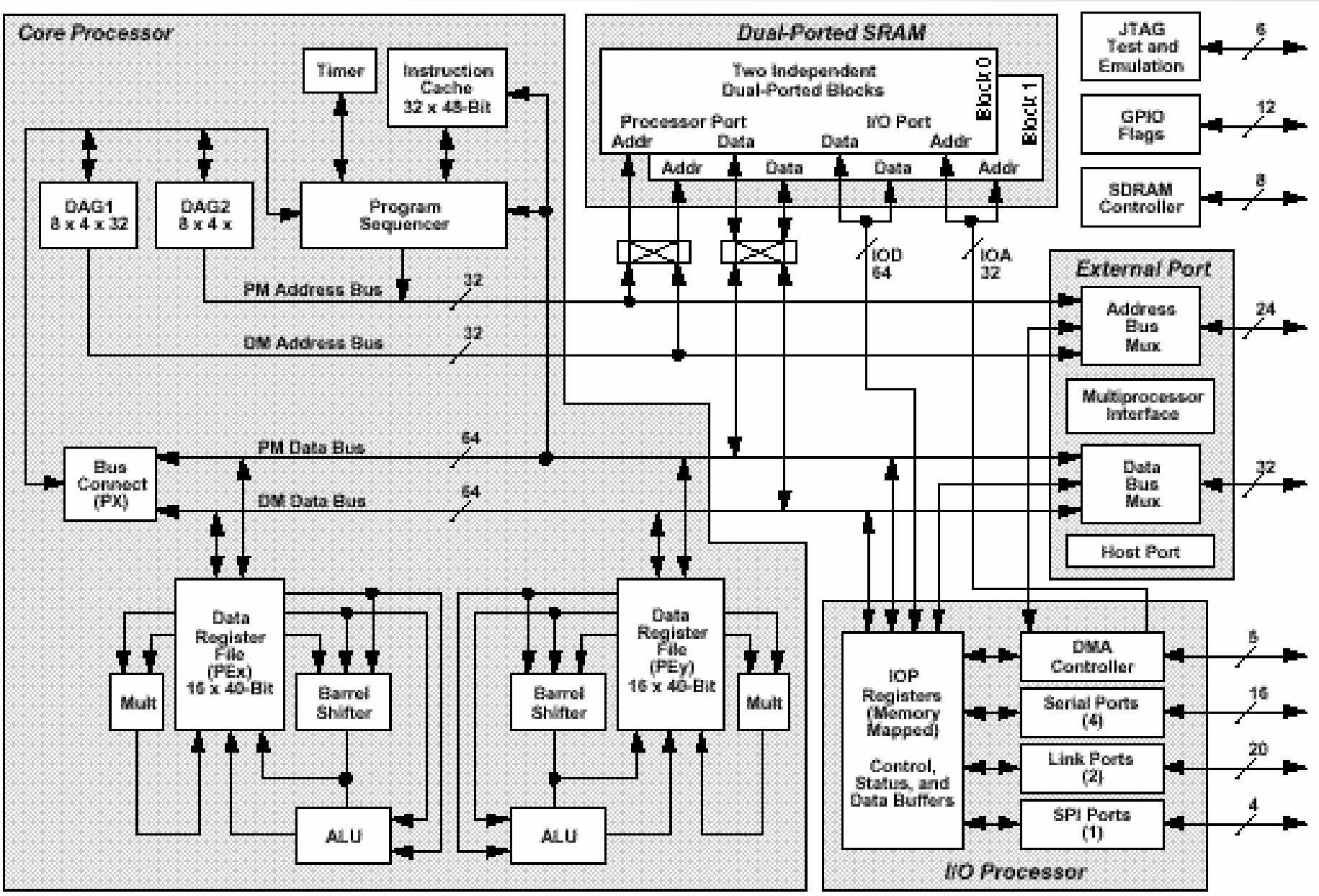
The ADSP-21261 Architektura jadra II

Sekcja 4

Generatory adresów danych

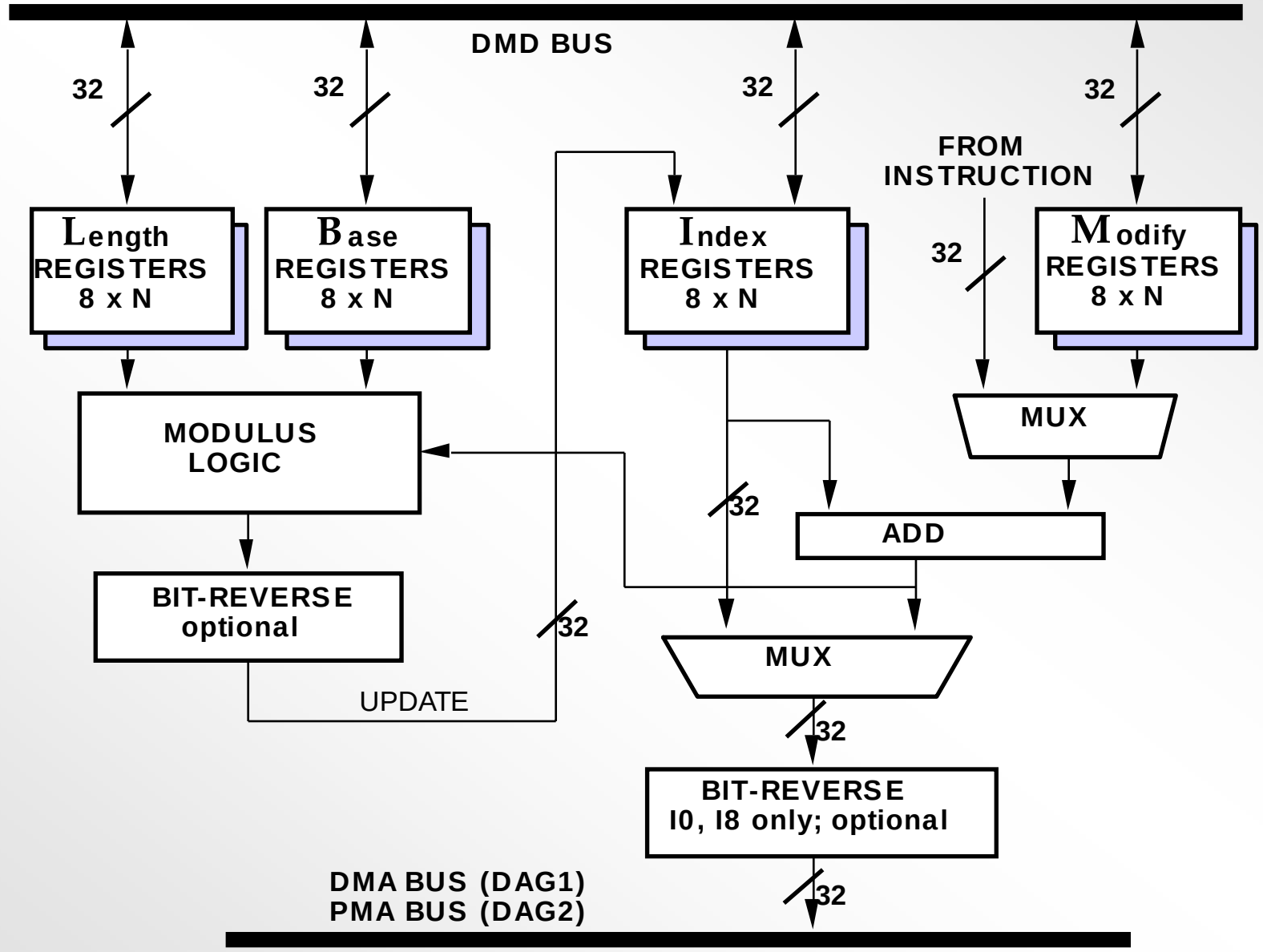
ADSP-21261-Diagram blokowy architektury



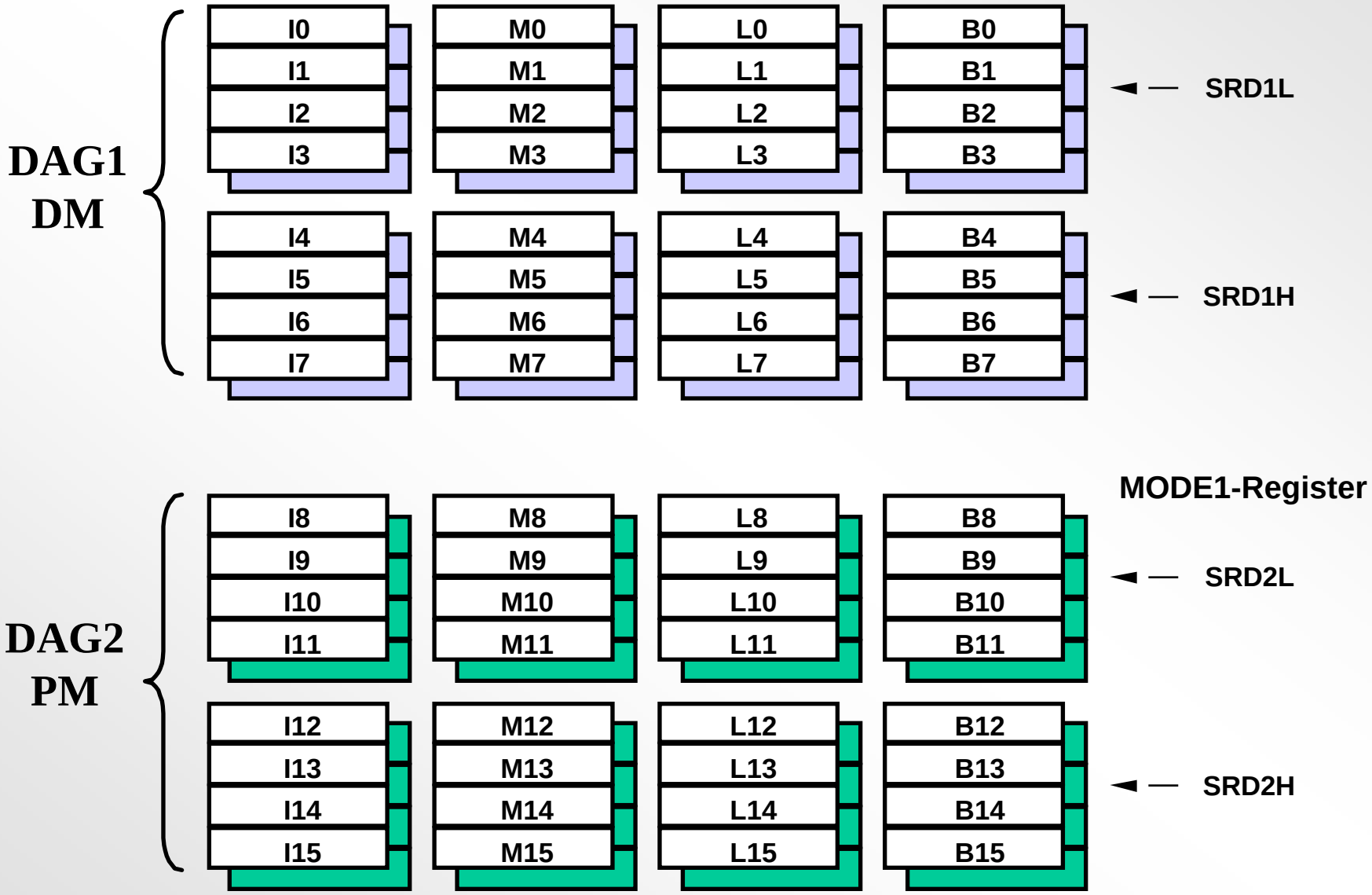
Metody adresowania

- **Immediate(natychmiastowe)/Direct(bezposred.) (32 bity)**
- **Indirect(posrednie)**
  - **Post-modify z aktualizacja**
  - **Pre-modify bez aktualizacji**
- **Addresowanie modulo dla buforów kolowych**
  - **Rejestr bazowy pozwala na niewymuszone umieszcznie bufora**
- **Bit Reversal(odwracanie bitow)**

Diagram blokowy generatora adresu danej(DAG)



Rejestry DAG (pokazane zamienniki)



Przykladowe instrukcje generatora adresów(DAG)

```
R3 = DM(0x00000000);      /* bezposredni odczyt pamieci */
DM(I3,M0) = 0x1234;       /* posredni zapis do pamieci */
DM(I4,M3) = TCOUNT;       /* posredni zapis do pamieci */
F0 = PM(I10,M10);         /* posredni odczyt pamieci (PM) */
F0 = PM(M10,I10);         /* posredni odczyt w/ pre-modify */
F0 = PM(0x400,I10);       /* posredni odczyt w/ immed. pre-mod */
MODIFY (I0,M0);           /* Post modify */
BITREV(I3,4);             /* odwrocenie bitow I3+4 */
```

Instrukcje DAG 1 z 2

Instrukcje Compute i Move lub Modify : (Uwaga: wielofunkcyjne operacje)
Items in italics sa opcjomalnymi czesciami instrukcji.

1. ***compute*,** | **DM(Ia, Mb) = dreg1** / **dreg1 = DM(Ia, Mb)** | **,** | **PM(Ic, Md) = dreg2** / **dreg2 = PM(Ic, Md)** | **;**

2. ***IF condition*** ***compute*** **;**

3a. ***IF condition*** ***compute*,** | **DM(Ia, Mb)** / **PM(Ic, Md)** | **= ureg ;**

3b. ***IF condition*** ***compute*,** | **DM(Mb, Ia)** / **PM(Md, Ic)** | **= ureg ;**

3c. ***IF condition*** ***compute*,** **ureg =** | **DM(Ia, Mb)** / **PM(Ic, Md)** | **;**

3d. ***IF condition*** ***compute*,** **ureg =** | **DM(Mb, Ia)** / **PM(Md, Ic)** | **;**

4a. ***IF condition*** ***compute*,** | **DM(Ia, <data6>)** / **PM(Ic, <data6>)** | **= dreg ;**

4b. ***IF condition*** ***compute*,** | **DM(<data6>, Ia)** / **PM(<data6>, Ic)** | **= dreg ;**

4c. ***IF condition*** ***compute*,** **dreg =** | **DM(Ia, <data6>)** / **PM(Ic, <data6>)** | **;**

4d. ***IF condition*** ***compute*,** **dreg =** | **DM(<data6>, Ia)** / **PM(<data6>, Ic)** | **;**

5. ***IF condition*** ***compute*,** **ureg1 = ureg2 ;**

Instrukcje DAG 2 z 2

6a. *IF condition* *shiftimm,* | DM(Ia, Mb) / PM(Ic, Md) | = dreg ;

6b. *IF condition* *shiftimm,* dreg = | DM(Ia, Mb) / PM(Ic, Md) | ;

7. *IF condition* *compute,* MODIFY | (Ia, Mb) / (Ic, Md) | ;

Immediate Move Instructions

14a. | DM(<addr32>) / PM(<addr32>) | = ureg ;

14b. ureg = | DM(<addr32>) / PM(<addr32>) | ;

15a. DM(<data32>, Ia) = ureg ;
PM(<data32>, Ic)

15b. ureg = | DM(<data32>, Ia) / PM(<data32>, Ic) | ;

16. | DM(Ia, Mb) / PM(Ic, Md) | = <data32> ;

17. ureg = <data32> ;

19a. BITREV (Ia,<data32>;

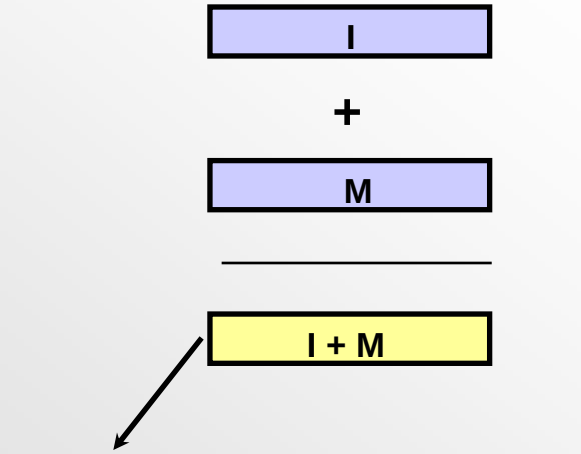
Adresowanie posrednie: Pre-Modify

ureg = DM (Mx , Ix)

ureg = PM (Mx , Ix)

Pre-modify z rejestrem M, bez aktualizacji

Przyklady: r1=pm(m10,i15);
r8=dm(m3,i1);



tylko wyjscie, bez aktualizacji

Pre-modify z bezposrednia wartoscia,

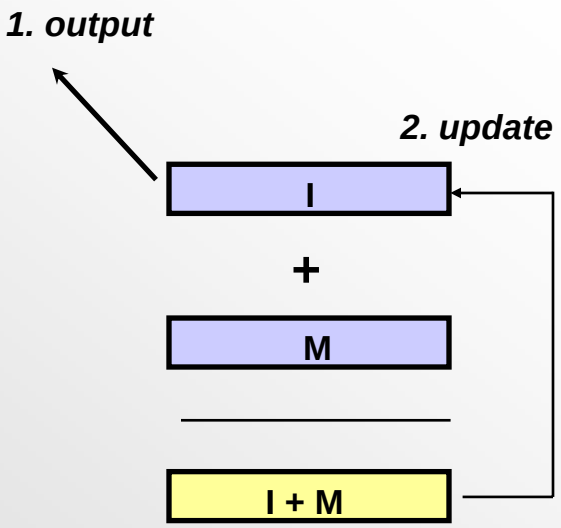
bez aktualizacji

Przyklady: if av r1=pm(0x11,i15);
dm(127,i5)=addr_1;

Adresowanie posrednie: Post-Modify

ureg = DM (Ix , Mx)

ureg = PM (Ix , Mx)



- **z rejestrem M , aktualizacja rejestru I**

Przyklady: f5=pm(i9,m12);
dm(i0,m3)=r3;
dm(i0,0)=0x1234;

- **z bezposrednia wartoscia, aktualizacja rejestru I**

Przyklady: f15=dm(i0,6);
if av r1=pm(i15,0x11);

Adresowanie posrednie: Modify

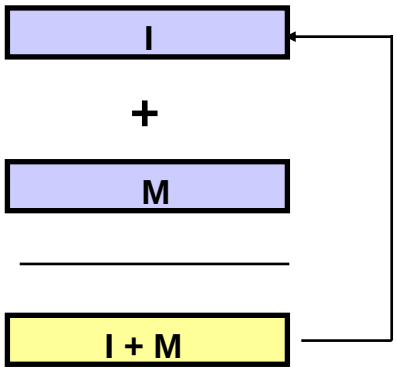
modify (Ix , Mx)

modify (Ix , Mx)

- **z rejestrem M , z/ aktualizacja**

Przyklady: modify (i15, m10);
modify (i5, m6);

only update, no output



- **bezposrednia wartosc, z/ aktualizacja**

Przyklad: modify (i4, 304);
modify (i8, 24);

Rejestrowe posrednie adresowanie pamieci

Post-modify z rejestrem M , aktualizacja rejestru I

Przyklady: f5=pm(i9,m12);
dm(i0,m3)=r3, r1=pm(i15,m10);

Pre-modify z rejestrem M , bez aktualizacji

Przyklady: r1=pm(m10,i15);
jump(m13,i11);

Post-modify z bezposrednia wartoscia, aktualizacja rejestru I

Przyklady: f15=dm(i0,6);
if av r1=pm(i15,0x11);

Pre-modify z bzposrednia wartoscia, bez aktualizacji

Przyklady: if av r1=pm(0x11,i15);
dm(127,i5)=laddr;

Buforowanie kolowe z DAGs

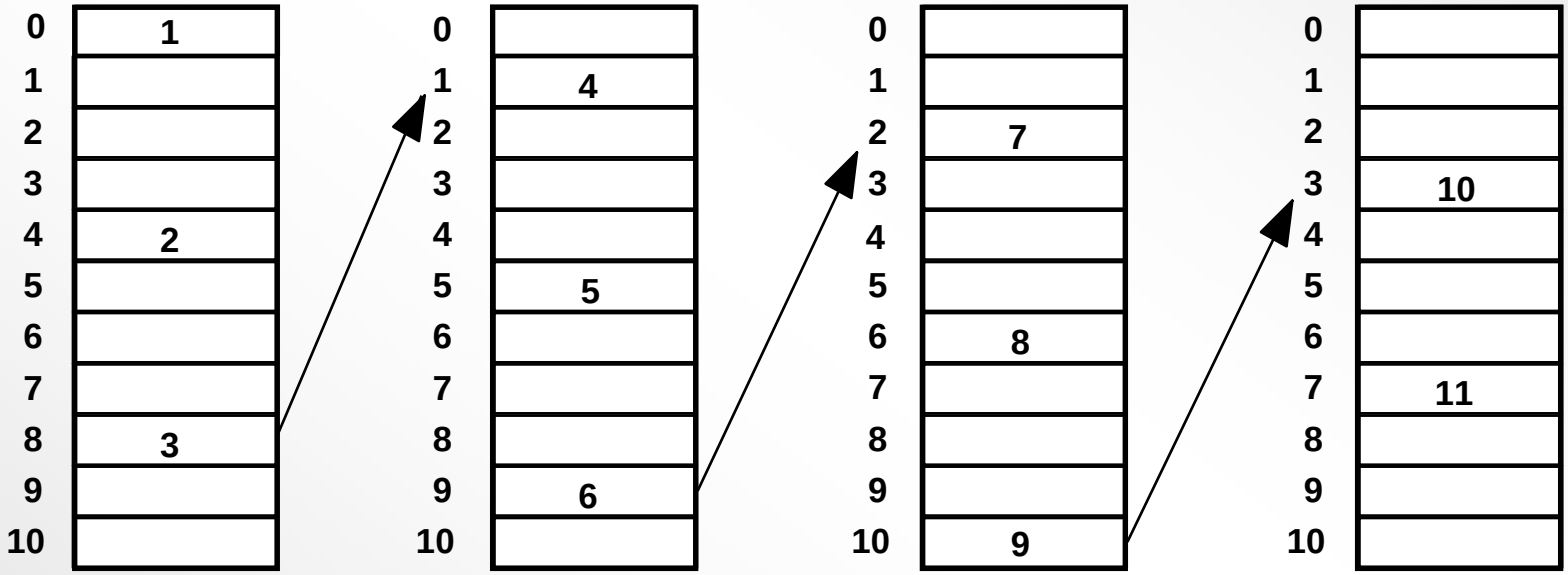
- **Bit Global circular buffer enable (CBUFEN) w trybie MODE1**
- **DAG1 i DAG2 wspieraja buforowanie kolowe**
- **Rejestry L okreslaja Length(dlugosc)**
- **Rejestry I okreslaja adres (Index) wewnatrz bufora**
- **Rejestry B precyzuja adres bazowy**

  Uwaga: Zainicjalizowanie rejestru B automatycznie inicjalizuje zwiazany z z nim rejestr I

- **Rejestry M okreslaja wartosc offsetu (Modify) moga byc dodatnie i ujemne**
- **tylko operacje Post-Modify pracuja z buforami kolowymi**

Bufory kolowe (Adresowanie Modulo)

Dlugosc = 11 **Adres bazowy= 0Modifier (rozmiar kroku) = 4**



Sequence shows order in which locations are accessed in one pass. Sequence repeats on subsequent passes.

Przykladowy kod adresowania modulo

```
#include “def21161.h”
.SECTION/DM  example_DM;          /*DM segment                        */
.VAR  Buff[11];                   /*Define Buffer                     */
                                  /*End of segment                    */
                                  /*                                  */
.SECTION/PM  example_PM;          /*PM segment                        */
L0 = length(Buff);                /*L0 = Length of Buff               */
M0 = 4;                           /*Modify value = 4                  */
B0 = Buff;                        /*B0 = Base address                 */
//I0 = Buff; done automatically     I0 = Start address of Buff
Bit Set MODE1 CBUFEN;             /*Enables circular buffering        */
R1 = DM (I0, M0);                 /*Fetch 0                           */
R2 = DM (I0, M0);                 /*Fetch 4                           */
R3 = DM (I0, M0);                 /*Fetch 8                           */
R4 = DM (I0, M0);                 /*Fetch 1                           */
•                                 /*                                  */
•                                 /*                                  */
•                                 /*                                  */
R11 = DM (I0, M0);                /*Fetch 7                           */
Bit CLR MODE1 CBUFEN;             /*Disables circular buffering       */
                                  /*End of segment                    */
```

ADSP-21161: Odwracanie bitów

Odwracanie bitów w ADSP-21161 moze byc zrealizowane dwoma sposobami:

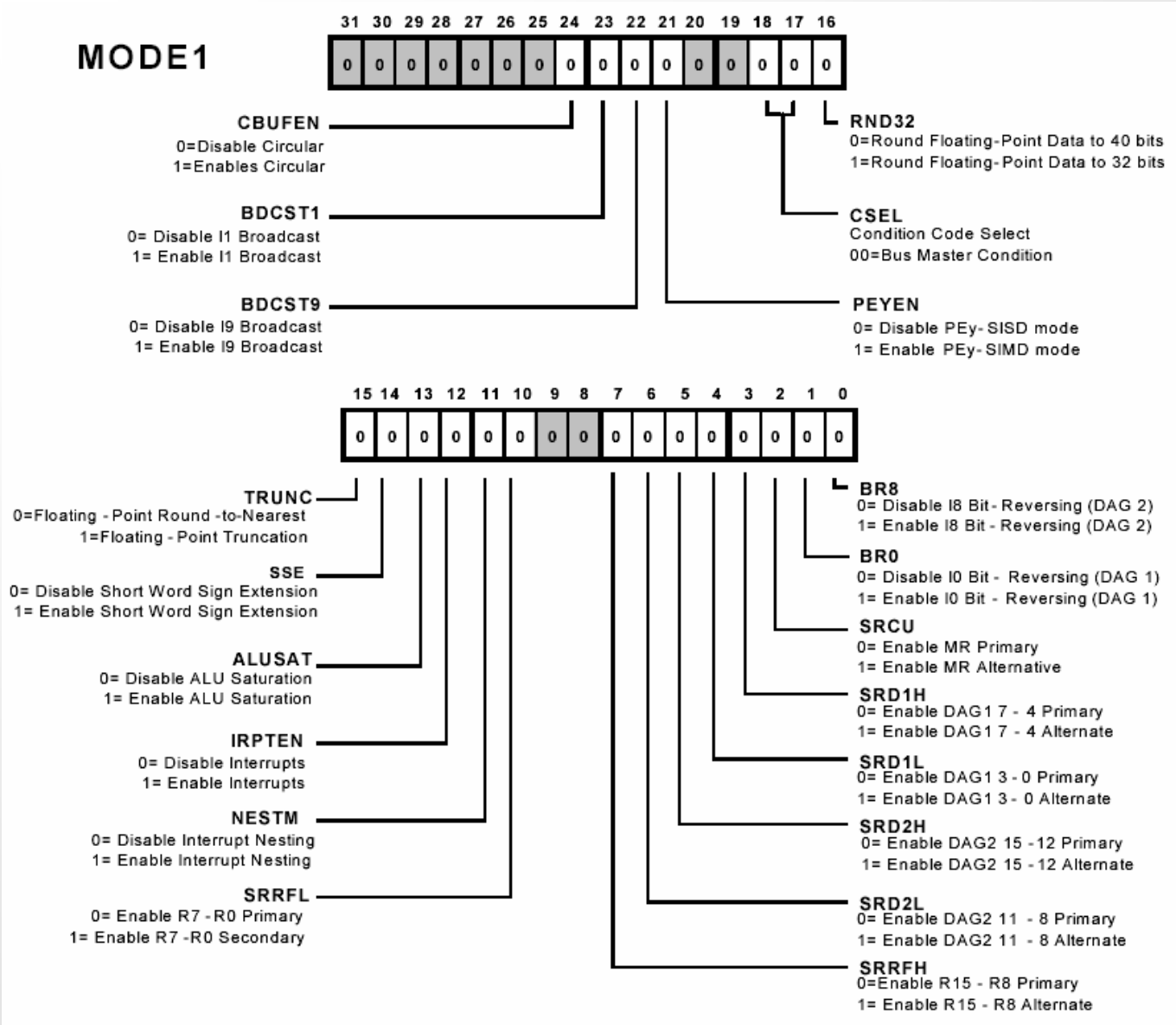
- Tryb Bit Reverse :

  - DAG1 bit-odwraca 32-bitowe wyjscie adresowe z I0
  - DAG2 bit-odwraca 32-bitowe wyjscie adresowe z I8
  - Zawartosci rejestru I0,I8 sa niezmieniane
  - Bit 1 trybu MODE1 (BR0) musi zostac ustawiony.
  - Bit 0 trybu MODE1 (BR8) musi bys ustawiony.

- Instrukcja odwracania bitów:

  -Instrukcje BITREV odwracaja bity rejestrów I0-I15.
  -Wyjscie nie jest adresem.

Ustawianie trybów DAG

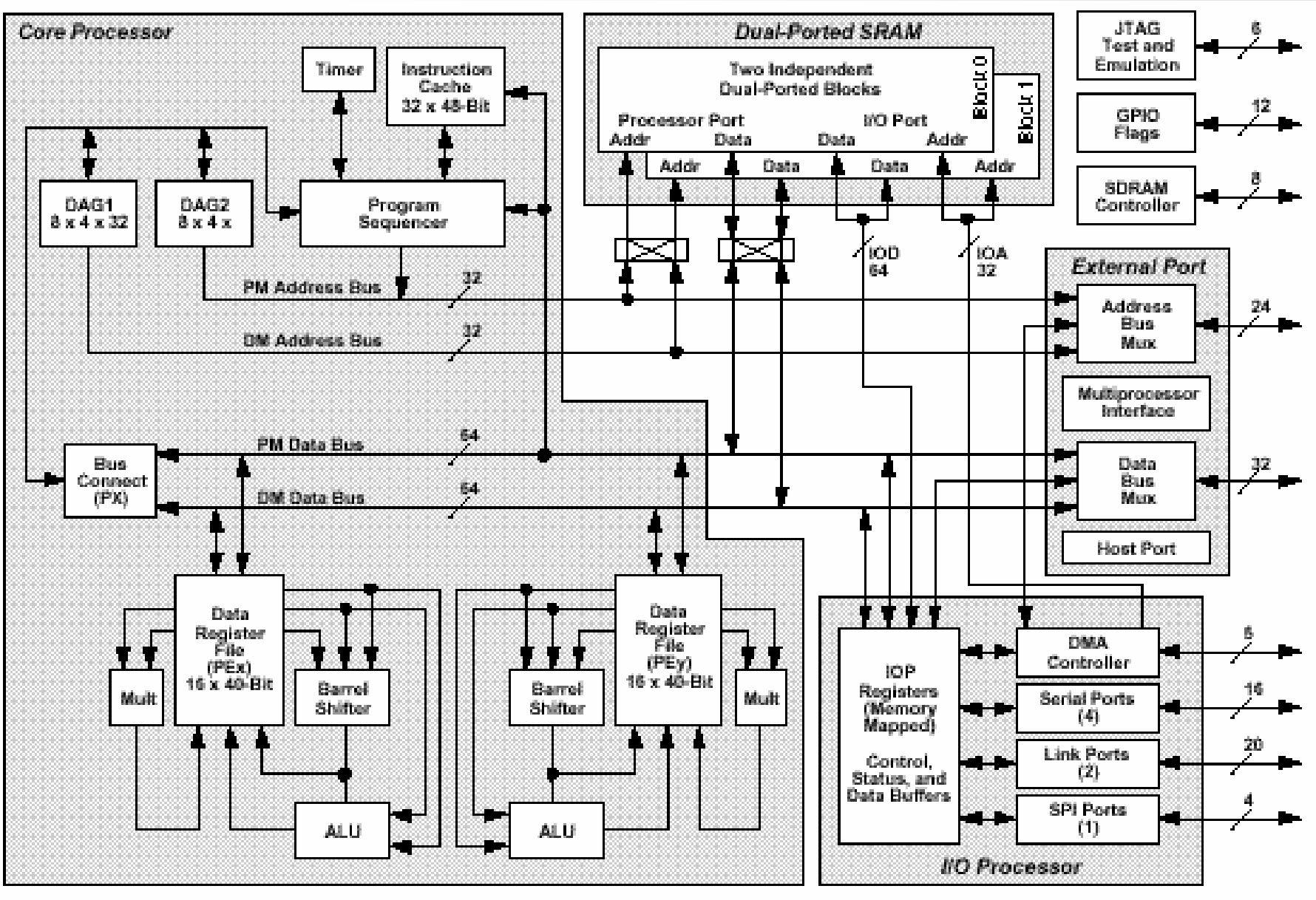


DAG (generator adresów danych)
Cwiczenia i Mini-Quiz

LAB 6

Program Sequencer

ADSP21161 diagram blokowy architektury



Program Sequencer: cechy

•Zachowuje: petle, skoki, podprogramy, przerwania, i instrukcje spoczynkowe

•zachowuje stosy:

- PC Stack: glebokosc=30, podprogramy&adres powrotu z przerwania, top petli Do
- Do Loop Stack: glebokosc=6, koncowy adres, warunek liczenia i zakonczenia
- Status Stack: glebokosc=15, zachowuje ASTATx/y & MODE1 dla IRQ2-0, Timer, i przerwania VIRPT

• 3 poziomowe wykonanie potokowe pozwala na szybkie wykonanie instrukcji:Fetch(pobrania) Decode(dekodowania) / Execute(wykonania)

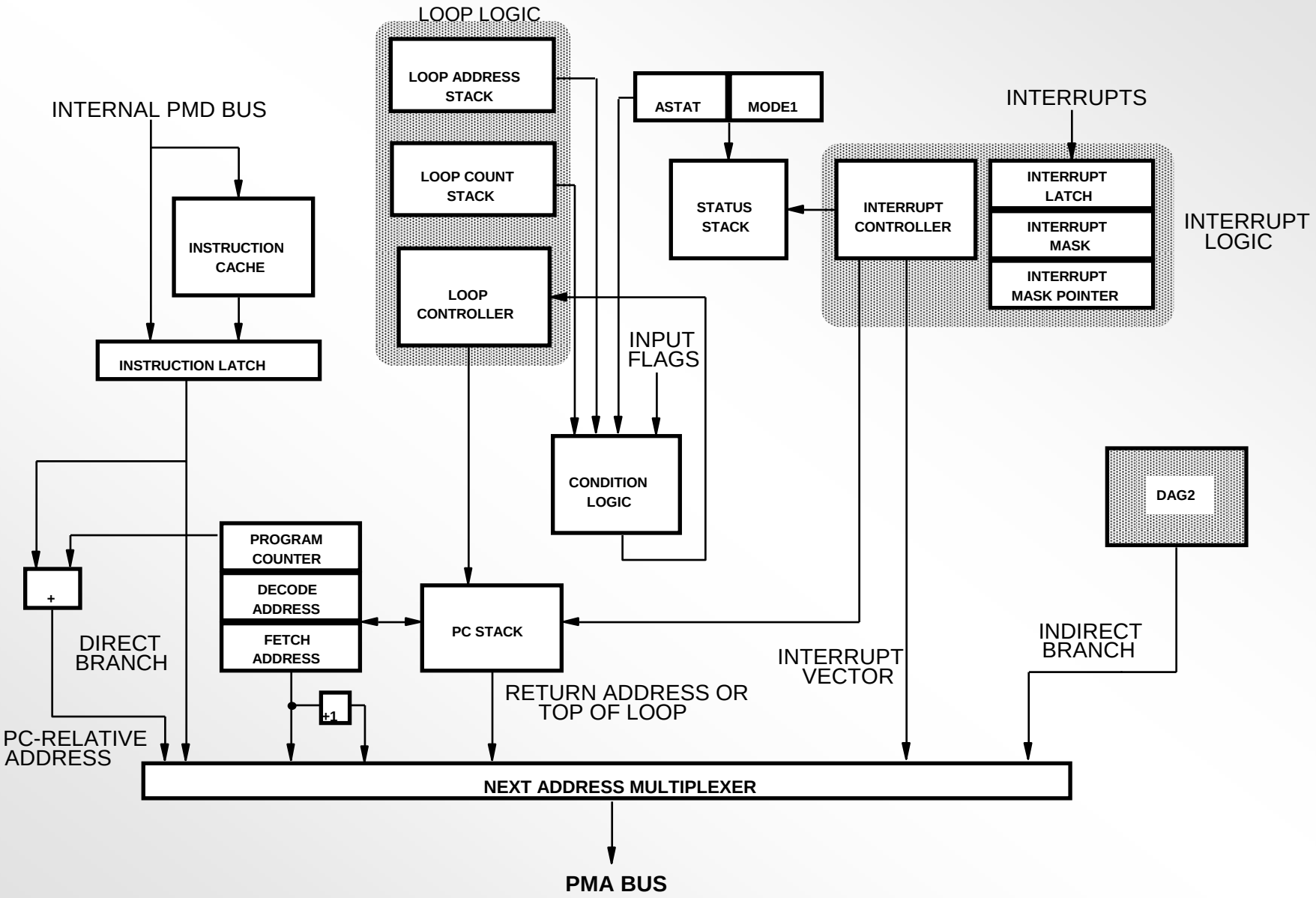
• Pamiec Cache pozwala na trzyprzewodowa odtworzenie z dwoch przestrzeni pamieci

• Pozwala na zrownoleglenie z obliczeniami dla zwiekszenia wydajnosci

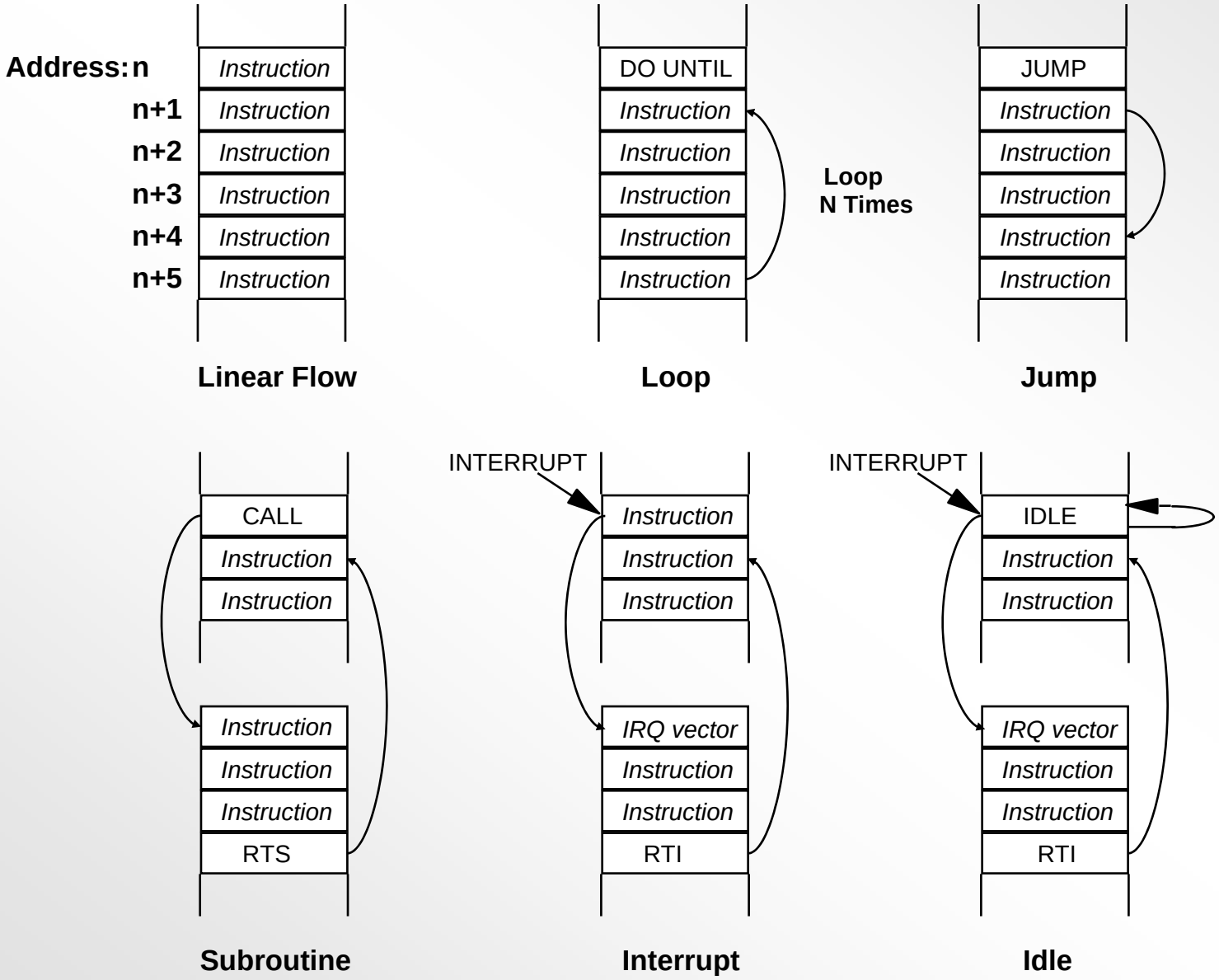
• Zero-Overhead petli Do Until

•opóznione skoki, Call, RTS, RTI

ADSP-21161: Program Sequencer



Zmiany w wykonaniu programu



Przykladowe instrukcje wykonania programu

```
if eq jump label;                          /* bezposredni skok */

if le jump (pc,25);                        /* wzgledny skok */

if ge jump (m8,I9);                        /* posledni skok */

if lt call rfft (DB);                      /* delayed call */
  i0=real_data;
  r0=Npoints;

if ne rts (DB), f0=f1*f2;                  /* delayed return & compute */
  r0=fix f0;
  dm(result)=r0;

lcntr=44, do samples until lce;            /* counter based do loop */
          r0=dm(i0,1);
samples:  dm(i1,1)=r0;

lcntr=23, do (pc,2) until lce;             /* early exit from do loop */
  r0=r0-r2;
  if eq jump outahere (la);
```

Instrukcje kontroli wykonywania programu(21161)

IF warunek JUMP <addr24> | (PC,<reladdr24>) [(DB | LA | CI | DB,LA | DB,CI)];

IF warunek CALL <addr24> | (PC,<reladdr24>) (DB);

IF warunek JUMP (Md,Ic) | (PC,<reladdr6>) [(DB | LA | CI | DB,LA | DB,CI)] , *oblicznie* | *ELSE obliczenie* ;

IF warunek CALL (Md,Ic) | (PC,<reladdr6>) (DB) , *coblicznie* | *ELSE obliczenie* ;

(CI) Clear Interrupt(wyzeruj przerwanie)
(LR) Loop Reentry(ponowne wejscie do petli)
(DB) Delayed Branch
(LA) Loop abort(przerwanie petli) (pop loop and PC stacks)

Instrukcje kontroli wykonania programu (21161) cd.

IF warunek JUMP (Md,Ic) | (PC,<reladdr6>) | , ELSE | *oblicznie, DM(Ia,Mb)=dreg* | *oblicznie , dreg=DM(Ia,Mb)* | ;

IF warunek RTS | (DB | LR | DB,LR) , | *obliczenie* | *ELSE obliczenie* | ;

IF warunek RTI (DB) , | *oblicznie* | *ELSE oblicznie* | ;

LCNTR = | <data16> | ureg | , DO | <addr24> | (PC,<reladdr24>) | UNTIL LCE ;

DO | <addr24> | (PC,<reladdr24>) | UNTIL termination ;

Potok instrukcji(Instruction Pipeline)

•Potok instrukcji pozwala ADSP-21161 operowac na szybszych cyklach
•Te cykle zachodza na siebie lub sa potokowe, daja wiecej czasu na wykonanie

Fetch Cycle(pobrania): procesor odczytuje instrukcje z PM lub Cache

Decode Cycle(dekodowania):instrukcja jest tlumaczona, generuje warunki, które steruja wykonaniem

Execute Cycle(wykonania): operacje sprecyzowane przez instrukcje sa wykonywane

czas (cykle) →

Execute Instruction			0X40005	0X40006	0X40007
Decode Instruction		0X40005	0X40006	0X40007	0X40008
Fetch Instruction	0X40005	0X40006	0X40007	0X40008	0X40009

Po Reset

Nieopoznione Galezie

•W dowolnej chwili gdy cykl wykonania programu jest przerwany przez galaz, licznik operacji musi ponownie rozpoczac dekodowanie i pobrac instrukcje na miejsce przeznacze

•Licznik operacji znosi wykonanie dwóch instrukcji ulokowanych po instrukcji branch (galaz)

•Instrukcja branch potrzebuje 3 cykli do wykonania sie

```
...                 N-1
if lt jump j;       N
f0=dm(i0,m2);       N+1
f1=f0*f2;           N+2
...                 N+3
```

czas (cykle) →

Execute Instruction	jump J N	nop	nop	J	J+1
Decode Instruction	N+1>nop	N+2>nop	J	J+1	J+2
Fetch Instruction	N+2	J	J+1	J+2	J+3

Opoznione Galezie

- Opozniona-Galaz pozwala programiscie na uzycie dwóch instrukcji po branch, dajac bardziej wydajny kod

- Dla warunkowych galezi te dwie instrukcje sa zawsze wykonywane -niezaleznie czy warunek jest spelniony czy nie

- Instrukcje branch potrzebuja nadal 3 cykli do wykonania

```
...                          N-1
if lt jump j (db);           N
  f0=dm(i0,m2);              N+1
  f1=f0*f2;                  N+2
...                          N+3
```

czas (cykle) →

Execute Instruction	jump J (db) N	N+1	N+2	J	J+1
Decode Instruction	N+1	N+2	J	J+1	J+2
Fetch Instruction	N+2	J	J+1	J+2	J+3

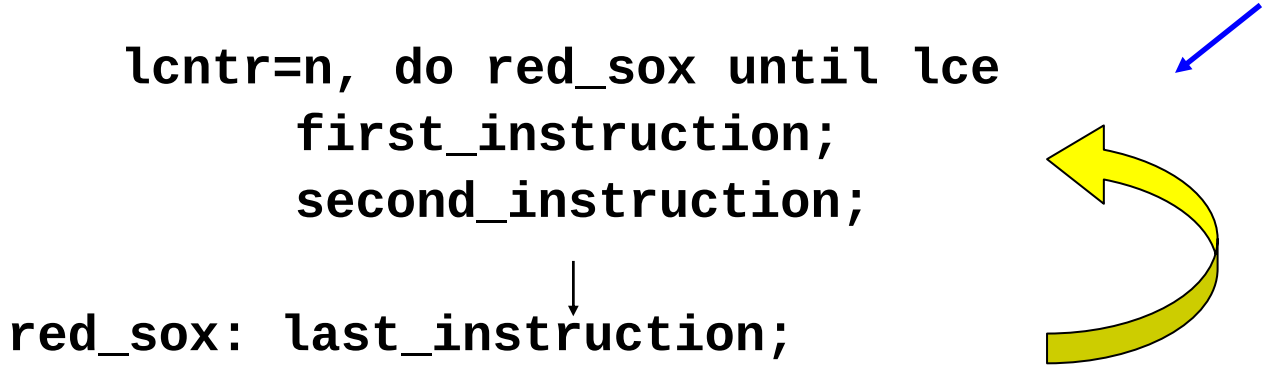
Zero-Overhead Looping

- Zastosowane do wydajnych petli sprzetowych, wylaczajac overhead instrukcji do dekrementacji licznika, testowania warunku, i branch

- Wszystkie dane potrzebne do maintain up to 6 zagniezdzonych petli sa przechowywane na stosach wewnatrz Program Sequencer(programowego licznika operacji)

- Licznik petli(loop counter) (LCNTR) lub warunek arytmetyczny moze zakonczyc petle

licznik petli przepelniony



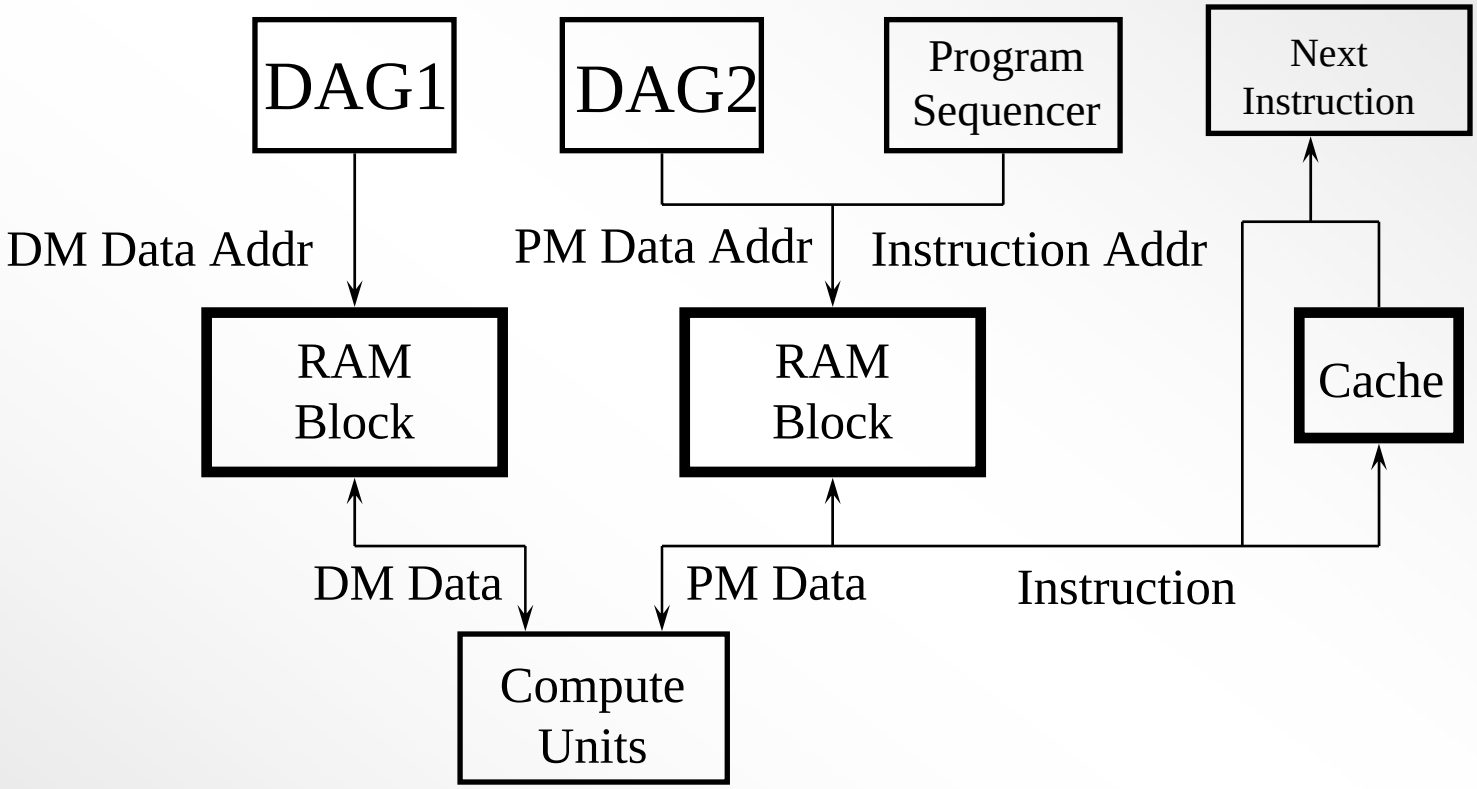
Pamiec podreczna rozkazów(Instruction Cache)

Zwalnia magistrale PM by umozliwic jednocyklowy dostep do danych z przestrzeni PM (DAG2)

- Chowa tylko instrukcje, które koliduja z data accesses na magistrali PM Data Bus
  - -Bardzo wydajne poniewaz nie kazda instrukcja bedzie miala dostep do szyny danychPM
  - -Druga droga, ustawienie wspoldzialajacych z wejsciami dla 32 instrukcji
  - - algorytm aktualizacji Least Recently Used (LRU)
- Przezroczysta dla programisty, uzycie petli Do Until zwieksza wydajnosc kodu
- Jesli instrukcja wykonywana na adresie *n* wymaga dostepu do magistrali PM data bus, zachodzi konflikt z instrukcja pobierana spod adresu n+2.
  - -jezeli instrukcja spod n+2 znajduje sie juz w pamieci podrecznej (a cache hit occurs)
  - -w innym wypadku pojawia sie zanik pamieci cache, i...
    - jeden cykl overhead jest tracony
    - instrukcja pod *n+2* jest ladowana do pamieci podrecznej

adres *n* `f0=f3*f4, PM(I9,m8)=f5;`
adres *n+1* `f0=f0*f5;`
adres *n+2* `f6=f6+f0;`

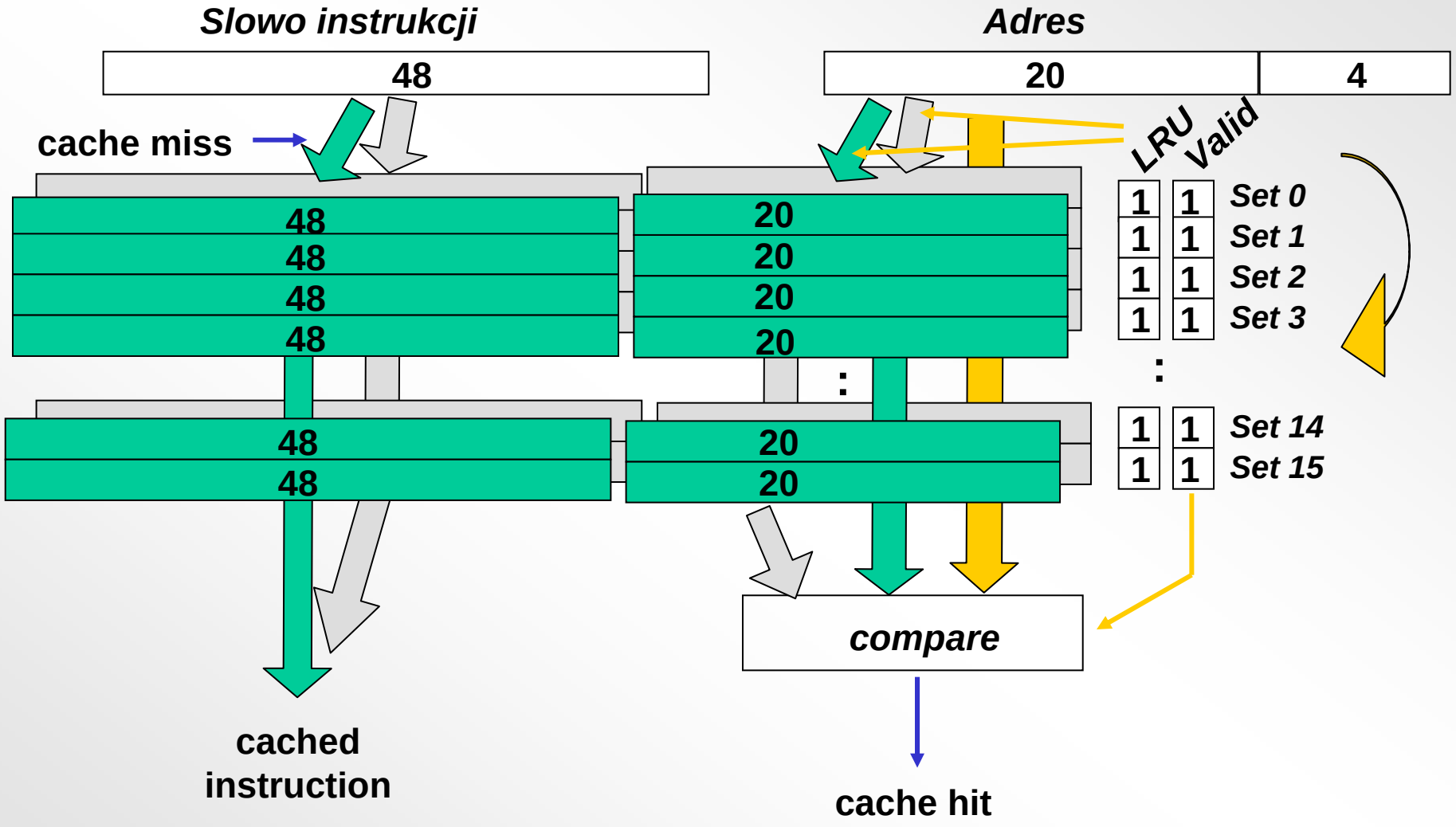
Pamiec podreczna pozwala na 3 sposoby dzialania magistrali



Single Data Transfer: f0=float(r0), r1=DM(i0,m0);

Dual Data Transfer: f0=f8*f12, f4=f0+f4, f8=DM(i0,m0), f12=PM(i8,m8);

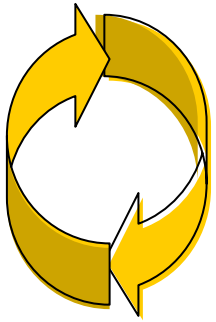
Funkcja pamieci podrecznej



Uwaga: ostatnie 4 bity adresu wybiera ustawienie pamieci podrecznej (way)

Zasmiecanie pamieci(Trashing the Cache)

Unikajcie nastepujacej sytuacji:



```
LOOP
[000300] R2 = PM(I9,M10);

[000310] R3 = PM(I10,M10);

[000320] PM(I8,M9) = R1;
```

- **Instrukcja pod 0x303, 0x313 i 0x323 konkuruja o ten sam ustawienia wejsciowe cache'u**
- **Rozwiazanie: przesuniecie instrukcji o jedna lokacje**

Obslugiwanie przerwan

(Interrupt Handling)

- **Przerwania moga byc generowane przez:**
  - zewnetrzne sygnaly przerwan: IRQ2-0
  - timer
  - przepelnienie bufora kolowego (Wrap) dla I7 i I15
  - przepelnienie stosu statusu lubpetli
  - zapelnienie stosu PC
  - przepelnienie, niedopelnienie lub bledny wynik obliczen
  - Vector Interrupt: VIRPT
  - Link Port Service Request: LSRQ
  - przerwania programowe uzytkownika : SFT3-0
  - 14 kanalów DMA
  - wykrycie nielegalnego warunku wejsciowego
- **IOP Register Access**
- **Unaligned 64-Bit Memory Access (for LW option)**

Obslugiwanie przerwan

- przerwania zewnetrzne moga reagowac na poziom lub zbocze, ustawianie w MODE2
- przerwania maja priorytet i moga byc maskowane w IMASK, LIRPTL
- bit Global Interrupt Enable w trybie MODE1
- tablica wektorów przerwan ma poczatek pod 0x00040000*
- Wektorowanie trwa min 3 cykle overhead (3 NOPy sa wykonywane)

**** no-boot mode: extern 0x200000***

Tablica wektorów przerwan i priorytety

Nazwa	Adres	Interrupt Vector Address Function
reserved	0x00	Reserved interrupt(zarezerwowane przerwanie)
RSTI	0x04	Reset vector address (niejawne IDLE na 0x40004)
IICD	0x08	Illegal input condition detected(nielegalny warunek wejsciowy wykryty)
SOVFI	0x0C	Status stack/loop stack overflow or PC stack full:
TMZHI	0x10	High priority timer interrupt(przerwanie timera o wysokim priorytecie)
VIRPTI	0x14	Vector Interrupt
IRQ2I	0x18	Hardware Interrupt 2 (IRQ2)(przerwanie sprzetowe)
IRQ1I	0x1C	Hardware Interrupt 1 (IRQ1)
IRQ0I	0x20	Hardware Interrupt 0 (IRQ0)
reserved	0x24	Reserved interrupt(zarezerwowane przerwanie)
SP0I	0x28	Serial port 0 channel A/B RX/TX buffers (DMA Channel 0 & 1)
SP1I	0x2C	Serial port 1 primary A, secondary B RX/TX buffers (DMA Channel 2 & 3)
SP2I	0x30	Serial port 2 primary A, secondary B RX/TX buffers (DMA Channel 4 & 5)
SP3I	0x34	Serial port 3 primary A, secondary B RX/TX buffers (DMA Channel 6 & 7)
LP0I	0x38	Link Buffer 0 (DMA Channel 8)
LP1I	0x3C	Link Buffer 1 (DMA Channel 9)
SPIRI	0x40	SPI Receive (DMA Channel 8)
SPITI	0x44	SPI Transmit (DMA Channel 9)

WIEKSZY PRIORYTET

Tablica wektorów przerwan i priorytety

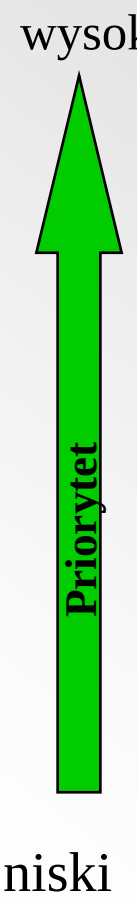
Nazwa	Adres	Interrupt Vector Address Function
reserved	0x48	Reserved Interrupt (zarezerwowane przerwanie)
reserved	0x4C	Reserved Interrupt
EP0I	0x50	External Port Buffer 0 (DMA Channel 10)
EP1I	0x54	External Port Buffer 0 (DMA Channel 11)
EP2I	0x58	External Port Buffer 0 (DMA Channel 12)
EP3I	0x5C	External Port Buffer 0 (DMA Channel 13)
LSRQI	0x60	Link service request
CB7I	0x64	DAG1 buffer 7 circular buffer overflow
CB15I	0x68	DAG2 buffer 15 circular buffer overflow
TMZLI	0x6C	Lower priority timer interrupt(przerwanie timera o mniejszym priorytecie)
FIXI	0x70	Fixed-point overflow interrupt(przerwanie/na/przepelnienie/staloprzecinkowe)
FLTOI	0x74	Floating-point overflow exception interrupt(zmienn. przecinkowe przepelnienia
FLTUI	0x78	Floating-point underflow exception interrupt(zmienn.przecinkowe niedomiaru)
FLTII	0x7C	Floating-point invalid exception interrupt(zmienn.przecinkowe nieprawid. wyjatku)
SFT0I	0x80	User software interrupt 0(przerwanie programoweuzytkownika)
SFT1I	0x84	User software interrupt 1
SFT2I	0x88	User software interrupt 2
SFT3I	0x8C	User software interrupt 3
reserved	0x90	Reserved Interrupt

MNIEJSZY **PRIORYTET**

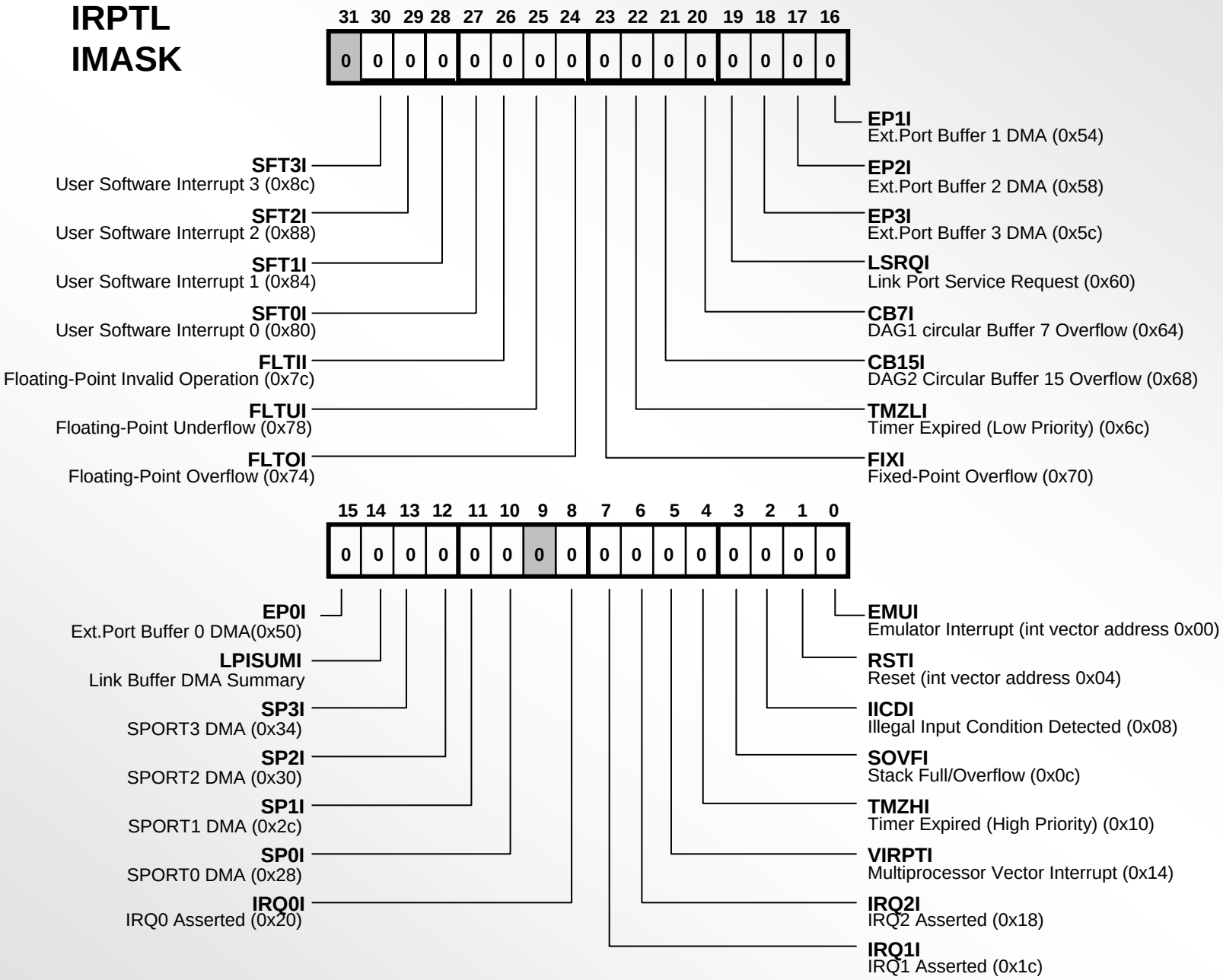
Przyklad wektora przerwan

po Reset

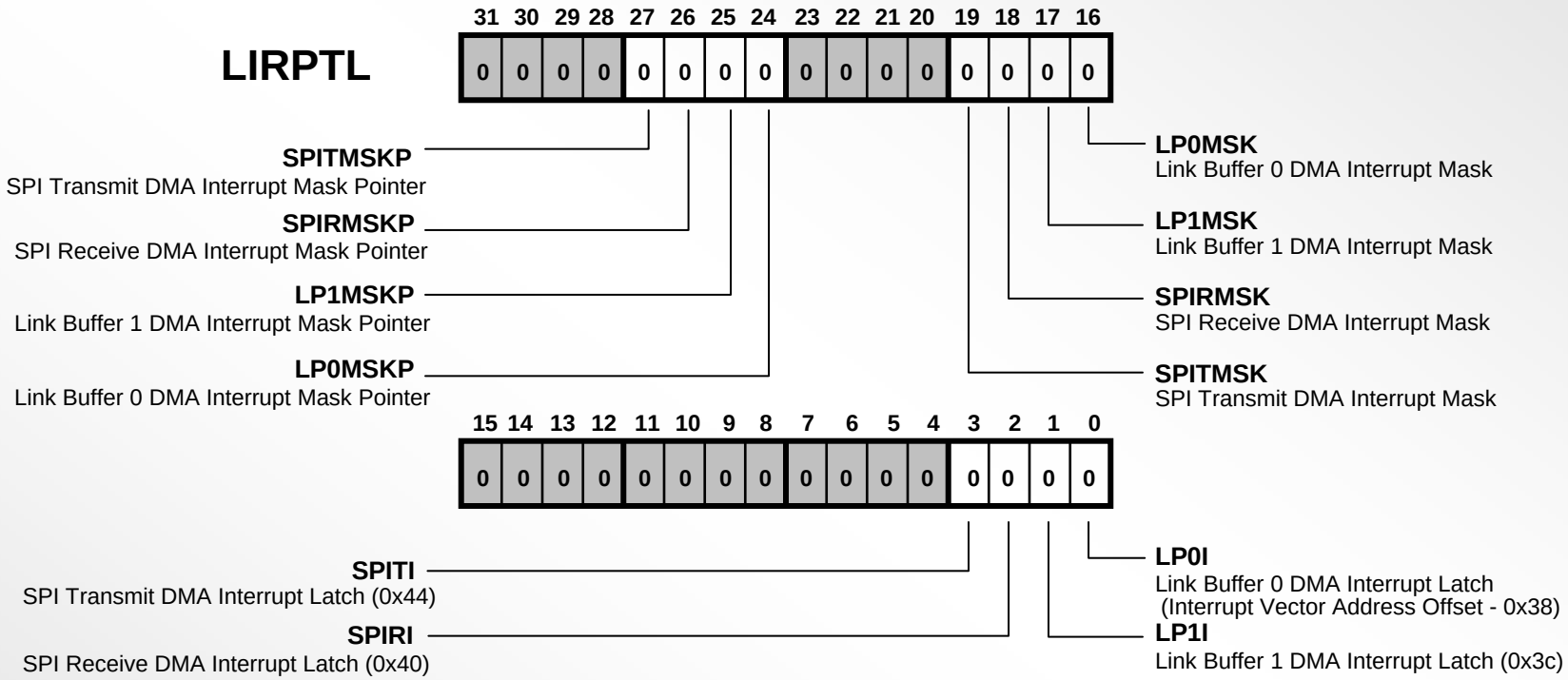
```
IVT_Start: nop; nop; nop; nop;                  /*  reserved   */
RSTI:  nop; JUMP start;rti;rti;                 /* RSTI reset vector */
       nop; nop; nop; nop;                      /*  reserved   */
SOVFI: rti; rti; rti; rti;                      /* SOVFI stack overflow */
TMZHI: rti; rti; rti; rti;                      /* TMZHI  high timer */
       rti; rti; rti; rti;                      /* VIRPTI vector interrupt */

       bit tgl ASTAT FLG0;rti;rti;rti;          /* IRQ2I  irq2 vector */
       rti; rti; rti; rti;                      /* IRQ1I  irq1 vector */
       rti; rti; rti; rti;                      /* IRQ0I  irq0 vector */
       nop; nop; nop; nop;                      /*  reserved   */

       jump ISR_Receive;rti;rti;rti;/* SP0I  DMA0/1 SP0 */
       rti; rti; rti; rti;                      /* SP1I  DMA2/3 SP1 */
       rti; rti; rti; rti;                      /* SP2I  DMA4/5 SP2 */
       rti; rti; rti; rti;                      /* SP3I  DMA6/7 SP3 */


ISR_Receive:     r1=dm(RX0A);     /*get input samples*/
                 dm(i0,m0)=r1;
                 r2=dm(i2,m2);
                 r3=r5 xor r4;
                 dm(i3,m4)=r3;
                 rti;
```



IRPTL i IMASK



LIRPTL


LIRPTL
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SPITMSKP
SPI Transmit DMA Interrupt Mask Pointer
SPIRMSKP
SPI Receive DMA Interrupt Mask Pointer
LP1MSKP
Link Buffer 1 DMA Interrupt Mask Pointer
LP0MSKP
Link Buffer 0 DMA Interrupt Mask Pointer
LP0MSK
Link Buffer 0 DMA Interrupt Mask
LP1MSK
Link Buffer 1 DMA Interrupt Mask
SPIRMSK
SPI Receive DMA Interrupt Mask
SPITMSK
SPI Transmit DMA Interrupt Mask
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SPITI
SPI Transmit DMA Interrupt Latch (0x44)
SPIRI
SPI Receive DMA Interrupt Latch (0x40)
LP0I
Link Buffer 0 DMA Interrupt Latch
(Interrupt Vector Address Offset - 0x38)
LP1I
Link Buffer 1 DMA Interrupt Latch (0x3c)

Stos statusu(Status Stack)

- **Zachowuje ASTAT i MODE1 tylko dla nastepujacych instrukcji:**
  - przerwanie zewnetrzne IRQ0, IRQ1, IRQ2
  - przerwanie przez timer
  - przerwanie VIRPT
- **Zaimplementowane w sprzecie, 15 poziomów glebokosci**
- **Inne przerwania moga uzyc jawnych instrukcji**

```
irq:   push STS; // saves MODE1 and ASTAT
       . . .
       rti (db);
         pop STS;
         nop;
```

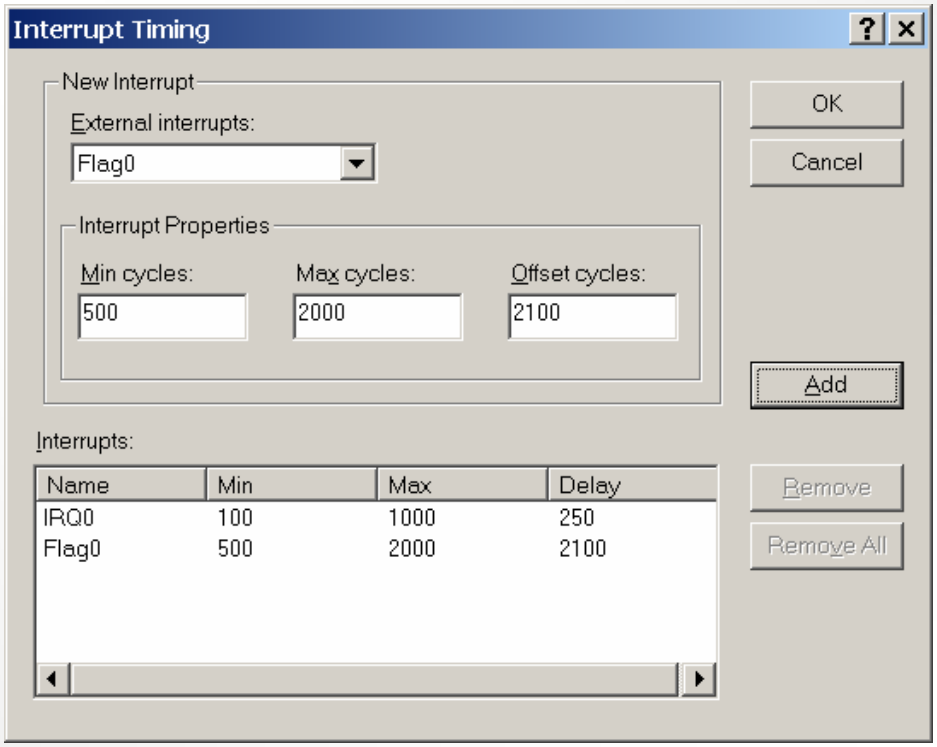
- bity **FLAG3-0 w ASTAT nie dzialaja**

Zwloka przerwania

(Interrupt Latency)

- **Minimalna zwloka przerwania**
  - **Synchronizacja i zatrzaskiwanie (1 cykl)**
  - **Rozpoznanie (1 cykl)**
  - **Rozgalezienie do wektora przerwan (2 cykle)**
- **Dodatkowe opoznienia**
  - **Brak pamieci Cache (1 cykl)**
  - **Opoznienie galezi (max. 2 cykle)**
  - **Ostatniej iteracji 1/2och instrukcji petli 'do until' (max. 3 cykle)**
  - **Pending off-chip accesses(opoznione dostepy do chipu)**
- **Skok z tablicy wektorów przerwan do programu obslugi**
  - **nieopoznione (3 cykle)**
  - **opoznione (1 cykl)**

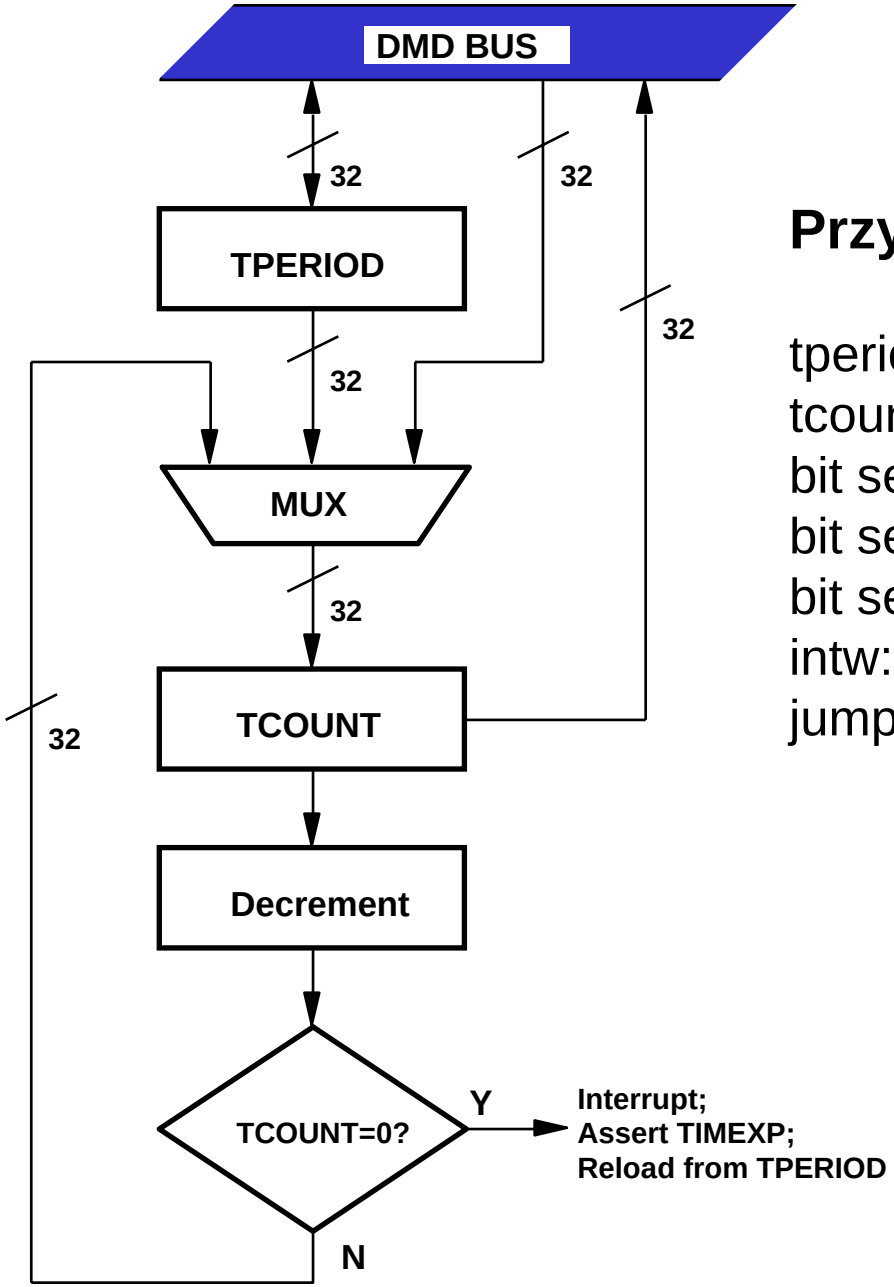
Symulowanie przerwan



Timer

- Generuje okresowe przerwania,
  - –zapewnia TIMEXP pin przez 4 cykle
  - –przerwania sa oblugiwane wtedy gdy sa globalnie zezwolone i niezamaskowane
- Timer zlicza wymagana liczbe cykli rozkazowych
- Timer jest uaktywniany przez ustawienie bitu w rejestrze MODE2 (rozpoczyna zliczanie przez timer)
- 2 32 bitowe rejestry systemowe kontroluja timer
  - TCOUNT: zawiera liczbe cykli rozkazowych do zliczenia. TCOUNT dekrementuje sie w kazdym cyklu rozkazowym.
  - TPERIOD: zawiera wartosc, która bedzie automatycznie zaladowana do TCOUNT kiedy odliczanie uplynie (TPERIOD + 1=okres timera)
- Kiedy TCOUNT=0 zgloszenie przerwania jest generowane i TIMEXP jest ustawiony w poziom wysoki przez 4 cykle rozkazowe
- TCOUNT i TPERIOD moga zostac odczytane/zapisane w dowolnym czasie

Timer



Przyklad kodu ustawiania timera:

```
tperiod = N-1;              /* #cykli w przedziale */
tcount =  tperiod;          /*init tcount    */
bit set imask 0x10;         /* odmaskow. przerwania *
bit set mode1 0x1000;  /* zezw. na przerwania */
bit set mode2   0x20;     /* uaktywnienie timera  */
intw:  idle;      /* instrukcja, oczekiwanie na przerw.*
jump  intw;     /* adres powrotu z przerwania  */
```

Programowalne flagi I/O

- **12 pinów I/O ogólnego przeznaczenia**
- **Programowalne dla wejscia lub wyjscia**
- **Wytwarzane wyjscie i probkowanie oparte na PLLICLK (wyjscie z CLKIN~ i CLKDBL~)**
- **Flagi 0-3**
  - Control =>Mode2
  - Status => ASTAT
  - pozwala na galezie warunkowe
- **Flagi 4-11 (IOFLAG – IOP register)**
  - Control => bity 8 -15 rejestru IOFLAG
  - Status => bity 0 - 7 rejestru IOFLAG
  - nie ma warunkowych galezi

SREG: Rejestry trybu

MODE1

- **Chwyta czesto zmieniane informacje o konfiguracji chipu**
  - 40/32 bitowy tryb zmiennoprzecinkowy
  - tryb Short word sign extend
  - tryb nasycenia ALU(ALU saturation mode)
  - tryb odwracania bitow(Bit reverse mode)
  - uaktywnienie podstawowego/alternatywnego rejestru
  - tryb Round lub Truncate(obcinania)
  - uaktywnienie Global interrupt
  - uaktywnienie zagniezdzania przerwan (Interrupt nesting enable)
  - uaktywnienie Global Circular Buffer
  - uaktywnienie SIMD

- **Automatycznie zachowywany na stosie statusu (obok ASTAT) przez przerwania przyciskami IRQ2-0, timer, i VIRPT, i jest odtworzony podczas powrotu**

MODE2

- **Chwyta nieczesto zmieniane informacje o konfiguracji chipu**
  - wrazliwosc przerwania na zbocze lub poziom(Interrupt edge or level sensitivity)
  - kierunek flag I/O
  - uaktywnienie timera(Timer enable)
  - uaktywnienie/ dezaktywacje/zawieszone dzialanie pamieci chache(Cache enable/disable/freeze)
- **Nie jest zachowywany w trakcie przerwan**

Tryb maskowania rejestru

Mode Mask Register

- **Kazdy bit w MMASK odpowiada bitowi w MODE1.**

- **Podczas wchodzenie w procedure obslugi przerwania lub uzycia instrukcji “push sts”, DSP umieszcza zawartosc MODE1 na stosie statusu i laduje MODE1 z wartoscia MMASK**

Uwaga: MMASK wylacza SIMD jako wartosc domyslna

Manipulacja Bitu Rejestru Systemu SREG

Skladnia:

BIT | SET / CLR / TGL / TST / XOR | sreg <data32>;

Przyklad:

```
bit set mode1 0x12000;
bit set mode1 RND32 | ALUSAT;
```
Sets 32-bit RND & ALU Saturation

```
bit tst ustat1 0x5;
if tf jump label;
```
Testuje bit 0 i bit 2 i skacze jesli prawda

```
bit clr ustat1 0x4;
```
kasuje bit 2 ustat1

- Jesli instrkcja **bit tst** lub **bit xor jest prawdziwa**, flaga BTF jest ustawiania w rejestrze ASTAT .

Warunek TF jest oparty na fladze BTF w rejestrze ASTAT .

- Dziala tylko na rejestrach systemowych i operuje na kazdym lub wszystkich 32 bitach równolegle.