Preface

This volume is intended to be used together with the P800M publications concerning programming.

Part 1 describes in great detail the powerful instruction set for the P800M computers and shows the programmer the functional operation, the syntax, the setting of the condition register, the instruction time and examples.
The instructions are grouped in the following operational categories :

Load and store instructions
Arithmetic instructions
Logical instructions
Character handling instructions
Branch instructions
Shift instructions
Table handling instructions
External transfer instructions
Control instructions
I/O instructions
String instructions

Table of Contents

PART 1 INSTRUCTION SET

Key to symbols used in the instruction set

Label	Identifier, or label, consisting of max. 6 characters of which the first must always be a letter. All instructions, and most of the assembler directives, may be preceded by a label.
*	Asterisk. Indicates: – indirect addressing – current value of location counter
[ ]	The syntactic item(s) between these brackets may be omitted
{ }	Choose one of the items between these brackets
r1	Register A1 . . . A15
r2	Register A1 . . . A15. Used as an index register in memory reference instructions.
r3	Register A1 . . . A7
m	Memory expression
k	Constant in bits 8–15 (short constant)
lk	Constant or address in bits 0–15 of the word following the instruction (long constant)
P	P-register. (Instruction counter)
T1	Register to register operation.
T2	Long constant instruction.
T3	Register addressing.
T3A	Register r2 is not the stackpointer A15
T3B	Register r2 is the stackpointer A15
TxS	The result must be stored in memory
T4	Direct addressing
T5	Indexed addressing
T6	Indirect addressing
T7	Indirect indexed addressing
T8	Short constant instruction
l/s	Load/store indicator. Load: bit 15 = 0 Store: bit 15 = 1
MD	Addressing mode
∧	Logical AND
∨	Logical OR
∨	Exclusive OR
**	Compare
/	Divide
x	Multiply
+	Add

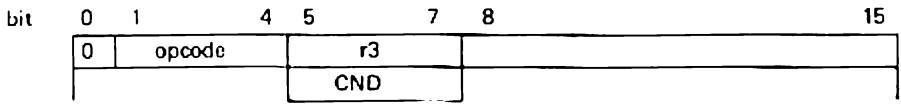
Instruction formats

Machine instructions conform to one of the following two formats:

- format 0
- format 1.

Format 0 instructions

Instructions of this type consist of one word, where the 16 bits indicate the following functions :

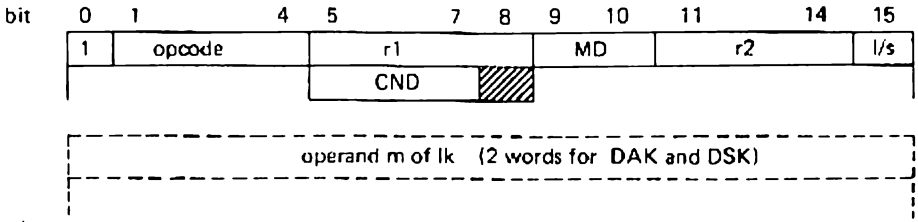


where

bit 0 – indicates the instruction format
bits 1–4 – operation code
bits 5–7 – one of the registers A1–A7 or the condition value in a Branch instruction.
bits 8–15 – the contents of this field varies according to the type of instruction and may contain one of the following values:
- an 8-bit *positive* constant (constant instruction)
- an even displacement value (branch instruction)
- an indication of the shift required (shift instruction)
- device address (I/O instruction) + function bits
- fixed parameters (miscellaneous instruction)

Format 1 instructions

Format 1 instructions perform a number of operations by reference to two of the 16 registers available for user access: one of these registers may point to a data item either in a word following the instruction or elsewhere in memory as it is possible to use that register as an index register.



where:

bit 0 – indicates the instruction format

bits 1–4 – operation code

bits 5–8 – one of the registers A1 . . . A15 specified as follows:
registers A1 . . . A7 are in group 0 and registers A8 . . . A15 are in group 1.
The group to which a register belongs is indicated by bit 8.
This may be either 0 (group 0) or 1 (group 1)
– in branch instruction, however, bits 5 to 7 inclusive indicate a condition value and bit 8 is not used.

bits 9–10 – addressing mode code. These bits will specify direct or indirect addressing, i.e. whether the word following the instruction, or another memory word, has to be taken into account.

bits 11–14 – the number of one of the 16 registers, expressed in the same way as in bits 5–8.

bit 15 – load/store indicator. Used in certain instructions to indicate that the result of the operation is to be placed either in the register shown by bits 5–8 (l/s = 0) or in a memory word (l/s = 1).

This type of instruction may be followed by a data word (16 bits) containing an address (m) or a positive or negative value. In the case of an address, bit 15 is not significant, except for character handling instructions.

The binary values of bits 5 through 8 in r1 and 11 through 14 for r2 are: 4 2 1 8, and in r3 4 2 1.

Example: A3 in r1 or r2 is written as 0110 and A12 as 1001. For r3 this is 011.
A12 cannot be specified in the field r3.

Registers

16 registers are available for use by the programmer. These 16 registers, which have the predefined symbols A0 through A15, are called the scratchpad. They may be addressed from either the instruction being carried out or from the toggle switches on the control panel.
The specific designation of registers within the scratchpad is:

P-register (A0)

This register is used to hold the address of the next instruction to be executed. It is incremented in steps of two if the program is to carry out in sequence, or altered to hold the required new address if a branch is to be performed.
The instruction counter (P) points always already to the next instruction before execution of an instruction.

Working registers (A1–A14)

The working registers may be used in any of the following ways:

- as accumulator where the data to be processed can be found in a register,
- as pointers where the contents of the specified registers contain the operand address rather than the operand itself.
- as index registers where the contents of the specified registers and the contents of the word following the instruction are summed to produce the operand address.

Register A15

This register is used by the interrupt system as the stackpointer and, as such, it is updated by the system whenever it is used for memory addressing.
It may be addressed by instruction in the same way as the registers A1 through A14.

Type of instruction

The instruction in the instruction set may use various methods of forming one of the operands to be used. To make a clear distinction between these methods, each instruction in the instruction set description has received a notation T1 thru T8 to indicate the manner in which the operand is formed. The latter is usually governed by the values of the format, address mode and the r2 field (bits 11 thru 14) in the instruction. The result of this operation may be an address which is called the effective memory address.

Type	Format	Mode	r2 field	Description
T1	1	00	≠ 0	Register to register operation
T2	1	01	0000	Long constant instruction
T3 (T3A) (T3B)	1	01	≠ 0	Address in register r2 (The register specified is not A15) (The register specified is A15)
T4	1	10	0000	Address in next word (direct addressing)
T5	1	10	≠ 0	Indexed addressing
T6	1	11	0000	Indirect addressing
T7	1	11	≠ 0	Indexed indirect addressing
T8	0	–	–	Short constant

T1 Register to register operation
The operand is the value in the register specified by r2.

T2 Long constant instruction
The operand is the value contained in the least significant word of the double length instruction.

T3 Address in register
The operand is held in memory. The memory address of the operand is the value in the register specified by r2.
T3A r2 ≠ A15
T3B r2 = A15

T4 Address in next word (direct addressing)
The operand is held in memory. The memory address of the operand is the value in the least significant word of the double length instruction.

T5 Indexed address in next word (indexed addressing)
The operand is held in memory. The memory address of the operand is found by adding the value in the register specified by r2 to the value in the least significant word of the double length instruction.

LD — *Load register* — **LD**

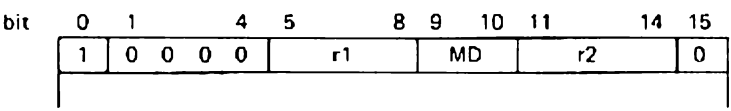
P851M
P852M
P856M
P857M

Syntax: [label]␣ LD [*]␣ r1, m [,r2]

The contents of the register specified by r1 are replaced by the contents of the effective memory address. This effective memory address can be found as follows:

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

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



Remark:
Restricted to system mode if r1 = A15.

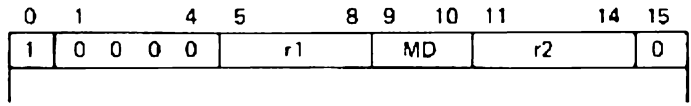
LDR *Load register/register* **LDR** P851M P852M P856M P857M

Syntax: [label]␣ LDR [•]␣ r1, r2

The 16 bits of the register specified by r1 are replaced either by the contents of the register specified by r2 (direct addressing) or by the contents of the effective memory address which can be found in the register specified by r2 (indirect addressing). In the last addressing mode, if r2 specifies the A15 register, the latter is assumed to be the stack. In this case, the pointer is updated (i.e. incremented by one word to point to the latest entry) before the transfer of data occurs.

Type	*Function*		*MD*	*Syntax*
T1	(r2)	→ r1	00	LDR r1, r2
T3A	((r2))	→ r1	01	LDR* r1, r2
T3B	(A15) + 2 → A15, ((A15))	→ r1	01	LDR* r1, A15

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



Remark:
Restricted to system mode if r1 = A15 or if type 3B.

LDK LDKL

Load constant

LDK LDKL

P851M P852M P856M P857M

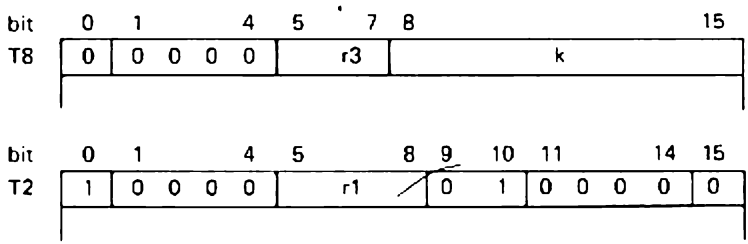
Syntax: [label]␣ LDK ␣ r3, k – T8
[label]␣ LDKL ␣ r1, lk – T2

T8 The positive constant k is loaded into bits 8 through 15 of the register specified in r3. The bits 0 through 7 are reset to zero.

T2 The positive or negative constant, which can be found in the word following the instruction, replaces the contents of the register specified by r1.

Type	*Function*	*Syntax*
T8	$k \rightarrow r3_{n-15}$ $0 \rightarrow r3_{0-7}$	LDK r3, k
T2	lk → r1	LDKL r1, lk

Condition register:
T8 Unchanged
T2 CR = 0 if lk = 0
1 if lk > 0
2 if lk < 0



Remark:
Restricted to system mode if r1 = A15.

ST

Store register

ST

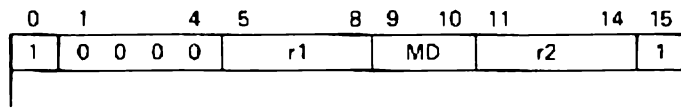
P851M
P852M
P856M
P857M

Syntax: [label]␣ ST [•]␣ r1, m [, r2]

The 16 bits of the register specified by r1 replace the contents of the effective memory address.

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

Condition register: Unchanged



Remark:
Restricted to system mode if r1 = A15.

STR *Store register/register* **STR**

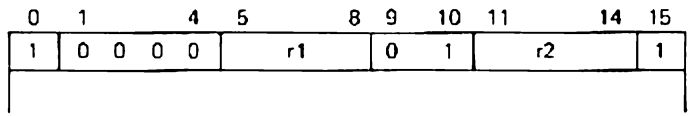
P851M
P852M
P856M
P857M

Syntax: [label]␣ STR ␣ r1, r2

The 16 bits of the register specified by r1 replace the contents of the memory address indicated in the register specified by r2 (indirect addressing). If A15 (stack pointer) is specified by r2 it is updated.

Type	*Function*	*Syntax*
T3A	(r1) → (r2)	STR r1, r2
T3B	(r1) → (A15), (A15) − 2 → A15	STR r1, A15

Condition register: Unchanged



Remark:
- An interrupt 'stack overflow' is generated when, for T3B type, the address reached by the pointer = </100. Bit 13 is set to 1 in PSW.
- Restricted to system mode if r1 = A15 or if type 3B.

ML — *Multiple load* — **ML** — P851M, P852M, P856M, P857M (softw.

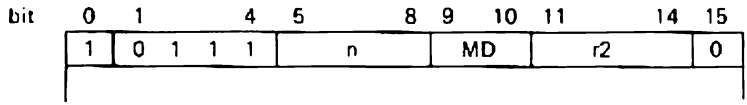
Syntax: [label]␣ ML [•]␣ n, m [, r2]

The contents of n consecutive registers (the first one being A1) are replaced by the contents of n consecutive memory locations (the first location is indicated by the effective memory address).

Type	*Function*			*MD*	*Syntax*
T4	(m) ... (m + n)	→	A1... An	10	ML n, m
T5	(m + (r2)) ... (m + (r2) + n)	→	A1... An	10	ML n, m, r2
T6	((m)) ... ((m) + n)	→	A1... An	11	ML• n, m
T7	((m + (r2))) ... ((m + (r2)) + n)	→	A1... An	11	ML• n, m, r2

n = number of registers (1 through 15)

Condition register: CR = 0 if (A1) = 0
1 if (A1) > 0
2 if (A1) < 0



Remark:
Restricted to system mode if n = 15.

MLK *Multiple load constant* **MLK** P851M P852M P856M P857M (softw. sim

Syntax: [label] ␣ MLK ␣ n

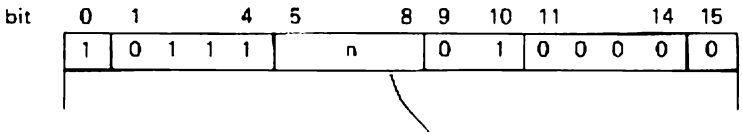
The contents of n successive registers are replaced by n values which must be given immediately after this instruction by means of a data statement. If n = 0 the instruction is trapped.

Type	*Function*	*Syntax*
T2	Ik1, Ik2 ,..., Ikn → A1, A2 ,..., An	MLK n DATA x,...,xn

n = number of registers (1 through 15)

Condition register:

CR = 0 if (A1) = 0
1 if (A1) > 0
2 if (A1) < 0



Remark:
Restricted to system mode if n = 15.

MLR *Multiple load/register* **MLR** P851M P852M P856M P857M (softw. sim)

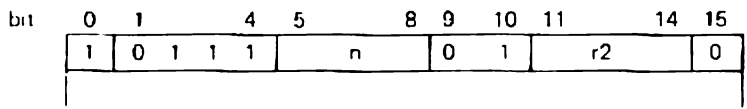
Syntax: [label] ␣ MLR ␣ n, r2

The contents of n consecutive registers (the first one being A1), are replaced by the contents of n consecutive memory locations. The first address of those locations is indicated by the contents of r2.
If r2 is the stackpointer A15, the system stackpointer is updated.

Type	*Function*		*Syntax*
T3A	((r2))	→ A1	MLR n, r2
	((r2) + 2)	→ A2	
	–		
	–		
	((r2) +2n – 2)	→ An	
T3B	(A15) + 2n	→ A15	MLR n, A15
	((A15))	→ A1	
	((A15) – 2)	→ A2	
	–		
	–		
	((A15) – 2n + 2)	→ An	

n = number of registers (1 through 15)

Condition register: CR = 0 if (A1) = 0
1 if (A1) > 0
2 if (A1) < 0



Remark:
- Restricted to system mode if n = 15 or if r2 = A15
- If 3B type, the contents must be even (P851M).

MS *Multiple store* **MS** P851M P852M P856M P857M (softw. sim)

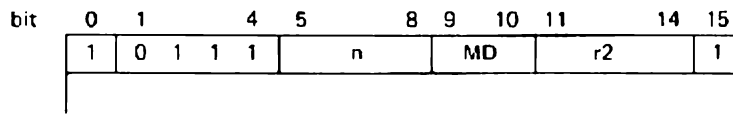
Syntax: [label] ␣ MS [*] ␣ n, m [, r2]

The contents of n consecutive memory locations (the first one is given by the effective memory address) are replaced by the contents of n consecutive registers.

Type	*Function*	*MD*	*Syntax*
T4	A1 . . An → m, . . , m + n	10	MS n, m
T5	A1 . . An → m + (r2), . . , m + (r2) + n	10	MS n, m, r2
T6	A1 . . An → (m), . . , (m) + n	11	MS* n, m
T7	A1 . . An → (m + (r2)), . . , (m + (r2)) + n	11	MS* n, m, r2

n = number of registers (1 through 15)

Condition register: Unchanged



Remark:
Restricted to system mode if n = 15.

MSR *Multiple store register* **MSR** P851M P852M P856M P857M (softw. sim

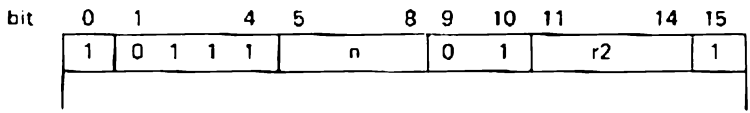
Syntax: [label] ␣ MSR ␣ n, r2

The contents of n consecutive registers (the first one is register A1) replace the contents of n consecutive memory locations. The first address of those locations is specified in r2. If r2 = the system stackpointer A15, the stack-pointer is updated by the contents of n registers.

Type	*Function*		*Syntax*
T3A	(A1)	→ (r2)	MSR n, r2
	(A2)	→ (r2) + 2	
	–		
	–		
	(An)	→ (r2) + 2n – 2	
T3B	(A1)	→ (A15)	MSR n, A15
	(A2)	→ (A15) – 2	
	–		
	–		
	(An)	→ (A15) – 2n + 2	
	(A15) – 2n	→ (A15)	

n = number of registers (1 through 15)

Condition register: Unchanged



Remark:
- An interrupt 'stack overflow' is generated when, for type T3B, the address reached by the pointer = </100. Bit 13 in PSW is set to 1.
- Restricted to system mode when n = 15 or r2 = A15.
- If 3B type, the A15 contents must be even (P851M).

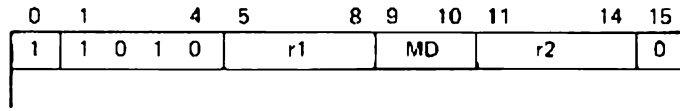
Syntax: [label] ␣ EL [*] ␣ r1, m [, r2]

The 16-bit contents of the effective memory address, specified in m and translated by the MMU, are loaded in register r1.

Type	*Function*	*MD*	*Syntax*
T4	(m) extended → r1	10	EL r1, m
T5	(m + (r2)) extended → r1	10	EL r1, m, r2
T6	((m)) extended → r1	11	EL* r1, m
T7	((m + (r2))) extended → r1	11	EL* r1, m, r2

Condition register:

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



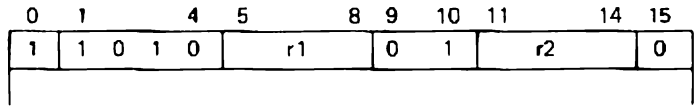
Remark:
This instruction may only be used in system mode.

Syntax: [label] ␣ ELR ␣ r1, r2

The 16-bit contents of the effective memory address pointed to in register r2, and translated by the MMU, are loaded in register r1.

Type	*Function*
T3	((r2)) extended → r1

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



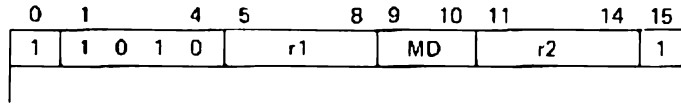
Remark:
This instruction may only be used in system mode.

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

The 16-bit contents of register r1 replace the contents of the effective memory address as translated by the MMU.

Type	*Function*	*MD*	*Syntax*
T4	(r1) → m, extended	10	ES r1, m
T5	(r1) → m + (r2), extended	10	ES r1, m, r2
T6	(r1) → (m), extended	11	ES* r1, m
T7	(r1) → (m + (r2)), extended	11	ES* r1, m, r2

Condition register: Unchanged



Remark:
This instruction may only be used in system mode.

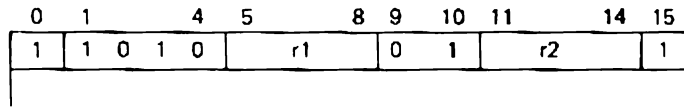
ESR	*Extended store/reg. (MMU option)*	ESR	P857M

Syntax: {label} ␣ ESR ␣ r1, r2

This instruction replaces the contents of the memory address specified in r2, and translated by the MMU, by the 16-bit contents of register r1.

Type	*Function*
T3	(r1) → (r2) extended

Condition register: Unchanged



Remark:
This instruction may only be used in system mode.

LDA *Load address* **LDA**

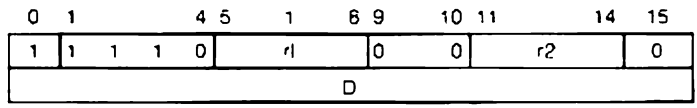
P853
P854
P858
P859

Syntax: [label] LDA r1,D,r2

This instruction loads the address specified in r2, incremented by the value D from the second instruction word, into the register specified by r1.

Type	*Function*
T1	(r2) + D -> r1

Condition register: Unchanged



Remark

- r1 must be ≠ 0
- restricted to system mode if r1 = A15

ADK
ADKL

Add constant

ADK
ADKL

P851M
P852M
P856M
P857M

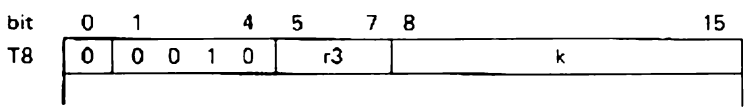
Syntax: [label] ␣ ADK ␣ r3, k T8
[label] ␣ ADKL ␣ r1, lk – T2

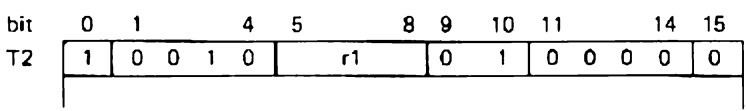
T8 The positive constant k is added to the contents of the register specified in r3. The result of the addition is placed in r3.

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

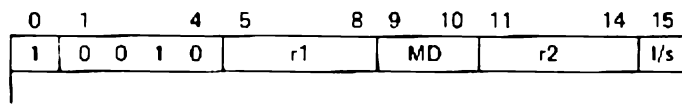
Syntax: [label] ␣ ADR [•] ␣ r1, r2
[label] ␣ ADRS ␣ r1, r2

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.

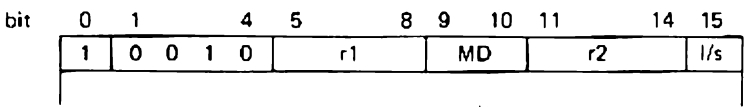
Syntax: [label] ␣ AD [S] [•] ␣ r1, m[, r2]

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.

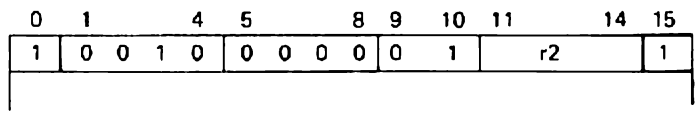
Syntax: [label] ␣ IMR ␣ r2

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

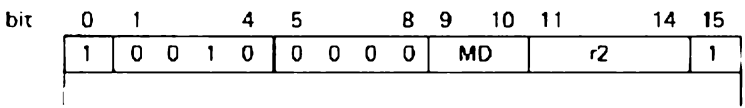
Syntax: [label] ␣ IM [*] ␣ m [, r2]

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

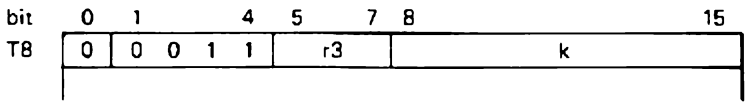
Syntax: [label] ␣ SUK␣ r3, k – T8
[label] ␣ SUKL ␣ r1, lk – T2

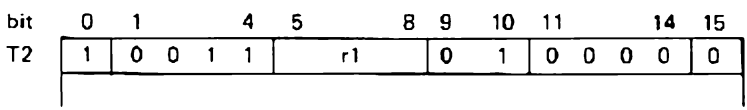
T8 The positive constant k is subtracted from the contents of the register specified in r3. The result is placed in r3.

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

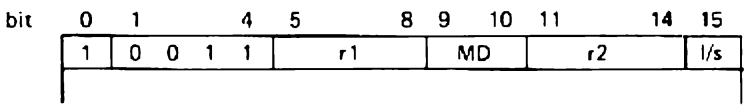
Syntax: [label] ␣ SUR [•] ␣ r1, r2
[label] ␣ SURS ␣ r1, r2

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

bit	0	1			4	5			8	9	10	11			14	15
	1	0	0	1	1	r1				MD		r2				l/s

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

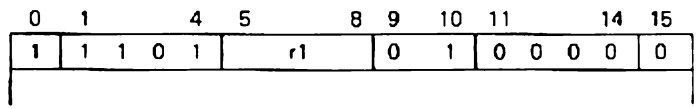
Syntax: [label] ␣ CWK ␣ r1, Ik

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) ↔ Ik → CR	CWK r1, Ik

Condition register:

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



Remark:
Restricted to system mode if r1 = A15.

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

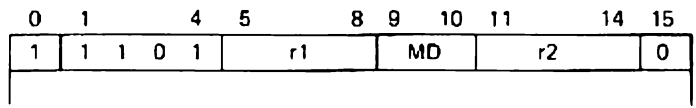
Syntax: [label] ␣ CWR [•] ␣ r1, r2

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.

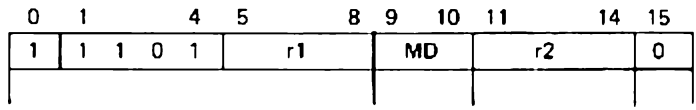
Syntax: [label] ␣ CW[•]␣ r1, m [, r2]

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

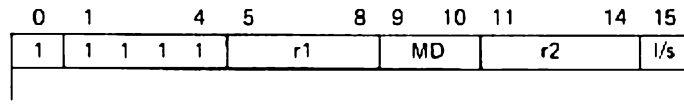
P851M P852M P856M P857M

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.

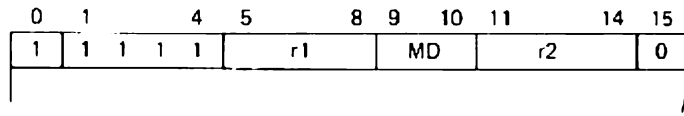
The logic complement of the contents of the 16-bit register specified by r2 (direct addressing) or the contents of the memory address indicated in the register specified by r2 replaces the contents of:

- (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	$\overline{(r2)} \rightarrow r1$	00	0	C1R r1, r2
T3	$\overline{((r2))} \rightarrow r1$	01	0	C1R• r1, r2
T3	$\overline{((r2))} \rightarrow (r2)$	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

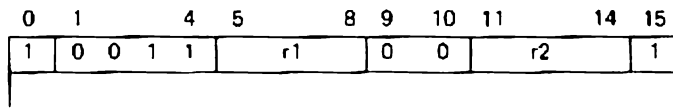
Twos complement.
Zero bits become one and vice versa, +1.

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).

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

bit	0	1			4	5				8	9	10	11			14	15
	1	0	0	1	1	0	0	0	0		0	1	r2				1

C2 *Twos complement* **C2**

P851M
P852M
P856M
P857M

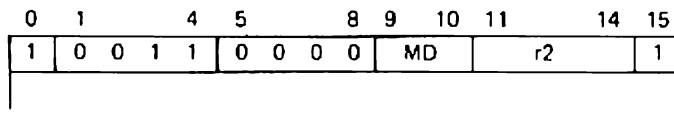
Syntax: [label] ␣ C2 [*] ␣ m [, r2]

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**

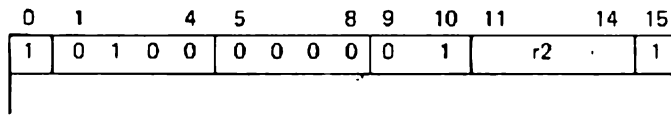
P851M
P852M
P856M
P857M

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



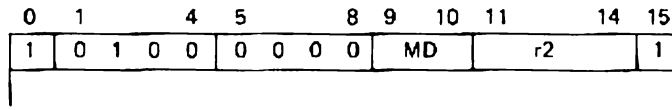
CM	*Clear memory*	CM	P851M P852M P856M P857M

Syntax: [label]␣ CM [*] ␣ m [, r2]

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

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

Condition register: Unchanged



MUK *Multiply with constant* **MUK** P851M P852M P856M P857M (softw. sim)

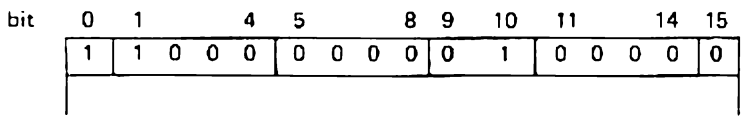
Syntax: [label] ␣ MUK ␣ lk

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

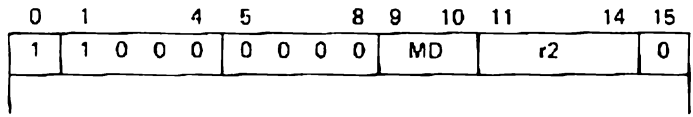
Syntax: [label] ␣ MUR [•] ␣ r2

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



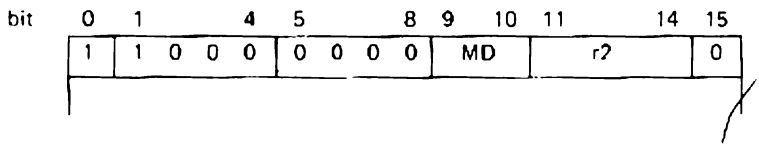
MU *Multiply* **MU** P851M P852M P856M P857M (softw. sim)

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)

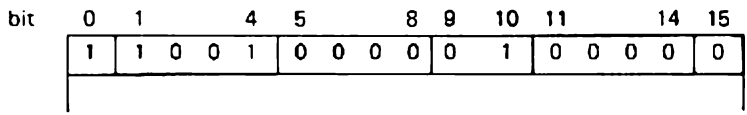
Syntax: [label]␣ DVK ␣ Ik

The contents of the registers A1, A2 are divided by the constant Ik. 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) / Ik	A2	A1

Condition register:

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



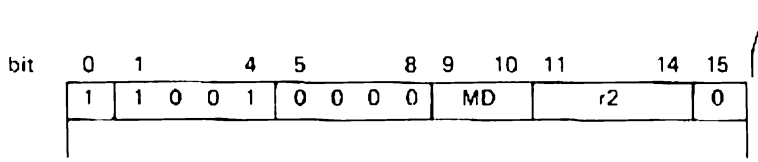
DVR *Divide register/register* **DVR** P851M P852M P856M P857M (softw. sim

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



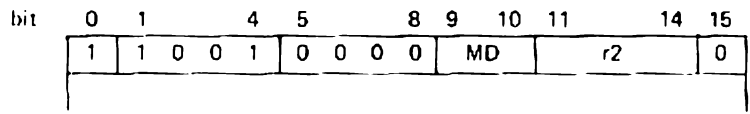
DV — *Divide* — DV — P851M P852M P856M P857M (softw. sim)

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)

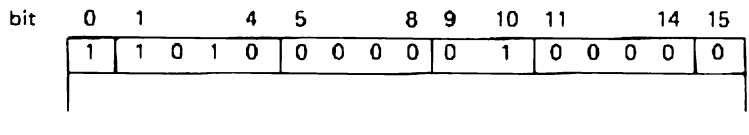
Syntax: [label] ␣ DAK ␣ lk_1 , lk_2

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.

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

bit	0	1			4	5			8	9	10	11			14	15
	1	1	0	1	0	0	0	0	0	MD		r2				0

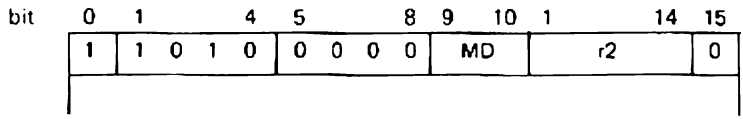
DA *Double add* **DA** P851M P852M P856M P857M (softw. sim)

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

bit	0	1			4	5			8	9	10	11			14	15
	1	1	0	1	1	0	0	0	0	0	1	0	0	0	0	0

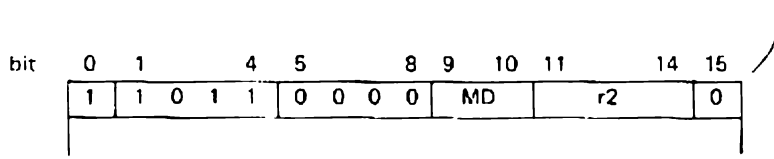
DSR *Double subtract reg./reg.* **DSR** P851M P852M P856M P857M (softw. sim)

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

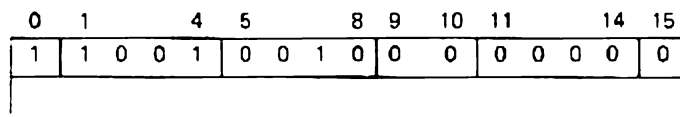
bit	0	1			4	5			8	9	10	11			14	15
	1	1	0	1	1	0	0	0	0	MD		r2				0

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



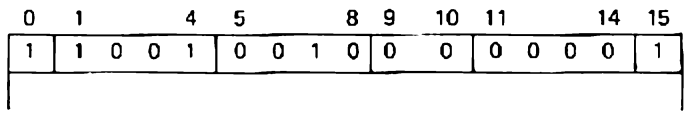
Syntax: [label]␣ FFX

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)



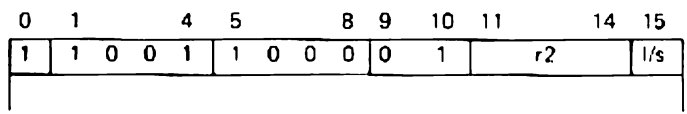
Syntax: [label]␣ FADR[S]␣ r2

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})



Syntax: [label]␣ FAD[S] [*]␣ m[, r2]

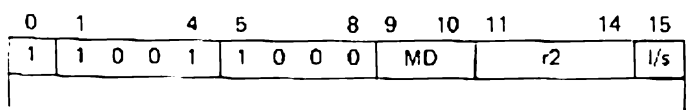
The floating point operand contained in the floating point accumulators FPA1, FPA2, FPA3 on the Floating Point Processor, is added to the floating point operand contained in three consecutive memory locations, the first one being indicated by the effective memory address. The sum is placed either in 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})



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 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	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})

bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	1	1	0	0	1	1	0	1	0	0	1	*r2*				l/s

Syntax: [label]␣ FSU[S] [*]␣ m[, r2]

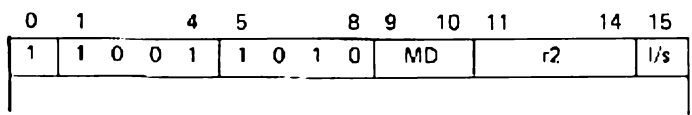
The floating point operand contained in three consecutive memory locations, the first of which is specified by the effective memory address, is subtracted from the floating point operand present in three accumulators FPA1, FPA2 and FPA3 on the Floating Point Processor. The result is placed either in FPA1, FPA2 and FPA3 or in three consecutive memory locations pointed to by the effective memory address, depending on the state of the l/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*	*l/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})



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})

bit	0	1			4	5			8	9	10	11			14	15
	1	1	0	0	1	1	1	0	0	0	1	r2				I/s

Syntax: [label]␣ FMU[S] [•]␣ m[, r2]

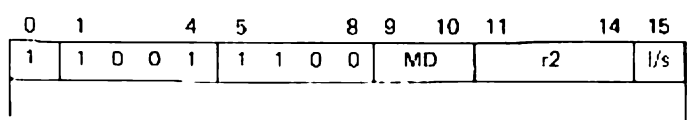
The floating point operand contained in the floating point processor accumulators FPA1, FPA2, FPA3, is multiplied by the floating point operand present in three consecutive memory locations, the first of which is indicated by the effective memory address. The result is placed either in FPA1, FPA2 and 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})



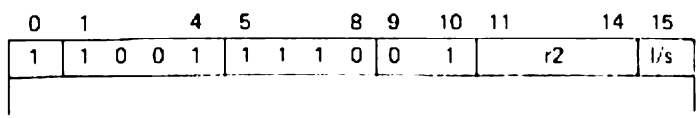
Syntax: [label]␣ FDVR[S]␣ r2

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 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	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]

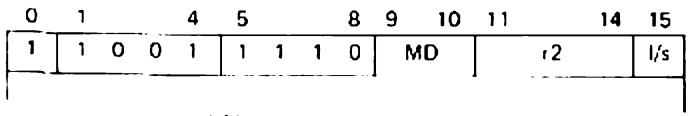
The floating point operand contained in the floating point processor accumulators FPA1, FPA2, FPA3, is divided by the floating point operand present in three consecutive memory locations, the first one being pointed at by the effective memory address. The result is placed either in FPA1,FPA2,FPA3 or 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^{14})
- arithmetic underflow (result exponent < -2^{14})
- Divisor = 0



ANK
ANKL

Logical AND with constant

ANK
ANKL

P851M
P852M
P856M
P857M

Syntax: [label]␣ ANK ␣r3, k – T8
[label]␣ ANKL ␣r1, lk – T2

Logical product

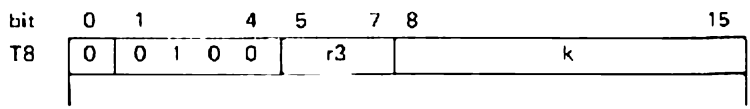
Bit in r3 or r1	Bit in k or lk	Logical product
0	0	0
0	1	0
1	0	0
1	1	1

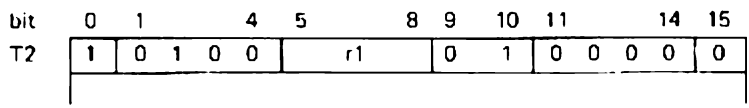
T8 The logical product of k and the contents of bits 8–15 of the register specified by r3 is placed in bits 8–15 of r3.
Bits 0–7 of this register are set to 0.

T2 The logical product of lk and the contents of the register specified by r1 is placed in r1.

Type	*Function*	*Syntax*
T8	$(r3)_{8\text{-}15} \wedge k \rightarrow r3_{8\text{-}15}$ $0 \rightarrow r3_{0\text{-}7}$	ANK r3, k
T2	$(r1) \wedge lk \rightarrow r1$	ANKL r1, lk

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





Remark:
- If T8, r3 must be ≠ 0. If T2, r1 must be ≠ 0.
- Restricted to system mode if r1 = A15.

ANR ANRS

Logical AND register/register

ANR ANRS

P851M P852M P856M P857M

Syntax: [label]␣ ANR [•]␣ r1, r2
[label]␣ ANRS ␣ r1, r2

Logical product

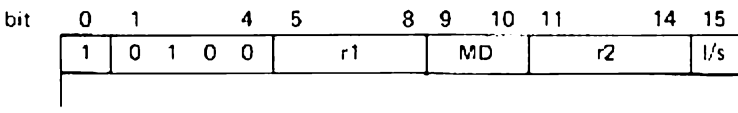
Bit in r1	Bit in 2nd operand	Logical product
0	0	0
0	1	0
1	0	0
1	1	1

The logical product of the contents of the register r1 and 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) is stored in:

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

Type	*Function*	*MD*	*I/s*	*Syntax*	
T1	(r1) ∧ (r2) → r1	00	0	ANR	r1, r2
T3	(r1) ∧ ((r2)) → r1	01	0	ANR*	r1, r2
T3	(r1) ∧ ((r2)) → (r2)	01	1	ANRS	r1, r2

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



Remark:
* If T1, then r1 must be ≠ 0. If T3, and I/s ≠ 0 then r1 must be ≠ 0.
* Restricted to system mode if r1 = A15.

AN ANS

Logical AND

AN ANS

P851M P852M P856M P857M

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

Logical product

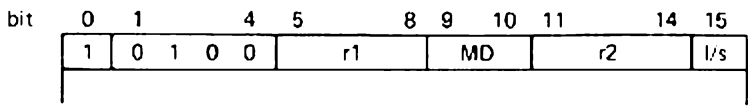
Bit in r1	Bit in 2nd operand	Logical product
0	0	0
0	1	0
1	0	0
1	1	1

The logical product of the contents of the effective memory address and the contents of the register specified by r1, is placed in this register, when the l/s indicator 0, or in the effective memory address, when l/s is 1.

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

Condition register:

CR = 0 if logical product = 0
1 if logical product > 0
2 if logical product < 0



Remark:
* If l/s = 0 then r1 must be ≠ 0.
* Restricted to system mode if r1 = A15.

ORK
ORKL

Logical OR with constant

ORK
ORKL

P851M
P852M
P856M
P857M

Syntax:

[label] ␣ ORK ␣ r3, k — T8
[label] ␣ ORKL ␣ r1, lk — T2

Logical union:

Bit in r3 or r1	Bit in k or lk	Logical union
0	0	0
0	1	1
1	0	1
1	1	1

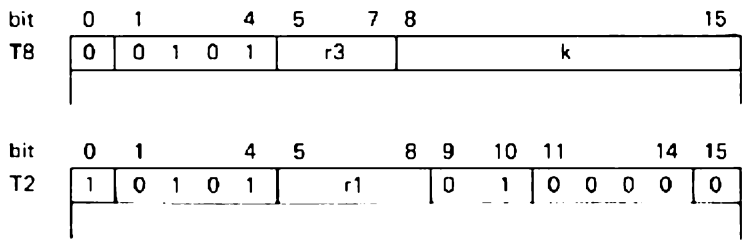
T8 A logical OR is performed on the contents of bits 8–15 of the register specified by r3 and the value of the constant k. The result is placed in bits 8–15 of the register specified by r3. Bits 0–7 of this register are set to zero.

T2 A logical OR is performed on the contents of the register specified by r1 and the value of the constant lk. The result of this operation is placed in the register specified by r1.

Type	*Function*			*Syntax*
T8	$(r3)_{8-15} \vee k \rightarrow r3_{8-15}$	$r3_{0-7}$	unchanged	ORK r3, k
T2	$(r1) \vee lk \rightarrow r1$			ORKL r1, lk

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0



Remark:
* If l/s = 0 then r1 must be ≠ 0.
* Restricted to system mode if r1 = A15.

Syntax: [label]␣ ORR [*]␣ r1, r2
[label]␣ ORRS ␣ r1, r2

Logical union:

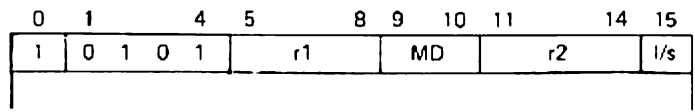
Bit in r1	Bit in 2nd operand	Logical union
0	0	0
0	1	1
1	0	1
1	1	1

The logical OR of the contents of the 16-bit register specified by r1 and the contents of the 16-bit register specified by r2 (direct addressing) or the contents of the memory address indicated by the register specified by r2 (indirect instruction) is placed:

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

Type	*Function*	*MD*	*I/s*	*Syntax*	
T1	(r1) ∨ (r2) → r1	00	0	ORR	r1, r2
T3	(r1) ∨ ((r2)) → r1	01	0	ORR*	r1, r2
T3	(r1) ∨ ((r2)) → (r2)	01	1	ORRS	r1, r2

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



Remark:
* If I/s = 0 then r1 must be ≠ 0.
* Restricted to system mode if r1 = A15.

OR ORS

Logical OR

OR ORS

P851M P852M P856M P857M

Syntax: [label]␣ OR[S] [•]␣ r1, m[, r2]

Logical union:

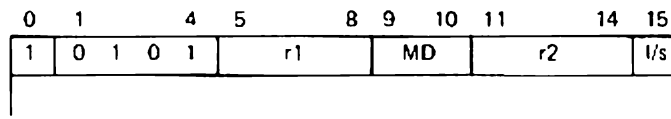
Bit in r1	Bit in 2nd operand	Logical union
0	0	0
0	1	1
1	0	1
1	1	1

The logical OR of the contents of the effective memory address and the contents of the register specified by r1 is placed either in the r1 register, when I/s bit = 0, or in the effective memory address, when I/s bit = 1.

Type	*Function*	*MD*	*I/s*	*Syntax*
T4	(r1) ∨ (m) → r1	10	0	OR r1, m
T4	(r1) ∨ (m) → m	10	1	ORS r1, m
T5	(r1) ∨ (m + (r2)) → r1	10	0	OR r1, m, r2
T5	(r1) ∨ (m + (r2)) → m + (r2)	10	1	ORS r1, m, r2
T6	(r1) ∨ ((m)) → r1	11	0	OR* r1, m
T6	(r1) ∨ ((m)) → (m)	11	1	ORS* r1, m
T7	(r1) ∨ ((m + (r2))) → r1	11	0	OR* r1, m, r2
T7	(r1) ∨ ((m + (r2))) → (m + (r2))	11	1	ORS* r1, m, r2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0



Remark:
* r1 must be ≠ 0.
* Restricted to system mode if r1 = A15.

Exclusive OR with constant

Syntax:

[label] ␣ XRK ␣ r3, k – T8
[label] ␣ XRKL ␣ r1, lk – T2

Exclusive OR:

Bit in r3 or r1	Bit in k or lk	Exclusive OR
0	0	0
0	1	1
1	0	1
1	1	0

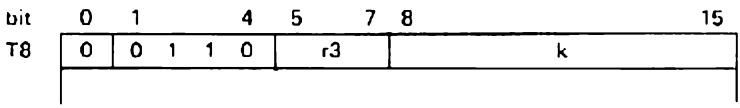
T8 The exclusive OR on the contents of bits 8–15 of the register specified by r3 and the value of k is placed in the register specified by r3.
Bits 0–7 of this register remain unchanged.

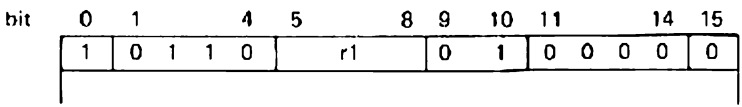
T2 The exclusive OR on the contents of the register specified by r1 and lk is placed in the register specified by r1.

Type	*Function*			*Syntax*	
T8	$(r3)_{8-15} \forall k \rightarrow r3_{8-15}$	$r3_{0-7}$	unchanged	XRK	r3, k
T2	$(r1) \forall lk \rightarrow r1$			XRK	r1, lk

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0





Remark:
* r1 and r3 must be ≠ 0.
* Restricted to system mode if r1 = A15.

XR XRS

Exclusive OR

XR XRS

P851M P852M P856M P857M

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

Exclusive OR:

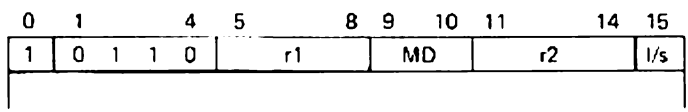
Bit in r1	Bit in 2nd operand	Exclusive OR
0	0	0
0	1	1
1	0	1
1	1	0

The exclusive OR of the contents of the effective memory address and the contents of the register specified by r1 is placed either in the register specified by r1, when the l/s bit = 0 or in the effective memory address, when l/s = 1.

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

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0



Remark:
* r1 must be ≠ 0.
* Restricted to system mode if r1 = A15.

XRR
XRRS

Exclusive OR register/register

XRR
XRRS

P851M
P852M
P856M
P857M

Syntax: [label]␣ XRR [•]␣ r1, r2
[label]␣ XRRS ␣ r1, r2

Exclusive OR:

Bit in r1	Bit in 2nd operand	Exclusive OR
0	0	0
0	1	1
1	0	1
1	1	0

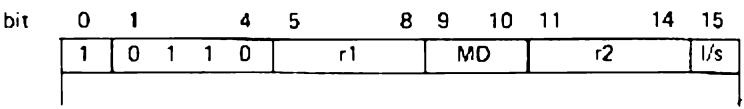
The exclusive OR of the contents of the 16-bit register specified by r1 and the contents of the 16-bit register specified by r2 (direct addressing) or the contents of the memory address indicated in the register specified by r2 (indirect addressing) are placed as follows:

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

Type	*Function*	*MD*	*I/s*	*Syntax*
T1	(r1) ∀ (r2) → r1	00	0	XRR r1, r2
T3	(r1) ∀ ((r2)) → r1	01	0	XRR* r1, r2
T3	(r1) ∀ ((r2)) → (r2)	01	1	XRRS r1, r2

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0 .



Remark:
• r1 must be ≠ 0.
• Restricted to system mode if r1 = A15.

TM *Test mask* **TM** P851M P852M P856M P857M

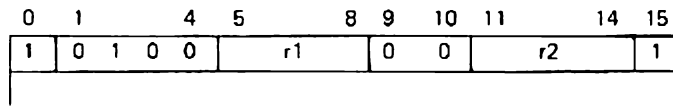
Syntax: [label]␣ TM ␣ r1, r2

The logical product (AND) of the contents of the register specified by r1 and the contents of the register specified by r2 is compared to zero.
The result of the comparison is stored in the condition register.
The contents of the register specified by r1 and r2 remain unchanged.

Type	*Function*
T1	[(r1) ∧ (r2)] ↔ 0 → CR

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0



Remark:
- r1 must be ≠ 0.
- Restricted to system mode if r1 = A15.

TNM *Test not mask* **TNM** P851M P852M P856M P857M

Syntax: [label] TNM r1, r2

The exclusive OR of the contents of the register specified by r1 and the contents of the register specified by r2 is compared with zero. The result of the comparison is stored in the condition register.
The initial contents of the register specified by r1 and the register specified by r2 remain unchanged.

Type	*Function*
T1	[(r1) ∀ (r2)] ↔ 0 → CR

Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0

bit	0	1			4	5			8	9	10	11			14	15
	1	0	1	1	0	r1				0	0	r2				1

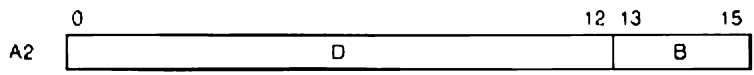
Remark:
* r1 must be ≠ 0.
* Restricted to system mode if r1 = A15.

Syntax: [label] TSB[*] m[,r2]

This instruction tests a bit in a bitstring, sets the condition register to the value of that bit, and sets the bit to 1.
The address of the first character of the bitstring is the instruction operand, found as follows:

Type	*address*
T4	m
T5	m + (r2)
T6	(m)
T7	(m + (r2))

The bit position in the string must be specified in register A2;
in addressing the operand it is used as shown below:



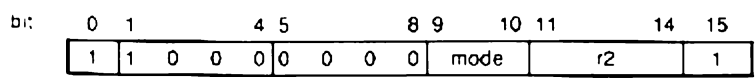
The bit displacement $A2_{0-15}$ is split up in the character displacement D and the bit number B.

The function of the instruction is:

Type	*Function*	*Mode*	*Syntax*	
T4	$(m + D)_B \rightarrow CR$ $1 \rightarrow (m + D)_B$	10	TSB	m
T5	$(m + (r2) + D)_B \rightarrow CR$ $1 \rightarrow (m + (r2) + D)_B$	10	TSB	m,r2
T6	$((m) + D)_B \rightarrow CR$ $1 \rightarrow ((m) + D)_B$	11	TSB*	m
T7	$((m + (r2)) + D)_B \rightarrow CR$ $1 \rightarrow ((m + (r2)) + D)_B$	11	TSB*	m,r2

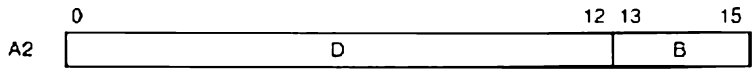
Condition register

CR = 0 if tested bit was 0
CR = 1 if tested bit was 1



Syntax: [label] TSBR r2

This instruction tests a bit in a bitstring, sets the condition register to the value of that bit, and sets the bit to 1.
The address of the first character of the string is contained in the register specified by r2.
The bit position in the string must be specified in register A2; in addressing the operand it is used as shown below:

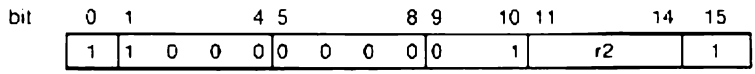


The bit displacement $A2_{0-15}$ is split up in the character displacement D and the bit number B

The function of the instruction is:

Type	*Function*
T3	$((r2) + D)_B \rightarrow CR$ $1 \rightarrow ((r2) + D)_B$

Condition register:
CR = 0 if the tested bit was 0
CR = 1 if the tested bit was 1

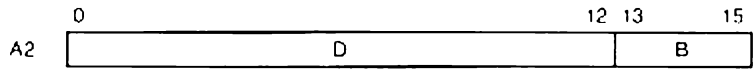


Syntax [label] TRB[*] m[r2]

This instruction tests a bit in a bitstring, sets the condition register to the value of that bit, and resets the bit to 0
The address of the first character of the bitstring is the instruction operand, found as follows:

Type	*Address*
T4	m
T5	m + (r2)
T6	(m)
T7	(m + (r2))

The bit position in the string must be specified in register A2; in addressing the operand it is used as shown below.



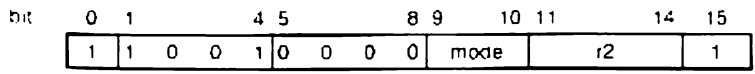
The bit displacement $A2_{0-15}$ is split up in the character displacement D and the bit number B.

The function of the instruction is:

Type	*Function*	*Mode*	*Syntax*	
T4	$(m + D)_B \rightarrow CR$ $0 \rightarrow (m + D)_B$	10	TRB	m
T5	$(m + (r2) + D)_B \rightarrow CR$ $0 \rightarrow (m + (r2) + D)_B$	10	TRB	m,r2
T6	$((m) + D)_B \rightarrow CR$ $0 \rightarrow ((m) + D)_B$	11	TRB*	m
T7	$((m + (r2)) + D)_B \rightarrow CR$ $0 \rightarrow ((m + (r2)) + D)_B$	11	TRB*	m,r2

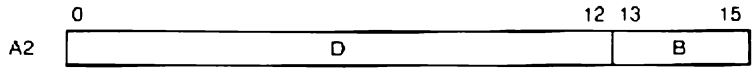
Condition register

CR = 0 if tested bit was 0
CR = 1 if tested bit was 1



Syntax. [label] TRBR r2

This instruction tests a bit in a bitstring, sets the condition register to the value of that bit, and resets the bit to 0.
The address of the first character of the string is contained in the register specified by r2
The bit position in the string must be specified in register A2; in addressing the operand it is used as shown below:

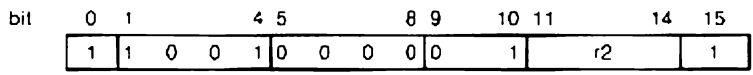


The bit displacement $A2_{0-15}$ is split up in the character displacement D and the bit number B.

The function of the instruction is:

Type	*Function*
T3	$((r2) + D)_B \rightarrow CR$ $0 \rightarrow ((r2) + D)_B$

Condition register:
CR = 0 if the tested bit was 0
CR = 1 if the tested bit was 1

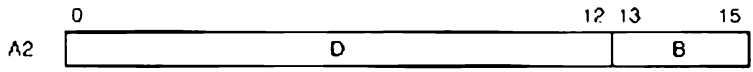


Syntax. [label] TB[*] m[.r2]

This instruction tests a bit in a bitstring, and sets the condition register to the value of that bit.
The address of the first character of the bitstring is the instruction operand, found as follows.

Type	*Address*
T4	m
T5	m + (r2)
T6	(m)
T7	(m + (r2))

The bit position in the string must be specified in register A2; in addressing the operand it is used as shown below:



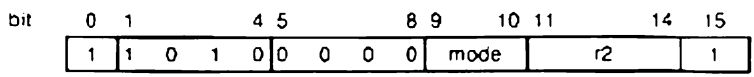
The bit displacement $A2_{0-15}$ is split in the character displacement D and the bit number B.

The function of the instruction is:

Type	*Function*	*Mode*	*Syntax*	
T4	$(m + D)_B \rightarrow CR$	10	TB	m
T5	$(m + (r2) + D)_B \rightarrow CR$	10	TB	m.r2
T6	$((m) + D)_B \rightarrow CR$	11	TB*	m
T7	$((m + (r2)) + D)_B \rightarrow CR$	11	TB*	m.r2

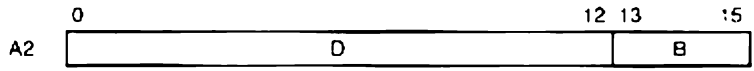
Condition register

CR = 0 if tested bit was 0
CR = 1 if tested bit was 1



Syntax: [label] TBR r2

This instruction tests a bit in a bitstring, sets the condition register to the value of that bit.
The address of the first character of the string is contained in the register specified by r2
The bit position in the string must be specified in register A2; in addressing the operand it is used as shown below:



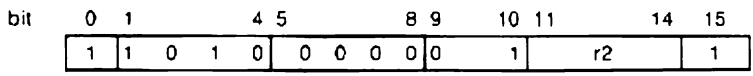
The bit displacement $A2_{0-15}$ is split up in the character displacement D and the bit number B

The function of the instruction is:

Type	*Function*
T3	$((r2) + D)_B \rightarrow CR$

Condition register:

CR = 0 if the tested bit was 0
CR = 1 if the tested bit was 1



ECR *Exchange characters register/register* **ECR**

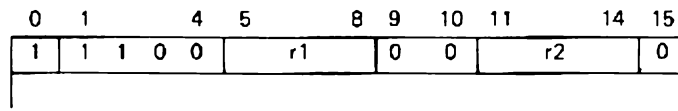
P851M
P852M
P856M
P857M

Syntax: [label] ␣ ECR ␣ r1, r2

The left and right-hand characters contained in the register specified by r2 are exchanged and then placed in the register specified by r1.
The old contents of the register specified by r2 are not changed.

Type	*Function*
T1	$(r2)_l \rightarrow r1_r$ and $(r2)_r \rightarrow r1_l$

Condition register: Unchanged



Remark:
- r1 must be ≠ 0.
- Restricted to system mode if r1 = A15.

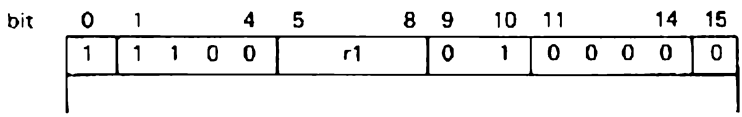
LCK *Load character with constant* **LCK** P851M P852M P856M P857M

Syntax: [label] ␣ LCK ␣ r1, lk

The left-hand character (bits 0–7) of the constant lk is copied to bits 8–15 (right-hand character) of the register specified by r1.
Bits 0–7 of r1 remain unchanged.

Type	*Function*
T2	$lk_l \rightarrow r1_r$

Condition register: Unchanged



Remark:
- r1 must be ≠ 0.
- Restricted to system mode if r1 = A15.

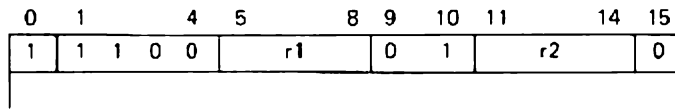
LCR	*Load character/register*	LCR	P851M P852M P856M P857M

Syntax: [label]␣ LCR ␣ r1, r2

The right-hand (odd address) 8-bit contents or the left-hand (even address) 8-bit contents of the effective memory addresses, specified in r2, substitute the least significant 8 bits of the register specified by r1.
Bits 0–7 of r1 remain unchanged.

Type	*Function*
T3	$((r2))_{l/r} \rightarrow r1_r$

Condition register: Unchanged



Remark:
- r1 must be ≠ 0.
- Restricted to system mode if r1 = A15.

LC

Load character

LC

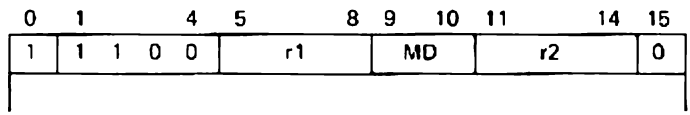
P851M
P852M
P856M
P857M

Syntax: [label] ␣ LC[•] ␣ r1, m[, r2]

This instruction allows to transfer the right-hand character of the contents of the effective memory address (odd address) or the left-hand character (even address) to bits 8—15 of the register specified by r1.
Bits 0—7 of r1 remain unchanged.

Type	*Function*		*MD*	*Syntax*	
T4	$(m)_{l/r}$	$\rightarrow r1_r$	10	LC	r1, m
T5	$(m + (r2))_{l/r}$	$\rightarrow r1_r$	10	LC	r1, m, r2
T6	$((m))_{l/r}$	$\rightarrow r1_r$	11	LC*	r1, m
T7	$((m + (r2)))_{l/r}$	$\rightarrow r1_r$	11	LC*	r1, m, r2

Condition register: Unchanged



Remark:
* r1 must be ≠ 0.
* Restricted to system mode if r1 = A15.

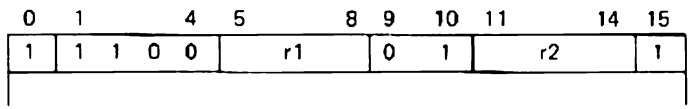
SCR	*Store character/register*	SCR	P851M P852M P856M P857M

Syntax: [label] ␣ SCR ␣ r1, r2

The least significant bits of the register specified by r1 replace the right-hand (odd address) or the left-hand (even address) 8 bit contents of the effective memory address indicated by r2.

Type	*Function*
T3	$(r1)_r \rightarrow (r2)_{r/l}$

Condition register: Unchanged



Remark:
- r1 must be ≠ 0.
- Restricted to system mode if r1 = A15.

SC *Store character* **SC**

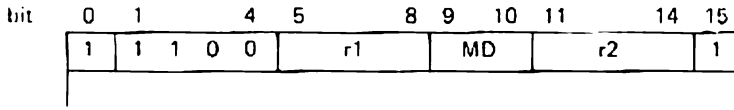
P851M
P852M
P856M
P857M

Syntax: [label]₁ , SC[*]₂ r1, m[, r2]

The least significant 8 bits of the register specified by r1, when address is odd, replace the right-hand 8 bits of the contents of the effective memory address or the left-hand 8 bits, when the address is even.
The unaffected half of the address remains unchanged.

Type	*Function*	*MD*	*Syntax*
T4	$(r1)_r \rightarrow$ m,r/l	10	SC r1, m
T5	$(r1)_r \rightarrow$ m + (r2) l/r	10	SC r1, m, r2
T6	$(r1)_r \leftarrow$ (m) r/l	11	SC* r1, m
T7	$(r1)_r \leftarrow$ (m + (r2)) l/r	11	SC* r1, m, r2

Condition register: Unchanged



Remark:
* r1 must be ≠ 0.
* Restricted to system mode if r1 = A15.

CCK *Compare character with constant* **CCK**

P851M
P852M
P856M
P857M

Syntax: [label]␣ CCK ␣ r1, Ik

Bits 8–15 (the right-hand character) of the register specified by r1 are compared with bits 0–7 (the left-hand character) of the constant Ik. The most significant bit of a character is not a sign bit. The result of the comparison is stored in the condition register.

Type	*Function*
T2	$(r1)_r \leftrightarrow Ik_l \rightarrow CR$

Condition register:

CR = 0 if $(r1)_r = Ik_l$
1 if $(r1)_r > Ik_l$
2 if $(r1)_r < Ik_l$

bit	0	1			4	5			8	9	10	11			14	15
	1	1	1	0	1	r1				0	1	0	0	0	0	1

Remark:
- r1 must be ≠ 0.
- Restricted to system mode if r1 = A15.

Compare character/register

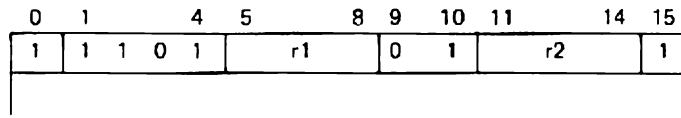
Syntax: [label] ␣ CCR ␣ r1, r2

The 8 least significant bits of the register specified by r1 are compared with the right-hand (if odd address) or left-hand (if even address) 8 bits of the contents of the effective memory address indicated in r2.
The result of the comparison is stored in the condition register.
The most significant bit of a character is considered not to be a sign bit.

Type	*Function*
T3	$(r1)_r \leftrightarrow ((r2))_{l/r} \rightarrow CR$

Condition register:

CR ← 0 if $(r1)_r = ((r2))_{l/r}$
1 if $(r1)_r > ((r2))_{l/r}$
2 if $(r1)_r < ((r2))_{l/r}$



Remark:
- r1 must be ≠ 0.
- Restricted to system mode if r1 = A15.

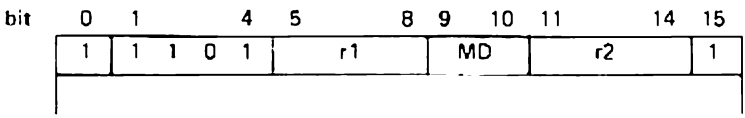
Syntax: [label]␣ CC [•]␣ r1, m[, r2]

The 8 least significant bits of the register specified by r1 are compared with the right-hand character of the contents of the effective memory address (odd address) or with the left-hand character of the contents of the effective memory address (even address).
The result of the operation is stored in the condition register.
The most significant bit of a character is considered not to be a sign bit.

Type	*Function*	*MD*	*Syntax*
T4	$(r1)_r \leftrightarrow (\ m\ )_{l/r}$ → CR	10	CC r1, m
T5	$(r1)_r \leftrightarrow (\ m\ + (r2))_{l/r}$ → CR	10	CC r1, m, r2
T6	$(r1)_r \leftrightarrow ((m))_{l/r}$ → CR	11	CC• r1, m
T7	$(r1)_r \leftrightarrow ((m\ + (r2)))_{l/r}$ → CR	11	CC• r1, m, r2

Condition register:

CR = 0 if $(r1)_r$ = (2nd operand)$_{l/r}$
1 if $(r1)_r$ > (2nd operand)$_{l/r}$
2 if $(r1)_r$ < (2nd operand)$_{l/r}$



Remark:
• r1 must be ≠ 0.
• Restricted to system mode if r1 = A15.

The branch instructions AB, ABL, ABR, ABI, RB and RF branch to an address or the contents of an address or register when a certain condition is fulfilled. If that condition does not arise the program determines the next instruction to be executed.
The condition is given by a number from 1 through 7 or by one or two letters.
The following table gives a survey:

Condition Notation

Cond. reg. contents	(cnd)			
	GENERAL	ARITHM.	COMPARE	I/O
0	(0)	(Z) Zero	(E) Equal	(A) Accepted
1	(1)	(P) Pos.	(G) Greater	(R) Refused
2	(2)	(N) Neg.	(L) Less	–
3	(3)	(O) Overfl.	–	(U) Unknown
NOT – Condition				
≠0	(4)	(NZ) Not Zero	(NE) Not Equal	(NA) Not Accepted
≠1	(5)	(NP) Not Pos.	(NG) Not Greater	(NR) Not Refused
≠2	(6)	(NN) Not Neg.	(NL) Not Less	–
n.s.	(7)	Unconditional		

Note:
The instruction counter P always points to the next instruction to be executed. Wherever in the description the notation (P) + 2 (or 4) appears, the hardware function is meant.
When the following program must be assembled calculate the displacement in locations as follows:

```
BEGIN  EQU   *
       HLT
       LDK   A1,/000A
       SUK   A1,2
       RF(Z) *+4
       RB    *-4
       ABL   *-8
       END   START
```

where *+4 refers to ABL
*-4 refers to SUK
*-8 refers to LDK

When the same program is to be put in memory with the toggle switches the value for RF(Z) *+4 is 5002 and not 5004 as the P register is already pointing to the next instruction.
The value for RB *-4 must be 5F06 and not 5F04, as the P-register is already pointing to the next instruction.
The address in the ABL instruction must be the relative address pointing to LDK.

The values put in memory for the program listed above must be:

```
207F    START   HLT
010A            LDK     A1,/000A
1902            SUK     A1,2
5002            RF(Z)   *+4
5F06            RB      *-4
8F20            ABL     *-8
0002
                END     START
```

AB
ABL

Absolute conditional branch

AB
ABL

P851M
P852M
P856M
P857M

Syntax: [label]␣ AB [(cnd)]␣ k — T8
[label]␣ ABL[(cnd)]␣ lk — T2

This instruction means that the next instruction to be executed is found either at the address specified by the constant (k indicating one of the first 256 addresses of the memory, lk being specified in the word following the instruction) or in normal sequence, depending on the (cnd) and the contents of the condition register.
If (cnd) is equal to (7) the next instruction is at the effective memory address. Note that if (cnd) is omitted, the default value is (7).
The least significant bit in either constant, is always zero (word addressing).
See also table and note on page 6.0.1.

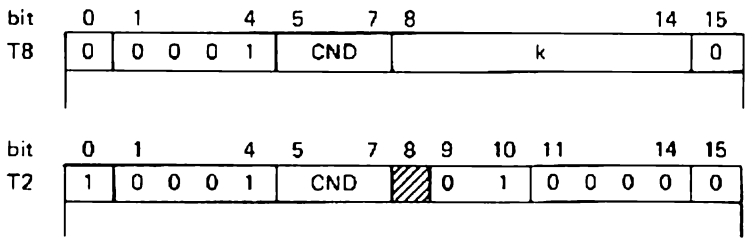
Effective branch:

Type	*Function*
T8	k → P
T2	lk → P

No branch:

Type	*Function*
T8	(P) + 2 → P
T2	(P) + 4 → P

Condition register: Unchanged



ABR *Absolute conditional branch to register* **ABR**

P851M
P852M
P856M
P857M

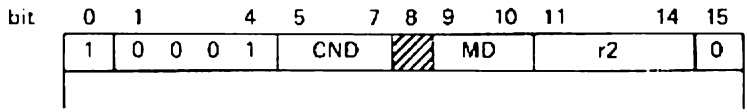
Syntax: [label]␣ ABR [[cnd)] [•]␣ r2

This instruction indicates that the address of next instruction to be executed is found either in the register specified by r2 or at the memory address indicated by the register or in normal sequence depending on (cnd) and the contents of the condition register.
If (cnd) = (7), the next instruction is at the effective memory address (unconditional branch).
If (cnd) is omitted, the default value is (7).
See also table and note on page 6.0.1.

Effective branch:	*Type*	*Function*	*MD*	*I/s*	*Syntax*
	T1	(r2) → P	00	n.s.	ABR(cnd) r2
	T3	((r2)) → P	01	0	ABR(cnd)* r2

No branch:	*Type*	*Function*	*MD*	*I/s*
	T1	(P) + 2 → P	00	n.s.
	T3	(P) + 2 → P	01	0

Condition register: Unchanged



Syntax: [label]␣ ABI[(cnd)] [•]␣ m[, r2]

The address of the next instruction to be executed is found either at the effective memory address or in the next instruction, depending on (cnd) and the contents of the condition register.
If (cnd) = (7) (see below) the instruction branched to is always at the effective memory address (unconditional branch).
In all other cases the program must first fulfil a condition before the branch takes place. If (cnd) is omitted, the default value is (7).
See also table and note on page 6.0.1.

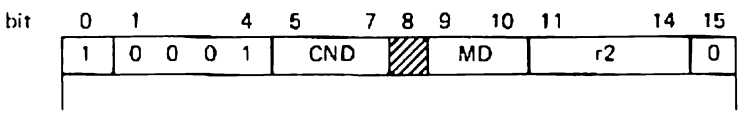
Effective branch:

Type	*Function*	*MD*	*Syntax*	
T4	(m) → P	10	ABI(cnd)	m
T5	(m + (r2)) → P	10	ABI(cnd)	m, r2
T6	((m)) → P	11	ABI(cnd)*	m
T7	((m + (r2))) → P	11	ABI(cnd)*	m, r2

No branch:

Type	*Function*	*MD*
T4	(P) + 4 → P	10
T5	(P) + 4 → P	10
T6	(P) + 4 → P	11
T7	(P) + 4 → P	11

Condition register: Unchanged



Syntax: [label]␣ RF [(cnd)]␣ m

This instruction indicates that the next instruction to be executed is found either at the effective memory address or in normal sequence, depending on (cnd) and the contents of the condition register. If (cnd) = (7) the next instruction can be found at the effective memory address (unconditional relative branch).
If (cnd) is omitted, the default value of (7) is assumed.

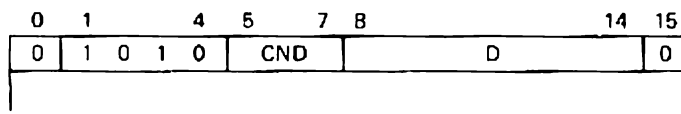
The assembler calculates from the effective memory address, a displacement D relative (forwards) to the current value of the instruction counter (P). This value is stored in bits 8–15 of the instruction as a positive number. Thus its maximum is 255. In programming terms, this means that this instruction can only be used to branch by ≤ 128 words.

See also table and note on page 6.0.1.

Type	*Function*	
T8	(P) + 2 + D → P	(branch effective)
T8	(P) + 2 → P	(no branch)

Example:
```
RF(Z)  END
RF(3)  *+12
```

Condition register: Unchanged



RB	*Relative backwards conditional branch*	RB	P851M P852M P856M P857M

Syntax: [label]␣ RB[(cnd)]␣ m

This instruction means that the next instruction to be executed is found either at the effective memory address or in normal sequence, depending on (cnd) and the contents of the condition register. If (cnd) = (7) the next instruction to be executed is found at the effective memory address. If (cnd) is omitted, the defaut value of (7) is assumed.

The assembler calculates from the effective memory address, a displacement D relative (backwards) to the current value of the instruction counter (P). This value is stored in bits 8–15 of the instruction as a positive number. Thus its maximum is 255. In programming terms, this means that this instruction can only be used to branch backwards by ≤ 128 words.

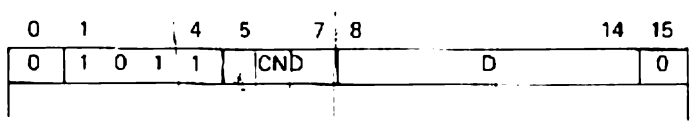
It should be noted that
␣ RB (cnd) ␣*
is equivalent to branch to itself and causes a continuous loop.

See also table and note on page 6.0.1.

Type	*Function*	
T8	(P) + 2 – D → P	(branch effective)
T8	(P) + 2 → P	(no branch)

Example:
RB(4) LABEL
RB(NE) *–2

Condition register: Unchanged



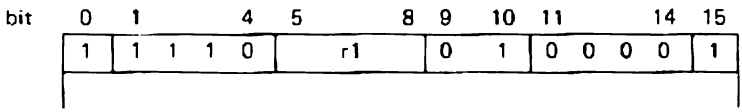
CF	*Call function*	CF	P851M P852M P856M P857M

Syntax: [label]␣␣ CF ␣ r1, lk

This instruction provides a link to a subroutine by storing successively the contents of the P-register and the program status word (PSW) in a memory stack. The PSW contains, amongst other things, the priority level and condition register. The stack pointer is held in the register specified by r1 and is automatically updated. Then a branch is made to the address specified by lk.
The subroutine must be terminated by an RTN instruction to branch back to the main program.

Type	*Function*
T2	(P) → (r1),(r1) − 2 → r1
	(PSW) → (r1),(r1) − 2 → r1
	lk → P

Condition register: Unchanged. Its contents shows the result of a previous operation and is stored in the memory stack for use on return from the subroutine.



Remark:
An interrupt 'stack overflow' is generated when r1 = A15 and the word address reached by the pointer = </100. Bit 13 is set in PSW.

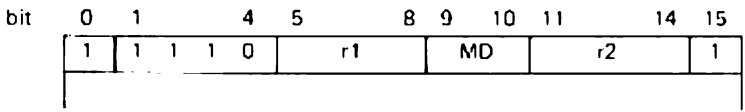
- r1 must be ≠ 0.
- Restricted to system mode if r1 = A15.
- The system stack and user stack are both built towards the lower addresses.
  P is stored first and next PSW.

Syntax: [label]␣ CFR[•]␣ r1, r2

This instruction provides a link to a subroutine by storing successively the contents of the P-register, which points to the next instruction of the main program, and the contents of the program status word (PSW) in a memory stack. The PSW contains, amongst other things the priority level and the condition register. The stack pointer held in the register specified by r1 is automatically updated by decreasing the stack pointer by 2, as the stack pointer is filled from the higher address towards the lower address.
Next a branch is made to the effective memory address specified by the contents of a register specified by r2.
The subroutine must be terminated by an RTN instruction to branch back to the main program.

Type	*Function*	*MD*	*Syntax*
T1,T3	(P) → (r1),(r1) − 2 → r1 (PSW) → (r1),(r1) 2 → r1		
	then:		
T1	(r2) → P	00	CFR r1, r2
T3	((r2)) → P	01	CFR* r1, r2

Condition register: Unchanged. Its contents shows the result of a previous operation and is stored in the memory stack for use on return from the subroutine.



Remark:
- An interrupt 'stack overflow' is generated when r1 − A15 and the word address reached by the pointer − < /100. Bit 13 in the PSW is set to 1.
- r1 must be ≠ 0.
- Restricted to system mode if r1 − A15.

Syntax: [label]␣ CFI[*]␣ r1, m[, r2]

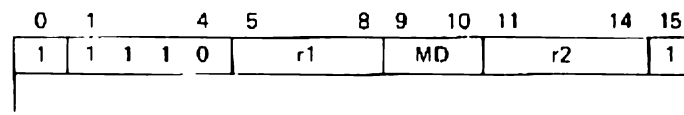
The instruction provides a link to a subroutine by storing successively the contents of the P-register, which points to the next instruction of the main program, and the contents of the program status word (PSW) in a memory stack. The PSW contains, amongst other things, the priority level and condition register. The stack pointer held in the register specified by r1 is automatically updated by decreasing the stack pointer by 2, as the stack pointer is filled from the higher address towards the lower address.
Next a branch is made to the contents of the effective memory address, i.e. the subroutine which has to be executed.
The subroutine must be terminated by an RTN instruction to branch back to the main program.

Type	*Function*		*MD*	*Syntax*
T4, T5	(P) → (r1)	(r1) − 2 → r1	n.a.	
T6, T7	(PSW) → (r1)	(r1) − 2 → r1	n.a.	

then:

T4	(m) → P	10	CFI r1, m
T5	(m + (r2)) → P	10	CFI r1, m, r2
T6	((m)) → P	11	CFI* r1, m
T7	((m + (r2))) → P	11	CFI* r1, m, r2

Condition register: Unchanged. Its contents shows the result of a previous operation and is stored in the memory stack for use on return from subprogram.



Remark:
- An interrupt 'stack overflow' is generated when r1 = A15 and the word address reached by the pointer = </100. Bit 13 of the PSW is set to 1.
- r1 must be ≠ 0.
- Restricted to system mode if r1 = A15.

RTN *Return from function (A1 . . . A14)* **RTN**

P851M
P852M
P856M
P857M

Syntax: [label]␣ RTN ␣ r2

This instruction allows the return from a subroutine to the main program. It must be the last instruction of such a routine. The instruction reloads the P-register and CR-register which have previously been loaded into a memory stack by a Call Function instruction.

Type	*Function*	
T3	(r2) + 2	→ r2
	$((r2))_{0-6}$	→ PLR
	$((r2))_{6-7}$	→ CR
	(r2) + 2	→ r2
	((r2))	→ P

Condition register: Reloaded from stack, bits 6 and 7 of the PSW → CR.

bit	0	1			4	5			8	9	10	11			14	15
	1	1	1	1	0	0	0	0	0	0	1	r2				0

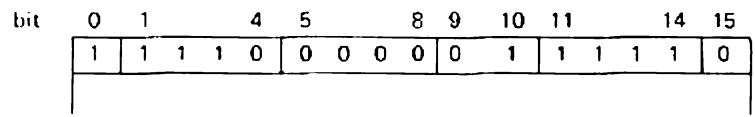
Remark:
r2 must be ≠ 0.

RTN	*Return from function (A15)*	RTN	P851M P852M P856M P857M

Syntax: [label]␣ RTN ␣ A15

This instruction allows the return from an interrupt routine, a trap routine or subroutine. It must therefore be the last instruction of that routine. The instruction reloads the PSW and P-register which have previously been loaded into a memory stack by a Call Function instruction. The stack-pointer A15 is automatically updated.
On the P852M bit 9 of the PSW (ENB) is always set to 1. On the other computers bit 9 must be set, if required.
Note: By forcing bit 15 of the PSW to 1, the user may switch the machine from system mode to user mode (P851M, P856M and P857M).

Type	*Function (P852M)*		*Function (other)*	
T3	(A15) + 2	→ A15	(A15) + 2	→ A15
	((A15)) 0–5	→ PLR	((A15)) 0–5	→ PLR
	((A15)) 6,7	→ CR	((A15)) 6,7	→ CR
	bit 9 is set to 1	→ ENB	((A15)) 9	→ ENB
	SU bit does not exist		((A15)) 15	→ SU
	(A15) + 2	→ A15	(A15)+ 2	→ A15
	((A15))	→ P	((A15))	→ P



Remark:
r2 must be ≠ 0.

EXR *Execute register* **EXR**

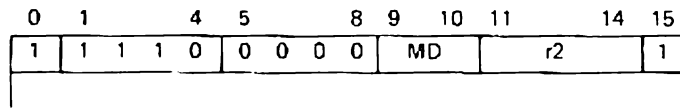
P851M
P852M
P856M
P857M

Syntax: [label]␣ EXR[*]␣ r2

This instruction executes the instruction in r2 (T1) or pointed to by the contents of r2 (T3). r2 may not contain a double word instruction, a CF instruction, RTN instruction or another EXR, EX or EXK instruction.

Type	*Function*	*MD*	*Syntax*
T1	(r2) is executed	00	EXR r2
T3	((r2)) is executed	01	EXR* r2

Condition register: CR is set by the instruction in (r2).



EXK

Execute constant

EXK

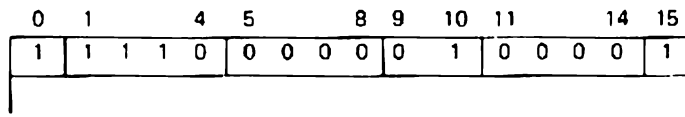
P851M
P852M
P856M
P857M

Syntax: [label]␣ EXK ␣ lk

This instruction performs the operand instruction contained in lk. The memory address may not contain a double word instruction, a CF instruction, RTN instruction or another EXK, EX or EXR instruction.

Type	*Function*
T2	lk is executed

Condition register: CR is set by the instruction in lk.



EX *Execute* **EX**

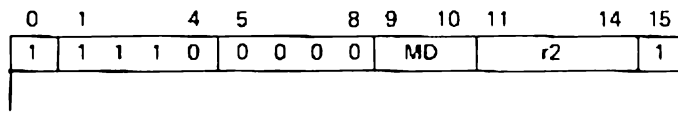
P851M
P852M
P856M
P857M

Syntax: [label]␣ EX[•]␣ m[, r2]

This instruction executes the operand instruction contained in the effective memory address. The memory address may not contain a double word instruction, a CF instruction, RTN instruction or another EX, EXK, or EXR instruction.

Type	*Function*		*MD*	*Syntax*
T4	(m)	is executed	10	EX m
T5	(m + (r2))	"	10	EX m, r2
T6	((m))	"	11	EX• m
T7	((m + (r2)))	"	11	EX• m, r2

Condition register: CR is set by the instruction in the effective memory address.



SLA *Single left arithmetic shift* **SLA**

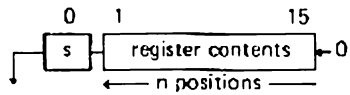
P851M
P852M
P856M
P857M

Syntax: [label] ␣ SLA ␣ r3, n

The bits of the register specified by r3 are shifted left n bit positions. Overflow occurs when the sign bit was modified during the operation. Vacant bits are filled with zeroes.

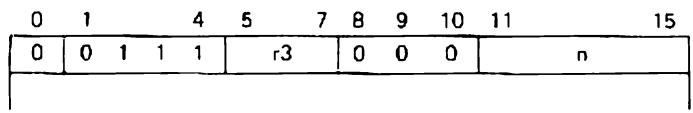
Type *Function*

T8



Condition register:

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



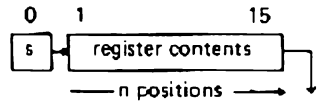
Remark:
r3 ≠ 0.

Syntax: [label]␣ SRA ␣ r3, n

The contents of the register specified by r3 are shifted right n bit positions. The sign bit is not changed. It is shifted into the vacant position(s) of the register. The vacant bit positions are filled with the same values as the sign bit, i.e. either 0 or 1.
If n ≥ 15, all bits of the register will be the same as the sign bit.

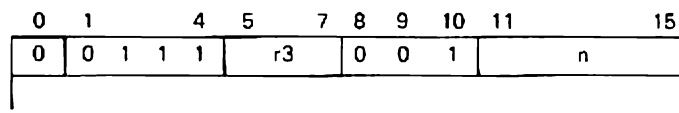
Type *Function*

T8



Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0



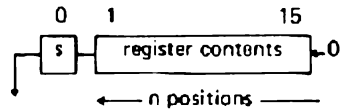
Remark:
r3 ≠ 0.

Syntax: [label] ␣ SLL ␣ r3, n

The bits of the register specified by r3 are shifted left n bit positions. Vacant bits become zero. After 16 or more shifts the whole register contains zero.

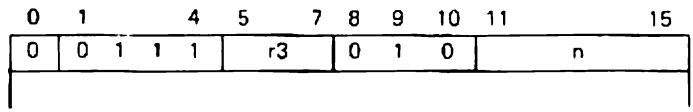
Type *Function*

T8



Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0



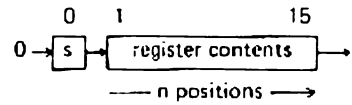
Remark:
r3 ≠ 0.

Syntax: [label] ␣ SRL ␣ r3, n

The contents of the register specified by r3 are shifted right n bit positions. Vacant bits become zero. After 16 or more shifts the register contains zero.

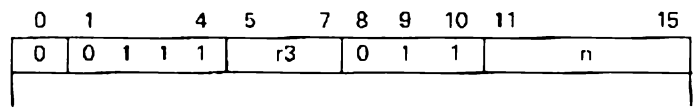
Type *Function*

T8



Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0



Remark:
r3 ≠ 0.

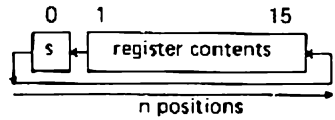
SLC	*Single left circular shift*	SLC	P851M P852M P856M P857M

Syntax: [label] ␣ SLC ␣ r3, n

The contents of the register specified by r3 are shifted left, end around, n bit positions.

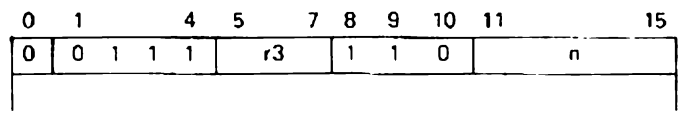
Type *Function*

T8



Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0



Remark:
r3 ≠ 0.

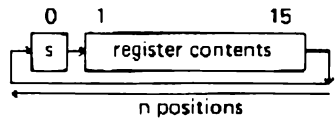
SRC *Single right circular shift* **SRC** P851M P852M P856M P857M

Syntax: [label]␣SRC␣r3, n

The contents of the register specified by r3 are shifted right, end around, n bit positions.

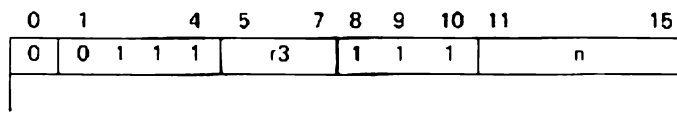
Type *Function*

T8



Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0



Remark:
r3 ≠ 0.

SLN *Single left and normalize shift* **SLN**

P851M
P852M
P856M
P857M

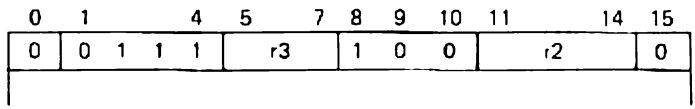
Syntax: [label] ␣ SLN . , r3, r2

The contents of the register specified by r3 are shifted left until the two most significant bits differ. The sign bit remains unaffected; zero bits are inserted in the least significant positions. The number of shifted positions is placed in the register specified by r2.

Type *Function*

T8

Condition register: Unchanged



Remark:
* r3 ≠ 0.
* If (r3) = 0 the number of shifted positions will be 16.
* Restricted to system mode if r2 = A15.

SRN *Single right and normalize shift* **SRN**

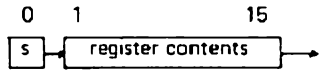
P851M
P852M
P856M
P857M

Syntax: [label]␣SRN␣r3, r2

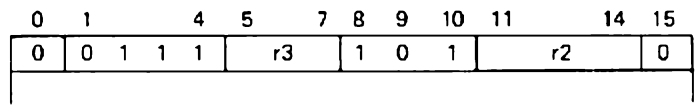
The contents of the register specified by r3 are shifted right until a 1-bit appears in bit 15 of that register.
The sign bit is not changed and is copied each time a shift is given.
The number of times a shift had to be performed is placed in the register specified in r2.

Type *Function*

TB



Condition register: Unchanged



Remark:
- r3 ≠ 0.
- If (r3) = 0 the number of shifted positions will be 16.
- Restricted to system mode if r2 = A15.

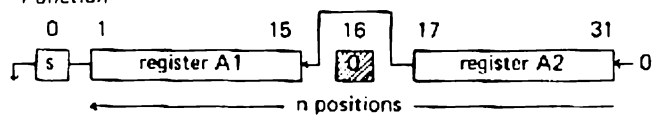
DLA *Double left arithmetic shift* **DLA** P851M P852M P856M P857M (Softw. sim.)

Syntax: [label] ␣ DLA ␣ n

This instruction treats the A1 and A2 register as one 31-bit register (bit 0 of A2 is set to zero). The contents are shifted left n positions and zeroes are placed from the right in vacated positions.
Overflow occurs when the sign bit is changed during execution of this instruction.

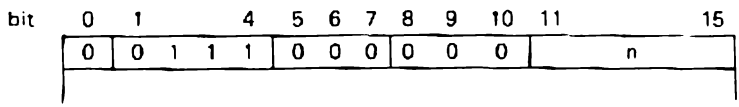
Type *Function*

T8



Condition register:

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



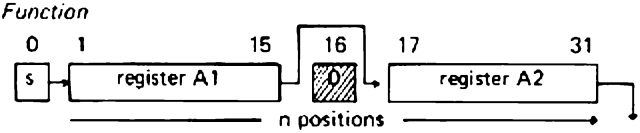
DRA *Double right arithmetic shift* **DRA** P851M P852M P856M P857M (Softw. sim.)

Syntax: [label] ␣ DRA ␣ n

This instruction treats the A1 and A2 registers as one 31-bit register. The contents are shifted right n positions and zeroes or ones are propagated into vacated positions depending on the value of the sign bit of A1.
After 30 or more shifts the two registers are filled the value of the sign bit (all zeroes or ones), except for the sign bit of A2 which is always set to 0.

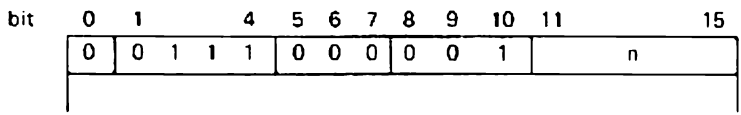
Type *Function*

T8



Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0



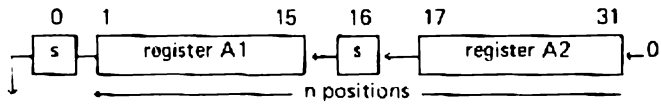
DLL — *Double left logical shift* — DLL — P851M, P852M, P856M, P857M (Softw. sim.)

Syntax: [label] ␣ DLL ␣ n

This instruction treats the registers A1 and A2 as one 32-bit register. The contents are shifted left n positions. Zeroes are propagated into vacated positions of A1 and A2.

Type *Function*

T8



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

bit

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	1	1	1	0	0	0	0	1	0	n			

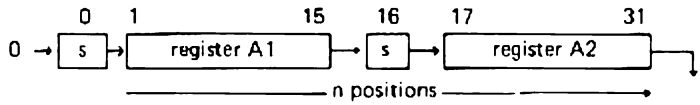
DRL *Double right logical shift* **DRL** P851M P852M P856M P857M (Softw. sim.)

Syntax: [label] ␣ DRL ␣ n

The A1 and A2 registers are treated as one 32-bit register. The contents are shifted right n positions. Zeroes are propagated into vacated positions. The max. number of shifts is 31.

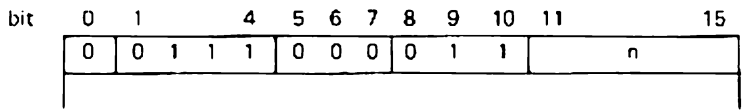
Type *Function*

T8



Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0



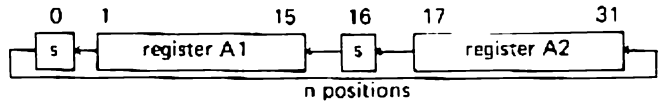
DLC *Double left circular shift* **DLC** P851M P852M P856M P857M (Softw. sim.)

Syntax: [label] ␣ **DLC** ␣ n

The A1 and A2 registers are treated as one 32-bit register. The contents are shifted left, end around, n positions.

Type *Function*

T8



Condition register:

CR = 0 if result = 0
1 if result > 0
2 if result < 0

bit

0	1			4	5	6	7	8	9	10	11			15
0	0	1	1	1	0	0	0	1	1	0	n			

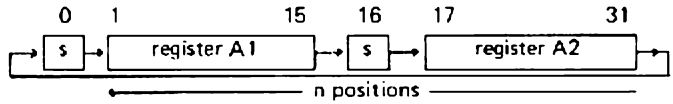
DRC *Double right circular shift* **DRC** P851M P852M P856M P857M (Softw. sim.)

Syntax: [label]␣ DRC ␣ n

The A1 and A2 registers are treated as one 32-bit register. The contents are shifted right, end around, n positions.

Type *Function*

T8



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

bit

0	1			4	5	6	7	8	9	10	11			15
0	0	1	1	1	0	0	0	1	1	1	n			

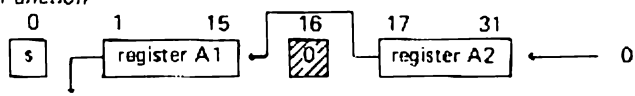
DLN *Double left and normalize shift* **DLN** P851M P852M P856M P857M (Softw. sim.)

Syntax: [label]␣ DLN ␣ r2

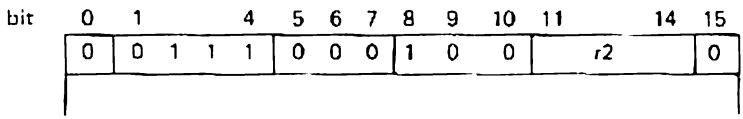
The A1 and A2 registers are treated as one 31-bit register. Its contents are shifted left until bit zero and bit one have a different value.
Zeroes are shifted, from the right hand side on, into vacated positions of the register. The sign bit of register A1 remains unchanged.
The number of shifted positions is stored in register r2.
The sign bit of A2 becomes zero.

Type *Function*

T8



Condition register: Unchanged



Remark:
Restricted to system mode if r2 = A15.

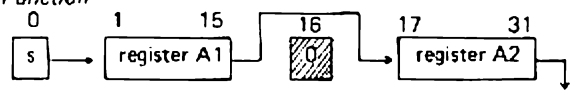
DRN — *Double right and normalize shift* — **DRN** — P851M, P852M, P856M, P857M (Softw. sim.)

Syntax: [label] ␣ DRN ␣ r2

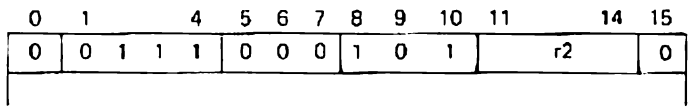
The A1 and A2 registers are treated as one 31-bit register. The contents are shifted right until a 1-bit appears in the least significant position of the register. The sign bit is shifted to the right each time a shift takes place. The number of shifted postions is stored in register r2.
The sign bit of A2 becomes zero.

Type *Function*

T8



Condition register: Unchanged



Remark:
Restricted to system mode if r1 = A15.

MVF	*Move Table Forward*	MVF	P857M

Syntax: [label]␣ MVF ␣ r2

This instruction copies a string of consecutive words from one memory area into another area, beginning with the last location from the buffer to be copied towards the start address of that buffer. Should the buffer to be copied and the receiving buffer overlap, the user must take care not to overwrite the contents of the locations in the buffer to be copied. Use in that case the instruction MVB.

- register A1 must be loaded with the start address of the memory area to be copied.
- register A2 must be loaded with the start address of the receiving buffer.
- register r2 must contain the number of characters to be copied (the number must be even and unsigned).

The execution of this instruction may be interrupted after any word transfer. When the interrupt is accepted the contents of the instruction counter, which is pointing to this instruction, are saved in the stack. The contents of A1 and A2 remain unchanged.

Register r2 contains the remaining number of characters to be transferred. The execution of this instruction is resumed when the interrupt has been serviced. When the execution is terminated A1 and A2 contain the initial values.

Type	*Function*
T8	(r2) − 2 → r2, ((A1) + (r2)) → (A2) + (r2)
	−
	−
	0 → r2, ((A1)) → (A2)

Condition register: Unchanged

bit	0	1			4	5		7	8		10	11			14	15
	0	1	1	1	0	0	0	0	0	0	0	r2				0

Remark:

- When used in system mode r2 = A15 or ≠ A15.
- When used in user mode r2 ≠ A15.
- r2 must be ≠ 0.

Syntax: [label] ␣ MVB ␣ r2

This instruction copies a string of consecutive words from one memory area into another area, beginning with the first location of the buffer to be copied towards the last location of that buffer. Should the buffer to be copied and the receiving buffer overlap, the user must take care not to overwrite the contents of locations in the buffer to be copied. Use in that case the instruction MVF.

- register A1 must contain the start address of the buffer to be copied.
- register A2 must contain the start address of the receiving buffer.
- register r2 must contain the number of characters to be copied. (The number must be even and unsigned.)

The execution of this instruction may be interrupted after any word transfer. When the interrupt is accepted the contents of the instruction counter, which points to this instruction, are saved in the stack. The contents of registers A1 and A2 point to the first location to be transferred when resuming the execution. Register r2 contains the remaining number of characters to be transferred.
The execution of the instruction is resumed when the instruction interrupt has been serviced. When the execution is terminated A1 and A2 point to the first address after the buffer.

Type	*Function*
T8	((A1)) → (A2) (r2) − 2 → r2; (A1) + 2 → A1; (A2) + 2 → A2; ((A1)) → (A2) − − 0 → r2; (A1) + 2 → A1; (A2) + 2 → A2

Condition register: Unchanged

bit	0	1			4	5		7	8	9	10	11			14	15
	0	1	1	1	1	0	0	0	0	0	0	r2				0

Remark:

- When used in system mode r2 = A15 or ≠ A15.
- When used in user mode r2 ≠ A15.
- r2 must be ≠ 0.

Syntax: [label] ␣ MVUS ␣ r2

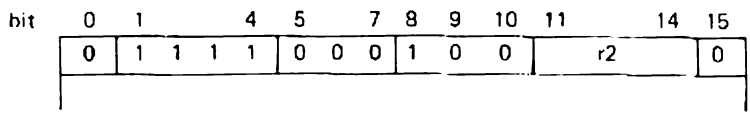
This instruction is used to copy a table of consecutive words from a user area (sending buffer) to a system area (receiving buffer), beginning with the first location towards its last location.

- register A1 must contain the logical start address (MMU) of the buffer to be copied.
- register A2 must contain the physical start address (NO MMU) of the receiving buffer.
- register r2 must be loaded with the number of characters to be copied. This number must be even and not signed.

The execution of this instruction may be interrupted after any word transfer. When the interrupt is accepted the contents of the instruction counter, which points to this instruction, are saved in the stack. The contents of A1 and A2 point to the first location to be transferred when the execution is resumed.
Register r2 contains the remaining number of characters to be transferred. The execution of this instruction is resumed when the interrupt is serviced. When the execution is terminated A1 and A2 point to the first address after the receiving buffer.

Type	*Function*
T8	((A1)) → (A2) (r2)− 2 → r2; (A1) + 2 → A1; (A2) + 2 → A2; ((A1)) → (A2) − − 0 → r2; (A1) + 2 → A1; (A2) + 2 → A2

Condition register: Unchanged



Remark:

- When used in system mode r2 = A15 or ≠ A15.
- When used in user mode r2 ≠ A15. In that case or if MMU is not available this instruction is the same as the MVB instruction.
- r2 must be ≠ 0.

Syntax : [label] ␣ MVSU ␣ r2

This instruction is used to copy a table of consecutive words from a system area (sending buffer) to a user area (receiving buffer), beginning with the last location of the sending area towards the first location.

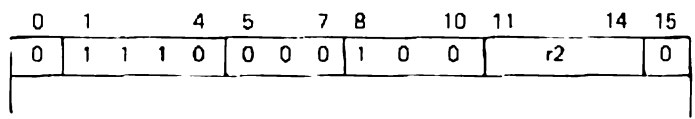
- register A1 must contain the physical start address (NO MMU) of the sending buffer.
- register A2 must contain the logical start address (MMU) of the receiving buffer.
- register r2 must be loaded with the number of characters to be copied (this number must be even and unsigned).

The execution of this instruction may be interrupted after any word transfer. When the interrupt is accepted the contents of the instruction counter, which points to this instruction, are saved in the stack. The contents of registers A1 and A2 remain unchanged.

Register r2 contains the remaining number of characters to be transferred. The execution of the instruction is resumed when the interrupt is serviced. When the execution is terminated A1 and A2 contain their initial values.

Type	*Function*
T8	(r2) − 2 → r2, ((A1) + (r2)) → (A2) + (r2) − − 0 → r2 , ((A1)) → (A2)

Condition register: Unchanged



Remark:

- When used in system mode r2 = A15 or ≠ A15.
- When used in user mode r2 ≠ A15. In that case or if MMU is not available this instruction is the same as the MVF instruction.
- r2 must be ≠ 0.

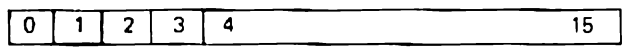
WER	*Write external register*	WER	P851M P852M P856M P857M

Syntax: [label]␣ WER ␣ r3, address

The contents of the register specified by r3 are transferred to the external register whose address is specified in bits 8–15. The contents of the register specified by r3 and the condition register remain unchanged.

Two WER instructions must be used to send two control words, one containing a buffer address and the second one containing the number of words or characters to be transferred, to two registers on the I/O Processor.

1st control word



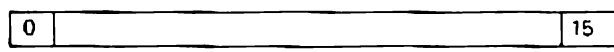
where: bit 0 = 0 if char. mode
1 if word mode

bit 1 = 0 if transfer is CU → MEM
1 if transfer is MEM → CU

bits 2, 3 are used to extend the memory address in 2nd control word to > 32K.

bits 4 through 15 = transfer length in words or characters.

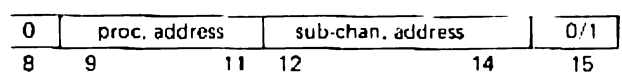
2nd control word



where: bits 0 through 14 = memory address

bit 15 (if char. mode) = 0 left hand character
1 right hand character

The layout of bits 8 through 15 of the WER instruction is:



bit 8 = 0

bits 9 through 11 must be set to zero for the P851M (only one processor connection.
On P852M, P856M or P857M it may be a number from 0 through 7.

bits 12 through 14 device address

bit 15 = 0 WER instruction for the 1st control word
1 WER instruction for the 2nd control word

Note: When a device must be addressed via a multiple control unit card, specify the lowest address.

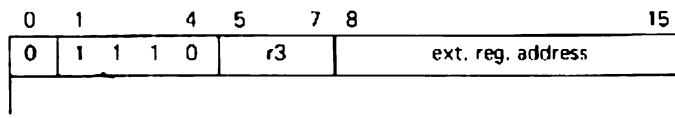
Example:

Output on cassette

LDK A1,/0084	Send 132 characters.
LDKL A2,BUFFER	Take the contents of 'BUFFER'.
WER A1,/A	The cassette has address /05 and is connected to I/O Processor numero 0. Bit 15 = 0 (1st control word).
WER A2,/8	Send the 2nd control word. Bit 15 = 1.

Type	*Function*
T8	(r3) → extern reg.

Condition register: Unchanged



Remark:

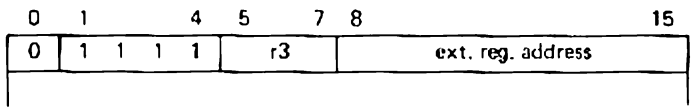
- r3 must be ≠ 0.
- This instruction may only be used in system mode.

Syntax: [label] ␣ RER ␣ r3, address

The contents of the external register, specified by its address, are transferred to the register specified by r3. The contents of the external register remain unchanged. Bits 6 and 7 of the external register are copied to the condition register.
Through this instruction the user can check how many characters or words have been transferred.

Type	*Function*
T8	(extern reg) → r3

Condition register: (extern reg 6,7) → CR



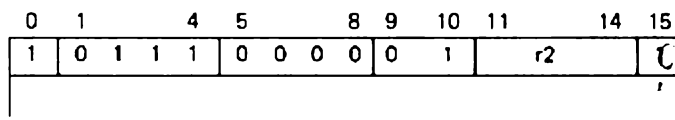
Remark:
- r3 must be ≠ 0.
- This instruction may only be used in system mode.

Syntax: [label]␣ TLR ␣ r2

This instruction loads 16 consecutive registers, TR0 through TR15, which are located on the MMU, with the contents of 16 consecutive memory locations, the first one being indicated in register r2.

Type	*Function*	
T3	((r2))	→ TR0
	((r2) + 2)	→ TR1
	–	
	–	
	–	
	((r2) + 15 x 2)	→ TR15

Condition register: Unchanged



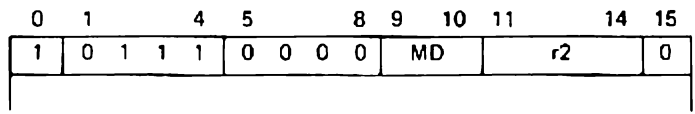
Remark:
This instruction is restricted to system mode.

Syntax: [label]␣ TL[•]␣ m[, r2]

This instruction loads 16 consecutive registers, TR0 through TR15, which are located on the MMU, with the contents of 16 consecutive memory locations.
The address of the first memory location is indicated by the effective memory address.

Type	*Function*		*Syntax*	
T4	(m) . . (m + 15x2)	→ TR0 . . TR15	TL	m
T5	(m + (r2)) . . (m + (r2) + 15x2)	→ TR0 . . TR15	TL	m, r2
T6	((m)) . . ((m) + 15x2)	→ TR0 . . TR15	TL*	m
T7	((m+ (r2))) . . ((m + (r2)) + 15x2)	→ TR0 . . TR15	TL*	m, r2

Condition register: Unchanged



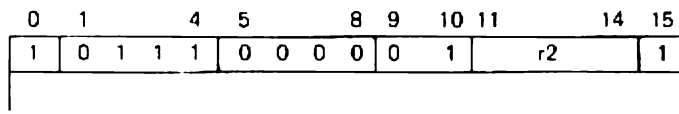
Remark:
This instruction is restricted to system mode.

Syntax: [label] ␣ TSR ␣ r2

This instruction places the contents of 16 consecutive registers, TR0 through TR15, located on the MMU, in 16 consecutive memory locations. The first memory location is indicated by the contents of register r2.

Type	*Function*
T3	(TR0) → (r2)
	(TR1) → (r2) + 2
	—
	—
	—
	(TR15) → (r2) + 15x2

Condition register: Unchanged



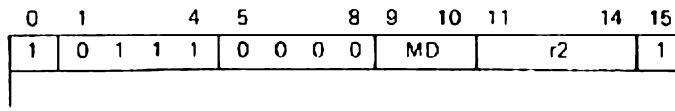
Remark:
This instruction is restricted to system mode.

Syntax: [label]_ TS[•]_ m[, r2]

The contents of 16 consecutive registers, TR0 through TR15, located on the MMU, replace the contents of 16 memory locations. The first memory location is indicated by the effective memory address.

Type	*Function*	*Syntax*	
T4	(TR0) → m		
	(TR1) → m + 2		
	–		
	(TR15) → m + 15x2	TS	m
T5	(TR0) → m + (r2)		
	(TR1) → m + (r2) + 2		
	–		
	(TR15) → m + (r2) + 15x2	TS	m, r2
T6	(TR0) →(m)		
	(TR1) →(m + 2)		
	–		
	(TR15) →(m + 15x2)	TS*	m
T7	(TR0) →(m + (r2))		
	(TR1) →(m + (r2) + 2)		
	–		
	(TR15) →(m + (r2) + 15x2)	TS*	m, r2

Condition register: Unchanged



Remark:
This instruction is restricted to system mode.

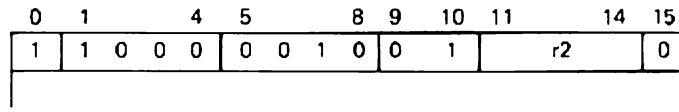
Syntax: [label]␣␣FLDR␣␣r2

The contents of three consecutive memory locations are loaded into three accumulators FPA1, FPA2, FPA3 on the Floating Point Processor. The first memory location is indicated in the register r2.

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

Condition register:

CR = 0 if floating point operand = 0
1 if floating point operand > 0
2 if floating point operand < 0



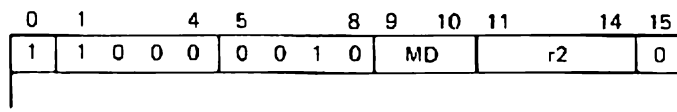
Syntax: [label]␣ FLD[•]␣ m[, r2]

The contents of three consecutive memory locations are loaded into three accumulators FPA1, FPA2 and FPA3 on the Floating Point Processor. The first memory location is indicated by the effective memory address.

Type	*Function*		*MD*	*Syntax*
T4	(m)	→ FPA1	10	FLD m
	(m + 2)	→ FPA2		
	(m + 4)	→ FPA3		
T5	(m + (r2))	→ FPA1	10	FLD m, r2
	(m + (r2) + 2)	→ FPA2		
	(m + (r2) + 4)	→ FPA3		
T6	((m))	→ FPA1	11	FLD* m
	((m) + 2)	→ FPA2		
	((m) + 4)	→ FPA3		
T7	((m + (r2)))	→ FPA1	11	FLD* m, r2
	((m + (r2)) + 2)	→ FPA2		
	((m + (t2)) + 4)	→ FPA3		

Condition register:

CR = 0 if floating point operand ≠ 0
1 if floating point operand > 0
2 if floating point operand < 0
3 unnormalized operand (operation aborted)

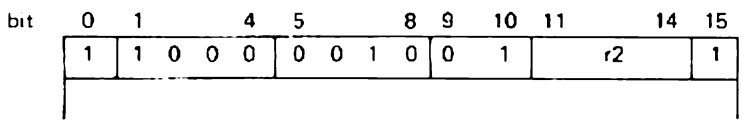


Syntax: [label]␣ FSTR ␣ r2

The contents of three accumulators FPA1, FPA2 and FPA3 on the Floating Point Processor replace the contents of three consecutive memory locations. The first location is indicated in register r2.

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

Condition register: Unchanged



Syntax: [label]␣ FST[•]␣ m[, r2]

The contents of three accumulators FPA1, FPA2 and FPA3 on the floating point processor replace the contents of three consecutive memory locations. The first location is indicated by the effective memory address.

Type	*Function*	*MD*	*Syntax*
T4	(FPA1) → m	10	FST m
	(FPA2) → m + 2		
	(FPA3) → m + 4		
T5	(FPA1) → m + (r2)	10	FST m, r2
	(FPA2) → m + (r2) + 2		
	(FPA3) → m + (r2) + 4		
T6	(FPA1) → (m)	11	FST• m
	(FPA2) → (m) + 2		
	(FPA3) → (m) + 4		
T7	(FPA1) → (m + (r2))	11	FST• m, r2
	(FPA2) → (m + (r2) + 2)		
	(FPA3) → (m + (r2) + 4)		

Condition register: Unchanged