Wojskowa Akademia Techniczna im. Jarosława Dąbrowskiego

Laboratorium z przedmiotu:
Architektura i organizacja komputerów

Sprawozdanie z ćwiczenia laboratoryjnego nr 6:

Realizacja operacji arytmetycznych w komputerze DLX

Spis treści

Treść zadania

Lab6 (11-12) Y4 prawdziwe

Pierwszym parametrem w tym zadaniu będzie **k = reszta z dzielenia całkowitego ostatniej cyfry numeru albumu autorki/ autora sprawozdania przez 6**. Oczywiście wartość tej reszty, czyli wynik operacji (ostatnia_cyfra mod 6) przyjmować będzie wartości ze zbioru [0,1,2,3,4,5].

Drugim parametrem będzie **nr = numer_w_dzienniku autorki/ autora sprawozdania (numer na liście grupy w USOS).**

Napisać program w asemblerze komputera WinDLX, który

1. Zadeklaruje statycznie (jak w przykładzie z mojej strony Lab5 dla zmiennej „tablica_B") **rozmiar = (10 + k)** elementowy wektor liczb całkowitych o nazwie **wektor**, pierwszy element o wartości równej **100+nr**, każdy następny o (**k+10)** większy.
   Na przykład dla osoby o numerze albumu kończącym się na 7 i numerze w dzienniku równym 5 będą to odpowiednio (k=1, nr = 5, liczba elementów wektora = 10 + k = 11): 105, 116, 127, …, 204, 215.
2. W pętli **policzy sumę elementów wektora dla jego wartości początkowych** do rejestru Rnr (dla osoby o numerze **5** do R**5**, dla osoby o numerze **10** do R**10** itd.) a następnie zapisze zawartość Rnr do zmiennej **suma1**. Dla powyższych danych suma1 = 1760.
3. Zadeklaruje stałą **stala**, równą iloczynowi (k+1) i nr, np. dla powyższego przykładu stala = (1+1) * 5 = 2 * 5 = 10.
4. W pętli **zwiększy zawartość każdego elementu wektora o stałą stala i zapisze w miejscu dotychczasowego elementu**. Na przykład dla powyższych danych nowe zawartości wektora byłyby równe odpowiednio 115, 126, 137, …, 214, 225.
5. W pętli policzy **sumę elementów wektora dla jego wartości po modyfikacji** do rejestru Rnr, a następnie zapisze zawartość Rnr do zmiennej **suma2**. Dla powyższych danych suma2 = 1870.
6. W rejestrze Rnr obliczy (**różnicę = suma2 - suma1**) i wynik zapisze do zmiennej **roznica**. Dla powyższych danych roznica = 110.
7. W rejestrze Rnr obliczy **iloczyn = rozmiar x stala** i wynik zapisze do zmiennej **iloczyn**. Dla powyższych danych iloczyn = 110.

W sprawozdaniu:

A. Zamieścić treść zadania z mojej strony. Jawnie podać wartości **k** i **nr**, wyniki obliczeń wartości **wektora (przed i po modyfikacji)** i wyników obliczeń: **suma1**, **suma2**, **roznica**, **iloczyn** – uzyskane dla obliczeń pisemnych.
B. Zamieścić listing napisanego przez siebie programu w postaci tekstowej, możliwej do „skopiowania" w przeglądarce typu Adobe Reader – nie zamieszczać tekstu programu w postaci obrazka. Muszę mieć możliwość skopiowania Waszego programu do mojej maszyny wirtualnej i sprawdzenia poprawności działania.
C. Zamieścić zrzutu ekranu z WinDLX z uzyskanymi wynikami, w tym
   a. na jednym z obrazków ze stanem początkowym wektora i wyzerowanymi zmiennymi wynikowymi (okienko Memory/ Display, odpowiednio skonfigurowane);
   b. na drugim z obrazków ze stanem wektora i wynikami obliczeń **suma1**, **suma2**, **roznica**, **iloczyn** – po zakończeniu wykonywania programu (okienko Memory/ Display, odpowiednio skonfigurowane);
   c. Na trzecim z obrazków stan zmaksymalizowanego okienka Menu/ Window/ **Statistics**.
D. Zamieścić algorytm swojego programu w postaci graficznej i krótko ten algorytm opisać, ze szczególnym uwzględnieniem warunków wyjścia z każdej pętli.
E. W zależności od swojej wartości **k** wybrać **jedną** z instrukcji swojego programu, odpowiednio

a. k=0,1 rozkaz typu load albo branch;
b. k=2,3 rozkaz typu arithmetic immediate;
c. k=4,5 rozkaz typu store
i opisać zmiany w rejestrach **R** i tymczasowych (**A, B, Imm** itp.) w trakcie kompletnego wykonania tego rozkazu przez poszczególne etapy komputera WinDLX, podobnie do mojego opisu na http://www.ita.wat.edu.pl/~a.miktus/AOK/Lab6/Etapy_potoku_DLX.html .
d. Opis ma być uzupełniony zrzutami ekranu z WinDLX, pokazującymi opisywane zmiany dla tej jednej, wybranej instrukcji.

Stopień trudności zadania:
1. Na ocenę **dst** punkty **1 – 2** zadania i **A – D** sprawozdania.
2. Na ocenę **db** punkty **1 – 5** zadania i **A – D** sprawozdania.
3. Na ocenę **bdb** punkty **1 – 7** zadania i **A – E** sprawozdania (czyli wszystko).

W przypadku stwierdzenia niesamodzielnej pracy = nieuczciwości studentów osoby oszukujące (dawca i biorcy) za zadanie otrzymują ocenę **zero do średniej**. To samo w przypadku niewykonania zadania i nieprzysłania sprawozdania w terminie.

A. Moje parametry i wyniki obliczeń

Parametry:

Parametr	Wartość
cyfra	6
k	cyfra mod 6 = 0
nr	10
rozmiar	10 + k = 10

Wyniki działań na wektorze:

Wektor	Wartości
Przed modyfikacją	110 120 130 140 150 160 170 180 190 200
Po modyfikacji	120 130 140 150 160 170 180 190 200 210

Obliczone wartości zmiennych:

Wynik obliczeń	Wartość
suma1	1550
suma2	1650
roznica	100
iloczyn	100

B. Kod programu

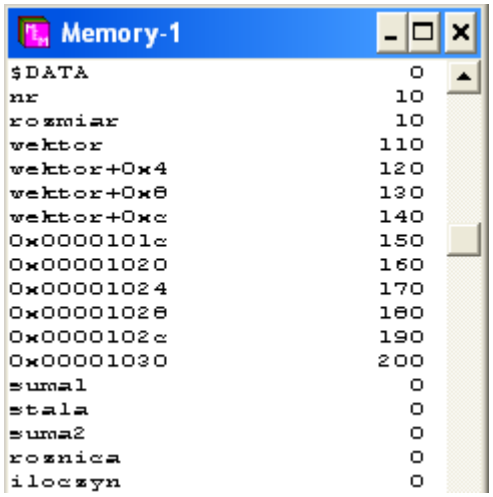
```
.data
k: .word 0
nr: .word 10
rozmiar: .word 10
wektor: .word 110, 120, 130, 140, 150, 160, 170, 180, 190, 200
;z2
```

```
suma1: .word 0
;z3
stala: .word 0
;z5
suma2: .word 0
;z6
roznica: .word 0
;z7
iloczyn: .word 0

.text
; z2: licze sume liczb w wektorze
    lw r1, rozmiar      ;pobierz rozmiar (licznik)
    addi r2, r0, wektor ;r2 wskazuje na pierwszy element wektora
    loop1:
        lw r3, 0(r2)     ;wczytaj liczbe
        add r10, r10, r3 ;dodaj liczbe
        addi r2, r2, #4  ;wskaz nastepna liczbe - sizeof(word)=4
        subi r1, r1, #1  ;zmniejsz licznik
        bnez r1, loop1   ;zakoncz jesli licznik = 0
    sw suma1, r10       ;wpisz do "suma1"

; z3: licze stala ((k+1)*nr)
    lw r1, k    ;pobierz k
    lw r2, nr   ;pobierz nr
    addi r1, r1, #1 ;k => k+1
    mult r3, r1, r2 ; (K+1)*nr
    sw stala, r3    ;wpisz do "stala"

; z4: zwiekszam wartosci w tablicy
    lw r1, rozmiar      ;pobierz rozmiar (licznik)
    addi r2, r0, wektor ;r2 wskazuje na pierwszy element wektora
    loop2:
        lw r4, 0(r2)     ;wczytaj liczbe
        add r5, r4, r3   ;r5 => liczba + stala
        sw 0(r2), r5     ;wpisanie liczby
        addi r2, r2, #4  ;wskaz nastepna liczbe - sizeof(word)=4
        subi r1, r1, #1  ;zmniejsz licznik
        bnez r1, loop2   ;zakoncz jesli licznik = 0

; z5: licze druga sume liczb w nowym wektorze
    lw r1, rozmiar      ;pobierz rozmiar
    addi r2, r0, wektor ;r2 wskazuje na pierwszy element wektora
    lw r10, suma2        ;zeruj rejestr r10
    loop3:
        lw r3, 0(r2)     ;wczytaj liczbe
        add r10, r10, r3 ;dodaj liczbe
        addi r2, r2, #4  ;wskaz nastepna liczbe - sizeof(word)=4
        subi r1, r1, #1  ;zmniejsz licznik
```

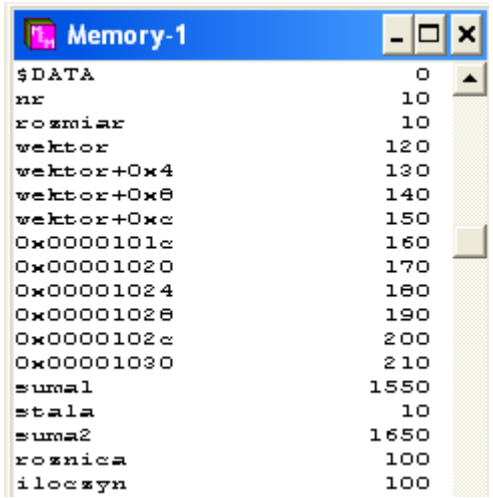
```
        bnez r1, loop3   ;zakoncz jesli licznik = 0
    sw suma2, r10       ;wpisz do "suma2"

; z6: licze roznice
    lw r1, suma1    ;pobierz suma1
    lw r2, suma2    ;pobierz suma2
    sub r10, r2, r1 ;oblicz roznice
    sw roznica, r10 ;wpisz do "roznica"

; z7: licze iloczyn
    lw r1, rozmiar    ;pobierz rozmiar
    lw r2, stala    ;pobierz stala
    mult r10, r2, r1 ;oblicz iloczyn
    sw iloczyn, r10 ;wpisz do "iloczyn"

trap 0
```

C. Zrzuty ekranu z wynikami

Stan początkowy wektora oraz zmiennych



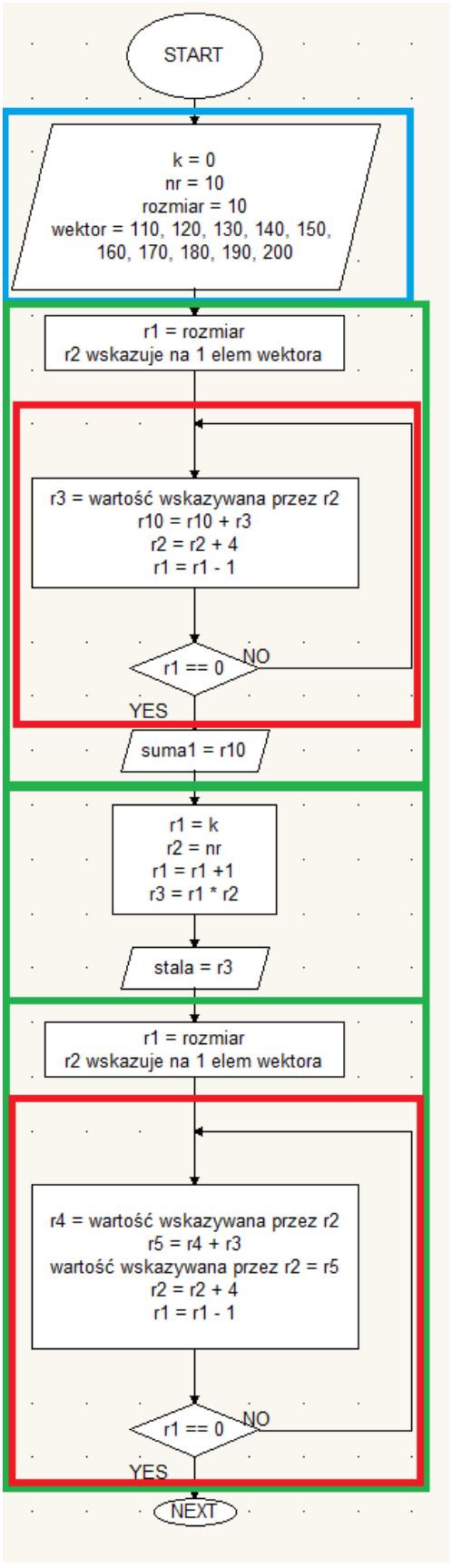
Stan wektora oraz zmiennych po wykonaniu programu



Widok okna statystyk

```
Total:
    284 Cycle(s) executed.
    ID executed by 183 Instruction(s).
    2 Instruction(s) currently in Pipeline.

Hardware configuration:
    Memory size: 32768 Bytes
    faddEX-Stages: 1, required Cycles: 2
    fmulEX-Stages: 1, required Cycles: 5
    fdivEX-Stages: 1, required Cycles: 19
    Forwarding enabled.

Stalls:
    RAW stalls: 70 (24.65% of all Cycles), thereof:
        LD stalls: 32 (45.71% of RAW stalls)
        Branch/Jump stalls: 30 (42.86% of RAW stalls)
        Floating point stalls: 8 (11.43% of RAW stalls)
    WAW stalls: 0 (0.00% of all Cycles)
    Structural stalls: 0 (0.00% of all Cycles)
    Control stalls: 27 (9.51% of all Cycles)
    Trap stalls: 7 (2.46% of all Cycles)
    Total: 104 Stall(s) (36.62% of all Cycles)

Conditional Branches):
    Total: 30 (16.39% of all Instructions), thereof:
        taken: 27 (90.00% of all cond. Branches)
        not taken: 3 (10.00% of all cond. Branches)

Load-/Store-Instructions:
    Total: 55 (30.05% of all Instructions), thereof:
        Loads: 40 (72.73% of Load-/Store-Instructions)
        Stores: 15 (27.27% of Load-/Store-Instructions)

Floating point stage instructions:
    Total: 2 (1.09% of all Instructions), thereof:
        Additions: 0 (0.00% of Floating point stage inst.)
        Multiplications: 2 (100.00% of Floating point stage inst.)
        Divisions: 0 (0.00% of Floating point stage inst.)

Traps:
    Traps: 1 (0.55% of all Instructions)
```

D. Algorytm działania programu



Wczytaj dane

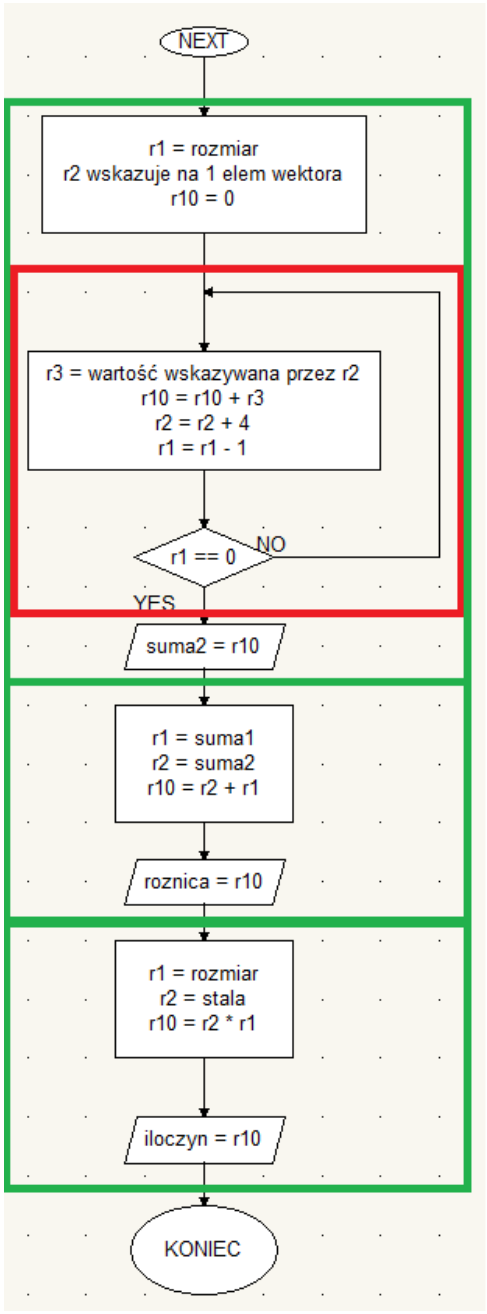
Zadanie 2

Loop1:
r1 to licznik
co krok jest dekrementowany o 1
jeśli r1 > 0 to kolejny krok
jeśli r1 = 0 to koniec pętli

Zadanie 3

Zadanie 4

Loop2:
r1 to licznik
co krok jest dekrementowany o 1
jeśli r1 > 0 to kolejny krok
jeśli r1 = 0 to koniec pętli



Zadanie 5

Loop3:
r1 to licznik
co krok jest dekrementowany o 1
jeśli r1 > 0 to kolejny krok
jeśli r1 = 0 to koniec pętli

Zadanie 6

Zadanie 7

E. Opis rozkazu typu load

Rozkaz: lw r1, rozmiar(r0)

Rozkaz pobiera wartość zmiennej „rozmiar” i wpisuje ją w komórkę r1

Stan początkowy:

```
PC=              256
IMAR=              0
IR=                0
A=                 0
AHI=               0
B=                 0
BHI=               0
BTA=               0
ALU=               0
ALUHI=             0
FPSR=              0
DMAR=              0
SDR=               0
SDRHI=             0
LDR=               0
LDRHI=             0
R0=                0
R1=                0
```

Kolejne kroki działania programu:

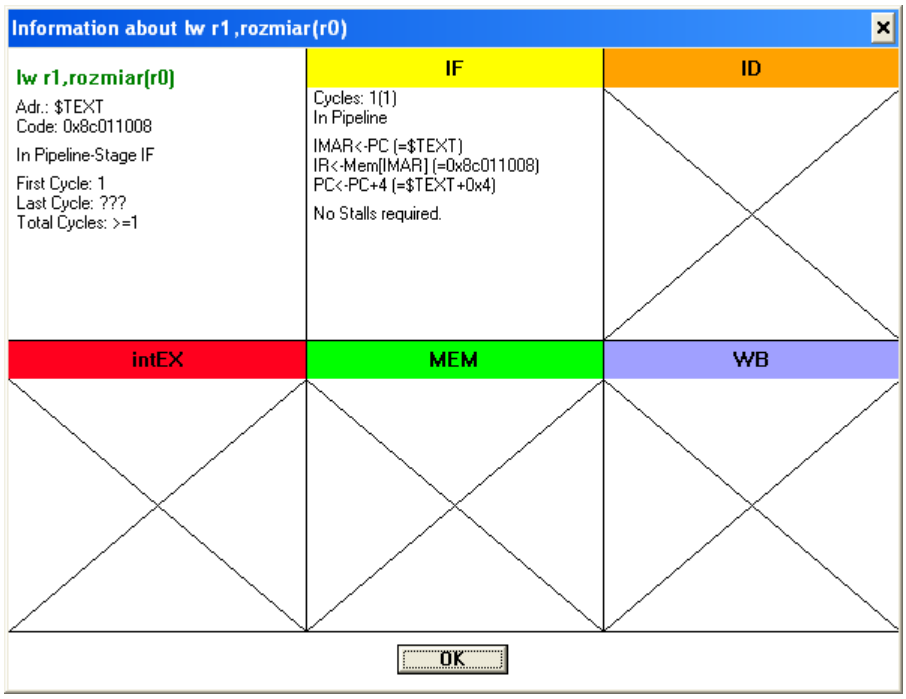
1) Instruction fetch (IF)

- IMAR <- PC – IMAR – pobranie do rejestru IMAR wartości PC
- IR <- MEM[IMAR] – pobranie do IR rozkazu z pamięci na miejscu o numerze IMAR
- PC <- PC+4 – zwiększenie wartości PC o 4 (aby wskazywał następny rozkaz w sekwencji)

IR <- MEM[PC] – pobranie do IR rozkazu z PO (o rozmiarze 4B)

NPC <- PC +4 - zwiększenie wartości NPC o 4 (aby wskazywał następny rozkaz)

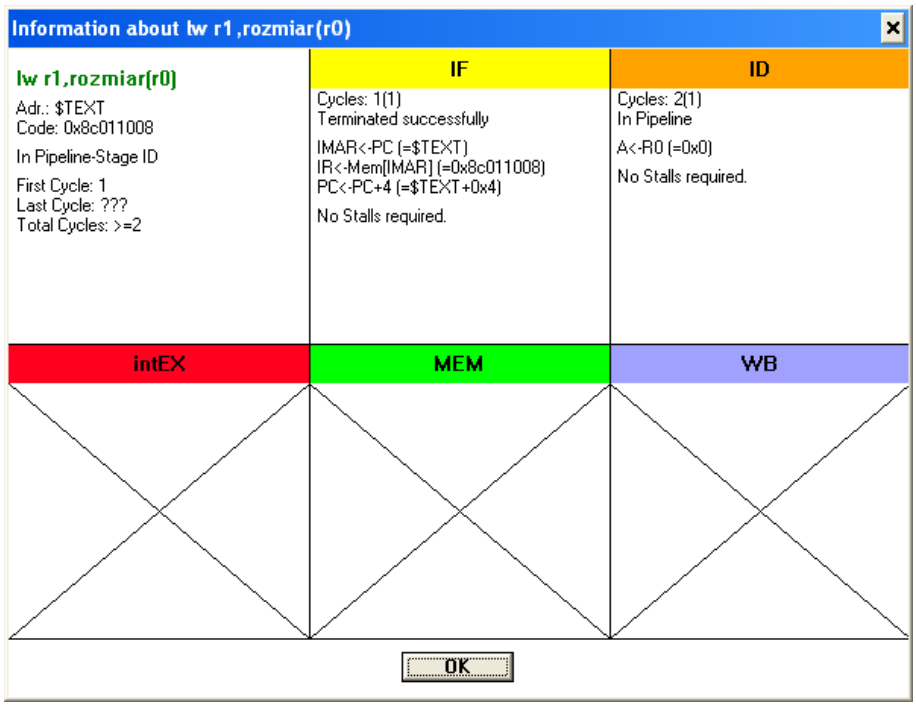
```
PC=              260
IMAR=            256
IR=    -194608741
A=                 0
AHI=               0
B=                 0
BHI=               0
BTA=               0
ALU=               0
ALUHI=             0
FPSR=              0
DMAR=              0
SDR=               0
SDRHI=             0
LDR=               0
LDRHI=             0
R0=                0
R1=                0
```



2) Instruction decode/register fetch (ID)

- A <- R0 – wpisanie do rejestru A wartości z r0 (w tym przypadku 0)

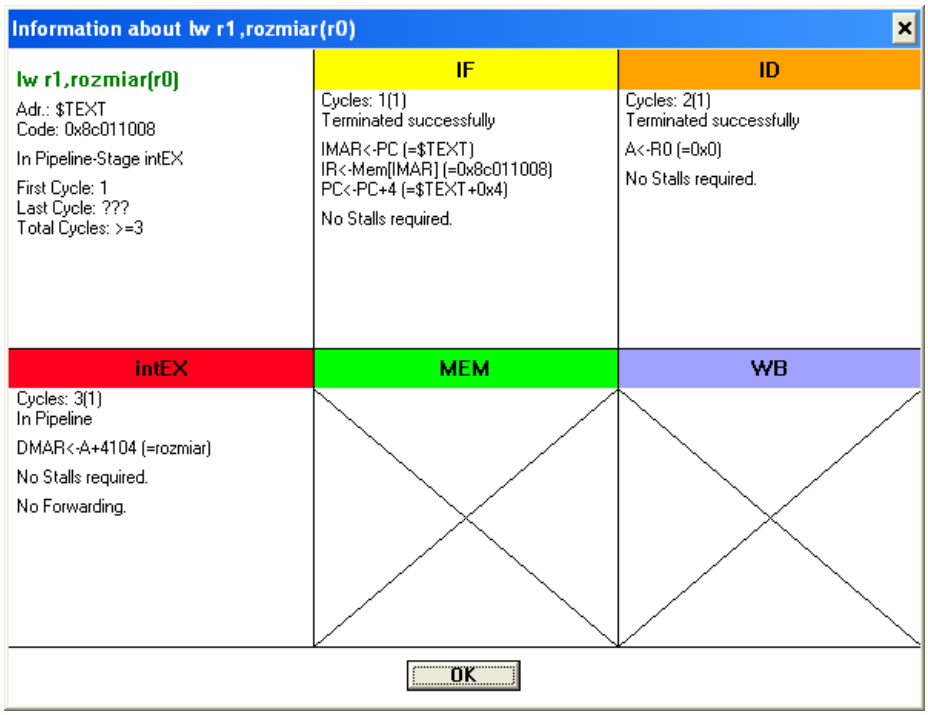
PC=	260
IMAR=	256
IR=	-194608741
A=	0
AHI=	0
B=	0
BHI=	0
BTA=	0
ALU=	0
ALUHI=	0
FPSR=	0
DMAR=	0
SDR=	0
SDRHI=	0
LDR=	0
LDRHI=	0
R0=	0
R1=	0



3) Execution/Effective address (EX)

- DMAR <- A + 4104 – oblicza adres efektywny argumentu i umieszcza go w rejestrze DMAR

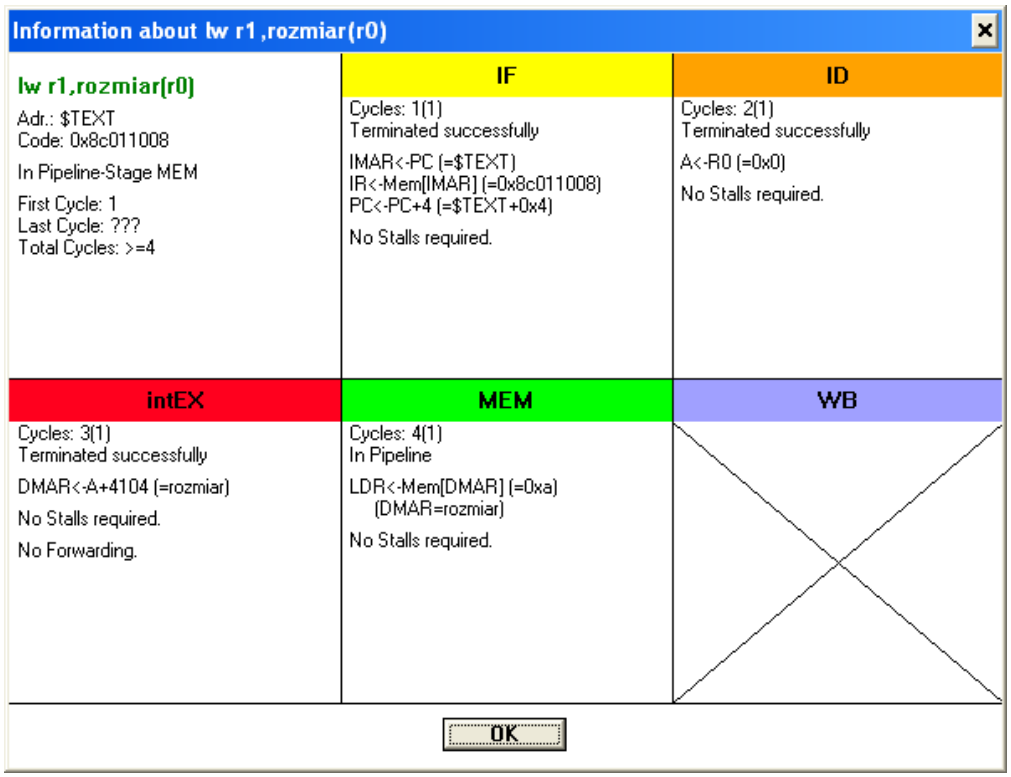
```
PC=              268
IMAR=            264
IR=     -194176614
A=                 0
AHI=               0
B=                 0
BHI=               0
BTA=               0
ALU=               0
ALUHI=             0
FPSR=              0
DMAR=           4104
SDR=               0
SDRHI=             0
LDR=               0
LDRHI=             0
R0=                0
R1=                0
```



4) Memory access/branch completion (MEM)

- LDR <- MEM[DMAR] – pobranie wartości z pamięci na podstawie wartości rejestru DMAR i zwrócenie jej do rejestru LDR (żeby mogło to się zadziać adres został przeniesiony na ALU)

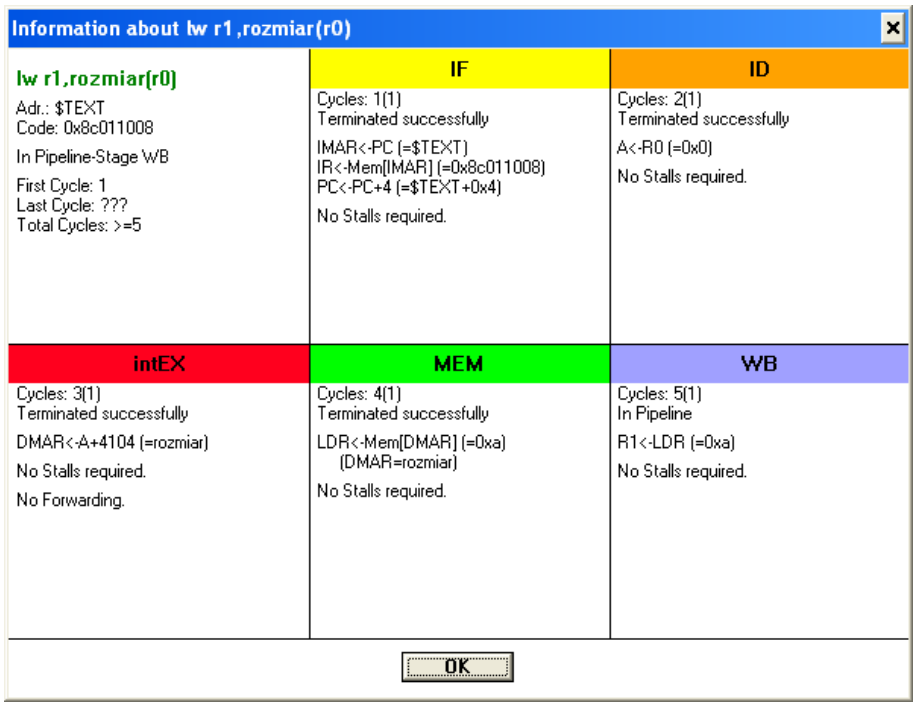
```
PC=            272
IMAR=          268
IR=       21100640
A=                0
AHI=              0
B=                0
BHI=              0
BTA=              0
ALU=           4108
ALUHI=            0
FPSR=             0
DMAR=             0
SDR=              0
SDRHI=            0
LDR=             10
LDRHI=            0
R0=               0
R1=               0
```



5) Write-back (WB)

- R1 <- LDR – załadowanie wartości, wpisanie na rejestr r1 wartość z rejestru LDR

```
PC=           276
IMAR=         272
IR=     541196292
A=              0
AHI=            0
B=              0
BHI=            0
BTA=            0
ALU=            0
ALUHI=          0
FPSR=           0
DMAR=        4108
SDR=            0
SDRHI=          0
LDR=            0
LDRHI=          0
R0=             0
R1=            10
```


Information about lw r1,rozmiar(r0)
lw r1,rozmiar(r0)
Adr.: $TEXT
Code: 0x8c011008
In Pipeline-Stage WB
First Cycle: 1
Last Cycle: ???
Total Cycles: >=5
IF
Cycles: 1(1)
Terminated successfully
IMAR<-PC (=$TEXT)
IR<-Mem[IMAR] (=0x8c011008)
PC<-PC+4 (=$TEXT+0x4)
No Stalls required.
ID
Cycles: 2(1)
Terminated successfully
A<-R0 (=0x0)
No Stalls required.
intEX
Cycles: 3(1)
Terminated successfully
DMAR<-A+4104 (=rozmiar)
No Stalls required.
No Forwarding.
MEM
Cycles: 4(1)
Terminated successfully
LDR<-Mem[DMAR] (=0xa)
(DMAR=rozmiar)
No Stalls required.
WB
Cycles: 5(1)
In Pipeline
R1<-LDR (=0xa)
No Stalls required.
OK