A simulator, why?

- Modern hardware architectures:
  - may be too complex to understand
  - may be not useful for teaching and demonstration purposes
  - may require additional costs for effective development (software development kit, test boards, etc.)
  - may add unnecessary complexities to the development cycle

- A simulated hardware architecture:
  - may be tailored to provide exactly the "right" features for teaching and demonstration purposes
  - may be provided with an integrated development kit, graphical user interface and debug tools
  - may be deployed on available CS lab equipment
  - will probably be a lot slower than the real one (not always a bad feature)

MIPS, MPS and uMPS

- MIPS: Microprocessor (without) Interlocking Pipe Stages
  - one of the original RISC processor architectures from the ‘80s
  - with a lot of interesting features
  - still widely used (on embedded systems, but also …)

- MPS:
  - a complete (simulated) computer system integrating an (emulated) MIPS R3000 CPU

- uMPS:
  - a complete (simulated) computer system integrating an (emulated) MIPS R3000 CPU with physical and virtual memory addressing
A MIPS processor, why?

- MIPS R3000 processor with MIPS I instruction set:
  - is reasonably easy to understand
  - provides useful features and insights for instructional purposes
  - documentation is widely available
  - is supported by the GNU gcc compiler and development kit
  - does not impose a fixed devices interface
- More info (and manuals too):

MPS and uMPS

- uMPS:
  - (almost) "all of the above"
  - ethernet-like network interfaces
  - physical and virtual memory addressing
  - a streamlined user interface
- Why use uMPS and not MPS?
  - Because having virtual memory "right from the beginning" adds unnecessary complexities when writing an OS from scratch…

MPS and uMPS may be compiled on:

- FreeBSD, GNU/Linux distributions (x86 and PPC)
- Sun Solaris

uMPS processor architecture

- The uMPS architecture
MIPS delayed load:

MIPS R2/3000 Delayed Load

MIPS delayed branch:

MIPS R2/3000 Delayed Branch
uMPS processor architecture

- **uMPS processor features:**
  - RISC-type integer instruction set on a load-store architecture
  - 32-bit word for registers/instructions/addressing (4 GB physical address space)
  - Pipelined execution, delayed loads and branches
  - 32 general purpose registers (GPR) denoted $0 \ldots 31$
    - Register $0$ is hardwired to zero (0)
    - Registers $1 \ldots 31$ (also with mnemonic designation)

- **uMPS processor features (cont’d):**
  - all of $1 \ldots 31$ registers may be used, but some conventions exist, for example:
    - $26$ and $27$ ($k0$ and $k1$) are reserved to kernel use
    - HI and LO, special registers for holding the results from multiplication and division operations
    - PC, the program counter

- **CP0 (CoProcessor 0) is incorporated into the main CPU and provides:**
  - two processor operation modes:
    - kernel-mode
    - user-mode
  - exception handling
  - virtual memory addressing

- **CP0 has 8 registers:**
  - Status register
  - used for exception handling:
    - Cause
    - EPC
  - used for virtual memory addressing:
    - Index
    - Random
    - EntryHi
    - EntryLo
    - BadVAddr
uMPS processor architecture

- **Miscellaneous uMPS processor features:**
  - **Endianness:**
    - the uMPS processor may operate in big-endian and little-endian mode (the emulator uses the endianness of the host architecture)
    - a different cross-compiler set is required
  - **CP1:** optional coprocessor for floating point support
    - unimplemented
    - processor traps if floating point instructions are executed or CP1 access is attempted

Big endianness:

```
+----------------------------------+---+
| High Addresses                  | Word address |
| 8 9 10 11                       | 8            |
| 4 5 6 7                         | 4            |
| 0 1 2 3                         | 0            |
```

Little endianness:

```
+----------------------------------+---+
| High Addresses                  | Word address |
| 11 10 9 8                       | 8            |
| 7 5 4                           | 4            |
| 3 2 1 0                         | 0            |
```

uMPS physical memory address format:

```
+-----+-------+
| PFN | Offset|
| 31   | .......|
| 12   | 11     |
|      | .......|
| 0    | 0      |
```
uMPS processor architecture

- **uMPS virtual memory address format:**
  - Segment Number, Virtual Page Number and Offset
  - **ASID** (Address Space Identifier): 0..63 (0 for Kernel)

<table>
<thead>
<tr>
<th>SEGNO</th>
<th>VPN</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>29</td>
<td>1211</td>
</tr>
</tbody>
</table>

Status register structure:

- **IE**: Interrupt Enable
- **KU**: Kernel/User mode (kernel = 0)
- **IM**: Interrupt Mask
- **VM**: Virtual Memory
- **BEV**: Bootstrap Exception Vector
- **CU**: Coprocessor Usable

uMPS processor status at bootstrap:

- CP0 is enabled
- Virtual Memory is off
- Bootstrap Exception Vector bit is on
- Processor is in Kernel mode
- **PC = 0x1FC0.0000** (in boot ROM)
**Exception handling:**
- EPC (Exception PC): is automatically corrected by the CPU if BD bit is set, to allow re-execution of the branch
- Cause:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>CE</td>
</tr>
<tr>
<td>30</td>
<td>28</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

**Cause explained:**
- IP: Interrupt Pending
- BD: Branch Delay
- CE: Coprocessor Error
- ExcCode

**ExcCode:**

<table>
<thead>
<tr>
<th>Number</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Int</td>
<td>External Device Interrupt</td>
</tr>
<tr>
<td>1</td>
<td>LInt</td>
<td>TLB-Modification Exception</td>
</tr>
<tr>
<td>2</td>
<td>TLE</td>
<td>TLB Invalid Exception: on a load instr. or instruction fetch</td>
</tr>
<tr>
<td>3</td>
<td>TLE</td>
<td>TLB Invalid Exception: on a store instr.</td>
</tr>
<tr>
<td>4</td>
<td>AdErr</td>
<td>Address Error Exception: on a Load or instruction fetch</td>
</tr>
<tr>
<td>5</td>
<td>AdErr</td>
<td>Address Error Exception: on a Store instr.</td>
</tr>
<tr>
<td>6</td>
<td>BErr</td>
<td>Bus Error Exception: on an instruction fetch</td>
</tr>
<tr>
<td>7</td>
<td>FErr</td>
<td>Bus Error Exception: on a Load/Store data access</td>
</tr>
<tr>
<td>8</td>
<td>SErr</td>
<td>Syscall Exception</td>
</tr>
<tr>
<td>9</td>
<td>Bp</td>
<td>Breakpoint Exception</td>
</tr>
<tr>
<td>10</td>
<td>RI</td>
<td>Reserved Instruction Exception</td>
</tr>
<tr>
<td>11</td>
<td>CF</td>
<td>Coprocessor Unusable Exception</td>
</tr>
<tr>
<td>12</td>
<td>C0</td>
<td>Arithmetic Overflow/Exception</td>
</tr>
<tr>
<td>13</td>
<td>D0</td>
<td>Bad Page Table</td>
</tr>
<tr>
<td>14</td>
<td>PMM</td>
<td>Page Table Miss</td>
</tr>
</tbody>
</table>

**Exception handling (cont’d):**
- Exception types:
  - Program Traps (PgmTrap)
  - SYSCALL/Breakpoint (SYS/Bp)
  - TLB Management (TLB)
  - Interrupts (Ints)
Exception handling (cont'd):
- Program Traps (PgmTrap)
  - Address Error (AdEL & AdES)
  - Bus Error (IBE & DBE)
  - Reserved Instruction (RI)
  - Coprocessor Unusable (CpU)
  - Arithmetic Overflow (Ov)
- SYSCALL/Breakpoint (SYS/Bp)
  - SYSCALL instruction
  - BREAK instruction

TLB Management (TLB):
- TLB-Modification (Mod)
- TLB-Invalid (TLBL & TLBS)
- Bad-PgTbl (BdPT)
- PTE-MISS (PTMs)

Interrupts (Ints):
- remember Status.IM mask and Status.IEc bit
- hardware and software interrupts

uMPS processor actions on exception:
- Basic operations:
  - EPC stores the current PC
  - BD bit is set if required
  - Cause.ExcCode is set
  - Status.VM, KU and IE stacks are pushed:

Exception-specific operations:
- Address Error (AdEL & AdES): set BadVAddr
- Coprocessor Unusable (CpU): set Cause.CE
- Interrupts (Ints): set Cause.IP
- TLB Management (TLB):
  - set BadVAddr
  - load EntryHi.SEGNO and EntryHi.VPN
uMPS processor architecture

ROM exception handler first task:
- to save the current processor state (the “old” one) and to load a new state (the “new” one)
- A processor state contains:
  - 1 word for the EntryHi CP0 register (contains the current ASID, EntryHi.ASID)
  - 1 word for the Cause CP0 register
  - 1 word for the Status CP0 register
  - 1 word for the PC (New) or EPC (Old)
  - 29 words for the GPR registers (GPR registers $0$, $k0$, and $k1$ are excluded)

But where is the ROM?

uMPS physical memory map (Kernel and User modes)

How it is mapped?

ROM and device registers area:
uMPS processor architecture

- But where is the processor state stored?
  - in the ROM reserved frame:
    - [Diagram of ROM reserved frame]

- But where?
  - in the bottom part of the ROM reserved frame:
    - [Diagram of ROM reserved frame]

uMPS processor architecture

- Ending the exception handling:
  - ROM handler (hopefully) will load a processor state and:
    - jump to some address
    - \textit{RFE} (Return From Exception): pop the \textit{KU}, \textit{IE} and \textit{VM} stacks

uMPS processor architecture

- Beware...
  - look at \textit{Cause} in Old area for knowing exactly what happened
  - remember that \textit{KU}, \textit{IE} and \textit{VM} stacks in Status were pushed before being stored, and will be popped when returning from the exception
  - remember that \textit{EPC} will point to the correct address to jump to after having serviced the exception (the \textit{BD} bit tells if it was the instruction at \textit{EPC} or the instruction in a branch delay slot to cause the exception)