Машинно-зависимые языки программирования

Савельев Игорь Леонидович

План

- Команды сопроцессора

1. Команды сопроцессора

Команды управления сопроцессором

Данная группа команд предназначена для общего управления работой сопроцессора.

Команда	Операнды	Пояснение	Описание
FINIT	—	CWR=037Fh; SWR=0; TWR=FFFFh; DPR=0; IPR=0;	Инициализация сопроцессора
FSTSW	dst AX	dst=SWR; AX = SWR;	Считать слово состояния сопроцессора в память
FSTCW	dst AX	dst=CWR; AX = CWR;	Считать слово управления сопроцессора в память
FLDCW	src	CWR=src;	Загрузить слово управления сопроцессора
FCLEX	—	SWR=SWR & 7F00h	Сброс флагов исключений
FINCSTP	—	TOP+=1;	Увеличение указателя стека сопроцессора на 1
FDECSTP	—	TOP-=1;	Уменьшение указателя стека сопроцессора на 1
FFREE	ST(i)	TAG(i)=11b	Очистка указанного регистра
FNOP	—	—	Пустая операция

1. Команды сопроцессора

FINIT

```
Breakpoint 1, main () at mytest.asm:11
11          finit
(gdb) info float
  R7: Empty   0x00000000000000000000
  R6: Empty   0x00000000000000000000
  R5: Empty   0x00000000000000000000
  R4: Empty   0x00000000000000000000
  R3: Empty   0x00000000000000000000
  R2: Empty   0x00000000000000000000
  R1: Empty   0x00000000000000000000
=>R0: Empty   0x00000000000000000000

Status Word:          0x0000
                        TOP: 0
Control Word:         0x037f   IM DM ZM OM UM PM
                        PC: Extended Precision (64-bits)
                        RC: Round to nearest
Tag Word:             0xffff
Instruction Pointer: 0x00:0x00000000
Operand Pointer:      0x00:0x00000000
Opcode:               0x0000
```

1. Команды сопроцессора

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
9B DD /7	FSTSW m2byte	Valid	Valid	Store FPU status word at m2byte after checking for pending unmasked floating-point exceptions.
9B DF E0	FSTSW AX	Valid	Valid	Store FPU status word in AX register after checking for pending unmasked floating-point exceptions.
DD /7	FNSTSW[1] m2byte	Valid	Valid	Store FPU status word at m2byte without checking for pending unmasked floating-point exceptions.
DF E0	FNSTSW[1] AX	Valid	Valid	Store FPU status word in AX register without checking for pending unmasked floating-point exceptions

```
var1 dw 00

fst st0

FSTSW ax
FSTSW [var1]
```

1. Команды сопроцессора

Переменная или выражение	Значение	
var1	0x0	Hex w

Регистр	Hex	
rax	0x0	0

1

fctrl	0x41037f	4260735
fstat	0x10041	65601
ftag	0x1	1

```
var1 dw 00

fst st0          ;1

FSTSW ax     ;2
FSTSW [var1]   ;3
```

2

Регистр	Hex	
rax	0x41	65

3

Переменная или выражение	Значение	
var1	0x41	Hex w

1. Команды сопроцессора

pcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
9B D9 /7	FSTCW m2byte	Valid	Valid	Store FPU control word to m2byte after checking for pending unmasked floating-point exceptions.
D9 /7	FNSTCW[1] m2byte	Valid	Valid	Store FPU control word to m2byte without checking for pending unmasked floating-point exceptions

```
var1 dw 00

FSTCW [var1]
```

3. Сопроцессор

```
section .data
mvar14  dt  13.515151
mvar15  dq 14.515151
tmp      dw 0
section .text
global CMAIN
CMAIN:
   fld tword [mvar14]
   fnstcw [tmp]
   and [tmp],   word 111111**00**11111111b;
   or [tmp],     word 000000**00**000000000b
   fldcw [tmp]
   fld tword [mvar14]
   fadd qword [mvar15]

; or [tmp], word 000000**10**000000000b
; or [tmp], word 000000**11**000000000b
ret
```

Флаги PC регистра CWR

st0	28.0303019999999994717188034343280 38	(raw 0x4003e03e0ef99806f132)
st1	28.0303019999999989408934197854250 67	(raw 0x4003e03e0ef99806f000)
st2	28.0303020477294921875	(raw 0x4003e03e0f0000000000)
st3	13.515151000000000004047318036271 01	(raw 0x4002d83e0ef99806f263)
st4	0	(raw 0x00000000000000000000)
st5	0	(raw 0x00000000000000000000)
st6	0	(raw 0x00000000000000000000)
st7	0	(raw 0x00000000000000000000)

1. Команды сопроцессора

```
section .data
mvar14 dd 13.51
tmp      dw 0
section .text
global CMAIN
CMAIN:
finit
fld dword [mvar14]
fnstcw [tmp]
and [tmp], word 1111001111111111b;
or [tmp],    word 0000000000000000b
fldcw [tmp]
fld dword [mvar14]
frndint

; or [tmp], word 0000010000000000b
; or [tmp], word 0000100000000000b
; or [tmp], word 0000110000000000b
ret
```

Флаги RC регистра CWR

st0	13	(raw 0x4002d000000000000000)
st1	14	(raw 0x4002e000000000000000)
st2	13	(raw 0x4002d000000000000000)
st3	14	(raw 0x4002e000000000000000)
st4	13.5100002288818359375	(raw 0x4002d828f60000000000)
st5	0	(raw 0x00000000000000000000)
st6	0	(raw 0x00000000000000000000)
st7	0	(raw 0x00000000000000000000)

Команды передачи данных вещественного типа

Используются в случае если операнд, применяемый в команде, имеет вещественный тип (4, 8 или 10-байтный).

Команда	Операнды	Пояснение	Описание
FLD	src	TOP_{SWR}-=1; ST(0)=src;	Загрузка операнда в вершину стека
FST	dst	dst=ST(0);	Сохранение вершины стека в память
FSTP	dst	dst=ST(0); TOP_{SWR}+=1;	Сохранение вершины стека в память с выталкиванием
FXCH	ST(i)	ST(0) ↔ ST(i)	Обмен значений ST(0) и ST(i)

Opcode	**Instruction**	**64-Bit Mode**	**Compat/Leg Mode**	**Description**
D9 /0	FLD *m32fp*	Valid	Valid	Push *m32fp* on to the FPU register stack.
DD /0	FLD *m64fp*	Valid	Valid	Push *m64fp* on to the FPU register stack.
DB /5	FLD *m80fp*	Valid	Valid	Push *m80fp* on to the FPU register stack.
D9 C0+i	FLD ST(i)	Valid	Valid	Push ST(i) onto the FPU register stack.

1. Команды сопроцессора

```
section .data
    mvar4  dd 3.14
    mvar8  dq 3.14
    mvar10 dt 3.14
section .text
global CMAIN
CMAIN:

    finit
    fld dword [mvar4]  ;1
    fld qword [mvar8]  ;2
    fld tword [mvar10] ;3
    fld st2            ;4

    ret
```

1

Регистр	Hex
st0	3.140000104904174804687 5
st1	0
st2	0
st3	0
st4	0

2

Регистр	Hex
st0	3.140000000000000124344978758017532 5
st1	3.140000104904174804687 5
st2	0
st3	0
st4	0

3

Регистр	Hex
st0	3.140000000000000000095409791178724 4
st1	3.140000000000000124344978758017532 5
st2	3.140000104904174804687 5
st3	0
st4	0

4

Регистр	Hex
st0	3.140000104904174804687 5
st1	3.140000000000000000095409791178724 4
st2	3.140000000000000124344978758017532 5
st3	3.140000104904174804687 5
st4	0

1. Команды сопроцессора

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 /2	FST *m32fp*	Valid	Valid	Copy ST(0) to *m32fp*.
DD /2	FST *m64fp*	Valid	Valid	Copy ST(0) to *m64fp.*
DD D0+i	FST ST(i)	Valid	Valid	Copy ST(0) to ST(i).
D9 /3	FSTP *m32fp*	Valid	Valid	Copy ST(0) to *m32fp* and pop register stack.
DD /3	FSTP *m64fp*	Valid	Valid	Copy ST(0) to *m64fp* and pop register stack.
DB /7	FSTP *m80fp*	Valid	Valid	Copy ST(0) to *m80fp* and pop register stack.
DD D8+i	FSTP ST(i)	Valid	Valid	Copy ST(0) to ST(i) and pop register stack.

1. Команды сопроцессора

```
finit
fld dword [mvar4]
fld qword [mvar8]
fld tword [mvar10]
fld st2

fst dword [mvar4]
fst st5
ret
```

```
19          fst dword [mvar4]
(gdb) info float
  R7: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R6: Valid   0x4000c8f5c28f5c28f800 +3.140000000000000124
  R5: Valid   0x4000c8f5c28f5c28f5c3 +3.14
=>R4: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R3: Empty   0x00000000000000000000
  R2: Empty   0x00000000000000000000
  R1: Empty   0x00000000000000000000
  R0: Empty   0x00000000000000000000

Status Word:         0x2000
                       TOP: 4
Control Word:        0x037f   IM DM ZM OM UM PM
                       PC: Extended Precision (64-bits)
                       RC: Round to nearest
Tag Word:            0x00ff
Instruction Pointer: 0x00:0x00000000
Operand Pointer:     0x00:0x00000000
Opcode:              0x0000
```

1. Команды сопроцессора

```
finit
fld dword [mvar4]
fld qword [mvar8]
fld tword [mvar10]
fld st2

fst dword [mvar4]
fst st5
ret
```

```
20          fst st5
(gdb) info float
  R7: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R6: Valid   0x4000c8f5c28f5c28f800 +3.140000000000000124
  R5: Valid   0x4000c8f5c28f5c28f5c3 +3.14
=>R4: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R3: Empty   0x00000000000000000000
  R2: Empty   0x00000000000000000000
  R1: Empty   0x00000000000000000000
  R0: Empty   0x00000000000000000000

Status Word:         0x2000
                       TOP: 4
Control Word:        0x037f   IM DM ZM OM UM PM
                       PC: Extended Precision (64-bits)
                       RC: Round to nearest
Tag Word:            0x00ff
Instruction Pointer: 0x00:0x00000000
Operand Pointer:     0x00:0x00000000
Opcode:              0x0000
```

1. Команды сопроцессора

```
finit
fld dword [mvar4]
fld qword [mvar8]
fld tword [mvar10]
fld st2

fst dword [mvar4]
fst st5
ret
```

```
22          ret
(gdb) info float
  R7: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R6: Valid   0x4000c8f5c28f5c28f800 +3.140000000000000124
  R5: Valid   0x4000c8f5c28f5c28f5c3 +3.14
=>R4: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R3: Empty   0x00000000000000000000
  R2: Empty   0x00000000000000000000
  R1: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R0: Empty   0x00000000000000000000

Status Word:          0x2000
                        TOP: 4
Control Word:         0x037f   IM DM ZM OM UM PM
                        PC: Extended Precision (64-bits)
                        RC: Round to nearest
Tag Word:             0x00f3
Instruction Pointer: 0x00:0x00000000
Operand Pointer:     0x00:0x00000000
Opcode:               0x0000
```

1. Команды сопроцессора

```
finit
fld dword [mvar4]
fld qword [mvar8]
fld tword [mvar10]
fld st2

fstp dword [mvar4]
fstp st5
ret
```

```
19          fst dword [mvar4]
(gdb) info float
  R7: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R6: Valid   0x4000c8f5c28f5c28f800 +3.140000000000000124
  R5: Valid   0x4000c8f5c28f5c28f5c3 +3.14
=>R4: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R3: Empty   0x00000000000000000000
  R2: Empty   0x00000000000000000000
  R1: Empty   0x00000000000000000000
  R0: Empty   0x00000000000000000000

Status Word:         0x2000
                       TOP: 4
Control Word:        0x037f   IM DM ZM OM UM PM
                       PC: Extended Precision (64-bits)
                       RC: Round to nearest
Tag Word:            0x00ff
Instruction Pointer: 0x00:0x00000000
Operand Pointer:     0x00:0x00000000
Opcode:              0x0000
```

1. Команды сопроцессора

```
finit
fld dword [mvar4]
fld qword [mvar8]
fld tword [mvar10]
fld st2

fstp dword [mvar4]
fstp st5
ret
```

```
20          fstp st5
(gdb) info float
  R7: Valid   0x4000c8f5c30000000000 +3.1400001049041748O5
  R6: Valid   0x4000c8f5c28f5c28f800 +3.1400000000000124
=>R5: Valid   0x4000c8f5c28f5c28f5c3 +3.14
  R4: Empty   0x4000c8f5c30000000000
  R3: Empty   0x00000000000000000000
  R2: Empty   0x00000000000000000000
  R1: Empty   0x00000000000000000000
  R0: Empty   0x00000000000000000000

Status Word:         0x2800
                       TOP: 5
Control Word:        0x037f   IM DM ZM OM UM PM
                       PC: Extended Precision (64-bits)
                       RC: Round to nearest
Tag Word:            0x03ff
Instruction Pointer: 0x00:0x00000000
Operand Pointer:     0x00:0x00000000
Opcode:              0x0000
```

1. Команды сопроцессора

```
finit
fld dword [mvar4]
fld qword [mvar8]
fld tword [mvar10]
fld st2

fstp dword [mvar4]
fstp st5
ret
```

```
22          ret
(gdb) info float
  R7: Valid   0x4000c8f5c30000000000 +3.140000104904174805
=>R6: Valid   0x4000c8f5c28f5c28f800 +3.140000000000000124
  R5: Empty   0x4000c8f5c28f5c28f5c3
  R4: Empty   0x4000c8f5c30000000000
  R3: Empty   0x00000000000000000000
  R2: Valid   0x4000c8f5c28f5c28f5c3 +3.14
  R1: Empty   0x00000000000000000000
  R0: Empty   0x00000000000000000000

Status Word:         0x3000
                       TOP: 6
Control Word:        0x037f   IM DM ZM OM UM PM
                       PC: Extended Precision (64-bits)
                       RC: Round to nearest
Tag Word:            0x0fcf
Instruction Pointer: 0x00:0x00000000
Operand Pointer:     0x00:0x00000000
Opcode:              0x0000
```

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 C8+i	FXCH ST(i)	Valid	Valid	Exchange the contents of ST(0) and ST(i).
D9 C9	FXCH	Valid	Valid	Exchange the contents of ST(0) and ST(1).

1. Команды сопроцессора

```
section .data
  mvar4 dd -3.14
  mvar8 dq 3.14
  mvar10 dq 6.28
section .text
global CMAIN
CMAIN:
  finit
  fld dword [mvar4]   ;1
  fld qword [mvar8]   ;2
  fld qword [mvar10] ;3
  fxch st2                ;4
  fxch                      ;5
```

3

Регистр	Hex
st0	6.28000000000000024868995751603506 51
st1	3.14000000000000012434497875801753 25
st2	-3.140000104904174804 6875
st3	0

4

Регистр	Hex
st0	-3.140000104904174804 6875
st1	3.14000000000000012434497875801753 25
st2	6.28000000000000024868995751603506 51
st3	0

5

Регистр	Hex
st0	3.14000000000000012434497875801753 25
st1	-3.140000104904174804 6875
st2	6.28000000000000024868995751603506 51
st3	0

1. Команды сопроцессора

```
section .data
mvar4 dd 3.14
section .text
global main
main:
    finit
    fld dword [mvar4] ;1
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4] ;9
    fst dword [mvar4]
    ret
```

```
(gdb) x /f &mvar4
0x404028 <mvar4>:       3.1400001
```

```
Breakpoint 1, main () at mytest.asm:11
11          finit
(gdb) info float
  R7: Empty   0x00000000000000000000
  R6: Empty   0x00000000000000000000
  R5: Empty   0x00000000000000000000
  R4: Empty   0x00000000000000000000
  R3: Empty   0x00000000000000000000
  R2: Empty   0x00000000000000000000
  R1: Empty   0x00000000000000000000
=>R0: Empty   0x00000000000000000000

Status Word:          0x0000
                        TOP: 0
Control Word:         0x037f   IM DM ZM OM UM PM
                        PC: Extended Precision (64-bits)
                        RC: Round to nearest
Tag Word:             0xffff
Instruction Pointer: 0x00:0x00000000
Operand Pointer:      0x00:0x00000000
Opcode:               0x0000
```

1. Команды сопроцессора

```
section .data
mvar4 dd 3.14
section .text
global main
main:
    finit
    fld dword [mvar4] ;1
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4] ;9
    fst dword [mvar4]
    ret
```

```
16          fld dword [mvar4]
(gdb) info float
  R7: Valid   0x4000c8f5c30000000000 +3.1400001049041748O5
  R6: Valid   0x4000c8f5c30000000000 +3.1400001049041748O5
  R5: Valid   0x4000c8f5c30000000000 +3.1400001049041748O5
=>R4: Valid   0x4000c8f5c30000000000 +3.1400001049041748O5
  R3: Empty   0x00000000000000000000
  R2: Empty   0x00000000000000000000
  R1: Empty   0x00000000000000000000
  R0: Empty   0x00000000000000000000

Status Word:          0x2000
                        TOP: 4
Control Word:         0x037f   IM DM ZM OM UM PM
                        PC: Extended Precision (64-bits)
                        RC: Round to nearest
Tag Word:             0x00ff
Instruction Pointer: 0x00:0x00000000
Operand Pointer:     0x00:0x00000000
Opcode:               0x0000
```

1. Команды сопроцессора

```
section .data
mvar4 dd 3.14
section .text
global main
main:
    finit
    fld dword [mvar4] ;1
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4] ;9
    fst dword [mvar4]
    ret
```

```
20          fld dword [mvar4] ;9
(gdb) info float
  R7: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R6: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R5: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R4: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R3: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R2: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R1: Valid   0x4000c8f5c30000000000 +3.140000104904174805
=>R0: Valid   0x4000c8f5c30000000000 +3.140000104904174805

Status Word:          0x0000
                        TOP: 0
Control Word:         0x037f   IM DM ZM OM UM PM
                        PC: Extended Precision (64-bits)
                        RC: Round to nearest
Tag Word:             0x0000
Instruction Pointer: 0x00:0x00000000
Operand Pointer:     0x00:0x00000000
Opcode:               0x0000
```

1. Команды сопроцессора

```
section .data
mvar4 dd 3.14
section .text
global main
main:
    finit
    fld dword [mvar4] ;1
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4]
    fld dword [mvar4] ;9
    fst dword [mvar4]
    ret
```

```
21          nop
(gdb) info float
=>R7: Special 0xffffc000000000000000 Real Indefinite (QNaN)
  R6: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R5: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R4: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R3: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R2: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R1: Valid   0x4000c8f5c30000000000 +3.140000104904174805
  R0: Valid   0x4000c8f5c30000000000 +3.140000104904174805

Status Word:         0x3a41   IE                       SF      C1
                       TOP: 7
Control Word:        0x037f   IM DM ZM OM UM PM
                       PC: Extended Precision (64-bits)
                       RC: Round to nearest
Tag Word:            0x8000
Instruction Pointer: 0x00:0x00000000
Operand Pointer:     0x00:0x00000000
Opcode:              0x0000
```

```
(gdb) x /f &mvar4
0x404028 <mvar4>:       -nan(0x400000)
```

Команды передачи данных целого типа Используются в случае если операнд, применяемый в команде, имеет целый тип (1, 2, 4 или 8-байтный).

Команда	Операнды	Пояснение	Описание
FILD	src	TOP_{SWR}-=1; ST(0)=src;	Загрузка операнда в вершину стека
FIST	dst	dst=ST(0);	Сохранение вершины стека в память
FISTP	dst	dst=ST(0); TOP_{SWR}+=1;	Сохранение вершины стека в память с выталкиванием

1. Команды сопроцессора

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
DF /0	FILD *m16int*	Valid	Valid	Push *m16int* o nto the FPU register stack.
DB /0	FILD *m32int*	Valid	Valid	Push *m32int* o nto the FPU register stack.
DF /5	FILD *m64int*	Valid	Valid	Push *m64int* o nto the FPU register stack.

1. Команды сопроцессора

```
section .data
    mvar2  dw 3
    mvar4  dd 3
    mvar8  dq 3
section .text
global CMAIN
CMAIN:
    finit
    fild word [mvar2]
    fild dword [mvar4]
    fild qword [mvar8]
    ret
```

st0	3	(raw 0x4000c000000000000000)
st1	3	(raw 0x4000c000000000000000)
st2	3	(raw 0x4000c000000000000000)
st3	0	(raw 0x00000000000000000000)
st4	0	(raw 0x00000000000000000000)
st5	0	(raw 0x00000000000000000000)
st6	0	(raw 0x00000000000000000000)
st7	0	(raw 0x00000000000000000000)

1. Команды сопроцессора

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
DF /2	FIST m16int	Valid	Valid	Store ST(0) in m16int.
DB /2	FIST m32int	Valid	Valid	Store ST(0) in m32int.
DF /3	FISTP m16int	Valid	Valid	Store ST(0) in m16int and pop register stack.
DB /3	FISTP m32int	Valid	Valid	Store ST(0) in m32int and pop register stack.
DF /7	FISTP m64int	Valid	Valid	Store ST(0) in m64int and pop register stack.

1. Команды сопроцессора

```
section .data
    mvar4 dd 3.9
    mivar4 dd 0
section .text
global CMAIN
CMAIN:
  finit
  fld dword [mvar4]
  fist dword [mivar4]
ret
```

mvar4	3.9000001	Float d
mivar4	4	Int d

st0	3.9000000953674316	(raw 0x4000f9999a0000000000)
st1	0	(raw 0x00000000000000000000)
st2	0	(raw 0x00000000000000000000)

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
DF /1	FISTTP m16int	Valid	Valid	Store ST(0) in m16int with truncation.
DB /1	FISTTP m32int	Valid	Valid	Store ST(0) in m32int with truncation.
DD /1	FISTTP m64int	Valid	Valid	Store ST(0) in m64int with truncation.

1. Команды сопроцессора

```
section .data
    mvar4 dd 3.9
    mivar4 dd 0
section .text
global CMAIN
CMAIN:
  finit
  fld dword [mvar4]
  fisttp dword [mivar4]
ret
```

mvar4	3.9000001	Float	d
mivar4	3	Int	d

```
float mvar4 = 3.9;
int mivar4 = (int) mvar4;
```

1. Команды сопроцессора

```
__inline long int  float2int(float flt)
{      int intgr;

       _asm
       {      fld flt
              fistp intgr
       } ;

       return intgr ;
}
```

Команды загрузки констант Команды загрузки констант не имеют операндов и загружают соответствующее константное значение в вершину стека сопроцессора.

Команда	Пояснение	Описание
FLDZ	TOP_{SWR}-=1; ST(0)=0;	Загрузка 0
FLD1	TOP_{SWR}-=1; ST(0)=1;	Загрузка 1
FLDPI	TOP_{SWR}-=1; ST(0)=3.1415926535;	Загрузка π
FLDL2T	TOP_{SWR}-=1; ST(0)=3.3219280948;	Загрузка $\log_2 10$
FLDL2E	TOP_{SWR}-=1; ST(0)=1.4426950408;	Загрузка $\log_2 e$
FLDLG2	TOP_{SWR}-=1; ST(0)=0.3010299956;	Загрузка lg 2
FLDLN2	TOP_{SWR}-=1; ST(0)=0.6931471805;	Загрузка ln 2

1. Команды сопроцессора

```
section .data
    mvar4  dd 3.9
    mivar4  dd 5
section .text
global CMAIN
CMAIN:
    finit
    fldz
    fisttp dword [mivar4]   ; mivar4 = 0
    ret
```

Арифметические команды вещественного типа

Схема расположения операндов вещественных команд традиционна для команд сопроцессора. Первый операнд по умолчанию (если не указан в команде) располагается в **вершине стека** сопроцессора — регистре ST(0), и на его место после выполнения команды записывается результат. Второй операнд может быть расположен либо в **памяти**, либо в другом **регистре стека** сопроцессора. По умолчанию в качестве второго операнда используется регистр ST(1). Допустимыми типами операндов в памяти являются вещественные форматы простой и двойной точности. В отличие от целочисленных арифметических команд, вещественные арифметические команды допускают большее разнообразие в сочетании местоположения операндов и самих команд для выполнения конкретного арифметического действия.

1. Команды сопроцессора

Команда	Операнды	Пояснение	Описание
FADD	dst, src	dst = dst + src;	Сложение вещественное
FADDP	ST(i), ST(0)	ST(i) = ST(i) + ST(0); TOP_{SWR}+=1;	Сложение вещественное с выталкиванием
FSUB	dst, src	dst = dst — src;	Вычитание вещественное
FSUBP	ST(i), ST(0)	ST(i) = ST(i) — ST(0); TOP_{SWR}+=1;	Вычитание вещественное с выталкиванием
FSUBR	dst, src	dst = src — dst;	Вычитание вещественное реверсивное
FSUBRP	ST(i), ST(0)	ST(i) = ST(0) — ST(i); TOP_{SWR}+=1;	Вычитание вещественное реверсивное с выталкиванием
FMUL	dst, src	dst = dst * src;	Умножение вещественное
FMULP	ST(i), ST(0)	ST(i) = ST(i) * ST(0); TOP_{SWR}+=1;	Умножение вещественное с выталкиванием
FDIV	dst, src	dst = dst / src;	Деление вещественное
FDIVP	ST(i), ST(0)	ST(i) = ST(i) / ST(0); TOP_{SWR}+=1;	Деление вещественное с выталкиванием
FDIVR	dst, src	dst = src /dst;	Деление вещественное реверсивное
FDIVRP	ST(i), ST(0)	ST(i) = ST(0) / ST(i); TOP_{SWR}+=1;	Деление вещественное реверсивное с выталкиванием

1. Команды сопроцессора

A = (1 + 2) * 4 + 3

1 2 + 4 × 3 + A =

Шаг	Позиция	Инструкции	Содержимое стека
1	1 2 + 4 * 3 + A =	FLD [const1]	1
2	1 2 + 4 * 3 + A =	FLD [const2]	2, 1
3	1 2 + 4 * 3 + A =	FADDP	3
4	1 2 + 4 * 3 + A =	FLD [const4]	4, 3
5	1 2 + 4 * 3 + A =	FMULP	12
6	1 2 + 4 * 3 + A =	FLD [const3]	3, 12
7	1 2 + 4 * 3 + A =	FADDP	15
8	1 2 + 4 * 3 + A =	-	(A), 15
9	1 2 + 4 * 3 + A =	FSTP [A]	-
10		окончание алгоритма	

1. Команды сопроцессора

Арифметические команды целого типа Целочисленные арифметические команды предназначены для работы на тех участках вычислительных алгоритмов, где в качестве исходных данных используются целые числа в памяти, имеющие размерность 4 или 8 байт. Перед выполнением команды целочисленное значение преобразуется к вещественному формату двойной расширенной точности (80 бит).

Команда	Операнды	Пояснение	Описание
FIADD	src	ST(0) = ST(0) + src;	Сложение целочисленное
FISUB	src	ST(0) = ST(0) — src;	Вычитание целочисленное
FISUBR	src	ST(0) = src — ST(0);	Вычитание целочисленное реверсивное
FIMUL	src	ST(0) = ST(0) * src;	Умножение целочисленное
FIDIV	src	ST(0) = ST(0) / src;	Деление целочисленное
FIDIVR	src	ST(0) = src / ST(0);	Деление целочисленное реверсивное

1. Команды сопроцессора

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D8 /0	FADD *m32fp*	Valid	Valid	Add *m32fp* to ST(0) and store result in ST(0).
DC /0	FADD *m64fp*	Valid	Valid	Add *m64fp* to ST(0) and store result in ST(0).
D8 C0+i	FADD ST(0), ST(i)	Valid	Valid	Add ST(0) to ST(i) and store result in ST(0).
DC C0+i	FADD ST(i), ST(0)	Valid	Valid	Add ST(i) to ST(0) and store result in ST(i).
DE C0+i	FADDP ST(i), ST(0)	Valid	Valid	Add ST(0) to ST(i), store result in ST(i), and pop the register stack.
DE C1	FADDP	Valid	Valid	Add ST(0) to ST(1), store result in ST(1), and pop the register stack.
DA /0	FIADD *m32int*	Valid	Valid	Add *m32int* to ST(0) and store result in ST(0).
DE /0	FIADD *m16int*	Valid	Valid	Add *m16int* to ST(0) and store result in ST(0).

```
fld dword [a]     ;
fadd dword [b] ; st0 = a + b
```

1. Команды сопроцессора

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D8 /4	FSUB *m32fp*	Valid	Valid	Subtract *m32fp* from ST(0) and store result in ST(0).
DC /4	FSUB *m64fp*	Valid	Valid	Subtract *m64fp* from ST(0) and store result in ST(0).
D8 E0+i	FSUB ST(0), ST(i)	Valid	Valid	Subtract ST(i) from ST(0) and store result in ST(0).
DC E8+i	FSUB ST(i), ST(0)	Valid	Valid	Subtract ST(0) from ST(i) and store result in ST(i).
DE E8+i	FSUBP ST(i), ST(0)	Valid	Valid	Subtract ST(0) from ST(i), store result in ST(i), and pop register stack.
DE E9	FSUBP	Valid	Valid	Subtract ST(0) from ST(1), store result in ST(1), and pop register stack.
DA /4	FISUB *m32int*	Valid	Valid	Subtract *m32int* from ST(0) and store result in ST(0).
DE /4	FISUB *m16int*	Valid	Valid	Subtract *m16int* from ST(0) and store result in ST(0).

```
fld dword [a]    ;
fld dword [b]    ;
fsubp            ; st0 = a - b
```

1. Команды сопроцессора

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D8 /5	FSUBR *m32fp*	Valid	Valid	Subtract ST(0) from *m32fp* and store result in ST(0).
DC /5	FSUBR *m64fp*	Valid	Valid	Subtract ST(0) from *m64fp* and store result in ST(0).
D8 E8+i	FSUBR ST(0), ST(i)	Valid	Valid	Subtract ST(0) from ST(i) and store result in ST(0).
DC E0+i	FSUBR ST(i), ST(0)	Valid	Valid	Subtract ST(i) from ST(0) and store result in ST(i).
DE E0+i	FSUBRP ST(i), ST(0)	Valid	Valid	Subtract ST(i) from ST(0), store result in ST(i), and pop register stack.
DE E1	FSUBRP	Valid	Valid	Subtract ST(1) from ST(0), store result in ST(1), and pop register stack.
DA /5	FISUBR *m32int*	Valid	Valid	Subtract ST(0) from *m32int* and store result in ST(0).
DE /5	FISUBR *m16int*	Valid	Valid	Subtract ST(0) from *m16int* and store result in ST(0).

1. Команды сопроцессора

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D8 /1	FMUL *m32fp*	Valid	Valid	Multiply ST(0) by *m32fp* and store result in ST(0).
DC /1	FMUL *m64fp*	Valid	Valid	Multiply ST(0) by *m64fp* and store result in ST(0).
D8 C8+i	FMUL ST(0), ST(i)	Valid	Valid	Multiply ST(0) by ST(i) and store result in ST(0).
DC C8+i	FMUL ST(i), ST(0)	Valid	Valid	Multiply ST(i) by ST(0) and store result in ST(i).
DE C8+i	FMULP ST(i), ST(0)	Valid	Valid	Multiply ST(i) by ST(0), store result in ST(i), and pop the register stack.
DE C9	FMULP	Valid	Valid	Multiply ST(1) by ST(0), store result in ST(1), and pop the register stack.
DA /1	FIMUL *m32int*	Valid	Valid	Multiply ST(0) by *m32int* and store result in ST(0).
DE /1	FIMUL *m16int*	Valid	Valid	Multiply ST(0) by *m16int* and store result in ST(0).

```
fld dword [a]    ;
fmul dword [b] ; st0 = a * b
```

1. Команды сопроцессора

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D8 /6	FDIV *m32fp*	Valid	Valid	Divide ST(0) by *m32fp* and store result in ST(0).
DC /6	FDIV *m64fp*	Valid	Valid	Divide ST(0) by *m64fp* and store result in ST(0).
D8 F0+i	FDIV ST(0), ST(i)	Valid	Valid	Divide ST(0) by ST(i) and store result in ST(0).
DC F8+i	FDIV ST(i), ST(0)	Valid	Valid	Divide ST(i) by ST(0) and store result in ST(i).
DE F8+i	FDIVP ST(i), ST(0)	Valid	Valid	Divide ST(i) by ST(0), store result in ST(i), and pop the register stack.
DE F9	FDIVP	Valid	Valid	Divide ST(1) by ST(0), store result in ST(1), and pop the register stack.
DA /6	FIDIV *m32int*	Valid	Valid	Divide ST(0) by *m32int* and store result in ST(0).
DE /6	FIDIV *m16int*	Valid	Valid	Divide ST(0) by *m16int* and store result in ST(0).

```
fld   dword [a]    ;
fdiv dword [b] ; st0 = a / b
```

1. Команды сопроцессора

$$x^2 + 7x + 6 = 0$$

```
section .data
  four DD 4.
  two DD 2.
section .bss
 a resd 1
 b resd 1
 c resd 1
 x1 resd 1
 x2 resd 1
 d resd 1
 ac resd 1
 bb resd 1
section .text
global CMAIN
CMAIN:
  mov dword [a], 1
  mov dword [b], 7
  mov dword [c], 6
```

```
FINIT ; иниц. 8087
                          ;----------ST(0)---------- ! ----------ST(1)----------!
FILD dword [b]            ; b                        ! ?
FMUL ST0,ST0              ; b*b                      ! ?
FST dword [bb]            ; копирование вершины стека ==> bb
FILD dword [a]            ; a                        ! b*b
FMUL dword [four]         ; 4*a                      ! b*b
FIMUL dword [c]           ; 4*a*c                    ! b*b
FST dword [ac]            ; копирование вершины стека ==> ac
FSUBP ST1,ST0             ;d=b*b-4*a*c               ! ?
FST dword [d]             ; копирование вершины стека ==> d
FSQRT                     ; sqrt(d)                  ! ?
FLD ST0                   ; sqrt(d)                  ! sqrt(d)
FCHS                      ; -sqrt(d)                 ! sqrt(d)
FIADD dword [b]           ; b-sqrt(d)                ! sqrt(d)
FCHS                      ; -b+sqrt(d)               ! sqrt(d)
FXCH ST1                  ; sqrt(d)                  ! -b+sqrt(d)
FIADD dword [b]           ; b+sqrt(d)                ! -b+sqrt(d)
FCHS                      ; -b-sqrt(d)               ! -b+sqrt(d)
FIDIV dword [a]           ; -b-sqrt(d)/a             ! -b+sqrt(d)
FDIV dword [two]          ; -b-sqrt(d)/a/2           ! -b+sqrt(d)
FSTP dword [x2]           ; -b+sqrt(d)               ! ?
FIDIV dword [a]           ; -b+sqrt(d)/a             ! -b+sqrt(d)
FDIV dword [two]          ; -b+sqrt(d)/a/2           ! ?
FSTP dword [x1]
```

1. Команды сопроцессора

$$x^2 + 7x + 6 = 0$$

x1 =-6, x2 =-1

x1	-1	Float d
x2	-6	Float d

st0	0	(raw 0x00000000000000000000)
st1	0	(raw 0x00000000000000000000)
st2	0	(raw 0x00000000000000000000)
st3	0	(raw 0x00000000000000000000)
st4	0	(raw 0x00000000000000000000)
st5	0	(raw 0x00000000000000000000)
st6	-6	(raw 0xc001c000000000000000)
st7	-1	(raw 0xbfff8000000000000000)
fctrl	0x37f	895
fstat	0x0	0
ftag	0xffff	65535

Команды сравнения данных

Команды данной группы выполняют сравнение значений числа в вершине стека и операнда, указанного в команде

Команда	Операнды	Пояснение	Описание
FCOM FUCOM	src	ST(0) — src	Вещественное сравнение
FCOMP FUCOMP	src	ST(0) — src; TOP_{SWR}+=1;	Вещественное сравнение с выталкиванием
FCOMPP FUCOMPP	—	ST(0) — ST(1); TOP_{SWR}+=2;	Вещественное сравнение с двойным выталкиванием
FCOMI FUCOMI	ST, ST(i)	ST(0) — ST(i)	Вещественное сравнение с модификацией EFLAGS
FCOMIP FUCOMIP	ST, ST(i)	ST(0) — ST(i); TOP_{SWR}+=1;	Вещественное сравнение с выталкиванием с модификацией EFLAGS
FXAM	—		Анализ ST(0)

1. Команды сопроцессора

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D8 /2	FCOM *m32fp*	Valid	Valid	Compare ST(0) with *m32fp*.
DC /2	FCOM *m64fp*	Valid	Valid	Compare ST(0) with *m64fp*.
D8 D0+i	FCOM ST(i)	Valid	Valid	Compare ST(0) with ST(i).
D8 D1	FCOM	Valid	Valid	Compare ST(0) with ST(1).
D8 /3	FCOMP *m32fp*	Valid	Valid	Compare ST(0) with *m32fp* and pop register stack.
DC /3	FCOMP *m64fp*	Valid	Valid	Compare ST(0) with m64fp and pop register stack.
D8 D8+i	FCOMP ST(i)	Valid	Valid	Compare ST(0) with ST(i) and pop register stack.
D8 D9	FCOMP	Valid	Valid	Compare ST(0) with ST(1) and pop register stack.
DE D9	FCOMPP	Valid	Valid	Compare ST(0) with ST(1) and pop register stack twice.

0. Ordered vs Unordered

FCOM vs FUCOM

Условие	C3	C2	C0
ST(0) > src	0	0	0
ST(0) < src	0	0	1
ST(0) = src	1	0	0
Недопустимая операция (#IA)	1	1	1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	C3	TOP			C2	C1	C0	ES	SF	PE	UE	OE	ZE	DE	IE

invalid-arithmetic-operand exception (#IA)

FCOM: один из операндов QNaN

FUCOM: один из операндов SNaN

0. Ordered vs Unordered

```
section .data
qnan dd 0x7fc00000
snan dd 0x7fa00000

section .text
global CMAIN
CMAIN:
    finit
    fld1
    fldz
    ;fld dword [qnan]
    ;fld dword [snan]
    fcom
    ret
```

FCOM

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	B	C3	TOP			C2	C1	C0	ES	SF	PE	UE	OE	ZE	DE	IE
0 – 1 0xc03100		0				0		1								0
0 – QNaN 0xc07501		1				1		1								1
QNaN – QNaN 0xc07501		1				1		1								1
QNaN – SNaN 0xc07501		1				1		1								1

0. Ordered vs Unordered

FUCOM

```
section .data
qnan dd 0x7fc00000
snan dd 0x7fa00000

section .text
global CMAIN
CMAIN:
    finit
    fld1
    fldz
    ;fld dword [qnan]
    ;fld dword [snan]
    fucom
    ret
```

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	B	C3	TOP			C2	C1	C0	ES	SF	PE	UE	OE	ZE	DE	IE
0 – 1 0xc03100		0				0		1								0
0 – QNaN 0xc07500		1				1		1								**0**
QNaN – QNaN 0xc07500		1				1		1								**0**
QNaN – SNaN 0xc07501		1				1		1								1

1. Команды сопроцессора

Условие	C3	C2	C0
ST(0) > src	0	0	0
ST(0) < src	0	0	1
ST(0) = src	1	0	0
Недопустимая операция (#IA)	1	1	1

SWR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	C3	TOP			C2	C1	C0	ES	SF	PE	UE	OE	ZE	DE	IE

SAHF (AH -> EFLAGS)

15	14	13	12	11	10	9	8
SF	ZF		AF		PF		CF

1. Команды сопроцессора

Мнемокод	Название	Условие перехода после команды CMP op_1, op_2	Значения флагов	Примечание
JE	Переход если равно	op1 = op2	ZF = 1	Для всех чисел
JNE	Переход если не равно	op1 ≠ op2	ZF = 0	
JL/JNGE	Переход если меньше	op1 < op2	SF ≠ OF	Для чисел со знаком
JLE/JNG	Переход если меньше или равно	op1 ≤ op2	SF ≠ OF или ZF = 1	
JG/JNLE	Переход если больше	op1 > op2	SF = OF и ZF = 0	
JGE/JNL	Переход если больше или равно	op1 ≥ op2	SF = OF	
JB/JNAE	Переход если ниже	op1 < op2	CF = 1	Для чисел без знака
JBE/JNA	Переход если ниже или равно	op1 ≤ op2	CF = 1 или ZF = 1	
JA/JNBE	Переход если выше	op1 > op2	CF = 0 и ZF = 0	
JAE/JNB	Переход если выше или равно	op1 ≥ op2	CF = 0	

1. Команды сопроцессора

```
segment  .data
   v0  dd 2.8
   v1  dd 2.9
segment .text
main :
   finit
   fld dword [v1]
   fld dword [v0]
   fcompp      ; сравниваем ST(0) и ST(1) ; ST(0) — ST(1);
   fstsw ax    ; данные регистра состояния загружаем в AX
   sahf        ; загружаем данные из AX в регистр FLAGS
   setb al     ; AL = true if st(0) < st(1)
   movzx eax, al ; проверяем значение AL
   ret

; eax == 1 , если v0 < v1
```

1. Команды сопроцессора

```
segment  .data
   v0  dd 2.8
   v1  dd 2.9
segment .text
main :
   finit
   fld dword [v1]
   fld dword [v0]
   fcompp          ; сравниваем ST(0) и ST(1)
   fstsw ax        ; данные регистра состояния загружаем в AX
   sahf            ; загружаем данные из AX в регистр FLAGS
   jnb set_10      ; если st(0) >= st(1) переход к set_10
   mov eax, 5;
   jmp end_proc
set_10:
   mov eax, 10
end_proc:
   ret
```

; eax == 5 , если v0 < v1; eax == 10, если v0 >= v1

Особый интерес представляет команда FCOMI (FUCOMI). Она сравнивает содержимое регистра ST(0) со значением операнда ST(i) и устанавливает биты ZF, PF, CF регистра EFLAGS в соответствии с таблицей. Анализ выполнения сравнения осуществляет последующая команда условного перехода (команда центрального процессора).

Условие	ZF	PF	CF	Переход
ST(0) > ST(i)	0	0	0	ja
ST(0) < ST(i)	0	0	1	jb
ST(0) = ST(i)	1	0	0	je
ST(0) >= ST(i)	*	0	0	jae
ST(0) <= ST(i)	*	0	*	jbe
Недопустимая операция (#IA)	1	1	1	

1. Команды сопроцессора

```
segment  .data
   v0  dd 2.8
   v1  dd 2.9
segment .text
main :
   finit
   fld dword [v1]
   fld dword [v0]
   fcomip st0, st1     ; сравниваем ST(0) и ST(1)
   setb al             ; AL = 1 если st(0) < st(1)
   movzx eax, al       ; проверяем значение AL
   ret

; eax == 1 , если v0 < v1
```

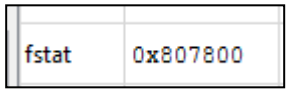
Команда FXAM проверяет содержимое регистра ST(0) и устанавливает биты C0, C2, C3 регистра swr в соответствии с таблицей. Бит C1 устанавливается равным знаковому биту ST(0).

Класс	C3	C2	C0
Неподдерживаемый формат	0	0	0
Нечисло (NaN)	0	0	1
Конечное число	0	1	0
Бесконечность	0	1	1
Ноль	1	0	0
Пустой регистр	1	0	1
Ненормированное число	1	1	0

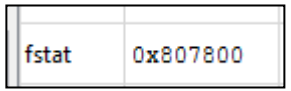
FTST (сравнение с 0) - FXAM

```
var1 dd 0
var2 dd 3.14
var3 dd -3.14

fld dword [var1]
ftst
Fxam

fstp st0
fst st0
ftst
fxam
```

0

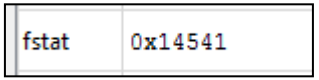


78: 01111000

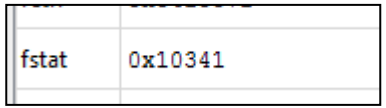


78: 01111000

NaN

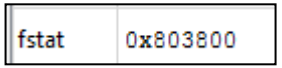


45: 01000101

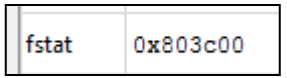


03: 00000011

3.14

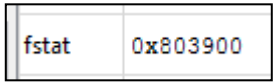


38: 00111000

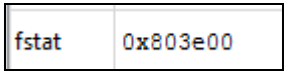


3c: 00111100

-3.14



39: 00111001



3e: 00111110

Команды сравнения данных целого типа

Команда	Операнды	Пояснение	Описание
FICOM	src	ST(0) — src	Сравнение с целым числом src
FICOMP	src	ST(0) — src; TOP_{SWR}+=1;	Сравнение с целым числом src с выталкиванием
FTST	—	ST(0)-0;	Анализ ST(0) (сравнение с нулем)

Команды сравнения данных целого типа

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
DE /2	FICOM *m16int*	Valid	Valid	Compare ST(0) with *m16int.*
DA /2	FICOM *m32int*	Valid	Valid	Compare ST(0) with *m32int.*
DE /3	FICOMP *m16int*	Valid	Valid	Compare ST(0) with *m16int* and pop stack register.
DA /3	FICOMP *m32int*	Valid	Valid	Compare ST(0) with *m32int* and pop stack register.

Команды условного перемещения FCMOVcc

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
DA C0+i	FCMOVB ST(0), ST(i)	Valid	Valid	Move if below (CF=1).
DA C8+i	FCMOVE ST(0), ST(i)	Valid	Valid	Move if equal (ZF=1).
DA D0+i	FCMOVBE ST(0), ST(i)	Valid	Valid	Move if below or equal (CF=1 or ZF=1).
DA D8+i	FCMOVU ST(0), ST(i)	Valid	Valid	Move if unordered (PF=1).
DB C0+i	FCMOVNB ST(0), ST(i)	Valid	Valid	Move if not below (CF=0).
DB C8+i	FCMOVNE ST(0), ST(i)	Valid	Valid	Move if not equal (ZF=0).
DB D0+i	FCMOVNBE ST(0), ST(i)	Valid	Valid	Move if not below or equal (CF=0 and ZF=0).
DB D8+i	FCMOVNU ST(0), ST(i)	Valid	Valid	Move if not unordered (PF=0).

Дополнительные инструкции

Дополнительные арифметические команды Команды этой группы не имеют операндов, производят действие с операндом в вершине стека сопроцессора. Результат выполнения операции сохраняется в регистре ST(0). Сбрасывают в 0 признак C1 при пустом стеке, устанавливают в 1 при округлении результата.

Команда	Пояснение	Описание
FSQRT	$ST(0) = \sqrt{ST(0)}$	Вычисление квадратного корня
FABS	$ST(0) = \lvert ST(0) \rvert$	Вычисление модуля
FCHS	ST(0) = -ST(0)	Изменение знака
FXTRACT	temp = ST(0); ST(0)=порядок(temp); TOP-=1; ST(0)=мантисса(temp);	Выделение порядка и мантиссы
FSCALE	$ST(0) = ST(0) \cdot 2^{ST(1)}$	Масштабирование по степеням 2
FRNDINT	ST(0)=(ST(0))	Округление ST(0)
FPREM	ST(0)=ST(0)-Q*ST(1)	Частичный остаток от деления

Команда FXTRACT – выделение порядка и мантиссы. Операнд-источник по умолчанию, хранящийся в регистре ST(0), разделяется на порядок и мантиссу, порядок сохраняется в ST(0), а затем мантисса помещается в стек, меняя при этом указатель вершины стека (поле top). Для операнда, хранящего мантиссу, знак и мантисса остаются неизменными по сравнению с операндом источника. Вместо порядка записывается 3FFFh. После выполнения команды регистр ST(1) хранит значение порядка исходного операнда.

Команда FSCALE– команда масштабирования: изменяет порядок значения, находящегося в вершине стека сопроцессора ST(0) на величину ST(1). Команда не имеет операндов. Величина в ST(1) рассматривается как число со знаком. Его прибавление к полю порядка вещественного числа в ST(0) означает его умножение на величину $2^{ST(1)}$. С помощью данной команды удобно масштабировать на степень двойки некоторую последовательность чисел в памяти. Для этого достаточно последовательно загружать числа в вершину стека, после чего применять команду FSCALE и сохранять значения обратно в памяти.
По сути (для норм.чисел) изменяет только значение экспоненты.

Команда FRNDINT – округляет значение, находящееся в вершине стека сопроцессора ST(0). Команда не имеет операндов. Сопроцессор имеет программно-аппаратные средства для выполнения операции округления тех результатов работы команд, которые не могут быть точно представлены. Но операция округления может быть проведена и принудительно к значению в регистре ST(0), для этого предназначена последняя команда в группе дополнительных команд — команда округления. Возможны четыре режима округления величины в ST(0), которые определяются значениями в поле RC управляющего регистра CWR. Для изменения режима округления используются команды FSTCWR и FLDCWR, которые, соответственно, записывают в память содержимое управляющего регистра сопроцессора и восстанавливают его обратно. Таким образом, пока содержимое этого регистра находится в памяти, можно установить необходимое значение поля RC.

1. Команды сопроцессора

Команда FPREM – получение частичного остатка от деления. Исходные значения делимого и делителя размещаются в стеке — делимое в ST(0), делитель в ST(1). Команда производит вычисления по формуле

$$ST(0)=ST(0)-Q*ST(1),$$

где Q – целочисленное частное от деления. Делитель рассматривается как некоторый модуль. Поэтому в результате работы команды получается остаток от деления по модулю. Физически работа команды заключается в реализации деление в столбик. При этом каждое промежуточное деление осуществляется отдельной командой FPREM. Цикл, центральное место в котором занимает команда FPREM, завершается, когда очередная полученная разность в ST(0) становится меньше значения модуля в ST(1). Судить об этом можно по состоянию флага C2 в регистре состояния swr:

- если C2=0, то работа команды fprem полностью завершена, так как разность в ST(0) меньше значения модуля в ST(1);
- если C2=1, то необходимо продолжить выполнение команды fprem, так как разность в ST(0) больше значения модуля в ST(1).

Таким образом, необходимо анализировать флаг C2 в теле цикла. Для этого C2 записывается в регистр флагов основного процессора с последующим анализом его командами условного перехода. Другой способ заключается в сравнении ST(0) и ST(1). Команда fprem не соответствует последнему стандарту на вычисления с плавающей точкой IEEE-754. По этой причине в систему команд сопроцессора i387 была введена команда fprem1, которая отличается от FPREM тем, что накладывается дополнительное требование на значение остатка в ST(0). Это значение не должно превышать половины модуля в ST(1). После полного завершения работы команды FPREM/FPREM1 (когда C2=0), биты C0, C3, C1 содержат значения трех младших разрядов частного.

Команды трансцендентных функций

Сопроцессор имеет ряд команд, предназначенных для вычисления значений тригонометрических функций, а также значений логарифмических и показательных функций. Значения аргументов в командах, вычисляющих результат тригонометрических функций, должны задаваться в радианах. Данная группа команд не имеет операндов. Результат сохраняется в регистре ST(0). Сбрасывает в 0 признак C1 при пустом стеке, устанавливают в 1 при округлении. Признак C2 устанавливается в 1 при выходе значения угла за границы диапазона $[-2^{63}; 2^{63}]$.

1. Команды сопроцессора

оманда	Пояснение	Описание
FSIN	ST(0) = sin(ST(0))	Вычисление синуса
FCOS	ST(0) = cos(ST(0))	Вычисление косинуса
FSINCOS	temp=ST(0); ST(0)=sin(temp); TOP-=1; ST(0)=cos(temp);	Вычисление синуса и косинуса
FPTAN	ST(0)=tg(ST(0)); TOP-=1; ST(0)=1.0;	Вычисление тангенса
FPATAN	ST(1)=atan(ST(1)/ST(0)); TOP+=1;	Вычисление арктангенса
F2XM1	$ST(0)=2^{ST(0)}-1$;	Вычисление выражения $y=2^x-1$
FYL2X	x=ST(0); y=ST(1); TOP+=1; $ST(0)=y*\log_2 x$;	Вычисление выражения $y*\log_2 x$
FYL2XP1	x=ST(0); y=ST(1); TOP+=1; $ST(0)=y*\log_2(x+1)$;	Вычисление выражения $y*\log_2(x+1)$

Команда F2XM1 вычисляет значение функции: 2^x-1. Исходное значение x размещается в вершине стека сопроцессора ST(0) и должно лежать в диапазоне [-1; 1]. Результат замещает значение в регистре ST(0). Эта команда может быть использована для вычисления показательных функций. 1 вычитается для того, чтобы получить точный результат, когда x близок к нулю. Поскольку нормированное число всегда содержит в качестве первой значащей цифры единицу, если в результате вычисления функции получается число 1,000000000456…, то команда F2XM1, вычитая 1 из этого числа, и затем, нормируя результат, формирует больше значащих цифр, то есть делает его более точным. Неявное вычитание единицы командой F2XM1 компенсируется командой FADD с единичным операндом.

Команда FYL2X вычисляет значение функции $ST(0)=ST(1)\cdot\log_2 ST(0)$. Значение x должно лежать в диапазоне $[0;\infty)$. Перед тем, как осуществить запись результата в вершину стека, команда FYL2X выталкивает значения x и y из стека, и только после этого производит запись результата в стек.

1. Команды сопроцессора

Команда FYL2XP1–вычисляет $ST(0)=ST(1)\cdot\log_2(ST(0)+1)$. Значение x должно лежать в диапазоне $[0;\infty)$. Поскольку специальной команды в сопроцессоре для операции возведения в степень нет, возведение в произвольную степень числа с любым основанием производится по формуле:

$$x^y=2^{y\cdot\log_2 x}$$

Вычисление значения выражения $z=y\cdot\log_2 x$ для любых $y>0$ и $x>0$ производится командой сопроцессора FYL2X. Вычисление 2^z-1 производится командой F2XM1. Лишнее действие вычитания 1 можно компенсировать сложением с единицей. Но в последнем действии есть тонкий момент, который связан с тем, что величина аргумента x должна лежать в диапазоне: [-1; 1]. В случае, если x превышает это значение (например, для вычисления 16^3) при вычислении выражения $3\cdot\log_2 16$ командой FYL2X, получим в стеке значение 12. Попытка вычислить значение 2^{12} командой F2XM1 ни к чему не приведет — результат будет не определен. В этой ситуации используется команда сопроцессора FSCALE, которая вычисляет значение выражения 2^x, но для **целых** значений x со знаком. Применив формулу

$$2^{a+b} = 2^a\cdot 2^b,$$

получаем решение проблемы. Разделяем дробный показатель степеней больших 1 по модулю на две части — целую и дробную. После этого вычисляем отдельно командами FSCALE и F2XM1 степени двойки и перемножаем результаты.

1. Команды сопроцессора

```
1.;x = b *log b (x)      3^9 = 2 ^ log2(3^9) =  2 ^ (9 * log2(3)) =
                    2 ^ (9 * log2(3) % 1)  *  2 ^ (9 * log2(3) / 1) =
                    2 ^ (9 * log2(3) % 1)-1 + 1) * 2 ^ (9 * log2(3) / 1)
2.power   dd  9
3.x       dd  3
4.result  dd  0
5.
6.; В коде
7.      finit               ;      st0          ;    st1        ;  st2
8.      fild    [power]     ;  9
9.      fild    [x]         ;  3                 ; 9
10.     fyl2x               ;  9 * log2(3)=14.26                ;
11.     fld1                ;  1                 ; 14.26        ;
12.     fld     st1         ; 14.26              ; 1            ; 14.26
13.     fprem               ; 0.26               ; 1            ; 14.26
14.     f2xm1               ; 2^0.26-1=0.20      ; 1            ; 14.26
15.     faddp               ; 1.20               ; 14.26
16.     fscale              ; 1.20* 2^14 =19683  ; 14.26
17.     fxch    st1         ; 14.26              ; 19683
18.     fstp    st0         ; 19683
19.     fistp   [result]    ; result = 3^9
```

MMX, SSE,…, AVX

4. SISD и т.п.

Таксономия (Классификация) Флинна (англ. Flynn's taxonomy) — общая классификация архитектур ЭВМ по признакам наличия параллелизма в потоках команд и данных. Была предложена Майклом Флинном в 1966 году и расширена в 1972 году.

Всё разнообразие архитектур ЭВМ в этой таксономии Флинна сводится к четырём классам:

ОКОД — Вычислительная система с одиночным потоком команд и одиночным потоком данных
(**SISD**, single instruction stream over a single data stream).
ОКМД — Вычислительная система с одиночным потоком команд и множественным потоком данных
(**SIMD**, single instruction, multiple data).
МКОД — Вычислительная система со множественным потоком команд и одиночным потоком данных
(**MISD**, multiple instruction, single data).
МКМД — Вычислительная система со множественным потоком команд и множественным потоком данных
(**MIMD**, multiple instruction, multiple data).

4. SISD и т.п.

SISD – пример Интелевского процессора, один поток команд в сегменте инструкций и один поток данных в сегменте данных. За один шаг одна инструкция обрабатывает одну порцию данных

MIMD – это различные параллельные решения, к примеру Analog Devices 21161 (tiger shark), содержит два ALU и в коде можно в одной строке написать две инструкции, в которые передать независимые данные.

MISD – как пример, вычисления на матрицах, когда значение ячейки матрицы подвергается разным преобразованиям в зависимости от шага вычислений. Мажоритарные отказоустойчивые системы (диверсифицированное программирование)

SIMD – несколько данных подвергаются одной и той же операции. Далее поподробнее…

4. SISD и т.п.

Предположим, вам нужно записать звучание нескольких музыкальных инструментов, или создать программу для аудиоконференции, в которой несколько человек могут говорить одновременно. Т.е. вам надо микшировать (mixing) звук.

Из курса физики вы знаете, что звук это волна, при сложении происходит явление интерференции, т.е. сложение волн

Так вы знаете, что по теореме Котельникова, используя квантование, вы можете оцифровать ваши аналоговые сигналы и получить, к примеру 20 мс отрывок в виде массива из 160 байт, где каждый байт – единичное значение вашего сигнала.

Задача – сложить поэлементно полученные массивы.
Использовать типовое решение, вида:

```
mov eax, [arr1 + esi]
mov ebx, [arr2 + edi]
add eax, ebx
Повторить
```

Не очень красиво и довольно долго.
Решение? **SIMD**
Создать расширение для процессора, позволяющее одну команду (к примеру сложение) реализовать одновременно для нескольких данных

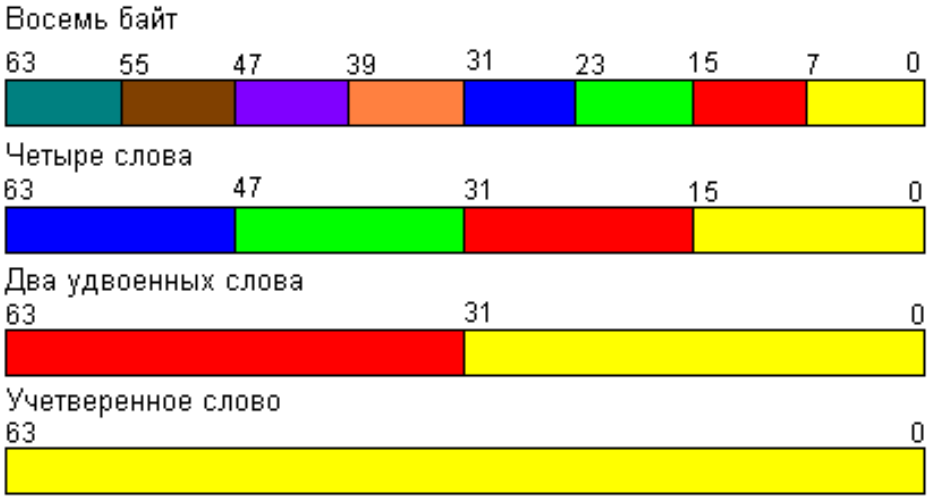
5. Расширение системы команд

MMX-расширение появилось в Pentium MMX (P55, январь 1997) и включало в себя 57 новых команд, предназначенных для обработки звуковых и видеосигналов. Позднее их поддержка появилась в K6 (Little Foot) от AMD и в 6x86MX от Cyrix.

MMX-расширение микропроцессора Pentium предназначено для поддержки приложений, ориентированных на работу с большими массивами данных целого типа, над которыми выполняются одинаковые операции. С данными такого типа обычно работают мультимедийные, графические, коммуникационные программы. По этой причине данное расширение архитектуры микропроцессоров Intel и названо *MultiMedia eXtensions* (MMX), что переводится как мультимедиа расширения.

5. Расширение системы команд

Основа программной компоненты – система команд MMX-расширения (те самые 57 новых команд) и четыре новых типа данных. MMX-команды являются естественным дополнением основной системы команд микропроцессора. Основным принципом их работы является одновременная обработка нескольких единиц однотипных данных одной командой. Основа аппаратной компоненты – 8 MMX регистров, каждый размером в 64 бит = 8 байт. MMX работает **только с целыми числами**; поддерживаются данные размером в 1, 2, 4 или 8 байт. То есть, один MMX регистр может содержать 8, 4, 2 или 1 операнд соответственно.



5. Расширение системы команд

На самом деле эти регистры не являются новыми, а MMX-расширение **использует регистры сопроцессора** (FPU). Как известно, регистры сопроцессора стека имеют размерность 80 бит, что касается MMX регистров, то их разрядность только 64 бита. Поэтому, когда регистры сопроцессора играют роль MMX-регистров, то доступными являются лишь их младшие 64 бита. К тому же, при работе стека сопроцессора в режиме MMX-расширения, он рассматривается не как стек, а как обычный регистровый массив с произвольным доступом. Таким образом, можно сказать, что расширения MMX реализованы в виде дополнительного режима, в который процессор может переключаться из обычного режима работы. Регистровый стек сопроцессора не может одновременно использоваться и по своему прямому назначению и как MMX-расширение, поэтому необходимо заботиться о его разделении и корректной работе с ним. Такое совмещение может снизить эффективность работы в случае попеременного использования обычных вычислений с плавающей точкой и работы в режиме MMX.

Данные, содержащиеся в MMX-регистрах, **можно покомпонентно складывать, умножать, вычитать**, выполнять разнообразные специфические, необходимые для мультимедиа приложений, операции, вроде сложения без переполнения, вычисления среднего арифметического и производить логические операции с битами (побитовый and, or, xor). **Делить, правда, нельзя**, есть ещё ограничения. Но многие операции можно делать на порядок быстрее, даже больше. Однако, применение MMX в особенности требует специальной ручной оптимизации, никакой компилятор тут существенно не поможет. Под MMX, например, оптимизируются разнообразные кодеки аудио файлов, алгоритмы работы которых хорошо сочетаются с MMX. Причём, не вся программа целиком, а небольшая часть, выполняющая основную работу, и это обстоятельство упрощает оптимизацию.

5. Расширение системы команд

Допустим, у нас есть два массива: A и B, длиной по 8 байтов каждый. Поставим себе задачу — прибавить к каждому элементу массива B соответствующий ему элемент массива A. Это приведет нас к такой программе:

```
                mov esi, A
                mov edi, B
                mov ecx, 8
OurLoop:
                mov al, [esi]
                add [edi], al
                inc esi
                inc edi
                loop OurLoop
```

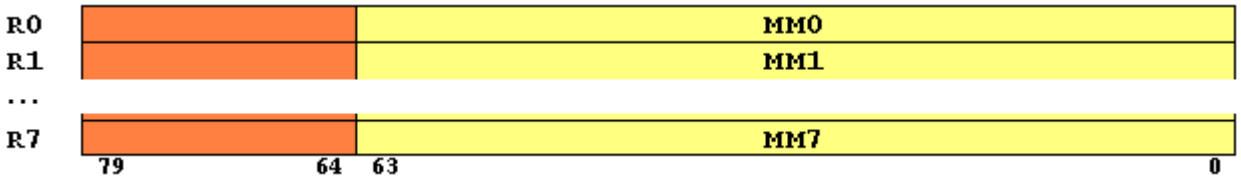
А это на MMX:

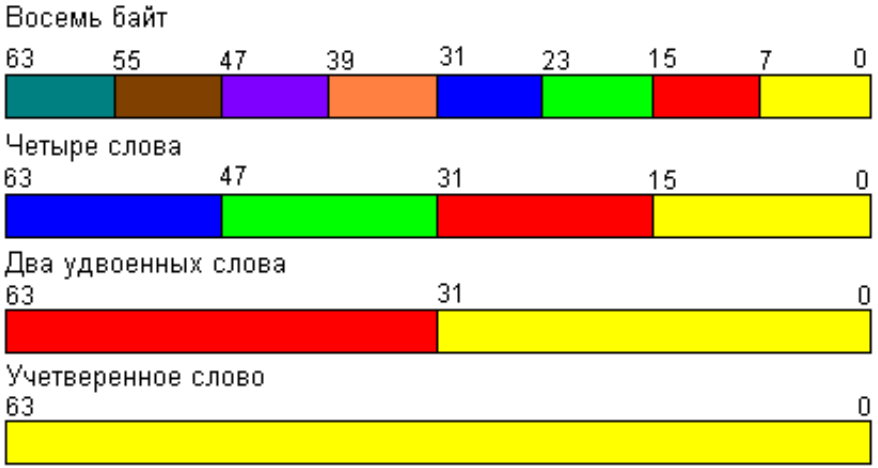
```
                movq mm0, [A]
                movq mm1, [B]
                paddb mm0, mm1
                movq [B], mm1
```

5. Расширение системы команд

Команды технологии MMX работают со следующими типами данных:

- упакованные байты (восемь байтов в одном 64-разрядном регистре)
- упакованные слова (четыре 16-разрядных слова в 64-разрядном регистре)
- упакованные двойные слова (два 32-разрядных слова в 64-разрядном регистре)
- 64-- разрядные слова.





5. Расширение системы команд

команды пересылки данных:

movd dst, src – пересылает 32-разрядные данные из памяти в регистры MMX и обратно или из целочисленных регистров процессора в регистры MMX и обратно;

movq dst,src – пересылает 64-разрядные упакованные данные из памяти в регистры MMX и обратно или между регистрами MMX;

Opcode/Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
MOVD mm, r/m32	A	V/V	MMX	Move doubleword from r/m32 to mm.
MOVQ mm, r/m64	A	V/N.E.	MMX	Move quadword from r/m64 to mm.
MOVD r/m32, mm	B	V/V	MMX	Move doubleword from mm to r/m32.
MOVQ r/m64, mm	B	V/N.E.	MMX	Move quadword from mm to r/m64.

арифметические команды:

padd[b, w, d, s[b,w], us[b,w]] - сложение элементов данных регистра. возможно насыщение; //s – насыщение знакового до максимального, us - …беззнакового

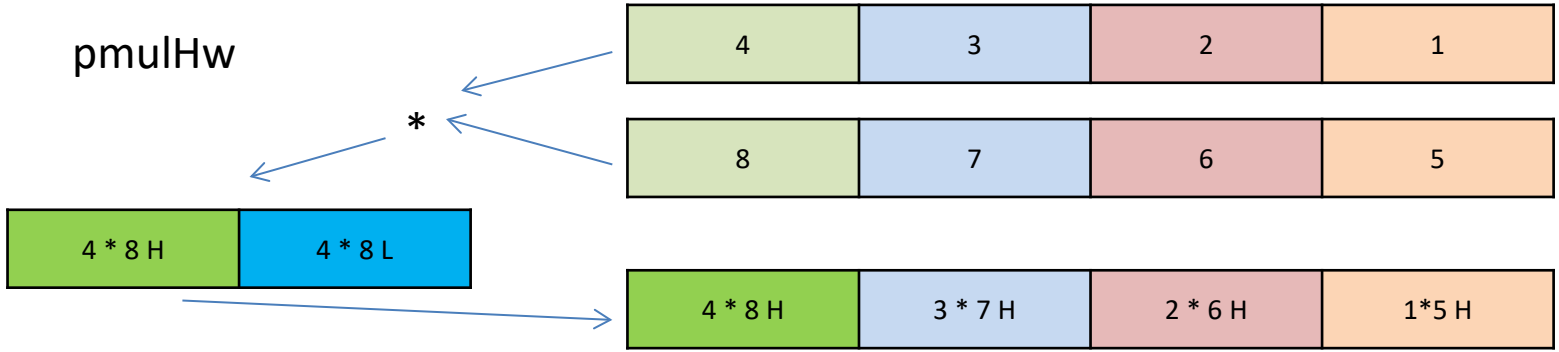
psub[b, w, d, s[b, w], us[b,w]] - вычитание упакованных элементов. Возможно насыщение; //s – насыщение знакового до максимального, us - …беззнакового

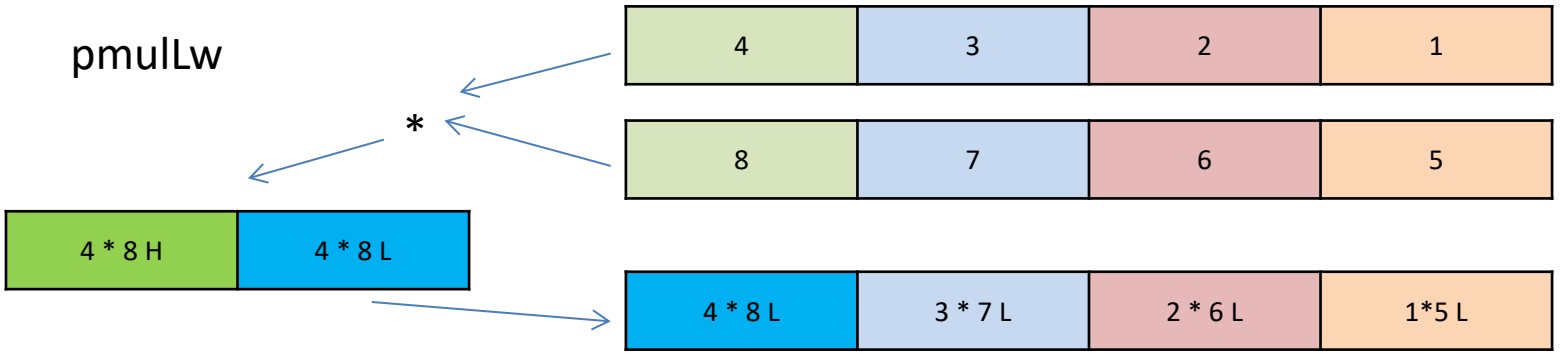
pmul[h,l]w - знаковое умножение упакованных слов. Возврат старших/младших слов результата;

punpckhwd - распаковка;

```
mov eax, 0x7e7F8081
movd mm0, eax
movd mm1, eax
paddb mm0, mm1 ; -nan(0x0fcfe0002)  - переполнение
movd mm0, eax
paddsb mm0, mm1; -nan(0x07f7f8080) – 7F и 80 = [127 и -128]
movd mm0, eax
paddusb mm0, mm1; -nan(0x0fcfefffff) – FF = 255
```

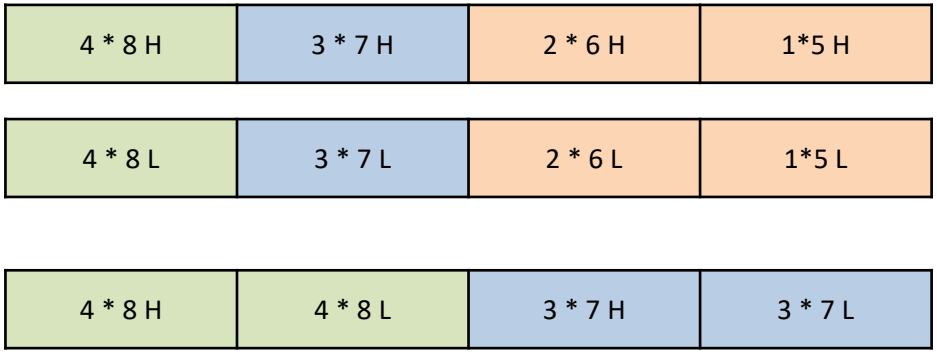
5. Расширение системы команд



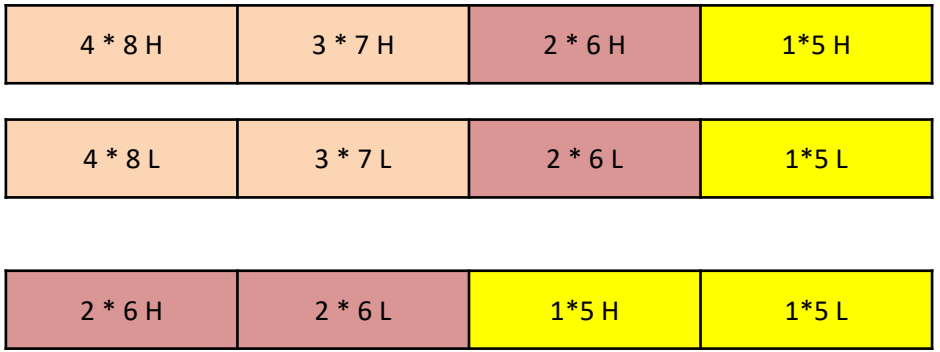


5. Расширение системы команд

punpckHw



punpckLw



5. Расширение системы команд

арифметические команды:

pmul[h,l]w - знаковое умножение упакованных слов. Возврат старших/младших слов результата;

punpckhwd - распаковка;

0x1111 * 0x1111 = 0x01234321

0x2222 *0x2222 = 0x048D0C84

A dq 0x2222000000001111

movq mm1,qword [a]; получаем 4 очередных элемента массива A (**word**), mm1= **0x2222000000001111**
movq mm0,qword [a]; получаем 4 очередных элемента массива B (**word**), mm0= **0x2222000000001111**
pmulhw mm0, mm1; mm0 = hi (A * B) mm0 = **0x048D000000000123**
movq mm2,qword [a]; mm2= 0x2222000000001111
pmullw mm1, mm2; mm1 = low (A * B) mm1 = **0x0C84000000004321**
movq mm2,mm0 ; mm2 = 0x048D000000000123
movq mm3,mm1; mm3 = **0x0C84000000004321**
punpckhwd mm3, mm2; объединяем mm3 = **0x048D0C8400000000**
punpcklwd mm1, mm0; объединяем mm1 = **0x0000000001234321**

5. Расширение системы команд

SSE появилось в Pentium III (ядро Katmai, сентябрь 1999) и насчитывало 70 новых команд. Позднее в Athlon XP (начиная с Palomino) его стали поддерживать и процессоры AMD. Аббревиатура SSE расшифровывается как *Streaming SIMD Extensions* (потоковые SIMD расширения).

SSE интересно прежде всего тем, что оперирует с данными **вещественного** типа одинарной (**single**) точности, которые используются в геометрических расчётах, то есть, приложениях трёхмерной графики, компьютерных играх, редакторах вроде 3DStudioMax, и многих других. С тех пор как в компьютерных играх вроде Quake текстурирование треугольников стало производиться при помощи видеоускорителей, большая надобность в целочисленных вычислениях отпала. На первое место вышла скорость операций с плавающей точкой, вроде перемножения вещественного вектора на вещественную матрицу.

При внедрении SSE процессор получил в дополнение к стандартным регистрам архитектуры x87 8 новых больших регистров размером по 128 бит, в каждом из которых содержится 4 32-битных вещественных числа. С четвёрками операндов можно покомпонентно производить следующие операции: сложить две четвёрки чисел, вычесть, перемножить, разделить. Вычислить одновременно 4 (обратных) квадратных корня, точно или приближённо. Ещё можно тасовать содержимое регистров, перекладывать данные из одних частей регистра в другие и производить некоторые другие аналогичные операции. Однако перемещение данных происходит не быстрее их сложения, так что эффективное использование SSE возможно только на подготовленных правильно упакованных данных.

5. Расширение системы команд

SSE2. Следующее расширение, являющееся логическим продолжением MMX и SSE появилось в Pentium 4 (начиная с Willamette). В Athlon 64 появилось начиная с Clawhammer.

В данное расширение включены 144 команды SSE2, ориентированные, в первую очередь, на работу с потоковыми данными. Подобно Pentium III, они также оперируют со 128-битными регистрами, но уже не только с четверками чисел одинарной точности, но и с любыми другими типами данных, которые умещаются в 128 бит. Это пары вещественных чисел двойной точности, шестнадцать однобайтовых целых, восьмерки двухбайтовых целых, пары восьмибайтовых целых etc. В результате получился некий симбиоз MMX и SSE.

SSE3 . Следующий набор появился в Pentium 4 начиная с Prescott и Athlon 64 начиная с Venice. Это расширение, имевшее поначалу имело рабочее название *Prescott New Instruction*, но получившее в итоге не совсем верное с технической точки зрения название SSE3, призвано облегчить оптимизацию программ под SSE и SSE2. Причём, в первую очередь, сделать более легкой полностью автоматическую оптимизацию программ средствами компилятора. То есть, для оптимизации необходимо будет просто перекомпилировать программу.

Некорректность названия SSE3 объясняется тем, что в отличие от других SIMD инструкций, где операции (например сложение) выполняются **вертикально** (между регистрами), здесь появилась возможность **горизонтального** выполнения операций (внутри одного регистра)

5. Расширение системы команд

```
a dd 300.0,   4.0,   4.0,   12.0
b dd    1.5,   2.5,   3.5,    4.5

movups xmm0, [a]
movups xmm1, [b]

mulps xmm0, xmm1

movups [a], xmm0
```

Movups – move Unaligned Packed Single

mulps – Multiply Packed Single

Переменная или выражение	Значение	Тип
a	{450,10,14,54}	Float d 4
Добавить...		Smart d Размер массива

xmm0		0x42580000416000004120000043e10000
xmm1		0x40900000406000004020000003fc00000

5. Расширение системы команд

Расширение	Регистр	Размер	Кол-во	Примечание
MMX	MM0-MM7	64	8	Целочисленные вычисления Скаляры и упакованные числа
SSE	XMM0-XMM7 (XMM15)	128	8 (16 в x64)	+ Вещественные числа одинарной точности
SSE2	XMM0-XMM7	128	8	+ Вещественные числа двойной точности
SSE3	XMM0-XMM15	128	16	+ «Горизонтальные» операции
SSE4	XMM0-XMM15	128	16	+ Доп.команды повышения быстродействия
AVX/ AVX2	YMM0-YMM15	256	16	Выравнивание на 32 байта (желательно) Расширенная поддержка 128 битных целых чисел Неразрушающие команды (без модификации регистров источников данных)
AVX-512	ZMM0-ZMM15	512	32	Быстродействие