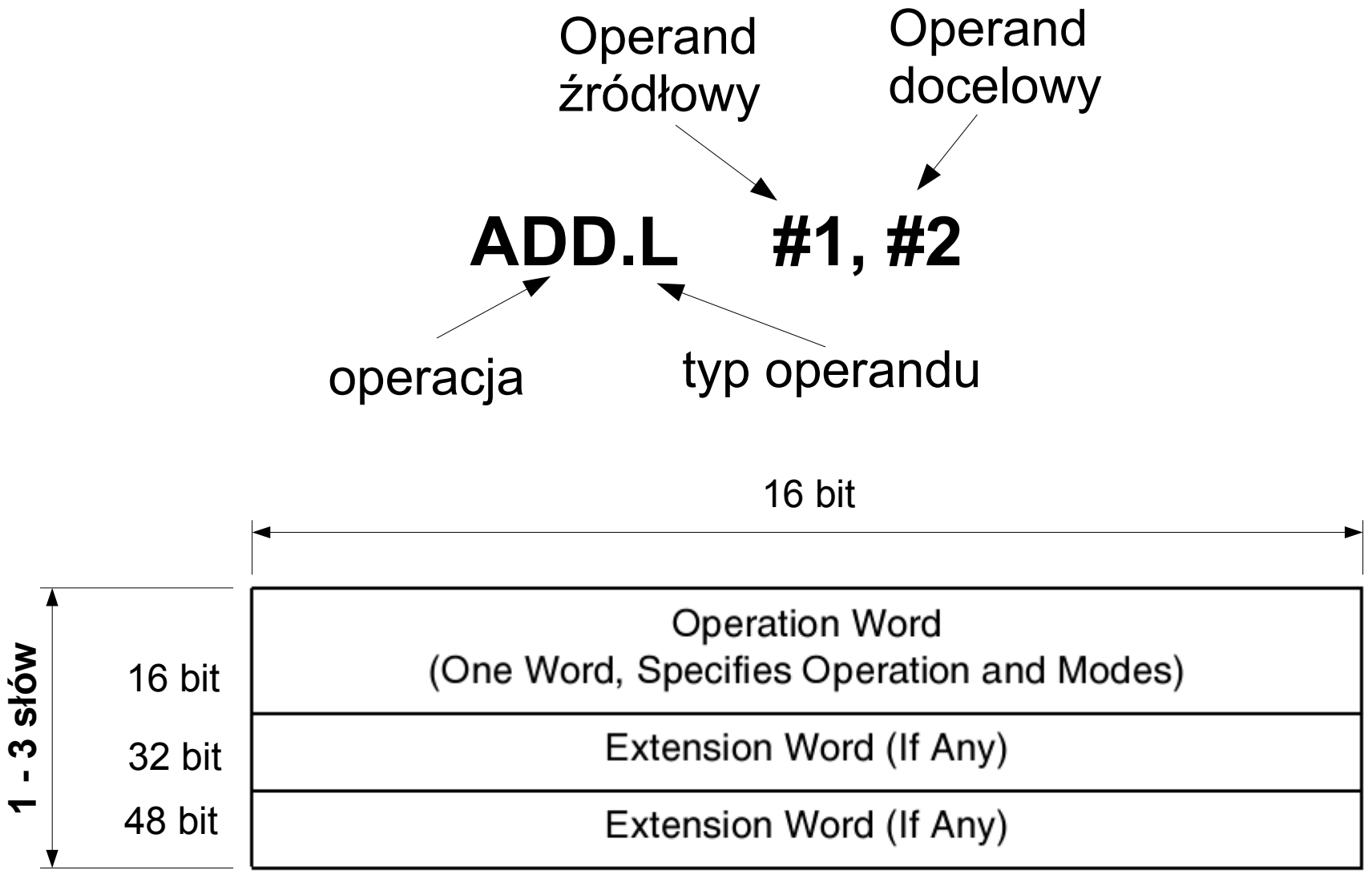
Tryby adresowania procesorów rodziny ColdFire

Format instrukcji (1)



Format instrukcji (2)

Operation Word Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	Effective Address					
										Mode			Register		

Extension Word Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	Register			W/L	Scale		0	Displacement							

Field	Definition
Instruction	
Mode	Addressing mode (see Table 2-3)
Register	General register number (see Table 2-3)
Extensions	
D/A	Index register type 0 = D*n* 1 = A*n*
W/L	Word/longword index size 0 = Sign-Extended Word 1 = Long Word
Scale	Scale factor 00 = 1 01 = 2 10 = 4 11 = 8 (supported only if FPU is present)

Przykładowa instrukcja (1)

ADDX **Add Extended** **ADDX**

First appeared in ISA_A

Operation: Source + Destination + CCR[X] → Destination

Assembler Syntax: ADDX.L Dy,Dx

Attributes: Size = longword

Description: Adds the source operand and CCR[X] to the destination operand and stores the result in the destination location. The size of the operation is specified as a longword.

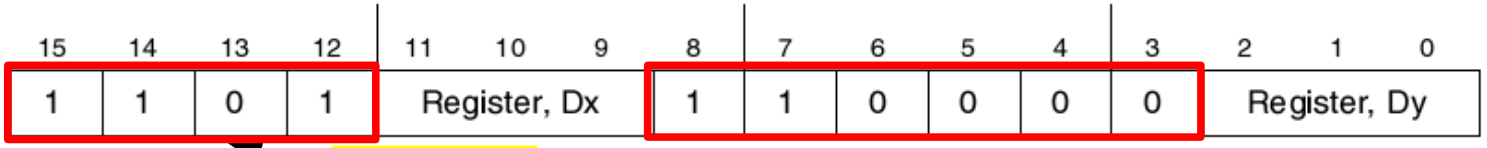
Condition Codes:

X	N	Z	V	C
*	*	*	*	*

X Set the same as the carry bit
N Set if the result is negative; cleared otherwise
Z Cleared if the result is non-zero; unchanged otherwise
V Set if an overflow is generated; cleared otherwise
C Set if an carry is generated; cleared otherwise

Normally CCR[Z] is set explicitly via programming before the start of an ADDX operation to allow successful testing for zero results upon completion of multiple-precision operations.

Instruction Format:



Instruction Fields:

- Register Dx field—Specifies the destination data register, Dx.
- Register Dy field—Specifies the source data register, Dy.

Długość instrukcji: 1 słowo

Przykładowa instrukcja (2)

Operation: Source + Destination → Destination

Assembler Syntax: ADD.L <ea>y,Dx
ADD.L Dy,<ea>x

Attributes: Size = longword

Description: Adds the source operand to the destination operand using binary addition and stores the result in the destination location. The size of the operation may only be specified as a longword. The mode of the instruction indicates which operand is the source and which is the destination as well as the operand size.

The Dx mode is used when the destination is a data register; the destination <ea>x mode is invalid for a data register.

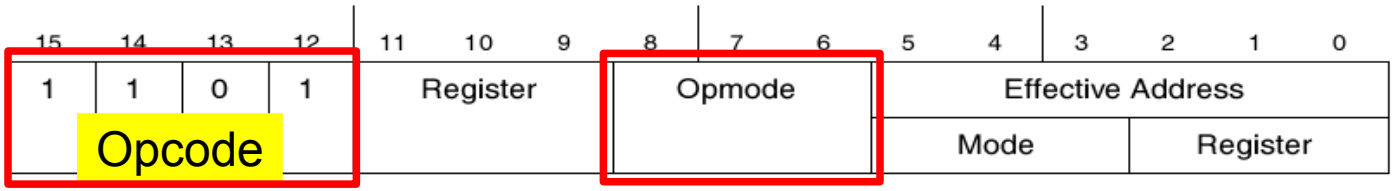
In addition, ADDA is used when the destination is an address register. ADDI and ADDQ are used when the source is immediate data.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

- X Set the same as the carry bit
- N Set if the result is negative; cleared otherwise
- Z Set if the result is zero; cleared otherwise
- V Set if an overflow is generated; cleared otherwise
- C Set if an carry is generated; cleared otherwise

Instruction Format:



Instruction Fields:

- Register field—Specifies the data register.
- Opmode field:

Długość instrukcji: 2-3 słów

Byte	Word	Longword	Operation
—	—	010	<ea>y + Dx → Dx
—	—	110	Dy + <ea>x → <ea>x

← **kod operacji**

Przykładowa instrukcja (3)

Instruction Fields (continued):

- Effective Address field—Determines addressing mode
  - — For the source operand <ea>y, use addressing modes listed in the following table:

Addressing Mode	Mode	Register
Dy	000	reg. number:Dy
Ay	001	reg. number:Ay
(Ay)	010	reg. number:Ay
(Ay) +	011	reg. number:Ay
– (Ay)	100	reg. number:Ay
(d_{16},Ay)	101	reg. number:Ay
(d_8,Ay,Xi)	110	reg. number:Ay

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

  - — For the destination operand <ea>x, use addressing modes listed in the following table:

Addressing Mode	Mode	Register
Dx	—	—
Ax	—	—
(Ax)	010	reg. number:Ax
(Ax) +	011	reg. number:Ax
– (Ax)	100	reg. number:Ax
(d_{16},Ax)	101	reg. number:Ax
(d_8,Ax,Xi)	110	reg. number:Ax

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

Notacja

Register Specifications	
A*n*	Any address register *n* (example: A3 is address register 3)
Ax, Ay	Destination and source address registers, respectively
D*n*	Any data register *n* (example: D5 is data register 5)
Dx, Dy	Destination and source data registers, respectively
Dw	Data register containing a remainder
Rc	Control register
R*n*	Any address or data register
Rx, Ry	Any destination and source registers, respectively
Xi	Index register, can be any address or data register; all 32-bits are used.
Subfields and Qualifiers	
#<data>	Immediate data following the instruction word(s).
( )	Identifies an indirect address in a register.
d_n	Displacement value, *n* bits wide (example: d_{16} is a 16-bit displacement).
sz	Size of operation: Byte (B), Word (W), Longword (L)
lsb, msb	Least significant bit, most significant bit
LSW, MSW	Least significant word, most significant word
SF	Scale factor for an index register

Tryby adresowania procesorów ColdFire

Addressing Modes	Syntax	Mode Field	Reg. Field	Data	Memory
Register Direct					
Data	Dn	000	reg. no.	X	—
Address	An	001	reg. no.	—	—
Register Indirect					
Address	(An)	010	reg. no.	X	X
Address with Postincrement	(An)+	011	reg. no.	X	X
Address with Predecrement	–(An)	100	reg. no.	X	X
Address with Displacement	(d_{16},An)	101	reg. no.	X	X
Address Register Indirect with Scaled Index and 8-Bit Displacement	(d_8,An,Xi*SF)	110	reg. no.	X	X
Program Counter Indirect with Displacement	(d_{16},PC)	111	010	X	X
Program Counter Indirect with Scaled Index and 8-Bit Displacement	(d_8,PC,Xi*SF)	111	011	X	X
Absolute Data Addressing					
Short	(xxx).W	111	000	X	X
Long	(xxx).L	111	001	X	X
Immediate	#<xxx>	111	100	X	X

Adresowanie natychmiastowe (1)

(Immediate addressing)

Operation Length	Location
Byte	Low-order byte of the extension word
Word	Entire extension word
Longword	High-order word of the operand is in the first extension word; the low-order word is in the second extension word.

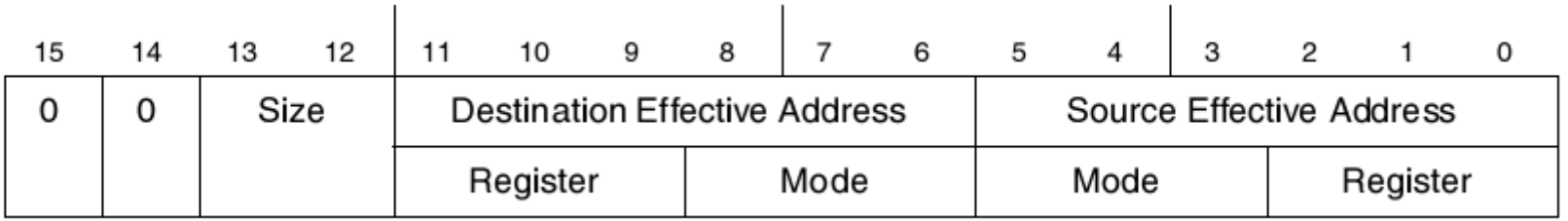
Generation	Operand given
Assembler Syntax	#<xxx>
EA Mode Field	111
EA Register Field	100
Number of Extension Words	1 or 2

```
| deklaracja adresu bazowego
      .equ      BaseAddress, 0x4000 0000

      MOVE.B    # 12, %D0
      MOVEA.B   # BaseAddress, %A5
      MOVEQ.L   # -127, %D7
      ADDI.L    # 0xABCD 1234, %D7

      ADDI.B    # 0x1A, %D4      Error: invalid instruction for this architecture,
                                  needs 68000 or higher
```

Adresowanie natychmiastowe (2)

Instruction Format:

MOVE



Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100

Addressing Mode	Mode	Register
Dy	000	reg. number:Dy
Ay	001	reg. number:Ay

68K GAS gp.asm page 1

```
1          .equ    IPSBAR, 0x40000000
2          .equ    PTCPAR, 0x0010005A
3          .equ    PTDPAR, 0x0010005B
4          .equ    DDRTC,  0x00100023
5          .equ    DDRTD,  0x00100024
6          .equ    PORTTC, 0x0010000F
7          .equ    PORTTD, 0x00100010
8
9                  _start:

10 0000    103C 0011          MOVE.B    #0x11,        %d0
11 0004    323C 2211          MOVE.W    #0x2211,      %d1
12 0008    247C 4433  2211    MOVE.L    #0x44332211,  %a2
12 000e
13
```

Size field

01 byte operation

11 word operation

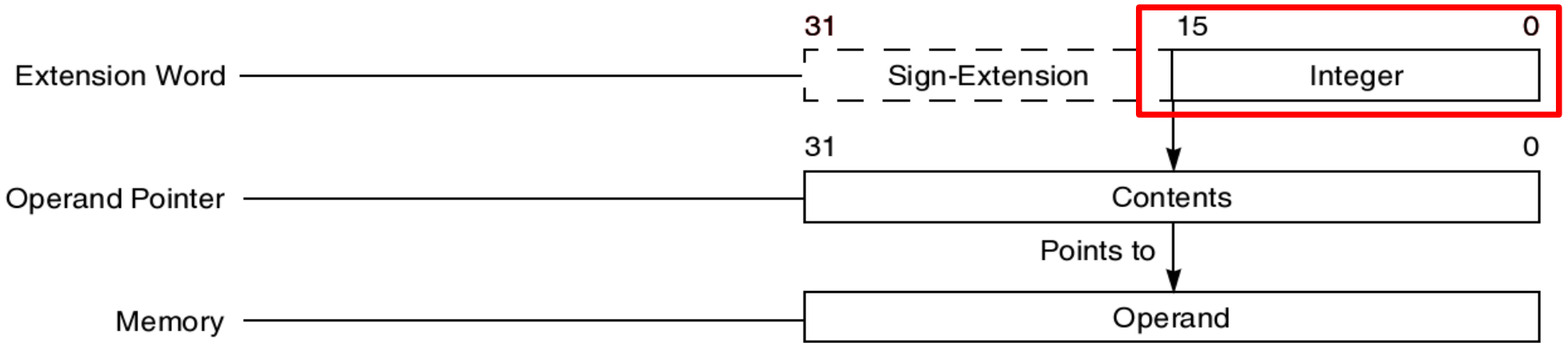
10 longword operation

Adresowanie bezpośrednie (1)

Operacje na urządzeniach I/O mapowanych na przestrzeń pamięci, których położenie w przestrzeni adresowej jest znane i niezmienne (w czasie wykonywania programu).

(Absolute/Direct Short Addressing)

Generation	EA Given
Assembler Syntax	(xxx).W
EA Mode Field	111
EA Register Field	000
Number of Extension Words	1



Absolute Short Addressing

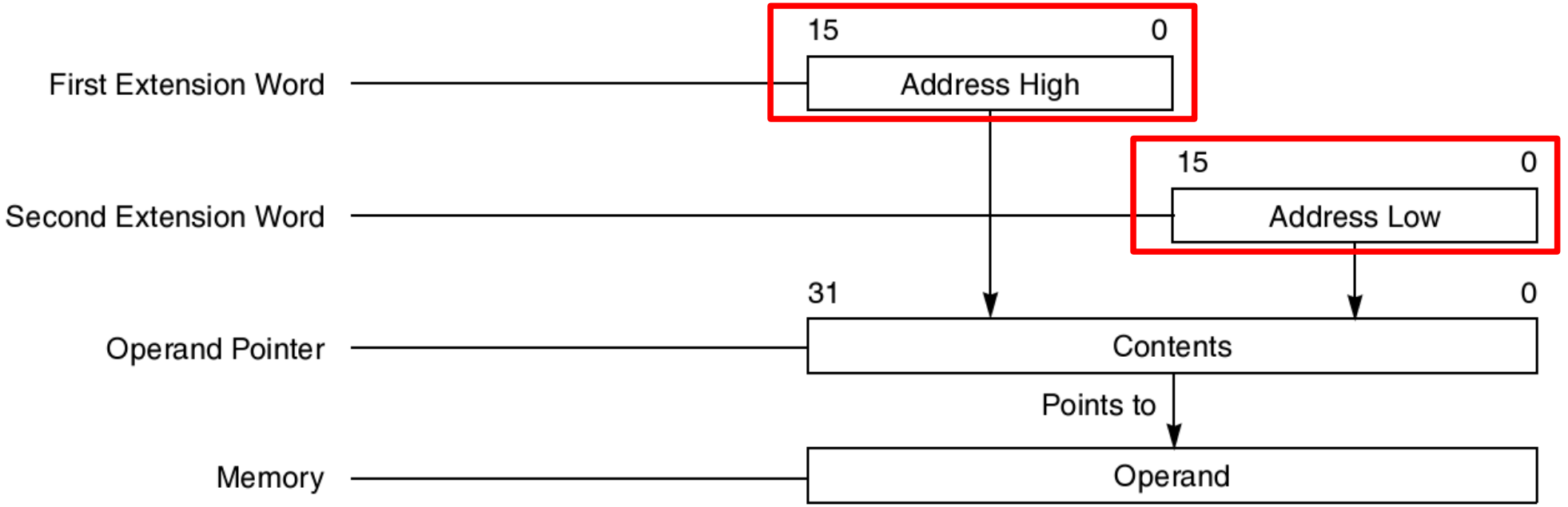
Adresowanie krótkie umożliwia odwołanie do pierwszych i ostatnich 32 kB przestrzeni adresowej.

(0x8000 – 0xFFFF) 16 bit => (0xFFFF–8000 - 0xFFFF FFFF) 32 bit

Adresowanie bezpośrednie (2)

(Absolute/Direct Long Addressing)

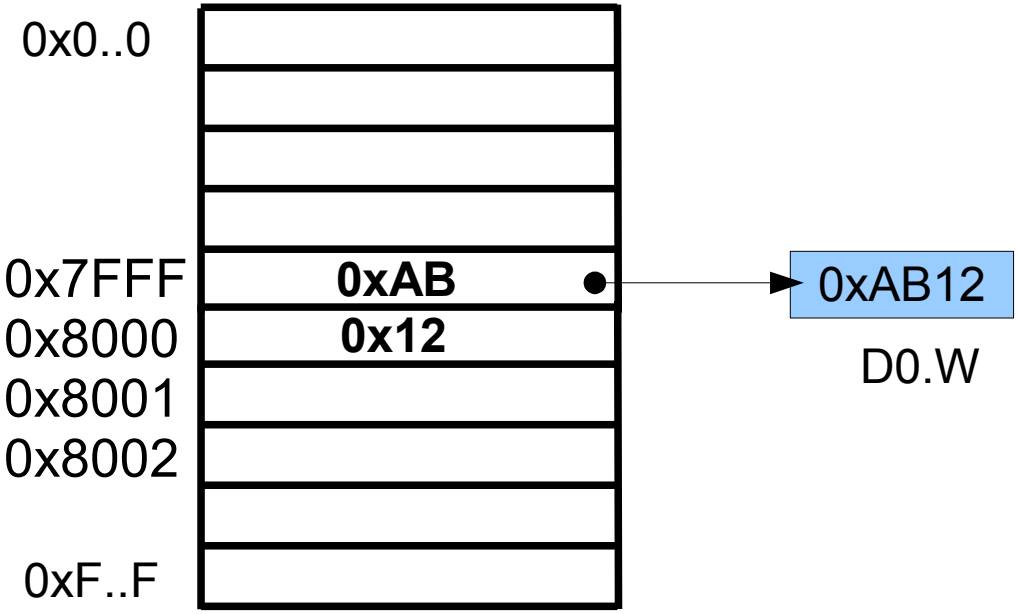
Generation	EA Given
Assembler Syntax	(xxx).L
EA Mode Field	111
EA Register Field	001
Number of Extension Words	2



Absolute Long Addressing

Adresowanie bezpośrednie (3)

(Adresowanie absolute)



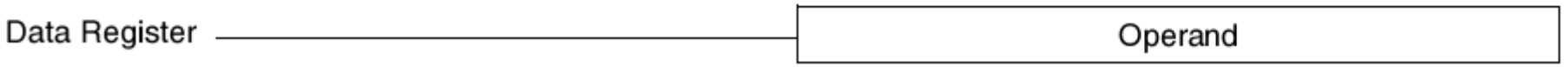

```
.equ PIT_PCSR, 0x4015 0000
.equ IPSBAR, 0x4000 0000
.equ GPTSCR1, 0x1 a006
```

MOVE.W	0x7FFF, %D0	|	3038 7FFF
MOVE.W	0x8000, %D0	| D. LONG A.	3039 0000 8000
MOVE.W	-0x8000, %D0	|	3038 8000
MOVE.L	0xDEAD BEEF, (%A0)	|	20B9 DEAD BEEF
MOVE.B	%d0, PIT_PCSR	| forma dozwolona	13C0 4015 0000
MOVE.B	%d0, (IPSBAR+GPTSCR1)	| forma z nawiasami	13C0 4001 A006

Adresowanie bezpośrednie rejestrowe (1)

(Data Register Direct Addressing)

Generation	EA = Dn
Assembler Syntax	Dn
EA Mode Field	000
EA Register Field	Register number
Number of Extension Words	0

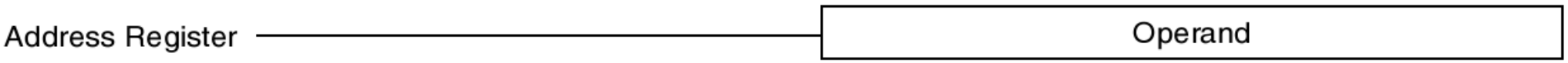



```
MOVE.B    %D0, %D1          | przepisanie zawartości rejestru D0 do D1
AND.L     0xEF, %D7         | maskowanie podwójnego słowa maską pod
                            | adresem 0xEF
MOVEQ.L   # -127, %D7
CLR.W     %D6               | wyczyszczenie młodszej części rejestru D6

ADDI.L  # 0xABCD 1234, %D7  | załadawanie rejestru D7

CLR.L     %D1               | zerowanie rejestru D1
```

Adresowanie bezpośrednie rejestrowe (2)

(Address Register Direct Addressing)

Generation	EA = An
Assembler Syntax	An
EA Mode Field	001
EA Register Field	Register number
Number of Extension Words	0



Address Register Direct

Adresowanie bezpośrednie rejestrowe (3)

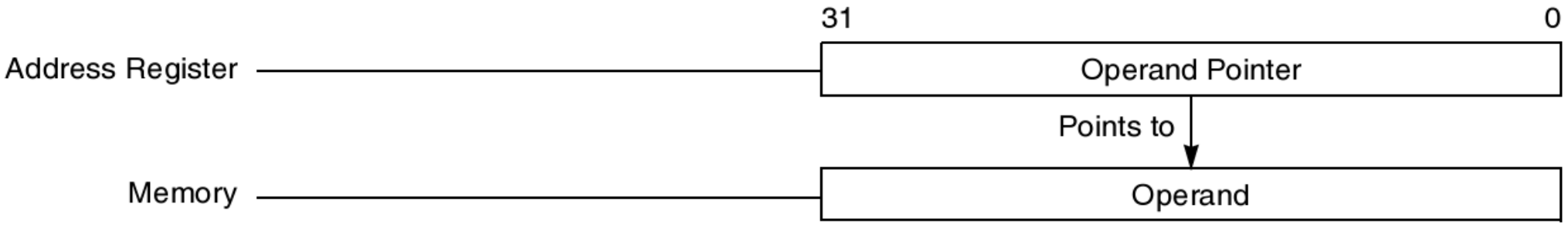
```
MOVE.B    %A0, %D1          | przepisanie zawartości (1 B) z rejestru A0 do D1
AND.L     0xEF, %A7         | maskowanie podwójnego słowa maską znajdującą się
                            | pod adresem 0xEF
MOVEQ.L   # -127, %A7       | szybkie załadowanie wartości -127 do rejestru A7

ADDI.L    # 0xABCD 1234, %A7    | dodanie wartości 0xABCD1234 do rejestru A7

CLR.L     %A1               | zerowanie rejestru A1
```

```
MOVE.B    %A0, %D1   Error: Operands mismatch – statement 'move.b %a0, %d0' ignored
MOVE.W    %A0, %D1           | przepisanie zawartości rejestru A0 do D1


MOVEQ.L   # -127, %A7       Error: Operands mismatch – statement 'moveq.l # -127, %a7'
ignored
MOVEA.L   # -127, %A7       | zapisanie wartości -127 do rejestru A7

ADDI.L  # 0xABCD 1234, %A7  Error: Operands mismatch – statement
                            'addi.l #0xABCD1234, %a7' ignored
ADDA.L  # 0xABCD 1234, %A7  | zapisanie rejestru A7 wartością 0xABCD1234

CLR.L     %A1          Error: Operands mismatch – statement 'clr.l %a1' ignored
MOVEA.L   0x0, %A1             | wyczyszczenie rejestru A1
```

Adresowanie pośrednie rejestrowe (1)

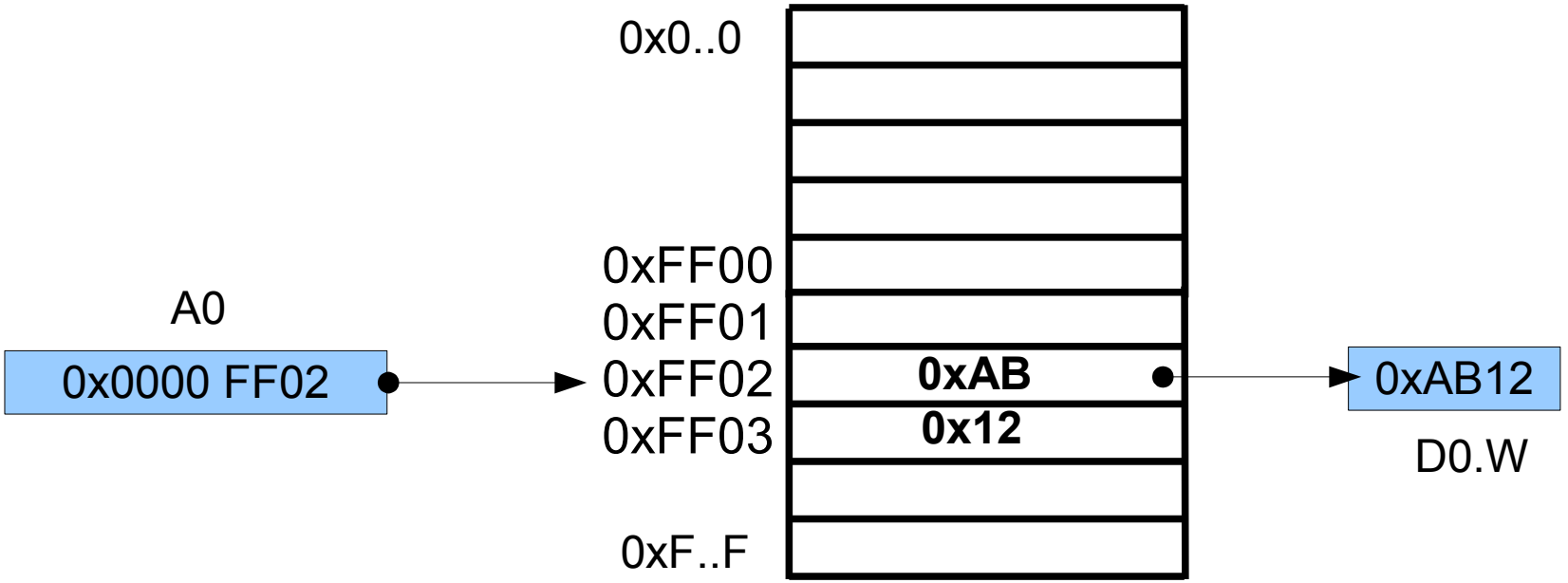
(Register Indirect Addressing)

Generation	EA = (An)
Assembler Syntax	(An)
EA Mode Field	010
EA Register Field	Register number
Number of Extension Words	0



Address Register Indirect

Adresowanie pośrednie rejestrowe (2)

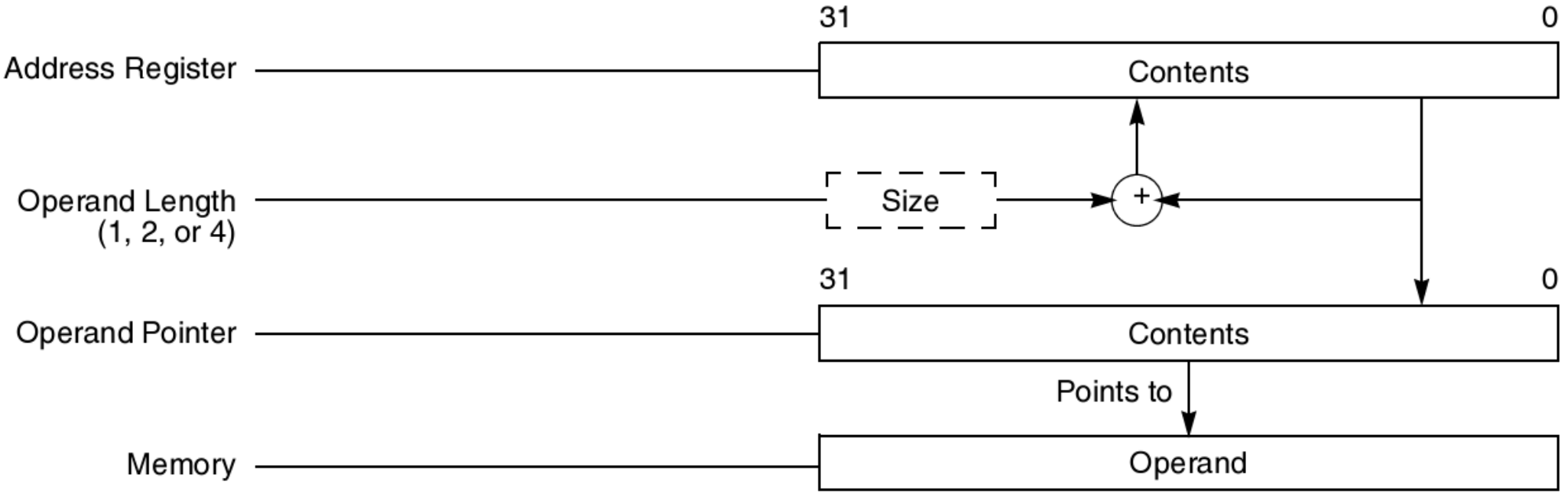



```
MOVE.B    # 12, (%A0)      | wpisuje liczbę 12 do komórki pamięci
                           | wskazywanej przez rejestr A0

MOVE.L    (%A2), %D0      | przesyła zawartość komórki pamięci
                           | wskazywanej przez rejestr A2, do rejestru D0

BSET.B    # 5, (%A7)       | testuje i ustawia 5 bit bajtu umieszczonego
                           | pod adresem wsk. przez A7 (tylko w 1B), jeżeli bit=0
                           | ustawia flagę Z

SUB.L     (%A3), %D0      | odejmuje zawartość komórki pamięci wskazywanej przez A3
                           | od rejestru D0
```

Adresowanie pośrednie rejestrowe z postinkrementacją (1)

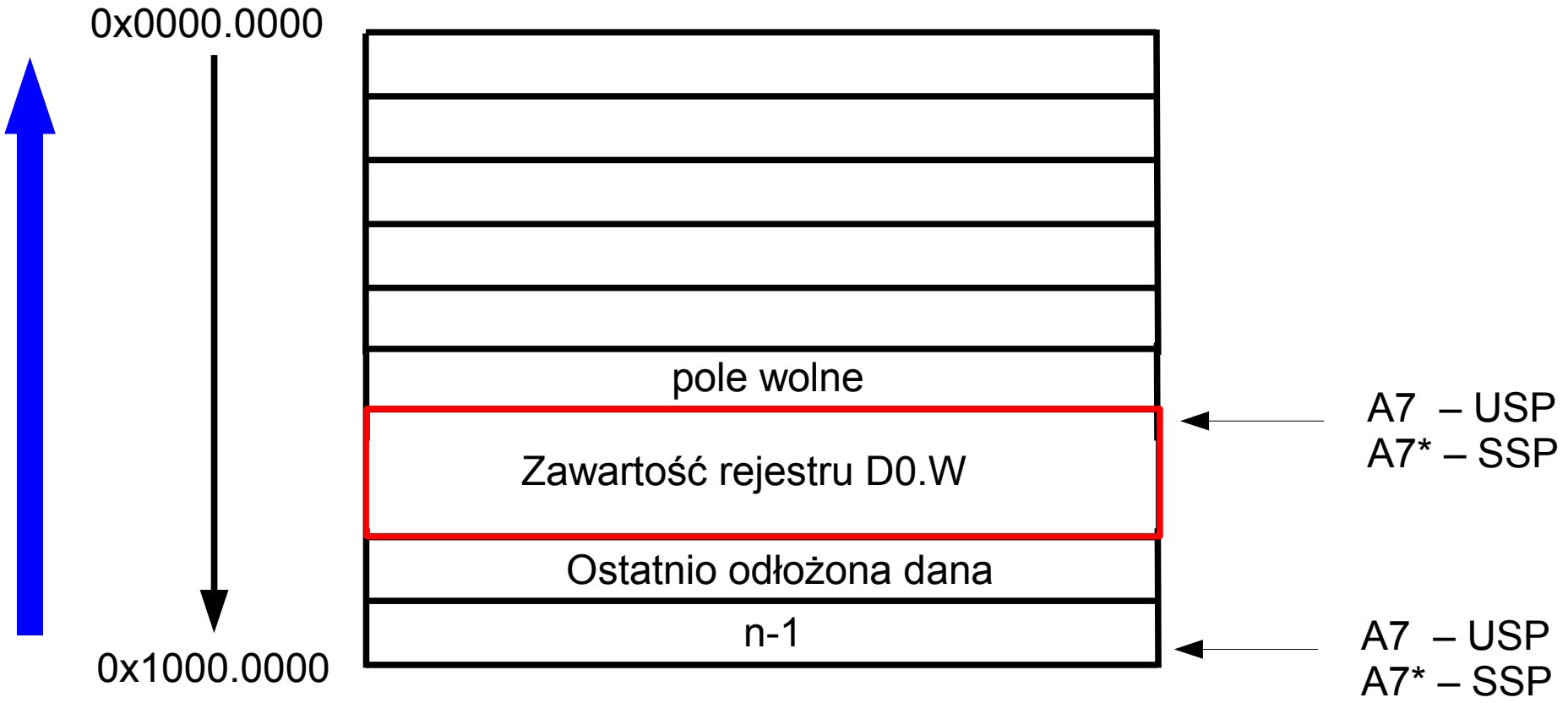
(Address Register Indirect with Postincrement Mode)

Generation	EA = (An); An = An + Size
Assembler Syntax	(An)+
EA Mode Field	011
EA Register Field	Register number
Number of Extension Words	0



Address Register Indirect with Postincrement

Adresowanie pośrednie rejestrowe z postinkrementacją (2)




```
MOVE.W    (%A7)+, %D0    | zdjęcie słowa ze stosu i zapisanie D0.W, po przesłaniu danej
                         | rejestr A7 jest zwiększany o 2

SUB.L     (%A2)+, %D7    | dodanie zaw. przechowywanej na stosie (A2) do rejestru D7
                         | wynik zapisany w rejestrze D7, zwiększenie A2 o 4

MOVE.L    (%A0)+, (%A1)+ | przesłanie zaw. komórki wskazywanej przez A0 do kom. wsk.
                         | przez A1, inkrementacja A0 oraz A1
```

Adresowanie pośrednie rejestrowe z postinkrementacją (3)

Obliczyć średnią arytmetyczną z wykonanych pomiarów. Zmierzone i przekonwertowane dane znajdują się w tablicy TABLE.

```
.equ        TABSIZE,    256
            ...
            ...
            ...

AVERAGE:

            •
            •
            •
            •
            •
            •
            •
            •
            •

TABLE:  .byte       0x01, 0xFE, 0x02, 0xDE, 0xAD, 0xBE, 0xEF, 0x04, ...
.even
```

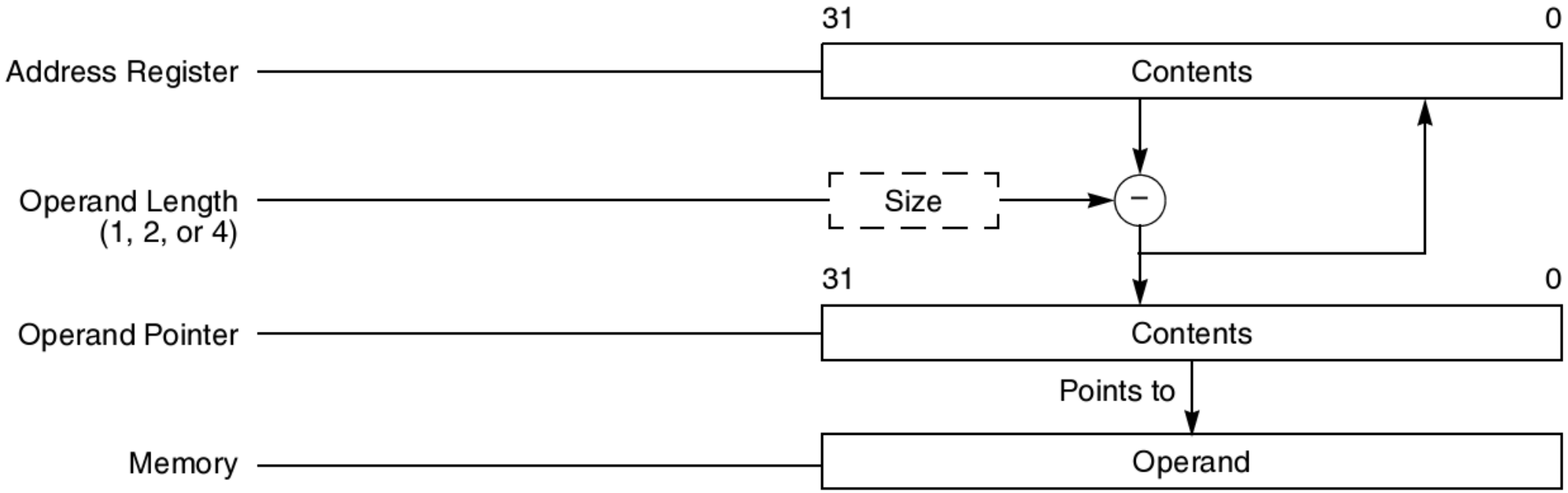
Adresowanie pośrednie rejestrowe z postinkrementacją (4)

```
.equ        TABSIZE,    256
            ...
            ...
            ...

AVERAGE:    MOVEA.L     #TABLE, %A0         | wpisz adres tablicy do A0
            LEA         TABLE, %A0          | wpisz adres tablicy do A0
            CLR.L       %D0                 | wyczyść D0
            CLR.L       %D1                 | wyczyść D1


M_LOOP:     MOVE.B      (%A0)+, %D0         | pobierz 1 bajt z tablicy TABLE [n], zwiększ A0
            ADD.L       %D0, %D1            | dodaj odczytaną wartość do całkowitej sumy
            CMPA.L      #(TABLE+TABSIZE), %A0     | czy przetworzono wszystkie dane?

            BNE         M_LOOP              | jeżeli nie, kontynuuj sumowanie
            ASR.W       #8, %D1             | podziel sumę przez 256
            RTS


TABLE:  .byte       0x01, 0xFE, 0x02, 0xDE, 0xAD, 0xBE, 0xEF, 0x04, ...
.even
```

Adresowanie pośrednie rejestrowe z predekrementacją (1)

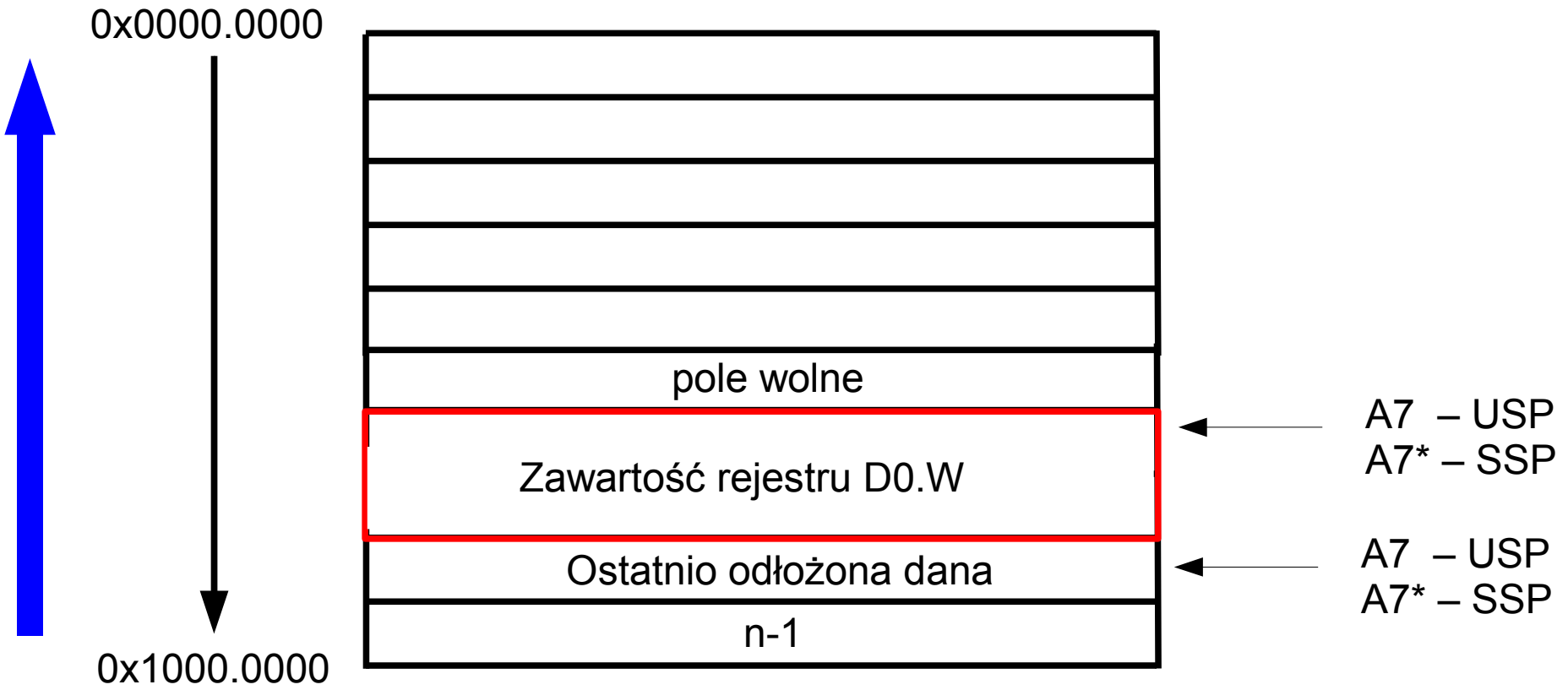
(Address Register Indirect with Predecrement Mode)

Generation	EA = (An) - Size; An = An - Size;
Assembler Syntax	–(An)
EA Mode Field	100
EA Register Field	Register number
Number of Extension Words	0



Address Register Indirect with Predecrement

Adresowanie pośrednie rejestrowe z predekrementacją (2)

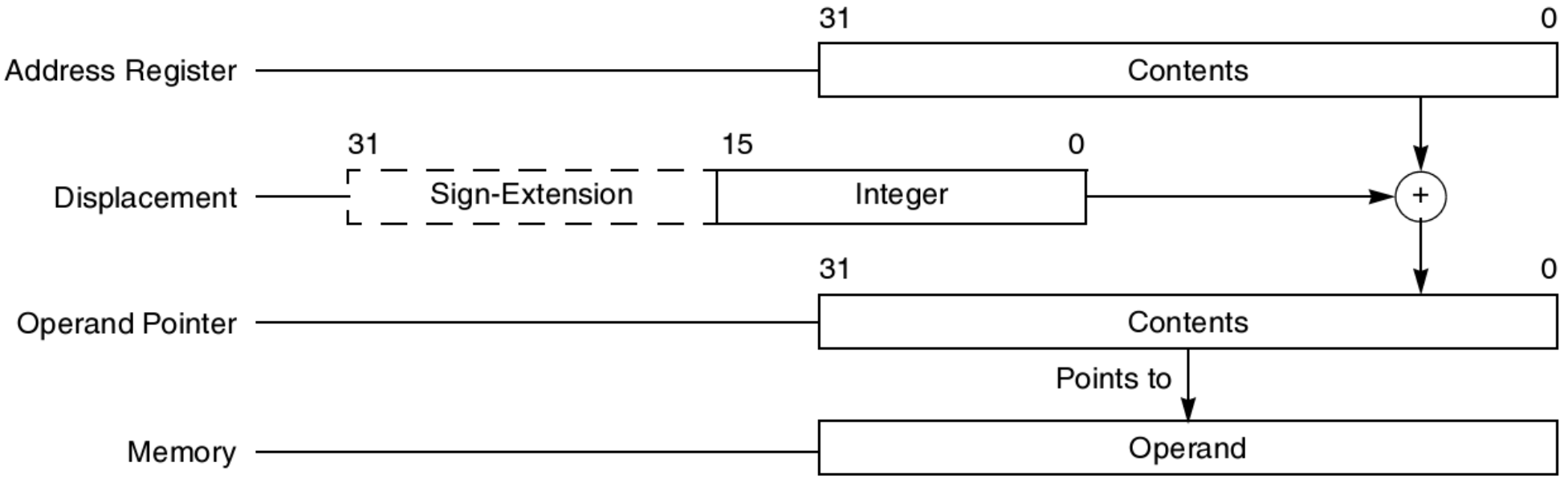



```
MOVE.W    %D0, -(%A7)     | zmniejszenie A7 o 2, odłożenie zawartości D0.W na stos,

ADD.L     -(%A2), %D0     | zmniejszenie A2 o 4, dodanie zaw. rejestru D0 do zmiennej
                          | znajdującej się w na stosie (pod adresem A2-4 bajty)
                          | oraz zapisanie wyniku w D0
```

Adresowanie pośrednie rejestrowe z przesunięciem (1)

(Address Register Indirect with Displacement Mode)

Generation	EA = (An) + d_{16}
Assembler Syntax	(d_{16},An)
EA Mode Field	101
EA Register Field	Register number
Number of Extension Words	1



Address Register Indirect with Displacement

Adresowanie pośrednie rejestrowe z przesunięciem (2)


```
MOVE.W    12(%A0), %D0        | wpisuje 2 bajty znajdującę się pod adresem
                              | (A0+12) do rejestru D0

MOVE.L    %A2, -0x10(%A5)     | przesyła zawartość komórki o adresie podanym a A2 do
                              | do komórki pamięci znajdującej się pod adresem (A5-16)

MOVE.L    0x100, -0x10(%A5)       Error: Operands mismatch – statement
                                  'move.l   0x100, -0x10(%A5)' ignored

MOVE.L    %D0, -0x8001(%A5)       Error: Displ. too large for this architecture; needs 68020
                                  'move.l   %D0, -0x8001(%A5)' ignored
```

Maksymalnie 16-bit ze znakiem, co umożliwia zaadresowanie ±32 kb pamięci od adresu bazowego

Adresowanie pośrednie rejestrowe z przesunięciem (3)

Operation: Source → Destination

Assembler Syntax: MOVE.sz <ea>y,<ea>x

- Destination Effective Address field—Specifies destination location, <ea>x; the table below lists possible data alterable addressing modes. The restrictions on combinations of source and destination addressing modes are listed in the table at the bottom of the next page.

Addressing Mode	Mode	Register
Dx	000	reg. number:Dx
Ax	—	—
(Ax)	010	reg. number:Ax
(Ax) +	011	reg. number:Ax
– (Ax)	100	reg. number:Ax
(d_{16},Ax)	101	reg. number:Ax
(d_8,Ax,Xi)	110	reg. number:Ax

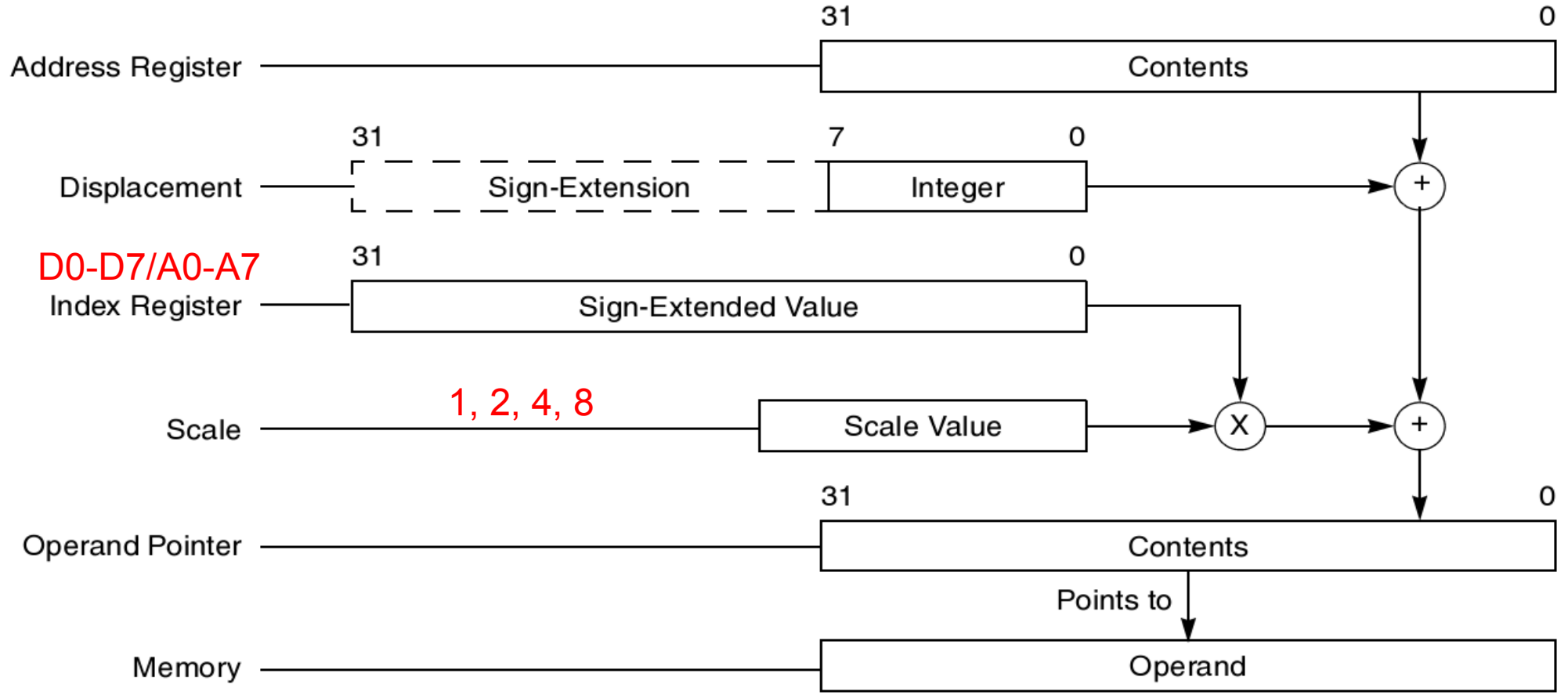
Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

Source Addressing Mode	Destination Addressing Mode
Dy, Ay, (Ay), (Ay)+,-(Ay)	All possible
(d_{16}, Ay), (d16, PC)	All possible except (d_8, Ax, Xi), (xxx).W, (xxx).L
(d8, Ay, Xi), (d8, PC, Xi), (xxx).W, (xxx).L, #<xxx>	All possible except (d_{16}, Ax), (d_8, Ax, Xi), (xxx).W, (xxx).L

Adresowanie pośrednie rejestrowe z indeksem oraz przesunięciem (1)

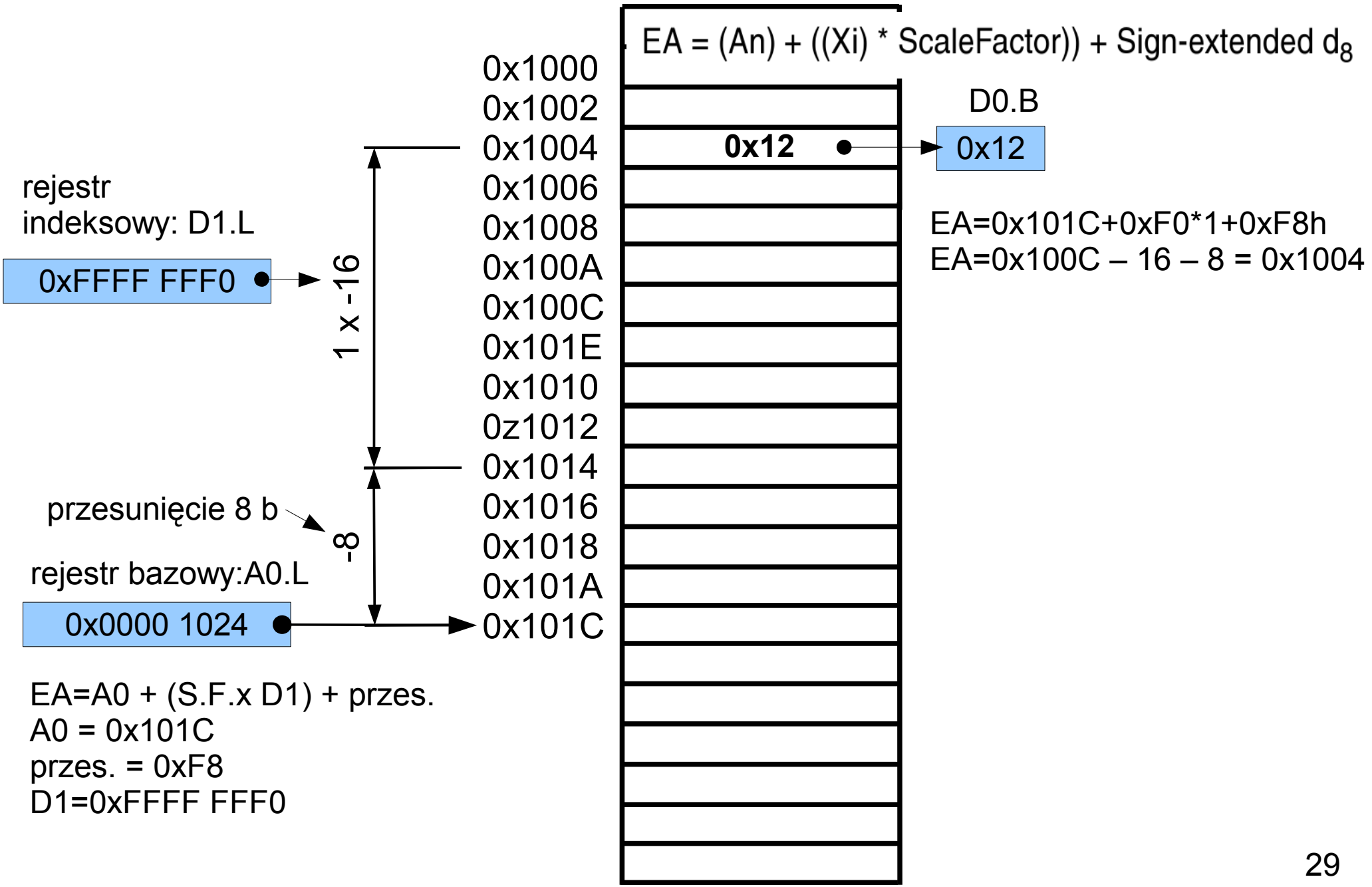
(Address Register Indirect with Index and displ. Mode)

Generation	EA = (An) + ((Xi) * ScaleFactor)) + Sign-extended d_8
Assembler Syntax	(d_8,An,Xi,Size*Scale)
EA Mode Field	110
EA Register Field	Register number
Number of Extension Words	1

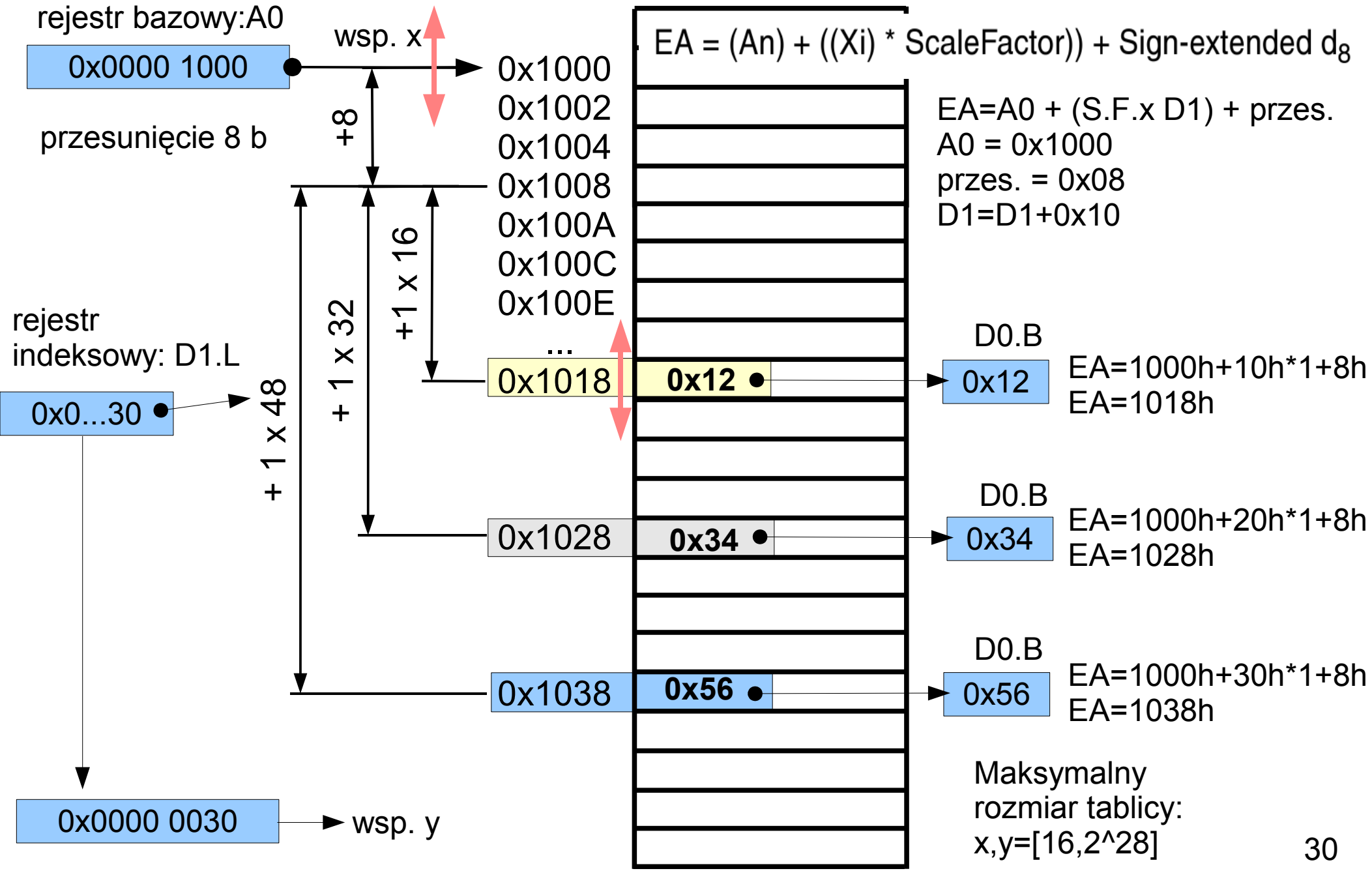


Address Register Indirect with Scaled Index and 8-Bit Displacement

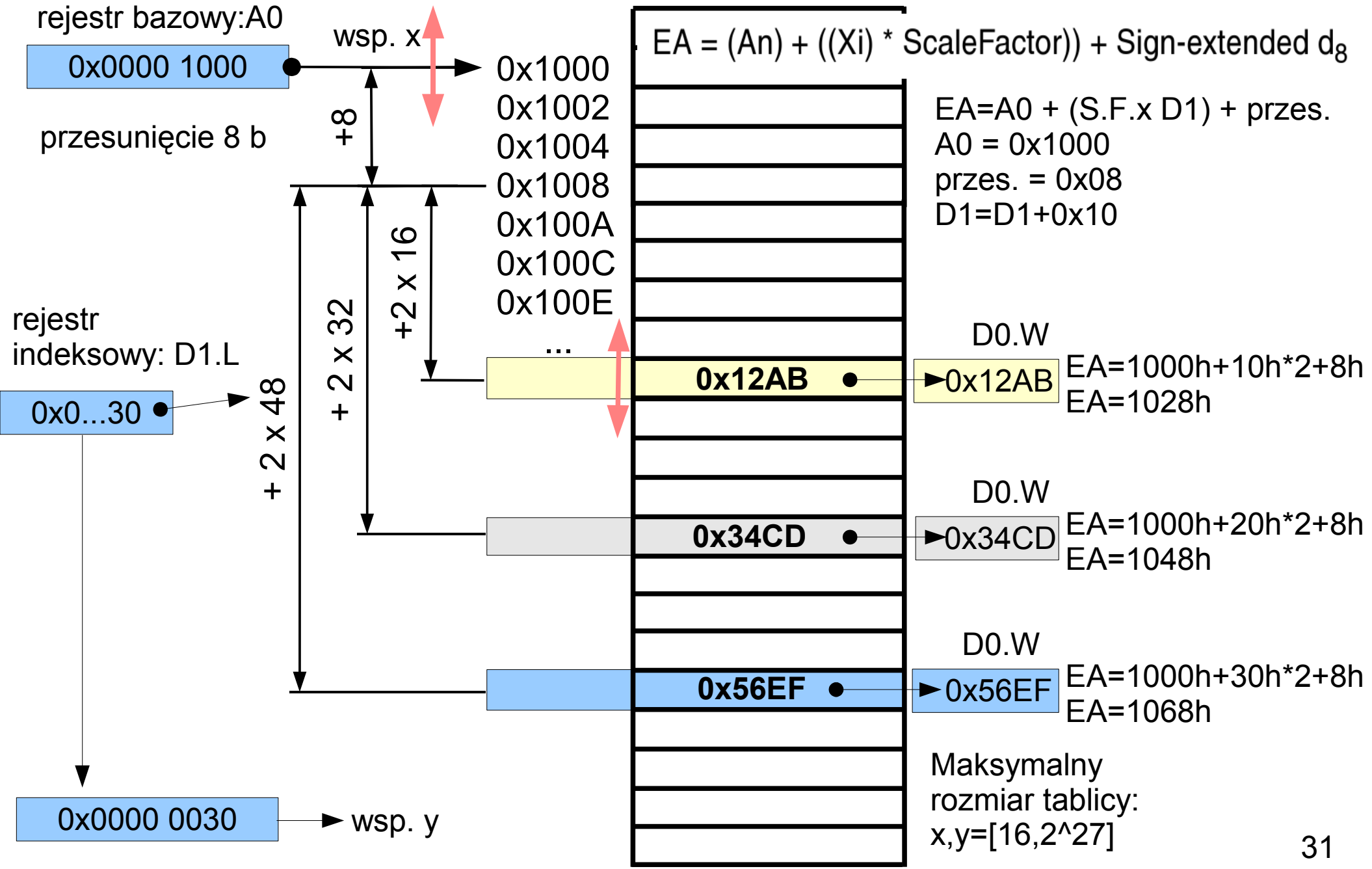
Adresowanie pośrednie rejestrowe z indeksem oraz przesunięciem (2)



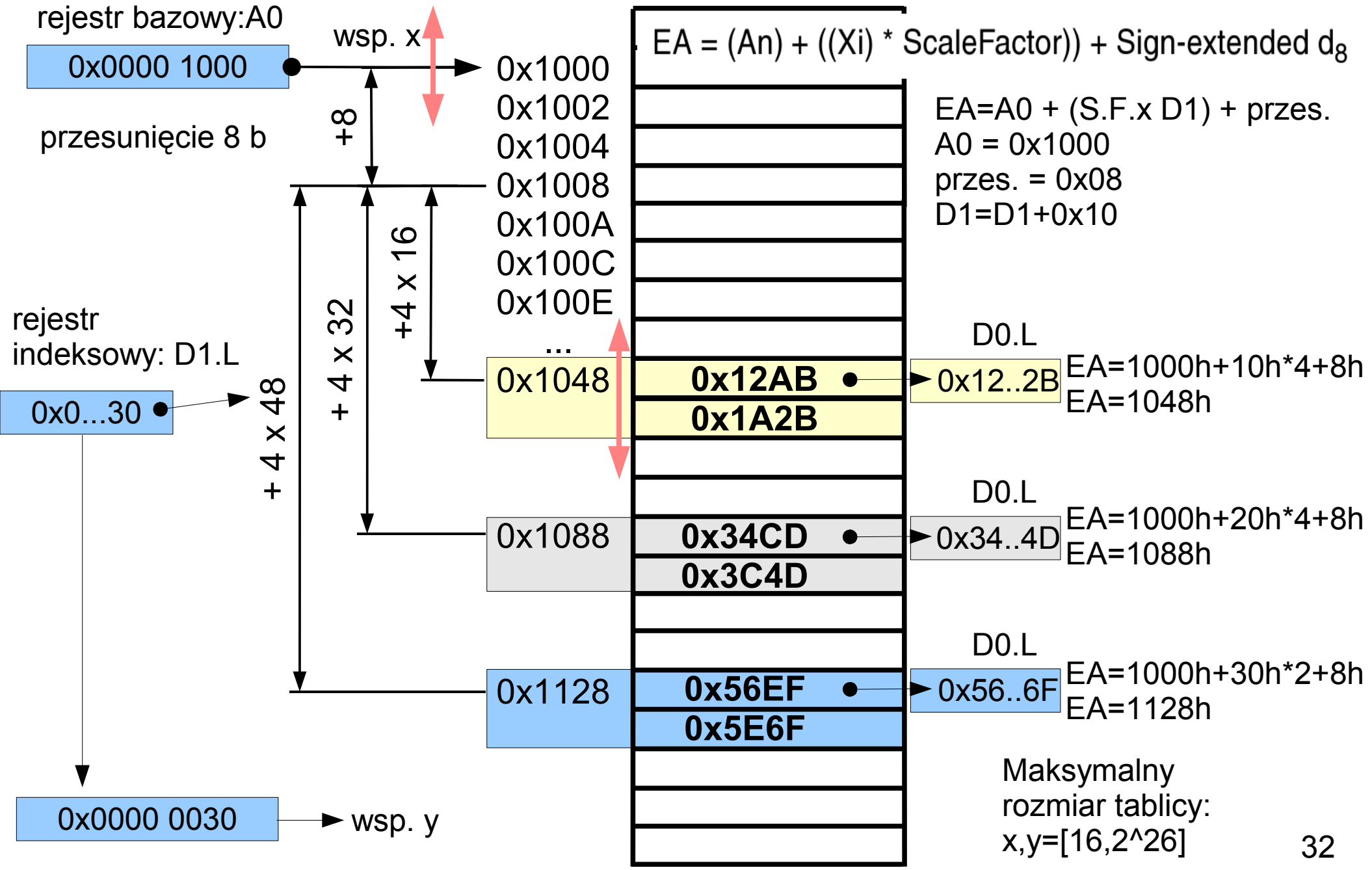
Adresowanie pośrednie rejestrowe z indeksem oraz przesunięciem (3)



Adresowanie pośrednie rejestrowe z indeksem oraz przesunięciem (4)



Adresowanie pośrednie rejestrowe z indeksem oraz przesunięciem (5)



Format instrukcji...

Operation Word Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	Effective Address					
										Mode			Register		

Extension Word Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	Register			W/L	Scale		0	Displacement							

Field	Definition
Instruction	
Mode	Addressing mode (see Table 2-3)
Register	General register number (see Table 2-3)
Extensions	
D/A	Index register type 0 = D*n* 1 = A*n*
W/L	Word/longword index size 0 = Sign-Extended Word 1 = Long Word
Scale	Scale factor 00 = 1 01 = 2 10 = 4 11 = 8 (supported only if FPU is present)

Adresowanie pośrednie rejestrowe z indeksem oraz przesunięciem (6)

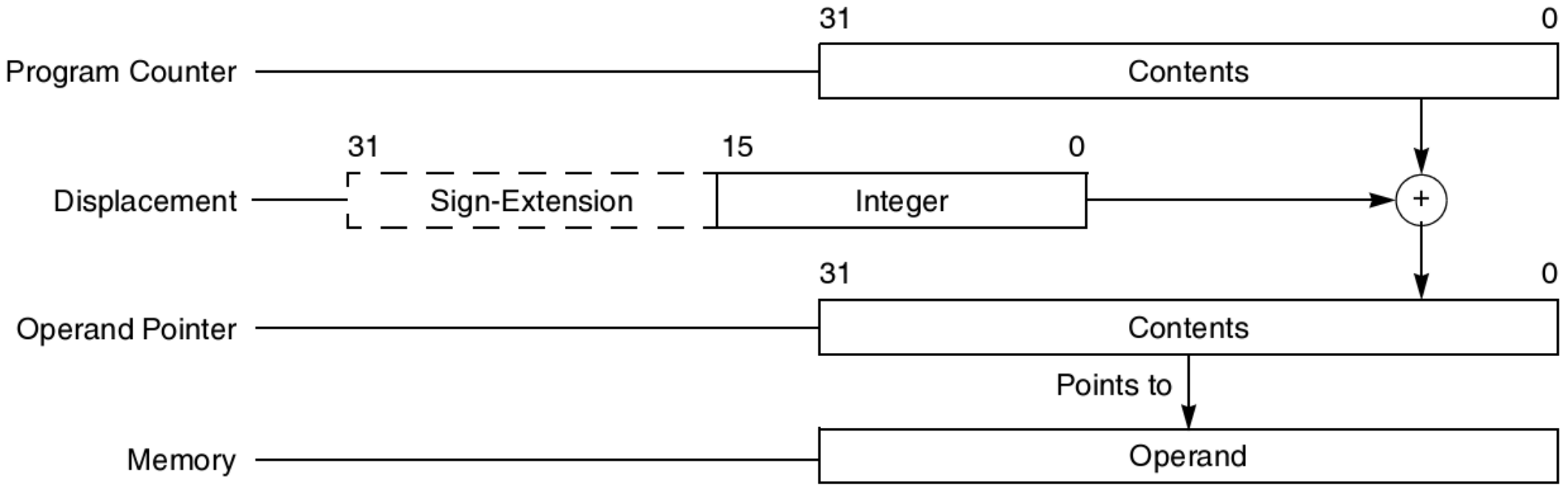
```
MOVE.L    0x0000 1000, %A0      | ustaw rejestr bazowy A0 na początek tablicy
MOVE.L    #0x20, %D1            | ustaw rejestr indeksowy (32)

MOVE.B    8(%A0, %D1*2), %D0    | wpisuje 1 bajt znajdujący się pod adresem
                                | (A0+8+D1*2) = 0x1048 do rejestru D0

MOVE.L    0x0000 101C, %A0      | ustaw rejestr bazowy A0 na początek tablicy
MOVE.L    #0xFFFFFFF0, %D1      | ustaw rejestr indeksowy (-16)
MOVE.L    -16, %D1              | ustaw rejestr indeksowy (-16)

MOVE.B    -8(%A0, %D1*1), %D0   | wpisuje 1 bajt znajdujący się pod adresem
                                | (A0-8+D1*1) do rejestru D0

MOVE.W    -8(%A0, %D1*2), %D0   | wpisuje 2 bajty znajdującę się pod adresem
                                | (A0-8+D1*2) do rejestru D0

MOVE.L    -8(%A0, %D1*4), %D0   | wpisuje 4 bajty znajdującę się pod adresem
                                | (A0-8+D1*4) do rejestru D0

MOVE  .L  -8(%A0, %D1*8), %D0   | Error: scale factor invalid on this architecture;
needa cpu32 or 68020 or higher – statment 'move -8(%A0, %D1*8), %D0   ignored
```

Adresowanie pośrednie licznikiem programu z przesunięciem (1)

(Program Counter Indirect with Displacement Mode)

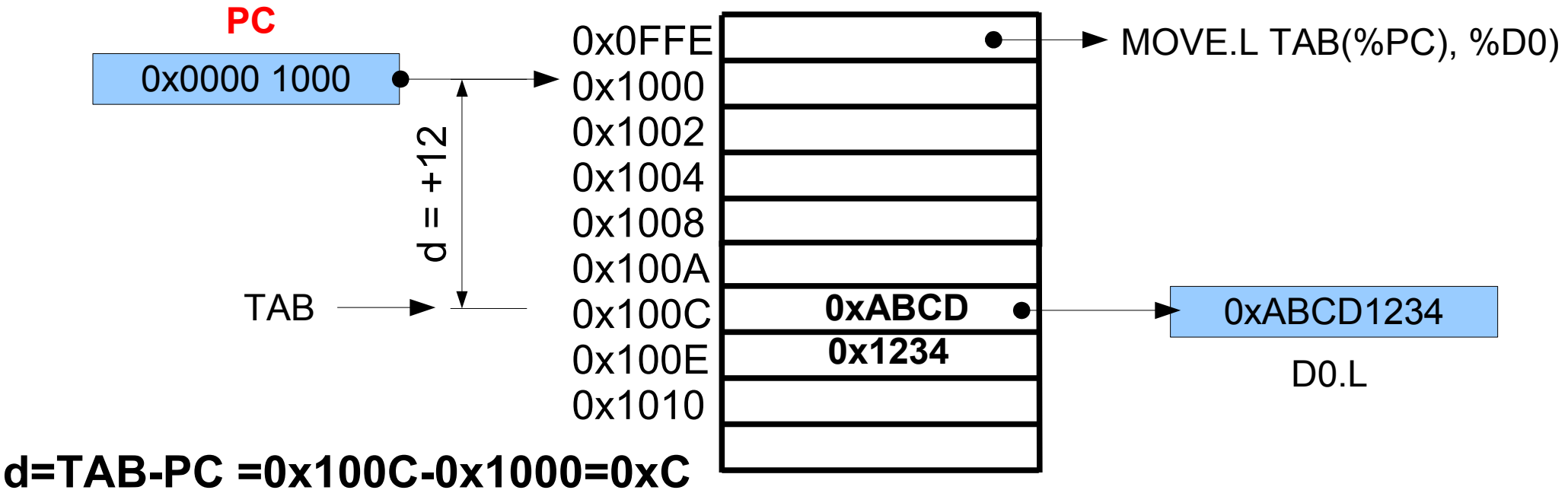
Generation	EA = (PC) + d_{16}
Assembler Syntax	(d_{16},PC)
EA Mode Field	111
EA Register Field	010
Number of Extension Words	1

Programy relokowalne PIC
(ang. position independent code)



Program Counter Indirect with Displacement

Adresowanie pośrednie licznikiem programu z przesunięciem (2)

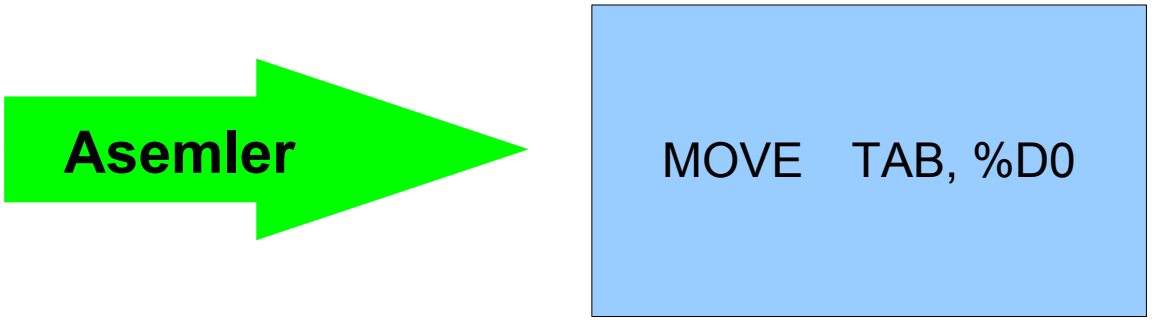


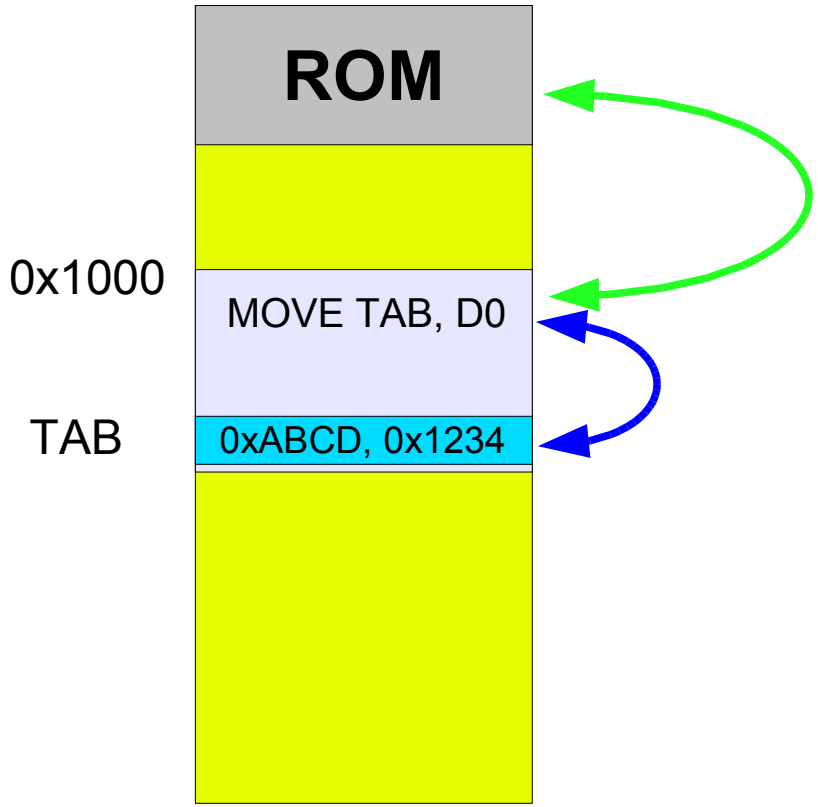
d=TAB-PC =0x100C-0x1000=0xC

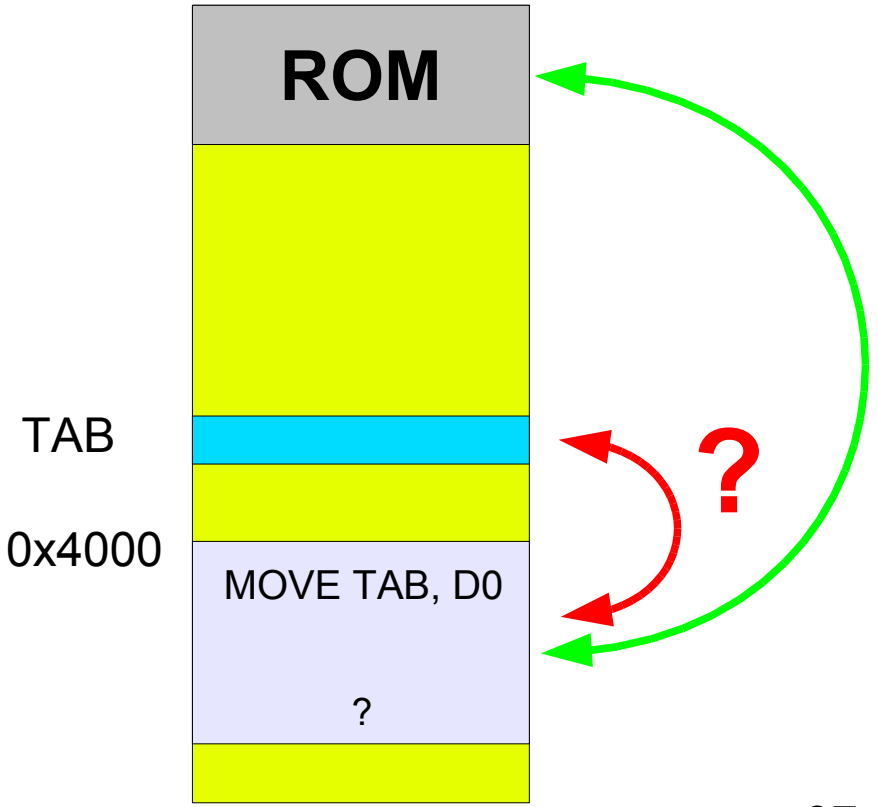
```
SUB1:
        LEA     TAB(%PC), %A0       | zapisz AE tablicy TAB w A0
        MOVE.L  (%A0), %D7          | zapisz w D7.L zawartość TAB
        RTS                         | powrót z podprogramu

TAB:    DC.W 0xABCD, 0x1234

SUB2:
        MOVE.L  TAB(%PC), %D7       | zapisz w D7.L zawartość TAB
        MOVE.L  %D7, TAB(%PC)       | operacja niepoprawna
        RTS                         | powrót z podprogramu
```

Programy relokowalne (1)

```
.org 0x1000

      MOVE TAB, %D0

      RTS

TAB DC.W    0xABCD, 0x1234
```

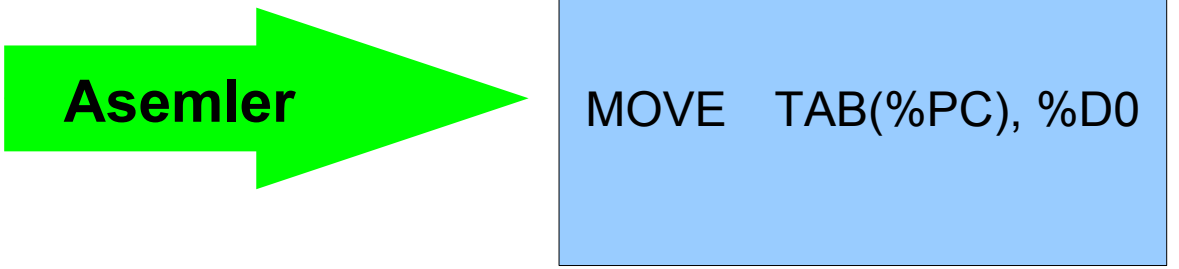


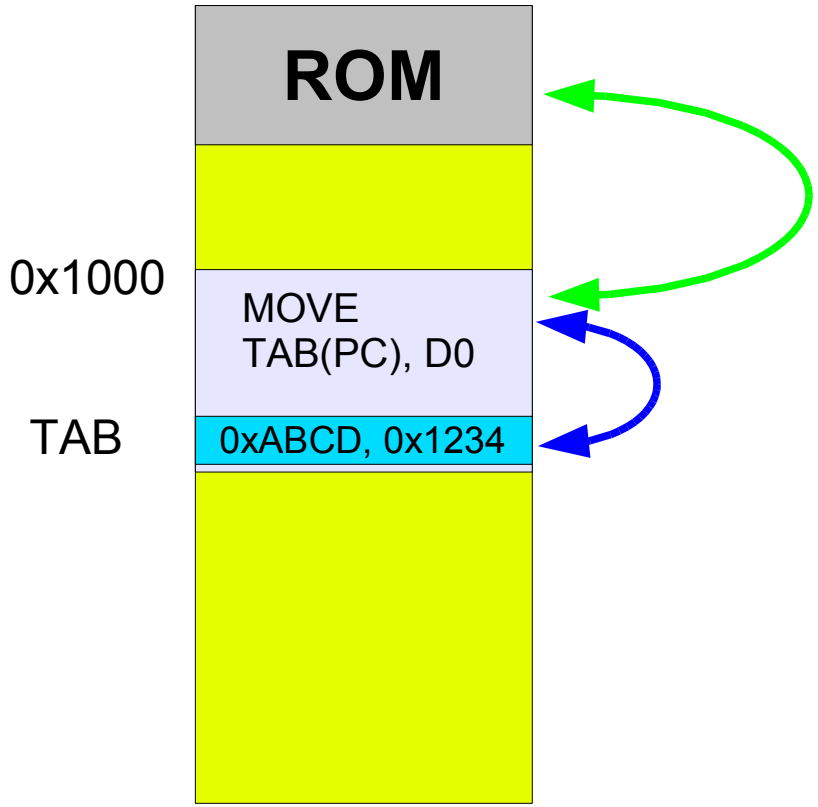


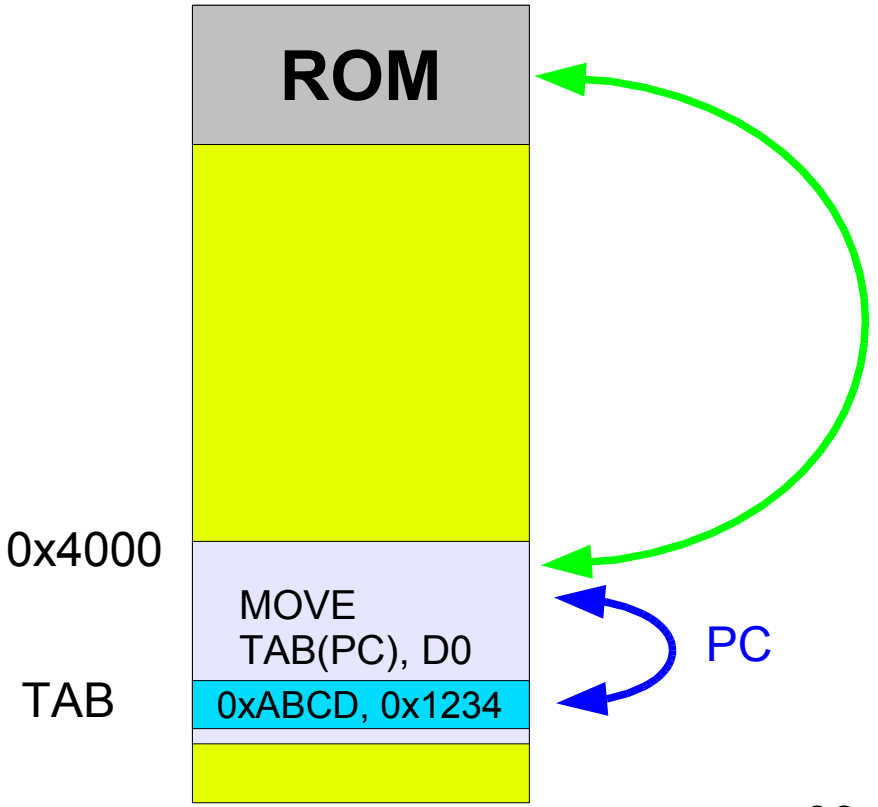


Programy relokowalne (2)

```
.org 0x1000

      MOVE TAB(%PC), %D0

      RTS

TAB DC.W    0xABCD, 0x1234
```



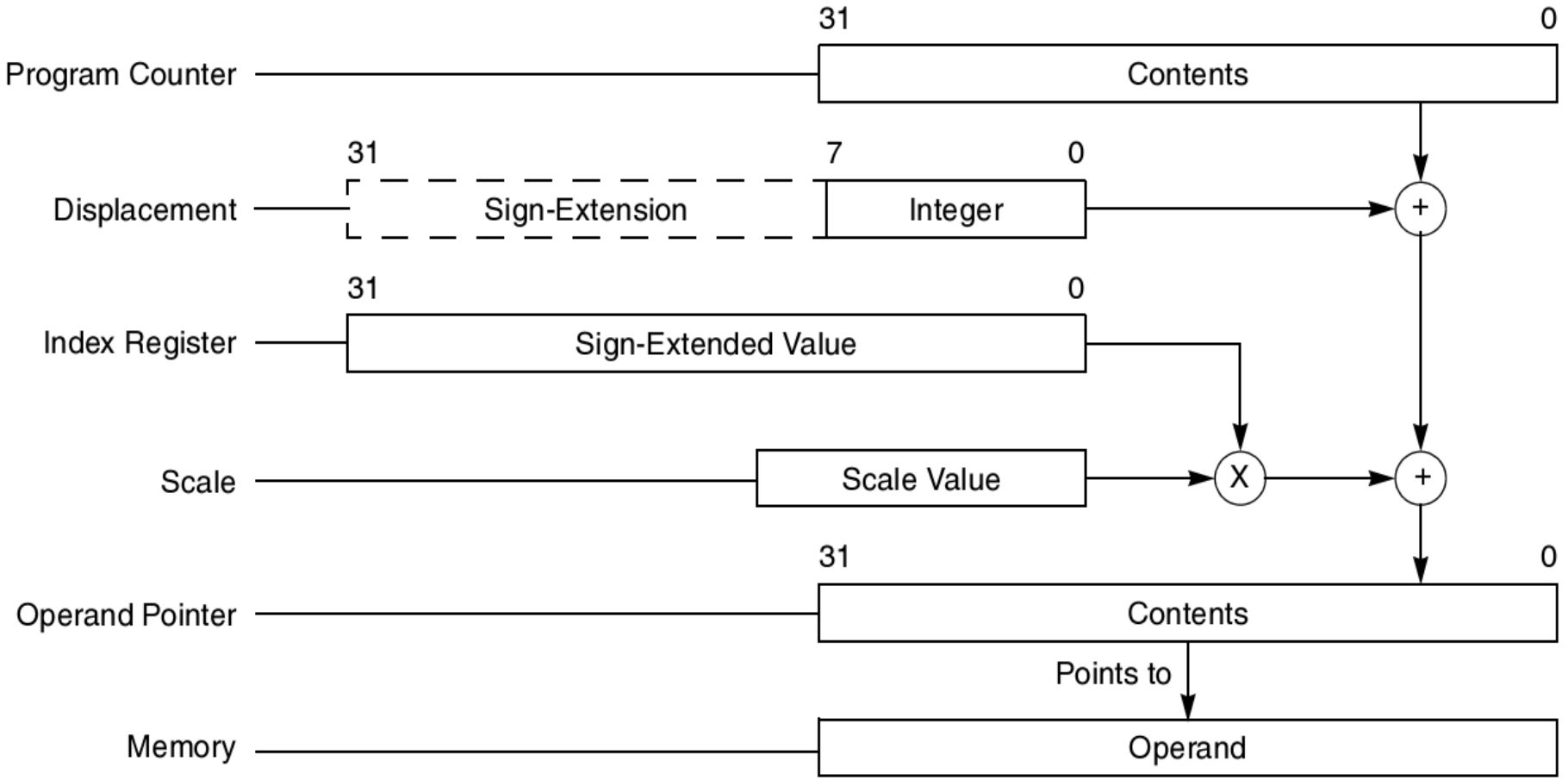




Adresowanie pośrednie licznikiem programu z indeksem oraz przesunięciem (1)

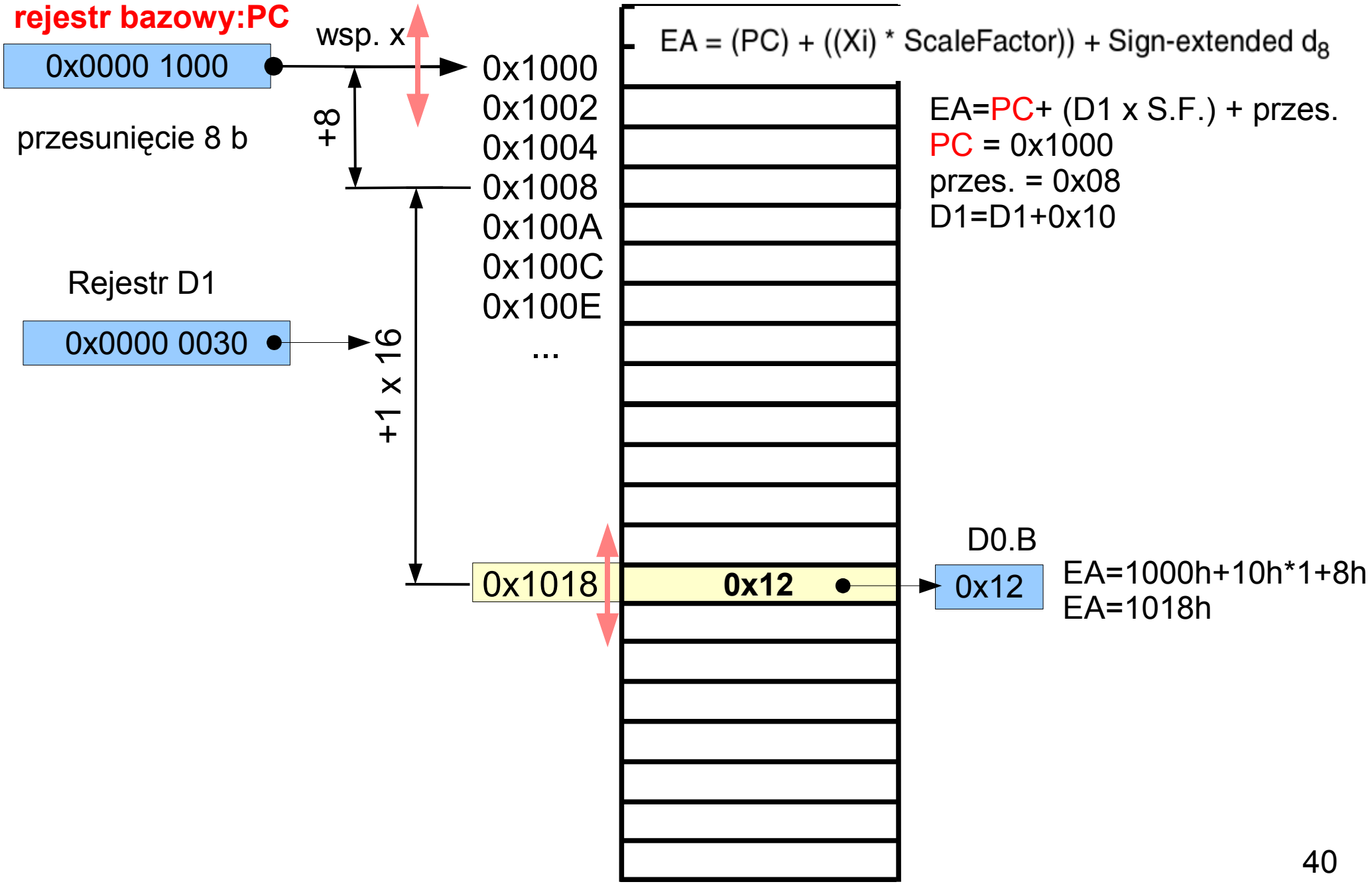
(Program Counter Indirect with Index and displ. Mode)

Generation	EA = (PC) + ((Xi) * ScaleFactor)) + Sign-extended d_8
Assembler Syntax	(d_8,PC,Xi,Size*Scale)
EA Mode Field	111
EA Register Field	011
Number of Extension Words	1



Program Counter Indirect with Scaled Index and 8-Bit Displacement

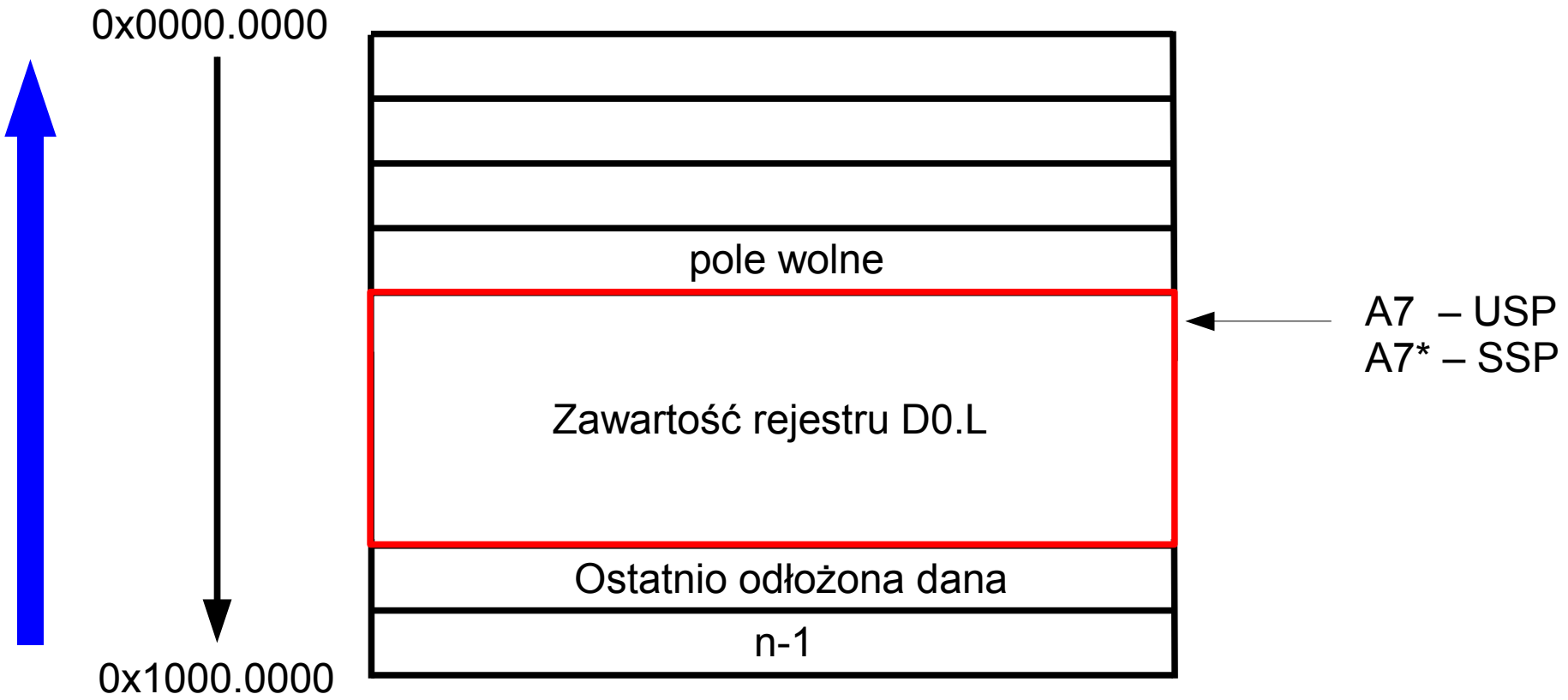
Adresowanie pośrednie licznikiem programu z indeksem oraz przesunięciem (2)



Adresowanie pośrednie licznikiem programu z indeksem oraz przesunięciem (3)

```
SUB2:
        MOVE.L      #0x5, %D1                   | ustaw rejestr indeksowy
        MOVE.B      TAB(%PC, %D1*1), %D0        | wpisuje 1 bajt znajdujący się pod adresem
                                                | (PC+(TAB-PC)+D1*1)  do rejestru D0
        MOVE.L      -16, %D1                    | ustaw rejestr indeksowy (-16)
        MOVE.B      -8(%PC, %D1*1), %D0         | wpisuje 1 bajt znajdujący się pod adresem
                                                | PC-8-D1*1) do rejestru D0

        MOVE.B      %D0, -8(%PC, %D1*1)         | Error: Operands mismatch –
                                                | statement 'move.b  %d0, -8(%pc, %d1*1)'
                                                |  ignored
        RTS
TAB:    DC.B        0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, ...
```

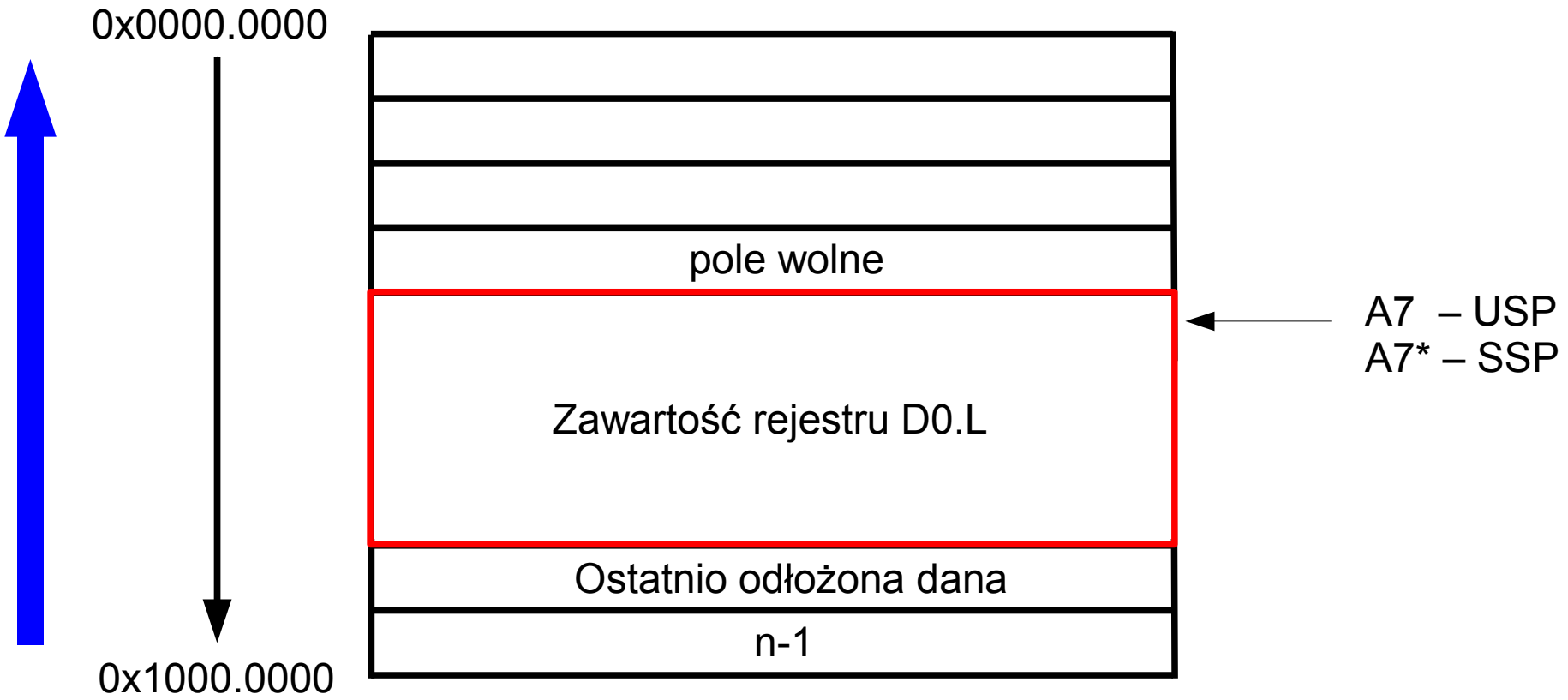
Stos (1)



Odłożenie zmiennej na stos (push)

```
MOVE.L    %D0, -(%A7)    | zmniejszenie A7 o 4, odłożenie zawartości D0.L na stos
```

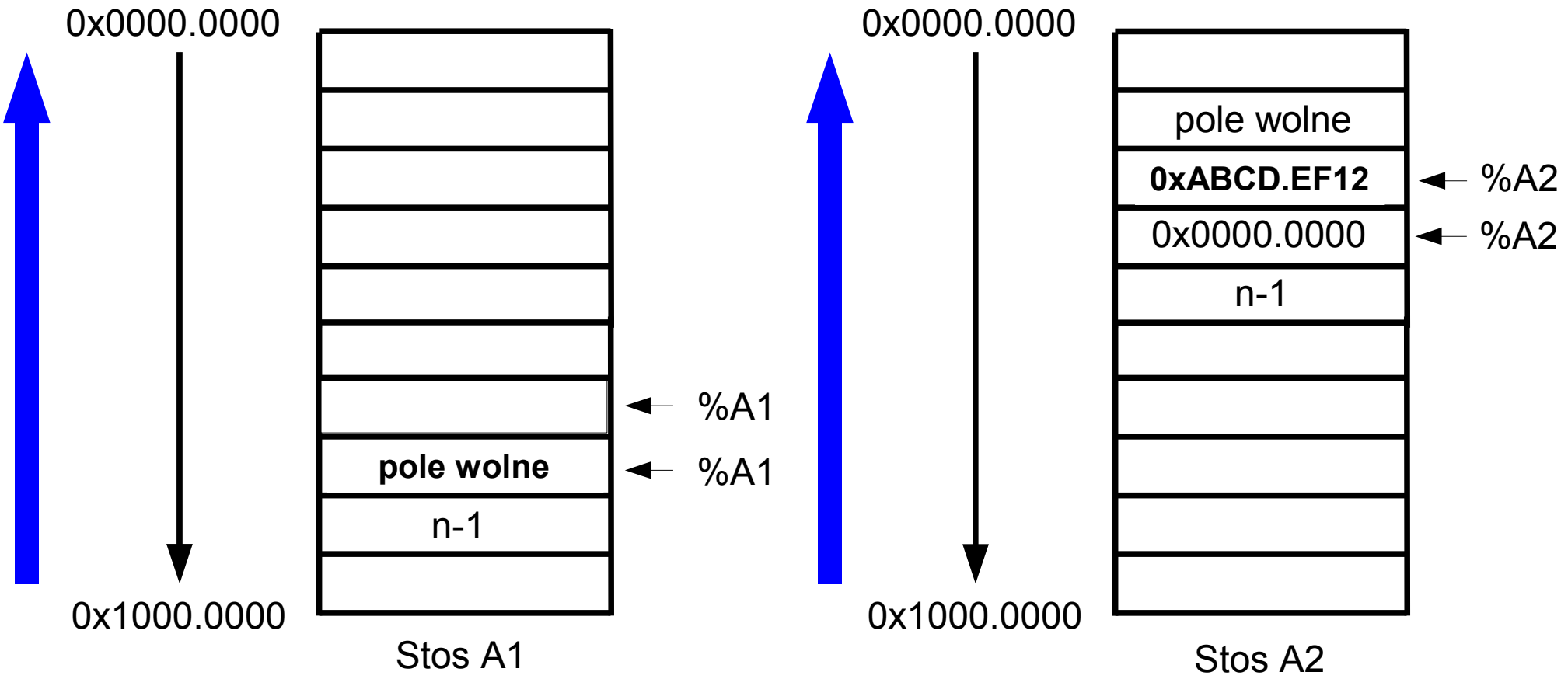
Stos (2)



Zdjęcie zmiennej ze stosu (pull)

```
MOVE.L      (%A7)+, %D0      | zwiększenie A7 o 4, zdjęcie zawartości D0.L ze stosu
```

Stos (3)



Zdjęcie zmiennej typu long ze stosu A1 i odłożenie na stos A2

MOVE.L (%A1)+, -(%A2)

Stos a GDB

```
START:
MOVE.L  #0x11112222,%D0
MOVE.L  #0xD1D1D1D1,%D1
MOVE.L  #0xD2D2D2D2,%D2

BSR     SUBPROGRAM
HALT

SUBPROGRAM: LEA         8(%A7), %A7
            MOVEM.L     D1-D2, %A7
                ...
                ...
                ...
            MOVEM.L     %A7, %D1-D2%
            LEA         8(%A7), %A7
            RTS
```

(gdb) bt 7

```
#0  SUBPROGRAM () at test.asm:35
#1  0xd1d1d1d1 in ?? ()
#2  0xd2d2d2d2 in ?? ()
#3  0x0001001a in START
                        () at test.asm:25
#4  0x00000000 in ?? ()
#5  0x00000000 in ?? ()
#6  0x00000000 in ?? ()
```