Real Time Embedded Systems

"System On Programmable Chip"
NIOSII Custom Instruction

rene.beuchat@epfl.ch
LAP/ISIM/IC/EPFL
Chargé de cours

rene.beuchat@hesge.ch
LSN/hepia
Prof. HES

Contents

- Introduction
- Custom Instructions on NIOS II
- Hardware
- Software access from C

References :

- *http://www.altera.com/literature/ug/ug_nios2_custom_instruction.pdf*
  (NIOS II Custom Instruction, User Guide, Altera January 2011)

Introduction

A processor has an initial Instruction Set Architecture defined by the processor design architect.

The instruction set is done to be the more efficient in general cases but not for special cases.

Processors as DSP (Digital Signal Processor) have specialized instructions and not good for general purpose applications.

With softcore processor, it’s possible to add instructions on the basic set available.

Introduction

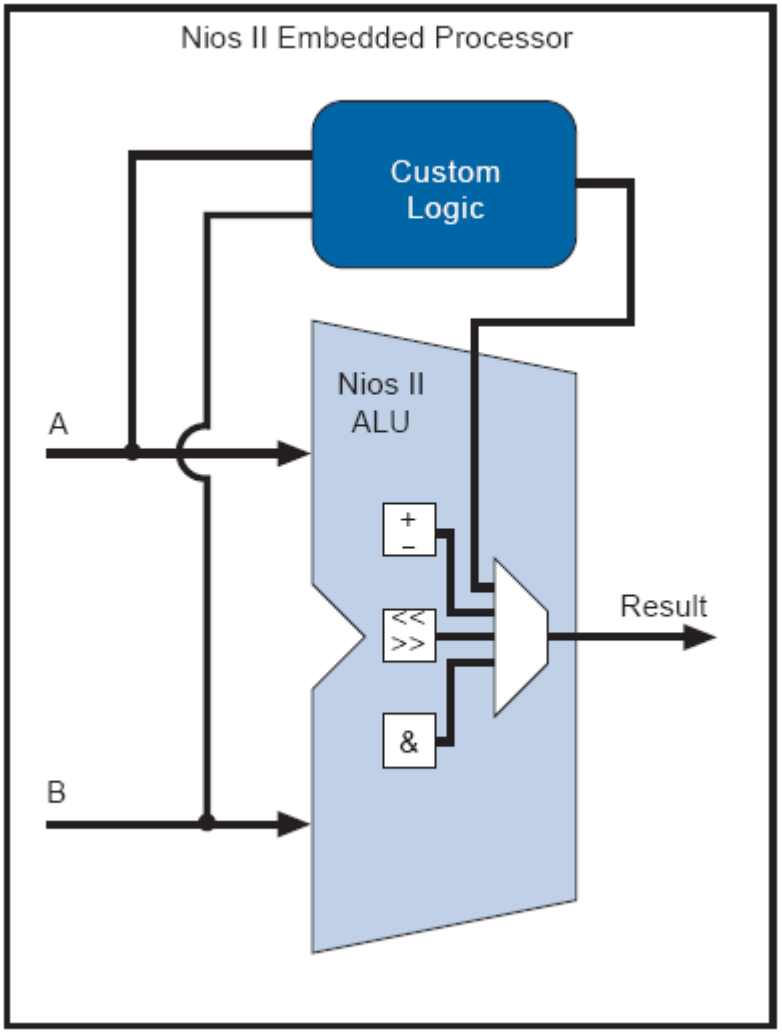
If the result is correct and enough fast, it's nice.

Some times it is necessary to accelerate the calculus time. Optimization is some time possible:

- Rewriting part of the code more efficiently
- Writing some part in assembly language
- Using a better processor or system
- Using specialize (co)processor
- Using multiprocessors
- → Transfer part of the code in hardware in an accelerator i.e. in a FPGA

Custom Instruction in FPGA

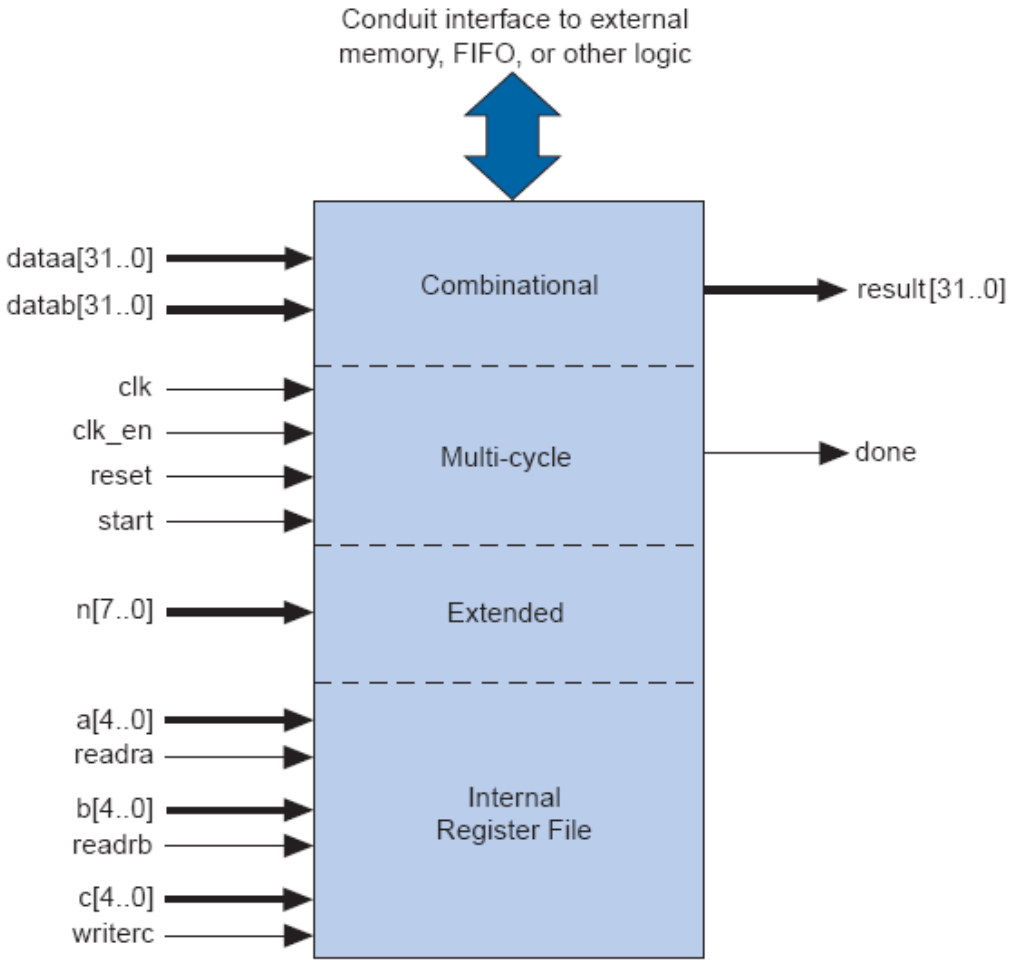
- In parallel to the normal ALU, custom logic can be added



Instruction implementation

The instruction can be:

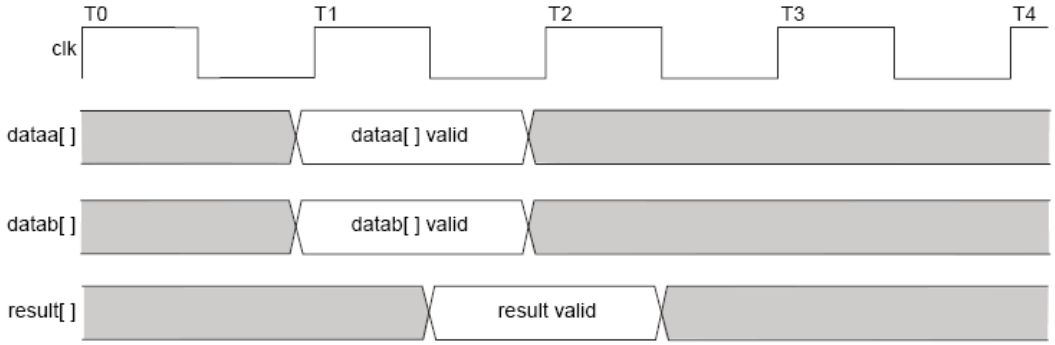
- **Combinational**
- **Multi-cycle**
- **Extended** (until 256 instructions)
- With internal **Regist File**
- **With external acces**



Combinatorial Instruction

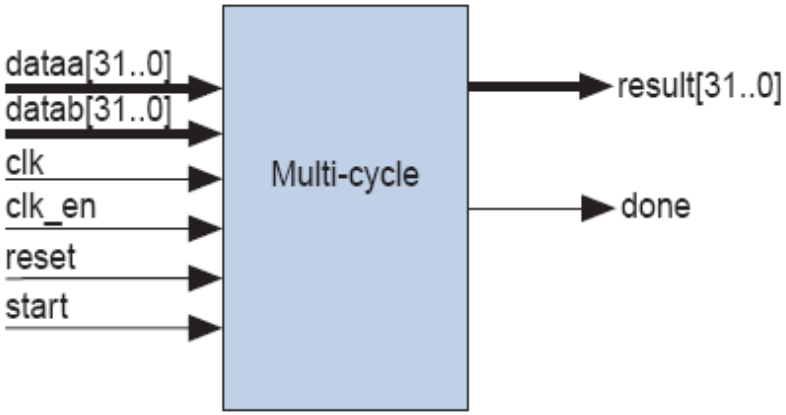
- 2 * 32 bits data, 32 bits result
- 1 clock cycle to resolve

Port Name	Direction	Required	Description
dataa[31:0]	Input	No	Input operand to custom instruction
datab[31:0]	Input	No	Input operand to custom instruction
result[31:0]	Output	Yes	Result of custom instruction



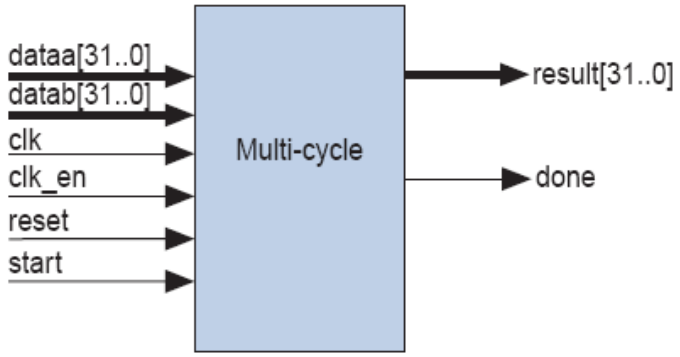
Multi-cycle Instruction

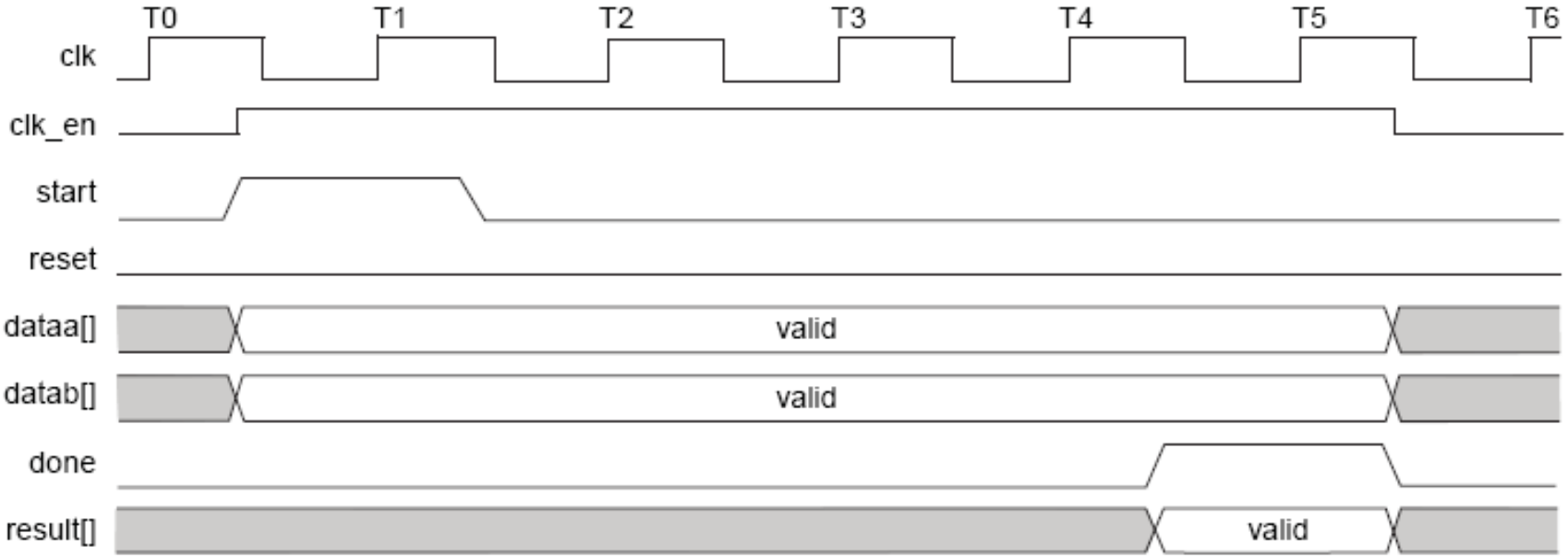
- 2 * 32 bits data, 32 bits result
- n clock cycle to resolve
- start – done handshake



Port Name	Direction	Required	Description
clk	Input	Yes	System clock
clk_en	Input	Yes	Clock enable
reset	Input	Yes	Synchronous reset
start	Input	No	Commands custom instruction logic to start execution
done	Output	No	Custom instruction logic indicates to the processor that execution is complete
dataa[31:0]	Input	No	Input operand to custom instruction
datab[31:0]	Input	No	Input operand to custom instruction
result[31:0]	Output	No	Result of custom instruction

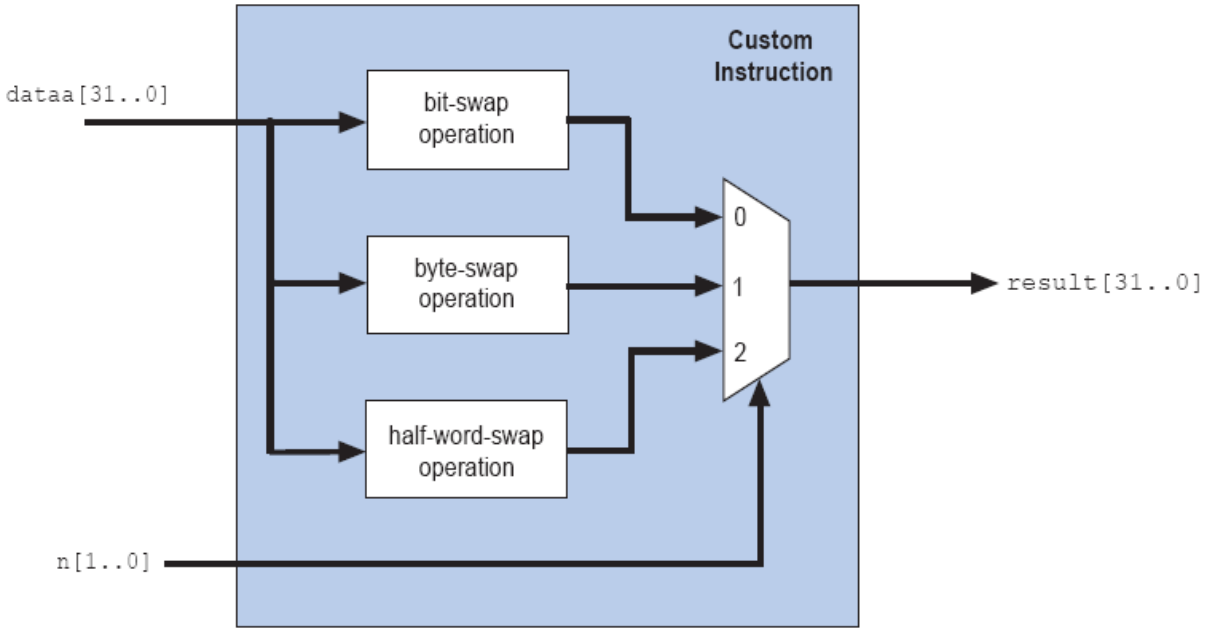
Multi-cycle Instruction





Extended instructions

- More than 1 instruction in the bloc
- Use a power of 2 instruction space (1, 2, 4, 8, …256)
- n[…] signals added for instruction index



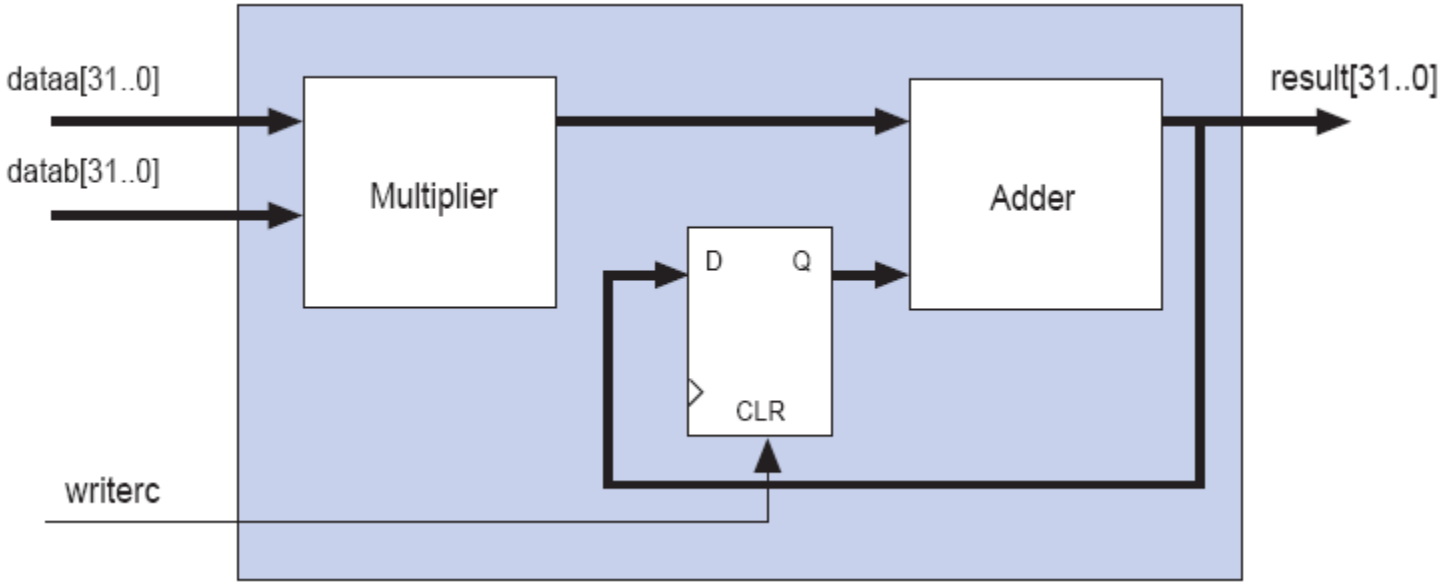
Internal Register File

- Internal registers → until 32
- 2 sources registers (Ra, Rb)
- 1 destination register (Rc)
- Can be mixed with Dataa, Datab or Datac 32 bits data bus

Port Name	Direction	Required	Description
readra	Input	No	If readra is high, the Nios II processor supplies dataa; if readra is low, custom instruction logic reads the internal register file indexed by a.
readrb	Input	No	If readrb is high, the Nios II processor supplies datab; if readrb is low, custom instruction logic reads the internal register file indexed by b.
writerc	Input	No	If writerc is high, the Nios II processor writes to the result port; if writerc is low, custom instruction logic writes to the internal register file indexed by c.
a[4:0]	Input	No	Custom instruction internal register file index
b[4:0]	Input	No	Custom instruction internal register file index
c[4:0]	Input	No	Custom instruction internal register file index

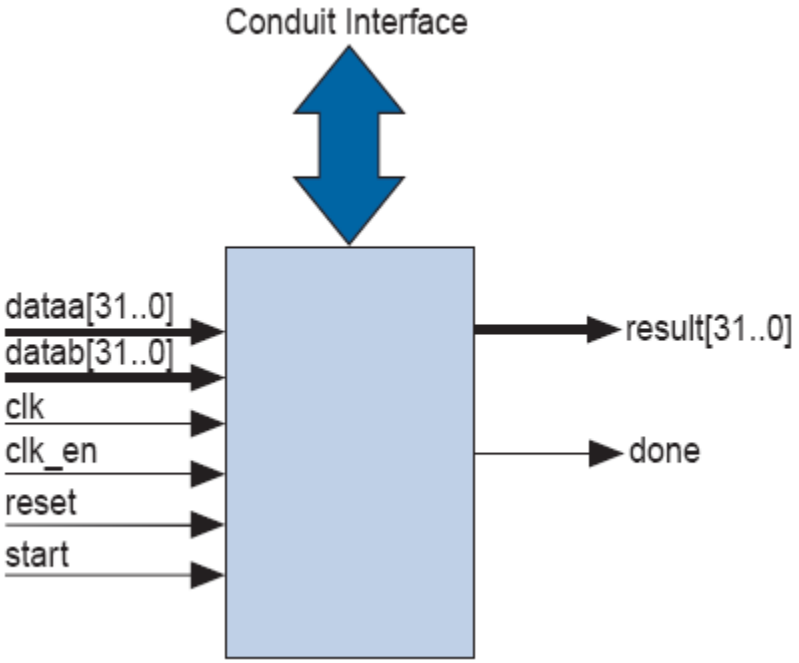
Internal Register File

- Example of mixed data selection
- dataa, datab and Rc



External interface

- External access allowed for multi-cycle access
- Available with internal register file too



Custom Instruction

- Access done in C with macro defined

```
#define ALT_CI_BITSWAP_N 0x00
#define ALT_CI_BITSWAP(A) __builtin_custom_ini(ALT_CI_BITSWAP_N,(A))
```

- Use extension of gcc
- Instruction: __builtin_custom_***o*n*i*$_1$*i*$_2$ *(instr num, input 1, input 2)***

  - Types of ***o i*$_1$ *i*$_2$**:
    - i integer
    - f float
    - p void *

Custom Instruction

- Example:
  - *void *__builtin_custom_**pnif** (int n, int dataa, float datab);*
  - ***pnif*** :
    - *p* output: void *
    - *n* separator
    - *i* input 1: integer
    - *f* input 2: float (32 bits)

    - All 3 parameters ***o*** i_1 i_2 are optional

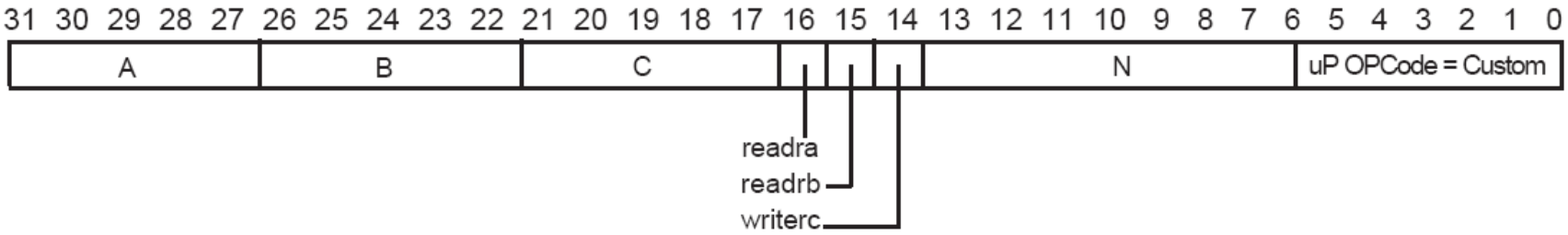
void __builtin_custom_nf (int n, float dataa);
float __builtin_custom_fnpi (int n, void * dataa, int datab);

Custom instruction example of use

- 2 instructions defined :

```
1. /* define void udef_macro1(float data); */
2. #define UDEF_MACRO1_N 0x00
3. #define UDEF_MACRO1(A) __builtin_custom_nf(UDEF_MACRO1_N, (A));
4. /* define float udef_macro2(void *data); */
5. #define UDEF_MACRO2_N 0x01
6. #define UDEF_MACRO2(B) __builtin_custom_fnp(UDEF_MACRO2_N, (B));
7.
8. int main (void)
9. {
10.     float a = 1.789;
11.     float b = 0.0;
12.     float *pt_a = &a;
13.
14.     UDEF_MACRO1(a);
15.     b = UDEF_MACRO2((void *)pt_a);
16.     return 0;
17. }
```

Assembly language definition



Instruction Fields: A = Register index of operand A
B = Register index of operand B
C = Register index of operand C
N = 8-bit number that selects instruction
readra = 1 if instruction uses processor's register rA, 0 otherwise
readrb = 1 if instruction uses processor's register rB, 0 otherwise
writerc = 1 if instruction provides result for processor's register rC, 0 otherwise

- Custom opcode : 0x32

Instruction assembly syntax

- Assembly syntaxe:
  - custom N, xC, xA, xB

  - N: instruction number from ***system.h***
  - x: r NIOS II register
    through dataa, datab and result
    readra, rb, rc at **1**
  - x: c Custom register file
    through a[], b[], c[] selection
    readra, rb, rc at **0**

Implementation

- Implementation in HDL (VHDL or Verilog) through SOPC Builder

Exercise

- To test the capabilities of software vs hardware implementation, design a NIOSII system to realize the basic function of bits mirror and swap:
- A 32 bits input a31..a0
- A 32 bits output result o31..o0
  - a31.. a24 → o7 .. o0 byte position change
  - a7 .. a0 → o31 .. o24
  - a23 .. a8 → o8 .. o23 bits order change !

Exercise

- This function can be done for a single variable
- This function can be done in a table of 1 to thousand of long data
  - ➢Do this function in C
  - ➢Implement it as a custom instruction
  - ➢Implement it in an accelerator module
  - ➢~~Implement it with the help of C2H~~
- Measure the performance in all cases