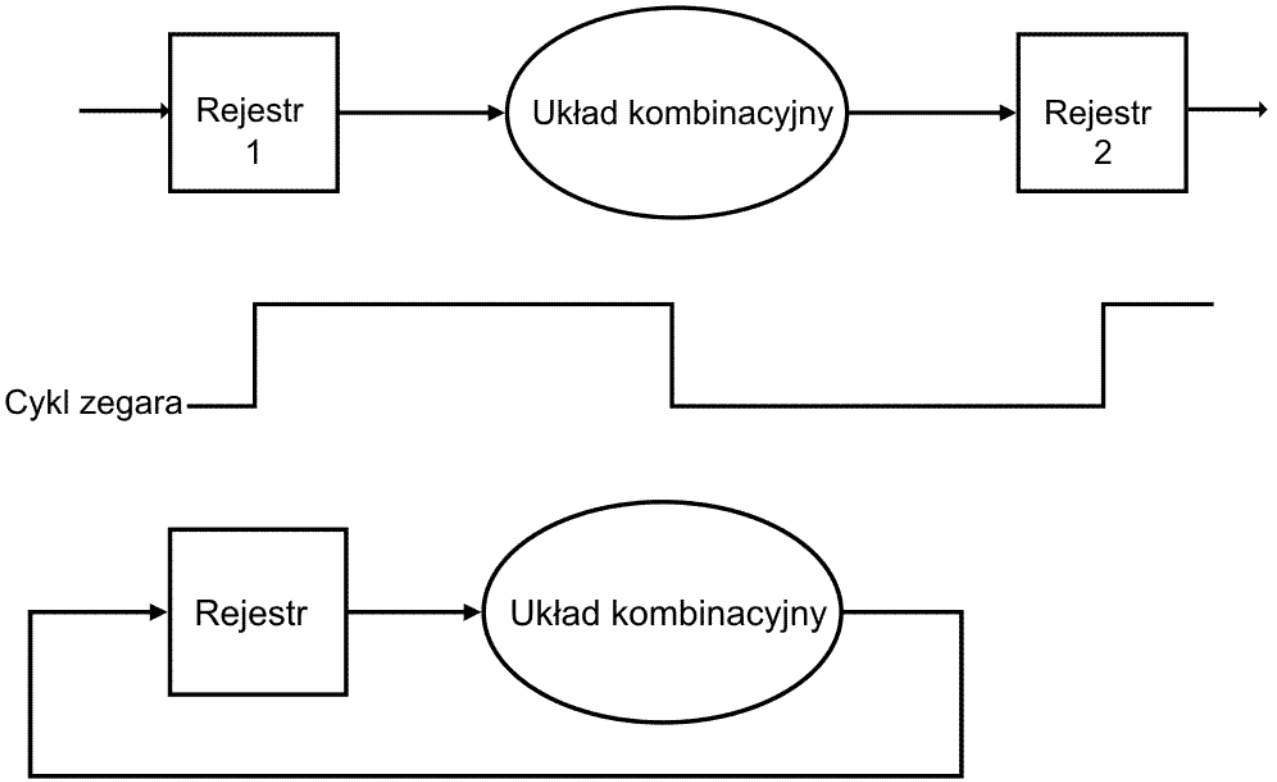
Architektura typu Single-Cycle

...czyli budujemy pierwszą maszynę parową

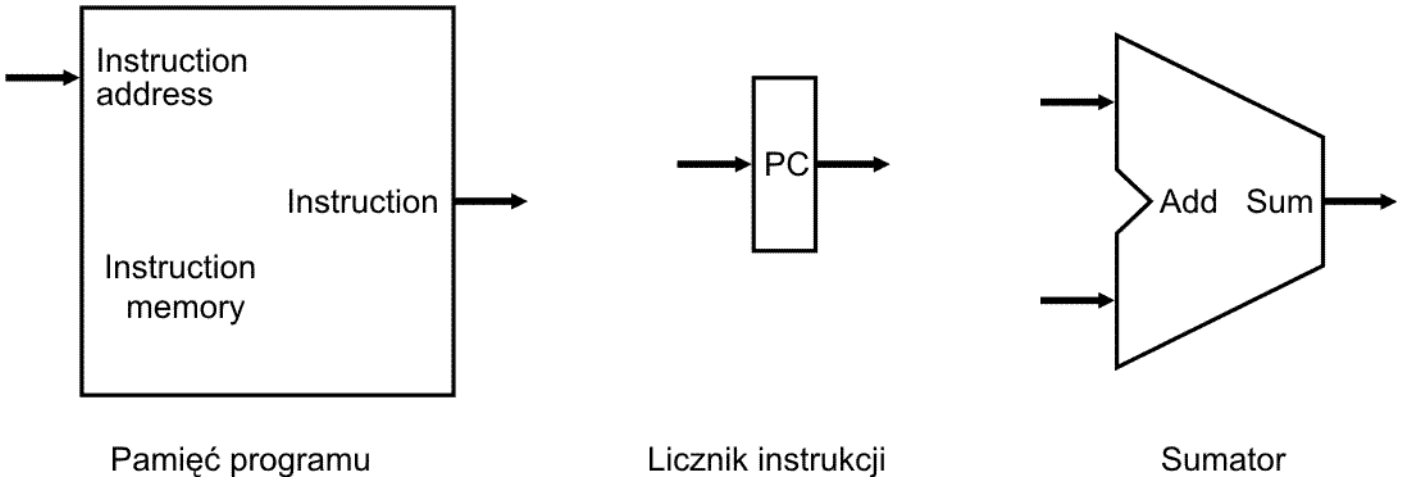
Przepływ danych

- W układach sekwencyjnych przepływ danych synchronizowany jest sygnałem zegara



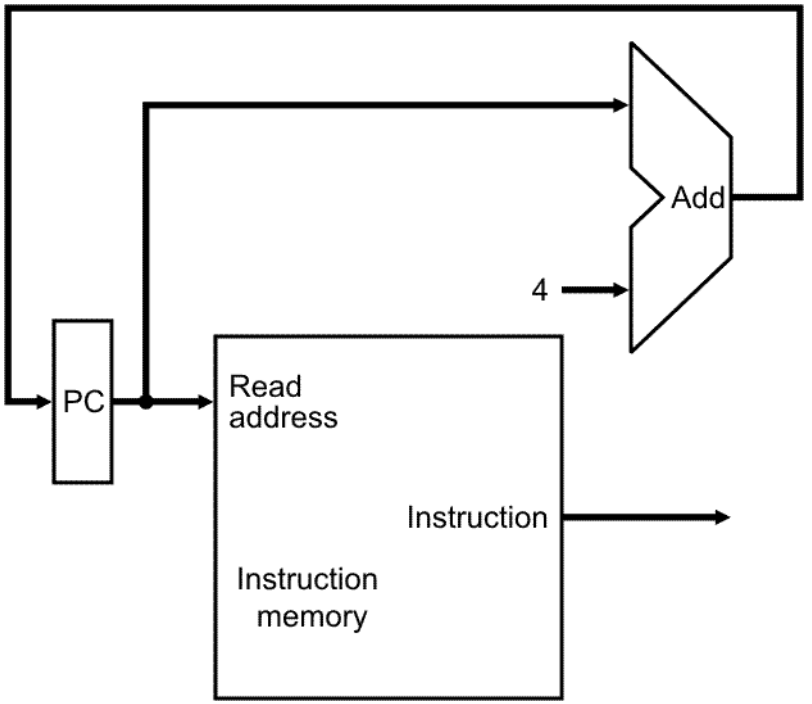
Elementy procesora - założenia

- Pamięć programu zawiera zapisany program
  - wszystkie instrukcje zajmują 4-bajty pamięci (32-bity)
  - magistrale *Instruction address* i *Instruction* są 32-bitowe
  - podanie adresu powoduje pojawianie się kodu instrukcji
- Rejestr PC zawiera adres (początku) instrukcji programu
- Sumator jest 32-bitowy



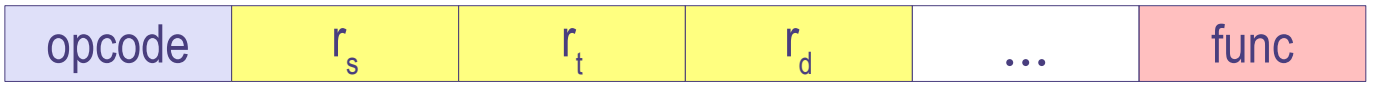
Układ pobierania instrukcji

- Pobieranie instrukcji – *Instruction Fetch*
  - sygnał zegara steruje zapisem do rejestru PC
  - w każdym cyklu zegara PC jest zwiększane o 4
  - na wyjściu pamięci pojawiają się kolejne instrukcje



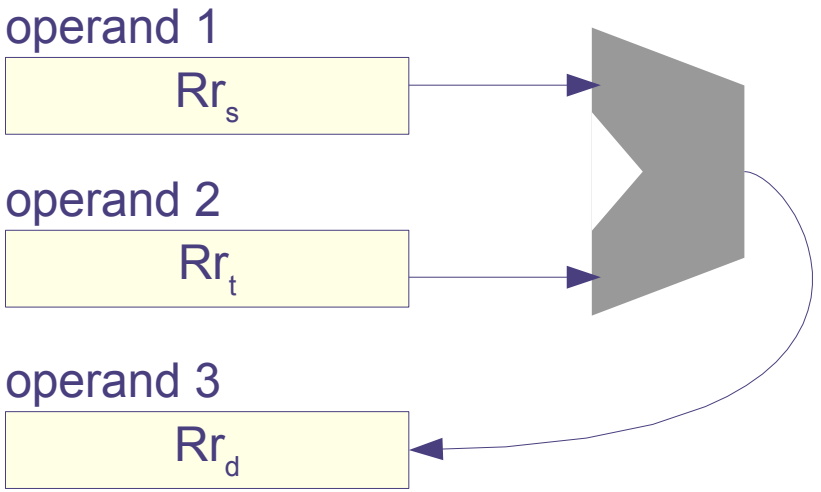
Instrukcje rejestrowe (*R-type*)

- Instrukcje rejestrowe wykonują operacje na wewnętrznych rejestrach procesora
  - dwa operandy źródłowe (Rr_s i Rr_t) znajdują się w rejestrach
  - wynik końcowy zapisywany jest do rejestru (Rr_d)
- Kod instrukcji R-type musi zawierać:
  - unikalny numer instrukcji (*opcode*)
  - numery trzech rejestrów wewnętrznych: r_s, r_t, t_d
  - rodzaj operacji arytmetycznej lub logicznej (func)



Adresowanie bezpośrednie rejestrowe

- (Register Direct Addressing)
- Operandy są w rejestrach wewnetrznych

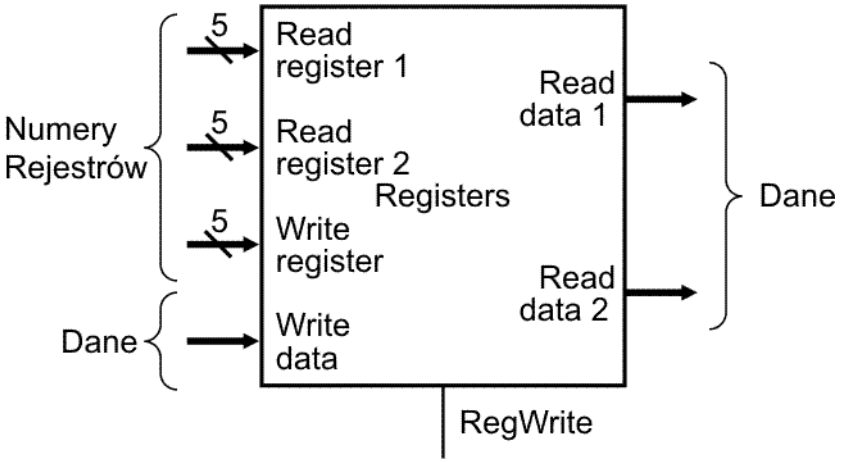


asembler:

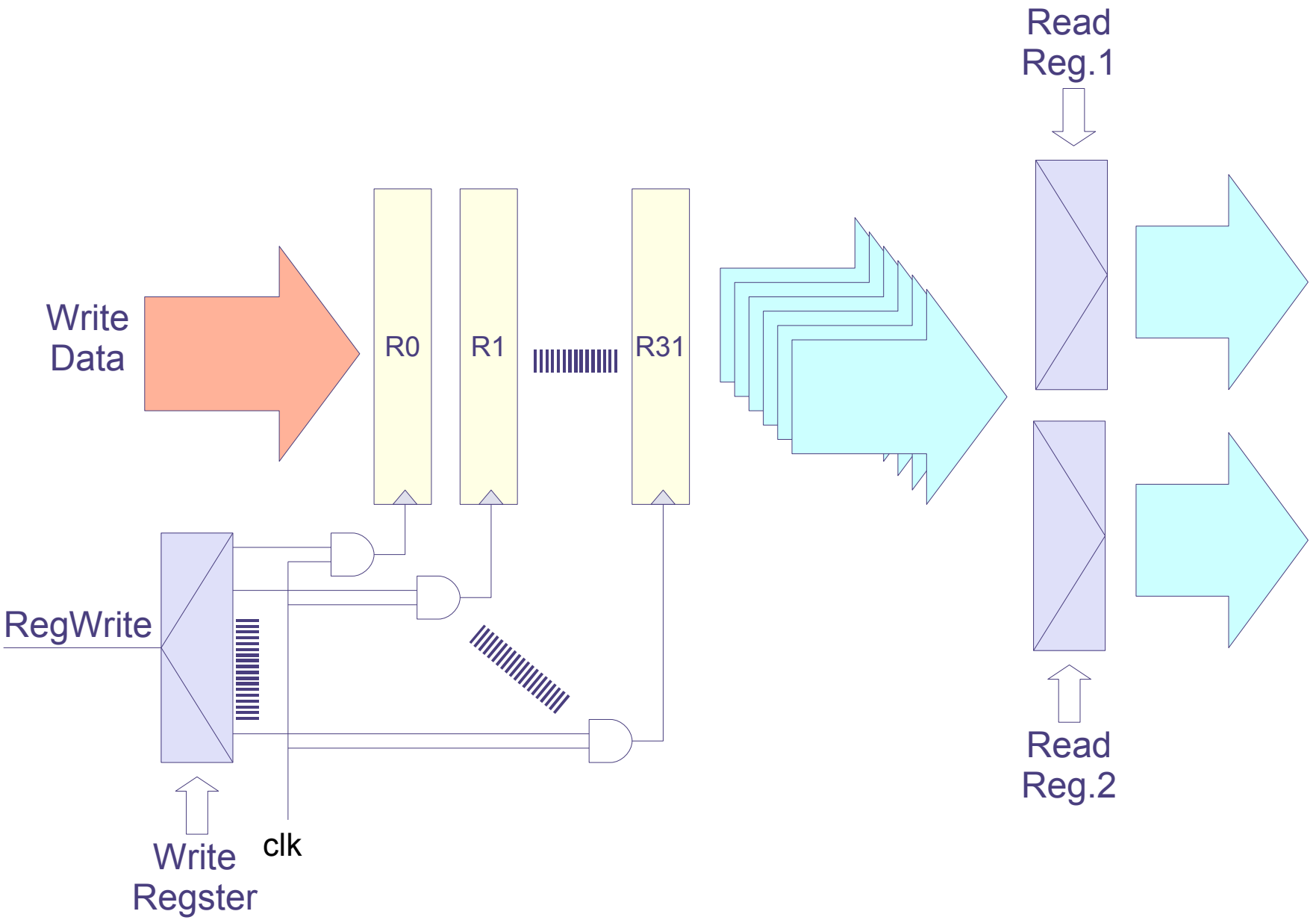
```
ADD  R1,R2,R7
SUB  R3,R6,R1
OR   R7,R3,R2
AND  R0,R2,R5
```

Plik rejestrów (*Register File*) - założenia

- Plik rejestrów zawiera 32 rejestry 32-bitowe
- Na wyjściu pliku dostępne są dwie wartości rejestrów o numerach podanych na wejścia *ReadRegister1&2*
- Numery rejestrów są 5-bitowe ($2^5 = 32$)
- Zapis do rejestru danych (*WriteData*) wymaga podania numeru rejestru (*WriteRegister*) i sygnału zezwolenia na zapis (*RegWrite*)
- Zapis jest synchronizowany sygnałem zegarowym

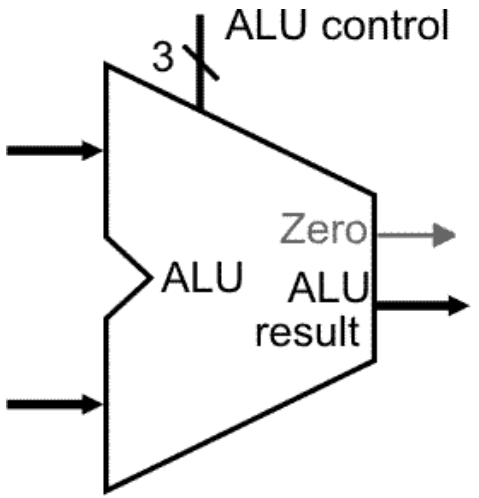


Plik rejestrów - koncepcja



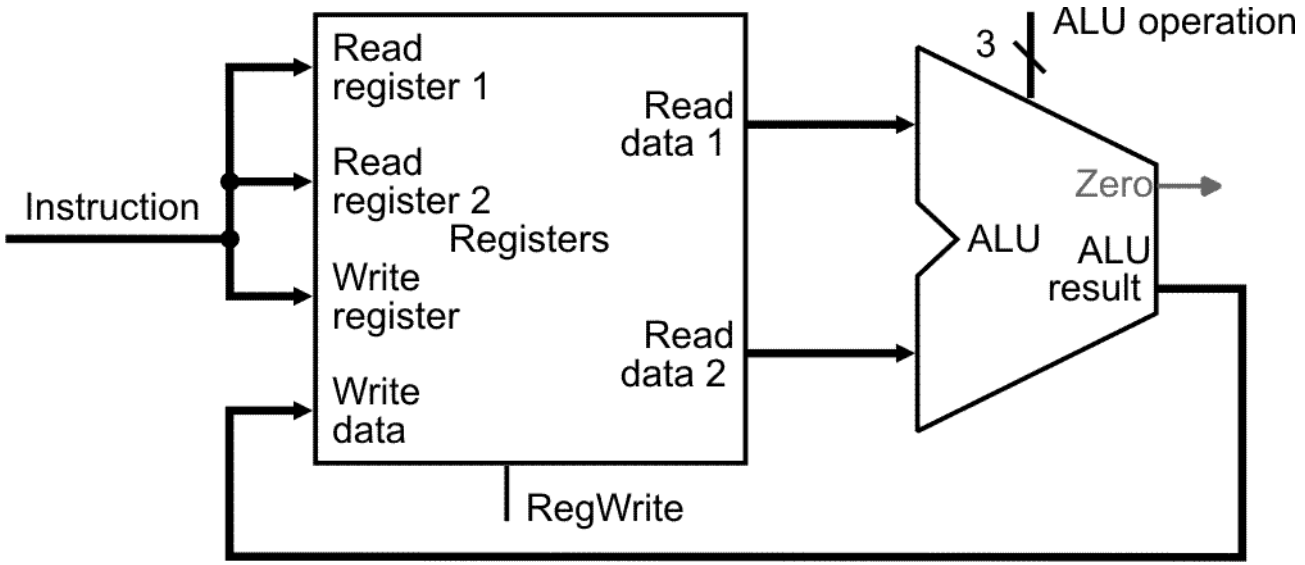
ALU - założenia

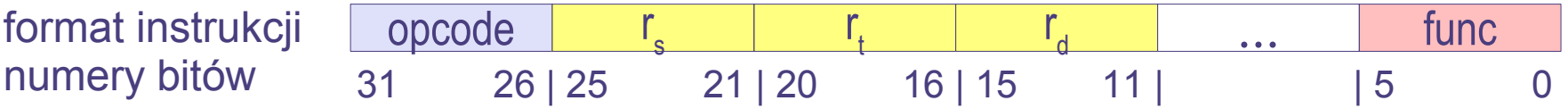
- Wyjścia i wyjście 32-bitowe (32-bitowe ALU)
- Dodawanie, odejmowanie, AND, OR
- Sterowanie 3-bitowe
- Tylko jeden sygnał kontrolny: zero



Wykonywanie instrukcji *R-type*

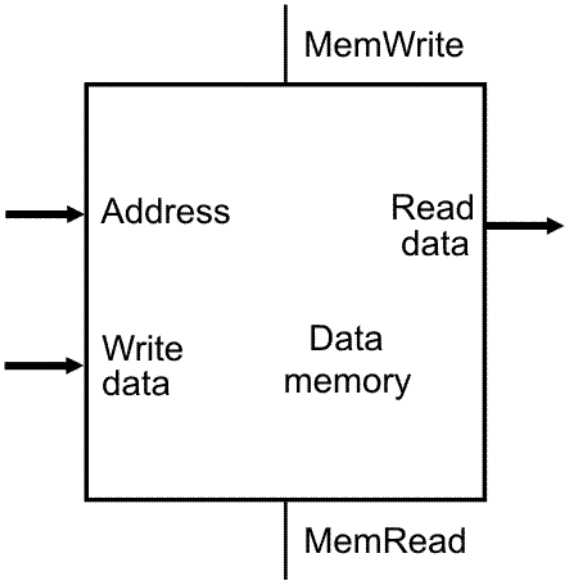
- Numery rejestrów z kodu instrukcji wybierają z pliku rejestrów rejestry źródłowe Rr_s i Rr_t
- Wynik operacji ALU jest zapisywany do rejestru Rr_d na zakończenie cyklu zegarowego (sygnał clk)





Pamięć danych - założenia

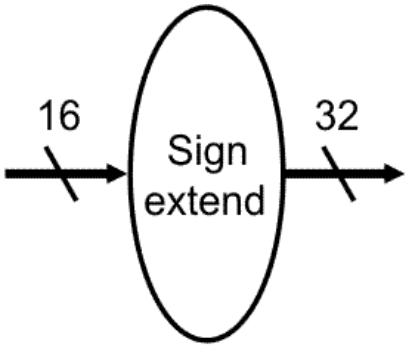
- Zawiera dane programu zorganizowane w 32-bitowe słowa (4B)
- Podanie sygnału zezwolenia *MemRead* powoduje pojawienie się na wyjściu *ReadData* zawartości pamięci spod adresu *Address*
- Zapis do danych pamięci (*WriteData*) odbywa się pod podany adres (*Address*) gdy aktywny jest sygnału zezwolenia na zapis (*MemWrite*)
- Zapis jest synchronizowany sygnałem zegarowym



Blok pamięci danych

Rozszerzenie znakowe - założenia

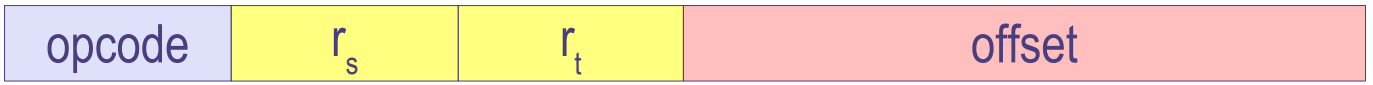
- Układ służy do zmiany reprezentacji liczby całkowitej 16-bitowej na 32-bitową z zachowaniem znaku (w sensie kodu U2)
- Układ jest kombinacyjny i nie wymaga taktowania



Blok rozszerzenia znakowego

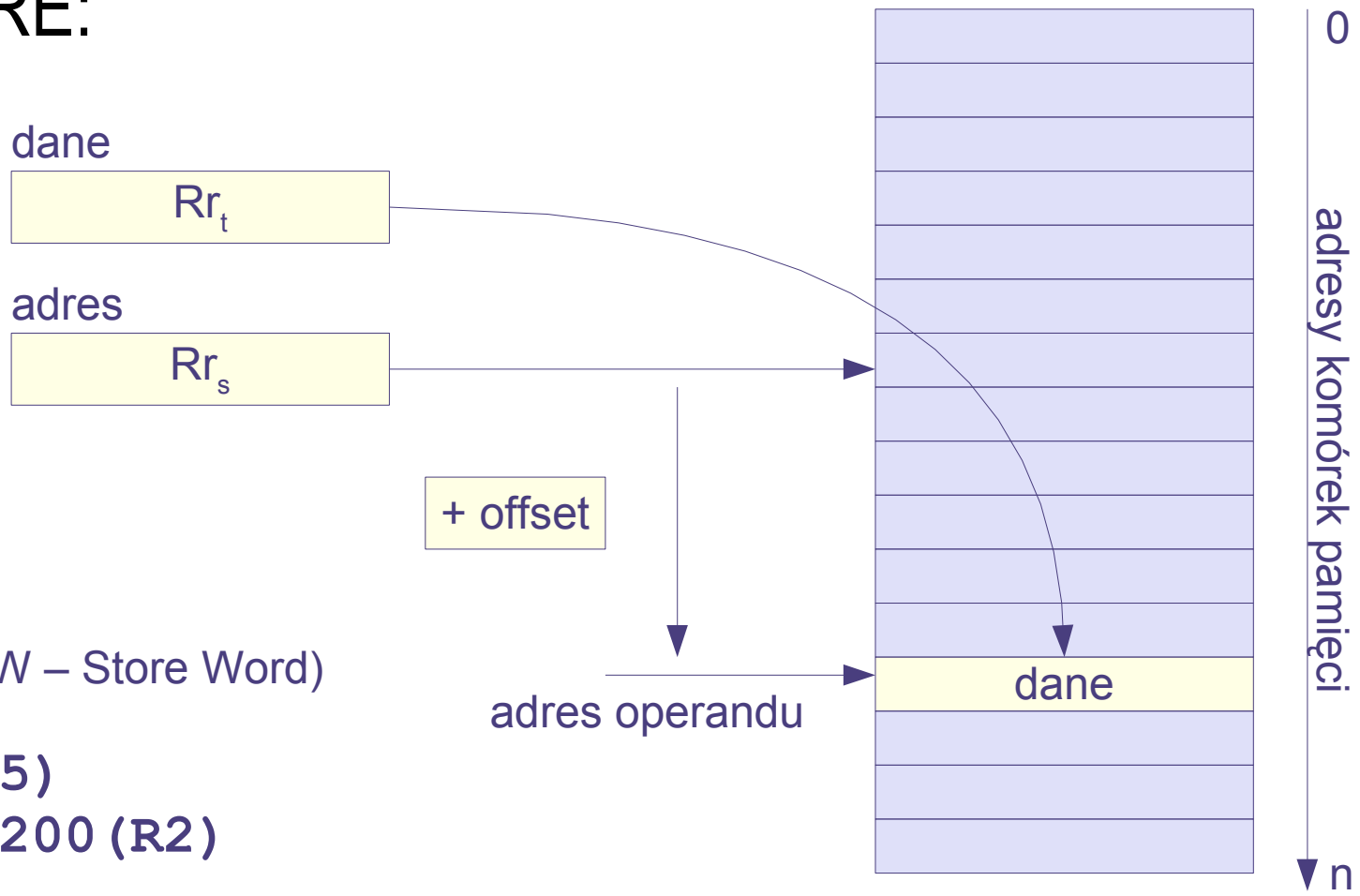
Instrukcje transferu (*Load/Store*)

- Instrukcje wykonują transfer danych:
  - z pamięci danych do rejestru wewnętrznego (*Load*)
  - z rejestru wewnętrznego do pamięci danych (*Store*)
  - dane do transferu: rejestr wewn. (Rr_s) i adres pamięci (Rr_t)
  - stała w kodzie instrukcji ułatwia dostęp do pamięci
- Kod instrukcji Load/Store musi zawierać:
  - unikalny numer instrukcji (*opcode*)
  - numery dwóch rejestrów wewnętrznych: r_s, r_t
  - stałą (*offset*), która jest dodawana do adresu bazowego



Adresowanie pośrednie rejestrowe

- ... z przesunięciem (Register Indirect with Offset Addressing)
- Offset może być liczbą dodatnią lub ujemną (U2)
- STORE:

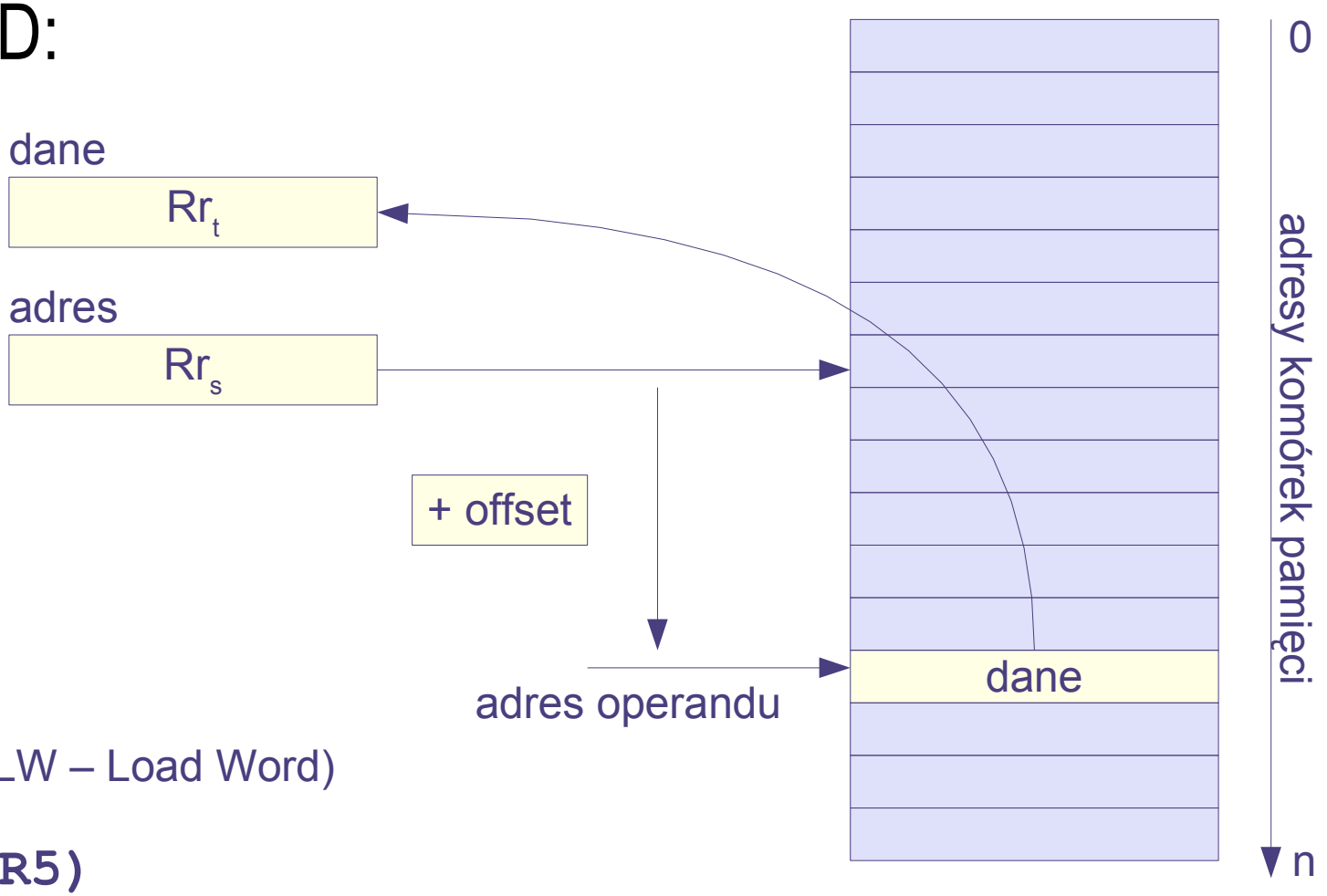


asembler: (SW – Store Word)

```
SW R7,(R5)
SW R1,0x200(R2)
```

Adresowanie pośrednie rejestrowe

- ... z przesunięciem (Register Indirect with Offset Addressing)
- LOAD:

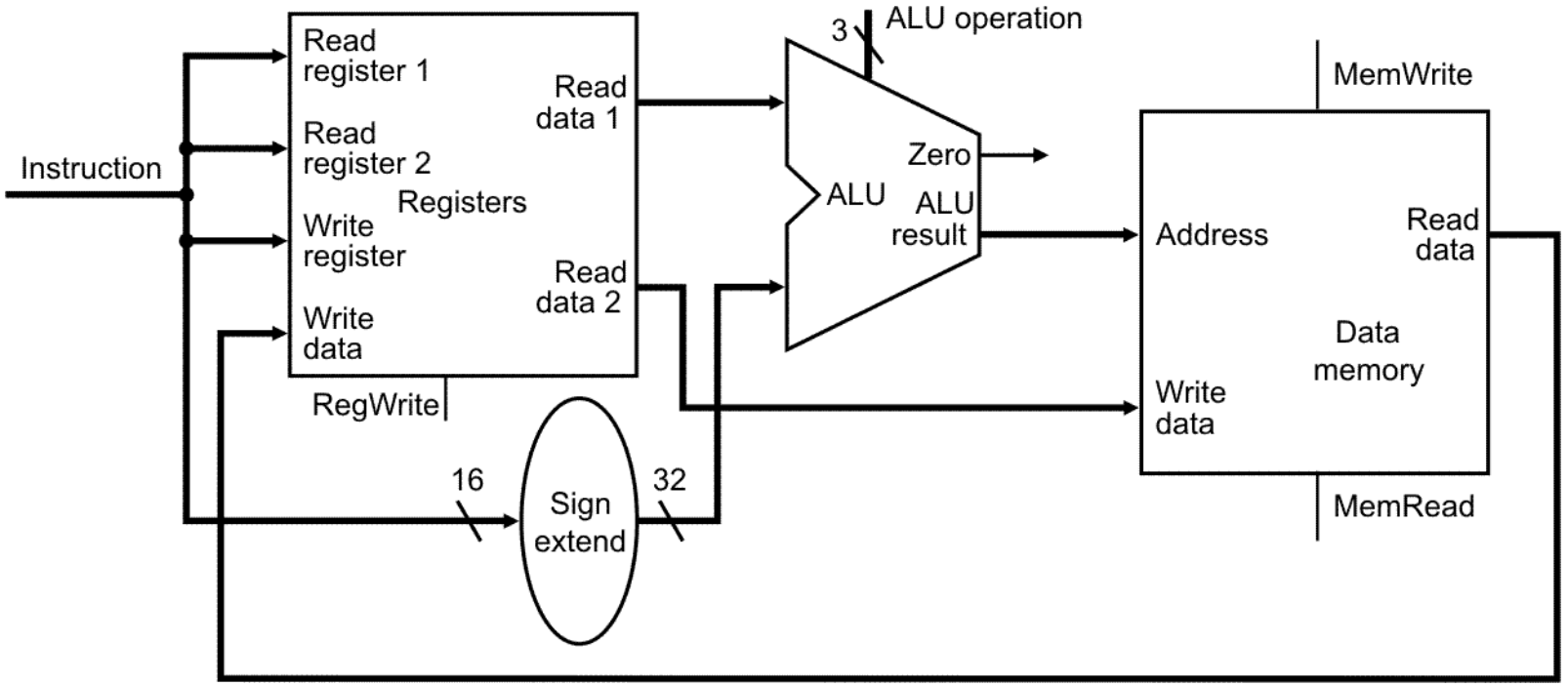


asembler: (LW – Load Word)

```
LW R7,(R5)
LW R1,0x200(R2)
```

Wykonywanie instrukcji *Load/Store*

- Rejestr Rr_s + offset adresuje pamięć danych (*Load/Store*)
- *Store*: Rejestr Rr_t na wejściu pamięci i będzie zapisany (sygnał *MemWrite* aktywny)
- *Load*: Dane z wyjścia pamięci bedą zapisane do rejestru Rrt (sygnały *MemRead i RegWrite* aktywne)

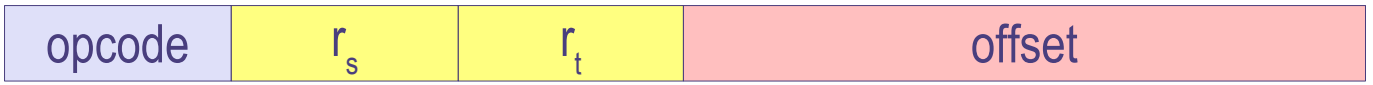


Instrukcje skoków

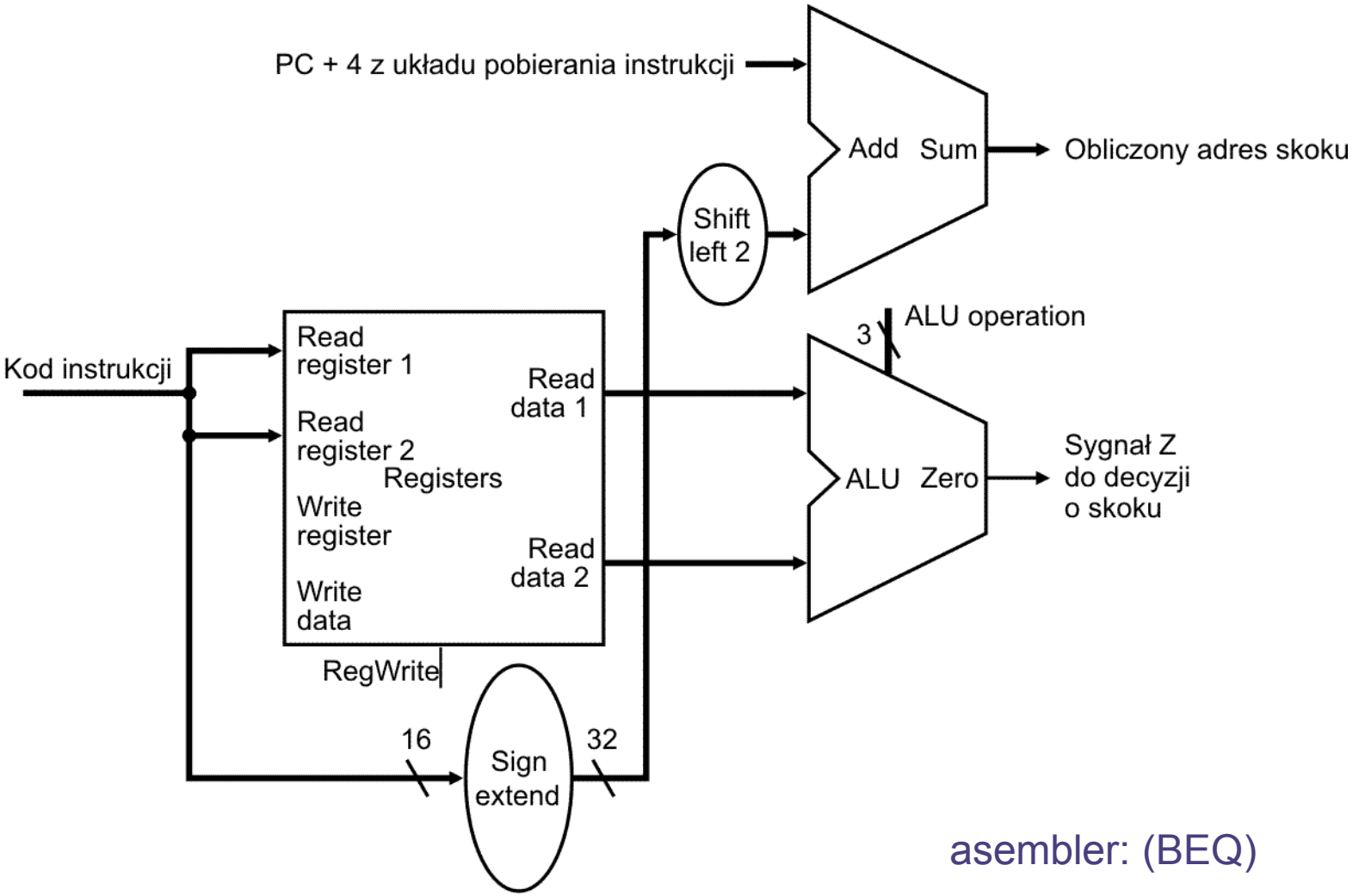
- Skok: odstępstwo od wykonania ciągu instrukcji zapisanych kolejno w pamięci
- Każda instrukcja skoku polega na modyfikacji rejestru PC
- Skoki absolutne (bezwzględne) i względne
  - absolutne – nowa zawartość ładowana do PC
  - względne – przesunięcie (+/-) dodawane do PC
- Skoki bezwarunkowe i warunkowe
  - bezwarunkowe – skok wykona się zawsze
  - warunkowe – skok wykona się w zależności od spełnienia warunku (obliczanego w ALU i sygnalizowanego bitami C,V,Z,N,)

Skok względny warunkowy (Z=1)

- **`BEQ Rx,Ry,offset`** (Branch if EQual)
  – skocz do adresu PC+offset*4 jeśli Rx=Ry (Z=1)
- Instrukcje zajmują 4 bajty, więc zakres skoku może być poszerzony: offset*4
- Kod instrukcji BEQ musi zawierać:
  - unikalny numer instrukcji (*opcode*)
  - numery dwóch rejestrów wewnętrznych: r_s, r_t
  - stałą (*offset*), która jest dodawana do adresu bazowego



Skok względny warunkowy (Z=1)

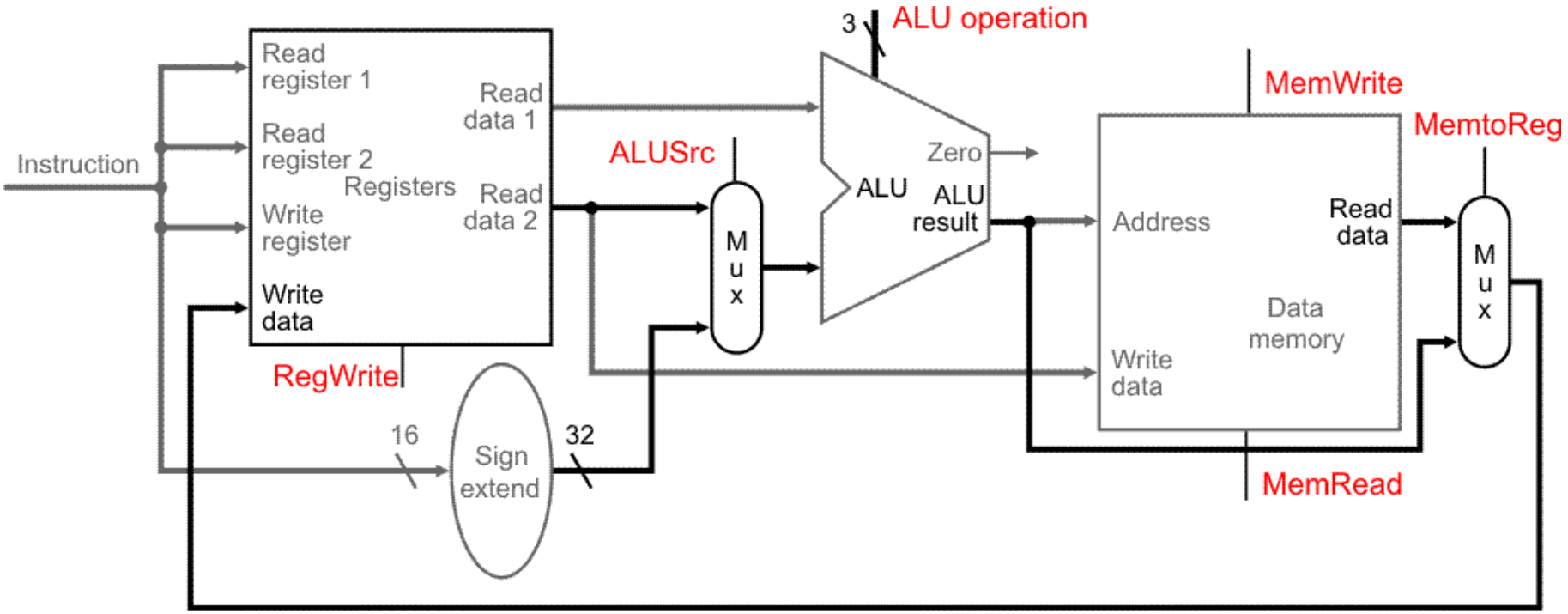


asembler: (BEQ)

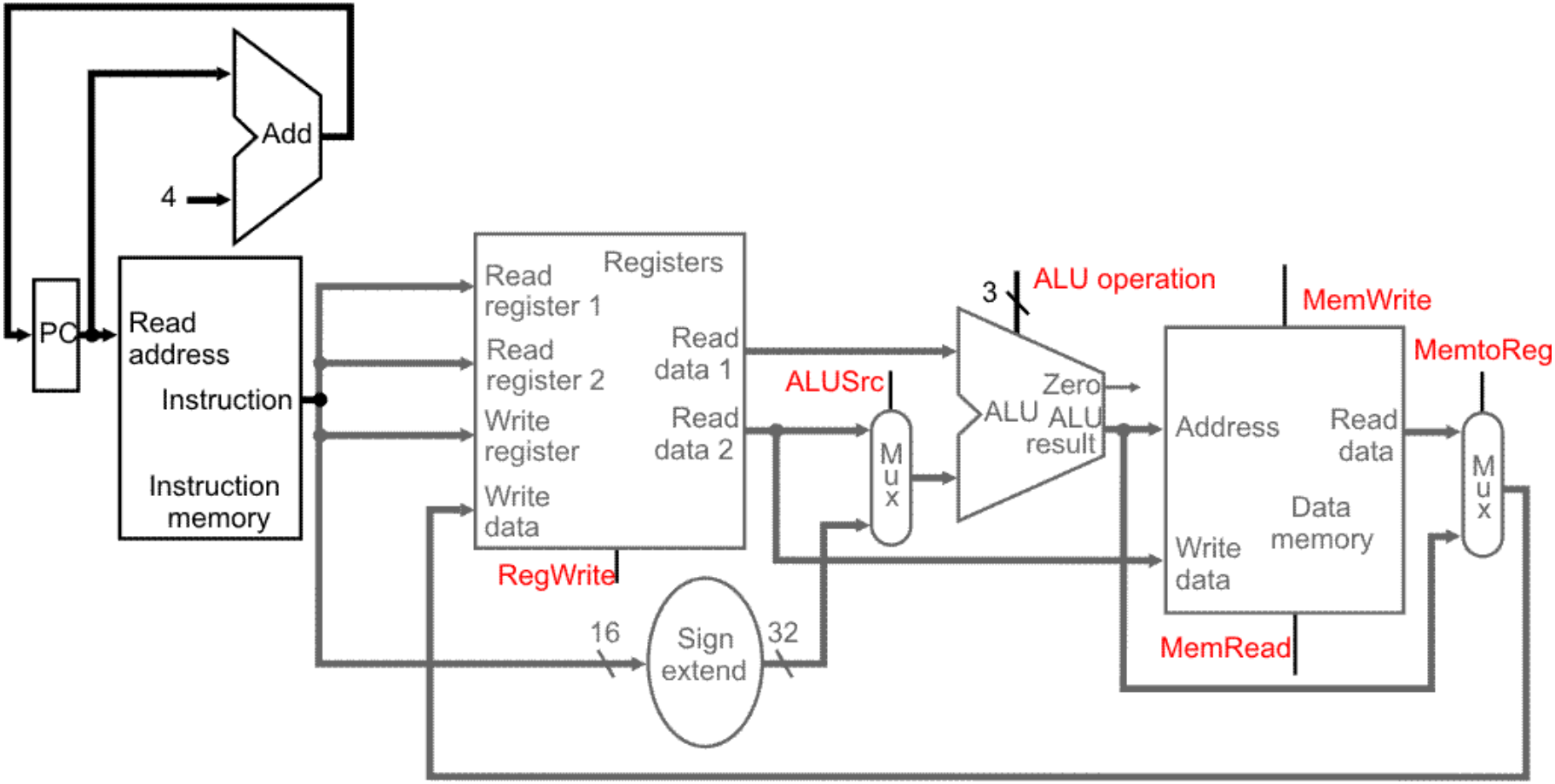
```
BEQ R1,R2,0x40
BEQ R0,R7,0xFF
```

Instrukcje *R-type* i *Load/Store* razem

- Multipleksery:
  - ALUSrc: wybór drugiego operandu dla ALU
  - MemtoReg: wybór danych do zapisu w rejestrze

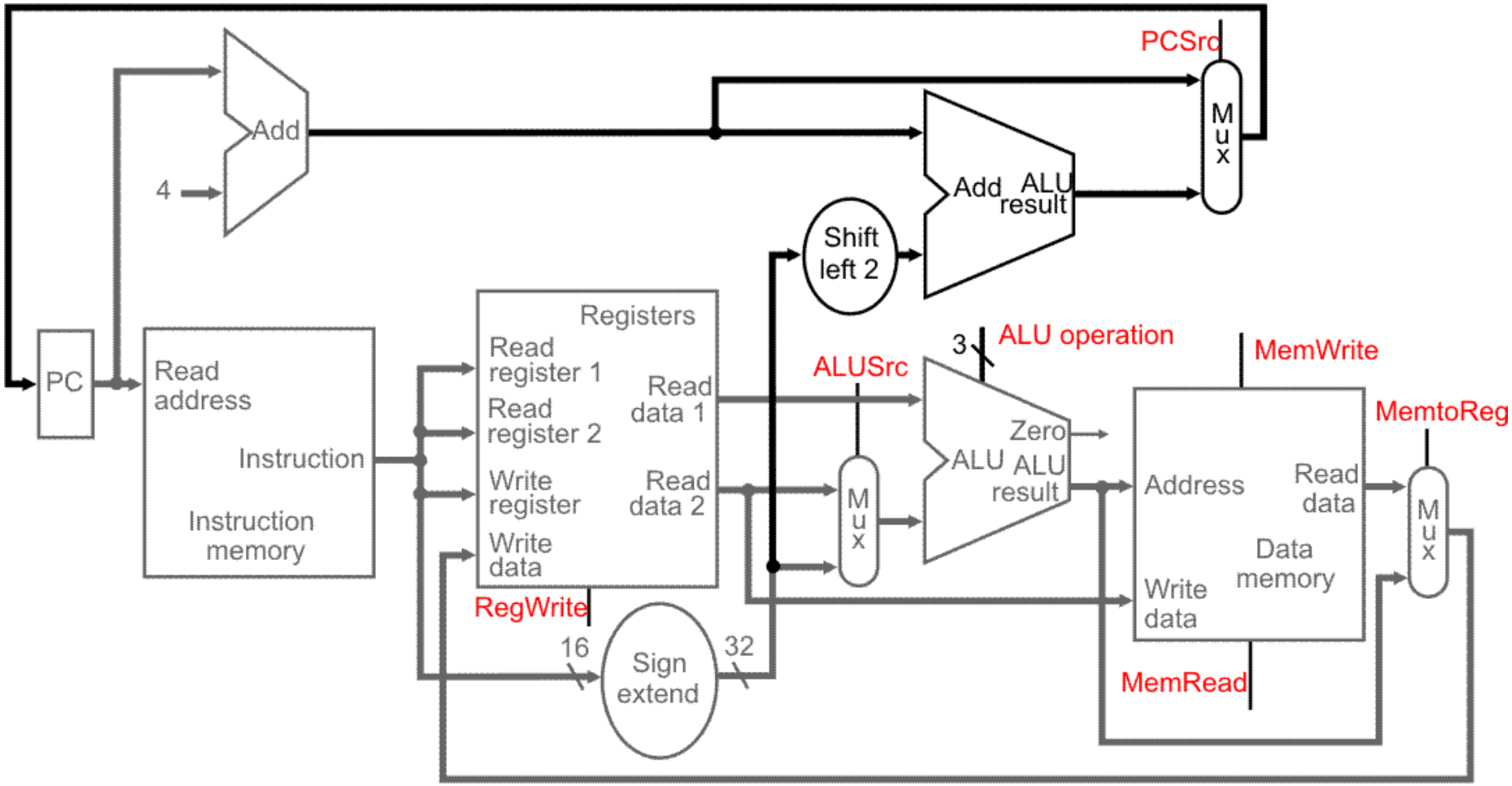


Fetch + R-type, Load/Store



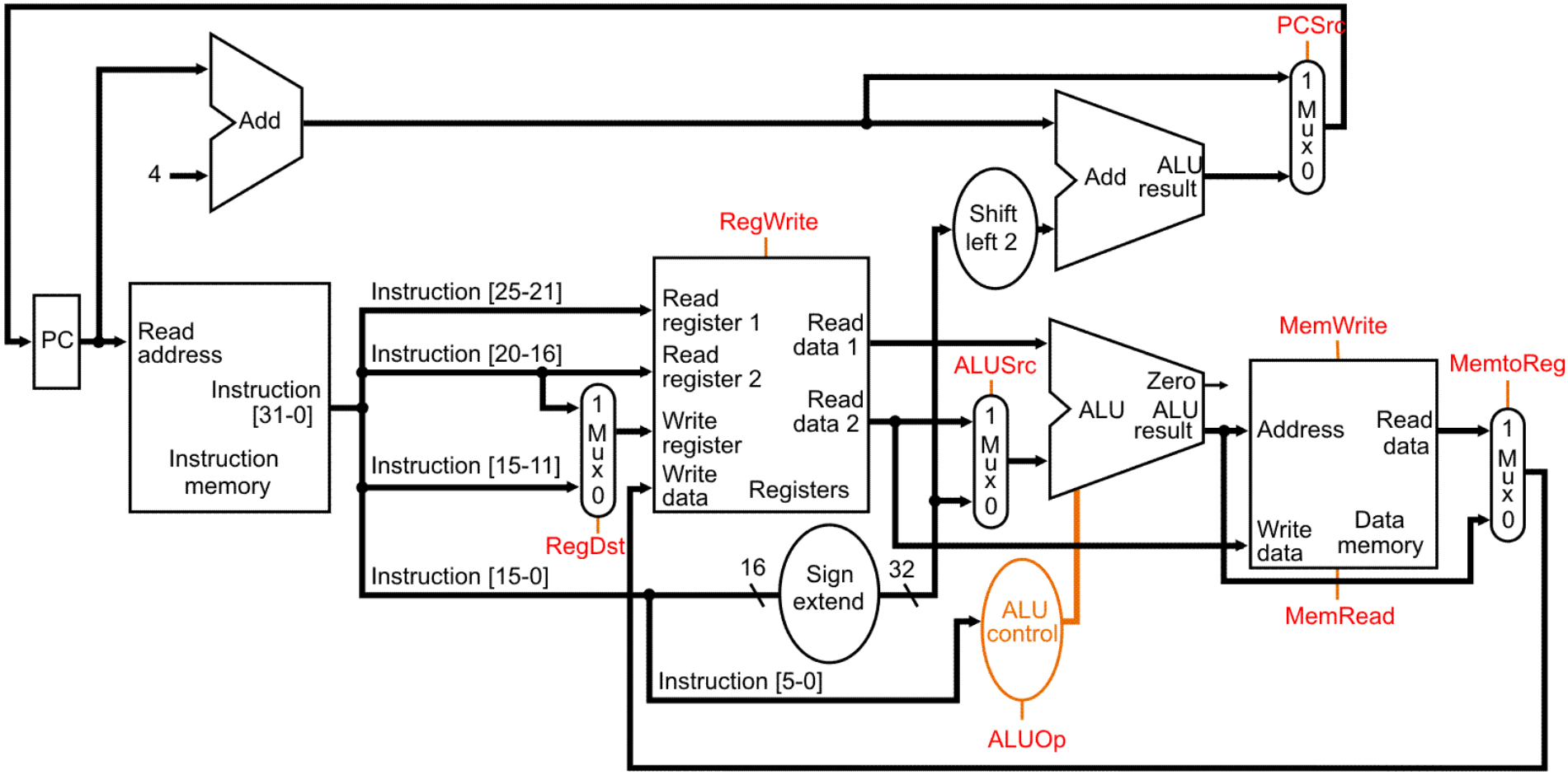
Fetch + R-type, Load/Store, Branch

- Multiplekser: PCSrc
  - wybór nowej wartości PC: PC+4 lub PC+offset*4



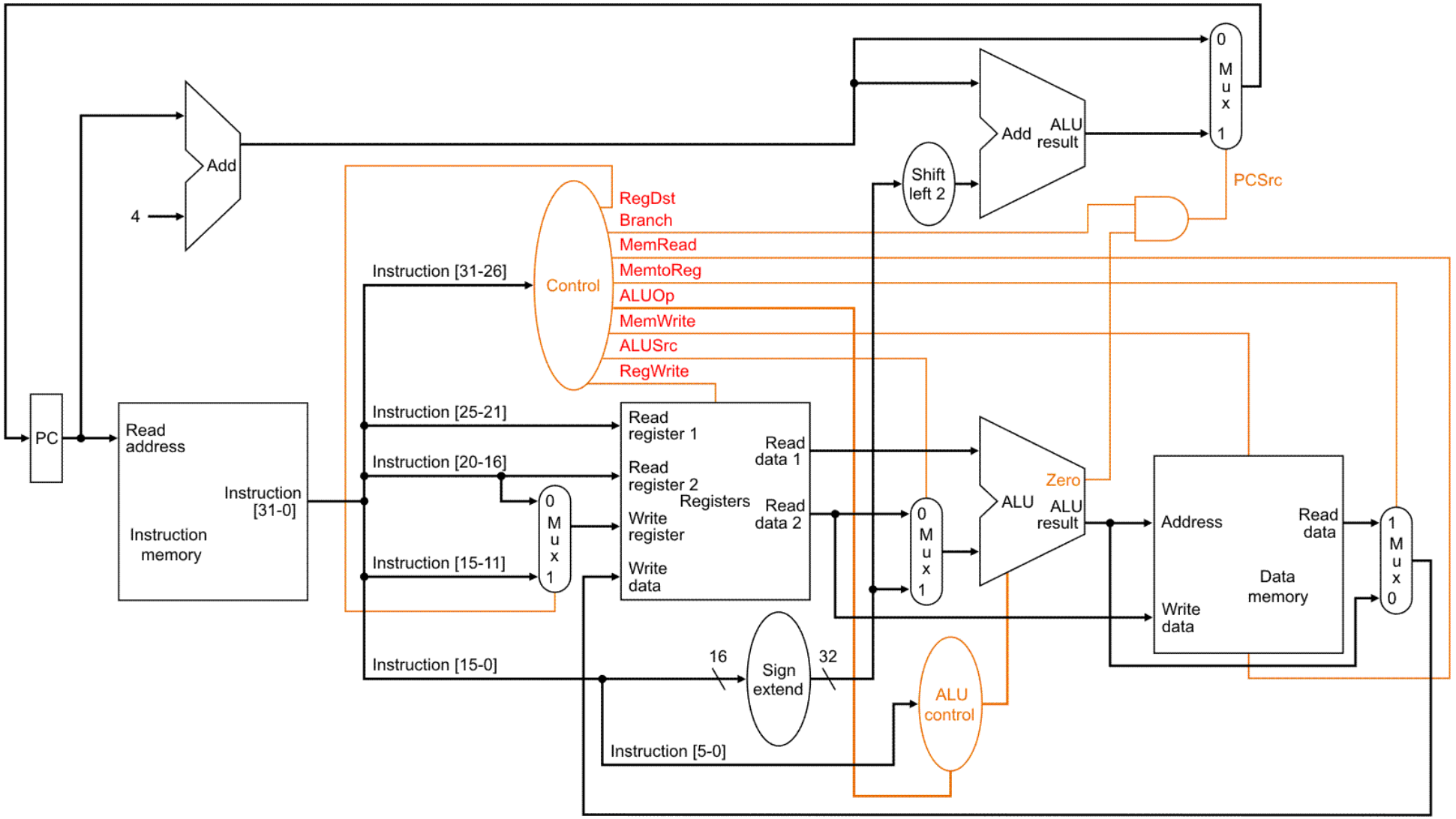
Korekcja zapisu do rejestru

- Multiplekser: RegDst
  - wybór prawidłowego numeru rejestru do modyfikacji (R-type $\rightarrow r_d$, Load $\rightarrow r_t$)



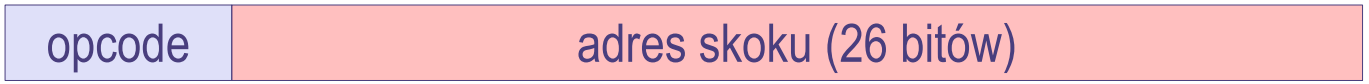
Jednostka sterująca

- *Control*: układ kombinacyjny generujący sygnały sterujące

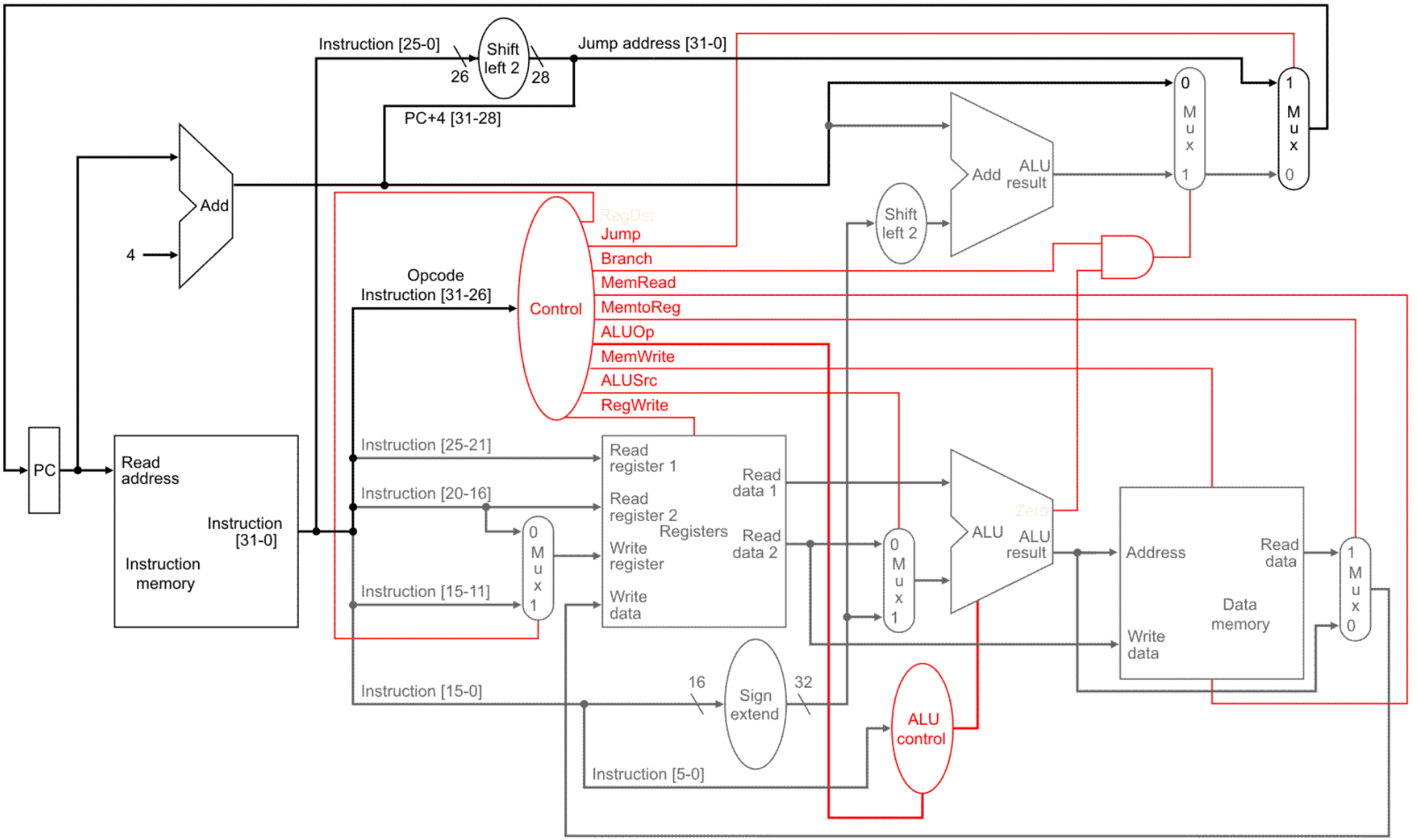


Skoki bezwarunkowe absolutne JMP

- Do PC ładowana jest nowa wartość
- Nie są sprawdzane żadne warunki
- Adres skoku, zawarty w instrukcji jest mnożony przez 4
- Brakujące (4) bity mogą być np. uzupełniane z bieżącej zawartości PC (skok w ramach „segmentu” pamięci).
- Multiplekser sterowany sygnałem *Jump*, do wyboru adresu następnej instrukcji

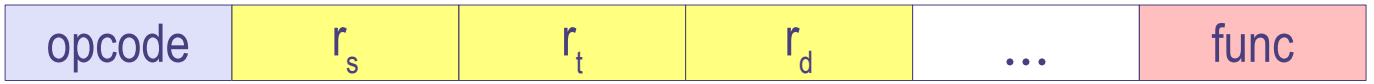


Kompletna architektura *Single-Cycle*

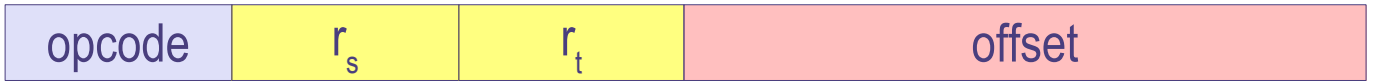


Zestaw instrukcji

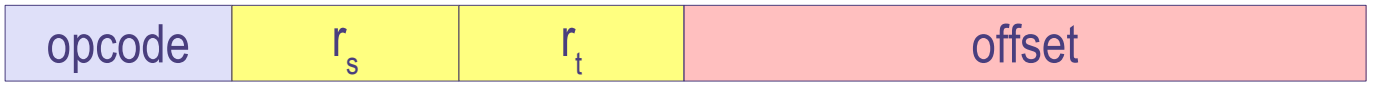
- Rejestrowe (*R-type*)



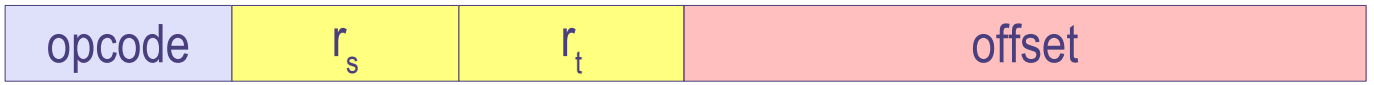
- Load



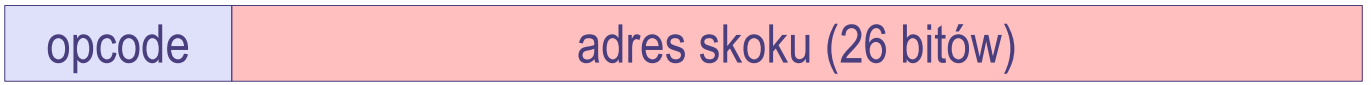
- Store



- BEQ

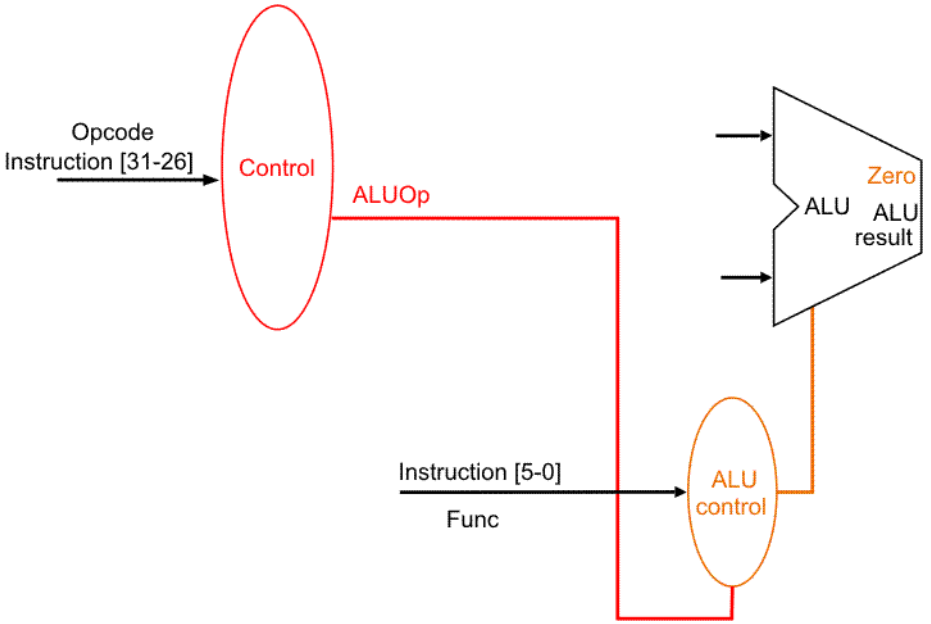


- JMP



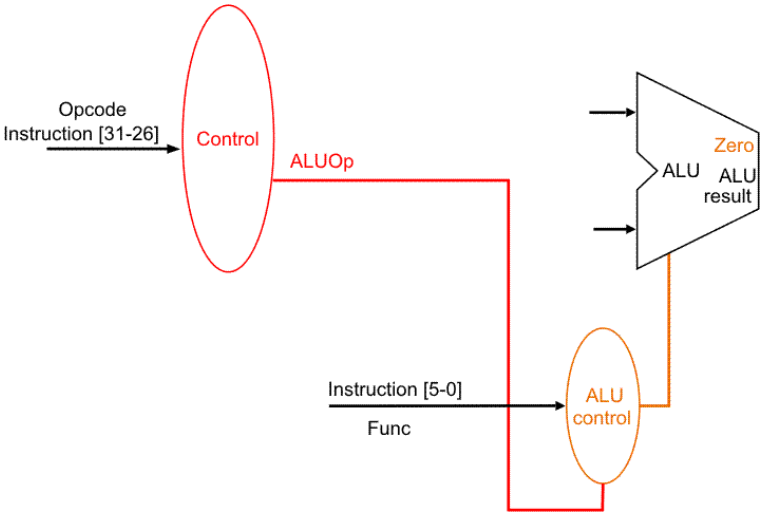
Sterowanie ALU

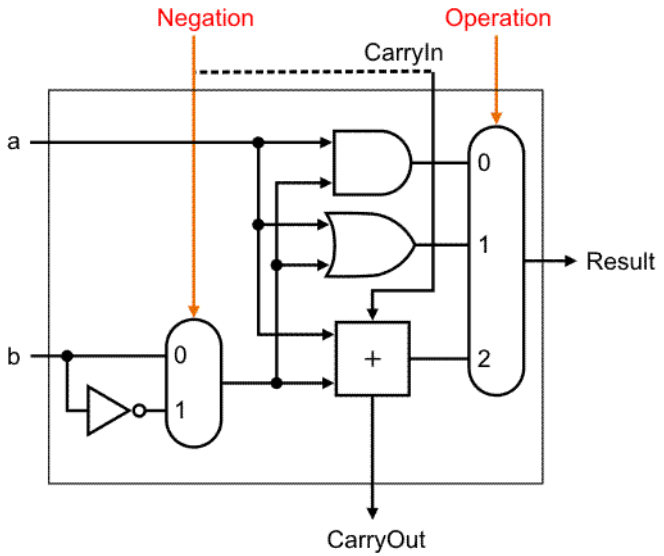
- Instrukcje *R-type* mają identyczny *opcode*, ale różnią się polem *Func*
- Główna jednostka sterująca (*Control*) rozpoznaje jedynie *opcode* instrukcji i ustawia sygnał *ALUop*
- Jednostka *ALU Control* uwzględnia pole Func tylko dla instrukcji *R-type* (zgłaszanych sygnałem *ALUop*)

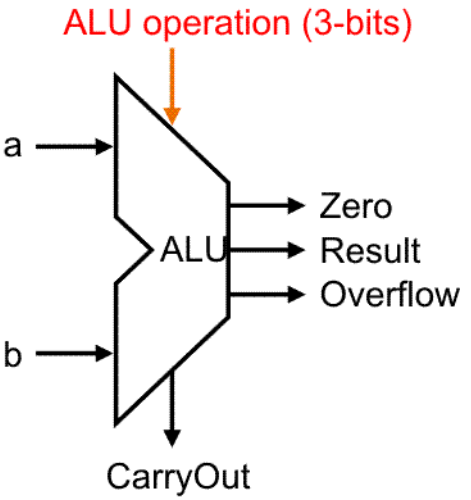


Sterowanie ALU

Func	ALU operation (Neg.+Oper)
AND	000
OR	001
ADD	010
SUB	110







Sterowanie ALU - podsumowanie

Opcode	ALUOp	operation	Func field	ALU action	ALU input
LW	00	load word	xxxxxx	add	010
SW	00	store word	xxxxxx	add	010
BEQ	01	branch equal	xxxxxx	subtract	110
R-type	10	ADD	100000	add	010
R-type	10	SUB	100010	subtract	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
Jump	xx	x	xxxxxx	x	xxx