1DT301, Computer Technology I

Lecture #6,

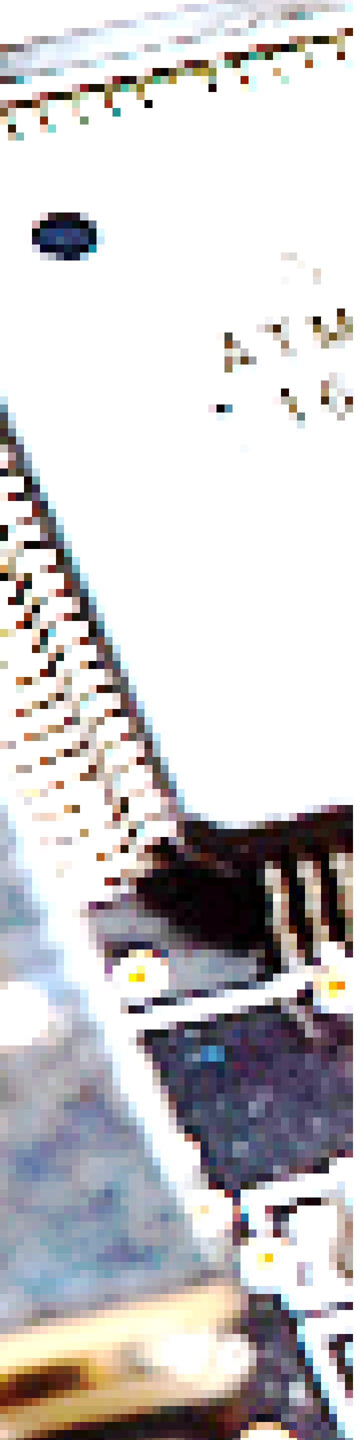
- Program example from exam
- Lab 3

5. Assembler

a) Analyze the program below and explain how the program works and what it will do. Input data in register R20. Explain what will happen with this data and where to find the result. 5p

b) Make a flowchart of the program that explains the function. 5p

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;          1ED022, Computer Technology I
;          Date: 2013-01-18
;          Anders Haggren
;          Function:
;          Exam example
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

subr_01:
          push counter                   ; Store registers on stack, counter and temp
          push temp
          clr counter                    ; counter = 0000 0000
          andi r20, 0x7F
          mov temp, r20
shift:
          lsr temp
          brcc next
          inc counter
next:
          cpi temp, 0
          brne shift

          lsr counter
          brcc end
          ori r20, 0x80

end:
          pop temp
          pop counter
          ret
```



ANDI – Logical AND with Immediate

Description:

Performs the logical AND between the contents of register Rd and a constant and places the result in the destination register Rd.

Operation:

(i) $Rd \leftarrow Rd \bullet K$

	Syntax:	Operands:	Program Counter:
(i)	ANDI Rd,K	$16 \le d \le 31$, $0 \le K \le 255$	$PC \leftarrow PC + 1$

16-bit Opcode:

0111	KKKK	dddd	KKKK

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S: $N \oplus V$, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is \$00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
    andi  r17,$0F   ; Clear upper nibble of r17
    andi  r18,$10   ; Isolate bit 4 in r18
    andi  r19,$AA   ; Clear odd bits of r19
```

Words: 1 (2 bytes)

Cycles: 1

5. Assembler

a) Analyze the program below and explain how the program works and what it will do. Input data in register R20. Explain what will happen with this data and where to find the result. 5p

b) Make a flowchart of the program that explains the function. 5p

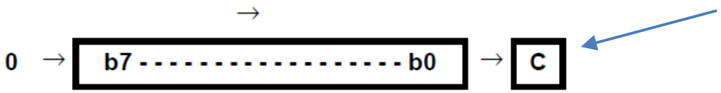
```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;          1ED022, Computer Technology I
;          Date: 2013-01-18
;          Anders Haggren
;          Function:
;          Exam example
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

subr_01:
          push counter              ; Store registers on stack, counter and temp
          push temp
          clr counter               ; counter = 0000 0000
          andi r20, 0x7F            ; mask r20 with 0111 1111
          mov temp, r20             ; result in r20
shift:
          lsr temp
          brcc next
          inc counter
next:
          cpi temp, 0
          brne shift

          lsr counter
          brcc end
          ori r20, 0x80

end:
          pop temp
          pop counter
          ret
```

LSR – Logical Shift Right

Description:

Shifts all bits in Rd one place to the right. Bit 7 is cleared. Bit 0 is loaded into the C Flag of the SREG. This operation effectively divides an unsigned value by two. The C Flag can be used to round the result.

Operation:



	Syntax:	Operands:	Program Counter:
(i)	LSR Rd	$0 \le d \le 31$	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0110

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	0	⇔	⇔

S: $N \oplus V$, For signed tests.

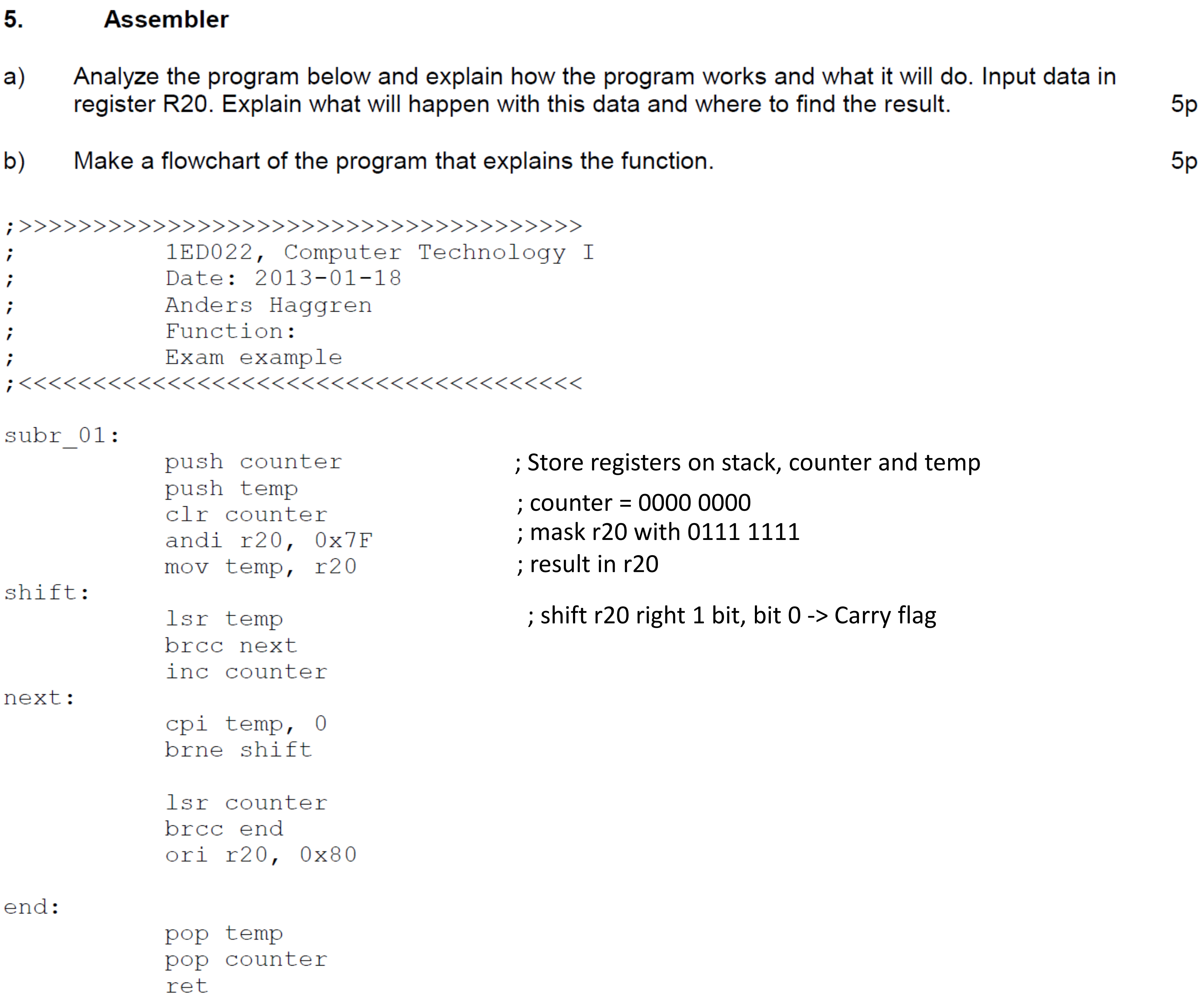
V: $N \oplus C$ (For N and C after the shift)

N: 0

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is \$00; cleared otherwise.

C: Rd0
Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.



5. Assembler

a) Analyze the program below and explain how the program works and what it will do. Input data in register R20. Explain what will happen with this data and where to find the result. 5p

b) Make a flowchart of the program that explains the function. 5p

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;          1ED022, Computer Technology I
;          Date: 2013-01-18
;          Anders Haggren
;          Function:
;          Exam example
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

subr_01:
          push counter              ; Store registers on stack, counter and temp
          push temp
          clr counter               ; counter = 0000 0000
          andi r20, 0x7F            ; mask r20 with 0111 1111
          mov temp, r20             ; result in r20
shift:
          lsr temp                  ; shift r20 right 1 bit, bit 0 -> Carry flag
          brcc next
          inc counter
next:
          cpi temp, 0
          brne shift

          lsr counter
          brcc end
          ori r20, 0x80

end:
          pop temp
          pop counter
          ret
```

BRCC – Branch if Carry Cleared

Description:

Conditional relative branch. Tests the Carry Flag (C) and branches relatively to PC if C is cleared. This instruction branch relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is rep sented in two's complement form. (Equivalent to instruction BRBC 0,k).

Operation:

(i) If C = 0 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	**Operands:**	**Program Counter:**
(i)	BRCC k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k000

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
          add   r22,r23     ; Add r23 to r22
          brcc  nocarry     ; Branch if carry cleared
          ...
nocarry:  nop               ; Branch destination (do nothing)
```

5. Assembler

a) Analyze the program below and explain how the program works and what it will do. Input data in register R20. Explain what will happen with this data and where to find the result. 5p

b) Make a flowchart of the program that explains the function. 5p

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;          1ED022, Computer Technology I
;          Date: 2013-01-18
;          Anders Haggren
;          Function:
;          Exam example
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

subr_01:
          push counter                  ; Store registers on stack, counter and temp
          push temp
          clr counter                   ; counter = 0000 0000
          andi r20, 0x7F                ; mask r20 with 0111 1111
          mov temp, r20                 ; result in r20
shift:
          lsr temp                       ; shift r20 right 1 bit, bit 0 -> Carry flag
          brcc next                      ; check if Carry flag = 0, if = 0 jump to next
          inc counter                    ; if Carry  flag = 1, increase counter
next:
          cpi temp, 0                    ; check if temp = 0
          brne shift                     ; temp not 0, jump to shift

          lsr counter                     ; temp = 0, shift counter 1 step right, bit 0 to carry
          brcc end                        ; if Carry flag = 0, jump to end
          ori r20, 0x80                   ; if carry flag =1, set bit 7 i r20

end:
          pop temp                        ; Reset registers from stack, counter and temp
          pop counter                     ; Return value in   Register r20
          ret
```

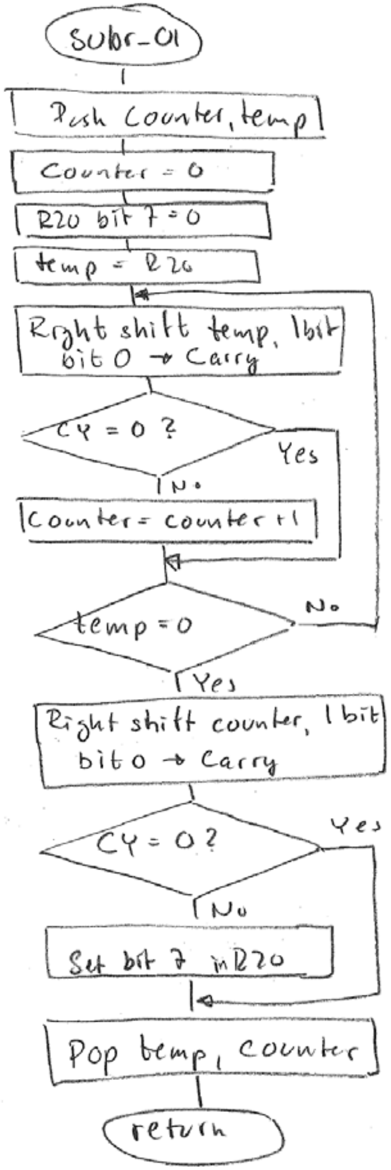
5. Assembler

a) Analyze the program below and explain how the program works and what it will do. Input data in register R20. Explain what will happen with this data and where to find the result. 5p

b) Make a flowchart of the program that explains the function. 5p

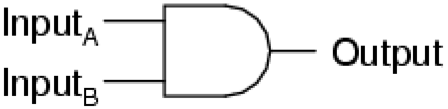
```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;          1ED022, Computer Technology I
;          Date: 2013-01-18
;          Anders Haggren
;          Function:
;          Exam example
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

subr_01:
          push counter
          push temp
          clr counter
          andi r20, 0x7F
          mov temp, r20
shift:
          lsr temp
          brcc next
          inc counter
next:
          cpi temp, 0
          brne shift

          lsr counter
          brcc end
          ori r20, 0x80

end:
          pop temp
          pop counter
          ret
```

The program counts the number of ones in r20, bit 0 -6. If the number is odd, the bit 7 in r20 will be set to one, otherwise bit 7 will remain zero. The program is thus checking parity and sets the parity bit to Even parity.



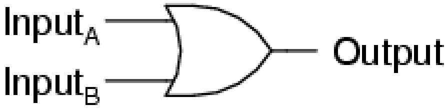
AND, OR, XOR

2-input AND gate



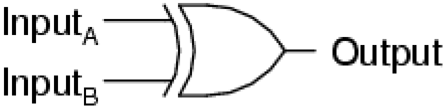
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR gate

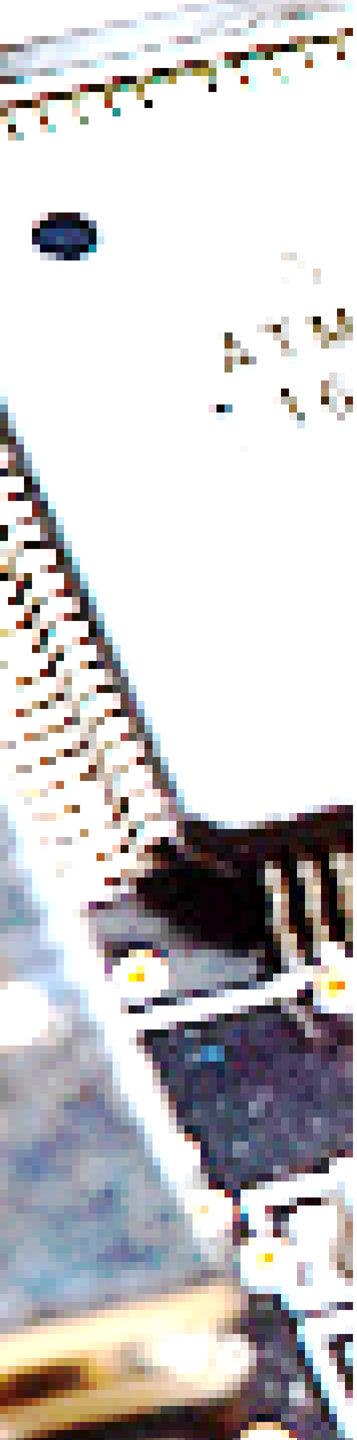


A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

Exclusive-OR gate



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0



AND – Logical AND

Description:

Performs the logical AND between the contents of register Rd and register Rr and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd • Rr

	Syntax:	Operands:	Program Counter:
(i)	AND Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0010	00rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S: N ⊕ V, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Set if the result is $00; cleared otherwise.

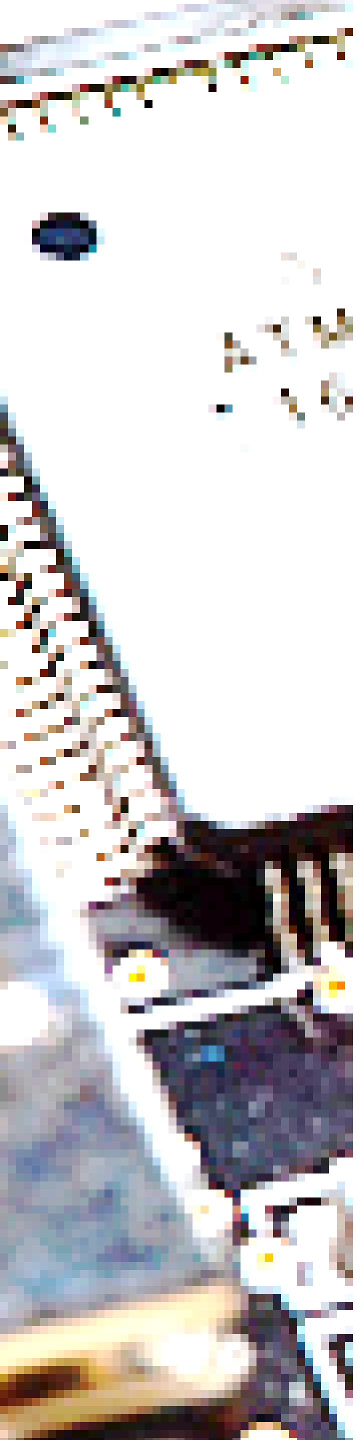
R (Result) equals Rd after the operation.

Example:

```
    and   r2,r3     ; Bitwise and r2 and r3, result in r2
    ldi   r16,1     ; Set bitmask 0000 0001 in r16
    and   r2,r16    ; Isolate bit 0 in r2
```

Words: 1 (2 bytes)

Cycles: 1



OR – Logical OR

Description:

Performs the logical OR between the contents of register Rd and register Rr and places the result in the destination register Rd.

	Operation:
(i)	Rd ← Rd v Rr

	Syntax:	Operands:	Program Counter:
(i)	OR Rd,Rr	$0 \le d \le 31$, $0 \le r \le 31$	PC ← PC + 1

16-bit Opcode:

0010	10rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S: $N \oplus V$, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7}\bullet \overline{R6} \bullet\overline{R5}\bullet \overline{R4}\bullet \overline{R3} \bullet\overline{R2}\bullet \overline{R1}\bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

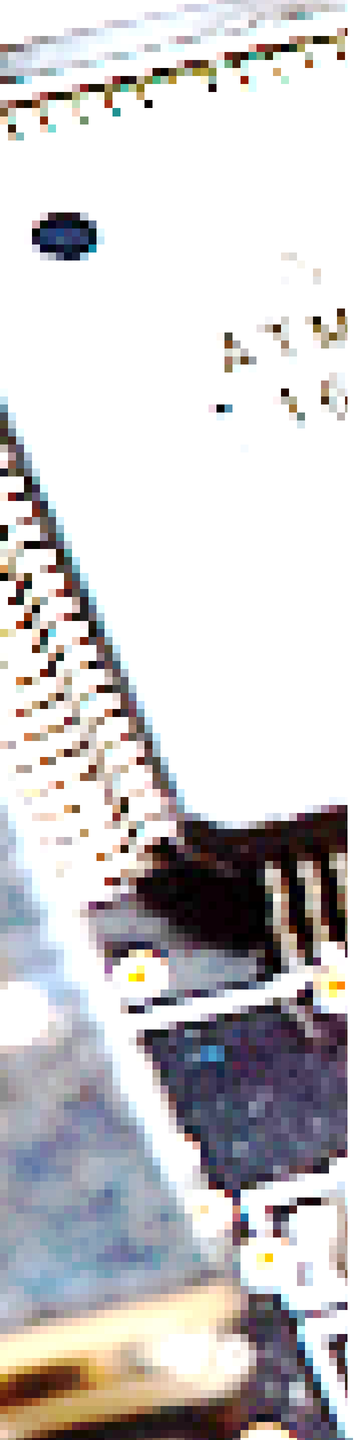
R (Result) equals Rd after the operation.

Example:

```
        or      r15,r16     ; Do bitwise or between registers
        bst     r15,6       ; Store bit 6 of r15 in T Flag
        brts    ok          ; Branch if T Flag set
        ...
    ok: nop                 ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1



EOR – Exclusive OR

Description:

Performs the logical EOR between the contents of register Rd and register Rr and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd ⊕ Rr

	Syntax:	Operands:	Program Counter:
(i)	EOR Rd,Rr	$0 \le d \le 31$, $0 \le r \le 31$	PC ← PC + 1

16-bit Opcode:

0010	01rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S: N ⊕ V, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is \$00; cleared otherwise.

R (Result) equals Rd after the operation.

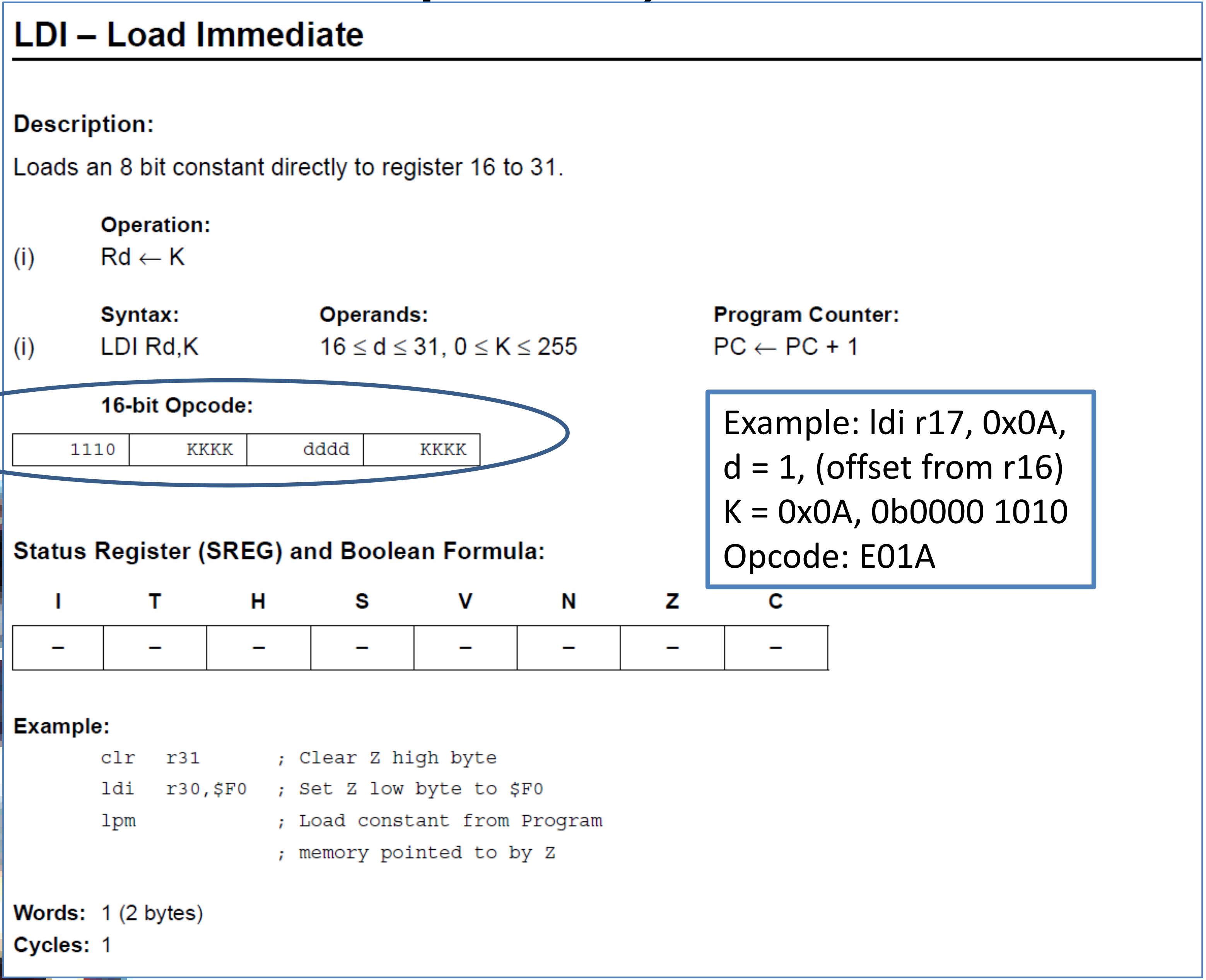
Example:

```
eor     r4,r4      ; Clear r4
eor     r0,r22     ; Bitwise exclusive or between r0 and r22
```

Words: 1 (2 bytes)

Cycles: 1

16-bit Opcode, ”hex code”



LDI – Load Immediate

Description:

Loads an 8 bit constant directly to register 16 to 31.

Operation:

(i) Rd ← K

	Syntax:	Operands:	Program Counter:
(i)	LDI Rd,K	$16 \leq d \leq 31$, $0 \leq K \leq 255$	PC ← PC + 1

16-bit Opcode:

1110	KKKK	dddd	KKKK

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

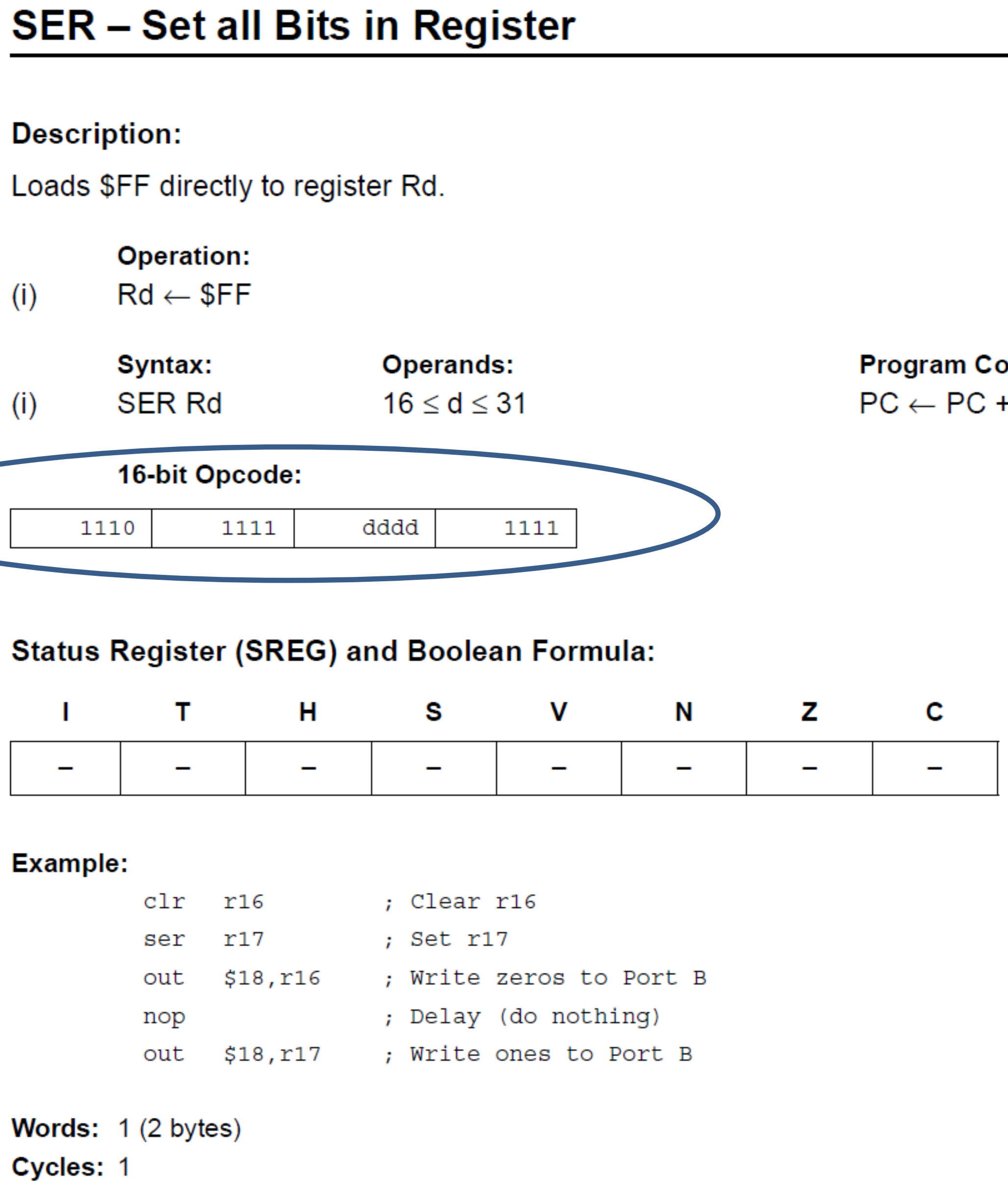
```
clr   r31        ; Clear Z high byte
ldi   r30,$F0    ; Set Z low byte to $F0
lpm              ; Load constant from Program
                 ; memory pointed to by Z
```

Words: 1 (2 bytes)

Cycles: 1

Example: ldi r17, 0x0A,
d = 1, (offset from r16)
K = 0x0A, 0b0000 1010
Opcode: E01A

16-bit Opcode, ”hex code”



SER – Set all Bits in Register

Description:

Loads $FF directly to register Rd.

Operation:

(i) Rd ← $FF

	Syntax:	**Operands:**	**Program Counter:**
(i)	SER Rd	16 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1110	1111	dddd	1111

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
    clr   r16          ; Clear r16
    ser   r17          ; Set r17
    out   $18,r16      ; Write zeros to Port B
    nop                ; Delay (do nothing)
    out   $18,r17      ; Write ones to Port B
```

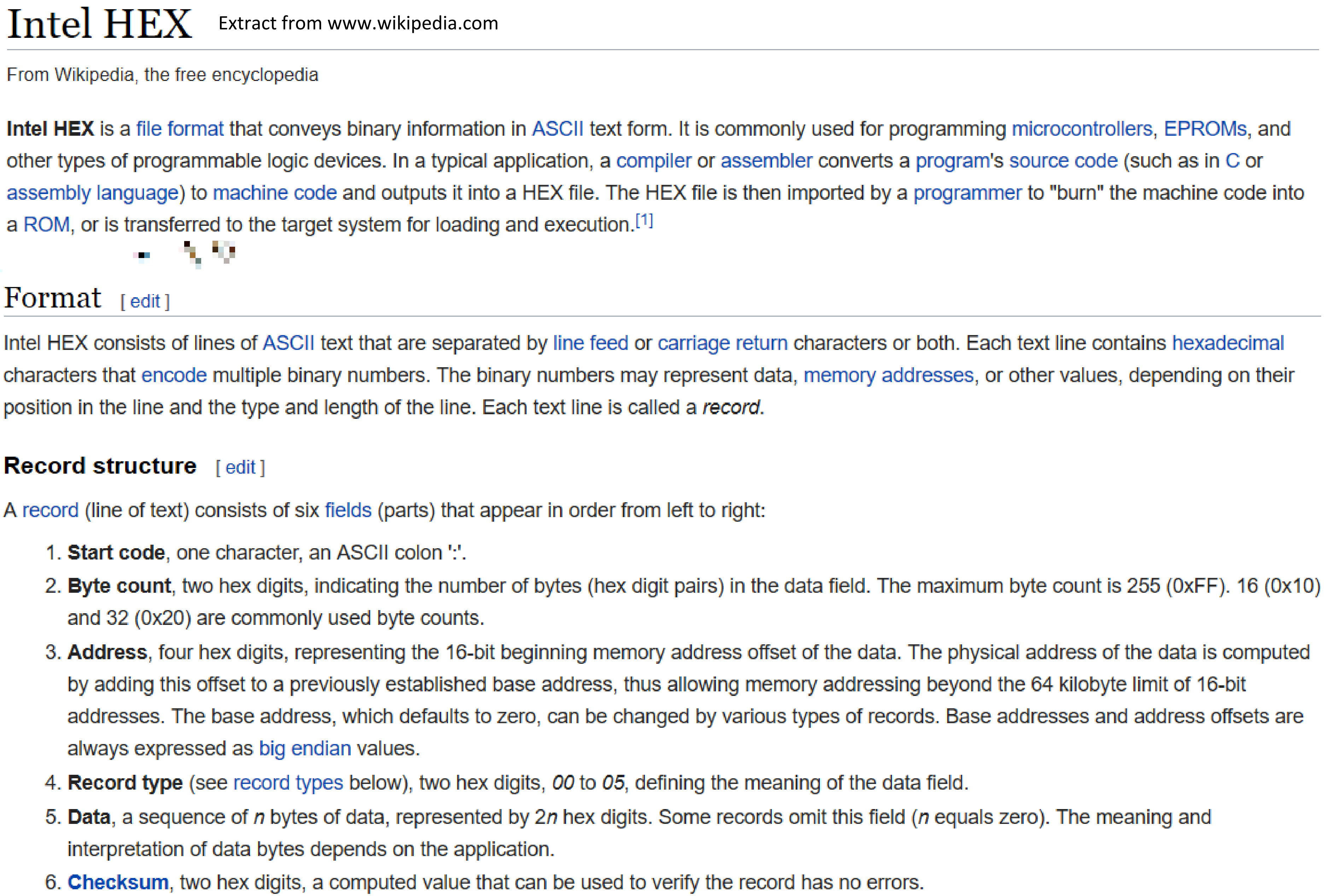
Words: 1 (2 bytes)

Cycles: 1

Example: *ser r17,*
d = 1, (offset from r16)
Opcode: EF1F

Same Opcode!

Compare with:
ldi r17, 0xFF
d = 1, (offset from r16)
K = 0xFF, 0b1111 1111
Opcode: EF1F



Intel HEX

Extract from www.wikipedia.com

From Wikipedia, the free encyclopedia

Intel HEX is a file format that conveys binary information in ASCII text form. It is commonly used for programming microcontrollers, EPROMs, and other types of programmable logic devices. In a typical application, a compiler or assembler converts a program's source code (such as in C or assembly language) to machine code and outputs it into a HEX file. The HEX file is then imported by a programmer to "burn" the machine code into a ROM, or is transferred to the target system for loading and execution.[1]

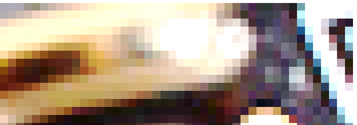
Format [edit]

Intel HEX consists of lines of ASCII text that are separated by line feed or carriage return characters or both. Each text line contains hexadecimal characters that encode multiple binary numbers. The binary numbers may represent data, memory addresses, or other values, depending on their position in the line and the type and length of the line. Each text line is called a *record*.

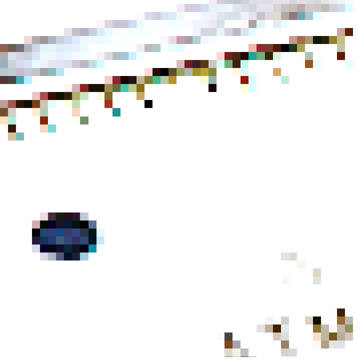
Record structure [edit]

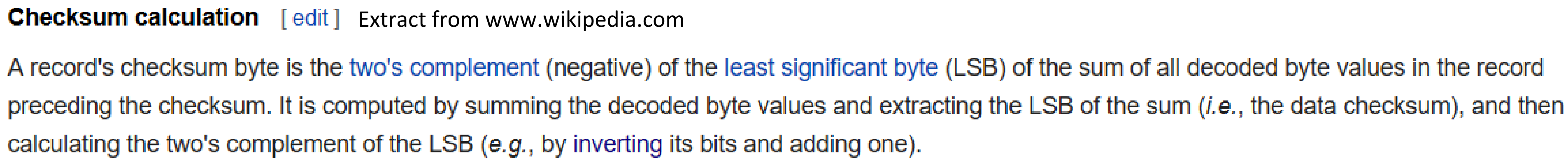
A record (line of text) consists of six fields (parts) that appear in order from left to right:

1. **Start code**, one character, an ASCII colon ':'.
2. **Byte count**, two hex digits, indicating the number of bytes (hex digit pairs) in the data field. The maximum byte count is 255 (0xFF). 16 (0x10) and 32 (0x20) are commonly used byte counts.
3. **Address**, four hex digits, representing the 16-bit beginning memory address offset of the data. The physical address of the data is computed by adding this offset to a previously established base address, thus allowing memory addressing beyond the 64 kilobyte limit of 16-bit addresses. The base address, which defaults to zero, can be changed by various types of records. Base addresses and address offsets are always expressed as big endian values.
4. **Record type** (see record types below), two hex digits, *00* to *05*, defining the meaning of the data field.
5. **Data**, a sequence of *n* bytes of data, represented by 2*n* hex digits. Some records omit this field (*n* equals zero). The meaning and interpretation of data bytes depends on the application.
6. **Checksum**, two hex digits, a computed value that can be used to verify the record has no errors.



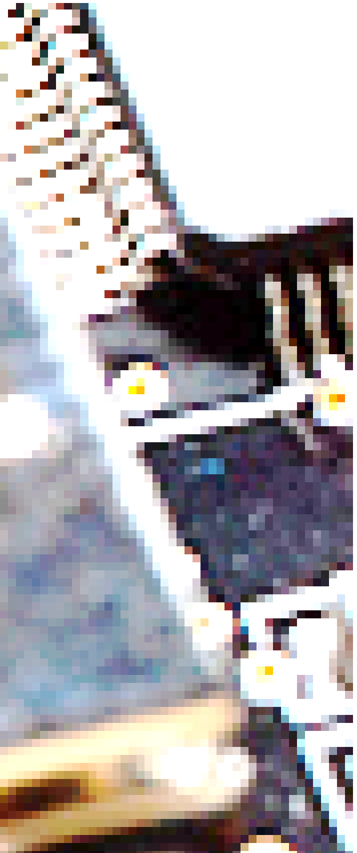
Check this link too:
http://www.sbprojects.com/knowledge/fileformats/intelhex.php



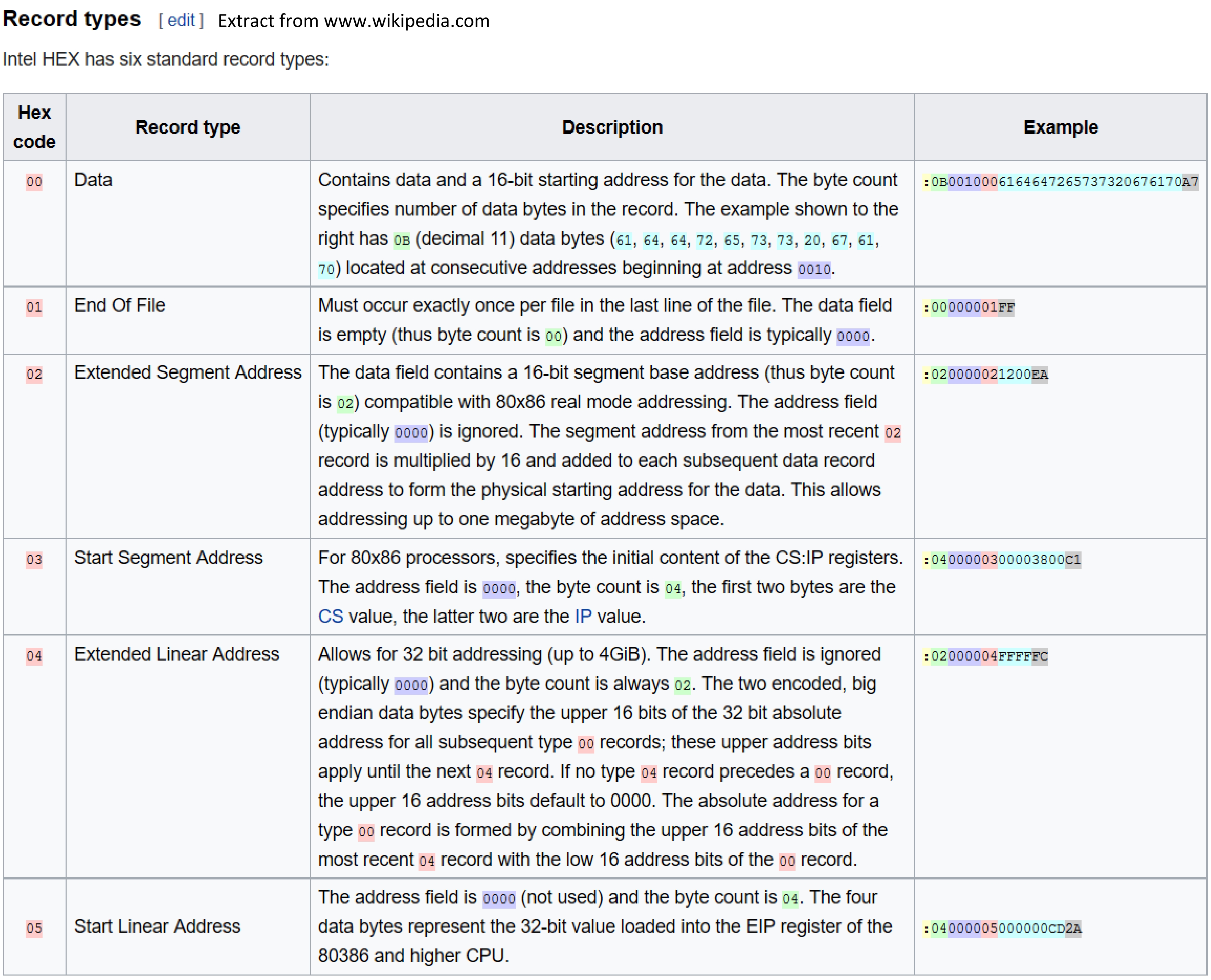


Checksum calculation [edit] Extract from www.wikipedia.com

A record's checksum byte is the two's complement (negative) of the least significant byte (LSB) of the sum of all decoded byte values in the record preceding the checksum. It is computed by summing the decoded byte values and extracting the LSB of the sum (*i.e.*, the data checksum), and then calculating the two's complement of the LSB (*e.g.*, by inverting its bits and adding one).



(Sum of all bytes) % 0xFF → Apply Two's complement. That's the checksum.



Record types [edit] Extract from www.wikipedia.com

Intel HEX has six standard record types:

Hex code	Record type	Description	Example
`00`	Data	Contains data and a 16-bit starting address for the data. The byte count specifies number of data bytes in the record. The example shown to the right has `0B` (decimal 11) data bytes (`61`, `64`, `64`, `72`, `65`, `73`, `73`, `20`, `67`, `61`, `70`) located at consecutive addresses beginning at address `0010`.	`:0B0010006164647265737320676170A7`
`01`	End Of File	Must occur exactly once per file in the last line of the file. The data field is empty (thus byte count is `00`) and the address field is typically `0000`.	`:00000001FF`
`02`	Extended Segment Address	The data field contains a 16-bit segment base address (thus byte count is `02`) compatible with 80x86 real mode addressing. The address field (typically `0000`) is ignored. The segment address from the most recent `02` record is multiplied by 16 and added to each subsequent data record address to form the physical starting address for the data. This allows addressing up to one megabyte of address space.	`:020000021200EA`
`03`	Start Segment Address	For 80x86 processors, specifies the initial content of the CS:IP registers. The address field is `0000`, the byte count is `04`, the first two bytes are the CS value, the latter two are the IP value.	`:0400000300003800C1`
`04`	Extended Linear Address	Allows for 32 bit addressing (up to 4GiB). The address field is ignored (typically `0000`) and the byte count is always `02`. The two encoded, big endian data bytes specify the upper 16 bits of the 32 bit absolute address for all subsequent type `00` records; these upper address bits apply until the next `04` record. If no type `04` record precedes a `00` record, the upper 16 address bits default to 0000. The absolute address for a type `00` record is formed by combining the upper 16 address bits of the most recent `04` record with the low 16 address bits of the `00` record.	`:02000004FFFFFC`
`05`	Start Linear Address	The address field is `0000` (not used) and the byte count is `04`. The four data bytes represent the 32-bit value loaded into the EIP register of the 80386 and higher CPU.	`:04000005000000CD2A`

Hexcode, example 1

```
.include "m2560def.inc"

loop:

ldi r16, 0xaa  ; load 0xAA to register r16
```

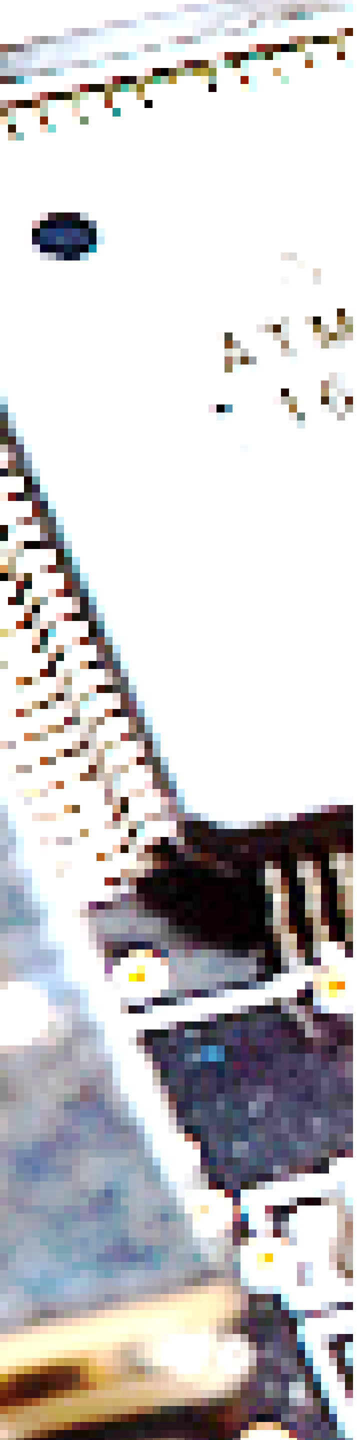
Hexcode:

```
:020000020000FC
:020000000AEA0A
:00000001FF
```

Checksum:

02+00+00+00+0A+EA= 0xF6 %FF = 0xF6 = 0b 1111 0110.

1-compl = 0000 1001. 2-compl = 0000 1010 = 0x0A



Hexcode, example 2

```
.include "m2560def.inc"

loop:
ldi r16, 0x0F
out DDRB, r16
ldi r17, 0b00110011
out PORTB, r17
```

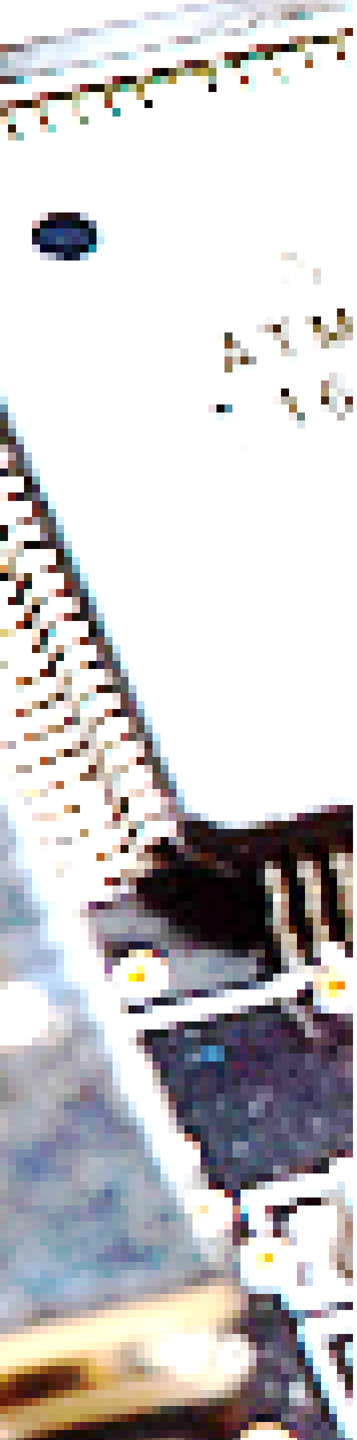
```
:020000020000FC
:080000000FE004B913E315B988
:00000001FF
```

Checksum:

08+00+00+00+0F+E0+O4+B9+13+E3+15+B9=378%FF=78

2’s-compl of 78 → 88

All in hex!



Lab 3