Computer Architecture and Operating Systems
Lecture 4: Instruction Set Architecture

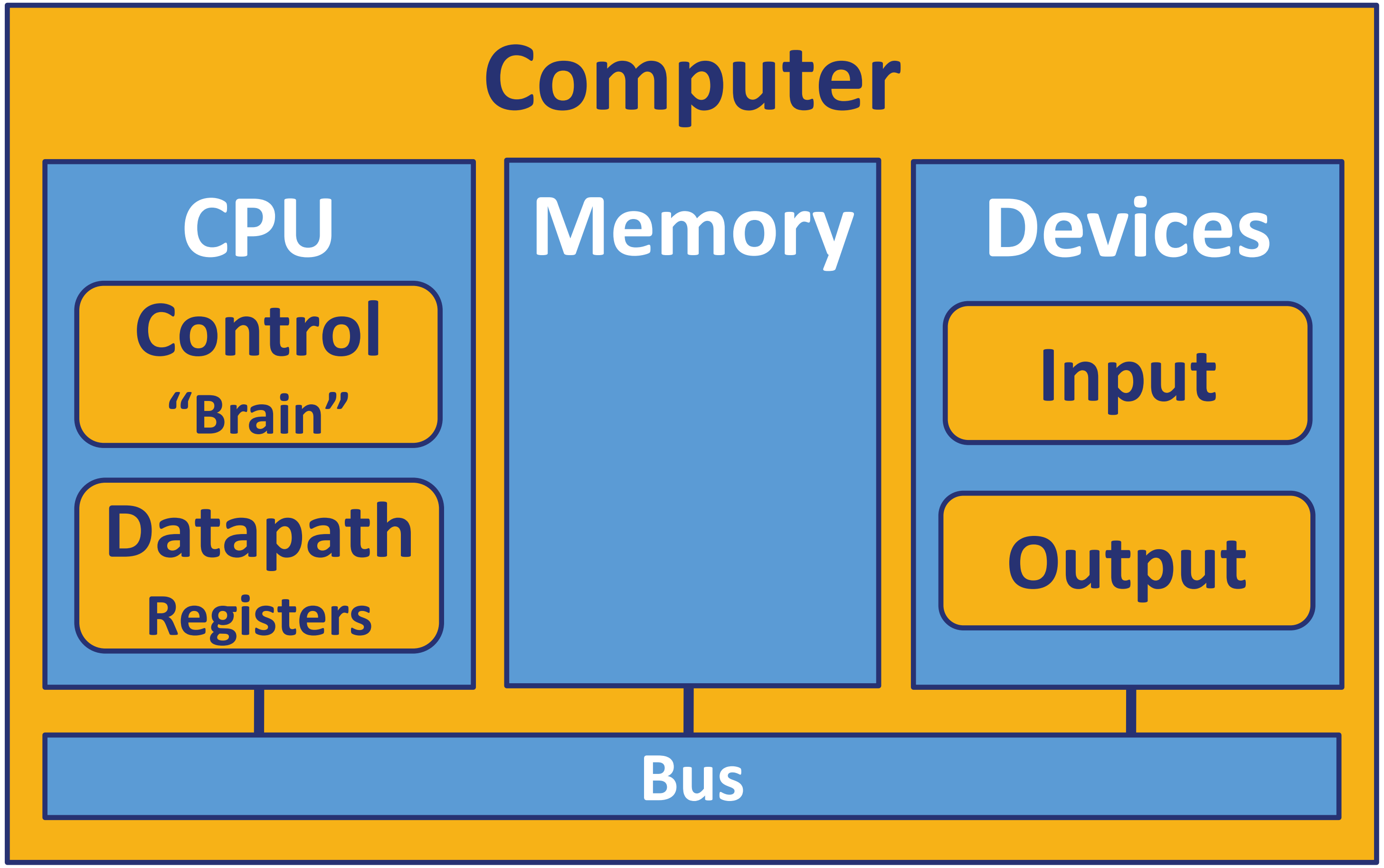
Andrei Tatarnikov

atatarnikov@hse.ru
@andrewt0301

How Computer Works

Main Parts:

- Control
- Datapath
- Memory
- Input
- Output



Stored Program Concept

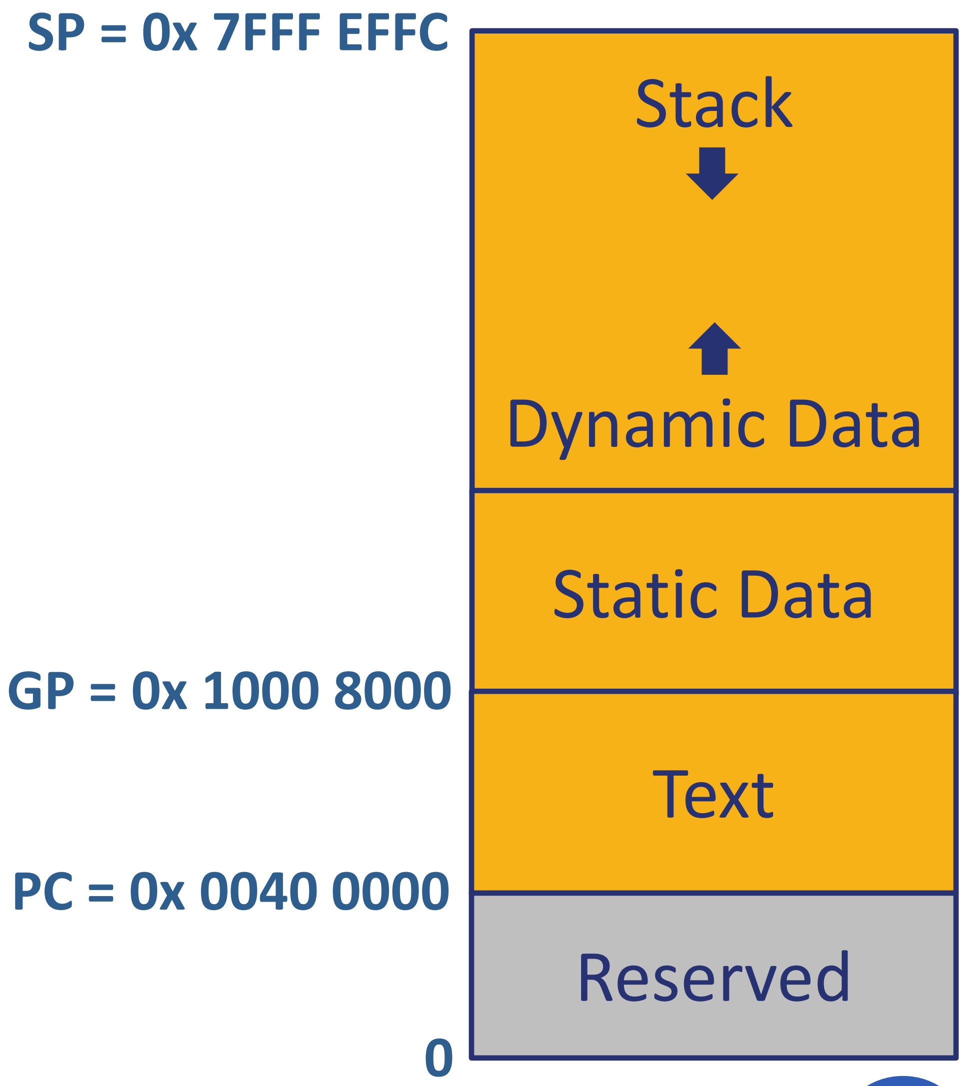
- 32-bit instructions and data stored in memory
- Program is a sequence of instructions
- To run a new program:
  - Simply load the new program into memory
- Program Execution:
  - CPU fetches (reads) instructions from memory in sequence
  - CPU performs the specified operations

Stored Program Representation

Program Counter (PC): keeps track of current instruction

Memory Layout

- Text: program code
- Static data: global variables
  - E.g., static variables in C, constant arrays and strings
- Dynamic data: heap
  - E.g., malloc in C, new in Java
- Stack: automatic storage



RISC-V Instructions

Name	Description	Version	Status
	Base		
RVWMO	Weak Memory Ordering	2.0	Ratified
RV32I	Base Integer Instruction Set, 32-bit	2.1	Ratified
RV64I	Base Integer Instruction Set, 64-bit	2.1	Ratified
RV128I	Base Integer Instruction Set, 128-bit	1.7	Open
	Extensions		
M	Standard Extension for Integer Multiplication and Division	2.0	Ratified
A	Standard Extension for Atomic Instructions	2.1	Ratified
F	Standard Extension for Single-Precision Floating-Point	2.2	Ratified
D	Standard Extension for Double-Precision Floating-Point	2.2	Ratified
G	Shorthand for the base integer set (I) and above extensions (MAFD)	N/A	N/A
Q	Standard Extension for Quad-Precision Floating-Point	2.2	Ratified
C	Standard Extension for Compressed Instructions	2.0	Ratified
ZiCSR	Control and Status Register (CSR)	2.0	Ratified
Zifencei	Instruction-Fetch Fence	2.0	Ratified
	And more standard and custom extensions…		

Design Principles

- **Design Principle 1:** Simplicity favors regularity
  - Regularity makes implementation simpler
  - Simplicity enables higher performance at lower cost
- **Design Principle 2:** Smaller is faster
  - 32 registers, fewer instructions
- **Design Principle 3:** Good design demands good compromises
  - Different formats complicate decoding, but allow 32-bit instructions uniformly
  - Keep formats as similar as possible

Six Instruction Formats

- **R-format:** instructions using 3 register inputs
  - add, xor, mul - arithmetic/logical ops
- **I-format:** instructions with immediates, loads
  - addi, lw, jalr, slli
- **S-format:** store instructions
  - sw, sb
- **SB-format:** branch instructions
  - beq, bge
- **U-format:** instructions with upper immediates
  - lui, auipc - upper immediate is 20-bits
- **UJ-format:** the jump instruction
  - jal

R-format Instructions

funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

add x9, x20, x21

0	21	20	0	9	51
0000000	10101	10100	000	01001	0110011

$0000\ 0001\ 0101\ 1010\ 0000\ 0100\ 1011\ 0011_{two} = 015A04B3_{16}$

- Arithmetic Instructions
  - opcode: operation code
  - rd: destination register number
  - funct3: 3-bit function code (additional opcode)
  - rs1 and rs2: first and second source register 5-bit numbers
  - funct7: 7-bit function code (additional opcode)

I-format Instructions

immediate	rs1	funct3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits

addi t0, t1, 123

0x123	6	0	5	19
000100100011	00110	000	00101	0010011

$0001\ 0010\ 0011\ 0011\ 0000\ 0010\ 1001\ 0011_{two} = 0x12330293_{16}$

- Immediate arithmetic and load instructions
  - rs1: source or base address register number
  - immediate: constant operand, or offset added to base address
    - 2s-complement, sign extended

S-format Instructions

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

sw t0, 4(t1)

0	5	6	2	4	35
0000000	00101	00110	010	00100	100011

$0000\ 0000\ 0101\ 0011\ 0010\ 0010\ 0010\ 0011_{two} = 0x00532223_{16}$

- Different immediate format for store instructions
  - rs1: base address register number
  - rs2: source operand register number
  - immediate: offset added to base address
    - Split so that rs1 and rs2 fields always in the same place

SB-format Instructions

	imm[10:5]	rs2	rs1	funct3	imm[4:1]		opcode

imm[12] imm[11]

		rs2	rs1	funct3	imm [4:1]		opcode

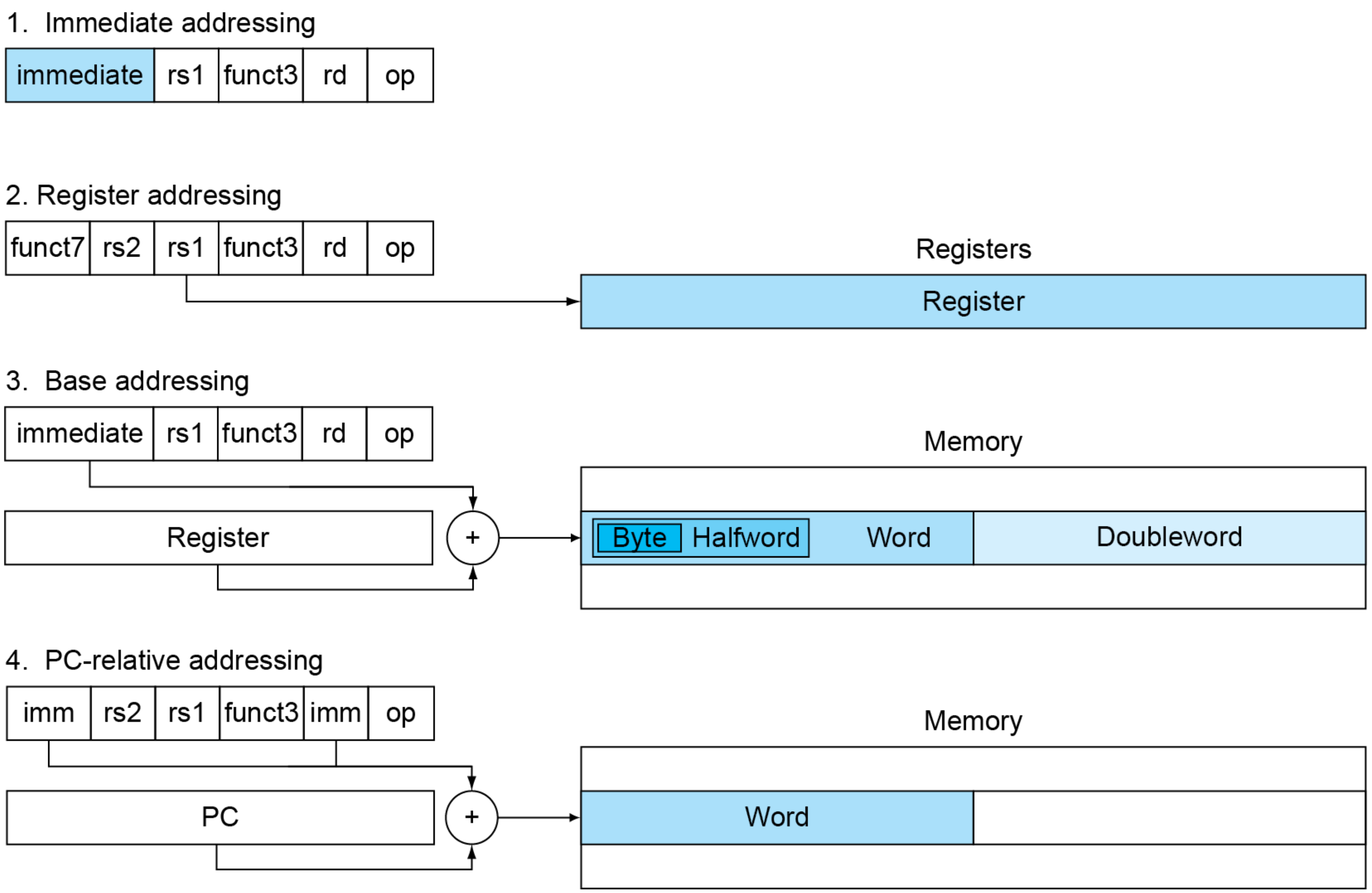
- Branch instructions specify
  - Opcode, two registers, target address
- Most branch targets are near branch
  - Forward or backward
- PC-relative addressing
  - Target address = PC + immediate × 2

UJ-format Instructions

	imm[10:1]		imm[19:12]	rd	opcode
imm[20]		imm[11]		5 bits	7 bits

U-format Instructions

RISC-V Addressing Summary



RISC-V Encoding Summary

Name	Field						Comments
(Field Size)	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	
R-type	funct7	rs2	rs1	funct3	rd	opcode	Arithmetic instruction format
I-type	immediate[11:0]		rs1	funct3	rd	opcode	Loads & immediate arithmetic
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Stores
SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Conditional branch format
UJ-type	immediate[20,10:1,11,19:12]				rd	opcode	Unconditional jump format
U-type	immediate[31:12]				rd	opcode	Upper immediate format

Any Questions?

```
            .text
__start:    addi t1, zero, 0x18
            addi t2, zero, 0x21
cycle:      beq t1, t2, done
            slt t0, t1, t2
            bne t0, zero, if_less
            nop
            sub t1, t1, t2
            j cycle
            nop
if_less:    sub t2, t2, t1
            j cycle
done:       add t3, t1, zero
```