Lab Assignment 1 (lab1):

RISC-V instruction set architecture and programming of Nios V/m processor
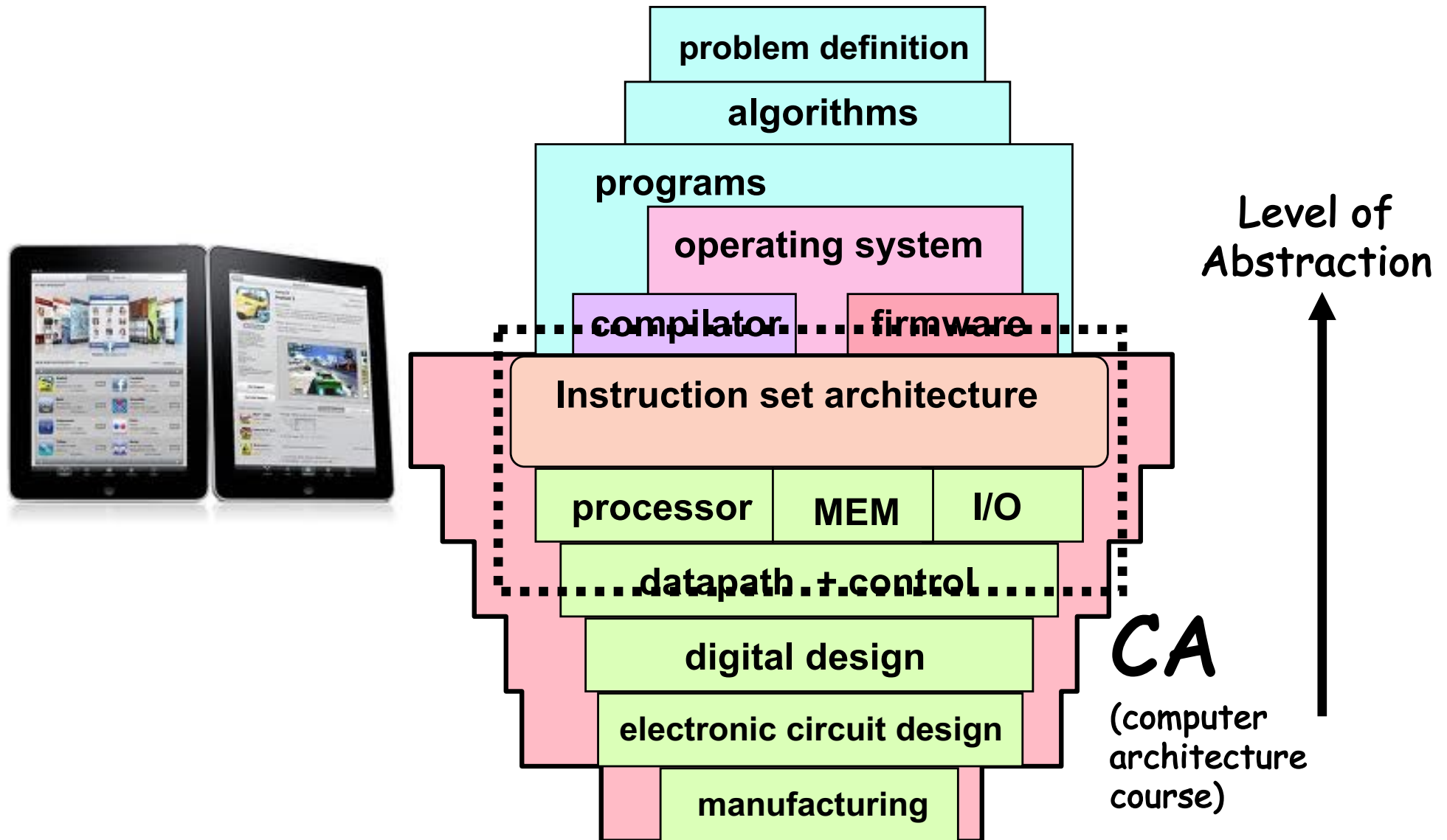
Computer Architecture (40969)
School of Computer Science (EII)
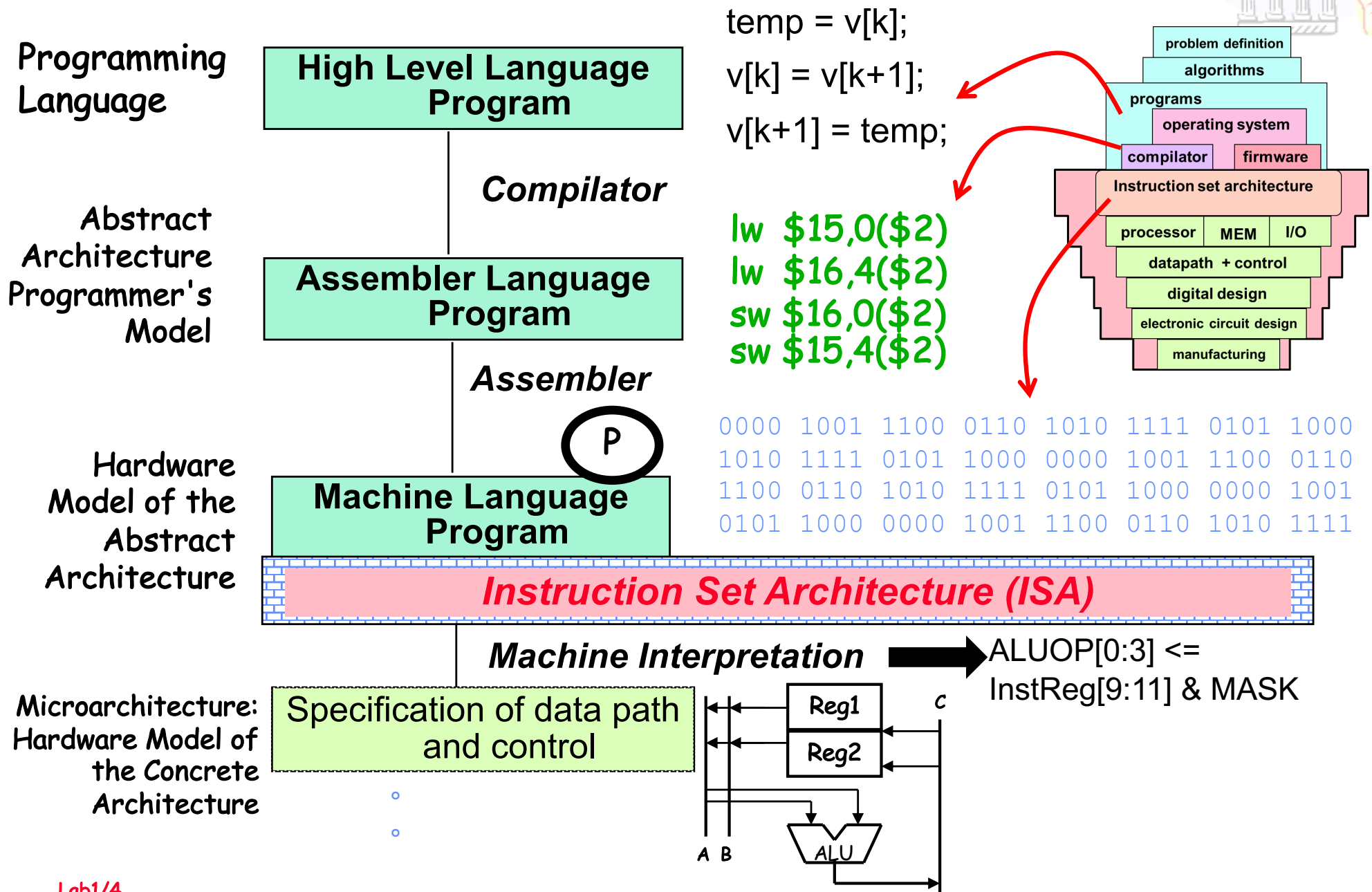University of Las Palmas de Gran Canaria

Summary

- Hierarchy of computer abstraction levels
- Elements of RISC-V instruction set architecture
- Nios V soft processor operating modes
- General purpose and control registers
- Access to the address apace
- Types of RISC-V instructions
- Example of a RISC-V assembler program
- Subroutines
- Part I: executing an example program
- Part II: designing and debugging a simple program
- Part III: modifying a machine instruction
- Part IV: programming the multiplication

Hierarchy of computer abstraction levels

problem definition

algorithms

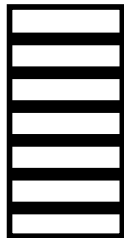programs

operating system

compiler   firmware

Instruction set architecture

| processor | MEM | I/O |

datapath + control

digital design

electronic circuit design

manufacturing

Level of Abstraction

CA

(computer architecture course)

Example of the Hierarchical Description

Programming Language

High Level Language Program

temp = v[k];

v[k] = v[k+1];

v[k+1] = temp;

Compilator

Abstract Architecture Programmer's Model

Assembler Language Program

lw $15,0($2)
lw $16,4($2)
sw $16,0($2)
sw $15,4($2)

Assembler

(P)

Hardware Model of the Abstract Architecture

Machine Language Program

0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111

Instruction Set Architecture (ISA)

Machine Interpretation ➡ ALUOP[0:3] <= InstReg[9:11] & MASK

Microarchitecture: Hardware Model of the Concrete Architecture

Specification of data path and control

problem definition
algorithms
programs
operating system
compilator   firmware
Instruction set architecture
processor   MEM   I/O
datapath + control
digital design
electronic circuit design
manufacturing

Reg1   C
Reg2
A B   ALU

Nios V

P: PROCESSOR

Registers

PC

ALU:
+, -, x, :, etc.

Control Unit

Nios® V Processor Reference Manual

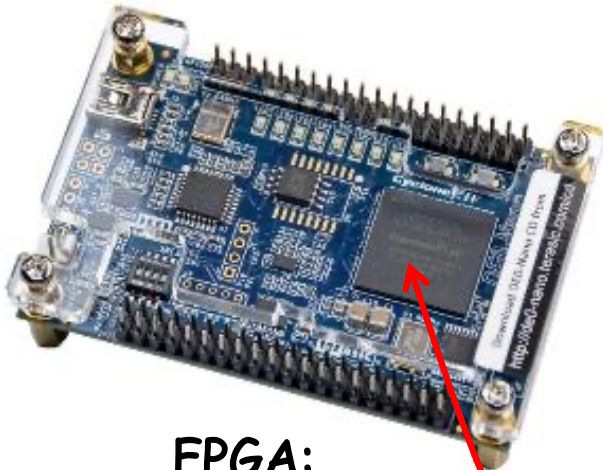Updated for Intel® Quartus® Prime Design Suite: **23.1**

intel.
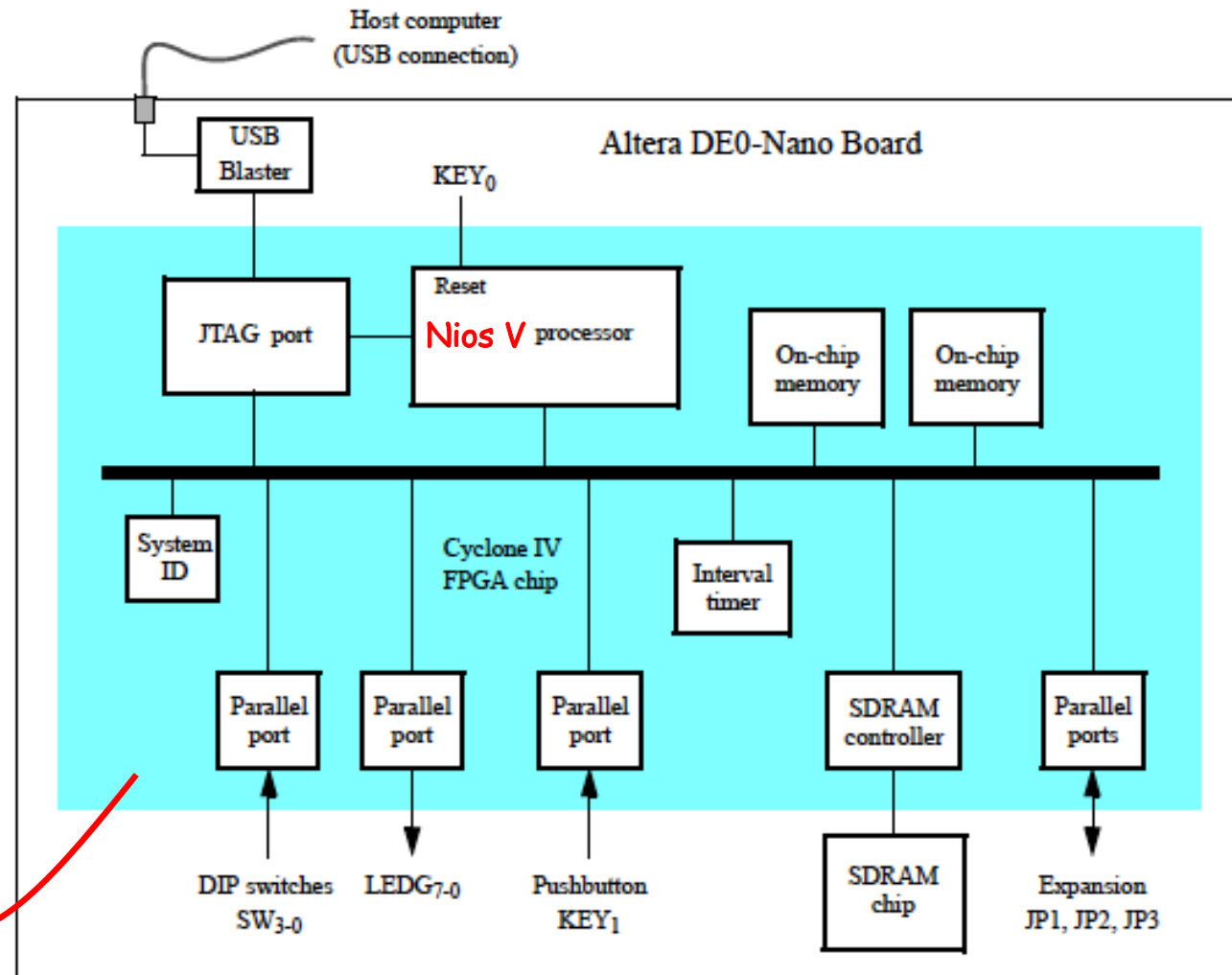
Online Version   683632
Send Feedback   2023.05.26

Nios V-based system-on-chip computer

Terasic
DE0-Nano board

FPGA:
field
programmable
gate array

Host computer
(USB connection)

Altera DE0-Nano Board

USB Blaster

KEY$_0$

Reset

JTAG port

Nios V processor

On-chip memory

On-chip memory

System ID

Cyclone IV FPGA chip

Interval timer

Parallel port

Parallel port

Parallel port

SDRAM controller

Parallel ports

DIP switches SW$_{3-0}$

LEDG$_{7-0}$

Pushbutton KEY$_1$

SDRAM chip
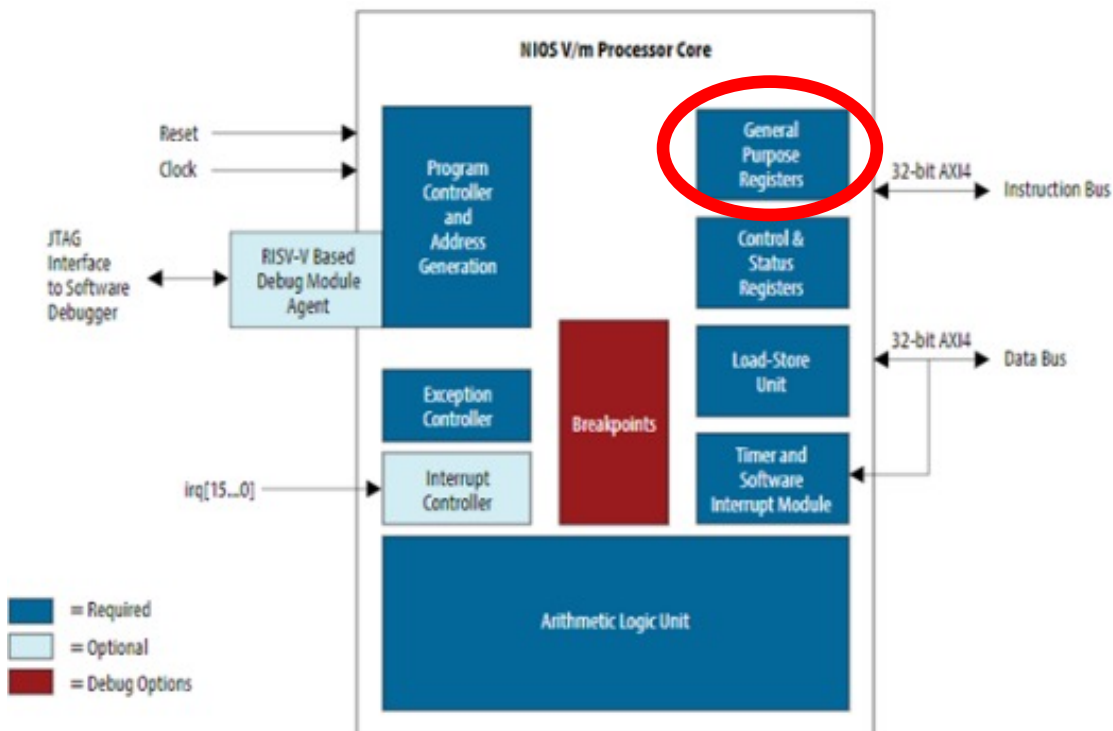
Expansion JP1, JP2, JP3

General-Purpose Register File

Nios V/m processor implementation supports a flat register file.

The register file contains thirty-two 32-bit general-purpose integer registers.
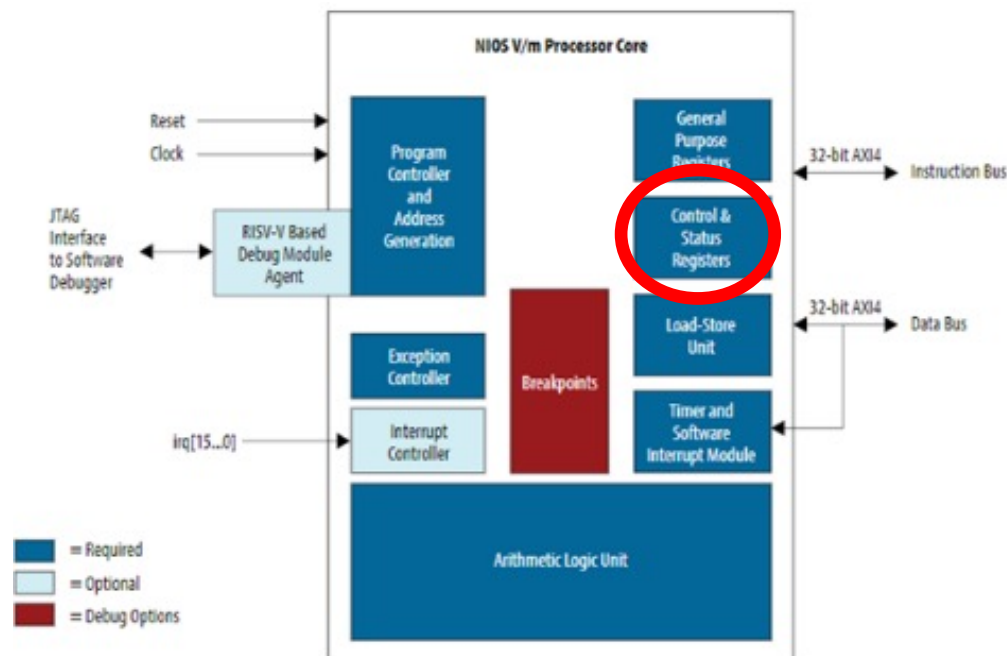

NIOS V/m Processor Core

Reset
Clock

JTAG Interface to Software Debugger

Program Controller and Address Generation

RISV-V Based Debug Module Agent

Exception Controller

Breakpoints

Interrupt Controller

irq[15...0]

General Purpose Registers

Control & Status Registers

Load-Store Unit

Timer and Software Interrupt Module

Arithmetic Logic Unit

32-bit AXI4 — Instruction Bus

32-bit AXI4 — Data Bus

■ = Required
□ = Optional
■ = Debug Options

Register Name	ABI Name	Description
x0	zero	Hard-Wired Zero
x1	ra	Return Address
x2	sp	Stack Pointer
x3	gp	Global Pointer
x4	tp	Thread Pointer
x5	t0	Temporary/Alternate Link Register
x6-7	t1-t2	Temporary Register
x8	s0/fp	Saved Register (Frame Pointer)
x9	s1	Saved Register
x10-11	a0-a1	Function Argument/Return Value Registers
x12-17	a2-a7	Function Argument Registers
x18-27	s2-s11	Saved Registers
x28-31	t3-t6	Temporary Registers

Control and Status Registers

Register	Description
misa	Machine ISA
mvendorid	Machine Vendor ID
marchid	Machine Architecture ID
mimpid	Machine Implementation ID
mstatus	Machine Status
mcause	Machine trap cause
mtvec	Trap vector base address

Register	Description
mhartid	Machine Hardware thread ID
mepc	Machine exception program counter
mie	Machine interrupt enable
mip	Machine interrupt pending
mtval	Machine trap value
mscratch	Scratch register

Table 1.3: RISC-V Machine Mode Registers

Connection to memory and input/output devices

Host computer (USB connection)

Altera DE0-Nano Board

USB Blaster

KEY$_0$

JTAG port

Reset

Nios V processor

On-chip memory

On-chip memory

System ID

Cyclone IV FPGA chip

Interval timer

Parallel port

Parallel port

Parallel port

SDRAM controller

Parallel ports

DIP switches SW$_{3-0}$

LEDG$_{7-0}$

Pushbutton KEY$_1$

SDRAM chip

Expansion JP1, JP2, JP3

SHARED ADDRESSING SPACE: MEMORY & INPUT / OUTPUT

0
1
2
.
.
.
100
101
.
.
10000
.
.
.

POINTER FOR AN ADDRESS

ADDRESS CONTENT

Lab1/11

RISC-V/Nios V addressing Modes

Addressing modes in RISC-V determine how memory addresses are calculated for load and store instructions. These modes define how the base address, offset, and index registers are combined to form the effective memory address.

- Immediate: `LW xd, imm(xs1)`
- Register: `LW xd, (xs1)`
- Base: `LW xd, offset(xs1)`
- Indexed: `LW xd, offset(xs1, xs2)`

RISC-V instruction types given their formats

RISC-V Instruction Set

Core Instruction Formats

31 27	26 25 24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20|10:1|11|19:12]				rd	opcode	J-type

RISC-V instruction types given the involved operations

RV32I Base Integer Instructions

ALU

Inst	Name	FMT	Opcode	funct3	funct7	Description (C)	Note
add	ADD	R	0110011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0110011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0110011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0110011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0110011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0110011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0110011	0x5	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0110011	0x5	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2	0x00	rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3	0x00	rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0		rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x4		rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x6		rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x7		rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	imm[5:11]=0x00	rd = rs1 << imm[0:4]	
srli	Shift Right Logical Imm	I	0010011	0x5	imm[5:11]=0x00	rd = rs1 >> imm[0:4]	
srai	Shift Right Arith Imm	I	0010011	0x5	imm[5:11]=0x20	rd = rs1 >> imm[0:4]	msb-extends
slti	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltiu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	

RISC-V instruction types given the involved operations

ALU

Standard Extensions

RV32M Multiply Extension

Inst	Name	FMT	Opcode	funct3	funct7	Description (C)
mul	MUL	R	0110011	0x0	0x01	rd = (rs1 * rs2)[31:0]
mulh	MUL High	R	0110011	0x1	0x01	rd = (rs1 * rs2)[63:32]
mulsu	MUL High (S) (U)	R	0110011	0x2	0x01	rd = (rs1 * rs2)[63:32]
mulu	MUL High (U)	R	0110011	0x3	0x01	rd = (rs1 * rs2)[63:32]
div	DIV	R	0110011	0x4	0x01	rd = rs1 / rs2
divu	DIV (U)	R	0110011	0x5	0x01	rd = rs1 / rs2
rem	Remainder	R	0110011	0x6	0x01	rd = rs1 % rs2
remu	Remainder (U)	R	0110011	0x7	0x01	rd = rs1 % rs2

RISC-V instruction types given the involved operations

RV32I Base Integer Instructions

MEMORY

Inst	Name	FMT	Opcode	funct3	funct7	Description (C)	Note
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	

RISC-V instruction types given the involved operations

RV32I Base Integer Instructions

BRANCH AND JUMPS

Inst	Name	FMT	Opcode	funct3	funct7	Description (C)	Note
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≥	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1 + imm	
ecall	Environment Call	I	1110011	0x0	imm=0x0	Transfer control to OS	
ebreak	Environment Break	I	1110011	0x0	imm=0x1	Transfer control to debugger	

RISC-V instruction types given the involved operations

Pseudo Instructions

Pseudoinstruction	Base Instruction(s)	Meaning
la rd, symbol	auipc rd, symbol[31:12] addi rd, rd, symbol[11:0]	Load address
l{b|h|w|d} rd, symbol	auipc rd, symbol[31:12] l{b|h|w|d} rd, symbol[11:0](rd)	Load global
s{b|h|w|d} rd, symbol, rt	auipc rt, symbol[31:12] s{b|h|w|d} rd, symbol[11:0](rt)	Store global
fl{w|d} rd, symbol, rt	auipc rt, symbol[31:12] fl{w|d} rd, symbol[11:0](rt)	Floating-point load global
fs{w|d} rd, symbol, rt	auipc rt, symbol[31:12] fs{w|d} rd, symbol[11:0](rt)	Floating-point store global
nop	addi x0, x0, 0	No operation
li rd, immediate	*Myriad sequences*	Load immediate
mv rd, rs	addi rd, rs, 0	Copy register
not rd, rs	xori rd, rs, -1	One's complement
neg rd, rs	sub rd, x0, rs	Two's complement
negw rd, rs	subw rd, x0, rs	Two's complement word
sext.w rd, rs	addiw rd, rs, 0	Sign extend word
seqz rd, rs	sltiu rd, rs, 1	Set if $=$ zero
snez rd, rs	sltu rd, x0, rs	Set if $\neq$ zero
sltz rd, rs	slt rd, rs, x0	Set if $<$ zero
sgtz rd, rs	slt rd, x0, rs	Set if $>$ zero
fmv.s rd, rs	fsgnj.s rd, rs, rs	Copy single-precision register
fabs.s rd, rs	fsgnjx.s rd, rs, rs	Single-precision absolute value
fneg.s rd, rs	fsgnjn.s rd, rs, rs	Single-precision negate
fmv.d rd, rs	fsgnj.d rd, rs, rs	Copy double-precision register
fabs.d rd, rs	fsgnjx.d rd, rs, rs	Double-precision absolute value
fneg.d rd, rs	fsgnjn.d rd, rs, rs	Double-precision negate
beqz rs, offset	beq rs, x0, offset	Branch if $=$ zero
bnez rs, offset	bne rs, x0, offset	Branch if $\neq$ zero
blez rs, offset	bge x0, rs, offset	Branch if $\leq$ zero
bgez rs, offset	bge rs, x0, offset	Branch if $\geq$ zero
bltz rs, offset	blt rs, x0, offset	Branch if $<$ zero
bgtz rs, offset	blt x0, rs, offset	Branch if $>$ zero
bgt rs, rt, offset	blt rt, rs, offset	Branch if $>$
ble rs, rt, offset	bge rt, rs, offset	Branch if $\leq$
bgtu rs, rt, offset	bltu rt, rs, offset	Branch if $>$, unsigned
bleu rs, rt, offset	bgeu rt, rs, offset	Branch if $\leq$, unsigned
j offset	jal x0, offset	Jump
jal offset	jal x1, offset	Jump and link
jr rs	jalr x0, rs, 0	Jump register
jalr rs	jalr x1, rs, 0	Jump and link register
ret	jalr x0, x1, 0	Return from subroutine
call offset	auipc x1, offset[31:12] jalr x1, x1, offset[11:0]	Call far-away subroutine
tail offset	auipc x6, offset[31:12] jalr x0, x6, offset[11:0]	Tail call far-away subroutine
fence	fence iorw, iorw	Fence on all memory and I/O

Example of a RISC-V assembler program

```
            .text                       /* executable code follows */
            .global _start
_start:
            la    x16, 0x10000010        /* green LED base address */
            la    x15, 0x10000040        /* slider switch base address */
            la    x17, 0x10000050        /* pushbutton KEY base address */
            la    x19, LEDG_bits
            lw    x6, 0(x19)             /* load pattern for LEDG lights */
DO_DISPLAY:
            lw    x4, 0(x15)             /* load slider (DIP) switches */
            lw    x5, 0(x17)             /* load pushbuttons */
            beq   x5, x0, NO_BUTTON
            mv    x6, x4                 /* use SW values on LEDG */
            add   a0, zero, x4
            add   a1, zero, 8
            jal   ra, rotl
            add   x4, a0, zero
            or    x6, x6, x4
            add   a0, zero, x4
            add   a1, zero, 8
            jal   ra, rotl
```

Program zone

Section 1: main

Example of a RISC-V assembler program

```
        add   x4, a0, zero
        or    x6, x6, x4
        add   a0, zero, x4
        add   a1, zero, 8
        jal   ra, rotl
        add   x4, a0, zero
        or    x6, x6, x4
WAIT:
        lw    x5, 0(x17)          /* load pushbuttons */
        bne   x5, x0, WAIT        /* wait for button release */
NO_BUTTON:
        sw    x6, 0(x16)          /* store to LEDG */
        add   a0, zero, x6
        add   a1, zero, 1
        jal   ra, rotl
        add   x6, a0, zero
        li    x7, 150000   /* delay counter */
DELAY:
        addi  x7, x7, -1
        bne   x7, x0, DELAY
        j     DO_DISPLAY
```

Program zone

Section 1: main

Example of a RISC-V assembler program

rotl:

    sll   a2, a0, a1

    sub   a4, zero, a1

    srl   a3, a0, a4

    or   a0, a2, a3

    ret

    .data   /* data follows */

    LEDG_bits:

    .word 0x0F0F0F0F

    .end

Program zone

Section 2: subroutine

Data zone

Subroutines

```
int add3 (int a, int b, int c)
{
return a + b +c;
}

int sum = 0;
main ()
{
    sum += add3 (1, 2, 3);
    sum += 10;
    sum += add3 (10, 20, 30);
}
```

3

2

1

2

1

2

4

4

caller   callee

main

Call site   ──────────────► Add3 (1,2,3)

Sum+= ◄──────────

Sum+=10

        ──────────────────────► Add3 (10,20,30)

Sum+= ◄──────────
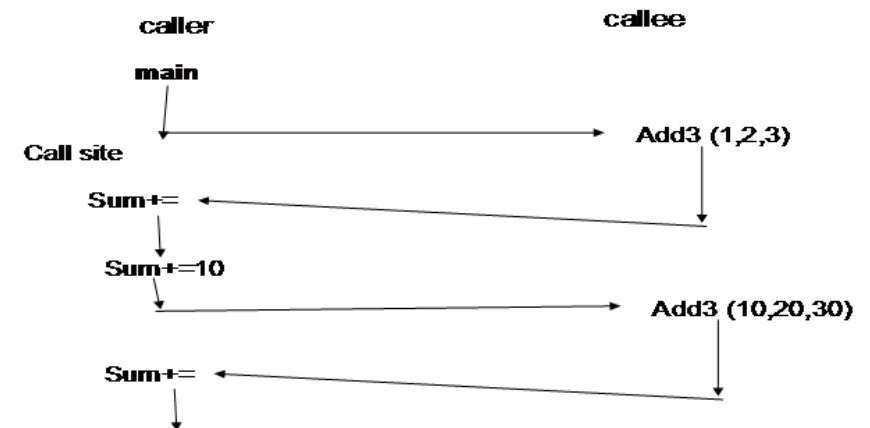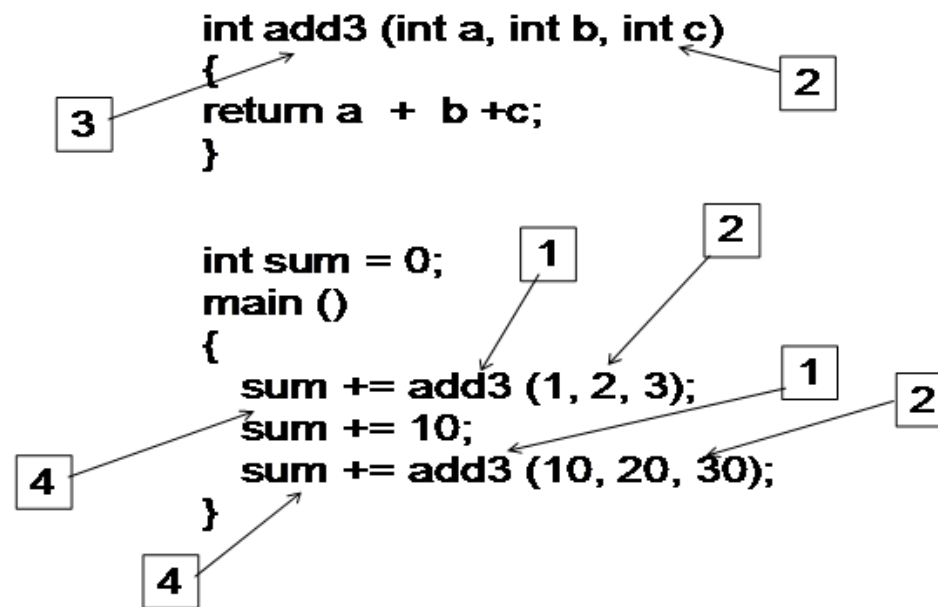
Stack

- LIFO data structure
- Stack Pointer: sp ← 0x017fff80
  - li sp, 0x017fff80

0x1000000	Instructions (.text section)	Nios V
	Statically defined data (global variables in C) (.data and .bss sections)	
	Heap: Dynamically allocated data	
	↓ ↑	
0x17fff80	Stack:	

Stack

Nios V
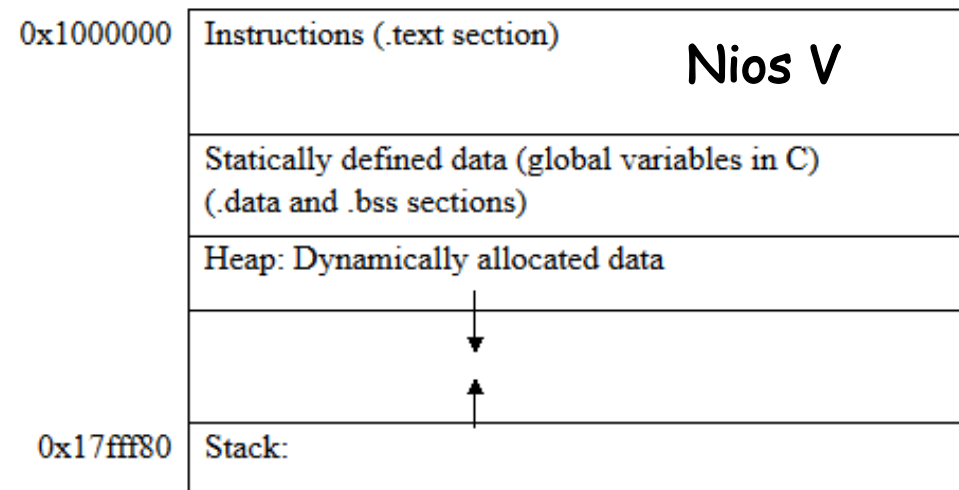
0x1000000	Instructions (.text section)
	Statically defined data (global variables in C) (.data and .bss sections)
	Heap: Dynamically allocated data
0x17fff80	Stack:

- Operations:
  - Push
    - » sw r31, 0(sp)
    - » addi sp, sp, -4
  - Pop
    - » addi sp, sp, 4
    - » lw r31, 0(sp)
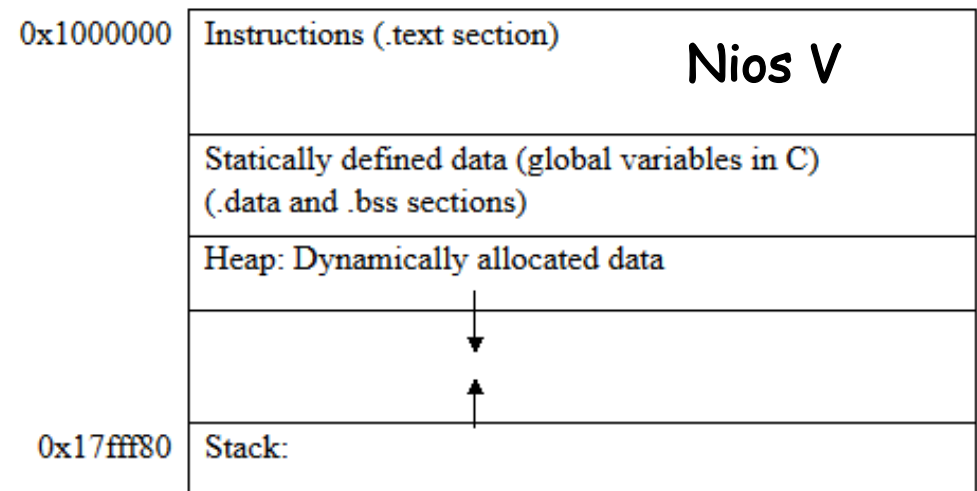
Example

sp →				
0x6fff0	01	02	03	04
0x6fff4	10	20	30	40
0x6fff8	11	22	33	44
0x6fffc	55	66	77	88

lw r9, 8(sp) -> r9 <-Mem32[sp+8] = 0x44332211

lb r10, 0xd(sp) -> r10<-Mem8[sp+0xd]=0x66

Subroutines in assembler

```
.org 0x1000
 boo:

        call coo

        ...

coo:

        ret
```

Calling to a subroutine:
$x1 = PC + 4$
$PC = coo$

Return from a subroutine,
equivalent to: j x1:
$PC = x1$

Remote Computer Architecture Laboratory (CAlab)

Local LAN network at the Computer Science building

VPN server

Internet

cable USB

Ethernet cable

Input router at the university

home

Desktop computer at the AClab with IP: <X>.<X>.<X>.<X>

Board: DE0-Nano
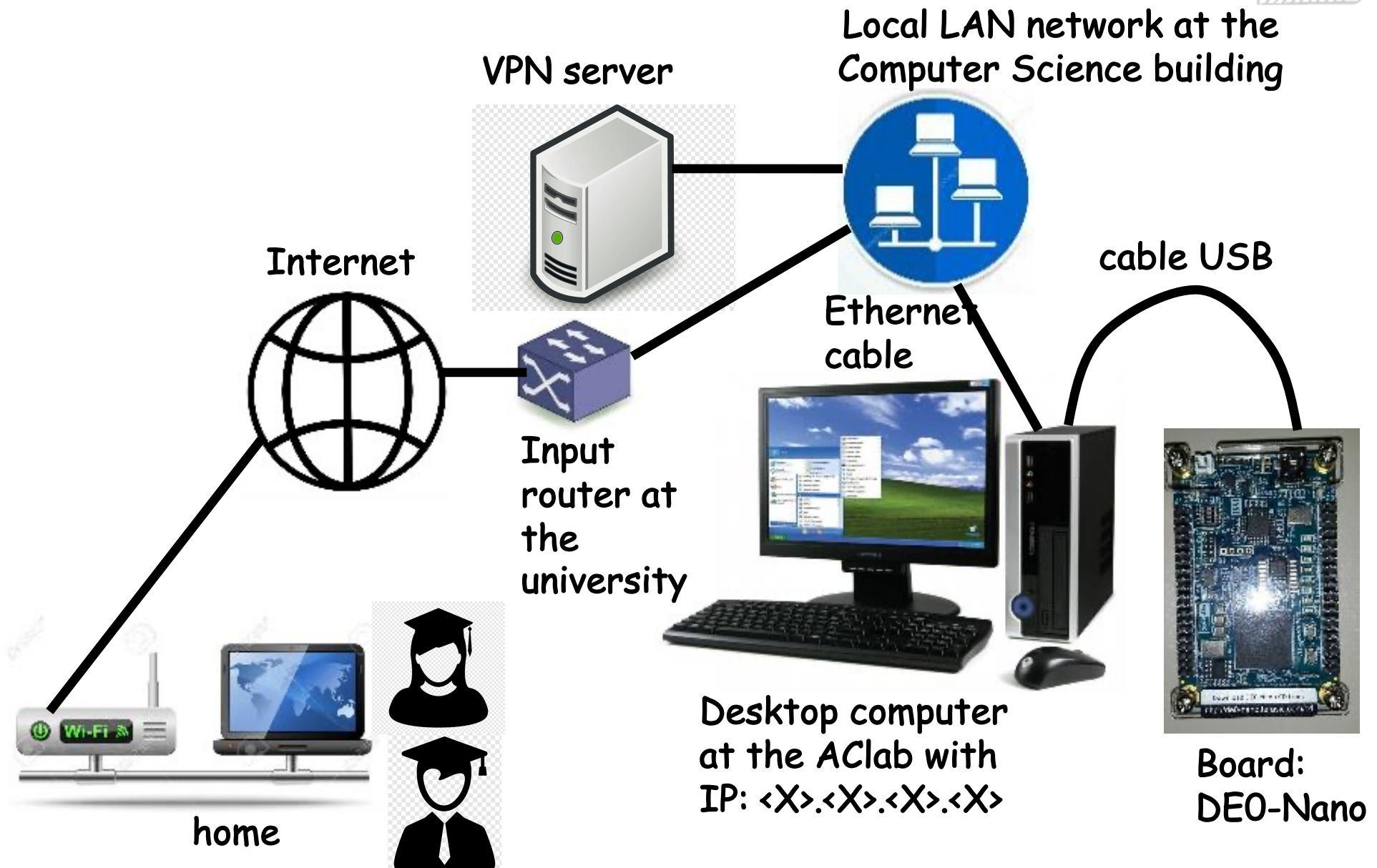
Hardware framework for Nios V/m

Quartus Prime Standard 23.1 Design Suite

Compilation Report - "NiosV/m"

Top-level Entity Name	DE0_Nano_Basic_Computer ← **SoC**
Family	**Cyclone IV E**
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	**7,829 / 22,320 (35 %)**
Total registers	**4118**
Total pins	95 / 154 (62 %)
Total virtual pins	0
Total memory bits	**135,808 / 608,256 (22 %)**
Embedded Mul. 9-bit ele	0 / 132 (0 %)
Total PLLs	1 / 4 (25 %)
FMAX (slow Model)	**119.33 MHz**

Hardware framework for Nios V/g

Quartus Prime Standard 23.1 Design Suite

Compilation Report - "NiosV/g"

Top-level Entity Name	DE0_Nano_Basic_Computer	← SoC
→ Family	Cyclone IV E	
Device	EP4CE22F17C6	
Timing Models	Final	
→ Total logic elements	11,409 / 22,320 (51 %)	
Total registers	6357	
Total pins	95 / 154 (62 %)	
Total virtual pins0		
→ Total memory bits	207,504 / 608,256 (34 %)	
Embedded Mul. 9-bit ele	8 / 132 (6 %)	
Total PLLs	1 / 4 (25 %)	
→ FMAX (slow Model)	91.22 MHz	

Part I: executing an example program

Nios V Command Shell (Quartus Prime 23.1std)

Terasic FPGA-based DE0-Nano board

Software framework

Nios V Command Shell
(Quartus Prime Standard 23.1 Design Suite)

Main commands:

Platform Designer 23.1 designs
the soft SoC system

BSP project
$ cd <BSP directory>
$ bash
$ niosv-bsp.exe -c -t=hal -s=<*.sopcinfo> settings.bsp
                |
                |--> output: settings.bsp

niosv-bsp.exe command

dbenitez@DESKTOP-928IF1H:/mnt/c/altera/12.1sp1/University_Program/NiosII_Computer_Systems/DE0-Nano/DE0-Nano_Basic_Comput
er_NiosVm/verilog/software/niosv/ACpractica1niosv/lab1_bsp$ niosv-bsp.exe -c -t=hal -s=../../../../nios_system.sopcinfo
settings.bsp
2024.10.28.10:27:41 Info: Searching for BSP components with category: os_software_element
2024.10.28.10:27:41 Info: Creating BSP settings.
2024.10.28.10:27:41 Info: Initializing SOPC project local software IP
2024.10.28.10:27:41 Info: Finished initializing SOPC project local software IP. Total time taken = 2 seconds
2024.10.28.10:27:42 Info: Searching for BSP components with category: driver_element
2024.10.28.10:27:42 Info: Searching for BSP components with category: software_package_element
2024.10.28.10:27:42 Info: Loading drivers from ensemble report.
2024.10.28.10:27:42 Info: Finished loading drivers from ensemble report.
2024.10.28.10:27:42 Info: Evaluating default script "c:\altera\23.1std\quartus\..\niosv\scripts\bsp-defaults\bsp-set-def
aults.tcl".
2024.10.28.10:27:42 Info: Tcl message: "STDIO character device is JTAG_UART"
2024.10.28.10:27:42 Info: Tcl message: "System timer device is intel_niosv_m_0"
2024.10.28.10:27:42 Info: Tcl message: "Default linker sections mapped to Onchip_memory_SRAM"
2024.10.28.10:27:42 Info: Tcl message: "No bootloader located at the reset address."
2024.10.28.10:27:42 Info: Tcl message: "Application ELF allowed to contain code at the reset address."
2024.10.28.10:27:42 Info: Tcl message: "The alt_load() facility is enabled."
2024.10.28.10:27:42 Info: Tcl message: "The .rwdata section is copied into RAM by alt_load()."
2024.10.28.10:27:42 Info: Saving BSP settings file.
2024.10.28.10:27:42 Info: Default memory regions will not be persisted in BSP Settings File.
2024.10.28.10:27:42 Info: Generated file "C:\altera\12.1sp1\University_Program\NiosII_Computer_Systems\DE0-Nano\DE0-Nano
_Basic_Computer_NiosVm\verilog\software\niosv\ACpractica1niosv\lab1_bsp\settings.bsp"
2024.10.28.10:27:42 Info: Generating BSP files in "C:\altera\12.1sp1\University_Program\NiosII_Computer_Systems\DE0-Nano
\DE0-Nano_Basic_Computer_NiosVm\verilog\software\niosv\ACpractica1niosv\lab1_bsp"
2024.10.28.10:27:42 Info: Default memory regions will not be persisted in BSP Settings File.
2024.10.28.10:27:42 Info: Generated file "C:\altera\12.1sp1\University_Program\NiosII_Computer_Systems\DE0-Nano\DE0-Nano
_Basic_Computer_NiosVm\verilog\software\niosv\ACpractica1niosv\lab1_bsp\settings.bsp"
2024.10.28.10:27:44 Info: Finished generating BSP files. Total time taken = 2 seconds

Software framework

Nios V Command Shell
(Quartus Prime Standard 23.1 Design Suite)

Main commands:

C source code

C program project
$ cd <source code directory>
$ niosv-app.exe -a=. -b=<BSP directory> -s=<*.c>   --> output: CMakeLists.txt

Makefile
$ cmake -S . -G "Unix Makefiles" -B build
                      |
                      |--> output: build directory and Makefile

$ make -C build   --> output: <*.elf>, executable file

Software framework

Nios V Command Shell
(Quartus Prime Standard 23.1 Design Suite)

Main commands:

C compiling
$ riscv32-unknown-elf-gcc.exe –I<include dir.> -c <*.c> -o <*.obj>

Assembling
$ riscv32-unknown-elf-as.exe <*.s> -o <*.s.obj> --> output: <*.s.obj>

RISC-V source code

Linking
$ riscv32-unknown-elf-ld.exe -g -o <*.elf> -T linker.x -nostdlib -e _start -u _start <*.s.obj>

--> output: <*.elf>

Software framework

Nios V Command Shell
(Quartus Prime Standard 23.1 Design Suite)

Main commands:

Dumping
$ riscv32-unknown-elf-objdump.exe -Sdtx <*.elf> > <*.elf.objdump>
                      |
                      |--> output: <*.elf.objdump>

Reporting
$ niosv-stack-report.exe -p riscv32-unknown-elf- <*.elf>

riscv32-unknown-elf-objdump.exe

```
3    lab1_part1.elf
4    architecture: riscv:rv32, flags 0x00000112:
5    EXEC_P, HAS_SYMS, D_PAGED
6    start address 0x08000000
7
8    Program Header:
9    0x70000003 off    0x000010b8 vaddr 0x00000000 paddr 0x00000000 align 2**0
10          filesz 0x0000002e memsz 0x00000000 flags r--
11   LOAD off    0x00001000 vaddr 0x08000000 paddr 0x08000000 align 2**12
12          filesz 0x000000b4 memsz 0x000000b4 flags r-x
13   LOAD off    0x000010b4 vaddr 0x080000b4 paddr 0x080000b8 align 2**12
14          filesz 0x00000004 memsz 0x00000004 flags rw-
15   LOAD off    0x000000bc vaddr 0x080000bc paddr 0x080000bc align 2**12
16          filesz 0x00000000 memsz 0x00000000 flags rw-
17
18   Sections:
19   Idx Name          Size      VMA       LMA       File off  Algn
20    0 .exceptions    00000000  08000000  08000000  000010b8  2**0
21                     CONTENTS
22    1 .text          000000b4  08000000  08000000  00001000  2**2
23                     CONTENTS, ALLOC, LOAD, READONLY, CODE
24    2 .rodata        00000000  080000b4  080000b4  000010b8  2**0
25                     CONTENTS, ALLOC, LOAD, DATA
26    3 .rwdata        00000004  080000b4  080000b8  000010b4  2**0
27                     CONTENTS, ALLOC, LOAD, DATA
28    4 .bss           00000000  080000bc  080000bc  000010bc  2**0
29                     ALLOC
30    5 .Onchip_memory_SRAM 00000000  080000bc  080000bc  000010b8  2**0
31                     CONTENTS
32    6 .Onchip_memory 00000000  09000020  09000020  000010b8  2**0
33                     CONTENTS
34    7 .riscv.attributes 0000002e  00000000  00000000  000010b8  2**0
35                     CONTENTS, READONLY
36   SYMBOL TABLE:
37   08000000 l    d  .exceptions  00000000 .exceptions
38   08000000 l    d  .text  00000000 .text
39   080000b4 l    d  .rodata  00000000 .rodata
40   080000b4 l    d  .rwdata  00000000 .rwdata
41   080000bc l    d  .bss 00000000 .bss
42   080000bc l    d  .Onchip_memory_SRAM  00000000 .Onchip_memory_SRAM
43   09000020 l    d  .Onchip_memory 00000000 .Onchip_memory
44   00000000 l    d  .riscv.attributes  00000000 .riscv.attributes
45   00000000 l    df *ABS*  00000000 lab1_part1.s.obj
46   080000b4 l       .rwdata  00000000 LEDG_bits
47   08000024 l       .text  00000000 DO_DISPLAY
48   08000078 l       .text  00000000 NO_BUTTON
```

```
67
68   Disassembly of section .text:
69
70   08000000 <_start>:
71    8000000: 10000837          lui a6,0x10000
72    8000004: 01080813          addi  a6,a6,16 # 10000010 <__alt_mem_Onchip_memory+0x7000010>
73    8000008: 100007b7          lui a5,0x10000
74    800000c: 04078793          addi  a5,a5,64 # 10000040 <__alt_mem_Onchip_memory+0x7000040>
75    8000010: 100008b7          lui a7,0x10000
76    8000014: 05088893          addi  a7,a7,80 # 10000050 <__alt_mem_Onchip_memory+0x7000050>
77    8000018: 00000997          auipc s3,0x0
78    800001c: 09c98993          addi  s3,s3,156 # 80000b4 <__tdata_end>
79    8000020: 0009a303          lw  t1,0(s3)
80
81   08000024 <DO_DISPLAY>:
82    8000024: 0007a203          lw  tp,0(a5)
83    8000028: 0008a283          lw  t0,0(a7)
84    800002c: 04028663          beqz  t0,8000078 <NO_BUTTON>
85    8000030: 00020313          mv  t1,tp
86    8000034: 00400533          add a0,zero,tp
87    8000038: 00800593          li  a1,8
88    800003c: 064000ef          jal ra,80000a0 <rotl>
89    8000040: 00050233          add tp,a0,zero
90    8000044: 00436333          or  t1,t1,tp
91    8000048: 00400533          add a0,zero,tp
92    800004c: 00800593          li  a1,8
93    8000050: 050000ef          jal ra,80000a0 <rotl>
94    8000054: 00050233          add tp,a0,zero
95    8000058: 00436333          or  t1,t1,tp
96    800005c: 00400533          add a0,zero,tp
97    8000060: 00800593          li  a1,8
98    8000064: 03c000ef          jal ra,80000a0 <rotl>
99    8000068: 00050233          add tp,a0,zero
100   800006c: 00436333          or  t1,t1,tp
101
102   08000070 <WAIT>:
103   8000070: 0008a283          lw  t0,0(a7)
104   8000074: fe029ee3          bnez  t0,8000070 <WAIT>
105
106   08000078 <NO_BUTTON>:
107   8000078: 00682023          sw  t1,0(a6)
108   800007c: 00600533          add a0,zero,t1
109   8000080: 00100593          li  a1,1
110   8000084: 01c000ef          jal ra,80000a0 <rotl>
111   8000088: 00050333          add t1,a0,zero
112   800008c: 000253b7          lui t2,0x25
113   8000090: 9f038393          addi  t2,t2,-1552 # 249f0 <_start-0x7fdb610>
114
```

niosv-stack-report.exe


```
dbenitez@DESKTOP-928IF1H:/mnt/c/altera/12.1sp1/University_Program/NiosII_Computer_Systems/DE0-Nano/DE0-Nano_Basic_Compu
er_NiosVm/verilog/software/niosv/ACpractica1niosv/lab1_bin$ make
riscv32-unknown-elf-ld.exe  -g -T ../practica1_bsp/linker.x -nostdlib -e _start -u _start --defsym __alt_stack_pointer=0
x08001F00 --defsym __alt_stack_base=0x08002000 --defsym __alt_heap_limit=0x8002000 --defsym __alt_heap_start=0x8002000
-o lab1_part1.elf lab1_part1.s.obj
riscv32-unknown-elf-ld.exe: warning: lab1_part1.elf has a LOAD segment with RWX permissions
niosv-stack-report.exe -p riscv32-unknown-elf- lab1_part1.elf
lab1_part1.elf
 * 184 B - Program size (code + initialized data).
 * 256 B - Free for stack.
 * 0 B - Free for heap.
riscv32-unknown-elf-objdump.exe  -Sdtx lab1_part1.elf > lab1_part1.elf.objdump
riscv32-unknown-elf-objcopy.exe -O binary lab1_part1.elf lab1_part1.hex
```

Software framework

Nios V Command Shell
(Quartus Prime Standard 23.1 Design Suite)

Main commands:

Verify USB connection is stablished between PC and DE0-Nano
$jtagconfig.exe
      1) USB-Blaster [USB-0]
       020F30DD   10CL025(Y|Z)/EP3C25/EP4CE22

Configuring FPGA of real board DE0-Nano
$ quartus_pgm.exe -c 1 -m JTAG -o "p;<*.sof>@1"

Downloading executable program to RAM memory
$ niosv-download.exe -g <*.elf>

Output messages on terminal
$ juart-terminal.exe   --> output: "Hello from Nios V, I am alive !!!"

Configuring the FPGA

LEDs turning on and off

Part II: Designing a simple program

Nios V Command Shell (Quartus Prime 23.1std)

Terasic FPGA-based DE0-Nano board

RISC-V example assembler program

```
lab1_part2.s

.text /* executable code follows */

.global _start
_start:

/* initialize base addresses of parallel ports */
la x15, RESULT /* x15:  point to the start of data section */
lw x16, 4(x15) /* x16:  counter, initialized with n */
addi x17, x16, 8 /* x17:  point to the first number */
lw x18, (x17) /* x18:  largest number found */

LOOP:
addi x16, x16, -1 /* Decrement the counter */
beq x16, zero, DONE /* Finished if r5 is equal to 0 */
addi x17, x17, 4 /* Increment the list pointer */
lw x19, (x17) /* Get the next number */
bge x18, x19, LOOP /* Check if larger number found */
add x18, x19, zero /* Update the largest number found */
j LOOP

DONE:
sw x18, (x15) /* Store the largest number into RESULT */

STOP:
j STOP /* Remain here if done */

.data /* software variables follow */

RESULT:
.skip 4 /* Space for the largest number found */

N:
.word 7 /* Number of entries in the list */

NUMBERS:
.word 4, 5, 3, 6, 1, 8, 2 /* Numbers in the list */

.end
```

Figure 7: Example of RISC-V program for the Nios V/m soft processor.

Open On-Chip Debugger (OpenOCD)

```
Terminal-1: OpenOCD

$ sh
$ openocd-cfg-gen ./niosv.cfg
$ openocd -f ./niosv.cfg
```

```
Terminal-2: Compile, link and configure

$ cd lab1_part2
$ sh
# compiling and link
$ make
# configuring the FPGA
$ make configure
```

```
Terminal-3: GDB debugger

$ cd lab1_part2
$ sh
$ riscv32-unknown-elf-gdb
(gdb) target extended-remote localhost:3333
(gdb) file lab1_part2.elf
(gdb) load
(gdb) info registers
```

Question 1

Question 1.
Examine the disassembled code of the `lab1_part2.elf` file (see `lab1_part2.elf.objdump` file). Note the difference in comparison with the original source code. Make sure that you understand the meaning of each instruction. Observe also that your program was loaded into memory locations with the starting address `0x08000000`. These addresses correspond to the on-chip SRAM memory, which was selected when specifying the system parameters.

Note that the pseudoinstruction `la x15, RESULT` in the original source code has been replaced with two machine instructions, `auipc a5,0x0` and `addi a5,a5,56`, which load the 32-bit address `RESULT` into register `a5` in two parts. `auipc a5,0x0` initializes the `a5` register to the current value of the PC register: `0x08000000`. `addi a5,a5,56` adds 56 =`0x38` to the `a5` register. The register `x5` is named `a5`.

- Examine the disassembled code to see the difference in comparison with the original source program. Make sure that you understand the meaning of each instruction.

Question 2

Remove the breakpoint. Then, set the Program Counter to 0x8000008, which will bypass the first two instructions which load the address RESULT into register a5. Also, set the value in register a5 to 0x8000004. Run the program.

```
Terminal-3: GDB debugger

(gdb) b *0x8000034
(gdb) continue
(gdb) info registers
```

Question 2.
What will be the result of this execution?

Part III: Modifying an instruction, bge

RISC-V Instruction Set

Core Instruction Formats

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

RV32I Base Integer Instructions

Inst	Name	FMT	Opcode	funct3	funct7	Description (C)	Note
bge	Branch ≥	B	1100011	0x5		if(rs1 >= rs2) PC += imm	

0x 8000024 : ff3958e3

1111 1111 0011 1001 0101 1000 1110 0011

1   111111 10011 10010 101   1000   1 1100011

imm[12] imm[10:5] rs2   rs1 funct3 imm[4:1] imm[11] opcode

Part III: Modifying an instruction, blt

RISC-V Instruction Set

Core Instruction Formats

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

RV32I Base Integer Instructions

Inst	Name	FMT	Opcode	funct3	funct7	Description (C)	Note
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	

0x 8000024 : ff39**4**8e3

1111 1111 0011 1001 0100 1000 1110 0011

1   111111 10011 10010 100   1000   1 1100011

imm[12] imm[10:5]   rs2   rs1 funct3 imm[4:1] imm[11] opcode

Result of Part III

```
(gdb) i r pc
pc              0x8000000        0x8000000
(gdb) c
Continuing.

Breakpoint 1, 0x08000034 in ?? ()
(gdb) i r pc
pc              0x8000034        0x8000034
(gdb) x 0x08000024
0x8000024:      0xff3948e3
(gdb) x 0x08000038
0x8000038:      0x00000001
```