

Éléments de langage d'assemblage

RISCV

Jeu d'instructions

Le RISCV est un jeu d'instruction de type RISC. Il possède des instructions codées sur 32 bits réparties dans 6 formats différents :

31	25	24 20	19 15	14 12	11	7	6	0	
funct7		rs2	rs1	funct3	rd		opcode		R
imm[11 :0]			rs1	funct3	rd		opcode		I
imm[11 :5]		rs2	rs1	funct3	imm[4 :0]		opcode		S
imm[12 :10 :5]		rs2	rs1	funct3	imm[4 :1 :11]		opcode		B
imm[31 :12]					rd		opcode		U
imm[20 :10 :1 ;11 :19 :12]					rd		opcode		J

Ci-contre un tableau récapitulatif des principales instructions, la mnémonique associée ainsi qu'un descriptif de l'opération effectuée dans la partie opérative du processeur.

Mnémonique	Format	Opcode/Func3/Func7	Opération	Nom complet
addi rd, rs1, imm	I	0x13/0	$R[rd]=R[rs1]+SignExtImm^1$	Addition avec immédiat
add rd, rs1, rs2	R	0x33/0/0	$R[rd]=R[rs1]+R[rs2]$	Addition non signée entre registres
and rd, rs1, rs2	R	0x33/7/0	$R[rd]=R[rs1] \& R[rs2]$	ET bit à bit entre registres
andi rd, rs1, imm	I	0x13/7	$R[rd]=R[rs1] \& SignExtImm^1$	ET bit à bit avec immédiat
auipc rd, imm	U	0x17	$R[rd]=PC+IR_{31..12} \parallel 0^{12}$	Addition PC avec immédiat
beq rs1, rs2, label	B	0x63/0	if($R[rs1]==R[rs2]$) $PC=PC+BranchAddr^2$	Branchement si égalité
bne rs1, rs2, label	B	0x63/1	if($R[rs1] \neq R[rs2]$) $PC=PC+BranchAddr^2$	Branchement si inégalité
jal rd, label	J	0x6f	$R[rd]=PC+4; PC=JumpAddr^3$	Saut inconditionnel avec lien
jalr rd, imm(rs1)	I	0x67/0	$R[rd]=PC+4; PC=R[rs1]+imm$	Saut inconditionnel registre avec lien
lui rd, imm	U	0x37	$R[rd]=IR_{31..12} \parallel 0^{12}$	Chargement immédiat haut
lw rd, imm(rs1)	I	0x03/2	$R[rd]=mem[R[rs1]+SignExtImm^1]$	Chargement registre
or rd, rs1, rs2	R	0x33/6/0	$R[rd]=R[rs1] \mid R[rs2]$	OU bit à bit registre à registre
ori rd, rs1, imm	I	0x13/6	$R[rd]=R[rs1] \mid SignExtImm^1$	OU bit à bit avec immédiat
slt rd, rs, rt	R	0x33/2/0	$R[rd]=(R[rs1]<R[rs2])?1:0$	Set Less Than registre à registre
slti rd, rs1, imm	I	0x13/2	$R[rd]=(R[rs1]<SignExtImm^1)?1:0$	Set Less Than avec immédiat
slli rd, rs1, sh	I	0x13/1	$R[rd]=R[rs1]<<SH(=IR_{24..20})$	Décalage à gauche
srli rd, rs1, sh	I	0x13/5	$R[rd]=R[rs1]>>SH(=IR_{24..20})$	Décalage logique à droite
sub rd, rs1, rs2	R	0x33/0/0x20	$R[rd]=R[rs1]-R[rs2]$	Soustraction
sw rs2, imm(rs1)	S	0x23/2	$mem[R[rs1]+StoreAddr^4]=R[rs2]$	Stockage registre
xori rd, rs1, imm	I	0x13/4	$R[rd]=R[rs1] \oplus SignExtImm^1$	OU exclusif bit à bit avec immédiat

1. $SignExtImm=IR_{31}^{20} \parallel IR_{31..20}$

2. $BranchAddr=IR_{31}^{20} \parallel IR_7 \parallel IR_{30..25} \parallel IR_{11..8} \parallel 0$

3. $JumpAddr=IR_{31}^{12} \parallel IR_{19..12} \parallel IR_{20} \parallel IR_{30..25} \parallel 0$

4. $StoreAddr=IR_{31}^{20} \parallel IR_{31..25} \parallel IR_{11..7}$
- j label \equiv jal rd, label avec rd=0

jr rs1 \equiv jalr rd, imm(rs1) avec rd=0 et imm=0

ret \equiv jr ra

lb/lh, chargement octet/demi-mot

sb/sh, sauvegarde octet/demi-mot
- Autres branchements :

bge, bgeu, blt, bltu

Comparaisons non-signées :

sltu, sltiu

Pseudo-instructions

Les pseudo-instructions facilitent l'écriture des programmes.

Pseudo-instruction	Opération	Expansion possible
li rd, cste	$R[rd]=cste$	lui rd, %hi(cste) ori rd, rd, %lo(cste)
la rd, symb	$R[rd]=adresse\ de\ symb$	auipc rd, %hi(symb) addi rd rd, %lo(symb)
lw rd, symb	$R[rd]=MEM[symb]$	auipc rd, %hi(symb) lw rd, %lo(symb)(rd)
~~ pareil pour lb, lbu, lh, lhu		
sw rs, symb, rt	$MEM[symb]=R[rs]$	auipc rt, %hi(symb) sw rs, %lo(symb)(rt)
~~ pareil pour sb, sh		
mv rd, rs	$R[rd]=R[rs]$	addi rd, rs, 0
nop	ne fait rien	addi x0, x0, 0
not rd, rs	$R[rd]=not\ R[rs]$	xori rd, rs, -1
seqz rd, rs	$R[rd]=1\ if\ rs=0\ else\ 0$	sltu rd, x0, rs
~~ similaire pour snez, sltz, sgtz		
beqz rs, label	saut si rs = 0	beq rs, zero, label
~~ similaire pour bnez, blez, bgez, bltz, bgtz		
bgt rs, rt, label	saut si rs > rt	blt rt, rs, label
~~ similaire pour ble, bgtu, bleu		
j label	saut inconditionnel	jal zero, label
jr rs	saut à l'adresse dans rs	jalr zero, 0(rs)
jal label	saut + maj de ra	jal ra, label
ret	retour de fonction	jalr zero, 0(ra)

Syntaxe assembleur

Quelques éléments de syntaxe :

1. Un nom de fichier est suffixé par « .s ».

2. Les commentaires démarrent par un # ou un ; et s'achèvent en fin de ligne. Une bonne pratique consiste à commenter les instructions en fin d'instruction pour faciliter la compréhension du code.

Ceci est une fonction qui fait des choses

sw ... ; sauve la valeur copiée dans la mémoire
3. Un entier hexadécimal est préfixé par 0x et un entier décimal est égal à lui-même ($250_{10}=0xFA_{16}$).

4. Une chaîne de caractères est entourée de guillemets et peut contenir des caractères d'échappements. Par exemple "**Chaîne avec retour à la ligne**\n".

5. Un label est suffixé par : et correspond soit à une adresse de variable, soit à une adresse de saut.

6. Un immédiat est une opérande contenue dans une instruction. Cette constante est toujours étendue de signe sur 32 bits, sa taille initiale varie en fonction du type d'instruction (I, S, B, U ou J).

7. Une ligne correspondant à une instruction débute par l'instruction suivie de ses arguments. Ceux-ci sont séparés par des virgules (**add t0, t1, t2**).
8. Les registres sont au nombre de 32 et leur utilisation dépend des conventions suivantes :
- | Nom matériel | Nom logiciel | Signification | Préservé ? |
|--------------|--------------|-----------------------|------------|
| x0 | zero | Zéro | Oui(0) |
| x1 | ra | Adresse de retour | Non |
| x2 | sp | Pointeur de pile | Oui |
| x3 | gp | Pointeur global | X |
| x4 | tp | Pointeur de tâche | X |
| x5-x7 | t0-t2 | Registres temporaires | Non |
| x8-x9 | s0-s1 | Registres préservés | Oui |
| x10-x17 | a0-a7 | Registres arguments | Non |
| x18-x27 | s2-s11 | Registres préservés | Oui |
| x28-x31 | t3-t6 | Registres temporaires | Non |
- ### Convention pour les appel de fonctions
- appelante (*caller*) : positionne les paramètres dans les registres (et la pile si nécessaire), saute à l'appelée.

— appelée (*callee*) : réserve la place nécessaire à son propre contexte d'exécution.

— 8 premiers paramètres dans a0 à a7 les autres dans la pile.

— a0 : contient la valeur de retour de la fonction.

— sp : contient l'adresse de la dernière case occupée dans pile.
- | | | | |
|-----------------|-------------------------------------|-----------|-----------|
| adresses hautes | espace pour paramètres (9, 10, ...) | optionnel | appelante |
| ↓ | adresse de retour | optionnel | appelée |
| ↓ | registres à sauver | optionnel | |
| ↓ | variables locales | optionnel | |
| adresses basses | espaces pour paramètres (9,10,...) | optionnel | |

