0. Introduction

IFT209 – Programmation système (Hiver 2019)

Enseignant

Michael Blondin

@ michael.blondin@usherbrooke.ca

info.usherbrooke.ca/mblondin

bureau D4-1024-1 au 1er étage

Horaire

Cours magistraux	Laboratoires
Lundi	Jeudi
15h30 – 17h20	08h30 – 10h20
D3-2035	D4-1023/1017

Horaire

Cours magistraux	Laboratoires
Lundi	Jeudi
15h30 – 17h20	08h30 – 10h20
D3-2035	D4-1023/1017 *

* Au D3-2035 les **10 jan.**, 17 jan. et 11 avr.

But du cours

- Comprendre l'architecture d'un ordinateur
- Connaître les types élémentaires de données et leur représentation
- Savoir manipuler les données en mémoire
- Se familiariser avec les langages de bas de niveau

Préalables

Préalable:

- IFT159 – Analyse et programmation

Préalable à:

- IFT320 – Systèmes d'exploitation
- IFT585 – Télématique

Utile pour:

- IFT580 – Compilation et interprétation des langages

Contenu

1. Systèmes de numération
2. Architecture des ordinateurs
3. Accès aux données
4. Prog. en assembleur
5. Nombres entiers
6. Tableaux
7. Structures de contrôle
8. Chaînes de bits/caractères
9. Sous-programmes
10. Nombres en virgule flottante
11. Entrées/sorties

Contenu

1. Systèmes de numération **2h**

- Écriture de nombres dans un système de numération
- Conversion de nombres entre systèmes

Contenu

2. Architecture des ordinateurs **4h**

- Architecture de von Neumann
- Mémoire principale, processeur et registres
- Jeux d'instructions
- Organisation d'un ordinateur

Contenu

3. Accès aux données — 4h

- Données
- Adresses
- Modes d’adressage
- Étapes de la vie d’un programme

Contenu

4. Programmation en assembleur **4h**

- Survol à partir de courts programmes
- Différence entre instructions et pseudo-instructions
- Programmation de haut niveau des entrées/sorties.

Contenu

5. Nombres entiers **4h**

- Représentations des entiers signés/non signés
- Report et débordement
- Instructions arithmétiques

Contenu

6. Tableaux **2h**

- Tableaux à une dimension
- Tableaux à deux dimensions

Contenu

7. Structures de contrôles — 4h

- Condition, branchement, séquence, sélection et itération
- Appel et retour de sous-programmes
- Notion d'environnement
- Récursivité

Contenu

8. Chaînes de bits/caractères **4h**

- Opérations logiques, décalages
- Codes de représentation de caractères
- Opérations sur les (chaînes de) caractères

Contenu

9. Sous-programmes **4h**

- Appel et retour
- Passage de paramètres
- Sauvegarde et récupération
- Récursivité

Contenu

10. Nombres en virgule flottante **4h**

- Représentations des nombres en virgule flottante
- Erreurs d'arrondi et de troncation
- Dépassement de capacité
- Norme IEEE 754
- Instructions arithmétiques

Contenu

11. Entrées/sorties — 10h

- Interrogation de bits d'états
- Interruptions
- Niveaux de priorités des interruptions
- Illustration à l'aide de dispositifs simples

Matériel

- Richard St-Denis: *L'architecture du processeur SPARC et sa programmation en langage d'assemblage*

- Copies papier disponibles au A8-151 pour ∼20$

Matériel

- Richard St-Denis: *L'architecture du processeur SPARC et sa programmation en langage d'assemblage*
- Copies papier disponibles au A8-151 pour ∼20$
- Complémenté par des notes électroniques

Évaluation

Laboratoires	10% ($6 \times 1{,}\overline{6}\%$)
Devoirs	30% ($5 \times 6\%$)
Examen périodique	25%
Examen final	35%

Devoirs

- 5 devoirs
- À faire en équipes de deux
- Environ 2 semaines par devoir

Laboratoires

- 6 laboratoires
- À faire en équipes de deux
- Échéancier:

Affichés:	mercredi	à 08h30
En classe:	jeudi	à 08h30
À remettre:	dimanche	à 23h59

Rétroaction non officielle vers la mi-session

Calendrier

Sujets	Laboratoire	Devoirs
1: Introduction, sys. numération	Cours	Devoir 1 ∼12 jours
2: Architecture, assembleur	Cours	
3: Accès aux données	Labo 1	Devoir 2 ∼14 jours
4: Nombres entiers	Travail devoir 2	
5: Tableaux	Labo 2	Devoir 3 ∼14 jours
6: Structures de contrôle	Labo 3	
7: Révision	Labo 4	—
8: Examen périodique	—	—

Calendrier

Sujets	Laboratoire	Devoirs
9: Relâche	—	—
10: Chaînes	Labo 5	Devoir 4 ∼14 jours
11: Sous-programmes	Travail devoir 4	
12: Nombre virg. flottante	Labo 6	
13: Entrées/sorties	Travail devoir 5	Devoir 5 ∼16 jours
14: Entrées/sorties	Révision	
15: Examen final	—	—
16: Examen final	—	—

Disponibilités

Sur **rendez-vous** à mon bureau

et

1h / semaine (à choisir maintenant)

Page Web

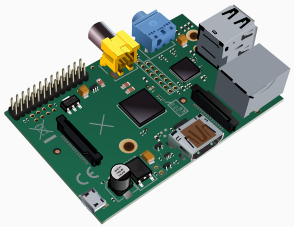
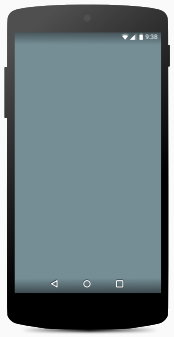
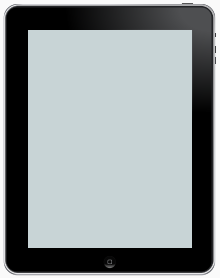
 info.usherbrooke.ca/mblondin/ift209

Langages d’assemblage

Langages

Majorité du cours:

ARMv8 (AArch64/ARM64)

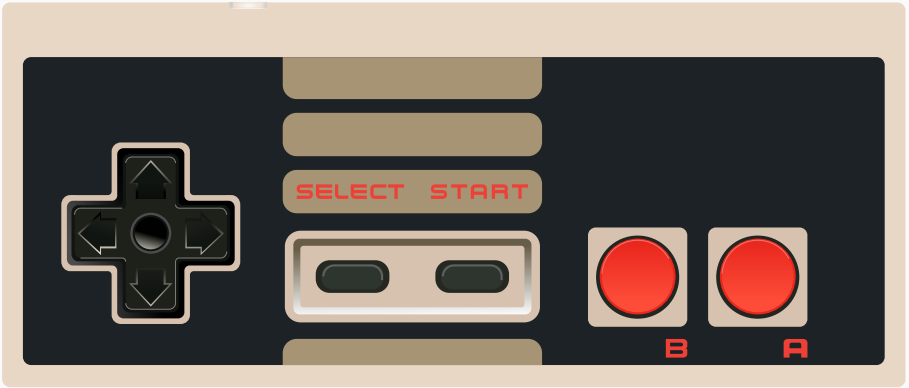

Langages

Majorité du cours:

ARMv8 (AArch64/ARM64)

Fin du cours (entrées/sorties):

NMOS 6502



Aujourd'hui:

(architecture ouverte et libre)

Courts programmes

1. Que fait ce code?

```
addi x1, x0, 2
addi x2, x0, 3
add  x3, x1, x2
```

2. Que vaut x3 en fonction de x1 à la fin?

```
  addi x2, x0, 0
  addi x3, x0, 0
debut:
  addi x2, x2, 1
  add  x3, x3, x2
  bne  x2, x1, debut
```

2. Que vaut x3 en fonction de x1 à la fin?

```
  addi x2, x0, 0      # x2 = 0
  addi x3, x0, 0      # x3 = 0
debut:                # faire:
  addi x2, x2, 1      #   x2 += 1
  add  x3, x3, x2     #   x3 += x2
  bne  x2, x1, debut # boucler si x2 != x1
```

2. Que vaut x3 en fonction de x1 à la fin?

```
  li x2, 0              # x2 = 0
  li x3, 0              # x3 = 0
debut:                  # faire:
  addi x2, x2, 1        #   x2 += 1
  add  x3, x3, x2       #   x3 += x2
  bne  x2, x1, debut # boucler si x2 != x1
```

3. Que vaut x3 en fonction de x1 à la fin?

```
  li x2, 0
debut:
  add  x2, x2, x1
  addi x1, x1, -1
  bne  x1, x0, debut
```

3. Que vaut x3 en fonction de x1 à la fin?

```
  li x2, 0                # x2 = 0
debut:                    # faire:
  add  x2, x2, x1         #   x2 += x1
  addi x1, x1, -1         #   x1 -= 1
  bne  x1, x0, debut # boucler si x1 != 0
```

4. Que vaut x3 en fonction de x1 à la fin?

```
addi x2, x1, 1
mul  x3, x1, x2
addi x2, x0, 2
div  x3, x3, x2
```

5. Que vaut x2 en fonction de x1 à la fin?

```
   li   x2, 0
   li   x3, 1
debut:
   beq  x1, x0, fin
   add  x4, x2, x3
   addi x2, x3, 0
   addi x3, x4, 0
   addi x1, x1, -1
   j    debut
fin:
```

5. Que vaut x2 en fonction de x1 à la fin?

```
  li   x2, 0          # x2 = 0
  li   x3, 1          # x3 = 1
debut:                #
  beq  x1, x0, fin    # boucler jusqu'à x1 == 0:
  add  x4, x2, x3     #   x4  = x2 + x3
  addi x2, x3, 0      #   x2  = x3
  addi x3, x4, 0      #   x3  = x4
  addi x1, x1, -1     #   x1 -= 1
  j    debut          #
fin:                  # fin
```

5. Que vaut x2 en fonction de x1 à la fin?

```
  li   x2, 0          # x2 = 0
  li   x3, 1          # x3 = 1
debut:                #
  beq  x1, x0, fin    # boucler jusqu'à x1 == 0:
  add  x4, x2, x3     #   x4  = x2 + x3
  mv   x2, x3         #   x2  = x3
  mv   x3, x4         #   x3  = x4
  addi x1, x1, -1     #   x1 -= 1
  j    debut          #
fin:                  # fin
```

6. Que fait ce code?

```
.data
chaine: .byte 97
        .byte 108
        .byte 108
        .byte 111
        .byte 0
.text
  la   x1, chaine
debut:
  lb   x2, 0(x1)
  beq  x2, x0, fin
  addi x2, x2, -32
  sb   x2, 0(x1)
  addi x1, x1, 1
  j    debut
fin:
```

6. Que fait ce code?

```
.data
chaine: .byte 97        # chaine = "allo"
        .byte 108       #
        .byte 108       #
        .byte 111       #
        .byte 0         #
.text                   #
  la   x1, chaine       # x1 = adr(chaine)
debut:                  # faire:
  lb   x2, 0(x1)        #   x2 = mem[x1]
  beq  x2, x0, fin      #   sortir si x2 == 0
  addi x2, x2, -32      #   x2 -= 32
  sb   x2, 0(x1)        #   mem[x1] = x2
  addi x1, x1, 1        #   x1 += 1
  j    debut            # boucler
fin:                    #
```

6. Que fait ce code?

```
.data
chaine: .asciiz "allo" # chaine = "allo"
                       #
                       #
                       #
                       #
                       #
.text                  #
  la   x1, chaine      # x1 = adr(chaine)
debut:                 # faire:
  lb   x2, 0(x1)       #   x2 = mem[x1]
  beq  x2, x0, fin     #   sortir si x2 == 0
  addi x2, x2, -32     #   x2 -= 32
  sb   x2, 0(x1)       #   mem[x1] = x2
  addi x1, x1, 1       #   x1 += 1
  j    debut           # boucler
fin:                   #
```

7. Comment ce code est-il compilé?

```
int fib(int n) {
  int x = 0;
  int y = 1;
  int z;

  for (int i = n; i > 0; i--) {
    z = x + y;
    x = y;
    y = z;
  }

  return x;
}
```

7. Comment ce code est-il compilé?

```
fib:
  mv    a5, a0
  blez  a0, .L4
  li    a0, 1
  li    a4, 0
  j     .L3
.L5:
  mv    a0, a3
.L3:
  addiw a5, a5, -1
  addw  a3, a4, a0
  mv    a4, a0
  bnez  a5, .L5
  ret
.L4:
  li    a0, 0
  ret
```

7. Comment ce code est-il compilé?

```
fib:                    #
  mv     a5, a0         #     i = n;
  blez   a0, .L4        #     if (i <= 0) goto L4;
  li     a0, 1          #     y = 1;
  li     a4, 0          #     x = 0;
  j      .L3            #     goto L3;
.L5:                    # L5:
  mv     a0, a3         #     y = z;
.L3:                    # L3:
  addiw  a5, a5, -1     #     i--;
  addw   a3, a4, a0     #     z = x + y;
  mv     a4, a0         #     x = y;
  bnez   a5, .L5        #     if (i != 0) goto L5;
  ret                   #     return y;
.L4:                    # L4:
  li     a0, 0          #
  ret                   #     return 0;
```

8. Combien de fois la boucle est-elle itérée?

```
float x = 0.1;
float y = 0;

while (y != 1.0) {
  y += x;
}
```

9. nop est exécuté pour quelles valeurs de x1?

```
  srli x2, x1, 1   # x2 = x1 >> 1
  andi x2, x2, 1   # x2 = x2 & 1
  beqz x2, fin
  nop
fin:
```

10. Comment se font les entrées/sorties?

```c
#include <stdio.h>

int main() {
  int m, n;

  scanf("%d", &m);
  scanf("%d", &n);
  printf("%d\n", m + n);
}
```