Évaluation Finale

Du Transistor au CPU

Chapitres couverts : 00 à 06 (tout le cours)

Durée : 2h00
Documents autorisés : Aucun
Barème : 40 points

Partie A : QCM (10 points)

Consignes : Une seule réponse correcte par question. +1 point par bonne réponse, 0 si faux.

Question 1 - Logique

La porte NAND est dite "universelle" car :

- [ ] A. Elle est utilisée dans tous les CPU
- [ ] B. On peut construire toutes les autres portes avec elle seule
- [ ] C. Elle fonctionne avec n'importe quelle tension
- [ ] D. Elle est la plus rapide

Question 2 - Arithmétique

En complément à deux sur 8 bits, quelle valeur représente `11111111` ?

- [ ] A. 255
- [ ] B. -1
- [ ] C. -127
- [ ] D. 0

Question 3 - ALU

Quel flag est activé quand le résultat d'une opération vaut exactement zéro ?

- [ ] A. N (Negative)
- [ ] B. Z (Zero)
- [ ] C. C (Carry)
- [ ] D. V (Overflow)

Question 4 - Mémoire

Combien de bits d'adresse faut-il pour adresser une RAM de 4096 mots ?

- [ ] A. 10 bits
- [ ] B. 12 bits
- [ ] C. 14 bits
- [ ] D. 16 bits

Question 5 - Registres

Dans l'architecture A32, quel registre spécial contient l'adresse de l'instruction en cours ?

- [ ] A. R0
- [ ] B. SP (R13)
- [ ] C. LR (R14)
- [ ] D. PC (R15)

Question 6 - ISA

L'architecture A32 est de type :

- [ ] A. CISC avec accès mémoire dans les opérations ALU
- [ ] B. RISC Load/Store avec opérations ALU sur registres uniquement
- [ ] C. Stack machine
- [ ] D. Accumulator machine

Question 7 - CPU

Dans le cycle d'exécution, quelle phase vient immédiatement après Decode ?

- [ ] A. Fetch
- [ ] B. Execute
- [ ] C. Memory
- [ ] D. Writeback

Question 8 - Contrôle

Le signal `reg_write = 1` signifie :

- [ ] A. On lit un registre
- [ ] B. On écrit dans un registre
- [ ] C. On accède à la mémoire
- [ ] D. On effectue un branchement

Question 9 - Assembleur

Pourquoi l'assembleur fait-il deux passes sur le code source ?

- [ ] A. Pour optimiser le code
- [ ] B. Pour résoudre les références vers l'avant (labels)
- [ ] C. Pour vérifier la syntaxe deux fois
- [ ] D. Pour compresser le binaire

Question 10 - Pipeline

Un data hazard se produit quand :

- [ ] A. Deux instructions utilisent la même porte logique
- [ ] B. Une instruction lit un registre qu'une instruction précédente n'a pas encore écrit
- [ ] C. Le PC est modifié de manière inattendue
- [ ] D. La mémoire est pleine

Partie B : Logique et Arithmétique (8 points)

Exercice 1 : Construction avec NAND (3 points)

Montrez comment construire un **XOR** en utilisant uniquement des portes NAND.

1. Donnez l'expression booléenne de XOR(A, B)
2. Décomposez-la en opérations réalisables avec NAND
3. Dessinez le circuit (ou décrivez-le textuellement)

Exercice 2 : Addition et Flags (3 points)

Effectuez l'addition suivante en binaire sur 8 bits (complément à deux) :

```
  01100100  (100 en décimal)
+ 01011010  (90 en décimal)
----------
```

a) Quel est le résultat en binaire et en décimal ?

b) Quels flags (N, Z, C, V) sont activés ? Justifiez pour chaque flag.

Exercice 3 : Représentation des nombres (2 points)

a) Quelle est la plage de valeurs représentables en complément à deux sur 8 bits ?

b) Pourquoi le complément à deux est-il préféré au signe-magnitude ?

Partie C : Mémoire et Architecture (10 points)

Exercice 4 : Conception mémoire (3 points)

On veut concevoir une RAM de 1024 mots de 32 bits.

a) Combien de bits d'adresse sont nécessaires ?

b) Combien de DFF sont nécessaires au total ?

c) Si le bus de données fait 32 bits, combien de lignes le bus d'adresse doit-il avoir ?

Exercice 5 : Décodage d'instruction (4 points)

Décodez cette instruction A32 :

```
1110 001 0100 1 0011 0010 0000 0000 1010
```

Remplissez le tableau :

Champ	Bits	Valeur binaire	Signification
cond	31-28		
class	27-25		
op	24-21		
S	20		
Rn	19-16		
Rd	15-12		

Exercice 6 : Trace d'exécution (3 points)

Tracez l'exécution de ce programme :

```
    MOV R0, #10
    MOV R1, #0
loop:
    ADD R1, R1, R0
    SUB R0, R0, #1
    CMP R0, #0
    B.NE loop
    HALT
```

a) Combien de fois la boucle s'exécute-t-elle ?

b) Que valent R0 et R1 à la fin ?

c) Que calcule ce programme ?

Partie D : CPU et Assembleur (12 points)

Exercice 7 : Chemin des données (4 points)

Pour l'instruction `LDR R1, [R2, #8]` :

a) Décrivez le chemin des données à travers le CPU (quels composants sont utilisés, dans quel ordre).

b) Quels signaux de contrôle sont actifs ? Donnez leur valeur :

- reg_write
- mem_read
- mem_write
- alu_src
- wb_src

Exercice 8 : Signaux de contrôle (3 points)

Complétez la table des signaux pour ces instructions :

Instruction	reg_write	mem_read	mem_write	alu_src	branch
ADD R1, R2, R3					
STR R0, [R1]					
B.EQ label					

Exercice 9 : Encodage assembleur (3 points)

Encodez manuellement l'instruction `SUB R4, R5, #15` en binaire 32 bits.

Format : `[cond:4][class:3][op:4][S:1][Rn:4][Rd:4][imm12:12]`

Donnez :

- La valeur de chaque champ
- Le résultat final en hexadécimal

Exercice 10 : Programme assembleur (2 points)

Écrivez un programme assembleur A32 qui calcule le maximum de deux nombres.

- R0 contient le premier nombre
- R1 contient le deuxième nombre
- Le résultat doit être dans R2

Barème Détaillé

Partie	Exercice	Points
A	QCM (10 × 1pt)	10
B	Ex1 XOR avec NAND	3
B	Ex2 Addition et Flags	3
B	Ex3 Représentation	2
C	Ex4 Conception mémoire	3
C	Ex5 Décodage instruction	4
C	Ex6 Trace d'exécution	3
D	Ex7 Chemin des données	4
D	Ex8 Signaux de contrôle	3

Notes pour la Correction

Partie A - Réponses QCM

1. **B** (universalité)
2. **B** (-1 en complément à deux)
3. **B** (Z = Zero)
4. **B** (12 bits, car $2^{12} = 4096$)
5. **D** (PC = R15)
6. **B** (RISC Load/Store)
7. **B** (Execute)
8. **B** (écriture registre)
9. **B** (références vers l'avant)
10. **B** (lecture avant écriture)

Partie B - Solutions

Ex1 - XOR avec NAND :

```
XOR(A,B) = (A AND NOT B) OR (NOT A AND B)
         = NAND(NAND(A, NAND(A,B)), NAND(B, NAND(A,B)))
```

Circuit : 4 portes NAND

Ex2 - Addition :

```
  01100100 (100)
+ 01011010 (90)
= 10111110 (190... mais en signé = -66)
```

- N = 1 (bit de poids fort = 1)
- Z = 0 (résultat ≠ 0)
- C = 0 (pas de retenue sortante sur 8 bits)
- V = 1 (overflow : deux positifs donnent un négatif)

Ex3 :

- a) [-128, +127]
- b) Le complément à deux permet d'utiliser le même additionneur pour les nombres signés et non-signés, et évite le double zéro (+0, -0).

Partie C - Solutions

Ex4 :

- a) 10 bits ($2^{10} = 1024$)
- b) 1024 × 32 = 32768 DFF
- c) 10 lignes

Ex5 :

Champ	Valeur	Signification
cond	1110	AL (toujours)
class	001	ALU immédiat
op	0100	ADD
S	1	Mettre à jour flags
Rn	0011	R3
Rd	0010	R2
imm12	10	#10

Instruction : `ADDS R2, R3, #10`

Ex6 :

- a) 10 fois (R0 va de 10 à 1)
- b) R0 = 0, R1 = 55
- c) Somme des entiers de 1 à 10 (55 = 10+9+8+...+1)

Partie D - Solutions

Ex7 :

a) Chemin :

1. RegFile lit R2 → Data_A
2. Extend extrait 8 → operand2
3. ALU calcule R2 + 8 → adresse
4. Memory lit MEM[adresse] → donnée
5. Donnée → Rd (R1)

b) Signaux :

- reg_write = 1
- mem_read = 1
- mem_write = 0

Ex8 :

Instruction	reg_write	mem_read	mem_write	alu_src	branch
ADD R1, R2, R3	1	0	0	0	0
STR R0, [R1]	0	0	1	1	0
B.EQ label	0	0	0	X	1

Ex9 :

- cond = 1110 (AL)
- class = 001 (imm)
- op = 0010 (SUB)
- S = 0
- Rn = 0101 (R5)
- Rd = 0100 (R4)
- imm12 = 000000001111 (15)

Binaire : `1110 001 0010 0 0101 0100 0000 0000 1111`

Hex : **0xE245400F**

Ex10 :

```
    CMP R0, R1
    B.GE r0_max
    MOV R2, R1      ; R1 est plus grand
    B done
r0_max:
    MOV R2, R0      ; R0 est plus grand ou égal
done:
    HALT
```

Fin de l'évaluation