Logika cyfrowa

Wykład 14: wielocyklowa implementacja RISC V

Marek Materzok

3 czerwca 2024

Implementacja wielocyklowa

Implementacja wielocyklowa – idea

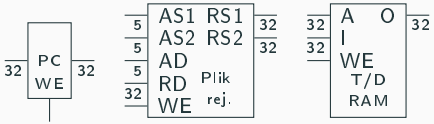
- Podział na ścieżkę sterowania i danych
- Instrukcje wykonywane w wielu cyklach zegara (różna liczba cykli dla różnych instrukcji)
- Pośrednie wyniki zapisywane w dodatkowych rejestrach (niearchitekturalnych)
- Stanowa ścieżka sterowania – automat skończony
- Architektura von Neumanna – wspólna pamięć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

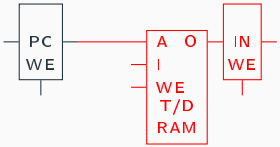
Elementy stanu architekturalnego

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



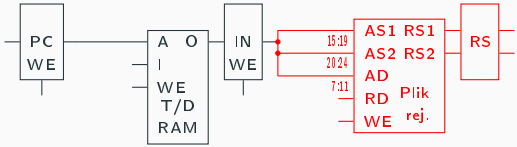
Ścieżka danych

Ścieżka danych



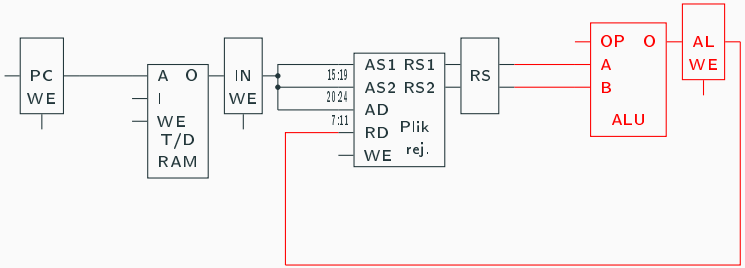
Pobranie instrukcji

Ścieżka danych



Odczyt rejestrów

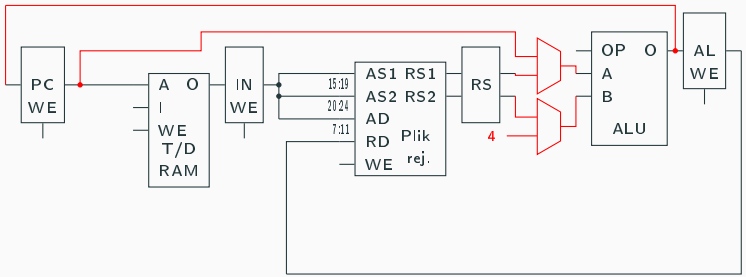
Ścieżka danych



Instrukcja OP

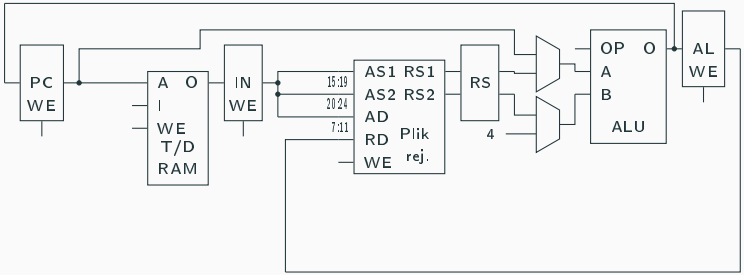
```
add rd, rs1, rs2
```

Ścieżka danych



Adres następnej instrukcji

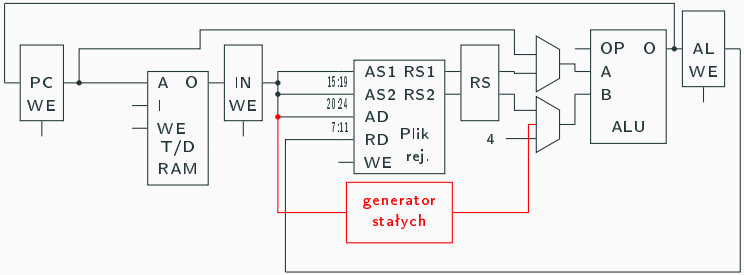
Ścieżka danych



Instrukcja OP-IMM

`addi rd, rs1, imm`

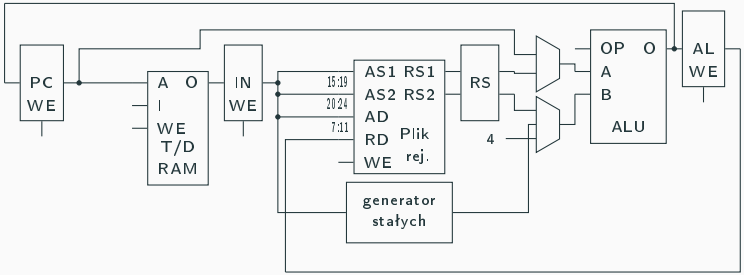
Ścieżka danych



Instrukcja OP-IMM

`addi rd, rs1, imm`

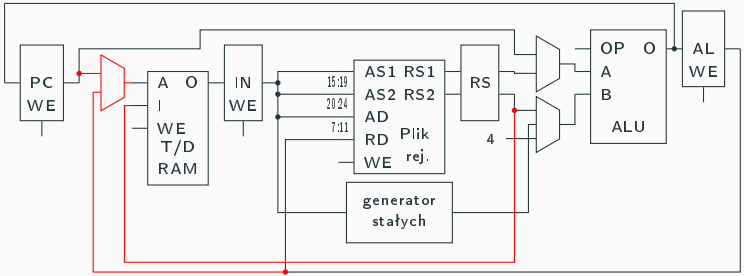
Ścieżka danych



Instrukcja STORE

`sw rs2, imm(rs1)`

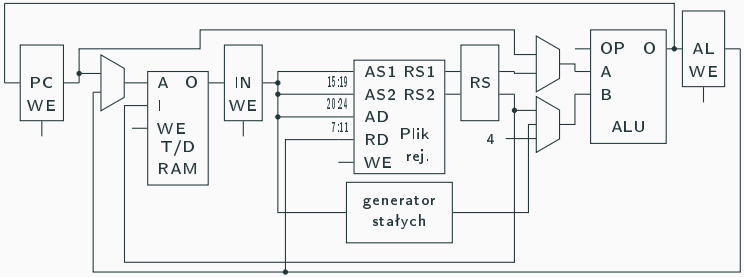
Ścieżka danych



Instrukcja STORE

`sw rs2, imm(rs1)`

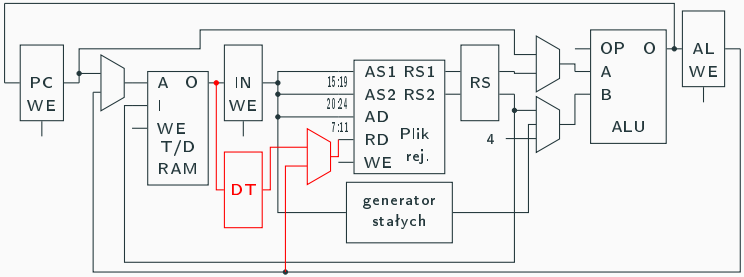
Ścieżka danych



Instrukcja LOAD

`lw rd, imm(rs1)`

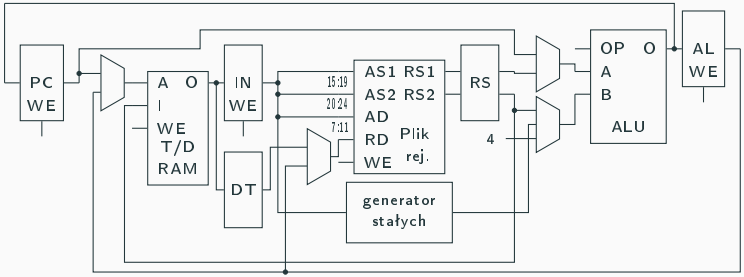
Ścieżka danych



Instrukcja LOAD

`lw rd, imm(rs1)`

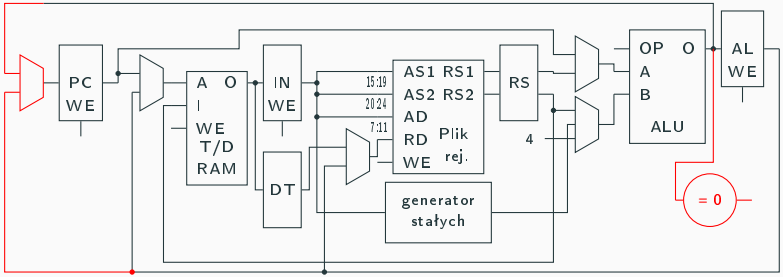
Ścieżka danych



Instrukcja BRANCH

`beq rs1, rs2, imm`

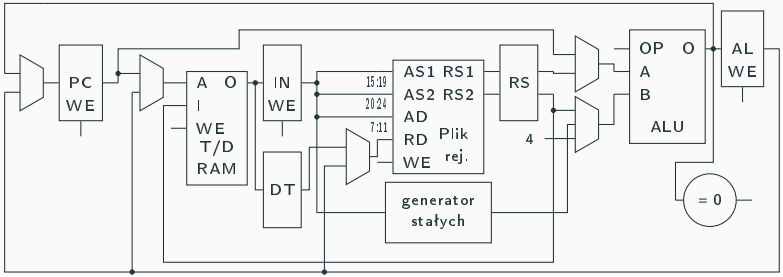
Ścieżka danych



Instrukcja BRANCH

`beq rs1, rs2, imm`

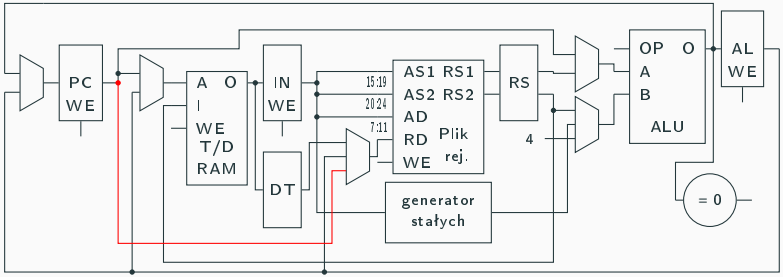
Ścieżka danych



Instrukcja JAL (Jump and Link)

`jal rd, imm`

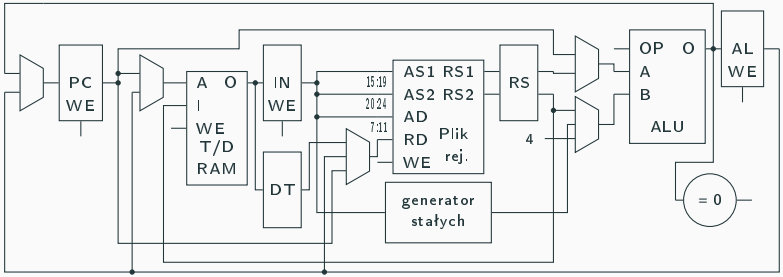
Ścieżka danych



Instrukcja JAL (Jump and Link)

`jal rd, imm`

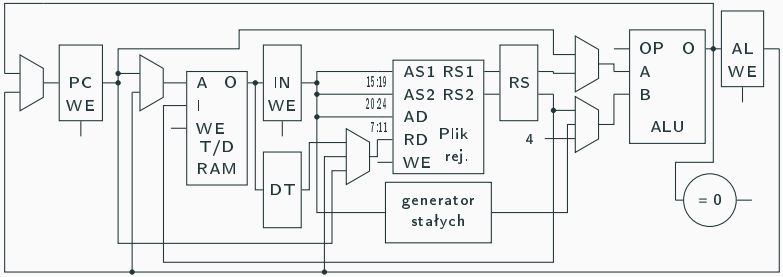
Ścieżka danych



Instrukcja JALR (Jump and Link Register)

`jalr rd, rs1, imm`

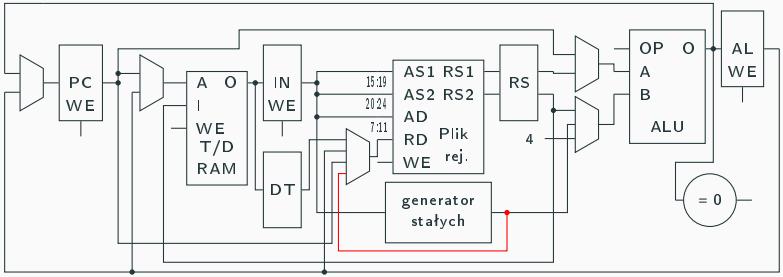
Ścieżka danych



Instrukcja LUI (Load Upper Immediate)

`lui rd, imm`

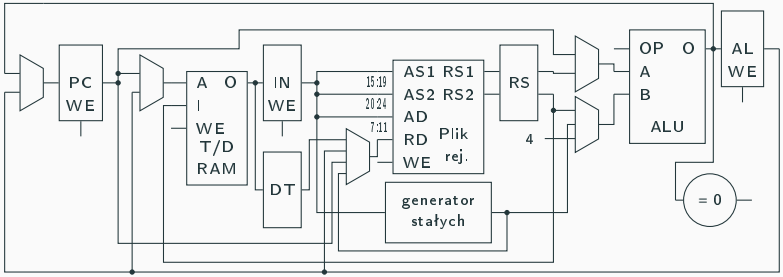
Ścieżka danych



Instrukcja LUI (Load Upper Immediate)

`lui rd, imm`

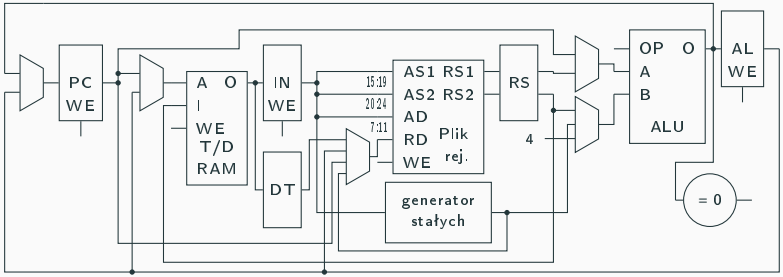
Ścieżka danych



Instrukcja AUIPC (Add Upper Immediate to Program Counter)

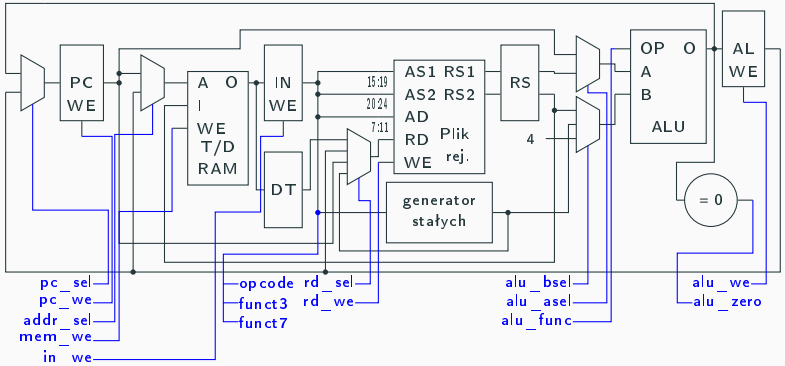
`auipc rd, imm`

Ścieżka danych



Gotowe!

Ścieżka danych



Sygnały sterujące i statusu

Porównanie ścieżek danych

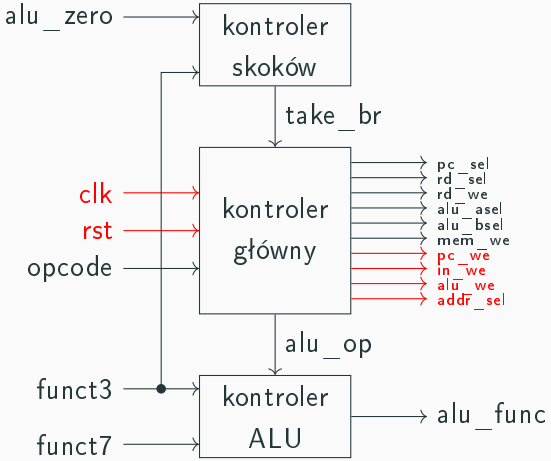
Jednocyklowa:

- 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



Kontrolery

- 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. $\mathrm{IN} \leftarrow \mathrm{mem}[\mathrm{PC}]$
   $\mathrm{AL} \leftarrow \mathrm{PC} + 4$

FETCH

Przykład – instrukcja OP (add)

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

Przykład – instrukcja OP (add)

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

Przykład – instrukcja OP (add)

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

Przykład – instrukcja OP, sygnały sterujące

1. addr_sel = PC — FETCH
   alu_op = ADD, alu_asel = PC, alu_bsel = 4
   in_we, alu_we
2. pc_sel = ALU — DECODE
   pc_we
3. alu_op = OP, alu_asel = RS1, alu_bsel = RS2 — *EXECUTE*
   alu_we
4. rd_sel = AL — ALU_WRITEBACK
   rd_we

Przykład – instrukcja LOAD

1. $\mathrm{IN} \leftarrow \mathrm{mem}[\mathrm{PC}]$ FETCH
   $\mathrm{AL} \leftarrow \mathrm{PC} + 4$

Przykład – instrukcja LOAD

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

Przykład – instrukcja LOAD

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

Przykład – instrukcja LOAD

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

Przykład – instrukcja LOAD

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

Przykład – instrukcja LOAD, sygnały sterujące

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

Przykład – instrukcja BRANCH (beq)

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

FETCH

Przykład – instrukcja BRANCH (beq)

1. $IN \leftarrow mem[PC]$ FETCH
   $AL \leftarrow PC + 4$
2. $RS1 \leftarrow reg[AS1]$ DECODE
   $RS2 \leftarrow reg[AS2]$
   $PC \leftarrow AL$
   $AL \leftarrow PC + IMM$

Przykład – instrukcja BRANCH (beq)

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

Przykład – instrukcja BRANCH (beq)

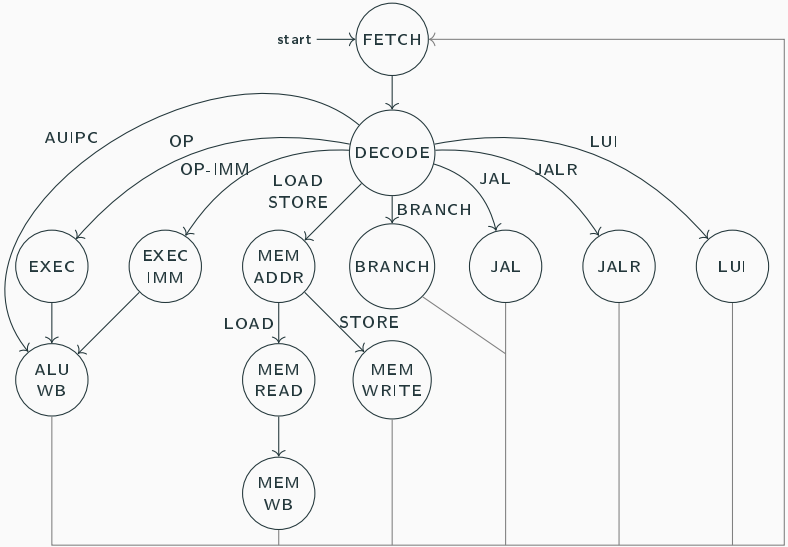
1. $\mathrm{IN} \leftarrow \mathrm{mem}[\mathrm{PC}]$ FETCH
   $\mathrm{AL} \leftarrow \mathrm{PC} + 4$
2. $\mathrm{RS1} \leftarrow \mathrm{reg}[\mathrm{AS1}]$ DECODE
   $\mathrm{RS2} \leftarrow \mathrm{reg}[\mathrm{AS2}]$
   $\mathrm{PC} \leftarrow \mathrm{AL}$
   $\mathrm{AL} \leftarrow \mathrm{PC}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

Operację czerwoną trzeba wykonać **zanim** kontroler się dowie, że ma wykonywać BRANCH!

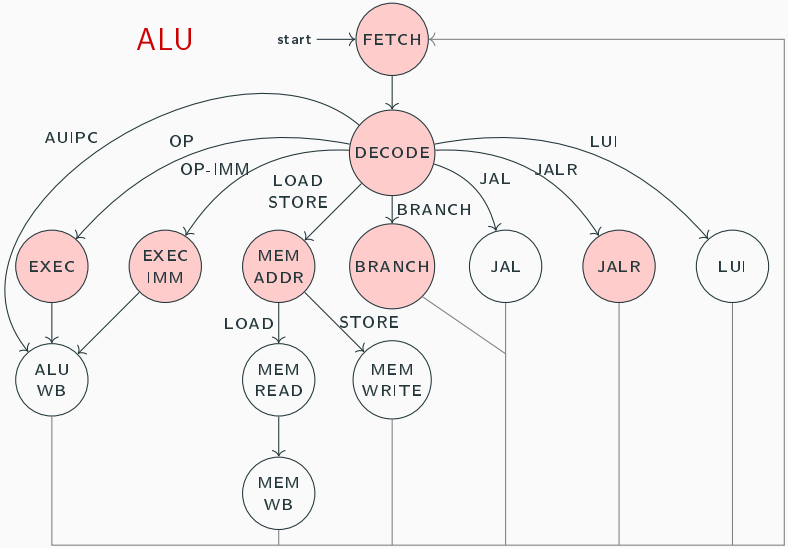
Przykład – instrukcja BRANCH, sygnały sterujące

1. addr_sel = PC — FETCH
   alu_op = ADD, alu_asel = PC, alu_bsel = 4
   pc_sel = ALU
   in_we, alu_we
2. pc_sel = ALU — DECODE
   alu_op = ADD, alu_asel = PC, alu_bsel = IMM
   pc_we, alu_we
3. pc_sel = AL — BRANCH
   pc_we = take_br
   alu_op = BRANCH, alu_asel = RS1, alu_bsel = RS2

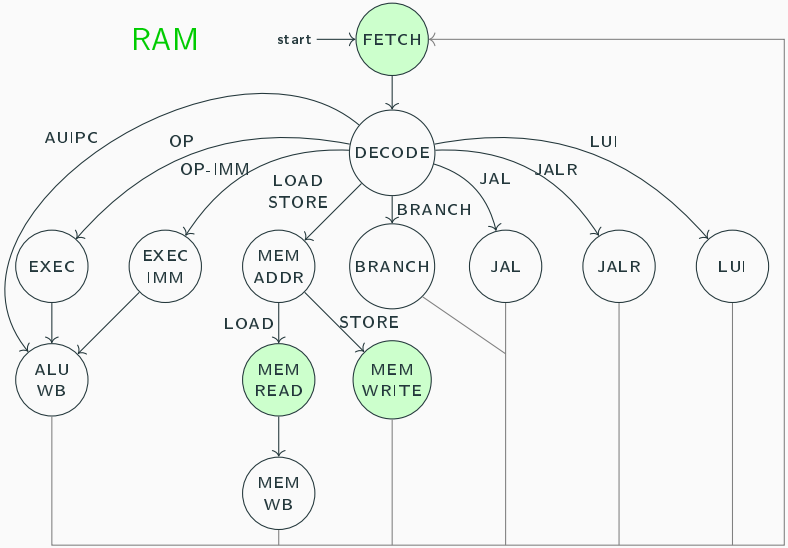
Automat skończony



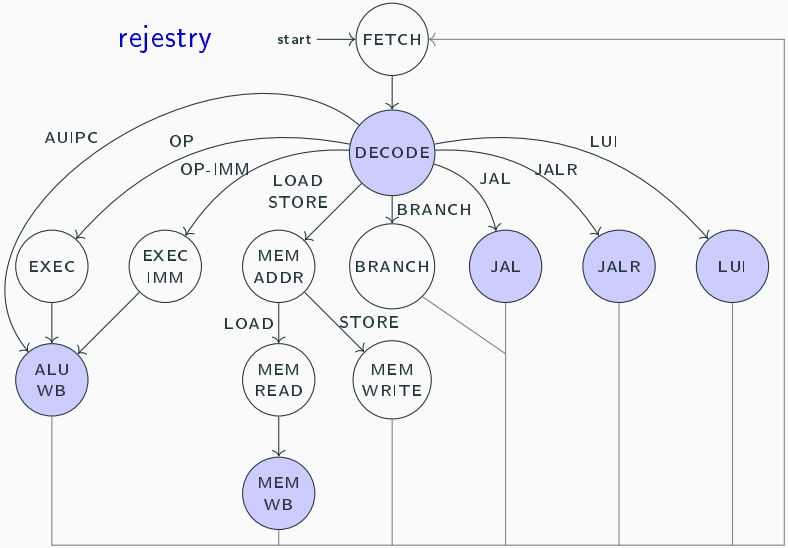
Automat skończony



Automat skończony

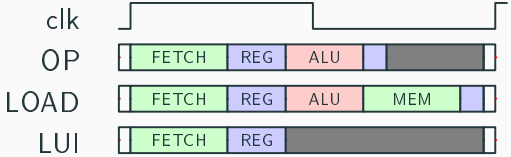


Automat skończony



Aspekty wydajnościowe

- Procesor jednocyklowy:



- Procesor wielocyklowy: