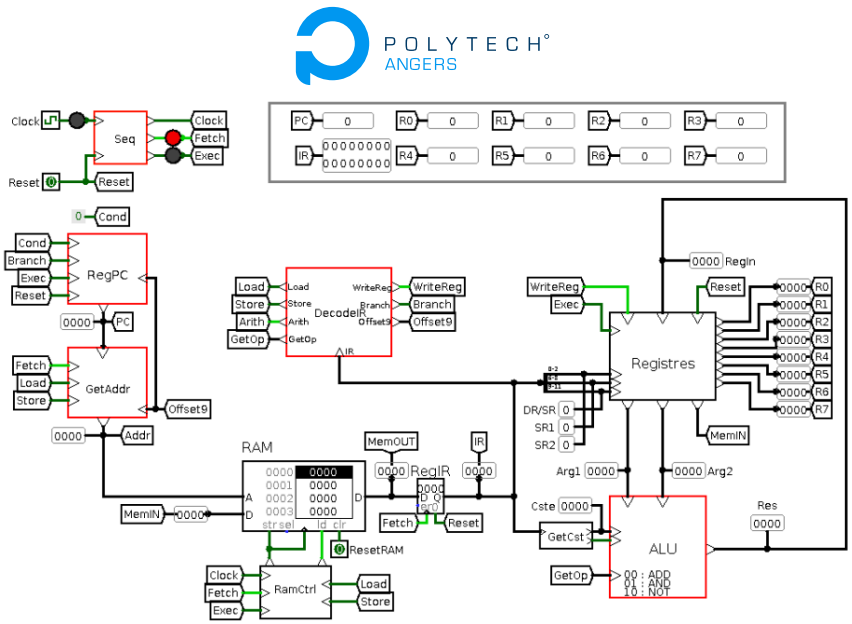
Microcontrôleur

Assembleur et architecture des ordinateurs avec Little Computer 3.

Nicolas Delanoue

Université d'Angers - Polytech Angers



Définition

Un programme informatique est un ensemble d'instructions et d'opérations destinées à être exécutées par un ordinateur.

Définition

Un programme informatique est un ensemble d'instructions et d'opérations destinées à être exécutées par un ordinateur.

Définition

Un programme source est un code écrit par un informaticien dans un langage de programmation. Il peut être compilé et/ou assemblé vers une forme binaire ou directement interprété.

Définition

Un programme informatique est un ensemble d'instructions et d'opérations destinées à être exécutées par un ordinateur.

Définition

Un programme source est un code écrit par un informaticien dans un langage de programmation. Il peut être compilé et/ou assemblé vers une forme binaire ou directement interprété.

Définition

Un programme binaire décrit les instructions à exécuter par un microprocesseur sous forme numérique. Ces instructions définissent un langage machine.

Exemple de programme source en c

```c
int a = 5;
int b = 4;
int c = a + b;
```

Exemple de programme source en c

```
int a = 5;
int b = 4;
int c = a + b;
```

Exemple de programme source en assembleur LC-3

```
ADD R1,R1,#5
ADD R2,R2,#4
ADD R3,R1,R2
```

Exemple de programme source en c

```
int a = 5;
int b = 4;
int c = a + b;
```

Exemple de programme source en assembleur LC-3

```
ADD R1,R1,#5
ADD R2,R2,#4
ADD R3,R1,R2
```

Le même programme en langage machine (binaire) LC-3

```
0001001001100101
0001010010100100
0001011001000010
```

Définition

L'assembleur est un programme qui va, pour chaque instruction du code assembleur, écrire le code binaire correspondant à cette action pour un processeur choisi.

Définition

L'assembleur est un programme qui va, pour chaque instruction du code assembleur, écrire le code binaire correspondant à cette action pour un processeur choisi.

Démonstration avec le simulateur PennSim

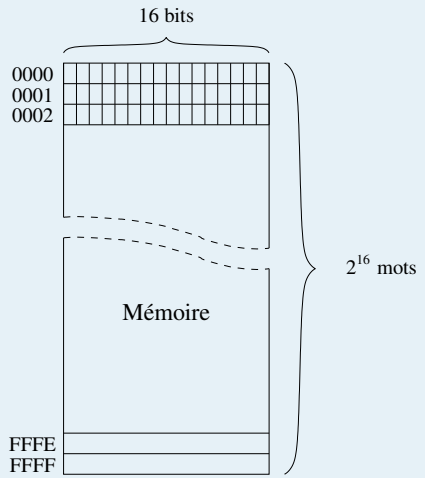
- Assemblage d'un programme, `as fichier.asm`
- Chargement d'un programme, `load fichier.obj`
- Exécution et observations (`set PC x3000`).

Définition - LC-3

Le Little Computer 3 est un ordinateur à vocation pédagogique.

Remarques

- Il n'existe pas de réalisation concrète de cette machine mais il existe des simulateurs permettant d'exécuter des programmes.
- Il est composé d'un microprocesseur (avec son jeu d'instructions),
- d'une mémoire de 2^{16} mots de 16 bits chacun,



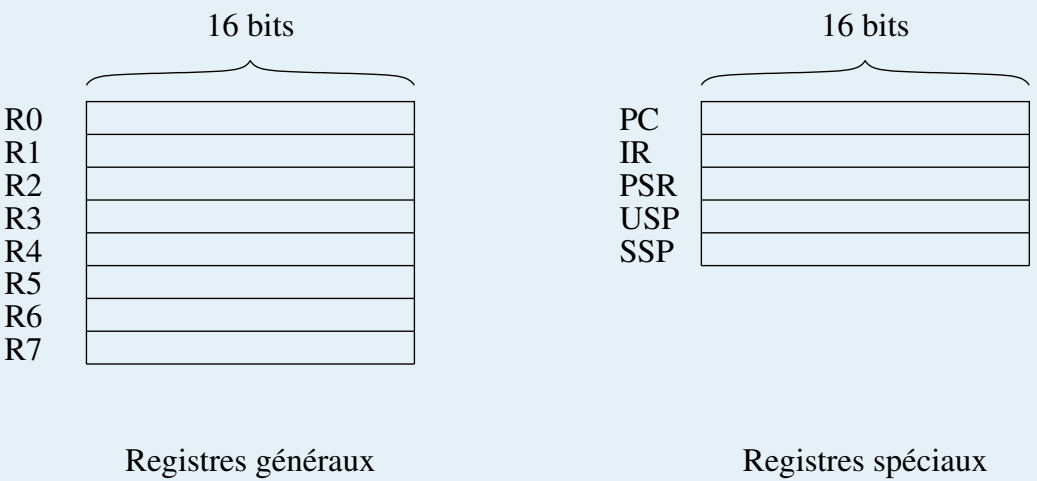
- ...

Registres

Le micro-processeur LC-3 dispose de

- 8 registres généraux de 16 bits appelés R0,. . .,R7.
- d'un registre de 16 bits nommé compteur de programme ou PC (transparent suivant),
- d'un registre d'instruction nommé IR pour Instruction Register.
- d'autres registres utiles pour les branchements . . .

Illustration



Définition

Le compteur ordinal ou pointeur d'instruction (en anglais : program counter) est le registre (souvent nommé PC) qui contient l'adresse mémoire de l'instruction prochainement exécutée.

Remarques

- Une fois l'instruction chargée, il est automatiquement incrémenté pour pointer l'instruction suivante.
- La valeur stockée dans ce registre peut être mise à jour par une instruction de branchement . . .

Indicateurs N, Z et P

Les indicateurs sont des registres de 1 bit. Les trois indicateurs N, Z et P font, en fait, partie du registre spécial PSR.

Explications

- Ils indiquent respectivement si une nouvelle valeur calculée est négative (n), nulle (z) et positive (p).
- Ces indicateurs sont utilisés par l'instruction de branchement conditionnel BR pour savoir si le branchement doit être pris ou non.

Instructions

Les instructions du LC-3 se répartissent en trois familles :

- les instructions arithmétiques et logiques (ADD, AND, NOT)
- les instructions de chargement et rangement (en lien avec la mémoire),
- les instructions de branchement (appelées aussi instructions de saut ou encore instructions de contrôle).

Instructions arithmétiques et logiques

Les instructions arithmétiques et logiques du LC-3 sont au nombre de trois :

- une instruction ADD pour l'addition,
- une instruction AND pour le et logique bit à bit,
- une instruction NOT pour la négation bit à bit.

Instructions arithmétiques et logiques

Les instructions arithmétiques et logiques du LC-3 sont au nombre de trois :

- une instruction ADD pour l'addition,
- une instruction AND pour le et logique bit à bit,
- une instruction NOT pour la négation bit à bit.

Exemples

- `ADD R3,R1,R2` additionne les valeurs contenues dans `R1` et `R2` puis stocke le résultat dans `R3`.

Instructions arithmétiques et logiques

Les instructions arithmétiques et logiques du LC-3 sont au nombre de trois :

- une instruction ADD pour l'addition,
- une instruction AND pour le et logique bit à bit,
- une instruction NOT pour la négation bit à bit.

Exemples

- `ADD R3,R1,R2` additionne les valeurs contenues dans `R1` et `R2` puis stocke le résultat dans `R3`.
- `ADD R3,R1,#5` additionne la valeur contenue dans `R1` et le nombre 5 puis stocke le résultat dans `R3`.

Instructions arithmétiques et logiques

Les instructions arithmétiques et logiques du LC-3 sont au nombre de trois :

- une instruction ADD pour l'addition,
- une instruction AND pour le et logique bit à bit,
- une instruction NOT pour la négation bit à bit.

Exemples

- `ADD R3,R1,R2` additionne les valeurs contenues dans `R1` et `R2` puis stocke le résultat dans `R3`.
- `ADD R3,R1,#5` additionne la valeur contenue dans `R1` et le nombre 5 puis stocke le résultat dans `R3`.
- `ADD R3,R1,xA` additionne la valeur contenue dans `R1` et le nombre 10 puis stocke le résultat dans `R3`.

Instructions arithmétiques et logiques

Les instructions arithmétiques et logiques du LC-3 sont au nombre de trois :

- une instruction ADD pour l'addition,
- une instruction AND pour le et logique bit à bit,
- une instruction NOT pour la négation bit à bit.

Exemples

- `ADD R3,R1,R2` additionne les valeurs contenues dans `R1` et `R2` puis stocke le résultat dans `R3`.
- `ADD R3,R1,#5` additionne la valeur contenue dans `R1` et le nombre 5 puis stocke le résultat dans `R3`.
- `ADD R3,R1,xA` additionne la valeur contenue dans `R1` et le nombre 10 puis stocke le résultat dans `R3`.
- `AND R3,R3,x0` réalise un et bit à bit entre `R3` et 0 puis stocke le résultat dans `R3`. Finalement, le registre `R3` contient 0.

Instructions arithmétiques et logiques

Les instructions arithmétiques et logiques du LC-3 sont au nombre de trois :

- une instruction ADD pour l'addition,
- une instruction AND pour le et logique bit à bit,
- une instruction NOT pour la négation bit à bit.

Exemples

- `ADD R3,R1,R2` additionne les valeurs contenues dans `R1` et `R2` puis stocke le résultat dans `R3`.
- `ADD R3,R1,#5` additionne la valeur contenue dans `R1` et le nombre 5 puis stocke le résultat dans `R3`.
- `ADD R3,R1,xA` additionne la valeur contenue dans `R1` et le nombre 10 puis stocke le résultat dans `R3`.
- `AND R3,R3,x0` réalise un et bit à bit entre `R3` et 0 puis stocke le résultat dans `R3`. Finalement, le registre `R3` contient 0.
- Comment faire un "ou logique" ?

Instructions de chargement et rangement

- LD (load) charge dans un registre la valeur stockée dans une case mémoire.
- ST (store) stocke dans une case mémoire la valeur stockée dans un registre.

Exemples

```
; partie dédiée au code
        LD R1, a
        ADD R1,R1,x1
        ST R1, b
; partie dédiée aux données
a:      .FILL x0005
b:      .FILL x0004
```

Instructions de chargement et rangement

- LD (load) charge dans un registre la valeur stockée dans une case mémoire.
- ST (store) stocke dans une case mémoire la valeur stockée dans un registre.

Exemples

```
; partie dédiée au code
        LD R1, a
        ADD R1,R1,x1
        ST R1, b
; partie dédiée aux données
a:      .FILL x0005
b:      .FILL x0004
```

Pour ne pas à retenir les adresses des cases mémoire, on peut utiliser des étiquettes (dans l'exemple précédente a et b sont des étiquettes).

Définition

L'instruction générale de branchement est l'instruction BR. Elle modifie le déroulement normal des instructions en changeant la valeur contenue par le registre PC.

Exemple

```
.orig x0000
        LD R0,a0
debut:  ADD R0,R0,0x0001
        ST R0,a0
        BR debut
a0:    .FILL 0x0000
.end
```

Différents type de branchement

- Le branchement peut être inconditionnel (BR) ou conditionnel.
- Le cas conditionnel signifie que le branchement est réellement exécuté seulement si une condition est vérifiée.

Brachement conditionnel

Des indicateurs n, z, et p sont examinés.
Si au moins un des indicateurs examinés vaut 1

- alors le branchement est pris,
- sinon, le programme continue avec l'instruction suivante.

Exemple

```
.orig x0000
        LD R0,a0
debut:  ADD R0,R0,#-1
        ST R0,a0
        BRp debut
a0:    .FILL #5
.end
```

If then else - Du c à l'assembleur [a]

a. https://www.cs.colostate.edu/~fsieker/misc/CtoLC3.html

C code	LC3 code

if (a < b) {
  // do something
}
else {
  // do something else
}
	LD R0, a ; load a
	LD R1, b ; load b
	NOT R1, R1 ; begin 2's complement of b
	ADD R1, R1, #1 ; R1 now has -b
	ADD R0, R0, R1 ; R0 = a + (-b)
	  ; condition code now set
	BRzp ELSE ; if false, skip over code
	
	  ; code to do something (the true clause)
	
	BR END_ELSE ; don't execute else code
	
	ELSE ; code for else clause here
	
	END_ELSE ; remainder of code after else

Récapitulatif “simplifiée” des instructions

Syntaxe	Action	nzp
`NOT DR,SR`	DR ← not SR	*
`ADD DR,SR1,SR2`	DR ← SR1 + SR2	*
`ADD DR,SR1,Imm5`	DR ← SR1 + val(Imm5)	*
`AND DR,SR1,SR2`	DR ← SR1 et SR2	*
`AND DR,SR1,Imm5`	DR ← SR1 et val(Imm5)	*
`LD DR,label`	DR ← mem[“label”]	*
`ST SR,label`	mem[“label”] ← SR	
`LDI DR,label`	DR ← mem[mem[“label”]]	*
`STI SR,label`	mem[mem[“label”]] ← SR	
`BR[n][z][p] label`	Si (cond) PC ← “label”	

Définition

Une directive d'assemblage permet de donner des ordres à l'assembleur.

Exemple

- `.ORIG ⟨adresse⟩` spécifie l'adresse à laquelle doit commencer le bloc d'instructions qui suit.
- `.END` termine un bloc d'instructions.
- `.FILL ⟨valeur⟩` réserve un mot de 16 bits et le remplit avec la valeur constante donnée en paramètre.
- `.STRINGZ ⟨chaine⟩` réserve un nombre de mots de 16 bits égal à la longueur de la chaîne de caractères terminée par un caractère nul et y place la chaîne.
- `.BLKW ⟨nombre⟩` réserve le nombre de mots de 16 bits passé en paramètre.

Définition

Un programme binaire décrit les instructions à exécuter par un microprocesseur sous forme numérique. Ces instructions définissent un langage machine.

Définition

Un code opération (opcode en anglais) est la partie d'une instruction en langage machine qui spécifie l'opération à effectuer.

Exemple

Instruction	**Codage**																**Code en hexa**
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	Op-code				DR			SR1			0	0	0	SR2			
ADD R2,R7,R5	0	0	0	1	0	1	0	1	1	1	0	0	0	1	0	1	0x15C5
	Op-code				DR			SR1			1	Imm5					
ADD R6,R6,-1	0	0	0	1	1	1	0	1	1	0	1	1	1	1	1	1	0x1DBF

Liste des codes opération

LC-3 Instructions

ADD	0001	DR	SR1	0	00	SR2
ADD	0001	DR	SR1	1	imm5	
AND	0101	DR	SR1	0	00	SR2
AND	0101	DR	SR1	1	imm5	
NOT	1001	DR	SR	111111		
BR	0000	n	z	p	PCoffset9	
JMP	1100	0	00	BaseR	000000	
JSR	0100	1	PCoffset11			
JSRR	0100	0	00	BaseR	000000	
RET	1100	0	00	111	000000	

LD	0010	DR	PCoffset9	
LDI	1010	DR	PCoffset9	
LDR	0110	DR	BaseR	offset6
LEA	1110	DR	PCoffset9	
ST	0011	SR	PCoffset9	
STI	1011	SR	PCoffset9	
STR	0111	SR	BaseR	offset6
TRAP	1111	0000	trapvect8	
RTI	1000	000000000000		
reserved	1101			

Définition

L'exécution d'une instruction se décompose en plusieurs phases et ce cycle de phase est appelé cycle d'instruction.

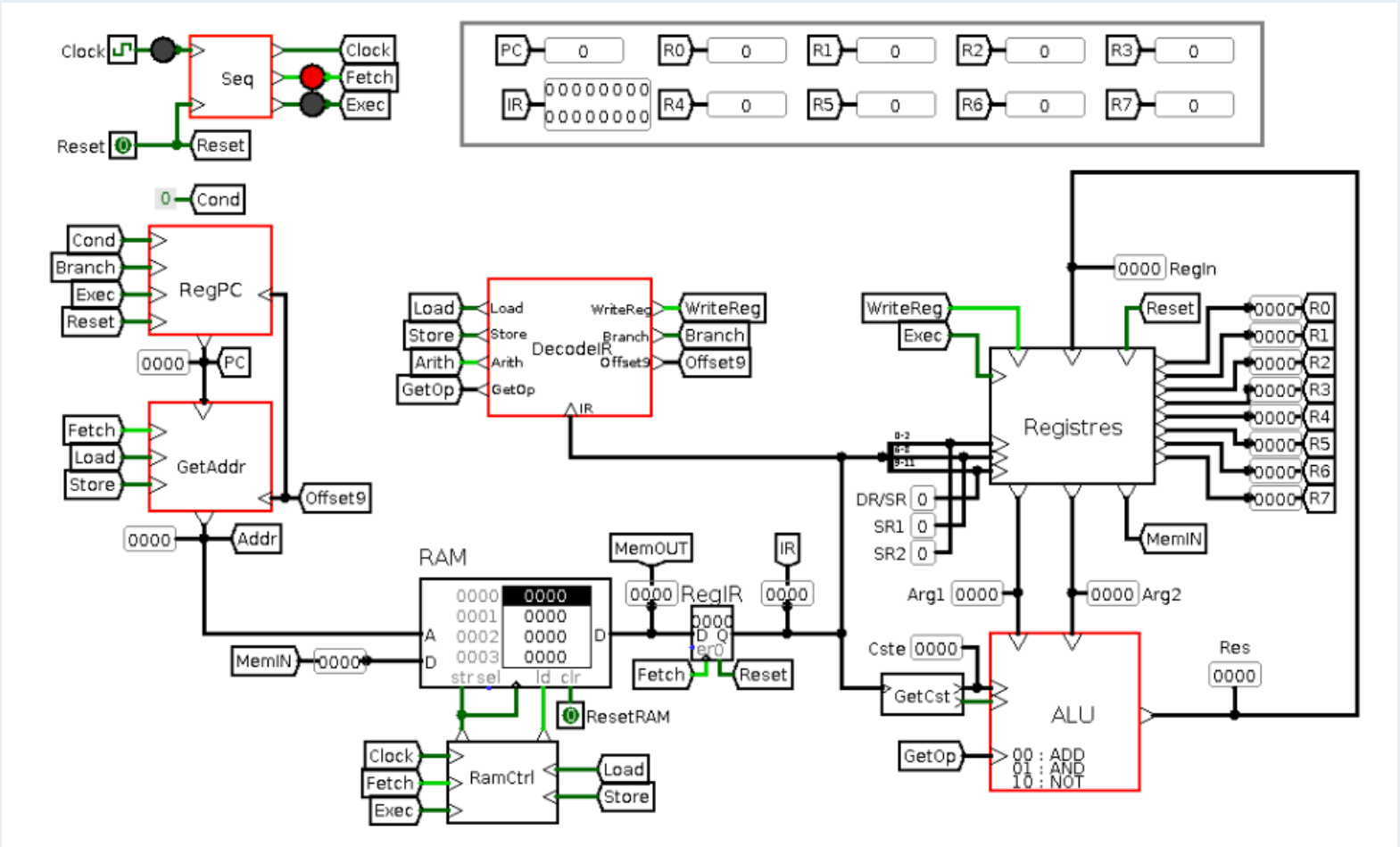
Pour chaque instruction, ces phases sont exécutées dans cet ordre :

1. FETCH : La prochaine instruction, pointée par PC, est chargée dans le registre d'instuction (IR). La valeur de PC est aussi incrémentée.
2. DECODE : L'instruction contenue dans IR est décodée pour savoir quelle partie matérielle doit être sollicitée (mémoire, ALU, ...)
3. EVALUATE ADDRESS : dans le cas où les adresses sont relatives.
4. FETCH OPERANDS : Les valeurs contenues dans les registres sont placées devant le bon circuit.
5. EXECUTE : Exécution de l'instruction.
6. STORE RESULT : Les valeurs obtenues sont stockées au bon endroit.

Exemple

```
ADD R2,R1,#5
```

Illustration avec Logisim



Conclusion

- Pipeline,
- Instructions réservées pour le système d'exploitation,
- ...