

Logika cyfrowa

Wykład 14: wielocyklowa implementacja RISC V

Marek Materzok

9 czerwca 2021

Implementacja wielocyklowa

Implementacja wielocyklowa – idea

- Podział na ścieżkę sterowania i danych
- Instrukcje wykonywane w wielu cyklach zegara (różna liczba cykli dla różnych instrukcji)
- Pośrednie wyniki zapisywane w dodatkowych rejestrach (niearchitekturalnych)
- Stanowa ścieżka sterowania – automat skończony
- Architektura von Neumanna – wspólna pamięć dla kodu i danych, zapytania o kod i dane w osobnych cyklach

Implementacja jedno- a wielocyklowa

- Implementacja jednocyklowa
 - Plus: prosta
 - Minus: długość cyklu ograniczona czasem wykonania najdłuższej instrukcji (LOAD)
 - Minus: dwa sumatory, dwie szyny pamięci
- Implementacja wielocyklowa
 - Plus: szybsze taktowanie
 - Plus: proste instrukcje wykonywane szybciej od skomplikowanych
 - Plus: reużycie dużych elementów (np. ALU) pomiędzy cyklami
 - Minus: koszt sekwencjonowania płacony wielokrotnie

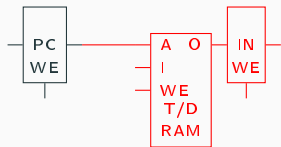
Elementy stanu architekuralnego

Wspólna pamięć dla kodu i danych (architektura von Neumanna):

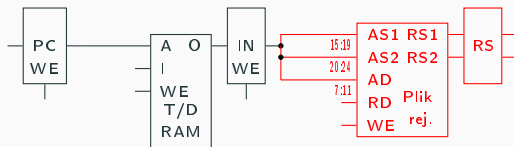




Ścieżka danych

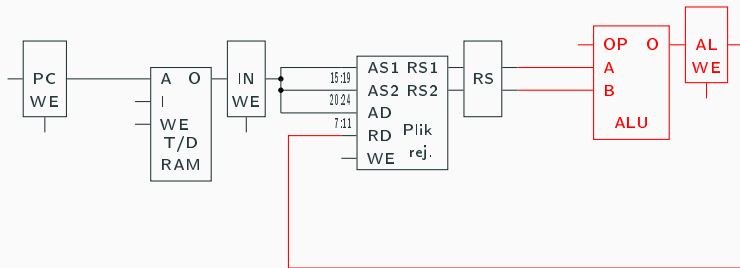


Pobranie instrukcji



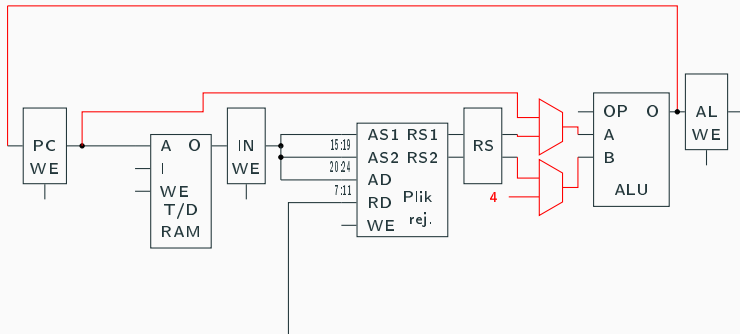
Odczyt rejestrów

Ścieżka danych



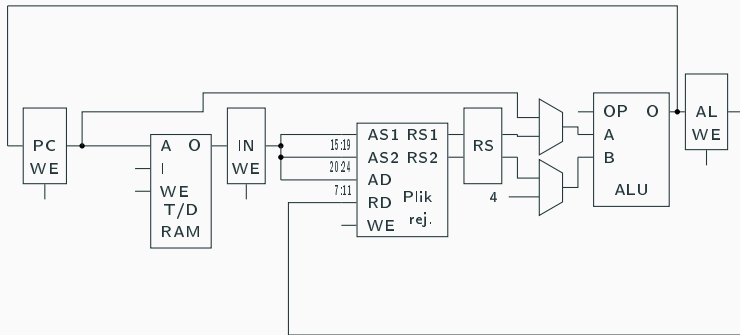
Instrukcja OP
add rd, rs1, rs2

Ścieżka danych



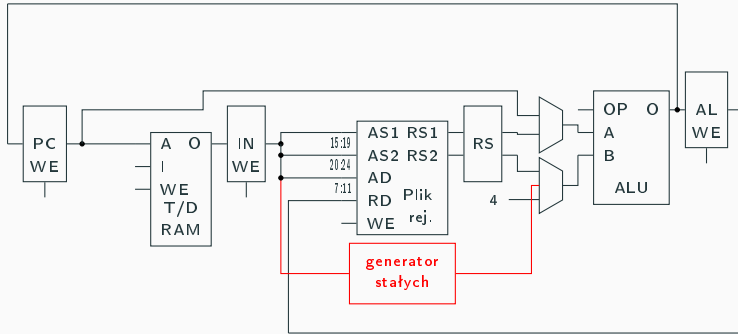
Adres następnej instrukcji

Ścieżka danych



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

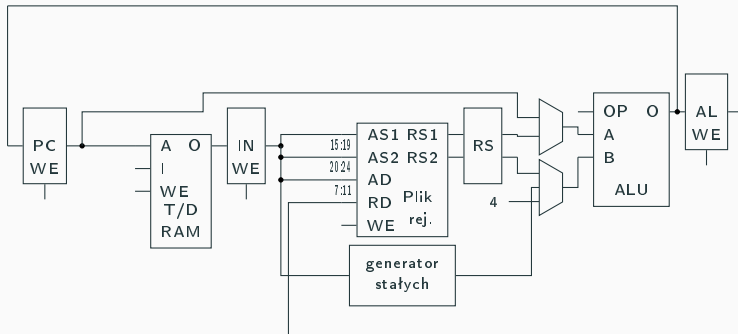
Ścieżka danych



Instrukcja OP-IMM

```
addi rd, rs1, imm
```

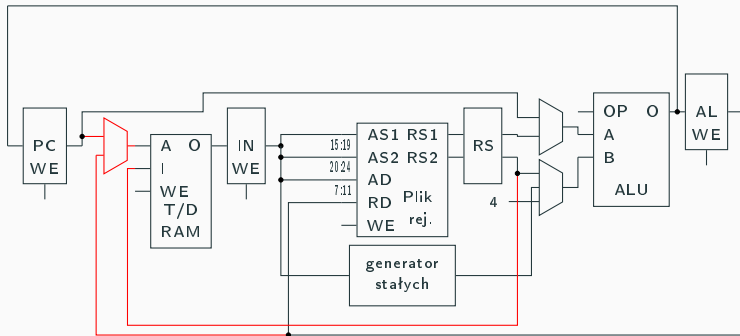
Ścieżka danych



Instrukcja STORE

sw rs2, imm(rs1)

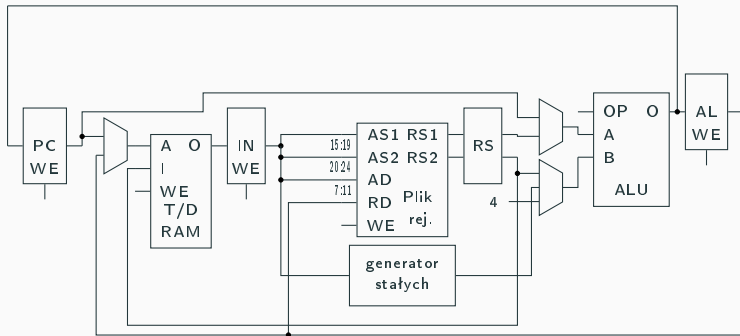
Ścieżka danych



Instrukcja STORE

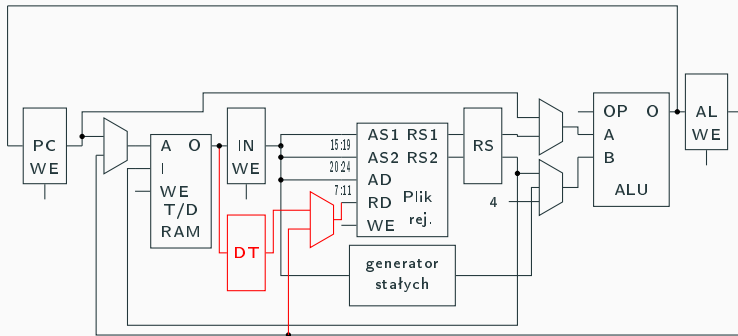
sw rs2, imm(rs1)

Ścieżka danych



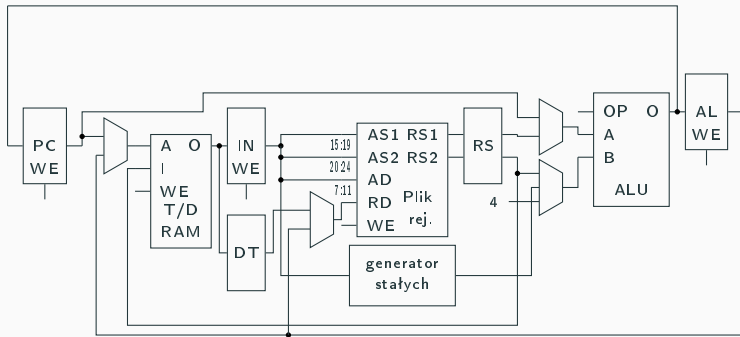
Instrukcja LOAD
`lw rd, imm(rs1)`

Ścieżka danych



Instrukcja LOAD
`lw rd, imm(rs1)`

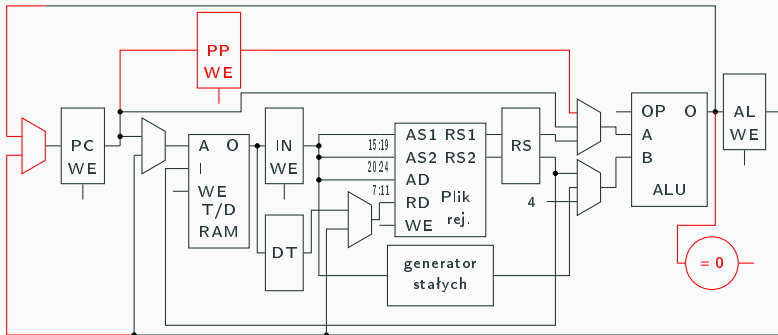
Ścieżka danych



Instrukcja BRANCH

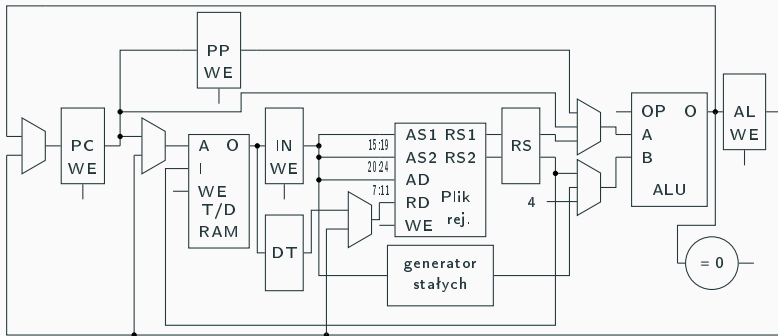
```
beq rs1, rs2, imm
```

Ścieżka danych



Instrukcja BRANCH
beq rs1, rs2, imm

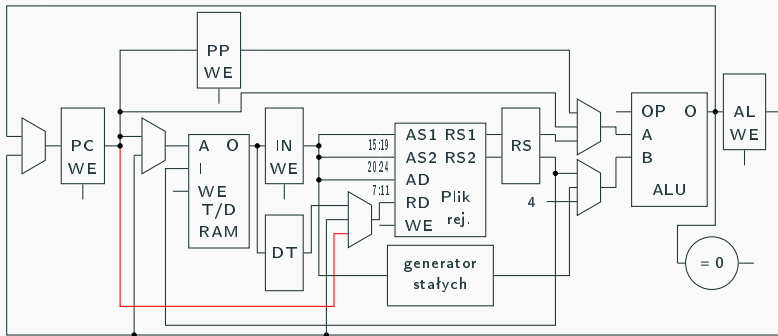
Ścieżka danych



Instrukcja JAL (Jump and Link)

`jal rd, imm`

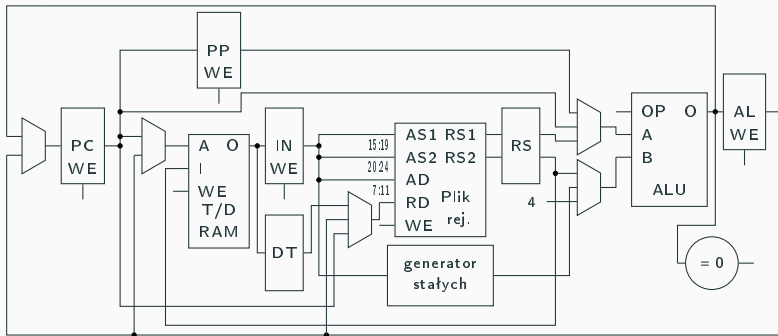
Ścieżka danych



Instrukcja JAL (Jump and Link)

`jal rd, imm`

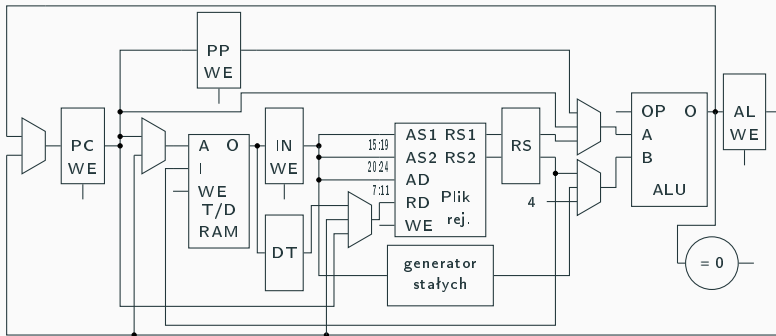
Ścieżka danych



Instrukcja JALR (Jump and Link Register)

```
jalr rd, rs1, imm
```

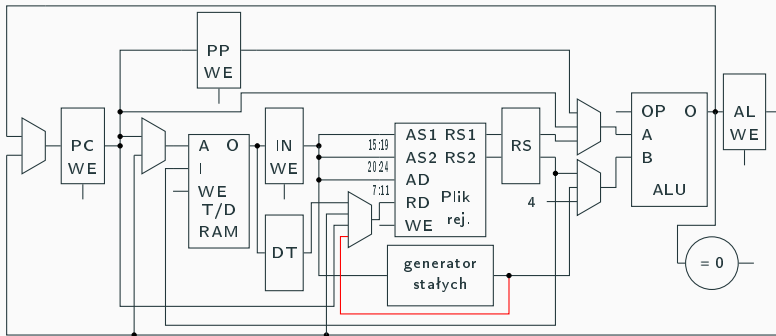
Ścieżka danych



Instrukcja LUI (Load Upper Immediate)

`lui rd, imm`

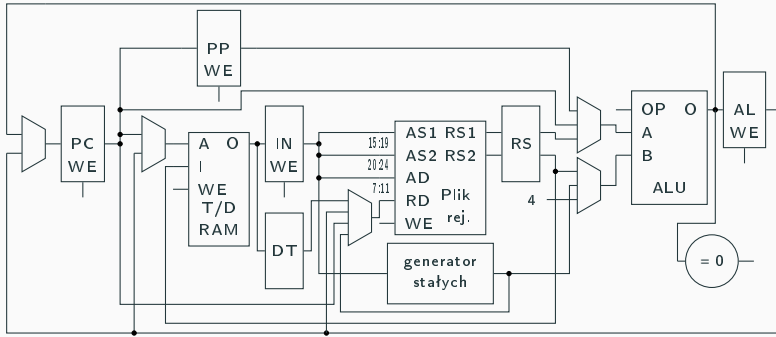
Ścieżka danych



Instrukcja LUI (Load Upper Immediate)

`lui rd, imm`

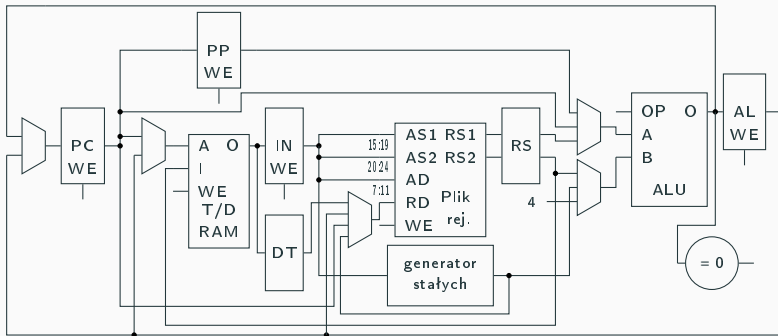
Ścieżka danych



Instrukcja AUIPC (Add Upper Immediate to Program Counter)

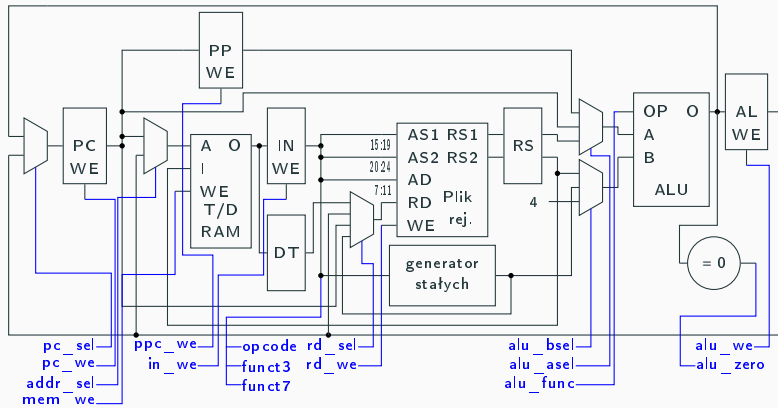
```
auipc rd, imm
```


Ścieżka danych



Gotowe!

Ścieżka danych



Sygnaly sterujące i statusu

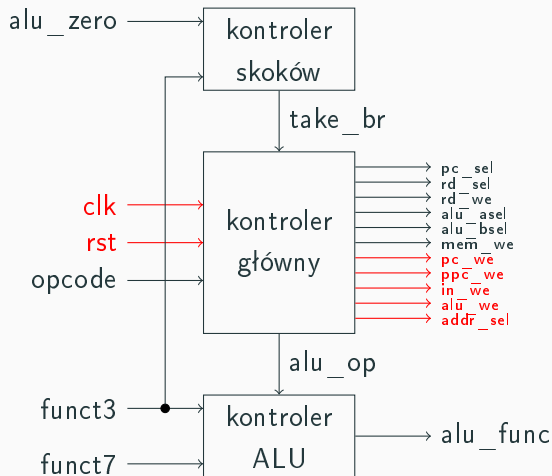
Jednocyklowa:

- 0 dodatkowych rejestrów
- 2 dodatkowe sumatory
- 4 multipleksery
- 7 sygn. sterujących

Wielocyklowa:

- 6 dodatkowych rejestrów
- 0 dodatkowych sumatorów
- 5 multiplekserów
- 12 sygn. sterujących

Ścieżka sterowania



- Kontroler skoków i ALU bez zmian
- Kontroler główny automatem skończonym
 - Sekwencje stanów zależne od instrukcji
 - Różna liczba cykli dla różnych instrukcji
 - Kontrola nad rejestrami niearchitekturalnymi (przechowującymi „wyniki pośrednie”)

Przykład – instrukcja OP (add)

1. $IN \leftarrow \text{mem}[PC]$
 $PC \leftarrow PC + 4$

FETCH

Przykład – instrukcja OP (add)

1. $IN \leftarrow \text{mem}[PC]$

$PC \leftarrow PC + 4$

2. $RS1 \leftarrow \text{reg}[AS1]$

$RS2 \leftarrow \text{reg}[AS2]$

FETCH

DECODE

Przykład – instrukcja OP (add)

1. $IN \leftarrow \text{mem}[PC]$ $PC \leftarrow PC + 4$	FETCH
2. $RS1 \leftarrow \text{reg}[AS1]$ $RS2 \leftarrow \text{reg}[AS2]$	DECODE
3. $AL \leftarrow RS1 + RS2$	EXECUTE

Przykład – instrukcja OP (add)

1. $IN \leftarrow \text{mem}[PC]$ $PC \leftarrow PC + 4$	FETCH
2. $RS1 \leftarrow \text{reg}[AS1]$ $RS2 \leftarrow \text{reg}[AS2]$	DECODE
3. $AL \leftarrow RS1 + RS2$	EXECUTE
4. $\text{reg}[AD] \leftarrow AL$	ALU_WRITEBACK

Przykład – instrukcja OP, sygnały sterujące

- | | |
|---|-----------------------|
| 1. <code>addr_sel = PC</code>
<code>alu_op = ADD, alu_asel = PC, alu_bsel = 4</code>
<code>pc_sel = ALU</code>
<code>in_we, pc_we</code> | FETCH |
| 2. — | DECODE |
| 3. <code>alu_op = OP, alu_asel = RS1, alu_bsel = RS2</code>
<code>alu_we</code> | <i>EXECUTE</i> |
| 4. <code>rd_sel = AL</code>
<code>rd_we</code> | ALU_WRITEBACK |

Przykład – instrukcja LOAD

1. $IN \leftarrow \text{mem}[PC]$
 $PC \leftarrow PC + 4$

FETCH

Przykład – instrukcja LOAD

1. $IN \leftarrow \text{mem}[PC]$

$PC \leftarrow PC + 4$

2. $RS1 \leftarrow \text{reg}[AS1]$

$RS2 \leftarrow \text{reg}[AS2]$

FETCH

DECODE

Przykład – instrukcja LOAD

1. $IN \leftarrow \text{mem}[PC]$

$PC \leftarrow PC + 4$

FETCH

2. $RS1 \leftarrow \text{reg}[AS1]$

$RS2 \leftarrow \text{reg}[AS2]$

DECODE

3. $AL \leftarrow RS1 + IMM$

MEM_ADDR

Przykład – instrukcja LOAD

1. $IN \leftarrow \text{mem}[PC]$ $PC \leftarrow PC + 4$	FETCH
2. $RS1 \leftarrow \text{reg}[AS1]$ $RS2 \leftarrow \text{reg}[AS2]$	DECODE
3. $AL \leftarrow RS1 + IMM$	MEM_ADDR
4. $DT \leftarrow \text{mem}[AL]$	MEM_READ

Przykład – instrukcja LOAD

1. $IN \leftarrow \text{mem}[PC]$ $PC \leftarrow PC + 4$	FETCH
2. $RS1 \leftarrow \text{reg}[AS1]$ $RS2 \leftarrow \text{reg}[AS2]$	DECODE
3. $AL \leftarrow RS1 + IMM$	MEM_ADDR
4. $DT \leftarrow \text{mem}[AL]$	MEM_READ
5. $\text{reg}[AD] \leftarrow DT$	MEM_WRITEBACK

Przykład – instrukcja LOAD, sygnały sterujące

1. <code>addr_sel = PC</code> <code>alu_op = ADD, alu_asel = PC, alu_bsel = 4</code> <code>pc_sel = ALU</code> <code>in_we, pc_we</code>	FETCH
2. —	DECODE
3. <code>alu_we</code> <code>alu_op = ADD, alu_asel = RS1, alu_bsel = IMM</code>	MEM_ADDR
4. <code>addr_sel = AL</code>	MEM_READ
5. <code>rd_sel = DT</code> <code>rd_we</code>	MEM_WRITEBACK

Przykład – instrukcja BRANCH (beq)

1. $IN \leftarrow \text{mem}[PC]$

$PP \leftarrow PC$

$PC \leftarrow PC + 4$

FETCH

Przykład – instrukcja BRANCH (beq)

1. $IN \leftarrow \text{mem}[PC]$

$PP \leftarrow PC$

$PC \leftarrow PC + 4$

FETCH

2. $RS1 \leftarrow \text{reg}[AS1]$

$RS2 \leftarrow \text{reg}[AS2]$

$AL \leftarrow PP + IMM$

DECODE

Przykład – instrukcja BRANCH (beq)

- | | |
|---|--------|
| 1. $IN \leftarrow \text{mem}[PC]$
$PP \leftarrow PC$
$PC \leftarrow PC + 4$ | FETCH |
| 2. $RS1 \leftarrow \text{reg}[AS1]$
$RS2 \leftarrow \text{reg}[AS2]$
$AL \leftarrow PP + IMM$ | DECODE |
| 3. jeśli $RS1 = RS2, PC \leftarrow AL$ | BRANCH |

Przykład – instrukcja BRANCH (beq)

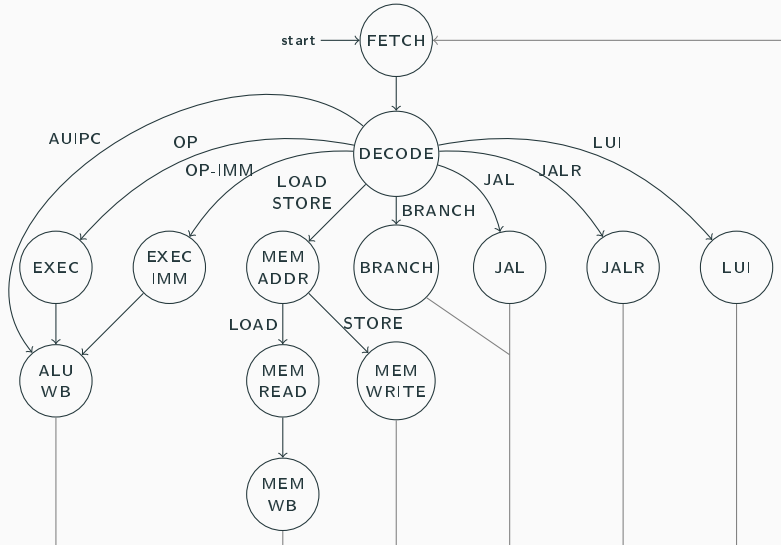
- | | |
|---|--------|
| 1. $IN \leftarrow \text{mem}[PC]$
$PP \leftarrow PC$
$PC \leftarrow PC + 4$ | FETCH |
| 2. $RS1 \leftarrow \text{reg}[AS1]$
$RS2 \leftarrow \text{reg}[AS2]$
$AL \leftarrow PP + IMM$ | DECODE |
| 3. jeśli $RS1 = RS2$, $PC \leftarrow AL$ | BRANCH |

Operacje **czerwone** trzeba wykonać **zanim** kontroler się dowie, że ma wykonywać BRANCH!

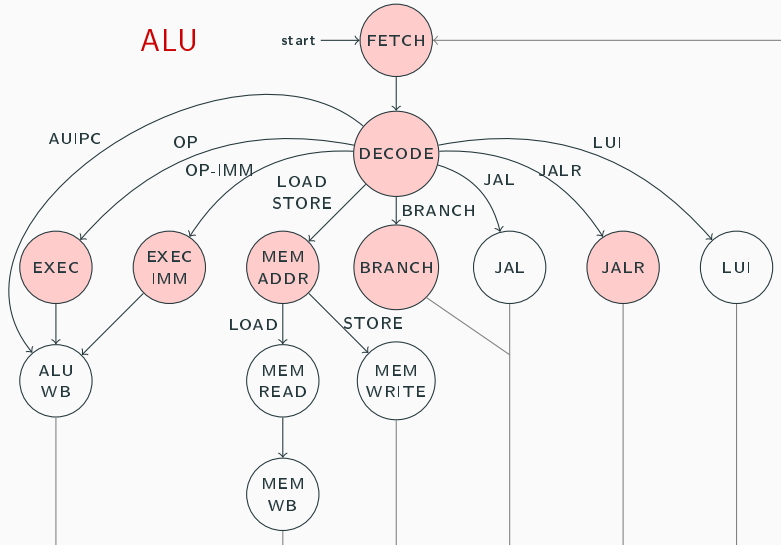
Przykład – instrukcja BRANCH, sygnały sterujące

- | | |
|--|--------|
| 1. <code>addr_sel = PC</code>
<code>alu_op = ADD, alu_asel = PC, alu_bsel = 4</code>
<code>pc_sel = ALU</code>
<code>in_we, pc_we, pp_we</code> | FETCH |
| 2. <code>alu_op = ADD, alu_asel = PP, alu_bsel = IMM</code>
<code>alu_we</code> | DECODE |
| 3. <code>pc_sel = AL</code>
<code>pc_we = take_br</code>
<code>alu_op = BRANCH, alu_asel = RS1, alu_bsel = RS2</code> | BRANCH |

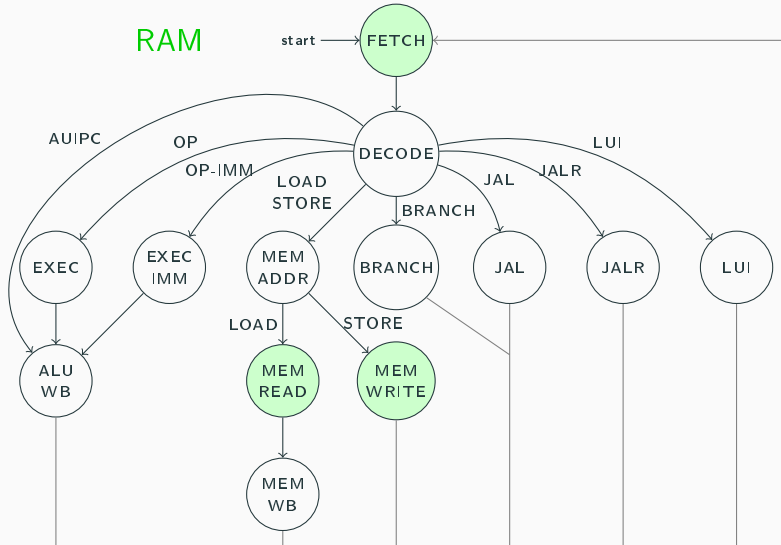
Automat skończony



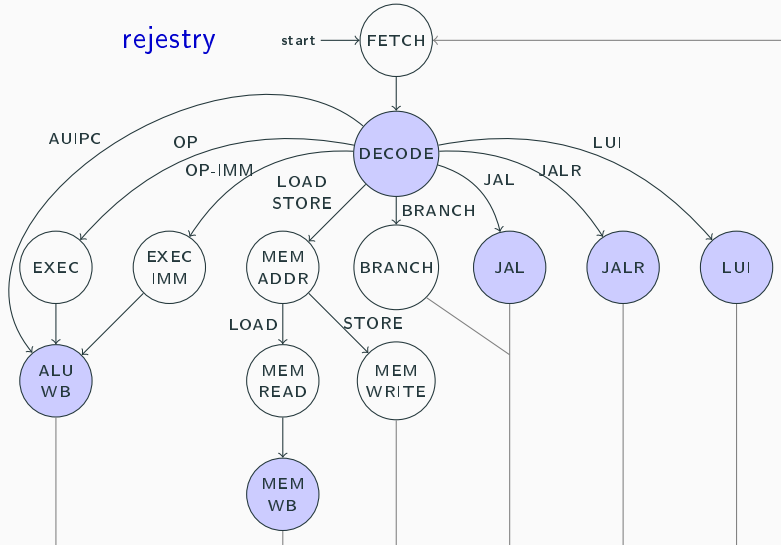
Automat skończony



Automat skończony

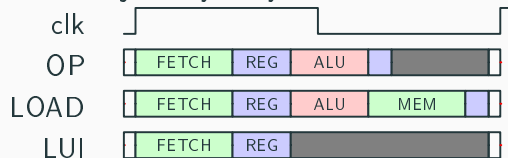


Automat skończony



Aspekty wydajnościowe

- Procesor jednocyklowy:



- Procesor wielocyklowy:

