Instructions du processeur RISC

Syntaxe du langage d'assemblage

Format d'une ligne

Toute ligne débutant par // est considérée comme un commentaire et est ignorée par l'assembleur.
Tout autre ligne est de la forme suivante : Etiquette: Instruction/directive opérandes

- L'étiquette est optionnelle (utile uniquement en cas de branchement).
- Instruction/directive est obligatoire pour signifier une instruction (voir liste ci-après) ou une directive d'assemblage (à l'heure actuelle une seule existe, de nom DAT, et permet de réserver de la mémoire).

Exemples :

```
tab     DAT 10
        DAT 9
        DAT 8
bouc    add R2,R1,R0
        sub R2,R2,R1
```

DAT Réservation de mémoire – ceci est une directive d'assemblage et pas une instruction. Elle est exploitée par l'assembleur et réserve et initialise un (unique) mot mémoire

DAT valeur

Valeur est une constante tenant sur 16 bits, positive ou négative, exprimée en décimal ou hexadécimal (8, -1, 0xffff)

Conventions lors de la description des instructions

- **Registres** : Rx désigne le registre numéro x. Rd est utilisé quand Rd est la destination de l'instruction (écriture), Rs quand c'est la source de l'instruction (lecture)
- **Constantes** : elles sont notées #const. Const est une valeur positive ou nulle stockable 8 bits, et peut être exprimée en décimal ou hexadécimal (#1, #0x42). Si vous souhaitez manipuler des plus grosses valeurs, ou des valeurs négatives, réservez de la mémoire via la directive DAT.
- **Adresses de mots mémoire** : elles seront notées *addr* dans la description des instructions. Une adresse sera représentée par un entier exprimé en décimal ou hexadécimal, inférieur à la taille de la mémoire, ou le nom d'une étiquette (44, 0x10, tab).
- **Adressage avec offset** : offset(Rd) représente l'adresse Rd+offset. Offset doit être une valeur positive ou nulle représentable sur 6 bits, exprimée en décimal ou hexadécimal.
- **Raccourcis assembleur** : pour toutes les instructions à trois opérandes OP Rd,Rs,Rb, il existe un raccourci OP Rd,Rs accepté par l'assembleur (fait alors Rd :=Rd OP Rs)

Liste des instructions

Instructions simples et utiles

HLT Halt : arrêt de la machine

NOP No OPeration – Instruction qui ne fait rien

INP Input – entrée

INP Rd,device

INP Rd,Ra

Lecture d'une valeur au clavier, mise dans Rd. L'argument device ou Ra est actuellement ignoré. La valeur lue doit être représentable sur 16 bits, peut être positive ou négative, et peut être entrée en décimal ou hexadécimal (0xffff, -55, 67).

OUT Affichage
OUT Rs,device: affichage de la valeur contenue dans Rs
OUT Rs,Ra Affichage de la valeur contenus dans Rs (Ra actuellement ignorée)
Ra ou device contienent le format de l'affichage : 4 affiche en signé, 5 en non signé, 6 en hexadécimal, sur device 7 en caractère.

Opérations arithmétiques

Les instructions ADD, SUB and MUL sont les mêmes pour des entiers signés et non signés, seule la division a une variante signée (DIV) et non signée (UDV). La multiplication étendue (MLX) est non signée et ne met à jour aucun flag sauf Z. Seules les instructions arithmétiques, l'instruction MVN, et l'instruction de comparaison positionnent les flags.

ADD Addition

ADD Rd,#const	Rd := Rd+#const
ADD Rd,Rs,Rb	Rd:=Rs+Rb
ADD SP,#const	SP:=SP+#const
ADD Rd,addr	Rd:=Rd + contenu du mot mémoire d'adresse *addr*

SUB Soustraction

SUB Rd,#const	Rd:=Rd-#const
SUB Rd,Rs,Rb	Rd:=Rs-Rb
SUB SP,#const	SP:=SP-#const
SUB Rd,addr	Rd:=Rd - contenu du mot mémoire d'adresse *addr*

DIV Division signée

DIV Rd,Rs	Rd:=Rd/Rs

UDV Division non signée

UDV Rd,#const	Rd:=Rd / #const
UDV Rd,Rs	Rd:=Rd / Rs

MUL Multiplication de deux entiers 16 bits, résultat sur 16 bits

MUL Rd,#const	Rd:=Rd * #const
MUL Rd,Rs	Rd :=Rd*Rs

MLX Extended multiply – multiplication non signée de deux entiers 16 bits, résultat sur 32 bits

MLX Rd, Rs	Rd :=Rd*Rs

Met le résultat de la multiplication dans les registres Rd et Rd+1. Tous les flags à 0 sauf le flag Z

MOD Modulo

MOD Rd,#const	Rd := Rd modulo #const
MOD Rd,Rs	Rd := Rd modulo Rs

Opérations de transfert

LDR Lecture d'un registre depuis la mémoire

LDR Rd,offset(Rn)	Initialise Rd au contenu de l'adresse Rn+offset
LDR Rd,addr	Initialise Rd au contenu de l'adresse *addr*
LDR Rd,offset(SP)	Initialise Rd au contenu de l'adresse SP+offset

STR Ecriture d'un registre dans la mémoire

STR Rs,offset(Rn) Ecrit Rs à l'adresse Rs+offset
STR Rs,addr Ecrit Rs à l'adresse addr
STR Rs,offset(SP) Ecrit Rs à l'adresse SP+offset

MOV Transfert registre vers registre ou chargement registre avec une valeur constante

MOV Rd,Rs Rd:=Rs
MOV Rd,#const Rd := #const
MOV Rd,flags/SP/LR/PC Rd := flags/SP/LR/PC

Opérations logiques (bit-à-bit)

AND Opération "et" bit-à-bit

AND Rd,#const Rd:=Rd et #const
AND Rd,Rs,Rb: Rd:=Rd et Rb

ORR Opération "ou" bit-à-bit

ORR Rd,#const Rd:=Rd ou #const
ORR Rd,Rs,Rb: Rd:=Rd ou Rb

XOR Opération "ou exclusif" bit-à-bit

XOR Rd,#const Rd:=Rd ou exclusif #const
XOR Rd,Rs,Rb: Rd:=Rd ou exclusif Rb

NEG Negation
NEG Rd,Rs Rd:=negation de Rs

Opérations de décalage et rotation

ASR Arithmetic shift right – décalage arithmétique à droite
ASR Rd,#count Rd est décalé de #count bits à droite
ASR Rd,Rs Rd est décalé vers le droit de Rs bits

LSL Logical Shift Left - Décalage bit-à-bit (logique) à gauche
LSL Rd,Rs,#count Rd:=Rs décalé à gauche de #count
LSL Rd,Rs,Rb Rd:=Rs décalé à gauche du contenu de Rb

LSR Logical Shift Right - Décalage bit-à-bit (logique) à droite

LSR Rd,Rs,#count Rd:=Rs décalé à droite de #count
LSR Rd,Rs,Rb Rd:=Rs décalé à droite du contenu de Rb

ROR Rotate right – Rotation à droite
ROR Rd, #count – rotation de Rd de #count bits vers la droite
ROR Rd,Rs – rotation de Rs bits vers la droite

Comparaisons et branchements

CMP Comparaison
CMP Rd,#const positionne les codes condition selon la comparaison de Rd et de la valeur #const

CMP Rb,Rs positionne les codes condition selon la comparaison de Rb et Rs

BRA Branch Always (branchement inconditionnel)
Existe aussi en forme BRA Rs

Branchements conditionnels
Bxx address

Conditions indépendantes du caractère signé/non signé
BCC Branch if Carry Clear (branchement si C=0)
BCS Branch if Carry Set (branchement si C=1)
BEQ Branch if EQual (branchement si égal – Z=1)
BNE Branch if Not Equal (branchement si non égal – Z=0)
BVC Branch of oVerflow clear (branchement si V=0)
BVS Branch of oVerflow set (branchement si V=1)
BMI Branch if MInus (branchement si négatif – N=1)
BPL Branch if PLus (branchement si strictement positif – N=0)

Conditions sur entiers signés
BGT Branch if Greater Than (branchement si strictement supérieur)
BGE Branch if Greater or Equal (branchement si supérieur ou égal)
BLT Branch if Less Than (branchement si strictement inférieur)
BLE Branch if Less than or Equal (branchement si inférieur ou égal)

Conditions sur entiers non signés
BHI Branch if HIgher than (branchement si strictement supérieur – C =1 et Z=0)
BHS Branch if Higher or same (branchement si supérieur ou égal – C=1)
BLO Branch if LOwer than (branchement si strictement inférieur – C=0)
BLS Branch if Lower or Same (branchement si inférieur ou égal – C=0 ou Z=1)

Appels de fonction et gestion de la pile

JMS JuMp to Subroutine - appel de fonction
JMS addr
Initialise LR (adresse de retour de la fonction) à PC+1 et branche à l'adresse addr (PC:=addr).

RET Return - Retour de fonction
Retourne à l'adresse contenue dans LR (adresse de retour de la fonction).

PSH Push in stack – empilage
PSH Rs Met le contenu de Rs à l'adresse contenue dans SP, puis décrémente SP. Pour empiler un registre spécial (PC, LR), il faut d'abord transférer son contenu dans un des registres généralistes R0 à R7.

POP Pop from stack – dépilage
POP Rd Incrémente SP puis met dans Rd le contenu du mot mémoire à l'adresse SP. Rd est un registre généraliste, qui peut par la suite si nécessaire être transféré dans un registre spécial (PC, LR).

Instructions d'utilisation moins courante

TST Test (et bit à bit)
TST Rb,Rs Positionne les flags selon le et bit à bit entre Rb et Rs. Peut être utilisé par exemple pour tester si un bit est positionné dans un entier.

MVN Move not – transfert avec opération non bit à bit
MVN Rd,Rs Rd:=non RS

ADC ADC Rd,Rs
ADD with carry included : comme un ADD mais ajoute le bit C (carry) s'il est à 1

SBC SBC Rd,Rs
SUB with carry included : comme un SUB mais si le bit C (Carry) est à 0 retire 1 au résultat

BIC Logical bit clear
BIC, Rd,Rs Clear in Rd the bits set in Rs

BIS Logical bit set
BIS, Rd,Rs Set in Rd the bits set in Rs