Plan

- Représentation des nombres
- Circuits logiques
- Unité Arithmétique et Logique
- Notions de temps et de mémorisation
- Contrôle et jonction des composants
- Evolution des ordinateurs – Historique
- **Un microprocesseur simple**
- Programmation d'un microprocesseur
- Système complet
- Les microprocesseurs actuels
- Exploitation de la performance des microprocesseurs

Processeurs RISC

- Architectures et jeux d'instructions complexes
  - ⇒ Temps de conception long (circuit de contrôle,…).
  - ⇒ L'évolution rapide de la technologie n'est pas pleinement exploitée.
  - ⇒ Instructions longues et complexes, latence mémoire → temps de cycle élevé.
  - ⇒ Compilateurs plus complexes.
- RISC (*Reduced Instruction Set Computer*):
  - Peu d'instructions, peu de modes d'adressage, peu de formats de données, taille fixe des instructions.
  - Opérandes dans registres uniquement pour accès rapide.
  - Découpage des instructions en étapes et pipeline.
  - ⇒ Conception simplifiée, temps de cycle bas.

Jeu d'Instructions

- Le jeu d'instructions décrit une représentation abstraite d'un processeur (ISA: *Instruction Set Architecture*).
- A un jeu d'instructions peut correspondre beaucoup d'implémentations différentes.
  - Exemple: jeu d'instructions x86 et processeurs d'Intel.
- Le jeu d'instructions doit permettre de:
  - Accéder à la mémoire.
  - Effectuer des opérations arithmétiques et logiques.
  - Contrôler le déroulement du programme.

Réalisation d'un Processeur Simple

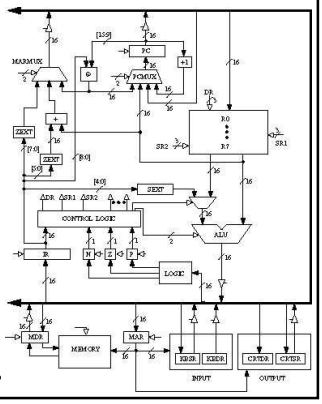
« Cahier des Charges » du LC-2

- On choisit un mot de 16-bits:
  - contraintes de taille et de rapidité du circuit
  - espace mémoire adressable
- Un processeur de type RISC (*load/store*, opérandes dans registres, instructions de taille fixe...)
- Jeu d'instructions minimal:
  - ALU: addition et le minimum d'opérations logiques; opérandes: soit exclusivement registre, soit registre et immédiat
  - Mémoire: adressage direct (~ absolu), indexé
  - Contrôle: branchement inconditionnel, appel de procédures, branchement conditionnel (tests possibles: nul, positif, négatif), appel système; adressage direct, indexé
- Questions:
  - Il faut définir le jeu d'instructions. Quel format ?
  - Il faut définir une implémentation simple du processeur. Quelle architecture ?

Organisation du Processeur

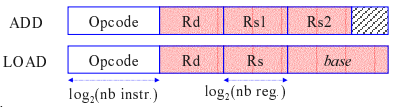
- Etapes d'exécution d'une instruction (1 étape = 1 cycle processeur):
  1. Envoi adresse instruction
  2. Chargement instruction
  3. Stockage instruction
  4. Décodage (lecture des opérandes)
  5. Calcul d'adresse
  6. Chargement d'opérandes depuis la mémoire
  7. Exécution
  8. Ecriture du résultat

ADD R5, R4, #3



LC-2

Jeu d'Instructions



- Taille des instructions = équilibre entre:
  - nombres d'instructions/opérandes,
  - nombre de registres,
  - accès mémoire (taille du champ *base*),
  - complexité du circuit de contrôle.
- Exemple: 16 bits pour mot/instructions:
  - 16 instructions → 4 bits d'opcode.
  - 12 bits pour opérandes (3 registres pour instructions ALU; exemple: ADD Rd←Rs1,Rs2)
    - ⇒ maximum 4 bits par numéro de registre (16 registres)
  - taille de la *base* (LOAD Rd←Rs1,*base*)
    - ⇒ dépend de la taille des registres (4 bits pour 16 registres, 6 bits pour 8 registres)
- Prédire les facteurs importants des prochaines architectures:
  - Ajout de nouvelles instructions
  - Augmentation du nombre de registres
  - Faciliter l'accès à de grandes zones mémoires (*base* de grande taille)
  - ...

Choix du Jeu d'Instructions

- Instructions logiques et arithmétiques:
  - Pouvoir effectuer toutes les opérations logiques.
  - Pouvoir effectuer toutes les opérations arithmétiques.
  - Deux modes d'adressage:
    - immédiat
    - direct

- `ADD Rd←Rs1,Rs2`
  - `Rd←Rs1+Rs2`
- `AND Rd←Rs1,Rs2`
  - `Rd←ET(Rs1,Rs2)`
- `ADD Rd←Rs,valeur`
  - `Rd←Rs+valeur`
- `AND Rd←Rs,valeur`
  - `Rd←ET(Rs,valeur)`
- `NOT Rd←Rs`
  - `Rd←NOT(Rs)`

Choix du Jeu d'Instructions

- Accès mémoire:
  - Chargement d'une donnée depuis la mémoire.
  - Stockage d'une donnée en mémoire.
  - Trois modes d'adressage:
    - direct
    - indexé
    - immédiat

- `LD Rd←`*`offset`*
- `ST Rs→`*`offset`*
  - Adressage direct
  - Adresse = PC[15:9],offset[8:0]
- `LDR Rd←Rs,`*`base`*
- `STR Rs2→Rs1,`*`base`*
  - Adressage indexé
  - Adresse = Rs1 + Base
- `LEA Rd←`*`offset`*
  - Adressage immédiat
  - Rd←PC[15:9],offset[8:0]

Choix du Jeu d'Instructions

- Contrôle:
  - Faire un saut.
  - Faire un saut conditionnel.
  - Faire un appel à une procédure, et retour.
  - Faire un appel à une procédure du système d'exploitation.
- Bits de condition:
  - Registres 1-bit utilisés par les instructions de branchement conditionnel.
  - Mis à jour par les instructions qui **écrivent** dans les registres.
  - **N** (Négatif), **P** (Positif), **Z** (Zéro).

- `JMP/JSR L,`*`offset`*
  - Adressage direct
  - Si L=1 R7←PC
  - PC←PC[15:9],offset[8:0]
- `JMPR/JSRR L,Rs,`*`base`*
  - Adressage indexé
  - Si L=1 R7←PC
  - PC←Rs + base
- `BR nzp `*`offset`*
  - Adressage direct
  - PC←PC[15:9],offset[8:0]
  - Test des registres condition dont le bit est à 1:
    - Si bit n=1, test du registre de condition N. valeur condition = valeur du registre N.
    - Si bits n=1 et p=1, valeur condition = OU(N,P).
- `RET`
  - PC←R7
- `TRAP `*`trapvect8`*
  - Saut à une routine système.
  - R7←PC
  - PC←@(00000000trapvec8)

Codage du Jeu d'Instructions

Opcode	Instruction
0000	**BR**
0100	**JMP/JSR**
1100	**JMPR/JSRR**
1111	**TRAP**
1101	**RET**

Contrôle

Opcode	Instruction
0100	**LD**
0110	**LDR**
1110	**LEA**
0011	**ST**
0111	**STR**

Mémoire

Opcode	Instruction
0001	**ADD**
0101	**AND**
1001	**NOT**

ALU

- ≤ 16 instructions → 4 bits d'opcode.
- Opcode:
  - Simplifier le circuit de contrôle.
  - Extensions possibles de l'*ISA*.

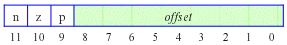
Codage du Jeu d'Instructions

- 8 registres → 3 bits.

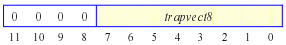
• ADD, AND:

• NOT:

• BR:

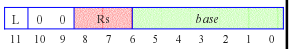


• TRAP:

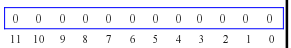


• JSR/JMP:

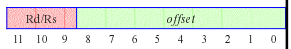
• JSRR/JMPR:



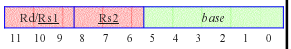
• RET:



• LD/ST:



• LDR/STR:



• LEA: