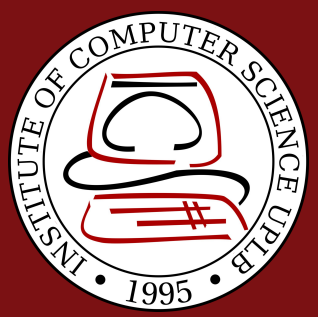
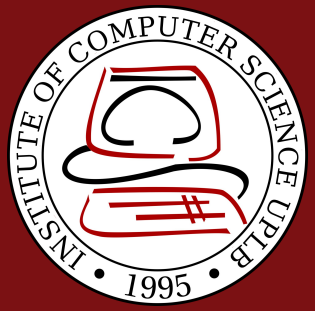




CMSC 132, First Semester AY 2020-2021

Computer Architecture

Joseph Anthony C. Hermocilla

ISA Design and Implementation

University of the Philippines
LOS BAÑOS

Resources

- https://github.com/jirawatsaesu/8-Bit-Simple-Processor
- David A. Patterson and John L. Hennessy. 2017. Computer Organization and Design, ARM Edition: The Hardware/Software Interface (ARM ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html
- https://github.com/hjl-tools/x86-psABI/wiki/X86-psABI
- https://docs.microsoft.com/en-us/cpp/build/x64-software-conventions
- http://www.cs.kent.edu/~durand/CS0/Notes/Chapter05/isa.html

Processor/Central Processing Unit (CPU)

- CPU = Datapath (ALU, Registers, Buses) + Control
- How do we program the CPU (tell it to do stuff)?

Instruction Set Architecture (ISA)

- An **abstraction** between the hardware and the lowest-level software
- **Includes anything programmers need to know to make a binary machine language program work correctly**
- Includes instructions and format, I/O operations, interrupts, addressing modes, registers, etc. (what you studied in CMSC 131!)

Instruction Set Architecture (ISA)

- Allows computer designers to talk about functions independently from the hardware that performs them
- This abstract **interface** enables many implementations (aka **microarchitectures**) of varying cost and performance to run identical software

Examples: ISA, Vendor, Product, Microarchitecture

- ISA: IA-32 and x86-64
  - Vendor(Product/Microarchitecture): Intel (Core i5/8th Gen-Kaby Lake Refresh) and AMD (Ryzen 5000/4th Gen-Zen 3)
- ISA: ARMv8 A64
  - Vendor(Product/Microarchitecture): MediaTek (Helio P70/Cortex-A73+Cortex-A53) and Qualcomm (Kryo 240/Cortex-A73+Cortex-A53)

Application Binary Interface (ABI)

- Combination of the basic instruction set and the operating system interface provided for application programmers
  - For general-purpose use, you really can't do much without an OS (CMSC 125!)
- Describes function calling conventions, parameter passing, sizes of C data types, executable file formats (ELF, PE)
- Examples: IA-32, x86-64 System V ABI (Linux), x64 (Windows)

ISA taxonomy based on where operands are stored

- Stack-based - uses a stack memory (LIFO), operations are performed on the top of the stack
- Accumulator-based - one register is designated as accumulator, use in instructions is implied
- General purpose - operands are explicitly named in the instruction
  - Register-to-register
  - Register-to-memory
  - Memory-to-register

Considerations in ISA design

- Types/Class of instructions (Operations in the Instruction set)
  - arithmetic/logic, data movement, branching/control flow, I/O, etc.
- Types and sizes of operands (in bits)
  - 8, 16, 32, 64, 128, floating point
- Addressing Modes
  - Register, direct, indirect, immediate, etc
- Addressing Memory
  - Byte-addressable, word-addressable, endianness
- Encoding and Instruction Formats
  - Opcode field, addresses field, mode field, etc.
- Compiler-related issues

Example: x86-64 instruction format (CISC)

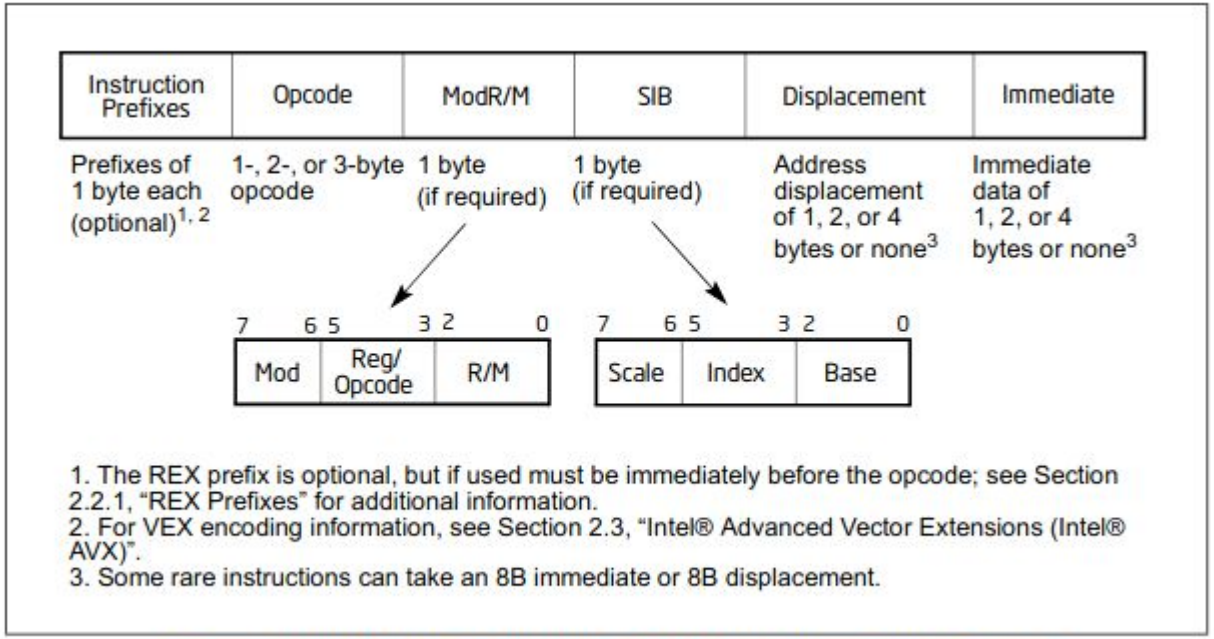


Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format

Example: ARMv8 A64 instruction format (RISC)

C4.1 A64 instruction set encoding

The A64 instruction encoding is:

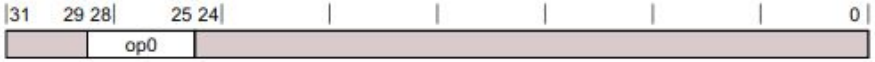


Table C4-1 Main encoding table for the A64 instruction set

Decode fields op0	Decode group or instruction page
0000	*Reserved*
0001	Unallocated.
0010	SVE Instructions. See *The Scalable Vector Extension (SVE)* on page A2-99.
0011	Unallocated.
100x	*Data Processing -- Immediate*
101x	*Branches, Exception Generating and System instructions* on page C4-271
x1x0	*Loads and Stores* on page C4-279
x101	*Data Processing -- Register* on page C4-310
x111	*Data Processing -- Scalar Floating-Point and Advanced SIMD* on page C4-320

The Optimal Machine Architecture (T.O.M.A.)
ISA

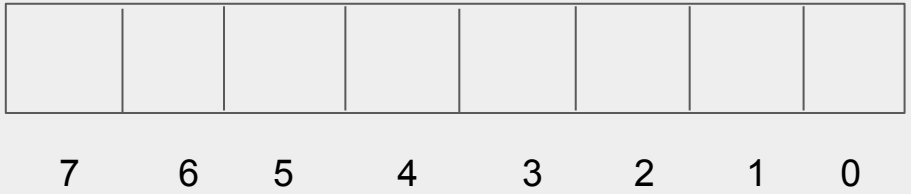
University of the Philippines
LOS BAÑOS

“Features”

- Has 4 8-bit registers named $s0, $s1, $s2, $s3 in assembly code
- Instruction memory is 8x8, address line is 3 bits
- Has a 3-bit Program Counter (PC) register
- Single-cycle - completes instruction execution in one clock cycle
- Has no data memory, thus has no load and store instructions
- Has no control transfer instructions

Instruction Format

- Bits 7,6 - opcode (op)
- Bits 5,4 - source register 1 (rs)
- Bits 3,2 - source register 2 (rt)
- Bits 1,0 - destination register (rd)/immediate



Supported Instructions

- **and** : rd <= rs AND rt (op=00)
- **add** : rd <= rs + rt (op=01)
- **sub** : rd <= rs - rt (op=10)
- **addi** : rs <= rt + immediate (op=11)

Example: Assembly Language Code ; Machine Code

```
addi $s0, $s0, 2        ; 11000010b, 0xC2
addi $s1, $s1, 1        ; 11010101b, 0xD5
addi $s2, $s2, 3        ; 11101011b, 0xEB
add $s3, $s0, $s1       ; 01000111b, 0x47
sub $s0, $s2, $s3       ; 10101100b, 0xAC
```

ASM Syntax: `<instruction> <dst>, <src1>, <src2/imm>`

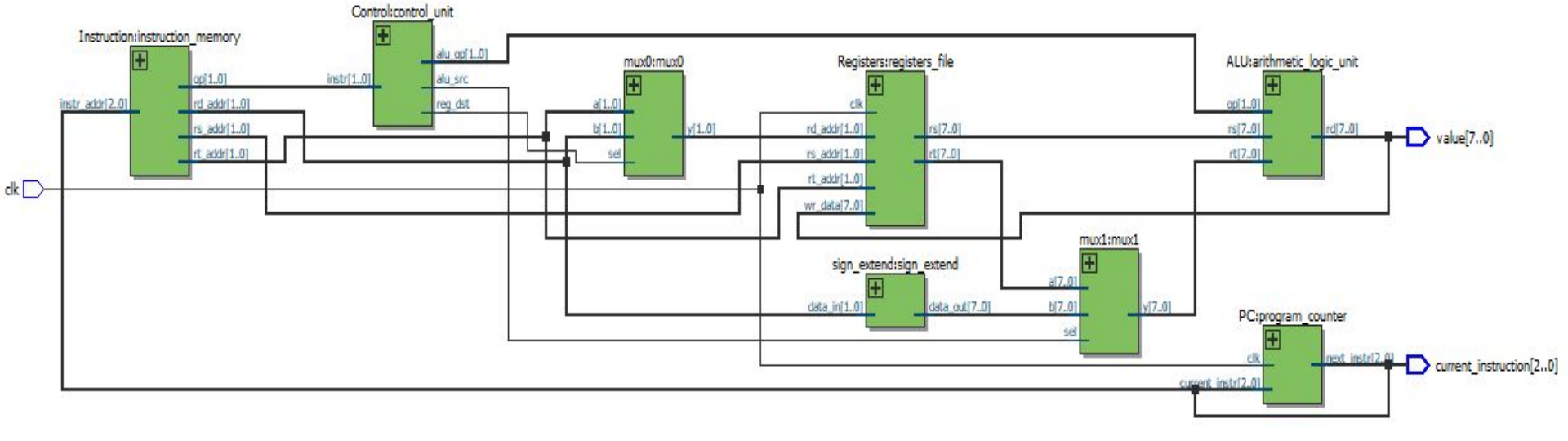
REDHORSE 500
An implementation of TOMA

University of the Philippines
LOS BAÑOS

Processor

```
ENTITY Processor IS
    PORT
    (
        clk :  IN  STD_LOGIC;
        current_instruction :  OUT  STD_LOGIC_VECTOR(2 DOWNTO 0);
        value :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0)
    );
END Processor;
```

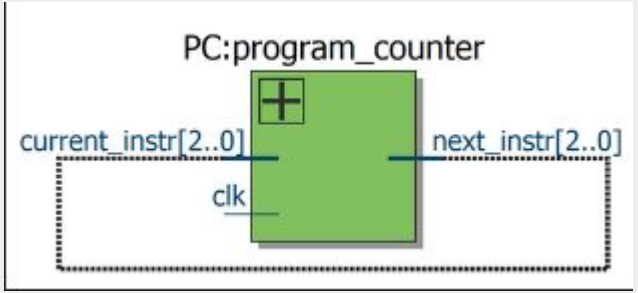
Processor


Instruction:instruction_memory
instr_addr[2..0]
op[1..0]
rd_addr[1..0]
rs_addr[1..0]
rt_addr[1..0]
Control:control_unit
instr[1..0]
alu_op[1..0]
alu_src
reg_dst
mux0:mux0
a[1..0]
b[1..0]
sel
y[1..0]
Registers:registers_file
clk
rd_addr[1..0]
rs_addr[1..0]
rt_addr[1..0]
wr_data[7..0]
rs[7..0]
rt[7..0]
ALU:arithmetic_logic_unit
op[1..0]
rs[7..0]
rt[7..0]
rd[7..0]
value[7..0]
clk
sign_extend:sign_extend
data_in[1..0]
data_out[7..0]
mux1:mux1
a[7..0]
b[7..0]
sel
y[7..0]
PC:program_counter
clk
current_instr[2..0]
next_instr[2..0]
current_instruction[2..0]

Program Counter

Program Counter

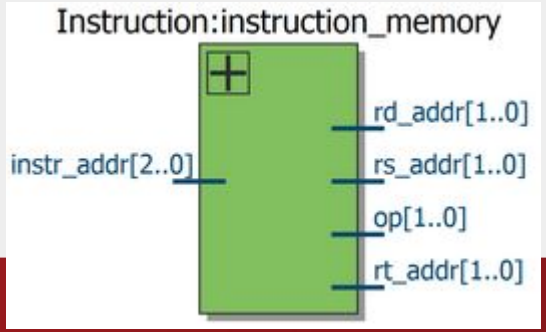
```
entity PC is
  port(
    clk           : in std_logic;
    current_instr : in std_logic_vector(2 downto 0);   -- current instruction
    next_instr    : out std_logic_vector(2 downto 0)   -- next instruction
  );
end PC;
```



Instruction Memory

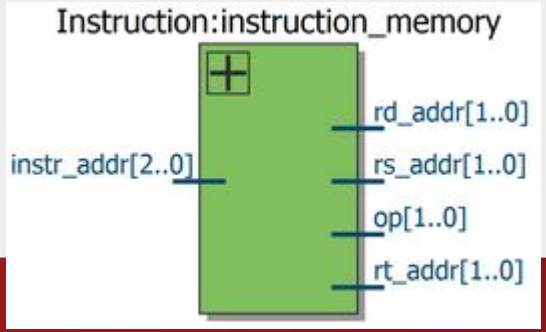
Instruction Memory

```
entity Instruction is
  port(
    instr_addr : in std_logic_vector(2 downto 0);    -- instruction address

    op         : out std_logic_vector(1 downto 0);   -- operation code
    rs_addr    : out std_logic_vector(1 downto 0);   -- source register 1 address
    rt_addr    : out std_logic_vector(1 downto 0);   -- source register 2 address
    rd_addr    : out std_logic_vector(1 downto 0)    -- destination register address
  );
end Instruction;
```



Instruction Memory - sample hard-coded contents

```
constant instr : instruction_set := (
"11000010",  -- addi $s0, $s0, 2
"11010101",  -- addi $s1, $s1, 1
"11101011",  -- addi $s2, $s2, 3
"01000111",  -- add  $s3, $s0, $s1
"10101100",  -- sub  $s0, $s2, $s3
"00000000",
"00000000",
"00000000"
);
```

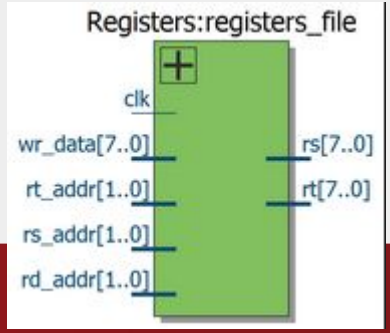


Register File

University of the Philippines
LOS BAÑOS

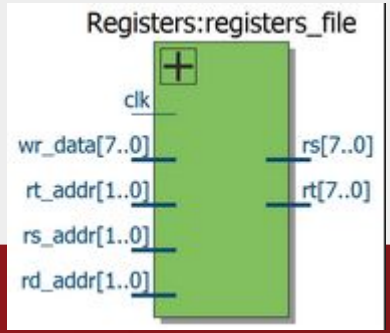
Register File

```
entity Registers is
  port(
    clk     : in std_logic;

    rs_addr : in std_logic_vector(1 downto 0);    -- source register 1 address
    rt_addr : in std_logic_vector(1 downto 0);    -- source register 2 address
    rd_addr : in std_logic_vector(1 downto 0);    -- destination register address
    wr_data : in std_logic_vector(7 downto 0);    -- write data to destination register

    rs      : out std_logic_vector(7 downto 0);   -- source register 1 value
    rt      : out std_logic_vector(7 downto 0)    -- source register 2 value
  );
end Registers;
```



Register File - hard-coded initial values

```
signal reg: registerFile := (
  "00000001",
  "00000010",
  "00000011",
  "00000100"
  );
```

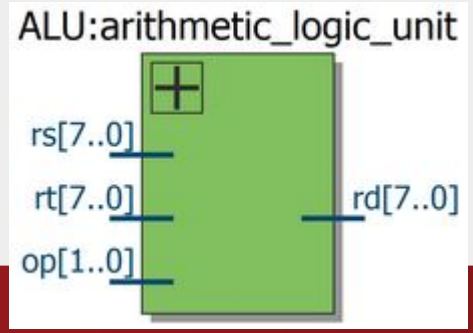


ALU

University of the Philippines
LOS BAÑOS

ALU

```
entity ALU is
  port(
    op  : in std_logic_vector(1 downto 0);   -- operation code

    rs  : in std_logic_vector(7 downto 0);   -- source register 1
    rt  : in std_logic_vector(7 downto 0);   -- source register 2
    rd  : out std_logic_vector(7 downto 0)   -- destination register
  );
end ALU;
```

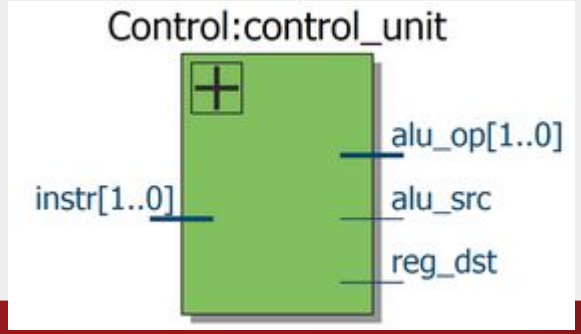


Control

University of the Philippines
LOS BAÑOS

Control

```
entity Control is
  port(
    instr   : in std_logic_vector(1 downto 0);   -- instruction

    alu_op  : out std_logic_vector(1 downto 0);  -- operation code of AlU
    alu_src : out std_logic;                     -- ALU select ADDi
    reg_dst : out std_logic                      -- select destination address register
  );
end Control;
```



Extras

University of the Philippines
LOS BAÑOS

Extras: Mux0 - for addi

```
entity mux0 is
  port(
    sel : in std_logic;                          -- select destination address
    a   : in std_logic_vector(1 downto 0);       -- source register address
    b   : in std_logic_vector(1 downto 0);       -- default destination address
    y   : out std_logic_vector(1 downto 0)       -- destination address
  );
end mux0;
```

Extras: Mux1 - for addi

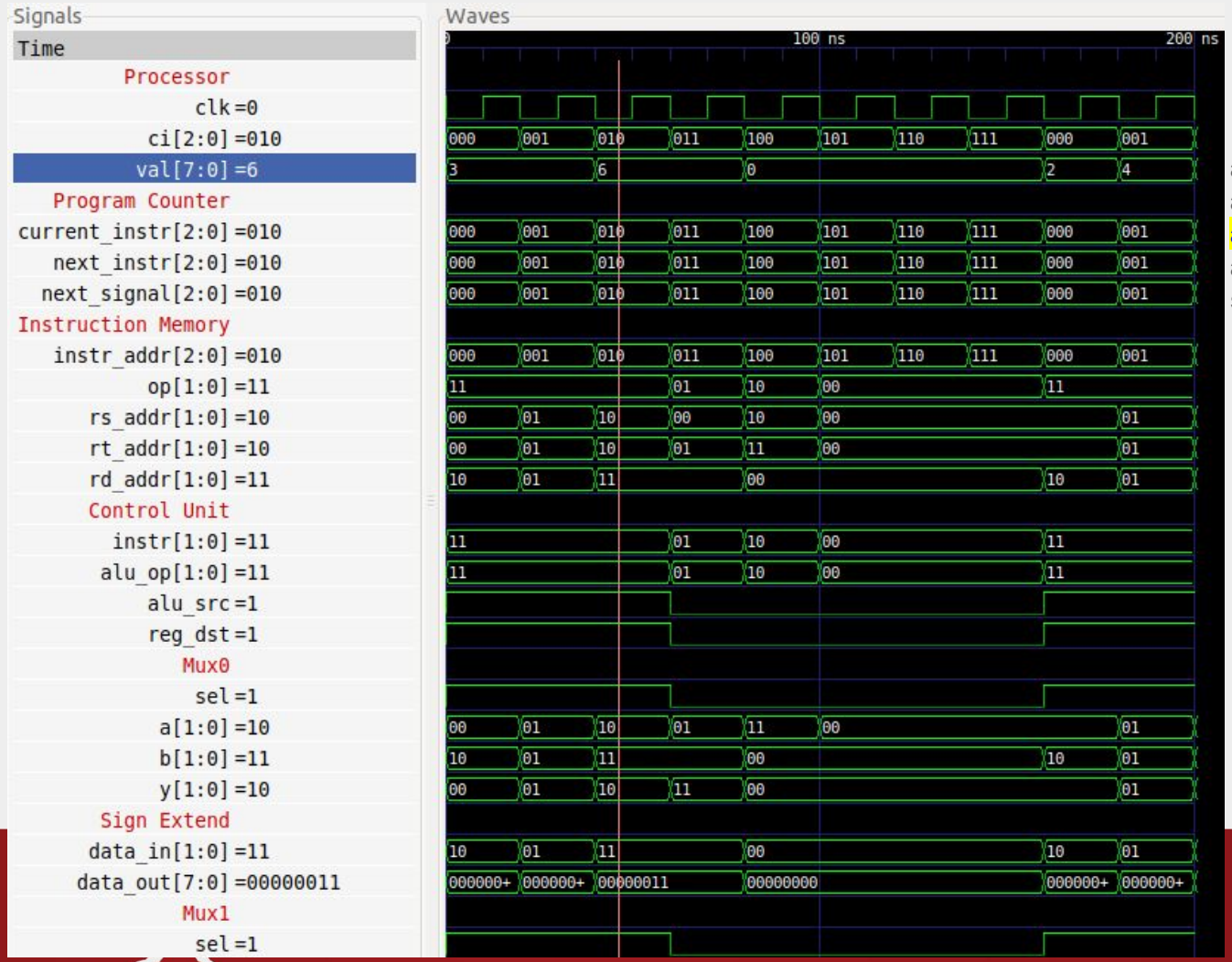
```
entity mux1 is
  port(
    sel : in std_logic;                          -- select data
    a   : in std_logic_vector(7 downto 0);       -- default data
    b   : in std_logic_vector(7 downto 0);       -- data from instruction
    y   : out std_logic_vector(7 downto 0)       -- data out
  );
end mux1;
```

Extras: sign_extend

```
entity sign_extend is
  port(
    data_in  : in std_logic_vector(1 downto 0);
    data_out : out std_logic_vector(7 downto 0)
  );
end sign_extend;
```

Processor

University of the Philippines
LOS BAÑOS




```
addi $s0, $s0, 2;11000010b
addi $s1, $s1, 1;11010101b
addi $s2, $s2, 3;11101011b
add $s3, $s0, $s1;01000111b
sub $s0, $s2, $s3;10101100b
```

Enjoy! :)

University of the Philippines
LOS BAÑOS