Microprocesseur RISC-V, horloge

Aghilas BOUSSAA, Marius CAPELLI, Olivier HENRY, Paul WANG

28 janvier 2025

Sommaire

1 ISA

2 ALU

3 RAM

4 Programme

Sommaire

1 ISA

2 ALU

3 RAM

4 Programme

RISC V

Figure – Logo de RISC-V [1].

1. RISC-V Foundation, CC BY-SA 4.0

Formats

31 : 25	24 : 20	19 : 15	14 : 12	11 : 7	6 : 0	type
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

Exemples

31 : 25	24 : 20	19 : 15	14 : 12	11 : 7	6 : 0	type
0_00000	rs2	rs1	_ _ _	rd	011_011	ALU
imm[11 : 0]		rs1	_ _ _	rd	001_011	ALU-I
imm[11 : 5]	rs2	rs1	_ _ _	imm[4 : 0]	0100011	Store
imm[12|10 : 5]	rs2	rs1	_ _ _	imm[4 : 1|11]	1100011	Branch
imm[31 : 12]				rd	0110111	LUI
imm[20|10 : 1|11|19 : 12]				rd	1101111	JAL

Instructions supportées

Catégorie	Instructions
ALU	or, and, add, sub, sll, slt, sltu, xor, srl, sra
ALU-I	ori, andi, addi, slli, slti, sltiu, xori, srli, srai
Jump	jal, jalr
Branch	beq, bne, blt, bge, bltu, bgeu
Load/Store	lb, lh, lw, ld, lbu, lhu, lwu, sb, sh, sw, sd
Autre	lui, auipc
Horloge	gtck, gdt, sdt

Sommaire

1 ISA

2 ALU

3 RAM

4 Programme

Carry-lookahead adder

```
27 def propgen(aandb:list[Variable], aorb:list[Variable]) ->
   ↪  tuple[list[Variable], list[Variable]]:
28   '''returns the list of propagates and generates for blocs of
     ↪  aligned 2**k bits'''
29   n = len(aorb)
30   if n == 1:
31     return aorb, aandb
32   m = n//2
33   pd, gd = propgen(aandb[m:n], aorb[m:n])
34   pg, gg = propgen(aandb[0:m], aorb[0:m])
35   p = pg + pd
36   g = gg + gd
37   p.append(pd[-1] & pg[-1])
38   g.append(gd[-1] | (pd[-1] & gg[-1]))
39   return p, g
```

Réutilisation des circuits

```
98   n = a.bus_size
99   # ensures that b2 = b when add/xor/and/or and b2 = -b when
     ↪  sub/slt/sltu
100  isnotsub = funct32 | (~funct75 & ~funct31)
101  b2 = Mux(isnotsub, ~b, b)
102  aorb = a | b2
103  aandb = a & b2
104  axorb = a ^ b2
105  aplusb, overflow = add([axorb[i] for i in range(n)], [aorb[i]
     ↪  for i in range(n)], [aandb[i] for i in range(n)],
     ↪  ~isnotsub)
106  L, sll_ret = sll(a, b)
107  return multimux([funct30, funct31, funct32], [
108    aplusb,
109    sll_ret,
110    slt(a, b, aplusb),
111    sltu(a, b, aplusb),
112    axorb,
113    sral(a, b, funct75),
114    aorb,
```

Multiplicateur peu intelligent

```
4  def multiplier(L:list[Variable], b):
5      res = []
6      n = len(L)
7      for i in range(n):
8          res.append(Mux(b[i], Constant("0"*n), L[i]))
9      for i in range(log2i(n)-1):
10         res2 = []
11         for j in range(0, n>>i, 2):
12             and_tmp = res[j] & res[j+1]
13             or_tmp = res[j] | res[j+1]
14             xor_tmp = res[j] ^ res[j+1]
15             res2.append(add([xor_tmp[i] for i in range(n)],
                ↪  [or_tmp[i] for i in range(n)], [and_tmp[i] for i
                ↪  in range(n)], Constant("0"))[0])
16         res = res2
17     return res[0]
```

Sommaire

1 ISA

2 ALU

3 RAM

4 Programme

Organisation

Mots de 8, 16, 32 ou 64 bits, adressable à l'octet, circulaire.

RAM	000	100	010	110	001	101	011	111
00	→	→	→	→	→	→	→	↵
10	→	→	→	→	→	→	→	↵
01	→	→	→	→	→	→	→	↵
11	→	→	→	→	→	→	→	↵

Exemple : 110 01

Calcul des adresses

```
38      addrl.append(Mux(addr_remain0 | ar1_or_ar2, addr, addrplus))
        ↪  # if addr_remain >= 100 then addrplus else addr
39      addrl.append(Mux(ar1_or_ar2, addr, addrplus)) # if
        ↪  addr_remain >= 010 then addrplus else addr
40      addrl.append(Mux(addr_remain2 | (addr_remain0 &
        ↪  addr_remain1), addr, addrplus)) # if addr_remain >= 110
        ↪  then addrplus else addr
41      addrl.append(Mux(addr_remain2, addr, addrplus)) # if
        ↪  addr_remain >= 001 then addrplus else addr
42      addrl.append(Mux(addr_remain2 & (addr_remain1 |
        ↪  addr_remain0), addr, addrplus)) # if addr_remain >= 101
        ↪  then addrplus else addr
43      addrl.append(Mux(ar1_and_ar2, addr, addrplus)) # if
        ↪  addr_remain >= 011 then addrplus else addr
44      addrl.append(Mux(ar1_and_ar2 & addr_remain0, addr, addrplus))
        ↪  # if addr_remain >= 111 then addrplus else addr
45      addrl.append(addr)
```

Données à écrire

```
67      # Déterminer ce que l'on écrit
68      RAM_write_datal = cycle(1, [RAM_write_data[i:i+8] for i in
        ↪  range(0, 64, 8)][::-1])
69
70      # Déterminer où on écrit (c'est pareil que pour voir ce que
        ↪  l'on écrit)
71      wws0 = RAM_word_size[0]
72      wws1 = RAM_word_size[1]
73      we00 = RAM_write_enable
74      we10 = RAM_write_enable & (wws0 | wws1)
75      we01 = RAM_write_enable & wws1
76      we11 = we01 & wws0
77      RAM_write_enablel = [we00, we11, we11, we11, we11, we01,
        ↪  we01, we10]
```

Modules

```
80      vall = []
81      for i in range(8):
82          vall.append(RAM(
83              addr_size,
                ↪ # address size
84              8,
                ↪ # word size
85              addrl[i],
                ↪ # read_addr
86              multimux(waddr_remain, cycle(i, RAM_write_enablel)),
                ↪ # write_enable
87              waddrl[i],
                ↪ # write_address
88              multimux(waddr_remain, cycle(i, RAM_write_datal))
                ↪ # write_data
89          ))
```

Sommaire

1 ISA

2 ALU

3 RAM

4 Programme

Horloge