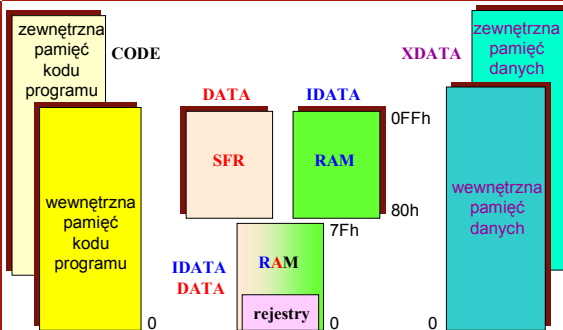


Technika cyfrowa 2

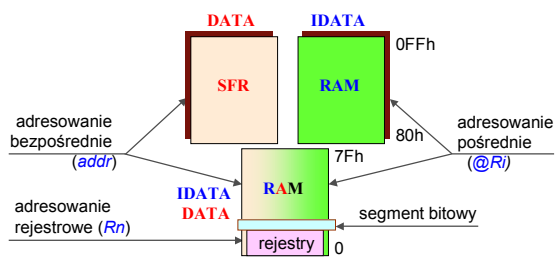
wykład 5

Katedra Metrologii Elektronicznej i Fotonicznej
Andrzej Stepień

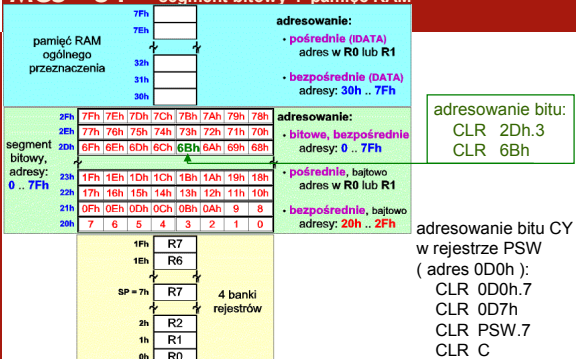
C51 – typy pamięci



C51 – adresowanie pamięci RAM i SFR



MCS[®] 51 - segment bitowy i pamięć RAM



MCS® 51 - rejestry specjalne (SFR): standard

| Rejestry specjalne (SPR): standard | | | | | | | | | |
|------------------------------------|------|------|-----|-----|-----|-----|--|------|------|
| 0F8h | B | | | | | | | | 0FFh |
| 0F0h | | | | | | | | | 0F7h |
| 0E8h | | | | | | | | | 0EFh |
| 0E0h | ACC | | | | | | | | 0E7h |
| 0D8h | | | | | | | | | 0DFh |
| 0D0h | PSW | | | | | | | | 0D7h |
| 0C8h | | | | | | | | | 0CFh |
| 0C0h | | | | | | | | | 0C7h |
| 0B8h | IP | | | | | | | | 0BFh |
| 0B0h | P3 | | | | | | | | 0B7h |
| 0A8h | IE | | | | | | | | 0AFh |
| 0A0h | P2 | | | | | | | | 0A7h |
| 98h | SCON | SBUF | | | | | | | 9Fh |
| 90h | P1 | | | | | | | | 97h |
| 88h | TCON | TMOD | TL0 | TL1 | TH0 | TH1 | | | 8Fh |
| 80h | P0 | SP | DPL | DPH | | | | PCON | 87h |

↑
adresowanie
bajtowe i bitowe

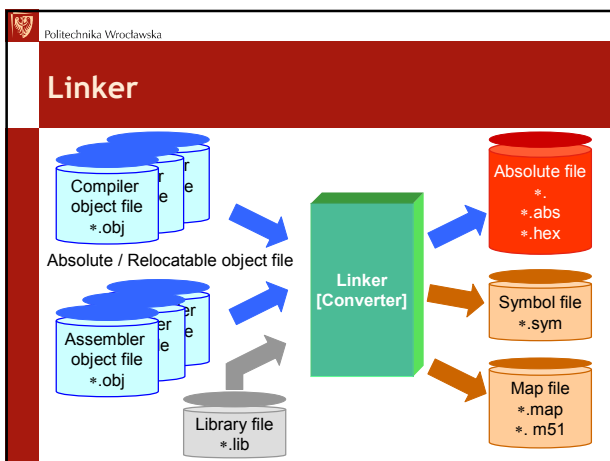
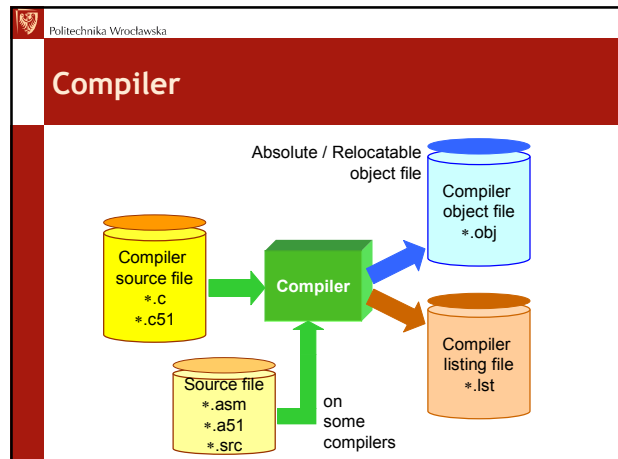
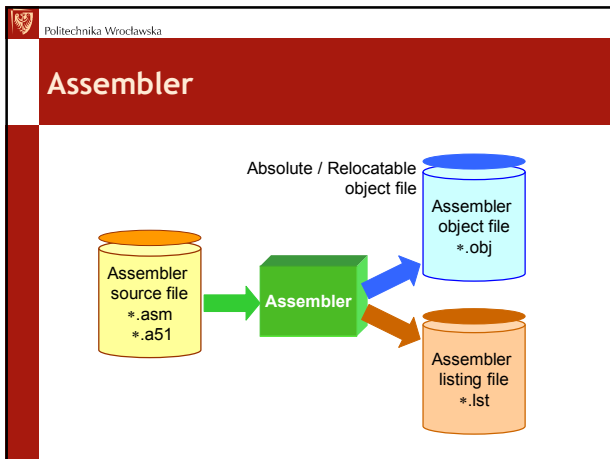
do wykorzystania w przyszłości

MCS® 51 - rejestry specjalne (SFR): C51, C515, C517A

| | | | | | | | | | | |
|------|---------|---------|--------|---------|---------|---------|---------|--------|--|------|
| 0F8h | P5 | | | P6 | | | IS0 | IS1 | | 0FFh |
| 0F0h | B | | | CM6 | CM7 | CMEN | CMSEL | | | 0F7h |
| 0E8h | P4 | MD0 | | MD1 | MD2 | MD4 | MD5 | ARCON | | 0FEh |
| 0E0h | ACC0 | CTCON | CM3 | | CM4 | CM1 | CM5 | CMH5 | | 0DFh |
| 0D8h | ADCON | ADDA0 | ADAT | P6 P7 | ADCON4 | P8 | CTRELL | CTRELH | | 0DEh |
| 0D0h | PSW | IRCON1 | CM0 | CMH0 | CM1 | CMH1 | CM2 | CMH2 | | 0Cfh |
| 0C8h | T2CON | CC4EN | CRCL | CRCH | TL2 | TH2 | CCL4 | CH4 | | 0C7h |
| 0C0h | IRCON0 | CCEN | CCL1 | CC1 | CCL2 | CCH2 | CCL3 | CCH3 | | 0BFh |
| 0B8h | IP IEN1 | IP1 | SORELH | \$IRELH | | | | | | 0BEh |
| 0B0h | P3 | SYSCON | | | | | | | | 0B7h |
| 0A8h | IE IEN0 | IP0 | SORELL | | | | | | | 0A6h |
| 0A0h | P2 | COMSETL | | COMCLR1 | COMCLR4 | SETMSK | CLRMASK | | | 0A7h |
| 98h | S0CON | S0BUF | EN2 | S1CON | S1BUF | \$IRELL | | | | 0A0h |
| 90h | P1 | XPAQE | DSEL | TL1 | TH0 | TH1 | | | | 9Fh |
| 88h | TCON | TMOD | TL0 | TL1 | TH0 | TH1 | | | | 8Fh |
| 80h | P0 | SP | DPL | DPH | WDTL | WDTH | WDREL | PCON | | 87h |

↑
adresowanie bajtowe i bitowe

C51, C515, C517A



Politechnika Wrocławska

Pierwszy program - zasady

instrukcja

[etykieta:] **mnemonik instrukcji** [*operand1*], [*operand2*], .. [komentarz]

dyrektywa

etykieta - symboliczny adres
instrukcja - rozkaz wykonywany przez procesor
mnemonik - symboliczna nazwa instrukcji
operand - argumenty instrukcji
dyrektywa - polecenie dla asemblera
komentarz - tekst pomijany przez asembler, od ';' do CR LF

Politechnika Wrocławska

Pierwszy program - c51 kod źródłowy

```

CSEG AT 0          ; pierwsza instrukcja rozpoczyna się od adresu 0
Początek:
MOV R7, #248       ; 1 R7 ← 248 = 0F8h
Skok1:
DJNZ R7, Skok1     ; 2 R7 ← R7-1
                   ; jeśli R7 ≠ 0 to wykonaj instrukcję o etykiecie Skok1
                   ; jeśli R7 = 0 to wykonaj następną instrukcję
Negacja:
CPL P1.1           ; 1 P1.1 ← NOT (P1.1)
JMP Początek       ; 2 rozpocznij wykonywanie instrukcji od etykiety Początek
END                ; koniec pliku źródłowego
  
```

Politechnika Wrocławska

Pierwszy program - pojęcia

```

CSEG AT 0          ; pierwsza instrukcja rozpoczyna się od adresu 0
Początek:
MOV R7, #80h       ; 1 R7 ← 80h=128
Skok1:
NOP
DJNZ R7, Skok1     ; 2 R7 ← R7-1
                   ; jeśli R7 ≠ 0 to wykonaj instrukcję o etykiecie Skok1
                   ; jeśli R7 = 0 to wykonaj następną instrukcję
Negacja:
CPL P1.1           ; 1 P1.1 ← NOT (P1.1)
JMP Początek       ; 2 rozpocznij wykonywanie instrukcji od etykiety Początek
END                ; koniec pliku źródłowego
  
```

```

Politechnika Wrocławska

A51 MACRO ASSEMBLER TEST1                                10/25/2005 23:02:11 PAGE 1
MACRO ASSEMBLER A51 V6.10
OBJECT MODULE PLACED IN .\test1.OBJ
ASSEMBLER INVOKED BY: E:\KEIL\C51\BIN\A51.EXE .\test1.a51 SET(SMALL) DEBUG EP
LOC OBJ      LINE  SOURCE
1             ;Przykład programu
2
3             CSEG  AT  0      ; pierwsza instrukcja rozpoczyna się
4             ;              od adresu 0
5 Poczatek:
6 MOV  R7, #248 ;1 R7 <= 248 = 0F8h
7 Skok1:
8 DJNZ R7, Skok1 ;2 R7 <= R7-1
9             ;      jeśli R7 <> 0 to wykonaj instrukcję
10            ;      o etykiecie Skok1
11            ;      jeśli R7 = 0 to wykonaj
12            ;      następną instrukcję
13 Negacja:
14 CPL  P1.1      ;1 P1.1 <= NOT (P1.1)
15 LJMP Poczatek ;2 rozpocznij wykonywanie instrukcji
16            ;      od etykiety Poczatek
17 END           ; koniec pliku źródłowego

```

```

Politechnika Wrocławska

Pierwszy program - C51 plik *.m51

BL51 BANKED LINKER/LOCATER V4.03 1                      10/25/2005 23:02:18 PAGE 1
BL51 BANKED LINKER/LOCATER V4.03, INVOKED BY:
E:\KEIL\C51\BIN\BL51.EXE test1.obj TO test1

INPUT MODULES INCLUDED:
test1.obj (TEST1)

LINK MAP OF MODULE: test1 (TEST1)

TYPE  BASE      LENGTH  RELOCATION  SEGMENT NAME
-----
***** DATA MEMORY *****
REG  0000H  0008H  ABSOLUTE  "REG BANK 0"

***** CODE MEMORY *****
CODE 0000H  0009H  ABSOLUTE

LINK/LOCATE RUN COMPLETE. 0 WARNING(S), 0 ERROR(S)

```

Pierwszy program - pamięć programu

Program memory address:

| Address | Hex | Assembly | Hex | Binary | Notes |
|----------------|-----|----------------|-----|----------|-----------------------|
| Poczatek: 0000 | 7F | MOV R7, #248 | 0F8 | 01111111 | rrr = 111 |
| 0001 | F8 | | 0F8 | | |
| Skok1: 0002 | DF | DJNZ R7, Skok1 | 0FE | 11101111 | rrr = 111 |
| 0003 | FE | | 0FE | | |
| 0004 | B2 | CPL P1.1 | 91 | 10111100 | |
| 0005 | 91 | | 91 | | |
| 0006 | 02 | LJMP Poczatek | 00 | 00000000 | addr _{15..8} |
| 0007 | 00 | | 00 | | |
| 0008 | 00 | | 00 | | addr _{7..0} |
| 0009 | xx | | | | |

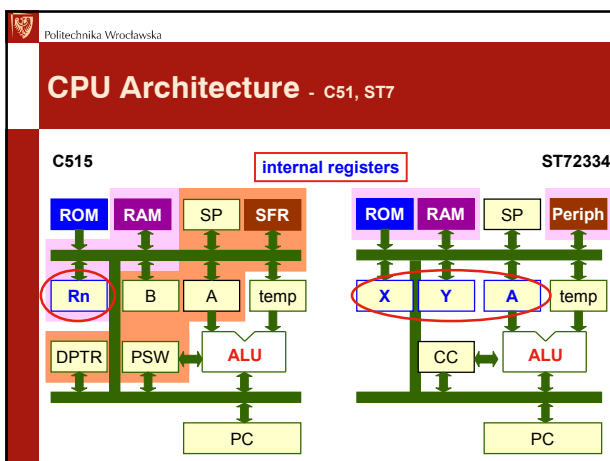
Operacje logiczne

Funkcje podstawowe: NOT, AND, OR
 uzupełniające: NAND, NOR, XOR
 przesuwanie zawartości rejestrów (shift, swap ..)
 odtwarzanie znaku liczby itp.

Dotyczą: rozkazów procesora
 instrukcji języka wysokiego poziomu, np. C
 dyrektyw asemblera
 dyrektyw kompilatora

Zastępują inne instrukcje procesora / upraszczają działanie procesora, np.

- zerowanie rejestru: ⇔ rejestr XOR rejestr
 rejestr AND 0x00 (dodatkowy operand ?)
- ustawianie bitów: ⇔ rejestr OR 0xFF (dodatkowy operand ?)



Literatura - C51, ST7

C51:
 8-Bit Microcontrollers C500. Architecture and Instruction Set. User's Manual. July 2000, Infineon Technologies AG.

ST7:
 ST7 FAMILY. PROGRAMMING MANUAL. Rev. 1.1, March 1999, STMicroelectronics.

Politechnika Wrocławska

Byte logical operations - C51, ST7

| C51 | | | ST7 | | | |
|------|----------|-----|------|-----------|-----|--|
| mnem | dst | src | mnem | dst | src | operation |
| ANL | dst, src | | AND | A, mem | | logical AND |
| ORL | dst, src | | OR | A, mem | | logical OR |
| XRL | dst, src | | XOR | A, mem | | logical XOR |
| CPL | A | | CPL | reg [mem] | | logical 1-complement (inverted) |
| | | | NEG | reg [mem] | | logical 2-complement (negate) |

Politechnika Wrocławska

Byte logical operations - C51, ST7 - AND

| C51 | | | | ST7 | | | |
|------|-------------------------------|-----|--------|------|----------|-----|-------|
| mnem | dst | src | flags | mnem | dst | src | flags |
| ANL | A, #data | | only P | AND | A, #byte | | N, Z |
| ANL | A, mem | | | AND | A, mem | | |
| ANL | A, reg | | | | | | |
| ANL | mem _{direct} , A | | | | | | |
| ANL | mem _{direct} , #data | | | | | | |

Operation:

$$A \leftarrow A \text{ AND } src$$

$$A \leftarrow A \text{ AND } reg$$

$$(mem_{direct}) \leftarrow (mem_{direct}) \text{ AND } A \quad \text{only C51}$$

$$(mem_{direct}) \leftarrow (mem_{direct}) \text{ AND } 8\text{-bit data}$$

Description:

ANL performs the bitwise logical AND operation between the variables indicated and stores the results in the destination variable.

Politechnika Wrocławska

Byte logical operations - C51, ST7 - NEG

| C51 | | | ST7 | | |
|------|-----|-----|------|-----|-----|
| mnem | dst | src | mnem | dst | src |
| | | | NEG | reg | |
| | | | NEG | mem | |

Operation:

$$dst \leftarrow (dst \text{ XOR } \$FF) + 1 \quad \text{or} \quad 00 - dst$$

Description:

The destination byte is read, then each bit is toggled (inverted), and the result is incremented before it is written at the destination byte (**negate, logical 2-complement**). The destination is either a memory byte or a register. The Carry is cleared if the result is zero. This instruction is used to negate signed values. This instruction is compact, and does not affect any register when used with RAM variables.

Politechnika Wrocławska

Program Status Word (PSW)

Register (C51):
addr in SFR = 0D0h

| | | | | | | | |
|----|----|----|-----|-----|----|----|---|
| CY | AC | F0 | RS1 | RS0 | OV | F1 | P |
|----|----|----|-----|-----|----|----|---|

Condition Code (CC) Register (ST7):

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | H | I | N | Z | C |
|---|---|---|---|---|---|---|---|

| C51 | | | ST7 | | |
|------|-----|-----|------|-----------|-----|
| mnem | dst | src | mnem | dst | src |
| SETB | bit | | BSET | dst, #pos | |
| CLR | bit | | BRES | dst, #pos | |
| CPL | bit | | | | |
| | | | BCP | scr, dst | |

operation

set bit

clear bit

complement bit

logical bit compare
{ N, Z } <= src AND A_i

Code Condition (CC) flag modification (**only ST7**):

SCF C ← 1, set Carry Flag

SIM I ← 1, set Interrupt Mask/Disable Interrupt

RCF C ← 0, reset Carry flag

RIM I ← 0, reset Interrupt Mask/Enable Interrupt

Politechnika Wrocławska

Bit logical operations - C51, ST7 - Set

| C51 | | | ST7 | | |
|------|-----|-------|------|----------------|-------|
| mnem | dst | flags | mnem | dst | flags |
| SETB | bit | no | BSET | dst, #pos | no |
| | | | | (pos = 0 .. 7) | |

Operation:

$$bit \leftarrow 1 \quad \text{for C51}$$

$$dst \leftarrow dst \text{ OR } (2^{pos}) \quad \text{for ST7}$$

Description:

SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.

Read the destination byte, set the corresponding bit (bit position), and write the result in destination byte. The destination is a memory byte. The bit position is a constant. This instruction is fast, compact, and does not affect any register.

Politechnika Wrocławska

Maskowanie bitów - zerowanie

Przykład dla C51: **wyzerować** podane bity rejestru R4: R₄₆, R₄₅, R₄₂ i R₄₀.

rejestr R4:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Maska EQU 0110 0101b ; deklaracja bitów maski

;maska:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

Zerowanie_bitów:

```

MOV  A, R4          ; A ← R4
ANL  A, #NOT Maska ; A ← A and NOT maska
                        ; A ← A and 1001 1010b
                        ; A = x00x x0x0b
MOV  R4, A          ; R4 ← A
  
```

Politechnika Wrocławska

Maskowanie bitów - ustawianie

Przykład dla C51: **ustawić** (wpisać stan HIGH) podane bity w wewnętrznej pamięci RAM (DATA) o adresie 5Ah: 5Ah₇, 5Ah₅, 5Ah₃ i 5Ah₀.

(5Ah)_{DATA}:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Maska EQU 1010 1001b ; deklaracja bitów maski
;maska:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
|---|---|---|---|---|---|---|---|

Ustawianie bitów: ; wersja dydaktyczna
MOV A, 5Ah ; A ← (5Ah)
ORL A, #Maska ; A ← A or maska
; A ← A or 1010 1001b
; A = 1x1x 1xx1b
; (5Ah) ← A

MOV 5Ah, A ; wersja podstawowa
ORL 5Ah, #Maska

Politechnika Wrocławska

Maskowanie bitów - negacja

Przykład dla C51: **zanegować** logicznie linie portu: P1.3, P1.4 i P1.6

port P1:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

np:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Maska EQU 0101 1000b ; deklaracja bitów maski
;maska:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

Negowanie bitów:
XRL P1, #Maska ; P1 ← P1 xor maska
; P1 ← P1 xor 0101 1000b
; P1 = x6x4 3xxxb
; P1 = 0011 0010b

Politechnika Wrocławska

Przesuwanie bitów - C51

przesunięcia w prawo (*analogicznie w lewo*):

RR A RL A
; A₇ → A₆ → A₅ → A₄ → A₃ → A₂ → A₁ → A₀ →

RRC A RLC A
; C → A₇ → A₆ → A₅ → A₄ → A₃ → A₂ → A₁ → A₀ →

W C51 obie instrukcje dotyczą tylko akumulatora A

Politechnika Wrocławska

Przesuwanie bitów - ST7

przesunięcia w prawo (*analogicznie w lewo*):

RRC reg [mem] RLC reg [mem]
; C → reg [mem] - Rotate Right Logical through Carry →

SRL reg [mem] SLL reg [mem]
0 → reg [mem] - Shift Right Logical → C

SRA reg [mem] SLA reg [mem]
MSB → reg [mem] - Shift Right Arithmetic → C

Politechnika Wrocławska

Wielobajtowe przesuwanie bitów - C51

przykład: przesunąć binarnie (podzielić przez 2) 2-bajtową zmienną zawartą w rejestrach: R7 (MSB) i R6 (LSB).

R7:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

R6:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|

C = ?

Przesun_2_bajty:
MOV A, R7 ; A ← R7
RRC A ; C → A₇ → A₆ .. A₁ → A₀ → C
MOV R7, A ; R7 ← A
MOV A, R6 ; A ← R6
RRC A ; C → A₇ → A₆ .. A₁ → A₀ → C
MOV R6, A ; R6 ← A

Politechnika Wrocławska

Porównania

Czy prawdziwe jest twierdzenie:

1111 1110b > 0000 0100b

Tak dla liczb całkowitych bez znaku (**Unsigned**), 254 > 4

Nie dla liczb całkowitych ze znakiem (**Signed**), -2 < 4

Politechnika Wrocławska

Interpretacja liczb całkowitych

1000 0010b = 82h

= **130**, dla 1-bajtowej liczby całkowitej bez znaku:
 $1 * 2^7 + 0 * 2^6 + 0 * 2^5 + 0 * 2^4 + 0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$
zakres: 0 .. 255

= **-126**, dla 1-bajtowej liczby całkowitej ze znakiem zapisanej w kodzie U2
 $-(128 - 0000 0010b) = -126$
 $-(NOT 1000 0010b + 1) = -(0111 1101b + 1) = -0111 1110b$
zakres: -128 .. +127

= **82**, dla 1-bajtowej liczby całkowitej zapisanej w kodzie BCD
zakres: 0 .. 99

Politechnika Wrocławska

Znaczni: C, H/AC, OV, P, N, Z

0x7F + 1 = 0111 1111b
0000 0001b
0100 0000b
C=0, N=1, OV AC/H

0xFF + 1 = 1111 1111b
0000 0001b
0000 0000b
C=1, N=0, OV AC/H

OV = C xor A_{7|6}
P = 0 dla parzystej liczby 1 w Acc (**C51**)

bez znaku: 127 + 1 = 128
ze znakiem: +127 + 1 = -128 !
255 + 1 = 256 !
-1 + 1 = 0

Politechnika Wrocławska

Conditional Branch - C51 (1/2)

| Test | Mnemonic | Operation |
|---------|--------------|-----------------------------|
| A = 0 | JZ rel | jump if Acc = 0 |
| A ≠ 0 | JNZ rel | jump if Acc ≠ 0 |
| C = 0 | JNC rel | jump if C = 0 |
| C = 1 | JC rel | jump if C = 1 |
| bit = 0 | JNB bit, rel | jump if bit = 0 |
| bit = 1 | JB bit, rel | jump if bit = 1 |
| bit = 1 | JBC bit, rel | jump if bit = 1 & clear bit |

Politechnika Wrocławska

Conditional Branch - ST7

| Test | Mnemonic | Operation |
|---------|----------------|---------------------------|
| bit = 0 | BTJF dst, #pos | jump if bit is false (0) |
| bit = 1 | BTJT dst, #pos | jump if bit is true (1) |
| xx | JRxx | jump if condition is true |

Politechnika Wrocławska

Condition

| Test | Mnem | Condition | Comment |
|------|------|--------------|--------------------------------|
| | NC | C = 0 | Not Carry |
| | UGE | C = 0 | Unsigned Greater or Equal (>=) |
| | C | C = 1 | Carry |
| | ULT | C = 1 | Unsigned Lower Than (<) |
| | NE | Z = 0 | Not Equal |
| | EQ | Z = 1 | Equal |
| | UGT | (C OR Z) = 0 | Unsigned Greater Than (>) |
| | ULE | (C OR Z) = 1 | Unsigned Lower or Equal (<=) |
| | T | True | (1) |
| | RF | False | (0) |
| | NH | H = 0 | Not Half-Carry |
| | RH | H = 1 | Half-Carry |
| | IL | | Interrupt Line is Low |
| | IH | | Interrupt Line is High |
| | NM | I = 0 | Not Interrupt Mask |
| | M | I = 1 | Interrupt Mask |
| | PL | N = 0 | Plus |
| | MI | N = 1 | Minus |

Politechnika Wrocławska

Addition - C51, ST7

C51

ADD A, { Rn, Adr, @Ri, #dana } ; A ← A + scr
Flags: C, AC, OV

ADDC A, { Rn, adr, @Ri, #dana } ; A ← A + scr + C
Flags: C, AC, OV

INC { A, Rn, adr, @Ri, DPTR } ; dst ← dst + 1
Flags: P

ST7

ADD A, mem ; A ← A + mem
Flags: C, H, N, Z

ADC A, mem ; A ← A + mem + 1
Flags: C, H, N, Z

INC { A, X, Y, mem } ; dst ← dst + 1
Flags: N, Z

Politechnika Wrocławska

Multiplication / Division (unsigned) - C51, ST7

C51

```
MUL AB ; BA ← A * B
Flags: C=0, P
      OV=1 if B≠0 (product
      greater than 0xFF)

DIV AB ; A ← INT (A/B)
      ; B ← MOD (A/B)
Flags: C=0, P
      OV=1 if B=0 (originally)
```

ST7

```
MUL X, A ; X:A ← X * A
MUL Y, A ; Y:A ← Y * A
Flags:
  C=0, H=0
```

Politechnika Wrocławska

Dodawanie binarne - algorytm

Dodać:

- dwie, 2-bajtowe zmienne bez znaku zawarte w pamięci danych
- obie zmienne zapisane w standardzie Little Endian
- suma zapisana w miejsce pierwszej zmiennej

Politechnika Wrocławska

Dodawanie binarne - C51 (1/2)

Dodawanie:

```
MOV A, Addr_1 ; A ← (Addr_1)
ADD A, Addr_2 ; A ← A + (Addr_2)
MOV Addr_2, A ; (Addr_2) ← A

MOV A, Addr_1+1 ; A ← (Addr_1+1)
ADDC A, Addr_2+1 ; A ← A + (Addr_2+1) + C
MOV Addr_2+1, A ; (Addr_2+1) ← A

CLR A ; A ← 0
MOV Acc.0, C ; A7 ← C
MOV Addr_2+2, A ; (Addr_2+2) ← A
```

Politechnika Wrocławska

Dodawanie binarne - C51 (2a/2)

Dodawanie:

```
MOV R0, #Addr_1
MOV R1, #Addr_2
MOV R2, #2
CLR C
```

Powtorz:

```
MOV A, @R0
ADDC A, @R1
MOV @R1, A
INC R0
INC R1
DJNZ R2, Powtorz
```

Politechnika Wrocławska

Dodawanie binarne - C51 (2b/2)

Dodawanie:

```
MOV R0, #Addr_1 ; R0=Addr_1, adresuje pierwszą zmienną
MOV R1, #Addr_2 ; R1=Addr_2, adresuje drugą zmienną
MOV R2, #2 ; R2=2, licznik powtórzeń pętli
CLR C ; C=0 przy dodawaniu najmniej znaczących bajtów
```

Powtorz:

```
MOV A, @R0 ; A ← (Addr_1)
ADDC A, @R1 ; A ← A + (Addr_2)
MOV @R1, A ; (Addr_2) ← A
INC R0 ; R0 ← R0+1
INC R1 ; R1 ← R1+1
DJNZ R2, Powtorz ; R2 ← R2-1 ; jeśli R2 ≠ 0 to skok do Powtorz
```

Diagram illustrating the loop logic with R2=2 and R2=1 states, showing the update of pointers R0 and R1.

Politechnika Wrocławska

Arytmetyka BCD korekcja dziesiętna - C51

Dod_BCD:

```
MOV A, #95h ; A ← 95h, 95h = 1001 0101b
ADD A, #85h ; A ← A + 85h, 85h = 1000 0101b
; C=1 AC=0, A = 1Ah = 0001 1010b

DA A ; A ← A + 6 ponieważ A3..0 > 9
; C=1 A = 20h = 0010 0000b
; A ← A + 60h ponieważ C=1
; C=1 +60h = 0110 0000b
; A = 80h = 1000 0000b
```

```
MOV R6, A ; R6 ← A
CLR A ; A ← 0
MOV Acc.0, C ; A0 ← C
MOV R7, A ; R7 ← A ; A = 0000 0001b
```

Politechnika Wrocławska

Arytmetyka BCD DEC_8 - C51

DEC-rementacja 8-bitowej zmiennej zapisanej w kodzie BCD, np. 00 - 1 = 99

DEC_arg8:

```

MOV A, Arg8      ; (Arg8) = 0BCD = 0000 0000b
ADD A, #99h      ; 99h = 1001 1001b
                  ; A = 1001 1001b, AC = 0, A3..A0 ≤ 9
                  ; C = 0, A7..A4 ≤ 9
                  ; przy odejmowaniu: przekroczenie
                  ; zakresu liczb BCD (0..99)
DA A             ; brak korekcji
MOV Arg8, A      ; (Arg8) ← 99

```

Politechnika Wrocławska

C390 - Maxim / Dallas Semiconductors

- arithmetic accelerator (*DS80C390.pdf, 80C390_userguide.pdf*)
- 5 dodatkowych rejestrów: MCNT0, MCNT1, MA, MB, MC,

| | | | | | | | | |
|------|-----|-------|-------|----|----|----|--|--|
| 0D8h | PSW | MCNT0 | MCNT1 | MA | MB | MC | | |
| 0D0h | | | | | | | | |
| 0C8h | | | | | | | | |

- wykonywane operacje arytmetyczne:
 - 32/16 bitowe **dzielenie** (9 t_{cy}): 32-bitowy ilorz, 16-bitowa reszta,
 - 16/16 bitowe **dzielenie** (6 t_{cy}): 16-bitowy ilorz, 16-bitowa reszta,
 - 16*16 bitowe **mnożenie** (6 t_{cy}): 32-bitowy iloczyn,
 - 32-bitowe **przesunięcie** (9 t_{cy}): 32-bitowy wynik,
 - 32-bitowa **normalizacja** (9 t_{cy}): 32-bitowa mantysa, 5-bitowa potęga,
- szybkość taktowania: $t_{cy} = 4 / f_{osc}$

Politechnika Wrocławska

C390 - Maxim / Dallas Semiconductors - operacje

kolejność wykonywania operacji:

Faza 1: pierwszy wpis do MA lub MB

Faza 2: ostatni wpis do MA, MB lub MCNT0

Faza 3: ostatni odczyt z MA, MB lub MCNT0

Warunki: MST=0, pierwszy odczyt z MA

Legenda: wpis do rejestrów obliczenia odczyt z rejestrów

Politechnika Wrocławska

C390 - Maxim / Dallas Semiconductors - typy operacji

| typ operacji | dzielenie 32 bity / 16 bitów | dzielenie 16 bitów / 16 bitów |
|--------------|------------------------------|-------------------------------|
| początek | MA ← dzielna (LSB) | MA ← dzielna (LSB) |
| w | MA ← dzielna (LSB+1) | MA ← dzielna (MSB) |
| p | MA ← dzielna (LSB+2) | |
| i | MA ← dzielna (MSB) | |
| s | MB ← dzielnik (LSB) | MB ← dzielnik (LSB) |
| koniec | MB ← dzielnik (MSB) | MB ← dzielnik (MSB) |

| | czekaj aż MST=0 | czekaj aż MST=0 |
|------------|--------------------|-------------------|
| początek o | MA → ilorz (MSB) | MA → ilorz (MSB) |
| d | MA → ilorz (LSB+2) | MA → ilorz (LSB) |
| c | MA → ilorz (LSB+1) | |
| z | MA → ilorz (MSB) | |
| y | MB → reszta (MSB) | MB → reszta (MSB) |
| koniec t | MB → reszta (LSB) | MB → reszta (LSB) |

