DATX11-23-34

RiscFlisp

Konstruktion av enkel 16-bitars ”enchipsdator” i VHDL.

Bakgrund

RISC V är en relativt ny processor men en instruktionsuppsättning som är ”open source”. EU har beslutat sig för att satsa på att det skall finnas kompetens inom Europa när det gäller högpresterande processorer och Chalmers deltar i det arbetet.
I grundkurserna lär vi idag ut hur en äldre typ av processorer är uppbyggda och fungerar.
Den enkla processor som vi använder för närvarande heter FLISP.

Projektbeskrivning

Arbetet utgör en förstudie av en enkel 16-bitars RISC -dator avsedd för utbildningar i Grundläggande datorteknik. Specifikationen har utformats för att ge en stor grad av frihet i utformandet av lösningen. Tanken är att implementera RiscFlisp i en FPGA och sätta upp en komplett utvecklingsmiljö.

Litteratur

https://en.wikipedia.org/wiki/RISC-V

Målgrupp

D, DV och E, förutsätter kunskap i VHDL eller att EDA322 Digital konstruktion läses parallellt.

Förslagslämnare

Roger Johansson och Arne Linde

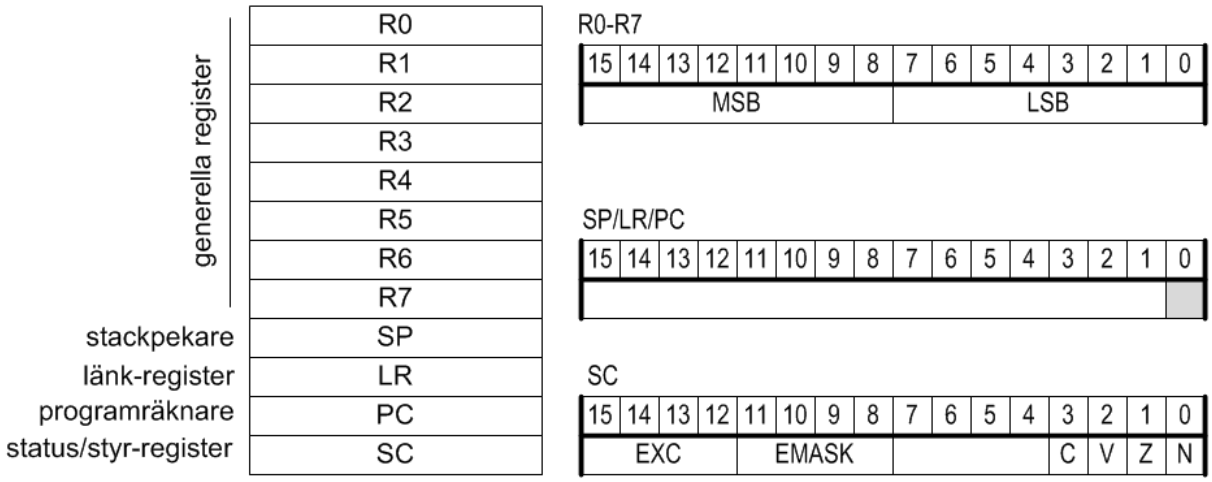
SPECIFIKATION AV PROCESSORN

Processor
16-bitars ”Load/Store”- arkitektur med 8 generella register och 4 speciella register

Adressrum
Max 64 kByte minne (byte-adresserat)

Periferikretsar
- Timerkrets
- GPIO
- UART

REGISTERUPPSÄTTNING



SC, förklaringar

N	
Z	
V	
C	
EMASK (Execution Mask)	
0000	Normal
0001	Fault
	Interrupt
1111	Single step instruction
EXC (Exception level)	Statusbitarna indikerar processorns exekveringsnivå (endast läsbara bitar)

INSTRUKTIONSGRUPPER OCH INSTRUKTIONSFORMAT

”Load/Store”

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Ins
0	1	0	0	0	Rd			const								LDIH
0	1	0	0	1	Rd			const								LDIL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Ins
0	1	0	1	0	Rd			Ri			offset					LDRBU
0	1	0	1	1	Rd			Ri			offset					LDRBS
0	1	1	0	0	Rd			Ri			offset >> 1					LDR
0	1	1	0	1	Rs			Ri			offset					STRB
0	1	1	1	0	Rs			Ri			offset >> 1					STR

”Register transfers”

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Ins
0	1	1	1	1	Rd			Rs					0	0	0	MOV
0	1	1	1	1	Rd			Rs					0	0	1	EXG

“Unary and Integer test”

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Ins
0	0	0	0	0	0	1							0	0	0	SXT
0	0	0	0	0	0	1							0	0	1	REV
0	0	0	0	0	0	1	Rd			Rs			0	1	0	NEG
0	0	0	0	0	0	1							0	1	1	NOT
0	0	0	0	0	0	1							4	0	0	CMP
0	0	0	0	0	2	0	Rs			const						CMPI

”Stack operations”

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Ins
		0	0	1	0	SC	LR	register_list								PSH
		0	0	1	1	SC	PC	register_list								PUL
																ADDI SP
																SUBI SP

”Misc.”

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Ins
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	NOP
0	0	0	1	1	1	1	1	const								TRAP

”Integer arithmetics expression evaluation”

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Ins
		1	0	0	0	1	Rd			Rs1			Rs2			ADD
		1			1	0										SUB
		1			1	1										AND
		1		1												OR
		1		1		1										EOR
		1		1	1											ASR
		1		1	1	1										LSL
		1	1													LSR
		1	1			1										ROL
		1	1		1	0										ROR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Ins
		0	1	0	1	0	Rsd			const						ADDI
		0	1	0	1	1				const						SUBI
		0	1	1						0	0	const				ASRI
		0	1	1		1				0	0	const				LSLI
		0	1	1		1				0	1	const				LSRI
		0	1	1		1				1	0	const				ROLI
		0	1	1		1				1	1	const				RORI

”Program (Flow) control”

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Ins
		0	1	0	0	1	Rd									BR
		0	1	0	0	0	Rd									BLR
1			0	0	0	1	offset >> 1									BEQ
1			0	0	1	0	offset >> 1									BNE
1			0	0	1	1	offset >> 1									BMI
1			0	1	0	0	offset >> 1									BPL
1			0	1	0	1	offset >> 1									BVS
1			0	1	1	0	offset >> 1									BVC
1			0	1	1	1	offset >> 1									BHI
1			1	0	0	0	offset >> 1									BHS
1			1	0	0	1	offset >> 1									BLO
1			1	0	1	0	offset >> 1									BLS
1			1	0	1	1	offset >> 1									BGT
1			1	1	0	0	offset >> 1									BGE
1			1	1	0	1	offset >> 1									BLT
1			1	1	1	0	offset >> 1									BLE
1			0	0	0	0	offset >> 1									BRA
1			1	1	1	1	offset >> 1									BL

TILLDELNINGAR

LDIL – Load immediate low

Placerar de 8 minst signifikanta bitarna av en 16-bitars konstant i destinations-registret. De 8 mest signikfikanta bitarna i registret nollställs.

Syntax:

`LDIH`	`Rd,PC,const`
`Rd`	Destination, (R0-R7).
`const`	Konstant, 16 bitar.

LDIH – Load immediate high

Syntax:

`LDIH`	`Rd,PC,const`
`Rd`	Destination, (R0-R7).
`const`	Konstant, 16 bitar.

LDR – Load register

En basadress bestäms genom att en konstant offset adderas till innehållet i ett basregister. Därefter kopieras ett 16-bitars ord, från denna adress, till destinationsregistret. Om konstanten är 0 ska den utelämnas.

Syntax:

`LDR`	`Rd,[Rb,imm5]`
`LDR`	`Rd,[Rb]`
`Rd`	Destination, (R0-R7).
`Rb`	Basregister för adressberäkningen, (R0-R7).
`imm5`	Konstant 2-64. Endast multiplar av 2 är tillåtna, kodas med 5 bitar.

LDRBU – Load register byte unsigned

En basadress bestäms genom att en konstant offset adderas till innehållet i ett basregister. Därefter kopieras ett 8-bitars ord, från denna adress, till destinationsregistret. De mest signifikanta 8 bitarna i destinationsregistret nollställs Om konstanten är 0 ska den utelämnas.

`LDRBU`	`Rd,[Rb,imm5]`
`LDRBU`	`Rd,[Rb]`
`Rd`	Destination, (R0-R7).
`Rb`	Basregister för adressberäkningen, (R0-R7).
`imm5`	Konstant 1-31.

LDRBS – Load register byte signed

En basadress bestäms genom att en konstant offset adderas till innehållet i ett basregister. Därefter kopieras ett 8-bitars ord, från denna adress, till destinationsregistret och teckenutvidgning till 16 bitar sker. Om konstanten är 0 ska den utelämnas.

`LDRBS`	`Rd,[Rb,imm5]`
`LDRBS`	`Rd,[Rb]`
`Rd`	Destination, (R0-R7).
`Rb`	Basregister för adressberäkningen, (R0-R7).
`imm5`	Konstant 1-31.

STR – Store register

En basadress bestäms genom att en konstant offset adderas till innehållet i ett basregister. Därefter kopieras ett 16-bitars ord, från källregistret till denna adress. Om konstanten är 0 ska den utelämnas.

Syntax:

`STR`	`[Rb,imm5],Rs`
`Rs`	Källa, (R0-R7).
`Rb`	Basregister för adressberäkningen, (R0-R7).
`imm5`	Konstant 0-62. Endast multiplar av 2 är tillåtna, kodas med 5 bitar.

STRB – Store register byte

En basadress bestäms genom att en konstant offset adderas till innehållet i ett basregister. Därefter kopieras de minst signifikanta 8-bitarna från källregistret till denna adress. Om konstanten är 0 ska den utelämnas.

`STRB`	`[Rb,imm5],Rs`
`Rd`	Destination, (R0-R7).
`Rb`	Basregister för adressberäkningen, (R0-R7).
`imm5`	Konstant 0-31.

MOV – Move register

Kopierar innehållet i källregistret till destinationsregistret. I den första formen kan alla register användas, då sker ingen flaggpåverkan. I den andra formen kan endast låga register användas, med denna form uppdateras flaggorna.

Syntax:

`MOV`	`Rd,Rs`
`Rs`	Källregister (R0-R7, SP, LR, PC,CC).
`Rd`	Destinationsregister (R0-R7, SP, LR, PC,CC).

Flaggor;

Påverkas endast då CC är destinationsregister

Z	Ettställs om resultatet blev 0, nollställs annars
C,V	Påverkas ej

SXT – Sign extend register

Teckenutvidgar tal med tecken, från 8-bitar i källregistret till 16 bitar och placerar resultatet i destinationsregistret.

Syntax:

`SXT`	`Rd,Rs`
`Rs`	Källregister (R0-R7).
`Rd`	Destinationsregister (R0-R7).

REV – Byte-reverse

Skifta endianordning genom att arrangera om bytes i källregistret till ett 16-bitars ord som placeras i destinationsregistret.

Syntax:

`REV`	`Rd,Rs`
`Rm`	Källregister (R0-R7).
`Rd`	Destinationsregister (R0-R7).

Detaljer:

Rd <15:8> = Rs<7:0>;
Rd <7:0> = Rs<15:8>;

UTTRYCKSEVALUERING

Aritmetik- och logik- operationer

ADD – Add register

Innehållen i två källregister adderas. Resultatet placeras i destinationsregistret. Flaggor uppdateras.

Syntax:

ADD	Rd,Rs1,Rs2
Rs1	Källregister 1 (R0-R7).
Rs2	Källregister 2 (R0-R7).
Rd	Destinationsregister (R0-R7).

Flaggor;

N	kopia av bit 16 hos resultatet
Z	Ettställs om resultatet blev 0, nollställs annars
C	Carry från additionen
V	Tvåkomplementspill

ADDI – Add immediate

En konstant adderas till innehållet i registret.

Syntax:

ADDI	Rsd,<imm6>
Rsd	Register (R0-R7).
imm6	Positiv konstant 0-63.

Flaggor;

N	kopia av bit 16 hos resultatet
Z	Ettställs om resultatet blev 0, nollställs annars
C	Carry från additionen
V	Tvåkomplementspill

SUB – Subtract register

Innehållet i register 2 subtraheras från innehållet i register 1. Resultatet placeras i destinationsregistret.

Syntax:

SUB	Rd,Rs1,Rs2
Rs1	Källregister 1 (R0-R7).
Rs2	Källregister 2 (R0-R7).
Rd	Destinationsregister (R0-R7).

Flaggor;

N	kopia av bit 16 hos resultatet
Z	Ettställs om resultatet blev 0, nollställs annars
C	Borrow från subtraktionen
V	Tvåkomplementspill

SUB – Subtract immediate

En konstant subtraheras från innehållet i registret.

Syntax:

SUBI	Rsd,<imm6>
Rsd	Register (R0-R7).
imm6	Positiv konstant 0-63.

Flaggor;

N	kopia av bit 16 hos resultatet
Z	Ettställs om resultatet blev 0, nollställs annars
C	Borrow från subtraktionen
V	Tvåkomplementspill

NEG – Negate register

Innehållet i källregistret subtraheras från 0. Resultatet placeras i destinationsregistret.

Syntax:

NEG	Rd,Rs
Rs	Källregister.
Rd	Destinationsregister.

Flaggor;

N	kopia av bit 16 hos resultatet
Z	Ettställs om resultatet blev 0, nollställs annars
C	Borrow från subtraktionen
V	1 om Rs=0, 0 annars

Bitoperationer

AND – Bitwise AND register

Instruktionen utför bitvis OCH mellan innehållen i källregistren. Resultatet placeras i destinationsregistret.

Syntax:

AND	Rd,Rs1,Rs2
Rs1	Källregister 1 (R0-R7).
Rs2	Källregister 2 (R0-R7).
Rd	Destinationsregister (R0-R7).

Flaggor;

N	kopia av bit 16 hos resultatet
Z	Ettställs om resultatet blev 0, nollställs annars
C,V	Påverkas ej

OR – Bitwise OR register

Instruktionen utför bitvis ELLER mellan innehållen i källregistren. Resultatet placeras i destinationsregistret.

Syntax:

OR	Rd,Rs1,Rs2
Rs1	Källregister 1 (R0-R7).
Rs2	Källregister 2 (R0-R7).
Rd	Destinationsregister (R0-R7).

Flaggor;

N	kopia av bit 16 hos resultatet
Z	Ettställs om resultatet blev 0, nollställs annars
C,V	Påverkas ej

EOR – Bitwise Exclusive OR

Instruktionen utför bitvis Exklusivt ELLER mellan innehållen i källregistren. Resultatet placeras i destinationsregistret.

Syntax:

EOR	Rd,Rs1,Rs2
Rs1	Källregister 1 (R0-R7).
Rs2	Källregister 2 (R0-R7).
Rd	Destinationsregister (R0-R7).

Flaggor;

N	kopia av bit 16 hos resultatet
Z	Ettställs om resultatet blev 0, nollställs annars
C,V	Påverkas ej

NOT – Bitwise NOT register

Instruktionen bildar komplementet av innehållet i källregistret och placerar detta i destinationsregistret.

Syntax:

NOT	Rd,Rs
Rs	Källregister.
Rd	Destinationsregister.

Flaggor;

N	kopia av bit 16 hos resultatet
Z	Ettställs om resultatet blev 0, nollställs annars
C,V	Påverkas ej

Jämförelse

`CMP` – Compare registers

Operander subtraheras (Rs1-Rs2), endast flaggregistret påverkas av operationen.

Syntax:

`CMP`	`Rs1,Rs2`
`Rs1`	Register (R0-R7).
`Rs2`	Register (R0-R7).

Flaggor;

N	kopia av bit 16 hos resultatet
Z	Ettställs om resultatet blev 0, nollställs annars
C	Borrow från subtraktionen
V	Tvåkomplementspill

`CMPI` – Compare immediate

Operander subtraheras (Rs-imm6), endast flaggregistret påverkas av operationen.

Syntax:

`CMPI`	`Rs,imm6`
`Rs`	Register (R0-R7).
`imm8`	Konstant -32..31.

Flaggor;

N	kopia av bit 31 hos resultatet
Z	Ettställs om resultatet blev 0, nollställs annars
C	Carry från additionen
V	Tvåkomplementspill

Skiftoperationer

`LSL` – Logical shift left register

Logiskt vänsterskift, innehållet i ett källregister skiftas och placeras i destinationsregistret.

Syntax:

`LSL`	`Rd,Rs1,Rs2`
`Rd`	Destinationsregister (R0-R7).
`Rs1`	Källregister 1 (R0-R7), skiftoperand.
`Rs2`	Källregister (R0-R7), de fyra minst signifikanta bitarns anger antal skift som ska utföras.

Flaggor;

N	kopia av bit 16 hos resultatet
Z	Ettställs om resultatet blev 0, nollställs annars
C	Carry från sista skift
V	Påverkas ej

`LSL` – Logical shift left immediate

Logiskt vänsterskift, innehållet i ett källregister skiftas och placeras i destinationsregistret.

Syntax:

`LSLI`	`Rd,Rs,#<imm4>`
`Rd`	Destinationsregister (R0-R7).
`Rs`	Källregister (R0-R7).
`imm4`	Konstanten anger antal skift som ska utföras.

Flaggor;

N	kopia av bit 31 hos resultatet
Z	Ettställs om resultatet blev 0, nollställs annars
C	Carry från sista skift
V	Påverkas ej

`LSR` – Logical shift right register

Logiskt högerskift, innehållet i ett källregister skiftas och placeras i destinationsregistret.

Syntax:

`LSR`	`Rd,Rs1,Rs2`
`Rd`	Destinationsregister (R0-R7).
`Rs1`	Källregister 1 (R0-R7), skiftoperand.
`Rs2`	Källregister (R0-R7), de fyra minst signifikanta bitarns anger antal skift som ska utföras.

Flaggor;

N	kopia av bit 16 hos resultatet
Z	Ettställs om resultatet blev 0, nollställs annars
C	Carry från sista skift
V	Påverkas ej

`LSR` – Logical shift right immediate

Logiskt högerskift, innehållet i källregistret skiftas och placeras i destinationsregistret. En konstant anger antal skift som ska utföras.

Syntax:

`LSRI`	`Rd,Rs,#<imm4>`
`Rd`	Destinationsregister (R0-R7).
`Rs`	Källregister (R0-R7).
`imm4`	Konstanten anger antal skift som ska utföras.

Flaggor;

N	kopia av bit 31 hos resultatet
Z	Ettställs om resultatet blev 0, nollställs annars
C	Carry från sista skift
V	Påverkas ej

`ASR` – Arithmetic shift right register

Aritmetiskt högerskift, innehållet i ett källregister skiftas och placeras i destinationsregistret.

Syntax:

`LSR`	`Rd,Rs1,Rs2`
`Rd`	Destinationsregister (R0-R7).
`Rs1`	Källregister 1 (R0-R7), skiftoperand.
`Rs2`	Källregister (R0-R7), de fyra minst signifikanta bitarns anger antal skift som ska utföras.

Flaggor;

N	kopia av bit 16 hos resultatet
Z	Ettställs om resultatet blev 0, nollställs annars
C	Carry från sista skift
V	Påverkas ej

`ASR` – Arithmetic shift right immediate

Logiskt högerskift, innehållet i källregistret skiftas och placeras i destinationsregistret. En konstant anger antal skift som ska utföras.

Syntax:

`ASRI`	`Rd,Rs,#<imm4>`
`Rd`	Destinationsregister (R0-R7).
`Rs`	Källregister (R0-R7).
`imm4`	Konstanten anger antal skift som ska utföras.

Flaggor;

N	kopia av bit 31 hos resultatet
Z	Ettställs om resultatet blev 0, nollställs annars
C	Carry från sista skift
V	Påverkas ej

`ROR` – Rotate right register

Rotation höger, innehållet i ett källregister skiftas och placeras i destinationsregistret. Carryflaggan ingår i skiftet, inskiftad bit hämtas från Carry och en utskiftad bit hamnar i nästa Carry.

Syntax:

`ROR` `Rd,Rs1,Rs2`

`Rd` Destinationsregister (R0-R7).

`Rs1` Källregister 1 (R0-R7), skiftoperand.

`Rs2` Källregister (R0-R7), de fyra minst signifikanta bitarns anger antal skift som ska utföras.

Flaggor;

N kopia av bit 16 hos resultatet

Z Ettställs om resultatet blev 0, nollställs annars

C Carry från sista skift

V Påverkas ej

`RORI` – Rotate right immediate

Rotation höger, innehållet i ett källregister skiftas och placeras i destinationsregistret. Carryflaggan ingår i skiftet, inskiftad bit hämtas från Carry och en utskiftad bit hamnar i nästa Carry.

Syntax:

`ASRI` `Rd,Rs,#<imm4>`

`Rd` Destinationsregister (R0-R7).

`Rs` Källregister (R0-R7).

`imm4` Konstanten anger antal skift som ska utföras.

Flaggor;

N kopia av bit 16 hos resultatet

Z Ettställs om resultatet blev 0, nollställs annars

C Carry från sista skift

V Påverkas ej

`ROL` – Rotate left register

Rotation vänster, innehållet i ett källregister skiftas och placeras i destinationsregistret. Carryflaggan ingår i skiftet, inskiftad bit hämtas från Carry och en utskiftad bit hamnar i nästa Carry.

Syntax:

`ROL` `Rd,Rs1,Rs2`

`Rd` Destinationsregister (R0-R7).

`Rs1` Källregister 1 (R0-R7), skiftoperand.

`Rs2` Källregister (R0-R7), de fyra minst signifikanta bitarns anger antal skift som ska utföras.

Flaggor;

N kopia av bit 16 hos resultatet

Z Ettställs om resultatet blev 0, nollställs annars

C Carry från sista skift

V Påverkas ej

`ROLI` – Rotate left immediate

Rotation vänster, innehållet i ett källregister skiftas och placeras i destinationsregistret. Carryflaggan ingår i skiftet, inskiftad bit hämtas från Carry och en utskiftad bit hamnar i nästa Carry.

Syntax:

`LSRI` `Rd,Rs,#<imm4>`

`Rd` Destinationsregister (R0-R7).

`Rs` Källregister (R0-R7).

`imm4` Konstanten anger antal skift som ska utföras.

Flaggor;

N kopia av bit 16 hos resultatet

Z Ettställs om resultatet blev 0, nollställs annars

C Carry från sista skift

V Påverkas ej

PROGRAMFLÖDESKONTROLL

`BRA` – Unconditional branch

Ovillkorlig programflödesändring, destinationsadressen placeras i PC.

Syntax:

`B` `label`

`label` Destination.

`B<condition>` – Conditional branch

Villkorlig programflödesändring, villkor dikterat av flaggorna N,V,Z,C och kombinationer av dessa evalueras. Om villkoret är uppfyllt placeras.destinationsadressen i PC, annars fortsätter exekveringen med nästa sekventiella instruktion..

Syntax:

`Bcond` `label`

`label` Destination.

Mnemonic	Funktion	Villkor
Enkla flaggtest		
`BCS/BHS`	"Hopp" om *carry*	C=1
`BCC/BLO`	"Hopp" om ICKE *carry*	C=0
`BEQ`	"Hopp" om *zero*	Z=1
`BNE`	"Hopp" om ICKE *zero*	Z=0
`BMI`	"Hopp" om *negative*	N=1
`BPL`	"Hopp" om ICKE *negative*	N=0
`BVS`	"Hopp" om *overflow*	V=1
`BVC`	"Hopp" om ICKE *overflow*	V=0
Jämförelse av tal utan tecken		
`BHI`	Villkor: Rn>Rm	C ● ! Z = 1
`BHS/BCS`	Villkor: Rn ≥ Rm	C=1
`BLO/BCC`	Villkor: Rn < Rm	C=0
`BLS`	Villkor: Rn ≤ Rm	! C + Z = 1
Jämförelse av tal med tecken		
`BGT`	Villkor: Rn > Rm	Z + (N ⊕ V) = 0
`BGE`	Villkor: Rn ≥ Rm	N ⊕ V = 0
`BLT`	Villkor: Rn < Rm	N ⊕ V = 1
`BLE`	Villkor: Rn ≤ Rm	Z + (N ⊕ V) = 1

Samma tabell kan uttryckas med C-operatorer och får då följande utseende:

C-operator	Betydelse	Datatyp	Instruktion
==	Lika	`signed/unsigned`	`BEQ`
!=	Skild från	`signed/unsigned`	`BNE`
<	Mindre än	`signed`	`BLT`
		`unsigned`	`BCC`
<=	Mindre än eller lika	`signed`	`BLE`
		`unsigned`	`BLS`
>	Större än	`signed`	`BGT`
		`unsigned`	`BHI`
>=	Större än eller lika	`signed`	`BGE`
		`unsigned`	`BCS`

`BL` – Branch with link

Subrutinanrop, adressen till nästa sekventiellt placerade instruktion placeras i LR, destinationsadressen placeras i PC,

Syntax:

`BL` `label`

`label` Destination.

`BR` – Branch register

Destinationsadressen, som finns i ett register, placeras i PC.

Syntax:

`BX` `Rs`

`Rs` Källregister (R0-R7) innehåller destinationsadressen.

Anm:

Om destinationsadressen är udda utlöses undantaget INVALIDPC.

`BLR` – Branch with link and exchange

Subrutinanrop, adressen till nästa sekventiellt placerade instruktion placeras i LR. Destinationsadressen, som finns i ett register, placeras i PC,

Syntax:

`BLX` `Rm`

`Rm` Källregister (R0-R7) innehåller destinationsadressen och den minst signifikanta biten anger hur instruktion på adressen ska tolkas, Thumb eller ARM..

SPECIELLA INSTRUKTIONER

`PSH`

`PUL`

`TRAP` – Trap

”Software trap”, instruktionen orsakar undantagshantering.De 8 minst signifikanta bitarna kan användas för att koda detaljerad systemspecifik information.

Syntax:

`TRAP` `#imm8`

`imm8` En konstant (0..255), ignoreras av processorn.

`NOP` – No operation

Instruktionen utför ingenting.

Syntax:

`NOP`