

Logika cyfrowa

Wykład 13: jednocyklowa implementacja RISC V

Marek Materzok

27 maja 2024

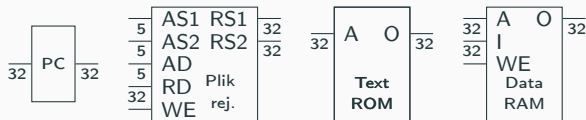
Implementacja jednocyklowa

Implementacja jednocyklowa – idea

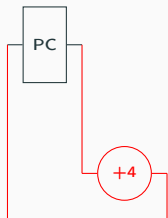
- Podział na ścieżkę sterowania i danych
- Każda instrukcja wykonywana w jednym cyklu zegara
- Bezstanowa ścieżka sterowania
- Architektura harwardzka (niezależna pamięć kodu i danych)

- Licznik instrukcji (PC – program counter)
- Plik rejestrów
 - 32 rejestry po 32 bity
 - Dwa porty do odczytu
 - Jeden port do zapisu
- Pamięć kodu (ROM)
- Pamięć danych (jednoportowa)

Elementy stanu architekturnego

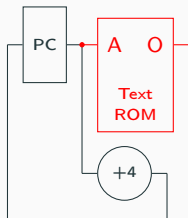






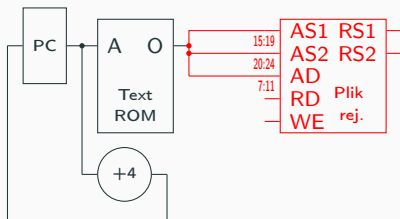
Adres następnej instrukcji

Ścieżka danych



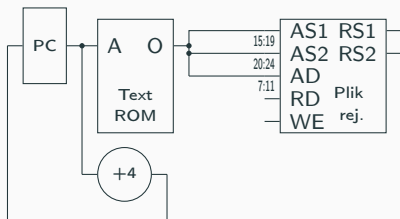
Pobranie instrukcji

Ścieżka danych



Wybór rejestrów

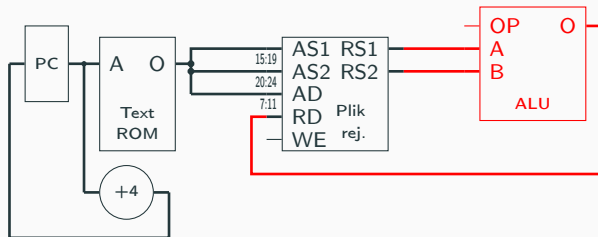
Ścieżka danych



Instrukcja OP

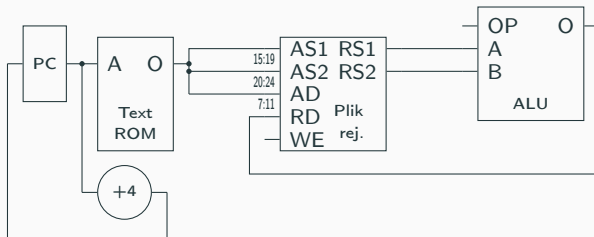
`add rd, rs1, rs2`

Ścieżka danych



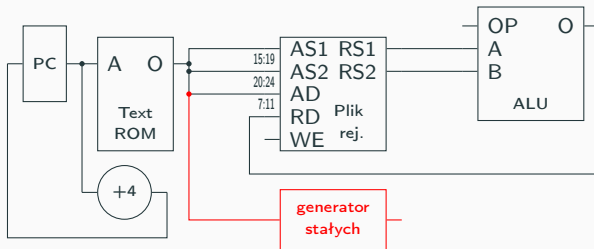
Instrukcja OP
add rd, rs1, rs2

Ścieżka danych



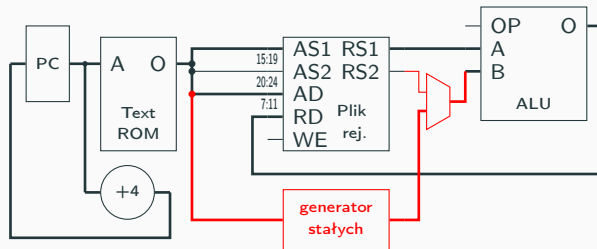
Instrukcja OP-IMM
`addi rd, rs1, imm`

Ścieżka danych



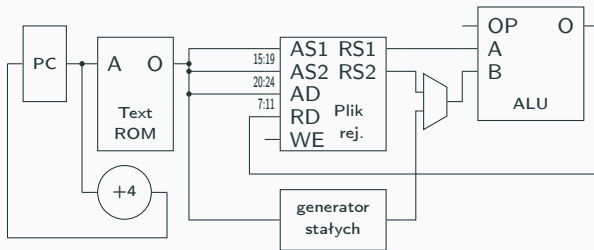
Instrukcja OP-IMM
`addi rd, rs1, imm`

Ścieżka danych



Instrukcja OP-IMM
`addi rd, rs1, imm`

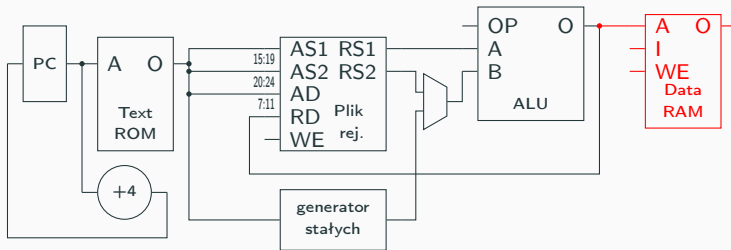
Ścieżka danych



Instrukcja STORE

sw rs2, imm(rs1)

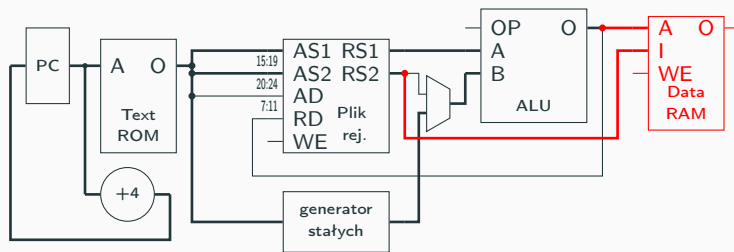
Ścieżka danych



Instrukcja STORE

sw rs2, imm(rs1)

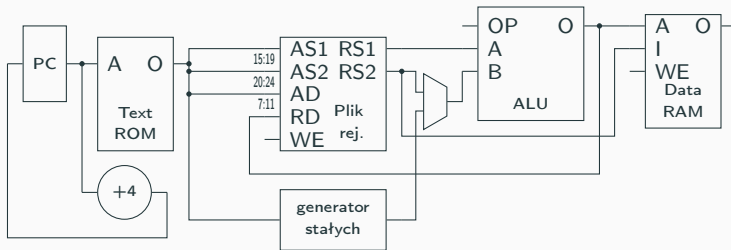
Ścieżka danych



Instrukcja STORE

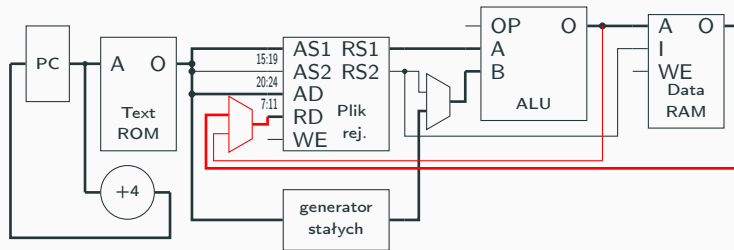
`sw rs2, imm(rs1)`

Ścieżka danych



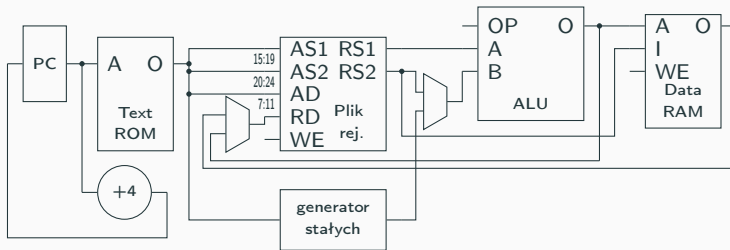
Instrukcja LOAD
`lw rd, imm(rs1)`

Ścieżka danych



Instrukcja LOAD
`lw rd, imm(rs1)`

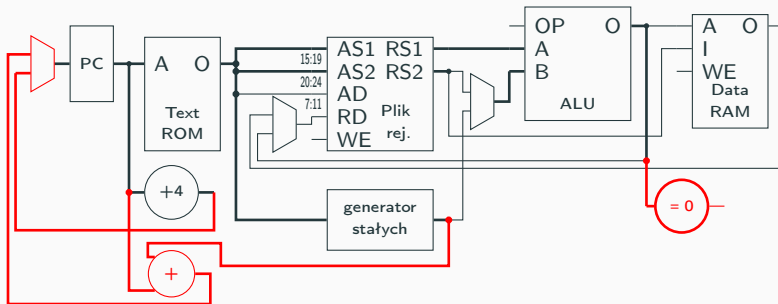
Ścieżka danych



Instrukcja BRANCH

`beq rs1, rs2, imm`

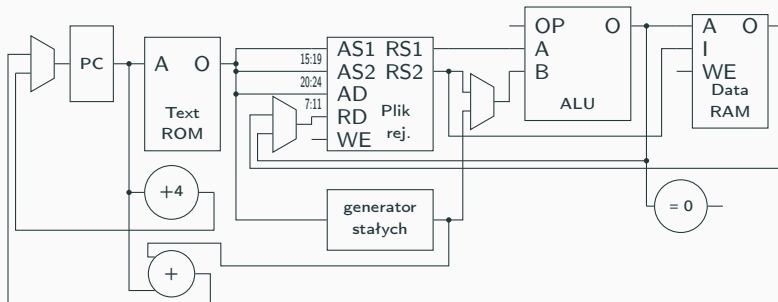
Ścieżka danych



Instrukcja BRANCH

`beq rs1, rs2, imm`

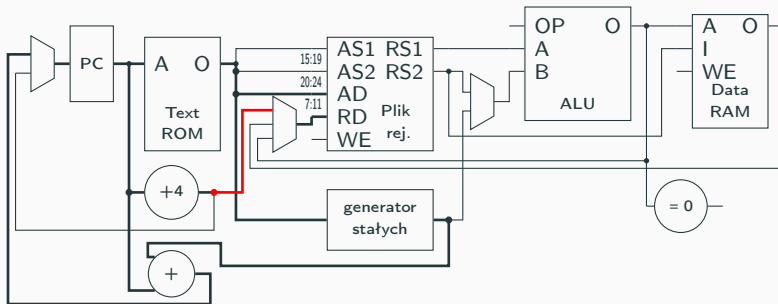
Ścieżka danych



Instrukcja JAL (Jump and Link)

`jal rd, imm`

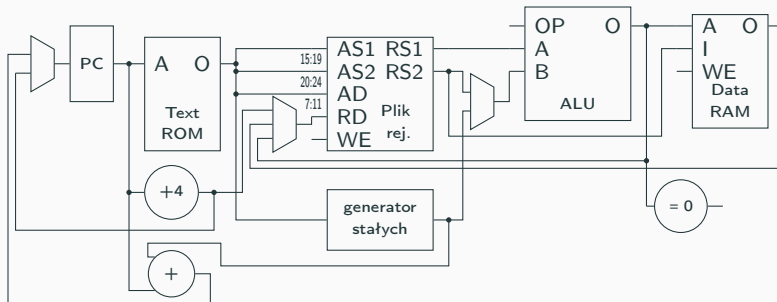
Ścieżka danych



Instrukcja JAL (Jump and Link)

jal rd, imm

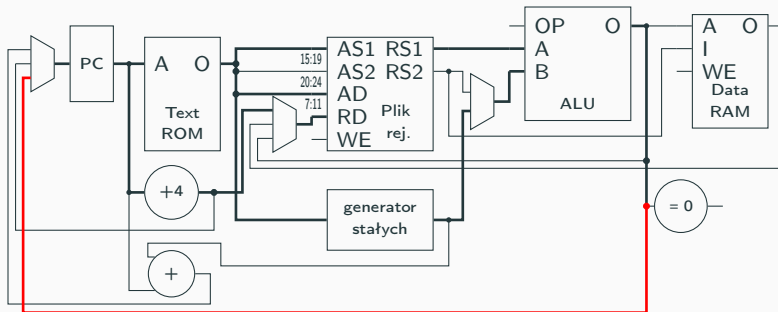
Ścieżka danych



Instrukcja JALR (Jump and Link Register)

`jalr rd, rs1, imm`

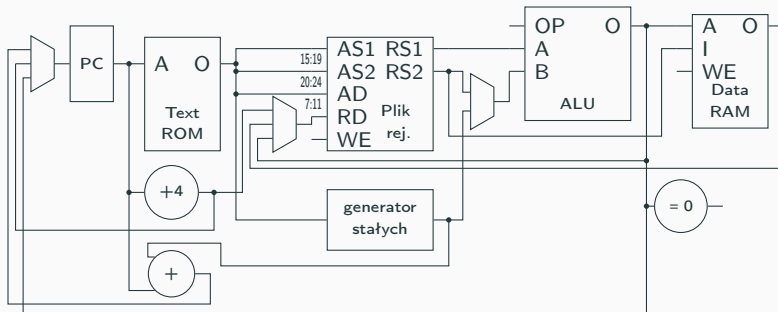
Ścieżka danych



Instrukcja JALR (Jump and Link Register)

`jlr rd, rs1, imm`

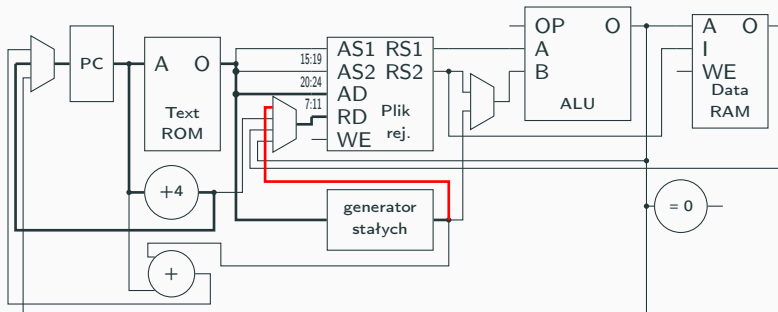
Ścieżka danych



Instrukcja LUI (Load Upper Immediate)

`lui rd, imm`

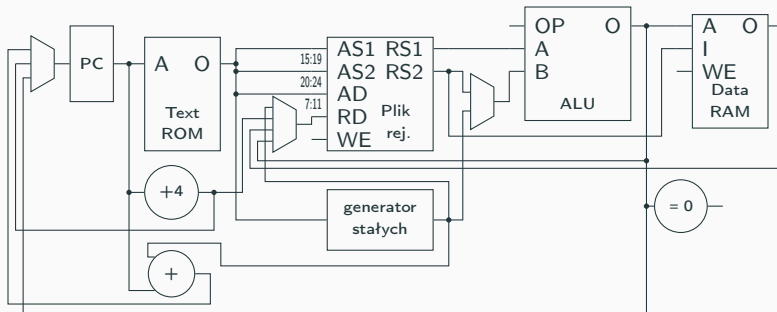
Ścieżka danych



Instrukcja LUI (Load Upper Immediate)

`lui rd, imm`

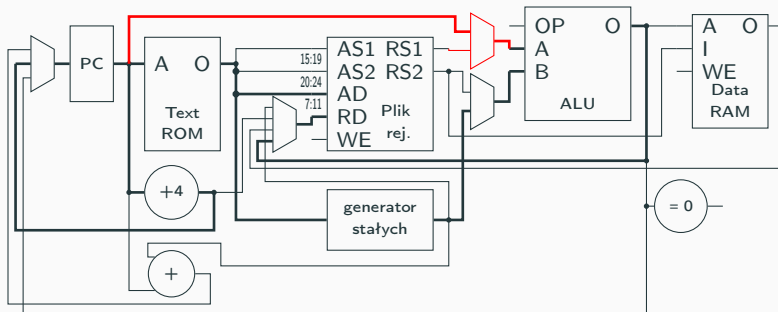
Ścieżka danych



Instrukcja AUIPC (Add Upper Immediate to Program Counter)

`auipc rd, imm`

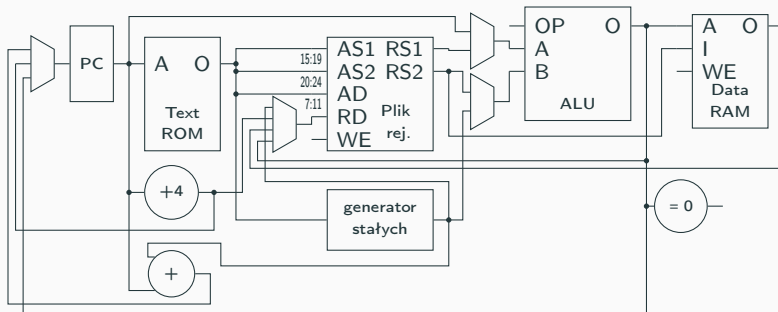
Ścieżka danych



Instrukcja AUIPC (Add Upper Immediate to Program Counter)

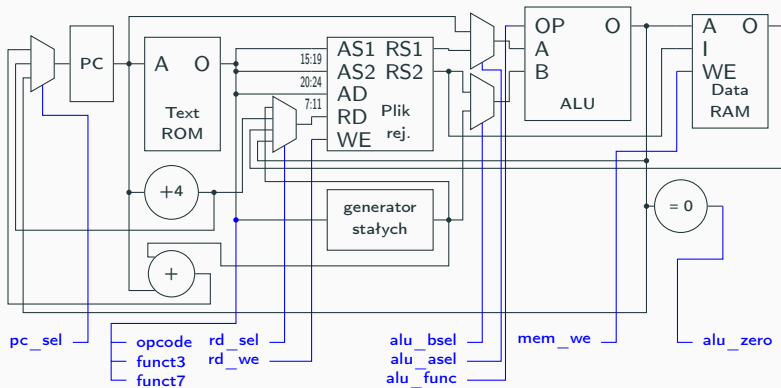
```
auipc rd, imm
```

Ścieżka danych



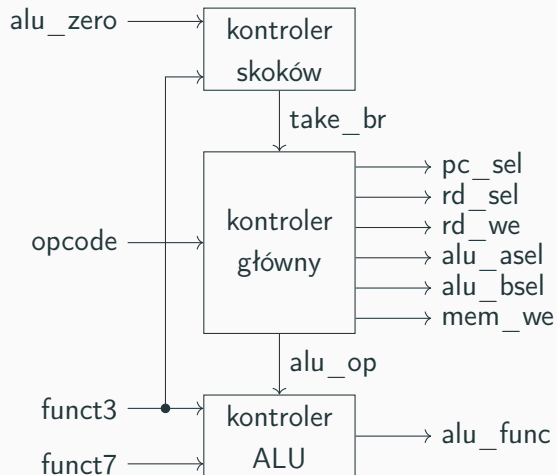
Gotowe!

Ścieżka danych



Sygnaly sterujące i statusu

Ścieżka sterowania



alu_zero	funct3	take_br
z	xx0 (beq,blt,bltu)	$\neg z$
z	xx1 (bne,bge,bgeu)	z

Kontroler ALU

alu_op	funct3	funct7	alu_func
ADD	xxx	xxxxxxx	ADD
BRANCH	00x	xxxxxxx	SEQ
BRANCH	10x	xxxxxxx	SLT
BRANCH	11x	xxxxxxx	SLTU
OP	000	x0xxxxx	ADD
OP	000	x1xxxxx	SUB
OP-IMM	000	xxxxxxx	ADD
OP, OP-IMM	001	xxxxxxx	SLL
OP, OP-IMM	010	xxxxxxx	SLT
OP, OP-IMM	011	xxxxxxx	SLTU
OP, OP-IMM	100	xxxxxxx	XOR
OP, OP-IMM	101	x0xxxxx	SRL
OP, OP-IMM	101	x1xxxxx	SRA
OP, OP-IMM	110	xxxxxxx	OR
OP, OP-IMM	111	xxxxxxx	AND

Kontroler główny

instr	opcode	take_br	pc_sel	rd_sel	rd_we	alu_ase	alu_bsel	alu_op	mem_we
OP	0110011	x	+4	ALU	1	RS1	RS2	OP	0
OP-IMM	0010011	x	+4	ALU	1	RS1	IMM	OP-IMM	0
BRANCH	1100011	0	+4	x	0	RS1	RS2	BRANCH	0
BRANCH	1100011	1	+IMM	x	0	RS1	RS2	BRANCH	0
JAL	1101111	x	+IMM	PC+4	1	x	x	x	0
JALR	1100111	x	ALU	PC+4	1	RS1	IMM	ADD	0
LOAD	0000011	x	+4	MEM	1	RS1	IMM	ADD	0
STORE	0100011	x	+4	x	0	RS1	IMM	ADD	1
LUI	0110111	x	+4	IMM	1	x	x	x	0
AUIPC	0010111	x	+4	ALU	1	PC	IMM	ADD	0

Implementacja w SystemVerilogu

<https://github.com/tilk/riscv-simple-sv>

- Prosta implementacja RV32I
- Zaimplementowanych kilka mikroarchitektur:
 - `singlecycle` – jednocykłowa (patrz: ten wykład)
 - `multicycle` – wielocykłowa
 - `pipeline` – potokowa
- Przetestowana przy użyciu oficjalnych testów (katalog tests)

- Zawiera: narzędzia binutils (assembler as, linker ld, itd.), kompilator gcc i inne
- Pakiety Debiana (wersja 11 „Bullseye”):
 - `binutils-riscv64-unknown-elf`
 - `gcc-riscv64-unknown-elf`
- Znajomość narzędzi nie jest wymagana – więcej na przedmiocie ASK

- Utworzyć nowy katalog:

```
$ mkdir sim
```

- Skopiować tam pliki wspólne i mikroarchitektury jednocyklowej:

```
$ cp core/common/* sim
```

```
$ cp core/singlecycle/* sim
```

- Dostosować plik konfiguracyjny `config.sv`
- Załadować wszystkie pliki z katalogu `sim` do DigitalJS (włącznie z plikami z kodem i danymi, jeśli użyte)

Kompilacja programów na riscv-simple-sv

Dla zainteresowanych!

- Plik `riscv-compile.tar` (na Canvasie) zawiera `Makefile`, skrypt linkera `link.ld` i plik `crt0.S` inicjalizujący wskaźnik stosu
- Poza narzędziami dla RISC V wymagany pakiet `srecord`
- Kompilacja generuje pliki obrazu pamięci kodu i danych, `test.text.vh` i `test.data.vh`
- Pliki obrazu można wgrać do katalogu `sim` z poprzedniego slajdu, wprowadzić nazwy plików obrazu do `config.sv`, i wysłać wszystko na DigitalJS