Векторизация

Немного о векторных x86-инструкциях

Векторные типы данных x86

Вещественные типы

биты			
127-96	95-64	63-32	31-0
32-бит вещ. число #3	32-бит вещ. число #2	32-бит вещ. число #1	32-бит вещ. число #0
64-бит вещ. число #1		64-бит вещ. число #0	

Целочисленные типы

биты							
127-96		95-64		63-32		31-0	
слово #7	слово #6	слово #5	слово #4	слово #3	слово #2	слово #1	слово #0
двойное слово #3		двойное слово #2		двойное слово #1		двойное слово #0	
учетверённое слово #1				учетверённое слово #0			

Векторные типы данных x86

Суффиксы команд:

	Размер	Суффиксы типов данный
Вещественные	32-bit	ss (скалярный) ps (упакованные)
	64-bit	sd (скалярный) pd (упакованный)
Целочисленные	8-bit	b
	16-bit	w
	32-bit	d
	64-bit	q
	128-bit	dq

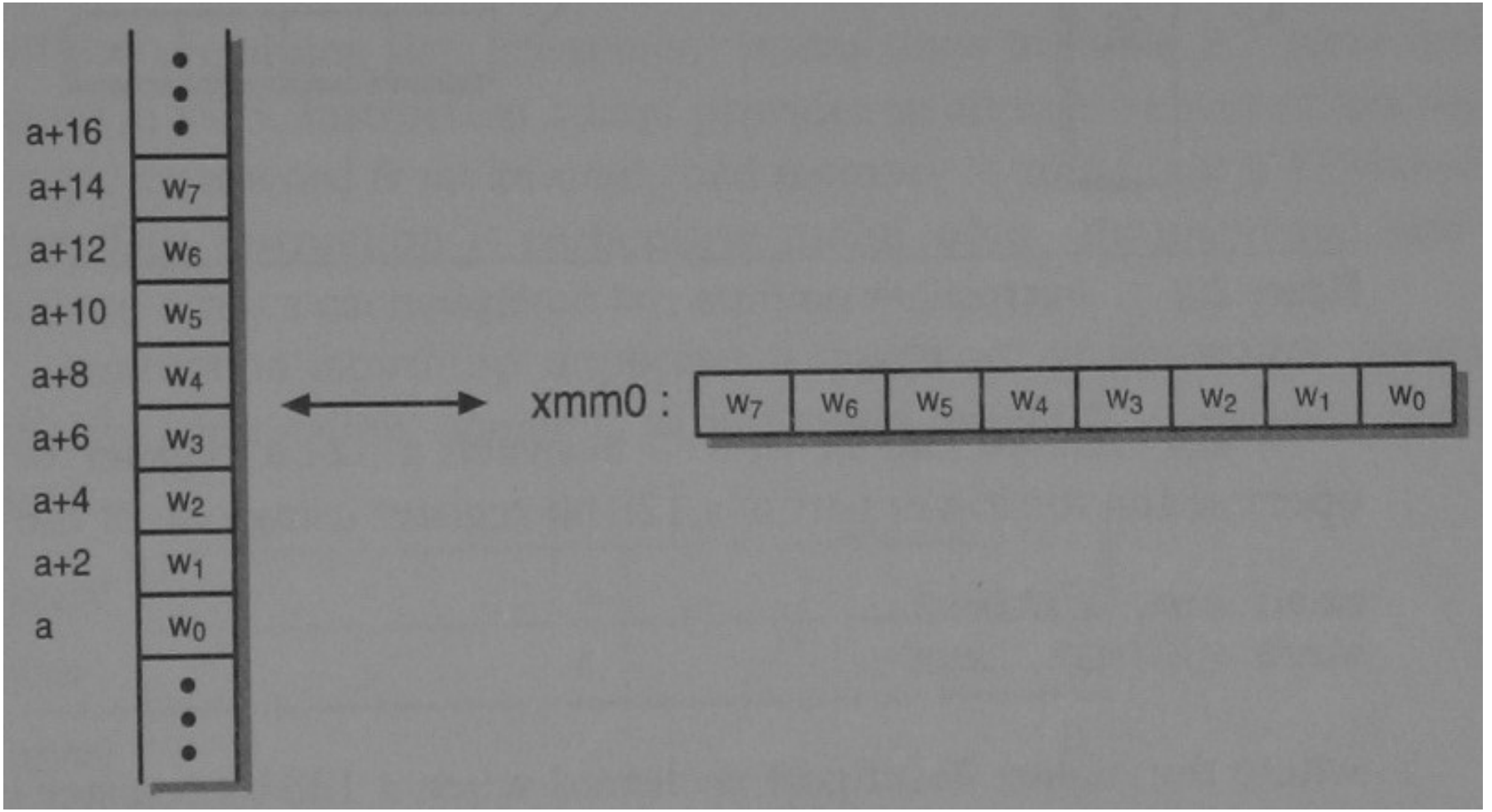
Формат инструкции

<instruction> xmm, xmm/m128

Инструкции пересылки данных #1

Инструкция	Суфикс типов данных	Описание
movdqa movdqu		Move 128-bit integer aligned Move 128-bit integer unaligned
mova movu	[ps, pd] [ps, pd]	Move floating point [32,64] bit align Move floating point [32,64] bit unalign
movhl movlh	[ps] [ps]	Move [32] bit floating point high to low Move [32] bit floating point low to high
movh movl	[ps, pd] [ps, pd]	Move high [32,64] bit floating point Move low [32,64] bit floating point
mov	[d, q, ss, sd]	Move [32-bit integer, 64-bit integer, 32-bit floating point, 64-bit floating point] scalar data
lddqu		Load 128-bit integer unaligned
mov<d/sh/sl>dup		Move and duplicate

Little endian order



Наименее значимые байты располагаются по меньшим адресам

Типы данных и пересылки данных

Что лучше использовать?

- movdqa
- movaps
- movapd

Зависит от того как вы интерпретируюте XMM регистр

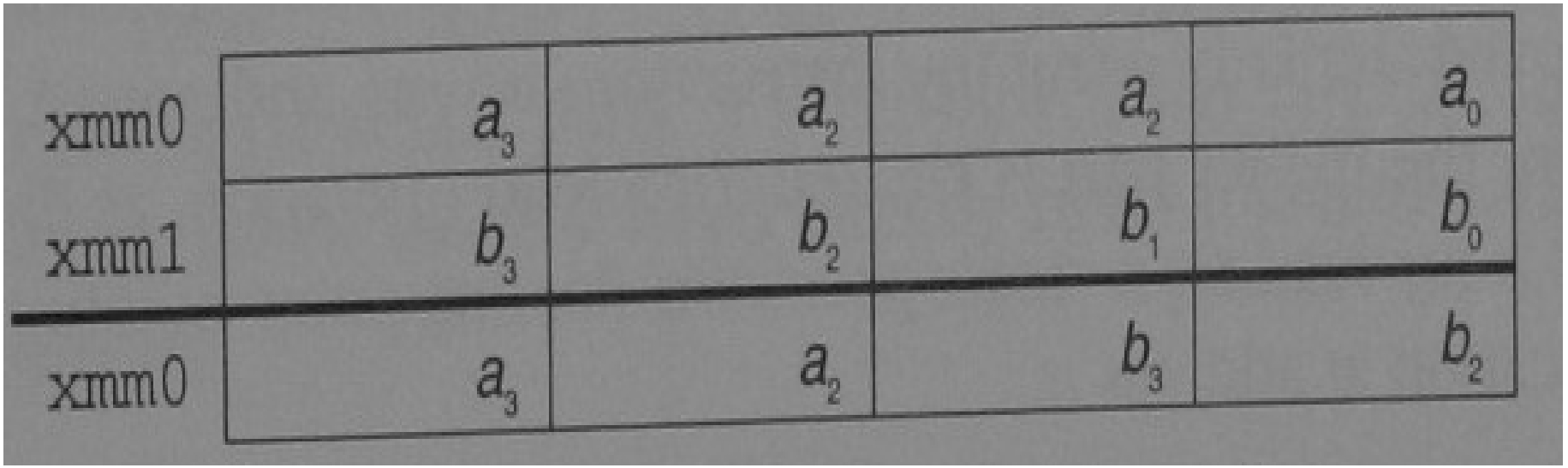
```
for(i=0; i < 128; i++)
{
    x[i] = y[i] + z[i];
}
```

=>

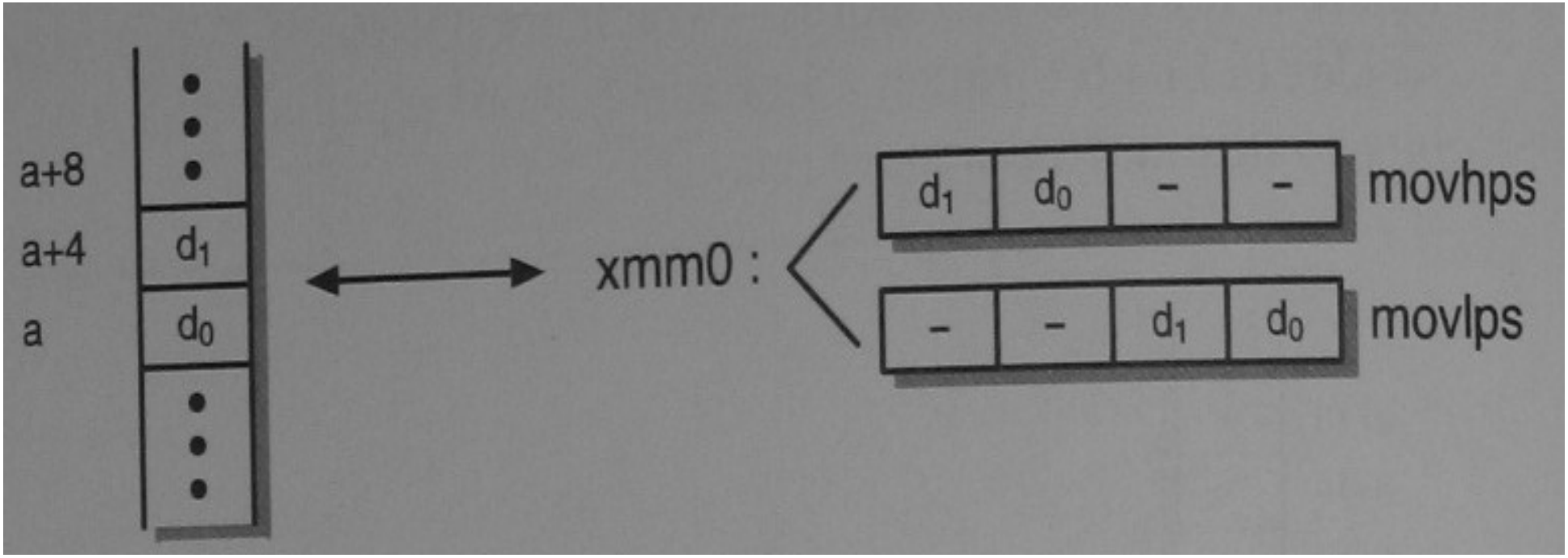
```
L:  movapd  xmm0, y[eax]  ; можно movdqa, но так быстрее
    paddd   xmm0, z[eax]  ; сложение векторов из double
    movapd  x[eax], xmm0
    add     eax, 16
    cmp     eax,  512
    jb      L
```

Пример: movhlps

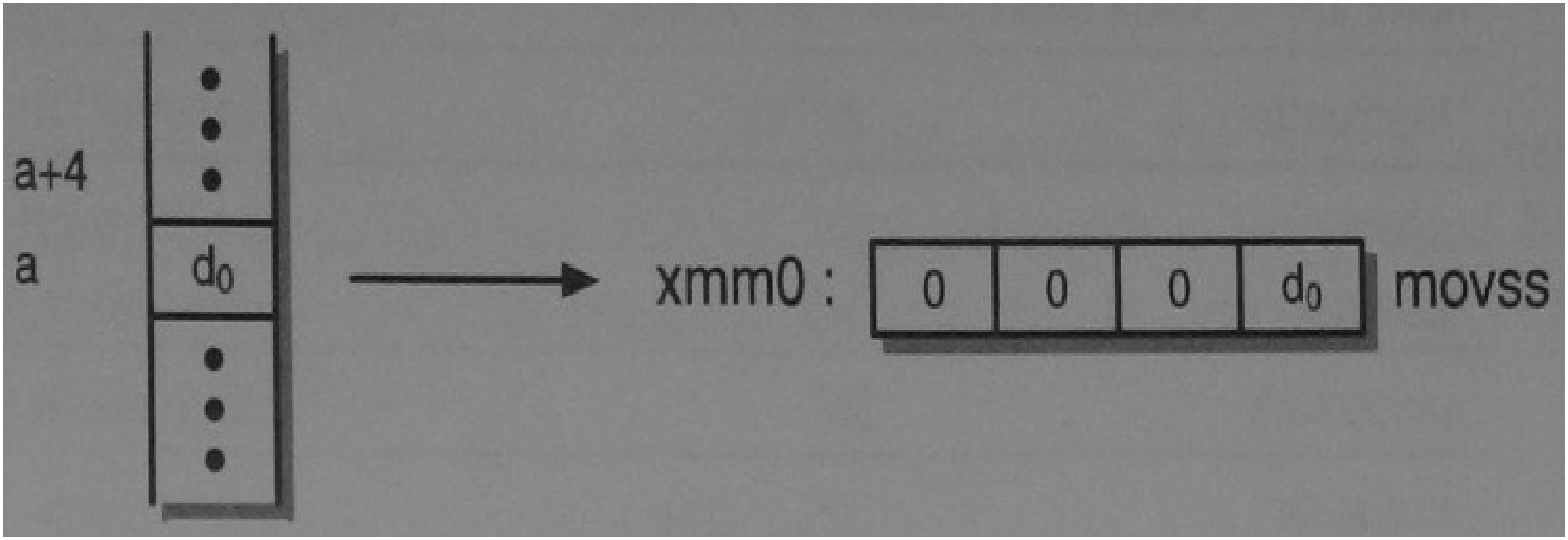
movhlps xmm0, xmm1



Инструкции: movhps/movlps

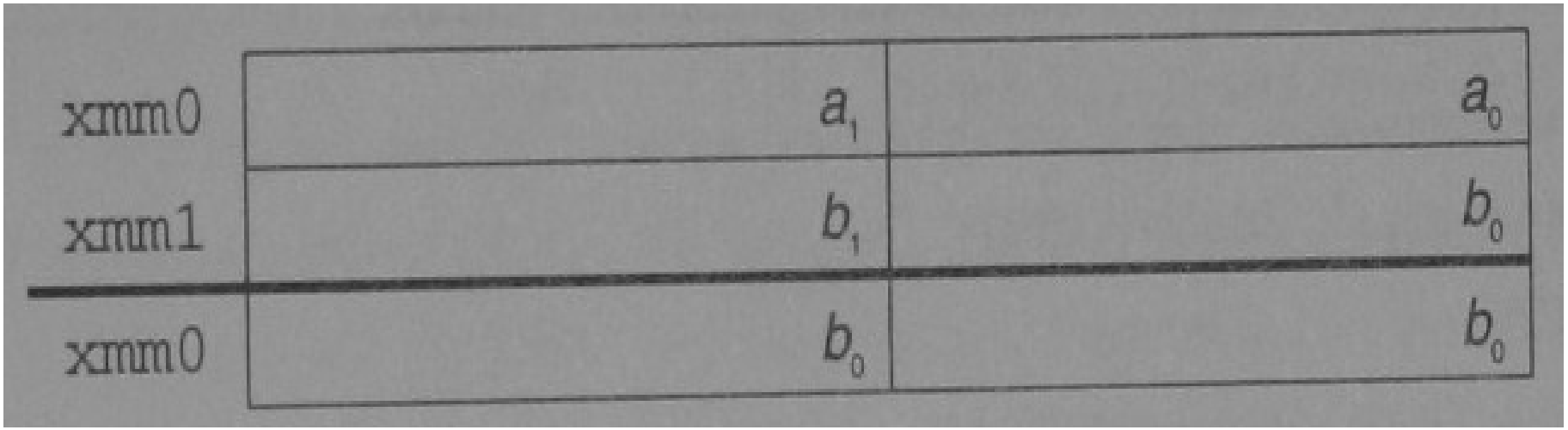


Инструкции: movss



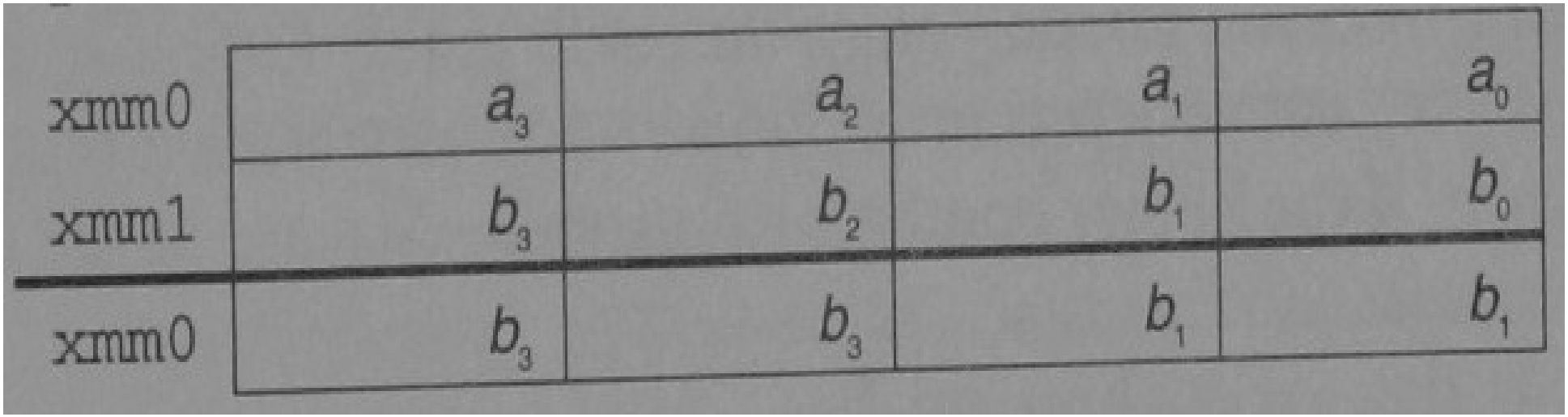
Инструкция: movddup

movddup xmm0, xmm1



Инструкция: movshdup

movshdup xmm0, xmm1

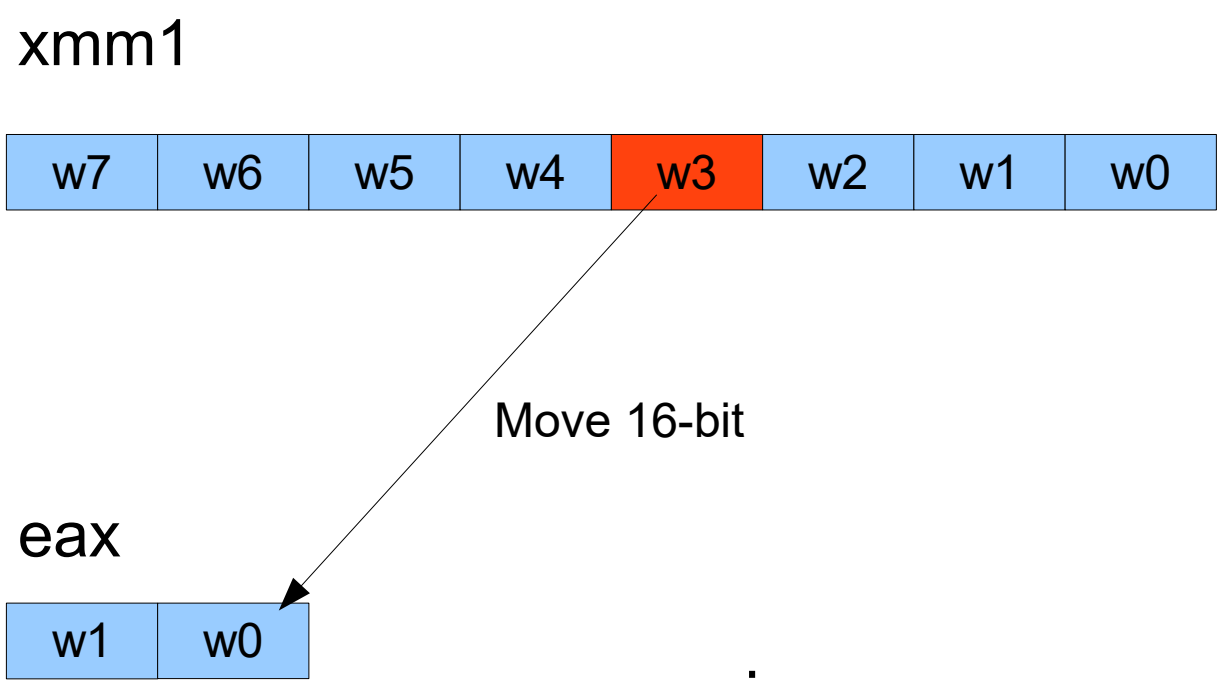


Инструкции пересылки данных #2

Инструкция	Суффикс типа данных	Описание
pextr pinstr	[w] [w]	Извлечь 32-bit целое Вставить 32-bit целое
pmovmsk	[b]	Создать маску из самых значимых бит компонент вектора
movmsk	[ps, pd]	Создать маску из самых значимых бит компонент вектора

Пример: pextrw

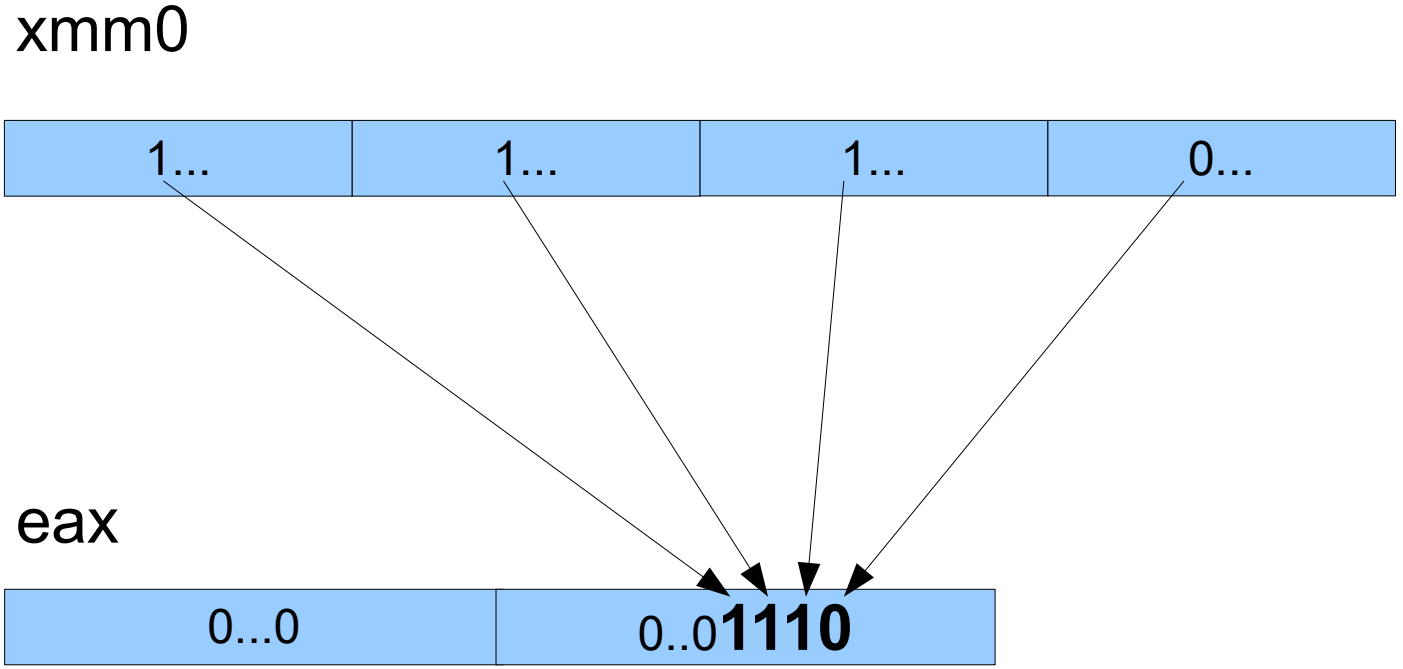
pextrw eax, xmm1, 3



pinsr — аналогичная команда
только в обратную сторону

Пример: movmskps

movmskps eax, xmm0



Аналогично действуют pmovmskb, movmskpd

Арифметические операции #1

Instruction	Suffix	Description
padd	[b,w,d,q]	packed addition (signed and unsigned)
psub	[b,w,d,q]	packed subtraction (signed and unsigned)
padds	[b,w]	packed addition with saturation (signed)
paddus	[b,w]	packed addition with saturation (unsigned)
psubs	[b,w]	packed subtraction with saturation (signed)
psubus	[b,w]	packed subtraction with saturation (unsigned)
pmins	[w]	packed minimum (signed)
pminu	[b]	packed minimum (unsigned)
pmaxs	[w]	packed maximum (signed)
pmaxu	[b]	packed maximum (unsigned)

Арифметические операции #2

Instruction	Suffix	Description
add	[ss,ps,sd,pd]	addition
div	[ss,ps,sd,pd]	division
min	[ss,ps,sd,pd]	minimum
max	[ss,ps,sd,pd]	maximum
mul	[ss,ps,sd,pd]	multiplication
sqrt	[ss,ps,sd,pd]	square root
sub	[ss,ps,sd,pd]	subtraction
rcp	[ss,ps]	approximated reciprocal
rsqrt	[ss,ps]	approximated reciprocal square root

Суффиксы типы данных [ps, pd] vs [ss, sd]

mul**ps** xmm0, xmm1

xmm0	a_3	a_2	a_1	a_0
xmm1	b_3	b_2	b_1	b_0
xmm0	$a_3 \cdot b_3$	$a_2 \cdot b_2$	$a_1 \cdot b_1$	$a_0 \cdot b_0$

mul**ss** xmm0, xmm1

xmm0	a_3	a_2	a_1	a_0
xmm1	b_3	b_2	b_1	b_0
xmm0	a_3	a_2	a_1	$a_0 \cdot b_0$

Идиоматические операции

Instruction	Suffix	Description
pavg	[b,w]	packed average with rounding (unsigned)
pmulh	[w]	packed multiplication high (signed)
pmulhu	[w]	packed multiplication high (unsigned)
pmull	[w]	packed multiplication low (signed and unsigned)
psad	[bw]	packed sum of absolute differences (unsigned)
pmadd	[wd]	packed multiplication and addition (signed)
addsub	[ps,pd]	floating-point addition/subtraction
hadd	[ps,pd]	floating-point horizontal addition
hsub	[ps,pd]	floating-point horizontal subtraction

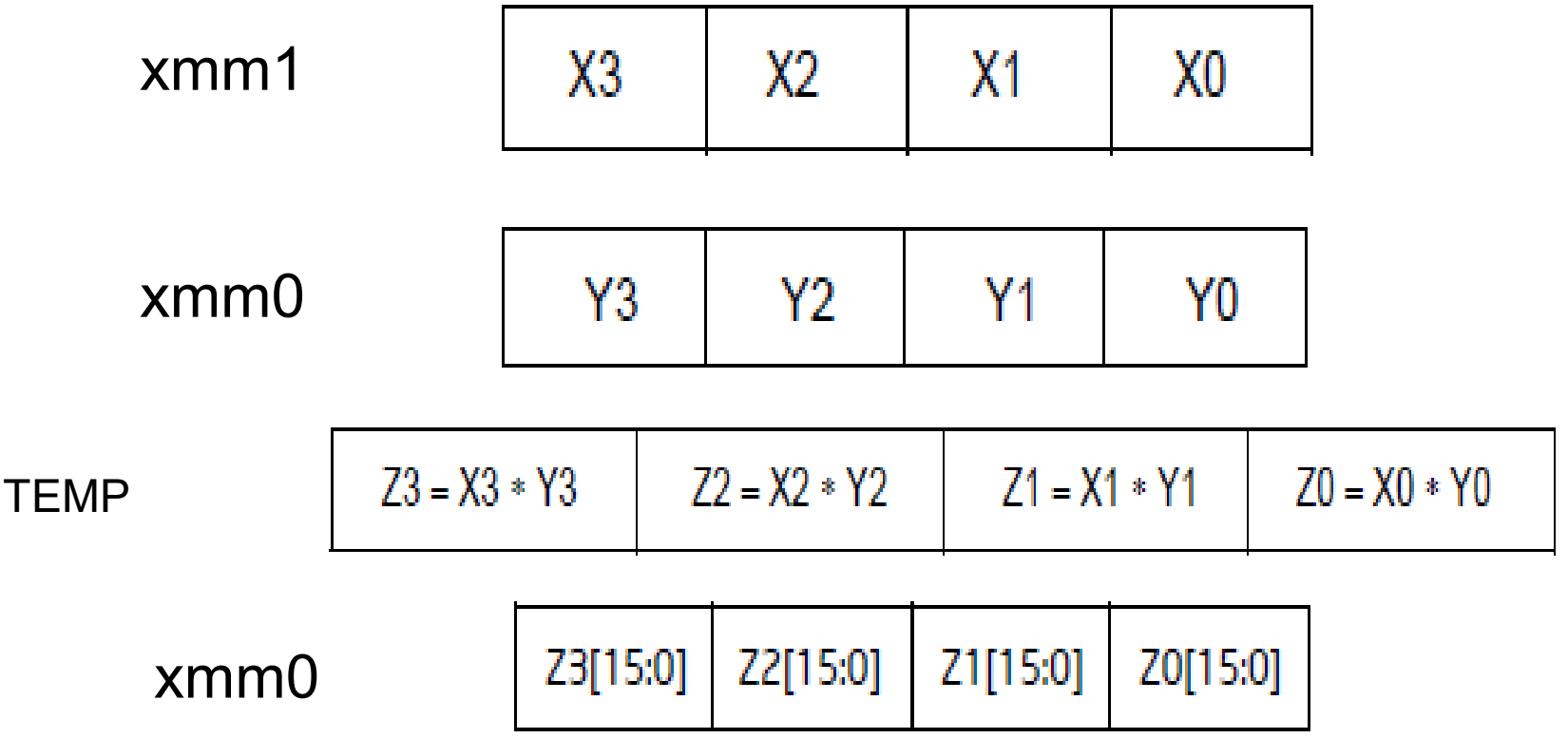
Инструкции: pavg[b,w]

pavg xmm0, xmm1

$$a_i := (a_i + b_i + 1) >> 1$$

Инструкция: pmullw

pmullw xmm0, xmm1



Логические операции

Instruction	Suffix	Description
pand		bitwise logical AND
pandn		bitwise logical AND-NOT
por		bitwise logical OR
pxor		bitwise logical XOR
and	[ps,pd]	bitwise logical AND
andn	[ps,pd]	bitwise logical AND-NOT
or	[ps,pd]	bitwise logical OR
xor	[ps,pd]	bitwise logical XOR

Операции сравнения

Instruction	Suffix	Description
pcmp<cc>	[b,w,d]	packed compare
cmp<cc>	[ss,ps,sd,pd]	floating-point compare

Пример: pcmpeqd

pcmpeqd xmm0, xmm1

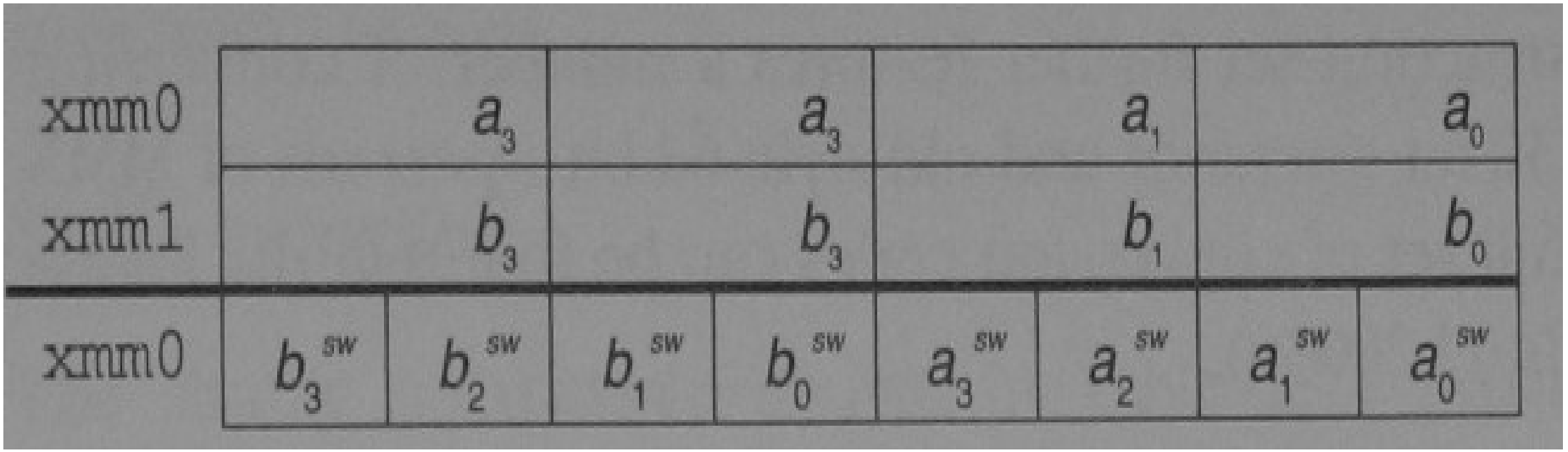
xmm0	11	0	105	110
xmm1	11	20	100	334
xmm0	0xffffffff	0x00000000	0x00000000	0x00000000

Операции преобразования

Instruction	Suffix	Description
packss	[wb, dw]	pack with saturation (signed)
packus	[wb]	pack with saturation (unsigned)
cvt<s2d>		conversion
cvtt<s2d>		conversion with truncation

Пример: packssdw

packssdw xmm0, xmm1



d^{sw} - сатурированное значение

Сдвиговые операции

Instruction	Suffix	Description
psll	[w,d,q,dq]	shift left logical (zero in)
psra	[w,d]	shift right arithmetic (sign in)
psrl	[w,d,q,dq]	shift right logical (zero in)

Инструкции перемешивания

Перестановка компонент вектора

Instruction	Suffix	Description
`pshuf`	[w,d]	packed shuffle
`pshufh`	[w]	packed shuffle high
`pshufl`	[w]	packed shuffle low
`shuf`	[ps,pd]	shuffle

Пример: pshufd

Общий формат:

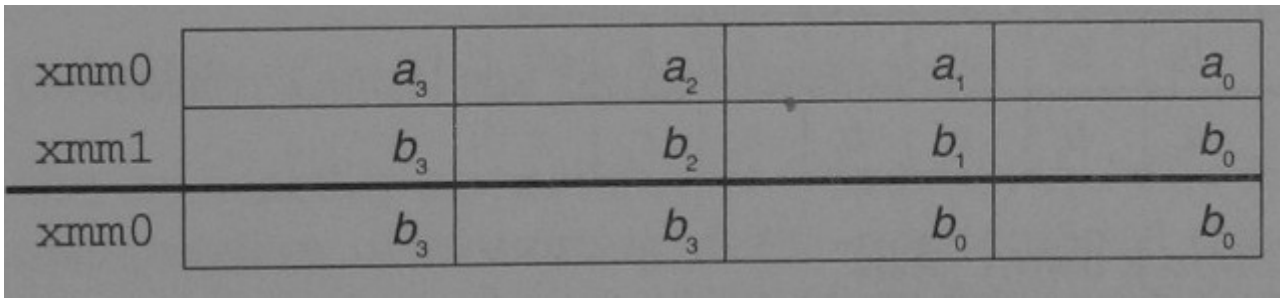
pshufd xmm, xmm/m128, imm8

Пример:

pshufd xmm0, xmm1, 240

240 = 11110000b

биты 2 * i и 2 * i + 1 в imm8 представляют собой 2-битовый номер двойного слова в операнде-источнике, который сохраняется в i-ой позиции операнда-приёмника

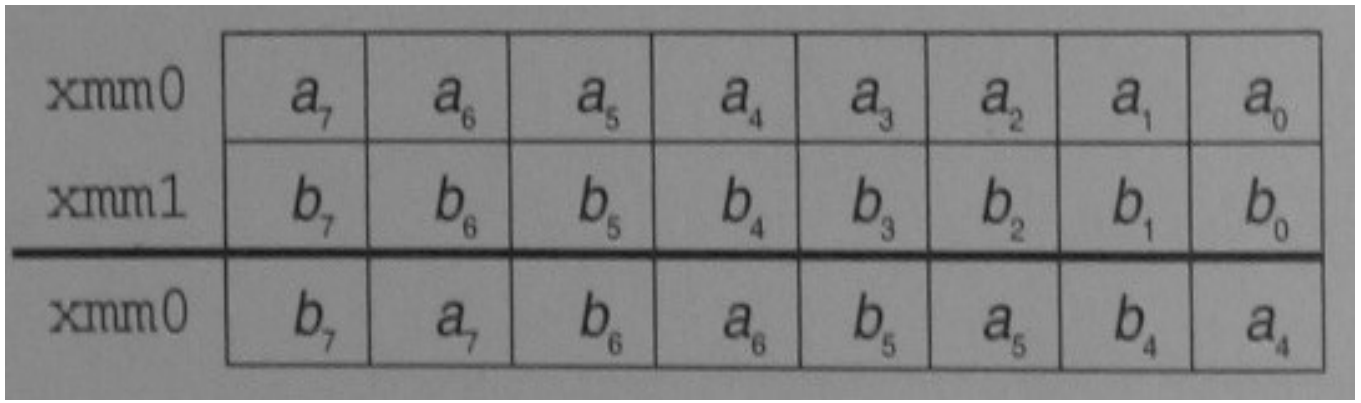


Инструкции чередования

Instruction	Suffix	Description
punpckh	[bw, wd, dq, qdq]	unpack high
punpckl	[bw, wd, dq, qdq]	unpack low
unpckh	[ps, pd]	unpack high
unpckl	[ps, pd]	unpack low

Пример: punpckhwd

punpckhwd xmm0, xmm1



Пример: punpckhwd

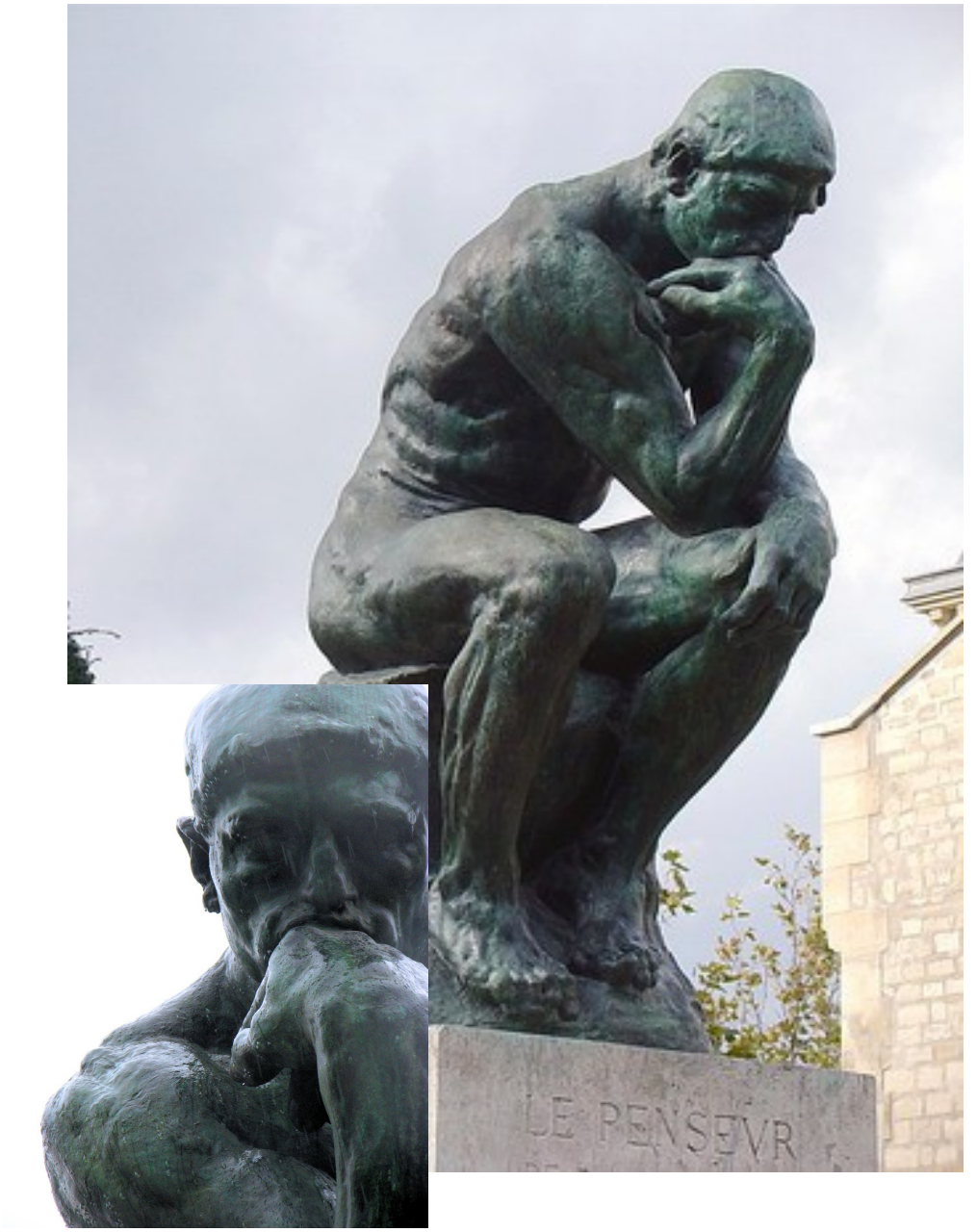
punpckhwd xmm0, xmm1

xmm0	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
xmm1	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
xmm0	b_7	a_7	b_6	a_6	b_5	a_5	b_4	a_4

Взаимосвязь перемешивания и чередования

```
shufps xmm0, xmm1, 68   ≡  movlhps  xmm0, xmm1
shufps xmm0, xmm0, 80   ≡  unpcklps xmm0, xmm0
shufps xmm0, xmm0, 160  ≡  movsldup xmm0, xmm0
shufps xmm0, xmm0, 230  ≡  movshdup xmm0, xmm0
shufps xmm0, xmm0, 238  ≡  movhlps  xmm0, xmm0
shufps xmm0, xmm0, 250  ≡  unpckhps xmm0, xmm0
shufpd xmm0, xmm0, 0    ≡  movddup  xmm0, xmm0
shufpd xmm0, xmm1, 0    ≡  unpcklpd xmm0, xmm1
shufpd xmm0, xmm1, 3    ≡  unpckhpd xmm0, xmm1
```

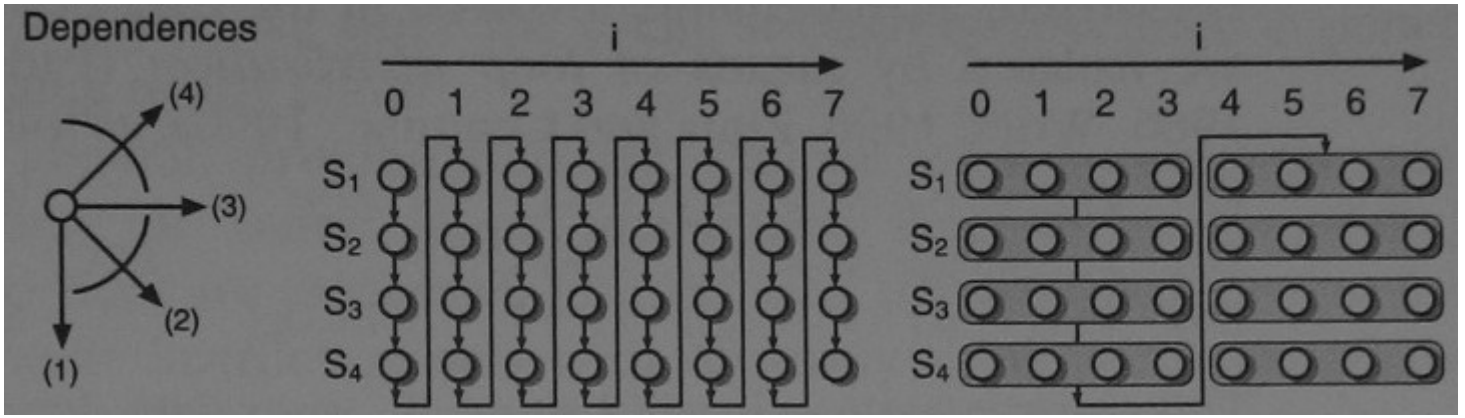
Размышления об алгоритме автоматической векторизации



Data dependency

```
for(i=0; i<U; i++)
{
      S1;
      S2;
      S3;
      S4;
}
```

до векторизации после векторизации



Векторизация сохраняет зависимости:

- loop-independent (1)
- loop-carried lexically forward (2)

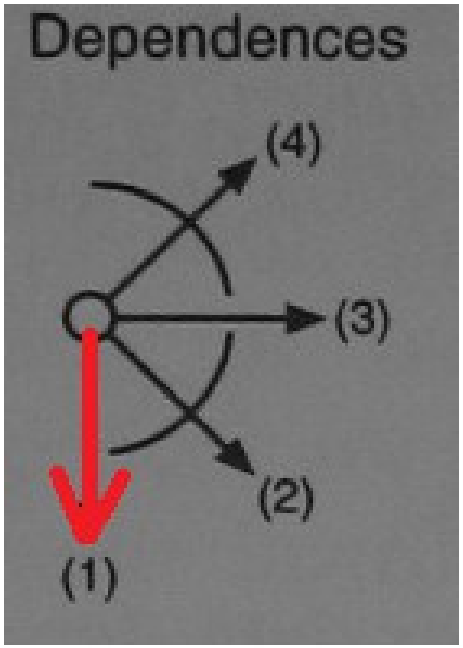
Векторизация часто не сохраняет зависимости:

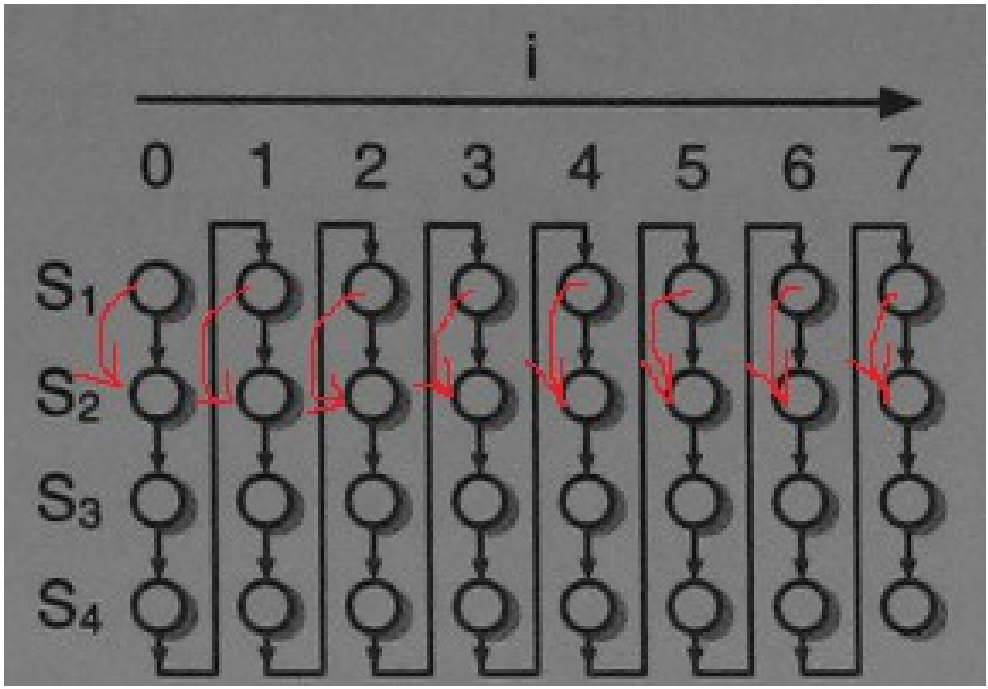
- loop-carried self (3)
- loop-carried lexically backward (4)

Data dependency

loop-independent

```
for(i=0; i<N; i++)
{
S1      A[i] = B[i] + C[i];
S2      D[i] = A[i] * E[i];
S3      F[i] = …;
S4      G[i] = …;
}
```

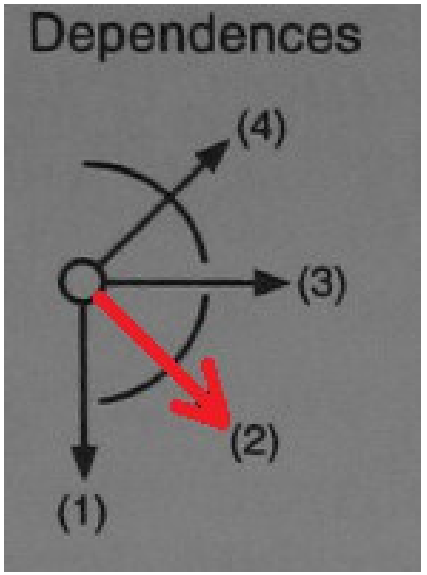


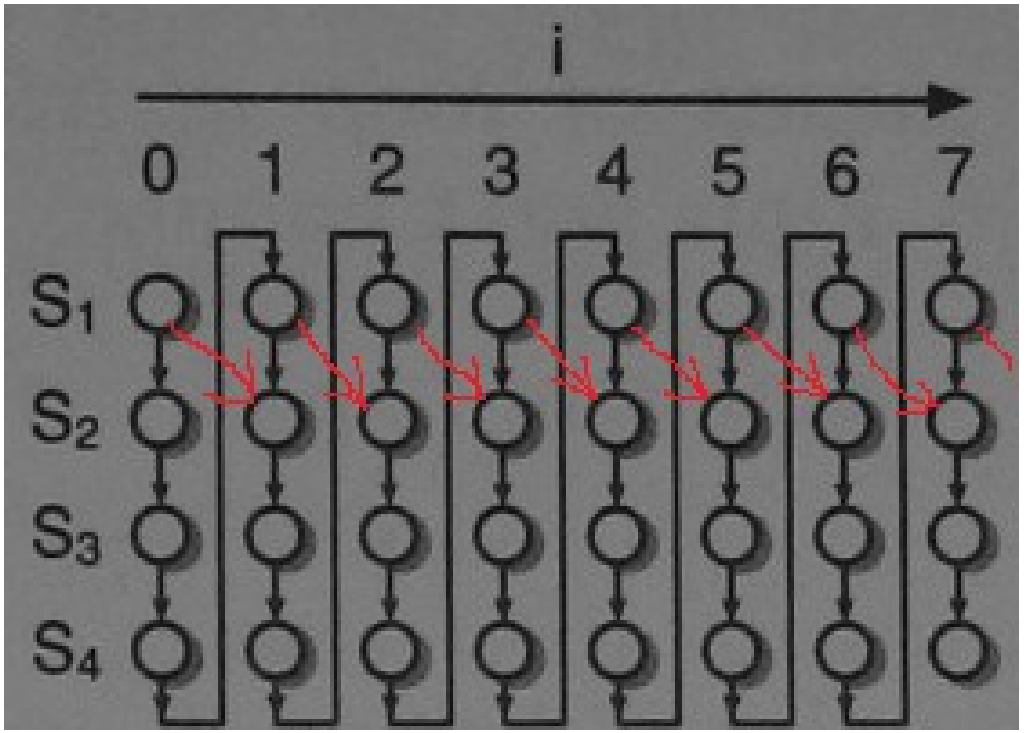


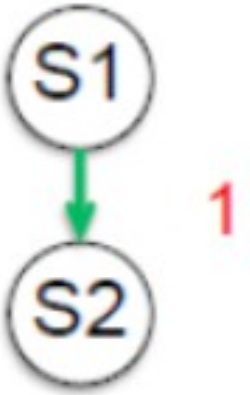
Data dependency

loop-carried lexically forward

```
for(i=0; i<N; i++)
{
S1      A[i] = ...;
S2      B[i] = ... A[i-1] ...;
S3      C[i] = …;
S4      D[i] = …;
}
```





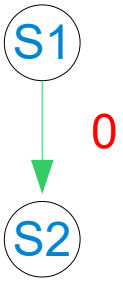


Data dependency

loop-independent

```
for(i=0; i<N; i++)
{
S1     A[i] = B[i] + C[i];
S2     D[i] = A[i] * E[i];
}
```

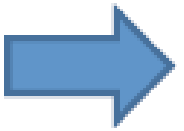
```
A[0:N-1] = B[0:N-1] + C[0:N-1];
D[0:N-1] = A[0:N-1] * E[0:N-1];
```



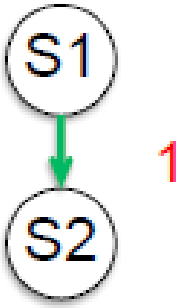
Data dependency

loop-carried lexically forward

```
   for (i=1; i<n; i++){
S1   a[i] = b[i] + 1;
S2   c[i] = a[i-1] + 2;
   }
```



```
a[1:n] = b[1:n] + 1;
c[1:n] = a[0:n-1] + 2;
```

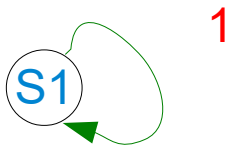


Data dependency

loop-carried self (flow dependency)

```
for(i=1; i<N; i++)
{
S1     A[i] = A[i-1] + B[i];
}
```

Не векторизуется



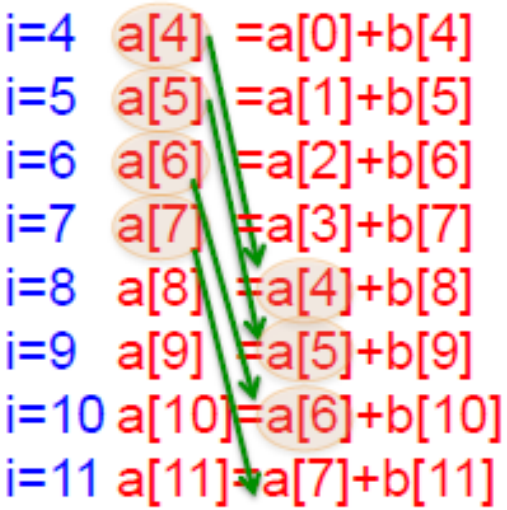
Data dependency

loop-carried self (flow dependency)

```
for(i=4; i<N; i++)
{
S1     A[i] = A[i-4] + B[i];
}
```

Если кол-во компонент в векторе не превышает 4

```
A[4:7]   = A[0:3] + B[4:7];
A[8:11] = A[4:7] + B[8:11];
...
```



4

Data dependency

loop-carried self (anti dependency)

+

loop-independent self (anti dependency)

```
for(i=0; i<N; i++)
{
S1     A[i] = A[i] + A[i+1];
}
```

```
A[0:N-1] = A[0:N-1] + A[1:N];
```

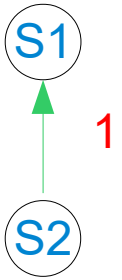
0, 1

Data dependency

loop-carried lexically backward

```
for(i=1; i<N; i++)
{
S1     A[i] = B[i] + C[i];
S2     D[i] = A[i+1] + 1.0;
}
```

Не векторизуется



Data dependency

Можно преобразовать

loop-carried lexically backward

в

loop-carried lexically forward

```
// пример с предыдущего
// слайда
for(i=1; i<N; i++)
{
S1     A[i] = B[i] + C[i];
S2     D[i] = A[i+1] + 1.0;
}
```

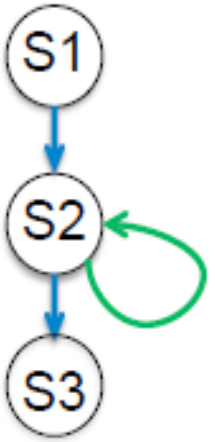
```
// можно векторизовать
for(i=1; i<N; i++)
{
S2     D[i] = A[i+1] + 1.0;
S1     A[i] = B[i] + C[i];
}
```

Data dependency

Loop distribution

```
  for (i=1; i<n; i++){
S1  b[i] = b[i] + c[i];
S2  a[i] = a[i-1]*a[i-2]+b[i];
S3  c[i] = a[i] + 1;
  }
```

```
b[1:n-1] = b[1:n-1] + c[1:n-1];
for (i=1; i<n; i++){
      a[i] = a[i-1]*a[i-2]+b[i];
}
c[1:n-1] = a[1:n-1] + 1;
```



Data dependency

Lexically backward зависимости, дуги которых не входят в циклы в DDG, могут быть преобразованы в lexically forward при помощи переупорядочения инструкции

Только циклы в DDG могут помешать векторизации

Такие циклы должны быть

- распознаны как идиомы и векторизованы
  или
- изолированы как отдельные циклы и оставлены не векторизованными

Data dependency

Приём freezing loops

```
for (i=1; i<n; i++) {
  for (j=1; j<n; j++) {
    a[i][j]=a[i][j]+a[i-1][j];
  }
}
```

Ignoring (freezing) the outer loop:

```
for (j=1; j<n; j++) {
    a[i][j]=a[i][j]+a[i-1][j];
  }
```

S1

```
for (i=1; i<n; i++) {
    a[i][1:n-1]=a[i][1:n-1]+a[i-1][1:n-1];
}
```

Алгоритм

Векторизуем самый вложенный цикл L с глубиной вложенности d

- Построим сокращённый DDG граф, который включает в себя все зависимости кроме
  - относящихся к внешним циклам (т. е. distance vector (a1, … ad) имеет х.б. один ak != 0 для k<d)
  - относящегося к циклу L, для которого distance vector (0, …, 0, ad) ad >= vl, vl — количество компонент в векторе
- Найти SCC на DDG и отсортировать полученные компоненты в топологическом порядке. Применить loop distrbution к циклу и расположить полученные циклы в порядке соответствующему топологическому
- Попытаться векторизовать в отдельности каждый полученный на предыдущем шаге цикл

см. приём freezing loops

см. пример о влиянии кол-ва компонент вектора

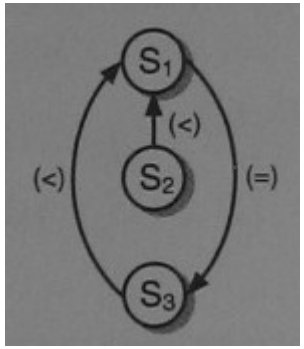
см. пример про loop distribution

Применение алгоритма

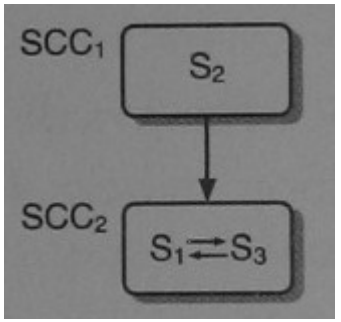
Исходный цикл

```
     for (i = 0; i < U; i++) {
S1:    a[i]   = c[i] * b[i];
S2:    b[i+1] = d[i] - e[i];
S3:    c[i+1] = a[i] * e[i];
     }
```

DDG



Поиск SCC



Топологический порядок

SCC1
SCC2

Преобразуем код

```
     for (i = 0; i < U; i++) { /* candidate vector loop */
S2:    b[i+1] = d[i] - e[i];
     }
     for (i = 0; i < U; i++) { /* sequential loop */
S1:    a[i]   = c[i] * b[i];
S3:    c[i+1] = a[i] * e[i];
     }
```

Циклы в DDG

Теперь мы работаем с SCC на DDG

частично мы уже рассмотрели такие случаи

рассмотрим остальные

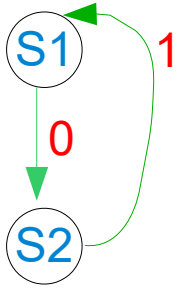
Циклы в DDG

loop-carried lexically backward

+

loop-independ lexically forward

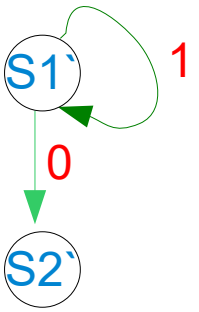
```
for(i=0; i<N-1; i++)
{
S1     B[i] = A[i] + 1.0;
S2     A[i+1] = B[i] + 2.0;
}
```




```
for(i=0; i<N-1; i++)
{
S1`     A[i+1] = A[i] + 1.0 + 2.0;
}

for(i=0; i<N-1; i++)
{
S2`     B[i] = A[i] + 1.0;
}
```

Forward substition



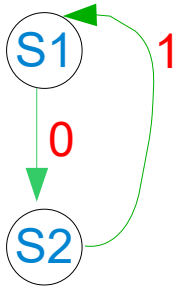
Циклы в DDG

loop-carried lexically backward

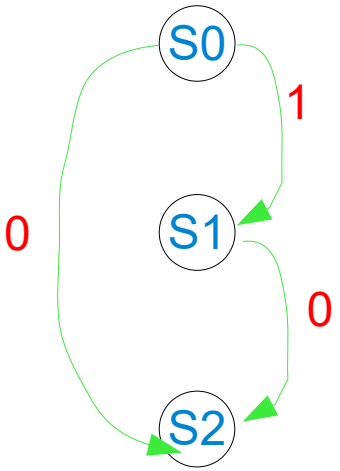
+

loop-independ lexically forward

```
for(i=0; i<N-1; i++)
{
S1     A[i] = B[i] + C[i];
S2     D[i] = (A[i] + A[i+1]) * 0.5;
}
```



```
for(i=0; i<N-1; i++)
{
S0     temp[i] = A[i+1];
S1     A[i] = B[i] + C[i];
S2     D[i] = (A[i] + temp[i]) * 0.5;
}
```



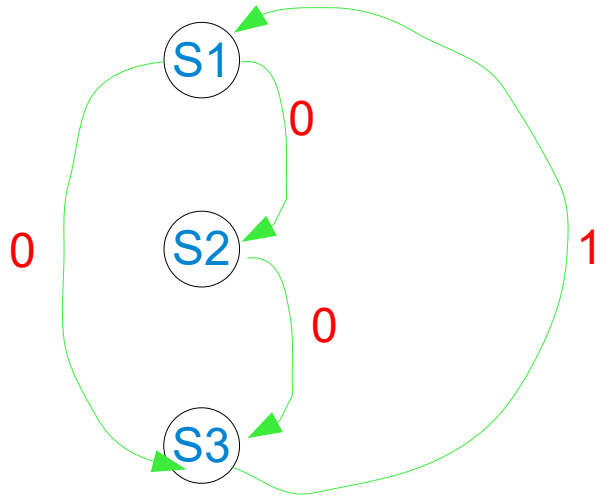
Node splitting

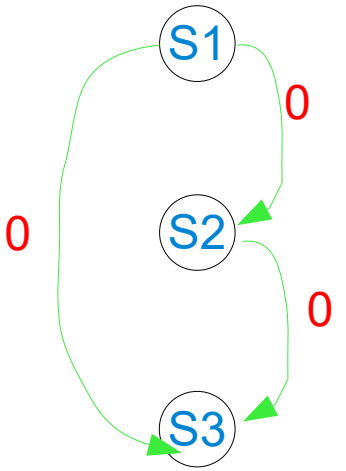
Циклы в DDG

```
  for (int i=0;i<n;i++){
S1  t = a[i];
S2  a[i] = b[i];
S3  b[i] = t;
  }
```

```
  for (int i=0;i<n;i++){
S1  t[i] = a[i];
S2  a[i] = b[i];
S3  b[i] = t[i];
  }
```

Scalar expansion

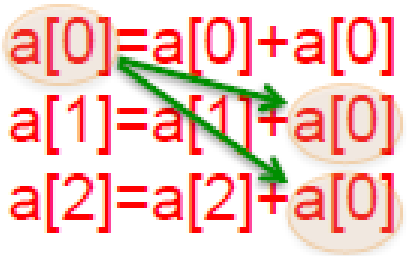




Циклы в DDG

```
for(i=0; i<N; i++)
{
S1      A[i] = A[i] + A[0];
}
```

```
A[0] = A[0] + A[0];
for(i=1; i<N; i++)
{
S1      A[i] = A[i] + A[0];
}
```

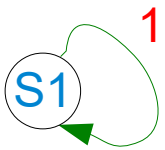


Loop peeling

Циклы в DDG

Операция редукции

```
sum = 0;
for(i=0; i<N; i++)
{
S1    sum = sum + A[i];
}
```



Приводит к реассоциации, что может изменить конечный результат вычислений в случае вещественных чисел

```
sum_vec = {0, 0, 0, 0};
for(i=0; i<N / 4; i++)
{
S1    sum_vec = sum_vec + A[i:i-3];
}
sum = 0;
sum += sum_vec[0];
sum += sum_vec[1];
sum += sum_vec[2];
sum += sum_vec[3];
```

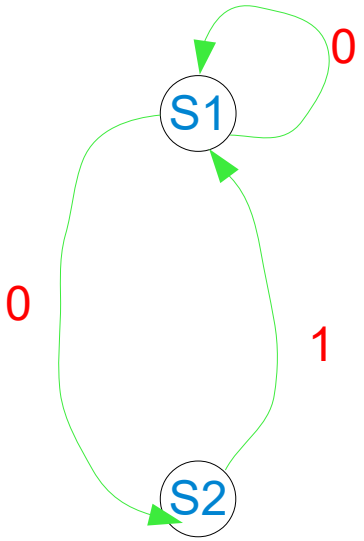
Кроме операции + могут быть другие операции, например * и т.д.

Циклы в DDG

Индуктивная переменная

```
float s = (float)0.0;
for (int i=0;i<LEN;i++){
  s += (float)2.;
  a[i] = s * b[i];
}
```

```
for (int i=0;i<LEN;i++){
  a[i] = (float)2.*(i+1)*b[i];
}
```



Векторизация и локальность

```
   for (int i=1;i<LEN;i++){
S1   a[i] = b[i] + c[i];
S2   d[i] = a[i] + e[i-1];
S3   e[i] = d[i] + c[i];
   }
```

S1 — векторизуется

S2, S3 — оставляются как есть

Векторизация может ухудшить локальность, если массивы очень большие

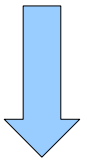
=>

векторизованный код может работать медленнее

Генерация векторизованного кода

```
for (i = 0; i < U; i++) { B(i); }
```

до векторизации



после векторизации

```
            i = 0;
prelude:    if (<run-time-tests>) goto cleanup;
            V = U - (U % vl)
            ...
vector:     for ( ; i < V; i+=vl) { B(i:i+vl-1:1); }
postlude:   ...
cleanup:    for ( ; i < U; i++)   { B(i);          }
exit:
```

Проверка кол-ва итераций

Векторизованный цикл

Оставшиеся итерации

Список литературы

- The Software Vectorization Handbook. A. Bik

Q / A

Backup

Data alignment
Conditional statements

Taking into account of language specific features

Тонкий момент языка C

$T_N = \{u8, s8, u16, s16\}$

	T_N	u32	s32	u64	s64	f32	f64
T_N	s32	u32	s32	u64	s64	f32	f64
u32	u32	u32	u32	u64	s64	f32	f64
s32	s32	u32	s32	u64	s64	f32	f64
u64	u64	u64	u64	u64	u64	f32	f64
s64	s64	s64	s64	u64	s64	f32	f64
f32	f32	f32	f32	f32	f32	f32	f64
f64	f64	f64	f64	f64	f64	f64	f64