Le Langage d'Assemblage et l'Assembleur (SRC)

IFT1225

Dép. d'IRO, Université de Montréal
(professeur E. Cerny)

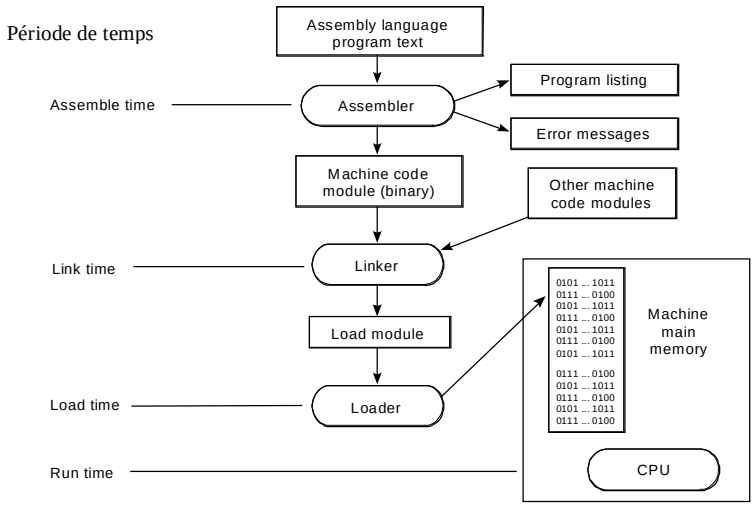
Sujets

- Le langage machine, le langage d'assemblage
- La tâche de l'assembleur
- L'organisation d'un assembleur
- Programmation en langage d'assemblage
  - Calcul et affectation de variables
  - Exécution conditionnelle, branchement
  - Boucle d'itération
  - Sousprogrammes
    - Pile
    - Transfert de contrôle
    - Passage de paramètres
  - Traitement d'exceptions

Le langage machine et le langage d'assemblage

- Programme en langage machine - séquence d'instructions et des blocs de données, en binaire, dans la mémoire
- Le langage d'assemblage - forme symbolique d'un programme en langage machine
  - lisible par les êtres humaines (+/-)
  - constantes, adresses et noms d'instructions symboliques
  - arithmétique lors d'assemblage - calculs d'adresses, constantes
  - instructions synthétiques (pas supporté par notre assembleur)
  - expansion des macroinstructions (pas supporté par notre assembleur)
  - directives à l'assemblage (pseudo-instructions)
    - allocation de mémoire
    - initialisation de mémoire
    - assemblage conditionnel (pas supporté par notre assembleur)

Le processus d'assemblage



Langage d'assemblage SRC

- Structure d'une ligne du code:

  étiquette: op opérandes ; commentaire

- Pseudo-instructions - directives à l'assembleur:

étiquette:	.org	valeur	; l'adresse de départ (origine)
étiquette:	.equ	valeur	; const. symbolique
étiquette:	.dc	valeur [, valeur]; const. en mémoire (mot)	
étiquette:	.dcb	valeur [, valeur]; const. en mémoire (octet)	
étiquette:	.dch	valeur [, valeur]; const. en mémoire (demi-mot)	
étiquette:	.db	compte	;réserver "compte" octets en mémoire
étiquette:	.dh	compte	;réserver "compte" demi-mots
étiquette:	.dw	compte	;réserver "compte" mots

- Un programme commence d'habitude par des définitions de constantes symboliques, défintion de l'origine, d'une région de données (peut être après le code) et le code du programme.

Exemple

#define seuil 200;
calculer *y = max(x, seuil)* comme *y = x; si seuil-x >= 0 alors y = seuil;*

```
seuil:  .equ    200
base:   .org    1000    ; début de la section de données
x:      .dw     1
y:      .dw     1
debut:  .org    2000    ; début de la section du  programme
        lar     r0, max ; adresse de branchement
        la      r1, seuil ; charger seuil dans r1
        ld      r3, x   ; charger x dans r3 (le resultat?)
        sub     r1, r1, r3 ; comparer x avec limit
        brmi    r0, r1  ; si r1<0 va à max car x>limit
        la      r3, seuil ; charger seuil
max:    st      r3, y   ; ranger le résultat
        stop            ; fin du programme
```

Itération

pour (i = 0; i < 10; i++) tableau[i] = i;

```
limit:    .equ     -10
          .org     1000
tableau:  .dw      20
debut:    .org     5000
          la       r1, 0
          la       r2,  tableau
          lar      r3, boucle
boucle:   st       r1, 0(r2)
          addi     r2, r2, 4
          addi     r1, r1, 1
          addi     r0, r1, limit
          brmi     r3, r0
. . .
```


Procédures et fonctions

- Réutilisation d'un algorithme à plusieurs endroits dans le programme
- Facilite maintenance du programme - une seule copie à mettre à jour
- Le programme complet peut prendre moins d'espace en mémoire
- Vitesse d'exécution moindre - temps pour transférer le contrôle, passer les paramètres et retourner les résultats.
- Transfert de contrôle avec les instructions **br**anch-and-**l**ink qui préservent l'adresse de retour dans un registre (SRC)
- Petites procédures avec peu de paramètres et locales à un seul programme - passage de paramètre et des résultats dans des registres
- Procédures réutilisables et celles écrites en langage de haut niveau utilisent un protocole fixe: en général, l'utilisation d'une ***pile*** pour sauvgarder l'adresse de retour, passer des paramètres et retrouver des résultats. Fonctions retournent leur résultat dans un registre (valeur ou son adresse). Fonctions prédéterminées de certains registres.
- Passage des paramètres par valeur ou par référence (adresse)

Pile "dernier-venu-premier-sorti" (en SRC)

opérations: **empiler** et **dépiler** (*push* et *pop, LIFO "Last-In-First-Out"*)

supposons r1 = pointeur de pile

initialiser le pointeur de pile par:

```
        la      r1, base_de_pile
```

empiler un mot qui est dans r2:

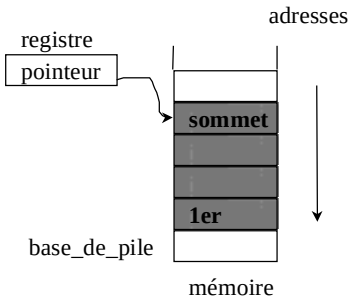
```
push:   addi    r1, r1, -4
        st      r2, (r1)
```

dépiler un mot dans r2:

```
pop:    ld      r2, (r1)
        addi    r1, r1, 4
```

mettre k-ième mot en sous du sommet dans r2:

```
get:    ld      r2, k*4(r1)
```



(Peut être réalisé dans l'autre sens, avec adresses croissants)

Procédure max(x, y, z) *z = max(x, y)*

```
;pointeur de pile = r1
; r0 = adr. de branch.
; r2 = PC temp.

appel:  addi    r1, r1, -8 ;push
        st      r2, (r1)   ; arg. 1
        st      r3, 4(r1)  ; arg. 2
        la      r0, max
; appeler max, PC dans r2
        brl     r2, r0
        ld      r2, (r1)  ; résultat
        addi    r1, r1, 4; pop
        . . .
```

Si *max* était une fonction… retourner le résultat *z* dans un registre déterminé par le protocole d'appel utilisé

```
; procédure max(x, y, z)
; passage d'args par valeur
; sur la pile pointée par r1
; z = max (x, y) sur la pile au retour
max:  addi   r1, r1, -4 ; push
      st     r2, (r1)   ; adr. de ret
      lar    r0, max1   ; adr. branch.
      ld     r2, 4(r1)  ; arg. 1
      ld     r3, 8(r1)  ; arg. 2
      sub    r2, r2, r3
      brmi   r0, r2
      ld     r3, 4(r1)
max1: st     r3, 8(r1)  ; résultat
      ld     r2, (r1)   ; pop adr. ret.
      addi   r1, r1, 8  ; et args.
      br     r2         ; retour ...
```

Passage par adresse??

Traitement d'Exceptions

- Exception = un événement hors de séquence normale d'exécution d'instructions, force un changement de cette séquence
- Externe - appelée plus souvent une *interruption*
  - causée par un événement à l'extérieur du processeur, p.e., entréee/sortie
  - habituellement *asynchrone* à l'exécution d'instructions
- Interne - appelée souvent un *trap*
  - causée par une conditions dans le processeur, p.e., perte d'alimentation - *asynchrone*, ou suite à l'exécution d'instructions - *synchrone*, p.e., division par 0, débordement de capacité, adresse inconnue ou invalide, erreur de parité dans la mémoire ou sur le bus, violation de protection d'accès à la mémoire, mauvais alignement de mots, défaut de page (mémoire virtuelle), etc.
- Traitement d'exceptions (interruptions et traps) uniforme; transfert de contrôle au système d'exploitation

Traitement d'exceptions

- Système des *priorités* pour différentier l'urgence du traitement
- Interruptions / Traps *masquées* et *nonmasquées* individuellement et ou globalement
- Interruptions traitées entre l'exécution de 2 instrcutions
- Certains traps doivent être traités lors de l'exécution d'instruction, p.e., défaut de page.
- Dans les machines modernes, le transfert de contrôle à une routine de service d'interruption se fait à l'aide de *vecteurs d'interruption*
  - un bloc des mots consécutifs en mémoire (souvent à partir de l'adresse 0), regroupés en *vecteurs de mots*
  - p.e., vecteur de 2 mots:
    - 1er mot: l'adresse ou la 1ère instruction de la routine de traitement
    - 2ème mot: un nouveau mot de statut du processeur (masque d'interruption, mode d'opération - usager/système, etc.)
- Autres possibilités: un seule vecteur + registre d'identification

Traitement d'exceptions

- Préservation de l'état du programme interrompu:
  - il n'est pas possible de prédire l'endroit exacte dans le programme ou l'interruption va arriver
  - une fois traitée, il faut retourner de la routine de service dans le programme interrompu et continuer…
  - Lors du transfert de contrôle dans la routine de service il faut sauver l'état du processeur (p.e., sur une pile)
  - Lors du retour du service il faut restaurer cet état.
- Lors du *changement de context* (du programme interrompu vers le service, ou entre 2 programmes par le système d'exploitation) il faut pouvoir inhiber toute interruption (sauf exceptions catastrophiques - NMI)

Traitement d'exceptions (SRC)

- Matériel additionnel - interruptions externes:
  - signaux *ireq* et *iack*
  - IE (*Interrupt Enable*): drapeau d'autorisation d'interruption
  - IPC <31..0>: registre d'adresse de retour
  - II<31..0>: registre d'identification de la source d'interruption
  - Isrc_info<15..0>: info fourni par la source -> dans II<15..0>
  - Isrc_vect<7..0>: type d'interruption fourni par la source
    forme l'adresse du vecteur d'interruption:
    Ivect<31..0> := 20@0#Isrc_vect<7..0>#4@0
    chaque vecteur consiste en 16 octets (4 mots) qui doivent contenir les instructions pour traiter l'interruption ou brancher à une routine de service.
- Instructions additionnelles: svi, ri, een, edi, rfi

Interprétation d'instructions modifiée

instruction_interpretation :=
(¬ Run∧Strt → Run ← 1:
Run∧¬ (ireq∧IE) → (I ← M[PC]: PC ← PC + 4;
instruction_execution):
Run∧(ireq∧IE) → (IPC ← PC⟨31..0⟩:
II⟨15..0⟩ ← Isrc_info⟨15..0⟩: iack ← 1:
IE ← 0: PC ← Ivect⟨31..0⟩; iack ← 0);
instruction_interpretation);

Remarques:
- IE mis à 0 - interruptions inhibées à l'entrée dans la routine de service, le programme doit ainsi décider si les interruptions peuvent être permises lors de service de l'interruption courante
- Contrôle transféré à l'adresse Ivect
- Avec plusieurs demandes d'interruption simultanées, sélection établie par un système de priorités externe (signaux multiple et/ou "daisy chain")

Instructions liées aux interruptions

- **Sauver et restaurer II et IPC:**

svi (:= op = 16) → (R[ra]⟨15..0⟩ ← II⟨15..0⟩: R[rb] ← IPC⟨31..0⟩):
ri (:= op = 17) → (II⟨15..0⟩ ← R[ra]⟨15..0⟩ : IPC⟨31..0⟩ ← R[rb]):

- **Retour de l'interruption (2 opérations indivisibles!)**

rfi (:= op = 29) → (PC ← IPC: IE ← 1): **est-ce un problème?**

- **Autoriser et inhiber le système de traitement d'exceptions**

een (:= op = 10) → (IE ← 1):
edi (:= op = 11) → (IE ← 0):

- **Programmeur doit**
  - initialiser le(s) vecteurs d'interruptions (charger par défaut rfi ou halt …?)
  - fournir le(s) routine(s) de service d'interruption

Routine de service d'interruption

- **Orgaisation possible:**

```
                .org    16*IntNumber
VecteurX:       br      ServeX  ; transférer contrôle a la routine
                . . .
                .org    1000
ServeX:         -empiler registres sur la pile (via  r1)
                svi     r2, r3  ; sauver l'état: II, IPC dans r2, r3
                -empiler r2, r3 sur la pile
                een             ; autorise d'autres interruptions
                -traitement spécifique à l'interruption
                edi             ; inhiber interruptions
                -depiler II, IPC dans r2, r3
                ri      r2, r3  ; restaurer II et IPC
                -depiler registres de la pile
                rfi             ; fin du traitement d'interruption
```


Exceptions lors de l'exécution d'une instruction

- Certaines exceptions arrivent pendant l'exécution d'instruction
  - défaut de page
  - exécution d'instructions longues (processeurs CISC) (*move string*)
  - mémoire inexistante
- Redémarrer l'instruction après avoir traité l'exception (défaut de page)
  - préserver le micro-état de la machine, ou
  - restaurer l'état avant l'exécution de l'instruction-cause de l'exception, et redémarrer
- Implantations en pipeline des processeurs doivent toujours traiter des interruptions pendant l'exécution de l'instruction
  - interruptions précises: faut restaurer l'état et continuer
  - interruption imprécises: min. d'information, arrête du programme, transfert au système d'exploitation
  - à voir avec processeurs à pipeline ...