NOM, Prénom :

DS Architecture des Ordinateurs

Correction en rouge
Explications en vert

30/01/2019

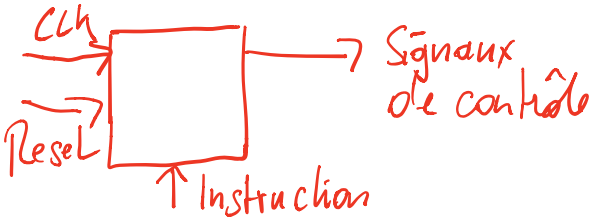
Durée 1h30.
Répondez sur le sujet.
REMPLISSEZ VOTRE NOM TOUT DE SUITE.
Tous documents autorisés, mais en papier seulement.
Crayon à papier accepté, de préférences aux ratures et surcharges.
Les questions de cours sont de difficulté variable et dans le désordre.
Les cadres donnent une idée de la taille des réponses attendues.
Si vous trouvez une question ambiguë, écrivez pourquoi et quelles seraient les réponses possibles.

1 Questions de cours

Dans les questions QCM, entourez soit V pour vrai, soit F pour faux. Si la proposition ne veut rien dire, entourez F. Il y a une pénalité pour les mauvaises réponses.

Pour toutes les questions qui concernent le MSP430, on rappelle qu'il s'agit d'un processeur 16 bits, que la mémoire est adressée par octets, que la pile est descendante avec pointeur sur case pleine, etc.

Q1. Dessinez ici un automate de contrôle d'une machine de von Neumann typique, vu de l'extérieur (sans les états ou transitions). On s'intéresse uniquement aux entrées et sorties du boîtier :



Selon l'architecture, il peut y avoir des entrées supplémentaires

Q2. ISA :

V F a Les ISA ARM et x86 sont compatibles, c.à.d. un logiciel ARM peut s'exécuter sur un CPU Intel
V F b Une instruction C se traduit toujours en une instruction assembleur
V F c Le CPU exécute les instructions assembleurs encodées en ASCII ou UTF8, selon l'architecture
V F d L'ISA ARM 32 bit (vue en cours) a des instructions de tailles identiques.

Q3. ISA :

V F a Un branchement peut modifier le PC
V F b Les instructions arithmétiques-logiques ARM peuvent avoir des sources en mémoire
V F c Toute valeur immédiate de 32 bit peut être encodée dans une instruction de 32 bit
V F d Un processeur 32 bit est capable d'exécuter des opérations arithmétiques sur des données 32 bit

Q4. Soit un processeur 32 bit encodant des sources constantes (valeurs immédiates etc.) en 12 bit avec une extension signée à 32 bit. Spécifiez les valeurs qui peuvent être encodées par cette architecture (cochez les réponses correctes) :

Voir le barrel shifter vu en cours

- [x] V [ ] F [a] 0x00000012 peut être encodé
- [ ] V [x] F [b] 0x00000fff peut être encodé
- [x] V [ ] F [c] 0xffffffff peut être encodé
- [ ] V [x] F [d] 0xfffff000 peut être encodé

Q5. On considère deux processeurs pipelinés différents :
CPU A est cadencé à 5Ghz avec un pipeline de 20 cycles.
CPU B est cadencé à 3Ghz avec un pipeline de 10 cycles.
En **ignorant** les conflits (bulles de pipeline), complétez le texte suivant :

Une instruction par cycle !

CPU A peut exécuter [5] milliards d'instructions par seconde.
CPU B peut exécuter [3] milliards d'instructions par seconde.

Cocher la réponse correcte:

- [x] [a] CPU A est plus performant
- [ ] [b] CPU B est plus performant

Q6. Raccourcir le pipeline d'un processeur le fera exécuter une instruction en moins de cycles d'horloge. En revanche, donnez une raison qui pourrait justifier le choix d'un pipeline plutôt long ?

Cela peut permettre d'augmenter la fréquence
Si on peut limiter les conflits,
cela augmentera le nombre d'instructions exécutées par cloch

Q7. Mettez une croix dans une cellule de la matrice suivante si la fonctionnalité de la ligne est réalisée (se trouve) dans la partie d'un processeur indiquée par la colonne :

	Chemins des données	Automate (FSM)
Calcul des drapeaux (N, Z, C, ...)	X	
Calcul de l'état suivant l'état *Instruction Fetch*		X
Registre stockant l'instruction	X	
ALU	X	

Q8. On considère un certain processeur qui adresse sa mémoire par octet (comme le pentium et le MSP430). Ce processeur dispose d'une instruction PUSH codées en 2 octets. Les cases de pile font 4 octets, la pile descend en mémoire, c'est à dire qu'empiler diminue le pointeur de pile. L'encodage des entiers est *little-endian*, c.à.d que les octets de poids faibles sont encodés aux adresses plus petites. Le pointeur de pile pointe vers la prochaine case vide de la pile.
Dans le tableau ci-dessous remplissez la ligne spécifiant les valeurs **après** l'exécution de l'instruction PUSH R3. Laissez vides les cases inchangées.

	Registres (16 bits)					Cases mémoires (octets)									
	PC	SP	R3	R5	R6	27	28	29	2a	2b	2c	2d	2e	2f	30
avant PUSH R3	0x0c	0x2c	6	d	e	0	d	e	a	d	b	e	e	f	0
après PUSH R3	0e	28									6	0	0	0	

Q9. Soit un ordinateur doté d'un cache 4-way associative de 64Ko organisé en lignes de 8 octets. Les adresses mémoires sont de taille 32 bit. Complétez le texte suivant :

Chaque case/adresse de la mémoire vive peut être placée dans [4] endroits différents dans le cache.

Le champs *offset* occupe [3] bits. Le champs *index* occupe [11] bits. Le champs *tag* occupe [18] bits.

Q10. Soit le code suivant remplissant un tableau T :

```
#define XSIZE 512
#define YSIZE 512
int T[XSIZE*YSIZE];
SomeFunction(T); // Appel à une fonction qui remplit le tableau T de valeurs ..
```

Lequel des deux fragments de code est plus efficace sur un ordinateur doté d'un cache ?

Code A :	Code B :

`int sum=0;`
`for (int y=0; y<YSIZE; ++y)`
`  for (int x=0; x<XSIZE; ++x)`
`  sum += T[y*XSIZE+x];`
	`int sum=0;`
	`for (int y=0; y<YSIZE; ++y)`
	`  for (int x=0; x<XSIZE; ++x)`
	`  sum += T[x*YSIZE+y];`

Cocher la réponse correcte:

- [a] Code A est plus performant
- [b] Code B est plus performant

Q11. Un MSP430 exécute ce code, qui prétend de réaliser quelque chose de bien utile.

Remarque : le MSP 430 accède à la mémoire en mode **little-endian**, c'est-à-dire que les valeurs 16bit sont stockées tel que l'octet de poids faible se trouve à l'adresse faible.

```
            mov #0x0005, R4
            mov #0x0a02, R3
            mov #0, R7
loop:
            mov @R3, R6
            cmp #0000, R4
            jz fin
            add R6, R7
            sub #1, R4
            add #2, R3
            jmp #loop
fin:
```

Ci-dessous vous trouvez le contenu d'un extrait de la mémoire avant l'exécution du programme.

```
0x0a00 05
0x0a01 00
0x0a02 01
0x0a03 00
0x0a04 01
0x0a05 00
0x0a06 14
0x0a07 0a
0x0a08 02
0x0a09 00
0x0a0a 00
0x0a0b 02
0x0a0c ff
0x0a0d ff
0x0a0e 2f
0x0a0f 3a
```

Donnez dans le tableau suivant les différentes valeurs de R7 obtenues à chaque itération de la boucle, c'est à dire au moment où PC = loop. Remarques : certaines cases à la fin du tableau peuvent être vides.

0	1	2	0A 16	0A 18	0c 18				

Quelles sont les valeurs de R3, R4, R6 et de R7 à la fin du programme ?

R3 : 0A0C R4 : 0 R6 : FFF R7 : 0c 18

Expliquez en une seule phrase l'objectif du programme :

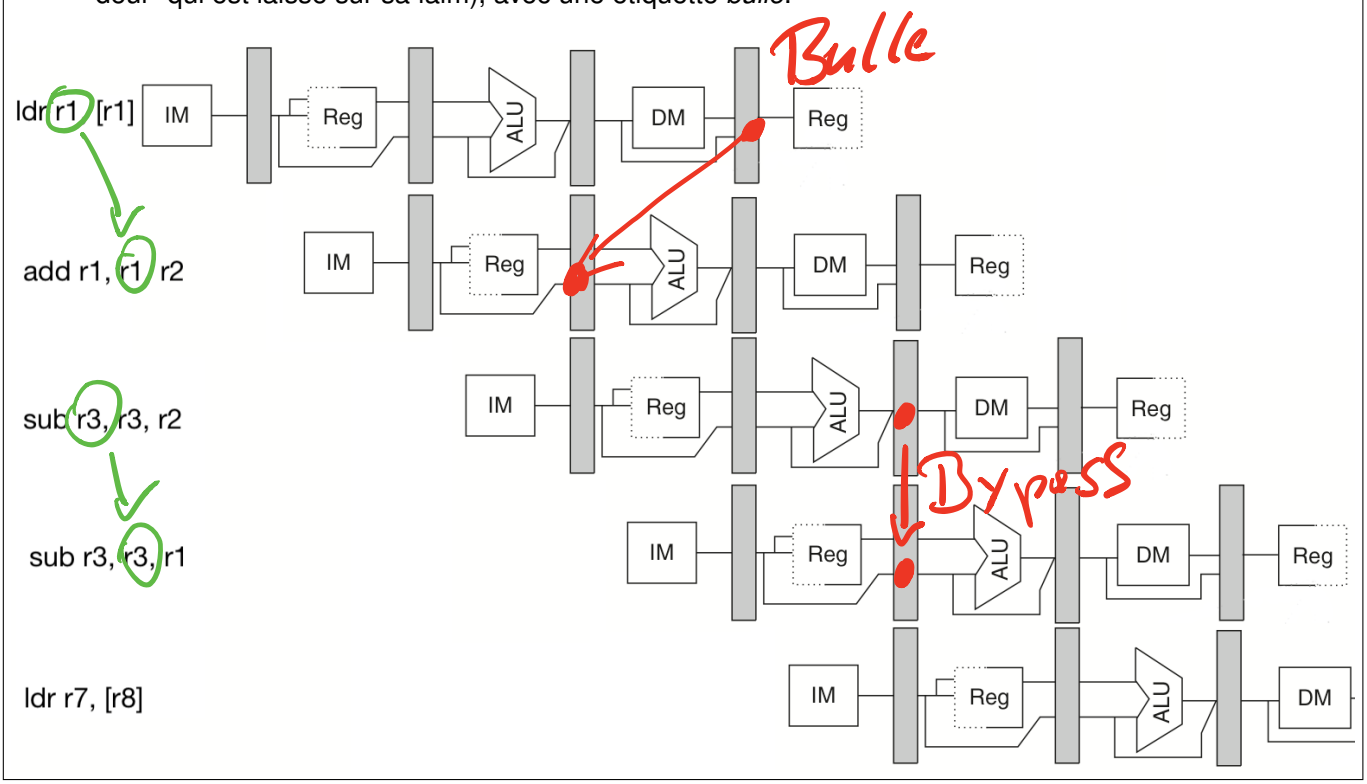
Calcul de la somme de 5 valeurs du tableau à l'adresse 0a02 (R3)

Q12. Le code suivant est exécuté sur le CPU RISC pipeliné vu en cours (rappel : la syntaxe assembleur est différente de la syntaxe MSP 430 ; le registre de destination est la première opérande) :

```
ldr r1, [r1]
add r1, r1, r2
sub r3, r3, r2
sub r3, r3, r1
ldr r7, [r8]
```

Sur le diagramme ci-dessous, indiquez

- les endroits (éventuels) nécessitant un *by-pass* (transmission directe des valeurs) pour éviter une bulle. Dessinez une flèche de l'endroit/registre source vers le registre destination, avec une étiquette *by-pass*.
- les endroits (éventuels) créant un conflit donnant lieu à une bulle, c'est à dire où le *by-pass* n'est pas possible. Marquez une flèche de l'endroit/registre source vers le registre destination (registre "demandeur" qui est laissé sur sa faim), avec une étiquette *bulle*.



2 Mode d'adressage indexé pour la Micromachine

On reprendra la micromachine comme elle a été conçue en TP, sans l'extension gérant les interruptions. Dans cet exercice, on dotera la micromachine d'un mode d'adresse « indexé »permettant d'implémenter les tableaux de manière plus efficace. On ajoutera un registre nommé « *X* », dont le contenu pourra servir comme offset supplémentaire à une adresse, portant le nombre de registres programmables de la micromachine à 3 : A, B, X.

Nous ajouterons également les instructions suivantes au jeu d'instructions :

codeop	mnémonique	remarques
0111	`*(A+X) -> B`	Lire de l'adresse A+X et stocker la valeur dans B
1011	`A->X`	copier le contenu de A vers X
1100	`++X?`	Incrémenter le contenu de X, mettre à jour les drapeaux Z et N
1010	`--X?`	Décrémenter le contenu de X, mettre à jour les drapeaux Z et N

Pour les instructions `++X?` et `--X?`, le drapeau *Z* sera activé si le nouveau contenu de X est égal à zéro, et le drapeau *N* sera activé si le nouveau contenu est négatif.
Dans la suite de cette section, nous implémenterons cette fonctionnalité.

Q13. Une micromachine modifiée exécute ce code « à trous » effectuant une recherche de la valeur « 0x42 »dans un tableau de longueur 5 stocké à l'adresse 0x10. Le résultat sera stocké dans la case 0x20.

Complétez ce code en spécifiant les deux instructions manquantes (indiquées par les cadres) :

```
            0xfb -> A          // 0xfb = -5 (valeur signée)
            A -> X
            0x15 -> A          // pointer (0x15) = begin array (0x10) + initial index(-5)
loop :
            *(A+X) -> B
            B - 0x42 ?
            JR found42 IFZ
            ++X?
            JR notfound IFZ
            JA loop

found42 :   0x01 -> A
            A -> *0x20
            JA fin

notfound :  0x00 -> A
            A -> *0x20

fin :
```

Soit le contenu suivant de la RAM :

```
0x00: 00 ef 83 37 43 79 f7 e8 1e b7 37 7d 86 ff fe 01
0x10: 7f 67 87 42 84 21 9d 83 58 f3 95 2c 31 a9 f0 02
(...)
```

Quelles sont les valeurs de A, B et X à la fin du programme ?

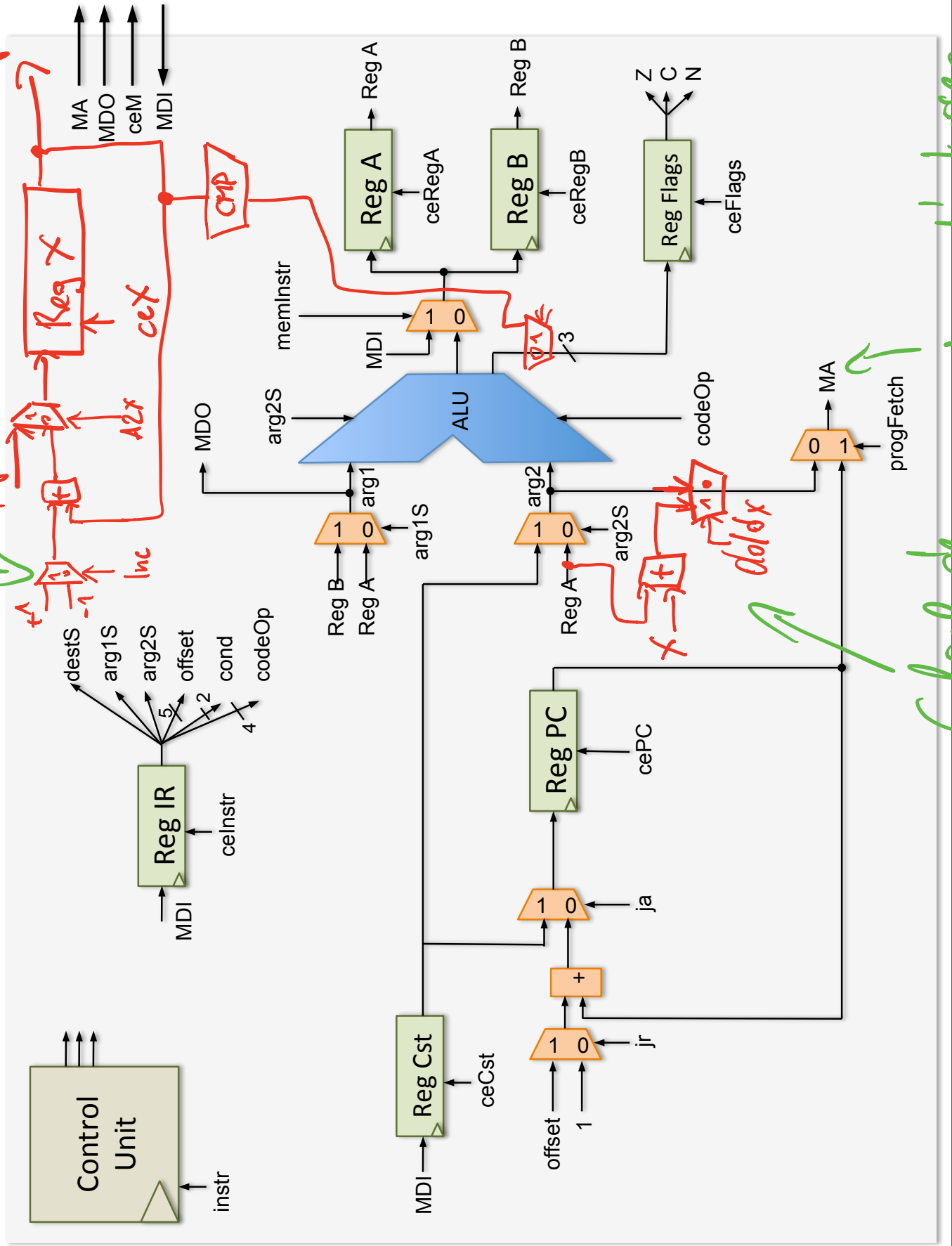
A : 1 B : 0x42 X : 0xFe

Q14. Quelles sont les tailles respectives des nouvelles instructions (en octets) ?

mnémonique	taille en octets
`*(A+X) -> B`	1
`A->X`	1
`++X?`	1
`--X?`	1

(pas de constante !)

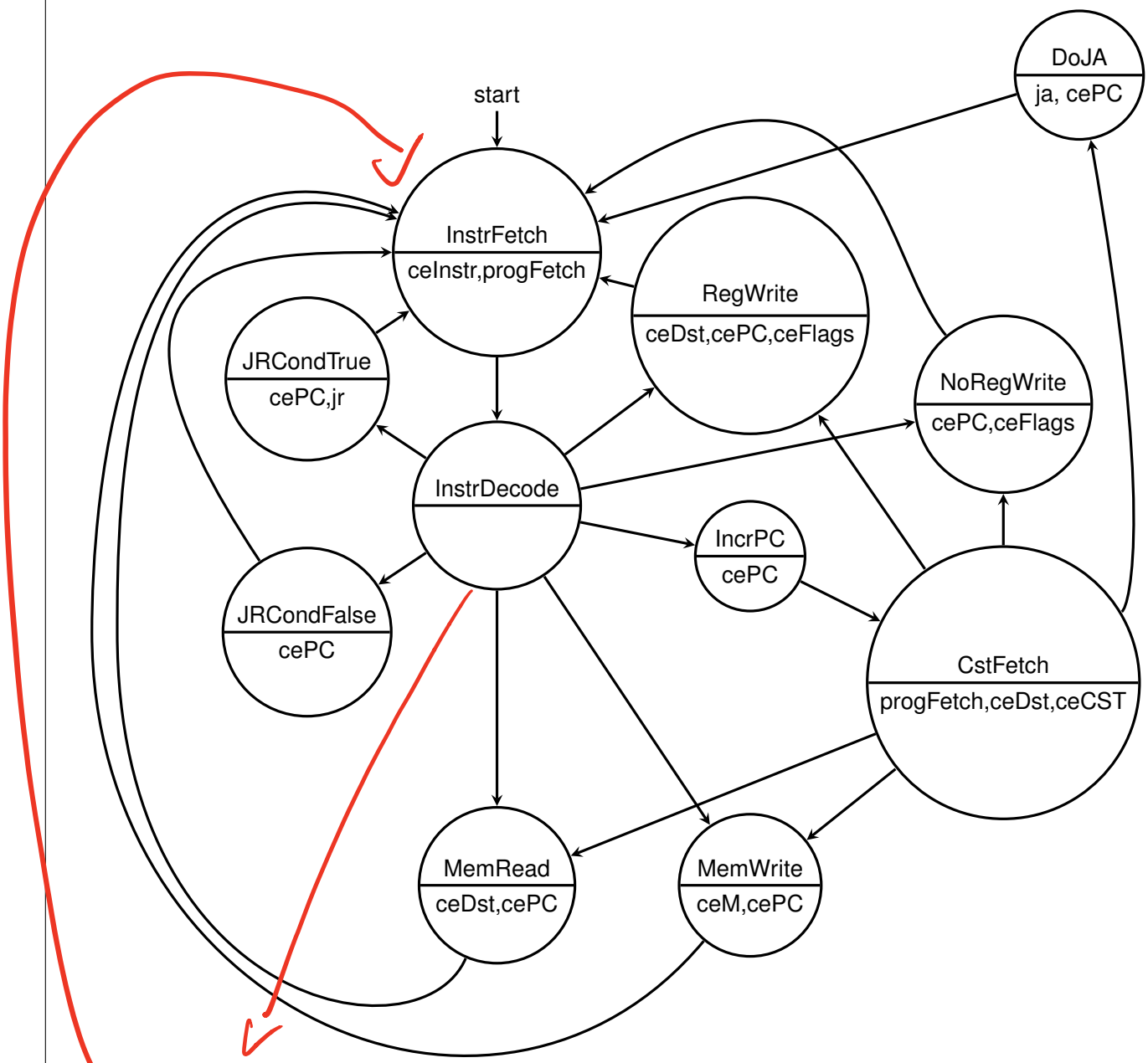
Q15. Dans la figure ci-dessous vous retrouvez le chemin des données de la micromachine, sans la gestion des interruptions. Effectuez les modifications nécessaires de telle manière à ce que le nouveau mode d'adressage soit possible (en supposant un automate adapté) : ajout de(s) composante(s), connections. Remarque : la solution est plus simple si vous n'utilisez **pas** l'ALU pour indexer. Prévoyez un additionneur dédié, comme nous l'avons fait pour incrémenter le PC.



Q16. Dans la figure suivante vous retrouvez l'automate de la micromachine (les tailles des états n'ont pas de signification, les signaux d'entrée n'ont pas été spécifiés).

Modifiez l'automate pour rendre possible l'instruction `*(A+X) -> B` en utilisant les modifications que vous avez faites au chemin des données. Vous pouvez ignorer les 3 autres nouvelles instructions.

Rappel : le signal `ceDst` sert à enregistrer le registre de destination A ou B.
Remarque : il convient de réutiliser les états existants quand cela est possible.



ceDst, cePC
doldx

commande indexation

Q17. On souhaite ajouter la possibilité de fournir l'adresse du tableau comme une constante de 8 bit, ce qui donnera deux variantes du mode d'adresse indexé :

Variante 1 :	Lire de l'adresse A+X et stocker la valeur dans B
Variante 2 :	Lire de l'adresse cst+X et stocker la valeur dans B

Les deux variantes utiliseront le même code opératoire. Entourez dans **le premier tableau** ci-dessous l'endroit le plus logique encodant la différence entre les deux variantes :

bit	7	6	5	4	3	2	1	0
instruction autres que JR	0	codeop				arg2S	arg1S	destS
saut relatif conditionnel	1	cond			offset signé sur 5 bits			

Notation	encodé par	valeurs possibles
`dest`	destS=instr[0]	`A` si destS=0, `B` si destS=1
`arg1`	arg1S=instr[1]	`A` si arg1S=0, `B` si arg1S=1
`arg2`	arg2S=instr[2]	`A` si arg2S=0, constante 8-bit si rg2S=1
`offset`	instr[5 :0]	offset signé sur 5 bits

Proposez une mnémonique pour cette instruction (= instruction en syntaxe assembleur) unifiant les deux variantes :

*(A+X) → arg2

Donnez en une phrase l'inconvénient de spécifier l'adresse comme une constante :

Instruction plus longue, cycle plus long