EC16 Microprocessor Core

Instruction Set Architecture V1.0

Summary

The EC16 is a microprocessor core written in VHDL with the following features

- 64K * 16-bit external memory space for code, data and I/O
- 256 * 16-bit internal ram for registers, pointers, stack and scratchpad
- 51 instructions
  - instruction length: 47x one word, 4x two words
  - speed: 18 one-cycle, 16 two-cycle, 9 three-cycle, 8 one/two-cycle (branch)
- 4 maskable prioritized interrupts

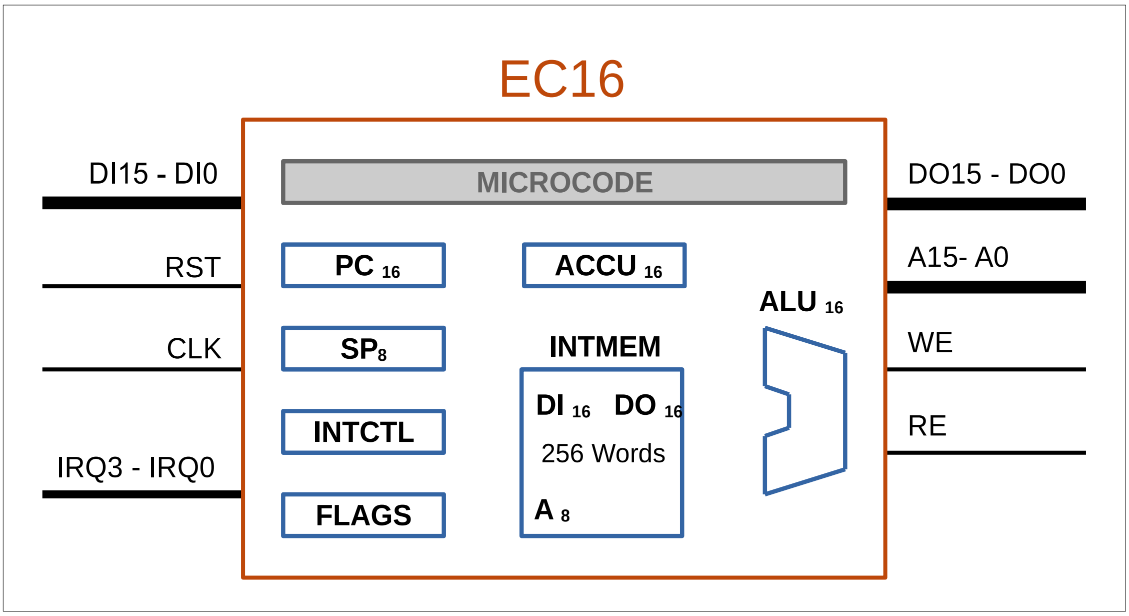

Figure 1: simplified block diagram

Memory Model

The EC16 has two completely separate memory spaces, the external memory space (EXTMEM) and a small but faster accessible internal memory (INTMEM).

While the EXTMEM space can be populated as required with ram, rom and I/O devices, the INTMEM space is always fully populated with an EBRam organized as 256 x 16 bit.

The INTMEM cells can be used as registers, pointers to INTMEM or EXTMEM locations, scratchpad ram or stack space. The stack pointer is a separate register that is initialized to 0xFF at reset and grows towards 0x00. The usual caution has to be exercised to avoid a conflict between the growing stack and the other variables.

Internal Memory (INTMEM) 256 x 16 bit		
Registers Pointers Sratchpad	0xFF ⇳ 0x00	Stackpointer ⇩

External Memory (EXTMEM) 64K x 16 bit	
0xFFFF ⇳	Code, Data, I/O
0x0020	IRQ3
0x0018	IRQ2
0x0010	IRQ1
0x0008	IRQ0
0x0000	Reset

The EXTMEM must be populated with at least a small amount of ROM (pre-initialized RAM) beginning at address 0 since the reset and interrupt vectors are located there. The rest of the address space can be used as required.

The EC16 has a reduced instruction set architecture that is strongly focussed on the use of the faster INTMEM. Each of the 256 INTMEM locations can be used as a register or a pointer and its 8-bit address is encoded in the lower half of the opcode which speeds up fetch- and execution time. Read and write access to the EXTMEM space is handled by only two instructions (MOVXI A INTMEM / MOVXI INTMEM A), both of which use an INTMEM cell to provide the address (indirect addressing via pointer). Nevertheless block operations can be very efficiently implemented with only three registers (source, destination and counter), in combination with the INC/DEC- and branch instructions.

Interrupts

The EC16 has four interrupt inputs : IRQ0 – IRQ3. IRQ0 has the highest, IRQ3 the lowest priority. All four interupts are triggered by a rising edge. Prioritization and execution are handled via the INTCTL block. This block contains i.a. three registers : IRR (Interrupt Request Register), IMASK and IIP (Interrupt In Progress).

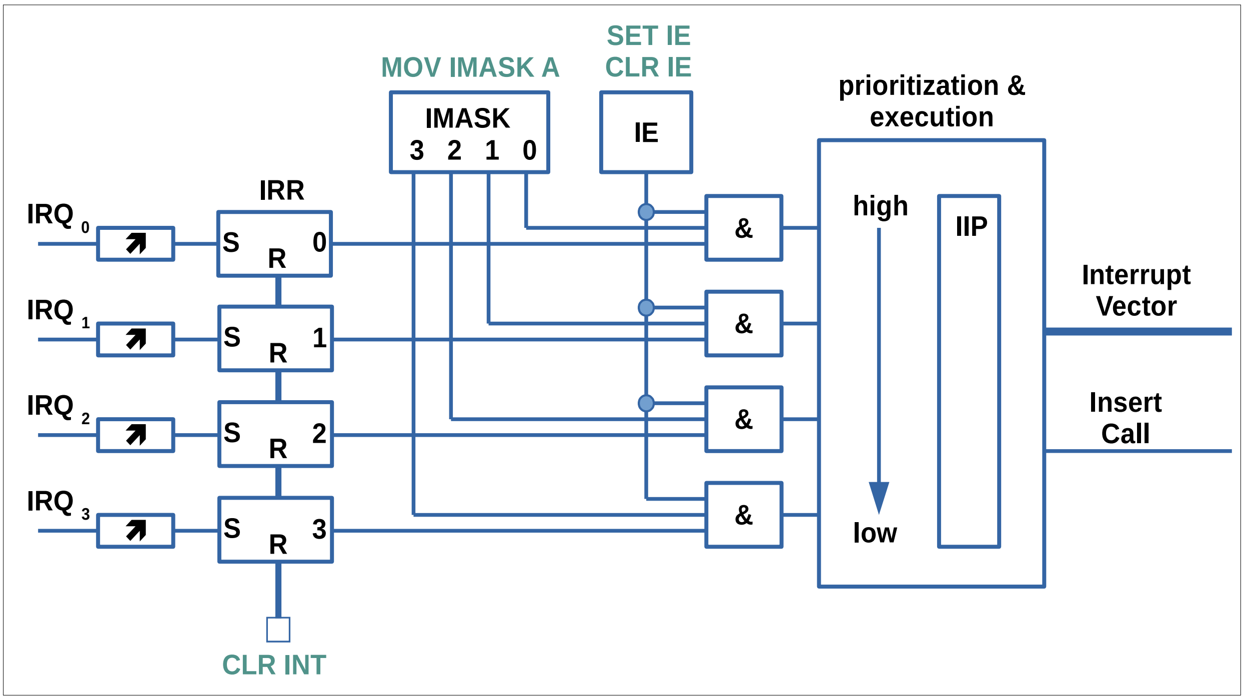

Figure 2: INTCTL (Interrupt Control Block)

A rising edge on IRQ_n is always registered in IRR_n regardless of the state of $IMASK_n$ and IE. If, however, the rising edge occurs in the same clock cycle as a CLR INT instruction (which resets all IRR flags at once), the clear instruction takes precedence.

$IMASK_n$ enables (1) or disables (0) IRQ_n. IMASK can be written with a MOV IMASK A instruction, which transfers the four least significant bits of the AKKU to $IMASK_{3\text{-}0}$.

The flag IE enables (1) or disables (0) all interrupts simultaneously. IE can be set/reset with the SET IE, CLR IE and the MOV FLAGS A instructions.

Unless it is cleared by a CLR INT instruction, the IRR_n flag remains set until its interrupt is executed. Interrupt execution happens as soon as IRR_n , $IMASK_n$ and IE are all set and no interrupt with higher priority is in progress.

The current instruction is finished, IRR_n is cleared and IIP_n is set instead, blocking all lower level interrupts. Then the address of the next instruction is pushed onto the stack and a call to the corresponding interrupt vector is executed. IIP_n stays set until the interrupt service routine is finished with a RETI instruction. RETI clears IIP_n, pops the return address from the stack and resumes executing from where it left of.

Notice that as soon as IRR_n is cleared, it will register the next rising edge on IRQ_n, even if the current ISR (Interrupt Service Routine) is still in progress.

Notice furthermore that at least one instruction of the interrupted program context is executed before the next interrupt is taken.

An IRQ with higher priority will interrupt the current ISR. This can be avoided if necessary by temporarily clearing the IE flag. Typical examples are atomic operations like a read-modify-write access to a peripheral or the handling of semaphores. Only the crucial instructions should be framed by a pair of CLR IE and SET IE.

Each interrupt level needs one word of stack space for the return address. No variables, registers or flags are saved automatically. It is up to the user to take care of it.

Instruction Set

The EC16 opcodes consist of three fields

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CYC1	CYC0	INST5	INST4	INST3	INST2	INST1	INST0	IA7	IA6	IA5	IA4	IA3	IA2	IA1	IA0

- **Bit 15:14** - CYC1:0 : number of clock cycles

CYC1:0	Number of clock cycles
00	1
01	2
10	3
11	2/1 (branch/no branch)

- **Bit 13:8** - INST5:0 : instruction index
- **Bit 7:0** - IA7:0 : depending on instruction :
  - INTMEM address (8 bit unsigned)
  - Branch offset (8 bit signed)
  - 0 if none of the above applies

Only the instructions **JMPD U16**, **CALLD U16**, **LOAD A U16** and **LOAD INTMEM U16** are two-word instructions, i.e. the opcode is followed by a 16 bit constant. In case of JMPD and CALLD this is the target address, in case of LOAD the value is loaded into the Accu respectively the INTMEM cell.

In the text and tables the following representation is used:

- INTMEM stands for an 8-bit unsigned INTMEM address (0 - 255)
- EXTMEM stands for a 16-bit unsigned EXTMEM address (0 - 65536)
- U16 stands for a 16-bit unsigned number
- u8 in OPcodes like 0x43u8 stands for an 8-bit unsigned INTMEM address (0 - 255), the actual opcodes being 0x4300 up to 0x43FF
- s8 in OPcodes like 0xC0s8 stands for an 8-bit signed address offset (-127 to +128), the actual opcodes being 0xC000 up to 0xC0FF

Instructions sorted by Function				
Mnemonic	OPCode	Cycles	Words	
Arithmetic				
ADD A INTMEM	0x43u8	2	1	A ← A + INTMEM
ADDC A INTMEM	0x41u8	2	1	A ← A + INTMEM + CARRY
CMP A INTMEM	0x44u8	2	1	Compare A and INTMEM
DEC INTMEM	0x46u8	2	1	Decrement INTMEM
INC INTMEM	0x47u8	2	1	Increment INTMEM
SUB A INTMEM	0x42u8	2	1	A ← A - INTMEM
SUBB A INTMEM	0x40u8	2	1	A ← A - INTMEM - CARRY
MUL A INTMEM	0x45u8	2	1	A * INTMEM, low(U32) → A, high(U32) → INTMEM
Logic				
AND A INTMEM	0x4Cu8	2	1	A ← A AND INTMEM
NOT A	0x2F00	1	1	A ← ~A
OR A INTMEM	0x4Du8	2	1	A ← A OR INTMEM
ROL A	0x2800	1	1	C ← A(15 ← 14 ← … ← 2 ← 1 ← 0) ← C
ROR A	0x2900	1	1	C → A(15 → 14 → … → 2 → 1 → 0) → C
SHL A	0x2A00	1	1	C ← A(15 ← 14 ← … ← 2 ← 1 ← 0) ← '0'
SHR A	0x2B00	1	1	'0' → A(15 → 14 → … → 2 → 1 → 0) → C
SWAP A	0x2500	1	1	A ← A(15..8) A(7..0)
XOR A INTMEM	0x4Eu8	2	1	A ← A XOR INTMEM
Flag Manipulation				
CLR C	0x1000	1	1	Clear CARRY flag
CLR IE	0x0200	1	1	Clear Interrupt Enable flag
CLR INT	0x0400	1	1	Clear all pending interrupts
SET C	0x1100	1	1	Set CARRY flag
SET IE	0x0300	1	1	Set Interrupt Enable flag
Program Flow Control				
BRCC S8	0xC0s8	2/1	1	Branch if CARRY flag is '0'
BRCS S8	0xC4s8	2/1	1	Branch if CARRY flag is '1'

BRNC S8	0xC2s8	2/1	1	Branch if NEG flag is '0'
BRNS S8	0xC6s8	2/1	1	Branch if NEG flag is '1'
BROC S8	0xC1s8	2/1	1	Branch if OV flag is '0'
BROS S8	0xC5s8	2/1	1	Branch if OV flag is '1'
BRZC S8	0xC3s8	2/1	1	Branch if ZERO flag is '0'
BRZS S8	0xC7s8	2/1	1	Branch if ZERO flag is '0'
CALLD U16	0xA100	3	2	Call subroutine, Direct 16-bit EXTMEM address
CALLI INTMEM	0xA3u8	3	1	Call subroutine, Indirect via pointer in INTMEM
JMPD U16	0xA000	3	2	Jump to Direct 16-bit EXTMEM address
JMPI INTMEM	0xA2u8	3	1	Jump Indirect via pointer in INTMEM
RETI	0x8500	3	1	Return from Interrupt
RETS	0x8400	3	1	Return from Subroutine
Data Transport				
LOAD A U16	0x6000	2	2	Load 16-bit value into A
LOAD INTMEM U16	0x61u8	2	2	Load 16-bit value into INTMEM
MOV A INTMEM	0x50u8	2	1	Move INTMEM to A
MOV A STATUS	0x1400	1	1	Move SP, IE and Flags to A
MOV FLAGS A	0x1200	1	1	Move A to Flags
MOV IMASK A	0x0500	1	1	Move A to Interrupt Mask
MOV INTMEM A	0x16u8	1	1	Move A to INTMEM
MOV SP A	0x1300	1	1	Move A to SP
MOVI A INTMEM	0x80u8	3	1	Move Indirect INTMEM to A. (See below)
MOVI INTMEM A	0x52u8	2	1	Move A to Indirect INTMEM. (See below)
MOVXI A INTMEM	0x83u8	3	1	Move EXTMEM (via pointer in INTMEM) to A
MOVXI INTMEM A	0x82u8	3	1	Move A (via pointer in INTMEM) to EXTMEM
POP A	0x5100	2	1	Pop value from Stack to A
PUSH A	0x1500	1	1	Push value in A to Stack
Miscellaneous				
NOP	0x0000	1	1	No operation

Instructions sorted by Mnemonic				
Mnemonic	OPCode	Cycles	Words	Action
ADD A INTMEM	0x43u8	2	1	A ← A + INTMEM
ADDC A INTMEM	0x41u8	2	1	A ← A + INTMEM + CARRY
AND A INTMEM	0x4Cu8	2	1	A ← A AND INTMEM
BRCC S8	0xC0s8	2/1	1	Branch if CARRY flag is '0'
BRCS S8	0xC4s8	2/1	1	Branch if CARRY flag is '1'
BRNC S8	0xC2s8	2/1	1	Branch if NEG flag is '0'
BRNS S8	0xC6s8	2/1	1	Branch if NEG flag is '1'
BROC S8	0xC1s8	2/1	1	Branch if OV flag is '0'
BROS S8	0xC5s8	2/1	1	Branch if OV flag is '1'
BRZC S8	0xC3s8	2/1	1	Branch if ZERO flag is '0'
BRZS S8	0xC7s8	2/1	1	Branch if ZERO flag is '0'
CALLD U16	0xA100	3	2	Call subroutine, Direct 16-bit EXTMEM address
CALLI INTMEM	0xA3u8	3	1	Call subroutine, Indirect via pointer in INTMEM
CLR C	0x1000	1	1	Clear CARRY flag

CLR IE	0x0200	1	1	Clear Interrupt Enable flag
CLR INT	0x0400	1	1	Clear all pending interrupts
CMP A INTMEM	0x44u8	2	1	Compare A and INTMEM
DEC INTMEM	0x46u8	2	1	Decrement INTMEM
INC INTMEM	0x47u8	2	1	Increment INTMEM
JMPD U16	0xA000	3	2	Jump to Direct 16-bit EXTMEM address
JMPI INTMEM	0xA2u8	3	1	Jump Indirect via pointer in INTMEM
LOAD A U16	0x6000	2	2	Load 16-bit value into A
LOAD INTMEM U16	0x61u8	2	2	Load 16-bit value into INTMEM
MOV A INTMEM	0x50u8	2	1	Move INTMEM to A
MOV A STATUS	0x1400	1	1	Move SP, IE and Flags to A
MOV FLAGS A	0x1200	1	1	Move A to Flags
MOV IMASK A	0x0500	1	1	Move A to Interrupt Mask
MOV INTMEM A	0x16u8	1	1	Move A to INTMEM
MOV SP A	0x1300	1	1	Move A to SP
MOVI A INTMEM	0x80u8	3	1	Move Indirect INTMEM to A. (See below)
MOVI INTMEM A	0x52u8	2	1	Move A to Indirect INTMEM. (See below)
MOVXI A INTMEM	0x83u8	3	1	Move EXTMEM (via pointer in INTMEM) to A
MOVXI INTMEM A	0x82u8	3	1	Move A (via pointer in INTMEM) to EXTMEM
MUL A INTMEM	0x45u8	2	1	A * INTMEM, low(U32) → A, high(U32) → INTMEM
NOP	0x0000	1	1	No operation
NOT A	0x2F00	1	1	A ← ~A
OR A INTMEM	0x4Du8	2	1	A ← A OR INTMEM
POP A	0x5100	2	1	Pop value from Stack to A
PUSH A	0x1500	1	1	Push value in A to Stack
RETI	0x8500	3	1	Return from Interrupt
RETS	0x8400	3	1	Return from Subroutine
ROL A	0x2800	1	1	C ← A(15 ← 14 ← … ← 2 ← 1 ← 0) ← C
ROR A	0x2900	1	1	C → A(15 → 14 → … → 2 → 1 → 0) → C
SET C	0x1100	1	1	Set CARRY flag
SET IE	0x0300	1	1	Set Interrupt Enable flag
SHL A	0x2A00	1	1	C ← A(15 ← 14 ← … ← 2 ← 1 ← 0) ← '0'
SHR A	0x2B00	1	1	'0' → A(15 → 14 → … → 2 → 1 → 0) → C
SUB A INTMEM	0x42u8	2	1	A ← A - INTMEM
SUBB A INTMEM	0x40u8	2	1	A ← A - INTMEM - CARRY
SWAP A	0x2500	1	1	A ← A(15..8) ⇔ A(7..0)
XOR A INTMEM	0x4Eu8	2	1	A ← A XOR INTMEM

Instructions sorted by OPCode				
Mnemonic	OPCode	Cycles	Words	
NOP	0x0000	1	1	No operation
CLR IE	0x0200	1	1	Clear Interrupt Enable flag
SET IE	0x0300	1	1	Set Interrupt Enable flag
CLR INT	0x0400	1	1	Clear all pending interrupts
MOV IMASK A	0x0500	1	1	Move A to Interrupt Mask
CLR C	0x1000	1	1	Clear CARRY flag

SET C	0x1100	1	1	Set CARRY flag
MOV FLAGS A	0x1200	1	1	Move A to Flags
MOV SP A	0x1300	1	1	Move A to SP
MOV A STATUS	0x1400	1	1	Move SP, IE and Flags to A
PUSH A	0x1500	1	1	Push value in A to Stack
MOV INTMEM A	0x16u8	1	1	Move A to INTMEM
SWAP A	0x2500	1	1	A ← A(15..8) ⇔ A(7..0)
ROL A	0x2800	1	1	C ← A(15 ← 14 ← … ← 2 ← 1 ← 0) ← C
ROR A	0x2900	1	1	C → A(15 → 14 → … → 2 → 1 → 0) → C
SHL A	0x2A00	1	1	C ← A(15 ← 14 ← … ← 2 ← 1 ← 0) ← '0'
SHR A	0x2B00	1	1	'0' → A(15 → 14 → … → 2 → 1 → 0) → C
NOT A	0x2F00	1	1	A ← ~A
SUBB A INTMEM	0x40u8	2	1	A ← A - INTMEM - CARRY
ADDC A INTMEM	0x41u8	2	1	A ← A + INTMEM + CARRY
SUB A INTMEM	0x42u8	2	1	A ← A - INTMEM
ADD A INTMEM	0x43u8	2	1	A ← A + INTMEM
CMP A INTMEM	0x44u8	2	1	Compare A and INTMEM
MUL A INTMEM	0x45u8	2	1	A * INTMEM, low(U32) → A, high(U32) → INTMEM
DEC INTMEM	0x46u8	2	1	Decrement INTMEM
INC INTMEM	0x47u8	2	1	Increment INTMEM
AND A INTMEM	0x4Cu8	2	1	A ← A AND INTMEM
OR A INTMEM	0x4Du8	2	1	A ← A OR INTMEM
XOR A INTMEM	0x4Eu8	2	1	A ← A XOR INTMEM
MOV A INTMEM	0x50u8	2	1	Move A to INTMEM
POP A	0x5100	2	1	Pop value from Stack to A
MOVI INTMEM A	0x52u8	2	1	Move A to Indirect INTMEM. (See below)
LOAD A U16	0x6000	2	2	Load 16-bit value into A
LOAD INTMEM U16	0x61u8	2	2	Load 16-bit value into INTMEM
MOVI A INTMEM	0x80u8	3	1	Move Indirect INTMEM to A. (See below)
MOVXI INTMEM A	0x82u8	3	1	Move A (via pointer in INTMEM) to EXTMEM
MOVXI A INTMEM	0x83u8	3	1	Move EXTMEM (via pointer in INTMEM) to A
RETS	0x8400	3	1	Return from Subroutine
RETI	0x8500	3	1	Return from Interrupt
JMPD U16	0xA000	3	2	Jump to Direct 16-bit EXTMEM address
CALLD U16	0xA100	3	2	Call subroutine, Direct 16-bit EXTMEM address
JMPI INTMEM	0xA2u8	3	1	Jump Indirect via pointer in INTMEM
CALLI INTMEM	0xA3u8	3	1	Call subroutine, Indirect via pointer in INTMEM
BRCC S8	0xC0s8	2/1	1	Branch if CARRY flag is '0'
BROC S8	0xC1s8	2/1	1	Branch if OV flag is '0'
BRNC S8	0xC2s8	2/1	1	Branch if NEG flag is '0'
BRZC S8	0xC3s8	2/1	1	Branch if ZERO flag is '0'
BRCS S8	0xC4s8	2/1	1	Branch if CARRY flag is '1'
BROS S8	0xC5s8	2/1	1	Branch if OV flag is '1'
BRNS S8	0xC6s8	2/1	1	Branch if NEG flag is '1'
BRZS S8	0xC7s8	2/1	1	Branch if ZERO flag is '0'

ADD A INTMEM	OP-Code	Arg	Words	Cycles
	0x43*u8*	-	1	2

Function:

ACCU ← ACCU + INTMEM(u8);

PC ← PC + 1

Affected Flags: ZERO, NEG, OVER, CARRY

Description:

Add ACCU and INTMEM.

The contents of ACCU and INTMEM (addressed by the lower 8 bits of the OP-Code) are added and the result is stored into the ACCU.

The flags are affected as follows:

- ZERO set when the result is zero, otherwise cleared
- NEG set when d15 of result is one, otherwise cleared
- OVER set when sign overflows, otherwise cleared
- CARRY set when result is bigger than 16 bits, otherwise cleared

Remarks:

This instruction does **not** add the current CARRY to the two summands. It can be used for a 16-bit addition or to add the lowest two words in a higher precision (e.g. 64-bit) addition.

Assembler example:

```
ADD A 0x20      ; adds INTMEM location 0x20 to the Accumulator
                ; assembles to : 0x4320
```

ADDC A INTMEM

OP-Code	Arg	Words	Cycles
0x41*u8*	-	1	2

Function:

ACCU ← ACCU + INTMEM(u8) + CARRY;

PC ← PC + 1

Affected Flags: ZERO, NEG, OVER, CARRY

Description:

ADD CARRY, ACCU and INTMEM.

The contents of ACCU, INTMEM (addressed by the lower 8 bits of the OP-Code) and CARRY flag are added and the result is stored into ACCU.

The flags are affected as follows:

- ZERO set when the result is zero, otherwise cleared
- NEG set when d15 of result is one, otherwise cleared
- OVER set when sign overflows, otherwise cleared
- CARRY set when result is bigger than 16 bits, otherwise cleared

Remarks:

ADDC is normally used in higher precision (e.g. 64-bit) additions. At first the lowest two words are added with the ADD instruction. Then the higher words are added with the ADDC instruction, taking the carries into account.

Assembler example:

```
ADDC A 0x20     ; adds Carry and INTMEM location 0x20 to the Accumulator
                ; assembles to : 0x4120
```

AND A INTMEM

OP-Code	Arg	Words	Cycles
0x4C*u8*	-	1	2

Function:

ACCU ← ACCU **AND** INTMEM(u8)

PC ← PC + 1

Affected Flags: ZERO, NEG, OVER

Description:

And ACCU and INTMEM.

The contents of ACCU and INTMEM (addressed by the lower 8 bits of the OP-Code) are ANDed bit-wise and the result is stored into the ACCU.

The flags are affected as follows:

- ZERO set when the result is zero, otherwise cleared
- NEG set when d15 of result is one, otherwise cleared
- OVER set to zero

Remarks:

none

Assembler example:

```
AND A 0x20      ; logical and of INTMEM location 0x20 and the Accumulator
                ; assembles to : 0x4C20
```

BRxx	OP-Code	Arg	Words	Cycles
	0xCx*s8*	-	1	2/1

Function:

Conditional branch

If condition is false then PC $\leftarrow$ PC + 1

If condition is true then PC $\leftarrow$ PC + s8 (branch)

Description:

Branch on (flag) clear / set.

If the condition is not met, the program simply continues with the next instruction. This takes one cycle.

If the condition is met, the offset (sign extended lower half of the OP-Code) is added to the program counter (PC). At this moment the PC is already pointing to the instruction following the branch instruction. So an offset of 0 will act as if no branch was taken. An offset of -1 will set the PC back to the branch instruction itself, thus creating an infinite loop. A branch reaches from -127 to +128 of its own address and takes two cycles.

x = branch taken

FLAG	CARRY		NEG		OVER		ZERO	
Mnemonic (Opcode)	0	1	0	1	0	1	0	1
BRCC (C0s8)	x							
BRCS (C4s8)		x						
BRNC (C2s8)			x					
BRNS (C6s8)				x				
BROC (C1s8)					x			
BROS (C5s8)						x		
BRZC (C3s8)							x	
BRZS (C7s8)								x

Remarks: none

Assembler example:

```
BRCC 0x1040     ; branches to EXTMEM address 0x1040 if CARRY is clear
                ; assembly depends on the instructions own address
                ; if BRCC is at location 0x1000 it assembles to 0xC03F
                ; if BRCC is at location 0x1080 it assembles to 0xC0BF
                ; the target address must always be the EXTMEM address
                ; the sign extended offset is calculated by the assembler
                ; the assembler also checks the range -127 to +128
```

CALLD U16

OP-Code	Arg	Words	Cycles
0xA100	U16	2	3

Function:

INTMEM(SP) ← PC

PC ← U16

Affected Flags: none

Description:

Call direct address.

The call address is the argument, i.e. the word following the opcode. First the return address, which is the address of the instruction following the argument, is pushed onto the stack. Then the PC is loaded with the call address and execution continues there.

Remarks:

One word of stack space must be available

Assembler example:

```
CALLD 0x1234    ; calls subroutine at EXTMEM location 0x1234
                ; assembles to 0xA100 0x1234
```

CALLI INTMEM

OP-Code	Arg	Words	Cycles
0xA3*u8*	-	1	3

Function:

INTMEM(SP) ← PC+1

PC ← INTMEM(u8)

Affected Flags: none

Description:

Call indirect via pointer in INTMEM.

The call address is in INTMEM (which is addressed by the lower 8 bits of the OP-Code). First the return address, which is the address of the instruction following the CALL, is pushed onto the stack. Then the PC is loaded with the call address and execution continues there.

Remarks:

One word of stack space must be available

Assembler example:

```
CALLI 0x12      ; calls subroutine at EXTMEM address that is contained
                ; in INTMEM location 0x12
                ; assembles to 0xA312
```

CLR C

OP-Code	Arg	Words	Cycles
0x1000	-	1	1

Function:

CARRY ← 0

PC ← PC + 1

Affected Flags: Carry

Description:

Clear CARRY.

The CARRY flag is set to zero. This can be used to prepare shift and rotate instructions.

Remarks:

none

Assembler example:

```
CLR C          ; clears CARRY flag
               ; assembles to 0x1000
```

CLR IE

OP-Code	Arg	Words	Cycles
0x0200	-	1	1

Function:

IE ← 0

PC ← PC + 1

Affected Flags: IE

Description:

Clear Interrupt Enable flag.

The Interrupt Enable flag is set to zero. Interrupts already in progress will not be affected. Pending interrupts will stay pending but will not be taken until IE is set to one.

Remarks:

none

Assembler example:

```
CLR IE          ; clears interrupt enable flag which disables all interrupts
                ; assembles to 0x0200
```

CLR INT

OP-Code	Arg	Words	Cycles
0x0400	-	1	1

Function:

IRRx ← 0

PC ← PC + 1

Affected Flags: none

Description:

Clear Interrupts.

All Interrupt Request Registers that hold pending interrupts are cleared to zero thus cancelling the requests.

Remarks:

none

Assembler example:

```
CLR INT        ; clears all pending interrupts
               ; assembles to 0x0400
```

CMP A INTMEM

OP-Code	Arg	Words	Cycles
0x4A*u8*	-	1	2

Function:

(Discarded) ← ACCU - INTMEM(u8);

PC ← PC + 1

Affected Flags: ZERO, NEG, OVER, CARRY

Description:

Compare ACCU and INTMEM.

The content of INTMEM (addressed by the lower 8 bits of the OP-Code) is subtracted from the ACCU. The result is discarded (not written to the ACCU) but the flags are set like in a normal SUB instruction. The flags are affected as follows:

- ZERO: set when the result is zero, otherwise cleared
- NEG: set when d15 of result is one, otherwise cleared
- OVER: set when result overflows into d15, otherwise cleared
- CARRY: (=borrow) set when result is less then zero, otherwise cleared

The ZERO and the CARRY flag show the result of the comparison

ZERO	CARRY	Comparison result
1	x	A = INTMEM(u8)
0	x	A ≠ INTMEM(u8)
0	0	A > INTMEM(u8)
0	1	A < INTMEM(u8)

Remarks:

none

Assembler example:

```
CMP A 0x20        ; compares INTMEM location 0x20 and the Accumulator
                  ; assembles to 0x4A20
```

DEC INTMEM

OP-Code	Arg	Words	Cycles
0x46*u8*	-	1	2

Function:

INTMEM(u8) ← INTMEM(u8) - 1;

PC ← PC + 1

Affected Flags: ZERO, NEG, OVER, CARRY

Description:

Decrement INTMEM.

The content of INTMEM (addressed by the lower 8 bits of the OP-Code) is decremented by one and stored back to the same location.

The flags are affected as follows:

ZERO:	set when the result is zero, otherwise cleared
NEG:	set when d15 of result is one, otherwise cleared
OVER:	set to zero
CARRY:	(=borrow) set when result is less then zero, otherwise cleared

Remarks:

This can be used for block transfers where the INTMEM serves as address pointer or as a loop counter.

Assembler example:

```
DEC 0x20        ; decrements value in INTMEM location 0x20 by one
                ; assembles to 0x4620
```

INC INTMEM

OP-Code	Arg	Words	Cycles
0x47*u8*	-	1	2

Function:

INTMEM(u8) ← INTMEM(u8) + 1;

PC ← PC + 1

Affected Flags: ZERO, NEG, OVER, CARRY

Description:

Increment INTMEM.

The content of INTMEM (addressed by the lower 8 bits of the OP-Code) is incremented by one and stored back to the same location.

The flags are affected as follows:

ZERO:	set when the result is zero, otherwise cleared
NEG:	set when d15 of result is one, otherwise cleared
OVER:	set to zero
CARRY:	set when result is bigger than 16 bits, otherwise cleared

Remarks:

This can be used for block transfers where the INTMEM serves as address pointer or as a loop counter.

Assembler example:

```
INC 0x20          ; increments value in INTMEM location 0x20 by one
                  ; assembles to 0x4720
```

JMPD U16

OP-Code	Arg	Words	Cycles
0xA000	U16	2	3

Function:

PC $\leftarrow$ U16

Affected Flags: none

Description:

Jump directly to address.

The jump address is the argument, i.e. the word following the opcode. The PC is loaded with the jump address and execution continues there.

Remarks:

none

Assembler example:

```
JMPD 0x1234        ; jumps to EXTMEM address 0x1234
                   ; assembles to 0xA000 followed by 0x1234
```

JMPI INTMEM

OP-Code	Arg	Words	Cycles
0xA2*u8*	-	1	3

Function:

PC $\leftarrow$ INTMEM(u8)

Affected Flags: none

Description:

Jump indirect via pointer in INTMEM.

The jump address is in INTMEM (which is addressed by the lower 8 bits of the OP-Code).

The jump address is loaded into the PC and execution continues there.

Remarks:

none

Assembler example:

```
JMPI 0x12       ; jumps to EXTMEM address that is contained
                ; in INTMEM location 0x12
                ; assembles to 0xA212
```

MOV INTMEM A

OP-Code	Arg	Words	Cycles
0x16*u8*	-	1	1

Function:

INTMEM(u8) ← A

PC ← PC + 1

Affected Flags: none

Description:

Move ACCU to INTMEM.

Copy ACCU to INTMEM (which is addressed by the lower 8 bits of the OP-Code)

Remarks:

none

Assembler example:

```
MOV 0x12 A      ; copy value in Accu to INTMEM location 0x12
                ; assembles to 0x1612
```

MOV A INTMEM	OP-Code	Arg	Words	Cycles
	0x50*u8*	-	1	2

Function:

A ← INTMEM(u8)

PC ← PC + 1

Affected Flags: none

Description:

Move INTMEM to ACCU.

Copy INTMEM (addressed by the lower 8 bits of the OP-Code) to ACCU.

Remarks:

none

Assembler example:

```
MOV A 0x12        ; copy value in INTMEM location 0x12 to Accu
                  ; assembles to 0x5012
```

MOV A STATUS

OP-Code	Arg	Words	Cycles
0x1400	-	1	1

Function:

A ← STATUS

PC ← PC + 1

Affected Flags: none

Description:

Move STATUS to ACCU.

The STATUS register reflects the stack pointer and the flags

d15 - d8	d7 - d5	d4	d3	d2	d1	d0
STACK POINTER	000	IE	ZERO	NEG	OVER	CARRY

Remarks:

none

Assembler example:

```
MOV A STATUS    ; copy value in stack pointer, IE and flags to Accu
                ; assembles to 0x1400
```

MOV FLAGS A

OP-Code	Arg	Words	Cycles
0x1200	-	1	1

Function:

FLAGS $\leftarrow$ A

PC $\leftarrow$ PC + 1

Description:

Move ACCU to FLAGS.

Copy the 5 least significant bits of ACCU to the flag bits.

With this instruction all flags can be initialized to a specific value.

d15 - d5	d4	d3	d2	d1	d0
- - - - - - - - - - -	IE	ZERO	NEG	OVER	CARRY

Remarks:

Flags IE and CARRY can also be set and cleared with their dedicated instructions.

The stack pointer can only be loaded with its dedicated instruction (MOV SP A)

Assembler example:

```
MOV FLAGS A     ; copy value in Accu to IE and flags
                ; assembles to 0x1200
```

MOV IMASK A

OP-Code	Arg	Words	Cycles
0x0500	-	1	1

Function:

IMASK ← A

PC ← PC + 1

Affected Flags: none

Description:

Move ACCU to IMASK.

The number of bits in the IMASK register depends on the interrupt hardware but they always start at d0 and are arranged gapless. If there are e.g. four interrupt lines, the mask bits are d0 – d3 with d0 (IRQ0) having the highest and d3 (IRQ3) the lowest priority.

For an interrupt to be enabled its corresponding IMASK bit must be set to one.

Remarks:

The IMASK register is write only

Assembler example:

```
MOV IMASK A     ; copy value in Accu to IMASK
                ; assembles to 0x0500
```

MOV SP A	OP-Code	Arg	Words	Cycles
	0x1300	-	1	1

Function:

SP ← A

PC ← PC + 1

Affected Flags: none

Description:

Move ACCU to SP.

Copy lower 8 bits of ACCU to stack pointer

Remarks:

To read back the content of register SP use the instructions MOV A STATUS and SWAP ACCU.

Assembler example:

```
MOV SP A        ; copy value in Accu to stackpointer
                ; assembles to 0x1300
```

LOAD INTMEM U16

OP-Code	Arg	Words	Cycles
0x61*u8*	U16	2	2

Function:

INTMEM(u8) ← U16

PC ← PC + 2

Affected Flags: none

Description:

Load 16-bit value into INTMEM.

INTMEM is addressed by the lower 8 bits of the OP-Code

Remarks:

none

Assembler example:

```
LOAD 0x12 0x55AA        ; move value 0x55AA into INTMEM location 0x12
                        ; assembles to 0x6112 followed by 0x55AA
```

LOAD A U16

OP-Code	Arg	Words	Cycles
0x6000	U16	2	2

Function:

A $\leftarrow$ U16

PC $\leftarrow$ PC + 2

Affected Flags: none

Description:

Load 16-bit value into ACCU.

Remarks:

none

Assembler example:

```
LOAD A 0x55AA        ; move value 0x55AA into Accu
                     ; assembles to 0x6000 followed by 0x55AA
```

MOVI INTMEM A

OP-Code	Arg	Words	Cycles
0x52*u8*	-	1	2

Function:

INTMEM ((low) INTMEM (u8)) ← A

PC ← PC + 1

Affected Flags: none

Description:

Move indirect ACCU to INTMEM.

The INTMEM argument is not the target address itself but a pointer to it. At first this pointer is fetched from INTMEM. Its lower 8 bits then address the actual INTMEM target location where the content of ACCU is copied to. The pointers upper 8 bits are ignored. This addressing mode is useful for moving and/or manipulating blocks of INTMEM in a loop by simply incrementing or decrementing the pointer.

Remarks:

none

Assembler example:

```
MOVI 0x12 A      ; copy value in Accu to INTMEM location whose address is contained
                 ; in INTMEM location 0x12
                 ; assembles to 0x5212
```

MOVI A INTMEM

OP-Code	Arg	Words	Cycles
0x80*u8*	-	1	3

Function:

A ← INTMEM ((low) INTMEM (u8))

PC ← PC + 1

Affected Flags: none

Description:

Move indirect INTMEM to ACCU.

The INTMEM argument is not the source address itself but a pointer to it. At first this pointer is fetched from INTMEM. Its lower 8 bits then address the actual INTMEM source location whose content is copied to the ACCU. The pointers upper 8 bits are ignored. This addressing mode is useful for moving and/or manipulating blocks of INTMEM in a loop by simply incrementing or decrementing the pointer.

Remarks:

none

Assembler example:

```
MOVI A 0x12        ; copy value of INTMEM  location whose address is contained
                   ; in INTMEM location 0x12 to Accu
                   ; assembles to 0x8012
```

MOVXI INTMEM A

OP-Code	Arg	Words	Cycles
0x82*u8*	-	1	3

Function:

EXTMEM (INTMEM (u8)) ← A

PC ← PC + 1

Affected Flags: none

Description:

Move ACCU to EXTMEM addressed by pointer in INTMEM.
At first the INTMEM is addressed by the lower 8 bits of the OP-Code. This memory location serves as a pointer. It points to the actual EXTMEM target address where the content of ACCU is copied to. The indirect addressing can be used to move and/or manipulate blocks of EXTMEM in a loop by incrementing or decrementing the pointer.

Remarks:

none

Assembler example:

```
MOVXI 0x12 A      ; copy value in Accu to EXTMEM location whose address is contained
                  ; in INTMEM location 0x12
                  ; assembles to 0x8212
```

MOVXI A INTMEM

OP-Code	Arg	Words	Cycles
0x83*u8*	-	1	3

Function:

A ← EXTMEM (INTMEM (u8))

PC ← PC + 1

Affected Flags: none

Description:

Move EXTMEM addressed by INTMEM to ACCU.

At first the INTMEM is addressed by the lower 8 bits of the OP-Code. This memory location serves as a pointer. It points to the actual EXTMEM location whose content is copied to ACCU. The indirect addressing can be used to move and/or manipulate blocks of EXTMEM in a loop by incrementing or decrementing the pointer.

Remarks:

none

Assembler example:

```
MOVI A 0x12        ; copy value of  EXTMEM  location whose address is contained
                   ; in INTMEM location 0x12 to Accu
                   ; assembles to 0x8312
```

MUL A INTMEM	OP-Code	Arg	Words	Cycles
	0x45u8	-	1	2

Function:

A ← low (A * INTMEM), INTMEM ← high (A * INTMEM)

PC ← PC + 1

Affected Flags: none

Description:

Multiply ACCU and INTMEM.

The contents of ACCU and INTMEM (addressed by the lower 8 bits of the OP-Code) are multiplied as U16. The result is a U32. The lower 16 bit of the result are stored into the ACCU, the higher 16 bit are stored into the INTMEM cell, overwriting the original content. No flags are affected.

Remarks:

none

Assembler example:

```
MUL A 0x12      ; multiply Accu and INTMEM  cell whose address is contained
                ; assembles to 0x4512
```

NOP	OP-Code	Arg	Words	Cycles
	0x0000	-	1	1

Function:

PC ← PC + 1

Affected Flags: none

Description:

No operation.

Only the PC is incremented by one. This instruction can be used for short delays or to fill unused memory.

Remarks:

none

Assembler example:

```
NOP            ;  No operation
               ; assembles to 0x0000
```

NOT A

OP-Code	Arg	Words	Cycles
0x2F00	-	1	1

Function:

ACCU ← **NOT** ACCU

PC ← PC + 1

Affected Flags: ZERO, NEG, OVER

Description:

Negate ACCU.

All bits of ACCU are negated and the result is stored into ACCU.

The flags are affected as follows:

- ZERO set when the result is zero, otherwise cleared
- NEG set when d15 of result is one, otherwise cleared
- OVER set to zero

Remarks:

Two consecutive NOT A leave the ACCU unchanged but set / clear the flags ZERO and NEG. This can be used to test if the ACCU contains zero or a negative value.

Assembler example:

```
NOT A           ;  Negates all bits of Accu
                ; assembles to 0x2F00
```

OR A INTMEM

OP-Code	Arg	Words	Cycles
0x4D*u8*	-	1	2

Function:

ACCU ← ACCU **OR** INTMEM(u8)

PC ← PC + 1

Affected Flags: ZERO, NEG, OVER

Description:

Or ACCU with INTMEM.

The contents of ACCU and INTMEM (addressed by the lower 8 bits of the OP-Code) are ORed bit-wise and the result is stored into the ACCU.

The flags are affected as follows:

- ZERO set when the result is zero, otherwise cleared
- NEG set when d15 of result is one, otherwise cleared
- OVER set to zero

Remarks:

none

Assembler example:

```
OR A 0x20        ; logical or of INTMEM location 0x20 and the Accumulator
                 ; assembles to 0x4D20
```

POP A

OP-Code	Arg	Words	Cycles
0x5100	-	1	2

Function:

SP $\leftarrow$ SP + 1

ACCU $\leftarrow$ INTMEM(SP)

PC $\leftarrow$ PC + 1

Affected Flags: none

Description:

Pop ACCU.

At first the stack pointer is incremented by one. Next the content of INTMEM (addressed by the stack pointer) is copied into ACCU.

Remarks:

none

Assembler example:

```
POP A            ; pops value from stack and puts it into Accu
                 ; assembles to 0x5100
```

PUSH A

OP-Code	Arg	Words	Cycles
0x1500	-	1	1

Function:

INTMEM(SP) ← ACCU

SP ← SP - 1

PC ← PC + 1

Affected Flags: none

Description:

Push ACCU.

At first the content of ACCU is copied into INTMEM (addressed by the stack pointer). Next the stack pointer is decremented by one.

Remarks:

none

Assembler example:

```
PUSH A          ; pushed value in Accu to stack
                ; assembles to 0x1500
```

RETI	OP-Code	Arg	Words	Cycles
	0x8500	-	1	3

Function:

SP $\leftarrow$ SP + 1

IIPx $\leftarrow$ 0

PC $\leftarrow$ INTMEM(SP)

Affected Flags: none

Description:

Return from interrupt.

At first the stack pointer is incremented by one and the execution of the current interrupt is ended by clearing the correspondent IIPx-Flag (Interrupt In Progress). Next the content of INTMEM (addressed by the stack pointer) is copied into PC and execution continues from there.

Remarks:

A valid return address must be on the stack from an interrupt call

Assembler example:

```
RETI                ; return from interrupt
                    ; assembles to 0x8500
```

RETS	OP-Code	Arg	Words	Cycles
	0x8400	-	1	3

Function:

SP ← SP + 1

PC ← INTMEM(SP)

Affected Flags: none

Description:

Return from subroutine.

At first the stack pointer is incremented by one. Next the content of INTMEM (addressed by the stack pointer) is copied into PC and execution continues from there.

Remarks:

A valid return address must be on the stack from a previous CALL instruction

Assembler example:

```
RETI                ; return from subroutine
                    ; assembles to 0x8400
```

ROL A	OP-Code	Arg	Words	Cycles
	0x2800	-	1	1

Function:

CARRY ← ACCU(15 ← 14 ← … ← 2 ← 1 ← 0) ← CARRY

PC ← PC + 1

Affected Flags: ZERO, NEG, OVER, CARRY

Description:

Rotate Left Accumulator through CARRY.

The ACCU is shiftet left one bit position. Its lowest bit receives the content of the CARRY. Its highest bit in turn goes into the CARRY.

The other flags are affected as follows:

- ZERO set when the result is zero, otherwise cleared
- NEG set when d15 of result is one, otherwise cleared
- OVER set to zero

Remarks:

The CARRY can be set or cleared beforehand to insert a one or a zero.

Assembler example:

```
ROL A        ; rotate Accu left through Carry
             ; assembles to 0x2800
```

ROR A

OP-Code	Arg	Words	Cycles
0x2900	-	1	1

Function:

CARRY → ACCU(15 → 14 → … → 2 → 1 → 0) → CARRY

PC ← PC + 1

Affected Flags: ZERO, NEG, OVER, CARRY

Description:

Rotate Right Accumulator through CARRY.

The ACCU is shiftet right one bit position. Its highest bit receives the content of the CARRY. Its lowest bit in turn goes into the CARRY.

The other flags are affected as follows:

- ZERO set when the result is zero, otherwise cleared
- NEG set when d15 of result is one, otherwise cleared
- OVER set to zero

Remarks:

The CARRY can be set or cleared beforehand.

Assembler example:

```
ROR A           ; rotate Accu right through Carry
                ; assembles to 0x2900
```

SET C

OP-Code	Arg	Words	Cycles
0x1100	-	1	1

Function:

CARRY ← 1

PC ← PC + 1

Affected Flags: CARRY

Description:

Set CARRY flag.

This can be used to prepare shift and rotate instructions.

Remarks:

none

Assembler example:

```
SET C          ; Sets CARRY flag
               ; assembles to 0x1100
```

# SET IE	OP-Code	Arg	Words	Cycles
	0x0300	-	1	1

Function:

IE ← 1

PC ← PC + 1

Affected Flags: IE

Description:

Set Interrupt Enable flag.

All pending and enabled interrupts will be accepted in the order of their priority.

(INT0 = highest priority)

Remarks:

none

Assembler example:

```
SET IE          ; Sets IE flag
                ; assembles to 0x0300
```

SHL A

OP-Code	Arg	Words	Cycles
0x2A00	-	1	1

Function:

CARRY ← ACCU(15 ← 14 ← … ← 2 ← 1 ← 0) ← 0

PC ← PC + 1

Affected Flags: ZERO, NEG, OVER, CARRY

Description:

Shift Left Accumulator into CARRY.

The ACCU is shiftet left one bit position. Its lowest bit receives a zero. Its highest bit in turn goes into the CARRY.

The other flags are affected as follows:

- ZERO set when the result is zero, otherwise cleared
- NEG set when d15 of result is one, otherwise cleared
- OVER set to zero

Remarks:

SHL A is equivalent to multiply by 2

Assembler example:

```
SHL A             ; shift Accu left into Carry
                  ; assembles to 0x2A00
```

SHR A	OP-Code	Arg	Words	Cycles
	0x2B00	-	1	1

Function:

0 → ACCU(15 → 14 → … → 2 → 1 → 0) → CARRY

PC ← PC + 1

Affected Flags: ZERO, NEG, OVER, CARRY

Description:

Shift Right Accumulator into CARRY.

The ACCU is shiftet right one bit position. Its highest bit receives a zero. Its lowest bit in turn goes into the CARRY.

The other flags are affected as follows:

- ZERO set when the result is zero, otherwise cleared
- NEG set when d15 of result is one, otherwise cleared
- OVER set to zero

Remarks:

SHR A is equivalent to division by 2

Assembler example:

```
SHR A        ; shift Accu right into Carry
             ; assembles to 0x2B00
```

SUB A INTMEM

OP-Code	Arg	Words	Cycles
0x42*u8*	-	1	2

Function:

ACCU ← ACCU - INTMEM(u8);

PC ← PC + 1

Affected Flags: ZERO, NEG, OVER, CARRY

Description:

The content of INTMEM (addressed by the lower 8 bits of the OP-Code) is subtracted from ACCU and the result is stored into ACCU.

The flags are affected as follows:

- ZERO set when the result is zero, otherwise cleared
- NEG set when d15 of result is one, otherwise cleared
- OVER set when sign overflows, otherwise cleared
- CARRY (=borrow) set when result is less then zero, otherwise cleared

Remarks:

This instruction does **not** subtract the current CARRY. It can be used for a 16-bit subtraction or to subtract the lowest two words in a higher precision (e.g. 64-bit) subtraction.

Assembler example:

```
SUB A 0x20      ; subtracts INTMEM location 0x20 from Accu
                ; assembles to : 0x4220
```

SUBB A INTMEM

OP-Code	Arg	Words	Cycles
0x40*u8*	-	1	2

Function:

ACCU ← ACCU - INTMEM(u8) - CARRY;

PC ← PC + 1

Affected Flags: ZERO, NEG, OVER, CARRY

Description:

Subtract with borrow.

The content of INTMEM (addressed by the lower 8 bits of the OP-Code) and CARRY are subtracted from ACCU and the result is stored into ACCU.

The flags are affected as follows:

- ZERO set when the result is zero, otherwise cleared
- NEG set when d15 of result is one, otherwise cleared
- OVER set when sign overflows, otherwise cleared
- CARRY (=borrow) set when result is less then zero, otherwise cleared

Remarks:

SUBB is normally used in higher precision (e.g. 64-bit) subtractions. The lowest two words are subtracted with the SUB instruction. Then the higher words are subtracted with the SUBB instruction, taking the carries (borrows) into account.

Assembler example:

```
SUBB A 0x20     ; subtracts INTMEM location 0x20 and Carry from Accu
                ; assembles to : 0x4020
```

# SWAP A	OP-Code	Arg	Words	Cycles
	0x2500	-	1	1

Function:

ACCU(15 .. 8) ← ACCU(7 .. 0), ACCU(7 .. 0) ← ACCU(15 .. 8)

PC ← PC + 1

Affected Flags: ZERO, NEG, OVER

Description:

SWAP ACCU

Swap the high and low bytes of ACCU.

The flags are affected as follows:

- ZERO set when the result is zero, otherwise cleared
- NEG set when d15 of result is one, otherwise cleared
- OVER set to zero

Remarks:

Assembler example:

```
SWAP A          ; Swap bytes d15..d8 and d7..d0 of Accu
                ; assembles to 0x2500
```

XOR A INTMEM	OP-Code	Arg	Words	Cycles
	0x4E*u8*	-	1	2

Function:

ACCU ← ACCU **XOR** INTMEM(u8)

PC ← PC + 1

Affected Flags: ZERO, NEG, OVER

Description:

The contents of ACCU and INTMEM (addressed by the lower 8 bits of the OP-Code) are XORed bit-wise and the result is stored into the ACCU.

The flags are affected as follows:

- ZERO set when the result is zero, otherwise cleared
- NEG set when d15 of result is one, otherwise cleared
- OVER set to zero

Remarks:

none

Assembler example:

```
XOR A 0x20      ; logical xor of INTMEM location 0x20 and the Accumulator
                ; assembles to 0x4E20
```