Aurélien Emmanuel, Diane Gallois-Wong, Raphaël Rieu-Helft, Nissim Zerbib

Montre digitale : rapport intermédiaire

Architecture :

On utilise un microprocesseur spécialement adapté au problème de la montre digitale, avec un jeu d'instructions réduit et une architecture personnalisée.

Changements par rapport à l'architecture vue en cours :

- On réduit autant que possible la taille des instructions et des mots mémoire (on passe à 16 bits)
- On enlève la RAM (donc un PC, une ROM contenant le programme, et 32 registres)
- On ajoute les fils d'entrées/sorties suivants, avec lesquels on communique via des instructions spécifiques.

Entrées/sorties :

- pour chaque chiffre à afficher, une nappe de 7 fils de sortie ("bâtons" de l'afficheur à cristaux liquides). Chaque fil est en sortie d'un flip-flop ("registre" dans le vocabulaire du simulateur), la nappe est protégée par un mux pour ne modifier sa valeur que si nécessaire et afficher l'heure en continu. En amont, un circuit Dec7 convertit l'entier à afficher en bâtonnets.
- Boutons d'arrêt/réglage de l'heure représentés par des fils d'entrées branchés sur des flip-flops dont le programme vérifie l'état régulièrement
- Un fil d'entrée branché sur un flip-flop tient lieu de quartz : à une fréquence fixée (1024 Hz si on y arrive, c'est traditionnellement ça mais la boucle du programme doit s'exécuter entièrement entre deux tics de quartz), le simulateur CAML passe l'entrée (donc le flip-flop) à 1.

Boucle principale (schématiquement) :

tester si on appuie sur des boutons d'arrêt/réglage de la montre, si oui gérer ça
WHILE (quartz à 0) FIN WHILE
Remettre le flip-flop de quartz à 0
Incrémenter un compteur de tics, le cas échéant incrémenter les secondes/minutes/heures/date (si le compteur atteint 1024) et changer les valeurs affichées à l'écran
GOTO début

Utilisation des registres :

Un registre fixé contient toujours le nombre de tics de quartz modulo 1024, on incrémente les secondes quand il passe à 0

Utiliser un registre fixé pour chaque chiffre affiché à l'écran (pour les minutes et les secondes, il suffit alors d'incrémenter et tester l'égalité à 6/10 pour savoir s'il y a retenue)
Pour les heures, on utilise un troisième registre qui contient le nombre d'heures total : il n'y a pas retenue à chaque fois que les heures passent à 4 modulo 10...
Pour les jours/mois/années, on utilise aussi un registre par chiffre et un registre pour le nombre

total. On utilise aussi encore un registre qui contient le nombre total de jours du mois courant pour éviter de le recalculer tous les jours.
On utilise donc au total une quinzaine de registres pour contenir les données de l'écran, et quelques autres registres réservés ($0, $at, $k0...) le reste servant de temporaires.

Jeu d'instructions :

On a besoin d'assez peu d'arithmétique si on stocke des informations dans beaucoup de registres : seulement de l'incrémentation, de tests d'égalité (à 6/10/24 essentiellement, mais on a besoin de plus pour les mois donc on en fait un général) et de calculs modulo 4 pour les années bissextiles.
On a pour l'instant décidé de restreindre les années gérées à une période strictement comprise entre 1900 et 2100 : les années bissextiles sont donc vraiment les multiples de 4.

On utilise aussi des instructions de flot de contrôle : on n'utilisera probablement pas vraiment toutes celles présentes dans le jeu d'instructions actuel.

3 formats d'instructions :

RR : opcode (4) rs (5) rt (5) unused (2)
RI : opcode (4) rs (5) imm (7)
I : opcode (4) imm (12)

Il y a très peu d'instructions (5 dans RI, 3 dans I, 3 dans RR) ; les opcodes commencent par 0 pour RI, 10 pour I et 11 pour RR, il reste 3 bits pour RI et 2 bits pour les autres formats pour distinguer les instructions.

Nom	Syntaxe	Sémantique		Format	Opcode
Flot de contrôle					
Constant Branch On Equal	cbeq rs rt	if (rs==rt) PC=PC+8	en MIPS : beq rs rt 1	RR	1110
Jump	j addr	PC = addr		I	1001
Jump Register	jr rs	PC = rs	Peut-être inutile	RI	0001
Jump And Link	jal addr	$ra = PC+8 ; PC = addr	Peut-être inutile	I	1011
Jump and Link Register	jalr rs	$ra = PC+8 ; PC = rs	Peut-être inutile	RI	0101
Arithmétique					
Load Immediate	li rd imm	rd = imm	Habituellement une pseudoinstruction, mais on peut le faire en une instruction en imposant que imm soit relativement petit.	RI	0100
Increment	incr rs rd	rd = rs + 1		RR	1101
Modulo 4	modf rs rd	rd = rs mod 4		RR	1100
Load Big Immediate	lbi bimm	$k0 = bimm	Pour stocker la fréquence du quartz, il faut 11 bits en 1024Hz	I	1000
Communication avec les entrées et sorties de la netlist					
Load Input	lin rd imm	rd = entrée n° imm du circuit		RI	0000
Send Digit	sd rs imm	envoie le contenu de rs à un Dec7, puis la représentation de chiffre obtenue à la sortie n° imm		RI	0110

On utilise aussi au moins une pseudoinstruction, et on réserve le registre $at pour stocker les valeurs temporaires des pseudoinstructions.

Pseudoinstructions		
Constant Branch On Equal Immediate	cbeqi rs imm	li $at imm ; cbeq rs $at

On utilise aussi un système de labels, gérés par la suite en précompilation.

Détail des registres utilisés et autres constantes

$t0, $t1, etc. (nombre à déterminer) : temporaires

$qa : tics quartz

$sc0 : unités secondes
$sc1 : dizaines secondes
$mn0 : unités minutes
$mn1 : dizaines minutes
$hr0 : unités heures
$hr1 : dizaines heures
$hr : total heures
$da0 : unités jours
$da1 : dizaines jours
$da : total jours
$mo0 : unités mois
$mo1 : dizaines mois
$mo : total mois
$yr0 : unités années
$yr1 : dizaines années
$yr : total années

$nda : nombre de jours dans le mois courant

$k0 : destination de l'instruction lbi ; on y stocke la fréquence du quartz, ou 1 si on veut faire tourner la montre le plus vite possible, on teste l'égalité entre $qa et $k0 à chaque tic de quartz.
$ra : adresse de retour
$zero : toujours 0
$at : Assembler Temporary, stocke les valeurs temporaires des pseudoinstructions

Immediate de Send Digit :
0 : unités secondes
1 : dizaines secondes
2 : unités minutes
3 : dizaines minutes
4 : unités heures
5 : dizaines heures
6 : unités jours
7 : dizaines jours
8 : unités mois
9 : dizaines mois
10 : unités ans
11 : dizaines ans

Immediate de Load Input :
0 : relié au quartz
1 : Pause
2 : Incrémenter le nombre courant (secondes/minutes/heures/jour/mois/an)
3 : Changer le nombre courant
4 : Changer de mode (vitesse normale/rapide)

Exemple de code : gestion du mois

```
        //choix du prochain mois

        incr $mo $mo
        cbeqi $mo 13 //si on était en décembre, repasser en janvier
        j L0
        li $mo 1
        li $mo0 1
        li $mo1 0
        j L1
L0:     incr $mo0 $mo0      //sinon, gérer les chiffres du mois
        cbeqi $mo0 10       //gérer l'overflow du chiffre des unités
        j L1
        li $mo 0
        incr $m1 $m1
L1:     sd $m0 8            //changement de l'affichage
        sd $m1 9            //changement de l'affichage

        //actualisation de $nda

        cbeqi $nda 31 //après un mois à <31 jours vient toujours un mois à 31 jours
        j L2
        li $nda 30 //la plupart du temps, après un mois à 31 jours, vient un à 30 jours...
        cbeqi $nda 8 //...sauf si c'est août...
        j L3
        j L2
L3:     cbeqi $mo 1 //...ou janvier...
        j L4
        j L2
L4:     cbeqi $mo 2 //...ou février
        j L5
        li $nda 28
        modf $an $t0
        cbeqi $to $zero //année bissextile?
        li $nda 29
        j L5
L2:     li $nda 31
L5:     cbeqi $mo 1
        j debut //retourner au début du code : on a fini le tour de boucle
        incr $an $an
        // puis incrémenter $an0, éventuellement $an1 etc.
```