Projet Systèmes numériques

Pierre-Alexandre Bazin, Mathis Bouverot & Arthur Rousseau

19 janvier 2021

ISA - Mémoire

Tous les entiers manipulés sont signés et sur 16 bits. La RAM a des adresses sur 16 bits, et ses mots sont de longueur 16 bits. L'entièreté des 2^{16} mots de la RAM sont adressables. Une partie de la RAM est réservée aux entrées et sorties du processeur.

`adresse de début`	`fonction`
0	aucune
$2^{16} - 16$	sorties
$2^{16} - 8$	entrées

ISA - Registres

Il y a 8 registres nommés rax, rbx, rcx, rdx, rex, rfx, rgx, rz (registre nul). Ces registres sont sur 16 bits.
Codage des registres :

rz	rax	rbx	rcx	rdx	rex	rfx	rgx
000	001	010	011	100	101	110	111

ISA - Instructions

Les instructions sont sur 32 bits. Leur schéma est le suivant :

imm	funct7	funct3	rs2	rs1	rd	opcode

avec

- `opcode` sur 3 bits
- `funct3` sur 3 bits
- `funct7` sur 1 bit
- `rd`, `rs2`, `rs1` sur 3 bits
- `imm` sur 16 bits

ISA - Instructions

Inst	Nom	Opcode	funct3	funct7	Description
`add`	ADD	011	000	0	`rd = rs1 + rs2`
`sub`	SUB	011	000	1	`rd = rs1 - rs2`
`or`	OR	011	100	0	`rd = rs1 or rs2`
`nand`	NAND	011	100	1	`rd = rs1 nand rs2`
`xor`	XOR	011	001	0	`rd = rs1 xor rs2`
`nxor`	NXOR	011	001	1	`rd = rs1 nxor rs2`
`and`	AND	011	101	0	`rd = (rs1 and rs2)`
`nor`	NOR	011	101	1	`rd = rs1 nor rs2`
`sll`	Shift left logical	011	011	0	`rd = rs1 « rs2`
`srl`	Shift right logical	011	010	0	`rd = rs1 » rs2`
`sra`	Shift right Arith	011	010	1	`rd = rs1 » rs2`
`seq`	Set equal	011	111	1	`rd = (rs1 = rs2)?1:0`
`slt`	Set less than	011	110	1	`rd = (rs1 < rs2)?1:0`

ISA - Instructions

Immediate

Inst	Nom	Opcode	funct3	funct7	Description
`addi`	ADD	001	000	0	`rd = rs1 + imm`
`subi`	SUB	001	000	1	`rd = rs1 - imm`
`ori`	OR	001	100	0	`rd = rs1 or imm`
`nandi`	NAND	001	100	1	`rd = rs1 nand imm`
`xori`	XOR	001	001	0	`rd = rs1 xor imm`
`nxori`	NXOR	001	001	1	`rd = rs1 nxor imm`
`andi`	AND	001	101	0	`rd = rs1 and imm`
`nori`	NOR	001	101	1	`rd = rs1 nor imm`
`slli`	Shift left logical	001	011	0	`rd = rs1 « imm`
`srli`	Shift right logical	001	010	0	`rd = rs1 » imm`
`srai`	Shift right Arith	001	010	1	`rd = rs1 » imm`
`seqi`	Set equal	001	111	1	`rd = (rs1 = imm)?1:0`
`slti`	Set less than	001	110	1	`rd = (rs1 < imm)?1:0`

ISA - Instructions

Inst	Nom	Opcode	funct3	funct7	Description
`lw`	Load word	000	000	0	`rd=M[rs1+imm]`
`sw`	Store word	010	000	0	`M[rs1+imm]=rs2`
`beq`	Branch $==$	110	100	1	`if(rs1 == rs2) PC=imm`
`bne`	Branch $!=$	110	101	1	`if(rs1 != rs2) PC=imm`
`ble`	Branch $\leqslant$	110	110	1	`if(rs1 <= rs2) PC=imm`
`blt`	Branch $<$	110	010	1	`if(rs1 < rs2) PC=imm`
`bge`	Branch $\geqslant$	110	011	1	`if(rs1 >= rs2) PC=imm`
`bgt`	Branch $>$	110	111	1	`if(rs1 > rs2) PC=imm`
`jal`	Jump and link	111	000	0	`rd=PC+4; PC=imm`
`jalr`	Jump and link reg	101	000	0	`rd=PC+4; PC=rs1+imm`
`jr`	Jump reg	100	000	0	`PC=rs1+imm`
`jmp`	Jump	110	001	1	`PC=imm`

ISA - Instructions

On ajoute du sucre syntaxique au niveau de l'assembleur : certaines instructions ISA sont regroupées en une seule instruction assembleur (exemple : `mov` correspond aux instructions ISA `lw`, `addi`, `sw` entre autres).

Assembleur - Exemple

Syntaxe proche de x86-64 :

```
; Exemple de programme minias
    .section text
L1:
    add %rax %rbx 42
    mov x %rax
    jmp L1
    .section data
x:
    .space 2
y:
```

Assembleur - Instructions

Les instructions assembleur sont plus 'haut niveau' :

- Pas de distinction entre add/addi, sub/subi, etc.
- Une seule instruction `mov` pour toutes les copies.

Assembleur - Labels

Il y a des labels pour éviter de manipuler directement des adresses ROM/RAM.

- Dans la section text : permet de faire des jump.
- Dans la section data : permet d'accéder à la RAM avec l'instruction `mov`.