INF600C — Sécurité des logiciels et exploitation de vulnérabilités
Jean Privat – Université du Québec à Montréal
Mercredi 24 avril 2019 — 3 heures

Aucun document n'est autorisé. L'usage de la calculatrice ou tout autre appareil électronique est interdit. L'annexe Pep/8 est détachable.

La lisibilité et la clarté des réponses et des payloads sont inclues dans la notation.

Attention ⚠ : contrairement à un lab ou à un CTF, la méthode essai-erreur ne fonctionne pas en examen.

— Portez attention aux décalages, aux adresses, aux tailles et au boutisme.
— Cherchez la simplicité pour minimiser le risque d'erreur.
— Ne passez pas trop de temps sur une question, quitte à revenir plus tard.

0x10 Question générale

Question 1 (5 points) : Que signifie le « E » du mécanisme de sécurité « PIE » ?

☐ Executable ☐ Extended ☐ Exception ☐ Emplacement ☐ Exploit

0x20 Rev Pep/8

Soit le code machine Pep/8 suivant (consultez l'annexe Pep/8 à la fin) :

```
41 00 13 31 00 2E 0A 00 12 C9 00 2E 55 00 13 04
00 03 00 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D
4E 4F 50 51 52 53 54 55 56 57 58 59 5A 0A 00 00
zz
```

Question 2 (5 points) : Quel sont les 4 premiers caractères affichés quand l'entrée est « 2 1 0 » ?

. . . . . .

Question 3 (5 points) : Quels sont les 4 derniers caractères affichés quand l'entrée est « 20 13 26 19 4 18 19 0 12 4 13 19 0 » ?

. . . . . .

Question 4 (5 points) : Proposez une entrée (*payload*) à utiliser pour que le programme affiche « FLAG ». Décrivez l'attaque utilisée.

. . . . . .

. . . . . .

. . . . . .

0x30 Pwn Pep/8

Voici le listing d'un programme Pep/8.

```
      Object
Addr  code   Symbol   Mnemon  Operand     Comment
------------------------------------------------------------------------------
0000  C00000 main:    LDA     0,i
0003  16000C          CALL    lire
              ;
              ; chaine a inverser
0006  000000 tab:     .BLOCK  5           ; #1c5a
      0000
              ;
              ; dernier caractere lu
000B  00     char:    .BLOCK  1           ; #1c
              ;
              ; lit la chaine dans tab
000C  C80000 lire:    LDX     0,i
000F  49000B lire_b:  CHARI   char,d
0012  D1000B          LDBYTEA char,d
0015  B00047          CPA     'G',i
0018  0A0030          BREQ    lire_f
001B  B0002E          CPA     '.',i
001E  08002A          BRLT    lire_n
0021  B0007A          CPA     'z',i
0024  10002A          BRGT    lire_n
0027  F50006          STBYTEA tab,x
002A  780001 lire_n:  ADDX    1,i
002D  04000F          BR      lire_b
0030  160033 lire_f:  CALL    ecrire
              ;
              ; ecrit la chaine a l'envers
0033  880001 ecrire:  SUBX    1,i
0036  08003F          BRLT    ecrire_f
              ;LDBYTEA tab,x
              ;CPA     ' ',i
              ;BRLT    ecrire
0039  550006          CHARO   tab,x
003C  040033          BR      ecrire
003F  00     ecrire_f:STOP
              ;
              ; hummm...
0040  464C41 flag:    .ASCII  "FLAG{INF600C_pour_la_vie_<3}\x00"
      477B49
      4E4636
      303043
      5F706F
      75725F
      6C615F
      766965
      5F3C33
      7D00
005D                  .END
```

Ce programme affiche les mots saisis à l'envers. La lettre G (comme « go ») est utilisée pour marquer la fin du mot. Par exemple « FLAG » devient « ALF » l'extraterrestre bien connu.

Question 5 (5 points) : Parmi les vulnérabilités suivantes, laquelle existe dans ce programme ?

☐ CWE-121 Débordement de tampon dans la pile (*Stack-based Buffer Overflow*)
☐ CWE-122 Débordement de tampon dans le tas (*Heap-based Buffer Overflow*)
☐ CWE-125 Lecture en dehors des limites d'un tampon (*Out-of-bounds Read*)
☐ CWE-193 Erreur d'une unité (*Off-by-one Error*)
☐ CWE-369 Division par zéro
☐ CWE-476 Déréférencement de NULL (*NULL Pointer Dereference*)
☐ CWE-416 Utilisation après libération (*Use After Free*)
☐ CWE-787 Écriture en dehors des limites d'un tampon (*Out-of-bounds Write*)

Question 6 (5 points) : Qu'affiche le programme pour la chaine « ESSORAGE » ?

.....

.....

Question 7 (5 points) : Quelle entrée permet d'afficher le mot « FLAGRANTE » ?

.....

.....

Question 8 (5 points) : Proposez une entrée (*payload*) à utiliser pour que le programme affiche le flag complet (identifiée par le symbole `flag`). Décrivez l'attaque utilisée et expliquez l'effet de l'entrée sur l'exécution du programme. Note : mon entrée fait 11 caractères.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

0x40 Corruption mémoire en C

Le programme `getflag` affiche les fichiers du répertoire `/quetes/final`, sauf ceux dont le nom commence par le préfixe « flag ».

```
$ ls -la /quetes/final
drwxr-xr-x 3 root root     4096 avr 15 09:02 .
drwxr-xr-x 9 root root     4096 jan  8 19:46 ..
-rw-r----- 1 root getflag    31 avr 24 14:00 abcd
-rwxr-sr-x 1 root getflag 12336 avr 24 14:00 getflag
-rw-r----- 1 root getflag    39 avr 24 14:00 flag
-rw-r----- 1 root getflag    26 avr 24 14:00 toto
```

Le code source du programme `getflag` est le suivant :

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <string.h>
5  void cleanup(char *str) { // Remplace tous les '.' par des octets nuls.
6          size_t i, len = strlen(str);
7          for(i=0; i<len; i++) if (str[i] == '.') str[i] = '\0';
8  }
9  int show(char *file) { // Affiche le fichier /quetes/final/<file>.
10         char chemin[32];
11         strcpy(chemin, "/quetes/final/");
12         strcat(chemin, file);
13         cleanup(chemin);
14         if (access(chemin, F_OK) == 0) {
15                 execlp("/bin/cat", "/bin/cat", chemin, NULL);
16                 perror("crash");
17         } else perror(chemin);
18         return 28717; // Hmmm...
19 }
20 int main(int argc, char **argv) {
21         if (argc != 2) {
22                 printf("usage: getflag fichier\n");
23                 return 1;
24         }
25         char *file = argv[1];
26         if (strncmp(file, "flag", 4) == 0) {
27                 fprintf(stderr, "%s: Fichier interdit.\n", file);
28                 return 1;
29         }
30         show(file);
31         return 1;
32 }
```

Consultez l'annexe à la fin pour la documentation des fonctions utilisées.

ASLR est activé sur le système mais le programme est compilé avec un seul mécanisme de sécurité : NX.
`CANARY: disabled, FORTIFY: disabled, NX: ENABLED, PIE: disabled, RELRO: disabled`

0x41 Vulnérabilités

Question 9 (5 points) : Parmi les vulnérabilités suivantes, laquelle existe dans ce programme ?

☐ CWE-121 Débordement de tampon dans la pile (*Stack-based Buffer Overflow*)
☐ CWE-122 Débordement de tampon dans le tas (*Heap-based Buffer Overflow*)
☐ CWE-125 Lecture en dehors des limites d'un tampon (*Out-of-bounds Read*)
☐ CWE-193 Erreur d'une unité (*Off-by-one Error*)
☐ CWE-369 Division par zéro
☐ CWE-476 Déréférencement de NULL (*NULL Pointer Dereference*)
☐ CWE-416 Utilisation après libération (*Use After Free*)
☐ CWE-787 Écriture en dehors des limites d'un tampon (*Out-of-bounds Write*)

Question 10 (5 points) : Si on commentait la ligne 13 « `cleanup(chemin);` », le programme contiendrait une vulnérabilité du type CWE-22 Traversée des chemins (*Path Traversal*). Comment exploiterait-on cette vulnérabilité pour afficher le contenu du fichier `/quetes/final/flag` ?

.....

.....

.....

.....

0x42 Segfault

Lorsqu'on exécute le programme avec `AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIJJJJ` comme argument, le programme termine avec une erreur de segmentation. Si on recommence avec `ltrace -i`, on obtient le message « `[0x47474646] SIGSEGV (Segmentation fault)` ».

Question 11 (5 points) : Quelle instruction du code machine a causé l'erreur de segmentation ? Dans quelle fonction du programme C se trouve cette instruction ?

.....

.....

.....

.....

Question 12 (3 points) : Pourquoi le processus ne peut pas continuer son exécution ?

.....

.....

.....

.....

0x50 Pwn 386

Voici le code machine désassemblé de deux fonctions du programme `getflag`.

```
08049228 <show>:
 8049228:55              push   ebp
 8049229:89 e5           mov    ebp,esp
 804922b:83 ec 20        sub    esp,0x20
 804922e:68 08 a0 04 08  push   0x804a008
 8049233:8d 45 e0        lea    eax,[ebp-0x20]
 8049236:50              push   eax
 8049237:e8 34 fe ff ff  call   8049070 <strcpy>
 804923c:83 c4 08        add    esp,0x8
 804923f:ff 75 08        push   DWORD PTR [ebp+0x8]
 8049242:8d 45 e0        lea    eax,[ebp-0x20]
 8049245:50              push   eax
 8049246:e8 15 fe ff ff  call   8049060 <strcat>
 804924b:83 c4 08        add    esp,0x8
 804924e:8d 45 e0        lea    eax,[ebp-0x20]
 8049251:50              push   eax
 8049252:e8 8b ff ff ff  call   80491e2 <cleanup>
 8049257:83 c4 04        add    esp,0x4
 804925a:6a 00           push   0x0
 804925c:8d 45 e0        lea    eax,[ebp-0x20]
 804925f:50              push   eax
 8049260:e8 4b fe ff ff  call   80490b0 <access>
 8049265:83 c4 08        add    esp,0x8
 8049268:85 c0           test   eax,eax
 804926a:75 27           jne    8049293 <show+0x6b>
 804926c:6a 00           push   0x0
 804926e:8d 45 e0        lea    eax,[ebp-0x20]
 8049271:50              push   eax
 8049272:68 17 a0 04 08  push   0x804a017
 8049277:68 17 a0 04 08  push   0x804a017
 804927c:e8 bf fd ff ff  call   8049040 <execlp>
 8049281:83 c4 10        add    esp,0x10
 8049284:68 20 a0 04 08  push   0x804a020
 8049289:e8 c2 fd ff ff  call   8049050 <perror>
 804928e:83 c4 04        add    esp,0x4
 8049291:eb 0c           jmp    804929f <show+0x77>
 8049293:8d 45 e0        lea    eax,[ebp-0x20]
 8049296:50              push   eax
 8049297:e8 b4 fd ff ff  call   8049050 <perror>
 804929c:83 c4 04        add    esp,0x4
 804929f:b8 2d 70 00 00  mov    eax,0x702d
 80492a4:c9              leave
 80492a5:c3              ret
```

```
080492a6 <main>:
 80492a6:55              push   ebp
 80492a7:89 e5           mov    ebp,esp
 80492a9:83 ec 04        sub    esp,0x4
 80492ac:83 7d 08 02     cmp    DWORD PTR [ebp+0x8],0x2
 80492b0:74 14           je     80492c6 <main+0x20>
 80492b2:68 26 a0 04 08  push   0x804a026
 80492b7:e8 74 fd ff ff  call   8049030 <printf>
 80492bc:83 c4 04        add    esp,0x4
 80492bf:b8 01 00 00 00  mov    eax,0x1
 80492c4:eb 4c           jmp    8049312 <main+0x6c>
 80492c6:8b 45 0c        mov    eax,DWORD PTR [ebp+0xc]
 80492c9:8b 40 04        mov    eax,DWORD PTR [eax+0x4]
 80492cc:89 45 fc        mov    DWORD PTR [ebp-0x4],eax
 80492cf:6a 04           push   0x4
 80492d1:68 3e a0 04 08  push   0x804a03e
 80492d6:ff 75 fc        push   DWORD PTR [ebp-0x4]
 80492d9:e8 e2 fd ff ff  call   80490c0 <strncmp>
 80492de:83 c4 0c        add    esp,0xc
 80492e1:85 c0           test   eax,eax
 80492e3:75 1d           jne    8049302 <main+0x5c>
 80492e5:a1 18 b3 04 08  mov    eax,ds:0x804b318
 80492ea:ff 75 fc        push   DWORD PTR [ebp-0x4]
 80492ed:68 43 a0 04 08  push   0x804a043
 80492f2:50              push   eax
 80492f3:e8 a8 fd ff ff  call   80490a0 <fprintf>
 80492f8:83 c4 0c        add    esp,0xc
 80492fb:b8 01 00 00 00  mov    eax,0x1
 8049300:eb 10           jmp    8049312 <main+0x6c>
 8049302:ff 75 fc        push   DWORD PTR [ebp-0x4]
 8049305:e8 1e ff ff ff  call   8049228 <show>
 804930a:83 c4 04        add    esp,0x4
 804930d:b8 01 00 00 00  mov    eax,0x1
 8049312:c9              leave
 8049313:c3              ret
```

0x51 Shellcode

Un attaquant voudrait profiter de la vulnérabilité identifiée pour injecter un shellcode avec un payload du style :
"A" * décalage + adressePile + nop * 100 + shellcode

Question 13 (5 points) :
Quelle est la valeur du décalage à utiliser ?

.....

Question 14 (5 points) :
Expliquez les différentes raisons pourquoi une telle injection de shellcode ne fonctionnerait pas ici.

.....

.....

.....

0x52 Flag

Question 15 (5 points) : Pour avoir le flag `/quetes/final/flag`, l'attaquant veut maintenant faire un *ret2text*. Expliquez simplement en quoi consiste ce type d'attaque.

.....

.....

.....

.....

Question 16 (2 points) : Pour son *ret2text*, l'attaquant a trouvé l'adresse d'une chaîne contenant le texte "`flag\0`". C'est en fait l'adresse de la chaîne littérale utilisée par le `strncmp`. Quelle ironie ! Quelle est cette adresse ?

.....

Question 17 (5 points) : L'attaquant utilise le payload suivant (l'attaquant l'a inutilement compliqué pour le rendre plus impressionnant). Il fait 40 octets, chaque octet est représenté par une paire hexadécimale. Partitionnez les octets du payload en 5 groupes, indiquez le rôle de chaque groupe, et, quand c'est pertinent, à quoi correspondent leurs valeurs. Note : un des groupe ne sert à rien en fait.

```
c15914e8b3bbcdea1053f2d5a36164e23e54c395f83f28920408f970a8f13ea00408fa8a40f43408
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

Question 18 (3 points) : Parmi les contremesures suivantes, lesquelles auraient empêché l'attaquant d'avoir `flag` avec cette attaque ?

☐ PIE (avec ASLR)
☐ RELRO (*relocation readonly*)
☐ Renforcement de source (*fortify source*)
☐ Canari de pile (*stack canary*)

0x53 Pwn

L'attaquant veut avoir un shell privilégié en exécutant `execlp("sh", "sh", "-p", NULL)`.

`execlp` est une fonction variadique mais l'ABI est simple : les paramètres (tous des `const char*`) sont empilés en commençant par le dernier argument et la valeur du résultat est retournée via le registre %eax.

Question 19 (3 points) : Grace au code source et au binaire désassemblé, expliquez comment vous trouvez une chaine "`sh\0`" et indiquez son adresse exacte.

.....

.....

Question 20 (3 points) : Il faut aussi une chaine "`-p\0`". Expliquez pourquoi l'attaquant en a besoin.

.....

.....

Question 21 (3 points) : En utilisant le code machine désassemblé, indiquez l'adresse d'une chaine "`-p\0`".

.....

Question 22 (3 points) : Dessinez la structure et indiquez le contenu que devrait avoir la pile juste avant de brancher sur le code de `execlp` qui donnerait un shell privilégié à l'attaquant.

Question 23 (5 points) : En utilisant les réponses aux questions précédentes, donnez le payload complet qui permet d'avoir ce shell privilégié. Expliquez chacun des éléments du payload.

Question bonus : Le programme `getflag` a été recompilé en 64 bits avec tous les mécanismes de sécurité : `CANARY: ENABLED, FORTIFY: ENABLED, NX: ENABLED, PIE: ENABLED, RELRO: FULL`

Quelle attaque utiliser pour avoir le flag `/quetes/final/flag` ? Indiquez et expliquez votre payload.

0x60 Extraits de pages des manuels de référence en ligne

`int access(const char *pathname, int mode);` vérifie si le processus appelant peut accéder au fichier `pathname`. Si `pathname` est un lien symbolique, il est déréférencé. Le mode indique les vérifications d'accès à effectuer. Il prend la valeur `F_OK` ou un masque contenant un OU binaire d'une des valeurs `R_OK`, `W_OK` et `X_OK`. `F_OK` teste l'existence du fichier. `R_OK`, `W_OK` et `X_OK` testent si le fichier existe et autorise respectivement la lecture, l'écriture et l'exécution. En cas de succès (toutes les permissions demandées sont autorisées, ou mode vaut `F_OK` et le fichier existe), 0 est renvoyé. En cas d'erreur (au moins une permission de mode est interdite, ou mode vaut `F_OK` et le fichier n'existe pas, ou d'autres erreurs se sont produites), -1 est renvoyé et `errno` contient le code d'erreur.

`int execlp(const char *file, const char *arg, ...);` remplace l'image du processus en cours par une nouvelle image du processus. L'argument initial est le chemin d'accès du fichier à exécuter. Si le nom fourni ne contient pas de barre oblique « `/` », le fichier est recherché dans la liste de répertoires, séparés par des deux-points, indiquée dans la variable d'environnement `PATH`. Les arguments « `const char *arg, ...` » décrivent une liste d'un ou plusieurs pointeurs sur des chaînes de caractères terminées par des caractères nuls (`\0`), qui constituent les arguments disponibles pour le programme à exécuter. Par convention, le premier argument doit pointer sur le nom du fichier associé au programme à exécuter. La liste des arguments doit se terminer par un pointeur NULL. `exec()` ne retourne que si une erreur s'est produite. La valeur de retour est alors -1, et `errno` contient le code d'erreur.

`void perror(const char *s);` affiche un message sur la sortie d'erreur standard, décrivant la dernière erreur rencontrée durant un appel système ou une fonction de bibliothèque. D'abord, (si `s` n'est pas `NULL` et `*s` n'est pas un octet nul), la chaîne de caractère `s` est imprimée, suivie d'un deux points (« : ») ou d'un blanc, puis le message, suivi d'un saut de ligne. Le numéro d'erreur est obtenu à partir de la variable externe errno, qui contient le code d'erreur lorsqu'un problème survient.

`sh` (`dash`) est un interpréteur de commandes (shell) compatible `sh` qui exécute les commandes lues depuis l'entrée standard ou depuis un fichier. Si l'interpréteur est lancé avec un identifiant (de groupe) d'utilisateur effectif différent de l'identifiant (de groupe) d'utilisateur réel et si l'option `-p` n'est pas fournie, alors les identifiants effectifs sont réinitialisés aux identifiants réels. Ceci est fait par défaut pour éviter les utilisations incorrectes de programmes privilégiés `setuid`/`setgid` via des fonctions comme `system()` ou `popen()`.

`char *strcat(char *dest, const char *src);` ajoute la chaîne `src` à la fin de la chaîne `dest` en écrasant l'octet nul (`\0`) final à la fin de `dest`, puis en ajoutant un nouvel octet nul final. La chaîne `dest` doit être assez grande pour accueillir le résultat. Si `dest` n'est pas assez grande, le comportement du programme est imprévisible. Les dépassements de tampon font partie des moyens préférés pour attaquer les programmes sécurisés.

`char *strcpy(char *dest, const char *src);` copie la chaîne pointée par `src`, y compris le caractère nul (`\0`) final dans la chaîne pointée par `dest`. La chaîne `dest` doit être assez grande pour accueillir la copie. Attention aux dépassements de tampon ! Si la chaîne de destination d'un `strcpy()` n'est pas suffisamment grande, n'importe quoi peut survenir. Un dépassement de tampon pour une chaîne de taille fixe est la technique favorite de pirates pour prendre le contrôle d'une machine. À chaque fois qu'un programme lit ou copie des données dans un tampon, le programme doit d'abord vérifier qu'il y a suffisamment de place. Ça peut ne pas être nécessaire si vous pouvez montrer qu'un dépassement est impossible, mais faites attention : les programmes changes au cours du temps, et ce qui était impossible peut devenir possible.

`size_t strlen(const char *s);` calcule la longueur de la chaîne de caractères `s`, sans compter l'octet nul final.

`int strncmp(const char *s1, const char *s2, size_t n);` compare les `n` (au plus) premiers octets des deux chaînes `s1` et `s2`. Elle renvoie un entier négatif, nul, ou positif, si `s1` est respectivement inférieure, égale ou supérieure à `s2`.

0x70 Annexe (détachable)

0x71 39 instructions Pep/8

Spécificateur Binaire	Hex	Instruction	Signification	Modes d'adressage	Conditions affectées
00000000	00	STOP	Arrêt de l'exécution du programme		
00000001	01	RETTR	Retour d'interruption		
00000010	02	MOVSPA	Placer SP dans A		
00000011	03	MOVFLGA	Placer NZVC dans A		
0000010a	04, 05	BR	Branchement inconditionnel	i,x	
0000011a	06, 07	BRLE	Branchement si inférieur ou égal	i,x	
0000100a	08, 09	BRLT	Branchement si inférieur	i,x	
0000101a	0A, 0B	BREQ	Branchement si égal	i,x	
0000110a	0C, 0D	BRNE	Branchement si non égal	i,x	
0000111a	0E, 0F	BRGE	Branchement si supérieur ou égal	i,x	
0001000a	10, 11	BRGT	Branchement si supérieur	i,x	
0001001a	12, 13	BRV	Branchement si débordement	i,x	
0001010a	14, 15	BRC	Branchement si retenue	i,x	
0001011a	16, 17	CALL	Appel de sous-programme	i,x	
0001100r	18, 19	NOTr	NON bit-à-bit du registre		NZ
0001101r	1A, 1B	NEGr	Opposé du registre		NZV
0001110r	1C, 1D	ASLr	Décalage arithmétique à gauche du registre		NZVC
0001111r	1E, 1F	ASRr	Décalage arithmétique à droite du registre		NZC
0010000r	20, 21	ROLr	Décalage cyclique à gauche du registre		C
0010001r	22, 23	RORr	Décalage cyclique à droite du registre		C
001001nn	24–27	NOPn	Interruption unaire pas d'opération		
00101aaa	28–2F	NOP	Interruption non unaire pas d'opération	i	
00110aaa	30–37	DECI	Interruption d'entrée décimale	d,n,s,sf,x,sx,sxf	NZV
00111aaa	38–3F	DECO	Interruption de sortie décimale	i,d,n,s,sf,x,sx,sxf	
01000aaa	40–47	STRO	Interruption de sortie de chaîne	d,n,sf	
01001aaa	48–4F	CHARI	Lecture caractère	d,n,s,sf,x,sx,sxf	
01010aaa	50–57	CHARO	Sortie caractère	i,d,n,s,sf,x,sx,sxf	
01011nnn	58–5F	RETn	Retour d'un appel avec n octets locaux		
01100aaa	60–67	ADDSP	Addition au pointeur de pile (SP)	i,d,n,s,sf,x,sx,sxf	NZVC
01101aaa	68–6F	SUBSP	Soustraction au pointeur de pile (SP)	i,d,n,s,sf,x,sx,sxf	NZVC
0111raaa	70–7F	ADDr	Addition au registre	i,d,n,s,sf,x,sx,sxf	NZVC
1000raaa	80–8F	SUBr	Soustraction au registre	i,d,n,s,sf,x,sx,sxf	NZVC
1001raaa	90–9F	ANDr	ET bit-à-bit du registre	i,d,n,s,sf,x,sx,sxf	NZ
1010raaa	A0–AF	ORr	OU bit-à-bit du registre	i,d,n,s,sf,x,sx,sxf	NZ
1011raaa	B0–BF	CPr	Comparer au registre	i,d,n,s,sf,x,sx,sxf	NZVC
1100raaa	C0–CF	LDr	Placer 2 octets (un mot) dans registre	i,d,n,s,sf,x,sx,sxf	NZ
1101raaa	D0–DF	LDBYTEr	Placer octet dans registre (bits 0-7)	i,d,n,s,sf,x,sx,sxf	NZ
1110raaa	E0–EF	STr	Ranger registre dans 1 mot	d,n,s,sf,x,sx,sxf	
1111raaa	F0–FF	STBYTEr	Ranger registre (bits 0-7) dans 1 octet	d,n,s,sf,x,sx,sxf	

0x72 8 directives Pep/8

Directive	Signification
.BYTE	Réserve 1 octet mémoire avec valeur initiale.
.WORD	Réserve 1 mot mémoire avec valeur initiale.
.BLOCK	Réserve un nombre d'octets mis à zéro.
.ASCII	Réserve l'espace mémoire pour une chaîne de caractères (ex : "Chaîne").
.ADDRSS	Réserve 1 mot mémoire pour un pointeur.
.EQUATE	Attribue une valeur à une étiquette.
.END	Directive obligatoire de fin d'assemblage qui doit être à la fin du code.
.BURN	Le programme se terminera à l'adresse spécifiée par l'opérande. Ce qui suit .BURN est écrit en ROM.

0x73 8 modes d'adressage Pep/8

Mode	aaa	a	Lettres	Opérande
Immédiat	000	0	i	Spec
Direct	001		d	mem[Spec]
Indirect	010		n	mem[mem[Spec]]
Sur la pile	011		s	mem[PP+Spec]]
Indirect sur la pile	100		sf	mem[mem[PP+Spec]
Indexé	101	1	x	mem[Spec + X]
Indexé sur la pile	110		sx	mem[PP+Spec+X]]
Indirect indexé sur la pile	111		sxf	mem[mem[PP+Spec]+X]

0x74 9 registres Pep/8

Symbole	r	Description	Taille
N		Négatif	1 bit
Z		Nul (Zero)	1 bit
V		Débordement (Overflow)	1 bit
C		Retenue (Carry)	1 bit
A	0	Accumulateur	2 octets (un mot)
X	1	Registre d'index	2 octets (un mot)
PP		Pointeur de pile (SP)	2 octets (un mot)
CO		Compteur ordinal (PC)	2 octets (un mot)
IR{		Spécificateur d'instruction	1 octet
	Spec	Spécificateur d'opérande	2 octets (un mot)

0x75 Table ASCII

Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex	
0	00	NUL '\0'	32	20	Espace ' '	64	40	@	96	60	‘
1	01	SOH (début d'en-tête)	33	21	!	65	41	A	97	61	a
2	02	STX (début de texte)	34	22	”	66	42	B	98	62	b
3	03	ETX (fin de texte)	35	23	#	67	43	C	99	63	c
4	04	EOT (fin de transmission)	36	24	$	68	44	D	100	64	d
5	05	ENQ (demande)	37	25	%	69	45	E	101	65	e
6	06	ACK (accusé de réception)	38	26	&	70	46	F	102	66	f
7	07	BEL '\a' (sonnerie)	39	27	’	71	47	G	103	67	g
8	08	BS '\b' (espace arrière)	40	28	(	72	48	H	104	68	h
9	09	HT '\t' (tab. horizontale)	41	29	)	73	49	I	105	69	i
10	0A	LF '\n' (changement ligne)	42	2A	*	74	4A	J	106	6A	j
11	0B	VT '\v' (tab. verticale)	43	2B	+	75	4B	K	107	6B	k
12	0C	FF '\f' (saut de page)	44	2C	,	76	4C	L	108	6C	l
13	0D	CR '\r' (retour chariot)	45	2D	-	77	4D	M	109	6D	m
14	0E	SO (hors code)	46	2E	.	78	4E	N	110	6E	n
15	0F	SI (en code)	47	2F	/	79	4F	O	111	6F	o
16	10	DLE (échap. transmission)	48	30	0	80	50	P	112	70	p
17	11	DC1 (commande dispositif 1)	49	31	1	81	51	Q	113	71	q
18	12	DC2 (commande dispositif 2)	50	32	2	82	52	R	114	72	r
19	13	DC3 (commande dispositif 3)	51	33	3	83	53	S	115	73	s
20	14	DC4 (commande dispositif 4)	52	34	4	84	54	T	116	74	t
21	15	NAK (accusé réception nég.)	53	35	5	85	55	U	117	75	u
22	16	SYN (synchronisation)	54	36	6	86	56	V	118	76	v
23	17	ETB (fin bloc transmission)	55	37	7	87	57	W	119	77	w
24	18	CAN (annulation)	56	38	8	88	58	X	120	78	x
25	19	EM (fin de support)	57	39	9	89	59	Y	121	79	y
26	1A	SUB (substitution)	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC (échappement)	59	3B	;	91	5B	[	123	7B	{
28	1C	FS (séparateur fichiers)	60	3C	<	92	5C	\	124	7C	|
29	1D	GS (séparateur de groupes)	61	3D	=	93	5D	]	125	7D	}
30	1E	RS (sép. enregistrements)	62	3E	>	94	5E	^	126	7E	~
31	1F	US (sép. de sous-articles)	63	3F	?	95	5F	_	127	7F	DEL