Extins

Extins (extract): `Rx=Ry[j:i]` (j > i)
(insert): `Rx[j:i]=Ry` (j > i)

Examples: R1=R5[14:3]
R1[1:0]=R2

Fifo

Fifo- managing supporter.

Examples: (R0,R1) = fifo_wr(R0,R1)
(R0,R1) = fifo_rd(R0,R1)

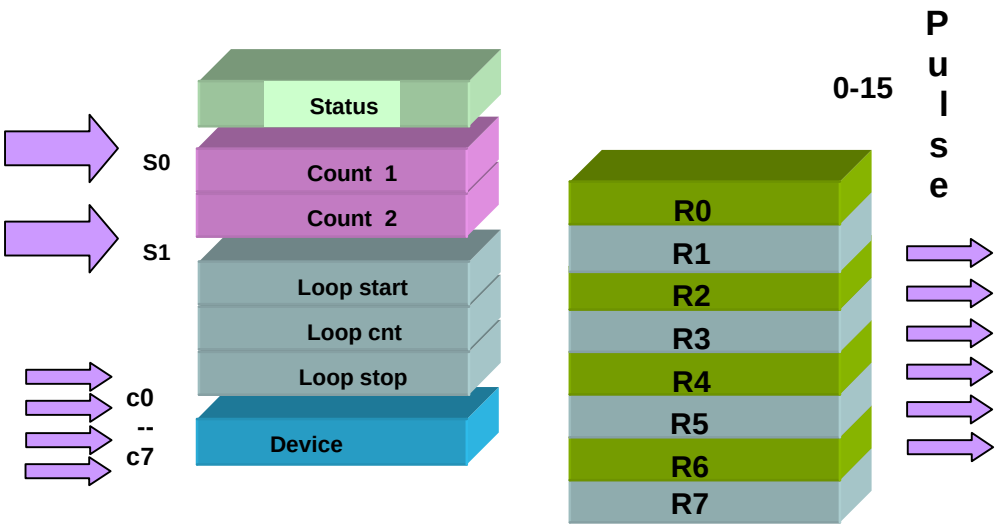
Decimal Binary & Hexadecimal format

Decimal: 2 formats- 1.**yy'dzz** **yy**- the number of bits. **z**-0-9 **d**-decimal
2.**zz** simply the number.

Binary: yy'bzzzzzz **yy**- the number of bits. **z**-0-1 **b-** Binary

Hexadecimal: yy'hzzzz **yy** - the number of bits. **z**-0-f **h-** Hexadecimal

Simplified programmer model



Memory – reading and writing to the memory

Writhing to memory : `RAM[R6/R7/++/add]=Rx/imd`

Registers:
RAM[R6] = R0 — RAM[R6++] = R4
RAM[R7] = R5_l — RAM[R6++] = R4 < pulse->p3 >
RAM[R7++] = R2 <R1=s1> — RAM[100+R7] = R5_l

Immediate address:
RAM[200]= R7
RAM[400]= R3_h
RAM[200]= R7

Immediate data:
RAM[R6++] = 16'h1000100010001001
RAM[R6] = 233
RAM[R7++] = 16'ha4bf

Reading from memory : `Rx=RAM[R6/R7/++/add]`

Registers:
R5 =RAM[R6] — R2=RAM[R7++] < pulse->p3 >
R5_l =RAM[R7] — R2_h=RAM[R6++] < pulse -> p1 >
R0=RAM[100+R6] — R3_h=RAM[10+R6

Immediate address: R7= RAM[200]

Wait load: This is a load command depending on the occurrence of a control signal.
`wait_load !cx/cx/!xxx_flag/xxx_flag Rx=sx/RAM[R6++/R7++]`

Examples:
L_send_data_c0c0, loop L_s L_s R5
L_s, wait_load c0 R0=RAM[R7++] pulse->p0 breakif !c0 // in a loop

Wait store: This command is used to pipeline output data into the ram via a register.
`wait_store !cx/cx RAM[R6++/R7++] = Rx = sx breakif !c/cx`

Examples:
wait_store !c1 RAM[R6++] = R1 = s0 breakif !c2 pulse->p2
wait_store !c5 RAM[R7++] = R5 = s0
wait_store !c1 RAM[R6++] = R1 = s0 pulse->p2

Registers: loading and moving registers.

Registers: `Rx=Ry`
R1=R3
R4=R7 pulse->p5

Immediate data: `Rx=imd`
R6 = 4523
R2= 16'h0a1b
R1 = 16'b0101110001110011

Push Pop: all the registers can be pushed or popped to/from shadow space.
`push/pop Rx/flags/loop_start/loop_stop/loop_cnt`

Examples:
push R0
push R0 flags R3 R4
pop flags R3 R4 R5

Alu commands: numerical and logical operations on registers and numbers.

Arithmetic and logical operations:

The operations that can be used on registers as well on Immediate values are: + plus ? - minus ? | or \ ^ xor \ & and.

Registers: R4 = -R3 + R2 , R3 = R3 - R2 `Rw = Rx OP Ry`
R6 = R6 & R7, R6 = R3 ^ R7 , R6 = R6 | R7

Immediate data*: R4= -R3 + 1, R4=2-R3 , `Rw = Rx/imd OP Ry/imd`
R3=R2&12, R5=R2^12 , R1=-R1+5

Shift operation: the right value of this operation is either a register or an immediate value. Keep in mind that only the 4 lsb are use in either ways.

Example: R2=R3>>5 `Rx=Ry >>/<< Rw/imd`
R2=R2>>R4

* Immediate data in the alu is up to +-128

External devices: device s1 s0 and pulse.

Device : 16 bit device divided in to 4 devices (0-3)

Examples: device_0 ->4'b1011 < R2 = s1 >< pulse -> p1 > `device_x->4'bxxxx`
device_3 ->4'bx111 < R2=RAM[R7] >< pulse -> p2 >

Pulse: 15 pulse lines (0-14) can be used independently or as an addition to other commands

Example: pulse -> p10 `pulse -> px`

External source: 2 external sources of 16 bits s0 and s1 can be sampled into the core as exclusive command or in addition to other commands

Example: R2=s1 `Rx=s1/s0`

Lin Search

Lin Search:

? ? This command is intended for use under a loop for pipelining data from the RAM.
? ? Usually the first RAM[address] should be prepared in the temp register before entering the loop body.
? ? When using > < operations the values are considered unsigned.
? ? When using a part of a register Rx<[x:y]> the lin search treatment to this value is as an undersigned number.
? ? After the search is complete the scs16 automatically reduce 2 steps from R7 to the place in the ram that the search value was found.

Examples:
breakif R5 == R2 = RAM[R7++]
breakif R3[9:4] ==R1 = RAM[R7+=R5]
breakif R5<R2 = RAM[R6+=R2]

Labels

Using Labels in loops: every label must be in the following syntax**- L_XXX,**

```
L_a  L_b  number/register*
```

Examples:

loop L_a L_b R5 — loop L_start L_start 100

loop L_start L_end R2 — loop L_a L_a 10'b0110111001

Wait label: used to hold the flow of the code.

```
wait <!> (Rx[ y] | Cx)
```

Examples: wait !c3 wait R3[14]

*L_a and L_b can be identical labels

If else

If :can came with any expression and with or with out a negative sigh

expression: R1>R2 **or** > < == != **or** R3[4] **or** cx*

```
if (<!>expression) {
        < Commands >
}
or
if (<!>expression) L_xxx        //jumping to the label
```

elsif: is valid to use only with a register bit **or** cx*.

```
elsif (bit / cx) {              // in the form R3[4]
        < Commands >
}
```

else: is used with no condition.

Example:

```
if (R2>R3) {
        < Commands >
}
elsif (R3[4]){
        < Commands >
}
elsif (R3[2]){
        < Commands >
        ….
else {
        < Commands >
}
```

***c external condition is 0-7.

goto / gosub

Using goto and gosub:

```
goto label or R5          |    gosub labe
goto L_label        |    gosub L_subName
```

Example:

```
gosub L_subName                 // goto subroutine
….                              // <--after the subroutine returning to this point
….
L_subName,   PUSH R3, R4        // keeping the registers values
             Do work …
             R5 = Result        // Kept in the main context.
             POP R3, R4
             Return             // every subroutine must end with return command
```