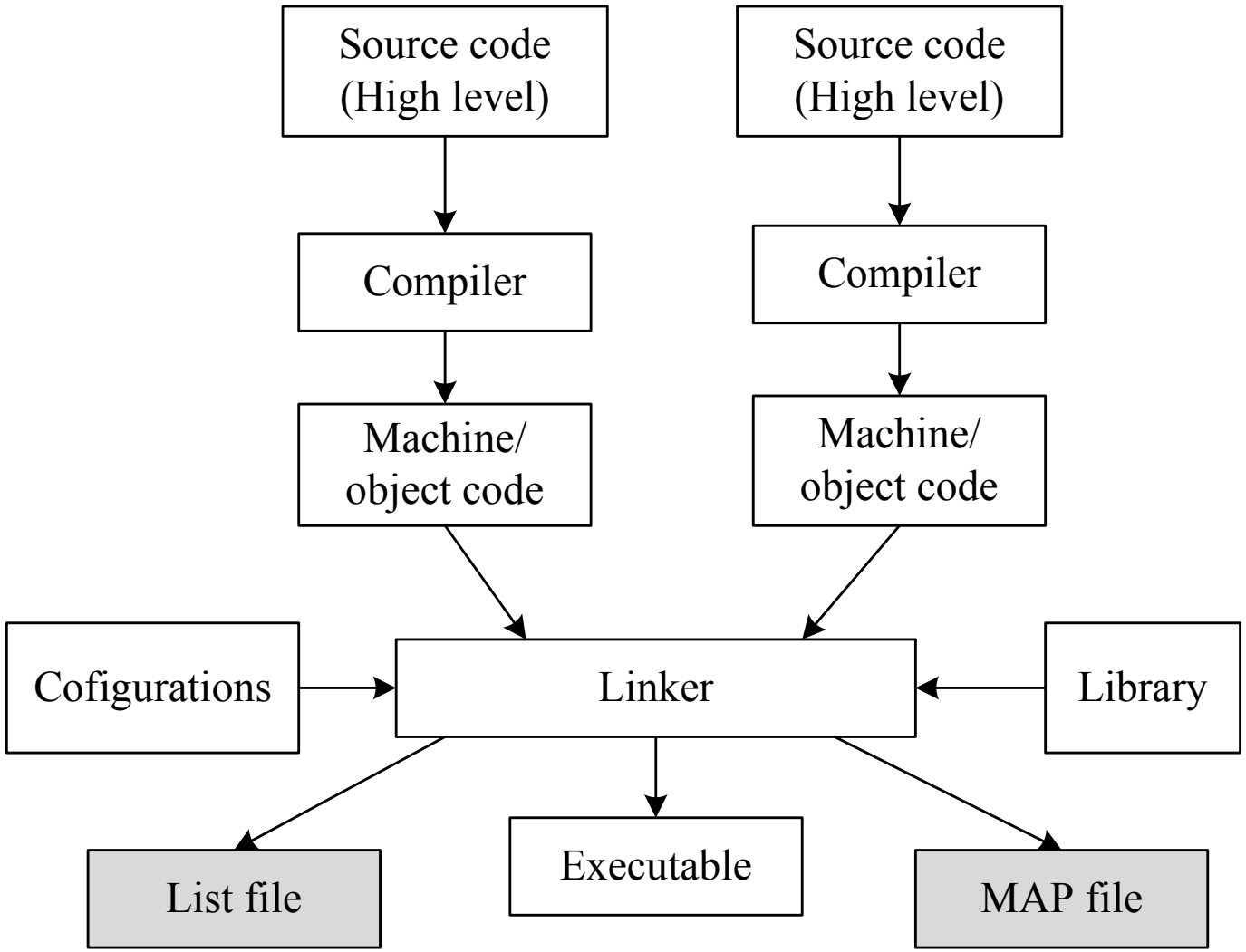
Compilation Process



Detailed Compilation Process



Linking Process



Suggested Reading

- Read Chapter 1 of *Computer Architecture: A Quantitative Approach* [Patterson and Hennessy, 2019].
- Read Appendix A of *Computer Architecture: A Quantitative Approach* [Patterson and Hennessy, 2019] for ISA principles.

Acknowledgment

- Preparation of this material was partly supported by Lampro Mellon Pakistan.

References

Flynn, M. J. (1972).
Some computer organizations and their effectiveness.
IEEE transactions on computers, 100(9):948–960.

Patterson, D. and Hennessy, J. (6th Edition, 2019).
Computer Architecture: A Quantitative Approach.
Morgan Kaufmann.