

UNIVERSITÀ DEGLI STUDI DI SASSARI

Progetto di fine corso:

INSTALLAZIONE E TEST DI DUE SIMULATORI ARCHITETTURALI: Gem5 e MarssX86



UNISS

UNIVERSITÀ
DEGLI STUDI
DI SASSARI

Studente:
Daniele Picciau
Monica Manai
Diego Perazzona

Matricola:
50056771
50057077
50057111

ANNO ACCADEMICO 2025-2026

1	Installazione e setup dei simulatori per Windows e macOS	2
1.1	Premessa fondamentale: architetture e compatibilità	2
1.2	Installazione e setup del simulatore Gem5	2
1.2.1	Installazione e setup di Gem5 su Windows	2
1.2.2	Installazione e setup di Gem5 su macOS	3
1.3	Installazione e setup del simulatore MarssX86	4
1.3.1	Docker	4
1.3.2	VirtualBox	5

1 Installazione e setup dei simulatori per Windows e macOS

1.1 Premessa fondamentale: architetture e compatibilità

Prima di procedere, è essenziale capire che questi simulatori non sono normali applicazioni installabili con un semplice clic. Entrambi sono nati in ambito accademico per Linux, ma hanno esigenze molto diverse legate alla loro "età".

Questi due simulatori architetturali (Gem5 e MarssX86) nascono e vengono sviluppati nativamente per sistemi Unix-like.

- **macOS** è un sistema certificato Unix. "Parla la stessa lingua" di Linux, quindi può eseguire molti di questi programmi nativamente (direttamente dal terminale), con il solo aiuto di alcune librerie esterne.
- **Windows** ha un'architettura completamente diversa, per cui non può eseguire nativamente questi programmi. Per poterli eseguire è quindi necessario utilizzare delle soluzioni di **virtualizzazione o sottosistemi** per creare l'ambiente Unix necessario.

Bisogna anche considerare che, anche avendo un sistema Linux (o Unix), c'è il problema delle librerie software necessarie per compilare il codice (il "Time Gap").

- **Gem5** è un software **moderno e adattivo**, che viene continuamente sviluppato e aggiornato. Funziona senza problemi sui sistemi operativi moderni: su macOS lo si compila direttamente mentre su Windows si usa WSL con un'installazione Linux recente (Ubuntu 20.04/22.04).
- **MarssX86** è un progetto "legacy" (molto vecchio, fermo ad 2012 circa). Richiede compilatori (GCC vecchi) e librerie che sono state rimosse dai sistemi operativi moderni da anni. Nè macOS moderno ne WSL possono eseguirlo facilmente, dunque è necessario creare una **Macchina virtuale** (con VirtualBox) che simuli un computer del 2014 con installato **Ubuntu 14.04**.

1.2 Installazione e setup del simulatore Gem5

Per quanto riguarda l'installazione del simulatore Gem5, la soluzione varia leggermente a seconda del sistema operativo utilizzato.

1.2.1 Installazione e setup di Gem5 su Windows

Per installare il simulatore su Windows la soluzione migliore è **WSL** (Windows Subsystem for Linux). Questo sottosistema permette di avere un terminale Ubuntu vero e proprio integrato in Windows. Dunque, l'installazione si compone di diversi passaggi:

- **Passo 1: attivazione di WSL**
 1. Aprire la **PowerShell** come amministratore;
 2. Digitare `wsl --install` e premere invio;
 3. Attendere il download e, una volta completato, riavviare il computer;
 4. Al riavvio, seguire le istruzioni a schermo per creare *username* e *password* Ubuntu.

- **Passo 2: installazione delle dipendenze**

Nel terminale di Ubuntu appena aperto incollare questi comandi uno alla volta:

```
//Aggiorna i pacchetti
sudo apt update && sudo apt upgrade -y
```

```
//Installa compilatori, Python e librerie necessarie a Gem5
sudo apt install build-essential git m4 scons zlib1g zlib1g-dev
    libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-
    dev python3-dev python-is-python3 libboost-all-dev pkg-config -y
```

- **Passo 3: scaricare e compilare Gem5**

```
//Clonare il repository del progetto nel proprio workspace
git clone https://github.com/gem5/gem5.git
```

```
//Entrare nella cartella "gem5"
cd workspace/gem5
```

```
//Avviare la compilazione (ci vorranno dai 10 ai 40 minuti)
//Questo comando usa tutti i core della tua CPU per fare prima
scons build/ARM/gem5.opt -j $(nproc)
```

Note: se si vuole simulare RISC-V, bisogna sostituire ARM con RISCV nel comando scons.

1.2.2 Installazione e setup di Gem5 su macOS

Su macOS, come detto al capitolo 1.1, è possibile compilare nativamente usando il terminale, ma è necessario un gestore di pacchetti chiamato **Homebrew**.

Anche in questo caso possiamo suddividere l'installazione in diversi passaggi:

- **Passo 1: preparazione dell'ambiente**

1. Aprire il terminale;
2. Se non si hanno, installare gli strumenti di sviluppo: `xcode-select --install`;
3. Installare Homebrew seguendo le istruzioni su <https://brew.sh/>.

- **Passo 2: installare le dipendenze**

```
brew install git scons python protobuf boost libpng hdf5 pkg-config
```

- **Passo 3: scaricare e compilare Gem5**

```
//Clonare il repository del progetto nel proprio workspace
git clone https://github.com/gem5/gem5.git
```

```
//Entrare nella cartella "gem5"
cd workspace/gem5
```

```
//Avviare la compilazione
//Questo comando usa tutti i core della tua CPU per fare prima
scons build/ARM/gem5.opt -j $(nproc)
```

Note: anche in questo caso, per simulare RISC-V, bisogna sostituire ARM con RISCV nel comando scons.

1.3 Installazione e setup del simulatore MarssX86

Come detto precedentemente al capitolo 1.1, MarssX86 è troppo vecchio per funzionare su WSL moderlo o su macOS. Esistono due strade che risolvono il problema, le quali funzionano sia per Windows che per macOS:

1.3.1 Docker

L'utilizzo di Docker è da prendere in considerazione nel caso in cui si voglia una maggiore rapidità, infatti, permette di scaricare un immagine di Ubuntu 14.04 e lavorarci dentro da terminale senza installare un intero sistema operativo.

- **Passo 1: Installare Docker**

1. Windows/macOS: scaricare e installare Docker desktop dal sito ufficiale <https://www.docker.com/>;
2. Avviarlo e assicurarsi che sia attivo (controllare l'icona della balena nella barra delle applicazioni).

- **Passo 2: Creare l'ambiente (il Dockerfile)**

Creare una cartella sul computer chiamata `marss_project` e al suo interno creare un file di testo chiamato `Dockerfile` in cui andrà incollato al suo interno il seguente contenuto:

```
// Si usa una versione vecchia di Ubuntu compatibile con MarssX86
FROM ubuntu:14.04

// Aggiornare i repository per trovare i vecchi pacchetti
RUN apt-get update && \
    apt-get install -y build-essential git scons zlib1g-dev python g++ \
    vim

// Impostare la cartella di lavoro
WORKDIR /root/marss

// Comando di default quando si entra nel container
CMD ["/bin/bash"]
```

- **Passo 3: costruire e avviare**

Aprire il terminale (o PowerShell) nella cartella dove è stato inserito il Dockerfile.

1. **Costruzione dell'immagine** tramite il comando:

```
docker build -t marss_env .
```

2. **Ingresso nel contenitore** tramite il comando:

```
docker run -it --name marss_container marss_env
```

Una volta dentro, il terminale "penserà" di essere nel 2014 garantendo la possibilità di scaricare e compilare MarssX86 lì dentro

1.3.2 VirtualBox

Invece, la VirtualBox è consigliata se si preferisce avere un'interfaccia grafica, perché permette di avere un "desktop" completo con mouse e finestre, usando una macchina virtuale classica.

- **Passo 1: scaricare e installare il software**

1. scaricare e installare **VirtualBox** al seguente url <https://www.virtualbox.org/>;
2. scaricare la ISO di Ubuntu 14.04 LTS (Trusty Tahr) <https://releases.ubuntu.com/14.04/>