

Rapport de Systèmes Numériques

Marc DUCRET

Florentin GUTH

Martin RUFFEL

Lionel ZOUBRITZKY

15 janvier 2017

1 Structure du système

1.1 Architecture du processeur

On choisit de stocker les mots sur 32 bits (4 octets). En effet, cela permettra d'utiliser des entiers (signés seulement, pour simplifier) de taille importante. On utilise 8 registres (codés sur 2 bits) en plus du *program counter*. Les adresses sont codées sur 17 bits, ce qui fait une *RAM* de 128 Ko.

1.2 Organisation de la *RAM*

Adresse	Nombre de mots	Données
0x00000	$s_w * s_h$	Contenu affiché à l'écran (caractères)
0x10000	1	Indique si il est nécessaire de redessiner l'écran
0x10001	1	Indique si le programme a terminé son exécution
0x10002	1	Largeur de l'écran s_w
0x10003	1	Hauteur de l'écran s_h
0x10004	1	Temps en secondes depuis le 01/01/1970 00h00
0x10005	?	État du clavier
0x11000	?	Variables du programme

TABLE 1 – Organisation de la *RAM*

1.3 Opérations implémentées

On a fait le choix d'une représentation peu factorisée, donc facile à décoder.

Pour obtenir le code hexadécimal d'une instruction, ajouter le code correspondant à l'opération, plus celui correspondant au(x) registre(s) (multiplié par 8 le cas échéant) plus éventuellement celui de la constante (multiplié par 64 le cas échéant)

Les instructions de décalage (**lsl** et **lsr**) ne prennent en compte que les 5 derniers bits du registre de décalage. Les instructions contenant une adresse fonctionnent de même modulo 2^{17} . **ZF** est le drapeau correspondant à un résultat nul, il est levé ou abaissé par l'instruction précédente.

	imm	weReg	ramOp	weRamOp	jzOp	jnzOp	branchOp	addOp	subOp	compOp	ltOp	eqOp	negOp	shiftOp	lrOp	movOp	orOp	xorOp	andOp	mulOp	Ø	Ø	Ø	Ø	Ø	Ø	R1			R2							
Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	Signification				Code hexadécimal
add	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		R1			R2			R1 « R1 + R2	0x41000000			
sub	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		R1			R2			R1 « R1 - R2	0x41800000			
sll	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0		R1			R2			R1 « R1 lsl R2	0x40040000			
srl	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0		R1			R2			R1 « R1 lsr R2	0x40060000			
and	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0		R1			R2			R1 « R1 & R2	0x40002000			
or	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0		R1			R2			R1 « R1 R2	0x40008000			
xor	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0		R1			R2			R1 « R1 ^ R2	0x40004000			
slt	0	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0		R1			R2			R1 « R1 < R2	0x41E00000			
sle	0	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0		R1			R2			R1 « R1 ≤ R2	0x41F00000			
seq	0	1	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0		R1			R2			R1 « R1 = R2	0x41D00000			
sne	0	1	0	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0		R1			R2			R1 « R1 ≠ R2	0x41D80000			
mul	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0		R1			R2			R1 « R1 * R2	0x40001000			
move	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0		R1			R2			R1 « R2	0x40010000			
sw	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0		R1			R2			RAM[R2] « R1	0x30010000			
lw	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0		R1			R2			R1 « RAM[R2]	0x60010000			
swi	1	0	1	1	0	0	0	0	0	IMMEDIATE																	R1			Ø	RAM[IMM] « R1			0xB0000000			
lwi	1	1	1	0	0	0	0	0	0	IMMEDIATE																	R1			Ø	R1 « RAM[IMM]			0xE0000000			
li	1	1	0	0	0	0	0	0	0	IMMEDIATE																	R1			Ø	R1 « IMM			0xC0000000			
j	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0		R1			Ø			PC « R1	0x0C010000			
jz	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0		R1			Ø			PC « R1 si ZF = 1	0x08010000			
jnz	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0		R1			Ø			PC « R1 si ZF = 0	0x04010000			
ji	1	0	0	0	1	1	0	0	0	IMMEDIATE																	Ø			Ø	PC « IMM			0x8C000000			
jzi	1	0	0	0	1	0	0	0	0	IMMEDIATE																	Ø			Ø	PC « IMM si ZF = 1			0x88000000			
jnzi	1	0	0	0	0	1	0	0	0	IMMEDIATE																	Ø			Ø	PC « IMM si ZF = 1			0x84000000			
beqi	1	0	0	0	1	0	1	1	1	IMMEDIATE																	R1			R2			PC « IMM si R1 = R2	0x8B800000			
bnei	1	0	0	0	0	1	1	1	1	IMMEDIATE																	R1			R2			PC « IMM si R1 ≠ R2	0x87800000			

2 Processeur

2.1 Fonctionnement général

Le processeur est écrit en *MiniJazz*, puis compilé en *net-list*. La *net-list* est ensuite compilée en C, puis en assembleur x86-64 par *gcc* afin d'être exécutée le plus rapidement possible.

Le code *MiniJazz* du processeur a été optimisé de la façon suivante :

- dérécurcification des fonctions pour opérer directement sur 32 bits,
- remplacement des nappes de fils par 32 variables,

dans le but d'éviter au maximum les recopies lors de la simulation du processeur.

2.2 Architecture du programme *MiniJazz*

3 Montre digitale

3.1 Le langage *Tong*

On utilise un langage intermédiaire qui sera ensuite compilé en assembleur (de notre processeur) puis en binaire. Ce langage, *Tong*, dispose des fonctionnalités suivantes :

- on peut déclarer des variables globales (représentant des entiers) dans le champ `.vars`,
- on écrit les instructions dans le champ `.prgm`,
- on peut déclarer des fonctions avec la syntaxe `@ main`, que l'on appelle avec `> main;`,
- on peut affecter des expressions aux variables : `x = (y * 2) & (5 | a);`,
- on peut effectuer des tests avec `? x = 1 { then } { else }`,
- on peut effectuer des boucles avec `! x < 60 { x = x + 1; }`,
- on peut effectuer des appels systèmes `> draw addr 'c'`,
- le temps en secondes depuis le 1^{er} janvier 1970 à minuit est stocké dans la variable `time`.

3.2 Compilation du *Tong*

Les variables locales reçoivent à la compilation une adresse statique dans la *RAM*. Les expressions sont quant à elles calculées dans les registres (si le nombre de registres ne suffit pas à calculer l'expression considérée, c'est à l'utilisateur de la découper en deux).

On a choisi de ne pas implémenter de pile. Ainsi, les appels de fonction sont *inlinés* (on interdit ainsi les fonctions récursives, qui peuvent être réalisées avec une boucle !), ce qui évite d'avoir à gérer les adresses de retour et privilégie la vitesse plutôt que la taille de l'exécutable.