

# *Wojskowa Akademia Techniczna im. Jarosława Dąbrowskiego*

Laboratorium z przedmiotu:  
Architektura i organizacja komputerów

## Sprawozdanie z ćwiczenia laboratoryjnego nr 7: **Hazardy danych w przetwarzaniu potokowym**

### Spis treści

A.	Treść zadania .....	2
B.	Kod programu .....	5
C.	Zrzut ekranu z wynikami.....	6
D.	Zawartość tablic TA oraz TB.....	6
	Zmienna SUMA .....	6
	Tablica TA – pierwsze i ostatnie 10 elementów.....	7
	Tablica TB – pierwsze i ostatnie 10 elementów .....	7
E.	Pomiar liczby cykli zegarowych.....	8
	Włączony forwarding .....	8
	Wyłączony forwarding.....	8
F.	Opisy hazardów w programie .....	9
	Włączony forwarding .....	Error! Bookmark not defined.
	Wyłączony forwarding.....	Error! Bookmark not defined.
G.	Wnioski – wpływ forwardingu na działanie programu.....	11

## A. Treść zadania

### Lab7 (13-14) Y4 prawdziwe

Parametrem zadania będzie **nr = numer\_w\_dzienniku autorki/ autora sprawozdania (numer na liście grupy w USOS)**.

Begin

Napisać program **L7\_nr** w assemblerze komputera WinDLX, który

1. Zadeklaruje dwie tablice przechowujące liczby całkowite:  
**TA** 110- elementową oraz  
**TB** 100-elementową  
a także zmienną zmiennoprzecinkową podwójnej precyzji **.double** o nazwie **Suma** i nada jej wartość początkową zero.
2. Komórki tablicy **TA** wypełni rosnąco kolejnymi liczbami całkowitymi począwszy od  $(1000 + nr)$  – dla  $nr = 5$  będą to odpowiednio 1005, 1006, 1007 itd.
3. Następnie dla każdego elementu tablicy **TB** wykona operację:  
 $TB[i] = (TA[i+2] + TA[i+5]) * 200$
4. W pętli obliczy do rejestru **Rnr** sumę elementów tablicy **TB**.
5. Zawartość **Rnr** zapisze po niezbędnej konwersji do zmiennej **Suma**.

End

Przed zakończeniem czasu zajęć wykonać zrzut ekranowy z wynikami uruchomienia programu **L7\_nr**, pokazujący po lewej stronie okna WinDLX 4 okienka podglądu:

- Pierwszego elementu **tablicy TB**
- Ostatniego elementu **tablicy TB**
- Zmiennej **.double Suma**
- Rejestrów **R i D**, w tym **Rnr z całkowitoliczbową** wersją sumy elementów tablicy **TB** i **D4 (dla całej grupy) ze zmiennoprzecinkową** wersją sumy elementów tablicy **TB**.

A po prawej stronie okna WinDLX zawartość całego okienka **Statistics**. Na zrzucie ma być widoczny **zegar systemowy** z aktualną godziną (prawy dolny róg ekranu Win 10).

Uwaga: symulacja pracy tego programu nie będzie natychmiastowa – troszkę to będzie trwało.

Przykładowy wygląd zrzutu ekranowego (suma STP - w R15, suma ZMP niepoprawnie dla Waszej grupy, ale to tylko przykład prezentacji wyniku - w D2 ):



Floating Point Stage Configuration

Count:

Delay:

Addition Units:

1

2

Multiplication Units:

1

5

Division Units:

1

19

Number of Units in each Class:  $1 \leq M \leq 8$ ,

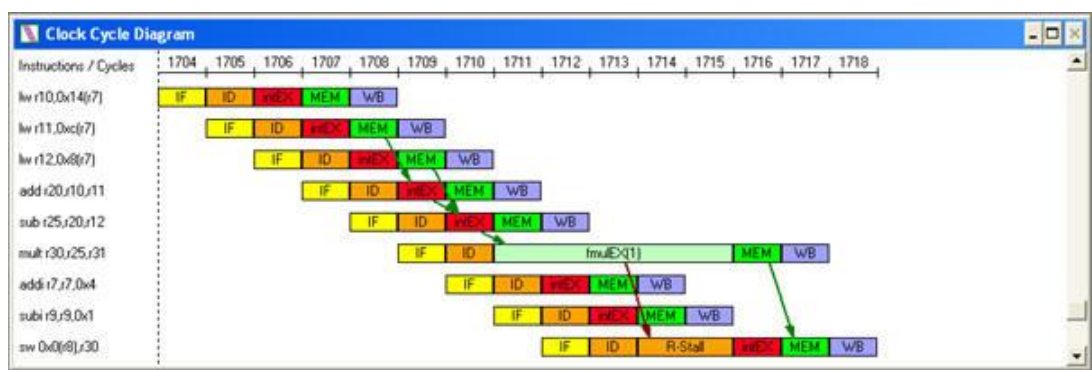
Delay (Clock Cycles):  $1 \leq N \leq 50$

WARNING: If you change the values, the processor will be reset automatically!

OK

Cancel

- B. W sprawozdaniu zamieścić zrzut okienka Clock Cycle Diagram, z odpowiednią zawartością, podobny do poniższego



i porównać diagramy cykli zegarowych dla wykonania jednej iteracji wypełniania tablicy TB - obliczeń z punktu 3 ( $TB[i] = \dots$ ) dla konfiguracji: **1) Enable Forwarding off** i **2) Enable Forwarding on**. Opisać co najmniej 2 występujące dla wyłączonego forwardingu w tej iteracji hazardy danych i przyczyny ich powstania. Wyjaśnić wpływ załączenia forwardingu na działanie WinDLX dla tych hazardów. Skorzystać przy tym z informacji w okienku Information, na przykład:

Information about mult r30,r25,r31		
<b>mult r30,r25,r31</b> Adr.: wypełnijTB+0x14 Code: 0x033f018 Terminated successfully First Cycle: 1709 Last Cycle: 1717 Total Cycles: 9	IF	ID
	Cycles: 1709(1) Terminated successfully IMAR<PC (=wypełnijTB+0x14) IR<Mem[IMAR] (=0x033f018) PC<PC+4 (=wypełnijTB+0x18) No Stalls required.	Cycles: 1710(1) Terminated successfully A<R25 (=0xc) B<R31 (=0xaa) No Stalls required.
<b>fmulEX[1]</b> Cycles: 1711(5) Terminated successfully ALU<A*B (=0xa802) [A=253, B=170] No Stalls required. Forwarding applied: A<0x1d [sub r25,r20,r12]	MEM	WB
	Cycles: 1716(1) Terminated successfully Nothing to do. No Stalls required.	Cycles: 1717(1) Terminated successfully R30<ALU (=0xa802) No Stalls required.

## B. Kod programu

```
.data
TA: .space 440
TB: .space 400
Suma: .double 0
licznik: .word 109
licznik2: .word 100
mnoznik: .word 200

.text
lw r1, licznik      ; pobierz licznik
addi r2, r0, #1010  ; podaj pierwsza wartosc
sw TA(r0), r2       ; wpisz pierwsza wartosc do TA
addi r3, r0, TA      ; Indeks pierwszego elementu TA

tablicaA:
addi r3, r3, #4      ; Nastepny indeks
addi r2, r2, #1      ; Nastepna wartosc
sw 0(r3), r2         ; wpisz do indeksu dana wartosc
subi r1, r1, #1      ; zmniejsz licznik
bnez r1, tablicaA    ; zakoncz, jesli licznik = 0

lw r4, licznik2      ; pobierz licznik2
lw r5, mnoznik       ; pobierz mnoznik
sw TB(r0), r11       ; pobierz indeks 1 elementu TB
lw r7, 4104          ; indeks TA[2]
lw r8, 4116          ; indeks TA[5]

tablicaB:
add r9, r7, r8        ; r9 = TA[i+2] + TA[i+5]
addi r3, r3, #4       ; nastepny indeks (TA[110] -> TB[0])
mult r11, r9, r5       ; r11 = (TA[i+2] + TA[i+5]) * 200
add r10, r10, r11     ; dodaj wartosc do sumy
sw 0(r3), r11         ; wpisz do indeksu wartosc
addi r7, r7, #1       ; nastepny indeks TA[i+2]
addi r8, r8, #1       ; nastepny indeks TA[i+5]
subi r4, r4, #1       ; zmniejsz licznik2
bnez r4, tablicaB    ; zakoncz, jesli licznik2 = 0

movi2fp F4, R10      ; pobierz jako float sume
cvti2d F4, F4        ; skonwertuj jako double
sd Suma, F4          ; wpisz do zmiennej Suma

trap 0
```

## C. Zrzut ekranu z wynikami

The screenshot shows the WINDLX application running in an Oracle VM VirtualBox. The main window displays several panels:

- Memory-3:** TB, 405400
- Memory-4:** TB+0x18c, 445000
- Memory-2:** Suma, 4.25e+07
- Register:** A table of registers (PC, IMAR, IR, A, AHI, B, BHI, BTA, ALU, ALUHI, FPCR, DMAP, SDR, SDRHI, LDR, LDRHI, R0-R9) and their values.
- Statistics:** A detailed report of execution statistics, including total cycles, instructions, hardware configuration, stalls, conditional branches, load/store instructions, and floating point stage instructions.

The **Statistics** panel provides the following data:

- Total:** 2279 Cycle(s) executed, 1458 Instruction(s) executed, 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: 610 (26.77% of all Cycles), LD stalls: 1 (0.16% of RAW stalls), Branch/Jump stalls: 209 (34.26% of RAW stalls), Floating point stalls: 400 (65.57% of RAW stalls), WAW stalls: 0 (0.00% of all Cycles), Structural stalls: 0 (0.00% of all Cycles), Control stalls: 207 (9.08% of all Cycles), Trap stalls: 3 (0.13% of all Cycles), Total: 820 Stall(s) (35.98% of all Cycles).
- Conditional Branches:** Total: 209 (14.33% of all Instructions), taken: 207 (99.04% of all cond. Branches), not taken: 2 (0.96% of all cond. Branches).
- Load-/Store-Instructions:** Total: 217 (14.88% of all Instructions), Loads: 5 (2.30% of Load-/Store-Instructions), Stores: 212 (97.70% of Load-/Store-Instructions).
- Floating point stage instructions:** Total: 100 (6.86% of all Instructions), Additions: 0 (0.00% of Floating point stage inst.), Multiplications: 100 (100.00% of Floating point stage inst.), Divisions: 0 (0.00% of Floating point stage inst.).

The **Register** panel shows the following values:

Register	Value
PC	368
IMAR	364
IR	0
A	0
AHI	0
B	0
BHI	0
BTA	0
ALU	0
ALUHI	0
FPCR	0
DMAP	4936
SDR	0
SDRHI	1099187822
LDR	0
LDRHI	0
R0	0
R1	0
R2	1119
R3	4932
R4	0
R5	200
R6	0
R7	1112
R8	1115
R9	2225

The **Statistics** panel also shows the following values:

Register	Value
Suma	4.252e+07
licznik	0
licznik2	0
monoznik	0
0x0000135c	0
0x00001360	0
0x00001364	0
0x00001368	0
0x0000136c	0
0x00001370	0
0x00001374	0
0x00001378	0

## D. Zawartość tablic TA oraz TB

Zmienna SUMA

The screenshot shows the **Memory-2** window with the variable **Suma** and its value **4.252e+07**.

## Tablica TA – pierwsze i ostatnie 10 elementów

(Po lewej – 10 od góry, po prawej – 10 od dołu)

Memory-1		Memory-5	
TA	1010	0x00001124	1083
\$DATA+0x4	1011	0x00001128	1084
\$DATA+0x8	1012	0x0000112c	1085
\$DATA+0xc	1013	0x00001130	1086
\$DATA+0x10	1014	0x00001134	1087
\$DATA+0x14	1015	0x00001138	1088
\$DATA+0x18	1016	0x0000113c	1089
\$DATA+0x1c	1017	0x00001140	1090
\$DATA+0x20	1018	0x00001144	1091
\$DATA+0x24	1019	0x00001148	1092
\$DATA+0x28	1020	0x0000114c	1093
\$DATA+0x2c	1021	0x00001150	1094
\$DATA+0x30	1022	0x00001154	1095
\$DATA+0x34	1023	0x00001158	1096
\$DATA+0x38	1024	0x0000115c	1097
\$DATA+0x3c	1025	0x00001160	1098
\$DATA+0x40	1026	0x00001164	1099
\$DATA+0x44	1027	0x00001168	1100
\$DATA+0x48	1028	0x0000116c	1101
\$DATA+0x4c	1029	0x00001170	1102
\$DATA+0x50	1030	0x00001174	1103
\$DATA+0x54	1031	0x00001178	1104
\$DATA+0x58	1032	0x0000117c	1105
\$DATA+0x5c	1033	0x00001180	1106
\$DATA+0x60	1034	0x00001184	1107
\$DATA+0x64	1035	0x00001188	1108
\$DATA+0x68	1036	0x0000118c	1109
\$DATA+0x6c	1037	0x00001190	1110
\$DATA+0x70	1038	0x00001194	1111
\$DATA+0x74	1039	0x00001198	1112
\$DATA+0x78	1040	0x0000119c	1113
\$DATA+0x7c	1041	0x000011a0	1114
\$DATA+0x80	1042	0x000011a4	1115
\$DATA+0x84	1043	0x000011a8	1116
\$DATA+0x88	1044	0x000011ac	1117
\$DATA+0x8c	1045	0x000011b0	1118
\$DATA+0x90	1046	0x000011b4	1119

## Tablica TB – pierwsze i ostatnie 10 elementów

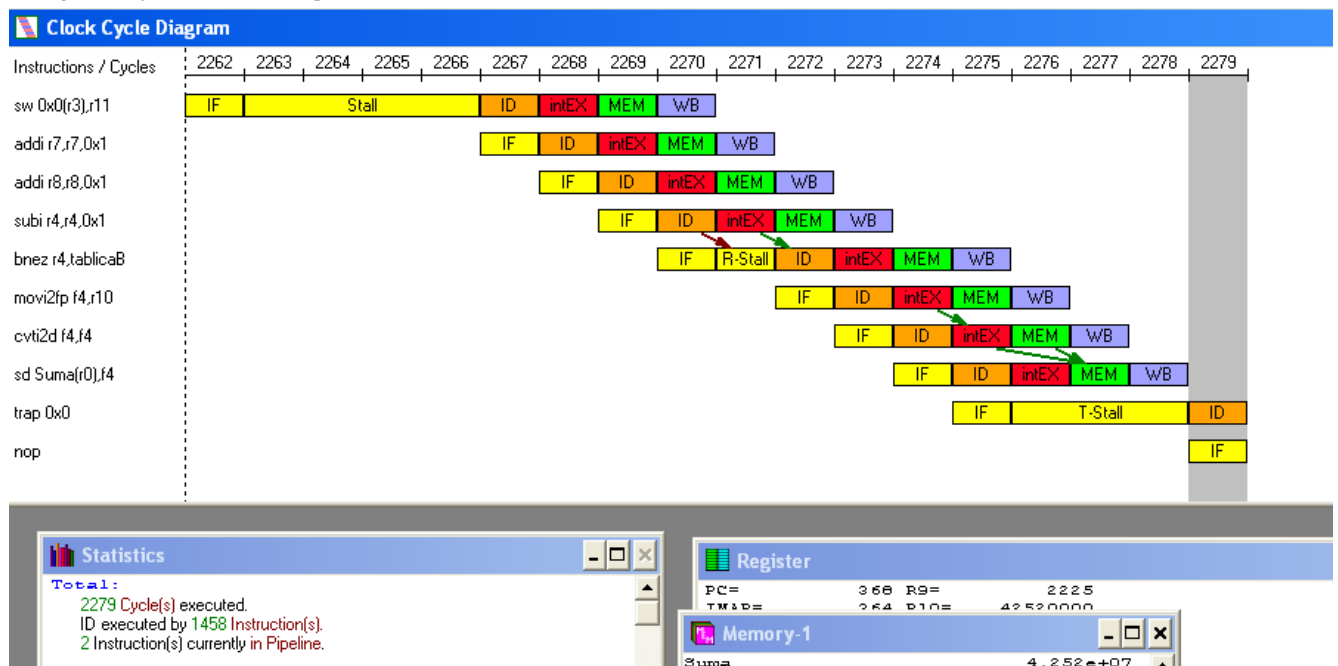
(Po lewej – 10 od góry, po prawej – 10 od dołu)

Memory-3		Memory-4	
TB	405400	TB+0x100	431000
TB+0x4	405800	TB+0x104	431400
TB+0x8	406200	TB+0x108	431800
TB+0xc	406600	TB+0x10c	432200
TB+0x10	407000	TB+0x110	432600
TB+0x14	407400	TB+0x114	433000
TB+0x18	407800	TB+0x118	433400
TB+0x1c	408200	TB+0x11c	433800
TB+0x20	408600	TB+0x120	434200
TB+0x24	409000	TB+0x124	434600
TB+0x28	409400	TB+0x128	435000
TB+0x2c	409800	TB+0x12c	435400
TB+0x30	410200	TB+0x130	435800
TB+0x34	410600	TB+0x134	436200
TB+0x38	411000	TB+0x138	436600
TB+0x3c	411400	TB+0x13c	437000
TB+0x40	411800	TB+0x140	437400
TB+0x44	412200	TB+0x144	437800
TB+0x48	412600	TB+0x148	438200
TB+0x4c	413000	TB+0x14c	438600
TB+0x50	413400	TB+0x150	439000
TB+0x54	413800	TB+0x154	439400
TB+0x58	414200	TB+0x158	439800
TB+0x5c	414600	TB+0x15c	440200
TB+0x60	415000	TB+0x160	440600
TB+0x64	415400	TB+0x164	441000
TB+0x68	415800	TB+0x168	441400
TB+0x6c	416200	TB+0x16c	441800
TB+0x70	416600	TB+0x170	442200
TB+0x74	417000	TB+0x174	442600
TB+0x78	417400	TB+0x178	443000
TB+0x7c	417800	TB+0x17c	443400
TB+0x80	418200	TB+0x180	443800
TB+0x84	418600	TB+0x184	444200
TB+0x88	419000	TB+0x188	444600
TB+0x8c	419400	TB+0x18c	445000

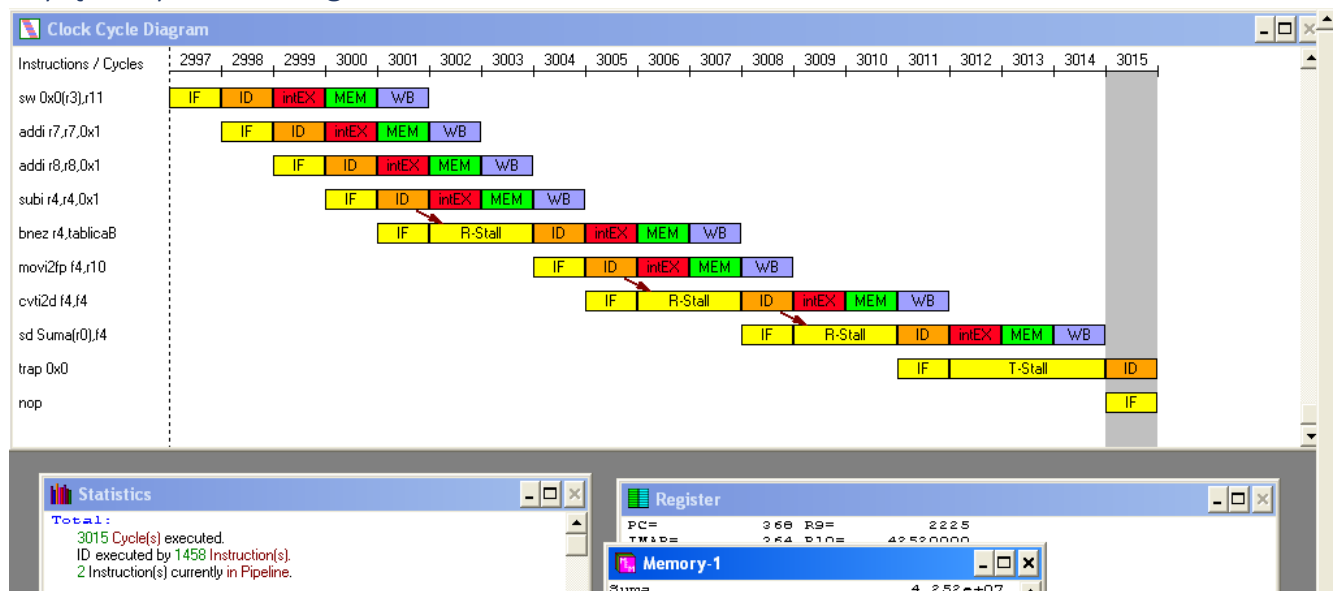
## E. Pomiar liczby cykli zegarowych

Forwarding	Ilość cykli	Wartość zmiennej Suma
Włączony	2279	4.252e+07
Wyłączony	3015	4.252e+07

### Włączony forwarding



### Wyłączony forwarding



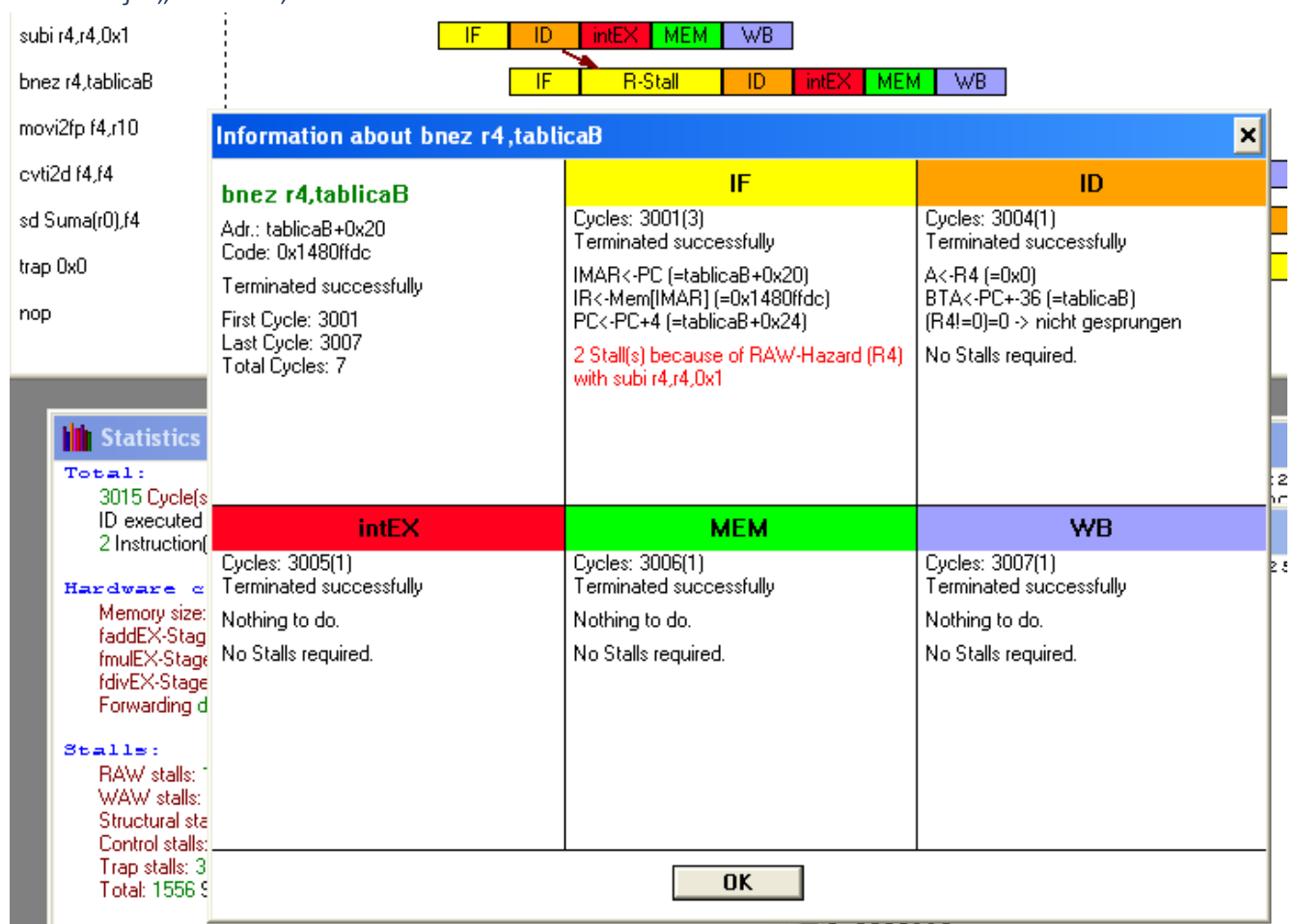


## F. Opisy hazardów dla wyłączanego forwardingu

Hazard jest to zjawisko, które wynika z faktu, że nie wszystkie instrukcje są w stanie wykonać się w trakcie jednego cyklu. Przez to, jeśli instrukcje następujące po wspomnianych wykorzystują te same elementy pamięci, przez wzgląd na bezpieczeństwo zmuszone są do wstrzymania działania i oczekiwania na pełne wykonanie swojego poprzednika. Jest to sytuacja niekorzystna, prowadząca do zwiększania się liczby cykli, w których wykonuje się program.

W celu zmniejszenia ilości cykli w programie można zastosować forwarding. Zmniejsza on czas przestoju (Stall) poprzez wykorzystywanie wyników poprzedniej instrukcji.

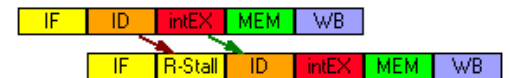
instrukcja „bnez r4, tablicaB”



Czerwona strzałka wskazuje miejsce, w którym konieczne jest zastosowanie „R-Stall”. Przyczyną tego jest to, że instrukcja wskazywana przez strzałkę (`bnez`) musi poczekać dodatkową ilość cykli (2), żeby poprzednia instrukcja (`subi`) mogła się wykonać (zjawisko hazardu). Przyczyna leży po stronie instrukcji poprzedniej, ponieważ wymagała ona więcej czasu na działanie, stąd instrukcja „`bnez`” musi czekać. W przypadku zastosowania forwardingu oprogramowanie wykorzystałoby wynik poprzedniej instrukcji do obliczeń, co skróciłoby czas wykonania tej instrukcji (w tym przypadku o 1 cykl).

Poniżej przedstawiony jest diagram uwzględniający forwarding, obrazujący skrócenie czasu wykonania o 1 cykl (zaznaczone przy pomocy zielonej strzałki).

```
subi r4,r4,0x1
bnez r4,tabelaB
```

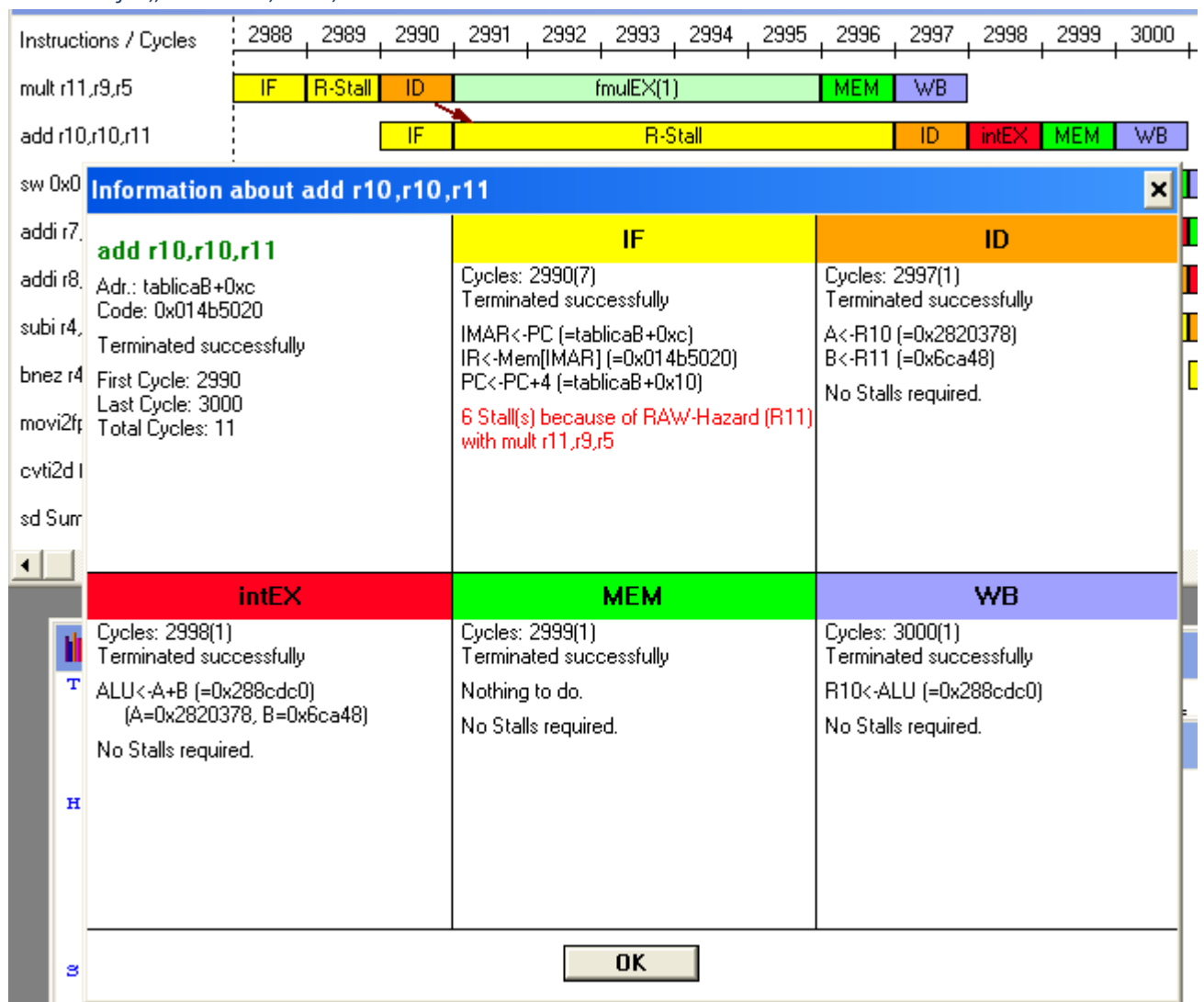


**intEX**

Cycles: 2273(1)  
 Terminated successfully  
 Nothing to do.  
 No Stalls required.  
 Forwarding applicated:  
 A<-0x0 (subi r4,r4,0x1)

Zastosowana operacja:

instrukcja „add r10, r10, r11”

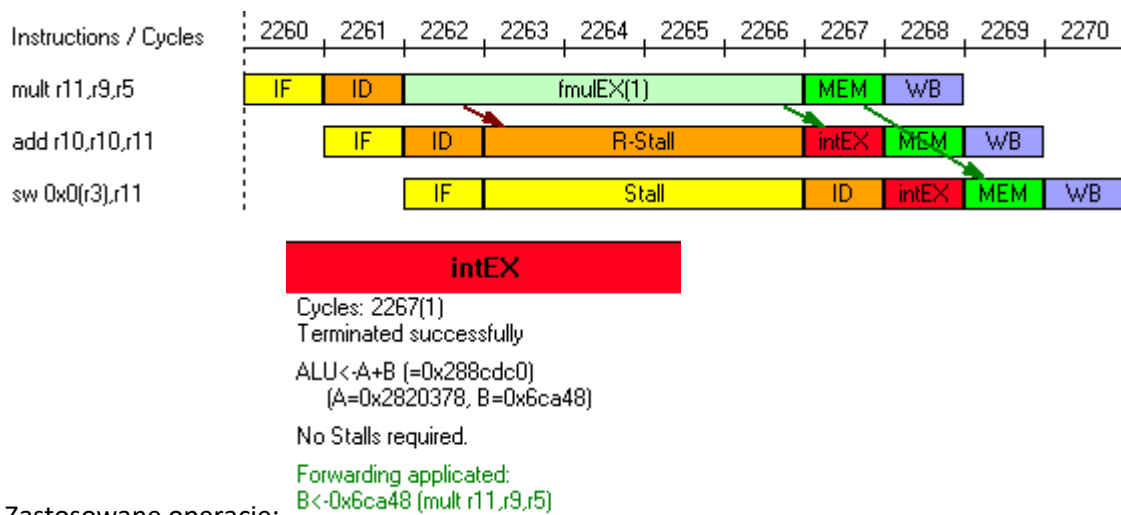


Ponieważ mnożenie jest dość złożone pod względem czasu wykonania, a instrukcja „add” potrzebuje tego wyniku do wykonania swojej instrukcji (potrzebuje wartości rejestru r11, do którego właśnie zwracany jest wynik mnożenia), musi ona się wstrzymać (dokładnie na 6 cyklach). Dopiero po zakończeniu mnożenia, wznowia działanie.

W przypadku zastosowania forwardingu czas wstrzymania zostanie zmniejszony o 2 cykle, ponieważ instrukcja dodawanie wykorzysta wartości obliczone w trakcie wykonania instrukcji mnożenia. W międzyczasie rozpoczyna się działanie kolejnej instrukcji (sw), która również potrzebuje do wykonania wartości rejestru r4, w

związku z tym również zostaje wstrzymana. Nie dzieje się to w przypadku wyłączonego forwardingu, gdyż tam instrukcja ta rozpoczyna swoje działanie dopiero po wznowieniu instrukcji „add”.

Poniżej przedstawiony jest diagram uwzględniający forwarding, z widoczną różnicą ilości wymaganych cykli.



Zastosowane operacje:

## G. Wnioski – wpływ forwardingu na działanie programu

1. Forwarding nie ma wpływu na wynik końcowy – wartość obliczonej sumy jest niezmienna
2. Przez zaistnienie zjawiska hazardu w danym momencie programu instrukcje muszą zostać wstrzymywane na jakiś czas (Stall), aby poprzednie były w stanie się w pełni wykonać.
3. Wstrzymanie wykonania instrukcji wynika z tego, że korzysta ona z rejestrów, które wykorzystywane są przez instrukcję poprzedzającą. Bez wstrzymania mogłoby dojść do nieścisłości, podczas których zostałaaby pobrana błędna wartość.
4. W celu optymalizacji ilości cykli potrzebnych do wykonania programu można zastosować forwarding. Wykorzystuje on ponownie wyniki poprzedniej operacji, co skraca czas jego wykonania.
5. Niewielkie optymalizacje wykonane przez forwarding wewnątrz pętli instrukcji zaczynają mieć kardynalne znaczenie w ostatecznym czasie działania programu.