Révision mi-session

GIF-1001 Ordinateurs: Structure et Applications, Hiver 2016

Jean-François Lalonde

Examen mi-session — logistique

- mardi 23 février de 14h30 à 16h20 (même heure que le cours)
- Nous devrons diviser le groupe en **trois**:
  - Andande à Charbonneau: **PLT**-4118
  - Corado-Castillo à Iquira: **VCH**-3870
  - Jean à Weber-Boisvert: **VCH**-2860

Format des nombres

- Entiers non-signés
- Entiers signés, notation complément-2
  - exemple: -5 en complément-2 sur 4 bits? sur 8 bits?
- IEEE754
  - pas besoin de connaître le format par coeur, mais il faut savoir l'interpréter.
- ASCII
  - comprendre comment ça fonctionne

Point hyper important à retenir™ #4

- A priori, nous ne pouvons pas savoir ce qu'une chaîne binaire signifie.
  - Ex: que veut dire 0x416C6C6F (sur 32 bits)?
  - La bonne réponse est: ça dépend!

entier non-signé	1097624687
entier signé	278356550
rationnel	14.47764
caractères ASCII	Allo

  - Il nous *faut* donc savoir quel format utiliser pour bien interpréter les données

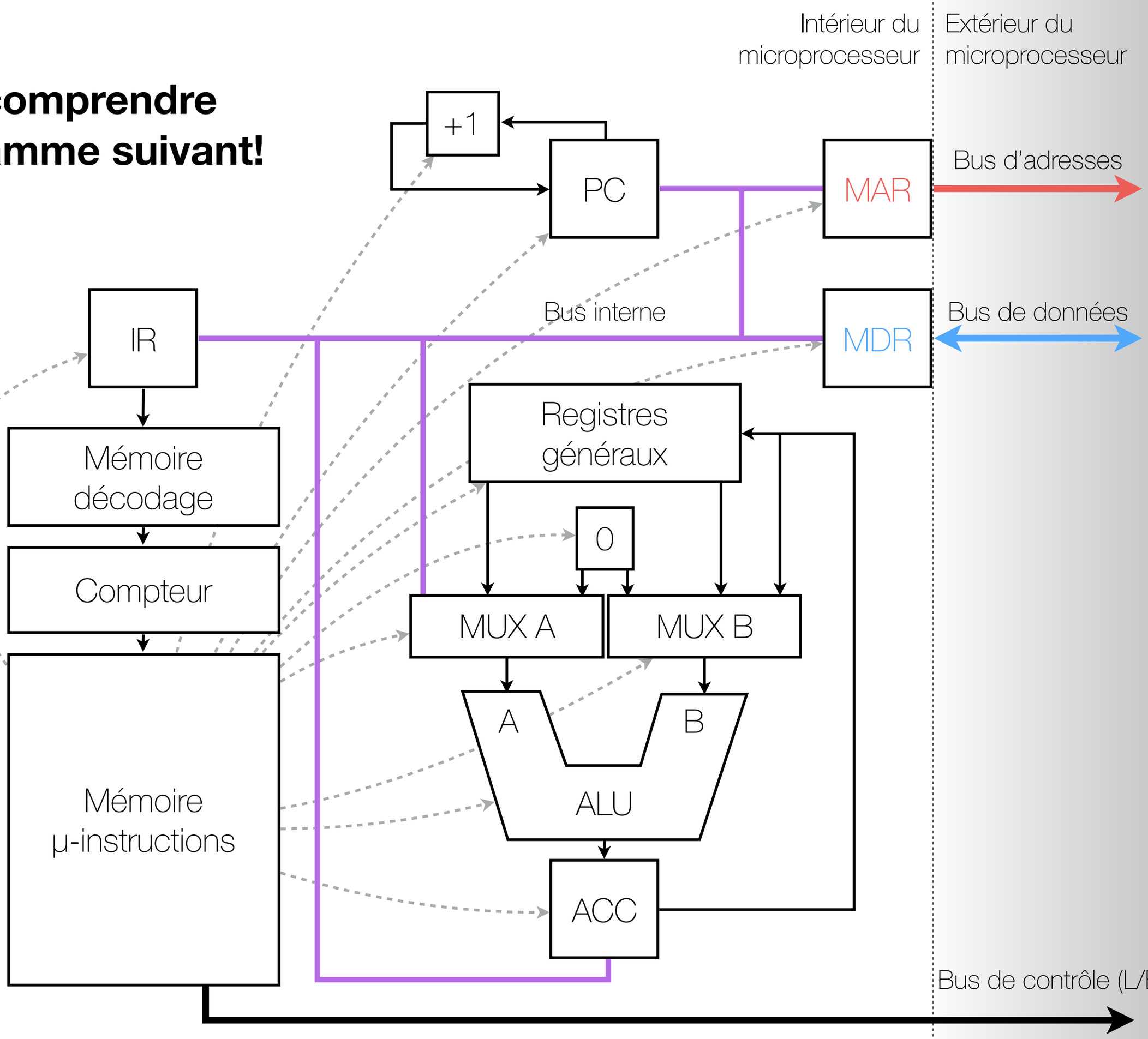
Structure interne

- Composantes principales: ALU, CCU, mémoires
- Cycle d'instructions?
  - fetch, decode, execute!
- Comment décrit-on une mémoire?
  - adresses & taille des mots
- Bus
  - Quels sont les 3 différents types?
  - À quoi sert un décodeur d'adresses?

Instructions & micro-instructions

- De quoi est composée une instruction?
  - opcode + paramètres
- Une instruction = séquence de micro-instructions
- Utilisez le simulateur de micro-instructions!

Bien comprendre le diagramme suivant!



ARM

- Connaître les éléments de base de l'architecture ARM
  - Taille des instructions?
  - Nombre de registres?
  - RISC ou CISC?
  - Structure de la mémoire et impact sur PC
    - à chaque instruction, PC = PC + 4
    - PC pointe à l'adresse de l'instruction exécutée **+ 8** (donc 2 instructions plus loin)

Assembleur ARM

- Être capable de **comprendre** des programmes simples
  - Comprendre le fonctionnement des instructions et des code de condition
- Être capable **d'écrire** des programmes (plus) simples
  - E.g. valeur absolue, "if/else", appel de fonction simple

Mnémonique	Description
`ADD Rd, Rs, Op1`	`Rd = Rs + Op1`
`ADC Rd, Rs, Op1`	`Rd = Rs + Op1 + Carry`
`AND Rd, Rs, Op1`	`Rd = Rs AND Op1`
`ASR Rd, Rs, #imm`	`Rd = Rs / 2`$^{\texttt{imm}}$
`Bcc Offset`	`PC = PC + Offset`, si `cc` est rencontré
`BLcc Offset`	Comme `B`, `LR` = Adr. de l'instr. suivante
`CMP Rs, Op1`	Change les drapeaux comme `Rs-Op1`
`LDR Rd, [Rs, Op2]`	`Rd = Mem[Rs + Op2]`
`LDR Rd, [Rs], Op2`	`Rd = Mem[Rs], Rs = Rs + Op2`
`LDR Rd, [Rs, Op2]!`	`Rs = Rs + Op2, Rd = Mem[Rs]`
`LSL Rd, Rs, #imm`	`Rd = Rs x 2`$^{\texttt{imm}}$
`MUL Rd, Rs, Op1`	`Rd = Rs x Op1`
`MVN Rd, Op1`	`Rd = !Op1` (inverse les bits)
`POP {Reg List}`	Met la liste de registres sur la pile
`PUSH {Reg List}`	Met la liste de registres sur la pile
`SBC Rd, Rs, Op1`	`Rd = Rs - Op1 - C`
`STR Rd, [Rs, Op2]`	`Mem[Rs + Op2] = Rd`
`STR Rd, [Rs], Op2`	`Mem[Rs] = Rd, Rs = Rs + Op2`
`STR Rd, [Rs, Op2]!`	`Rs = Rs + Op2, Mem[Rs] = Rd`
`SUB Rd, Rs, Op1`	`Rd = Rs - Op1`

Mnémonique	Condition	Mnémonique	Condition
`CS`	Carry Set	`CC`	Carry Clear
`EQ`	Equal (Zero Set)	`NE`	Not Equal (Zero Clear)
`VS`	Overflow Set	`VC`	Overflow Clear
`GT`	Greater Than	`LT`	Less Than
`GE`	Greater Than or Equal	`LE`	Less Than or Equal
`PL`	Plus (Positive)	`MI`	Minus (Negative)
`HI`	Higher Than	`LO`	Lower Than
`HS`	Higher or Same	`LS`	Lower or Same