Licence Informatique 3
X31I050

Université de Nantes
2021/2022

Projet de travaux pratiques
Réalisation du processeur *Fifth*

Le projet est à effectuer en binôme (ou exceptionnellement, en monôme). Le fichier *Logisim* commenté ainsi que le rapport de projet, *impérativement* au format PDF, sont à déposer sur Madoc sous la forme d'une seule archive dont le nom sera composé des noms de chacun des étudiants du binôme — ex. `tartempion-dupont.tar.gz`. La décompression de cette archive devra créer un répertoire `tartempion-dupont/` contenant les fichiers. On re-précisera en commentaire dans le fichier *Logisim* les noms et prénoms de chacun des étudiants.

Un projet rendu après la **date limite fixée au dimanche 12 décembre 2021, 23h55**, sera affecté d'un malus augmentant exponentiellement avec le nombre de jours de retard.

On souhaite réaliser le processeur *Fifth* dans *Logisim* (voir figure 1). Le *Fifth* est un processeur 16 bits composé de deux piles :

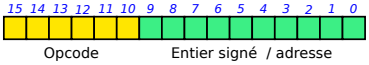
- Une pile de données sur laquelle sont empilés les arguments des instructions ainsi que les paramètres des fonctions ;
- Une pile d'appels sur laquelle sont empilées les adresses de retour des fonctions.

Chaque pile est constituée de 16 registres $r_0, \ldots, r_{15}$ de 16 bits. Le fond d'une pile correspond au registre r_{15} ; la pile croît vers le registre r_0. Chaque pile dispose d'un registre SP de 4 bits pointant sur la dernière case utilisée.

Table 1 – Jeu d'instructions du *Fifth*

Instruction	**opcode**	**Exécution**
`add`	`000000`	stack[SP+1] ← stack[SP+1] + stack[SP] ; pop
`sub`	`000001`	stack[SP+1] ← stack[SP+1] − stack[SP] ; pop
`neg`	`000010`	stack[SP] ← −stack[SP]
`mul`	`000011`	stack[SP+1] ← stack[SP+1] × stack[SP] ; pop
`div`	`000100`	stack[SP+1] ← stack[SP+1] ÷ stack[SP] ; pop
`rem`	`000101`	stack[SP+1] ← stack[SP+1] % stack[SP] ; pop
`not`	`000110`	stack[SP] ← ¬stack[SP]
`and`	`000111`	stack[SP+1] ← stack[SP+1] ∧ stack[SP] ; pop
`or`	`001000`	stack[SP+1] ← stack[SP+1] ∨ stack[SP] ; pop
`shl`	`001001`	stack[SP+1] ← stack[SP+1] << stack[SP] ; pop
`sra`	`001010`	stack[SP+1] ← stack[SP+1] >> stack[SP] ; pop
`lt`	`001011`	stack[SP+1] ← (stack[SP+1] < stack[SP]) ; pop
`eq`	`001100`	stack[SP+1] ← (stack[SP+1] == stack[SP]) ; pop
`gt`	`001101`	stack[SP+1] ← (stack[SP+1] > stack[SP]) ; pop
`jmp` *adr*	`001110`	PC ← adr
`jmpt` *adr*	`001111`	if stack[SP] : PC ← adr ; pop
`jmpf` *adr*	`010000`	if !stack[SP] : PC ← adr ; pop
`push` *val*	`010001`	stack[SP-1] ← val ; --SP
`push` *[adr]*	`010010`	stack[SP-1] ← MEM[adr] ; --SP
`pop`	`010011`	SP ← SP + 1
`pop` *[adr]*	`010100`	MEM[adr] ← stack[SP] ; pop
`call` *adr*	`010101`	stackC[SPc − 1] ← PC + 1 ; --SPc ; `jmp` *adr*
`ret`	`010110`	PC ← stackC[SPc] ; ++SPc
`dup`	`010111`	push stack[SP]
`halt`	`011000`	–

Le jeu d'instructions du *Fifth* est décrit dans le tableau 1. Toutes les instructions sont codées sur 16 bits, avec un *opcode* de 6 bits et une partie sur 10 bits pouvant recevoir un entier signé en complément à 2 ou une adresse absolue.



Lors de son utilisation, la valeur sur 10 bits doit obligatoirement être étendue à 16 bits car toutes les unités fonctionnelles du *Fifth* travaillent avec des quantités de 16 bits. Le registre PC contient l'adresse de l'instruction courante dans la mémoire ROM de programme.
La table 2 présente un exemple de programme et sa traduction en langage d'assemblage et en code machine *Fifth*.
On notera :

— Les paramètres d'une fonction sont passés sur la pile dans l'ordre de déclaration ;
— Tous les sauts sont faits avec des adresses absolues et non relatives (la première instruction est à l'adresse `0x0000`) ;
— Chaque programme doit se terminer par l'instruction `halt`, qui assure que les différents circuits ne répondent plus à l'horloge ;
— Un signal `resetSig` permet de réinitialiser le registre PC ainsi que les deux piles du *Fifth*.

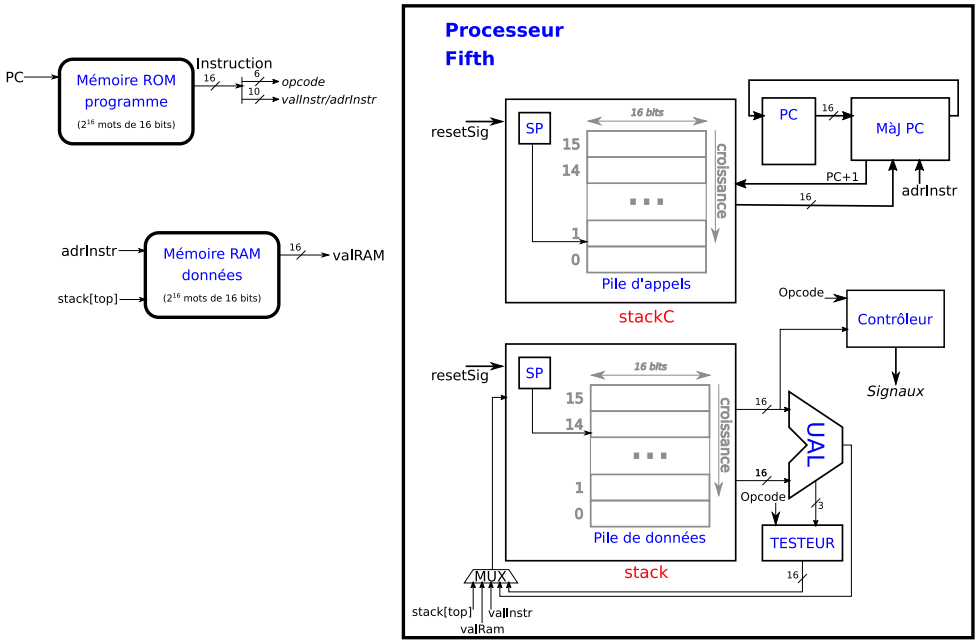


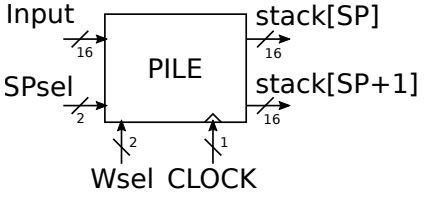
FIGURE 1 – Le processeur *Fifth* (vue partielle — tous les éléments ne sont pas décrits)

L'objectif du projet est de réaliser pas à pas un processeur *Fifth* fonctionnel dans *Logisim*. Pour cela, on respectera les bonnes pratiques vues lors des séances de travaux pratiques :

— Nommage de tous les signaux d'entrée/sortie (*label*) ;
— Commentaires à bon escient des parties importantes ;
— Utilisation de sous-circuits pour réduire la complexité des circuits ;
— Utilisation de *tunnels* et de *splitters* pour réduire le nombre de fils.

1 La pile

Un circuit de pile est composé de 16 registres r_0, …, r_{15} de 16 bits et d'un registre SP de 4 bits indiquant le numéro du dernier registre utilisé dans la pile.
La pile dispose de quatre entrées et deux sorties :



`Input`. L'entier sur 16 bits à stocker sur la pile ;

`SPsel`. La modification à effectuer sur le registre SP :

00. Remettre à zéro le registre,

01. Incrémenter le registre (pop),

10. Pas de modification,

11. Décrémenter le registre (push) ;

Table 2 – Exemple de programme pour le *Fifth*

Programme C

```c
int main(void)
{
  int x = fact(5);
}

int fact(int n)
{
  if (n == 0) {
    return 1;
  } else {
    return n*fact(n-1);
  }
}
```

Programme ASM *Fifth*

```
main:
  push 5
  call fact
  pop [0000]
  halt

fact:
if:
  dup
  push 0
  eq
  jmpf else
then:
  pop
  push 1
  jmp endif
else:
  dup
  push 1
  sub
  call fact
  mul
endif:
  ret
```

Code machine *Fifth*

```
# main:
0x4405
0x5404
0x5000
0x6000
# fact:
# if:
0x5c00
0x4400
0x3000
0x400b
# then:
0x4c00
0x4401
0x3810
# else:
0x5c00
0x4401
0x0400
0x5404
0x0c00
# endif:
0x5800
```

`Wsel`. Le sélecteur de registre devant recevoir une nouvelle valeur :

00. `stack[SP]`,

01. `stack[SP-1]`,

10. `stack[SP+1]`,

11. Pas d'écriture ;

`CLOCK`. Horloge régulant la lecture des entrées des registres ;

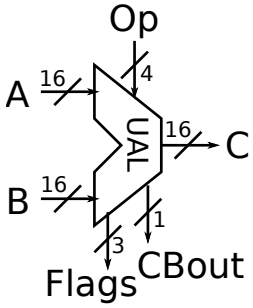
`stack[SP]`. Valeur courante sur le haut de la pile ;

`stack[SP+1]`. Valeur se trouvant juste sous le haut de la pile ;

1. Donner la valeur des indicateurs `SPsel` et `Wsel` des deux piles du *Fifth* pour toutes les instructions supportées ;
2. Créer le circuit pour une pile dans *Logisim*.

2 L'Unité Arithmétique et Logique

L'Unité Arithmétique et Logique (UAL) apporte toutes les fonctionnalités de calcul du *Fifth* et permet en particulier d'exécuter les onze opérations supportées par le processeur (`add`, `sub`, `neg`, `mul`, `div`, `rem`, `not`, `and`, `or`, `shl`, `sra`).
L'UAL comporte trois entrées et trois sorties :



`A`. Premier argument sur 16 bits. C'est l'argument de gauche des opérations binaires et celui utilisé pour les opérations unaires ;

`B`. Deuxième argument sur 16 bits. C'est l'argument de droite des opérations binaires ;

`Op`. Codage sur 4 bits de l'opération à effectuer ;

`C`. Résultat de l'opération sur 16 bits entre `A` et `B` ;

`CBout`. *Carry* (si l'opération demandée est une addition) ou *Borrow* (si l'opération demandée est une soustraction) éventuelle. Sa valeur importe peu si l'opération n'est ni une addition, ni une soustraction ;

`Flags`. Indicateurs `OF` (*Overflow Flag*), `SF` (*Sign Flag*) et `ZF` (*Zero Flag*).

1. Soient A_{15}, B_{15} et C_{15} les bits de poids forts des opérandes et du résultat. Donnez les formules logiques pour calculer `OF` dans le cas de l'addition et dans celui de la soustraction ;
2. Implémenter l'UAL dans *Logisim*.

3 Le testeur

Les instructions `lt`, `eq` et `gt` prennent les deux valeurs sur le dessus de la pile de données et empilent le résultat du test correspondant. Le module de test prend en entrées l'opcode d'une instruction de test courante ainsi que les trois indicateurs `OF`, `SF` et `ZF` venant de l'UAL ; il retourne la valeur 1 ou 0 sur 16 bits correspondant au résultat du test.

1. Déterminer la valeur de chaque test en fonction des trois indicateurs ;
2. Implémenter le module de test dans *Logisim*.

4 Le module de mise à jour de PC

Le module de mise à jour détermine la nouvelle valeur du registre `PC` en fonction de sa valeur courante et du type d'instruction courante :

— Si l'instruction est un saut inconditionnel (`jmp` ou `call`), `PC` est fixé à la valeur de l'adresse passée en paramètre ;

— Si l'instruction est un saut conditionnel (`jmpt` et `jmpf`), le module fixe `PC` à `PC+1` ou à l'adresse passée en paramètre en fonction du résultat du test ;
— Si l'instruction est un `ret`, le module fixe `PC` à l'adresse se trouvant sur le haut de la pile d'appel ;
— Dans tous les autres cas, `PC` est incrémenté.

Implémenter le module de mise à jour de `PC`.

5 Le contrôleur

Le contrôleur détermine la valeur de tous les signaux utilisés dans le *Fifth* (sélecteurs de multiplexeurs, …). Il prend en entrées l'opcode de l'instruction courante et la valeur se trouvant sur le haut de la pile de données (correspondant au résultat du dernier test effectué, le cas échéant) et retourne l'ensemble des signaux nécessaires à la bonne exécution des instructions du *Fifth*.

1. Donner le tableau récapitulant pour chaque instruction la valeur des différents signaux ;
2. Implémenter le module de contrôle dans *Logisim*. On pourra utiliser une mémoire ROM contenant à l'adresse i l'ensemble des valeurs des signaux pour l'instruction d'opcode i.

6 Le Fifth

1. En utilisant l'ensemble des circuits réalisés plus haut, implémenter un *Fifth* et le connecter à une mémoire ROM de programme de 2^{16} mots de 16 bits et à une mémoire RAM de données de 2^{16} mots de 16 bits également ;
2. Tester le *Fifth* avec le programme donné en exemple dans la table 2 ;
3. Proposer un autre programme non trivial pour tester les possibilités du *Fifth*. On donnera la version C, la version assembleur et le code machine correspondant.