VLIW DSPs, MIPS Supporting, & Eclipse Integration
First Annual gem5 User Workshop

Deyuan Guo and Hu He
Institute of Microelectronics, Tsinghua University
{guodeyuan, hehu}@tsinghua.edu.cn
December 2nd, 2012

Outline

- Part I. VLIW DSPs
- Part II. MIPS Supporting
- Part III. Eclipse Integration

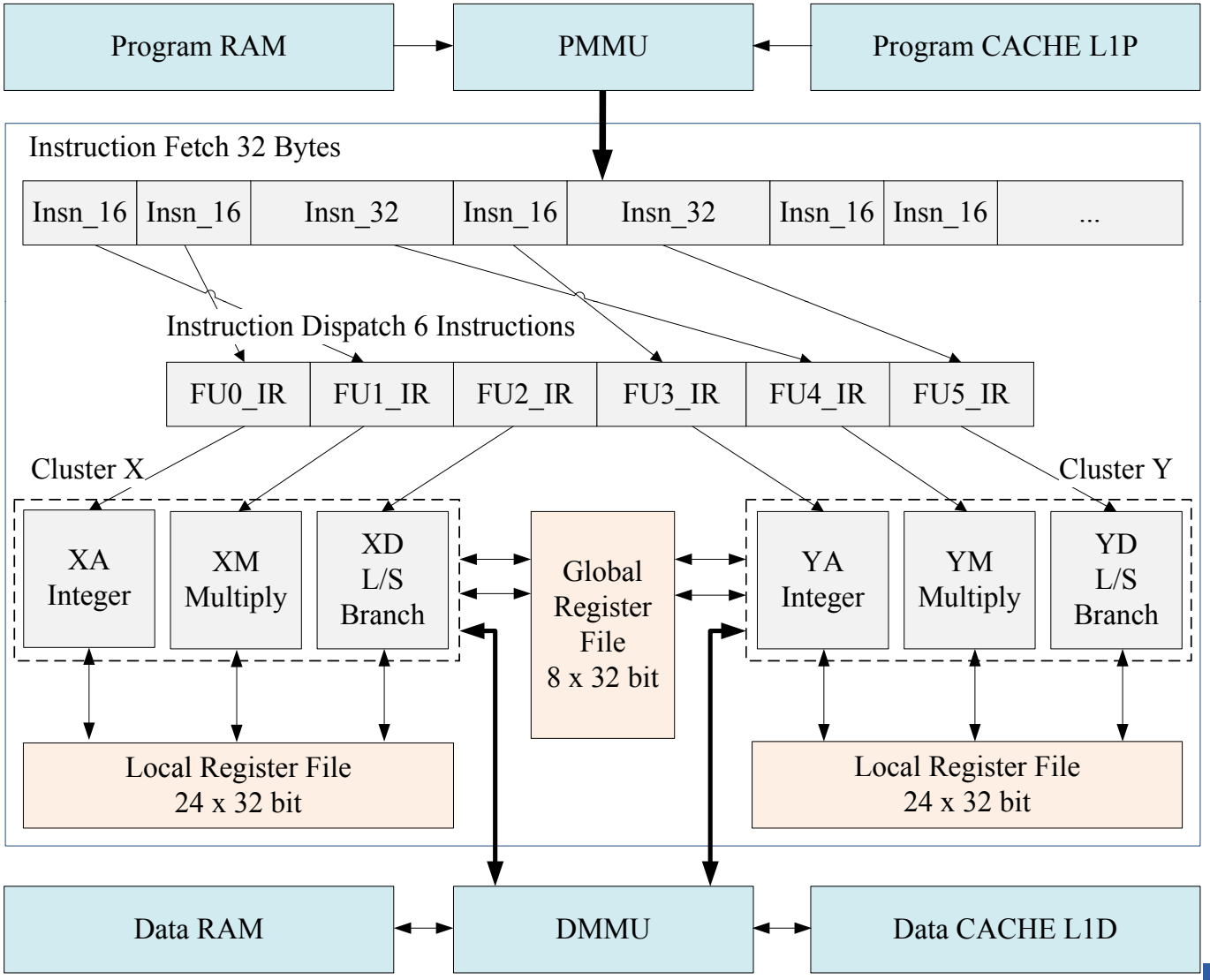
Part I. VLIW DSPs

1.1 VLIW Architecture
1.2 Key Features
1.3 Problems
1.4 VliwSimple CPU Model
1.5 VliwPipeline CPU Model
1.6 Toolchain Integration
1.7 Hardware Implementation
1.8 Benchmarks

VLIW Architecture - Lily

- Lily, 2007

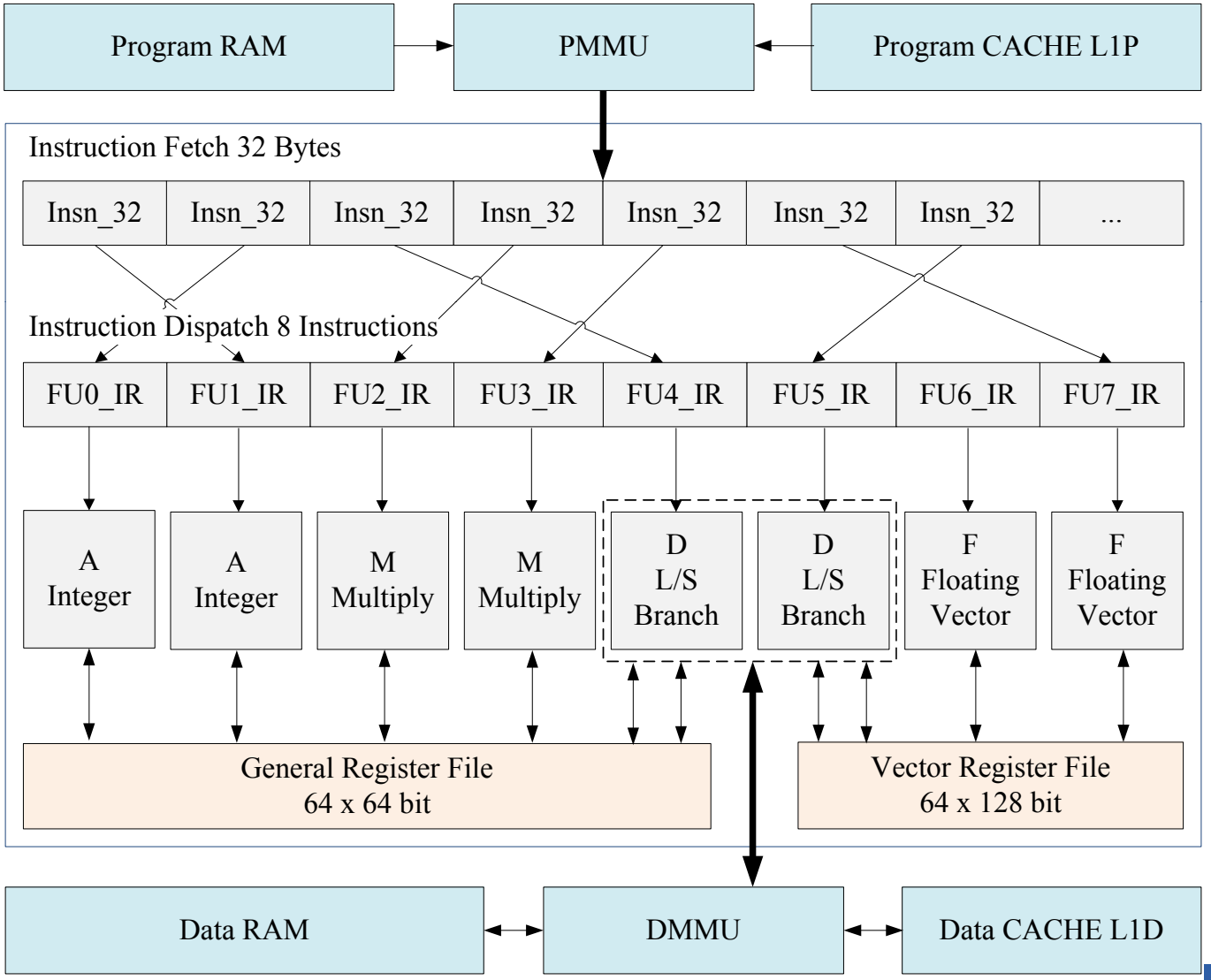
- 6-Issue VLIW
- 2 Clusters
- 16/32-bit Instruction Word



VLIW Architecture - Magnolia

- Magnolia, 2009

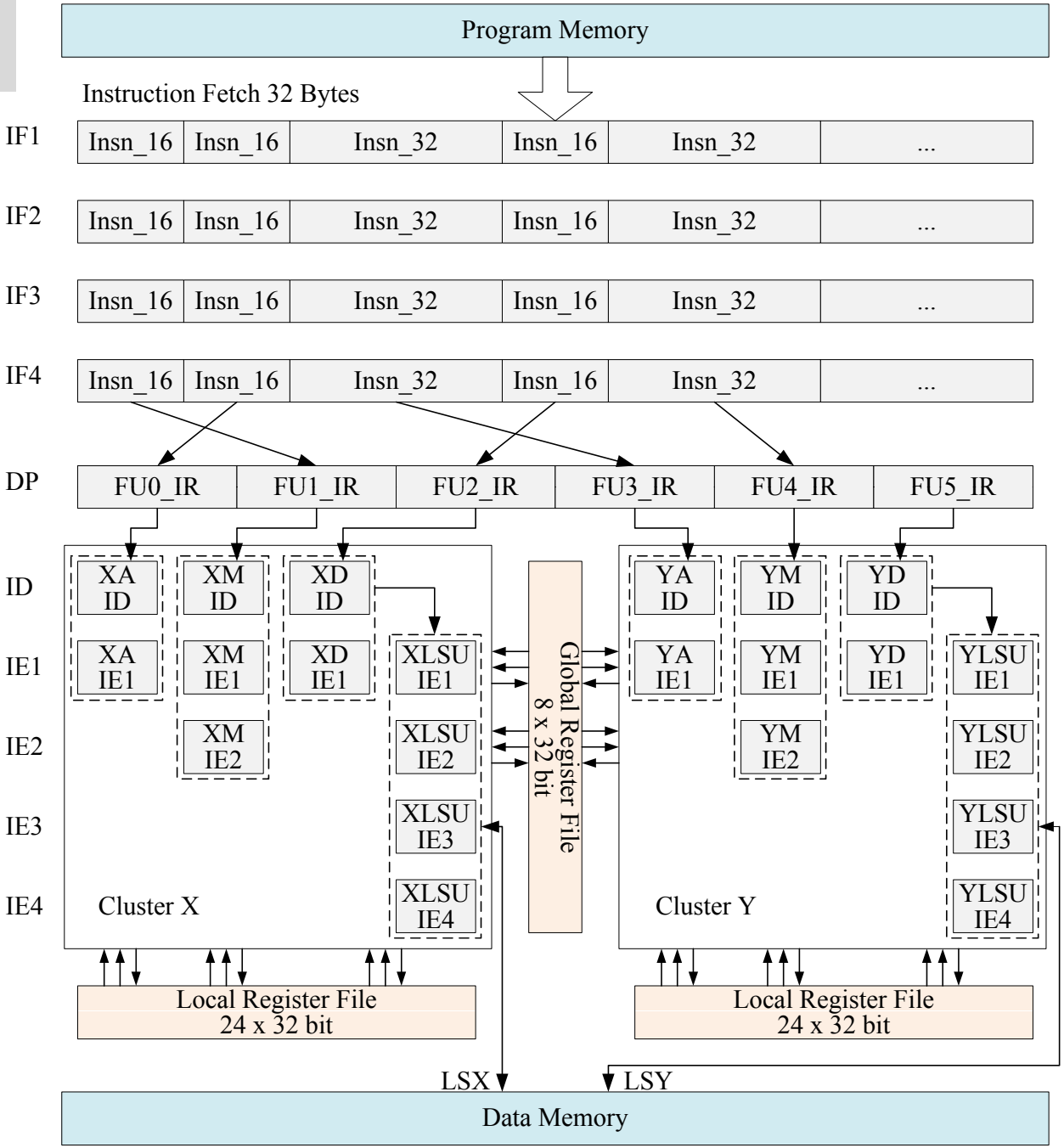
- 8-Issue VLIW
- 32-bit Instruction Word
- Vector



VLIW Pipeline

- 10 pipeline stages of Lily DSP:

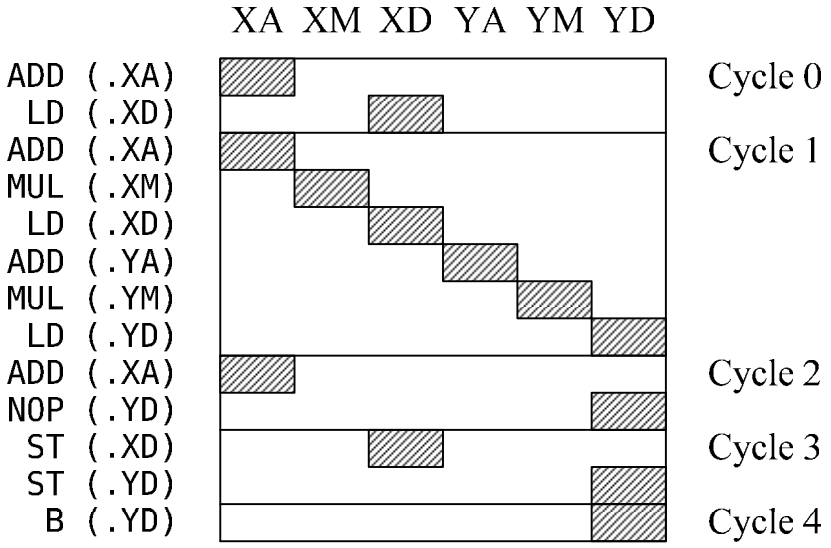
- Fetch: IF1~IF4
- Dispatch: DP
- Decode: ID
- Execute: IE1~IE4



Feature 1 - Explicit Parallelism

- Compiler is responsible for the instruction parallelism.
- Use the order of function units to indicate the instruction parallelism. If the function units of two adjacent instructions are
  - In ascending order, the two instructions will be issued concurrently.
  - Or else, the issue of the latter instruction will be delayed to next cycle.
- Benefits:
  - Save 1 bit to double the encoding space of the instruction sets.
- Side effect:
  - Not compatible with RISC execution style.
  - Some extra NOP.
- An example which cannot convert from parallel to serial:

```
   ADD (.XA) X0, X1, #4
|| ADD (.XM) X1, X0, #4
```



Instruction Parallelism Judgement
- The Order of Function Units

Feature 2 - Cycle Accurate Execution

- The execution cycle of every instruction is exposed to software level.
  - The write-back operation is delayed, same as a delay slot.
- Benefits:
  - Make the usage of registers be more efficient, relieve the tension of register files.
- Side effect:
  - Not compatible with RISC execution style.
- Another example which is not compatible with serial style:

```
(Cycle 0)  LD r0, *r1       ;Memory Load, takes 4 cycles.
(Cycle 1)  ADD r2, r2, r0   ;This r0 is the previous value.
(Cycle 2)  NOP
(Cycle 3)  NOP              ;LD write r0 at the end of cycle 3.
(Cycle 4)  ADD r3, r3, r0   ;Now r0 is the result of LD.
```

Feature 3 - Conditional Execution

- Each instruction can use a register as an executing condition.
  - The true or false of the condition depends on the value of the condition register and a true or false flag.
  - No condition or a true condition will take no effect on execution.
  - A false condition will turn an instruction into a NOP.

- Examples:
  - Used in common instructions:

```
[!r1] ADD r2, r3, r4
```

  - Used in branch instructions:

```
[r0]  B @label
```

  - Used as mutually exclusive assignment:

```
[r2]  MOV.R r3, r4
[!r2] MOV.R r3, r5
```

Feature 4 - Delayed Branch

- An ordinary branch instruction costs 6 cycles in our VLIW DSPs.
  - Without branch prediction hardware, 5 of 6 cycles are wasted in VLIW mode.
- A delayed branch instruction has 5 cycle delay slot.
  - In VLIW mode, each delay slot contains a parallel instruction group (6 or 8 instructions at most, depends on the VLIW issue width).
  - Example:

```
(     Cycle 0)    BD (.D) @label              ;A delayed branch instruction

(Slot Cycle 1)    ADD (.A) r0, r1, r2         ;Parallel instruction group
                || MUL (.M) r3, r4, r5
                || LD (.D) r6, *r7

(Slot Cycle 2)    SUB (.A) r8, r9, r10        ;Parallel instruction group
                || MUL (.M) r11, r12, r13

(Slot Cycle 3)    ...                         ;Parallel instruction group
(Slot Cycle 4)    ...                         ;Parallel instruction group
(Slot Cycle 5)    ...                         ;Parallel instruction group

(     Cycle 6)    ...                         ;Branch target instructions
```

Problems

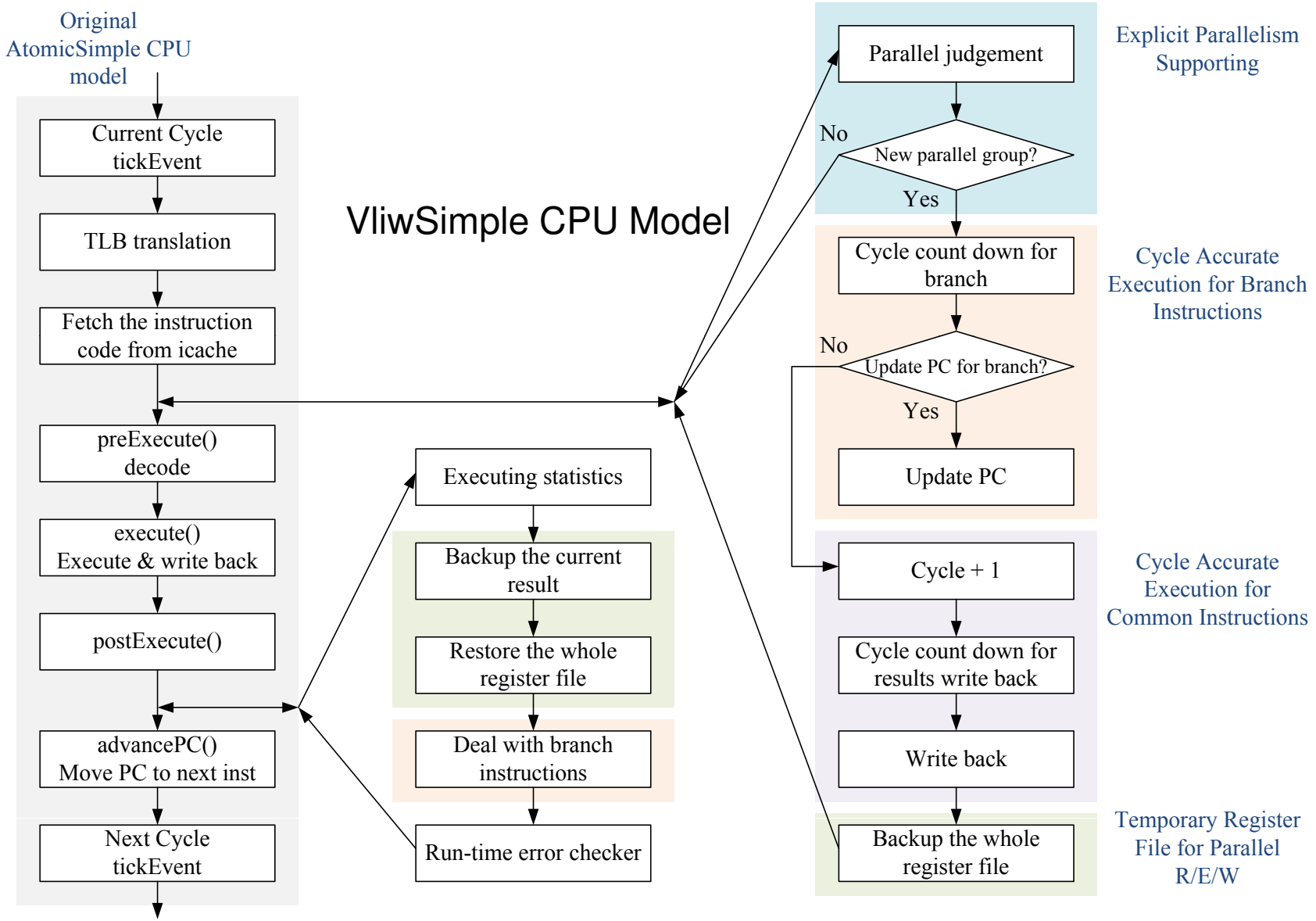
- How to implement the VLIW features in gem5?
  - The explicit parallelism.
  - The delayed write back operations.
  - The conditional execution.
  - The delayed branch operations.
- What we have done?
  - Create a new ISA folder for our DSPs in src/arch.
  - Create a VliwSimple CPU model based on AtomicSimple.
  - Create a VliwPipeline CPU model based on InOrder.
  - Run gem5 in the System-Call Emulation (SE) mode.

VliwSimple CPU Model

- Implement a VLIW extension in AtomicSimple CPU models:
  - Add the instruction parallelism judgement.
  - Use delayed write-back to deal with cycle accurate execution.
  - Delay 5 cycles to update the PC to realize the delayed branch.
  - Use temporary register file to deal with instruction parallel R/E/W, for the execute and write back operations are binded together in atomic model.
  - Use the original atomic memory read/write mechanism.
- Implement the conditional execution in ISA template.

- Advantage:
  - Minimal changes.
  - Easy to switch between VLIW and RISC.

VliwSimple CPU Model



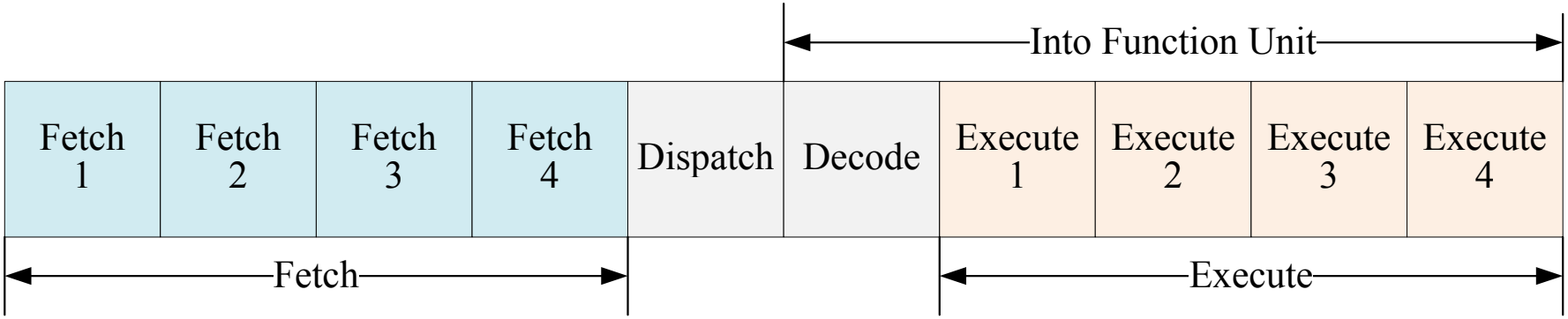
VliwSimple CPU Model

- Implement the conditional execution in gem5 ISA template.

```
def template BasicExecute {{
    Fault %(class_name)s::execute(%(CPU_exec_context)s *xc,
                                  Trace::InstRecord *traceData) const
    {
        Fault fault = NoFault;

        bool condition = false;
        if (COND_REG == 0) {
            condition = true;
        } else {
            int64_t cond_reg = xc->readCondReg(COND_REG);
            if ((NEG_COND == 0 && cond_reg != 0) || (NEG_COND == 1 && cond_reg == 0))
                condition = true;
        }

        if (condition) { ///////////////////////
        %(fp_enable_check)s;
        %(op_decl)s;
        %(op_rd)s;
        %(code)s;

        if (fault == NoFault) {
            %(op_wb)s;
        }
        } ///////////////////////////////////////
        return fault;
    }
}};
```

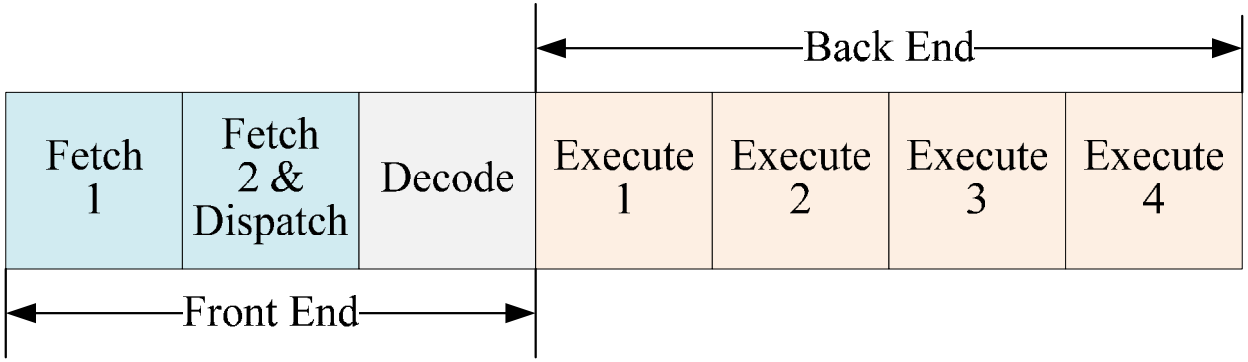
VliwPipeline CPU Model

- VLIW DSP's pipeline stage:
  - 4 stages: Fetch.
  - 1 stage: Dispatch.
  - 1 stage: Decode.
  - 4 stages: Execute.



VliwPipeline CPU Model

- Simulate the VLIW pipeline based on InOrder CPU model:
  - Simplify the fetch stages.
  - The InOrder pipeline is divided into 2 parts: Front End and Back End.



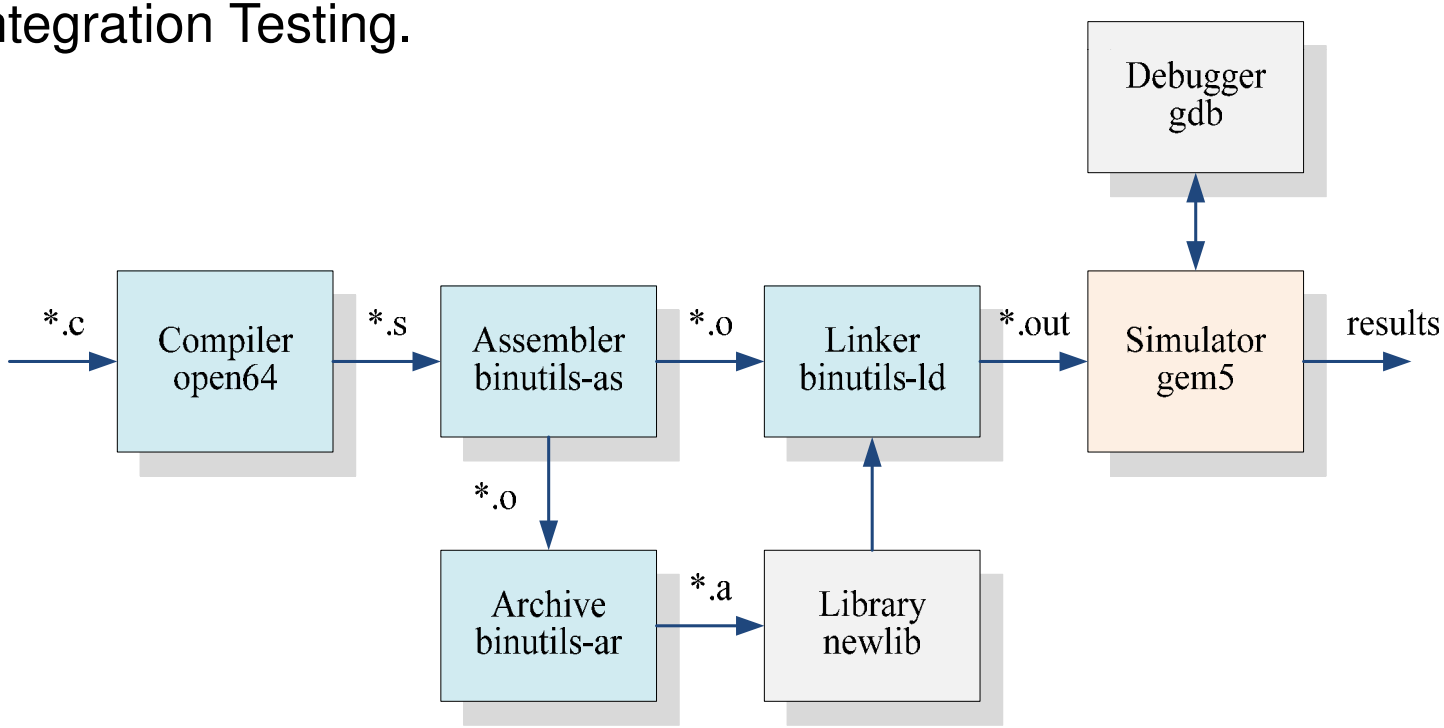
VliwPipeline CPU Model

- Simulate the VLIW pipeline based on InOrder CPU model:
  - Define the pipeline stages in cpu/inorder/cpu.cc:

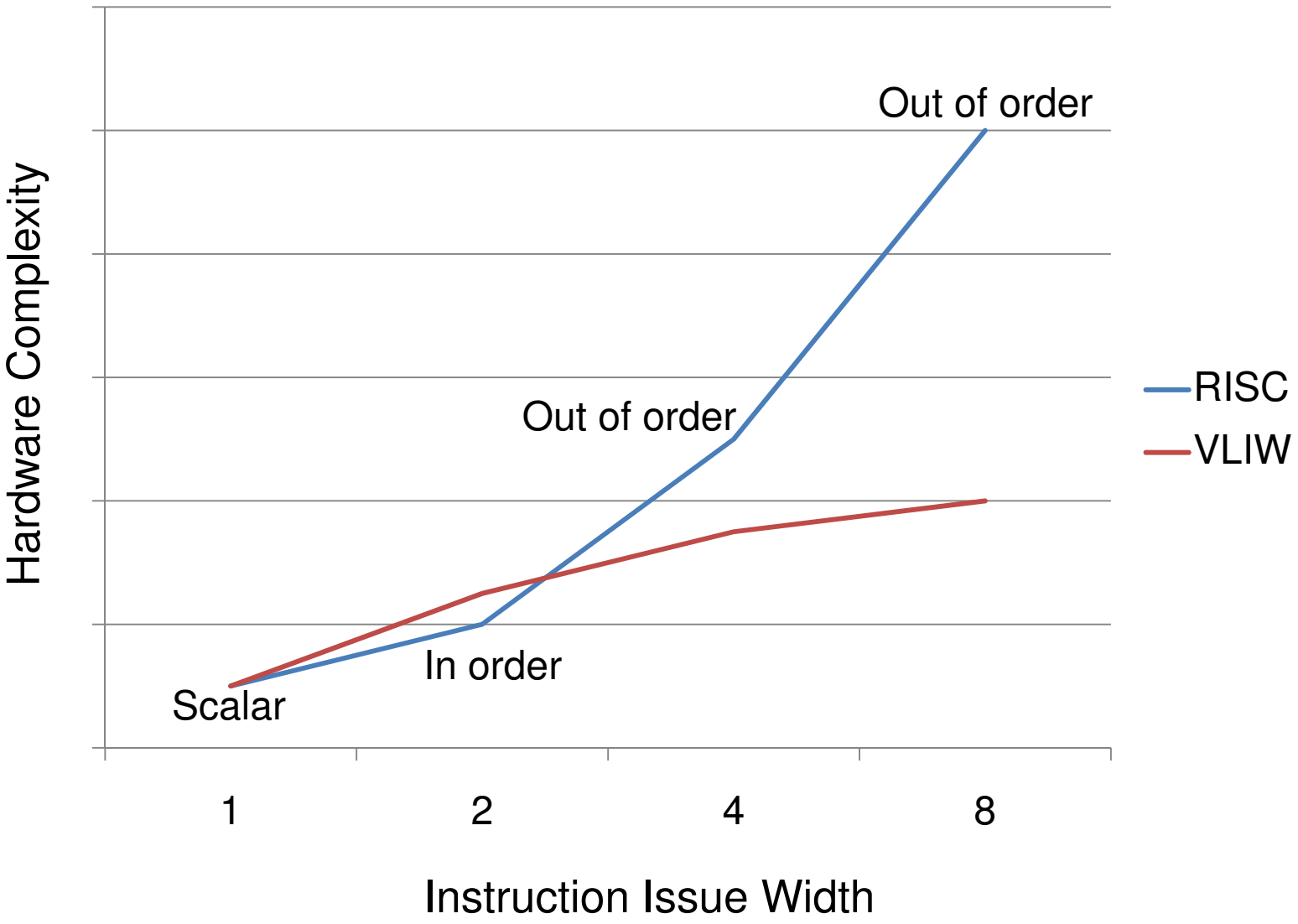
```
InOrderCPU::createFrontEndSked() {
     StageScheduler F(res_sked, stage_num++);
     StageScheduler FD(res_sked, stage_num++);
     StageScheduler D(res_sked, stage_num++);

     F.needs(FetchSeq, FetchSeqUnit::AssignNextPC);
     F.needs(ICache, FetchUnit::InitiateFetch);
     FD.needs(ICache, FetchUnit::CompleteFetch);
     D.needs(Decode, DecodeUnit::DecodeInst);
     D.needs(FetchSeq, FetchSeqUnit::UpdateTargetPC);
}

InOrderCPU::createBackEndSked() {
     StageScheduler E1(res_sked, stage_num++);
     StageScheduler E2(res_sked, stage_num++);
     StageScheduler E3(res_sked, stage_num++);
     StageScheduler E4(res_sked, stage_num++);

     E1.needs(ExecUnit, ExecutionUnit::ExecuteInst);
     E2.needs(RegManager, UseDefUnit::WriteDestReg,idx);
     ...
}
```

Toolchain Integration

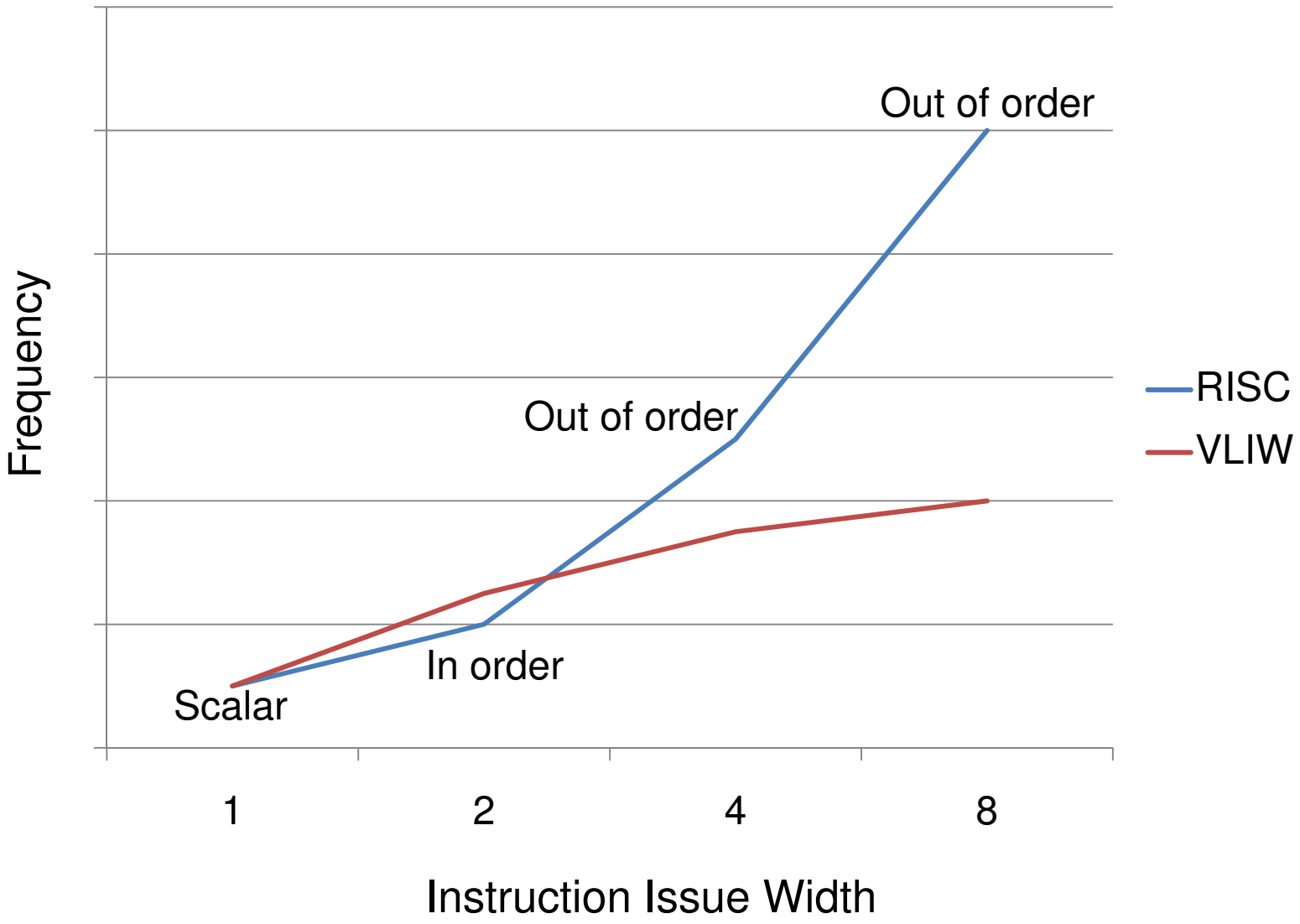
- Toolchain of the VLIW DSPs:
  - open64 - binutils - newlib - gem5 - gdb
- Some features are supported in multiple tools:
  - Syscall: in newlib and gem5.
  - DWARF: in open64, binutils and gdb.
  - ABI: in open64, gem5 and gdb.
  - Integration Testing.



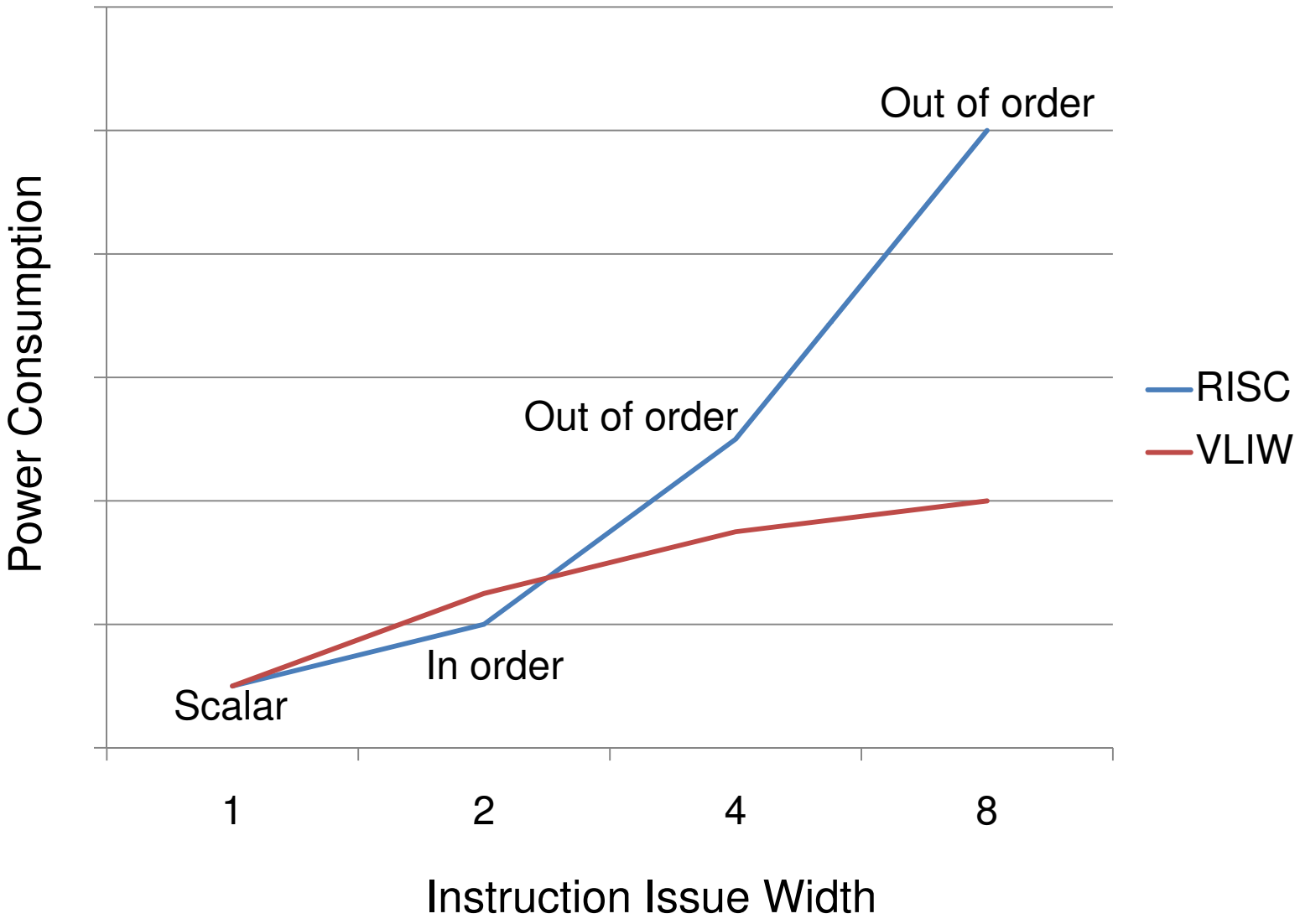
RISC or VLIW or Both?



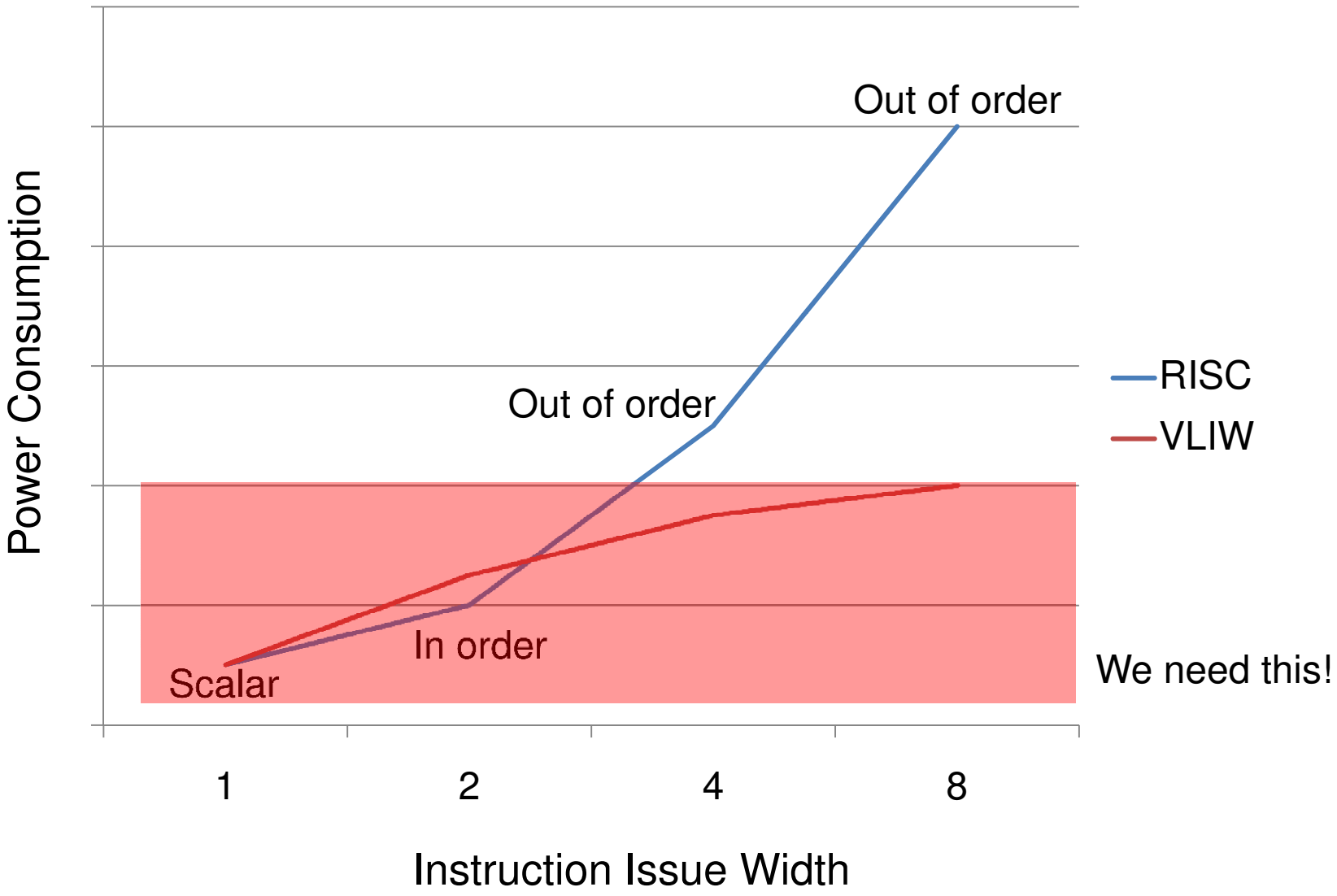
RISC or VLIW or Both?



RISC or VLIW or Both?



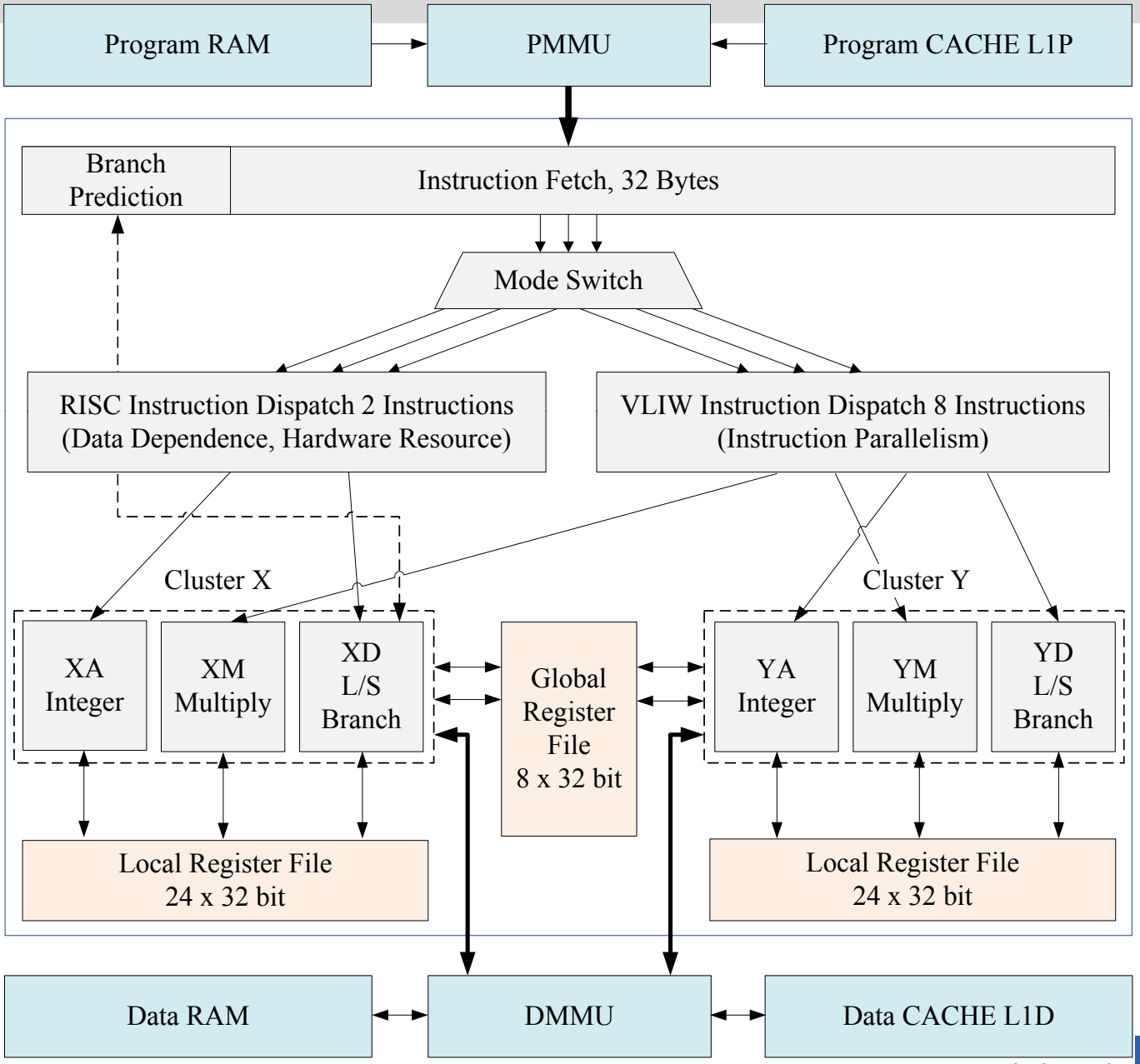
RISC or VLIW or Both?



VLIW Architecture - Lily2

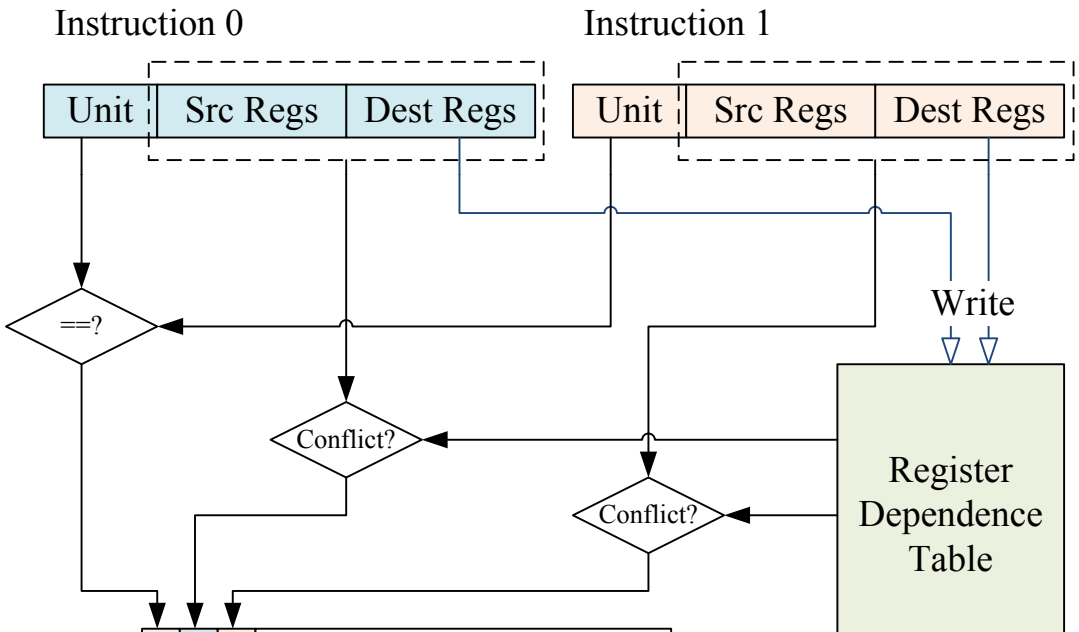
- Lily2, 2012

- 6-Issue VLIW
- 2-Issue RISC
- Dual Mode
- 16/32-bit Instruction Word



Hardware - Instruction Dependence

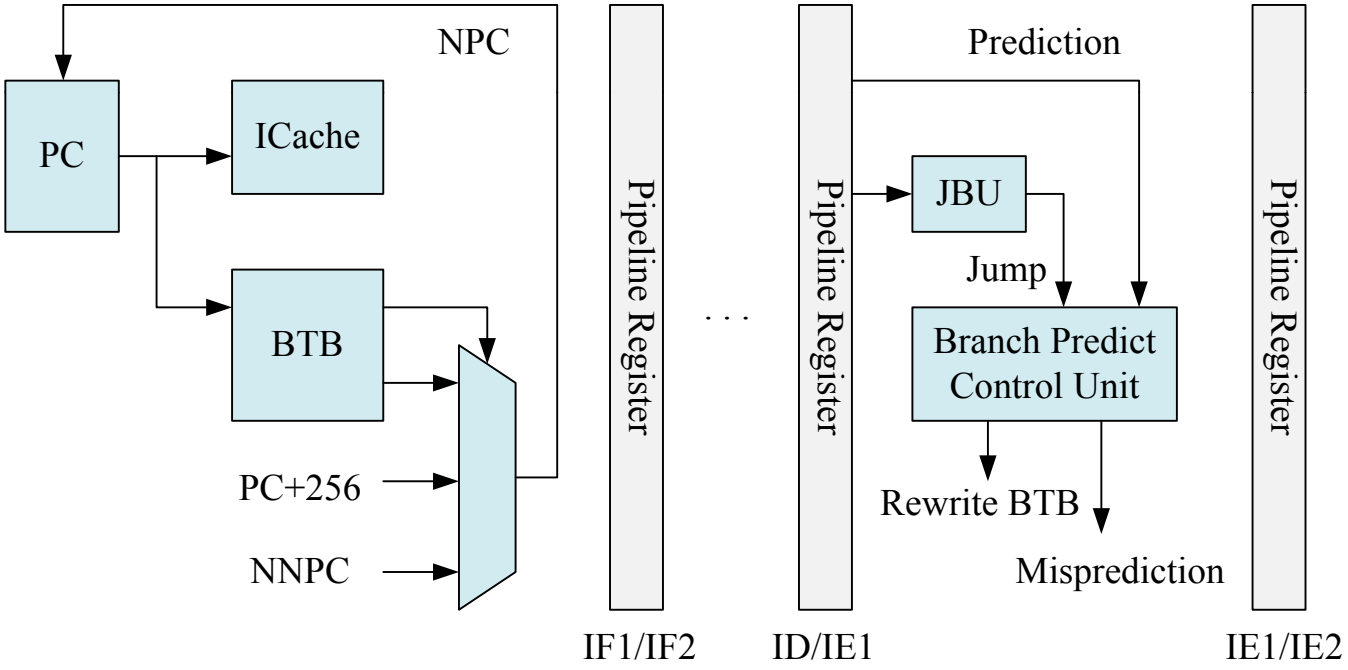
- A hardware implementation for a 2-issue in-order dispatch unit.
  - Check hardware resource conflict, e.g. function unit.
  - Check register's conflict caused by data dependence.
  - Use SETV/SETR instructions to switch between VLIW and RISC mode.



0	0	0	Dispatch Ins 0 & Ins 1
0	0	1	Dispatch Ins 0
0	1	0	Wait
0	1	1	Wait
1	0	0	Dispatch Ins 0
1	0	1	Dispatch Ins 0
1	1	0	Wait
1	1	1	Wait

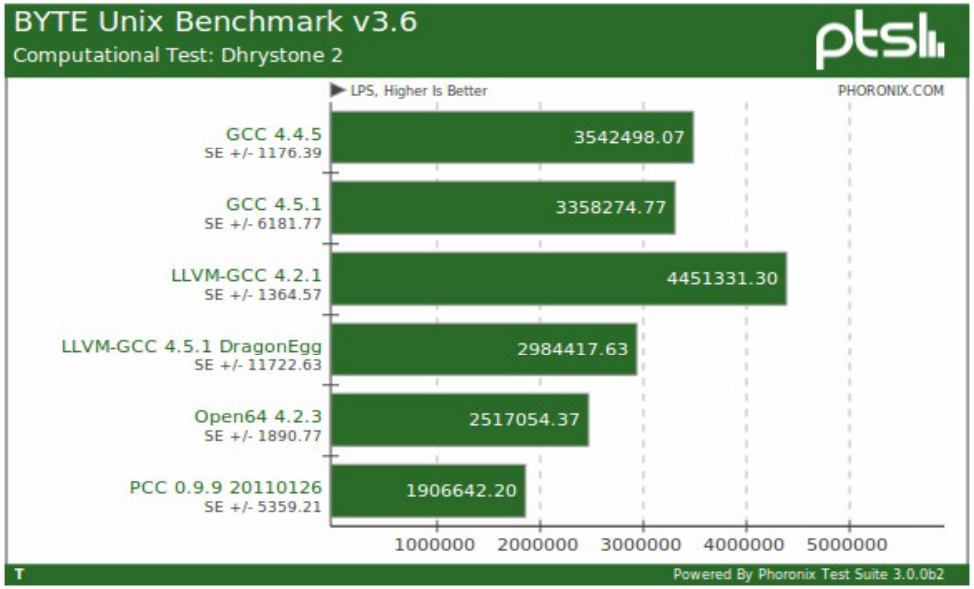
Hardware - 2-bit-predictor of Lily RISC Mode

- A hardware implementation for a 2-bit branch predictor.
  - Implemented in IF1 and IE1 pipeline stages.
  - Improve the branch performance of the RISC mode.



Benchmarks

- GCC Torture Tests
  - More than 2,000 testcases for compiling and executing.
- Dhrystone 2 (Open64 compiler)
  - Magnolia: 8-issue VLIW, each branch instruction takes 6 cycles.
    1.37 DMIPS/MHz (416 cycles per run).
  - Lily RISC mode: 2-issue in-order, 3 function units, with branch prediction.
    1.26 DMIPS/MHz (451 cycles per run).
- Library Optimization
  - The C implementation of strcmp() in the newlib takes 135 cycles.
  - The 8-issue VLIW ASM of strcmp() takes 25 cycles.
- Compiler performance
  - Open64 vs. GCC
    30% worse?



A Linux Compiler Deathmatch: GCC, LLVM, DragonEgg, Open64, Etc...
via: http://www.phoronix.com

Part II. MIPS Supporting

2.1 MIPS64

2.2 MIPS32 FS Mode

MIPS64

- Differences between MIPS64 and MIPS32:
  - Bitwidth of data, address, registers.
  - System call numbers and ABI.
  - Some MIPS32 instructions are extended to 64bit.
  - Some new 64bit instructions are added.
- Changes in gem5 source code:
  - Implement the 64-bit instructions in decoder.isa.
  - Fix some 64-bit features, e.g. register bitwidth, syscall numbers, etc.
  - Fix register file definitions and accessing interfaces in CPU models.
- Conclusions:
  - gem5 can support both 32bit and 64bit processors well.
  - gem5 has a good flexibility and portability.

MIPS32 FS Mode

- Software requirement:
  - gem5
  - busybox-1.19.3
  - linux-3.1-rc4
  - MIPS cross compiler from CodeSourcery (gcc, binutils, gdb, etc.)
- Steps:
  - Build BusyBox, which can help to create the root file system.
  - Create a root file system, within the BusyBox executable files.
  - Build the Linux kernel image, with the root file system embedded as a RAM filesystem.
  - Boot the Linux kernel in gem5.
- Hardware Solution:
  - Take ARM as a reference, like a kind of SoC.
  - Support the least hardware devices both in gem5 and kernel:
    Board: RealView PBX
    Bus: AMBA
    Interrupt Controller: GIC
    Timer: sp804
    UART: pl011

MIPS32 FS - BusyBox

- About Busybox
  - Combine many UNIX utilities into a single executable, such as coreutils, fileutils, shellutils, etc.
  - Convenient to create the root file system and Linux kernel image.
- How to build BusyBox:

```
cd busybox-1.19.3
make menuconfig
make
make install
```

- The following settings in menu configuration should be concerned:
  - `Build BusyBox as a static binary (no shared libs)`
  - `Set Cross Compiler prefix: mips-linux-gnu-`
  - Switch off some unnecessary utilities.

MIPS32 FS - Root File System

- Create the Root File System:

```
sudo mkdir root_fs
cd root_fs
```

  - Create the necessary folders:

```
sudo mkdir bin boot dev etc home lib mnt proc root sbin
sudo mkdir sys tmp usr var etc/init.d
```

  - Create the necessary files:

```
sudo touch etc/init.d/rcS
sudo touch etc/mdev.conf
sudo chmod 755 etc/init.d/rcS
sudo chmod 755 etc/mdev.conf
```

  - Create the console and null device:

```
sudo mknod -m 600 dev/console c 5 1
sudo mknod -m 666 dev/null c 1 3
```

  - Copy the BusyBox install path:

```
sudo cp busybox-1.19.3/_install/* ./ -rf
```

  - Create an init link:

```
sudo ln -s bin/busybox init
```

MIPS32 FS - Linux Kernel

- Before build the Linux kernel, we should setup the build environment:

```
TOOLCHAIN_TOP_DIR=/Absolute_Path_of_MIPS_Toolchain
export ARCH=mips
export CROSS_COMPILER=mips-linux-gnu-
export PATH=${TOOLCHAIN_TOP_DIR}/bin:$PATH
```

- How to build the Linux kernel:

```
make menuconfig
make
```

- The following settings in menu configuration should be concerned:
  - `Machine selection -> System type.`
    `Here create a new machine type for MIPS.`
  - `CPU selection -> CPU type.`
  - `Initial RAM filesystem and RAM disk (initramfs/initrd) support.`
    `Here set the absolute path of root_fs including BusyBox.`
- Changes in Linux source code:
  - Copy and modify some device drivers from arm-realview to mips-realview.
  - Not a final solution.

MIPS32 FS - gem5

- Changes in gem5 source code:
  - Copy some device implementations from ARM to MIPS.
    `src/dev/mips/RealView.py`
    `src/dev/mips/realview.cc/hh`
    `src/dev/mips/gic.cc/hh`
    `src/dev/mips/amba_device.cc/hh`
    `src/dev/mips/amba_fake.cc/hh`
    `src/dev/mips/rv_ctrl.cc/hh`
    `src/dev/mips/pl011.cc/hh`
    `src/dev/mips/timer_sp804.cc/hh`
  - Copy some system configurations from ARM to MIPS.
    `src/arch/mips/linux/system.cc/hh`
  - Fix the function of TLB, Interrupt, etc.

- Use gdb and m5term to debug the kernel.
- Because there might not be a RealView board for MIPS, this is not a final solution.

MIPS32 FS - Boot the Linux Kernel in gem5

- Run the MIPS_FS mode in gem5:

```
./build/MIPS_FS/gem5.opt configs/example/fs.py
```

- Use m5term to monitor the gem5:

```
./util/term/m5term localhost 3456
```

```
==== m5 slave terminal: Terminal 0 ====
Linux version 3.1.0-rc4-svn12 (guody@guody-Studio-1458) (gcc
version 4.5.2 (Sourcery CodeBench Lite 2011.03-93) ) #10 Thu Sep 27
19:05:40 HKT 2012
sched_clock: 32 bits at 24MHz, resolution 41ns, wraps every
178956ms
...
Please press Enter to activate this console.

BusyBox v1.19.3 (2012-09-26 18:04:12 HKT) built-in shell (ash)
    Enter 'help' for a list of built-in commands.

#
# ls
boot     etc      init     linuxrc  proc     sbin     tmp      var
bin      dev      home     lib      mnt      root     sys      usr
#
```

Part III. Eclipse Integration

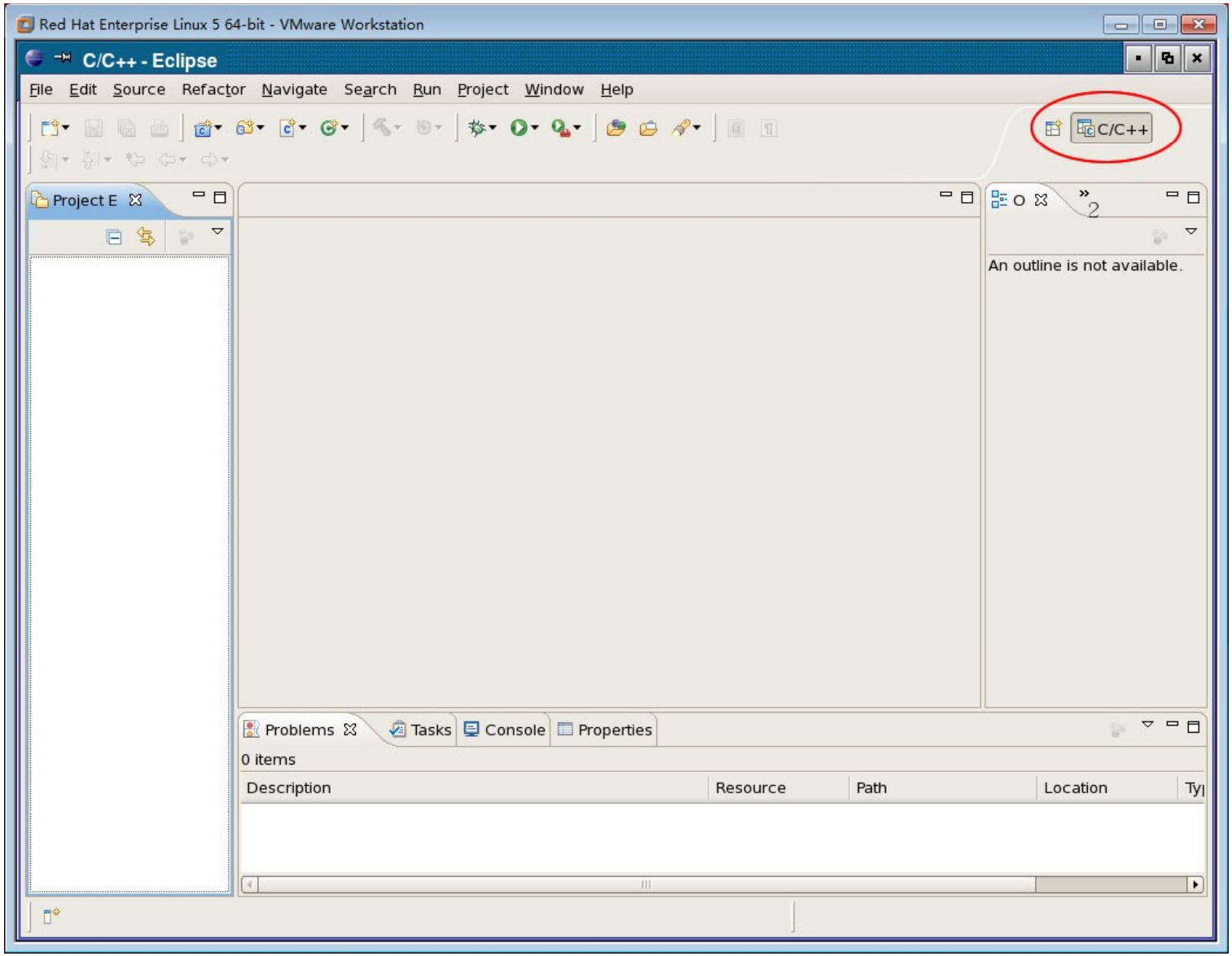
3.1 Overview
3.2 Configuration Steps

Eclipse Integration Overview

- Eclipse
  - A multi-language software development environment.
  - An integrated development environment (IDE).
  - An extensible plug-in system.
- Eclipse CDT
  - Plug-in: C/C++ Development Tools for Eclipse
- Goals:
  - Constitute a graphical cross-platform development environment.
  - Use gem5 as simulator and gdb server.
  - Integrate the cross toolchain into Eclipse:
    gem5, gcc, binutils, gdb, etc.

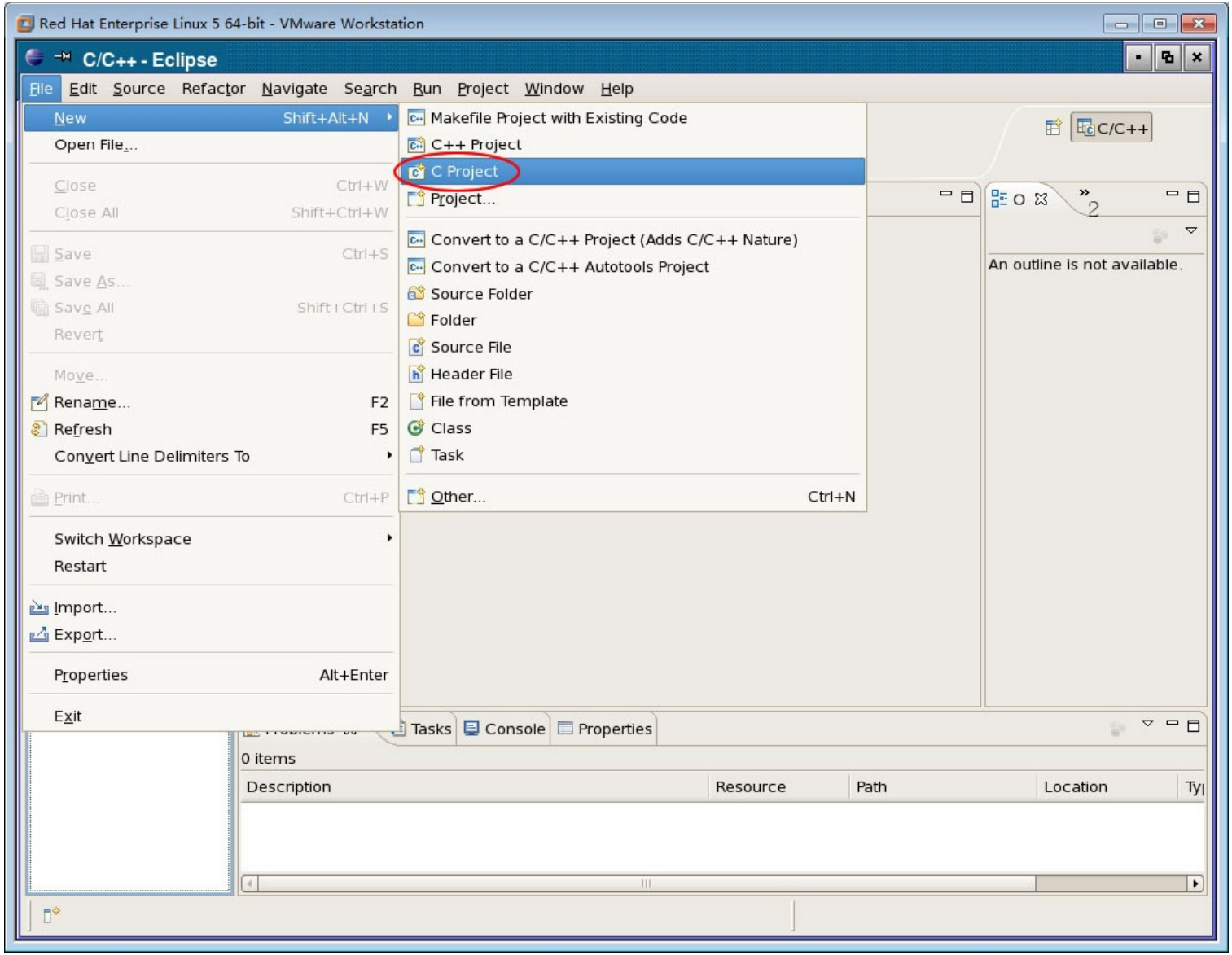
Eclipse Integration

1. The C/C++ perspective in Eclipse.



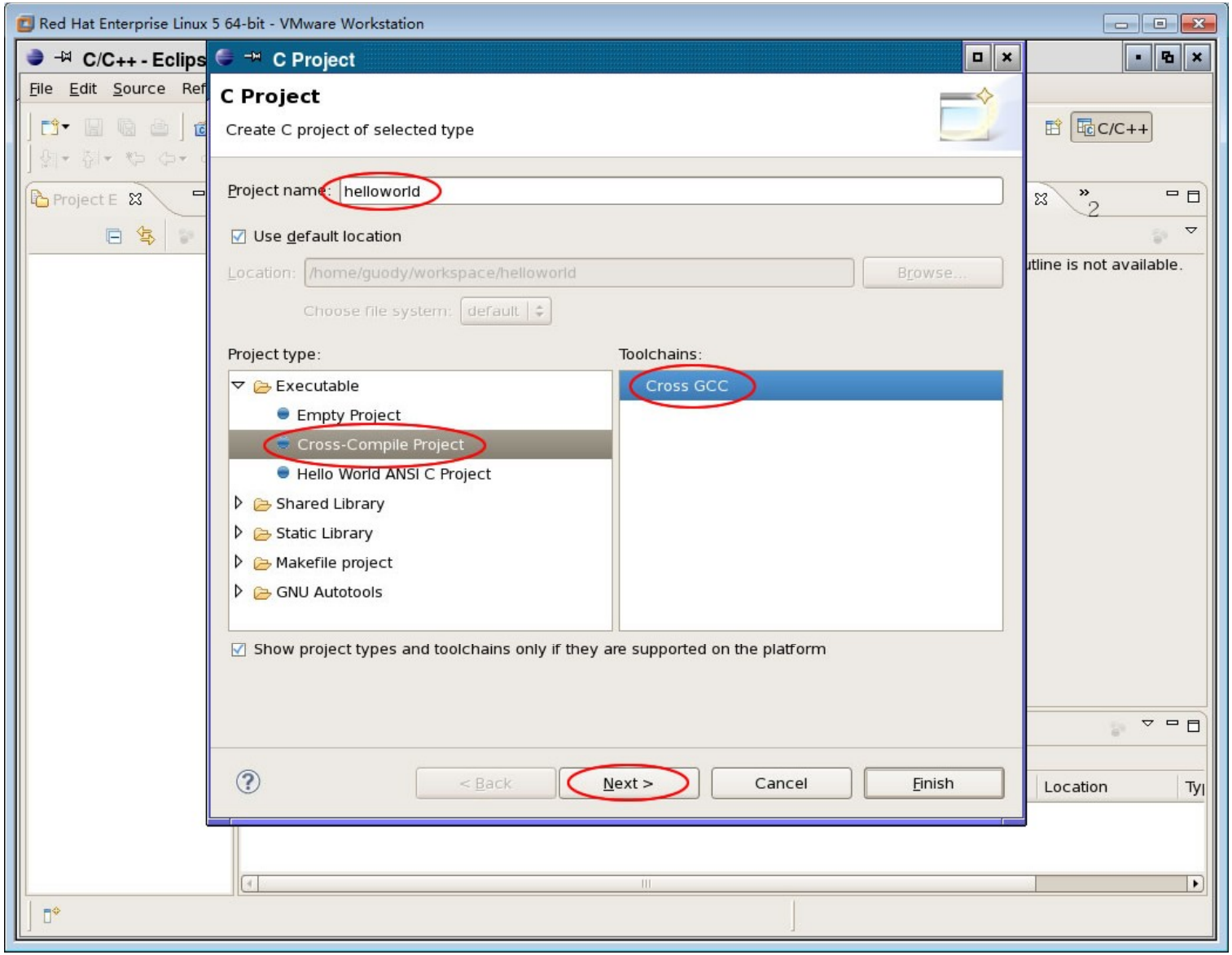
Eclipse Integration

2. Create a new C project.



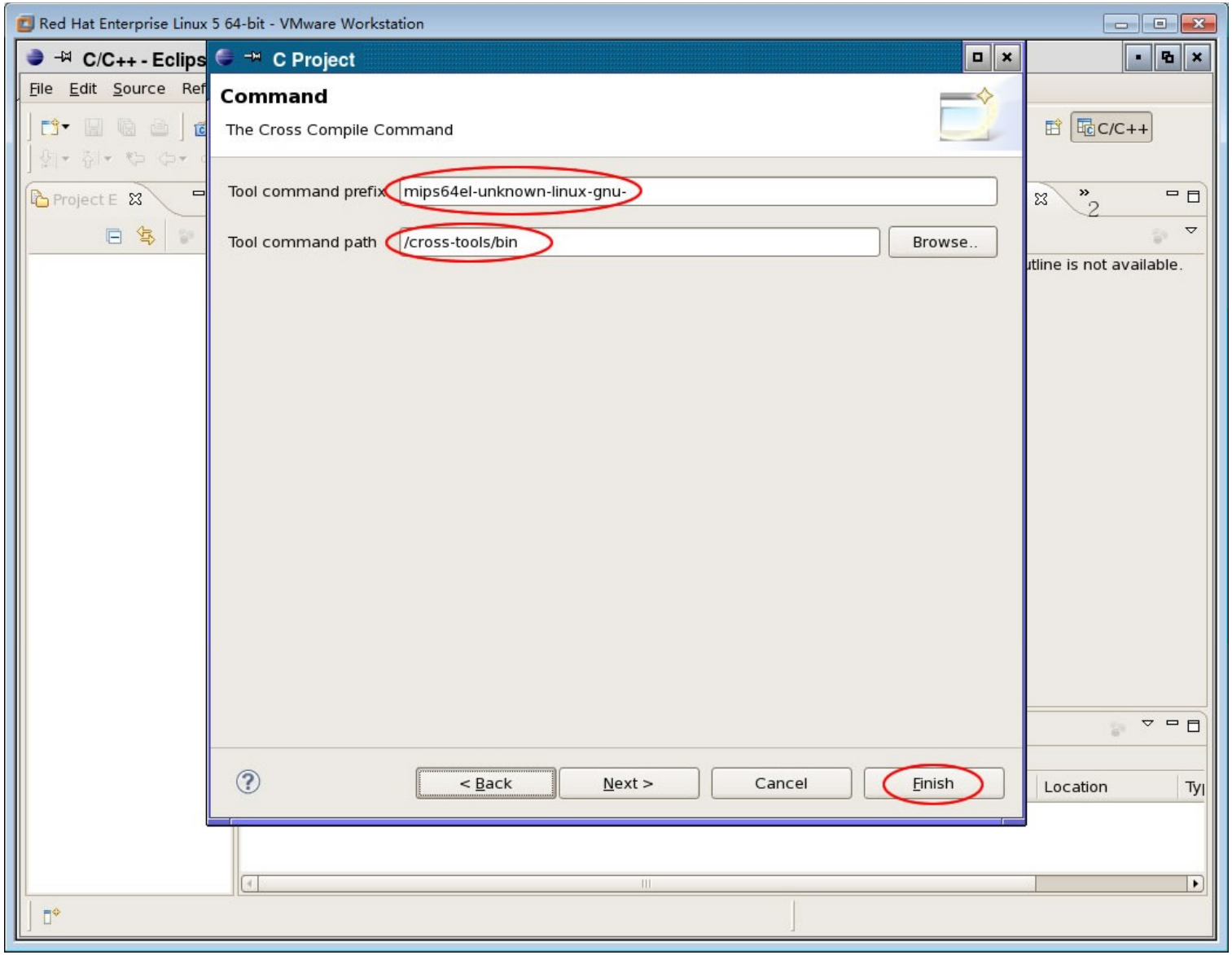
Eclipse Integration

3. Create a cross-compile project named “helloworld”.



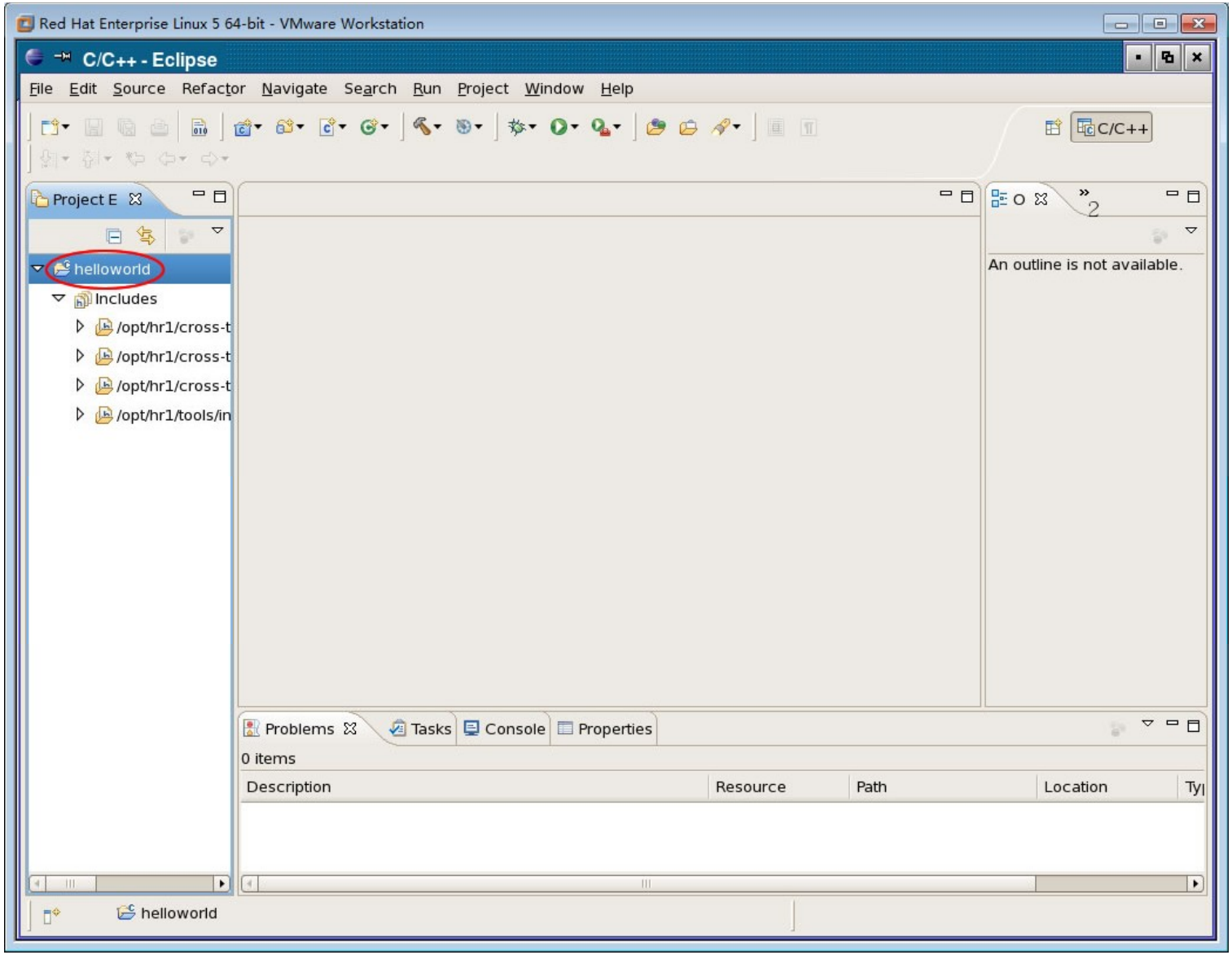
Eclipse Integration

4. Set the prefix and path of the cross compiler, assembler, linker, etc.



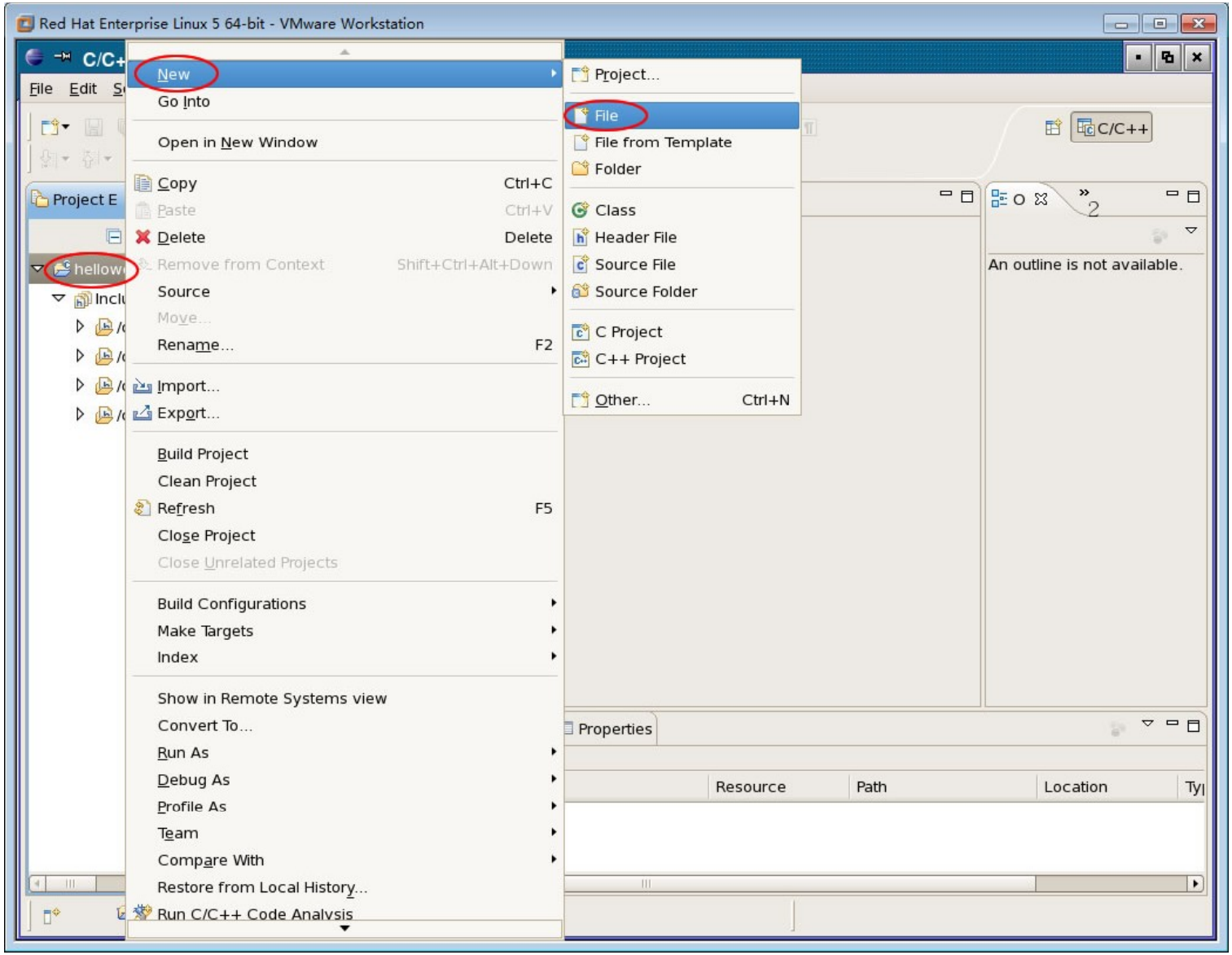
Eclipse Integration

5. Now we can see the “helloworld” project in the Project Explorer.



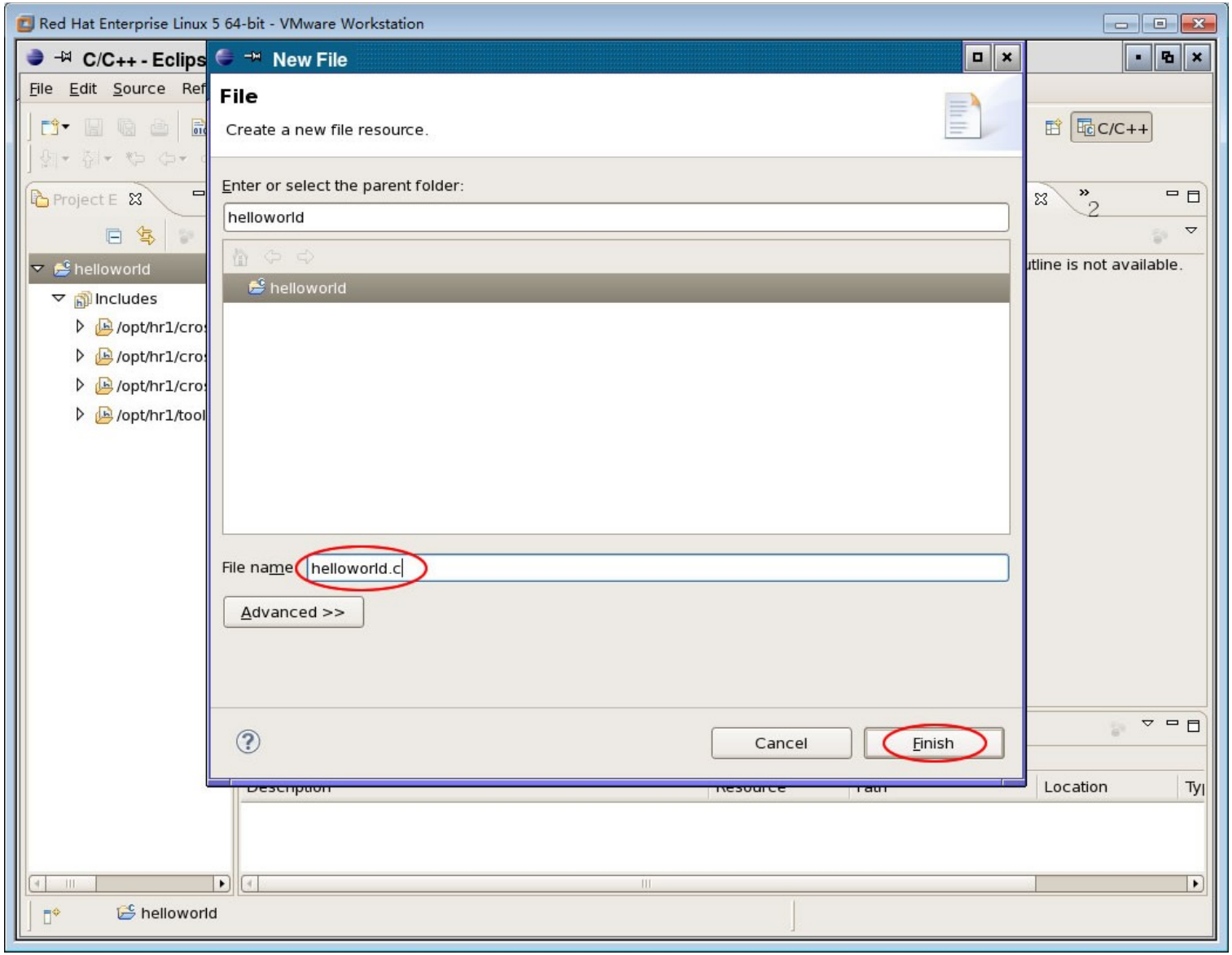
Eclipse Integration

6. Right click on the project name and create a new file.



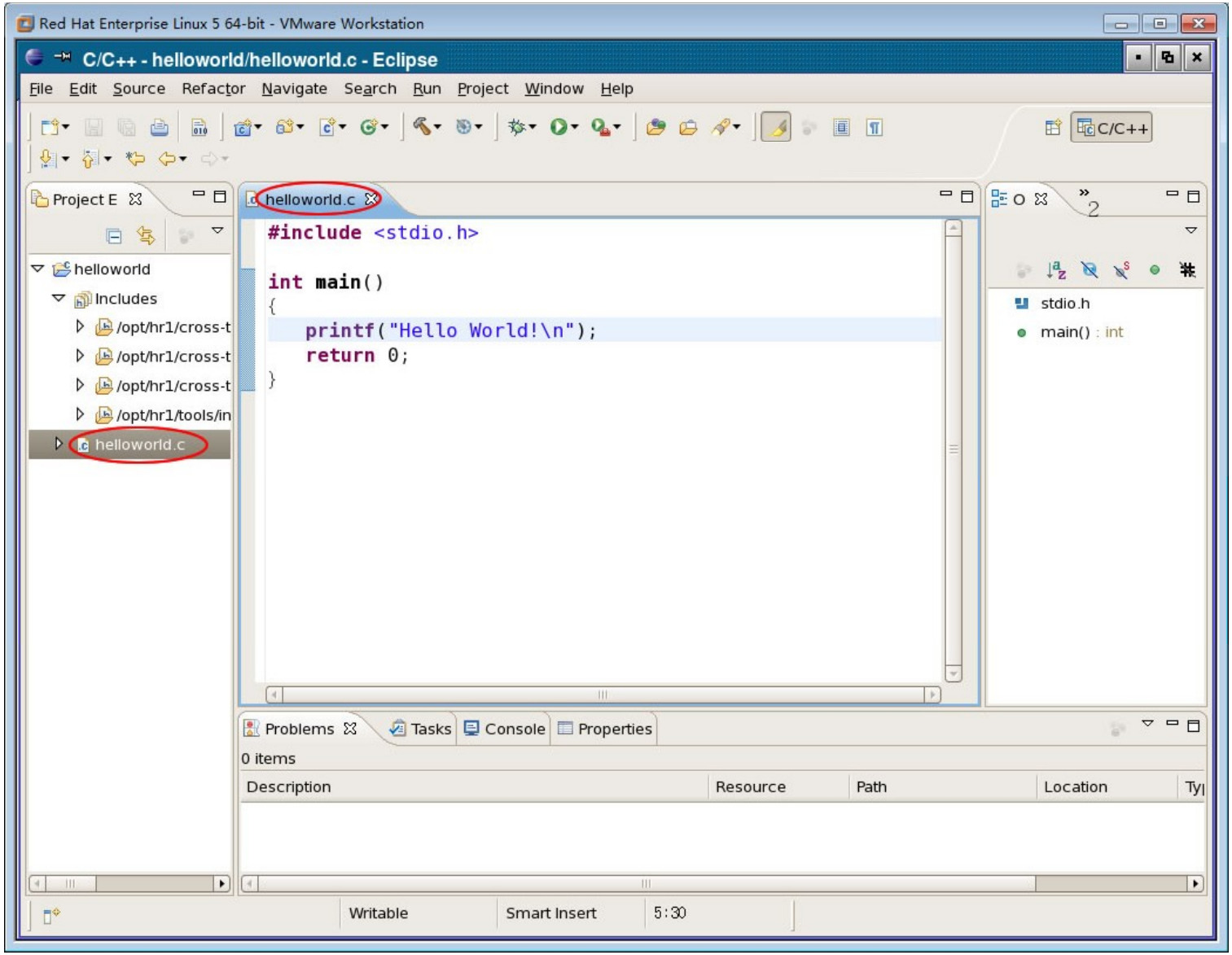
Eclipse Integration

7. Create helloworld.c.



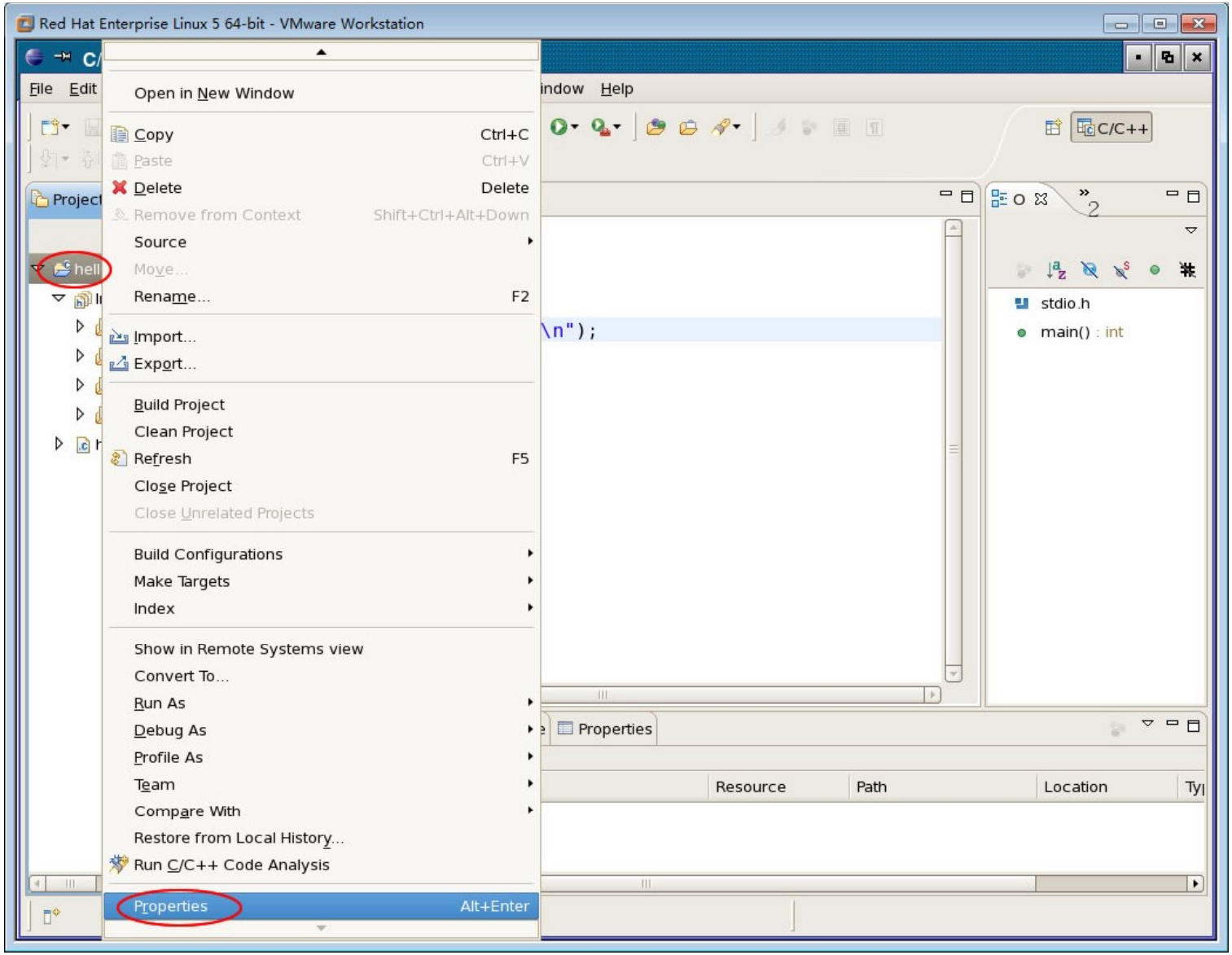
Eclipse Integration

8. Write some C codes.



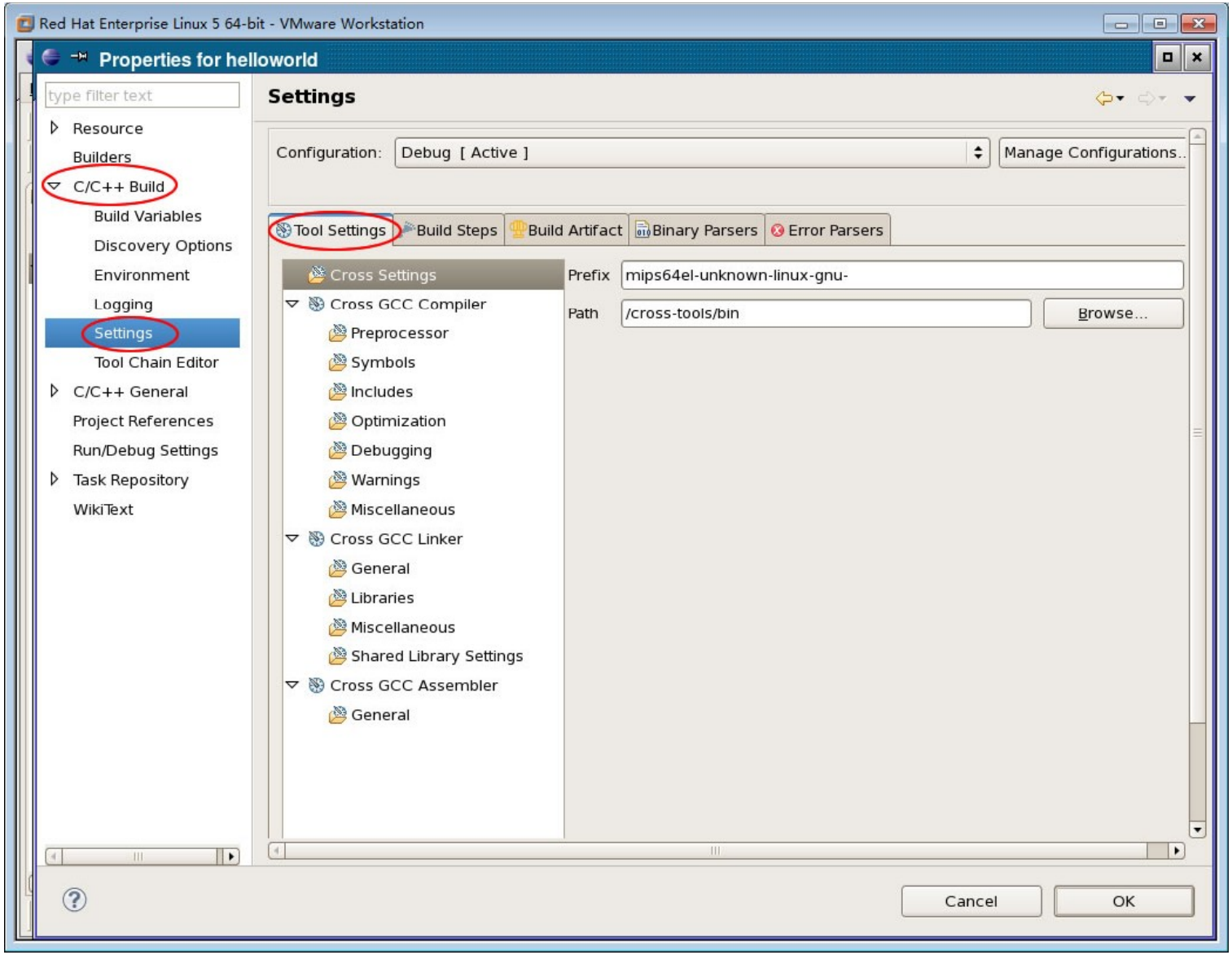
Eclipse Integration

9. Right click on the project name and select “Properties”.



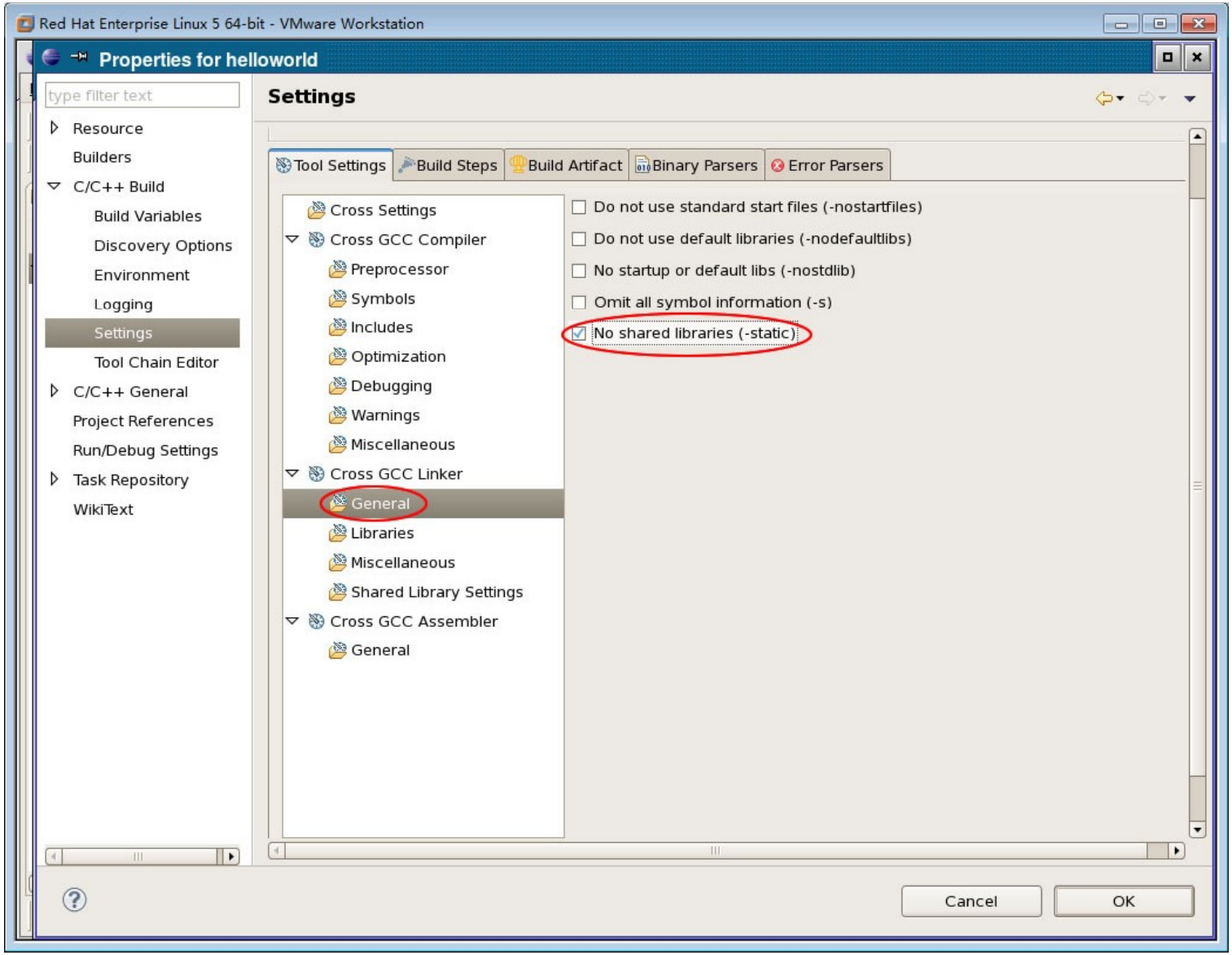
Eclipse Integration

10. Select “C/C++ Build” -> “Settings” -> “Tool Settings”.



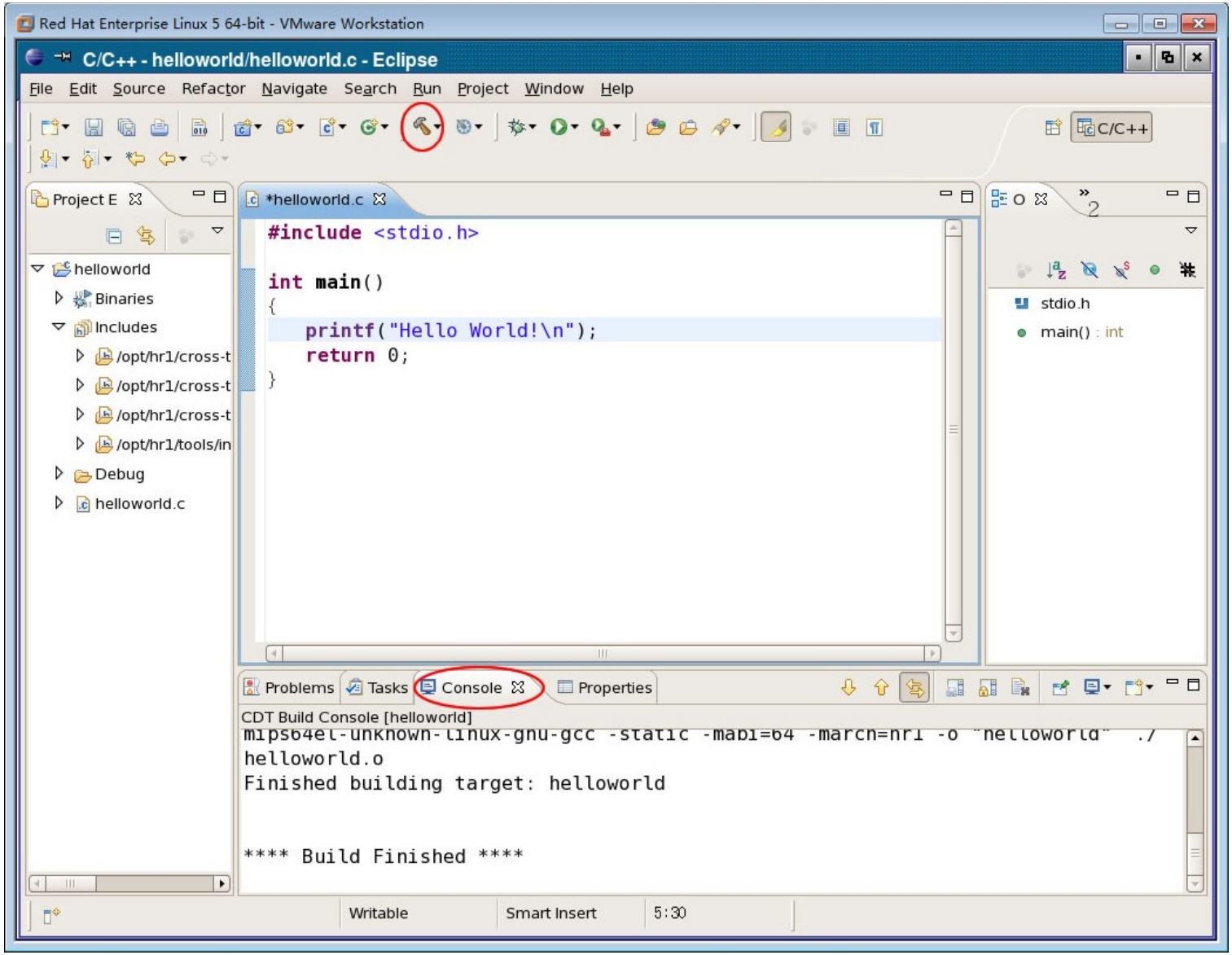
Eclipse Integration

11. Configure the linker to use static library for the SE mode of gem5.



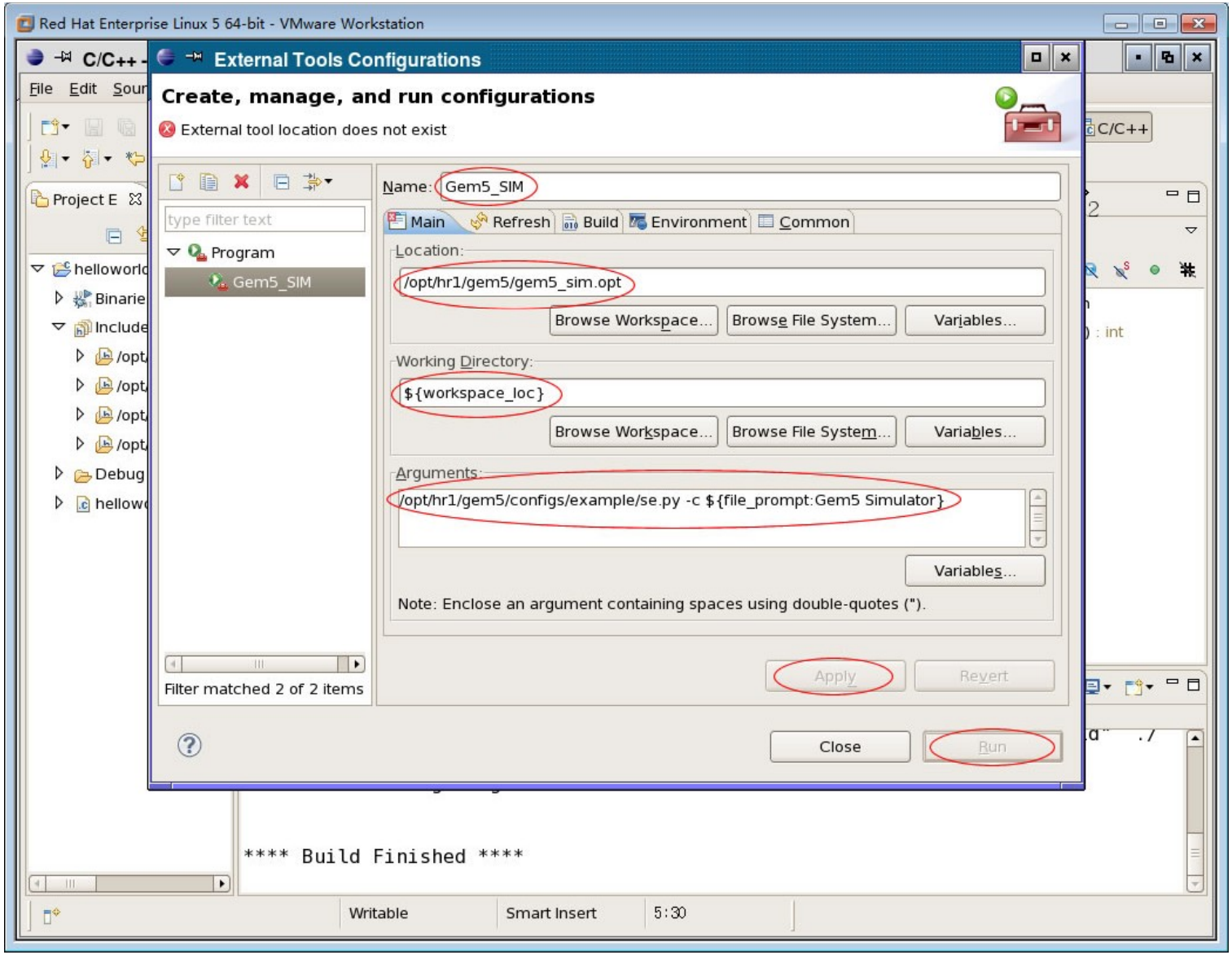
Eclipse Integration

12. Click the “Build” button, information shows in the Console window.



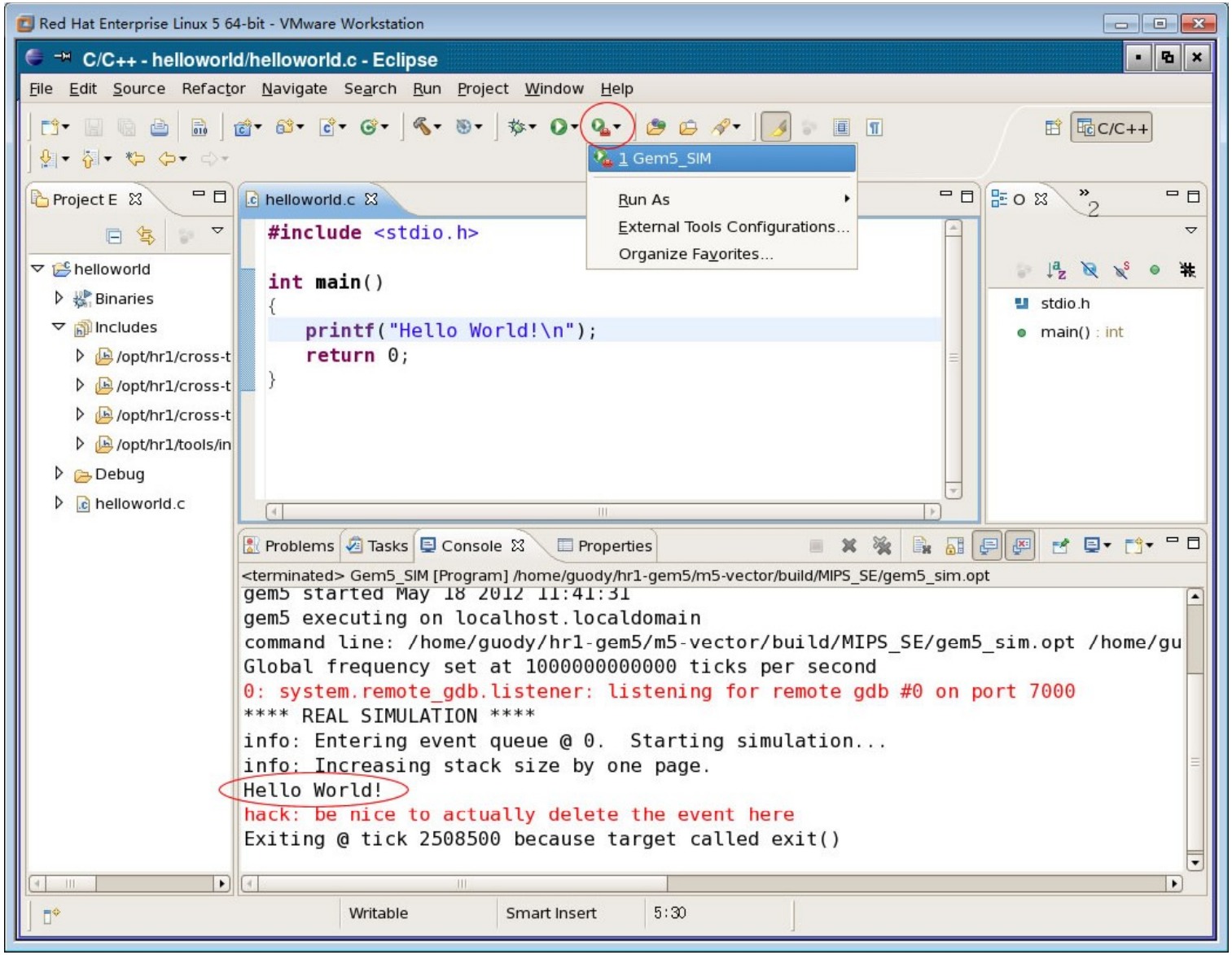
Eclipse Integration

13. Configure gem5 as an external tool. Set the path and arguments for gem5.



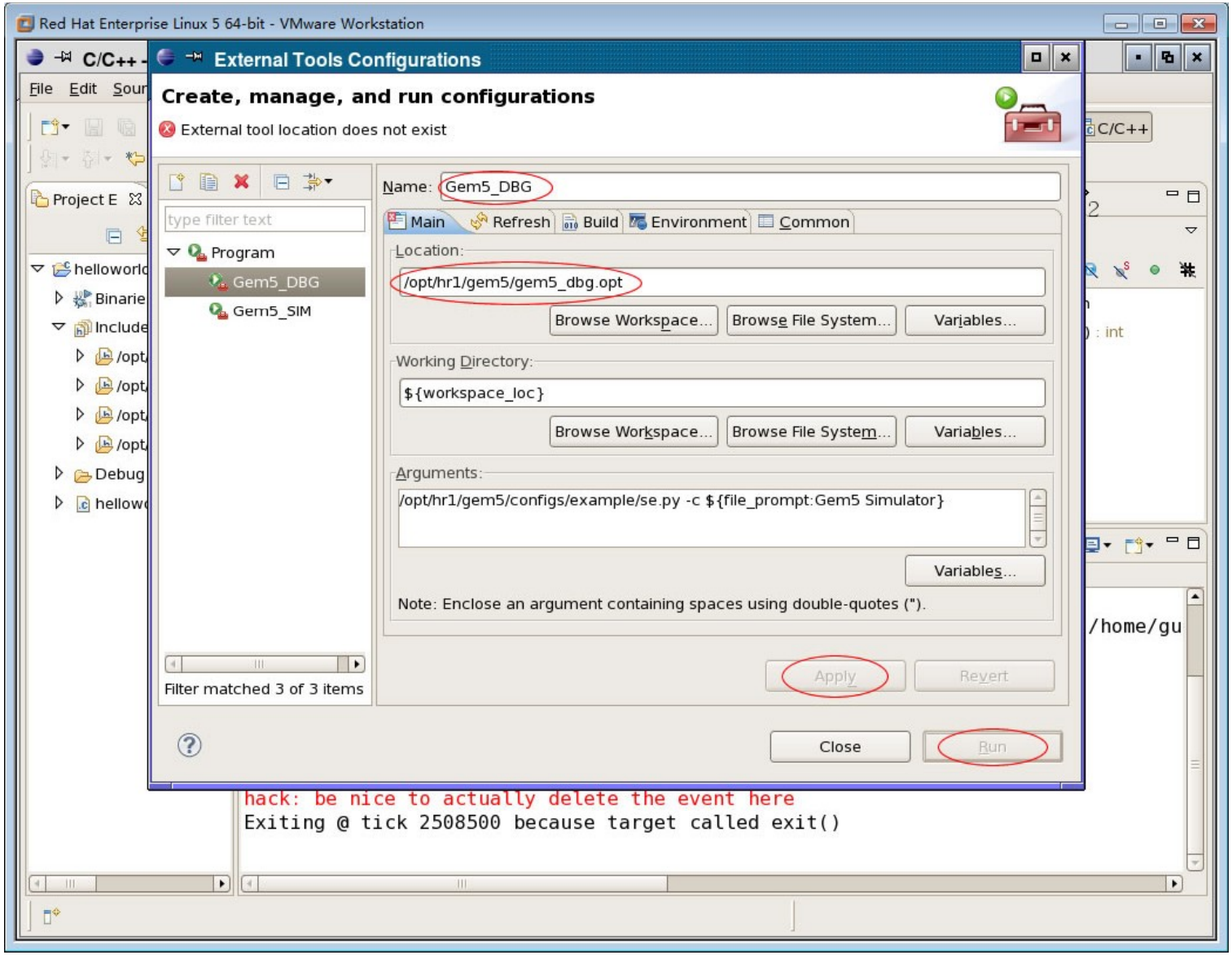
Eclipse Integration

14. Click the gem5 button to get the simulation results.



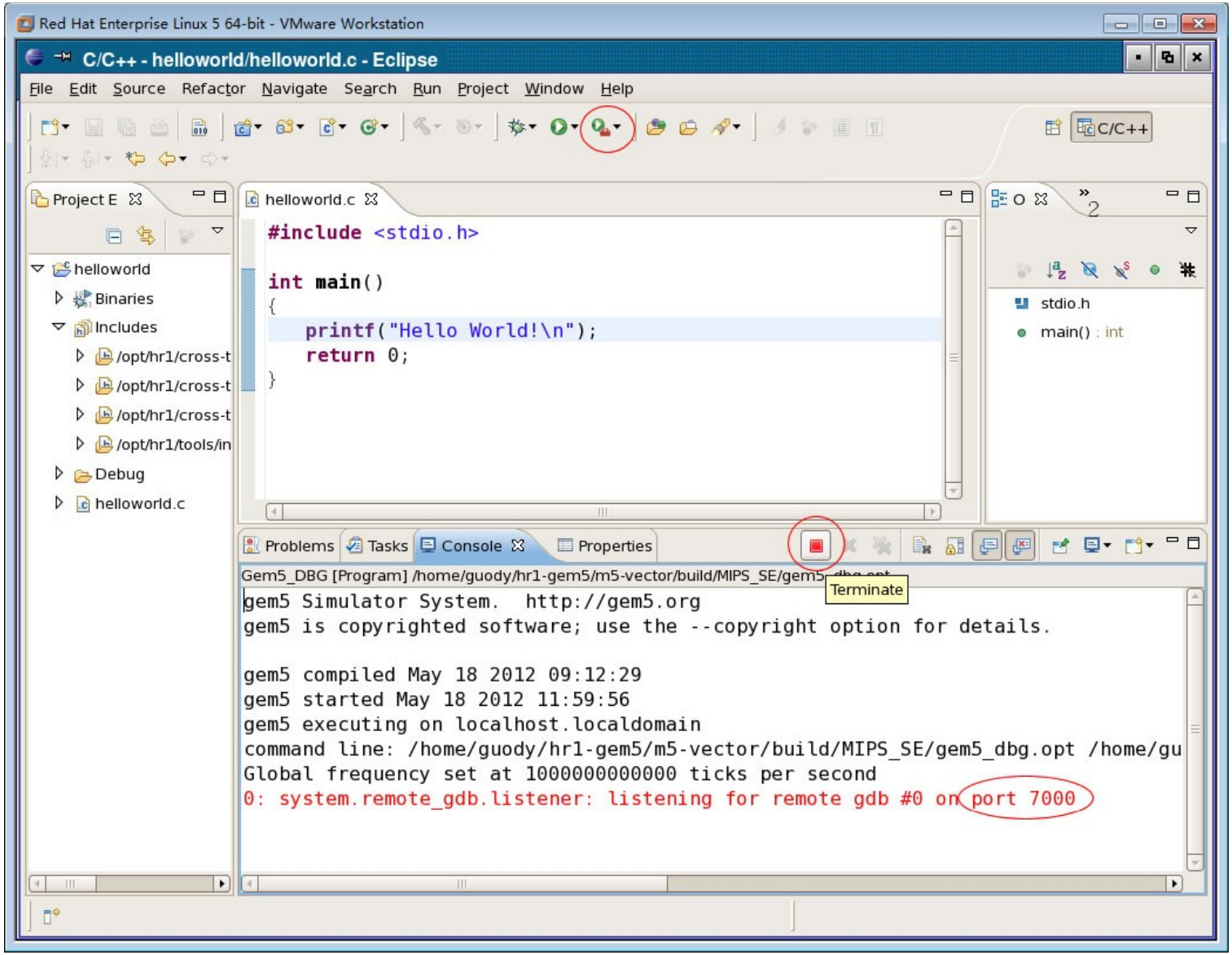
Eclipse Integration

15. Create another external tool for debugging.



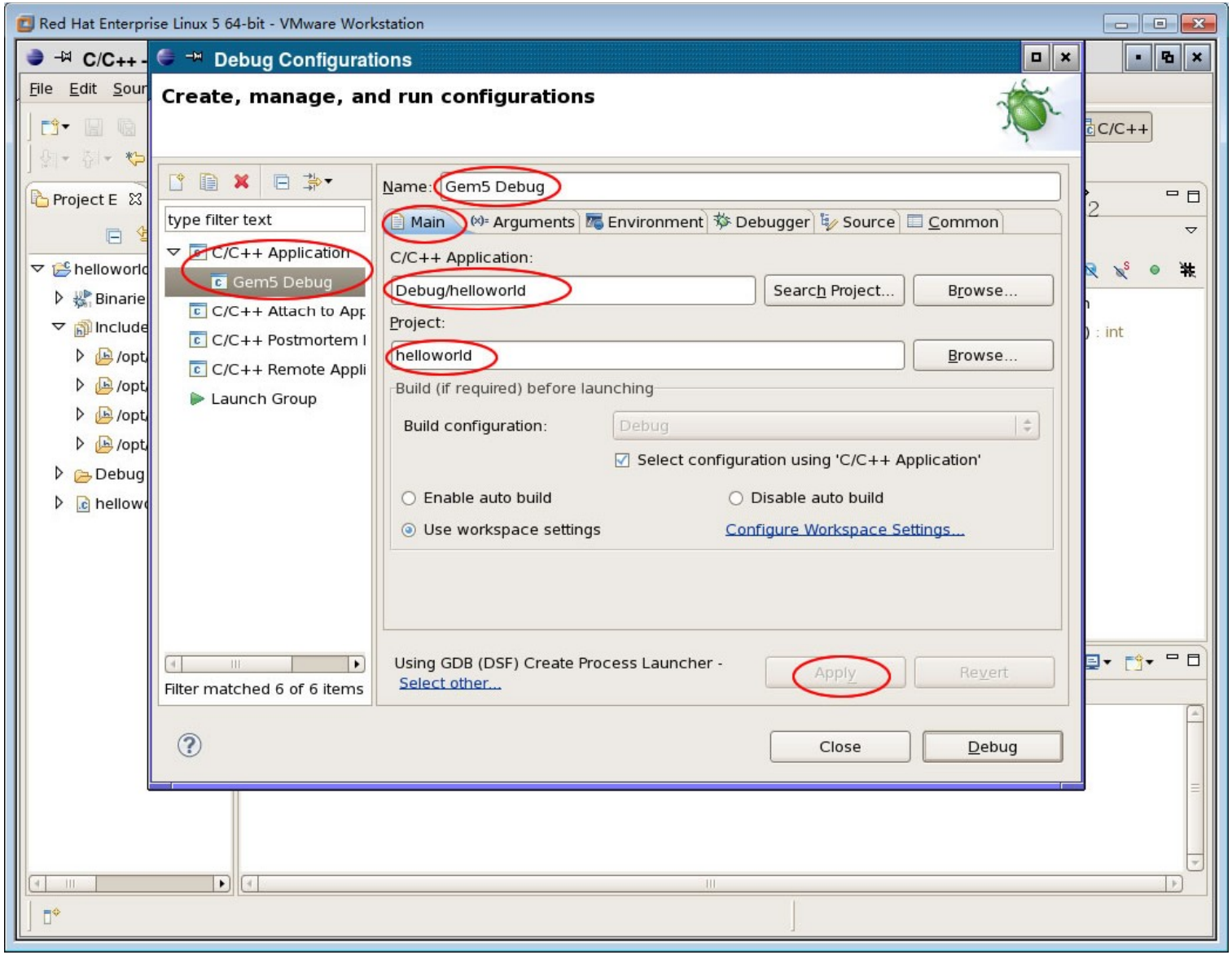
Eclipse Integration

16. This version of gem5 will always wait for a remote gdb connection.



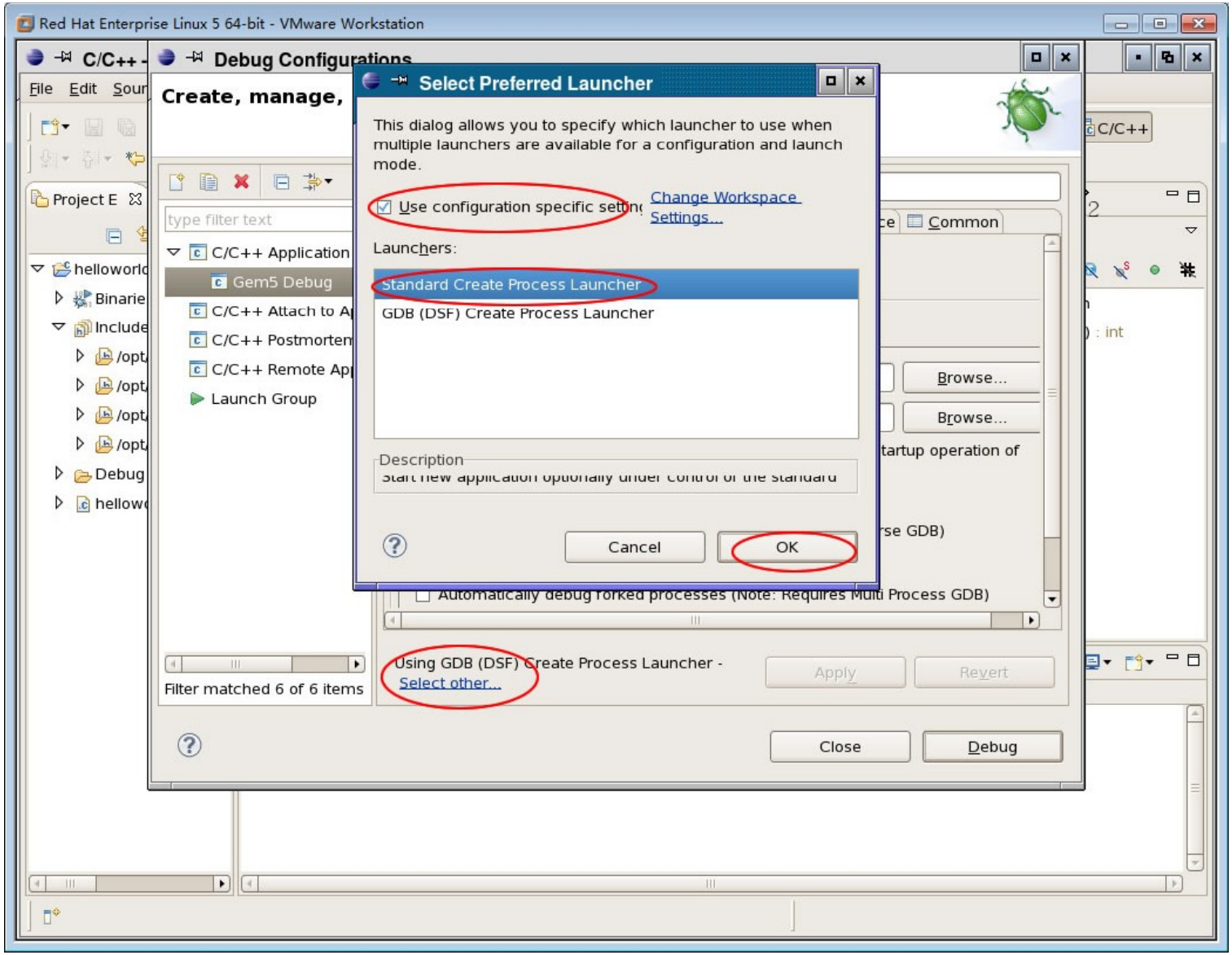
Eclipse Integration

17. Now choose the “Debug Configurations”, and set the arguments for gdb.



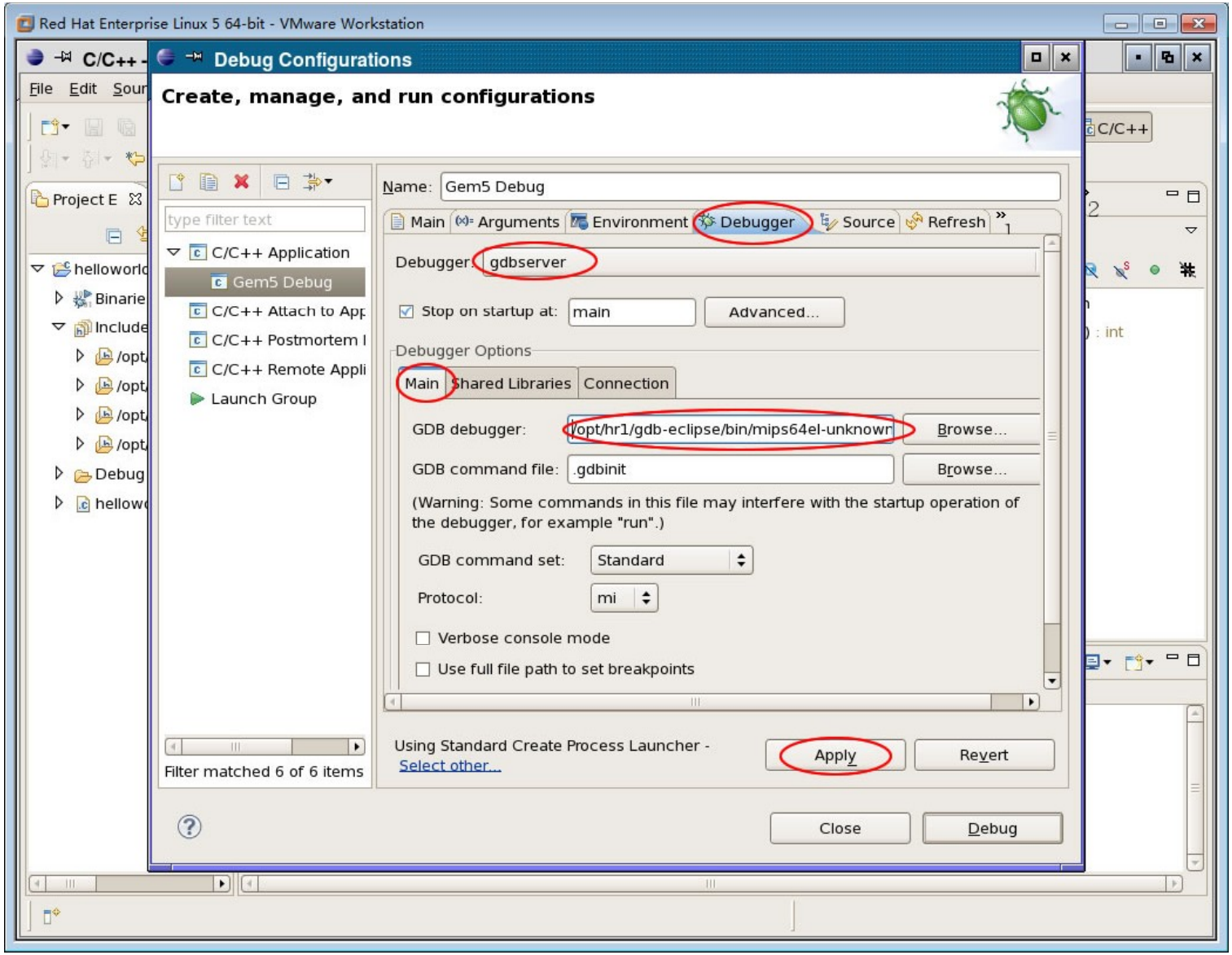
Eclipse Integration

18. Choose the “Standard Create Process Launcher” for gdb server.



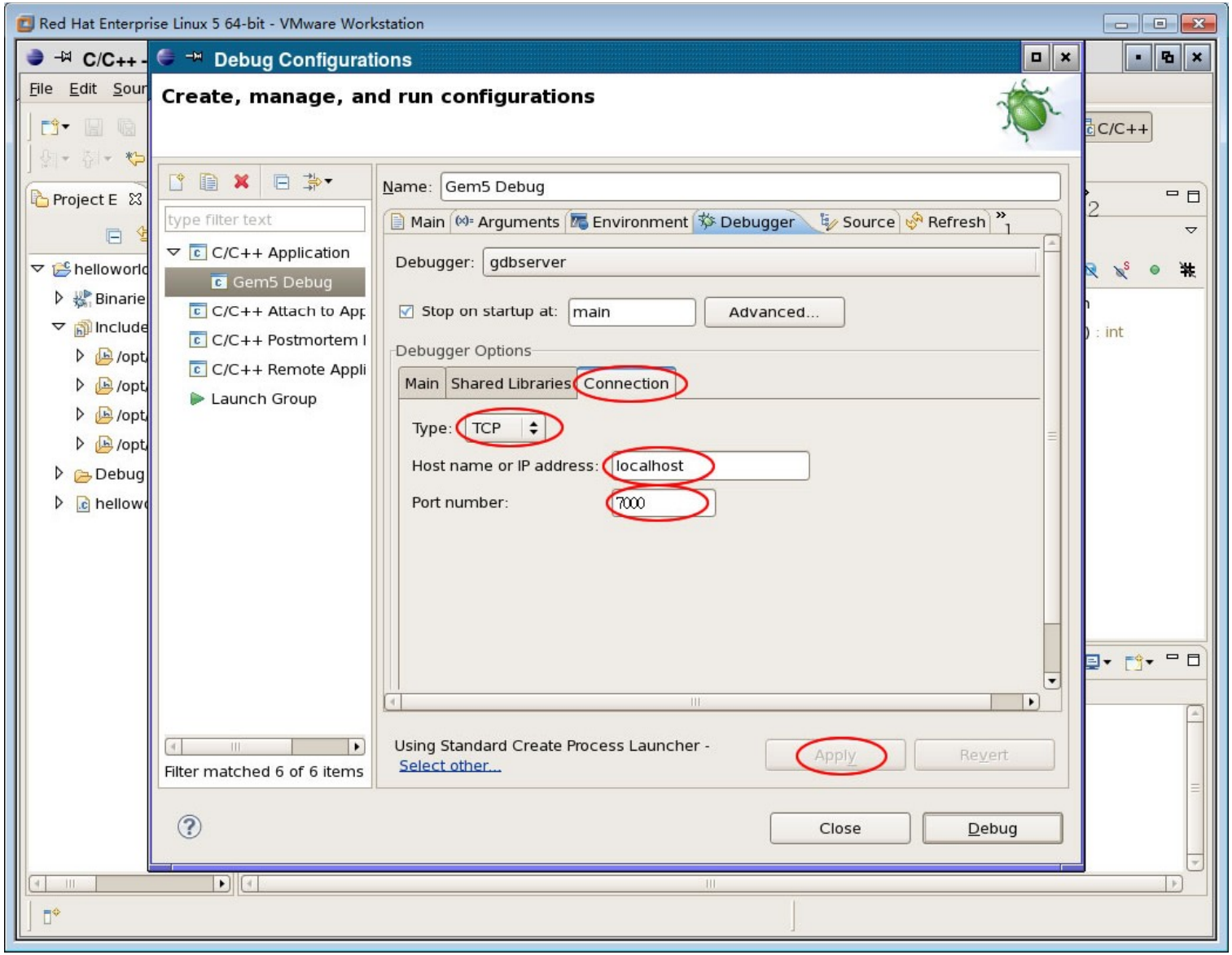
Eclipse Integration

19. Configure the path of cross gdb, and use gdbserver mode.



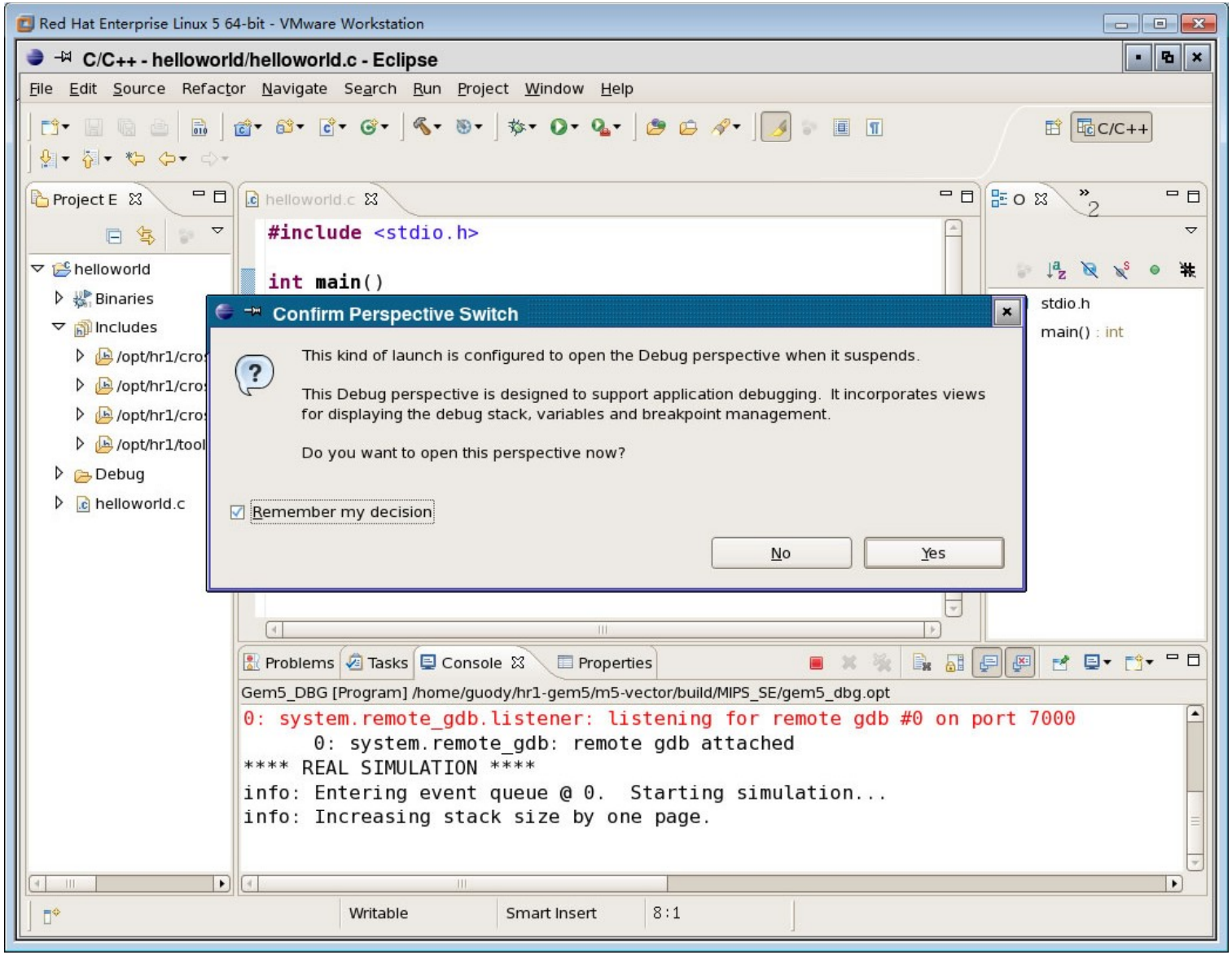
Eclipse Integration

20. Configure the IP address and port for gdb to connect to gem5 simulator.



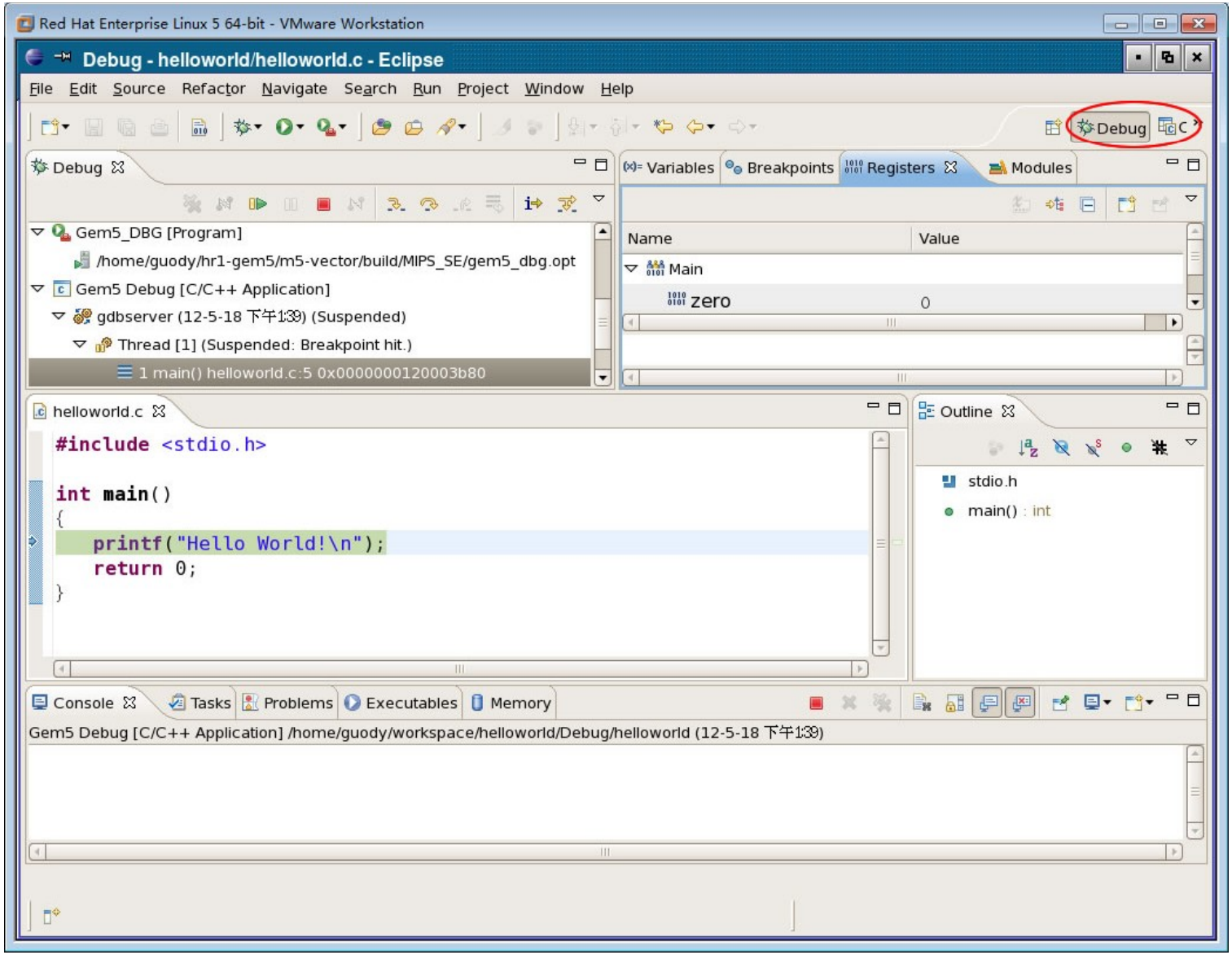
Eclipse Integration

21. After click “Debug”, Eclipse will change to Debug perspective.



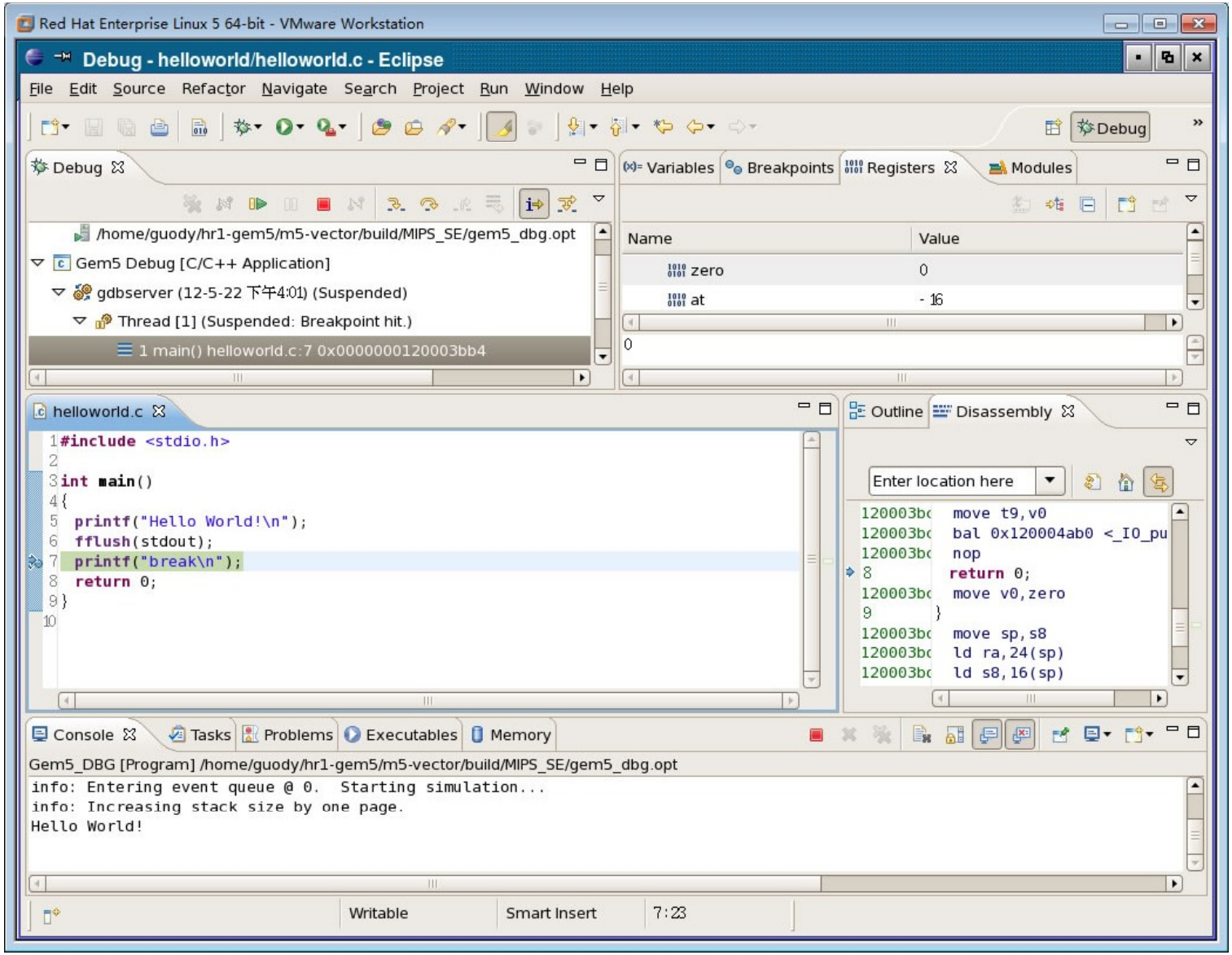
Eclipse Integration

22. The Debug perspective.



Eclipse Integration

23. Graphical debugging: registers, disassembly, breakpoints, outputs, etc.



Summary

- Part I. VLIW DSPs
  - 3 VLIW DSP architectures are introduced.
  - Some VLIW features.
  - 2 CPU model in gem5: VliwSimple and VliwPipeline.
  - Hardware implementation and benchmarks.
- Part II. MIPS Supporting
  - The MIPS64 extension.
  - MIPS32 FS mode supporting in gem5.
- Part III. Eclipse Integration
  - The configuration steps to integrate gem5 into Eclipse IDE.

Thank You!

Thank the following people in our laboratory for their contributions:

Software: Xu Yang, Zhengxing Li, Xiaotian Li, Daoling Zhang, Wu Bai, Minchao Chen, Dalin Zhu.

Hardware: Zheng Shen, Yuan Liu, Chen Chen, Junping Ma, Fengyang Chen, Yong Du.

Contact us:

Hu He: hehu@tsinghua.edu.cn

Deyuan Guo: guodeyuan@tsinghua.edu.cn

Home Page: http://dsp.ime.tsinghua.edu.cn