NATIONAL RESEARCH UNIVERSITY

Computer Architecture and Operating Systems
Lecture 5: Assembly Programming – Branches and Arrays

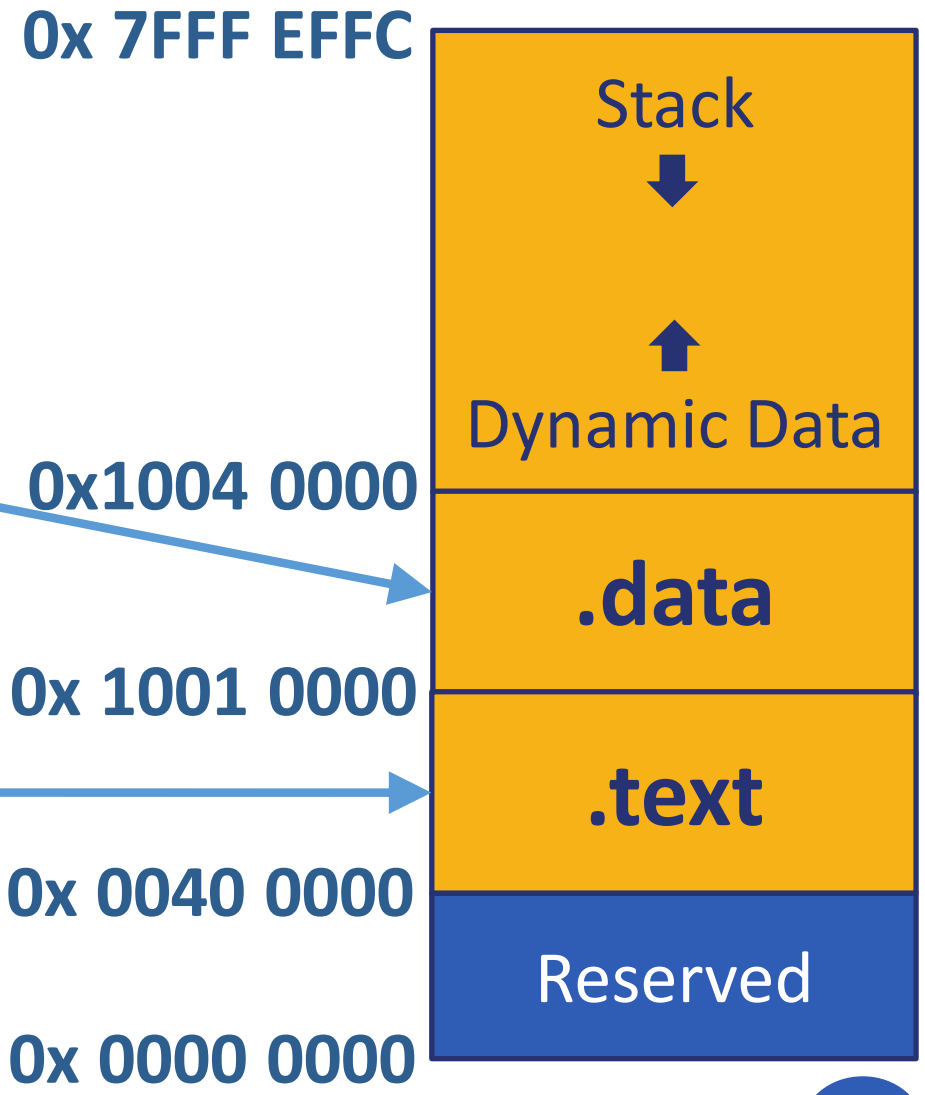
Andrei Tatarnikov

atatarnikov@hse.ru
@andrewt0301

Program Structure and Memory Layout

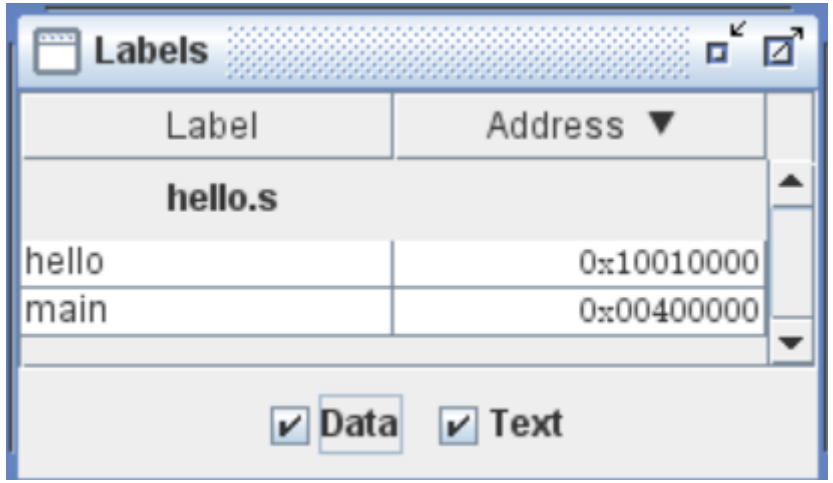
```
        .data
hello:
        .string "Hello, world!"

        .text
main:
        li a7, 4
        la a0, hello
        ecall
```



Labels

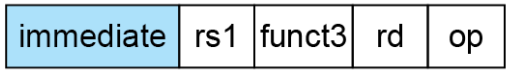
- **Labels** are symbolic names for addresses (in the .data or .text segment).
- **Labels** are used by control-flow instructions (branches and jumps).
- **Labels** are used by load and store instructions.



Addressing

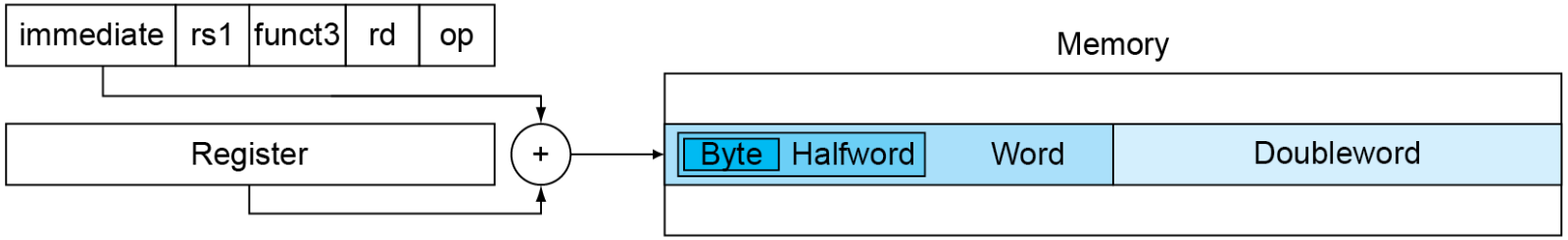
Addresses can be represented in several ways

1. Immediate addressing

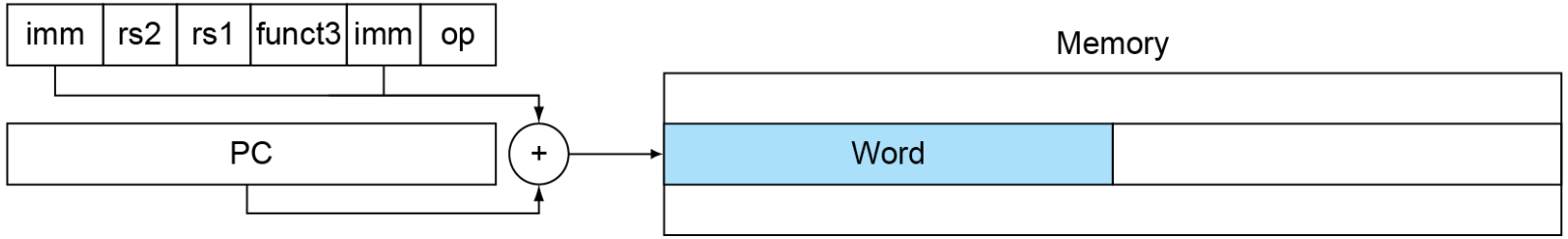


2. Register addressing

3. Base addressing



4. PC-relative addressing



Program Counter

- **Program Counter (PC)** is a special register that stores the address of the currently executed instruction.
- When an instruction is executed, the PC is incremented by the size of the instruction (4 bytes) to point to the next instruction.
- Branch and jump instructions assign to the PC new addresses to change the control flow.
- Branch instructions use PC-relative addresses (increment or decrement current value by an offset).

Branch Instructions

Branch Instructions

- Branch = beq rs1, rs2, *label*
- Branch ≠ bne rs1, rs2, *label*
- Branch < blt rs1, rs2, *label*
- Branch ≥ bge rs1, rs2, *label*
- Branch < Unsigned bltu rs1, rs2, *label*
- Branch ≥ Unsigned bgeu rs1, rs2, *label*

Branch Pseudoinstructions

Branch Pseudoinstructions

▪ Branch unconditionally	b	*label*
▪ Branch = 0	beqz	rs1, *label*
▪ Branch ≥ 0	bgez	rs1, *label*
▪ Branch >	bgt	rs1, rs2, *label*
▪ Branch > Unsigned	bgtu	rs1, rs2, *label*
▪ Branch > 0	bgtz	rs1, *label*
▪ Branch ≤	ble	rs1, rs2, *label*
▪ Branch ≤ Unsigned	bleu	rs1, rs2, *label*
▪ Branch ≤ 0	blez	rs1, *label*
▪ Branch < 0	bltz	rs1, *label*
▪ Branch ≠ 0	bnez	rs1, *label*

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
```