Befehlssatz des D·CORE Prozessors

Mnemonic	Codierung			Hex	Bedeutung
					ALU-Operationen
mov	`0010 0000`	⟨*yyyy*⟩	⟨*xxxx*⟩	`20yx`	`R[x] = R[y]`
addu	`0010 0001`	⟨*yyyy*⟩	⟨*xxxx*⟩	`21yx`	`R[x] = R[x] + R[y]`
addc	`0010 0010`	⟨*yyyy*⟩	⟨*xxxx*⟩	`22yx`	`R[x] = R[x] + R[y] + C` (updates C)
subu	`0010 0011`	⟨*yyyy*⟩	⟨*xxxx*⟩	`23yx`	`R[x] = R[x] - R[y]`
and	`0010 0100`	⟨*yyyy*⟩	⟨*xxxx*⟩	`24yx`	`R[x] = R[x] AND R[y]` (bitwise)
or	`0010 0101`	⟨*yyyy*⟩	⟨*xxxx*⟩	`25yx`	`R[x] = R[x] OR  R[y]` (bitwise)
xor	`0010 0110`	⟨*yyyy*⟩	⟨*xxxx*⟩	`26yx`	`R[x] = R[x] XOR R[y]` (bitwise)
not	`0010 0111`	⟨****⟩	⟨*xxxx*⟩	`27*x`	`R[x] = NOT R[x]` (bitwise)
					Shift-Operationen
lsl	`0010 1000`	⟨*yyyy*⟩	⟨*xxxx*⟩	`28yx`	`R[x] = R[x] <<  R[y].<3:0>`
lsr	`0010 1001`	⟨*yyyy*⟩	⟨*xxxx*⟩	`29yx`	`R[x] = R[x] >>> R[y].<3:0>`
asr	`0010 1010`	⟨*yyyy*⟩	⟨*xxxx*⟩	`2Ayx`	`R[x] = R[x] >>  R[y].<3:0>`
lslc	`0010 1100`	⟨****⟩	⟨*xxxx*⟩	`2C*x`	`R[x] = R[x] <<  1, C=R[X].15`
lsrc	`0010 1101`	⟨****⟩	⟨*xxxx*⟩	`2D*x`	`R[x] = R[x] >>> 1, C=R[X].0`
asrc	`0010 1110`	⟨****⟩	⟨*xxxx*⟩	`2E*x`	`R[x] = R[x] >>  1, C=R[X].0`
					Vergleichs-Operationen
cmpe	`0011 0000`	⟨*yyyy*⟩	⟨*xxxx*⟩	`30yx`	`C = (R[x] == R[y])`
cmpne	`0011 0001`	⟨*yyyy*⟩	⟨*xxxx*⟩	`31yx`	`C = (R[x] != R[y])`
cmpgt	`0011 0010`	⟨*yyyy*⟩	⟨*xxxx*⟩	`32yx`	`C = (R[x] >  R[y])` (signed)
cmplt	`0011 0011`	⟨*yyyy*⟩	⟨*xxxx*⟩	`33yx`	`C = (R[x] <  R[y])` (signed)
					Immediate-Operationen
movi	`0011 0100`	⟨*cccc*⟩	⟨*xxxx*⟩	`34cx`	`R[x] = 0x000`⟨*c*⟩
addi	`0011 0101`	⟨*cccc*⟩	⟨*xxxx*⟩	`35cx`	`R[x] = R[x] + 0x000`⟨*c*⟩
subi	`0011 0110`	⟨*cccc*⟩	⟨*xxxx*⟩	`36cx`	`R[x] = R[x] - 0x000`⟨*c*⟩
andi	`0011 0111`	⟨*cccc*⟩	⟨*xxxx*⟩	`37cx`	`R[x] = R[x] AND 0x000`⟨*c*⟩
lsli	`0011 1000`	⟨*cccc*⟩	⟨*xxxx*⟩	`38cx`	`R[x] = R[x] <<  `⟨*c*⟩
lsri	`0011 1001`	⟨*cccc*⟩	⟨*xxxx*⟩	`39cx`	`R[x] = R[x] >>> `⟨*c*⟩
bseti	`0011 1010`	⟨*cccc*⟩	⟨*xxxx*⟩	`3Acx`	`R[x] = R[x]	(1 << `⟨*c*⟩`)` (set bit)
bclri	`0011 1011`	⟨*cccc*⟩	⟨*xxxx*⟩	`3Bcx`	`R[x] = R[x] & !(1 << `⟨*c*⟩`)` (clear bit)
					Speicher-Operationen
ldw	`0100` ⟨*cccc*⟩	⟨*yyyy*⟩	⟨*xxxx*⟩	`4cyx`	`R[x] = MEM( R[y] + (0x000`⟨*c*⟩`<<1) )`
stw	`0101` ⟨*cccc*⟩	⟨*yyyy*⟩	⟨*xxxx*⟩	`5cyx`	`MEM( R[y] + (0x000`⟨*c*⟩`<<1) ) = R[x]`
					Kontrollfluss / Sprung-Operationen
br	`1000` ⟨*iiii*⟩	⟨*iiii*⟩	⟨*iiii*⟩	`8iii`	`PC = PC+2+`⟨*imm12*⟩
jsr	`1001` ⟨*iiii*⟩	⟨*iiii*⟩	⟨*iiii*⟩	`9iii`	`R[15] = PC+2; PC = PC+2+`⟨*imm12*⟩ (call)
bt	`1010` ⟨*iiii*⟩	⟨*iiii*⟩	⟨*iiii*⟩	`Aiii`	`(C=1) ? PC = PC+2+`⟨*imm12*⟩` : PC=PC+2`
bf	`1011` ⟨*iiii*⟩	⟨*iiii*⟩	⟨*iiii*⟩	`Biii`	`(C=0) ? PC = PC+2+`⟨*imm12*⟩` : PC=PC+2`
jmp	`1100` ⟨****⟩	⟨****⟩	⟨*xxxx*⟩	`C**x`	`PC = R[x]`
halt	`1111` ⟨****⟩	⟨****⟩	⟨****⟩	`F***`	`halt`
andere Opcodes illegal					

⟨*xxxx*⟩, x: 4-bit Index des Quell- und Zielregisters RX   ⟨*xxxx*⟩ – binär, x – hex
⟨*yyyy*⟩, y: 4-bit Index des Quellregisters RY   ⟨*yyyy*⟩ – binär, y – hex
⟨*cccc*⟩, c: 4-bit Konstante IMM4   ⟨*cccc*⟩ – binär, c – hex
     iii: 12-bit sign-extended Konstante IMM12   ⟨*iiii*⟩ – binär, i – hex
⟨****⟩, *: don't care