Echtzeit–Videoverarbeitung

x86–Assembler Programmierung
MMX

Marcus Barkowsky

Überblick

Nachträge zu x86–Befehlssatz

Assembler
- Pseudo Operationen

MMX–Erweiterung
- Register
- Befehle
- Beispiele

Optimierung für P6–Prozessoren
- Sprünge
- Pipelining
- Cache
- Diverses

Beispiele

SETcc: Set Byte on Condition

Verwendung:

Syntax	Beispiel
setcc r/m8	setcc al

Aufgabe:

Das angegebene Byte Register oder das Byte im Speicher wird mit 1 geladen, wenn die zugehörige Bedingung zutrifft, sonst mit 0. Die Bedingungen sind wie bei Jcc.

Flags:

Keine Änderung

CMOVcc: Conditional Move

Verwendung:

Syntax	Beispiel
cmovcc r16,r16	cmovae ax,bx
cmovcc r16,m16	cmovle ax,[WVar]
cmovcc r32,r32	cmovnz eax,ebx
cmovcc r32,m32	cmovz eax,[Wvar]

Aufgabe:

Wenn die angegebene Bedingung zutrifft, verhält sich der Befehl wie ein gewöhnlicher MOV, sonst wie ein NOP (no operation). Die Bedingungen sind wie bei Jcc

Flags:

Keine Änderung

CMPXCHG: Compare and Exchange

Verwendung:

Syntax	Beispiel
cmpxchg r/m,reg	cmpxchg ebx,ecx

Aufgabe:

Dieser Befehl führt einen Vergleich des AL/AX/EAX Registers mit dem ersten Operanden durch und setzt das ZF entsprechend. Falls AL/AX/EAX gleich dem 1. Operanden ist, wird der 2. Operand in den 1. Operanden geladen (wie MOV), sonst wird der 2. Operand in das AL/AX/EAX Register geladen.
cmpxchg ebx,ecx: if(eax==ebx) ebx=ecx; else eax=ebx;

Flags:

ZF wird entsprechend dem Vergleich gesetzt

CMPXCHG8B: Compare and Exchange 8 Bytes

Verwendung:

Syntax	Beispiel
cmpxchg8b m64	cmpxchg [Qvar]

Aufgabe:

Der 64 Bit Wert in EDX:EAX wird mit dem Operanden verglichen. Falls sie gleich sind, wird der 64 Bit Wert in ECX:EBX (!) im Operanden gespeichert, sonst wird EDX:EAX mit dem Operanden geladen.
if(EDX:EAX==*m64) *m64=(ECX:EBX); else EDX:EAX=*m64;

Flags:

ZF wird entsprechend dem Vergleich gesetzt

Pseudo−Operationen des nasm

Aufgaben:

- Assembler:
  - Lokalisierung von Assembler−Code
  - Festlegung von Konstanten
  - Deklaration von Variablen

- Präprozessor
  - Einbindung anderer Dateien
  - Definition von Strukturen
  - Verwendung von symbolischen Ausdrücken
  - Makros für häufig benutzte Funktionsblöcke

Konstanten und Operatoren

Zahlenformate:

- Hexadezimal:

```
mov ax,a2h
mov ax,$0a2   ; die 0 ist notwendig
mov ax,0xa2
```

- Oktal:

```
mov ax,777q
```

- Binär:

```
mov ax,10010011b
```

Operatoren:

C−Nomenklatur: | ^ & ~ << >> + − * / %

zusätzlich: // %% für Division/Modulo vorzeichenbehafteter Zahlen

Labels

```
label1:   mov ax,0      ; Globales Label
.loop:    mov bx,ax     ; Lokales Label unter label1
.test:                  ; ebenso
          jne .loop     ; Springt nach label1.loop
label2:   mov bx,0      ; Zweites globales Label
.loop:    mov ax,bx     ; lokales Label von label2
          jne .loop     ; springt nach label2.loop
          jmp label1.loop  ; Sprung nach oben
          jmp .test     ; nicht zulässig, da nicht im label2 Kontext
```

SECTION: Auswahl der aktuellen Section

Verwendung:

Syntax	Beispiel
SECTION <name>	SECTION .data

Aufgabe:

Alle folgenden Code−Zeilen werden in die betreffende Sektion geschrieben.

Verfügbare Segmente/Sektionen:

- .text: Programmcode und unter bestimmten Umständen vorinitialisierte Konstanten. Nicht beschreibbar (read−only)
- .data: Vorinitialisierte Variablenbereiche, überschreibbar
- .bss: Nicht initialisierte Speicherbereiche

DB/DW/DD/DQ/DT: Declaring initialised data

Verwendung:

Syntax	Beispiel
db <byte_data>	db 0x55,0x56,'a'
dw <word_data>	dw 'ond',27,0xf0
dd <doublewords>	dd 0x32948290
dq <double_nr>	dq 1.9283e20
dt <extprec_nr>	dt 29384.2938e10

Aufgabe:

An dieser Stelle im Programmcode wird ein Byte/Wort/Doppelwort (db/dw/dd) oder ein Gleitkommawert mit einfacher, doppelter oder erweiterter Präzision (dd/dq/dt) eingefügt. Falls Zeichenketten angegeben werden, so werden diese bei Bedarf mit Nullen aufgefüllt (Beispiel: dw 'abc' = db 0x41,0x42,0x43,0x0). Es findet allerdings keine automatische Null−Terminierung von Zeichenketten statt, wie sie für C−Funktionen benötigt werden, daher stets Null−Byte anhängen. Diese Daten sollten in das .data Segment geschrieben werden.

RESB/RESW/RESD/RESQ/REST:Declaring uninitialised data

Verwendung:

Syntax	Beispiel
resb <count>	resb 64
resw <count>	resw 1
resd <count>	resd 12
resq <count>	resq 10
rest <count>	rest 16

Aufgabe:

Mit diesen Anweisungen wird Speicher für die übergebene Anzahl Bytes/Worte/Doppelworte (resb/resw/resd), bzw. Floating−Point Zahlen/Extended precision (resq/rest) reserviert.
Diese Anweisung sollte im .bss Segment stehen.

TIMES: Repeating instructions or data

Verwendung:

Syntax	Beispiel
times <count> <instruction>	times 64 db 0

Aufgabe:

Das Prefix times wiederholt schreibt <count> Wiederholungen des nachfolgenden Befehls.

Beispiel: times 100 resb 1 ist identisch mit resb 100

Beispiel: times 64 db 0 schreibt 64 Nullen an die aktuelle Stelle

Besondere Form: $ gibt die aktuelle Position im Code an und kann zusammen mit Labels zur Differenzberechnung verwendet werden.

```
buffer:   db 'hello, world'
          times 64-$+buffer db ' '
```

Wiederholt db ' ' so lange, bis 64 Bytes aufgefüllt wurden

EQU: Defining Constants

Verwendung:

Syntax	Beispiel
label: equ <operand>	const64: equ 64

Aufgabe:

Mit equ werden Labels auf festzulegende Werte gesetzt, anstelle den aktuellen Programmcounter anzunehmen.
Besondere Form: $ gibt die aktuelle Position im Code an und kann zusammen mit Labels zur Differenzberechnung verwendet werden.

```
buffer:     db 'hello, world'
bufferlen:  equ $-buffer
```

EXTERN: Importing Symbols from Other Modules

Verwendung:

Syntax	Beispiel
extern <label>	extern _printf

Aufgabe:

Definiert ein Symbol, welches nicht im aktuellen Assembler−Programm definiert wird, aber verwendet werden soll.

GLOBAL: Exporting Symbols to Other Modules

Verwendung:

Syntax	Beispiel
global <label>	global asm_memadd

Aufgabe:

Definiert ein Symbol, welches im aktuellen Assembler−Programm definiert wird und außerhalb dieses Kontextes Verwendung finden soll.

%include: Including other files

Verwendung:

Syntax	Beispiel
%include filename	%include "asm_c.mac"

Aufgabe:

Die angegebene Datei wird an dieser Stelle in den Programmtext eingefügt.

%define: Single−Line Macros

Verwendung:

Syntax	Beispiel
%define fct(par) instr par	%define madd(a,b) a+b

Aufgabe:

Verhält sich wie die C−Präprozessoranweisung #define:
Es können Konstanten oder Funktionen vordefiniert werden, die im Assembler−Programm ersetzt werden. Es findet eine Textersetzung statt, bevor der Code compiliert wird. Die Definition endet am Zeilenende.
Beispiele:
%define PI 3.147
%define a(x1) mov ax,x1

%assign: Preprocessor Variables

Verwendung:

Syntax	Beispiel
%assign varname value	%assign i i+1

Aufgabe:

Definiert eine Variable, mit der arithmetische Operationen möglich sind. Auch hier findet die Ersetzung vor der Compilation statt.

%macro: Multi−Line Macros

Verwendung:

Syntax	Beispiel
%macro name parcount < some code > %endmacro	%macro pro 1 sub ebp,%1 %endmacro

Aufgabe:

Ähnlich wie %define, es können allerdings mehrere Zeilen verwendet werden. Die Parameterübergabe erfolgt mittels Nummern. Lokale Labels werden mit 2 Prozentzeichen geschrieben. Weitere Einzelheiten siehe nasm−Manual

Beispiel:

```
%macro mymacro 2
                mov ecx,%1          ; ersten Parameter in ecx laden
%%myloop:       loop %%myloop       ; Makro−lokales Label, wird bei jeder Makro−
                                    ; expansion mit neuem Wert belegt
.%{2}1goon:                         ; Schutz für 2.Makroparameter mit {}
%endmacro
```

%if: Conditional Assembly

Verwendung:

Syntax	Beispiel

%if condition
  < some code >
%elif condition
  < some code >
%else
  < some code >
%endif
	%if myvalue==0
	  mov eax,42
	%else
	  mov ecx,27
	%endif

Aufgabe:

Bedingte Übersetzung von Programmcode. Besonders sinnvoll in Makros einsetzbar, um in Abhängigkeit eines Parameters zu arbeiten.
Die Ersetzung findet vor der Compilation statt. Einzelheiten siehe nasm−Manual.

%rep: Preprocessor loops

Verwendung:

Syntax	Beispiel
%rep count < some code > %endrep	%rep 64 stosb %endrep

Aufgabe:

Schreibe den nachfolgenden Text mit count Wiederholungen. Verhält sich ähnlich wie times, wird jedoch bereits vom Präprozessor ausgeführt.

Beispiel zum Präprozessor:

```
section .data

%macro CLIP_Y 1
%if %1<16
      db 16
%elif %1>240
      db 240
%else
      db %1
%endif
%endmacro


clip_char_table_y:
%assign i 0
%rep 256
CLIP_Y i
%assign i i+1
%endrep
```

Präprozessor

```
clip_char_table_y:

%line 382+1 asm_memadd.asm
      db 16
%line 382+0 asm_memadd.asm
      db 16
      db 16

      ...

      db 16
      db 16
      db 16
      db 17
      db 18

      ...

      db 237
      db 238
      db 239
      db 240
      db 240

      ...

      db 240
      db 240
      db 240
%line 383+1 asm_memadd.asm
```

Assembler

```
0003d0 10 10 10 10 10 10 10 10
0003d8 10 10 10 10 10 10 10 10
0003e0 10 11 12 13 14 15 16 17
0003e8 18 19 1a 1b 1c 1d 1e 1f
0003f0 20 21 22 23 24 25 26 27
0003f8 28 29 2a 2b 2c 2d 2e 2f
000400 30 31 32 33 34 35 36 37
000408 38 39 3a 3b 3c 3d 3e 3f
000410 40 41 42 43 44 45 46 47
000418 48 49 4a 4b 4c 4d 4e 4f
000420 50 51 52 53 54 55 56 57
000428 58 59 5a 5b 5c 5d 5e 5f
000430 60 61 62 63 64 65 66 67
000438 68 69 6a 6b 6c 6d 6e 6f
000440 70 71 72 73 74 75 76 77
000448 78 79 7a 7b 7c 7d 7e 7f
000450 80 81 82 83 84 85 86 87
000458 88 89 8a 8b 8c 8d 8e 8f
000460 90 91 92 93 94 95 96 97
000468 98 99 9a 9b 9c 9d 9e 9f
000470 a0 a1 a2 a3 a4 a5 a6 a7
000478 a8 a9 aa ab ac ad ae af
000480 b0 b1 b2 b3 b4 b5 b6 b7
000488 b8 b9 ba bb bc bd be bf
000490 c0 c1 c2 c3 c4 c5 c6 c7
000498 c8 c9 ca cb cc cd ce cf
0004a0 d0 d1 d2 d3 d4 d5 d6 d7
0004a8 d8 d9 da db dc dd de df
0004b0 e0 e1 e2 e3 e4 e5 e6 e7
0004b8 e8 e9 ea eb ec ed ee ef
0004c0 f0 f0 f0 f0 f0 f0 f0 f0
0004c8 f0 f0 f0 f0 f0 f0 f0 f0
```

STRUC: Declaring structure data types

Verwendung:

Syntax	Beispiel
struc structuretype label1: RESs count label2: RESs count endstruc	struc mytype .input: resd 1 .output: resw 1 .str: resb 32 endstruc

Aufgabe:

Definiert die Offsets innerhalb einer Struktur. Verhält sich ähnlich wie equ, allerdings kann mit den res−Befehlen die Datengröße der einzelnen Felder angegeben werden, anstatt die Offsets direkt anzugeben. In obigem Beispiel ist anschließend z.B. mytype.str mit 4+2=6 definiert. Darüberhinaus wird die Länge der Struktur in mytype_size gespeichert.

Vorsicht: Es handelt sich um Offset−Konstanten und nicht, wie bei C üblich um Adressen real vorhandener Strukturen.

ISTRUC: Declaring instances of structures

Verwendung:

Syntax	Beispiel
structure: istruc structuretype at member, dd value iend	mystruct: istruct mytype at .input dw, 1234567 at .output dd, 127 at .str, db 'hello, world',13,10,0 iend

Aufgabe:

Reserviert Speicher für eine Struktur und belegt diese vor. Der Zugriff erfolgt dann beispielsweise mit mov eax,[mystruct+mystruct.output]

ALIGN: Data Alignment

Verwendung:

Syntax	Beispiel
align boundary	align 4
align boundary, command	align 4, db 0
alignb boundary	alignb 4

Aufgabe:

Reserviert soviel Speicher, daß die nächste Codezeile auf einer Speichergrrenze mit dem angegebenen Wert liegt. Falls kein command angegeben wird, werden NOPs (no operation) eingefügt, sonst wird der angegebene Befehl verwendet. Die alignb−Version des Befehls ist für das nicht−initialisierte Datensegment geeignet und entspricht einem align boundary, resb 1.

MMX−Übersicht

Register:
- mm0..mm7: 64 Bit Register

Aufteilung der Register in
- B: 8 Bytes (8x8)
- W: 4 Words (4x16)
- D: 2 Doublewords (2x32)

Saturation:
- S: default: signed saturation (clipping auf −128, +127)
- US: unsigned saturation (clipping auf 255,0)
- : kein Clipping

Sonstiges:
- Keine Floating−Point Operationen parallel
- MMX−Sequenz stets mit EMMS (Empty Multimedia State) abschließen

EMMS: Empty Multimedia State

Verwendung:

Syntax	Beispiel
emms	emms

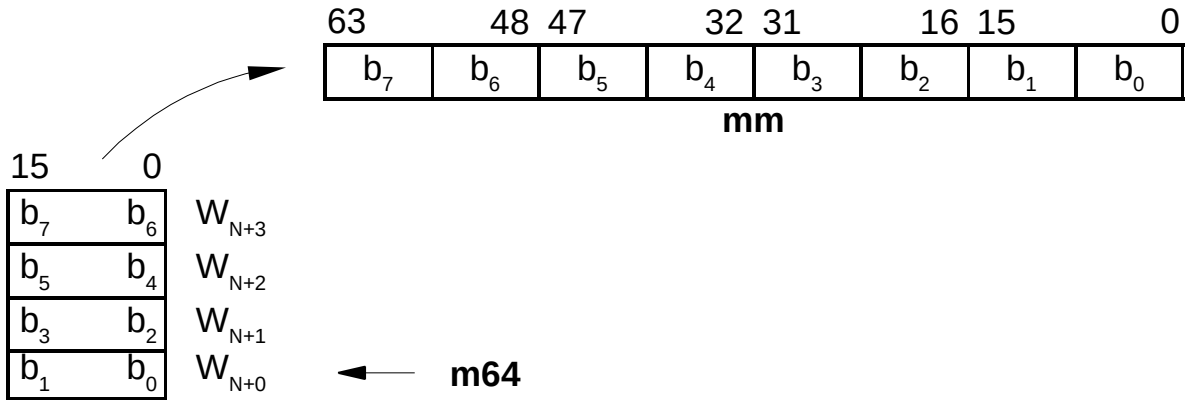
Aufgabe:

Freigabe der Floating−Point Einheit nach Abarbeitung von MMX−Befehlen. Muß stets am Ende jeden Assembler−Quelltextes, der MMX verwendet aus−geführt werden.

MOVQ: Move 64 Bits

Verwendung:

Syntax	Beispiel
movq mm,mm/m64	movq mm0,[Qvar]
movq mm/m64,mm	movq [QVar],mm1



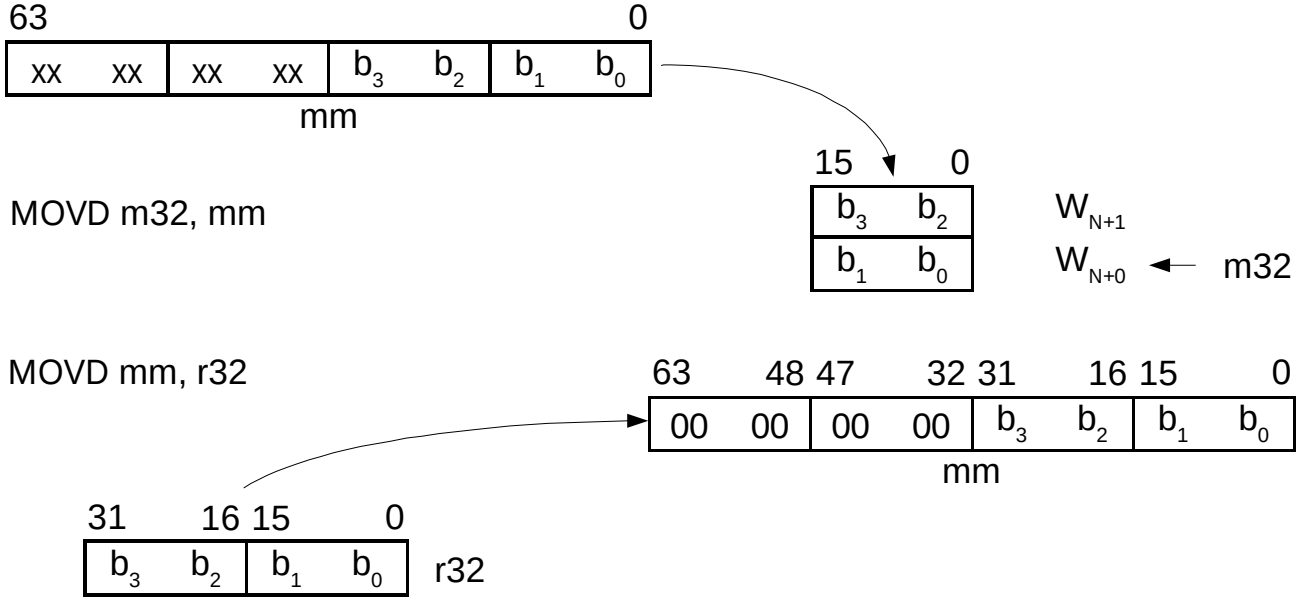
Aufgabe:

Kopieren von Registern untereinander oder von Speicher in Register, jeweils 64 Bit breit.

MOVD: Move 32 Bits

Verwendung:

Syntax	Beispiel
movd mm,r/m32	movd mm0,[Wvar]
movd mm/m32,mm	movd eax,mm1



Aufgabe:

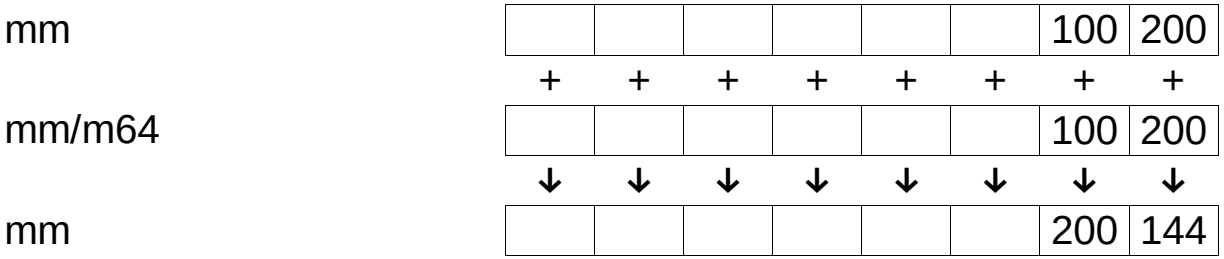
Speichert die unteren 32 Bit eines MMX−Register in 32 Bit Register oder Speicherzellen, bzw. liest in die unteren 32 Bit eines MMX−Register Werte aus 32 Bit Registern oder Speicherzellen ein; der Rest des Registers wird 0 gesetzt.

PADDB/W/D: Packed Add

Verwendung:

Syntax	Beispiel
paddb mm,mm/m64	paddb mm0,mm1
paddw mm,mm/m64	paddw mm0,mm1
paddd mm,mm/m64	paddd mm0,[QVar]

PADDB mm,mm/m64



Aufgabe:

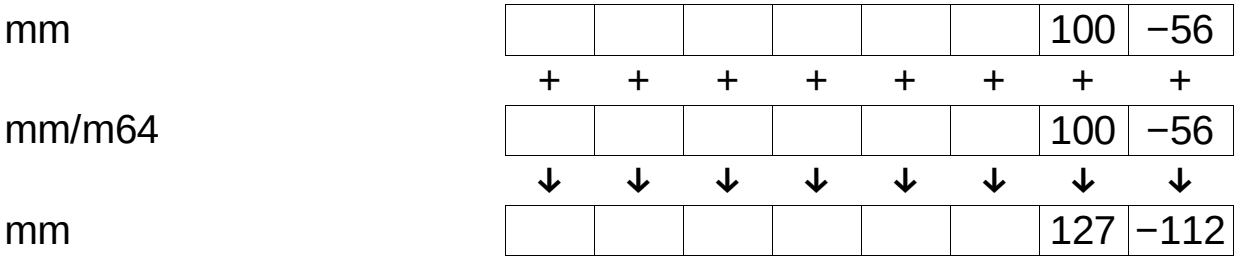
Führt eine Addition aller 8 Bytes (PADDB), aller 4 Wörter (PADDW), bzw. beider Doppelwörter (PADDD) durch.

PADDSB/W: Packed Add with Saturation

Verwendung:

Syntax	Beispiel
paddsb mm,mm/m64	paddsb mm0,mm1
paddsw mm,mm/m64	paddsw mm0,mm1

PADDSB mm,mm/m64



Aufgabe:

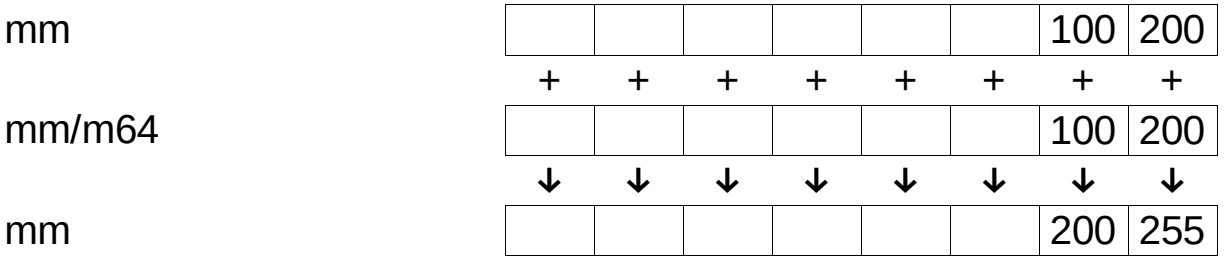
Führt eine Addition aller 8 Bytes (PADDSB), bzw. aller 4 Wörter (PADDSW) durch. Das Ergebnis wird auf [−128;127] saturiert.

PADDUSB/W: Packed Add Unsigned with Saturation

Verwendung:

Syntax	Beispiel
paddusb mm,mm/m64	paddusb mm0,mm1
paddusw mm,mm/m64	paddusw mm0,mm1

PADDUSB mm,mm/m64



Aufgabe:

Führt eine Addition aller 8 Bytes (PADDUSB), bzw aller 4 Wörter (PADDUSW) durch. Das Ergebnis wird auf [0;255] saturiert.

PSUBB/W/D: Packed Subtract

Verwendung:

Syntax	Beispiel
psubb mm,mm/m64	psubb mm0,[Qvar]
psubw mm,mm/m64	psubw mm0,[Qvar]
psubd mm,mm/m64	psubd mm0,[Qvar]

Aufgabe:

Führt eine Subtraktion aller 8 Bytes (PSUBB), aller 4 Wörter (PSUBW), bzw. beider Doppelwörter (PSUBD) durch.

PSUBSB/W: Packed Subtract with Saturation

Verwendung:

Syntax	Beispiel
psubsb mm,mm/m64	psubsb mm0,[Qvar]
psubsw mm,mm/m64	psubsw mm0,[Qvar]

Aufgabe:

Führt eine Subtraktion aller 8 Bytes (PSUBSB) oder aller 4 Wörter (PSUBSW) durch. Das Ergebnis wird auf [−128;127] saturiert.

PSUBUSB/W: Packed Subtract Unsigned with Saturation

Verwendung:

Syntax	Beispiel
psubusb mm,mm/m64	psubusb mm0,[Qvar]
psubusw mm,mm/m64	psubusw mm0,[Qvar]

Aufgabe:

Führt eine Subtraktion aller 8 Bytes (PSUBUSB) oder aller 4 Wörter (PSUBUSW) durch. Das Ergebnis wird auf [0;255] saturiert.

PAND: Bitwise Logical And

Verwendung:

Syntax	Beispiel
pand mm,mm/m64	pand mm0,[Qvar]

Aufgabe:

Wendet ein logisches UND auf alle 2/4/8 Doppelworte/Worte/Bytes an.

PANDN: Bitwise Logical AND NOT

Verwendung:

Syntax	Beispiel
pandn mm,mm/m64	pandn mm0,[Qvar]

Aufgabe:

Invertiert alle Bits des 1.Operanden und wendet anschließend ein logisches UND auf alle 2/4/8 Doppelworte/Worte/Bytes an.

POR: Bitwise Logical Or

Verwendung:

Syntax	Beispiel
por mm,mm/m64	por mm0,[Qvar]

Aufgabe:

Wendet ein logisches Oder auf alle 2/4/8 Doppelworte/Worte/Bytes an.

PXOR: Bitwise Logical XOR

Verwendung:

Syntax	Beispiel
pxor mm,mm/m64	pxor mm0,[Qvar]

Aufgabe:

Wendet ein logisches Exklusiv−Oder auf alle 2/4/8 Doppelworte/Worte/Bytes an.

PSLLW/D/Q: Packed Shift Left Logical

Verwendung:

Syntax	Beispiel
psslw mm,mm/m64	psslw mm0,[Qvar]
psslw mm,imm8	psslw mm1,7
pssld mm,mm/m64	pssld mm0,[Qvar]
pssld mm,imm8	pssld mm1,7
psslq mm,mm/m64	psslq mm0,[Qvar]
psslq mm,imm8	psslq mm1,7

Aufgabe:

Shifted alle 2/4/8 Doppelworte/Worte/Bytes um die angegebene Anzahl nach links. Es werden Nullen nachgeschoben.

PSRLW/D/Q: Packed Shift Right Logical

Verwendung:

Syntax	Beispiel
psrlw mm,mm/m64	psrlw mm0,[Qvar]
psrlw mm,imm8	psrlw mm1,7
psrld mm,mm/m64	psrld mm0,[Qvar]
psrld mm,imm8	psrld mm1,7
psrlq mm,mm/m64	psrlq mm0,[Qvar]
psrlq mm,imm8	psrlq mm1,7

Aufgabe:

Shifted alle 2/4/8 Doppelworte/Worte/Bytes um die angegebene Anzahl nach rechts. Es werden Nullen nachgeschoben.

PSRAW/D/Q: Packed Shift Right Arithmetic

Verwendung:

Syntax	Beispiel
psraw mm,mm/m64	psraw mm0,[Qvar]
psraw mm,imm8	psraw mm1,7
psrad mm,mm/m64	psrad mm0,[Qvar]
psrad mm,imm8	psrad mm1,7
psraq mm,mm/m64	psraq mm0,[Qvar]
psraq mm,imm8	psraq mm1,7

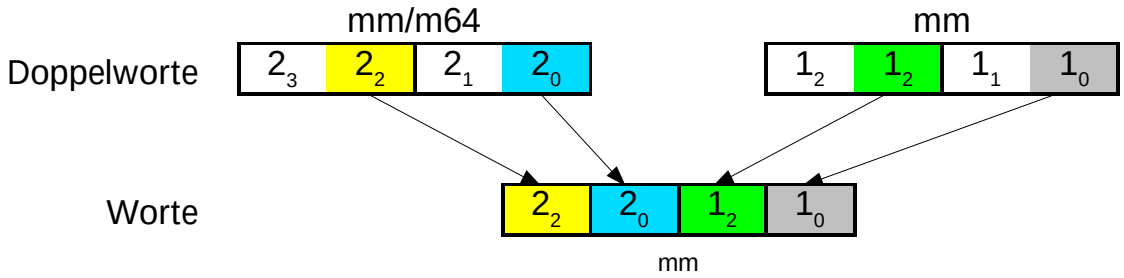
Aufgabe:

Shifted alle 2/4/8 Doppelworte/Worte/Bytes um die angegebene Anzahl nach rechts. Das Vorzeichenbit wird repliziert.

PACKSSWB/PACKSSDW: Pack with Signed Saturation

Verwendung:

Syntax	Beispiel
packsswb mm,mm/m64	packsswb mm0,[Qvar]
packssdw mm,mm/m64	packssdw mm0,[Qvar]



Aufgabe:

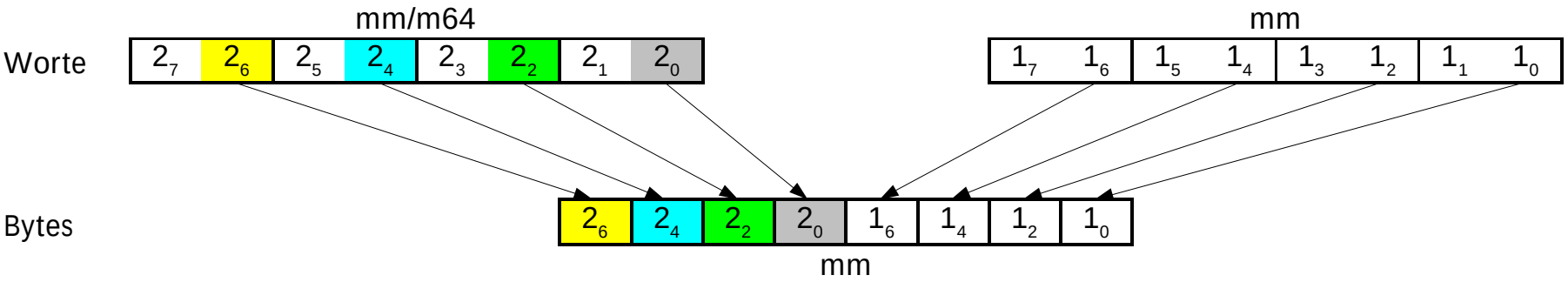
Ausgangspunkt sind die beiden angegebenen MMX−Register. Es wird nun eine Datenreduktion auf die Hälfte vorgenommen, in der mit Signed Saturation gearbeitet wird, z.B. wird bei PACKSSWB je ein Word zu einem Byte umge−wandelt; war das Wort größer als 127, so wird das Byte auf 127 gesetzt.
Das Ergebnis paßt in 1 MMX−Register: den 1.Operanden.

PACKUSWB: Pack with unsigned Saturation

Verwendung:

Syntax	Beispiel
packsuswb mm,mm/m64	packsuswb mm0,[Qvar]

PACKUSWB mm, mm/m64



Aufgabe:

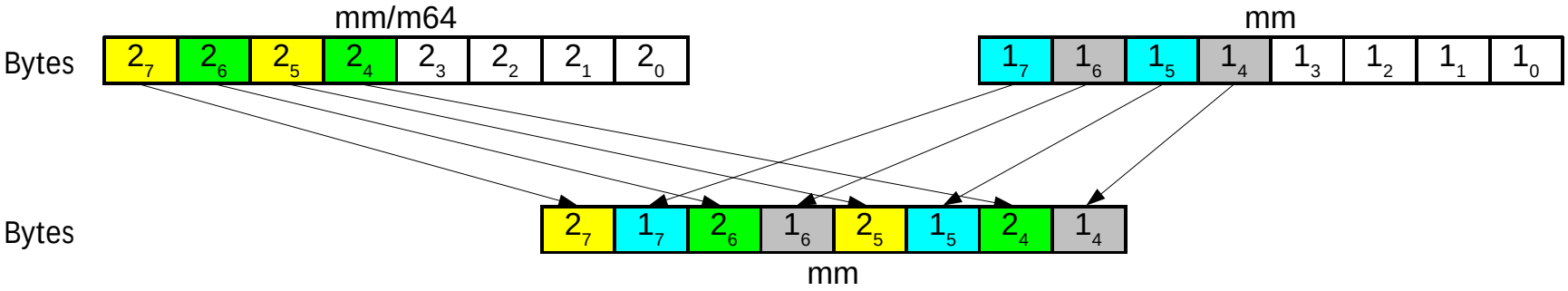
Wie PACKSSWB, nur werden Eingangswerte von −32768 bis 0 auf 0 gesetzt und Werte von 255−32767 auf 255. Der Eingang wird also vorzeichenbehaftet interpretiert.

PUNPCKHBW/WD/DQ: Unpack High Packed Data

Verwendung:

Syntax	Beispiel
punpckhbw mm,mm/m64	punpckhbw mm0,[Qvar]
punpckhwd mm,mm/m64	punpckhwd mm0,[Qvar]
punpckhdq mm,mm/m64	punpckhdq mm0,[Qvar]

PUNPCKHBW mm, mm/m64



Aufgabe:

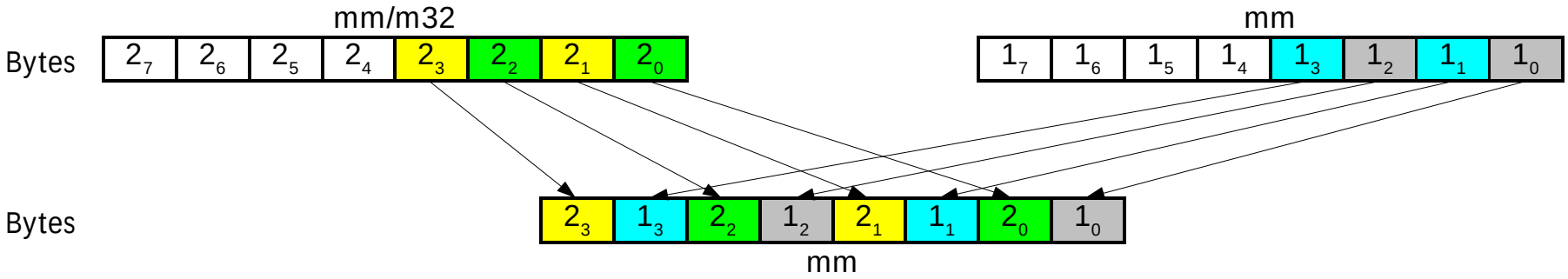
Bei PUNPCKHBW werden die vier höchsten Bytes der beiden angegebenen Operanden verwendet. Sie werden abwechselnd aus dem einen und dem anderen Operanden ausgelesen und in das Zielregister gespeichert. Die anderen beiden Befehle verfahren analog mit Words oder Doublewords.

PUNPCKLBW/WD/DQ: Unpack Low Packed Data

Verwendung:

Syntax	Beispiel
punpcklbw mm,mm/m64	punpcklbw mm0,[Qvar]
punpcklwd mm,mm/m64	punpcklwd mm0,[Qvar]
punpckldq mm,mm/m64	punpckldq mm0,[Qvar]

PUNPCKLBW mm, mm/m32



Aufgabe:

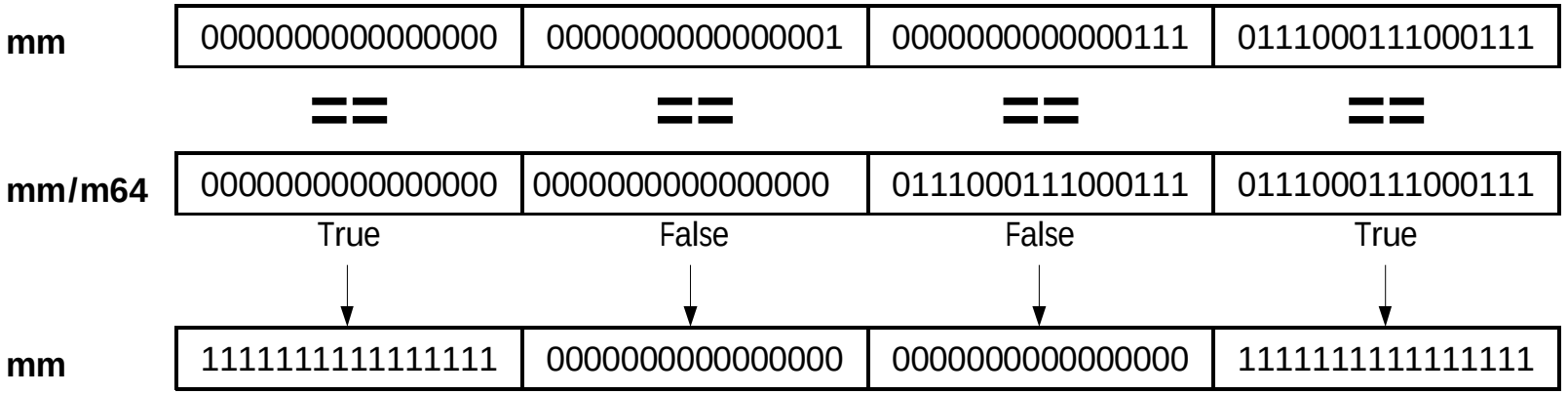
Verhält sich wie PUNPCKHBW, nur werden in diesem Falle die unteren Bytes/Wörter/Doppelwörter verwendet, um alternierend in das Zielregister geschrieben zu werden.

PCMPEQB/W/D: Packet Compare for equal

Verwendung:

Syntax	Beispiel
pcmpeqb mm,mm/m64	pcmpeqb mm0,[Qvar]
pcmpeqw mm,mm/m64	pcmpeqw mm0,mm1
pcmpeqd mm,mm/m64	pcmpeqb mm0,[Qvar]

PCMPEQW mm, mm/m64



Aufgabe:

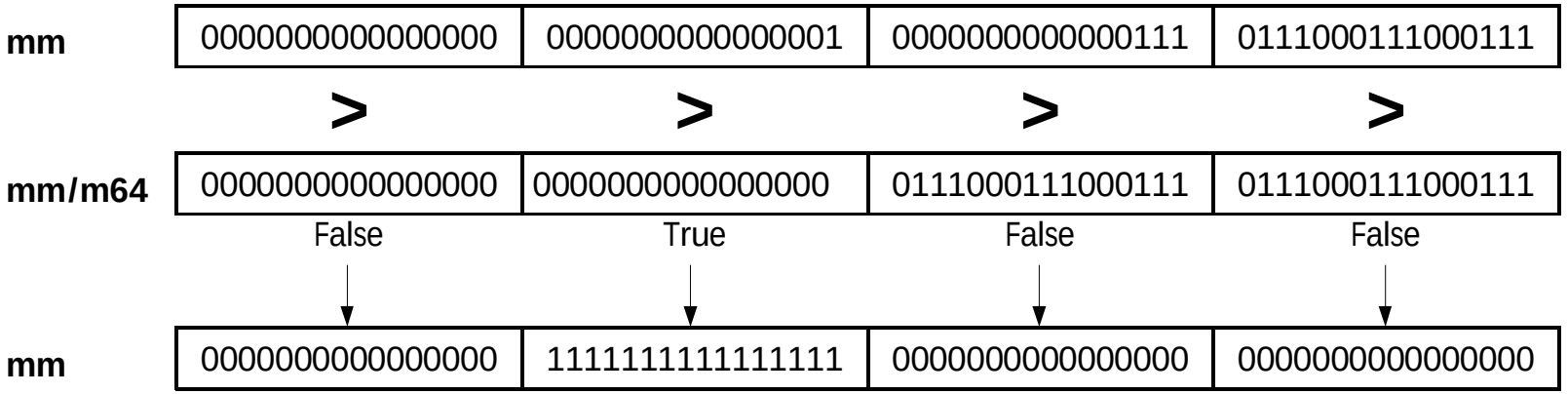
Bei PCMPEQB wird für jedes Byte 1 Vergleich der beiden Operanden durchgeführt. Wenn beide Operandenbytes gleich sind, so werden im resultierenden Register alle Bits gesetzt, sonst gelöscht.

PCMPGTB/W/D: Packet Compare for Greater−Than

Verwendung:

Syntax	Beispiel
pcmpgtb mm,mm/m64	pcmpgtb mm0,[Qvar]
pcmpgtw mm,mm/m64	pcmpgtw mm0,mm1
pcmpgtd mm,mm/m64	pcmpgtb mm0,[Qvar]

PCMPGTW mm, mm/m64



Aufgabe:

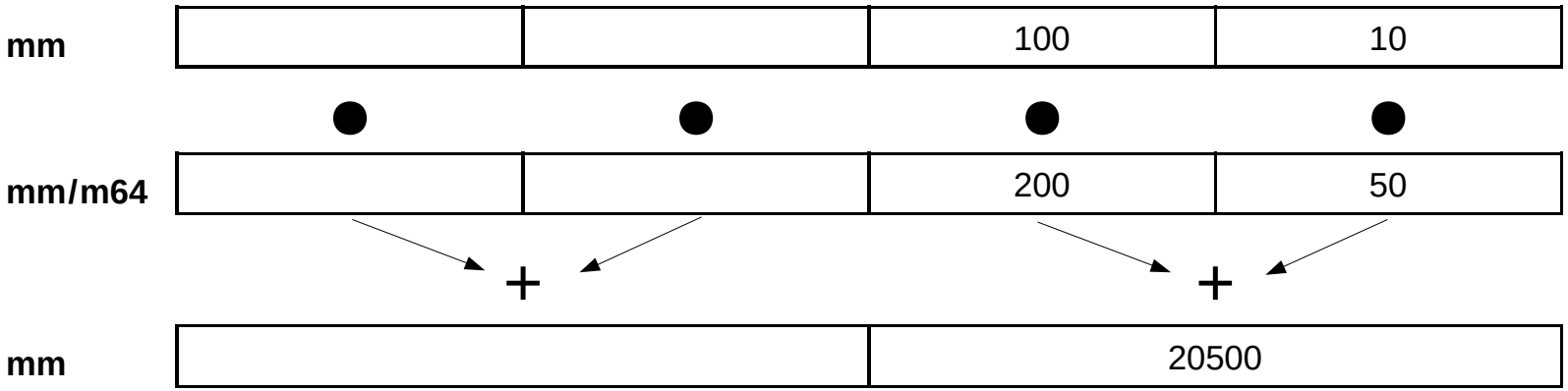
Bei PCMPGTB wird für jedes Byte 1 Vergleich der beiden Operanden durchge−führt. Wenn der Wert im 1. Operanden größer ist, als derjenige im zweiten Operand, so werden im resultierenden Register alle Bits gesetzt, sonst gelöscht.

PMADDWD: Packed Multiply and Add

Verwendung:

Syntax	Beispiel
pmaddwd mm,mm/m64	pmaddwd mm0,[Qvar]

PADDWD mm, mm/m64



Aufgabe:

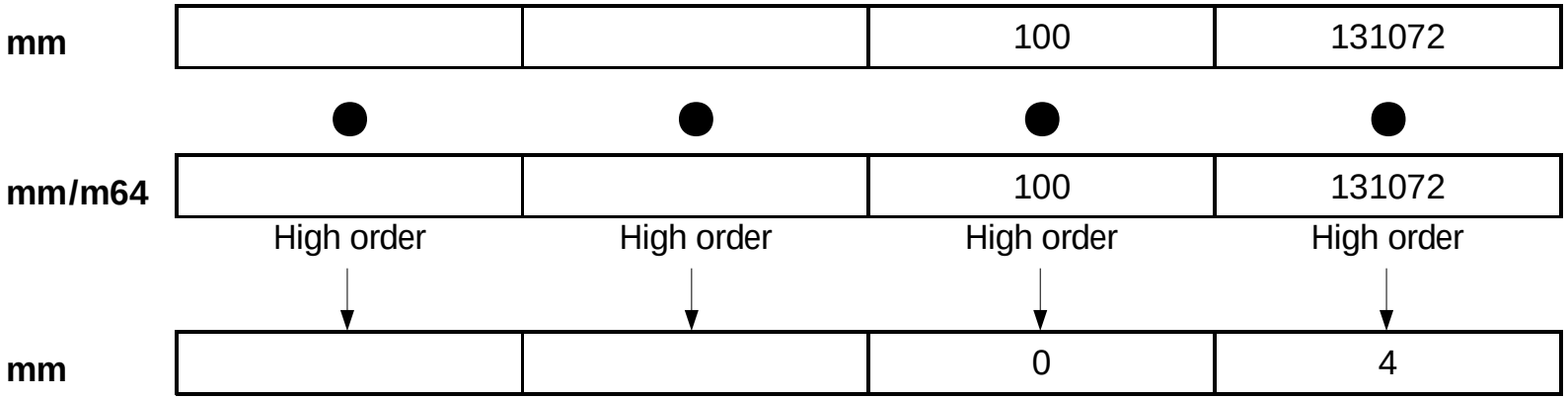
Multipliziert die vier Worte, addiert jeweils 2 hiervon und schreibt das Ergebnis als 2 Doppelworte in das Zielregister.

PMULHW: Packed Multiply High

Verwendung:

Syntax	Beispiel
pmulhw mm,mm/m64	pmulhw mm0,mm1

PMULHW mm, mm/m64



Aufgabe:

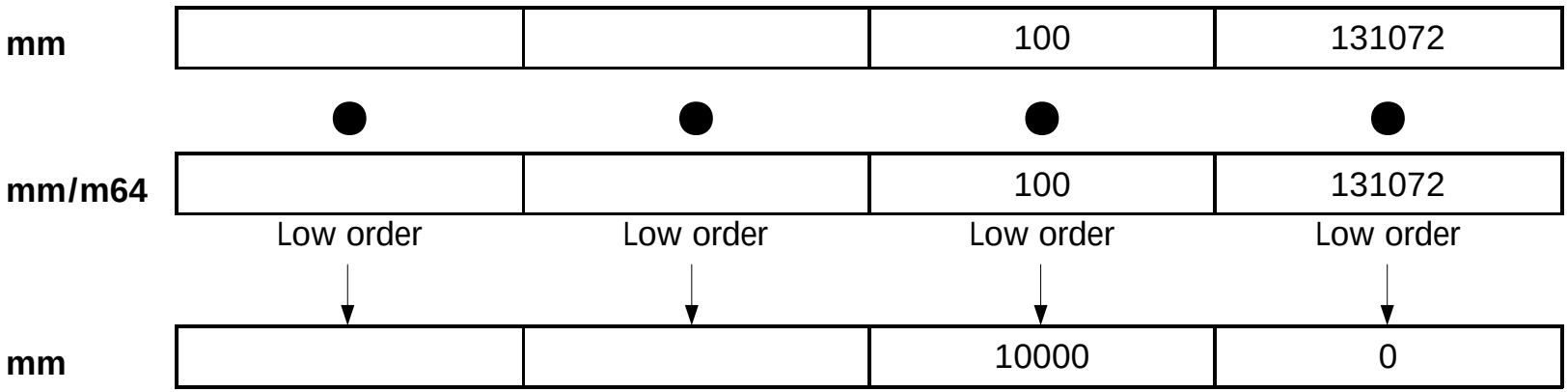
Multipliziert 4 Wörter und speichert den jeweiligen höherwertigen Anteil im Zielregister.

PMULLW: Packed Multiply Low

Verwendung:

Syntax	Beispiel
pmullw mm,mm/m64	pmullw mm0,mm1

PMULLW mm, mm/m64



Aufgabe:

Multipliziert 4 Wörter und speichert den jeweiligen niederwertigen Anteil im Zielregister.

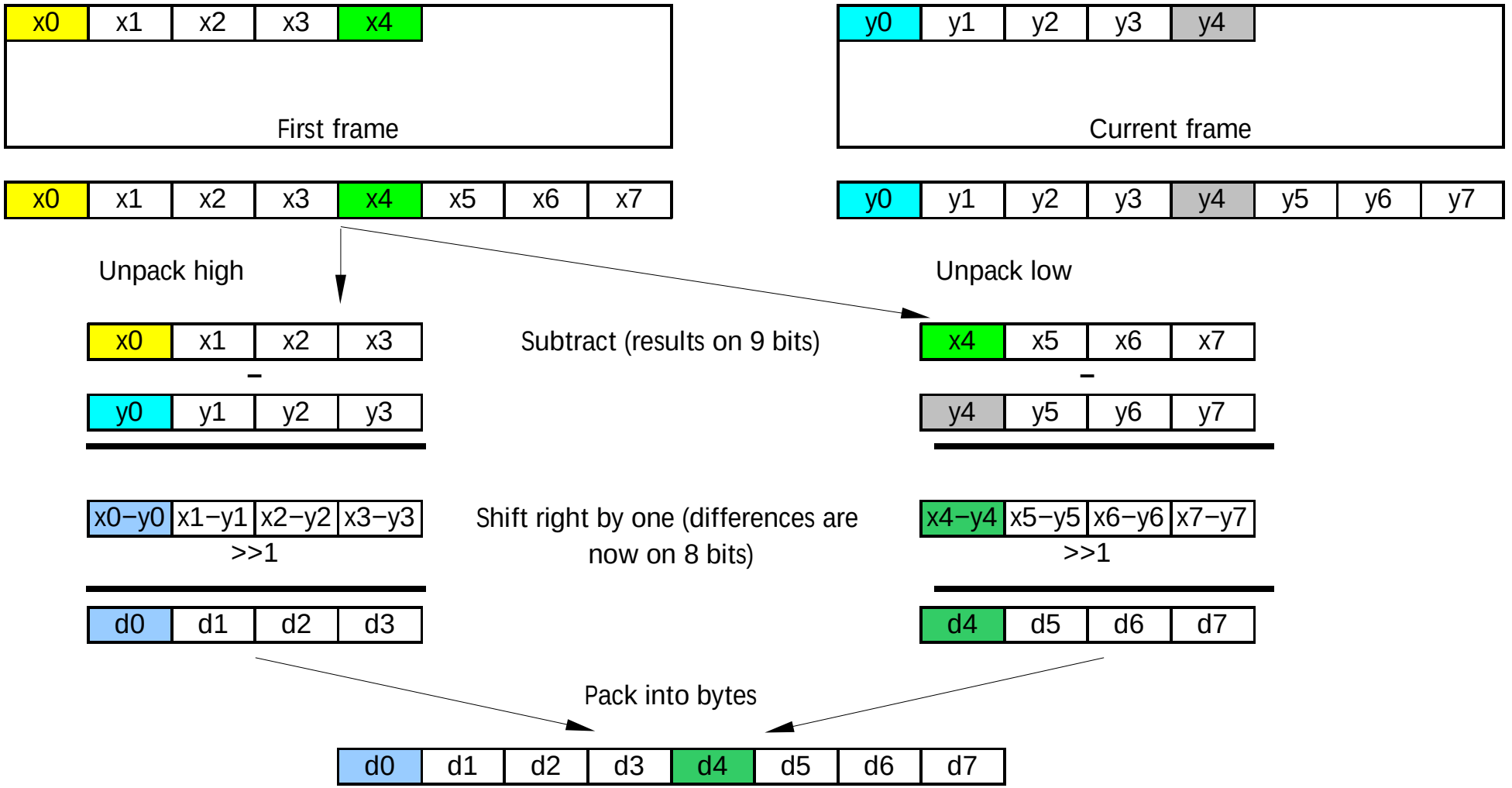
Beispiel: Addition von Speicherbereichen

ecx= Anzahl zu addierender Bytes, durch 8 dividierbar
esi=Quell−Adresse, edi= Ziel−Adresse

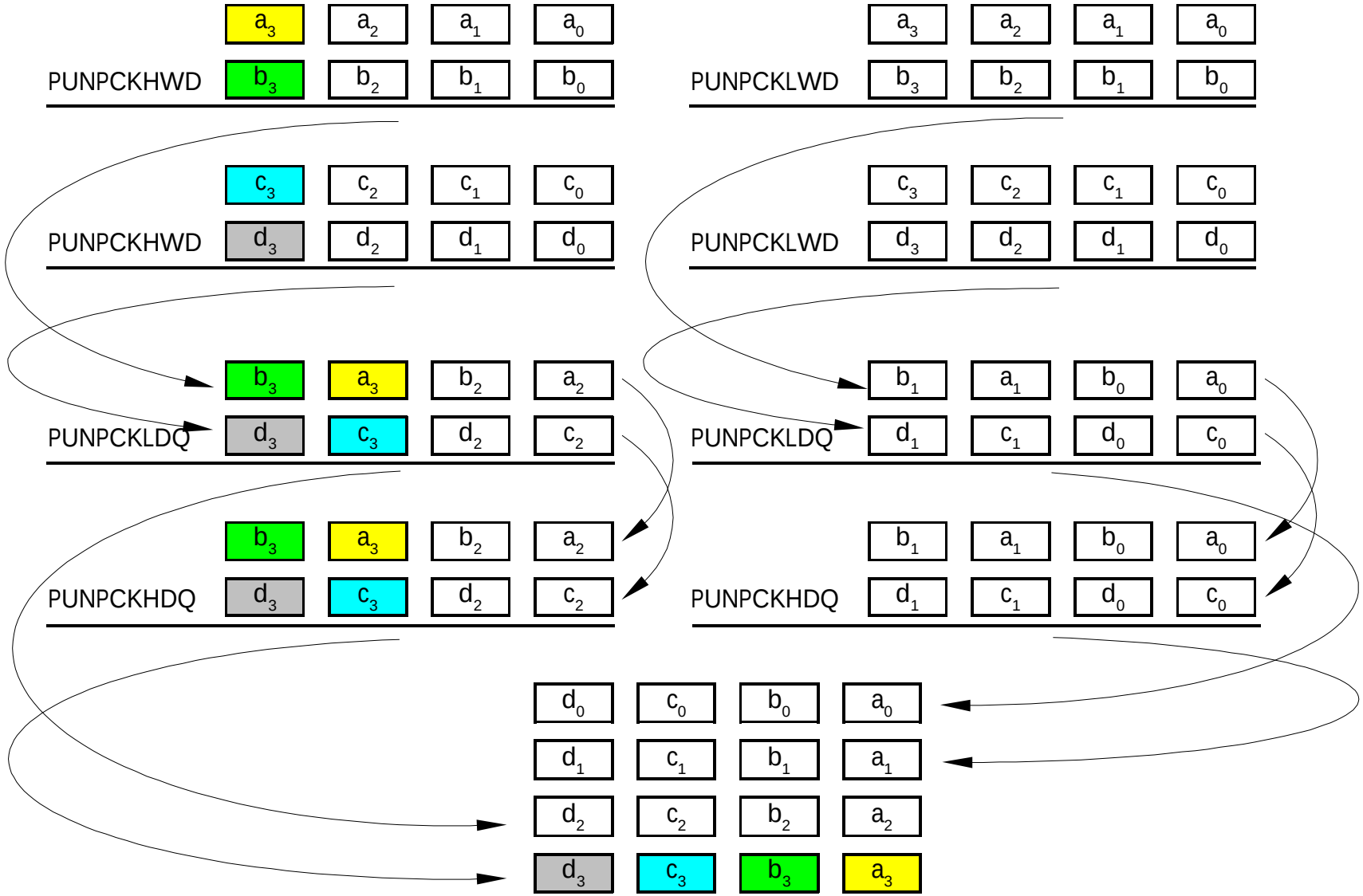
```
        shr ecx,3
        mov edx,0
.loop:  movq mm0, [esi+8*edx]
        paddb mm0, [edi+8*edx]
        movq [edi+8*edx], mm0
        inc edx
        loop .loop

        emms
```

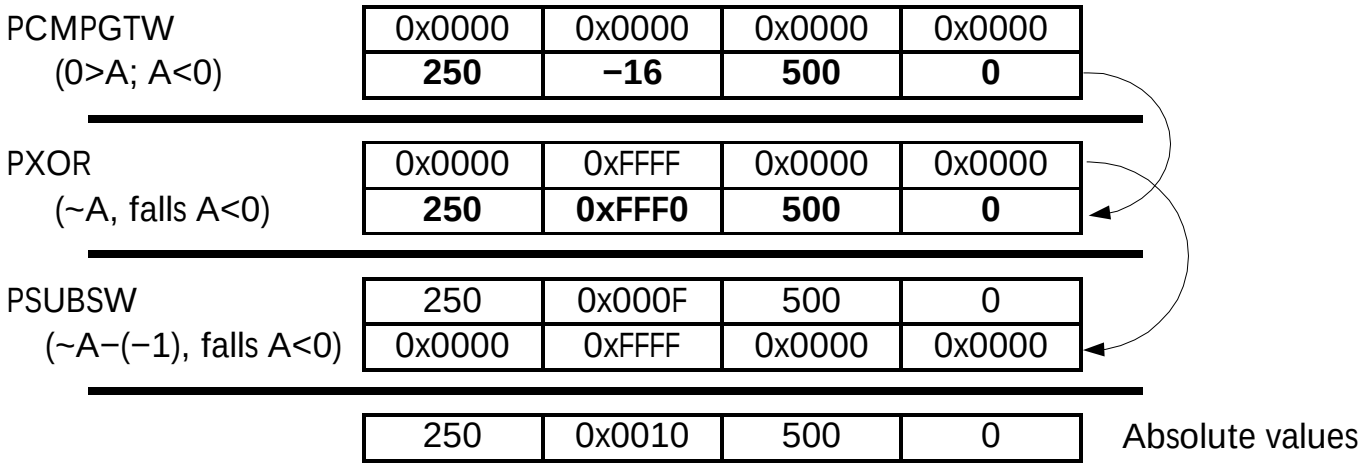
Beispiel: Subtraktion zweier Bilder



Beispiel: Transposition einer Matrix

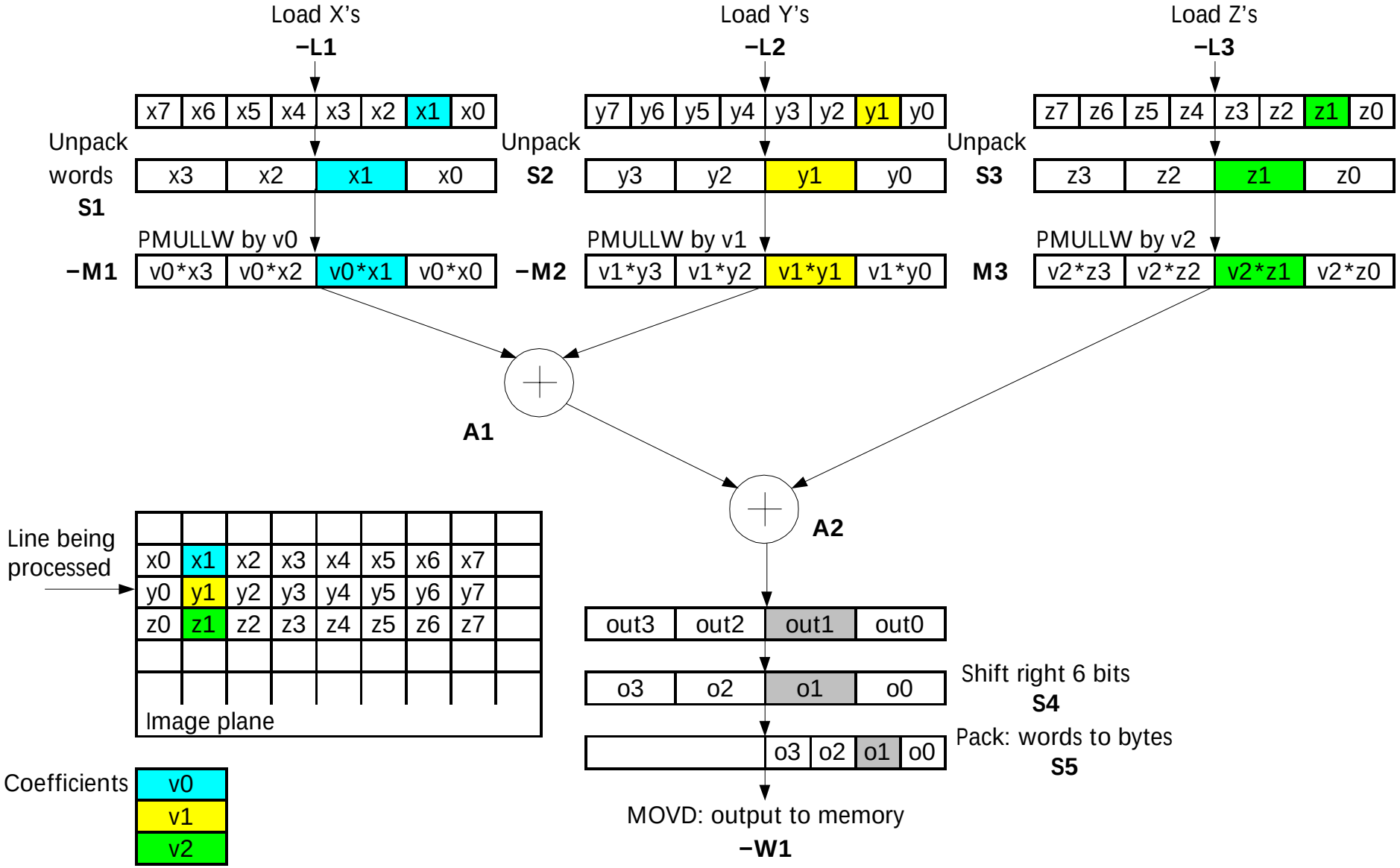


Beispiel: Absolutwertberechnung



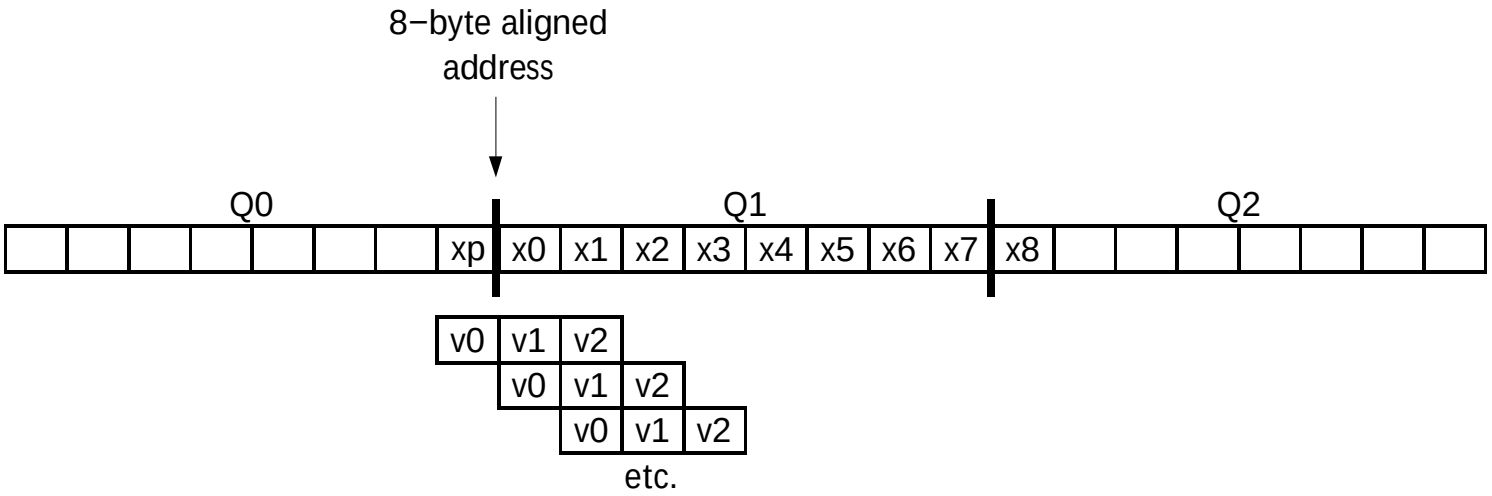
2er Komplement: −A = (~A)+1

Beispiel: FIR−Filter mit 3 Koeffizienten Y−Richtung

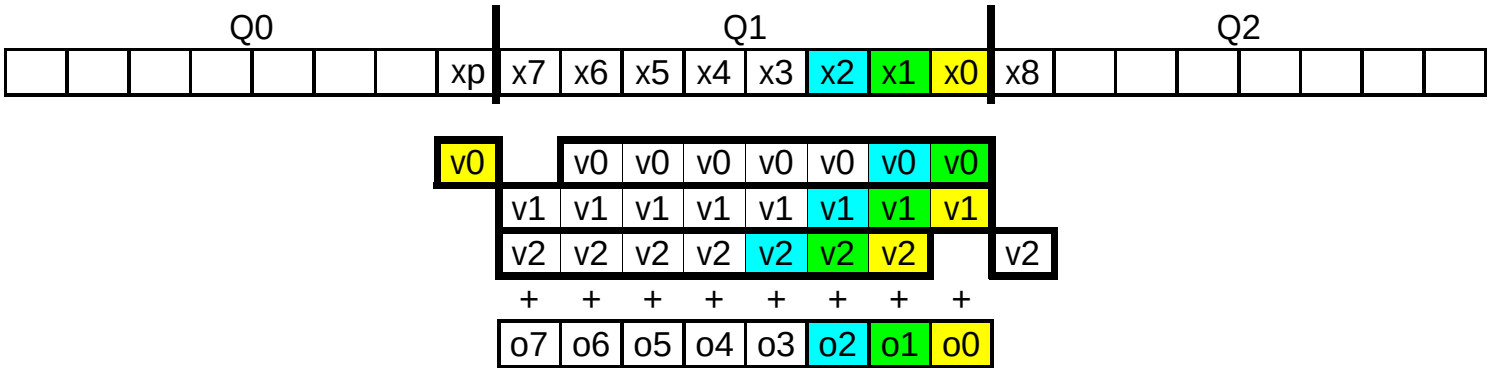


Beispiel: FIR−Filter mit 3 Koeffizienten X−Richtung

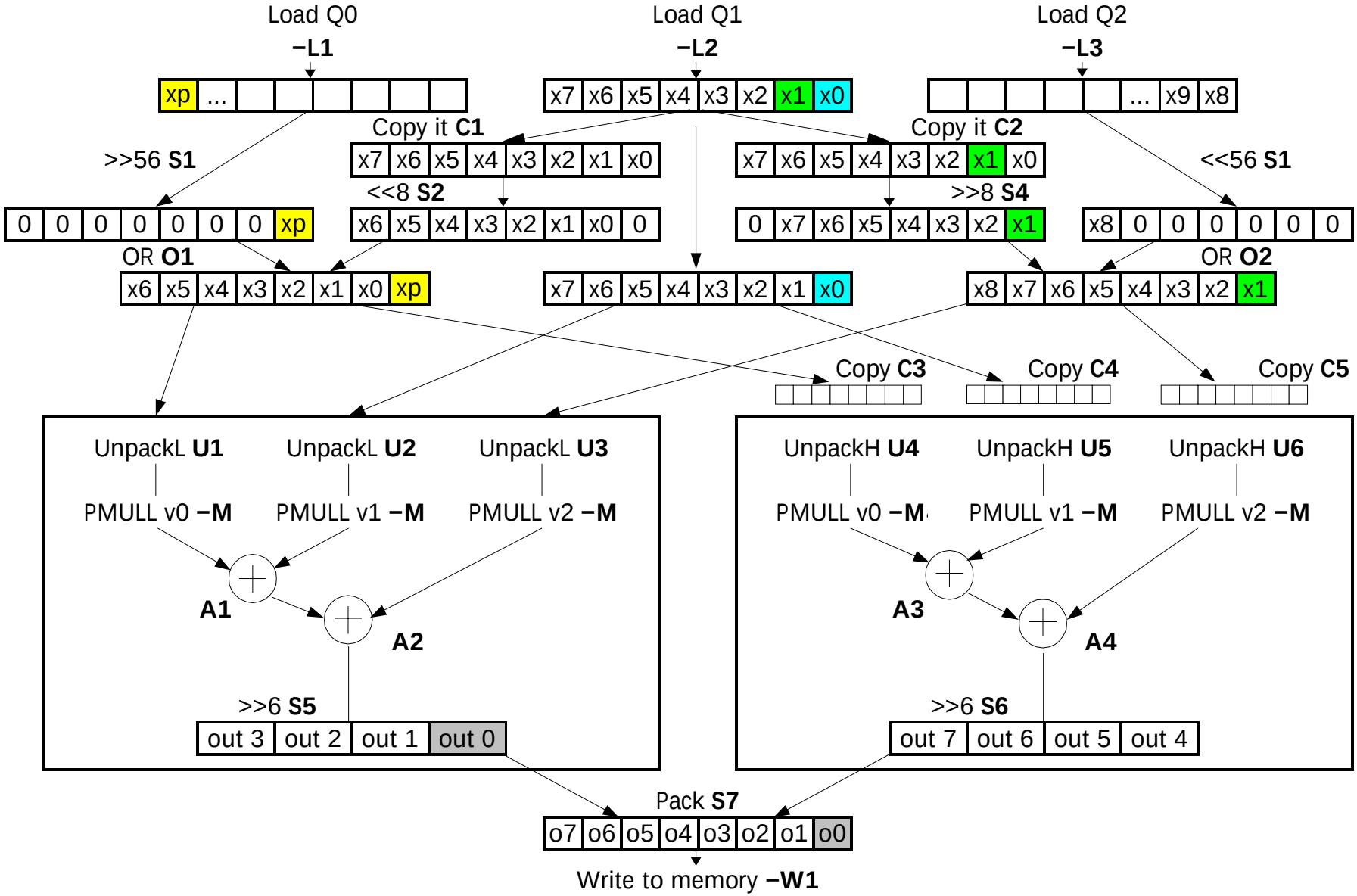
Theorie:



Praxis:



Beispiel: FIR−Filter mit 3 Koeffizienten X−Richtung



Beispiel: FIR−Filter mit 3 Koeffizienten

Optimale Allokierung nach U,V−Pipeline−Pairing:

Vertikaler FIR−Filter

Copy 1	Copy 2	Schleife
−W1	M3	
−L1	A1	
−L2 S1		
−M1	A2	
S		add edi,8
−M2	S4	
−L3	S5	
S3		add esi,8
M3	−W1	
A1	−L1	
	−L2 S1	
A2	−M1	
	S2	dec ecx
S4	−M2	
S5	−L3	
	S3	jge loop_top

Horizontaler FIR−Filter

U−Pipe	V−Pipe
−L1	(S5)
−L2	(A4)
C1	(S6)
C2	(S7)
−L3	S2
(−W1)	S1
O1	S4
C3	S3
O2	C4
U1	C5
−M1	U2
−M2	U3
−M3	U4
−M4	U5
−M5	U6
−M6	A1
A2	dec ecx
A3	jge loop_top

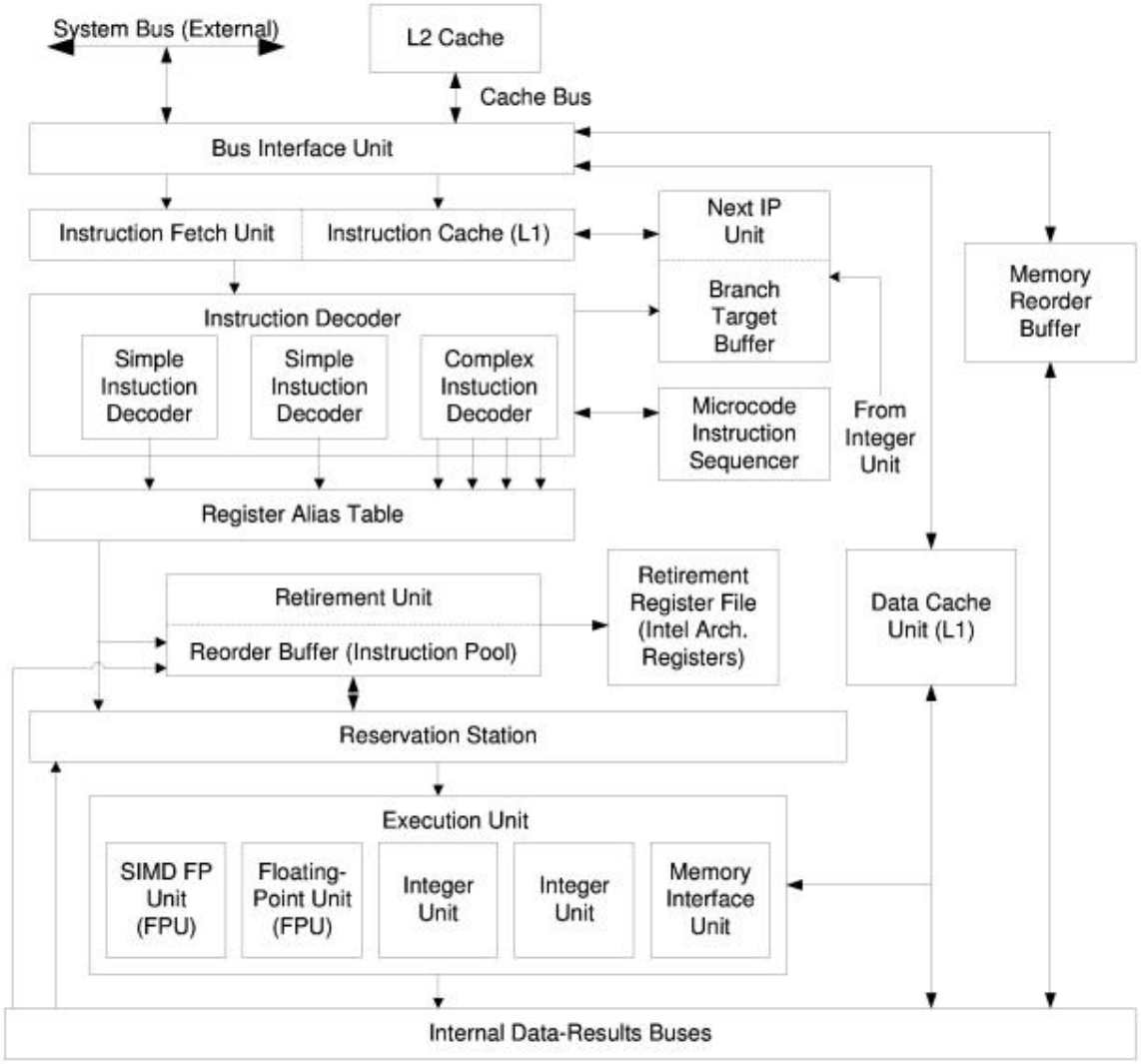
Optimierung

- Architekturüberblick

- Instruction Decoder
- Sprungoptimierung
- Pairing
- Pipelining
- Cache

Überblick P6 Architektur

Pentium Pro
Pentium II
Pentium III



Instruction Decoder

3 Instruction Decoder verfügbar

- 1 Complex Instruction Decoder: Bearbeitet maximal 4Mikro−Ops pro Takt
- 2 Simple Instruction Decoder: Bearbeiten nur Befehle, die 1Mikro−Op benötigen
- Wenn 2 komplexe Befehle hintereinander stehen, bleiben die beiden Simple Instruction Decoder leer, d.h. die Befehle benötigen je 1 Takt

Konsequenzen

- Möglichst viele 1Mikro−Op Befehle verwenden
- Falls möglich, Code so schreiben, daß die Mikro−Op Folge 4−1−1 eingehalten wird, also einen komplexen Befehl mit 2 weniger komplexen abschließen
- op reg,mem Adressierung benötigt stets 2 Mikro−Ops

Sprungoptimierung

Default−Sprungvorhersage

- Bedingte Sprünge zu vorherigen Codestellen sind wahrscheinlich
- Bedingte Sprünge zu nachfolgenden Codestellen sind unwahrscheinlich

Optimierungsmöglichkeiten

- Default−Sprungvorhersage beachten, z.B. keine Schleifen bauen, die nach vorn springen
- Sprünge vermeiden, z.B. durch SETcc
- Kleine Schleifen vermeiden, indem mehrere Daten auf einmal behandelt werden (loop unrolling)

Pairing

Zusammenstellen von 2 Befehlen zur Parallelverarbeitung (Pairing)

- Besonders wichtig auf Pentium Prozessoren, P6 Prozessoren können Out−Of−Order Execution
- Der erste Befehl wird in der U−Pipeline, der zweite in der V−Pipeline bearbeitet

U Pipeline			V Pipeline		
mov reg,reg	alu reg,imm	push reg	mov reg,reg	alu reg,imm	push reg
mov reg,mem	alu mem,imm	push imm	mov reg,mem	alu mem,imm	push imm
mov mem,reg	alu eax,imm	pop reg	mov mem,reg	alu eax,imm	pop reg
mov reg,imm	alu mem,reg	nop	mov reg,imm	alu mem,reg	nop
mov mem,imm	alu reg,mem	shift/rot by 1	mov mem,imm	alu reg,mem	jmp
mov eax,mem	inc/dec reg	shift by imm	mov eax,mem	inc/dec reg	jcc
mov mem,eax	inc/dec mem	test reg, r/m	mov mem,eax	inc/dec mem	call
alu reg,reg	lea reg,mem	test acc,imm	alu reg,reg	lea reg,mem	nop
				test reg,r/m	test acc,imm

alu: Arithmetische oder logische Anweisung (add, sub, and)
acc: Accumulator (eax oder ax)

Pairing mit MMX–Befehlen

MMX Befehle in U und V Pipeline

- Der Prozessor verfügt nur über eine Shifter– und eine Multiplizierer–Einheit; sie können in jeder Pipeline angesprochen werden.
- Zugriffe auf die gewöhnlichen Integer–Rechenregister oder auf Speicher können nur in der U–Pipeline vorgenommen werden
- Das Zielregister in der U–Pipeline darf in der V–Pipeline nicht verwendet werden
- Nicht–MMX Befehle können in U oder V–Pipe verwendet werden, wenn in dem gleichzeitig zu verarbeitenden MMX–Befehl kein Integer–Rechenregister oder Speicherzugriff erfolgt. Die Integer–Befehle müssen gemäß Tabelle gleichzeitig bearbeitbar sein
- Der MMX–Multiply Befehl hat eine Latenzzeit von 3 Taktzyklen. Vorher sollte das Ergebnis nicht verwendet werden

Pairing–Optimierung durch Loop–Unrolling für MMX

Prinzip: Bei einem Schleifendurchlauf mehrere Datenpakete berechnen

Nomenklatur für Optimierung:

- S : Zugriff auf Shifter (auch Pack, Unpack)
- M : Zugriff auf Multiplizierer
- A,..: sonstige Zugriffe
- – : Modifikator: Operation greift auf Speicher oder Integer Register zu

Regeln:

- Höchstens EINE S–Operation pro Zeile
- Höchstens EINE M–Operation pro Zeile
- Höchstens EIN Bindestrich pro Zeile
- Ergebnis der Multiplikation darf frühestens 3 Taktzyklen später benutzt werden

Beispiel zur Pairing−Optimierung

Matrix−Produkt: $\mathrm{dotprod}(X,Y)=\sum_{i=0}^{L} x_i \cdot y_i$

esi: Zeiger auf x, edi: Zeiger auf y, ecx: Zeilen/Spaltenlänge

```
dotprod:
        movq        mm0,[esi+8*ecx]     ; −L (Load with memory access)
        pmaddwd     mm0,[edi+8*ecx]     ; −M (Multiplay with mem access)
        paddd       mm1,mm0             ; A (Addition)
        dec ecx
        jge dotprod
```

Optimal gepackt:

3 Instanzen

(1)	(2)	(3)	Pipe
−L			U
	A		V
−M			U
			V
	−L		U
		A	V
	−M		U
			V
		−L	U
A			V
		−M	U
			V

Pipelining

Pipeline stalls:

- Effekt: Die Befehlspipeline wird erst komplett abgearbeitet, bevor der Befehl ausgeführt wird. (Bei P6: 7 Zyklen)
- Auftreten:
  - Bei Verwendung eines Registers: Schreibzugriff auf kleine Datenbreite, anschließend Lesen mit großer Datenbreite (mov ax,8; add ecx,eax)
    Ausnahme: XOR zum Löschen und ADD,SUB bei Rechenregistern
    (Beispiel: xor eax, eax; mov al, [mem8]; add eax,ebx)
  - Sequentielle Speicherzugriffe mit unterschiedlicher Datenbreite
    (z.B. movq mm0,[mem64]; mov eax,[mem32])

Cache

Größe:

First Level Cache: 16kByte 4fach assoziativ
Second Level Cache: 256 oder 512kByte

Alignment:

Prozessor benötigt 2 Zugriffe auf Cache, weil 32Byte Cache−Line Grenze überschritten wurde. (6−9 Zyklen)

7	6	5	4	3	2	1	0	31	30	29	28	27	26	25	24	23	22	21	20
30	71	13	27	45	65	7e	35	93	22	11	58	87	7d	9e	ef	cc	3c	ed	7f

32Bit Zugriff	OK	Misaligned	OK	OK

Prefetch−Einheit arbeitet auf 16 Bit, daher:

- Loop Labels sollten auf 16Bit Grenze ausgerichtet sein