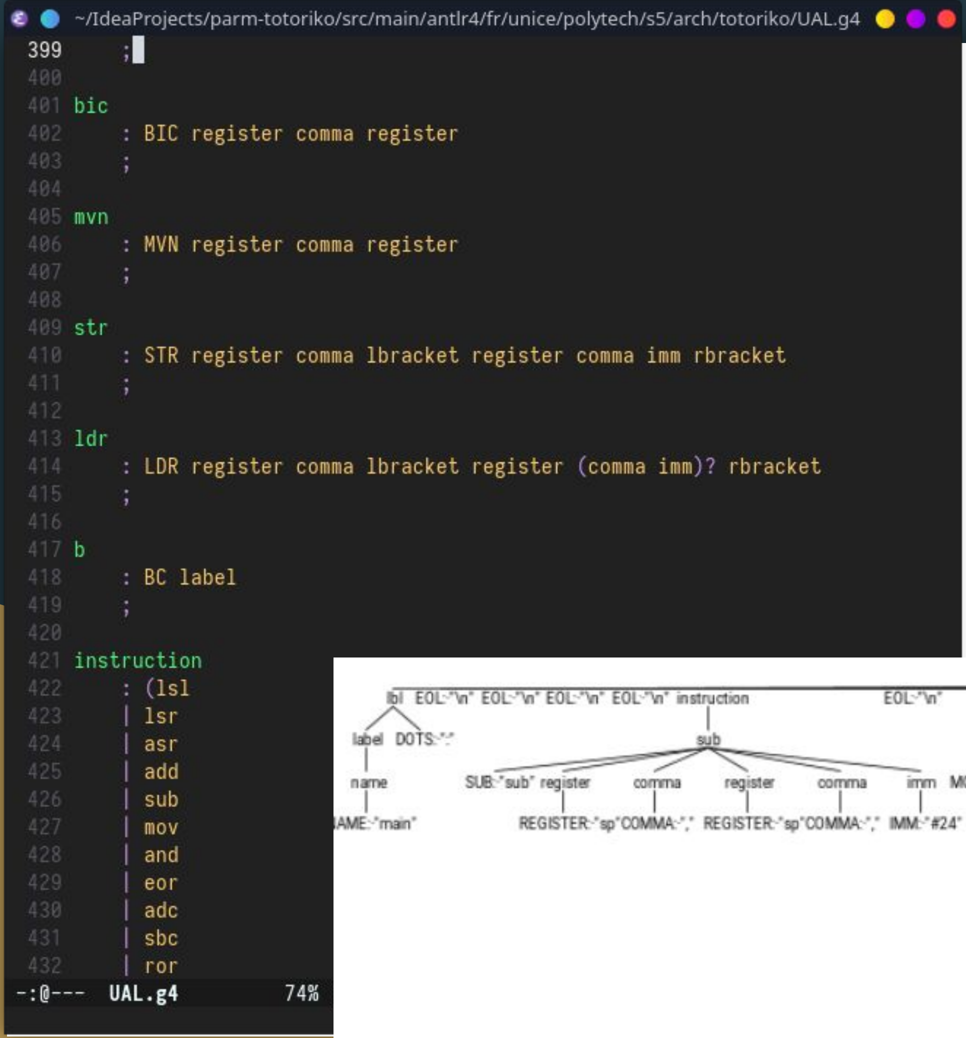
Projet P-ARM

Groupe : Totoriko.



~/IdeaProjects/parm-totoriko/src/main/antlr4/fr/unice/polytech/s5/arch/totoriko/UAL.g4

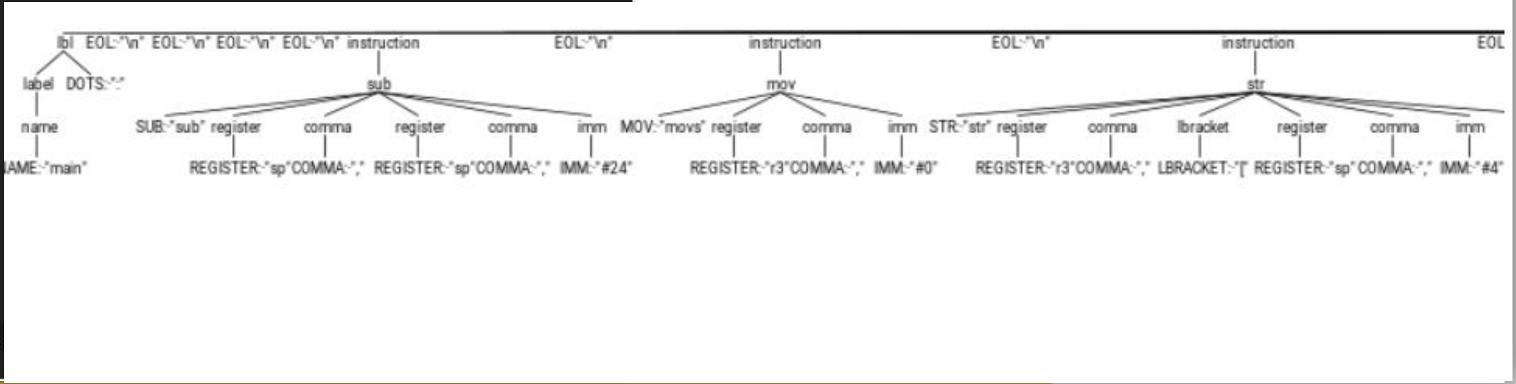
```
    ;

bic
    : BIC register comma register
    ;

mvn
    : MVN register comma register
    ;

str
    : STR register comma lbracket register comma imm rbracket
    ;

ldr
    : LDR register comma lbracket register (comma imm)? rbracket
    ;

b
    : BC label
    ;

instruction
    : (lsl
    | lsr
    | asr
    | add
    | sub
    | mov
    | and
    | eor
    | adc
    | sbc
    | ror
```

```
-:@---  UAL.g4        74%
```

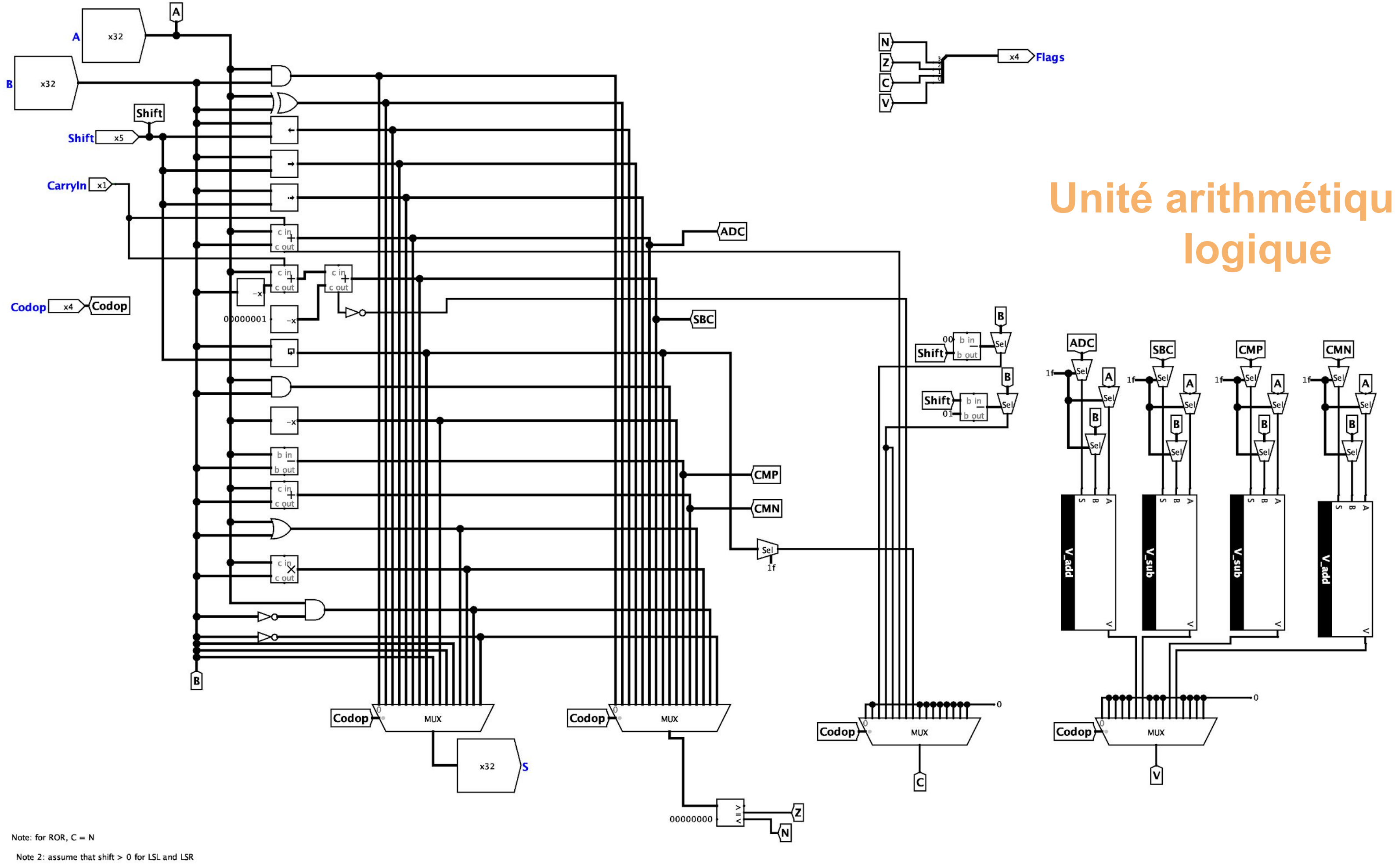
Parseur

ANTLR (Another Tool For Language Recognition)

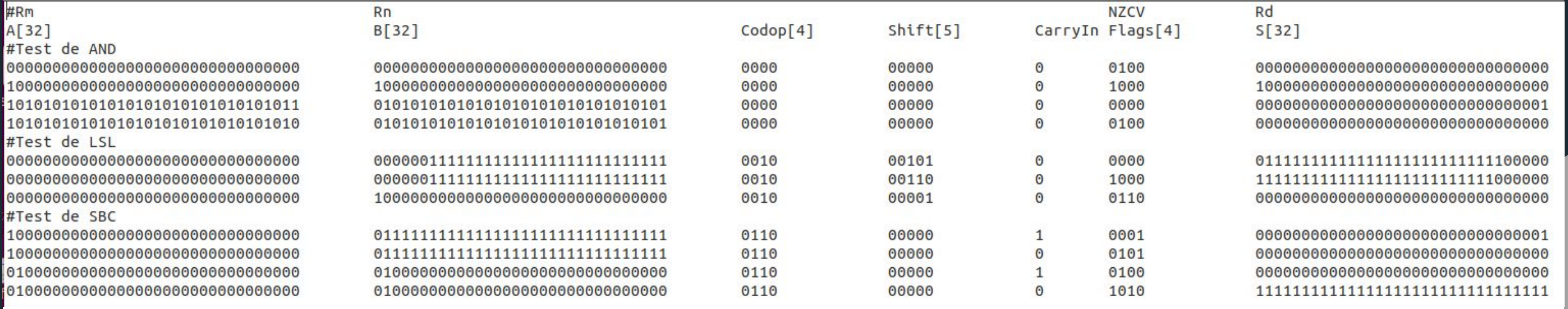
Java API


lbl EOL:"\n" EOL:"\n" EOL:"\n" EOL:"\n" instruction EOL:"\n" instruction EOL:"\n" instruction EOL
label DOTS:":"
name
AME:"main"
sub
SUB:"sub" register comma register comma imm
REGISTER:"sp" COMMA:"," REGISTER:"sp" COMMA:"," IMM:"#24"
mov
MOV:"movs" register comma imm
REGISTER:"r3" COMMA:"," IMM:"#0"
str
STR:"str" register comma lbracket register comma imm
REGISTER:"r3" COMMA:"," LBRACKET:"[" REGISTER:"sp" COMMA:"," IMM:"#4"

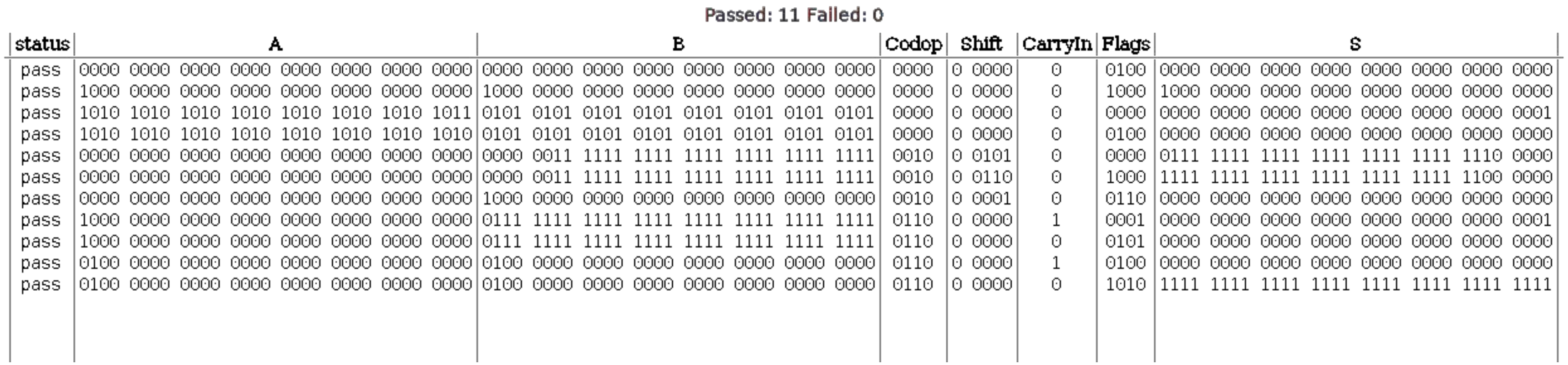
Unité arithmétique et logique



Tests unitaires:



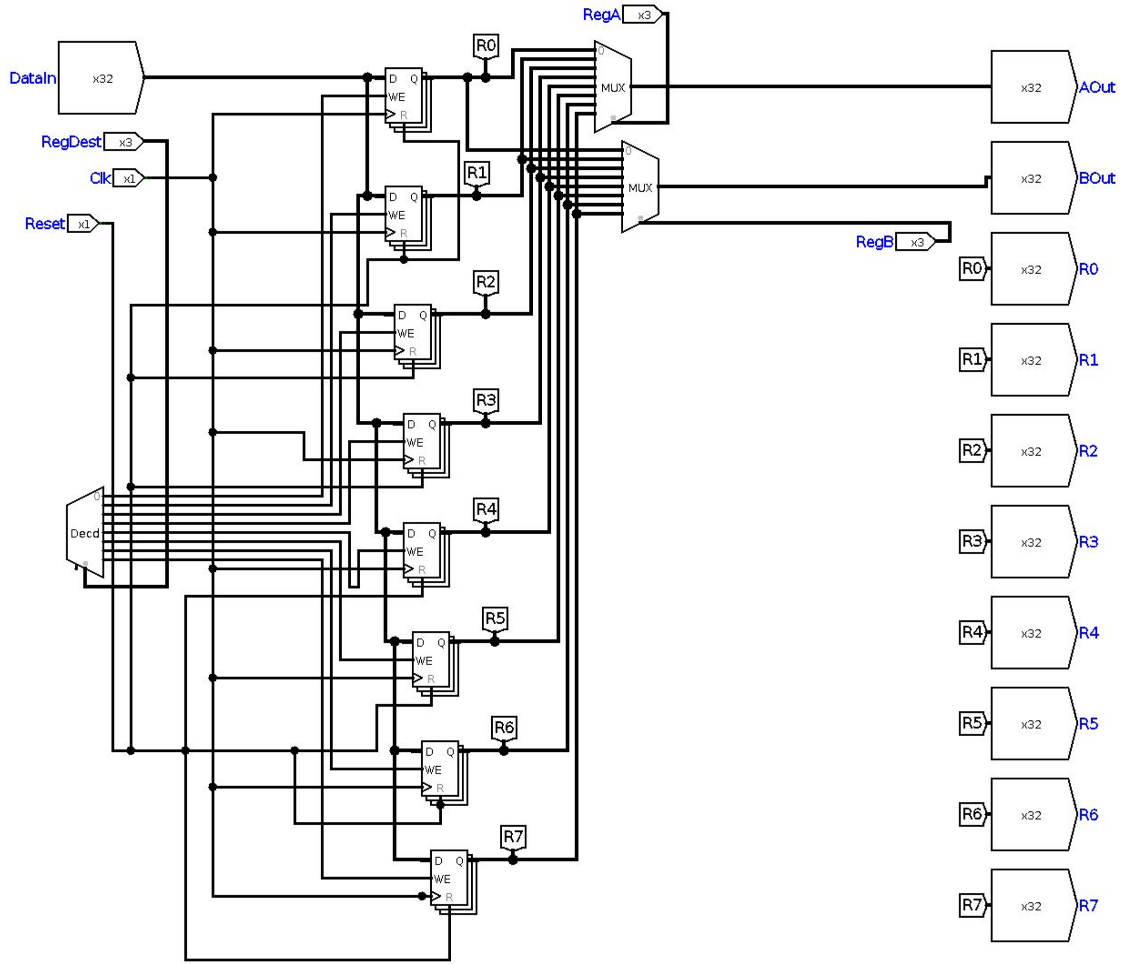
```
#Rm                                     Rn                                                                              NZCV            Rd
A[32]                                   B[32]                                   Codop[4]        Shift[5]        CarryIn Flags[4]        S[32]
#Test de AND
00000000000000000000000000000000        00000000000000000000000000000000        0000            00000           0       0100            00000000000000000000000000000000
10000000000000000000000000000000        10000000000000000000000000000000        0000            00000           0       1000            10000000000000000000000000000000
10101010101010101010101010101011        01010101010101010101010101010101        0000            00000           0       0000            00000000000000000000000000000001
10101010101010101010101010101010        01010101010101010101010101010101        0000            00000           0       0100            00000000000000000000000000000000
#Test de LSL
00000000000000000000000000000000        00000011111111111111111111111111        0010            00101           0       0000            01111111111111111111111111100000
00000000000000000000000000000000        00000011111111111111111111111111        0010            00110           0       1000            11111111111111111111111111000000
00000000000000000000000000000000        10000000000000000000000000000000        0010            00001           0       0110            00000000000000000000000000000000
#Test de SBC
10000000000000000000000000000000        01111111111111111111111111111111        0110            00000           1       0001            00000000000000000000000000000001
10000000000000000000000000000000        01111111111111111111111111111111        0110            00000           0       0101            00000000000000000000000000000000
01000000000000000000000000000000        01000000000000000000000000000000        0110            00000           1       0100            00000000000000000000000000000000
01000000000000000000000000000000        01000000000000000000000000000000        0110            00000           0       1010            11111111111111111111111111111111
```



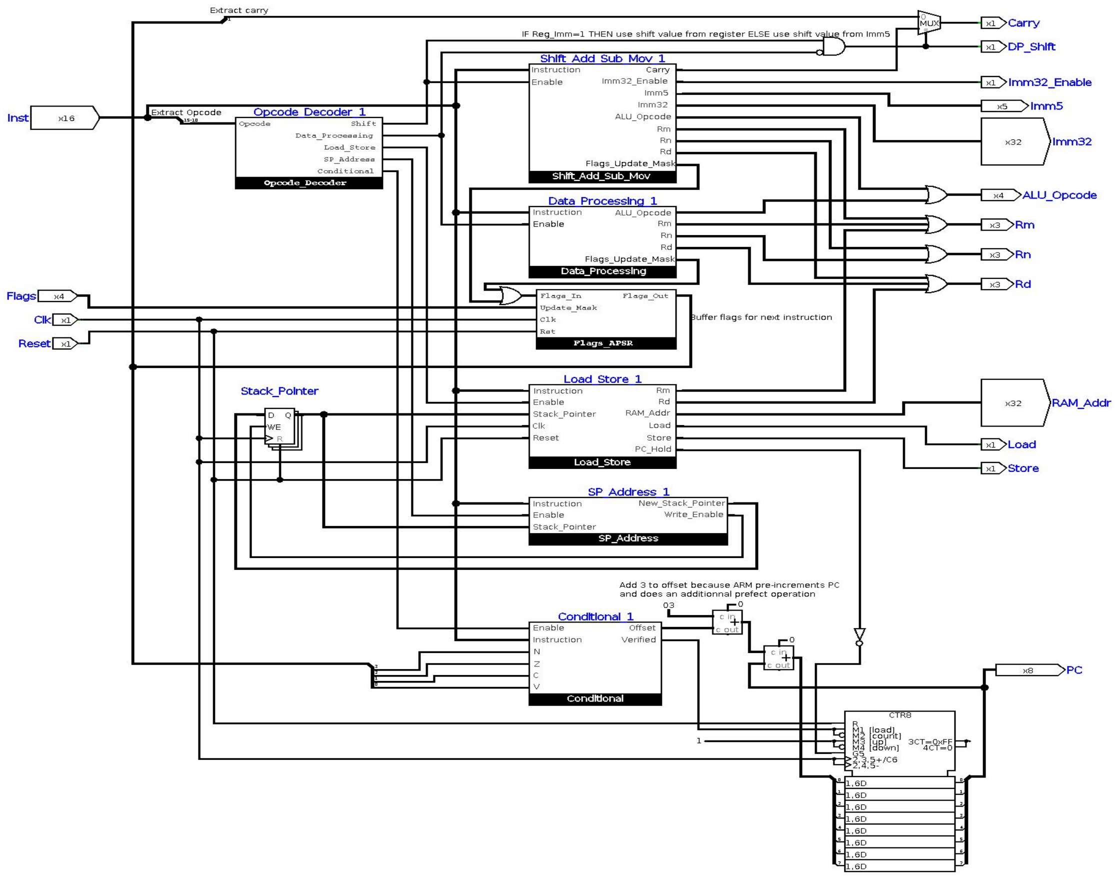
Passed: 11 Failed: 0

status	A	B	Codop	Shift	CarryIn	Flags	S
pass	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000	0000	0 0000	0	0100	0000 0000 0000 0000 0000 0000 0000 0000
pass	1000 0000 0000 0000 0000 0000 0000 0000	1000 0000 0000 0000 0000 0000 0000 0000	0000	0 0000	0	1000	1000 0000 0000 0000 0000 0000 0000 0000
pass	1010 1010 1010 1010 1010 1010 1010 1011	0101 0101 0101 0101 0101 0101 0101 0101	0000	0 0000	0	0000	0000 0000 0000 0000 0000 0000 0000 0001
pass	1010 1010 1010 1010 1010 1010 1010 1010	0101 0101 0101 0101 0101 0101 0101 0101	0000	0 0000	0	0100	0000 0000 0000 0000 0000 0000 0000 0000
pass	0000 0000 0000 0000 0000 0000 0000 0000	0000 0011 1111 1111 1111 1111 1111 1111	0010	0 0101	0	0000	0111 1111 1111 1111 1111 1111 1110 0000
pass	0000 0000 0000 0000 0000 0000 0000 0000	0000 0011 1111 1111 1111 1111 1111 1111	0010	0 0110	0	1000	1111 1111 1111 1111 1111 1111 1100 0000
pass	0000 0000 0000 0000 0000 0000 0000 0000	1000 0000 0000 0000 0000 0000 0000 0000	0010	0 0001	0	0110	0000 0000 0000 0000 0000 0000 0000 0000
pass	1000 0000 0000 0000 0000 0000 0000 0000	0111 1111 1111 1111 1111 1111 1111 1111	0110	0 0000	1	0001	0000 0000 0000 0000 0000 0000 0000 0001
pass	1000 0000 0000 0000 0000 0000 0000 0000	0111 1111 1111 1111 1111 1111 1111 1111	0110	0 0000	0	0101	0000 0000 0000 0000 0000 0000 0000 0000
pass	0100 0000 0000 0000 0000 0000 0000 0000	0100 0000 0000 0000 0000 0000 0000 0000	0110	0 0000	1	0100	0000 0000 0000 0000 0000 0000 0000 0000
pass	0100 0000 0000 0000 0000 0000 0000 0000	0100 0000 0000 0000 0000 0000 0000 0000	0110	0 0000	0	1010	1111 1111 1111 1111 1111 1111 1111 1111

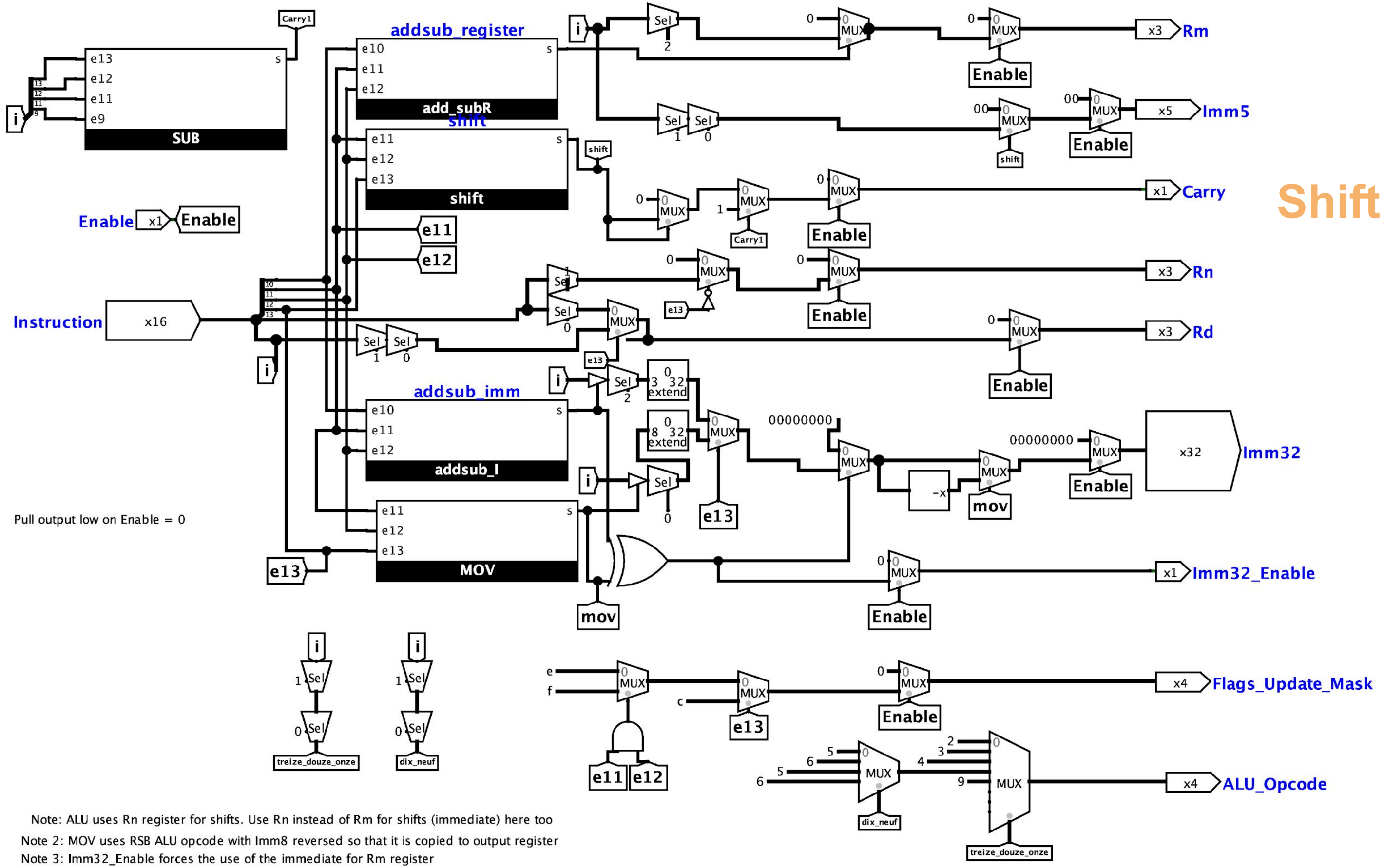
Banc de registre



Contrôleur


Extract carry
IF Reg_Imm=1 THEN use shift value from register ELSE use shift value from Imm5
MUX
Carry
DP_Shift
Shift_Add_Sub_Mov_1
Instruction
Enable
Carry
Imm32_Enable
Imm5
Imm32
ALU_Opcode
Rm
Rn
Rd
Flags_Update_Mask
Shift_Add_Sub_Mov
Imm32_Enable
Imm5
Imm32
Inst
x16
Extract Opcode
Opcode_Decoder_1
Opcode
Shift
Data_Processing
Load_Store
SP_Address
Conditional
Opcode_Decoder
Data_Processing_1
Data_Processing
ALU_Opcode
Rm
Rn
Rd
Flags
Clk
Reset
Flags_In
Update_Mask
Flags_Out
Flags_APSR
Buffer flags for next instruction
Stack_Pointer
Load_Store_1
Stack_Pointer
RAM_Addr
Load
Store
PC_Hold
Load_Store
RAM_Addr
SP_Address_1
New_Stack_Pointer
Write_Enable
SP_Address
Add 3 to offset because ARM pre-increments PC and does an additionnal prefect operation
03
Conditional_1
Offset
Verified
N
Z
C
V
Conditional
PC
CTR8
M1 [load]
M2 [count]
M3 [up]
M4 [down]
G5
3CT=0xFF
4CT=0
2,3,5+/C6
2,4,5-
1,6D

Shift, add, sub, mov



Pull output low on Enable = 0

Note: ALU uses Rn register for shifts. Use Rn instead of Rm for shifts (immediate) here too

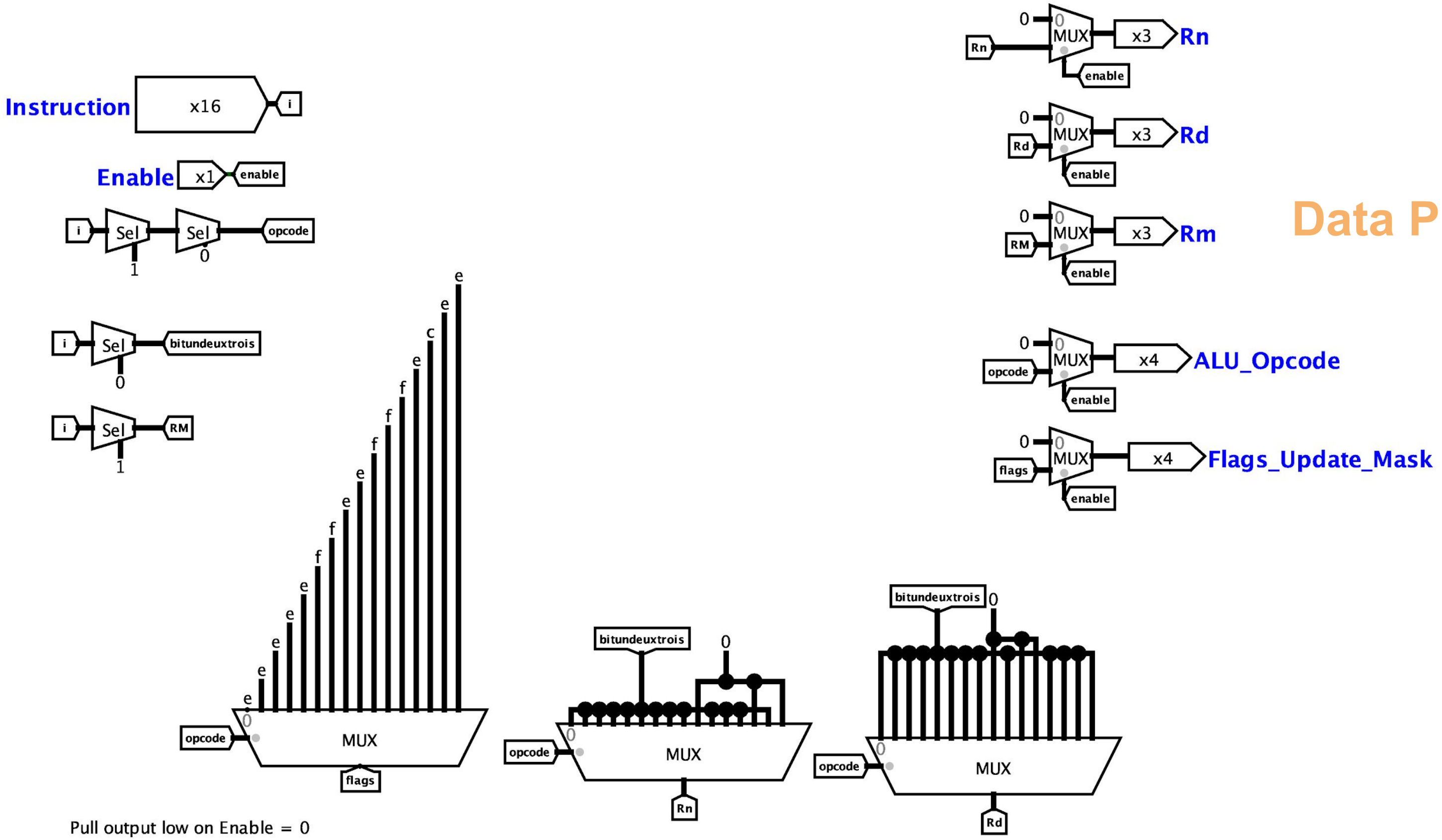
Note 2: MOV uses RSB ALU opcode with Imm8 reversed so that it is copied to output register

Note 3: Imm32_Enable forces the use of the immediate for Rm register

Enable carry for SUB since it is reversed in ALU

ALU will use Imm32 instead of Rm

Data Processing



Pull output low on Enable = 0

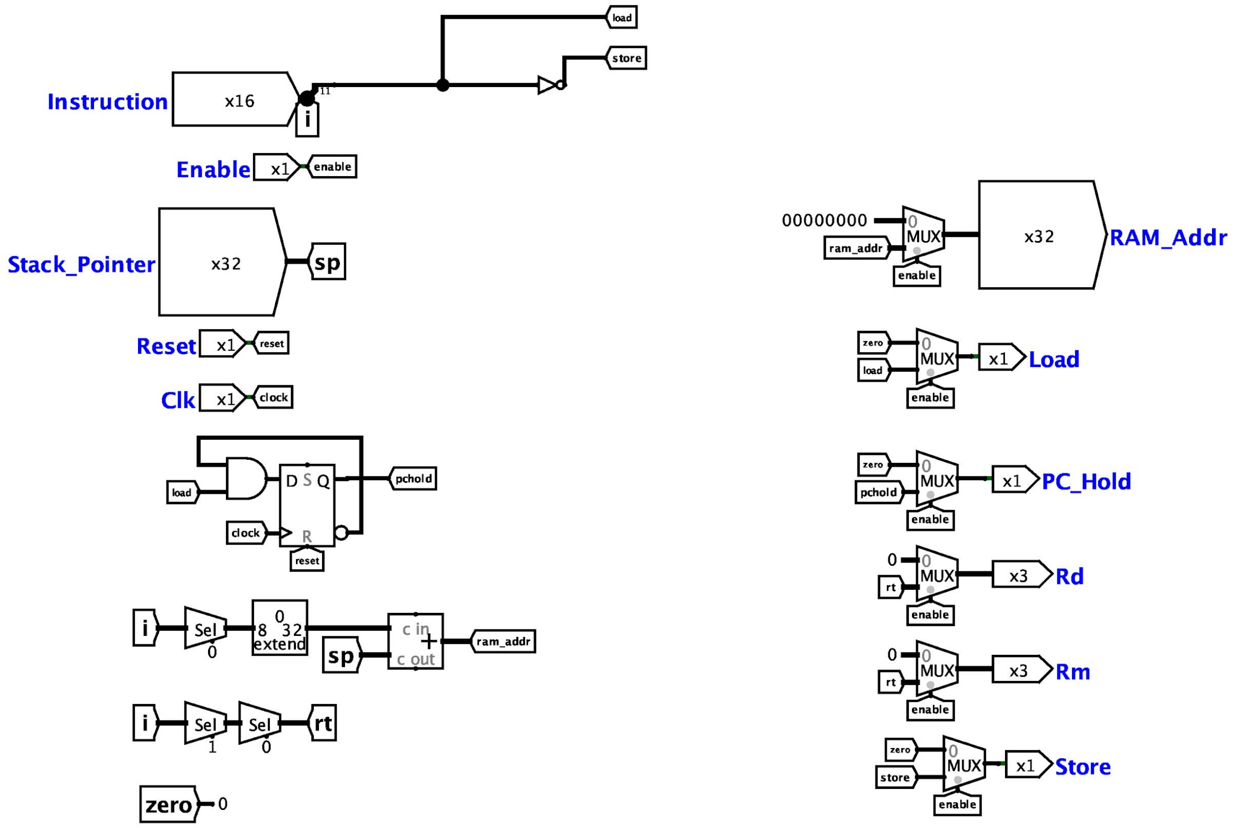
Note: instructions that does not save the result will still have the second operand as the destination register, the ALU will copy the second register to the destination register

Note 2: RSB instruction has Rn as 1st operand.

MUL instruction has Rn as 1st operand and Rdm as 2nd operand

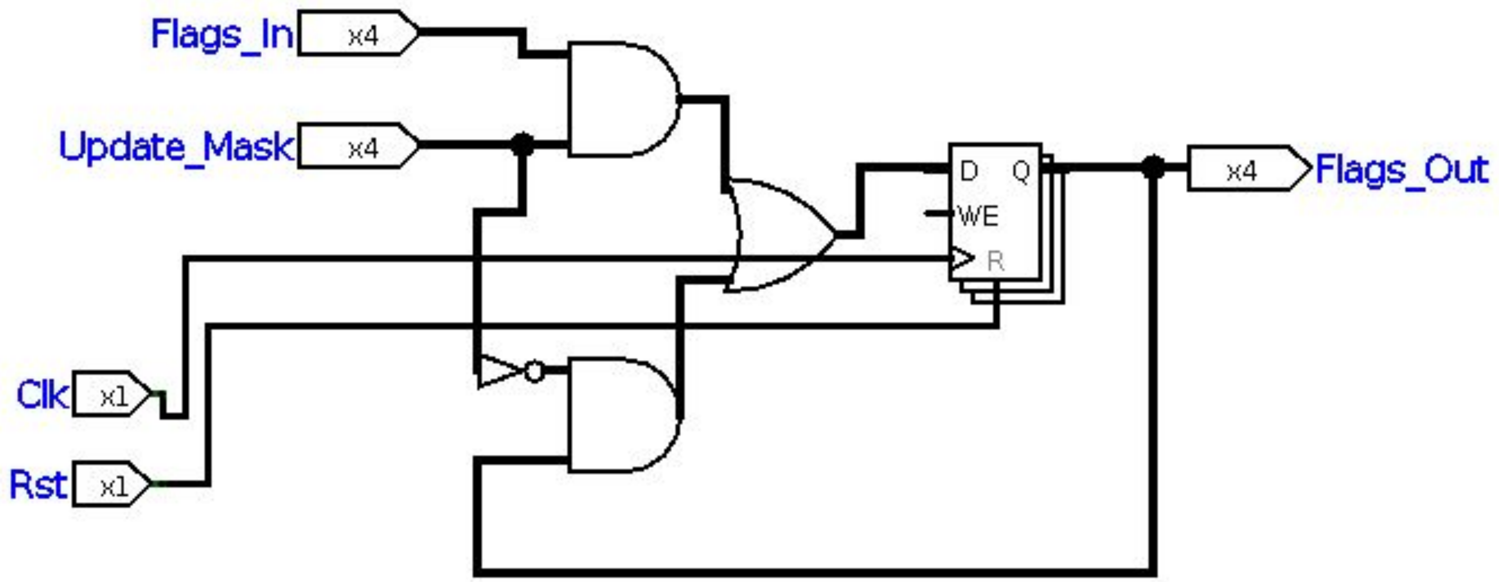
For simplification purposes, Rm is used for 1st operand both here and in the ALU.

Load Store

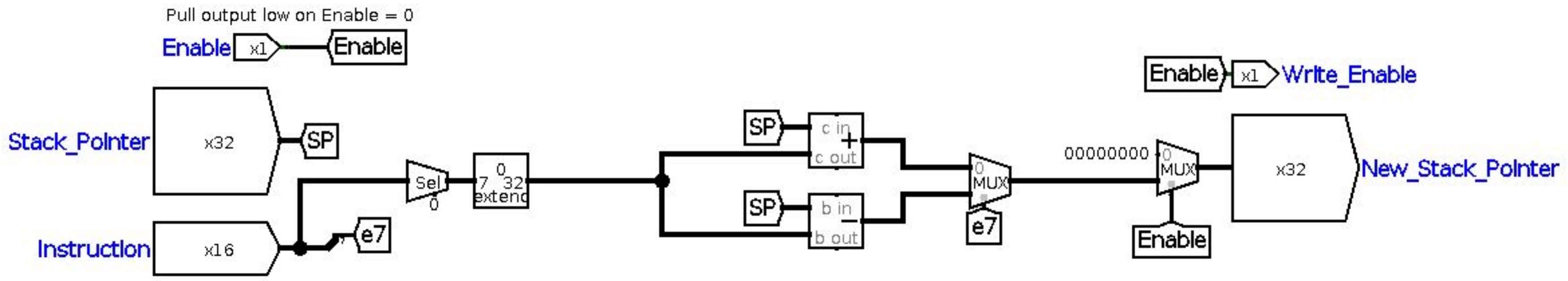


Delay load and hold PC for 1 cycle for the RAM to send out requested data

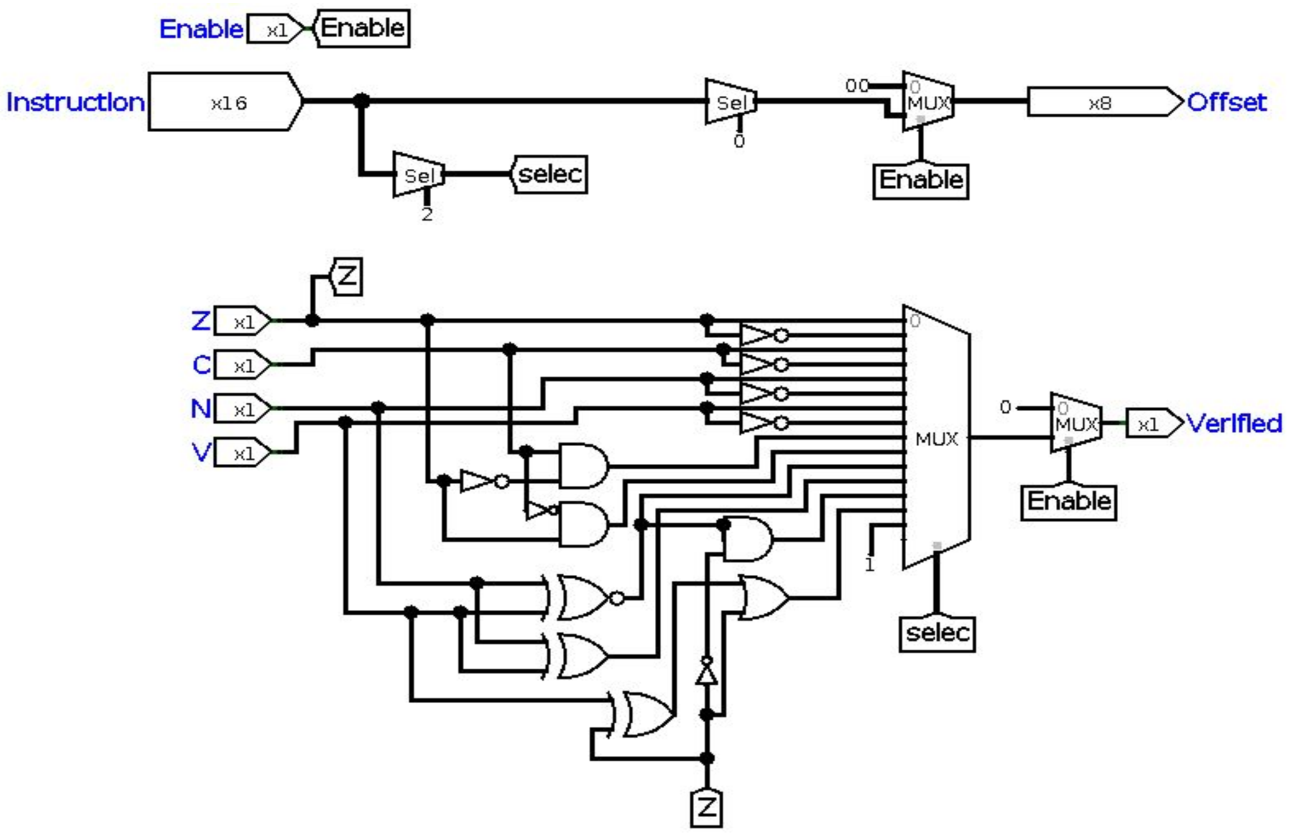
Flags_APSR



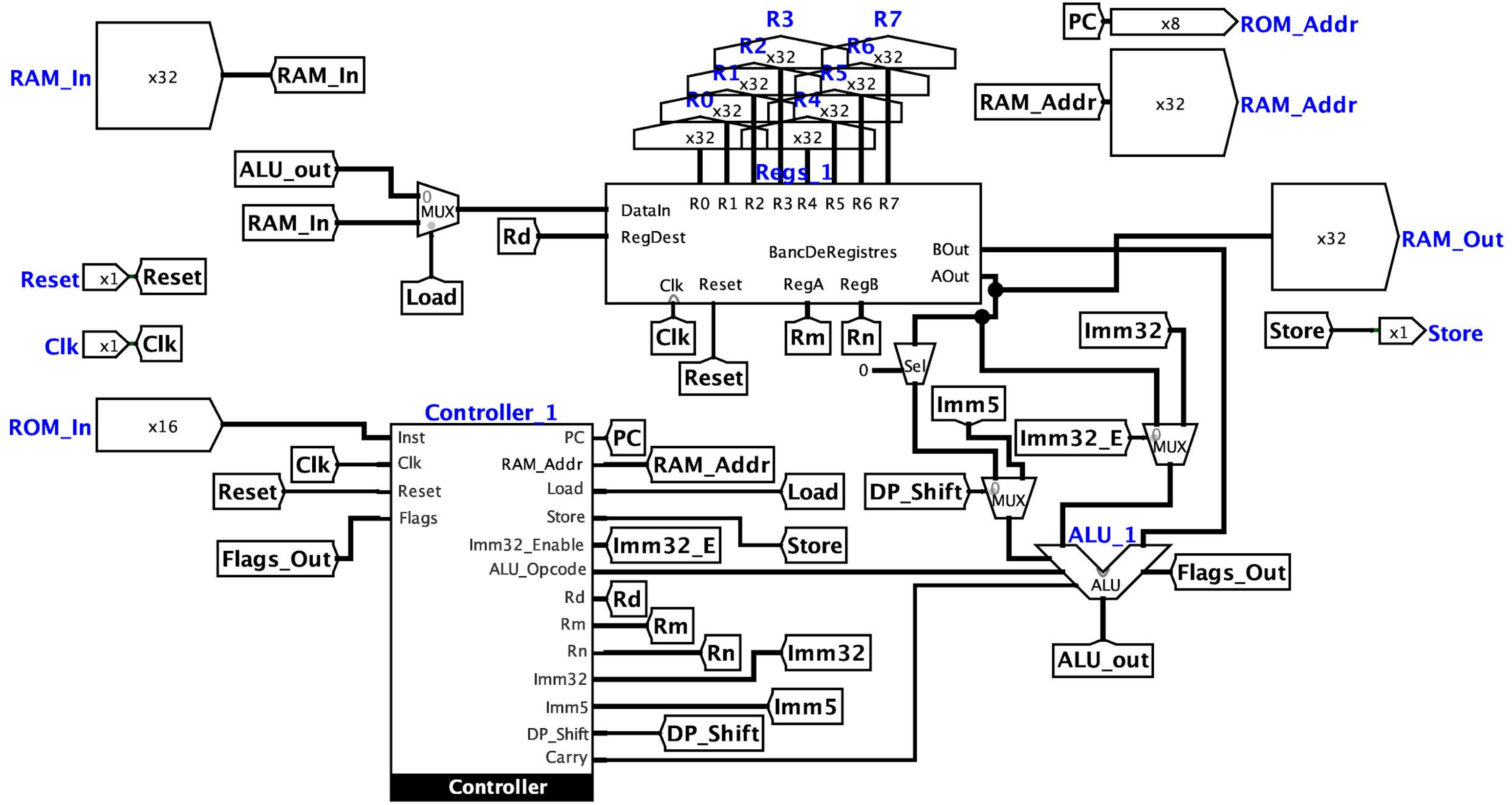
SP_Address


Pull output low on Enable = 0
Enable
x1
Enable
Stack_Pointer
x32
SP
Instruction
x16
e7
Sel
0
7
32
extend
SP
c in
c out
SP
b in
b out
MUX
e7
00000000
MUX
Enable
Enable
x1
Write_Enable
x32
New_Stack_Pointer

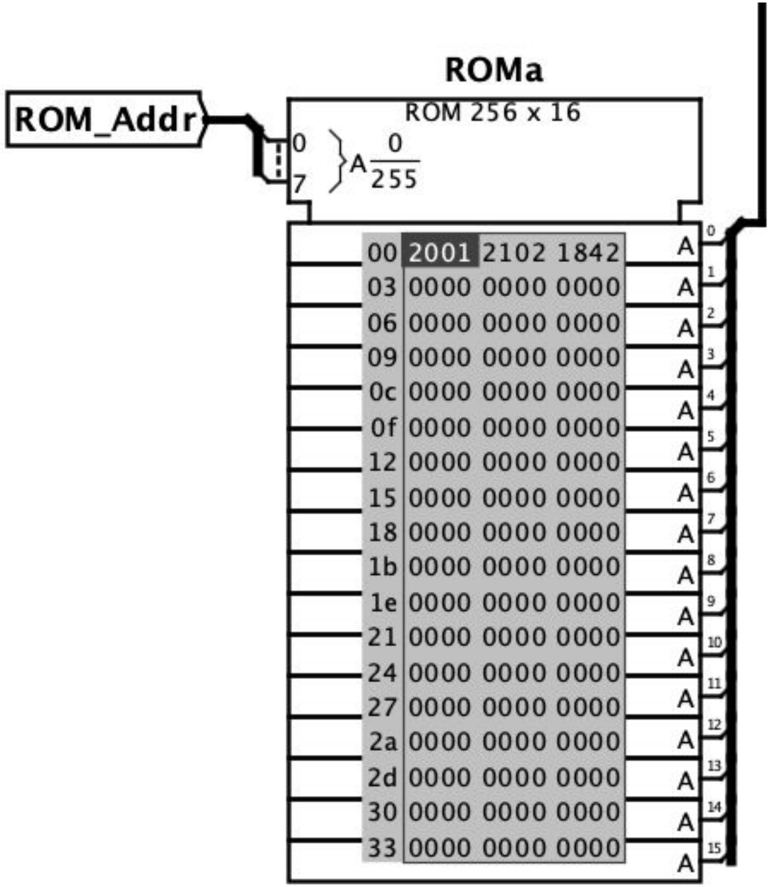
Conditional



Processeur


RAM_In
x32
RAM_In
ALU_out
RAM_In
MUX
Load
Rd
Reset
x1
Reset
Clk
x1
Clk
R3
R7
R2
R6
R1
R5
R0
R4
x32
Regs_1
DataIn
RegDest
R0 R1 R2 R3 R4 R5 R6 R7
BancDeRegistres
BOut
AOut
Clk
Reset
RegA
RegB
Clk
Reset
Rm
Rn
PC
x8
ROM_Addr
RAM_Addr
x32
RAM_Addr
x32
RAM_Out
Store
x1
Store
Imm32
Sel
0
Imm5
Imm32_E
MUX
DP_Shift
MUX
ALU_1
ALU
Flags_Out
ALU_out
ROM_In
x16
Controller_1
Inst
Clk
Reset
Flags
PC
RAM_Addr
Load
Store
Imm32_Enable
ALU_Opcode
Rd
Rm
Rn
Imm32
Imm5
DP_Shift
Carry
Controller
Clk
Reset
Flags_Out
PC
RAM_Addr
Load
Imm32_E
Store
Rd
Rm
Rn
Imm32
Imm5
DP_Shift

ROM



```
00 2001 2102 1842
03 0000 0000 0000
06 0000 0000 0000
09 0000 0000 0000
0c 0000 0000 0000
0f 0000 0000 0000
12 0000 0000 0000
15 0000 0000 0000
18 0000 0000 0000
1b 0000 0000 0000
1e 0000 0000 0000
21 0000 0000 0000
24 0000 0000 0000
27 0000 0000 0000
2a 0000 0000 0000
2d 0000 0000 0000
30 0000 0000 0000
33 0000 0000 0000
```