仮想機械(2)

仮想マシン

- 復習
- 仮想マシンの概要
- hsm仮想マシン

Hsm (HiStackMachine)の概要(1)

- 演習で用いる仮想機械(スタックマシン）
- 構成

プログラムP

・命令列の置き場

プログラムカウンタ(pc)

・次に実行する命令を指示

スタック(S)

・演算対象（被演算数、演算結果）を置く

・記憶域

スタックポインタ(sp)

・スタックトップを指す

フレームポインタ(fp)

・関数（手続き）のフレームの開始アドレス（後の講義で説明）

Hsm (HiStackMachine)の概要(2)

- 命令セット

  (1) ロード・ストア命令

  ・ロード命令：スタックトップに値を置く。

  ・ストア命令：記憶域として確保した所に値を保存する。

  ・記憶域の確保、開放の命令

  (2) 演算命令

  ・算術演算、関係演算。(論理演算はない)

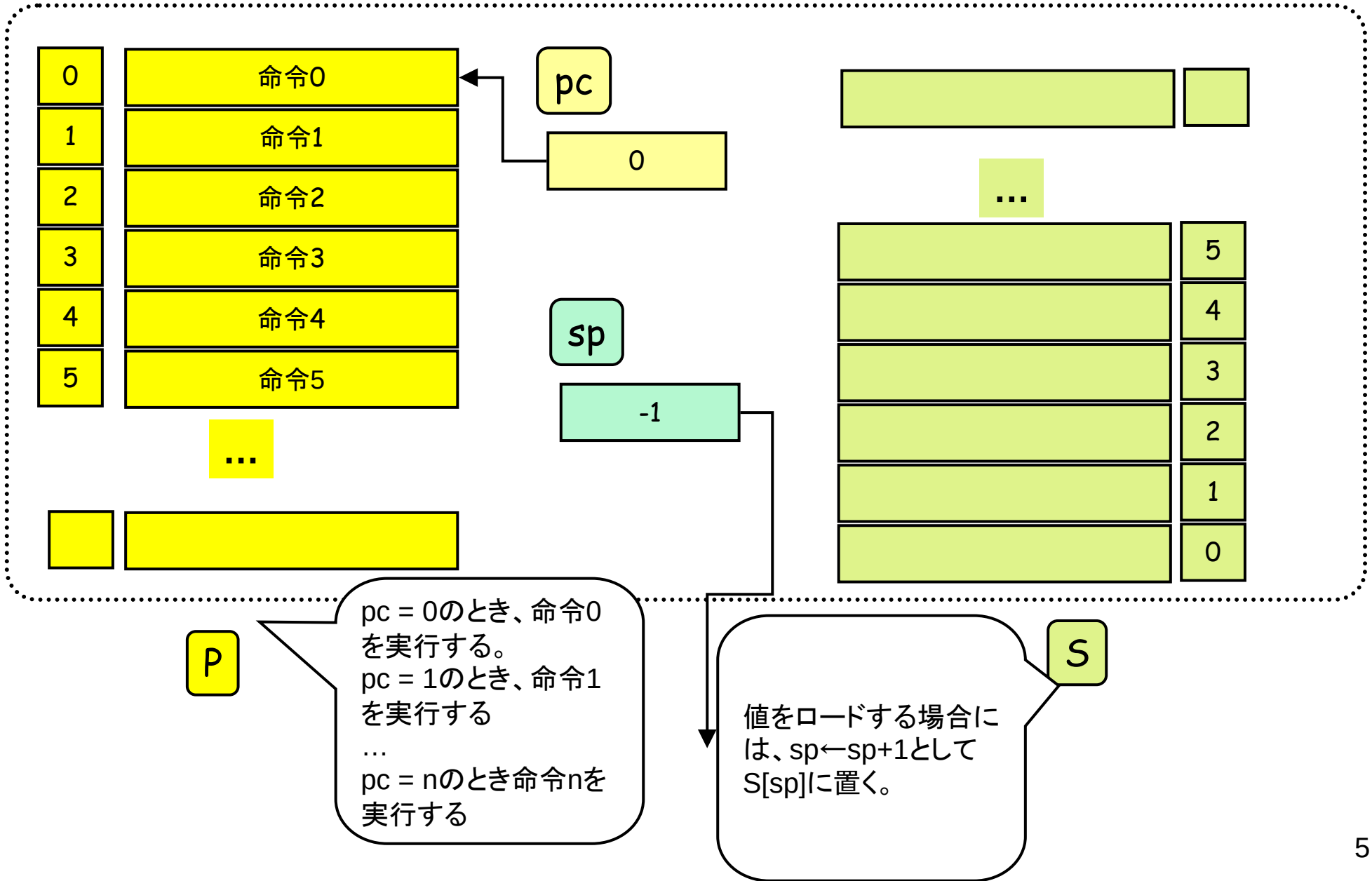
  (3) ジャンプ命令、制御命令

  ・無条件ジャンプ、条件ジャンプ、停止命令

  (4) 入出力命令

  ・入力、出力

hsmの構成図


0
命令0
1
命令1
2
命令2
3
命令3
4
命令4
5
命令5
...
pc
0
sp
-1
...
5
4
3
2
1
0
P
pc = 0のとき、命令0を実行する。
pc = 1のとき、命令1を実行する
…
pc = nのとき命令nを実行する
S
値をロードする場合には、sp←sp+1としてS[sp]に置く。

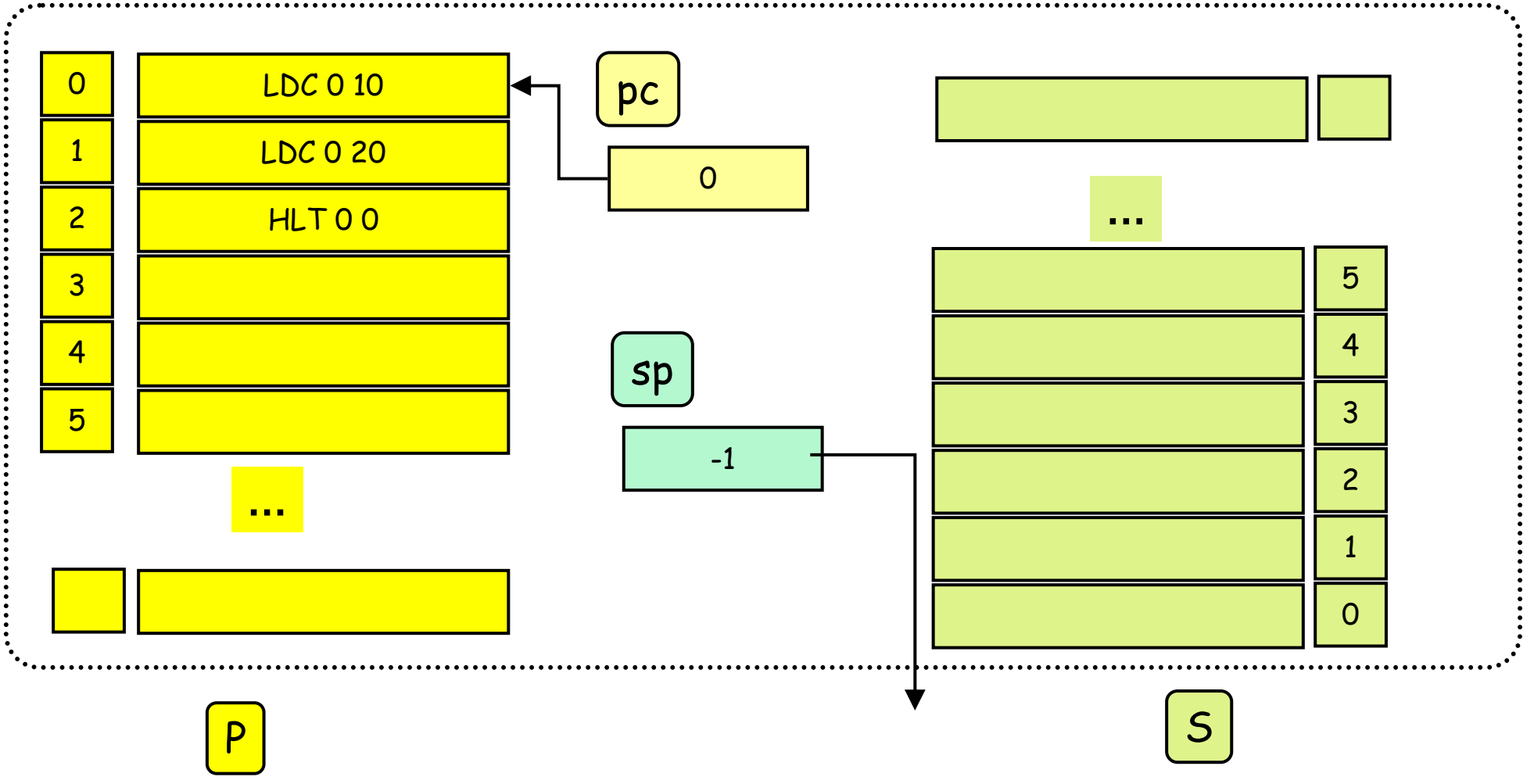
hsmの実行例

HLT 0 0 .. プログラムの終了
LDC 0 N … Nをスタックトップの上に積む。
sp++ ; S[sp] ← N ; pc++

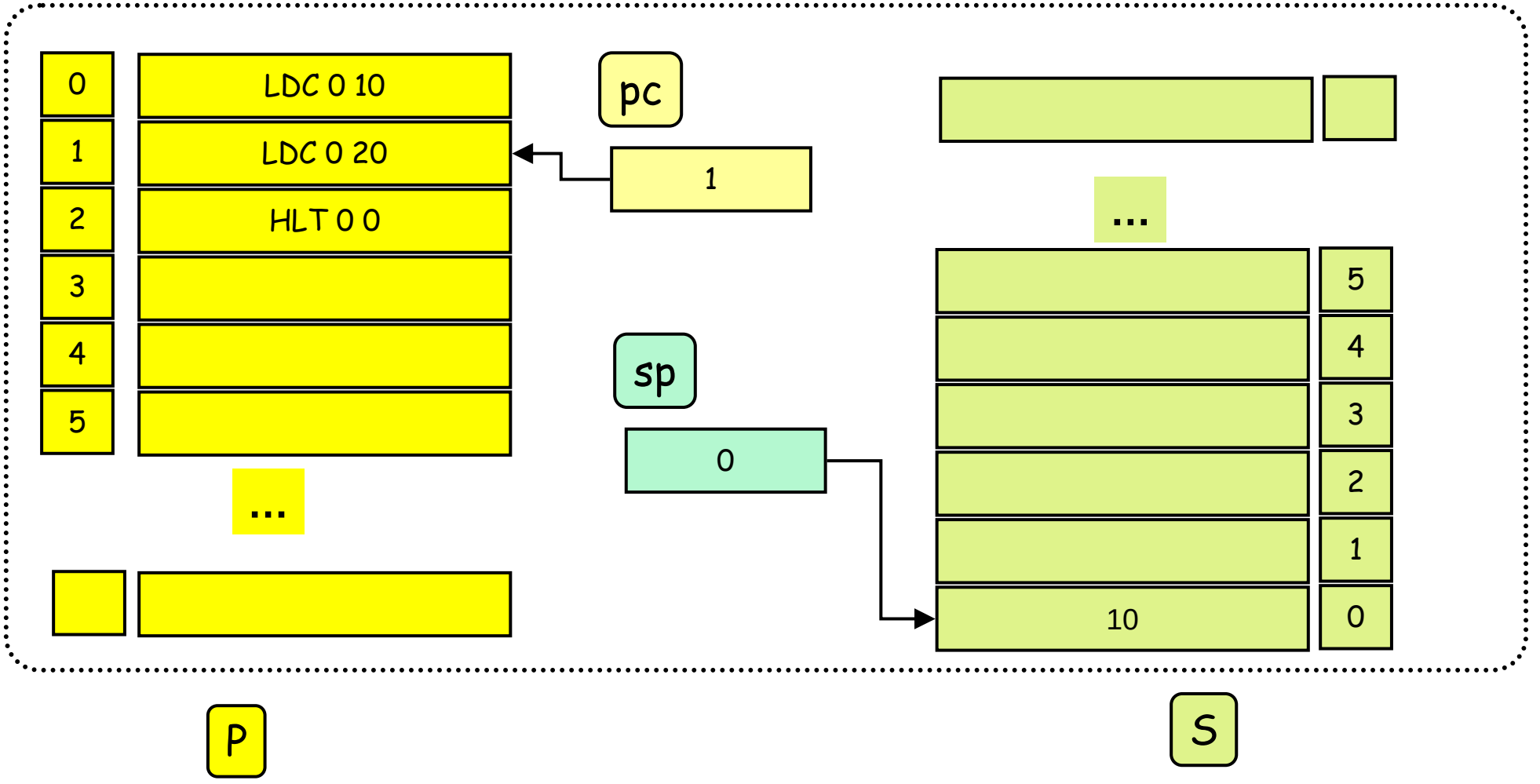
注:
spは、スタックトップ(一番最後に値をロードした場所)を指す解釈と、次に値をロードする場所(一番最後に値をロードした場所の一段上)を指す解釈が可能。

本講義、演習では前者の解釈を採用する。

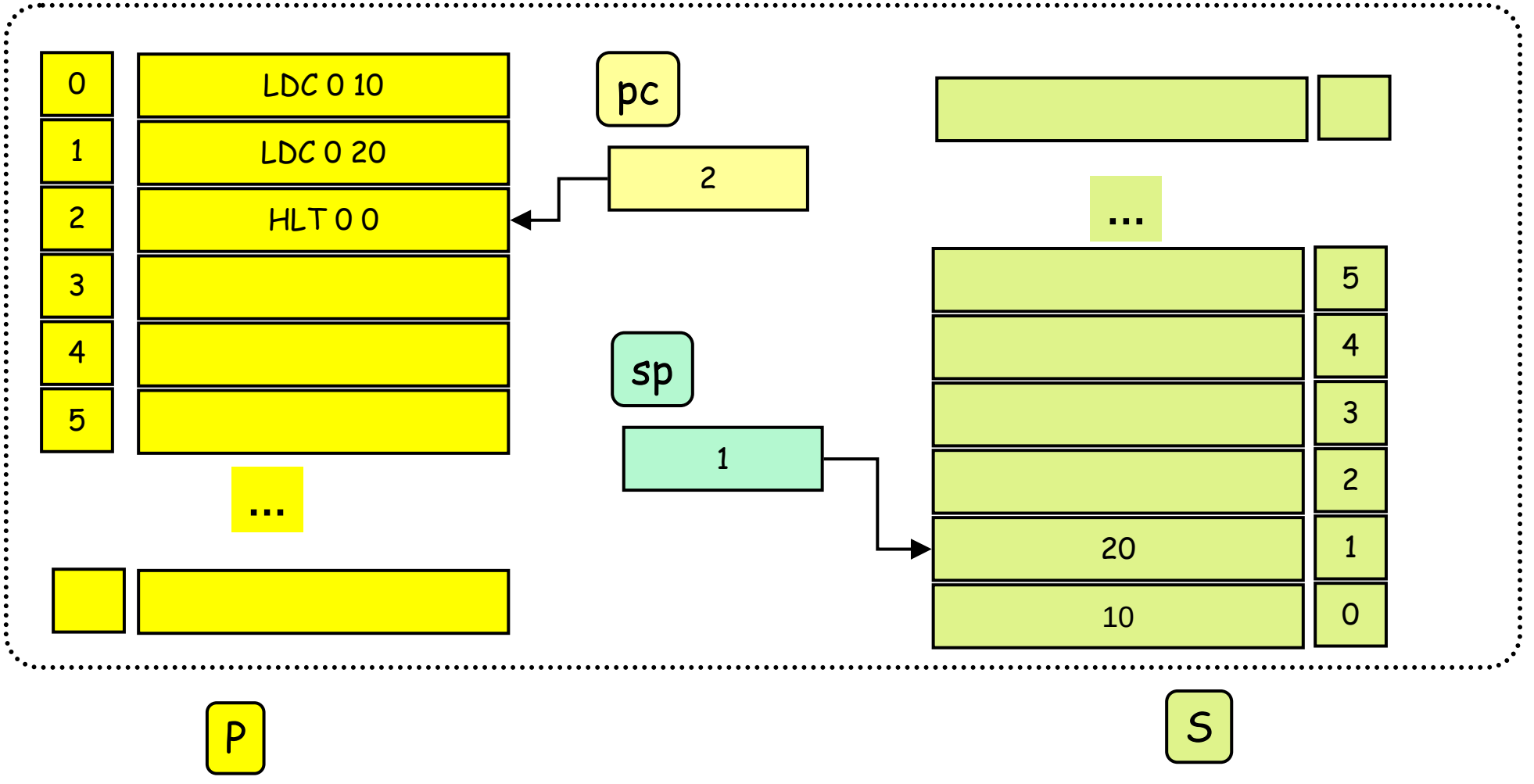
実行例(1)



実行例(2)



実行例(3)

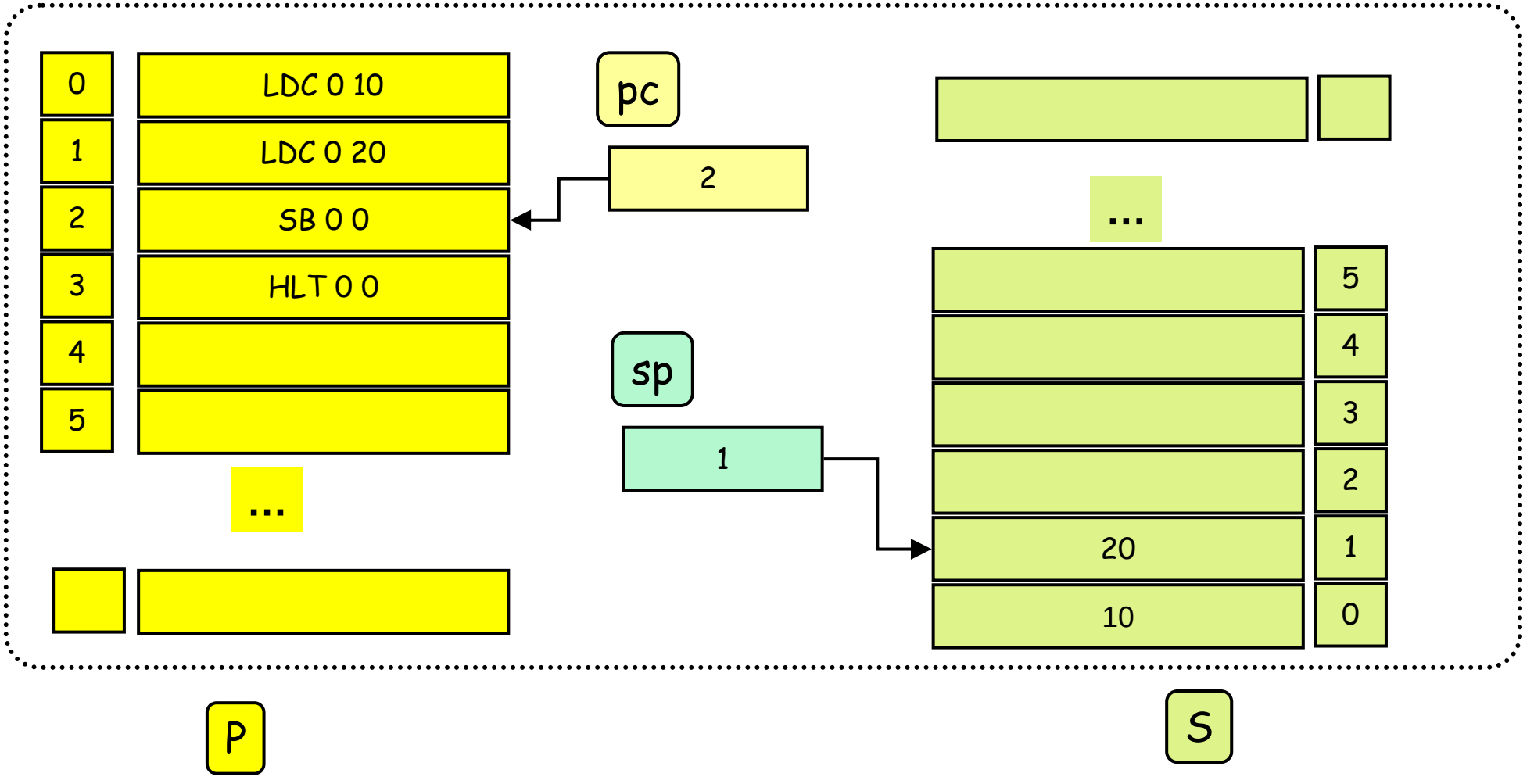


演算命令

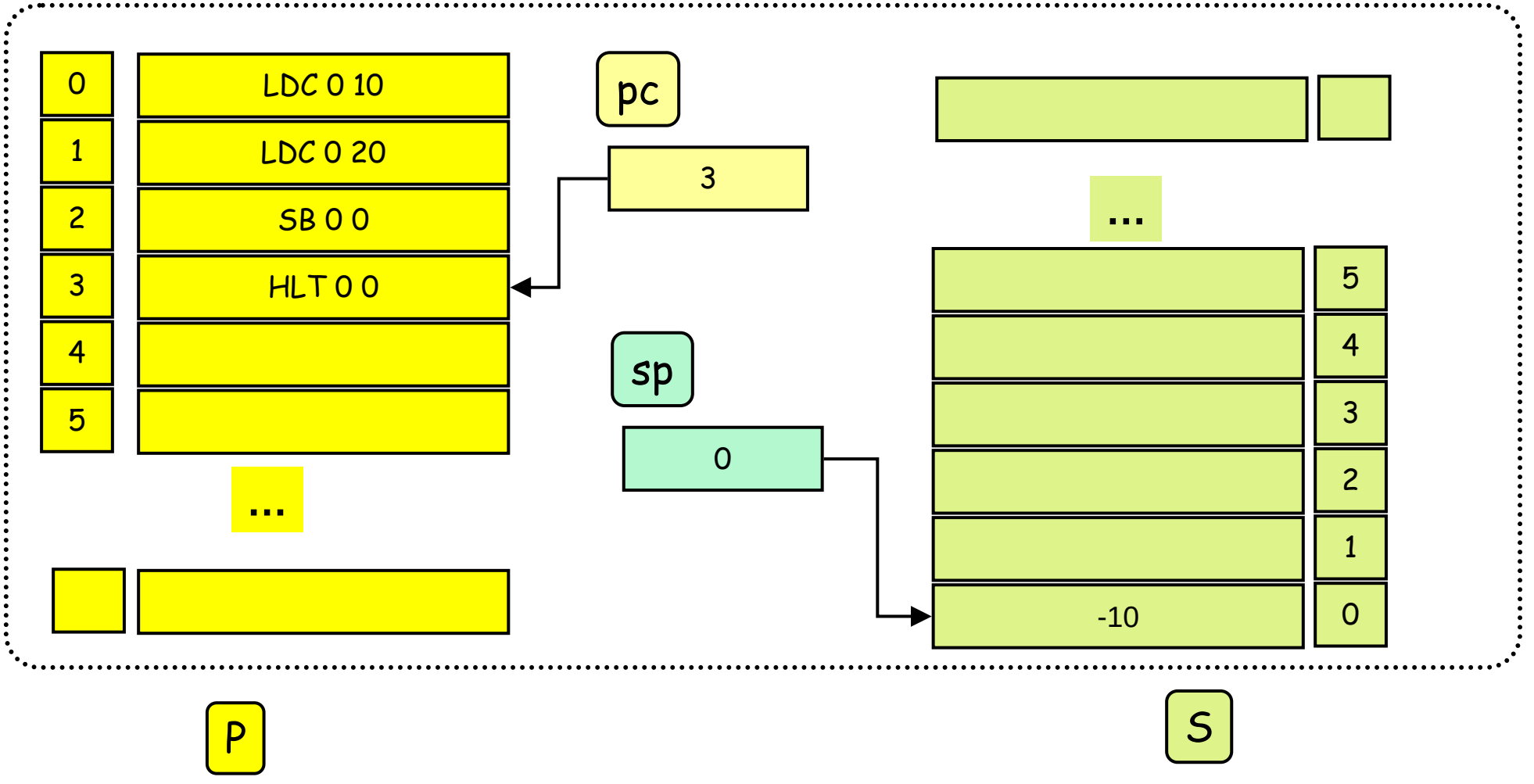
```
SB  0  0      引き算命令:
                  sp-- ; S[sp] ←S[sp]-S[sp+1]; pc++
NEG 0 0       符号反転命令:
                  S[sp] ← -S[sp]; pc++;
LE     0  0   関係演算命令<=
                  sp-- ; if (S[sp] <= S[sp+1) then S[sp]←1
                           else S[sp] ← 0; pc++
```

算術演算:
AD, SB,ML,DV, NEG … +, -, *, /, 反転
関係演算:
EQ, NEQ, LT,LE,GT,GE … ==, !=, <, <=, >, >=

実行例(4)



実行例(5)



出力命令

WRI　0　0　　整数出力命令:
S[sp]の値を出力; sp-- ; pc++

練習問題(1)

- 3+5*-1を計算するhsmマシン語のプログラムを書け。
- そのプログラムの動作をシミュレートせよ。

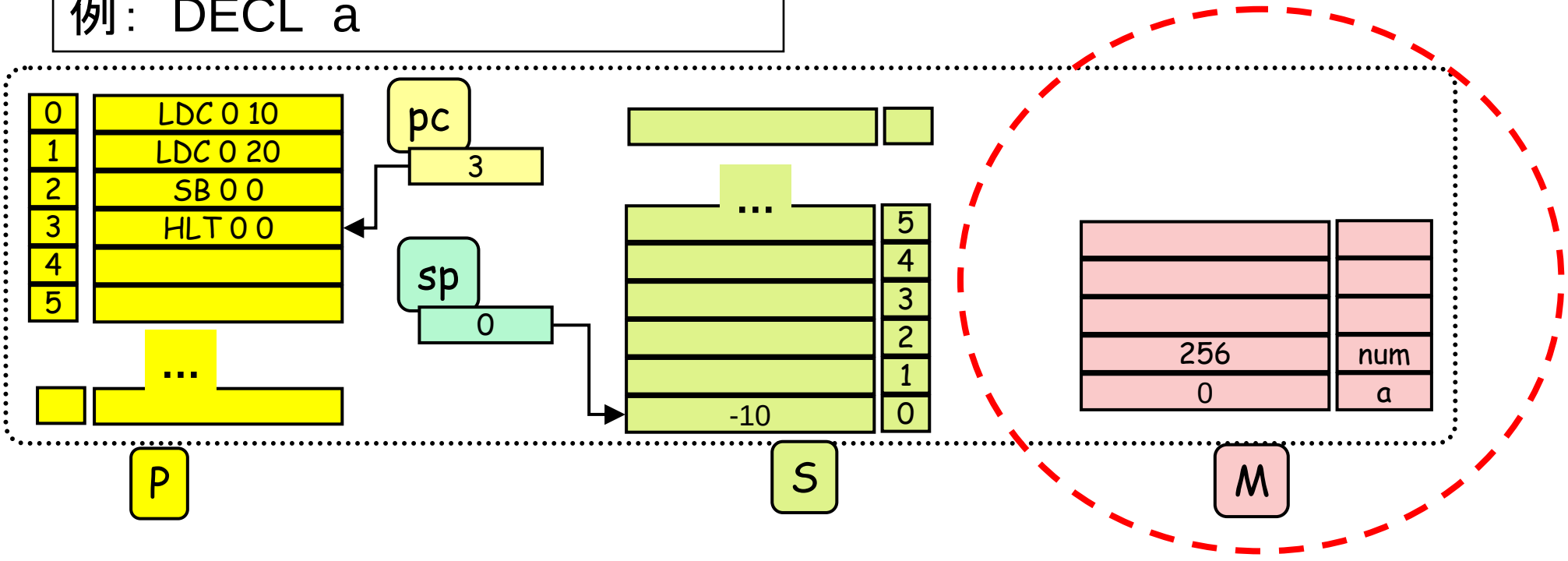
メモリの確保(疑似命令)

DECL *name*　メモリを確保する。
ただし、変数名を
*name*として、
M (*name*)←0; pc++

疑似命令→実際にはhsmにはない命令。

便宜的に、メモリ(M)を仮定する。
変数の機能であり、名前を使ってメモリの位置を指定し、格納されている値を読んだり(参照)、その位置に値を格納すること(代入)が可能。

例: DECL num
例: DECL a



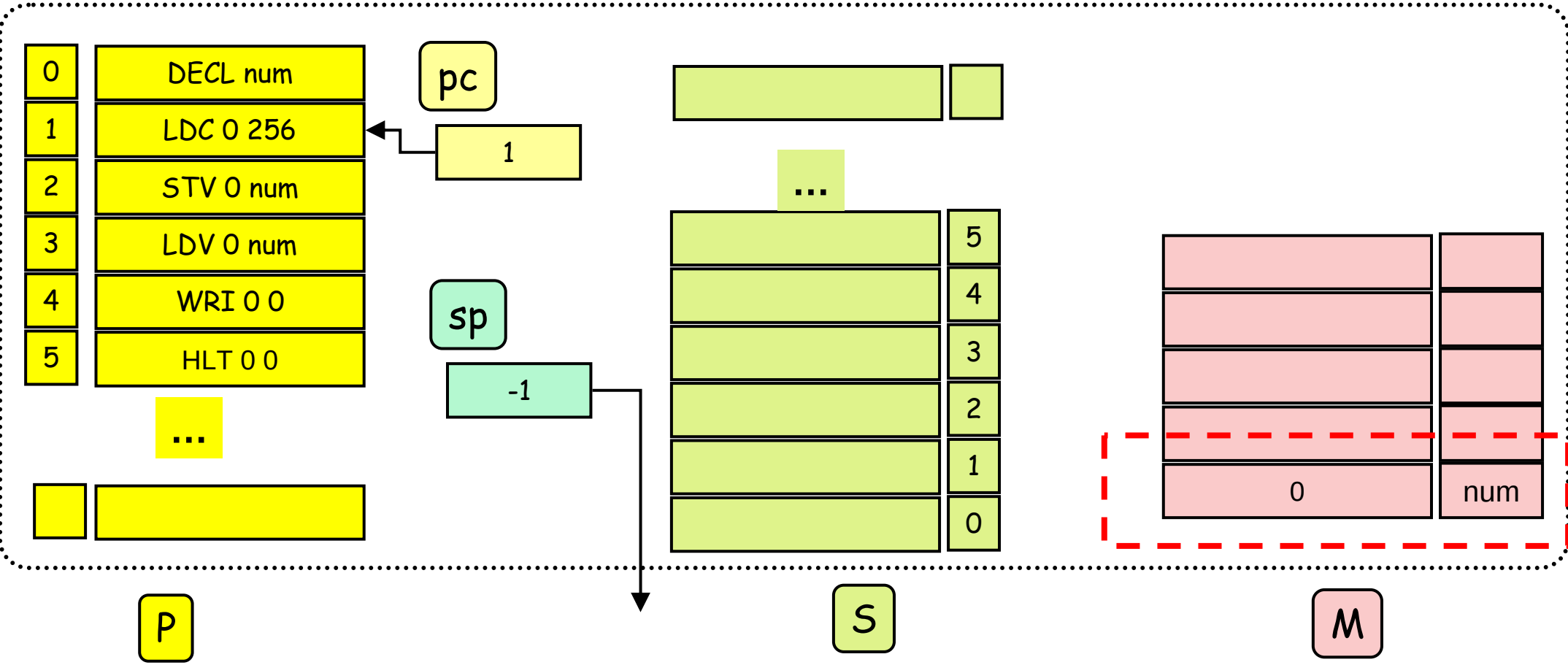
ロード命令、ストア命令(疑似命令）

STV 0 *name*	ストア命令（代入）
	M(name) ← S[sp]; sp-- ; pc++;
LDV 0 *name*	ロード命令（参照＝コピー）
	sp++; S[sp] ← M(name); pc++;

STV 0 str	(M(str)にスタックトップの値を代入)
LDV 0 str	(M(str)の内容をスタックに積む)

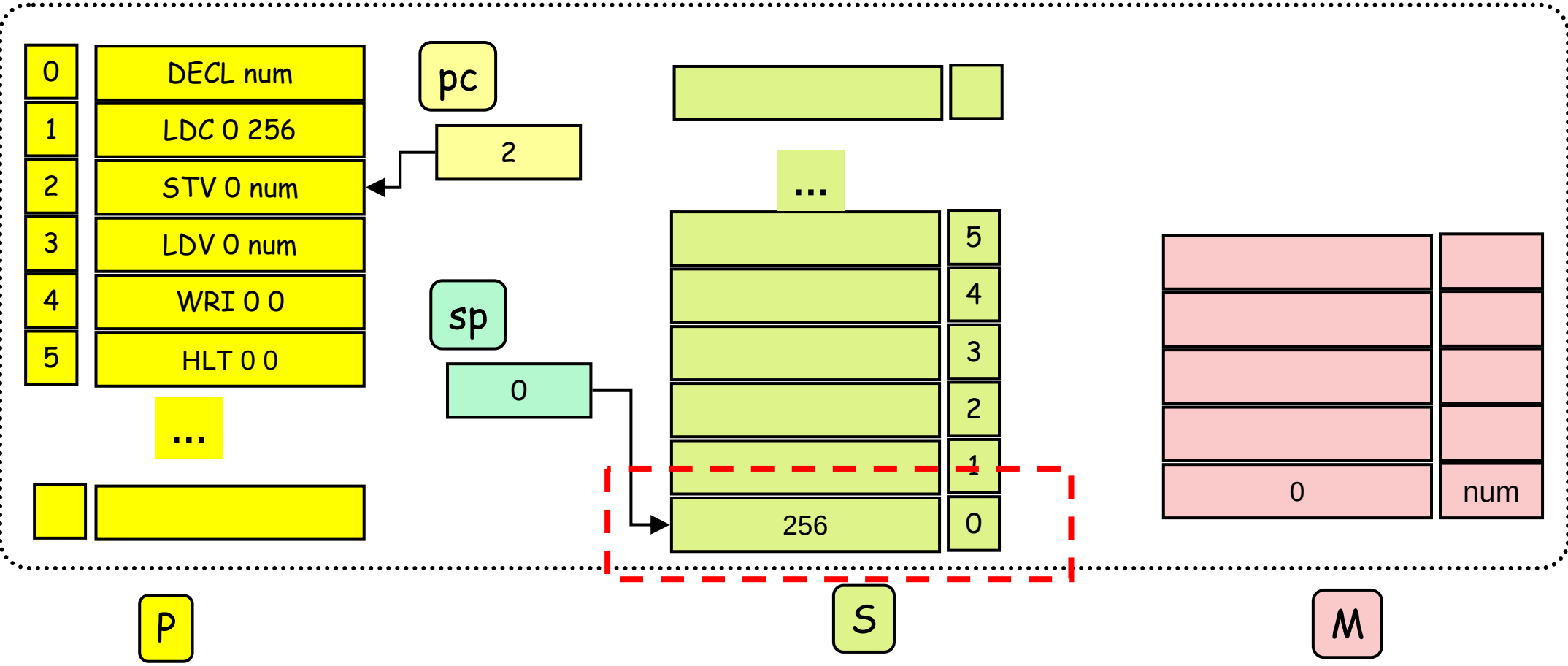
変数機能(疑似命令による)

```
int num;
num = 256;
putint(num);
```



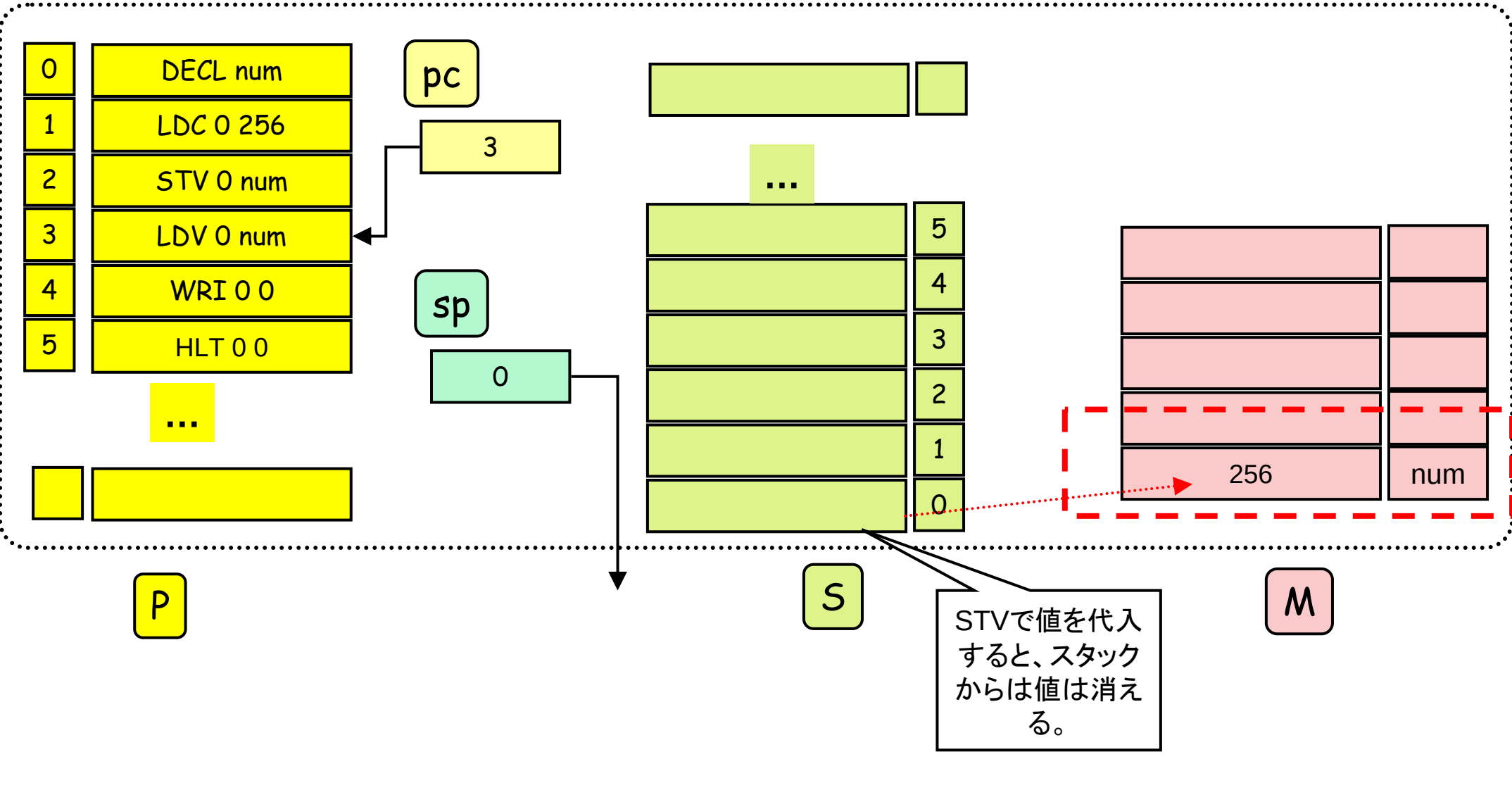
変数機能(疑似命令による)

```
int num;
num = 256;
putint(num);
```



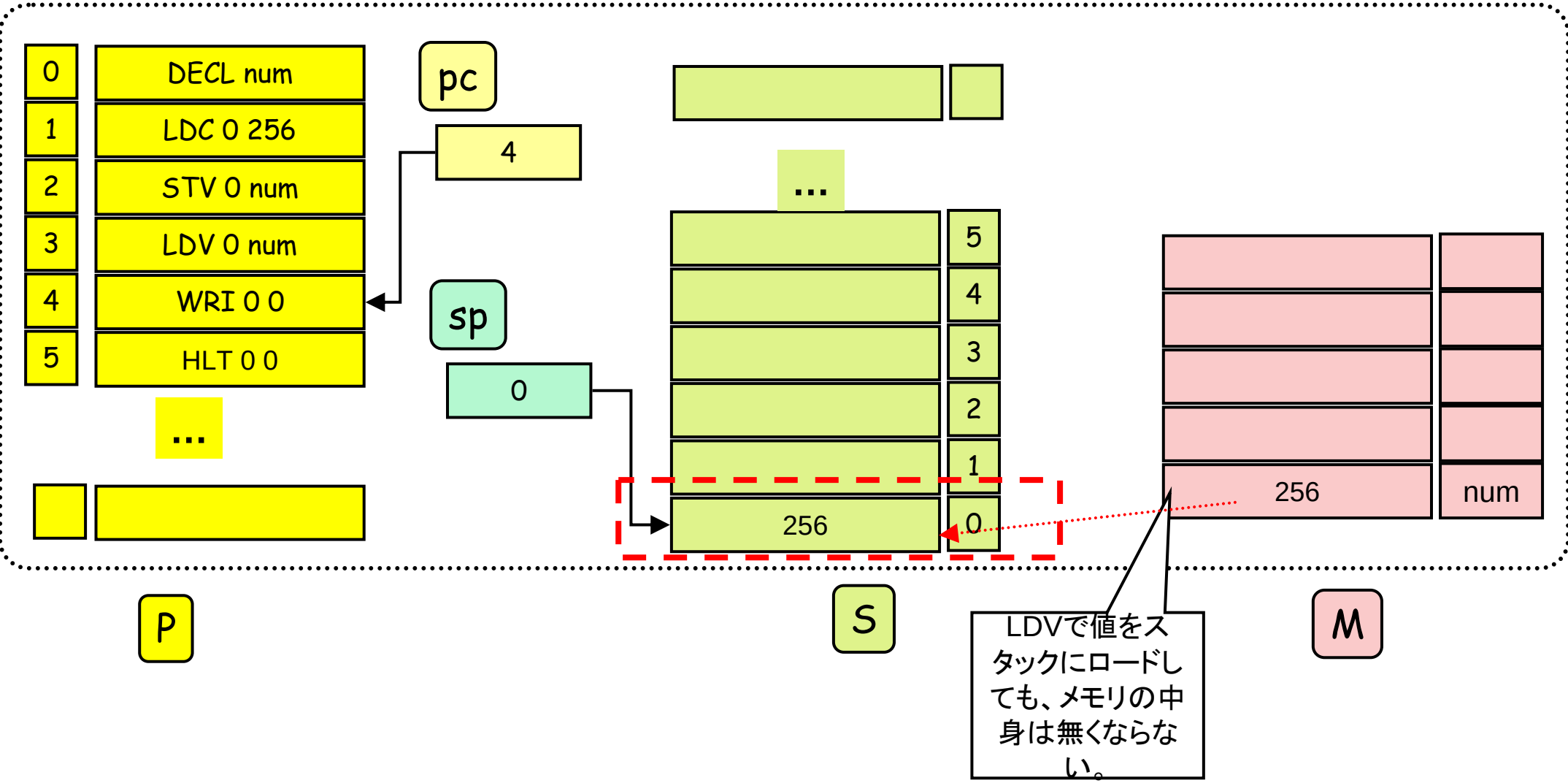
変数機能(疑似命令による)

```
int num;
num = 256;
putint(num);
```



変数機能(疑似命令による)

```
int num;
num = 256;
putint(num);
```



例（仮装機械の状態変化を各自でトレースしてみるとよい）

```
int main() {
  int num, result;
  num = 256;
  result = 2 * num;
  putint(result);
}
```

```
DECL num
DECL result
LDC 0 256
STV 0 num
LDC 0 2
LDV 0 num
ML   0 0
STV 0 result
LDV 0 result
WRI 0 0
HLT 0 0
```

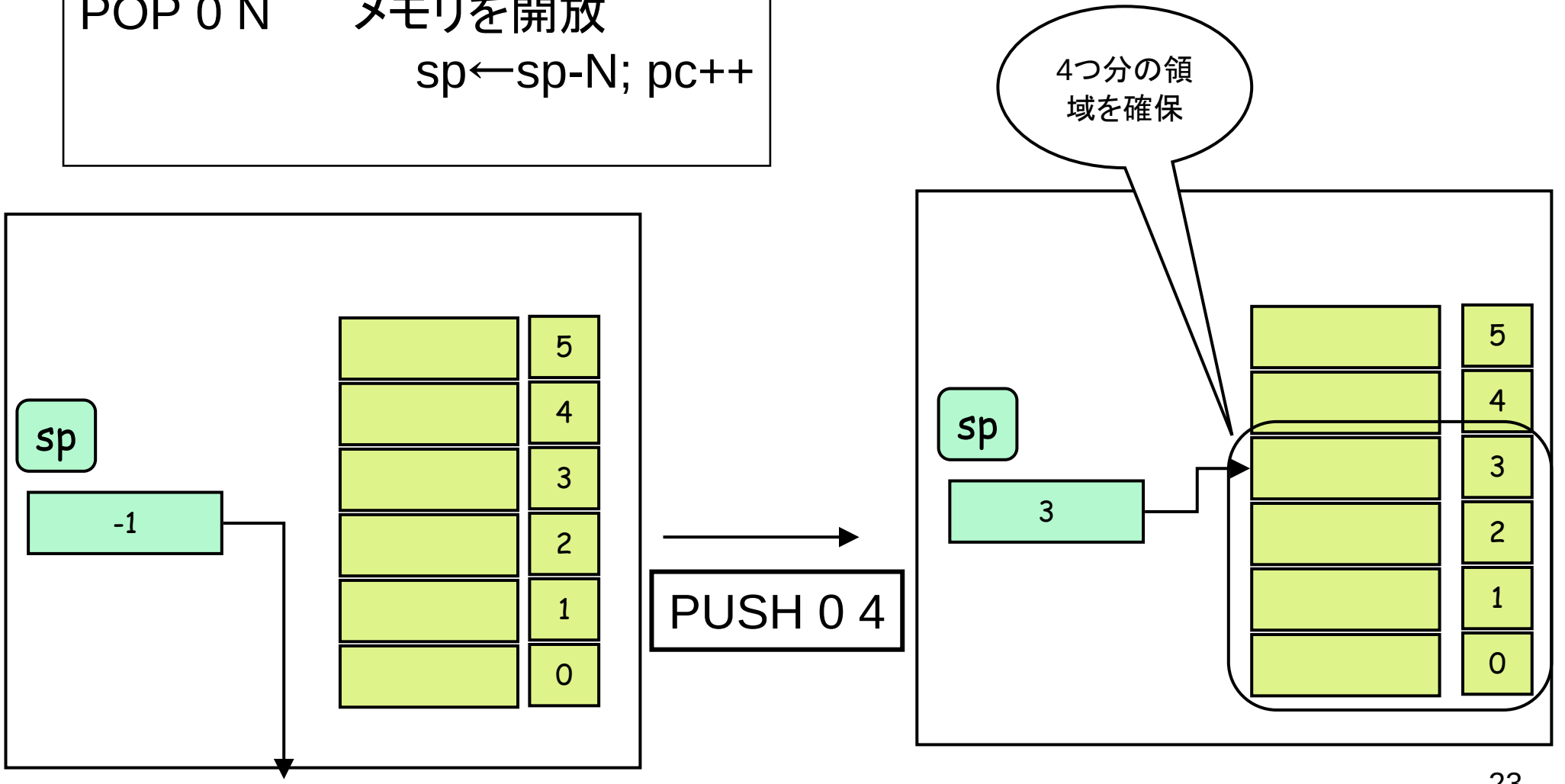
ロード命令、ストア命令(暫定版）

STV 0 N　ストア命令
　S[N] ←S[sp]; sp--; pc++
LDV 0 N　ロード命令
　sp++; S[sp]←S[N]; pc++
LDC 0 N　即値ロード命令
　sp++; S[sp]←N; pc++

Hsmのwebページの説明では、
STV p q
s[base(p)+q]=s[t];t=t-1;pc=pc+1;
となっている。p=0のときは、base(p)=0である。Pが0以外の場合については後の講義で説明する。

メモリの確保と開放

PUSH 0 N　　メモリを確保
　　sp←sp+N; pc++
POP 0 N　　メモリを開放
　　sp←sp-N; pc++



練習問題(2)

- 次のプログラムの動作を説明せよ。

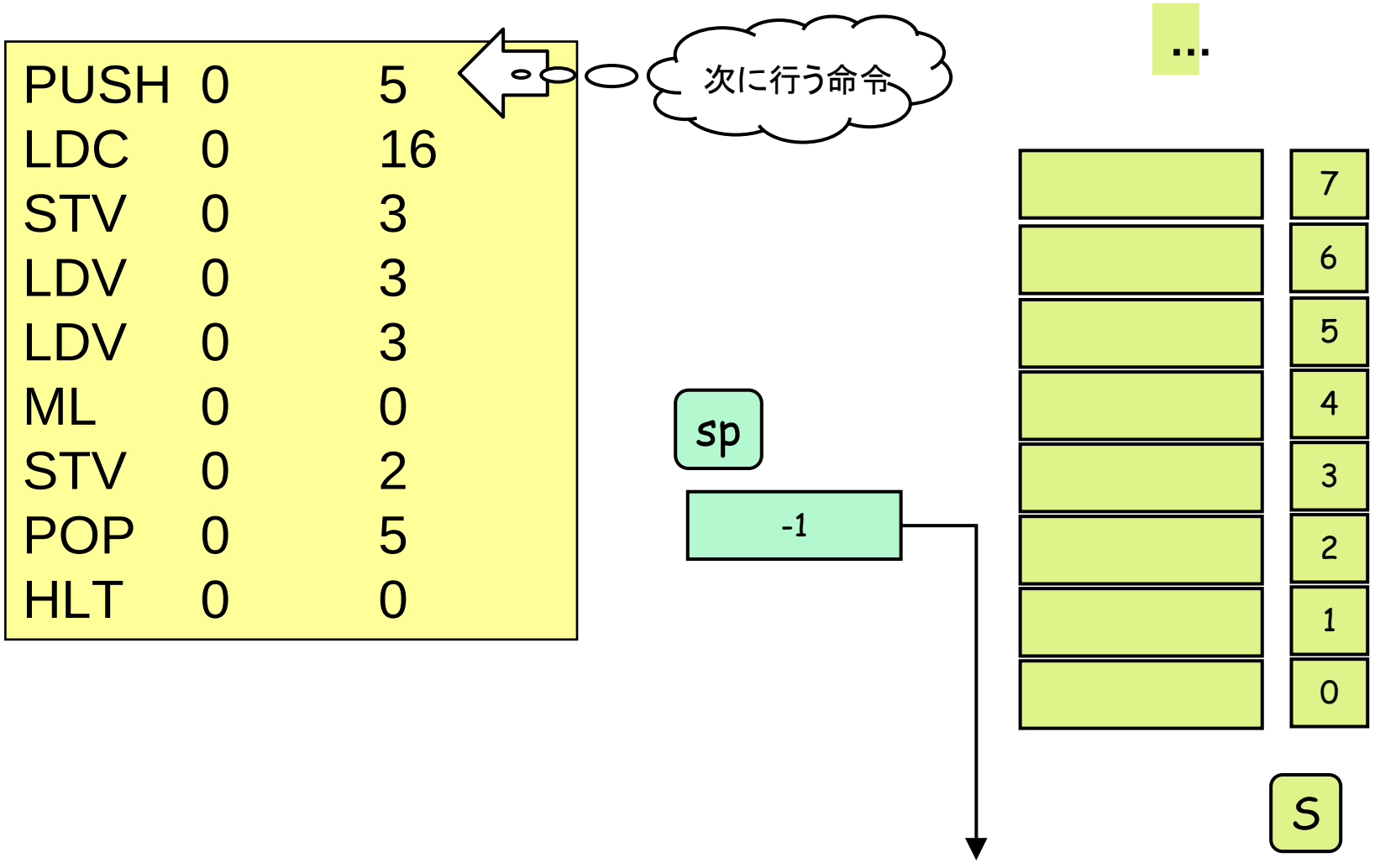
(1)

```
PUSH  0     5
LDC   0     16
STV   0     3
LDV   0     3
LDV   0     3
ML    0     0
STV   0     2
POP   0     5
HLT   0     0
```

(2)

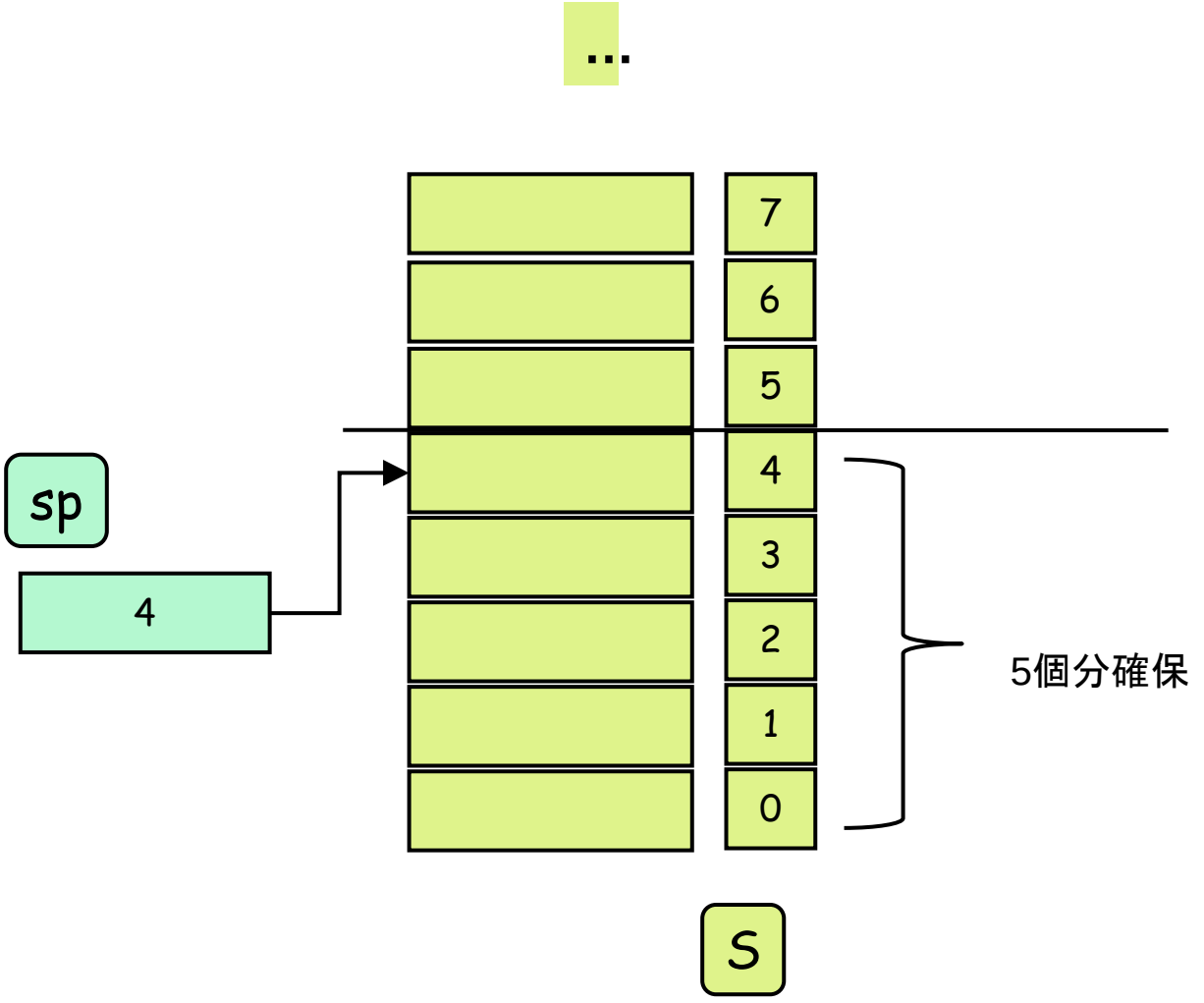
```
PUSH  0     3
LDC   0     10
STV   0     2
LDC   0     5
STV   0     1
LDV   0     2
LDV   0     2
ML    0     0
LDV   0     1
LDV   0     2
ML    0     0
AD    0     0
STV   0     0
POP   0     3
HLT   0     0
```

実行例(ロードとストア)


PUSH 0 5
LDC 0 16
STV 0 3
LDV 0 3
LDV 0 3
ML 0 0
STV 0 2
POP 0 5
HLT 0 0
次に行う命令
...
sp
-1
7
6
5
4
3
2
1
0
S

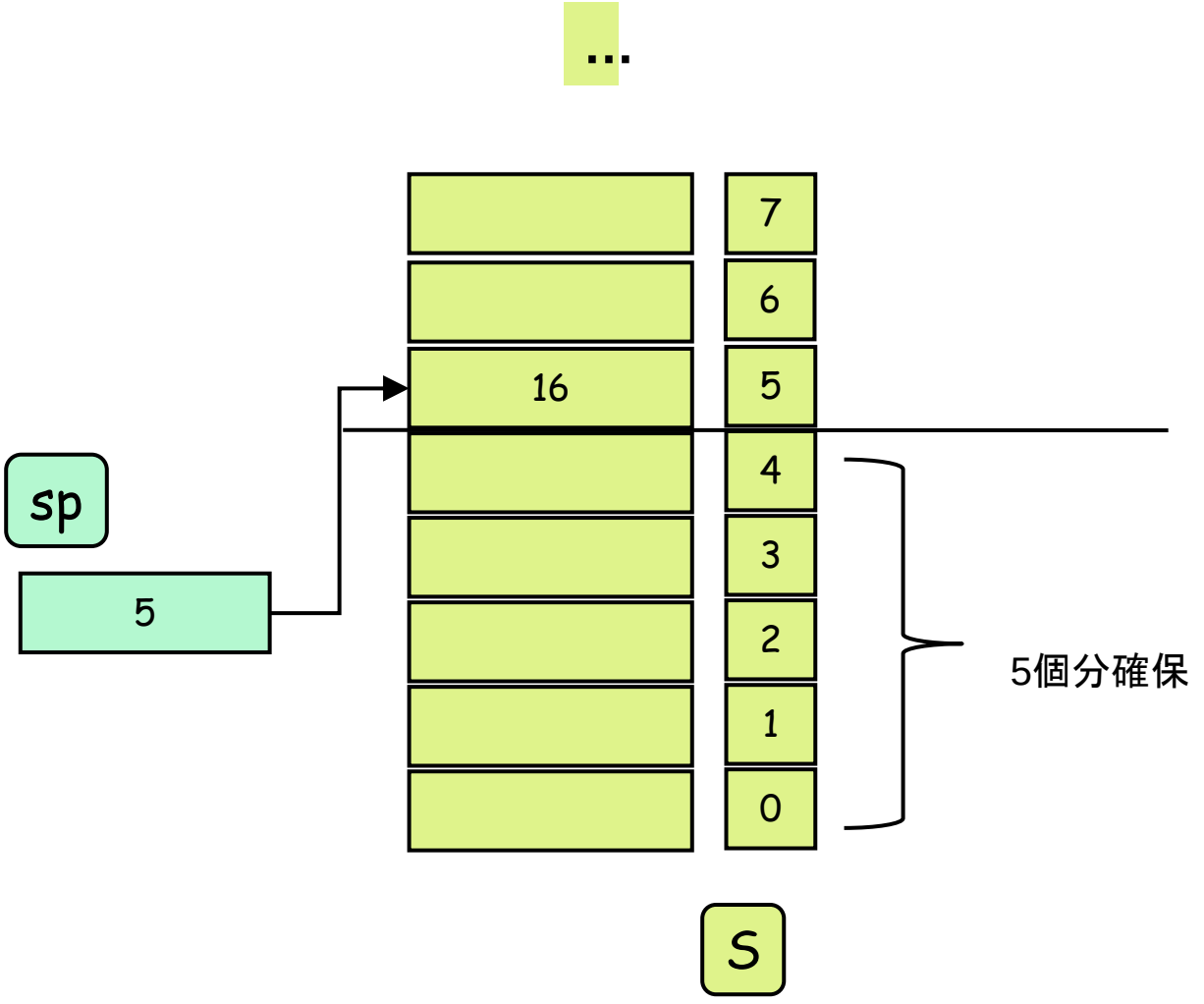
実行例(ロードとストア)

```
PUSH  0    5
LDC   0    16
STV   0    3
LDV   0    3
LDV   0    3
ML    0    0
STV   0    2
POP   0    5
HLT   0    0
```



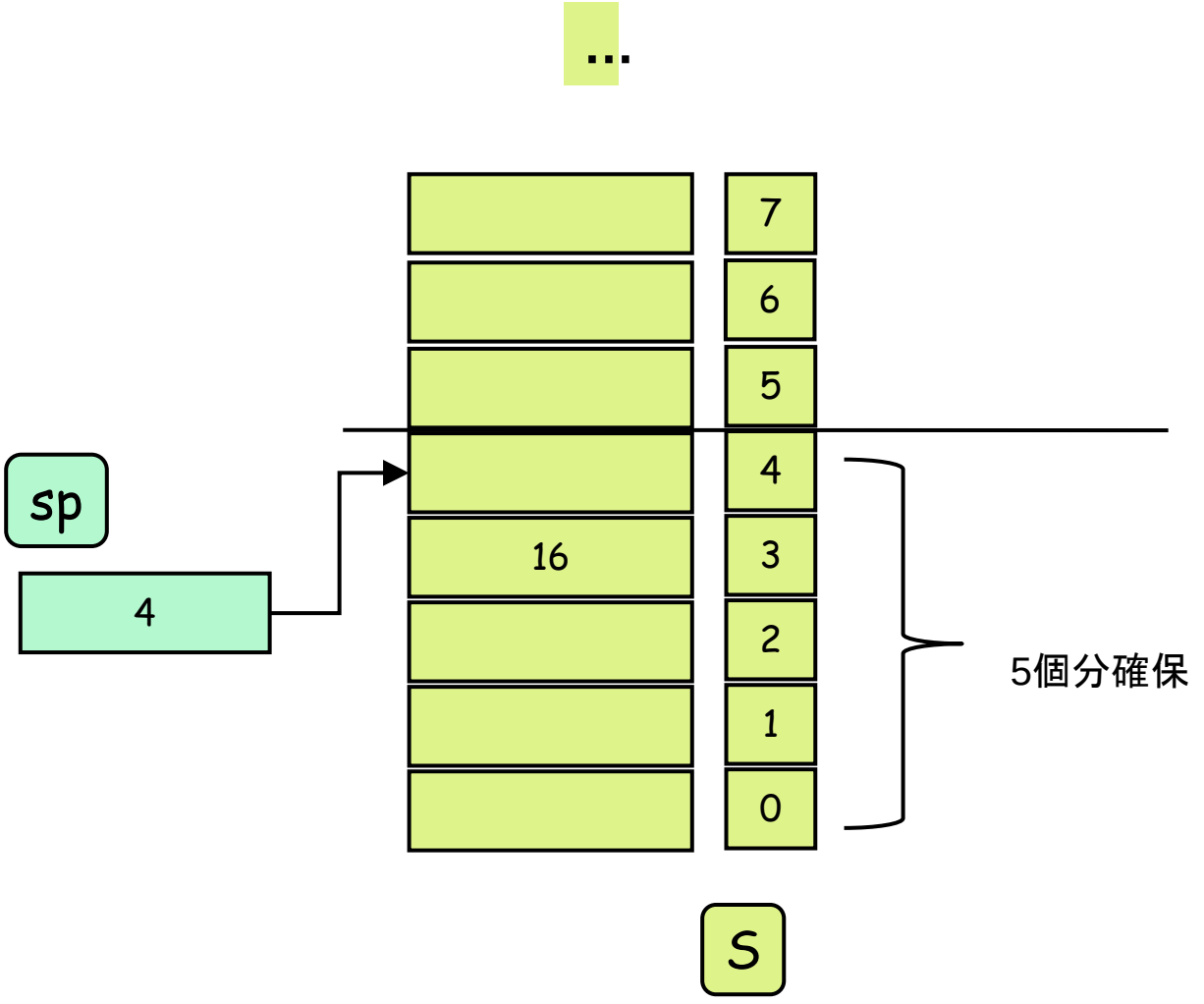
実行例(ロードとストア)

```
PUSH  0    5
LDC   0    16
STV   0    3
LDV   0    3
LDV   0    3
ML    0    0
STV   0    2
POP   0    5
HLT   0    0
```



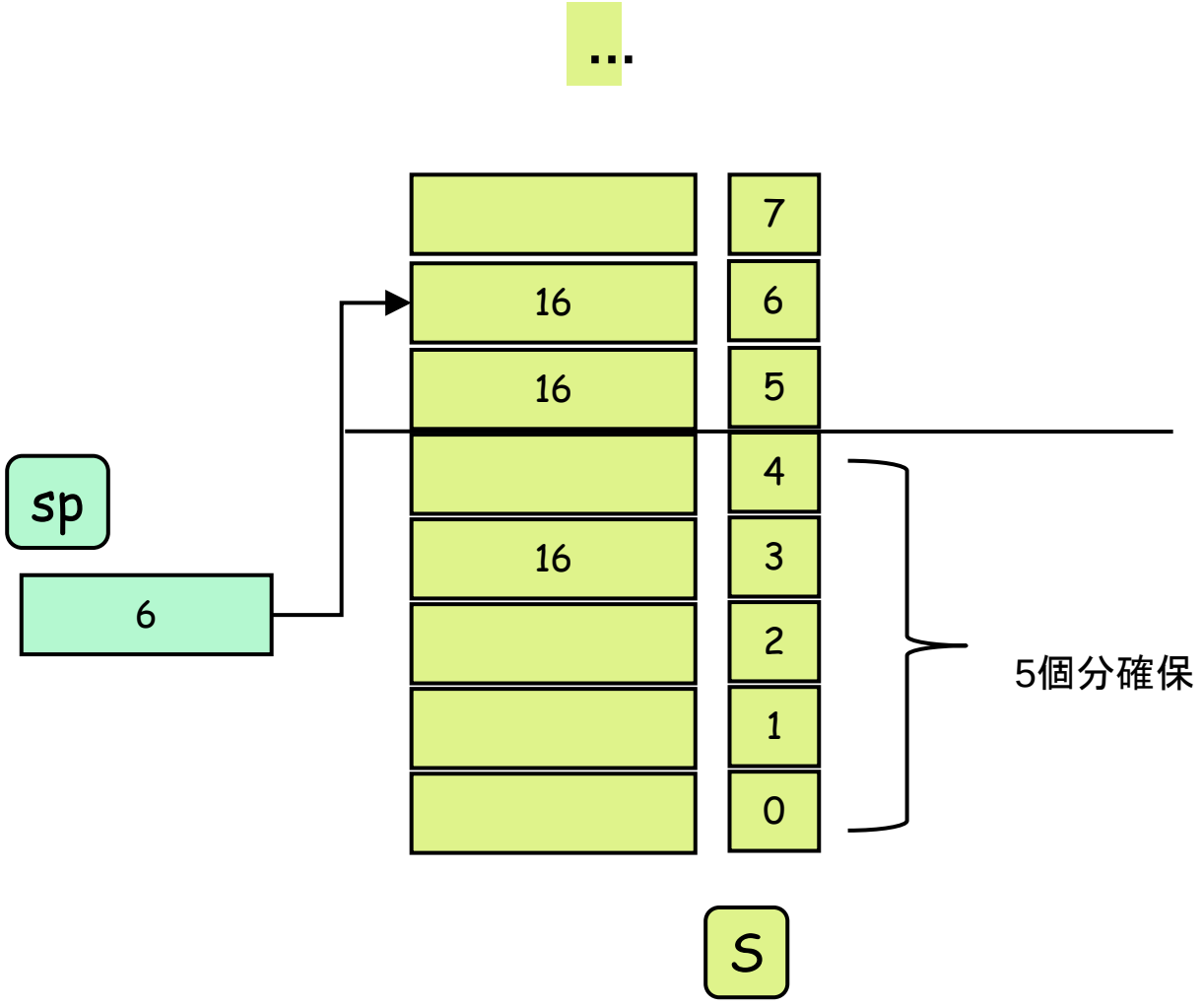
実行例(ロードとストア)

```
PUSH  0     5
LDC   0     16
STV   0     3
LDV   0     3
LDV   0     3
ML    0     0
STV   0     2
POP   0     5
HLT   0     0
```



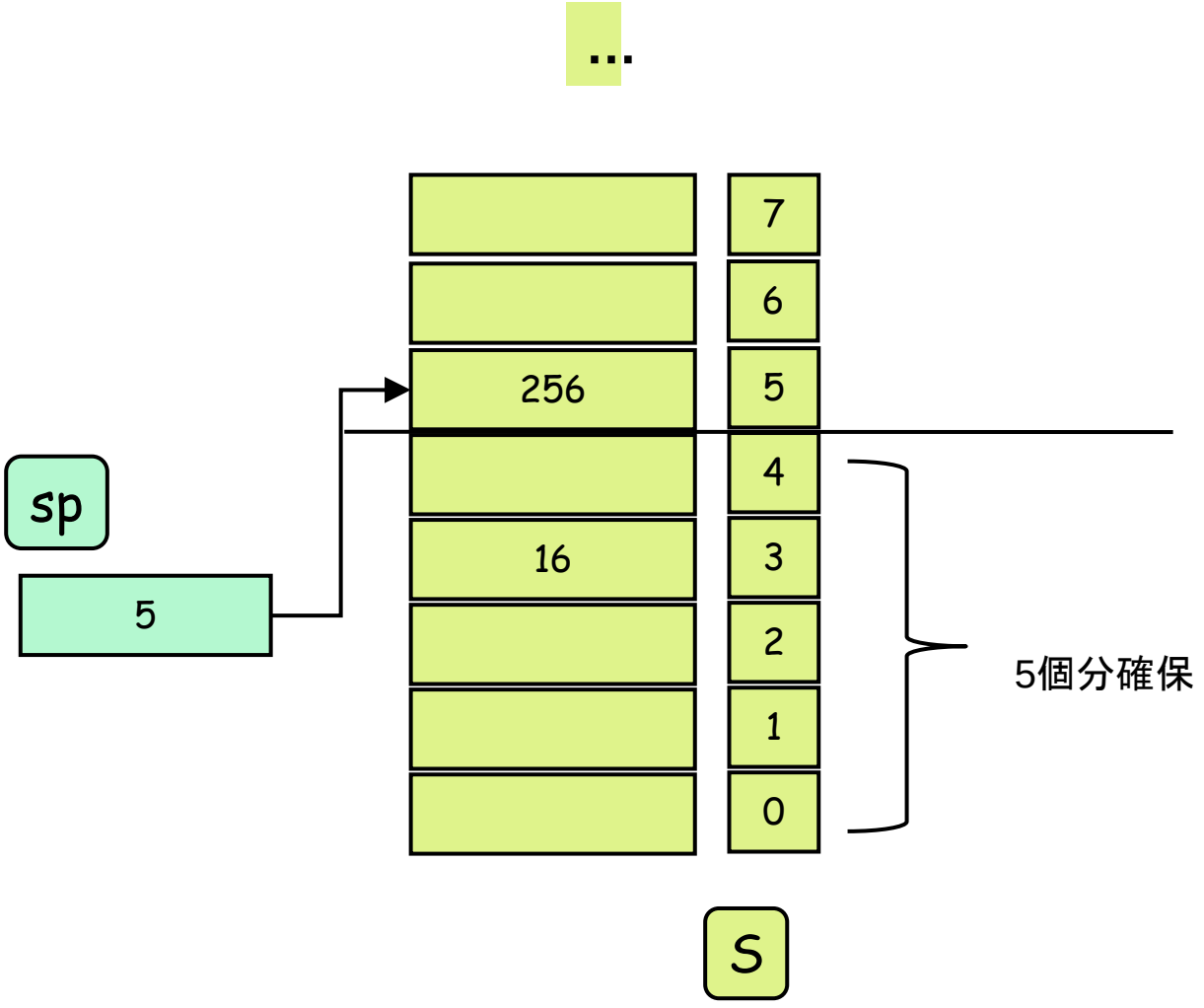
実行例(ロードとストア)

```
PUSH  0    5
LDC   0    16
STV   0    3
LDV   0    3
LDV   0    3
ML    0    0
STV   0    2
POP   0    5
HLT   0    0
```



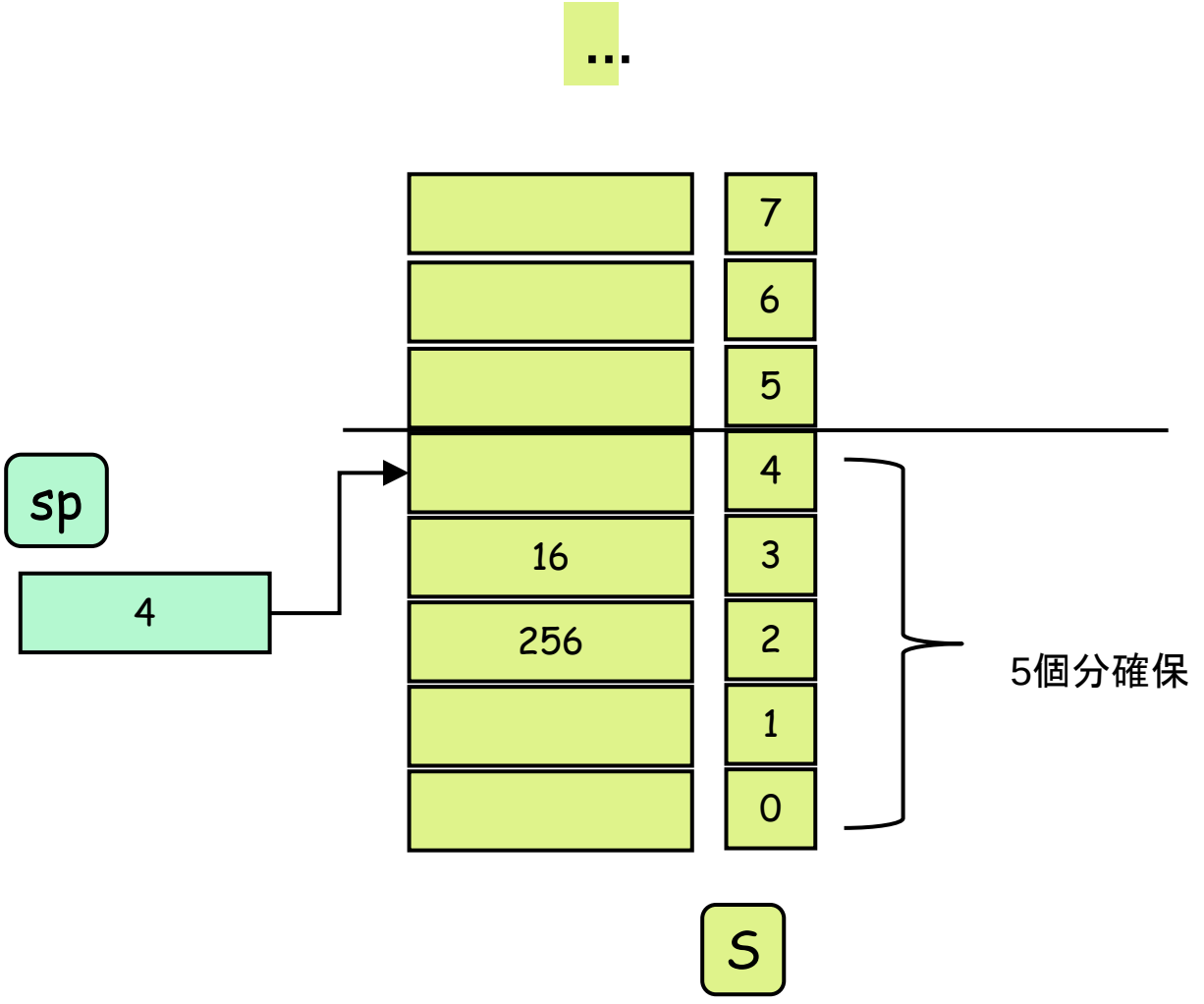
実行例(ロードとストア)

```
PUSH  0   5
LDC   0   16
STV   0   3
LDV   0   3
LDV   0   3
ML    0   0
STV   0   2
POP   0   5
HLT   0   0
```



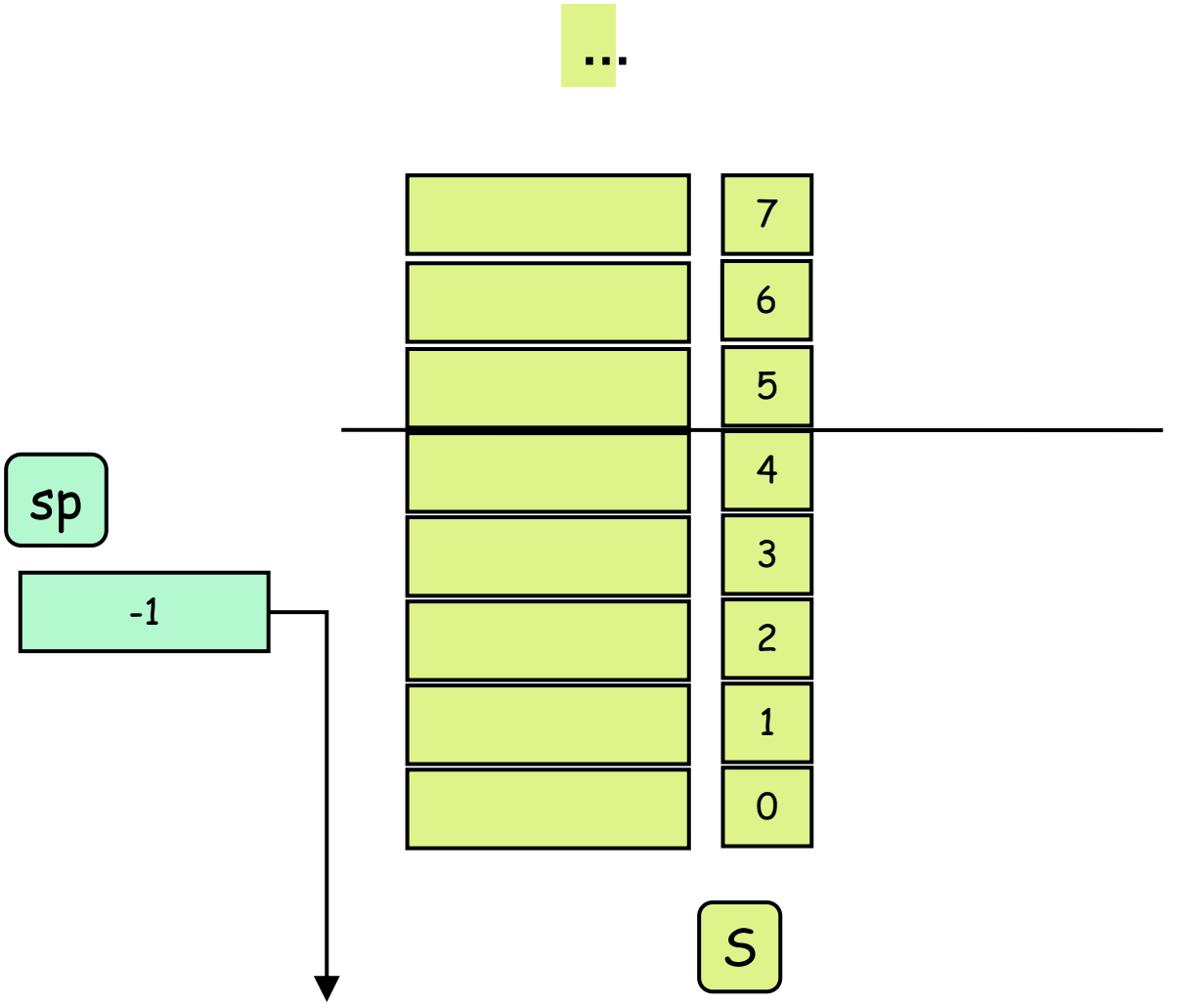
実行例(ロードとストア)

```
PUSH  0   5
LDC   0   16
STV   0   3
LDV   0   3
LDV   0   3
ML    0   0
STV   0   2
POP   0   5
HLT   0   0
```



実行例(ロードとストア)

```
PUSH  0   5
LDC   0   16
STV   0   3
LDV   0   3
LDV   0   3
ML    0   0
STV   0   2
POP   0   5
HLT   0   0
```



制御命令

```
J  0  N     ジャンプ命令:
                pc ←N;
FJ  0 N     条件ジャンプ命令:
                if (S[sp] == 0)
                  pc ← N;
                else
                   pc++;
                sp--;
```

```
J 0 20
```

20番地に飛ぶ。
(次の命令は
P[20])

```
LDC 0 1
LDC 0 2
EQ 0 0
FJ 0 20
```

1==2が偽(0)であれば20番地に
飛ぶ。そうでなければ次の命令へ。
（Fall through
(この場合1==2はa偽であるから、
必ず20番地に飛ぶ)

出力命令

```
WRI  0  0     整数表示:
                  S[sp]を表示; sp--;  pc++;
WRC 0 0   文字表示:
                  文字コードS[sp]に対する文字の表示;
                  sp--; pc++;
WNL 0 0   改行表示:
                  改行; pc++;
```

```
LDC 0 1
WRI 0 0
```

整数1を表示

```
LDC 0 70
WRC 0 0
```

文字コード70番の文字(F)を表示

練習問題

- FJ命令の動きに注意して、VMの動作をシミュレートしてみよ。（紙と鉛筆を使って手で確認せよ。その後hsmで確かめると良い。）
- 最初の命令が0: LDC 0 2の場合どうなるかも考えよ。

```
0: LDC 0 1
1: LDC 0 2
2: EQ 0 0
3: FJ 0 6
4: LDC 0 84
5: WRC 0 0
6: HLT 0 0
```

```
0: LDC 0 1
1: LDC 0 2
2: EQ 0 0
3: FJ 0 7
4: LDC 0 84
5: WRC 0 0
6: J 0 9
7: LDC 0 70
8: WRC 0 0
9: HLT 0 0
```