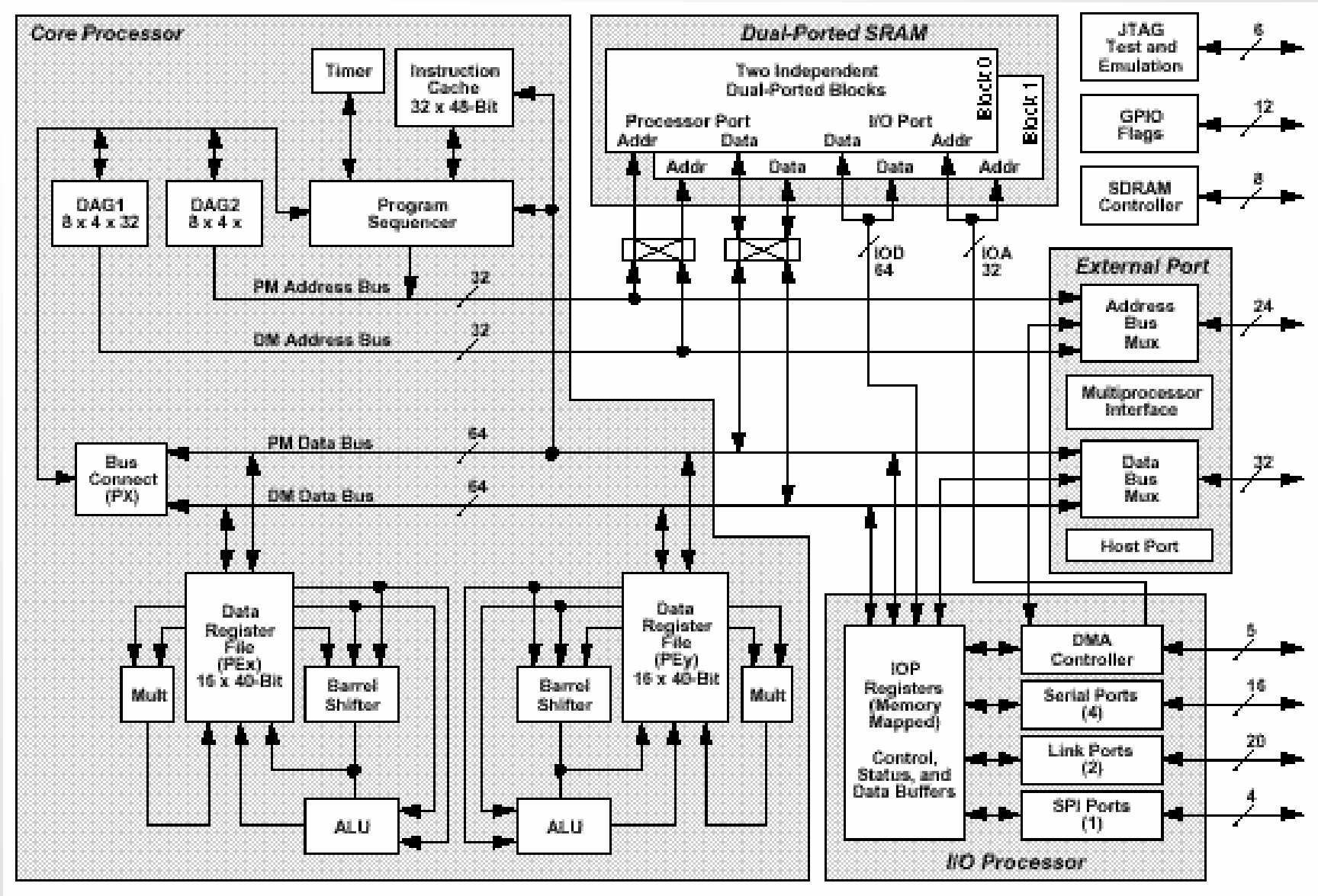
ADSP-21161 Architektura jadra I

Sekcja 3

ADSP-21161 Diagram blokowy architektury



Rejestry i typy danych

ADSP-21161: Grupy Rejestrów

Universal Registers (UREG)
Complementary Register (CREG)

- **System Registers (SREG)**
- **Register File (DREG)**
- **Data Address Generator(DAG)**
- **Bus Exchange (Px)**
- **Timer**
- **Program Sequencer**

Computation Unit Registers

- **Multiplier Results - MR**

I/O Processor Registers
(memory mapped)

- **System Control Registers (SC)**
- **DMA Address Registers (DA)**
- **DMA buffer Registers (DB)**
- **Link & Serial Port (LSP)**
- **SPI Port**

Universal Registers : Przyklady

- **Universal Registers sa dostepne dla innych rejestrów uniwersalnych jak również dla pamieci danych**

Lokalizacja	Rejestr(y)	Funkcja
DREG	R15 - R0	Rejestry danych w register file (PEx)
CUREG of DREG	S15 - S0	Complimentary data registers (PEy)
Program Sequencer	PC	Licznik rozkazów *(tylko do odczytu)*
Address Generators	I7 - I0	indeksowe rejestry DAG1
Timer	TPERIOD	okres timera
System Registers (SREG)	MODE1	Mode control & status
	USTAT1	User status register 1
	USTAT4	User status register 4

Rejestry I/O procesora

- **Rejestry mapy pamieci(Memory mapped registers)**
- **Efekt opoznienia cyklu podczas zapisywania rejestrów IOP**
- **Dostep do adresów pamieci uzywajac :**
  - rdzenia DSP (DSP core)
    - szyna PM (PM bus)
    - szynaDM (DM bus)
  - Urzadzen peryferyjnych
    - szyna I/O (I/O bus)

UREG-CUREG: rejestry komplementarne

Pewne rejestry wewnatrz UREG maja rejestry komplementarne. Rejestry komplementarne (CREG) sa rejestrami w PEy i sa uzywane w trybie SIMD.

Przyklad:

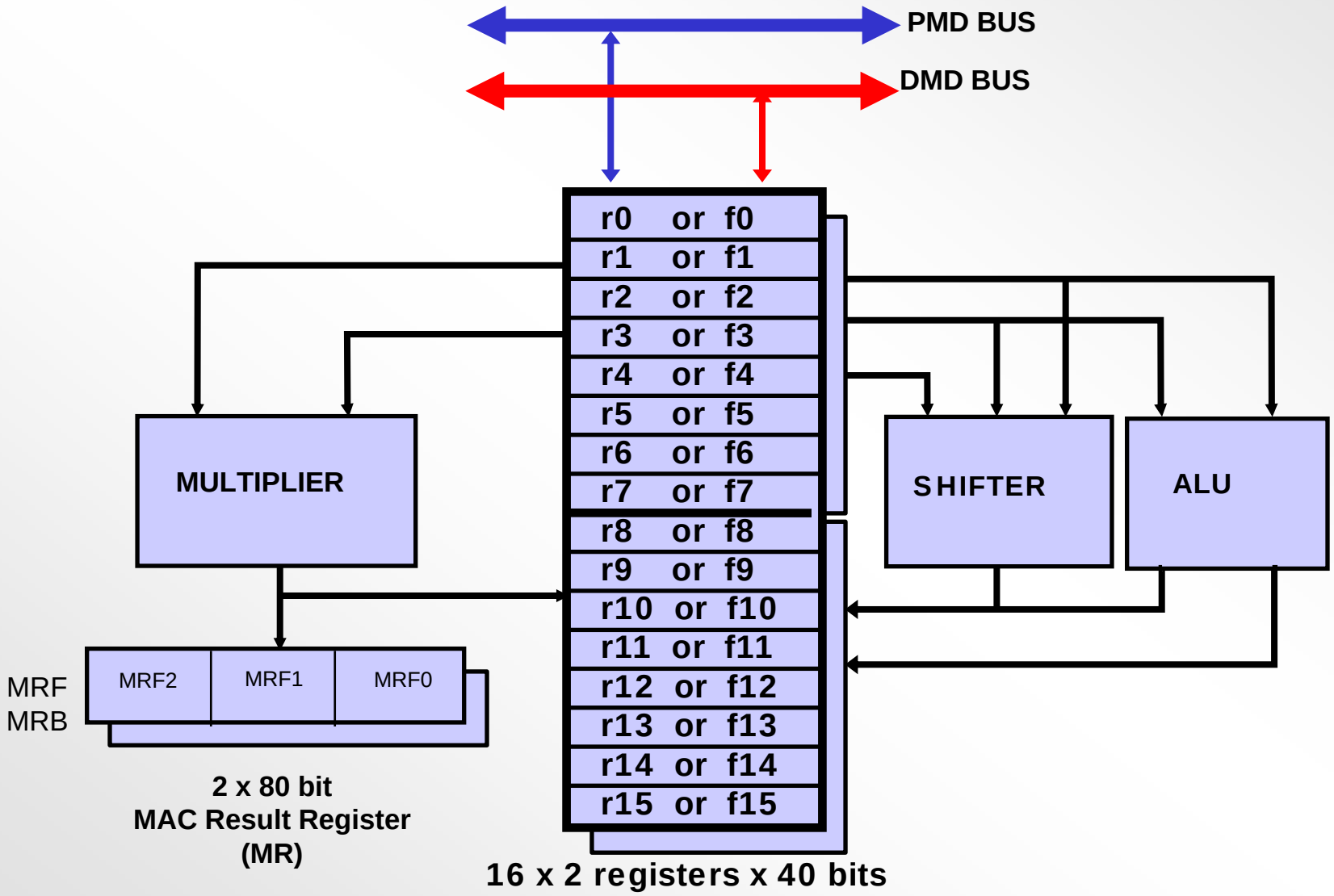
- **DREG ma rejestry komplementarne w PEy**
  - PEx → PEy
- **SREG ma rejestry CREG w PEy**
  - USTAT1 → USTAT2
  - USTAT3 → USTAT4
  - ASTATx → ASTATy
  - STKYx → STKYy

Plik Rejestru Danych: cechy

- **Jeden plik rejestru dla kazdego elementu przetwarzania**
- **Do przenoszenia danych pomiedzy jednostkami obliczeniowymi i pamieci i do przechowywania posrednich wyników.**
- **Swobodny przeplyw danych pomiedzy jednostkami obliczeniowymi i pamiecia.**
- **Przenoszenie 9 slów na cykl:**
  - 2 przeslania do pamieci;
  - 4 odczyty rejestrów, 3 zapisy do rejestrow z jednostkami obliczeniowymi
- **16 rejestrów x 40 bitowej szerokosci**

  –Wsparcie dla przechowywania liczb 40 bitowych zmiennoprzecinkowych o rozszerzonej precyzji, 32 bitowych zmienno- i stalo przecinkowych o normalnej precyzji
- **16 dodatkowych rejestrow pomocniczych Secondary Registers (dla szybkiego przelaczania kontekstów, dostepne przez odpowiedni zapis do rejestru MODE1**

ADSP-1161 Plik Rejestru



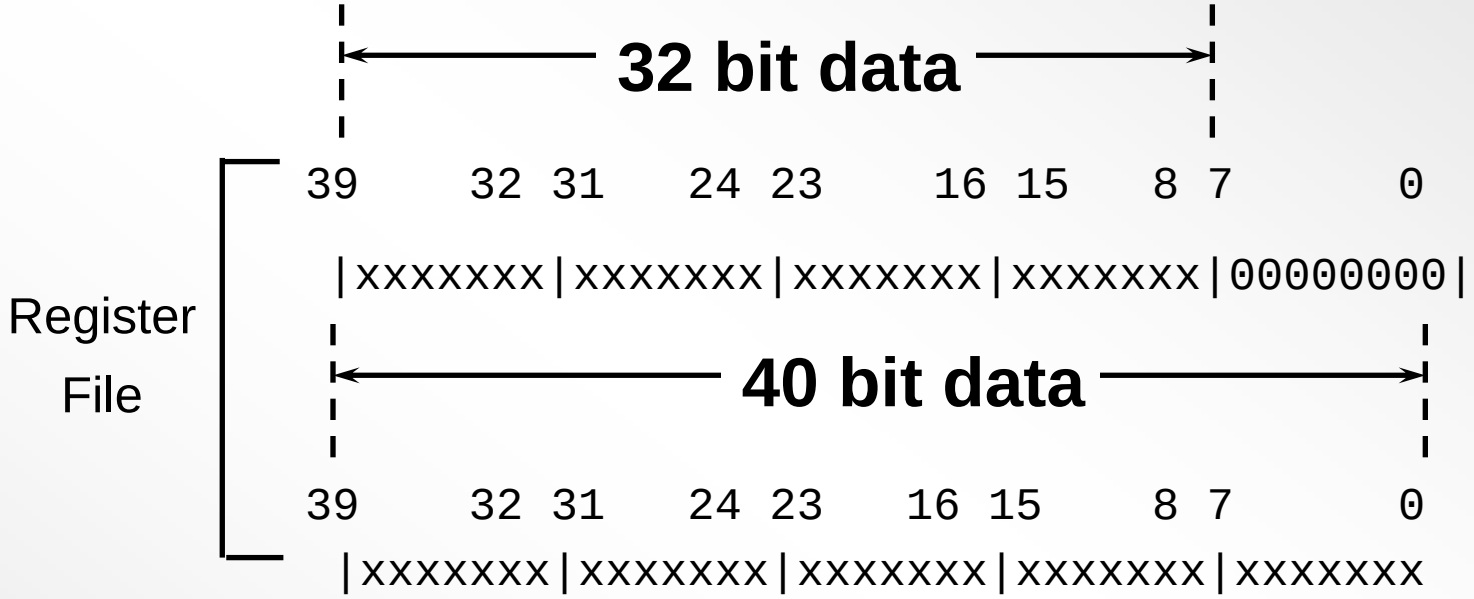
Typy danych

- **zmiennoprzecinkowe: “f” numer rejestru (f0 - f15)**
  - 32-bitowe o pojedynczej precyzji (wybrane przez bit RND32 w MODE1)
  - 40-bitowe rozszerzone o pojedynczej precyzji (domyslne po reset)

- **staloprzecinkowe: “r” numer rejestru (r0 - r15)**
  - 32-bitowa liczba calkowita, ze znakiem i bez znaku
  - 32-bitowe ulamkowe , ze znakiem i bez znaku
  - dla 64 bitowych rezultatów MPY mnozenia/gromadzenia, 80-bitowych akumulacji

- **PEy wykona te same instrukcje co PEx**
  - Operacje zmiennoprzecinkowe w PEx beda mialy rezultat w operacjach zmienno-przecinkowych w PEy
  - Rejestry Data Registers w PEy moga byc dostepne jako “s” rejestry (s0-s15)
    - Uzywane tylko do przenoszenia danych

Wyrównanie danych dla rejestrów danych



32 bitowy format staloprzecinkowy

przyklady

Bit	31	30	29	...	2	1	0
Weight	-2^{31}	2^{30}	2^{29}	...	2^{2}	2^{1}	2^{0}

Sign bit — Liczba calkowita ze znakiem

-3 = 0xfffffffd

3 = 0x00000003

Bit	31	30	29	...	2	1	0
Weight	2^{31}	2^{30}	2^{29}	...	2^{2}	2^{1}	2^{0}

Liczba calkowita bez znaku

2^31 = 0x80000000

3 = 0x00000003

Bit	31	30	29	...	2	1	0
Weight	-2^{0}	2^{-1}	2^{-2}	...	2^{-29}	2^{-30}	2^{-31}

Sign bit — Liczba ulamkowa ze znakiem

.25 = 0x20000000

.75 = 0x60000000

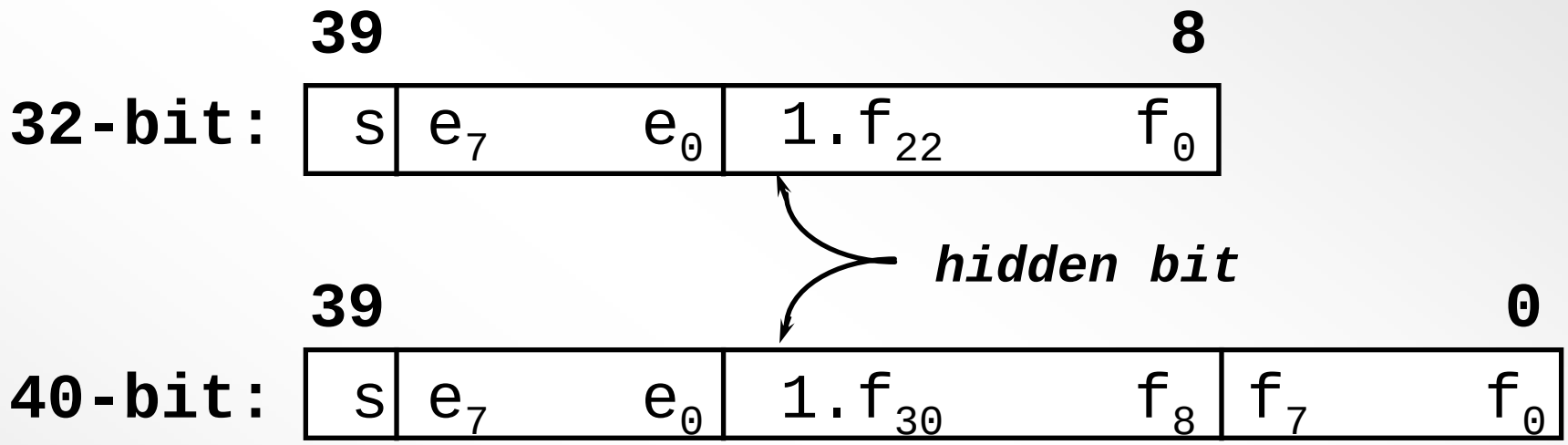
Bit	31	30	29	...	2	1	0
Weight	2^{-1}	2^{-2}	2^{-3}	...	2^{-30}	2^{-31}	2^{-32}

Liczba ulamkowa bez znaku

.25 = 0x40000000

.75 = 0xc0000000

Formaty zmiennoprzecinkowe



typ	exponenta	ulamek	wartosc	przyklad	reprezentacja
Normal	1 ≤e≤254	any	$(-1)^s(1.f)2^{(e-127)}$	0x3f800000	1 . 0
Zero	0	0	zero	0x00000000	0 . 0
Infinity	255	0	infinity	0x7f800000	1.# I N F
NAN	255	non-zero	undefined	0x7f800001	1.# N A N
Denormal	0	any	zero	0x00000001	1.# I N D

Przyklady konwersji zmiennoprzecinkowych

Zamien 0x40400000 na liczbe zmiennoprzecinkowa

Formula: $(-1)^s(1.f)2^{e-127}$

0x40400000=|0|100 0000 0|100 0000 0000 0000 0000 0000

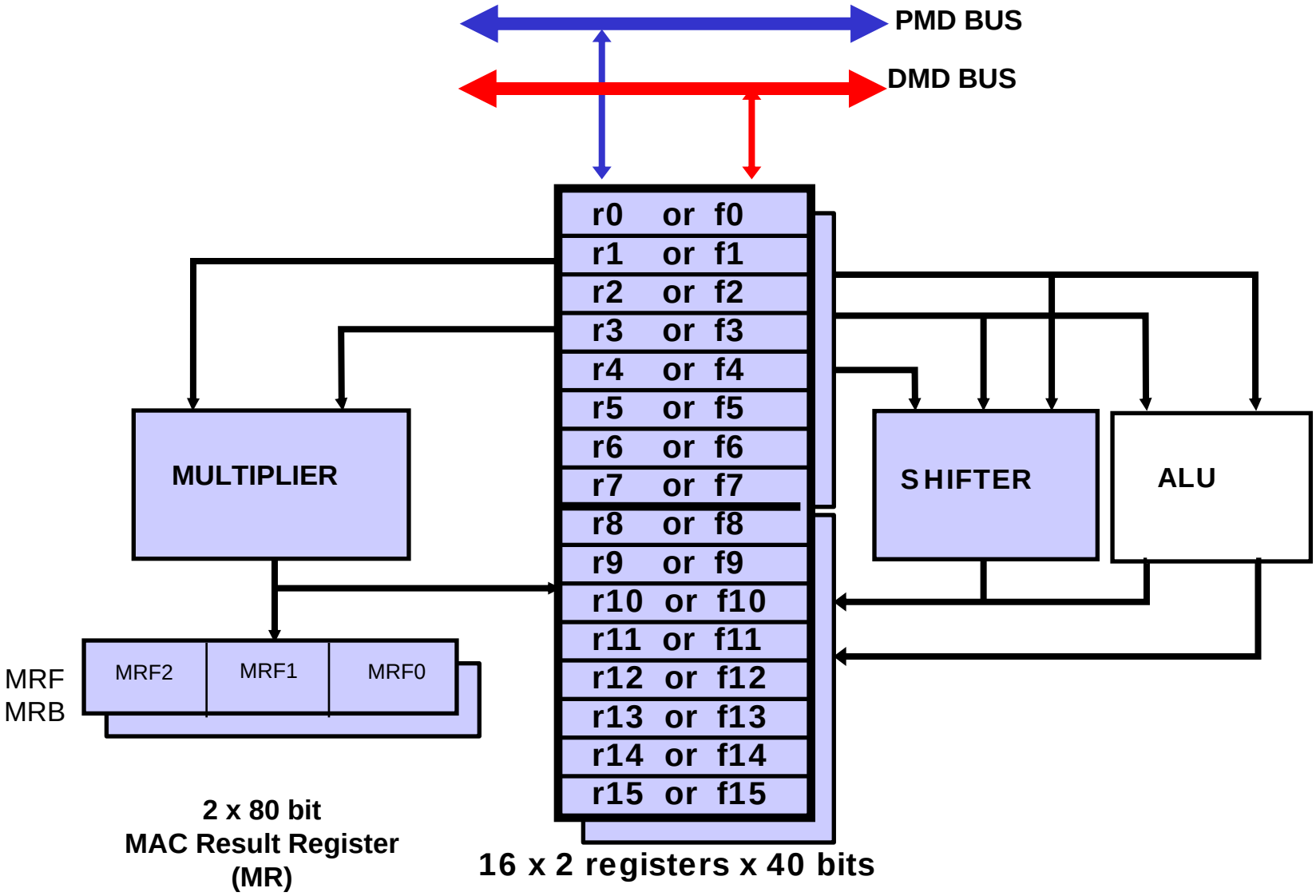
bit znaku=0 exponenta=128 pole ulamkowe =1.5(f=0.5)
(1. is assumed)

$(-1^{0})x(1.5)x(2^{128-127}) = 1.5\ x(2^{1}) = 3.0$

Wprowadznie do Register File i Debugger (programu uruchomieniowego) Cwiczenia

LAB 1

ALU


PMD BUS
DMD BUS
r0 or f0
r1 or f1
r2 or f2
r3 or f3
r4 or f4
r5 or f5
r6 or f6
r7 or f7
r8 or f8
r9 or f9
r10 or f10
r11 or f11
r12 or f12
r13 or f13
r14 or f14
r15 or f15
MULTIPLIER
SHIFTER
ALU
MRF
MRB
MRF2
MRF1
MRF0
2 x 80 bit
MAC Result Register
(MR)
16 x 2 registers x 40 bits

Cechy ALU

- **Wykonuje dzialania na danych zmienno- i staloprzecinkowych**
  - –dodawanie, odejmowanie, jednoczesne dodawanie/odejmowanie, obliczanie sredniej
  - –konwersja formatów (stalo - do zmiennoprzecinkowe . i vice versa)
  - –wartosc bezwzgledna, przebieg, minimum, maximum, obcinanie
  - –porównywanie: 8-bitowy sumaryczny status dla porównan sekwencyjnych

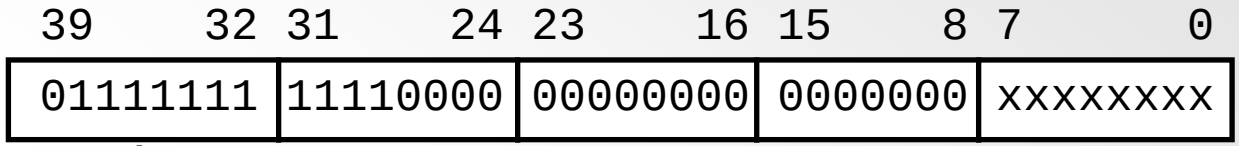
- **Wykonuje operacje specyficzne dla danych staloprzecinkowych**
  - logiczne AND, OR, XOR, NOT;
  - dodawanie staloprzecinkowe z przeniesieniem, odejmowanie z pozyczka, inkrementacje, dekrementacje;
  - tryb nasycenia ALU (ustawainy w rejestrze MODE1);

- **Wykonuje operacje specyficzne dla danych zmiennoprzecinkowych**
  - logika dwójkowa, skalowanie, mantysa;
  - odwrotnosc danej dla dzielenia
  - odwrotnosc pierwiastka kwadratowego z danej dla pierwiastkowania

Przykladowe instrukcje ALU

```
F0 = F1 + F2;                          /* dodawanie Floating Point  */

F7 = F5 + F6, F8 = F5 - F6;  /* podwojne dod./odejmowanie Fltg-Pt. */

F15 = PASS F3;          /* Rejestr xfer do ustawiania flag ALU  */

F14 = MIN(F13,F14);                    /* minimum Floating Point  */

R7 = R6 AND R7;                        /* Bitowa staloprzecinkowa AND */

R8 = FIX F14;                          /* konwersja Float-to-Fixed */

F6 = CLIP F3 BY F4;           /* Fltg-Pt. Clip(obciecie) (saturate F3 by |F4|) */
```

ALU: Przyklad konwersji Fixed-to-Float




```
R0=dm(ADC);
/*R0=0x7ff00000*/Konwersja
```

Konwersja bez skalowania : (32 bitowa l.calkowita do dzisietnej)

```
F2=FLOAT R0;  /*F2=2.146E+9*/
```

Konwersja ze skalowaniem : (32 bitowa l.calkowita do dzisietnej ze skalowaniem)

```
R1=-31;
F2=FLOAT R0 BY R1; /*F2=2.146E+9/(2^31)=0.9995*/
```

Skalowanie wejsciowej wartosci do wartosci pomiedzy: +1.0-lsb to -1.0

Skalowanie wsteczne :

```
R1=31;
R0=FIX F2 BY R1;
dm(DAC)=R0;  /*R0=0x7ff000
```

Instrukcje ALU: Fixed Point(staloprzecinkowe)

```
Rn=Rx+Ry;              Rn=-Rx;               Rn=Rx+CI-1;
Rn=Rx-Ry;              Rn=ABS Rx;            Rn=Rx+1;
Rn=Rx+Ry,Rm=Rx-Ry;     Rn=PASS Rx;           Rn=Rx-1;
Rn=Rx+Ry+CI;           Rn=MIN(Rx,Ry);        Rn=Rx AND Ry;
Rn=Rx-Ry+CI-1;         Rn=MAX(Rx,Ry);        Rn=Rx OR Ry;
Rn=(Rx+Ry)/2;          Rn=CLIP Rx by Ry;     Rn=Rx XOR Ry;
COMP(Rx,Ry);           Rn=Rx+CI;             Rn=NOT Rx;
```

Rn,Rm,Rx,Ry <=> R15-R0; register file location, fixed point

Instrukcje ALU: Floating Point (zmiennoprzecinkowe)

```
Fn=Fx+Fy;                 Fn=ABS Fx;              Rn=LOGB Fx;
Fn=Fx-Fy;                 Fn=PASS Fx;             Rn=FIX Fx BY Ry;
Fn=Fx+Fy,Fm=Fx-Fy;        Fn=MIN(Fx,Fy);          Rn=FIX Fx;
Fn=ABS(Fx+Fy);            Fn=MAX(Fx,Fy);          Fn=FLOAT Rx BY Ry;
Fn=ABS(Fx-Fy);            Fn=CLIP Fx by Fy;       Fn=FLOAT Rx;
Fn=(Fx+Fy)/2;             Fn=RND Fx;              Rn=RECIPS Fx;
COMP(Fx,Fy);              Fn=SCALB Fx BY Ry;      Fn=RSQRTS Fx;
Fn=-Fx;                   Rn=MANT Fx;             Fn=Fx COPYSIGN Fy;
```

Rn,Rm,Rx,Ry <=> R15-R0; register file location, fixed point
Fn,Fm,Fx,Fy <=> F15-F0; register file location, floating point

Flagi statusu ALU

ASTATx/y

Bit	*Nazwa*	*Definicja*
0	AZ	ALU wynik zero lub niedomiar zmiennoprzecinkowy
1	AV	ALU nadmiar
2	AN	ALU wynik ujemny
3	AC	ALU przeniesienie staloprzecinkowe
4	AS	ALU X znak wejscia (operacje ABS i MANT)
5	AI	ALU niewazna operacja zmiennoprzecinkowa
10	AF	ostatnia operacja ALU byla zmiennoprzecinkowa
31-24	CACC	Compare Accumulation register (wyniki 8 ostatnich operacji operacji porownania)

STKYx/y

Bit	*Nazwa*	*Definicja*
0	AUS	ALU niedomiar zmiennoprzecinkowy
1	AVS	ALU nadmiar zmiennoprzecinkowy
2	AOS	ALU staloprzecinkowy nadmiar
5	AIS	ALU niewazna operacja zmiennoprzecinkowa

ALU:Kody warunkowego wykonania

Przyklad: if eq R0 = R1 + R2;

eq	**ALU = zero**
lt	**ALU < zero**
le	**ALU < lub = zero**
ac	**ALU przeniesienie**
av	**ALU przepelnienie**
ne	**ALU nie = zero**
ge	**ALU > lub = zero**
gt	**ALU > zero**
NOT ac	**ALU bez przeniesienia**
NOT av	**ALU bez przepelnienia**

ALU: Przyklad flag statusu/warunków

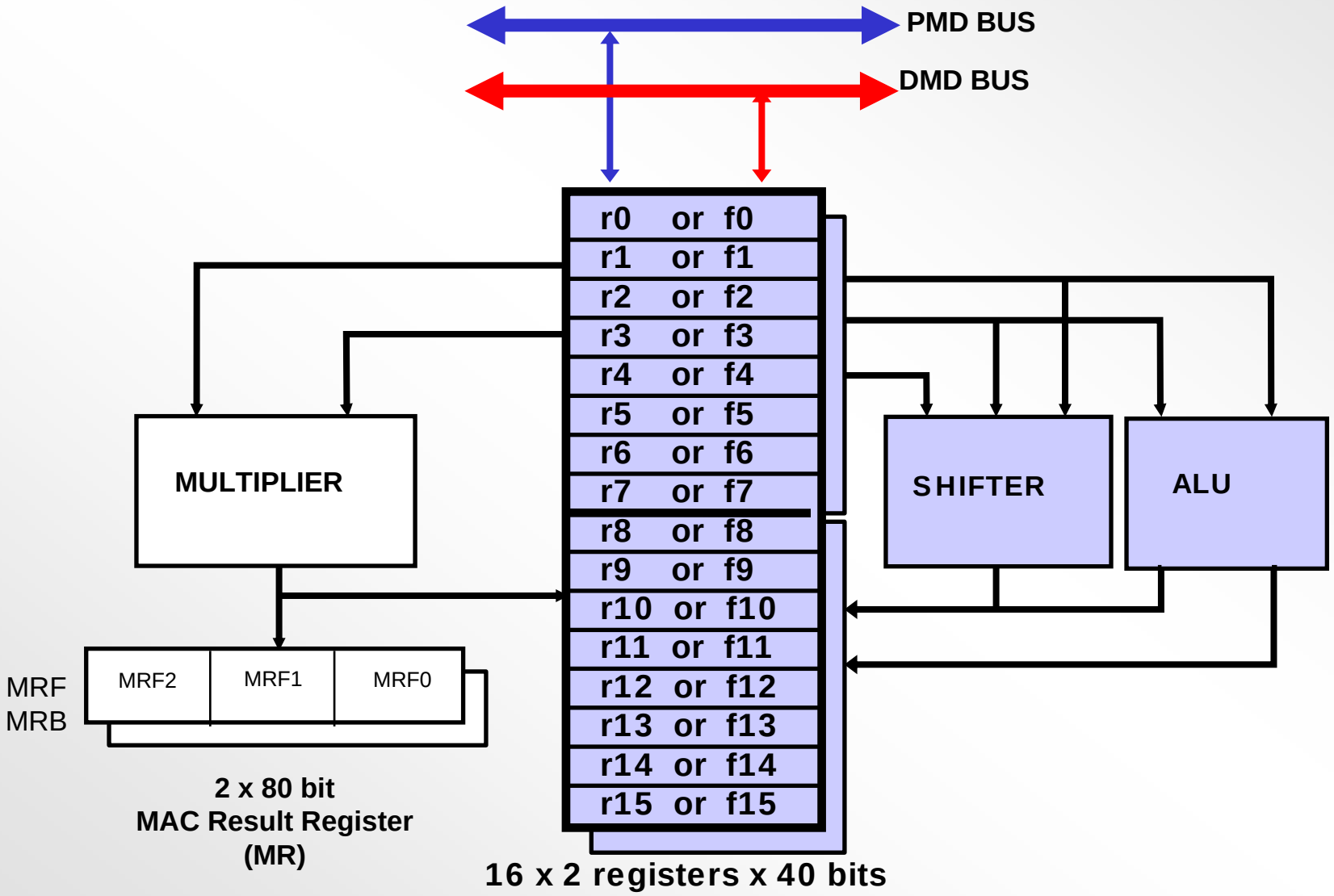
```
R0 = 5;
R1 = 10;
R2 = R0 - R1;
{Flags Set:
    AZ=0, AN=1, AF=0, AV=0}
IF LT JUMP LABEL1;
```

```
F0 = 5.3E3;
F1 = 7.6E4;
F2 = F0 - F1;
{Flags Set:
    AZ=0, AN=1, AF=1, AV=0}
IF LT JUMP LABEL1;
```

ALU: Cwiczenia i mini-quiz

LAB 2

Multiplier / MAC(mnoznik)



Cechy MAC

- **32/40-bitowy zmiennoprzecinkowy mnoznik**
  - MAC uzywany z ALU dla 1-cyklowego potokowego MAC
  - 32/40-bitowe zmiennoprzecinkowe mnozenie
    - 40-bitowe wyniki, opcjonalnie zaokraglone do 32 bitów

- **32-bit staloprzecinkowy MAC**
  - 64-bitowe wyniki, liczby calkowite i ulamkowe
  - 80-bitowe podwójne akumulatory
  - 32-bitowe wyniki (zapamietywane do pliku rejestru)

- **Operacje rejestru Multiplier Result (MR) Register**
  - zaokraglanie, nasycanie, i czyszczenie rejestru z wynikami

MAC: przykladowe instrukcje

```
MRF = R1 * R2;                  // staloprzecinkowe mnozenie, domyslne tryby
R15 = R14 * R13 (SSFR);         // staloprzecinkowe mnozenie, z zaokraglaniem
                                //  wynik, argumenty ulamkowe ze znakiem
MRF = MRF + R7 * R8 (SSF);      // staloprzecinkowe MAC, l.calkowita ze znakiem
                                //  argumenty
MRF = 0;                        // Clear foreground fixed-pt. accum.
MRF = SAT MRF;                  // nasyc pierwszoplanowy akumulator
R5 = RND MRB;                   // transfer zaokraglonych 32-bitowych wartosci
                                //  z pierwsz. akum. do rejestru
F0 = F1 * F2;                   // zmiennoprzecinkowe mnozenie
```

Multiplier/MAC Instrukcje

Przyklad:

MRF = Rx * Ry (SSFR);

(_ _ _)	**S**	**Signed Input(wejscie ze znakiem)**
	U	**Unsigned Input(wejscie_bez_znaku)**
	I	**Integer input(s)(calkowite wejscie(a))**
	F	**Fractional input(s)(ulamkowe wejscie(a))**
	FR	**ulamkowe wejscia, zaokraglone wyjscie**

Rn, Rx, Ry	R15-R0; register file locations, staloprzecinkowa
Fn, Fx, Fy	F15-F0; register file locations, zmiennoprzecinkowa
MRF	MR2F, MR1F, MR0F; accumulatory wyniku MAC, pierwszoplanowy
MRB	MR2B, MR1B, MR0B; accumulatory wyniku MAC, drugoplanowy
(SF)	Domyslny format dla 1-wejsciowych operacji
(SSF)	Domyslny format dla 2-wejsciowych operacji

Instrukcje staloprzecinkowe MAC'a

Rn MRF MRB	= Rx * Ry (	S U	S U	F I FR	) ;

Rn = MRF Rn = MRB MRF = MRF MRB = MRB	+ Rx * Ry (	S U	S U	F I FR	) ;

Rn = MRF Rn = MRB MRF = MRF MRB = MRB	– Rx * Ry (	S U	S U	F I FR	);

MRF MRB	= 0 ;

Rn = SAT MRF Rn = SAT MRB MRF = SAT MRF MRB = SAT MRB	(SF); (UI); (SI) ; (UF);

Rn = RND MRF Rn = RND MRB MRF = RND MRF MRB = RND MRB	(SF) ; (UF);

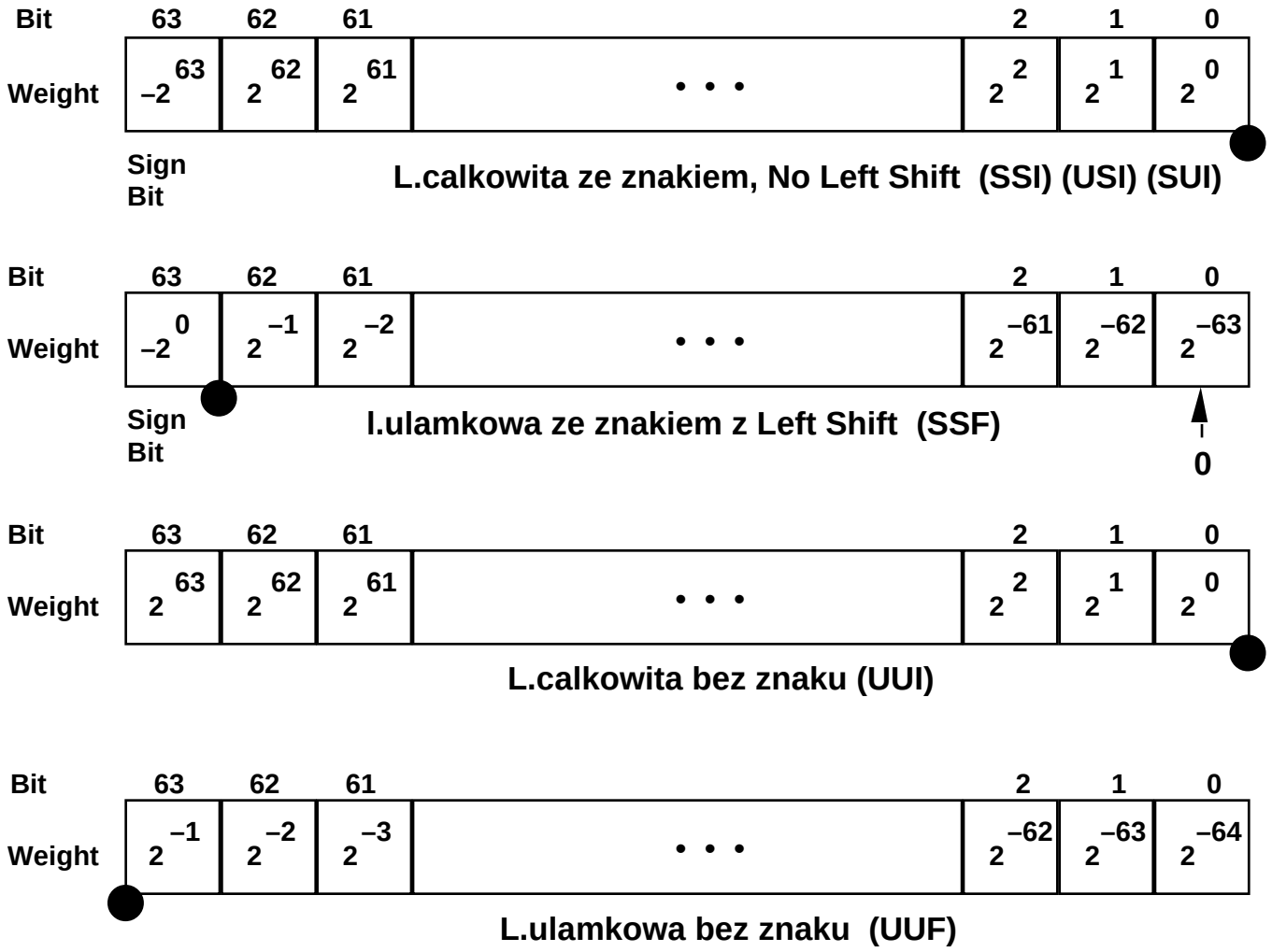
MRxF MRxB	= Rn ;

Rn = | MRxF
MRxB | ;

Instrukcje zmiennoprzecinkowe MAC'a

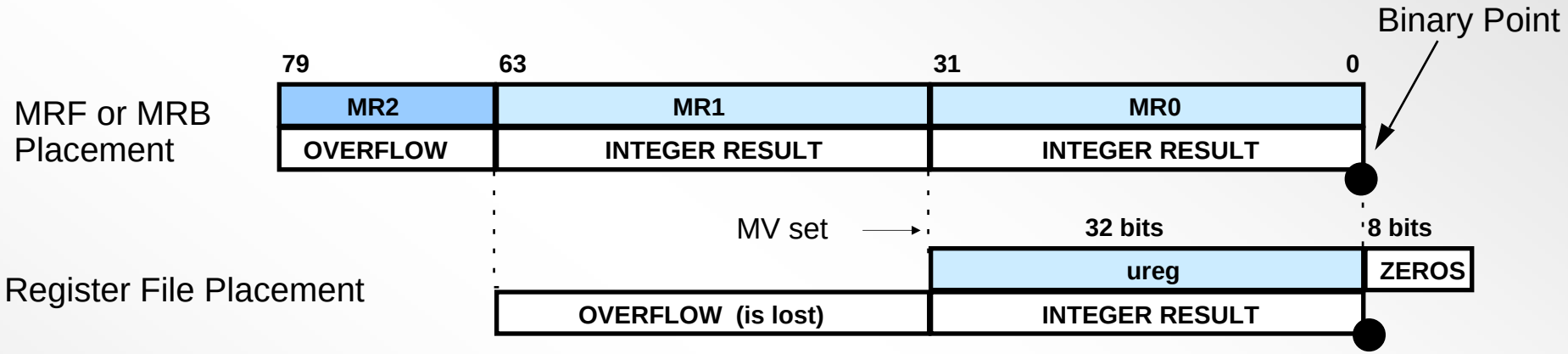
Fn = Fx * Fy;

Staloprzecinkowe formaty danych wyników MAC'a

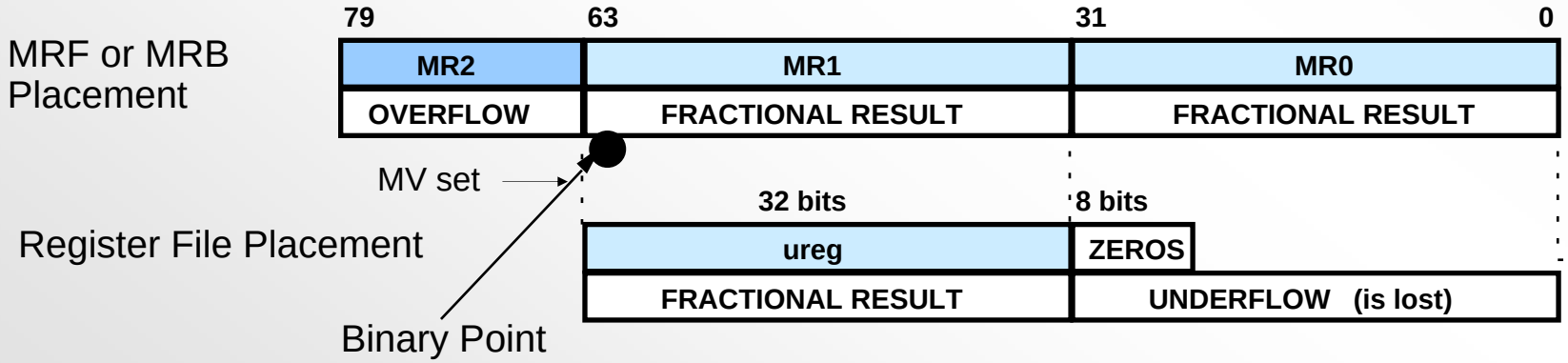


Calkowite/ulamkowe staloprzecinkowe mnozenie

- **Umiejscowienie staloprzecinkowego calkowitego wyniku mnozenia**



- **Umiejscowienie staloprzecinkowego ulamkowego wyniku mnozenia**



Zaokraglanie mnozenia staloprzecinkowego

- Zaokraglanie moze wystapic jako czesc mnozenia(SSFR),lub jawna instrukcja (RND)
- Staloprzecinkowe zaokraglanie uzywa sie do wynikow staloprz. ulamkowych
- 80-bitowe wyniki “rounded to nearest”(do najblizszej) wartosci 32-bitowej

```
Command                  MRF2     MRF1   MRF0
mrf = r1 * r2 (ssf);   0000   7999 9999    9999 9999
mrf = r1 * r2 (ssfr); 0000   7999 999A    0000 0000
```

Nasycanie mnozenia staloprzecinkowego

- Instrukcja nasycenia Saturation (SAT) ustawia MR do max. wartosci jesli MR jest przepelnione (MV=1).
- wyniki Fixed Point Saturation moga byc zwrócone do rejestrów MR lub do pliku rejestru
- Mozliwe maksymalne wartosci:

Typ mnozenia	**MRF**	**MRF1**	**MRF0**
ulamkowe ze zn	0000	7FFFFFFF	FFFFFFFF
	FFFF	80000000	00000000
calkowite ze zn	0000	00000000	7FFFFFFF
	FFFF	FFFFFFFF	80000000
ulamkowe b/znaku	0000	FFFFFFFF	FFFFFFFF
calkowite b/znaku	0000	00000000	FFFFFFFF

Mnozenie zmiennoprzecinkowe

- Zmiennoprzecinkowe wejscia/wyniki moga byc 32-bitowe lub 40-bitowe (MODE1/RND32)
- Zmiennoprzecinkowe mnozenia moga byc “Rounded to Nearest” lub “Rounded(to Zero(zaokra. do zera)/Truncation(obcinane)” (MODE1/TRUNC)
- Miejscem przeznacznia zmniennoprzecinkowych mnozen jest plik rejestru. Na przyklad:

```
F12 = F3 * F7;
```

- zmiennoprzecinkowe mnozenie/sumowanie (MAC) jest osiagane przez uzycie wielofunkcyjnych instrukcji (w/ ALU).

Przyklad:

(podwojny dostep do danych, pobranie/zaladowanie do pamieci F3 i F7);

```
F12 = F3 * F7,  (podwojny dostep do danych, F3,F7);
F12 = F3 * F7,  F8 = F8 + F12;
                F8 = F8 + F12;
```

Flagi statusu MAC'a

ASTATx/y

Bit	*Nazwa*	*Definicja*
6	MN	Multiplier result negative(wynik ujemny mnozenia)
7	MV	Multiplier overflow(MAC nadmiar)
8	MU	Multiplier underflow(MAC niedomiar)
9	M	Multiplier floating-point invalid operation(zmienn. niewazna op.)

STKYx/y

Bit	*Nazwa*	*Definicja*
6	MOS	Multiplier fixed-point overflow(MAC staloprzecinkowy nadmiar)
7	MVS	Multiplier floating-point overflow(MAC zmiennoprz. nadmiar)
8	MUS	Multiplier underflow(MAC niedomiar)
9	MIS	Multiplier floating-point invalid operation(zmienn.niewazna op.)

Multiplier/MAC: warunkowo wykonywane kody

ms	Multiplier sign(znak mnozenia)
mv	Multiplier overflow(przepelnienie mnozenia)
NOT ms	Not multiplier sign(~znak mnozenia)
NOT mv	Not multiplier overflow(~przepelnienie mnozenia)

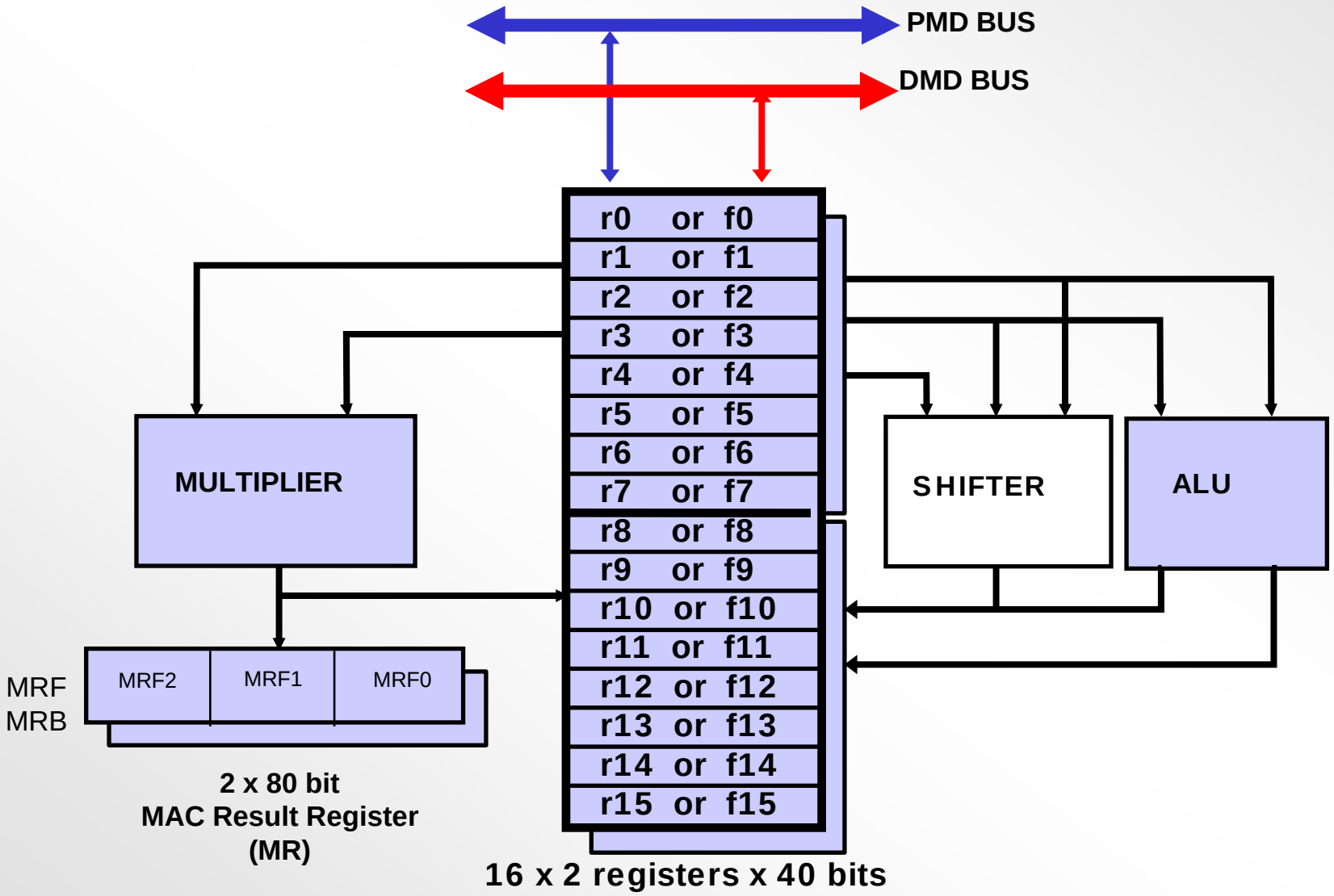
```
r0= 0x7FFFFFFF;
r1= 0x50;
mrf=r0*r1(SSI);        {mv is set}
if mv jump save;
save:
        R0=MR0F;
        R1=MR1F;
        R2=MR2F;
```

```
r0=-5;
r1=2;
r2=r0*r1(SSI);         {mn is set}
if ms jump add;
add:
        r3=r0+r1;
```

MAC: Cwiczenia i mini-quiz

LAB 3

SHIFTER(przesuwnik)



ADSP-21161 32bitowy przesuwnik Barrela: cechy

- Przesuniecia arytmetyczne i logiczne
- Obracanie slów
- manipulowanie polem danych (ekstrakcja ,skladanie)
- ustawianie/ czyszczenie / przelaczanie / testowanie bitu
- staloprzecinkowe/zmiennoprzecinkowe operacje konwersji, wlaczajac wyciaganie wykladnika, liczbe wiodacych 1 lub 0

32bitowy przesuwnik Barrela
Przykladowe instrukcje

```
R3 = LSHIFT R4 BY -11;          //  R4 logicznie przesuniety w prawo o
                                //     11 bitow, resultat w R3
R1 = R1 OR LSHIFT R2 BY 8;      //  R2 logicznie przesuniety w lewo o 8 bitow,
                                //     resultat "OR"ed w R1
R15 = ASHIFT R14 by R3;         //  R14 arytmetycznie przesuniety o wartosc
                                //    w R3, wynik w R15
R10 = FDEP R9 by 16:8;          //  8 LSBs R9 sa skladane do
                                //     R10 zaczynajac od bit 16 w R10
R4 = FEXT R6 BY R5;             //  Pole wewnatrz R6 jest ekstraktowane i
                                //     umieszczane w  R4 right-aligned
R5 = LEFTZ R0;                  //  R5 pobiera liczbe wiodacych
                                //    zer w R0
```

ADSP-21161 32bitowy przesuwnik Barrela Instrukcje

Shifter

Rn = LSHIFT Rx BY Ry
Rn = Rn OR LSHIFT Rx BY Ry
Rn = ASHIFT Rx BY Ry
Rn = Rn OR ASHIFT Rx BY Ry
Rn = ROT Rx BY RY
Rn = BCLR Rx BY Ry
Rn = BSET Rx BY Ry
Rn = BTGL Rx BY Ry
BTST Rx BY Ry
Rn = FDEP Rx BY Ry
Rn = Rn OR FDEP Rx BY Ry
Rn = FDEP Rx BY Ry (SE)
Rn = Rn OR FDEP Rx BY Ry (SE)
Rn = FEXT Rx BY Ry
Rn = FEXT Rx BY Ry (SE)
Rn = EXP Rx
Rn = EXP Rx (EX)
Rn = LEFTZ Rx
Rn = LEFTO Rx

Shifter Immediate

Rn = LSHIFT Rx BY <data8>
Rn = Rn OR LSHIFT Rx BY <data8>
Rn = ASHIFT Rx BY <data8>
Rn = Rn OR ASHIFT Rx BY <data8>
Rn = ROT Rx BY <data8>
Rn = BCLR Rx BY <data8>
Rn = BSET Rx BY <data8>
Rn = BTGL Rx BY <data8>
BTST Rx BY <data8>
Rn = FDEP Rx BY <bit6>:<len6>
Rn = Rn OR FDEP Rx BY <bit6>:<len6>
Rn = FDEP Rx BY <bit6>:<len6> (SE)
Rn = Rn OR FDEP Rx BY <bit6>:<len6> (SE)
Rn = FEXT Rx BY <bit6>:<len6>
Rn = FEXT Rx BY <bit6>:<len6> (SE)

Rn, Rx, Ry	R15-R0; register file location, fixed-point
<bit6>:<len6>	6-bit immediate bit position and length values (for shifter immediate operations)

Flagi statusu przesuwnika

(Shifter Status Flags)

ASTATx/y

Bit	*Nazwa*	*Definicja*
11	SV	Shifter overflow of bits to left of MSB
12	SZ	Shifter result zero
13	SS	Shifter input sign for exponent extract

Shifter: warunkowo wykonywane kody

s	**Shifter overflow of bits to left of MSB(nadmiar)**
sz	**Shifter result zero(wynik przesuwnika=0)**
NOT sv	**Not Shifter overflow(~ nadmiar przesuwnika)**
NOT sz	**Not Shifter zero(~wynik przesuwnika=0)**

*jesli program jawnie zapisuje rejestr ASTAT niepelnie w tym samym cyklu, w którym przesuwnik wykonuje jakas operacje, jawny zapis do ASTAT zastepuje aktualizacje flagi wywolana przez dana operacje przesuwania

ADSP-21161 32bitowy przesuwnik Barrela
Logiczne i arytmetyczne przesuwanie

Przed:

R0

39	31	23	15	7	0
11111111	00000000	00000000	00000000	00000000	

FF00000000

R1

39	31	23	15	7	0
11111111	11111111	11111111	11111000	00000000	

FFFFFFF800

Logiczne przesuniecie

A po: R2 = LSHIFT R0 BY -8 or R2 = LSHIFT R0 BY R1

R2

39	31	23	15	7	0
00000000	11111111	00000000	00000000	00000000	

00FF000000

Arytmetyczne przesuniecie

A po: R2 = ASHIFT R0 BY -8 or R2 = ASHIFT R0 BY R1

R2

39	31	23	15	7	0
11111111	11111111	00000000	00000000	00000000	

FFFF000000

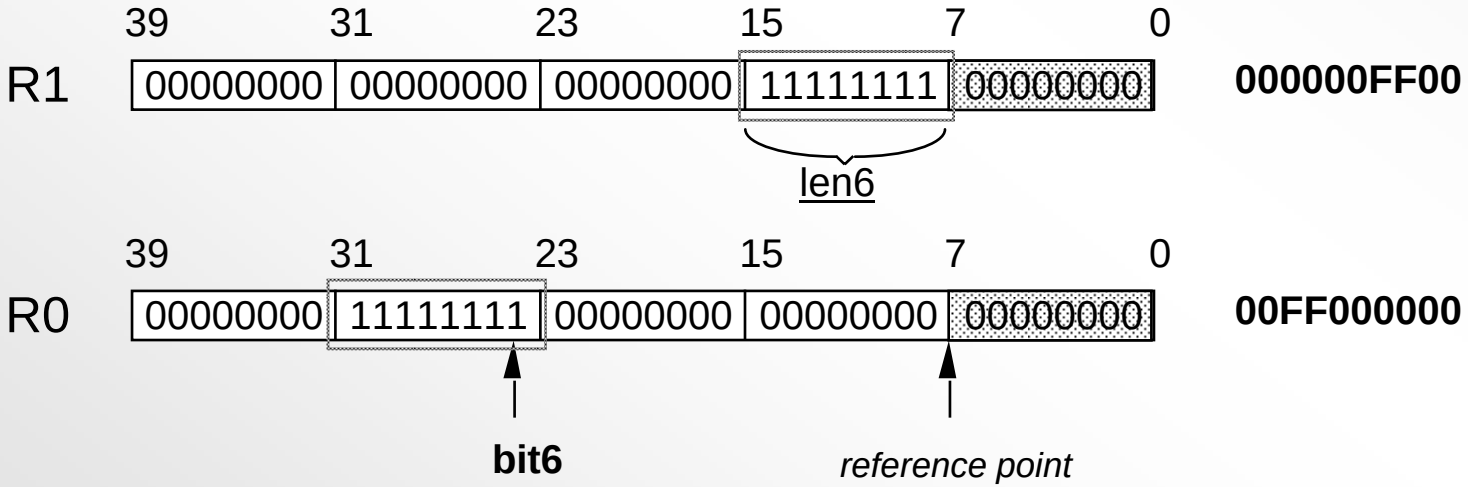
ADSP-21161 32bitowy przesuwnik Barrela
Przenoszenie pola bitowego

Rn = *Rn OR* FDEP Rx BY Ry *(SE)*;

Rn = *Rn OR* FDEP Rx BY <bit6>:<len6>*(SE)*;

Przyklad:

R0 = FDEP R1 BY 16:8;



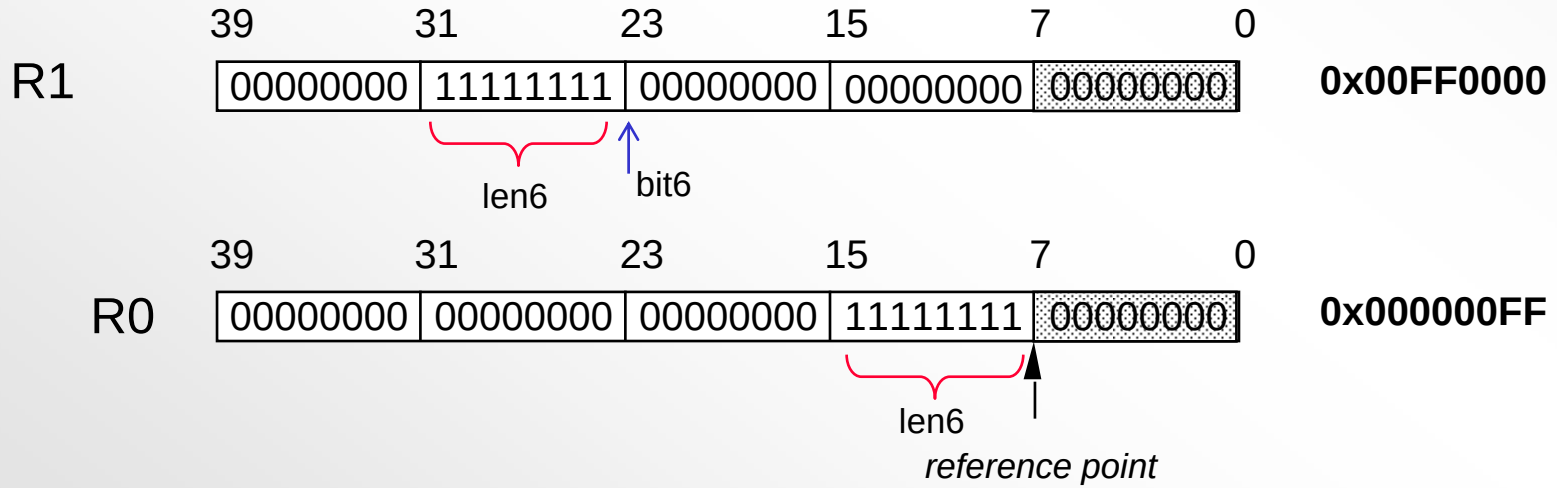
ADSP-21161 3bitowy przesuwnik Barrela
Wydobycie pola bitowego

Rn = *Rn OR* FEXT Rx BY Ry *(SE)*;

Rn = *Rn OR* FEXT Rx BY <bit6>:<len6> *(SE)*;

Przyklad:

R0 = FEXT R1 BY 16:8;



ADSP-21161 3bitowy przesuwnik Barrela
Operacje bitowe

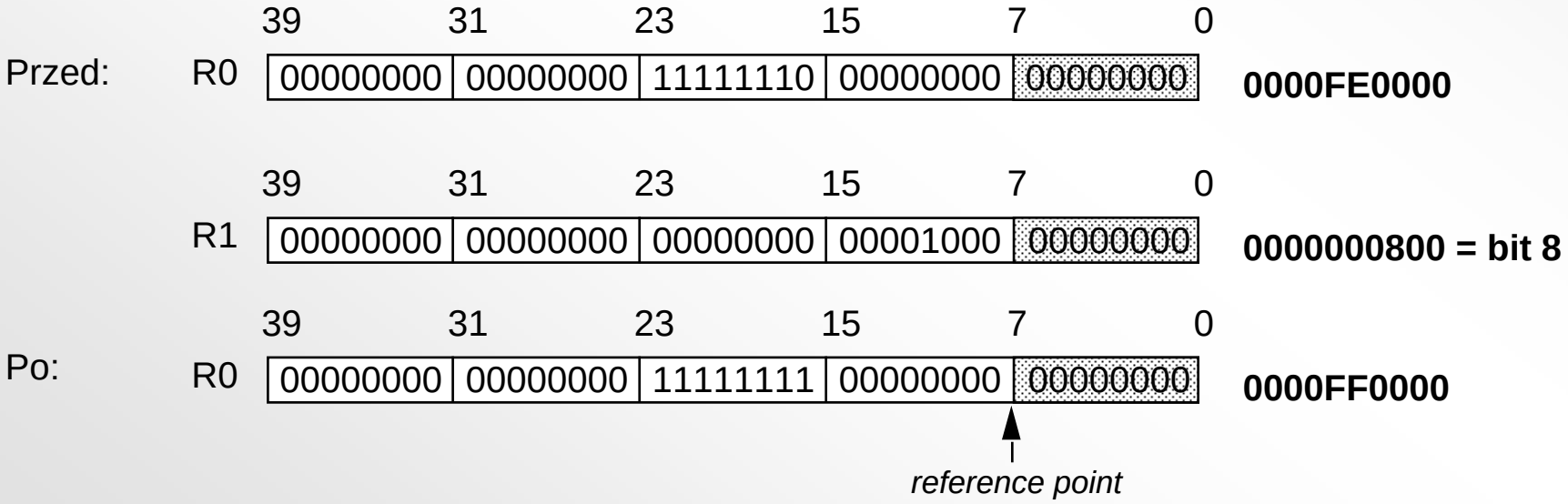
Rn = BCLR Rx BY Ry
Rn = BCLR Rx BY <data8>

Rn = BTGL Rx BY Ry
Rn = BTGL Rx BY <data8>

Rn = BSET Rx BY Ry
Rn = BSET Rx BY <data8>

BTST Rx BY Ry
BTST Rx BY <data8>

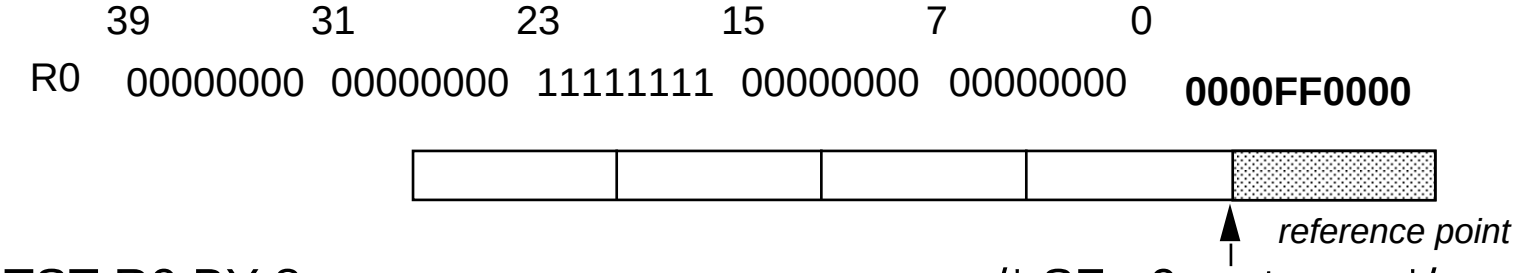
Przyklad: R0 = BSET R0 BY R1;



32bitowy przesuwnik Barrela

BTST Rx BY Ry
BTST Rx BY <data8>

Przyklad:



BTST R0 BY 8; /* SZ= 0, not zero */
IF SZ JUMP skocz gdzies; /* No jump */

BTST R0 BY 7; /* SZ = 1, zero */
IF SZ JUMP *skocz gdzies indziej;* /* Jump occurs*/

32bitowy przesuwnik Barrela
Przyklady i mini-quiz

LAB 4