5. Architektura procesoru – posuny a rotace, předávání řízení

JMP

instrukce předávání řízení

JMP (nepodmíněný skok – *jump*)

- skok uvnitř jednoho (kódového) segmentu (mění se pouze EIP):
  - krátký (*short jump*) = skok do vzdálenosti <-128; 127> bytů
  - blízký (*near jump*) = skok v rámci aktuálního segment (64 KiB)
- skok do jiného (kódového) segmentu (mění se CS a EIP):
  - vzdálený (*far jump*)
- skok dále může být
  - přímý (*direct*) x nepřímý (*indirect*)

1. `JMP rel`	… relativní, přímý, krátký
2. `JMP ofs`	… absolutní, přímý, blízký
3. `JMP reg`	… absolutní, nepřímý, blízký
4. `JMP [mem]`	… absolutní, nepřímý, blízký/vzdálený
5. `JMP seg:ofs`	… absolutní, přímý, vzdálený

```
JMP short/near rel
EIP += rel;
```

```
JMP near ofs/reg/[mem]
EIP = ofs/reg/[mem];
```

```
JMP far seg:ofs/[mem]
CS = seg/[mem+4];
EIP = ofs/[mem];
```

J*cc*, CMOV*cc*

podmíněné instrukce

Podmíněné instrukce (*Conditional Instructions*)

- Podmínku označujeme *cc*, negace podmínky = přidáme písmenko N (*Ncc*) = not *cc*
- Vykonání instrukce je podmíněno splněním dané podmínky (*cc*)
- Podmínka = nastavení příznaku (C, S, …) nebo kombinace příznaků (A, G, …)

označení (*cc*)	C	Z	S	O	P,PE	NC	NZ	NS	NO	NP,PO
příznak	CF=1	ZF=1	SF=1	O=1	P=1	CF=0	ZF=0	SF=0	O=0	P=0

podmínka	bez znaménka		se znaménkem		negovaná podmínka	bez znaménka		se znaménkem	
	označení (*cc*)	příznaky	označení (*cc*)	příznaky		označení (*cc*)	příznaky	označení (*cc*)	příznaky
==	**E** Z	ZF=1	**E** Z	ZF=1	!=	**NE** NZ	ZF=0	**NE** NZ	ZF=0
>	**A** NBE	CF=0 && ZF=0	**G** NLE	ZF=0 && SF=OF	<=	**NA** BE	CF=1 || ZF=1	**NG** LE	ZF=1 || SF≠OF
<	**B** NAE,C	CF=1	**L** NGE	SF≠OF	>=	**NB** AE,NC	CF=0	**NL** GE	SF=OF
>=	**AE** NB,NC	CF=0	**GE** NL	SF=OF	<	**NAE** B,C	CF=1	**NGE** L	SF≠OF
<=	**BE** NA	CF=1 || ZF=1	**LE** NG	ZF=1 || SF≠OF	>	**NBE** A	CF=0 && ZF=0	**NLE** G	ZF=0 && SF=OF

Jcc
instrukce předávání řízení

- Podmíněný skok (*conditional jump*) se provede pouze v případě, že je splněna podmínka *cc*
- Pouze relativní, přímé, blízké skoky (*relative, direct, near*)

```
1. Jcc rel
```

```
Jcc rel
if (cc) EIP += rel;
```

- Název instrukce lze popsat takto:

JCXZ, **J**ECXZ, **J**([N](C | Z | S | O | P | E | (A | B | G | L)[E]))|(PE|PO))

  - **J** … Jump
  - CXZ nebo ECXZ … ((E)CX == 0)? nebo podmínka … *cc*
  - volitelně: N … Not (= negace podmínky), podmínka: příznak nebo různé kombinace porovnání znaménkových/bezznaménkových čísel
    - příznaky: C, Z, S, O, P, PE, PO
    - porovnání čísel bez znaménka: E, A, B, AE, BE
    - porovnání čísel se znaménkem: E, G, L, GE, LE

cc ∈ {C,Z,S,O,P,PO,PE,NC,NZ,NS,NO,E,NE,A,NA,B,NB,G,NG,L,NL,AE,NAE,GE,NGE,BE,NBE,LE,NLE}

CMOV*cc*
podmíněná přenosová instrukce

```
1. CMOVcc r16/r32,r16/r32
2. CMOVcc r16/r32,mem
```

```
CMOVcc dst,src
if (cc) dst = src;
```

Oba operandy musí mít stejnou velikost (16, nebo 32 bitů).

```
CMOVAE  AX,BX
CMOVNZ  EBX,ECX
CMOVPO  AX,[p1]
CMOVE   EAX,2
```

- Název instrukce lze popsat podobně jako u J*cc*:
  CMOV([N](C | Z | S | O | P | E | (A | B | G | L)[E]))|(PE|PO))

r16 ∈ {AX,BX,CX,DX,SI,DI,BP,SP}, r32 ∈ {EAX,EBX,ECX,EDX,ESI,EDI,ESP,EBP}, mem = paměť
cc ∈ {C,Z,S,O,P,PO,PE,NC,NZ,NS,NO,E,NE,A,NA,B,NB,G,NG,L,NL,AE,NAE,GE,NGE,BE,NBE,LE,NLE}

LOOP*cc*

instrukce předávání řízení

- Podmíněný skok, který se provede pouze v případě, že je splněna podmínka ECX ≠ 0 = cyklus FOR pro daný počet opakování
- Pouze relativní, přímé, krátké skoky (*relative, direct, short*)

1. LOOP*cc* rel8

```
LOOPcc rel8
ECX = ECX – 1;
if (ECX != 0 && cc) EIP += rel8;
```

- Název instrukce lze popsat takto:

LOOP[[N]E]

  - LOOP … skočí ⇔ ECX != 0
  - LOOPE … skočí ⇔ ECX != 0 && ZF == 1
  - LOOPNE … skočí ⇔ ECX != 0 && ZF == 0
- Příznak ZF v podmíněné instrukci LOOP*cc* může být nastaven některou z instrukcí, která mění příznaky (například CMP)

Použití LOOP (cyklus s daným počtem opakování)

```
    MOV ECX,10
    MOV EAX,0
ptr1:
    ADD EAX,1  ; zvýší EAX o 1, nastaví EFLAGS
    LOOP ptr1  ; skočí ⇔ ECX != 0 (na začátku ECX=10… cyklus se provede 10x)
    …          ; ECX = 0, EAX = 10
```

```
int EAX = 0, ECX;
for (ECX = 10; ECX != 0; ECX--) EAX++;
                    nebo
do { EAX++; } while (--ECX != 0);
```

dvě alternativy, rozdíl = LOOP nemění příznaky

```
ptr1: ; alternativa předchozího cyklu LOOP
    ADD EAX,1   ; zvýší EAX o 1, nastaví EFLAGS
    DEC ECX     ; sníží ECX o 1, nastaví EFLAGS
    JNZ ptr1    ; skočí ⇔ ECX != 0 (na začátku ECX=10… cyklus se provede 10x)
    …           ; ECX = 0, EAX = 10
```

```
    MOV ECX,10
    MOV EAX,5
ptr2:
    SUB EAX,1     ; sníží EAX o 1, nastaví EFLAGS, ve chvíli, kdy EAX == 0, nastaví ZF na 1
    LOOPNE ptr2   ; skočí ⇔ ECX != 0 && ZF == 0, což bude dokud EAX != 0 (5x)
    …             ; ECX = 5, EAX = 0
```

```
int EAX = 5, ECX = 10;
do {
  EAX--;
} while (--ECX != 0 && EAX != 0);
```

Jak nastavit příznaky? (1.)

- Instrukcemi CMP a TEST
- Dalšími aritmetickými instrukcemi
- Instrukcemi pro ovládání příznaků: STD, STC, CLC, CLD, CMC, LAHF, SAHF

STC, STC, CLC, CLD, CMC

instrukce ovládání příznaků

```
STC (Set CF)
CF = 1;
```

```
STD (Set DF)
DF = 1;
```

```
CLC (Clear CF)
CF = 0;
```

```
CLD (Clear DF)
DF = 0;
```

```
CMC (Complement CF)
CF = ~CF;
```

Jak nastavit příznaky? (2.)

LAHF, SAHF
instrukce ovládání příznaků

```
LAHF (Load FLAGS into AH register)
AH = SF:ZF:0:AF:0:PF:1:CF;
```

```
SAHF (Store AH into FLAGS register)
SF:ZF:0:AF:0:PF:1:CF = AH;
```

- Změna příznaku:
  - nahrajeme FLAGS do AH
  - změníme AH
  - uložíme AH do FLAGS
- Například změna (vynulování) příznaku SF (bit 7 ~ 2^7 = 128):

```
    LAHF
    AND AH,0x7F ; 0x7F = 0111 1111
    SAHF
    JS ptr2  ; skočí ⇔ SF = 1, SF by mělo být 0
    …
ptr2:
```

Jak nastavit příznaky? (3.)

- Rozdíl dvou čísel: CMP
  - CMP src1,src2 … nastaví příznaky, jakoby došlo k výpočtu rozdílu src1 – src2
  - na základě výsledku můžeme [ne]provést podmíněnou instrukci

```
    MOV EAX,10
    MOV EBX,5
    CMP EAX,EBX     ; provede EAX – EBX, nastaví EFLAGS, výsledek neuloží
    JZ ptr1         ; skočí ⇔ ZF = 1 ⇔ EAX = EBX
    …dělej něco…
ptr1:
```

```
int EAX = 10, EBX = 5;
if (EAX != EBX) { …dělej něco… }
```

- Logický součin dvou čísel: TEST
  - TEST src1,src2 … nastaví příznaky, jakoby došlo k výpočtu logického součinu src1 & src2
  - na základě výsledku můžeme [ne]provést podmíněnou instrukci

```
    MOV EAX,0x83   ; = 00…0010000011
    TEST EAX,0x02  ; provede EAX & 2, nastaví EFLAGS, výsledek neuloží
    JNZ ptr2       ; skočí ⇔ ZF = 0 ⇔ bit 1 registru EAX = 1
    …dělej něco…
ptr2:
```

```
int EAX = 0x83;
if ((EAX & 2) == 0) { …dělej něco… }
```

SHL, SAL, SHR, SAR
instrukce posuvů a rotací

mění příznaky:	OF	SF	ZF	AF	CF	PF

SHL dst,src (*Shift Logical Left*)
SAL dst,src (*Shift Arithmetic Left*)
SHR dst,src (*Shift Logical Right*)
SAR dst,src (*Shift Arithmetic Right*)

```
1. SHL/SAL/SHR/SAR reg,CL
2. SHL/SAL/SHR/SAR reg(,imm8)  // bez imm8 = posun o 1
3. SHL/SAL/SHR/SAR mem,CL
4. SHL/SAL/SHR/SAR mem(,imm8)  // bez imm8 = posun o 1
```

```
SHL/SAL dst,src
dst = dst * (2^src);
```

```
SHR dst,src
dst = dst / (2^src); // unsigned divide
```

```
SAR dst,src
dst = dst / (2^src); // signed divide
```

Příznaky ZF, SF a PF nastaví podle výsledku instrukce a příznak AF není definován.

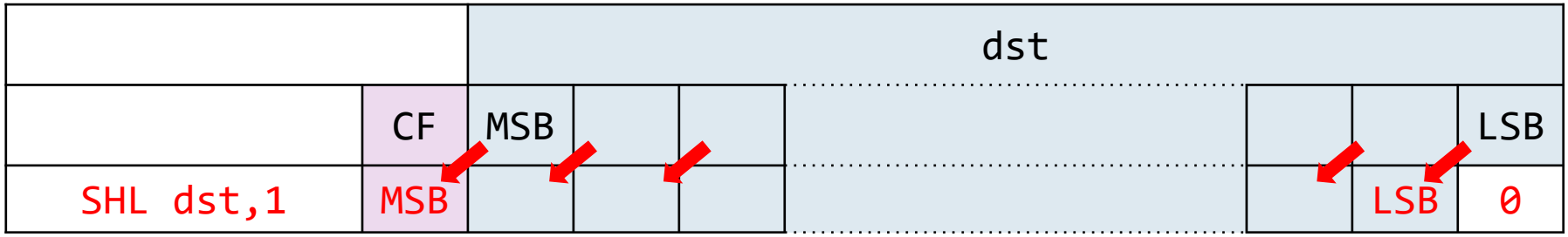
Příznak CF má hodnotu bitu, který byl poslední posunut „ven“ z cílového operandu.

reg ∈ {AL,BL,CL,DL,AH,BH,CH,DH,AX,BX,CX,DX,SI,DI,BP,SP,EAX,EBX,ECX,EDX,ESI,EDI,ESP,EBP}
mem = paměť, imm8 = přímý, 8bitový operand (konstanta)

SHL/SAL dst,src

```
SHL/SAL dst,src
for(int i=0; i < src; i++) {
  dst = dst + dst;
  // CF, SF, ZF, PF se nastaví dle výsledku součtu
}
```

```
SHL/SAL dst,src
for(int i=0; i < src; i++) {
  CF = MSB(dst); dst = dst << 1;
  // SF, ZF, PF se nastaví dle výsledku posunutí
}
```

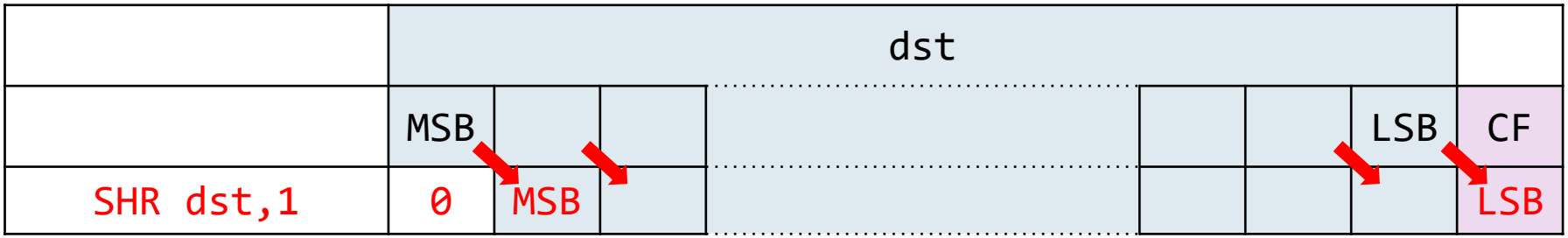


	CF	AX															
SHL AX(,1)	?	1	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1
	1	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1	0
	CF	BX															
SHL BX,2	?	0	0	1	0	0	1	0	1	1	1	0	1	1	1	1	0
	0	1	0	0	1	0	1	1	1	0	1	1	1	1	0	0	0

SHR dst,src

```
SHR dst,src
for(int i=0; i < src; i++) {
  CF = dst % 2;
  dst = dst / 2; // SHR: dělení bez znaménka
} // SF, ZF, PF se nastaví dle výsledku dělení
```

```
SHR dst,src
for(int i=0; i < src; i++) {
  CF = LSB(dst); dst = dst >> 1;
  // SF, ZF, PF se nastaví dle výsledku posunutí
}
```

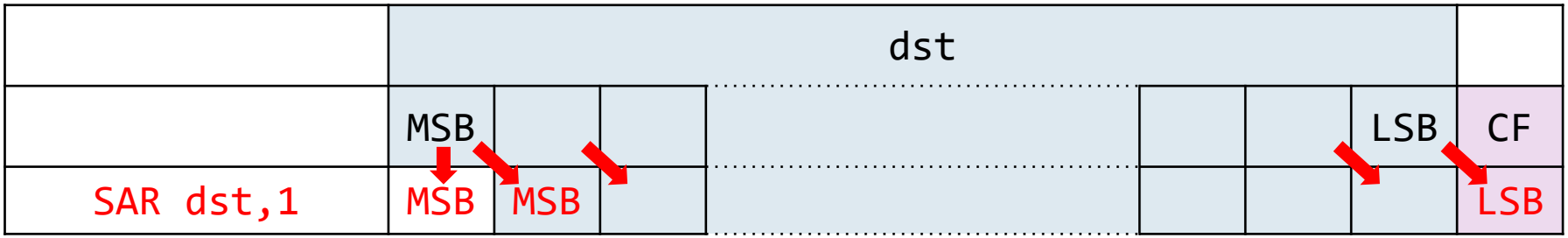


	AX																CF
SHR AX(,1)	1	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1	?
	0	1	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1
	BX																CF
SHR BX,2	1	0	1	0	0	1	0	1	1	1	0	1	1	1	1	0	?
	0	0	1	0	1	0	0	1	0	1	1	1	0	1	1	1	1

SAR dst,src

```
SAR dst,src
for(int i=0; i < src; i++) {
  CF = dst % 2;
  dst = dst / 2; // SAR: dělení se znaménkem
} // SF, ZF, PF se nastaví dle výsledku dělení
```

```
SAR dst,src
for(int i=0; i < src; i++) {
  CF=LSB(dst); s=MSB(dst)*2^sizeof(dst);
  dst = (dst >> 1) + s;
} // SF, ZF, PF se nastaví dle výsledku
```



	AX																CF
SAR AX(,1)	1	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1	?
	1	1	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1
	BX																CF
SAR BX,2	1	0	1	0	0	1	0	1	1	1	0	1	1	1	1	0	?
!!! ->	1	1	1	0	1	0	0	1	0	1	1	1	0	1	1	1	1

ROL, ROR, RCL, RCR
instrukce posuvů a rotací

mění příznaky:	OF	SF	ZF	AF	CF	PF

ROL dst,src (*Rotate Left*)
ROR dst,src (*Rotate Right*)
RCL dst,src (*Rotate through Carry Left*)
RCR dst,src (*Rotate through Carry Right*)

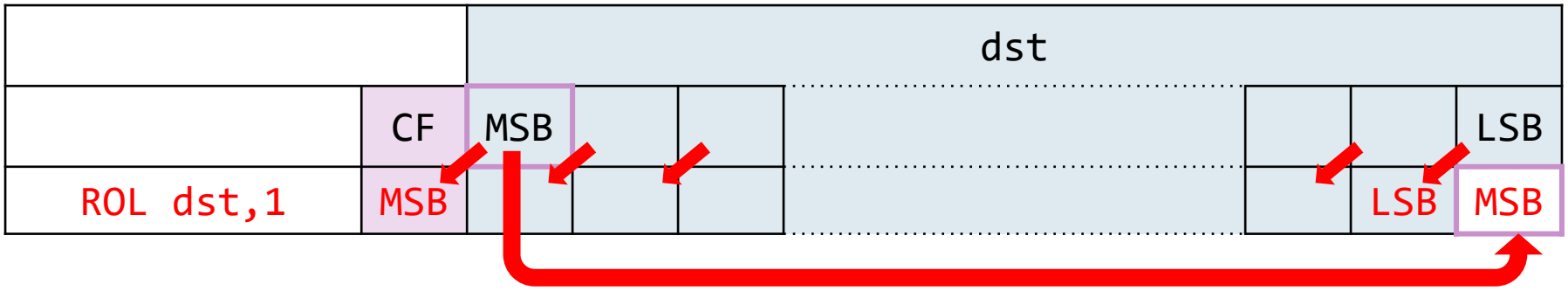
Příznak CF má hodnotu bitu, který byl poslední posunut „ven“ z cílového operandu v průběhu rotace.

```
1. ROL/ROR/RCL/RCR reg,CL
2. ROL/ROR/RCL/RCR reg(,imm8)  // bez imm8 = rotace o 1
3. ROL/ROR/RCL/RCR mem,CL
4. ROL/ROR/RCL/RCR mem(,imm8)  // bez imm8 = rotace o 1
```

```
ROL/ROR dst,src
for(i=0; i<src; i++) {
  ROL: tmp = MSB(dst);
       dst = dst * 2 + tmp;
  ROR: tmp = LSB(dst);
       dst = dst / 2 + tmp*2^sizeof(dst);
}
CF = tmp;
```

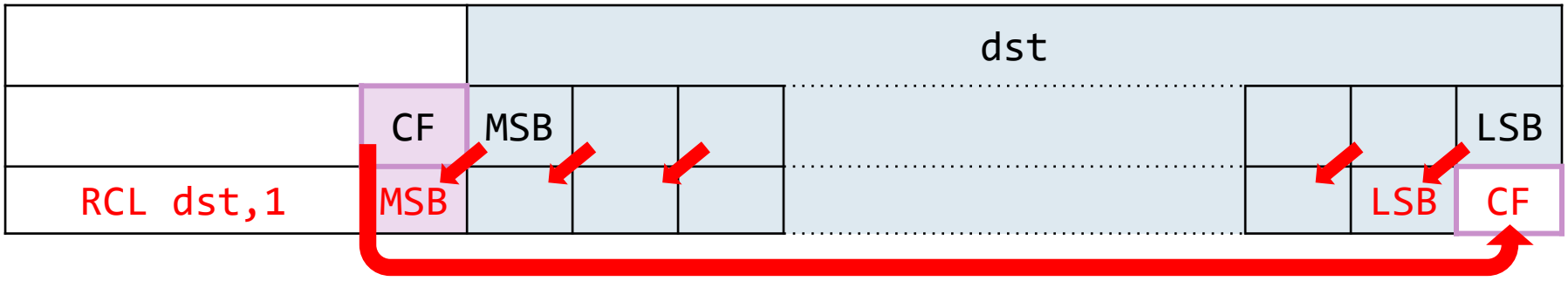
```
RCL/RCR dst,src
for(i=0; i<src; i++) {
  ROL: tmp = MSB(dst);
       dst = dst * 2 + CF;
  ROR: tmp = LSB(dst);
       dst = dst / 2 + CF*2^sizeof(dst);
  CF = tmp;
}
```

ROL dst,src



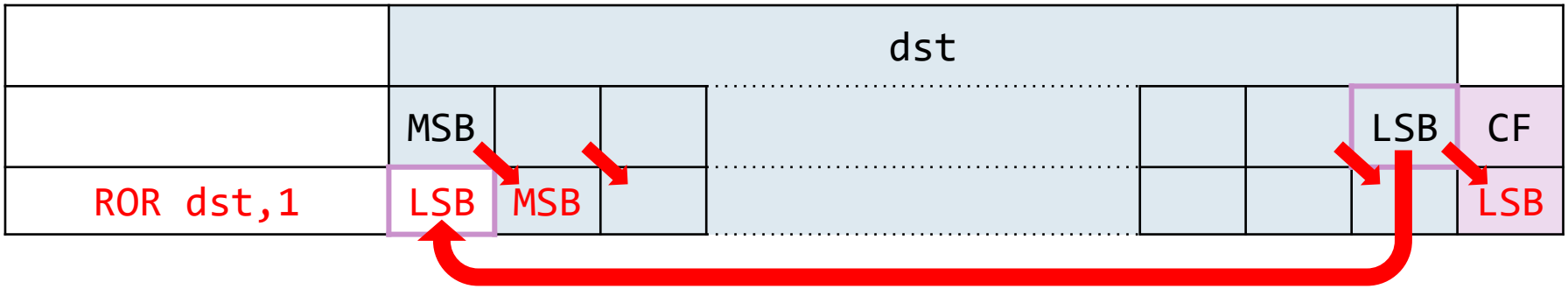
	CF	AX														
ROL AX(,1)	?	1	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1
	1	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1	1
	CF	BX															
ROL BX,2	?	0	0	1	0	0	1	0	1	1	1	0	1	1	1	1	0
	0	1	0	0	1	0	1	1	1	0	1	1	1	1	0	0	0

RCL dst,src



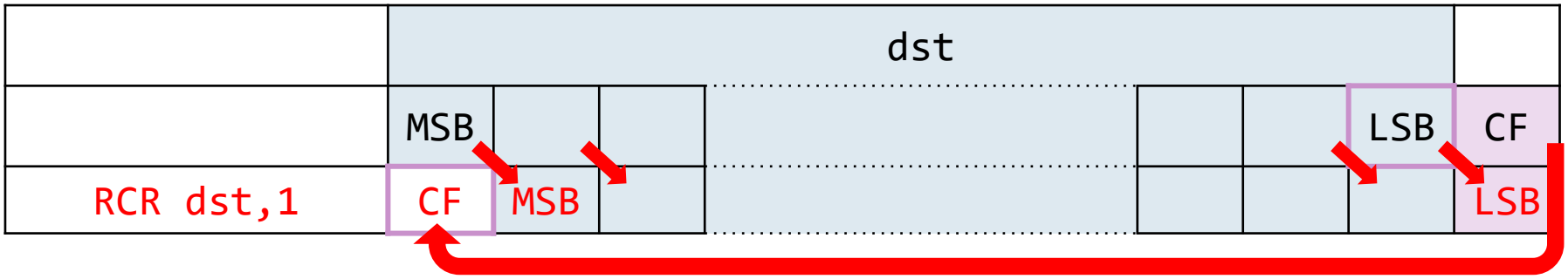
	CF	AX														
RCL AX(,1)	?	1	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1
	1	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1	?
	CF	BX															
RCL BX,2	?	0	0	1	0	0	1	0	1	1	1	0	1	1	1	1	0
	0	1	0	0	1	0	1	1	1	0	1	1	1	1	0	?	0

ROR dst,src

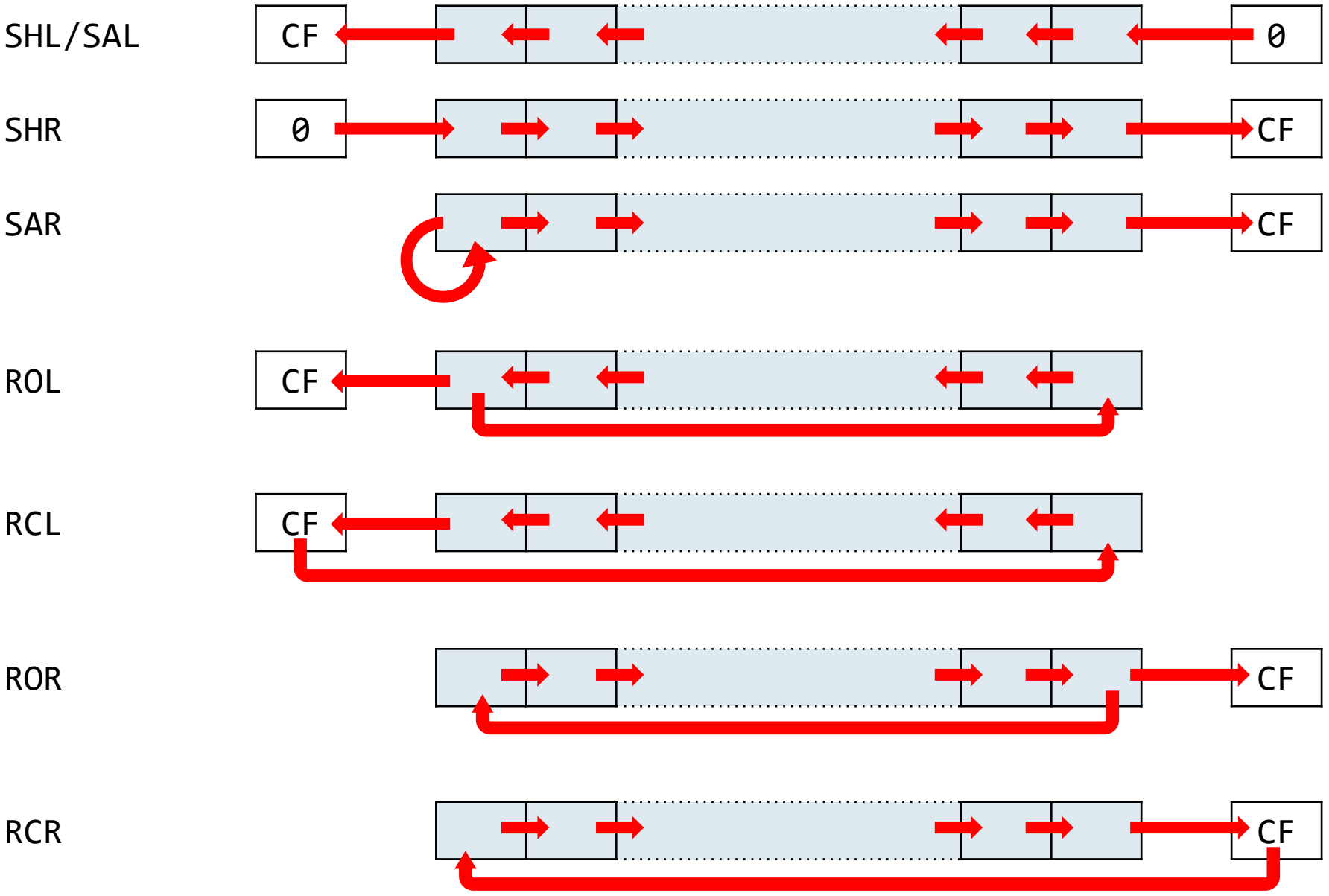


	AX																CF
ROR AX(,1)	1	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1	?
	1	1	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1
	BX																CF
ROR BX,2	1	0	1	0	0	1	0	1	1	1	0	1	1	1	1	0	?
	1	0	1	0	1	0	0	1	0	1	1	1	0	1	1	1	1

RCR dst,src



	AX																CF
RCR AX(,1)	1	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1	?
	?	1	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1
	BX																CF
RCR BX,2	1	0	1	0	0	1	0	1	1	1	0	1	1	1	1	0	?
	0	?	1	0	1	0	0	1	0	1	1	1	0	1	1	1	1


SHL/SAL
CF
0
SHR
0
CF
SAR
CF
ROL
CF
RCL
CF
ROR
CF
RCR
CF

MOVS, CMPS, SCAS, LODS, STOS

řetězové instrukce

(ne)mění příznaky	OF	SF	ZF	AF	CF	PF

Řetězové instrukce - princip

- Řetězové instrukce poznáme podle písmene „S“ na konci názvu:
  - MOV**S**, CMP**S**, SCA**S**, LOD**S**, STO**S**, IN**S**, OUT**S**
- Pracují buď:
  - pouze s pamětí (MOVS, CMPS)
  - s pamětí a registrem „A“ (**A**L, **A**X, E**A**X) (SCAS, LODS, STOS)
  - se vstupně/výstupními porty (INS/OUTS) a registrem „A“
- Ukazatele na data jsou v registrech DS:ESI a ES:EDI:
  - zdrojová data jsou uložena na adrese DS:ESI („SI“ = *Source Index*)
  - cílová data jsou uložena na adrese ES:EDI („DI“ = *Destination Index*)
  - pozor na SCAS – čte data z ES:EDI
- Instrukce pracují se slabikami, slovy nebo dvojslovy (**B**yte/**W**ord/**D**oubleword):
  - v NASM poznáme podle přípony B/W/D na konci instrukce
- Instrukce automaticky zvyšuje/snižuje indexové registry ESI a EDI:
  - řídí se příznakem DF (*Direction Flag*)
  - zvyšuje ⇔ DF == 0 (DF nastavíme na 0 instrukcí CLD)
  - snižuje ⇔ DF == 1 (DF nastavíme na 1 instrukcí STD)

MOVS
řetězová instrukce přesunová

```
MOVSB/MOVSW/MOVSD (Move String)
switch (sizeof instruction) {
  case B: type = byte; step = 1; break;
  case W: type = word; step = 2; break;
  case D: type = dword; step = 4;
}
direction = (DF)? -1 : 1;
tmpSrc = (type) DS:[ESI];
(type) ES:[EDI] = tmpSrc;
ESI += step*direction;
EDI += step*direction;
```

Nemění příznaky.

Mění hodnotu registrů ESI a EDI a hodnotu v paměti.

MOVSB DF=0	adresa *adr*	mem[*adr*]	
	0xFFFFFFFF	67	
	0x00001003	64	
	0x00001002	FF	
ES:EDI ↑	0x00001001	66	
ES:EDI	0x00001000	~~65~~	01
	0x00000103	22	
	0x00000102	33	
DS:ESI ↑	0x00000101	10	
DS:ESI	0x00000100	01	
	0x00000001	FF	
	0x00000000	12	

CMPS

řetězová instrukce aritmetická

```
CMPSB/CMPSW/CMPSD (Compare Strings)
switch (sizeof instruction) {
  case B: type = byte; step = 1; break;
  case W: type = word; step = 2; break;
  case D: type = dword; step = 4;
}
direction = (DF)? -1 : 1;
tmpSrc = (type) DS:[ESI];
tmpDst = (type) ES:[EDI];
ESI += step*direction;
EDI += step*direction;
EFLAGS ~ tmpSrc – tmpDst;
```

Mění příznaky EFLAGS.

Mění hodnotu registrů ESI a EDI.

CMPSB DF=0	adresa *adr*	[*adr*]	FLAGS
	0xFFFFFFFF	67	
	0x00001003	64	
	0x00001002	FF	
ES:EDI	0x00001001	66	
ES:EDI	0x00001000	65	1 – 0x65
			ZF=0
	0x00000103	22	CF=1
	0x00000102	33	OF=0
DS:ESI	0x00000101	10	SF=1
DS:ESI	0x00000100	01	
	0x00000001	FF	
	0x00000000	12	

SCAS

řetězová instrukce aritmetická

```
SCASB/SCASW/SCASD (Scan String)
switch (sizeof instruction) {
  case B: type = byte; step = 1; break;
  case W: type = word; step = 2; break;
  case D: type = dword; step = 4;
}
direction = (DF)? -1 : 1;
EFLAGS ~ AL/AX/EAX – (type) ES:[EDI];
EDI += step*direction;
```

Mění příznaky EFLAGS.

Mění hodnotu registru EDI.

SCASB DF=1,AL=1	adresa *adr*	[*adr*]	FLAGS
	0xFFFFFFFF	67	
	0x00001003	64	
	0x00001002	FF	
ES:EDI	0x00001001	66	1 – 0x66
ES:EDI	0x00001000	65	ZF=0
			CF=1
	0x00000103	22	OF=0
	0x00000102	33	SF=1
	0x00000101	10	
	0x00000100	01	
	0x00000001	FF	
	0x00000000	12	

LODS

řetězová instrukce přesunová

```
LODSB/LODSW/LODSD (Load String)
switch (sizeof instruction) {
  case B: type = byte; step = 1; break;
  case W: type = word; step = 2; break;
  case D: type = dword; step = 4;
}
direction = (DF)? -1 : 1;
AL/AX/EAX = (type) DS:[ESI];
ESI += step*direction;
```

Nemění příznaky.

Mění hodnotu registru ESI a AL/AX/EAX.

LODSB DF=0	adresa *adr*	mem[*adr*]	
	0xFFFFFFFF	67	
	0x00001003	64	
	0x00001002	FF	
	0x00001001	66	
	0x00001000	65	
	0x00000103	22	
	0x00000102	33	
DS:ESI ↑	0x00000101	10	
DS:ESI	0x00000100	01	AL=1
	0x00000001	FF	
	0x00000000	12	

STOS

řetězová instrukce přesunová

```
STOSB/STOSW/STOSD (Store String)
switch (sizeof instruction) {
  case B: type = byte; step = 1; break;
  case W: type = word; step = 2; break;
  case D: type = dword; step = 4;
}
direction = (DF)? -1 : 1;
(type) ES:[EDI] = AL/AX/EAX;
EDI += step*direction;
```

Nemění příznaky.

Mění hodnotu registru EDI a hodnotu v paměti.

STOSB DF=0, AL=1	adresa *adr*	mem[*adr*]	
	0xFFFFFFFF	67	
	0x00001003	64	
	0x00001002	FF	
ES:EDI ↑	0x00001001	66	
ES:EDI	0x00001000	~~65~~	01
	0x00000103	22	
	0x00000102	33	
	0x00000101	10	
	0x00000100	01	
	0x00000001	FF	
	0x00000000	12	

Řetězové instrukce bez řetězových instrukcí

```
SEGMENT .data
  string1 DB 11,20,32,55
  string2 DB 0,0,0,0
SEGMENT .text
  CLD
  MOV ESI,string1
  MOV EDI,string2
```

```
;MOVSB
  MOV AL,[ESI]
  MOV [EDI],AL
  INC ESI
  INC EDI
```

```
;CMPSB
  MOV AL,[ESI]
  MOV AH,[EDI]
  INC ESI
  INC EDI
  CMP AL,AH
```

```
;SCASB
  MOV AL,32
  MOV AH,[EDI]
  ADD EDI,1
  CMP AL,AH
```

```
;LODSB
  MOV AL,[ESI]
  INC ESI
```

```
;STOSB
  MOV [EDI],AL
  INC EDI
```

```
;MOVSW
  LODSW
  STOSW
```

```
;CMPSW
  MOV AX,[ESI]
  MOV BX,[EDI]
  ADD ESI,2
  ADD EDI,2
  CMP AX,BX
```

```
;SCASD
 MOV EAX,33
 MOV EBX,[EDI]
 ADD EDI,4
 CMP EAX,EBX
```

```
;LODSW
  MOV AX,[ESI]
  ADD ESI,2
```

```
;STOSD
 MOV [EDI],EAX
 ADD EDI,4
```

REP, REPE, REPZ, REPNE, REPNZ
prefixy řetězových instrukcí

```
REP instruction
do {
  instruction;
  ECX--;
} while (ECX != 0)
```

```
REPE/REPZ/REPNE/REPNZ instruction
do {
  instruction;
  ECX--;
} while (ECX != 0 && ZF == 1)
} while (ECX != 0 && ZF == 0)
```

Opakování řetězové instrukce - prefixy

```
SEGMENT .data
  string1 DB 11,20,32,55
  string2 DB 11,20,0,0
SEGMENT .text
  CLD
  MOV ESI,string1
  MOV EDI,string2
  MOV ECX,4
```

```
;REP MOVSB
cyklus:
  MOV AL,[ESI]
  MOV [EDI],AL
  INC ESI
  INC EDI
  LOOP cyklus
```

```
;REPE CMPSB
cyklus:
  MOV AL,[ESI]
  MOV AH,[EDI]
  INC ESI
  INC EDI
  CMP AL,AH
  LOOPE cyklus
```

```
;REPNE SCASB
cyklus:
  MOV AL,11
  MOV AH,[EDI]
  ADD EDI,1
  CMP AL,AH
  LOOPNE cyklus
```

```
;REP MOVSB – neumožní úpravy
cyklus:
  LODSB
;  … tady můžu udělat něco s AL,
;  … na rozdíl od MOVS
  STOSB
  LOOP cyklus
```

Řetězové instrukce – k čemu slouží?

MOVS = kopírování řetězců v paměti z jednoho místa na druhé (ve spojení s REP)

CMPS = porovnávání řetězců (ve spojení s REPE/REPNE)

SCAS = vyhledávání znaku v řetězci (ve spojení s REPE/REPNE)

LODS+STOSB = kopírování řetězců v paměti z jednoho místa na druhé s možností modifikace hodnot

```
SEGMENT .data
    string1 DB „ahoj“
    string2 DB ’a’,’h’,0,0

SEGMENT .text
    …
    CLD
    MOV ESI,string1
    MOV EDI,string2
    MOV ECX,4
```

pokračování viz některý z rámečků vpravo ->

```
; kopie řetězců
REP MOVSB
```

```
; porovnání řetězců
REPE CMPSB ; ECX = ?, EFLAGS = ?
```

```
; vyhledání hodnoty/znaku ’h’ v řetězci string2
      MOV AL,’h’
REPNE SCASB ; ECX = ?, EFLAGS = ?
```

```
; kopie řetězce a zároveň změna velkých znaků
; na malé znaky (pokud byly malé, nic se nemění)
cyklus:
    LODSB
    OR AL,0x20
    STOSB
    LOOP cyklus
```