ADK
ADKL

Add constant

ADK
ADKL

P851M
P852M
P856M
P857M

Syntax:

[label] ␣ ADK ␣ r3, k — T8
[label] ␣ ADKL ␣ r1, lk — T2

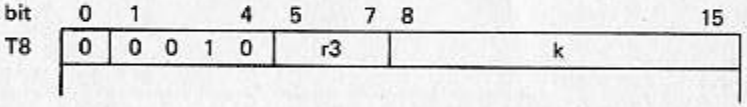
T8 The positive constant k is added to the contents of the register specified in r3. The result of the addition is placed in r3.

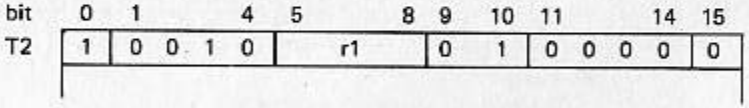
T2 The positive or negative constant lk is added to the contents of the register specified in r1. The result of the addition is placed in r1.

Type	*Function*	*Syntax*
T8	(r3) + k → r3	ADK r3, k
T2	(r1) + lk → r1	ADKL r1, lk

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow





Remark:
Restricted to system mode if r1 = A15.

ADR
ADRS

Addition register/register

ADR
ADRS

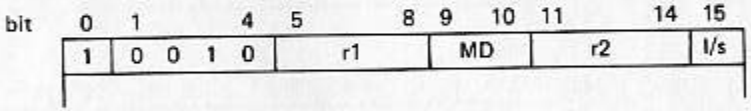
P851M
P852M
P856M
P857M

Syntax: [label] ␣ ADR [*] ␣ r1, r2
[label] ␣ ADRS ␣ r1, r2

The contents of the register specified by r1 are added either to the contents of the register specified by r2 (direct addressing), in which case the sum is always placed in the register specified by r1, or to the contents of the memory address indicated in the register specified by r2 (indirect addressing). In that case the sum is placed either in the register specified by r1 (the l/s indicator being 0) or in the memory address (l/s = 1).

Type	*Function*	*MD*	*l/s*	*Syntax*
T1	(r1) + (r2) → r1	00	n.s.	ADR r1, r2
T3	(r1) + ((r2)) → r1	01	0	ADR* r1, r2
T3	(r1) + ((r2)) → (r2)	01	1	ADRS r1, r2

Condition register:
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow



Remarks:
* When l/s = 1 (store), r1 must be ≠ 0.
* Restricted to system mode if r1 = A15.

AD
ADS

Addition

AD
ADS

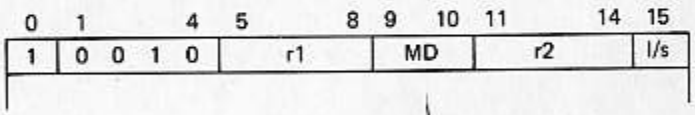
P851M
P852M
P856M
P857M

Syntax: [label] ␣ AD [S] [*] ␣ r1, m[, r2]

The contents of the effective memory address are added to the contents of the register specified by r1.
The sum is placed either in the register specified by r1, in which case the load/store must be 0, or in the effective memory address when the load/store indicator is 1.

Type	*Function*	*MD*	*l/s*	*Syntax*
T4	(r1) + (m) → r1	10	0	AD r1, m
T4	(r1) + (m) → m	10	1	ADS r1, m
T5	(r1) + (m + (r2)) → r1	10	0	AD r1, m, r2
T5	(r1) + (m + (r2)) → m + (r2)	10	1	ADS r1, m, r2
T6	(r1) + ((m)) → r1	11	0	AD* r1, m
T6	(r1) + ((m)) → (m)	11	1	ADS* r1, m
T7	(r1) + ((m + (r2))) → r1	11	0	AD* r1, m, r2
T7	(r1) + ((m + (r2))) → (m + (r2))	11	1	ADS* r1, m, r2

Condition register:
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow



Remarks:
* When l/s = 1, r1 must be ≠ 0.
* Restricted to system mode if r1 = A15.

IMR	*Increment memory/register*	IMR	P851M P852M P856M P857M

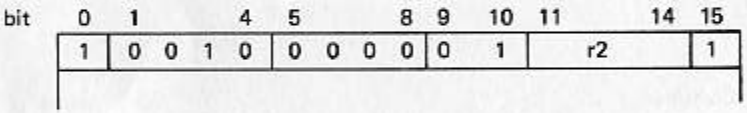
Syntax: [label] ␣ IMR ␣ r2

The contents of the effective memory address indicated in the register specified by r2 (indirect) are increased by one.

Type	*Function*	*Syntax*
T3	((r2)) + 1 → (r2)	IMR r2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow



IM	*Increment memory*	IM	P851M P852M P856M P857M

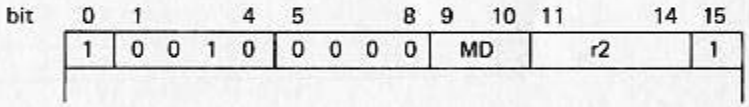
Syntax: [label] ␣ IM [*] ␣ m [, r2]

This instruction increases by 1 the contents of the effective memory address, after which the value of the effective memory address is replaced by the new value.

Type	*Function*	*MD*	*Syntax*
T4	(m) + 1 → m	10	IM m
T5	(m + (r2)) + 1 → m + (r2)	10	IM m, r2
T6	((m)) + 1 → (m)	11	IM* m
T7	((m + (r2))) + 1 → (m + (r2))	11	IM* m, r2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow



SUK SUKL	*Subtract constant*	SUK SUKL	P851M P852M P856M P857M

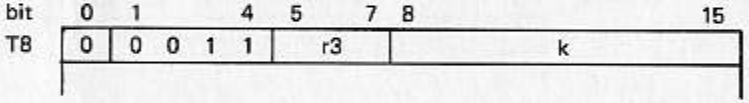
Syntax:
[label] ␣ SUK␣ r3, k — T8
[label] ␣ SUKL ␣ r1, lk — T2

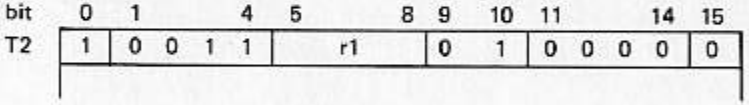
T8 The positive constant k is subtracted from the contents of the register specified in r3. The result is placed in r3.

T2 The positive or negative constant lk is subtracted from the contents of the register specified in r1. The result is placed in r1.

Type	*Function*	*Syntax*
T8	(r3) − k → r3	SUK r3, k
T2	(r1) − lk → r1	SUKL r1, lk

Condition register:
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow





Remark:
Restricted to system mode if r1 = A15.

SUR SURS	*Subtract register/register*	SUR SURS	P851M P852M P856M P857M

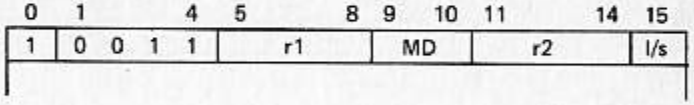
Syntax:
[label] ␣ SUR [*] ␣ r1, r2
[label] ␣ SURS ␣ r1, r2

The contents of the register specified by r2 (direct addressing) or the contents of the memory address indicated in the register specified by r2 (indirect addressing) are subtracted from the contents of the 16-bit register specified by r1. The result of this operation is placed:

− (direct addressing) : in the register specified by r1
− (indirect addressing) : either in the register specified by r1 (l/s = 0) in the memory address indicated in the register specified by r2 (l/s = 1).

Type	*Function*	*MD*	*l/s*	*Syntax*
T1	(r1) − (r2) → r1	00	0	SUR r1, r2
T3	(r1) − ((r2)) → r1	01	0	SUR* r1, r2
T3	(r1) − ((r2)) → (r2)	01	1	SURS r1, r2

Condition register:
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow



Remark:
* When l/s = 1, r1 must be ≠ 0
* Restricted to system mode if r1 = A15.

SU
SUS

Subtract word

SU
SUS

P851M
P852M
P856M
P857M

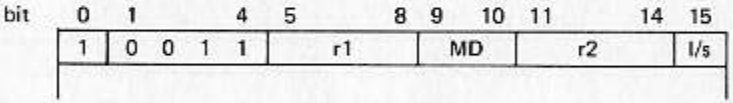
Syntax: [label]␣ SU[S] [*]␣ r1, m[, r2]

The contents of the effective memory address are subtracted from the contents of the register specified by r1. The result is placed in the register specified by r1, when the l/s bit is 0, or in the effective memory address when l/s is 1.

Type	*Function*	*MD*	*l/s*	*Syntax*	
T4	(r1) – (m) → r1	10	0	SU	r1, m
T4	(r1) – (m) → m	10	1	SUS	r1, m
T5	(r1) – (m + (r2)) → r1	10	0	SU	r1, m, r2
T5	(r1) – (m + (r2)) → m + (r2)	10	1	SUS	r1, m, r2
T6	(r1) – ((m)) → r1	11	0	SU*	r1, m
T6	(r1) – ((m)) → (m)	11	1	SUS*	r1, m
T7	(r1) – ((m + (r2))) → r1	11	0	SU*	r1, m, r2
T7	(r1) – ((m + (r2))) → (m + (r2))	11	1	SUS*	r1, m, r2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow



Remark:
* When the l/s bit = 1, r1 must be ≠ 0
* Restricted to system mode if r1 = A15.

CWK

Compare word with constant

CWK

P851M
P852M
P856M
P857M

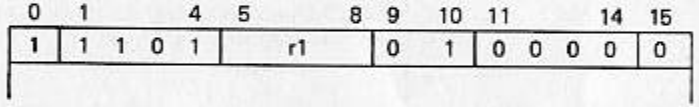
Syntax: [label] ␣ CWK ␣ r1, lk

The contents of the register specified by r1 are compared with the constant. The result of this comparison is stored in the condition register.

Type	*Function*	*Syntax*
T2	(r1) ↔ lk → CR	CWK r1, lk

Condition register:

CR = 0 if (r1) = lk
1 if (r1) > lk
2 if (r1) < lk



Remark:
Restricted to system mode if r1 = A15.

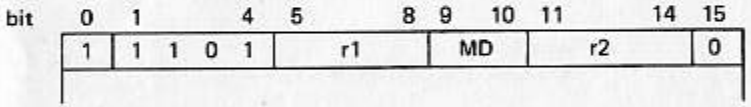
CWR *Compare words register/register* **CWR** P851M P852M P856M P857M

Syntax: [label] ␣ CWR [*] ␣ r1, r2

The contents of the 16-bit register specified by r1 are compared with the contents of the 16-bit register specified by r2 (direct addressing) or with the contents of the memory address held in the register specified by r2 (indirect addressing).
The result of the comparison is stored in the condition register.

Type	*Function*	*MD*	*I/s*	*Syntax*
T1	(r1) ↔ (r2) → CR	00	0	CWR r1, r2
T3	(r1) ↔ ((r2)) → CR	01	0	CWR* r1, r2

Condition register:
CR = 0 if (r1) = (2nd operand)
1 if (r1) > (2nd operand)
2 if (r1) < (2nd operand)



Remark:
Restricted to system mode if r1 = A15.

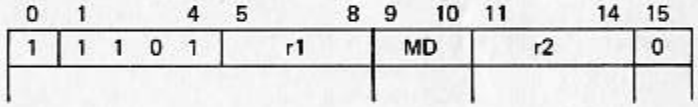
CW *Compare words* **CW** P851M P852M P856M P857M

Syntax: [label] ␣ CW [*] ␣ r1, m [, r2]

The contents of the 16-bit register specified by r1 are compared with the contents of the effective memory address which is found in the word following the instruction.
The result of this comparison is stored in the condition register.

Type	*Function*	*MD*	*Syntax*
T4	(r1) ↔ (m) → CR	10	CW r1, m
T5	(r1) ↔ (m + (r2)) → CR	10	CW r1, m, r2
T6	(r1) ↔ ((m)) → CR	11	CW* r1, m
T7	(r1) ↔ ((m + (r2))) → CR	11	CW* r1, m, r2

Condition register:
CR = 0 if (r1) = 2nd operand
1 if (r1) > 2nd operand
2 if (r1) < 2nd operand



Remark:
Restricted to system mode if r1 = A15.

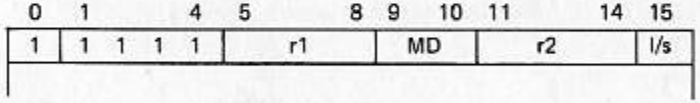
C1 C1S	*Ones complement*	C1 C1S	P851M P852M P856M P857M

Syntax: [label] ␣ C1 [*]␣ r1, m [, r2]
[label] ␣ C1S [*]␣ m [, r2]

Logic Complement: One bits in the specified word or register become 0 and vice versa. The logic complement of the effective memory address replaces either the contents of the 16-bit register specified by r1 or the contents of the effective memory address, depending on the state of the I/s indicator.

Type	*Function*		*MD*	*I/s*	*Syntax*	
T4	$\overline{(m)}$	→ r1	10	0	C1	r1, m
T4	$\overline{(m)}$	→ m	10	1	C1S	m
T5	$\overline{(m + (r2))}$	→ r1	10	0	C1	r1, m, r2
T5	$\overline{(m + (r2))}$	→ m + (r2)	10	1	C1S	m, r2
T6	$\overline{((m))}$	→ r1	11	0	C1*	r1, m
T6	$\overline{((m))}$	→ (m)	11	1	C1S*	m
T7	$\overline{((m + (r2)))}$	→ r1	11	0	C1*	r1, m, r2
T7	$\overline{((m + (r2)))}$	→ (m + (r2))	11	1	C1S*	m, r2

Condition register: CR = 0 if result = 0
1 if result > 0
2 if result < 0



Remark:
* When I/s = 0, r1 must be = 0
* Restricted to system mode when r1 = A15.

C1R C1RS	*Ones complement register/register*	C1R C1RS	P851M P852M P856M P857M

Syntax: [label]␣ C1R [*]␣ r1, r2
[label]␣ C1RS ␣ r2

Logic complement: Bits which contained 1 in the specified register become 0, and vice versa.

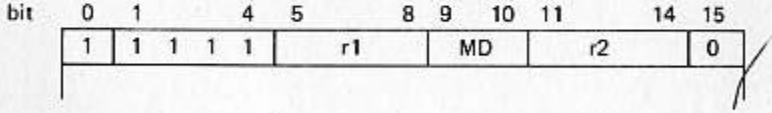
The logic complement of the contents of the 16-bit register specified by r2 (direct addressing) or the contents of the memory address indicated in the register specified by r2 replaces the contents of:

– (direct addressing) : the register specified by r1
– (indirect addressing) : either the register specified by r1 (I/s = 0) or the memory address indicated in the register specified by r2 (I/s = 1).

If r1 is not specified, the default value will be P.

Type	*Function*	*MD*	*I/s*	*Syntax*	
T1	$\overline{(r2)}$ → r1	00	0	C1R	r1, r2
T3	$\overline{((r2))}$ → r1	01	0	C1R*	r1, r2
T3	$\overline{((r2))}$ → (r2)	01	1	C1RS	r2

Condition register: CR = 0 if result = 0
1 if result > 0
2 if result < 0



Remark:
* When I/s = 0, r1 must be = 0
* Restricted to system mode when r1 = A15.

NGR *Negate register* **NGR** P851M P852M P856M P857M

Syntax: [label]␣ NGR ␣ r1, r2

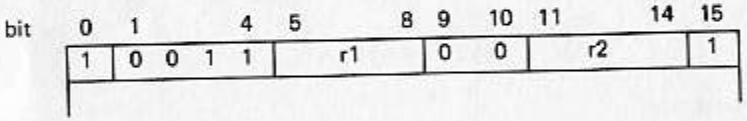
Twos complement.
Zero bits become one and vice versa, +1.

The twos complement of the contents of the register specified by r2 replaces the contents of the register specified by r1.

Type	*Function*	*Syntax*
T1	0 – (r2) → r1	NGR r1, r2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow



Remark:
* r1 must be ≠ 0
* Restricted to system mode when r1 = A15 (not for P851M).

C2R *Twos complement/register* **C2R** P851M P852M P856M P857M

Syntax: [label]␣ C2R ␣ r2

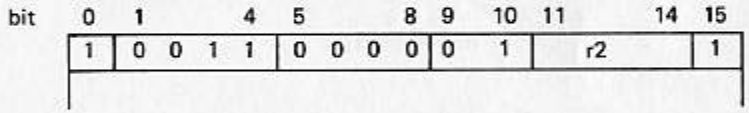
Twos complement.
Zero bits become one and vice versa, +1.

The twos complement (or negative) of the contents of the effective memory address replaces the old contents of this address.

Type	*Function*	*Syntax*
T3	0 – ((r2)) → (r2)	C2R r2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow



C2 — *Twos complement* — **C2**

P851M
P852M
P856M
P857M

Syntax: [label] ␣ C2 [*] ␣ m [, r2]

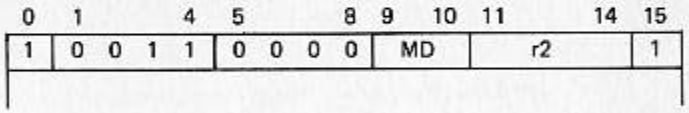
Twos complement.
Zero bits become one and vice versa, +1.

The twos complement (or negative) of the contents of the effective memory address, indicated by the word following the instruction, replaces the old contents.

Type	*Function*	*MD*	*Syntax*
T4	0 – (m) → m	10	C2 m
T5	0 – (m + (r2)) → m + (r2)	10	C2 m, r2
T6	0 – ((m)) → (m)	11	C2* m
T7	0 – ((m + (r2))) → (m + (r2))	11	C2* m, r2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow



CMR — *Clear memory/register* — **CMR**

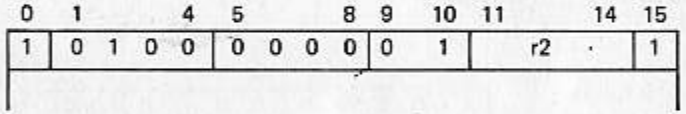
P851M
P852M
P856M
P857M

Syntax: [label] ␣ CMR ␣ r2

The contents of the memory address specified in the register specified by r2 are reset to 0.

Type	*Function*	*Syntax*
T3	0 → (r2)	CMR r2

Condition register: Unchanged



CM

Clear memory

CM

P851M
P852M
P856M
P857M

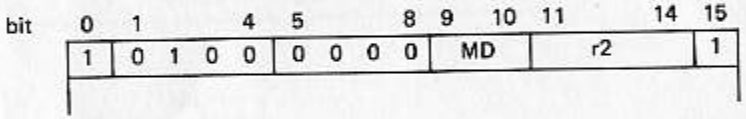
Syntax: [label]␣ CM [*] ␣ m [, r2]

The contents of the effective memory address are reset to 0.

Type	*Function*
T4	0 → m
T5	0 → m + (r2)
T6	0 → (m)
T7	0 → (m + (r2))

MD	*Syntax*
10	CM m
10	CM m, r2
11	CM* m
11	CM* m, r2

Condition register: Unchanged



MUK

Multiply with constant

MUK

P851M
P852M
P856M
P857M

(softw. sim)

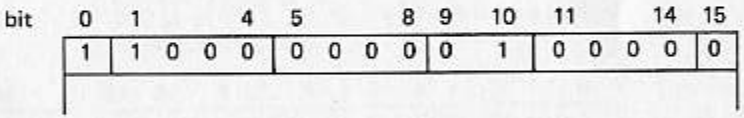
Syntax: [label]␣ MUK ␣ lk

The constant lk is multiplied by the constant of register A2. The result of the multiplication is loaded as a 31-bit product in registers A1 and A2. Bit 0 of A2 is reset to zero. The sign bit of A1 is the sign of the result. Overflow occurs if the result $> 2^{30}-1$.
In that case the two registers contain only the 30 least significant bits while the sign bit may or may not be correct.

Type	*Function*
T2	(A2) x lk → A1, A2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow



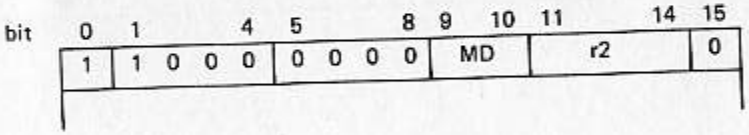
MUR *Multiply register/register* **MUR** P851M P852M P856M P857M (softw.

Syntax: [label]␣ MUR[*]␣ r2

The contents of the register specified by r2 (direct addressing), or the contents of the memory address indicated in r2 (indirect addressing) are multiplied by the contents of A2. The result is loaded as a 31-bit product in A1, A2. The most significant bit of A2 is reset to zero. The sign of the product is stored in the sign bit of A1.
Overflow occurs if the result $> 2^{30} - 1$.
In that case the two registers contain only the 30 least significant bits while the sign bit may or may not be correct.

Type	*Function*		*MD*	*Syntax*	
T1	(A2) x (r2)	→ A1, A2	00	MUR	r2
T3	(A2) x ((r2))	→ A1, A2	01	MUR*	r2

Condition register:
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow



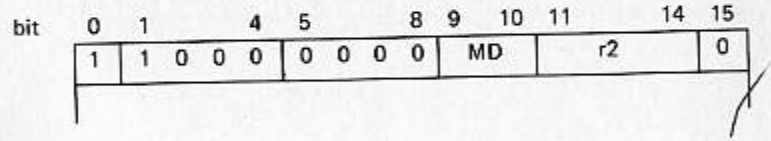
MU *Multiply* **MU** P851M P852M P856M P857M (softw. sim)

Syntax: [label]␣ MU[*]␣ m[, r2]

The contents of register A2 are multiplied by the contents of the effective memory address. The result of this multiplication is loaded as a 31-bit product in registers A1, A2. The most significant bit of A2 is reset to zero. The sign of the product is stored in the sign bit of register A1.
Overflow occurs if result $> 2^{30} - 1$.
In that case the two registers contain only the 30 least significant bits while the sign bit may or may not be correct.

Type	*Function*		*MD*	*Syntax*	
T4	(A2) x (m)	→ A1, A2	10	MU	m
T5	(A2) x (m + (r2))	→ A1, A2	10	MU	m, r2
T6	(A2) x ((m))	→ A1, A2	11	MU*	m
T7	(A2) x ((m + (r2)))	→ A1, A2	11	MU*	m, r2

Condition register:
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow



DVK *Divide by constant* **DVK** P851M P852M P856M P857M (softw. sim)

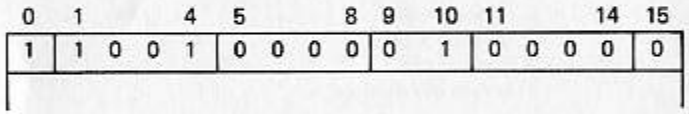
Syntax: [label]␣ DVK ␣ lk

The contents of the registers A1, A2 are divided by the constant lk. The quotient is placed in register A2, the remainder in register A1. Overflow occurs when the quotient exceeds 15 bits. In that case the contents of A1 and A2 are not significant. See also the note under DV on page 3.0.24.

Type	*Function*	*Q*	*R*
T2	(A1, A2) / lk	A2	A1

Condition register:

CR = 0 if (A2) = 0
1 if (A2) > 0
2 if (A2) < 0
3 in case of overflow



DVR *Divide register/register* **DVR** P851M P852M P856M P857M (softw. sim)

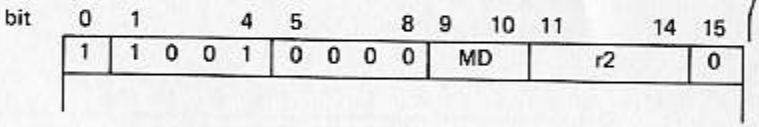
Syntax: [label]␣ DVR[*]␣ r2

The contents of the registers A1 and A2 are divided by the contents of r2 (direct addressing), or the contents of the memory address indicated in r2 (indirect addressing). The quotient is placed in register A2, the remainder in A1.
Overflow occurs if the quotient exceeds 15 bits. In that case the contents of A1 and A2 are not significant.
See also the note under DV on page 3.0.24.

Type	*Function*	*Q*	*R*	*MD*	*Syntax*	
T1	(A1, A2) / (r2)	→ A2	A1	00	DVR	r2
T3	(A1, A2) / ((r2))	→ A2	A1	01	DVR*	r2

Condition register:

CR = 0 if (A2) = 0
1 if (A2) > 0
2 if (A2) < 0
3 in case of overflow



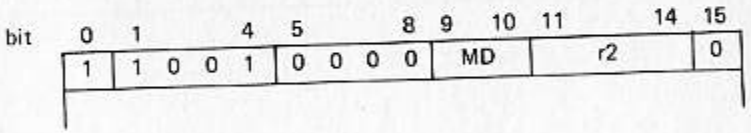
DV — *Divide* — **DV** — P851M, P852M, P856M, P857M (softw. sim)

Syntax: [label]␣ DV[*]␣ m[, r2]

The contents of the registers A1 and A2 are divided by the contents of the effective memory address. The quotient is placed in register A2. The remainder in register A1.
The sign of the remainder is equal to the original sign of A1, A2, except when the remainder is equal to zero (always positive).
Overflow occurs when the quotient exceeds 15 bits. In that case the contents of A1 and A2 are destroyed except when the division is equal to zero.

Type	*Function*	*Q*	*R*	*MD*	*Syntax*
T4	(A1, A2) / (m)	→ A2	A1	10	DV m
T5	(A1, A2) / (m + (r2))	→ A2	A1	10	DV m, r2
T6	(A1, A2) / ((m))	→ A2	A1	11	DV* m
T7	(A1, A2) / ((m + (r2)))	→ A2	A1	11	DV* m, r2

Condition register:
CR = 0 if (A2) = 0
1 if (A2) > 0
2 if (A2) < 0
3 in case of overflow



Note:
An erroneous result is given when the most significant word of the dividend is equal to the twos complement of the divisor.

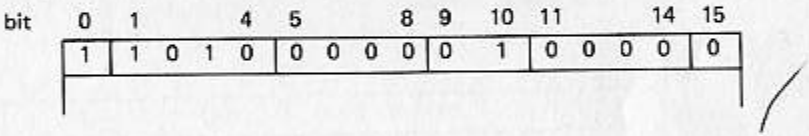
DAK — *Double add with constant* — **DAK** — P851M, P852M, P856M, P857M (softw. sim)

Syntax: [label]␣ DAK ␣ lk_1 , lk_2

A constant consisting of 32 bits (bit 0 of first word is sign bit; bit 0 of second word is not used) is added to the contents of registers A1 and A2. The sum is placed in A1, A2. Bit 0 of A2 is set to zero. Bit 0 of A1 is the sign bit.

Type	*Function*
T2	lk1, lk2 + (A1, A2) → A1, A2

Condition register:
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow



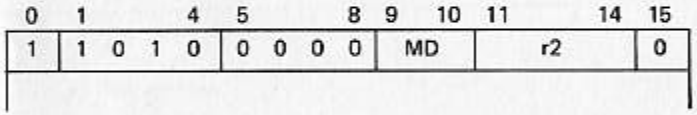
DAR *Double add register/register* **DAR** P851M P852M P856M P857M (softw. sim)

Syntax: [label]␣ DAR[*]␣ r2

The contents of two consecutive registers, the first one specified in r2 (direct addressing), or the contents of two consecutive words. The address of the first one being indicated in r2 (indirect addressing) are added to the contents of A1 and A2. Bit 0 of A2 is set to zero. The sign bit of the result is the sign bit of A1.

Type	*Function*	*MD*	*Syntax*
T1	(r2, r2 + 1) + (A1, A2) → A1, A2	00	DAR r2
T3	((r2),(r2) + 2) + (A1, A2) → A1, A2	01	DAR* r2

Condition register:
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow



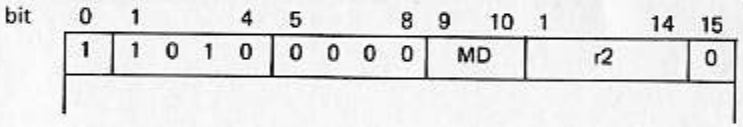
DA *Double add* **DA** P851M P852M P856M P857M (softw. sim)

Syntax: [label]␣ DA[*]␣ m[, r2]

The contents of the effective memory address and the contents of the effective memory address + 2 are added to the contents of the registers A1 and A2. The sum is placed in those registers.
The sign bit in A2 is set to zero. The sign bit of the parameters and result is the sign bit of A1.

Type	*Function*	*MD*	*Syntax*
T4	(m, m + 2) + (A1, A2) → A1, A2	10	DA m
T5	(m + (r2), m + (r2) + 2) + (A1, A2) → A1, A2	10	DA m, r2
T6	((m),(m) + 2) + (A1, A2) → A1, A2	11	DA* m
T7	((m + (r2)), (m + (r2) + 2))+ (A1, A2) → A1, A2	11	DA* m, r2

Condition register:
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow



DSK *Double subtract with constant* **DSK** P851M P852M P856M P857M (softw. sim)

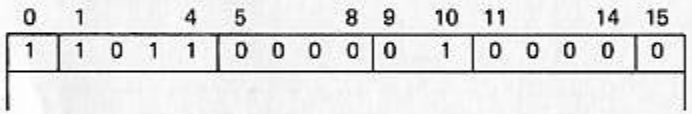
Syntax: [label]␣ DSK ␣ lk_1, lk_2

A constant consisting of 32 bits (bit 0 of first word is sign bit; bit 0 of second word is not used) is subtracted from the contents of registers A1, A2. The result is placed in A1, A2. Bit 0 of A2 is set to zero. Bit 0 of A1 is the sign bit.

Type	*Function*
T2	(A1, A2) – lk1, lk2 → A1, A2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow



DSR *Double subtract reg./reg.* **DSR** P851M P852M P856M P857M (softw. sim)

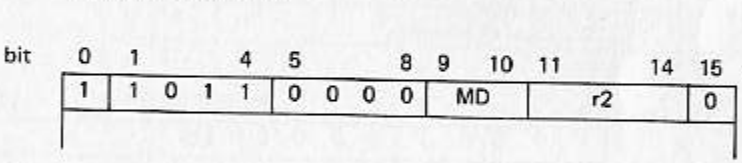
Syntax: [label]␣ DSR [*]␣ r2

The contents of two consecutive registers, the first one being specified in r2 (direct addressing), or the contents of two consecutive words, the address of the first one being indicated in r2 (indirect addressing) are subtracted from the contents of the registers A1 and A2. Bit 0 of A2 is reset to zero. Bit 0 of A1 is the sign bit.

Type	*Function*	*MD*	*Syntax*
T1	(A1, A2) – (r2 , r2 + 1) → A1, A2	00	DSR r2
T3	(A1, A2) – ((r2), (r2 + 1)) → A1, A2	01	DSR* r2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow



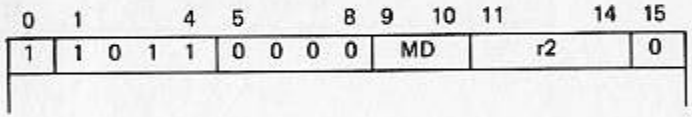
DS — *Double subtract* — **DS** — P851M P852M P856M P857M (softw. sim.)

Syntax: [label] ␣ DS[•] ␣ m[, r2]

The contents of the effective memory address and the contents of the effective memory address + 2 are subtracted from the contents of the registers A1 and A2. The result is placed in A1, A2. The sign bit in A2 is set to zero.
The sign bit of the parameters and the result is the sign bit of register A1.

Type	*Function*	*MD*	*Syntax*
T4	(A1, A2) − (m , m + 2) → A1, A2	10	DS m
T5	(A1, A2) − (m + (r2) , m + (r2) + 2) → A1, A2	10	DS m, r2
T6	(A1, A2) − ((m), (m) + 2) → A1, A2	11	DS* m
T7	(A1, A2) − ((m + (r2)), (m + (r2)) + 2) → A1, A2	11	DS* m, r2

Condition register:
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow



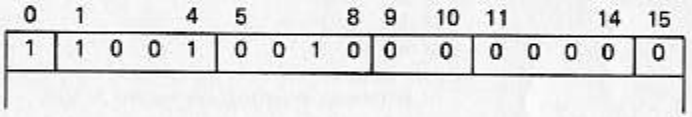
FFL — *Integer to floating point (F.P.P. option)* — **FFL** — P857M

Syntax: [label] ␣ FFL

The contents of the registers A1, and A2, being a double precision integer, are sent to the Floating Point Processor where the integer is converted into a floating point operand. The result is stored in three accumulators FPA1, FPA2 and FPA3 on the Floating Point Processor.

Type	*Function*
T1	(A1), (A2) → F.P. operand → FPA1, FPA2, FPA3

Condition register:
CR = 0 if result = 0
1 if result > 0
2 if result < 0



FFX *Floating point to integer (F.P.P. option)* **FFX** P857M

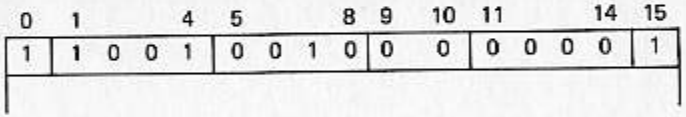
Syntax: [label]␣ FFX

The floating point operand contained in three accumulators FPA1, FPA2 and FPA3, situated on the Floating Point Processor, is converted into a double precision integer. The result is placed in the registers A1 and A2. During this operation the number may be truncated (loss of least significant bits).
An overflow occurs if the integer is greater than $2^{30}-1$ or smaller than -2^{30}. An interrupt is generated by the Floating Point Processor when an abnormal condition occurs. CR is set to 3.

Type	*Function*
T1	(FPA1, FPA2, FPA3) → integer → A1, A2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 abnormal condition:
– arithmetic overflow (exponent > 30)



FADR FADRS *Floating point addition/register (F.P.P. option)* **FADR FADRS** P857M

Syntax: [label]␣ FADR[S]␣ r2

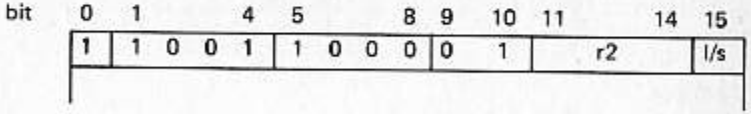
The floating point operand contained in three accumulators FPA1, FPA2 and FPA3 on the Floating Point Processor, is added to the floating point operand present in three consecutive memory locations. The first memory location is indicated by r2. The result is placed either in FPA1, FPA2 and FPA3 or in three consecutive memory locations, depending on the state of the l/s indicator.
An interrupt is generated by the Floating Point Processor when an abnormal condition occurs. CR is set to 3.

Type	*Function*
T3	(FPA1,FPA2,FPA3) + ((r2)),((r2)+2),((r2)+4) → FPA1,FPA2,FPA3
T3S	(FPA1,FPA2,FPA3) + ((r2)),((r2)+2),((r2)+4) → (r2),(r2)+2,(r2)+4

Type	*l/s*	*Syntax*
T3	0	FADR r2
T3S	1	FADRS r2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result <0
3 abnormal conditions:
– unnormalized operand (operation aborted)
– arithmetic overflow (result exponent > or = 2^{15})
– arithmetic underflow (result exponent < -2^{15})



FAD — *Floating point addition (F.P.P. option)* — **FAD** **P857M**

Syntax: [label]␣ FAD [S] [*]␣ m [, r2]

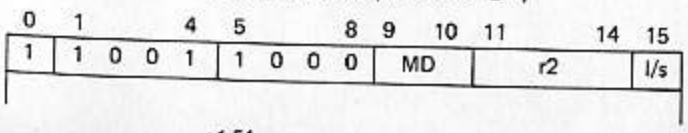
The floating point operand contained in the floating point accumulators FPA1, FPA2, FPA3 on the Floating Point Processor, is added to the floating point operand contained in three consecutive memory locations, the first one being indicated by the effective memory address. The sum is placed either in accumulators FPA1, FPA2 and FPA3 or in three consecutive memory locations pointed to by the effective memory address, depending on the state of the I/s indicator.
An interrupt is generated by the Floating Point Processor when an abnormal condition occurs. CR is set to 3.

Type	*Function*
T4	(FPA1,FPA2,FPA3) + (m),(m + 2),(m + 4) → FPA1,FPA2,FPA3
T4S	(FPA1,FPA2,FPA3) + (m),(m + 2),(m + 4) → m, m + 2, m + 4
T5	(FPA1,FPA2,FPA3) + (m + (r2)),(m + (r2) + 2), (m + (r2) + 4) → → FPA1,FPA2,FPA3
T5S	(FPA1,FPA2,FPA3) + (m + (r2)),(m + (r2) + 2), (m + (r2) + 4) → → m + (r2), m + (r2) + 2, m + (r2) + 4
T6	(FPA1,FPA2,FPA3) + ((m)),((m + 2)),((m + 4)) → FPA1,FPA2,FPA3
T6S	(FPA1,FPA2,FPA3) + ((m)),((m + 2)),((m + 4)) → (m), (m + 2), (m + 4)
T7	(FPA1,FPA2,FPA3) + ((m + (r2))),((m + (r2) + 2)),((m + (r2) + 4)) → → FPA1,FPA2,FPA3
T7S	(FPA1,FPA2,FPA3) + ((m + (r2))),((m + (r2) + 2)),((m + (r2) + 4)) → → (m + (r2)), (m + (r2) + 2), (m + (r2) + 4)

Type	*MD*	*I/s*	*Syntax*	
T4	10	0	FAD	m
T4S	10	1	FADS	m
T5	10	0	FAD	m, r2
T5S	10	1	FADS	m, r2
T6	11	0	FAD*	m
T6S	11	1	FADS*	m
T7	11	0	FAD*	m, r2
T7S	11	1	FADS*	m, r2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 abnormal condition:
- unnormalized operand (operation aborted)
- arithmetic overflow (result exponent > or = 2^{15})
- arithmetic underflow (result exponent < -2^{15})



FSUR FSURS — *Floating point subtract/register (F.P.P. option)* — **FSUR FSURS** **P857M**

Syntax: [label]␣ FSUR [S]␣ r2

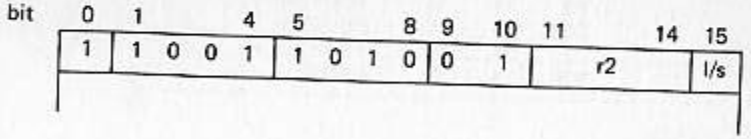
The floating point operand contained in three consecutive memory locations, the first one being specified by r2, is subtracted from the floating point operand in the three accumulators FPA1, FPA2 and FPA3 on the Floating Point Processor. The result is placed either in FPA1, FPA2, FPA3 or in three consecutive memory locations, depending on the state of the I/s indicator.
An interrupt is generated by the Floating Point Processor when an abnormal condition occurs. CR is set to 3.

Type	*Function*
T3	(FPA1,FPA2,FPA3) − ((r2)),((r2) + 2),((r2) + 4) → FPA1,FPA2,FPA3
T3S	(FPA1,FPA2,FPA3) − ((r2)),((r2) + 2),((r2) + 4) → (r2),(r2) + 2,(r2) + 4

Type	*I/s*	*Syntax*	
T3	0	FSUR	r2
T3S	1	FSURS	r2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 abnormal condition:
- unnormalized operand (operation aborted)
- arithmetic overflow (result exponent > or = 2^{15})
- arithmetic underflow (result exponent < -2^{15})



FSU *Floating point subtract (F.P.P. option)* **FSU** **P857M**

Syntax: [label]␣ FSU[S] [*]␣ m[, r2]

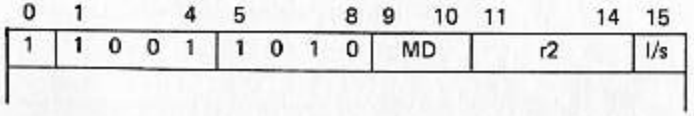
The floating point operand contained in three consecutive memory locations, the first of which is specified by the effective memory address, is subtracted from the floating point operand present in three accumulators FPA1, FPA2 and FPA3 on the Floating Point Processor. The result is placed either in FPA1, FPA2 and FPA3 or in three consecutive memory locations pointed to by the effective memory address, depending on the state of the I/s indicator. An interrupt is generated by the Floating Point Processor when an abnormal condition occurs.

Type	*Function*
T4	(FPA1,FPA2,FPA3) – (m),(m + 2),(m + 4) → FPA1,FPA2,FPA3
T4S	(FPA1,FPA2,FPA3) – (m),(m + 2),(m + 4) → m, m + 2, m + 4
T5	(FPA1,FPA2,FPA3) – (m + (r2)),(m + (r2) + 2),(m + (r2) + 4) → → FPA1,FPA2,FPA3
T5S	(FPA1,FPA2,FPA3) – (m + (r2)),(m + (r2) + 2),(m + (r2) + 4) → → m + (r2), m + (r2) + 2, m + (r2) + 4
T6	(FPA1,FPA2,FPA3) – ((m)),((m + 2)),((m + 4)) → FPA1,FPA2,FPA3
T6S	(FPA1,FPA2,FPA3) – ((m)),((m + 2)),((m + 4)) → (m),(m + 2),(m + 4)
T7	(FPA1,FPA2,FPA3) – ((m + (r2))),((m + (r2) + 2)),((m + (r2) + 4)) → → FPA1,FPA2,FPA3
T7S	(FPA1,FPA2,FPA3) – ((m + (r2))),((m + (r2) + 2)),((m + (r2) + 4)) → → (m + (r2)), (m + (r2) + 2), (m + (r2) + 4)

Type	*MD*	*I/s*	*Syntax*
T4	10	0	FSU m
T4S	10	1	FSUS m
T5	10	0	FSU m, r2
T5S	10	1	FSUS m, r2
T6	11	0	FSU* m
T6S	11	1	FSUS* m
T7	11	0	FSU* m, r2
T7S	11	1	FSUS* m, r2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 abnormal condition:
– unnormalized operand (operation aborted)
– arithmetic overflow (result exponent > or = 2^{15})
– arithmetic underflow (result exponent < -2^{15})



FMUR FMURS *Floating point multiply/register (F.P.P. option)* **FMUR FMURS** **P857M**

Syntax: [label]␣ FMUR[S]␣ r2

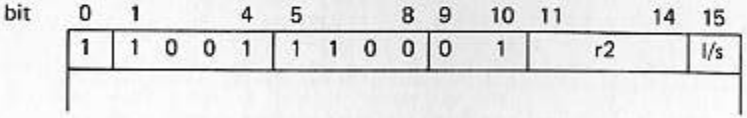
The floating point operand contained in the floating point accumulators FPA1, FPA2, FPA3 is multiplied by the floating point operand present in three consecutive memory locations, the first one being indicated in r2. The result is placed either in FPA1, FPA2, FPA3 or in three consecutive memory locations, pointed at by r2, depending on the state of the I/s indicator.
An interrupt is generated by the Floating Point Processor when an abnormal condition occurs. CR is set to 3.

Type	*Function*
T3	(FPA1,FPA2,FPA3) x ((r2)),((r2) + 2),((r2) + 4) → FPA1,FPA2,FPA3
T3S	(FPA1,FPA2,FPA3) x ((r2)),((r2) + 2),((r2) + 4) → (r2),(r2) + 2,(r2) + 4

Type	*I/s*	*Syntax*
T3	0	FMUR r2
T3S	1	FMURS r2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 abnormal condition:
– unnormalized operand (operation aborted)
– arithmetic overflow (result exponent > or = 2^{15})
– arithmetic underflow (result exponent < -2^{15})



FMU	*Floating point multiply (F.P.P. option)*	FMU	P857M

Syntax: [label]␣ FMU[S] [*]␣ m[, r2]

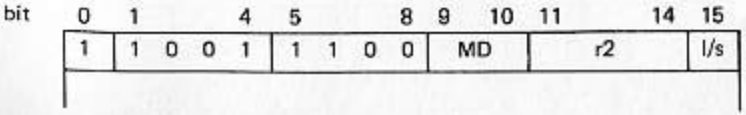
The floating point operand contained in the floating point processor accumulators FPA1, FPA2, FPA3, is multiplied by the floating point operand present in three consecutive memory locations, the first of which is indicated by the effective memory address. The result is placed either in FPA1, FPA2 and FPA3 or in three consecutive memory locations, pointed at by the effective memory address, depending on the state of the I/s indicator.
An interrupt is generated by the Floating Point Processor when an abnormal condition occurs. CR is set to 3.

Type	*Function*
T4	(FPA1,FPA2,FPA3) x (m),(m + 2),(m + 4) → FPA1,FPA2,FPA3
T4S	(FPA1,FPA2,FPA3) x (m),(m + 2),(m + 4) → m, m + 2, m + 4
T5	(FPA1,FPA2,FPA3) x (m + (r2)),(m + (r2) + 2),(m + (r2) + 4) → → FPA1,FPA2,FPA3
T5S	(FPA1,FPA2,FPA3) x (m + (r2)),(m + (r2) + 2),(m + (r2) + 4) → → m + (r2), m + (r2) + 2, m + (r2) + 4
T6	(FPA1,FPA2,FPA3) x ((m)),((m + 2)),((m + 4)) → FPA1,FPA2,FPA3
T6S	(FPA1,FPA2,FPA3) x ((m)),((m + 2)),((m + 4)) → (m),(m + 2),(m + 4)
T7	(FPA1,FPA2,FPA3) x ((m + (r2))),((m + (r2) + 2)),((m + (r2) + 4)) → → FPA1,FPA2,FPA3
T7S	(FPA1,FPA2,FPA3) x ((m + (r2))),((m + (r2) + 2)),((m + (r2) + 4)) → → (m + (r2)), (m + (r2) + 2), (m + (r2) + 4)

Type	*MD*	*I/s*	*Syntax*	
T4	10	0	FMU	m
T4S	10	1	FMUS	m
T5	10	0	FMU	m, r2
T5S	10	1	FMUS	m, r2
T6	11	0	FMU*	m
T6S	11	1	FMUS*	m
T7	11	0	FMU*	m, r2
T7S	11	1	FMUS*	m, r2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 abnormal condition:
– unnormalized operand (operation aborted)
– arithmetic overflow (result exponent > or = 2^{15})
– arithmetic underflow (result exponent < -2^{15})



FDVR FDVRS	*Floating point divide/register (F.P.P. option)*	FDVR FDVRS	P857M

Syntax: [label]␣ FDVR[S]␣ r2

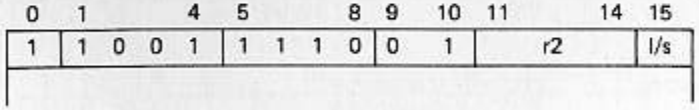
The floating point operand contained in the floating point processor accumulators FPA1, FPA2, FPA2, is divided by the floating point operand present in three consecutive memory locations, the first of which is indicated by r2.
The quotient is placed either in FPA1, FPA2, FPA3 or in three consecutive memory locations, pointed at by r2, depending on the state of the I/s indicator.
An interrupt is generated by the Floating Point Processor when an abnormal condition occurs. CR is set to 3.

Type	*Function*
T3	(FPA1, FPA2,FPA3) / ((r2)),((r2) + 2),((r2) + 4) → FPA1,FPA2,FPA3
T3S	(FPA1, FPA2,FPA3) / ((r2)),((r2) + 2),((r2) + 4) → (r2),(r2) + 2,(r2) + 4

Type	*I/s*	*Syntax*	
T3	0	FDVR	r2
T3S	1	FDVRS	r2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 abnormal condition:
– unnormalized operand (operation aborted)
– arithmetic overflow (result exponent > or = 2^{15})
– arithmetic underflow (result exponent < -2^{15})
– Divisor = 0



Syntax: [label]␣ FDV[S] [*]␣ m[, r2]

The floating point operand contained in the floating point processor accumulators FPA1, FPA2, FPA3, is divided by the floating point operand present in three consecutive memory locations, the first one being pointed at by the effective memory address. The result is placed either in FPA1,FPA2,FPA3 or in the three consecutive memory locations pointed at by the effective memory address, depending on the state of the I/s indicator.
An interrupt is generated by the Floating Point Processor when an abnormal condition occurs. CR is set to 3.

Type	*Function*
T4	(FPA1,FPA2,FPA3) / (m),(m + 2),(m + 4) → FPA1,FPA2,FPA3
T4S	(FPA1,FPA2,FPA3) / (m),(m + 2),(m + 4) → m, m + 2, m + 4
T5	(FPA1,FPA2,FPA3) / (m + (r2)),(m + (r2) + 2),(m + (r2) + 4) → → FPA1,FPA2,FPA3
T5S	(FPA1,FPA2,FPA3) / (m + (r2)),(m + (r2) + 2),(m + (r2) + 4) → → m + (r2), m + (r2) + 2, m + (r2) + 4
T6	(FPA1,FPA2,FPA3) / ((m)),((m + 2)),((m + 4)) → FPA1,FPA2,FPA3
T6S	(FPA1,FPA2,FPA3) / ((m)),((m + 2)),((m + 4)) → (m),(m + 2),(m + 4)
T7	(FPA1,FPA2,FPA3) / ((m + (r2))),((m + (r2) + 2)),((m + (r2) + 4)) → → FPA1,FPA2,FPA3
T7S	(FPA1,FPA2,FPA3) / ((m + (r2))),((m + (r2) + 2)),((m + (r2) + 4)) → → (m + (r2)), (m + (r2) + 2), (m + (r2) + 4)

Type	*MD*	*I/s*	*Syntax*	
T4	10	0	FDV	m
T4S	10	1	FDVS	m
T5	10	0	FDV	m, r2
T5S	10	1	FDVS	m, r2
T6	11	0	FDV*	m
T6S	11	1	FDVS*	m
T7	11	0	FDV*	m, r2
T7S	11	1	FDVS*	m, r2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 abnormal condition:
- unnormalized operand (operation aborted)
- arithmetic overflow (result exponent > or = 2^{15})
- arithmetic underflow (result exponent < -2^{15})
- Divisor = 0