Recap: TRaX Thread

- Instruction Cache
- Thread
  - PC
  - RF
  - Stack
  - RAM
- FUs
  - Int Add
  - FP Mul
  - FP Inv
  - ...
- L1
- L2
- DRAM
TRaX Memory Models

Main memory

Instruction memory

Instruction Cache

Thread

PC

RF

Stack

RAM

DRAM

L2

L1

Program memory
TRaX Memories

- Instruction memory
  - Isolated from other memories
  - Branch addresses are explicitly in instruction memory

- Local stack
  - Compiler’s playground
  - No malloc libraries

- Global (main) memory
  - Unused so far
  - This limits our programs to operating on tiny data
# Programming Models

<table>
<thead>
<tr>
<th></th>
<th>Most computers</th>
<th>TRaX</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main memory</strong></td>
<td>Automatically handled by compiler</td>
<td>Explicitly handled by programmer</td>
</tr>
<tr>
<td><strong>Stack</strong></td>
<td>Abstraction of the OS/compiler</td>
<td>Automatically handled by compiler</td>
</tr>
<tr>
<td><strong>Instruction memory</strong></td>
<td>Invisible to programmer</td>
<td>Invisible to programmer</td>
</tr>
</tbody>
</table>
# Programming Models

<table>
<thead>
<tr>
<th></th>
<th>Most computers</th>
<th>TRaX</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main memory</strong></td>
<td>Automatically handled by compiler</td>
<td>Explicitly handled by programmer</td>
</tr>
<tr>
<td><strong>Stack</strong></td>
<td>Abstraction of the OS/compiler</td>
<td>Automatically handled by compiler</td>
</tr>
<tr>
<td><strong>Instruction memory</strong></td>
<td>Invisible to programmer</td>
<td>Invisible to programmer</td>
</tr>
</tbody>
</table>
Instruction Memory

- Loaded by simulator at runtime
  - Assembler.cc
- Word addressed
- Read only
- Not accessible by programmer
- Shared by multiple threads
- Single-cycle access
Local Memory (Stack)

- `.data`, `.text` loaded by simulator at runtime
  - Assembler.cc

- Byte addressed

- Read/Write

- Accessed indirectly by programmer (through compiler)

- All threads own individual unit
  - Not visible by any other thread

- Single-cycle access
Global (main) Memory

- Certain data pre-loaded by simulator
  - Can load anything you want
  - Usually assumes RT data needed (resolution, geometry, etc…)

- Word addressed

- Read/Write

- Accessed explicitly by programmer
  - loadf, storef, loadi, storei

- Shared by all threads

- Variable access time
Main Memory (red stuff)
Main Memory

- One giant address space

- Handled by 3 units:
  - L1Cache
  - L2Cache
  - USIMM (off-chip DRAM)
  - More on these later
Accessing Main Memory

- Main memory accepts just 2 instructions:
  - LOAD
  - STORE

- Not to be confused with:
  - LW, LWI, lbu, lbui, ...
  - SW, SWI, sb, sh, ...
Accessing Main Memory

- Word addressed

- Untyped
  - All “pointers” to main memory are just int

- Triangle \( t = *((\text{Triangle}*)\text{tri	extunderscore addr}) \) ❌
  - Compiler will generate stack loads, not main mem loads
  - Or: overload the * operator?

- Triangle \( t = \text{LoadTriangle} (\text{tri	extunderscore addr}) \) ✔
  - Helper method that LOADs necessary data
Compiler Instrinsics (trax.hpp)

- int loadi (int base, int offset)
  - Returns integer at address (base + offset)
- float loadf (int base, int offset)
  - Returns float at address (base + offset)
- void storei(int value, int base, int offset)
  - Stores value to address (base + offset)
- void storef(float value, int base, int offset)
  - Stores value to address (base + offset)

- “offset” arguments are optional, must be immediate
Most Computers:

Sphere* sph_ptr = ...;
Sphere s = *sph_ptr;

Compiler generates

LWI r11, r1, 252
LWI r8, r1, 260
LWI r6, r1, 256
...
LWI r9, r1, 292

TRaX:

int sph_addr = ...;
Sphere s = LoadSph(sph_addr);

You provide LoadSph source code

Center = Point(loadf(sph_addr, 0),
               (loadf(sph_addr, 1),
               ...

Compiler generates

LOAD r4, r5, 0
LOAD r7, r5, 1
LOAD r6, r5, 2 ...

Why Separate Memory Spaces?

- Most computers:
  - Any code you write may “dirty” the caches
  - Bigger caches to handle this?
  - Simpler programming model

- TRaX:
  - Precise control over which ops access caches/DRAM
  - Reserve expensive memory ops for scene data
  - Complicates programming model
  - Enables domain-specific optimizations
What’s in Main Memory?

- **Constants:**
  - Resolution, pointers (start_fb), etc...

- **Scene:**
  - Triangles, BVH/Grid, Materials, Framebuffer
  - Or anything you want (modify the memory loader)

- **Free:**
  - Use for any purpose
TRaX Constants (trax.hpp)

```
#define TRAX_XRES       1
#define TRAX_INV_XRES   2
#define TRAX_F_XRES     3
...
```

- Most of these are pointers (remember, pointer is just int)

- X resolution stored at address 1:
  - All equivalent:
    - int xres = loadi(TRAX_XRES);
    - int xres = loadi(1);
    - int xres = GetXRes()
Specifying Main Memory (config file)

MEMORY 100 536870912

- Latency only used if --disable-usimm
  - Naïve memory model (faster simulation)

- Capacity is in words (x4 = bytes)
  - Must be power of 2
  - loadi(TRAX_MEM_SIZE) == Capacity
Framebuffer

```c
int start_fb = loadi(7);
```

- `start_fb` is now a pointer to the framebuffer
  - Address 7 is a pointer to a pointer

- Framebuffer implied to live in address range:
  - `[start_fb .. (start_fb + GetXRes * GetYRes * 3)]`
Scene Data Pointers

- Light : loadi(TRAX_START_LIGHT)
- Camera : loadi(TRAX_START_CAMERA)
- Model
  - BVH/Grid : loadi(TRAX_START_SCENE)
  - Triangles : loadi(TRAX_START_TRIANGLES)
  - Vertex normals : ...
  - Texture coordinates : ...
- Materials : loadi(TRAX_START_MATLS)
Memory Loader

- Most of this data is specified by simtrax arguments
- Addresses will be determined by size of scene data

- `--view-file`
  - Camera data

- `--model`
  - Geometry (.obj or .iw format)
  - BVH info (built from geometry)
  - Material info (.obj files specify a .mtl file)

- `--light-file`
  - Light