§ 6.2. Modul de adresare circulară

- Câmpurile BK0 și BK1 din registrul AMR specifică dimensiunile blocului folosit pentru adresarea circulară.

Instrucțiunile LD și ST

- La fel ca la adresarea liniară operandului *offsetR*/*cst* este deplasat spre stânga cu 3, 2, 1 sau 0 în funcție de lungimea datei, iar apoi este realizată o adunare sau o scădere din *baseR* în scopul determinării adresei finale. În plus la adresarea circulară biții *N* până la 0 ai rezultatului sunt modificați, iar biții 31 până la (*N* + 1) rămân nemodificați după operația aritmetică efectuată în scopul determinării adresei finale. Adresa obținută este limitată în domeniul 2^{N+1}, în funcție de dimensiunea operandului *offsetR*/*cst*.

- Dimensiunea bufferului circular din registrul AMR nu este scalată. De aceea, pentru a realiza, o adresare circulară a unui bloc de 8 cuvinte, dimensiunea 32 trebuie specificată, i.e. *N* = 4.

- Exemplu:

AMR = 0004 0001h ⇒ Registrul BK0 și *N* = 4 (i.e. bufferul are o lungime de 32 x 8 biți, 16 x 16 biți sau 8 x 32 biți)

```
LDW         .D1         *++A4[9],A1
```

Before LDW		1 cycle after LDW[1]		5 cycles after LDW	
A4	0000 0100h	A4	0000 0104h	A4	0000 0104h
A1	xxxx xxxxh	A1	xxxx xxxxh	A1	1234 5678h
mem 104h	1234 5678h	mem 104h	1234 5678h	mem 104h	1234 5678h

9h cuvinte = 24h bytes → 24h bytes = (4h + 20h) bytes

domeniul 100h – 11Fh → 124h – 20h = 104h

❒ Instrucțiunile ADDA și SUBA

- La fel ca la adresarea liniară operandului *offsetR*/*cst* este deplasat spre stânga cu 3, 2, 1 sau 0 în funcție de lungimea datei, iar apoi este realizată o adunare sau o scădere din *baseR* în scopul deerminării adresei finale. În plus la adresarea circulară biții *N* până la 0 ai rezultatului sunt modificați, iar biții 31 până la (*N* + 1) sunt nemodificați după operația aritmetică efectuată în scopul determinării adresei finale. Adresa obținută este limitată în domeniul 2^{N+1}, în funcție de dimensiunea operandului *offsetR*/*cst*.

- Dimensiunea bufferului circular din registrul AMR nu este scalată. De aceea, pentru a realiza, o adresare circulară a unui bloc de 8 cuvinte, dimensiunea 32 trebuie specificată, i.e. *N* = 4.

Exemplu:

AMR = 0004 0001h ⇨ Registrul BK0 și *N* = 4 (i.e. bufferul are o lungime de 32 x 8 biți, 16 x 16 biți sau 8 x 32 biți)

```
ADDAH       .D1       A4,A1,A4
```

	Before ADDAH		1 cycle after ADDAH[1]
A4	0000 0100h	A4	0000 0106h
A1	0000 0013h	A1	0000 0013h

13h x 16 biți = 26h bytes → 26h bytes = (6h + 20h) bytes

domeniul 100h – 11Fh → 126h – 20h = 106h

❒ Considerații privind adresarea circulară cu memorie nealiniată

- Când adresarea circulară este validată, modificarea adresei și accesul la memorie se realizează în același mod ca și pentru secvența echivalentă de accesare a unui octet (byte).
- Lungimea bufferului circular trebuie să fie cel puțin egală cu cea a datei care va fi accesată.
- Pentru accesarea nealiniată a unui buffer circular se aplică calculul adresării circulare la adresele de memorie adiacente. Ca urmare, accesul nealiniat în apropiere de limita bufferului circular, conduce la citirea corectă a datelor situate la ambele capete ale bufferului circular. Se produce, astfel, bufferul circular prin închiderea capetelor sale.

- Exemplu: - buffer circular de lungime 16 octeți.

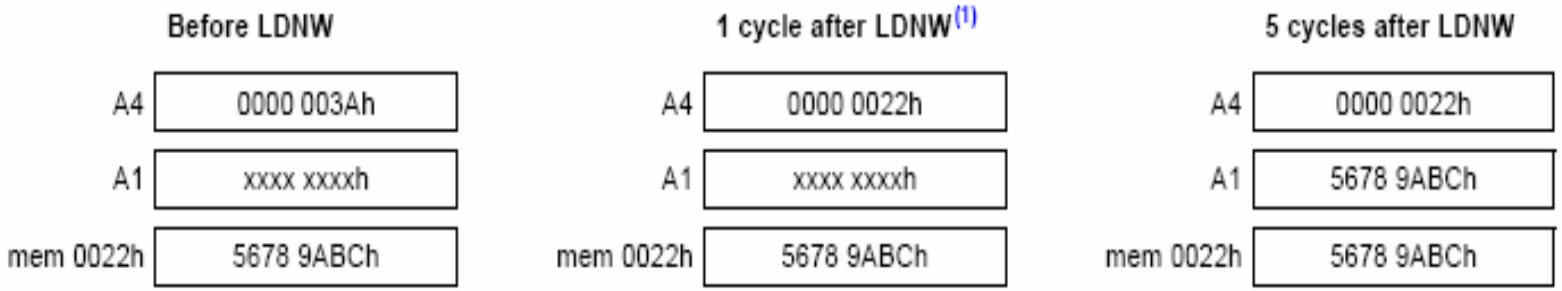
1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3	3
7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8
x	x	x	x	x	x	x	x	x	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	x	x	x	x	x	x	x	x	x

2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8
h	i	j	k	l	m	n	o	p	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	a	b	c	d	e	f	g	h	i

AMR = 0004 0001h

Registrul BK0 și *N* = 4 (i.e. bufferul are o lungime de 32 x 8 biți, 16 x 16 biți sau 8 x 32 biți)

```
LDNW    .D1         *++A4[2],A1
```



2h x 32 biți = 8h bytes

003Ah + 8h = 42h

42h – 20h = 22h

§ 6.3. Sintaxa folosită la generarea adreselor pentru instrucțiunile de încărcare /stocare

- Sintaxa folosită în limbajul de asamblare pentru o adresă indirectă către o locație de memorie

Addressing Type	No Modification of Address Register	Preincrement or Predecrement of Address Register	Postincrement or Postdecrement of Address Register
Register indirect	*R	*++R	*R++
		*- -R	*R- -
Register relative	*+R[*ucst5*]	*++R[*ucst5*]	*R++[*ucst5*]
	*-R[*ucst5*]	*- -R[*ucst5*]	*R- -[*ucst5*]
Register relative with 15-bit constant offset	*+B14/B15[*ucst15*]	not supported	not supported
Base + index	*+R[*offsetR*]	*++R[*offsetR*]	*R++[*offsetR*]
	*-R[*offsetR*]	*- -R[*offsetR*]	*R- -[*offsetR*]

- Observație: pentru încărcarea unei offset de valoare mare se utilizează registrul B14 sau B15 ca registru de bază și o constantă fără semn de 15 biți (*ucst15*) ca și offset.

- Adresa de memorie este formată folosind registrul adresei de bază (*baseR*) și, opțional, un offset care este fie un registru (*offsetR*) sau o constantă fără semn de 5 biți (*ucst5*).
- Opțiunile generatorului de adresă

Mode Field				Syntax	Modification Performed
0	0	0	0	*-R[*ucst5*]	Negative offset
0	0	0	1	*+R[*ucst5*]	Positive offset
0	1	0	0	*-R[*offsetR*]	Negative offset
0	1	0	1	*+R[*offsetR*]	Positive offset
1	0	0	0	*- -R[*ucst5*]	Predecrement
1	0	0	1	*++R[*ucst5*]	Preincrement
1	0	1	0	*R- -[*ucst5*]	Postdecrement
1	0	1	1	*R++[*ucst5*]	Postincrement
1	1	0	0	*--R[*offsetR*]	Predecrement
1	1	0	1	*++R[*offsetR*]	Preincrement
1	1	1	0	*R- -[*offsetR*]	Postdecrement
1	1	1	1	*R++[*offsetR*]	Postincrement

❍ Exemplu:

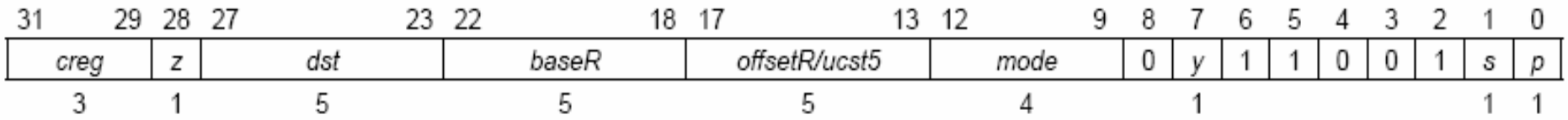
- LDW – LoaD Word from memory with a 5-bit unsigned constant offset or register offset

Sintaxă:

- cu registru offset: LDW (.unit)*+*baseR[offsetR], dst*
- cu constantă fără semn de 5 biți: LDW (.unit)*+*baseR[ucst5], dst*

unit = .D1 sau .D2;
baseR = registru de bază;
offsetR = registru de offset;
ucst5 = constantă fără semn de 5 biți;
dst = destinație (registru de uz general).

Cuvânt de cod:



- *offsetR* și *baseR* trebuie să fie în aceeași set de registre din aceeați cale ca și unitatea .D utilizată: *y* = 0 selectează unitatea .D1 (*offsetR* și *baseR* sunt în setul de registre A), iar *y* = 1 selectează unitatea .D1 (*offsetR* și *baseR* sunt în setul de registre B).
- *offsetR*/*ucst5* este scalat printr-o deplasare spre stânga cu 2 biți, după care este adunat la sau scăzut din *baseR*.În funcție de opțiunile generatorului de adrese (preincrementare, predecrementare, post incrementare, postdecrementareoffset pozitiv sau offset negativ) rezultatul calculului este adresa locației de memorie. Pentru opțiunile de postincrementare și postdecrementare valoarea lui *baseR* dinainainte de adunare sau scădere este adresa locației de memorie.

- Modul de adressare liniară este implicit. Pentru adresarea circulară trebuie programat corespunzător registrul AMR.
- Regsistrul dst poate fi în oricare set de registre, indiferent de unitatea .D, *baseR* și *offsetR* utilizați. Bitul *s* stabilește unde va fi încărcat registrul *dst*: dacă *s* = 0 (1), atunci registrul *dst* va fi încărcat în setul de registre A (B).
- Dacă între paranteze nu este specificat nici un registru, atunci incrementarea/decrementarea este realizată implicit cu o unitate. Sintaxa **baseR* nu modifică registrul *baseR*. Sintaxa **baseR[a]* reprezintă un offset egal cu *a* cuvinte, iar sintaxa **baseR(a)* reprezintă un offset egal cu *a* octeți.

❍ Exemple:

Example 1

```
LDW .D1     *A10,B1
```

	Before instruction		1 cycle after instruction		5 cycles after instruction
B1	0000 0000h	B1	0000 0000h	B1	21F3 1996h
A10	0000 0100h	A10	0000 0100h	A10	0000 0100h
mem 100h	21F3 1996h	mem 100h	21F3 1996h	mem 100h	21F3 1996h

Example 2

```
LDW .D1     *A4++[1],A6
```

	Before instruction		1 cycle after instruction		5 cycles after instruction
A4	0000 0100h	A4	0000 0104h	A4	0000 0104h
A6	1234 4321h	A6	1234 4321h	A6	0798 F25Ah
AMR	0000 0000h	AMR	0000 0000h	AMR	0000 0000h
mem 100h	0798 F25Ah	mem 100h	0798 F25Ah	mem 100h	0798 F25Ah
mem 104h	1970 19F3h	mem 104h	1970 19F3h	mem 104h	1970 19F3h

Example 3

```
LDW .D1     *++A4[1],A6
```

	Before instruction		1 cycle after instruction		5 cycles after instruction
A4	0000 0100h	A4	0000 0104h	A4	0000 0104h
A6	1234 5678h	A6	1234 5678h	A6	0217 6991h
AMR	0000 0000h	AMR	0000 0000h	AMR	0000 0000h
mem 104h	0217 6991h	mem 104h	0217 6991h	mem 104h	0217 6991h

Example 4

```
LDW .D1 *++A4[A12],A8
```

	Before instruction		1 cycle after instruction		5 cycles after instruction
A4	0000 40B0h	A4	0000 40C8h	A4	0000 40C8h
A8	0000 0000h	A8	0000 0000h	A8	DCCB BAA8h
A12	0000 0006h	A12	0000 0006h	A12	0000 0006h
AMR	0000 0000h	AMR	0000 0000h	AMR	0000 0000h
mem 40C8h	DCCB BAA8h	mem 40C8h	DCCB BAA8h	mem 40C8h	DCCB BAA8h

Example 5

```
LDW .D1 *++A4(8),A8
```

	Before instruction		1 cycle after instruction		5 cycles after instruction
A4	0000 40B0h	A4	0000 40B8h	A4	0000 40B8h
A8	0000 0000h	A8	0000 0000h	A8	9AAB BCCDh
AMR	0000 0000h	AMR	0000 0000h	AMR	0000 0000h
mem 40B8h	9AAB BCCDh	mem 40B8h	9AAB BCCDh	mem 40B8h	9AAB BCCDh

§ 7. Funcționarea pipeline

- Fazele funcționării pipeline sunt împărțite în trei etape: extragere, decodificare și execuție.
- Etapa de extragere are 4 faze, cea de decodificare are 2 faze, iar cea de execuție are un număr variabil de faze în funcție de tipul instrucțiunii.

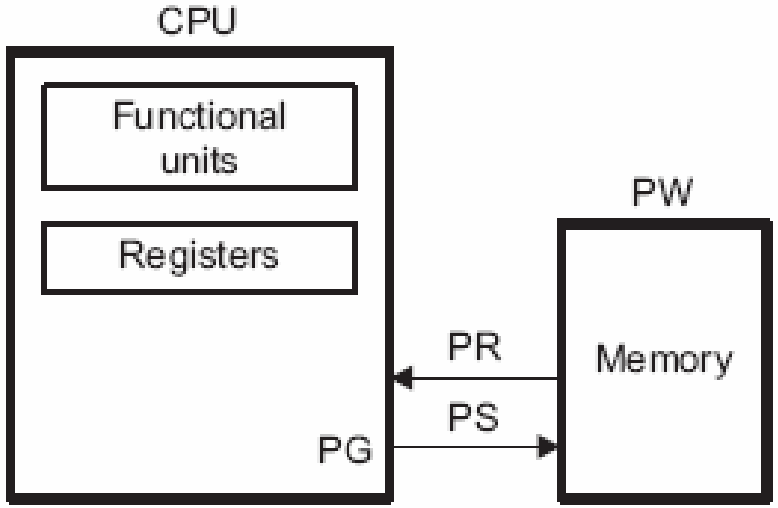
❒ Etapa de extragere

- Fazele etapei de extragere sunt:

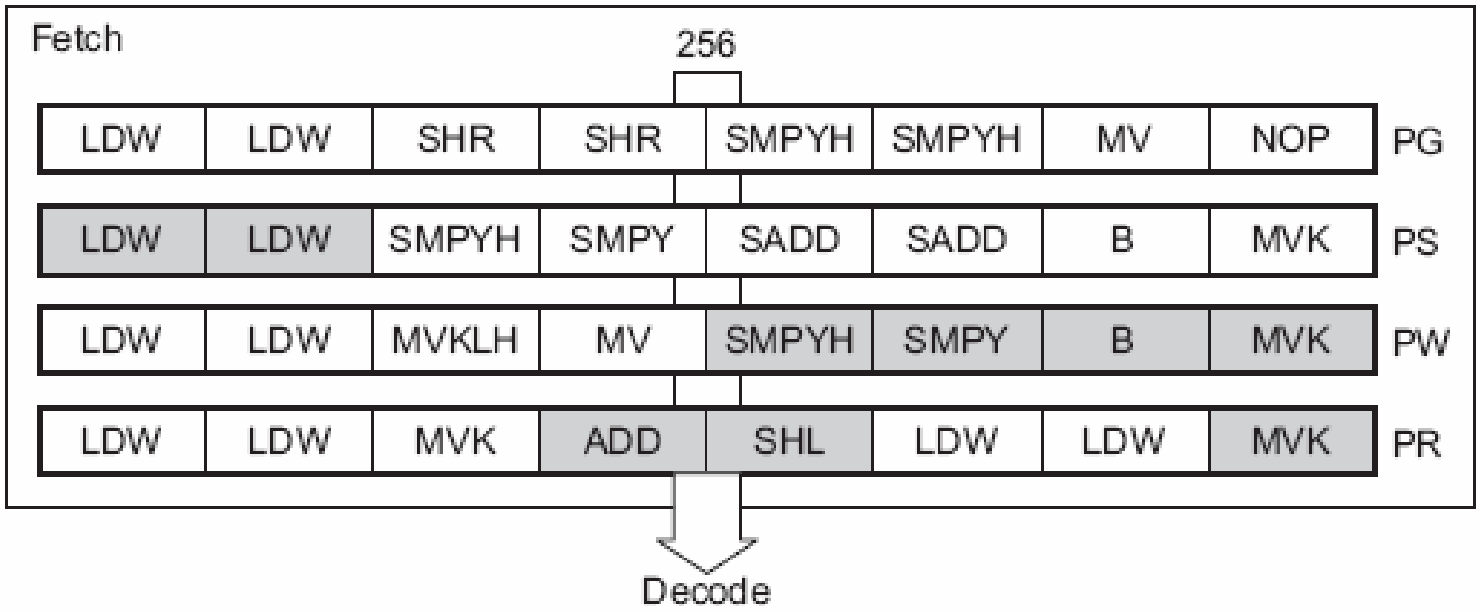
- *generarea adresei locației din memoria de program* (PG – Program address Generate);
- *trimiterea adresei locației din memoria de program* (PS – Program adress Send);

- *așteptarea stabilirii accesului la memoria de program* (PW – Program access ready Wait);
- *recepționarea pachetului extras din memoria program* (PR – Program fetch packet Receive).

- În faza PG adresa locației din memoria de program este generată în CPU.
- În faza PS adresa locației din memoria de program este transmisă memoriei.
- În faza PW are loc citirea memoriei.
- În faza PR pachetul extras este recepționant de către CPU.



Exemplu:



Primul pachet extras (faza PR) conține 4 pachete de execuție, al doilea și al treilea (fazele PW și PS) conțin 2 pachete fiecare, iar ultimul pachet extras (faza PG) conține un singur pachet de execuție de 8 instrucțiuni.

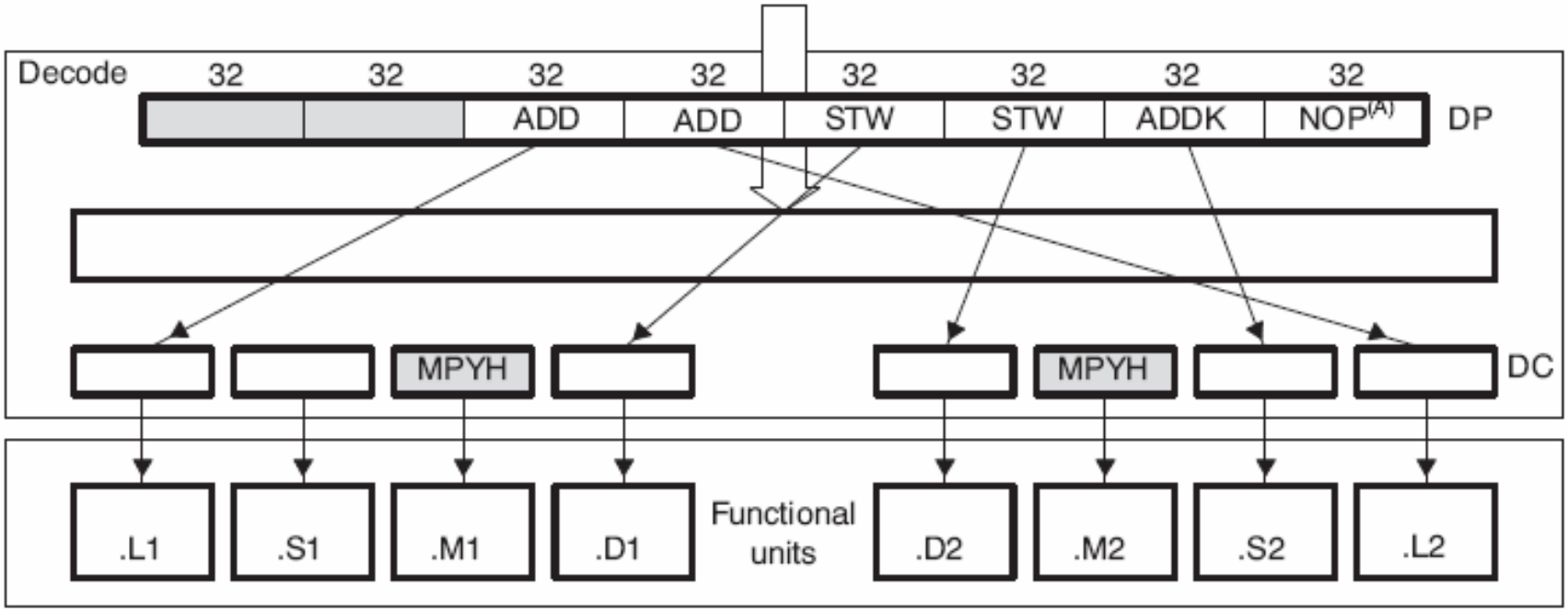
❒ Etapa de decodificare

- Fazele etapei de decodificare sunt:
  - *trimiterea instrucțiunii* (DP – Instruction DisPatch);
  - *decodificarea instrucțiunii* (DC – Instruction DeCode).

- În faza DP pachetele extrase sunt împărțite în pachete de execuție. Pachetele de execuție conțin o instrucțiune sau două până la opt instrucțiuni paralele. În timpul fazei DP se stabilesc pentru instrucțiunile dintr-un pachet de execuție unitățile lor funcționale.

- În faza DC registrele sursă, registrele destinație și căile asociate sunt decodificate pentru execuția instrucțiunilor în unitățile funcționale.

Exemplu:



A NOP is not dispatched to a functional unit.

- Pachetul extras conține două pachete de execuție care urmează a fi procesate în cele două faze. Ultimele 6 instrucțiuni sunt în paralel și formează un pachet de execuție. Acest pachet de execuție se află în faza DP. Săgețile indică unitatea funcțională stabilită pentru fiecare instrucțiune. Instrucțiunii NOP nu i se atribuie nici o unitate funcțională deoarece nu are asociată nici o execuție.

- Primele două instrucțiuni din pachetul extras (casuțele cu gri) sunt executate în paralel. Ele au fost în faza DP în ciclul anterior. Acest pachet de execuție conține două instrucțiuni MPYH , care se află în acest moment îm faza DC. Se observă că în această fază nu sunt instrucțiuni decodificate în unitățile funcționale .L, .S și .D.

❒ Etapa de execuție

- Porțiunea de execuție din cadrul pipeline este subdivizată în 5 faze (E1 – E5). Fiecare instrucțiune necesită un anumit număr din aceste faze pentru execuție.

- Exemplu: