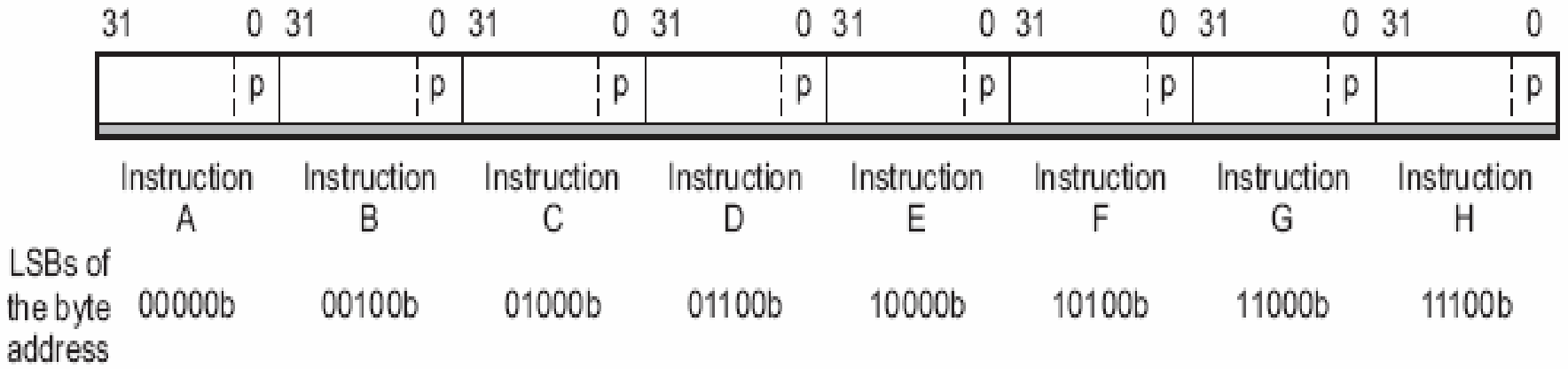
§ 4. Operarea în paralel

- Sunt extrase simultan din memoria de program 8 cuvinte de 32 de biți corespunzătoare a 8 instrucțiuni. Cele 8 cuvinte de 32 de biți constituie un *pachet extras*.

- Formatul pechetului extras este prezentat în figura de mai jos:



- Execuția instrucțiunilor este parțial controlată, în fiecare instrucțiune, de un bit – bitul *p*.
- Bitul *p* (bitul 0) aferent unei instrucțiuni determină dacă instrucțiunea este execută în paralel cu altă instrucțiune.
- Biții *p* sunt scanați de la stânga la dreapta (de la cea mai mică adresă la cea mai mare).
- Exemple:

  - dacă bitul *p* din instrucțiunea *I* este 1, atunci instrucțiunea (*I* + 1) este executată în paralel (în acelaşi ciclu mașină) cu instrucțiunea *I*;

  - dacă bitul *p* din instrucțiunea *I* este 0, atunci instrucțiunea (*I* + 1) este executată după instrucțiunea *I* (în ciclul mașină următor).
- Toate instucțiunile care sunt executate în paralel constituie un pachet de execuție. Un pachet de execuție poate conține până la 8 instrucțiuni. Fiecare instrucțiune din pachet trebuie să utilizeze o unitate funcțională diferită.

- Valorile biților *p* stabilesc următoarele secvențe de execuții pentru cele 8 instrucțiuni:
  - Execuția serială – instrucțiunile sunt executate secvențial

The eight instructions are executed sequentially.

This *p*-bit pattern:

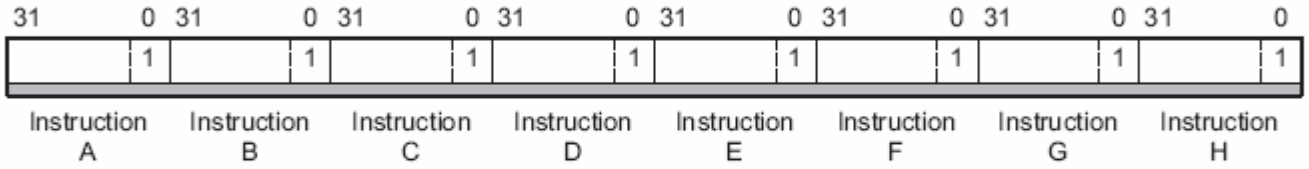
results in this execution sequence:

Cycle/Execute Packet	Instructions
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H

❒ Execuția paralelă – instrucțiunile sunt executate în paralel

All eight instructions are executed in parallel.

This *p*-bit pattern:

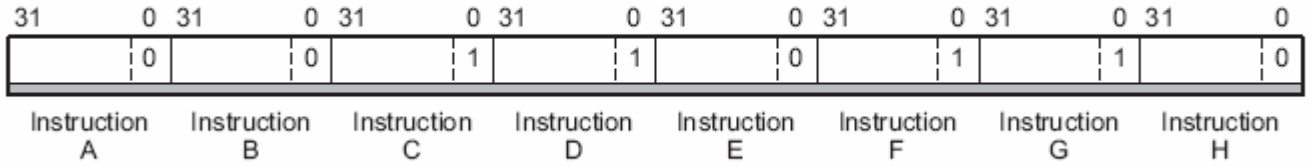


results in this execution sequence:

Cycle/Execute Packet	Instructions							
1	A	B	C	D	E	F	G	H

❒ Execuția parțial serială – instrucțiunile sunt executate în serie și în paralel

This *p*-bit pattern:



results in this execution sequence:

Cycle/Execute Packet	Instructions		
1	A		
2	B		
3	C	D	E
4	F	G	H

- În limbaj de asamblare – barele verticale || plasate în fața unei instrucțiuni semnifică faptul că instrucțiunea respectivă este executată în paralel cu instrucțiunea anterioară.
- Pentru exemplul prezentat la execuția parțial serială:

```
    Instrucțiunea A
    Instrucțiunea B
    Instrucțiunea C
||  Instrucțiunea D
||  Instrucțiunea E
    Instrucțiunea F
||  Instrucțiunea G
||  Instrucțiunea H
```

- Observație: dacă în urma unei instrucțiuni de salt, saltul are loc în mijlocul pachetului de execuție, atunci toate instrucțiunile de la adrese mai mici sunt ignorate.
- Exemplu: dacă saltul are loc la adresa corespunzătoare instrucțiunii D, atunci doar instrucțiunile D și E sunt executate. Instrucțiunea C, chiar dacă este în același pachet de execuție, este ignorată. Instrucțiunile A și B sunt, de asemenea, ignorate deoarece ele se află în pachete care sunt executate anterior.

~~Instrucțiunea A~~
~~Instrucțiunea B~~
~~Instrucțiunea C~~
|| Instrucțiunea D
|| Instrucțiunea E
~~Instrucțiunea F~~
~~|| Instrucțiunea G~~
~~|| Instrucțiunea H~~

§ 5. Operări condiționate

- Execuția instrucțiunilor pentru majoritatea dintre acestea poate fi condiționată.
- Execuția condiționată a unei instrucțiuni este stabilită de 3 biți (*creg*) care specifică registrul folosit pentru execuția condiționată și de un bit (*z*) care specifică dacă testul este pentru 0 sau pentru 1. Cei mai semnificativi 4 biți ai cuvântul de cod corespund lui *creg* (biții 29-31) și lui *z* (bitul 28).
- Conținutul registrului pentru execuția condiționată este testat la începutul etapei E1 a funcționării pipeline. Dacă *z* = 1(0), atunci execuția instrucțiunii are loc dacă conținutul registrului pentru execuția condiționată este egal cu zero (unu). Dacă *creg* = 0 și *z* = 0, atunci instrucțiunea este întotdeauna executată.

Registrele folosite pentru execuția condiționată a unei instrucțiuni

Specified Conditional Register		*creg*			*z*
	Bit:	31	30	29	28
Unconditional		0	0	0	0
Reserved		0	0	0	1
B0		0	0	1	z
B1		0	1	0	z
B2		0	1	1	z
A1		1	0	0	z
A2		1	0	1	z
A0		1	1	0	z
Reserved		1	1	x(1)	x(1)

(1) x can be any value.

- În limbajul de asamblare instrucțiunile a căror execuție este condiționată sunt marcate prin paranteze drepte [ ], în interiorul cărora se scrie registru folosit pentru execuția condiționată.

- Exemple:

[B0] ADD .L1 A1, A2, A3

- instrucțiunea ADD este executată dacă conținutul registrului B0 nu este egal cu zero;

[! B0] ADD .L1 A1, A2, A3

- instrucțiunea ADD este executată dacă conținutul registrului B0 este egal cu zero;

§ 6. Modurile de adresare

- Modurile de adresare ale procesorului sunt: liniară, circulară folosind registrul BK0 și circulară folosind registrul BK1.
- Modurile de adresare sunt specificate prin intermediul registrului pentru modurile de adresare AMR (Adressing Mode Register).
- Toate registrele pot realiza adresarea liniară. În schimb, adresarea circulară poate fi realizată numai de registrele A4-A7 (unitatea funcțională .D1) și de registrele B4-B7 (unitatea funcțională .D2).
- Instucțiunile:

  LDB(U)/LDH(U)/LDW, STB/STH/STW, LDNW, STNDW, STNW, LDDW, STDW, ADDAB/ADDAH/ADDAW/ADDAD și **SUBAB/SUBAH/SUBAW,** utilizează registrul AMR pentru determinarea calculelor adresei care trebuie realizate pentru registre.

§ 6.1. Modul de adresare liniară

❒ Instrucțiunile LD și ST

❍ Pentru instrucțiunile de încărcare (LD) și stocare (ST), în modul de adresare liniară sunt realizate deplasări ale operandului *offsetR*/*cst* (*offsetR* este registru de offset, iar *cst* este o constantă) spre stânga cu 3, 2, 1 sau 0 pentru cuvânt dublu (2x32 biți), cuvânt (32 biți), jumătate de cuvânt (16 biți) sau byte (8 biți), iar apoi este realizată o adunare sau o scădere la *baseR* (registru adresei de bază) funcție de operația specificată. Instrucțiunile **LDNW** și **STNDW** permit, de asemenea, deplasări nescalate. În modul fără scalare, operandul *offsetR*/*cst* nu este deplasat înainte de operația de adunare sau scădere din *baseR*.

- Pentru opțiunile de generare a adreselor – preincrementare, predecrementare, offset pozitiv, ofsset negativ, rezultatul calculului este adresa de memorie care va fi accesată.

- Pentru operațiile de postincrementare sau postdecrementare valoarea *baseR* dinaintea operației de adunare sau scădere este adresa de memorie care va fi accesată.

Instrucțiunile ADDA și SUBA

- Pentru instrucțiunile de adunare și scădere cu numere întregi , în modul de adresare liniară operandul *scr1*/*cst* (*scr1* – sursa 1) este deplasat spre stânga cu 3, 2, 1 sau 0 pentru cuvânt dublu, cuvânt, jumătate de cuvânt sau byte, iar apoi este realizată operația de adunare sau de scădere specificată.

§ 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