Microprocesseur RISC-V, horloge

Aghilas BOUSSAA, Marius CAPELLI, Olivier HENRY, Paul WANG

28 janvier 2025

Sommaire

Sommaire

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

Carry-lookahead adder

```python
def propgen(aandb:list[Variable], aorb:list[Variable]) -> tuple[list[Variable], list[Variable]]:
  '''returns the list of propagates and generates for blocs of aligned 2**k bits'''
  n = len(aorb)
  if n == 1:
    return aorb, aandb
  m = n//2
  pd, gd = propgen(aandb[m:n], aorb[m:n])
  pg, gg = propgen(aandb[0:m], aorb[0:m])
  p = pg + pd
  g = gg + gd
  p.append(pd[-1] & pg[-1])
  g.append(gd[-1] | (pd[-1] & gg[-1]))
  return p, g
```

Réutilisation des circuits

```python
n = a.bus_size
# ensures that b2 = b when add/xor/and/or and b2 = -b when sub/slt/sltu
isnotsub = funct32 | (~funct75 & ~funct31)
b2 = Mux(isnotsub, ~b, b)
aorb = a | b2
aandb = a & b2
axorb = a ^ b2
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)
L, sll_ret = sll(a, b)
return multimux([funct30, funct31, funct32], [
  aplusb,
  sll_ret,
  slt(a, b, aplusb),
  sltu(a, b, aplusb),
  axorb,
  sral(a, b, funct75),
  aorb,
```

Multiplicateur peu intelligent

```python
def multiplier(L:list[Variable], b):
    res = []
    n = len(L)
    for i in range(n):
        res.append(Mux(b[i], Constant("0"*n), L[i]))
    for i in range(log2i(n)-1):
        res2 = []
        for j in range(0, n>>i, 2):
            and_tmp = res[j] & res[j+1]
            or_tmp = res[j] | res[j+1]
            xor_tmp = res[j] ^ res[j+1]
            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])
        res = res2
    return res[0]
```

Sommaire

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

```
addrl.append(Mux(addr_remain0 | ar1_or_ar2, addr, addrplus)) # if addr_remain >= 100 then addrplus else addr
addrl.append(Mux(ar1_or_ar2, addr, addrplus)) # if addr_remain >= 010 then addrplus else addr
addrl.append(Mux(addr_remain2 | (addr_remain0 & addr_remain1), addr, addrplus)) # if addr_remain >= 110 then addrplus else addr
addrl.append(Mux(addr_remain2, addr, addrplus)) # if addr_remain >= 001 then addrplus else addr
addrl.append(Mux(addr_remain2 & (addr_remain1 | addr_remain0), addr, addrplus)) # if addr_remain >= 101 then addrplus else addr
addrl.append(Mux(ar1_and_ar2, addr, addrplus)) # if addr_remain >= 011 then addrplus else addr
addrl.append(Mux(ar1_and_ar2 & addr_remain0, addr, addrplus)) # if addr_remain >= 111 then addrplus else addr
addrl.append(addr)
```

Données à écrire

```python
# Déterminer ce que l'on écrit
RAM_write_datal = cycle(1, [RAM_write_data[i:i+8] for i in range(0, 64, 8)][::-1])

# Déterminer où on écrit (c'est pareil que pour voir ce que l'on écrit)
wws0 = RAM_word_size[0]
wws1 = RAM_word_size[1]
we00 = RAM_write_enable
we10 = RAM_write_enable & (wws0 | wws1)
we01 = RAM_write_enable & wws1
we11 = we01 & wws0
RAM_write_enablel = [we00, we11, we11, we11, we11, we01, we01, we10]
```

Modules

```python
    vall = []
    for i in range(8):
        vall.append(RAM(
            addr_size,  # address size
            8,  # word size
            addrl[i],  # read_addr
            multimux(waddr_remain, cycle(i, RAM_write_enablel)),  # write_enable
            waddrl[i],  # write_address
            multimux(waddr_remain, cycle(i, RAM_write_datal))  # write_data
        ))
```

Sommaire

Fonctionnalités

- Compte les secondes, minutes, heures, jours (dans le mois), mois, années
- Ne compte pas les jours dans la semaine
- Ne compte pas les changements d'heure

Programme

```
verification: # vérifie si on change de seconde
    gtck x1
    bne x5, x1, ajout_seconde
    jal x0, verification

ajout_seconde:
    addi x5, x1, 0
    # si on est à 60s
    gsec x1
    addi  x2, x0, 60
    beq x2, x1 , ajout_minute
    addi x1, x1, 1
    ssec x1
    jal x0, verification
```